Architectural Techniques for Improving NAND Flash Memory Reliability

Yixin Luo

CMU-CS-18-101
March 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee
Onur Mutlu, Chair
Phillip B. Gibbons
James C. Hoe
Yu Cai, SK Hynix
Erich F. Haratsch, Seagate Technology

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by Samsung, Qualcomm, AMD, the Oracle Software Engineering Innovation Foundation, Facebook, and the National Science Foundation under grant numbers CCF-0953246, CNS-1065112, CNS-1320531, and CNS-1409723.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Flash Memory, NAND Flash Memory, 3D NAND Flash Memory, Solid-state Drives (SSD), Nonvolatile Memory (NVM), Integrated Circuit Reliability, Error Characterization, Error Mitigation, Error Correction, Error Recovery, Data Recovery, Process Variation, Data Retention, Memory Systems, Memory Controllers, Data Storage Systems, Fault Tolerance, Computer Architecture

Abstract

Over the past decade, NAND flash memory has rapidly grown in popularity within modern computing systems, thanks to its short random access latency, high internal parallelism, low static power consumption, and small form factor. Today, NAND flash memory is widely used as the primary storage medium for smartphones, personal laptops, and data center servers. This growth in flash memory popularity has been sustained by the decreasing cost per bit of NAND flash memory devices over each technology generation, which is due to the increased storage density. The higher density, however, comes at a cost of reduced storage reliability. Unreliable primary data storage could lead to permanent loss of valuable data. Thus, we must improve NAND flash memory reliability to prevent data loss.

Existing techniques to keep NAND flash memory reliable are costly. For example, a strong error correcting code (ECC), such as low-density parity-check (LDPC) code, is typically used to tolerate up to a relatively high raw bit error rate (e.g., between 10^{-3} and 10^{-2}) from the flash memory. However, such ECC requires significant redundancy, high latency, and high area overhead in today's designs. To tolerate more errors in future generations of NAND flash memories, a stronger ECC is needed, which requires an even larger amount of data redundancy, latency and area overhead than the ECC used in today's flash memory. Such ECC is not only very costly, but it also may not be as effective as other novel techniques that are specialized for different error types. Our goal in this dissertation is to greatly improve flash memory reliability at low cost.

We identify three opportunities to improve the cost-efficiency of flash reliability enhancement techniques. First, we can adapt the flash controller to various NAND flash memory error characteristics, or even to the error characteristics of each individual flash chip. Second, we can adapt the flash controller to how the host uses the NAND flash memory, e.g., application access patterns and environmental temperature. Third, the flash chips are typically managed by a powerful controller within the Solid-State Drive (SSD). This powerful computing resource is underutilized when the SSD is idle or when the workload has low access intensity. We can use the flash controller to optimize flash reliability in the background without capacity or performance loss.

To exploit these opportunities in improving flash memory reliability, the main thesis of our approach is to specialize the flash controller algorithms to the device and workload characteristics, rather than using powerful but expensive generic error tolerance techniques such as ECC. In this dissertation, we (1) develop a new understanding of the NAND flash memory error characteristics and the workload behavior through rigorous experimental characterization, and (2) design smart flash controller

algorithms based on this new understanding to improve flash reliability at low cost. To this end, we make four major contributions.

First, we propose a new technique called *WARM* to improve flash memory lifetime. The key idea is to identify and exploit the write-hotness of the workload in the flash controller in order to improve flash reliability. We show that existing write-hotness agnostic techniques lead to redundant refresh operations that degrade flash lifetime. WARM manages write-hot data and write-cold data differently, and effectively improves flash lifetime with low hardware and performance overhead.

Second, we propose a new framework to learn an online flash channel model that predicts the underlying threshold voltage distribution of each flash chip. The threshold voltage distribution decides the error characteristics of the flash chip and changes over time due to flash memory wearout. We show that existing analytical threshold voltage distribution models are unsuitable for online flash channel modeling, as they are either inaccurate or expensive to compute. We show that Student's t-distribution and the power law distribution can be used to model the static and dynamic (changing) behavior of the threshold voltage distribution with high accuracy and low latency. We also show that a variety of existing techniques can be tuned using this online model to improve flash memory lifetime.

Third, we perform the first detailed, comprehensive characterization and analysis of 3D NAND flash memory errors. Through this analysis, we identify three new error characteristics in 3D NAND flash memory due to its unique structure and cell design. We develop models for two of the new error characteristics that are significant in current-generation 3D NAND flash chips. We develop four new mechanisms within the flash controller to mitigate the three new error types, and thus greatly reduce the error rate, at low cost.

Fourth, we perform the first experimental characterization of the self-recovery effect on 3D NAND flash memory and show that dwell time, i.e., the idle time between write cycles, and temperature significantly impact retention loss speed and program accuracy. We develop a new unified model of these effects, called the Unified self-Recovery and Temperature model (URT). Using this model, we propose a new technique called *HeatWatch* to mitigate errors due to early retention loss in 3D NAND flash memory. HeatWatch reduces the raw bit error rate by tuning the read reference voltages to the dwell time of the workload and the operating temperature of the flash memory. We show that HeatWatch efficiently tracks the temperature and dwell time of NAND flash memory and greatly mitigates retention errors in 3D NAND flash memory using this information.

Overall, this dissertation (1) deepens the understanding of the error characteristics of *both* planar and 3D NAND flash memory through rigorous experimental characterization and, (2) develops new flash controller algorithms that improve NAND flash memory reliability (both lifetime and error rate) at low cost by taking advantage of the flash device and workload characteristics that we find based on our new understandings.

Acknowledgments

First of all, I would like to thank my adviser, Onur Mutlu, for always trusting me to do good work, encouraging me whenever a paper gets rejected, giving me enough resources and opportunities to do great research, and pushing me to keep improving my presentation and writing skills.

I am grateful to Saugata Ghose, Yu Cai, and Erich Haratsch for being both my mentors and collaborators. I am grateful to the members of my PhD committee: Phillip Gibbons and James Hoe for their valuable feedback and for making the final steps towards my PhD very smooth. I am grateful to Deb Cavlovich who allowed me to focus on my research by magically solving all other problems.

I am grateful to SAFARI group members that were more than just lab mates. Saugata Ghose was not only a great mentor and collaborator to me who helped me improve in all aspects, but also a great friend who was always available for help. Hongyi Xin was like a big brother to me who was always supportive and gave me kind advice that kept me on track during my early struggles, and was never disappointing to have a fun discussion with about bioinformatics or history. Rachata Ausavarungnirun was a fantastic cook and was always kind to share his delicious food that reminded me of my grandmother's cooking. Donghyuk Lee encouraged me when it was most needed, and taught me his highest work ethic and kindness by example. Justin Meza helped me improve my writing and presentation skills during my early years in a very friendly manner. Chris Fallin was kind and patient enough to share his wisdom and time for insightful research discussions, which helped to improve my research skills. Kevin Chang was a perfect role model for me to follow, who also happened to share similar hobbies with me. Yoongu Kim pushed me to work harder and keep improving my research, and set a high standard for the group in terms of quality of research, writing, and presentation. Vivek Seshadri, Gennady Pekhimenko, Samira Khan were always eager to share valuable research and career advice. Kevin Hsieh was a great roommate at the PDL Retreat, and was always friendly to chat with. Amirali Boroumand was a nice buddy until he coldbloodedly left us for nicer weather in California. I also thank other members of the SAFARI group for their assistance and support: Yang Li, Nandita Vijaykumar, Jamie Liu, HanBin Yoon, Ben Jaiyen, Damla Senol, Arash Tavakkol, Minesh Patel, and Jeremie Kim.

During my time at Carnegie Mellon, I met a lot of wonderful people: Haoxian Chen, Yan Gu, Yihan Sun, Yu Zhao, Yong He, Yanzhe Yang, Junchen Jiang, Xuezhi Wang, Abutalib Aghayev, Jin Kyu Kim, Joy Arulraj, Lin Ma, Michael Zhang, Huanchen Zhang, Jinliang Wei, Dominic Chen, Guangshuo Liu, Junyan Zhu, Tianshi Li, Jiyuan Zhang, and many others who helped and supported me in many different ways. I am also grateful to people in the PDL and CALCM groups, for

accepting me into their communities and for all the feedback and comments that helped improve my work significantly.

I am grateful to my internship mentors for giving me the opportunities and resources to do research that is not only practical enough to benefit the company but also forward-looking enough to lead to this thesis. At Microsoft Research, I had the privilege to closely work with Jie Liu, Sriram Govindan, Bikash Sharma, Mark Santaniello, and Aman Kansal. At Seagate, I had the privilege to closely work with Erich Haratsch, Ludovic Danjean, Thuy Nguyen, Hakim Alhussien, Sundararajan Sankaranarayanan, Lei Chen, Hongmei Xie, and many others.

I would like to acknowledge the enormous love and support that I received from my friends and family. Firstly, I would like to thank my girlfriend, Fan Yang, for all her understanding and support that made me brave and focused enough to finish my PhD. I would like to thank my friends all over the world: Kaiyu Shen, Zixiao Chen, Siyuan Sun, Haishan Zhu, Yifei Huang, Yuqin Mu, Pengyao Chen, Xuanxuan and her lovely family, and many others who really helped me through my hardest days and brought me a lot of joy whenever I visited. Lastly, I would like to thank my parents, who raised me to be the person I am today. I thank my mother, Yi, for her encouragement, support, love, and sacrifice. I thank my father, Wenping, for setting a high standard for success.

Contents

List of Figures

List of Tables

Chapter 1

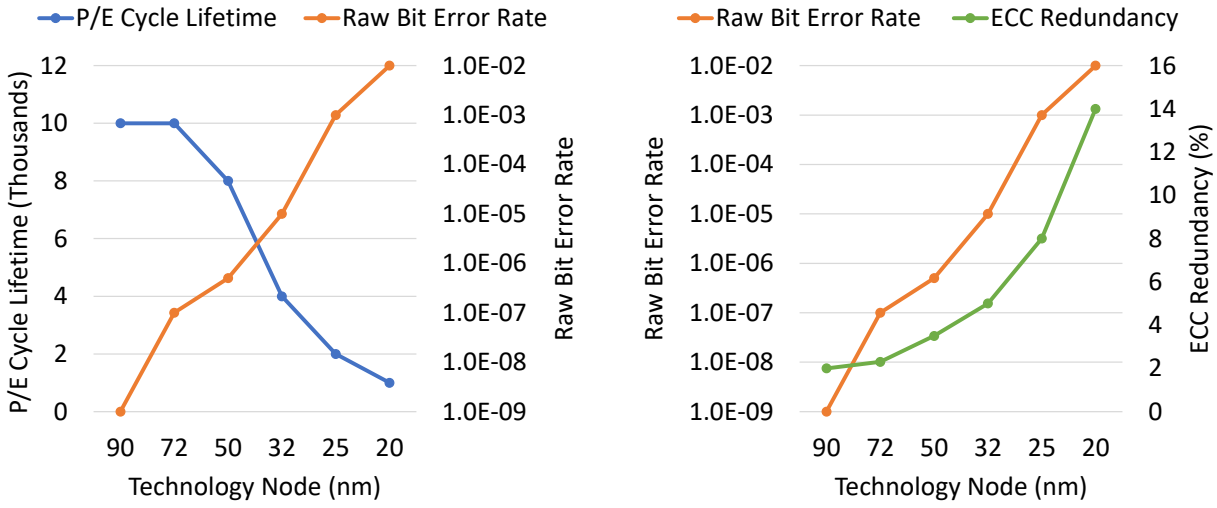
Introduction

1.1 The Problem: The Cost of Flash Reliability

In many modern servers and mobile devices, NAND flash memory (i.e., "flash" or "NAND flash") is used as the primary persistent storage device due to its lower access latency compared to a magnetic disk drive. As we generate more data at an increasingly faster speed, the need for higher density NAND flash memory also increases. In the past decade, flash density has increased by more than 1000× through aggressive process technology scaling from 90 nm to 15 nm. This rapid increase in flash density, however, has come at the cost of severely degraded flash memory reliability and lifetime, as is shown in Figure 1.1a. For example, the number of times that a cell can be reliably programmed and erased before wearing out (i.e., *P/E cycle lifetime*, shown as the blue curve in Figure 1.1a) has dropped from 10,000 times for 72 nm NAND flash (for a single-level cell, i.e., SLC device) to only 1,000 times for 20 nm NAND flash (for a triple-level cell, i.e., TLC device). In the meantime, data reliability, measured by the number of bit errors in a unit of data stored within the flash memory (i.e., *raw bit error rate*, shown as the orange curve in Figure 1.1a), increased by seven orders of magnitude from technology node 90 nm to 20 nm. Since the data stored on primary storage is often the only copy, unreliable data storage could lead to permanent loss of valuable data. Thus, we must improve NAND flash memory reliability to prevent data loss.

1.1.1 Flash Reliability Problems

First, we briefly introduce the basics of NAND flash memory to better understand the flash reliability problems. More detailed background on modern flash-memory-based solid-state drives (SSDs) and NAND flash memory can be found in Chapter 2. In NAND flash memory, each *flash cell* consists of a transistor that can store charge. A flash cell represents a certain data value based on the *threshold voltage* (V_{th}) of its transistor. In *multi-level cell* (MLC) flash memory, each cell stores two bits of data. A threshold voltage window (i.e., a *state*) is assigned for each possible two-bit value. Figure 1.2 shows the four possible states (i.e., ER, P1, P2, P3) in MLC flash memory, along with their corresponding two-bit values (i.e., the most significant bit, MSB, or, the least significant bit, LSB). The state of each flash cell can be *read* by applying one of three *read*



(a) Decreasing flash reliability

(b) Increasing ECC Redundancy

Figure 1.1: (a) Flash reliability (i.e., P/E cycle lifetime and raw bit error rate) and (b) ECC redundancy for each NAND flash memory technology node (for single-level cell, i.e., SLC devices for 90 nm to 72 nm technology nodes; for multi-level cell, i.e., MLC devices for 50 nm technology node; for triple-level cell, i.e., TLC devices for 32 nm to 20 nm technology nodes). Reproduced from [244].

reference voltages (i.e., V_a, V_b, and V_c) to the cell. Before each flash cell can be *programmed* to a new state, i.e., be written, the flash cell needs to be *erased* to the ER state. Due to the combination of *program variation*, i.e., the variation in program operations, and *manufacturing process variation*, the threshold voltage of cells programmed to the same state follow a Gaussian-like distribution across the voltage window of the state [23, 195, 260], depicted as the curve for each state in Figure 1.2.

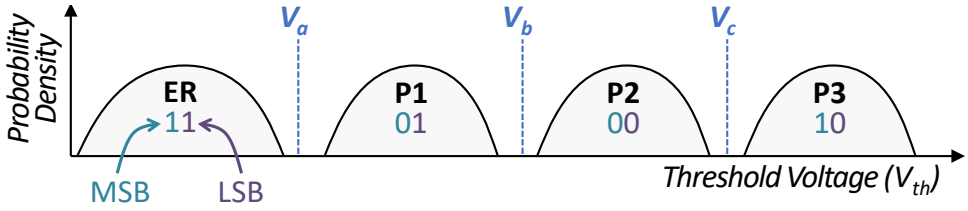


Figure 1.2: Threshold voltage distribution for MLC NAND flash memory.

The reliability problems in NAND flash memory are caused by various types of noise that arise from writing, reading, or idling of the NAND flash memory, or from process variation [21, 22, 23, 24, 26, 27, 33, 35, 219, 252]. Such noise shifts the threshold voltage distribution across the read reference voltages, as we show by comparing the relative position of the original and the shifted distributions in Figure 1.3. As a consequence of this shift, some of the cells are misread as being in a different state than the state they were programmed to. This phenomenon leads to a number of *raw bit errors*. More detailed background on the various types of NAND flash memory

errors can be found in Section 3.1. These errors include P/E cycling errors [23, 195, 260], cell-to-cell program interference errors [24, 26], program errors [34, 195, 260], read disturb errors [35, 260], retention errors [22, 27], and process variation errors [21, 269]. For instance, a flash cell wears out each time we write data to it via program or erase operations. Thus, the *P/E cycling error* rate increases over multiple program/erase operation cycles, or *P/E cycles*. As another example, the electrical charge stored in a flash cell leaks over time. Thus, the *retention error* rate increases during the idle time after the data is programmed to the cell, or *retention time*. To limit the raw bit error rate to a tolerable level, flash vendors guarantee reliable operation only for a limited number of *P/E cycle lifetime* and a limited amount of *retention time* [124].

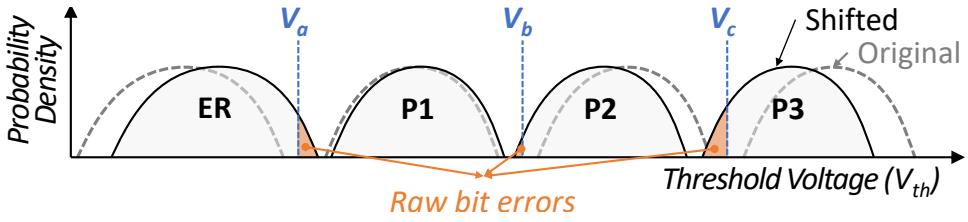


Figure 1.3: Shifted threshold voltage distribution for MLC NAND flash memory. Reproduced from [199].

1.1.2 The Cost of Improving Flash Reliability

To improve the reliable operation of NAND flash memory in the presence of raw bit errors, the common modern approach is to trade-off storage density or performance for higher reliability. We call this the "naive approach". For example, a strong error correcting code (ECC) or a redundant array of independent disk (RAID) techniques use *data redundancy* to detect and correct raw bit errors, trading off storage density for better reliability. As Figure 1.1b shows, in order to tolerate the significant increase in raw bit error rate that happens when going from a 90 nm technology node to a 20 nm technology node, the amount of redundant storage reserved for ECC (i.e., *ECC redundancy*, shown as the green curve in Figure 1.1b) has to increase exponentially from $\sim$2% to $\sim$14%. Today's NAND flash memory-based solid-state drives (SSDs) use low-density parity-check (LDPC) codes. The ECC redundancy required by LDPC codes for a certain *error correction capability*, i.e., the ability to correct a certain number of errors, is already approaching the theoretical limit, known as the *Shannon limit* [203, 204, 294]. Further improving the error correction capability of the ECC requires either higher ECC redundancy or larger coding granularity. The former greatly degrades storage density. The latter greatly degrades performance and also increases the hardware overhead of the ECC decoder/encoder [22, 25]. Another naive way to improve NAND flash memory reliability is to increase the precision of program and erase operations to mitigate write-induced errors. However, this method significantly increases write latency, degrading SSD performance.

Compared to the naive approach of providing better ECC at high cost and high performance overhead, we believe it is more appealing to develop new techniques to achieve better flash reliability *without* trading off storage density or performance. These new techniques would enable flash vendors to scale NAND flash memory density more aggressively and would make NAND

flash technology appealing for even more use cases and applications in future computing systems. These new techniques would also enable us to increase the storage density or performance by tuning down flash reliability when it is not required. For example, in geo-replicated data centers, data reliability is already handled by application-level error tolerance techniques. Thus, in these data centers, single-device reliability can be lowered to improve storage density by using quadruple-level cell (QLC) technology, which is very unreliable. If we could accomplish this without degrading data loss guarantees, we could buy back the data redundancy caused by data center geo-replication, reducing the total cost of ownership (TCO) of the data center. Our goal in this dissertation is to build a fundamental understanding of flash devices and application behavior to enable such techniques.

1.2 Related Work

Several prior works characterize NAND flash memory errors and develop mechanisms to improve flash reliability. In this section, we discuss some closely related prior approaches. We group prior works based on their high-level approach and discuss their shortcomings.

1.2.1 NAND Flash Memory Error Characterization

Prior work has characterized all types of raw bit errors in planar (or 2D) NAND flash memory, including P/E cycling errors [23, 33, 195, 219, 260], program errors [34, 195, 260], cell-to-cell program interference errors [24, 26, 33], retention errors [22, 27, 33, 219], read disturb errors [33, 35, 219, 252], and process variation errors [21, 269]. The findings from these characterizations can be found in Section 3.1.

While these works altogether have comprehensively studied the error characteristics of planar NAND flash memory, the raw data from these past characterization works are unavailable to the public for further analysis. These past characterizations were also done on the devices that are old by today's standards. Our new and more detailed characterization of much newer planar NAND flash memory in Chapter 5 allow us to develop new, and more accurate, online models and mitigation techniques for raw bit errors in more modern flash memory chips. Our characterization in Chapter 6 covers all these errors in the new 3D NAND flash memory (or *3D NAND*) devices and compares the error characteristics of 3D NAND with the results in prior work to uncover the differences between 3D NAND and planar NAND flash memories.

1.2.2 Improving Flash Reliability with Device Awareness

Flash memory device characteristics (i.e., NAND flash memory error characteristics) significantly affect flash reliability [21, 23, 27, 32, 33, 35, 219, 252]. Many prior works propose mechanisms to take advantage of these device characteristics to improve flash reliability. For example, prior work proposes mechanisms that periodically learn and adapt the read reference voltage to the threshold voltage distribution shift to mitigate P/E cycling errors, retention errors, and read disturb errors [27, 35, 252]. To mitigate cell-to-cell program interference errors, prior work proposes a technique that reads the values stored in neighboring cells to assist correcting

errors when a normal read operation fails [24, 26]. To reduce program errors in modern MLC NAND flash memory, recent work proposes to buffer the data stored in the LSB page within the flash chip [34]. To mitigate retention errors, prior work proposes to use various flash refresh techniques that periodically rewrite all data with high retention age [22, 25]. To mitigate read disturb errors, prior work proposes to opportunistically reduce the pass-through voltage, i.e., the highest possible read reference voltage, when the retention age is low [35].

While these works demonstrate that improving flash reliability with device awareness has many benefits in planar NAND, none of these works exploit an online model of flash device characteristics or design mechanisms customized for 3D NAND. In Chapter 5, we propose new techniques that further increase flash device characteristics awareness to improve flash reliability by learning an online model of the flash cells within the flash controller and adapting flash controller policies to this online model. The mechanisms we propose in Chapters 6 and 7 are designed for the unique error characteristics that we find in 3D NAND.

1.2.3 Improving Flash Reliability with Workload Awareness

Flash reliability varies significantly depending on the workload data access pattern [22, 24, 26, 32, 33, 35]. Thus, to improve flash reliability and lifetime, prior work proposes better flash management techniques to ensure a friendly data access pattern by optimizing flash controller algorithms. For example, to reduce unnecessary erase operations, prior work optimizes the flash page allocation policy to achieve higher spatial locality of write operations [94, 201, 202, 254]. Prior work also proposes techniques to minimize P/E cycles consumed by metadata within the flash controller [71, 255]. To mitigate program interference errors, prior work proposes to use certain program sequence, instead of allowing random writes, also managed by the controller [24, 26, 255]. To mitigate read disturb errors, prior work proposes to redistribute read-hot pages across different flash blocks [185].

While these works demonstrate several effective examples of improving flash reliability with workload awareness, none of these works design mechanisms to mitigate retention errors. The technique we propose in Chapter 4 partitions data with different write-hotness and applies the most suitable flash management policy for each partition to mitigate retention errors. The techniques we propose in Chapter 7 exploits workload write-intensity awareness and environment temperature awareness to mitigate retention errors in 3D NAND.

1.2.4 Summary

Demonstrated by these prior works, exploiting both device characteristics and application behavior in the flash controller often leads to significant reliability improvements with low overhead. This dissertation further improves, expands, or complements these techniques by exploiting *newly-discovered* (1) modern device characteristics and (2) application behavior characteristics, to greatly improve both bit error rate and lifetime of the state-of-the-art flash memory devices.

1.3 Thesis Statement and Overview

Our goal in this dissertation is to improve flash reliability at low cost and with low performance overhead. As we can see, the naive approach of providing better ECC trades off storage density or performance for improving flash reliability, and thus does *not* meet our goal of low cost and low performance overhead. To this end, our thesis statement is that,

> *NAND flash memory reliability can be improved at low cost and with low performance overhead by deploying various architectural techniques that are aware of higher-level application behavior and underlying flash device characteristics.*

Our approach is to understand flash memory device error characteristics and workload behavior through rigorous experimental characterization, and to design intelligent and efficient flash controller algorithms that utilize this understanding to improve flash reliability. This approach is based on *three* observations. First, we can take advantage of higher-level application behavior such as write frequency and locality, and develop the most suitable and customized flash reliability techniques for different types of data stored in the NAND flash memory. Second, we can take advantage of underlying device characteristics, such as variations in temperature, retention time, or error rate, to develop more efficient device-aware reliability techniques. Third, we can take advantage of the unused computing resources in the flash controller during idle time to enable more effective flash reliability techniques. In this dissertation, we investigate four directions to exploit the above three observations and efficiently improve flash reliability.

1.3.1 WARM—Write-hotness Aware Retention Management

Due to charge leakage from the flash cells, data retention errors increase over time after the data has been programmed onto NAND flash memory, i.e., the *retention time*. To mitigate retention errors, we limit the amount of retention time in NAND flash memory by providing a limited amount of guaranteed *internal retention time*, the duration for which the flash memory correctly holds data within its P/E cycle lifetime. Flash lifetime can be extended by relaxing this *internal retention time*. However, such relaxation cannot be exposed externally to the workload to avoid altering the expected *data integrity* property of a flash memory device, or the *non-volatility* property expected from a storage device. Reliability mechanisms, most prominently *refresh*, restore the duration of data integrity, but greatly reduce the lifetime improvement from retention time relaxation by performing a large number of write operations. We find that retention time relaxation can be achieved more efficiently by exploiting heterogeneity in *write-hotness*, i.e., the frequency at which each page is written.

We propose WARM, a write-hotness aware retention management policy for flash memory. *This is an example of our approach that exploits application-level* write-hotness *and device level* retention *characteristics to improve flash lifetime.* The key idea of WARM is to identify and to physically group together write-hot data within the flash device, allowing the flash controller to selectively perform retention time relaxation with little cost. When applied alone, WARM improves overall flash lifetime by an average of 3.24× over a baseline that does not relax the internal retention time, across a variety of real I/O workload traces. When WARM is applied together with an adaptive refresh mechanism, the average lifetime improves by 12.9× over the

baseline. More details are in Chapter 4 and our MSST 2015 paper [194].

1.3.2 Online Flash Channel Modeling and Its Applications

NAND flash memory can be treated as a noisy channel. Each flash cell stores data as the *threshold voltage* of a floating gate transistor. The threshold voltage can shift as a result of various types of circuit-level noise, introducing errors when data is read from the channel and ultimately reducing flash lifetime. An accurate model of the threshold voltage distribution across flash cells can enable mechanisms within the flash controller that improve channel reliability and device lifetime. Unfortunately, existing threshold voltage distribution models are either *not* accurate enough or have *high* computational complexity, which makes them unsuitable for *online* implementation within the controller.

We propose a new, low-complexity flash memory channel model, built upon a modified version of the Student's t-distribution and the power law, which captures the threshold voltage distribution and predicts future distribution shifts as wear increases. *This is an example of our approach that exploits the unused computing resources in the flash controller and enables greater device-awareness.* Using our experimental characterization of the state-of-the-art 1X nm (i.e., 15–19 nm) multi-level cell planar NAND flash chips, we show that our model is highly accurate (with an average modeling error of 0.68%), and also simple to compute within the flash controller (requiring 4.41 times less computation time than the most accurate prior model, with negligible decrease in accuracy). Our model also predicts future threshold voltage distribution shifts with a 2.72% average modeling error.

We demonstrate several example applications of our new model in the flash controller, which improve flash channel reliability significantly, including a new mechanism to predict the remaining lifetime of a flash device. Our evaluations for two of these applications show that our model: (1) helps improve flash memory lifetime by 48.9% and/or (2) enables the flash device to safely sustain 69.9% more write operations than manufacturer specifications. More details are in Chapter 5 and our IEEE JSAC 2016 paper [195].

1.3.3 3D NAND Flash Memory Error Characterization and Mitigation

Compared to planar NAND flash memory, 3D NAND flash memory uses a new flash cell design, and vertically stacks dozens of silicon layers in a single chip. This allows 3D NAND flash memory to increase storage density using a much less aggressive manufacturing process technology than planar NAND flash memory. The circuit-level and structural changes in 3D NAND flash memory significantly alter how different error sources affect the reliability of the memory.

Through experimental characterization of real, state-of-the-art 3D NAND flash memory chips, we find that 3D NAND flash memory exhibits *three* new error sources that were *not* previously observed in planar NAND flash memory: (1) *layer-to-layer process variation*, where the error rate of each layer of memory in the 3D stack is very different; (2) *early retention loss*, where charge leaks quickly out of a flash cell soon after the cell is programmed; and (3) *retention interference*, where the charge leakage speed of a flash cell depends on the value stored in the neighboring cell.

Based on our experimental results, we develop new analytical models for layer-to-layer process variation and retention loss effects in 3D NAND flash memory. Motivated by our new findings and models, we develop four new techniques to mitigate process variation and early retention loss in 3D NAND flash memory. *These techniques are examples of our approach that exploits device awareness in the flash controller.* Our first technique, layer-to-layer variation aware reading (LaVAR), reduces the effect of layer-to-layer process variation by tuning the read reference voltage for each layer. Our second technique, layer-interleaved RAID (LI-RAID), reorganizes how pages from different layers are paired together for RAID to reduce the overall error count. Our third technique, retention model aware reading (ReMAR), uses our retention model to adapt the read reference voltage *to* each cell based on the cell's retention age. Our fourth technique, neighbor-cell assisted retention interference correction (NARIC), mitigates retention interference by predicting and adapting the read reference voltages to the amount of interference during each read operation. Compared with similar state-of-the-art error mitigation techniques developed for planar NAND flash memory, LaVAR and ReMAR reduce the average raw bit error rate by 51.9% and 43.3%, respectively, while LI-RAID reduces the worst-case RBER by 66.9%. We conclude that our newly-proposed techniques successfully mitigate the new error patterns that we discover in 3D NAND flash memory. More details are in Chapter 6 and our paper under submission [198].

1.3.4 HeatWatch: Self-Recovery and Temperature Aware Retention Error Mitigation

NAND flash memory wearout can be partially repaired on its own during the idle time between program or erase operations (known as the *dwell time*), via a phenomenon known as the *self-recovery effect* [224, 337]. We can exploit the self-recovery effect to *significantly* improve flash lifetime, by applying *high temperature* to the flash memory during P/E cycling to amplify the self-recovery effect. As NAND flash memory lifetime continues to reduce due to reduced reliability, the self-recovery effect provides an appealing opportunity to mitigate the poor lifetime.

While flash self-recovery has been studied for planar 2D NAND flash memory in the past [224, 337], the self-recovery effect in 3D NAND flash memory is *not* well known, despite the rapidly-growing commercial popularity of 3D NAND flash memory. We close this gap by characterizing the effects of self-recovery and temperature on *real, state-of-the-art 3D NAND devices*. We show that these effects significantly change the *program accuracy* (i.e., the robustness of flash program operations) and the retention loss speed (i.e., the speed at which a flash cell leaks charge). We demonstrate that self-recovery and temperature affect 3D NAND flash *quite differently* from how they affect planar NAND flash, rendering prior models of self-recovery and temperature ineffective for 3D NAND flash. Using our characterization results, we develop a new unified model for 3D NAND self-recovery and temperature effects called *URT*, Unified self-Recovery and Temperature. URT provides a model for raw bit error rate and threshold voltage distribution based on wearout, retention time, dwell time, and temperature.

Based on our new findings and our new model, we propose *HeatWatch*, a mechanism that aims to improve 3D NAND flash reliability and lifetime. *This is an example of our approach that improves flash reliability by exploiting device-level behavior.* The key idea of HeatWatch

is to optimize read operations by adapting to the dwell time of the workload and the current operating temperature. HeatWatch first efficiently tracks flash temperature and the time of each operation online. Then, HeatWatch uses this information to apply URT in order to optimize the read reference voltages. Our detailed experimental evaluations show that HeatWatch reduces the raw bit error rate by 93.5% and improves flash lifetime by 3.85× over a baseline using a fixed read voltage, averaged across 28 real workload traces. More details are in Chapter 7 and our HPCA 2018 paper [199].

1.4 Thesis Outline

In Chapter 2, we introduce the basics of modern SSDs and NAND flash memory. In Chapter 3, we provide additional background and discuss related work on NAND flash memory reliability. A significant fraction of the material in Chapter 2 and Chapter 3 is borrowed from the author's co-authored work that appears as an invited paper in the Proceedings of the IEEE [33] and placed on arxiv.org [32]. In Chapter 4, we introduce WARM, which is published in MSST [194]. In Chapter 5, we introduce online flash channel modeling, which is published in JSAC [195]. In Chapter 6, we introduce our characterization, modeling, and mitigation techniques for 3D NAND flash memory, which is currently under submission [198]. In Chapter 7, we introduce HeatWatch, which is published in HPCA [199].

1.5 Contributions

To our knowledge, this dissertation is the first to propose various *device-* and *workload*-aware techniques in the flash controller that significantly improve *both* planar NAND and 3D NAND reliability at low cost. This dissertation makes the following major contributions to the field:

1. We propose a new technique called *WARM* that exploits the write-hotness of the workload in the flash controller to eliminate redundant flash refresh operations. Chapter 4 describes WARM in detail.

    1.1. We propose *Write-hotness Aware Retention Management* (WARM), a heterogeneous retention management policy for NAND flash memory that physically partitions write-hot data and write-cold data into two separate groups of flash blocks, so that we can relax the retention time for only the write-hot data *without* the need for refreshing such data. We show that doing so improves flash lifetime by 3.24× on average across a variety of server and system workloads, over a write-hotness-oblivious baseline that does not perform any refresh. WARM can also be combined with an adaptive refresh mechanism [22, 25] to further improve flash lifetime by 12.9× over the baseline.

    1.2. We propose a mechanism that combines write-hotness-aware retention management with an adaptive refresh mechanism [22, 25]. By using WARM and applying refresh to write-cold data, we can further improve flash lifetime due to the benefits of both techniques. We show that the combined approach improves flash lifetime by 1.21× over using adaptive refresh alone homogeneously across the entire flash memory.

1.3. We propose a simple, yet effective, window-based online algorithm to identify frequently-written pages. This mechanism can dynamically adapt to workload behavior and correctly size the identified subset of write-hot pages. This mechanism fully exploits the existing data structures in the flash controller to keep track of the write-hotness to reduce the tracking overhead. We believe this mechanism can also be used for purposes other than lifetime management, such as cache performance and energy management.

2. We propose a new framework that effectively learns an online model for the flash channel, while the controller and memory are under operation. Our new model can be learned with low performance overhead and can accurately emulate the error characteristics of each flash chip. Chapter 5 describes our framework and its applications.

   2.1. We provide an experimental characterization of the threshold voltage distribution, and how the distribution changes with wear, for state-of-the-art 1X-nm MLC planar NAND flash memory chips. Like prior work, we find that program errors can cause the tail of the distribution to fatten significantly, but, *unlike* prior work, we observe that this fat tail can show up *much earlier* in the lifetime of the flash device than previously thought.

   2.2. We propose a new, simple, and accurate static model for the threshold voltage distribution of MLC NAND flash memory at a particular P/E cycle count, based upon our modified version of the Student's t-distribution. The model is capable of accurately capturing the threshold voltage distribution, with a 0.68% average modeling error, while requiring little computation in the flash controller.

   2.3. We propose a new model to dynamically estimate how the threshold voltage distribution shifts as a function of P/E cycles. This model works in conjunction with our proposed static model, and it accurately predicts how the threshold voltage distribution changes in the future, with an average modeling error of 2.72%.

   2.4. We demonstrate several *practical* uses of our online threshold voltage distribution model in a flash controller, which allows the flash controller to dynamically adapt its policies to threshold voltage shifts and thereby better improve flash memory reliability. We propose a new mechanism to estimate the actual remaining flash lifetime, based on the expected growth in bit error rate. Our mechanisms improve flash memory lifetime by 48.9% and/or enable the flash device to safely endure 69.9% more P/E cycles than the manufacturer specification.

3. We perform a detailed, comprehensive characterization and analysis of 3D NAND flash memory errors using state-of-the-art, real 3D NAND flash memory chips. Through this analysis, we observe three new error characteristics in 3D NAND flash memory due to its unique structure and cell design. To mitigate these errors, we develop four new mechanisms within the flash controller, called *LaVAR*, *LI-RAID*, *ReMAR*, and *NARIC*. Chapter 6 describes our characterization and analysis, and the proposed mechanisms in detail.

   3.1. We perform the *first comprehensive experimental characterization* of 3D NAND flash memory errors using real, state-of-the-art MLC 3D NAND flash memory chips.

Based on this characterization, we present an in-depth comparison and analysis of the different error characteristics of the well-known error types between 3D NAND and planar NAND flash memories.

3.2. We present the *first in-depth analysis* of layer-to-layer process variation, early retention loss, and retention interference, *three new error characteristics* inherent to 3D NAND flash memory.

3.3. We develop *new analytical models* for (1) layer-to-layer process variation and (2) early retention loss in 3D NAND flash memory.

3.4. We propose a new mechanism called *Layer Variation Aware Reading* (LaVAR) to mitigate the effect of layer-to-layer process variation. LaVAR uses our layer-to-layer process variation model to fine-tune the read reference voltage independently for each 3D stack memory layer. On average, LaVAR reduces raw bit error rate by 43.3% over a variation-agnostic baseline.

3.5. We propose a new RAID (i.e., Redundant Array of Independent Disks) scheme called *Layer-Interleaved RAID* (LI-RAID) to mitigate the error rate variation between pages within each block caused by manufacturing process variation and MSB-LSB variation. LI-RAID eliminates the page with the worst-case error rate within a block by pairing up pages from different layers and from different bits within a cell. By doing this, LI-RAID reduces 99th-percentile tail raw bit error rate by 63.8% over a baseline using LaVAR.

3.6. We propose a new mechanism called *Retention Model Aware Reading* (ReMAR) to mitigate early retention errors in 3D NAND. ReMAR tracks the retention age of the data by recording the programming time of each block and uses our retention model to adapt the read reference voltage to the retention age of the data. On average, ReMAR reduces raw bit error rate by 51.9% over a retention-agnostic baseline.

3.7. We propose *Neighbor-cell Assisted Retention Interference Correction* (NARIC) to mitigate retention interference in 3D NAND devices. NARIC improves upon Neighbor-cell Assisted Correction (NAC) [26], previously proposed to mitigate only cell-to-cell program interference. In addition to NAC, NARIC also predicts and adapts the read reference voltages to the amount of retention interference based on the threshold voltage of the cells on adjacent wordlines.

4. We propose a new technique called *HeatWatch*, a new mechanism to improve 3D NAND flash memory reliability. We observe that, due to self-recovery, data retention in 3D NAND is significantly affected by temperature and dwell time. HeatWatch significantly improves flash reliability in 3D NAND by adapting the flash controller algorithms to self-recovery and temperature. Chapter 7 describes our characterization of self-recovery effect and HeatWatch in detail.

4.1. Using real, state-of-the-art 3D charge trap NAND flash chips from a major vendor, we experimentally characterize the effects of self-recovery and temperature on retention loss speed and program variation. We show that 3D NAND flash memory exhibits different self-recovery and temperature effects than planar NAND flash memory.

4.2. Based on our experimental characterization data, we construct URT, a unified model for retention loss, wearout, self-recovery, and temperature in 3D NAND flash memory. Our model quantifies these four effects to accurately predict the raw bit error rate and threshold voltage shift.

4.3. We propose HeatWatch, a mechanism for 3D NAND flash memory that improves flash reliability and lifetime. HeatWatch (1) tracks the temperature, dwell time, and retention time online, and (2) sends this information to URT to accurately predict the optimal read reference voltage. By using the predicted optimal read reference voltage for flash read operations, HeatWatch reduces the raw bit error rate by 93.5%, and improves flash lifetime by 3.85×, over a baseline that uses a fixed read reference voltage.

Chapter 2

Basics of Modern SSDs and NAND Flash Memory

In this chapter, we introduce the basic background of how modern SSDs work internally (Section 2.1), and how NAND flash memory operates to store data (Section 2.2). This background knowledge helps us understand the root cause of reliability issues for flash-memory-based SSDs as well as the state-of-the-art techniques to mitigate these issues, which are introduced in Chapter 3, and the new techniques we propose in Chapters 4–7.

2.1 State-of-the-Art SSD Architecture

In order to understand the root causes of reliability issues within SSDs, we first provide an overview of the system architecture of a state-of-the-art SSD. The SSD consists of a group of NAND flash memories (or *chips*) and a *controller*, as shown in Figure 2.1. A host computer communicates with the SSD through a high-speed host interface (e.g., AHCI, NVMe; see Section 2.1.3), which connects to the SSD controller. The controller is then connected to each of the NAND flash chips via memory *channels*.

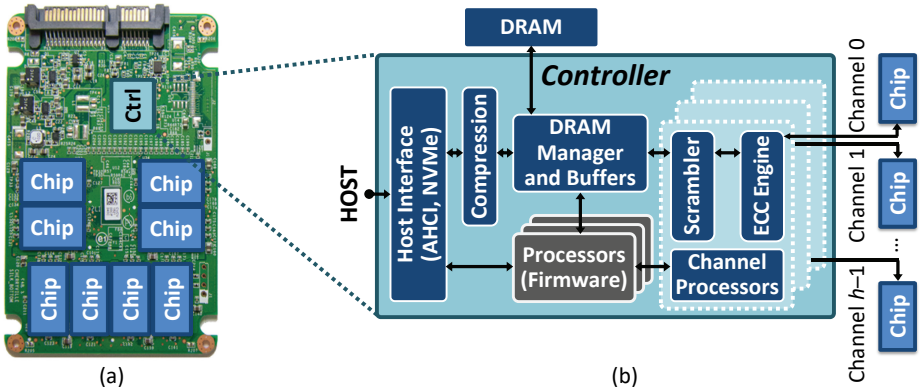


Figure 2.1: (a) SSD system architecture, showing controller (Ctrl) and chips. (b) Detailed view of connections between controller components and chips. Adapted from [32].

2.1.1 Flash Memory Organization

Figure 2.2 shows an example of how NAND flash memory is organized within an SSD. The flash memory is spread across multiple flash chips, where each chip contains one or more flash *dies*, which are individual pieces of silicon wafer that are connected together to the pins of the chip. Contemporary SSDs typically have 4–16 chips per SSD, and can have as many as 16 dies per chip. Each chip is connected to one or more physical memory channels, and these memory channels are not shared across chips. A flash die operates independently of other flash dies, and contains between one and four *planes*. Each plane contains hundreds to thousands of flash *blocks*. Each block is a 2D array that contains hundreds of rows of flash cells (typically 256–1024 rows) where the rows store contiguous pieces of data. Much like banks in a multi-bank memory (e.g., DRAM banks [40, 150, 152, 166, 168, 169, 170, 225, 234, 235]), the planes can execute flash operations in parallel, but the planes within a die share a single set of data and control buses [2]. Hence, an operation can be started in a different plane in the same die in a pipelined manner, every cycle. Figure 2.2 shows how blocks are organized within chips across multiple channels. In the rest of this work, without loss of generality, we assume that a chip contains a single die.

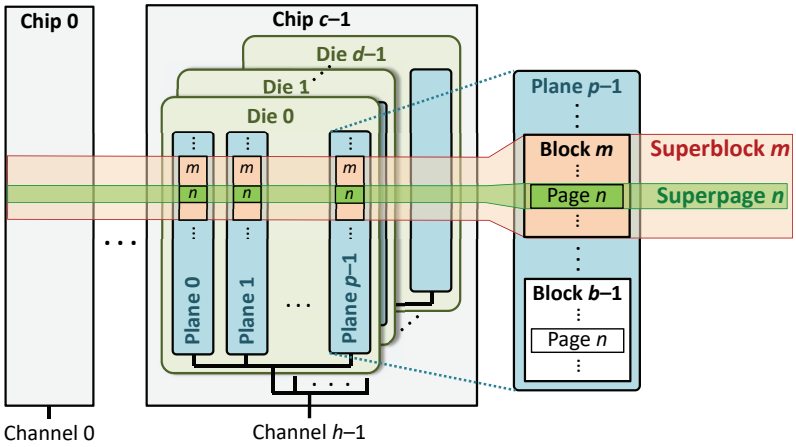


Figure 2.2: Flash memory organization. Reproduced from [32].

Data in a block is written at the unit of a *page*, which is typically between 8 and 16 kB in size in NAND flash memory. All read and write operations are performed at the granularity of a page. Each block typically contains hundreds of pages. Blocks in each plane are numbered with an ID that is unique within the plane, but is shared across multiple planes. Within the block, each page is numbered in sequence. The controller firmware groups blocks with the same ID number across multiple chips and planes together into a *superblock*. Within each superblock, the pages with the same page number are considered a *superpage*. The controller *opens* one superblock (i.e., an empty superblock is selected for write operations) at a time, and typically writes data to the NAND flash memory one superpage at a time to improve sequential read/write performance and make error correction efficient, since some parity information is kept at superpage granularity (see Section 2.1.3). Having the ability to write to all of the pages in a superpage simultaneously, the SSD can fully exploit the internal parallelism offered by multiple planes/chips, which in turn maximizes write throughput.

2.1.2 Memory Channel

Each flash memory channel has its own data and control connection to the SSD controller, much like a main memory channel has to the DRAM controller [87, 102, 103, 117, 150, 151, 153, 155, 226, 231, 234, 235, 305, 306, 307]. The connection for each channel is typically an 8- or 16-bit wide bus between the controller and one of the flash memory chips [2]. Both data and flash commands can be sent over the bus.

Each channel also contains its own control signal pins to indicate the type of data or command that is on the bus. The *address latch enable* (ALE) pin signals that the controller is sending an address, while the *command latch enable* (CLE) pin signals that the controller is sending a flash command. Every rising edge of the *write enable* (WE) signal indicates that the flash memory should write the piece of data currently being sent on the bus by the SSD controller. Similarly, every rising edge of the *read enable* (RE) signal indicates that the flash memory should send the next piece of data from the flash memory to the SSD controller.

Each flash memory die connected to a memory channel has its own *chip enable* (CE) signal, which selects the die that the controller currently wants to communicate with. On a channel, the bus broadcasts address, data, and flash commands to all dies within the channel, but only the die whose CE signal is active reads the information from the bus and executes the corresponding operation.

2.1.3 SSD Controller

The SSD controller, shown in Figure 2.1b, is responsible for (1) handling I/O requests received from the host, (2) ensuring data integrity and efficient storage, and (3) managing the underlying NAND flash memory. To perform these tasks, the controller runs firmware, which is often referred to as the *flash translation layer* (FTL). FTL tasks are executed on one or more embedded processors that exist inside the controller. The controller has access to DRAM, which can be used to store various controller metadata (e.g., how host memory addresses map to physical SSD addresses) and to cache relevant (e.g., frequently accessed) SSD pages [213, 279].

When the controller handles I/O requests, it performs a number of operations on both the requests and the data. For requests, the controller *schedules* them in a manner that ensures correctness and provides high/reasonable performance. For data, the controller *scrambles* the data to improve raw bit error rates, performs *ECC encoding/decoding*, and in some cases *compresses/decompresses* and/or *encrypts/decrypts* the data and employs *superpage-level data parity*. To manage the NAND flash memory, the controller runs *firmware* that maps host data to physical NAND flash pages, performs *garbage collection* on flash pages that have been invalidated, applies *wear leveling* to evenly distribute the impact of writes on NAND flash reliability across all pages, and manages bad NAND flash blocks. We briefly examine the various tasks of the SSD controller.

Scheduling Requests

The controller receives I/O requests over a *host controller interface* (shown as *Host Interface* in Figure 2.1b), which consists of a system I/O bus and the protocol used to communicate along

the bus. When an application running on the host system needs to access the SSD, it generates an I/O request, which is sent by the host over the host controller interface. The SSD controller receives the I/O request, and inserts the request into a queue. The controller uses a *scheduling policy* to determine the order in which the controller processes the requests that are in the queue. The controller then sends the request selected for scheduling to the FTL (part of the *Firmware* shown in Figure 2.1b).

The host controller interface determines how requests are sent to the SSD and how the requests are queued for scheduling. Two of the most common host controller interfaces used by modern SSDs are the Advanced Host Controller Interface (AHCI) [116] and NVM Express (NVMe) [247, 316, 317]. AHCI builds upon the Serial Advanced Technology Attachment (SATA) system bus protocol [290], which was originally designed to connect the host system to magnetic hard disk drives. AHCI allows the host to use advanced features with SATA, such as *native command queuing* (NCQ). When an application executing on the host generates an I/O request, the application sends the request to the operating system (OS). The OS sends the request over the SATA bus to the SSD controller, and the controller adds the request to a single *command queue*. NCQ allows the controller to schedule the queued I/O requests in a different order than the order in which requests were received (i.e., requests are scheduled *out of order*). As a result, the controller can choose requests from the queue in a manner that maximizes the overall SSD performance (e.g., a younger request can be scheduler earlier than an older request that requires access to a plane that is occupied with serving another request). A major drawback of AHCI and SATA is the limited throughput they enable for SSDs [346], as the protocols were originally designed to match the much lower throughput of magnetic hard disk drives. For example, a modern magnetic hard drive has a sustained read throughput of 300 MB/s [289], whereas a modern SSD has a read throughput of 3500 MB/s [283]. However, AHCI and SATA are widely deployed in modern computing systems, and they currently remain a common choice for the SSD host controller interface.

To alleviate the throughput bottleneck of AHCI and SATA, many manufacturers have started adopting host controller interfaces that use the PCI Express (PCIe) system bus [264]. A popular standard interface for the PCIe bus is the NVM Express (NVMe) interface [247]. Unlike AHCI, which requires an application to send I/O requests through the OS, NVMe directly exposes multiple SSD I/O queues to the applications executing on the host. By directly exposing the queues to the applications, NVMe simplifies the software I/O stack, eliminating most OS involvement [346], which in turn reduces communication overheads. An SSD using the NVMe interface maintains a separate set of queues for *each* application (as opposed to the single queue used for all applications with AHCI) within the host interface. With more queues, the controller (1) has a larger number of requests to select from during scheduling, increasing its ability to utilize idle resources (i.e., *channels*, *dies*, *planes*; see Section 2.1.1); and (2) can more easily manage and control the amount of interference that an application experiences from other concurrently-executing applications. Currently, NVMe is used by modern SSDs that are designed mainly for high-performance systems (e.g., enterprise servers, data centers [345, 346]). Recent work describes the state-of-the-art request scheduling algorithms in more detail [316, 317].

Flash Translation Layer

The main duty of the FTL (which is part of the *Firmware* shown in Figure 2.1) is to manage the mapping of *logical addresses* (i.e., the address space utilized by the host) to *physical addresses* in the underlying flash memory (i.e., the address space for actual locations where the data is stored, visible only to the SSD controller) for each page of data [63, 94]. By providing this indirection between address spaces, the FTL can *remap* the logical address to a different physical address (i.e., move the data to a different physical address) *without* notifying the host. Whenever a page of data is written to by the host or moved for underlying SSD maintenance operations (e.g., garbage collection [44, 350]; see Section 2.1.3), the old data (i.e., the physical location where the overwritten data resides) is simply marked as invalid in the physical block's *metadata*, and the new data is written to a page in the flash block that is currently open for writes (see Section 2.2.4 for more detail on how writes are performed).

The FTL is also responsible for *wear leveling*, to ensure that all of the blocks within the SSD are evenly worn out [44, 350]. By evenly distributing the *wear* (i.e., the number of P/E cycles that take place) *across* different blocks, the SSD controller reduces the heterogeneity of the amount of wearout across these blocks, thereby extending the lifetime of the device. The wear-leveling algorithm is invoked when the current block that is being written to is full (i.e., no more pages in the block are available to write to), and it enables the controller to select a new block from the *free list* to direct the future writes to. The wear-leveling algorithm dictates which of the blocks from the free list is selected. One simple approach is to select the block in the free list with the lowest number of P/E cycles to minimize the variance of the wearout amount across blocks, though many algorithms have been developed for wear leveling [43, 83].

Garbage Collection

When the host issues a write request to a logical address stored in the SSD, the SSD controller performs the write *out of place* (i.e., the updated version of the page data is written to a different physical page in the NAND flash memory), because in-place updates cannot be performed (see Section 2.2.4). The old physical page is marked as *invalid* when the out-of-place write completes. *Fragmentation* refers to the waste of space within a block due to the presence of invalid pages. In a *fragmented* block, a fraction of the pages are invalid, but these pages are unable to store new data until the page is erased. Due to circuit-level limitations, the controller can perform erase operations only at the granularity of an *entire block* (see Section 2.2.4 for details). As a result, until a fragmented block is erased, the block wastes physical space within the SSD. Over time, if fragmented blocks are not erased, the SSD will run out of pages that it can write new data to. The problem becomes especially severe if the blocks are highly fragmented (i.e., a large fraction of the pages within a block are invalid).

To reduce the negative impact of fragmentation on usable SSD storage space, the FTL periodically performs a process called *garbage collection*. Garbage collection finds highly-fragmented flash blocks in the SSD and recovers the wasted space due to invalid pages. The basic garbage collection algorithm [44, 350] (1) identifies the highly-fragmented blocks (which we call the *selected blocks*), (2) migrates any valid pages in a selected block (i.e., each valid page is written to a new block, its virtual-to-physical address mapping is updated, and the page in the selected

block is marked as invalid), (3) erases each selected block (see Section 2.2.4), and (4) adds a pointer to each selected block into the *free list* within the FTL. The garbage collection algorithm typically selects blocks with the highest number of invalid pages. When the controller needs a new block to write pages to, it selects one of the blocks currently in the free list.

We briefly discuss five optimizations that prior works propose to improve the performance and/or efficiency of garbage collection [2, 61, 94, 98, 105, 194, 271, 335, 350]. First, the garbage collection algorithm can be optimized to determine the most efficient frequency to invoke garbage collection [271, 350], as performing garbage collection too frequently can delay I/O requests from the host, while not performing garbage collection frequently enough can cause the controller to stall when there are no blocks available in the free list. Second, the algorithm can be optimized to select blocks in a way that reduces the number of page copy and erase operations required each time the garbage collection algorithm is invoked [98, 271]. Third, some works reduce the latency of garbage collection by using multiple channels to perform garbage collection on multiple blocks in parallel [2, 105]. Fourth, the FTL can minimize the latency of I/O requests from the host by pausing erase and copy operations that are being performed for garbage collection, in order to service the host requests immediately [61, 335]. Fifth, pages can be grouped together such that all of the pages within a block become invalid around the same time [94, 105, 194]. For example, the controller can group pages with (1) a similar degree of *write-hotness* (i.e., the frequency at which a page is updated; see Section 3.2.6) or (2) a similar *death time* (i.e., the time at which a page is overwritten). Garbage collection remains an active area of research.

Flash Reliability Management

The SSD controller performs many background optimizations that improve flash reliability. These flash reliability management techniques, as we will discuss in more detail in Section 3.2, can effectively improve flash lifetime at a very low cost, since the optimizations are usually performed during idle times, when the interference with the running workload is minimized. These management techniques sometimes require small metadata storage in memory (e.g., for storing the near-optimal read reference voltages [27, 35, 195]), or require a timer (e.g., for triggering refreshes in time [22, 25]).

Compression

Compression can reduce the size of the data written to minimize the number of flash cells worn out by the original data. Some controllers provide compression, as well as decompression, which reconstructs the original data from the compressed data stored in the flash memory [181, 364]. The controller may contain a *compression engine*, which, for example, performs the LZ77 or LZ78 algorithms. Compression is optional, as some types of data being stored by the host (e.g., JPEG images, videos, encrypted files, files that are already compressed) may not be compressible.

Data Scrambling and Encryption

The occurrence of errors in flash memory is highly dependent on the data values stored into the memory cells [21, 24, 26]. To reduce the dependence of the error rate on data values, an SSD controller first scrambles the data before writing it into the flash chips [36, 142]. The key idea of scrambling is to probabilistically ensure that the actual value written to the SSD contains an equal number of randomly distributed zeroes and ones, thereby minimizing any data-dependent behavior. Scrambling is performed using a reversible process, and the controller *descrambles* the data stored in the SSD during a read request. The controller employs a *linear feedback shift register* (LFSR) to perform scrambling and descrambling. An n-bit LFSR generates 2^{n-1} bits worth of pseudo-random numbers without repetition. For each page of data to be written, the LFSR can be seeded with the *logical* address of that page, so that the page can be correctly descrambled even if maintenance operations (e.g., garbage collection) migrate the page to another physical location, as the logical address is unchanged. (This also reduces the latency of maintenance operations, as they do not need to descramble and rescramble the data when a page is migrated.) The LFSR then generates a pseudo-random number based on the seed, which is then XORed with the data to produce the scrambled version of the data. As the XOR operation is reversible, the same process can be used to descramble the data.

In addition to the data scrambling employed to minimize data value dependence, several SSD controllers include data encryption hardware [64, 104, 331]. An SSD that contains data encryption hardware within its controller is known as a *self-encrypting drive* (SED). In the controller, data encryption hardware typically employs AES encryption [64, 69, 243, 331], which performs multiple rounds of substitutions and permutations to the unencrypted data in order to encrypt it. AES employs a separate key for each round [69, 243]. In an SED, the controller contains hardware that generates the AES keys for each round, and performs the substitutions and permutations to encrypt or decrypt the data using dedicated hardware [64, 104, 331].

Error-Correcting Codes

ECC is used to detect and correct the raw bit errors that occur within flash memory. A host writes a page of data, which the SSD controller splits into one or more chunks. For each chunk, the controller generates a *codeword*, consisting of the chunk and a correction code. The strength of protection offered by ECC is determined by the *coding rate*, which is the chunk size divided by the codeword size. A higher coding rate provides weaker protection, but consumes less storage, representing a key reliability tradeoff in SSDs.

The ECC algorithm employed (typically BCH [14, 107, 177, 297] or LDPC [84, 85, 203, 204, 297, 362]; see Section 3.3), as well as the length of the codeword and the coding rate, determine the total *error correction capability*, i.e., the maximum number of raw bit errors that can be corrected by ECC. ECC engines in contemporary SSDs are able to correct data with a relatively high raw bit error rate (e.g., between 10^{-3} and 10^{-2} [124]) and return data to the host at an error rate that meets traditional data storage reliability requirements (e.g., a post-correction error rate of 10^{-15} in the JEDEC standard [121]). The *error correction failure rate* (P_{ECFR}) of an ECC implementation, with a codeword length of l where the codeword has an error correction

capability of t bits, can be modeled as:

$$P_{ECFR} = \sum_{k=t+1}^{l} \binom{l}{k} (1 - \text{BER})^{(l-k)} \text{BER}^k \tag{2.1}$$

where BER is the bit error rate of the NAND flash memory. We assume in this equation that errors are independent and identically distributed.

In addition to the ECC information, a codeword contains cyclic redundancy checksum (CRC) parity information [279]. When data is being read from the NAND flash memory, there may be times when the ECC algorithm incorrectly indicates that it has successfully corrected all errors in the data, when uncorrected errors remain. To ensure that incorrect data is not returned to the user, the controller performs a CRC check in hardware to verify that the data is error free [266, 279].

Data Path Protection

In addition to protecting the data from raw bit errors within the NAND flash memory, newer SSDs incorporate error detection and correction mechanisms throughout the SSD controller, in order to further improve reliability and data integrity [279]. These mechanisms are collectively known as *data path protection*, and protect against errors that can be introduced by the various SRAM and DRAM structures that exist within the SSD.[1] Figure 2.3 illustrates the various structures within the controller that employ data path protection mechanisms. There are three data paths that require protection: (1) the path for data written by the host to the flash memory, shown as a red solid line in Figure 2.3; (2) the path for data read from the flash memory by the host, shown as a green dotted line; and (3) the path for metadata transferred between the firmware (i.e., FTL) processors and the DRAM, shown as a blue dashed line.

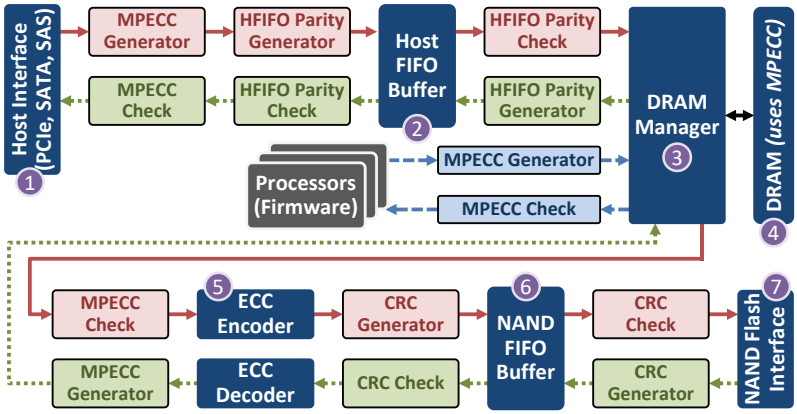


Figure 2.3: Data path protection employed within the controller. Reproduced from [32].

In the write data path of the controller (the red solid line shown in Figure 2.3), data received from the host interface (❶ in the figure) is first sent to a host FIFO buffer (❷). Before the data is written into the host FIFO buffer, the data is appended with *memory protection ECC* (MPECC)

[1] See Section 3.5 for a discussion on the possible types of errors that can be present in DRAM.

and *host FIFO buffer* (HFIFO) parity [279]. The MPECC parity is designed to protect against errors that are introduced when the data is stored within DRAM (which takes place later along the data path), while the HFIFO parity is designed to protect against SRAM errors that are introduced when the data resides within the host FIFO buffer. When the data reaches the head of the host FIFO buffer, the controller fetches the data from the buffer, uses the HFIFO parity to correct any errors, discards the HFIFO parity, and sends the data to the DRAM manager (❸). The DRAM manager buffers the data (which still contains the MPECC information) within DRAM (❹), and keeps track of the location of the buffered data inside the DRAM. When the controller is ready to write the data to the NAND flash memory, the DRAM manager reads the data from DRAM. Then, the controller uses the MPECC information to correct any errors, and discards the MPECC information. The controller then encodes the data into an ECC codeword (❺), generates CRC parity for the codeword, and then writes both the codeword and the CRC parity to a NAND flash FIFO buffer (❻) [279]. When the codeword reaches the head of this buffer, the controller uses CRC parity to detect any errors in the codeword, and then dispatches the data to the flash interface (❼), which writes the data to the NAND flash memory. Until the controller successfully write the data to the NAND flash memory, the data write is not considered durable. This is because the data stored in the DRAM or FIFO buffers can be lost if there is a power failure event [3]. The read data path of the controller (the green dotted line shown in Figure 2.3) performs the same procedure as the write data path, but in reverse order [279].

Aside from buffering data along the write and read paths, the controller uses the DRAM to store essential metadata, such as the table that maps each host data address to a physical block address within the NAND flash memory [213, 279]. In the metadata path of the controller (the blue dashed line shown in Figure 2.3), the metadata is often read from or written to DRAM by the firmware processors. In order to ensure correct operation of the SSD, the metadata must not contain any errors. As a result, the controller uses memory protection ECC (MPECC) for the metadata stored within DRAM [193, 279], just as it did to buffer data along the write and read data paths. Due to the lower rate of errors in DRAM compared to NAND flash memory (see Section 3.5), the employed memory protection ECC algorithms are not as strong as BCH or LDPC. We describe common ECC algorithms employed for DRAM error correction in Section 3.5.

Bad Block Management

Due to process variation or uneven wearout, a small number of flash blocks may have a much higher raw bit error rate (RBER) than an average flash block. Mitigating or tolerating the RBER on these flash blocks often requires a much higher cost than the benefit of using them. Thus, it is more efficient to identify and record these blocks as *bad blocks*, and avoid using them to store useful data. There are two types of bad blocks: *original bad blocks* (OBBs), which are defective due to manufacturing issues (e.g., process variation), and *growth bad blocks* (GBBs), which fail during runtime [318].

The flash vendor performs extensive testing, known as *bad block scanning*, to identify OBBs when a flash chip is manufactured [218]. Initially, all blocks are kept in the erased state, and contain the value 0xFF in each byte (see Section 2.2.1). Inside each OBB, the bad block scanning procedure writes a specific data value (e.g., 0x00) to a specific byte location within the block that indicates the block status. A good block (i.e., a block without defects) is not modified, and thus

its block status byte remains at the value 0xFF. When the SSD is powered up for the first time, the SSD controller iterates through all blocks and checks the value stored in the block status byte of each block. Any block that does not contain the value 0xFF is marked as bad, and is recorded in a *bad block table* stored in the controller. A small number of blocks in each plane are set aside as *reserved blocks* (i.e., blocks that are not used during normal operation), and the bad block table automatically remaps any operation originally destined to an OBB to one of the reserved blocks. The bad block table remaps an OBB to a reserved block in the same plane, to ensure that the SSD maintains the same degree of parallelism when writing to a superpage, thus avoiding performance loss. Less than 2% of all blocks in the SSD are expected to be OBBs [248].

The SSD identifies growth bad blocks during runtime by monitoring the status of each block. Each superblock contains a bit vector indicating which of its blocks are GBBs. After each program or erase operation to a block, the SSD reads the *status reporting registers* to check the operation status. If the operation has failed, the controller marks the block as a GBB in the superblock bit vector. At this point, the controller uses superpage-level parity to recover the data that was stored in the GBB (see Section 2.1.3), and *all data in the superblock* is copied to a different superblock. The superblock containing the GBB is then erased. When the superblock is subsequently opened, blocks marked as GBBs are *not* used, but the remaining blocks can store new data.

Superpage-Level Parity

In addition to ECC to protect against bit-level errors, many SSDs employ RAID-like parity [76, 132, 217, 262]. The key idea is to store parity information within each superpage to protect data from ECC failures that occur within a single chip or plane. Figure 2.4 shows an example of how the ECC and parity information are organized within a superpage. For a superpage that spans across multiple chips, dies, and planes, the pages stored within one die or one plane (depending on the implementation) are used to store parity information for the remaining pages. Without loss of generality, we assume for the rest of this section that a superpage that spans c chips and d dies per chip stores parity information in the pages of a single die (which we call the *parity die*), and that it stores user data in the pages of the remaining $(c \times d) - 1$ dies. When all of the user data is written to the superpage, the SSD controller XORs the data together one plane at a time (e.g., in Figure 2.4, all of the pages in Plane 0 are XORed with each other), which produces the parity data for that plane. This parity data is written to the corresponding plane in the parity die, e.g., Plane 0 page in Die $(c \times d) - 1$ in the figure.

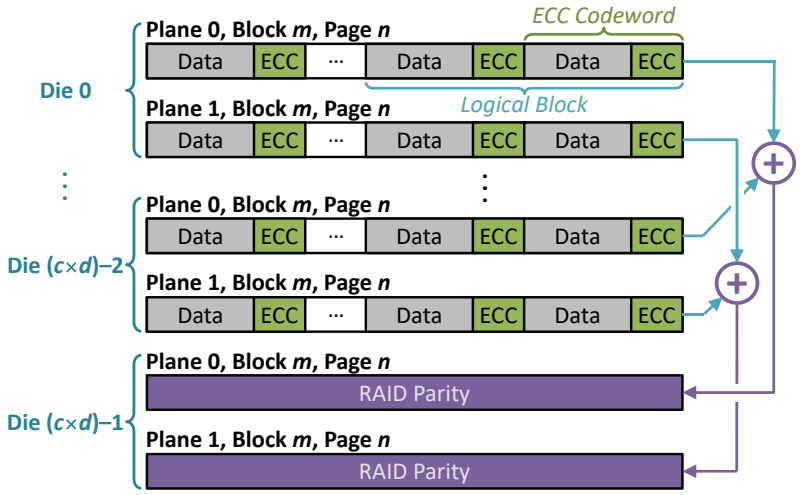


Figure 2.4: Example layout of ECC codewords, logical blocks, and superpage-level parity for superpage n in superblock m. In this example, we assume that a logical block contains two codewords. Reproduced from [32].

The SSD controller invokes superpage-level parity when an ECC failure occurs during a host software (e.g., OS, file system) access to the SSD. The host software accesses data at the granularity of a *logical block* (LB), which is indexed by a *logical block address* (LBA). Typically, an LB is 4 kB in size, and consists of several ECC codewords (which are usually 512 BB to 2 kB in size) stored consecutively within a flash memory page, as shown in Figure 2.4. During the LB access, a read failure can occur for one of two reasons. First, it is possible that the LB data is stored within a *hidden* GBB (i.e., a GBB that has not yet been detected and excluded by the bad block manager). The probability of storing data in a hidden GBB is quantified as P_{HGBB}. Note that because bad block management successfully identifies and excludes most GBBs, P_{HGBB} is much lower than the total fraction of GBBs within an SSD. Second, it is possible that at least one ECC codeword within the LB has *failed* (i.e., the codeword contains an error that cannot be corrected by ECC). The probability that a codeword fails is P_{ECFR} (see Section 2.1.3). For an LB that contains K ECC codewords, we can model P_{LBFail}, the overall probability that an LB access fails (i.e., the rate at which superpage-level parity needs to be invoked), as:

$$P_{LBFail} = P_{HGBB} + [1 - P_{HGBB}] \times [1 - (1 - P_{ECFR})^K] \tag{2.2}$$

In Equation 2.2, P_{LBFail} consists of (1) the probability that an LB is inside a hidden GBB (left side of the addition); and (2) for an LB that is not in a hidden GBB, the probability of any codeword failing (right side of the addition).

When a read failure occurs for an LB in plane p, the SSD controller reconstructs the data using the other LBs in the same superpage. To do this, the controller reads the LBs stored in plane p in the other $(c \times d) - 1$ dies of the superpage, including the LBs in the parity die. The controller then XORs all of these LBs together, which retrieves the data that was originally stored in the LB whose access failed. In order to correctly recover the failed data, all of the LBs from the $(c \times d) - 1$ dies must be correctly read. The overall superpage-level parity failure probability P_{parity} (i.e., the probability that more than one LB contains a failure) for an SSD with c chips of

flash memory, with d dies per chip, can be modeled as [262]:

$$P_{parity} = P_{LBFail} \times [1 - (1 - P_{LBFail})^{(c \times d)-1}] \tag{2.3}$$

Thus, by designating one of the dies to contain parity information (in a fashion similar to RAID 4 [262]), the SSD can tolerate the *complete failure* of the superpage data in one die without experiencing data loss during an LB access.

2.1.4 Design Tradeoffs for Reliability

Several design decisions impact the SSD *lifetime* (i.e., the duration of time that the SSD can be used within a bounded probability of error without exceeding a given performance overhead). To capture the tradeoff between these decisions and lifetime, SSD manufacturers use the following model:

$$\text{Lifetime (Years)} = \frac{\text{PEC} \times (1 + \text{OP})}{365 \times \text{DWPD} \times \text{WA} \times R_{compress}} \tag{2.4}$$

In Equation 2.4, the numerator is the total number of full drive writes the SSD can endure (i.e., for a drive with an X-byte capacity, the number of times X bytes of data can be written). The number of full drive writes is calculated as the product of PEC, the total P/E cycle *endurance* of each flash block (i.e., the number of P/E cycles the block can sustain before its raw error rate exceeds the ECC correction capability), and $1 + \text{OP}$, where OP is the *overprovisioning factor* selected by the manufacturer. Manufacturers overprovision the flash drive by providing more physical block addresses, or PBAs, to the SSD controller than the *advertised capacity* of the drive, i.e., the number of logical block addresses (LBAs) available to the operating system. Overprovisioning improves performance and endurance, by providing additional free space in the SSD so that maintenance operations can take place without stalling host requests. OP is calculated as:

$$\text{OP} = \frac{\text{PBA count} - \text{LBA count}}{\text{LBA count}} \tag{2.5}$$

The denominator in Equation 2.4 is the number of full drive writes per year, which is calculated as the product of days per year (i.e., 365), DWPD, and the ratio between the total size of the data written to flash media and the size of the data sent by the host (i.e., $\text{WA} \times R_{compress}$). DWPD is the number of full disk writes per day (i.e., the number of times per day the OS writes the advertised capacity's worth of data). DWPD is typically less than 1 for read-intensive applications, and could be greater than 5 for write-intensive applications [22]. WA (*write amplification*) is the ratio between the amount of data written into NAND flash memory by the controller over the amount of data written by the host machine. Write amplification occurs because various procedures (e.g., garbage collection [44, 350]; and remapping-based refresh, Section 3.2.3) in the SSD perform additional writes in the background. For example, when garbage collection selects a block to erase, the pages that are remapped to a new block require background writes. $R_{compress}$, or the compression ratio, is the ratio between the size of the compressed data and the size of the uncompressed data, and is a function of the entropy of the stored data and the efficiency of the compression algorithms employed in the SSD controller. In Equation 2.4, DWPD and $R_{compress}$ are largely determined by the workload and data compressibility, and cannot be changed to optimize flash lifetime. For controllers that do not implement compression, we set R compress to

1. However, the SSD controller can trade off other parameters between one another to optimize flash lifetime. We discuss the most salient tradeoffs next.

Tradeoff Between Write Amplification and Overprovisioning. As mentioned in Section 2.1.3, due to the granularity mismatch between flash erase and program operations, garbage collection occasionally remaps remaining valid pages from a selected block to a new flash block, in order to avoid block-internal fragmentation. This remapping causes additional flash memory writes, leading to *write amplification*. In an SSD with more overprovisioned capacity, the amount of write amplification decreases, as the blocks selected for garbage collection are older and tend to have fewer valid pages. For a greedy garbage collection algorithm and a random-access workload, the correlation between WA and OP can be calculated [75, 108], as shown in Figure 2.5. In an ideal SSD, both WA and OP should be minimal, i.e., WA = 1 and OP = 0%, but in reality there is a tradeoff between these parameters: when one increases, the other decreases. As Figure 2.5 shows, WA can be reduced by increasing OP, and with an infinite amount of OP, WA converges to 1. However, the reduction of WA is smaller when OP is large, resulting in diminishing returns.

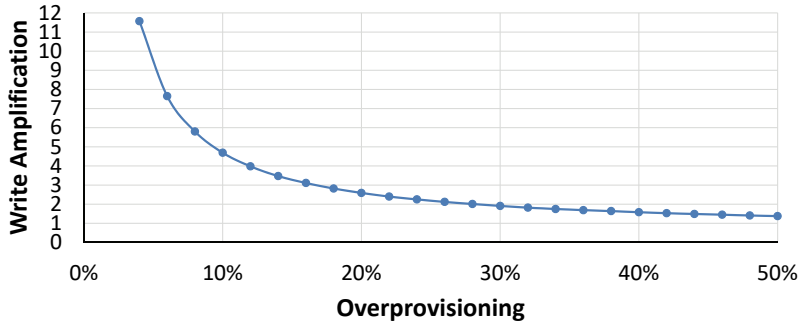


Figure 2.5: Relationship between write amplification (WA) and the overprovisioning factor (OP). Reproduced from [32].

In reality, the relationship between WA and OP is also a function of the storage space utilization of the SSD. When the storage space is *not* fully utilized, many more pages are available, reducing the need to invoke garbage collection, and thus WA can approach 1 without the need for a large amount of OP.

Tradeoff Between P/E Cycle Endurance and Overprovisioning. PEC and OP can be traded against each other by adjusting the amount of redundancy used for error correction, such as ECC and superpage-level parity (as discussed in Section 2.1.3). As the error correction capability increases, PEC increases because the SSD can tolerate the higher raw bit error rate that occurs at a higher P/E cycle count. However, this comes at a cost of reducing the amount of space available for OP, since a stronger error correction capability requires higher redundancy (i.e., more space). Table 2.1 shows the corresponding OP for four different error correction configurations for an example SSD with 2.0 TB of advertised capacity and 2.4 TB (20% extra) of physical space. In this table, the top two configurations use ECC-1 with a coding rate of 0.93, and the bottom two configurations use ECC-2 with a coding rate of 0.90, which has higher redundancy than ECC-1. Thus, the ECC-2 configurations have a lower OP than the top two. ECC-2, with its higher redundancy, can correct a greater number of raw bit errors, which in turn increases the

P/E cycle endurance of the SSD. Similarly, the two configurations with superpage-level parity have a lower OP than configurations without superpage-level parity, as parity uses a portion of the overprovisioned space to store the parity bits.

Table 2.1: Tradeoff between strength of error correction configuration and amount of SSD space left for overprovisioning.

Error Correction Configuration	Overprovisioning Factor
ECC-1 (0.93), no superpage-level parity	11.6%
ECC-1 (0.93), with superpage-level parity	8.1%
ECC-2 (0.90), no superpage-level parity	8.0%
ECC-2 (0.90), with superpage-level parity	4.6%

When the ECC correction strength is increased, the amount of overprovisioning in the SSD decreases, which in turn increases the amount of write amplification that takes place. Manufacturers must find and use the correct tradeoff between ECC correction strength and the overprovisioning factor, based on which of the two is expected to provide greater reliability for the target applications of the SSD.

2.2 NAND Flash Memory Basics

A number of underlying properties of the NAND flash memory used within the SSD affect SSD management, performance, and reliability [12, 16, 219]. In this section, we present a primer on NAND flash memory and its operation, to prepare the reader for understanding our further discussion on error sources (Section 3.1) and mitigation mechanisms (Section 3.2). Recall from Section 2.1.1 that within each plane, flash cells are organized as multiple 2D arrays known as flash blocks, each of which contains multiple pages of data, where a page is the granularity at which the host reads and writes data. We first discuss how data is stored in NAND flash memory. We then introduce the three basic operations supported by NAND flash memory: read, program, and erase.

2.2.1 Storing Data in a Flash Cell

NAND flash memory stores data as the *threshold voltage* of each flash cell, which is made up of a *floating gate transistor*. Figure 2.6 shows a cross section of a floating gate transistor. On top of a flash cell is the *control gate* (CG) and below is the floating gate (FG). The floating gate is insulated on both sides, on top by an inter-poly oxide layer and at the bottom by a tunnel oxide layer. As a result, the electrons programmed on the floating gate do not discharge even when flash memory is powered off.

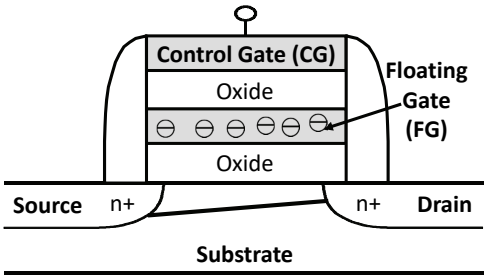


Figure 2.6: Flash cell (i.e., floating gate transistor) cross section. Reproduced from [32].

For *single-level cell* (SLC) NAND flash, each flash cell stores a 1-bit value, and can be programmed to one of two threshold voltage states, which we call the ER and P1 states. *Multi-level cell* (MLC) NAND flash stores a 2-bit value in each cell, with four possible states (ER, P1, P2, and P3), and *triple-level cell* (TLC) NAND flash stores a 3-bit value in each cell with eight possible states (ER, P1–P7). Each state represents a different value, and is assigned a *voltage window* within the range of all possible threshold voltages. Due to variation across *program* operations, the threshold voltage of flash cells programmed to the same state is initially distributed across this voltage window.

Figure 2.7 illustrates the threshold voltage distribution of MLC (top) and TLC (bottom) NAND flash memories. The x-axis shows the threshold voltage (V_{th}), which spans a certain voltage range. The y-axis shows the probability density of each voltage level across all flash memory cells. The threshold voltage distribution of each threshold voltage state can be represented as a probability density curve that spans over the state's voltage window.

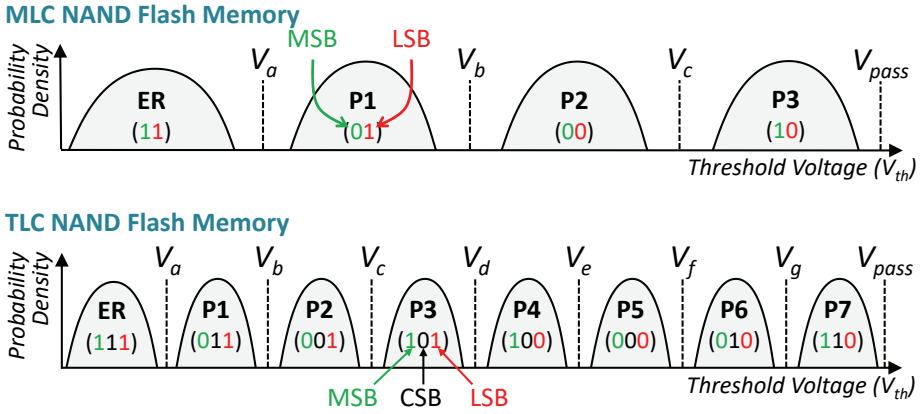


Figure 2.7: Threshold voltage distribution of MLC (top) and TLC (bottom) NAND flash memory. Reproduced from [32].

We label the distribution curve for each state with the name of the state and a corresponding bit value. Note that some manufacturers may choose to use a different mapping of values to different states. The bit values of adjacent states are separated by a Hamming distance of 1. We break down the bit values for MLC into the most significant bit (MSB) and least significant bit (LSB), while TLC is broken down into the MSB, the center significant bit (CSB), and the LSB.

The boundaries between neighboring threshold voltage windows, which are labeled as V_a, V_b, and V_c for the MLC distribution in Figure 2.7, are referred to as *read reference voltages*. These voltages are used by the SSD controller to identify the voltage window (i.e., state) of each cell upon reading the cell.

2.2.2 Flash Block Design

Figure 2.8 shows the high-level internal organization of a NAND flash memory block. Each block contains multiple rows of cells (typically 128–512 rows). Each row of cells is connected together by a common *wordline* (WL, shown horizontally in Figure 2.8), typically spanning 32K–64K cells. All of the cells along the wordline are logically combined to form a page in an SLC NAND flash memory. For an MLC NAND flash memory, the MSBs of all cells on the same wordline are combined to form an *MSB page*, and the LSBs of all cells on the wordline are combined to form an *LSB page*. Similarly, a TLC NAND flash memory logically combines the MSBs on each wordline to form an MSB page, the CSBs on each wordline to form a *CSB page*, and the LSBs on each wordline to form an LSB page. In MLC NAND flash memory, each flash block contains 256–1024 flash pages, each of which are typically 8 kB to 16 kB in size.

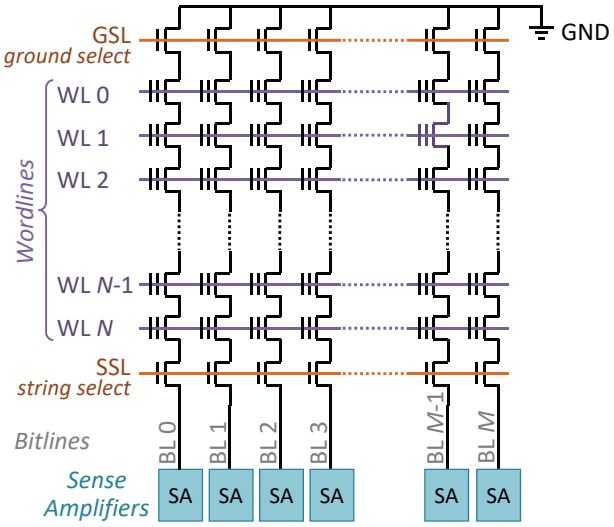


Figure 2.8: Internal organization of a flash block. Reproduced from [32].

Within a block, all cells in the same column are connected in series to form a *bitline* (BL, shown vertically in Figure 2.8) or *string*. All cells in a bitline share a common ground (GND) on one end, and a common *sense amplifier* (SA) on the other for reading the threshold voltage of one of the cells when decoding data. Bitline operations are controlled by turning the *ground select line* (GSL) and *string select line* (SSL) transistor of each bitline on or off. The SSL transistor is used to enable operations on a bitline, and the GSL transistor is used to connect the bitline to ground during a read operation [223]. The use of a common bitline across multiple rows reduces the amount of circuit area required for read and write operations to a block, improving storage density.

2.2.3 Read Operation

Data can be read from NAND flash memory by applying read reference voltages onto the control gate of each cell, to sense the cell's threshold voltage. To read the value stored in a single-level cell, we need to distinguish only the state with a bit value of 1 from the state with a bit value of 0. This requires us to use only a single read reference voltage. Likewise, to read the LSB of a multi-level cell, we need to distinguish only the states where the LSB value is 1 (ER and P1) from the states where the LSB value is 0 (P2 and P3), which we can do with a single read reference voltage (V_b in the top half of Figure 2.7). To read the MSB page, we need to distinguish the states with an MSB value of 1 (ER and P3) from those with an MSB value of 0 (P1 and P2). Therefore, we need to determine whether the threshold voltage of the cell falls between V_a and V_c, requiring us to apply each of these two read reference voltages (which can require up to two consecutive read operations) to determine the MSB.

Reading data from a triple-level cell is similar to the data read procedure for a multi-level cell. Reading the LSB for TLC again requires applying only a single read reference voltage (V_d in the bottom half of Figure 2.7). Reading the CSB requires two read reference voltages to be applied, and reading the MSB requires four read reference voltages to be applied.

As Figure 2.8 shows, cells from multiple wordlines (WL in the figure) are connected in series on a *shared* bitline (BL) to the sense amplifier, which drives the value that is being read from the block onto the memory channel for the plane. In order to read from a single cell on the bitline, *all of the other cells* (i.e., *unread* cells) on the same bitline must be switched on to allow the value that is being read to propagate through to the sense amplifier. The NAND flash memory achieves this by applying the *pass-through voltage* onto the wordlines of the unread cells, as shown in Figure 2.9a. When the pass-through voltage (i.e., the maximum possible threshold voltage V_{pass}) is applied to a flash cell, the source and the drain of the cell transistor are connected, regardless of the voltage of the floating gate. Modern flash memories guarantee that all *unread* cells are *passed through* to minimize errors during the read operation [35].

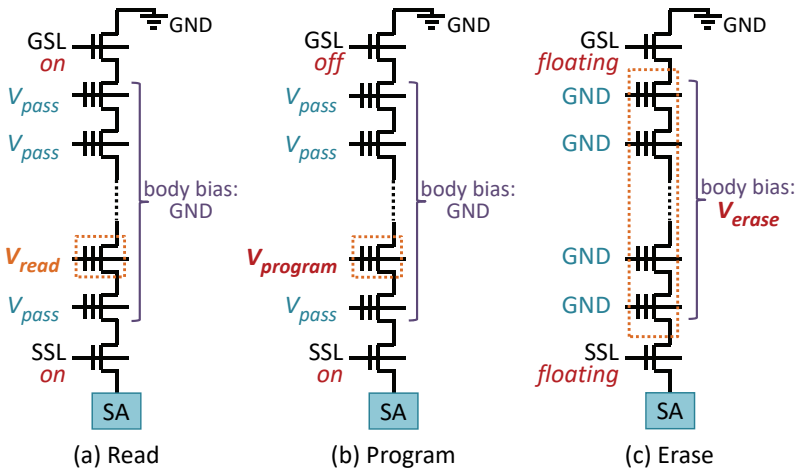


Figure 2.9: Voltages applied to flash cell transistors on a bitline to perform (a) read, (b) program, and (c) erase operations. Reproduced from [32].

2.2.4 Program and Erase Operations

The threshold voltage of a floating gate transistor is controlled through the injection and ejection of electrons through the tunnel oxide of the transistor, which is enabled by the FowlerNordheim (FN) tunneling effect [12, 81, 263]. The tunneling current (J_{FN}) [16, 263] can be modeled as:

$$J_{FN} = \alpha_{FN} E_{ox}^2 e^{-\beta_{FN}/E_{ox}} \tag{2.6}$$

In Equation 2.6, α_{FN} and β_{FN} are constants, and E_{ox} is the electric field strength in the tunnel oxide. As Equation 2.6 shows, J_{FN} is exponentially correlated with E_{ox}.

During a program operation, electrons are injected into the floating gate of the flash cell from the substrate when applying a high positive voltage to the control gate (see Figure 2.6 for a diagram of the flash cell). The pass-through voltage is applied to all of the other cells on the same bitline as the cell that is being programmed as shown in Figure 2.9b. When data is programmed, charge is transferred into the floating gate through FN tunneling by repeatedly pulsing the programming voltage, in a procedure known as *incremental step-pulse programming* (ISPP) [12, 219, 308, 327]. During ISPP, a high programming voltage ($V_{program}$) is applied for a very short period, which we refer to as a *step-pulse*. ISPP then verifies the current voltage of the cell using the voltage V_{verify}. ISPP repeats the process of applying a step-pulse and verifying the voltage until the cell reaches the desired target voltage. In the modern all-bitline NAND flash memory, all flash cells in a single wordline are programmed concurrently. During programming, when a cell along the wordline reaches its target voltage but other cells have yet to reach their target voltage, ISPP *inhibits* programming pulses to the cell by turning off the SSL transistor of the cell's bitline.

In SLC NAND flash and older MLC NAND flash, *one-shot programming* is used, where all of the ISPP step-pulses required to program a cell are applied back to back until all cells in the wordline are fully programmed. One-shot programming does *not* interleave the program operations to a wordline with the program operations to another wordline. In newer MLC NAND flash, the lack of interleaving between program operations can introduce a significant amount of cell-to-cell program interference on the cells of immediately-adjacent wordlines (see Section 3.1.3).

To reduce the impact of program interference, the controller employs *two-step programming* for sub-40 nm MLC NAND flash [24, 256]: it first programs the LSBs into the erased cells of an unprogrammed wordline, and then programs the MSBs of the cells using a separate program operation [23, 34, 255, 256]. Between the programming of the LSBs and the MSBs, the controller programs the LSBs of the cells in the wordline immediately above [23, 34, 255, 256]. Figure 2.10 illustrates the two-step programming algorithm. In the first step, a flash cell is *partially programmed* based on its LSB value, either staying in the ER state if the LSB value is 1, or moving to a temporary state (TP) if the LSB value is 0. The TP state has a mean voltage that falls between states P1 and P2. In the second step, the LSB data is first read back into an internal buffer register within the flash chip to determine the cell's current threshold voltage state, and then further programming pulses are applied based on the MSB data to increase the cell's threshold voltage to fall within the voltage window of its final state. Programming in MLC NAND flash is discussed in detail in [34] and [23].

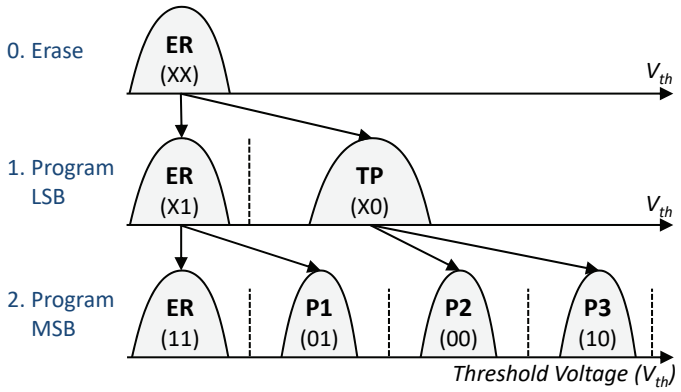


Figure 2.10: Two-step programming algorithm for MLC flash. Reproduced from [32].

TLC NAND flash takes a similar approach to the two-step programming of MLC, with a mechanism known as *foggy-fine programming* [182], which is illustrated in Figure 2.11. The flash cell is first partially programmed based on its LSB value, using a *binary* programming step in which very large ISPP step-pulses are used to significantly increase the voltage level. Then, the flash cell is partially programmed again based on its CSB and MSB values to a new set of temporary states (these steps are referred to as *foggy* programming, which uses smaller ISPP step-pulses than binary programming). Due to the higher potential for errors during TLC programming as a result of the narrower voltage windows, all of the programmed bit values are buffered after the binary and foggy programming steps into SLC buffers that are reserved in each chip/plane. Finally, *fine* programming takes place, where these bit values are read from the SLC buffers, and the smallest ISPP step-pulses are applied to set each cell to its final threshold voltage state. The purpose of this last fine programming step is to fine tune the threshold voltage such that the threshold voltage distributions are tightened (bottom of Figure 2.11).

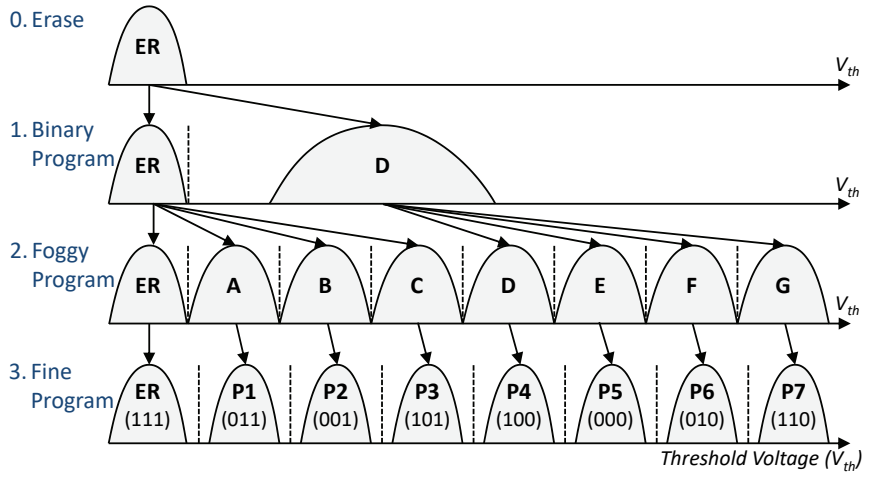


Figure 2.11: Foggy-fine programming algorithm for TLC flash. Reproduced from [32].

Though programming sets a flash cell to a specific threshold voltage using programming pulses, the voltage of the cell can drift over time after programming. When no external voltage

is applied to any of the electrodes (i.e., CG, source, and drain) of a flash cell, an electric field still exists between the FG and the substrate, generated by the charge present in the FG. This is called the *intrinsic electric field* [16], and it generates *stress-induced leakage current* (SILC) [12, 73, 242], a weak tunneling current that leaks charge away from the FG. As a result, the voltage that a cell is programmed to may not be the same as the voltage read for that cell at a subsequent time.

In NAND flash, a cell can be reprogrammed with new data *only after* the existing data in the cell is erased. This is because ISPP can only *increase* the voltage of the cell. The erase operation resets the threshold voltage state of *all cells in the flash block* to the ER state. During an erase operation, electrons are ejected from the FG of the flash cell into the substrate by inducing a high negative voltage on the cell transistor. The negative voltage is induced by setting the CG of the transistor to GND, and biasing the transistor body (i.e., the substrate) to a high voltage (V_{erase}), as shown in Figure 2.9c. Because all cells in a flash block share a common transistor substrate (i.e., the bodies of all transistors in the block are connected together), a flash block must be erased in its entirety [223].

Chapter 3

Flash Memory Reliability: Background and Related Work

In this chapter, we provide the background and related work on reliability issues in NAND flash memory. First, we provide a thorough introduction on NAND flash memory error characteristics concluded from prior work (Section 3.1). Second, we survey state-of-the-art techniques in modern SSDs that mitigate NAND flash memory errors (Section 3.2). Third, we introduce state-of-the-art error correction and recovery techniques that tolerate NAND flash memory errors (Section 3.3). Fourth, we introduce the state-of-the-art 3D NAND flash memory technology and the emerging reliability issues for 3D NAND devices using this technology (Section 3.4). Fifth, we discuss similar errors in other memory technologies and how we tolerate them in modern computing systems (Section 3.5).

3.1 NAND Flash Memory Error Characteristics

Each block in NAND flash memory is used in a cyclic fashion, as is illustrated by the observed raw bit error rates seen over the lifetime of a flash memory block in Figure 3.1. At the beginning of a *cycle*, known as a *program/erase (P/E) cycle*, an erased block is *opened* (i.e., selected for programming). Data is then programmed into the open block one page at a time. After all of the pages are programmed, the block is closed, and none of the pages can be reprogrammed until the whole block is erased. At any point before erasing, read operations can be performed on a *valid* programmed page (i.e., a page containing data that has not been modified by the host). A page is marked as invalid when the data stored at that page's logical address by the host is modified. As ISPP can only inject more charge into the floating gate but cannot remove charge from the gate, it is not possible to modify data to a new arbitrary value *in place* within existing NAND flash memories. Once the block is erased, the P/E cycling behavior repeats until the block is *worn out* (i.e., the block can no longer avoid data loss over the course of the minimum data retention period guaranteed by the manufacturer). Although the 5x-nm (i.e., 50 nm to 59 nm) generation of MLC NAND flash could endure $\sim$10,000 P/E cycles per block before being worn out, modern 1x-nm (i.e., 15 nm to 19 nm) MLC and TLC NAND flash can endure only $\sim$3,000 and $\sim$1,000 P/E cycles per block, respectively [157, 205, 260, 355].

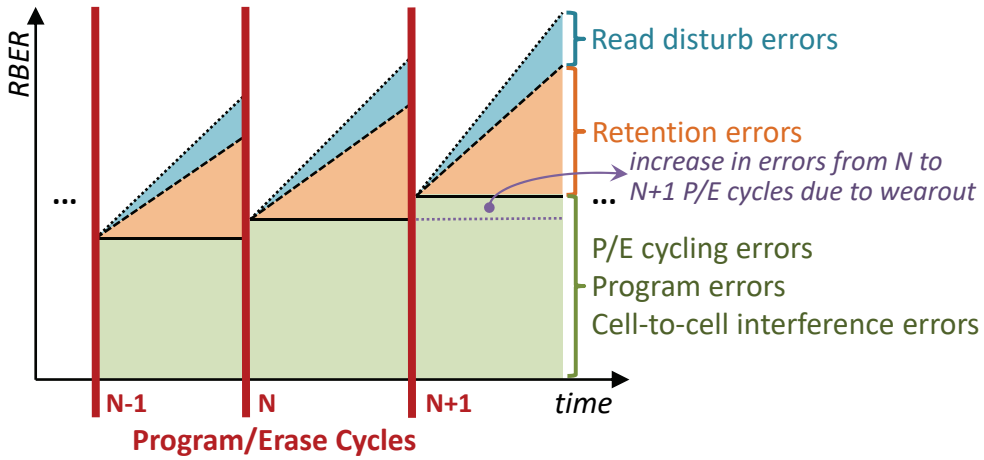


Figure 3.1: Pictorial depiction of errors accumulating within a NAND flash block as P/E cycle count increases. Reproduced from [32].

As shown in Figure 3.1, several different types of errors can be introduced at any point during the P/E cycling process: *P/E cycling errors*, *program errors*, errors due to *cell-to-cell program interference*, *data retention errors*, and errors due to *read disturb*. As discussed in Section 2.2.1, the threshold voltage of flash cells programmed to the same state is distributed across a voltage window due to variation across program operations and across different flash cells. Several types of errors introduced during the P/E cycling process, such as data retention and read disturb, cause the threshold voltage distribution of each state to shift and widen. Due to the shift and widening, the tails of the distributions of each state can enter the margin that originally existed between each of the two neighboring states' distributions. Thus, the threshold voltage distributions of different states can start overlapping, as shown in Figure 3.2. When the distributions overlap with each other, the read reference voltages can no longer correctly identify the state of some flash cells in the overlapping region, leading to *raw bit errors* during a read operation.

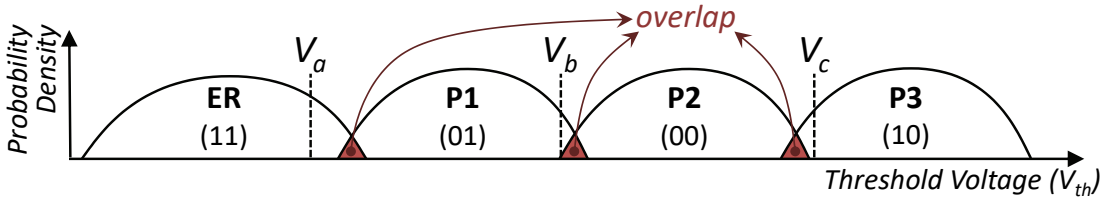


Figure 3.2: Threshold voltage distribution shifts and widening can cause the distributions of two neighboring states to overlap with each other (compare to Figure 2.7), leading to read errors. Reproduced from [32].

In this section, we discuss the causes of each type of error in detail. We later discuss mitigation techniques for these flash memory errors in Section 3.2, and provide procedures to recover in the event of data loss in Section 3.3.

3.1.1 P/E Cycling Errors

A P/E cycling error occurs when either (1) an erase operation fails to reset a cell to the ER state; or (2) when a program operation fails to set the cell to the desired target state. P/E cycling errors occur because electrons become trapped in the tunnel oxide after stress from repeated P/E cycles. Errors due to such electron trapping (which we refer to as *P/E cycling noise*) continue to accumulate over the lifetime of a NAND flash block. This behavior is called *wearout*, and it refers to the phenomenon where, as more writes are performed to a block, there are a greater number of raw bit errors that must be corrected, exhausting more of the fixed error correction capability of the ECC (see Section 2.1.3).

More findings on the nature of wearout and the impact of wearout on NAND flash memory errors and lifetime can be found in prior works from our research group [19, 21, 23, 195]. Recent work studies P/E cycling errors and proposes to change the ER state distribution to securely hide data in flash [365].

3.1.2 Program Errors

Program errors occur when data read directly from the NAND flash array contains errors, and the erroneous values are used to program the new data. Program errors occur in two major cases: (1) partial programming during two-step or foggy-fine programming, and (2) *copyback* (i.e., when data is copied inside the NAND flash memory during a maintenance operation) [109]. During two-step programming for MLC NAND flash memory (see Figure 2.10), in between the LSB and MSB programming steps of a cell, threshold voltage shifts can occur on the partially-programmed cell. These shifts occur because several other read and program operations to cells in *other* pages within the same block may take place, causing interference to the partially-programmed cell. Figure 3.3 illustrates how the threshold distribution of the ER state widens and shifts to the right after the LSB value is programmed (step 1 in the figure). The widening and shifting of the distribution causes some cells that were originally partially programmed to the ER state (with an LSB value of 1) to be misread as being in the TP state (with an LSB value of 0) during the *second* programming step (step 2 in the figure). As shown in Figure 3.3, the misread LSB value leads to a program error when the final cell threshold voltage is programmed [34, 195, 260]. Some cells that should have been programmed to the P1 state (representing the value 01) are instead programmed to the P2 state (with the value 00), and some cells that should have been programmed to the ER state (representing the value 11) are instead programmed to the P3 state (with the value 10).

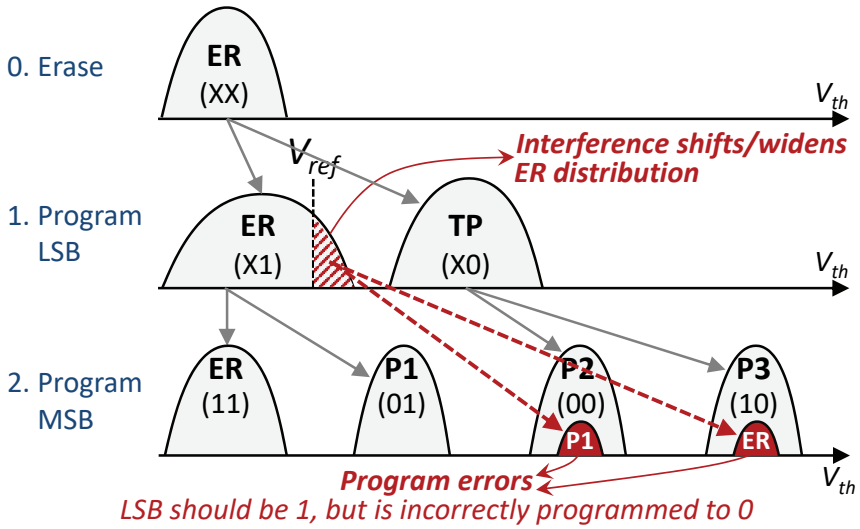


Figure 3.3: Impact of program errors during two-step programming on cell threshold voltage distribution. Reproduced from [32].

More findings on the nature of program errors and the impact of program errors on NAND flash memory lifetime can be found in prior works from our research group [34, 195].

3.1.3 Cell-to-Cell Program Interference Errors

Program interference refers to the phenomenon where the programming of a flash cell induces errors on adjacent flash cells within a flash block [24, 26, 68, 88, 174]. The interference occurs due to *parasitic capacitance coupling* between these cells. As a result, when the threshold voltage of an adjacent flash cell increases, the threshold voltage of the *victim cell* increases as well. The unintended threshold voltage shifts can eventually move a cell into a different state than the one it was originally programmed to, leading to a bit error.

Prior works have shown, based on the experimental analysis of modern MLC NAND flash memory chips, that the threshold voltage change of the victim cell can be accurately modeled as a linear combination of the threshold voltage changes of the adjacent cells when they are programmed, using linear regression with least-square-error estimation [24, 26]. The cells that are physically located immediately next to the victim cell (called the *immediately-adjacent cells*) are the major contributors to the cell-to-cell interference of a victim cell [24]. Figure 3.4 shows the eight immediately-adjacent cells for a victim cell in 2D planar NAND flash memory.

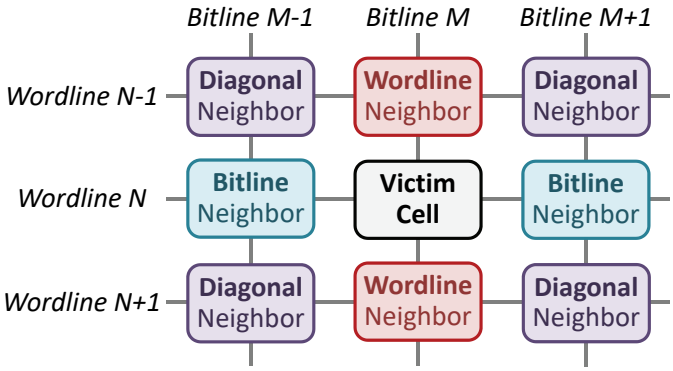


Figure 3.4: Immediately-adjacent cells that can induce program interference on a victim cell that is on wordline N and bitline M. Reproduced from [32].

The amount of interference that program operations to the immediately-adjacent cells can induce on the victim cell is expressed as:

$$\Delta V_{victim} = \sum_{X} K_X \Delta V_X \tag{3.1}$$

where ΔV_{victim} is the change in voltage of the victim cell due to cell-to-cell program interference, K_X is the *coupling coefficient* between cell X and the victim cell, and ΔV_X is the threshold voltage change of cell X during programming. The coupling coefficient is directly related to the effective capacitance C between cell X and the victim cell, which can be calculated as:

$$C = \varepsilon S/d \tag{3.2}$$

where ε is the permittivity, S is the effective cell area of cell X that faces the victim cell, and d is the distance between the cells. Of the immediately-adjacent cells, the wordline neighbor cells have the greatest coupling capacitance with the victim cell, as they likely have a large effective facing area to, and a small distance from, the victim cell compared to other surrounding cells.

The coupling coefficient grows as the feature size decreases [24, 26]. As NAND flash memory process technology scales down to smaller feature sizes, cells become smaller and get closer to each other, which increases the effective capacitance between them. As a result, at smaller feature sizes, it is easier for an immediately-adjacent cell to induce program interference on a victim cell. We conclude that (1) the program interference an immediately-adjacent cell induces on a victim cell is primarily determined by the distance between the cells and the immediately-adjacent cell's effective area facing the victim cell; and (2) the wordline neighbor cell causes the highest such interference, based on empirical measurements.

More findings on the nature of cell-to-cell program interference and the impact of cell-to-cell program interference on NAND flash memory errors and lifetime can be found in prior works from our research group [19, 24, 26, 34].

3.1.4 Data Retention Errors

Retention errors are caused by charge leakage over time after a flash cell is programmed, and are the dominant source of flash memory errors, as demonstrated previously [21, 22, 25, 27,

219, 312]. As flash memory process technology scales to smaller feature sizes, the capacitance of a flash cell, and the number of electrons stored on it, decreases. State-of-the-art (i.e., 1x-nm) MLC flash memory cells can store only ∼100 electrons [355]. Gaining or losing several electrons on a cell can significantly change the cell's voltage level and eventually alter its state. Charge leakage is caused by the unavoidable trapping of charge in the tunnel oxide [27, 173]. The amount of trapped charge increases with the electrical stress induced by repeated program and erase operations, which degrade the insulating property of the oxide.

Two failure mechanisms of the tunnel oxide lead to retention loss. *Trap-assisted tunneling* (TAT) occurs because the trapped charge forms an electrical tunnel, which exacerbates the weak tunneling current, SILC (see Section 2.2.4). As a result of this TAT effect, the electrons present in the floating gate (FG) leak away much faster through the intrinsic electric field. Hence, the threshold voltage of the flash cell decreases over time. As the flash cell wears out with increasing P/E cycles, the amount of trapped charge also increases [27, 173], and so does the TAT effect. At high P/E cycles, the amount of trapped charge is large enough to form percolation paths that significantly hamper the insulating properties of the gate dielectric [27, 73], resulting in retention failure. *Charge detrapping*, where charge previously trapped in the tunnel oxide is freed spontaneously, can also occur over time [27, 73, 173, 347]. The charge polarity can be either negative (i.e., electrons) or positive (i.e., holes). Hence, charge detrapping can either decrease or increase the threshold voltage of a flash cell, depending on the polarity of the detrapped charge.

More findings on the nature of data retention and the impact of data retention behavior on NAND flash memory errors and lifetime can be found in prior works from our research group [19, 21, 22, 25, 27].

3.1.5 Read Disturb Errors

Read disturb is a phenomenon in NAND flash memory where reading data from a flash cell can cause the threshold voltages of other (unread) cells in the same block to shift to a higher value [21, 35, 68, 88, 219, 252, 310]. While a single threshold voltage shift is small, such shifts can accumulate over time, eventually becoming large enough to alter the state of some cells and hence generate *read disturb errors*.

The failure mechanism of a read disturb error is similar to the mechanism of a normal program operation. A program operation applies a high programming voltage (e.g., +15 V) to the cell to change the cell's threshold voltage to the desired range. Similarly, a read operation applies a *high pass-through voltage* (e.g., +6 V) to *all other cells* that share the same bitline with the cell that is being read. Although the pass-through voltage is not as high as the programming voltage, it still generates a *weak programming effect* on the cells it is applied to [35], which can unintentionally change these cells' threshold voltages.

More findings on the nature of read disturb and the impact of read disturb on NAND flash memory errors and lifetime can be found in prior works from our research group [35].

3.1.6 Self-Recovery Effect

NAND flash memory has a limited lifetime because of transistor *wearout* as a flash cell is repeatedly programmed and erased. After each additional P/E cycle, a greater number of electrons get

inadvertently trapped within the flash cell, which changes the threshold voltage of the transistor [32, 33]. This threshold voltage change introduces errors and, thus, reduces the flash lifetime. Some of these inadvertently-trapped electrons gradually *escape* during the idle time between consecutive P/E cycles, i.e., *the dwell time*. The escape (i.e., *detrapping*) of the inadvertently-trapped electrons is known as the *self-recovery effect* [220], as it partially undoes (i.e., *repairs*) the wearout of the cell.

The self-recovery effect repairs the damage caused by flash wearout during the time between two P/E cycles, by *detrapping* some of the inadvertently-trapped charge [50, 176, 220, 224, 337, 338]. In this dissertation, we refer to the delay between consecutive program operations as the *dwell time*. The amount of repair done by self-recovery is affected by two factors: (1) dwell time and (2) operating temperature.

During the dwell time of a flash cell, a fraction of the charge that was inadvertently trapped in the tunnel oxide is slowly detrapped [220]. The reduction of inadvertently-trapped charge in a cell reduces the number of retention and program variation errors, and thus can extend the NAND flash memory lifetime. For a fixed retention time, a larger dwell time reduces the number of retention errors [220]. A *recovery cycle* refers to a P/E cycle where the program operation is followed by an extended dwell time. Since 3D NAND flash memory errors are dominated by retention errors [55, 221, 353], reducing the retention error rate by performing recovery cycles can increase flash lifetime significantly.

A higher operating temperature for NAND flash memory increases electron mobility [27, 220]. As a result, a short retention time at high temperature has the *same* retention loss effect as a longer retention time at room temperature [27], which we call the *effective retention time*. Similarly, a short dwell time at high temperature has the *same* self-recovery effect as a longer dwell time at room temperature [220], called the *effective dwell time*. The equivalence between time elapsed at a certain temperature and the corresponding effective time at room temperature can be modeled using Arrhenius' Law [6, 27, 126, 220]:

$$AF(T_1, T_2) = \frac{t_1}{t_2} = \exp\left(\frac{E_a}{k_B} \cdot \left(\frac{1}{T_1} - \frac{1}{T_2}\right)\right) \tag{3.3}$$

In Equation 3.3, AF is the *acceleration factor* between t_1 and t_2, where t_1 is the retention or dwell time under temperature T_1, and t_2 is the retention or dwell time under temperature T_2. k_B is the Boltzmann constant, which is $8.62\text{e}-5\,\text{eV/K}$. E_a is the activation energy, which is a manufacturing-process-dependent constant. For a planar NAND flash memory device, $E_a = 1.1\,\text{eV}$ [124]. To our knowledge, there is no public literature that reports the value of E_a for 3D NAND flash memory.

Prior work has proposed idealized circuit-level models for the self-recovery (or self-healing) effect [224, 337], demonstrating significant opportunities for using the self-recovery effect to improve flash reliability and lifetime. Based on the assumptions about how self-recovery effect works, prior work has also proposed techniques to exploit this effect to improve flash lifetime such as heal-leveling [45], write throttling [176], and heat-accelerated self-recovery [337]. However, these previous results are not yet convincing enough to show that self-recovery effect can successfully improve flash lifetime on real devices, because they lack real experimental data and evidence supporting the self-recovery effect on modern flash devices. Our characterization in

Chapter 7 is the first to demonstrate and comprehensively evaluate the benefit of self-recovery effect using experimental data from real 3D NAND flash memory chips.

3.1.7 Large-Scale Studies on SSD Errors

The error characterization studies we have discussed so far examine the susceptibility of real NAND flash memory devices to specific error sources, by conducting controlled experiments on individual flash devices in controlled environments. To examine the *aggregate* effect of these error sources on flash devices that operate in the field, several recent studies have analyzed the reliability of SSDs deployed at a large scale (e.g., hundreds of thousands of SSDs) in production data centers [213, 241, 285]. Unlike the controlled low-level error characterization studies discussed in Sections 3.1.1 through 3.1.5, these large-scale studies analyze the observed errors and error rates in an *uncontrolled* manner, i.e., based on real data center workloads operating at field conditions (as opposed to carefully controlling access patterns and operating conditions). As such, these large-scale studies can study flash memory behavior and reliability using only a black-box approach, where they are able to access only the registers used by the SSD to record select statistics. Because of this, their conclusions are usually correlational in nature, as opposed to identifying the underlying causes behind the observations. On the other hand, these studies incorporate the effects of a real system, including the system software stack and real workloads [213] and real operational conditions in data centers, on the flash memory devices, which is not present in the controlled small-scale studies.

These recent large-scale studies have made a number of observations across large sets of SSDs employed in the data centers of large internet companies: Facebook [213], Google [285], and Microsoft [241]. We highlight six key observations from these studies about the *SSD failure rate*, which is the fraction of SSDs that have experienced at least one uncorrectable error.

First, the number of uncorrectable errors observed varies significantly for each SSD. Figure 3.5 shows the distribution of uncorrectable errors per SSD across a large set of SSDs used by Facebook. The distributions are grouped into six different *platforms* that are deployed in Facebook's data center.[1] For every platform, prior work observes that the top 10% of SSDs, when sorted by their uncorrectable error count, account for over 80% of the total uncorrectable errors observed across all SSDs for that platform. Prior work finds that the distribution of uncorrectable errors across all SSDs belonging to a platform follows a Weibull distribution, which is shown using a solid black line in Figure 3.5.

[1] Each platform has a different combination of SSDs, host controller interfaces, and workloads. The six platforms are described in detail in [213].

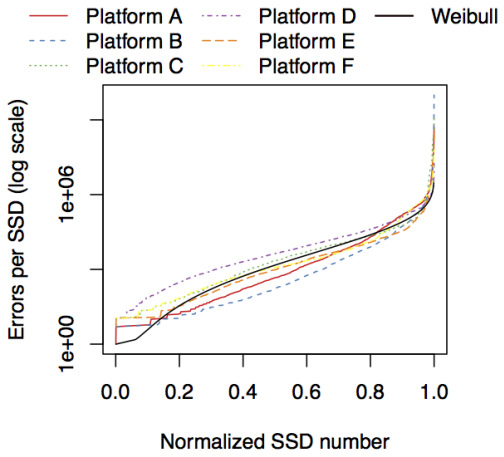


Figure 3.5: Distribution of uncorrectable errors across SSDs used in Facebook's data centers. Reproduced from [213].

Second, the SSD failure rate does *not* increase monotonically with the P/E cycle count. Instead, prior work observes several *distinct* periods of reliability, as illustrated pictorially and abstractly in Figure 3.6, which is based on data obtained from analyzing errors in SSDs used in Facebook's data centers [213]. The failure rate increases when the SSDs are relatively new (shown as the *early detection* period in Figure 3.6), as the SSD controller identifies unreliable NAND flash cells during the initial read and write operations to the devices and removes them from the address space (see Section 2.1.3). As the SSDs are used more, they enter the *early failure* period, where failures are less likely to occur. When the SSDs approach the end of their lifetime (*useful life/wearout* in the figure), the failure rate increases again, as more cells become unreliable due to wearout. Figure 3.7 shows how the measured failure rate changes as more writes are performed to the SSDs (i.e., how real data collected from Facebook's SSDs corresponds to the pictorial depiction in Figure 3.6) for the same six platforms shown in Figure 3.5. Prior work observes that the failure rates in each platform exhibit the distinct periods that are illustrated in Figure 3.6. For example, let us consider the SSDs in Platforms A and B, which have more data written to their cells than SSDs in other platforms. Prior work observes from Figure 3.7 that for SSDs in Platform A, there is an 81.7% increase from the failure rate during the early detection period to the failure rate during the wearout period [213].

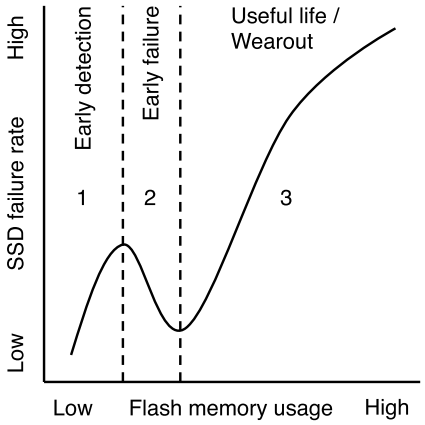


Figure 3.6: Pictorial and abstract depiction of the pattern of SSD failure rates observed in real SSDs operating in a modern data center. An SSD fails at different rates during distinct periods throughout the SSD lifetime. Reproduced from [213].

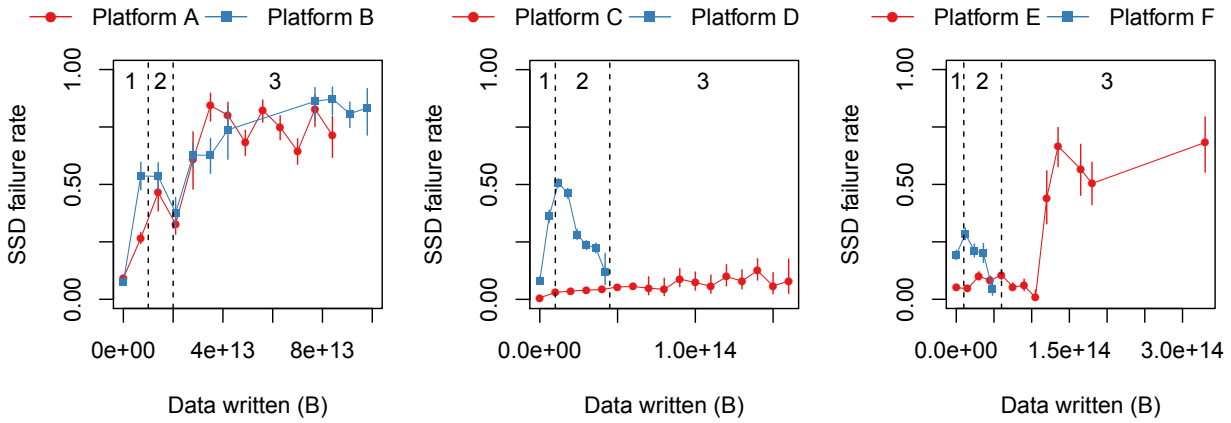


Figure 3.7: SSD failure rate vs. the amount of data written to the SSD. The three periods of failure rates, shown pictorially and abstractly in Figure 3.6, are annotated on each graph: (1) early detection, (2) early failure, and (3) useful life/wearout. Reproduced from [213].

Third, the raw bit error rate grows with the age of the device even if the P/E cycle count is held constant, indicating that mechanisms such as silicon aging likely contribute to the error rate [241].

Fourth, the observed failure rate of SSDs has been noted to be significantly higher than the failure rates specified by the manufacturers [285].

Fifth, higher operating temperatures can lead to higher failure rates, but modern SSDs employ throttling techniques that reduce the access rates to the underlying flash chips, which can greatly reduce the negative reliability impact of higher temperatures [213]. For example, Figure 3.8 shows the SSD failure rate as the SSD operating temperature varies, for SSDs from the same six platforms shown in Figure 3.5 [213]. Prior work observes that at an operating temperature range of 30 °C to 40 °C, SSDs either (1) have similar failure rates across the different temperatures, or (2) experience slight increases in the failure rate as the temperature increases. As the tempera-

ture increases beyond 40 °C, the SSDs fall into three categories: (1) temperature-sensitive with increasing failure rate (Platforms A and B), (2) less temperature-sensitive (Platforms C and E), and (3) temperature-sensitive with decreasing failure rate (Platforms D and F). There are two factors that affect the temperature sensitivity of each platform: (1) some, but not all, of the platforms employ techniques to throttle SSD activity at high operating temperatures to reduce the failure rate (e.g., Platform D); and (2) the platform configuration (e.g., the number of SSDs in each machine, system airflow) can shorten or prolong the effects of higher operating temperatures.

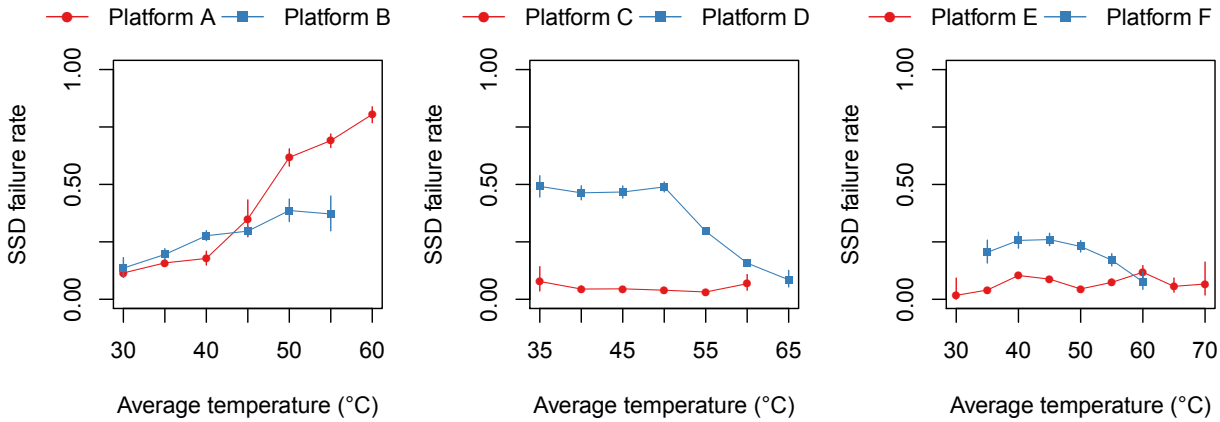


Figure 3.8: SSD failure rate vs. operating temperature. Reproduced from [213].

Sixth, while SSD failure rates are higher than specified by the manufacturers, the overall occurrence of *uncorrectable* errors is lower than expected [213] because (1) effective bad block management policies (see Section 2.1.3) are implemented in SSD controllers; and (2) certain types of error sources, such as read disturb [213, 241] and incomplete erase operations [241], have yet to become a major source of uncorrectable errors at the system level.

3.2 Error Mitigation

Several different types of errors can occur in NAND flash memory, as we described in Section 3.1. As NAND flash memory continues to scale to smaller technology nodes, the magnitude of these errors has been increasing [205, 260, 354]. This, in turn, uses up the limited error correction capability of ECC more rapidly than in past flash memory generations and shortens the lifetime of modern SSDs. To overcome the decrease in lifetime, a number of error mitigation techniques have been designed. These techniques exploit intrinsic properties of the different types of errors to reduce the rate at which they lead to raw bit errors. In this section, we discuss how the flash controller mitigates each of the error types via various proposed error mitigation mechanisms. Table 3.1 shows the techniques we overview and which errors (from Section 3.1) they mitigate.

Table 3.1: List of different types of errors mitigated by various NAND flash error mitigation mechanisms.

Mitigation Mechanism	*Error Type*				
	P/E Cycling [21, 23, 195] (§3.1.1)	***Program*** [34, 195, 260] (§3.1.2)	***Cell-to-Cell Interference*** [21, 24, 26, 174] (§3.1.3)	***Data Retention*** [21, 22, 25, 27, 219] (§3.1.4)	***Read Disturb*** [21, 35, 88, 219] (§3.1.5)
Shadow Program Sequencing [24, 34] (Section 3.2.1)			X		
Neighbor-Cell Assisted Error Correction [26] (Section 3.2.2)			X		
Refresh [22, 25, 222, 250] (Section 3.2.3)				X	X
Read-Retry [23, 82, 349] (Section 3.2.4)	X			X	X
Voltage Optimization [27, 35, 122] (Section 3.2.5)	X			X	X
Hot Data Management [95, 96, 194] (Section 3.2.6)	X	X	X	X	X
Adaptive Error Mitigation [31, 51, 99, 332, 336] (Section 3.2.7)	X	X	X	X	X

3.2.1 Shadow Program Sequencing

As discussed in Section 3.1.3, cell-to-cell program interference is a function of the distance between the cells of the wordline that is being programmed and the cells of the victim wordline. The impact of program interference is greatest on a victim wordline when either of the victim's immediately-adjacent wordlines is programmed (e.g., if we program WL1 in Figure 2.8, WL0 and WL2 experience the greatest amount of interference). Early MLC flash memories used one-shot programming, where both the LSB and MSB pages of a wordline are programmed at the same time. As flash memory scaled to smaller process technologies, one-shot programming resulted in much larger amounts of cell-to-cell program interference. As a result, manufacturers introduced two-step programming for MLC NAND flash (see Section 2.2.4), where the SSD controller writes values of the two pages within a wordline in two independent steps.

The SSD controller minimizes the interference that occurs during two-step programming by using *shadow program sequencing* [24, 34, 255] to determine the order that data is written to different pages in a block. If we program the LSB and MSB pages of the same wordline back to back, as shown in Figure 3.9a, both programming steps induce interference on a *fully-programmed wordline* (i.e., a wordline where both the LSB and MSB pages are already written).

For example, if the controller programs both pages of WL1 back to back, shown as bold page programming operations in Figure 3.9a, the program operations induce a high amount of interference on WL0, which is fully programmed. The key idea of shadow program sequencing is to ensure that a fully-programmed wordline experiences interference minimally, i.e., *only* during MSB page programming (and *not* during LSB page programming). In shadow program sequencing, prior work assigns a unique page number to each page within a block, as shown in Figure 3.9b. The LSB page of wordline i is numbered page $2i-1$, and the MSB page is numbered page $2i+2$. The only exceptions to the numbering are the LSB page of wordline 0 (page 0) and the MSB page of the last wordline n (page $2n+1$). Two-step programming writes to pages in *increasing* order of page number inside a block [24, 34, 255], such that a *fully-programmed* wordline experiences interference only from the MSB page programming of the wordline directly above it, shown as the bold page programming operation in Figure 3.9b. With this programming order/sequence, the LSB page of the wordline above, and both pages of the wordline below, do *not* cause interference to fully-programmed data [24, 34, 255], as these two pages are programmed *before* programming the MSB page of the given wordline. Foggy-fine programming in TLC NAND flash (see Section 2.2.4) uses a similar ordering to reduce cell-to-cell program interference, as shown in Figure 3.9c.

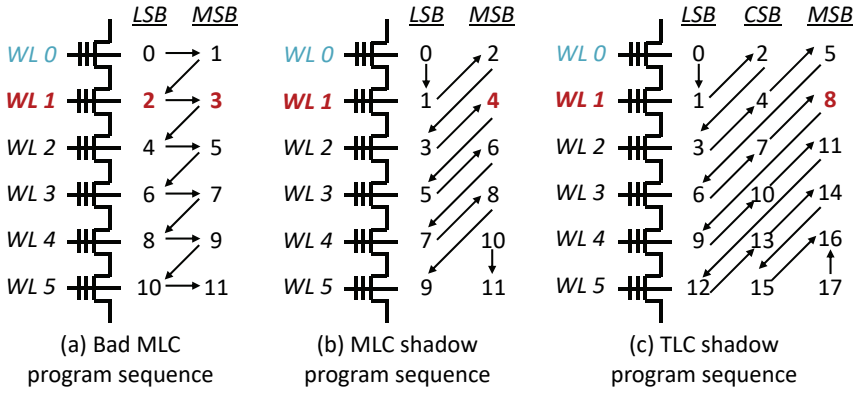


Figure 3.9: Order in which the pages of each wordline (WL) are programmed using (a) a bad programming sequence, and using shadow sequencing for (b) MLC and (c) TLC NAND flash. The bold page programming operations for WL1 induce cell-to-cell program interference when WL0 is fully programmed. Reproduced from [32].

Shadow program sequencing is an effective solution to minimize cell-to-cell program interference on fully-programmed wordlines during two-step programming, and is employed in commercial SSDs today.

3.2.2 Neighbor-Cell Assisted Error Correction

The threshold voltage shift that occurs due to program interference is highly correlated with the values stored in the cells of the *immediately-adjacent wordlines*, as we discussed in Section 3.1.3. Due to this correlation, knowing the value programmed in the immediately-adjacent cell (i.e., a *neighbor cell*) makes it easier to correctly determine the value stored in the flash cell that is being

read [26]. We describe a recently-proposed error correction method that takes advantage of this observation, called *neighbor-cell-assisted error correction* (NAC). The key idea of NAC is to use the data values stored in the cells of the immediately-adjacent wordline to determine a better set of read reference voltages for the wordline that is being read. Doing so leads to a more accurate identification of the logical data value that is being read, as the data in the immediately-adjacent wordline was *partially responsible* for shifting the threshold voltage of the cells in the wordline that is being read when the immediately-adjacent wordline was programmed.

Figure 3.10 shows an operational example of NAC that is applied to eight bitlines (BL) of an MLC flash wordline. The SSD controller first reads a flash page from a wordline using the standard read reference voltages (step 1 in Figure 3.10). The bit values read from the wordline are then buffered in the controller. If there are no errors uncorrectable by ECC, the read was successful, and nothing else is done. However, if there are errors that are *uncorrectable* by ECC, we assume that the threshold voltage distribution of the page shifted due to cell-to-cell program interference, triggering further correction. In this case, NAC reads the LSB and MSB pages of the wordline *immediately above* the requested page (i.e., the *adjacent* wordline that was programmed *after* the requested page) to classify the cells of the requested page (step 2). NAC then identifies the cells adjacent to (i.e., connected to the same bitline as) the ER cells (i.e., cells in the immediately above wordline that are in the ER state), such as the cells on BL1, BL3, and BL7 in Figure 3.10. NAC rereads these cells using read reference voltages that *compensate for* the threshold voltage shift caused by programming the adjacent cell to the ER state (step 3). If ECC can correct the remaining errors, the controller returns the corrected page to the host. If ECC fails again, the process is repeated using a different set of read reference voltages for cells that are adjacent to the P1 cells (step 4). If ECC continues to fail, the process is repeated for cells that are adjacent to P2 and P3 cells (steps 5 and 6, respectively, which are not shown in the figure) until either ECC is able to correct the page or all possible adjacent values are exhausted.

		BL0	BL1	BL2	BL3	BL4	BL5	BL6	BL7
	Originally-programmed value	11	00	01	10	11	00	01	00
	1. Read (using V_{opt}) with *errors*	01	00	00	00	11	10	00	01
NAC	2. Read **adjacent wordline**	P2	ER	P2	ER	P1	P3	P1	ER
NAC	3. *Correct cells* adjacent to ER	01	00	00	***10***	11	10	00	***00***
NAC	4. *Correct cells* adjacent to P1	01	00	00	10	11	10	***01***	00

Figure 3.10: Overview of neighbor-cell-assisted error correction (NAC). Reproduced from [32].

NAC extends the lifetime of an SSD by reducing the number of errors that need to be corrected using the limited correction capability of ECC. With the use of experimental data collected from real MLC NAND flash memory chips, prior work shows that NAC extends the NAND flash memory lifetime by 33% [26]. Previous work from our research group [26] provides a detailed description of NAC, including a theoretical treatment of why it works and a practical implementation that minimizes the number of reads performed, even in the case when the neighboring wordline itself has errors.

3.2.3 Refresh Mechanisms

As we see in Figure 3.1, during the time period after a flash page is programmed, retention (Section 3.1.4) and read disturb (Section 3.1.5) can cause an increasing number of raw bit errors to accumulate over time. This is particularly problematic for a page that is not updated frequently. Due to the limited error correction capability, the accumulation of these errors can potentially lead to data loss for a page with a *high retention age* (i.e., a page that has not been programmed for a long time). To avoid data loss, *refresh mechanisms* have been proposed, where the stored data is periodically read, corrected, and reprogrammed, in order to eliminate the retention and read disturb errors that have accumulated prior to this periodic read/correction/reprogramming (i.e., refresh). The concept of refresh in flash memory is thus conceptually similar to the refresh mechanisms found in DRAM [39, 120, 186, 187]. By performing refresh and limiting the number of retention and read disturb errors that can accumulate, the lifetime of the SSD increases significantly. In this section, we describe three types of refresh mechanisms used in modern SSDs: remapping-based refresh, in-place refresh, and read reclaim.

Remapping-Based Refresh. Flash cells must first be erased before they can be reprogrammed, due to the fact the programming a cell via ISPP can only increase the charge level of the cell but not reduce it (Section 2.2.4). The key idea of *remapping-based refresh* is to periodically read data from each valid flash block, correct any data errors, and *remap the data to a different physical location*, in order to prevent the data from accumulating too many retention errors [19, 22, 25, 222, 250]. During each refresh interval, a block with valid data that needs to be refreshed is selected. The valid data in the selected block is read out page by page and moved to the SSD controller. The ECC engine in the SSD controller corrects the errors in the read data, including retention errors that have accumulated since the last refresh. A new block is then selected from the free list (see Section 2.1.3), the error-free data is programmed to a page within the new block, and the logical address is remapped to point to the newly-programmed physical page. By reducing the accumulation of retention and read disturb errors, remapping-based refresh increases SSD lifetime by an average of 9x for a variety of disk workloads [22, 25].

Prior work proposes extensions to the basic remapping-based refresh approach. One work, *refresh SSDs*, proposes a refresh scheduling algorithm based on an earliest deadline first policy to guarantee that all data is refreshed in time [222]. The *quasi-nonvolatile SSD* proposes to use remapping-based refresh to choose between improving flash endurance and reducing the flash programming latency (by using larger ISPP step-pulses) [250]. In the quasi-nonvolatile SSD, refresh requests are deprioritized, scheduled at idle times, and can be interrupted after refreshing any page within a block, to minimize the delays that refresh can cause for the response time of pending workload requests to the SSD. A refresh operation can also be triggered proactively based on the data read latency observed for a page, which is indicative of how many errors the page has experienced [30]. Triggering refresh *proactively* based on the observed read latency (as opposed to doing so *periodically*) improves SSD latency and throughput [30]. Whenever the read latency for a page within a block exceeds a fixed threshold, the valid data in the block is refreshed, i.e., remapped to a new block [30].

In-Place Refresh. A major drawback of remapping-based refresh is that it performs *additional writes* to the NAND flash memory, accelerating wearout. To reduce the wearout overhead

of refresh, prior work proposes *in-place refresh* [19, 22, 25]. As data sits unmodified in the SSD, data retention errors dominate [21, 25, 312], leading to charge loss and causing the threshold voltage distribution to shift to the left, as we showed in Section 3.1.4. The key idea of in-place refresh is to incrementally replenish the lost charge of each page *at its current location*, i.e., in place, without the need for remapping.

Figure 3.11 shows a high-level overview of in-place refresh for a wordline. The SSD controller first reads all of the pages in the wordline (❶ in Figure 3.11). The controller invokes the ECC decoder to correct the errors within each page (❷), and sends the corrected data back to the flash chips (❸). In-place refresh then invokes a modified version of the ISPP mechanism (see Section 2.2.4), which we call *Verify-ISPP* (V-ISPP), to compensate for retention errors by restoring the charge that was lost. In V-ISPP, we first verify the voltage currently programmed in a flash cell (❹). If the current voltage of the cell is *lower* than the target threshold voltage of the state that the cell should be in, V-ISPP pulses the programming voltage in steps, gradually injecting charge into the cell until the cell returns to the target threshold voltage (❺). If the current voltage of the cell is *higher* than the target threshold voltage, V-ISPP inhibits the programming pulses to the cell.

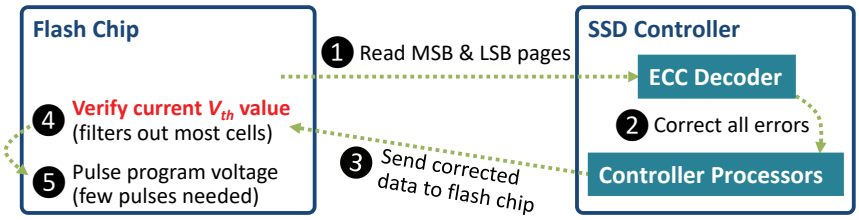


Figure 3.11: Overview of in-place refresh mechanism for MLC NAND flash memory. Reproduced from [32].

When the controller invokes in-place refresh, it is unable to use shadow program sequencing (Section 3.2.1), as all of the pages within the wordline have already been programmed. However, unlike traditional ISPP, V-ISPP does not introduce a high amount of cell-to-cell program interference (Section 3.1.3) for two reasons. First, V-ISPP programs *only* those cells that have retention errors, which typically account for less than 1% of the total number of cells in a wordline selected for refresh [22]. Second, for the small number of cells that are selected to be refreshed, their threshold voltage is usually only slightly lower than the target threshold voltage, which means that only a few programming pulses need to be applied. As cell-to-cell interference is linearly correlated with the threshold voltage change to immediately-adjacent cells [24, 26], the small voltage change on these in-place refreshed cells leads to only a small interference effect.

One issue with in-place refresh is that it is unable to correct retention errors for cells in lower-voltage states. Retention errors cause the threshold voltage of a cell in a lower-voltage state to *increase*, but V-ISPP *cannot* decrease the threshold voltage of a cell. To achieve a balance between the wearout overhead due to remapping-based refresh and errors that increase the threshold voltage due to in-place refresh, prior work proposes *hybrid in-place refresh* [19, 22, 25]. The key idea is to use in-place refresh when the number of program errors (caused due to reprogramming) is within the correction capability of ECC, but to use remapping-based refresh if the number of program errors is too large to tolerate. To accomplish this, the controller tracks the number of

right-shift errors (i.e., errors that move a cell to a higher-voltage state) [22, 25]. If the number of right-shift errors remains under a certain threshold, the controller performs in-place refresh; otherwise, it performs remapping-based refresh. Such a hybrid in-place refresh mechanism increases SSD lifetime by an average of 31x for a variety of disk workloads [22, 25].

Read Reclaim to Reduce Read Disturb Errors. We can also mitigate read disturb errors using an idea similar to remapping-based refresh, known as *read reclaim*. The key idea of read reclaim is to remap the data in a block to a new flash block, if the block has experienced a high number of reads [95, 96, 148]. To bound the number of read disturb errors, some flash vendors specify a maximum number of tolerable reads for a flash block, at which point read reclaim rewrites the data to a new block (just as is done for remapping- based refresh).

Adaptive Refresh and Read Reclaim Mechanisms. For the refresh and read reclaim mechanisms discussed above, the SSD controller can (1) invoke the mechanisms at fixed regular intervals; or (2) *adapt* the rate at which it invokes the mechanisms, based on various conditions that impact the rate at which data retention and read disturb errors occur. By adapting the mechanisms based on the current conditions of the SSD, the controller can reduce the overhead of performing refresh or read reclaim. The controller can adaptively adjust the rate that the mechanisms are invoked based on (1) the wearout (i.e., the current P/E cycle count) of the NAND flash memory [22, 25]; or (2) the temperature of the SSD [21, 27].

As we discuss in Section 3.1.4, for data with a given retention age, the number of retention errors grows as the P/E cycle count increases. Exploiting this P/E cycle dependent behavior of retention time, the SSD controller can perform refresh less frequently (e.g., once every year) when the P/E cycle count is low, and more frequently (e.g., once every week) when the P/E cycle count is high, as proposed and described in prior works from our research group [22, 25]. Similarly, for data with a given read disturb count, as the P/E cycle count increases, the number of read disturb errors increases as well [35]. As a result, the SSD controller can perform read reclaim less frequently (i.e., it increases the maximum number of tolerable reads per block before read reclaim is triggered) when the P/E cycle count is low, and more frequently when the P/E cycle count is high.

Prior works demonstrate that for a given retention time, the number of data retention errors increases as the NAND flash memory's operating temperature increases [21, 27]. To compensate for the increased number of retention errors at high temperature, a state-of-the-art SSD controller adapts the rate at which it triggers refresh. The SSD contains sensors that monitor the current environmental temperature every few milliseconds [213, 329]. The controller then uses the Arrhenius equation [6, 222, 344] to estimate the rate at which retention errors accumulate at the current temperature of the SSD. Based on the error rate estimate, the controller decides if it needs to increase the rate at which it triggers refresh to ensure that the data is not lost.

By employing adaptive refresh and/or read reclaim mechanisms, the SSD controller can successfully reduce the mechanism overheads while effectively mitigating the larger number of data retention errors that occur under various conditions.

3.2.4 Read-Retry

In earlier generations of NAND flash memory, the read reference voltage values were fixed at design time [23, 219]. However, several types of errors cause the threshold voltage distribution to shift, as shown in Figure 3.2. To compensate for threshold voltage distribution shifts, a mechanism called *read-retry* has been implemented in modern flash memories (typically those below 30 nm for planar flash [23, 82, 296, 349]).

The read-retry mechanism allows the read reference voltages to dynamically adjust to changes in distributions. During read-retry, the SSD controller first reads the data out of NAND flash memory with the default read reference voltage. It then sends the data for error correction. If ECC successfully corrects the errors in the data, the read operation succeeds. Otherwise, the SSD controller reads the memory again with a *different* read reference voltage. The controller repeats these steps until it either successfully reads the data using a certain set of read reference voltages or is unable to correctly read the data using all of the read reference voltages that are available to the mechanism.

While read-retry is widely implemented today, it can significantly increase the overall read operation latency due to the multiple read attempts it causes [27]. Mechanisms have been proposed to reduce the number of read-retry attempts while taking advantage of the effective capability of read-retry for reducing read errors, and read-retry has also been used to enable mitigation mechanisms for various other types of errors, as we describe in Section 3.2.5. As a result, read-retry is an essential mechanism in modern SSDs to mitigate read errors (i.e., errors that manifest themselves during a read operation).

3.2.5 Voltage Optimization

Many raw bit errors in NAND flash memory are affected by the various voltages used within the memory to enable reading of values. We give two examples. First, a suboptimal *read reference voltage* can lead to a large number of read errors (Section 3.1), especially after the threshold voltage distribution shifts. Second, as we saw in Section 3.1.5, the *pass-through voltage* can have a significant effect on the number of read disturb errors that occur. As a result, optimizing these voltages such that they minimize the total number of errors that are induced can greatly mitigate error counts. In this section, we discuss mechanisms that can discover and employ the optimal[2] read reference and pass-through voltages.

Optimizing Read Reference Voltages Using Disparity-Based Approximation and Sampling. As we discussed in Section 3.2.4, when the threshold voltage distribution shifts, it is important to move the read reference voltage to the point where the number of read errors is minimized. After the shift occurs and the threshold voltage distribution of each state widens, the distributions of different states may overlap with each other, causing many of the cells within the overlapping regions to be misread. The number of errors due to misread cells can be minimized by setting the read reference voltage to be exactly at the point where the distributions of two neighboring states intersect, which we call the *optimal read reference voltage*

[2]Or, more precisely, near-optimal, if the read-retry steps are too coarse grained to find the optimal voltage.

(V_{opt}) [24, 26, 27, 195, 252], illustrated in Figure 3.12. Once the optimal read reference voltage is applied, the raw bit error rate is minimized, improving the reliability of the device.

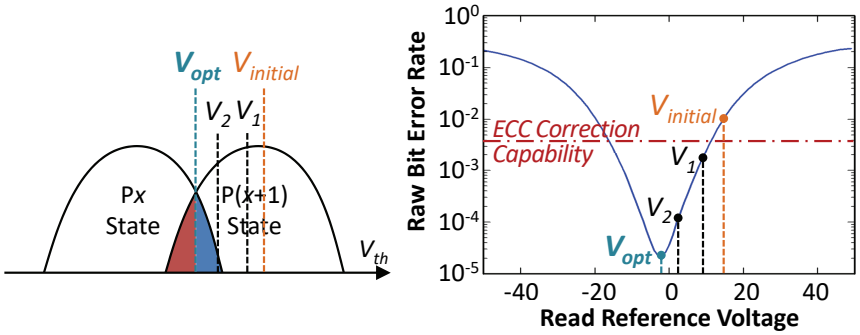


Figure 3.12: Finding the optimal read reference voltage after the threshold voltage distributions overlap (left), and raw bit error rate as a function of the selected read reference voltage (right). Reproduced from [32].

One approach to finding V_{opt} is to adaptively learn and apply the optimal read reference voltage for each flash block through sampling [27, 52, 65, 339]. The key idea is to periodically (1) use *disparity* information (i.e., the ratio of 1s to 0s in the data) to attempt to find a read reference voltage for which the error rate is lower than the ECC correction capability; and to (2) use *sampling* to efficiently tune the read reference voltage to its optimal value to reduce the read operation latency. Prior characterization of real NAND flash memory [27, 252] found that the value of V_{opt} does *not* shift greatly over a short period of time (e.g., a day), and that all pages within a block experience *similar* amounts of threshold voltage shifts, as they have the same amount of wearout and are programmed around the same time [27, 252]. Therefore, we can invoke the V_{opt} learning mechanism periodically (e.g., daily) to efficiently tune the *initial read reference voltage* (i.e., the first read reference voltage used when the controller invokes the read-retry mechanism, described in Section 3.2.4) for each flash block, ensuring that the initial voltage used by read-retry stays close to V_{opt} even as the threshold voltage distribution shifts.

The SSD controller searches for V_{opt} by counting the number of errors that need to be corrected by ECC during a read. However, there may be times where the initial read reference voltage ($V_{initial}$) is set to a value at which the number of errors during a read exceeds the ECC correction capability, such as the raw bit error rate for $V_{initial}$ in Figure 3.12 (right). When the ECC correction capability is exceeded, the SSD controller is unable to count how many errors exist in the raw data. The SSD controller uses *disparity-based read reference voltage approximation* [52, 65, 339] for each flash block to try to bring $V_{initial}$ to a region where the number of errors does not exceed the ECC correction capability. Disparity-based read reference voltage approximation takes advantage of data scrambling. Recall from Section 2.1.3 that to minimize data value dependencies for the error rate, the SSD controller scrambles the data written to the SSD to probabilistically ensure that an equal number of 0s and 1s exist in the flash memory cells. The key idea of disparity-based read reference voltage approximation is to find the read reference voltages that result in approximately 50% of the cells reading out bit value 0, and the other 50% of the cells reading out bit value 1. To achieve this, the SSD controller employs a binary search algorithm, which tracks the ratio of 0s to 1s for each read reference voltage it tries. The binary

search tests various read reference voltage values, using the ratios of previously tested voltages to narrow down the range where the read reference voltage can have an equal ratio of 0s to 1s. The binary search algorithm continues narrowing down the range until it finds a read reference voltage that satisfies the ratio.

The usage of the binary search algorithm depends on the type of NAND flash memory used within the SSD. For SLC NAND flash, the controller searches for only a single read reference voltage. For MLC NAND flash, there are three read reference voltages: the LSB is determined using V_b, and the MSB is determined using both V_a and V_c (see Section 2.2.3). Figure 3.13 illustrates the search procedure for MLC NAND flash. First, the controller uses binary search to find V_b, choosing a voltage that reads the LSB of 50% of the cells as data value 0 (step 1 in Figure 3.13). For the MSB, the controller uses the discovered V_b value to help search for V_a and V_c. Due to scrambling, cells should be equally distributed across each of the four voltage states. The controller uses binary search to set V_a such that 25% of the cells are in the ER state, by ensuring that half of the cells *to the left of* V_b are read with an MSB of 0 (step 2). Likewise, the controller uses binary search to set V_c such that 25% of the cells are in the P3 state, by ensuring that half of the cells *to the right of* V_b are read with an MSB of 0 (step 3). This procedure is extended in a similar way to approximate the voltages for TLC NAND flash.

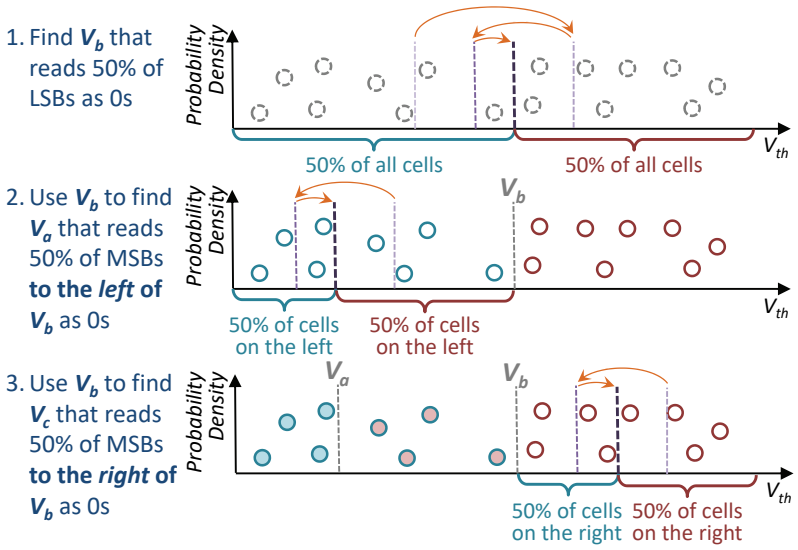


Figure 3.13: Disparity-based read reference voltage approximation to find $V_{initial}$ for MLC NAND flash memory. Each circle represents a cell, where a dashed border indicates that the LSB is undetermined, a solid border indicates that the LSB is known, a hollow circle indicates that the MSB is unknown, and a filled circle indicates that the MSB is known. Reproduced from [32].

If disparity-based approximation finds a value for $V_{initial}$ where the number of errors during a read can be counted by the SSD controller, the controller invokes *sampling-based adaptive* V_{opt} *discovery* [27] to minimize the error count, and thus reduce the read latency. Sampling-based adaptive V_{opt} discovery learns and records V_{opt} for the *last-programmed page* in each block. Prior work samples only the last-programmed page because it is the page with the lowest data

retention age in the flash block. As retention errors cause the higher-voltage states to shift to the left (i.e., to lower voltages), the last-programmed page usually provides an *upper bound* of V_{opt} for the entire block.

During sampling-based adaptive V_{opt} discovery, the SSD controller first reads the last-programmed page using $V_{initial}$, and attempts to correct the errors in the raw data read from the page. Next, it records the number of raw bit errors as the current lowest error count N_{ERR}, and sets the applied read reference voltage (V_{ref}) as $V_{initial}$. Since V_{opt} typically decreases over retention age, the controller first attempts to lower the read reference voltage for the last-programmed page, decreasing the voltage to $V_{ref} - \Delta V$ and reading the page. If the number of corrected errors in the new read is less than or equal to the old N_{ERR}, the controller updates N_{ERR} and V_{ref} with the new values. The controller continues to lower the read reference voltage until the number of corrected errors in the data is greater than the old N_{ERR} or the lowest possible read reference voltage is reached. Since the optimal threshold voltage might increase in rare cases, the controller also tests increasing the read reference voltage. It increases the voltage to $V_{ref} + \Delta V$ and reads the last-programmed page to see if N_{ERR} decreases. Again, it repeats increasing V_{ref} until the number of corrected errors in the data is greater than the old N_{ERR} or the highest possible read reference voltage is reached. The controller sets the initial read reference voltage of the block as the value of V_{ref} at the end of this process so that the next time an uncorrectable error occurs, read-retry starts at a $V_{initial}$ that is hopefully closer to the optimal read reference voltage (V_{opt}).

During the course of the day, as more retention errors (the dominant source of errors on already-programmed blocks) accumulate, the threshold voltage distribution shifts to the left (i.e., voltages decrease), and the initial read reference voltage (i.e., $V_{initial}$) is now an upper bound for the read-retry voltages. Therefore, whenever read-retry is invoked, the controller now needs to only decrease the read reference voltages (as opposed to traditional read-retry, which tries *both* lower and higher voltages [27]). Sampling-based adaptive V_{opt} discovery improves the *endurance* (i.e., the number of P/E cycles before the ECC correction capability is exceeded) of the NAND flash memory by 64% and reduces error correction latency by 10% [27], and is employed in some modern SSDs today.

Other Approaches to Optimizing Read Reference Voltages. One drawback of the sampling-based adaptive technique is that it requires time and storage overhead to find and record the per-block initial voltages. To avoid this, the SSD controller can employ an accurate *online threshold voltage distribution model* [19, 23], which can efficiently track and predict the shift in the distribution over time. The model represents the threshold voltage distribution of each state as a probability density function (PDF), and the controller can use the model to calculate the intersection of the different PDFs. The controller uses the PDF in place of the threshold voltage sampling, determining V_{opt} by calculating the intersection of the distribution of each state in the model. Chapter 5 demonstrates an example of this approach.

Other prior work examines adapting read reference voltages based on P/E cycle count, retention age, or read disturb. In one such work, the controller periodically learns read reference voltages by testing three read reference voltages on six pages per block, which the work demonstrates to be sufficiently accurate [252]. Similarly, error correction using LDPC soft decoding (see Section 3.3.2) requires reading the same page using multiple sets of read reference voltages to provide fine-grained information on the probability of each cell representing a bit value 0 or

a bit value 1. Another prior work optimizes the read reference voltages to increase the ECC correction capability without increasing the coding rate [326].

Optimizing Pass-Through Voltage to Reduce Read Disturb Errors. As we discussed in Section 3.1.5, the vulnerability of a cell to read disturb is directly correlated with the voltage difference ($V_{pass} - V_{th}$) through the cell oxide [35]. Traditionally, a single V_{pass} value is used *globally* for the entire flash memory, and the value of V_{pass} must be higher than *all* potential threshold voltages within the chip to ensure that unread cells along a bitline are turned on during a read operation (see Section 2.2.3). To reduce the impact of read disturb, we can tune V_{pass} to reduce the size of the voltage difference ($V_{pass} - V_{th}$). However, it is difficult to reduce V_{pass} *globally*, as any cell with a value of $V_{th} > V_{pass}$ introduces an error during a read operation (which we call a *pass-through error*).

In prior work, we propose a mechanism that can dynamically lower V_{pass} while ensuring that it can correct any new pass-through errors introduced. The key idea of the mechanism is to lower V_{pass} only for those blocks where ECC has enough leftover error correction capability (see Section 2.1.3) to correct the newly introduced pass-through errors. When the retention age of the data within a block is low, prior work finds that the raw bit error rate of the block is much lower than the rate for the block when the retention age is high, as the number of data retention and read disturb errors remains low at low retention age [35, 96]. As a result, a block with a low retention age has significant *unused* ECC correction capability, which we can use to correct the pass-through errors we introduce when we lower V_{pass}, as shown in Figure 3.14. Thus, when a block has a low retention age, the controller lowers V_{pass} aggressively, making it much less likely for read disturbs to induce an uncorrectable error. When a block has a high retention age, the controller also lowers V_{pass}, but does not reduce the voltage aggressively, since the limited ECC correction capability now needs to correct retention errors, and might not have enough unused correction capability to correct many new pass-through errors. By reducing V_{pass} aggressively when a block has a low retention age, we can extend the time before the ECC correction capability is exhausted, improving the flash lifetime.

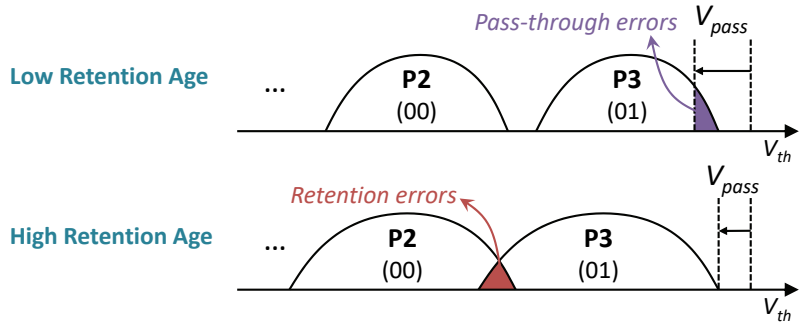


Figure 3.14: Dynamic pass-through voltage tuning at different retention ages. Reproduced from [32].

The previously-proposed read disturb mitigation mechanism [35] learns the minimum pass-through voltage for each block, such that all data within the block can be read correctly with ECC. The previously-proposed learning mechanism works online and is triggered periodically

(e.g., daily). The mechanism is implemented in the controller, and has two components. It first finds the size of the ECC margin M (i.e., the unused correction capability) that can be exploited to tolerate additional read errors for each block. Once it knows the available margin M, the previously-proposed mechanism calibrates V_{pass} on a per-block basis to find the lowest value of V_{pass} that introduces no more than M additional raw errors (i.e., there are no more than M cells where $V_{th} > V_{pass}$). The findings on MLC NAND flash memory show that the mechanism can improve flash endurance by an average of 21% for a variety of disk workloads [35].

Programming and Erase Voltages. Prior work also examines tuning the programming and erase voltages to extend flash endurance [122]. By decreasing the two voltages when the P/E cycle count is low, the accumulated wearout for each program or erase operation is reduced, which, in turn, increases the overall flash endurance. Decreasing the programming voltage, however, comes at the cost of increasing the time required to perform ISPP, which, in turn, increases the overall SSD write latency [122].

3.2.6 Hot Data Management

The data stored in different locations of an SSD can be accessed by the host at different rates. These pages exhibit high temporal write locality, and are called *write-hot* pages. Likewise, pages with a high amount of temporal read locality (i.e., pages that are accessed by a large fraction of the read operations) are called *read-hot* pages. A number of issues can arise when an SSD does not distinguish between write-hot pages and *write-cold* pages (i.e., pages with low temporal write locality), or between read-hot pages and *read-cold* pages (i.e., pages with low temporal read locality). For example, if write-hot pages and write-cold pages are stored within the same block, refresh mechanisms (which operate at the block level; see Section 3.2.3) *cannot* avoid refreshes to pages that were overwritten recently. This increases not only the energy consumption but also the write amplification due to remapping-based refresh [194]. Likewise, if read-hot and read-cold pages are stored within the same block, read-cold pages are unnecessarily exposed to a high number of read disturb errors [95, 96]. *Hot data management* refers to a set of mechanisms that can identify and exploit write-hot or read-hot pages in the SSD. The key idea common to such mechanisms is to apply special SSD management policies by placing hot pages and cold pages into *separate* flash blocks. Chapter 4 demonstrates an example technique using this idea.

Prior work [325] proposes to reuse the correctly functioning flash pages within bad blocks (see Section 2.1.3) to store write-cold data. This technique increases the total number of usable blocks available for overprovisioning, and extends flash lifetime by delaying the point at which each flash chip reaches the upper limit of bad blocks it can tolerate.

RedFTL identifies and replicates read-hot pages across multiple flash blocks, allowing the controller to evenly distribute read requests to these pages across the replicas [95]. Other works reduce the number of read reclaims (see Section 3.2.3) that need to be performed by mapping read-hot data to particular flash blocks and lowering the maximum possible threshold voltage for such blocks [29, 96]. By lowering the maximum possible threshold voltage for these blocks, the SSD controller can use a lower V_{pass} value (see Section 3.2.5) on the blocks without introducing any additional errors during a read operation. To lower the maximum threshold voltage in these blocks, the width of the voltage window for each voltage state is decreased, and each voltage

window shifts to the left [29, 96]. Another work applies stronger ECC encodings to *only* read-hot blocks based on the total read count of the block, in order to increase SSD endurance without significantly reducing the amount of overprovisioning [28] (see Section 2.1.4 for a discussion on the tradeoff between ECC strength and overprovisioning).

3.2.7 Adaptive Error Mitigation Mechanisms

Due to the many different factors that contribute to raw bit errors, error rates in NAND flash memory can be highly variable. Adaptive error mitigation mechanisms are capable of adapting error tolerance capability to the error rate. They provide stronger error tolerance capability when the error rate is higher, improving flash lifetime significantly. When the error rate is low, adaptive error mitigation techniques reduce error tolerance capability to lower the cost of the error mitigation techniques. In this section, we examine two types of adaptive techniques: (1) multi-rate ECC and (2) dynamic cell levels.

Multi-Rate ECC. Some works propose to employ multiple ECC algorithms in the SSD controller [31, 51, 99, 111, 336]. Recall from Section 2.1.4 that there is a tradeoff between ECC strength (i.e., the coding rate; see Section 2.1.3) and overprovisioning, as a codeword (which contains a data chunk *and* its corresponding ECC information) uses more bits when stronger ECC is employed. The key idea of multi-rate ECC is to employ a weaker codeword (i.e., one that uses fewer bits for ECC) when the SSD is relatively new and has a smaller number of raw bit errors, and to use the saved SSD space to provide additional overprovisioning, as shown in Figure 3.15.

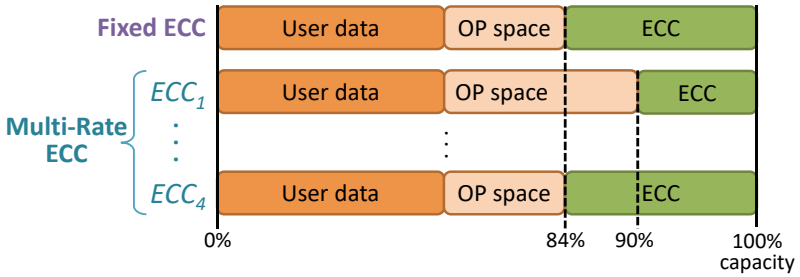


Figure 3.15: Comparison of space used for user data, overprovisioning, and ECC between a fixed ECC and a multi-rate ECC mechanism. Reproduced from [32].

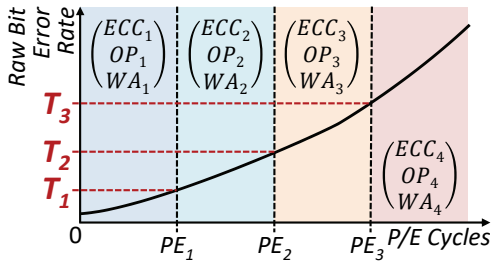


Figure 3.16: Illustration of how multi-rate ECC switches to different ECC codewords (i.e., ECC_i) as the RBER grows. OP_i is the overprovisioning factor used for engine ECC_i, and WA_i is the resulting write amplification value. Reproduced from [32].

Let us assume that the controller contains a configurable ECC engine that can support n different types of ECC codewords, which we call ECC_i. Figure 3.15 shows an example of multi-rate ECC that uses four ECC engines, where ECC_1 provides the weakest protection but has the smallest codeword, while ECC_4 provides the strongest protection with the largest codeword. We need to ensure that the NAND flash memory has enough space to fit the largest codewords, e.g., those for ECC_4 in Figure 3.15. Initially, when the raw bit error rate (RBER) is low, the controller employs ECC_1, as shown in Figure 3.16. The smaller codeword size for ECC_1 provides additional space for overprovisioning, as shown in Figure 3.15, and thus reduces the effects of write amplification. Multi-rate ECC works on an interval-by-interval basis. Every interval (in this case, a predefined number of P/E cycles), the controller measures the RBER. When the RBER exceeds the threshold set for transitioning from a weaker ECC to a stronger ECC, the controller switches to the stronger ECC. For example, when the SSD exceeds the first RBER threshold for switching (T_1 in Figure 3.16), the controller starts switching from ECC_1 to ECC_2. When switching between ECC engines, the controller uses the ECC_1 engine to decode data the next time the data is read out, and stores a new codeword using the ECC_2 engine. This process is repeated during the lifetime of flash memory for each stronger engine ECC_i, where each engine has a corresponding threshold that triggers switching [31, 51, 99], as shown in Figure 3.16.

Multi-rate ECC allows the same maximum P/E cycle count for each block as if ECC_n was used throughout the lifetime of the SSD, but reduces write amplification and improves performance during the periods where the lower strength engines are employed, by providing additional overprovisioning (see Section 2.1.4) during those times. As the lower-strength engines use smaller codewords (e.g., ECC_1 versus ECC_4 in Figure 3.15), the resulting free space can instead be employed to further increase the amount of overprovisioning within the NAND flash memory, which in turn increases the total lifetime of the SSD. We compute the lifetime improvement by modifying Equation 2.4 (Section 2.1.4) to account for each engine, as follows:

$$\text{Lifetime} = \sum_{i=1}^{n} \frac{\text{PEC}_i \times (1 + \text{OP}_i)}{365 \times \text{DWPD} \times \text{WA}_i \times R_{compress}} \tag{3.4}$$

In Equation 3.4, WA_i and OP_i are the write amplification and overprovisioning factor for ECC_i, and PEC_i is the number of P/E cycles that ECC_i is used for. Manufacturers can set parameters to maximize SSD lifetime in Equation 3.4, by optimizing the values of WA_i and OP_i.

Figure 3.17 shows the lifetime improvements for a four-engine multi-rate ECC, with the coding rates for the four ECC engines (ECC_1–ECC_4) set to 0.90, 0.88, 0.86, and 0.84 (recall that a *lower* coding rate provides stronger protection; see Section 2.1.4), over a fixed ECC engine that employs a coding rate of 0.84. We see that the lifetime improvements of using multi-rate ECC are: (1) significant, with a 31.2% increase if the baseline NAND flash memory has 15% overprovisioning; and (2) greater when the SSD initially has a smaller amount of overprovisioning.

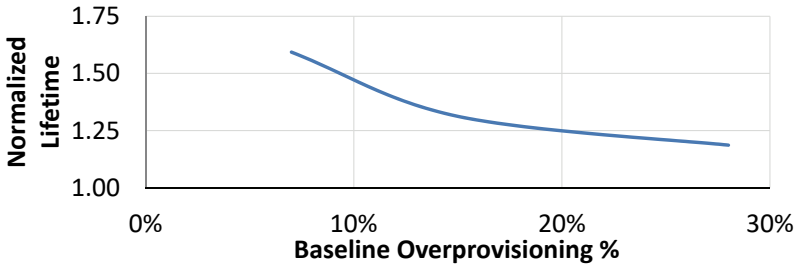


Figure 3.17: Lifetime improvements of using multi-rate ECC over using a fixed ECC coding rate. Reproduced from [32].

Dynamic Cell Levels. A major reason that errors occur in NAND flash memory is because the threshold voltage distribution of each state overlaps more with those of neighboring states as the distributions widen over time. Distribution overlaps are a greater problem when more states are encoded within the same voltage range. Hence, TLC flash has a much lower endurance than MLC, and MLC has a much lower endurance than SLC (assuming the same process technology node). If we can increase the margins between the states' threshold voltage distributions, the amount of overlap can be reduced significantly, which in turn reduces the number of errors.

Prior work proposes to increase margins by *dynamically* reducing the number of bits stored within a cell, e.g., by going from three bits that encode eight states (TLC) to two bits that encode four states (equivalent to MLC), or to one bit that encodes two states (equivalent to SLC) [29, 332]. Recall that TLC uses the ER state and states P1–P7, which are spaced out approximately equally. When we *downgrade* a flash block (i.e., reduce the number of states its cells can represent) from eight states to four, the cells in the block now employ only the ER state and states P3, P5, and P7. As we can see from Figure 3.18, this provides large margins between states P3, P5, and P7, and provides an even larger margin between ER and P3. The SSD controller maintains a list of all of the blocks that have been downgraded. For each read operation, the SSD controller checks if the target block is in the downgraded block list, and uses this information to interpret the data that it reads out from the wordline of the block.

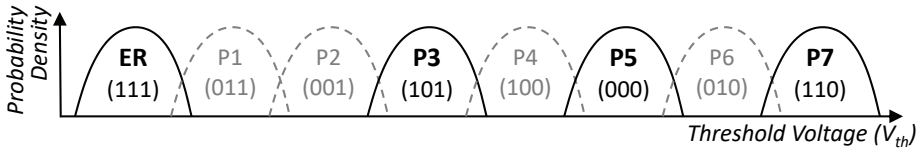


Figure 3.18: States used when a TLC cell (with 8 states) is downgraded to an MLC cell (with 4 states). Reproduced from [32].

A cell can be downgraded to reduce various types of errors (e.g., wearout, read disturb). To reduce wearout, a cell is downgraded when it has high wearout. To reduce read disturb, a cell can be downgraded if it stores *read-hot* data (i.e., the most frequently read data in the SSD). By using fewer states for a block that holds read-hot data, we can reduce the impact of read disturb because it becomes harder for the read disturb mechanism to affect the distributions enough for them to overlap. As an optimization, the SSD controller can employ various hot-cold data partitioning mechanisms (e.g., [28, 29, 95, 194]) to keep read-hot data in specially designated blocks [28, 29, 95, 96], allowing the controller to reduce the size of the downgraded block list and isolate the impact of read disturb from *read-cold* (i.e., infrequently read) data.

Another approach to dynamically increasing the distribution margins is to perform program and erase operations more slowly when the SSD write request throughput is low [29, 122]. Slower program/erase operations allow the final voltage of a cell to be programmed more precisely, and reduce the amount of oxide degradation that occurs during programming. As a result, the distribution of each state is initially much narrower, and subsequent widening of the distributions results in much lower overlap for a given P/E cycle count. This technique improves the SSD lifetime by an average of 61.2% for a variety of disk workloads [122]. Unfortunately, the slower program/erase operations come at the cost of higher SSD latency, and are thus not applied during periods of high write traffic. One way to mitigate the impact of the higher write latency is to perform slower program/erase operations only during garbage collection, which ensures that the higher latency occurs only when the SSD is idle [29]. As a result, read and write requests from the host do not experience any additional delays.

3.3 Error Correction and Data Recovery Techniques

Now that we have described a variety of error mitigation mechanisms that can target various types of error sources, we turn our attention to the error correction flow that is employed in modern SSDs as well as *data recovery techniques* that can be employed when the error correction flow fails to produce correct data. In this section, we briefly overview the major error correction steps an SSD performs when reading data. We first discuss two ECC encodings that are typically used by modern SSDs: Bose–Chaudhuri–Hocquenghem (BCH) codes [14, 107, 177, 297] and low-density parity-check (LDPC) codes [84, 85, 203, 204, 297] (Section 3.3.1). Next, we go through example error correction flows for an SSD that uses either BCH codes or LDPC codes (Section 3.3.2). Then, we compare the error correction strength (i.e., the number of errors that ECC can correct) when we employ BCH codes or LDPC codes in an SSD (Section 3.3.3). Finally, we discuss techniques that can rescue data from an SSD when the BCH/LDPC decoding fails to correct all errors (Section 3.3.4).

3.3.1 Error-Correcting Codes Used in SSDs

Modern SSDs typically employ one of two types of ECC. Bose–Chaudhuri–Hocquenghem (BCH) codes allow for the correction of multiple bit errors [14, 107, 177, 297], and are used to correct the errors observed during a *single* read from the NAND flash memory [177]. Low-density parity-check (LDPC) codes employ information accumulated over *multiple* read

operations to determine the likelihood of each cell containing a bit value 1 or a bit value 0 [84, 85, 203, 204, 297], providing stronger protection at the cost of greater decoding latency and storage overhead [326, 362]. Next, we describe the basics of BCH and LDPC codes.

Bose–Chaudhuri–Hocquenghem (BCH) Codes

BCH codes [14, 107, 177, 297] have been widely used in modern SSDs during the past decade due to their ability to detect and correct multi-bit errors while keeping the latency and hardware cost of encoding and decoding low [48, 177, 207, 215]. For SSDs, BCH codes are designed to be *systematic*, which means that the original data message is embedded *verbatim* within the codeword. Within an n-bit codeword (see Section 2.1.3), error-correcting codes use the first k bits of the codeword, called *data bits*, to hold the data message bits, and the remaining $(n-k)$ bits, called *check bits*, to hold error correction information that protects the data bits. BCH codes are designed to *guarantee* that they correct up to a certain number of raw bit errors (e.g., t error bits) within each codeword, which depends on the values chosen for n and k. A stronger error correction strength (i.e., a larger t) requires more redundant check bits (i.e., $(n-k)$) or a longer codeword length (i.e., n).

A BCH code [14, 107, 177, 297] is a linear block code that consists of check bits generated by an algorithm. The codeword generation algorithm ensures that the check bits are selected such that the check bits can be used during a parity check to detect and correct up to t bit errors in the codeword. A BCH code is defined by (1) a generator matrix G, which informs the generation algorithm of how to generate each check bit using the data bits; and (2) a parity check matrix H, which can be applied to the codeword to detect if any errors exist. In order for a BCH code to guarantee that it can correct t errors within each codeword, the minimum separation d (i.e., the Hamming distance) between valid codewords must be at least $d = 2t+1$ [297].

BCH Encoding. The codeword generation algorithm encodes a k-bit data message m into an n-bit BCH codeword c, by computing the dot product of m and the generator matrix G (i.e., $c = m \cdot G$). G is defined within a finite Galois field $GF(2^d) = \{0, \alpha^0, \alpha^1, \ldots, \alpha^{2^d-1}\}$, where α is a *primitive element* of the field and d is a positive integer [77]. An SSD manufacturer constructs G from a set of polynomials $g_1(x), g_2(x), \ldots g_{2t}(x)$, where $g_i(\alpha^i) = 0$. Each polynomial generates a *parity bit*, which is used during decoding to determine if any errors were introduced. The i-th row of G encodes the i-th polynomial $g_i(x)$. When decoding, the codeword c can be viewed as a polynomial $c(x)$. Since $c(x)$ is generated by $g_i(x)$ which has a root α^i, α^i should also be a root of $c(x)$. The parity check matrix H is constructed such that cH^t calculates $c(\alpha_i)$. Thus, the element in the i-th row and j-th column of H is $H_{ij} = \alpha^{(j-1)(i+1)}$. This allows the decoder to use H to quickly determine if any of the parity bits do not match, which indicates that there are errors within the codeword. BCH codes in SSDs are typically designed to be *systematic*, which guarantees that a verbatim copy of the data message is embedded within the codeword. To form a systematic BCH code, the generator matrix and the parity check matrix are transformed such that they contain the identity matrix.

BCH Decoding. When the SSD controller is servicing a read request, it must extract the data bits (i.e., the k-bit data message m) from the BCH codeword that is stored in the NAND flash memory chips. Once the controller retrieves the codeword, which we call r, from NAND flash

memory, it sends r to a BCH decoder. The decoder performs five steps, as illustrated in Figure 3.19, which correct the retrieved codeword r to obtain the originally-written codeword c, and then extract the data message m from c. In Step 1, the decoder uses *syndrome calculation* to detect if any errors exist within the retrieved codeword r. If no errors are detected, the decoder uses the retrieved codeword as the original codeword, c, and skips to Step 5. Otherwise, the decoder continues on to correct the errors and recover c. In Step 2, the decoder uses the syndromes from Step 1 to construct an *error location polynomial*, which encodes the locations of each detected bit error within r. In Step 3, the decoder extracts the specific location of each detected bit error from the error location polynomial. In Step 4, the decoder corrects each detected bit error in the retrieved codeword r to recover the original codeword c. In Step 5, the decoder extracts the data message from the original codeword c. We describe the algorithms most commonly used by BCH decoders in SSDs [56, 177, 191] for each step in detail below.

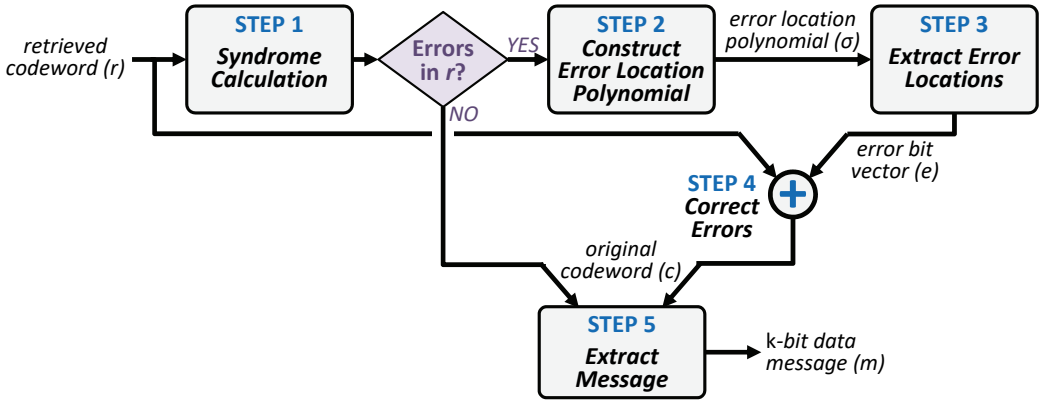


Figure 3.19: BCH decoding steps.

Step 1—Syndrome Calculation: To determine whether the retrieved codeword r contains any errors, the decoder computes the *syndrome vector*, S, which indicates how many of the parity check polynomials no longer match with the parity bits originally computed during encoding. The i-th syndrome, S_i, is set to one if parity bit i does *not* match its corresponding polynomial, and to zero otherwise. To calculate S, the decoder calculates the dot product of r and the parity check matrix H (i.e., $S = r \cdot H$). If every syndrome in S is set to 0, the decoder does not detect any errors within the codeword, and skips to Step 5. Otherwise, the decoder proceeds to Step 2.

Step 2—Constructing the Error Location Polynomial: A state-of-the-art BCH decoder uses the Berlekamp–Massey algorithm [11, 48, 209, 280] to construct an error location polynomial, $\sigma(x)$, whose roots encode the error locations of the codeword:

$$\sigma(x) = 1 + \sigma_1 \cdot x + \sigma_2 \cdot x^2 + \ldots + \sigma_b \cdot x^b \tag{3.5}$$

In Equation 3.5, b is the number of raw bit errors in the codeword.

The polynomial is constructed using an iterative process. Since b is not known initially, the algorithm initially assumes that $b = 0$ (i.e., $\sigma(x) = 1$). Then, it updates $\sigma(x)$ by adding a *correction term* to the equation in each iteration, until $\sigma(x)$ successfully encodes all of the errors that were detected during syndrome calculation. In each iteration, a new correction term

is calculated using both the syndromes from Step 1 and the $\sigma(x)$ equations from prior iterations of the algorithm, as long as these prior values of $\sigma(x)$ satisfy certain conditions. This algorithm successfully finds $\sigma(x)$ after $n = (t+b)/2$ iterations, where t is the maximum number of bit errors correctable by the BCH code [77].

Note that (1) the highest order of the polynomial, b, is directly correlated with the number of errors in the codeword; (2) the number of iterations, n, is also proportional to the number of errors; (3) each iteration is compute-intensive, as it involves several multiply and add operations; and (4) this algorithm cannot be parallelized across iterations, as the computation in each iteration is dependent on the previous ones.

Step 3—Extracting Bit Error Locations from the Error Polynomial: A state-of-the-art decoder applies the Chien search [53, 297] on the error location polynomial to find the location of *all* raw bit errors that have been detected during Step 1 in the retrieved codeword r. Each bit error location is encoded with a known function f [280]. The error polynomial from Step 2 is constructed such that if the i-th bit of the codeword has an error, the error location polynomial $\sigma(f(i)) = 0$; otherwise, if the i-th bit does *not* have an error, $\sigma(f(i)) \neq 0$. The Chien search simply uses trial-and-error (i.e., tests if $\sigma(f(i))$ is zero), testing each bit in the codeword starting at bit 0. As the decoder needs to correct only the first k bits of the codeword that contain the data message m, the Chien search needs to evaluate only k different values of $\sigma(f(i))$. The algorithm builds a bit vector e, which is the same length as the retrieved codeword r, where the i-th bit of e is set to one if bit i of r contains a bit error, and is set to zero if bit i of r does *not* contain an error, or if $i \geq k$ (since there is no need to correct the parity bits).

Note that (1) the calculation of $\sigma(f(i))$ is compute-intensive, but can be parallelized because the calculation of each bit i is independent of the other bits, and (2) the complexity of Step 3 is linearly correlated with the number of detected errors in the codeword.

Step 4—Correcting the Bit Errors: The decoder corrects each detected bit error location by flipping the bit at that location in the retrieved codeword r. This simply involves XORing r with the error vector e created in Step 3. After the errors are corrected, the decoder now has the estimated value of the originally-written codeword c (i.e., $c = r \oplus e$). The decoded version of c is only an *estimate* of the original codeword, since if r contains more bit errors than the maximum number of errors (t) that the BCH can correct, there may be some uncorrectable errors that were *not* detected during syndrome calculation (Step 1). In such cases, the decoder cannot *guarantee* that it has determined the actual original codeword. In a modern SSD, the bit error rate of a codeword *after* BCH correction is expected to be less than 10^{-15} [121].

Step 5—Extracting the Message from the Codeword: As we discuss above, during BCH codeword encoding, the generator matrix G contains the identity matrix, to ensure that the k-bit message m is embedded verbatim into the codeword c. Therefore, the decoder recovers m by simply truncating the last $(n-k)$ bits from the n-bit codeword c.

BCH Decoder Latency Analysis. We can model the latency of the state-of-the-art BCH decoder (T^{dec}_{BCH}) that we described above as:

$$T^{dec}_{BCH} = T_{Syndrome} + N \cdot T_{Berlekamp} + \frac{k}{p} \cdot T_{Chien} \tag{3.6}$$

In Equation 3.6, $T_{Syndrome}$ is the latency for calculating the syndrome, which is determined by

the size of the parity check matrix H; $T_{Berlekamp}$ is the latency of one iteration of the Berlekamp–Massey algorithm; N is the total number of iterations that the Berlekamp–Massey algorithm performs; T_{Chien} is the latency for deciding whether or not a single bit location contains an error, using the Chien search; k is the length of the data message m; and p is the number of bits that are processed in parallel in Step 3. In this equation, $T_{Syndrome}$, $T_{Berlekamp}$, k, and p are constants for a BCH decoder implementation, while N and T_{Chien} are proportional to the raw bit error count of the codeword. Note that Steps 4 and 5 can typically be implemented such that they take less than one clock cycle in modern hardware, and thus their latencies are *not* included in Equation 3.6.

Low-Density Parity-Check (LDPC) Codes

LDPC codes [84, 85, 203, 204, 297] are now used widely in modern SSDs, as LDPC codes provide a stronger error correction capability than BCH codes, albeit at a greater storage cost [326, 362]. LDPC codes are one type of *capacity-approaching codes*, which are error-correcting codes that come close to the *Shannon limit*, i.e., the maximum number of data message bits (k_{max}) that can be delivered without errors for a certain codeword size (n) under a given error rate [292, 293]. Unlike BCH codes, LDPC codes cannot *guarantee* that they will correct a minimum number of raw bit errors. Instead, a good LDPC code guarantees that the *failure rate* (i.e., the fraction of all reads where the LDPC code cannot successfully correct the data) is less than a target rate for a given number of bit errors. Like BCH codes, LDPC codes for SSDs are designed to be *systematic*, i.e., to contain the data message verbatim within the codeword.

An LDPC code [84, 85, 203, 204, 297] is a linear code that, like a BCH code, consists of check bits generated by an algorithm. For an LDPC code, these check bits are used to form a bipartite graph, where one side of the graph contains nodes that represent each bit in the codeword, and the other side of the graph contains nodes that represent the parity check equations used to generate each parity bit. When a codeword containing errors is retrieved from memory, an LDPC decoder applies *belief propagation* [265] to iteratively identify the bits within the codeword that are *most likely* to contain a bit error.

An LDPC code is defined using a binary parity check matrix H, where H is very sparse (i.e., there are few ones in the matrix). Figure 3.20a shows an example H matrix for a seven-bit codeword c (see Section 2.1.3). For an n-bit codeword that encodes a k-bit data message, H is sized to be an $(n-k) \times n$ matrix. Within the matrix, each *row* represents a *parity check equation*, while each *column* represents one of the seven bits in the codeword. As our example matrix has three rows, this means that our error correction uses three parity check equations (denoted as f). A bit value 1 in row i, column j indicates that parity check equation f_i contains bit c_j. Each parity check equation XORs all of the codeword bits in the equation to see whether the output is zero. For example, parity check equation f_1 from the H matrix in Figure 3.20a is:

$$f_1 = c_1 \oplus c_2 \oplus c_4 \oplus c_5 = 0 \tag{3.7}$$

This means that c is a valid codeword only if $H \cdot c^T = 0$, where c^T is the transpose matrix of the codeword c.

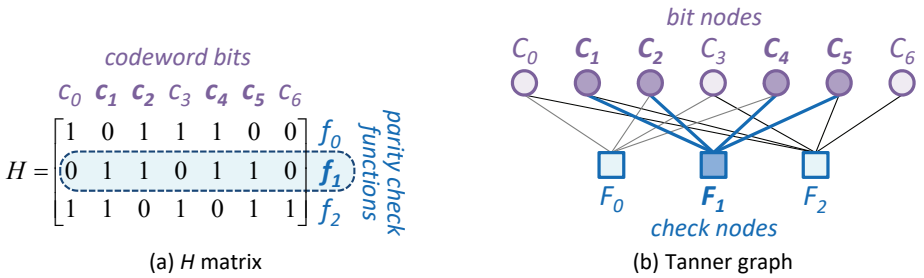


Figure 3.20: Example LDPC code for a seven-bit codeword with a four-bit data message (stored in bits c_0, c_1, c_2, and c_3) and three parity check equations (i.e., $n = 7$, $k = 4$), represented as (a) an H matrix and (b) a Tanner graph.

In order to perform belief propagation, H can be represented using a *Tanner graph* [313]. A Tanner graph is a bipartite graph that contains *check nodes*, which represent the parity check equations, and *bit nodes*, which represent the bits in the codeword. An edge connects a check node F_i to a bit node C_j only if parity check equation f_i contains bit c_j. Figure 3.20b shows the Tanner graph that corresponds to the H matrix in Figure 3.20a. For example, since parity check equation f_1 uses codeword bits c_1, c_2, c_4, and c_5, the F_1 check node in Figure 3.20b is connected to bit nodes C_1, C_2, C_4, and C_5.

LDPC Encoding. As was the case with BCH, the LDPC codeword generation algorithm encodes a k-bit data message m into an n-bit LDPC codeword c by computing the dot product of m and a generator matrix G (i.e., $c = m \cdot G$). For an LDPC code, the generator matrix is designed to (1) preserve m verbatim within the codeword, and (2) generate the parity bits for each parity check equation in H. Thus, G is defined using the parity check matrix H. With linear algebra based transformations, H can be expressed in the form $H = [A, I_{(n-k)}]$, where H is composed of A, an $(n-k) \times k$ binary matrix, and $I_{(n-k)}$, an $(n-k) \times (n-k)$ identity matrix [129]. The generator matrix G can then be created using the composition $G = [I_k, A^T]$, where A^T is the transpose matrix of A.

LDPC Decoding. When the SSD controller is servicing a read request, it must extract the k-bit data message from the LDPC codeword r that is stored in NAND flash memory. In an SSD, an LDPC decoder performs multiple *levels* of decoding [77, 323, 362], which correct the retrieved codeword r to obtain the originally-written codeword c and extract the data message m from c. Initially, the decoder performs a single level of *hard decoding*, where it uses the information from a single read operation on the codeword to attempt to correct the codeword bit errors. If the decoder cannot correct all errors using hard decoding, it then initiates the first level of *soft decoding*, where a *second* read operation is performed on the *same* codeword using a *different* set of read reference voltages. The second read provides *additional* information on the *probability* that each bit in the codeword is a zero or a one. An LDPC decoder typically uses multiple levels of soft decoding, where each new level performs an additional read operation to calculate a more accurate probability for each bit value. We discuss multi-level soft decoding in detail in Section 3.3.2.

For each level, the decoder performs five steps, as illustrated in Figure 3.21. At each level,

the decoder uses two pieces of information to determine which bits are *most likely* to contain errors: (1) the *probability* that each bit in r is a zero or a one, and (2) the parity check equations. In Step 1 (Figure 3.21), the decoder computes an initial *log likelihood ratio* (LLR) for each bit of the stored codeword. We refer to the initial codeword LLR values as L, where L_j is the LLR value for bit j of the codeword. L_j expresses the likelihood (i.e., *confidence*) that bit j *should be* a zero or a one, based on the current threshold voltage of the NAND flash cell where bit j is stored. The decoder uses L as the initial *LLR message* generated using the bit nodes. An LLR message consists of the LLR values for each bit, which are updated by and communicated between the check nodes and bit nodes during each step of belief propagation.[3] In Steps 2 through 4, the belief propagation algorithm [265] iteratively updates the LLR message, using the Tanner graph to identify those bits that are most likely to be incorrect (i.e., the codeword bits whose (1) bit nodes are connected to the largest number of check nodes that currently contain a parity error, and (2) LLR values indicate low confidence). Several decoding algorithms exist to perform belief propagation for LDPC codes. The most commonly-used belief propagation algorithm is the *min-sum algorithm* [49, 80], a simplified version of the original sum-product algorithm for LDPC [84, 85] with near-equivalent error correction capability [5]. During each iteration of the min-sum algorithm, the decoder identifies a set of codeword bits that likely contain errors and thus need to be flipped. The decoder accomplishes this by (1) having each check node use its parity check information to determine how much the LLR value of each bit should be updated by, using the most recent LLR messages from the bit nodes; (2) having each bit node gather the LLR updates from each bit to generate a new LLR value for the bit, using the most recent LLR messages from the check nodes; and (3) using the parity check equations to see if the values predicted by the new LLR message for each node are correct. The min-sum algorithm terminates under one of two conditions: (1) the predicted bit values after the most recent iteration are all correct, which means that the decoder now has an estimate of the original codeword c, and can advance to Step 5; or (2) the algorithm exceeds a predetermined number of iterations, at which point the decoder moves onto the next decoding level, or returns a decoding failure if the maximum number of decoding levels have been performed. In Step 5, once the errors are corrected, and the decoder has the original codeword c, the decoder extracts the k-bit data message m from the codeword. We describe the steps used by a state-of-the-art decoder in detail below, which uses an optimized version of the min-sum algorithm that can be implemented efficiently in hardware [92, 93].

[3]Note that an LLR message is *not* the same as the k-bit data message. The *data message* refers to the actual data stored within the SSD, which, when read, is modeled in information theory as a message that is transmitted across a noisy communication channel. In contrast, an *LLR message* refers to the updated LLR values for each bit of the codeword that are exchanged between the check nodes and the bit nodes during belief propagation. Thus, there is no relationship between a data message and an LLR message.

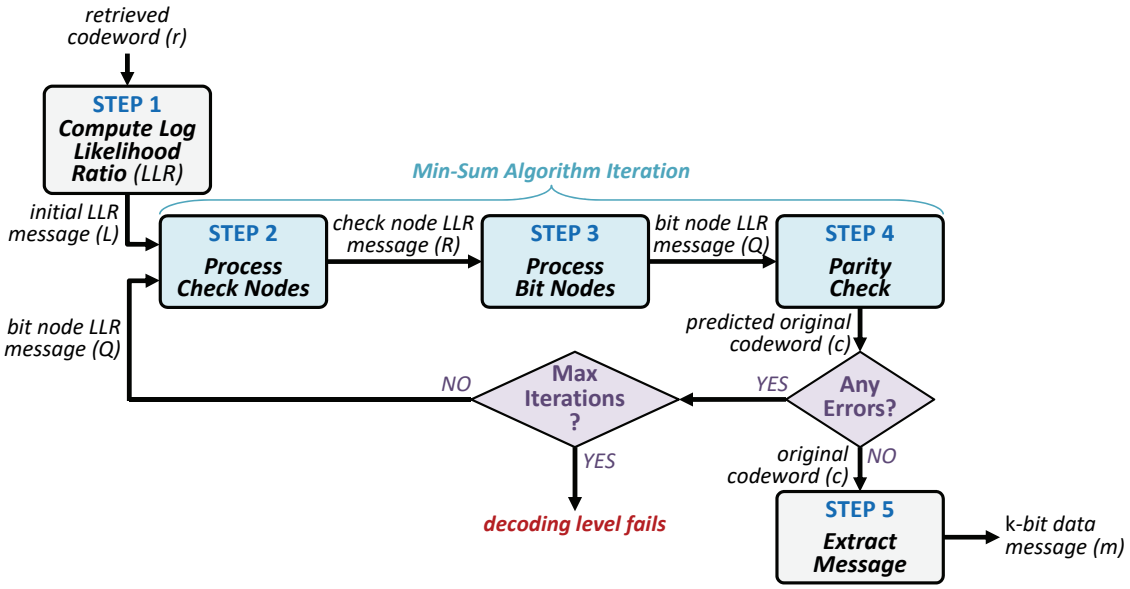


Figure 3.21: LDPC decoding steps for a single level of hard or soft decoding.

Step 1—Computing the Log Likelihood Ratio (LLR): The LDPC decoder uses the *probability* (i.e., *likelihood*) that a bit is a zero or a one to identify errors, instead of using the bit values directly. The *log likelihood ratio* (LLR) is the probability that a certain bit is zero, i.e., $P(x=0|V_{th})$, over the probability that the bit is one, i.e., $P(x=1|V_{th})$, given a certain threshold voltage range (V_{th}) bounded by two threshold voltage values (i.e., the maximum and the minimum voltage of the threshold voltage range) [326, 362]:

$$\text{LLR} = \log \frac{P(x=0|V_{th})}{P(x=1|V_{th})} \tag{3.8}$$

The sign of the LLR value indicates whether the bit is likely to be a zero (when the LLR value is positive) or a one (when the LLR value is negative). A *larger* magnitude (i.e., absolute value) of the LLR value indicates a *greater* confidence that a bit should be zero or one, while an LLR value closer to zero indicates low confidence. The bits whose LLR values have the smallest magnitudes are the ones that are most likely to contain errors.

There are several alternatives for how to compute the LLR values. A common approach for LLR computation is to treat a flash cell as a communication channel, where the channel takes an input program signal (i.e., the target threshold voltage for the cell) and outputs an observed signal (i.e., the current threshold voltage of the cell) [23]. The observed signal differs from the input signal due to the various types of NAND flash memory errors. The communication channel model allows us to break down the threshold voltage of a cell into two components: (1) the expected signal; and (2) the additive signal noise due to errors. By enabling the modeling of these two components separately, the communication channel model allows us to estimate the current threshold voltage distribution of each state [23]. The threshold voltage distributions can be used to predict how likely a cell within a certain voltage region is to belong to a particular voltage state.

One popular variant of the communication channel model assumes that the threshold voltage distribution of each state can be modeled as a Gaussian distribution [23]. If we use the mean observed threshold voltage of each state (denoted as μ) to represent the signal, we find that the P/E cycling noise (i.e., the shift in the distribution of threshold voltages due to the accumulation of charge from repeated programming operations; see Section 3.1.1) can be modeled as *additive white Gaussian noise* (AWGN) [23], which is represented by the standard deviation of the distribution (denoted as σ). The closed-form AWGN-based model can be used to determine the LLR value for a cell with threshold voltage y, as follows:

$$\mathrm{LLR}(y) = \frac{\mu_1^2 - \mu_0^2}{2\sigma^2} + \frac{y(\mu_0 - \mu_1)}{\sigma^2} \tag{3.9}$$

where μ_0 and μ_1 are the mean threshold voltages for the distributions of the threshold voltage states for bit value 0 and bit value 1, respectively, and σ is the standard deviation of both distributions (assuming that the standard deviation of each threshold voltage state distribution is equal). Since the SSD controller uses threshold voltage ranges to categorize a flash cell, we can substitute μ_{R_j}, the mean threshold voltage of the threshold voltage range R_j, in place of y in Equation 3.9.

The AWGN-based LLR model in Equation 3.9 provides only an estimate of the LLR, because (1) the actual threshold voltage distributions observed in NAND flash memory are *not* perfectly Gaussian in nature [23, 195]; (2) the controller uses the mean voltage of the threshold voltage range to *approximate* the actual threshold voltage of a cell; and (3) the standard deviations of each threshold voltage state distribution are *not* perfectly equal. A number of methods have been proposed to improve upon the AWGN-based LLR estimate by: (1) using nonlinear transformations to convert the AWGN-based LLR into a more accurate LLR value [341]; (2) scaling and rounding the AWGN-based LLR to compensate for the estimation error [342]; (3) initially using the AWGN-based LLR to read the data, and, if the read fails, using the ECC information from the failed read attempt to optimize the LLR and to perform the read again with the optimized LLR [66]; and (4) using online and offline training to empirically determine the LLR values under a wide range of conditions (e.g., P/E cycle count, retention time, read disturb count) [340]. The SSD controller can either compute the LLR values at runtime, or statically store precomputed LLR values in a table.

Once the decoder calculates the LLR values for each bit of the codeword, which we call the initial LLR message L, the decoder starts the first iteration of the min-sum algorithm (Steps 2–4 below).

Step 2—Check Node Processing: In every iteration of the min-sum algorithm, each check node i (see Figure 3.20) generates a revised check node LLR message R_{ij} to send to each bit node j (see Figure 3.20) that is connected to check node i. The decoder computes R_{ij} as:

$$R_{ij} = \delta_{ij}\kappa_{ij} \tag{3.10}$$

where δ_{ij} is the sign of the LLR message, and κ_{ij} is the magnitude of the LLR message. The decoder determines the values of both δ_{ij} and κ_{ij} using the bit node LLR message Q'_{ji}. At a high level, each check node collects LLR values sent from each bit node (Q'_{ji}), and then determines how much each bit's LLR value should be adjusted using the parity information available at the

check node. These LLR value updates are then bundled together into the LLR message R_{ij}. During the first iteration of the min-sum algorithm, the decoder sets $Q'_{ji} = L_j$, the initial LLR value from Step 1. In subsequent iterations, the decoder uses the value of Q'_{ji} that was generated in Step 3 of the *previous* iteration. The decoder calculates δ_{ij}, the sign of the check node LLR message, as:

$$\delta_{ij} = \prod_{J} \text{sgn}(Q'_{Ji}) \tag{3.11}$$

where J represents all bit nodes connected to check node i *except* for bit node j. The sign of a bit node indicates whether the value of a bit is predicted to be a zero (if the sign is positive) or a one (if the sign is negative). The decoder calculates κ_{ij}, the magnitude of the check node LLR message, as:

$$\kappa_{ij} = \min_{J} |Q'_{Ji}| \tag{3.12}$$

In essence, the smaller the magnitude of Q'_{ji} is, the more uncertain we are about whether the bit should be a zero or a one. At each check node, the decoder updates the LLR value of each bit node j, adjusting the LLR by the smallest value of Q' for any of the other bits connected to the check node (i.e., the LLR value of the most uncertain bit aside from bit j).

Step 3—Bit Node Processing: Once each check node generates the LLR messages for each bit node, we combine the LLR messages received by each bit node to update the LLR value of the bit. The decoder first generates the LLR messages to be used by the check nodes in the next iteration of the min-sum algorithm. The decoder calculates the bit node LLR message Q_{ji} to send from bit node j to check node i as follows:

$$Q_{ji} = L_j + \sum_{I} R_{Ij} \tag{3.13}$$

where I represents all check nodes connected to bit node j *except* for check node i, and L_j is the original LLR value for bit j generated in Step 1. In essence, for each check node, the bit node LLR message combines the LLR messages from the *other check nodes* to ensure that all of the LLR value updates are propagated globally across all of the check nodes.

Step 4—Parity Check: After the bit node processing is complete, the decoder uses the revised LLR information to predict the value of each bit. For bit node j, the predicted bit value P_j is calculated as:

$$P_j = L_j + \sum_{i} R_{ij} \tag{3.14}$$

where i represents *all* check nodes connected to bit node j, *including* check node i, and L_j is the original LLR value for bit j generated in Step 1. If P_j is positive, bit j of the original codeword c is predicted to be a zero; otherwise, bit j is predicted to be a one. Once the predicted values have been computed for all bits of c, the H matrix is used to check the parity, by computing $H \cdot c^T$. If $H \cdot c^T = 0$, then the predicted bit values are correct, the min-sum algorithm terminates, and the decoder goes to Step 5. Otherwise, at least one bit is still incorrect, and the decoder goes back to Step 2 to perform the next iteration of the min-sum algorithm. In the next iteration, the min-sum algorithm uses the updated LLR values from the current iteration to identify the next set of bits that are most likely incorrect and need to be flipped.

The current decoding level fails to correct the data when the decoder cannot determine the correct codeword bit values after a predetermined number of min-sum algorithm iterations. If the decoder has more soft decoding levels left to perform, it advances to the next soft decoding level. For the new level, the SSD controller performs an additional read operation using a different set of read reference voltages than the ones it used for the prior decoding levels. The decoder then goes back to Step 1 to generate the new LLR information, using the output of *all* of the read operations performed for each decoding level so far. We discuss how the number of decoding levels and the read reference voltages are determined, as well as what happens if *all* soft decoding levels fail, in Section 3.3.2.

Step 5—Extracting the Message from the Codeword: As we discuss above, during LDPC codeword encoding, the generator matrix G contains the identity matrix, to ensure that the codeword c includes a verbatim version of m. Therefore, the decoder recovers the k-bit data message m by simply truncating the last $(n-k)$ bits from the n-bit codeword c.

3.3.2 Error Correction Flow

For both BCH and LDPC codes, the SSD controller performs several stages of error correction to retrieve the data, known as the *error correction flow*. The error correction flow is invoked when the SSD performs a read operation. The SSD starts the read operation by using the initial read reference voltages ($V_{initial}$; see Section 3.2.5) to read the raw data stored within a page of NAND flash memory into the controller. Once the raw data is read, the controller starts error correction.

Algorithm 1 Example BCH/LDPC Error Correction Procedure

First Stage: BCH/LDPC Hard Decoding

```
Controller gets stored V_initial values to use as V_ref
Flash chips read page using V_ref
ECC decoder decodes BCH/LDPC
if ECC succeeds then
  Controller sends data to host; exit algorithm
else if number of stage iterations not exceeded then
  Controller invokes V_ref optimization to find new V_ref;
             repeats first stage
end
```

Second Stage (BCH only): NAC

```
Controller reads immediately-adjacent wordline W
while ECC fails and all possible voltage states for
             adjacent wordline not yet tried do
  Controller goes to next neighbor voltage state V
  Controller sets V_ref based on neighbor voltage state V
  Flash chips read page using V_ref
  Controller corrects cells adjacent to W's cells that
             were programmed to V
ECC decoder decodes BCH
if ECC succeeds then
  Controller sends data to host; exit algorithm
  end
end
```

Second Stage (LDPC only): Level X LDPC Soft Decoding

```
while ECC fails and X < maximum level N do
  Controller selects optimal value of V_ref^X
  Flash chips do read-retry using V_ref^X
  Controller recomputes LLR_X^R0 to LLR_X^RX
  ECC decoder decodes LDPC
  if ECC succeeds then
    Controller sends data to host; exit algorithm
else
    Controller goes to soft decoding level X + 1
    end
end
```

Third Stage: Superpage-Level Parity Recovery

```
Flash chips read all other pages in the superpage
Controller XORs all other pages in the superpage
if data extraction succeeds then
  Controller sends data to host
else
  Controller reports uncorrectable error
end
```

Algorithm 1 lists the three stages of an example error correction flow, which can be used to decode either BCH codes or LDPC codes. In the first stage, the ECC engine performs *hard decoding* on the raw data. In hard decoding, the ECC engine uses only the *hard* bit value information (i.e., either a 1 or a 0) read for a cell using a *single* set of read reference voltages. If the first stage succeeds (i.e., the controller detects that the error rate of the data after correction is lower than a predetermined threshold), the flow finishes. If the first stage fails, then the flow moves on to the second stage of error correction. The second stage differs significantly for BCH and for LDPC, which we discuss below. If the second stage succeeds, the flow terminates; otherwise, the flow moves to the third stage of error correction. In the third stage, the controller tries to correct the errors using the more expensive superpage-level parity recovery (see Section 2.1.3). The steps for superpage-level parity recovery are shown in the third stage of Algorithm 1. If the data can be extracted successfully from the other pages in the superpage, the data from the target page can be recovered. Whenever data is successfully decoded or recovered, the data is sent to the host (and it is also reprogrammed into a new physical page to ensure that the *corrected* data values are stored for the logical page). Otherwise, the SSD controller reports an uncorrectable error to the host.

Figure 3.22 compares the error correction flow with BCH codes to the flow with LDPC codes. Next, we discuss the flows used with both BCH codes (Section 3.3.2) and LDPC codes (Section 3.3.2).

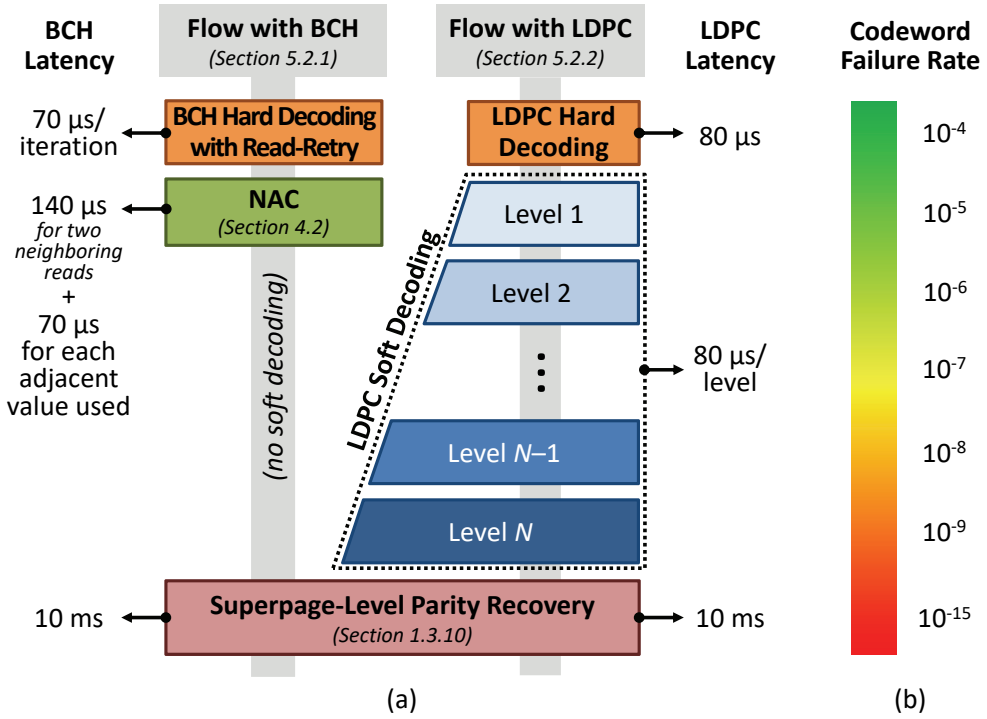


Figure 3.22: (a) Example error correction flow using BCH codes and LDPC codes, with average latency of each BCH/LDPC stage. (b) The corresponding codeword failure rate for each LDPC stage. Adapted from [32].

Flow Stages for BCH Codes

An example flow of the stages for BCH decoding is shown on the left-hand side of Figure 3.22a. In the first stage, the ECC engine performs BCH hard decoding on the raw data, which reports the total number of bit errors in the data. If the data cannot be corrected by the implemented BCH codes, many controllers invoke read-retry (Section 3.2.4) or read reference voltage optimization (Section 3.2.5) to find a new set of read reference voltages (V_{ref}) that lower the raw bit error rate of the data from the error rate when using $V_{initial}$. The controller uses the new V_{ref} values to read the data again, and then repeats the BCH decoding. We discuss the algorithm used to perform decoding for BCH codes in Section 3.3.1.

If the controller exhausts the maximum number of read attempts (specified as a parameter in the controller), it employs correction techniques such as neighbor-cell-assisted correction (NAC; see Section 3.2.2) to further reduce the error rate, as shown in the second BCH stage of Algorithm 1. If NAC cannot successfully read the data, the controller then tries to correct the errors using the more expensive superpage-level parity recovery (see Section 2.1.3).

Flow Stages for LDPC Codes

An example flow of the stages for LDPC decoding is shown on the right-hand side of Figure 3.22a. LDPC decoding consists of three major steps. First, the SSD controller performs LDPC hard decoding, where the controller reads the data using the optimal read reference volt-

ages. The process for LDPC hard decoding is similar to that of BCH hard decoding (as shown in the first stage of Algorithm 1), but does not typically invoke read-retry if the first read attempt fails. Second, if LDPC hard decoding cannot correct all of the errors, the controller uses LDPC *soft decoding* to decode the data (which we describe in detail below). Third, if LDPC soft decoding also cannot correct all of the errors, the controller invokes superpage-level parity. We discuss the algorithm used to perform hard and soft decoding for LDPC codes in Section 3.3.1.

Soft Decoding. Unlike BCH codes, which require the invocation of expensive superpage-level parity recovery immediately if the hard decoding attempts (i.e., BCH hard decoding with read-retry or NAC) fail to return correct data, LDPC decoding fails more gracefully: it can perform multiple levels of *soft decoding* (shown in the second stage of Algorithm 1) after hard decoding fails before invoking superpage-level parity recovery [326, 362]. The key idea of soft decoding is to use *soft* information for each cell (i.e., the *probability* that the cell contains a 1 or a 0) obtained from *multiple* reads of the cell via the use of different sets of read reference voltages [77, 84, 85, 203, 204, 297, 362]. Soft information is typically represented by the *log likelihood ratio* (LLR; see Section 3.3.1).

Every additional level of soft decoding (i.e., the use of a new set of read reference voltages, which we call V_{ref}^{X} for level X) increases the strength of the error correction, as the level *adds* new information about the cell (as opposed to hard decoding, where a new decoding step simply *replaces* prior information about the cell). The new read reference voltages, unlike the ones used for hard decoding, are optimized such that the amount of useful information (or *mutual information*) provided to the LDPC decoder is maximized [326]. Thus, the use of soft decoding reduces the frequency at which superpage-level parity needs to be invoked.

Figure 3.23 illustrates the read reference voltages used during LDPC hard decoding and during the first two levels of LDPC soft decoding. At each level, a new read reference voltage is applied, which divides an existing threshold voltage range into two ranges. Based on the bit values read using the various read reference voltages, the SSD controller bins each cell into a certain V_{th} range, and sends the bin categorization of all the cells to the LDPC decoder. For each cell, the decoder applies an LLR value, precomputed by the SSD manufacturer, which corresponds to the cell's bin and decodes the data. For example, as shown in the bottom of Figure 3.23, the three read reference voltages in Level 2 soft decoding form four threshold voltage ranges (i.e., R0–R3). Each of these ranges corresponds to a different LLR value (i.e., LLR_2^{R0} to LLR_2^{R3}, where LLR_i^{Rj} is the LLR value for range R_j in soft decoding level i). Compared with hard decoding (shown at the top of Figure 3.23), which has only two LLR values, Level 2 soft decoding provides more accurate information to the decoder, and thus has stronger error correction capability.

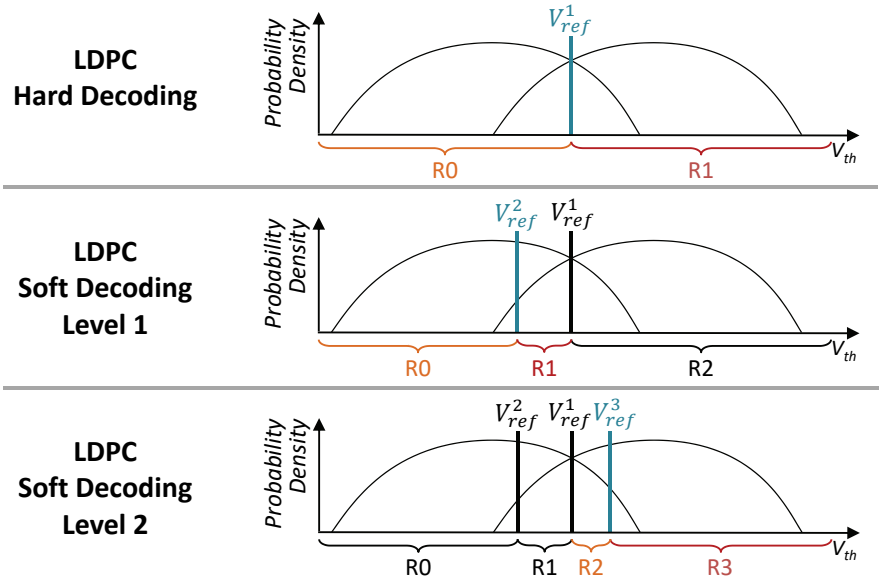


Figure 3.23: LDPC hard decoding and the first two levels of LDPC soft decoding, showing the V_{ref} value added at each level, and the resulting threshold voltage ranges (R0–R3) used for flash cell categorization. Adapted from [32].

Determining the Number of Soft Decoding Levels. If the final level of soft decoding, i.e., level N in Figure 3.22a, fails, the controller attempts to read the data using superpage-level parity (see Section 2.1.3). The number of levels used for soft decoding depends on the improved reliability that each additional level provides, taking into account the latency of performing additional decoding. Figure 3.22b shows a rough estimation of the average latency and the codeword failure rate for each stage. There is a tradeoff between the number of levels employed for soft decoding and the expected read latency. For a smaller number of levels, the additional reliability can be worth the latency penalty. For example, while a five-level soft decoding step requires up to 480 µs, it effectively reduces the codeword failure rate by five orders of magnitude. This not only improves overall reliability, but also reduces the frequency of triggering expensive superpage-level parity recovery, which can take around 10 ms [99]. However, manufacturers limit the number of levels, as the benefit of employing an additional soft decoding level (which requires more read operations) becomes smaller due to diminishing returns in the number of additional errors corrected.

3.3.3 BCH and LDPC Error Correction Strength

BCH and LDPC codes provide different strengths of error correction. While LDPC codes can offer a stronger error correction capability, soft LDPC decoding can lead to a greater latency for error correction. Figure 3.24 compares the error correction strength of BCH codes, hard LDPC codes, and soft LDPC codes [101]. The x-axis shows the raw bit error rate (RBER) of the data being corrected, and the y-axis shows the *uncorrectable bit error rate* (UBER), or the error rate after correction, once the error correction code has been applied. The UBER is defined as the ECC codeword (see Section 2.1.3) failure rate divided by the codeword length [124]. To ensure

a fair comparison, we choose a similar codeword length for both BCH and LDPC codes, and use a similar coding rate (0.935 for BCH, and 0.936 for LDPC) [101]. We make two observations from Figure 3.24.

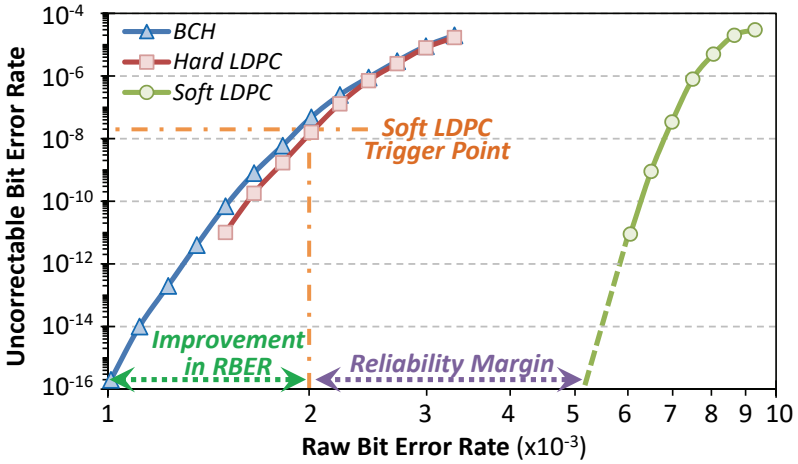


Figure 3.24: Raw bit error rate versus uncorrectable bit error rate for BCH codes, hard LDPC codes, and soft LDPC codes. Reproduced from [32].

First, we observe that the error correction strength of the hard LDPC code is similar to that of the BCH codes. Thus, on its own, hard LDPC does not provide a significant advantage over BCH codes, as it provides an equivalent degree of error correction with similar latency (i.e., one read operation). Second, we observe that soft LDPC decoding provides a significant advantage in error correction capability. Contemporary SSD manufacturers target a UBER of 10^{-16} [124]. The example BCH code with a coding rate of 0.935 can successfully correct data with an RBER of 1.0×10^{-3} while remaining within the target UBER. The example LDPC code with a coding rate of 0.936 is more successful with soft decoding, and can correct data with an RBER as high as 5.0×10^{-3} while remaining within the target UBER, based on the error rate extrapolation shown in Figure 3.24. While soft LDPC can tolerate up to five times the raw bit errors as BCH, this comes at a cost of latency (not shown on the graph), as soft LDPC can require several additional read operations after hard LDPC decoding fails, while BCH requires only the original read.

To understand the benefit of LDPC codes over BCH codes, we need to consider the combined effect of hard LDPC decoding and soft LDPC decoding. As discussed in Section 3.3.2, soft LDPC decoding is invoked *only when hard LDPC decoding fails*. To balance error correction strength with read performance, SSD manufacturers can require that the hard LDPC failure rate cannot exceed a certain threshold, and that the overall read latency (which includes the error correction time) cannot exceed a certain target [99, 101]. For example, to limit the impact of error correction on read performance, a manufacturer can require 99.99% of the error correction operations to be completed after a single read. To meet our example requirement, the hard LDPC failure rate should not be greater than 10^{-4} (i.e., 99.99%), which corresponds to an RBER of 2.0×10^{-3} and a UBER of 10^{-8} (shown as *Soft LDPC Trigger Point* in Figure 3.24). For only the data that contains one or more failed codewords, soft LDPC is invoked (i.e., soft LDPC is invoked only 0.01% of the time). For our example LDPC code with a coding rate of 0.936, soft

LDPC decoding is able to correct these codewords: for an RBER of 2.0×10^{-3}, using soft LDPC results in a UBER well below 10^{-16}, as shown in Figure 3.24.

To gauge the combined effectiveness of hard and soft LDPC codes, we calculate the overhead of using the combined LDPC decoding over using BCH decoding. If 0.01% of the codeword corrections fail, we can assume that in the worst case, each failed codeword resides in a different flash page. As the failure of a single codeword in a flash page causes soft LDPC to be invoked for the entire flash page, our assumption maximizes the number of flash pages that require soft LDPC decoding. For an SSD with four codewords per flash page, our assumption results in up to 0.04% of the data reads requiring soft LDPC decoding. Assuming that the example soft LDPC decoding requires seven additional reads, this corresponds to 0.28% more reads when using combined hard and soft LDPC over BCH codes. Thus, with a 0.28% overhead in the number of reads performed, the combined hard and soft LDPC decoding provides twice the error correction strength of BCH codes (shown as *Improvement in RBER* in Figure 3.24).

In our example, the lifetime of an SSD is limited by both the UBER and whether more than 0.01% of the codeword corrections invoke soft LDPC, to ensure that the overhead of error correction does not significantly increase the read latency [101]. In this case, when the lifetime of the SSD ends, we can still read out the data correctly from the SSD, albeit at an increased read latency. This is because even though we capped the SSD lifetime to an RBER of 2.0×10^{-3} in our example shown in Figure 3.24, soft LDPC is able to correct data with an RBER as high as 5.0×10^{-3} while still maintaining an acceptable UBER (10^{-16}) based on the error rate extrapolation shown. Thus, LDPC codes have a margin, which we call the *reliability margin* and show in Figure 3.24. This reliability margin enables us to trade off lifetime with read latency.

We conclude that with a combination of hard and soft LDPC decoding, an SSD can offer a significant improvement in error correction strength over using BCH codes.

3.3.4 SSD Data Recovery

When the number of errors in data exceeds the ECC correction capability and the error correction techniques in Sections 3.3.2 and 3.3.2 are unable to correct the read data, then data loss can occur. At this point, the SSD is considered to have reached the end of its lifetime. In order to avoid such data loss and *recover* (or, *rescue*) the data from the SSD, we can harness the understanding of data retention and read disturb behavior. The SSD controller can employ two conceptually similar mechanisms, *Retention Failure Recovery* (RFR) [27] and *Read Disturb Recovery* (RDR) [35], to undo errors that were introduced into the data as a result of data retention and read disturb, respectively. The key idea of both of these mechanisms is to exploit the wide variation of different flash cells in their susceptibility to data retention loss and read disturbance effects, respectively, in order to correct some of the errors *without* the assistance of ECC so that the remaining error count falls within the ECC error correction capability.

When a flash page read fails (i.e., uncorrectable errors exist), RFR and RDR record the current threshold voltages of each cell in the page using the read-retry mechanism (see Section 3.2.4), and identify the cells that are *susceptible* to generating errors due to retention and read disturb (i.e., cells that lie at the tails of the threshold voltage distributions of each state, where the distributions overlap with each other), respectively. We observe that some flash cells are more likely to be affected by retention leakage and read disturb than others, as a result of

process variation [27, 35]. We call these cells retention/read disturb *prone*, while cells that are less likely to be affected are called retention/read disturb *resistant*. RFR and RDR classify the susceptible cells as retention/read disturb prone or resistant by inducing *even more* retention and read disturb on the failed flash page, and then recording the new threshold voltages of the susceptible cells. We classify the susceptible cells by observing the magnitude of the threshold voltage shift due to the additional retention/read disturb induction.

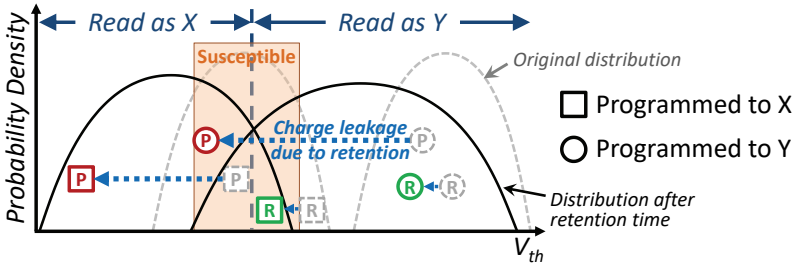


Figure 3.25: Some retention-prone (P) and retention-resistant (R) cells are incorrectly read after charge leakage due to retention time. RFR identifies and corrects the incorrectly read cells based on their leakage behavior. Reproduced from [32].

Figure 3.25 shows how the threshold voltage of a retention-prone cell (i.e., a *fast-leaking* cell, labeled P in the figure) decreases over time (i.e., the cell shifts to the left) due to retention leakage, while the threshold voltage of a retention- resistant cell (i.e., a *slow-leaking* cell, labeled R in the figure) does not change significantly over time. Retention Failure Recovery (RFR) uses this classification of retention- prone versus retention-resistant cells to correct the data from the failed page *without* the assistance of ECC. Without loss of generality, let us assume that we are studying susceptible cells near the intersection of two threshold voltage distributions X and Y, where Y contains higher voltages than X. Figure 3.25 highlights the region of cells considered susceptible by RFR using a box, labeled *Susceptible*. A susceptible cell within the box that is retention prone likely belongs to distribution Y, as a retention-prone cell shifts rapidly to a lower voltage (see the circled cell labeled P within the *susceptible* region in the figure). A retention-resistant cell in the same *susceptible* region likely belongs to distribution X (see the boxed cell labeled R within the *susceptible* region in the figure).

Similarly, Read Disturb Recovery (RDR) uses the classification of read disturb prone versus read disturb resistant cells to correct data. For RDR, disturb-prone cells shift more rapidly to higher voltages, and are thus likely to belong to distribution X, while disturb-resistant cells shift little and are thus likely to belong to distribution Y. Both RFR and RDR correct the bit errors for the susceptible cells based on such *expected* behavior, reducing the number of errors that ECC needs to correct.

RFR and RDR are highly effective at reducing the error rate of failed pages, reducing the raw bit error rate by 50% and 36%, respectively, as shown in prior works from our research group [27, 35], where more detailed information and analyses can be found.

3.4 Emerging Reliability Issues for 3D NAND Flash Memory

While the demand for NAND flash memory capacity continues to grow, manufacturers have found it increasingly difficult to rely on manufacturing process technology scaling to achieve increased capacity [257]. Due to a combination of limitations in manufacturing process technology and the increasing reliability issues as manufacturers move to smaller process technology nodes, planar (i.e., 2D) NAND flash scaling has become difficult for manufacturers to sustain. This has led manufacturers to seek alternative approaches to increase NAND flash memory capacity.

Recently, manufacturers have begun to produce SSDs that contain *three-dimensional* (3D) NAND flash memory [115, 131, 214, 216, 257, 353]. In 3D NAND flash memory, *multiple layers* of flash cells are stacked vertically to increase the density and to improve the scalability of the memory [353]. In order to achieve this stacking, manufacturers have changed a number of underlying properties of the flash memory design.

In this section, we examine these changes, and discuss how they affect the reliability of the flash memory devices. In Section 3.4.1, we discuss the flash memory cell design commonly used in contemporary 3D NAND flash memory, and how these cells are organized across the multiple layers. In Section 3.4.2, we discuss how the reliability of 3D NAND flash memory compares to the reliability of the planar NAND flash memory that we have discussed so far in this chapter. In Section 3.4.3, we briefly discuss error mitigation mechanisms that cater to emerging reliability issues in 3D NAND flash memory. In Chapters 6 and 7, we perform a comprehensive error characterization for 3D NAND using real, state-of-the-art 3D NAND devices, and propose new techniques to mitigate raw bit errors in 3D NAND flash memory.

3.4.1 3D NAND Flash Design and Operation

As we discuss in Section 2.2.1, NAND flash memory stores data as the threshold voltage of each flash cell. In planar NAND flash memory, we achieve this using a floating-gate transistor as a flash cell, as shown in Figure 2.6. The floating-gate transistor stores charge in the floating gate of the cell, which consists of a conductive material. The floating gate is surrounded on both sides by an oxide layer. When high voltage is applied to the control gate of the transistor, charge can migrate through the oxide layers into the floating gate due to Fowler-Nordheim (FN) tunneling [81] (see Section 2.2.4).

Most manufacturers use a *charge trap transistor* as the flash cell in 3D NAND flash memories, instead of using a floating-gate transistor. Figure 3.26 shows the cross section of a charge-trap transistor. Unlike a floating-gate transistor, which stores data in the form of charge within a *conductive* material, a charge trap transistor stores data as charge within an *insulating* material, known as the *charge trap*. In a 3D circuit, the charge trap wraps around a cylindrical transistor substrate, which contains the source (labeled *S* in Figure 3.26) and drain (labeled *D* in the figure), and a control gate wraps around the charge trap. This arrangement allows the channel between the source and drain to form *vertically* within the transistor. As is the case with a floating-gate transistor, a tunnel oxide layer exists between the charge trap and the substrate, and a gate oxide layer exists between the charge trap and the control gate.

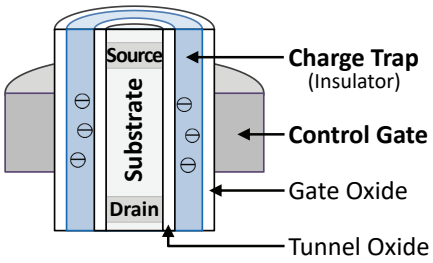


Figure 3.26: Cross section of a charge trap transistor, used as a flash cell in 3D charge trap NAND flash memory.

Despite the change in cell structure, the mechanism for transferring charge into and out of the charge trap is similar to the mechanism for transferring charge into and out of the floating gate. In 3D NAND flash memory, the charge trap transistor typically employs FN tunneling to change the threshold voltage of the charge trap [136, 257].[4] When high voltage is applied to the control gate, electrons are injected into the charge trap from the substrate. As this behavior is similar to how electrons are injected into a floating gate, read, program, and erase operations remain the same for both planar and 3D NAND flash memory.

Figure 3.27 shows how multiple charge trap transistors are physically organized within 3D NAND flash memory to form flash blocks, wordlines, and bitlines (see Section 2.2.2). As mentioned above, the channel within a charge trap transistor forms vertically, as opposed to the horizontal channel that forms within a floating-gate transistor. The vertical orientation of the channel allows us to stack multiple transistors *on top of each other* (i.e., along the z-axis) within the chip, using 3D-stacked circuit integration. The vertically-connected channels form one bitline of a flash block in 3D NAND flash memory. Unlike in planar NAND flash memory, where only the substrates of flash cells on the same bitline are connected together, flash cells along the same bitline in 3D NAND flash memory share a common substrate and a common insulator (i.e., charge trap). The FN tunneling induced by the control gate of the transistor forms a tunnel only in a local region of the insulator, and, thus, electrons are injected only into that local region. Due to the strong insulating properties of the material used for the insulator, different regions of a single insulator can have different voltages. This means that each region of the insulator can store a different data value, and thus, the data of *multiple* 3D NAND flash memory cells can be stored reliably in a *single* insulator. This is because the FN tunneling induced by the control gate of the transistor forms a tunnel only in a *local* region of the insulator, and, thus, electrons are injected only into that local region.

[4]Note that *not* all charge trap transistors rely on FN tunneling. Charge trap transistors used for NOR flash memory change their threshold voltage using *channel hot electron injection*, also known as *hot carrier injection* [200].

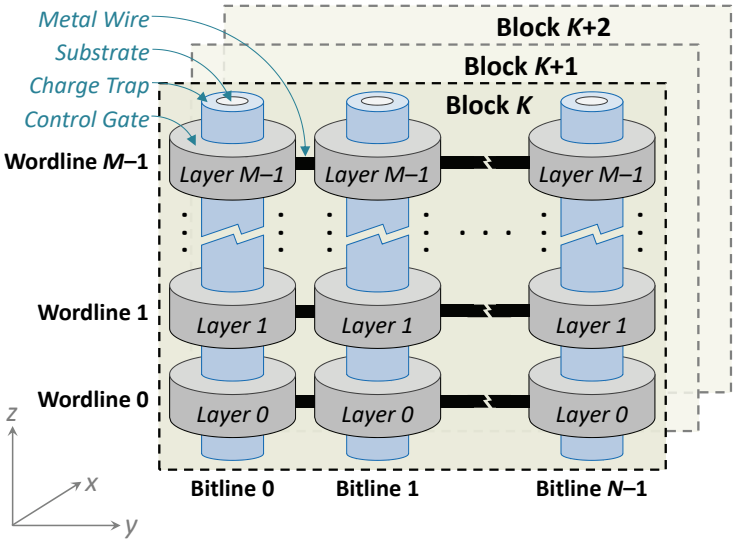


Figure 3.27: Organization of flash cells in an M-layer 3D charge trap NAND flash memory chip, where each block consists of M wordlines and N bitlines.

Each cell along a bitline belongs to a different *layer* of the flash memory chip. Thus, a bitline crosses *all* of the layers within the chip. Contemporary 3D NAND flash memory contains 24–96 layers [79, 131, 143, 257, 319, 353]. Along the y-axis, the control gates of cells within a *single layer* are connected together to form one wordline of a flash block. As we show in Figure 3.27, a block in 3D NAND flash memory consists of all of the flash cells within the same y-z plane (i.e., all cells that have the same coordinate along the x-axis). Note that, while not depicted in Figure 3.27, each bitline within a 3D NAND flash block includes a sense amplifier and two selection transistors used to select the bitline (i.e., the SSL and GSL transistors; see Section 2.2.2). The sense amplifier and selection transistors are connected in series with the charge trap transistors that belong to the same bitline, in a similar manner to the connections shown for a planar NAND flash block in Figure 2.8. More detail on the circuit-level design of 3D NAND flash memory can be found in [119, 136, 159, 311].

Due to the use of multiple layers of flash cells within a single NAND flash memory chip, which greatly increases capacity per unit area, manufacturers can achieve high cell density *without* the need to use small manufacturing process technologies. For example, state-of-the-art planar NAND flash memory uses the 15 nm to 19 nm feature size [195, 260]. In contrast, contemporary 3D NAND flash memory uses larger feature sizes (e.g., 30 nm to 50 nm) [282, 353]. The larger feature sizes reduce manufacturing costs, as their corresponding manufacturing process technologies are much more mature and have a higher yield than the process technologies used for small feature sizes. As we discuss in Section 3.4.2, the larger feature size also has an effect on the reliability of 3D NAND flash memory.

3.4.2 Errors in 3D NAND Flash Memory

While the high-level behavior of 3D NAND flash memory is similar to the behavior of 2D planar NAND flash memory, there are a number of differences between the reliability of 3D NAND

flash and planar NAND flash, which we will look into in Chapter 6. There are two reasons for the differences in reliability: (1) the use of charge trap transistors instead of floating-gate transistors, and (2) moving to a larger manufacturing process technology. We categorize the changes based on the reason for the change below.

Effects of Charge Trap Transistors. Compared to the reliability issues discussed in Section 3.1 for planar NAND flash memory, the use of charge trap transistors introduces two key differences: (1) *early retention loss* [55, 221, 353], and (2) a *reduction in P/E cycling errors* [257, 353].

First, early retention loss refers to the rapid leaking of electrons from a flash cell soon after the cell is programmed [55, 353]. Early retention loss occurs in 3D NAND flash memory because charge can now migrate out of the charge trap in *three* dimensions. In planar NAND flash memory, charge leakage due to retention occurs across the tunnel oxide, which occupies two dimensions (see Section 3.1.4). In 3D NAND flash memory, charge can leak across *both* the tunnel oxide *and* the insulator that is used for the charge trap, i.e., across *three* dimensions. The additional charge leakage takes place for only a few seconds after cell programming. After a few seconds have passed, the impact of leakage through the charge trap decreases, and the long-term cell retention behavior is similar to that of flash cells in planar NAND flash memory [55, 221, 353].

Second, P/E cycling errors (see Section 3.1.1) reduce with 3D NAND flash memory because the tunneling oxide in charge trap transistors is *less* susceptible to breakdown than the oxide in floating-gate transistors during high-voltage operation [221, 353]. As a result, the oxide is less likely to contain trapped electrons once a cell is erased, which in turn makes it less likely that the cell is subsequently programmed to an incorrect threshold voltage. One benefit of the reduction in P/E cycling errors is that the endurance (i.e., the maximum P/E cycle count) for a 3D flash memory cell has increased by more than an order of magnitude [258, 259].

Effects of Larger Manufacturing Process Technologies. Due to the use of larger manufacturing process technologies for 3D NAND flash memory, many of the errors in 2D planar NAND flash (see Section 3.1) are not as prevalent in 3D NAND flash memory. For example, while read disturb is a prominent source of errors at small feature sizes (e.g., 20 nm to 24 nm), its effects are small at larger feature sizes [35]. Likewise, there are much fewer errors due to cell-to-cell program interference (see Section 3.1.3) in 3D NAND flash memory, as the physical distance between neighboring cells is much larger due to the increased feature size. As a result, both cell-to-cell program interference and read disturb are *currently* not major issues in 3D NAND flash memory reliability [257, 259, 353].

One advantage of the lower cell-to-cell program interference is that 3D NAND flash memory uses the older *one-shot programming* algorithm [258, 259, 356] (see Section 2.2.4). In planar NAND flash memory, one-shot programming was replaced by two-step programming (for MLC) and foggy-fine programming (for TLC) in order to reduce the impact of cell-to-cell program interference on fully-programmed cells (as we describe in Section 2.2.4). The lower interference in 3D NAND flash memory makes two-step and foggy-fine programming unnecessary. As a result, none of the cells in 3D NAND flash memory are partially-programmed, significantly reducing the number of program errors (see Section 3.1.2) that occur [259].

Unlike the effects on reliability due to the use of a charge trap transistor, which are likely longer-term, the effects on reliability due to the use of larger manufacturing process technologies are expected to be shorter-term. As manufacturers seek to further increase the density of 3D NAND flash memory, they will reach an upper limit for the number of layers that can be integrated within a 3D-stacked flash memory chip, which is currently projected to be in the range of 300–512 layers [162, 175]. At that point, manufacturers will once again need to scale down the chip to *smaller* manufacturing process technologies [353], which, in turn, will reintroduce high amounts of read disturb and cell-to-cell program interference (just as it happened for planar NAND flash memory [24, 26, 35, 154, 256]).

3.4.3 Changes in Error Mitigation for 3D NAND Flash Memory

Due to the reduction in a number of sources of errors, fewer error mitigation mechanisms are currently needed for 3D NAND flash memory. For example, because the number of errors introduced by cell-to-cell program interference is currently low, manufacturers have *reverted* to using one-shot programming (see Section 2.2.4) for 3D NAND flash [258, 259, 356]. As a result of the currently small effect of read disturb errors, mitigation and recovery mechanisms for read disturb (e.g., pass-through voltage optimization in Section 3.2.5, Read Disturb Recovery in Section 3.3.4) may not be needed, for the time being. We expect that once 3D NAND flash memory begins to scale down to smaller manufacturing process technologies, approaching the current feature sizes used for planar NAND flash memory, there will be a significant need for 3D NAND flash memory to use many, if not all, of the error mitigation mechanisms we discuss in Section 3.2.

To our knowledge, no mechanisms have been designed yet to reduce the impact of early retention loss, which is a new error mechanism in 3D NAND flash memory. This is in part due to the reduced overall impact of retention errors in 3D NAND flash memory compared to planar NAND flash memory [55], since a larger cell contains a greater number of electrons than a smaller cell at the same threshold voltage. As a result, existing refresh mechanisms (see Section 3.2.3) can be used to tolerate errors introduced by early retention loss with little modification. However, as 3D NAND flash memory scales into future smaller technology nodes, the early retention loss problem may require new mitigation techniques.

While new error mitigation mechanisms have yet to emerge for 3D NAND flash memory, rigorous studies that examine error characteristics of and error mitigation techniques for 3D NAND flash memories are yet to be published. These studies (1) may expose additional sources of errors that have not yet been observed, and that may be unique to 3D NAND flash memory; and (2) can enable a solid understanding of current error mechanisms in 3D NAND flash memory so that appropriate specialized mitigation mechanisms can be developed. We expect that future works will experimentally examine such sources of errors, and will potentially introduce novel mitigation mechanisms for these errors. Thus, the field (both academia and industry) is currently in much need of rigorous experimental characterization and analysis of 3D NAND flash memory devices. Our characterization in Chapter 6 is the first in open literature to comprehensively characterization all types of NAND flash memory errors in 3D NAND using real, state-of-the-art MLC 3D charge trap NAND flash memory chips.

3.5 Similar Errors in Other Memory Technologies

As we discussed in Section 3.1, there are five major sources of errors in flash-memory-based SSDs. Many of these error sources can also be found in other types of memory and storage technologies. In this section, we take a brief look at the major reliability issues that exist within DRAM and in emerging nonvolatile memories. In particular, we focus on DRAM in our discussion, as modern SSD controllers have access to dedicated DRAM of considerable capacity (e.g., 1 GB for every 1 TB of SSD capacity), which exists within the SSD package (see Section 2.1). Major sources of errors in DRAM include data retention, cell-to-cell interference, and read disturb. There is a wide body of work on mitigation mechanisms for the DRAM and emerging memory technology errors we describe in this section, but we explicitly discuss only a select number of them here, since a full treatment of such mechanisms is out of the scope of this current chapter.

3.5.1 Cell-to-Cell Interference Errors in DRAM

Another similarity between the capacitive DRAM cell and the floating gate cell in NAND flash memory is that they are both vulnerable to cell-to-cell interference. In DRAM, one important way in which cell-to-cell interference exhibits itself is the data-dependent retention behavior, where the retention time of a DRAM cell is dependent on the values written to *nearby* DRAM cells [137, 138, 139, 140, 187, 261]. This phenomenon is called *data pattern dependence* (DPD) [187]. Data pattern dependence in DRAM is similar to the data-dependent nature of program interference that exists in NAND flash memory (see Section 3.1.3). Within DRAM, data dependence occurs as a result of parasitic capacitance coupling (between DRAM cells). Due to this coupling, the amount of charge stored in one cell's capacitor can inadvertently affect the amount of charge stored in an adjacent cell's capacitor [137, 138, 139, 140, 187, 261]. As DRAM cells become smaller with technology scaling, cell-to-cell interference worsens because parasitic capacitance coupling between cells increases [137, 187]. More findings on cell-to-cell interference and the data-dependent nature of cell retention times in DRAM, along with experimental data obtained from modern DRAM chips, can be found in prior works from our research group [38, 137, 138, 139, 140, 187, 261, 272].

3.5.2 Data Retention Errors in DRAM

DRAM uses the charge within a capacitor to represent one bit of data. Much like the floating gate within NAND flash memory, charge leaks from the DRAM capacitor over time, leading to data retention issues. Charge leakage in DRAM, if left unmitigated, can lead to much more rapid data loss than the leakage observed in a NAND flash cell. While leakage from a NAND flash cell typically leads to data loss after several days to years of retention time (see Section 3.1.4), leakage from a DRAM cell leads to data loss after a retention time on the order of *milliseconds* to *seconds* [187].

The retention time of a DRAM cell depends upon several factors, including (1) manufacturing process variation and (2) temperature [187]. Manufacturing process variation affects the amount of current that leaks from each DRAM cell's capacitor and access transistor [187]. As a result, the

retention time of the cells within a single DRAM chip vary significantly, resulting in *strong cells* that have high retention times and *weak cells* that have low retention times within each chip. The operating temperature affects the rate at which charge leaks from the capacitor. As the operating temperature increases, the retention time of a DRAM cell decreases exponentially [97, 187]. Figure 3.28 shows the change in retention time as we vary the operating temperature, as measured from real DRAM chips [187]. In Figure 3.28, prior work normalizes the retention time of each cell to its retention time at an operating temperature of 50 °C. As the number of cells is large, prior work groups the normalized retention times into bins, and plot the density of each bin. Prior work draws two exponential-fit curves: (1) the *peak* curve, which is drawn through the most populous bin at each temperature measured; and (2) the *tail* curve, which is drawn through the lowest non-zero bin for each temperature measured. Figure 3.28 provides us with three major conclusions about the relationship between DRAM cell retention time and temperature. First, both of the exponential-fit curves fit well, which confirms the exponential decrease in retention time as the operating temperature increases in modern DRAM devices. Second, the retention times of different DRAM cells are affected very differently by changes in temperature. Third, the variation in retention time across cells increases greatly as temperature increases. More analysis of factors that affect DRAM retention times can be found in recent works from our research group [70, 137, 138, 139, 140, 187, 261, 272].

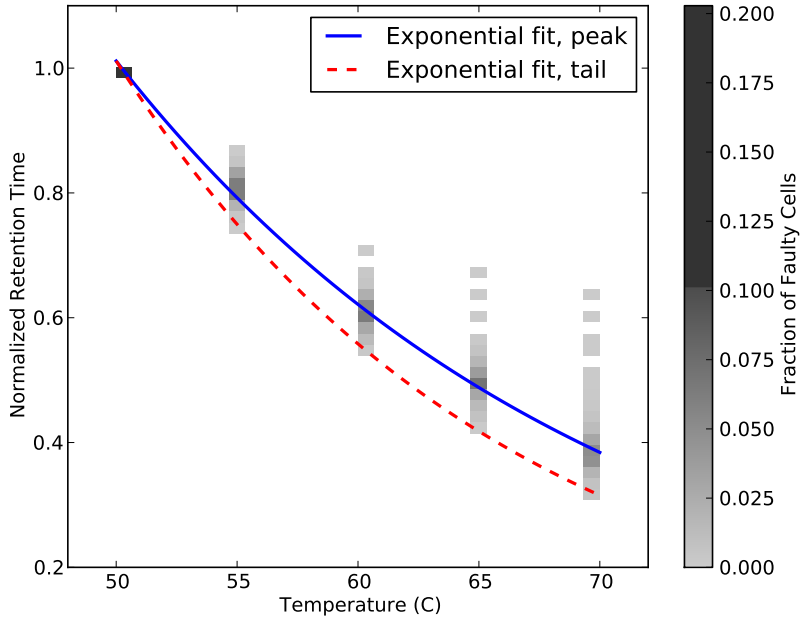
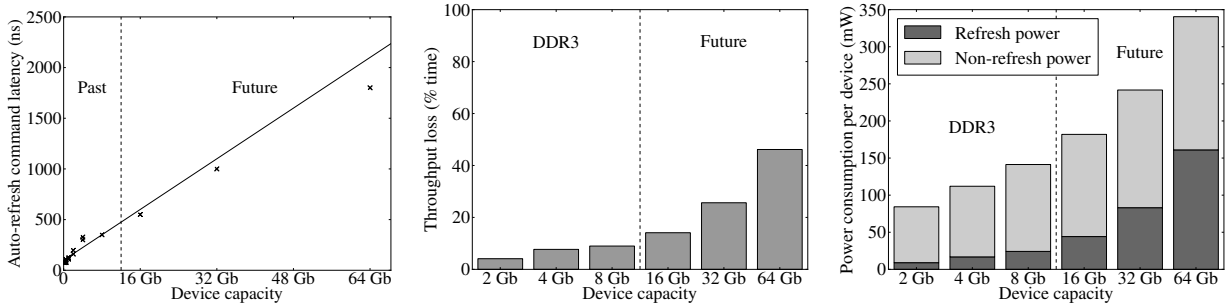


Figure 3.28: DRAM retention time vs. operating temperature, normalized to the retention time of each DRAM cell at 50 °C. Reproduced from [187].

Due to the rapid charge leakage from DRAM cells, a DRAM controller periodically refreshes all DRAM cells in place [39, 120, 137, 186, 187, 261, 272] (similar to the techniques discussed in Section 3.2.3, but at a much smaller time scale). DRAM standards require a DRAM cell to be refreshed once every 64 ms [120]. As the density of DRAM continues to increase over successive product generations (e.g., by 128x between 1999 and 2017 [38, 41]), enabled by the scaling of

DRAM to smaller manufacturing process technology nodes [206], the performance and energy overheads required to refresh an entire DRAM module have grown significantly [39, 186]. It is expected that the refresh problem will get worse and limit DRAM density scaling, as described in a recent work by Samsung and Intel [134] and by our group [186]. Refresh operations in DRAM cause both (1) performance loss and (2) energy waste, both of which together lead to a difficult technology scaling challenge. Refresh operations degrade performance due to three major reasons. First, refresh operations increase the memory latency, as a request to a DRAM bank that is refreshing must wait for the refresh latency before it can be serviced. Second, they reduce the amount of bank-level parallelism available to requests, as a DRAM bank cannot service requests during refresh. Third, they decrease the row buffer hit rate, as a refresh operation causes all open rows in a bank to be closed. When a DRAM chip scales to a greater capacity, there are more DRAM rows that need to be refreshed. As Figure 3.29a shows, the amount of time spent on each refresh operation scales linearly with the capacity of the DRAM chip. The additional time spent on refresh causes the DRAM data throughput loss due to refresh to become more severe in denser DRAM chips, as shown in Figure 3.29b. For a chip with a density of 64 Gbit, nearly 50% of the data throughput is lost due to the high amount of time spent on refreshing all of the rows in the chip. The increased refresh time also increases the effect of refresh on power consumption. As prior work observes from Figure 3.29c, the fraction of DRAM power spent on refresh is expected to be the dominant component of the total DRAM power consumption, as DRAM chip capacity scales to become larger. For a chip with a density of 64 Gbit, nearly 50% of the DRAM chip power is spent on refresh operations. Thus, refresh poses a clear challenge to DRAM scalability.



(a) Refresh latency (b) Throughput loss (c) Power consumption

Figure 3.29: Negative performance and power consumption effects of refresh in contemporary and future DRAM devices. We expect that as the capacity of each DRAM chip increases, (a) the refresh latency, (b) the DRAM throughput lost during refresh operations, and (c) the power consumed by refresh will all increase. Reproduced from [186].

To combat the growing performance and energy overheads of refresh, two classes of techniques have been developed. The first class of techniques reduce the *frequency* of refresh operations without sacrificing the reliability of data stored in DRAM (e.g., [8, 118, 137, 139, 140, 186, 261, 272, 324]). Various experimental studies of real DRAM chips (e.g., [102, 137, 138, 147, 172, 186, 187, 261, 272]) have studied the data retention time of DRAM cells in modern chips. Figure 3.30 shows the retention time measured from seven different real

DRAM modules (by manufacturers A, B, C, D, and E) at an operating temperature of 45 °C, as a cumulative distribution (CDF) of the fraction of cells that have a retention time less than the x-axis value [187]. Prior work observes from the figure that even for the DRAM module whose cells have the worst retention time (i.e., the CDF is the highest), fewer than only 0.001% of the total cells have a retention time smaller than 3 s at 45 °C. As shown in Figure 3.28, the retention time decreases exponentially as the temperature increases. We can extrapolate the observations from Figure 3.30 to the worst-case operating conditions by using the tail curve from Figure 3.28. DRAM standards specify that the operating temperature of DRAM should not exceed 85 °C [120]. Using the tail curve, prior work finds that a retention time of 3 s at 45 °C is equivalent to a retention time of 246 ms at the worst-case temperature of 85 °C. Thus, the vast majority of DRAM cells can retain data without loss for much longer than the 64 ms retention time specified by DRAM standards. The other experimental studies of DRAM chips have validated this observation as well [102, 137, 138, 147, 172, 186, 261, 272].

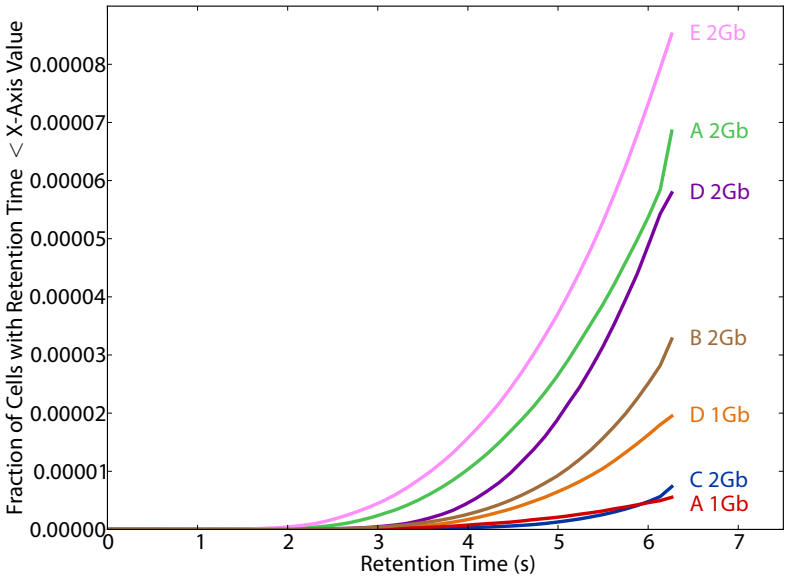


Figure 3.30: Cumulative distribution of the number of cells in a DRAM module with a retention time less than the value on the x-axis, plotted for seven different DRAM modules. Reproduced from [187].

A number of works take advantage of this variability in data retention time behavior across DRAM cells, by introducing heterogeneous refresh rates, i.e., different refresh rates for different DRAM rows. Thus, these works can reduce the frequency at which the vast majority of DRAM rows within a module are refreshed (e.g., [8, 118, 137, 139, 186, 187, 261, 272, 324]). For example, the key idea of RAIDR [186] is to refresh the *strong* DRAM rows (i.e., those rows that can retain data for much longer than the minimum 64 ms retention time in the DDR4 standard [120]) less frequently, and refresh the *weak* DRAM rows (i.e., those rows that can retain data only for the minimum retention time) more frequently. The major challenge in such works is how to accurately identify the retention time of each DRAM row. To solve this challenge, many recent works examine (online) DRAM retention time profiling techniques [137, 138, 140, 187, 261, 272].

The second class of techniques reduce the interference caused by refresh requests on demand requests (e.g., [39, 230, 304]). These works either change the scheduling order of refresh requests [39, 230, 304] or slightly modify the DRAM architecture to enable the servicing of refresh and demand requests in parallel [39].

One critical challenge in developing techniques to reduce refresh overheads is that it is getting significantly more difficult to determine the minimum retention time of a DRAM cell, as prior works have shown experimentally on modern DRAM chips [137, 138, 187, 261, 272]. Thus, determining the correct rate at which to refresh DRAM cells has become more difficult, as also indicated by industry [134]. This is due to two major phenomena, both of which get worse (i.e., become more prominent) with manufacturing process technology scaling. The first phenomenon is *variable retention time* (VRT), where the retention time of some DRAM cells can change drastically over time, due to a memoryless random process that results in very fast charge loss via a phenomenon called *trap-assisted gate-induced drain leakage* [187, 272, 278, 348]. VRT, as far as we know, is very difficult to test for, because there seems to be no way of determining that a cell exhibits VRT until that cell is observed to exhibit VRT, and the time scale of a cell exhibiting VRT does not seem to be bounded, based on the current experimental data on modern DRAM devices [187, 261]. The second phenomenon is *data pattern dependence* (DPD), which we discuss in Section 3.5.1. Both of these phenomena greatly complicate the accurate determination of minimum data retention time of DRAM cells. Therefore, data retention in DRAM continues to be a vulnerability that can greatly affect DRAM technology scaling (and thus performance and energy consumption) as well as the reliability and security of current and future DRAM generations.

More findings on the nature of DRAM data retention and associated errors, as well as relevant experimental data from modern DRAM chips, can be found in prior works from our research group [38, 39, 102, 137, 138, 139, 140, 172, 186, 187, 233, 261, 272].

3.5.3 Read Disturb Errors in DRAM

Commodity DRAM chips that are sold and used in the field today exhibit read disturb errors [156], also called *RowHammer*-induced errors [233], which are *conceptually* similar to the read disturb errors found in NAND flash memory (see Section 3.1.5). Repeatedly accessing the same row in DRAM can cause bit flips in data stored in adjacent DRAM rows. In order to access data within DRAM, the row of cells corresponding to the requested address must be *activated* (i.e., opened for read and write operations). This row must be *precharged* (i.e., closed) when another row in the same DRAM bank needs to be activated. Through experimental studies on a large number of real DRAM chips, prior work shows that when a DRAM row is activated and precharged repeatedly (i.e., *hammered*) enough times within a DRAM refresh interval, one or more bits in physically-adjacent DRAM rows can be flipped to the wrong value [156].

Prior work tested 129 DRAM modules manufactured by three major manufacturers (A, B, and C) between 2008 and 2014, using an FPGA-based experimental DRAM testing infrastructure [102] (more detail on the experimental setup, along with a list of all modules and their characteristics, can be found in the original RowHammer paper [156]). Figure 3.31 shows the rate of RowHammer errors, with the 129 modules that prior work tested categorized based on their manufacturing date. Prior work finds that 110 of the tested modules exhibit RowHammer

errors, with the earliest such module dating back to 2010. In particular, prior work finds that *all* of the modules manufactured in 2012–2013 that prior work tested are vulnerable to RowHammer. Like with many NAND flash memory error mechanisms, especially read disturb, RowHammer is a recent phenomenon that especially affects DRAM chips manufactured with more advanced manufacturing process technology generations.

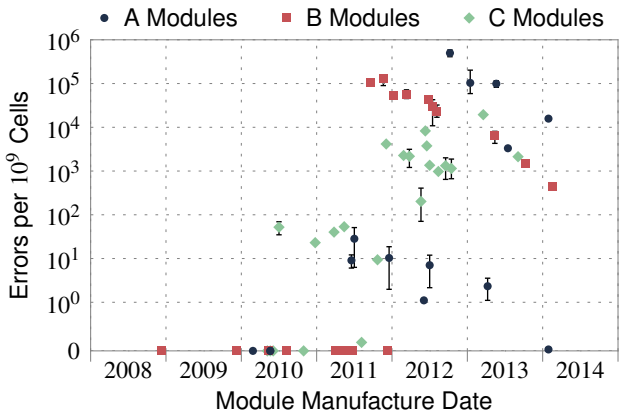


Figure 3.31: RowHammer error rate vs. manufacturing dates of 129 DRAM modules we tested. Reproduced from [156].

Figure 3.32 shows the distribution of the number of rows (plotted in log scale on the y-axis) within a DRAM module that flip the number of bits along the x-axis, as measured for example DRAM modules from three different DRAM manufacturers [156]. Prior work makes two observations from the figure. First, the number of bits flipped when we hammer a row (known as the *aggressor row*) can vary significantly within a module. Second, each module has a different distribution of the number of rows. Despite these differences, prior work finds that this DRAM failure mode affects more than 80% of the DRAM chips prior work tested [156]. As indicated above, this read disturb error mechanism in DRAM is popularly called RowHammer [233].

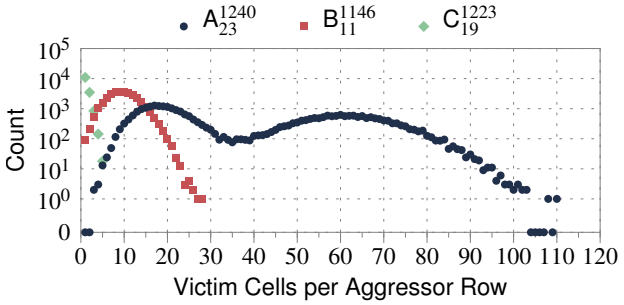


Figure 3.32: Number of victim cells (i.e., number of bit errors) when an aggressor row is repeatedly activated, for three representative DRAM modules from three major manufacturers. We label the modules in the format X_n^{yyww}, where X is the manufacturer (A, B, or C), $yyww$ is the manufacture year (yy) and week of the year (ww), and n is the number of the selected module. Reproduced from [156].

Various recent works show that RowHammer can be maliciously exploited by user-level software programs to (1) induce errors in existing DRAM modules [156, 233] and (2) launch attacks to compromise the security of various systems [15, 18, 90, 91, 233, 276, 287, 288, 322, 343]. For example, by exploiting the RowHammer read disturb mechanism, a user-level program can gain kernel-level privileges on real laptop systems [287, 288], take over a server vulnerable to RowHammer [90], take over a victim virtual machine running on the same system [15], and take over a mobile device [322]. Thus, the RowHammer read disturb mechanism is a prime (and perhaps the first) example of how a circuit-level failure mechanism in DRAM can cause a practical and widespread system security vulnerability. We believe similar (yet likely more difficult to exploit) vulnerabilities exist in MLC NAND flash memory as well, as described in recent work from our research group [34].

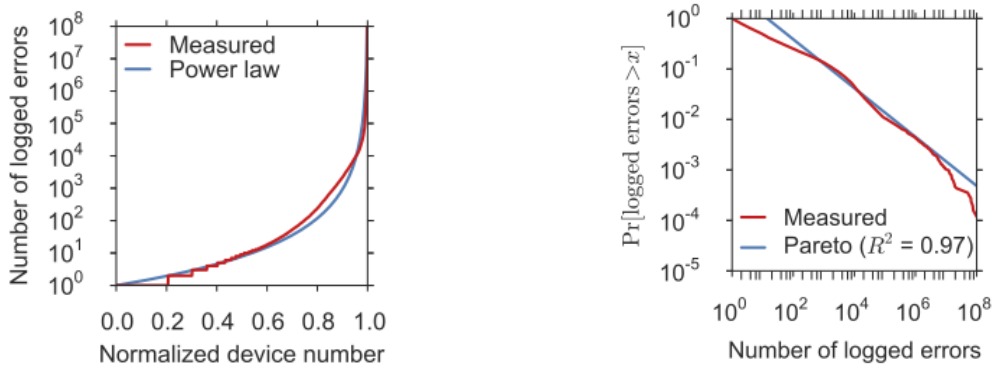
Note that various solutions to RowHammer exist [149, 156, 233], but we do not discuss them in detail here. Recent work from our research group [233] provides a comprehensive overview. A very promising proposal is to modify either the memory controller or the DRAM chip such that it probabilistically refreshes the physically-adjacent rows of a recently-activated row, with very low probability. This solution is called *Probabilistic Adjacent Row Activation* (PARA) [156]. Prior work shows that this low-cost, low-complexity solution, which does not require any storage overhead, greatly closes the RowHammer vulnerability [156].

The RowHammer effect in DRAM worsens as the manufacturing process scales down to smaller node sizes [156, 233]. More findings on RowHammer, along with extensive experimental data from real DRAM devices, can be found in prior works from our research group [149, 156, 233].

3.5.4 Large-Scale DRAM Error Studies

Like flash memory, DRAM is employed in a wide range of computing systems, at scale. Thus, there is a similar need to study the aggregate behavior of errors observed in a large number of DRAM chips deployed in the field. Akin to the large-scale flash memory SSD reliability studies discussed in Section 3.1.7, a number of experimental studies characterize the reliability of DRAM at large scale in the field (e.g., [113, 211, 284, 301, 302]). We highlight three notable results from these studies.

First, as prior work saw for large-scale studies of SSDs (see Section 3.1.7), the number of errors observed varies significantly for each DRAM module [211]. Figure 3.33a shows the distribution of correctable errors across the *entire fleet* of servers at Facebook over a fourteen-month period, omitting the servers that did not exhibit any correctable DRAM errors. The x-axis shows the normalized device number, with devices sorted based on the number of errors they experienced in a month. As prior work saw in the case of SSDs, a small number of servers accounts for the majority of errors. As prior work sees from Figure 3.33a, the top 1% of servers account for 97.8% of all observed correctable DRAM errors. The distribution of the number of errors among servers follows a power law model. Prior work shows the probability density distribution of correctable errors in Figure 3.33b, which indicates that the distribution of errors across servers follows a Pareto distribution, with a decreasing hazard rate [211]. This means that a server that has experienced more errors in the past is likely to experience more errors in the future.



(a) (b)

Figure 3.33: Distribution of memory errors among servers with errors (a), which resembles a power law distribution. Memory errors follow a Pareto distribution among servers with errors (b). Reproduced from [211].

Second, unlike SSDs, DRAM does *not* seem to show any clearly discernible trend where higher utilization and age lead to a greater raw bit error rate [211].

Third, the increase in the density of DRAM chips with technology scaling leads to higher error rates [211]. The latter is illustrated in Figure 3.34, which shows how different DRAM chip densities are related to device failure rate. We can see that there is a clear trend of increasing failure rate with increasing chip density. Prior work finds that the failure rate increases because despite small improvements in the reliability of an *individual* cell, the quadratic increase in the number of cells per chip greatly increases the probability of observing a single error in the whole chip [211].

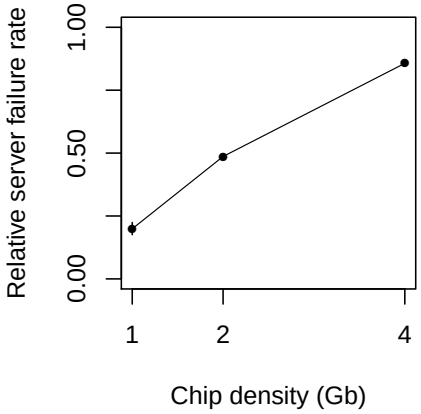


Figure 3.34: Relative failure rate for servers with different chip densities. Higher densities (related to newer technology nodes) show a trend of higher failure rates. Reproduced from [211]. See Section II-E of [211] for the complete definition of the metric plotted on the y-axis, i.e., *relative server failure rate*.

3.5.5 Latency-Related Errors in DRAM

Various experimental studies examine the tradeoff between DRAM reliability and latency [37, 38, 41, 42, 102, 146, 167, 171, 172]. These works perform extensive experimental studies on real DRAM chips to identify the effect of (1) temperature, (2) supply voltage, and (3) manufacturing process variation that exists in DRAM on the latency and reliability characteristics of different DRAM cells and chips. The temperature, supply voltage, and manufacturing process variation all dictate the amount of time that each cell needs to safely complete its operations. Several of the works from our research group [41, 42, 171, 172] examine how one can *reliably* exploit different effects of variation to improve DRAM performance or energy consumption.

Adaptive-Latency DRAM (AL-DRAM) [172] shows that significant variation exists in the access latency of (1) different DRAM modules, as a result of manufacturing process variation; and (2) the same DRAM module over time, as a result of varying operating temperature, since at low temperatures DRAM can be accessed faster. The key idea of AL-DRAM is to adapt the DRAM latency to the operating temperature and the DRAM module that is being accessed. Experimental results show that AL-DRAM can reduce DRAM read latency by 32.7% and write latency by 55.1%, averaged across 115 DRAM modules operating at 55 °C [172].

Voltron [42] identifies the relationship between the DRAM supply voltage and access latency variation. Voltron uses this relationship to identify the combination of voltage and access latency that minimizes system-level energy consumption without exceeding a user-specified threshold for the maximum acceptable performance loss. For example, at an average performance loss of only 1.8%, Voltron reduces the DRAM energy consumption by 10.5%, which translates to a reduction in the overall system energy consumption of 7.3%, averaged over seven memory-intensive quad-core workloads [42].

Flexible-Latency DRAM (FLY-DRAM) [41] captures access latency variation across DRAM cells *within* a single DRAM chip due to manufacturing process variation. For example, Figure 3.35 shows how the bit error rate (BER) changes if we reduce one of the timing parameters used to control the DRAM access latency below the minimum value specified by the manufacturer [41]. Prior work uses an FPGA-based experimental DRAM testing infrastructure [102] to measure the BER of 30 real DRAM modules, over a total of 7500 rounds of tests, as we lower the t_{RCD} timing parameter (i.e., how long it takes to open a DRAM row) below its standard value of 13.125 ns.[5] In this figure, prior work uses a box plot to summarize the bit error rate measured during each round. For each box, the bottom, middle, and top lines indicate the 25th, 50th, and 75th percentile of the population. The ends of the whiskers indicate the minimum and maximum BER of all modules for a given t_{RCD} value. Each round of BER measurement is represented as a single point overlaid upon the box. From the figure, prior work makes three observations. First, the BER decreases exponentially as we reduce t_{RCD}. Second, there are no errors when t_{RCD} is at 12.5 ns or at 10.0 ns, indicating that manufacturers provide a significant latency *guard-band* to provide additional protection against process variation. Third, the BER variation across different models becomes smaller as t_{RCD} decreases. The reliability of a module operating at $t_{RCD} = 7.5$ ns varies significantly based on the DRAM manufacturer and model. This variation occurs because the number of DRAM cells that experience an error within a DRAM chip varies

[5] More detail on the experimental setup, along with a list of all modules and their characteristics, can be found in the original FLY-DRAM paper [41].

significantly from module to module. Yet, the BER variation across different modules operating at $t_{RCD} = 2.5$ ns is much smaller, as most modules fail when the latency is reduced so significantly.

From other experiments that prior work describes in the FLY-DRAM paper [41], prior work finds that there is spatial locality in the slower cells, resulting in *fast regions* (i.e., regions where all DRAM cells can operate at significantly-reduced access latency without experiencing errors) and *slow regions* (i.e., regions where *some* of the DRAM cells *cannot* operate at significantly-reduced access latency without experiencing errors) within each chip. To take advantage of this heterogeneity in the reliable access latency of DRAM cells within a chip, FLY-DRAM (1) categorizes the cells into fast and slow regions; and (2) lowers the overall DRAM latency by accessing fast regions with a lower latency. FLY-DRAM lowers the timing parameters used for the fast region by as much as 42.8% [41]. FLY-DRAM improves system performance for a wide variety of real workloads, with the average improvement for an eight-core system ranging between 13.3% and 19.5%, depending on the amount of variation that exists in each module [41].

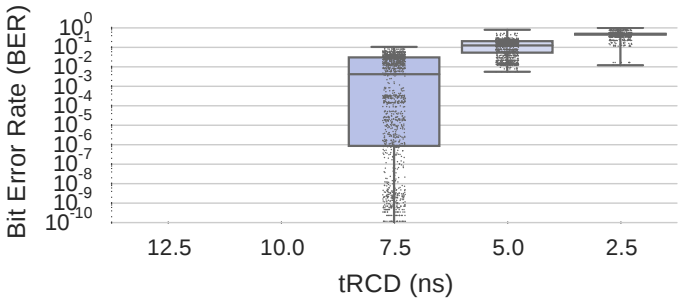


Figure 3.35: Bit error rates of tested DRAM modules as we reduce the DRAM access latency (i.e., the t_{RCD} timing parameter). Reproduced from [41].

Design-Induced Variation-Aware DRAM (DIVA-DRAM) [171] identifies the latency variation within a single DRAM chip that occurs due to the architectural design of the chip. For example, a cell that is further away from the row decoder requires a longer access time than a cell that is close to the row decoder. Similarly, a cell that is farther away from the wordline driver requires a larger access time than a cell that is close to the wordline driver. DIVA-DRAM uses design-induced variation to reduce the access latency to different parts of the chip. One can further reduce latency by sacrificing some amount of reliability and performing error correction to fix the resulting errors [171]. Experimental results show that DIVA-DRAM can reduce DRAM read latency by 40.0% and write latency by 60.5% [171]. In an eight-core system running a wide variety of real workloads, DIVA-DRAM improves system performance by an average of 13.8% [171].

More information about the errors caused by reduced latency and reduced voltage operation in DRAM chips and the tradeoff between reliability and latency and voltage can be found in prior works from our research group [38, 41, 42, 102, 167, 171, 172, 193].

3.5.6 Error Correction in DRAM

In order to protect the data stored within DRAM from various types of errors, some (but not all) DRAM modules employ ECC [193]. The ECC employed within DRAM is much weaker than the ECC employed in SSDs (see Section 3.3) for various reasons. First, DRAM has a much lower access latency, and error correction mechanisms should be designed to ensure that DRAM access latency does not increase significantly. Second, the error rate of a DRAM chip tends to be lower than that of a flash memory chip. Third, the granularity of access is much smaller in a DRAM chip than in a flash memory chip, and hence sophisticated error correction can come at a high cost. The most common ECC algorithm used in commodity DRAM modules is *SECDED* (single error correction, double error detection) [193]. Another ECC algorithm available for some commodity DRAM modules is *Chipkill*, which can tolerate the failure of an *entire* DRAM chip within a module [74] at the expense of higher storage overhead and higher latency. For both SECDED and Chipkill, the ECC information is stored on one or more extra chips within the DRAM module, and, on a read request, this information is sent alongside the data to the memory controller, which performs the error detection and correction.

As DRAM scales to smaller technology nodes, its error rate continues to increase [134, 156, 206, 211, 232, 233, 236]. Effects like read disturb [156], cell-to-cell interference [137, 138, 139, 140, 187, 261], and variable retention time [137, 187, 261, 272] become more severe [134, 156, 232, 233, 236]. As a result, there is an increasing need for (1) employing ECC algorithms in *all* DRAM chips/modules; (2) developing more sophisticated and efficient ECC algorithms for DRAM chips/modules; and (3) developing error-specific mechanisms for error correction. To this end, recent work follows various directions. First, in-DRAM ECC, where correction is performed within the DRAM module itself (as opposed to in the controller), is proposed [134]. One work shows how exposing this in-DRAM ECC information to the memory controller can provide Chipkill-like error protection at much lower overhead than the traditional Chipkill mechanism [238]. Second, various works explore and develop stronger ECC algorithms for DRAM (e.g., [144, 145, 330]), and explore how to make ECC more efficient based on the current DRAM error rate (e.g., [4, 58, 74, 171, 320]). Third, recent work shows how the cost of ECC protection can be reduced by (1) exploiting *heterogeneous reliability memory* [193], where different portions of DRAM use different strengths of error protection based on the error tolerance of different applications and different types of data [190, 193], and (2) using the additional DRAM capacity that is otherwise used for ECC to improve system performance when reliability is not as important for the given application and/or data [197].

Many of these works that propose error mitigation mechanisms for DRAM do *not* distinguish between the characteristics of different types of errors. We believe that, in addition to providing sophisticated and efficient ECC mechanisms in DRAM, there is also significant value in and opportunity for exploring *specialized* error mitigation mechanisms that are *customized for different error types*, just as it is done for flash memory (as we discussed in Section 3.2). One such example of a specialized error mitigation mechanism is targeted to fix the RowHammer read disturb mechanism, and is called *Probabilistic Adjacent Row Activation* (PARA) [156, 233], as we discussed earlier. Recall that the key idea of PARA is to refresh the rows that are physically adjacent to an activated row, with a very low probability. PARA is shown to be very effective in fixing the RowHammer problem at no storage cost and at very low performance overhead [156].

PARA is a specialized yet very effective solution for fixing a specific error mechanism that is important and prevalent in modern DRAM devices.

3.5.7 Errors in Emerging Nonvolatile Memory Technologies

DRAM operations are several orders of magnitude faster than SSD operations, but DRAM has two major disadvantages. First, DRAM offers orders of magnitude less storage density than NAND-flash-memory-based SSDs. Second, DRAM is volatile (i.e., the stored data is lost on a power outage). Emerging nonvolatile memories, such as *phase-change memory* (PCM) [163, 164, 165, 273, 333, 351, 363], *spin-transfer torque magnetic RAM* (STT-RAM or STT-MRAM) [161, 237], *metal-oxide resistive RAM* (RRAM) [334], and *memristors* [62, 303], are expected to bridge the gap between DRAM and SSDs, providing DRAM-like access latency and energy, and at the same time SSD-like large capacity and nonvolatility (and hence SSD-like data persistence). These technologies are also expected to be used as part of *hybrid memory systems* (also called *heterogeneous memory systems*), where one part of the memory consists of DRAM modules and another part consists of modules of emerging technologies [46, 59, 60, 128, 183, 210, 212, 267, 273, 274, 275, 351, 352, 357, 360]. PCM-based devices are expected to have a limited lifetime, as PCM can only endure a certain number of writes [163, 273, 333], similar to the P/E cycling errors in NAND-flash-memory-based SSDs (though PCM's write endurance is higher than that of SSDs). PCM suffers from (1) *resistance drift* [114, 268, 333], where the resistance used to represent the value becomes higher over time (and eventually can introduce a bit error), similar to how charge leakage in NAND flash memory and DRAM lead to retention errors over time; and (2) *write disturb* [127], where the heat generated during the programming of one PCM cell dissipates into neighboring cells and can change the value that is stored within the neighboring cells. STT-RAM suffers from (1) *retention failures*, where the value stored for a single bit (as the magnetic orientation of the layer that stores the bit) can flip over time; and (2) *read disturb* (a conceptually different phenomenon from the read disturb in DRAM and flash memory), where reading a bit in STT-RAM can inadvertently induce a write to that same bit [237]. Due to the nascent nature of emerging nonvolatile memory technologies and the lack of availability of large-capacity devices built with them, extensive and dependable experimental studies have yet to be conducted on the reliability of real PCM, STT-RAM, RRAM, and memristor chips. However, we believe that error mechanisms conceptually or abstractly similar to those we discussed in this chapter for flash memory and DRAM are likely to be prevalent in emerging technologies as well (as supported by some recent studies [7, 127, 141, 237, 298, 299, 361]), albeit with different underlying mechanisms and error rates.

Chapter 4

WARM—Write-hotness Aware Retention Management

As we have introduced in Section 3.1, retention errors are one of the most dominant error sources in NAND flash memory. Retention errors are caused by charge leakage from the flash cells after the data has been programmed into the cells. Our prior work has shown that *retention errors not only degrade data reliability, but also lead to performance degradation* due to increased read-retry attempts [27]. In Section 3.2, we have surveyed many techniques that can mitigate retention errors proposed by prior work, including refresh, read-retry, voltage optimization, etc. Among these techniques, refresh is considered to be the most effective in improving flash lifetime and has recently been implemented in real SSDs [22, 25, 295]. However, we observe that, while refresh improves flash lifetime significantly by relaxing the retention time constraint, *the refresh operation itself can consume the majority of the improved flash lifetime*, wasting opportunity for further flash lifetime improvements.

In this chapter, we introduce *Write-hotness Aware Retention Management* (WARM), which exploits workload *write-hotness* and device *data retention* characteristics to improve flash lifetime. The goal of WARM is to eliminate redundant refreshes for write-hot pages with minimal storage and performance overhead. This work proposes a write hotness-aware flash memory retention management policy, WARM. The *first* key idea of WARM is to effectively partition pages stored in flash memory into two groups based on the write frequency of the pages. The *second* key idea of WARM is to apply different management policies to the two different groups of flash pages/blocks to improve the lifetime of the flash device.

First, we look into flash memory data retention characteristics and SSD workload characteristics to show that redundant flash refresh operations are expensive (Section 4.1). Second, we discuss a novel, lightweight approach to dynamically identifying and partitioning write-hot versus write- cold pages (Section 4.2.1). Third, we describe how WARM optimizes flash management policies, such as garbage collection and wear-leveling, in a partitioned flash memory, and show how WARM integrates with a refresh mechanism to provide further flash lifetime improvements (Section 4.2.2). Fourth, we evaluate the flash lifetime improvement delivered by WARM, and the hardware and performance overhead to implement WARM (Section 4.4). Finally, we conclude with the contributions of WARM (Section 4.6).

4.1 Motivation

As flash memory density has continued to increase, the endurance of flash devices has been rapidly decreasing. Relaxing the *internal* retention time of flash devices can significantly improve upon this endurance (Section 4.1.1), but this cannot simply be externally exposed, as the relaxation would impact the data integrity guarantee. Periodically performing data refresh allows the flash device to relax the internal retention time while maintaining the data integrity guarantee [21, 22, 25, 188, 189, 222, 250]. Unfortunately, for real-world workloads, these refresh operations consume a large portion of the extra endurance gained from internal retention time relaxation, as we describe in Section 4.1.2. In order to buy back the endurance, *we aim to eliminate redundant refresh operations on write-hot data*, as the write-hot flash pages incur the vast majority of writes (Section 4.1.3). We use the insights from this section to design a write-hotness aware retention management policy for flash memory, which we describe in Section 4.2.

4.1.1 Retention Time Relaxation

Traditionally, data stored within a block is retained for some amount of time. This retention time is dependent on a number of factors (e.g., the number of P/E cycles already performed on the block, process variation). Flash devices guarantee a minimum data integrity time. The endurance of flash memory is a factor of how many P/E cycles can take place before the internal retention time falls below this minimum guarantee.

Prior work has shown that P/E cycle endurance of flash memory can be significantly improved by relaxing the internal retention time [21, 22, 25, 188, 189, 222, 250]. We extrapolate the endurance numbers under different internal retention times and plot them in Figure 4.1. The horizontal axis shows the flash endurance, expressed as the number of P/E cycles before the device experiences retention failures. Each bar shows the number of P/E cycles a flash block can tolerate for a given internal retention time. With a three-year internal retention time, which is the typical retention period used in today's flash drives, each flash cell can endure only 3,000 P/E cycles. However, as we relax the internal retention time to three days, flash endurance can improve by up to 50× (i.e., 150,000 P/E cycles). Hence, relaxing the amount of time that flash memory is required to internally retain data can potentially lead to great improvements in endurance.

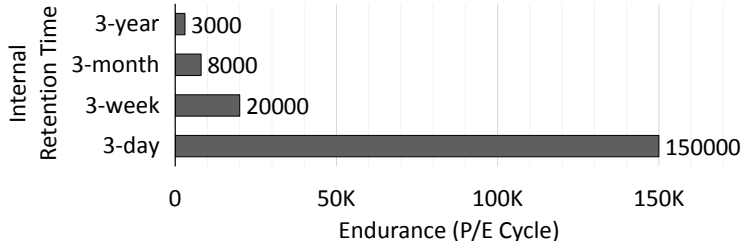


Figure 4.1: P/E cycle endurance from different amounts of internal retention time without refresh. (Data extrapolated from prior work [22, 25].)

4.1.2 Refresh Overhead Mitigation

In order to compensate for the reduced internal retention time, refresh operations are introduced to maintain the data integrity guarantee provided to the user [22, 25]. When the internal retention time of a flash block expires, the data stored in the block can be simply remapped to another block (i.e., all valid pages within a block are read, corrected, and then reprogrammed to a different block) to extend the duration of data integrity. Several variants of refresh have been proposed for flash memory [22, 25, 188, 189, 222, 250]. Remapping-based flash correct-and-refresh (FCR) involves the lowest implementation overhead, by triggering refreshes at a fixed refresh frequency to guarantee the retention time never falls below a predetermined threshold [22, 25].

Although relaxing internal retention time increases flash endurance, each refresh operation consumes a portion of this extra endurance, leading to significantly reduced lifetime improvements. The curves in Figure 4.2 plot the relation between the fraction of the extra endurance cycles consumed by refresh operations for a 256 GB flash drive and the write intensity of the workload (expressed as the average number of writes the workload issues to the drive per day) for a refresh mechanism with various refresh intervals (ranging from three days to three years). When the write intensity is as low as 10^5 writes/day, refresh operations can consume up to 99% of the total endurance when the data is refreshed every three days (regardless of how recently the data was written). The data points in Figure 4.2 show the actual fraction of writes that are due to refresh for each workload that we evaluate in Section 4.4. Fourteen of the sixteen workloads are disk traces from real applications, and they all have a write frequency less than or equal to 10^7 writes/day. The remaining two workloads are I/O benchmarks (`iozone` and `postmark`) with higher write frequencies, which do not represent the typical usage of flash devices. Unfortunately, refresh operations consume a significant fraction of the extra endurance for all fourteen real-world workloads. In this chapter, we aim to reduce the fraction of endurance consumed as overhead, in order to better utilize the extra endurance gained from retention time relaxation and thus improve flash lifetime.

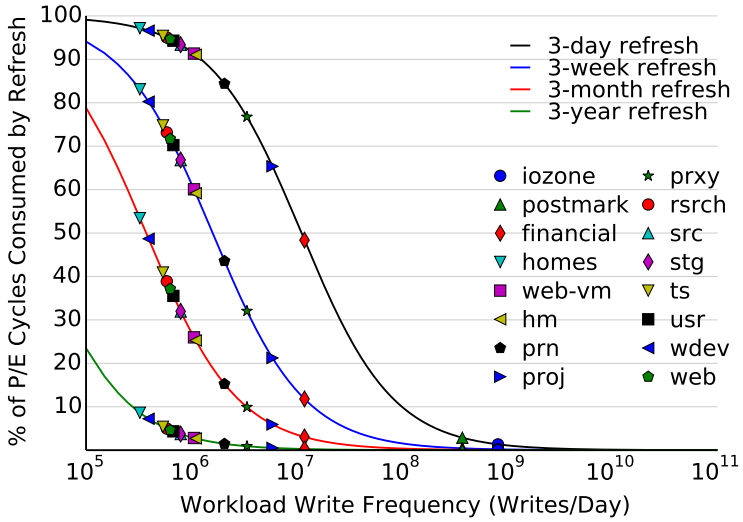


Figure 4.2: Fraction of P/E cycles consumed by refresh operations.

4.1.3 Opportunities to Exploit *Write-Hotness*

Many of the management policies for flash memory are focused on writes, as write operations reduce the lifetime of the device. While these algorithms were designed to evenly distribute device wear-out across blocks, they crucially ignore the fine-grained behavior of writes across pages within an application. We observe that write frequency can be quite heterogeneous across different pages. While some pages are often written to (*write-hot pages*), other pages are infrequently updated (*write-cold pages*). Figure 4.3 shows the write distribution for all sixteen of our applications (described in Table 4.2). We observe that for all but one of our workloads (`postmark`), only a very small fraction (i.e., less than 1%) of the total application data receives the vast majority of the write requests. In fact, from Figure 4.3, we observe that for ten of our applications, a very small fraction of all data pages (i.e., less than 1%) are the destination of *nearly 100%* of the write requests. Note that our workloads use a total memory footprint of 217.6GB each, and that 1% of the total application data represents 2.176GB. We conclude from this figure that only a small portion of the pages in each application are *write-hot*, and that the discrepancy between the write rate to write-hot pages and the write rate to write-cold pages is highly skewed.

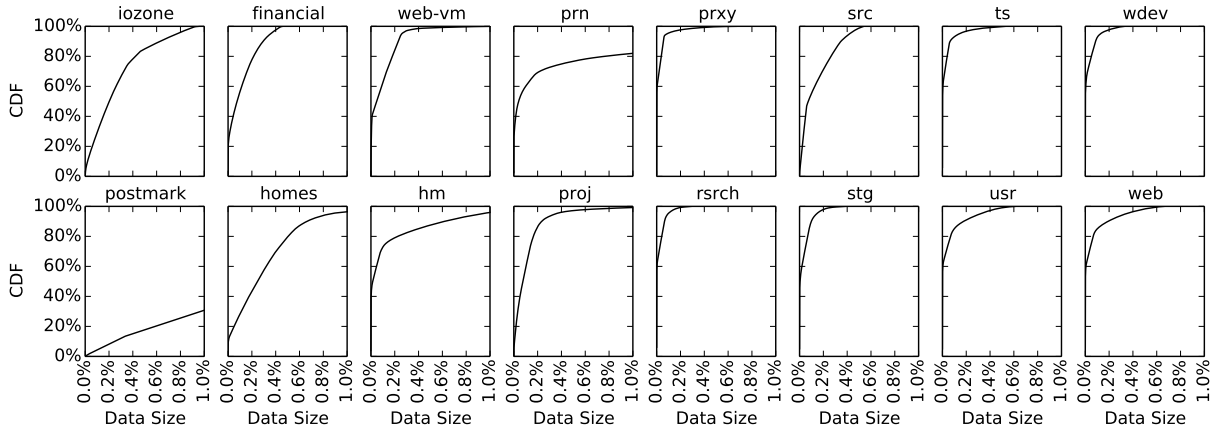


Figure 4.3: Cumulative distribution function of writes to pages for 16 evaluated workload traces. Total data footprints for our workloads are 217.6GB, i.e., 1.0% on the x-axis represents 2.176GB of data.

In a typical flash device, page allocation policies are oblivious to the frequency of writes, resulting in blocks that contain a random distribution of interspersed write-hot and write-cold pages. We find that such obliviousness to the program write patterns forces several of our "general" flash management algorithms to be inefficient. One such example is refresh, where the increased number of program/erase operations greatly limits the potential endurance gains that can be achieved. Refreshes are only necessary to maintain the integrity of data that has not yet been overwritten, as a new write operation *naturally* refreshes the data by placing the page into a new block. In other words, if a page is written to often enough (i.e., at a higher frequency than the refresh rate), any refresh operation to that page will be *redundant*.

Our goal is to enable such a mechanism that can eliminate refresh operations to write-hot pages. Unfortunately, remapping-based refresh operations are performed at a *block granularity*, which is much coarser than page granularity, as flash devices only provide a block-granularity

erase mechanism. Therefore, if at least one page within the block is write-cold, the *whole block* must be refreshed, foregoing any potential P/E cycle savings from skipping the refresh operation. As the conventional page allocation algorithm is oblivious to write-hotness, there is a very low probability that a block contains *only* write-hot pages.

Unless we change the page allocation policy, it is impractical to simply modify the refresh mechanism to skip refreshes for blocks containing only write-hot pages. If flash management policies were made aware of the write-hotness of a page, we could group write-hot pages together in a block such that the entire block does not require a refresh operation. This would allow us to efficiently skip refreshes to a much larger number of flash blocks that are formed as such. In addition, since write-cold pages would be grouped together into blocks, we could also reduce wear-out on these blocks through more efficient garbage collection. As write-cold pages are rarely overwritten, all of the pages within a write-cold block are more likely to remain valid, requiring much less frequent compaction. Our goal for WARM, our proposed management policy, is to *physically separate write-hot pages from write-cold pages into disjoint sets of flash blocks*, so that we can significantly improve flash lifetime.

4.2 Mechanism

In this section, we introduce WARM, our proposed write-hotness-aware flash memory retention management policy. The first key idea of WARM is to effectively partition pages stored in flash into two groups based on the write frequency of the pages. The second key idea of WARM is to apply different management policies to the two different groups of pages/blocks. We first discuss a novel, lightweight approach to dynamically identifying and partitioning write-hot versus write-cold pages (Section 4.2.1). We then describe how WARM optimizes flash management policies, such as garbage collection and wear-leveling, in a partitioned flash memory, and show how WARM integrates with a refresh mechanism to provide further flash lifetime improvements (Section 4.2.2). We discuss the hardware overhead required to implement WARM (Section 4.2.3). We show in Section 4.4 that WARM is effective at delivering significant flash lifetime improvements (by an average of 3.24× over a conventional management policy without refresh), and can do so with a minimal performance overhead (averaging 1.3%).

4.2.1 Partitioning Data Using Write-Hotness

Identifying Write-Hot and Write-Cold Data

Figure 4.4 illustrates the high-level concept of our write-hot data identification mechanism. We maintain two *virtual queues*, one for write-hot data and another for write-cold data, which order all of the hot and cold data, respectively, by the time of the last write. The purpose of the virtual queues is to partition write-hot and write-cold data in a space-efficient way. The partitioning mechanism provides methods of promoting data from the cold virtual queue to the hot virtual queue, and for demoting data from the hot virtual queue to the cold virtual queue. The promotion and demotion decisions are made such that write-hot pages are quickly identified (after two writes in quick succession to the page), and write-cold pages are seldom misidentified as write-hot pages

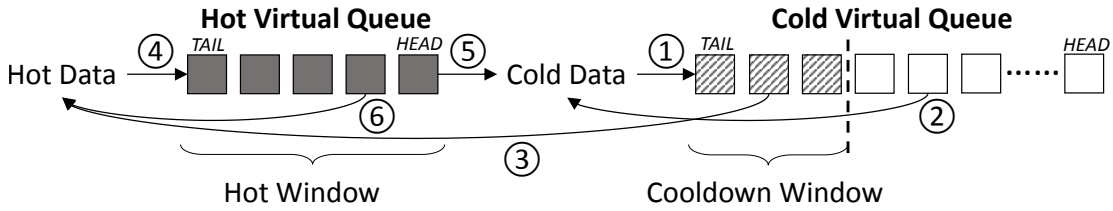


Figure 4.4: Write-hot data identification algorithm using two virtual queues and monitoring windows.

(and are quickly demoted if they are). Note that the cold virtual queue is divided into two parts, with the part closer to the tail known as the *cooldown window*. The purpose of the cooldown window is to identify those pages that are most recently written to. The pages in the cooldown window are the only ones that can be immediately promoted to the hot virtual queue (as soon as they receive a write request). We walk through examples for both of these migration decisions.

Initially, all data is stored in the cold virtual queue. Any data stored in the cold virtual queue is defined to be *cold*. When data (which we call Page C) is first identified as cold, a corresponding queue entry is pushed into the tail of the cold virtual queue (①). This entry progresses forward in the queue as other cold data is written. If Page C is written to again after it leaves the cooldown window (②), then its queue entry will be removed from the cold virtual queue and reinserted at the queue tail (①). This allows the queue to maintain ordering based on the time of the most recent write to each page.

If a cold page starts to become hot (i.e., it starts being written to frequently), a *cooldown window* at the tail end of the cold virtual queue provides these pages with a chance to be promoted into the hot virtual queue. The cooldown window monitors the most recently inserted (i.e., most recently written) cold data. Let us assume that Page C has just been inserted into the tail of the cold virtual queue (①). If Page C is written to again while it is still within the cooldown window, it will be immediately promoted to the hot virtual queue (③). If, on the other hand, Page C is not written to again, then Page C will eventually be pushed out of the cooldown window portion of the cold virtual queue, at which point Page C is determined to be *cold*. Requiring a two-step promotion process from cold to hot (with the use of a cooldown window) allows us to avoid incorrectly promoting cold pages due to infrequent writes. This is important for two reasons: (1) hot storage capacity is limited, and (2) promoted pages will not be refreshed, which for cold pages could result in data loss. With our two-step approach, if Page C is cold and is written to only once, it will remain in the cold queue, though it will be moved into the cooldown window (②) to be monitored for subsequent write activity.

Any data stored in the hot virtual queue is identified as *hot*. Newly-identified hot data, which we call Page H, is inserted into the tail of the hot virtual queue (④). The hot virtual queue length is maximally bounded by a *hot window* size to ensure that the most recent writes to all hot data pages were performed within a given time period. (We discuss how this window is sized in Section 4.2.1.) The assumption here is that infrequently-written pages in the hot virtual queue will eventually progress to the head of the queue (⑤). If the entry for Page H in the hot virtual queue reaches the head of the queue and must now be evicted, we demote Page H into the cooldown window of the cold virtual queue (①), and move the page out of the hot virtual queue.

In contrast, a write to a page in the hot virtual queue simply moves that page to the tail of the hot virtual queue (⑥).

Partitioning the Flash Device

Figure 4.5 shows how we apply the identification mechanism from Section 4.2.1 to perform physical page partitioning inside flash, with labels that correspond to the actions from Figure 4.4. We first separate all of the flash blocks into two *allocation pools*, one for hot data and another for cold data. The *hot pool* contains enough blocks to store every page in the hot virtual queue (whose sizing is described in Section 4.2.1), as well as some extra blocks to tolerate management overhead (e.g., erasing on garbage collection). The *cold pool* contains all of the remaining flash blocks. Note that blocks can be moved between the two pools when the queues are resized.

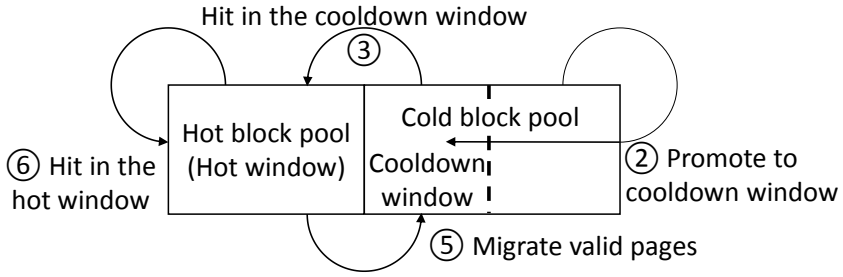


Figure 4.5: Write-hotness aware retention management policy overview.

To simplify the hardware required to implement the virtual queues, we exploit the fact that pages are written sequentially into the hot pool blocks. Consecutive writes to hot pages will be placed in the same block, which means that a single block in the hot virtual queue will hold *all* of the oldest pages. As a result, we can track the hot virtual queue at a block granularity instead of a page granularity, which allows us to significantly reduce the size of the hot virtual queue.

Tuning the Partition Boundary

Since the division between hot and cold data can be dependent on both application and phase characteristics, we need to provide a method for dynamically adjusting the size of our hot and cold pools periodically. Every block is allocated to one of the two pools, so any increase in the hot pool size will always be paired with a corresponding decrease in the cold pool size, and vice versa. Our dynamic sizing mechanism must ensure that: (1) the hot pool size is such that every page in the hot pool will be written to more frequently than the hot pool retention time (which is relaxed as the hot pool does not employ refresh), and (2) the lifetime of the blocks in the cold pool is maximized. To this end, we describe an algorithm that tunes the partitioning of blocks between the hot and cold pools.

The partitioning algorithm starts by setting an upper bound for the hot window, to ensure that every page in the window will be written to at a greater rate than the fixed hot pool retention time. Recall that the hot pool retention time is relaxed to provide greater endurance (Section 4.1.1). We estimate this size by collecting the number of writes to the hot pool, to find the average write frequency and estimate the time it takes to fill the hot window. We compare the time to fill the

window to the hot pool retention time, and if the fill time exceeds the retention time, we shrink the hot pool size to reduce the required fill time. This hot pool size determines the initial partition boundary between the hot pool and the cold pool.

We then tune this partition boundary to maximize the lifetime of the cold pool, since we do not relax retention time for the blocks in the cold pool. Assuming that wear-leveling evenly distributes the page writes within the cold pool, we can use the *endurance capacity* metric (i.e., the total number of writes the cold pool can service), which is the product of the remaining endurance of a block[1] and the cold pool size, to estimate the lifetime of blocks in the cold pool:

$$Endurance\,Capacity = Remaining\,Endurance \times Cold\,Pool\,Size \tag{4.1}$$

$$Lifetime = \frac{Endurance\,Capacity}{Cold\,Write\,Frequency} \propto \frac{Cold\,Pool\,Size}{Cold\,Write\,Frequency} \tag{4.2}$$

We divide the *endurance capacity* by the cold write frequency (writes per day) to determine the number of days remaining before the cold pool is worn out. We use hill climbing to find the partition boundary at which the cold pool size maximizes the flash lifetime. The cold write frequency is dependent on cold pool size, because as the cold pool size increases, the hot pool size correspondingly shrinks, shifting writes of higher frequency into the cold pool.

Finally, once the partition boundary converges to obtain the maximum lifetime, we must adjust what portion of the cold pool belongs in the cooldown window. We size this window to minimize the ping-ponging of requests between the hot and cold pools. For this, we want to maximize the number of hot virtual queue hits (⑥ in Figure 4.4), while minimizing the number of requests evicted from the hot window (⑤ in Figure 4.4). We maintain a counter of each of these events, and then use hill climbing on the cooldown window size to maximize the utility function $Utility = (⑥ - ⑤)$.

In our work, we limit the hot pool size to the number of over-provisioned blocks within the flash device (i.e., the extra blocks beyond the visible capacity of the device). While the hot pages are expected to represent only a small portion of the total flash capacity (see Section 4.1.3), there may be rare cases where the size limit prevents the hot pool from holding all of the hot data (i.e., the hot pool is significantly undersized). In such a case, some less-hot pages are forced to reside in the cold pool, and lose the benefits of WARM (i.e., endurance improvements from relaxed retention times). WARM will not, however, incur any further write overhead from keeping the less-hot pages in the cold pool. For example, the dynamic sizing of the cooldown window prevents the less-hot pages from going back and forth between the hot and cold pools.

4.2.2 Flash Management Policies

WARM partitions all of the blocks in a flash device into two pools, storing write-hot data in the blocks belonging to the *hot pool*, and storing write-cold data in the blocks belonging to the *cold pool*. Because of the different degrees of write-hotness of the data in each pool, WARM also applies different management policies (i.e., refresh, garbage collection, and wear-leveling) to each pool, to best extend their lifetime. We next describe these management policies for each pool, both when WARM is applied alone and when WARM is applied along with refresh.

[1]Due to wear-leveling, the remaining endurance (i.e., the number of P/E operations that can still be performed on the block) is the same across all of the blocks.

WARM-Only Management

WARM relaxes the internal retention time of only the blocks in the hot pool, without requiring a refresh mechanism for the hot pool. Within the cold pool, WARM applies conventional garbage collection (i.e., finding the block with the fewest valid pages to minimize unnecessary data movement) and wear-leveling policies. Since the flash blocks in the cold pool contain data with much lower write frequencies, they (1) consume a smaller number of P/E cycles, and (2) experience much lower fragmentation (which only occurs when a page is updated), thus reducing garbage collection activities. As such, the lifetime of blocks in the cold pool increases even when conventional management policies are applied.

Within the hot pool, WARM applies simple, in-order garbage collection (i.e., finding the oldest block) and no wear-leveling policies. WARM performs writes to hot pool blocks in *block order* (i.e., it starts on the block with the lowest ID number, and then advances to the block with the next lowest ID number) to maintain a sequential ordering by write time. Writing pages in block order enables garbage collection in the hot pool to also be performed in block order. Due to the higher write frequency in the hot pool, all data in the hot pool is valid for a shorter amount of time. Most of the pages in the oldest block are already invalid when the block is garbage collected, increasing garbage collection efficiency. Since both writing and garbage collection are performed in block order, each of the blocks will be *naturally* wear-leveled, as they will all incur the same number of P/E cycles. Thus, we do not need to apply any additional wear-leveling policy.

Combining WARM with Refresh

WARM can also be used in conjunction with a refresh mechanism to reap additional endurance benefits. WARM, on its own, can significantly extend the lifetime of a flash device by enabling retention time relaxation on only the write-hot pages. However, these benefits are limited, as the cold pool blocks will eventually exhaust their endurance at the original internal retention time. (Recall from Figure 4.1 that endurance decreases significantly as the selected internal retention time increases.) While WARM cannot enable retention time relaxation on the cold pool blocks due to infrequent writes to such blocks, a refresh mechanism can enable the relaxation, greatly extending the endurance of the cold pool blocks. WARM still provides benefits over a refresh mechanism for the hot pool blocks, since it avoids unnecessary write operations that refresh operations would incur.

When WARM and refresh are combined, we split the lifetime of the flash device into two phases. The flash device starts in the *pre-refresh phase*, during which the same management policies as WARM-only are applied. Note that during this phase, internal retention time is only relaxed for the hot pool blocks. Once the endurance *at the original retention time* is exhausted, we enter the *refresh phase*, during which the same management policies as WARM-only are applied *and* a refresh policy (such as FCR [22]) is applied to the cold pool to avoid data loss. During this phase, the retention time is relaxed for all blocks. Note that during both phases, the internal retention time for hot pool blocks is *always* relaxed *without* the need for a refresh policy.

During the refresh phase, WARM also performs global wear-leveling to prevent the hot pool from being prematurely worn out. The global wear-leveling policy rotates the *entire* hot pool to a

new set of physical flash blocks (which were previously part of the cold pool) every 1K hot block P/E cycles. Over time, this rotation will use all of the flash blocks in the device for the hot pool for one 1K P/E cycle interval. Thus, WARM wears out all of the flash blocks equally despite the heterogeneity in write-frequency between the two pools.

4.2.3 Implementation and Overheads

The logic overhead of the proposed mechanism is minimal. Thanks to the simplicity of the write-hot data identification algorithm, WARM can be integrated within an existing FTL, allowing it to be implemented in the flash controller that already exists in modern flash drives.

For our dynamic window tuning mechanism, four 32-bit counters are required. Two counters track the number of writes to the hot and cold pools. A third counter tracks the number of hot virtual queue write hits (⑥ in Figure 4.4). The fourth counter tracks the number of pages moved from the hot virtual queue into the cooldown window (⑤ in Figure 4.4).

The memory and storage overheads for the proposed mechanism are small. Recall that the cooldown window can remain relatively small. We need to store data that tracks which blocks belong to the cooldown window, and which blocks belong to the hot pool. From our evaluation, we find that a 128-block maximum cooldown window size is sufficient. This requires us to store the block ID of 128 blocks, for a storage overhead of 128×8B=1KB. As the blocks belonging to the hot pool are written to in order of block ID, we require even lower overhead for them. We allocate a contiguous series of block IDs to the hot pool, reducing the tracking overhead to four registers totaling 32B: the starting ID of the series, the current size of the pool, a pointer to the most recently written block (i.e., the block at the tail of the hot virtual queue), and a pointer to the oldest block yet to be erased (i.e., the block at the head of the hot virtual queue). All this information can be buffered inside the memory of the flash controller in order to accelerate write operations.

While WARM saves a significant amount of unnecessary refreshes in the hot data pool, the proposed mechanism has the potential to indirectly generate extra write operations that consume some endurance cycles. First, WARM generates extra write operations when demoting a hot page to the cold data pool (⑤). Second, partitioning flash blocks into two allocation pools can sometimes increase garbage collection activities. This is because one of the pools may have a smaller number of blocks available in the free list, requiring more frequent invocation of garbage collection. All of these overheads are accounted for in our evaluation in Section 4.4, and our results factor in all additional writes. As we show in Section 4.4, WARM is designed to minimize these overheads such that lifetime improvements are not overshadowed, and the resulting impact on response time is minimal.

4.3 Methodology

We use DiskSim-4.0 [17] with SSD extensions [2] to evaluate WARM. Table 4.1 lists the parameters of our simulated NAND flash-based SSD. The latencies (the first four rows of the table) are from real NAND flash chip measurements [106]. The sizes (rows 5–8) represent a modern commercial NAND flash specification [1]. Flash endurance and refresh period are measured from

real NAND flash devices [22, 25].

Table 4.1: Parameters of the simulated flash-based SSD.

Parameter	**Value**
Page read to register latency	25μs
Page write from register latency	200μs
Block erase latency	1.5ms
Data bus latency	50μs
Page/block size	8KB/1MB
Die/package size	8GB/64GB
Total storage capacity (inclòver-provisioning)	256GB
Over-provisioning	15%
Endurance for 3-year retention time (P/E cycles)	3,000
Endurance for 3-day retention time (P/E cycles)	150,000

We run each simulation with I/O traces collected from a wide range of real workloads with different use cases [158, 239, 321]. We also select two popular synthetic file system benchmarks to stress our mechanism with higher write rate applications [135, 246]. Table 4.2 lists the name, source, length, and description of each trace. To compute the lifetime of each configuration, we assume the trace is repeated until the flash drive fails. We fill all the usable space of the flash drive with data, to mimic worst-case usage conditions and to trigger garbage collection activities within the trace duration. Similar to the approach employed in prior work [21, 22, 25], the overall flash lifetime is derived using the average write frequency of one run, which consists of writes generated by the trace and by garbage collection, as well as by refresh operations during the refresh phase. We use this methodology since it is impossible to simulate multi-year-long traces that drain the flash lifetime.

4.4 Evaluations

In this section, we evaluate and compare *six* configurations:

- `Baseline` does not include WARM or refresh, and uses conventional garbage collection and wear-leveling policies, as described in Section 4.1.
- `WARM` uses the proposed write-hotness aware retention management policy that we described in Section 4.2.
- `FCR` adds a remapping-based refresh mechanism to `Baseline`. Our refresh mechanism is similar to the remapping-based FCR described in prior work [22, 25], but refresh is *not* performed in the pre-refresh phase (see Section 4.2.2) to reduce unnecessary overhead. During the refresh phase (see Section 4.2.2), `FCR` refreshes all valid blocks every three days, which yields the best endurance improvement.
- `WARM+FCR` uses write-hotness aware retention management alongside 3-day refresh (Section 4.2.2) to achieve maximum lifetime.

Table 4.2: Source and description of simulated traces.

Trace	Source	Length	Workload Description
	Synthetic Workloads		
iozone	IOzone [246]	16 min	File system benchmark
postmark	Postmark [135]	8.3 min	File system benchmark
	Real-World Workloads		
financial	UMass [321]	1 day	Online transaction processing
homes	FIU [158]	21 days	Research group activities
web-vm	FIU [158]	21 days	Web mail proxy server
hm	MSR [239]	7 days	Hardware monitoring
prn	MSR [239]	7 days	Print server
proj	MSR [239]	7 days	Project directories
prxy	MSR [239]	7 days	Firewall/web proxy
rsrch	MSR [239]	7 days	Research projects
src	MSR [239]	7 days	Source control
stg	MSR [239]	7 days	Web staging
ts	MSR [239]	7 days	Terminal server
usr	MSR [239]	7 days	User home directories
wdev	MSR [239]	7 days	Test web server
web	MSR [239]	7 days	Web/SQL server

- `ARFCR` adds the ability to progressively increase refresh frequency on top of the remapping-based refresh mechanism (similar to adaptive-rate FCR [22, 25]). The refresh frequency increases as the retention capabilities of the flash memory decrease, in order to minimize the overhead of write-hotness-oblivious refresh.
- `WARM+ARFCR` adds WARM alongside the adaptive-rate refresh mechanism.

To provide insights into our results, we first show the hot pool sizes and the cooldown window sizes as determined by WARM for each of the configurations (Section 4.4.1). We then use four metrics to show the benefits and costs associated with our mechanism:

- We evaluate all configurations in terms of *overall lifetime* (Section 4.4.2).
- We evaluate the gain in *endurance capacity*, the aggregate number of write requests that the flash device can endure *across all pages*, for `WARM` with respect to `Baseline` (Section 4.4.3). We use this metric as an indicator of how many additional writes we can sustain to the flash device with our mechanism.
- We evaluate and break down the *total number of writes* consumed by `FCR` and `WARM+FCR` during the refresh phase, to demonstrate how our mechanism reduces the write overhead of retention time relaxation (Section 4.4.4).
- We evaluate the *average response time*, the mean latency for the flash device to service a host request, for both `Baseline` and `WARM` to demonstrate the performance overhead of using WARM (Section 4.4.5).

Finally, we show sensitivity studies on flash memory over-provisioning and the refresh rate

(Section 4.4.6).

4.4.1 Hot Pool and Cooldown Window Sizes

Table 4.3 lists the hot pool and the cooldown window sizes learned by WARM for each of our WARM-based configurations. To allow WARM to quickly adapt to different workload behaviors, we set the smallest step size by which the hot pool size can change to 2% of the total flash drive capacity, and we restrict the cooldown window sizes to power-of-two block counts. For WARM+ARFCR, as the refresh frequency of the flash cells increases (going to the right in Table 4.3), the hot pool size generally reduces. This is because WARM automatically selects a smaller hot pool size to ensure that the data in the hot pool has a high enough write intensity to skip refreshes. Naturally, as the internal retention time of a cell decreases, previously write-hot pages with a write rate slower than the new retention time no longer qualify as hot, thereby reducing the number of pages that need to be maintained in the hot pool. WARM adaptively selects different hot pool sizes based on the fraction of write-hot data in each particular workload. Similarly, WARM intelligently selects the best cooldown window size for each workload, such that it minimizes the number of cold pages that are misidentified as hot and considered for promotion to the hot pool. As such, our analysis indicates that WARM can intelligently and adaptively adjust the hot pool size and the cooldown window size to achieve maximum lifetime.

Table 4.3: Hot pool and cooldown window sizes as set dynamically by WARM. H%: Hot pool size as a percentage of total flash drive capacity. CW: Cooldown window size in number of blocks.

Trace	**WARM**		**WARM +FCR**		**WARM+ARFCR**					
					3-month		**3-week**		**3-day**	
	H%	**CW**	**H%**	**CW**	**H%**	**CW**	**H%**	**CW**	**H%**	**CW**
iozone	10	8	10	8	10	8	10	8	10	8
postmark	2	128	4	128	4	128	4	128	4	128
financial	10	4	10	4	10	4	10	4	10	4
homes	10	128	4	32	10	4	10	4	4	32
web-vm	10	128	10	32	10	4	10	4	10	32
hm	10	128	10	128	10	32	10	32	10	128
prn	10	128	10	128	8	4	10	128	10	128
proj	10	4	10	4	10	4	10	4	10	4
prxy	10	4	10	4	10	4	10	4	10	4
rsrch	6	128	6	128	10	4	10	4	6	128
src	10	128	8	128	10	32	10	32	8	128
stg	10	128	8	128	10	4	10	4	8	128
ts	10	128	6	128	10	4	10	4	6	128
usr	6	128	6	128	10	4	10	4	6	128
wdev	6	128	4	128	10	128	10	128	4	128
web	6	128	6	128	10	4	10	4	6	128

4.4.2 Lifetime Improvement

Figure 4.6 shows the lifetime in days (using a logarithmic scale) for all six of our evaluated configurations. Figure 4.7a shows the lifetime improvement of WARM when normalized to the lifetime of Baseline. The mean lifetime improvement for WARM across all of our workloads is 3.24× over Baseline. In addition, WARM+FCR improves the mean lifetime over FCR alone by 1.30× (Figure 4.7b), leading to a mean improvement for combined WARM and FCR over Baseline of 10.4× (as opposed to 8.0× with FCR alone). WARM+ARFCR improves the mean lifetime over ARFCR by 1.21× (Figure 4.7c), leading to a mean improvement for combined WARM and ARFCR over Baseline of 12.9× (as opposed to 10.7× with ARFCR alone). Even for our worst performing workload, postmark, in which the amount of hot data and the fraction of writes due to refresh are very low (as discussed in Sections 4.4.3 and 4.4.4), the overall lifetime improves by 8% when WARM is applied without refresh, and remains unaffected with respect to FCR when WARM+FCR is applied. We conclude that WARM can adjust to workload behavior and effectively improve overall flash lifetime, either when used on its own or when used together with a refresh mechanism, without adverse impacts.

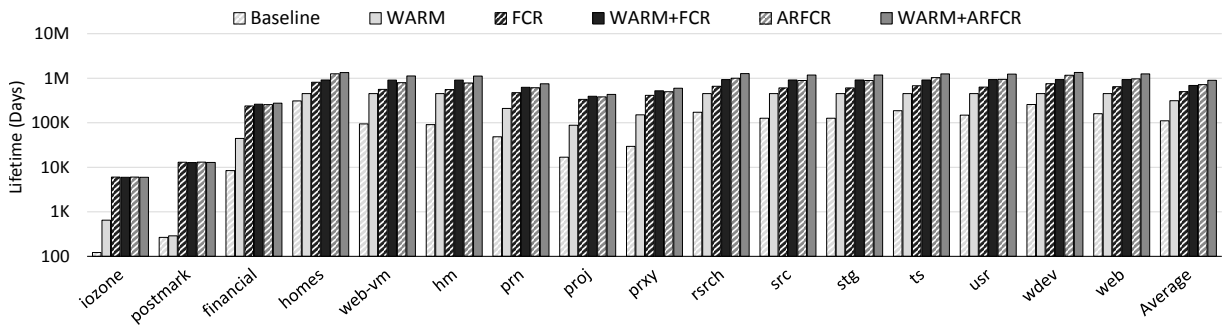
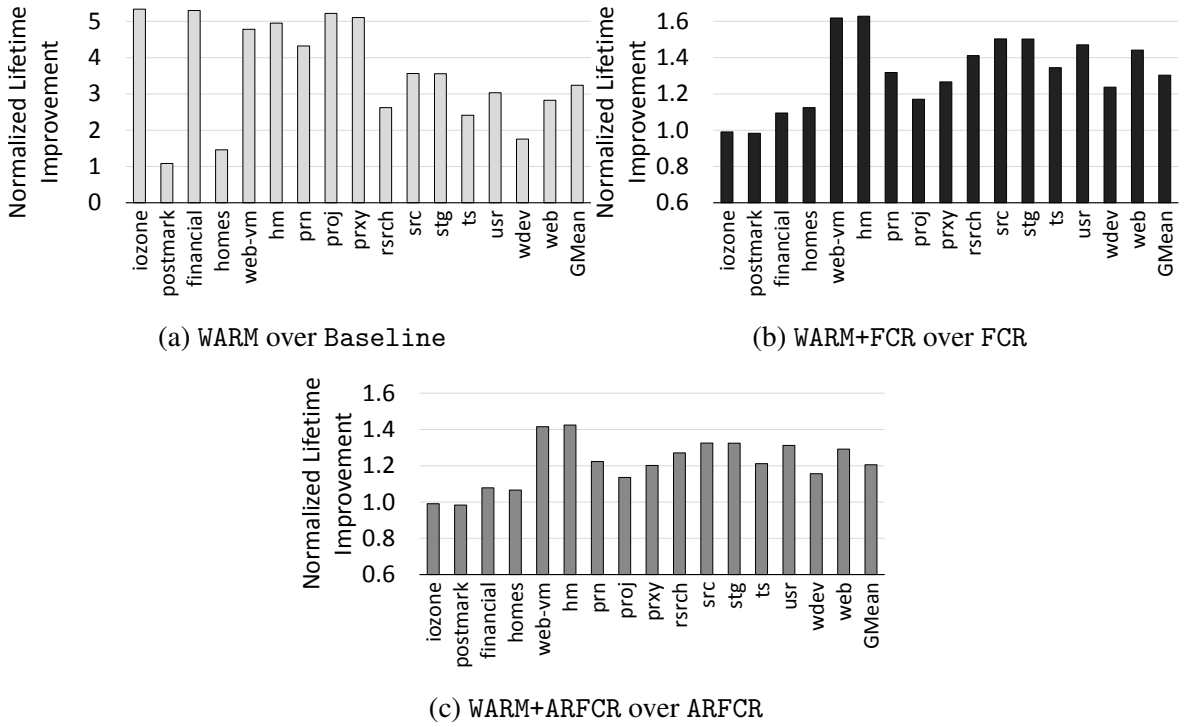


Figure 4.6: Absolute flash memory lifetime for Baseline, WARM, FCR, WARM+FCR, ARFCR, and WARM+ARFCR configurations. Note that the y-axis uses a log scale.

4.4.3 Improvement in Endurance Capacity

Figure 4.8 plots the normalized *endurance capacity* of WARM for each workload split up by the endurance for both the hot and cold data pools. The endurance capacity is defined as the total number of write operations the entire flash device can sustain before wear-out. On average, WARM improves the total endurance capacity by 3.6× over Baseline. Note that the endurance capacity varies across different workloads, in relation to the number of hot writes that can be identified by the mechanism. For example, postmark contains only a limited amount of write-hot data (as is shown in Figure 4.3), which results in only minor endurance capacity improvement (8%). Unlike the other workloads, the majority of the endurance capacity for postmark remains within the cold pool, as the workload exhibits very low write locality.



(a) `WARM` over `Baseline`

(b) `WARM+FCR` over `FCR`

(c) `WARM+ARFCR` over `ARFCR`

Figure 4.7: Normalized flash memory lifetime improvement when WARM is applied on top of `Baseline`, `FCR`, and `ARFCR` configurations.

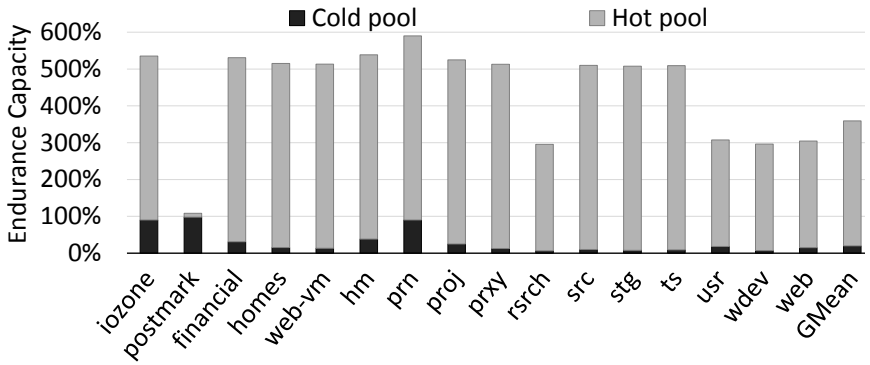


Figure 4.8: `WARM` endurance capacity, normalized to `Baseline`.

In contrast, the endurance capacity for all of our other workloads mainly comes from the hot pool, despite the size of the hot pool being significantly smaller than that of the cold pool. WARM in essence "converts" blocks from normal internal retention time (those in the cold pool) into relaxed internal retention time (hot pool) for the write-hot portion of data. Blocks with a relaxed retention time can tolerate a much larger number of writes (as shown in Figure 4.1). As Figure 4.3 shows, the vast majority of overall writes are to a small fraction of pages that are write-hot. This allows WARM to improve the overall flash endurance capacity by using a small number of blocks with a relaxed retention time to house the write-hot pages. We conclude that WARM can effectively improve endurance capacity even when applied on its own.

4.4.4 Reduction of Refresh Operations

Figure 4.9 breaks down the percentage of endurance (P/E cycles) used for the host's write requests, for management operations, and for refresh requests during the refresh phase. Two bars are shown side by side for each application. The first bar shows the number of total writes for FCR, normalized to 100%. The second bar shows a similar breakdown for WARM+FCR, *normalized to the number of writes for* FCR. Although the two synthetic workloads (iozone and postmark) do not show much reduction in total write frequency (because host writes dominate their flash endurance usage, as shown in Figure 4.2), the number of writes across all sixteen of our workloads is reduced by an average of 5.3%.

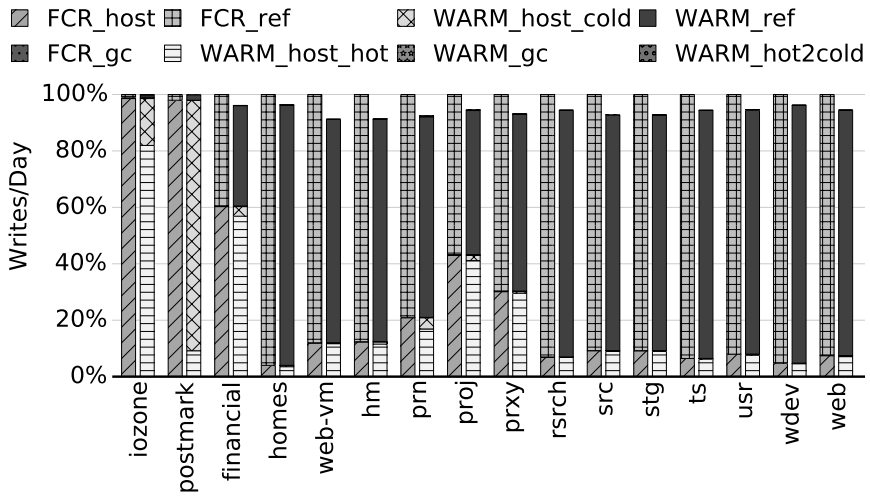


Figure 4.9: Flash writes for FCR (left bar) and WARM+FCR (right bar), broken down into host writes to the hot/cold pool (host_hot/host_cold), garbage collection writes (gc), refresh writes (ref), and writes generated by WARM for migrations from the hot pool to the cold pool (hot2cold).

From the breakdown of the write requests, we can see that the reduction in write count mainly comes from the decreased number of refresh requests after applying WARM. In contrast, the additional overhead in WARM due to migrating data from the hot pool to the cold pool is minimal (shown as WARM_hot2cold). This suggests that the write locality behavior within many of the

hot pool pages lasts throughout the lifetime of the application, and thus these pages do not need to be evicted from the hot pool.[2], as explained in Section 4.2.1. We conclude that WARM+FCR, by providing refresh-free retention time relaxation for hot data, can reduce a significant fraction of unnecessary refresh writes, and that WARM+FCR can utilize the flash endurance more effectively during the refresh phase.

4.4.5 Impact on Performance

As we discussed in Section 4.2.3, WARM has the potential to generate additional write operations. First, when a page is demoted from the hot pool to the cold pool (which happens when another page is being promoted into the hot pool), an extra write will be required to move the page into a block in the cold pool.[3] Second, as one of the pools may have fewer blocks available in its free list (which is dependent on how our partitioning algorithm splits up the flash blocks into the hot and cold pools), garbage collection may need to be invoked more frequently when a new page is required. To understand the impact of these additional writes, we evaluate how WARM affects the average response time of the FTL.

Figure 4.10 shows the average response time for WARM, normalized to the Baseline response time. Across all of our workloads, the average performance reduces by only 1.3%. Even in the worst case (homes), WARM only has a performance penalty of 5.8% over Baseline. The relatively significant overhead for homes is due to the write-hot portion of its data changing frequently over time within the trace. This is likely because the user operates on different files, which effectively shifts the write locality to an entirely different set of pages whenever a new file is operated on. The shifting of the write-hot page set evicts *all* of the write-hot pages from the hot pool, which as we stated above incurs several additional writes.

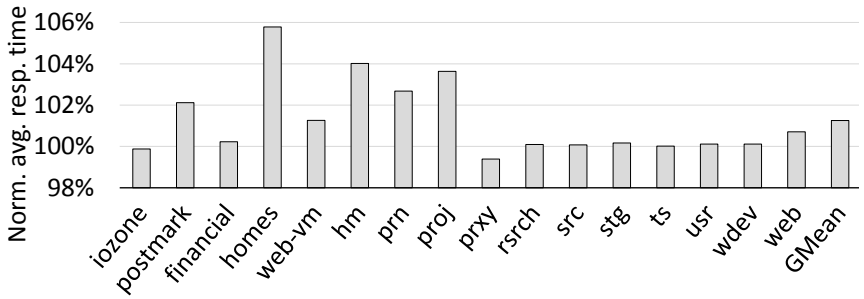


Figure 4.10: WARM average response time, normalized to Baseline.

For most of the workloads, any performance degradation is negligible (<2%), and is a result of the increased garbage collection that occurs in the hot pool due to its small free list size. For some other workloads, such as prxy, we find that the performance actually *improves* slightly with WARM, because of the reduction in data movement induced by garbage collection. This savings

[2]Migrations from the cold pool to the hot pool are not broken down separately, as such migrations are performed during the host write request itself and do not incur additional writes

[3]In contrast, promoting a page from the cold pool to the hot pool does not incur additional writes, as promotion only occurs when that page is being written. Since a write was needed regardless, the promotion is free.

is thanks to grouping write-cold data together, which greatly lessens the degree of fragmentation within the majority of the flash blocks (those within the cold pool). Overall, we conclude that across all of our workloads, the performance penalty of using WARM is minimal.

4.4.6 Sensitivity Studies

Figure 4.11 compares the flash memory lifetime under different capacity over-provisioning assumptions. In high-end server-class flash drives, the amount of capacity over-provisioning is higher than that in consumer-class flash drives to provide an overall longer lifetime and higher reliability. In this figure, we evaluate the lifetime improvement of the same six configurations using 30% of the flash blocks for over-provisioning to represent a server-class flash drive (all other parameters from Table 4.1 remain the same). We also show the lifetime of the six configurations on a consumer-class flash drive with 15% over-provisioning (which we assumed in our evaluations until now). We show that the lifetime improvement of WARM become more significant as over-provisioning increases. The lifetime improvement delivered by WARM over Baseline, for example, increases to 4.1×, while the improvement of WARM+ARFCR over Baseline increases to 14.4×. We conclude that WARM delivers higher lifetime improvements as over-provisioning increases.

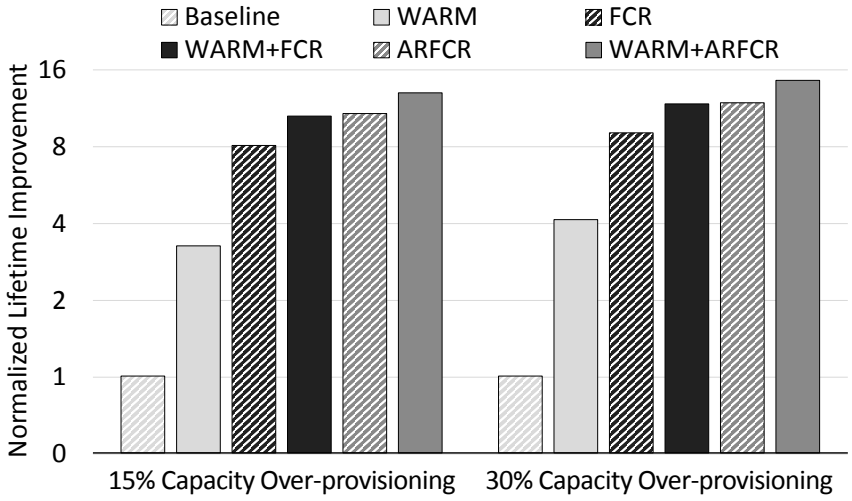


Figure 4.11: Flash memory lifetime improvement for WARM, FCR, WARM+FCR, ARFCR, and WARM+ARFCR configurations under different amounts of over-provisioning, normalized to the Baseline lifetime *for each over-provisioning amount*. Note that the y-axis uses a log scale.

Figure 4.12 compares the flash memory lifetime improvement for WARM+FCR over FCR under different refresh rate assumptions. Our evaluation has so far assumed a three-day refresh period for FCR. In this figure, we change this assumption to three-month and three-week refresh periods, and compare the corresponding lifetime improvement. As we see from this figure, the lifetime improvement delivered by WARM+FCR drops significantly as the refresh period becomes longer. This is because a smaller fraction of the endurance is consumed by refresh operations as the rate of refresh decreases (as shown in Figure 4.2), which is where our major savings come from.

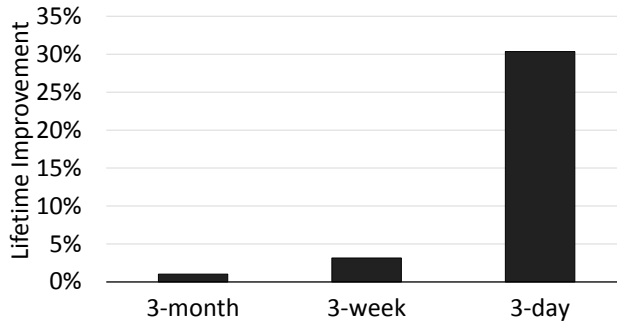
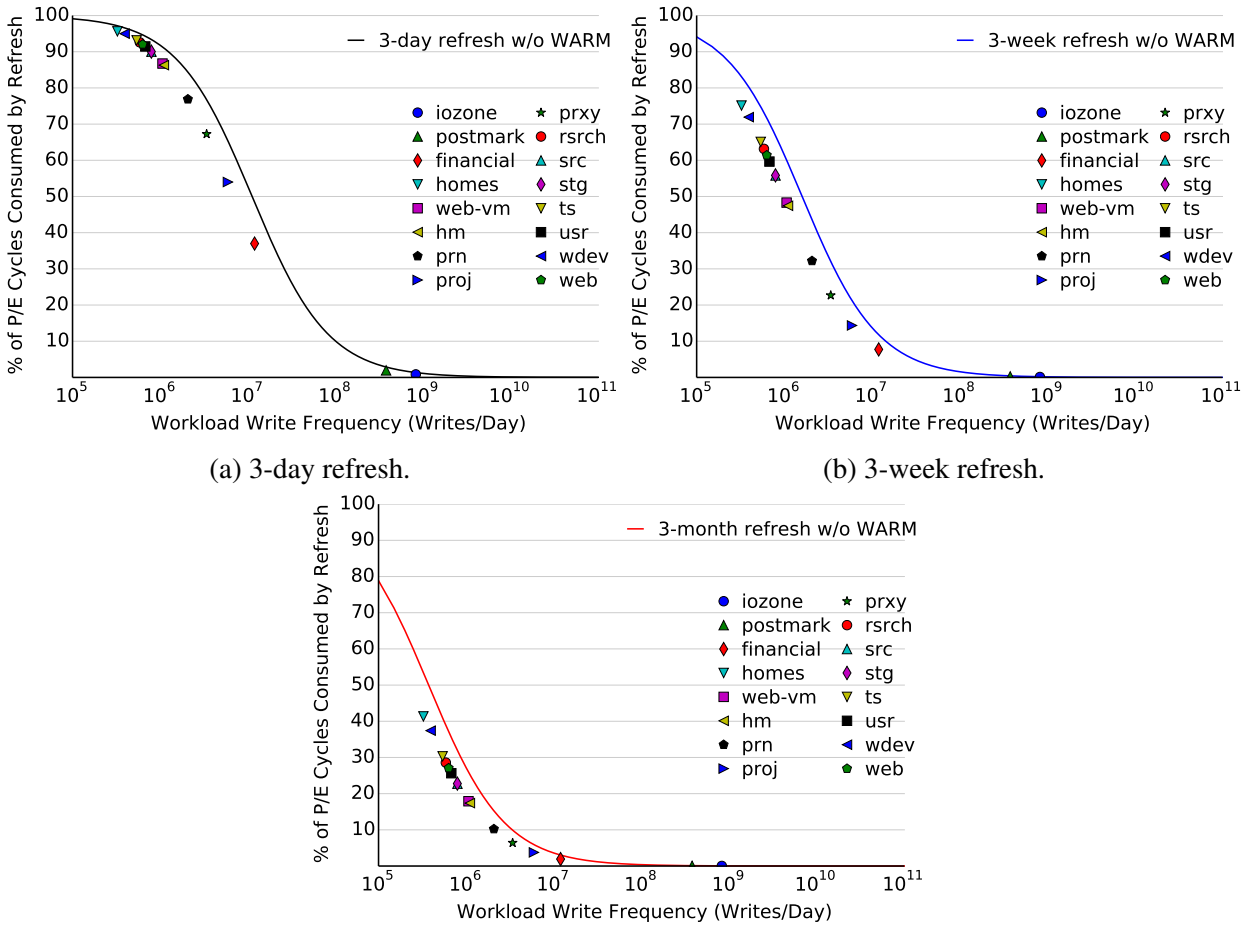


Figure 4.12: Flash memory lifetime improvements for WARM+FCR over FCR under different refresh rate assumptions.

Figure 4.13 illustrates how WARM+FCR reduces the fraction of P/E cycles consumed by refresh operations, over FCR only, as we sweep over longer refresh periods. Note that the x-axis in the figure uses a log scale. The solid lines in the figure illustrate the fraction of P/E cycles consumed by refresh for FCR only, as was shown in Figure 4.2. The figure shows that for as the refresh interval increases, WARM+FCR is effective at reducing the number of writes that are consumed by refresh, but that these make up a smaller portion of the total P/E cycles, hence the smaller improvements over FCR alone. As flash memory becomes denser and less reliable, we expect it to require *more frequent* refreshes in order to maintain a useful lifetime, at which point WARM can deliver greater improvements. We conclude that WARM+FCR delivers higher lifetime improvements as the refresh rate increases.



(a) 3-day refresh.

(b) 3-week refresh.

(c) 3-month refresh.

Figure 4.13: Fraction of P/E cycles consumed by refresh operations *after* applying WARM+FCR for a (a) 3-day, (b) 3-week, or (c) 3-month refresh period. Solid trend lines show the fraction consumed by FCR only, from Figure 4.2, for comparison. Note that the x-axis uses a log scale.

4.5 Limitations

WARM relies on the existence of write-hot data in many existing write-intensive workloads. If the workload is not write-intensive, or if all data in the workload is equally write-hot, WARM cannot identify any write-hot data and skip any refreshes. In these unlikely cases, WARM does not provide any benefit or penalty in flash lifetime.

4.6 Conclusion

In this chapter, we introduce WARM, a write-hotness aware retention management policy for NAND flash memory that is designed to extend its lifetime. We find that pages with different degrees of *write-hotness* have widely ranging retention time requirements. WARM allows us to relax the flash retention time for write-hot data without the need for refresh, by exploiting

the high write frequency of this data. On its own, WARM improves the lifetime of flash by an average of 3.24× over a conventionally-managed flash device, across a range of real I/O workload traces. When combined with refresh mechanisms, WARM eliminates redundant refresh operations to recently written data. In conjunction with an adaptive refresh mechanism, WARM extends the average flash lifetime by 12.9×, which represents a 1.21× increase over using the adaptive refresh mechanism alone. We conclude that WARM is an effective policy for improving overall flash lifetime with or without refresh mechanisms.

Chapter 5

Online Flash Channel Modeling and Its Applications

As we have introduced in Section 2.2, the threshold voltage value of a flash cell is used to represent the data that is stored within the cell. In Section 3.1, we have shown that the threshold voltage of the cell changes (i.e., shifts) as a result of many different types of circuit-level noise. Some cells' threshold voltages might shift enough to cross over to neighboring voltage windows. The value of such a cell would be misread on a flash memory channel read, causing an error. Thus, *knowing how the threshold voltage distribution shifts within the flash controller can help to mitigate these errors*.

In this chapter, we introduce a new framework that exploits the unused computing resources in the flash controller to enable greater device awareness by exploiting *an online flash channel model*. Our goal is to build an *accurate* and *easy-to-compute* model of the threshold voltage distribution of modern MLC NAND flash memory. This model must be practical to implement, as we intend to use it *online* to design flash controllers that can adapt to the changing NAND flash memory behavior. Our model can (1) *statically* determine the threshold voltage distribution at a given level of wear-out (i.e., a given P/E cycle count), and (2) *dynamically* predict how this threshold voltage distribution shifts over time as a result of the P/E cycling effect. The key idea is to learn this online flash channel model with low overhead and use this model in multiple components within the flash controller to help improve flash reliability.

Section 5.1 describes the motivation of developing an online flash channel model. To build our flash channel model, we first use the methodology described in Section 5.2 to characterize the threshold voltage distribution (i.e., the flash channel) under different P/E cycles using real 1X nm MLC NAND flash chips. Second, in Section 5.3, we construct a static threshold voltage distribution model that can fit the characterized distribution under any given P/E cycle count. Third, in Section 5.4, we construct a dynamic P/E cycling model that predicts how each parameter of the static distribution model changes after some number of further P/E cycles. Finally, in Section 5.5, we demonstrate several example use cases in the flash controller that utilize the complete model to enhance the performance and reliability of the NAND flash memory device. We conclude the contributions of our online flash channel model in Section 5.8.

5.1 Motivation

Having online information on the current threshold voltage distribution across all of the flash cells within a flash memory chip (i.e., the *static* distribution), as well as how this distribution changes over time (i.e., the *dynamic* distribution), is important to quantify errors and develop techniques to improve the reliability of the flash device. First, the static distribution can be used to determine the number of errors that would occur for any read reference voltage that is applied. This data can be used by the flash controller to select the read reference voltage that minimizes the error rate. Lowering the error rate increases the lifetime of the flash device, as it delays the time at which the number of errors becomes too large for the built-in ECC mechanism to successfully correct. Second, knowing how the dynamic distribution changes over time (i.e., as more writes are performed) is important, as it can guide flash controller mechanisms that adjust various flash parameters online (e.g., ECC strength [101, 188, 336], read reference voltages [27], pass-through voltage [35]) to increase the flash memory lifetime. Prior proposals to adjust these parameters (e.g., [27, 35]) rely on a trial-and-error approach to select parameters with low error rates, which can be inaccurate, high-latency, and suboptimal in terms of lifetime improvement. In both cases (static and dynamic), the threshold voltage distribution must be determined at runtime by the flash controller. Therefore, it is critical to design a *practical* and *low-complexity* mechanism to determine the distribution and the shifts in the distribution.

In order for the model to be useful to help flash controller algorithms, it should have certain properties. First, the model needs to be *accurate* for all threshold voltages and at all P/E cycles. This is because inaccurate information can lead a flash controller to make suboptimal decisions, hurting lifetime improvements. Second, the model needs to be *easy to compute*, because the flash controller has only limited computational resources. While we base our model off of a distribution that is easy to compute, we can further reduce online computation with the dynamic component of our model, by performing only a few online static characterizations of the threshold voltage distribution, and then using the simpler dynamic model to predict shifts in these initial characterizations at very low cost over P/E cycles.

5.2 Characterization Methodology

To build our model, we perform an experimental characterization of the threshold voltage distribution on real state-of-the-art 1X-nm (i.e., 15-19nm) MLC NAND flash chips. This characterization is essential to verify that the model we develop accurately captures the behavior of a real, modern device.

We collect experimental characterization data on the threshold voltage distribution using an FPGA-based NAND flash testing platform [20] with state-of-the-art 1X-nm MLC NAND flash chips. We use the read-retry technique [23, 27] (described in Section 3.2.4) to sweep *all possible* read reference voltages and determine the threshold voltage value for each cell. We program and erase these blocks to 11 different wear levels, up to 20K P/E cycles, using known pseudo-random data. The manufacturer-specified P/E cycle endurance for the tested flash chips is 3000 P/E cycles. All tests are performed at room temperature with a 5-second dwell time.[1]

[1] *Dwell time* is the time duration between an erase operation and the next program operation to the same flash

Figure 5.1 shows the threshold voltage distribution for each of the cell states. The read-retry capability on the MLC NAND flash memory chip allows us to fine-tune each read reference voltage (V_a, V_b, and V_c) to one of 101 different steps (a total of 303 read reference voltage steps, labeled as V_1 to V_{303} from left to right). Note that V_1 does not extend all the way to the lowest possible threshold voltage for the ER state, and V_{303} does not extend to the highest possible threshold voltage for the P3 state. In this chapter, we normalize the threshold voltage values such that the distance between most of the adjacent read reference voltage steps is one, as the exact values are proprietary information. The distances between steps V_{101} and V_{102} and between steps V_{202} and V_{203} are much larger than the typical distance between voltage steps,[2] as shown in Figure 5.1. As a result, the voltage step V_{303} has a normalized voltage value that is greater than 303. Overall, the threshold voltage range is divided by these read reference voltages into 304 bins, labeled as bin_0 to bin_{303}. Each flash cell can be classified into one of these bins based on the threshold voltage value read from the cell. If the read reference voltage is higher than the threshold voltage of the cell, the value read out from the flash device is 1, otherwise the value read out is 0. For a cell whose threshold voltage falls between two neighboring read reference voltages (V_k and V_{k+1}), the cell is placed into bin_k, as illustrated in Figure 5.1.

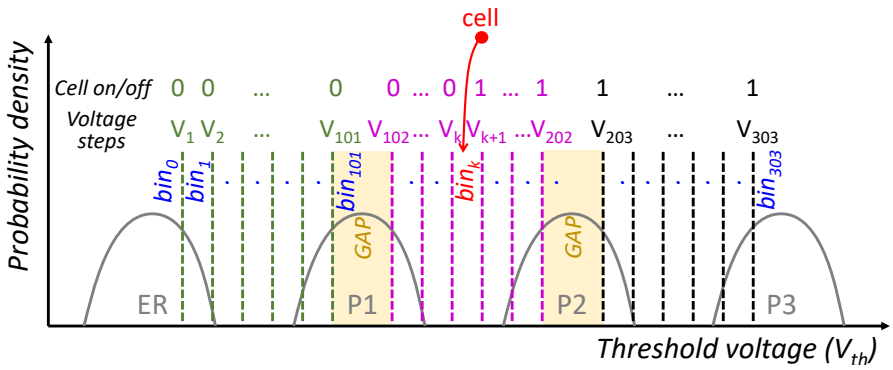


Figure 5.1: Methodology for finding the threshold voltage of an MLC NAND flash memory cell.

After classifying every cell using the methodology described above, we count the number of flash cells with state $X \in \{\text{ER, P1, P2, P3}\}$ in bin k as $H_k(X)$. Equation 5.1 shows how we then normalize the bin counts as the probability density of each bin, $P_k(X)$. Note that in our characterization, we assign each flash cell to the threshold voltage distribution of the *correct state* that it was originally programmed to, as we know the data value that we programmed. The characterized bin density can be viewed as a discretized version of the measured distribution, which our model is constructed to fit.

$$P_k(X) = \frac{H_k(X)}{\Sigma_{i=0}^{303} H_i(X)} \tag{5.1}$$

cell.

[2]Some flash vendors choose to provide fewer read reference voltages near the peak of the distribution of each state. This is because flash cells near the have threshold voltages far away from the default read reference voltage, and hence are less likely to have errors.

5.3 Static Distribution Model

We construct a *static* threshold voltage distribution model that can fit the characterized threshold voltage distribution well under any P/E cycle count, based on data collected using the methodology described in Section 5.2. Recall that this model needs to be (1) accurate for all threshold voltages and at any given P/E cycle, and (2) easy to evaluate within the flash controller. While a more complex model can satisfy the accuracy requirements, it can be difficult to compute the model on the fly given the limited computational resources in a flash controller. In this section, we first describe two state-of-the-art models, each of which meets only one of our two requirements. The first previously-proposed model [23], based on a *Gaussian distribution*, is simple and easy to compute, but is not accurate enough for raw bit error rate estimation (Section 5.3.1). The second previously-proposed model [260], based on a *normal-Laplace distribution*, is accurate, but requires significant computational resources, taking 10.7x the computation time of the Gaussian-based model (Section 5.3.2). We propose a new model, based on our modified version of the *Student's t-distribution* [300], which satisfies both of our requirements, maintaining the accuracy of the normal-Laplace-based model while requiring 4.41x less computation time (Section 5.3.3). Finally, we validate and compare the three models (Section 5.3.4).

5.3.1 Gaussian-based Model

The Gaussian-based model assumes that the threshold voltage distribution of each state follows a Gaussian (i.e., normal) distribution [23]. Equation 5.2 shows how the Gaussian-based model estimates the probability density for state X (i.e., ER, P1, P2, and P3) in each bin k, denoted as $G_k(X)$:

$$G_k(X) = GCDF(V_k, \mu_X, \sigma_X) - GCDF(V_{k-1}, \mu_X, \sigma_X) \tag{5.2}$$

The density $G_k(X)$ is calculated as the difference between the Gaussian cumulative distribution function ($GCDF$) of the bin's two boundaries, V_k and V_{k-1}. The Gaussian-based model has two variables for each state: μ_X is the mean of the distribution, and σ_X is the standard deviation of the distribution. In total, the Gaussian threshold voltage distribution model has eight parameters.

The intuition behind using a Gaussian distribution is twofold. First, the threshold voltage distribution is a result of physical noise and manufacturing process variation, which naturally follow a Gaussian distribution. During a program operation, the flash controller uses ISPP (see Section 2.2.4), iteratively increasing the threshold voltage until the desired threshold voltage level is achieved. Each programming step increases the threshold voltage of a cell by a small random amount. As programming subjects the cell to random physical noise, the threshold voltage distribution of each state naturally approximates a Gaussian distribution [23].

Second, the Gaussian-based model can be computed quickly, and is easily implementable in the flash controller hardware if we use a *z-table*, a lookup table that stores the *precomputed* cumulative distribution function of the standard Gaussian distribution. Equation 5.3 shows how the z-table simplifies the computation of $GCDF$. First, we calculate the z-scores $Z = \frac{V-\mu}{\sigma}$ for V_k and V_{k-1}. Then, we calculate $\Phi(Z)$, the precomputed cumulative distribution function of Z,

by looking up the z-score in the z-table. The two z-scores (one each for V_k and V_{k-1}) are then combined to get $G_k(X)$, using Equation 5.2.

$$GCDF(V, \mu, \sigma) = \Phi(Z) = z\text{-}table(Z) \tag{5.3}$$

The goal of static modeling is to fit the estimated distribution $G_k(X)$ to the measured distribution $P_k(X)$. We use Kullback-Leibler divergence [160] to estimate the accuracy of the model (i.e., the error between the estimated and measured distributions). The Kullback-Leibler divergence between the measured and the estimated probability density for each bin (P_k and G_k, respectively) can be mathematically defined as:

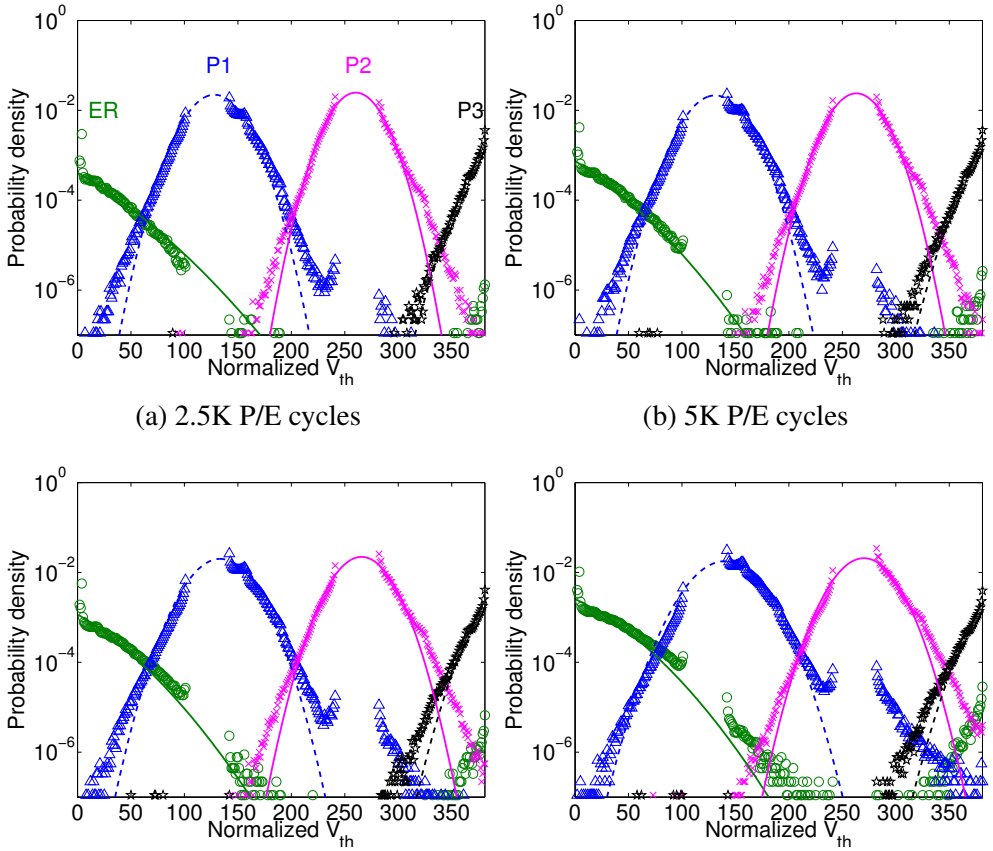
$$D_{K-L} = \sum_{k=1}^{N_{bin}} P_k \log(\frac{P_k}{G_k}) \tag{5.4}$$

We use the Nelder-Mead simplex method [245] to minimize the error, in order to learn the model under different P/E cycles. We use a reasonable initial guess of the parameters for the Nelder-Mead simplex method, allowing us to quickly approach the best fit.

Figure 5.2 shows the distribution measured by our experimental characterization using markers, and shows how the Gaussian-based model (the curves depicted with solid or dashed lines) fits to this data at different P/E cycle counts. The x-axis is the normalized threshold voltage, and the y-axis is the probability density function at each normalized threshold voltage in log scale. In this figure, from left to right, we show the threshold voltage distribution of the ER state, the P1 state, the P2 state, and the P3 state. We show the modeled distributions of the ER and P2 states using solid lines, and the modeled distributions of the P1 and P3 states using dashed lines.

We observe that the Gaussian-based model has two limitations, which are demonstrated in Figure 5.2. First, the threshold voltage distribution of each state as measured from real flash chips has a *fatter tail* than that of a Gaussian distribution, and the left and right tails of each state have different sizes. We observe the fat tail by comparing the measured distribution of each state to the modeled distribution when the probability density is low (i.e., less than 10^{-4}), and find that the measured distribution has a much greater density than the modeled distribution at the tail. This is because the Gaussian distribution has only two parameters for each state, which capture only the center (μ) and the width (σ) of the distribution. We observe the asymmetric tail by comparing the densities of the left and right tails of the P2 state distribution. Unfortunately, the Gaussian distribution has no way to fine tune the ratio between the left and right tails, or the ratio between the tails and the body of the distribution.

Second, the measured distribution demonstrates large second peaks in the distributions of the ER and P1 states, which are not captured by the Gaussian-based model. These second peaks are evidence of a significant number of program errors (see Section 2.2.4). Figure 5.2 shows that the ER state distribution (the leftmost distribution) has a *second* peak that shows up under the P3 state distribution, and that the P1 state distribution (the second distribution from the left) has a second peak under the P2 state distribution. These second peaks occur as a result of the two-step programming mechanism used in MLC NAND flash memory. As we discuss in Section 2.2.4, *program errors* can be introduced for the ER and P1 states as a result of intermediate operations that take place while a cell is partially programmed, which causes the LSB to be misread.



(a) 2.5K P/E cycles

(b) 5K P/E cycles

(c) 10K P/E cycles

(d) 20K P/E cycles

Figure 5.2: Gaussian-based model (solid/dashed lines) vs. data measured from real NAND flash chips (markers) under different P/E cycle counts.

As we observe in Figure 5.2, both types of inaccuracies occur *throughout all P/E cycle counts* (from 2.5K to 20K), and are not, as prior work had shown [260], exclusive to high wear-out scenarios (e.g., when the P/E cycle count is higher than the vendor-specified lifetime). The magnitudes of the fatter tails and program error peaks increase as wear-out (i.e., P/E cycle count) increases. As we can see, even though the Gaussian-based model captures the general trend for the threshold voltage distribution and is easy to compute, it is limited in its accuracy, especially at higher P/E cycles.

Section 5.3.4 quantifies the modeling error and computational requirements of the Gaussian-based model.

5.3.2 Normal-Laplace-based Model

To overcome the limitations of the Gaussian-based model, prior work [260] proposes to modify the model to increase its accuracy. This modified threshold voltage distribution model *assumes* that the distribution of each state follows a normal-Laplace distribution, and accounts for the peaks that result from misprogramming some cells that should be in the ER and P1 states into

the P3 and P2 states, respectively [260].

The normal-Laplace distribution combines the normal (Gaussian) distribution with the Laplace distribution, which adds an exponential component to both tails of the distribution. As we observe in Figure 5.2, this is similar to the measured behavior of the threshold voltage distribution. Note that the figure is in log scale, and as a result, the exponential component at the tails of the model appears as a straight line in the figure. By combining the two probability distributions, we can maintain the Gaussian distribution at the center of the distribution, and also model the fat tail more accurately.

However, computing the normal-Laplace distribution becomes much more complex than the Gaussian distribution, as the normal-Laplace distribution is *not* a simple superposition of the Gaussian and Laplace distributions. Equation 5.5 shows how we compute the cumulative distribution function for the normal-Laplace distribution [277]:

$$\begin{aligned} &NCDF(V,\mu,\sigma,\alpha,\beta,\lambda) \\ &\quad = \Phi(Z) - \phi(Z)\frac{\beta R(\alpha\sigma - Z) - \alpha R(\beta\sigma + Z)}{\alpha+\beta} \end{aligned} \tag{5.5}$$

This distribution adds two new parameters, α and β, which can be adjusted to model the right and left tail sizes, respectively. In Equation 5.5, $Z = \frac{V-\mu}{\sigma}$ is the z-score; Φ and ϕ are the cumulative distribution function and probability density function of the standard Gaussian distribution, respectively; and $R(x) = \frac{1-\Phi(x)}{\phi(x)}$ is Mills' ratio for the Gaussian distribution. $\Phi(Z)$ can be obtained by looking up the z-table, as was done for Equation 5.3. $\phi(Z)$ can be approximated as $\phi(Z) = \frac{\Phi(Z+\delta)-\Phi(Z-\delta)}{2\delta}$.

The normal-Laplace-based model adds two further parameters, λ_{ER} and λ_{P1}, to model the probability of program errors occurring for cells programmed to the ER and P1 states, respectively. This model assumes that the threshold voltage distribution of the cells with program errors has the same shape (i.e., the same parameters) as the distribution of the state the cells were incorrectly programmed into (e.g., the cells that should be in the ER state but were programmed into the P3 state will have a distribution with the same shape as the correct cells in the P3 state). This is because once the cells are incorrectly programmed to another state, they are treated as if they belong to that other state, and thus it is natural for them to follow the same distribution as the correct cells in that state. Equation 5.6 shows how the normal-Laplace model estimates the probability density for state X being misprogrammed to state Y in bin k, which is denoted as $N_k(X)$:

$$\begin{aligned} N_k(X) = &(1-\lambda_X)NCDF(V_k,\mu_X,\sigma_X,\alpha_X,\beta_X) \\ &+ \lambda_X NCDF(V_k,\mu_Y,\sigma_Y,\alpha_Y,\beta_Y) \\ &- (1-\lambda_X)NCDF(V_{k-1},\mu_X,\sigma_X,\alpha_X,\beta_X) \\ &- \lambda_X NCDF(V_{k-1},\mu_Y,\sigma_Y,\alpha_Y,\beta_Y) \end{aligned} \tag{5.6}$$

This density is calculated as the difference of the *NCDF* at the bin's two boundaries, V_k and V_{k-1}. The normal-Laplace-based model allows each state to have at most five parameters (20 parameters over all four states). μ and σ are the mean and standard deviation, respectively; α

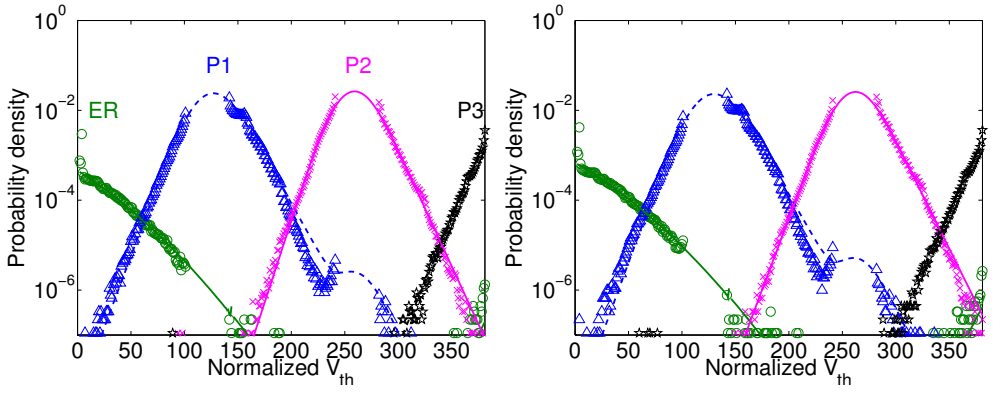
and β are the tail sizes; and λ_X is the probability that a cell that should actually be in state X is incorrectly programmed.

Following prior work [260], we eliminate four unnecessary parameters of the model, which include λ_{P2}, λ_{P3}, β_{ER}, and α_{P3}. λ_{P2} and λ_{P3} are estimated as zero, as program errors for cells that should be in the P2 or P3 states seldom occur. We also assume that the left and right tails are the same size for the ER and P3 states (i.e., $\beta_{ER} = \alpha_{ER}$ and $\alpha_{P3} = \beta_{P3}$), because the read-retry mechanism prevents us from measuring the left tail of the ER state and the right tail of the P3 state. As we did in Section 5.3.1, and following prior work [260], we use Kullback-Leibler divergence error [160] as the objective function, and we use the Nelder-Mead simplex method [245] with a reasonable initial guess to learn the best parameters under different P/E cycles.

Figure 5.3 shows the modeled distribution of each state as curves with solid or dashed lines, and shows the distribution measured from real chips using markers. As we can see, *the normal-Laplace-based model fits the measured distribution much better than the Gaussian-based model*. The modeled tails for the ER, P2, and P3 states follow the measured distribution very closely, thanks to the tail size parameters. Also, the distributions of the ER and P1 states take the program error rate into account, and allow the model to correctly include two peaks for the distributions of the ER and P1 states.

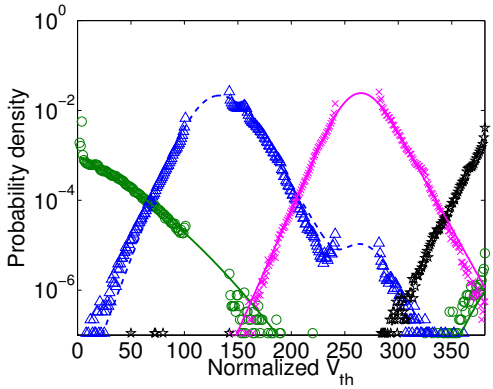
Unfortunately, although the normal-Laplace model is based on the Gaussian model, the computational requirements of the model are much more complex. This is not only because the model adds three more parameters for each state, but also because we now cannot eliminate μ and σ using the z-score. Thus, directly computing the model requires many more floating point operations than the Gaussian model (as we demonstrate in Section 5.3.4). As such, even though the normal-Laplace model fits the measured threshold voltage distribution accurately, it is less practical to implement.

Section 5.3.4 quantifies the modeling error and computational requirements of the normal-Laplace-based model.

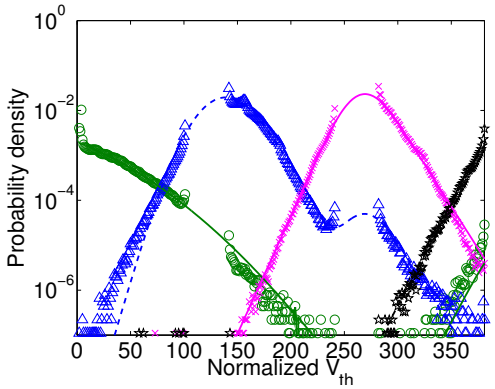


(a) 2.5K P/E cycles

(b) 5K P/E cycles



(c) 10K P/E cycles



(d) 20K P/E cycles

Figure 5.3: Normal-Laplace-based model (solid/dashed lines) vs. data measured from real NAND flash chips (markers) under different P/E cycle counts.

5.3.3 Student's t-based Model

Recall that we need a threshold voltage model that is both accurate and easy to compute. As we can see, the Gaussian-based model is simple and fast, but does not meet the accuracy requirement. In contrast, the normal-Laplace-based model fixes the accuracy problem, but uses significantly more complex calculations. We thus aim to develop a model that meets both of our requirements at the same time.

We propose to modify the Student's t-distribution [300] so that it can be used to model the threshold voltage distribution. The Student's t-distribution is a well-known distribution used in statistics that describes samples drawn from a normally-distributed population. The Student's t-distribution is typically used to estimate the true mean of a large, normally-distributed population whose standard deviation is unknown, using only a small sample from the population. Compared to the standard normal distribution, the Student's t-distribution uses an extra parameter, ν, to represent the *degrees of freedom* (i.e., the ratio of the sample size relative to the population size). As ν increases (i.e., the sample size becomes larger), the Student's t-distribution moves closer to a standard normal distribution. However, instead of using the distribution for its original purpose, we use this distribution for a completely different role. We find that ν can be used to tune the size

of the distribution tail. When $\nu \to +\infty$, the Student's t-distribution becomes a standard Gaussian distribution, which has a smaller tail. When $\nu \to 0$, the distribution instead has a fatter tail. We generalize the standard Student's t-distribution using the z-score $Z = \frac{V-\mu}{\sigma}$, such that the center and the width of the distribution can be scaled (as was done for the Gaussian distribution). We also allow the left and right tails of the distribution to have different values of ν, which we denote as β and α for the left and right tails, respectively. Thus, our modified Student's t-distribution can fit our measured threshold voltage distribution better than the original Student's t-distribution.

We use *precomputation* to simplify the calculation of the cumulative distribution function for our modified Student's t-distribution ($TCDF$). Similar to the precomputed z-tables available for the Gaussian-based model, we look up values in the precomputed *t-tables* commonly available for the Student's t-distribution to determine the $TCDF$ values. Each t-table contains $TCDF$ values over a range of Z values *for a single* ν.[3] Equation 5.7 shows how we calculate $TCDF$ using the precomputed t-table:

$$TCDF(V,\mu,\sigma,\alpha,\beta) = \begin{cases} t\text{-}table_{\beta}(Z) & V \leq \mu \\ t\text{-}table_{\alpha}(Z) & V > \mu \end{cases} \tag{5.7}$$

We first compare V with μ to observe whether V is on the left side or the right side of the distribution. Then, depending on the result of the comparison, we use the corresponding tail parameter α or β as ν to select the correct t-table. Finally, we compute the z-score Z to look up $TCDF$ in the selected t-table.

Equation 5.8 shows how our Student's t-based model estimates the density for cells that should be in state X but are incorrectly programmed to state Y in bin k, which is denoted as $T_k(X)$:

$$\begin{aligned} T_k(X) = {} & (1-\lambda_X)TCDF(V_k,\mu_X,\sigma_X,\alpha_X,\beta_X) \\ & + \lambda_X TCDF(V_k,\mu_Y,\sigma_Y,\alpha_Y,\beta_Y) \\ & - (1-\lambda_X)TCDF(V_{k-1},\mu_X,\sigma_X,\alpha_X,\beta_X) \\ & - \lambda_X TCDF(V_{k-1},\mu_Y,\sigma_Y,\alpha_Y,\beta_Y) \end{aligned} \tag{5.8}$$

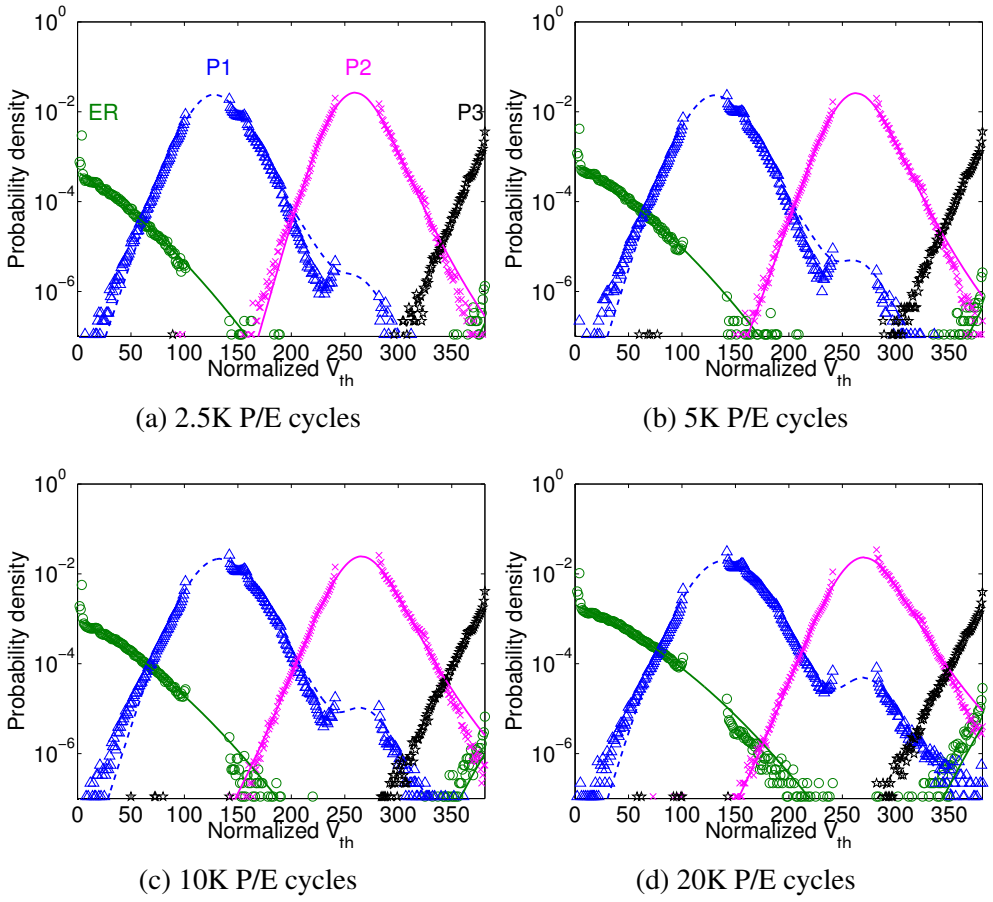
Similar to the normal-Laplace-based model (Section 5.3.2), our Student's t-based model uses λ to estimate such program errors caused by the two-step programming mechanism (see Section 2.2.4). Again, like the normal-Laplace-based model, our Student's t-based model assumes that the distribution of these cells has the same parameters as the cells correctly programmed into state Y.

We set λ_{P2} and λ_{P3} to zero, $\beta_{ER} = \alpha_{ER}$, and $\alpha_{P3} = \beta_{P3}$ for the same reasons as the normal-Laplace-based model (see Section 5.3.2). Putting everything together, we use Kullback-Leibler divergence error [160] as our objective function, and use the Nelder-Mead simplex method [245] with a reasonable initial guess to learn the best parameters for the model under different P/E cycles, as described in Section 5.3.1.

Figure 5.4 shows our modeled Student's t-based distribution as curves with solid or dashed lines, once again showing the distribution measured from real chips with markers. The figure

[3]The t-table can also be thought of as a two-dimensional array, where each entry corresponds to a unique pair of (Z, ν) values.

shows that our Student's t-based model fits perfectly when the probability density is greater than 10^{-4}. The differences between our Student's t-based model and the measured distribution are within 10^{-6}. The difference becomes non-trivial only for the left tail of the P1 state and the right tail of P2 state. The accuracy improvements over the Gaussian model are similar to those of the normal-Laplace-based model. This shows that our Student's t-based model, like the normal-Laplace-based model, makes good use of its extra parameters (both models have 16 parameters) to cater to the program errors and fat tails that the measured distribution has.



(a) 2.5K P/E cycles

(b) 5K P/E cycles

(c) 10K P/E cycles

(d) 20K P/E cycles

Figure 5.4: Our new Student's t-based model (solid/dashed lines) vs. data measured from real NAND flash chips (markers) under different P/E cycle counts.

5.3.4 Model Validation and Comparison

Accuracy. To quantitatively compare the accuracy of each model and validate them, we compute the Kullback-Leibler (K-L) divergence [160] between the modeled and the measured distributions, as K-L divergence measures the difference between two distributions (see Section 5.3.1). Table 5.1 and Figure 5.5 show the modeling error of the three models across a range of P/E cycle counts. We observe two types of behavior shared by all three models. First, as the P/E cycle count increases, the modeling error increases. Second, the increase in modeling error is

more rapid at *smaller* P/E cycle counts, and slower at higher P/E cycle counts. As we see in Section 5.4, this is because the threshold voltage distribution is affected by the P/E cycling effect more significantly at smaller P/E cycle counts.

Table 5.1: Modeling error of the evaluated threshold voltage distribution models, at various P/E cycle counts.

P/E Cycles	0	2.5K	5K	7.5K	10K	12K	14K	16K	18K	20K	**AVG**
Gaussian	.99%	1.8%	1.6%	1.8%	1.9%	2.4%	3.1%	8.7%	2.1%	2.3%	**2.6%**
Normal-Laplace	.34%	.46%	.55%	.61%	.63%	.67%	.68%	.70%	.67%	.67%	**.61%**
Student's t	.37%	.51%	.61%	.68%	.70%	.76%	.76%	.78%	.76%	.78%	**.68%**

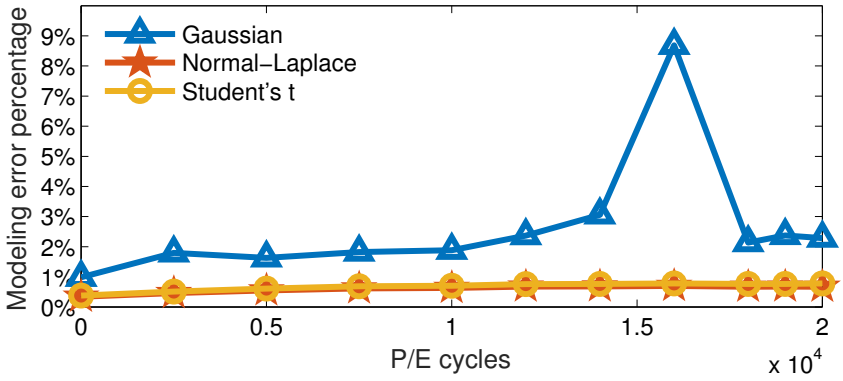


Figure 5.5: Modeling error of the evaluated threshold voltage distribution models, at various P/E cycle counts.

Comparing the three models in Figure 5.5, we make two observations. First, the average Kullback-Leibler divergence error for the Gaussian-based model is 2.64%, which is 4.32x and 3.88x greater than the error of the normal-Laplace-based and our Student's t-based models, respectively. We also see that this error can be as large as 8.7%, leading to high inaccuracy. This is mainly due to two limitations of the Gaussian-based model. As mentioned in Section 5.3.1, the Gaussian-based model (1) cannot adjust its tail size to fit with the fat and asymmetric tails of the observed distributions of the voltage states, and (2) does not account for misprogrammed cells that form a second peak in the distributions of the ER and P1 states.

Second, the modeling errors of the normal-Laplace-based and our Student's t-based models are very close (averaged across all tested P/E cycles, 0.61% for the normal-Laplace-based model, and 0.68% for our Student's t-based model). The maximum difference in error between these two models is 0.11%, at 20K P/E cycles (already well beyond the rated lifetime of the flash chip, which is 3K P/E cycles).

Complexity. All three of the models require online *iterative computation* whenever the flash controller needs to generate a characterization of the threshold voltage distribution at a new P/E cycle count. For each iterative computation, hundreds to thousands of iterations of the model computation algorithm must be executed before the model reaches high accuracy (i.e., the model

converges, or in other words, reaches *convergence*). As we alluded to in Section 5.3.2, while the normal-Laplace-based model is accurate, it requires significant computation during each iteration, and cannot be precomputed and stored in a lookup table, making it less practical for use within a flash controller. To compare the complexity of the three models, we summarize their computation overhead in terms of the number of floating-point operations and table lookups performed for each iteration, as well as their storage overhead in terms of lookup table size. Table 5.2 compares the three models. As we can see, the normal-Laplace-based model requires 91,200 operations per iteration (which involves computing $N_k(X)$ for four states in each of the 304 threshold voltage bins). Assuming that each floating-point operation takes the same number of cycles, the normal-Laplace-based model is 10.71x slower than the Gaussian-based model. In contrast, our Student's t-based model takes 4.41x less computation time than the normal-Laplace-based model, with near-identical accuracy. Our Student's t-based model is only 2.43x slower than the Gaussian-based model, but has a 74% smaller modeling error.

Table 5.2: Computation and storage complexity comparison for the three evaluated threshold distribution models.

Model	Gaussian	Normal-Laplace	Student's t
Operations	8512	91200	20672
Storage	640B	3.84KB	25.6KB

The third row of the table shows the storage overhead for the z-table or t-tables used by each model (which are populated in the flash controller's DRAM when the flash device is powered up). The Gaussian-based model needs only 640B to store the useful range of the z-table. The normal-Laplace-based model requires a larger lookup range for the z-table, increasing the storage overhead to 3.84KB. Our Student's t-based model requires storing multiple t-tables (one table per value of ν), and uses 25.6KB of storage in total. We find that all three storage overhead values are negligible, as these tables are easily stored within the flash controller's DRAM, which is usually sized to be a fixed fraction of the flash storage capacity (e.g., 1GB memory for a 512GB drive).

Latency. The flash controller builds a threshold voltage distribution model in two steps: *characterization* and *model computation*. First, we identify the threshold voltages of each cell in a sampled flash wordline by performing 303 read operations, one read for each read reference voltage level (using the approach described in Section 5.2). This *characterization* takes 30.3 ms for the wordline, assuming a typical read latency of 100μs. Second, once characterization is complete, the controller *computes* the model, using a combination of the precomputed tables stored in DRAM and the characterized voltages. To calculate the overhead, we assume that each of the models takes 1000 iterations to converge, and that computation is performed on a 1GHz embedded processor that completes one instruction per cycle.

Figure 5.6 shows the overall latency for the three models we evaluate, broken down into characterization latency (which is the same for all three models) and model computation latency. The computation overhead of the normal-Laplace-based model dominates its overall latency, while the computation overhead of our Student's t-based model is much smaller than the characterization latency. As a result, our Student's t-based model has a 58.0% lower overall latency than the normal-Laplace-based model. Since the fixed characterization latency dominates overall

latency in both our Student's t-based model and the Gaussian-based model, our model is only 31.3% slower in overall latency than the Gaussian-based model, while it reduces modeling error by 74%.

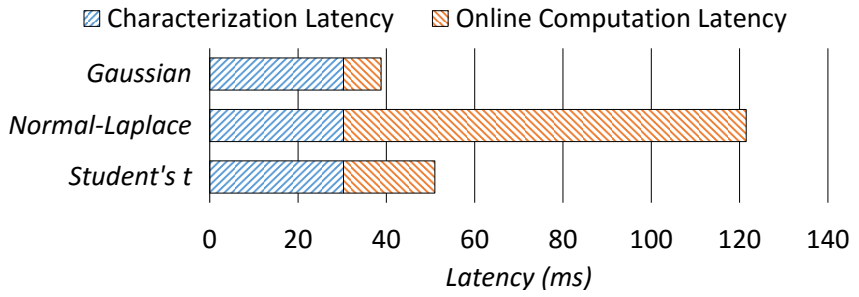


Figure 5.6: Overall latency breakdown of the three evaluated threshold voltage distribution models for static modeling.

The frequency with which the characterization and modeling procedure is triggered depends purely on the application making use of the threshold voltage distribution model. Note that the choice of model should not change the frequency at which the procedure is executed (as each model provides an equivalent end result). As an example, we can determine the amortized overhead per 4KB read/write operation for one application of our model, which predicts the optimal read reference voltage (see Section 5.5.2). The prediction mechanism requires us to repeat the characterization and modeling procedure only once every 1000 P/E cycles. For a flash device with 512 pages per block, if we conservatively assume a read-to-write ratio of 1:1, the average overhead amortized over each read/write operation is 49.8ns using our static Student's t-based model [196].

Summary. In summary, the majority of the accuracy improvement over the Gaussian-based model comes from (1) accounting for the program errors for the erased and P1 states, and (2) accounting for the fat tails of each state. Our Student's t-based model, as well as the previously-proposed normal-Laplace-based model, both contain these improvements, and hence achieve similar accuracy.

Our Student's t-distribution based model has much lower complexity than the normal-Laplace-based model due to its ability to exploit precomputation. We show in Section 4.3 that the CDF of the Student's t-distribution can be simplified into a simple table lookup using the z-score, Z, and the degrees of freedom, ν. We are unaware of a similar precomputation-based approach that can be applied to the normal-Laplace model.

We conclude that our new Student's t-based model achieves the high accuracy of the normal-Laplace-based model while requiring significantly less complexity and latency to compute. As such, we believe that our Student's t-based model meets the requirements of accuracy and simplicity, and is a practical model for implementation within the flash controller.

5.4 Dynamic Modeling

We now construct a *dynamic* threshold voltage distribution model, building off of our Student's t-based static model in Section 5.3.3, to capture how the threshold voltage distribution *changes* as the program/erase (P/E) cycle count increases. Again, we must ensure that this dynamic model is accurate, and that it is easy to compute, as we aim to implement the model within the flash controller. To construct the model, we first analyze how each of the individual parameters making up our Student's t-based model change over the P/E cycle count (Section 5.4.1). By analyzing the meaning of each parameter and observing how it changes, we gain new insights on how the threshold voltage shifts with increasing P/E cycle count. We then use these new insights to construct a model using the power law, which can successfully predict the *future* changes to each of these parameters based on the *current* threshold voltage distribution (Section 5.4.2). Finally, we validate this model (Section 5.4.3).

5.4.1 Static Model Trends Over P/E Cycles

In order to analyze and observe how the parameters for our Student's t-based model change as P/E cycle count increases, we first need to understand what each parameter means. As we discuss in Section 5.3.3, our Student's t-based model has 16 parameters. Four of them are the mean values for each state X's threshold voltage distribution (μ_X). Another four parameters are the standard deviation values of the threshold voltage distribution of each state X (σ_X). Three of them are the left tail sizes of the P1, P2, and P3 state distributions (β_X), and another three are the right tail sizes of the ER, P1, and P2 state distributions (α_X). (Recall from Section 5.3.2 that the left tail of the ER state and the right tail of the P3 state cannot be observed experimentally, so we assume that they equal the right tail of the ER state and the left tail of the P3 state, respectively.) The remaining two parameters are the probability of program errors, occurring for cells programmed into the ER and P1 states (λ_X).

Mean. The mean value of each state represents the center of the distribution. In our Student's t-based model, the majority of the mass of the threshold voltage distribution for each state is near the center. Thus, a change in the mean reflects how the P/E cycle count generally affects the threshold voltages of *all* cells in each state.

Figure 5.7 plots the mean values obtained from sample Student's t-based models constructed over a range of 20K P/E cycles, shown as circles. The x-axis shows the P/E cycle count, while the y-axis shows the normalized threshold voltage of the mean. Each graph plots the mean value for a different state, which is labeled at the top of the graph. We make three observations from this figure. First, the mean value of each state's distribution increases monotonically with P/E cycle count. Second, the mean value increases faster at lower P/E cycle counts, then slows down to a constant rate of increase after 5K P/E cycles. Third, the mean value shifts more quickly for lower threshold voltage states (ER, P1).

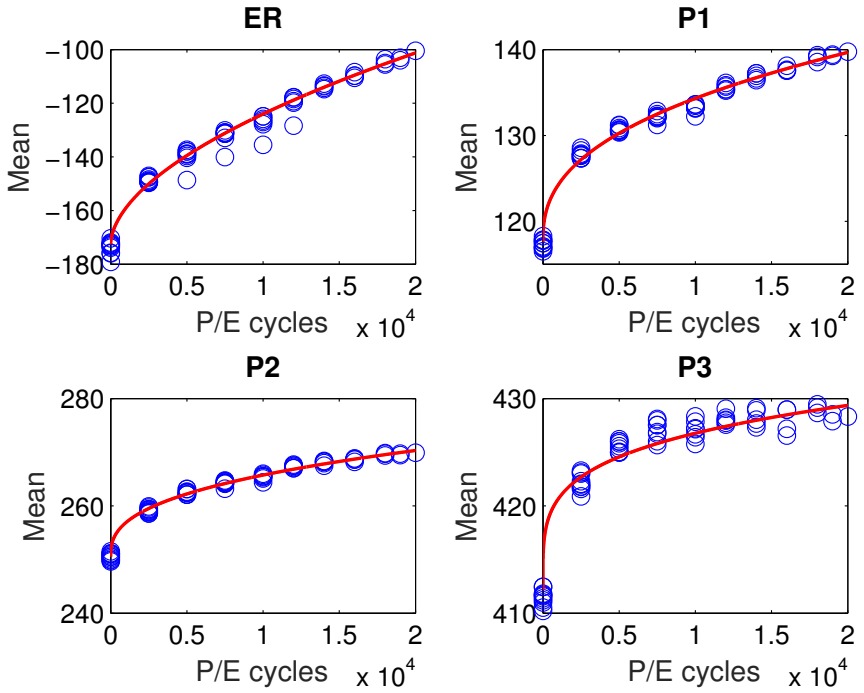


Figure 5.7: Change in mean value of each state's threshold voltage distribution as P/E cycle count increases, for the static Student's t-based model (blue circles) and the dynamic model (red line).

Standard Deviation. The standard deviation of each state represents the width of the distribution. Similar to the Gaussian distribution, the Student's t-distribution contains the vast majority ($\sim$95%) of its mass within two standard deviations. Thus, the change in standard deviation reflects how P/E cycle count affects the threshold voltage variation among flash cells.

Figure 5.8 plots the standard deviation values obtained from our Student's t-based model as circles. For this figure, the y-axis shows the standard deviation in terms of normalized threshold voltage. We make three observations from this figure. First, the standard deviation of each state's distribution increases monotonically with P/E cycle count. Second, the standard deviations of the P1 and P2 states increase linearly with P/E cycle count. Third, like the mean, the standard deviation increases faster at lower P/E cycle counts, then slows down to a constant rate of increase after 5K P/E cycles.

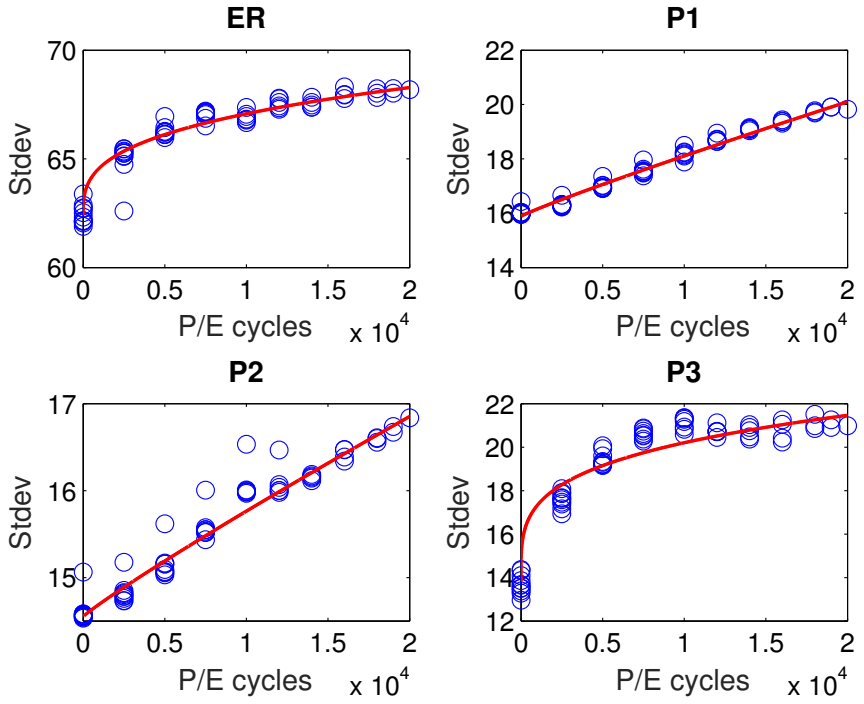


Figure 5.8: Change in standard deviation of each state's threshold voltage distribution as P/E cycle count increases, for the static Student's t-based model (blue circles) and the dynamic model (red line).

Tail Values. The tail values of each state represent the size and shape of the distribution tail. Recall from Section 5.3.3 that we use ν, which actually represents the degrees of freedom, to control how fat the tail of the model is. Thus, the tail value reflects how the P/E cycle count affects the number of outlier cells (i.e., the number of cells that lie at the tail).

Figure 5.9 plots the tail values obtained from our Student's t-based model as circles. In this figure, the y-axis shows the value of ν, where a lower value of ν corresponds to a fatter tail. We make three observations. First, the range of values for the tail sizes of the ER and P3 states is much smaller in comparison to the tail sizes of the distributions of the other states. Second, the sizes of both tails for the P1 state increase with P/E cycle count. Third, the tail sizes of the P2 state decrease as P/E cycle count increases.

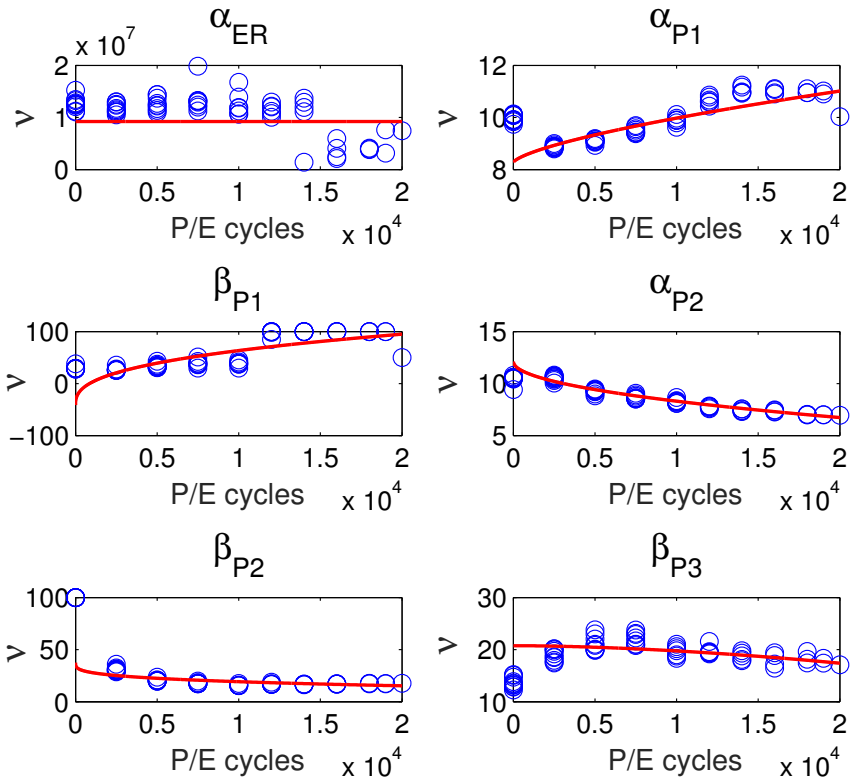


Figure 5.9: Change in tail values (ν) of each state's threshold voltage distribution as P/E cycle count increases, for the static Student's t-based model (blue circles) and the dynamic model (red line).

Probability of Program Errors. The program error probability λ_X represents the percentage of cells that should be programmed into state X, but are instead misprogrammed to a different state, as a result of two-step programming (see Section 2.2.4). In our model, we assume that only certain types of program errors exist (ER→P3 and P1→P2), as program errors flip the value of only the LSB within a cell and can only *increase* the threshold voltage.

Figure 5.10 plots the program error probability obtained from our Student's t-based model as circles. For this graph, the y-axis shows the $\log_{10}$ value of the program error probability. We make two observations. First, the program error rate increases with P/E cycle count. Second, the number of program errors increases more rapidly at lower P/E cycle counts, and then slows down to a constant rate of increase at higher P/E cycle counts.

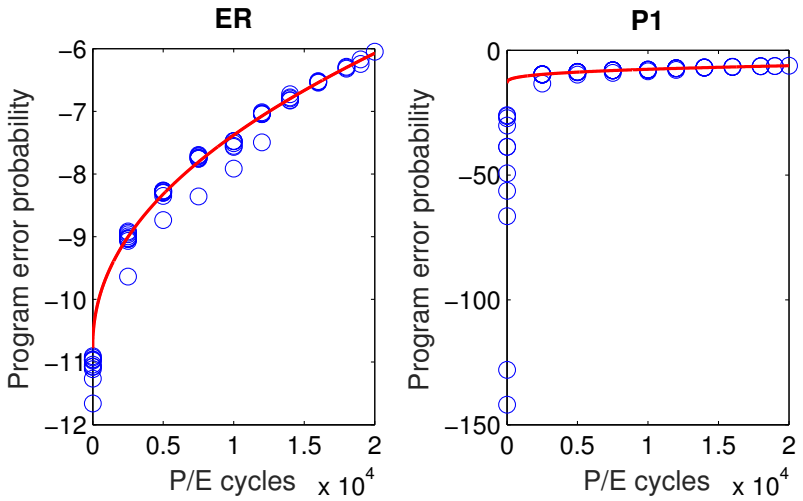


Figure 5.10: Change in log value of the program error probability as P/E cycle count increases, for the static Student's t-based model (blue circles) and the dynamic model (red line).

5.4.2 Power Law-based Model

Now that we have characterized how each of the parameters for our Student's t-based model changes with respect to the P/E cycle count, we use this characterization to develop a *dynamic* model of the threshold voltage distribution. A dynamic model can reduce the total computation effort for the threshold voltage distribution significantly, by requiring as little as a *single* static model *characterization* for the entire lifetime of the flash device. The dynamic model takes the static characterization-based model(s) generated in the past, and simply adjusts the model parameters at higher P/E cycle counts based on its prediction of how each parameter would change with P/E cycle count (without requiring any further characterization). Without the dynamic model, a static model of the characterization must be generated *every time* a new threshold voltage distribution is requested by the controller (e.g., after a fixed number of P/E cycles have occurred), with each characterization requiring a large number of read-retry operations (see Section 5.3.4). These read-retry operations increase the accuracy of the model, but interfere with and slow down host commands. Our goal is to build a dynamic model that is accurate and easy to compute (such that it requires only a small number of characterizations), so that it can be used within the flash controller.

In Section 5.4.1, we observe that all of the parameters can increase, decrease, or remain relatively constant. We also observe that the rate at which increases and decreases occur differs between lower P/E cycle counts and higher P/E cycle counts. Our dynamic model must be able to represent all of these behaviors. We find that the *power law* satisfies all of these characteristics. Equation 5.9 shows the power law function, which models each parameter from our Student's t-based model, Y, as a function of the P/E cycle count (x):

$$Y = a \times x^b + c \tag{5.9}$$

The power, b, can be set to a positive value to represent an increasing trend, or can be set to a negative value to represent a decreasing trend. b can also control the difference in slope at

different P/E cycle counts. For example, when $b < 1$, the modeled parameter Y changes faster at *lower* P/E cycle counts, and when $b > 1$, Y changes faster at *higher* P/E cycle counts.

To observe how well the power law models changes to the parameters of our Student's t-based model, we fit the power law to the values of each of the parameters as measured over several P/E cycle counts (see Section 5.4.1).[4] We use mean squared error (MSE) to estimate the error, where the divergence between the measured and estimated parameters (Y_i and $\hat{Y}_i$, respectively) can be mathematically defined as: $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$. We use the Nelder-Mead simplex method [245], with a reasonable initial guess, to fit the trend.

Figures 5.7, 5.8, 5.9, and 5.10 show the power law-based models fit to the trends of each of our parameters as solid lines. We fit the power law to the static model parameter values generated over a range of 20K P/E cycles. We observe that the predictions from the power law fit very well with the actual parameters measured from our Student's t-based model, which are shown as blue circles. We next quantify the accuracy of our power law-based dynamic model.

5.4.3 Model Validation

We validate our dynamic model by using it to predict the threshold voltage distribution at 20K P/E cycles. We perform threshold voltage distribution characterizations at 2.5K, 5K, 7.5K, and 10K P/E cycles, and use these parameters to predict the distribution at 20K P/E cycles. Figure 5.11 shows the comparison between the actual characterized distribution (markers) and the distribution predicted by our dynamic model (solid or dashed curves) at 20K P/E cycles. The modeling error for the dynamic model is only 2.72%, which is close to the modeling error of directly using a static Gaussian-based model at 20K P/E cycles. The dynamic model avoids the need to perform the extensive read-retry characterization that all static models, including the Gaussian-based model, would require.

[4] We exclude 0 P/E cycle results when modeling, as they show a completely different behavior than results at any other P/E cycle count.

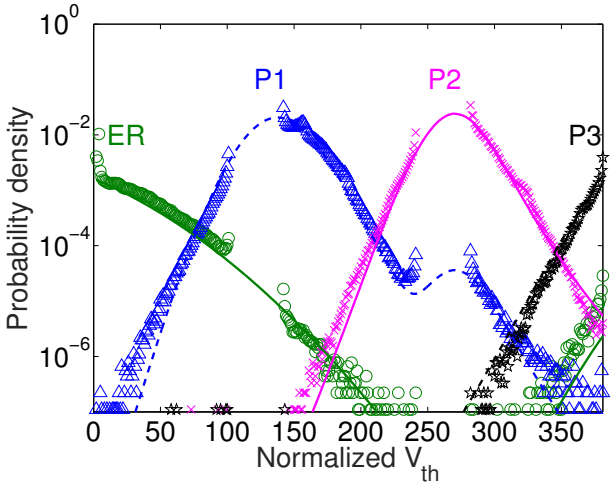


Figure 5.11: Threshold voltage distribution as predicted by our dynamic model for 20K P/E cycles, using characterization data from 2.5K, 5K, 7.5K, and 10K P/E cycles, shown as solid/dashed lines. Markers represent data measured from real NAND flash chips at 20K P/E cycles.

Figure 5.12 shows how the modeling error of our dynamic model decreases for a prediction at 20K P/E cycles as the number of characterized data points increases. The number of characterized data points represents the N earliest static models out of a range that consists of static models for 2.5K, 5K, 7.5K, 10K, 12K, 14K, 16K, 18K, and 19K P/E cycles. Note that we start with three characterization data points, which allows the dynamic model to observe a trend in the change of each parameter. This figure shows that the error rate decreases rapidly as we increase the number of data points used to train the dynamic model.

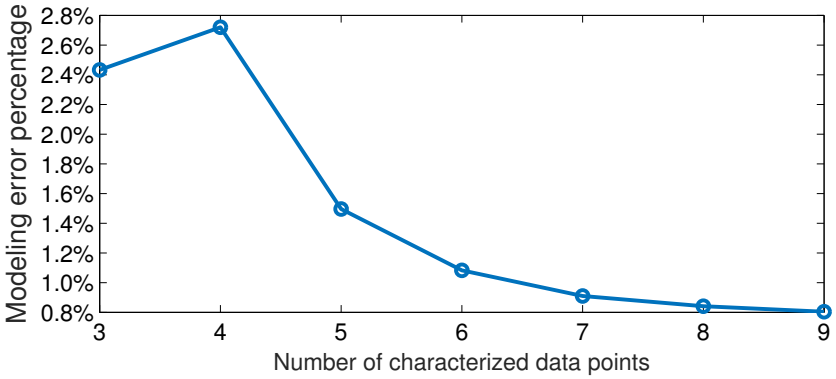


Figure 5.12: Modeling error of predicted threshold voltage distribution for our dynamic model at 20K P/E cycles, using characterization data from N different P/E cycles.

We conclude that our dynamic model successfully predicts an accurate threshold voltage distribution for a P/E cycle count it has not observed, based on only prior characterization data, and is thus practical for use in the flash controller.

5.5 Example Applications

Now that we have developed our threshold voltage distribution model, we demonstrate three example applications within the flash controller that take advantage of the model to enhance the reliability of the flash device. The first application, described in Section 5.5.1, uses our model to accurately estimate the raw bit error rate. The second application, described in Section 5.5.2, uses our model to accurately predict the optimal read reference voltage. The third application, described in Section 5.5.3, uses our model to estimate the expected lifetime of the flash memory device, to safely achieve higher P/E cycle endurance than manufacturer specification.

5.5.1 Raw Bit Error Rate Estimation

The raw bit error rate (i.e., the probability of reading an incorrect state for a flash cell), or RBER, is important not only because it is a measure of the reliability of a flash device, but because it also can be used to determine the lifetime and performance of the flash drive [27]. The raw bit error rate can be used to enable several optimizations in the flash controller. For example, accurately estimating the current raw bit error rate allows us to safely utilize the *currently unused* ECC correction capability to accelerate program operations [122], relax the retention time [27, 188], and reduce the effects of read disturbance [35]. Accurate estimation of the raw bit error rate enables other optimizations, such as predicting the optimal read reference voltage [24] or performing error rate based wear-leveling.

To estimate the raw bit error rate based on the static threshold voltage distribution model, we use the static model to calculate the cumulative distribution function (CDF) for each state at each of our read reference voltages (V_a, V_b, and V_c), and use this data to determine how many cells are misread. For example, if there are cells in the distribution of the ER state whose threshold voltages are greater than V_a, they will be misread. By calculating the ER state CDF up to V_a, we know what percentage of cells will be *correctly* read. We subtract this value from 1 to obtain the percentage of cells that will be *misread* (and will thus contribute to the raw bit error rate).

Figure 5.13 shows the actual measured raw bit error rate and the modeled raw bit error rates using the three static models from Section 5.3, for different P/E cycle counts. The x-axis shows P/E cycle count, and the y-axis shows the measured or model-predicted raw bit error rate. The three graphs show the average error rate for only the LSB pages, only the MSB pages, and for all of the pages. We make two observations from this data. First, the normal-Laplace-based and our Student's t-based models give a much better estimate of the raw bit error rate than the Gaussian-based model. Averaged across all P/E cycle counts, our Student's t-based model estimates the RBER for all pages within 13.0% of the actual measured RBER, while the normal-Laplace-based model is within 14.9% and the Gaussian-based model is only within 44.7%. This is due to the limitations of the Gaussian-based model, as it cannot adjust the tail size or take program errors into account. Second, the normal-Laplace-based and our Student's t-based models tend to overestimate the error rate, which is usually safe for the purposes of many optimizations, because overestimation results in more than adequate ECC correction capability to remain available for these errors. In contrast, the Gaussian-based model always *underestimates* the raw bit error rate, which, if used for an optimization that relies on an RBER estimation, can cause the number of errors to exceed the correction capability of ECC, resulting in uncorrectable errors during reads.

We conclude that our Student's t-based model is effective at providing an accurate estimate of the raw bit error rate for use by the flash controller.

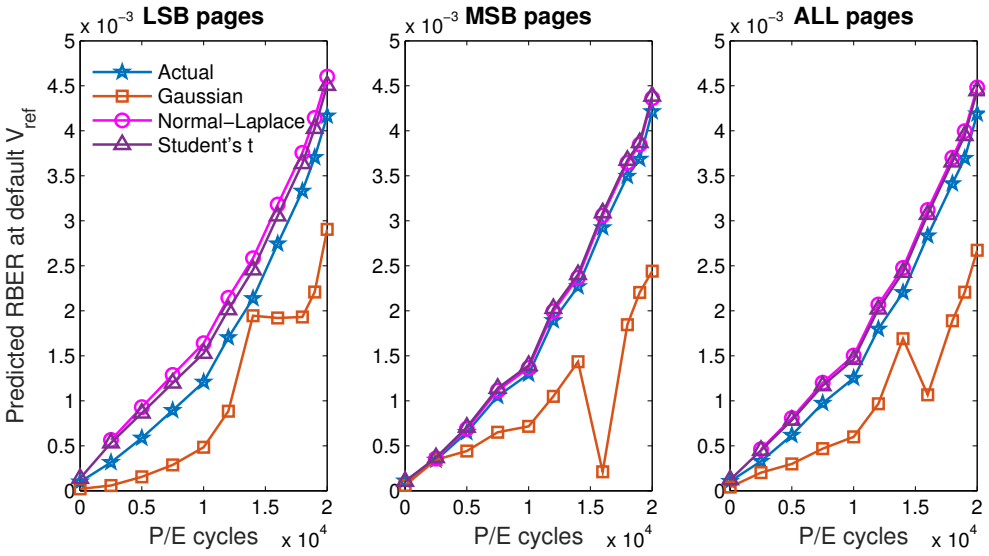


Figure 5.13: Actual and modeled raw bit error rate using the three evaluated threshold voltage distribution models when reading with fixed *default* read reference voltages (V_{ref}), across different P/E cycle counts.

5.5.2 Optimal Read Reference Voltage Prediction

As we discussed in Section 3.1, when the threshold voltage distribution shifts, it is important to move the read reference voltage to the point where the number of read errors is minimized. After the shift occurs, the threshold voltage distributions of each state may overlap with each other, causing many of the cells within the overlapping regions to be misread. The number of errors due to misread cells can be minimized by setting the read reference voltage to be at the point where the distributions of two neighboring states *intersect*, which we call the *optimal read reference voltage* (V_{opt}) [24]. Once the optimal read reference voltage is applied, the raw bit error rate is minimized, improving the reliability of the device. Furthermore, since fewer errors are corrected, and fewer read-retries are needed, read latency is also significantly reduced [27].

Prior work proposes to learn and record the optimal read reference voltage periodically [27, 252, 309] by sampling the threshold voltages of *some* of the cells in each flash block, but this sampling requires time and storage overhead. With our new distribution model, we can determine the optimal read reference voltage from the model and predict how it changes, without having to exhaustively learn it for each block. From our threshold voltage distribution model, we can predict the optimal read reference voltage by finding the point at which the probability density functions of the distributions of two neighboring states are the same (i.e., the intersection of the two distributions).

Figure 5.14 plots the actual measured and modeled optimal read reference voltage using the

three static models from Section 5.3, at different P/E cycle counts.[5] Each graph shows the voltage chosen for one of the three read reference voltages (V_a, V_b, and V_c) used to distinguish between the distributions of two neighboring states. The x-axis shows the P/E cycle count, while the y-axis shows the normalized optimal read reference voltage. We make three observations from this result. First, the normal-Laplace-based and our Student's t-based models slightly overestimate all three optimal read reference voltages. Second, the Gaussian-based model underestimates the optimal read reference voltages in most cases, and has glitches of underestimation as large as 17 voltage steps. We suspect that this is because the Gaussian-based model cannot capture the asymmetric tail sizes of the distribution. Third, at 0 P/E cycles, the read reference voltages predicted using the normal-Laplace-based model deviate significantly from the actual optimal read reference voltages. We find that the normal-Laplace-based model has difficulty converging to a good value at 0 P/E cycles, while our Student's t-based model does not experience any such difficulty.

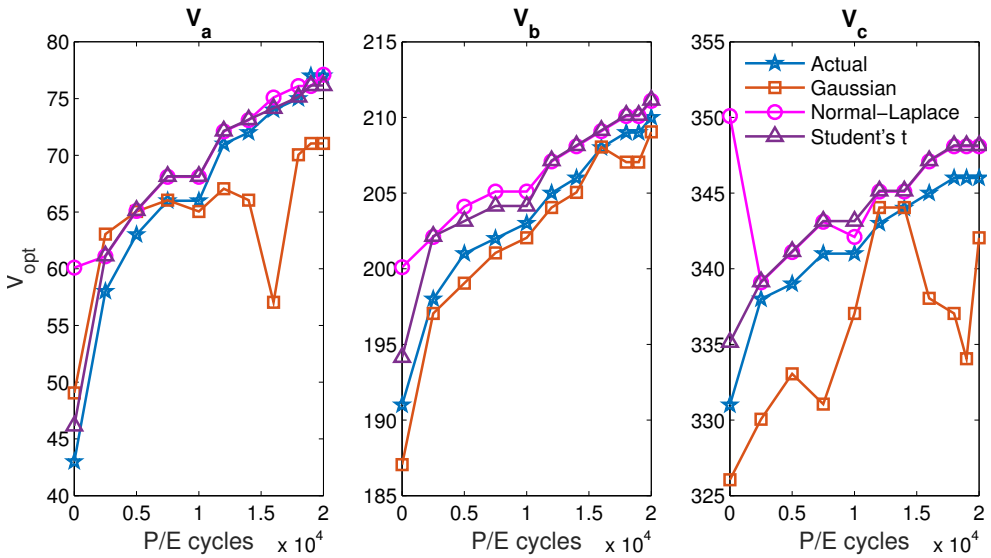


Figure 5.14: Actual and modeled *optimal* read reference voltages (V_{opt}) using the three evaluated threshold voltage distribution models at different P/E cycle counts.

Figure 5.15 shows the RBER when we use the actual optimal read reference voltage to read data, as well as the RBER when we use the optimal read reference voltages predicted by each of the three static models from Section 5.3, at different P/E cycle counts. As we did for Figure 5.13, we show the average error rate for only the LSB pages, only the MSB pages, and for all of the pages. We observe that the prediction generated from the Gaussian-based model results in a significantly higher MSB error rate than the actual optimal voltage. The normal-Laplace-based and our Student's t-based models generate read reference voltage predictions that result in near-optimal RBER (within 1.5% and 1.1%, respectively, of the optimal RBER), despite some difference between the actual optimal read reference voltage and the model-predicted voltages.

[5]Note that the default read reference voltages are (V_a,V_b,V_c) = (50,190,330). We observe that the actual optimal read reference voltage can be higher than the default read reference voltage by as much as 27 voltage steps.

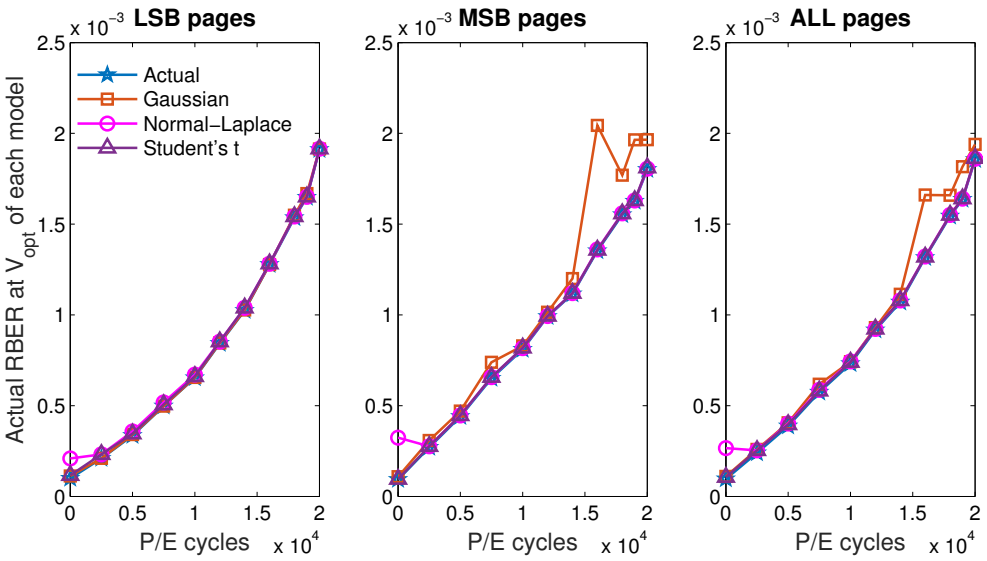


Figure 5.15: RBER achieved by actual and modeled *optimal* read reference voltages (V_{opt}) using the three evaluated threshold voltage distribution models at different P/E cycle counts.

We evaluate how using the optimal voltages predicted by each model can improve flash lifetime compared to using the default read reference voltages. We assume that we have a state-of-the-art LDPC decoder, which can tolerate a raw bit error rate as high as 5×10^{-3} [100] while still keeping the required uncorrectable error rate below 10^{-15} [121] during the flash device's lifetime.[6] We also assume that our flash device refreshes its data every three weeks, limiting the number of retention and read disturb errors that occur [22]. Using the actual (i.e., ideal) optimal read reference voltage, flash lifetime improves by 50.6%. Both our Student's t-based model and the normal-Laplace-based model come very close to the ideal improvement, providing a *48.9% lifetime improvement* with our Student's t-based model. Due to its lower accuracy, the Gaussian-based model achieves only a 38.5% improvement.

5.5.3 Expected Lifetime Estimation

Due to the increasing raw bit error rate at higher P/E cycle counts, flash memory can endure only a limited number of writes. To make sure that all data stored in the flash drive is reliable over the course of a predefined device lifetime (typically several years), enterprise users limit the number of writes to each flash drive. Due to process variation, different flash chips can have different raw bit error rates and thus different P/E cycle endurance. However, flash vendors conservatively set the flash drive's P/E cycle endurance to the *worst case* (i.e., to the lowest endurance value out of all of the chips that they produce), as they do not know how fast each individual flash chip wears out over time. In fact, prior work has tested six commercial flash drives, and found that they all surpassed their official endurance specifications by an average of 81% [86].

If the flash controller can monitor how fast each flash chip wears out due to flash writes, the

[6]To tolerate variation in the raw bit error rate, we assume that 10% of the total ECC correction capability is reserved, lowering the maximum tolerable raw bit error rate.

users can determine the *actual* endurance limit of the flash drive, and write more data to it without worrying about prematurely wearing out the drive and losing data. The model proposed in this chapter enables raw bit error rate prediction for *future P/E cycle counts*. Thus, the controller can predict the endurance limit of each flash chip by iterating through our dynamic model to predict the point at which the raw bit error rate exceeds the ECC correction capability (i.e., when the lifetime *actually* ends). The flash controller then communicates this prediction to the file system to allow higher write intensity to the flash drive.

We estimate the lifetime improvements using this technique, with the same assumptions we made in Section 5.5.2 and the data shown in Section 5.5.1. With our dynamic model, we safely achieve *69.9% higher P/E cycle endurance* than manufacturer specification. This translates to 69.9% more tolerable writes per day, if we assume that the flash device will be used for the same number of years (i.e., lifetime) as before.

5.5.4 Soft Information Estimation for LDPC Codes

To tolerate flash errors more efficiently, today's flash controllers use LDPC codes to detect and correct multiple raw bit errors in the data read from the flash memory channel [78, 326, 362]. An LDPC code can use *soft information* about each bit to increase the probability of correcting the raw bit errors. This soft information is provided by the flash controller, which estimates the probability of each bit being a 1 or a 0 using the threshold voltage of a cell. A modern flash controller typically obtains this probability from a Gaussian-based model for the threshold voltage distribution, since the soft information can be computed as a quadratic function of the threshold voltage. However, as we have shown in Section 5.3, the Gaussian-based model underestimates the probability density at the tail of the distribution, and does not model program errors. Thus, the model can provide inaccurate information to the LDPC decoder. This compromises the error correction capability of the LDPC codes, thus reducing the reliability and performance of the flash drive.

With the models proposed in this chapter, we now have an accurate threshold voltage distribution model that adapts to the P/E cycle of each block, and can be implemented within the flash controller. Using our Student's t-based model, we can accurately and efficiently compute the probability density for any threshold voltage range to provide *accurate* soft information to the flash controller. By increasing the accuracy of this soft information, we effectively increase the error correction capability of the LDPC code, which can lead to longer flash lifetime and better read performance [78, 326, 362]. We leave the precise implementation of such a mechanism for future work.

5.5.5 Improving Flash Performance

While the applications of our threshold voltage distribution model that we have discussed in Sections 5.5 and 5.5.4 aim to improve reliability, they can also improve flash performance. For example, by predicting and applying the optimal read reference voltage (Section 5.5.2), we can greatly lower the probability that read-retries need to be performed for a read operation, which also reduces the number of ECC decoding iterations, both of which lead to a lower read latency [27].

Other applications can also take advantage of our model to improve flash performance. For example, we can minimize the ECC decoding latency by adaptively applying a weaker ECC code when the raw bit error rate indicated by our model is low [100, 101, 336, 358]. We expect and hope future work to evaluate the performance benefits of these applications, and to propose other new applications of our online model that can improve flash performance.

5.6 Related Work

To our knowledge, our work in this chapter is the first to (1) propose a threshold voltage distribution model that is both highly accurate and computationally efficient, (2) propose a dynamic threshold voltage distribution model that predicts how the parameters of this model change with increasing program/erase cycle count, and (3) demonstrate several new practical uses of this threshold voltage distribution model within a flash controller to improve flash memory reliability.

We have already comprehensively compared our Student's t-based static model to the two most relevant models based on real characterization results, the Gaussian-based model [23, 227] and the normal-Laplace-based model [260], in Sections 5.3.1, 5.3.2, and 5.5. We show that our Student's t-based model has an error rate within 0.11% of the error rate of the highly-accurate normal-Laplace model, while requiring 4.41x less computation time. Several prior works fit the threshold voltage distribution to other models that are either less accurate or more complex, such as the beta distribution [23], gamma distribution [23], log-normal distribution [23], Weibull distribution [23], and beta-binomial probability distribution [315]. Other prior works model the threshold voltage distribution based on idealized circuit-level models [78, 227, 251]. These models capture some of the desired threshold voltage distribution behavior, but are less accurate than those derived from real characterization.

A few works also propose dynamic models of the threshold voltage distribution shifts based on the power law [23, 24, 260]. While these models are sufficient for offline analysis, they are unsuitable for deployment in today's flash controllers, as they fail to achieve high accuracy and low computational complexity at the same time. Our dynamic model also uses the power law, but is based on our new, accurate, and low-complexity Student's t-based static model. We show that our model has an error rate of only 2.72% when estimating the distribution at 20K P/E cycles, even though it uses characterization data collected at only four different P/E cycle counts from the past (up to 10K P/E cycles). While other dynamic models based on idealized circuit models exist [78, 251], they are not validated with real characterization data, and cannot achieve the same accuracy as our model.

Prior works propose and evaluate techniques for raw bit error rate estimation [260, 270], optimal read reference voltage estimation [27, 252, 253, 309], and LDPC soft decoding [78, 326, 362]. These works utilize a threshold voltage distribution model only offline, or do not utilize a threshold voltage distribution model at all. We show that, by utilizing our model, we can effectively and practically guide such flash reliability mechanisms *online* in the flash controller. We also provide a new mechanism to *exploit* process variation for *higher* flash endurance, by predicting and safely utilizing the remaining lifetime of a flash device online. Prior works propose to only *tolerate* error rate variation and process variation to improve flash lifetime [180, 228, 229].

We note that several prior works have already extensively studied the impact of retention behavior on the threshold voltage distribution using real hardware [21, 22, 25, 27]. They show that commonly-employed refresh mechanisms in flash devices can successfully mitigate most of the impact of retention on the threshold voltage distribution [22, 25, 194]. As a result, we expect that even without capturing the effects of retention, our proposed threshold voltage distribution model will work well in practice.

5.7 Limitations

Our online model captures only P/E cycling and two-step programming effects, which are two of the most dominant error sources in 1X nm MLC planar NAND flash memory. Currently, our online model does not model and mitigate retention and read disturb errors because they are successfully mitigated by commonly-employed flash refresh mechanisms [22, 222, 250], which can be combined with our online model to provide greater reduction in raw bit errors. Furthermore, our online model can be extended to model the threshold voltage shift due to retention or read disturb. Online modeling effectively reduces retention errors when refresh becomes less effective in 3D NAND, as we show in Chapter 6 and 7.

5.8 Conclusion

In this chapter, we introduce a new threshold voltage distribution model for modern NAND flash memory devices. Our model is based on a new experimental characterization of the threshold voltage distribution and how it shifts over time using state-of-the-art 1X-nm MLC NAND flash chips. Our characterization shows that the threshold voltage distribution can be approximated using our modified version of the Student's t-distribution, and that the amount by which the distribution shifts as the P/E cycle count increases is governed by the power law. Our new model, which combines these two observations in its static and dynamic components, is capable of accurately capturing the current and predicting the future threshold voltage distribution of flash memory cells. We show that our model achieves low modeling error, and is computationally simple enough to implement online in a flash controller. We demonstrate various applications of our model in a flash controller. We show that these applications improve flash lifetime by 48.9% and/or enable the flash device to safely utilize 69.9% more P/E cycles than manufacturer specification. We conclude that our proposed threshold voltage distribution model for modern MLC NAND flash memory devices is practical and effective. We hope that this dissertation inspires future work to improve upon our online flash channel model, and to develop and evaluate new techniques that take advantage of such a model to increase flash memory reliability and performance.

Chapter 6

3D NAND Flash Memory Error Characterization and Mitigation

In order for planar NAND flash memory to continually increase the SSD capacity and decrease the *cost-per-bit* of the SSD, flash vendors have to aggressively scale NAND flash memory to smaller manufacturing process technologies. This, however, comes at the cost of the decreasing flash reliability [21, 33, 219], as we have shown in Section 3.1. Due to a combination of manufacturing process limitations and decreasing reliability, it has become increasingly difficult for manufacturers to continue to scale the density of planar NAND flash memory [257].

To overcome this scaling challenge, 3D NAND flash memory has recently been introduced [115, 131, 257] (see Section 3.4 for a comparison between planar NAND and 3D NAND technology). Previous publicly-available experimental studies on NAND flash memory errors using real flash memory chips (e.g., [21, 22, 23, 24, 26, 27, 34, 35, 195, 219, 260]) have all been on planar NAND devices. As 3D NAND flash memory is already being deployed at a large scale in new computer systems, there is a lack of available knowledge on the error characteristics of real 3D NAND flash chips, which makes it harder to estimate the reliability characteristics of systems that employ such chips.

In this chapter, our goal is to (1) identify and understand the *new* error characteristics of 3D NAND flash memory (i.e., those that did not exist previously in planar NAND flash memory), and (2) propose new mechanisms to mitigate prevailing 3D NAND flash errors. We aim to achieve these goals via rigorous experimental characterization of real, state-of-the-art 3D NAND flash memory chips from a major flash vendor. Based on our comprehensive characterization and analysis, we identify *three new error characteristics* that were not previously observed in planar NAND flash memory, but are fundamental to the new architecture of 3D NAND flash memory. (1) 3D NAND flash exhibits *layer-to-layer process variation*, a new phenomenon specific to the 3D nature of the device, where the average error rate of each 3D-stacked layer in a chip is significantly different (Section 6.2.1). We are the *first* to provide detailed experimental characterization results of layer-to-layer process variation in real flash devices in open literature. (2) 3D NAND flash memory experiences *early retention loss*, a new phenomenon where the number of errors due to charge leakage increases quickly within several hours after programming, but then increases at a much slower rate (Section 6.2.2). We are the *first* to perform an extended duration observation of early retention loss. While prior studies examine the impact of early retention

loss over only the first 5 minutes after data is written, we examine the impact of early retention loss over 24 days. (3) 3D NAND flash memory experiences *retention interference*, a new phenomenon where the rate at which charge leaks from a flash cell is dependent on the amount of charge stored in neighboring flash cells (Section 6.2.3).

Our experimental observations indicate that we must revisit the error models and the error mitigation mechanisms devised for planar NAND flash, as they are no longer accurate for 3D NAND flash behavior. To this end, we develop *new analytical models* of (1) the layer-to-layer process variation in 3D NAND flash memory (Section 6.4.1), and (2) retention loss in 3D NAND flash memory (Section 6.4.2). Both models are useful for developing techniques to mitigate raw bit errors in 3D NAND flash memory. Our models estimate the raw bit error rate (RBER), threshold voltage distribution, and the *optimal read reference voltage* (i.e., the voltage at which the raw bit errors are minimized when applied during a read operation) for each flash page.

We propose *four new techniques* to mitigate the unique layer-to-layer process variation and early retention loss errors observed in 3D NAND flash memory. Our first technique, LaVAR, reduces process variation by fine-tuning the read reference voltage independently for each layer (Section 6.5.1). Our second technique, LI-RAID, is a new RAID scheme that eliminates the page with the worst-case reliability within each block by changing how we pair up pages from different flash blocks (Section 6.5.2). Our third technique, ReMAR, reduces retention errors in 3D NAND flash memory by tracking the retention age of the data using our retention model and adapting the read reference voltage to the data age (Section 6.5.3). Our fourth technique, ReNAC, predicts and adapts the read reference voltage to the amount of retention interference during each read operation (Section 6.5.4). These four techniques are complementary, and can be combined together to significantly improve NAND flash reliability. Compared to a state-of-the-art baseline, our techniques provide a combined 3D NAND flash memory lifetime improvement of 85.0%. Alternatively, in the case where a NAND flash manufacturer wants to keep the lifetime of the 3D NAND flash memory device constant, our techniques reduce the storage overhead required to hold error correction information by 78.9%.

6.1 3D NAND Error Characterization Overview

Our goal is to identify and understand new error characteristics in 3D NAND flash memory, through rigorous experimental characterization of real, state-of-the-art 3D NAND flash memory chips. We use the observations and analysis of this characterization to (1) compare how the reliability of a 3D NAND flash memory chip differs from that of a planar NAND flash memory chip, (2) develop a model of how each new error source affects the error rate of 3D NAND flash memory, (3) understand if and how these reliability characteristics will change with future generations of 3D NAND flash memory, and (4) develop mechanisms that can mitigate new error sources in 3D NAND flash memory.

For our characterization, we use the methodology discussed in Section 6.1.1. First, we perform a detailed characterization and analysis of three error characteristics that are drastically different in 3D NAND flash memory than in planar NAND flash: process variation (Section 6.2.1), retention errors (Section 6.2.2), and retention interference (Section 6.2.3). In addition to identifying *new* error sources in 3D NAND flash memory, we use our methodology to corroborate

and quantify 3D NAND error characteristics that are a result of error sources that were previously identified in planar NAND flash memory, including data retention [22, 27, 55, 257], P/E cycling [23, 195, 257, 260], program interference [24, 26, 257], read disturb [35, 260], and process variation [21, 269]. We summarize our findings for these error types in Section 6.2.4, and provide detailed results on our characterization of these previously-identified error sources in Section 6.3.

6.1.1 Methodology

We experimentally characterize several real, state-of-the-art 3D MLC NAND flash memory chips from a single vendor.[1] We use a NAND flash characterization platform similar to prior work [20], which allows us to issue *read-retry* commands directly to the flash chip. The read-retry command allows us to fine-tune the read reference voltage used for each read operation. The smallest amount by which we can change the read reference voltage is called a *voltage step*. We conduct all experiments at room temperature (20 °C).

We use two metrics to evaluate 3D NAND reliability. First we show the *raw bit error rate* (RBER), which is the rate at which errors occur in the data *before error correction*. We show the RBER when we read data using the *optimal read reference voltage* (V_{opt}), which is the read reference voltage that generates the fewest errors in the data.[2]

Second, we show how the various error sources change the *threshold voltage distribution*. These change (i.e., shifting and widening) in threshold voltage distribution directly leads to raw bit errors in the flash memory. To obtain the distribution, we first use the read-retry command to sweep over all possible voltage values, in order to identify the threshold voltage of each cell.[3] Then we use this data to calculate the probability density of each state at every possible threshold voltage value. As part of our analysis, we fit the threshold voltage distribution of each state to a Gaussian distribution. We use the *mean* of the Gaussian model to represent how the distribution shifts as a result of errors, and we use the *standard deviation* of the model to represent how the distribution widens. Throughout this chapter, we present normalized voltage values instead of the actual voltage values, as the latter are proprietary to NAND flash memory manufacturers. A normalized voltage of 1 represents a single voltage step.

We show two examples in Figure 6.1 to visualize how well this simple Gaussian model captures the change in the measured threshold voltage distribution. Figure 6.1 shows the measured and modeled distributions under two conditions: (1) 0 P/E cycles, 0-day retention, and 0 read disturbs (i.e., the data contains few errors); and (2) 10K P/E cycles, 3-day retention, and 900K read disturbs (i.e., the data contains a high number of errors). We plot the distribution read from 3D NAND chips using dots. We use a solid line to show a fitted Gaussian distribution for each state. The average root mean square error of the fitted distributions is 1.6×10^{-3}. We observe, from

[1] The trends we observe from the characterization are expected to be similar for 3D charge trap flash manufactured by different vendors, as their 3D flash architectures are similar in design.

[2] We show RBER at the optimal read reference voltage to accurately represent the reliability of NAND flash memory, as SSD controllers tune the read reference voltage to a near-optimal point to extend the NAND flash lifetime [27, 195, 252].

[3] We refer to Section 5.2 for more detail on the methodology to obtain the threshold voltage distribution [195, 260].

this figure, that after the chip is worn out, the threshold voltage distribution is shifted by all types of noise, reducing the error margins between neighboring states, which leads to more raw bit errors in the data. Thus, showing how these distributions are affected by various noises helps us understand how raw bit errors occur and mitigate these errors more effectively.

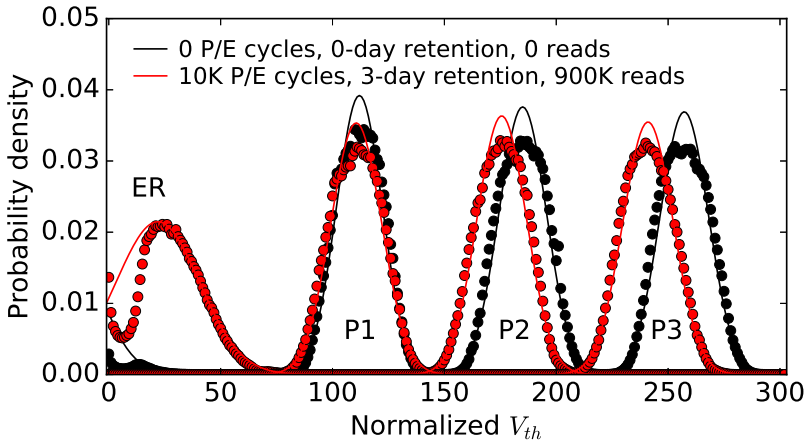


Figure 6.1: 3D NAND threshold voltage distribution before (black) and after (red) the data is subject to a high number of errors.

In the following sections, we directly show the mean and the standard deviation of the fitted threshold voltage distributions instead of the distribution itself, for the simplicity of the results.

6.2 Key Characterization Results

6.2.1 Layer-to-Layer Process Variation

Process variation refers to the variation in the attributes of flash cells when they are fabricated (see Section 3.1). Due to process variation, some flash cells can have a higher RBER than others, making these cells the limiting factor of overall flash memory reliability. In 3D NAND flash memory, process variation can occur along all three axes of the memory (see Figure 3.27). Among the three axes, we expect the variation along the z-axis (i.e., layer-to-layer variation) to be the most significant, due to the new challenge of stacking multiple flash cells across layers. Prior work has shown that current circuit etching technologies are unable to produce identical 3D NAND cells when punching through multiple stacked layers, leading to significant variation in the error characteristics of flash cells that reside in different layers [112, 328].

To characterize layer-to-layer process variation errors within a flash block, we first wear out the block by programming random data to each page in the block until the block reaches 10K P/E cycles. Then, we compare the collective characteristics of the flash cells in one layer with those in another layer. We repeat this experiment for flash blocks on multiple chips to verify all of our findings.

Observations. Figure 6.2 shows the RBER variation along the z-axis (i.e., across layers)

for a flash block with 10K P/E cycles.[4] The top graph breaks down the errors into MSB and LSB page errors; the bottom graph breaks down the errors according to the original and current state of each cell. In the top graph, the solid curve and the dotted curve show the results for two blocks that were randomly selected from two different flash chips. We make five observations from Figure 6.2. First, both of the MSB and LSB error rates vary significantly across layers. For example, MSB page on normalized layer 55 in the middle has an RBER 6 $\times$ higher than normalized layer 0. Second, ER $\leftrightarrow$ P1 and P1 $\leftrightarrow$ P2 errors vary significantly across layers, while P2 $\leftrightarrow$ P3 errors remain similar across layers. Recall from Figure 2.7 that ER $\leftrightarrow$ P1 errors are MSB errors and P1 $\leftrightarrow$ P2 errors are LSB errors. Third, the top half of the layers have lower error rates than the bottom half. Fourth, the middle layers have much a higher RBER than other layers. Fifth, the RBER variation we observe is consistent across two randomly selected blocks from two different chips.

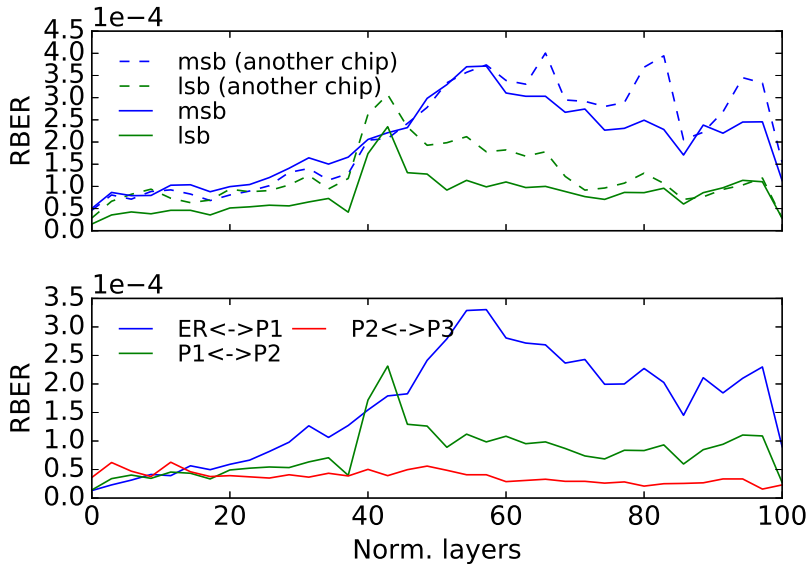


Figure 6.2: Layer-to-layer variation of RBER.

Figure 6.3 shows how the optimal read reference voltages vary across layers. Three subfigures show the optimal read reference voltages for V_a, V_b, and V_c. We make two observations from this figure. First, the optimal voltages for V_a and V_b vary significantly across layers, but the optimal V_c does not change by much. Second, the optimal values of V_a and V_b increase in the top half of the layers, but decrease in the bottom half.

[4]We normalize the number of layers from 0 (the top-most layer) to 100 (the bottom-most layer). The chips we use for characterization have 30 to 40 layers.

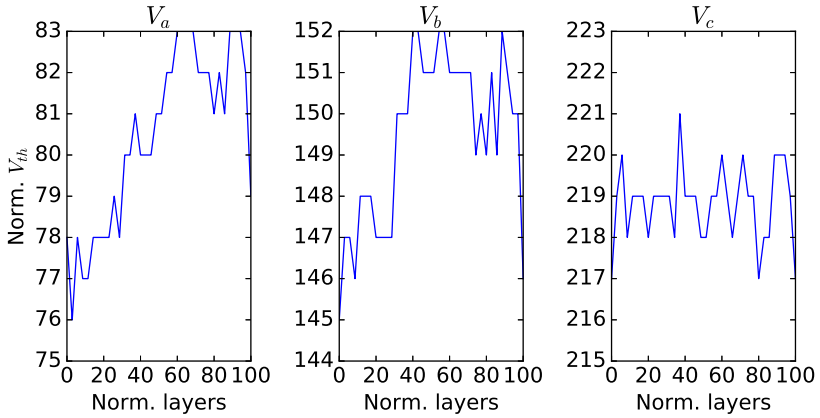


Figure 6.3: Optimal read reference voltage variation across layers.

Insights. We show that the layer-to-layer process variation is significant, which is unique to 3D NAND flash memory. In the future, as 3D NAND flash devices scale along the z-axis, more layers will be stacked vertically along each bitline. This will further exacerbate the effect of layer-to-layer process variation, making it even more important to study and mitigate this effect further.

6.2.2 Early Retention Loss

Retention errors are flash errors that accumulate after data has been programmed to the flash cells [22, 27] (see Section 3.1). Because 3D NAND flash memory typically uses a different cell design (i.e., the charge trap cell described in Section 3.4) than planar NAND flash memory (which uses floating-gate cells), it has drastically different retention error characteristics. The charge trap flash cells used in 3D NAND flash memory suffer from early retention loss, i.e., fast charge loss within a few seconds. This phenomenon has been observed by prior works using circuit-level characterization [47, 55]. However, due to limitations of the methodology used by these prior works, openly-available characterizations of early retention loss in 3D charge trap NAND flash devices document retention loss behavior for up to only 5 minutes after the data is written (i.e., for a maximum *retention age* of 5 minutes). This limited window is insufficient for understanding early retention loss under real workloads, which typically have much longer retention ages [194].

Our goal is to experimentally characterize early retention loss in 3D NAND flash memory for a large range of retention ages (e.g., from several minutes to several weeks). First, we randomly select 11 flash blocks within each chip and write pseudo-random data to each page within the block to wear the blocks out. We wear each block to a different number of P/E cycles, so that we have error data for every 1K P/E cycles between 0 and 10K P/E cycles.[5] Then we program pseudo-random data to each flash block, and wait for up to 24 days under room temperature. To characterize retention loss, we measure the RBER and the threshold voltage distribution at

[5]For all experiments throughout the chapter, we consistently assume a 0.5-second *dwell time*, which is the length of time between consecutive program/erase operations.

nine different retention ages, ranging from 7 minutes to 24 days. To minimize the impact of other errors, and to allow us to include very low retention ages, we characterize only the first 72 flash pages within each block. We believe that the observations we make on these flash cells are representative of the entire chip, and we can generalize the observations to all 3D NAND cells. Our threshold voltage distribution analysis is provided in Section 6.3.2.

Observations. Figure 6.4 shows the comparison between the retention error rate of 3D NAND and planar NAND flash memory at 10,000 P/E cycles. To do this comparison, we perform the same experiment as above for planar NAND flash memory chips, and extend the retention error rate trend to the same time scale using a linear fit. We observe that the retention error rate changes much more slowly for planar NAND than for 3D NAND flash memory. Although the 3D NAND flash chip has lower RBER than the planar NAND flash chip shortly after programming, the RBER becomes higher on the 3D NAND flash chip after $\sim$2 hours of retention time. This demonstrates the early retention error in 3D NAND flash memory, where the RBER quickly increases by an order of magnitude in $\sim$3 hours, and by another order of magnitude in $\sim$11 days. In contrast, RBER increases slowly over retention time for planar NAND flash.

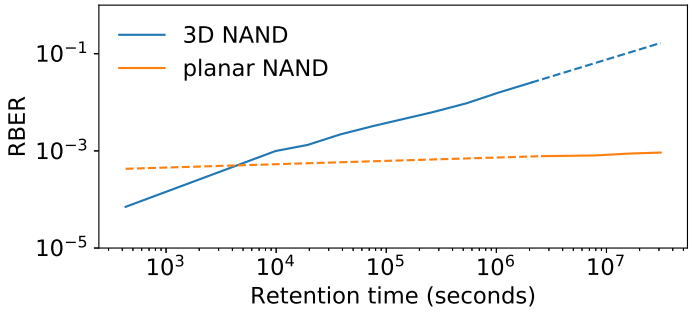


Figure 6.4: Retention error rate comparison between 3D NAND and planar NAND flash memory.

Figure 6.5 plots how the optimal read reference voltage changes with retention age. The three subfigures show the optimal voltages for V_a, V_b, and V_c. We make three observations from this figure. First, the relation between the optimal read reference voltages for V_b or V_c and the retention age can be modeled as: $V = A \cdot \log(t) + B$. Second, the optimal read reference voltages for V_b and V_c decrease significantly as retention time increases, whereas V_a remains relatively constant. Third, the optimal read reference voltages of V_b and V_c change rapidly when the retention age is low, but they change slowly when the retention age is high.

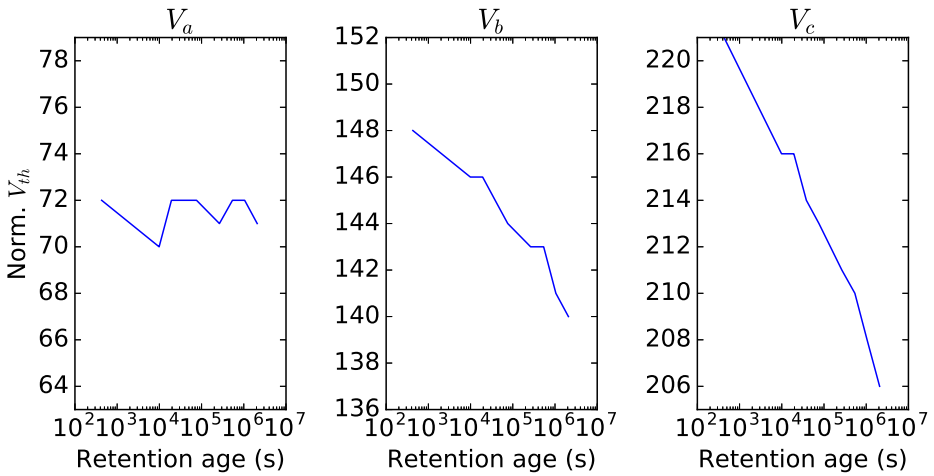


Figure 6.5: Optimal read reference voltages, for varying retention ages.

Insights. We compare the errors caused by retention loss in 3D NAND to that in planar NAND, using our results in Figure 6.4 as well as the results reported in prior work [22, 27, 219]. We find *two* major differences in 3D NAND. More results and insights are in Section 6.3.2.

First, 3D NAND error increases much faster when the retention age is low than when the retention age is high. As we have shown in Figure 6.4, both the logarithm of the RBER and the shift in the threshold voltage approximately fit a linear function of the logarithm of the retention age. This means that the retention loss is *steep* when retention age is *low*, but the retention loss flattens out when the retention age is high. This is a result of the early retention phenomenon in 3D NAND flash memory. There are two possible reasons for early retention loss. First, the tunnel oxide layer is thinner in 3D NAND [282, 359]. This is because a 3D charge trap cell uses an insulator to store charge, which makes the cell immune to the short circuiting caused by stress-induced leakage current (SILC). Thus, the tunnel oxide layer in a 3D charge trap cell is designed to be thinner to improve program speed. Second, the charge can now migrate out of the charge trap in three dimensions [55]. In planar NAND flash memory, charge leakage due to retention occurs across the tunnel oxide. In 3D NAND flash memory, the charge can leak across both the tunnel oxide *and* the insulator being used for the charge trap, which we discuss further in Section 6.2.3. However, as we show in Figure 6.4 for planar NAND flash memory, we do not observe a large difference in retention loss between low and high retention ages [27, 219].

Second, the optimal read reference voltage for V_b in 3D NAND flash memory shifts with retention age. However, in planar NAND flash memory, the optimal voltage of V_b does not change by much [27]. This makes adjusting the optimal read reference voltages even more important for 3D NAND flash memory than for planar NAND flash memory.

6.2.3 Retention Interference

Retention interference is the phenomenon that the speed of retention loss for a cell depends on the threshold voltage of a neighboring cell. Retention interference is unique to 3D NAND flash

memory, as cells along the same bitline in 3D NAND flash memory share the same charge trap layer. If two neighboring cells are at different threshold voltages, charge can leak away from the cell with a higher threshold voltage to the cell with a lower threshold voltage over time [55].

We use the same data used for retention loss in Section 6.2.2 to observe the effect of retention interference. To eliminate any noise due to program interference, we use neighboring cells on the previous wordline to establish the interference correlation, as these cells are not greatly affected by program interference. We also ignore victim cells that are in the ER state, as they are significantly affected by program interference on both sides along the bitline. By eliminating program interference issues, the cells should experience a similar threshold voltage shift *except for* the effects of retention interference. To find the retention interference, we first group all the cells based on their current state, and the state stored in the previous wordline. Then, we compare the amount by which the threshold voltages shift over a 24-day retention age, for each group, to observe how the cells are impacted by neighboring cells.

Observations. Figure 6.6 shows the average threshold voltage shift over 24-day retention age, broken down by the state of the victim cell (V) and the state of the neighboring cell (N). Each bar represents a different (V, N) pair. Different shades represent the different states of the neighboring cell, as labeled in the legend. Every 4 bars are grouped by the state of the victim cell, as labeled on the right side. From Figure 6.6, we observe that the threshold voltage shift over retention age is lower when the neighboring cell is in a higher-voltage state (e.g., the P3 state).

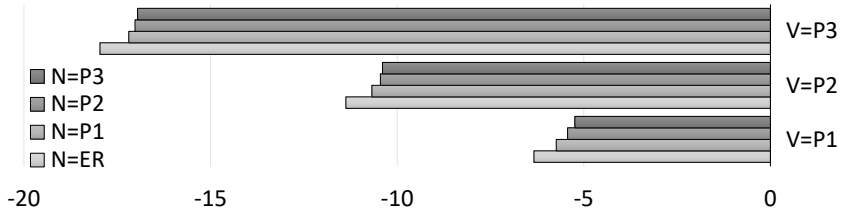


Figure 6.6: Retention interference at 10K P/E cycles.

Insights. We are the first to discover and characterize retention interference in 3D NAND flash memory. Our observation from Figure 6.6 shows that the amount of retention loss for a flash cell is correlated with its neighboring cell's state. We expect retention interference to become stronger as we shrink the manufacturing process technology in future 3D NAND flash memory devices. This is because the distance between neighboring cells will decrease, and fewer electrons will be stored within each flash cell, increasing the susceptibility of the cells to interference from neighboring cells.

6.2.4 Summary

In addition to the three new error sources we find in 3D NAND flash memory, we also characterize the behavior of other known error sources in 3D NAND flash memory and compare them to their behavior in planar NAND flash memory. We present a high-level summary of our findings for these errors here, and provide detailed results and analyses on these error sources in Section 6.3:

- Unlike in planar NAND, we do *not* observe any *programming errors* in 3D NAND [34, 195, 260] (Section 6.1.1).
- The P/E cycling effect in 3D NAND flash memory follows a linear trend, which is similar to that in planar NAND flash memory using an older manufacturing process technology (e.g., 20 nm to 24 nm) [23]. However, in sub-20 nm planar NAND flash memory, the P/E cycling effect exhibits a *power-law* trend [195, 260] (Section 6.3.1).
- 3D NAND flash memory experiences 40% *less program interference* than 20 nm to 24 nm planar NAND flash memory [24, 26] (Section 6.3.1).
- 3D NAND flash memory experiences 96.7% *weaker read disturb* than 20 nm to 24 nm planar NAND flash memory [35]. The impact of read disturb is low enough in 3D NAND flash memory that it does *not* require significant error mitigation (Section 6.3.3).

Note that these differences are mainly due to the larger manufacturing process technology currently used in 3D NAND flash memory, and thus are not the focus of this chapter. In comparison, the new error characteristics that we focus on (layer-to-layer process variation, early retention loss, retention interference) are caused by fundamental changes introduced in 3D NAND flash memory.

We summarize the key differences between 3D NAND and planar NAND flash memory, in terms of error characteristics and the expected trends for future 3D NAND devices, in Table 6.1. The first column of this table lists the attributes we study. The second column shows the key differences in the observations that we make in 3D NAND flash memory. The third column shows the fundamental cause of each difference. The last column shows the expected trend of each difference in future 3D NAND flash devices. We provide the necessary characterizations and models that help us quantitatively understand these differences in Section 6.3.

Attribute	Observation in 3D NAND	Cause of Difference	Future Trend
Process Variation (Section 6.2.1, Section 6.3.4, 6.3.5)	Layer-to-layer process variation is significant	Vertical stacking of flash cells	Process variation will increase as we stack more cells vertically
Retention Loss (Section 6.2.2, 6.2.3, Section 6.3.2)	Early retention loss	Charge-trap cell	Early retention loss will continue if charge-trap cell is used
	Retention interference	Vertical stacking of flash cells	Retention interference will increase when smaller process technology is used
P/E Cycling (Section 6.3.1)	Distribution parameters change over P/E cycle following linear trend instead of power-law trend	Larger manufacturing process technology	P/E cycle trend will go back to power-law trend when smaller process technology is used
Program Interference (Section 6.3.1)	Wordline-to-wordline interference along z-axis	Vertical stacking of flash cells	Will stay true in 3D NAND
	40% lower program interference correlation	Larger manufacturing process technology	Program interference correlation will increase when smaller process technology is used
V_{th} Distribution (Section 6.1.1)	ER and P1 states have no programming errors	Use of one-shot programming instead of two-step programming	Programming errors may come back if two-step programming is used
Read Disturb (Section 6.3.3)	96.7% smaller read disturb effect	Larger manufacturing process technology	Read disturb effect will increase when smaller process technology is used

Table 6.1: Summary of flash error characteristics of 3D NAND and planar NAND flash memory.

6.3 Comprehensive Characterization Results

6.3.1 Write-Induced Errors

We now analyze how each type of write-induced error affects the RBER and the threshold voltage distribution of 3D NAND flash memory.

Program/Erase Variation Errors

A program/erase variation error, or P/E cycling error, occurs when the SSD cannot properly erase or program a cell because some electrons are trapped within the cell [23, 219] (see Section 3.1). To study the impact of program/erase variation errors, we randomly select a flash block within each 3D NAND chip, and wear out the block by programming random data to each page in the block until the block reaches 16K P/E cycles.[6] Using the methodology described in Section 6.1.1, we obtain the overall RBER and the threshold voltage of each cell at various P/E cycle counts.[7]

Observations. Figure 6.7 shows how the RBER increases as the P/E cycle count increases. The top graph breaks down the errors into which page (i.e., LSB or MSB) they occur in. The bottom graph breaks down the errors based on how the error changed the cell state due to a shift in the cell threshold voltage. If the error resulted in *either* the LSB or MSB (but not both) being incorrect, we refer to that as a single-bit error (e.g., an error that shifts a cell originally programmed in the ER state to the P1 state, or vice versa; labeled ER $\leftrightarrow$ P1 in the graph). If both the LSB and MSB are incorrect as a result of the shift, we refer to that as a multi-bit error. We make four observations from Figure 6.7. First, both LSB and MSB errors increase as the P/E cycle count increases, following an exponential trend. Second, ER $\leftrightarrow$ P1 errors increase at a much faster rate as the P/E cycle count increases, with respect to the other types of cell state changes, and ER $\leftrightarrow$ P1 errors become the dominant MSB error type when the P/E cycle count reaches 6K P/E cycles. Third, multi-bit errors are rare, and they occur as early as 1K P/E cycles. Fourth, MSB pages have a 2.1 $\times$ higher error rate than LSB pages.

[6] For all experiments throughout this chapter, we consistently assume a 0.5-second dwell time, which is the length of time between consecutive program/erase operations.

[7] Due to limitations with our experiment platform, each data point at a particular P/E cycle count has an average retention age of 50 minutes.

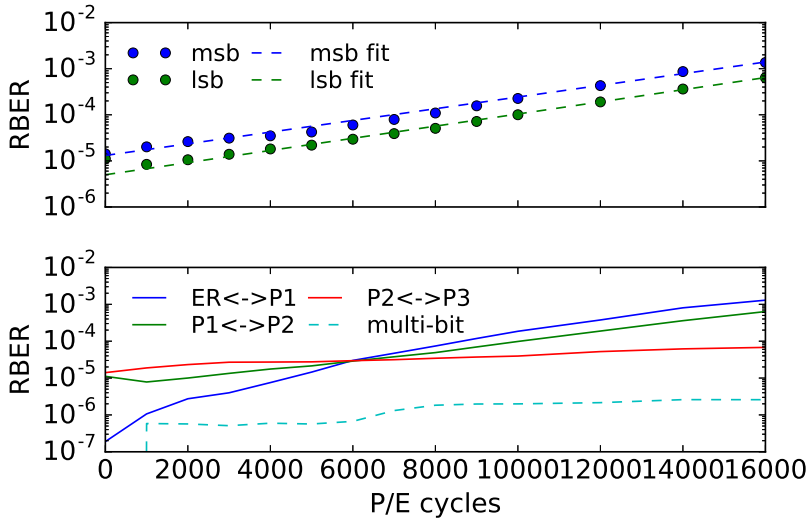


Figure 6.7: Program/erase variation errors vs. P/E cycles.

Figures 6.8 and 6.9 show how the Gaussian model mean and standard deviation, respectively, change with P/E cycle count for the threshold voltage distribution. Each figure is broken down into four subfigures, with each subfigure showing the model parameter change for a different state. We make four observations from Figures 6.8 and 6.9. First, the mean and standard deviation of all states follow a linear trend.[8] Second, the threshold voltage distributions for the ER state and the P1 state shift to higher voltages, while the distributions for the P2 state and the P3 state shift to lower voltages. Third, the threshold voltage distributions for all four states become wider as the P/E cycle count increases (i.e., the standard deviation increases). Fourth, the magnitude of the shift and widening is much larger for the ER state than it is for the other three states (i.e., P1, P2, P3). The shift and the widening of the distribution explains what we observe on RBER in Figure 6.7. The RBER between two neighboring states is equivalent to the overlapped area of the two states' distributions. Since the overlapping tail of each distribution is an exponential function [195, 260]—as the distributions shift towards each other and widens at a linear speed—the overlapping area should increase exponentially, as we observe in Figure 6.7.

[8]For the ER state, a linear fit has a 5.9% higher error rate than a power-law fit. However, we choose the linear fit due to its simplicity.

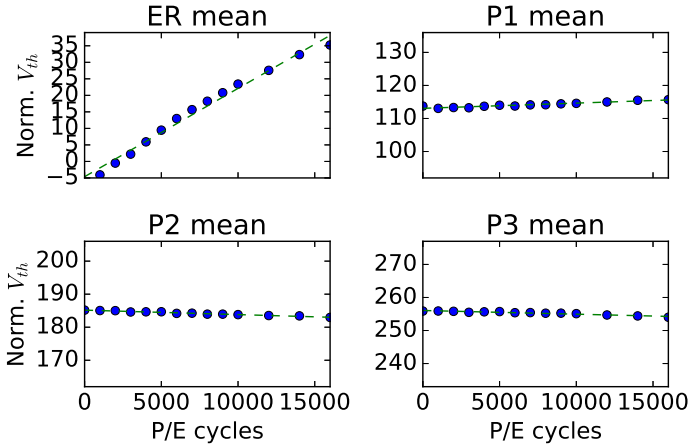


Figure 6.8: Mean of distribution for program/erase variation error model, as the P/E cycle count increases.

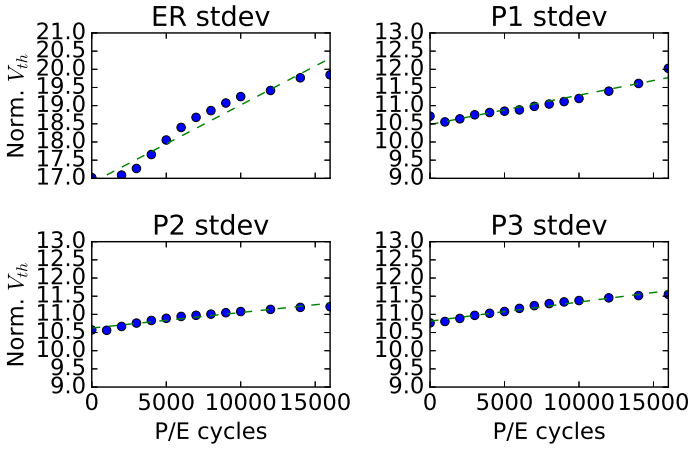


Figure 6.9: Standard deviation of distribution for program/erase variation error model, as the P/E cycle count increases.

Figure 6.10 shows how the optimal read reference voltages change as the P/E cycle count increases. This figure contains three subfigures, each of which shows the optimal voltage for V_a, V_b, and V_c (see Figure 2.7). We make two observations from this figure. First, the optimal voltage for V_a increases rapidly as the P/E cycle count increases: after 16K P/E cycles, the voltage goes up by more than 20 voltage steps. Second, the optimal voltage for V_b and V_c remain almost constant as the P/E cycle count increases: neither voltage changes by more than 4 voltage steps after 16K P/E cycles.

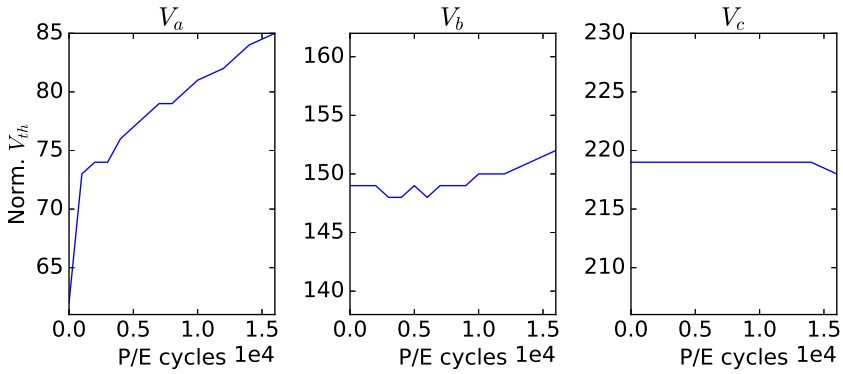


Figure 6.10: Optimal read reference voltages vs. P/E cycles.

Insights. To compare the error characteristics of 3D NAND to that of planar NAND flash memory, we refer to equivalent observations on planar NAND reported by prior works [23, 195, 260], and compare them to our findings for 3D NAND. We find two key differences. First, for 3D NAND, the threshold voltage distributions for the P2 state and the P3 state shift to lower voltages as the P/E cycle count increases. In contrast, for planar NAND flash memory, the distributions of both states shift to *higher* voltages [23, 195, 260]. This could be due to the retention noise that occurs during characterization, which lowers the threshold voltage of cells in higher voltage states (i.e., P2 and P3) [55]. Second, for 3D NAND, the *change* in the mean of each state distribution exhibits a linear trend. However, in sub-20 nm planar NAND flash memory, the change in the mean exhibits a *power-law* trend [195, 260]. In sub-20 nm planar NAND flash memory, the mean of each state distribution increases more rapidly at lower P/E cycle counts than in higher P/E cycle counts, resulting in power-law behavior. However, we note that planar NAND flash memory using an older manufacturing process technology (e.g., 20 nm to 24 nm) exhibits a linear trend for the distribution mean [23], just as we see for 3D NAND. Thus, we believe that when the manufacturing process technology scales below a certain size, the change in the distribution mean transitions from linear behavior to power-law behavior. As a result, when future 3D NAND scales down to a sub-20 nm manufacturing process technology, we expect that it too will exhibit power-law behavior.

Program Interference

When a cell (which we call the *aggressor cell*) is being programmed, cell-to-cell program interference can cause the threshold voltage of nearby flash cells (which we call *victim cells*) to increase unintentionally [24, 26] (see Section 3.1). In 3D NAND, there are two types of program interference that can occur. The first, *wordline-to-wordline program interference*, affects victim cells along the z-axis from the cell being programmed (see Figure 3.27). These victim cells are physically next to the cell being programmed, and belong to the same bitline (and thus the same flash block). The second, *bitline-to-bitline program interference*, affects victim cells along the x-axis or y-axis from the cell being programmed. Bitline-to-bitline program interference can affect victim cells in the same wordline (i.e., cells on the y-axis), or it can affect victim cells that belong to other flash blocks (i.e., cells on the x-axis).

To quantitatively analyze the effect of program interference, we use the same experimental data that we have for program/erase variation errors (see Section 6.3.1). The amount of program interference on a victim cell correlates with the threshold voltage change of the aggressor cell [24]. This *interference correlation* makes the threshold voltage of a victim cell dependent on the value of the aggressor cell. The strength of this correlation can be quantified as $\frac{\Delta V_{victim}}{\Delta V_{aggressor}}$, which is a property of the NAND device and is largely dependent on the distance between the cells [174]. We estimate $\Delta V_{aggressor}$ by calculating the threshold voltage difference between the aggressor cell's final state and the ER state. We estimate ΔV_{victim} by calculating the difference between the victim cell's threshold voltage with and without program interference.

Observations. Figure 6.11 shows the interference correlation for a victim cell, *as a result of programming on aggressor cells of varying distances and directions.* We make two observations from this figure. First, for a victim cell (i.e., the cell in BL M, WL N in Figure 6.11), program interference is dominated by wordline-to-wordline interference from the next wordline (i.e., BL M, WL N+1). Second, all of the other types of interference have a much smaller interference correlation.

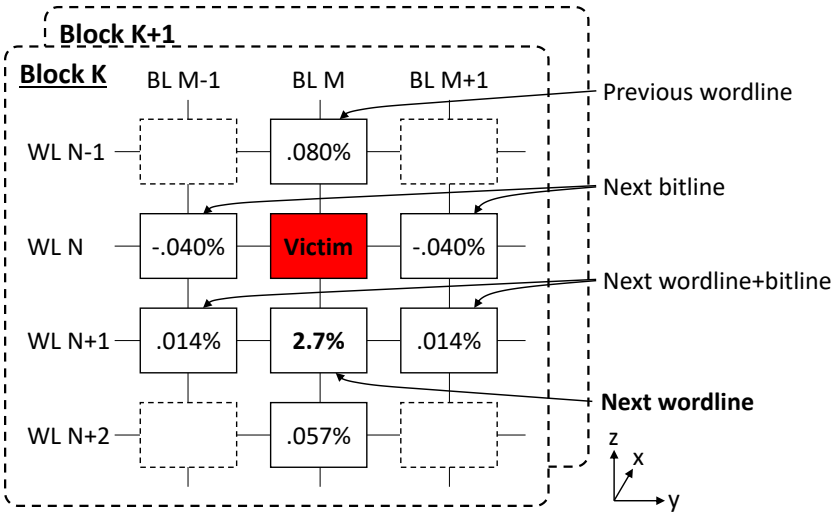


Figure 6.11: Interference correlation for a victim cell, as a result of programming on aggressor cells of varying distance.

Figure 6.12 shows the how much the threshold voltage of a victim cell shifts when a neighboring aggressor cell is programmed to the P3 state, which generates the largest possible program interference. We separate the threshold voltage shifts by both the interference type and the state of the victim cell. We ignore any case where the shift is less than 1 voltage step. This figure also shows how the threshold voltage shifts due to program interference changes with the P/E cycle count. We make three observations from Figure 6.12. First, the effect of program interference decreases as the P/E cycle count increases. Second, the program interference induced by an aggressor cell in the next wordline decreases when the victim cell is in a higher-voltage state. Third, the program interference induced by an aggressor cell in the previous wordline (i.e., WL N-1 in Figure 6.11) affects the threshold voltage distribution of the ER state for a victim cell, but it has little effect on the distributions of the other three states (i.e., P1, P2, P3).

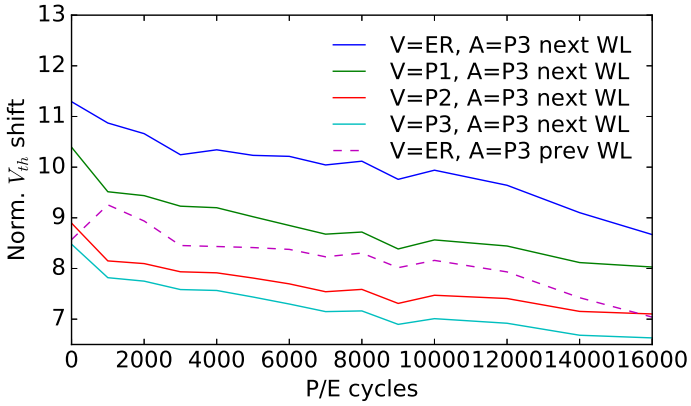


Figure 6.12: Interference vs. P/E cycle.

Insights. We compare the program interference in 3D NAND to the program interference observed in planar NAND, as reported in prior work [24, 26]. We find one major difference between them. The interference correlation of program interference from a directly-adjacent cell in 3D NAND is 40% lower than the interference correlation in planar NAND flash memory, which is around 4.5% in 20 nm to 24 nm planar NAND flash memory [24]. A similar conclusion is drawn by prior work [257] which shows that 3D NAND has 84% lower program interference than 15 nm to 19 nm planar NAND flash memory. However, this reduction in interference correlation is due to result the larger manufacturing process technology used in the current generation of 3D NAND. We are unable to find literature that reports program interference correlation in 30 nm to 50 nm planar NAND, which could provide a direct comparison for how we expect interference to change in 3D NAND. However, prior work has shown that the strength of interference correlation between neighboring cells is correlated with the distance between the cells [174]. Thus, as 3D NAND scales to a smaller manufacturing process technology, the cells move closer to each other, and the program interference effect increases, as it did in planar NAND flash memory.

Note that we are the first to compare how the threshold voltage shift caused by program interference changes with the P/E cycle count. As we discuss in our first observation for Figure 6.12, the program interference effect surprisingly decreases as the P/E cycle count increases. This observation is a result of the ER state distribution shift due to wearout (see Section 6.3.1). As the threshold voltage distribution of the ER state shifts higher as the P/E cycle count increases, the difference in voltage between the ER state and the P3 state reduces. This mitigates $\Delta V_{aggressor}$ and thus also reduces ΔV_{victim}, which thus reduces the interference correlation.

6.3.2 Early Retention Loss

In this section, we present the results and analysis of retention loss in 3D NAND in addition to the key findings from Section 6.2.2. We use the same methodology as described in Section 6.2.2.

Observations. Figure 6.13 shows the how RBER increases with retention age for a block at 10K P/E cycles. The top graph breaks down the errors into MSB and LSB page errors; the bottom figure breaks down the errors according to the change in cell state as a result of the errors. We make four observations from Figure 6.13. First, the logarithm of RBER is linearly correlated

with the logarithm of retention age. Second, the MSB error rate increases faster than the LSB error rate as the retention age increases. Third, retention errors are dominated by P2 $\leftrightarrow$ P3 and P1 $\leftrightarrow$ P2 errors. Fourth, the error rate between any two neighboring states increases with retention age. We see that ER $\leftrightarrow$ P1 and P2 $\leftrightarrow$ P3 errors increase faster than P1 $\leftrightarrow$ P2 errors.

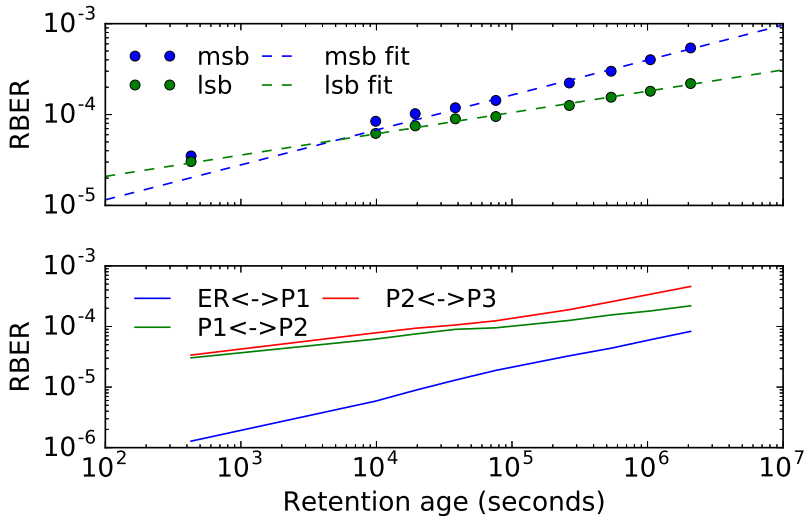


Figure 6.13: RBER variation across retention age, broken down by (1) MSB or LSB page, and by (2) the state transition of each flash cell.

Figures 6.14 and 6.15 show how the mean and the standard deviation, respectively, of the Gaussian model for retention errors changes with retention age. Each subfigure shows the parameter for a different state, as labeled on the top. We make five observations from these two figures. First, the correlation between any distribution parameter (P) and the retention age (t) can be modeled as: $P = A \cdot \log(t) + B$. Second, the threshold voltage distribution shifts much faster when the retention age is low. Third, the distributions of the P1, P2, and P3 states shift lower with retention age, but the distribution of the ER state shifts higher. Fourth, the distributions of the ER and P3 states shift faster than the distributions of the P1 and P2 states as the retention age increases. Fifth, retention has little effect on the distribution width.

Insights. We compare the errors due to retention loss in 3D NAND to that in planar NAND flash memory, as reported in prior work [22, 27, 219]. We find another major differences in 3D NAND in terms of threshold voltage distribution in addition to those discussed in Section 6.2.2. We find that retention loss in 3D NAND shifts the threshold voltage distributions of the P1, P2 and P3 programmed states lower, and has little effect on the width of the distribution of each state. In contrast, the retention loss of planar NAND flash memory does *not* shift the P1 and P2 state distributions by much, and the retention loss increases the width of each state's distribution *significantly* [27]. This indicates that a mechanism that adjusts the optimal read reference voltage to the threshold voltage shift caused by retention can be more effective on 3D NAND than on planar NAND.

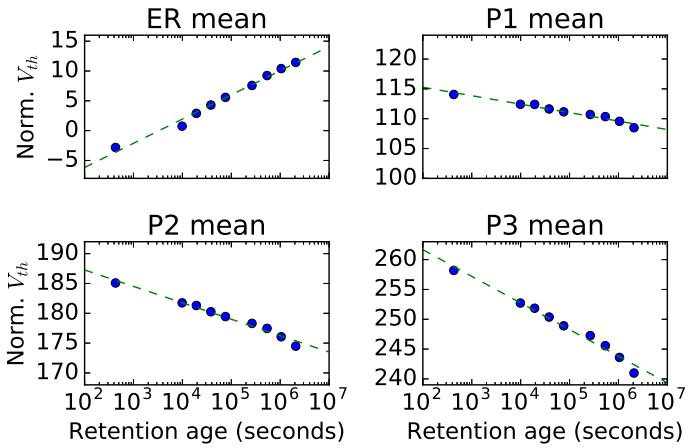


Figure 6.14: Mean of distribution for retention loss error model, as retention age increases.

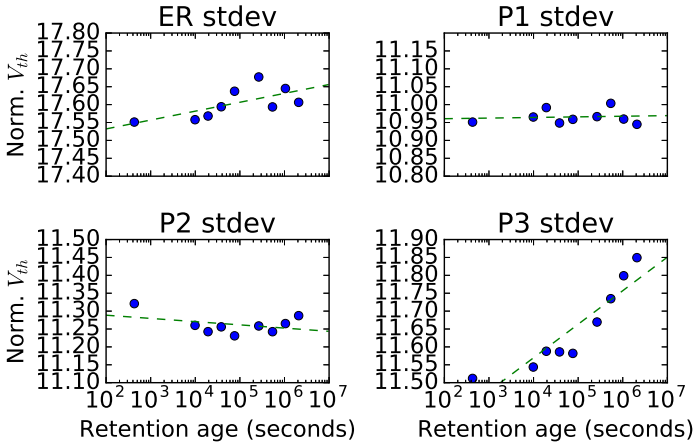


Figure 6.15: Standard deviation of distribution for retention loss error model, as retention age increases.

6.3.3 Read-Induced Errors

We now analyze how each type of read-induced error affects the RBER and the threshold voltage distribution of 3D NAND flash memory.

Read Variation Errors

Read variation errors are random raw flash errors that happen due to random fluctuation of the sense amplifier (see Section 3.1). Read variation errors are not well-studied by prior work. However, since they add uncertainty to the outcome of each read operation, read variation errors are important because they can affect threshold voltage characterization result if not handled properly.

To quantify the read variation errors, we simply use the data in Section 6.2.2. Since we sweep the read reference voltage to obtain V_{th} of each cell, we can simulate reads by comparing V_{ref} to V_{th} directly. Since the V_{th} characterization involves multiple reads, the simulated reads are inherently resistant to read variation errors. Thus, we can identify read variation errors by comparing the value actually read out from the flash chip with the value from the simulated read.

Observations. Figure 6.16 shows the correlation between the read variation error rate and the total RBER for LSB and MSB page errors. We make two observations from this figure. First, the read variation error rate for both LSB and MSB pages is linearly correlated with the overall RBER. Second, the MSB page has a higher read variation error rate than the LSB page.

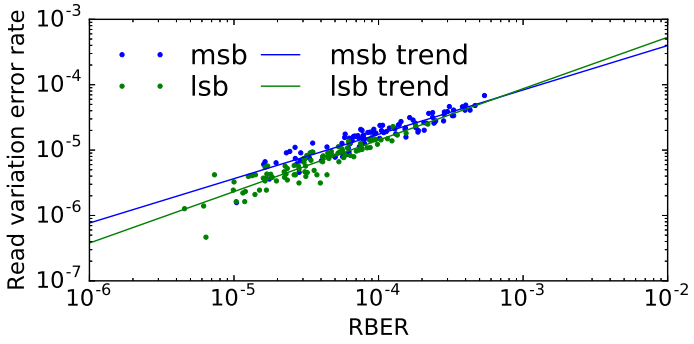


Figure 6.16: Read variation error, varying over the RBER.

Figure 6.17 shows the correlation between the read variation error rate and the *read offset*, which is $V_{ref} - V_{th}$. We observe that as the read offset increases, the read variation error rate decreases exponentially. This corroborates the first observation from Figure 6.16 because, when the RBER is high, the threshold voltage distributions of neighboring states overlap with each other by a greater amount. This causes a larger number of cells to be close to the read reference voltage value, increasing the probability that a read variation error occurs.

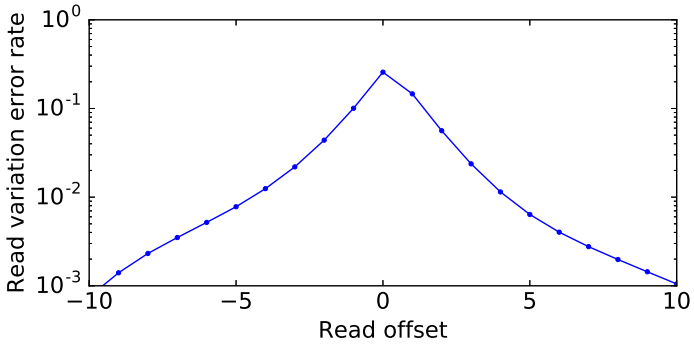


Figure 6.17: Read variation error vs. read offset.

Insights. We are the first to discover and quantify the extent of read variation errors, and to show the correlation of these errors with the RBER and with the read reference voltage. For cells with a threshold voltage close to the read reference voltage, it is more difficult for the sense amplifier to detect whether the cell is on or off (due to the small delta), which causes the random fluctuations. We find that read variation errors are one of the dominant sources of errors in 3D NAND flash memory.

Read Disturb Errors

Read disturb errors accumulate in a cell when any other cell on the same bitline is read [35, 252] (see Section 3.1). Read disturb errors are caused by the high pass-through voltage applied on the unread cells.

To characterize read disturb errors, we first randomly select 11 flash blocks and use random data to wear out each block to 0 to 10K P/E cycles. Then, we program random data to each flash block. To minimize the impact of other errors, especially retention errors due to early retention loss, we wait until the data has 2-day retention age before inducing read disturb. This ensures that, according to our results in Section 6.2.2, retention loss can only shift the distribution of each state by at most 1 voltage step during the characterization process. To induce read disturb in the flash block, we repeatedly read from a wordline within the block for up to 900K times (i.e., up to 900K read disturbs). During this process, to characterize read disturb effect, we obtain the RBER and threshold voltage distribution at ten different read disturb counts from 0 to 900K.

Observations. Figure 6.18 plots how RBER increases over read disturb under 10K P/E cycle. The top figure breaks down raw bit errors into LSB and MSB page errors; the bottom figure breaks down the errors according to V_{th} state transition. We make three observations from Figure 6.18. First, LSB and MSB errors can be modeled as a linear function of the read disturb count. Second, LSB errors increase faster than MSB errors. Third, the increase in LSB error is caused by the significant increase of ER↔P1 errors, while P2↔P3 errors slightly decrease over read disturb.

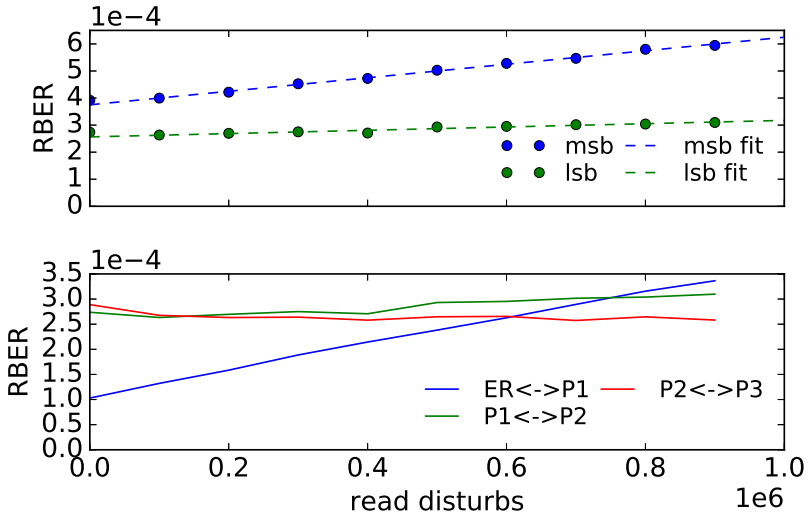


Figure 6.18: RBER vs. read disturb counts.

Figures 6.19 and 6.20 show how the distribution parameters change over read disturb. Each subfigure shows the parameter for a different state, labeled on the top. We make four observations from these two figures. First, the change in distribution parameters can be modeled as a linear function of read disturb counts. Second, the ER state distribution shifts significantly higher over read disturb, whereas the programmed states does not shift much over read disturb. The increase in the distribution mean is lower for a higher V_{th} state. In fact, P3 state distribution even shifts slightly lower. Third, the distribution width of each state (i.e., standard deviation) decreases slightly over read disturb counts. Fourth, the distribution width of a higher V_{th} state decreases faster than that of a lower V_{th} state.

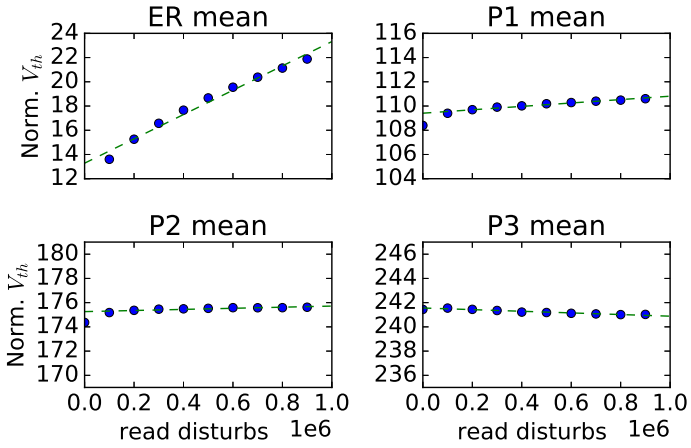


Figure 6.19: Distribution mean vs. read disturb counts.

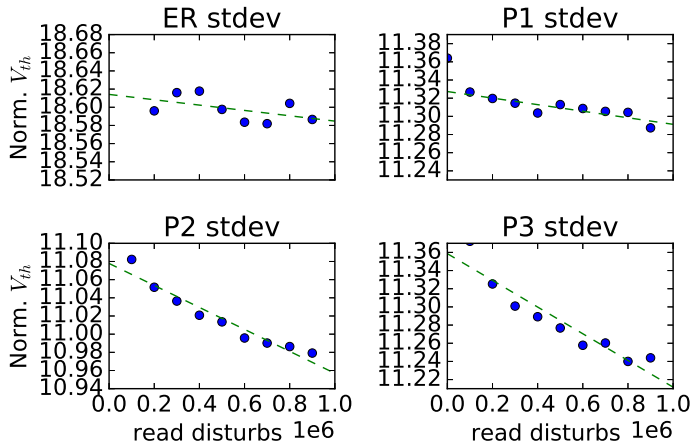


Figure 6.20: Distribution standard deviation vs. read disturb counts.

Figure 6.21 shows how the optimal read reference voltages change due to read disturb. Three subfigures show the optimal voltages for V_a, V_b, and V_c. We make two observations from this figure. First, the optimal voltage for V_a increases linearly over read disturb. Second, the optimal voltages of V_b and V_c change by less than 3 voltage steps over 900K read disturbs.

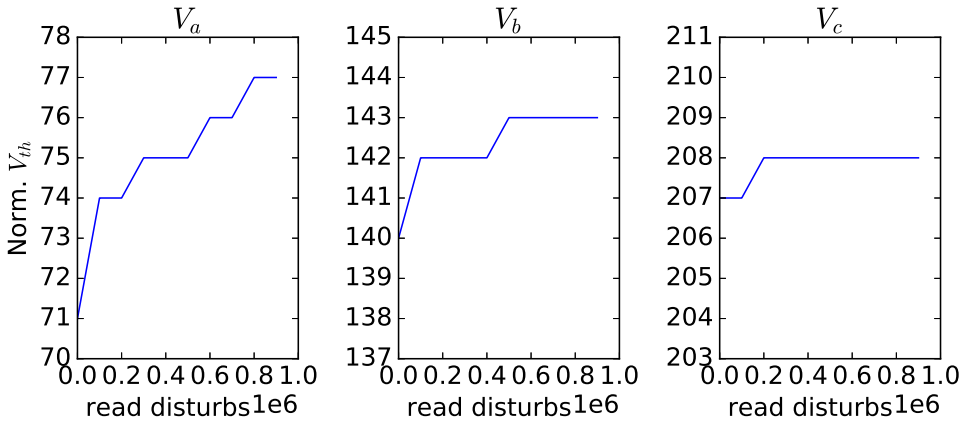


Figure 6.21: Optimal read reference voltages vs. read disturb counts.

Figure 6.22 plots the slopes of RBER vs. read disturb (e.g., the slopes of the fitted curves in Figure 6.18) at different P/E cycles. We show the slope separately for LSB and MSB RBER. The steepness of the slope shows the sensitivity of RBER to read disturb effect. We make two observations from this figure. First, the rate of LSB errors increase much faster than the rate of MSB errors as the number of read disturbs increases. Second, for both LSB and MSB errors, the slope increases super-linearly as the P/E cycle count increases.

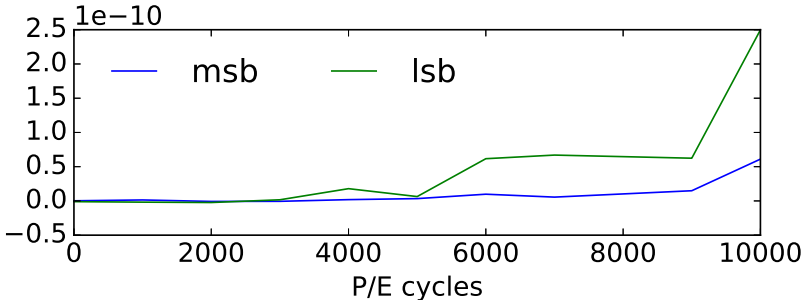


Figure 6.22: Read disturb error increase rate vs. P/E cycle.

Figure 6.23 plots the slopes of distribution mean vs. read disturb (e.g., the slopes of the fitted curves in Figure 6.19) at different P/E cycles. Each subfigure shows the slope for a different state, labeled on the top, reflecting how fast the distribution shifts by read disturb. We make two observations from this figure. First, the slope for each state increases linearly over P/E cycles. Second, the slope for ER state increases significantly over P/E cycles, whereas the slopes for the programmed states increase by little.

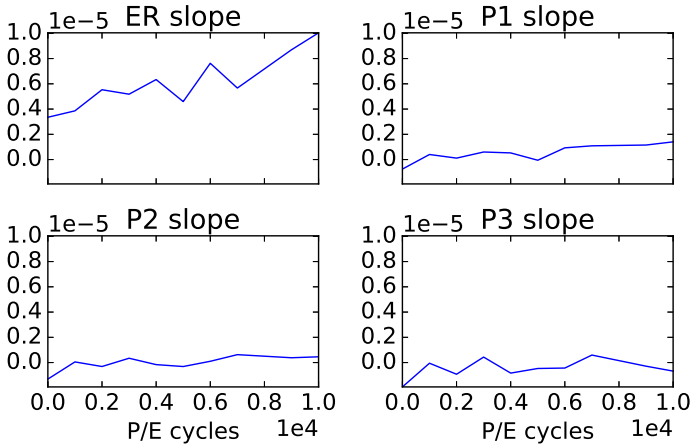


Figure 6.23: Distribution mean increase rate vs. P/E cycle.

Insights. We compare the read disturb effect in 3D NAND to that in planar NAND reported in prior work [35]. We make the observation that, although RBER increase linearly over read disturb in both 3D NAND and planar NAND, the slope of increase (i.e., the sensitivity to read disturb) at 10K P/E cycles is 96.7% lower in 3D NAND than in planar NAND [35]. We believe that this difference in sensitivity to read disturb is due to the use of a larger process technology in 3D NAND, 30 nm to 40 nm. In contrast, the results from prior work are for a 20 nm to 24 nm planar NAND. Thus, we expect the sensitivity to read disturb in 3D NAND to increase in the future as we shrink the process technology.

6.3.4 Layer-To-Layer Process Variation

In this section, we present the results and analysis of process variation in 3D NAND in addition to the key findings from Section 6.2.1. We use the same methodology as described in Section 6.2.1.

Figures 6.24 and 6.25 show the change in threshold voltage distribution mean and standard deviation of each state, respectively, for a model of layer-to-layer process variation, for a flash block with 10K P/E cycles. Each subfigure shows the parameter for a different state, as labeled on the top. We make three observations from these two figures. First, the ER state distribution is shifted by as much as 25 voltage steps across layers, while the distributions of the other three states do not shift by much. Second, the mean of the ER state keeps increasing in the top half of the layers, but remains constant in the bottom half. Third, the distribution width of the P1 state increases in the top half of the layers, and decreases in the bottom half. However, the distribution widths of the P2 and P3 states vary by a smaller amount.

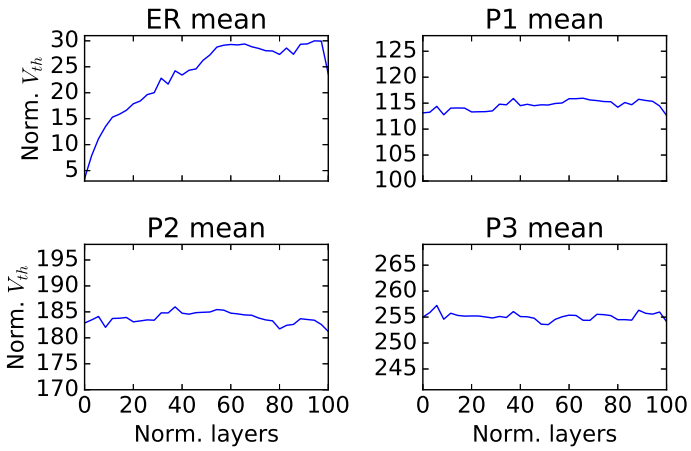


Figure 6.24: Layer-to-layer variation of distribution mean.

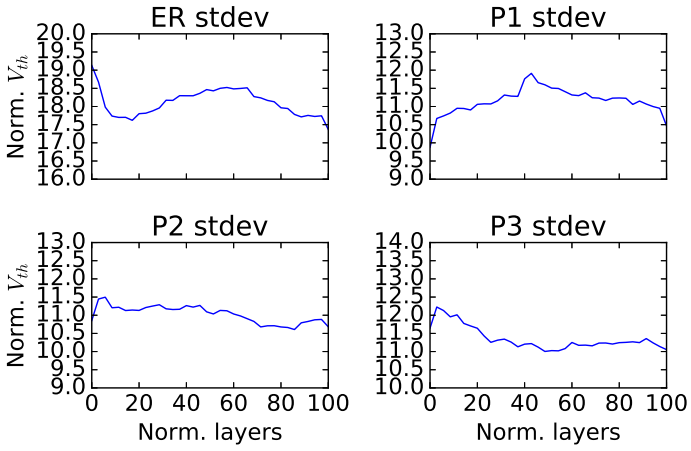


Figure 6.25: Layer-to-layer variation of distribution width.

6.3.5 Bitline-to-Bitline Process Variation

We perform a similar analysis on the variation of RBER and threshold voltage distribution along the y-axis (i.e., across groups of bitlines) for a flash block with 10K P/E cycles. Figure 6.26 shows the RBER variation along the y-axis for a flash block with 10K P/E cycles. To reduce the impact of random process variation noise, we normalized the bitlines from 0 to 88 by averaging the RBER of every ∼3000 neighboring bitlines. The top graph breaks down the errors into MSB and LSB page errors; the bottom graph breaks down the errors according to the original and current state of each cell. Figure 6.27 shows change in threshold voltage distribution mean of each state along the y-axis. Each subfigure shows the parameter for a different state, as labeled on the top. We make two observations from this analysis. First, the variation along the y-axis is much smaller compared to the variation across the z-axis. Second, for every ∼64K consecutive bitlines within a flash block, we observe similar process variation, indicating some form of repetition.

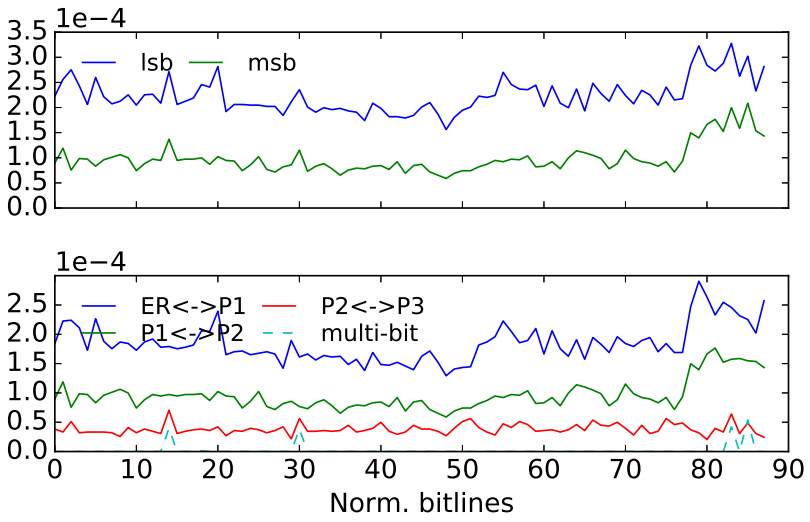


Figure 6.26: RBER variation across bitline.

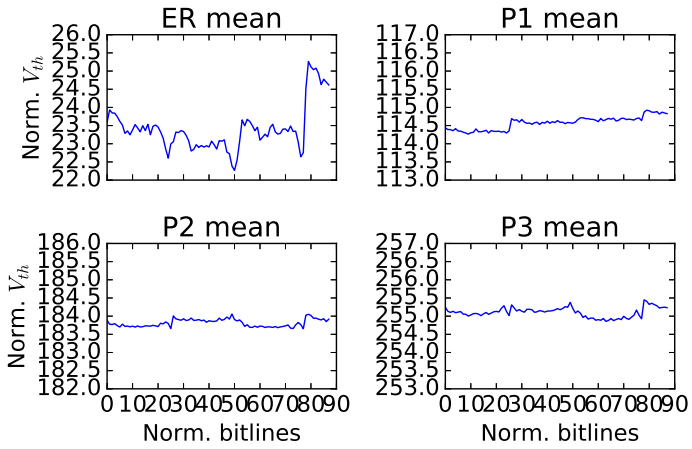


Figure 6.27: Distribution mean variation across bitlines.

6.4 3D NAND Error Models

In previous sections, we have established a basic understanding of the similarities and differences between 3D NAND and planar NAND flash memory in terms of reliability. In this section, we quantify these differences by developing analytical models of the process variation (Section 6.4.1) and retention loss (Section 6.4.2) in 3D NAND flash memory. The insights from these models motivate us to develop new error mitigation mechanisms for 3D NAND flash memory. The retention model and the model parameters are also useful for comparing the reliability of newer or older generations of NAND flash memory with our tested 3D NAND flash chips.

6.4.1 Process Variation Model

We first construct a model of the optimal read reference voltage (V_{opt}) variation across different layers, based on the results in Section 6.2.1. Since there are only a limited number of layers, we use a non-parametric model that estimates the offset between the V_{opt} on each layer and the overall V_{opt} for the entire flash block using the results across 10 different P/E cycles. The result of the model is shown in Figure 6.3. Since the overall per-block V_{opt} does not take layer-to-layer process variation, we call it *variation-agnostic* V_{opt}. The per-layer V_{opt} can be calculated by adding the modeled offset to the variation-agnostic V_{opt}, which accounts for the process variation. Thus we call it *variation-aware* V_{opt}. Many prior works already provide models and mechanisms to obtain variation-agnostic V_{opt} at a low cost [27, 195, 252].

Since the layer-to-layer variation in 3D NAND causes variation in RBER within a flash block, the overall RBER is no longer sufficient to represent the reliability of all pages within that block. Instead, we model the variation in per-page RBER within a flash block as a gamma distribution (i.e., $gamma(x,\theta,k) = \frac{x^{k-1}e^{-\frac{x}{s}}}{\Gamma(k)\theta^a}$). Figure 6.28 shows the probability density for per-page RBER within a block with 10K P/E cycles. The bars show the measured per-page RBERs categorized into 50 bins, and the curves are the fitted gamma distributions whose parameters are shown on the legend. The blue and red colors represent the RBER distributions when reading

with the variation-agnostic V_{opt} and with the variation-aware V_{opt}, respectively. We make two observations from this figure. First, the average RBER is reduced from $1.6 \cdot 10^{-4}$ to $1.4 \cdot 10^{-4}$ by applying variation-aware V_{opt}. Second, a few flash pages have a much higher RBER than the average RBER (e.g., $> 4 \cdot 10^{-4}$). These pages are LSB pages that reside in the middle layers, as is shown in Figure 6.2.

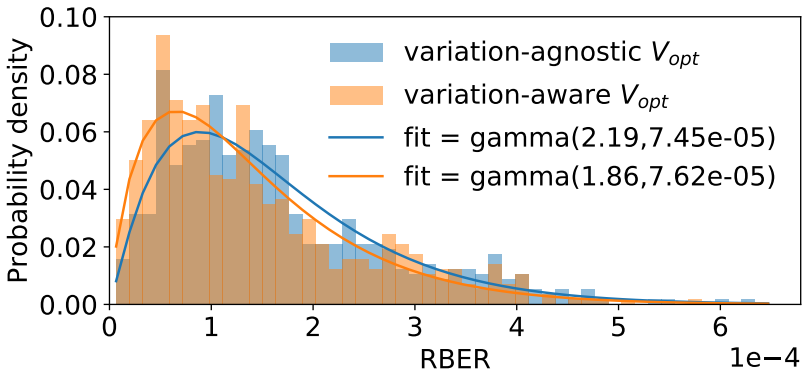


Figure 6.28: RBER distribution within a flash block.

6.4.2 Retention Loss Model

To quantify the retention loss in 3D NAND, we construct a model of retention loss in 3D NAND as a function of retention time (t) and P/E cycle lifetime (PEC). We use multivariate linear regression to fit the model and use root mean square error as the loss function. Following the observations in Section 6.2.2, we perform natural logarithm transformation on RBER and retention time. Also following the observations, we break down our model into two steps. The first step models the retention loss at a certain P/E cycle as a logarithmic function of retention time. The second step models how P/E cycle change the parameters of retention loss.

Table 6.2 shows all of the parameters we use to model the RBER and the threshold voltage as a function of the P/E cycle count (PEC) and the retention time (t). In this table, the first column shows the modeled variable for each row. The second column shows how we model retention loss, as we have shown in Figures 6.13, 6.14 and 6.15. The third and fourth columns show how each parameter in the retention model changes over P/E cycles. The last column shows the root mean square error of our model.

Variable	**Retention model**		Model parameters	**Model error**
MSB RBER	$\log(RBER) = A \cdot \log(t) + B$	$A = 7.1 \cdot 10^{-6} \cdot PEC + 0.34$	$B = 4.1 \cdot 10^{-5} \cdot PEC - 13.83$	$2.9 \cdot 10^{-5}$
LSB RBER	$\log(RBER) = A \cdot \log(t) + B$	$A = 6.6 \cdot 10^{-6} \cdot PEC + 0.16$	$B = 1.2 \cdot 10^{-4} \cdot PEC - 13.05$	$8.2 \cdot 10^{-6}$
ER mean	$mean = A \cdot \log(t) + B$	$A = 1.0 \cdot 10^{-4} \cdot PEC + 0.75$	$B = 1.5 \cdot 10^{-3} \cdot PEC - 27.28$	1.60
P1 mean	$mean = A \cdot \log(t) + B$	$A = -1.9 \cdot 10^{-5} \cdot PEC - 0.40$	$B = 3.5 \cdot 10^{-4} \cdot PEC + 114.47$	0.27
P2 mean	$mean = A \cdot \log(t) + B$	$A = -4.7 \cdot 10^{-5} \cdot PEC - 0.71$	$B = 3.2 \cdot 10^{-4} \cdot PEC + 189.58$	0.30
P3 mean	$mean = A \cdot \log(t) + B$	$A = -7.4 \cdot 10^{-5} \cdot PEC - 1.21$	$B = 5.8 \cdot 10^{-4} \cdot PEC + 264.86$	0.53
ER stdev	$stdev = A \cdot \log(t) + B$	$A = 1.2 \cdot 10^{-5} \cdot PEC - 0.11$	$B = 1.6 \cdot 10^{-6} \cdot PEC + 17.02$	0.39
P1 stdev	$stdev = A \cdot \log(t) + B$	$A = -1.3 \cdot 10^{-6} \cdot PEC + 9.8 \cdot 10^{-3}$	$B = 7.6 \cdot 10^{-5} \cdot PEC + 10.21$	0.05
P2 stdev	$stdev = A \cdot \log(t) + B$	$A = -2.1 \cdot 10^{-6} \cdot PEC + 9.9 \cdot 10^{-3}$	$B = 6.7 \cdot 10^{-5} \cdot PEC + 10.66$	0.047
P3 stdev	$stdev = A \cdot \log(t) + B$	$A = 2.9 \cdot 10^{-6} \cdot PEC + 0.014$	$B = 3.3 \cdot 10^{-5} \cdot PEC + 10.83$	0.059
V_a	$V_a = A \cdot PEC + B$	$A = 1.2 \cdot 10^{-3}$	$B = 60.52$	2.36
V_b	$V_b = A \cdot \log(t) + B$	$A = 1.51 \cdot 10^{-6} \cdot PEC - 0.75$	$B = 152.67$	0.55
V_c	$V_c = A \cdot \log(t) + B$	$A = 2.43 \cdot 10^{-5} \cdot PEC - 1.27$	$B = 229.65$	0.64

Table 6.2: Model parameters for 3D NAND retention loss. t is retention time, PEC is P/E cycle lifetime.

6.5 3D NAND Error Mitigation Techniques

Motivated by our new findings in Section 6.1, we aim to design new techniques that mitigate the three unique error effects in 3D NAND flash memory. We propose four error mitigation mechanisms. To mitigate layer-to-layer process variation, we propose LaVAR and LI-RAID. LaVAR learns the process variation model we developed in Section 6.4.1 online in the SSD controller, and uses this model to predict and apply a optimal read reference voltage that is fine-tuned for each layer (Section 6.5.1). LI-RAID is a new RAID scheme that reduces the RBER variation induced by layer-to-layer process variation in 3D NAND flash memory (Section 6.5.2). To mitigate retention loss in 3D NAND flash memory, we propose ReMAR, a new technique that tracks the retention age information within the SSD controller and uses the retention model we developed in Section 6.4.2 to predict and apply the optimal read reference voltage fine-tuned for the retention age of the data (Section 6.5.3). To mitigate retention interference, we propose ReNAC, which is adapted from an existing technique originally designed to reduce program interference in planar NAND to also account for retention interference in 3D NAND flash memory (Section 6.5.4).

6.5.1 LaVAR: Layer Variation Aware Reading

Motivation: In planar NAND, existing techniques assume that V_{opt} is the same across all pages within a flash block [27, 252]. However, as our results in Section 6.2.1 show, this assumption no longer works in 3D NAND because of layer-to-layer process variation. To mitigate this problem in 3D NAND, we devise a new mechanism, LaVAR, that uses the process variation model we developed in Section 6.4.1 to fine-tune read reference voltage in the flash controller.

Mechanism: The key idea of LaVAR is to identify which layer the flash controller is reading and apply the variation-aware V_{opt} predicted by the model in Section 6.4.1. Since the process variation is similar across blocks and is consistent across P/E cycles, the V_{opt} model can be learned *offline* for each chip through an extensive characterization of a single flash block. To do this, the SSD controller randomly picks a flash block and records the difference between variation-agnostic V_{opt} and per-block variation-aware V_{opt}. The controller then computes and stores the average V_{opt} difference for each layer in a lookup table. Note that V_c variation does not need to be modeled, since V_c is unaffected by layer-to-layer process variation. After this lookup table is constructed, the SSD controller simply looks up the V_{opt} difference according to the layer of each read operation and add the offset to the per-block V_{opt} predicted by existing techniques [27, 195, 252]. By applying variation-aware V_{opt}, LaVAR applies a more accurate V_{opt} for 3D NAND than existing techniques, and reduces the RBER.

Overhead: Assuming that the 3D NAND has N layers and V_{opt} difference is 1 $Byte$, the memory overhead of caching the lookup table in the SSD controller is $2N$ Bytes. The latency overhead of each read operation is negligible as LaVAR only requires a table lookup and an addition to obtain variation-aware V_{opt}, which take less than $100ns$. Since the lookup table is shared across all blocks, it only needs to be learned once, and can be finished gradually in the background. Thus the performance overhead is negligible.

Evaluation: Figure 6.29 compares the RBER achieved by LaVAR (`process variation aware`) to that by an existing read reference voltage tuning technique (`process variation agnostic`) designed for planar NAND [27, 195, 252] that applies a per-block V_{opt}. We evaluate

the average RBER achieved by each mechanism by simulating reads using our characterization data in Section 6.2.1. On average, LaVAR reduces RBER by 43.3%. The benefit comes from tuning the read reference voltage towards the per-page optimal read reference voltage by an offset learned by our model. The percentage of RBER reduction becomes smaller as P/E cycle increases because the overall RBER increases exponentially as the flash memory wears out, decreasing the fraction of process variation errors.

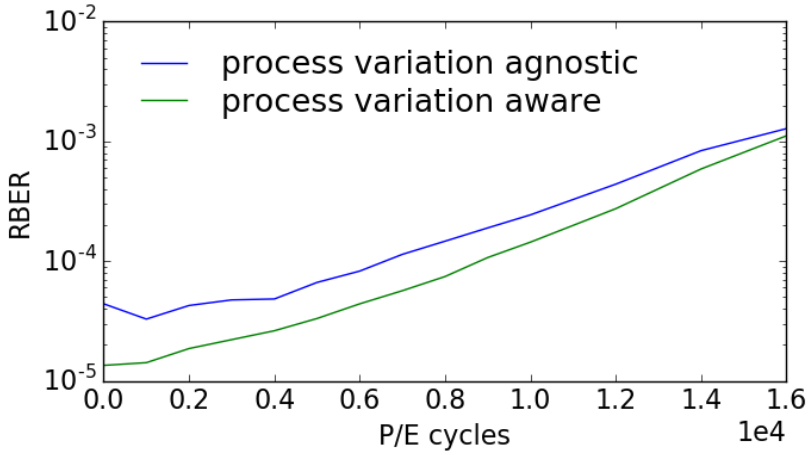


Figure 6.29: RBER reduction using process-variation-aware optimal read reference voltage.

6.5.2 LI-RAID: Layer-Interleaved RAID

Motivation: As we show in Section 6.4.1, even after applying the variation-aware V_{opt}, the worst-case per-page RBER is more than $2\times$ higher than the average per-block RBER due to layer-to-layer process variation. In addition, neighboring MSB pages and LSB pages also have very different RBERs and different tolerance to different types of errors, as shown in Sections 6.2.1 and 6.2.2. In enterprise SSDs, RAID [9, 262] is applied in addition to ECC across multiple flash chips to tolerate process variation across chips. However, state-of-the-art RAID schemes do not consider layer-to-layer variation and MSB–LSB variation. These schemes group MSB or LSB pages in the same layer together in a RAID group. As a result, the reliability of the SSD is limited by the RBER of the weakest (i.e., the least reliable) RAID group that contains the MSB or LSB pages from the least reliable layer across all chips. We aim to develop a new RAID scheme that eliminates these low-reliability RAID groups by equalizing the RBER among different RAID groups.

Mechanism: We propose LI-RAID, a new RAID scheme that can tolerate the process variation across layers in addition to the variation across chips. Instead of grouping pages in the same layer together, we select pages from different chips and different layers and group them together, such that the pages from the least reliable layer are *interleaved* across multiple RAID groups. Thus, the new groups formed by LI-RAID have a more evenly-distributed reliability than the groups formed using traditional RAID schemes. We assume, without loss of generality, that there are

m chips in the SSD, and each RAID group contains m pages, one from each chip. We also assume that each block contains n wordlines, and that the layer numbers of each wordline are in ascending order (e.g., the wordline in layer i has a lower wordline number than its neighboring wordline in layer $i+1$). Thus, LI-RAID groups together the MSB page of wordline 0, the LSB page of wordline $\frac{n}{m}$, the MSB page of wordline $2 \cdot \frac{n}{m}$, ..., the MSB page of wordline $(m-1) \cdot \frac{n}{m}$. Figure 6.30 shows an example LI-RAID layout on an SSD with 4 chips and with 4 wordlines within each flash block. Flash pages in the same RAID group are highlighted in the same color. In this way, LI-RAID redistributes the less reliable pages within each chip across different RAID groups, eliminating the worst-case RAID groups that bottlenecks SSD reliability.

Page/Wordline	Chip 0	Chip 1	Chip 2	Chip 3
MSB/Wordline 0	Group 0	Blank	Group 4	Group 3
LSB/Wordline 0	Group 1	Blank	Group 5	Group 2
MSB/Wordline 1	Group 2	Group 1	Blank	Group 5
LSB/Wordline 1	Group 3	Group 0	Blank	Group 4
MSB/Wordline 2	Group 4	Group 3	Group 0	Blank
LSB/Wordline 2	Group 5	Group 2	Group 1	Blank
MSB/Wordline 3	Blank	Group 5	Group 2	Group 1
LSB/Wordline 3	Blank	Group 4	Group 3	Group 0

Figure 6.30: LI-RAID layout example for an SSD with 4 chips and with 4 wordlines in each flash block.

Note that for some chips, the LI-RAID layout may potentially violate the program sequence recommended by flash vendors, where wordlines within each flash block must be programmed *in order* to minimize harmful program interference [24, 26, 33, 255]. For example, in Chip 2 in Figure 6.30, Wordline 0 is programmed *after* Wordline 2. If we were to write to Wordline 2 after Wordline 1 is programmed, Wordline 2 would experience approximately *double* the program interference (i.e., from programming both Wordline 3 and Wordline 1). To avoid introducing any additional program interference, we deliberately leave the flash pages in the last RAID group *empty*, which are shown as "Blank" pages shaded in gray in Figure 6.30. As a result, LI-RAID provides the same reliability guarantee as the recommended program sequence, by guaranteeing that any data stored in a flash page experiences program interference from at most one neighboring wordline.

Overhead: The wordlines left blank in LI-RAID incur a small additional storage overhead compared to a conventional RAID scheme. Only *one* wordline within a flash block is left blank. In modern NAND flash memory, each flash block typically contains more than 250 flash pages. Thus, the additional storage overhead is less than 0.4%. LI-RAID does not incur additional computational overhead because it computes parity in the same way as a conventional RAID, and only reorganizes the RAID groups differently. Because we do not change the data layout across flash blocks, the flash transation layer (FTL) and the garbage collection (GC) algorithm remain the same as in a conventional RAID scheme.

Evaluation: Figure 6.31 plots the 99th-percentile RBER when applying different error mitigation techniques at 10,000 P/E cycles. As we show in Figure 6.28, due to layer-to-layer variation, per-page RBER is distributed over a wide range according to a gamma distribution. Thus, several worst-case flash pages within a block may become unusable (i.e., their RBER exceeds the ECC correction capability) before the *overall* RBER of the flash chip exceeds the ECC correction capability. We use the 99th-percentile RBER to represent the reliability of these worse-case flash pages. In this figure, the baseline applies the per-block variation-agnostic optimal read reference voltage (i.e., variation-agnostic V_{opt}), achieving a 99th-percentile RBER of $4.8 \cdot 10^{-4}$. When we apply the variation-aware V_{opt} proposed in Section 6.5.1, the 99th-percentile RBER is reduced by 9.6% over the baseline, to $4.3 \cdot 10^{-4}$. LI-RAID reduces the 99th-percentile RBER by 66.9% over the baseline, to only $1.6 \cdot 10^{-4}$. Thus, by grouping flash pages on less reliable layers with pages on more reliable layers, and by grouping MSB pages with LSB pages, LI-RAID reduces the probability of unusable pages within a block, reducing the number of retired flash blocks due to ECC failures. This greatly reduces the probability of data loss and increases the available storage space near the end of the SSD lifetime.

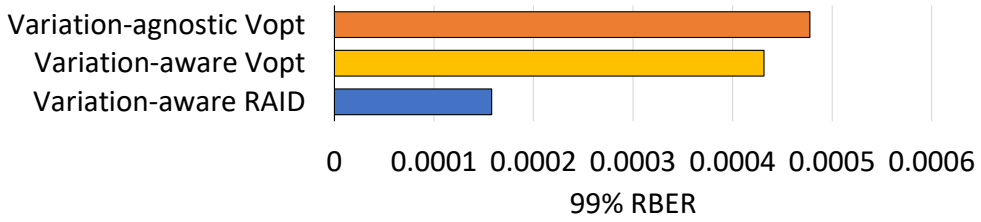


Figure 6.31: Worst-case RBER at 10,000 P/E cycles when applying different error mitigation techniques.

6.5.3 ReMAR: Retention Model Aware Reading

Motivation: As we show in Section 6.2.2, due to early retention loss, retention errors increase much faster in 3D NAND than in planar NAND. Thus, mitigating retention errors has become more important in 3D NAND than in planar NAND, as it has greater impact on SSD reliability. However, as we show in our model in Section 6.4.2, early retention loss is proportional to the logarithm of time. This means that the majority of the retention errors and threshold voltage shifts happen shortly after programming. As a result, traditional retention mitigation techniques developed for planar NAND may become less effective on 3D NAND. For example, FCR [22, 250], a mechanism that remaps all data periodically, allows planar NAND to tolerate 50 $\times$ more P/E cycles with 3-day refresh period. But this number reduces to only 2.7 $\times$ for 3D NAND due to early retention loss. Figure 6.32 shows how increasing refresh period exponentially only improves 3D NAND endurance at a linear speed. This motivates us to explore new ways to mitigate retention errors in 3D NAND.

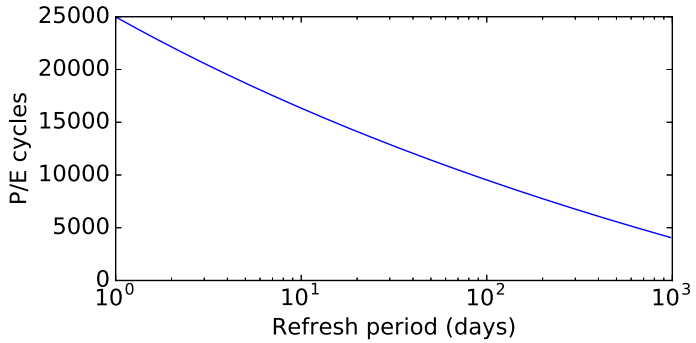


Figure 6.32: Achievable 3D NAND endurance when refreshing periodically at different intervals.

Mechanism: The key idea of ReMAR is to accurately track the retention age of the data and apply the optimal read reference voltage predicted by our model in Section 6.4.2. First, ReMAR constructs the same linear models proposed in Section 6.4.2 *online* to accurately predict the optimal V_a, V_b, and V_c. Similar to the distribution parameter model used in Section 6.4.2, we model the optimal V_b and V_c as: $V = (a \cdot PEC + b) \cdot \log(t) + c \cdot PEC + d$. We model the optimal V_a as: $V_a = a \cdot PEC + b$, since V_a is not affected by retention age. To construct this model *online*, the controller randomly selects a block during each idle period and records the optimal read reference voltage of the block, learned by sweeping the read reference voltages as is done in prior work [27], and its corresponding P/E cycle (PEC) and retention age (t). Over time, these data samples should be able to cover a range of P/E cycles and retention ages.[9] Note that as the P/E cycle count of the SSD increases, the accuracy of the model increases, because more data samples are collected. Once this online model is constructed, it is used in the controller to predict the optimal read reference voltage on each read operation. To do this, the SSD controller stores the P/E cycle and the program time of each block as metadata. During each read operation, the controller computes the retention age for each read by subtracting the program time from the read time. Using the recorded P/E cycle and the computed retention age of the data, ReMAR applies the online model to predict V_a, V_b, and V_c. By accurately predicting and applying the optimal read reference voltages, ReMAR hopes to increase the accuracy of read operations and thereby decreasing the raw bit error rate.

Overhead: The memory and storage overhead is 800KB for a 1TB SSD assuming a 5MB block size and 4B time size. The performance overhead of each read operation is small as we only need a few CPU cycles (on the order of 100ns) in the flash controller to compute V_{opt}, which is negligible compared to flash read latency (on the order of 10μs). The performance overhead of learning the model can be hidden by only learning during idle time.

The controller uses universal time for program and read times, such that the recorded time is valid after reboot. To do this, the controller needs a real-time clock to keep track of the current time. Without a power source on the SSD, the controller needs a special command to synchronize the current time with the host when it boots up. The program time of each block is cached in the memory of the controller, along with other metadata that already exists such as the logical

[9]The SSD controller can also perform explicit characterization if certain data range is missing.

address map and the P/E cycle of each block.

Evaluation: Figure 6.33 compares the RBER achieved by ReMAR (`retention aware`) to the state-of-the-art read reference voltage tuning technique for planar NAND that only adapts V_a to P/E cycle (`retention agnostic`). The RBER results are obtained by simulating reads on the characterization data used in Section 6.2.2. We assume that the average retention time of the data is 24 days. On average, ReMAR reduces RBER by 51.9%. As P/E cycle increases, the benefit of ReMAR becomes larger because the threshold voltage shifts faster over retention age. By accurately tracking retention age, and by using our online model, ReMAR can adapt to this shift and reduce RBER.

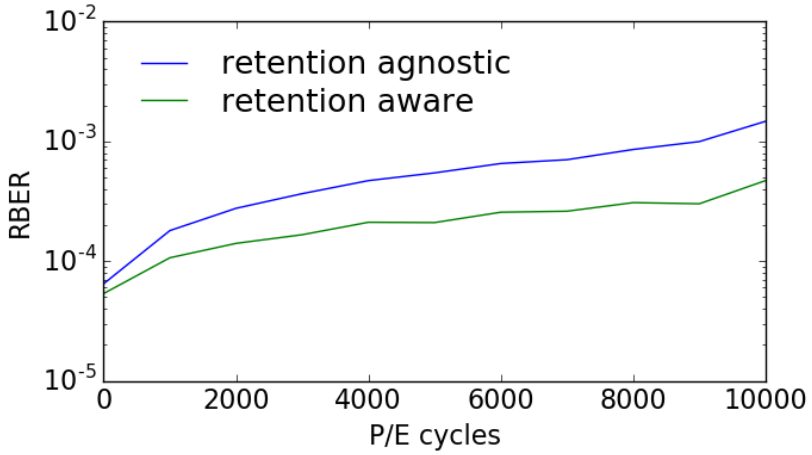


Figure 6.33: RBER reduction using retention-aware optimal read reference voltage.

6.5.4 ReNAC: Mitigating Retention Interference

Motivation: As we observe in Section 6.2.3, due to retention interference, the amount of threshold voltage shift of a victim cell during a certain amount of retention time is affected by the value stored in the neighboring cell. This presents a similar data dependency as that induced by program interference, where the amount of threshold voltage shift of a victim cell during programming operation also depends on the value stored in the neighboring cell [24, 26]. To account for this data dependency, prior work proposes neighbor-cell assisted correction (NAC), which compensates this threshold voltage shift by adjusting the read reference voltage based on the neighboring cell's value [26]. However, this mechanism does not account for retention interference, which is new in 3D NAND flash memory. Thus, we adapt NAC for 3D NAND flash memory to account for the new retention interference.

Mechanism: The key idea of ReNAC is to use the data stored in the neighbor cell to predict and adapt to the amount of retention interference on a victim cell. Using similar techniques from Section 6.4.2, ReNAC first develops an online model of retention interference as a function of the retention age and the neighbor cell's state. The SSD controller obtains the retention age of each block using a similar mechanism to ReMAR, and computes and applies the neighbor

cell dependent read offset at that retention age from the model. We expect that, as retention interference increases in future 3D NAND devices, the benefit of ReNAC will increase. We leave a quantitative evaluation of ReNAC for future work.

6.5.5 Implications on Systems Reliability

The mechanisms we propose in this section can be combined together to achieve significant reduction in average and worst-case RBER. We demonstrate the benefits that these reliability improvements can provide for computer systems, such as flash memory lifetime, sustainable workload write intensity, and error correction.

In Figure 6.34, we compare and contrast the reliability of three example SSDs: (1) *Baseline*, an SSD that uses a fixed, default read reference voltage and deploys a conventional RAID scheme; (2) *State-of-the-art*, an SSD that uses the optimal read reference voltage predicted by existing mechanisms designed for planar NAND [27, 195, 252, 260] and deploys a conventional RAID scheme; (3) *This Work*, an SSD that uses the optimal read reference voltage predicted by LaVAR and ReMAR, and deploys the LI-RAID scheme. In this figure, we plot the worst-case RBER instead of the average RBER, because the worst-case RBER limits flash memory lifetime. Because RBER increases over the P/E cycle count, if the worst-case RAID group has a high enough RBER, NAND flash memory can no longer guarantee reliable operation.

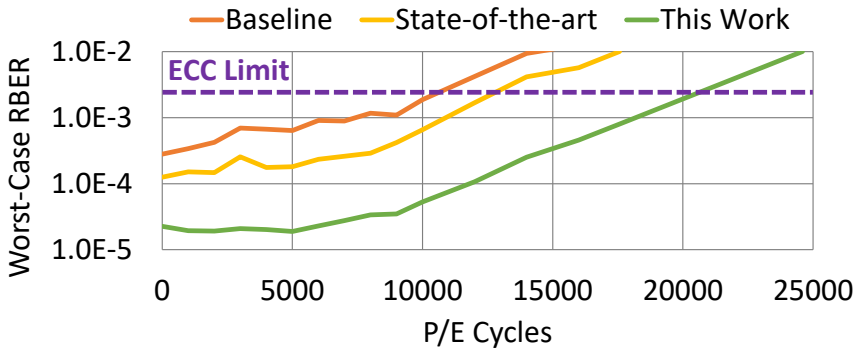


Figure 6.34: RBER reduction using retention-aware optimal read reference voltage.

Assuming that the ECC deployed on the SSD can correct up to $3 \cdot 10^{-3}$ RBER [27, 33] (i.e., the ECC limit, the purple dotted line in Figure 6.34), we can calculate the lifetime of each example SSD. The flash memory lifetime ends when the worst-case RBER goes above the ECC limit. We find that *State-of-the-art* improves flash lifetime by 23.8%, and *This Work* improves flash lifetime by 85.0% over the *Baseline*. When the SSD is deployed in a server, which has a fixed device lifetime, the server has to throttle the write frequency to a certain *drive writes per day* (DWPD) to ensure the SSD can operate reliable during the fixed lifetime. In this case, the flash lifetime improvements translates into an increased threshold of 85.0% higher DWPD for SSDs in a server.

In modern SSDs, the storage overhead for error correction, i.e., *ECC redundancy*, increases in each generation to better tolerate the degraded flash reliability due to aggressive scaling. For example, to tolerate an RBER of up to $3 \cdot 10^{-3}$ for the *Baseline* SSD at the end of its lifetime, a

modern BCH code [107] requires 12.8% storage overhead for the redundant ECC bits [72]. By deploying various error mitigation techniques like the ones we propose in this section, we can significantly reduce this error correction overhead without sacrificing flash reliability. Assuming all three example SSDs achieve the same lifetime, and the same reliability (i.e., uncorrectable error rate) at the end of their lifetime, *State-of-the-art* reduces ECC redundancy to 7.4%, and *This Work* reduces ECC redundancy to 2.7%, which is 78.9% lower than *Baseline*. We leave the evaluation of the performance improvement due to a weaker ECC requirement for future work [48, 177].

6.6 Limitations

The observations and analysis results only apply to 3D NAND based on a charge trap cell design. The error characteristics of future 3D NAND chips may also change when the manufacturing process technology starts shrinking again if adding more stacked layers become less economically viable. In a smaller process technology node, two-step programming errors, read disturb errors, and retention interference errors may become dominant; thus our model needs to be extended to consider all of them. Our techniques focus on mitigating layer-to-layer process variation and retention errors that are unique to 3D NAND flash memories. Our techniques do not reduce other errors caused by read disturb or program interference because prior works already provide techniques to mitigate them [24, 35]. Our retention model and ReMAR do not consider self-recovery and temperature effects, which could affect flash reliability significantly. In Chapter 7, we extend our retention model to consider these effects, and provide mechanisms to track the amount of self-recovery and temperature effects.

6.7 Conclusion

We develop an understanding of three new error characteristics in 3D NAND flash memory through rigorous experimental characterization of real, state-of-the-art 3D NAND flash memory chips: layer-to-layer process variation, early retention loss, and retention interference. We analyze and show that these new error characteristics are fundamentally caused by changes introduced in the 3D NAND flash memory architecture, and predict that these errors will increase in future 3D NAND flash memories. To handle these three new error characteristics in 3D NAND, we develop new analytical models for layer-to-layer process variation and early retention loss in 3D NAND flash memory, and propose four new techniques that utilize our models to improve the reliability of 3D NAND flash memory. Our evaluations show that our newly-proposed techniques successfully mitigate the new error patterns that we discover in 3D NAND flash memory. We hope that the error characterization and analysis performed in this work motivates future studies on the reliability of 3D NAND, and that it inspires new error mitigation mechanisms that cater to 3D NAND flash memory.

Chapter 7

HeatWatch: Self-Recovery and Temperature Aware Retention Error Mitigation

As we have shown in Chapter 6, retention errors dominate 3D NAND flash memory errors. Thus, mitigating retention errors in 3D NAND can yield significant flash reliability improvements. We find promising opportunity to mitigate flash retention by exploiting *self-recovery*, which we have introduced in Section 3.1.6. This opportunity has largely been ignored in recent literature, as the self-recovery effect is not well understood, especially in 3D NAND flash memory.

In this chapter, we exploit the 3D NAND flash memory's device-level self-recovery behavior to improve flash reliability. The goal of this chapter is to (1) perform a detailed experimental characterization of the self-recovery effect using real, state-of-the-art 3D NAND flash memory devices, and (2) exploit our findings by developing HeatWatch, a new mechanism to help self-recovery in 3D NAND flash memory. The key idea of HeatWatch is to *adapt the read reference voltage to the dwell time (i.e., the idle time between consecutive program cycles) and retention time of the workload as well as the SSD's operating temperature*.

First, we extensively characterize how self-recovery and temperature affects 3D NAND flash reliability using real, state-of-the-art MLC 3D NAND flash memory chips (Section 7.1). Second, based on the findings of our characterization, we propose *URT*, a new unified model of self-recovery, temperature, retention loss, and wearout for 3D NAND flash memory (Section 7.2). Third, using URT, we propose and evaluate *HeatWatch*, a mechanism to improve flash reliability and lifetime by optimizing read operations in 3D NAND flash memory for the amount of self-recovery allowed by the workload and the operating temperature of the flash memory (Section 7.3).

7.1 Characterizing the Self-Recovery Effect

To understand the behavior of the self-recovery effect in 3D NAND flash memory, we perform an extensive characterization of the effect using *real*, state-of-the-art 3D NAND flash memory chips. Our goal in this characterization is to answer the following research questions:

- How does the dwell time affect retention and program variation errors?
- What is the correlation between dwell time and the magnitude of the self-recovery effect?
- How does the operating temperature affect the number of retention and program variation errors?
- How do the benefits of self-recovery change based on the number of performed *recovery cycles*?

We make all of our characterization data, including results not shown in this chapter for brevity, available in an extended report [192] and online [281].

We use the observations and analysis from our characterization to drive the design of a new model of 3D NAND flash memory reliability, as described in Section 7.2.

7.1.1 Characterization Methodology

To answer the above research questions, we design new experiments to test how flash reliability changes with different dwell times and temperatures. In these experiments, we use state-of-the-art 30- to 40-layer 3D charge trap NAND flash chips from a major vendor.[1] We use a NAND flash memory characterization platform that fits in the SSD form factor, and contains a special version of the firmware in the SSD controller. We use a server machine to issue remote procedure calls (RPC) [13] to the firmware over a Serial ATA (SATA) [290] connection. These RPCs trigger commands to be sent directly to the flash chips, and can transfer the raw data (i.e., data with raw bit errors) directly from the flash chips to the server *without* applying error correction (ECC). This allows us to observe the effect of dwell time and operating temperature on the raw bit error rate of each flash chip.

We use two metrics to evaluate flash reliability. First, we measure the *raw bit error rate* (RBER), which is the rate at which errors occur in the data *before error correction*. To calculate RBER, we read data from a NAND flash memory chip using the default read reference voltage, and compare the data using a pristine server-side copy of the data that was originally written to the chip. Second, we show the statistical mean of the threshold voltage distribution of each high-voltage state (i.e., P1, P2, P3). As we mention in Section 3.1, retention loss and program variation cause the threshold voltages of cells to shift, which leads to the raw bit errors.[2] To obtain the threshold voltage distribution of a flash page, we perform multiple read operations to sweep the range of all positive read reference voltages, using the *read-retry* command on the NAND flash memory chip [23, 27, 82].[3] Read-retry allows us to fine-tune the read reference voltage used for each read operation. The smallest amount by which we can change the read reference voltage is called a *voltage step*. We normalize each threshold voltage value to the number of voltage steps needed to reach that particular voltage value.[1]

Limitations. In our experiments, we characterize 3D NAND flash memory chips of the same

[1] We do not disclose the exact number of stacked layers in the chips, to protect the anonymity of the flash vendor, and we cannot disclose the *exact* voltage values, as this is proprietary information.

[2] We are unable to show the full threshold voltage distribution for the ER state, because the ER state threshold voltages are negative, and our platform cannot measure negative voltage values. This is similar to prior works [23, 27, 33, 35].

[3] Due to space limitations, we refer the reader to prior works [195, 260] for a detailed methodology on how to obtain the threshold voltage distribution.

model from one major vendor. Our approach ensures that any variation that we observe in our characterization is the result of only manufacturing process variation, and not a result of differences in flash chip architecture or different manufacturing techniques used by different vendors. While we do not take vendor-related variation into account, we believe that our general qualitative findings on the effects of self-recovery and temperature can be generalized to 3D charge trap NAND flash memory manufactured by other vendors. This is because the underlying structure of 3D charge trap cells (see Section 3.4) is similar across different vendors [55, 221, 257]. Thus, while the exact numbers reported in this chapter may differ from vendor to vendor, our qualitative findings, which are a result of charge detrapping from the tunnel oxide (see Section 3.1.6), should be similar across vendors.

We are unable to perform repeated runs of our test procedures on the *same* block, as each run of a test procedure induces further wearout on a block. The amount of wearout affects the error rate of NAND flash memory [23, 32, 33, 195, 219, 260]. To ensure an accurate comparison between multiple test procedure runs, we use eight *target* wordlines in the same stack layer from eight randomly-selected flash blocks that are set to the *same* level of wearout for the same test procedure. By selecting wordlines in the same layer, we eliminate the potential impact of cross-layer process variation. Note that we do not characterize *chip-to-chip* process variation, as an accurate study of such variation requires a large-scale study of a large number (e.g., hundreds) of 3D NAND flash memory chips, which we do not have access to. Hence, we leave such a large-scale study for future work.

7.1.2 Characterizing the Dwell Time Effect

To measure the effect of dwell time on flash reliability (see Section 3.1.6), we characterize the RBER and the threshold voltage distribution. We wear out each of our target blocks by repeatedly writing pseudorandom data until the block reaches 3,000 P/E cycles. For the last 300 P/E cycles, we use a different dwell time for each block, spanning a range of 64 s to 8192 s. Prior work shows that the magnitude of the self-recovery effect is correlated with the dwell time for only the last 10% of P/E cycles performed on a block [220]. We show in Section 7.1.4 that the dwell time used during only the *last 20 P/E cycles* affects self-recovery.

We measure how the dwell time affects retention loss speed and program variation, by performing a *retention test* on each target wordline *immediately* after the block containing the wordline reaches 3,000 P/E cycles. In this test, we program pseudorandom data to the target wordline, and repeatedly measure the threshold voltage distribution using the methodology described in Section 7.1.1 for up to 71 min (i.e., 4260 s) after the data was written. We conduct this experiment at an environmental temperature of 70 °C, which accelerates both self-recovery and retention loss to reduce the experiment duration to a reasonable length [220].[4] We later repeat a small portion of the test under room temperature (20 °C), and verify that all of our observed trends remain the same.

Effect on RBER. First, we study how self-recovery affects the raw bit error rate. Figure 7.1 shows the RBER as retention time (t_s) increases, for different dwell times (t_d) used for the last

[4]Based on Arrhenius' Law [6], the same experiment would take more than 11 years to finish had we performed it at room temperature (20 °C).

300 P/E cycles. We use a color gradient for the curves, where the reddest (topmost) curve has the shortest dwell time, and the blackest (bottommost) curve has the longest dwell time. Note that the x-axis and y-axis are both in log scale.

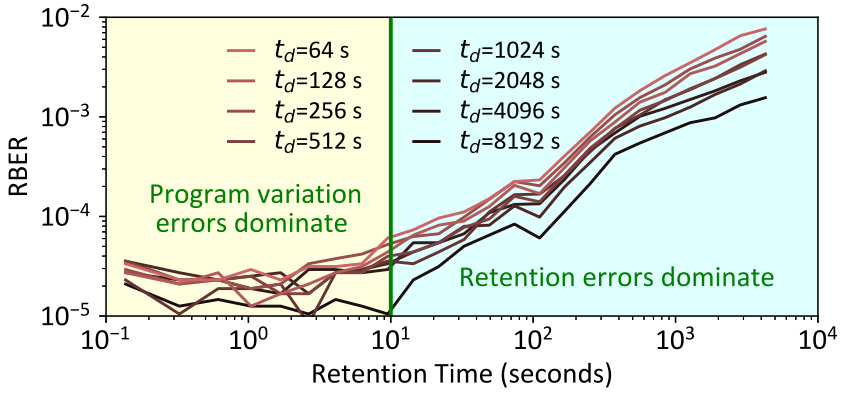


Figure 7.1: Change in RBER over retention time for flash pages that were programmed using different dwell times.

We make two observations from Figure 7.1. First, when the retention time is short (i.e., $t_r < 10$ s), the RBER is *similar* across different dwell times. During this time, the RBER is dominated by program variation errors [21, 195, 219]. Recall that a longer dwell time increases the amount of detrapped charge during self-recovery. However, since the RBER is similar across different curves regardless of the dwell time, this means that self-recovery does *not* mitigate program variation errors. Therefore, unlike previous works [50, 224, 337], we conclude that self-recovery does *not* repair *all* of the errors that occur due to wearout in 3D NAND flash memory. Second, when the retention time is large (i.e., $t_r > 10$ s), there is a strong correlation between a longer dwell time and a lower RBER. During this time, the RBER is dominated by retention errors (hence the growth in RBER as the retention time increases). We conclude that a longer dwell time after an erase operation mitigates retention errors, but not program variation errors, in 3D NAND flash memory.

Effect on Threshold Voltage. Next, we study the threshold voltage distribution of the flash pages under test, to understand how self-recovery affects the threshold voltage shift (and thus the RBER) due to retention loss. Figure 7.2 shows the threshold voltage distribution *before* (black dots, $t_r = 1$ min) and *after* (red dots, $t_r = 71$ min) a large retention time elapses, for a flash page programmed using a 64 s dwell time (top plot), and for a flash page programmed using a 8192 s dwell time (bottom plot). We observe from the figure that when the dwell time is higher, the threshold voltage distribution shift due to retention loss is significantly smaller.

To quantify the threshold voltage shift as a function of the dwell time, we plot the statistical means of the threshold voltage distributions of cells programmed to the P1, P2, and P3 states in Figure 7.3. We use the same color gradient that we used in Figure 7.1 to represent the different dwell times. Note that for these experiments, the smallest retention time that we show on the x-axis ($t_r = 64$ s) is much larger than the smallest retention time shown in Figure 7.1 ($t_r = 0.5$ s), because it takes significantly longer for us to sweep the threshold voltage of the cells in a wordline

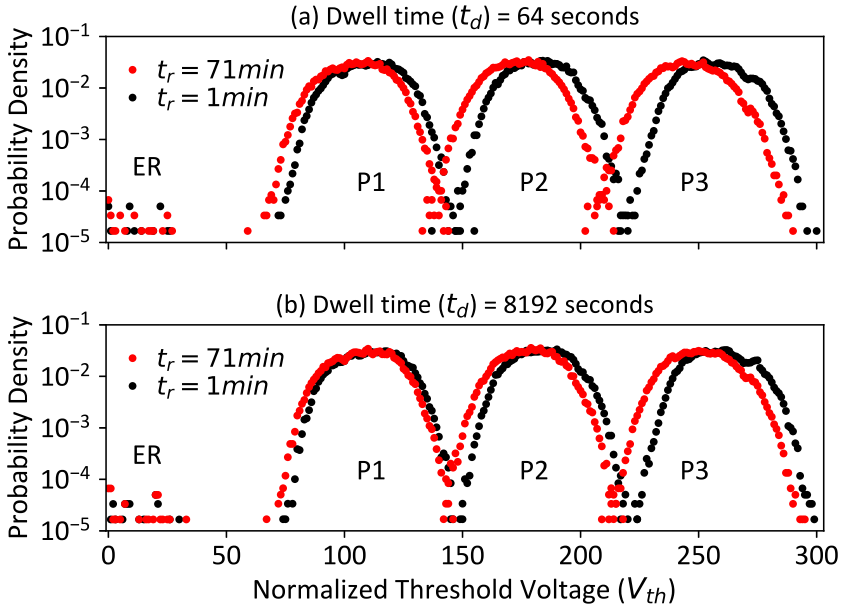


Figure 7.2: Threshold voltage distribution before and after a long retention time, for different dwell times.

(as in Figure 7.3), compared to simply measuring the RBER of the wordline (as in Figure 7.1). We make two observations from the figure. First, for all three states, the mean threshold voltage changes by a smaller amount when the dwell time is higher, corroborating the threshold voltage distribution shifts shown in Figure 7.2. Second, for a fixed dwell time, the change in voltage is linearly related with the log of retention time.[5] We use this relationship to develop a simple model that can quantify how retention loss speed changes with dwell time (see below).

We also calculate how the width of the distribution changes due to retention loss for different dwell times (not plotted). We observe that the change in the distribution width is relatively small, and thus choose to ignore it to simplify the analysis.

Effect on Retention Loss Speed and Program Variation. To quantify how self-recovery changes (1) retention loss speed and (2) program variation, we construct a simple model of how the threshold voltage and RBER change due to these two factors. As we observe in Figure 7.3, the threshold voltage distribution mean is linearly correlated with the logarithm of the retention time (t_r). Thus, we fit our measurements to the following linear model, for a given dwell time:

$$Y(t_r) = \alpha \cdot \ln(t_r) + \beta \tag{7.1}$$

In this model, Y can represent either (1) the mean of the threshold voltage distribution of each high-voltage state (i.e., P1/P2/P3); or (2) the logarithm of the RBER, i.e., $\log(RBER)$;[6] based on

[5] A similar linear relationship between the change in threshold voltage and the log of the retention time is observed for planar NAND flash memory in past works [124, 220].

[6] We model the *logarithm* of the RBER, because when retention loss linearly shifts the threshold voltage distribution, which roughly follows a Gaussian distribution [195], the linear distribution shift results in a logarithmic change in RBER, which is quantified as the overlapping area between two neighboring distributions.

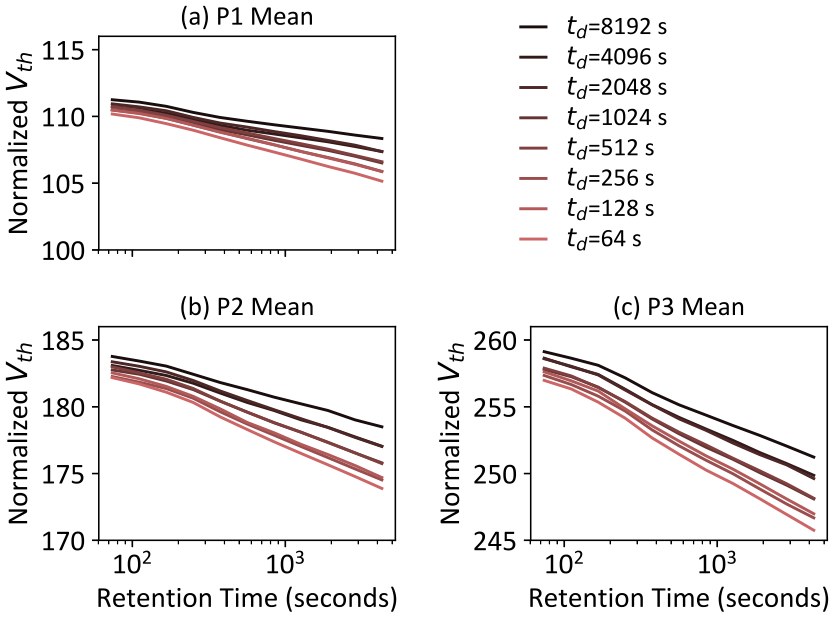


Figure 7.3: Threshold voltage distribution mean vs. retention time for different dwell times.

the values chosen for α and β. α represents the retention loss speed. β represents the *program offset*, which is the initial value of Y immediately after programming.

We use the *absolute value* of the program offset (i.e., $|\beta|$) to quantify the impact of program variation. For the threshold voltage distribution mean of each high-voltage state, Y and β are positive, and a *more positive* program offset results in a *higher* initial mean. As we observe under *Effect on Threshold Voltage* in Section 7.1.3, when the mean voltages of neighboring distributions increase, the overlap between the distributions decreases, which in turn reduces the number of program variation errors. For $\log(RBER)$, Y and β are negative, because the RBER is always less than 1. A *more negative* program offset (i.e., a greater $|\beta|$) corresponds to a *more negative* initial value of $\log(RBER)$ (i.e., *fewer* errors).

For each dwell time, we fit Equation 7.1 to our experimental characterization data in order to calculate the values of α and $|\beta|$ when Y represents (1) the mean voltage of each higher-voltage state, or (2) $\log(RBER)$. Figure 7.4 (left) illustrates the relation between dwell time and retention loss speed (α), normalized to the greatest observed retention loss speed. Figure 7.4 (right) illustrates the relation between dwell time and program offset ($|\beta|$), normalized to the greatest observed program offset. Note that the x-axis (i.e., the dwell time) is in log scale. In both figures, the markers represent our measured data points from real NAND flash memory chips, and the dashed lines show a linear trend line for the data.

We make two key observations from Figure 7.4. First, the self-recovery effect reduces the retention loss speed linearly with the logarithm of dwell time. We observe, however, that the change in retention loss speed is *different for each state*. As Figure 7.4 (left) shows, our data points follow the linear trend line closely (with an R^2 value larger than 90% for each state and for the RBER). Second, as we concluded from Figure 7.1, self-recovery has little effect on program variation within the tested dwell time range. As Figure 7.4 (right) shows, the maximum

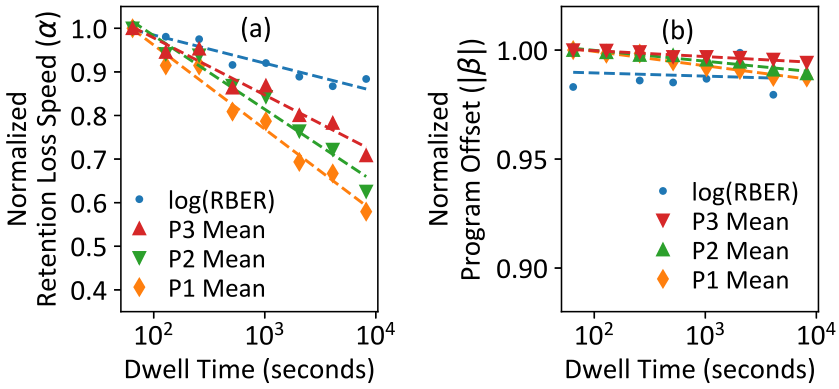


Figure 7.4: Retention loss speed (left) and program offset (right), for different dwell times.

difference in program offset for any of our threshold voltage states is less than 2.1%. Note that our second finding is *new*, and it shows that, unlike previous findings for planar NAND flash memory [50, 224, 337, 338], self-recovery does *not* reduce the number of program variation errors, and hence the amount of wear, in 3D NAND flash memory.

7.1.3 Characterizing the Temperature Effect

Next, we measure the effect of temperature on self-recovery and flash reliability (see Section 3.1.6). Similar to the experiment in Section 7.1.2, we use eight target wordlines in the same stack layer from randomly-selected flash blocks. First, for each block, we wear out the block in 1,000 P/E cycle intervals up to a total of 10,000 P/E cycles, writing pseudorandom data and using a fixed dwell time of 0.5 s. We then put the test chip in a temperature-controlled box, and set a target temperature. After the temperature of the test chip settles to the target temperature, we perform 20 more P/E cycles to each target wordline at the target temperature, using a 0.5 s dwell time. Though these P/E cycles are performed at different temperatures for each test, the dwell time we use is small, and thus we assume that the difference between the equivalent dwell times at room temperature are small across our tests. Then, we perform the retention test described in Section 7.1.2 for all target wordlines up to a retention time of 71 min. We repeat the retention test under a range of target temperatures in each round: 0, 10, 20, 28, 35, 50, 60, and 70 °C. During the retention test, data is both programmed and read under the target temperature.

Effect on RBER. First, we study how the RBER changes with retention time under different temperatures, as shown in Figure 7.5 for a wordline with 10,000 P/E cycles. Each curve represents the RBER under a different programming temperature. We use a color gradient to indicate the temperature: the reddest color represents the highest temperature (70 °C) and the blackest represents the lowest temperature (0 °C).

We make two key observations from the figure. First, when the retention time is small (i.e., $t_r < 2 \cdot 10^2$), the RBER is lower for higher temperatures (i.e., the red curves). Recall that when the retention time is small, the RBER is dominated by program variation errors [21, 195, 219]. Thus, we expect that the RBER decreases with higher temperatures because higher programming tem-

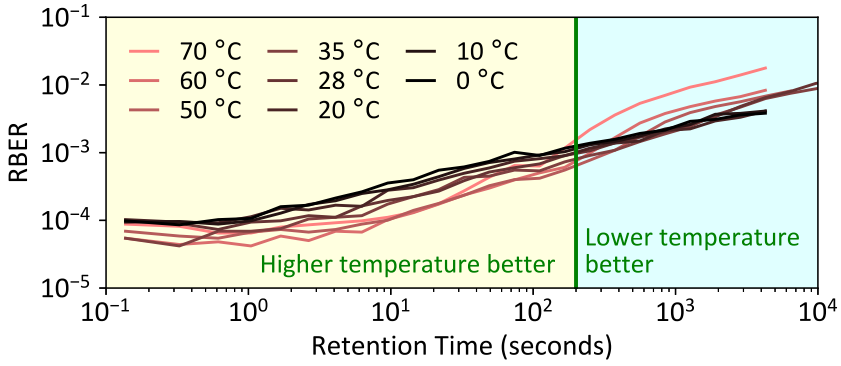


Figure 7.5: RBER over retention time at 10,000 P/E cycles under different programming temperatures.

peratures *decrease* the number of program variation errors (we discuss this in more detail under *Effect on Threshold Voltage* below). Second, when the retention time is larger (i.e., $t_r > 2 \cdot 10^2$), the RBER becomes higher for higher programming temperatures. This is because as the temperature increases, the retention errors increase at a faster rate. Due to this faster growth, the RBER for a higher temperature overtakes the RBER for a lower temperature at a retention time between e2 s to e3 s. This indicates that the threshold voltage shift due to high-temperature retention is faster than that for low-temperature retention, which is in line with Arrhenius' Law [6] (see Section 3.1.6).

Effect on Threshold Voltage. Next, we study how the programming temperature affects the threshold voltage of a flash cell. We begin by studying how the initial threshold voltage (i.e., the threshold voltage immediately after programming) changes with temperature. We measure the threshold voltage distribution under different programming temperatures, and then fit our data to Equation 7.1 to compensate for any retention loss that occurs during the measurements. Figure 7.6 shows the resulting threshold voltage distribution for each state, at 0 °C (the black dotted curves) and at 70 °C (the red curves). Note that the ER state distribution at 70 °C completely falls below the lowest voltage that we can measure, and hence is not shown.

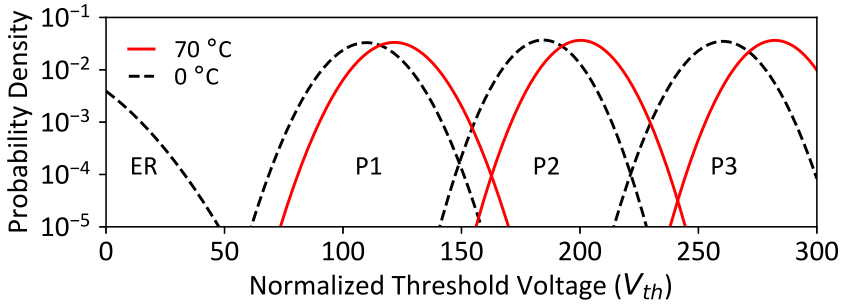


Figure 7.6: Threshold voltage distribution right after programming at different programming temperatures, predicted by our retention loss model (Equation 7.1).

We make two observations from Figure 7.6. First, the higher programming temperature shifts

the P1, P2, and P3 states to higher threshold voltages, and the ER state to lower threshold voltages. The threshold voltage shifts are likely due to the increased electron mobility under high temperature, which improves the speed of the program operation (and likely the erase operation as well [220, 332]). As a result, each programming pulse adds a greater amount of charge. Second, due to the threshold voltage distribution shifts, the amount of overlap between the P1 and P2 distributions, and between the P2 and P3 distributions, decreases at a higher programming temperature. This is because while the distribution means shift to higher voltages at a higher programming temperature, the distribution widths do not change. Because of the smaller amount of overlap between two neighboring distributions, there are fewer program variation errors at higher temperatures, as we have shown in Figure 7.5.

Next, we study how threshold voltage shifts due to retention loss change with the programming temperature. For brevity, we do not plot these results. We observe that when the retention time is large ($t_r > 2 \cdot 10^2$), the retention loss speed increases due to high temperature, which is similar in nature to the effect of programming temperature on RBER.

Effect on Retention Loss Speed and Program Variation. We use our model from Equation 7.1 to calculate the retention loss speed and program offset for each programming temperature, based on our characterization data. Figure 7.7 illustrates how the programming temperature affects retention loss speed (left) and program offset (right). We fit a quadratic trend line for retention loss speed, and a linear trend line for program offset (shown as dashed lines).

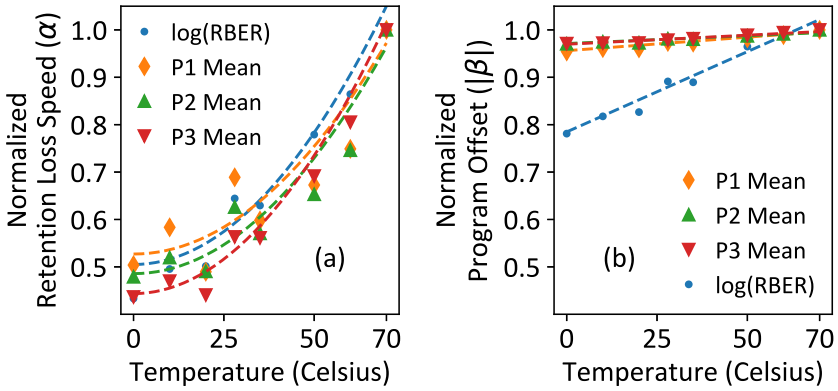


Figure 7.7: Retention loss speed (left) and program offset (right) across different programming temperatures.

We make two observations from Figure 7.7. First, a higher temperature accelerates retention loss at a superlinear rate. We show in Section 7.2 that this trend complies with Arrhenius' Law [6]. Second, we find that the programming temperature affects program variation. This effect has *not* been accounted for in prior work, which usually assumes that program operations are performed at room temperature [124]. In Figure 7.7 (right), we find that the program offset is higher at higher programming temperatures. As already shown in Figure 7.6, the higher initial threshold voltage at higher programming temperatures reduces the amount of overlap between neighboring threshold voltage distributions, which in turn *reduces* the number of program variation errors. We conclude that higher temperature *increases* retention errors but *reduces* program

variation errors.

7.1.4 Characterizing the Recovery Cycle Effect

We measure the effect of *recovery cycles*, i.e., P/E cycles performed with a long dwell time, on self-recovery and flash reliability. We measure how the *number* of recovery cycles affects retention loss speed. We focus on retention loss speed in this experiment because, as we saw in Section 7.1.2, the dwell time does *not* affect program variation. We first wear out each block by repeatedly writing pseudorandom data with a dwell time of 0.5 s, until the block reaches 3,000 P/E cycles. Then, we start self-recovery, performing recovery cycles using a 6-hour dwell time. During the idle time of each recovery cycle, we perform our 71-minute retention test at an operating temperature of 70 °C to measure the current retention loss speed. We keep performing recovery cycles until the change in retention loss speed is less than 1%, which indicates that an additional recovery cycle does *not* significantly increase the effect of self-recovery.

Effect on Retention Loss Speed. Based on our characterization data, we calculate the retention loss speed (α) after each recovery cycle. We use Equation 7.1 to calculate α, as we did in Figures 7.4 and 7.7. Figure 7.8 shows how the retention loss speed changes as a function of the number of recovery cycles performed. We fit power law trend lines to the data, shown as a dashed line in the figure.

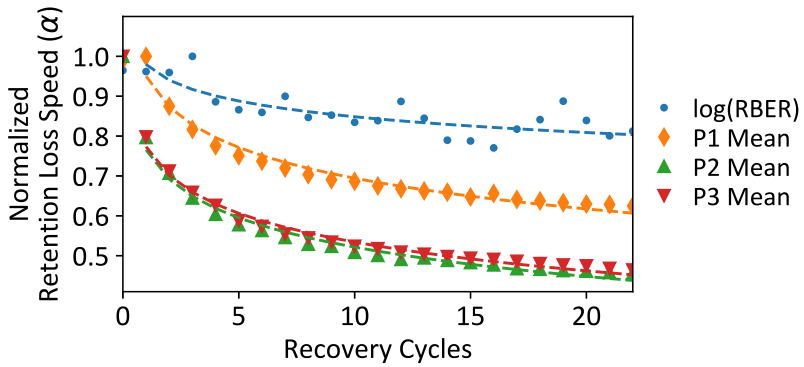


Figure 7.8: Retention loss speed vs. recovery cycles.

We make two key observations from Figure 7.8. First, to our surprise, the reduction in retention loss speed due to self-recovery becomes insignificant very quickly. We find that, for wordlines that have endured 3,000 P/E cycles, most of the benefits of self-recovery occur within 22 recovery cycles for 3D NAND flash memory. This is much smaller than the number of cycles required according to prior work [220], which finds that most of the benefits of self-recovery occur only when the number of recovery cycles is 10% of the total P/E cycle count. In other words, according to prior work, it should have taken at least 300 recovery cycles to achieve most of the benefits of self-recovery. Importantly, this implies that we can reap the benefits of self-recovery with a much lower overhead (i.e., significantly fewer recovery cycles) than previously-proposed mechanisms [67, 125, 220]. Thus, we can improve the overall performance of NAND

flash memory devices that perform self-recovery. Second, the RBER does *not* change significantly *until after* the first three recovery cycles. To reduce the latency of self-recovery, prior works [50, 224, 337, 338] distribute recovery cycles throughout the flash lifetime, and periodically perform only a single recovery cycle. Based on our observation, performing only one recovery cycle may *not* change the RBER significantly, and these mechanisms may *not* significantly improve the flash lifetime as currently designed.

7.1.5 Summary of Key Observations

We conclude that (1) the self-recovery effect reduces retention loss speed linearly with the logarithm of dwell time, and has little effect on program variation; (2) the temperature effect increases retention loss speed at a superlinear rate, and increases program variation; and (3) the reduction in retention loss speed due to self-recovery becomes insignificant after 20 recovery cycles.

7.2 Self-Recovery Effect Modeling

We use our observations from Section 7.1 to construct *Unified Recovery and Temperature* (URT), a comprehensive analytical model of the impact of retention, wearout, self-recovery, and temperature on two output parameters: (1) the threshold voltage of a flash cell, and (2) the raw bit error rate (RBER) of a flash page. URT calculates each output parameter Y as:

$$Y = Y_0 + \Delta Y(t_r \cdot AF, t_d \cdot AF, PEC) \tag{7.2}$$

In the equation, Y_0 is the initial value of the output parameter immediately after a cell is programmed, and ΔY is the change in the output parameter due to retention loss. ΔY is a function of the (1) retention time (t_r) and (2) the dwell time (t_d), both of which are scaled by an *acceleration factor* AF (see below), and (3) the P/E cycle count (PEC).

URT consists of three components. The *program variation component* (PVM; Section 7.2.1) predicts Y_0 based on the amount of program variation that took place. The *effective retention/dwell time component* (RDTM; Section 7.2.2) computes AF, which scales the retention or dwell time at the current temperature of the NAND flash memory to the *effective time* at room temperature that has the same impact on Y. The *self-recovery and retention component* (SRRM; Section 7.2.3), predicts ΔY based on the effective retention/dwell time and the P/E cycle count. We show how URT can be used to improve flash reliability in Section 7.3.

7.2.1 Program Variation Component

First, we build a *program variation model* (PVM) to predict the *initial values* (Y_0 in Equation 7.2) of the threshold voltage and RBER immediately after a cell is programmed. The initial values are determined by (1) the target programming voltage, which is fixed for each state, and (2) the program offset (Section 7.1). Recall that program offset is affected by the programming temperature (Section 7.1.3). Prior work shows that the P/E cycle count significantly affects program offset as well [23, 24, 195, 219].

To account for both variables (i.e., programming temperature and P/E cycle count), we use a multivariate linear regression model to model program variation:

$$Y_0 = A \cdot T_p \cdot PEC + B \cdot T_p + C \cdot PEC + D \tag{7.3}$$

In PVM, Y_0 is a function of the P/E cycle count of the flash cell (PEC) and the programming temperature (T_p), which are input parameters. A, B, C, and D are model constants that change based on which value we model (e.g., initial threshold voltage, initial log of RBER). We fit PVM to our characterization data using the ordinary least squares implemented in Statsmodels [286], and conclude that the model fits well, with an R^2 value of 91.7%. We provide the values of all the fitted model parameters online [192].

7.2.2 Effective Retention/Dwell Time Component

Next, we build an *effective retention/dwell time model* (RDTM) to calculate the *acceleration factor* (AF in Equation 7.2), which scales the retention time or dwell time under *any* temperature (t_{real}) to the effective time under room temperature (t_{room}). Arrhenius' Law [6] (see Section 3.1.6) is commonly used by prior works to scale the retention time and dwell time of flash memory across different temperatures (e.g., [22, 25, 27, 126, 220]). RDTM uses the same general equation as Arrhenius' Law (Equation 3.3):

$$AF = \frac{t_{real}}{t_{room}} = \exp\left(\frac{E_a}{k_B} \cdot \left(\frac{1}{T_{real}} - \frac{1}{T_{room}}\right)\right) \tag{7.4}$$

In RDTM, AF is a function of the room temperature (T_{room}), the current temperature of the NAND flash memory (T_{real}), and the activation energy (E_a). k_B is the Boltzmann constant. To accurately model the amplification factor, we (1) experimentally calculate a new value of E_a that we can use for 3D NAND flash memory; and (2) verify the accuracy of Arrhenius' Law through experimental characterization, which no previous work has done for 3D NAND flash memory.

While E_a is commonly accepted to be 1.1 eV for *planar* NAND flash memory [27, 126], we cannot use the same value of E_a for *3D* NAND flash memory, due to changes in materials and manufacturing process. Fortunately, we have extensive real device characterization data on retention loss at different temperatures (Section 7.1.3), which enables us to determine the correct E_a for 3D NAND flash memory. We define t_1 as the time required for a 3D NAND flash memory device to experience a fixed amount of retention loss, ΔY, at temperature T_1. We define t_2 as the time required for the *same* amount of retention loss to occur at temperature T_2. Using Equation 3.3, the activation energy (E_a) can be calculated as:

$$E_a = \frac{k_B \cdot \ln\left(\frac{t_1}{t_2}\right) \cdot T_1 T_2}{T_2 - T_1} \tag{7.5}$$

We define t_1 as the time required for 3D NAND flash memory to experience a fixed amount of retention loss, ΔY, at temperature T_1. We define t_2 as the time required for the *same* amount of retention loss to occur at temperature T_2.

We choose $T_2 = 343.15\,\text{K}$ (70 °C) as the reference temperature, and $t_2 = 3600\,\text{s}$ as the reference retention time, as our model is more resilient to noise at a larger retention time. Using

our characterization data from Section 7.1.3, we find the equivalent t_1 for different temperature values of T_1, spanning 20 °C to 70 °C, and for different P/E cycle counts, spanning 1,000–10,000 P/E cycles. We use ordinary least squares estimates to fit Equation 7.5 to these data points. From the fit, we determine that for the best fit, $E_a = 1.04\,\text{eV}$ for the 3D NAND flash memory chips that we test. The 95% confidence interval for E_a is 1.01 eV to 1.08 eV. The value of E_a is based on the materials used for the cell, and should be similar for 3D charge trap cells manufactured by other vendors [55, 123, 221].

Next, we verify that Arrhenius' Law holds for 3D NAND flash memory, by calculating the coefficient of determination (R^2) of the fit to the equation for Arrhenius' Law. We find that $R^2 = 0.76$. This means that Arrhenius' Law explains 76% of the variations due to temperature. This is a good fit given that we use a single value for E_a (best fit) across *all* of our data points, because it is known that activation energy changes across different temperatures and P/E cycle counts [10, 208]. We use a single value of E_a regardless of temperature and P/E cycle count to simplify RDTM. We leave more accurate activation energy modeling for future work.

7.2.3 Self-Recovery and Retention Component

We build a *self-recovery and retention model* (SRRM) to accurately predict the *threshold voltage shift and change in RBER* (ΔY in Equation 7.2) due to retention loss. SRRM predicts the effect of (1) effective retention time, (2) effective dwell time, and (3) P/E cycle count, which all directly affect retention loss speed (see Section 7.1.2).

To construct SRRM, we repeat our dwell time experiments from Section 7.1.2 at *room temperature*. We cover a slightly larger dwell time range than our previous experiments, testing from 32 s to 4.6 h. To include the effect of the P/E cycle count, we perform the retention test described in Section 7.1.2 for up to a 24-day retention time under room temperature, using eight randomly-selected flash blocks, and spanning a range between 1,000 and 10,000 P/E cycles at 1,000-P/E-cycle intervals. We observe similar trends in terms of retention time, dwell time, and temperature sensitivity as the findings we observe at a higher temperature in Section 7.1. For brevity, we do not plot these results, but we provide them online [192].

From an analysis of the results of these experiments, we find that the threshold voltage shift in 3D NAND flash memory is much less sensitive to the P/E cycle count than planar NAND flash memory. Thus, we develop a new model that predicts the impact of retention and self-recovery on 3D NAND flash memory, instead of relying on prior models for planar NAND flash memory. Our SRRM model is as follows:

$$\Delta Y(t_{er}, t_{ed}, PEC) = b \cdot (PEC + c) \cdot \ln\left(1 + \frac{t_{er}}{t_0 + a \cdot t_{ed}}\right) \tag{7.6}$$

In SRRM, ΔY is a function of three input parameters: (1) the effective retention time of the data stored in the cell (t_{er}), (2) the effective dwell time (t_{ed}), and (3) the P/E cycle count for a flash cell (PEC). The model has four constants, whose values change depending on which output parameter (ΔY) we are evaluating: b and c control the impact of P/E cycle count on retention loss speed, and t_0 and a control the impact of dwell time on retention loss speed. To determine the values for these constants, we use the non-linear least squares algorithm implemented in SciPy [130, 179] to fit SRRM to the characterization data we collected.

Figure 7.9 illustrates how predictions from SRRM compare with our measured characterization data. Figure 7.9a compares the SRRM predictions and measured values of the threshold voltage shift for cells in state P1, P2, or P3. Figure 7.9b compares the SRRM predictions and measured values of the change in the log of RBER. Each cross ('x') in the figure represents a data point, where the x-coordinate of each data point is the value predicted by SRRM, and the y-coordinate of each data point is the value measured during our characterization. The dashed line shows the perfect fit (i.e., where the predicted and measured values are equal). We observe that for both the threshold voltage shift and the change in RBER, all of the data points are very close to the perfect fit line. Thus, SRRM accurately predicts both output parameters. Overall, the percentage root mean square error (*%RMSE*) for SRRM is 4.9%. As a comparison, if we were to predict these two output parameters using UDM [220], a previously-proposed model for retention loss in planar NAND flash memory, the average *%RMSE* of the predictions would be 17.8%, which is much less accurate than the predictions from our model. We conclude that SRRM is highly accurate for predicting the change in RBER and the threshold voltage shift at room temperature in 3D NAND flash memory.

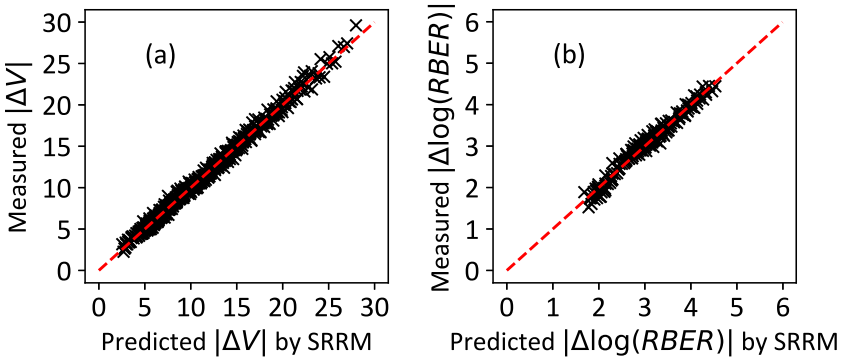


Figure 7.9: SRRM prediction accuracy.

7.3 Improving 3D NAND Reliability

Our *goal* in this section is to improve flash reliability and lifetime by developing a new mechanism that makes use of our findings (Section 7.1) and our new model (Section 7.2). Our new mechanism is called *HeatWatch*.

7.3.1 Observations

We make three key observations from the following three experiments that lead to the design of HeatWatch. First, we observe that *SSD write intensity and the SSD environmental temperature significantly impact flash lifetime*. The write intensity of an SSD is measured as the number of full drive writes per day. Given a fixed SSD size, the write intensity is inversely proportional to the average dwell time, thus affecting flash lifetime (Section 7.1.2). This is because modern SSDs use *wear-leveling* techniques to keep all flash blocks in the SSD at a similar P/E cycle

count [32, 33, 43, 83]. The environmental temperature affects the flash lifetime (Section 7.1.3), because it changes the temperature of the underlying NAND flash memory.

Figure 7.10 shows the flash lifetime under different write intensities and environmental temperatures, assuming that the vendor guarantees a three-month retention time for the data, which is typical for enterprise SSDs [22, 27, 32, 33, 250]. The SSD write intensity is shown on the x-axis in log scale. We plot the results by using separate curves for each temperature that we evaluate, which ranges from 0 °C to 70 °C.

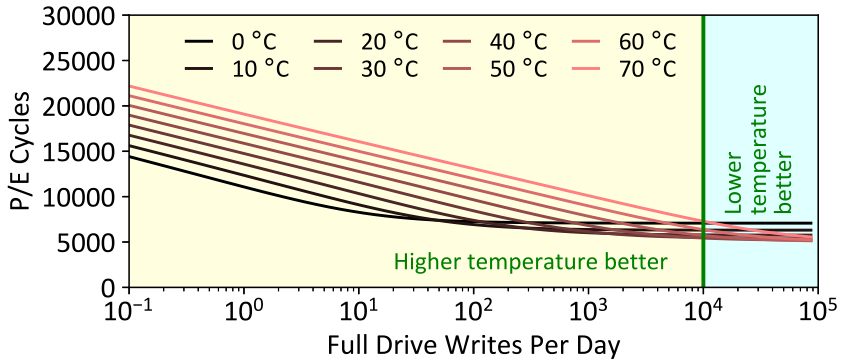


Figure 7.10: Change in flash lifetime due to write intensity and environmental temperature ($t_r =$ 3 months).

From the figure, we see that the flash lifetime initially decreases as SSD write intensity increases, but stops decreasing at around 5,000 P/E cycles. When the write intensity is low ($< 10^4$ drive writes/day, which covers the range of write intensities of most contemporary workloads [22]), a higher temperature is desirable, as it improves both the effective dwell time and program variation and thus leads to a longer lifetime. In contrast, when the write intensity is high, a lower temperature is better due to an improved effective retention time. Note that these curves drastically shift (not shown) (1) with different assumptions about retention time, or (2) when the temperature is not constant. Thus, we find that no single temperature or temperature range is ideal.

Second, we observe that *the choice of the read reference voltage (V_{ref}) significantly affects RBER and flash lifetime*. The voltage at which the lowest RBER can be achieved is called the *optimal read reference voltage* (V_{opt}). V_{opt} changes based on conditions such as retention time and P/E cycle count. Adapting the optimal read reference voltage to these changing conditions significantly increases flash lifetime [23, 24, 27, 32, 33, 195, 260]. Based on our experiments under room temperature, Figure 7.11 shows how the RBER increases as the applied read reference voltage becomes further away from the optimal read reference voltage (which we refer to as the $|V_{ref} - V_{opt}|$ *distance*). We find that the RBER increases exponentially as the $|V_{ref} - V_{opt}|$ distance increases. We conclude, as prior works have [24, 27, 32, 33, 195], that it is important to accurately predict the optimal read reference voltage, as even a small $|V_{ref} - V_{opt}|$ distance can greatly increase the error rate (and thus hurt the flash lifetime).

Third, we observe that *the optimal read reference voltage in 3D NAND flash memory changes over time, and can be accurately predicted as the value that falls in the middle of two neighboring*

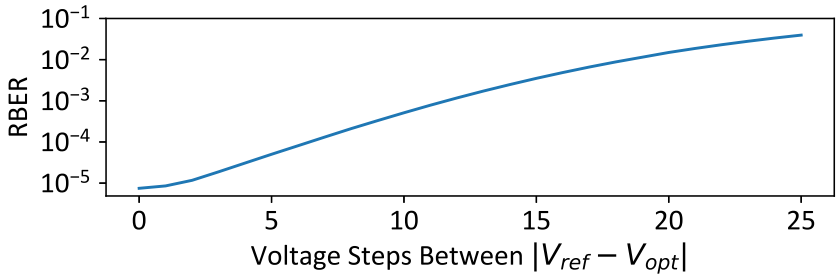


Figure 7.11: RBER vs. $|V_{ref} - V_{opt}|$ distance.

threshold voltage distributions. Figure 7.12 shows the *measured* V_{opt} from our characterization (blue dots in the figure), and the value of V_{opt} calculated by using our URT model to predict the threshold voltage distributions of each state (orange curve), for read reference voltages V_b and V_c (see Section 3.4). The x-axis shows the retention time of the data in log scale. We see that after 4000 s of retention time, the measured optimal read reference voltages for V_b and V_c change by 8 and 11 voltage steps, respectively.[7] Comparing the blue dots with the orange curves, we find that our URT-based V_{opt} prediction is *always* within 1 voltage step of the empirical optimal read reference voltage. We conclude that URT accurately predicts the optimal read reference voltage.

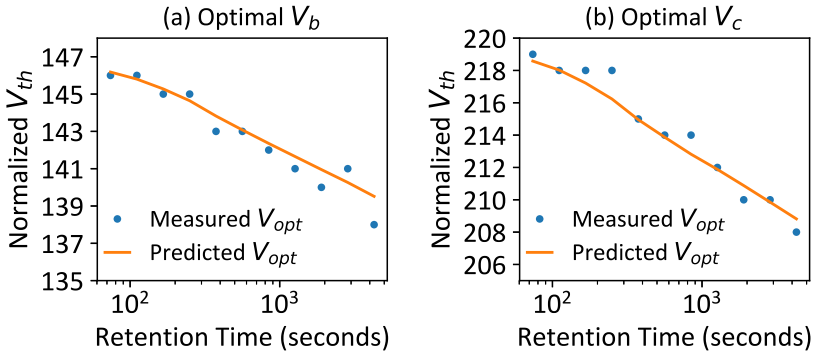


Figure 7.12: Measured and URT-predicted V_{opt}.

7.3.2 HeatWatch Mechanism

Motivated by our observations in Section 7.3.1, we propose *HeatWatch*, a mechanism that improves flash reliability and lifetime by predicting and applying the optimal read reference voltage using our URT model from Section 7.2. HeatWatch consists of (1) three *tracking components*, which monitor and efficiently record the SSD temperature, dwell time, retention time, and P/E cycle count; and (2) two *prediction components*, which compute the URT model using this tracked information to accurately predict the optimal read reference voltage for each read operation.

[7]Our characterization shows that V_a does *not* change significantly based on retention time, so we do not plot it. We find that URT accurately predicts the lack of change in V_a.

Tracking Component 1: SSD Temperature. Modern SSDs already contain multiple temperature sensors to prevent overheating [178, 213]. HeatWatch *efficiently* logs the readings from these sensors, which the RDTM component of URT (see Section 7.2.2) uses to scale the dwell time and retention time. HeatWatch records the temperature every second, and *precomputes* the acceleration factors (AF_i) for every *logarithmic time interval i*. HeatWatch uses logarithmic intervals because the effects of dwell time and retention time are logarithmic with respect to their duration (see Section 7.1.2). HeatWatch uses RDTM to calculate AF_0. For all other intervals, AF_i is calculated as a piecewise integral, by summing up the *two* most recent values of AF_{i-1}, since interval i covers *double* the amount of time as interval $i-1$. Therefore, for each interval, HeatWatch stores the current *and* previous values of AF_i, in an *acceleration factor log*.

Tracking Component 2: Dwell Time. The self-recovery effect is dependent on the average dwell time across *multiple* P/E cycles (Section 7.3.1). The average dwell time is determined by the workload write intensity. Thus, we use the SSD controller to (1) monitor the workload write intensity online, and (2) calculate the average dwell time for each flash block. HeatWatch does *not* need to track the variation in dwell time across different flash pages within the *same* block, as we find from our experimental characterization that the effect of page-to-page dwell time variation is negligible (Section 7.3.1).

HeatWatch approximates the effective dwell time by taking the average unscaled dwell time across the last 20 P/E cycles, and scaling it using RDTM. The SSD controller keeps a counter that tracks the amount of data written to the SSD, and logs the timestamps of the last 20 full drive writes. When a flash block is programmed during drive write n, the SSD controller calculates the average unscaled dwell time as the difference between the current time and the timestamp of drive write $n-20$. Then, the SSD controller computes and stores the effective room temperature dwell time by scaling it using the aforementioned acceleration factor log.

Tracking Component 3: P/E Cycles and Retention Time. The SSD controller already records the P/E cycle count of each block to use in wear-leveling algorithms [32, 33, 43, 83]. To track the retention time of each flash block, HeatWatch simply logs the timestamp when the block is selected for programming. Then, HeatWatch calculates the effective retention time for each read operation as the difference between the program time and read time, scaled by RDTM.

Prediction Component 1: Optimizing the Read Reference Voltage. The optimal read reference voltage between two states can be predicted accurately by averaging the means of the threshold voltage distributions for each state (Section 7.3.1). As HeatWatch knows the P/E cycle count, programming temperature, effective dwell time, and effective retention time, it can use the URT model from Section 7.2 to predict the means of the threshold voltage distributions for each state, and thus the optimal read reference voltage. For each read operation, the SSD controller simply gathers all the metadata for the flash block that is to be read, and then predicts and applies the optimal read reference voltage. The information gathering and prediction happen *after* the FTL translates the logical address of the read to a physical address (see [32, 33] for detail), since the information for the flash block is indexed using the physical address.

Prediction Component 2: Fine-Tuning URT Model Parameters Online. To accommodate for chip-to-chip variation, URT learns its model parameters *online*. We initialize the URT

model parameters using a set of parameters that have been learned offline, which the vendor can provide at manufacture time. Over time, URT fine-tunes its model parameters by (1) sampling a number of flash wordlines in the chip (10 in our evaluations), which are selected at random from blocks that span a range of different P/E cycles, effective dwell/retention time, and programming temperatures; (2) learning the optimal read reference voltages for the sampled flash wordlines online, using a technique similar to Retention Optimized Reading (ROR) [27]; and (3) using the sampled data to fit the fine-tuned URT model parameters for each chip, which can be done relatively easily in the firmware with little overhead [195]. The overall latency for online training is dominated by the latency to predict the optimal read reference voltage for each wordline. In the worst case, ROR performs 300 read operations per wordline, using a different voltage step for each read. For the 10 wordlines sampled by URT model tuning, assuming a read latency of 100 µs, the total worst-case latency of URT model tuning is 0.3 s. Note that this tuning procedure needs to be performed only infrequently (e.g., every 1,000 P/E cycles), and can be performed in the background (i.e., when the chip is idle), thus incurring negligible performance overhead.

Storage Overhead. HeatWatch needs to store three sets of information. (1) HeatWatch stores the acceleration factor for only logarithmic time intervals from 0.5 s to 1 year (26 intervals in total). HeatWatch stores the current and previous value of each acceleration factor, as described in *Tracking SSD Temperature* above. Assuming that each acceleration factor is stored as a 4 B floating-point number, the total log requires 208 B of storage. (2) HeatWatch stores the programming temperature, dwell time, and program time for each flash block. Assuming that each piece of information uses 4 B, for a 1 TB SSD with an 8 MB flash block size, HeatWatch uses 1.5 MB of storage in total to store this information. (3) HeatWatch needs a 32-bit counter, and must store the timestamps for the last 20 full disk writes (Section 7.1.4), which requires 84 B of storage. In total, the three sets of information require less than 1.6 MB of storage. To minimize the performance overhead of accessing this data, HeatWatch buffers the data in the on-board DRAM in the SSD controller [32, 33]. The storage overhead is very small, as a 1 TB SSD *typically* contains 1 GB of DRAM for caching [32, 33].

Latency Overhead. HeatWatch performs two operations that contribute to its latency. (1) Every second, HeatWatch updates the acceleration factor log with the latest temperature reading. This update can be done in the background, and, thus, its performance overhead is negligible. (2) HeatWatch computes the URT model during each read operation, which involves performing *only* 16 arithmetic computations in the SSD controller (Section 7.2). The model computation latency is negligible compared to the flash read latency ($>$25 µs [89]).

7.3.3 Evaluation

To evaluate HeatWatch, we examine the raw bit error rate (RBER) and lifetime of four different configurations:

- *Fixed* V_{ref}, which always uses the default read reference voltage to read the data.
- *Retention-Only*, which predicts the optimal read reference voltage based on a model that considers only P/E cycle count and retention time [23, 24, 27, 195, 252, 260]. Note that this model always assumes a *fixed* retention loss speed, regardless of the dwell time and temperature.

- *HeatWatch*, our proposed mechanism to accurately predict the optimal read reference voltage by tracking dwell time and temperature and using our URT model.
- *Oracle*, which always *ideally* uses the measured optimal read reference voltage, and does *not* incur any performance overhead. Note that this is *not* implementable.

We evaluate these four configurations using 28 commonly-used real storage traces [239], which have varying write intensities. Each trace represents seven days of workload data, and contains timestamps we can use to calculate the dwell time and retention time of each access. We simulate temperature variation over the course of a day as the superposition of a sinusoidal function and some Gaussian noise. The sinusoidal model has a mean of 35 °C, an amplitude of 15 °C, and a 1-day period, representing how the temperature changes during a daily cycle. The Gaussian noise model that we use has a standard deviation of 3 °C.

Figure 7.13 shows how the RBER increases with P/E cycle count for our four evaluated configurations, using a workload that appends *all* 28 disk traces together to mimic an SSD that runs multiple workloads back-to-back. The dotted line shows an error correction capability (see Section 3.1) of $2 \cdot 10^{-3}$ errors per bit [32, 33]. We determine the lifetime for each configuration using the point at which the RBER intersects the error correction capability. We use *Fixed* V_{ref}, which has the highest RBER, as our baseline. From the figure, we see that *Retention-Only* reduces the RBER by 83.1%, on average across all P/E cycles, compared to the baseline, *HeatWatch* reduces the RBER by 93.5% compared to the baseline. This is very close to the average RBER improvement under *Oracle* (93.9%). *HeatWatch* significantly improves lifetime due to its RBER improvement. The lifetime with *HeatWatch* is 21,065 P/E cycles, which is 3.19× and 1.29× the lifetime with *Fixed* V_{ref} and *Retention-Only*, respectively. *HeatWatch* comes within only 200 P/E cycles of the *Oracle* lifetime.

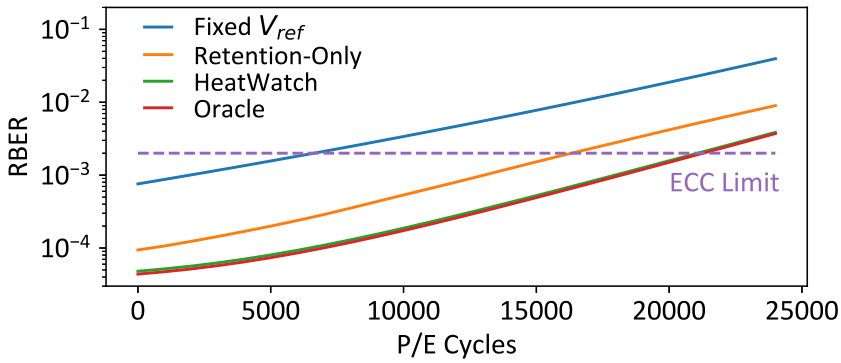


Figure 7.13: RBER vs. P/E cycle count.

We repeat the same experiment for each workload individually, and determine the lifetime for each workload under the four configurations, as shown in Figure 7.14. On average across all of our workloads, The lifetime under *Retention-Only* is 3.11× the lifetime of the *Fixed* V_{ref} baseline. *HeatWatch* improves the lifetime further, with a lifetime that is 3.85× the baseline lifetime. Again, this is very close to the lifetime improvement of *Oracle* (3.89×).

We conclude that by incorporating dwell time and temperature information to predict the optimal read reference voltage, HeatWatch improves the lifetime of 3D NAND flash memory de-

Figure 7.14: P/E cycle lifetime for each workload.

vices over a state-of-the-art mechanism [252], and approaches the lifetime of an ideal mechanism that has perfect knowledge of the optimal read reference voltage.

7.4 Related Work

To our knowledge, this dissertation is the first to (1) experimentally characterize and accurately model the self-recovery and temperature effects in 3D NAND flash memory; and (2) devise a mechanism that improves 3D NAND flash memory lifetime by comprehensively taking into account retention time, wearout, self-recovery, and temperature. We discuss the closely-related works.

Data Retention in NAND Flash Memory. Many prior works focus on flash memory retention loss and retention errors, and show that retention loss is the most dominant source of errors in modern NAND flash memory [22, 27, 32, 33, 57, 219, 250, 252]. These works do *not* consider the effects of self-recovery and temperature on retention loss. Our work investigates these effects through an extensive characterization of state-of-the-art 3D NAND flash memory chips.

3D NAND Flash Memory Characterization. Recent works study the error characteristics of 3D NAND flash memory, and identify differences between 3D and planar NAND flash memory due to memory cell design and architectural changes [32, 33, 55, 221, 257]. None of these works provide a detailed characterization of the impact of self-recovery, retention, P/E cycle count, and temperature on real 3D NAND flash memory.

Retention Loss Models. Our URT model is inspired by and improves upon the Unified Detrapping Metric (UDM) model [220]. There are three reasons why prior models developed for planar NAND flash memory, such as UDM, are insufficient for 3D NAND flash memory. First, 3D charge trap cells are more resilient to P/E cycling than the floating-gate cells used by planar NAND flash memory [55, 221, 257]. Thus, the PEC component in our model (Equation 7.6) is different from the equivalent component in UDM. Second, 3D NAND flash memory has a different activation energy than planar NAND flash memory, as we experimentally show in Section 7.2.2. Third, 3D NAND flash memory reliability is affected by the programming temperature, as we show in Section 7.2.1. Because UDM does *not* accurately capture these changes, its error rate for 3D NAND flash memory is 3.6× greater than the error rate for the SRRM

component of our new URT model, as we show in Section 7.2.3.

Improving Flash Reliability. Many prior works propose mechanisms to improve flash lifetime and reduce raw bit errors (see [32, 33] for a detailed survey). For example, flash refresh techniques limit the number of retention errors to achieve higher P/E cycle lifetime [22, 25, 194, 250]. Prior work also adjusts the read reference voltage according to P/E cycle count and retention time to reduce the RBER [23, 24, 27, 195, 252, 260]. Different from prior work, we develop a new mechanism that tracks workload write intensity and SSD temperature online and adjusts read reference voltage accordingly to improve flash lifetime. We compare our mechanism to prior mechanisms that are agnostic to these factors and show that it can significantly reduce RBER and improve flash lifetime.

7.5 Limitations

Neither our URT model nor the HeatWatch mechanism consider other error sources in 3D NAND flash memory such as retention interference, process variation, and read disturb, but they can be augmented in the future to consider all possible error types to further improve 3D NAND reliability. The activation energy may change under different temperatures and P/E cycles. To simplify our URT model, we use a single static value for activation energy, which is good enough for our use case. When the temperature variation is larger, we may need a dynamic model to estimate the activation energy to more accurately predict the temperature effect. We do not consider temperature variation across layers, which can happen if the NAND flash chip heats up from inside during program/erase operations. We expect future work to improve our model and technique to consider temperature variation. It is possible that certain FTL algorithms, such as WARM, may write to certain flash blocks more frequently, which makes the average dwell time of these blocks significantly shorter than the average. This limits the accuracy of the HeatWatch dwell time tracking component. For these FTL algorithms, we may need to track dwell time differently for these frequently-written blocks.

7.6 Conclusion

We perform the first detailed *experimental* characterization of the impact of self-recovery and temperature on the reliability of 3D NAND flash memory. We find that due to significant changes in the memory design and manufacturing process, prior findings and models for planar NAND flash memory are *not* accurate for 3D NAND flash memory. We use key findings from our characterization to develop URT, a unified and accurate cell threshold voltage and raw bit error rate model that takes into account the combined effects of self-recovery, temperature, retention loss, and wearout. We develop a new mechanism, HeatWatch, that uses URT to dynamically adapt the read reference voltage to the data retention time, dwell time, SSD temperature, and wearout. We show that HeatWatch greatly reduces the raw bit error rate and improves flash lifetime. We conclude that the effects of self-recovery and temperature in 3D NAND flash memory can be accurately modeled and successfully used to improve flash reliability. We hope that our data,

model, and new mechanism inspire others to develop other new mechanisms that take advantage of the self-recovery and temperature effects in 3D NAND flash memory.

Chapter 8

System-Level Implications and Lessons Learned

This dissertation proposes several mechanisms that improve flash reliability by mitigating raw bit errors that occur on flash devices. In this chapter, we discuss the system-level implications of these mechanisms and a summary of lessons learned.

8.1 System-Level Implications

Throughout this dissertation, we have focused on evaluating the traditional metrics of flash reliability, P/E cycle lifetime. This P/E cycle metric assumes that the same workload runs on a each individual SSD (i.e., the workload cannot be redistributed across multiple SSDs). This metric also assumes the NAND flash memory always deploys a fixed amount of ECC that has a fixed RBER limit (as illustrated in Figure 6.34 and 7.13 as the *ECC limit*). However, these assumptions may change in a real system. For example, in a distributed storage system, certain SSDs may contain more write-hot data than the others due to uneven workload distribution across server machines [110], and the flash controller may use a simpler ECC with lower storage overhead. In this section, we discuss the system-level implications or the reliability impact of our mechanisms under different assumptions.

8.1.1 Impact on Tolerable Write Frequency

As we have mentioned in Section 5.5.3 and 6.5.5, server machines in enterprises and data centers, along with the flash memory devices used in those machines, are replaced after a predefined period, typically several years. Since a flash memory device can tolerate a limited total number of writes before having to be replaced, it can only tolerate a certain number of writes *per day* on average (i.e., *tolerable write frequency*). To ensure that each flash memory device does not exceed its P/E cycle lifetime prematurely before being replaced, the system software throttles the write frequency of each flash memory device to only a few Drive Writes Per Day (DWPD) [176]. Instead of increasing the flash device lifetime, we can use our mechanisms to improve the tolerable write frequency. When the P/E cycle lifetime improves after deploying the mechanisms

proposed in this dissertation, the tolerable write frequency by each flash memory device increases proportionally, assuming the ECC limit remains the same. Thus, the increase in tolerable write frequency is the same as the increase in lifetime: WARM improves tolerable write frequency (or DRPD) by 3.24×; Two applications of our online flash channel model improve DWPD by 48.9% and 69.9%; Combining LaVAR, LI-RAID and ReMAR improves 3D NAND DWPD by 85.0% (Section 6.5.5); HeatWatch improves DWPD by 3.85×.

8.1.2 Impact on ECC Cost

As we have shown in Figure 1.1b and in Section 6.5.5, sustaining a high ECC limit incurs a high storage overhead of the ECC bits (i.e., *ECC redundancy*). Using BCH code [14] as an example, assuming that the codeword length is fixed (i.e., 8 kB), the ECC limit is linearly correlated with ECC redundancy [72]. Instead of increasing the lifetime, we can use our mechanisms to reduce ECC redundancy, since we need to tolerate fewer raw bit errors by the end of the flash lifetime.

To calculate the reduction in the required ECC cost, we use the same method and assumptions as we use in Section 6.5.5. First, we obtain the P/E cycle lifetime of the baseline SSD using state-of-the-art error correction mechanisms that can tolerate up to $3 \cdot 10^{-3}$ raw bit error rate. This requires to a 12.8% ECC redundancy using a BCH code [72]. Second, we obtain the worst raw bit error rate of our mechanism at the end of the same P/E cycle lifetime of the baseline SSD. Third, we calculate the ECC redundancy required by our mechanism to achieve the same data reliability in terms of the error correction failure rate (P_{ECFR} in Equation 2.1). The required ECC redundancy is obtained by varying the number of bits within a codeword correctable by the ECC (t in Equation 2.1) until P_{ECFR} reaches 10^{-15} required by the JEDEC standard [121]. Our online flash channel model reduces ECC redundancy by 37.5%; Combining LaVAR, LI-RAID and ReMAR reduce ECC redundancy by 78.9% (Section 6.5.5); HeatWatch reduces ECC redundancy by 78.2%. ECC redundancy savings when using LDPC code should be similar, but are much more difficult to evaluate because LDPC codes do not have a fixed ECC limit [32, 33].

8.1.3 Impact on Performance and Flash Management Policies

We have already shown that the performance overhead of each proposed mechanism is small. The amortized cost for training and predicting using an online model is negligible (i.e., <1%) compared to flash read latency. In the meantime, our mechanisms reduce raw bit error rate at any P/E cycles, which leads to reductions in ECC decoding latency and in read-retry counts, and thus reduces the overall flash read latency, comparing to a baseline SSD with the same amount of P/E cycles. The exact reduction in flash read latency depends on several design choices of the ECC code: (1) a shorter codeword length or fewer ECC bits simplify decoder logic thus reducing the ECC decoding latency. However, this reduces the error correction strength of the ECC, leading to more read-retries as the weaker ECC is more likely to fail, (2) a stronger ECC code may increase ECC decoding latency, but reduces read-retry counts. A similar tradeoff exists when designing a soft-decoding LDPC code to balance the ECC latency and soft-decoding sensing levels. WARM affects existing flash management policies by dividing flash blocks into write-hot and write-cold block pools, which increases the average response time by 1.3% across all workloads; All of our online model-based techniques, including online flash channel modeling, LaVAR, ReMAR

and HeatWatch do not change the data layout or mapping, thus does not increase FTL or GC overhead; LI-RAID only changes the data layout within a flash block, and only reduces the storage capacity by less than 0.4%, thus has negligible overhead to the FTL and GC.

8.2 Lessons Learned

This dissertation provides several new analyses and techniques to improve flash reliability efficiently. When doing these analyses and developing these new techniques, we learned two important lessons that applies for future research in this direction. In this section, we summarize these two lessons.

8.2.1 Combining Large-Scale and Small-Scale Characterization Studies

This dissertation focuses on small-scale studies that perform extensive characterization of only a few NAND flash memory devices. While our observations should apply to any device that uses similar manufacturing technologies (Section 5.2, 6.1, and 7.1.5), and our online modeling techniques can adapt to any variation across chips (Section 5.3 and 7.3), we have not been able to verify this behavior across a large number of devices. Ideally, we would like to combine an in-depth small-scale study with a lightweight large-scale study that does not require characterizing the full threshold voltage distribution, which complement each other to provide a stronger result. Our small-scale study provides a deeper understanding of the characteristics of different error types. A large-scale study can show the effectiveness of our proposed mechanisms in real deployment. We are unable to do this now due to limitations in the number of chips available.

8.2.2 Improve Systems Reliability Rather Than Device Reliability Alone

This dissertation focuses on reducing raw bit errors within the flash chips. These errors are usually invisible to application and system designers because a vast majority of them are already contained within the SSD using strong but expensive ECCs. While containing all raw bit errors within the SSD provides a strong device reliability and reduces the complexity of system design, it leads to a suboptimal design that uses less cost-efficient techniques to improve reliability. For example, the ECC currently designed for the least-reliable flash chip may consume more storage overhead than needed on an average flash chip; and a very low uncorrectable error rate may not be necessary in a distributed storage system because the data is already replicated in other servers to tolerate more catastrophic server failures. An ideal design should combine device-level techniques that mitigates raw bit errors cost-effectively and provides reasonable device reliability, and system-level techniques that utilize many devices to tolerate device failures. This design could lead to significant cost savings while achieving higher overall system reliability.

Chapter 9

Conclusions

In this dissertation, we present a multitude of low-cost architectural techniques to improve NAND flash memory reliability. Following our thesis statement, all of our proposed techniques (1) take advantage of device-level error characteristics or workload characteristics, (2) can be implemented with low overhead in the flash controller as a part of the firmware, and (3) improve NAND flash reliability at low cost.

First, we propose a new technique that exploits workload *write-hotness* and device *data retention* characteristics to improve flash lifetime. We observe that pages with different degrees of *write-hotness* have widely-ranging retention time requirements. WARM aims to eliminate redundant refreshes for write-hot pages with minimal storage and performance overhead. The first key idea of WARM is to effectively partition pages stored in flash into two groups based on the write frequency of the pages. The second key idea of WARM is to apply the most suitable flash management policies to the two different groups of flash pages/blocks. Our evaluations show that WARM eliminates a significant amount of unnecessary refreshes and improves flash lifetime significantly, and that WARM can also be combined with refresh techniques to provide larger benefits in flash lifetime.

Second, we propose a new framework for *online flash channel modeling*, which learns and exploits an online threshold voltage distribution model in the flash controller to improve flash reliability. We observe from our characterization of real, state-of-the-art planar NAND flash memory chips that (1) the threshold voltage distribution can be approximated using a modified version of the Student's t-distribution, and that (2) the amount by which the distribution shifts as the P/E cycle count increases is governed by the power law. Using our characterization results, we build an *accurate* and *easy-to-compute* model of the threshold voltage distribution of modern MLC NAND flash memory. We demonstrate various applications of our model in a flash controller. Our evaluations show that these applications improve flash lifetime significantly.

Third, this dissertation provides the *first comprehensive* experimental characterization and modeling of 3D NAND device characteristics using real, state-of-the-art MLC 3D NAND flash memory chips, and proposes four mechanisms to exploit these device characteristics in the flash controller for improving flash reliability. We find several differences in the device characteristics between 3D and planar NAND devices. Three of the key differences we find are inherent to the internal architecture of 3D NAND flash memory: layer-to-layer process variation, early retention loss, and retention interference. We develop models and techniques to exploit these 3D NAND

flash device characteristics. Our evaluations show that, when combined together, our techniques improve flash lifetime significantly over state-of-the-art error mitigation techniques developed for planar NAND flash memory.

Fourth, this dissertation is the first to experimentally characterize and model the self-recovery and temperature effect in 3D NAND flash memory using real 3D NAND devices, and to propose a new technique called *HeatWatch*, which exploits the awareness of the workload's write-intensity and the SSD temperature in the flash controller to improve flash reliability. Through the characterization, we observe that (1) a longer dwell time slows down flash retention loss speed, and (2) high temperature accelerates flash retention loss but improves program accuracy. We develop URT, a unified model for the combined effects of self-recovery, temperature, retention loss, and wearout. Our evaluations show that URT accurately predicts 3D NAND device characteristics. HeatWatch efficiently tracks the dwell time of the workload and the temperature of the SSD, and uses URT to dynamically adjust the read reference voltage to the workload write-intensity and SSD temperature. Our evaluations on 28 real workload traces show that, by accurately predicting and applying the optimal read reference voltage, HeatWatch improves flash lifetime by 3.85×, over a baseline that uses a fixed read reference voltage.

Finally, this dissertation provides an analysis of the system-level implications of our proposed techniques. We show that, aside from improving flash lifetime, our techniques can also improve tolerable write frequency and reduce ECC cost when a longer flash lifetime is not needed. In addition, we show that our techniques may reduce read latency and add very little overhead to the existing flash management policies.

Chapter 10

Future Research Directions

This dissertation has shown several example approaches that significantly improve NAND flash reliability at a low cost by making the flash controller device- and workload-aware. We believe that it is promising to continue exploring along this direction, because (1) NAND flash memory continues to grow in population as it becomes a cheaper and more accessible technology and (2) flash reliability issue will grow as flash vendors more aggressively increase the density of NAND flash memory to satisfy the demand. In this chapter, we describe potential research directions to further improve the reliability and efficiency of NAND flash memory in a system.

10.1 Temperature Effects on Read Operations

In Chapter 7, we have shown that SSD temperature significantly affects retention loss speed and program variation of NAND flash memory. Recent work shows that temperature could also affect read operations significantly (i.e., *read temperature variation*) [184], and proposes circuit-level techniques to compensate for this type of temperature variation [54, 314]. In reality, read temperatures can change significantly in a very short time, especially on mobile devices [27], leading to a significant increase in raw bit error rate under extreme temperatures.

To the best of our knowledge, there is no characterization data or model available for read temperature effect in open literature. So we believe that it is valuable to study this phenomenon and propose flash controller techniques to tolerate the read temperature variation. We believe that the flash controller can learn an accurate online model, like the ones we use in Chapter 5, 6, and 7, to compensate for the read temperature variation better than existing circuit-level techniques. The key challenge in understanding the read temperature effect is to design a rigorous testing procedure that eliminates the potential noise caused by the unknown amount of retention errors introduced when the read temperature slowly converges to the target level. It is difficult to accurately quantify the number of retention errors during this time because the temperature, and hence the retention loss speed, change at a variable speed. Hence, one potential way to rigorously characterize read temperature effect is to perform the characterization under low temperature such that the number of newly introduced retention errors is minimal. Another potential way is to constantly monitor the temperature and use integration to compute the retention loss scaled by the varying temperature, like HeatWatch does when precomputing the temperature

amplification factor (Section 7.3), and account for the retention errors in the characterization.

To conclude, we believe it is interesting to (1) design a rigorous testing procedure to characterize read temperature variation, (2) understand the cause of read temperature variation, e.g., it might be caused by the temperature sensitivity of the sense amplifier, (3) develop new models for read temperature variation, and (4) design new flash controller techniques to mitigate and compensate for read temperature variation.

10.2 SSD Errors At Scale

Today's data center servers already use SSDs as a high-performance alternative to hard disk drives to store frequently-accessed persistent data. As the storage density of NAND-flash-based SSDs continues to increase, and the price of SSDs continues to decrease, data center servers are increasingly likely to deploy SSDs instead of hard disk drives as the primary storage medium. This creates both opportunities and challenges for improving SSD reliability. In this section, we discuss these opportunities and challenges, and discuss several potential research directions for improving SSD reliability at scale.

10.2.1 3D NAND Errors In the Field

As we have discussed in Section 3.1.7, recent works have analyzed the reliability of hundreds of thousands of SSDs in production data centers [211, 240, 241, 285]. However, none of these works include any analysis of SSDs using 3D NAND because 3D NAND technology has only been recently introduced and has not accumulated enough device hours in the field to perform long-term studies. As we have discussed in Section 3.4, 3D NAND devices have a different flash cell design, a different flash chip organization, and use a larger manufacturing process technology than planar NAND devices. Thereby, as we have shown in Section 6.2 using a controlled error study, 3D NAND devices have different error characteristics from planar NAND. Hence, we expect that 3D NAND devices in the field will also demonstrate different reliability characteristics.

In addition, the density of future 3D NAND devices is increased using different methods than for planar NAND. For planar NAND, manufacturers increased its density in each product generation using aggressive process technology shrinking. This, unfortunately, decreases the flash cell size as well as the distance between cells, thus significantly decreasing flash reliability. For 3D NAND, in the foreseeable future, manufacturers can increase storage density by increasing the number of stacked layers instead of using aggressive process technology scaling. Hence, future generation 3D NAND devices are more likely have larger layer-to-layer process variation. Thus, using a large-scale field study could allow us to observe new challenges for scaling the 3D NAND devices as data centers will deploy multiple generations of 3D NAND chips over time.

To conclude, we believe it is interesting to (1) investigate 3D NAND error characteristics in the field, and compare the characteristics to those of planar NAND, (2) compare 3D NAND error characteristics across multiple generations to identify new challenges in scaling flash density, such as layer-to-layer process variation, (3) investigate SSD failure due to other components than

flash chips, such as the controller and the command or data bus, (4) investigate 3D NAND chip-to-chip process variation by comparing error characteristics across many devices, and (5) investigate and compare the effectiveness of various state-of-the-art error mitigation or error recovery techniques in the field, such as WARM (Chapter 4), online flash channel modeling (Chapter 5), HeatWatch (Chapter 7), RFR [27], and RDR [35]. We believe that the insights derived from these investigations can enable and inspire new techniques, especially new system-level techniques [110], to tolerate SSD errors more efficiently.

10.2.2 Predicting and Preventing SSD Failures

SSD failures caused by uncorrectable errors on a single SSD are designed to be very infrequent (e.g., typically less than 10^{-15}), due to the use of strong ECC within the controller. In a large-scale data center, however, the occurrence of these infrequent uncorrectable errors is amplified due to the use of *hundreds of thousands* of SSDs at the same time. As a result, uncorrectable errors become a relatively frequent event in a data center, which can result in frequent machine failures and data loss.

SSD failures caused by component failures are also infrequent on a single SSD, because each SSD contains only a few components in total [32, 33, 89, 219]. In a data center, however, SSD component failures are frequent because the data center consists of thousands of SSD controllers, millions of flash chips, and millions of data buses [213, 241, 285]. Such component failures can be catastrophic, because they can lead to many uncorrectable errors at the same time, and to immediate loss of large chunks of data.

Fortunately, these SSD failures are typically preceded by early warning signs, such as correctable errors, and have strong spatial locality [213, 241, 285]. Thus, we believe it is possible to prevent a majority of these failures by predicting them in advance. We believe that by investigating SSD failures in the field (Section 10.2.1), we can identify predictable patterns of these SSD failures. Based on these patterns, we can (1) develop models of SSD failures, including models of uncorrectable errors and models of SSD component failures, (2) predict SSD failures before they happen, and (3) design mechanisms to prevent a majority of SSD failures from affecting system and data reliability, e.g., taking SSDs offline before failure happens, or duplicating data to more reliable SSDs in advance.

10.2.3 Tolerating Reliability Variation Across SSDs

Improving and managing SSD reliability at a larger scale introduces new challenges, as reliability can vary significantly across different SSDs within a data center. There are *four* major reasons for the variation in SSD reliability. First, the SSDs within a data center might have been deployed at different times. Thus, different batches of SSDs may use different generations of flash technology, which have very different reliabilities. They may also have different deployment dates, leading to different amounts of wearout on the SSD. Second, even for SSDs within the same batch, SSD reliability can vary due to the process variation across different flash chips. For example, some SSDs may consist of more reliable flash chips which have longer endurance and can store data for a longer retention time. Third, even in an ideal world without any process variation, each SSD within a data center may run a different workload throughout its lifetime. For example,

write intensity can vary by as much as 5,500× across different workloads [22, 110], leading to drastically different P/E cycle lifetimes for different SSDs. The reliability variation can cause failures to happen sooner on some SSDs than the designed lifetime of the SSD, requiring more frequent SSD replacement in a data center.

We believe that by tolerating reliability variation across SSDs in a data center, the overall storage reliability can be improved, and the cost for managing SSD reliability can be reduced. There can be many ways to tolerate this variation. For example, (1) we can apply RAID across different SSDs and tolerate SSD failures using erasure codes [297]. To minimize the RAID failure rate, less reliable SSDs should be evenly distributed across different RAID groups instead of being arbitrarily grouped together. As another example, (2) we can either apply global wear-leveling across all SSDs within a data center to mitigate such variation [110], or move the data (e.g., write-hot vs. write-cold data) to its most suitable SSD (e.g., move write-hot data to less reliable SSDs) [194, 199]. We believe it is interesting to compare the benefit of these two solutions for different use cases.

10.3 Enabling Cold Storage in SSDs

SSDs are already popular for storing frequently-accessed data in data centers. The key challenges for the larger-scale deployment of SSDs are (1) higher cost compared to hard disk drives, and (2) limited retention time guarantee. The advent of 3D NAND technology increases the density potential of NAND flash memory, but increases manufacturing costs compared to planar NAND due to the more complex manufacturing process required. Furthermore, as we have shown in Chapter 6, due to early retention loss, 3D NAND also has greater retention errors than planar NAND. Thus, we believe it is important to reduce SSD cost by increasing both storage density and SSD retention time. In this section, we discuss the problems and potential directions associated with each potential solution.

10.3.1 Identifying Suitable Data for SSD Cold Storage

Similar to our approach for WARM (Chapter 4), we would like to identify suitable data for cold storage in SSDs and manage them in a more efficient way. For flash-based SSDs, write operations not only consume P/E cycle lifetime but also reduce the dwell time of the SSD, which accelerates retention loss (Section 7.1). Thus, write-cold data is more suitable for SSD cold storage. However, infrequently-accessed data (i.e., read-cold and write-cold data) should be stored in the cheapest possible storage such as tape. Thus, we believe write-cold, read-hot data can benefit the most from the fast random access performance of SSD cold storage.

We believe it is interesting to investigate the best technique to identify write-cold, read-hot data, such as: (1) identifying data used by read-only applications, which requires frequent accesses to large amounts of static data that do not change, (2) identifying write-cold data within each SSD using multiple queues like WARM, Bloom filters [186], log-structured merge trees [249], or any other write-cold data identification techniques, or (3) using programmer annotations [133]. After we identify such data, the SSD controller or the storage manager can decide to aggregate the write-cold, write-hot data and manage it in a more efficient way.

10.3.2 Increasing SSD Retention Time

To use SSDs for cold storage, the SSDs must provide a long enough retention time. Otherwise, as we have shown in WARM, frequent refreshes will eat away the majority of the P/E cycle lifetime and potentially reduce SSD performance. SSD cold storage has several properties that could benefit SSD reliability and extend the retention time. First, write-cold data is seldom updated, and thus consumes fewer P/E cycles. As a result, SSDs used for cold data have a longer lifetime. Second, as we have shown in Section 7.1, thanks to flash memory self-recovery, a longer dwell time slows down flash retention loss. Thus, if an SSD contains only write-cold data, the dwell time of all flash blocks within the SSD will be long, increasing the retention time of the SSD.

We believe it is interesting to investigate techniques to further improve the retention time for SSD cold storage to make it more appealing than cold data storage using hard disk drives. First, we can investigate techniques that partition the data identified for SSD cold storage (e.g., write-cold, read-hot data) to a separate pool of SSDs or flash chips. This will reduce the dwell time for cold data, and increase the dwell time for other data. This can lead to more efficient flash policies for all data, as it reduces the retention loss speed for cold data (Chapter 7), and reduces the required retention time guarantee for the other data (Chapter 4). Second, we can investigate suitable flash management policies for write-cold data. Note that, according to our findings in Section 6.3, read disturb errors are minimal in 3D NAND. Also note that, for write-cold data, the vast majority of writes are for refresh operations, and thus its dwell time is approximately equal to its retention time. This could allow for much higher lifetimes for SSDs used for cold storage. Third, once write-cold data is stored on separate SSDs, we can even manage SSD cooling by using the best storage temperature that maximizes SSD retention time. Since writes are infrequent in cold storage, we can schedule them when the temperature is high, which reduces SSD programming errors. Fourth, we can apply retention error recovery techniques to further relax retention time constraints for SSD cold storage. Since the data in cold storage will remain static for a long time, the SSD errors in cold storage are dominated by retention errors. Thus, retention error recovery can be very effective at correcting errors in SSD cold storage even when the raw bit error rate exceeds the ECC correction capability.

10.3.3 Increasing SSD Capacity

Scaling NAND flash density is hard because it often trades off flash reliability. In SSD cold storage, however, we can give up certain unused SSD reliability and performance to increase SSD capacity. This lowers the cost-per-bit of SSD cold storage, which makes it more appealing to use. We believe there are two reasons we can trade off some aspects of SSD reliability and performance for cold storage. First, since SSD cold storage does not require a high P/E cycle lifetime, we can limit SSD lifetime to only a few hundred P/E cycles. This allows for much more aggressive scaling to increase the density of SSD cold storage, leading to a higher cost-efficiency. Second, since the write performance is less important for cold storage, we can use SSD write modes with long delay. By trading off P/E cycle lifetime and write performance, we believe we can significantly improve SSD capacity, and hence reduce the cost for SSD cold storage through many ways. For example, we can configure the flash chips to TLC or QLC mode for SSD cold storage, which increases the number of bits stored on each flash cell, in turn reducing the error

margin and increasing SSD write latency. To compensate for the potentially increased raw bit error rate in TLC or QLC NAND flash memory, we can develop more aggressive error mitigation techniques tailored for SSD cold storage to tolerate these errors. For example, we can configure SSD cold storage to use a smaller program step size to trade off write performance. This could allow us to use weaker ECC on SSD cold storage, which frees up SSD capacity that originally used for ECC redundancy.

Other Works of This Author

During the course of my Ph.D., I have had the opportunity to collaborate with many of my fellow graduate students. These projects not only help me to learn about NAND flash memory, DRAM, and cost-reliability trade-offs, which is useful for this dissertation, but also help me gain insights, ideas, and skills to conduct good research, which is useful when writing this dissertation. In this chapter, I would like to acknowledge these projects and my other works related to this dissertation.

In collaboration with Justin Meza, I have worked on single-level storage systems. We rethought the interface for efficiently managing a heterogeneous memory and storage system which consists of DRAM, NVM, NAND flash memory, and hard disk drive. We explored the design of a *Persistent Memory Manager* that coordinates the management of memory and storage under a single hardware unit in a single address space [210]. Efficient management of NVM and flash reliability is one of the concerns for designing the persistent memory manager.

In collaboration with Vivek Seshadri, I have worked on processing using DRAM. We propose RowClone, a new and simple mechanism to perform bulk copy and initialization completely within DRAM using the internal row buffers [291]. This gave me the opportunity to learn DRAM architecture, which has a lot in common with NAND flash memory.

During my PhD, I have also worked on improving the cost-reliability trade-offs of DRAMs in data centers. First, we propose the idea of *heterogeneous-reliability memory*, which is a hardware/software cooperative system design that chooses the best memory reliability provisioning for each chunk of data to lower data center cost while achieving high reliability [193]. Second, we propose *Capacity- and Reliability-Adaptive Memory* (CREAM), a hardware mechanism that adapts error-correcting DRAM modules to offer multiple levels of error protection, and provides the capacity saved from using weaker protection to applications [197].

In collaboration with Yu Cai, I have worked on many ideas to improve flash reliability aside from my dissertation work. Over the years, we propose several online or offline techniques to mitigate data retention errors [27], read disturb errors [35], and program errors [34]. We have also worked on a survey and tutorial of the best practices for error characterization, mitigation, and recovery in flash-based SSDs [33]. In collaboration with Aya Fukami, I have worked on improving the reliability of chip-off forensic analysis of NAND flash memory devices [82]. In collaboration with Arash, we have worked on improving the performance and fairness of the I/O request scheduler in the SSD controller [317]. All of these works are closely related to this dissertation. During these collaborations, we all gained valuable expertise on improving flash reliability by learning from each other.

Bibliography

[1] Michael Abraham. NAND Flash Architecture and Specification Trends. In *Flash Memory Summit*, 2012. 4.3

[2] N. Agrawal, V. Prabhakaran, and T. Wobber. Design Tradeoffs for SSD Performance. In *USENIX ATC*, 2008. 2.1.1, 2.1.2, 2.1.3, 4.3

[3] Saba Ahmadian, Farhad Taheri, Mehrshad Lotfi, Maryam Karimi, and Hossein Asadi. Investigating Power Outage Effects on Reliability of Solid-State Drives. In *DATE*, 2018. 2.1.3

[4] A. R. Alameldeen, I. Wagner, Z. Chisthi, W. Wu, C. Wilkerson, and S.-L. Lu. Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes. In *ISCA*, 2011. 3.5.6

[5] A. Anastasopoulos. A Comparison Between the Sum-Product and the Min-Sum Iterative Detection Algorithms Based on Density Evolution. In *GLOBECOM*, 2001. 3.3.1

[6] Svante Arrhenius. Über die dissociationswärme und den einfluss der temperatur auf den dissociationsgrad der elektrolyte. *Zeitschrift für physikalische Chemie*, 1889. 3.1.6, 3.2.3, 4, 7.1.3, 7.1.3, 7.2.2

[7] A. Athmanathan, M. Stanisavljevic, N. Papandreou, H. Pozidis, and E. Eleftheriou. Multilevel-Cell Phase-Change Memory: A Viable Technology. *J. Emerg. Sel. Topics Circuits Syst.*, Mar. 2016. 3.5.7

[8] S. Baek, S. Cho, and R. Melhem. Refresh Now and Then. *IEEE Trans. Computers*, Aug. 2014. 3.5.2, 3.5.2

[9] Mahesh Balakrishnan, Asim Kadav, Vijayan Prabhakaran, and Dahlia Malkhi. Differential raid: Rethinking raid for ssd reliability. *ACM TOS*, 2010. 6.5.2

[10] Hanmant P Belgal, Nick Righos, Ivan Kalastirsky, Jeff J Peterson, Robert Shiner, and Neal Mielke. A New Reliability Model for Post-Cycling Charge Retention of Flash Memories. In *IRPS*, 2002. 7.2.2

[11] E. R. Berlekamp. Nonbinary BCH Decoding. In *ISIT*, 1967. 3.3.1

[12] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti. Introduction to Flash Memory. *Proc. IEEE*, Apr. 2003. 2.2, 2.2.4, 2.2.4, 2.2.4

[13] Andrew D. Birrell and Bruce Jay Nelson. Implementing Remote Procedure Calls. *TOCS*, Feb. 1984. 7.1.1

[14] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. On A Class of Error Correcting

Binary Group Codes. *Information and control*, 1960. 2.1.3, 3.3, 3.3.1, 3.3.1, 8.1.2

[15] E. Bosman, K. Razavi, H. Bos, and C. Guiffrida. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In *SP*, 2016. 3.5.3

[16] J. E. Brewer and M. Gill. *Nonvolatile Memory Technologies With Emphasis on Flash: A Comprehensive Guide to Understanding and Using NVM Devices*. Wiley, Hoboken, NJ, USA, 2008. 2.2, 2.2.4, 2.2.4

[17] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger. The DiskSim Simulation Environment Version 4.0 Reference Manual. Technical Report CMU-PDL-08-101, Carnegie Mellon Univ. Parallel Data Lab, 2008. 4.3

[18] W. Burleson, O. Mutlu, and M. Tiwari. Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer. In *DAC*, 2016. 3.5.3

[19] Y. Cai. *NAND Flash Memory: Characterization, Analysis, Modelling, and Mechanisms*. PhD thesis, Carnegie Mellon Univ., 2012. 3.1.1, 3.1.3, 3.1.4, 3.2.3, 3.2.3, 3.2.5

[20] Y. Cai, E. F. Haratsch, M. P. McCartney, and K. Mai. FPGA-Based Solid-State Drive Prototyping Platform. In *FCCM*, 2011. 5.2, 6.1.1

[21] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis. In *DATE*, 2012. 1.1.1, 1.2.1, 1.2.2, 2.1.3, 3.1.1, 3.1.4, 3.1.5, 3.1, 3.2.3, 3.2.3, 4.1, 4.1.1, 4.3, 5.6, 6, 6.1, 7.1.2, 7.1.3

[22] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai. Flash Correct and Refresh: Retention Aware Management for Increased Lifetime. In *ICCD*, 2012. 1.1.1, 1.1.2, 1.2.1, 1.2.2, 1.2.3, 1.1., 1.2., 2.1.3, 2.1.4, 3.1.4, 3.1, 3.2.3, 3.2.3, 4, 4.1, 4.1.1, 4.1, 4.1.2, 4.2.2, 4.3, 4.3, 4.4, 5.5.2, 5.6, 5.7, 6, 6.1, 6.2.2, 6.2.2, 6.3.2, 6.5.3, 7.2.2, 7.3.1, 7.3.1, 7.4, 10.2.3

[23] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. Threshold Voltage Distribution in NAND Flash Memory: Characterization, Analysis, and Modeling. In *DATE*, 2013. 1.1.1, 1.1.1, 1.2.1, 1.2.2, 2.2.4, 3.1.1, 3.1, 3.2.4, 3.2.5, 3.3.1, 3.3.1, 5.2, 5.3, 5.3.1, 5.3.1, 5.6, 6, 6.1, 6.2.4, 6.3.1, 6.3.1, 7.1.1, 2, 7.2.1, 7.3.1, 7.3.3, 7.4

[24] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai. Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation. In *ICCD*, 2013. 1.1.1, 1.2.1, 1.2.2, 1.2.3, 2.1.3, 2.2.4, 3.1.3, 3.1.3, 3.1, 3.2.1, 3.2.3, 3.2.5, 3.4.2, 5.5.1, 5.5.2, 5.6, 6, 6.1, 6.2.4, 6.3.1, 6.3.1, 6.5.2, 6.5.4, 6.6, 7.2.1, 7.3.1, 7.3.3, 7.4

[25] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai. Error Analysis and Retention-Aware Error Management for NAND Flash Memory. *Intel Technology Journal (ITJ)*, 2013. 1.1.2, 1.2.2, 1.1., 1.2., 2.1.3, 3.1.4, 3.1, 3.2.3, 3.2.3, 4, 4.1, 4.1.1, 4.1, 4.1.2, 4.3, 4.3, 4.4, 5.6, 7.2.2, 7.4

[26] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai. Neighbor Cell Assisted Error Correction in MLC NAND Flash Memories. In *SIGMETRICS*, 2014. 1.1.1, 1.2.1, 1.2.2, 1.2.3, 3.7., 2.1.3, 3.1.3, 3.1.3, 3.1, 3.2.2, 3.2.2, 3.2.3, 3.2.5, 3.4.2, 6, 6.1, 6.2.4, 6.3.1, 6.3.1, 6.5.2, 6.5.4

[27] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu. Data Retention in MLC NAND

Flash Memory: Characterization, Optimization, and Recovery. In *HPCA*, 2015. 1.1.1, 1.2.1, 1.2.2, 2.1.3, 3.1.4, 3.1.6, 3.1, 3.2.3, 3.2.4, 3.2.5, 3.2.5, 3.2.5, 3.3.4, 3.3.4, 4, 5.1, 5.2, 5.5.1, 5.5.2, 5.5.5, 5.6, 6, 6.1, 2, 6.2.2, 6.2.2, 6.3.2, 6.4.1, 6.5.1, 6.5.3, 6.5.5, 6.5.5, 7.1.1, 2, 7.2.2, 7.2.2, 7.3.1, 7.3.1, 7.3.2, 7.3.3, 7.4, 10.1, 10.2.1, 10.3.3

[28] Y. Cai, Y. Wu, and E. F. Haratsch. Hot-Read Data Aggregation and Code Selection. U.S. Patent Appl. 14/192,110, 2015. 3.2.6, 3.2.7

[29] Y. Cai, Y. Wu, and E. F. Haratsch. System to Control a Width of a Programming Threshold Voltage Distribution Width When Writing Hot-Read Data. U.S. Patent 9,218,885, 2015. 3.2.6, 3.2.7, 3.2.7

[30] Y. Cai, Y. Wu, N. Chen, E. F. Haratsch, and Z. Chen. Systems and Methods for Latency Based Data Recycling in a Solid State Memory System. U.S. Patent 9,424,179, 2016. 3.2.3

[31] Y. Cai, Y. Wu, and E. F. Haratsch. Error Correction Code (ECC) Selection Using Probability Density Functions of Error Correction Capability in Storage Controllers With Multiple Error Correction Codes. U.S. Patent 9,419,655, 2016. 3.1, 3.2.7, 3.2.7

[32] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu. Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery. arXiv:1711.11427 [cs.AR], 2017. 1.2.2, 1.2.3, 1.4, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 3.1, 3.2, 3.3, 3.4, 3.1.6, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, 3.15, 3.16, 3.17, 3.18, 3.22, 3.23, 3.24, 3.25, 7.1.1, 7.3.1, 7.3.1, 7.3.2, 7.3.3, 7.4, 8.1.2, 10.2.2

[33] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. *Proc. IEEE*, Sep. 2017. 1.1.1, 1.2.1, 1.2.2, 1.2.3, 1.4, 3.1.6, 6, 6.5.2, 6.5.5, 2, 7.1.1, 7.3.1, 7.3.1, 7.3.2, 7.3.3, 7.4, 8.1.2, 10.2.2, 10.3.3

[34] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch. Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques. In *HPCA*, 2017. 1.1.1, 1.2.1, 1.2.2, 2.2.4, 3.1.2, 3.1.2, 3.1.3, 3.1, 3.2.1, 3.5.3, 6, 6.2.4, 10.3.3

[35] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *DSN*, 2015. 1.1.1, 1.2.1, 1.2.2, 1.2.3, 2.1.3, 2.2.3, 3.1.5, 3.1, 3.2.3, 3.2.5, 3.2.5, 3.3.4, 3.3.4, 3.4.2, 5.1, 5.5.1, 6, 6.1, 6.2.4, 6.3.3, 6.3.3, 6.6, 2, 10.2.1, 10.3.3

[36] J. Cha and S. Kang. Data Randomization Scheme for Endurance Enhancement and Interference Mitigation of Multilevel Flash Memory Devices. *ETRI Journal*, Feb. 2013. 2.1.3

[37] Karthik Chandrasekar, Sven Goossens, Christian Weis, Martijn Koedam, Benny Akesson, Norbert Wehn, and Kees Goossens. Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization. In *DATE*, 2014. 3.5.5

[38] K. K. Chang. *Understanding and Improving the Latency of DRAM-Based Memory Systems*. PhD thesis, Carnegie Mellon Univ., 2017. 3.5.1, 3.5.2, 3.5.2, 3.5.5, 3.5.5

[39] K. K. Chang, Donghyuk Lee, Z. Chishti, A.R. Alameldeen, C. Wilkerson, Yoongu Kim, and O. Mutlu. Improving DRAM Performance by Parallelizing Refreshes With Accesses. In *HPCA*, 2014. 3.2.3, 3.5.2, 3.5.2

[40] K. K. Chang, P. J. Nair, S. Ghose, D. Lee, M. K. Qureshi, and O. Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*, 2016. 2.1.1

[41] Kevin K. Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization. In *SIGMETRICS*, 2016. 3.5.2, 3.5.5, 5, 3.35, 3.5.5

[42] Kevin K. Chang, Abdullah Giray Yaglikci, Aditya Agrawal, Niladrish Chatterjee, Saugata Ghose, Abhijith Kashyap, Hasan Hassan, Donghyuk Lee, Mike O'Connor, and Onur Mutlu. Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. In *SIGMETRICS*, 2017. 3.5.5, 3.5.5

[43] L.-P. Chang. On Efficient Wear Leveling for Large-Scale Flash-Memory Storage Systems. In *SAC*, 2007. 2.1.3, 7.3.1, 7.3.2

[44] L.-P. Chang, T.-W. Kuo, and S.-W. Lo. Real-Time Garbage Collection for Flash-Memory Storage Systems of Real-Time Embedded Systems. *ACM Trans. Embedded Comput. Syst.*, Nov. 2004. 2.1.3, 2.1.3, 2.1.4

[45] Yu-Ming Chang, Yuan-Hao Chang, Jian-Jia Chen, Tei-Wei Kuo, Hsiang-Pang Li, and Hang-Ting Lue. On Trading Wear-Leveling With Heal-Leveling. In *DAC*, 2014. 3.1.6

[46] Niladrish Chatterjee, Manjunath Shevgoor, Rajeev Balasubramonian, Al Davis, Zhen Fang, Ramesh Illikkal, and Ravi Iyer. Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access. In *MICRO*, 2012. 3.5.7

[47] Chih-Ping Chen, Hang-Ting Lue, Chih-Chang Hsieh, Kuo-Pin Chang, Kuang-Yeu Hsieh, and Chih-Yuan Lu. Study of fast initial charge loss and it's impact on the programmed states Vt distribution of charge-trapping NAND Flash. In *IEDM*, 2010. 6.2.2

[48] Chin-Long Chen. High-Speed Decoding of BCH Codes (Corresp.). *IEEE Trans. Inf. Theory*, Mar. 1981. 3.3.1, 3.3.1, 6.5.5

[49] J. Chen and M. P. C. Fossorier. Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity Check Codes. *IEEE Trans. Comm.*, Aug. 2002. 3.3.1

[50] Renhai Chen, Yi Wang, and Zili Shao. DHeating: Dispersed Heating Repair for Self-Healing NAND Flash Memory. In *CODES+ISSS*, 2013. 3.1.6, 7.1.2, 7.1.2, 7.1.4

[51] T.-H. Chen, Y.-Y. Hsiao, Y.-T. Hsing, and C.-W. Wu. An Adaptive-Rate Error Correction Scheme for NAND Flash Memory. In *VTS*, 2009. 3.1, 3.2.7, 3.2.7

[52] Z. Chen, E. F. Haratsch, S. Sankaranarayanan, and Y. Wu. Estimating Read Reference Voltage Based on Disparity and Derivative Metrics. U.S. Patent 9,417,797, 2016. 3.2.5

[53] R. T. Chien. Cyclic Decoding Procedures for the Bose–Chaudhuri–Hocquenghem Codes. *IEEE Trans. Inf. Theory*, Oct. 1964. 3.3.1

[54] Tae-hee Cho and Yeong-Taek Lee. Multi-level flash memory with temperature compensation, 2005. US Patent 6,870,766. 10.1

[55] Bongsik Choi, Sang Hyun Jang, Jinsu Yoon, Juhee Lee, Minsu Jeon, Yongwoo Lee, Jungmin Han, Jieun Lee, Dong Myong Kim, Dae Hwan Kim, Chan Lim, Sungkye Park, and Sung-Jin Choi. Comprehensive Evaluation of Early Retention (Fast Charge Loss Within a Few Seconds) Characteristics in Tube-Type 3-D NAND Flash Memory. In *VLSIT*, 2016. 3.1.6, 3.4.2, 3.4.3, 6.1, 6.2.2, 6.2.2, 6.2.3, 6.3.1, 7.1.1, 7.2.2, 7.4

[56] H. Choi, W. Liu, and W. Sung. VLSI Implementation of BCH Error Correction for Multilevel Cell NAND Flash Memory. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Jul. 2009. 3.3.1

[57] Wonil Choi, Mohammad Arjomand, Myoungsoo Jung, and Mahmut Kandemir. Exploiting Data Longevity for Enhancing the Lifetime of Flash-based Storage Class Memory. In *SIGMETRICS*, 2017. 7.4

[58] C. Chou, P. Nair, and M. K. Qureshi. Reducing Refresh Power in Mobile Devices With Morphable ECC. In *DSN*, 2015. 3.5.6

[59] C.-C. Chou, A. Jaleel, and M. K. Qureshi. CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In *MICRO*, 2014. 3.5.7

[60] C.-C. Chou, A. Jaleel, and M. K. Qureshi. BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches. In *ISCA*, 2015. 3.5.7

[61] Siddharth Choudhuri and Tony Givargis. Deterministic Service Guarantees for NAND Flash Using Partial Block Cleaning. In *CODES+ISSS*, 2008. 2.1.3

[62] L. Chua. Memristor—The Missing Circuit Element. *IEEE Trans. Circuit Theory*, Sep. 1971. 3.5.7

[63] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee S.-W. Lee, and H.-J. Song. A Survey of Flash Translation Layer. *J. Syst. Archit.*, May/Jun. 2009. 2.1.3

[64] R. Codandaramane. Securing the SSDs — NVMe Controller Encryption. In *Flash Memory Summit*, 2016. 2.1.3

[65] E. T. Cohen. Zero-One Balance Management in a Solid-State Disk Controller. U.S. Patent 8,839,073, 2014. 3.2.5

[66] E. T. Cohen, Y. Cai, E. F. Haratsch, and Y. Wu. Method to Dynamically Update LLRs in an SSD Drive and/or Controller. U.S. Patent 9,329,935, 2015. 3.3.1

[67] Christian Monzio Compagnoni, Carmine Miccoli, Riccardo Mottadelli, Silvia Beltrami, Michele Ghidotti, Andrea L Lacaita, Alessandro S Spinelli, and Angelo Visconti. Investigation of The Threshold Voltage Instability After Distributed Cycling in Nanoscale NAND Flash Memory Arrays. In *IRPS*, 2010. 7.1.4

[68] Jim Cooke. The Inconvenient Truths of NAND Flash Memory. *Flash Memory Summit*, 2007. 3.1.3, 3.1.5

[69] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, Berlin, Germany;

Heidelberg, Germany; New York, NY, USA, 2002. 2.1.3

[70] Anup Das, Hasan Hassan, and Onur Mutlu. VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency. In *DAC*, 2018. 3.5.2

[71] Niv Dayan, Philippe Bonnet, and Stratos Idreos. GeckoFTL: Scalable Flash Translation Techniques For Very Large Flash Devices. In *SIGMOD*, 2016. 1.2.3

[72] Eric Deal. Trends in NAND Flash Memory Error Correction. *Cyclic Design*, 2009. 6.5.5, 8.1.2

[73] R. Degraeve, F. Schuler, B. Kaczer, M. Lorenzini, D. Wellekens, P. Hendrickx, M. van Duuren, G. J. M. Dormans, J. Van Houdt, L. Haspeslagh, G. Groeseneken, and G. Tempel. Analytical Percolation Model for Predicting Anomalous Charge Loss in Flash Memories. *IEEE Trans. Electron Devices*, Sep. 2004. 2.2.4, 3.1.4

[74] T. J. Dell. A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. Technical report, IBM Microelectron. Division, 1997. 3.5.6

[75] P. Desnoyers. Analytic Modeling of SSD Write Performance. In *SYSTOR*, 2012. 2.1.4

[76] C. Dirik and B. Jacob. The Performance of PC Solid-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization. In *ISCA*, 2009. 2.1.3

[77] Lara Dolecek. Making Error Correcting Codes Work for Flash Memory. In *Flash Memory Summit*, 2014. 3.3.1, 3.3.1, 3.3.1, 3.3.2

[78] Guiqiang Dong, Ningde Xie, and Tong Zhang. Enabling NAND Flash Memory Use Soft-Decision Error Correction Codes at Minimal Read Latency Overhead. *IEEE Trans. on Circuits and Systems*, 2013. 5.5.4, 5.6

[79] Jim Elliott and Jaeheon Jeong. Advancements in SSDs and 3D NAND Reshaping Storage Market. Keynote presentation at *Flash Memory Summit*, 2017. 3.4.1

[80] M. P. C. Fossorier, M. Mihaljević, and H. Imai. Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation. *IEEE Trans. Comm.*, May 1999. 3.3.1

[81] R. H. Fowler and L. Nordheim. Electron Emission in Intense Electric Fields. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 1928. 2.2.4, 3.4.1

[82] A. Fukami, S. Ghose, Y. Luo, Y. Cai, and O. Mutlu. Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices. *Digital Investigation*, Mar 2017. 3.1, 3.2.4, 7.1.1, 10.3.3

[83] E. Gal and S. Toledo. Algorithms and Data Structures for Flash Memories. *ACM Comput. Surv.*, Jun. 2005. 2.1.3, 7.3.1, 7.3.2

[84] R. G. Gallager. Low-Density Parity-Check Codes. *IRE Trans. Inf. Theory*, Jan. 1962. 2.1.3, 3.3, 3.3.1, 3.3.1, 3.3.1, 3.3.2

[85] Robert G Gallager. Low-Density Parity-Check Codes. *Information Theory, IRE Transactions on*, 1962. 2.1.3, 3.3, 3.3.1, 3.3.1, 3.3.1, 3.3.2

[86] Geoff Gasior. The SSD Endurance Experiment: They're All Dead. http://techreport.com/review/27909/the-ssd-endurance-experiment-theyre-all-dead, 2015. 5.5.3

[87] S. Ghose, H. Lee, and J. F. Martínez. Improving Memory Scheduling via Processor-Side Load Criticality Information. In *ISCA*, 2013. 2.1.2

[88] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *MICRO*, 2009. 3.1.3, 3.1.5, 3.1

[89] Laura M Grupp, John D Davis, and Steven Swanson. The Bleak Future of NAND Flash Memory. In *FAST*, 2012. 7.3.2, 10.2.2

[90] D. Gruss, C. Maurice, and S. Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In *DIMVA*, 2016. 3.5.3

[91] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom. Another Flip in the Wall of Rowhammer Defenses. arXiv:1710.00551 [cs.CR], 2017. 3.5.3

[92] K. Gunnam. LDPC Decoding: VLSI Architectures and Implementations. In *Flash Memory Summit*, 2014. 3.3.1

[93] K. K. Gunnam, G. S. Choi, M. B. Yeary, and M. Atiquzzaman. VLSI Architectures for Layered Decoding for Irregular LDPC Codes of WiMax. In *ICC*, 2007. 3.3.1

[94] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *ASPLOS*, 2009. 1.2.3, 2.1.3, 2.1.3

[95] K. Ha, J. Jeong, and J. Kim. A Read-Disturb Management Technique for High-Density NAND Flash Memory. In *APSys*, 2013. 3.1, 3.2.3, 3.2.6, 3.2.7

[96] K. Ha, J. Jeong, and J. Kim. An Integrated Approach for Managing Read Disturbs in High-Density NAND Flash Memory. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, Jul. 2016. 3.1, 3.2.3, 3.2.5, 3.2.6, 3.2.7

[97] T. Hamamoto, S. Sugiura, and S. Sawada. On the Retention Time Distribution of Dynamic Random Access Memory (DRAM). *IEEE Trans. Electron Devices*, Jun. 1998. 3.5.2

[98] Longzhe Han, Yeonseung Ryu, and Keunsoo Yim. CATA: A Garbage Collection Scheme for Flash Memory File Systems. In *UIC*, 2006. 2.1.3

[99] E. F. Haratsch. Media Management for High Density NAND Flash Memories. In *Flash Memory Summit*, 2016. 3.1, 3.2.7, 3.2.7, 3.3.2, 3.3.3

[100] Erich F. Haratsch. LDPC Code Concepts and Performance on High-Density Flash Memory. In *Flash Memory Summit*, 2014. 5.5.2, 5.5.5

[101] Erich F. Haratsch. Controller Concepts for 1y/1z nm and 3D NAND Flash. In *Flash Memory Summit*, 2015. 3.3.3, 3.3.3, 5.1, 5.5.5

[102] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu. SoftMC: A Flexible and Practical Open-Source Infrastructure for

Enabling Experimental DRAM Studies. In *HPCA*, 2017. 2.1.2, 3.5.2, 3.5.2, 3.5.3, 3.5.5, 3.5.5

[103] Hasan Hassan, Gennady Pekhimenko, Nandita Vijaykumar, Vivek Seshadri, Donghyuk Lee, Oguz Ergin, and Onur Mutlu. ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality. In *HPCA*, 2016. 2.1.2

[104] J. Haswell. SSD Architectures to Ensure Security and Performance. In *Flash Memory Summit*, 2016. 2.1.3

[105] Jun He, Sudarsun Kannan, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. The Unwritten Contract of Solid State Drives. In *EuroSys*, 2017. 2.1.3

[106] Jason Heidecker. Flash Memory Reliability: Read, Program, and Erase Latency Versus Endurance Cycling. Technical Report 10-19, Jet Propulsion Lab, 2010. 4.3

[107] A. Hocquenghem. Codes Correcteurs d'Erreurs. *Chiffres*, Sep. 1959. 2.1.3, 3.3, 3.3.1, 3.3.1, 6.5.5

[108] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write Amplification Analysis in Flash-Based Solid State Drives. In *SYSTOR*, 2009. 2.1.4

[109] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang. Performance Impact and Interplay of SSD Parallelism Through Advanced Commands, Allocation Strategy and Data Granularity. In *ICS*, 2011. 3.1.2

[110] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K Qureshi. Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds. In *FAST*, pages 375–390, 2017. 8.1, 10.2.1, 10.2.3

[111] P. Huang, P. Subedi, X. He, S. He, and K. Zhou. FlexECC: Partially Relaxing ECC of MLC SSD for Better Cache Performance. In *USENIX ATC*, 2014. 3.2.7

[112] Chun-Hsiung Hung, Meng-Fan Chang, Yih-Shan Yang, Yao-Jen Kuo, Tzu-Neng Lai, Shin-Jang Shen, Jo-Yu Hsu, Shuo-Nan Hung, Hang-Ting Lue, Yen-Hao Shih, et al. Layer-Aware Program-and-Read Schemes for 3D Stackable Vertical-Gate BE-SONOS NAND Flash Against Cross-Layer Process Variations. *JSSC*, 50(6):1491–1501, 2015. 6.2.1

[113] A. Hwang, I. Stefanovici, and B. Schroeder. Cosmic Rays Dont Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design. In *ASPLOS*, 2012. 3.5.4

[114] D. Ielmini, A. L. Lacaita, and D. Mantegazza. Recovery and Drift Dynamics of Resistance and Threshold Voltages in Phase-Change Memories. *IEEE Trans. Electron Devices*, Apr. 2007. 3.5.7

[115] Jae-Woo Im, Woopyo Jeong, Doo-Hyun Kim, Sangwan Nam, Dong-Kyo Shim, Myung-Hoon Choi, Hyun-Jun Yoon, Dae-Han Kim, Youse Kim, Hyun Wook Park, Dong-Hun Kwak, Sang-Won Park, Seok-Min Yoon, Wook-Ghee Hahn, Jinho Ryu, Sang-Won Shim, Kyung-Tae Kang, Sung-Ho Choi, Jeong-Don Ihm, Young-Sun Min, In-Mo Kim, Doosub Lee, Ji-Ho Cho, Ohsuk Kwon, Ji-Sang Lee, Moosung Kim, Sang-Hyun Joo, Jae-hoon Jang, Sang-Won Hwang, Dae-Seok Byeon, Hyang-Ja Yang, Ki-Tae Park, Kyehyun Kyung,

and Jeong-Hyuk Choi. A 128Gb 3b/Cell V-NAND Flash Memory with 1Gb/s I/O Rate. In *ISSCC*, 2015. 3.4, 6

[116] Intel Corp. *Serial ATA Advanced Host Controller Interface (AHCI) 1.3.1*, 2012. 2.1.3

[117] Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *ISCA*, 2008. 2.1.2

[118] C. Isen and L. John. ESKIMO — Energy Savings Using Semantic Knowledge of Inconsequential Memory Occupancy for DRAM Subsystem. In *MICRO*, 2009. 3.5.2, 3.5.2

[119] J. Jang, H.-S. Kim, W. Cho, H. Cho, J. Kim, S. I. Shim, Y. Jang, J.-H. Jeong, B.-K. Son, D. W. Kim, K. Kim, J.-J. Shim, J. S. Lim, K.-H. Kim, S. Y. Yi, J.-Y. Lim, D. Chung, H.-C. Moon, S. Hwang, J.-W. Lee, Y.-H. Son, U.-I. Chung, and W.-S. Lee. Vertical Cell Array Using TCAT (Terabit Cell Array Transistor) Technology for Ultra High Density NAND Flash Memory. In *VLSIT*, 2009. 3.4.1

[120] JEDEC Solid State Technology Assn. *DDR4 SDRAM Standard*, 2013. 3.2.3, 3.5.2, 3.5.2, 3.5.2

[121] JEDEC Solid State Technology Assn. Failure Mechanisms and Models for Semiconductor Devices. *JEDEC Publication JEP122H*, 2016. 2.1.3, 3.3.1, 5.5.2, 8.1.2

[122] Jaeyong Jeong, Sangwook Shane Hahn, Sungjin Lee, and Jihong Kim. Lifetime Improvement of NAND Flash-Based Storage Systems Using Dynamic Program and Erase Scaling. In *FAST*, 2014. 3.1, 3.2.5, 3.2.7, 5.5.1

[123] Woopyo Jeong, Jae-woo Im, Doo-Hyun Kim, Sang-Wan Nam, Dong-Kyo Shim, Myung-Hoon Choi, Hyun-Jun Yoon, Dae-Han Kim, You-Se Kim, Hyun-Wook Park, et al. A 128 Gb 3b/Cell V-NAND Flash Memory with 1 Gb/s I/O Rate. *JSSC*, Jan. 2016. 7.2.2

[124] JEDEC Standard JESD218. Solid-State Drive (SSD) Requirements and Endurance Test Method. *Arlington, VA, JEDEC Solid State Technology Association*, 2010. 1.1.1, 2.1.3, 3.1.6, 3.3.3, 3.3.3, 5, 7.1.3

[125] JEDEC Standard JESD22-A117C. Electrically Erasable Programmable ROM (EEPROM) Program/Erase Endurance and Data Retention Stress Test. *Arlington, VA, JEDEC Solid State Technology Association*, 2011. 7.1.4

[126] JEDEC Standard JESD91A. Method for Developing Acceleration Models for Electronic Component Failure Mechanisms. *Arlington, VA, JEDEC Solid State Technology Association*, 2003. 3.1.6, 7.2.2, 7.2.2

[127] L. Jiang, Y. Zhang, and J. Yang. Mitigating Write Disturbance in Super-Dense Phase Change Memories. In *DSN*, 2014. 3.5.7

[128] Xiaowei Jiang, N. Madan, Li Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, D. Solihin, and R. Balasubramonian. CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms. In *HPCA*, 2010. 3.5.7

[129] S. J. Johnson. Introducing Low-Density Parity-Check Codes. `http://sigpromu.org/sarah/SJohnsonLDPCintro.pdf`. 3.3.1

[130] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open Source Scientific Tools for

Python. http://www.scipy.org/. 7.2.3

[131] Dongku Kang, Woopyo Jeong, Chulbum Kim, Doo-Hyun Kim, Yong-Sung Cho, Kyung-Tae Kang, Jinho Ryu, Kyung-Min Kang, Sungyeon Lee, Wandong Kim, Hanjun Lee, Jaedoeg Yu, Nayoung Choi, Dong-Su Jang, Jeong-Don Ihm, Doo-Gon Kim, Young-Sun Min, Moosung Kim, Ansoo Park, Jae-Ick Son, In-Mo Kim, Pansuk Kwak, Bong-Kil Jung, Doosub Lee, Hyunggon Kim, Hyang-Ja Yang, Dae-Seok Byeon, Ki-Tae Park, Kye-hyun Kyung, and Jeong-Hyuk Choi. 7.1 256Gb 3b/cell V-NAND Flash Memory With 48 Stacked WL Layers. In *ISSCC*, 2016. 3.4, 3.4.1, 6

[132] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A Superblock-Based Flash Translation Layer for NAND Flash Memory. In *EMSOFT*, 2006. 2.1.3

[133] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The Multi-Streamed Solid-State Drive. In *HotStorage*, 2014. 10.3.1

[134] Uksong Kang, Hak-Soo Yu, Churoo Park, Hongzhong Zheng, John Halbert, Kuljit Bains, SeongJin Jang, and Joosun Choi. Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling. In *Memory Forum*, 2014. 3.5.2, 3.5.2, 3.5.6

[135] J. Katcher. Postmark: A New File System Benchmark. Technical Report TR3022, Network Appliance, 1997. 4.3, 4.2

[136] R. Katsumata, M. Kito, Y. Fukuzumi, M. Kido, H. Tanaka, Y. Komori, M. Ishiduki, J. Matsunami, T. Fujiwara, Y. Nagata, L. Zhang, Y. Iwata, R. Kirisawa, H. Aochi, and A. Nitayama. Pipe-Shaped BiCS Flash Memory with 16 Stacked Layers and Multi-Level-Cell Operation for Ultra High Density Storage Devices. In *VLSIT*, 2009. 3.4.1, 3.4.1

[137] S. Khan, D. Lee, Y. Kim, A. Alameldeen, C. Wilkerson, and O. Mutlu. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014. 3.5.1, 3.5.2, 3.5.2, 3.5.2, 3.5.2, 3.5.6

[138] S. Khan, D. Lee, and O. Mutlu. PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM. In *DSN*, 2016. 3.5.1, 3.5.2, 3.5.2, 3.5.2, 3.5.6

[139] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu. A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM. *IEEE Comput. Archit. Lett.*, 2016. 3.5.1, 3.5.2, 3.5.2, 3.5.2, 3.5.6

[140] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu. Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content. In *MICRO*, 2017. 3.5.1, 3.5.2, 3.5.2, 3.5.2, 3.5.6

[141] W.-S. Khwa, M.-F. Chang, J.-Y. Wu, M.-H. Lee, T.-H. Su, K.-H. Yang, T.-F. Chen, T.-Y. Wang, H.-P. Li, M. Brightsky, S. Kim, H.-L. Lung, and C. Lam. A Resistance-Drift Compensation Scheme to Reduce MLC PCM Raw BER by Over 100x for Storage-Class Memory Applications. In *ISSCC*, 2016. 3.5.7

[142] C. Kim et al. A 21 nm High Performance 64 Gb MLC NAND Flash Memory with 400 MB/s Asynchronous Toggle DDR Interface. *JSSC*, 2012. 2.1.3

[143] Chulbum Kim, Ji-Ho Cho, Woopyo Jeong, Il-han Park, Hyun-Wook Park, Doo-Hyun Kim, Daewoon Kang, Sunghoon Lee, Ji-Sang Lee, Wontae Kim, et al. A 512Gb 3b/Cell 64-

Stacked WL 3D V-NAND Flash Memory. In *ISSCC*, 2017. 3.4.1

[144] J. Kim, M. Sullivan, and M. Erez. Bamboo ECC: Strong, Safe, and Flexible Codes for Reliable Computer Memory. In *HPCA*, 2015. 3.5.6

[145] J. Kim, M. Sullivan, S.-L. Gong, and M. Erez. Frugal ECC: Efficient and Versatile Memory Error Protection Through Fine-Grained Compression. In *SC*, 2015. 3.5.6

[146] J. Kim, M. Patel, H. Hassan, and O. Mutlu. The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices. In *HPCA*, 2018. 3.5.5

[147] K. Kim and J. Lee. A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs. *IEEE Electron Device Lett.*, Aug. 2009. 3.5.2

[148] N. Kim and J.-H. Jang. Nonvolatile Memory Device, Method of Operating Nonvolatile Memory Device and Memory System Including Nonvolatile Memory Device. U.S. Patent 8,203,881, 2012. 3.2.3

[149] Y. Kim. *Architectural Techniques to Enhance DRAM Scaling*. PhD thesis, Carnegie Mellon Univ., 2015. 3.5.3

[150] Y. Kim and O. Mutlu. Memory Systems. In *Computing Handbook*. CRC Press, Boca Raton, FL, USA, 3 edition, 2014. 2.1.1, 2.1.2

[151] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers. In *HPCA*, 2010. 2.1.2

[152] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *ISCA*, 2012. 2.1.1

[153] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Comput. Archit. Lett.*, Jan.–Jun. 2016. 2.1.2

[154] Y. S. Kim, D. J. Lee, C. K. Lee, H. K. Choi, S. S. Kim, J. H. Song, D. H. Song, J.-H. Choi, K.-D. Suh, and C. Chung. New Scaling Limitation of the Floating Gate Cell in NAND Flash Memory. In *IRPS*, 2010. 3.4.2

[155] Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter. Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In *MICRO*, 2010. 2.1.2

[156] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*, 2014. 3.5.3, 3.31, 3.5.3, 3.32, 3.5.3, 3.5.6

[157] Y. Koh. NAND Flash Scaling Beyond 20nm. In *IMW*, 2009. 3.1

[158] R. Koller and R. Rangaswami. I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance. *TOS*, 2010. 4.3, 4.2

[159] Y. Komori, M. Kido, M. Kito, R. Katsumata, Y. Fukuzumi, H. Tanaka, Y. Nagata, M. Ishiduki, H. Aochi, and A. Nitayama. Disturbless Flash Memory Due to High Boost

Efficiency on BiCS Structure and Optimal Memory Film Stack for Ultra High Density Storage Device. In *IEDM*, 2008. 3.4.1

[160] Solomon Kullback and Richard A Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 1951. 5.3.1, 5.3.2, 5.3.3, 5.3.4

[161] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative. In *ISPASS*, 2013. 3.5.7

[162] Mark LaPedus. How to Make 3D NAND. Semiconductor Engineering, 2016. 3.4.2

[163] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA*, 2009. 3.5.7

[164] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Phase Change Memory Architecture and the Quest for Scalability. *Commun. ACM*, Jul. 2010. 3.5.7

[165] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase-Change Technology and the Future of Main Memory. *IEEE Micro*, Feb. 2010. 3.5.7

[166] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt. Improving Memory Bank-Level Parallelism in the Presence of Prefetching. In *MICRO*, 2009. 2.1.1

[167] D. Lee. *Reducing DRAM Energy at Low Cost by Exploiting Heterogeneity*. PhD thesis, Carnegie Mellon Univ., 2016. 3.5.5, 3.5.5

[168] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu. Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture. In *HPCA*, 2013. 2.1.1

[169] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu. Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM. In *PACT*, 2015. 2.1.1

[170] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. *ACM TACO*, Jan. 2016. 2.1.1

[171] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu. Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms. In *SIGMETRICS*, 2017. 3.5.5, 3.5.5, 3.5.6

[172] Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu. Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case. In *HPCA*, 2015. 3.5.2, 3.5.2, 3.5.5, 3.5.5

[173] J.-D. Lee, J.-H. Choi, D. Park, and K. Kim. Degradation of Tunnel Oxide by FN Current Stress and Its Effects on Data Retention Characteristics of 90 nm NAND Flash Memory Cells. In *IRPS*, 2003. 3.1.4

[174] Jae-Duk Lee, Sung-Hoi Hur, and Jung-Dal Choi. Effects of Floating-Gate Interference on NAND Flash Memory Cell Operation. *IEEE Electron Device Letters*, 2002. 3.1.3, 3.1, 6.3.1, 6.3.1

[175] Sang-Yun Lee. Limitations of 3D NAND Scaling. EE Times, 2017. 3.4.2

[176] Sungjin Lee, Taejin Kim, Kyungho Kim, and Jihong Kim. Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling. In *FAST*, 2012. 3.1.6, 3.1.6, 8.1.1

[177] Y. Lee, H. Yoo, I. Yoo, and I.-C. Park. 6.4 Gb/s Multi-Threaded BCH Encoder and Decoder for Multi-Channel SSD Controllers. In *ISSCC*, 2012. 2.1.3, 3.3, 3.3.1, 3.3.1, 6.5.5

[178] Tom Lenny. The Maturity of NVM Express^{TM}. In *Flash Memory Summit*, 2014. 7.3.2

[179] Kenneth Levenberg. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics*, 1944. 7.2.3

[180] Jiangpeng Li, Kai Zhao, Jun Ma, and Tong Zhang. Realizing Unequal Error Correction for NAND Flash Memory at Minimal Read Latency Overhead. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 2014. 5.6

[181] Jiangpeng Li, Kai Zhao, Xuebin Zhang, Jun Ma, Ming Zhao, and Tong Zhang. How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime? In *FAST*, 2015. 2.1.3

[182] Y. Li, C. Hsu, and K. Oowada. Non-Volatile Memory and Method With Improved First Pass Programming. U.S. Patent 8,811,091, 2014. 2.2.4

[183] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu. Utility-Based Hybrid Memory Management. In *CLUSTER*, 2017. 3.5.7

[184] Yan Li. 3 Bit Per Cell NAND Flash Memory on 19nm Technology. *Flash Memory Summit*, 2012. 10.1

[185] Chun-Yi Liu, Yu-Ming Chang, and Yuan-Hao Chang. Read Leveling for Flash Storage Systems. In *SYSTOR*, 2015. 1.2.3

[186] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *ISCA*, 2012. 3.2.3, 3.5.2, 3.29, 3.5.2, 3.5.2, 10.3.1

[187] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *ISCA*, 2013. 3.2.3, 3.5.1, 3.5.2, 3.28, 3.5.2, 3.5.2, 3.30, 3.5.2, 3.5.6

[188] R.-S. Liu, C.-L. Yang, and W. Wu. Optimizing NAND Flash-Based SSDs via Retention Relaxation. In *FAST*, 2012. 4.1, 4.1.1, 4.1.2, 5.1, 5.5.1

[189] R.-S. Liu, C.-L. Yang, C.-H. Li, and G.-Y. Chen. Duracache: A Durable SSD Cache Using MLC NAND Flash. In *DAC*, 2013. 4.1, 4.1.1, 4.1.2

[190] S. Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn. Flikker: Saving DRAM Refresh-Power Through Critical Data Partitioning. In *ASPLOS*, 2011. 3.5.6

[191] W. Liu, J. Rho, and W. Sung. Low-Power High-Throughput BCH Error Correction VLSI Design for Multi-Level Cell NAND Flash Memories. In *SIPS*, 2006. 3.3.1

[192] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu. Exploiting Self-Recovery and Temperature Awareness to Improve 3D NAND Flash Memory Reliability. Technical Report 2018-001, Carnegie Mellon Univ., SAFARI Research Group, 2018. 7.1, 7.2.1, 7.2.3

[193] Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Apoorv

Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu. Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory. In *DSN*, 2014. 2.1.3, 3.5.5, 3.5.6, 10.3.3

[194] Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu. WARM: Improving NAND Flash Memory Lifetime with Write-Hotness Aware Retention Management. In *MSST*, 2015. 1.3.1, 1.4, 2.1.3, 3.1, 3.2.6, 3.2.7, 5.6, 6.2.2, 7.4, 10.2.3

[195] Yixin Luo, Saugata Ghose, Yu Cai, Erich F Haratsch, and Onur Mutlu. Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory. *IEEE JSAC*, 34(9):2294–2311, 2016. 1.1.1, 1.1.1, 1.2.1, 1.3.2, 1.4, 2.1.3, 3.1.1, 3.1.2, 3.1.2, 3.1, 3.2.5, 3.3.1, 3.4.1, 6, 6.1, 2, 3, 6.2.4, 6.3.1, 6.3.1, 6.4.1, 6.5.1, 6.5.5, 3, 7.1.1, 7.1.2, 6, 7.1.3, 7.2.1, 7.3.1, 7.3.2, 7.3.3, 7.4

[196] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. An Accurate and Practical Threshold Voltage Distribution Model for Modern MLC NAND Flash Memory. Technical Report 2016-006, Carnegie Mellon Univ., SAFARI Research Group, 2016. 5.3.4

[197] Yixin Luo, Saugata Ghose, Tianshi Li, Sriram Govindan, Bikash Sharma, Bryan Kelly, Amirali Boroumand, and Onur Mutlu. Using ECC DRAM to Adaptively Increase Memory Capacity. arXiv:1706.08870 [cs.AR], 2017. 3.5.6, 10.3.3

[198] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation. In *under submission to SIGMETRICS*, 2018. 1.3.3, 1.4

[199] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature-Awareness. In *HPCA*, 2018. 1.3, 1.3.4, 1.4, 10.2.3

[200] S. Luryi, A. Kastalsky, A. C. Gossard, and R. H. Hendel. Charge Injection Transistor Based on Real-Space Hot-Electron Transfer. *IEEE Trans. Electron Devices*, Jun. 1984. 4

[201] Dongzhe Ma, Jianhua Feng, and Guoliang Li. LazyFTL: A Page-Level Flash Translation Layer Optimized for NAND Flash Memory. In *SIGMOD*, 2011. 1.2.3

[202] Dongzhe Ma, Jianhua Feng, and Guoliang Li. A survey of address translation technologies for flash memories. *CSUR*, 2014. 1.2.3

[203] D. J. C. MacKay and R. M. Neal. Near Shannon Limit Performance of Low Density Parity Check Codes. *IET Electron. Lett.*, Mar. 1997. 1.1.2, 2.1.3, 3.3, 3.3.1, 3.3.1, 3.3.2

[204] David JC MacKay and Radford M Neal. Near Shannon Limit Performance of Low Density Parity Check Codes. *Electronics letters*, 1996. 1.1.2, 2.1.3, 3.3, 3.3.1, 3.3.1, 3.3.2

[205] A. Maislos. A New Era in Embedded Flash Memory. In *Flash Memory Summit*, 2011. 3.1, 3.2

[206] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens. Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM). *IBM J. Research Develop.*, Mar. 2002. 3.5.2, 3.5.6

[207] A. Marelli and R. Micheloni. BCH and LDPC Error Correction Codes for NAND Flash

Memories. In *3D Flash Memories*. Springer, Dordrecht, Netherlands, 2016. 3.3.1

[208] Todd Marquart. Practical Approach to Determining SSD Reliability. In *Flash Memory Summit*, 2015. 7.2.2

[209] J. L. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Trans. Inf. Theory*, Jan. 1969. 3.3.1

[210] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu. A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory. In *WEED*, 2013. 3.5.7, 10.3.3

[211] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *DSN*, 2015. 3.5.4, 3.33, 3.5.4, 3.34, 3.5.6, 10.2.1

[212] Justin Meza, Jichuan Chang, HanBin Yoon, Onur Mutlu, and Parthasarathy Ranganathan. Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management. *IEEE Comput. Archit. Lett.*, Feb. 2012. 3.5.7

[213] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. A Large-Scale Study of Flash Memory Failures in The Field. In *SIGMETRICS*, 2015. 2.1.3, 2.1.3, 3.1.7, 1, 3.5, 3.1.7, 3.6, 3.7, 3.1.7, 3.8, 3.1.7, 3.2.3, 7.3.2, 10.2.2

[214] R. Micheloni, editor. *3D Flash Memories*. Springer Netherlands, Dordrecht, Netherlands, 2016. 3.4

[215] R Micheloni, R Ravasio, A Marelli, E Alice, V Altieri, A Bovino, L Crippa, E Di Martino, L D'Onofrio, A Gambardella, et al. A 4Gb 2b/Cell NAND Flash Memory with Embedded 5b BCH ECC for 36MB/s System Read Throughput. In *ISSCC*, 2006. 3.3.1

[216] R. Micheloni, S. Aritome, and L. Crippa. Array Architectures for 3-D NAND Flash Memories. *Proc. IEEE*, Sep. 2017. 3.4

[217] Micron Technology, Inc. *Memory Management in NAND Flash Arrays*, 2005. 2.1.3

[218] Micron Technology, Inc. *Bad Block Management in NAND Flash Memory*, 2011. 2.1.3

[219] N. Mielke, T. Marquart, N.Wu, J.Kessenich, H. Belgal, E. Schares, and F. Triverdi. Bit Error Rate in NAND Flash Memories. In *IRPS*, 2008. 1.1.1, 1.2.1, 1.2.2, 2.2, 2.2.4, 3.1.4, 3.1.5, 3.1, 3.2.4, 6, 6.2.2, 6.3.1, 6.3.2, 7.1.1, 7.1.2, 7.1.3, 7.2.1, 7.4, 10.2.2

[220] Neal Mielke, Hanmant P Belgal, Albert Fazio, Qingru Meng, and Nick Righos. Recovery Effects in The Distributed Cycling of Flash Memories. In *IRPS*, 2006. 3.1.6, 7.1.2, 5, 7.1.3, 7.1.4, 7.2.2, 7.2.3, 7.4

[221] Kyoji Mizoguchi, Tomonori Takahashi, Seiichi Aritome, and Ken Takeuchi. Data-Retention Characteristics Comparison of 2D and 3D TLC NAND Flash Memories. In *IMW*, 2017. 3.1.6, 3.4.2, 7.1.1, 7.2.2, 7.4

[222] V. Mohan, S. Sankar, and S. Gurumurthi. reFresh SSDs: Enabling High Endurance, Low Cost Flash in Datacenters. Technical Report CS-2012-05, Univ. of Virginia, 2012. 3.1, 3.2.3, 3.2.3, 4.1, 4.1.1, 4.1.2, 5.7

[223] Vidyabhushan Mohan. *Modeling The Physical Characteristics of NAND Flash Memory*.

PhD thesis, University of Virginia, 2010. 2.2.2, 2.2.4

[224] Vidyabhushan Mohan, Taniya Siddiqua, Sudhanva Gurumurthi, and Mircea R Stan. How I Learned to Stop Worrying and Love Flash Endurance. In *HotStorage*, 2010. 1.3.4, 3.1.6, 3.1.6, 7.1.2, 7.1.2, 7.1.4

[225] T. Moscibroda and O. Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems. In *USENIX Security*, 2007. 2.1.1

[226] T. Moscibroda and O. Mutlu. Distributed Order Scheduling and Its Application to Multi-Core DRAM Controllers. In *PODC*, 2008. 2.1.2

[227] Ravi Motwani. Estimation of Flash Memory Level Distributions Using Interpolation Techniques for Optimizing the Read Reference. In *GLOBECOM*, 2015. 5.6

[228] Ravi Motwani and Chong Ong. Design of LDPC Coding Schemes for Exploitation of Bit Error Rate Diversity Across Dies in NAND Flash Memory. In *ICNC*, 2013. 5.6

[229] Ravi Motwani and Chong Ong. Soft Decision Decoding of RAID Stripe for Higher Endurance of Flash Memory Based Solid State Drives. In *ICNC*, 2015. 5.6

[230] J. Mukundan, H. Hunter, K.-H. Kim, J. Stuecheli, and J. F. Martínez. Understanding and Mitigating Refresh Overheads in High-Density DDR4 DRAM Systems. In *ISCA*, 2013. 3.5.2

[231] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda. Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning. In *MICRO*, 2011. 2.1.2

[232] O. Mutlu. Memory Scaling: A Systems Architecture Perspective. In *IMW*, 2013. 3.5.6

[233] O. Mutlu. The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser. In *DATE*, 2017. 3.5.2, 3.5.3, 3.5.3, 3.5.3, 3.5.6

[234] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO*, 2007. 2.1.1, 2.1.2

[235] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems. In *ISCA*, 2008. 2.1.1, 2.1.2

[236] O. Mutlu and L. Subramanian. Research Problems and Opportunities in Memory Systems. *SUPERFRI*, 2015. 3.5.6

[237] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz. STT-RAM Scaling and Retention Failure. *Intel Technol. J.*, May 2013. 3.5.7

[238] P. J. Nair, V. Sridharan, and M. K. Qureshi. XED: Exposing On-Die Error Detection Information for Strong Memory Reliability. In *ISCA*, 2016. 3.5.6

[239] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-Loading: Practical Power Management for Enterprise Storage. *TOS*, 2008. 4.3, 4.2, 7.3.3

[240] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. SSD Failures in Datacenters: What, When and Why? In *SIGMETRICS*, 2016. 10.2.1

[241] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand

Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. Ssd failures in datacenters: What? when? and why? In *SYSTOR*, 2016. 3.1.7, 3.1.7, 3.1.7, 10.2.1, 10.2.2

[242] K. Naruke, S. Taguchi, and M. Wada. Stress Induced Leakage Current Limiting to Scale Down EEPROM Tunnel Oxide Thickness. In *IEDM*, 1988. 2.2.4

[243] National Inst. of Standards and Technology. *Specification for the Advanced Encryption Standard (AES)*, 2001. 2.1.3

[244] Hagop Nazarian and Sylvain Dubois. The drive for SSDs: Whats holding back NAND flash? `https://www.edn.com/Home/PrintView?contentItemId=4424905`, 2013. Online; accessed October 2017. 1.1

[245] John A Nelder and Roger Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 1965. 5.3.1, 5.3.2, 5.3.3, 5.4.2

[246] William D Norcott and Don Capps. IOzone Filesystem Benchmark. `http://www.iozone.org`, 2003. 4.3, 4.2

[247] NVM Express, Inc. *NVM Express Specification, Revision 1.3*, 2017. 2.1.3

[248] Openmoko. NAND Bad Blocks. `http://wiki.openmoko.org/wiki/NAND_bad_blocks`, 2012. 2.1.3

[249] Patrick ONeil, Edward Cheng, Dieter Gawlick, and Elizabeth ONeil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996. 10.3.1

[250] Y. Pan, G. Dong, Q. Wu, and T. Zhang. Quasi-Nonvolatile SSD: Trading Flash Memory Nonvolatility to Improve Storage System Performance for Enterprise Applications. In *HPCA*, 2012. 3.1, 3.2.3, 4.1, 4.1.1, 4.1.2, 5.7, 6.5.3, 7.3.1, 7.4

[251] Yangyang Pan, Guiqiang Dong, and Tong Zhang. Exploiting Memory Device Wear-Out Dynamics to Improve NAND Flash Memory System Performance. In *FAST*, 2011. 5.6

[252] Nikolaos Papandreou, Thomas Parnell, Haralampos Pozidis, Thomas Mittelholzer, Evangelos Eleftheriou, Charles Camp, Thomas Griffin, Gary Tressler, and Andrew Walls. Using Adaptive Read Voltage Thresholds to Enhance The Reliability of MLC NAND Flash Memory Systems. In *GLSVLSI*, 2014. 1.1.1, 1.2.1, 1.2.2, 3.1.5, 3.2.5, 3.2.5, 3.2.5, 5.5.2, 5.6, 2, 6.3.3, 6.4.1, 6.5.1, 6.5.5, 7.3.3, 7.3.3, 7.4

[253] Nikolaos Papandreou, Thomas Parnell, Haralampos Pozidis, Thomas Mittelholzer, Evangelos Eleftheriou, Charles Camp, Thomas Griffin, Gary Tressler, and Andrew Walls. Enhancing the Reliability of MLC NAND Flash Memory Systems by Read Channel Optimization. *TODAES*, 2015. 5.6

[254] Dongchul Park, Biplob Debnath, and David Du. CFTL: A Convertible Flash Translation Layer Adaptive to Data Access Patterns. In *SIGMETRICS*, 2010. 1.2.3

[255] Jisung Park, Jaeyong Jeong, Sungjin Lee, Youngsun Song, and Jihong Kim. Improving Performance and Lifetime of NAND Storage Systems Using Relaxed Program Sequence. In *DAC*, 2016. 1.2.3, 2.2.4, 3.2.1, 6.5.2

[256] Ki-Tae Park, Myounggon Kang, Doogon Kim, Soon-Wook Hwang, Byung Yong Choi,

Yeong-Taek Lee, Changhyun Kim, and Kinam Kim. A Zeroing Cell-To-Cell Interference Page Architecture With Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories. *JSSC*, 2008. 2.2.4, 3.4.2

[257] Ki-Tae Park, Sangwan Nam, Dae-Han Kim, Pansuk Kwak, Doosub Lee, Yoon-Hee Choi, Myung-Hoon Choi, Dong-Hun Kwak, Doo-Hyun Kim, Minsu Kim, Hyun Wook Park, Sang-Won Shim, Kyung-Min Kang, Sang-Won Park, Kangbin Lee, Hyun-Jun Yoon, Kui-han Ko, Dong-Kyo Shim, Yang-Lo Ahn, Jinho Ryu, Donghyun Kim, Kyunghwa Yun, Joonsoo Kwon, Seunghoon Shin, Dae-Seok Byeon, Kihwan Choi, Jin-Man Han, Kyehyun Kyung, Jeong-Hyuk Choi, and Kinam Kim. Three-Dimensional 128 Gb MLC Vertical NAND Flash Memory With 24-WL Stacked Layers and 50 MB/s High-Speed Programming. *J. Solid-State Circuits*, Jan. 2015. 3.4, 3.4.1, 3.4.1, 3.4.2, 6, 6.1, 6.3.1, 7.1.1, 7.4

[258] T. Parnell. NAND Flash Basics & Error Characteristics: Why Do We Need Smart Controllers? In *Flash Memory Summit*, 2016. 3.4.2, 3.4.3

[259] T. Parnell and R. Pletka. NAND Flash Basics & Error Characteristics – Why Do We Need Smart Controllers? In *Flash Memory Summit*, 2017. 3.4.2, 3.4.3

[260] Thomas Parnell, Nikolaos Papandreou, Thomas Mittelholzer, and Haralampos Pozidis. Modelling of the Threshold Voltage Distributions of Sub-20nm NAND Flash Memory. In *GLOBECOM*, 2014. 1.1.1, 1.1.1, 1.2.1, 3.1, 3.1.2, 3.2, 3.1, 3.4.1, 5.3, 5.3.1, 5.3.2, 5.3.2, 5.6, 6, 6.1, 3, 6.2.4, 6.3.1, 6.3.1, 6.5.5, 3, 7.1.1, 7.3.1, 7.3.3, 7.4

[261] M. Patel, J. Kim, and O. Mutlu. The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions. In *ISCA*, 2017. 3.5.1, 3.5.2, 3.5.2, 3.5.2, 3.5.2, 3.5.6

[262] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD*, 1988. 2.1.3, 2.1.3, 2.1.3, 6.5.2

[263] P. Pavan, R. Bez, P. Olivo, and E. Zanoni. Flash Memory Cells—An Overview. *Proc. IEEE*, Aug 1997. 2.2.4

[264] PCI-SIG. *PCI Express Base Specification Revision 3.1a*, 2015. 2.1.3

[265] J. Pearl. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In *AAAI*, 1982. 3.3.1, 3.3.1

[266] W. W. Peterson and D. T. Brown. Cyclic Codes for Error Detection. *Proc. IRE*, Jan. 1961. 2.1.3

[267] Sujay Phadke and S. Narayanasamy. MLP Aware Heterogeneous Memory System. In *DATE*, 2011. 3.5.7

[268] A. Pirovano, A. L. Lacaita, F. Pellizzer, S. A. Kostylev, A. Benvenuti, and R. Bez. Low-Field Amorphous State Resistance and Threshold Voltage Drift in Chalcogenide Materials. *IEEE Trans. Electron Devices*, Jun. 2004. 3.5.7

[269] Pravin Prabhu, Ameen Akel, Laura M Grupp, S Yu Wing-Kei, G Edward Suh, Edwin Kan, and Steven Swanson. Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations. In *TRUST*, 2011. 1.1.1, 1.2.1, 6.1

[270] Antonios Prodromakis, Stelios Korkotsides, and Theodore Antonakopoulos. MLC NAND

Flash Memory: Aging Effect and Chip/Channel Emulation. *Microprocessors and Microsystems*, 2015. 5.6

[271] Zhiwei Qin, Yi Wang, Duo Liu, Zili Shao, and Yong Guan. MNFTL: An Efficient Flash Translation Layer for MLC NAND Flash Memory Storage Systems. In *DAC*, 2011. 2.1.3

[272] M. Qureshi, D. H. Kim, S. Khan, P. Nair, and O. Mutlu. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *DSN*, 2015. 3.5.1, 3.5.2, 3.5.2, 3.5.2, 3.5.2, 3.5.6

[273] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *ISCA*, 2009. 3.5.7

[274] Moinuddin K. Qureshi and Gabe H. Loh. Fundamental Latency Trade-Off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *MICRO*, 2012. 3.5.7

[275] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. Page Placement in Hybrid Memory Systems. In *ICS*, 2011. 3.5.7

[276] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Guiffrida, and H. Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In *USENIX Security*, 2016. 3.5.3

[277] William J Reed. The Normal-Laplace Distribution and Its Relatives. In *Advances in Distribution Theory, Order Statistics, and Inference*. Springer, 2006. 5.3.2

[278] P. J. Restle, J. W. Park, and B. F. Lloyd. DRAM Variable Retention Time. In *IEDM*, 1992. 3.5.2

[279] D. Rollins. *A Comparison of Client and Enterprise SSD Data Path Protection*. Micron Technology, Inc., 2011. 2.1.3, 2.1.3, 2.1.3, 2.1.3

[280] William Ryan and Shu Lin. *Channel Codes: Classical and Modern*. Cambridge Univ. Press, Cambridge, UK, 2009. 3.3.1, 3.3.1

[281] SAFARI Research Group. 3D NAND Flash Memory Characterization Data. `https://github.com/CMU-SAFARI/HeatWatch`. 7.1

[282] Samsung Electronics Co., Ltd. Samsung V-NAND Technology, 2014. `http://www.samsung.com/us/business/oem-solutions/pdfs/V-NAND_technology_WP.pdf`. 3.4.1, 6.2.2

[283] Samsung Electronics Co., Ltd. *Samsung SSD 960 PRO M.2 Data Sheet Rev. 1.1*, 2017. 2.1.3

[284] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM Errors in the Wild: A Large-Scale Field Study. In *SIGMETRICS*, 2009. 3.5.4

[285] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash Reliability in Production: The Expected and The Unexpected. In *FAST*, 2016. 3.1.7, 3.1.7, 10.2.1, 10.2.2

[286] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and Statistical Modeling with Python. In *SciPy*, 2010. 7.2.1

[287] M. Seaborn and T. Dullien. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. In *BlackHat*, 2015. 3.5.3

[288] M. Seaborn and T. Dullien. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. Google Project Zero Blog, 2015. 3.5.3

[289] Seagate Technology LLC. *Enterprise Performance 15K HDD Data Sheet*, 2016. 2.1.3

[290] Serial ATA International Organization. *Serial ATA Revision 3.3 Specification*, 2016. 2.1.3, 7.1.1

[291] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al. Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization. In *MICRO*, pages 185–197. IEEE, 2013. 10.3.3

[292] C. E. Shannon. A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, Jul. 1948. 3.3.1

[293] C. E. Shannon. A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, Oct. 1948. 3.3.1

[294] Claude E. Shannon and Warren Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, USA, 1949. ISBN 0252725484. 1.1.2

[295] Shawn Knight. Samsung to fix slow 840 EVO SSDs with "periodic refresh" feature. `https://www.techspot.com/news/60362-samsung-fix-slow-840-evo-ssds-periodic-refresh.html`, 2015. TechSpot. 4

[296] H. Shim, S.-S. Lee, B. Kim, N. Lee, D. Kim, H. Kim, B. Ahn, Y. Hwang, H. Lee, J. Kim, Y. Lee, H. Lee, J. Lee, S. Chang, J. Yang, S. Park, S. Aritome, S. Lee, K.-O. Ahn, G. Bae, and Y. Yang. Highly Reliable 26nm 64Gb MLC E2NAND (Embedded-ECC & Enhanced-Efficiency) Flash Memory With MSP (Memory Signal Processing) Controller. In *VLSIT*, 2011. 3.2.4

[297] L. Shu and D. J. Costello. *Error Control Coding*. Prentice-Hall, Englewood Cliffs, NJ, USA, 2 edition, 2004. 2.1.3, 3.3, 3.3.1, 3.3.1, 3.3.1, 3.3.1, 3.3.2, 10.2.3

[298] S. Sills, S. Yasuda, J. Strand, A. Calderoni, K. Aratani, A. Johnson, and N. Ramaswamy. A Copper ReRAM Cell for Storage Class Memory Applications. In *VLSIT*, 2014. 3.5.7

[299] S. Sills, S. Yasuda, A. Calderoni, C. Cardon, J. Strand, K. Aratani, and N. Ramaswamy. Challenges for High-Density 16Gb ReRAM with 27nm Technology. In *VLSIC*, 2015. 3.5.7

[300] MR Speigel. Theory and Problems of Probability and Statistics. *McGraw-Hill*, 1992. 5.3, 5.3.3

[301] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults. In *SC*, 2013. 3.5.4

[302] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B. Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. Memory Errors in Modern Systems: The Good, The Bad, and the Ugly. In *ASPLOS*, 2015. 3.5.4

[303] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The Missing Memristor Found. *Nature*, May 2008. 3.5.7

[304] J. Stuecheli, D. Kaseridis, H. C. Hunter, and L. K. John. Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory. In *MICRO*, 2010. 3.5.2

[305] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu. The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost. In *ICCD*, 2014. 2.1.2

[306] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu. BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling. *IEEE Trans. Parallel Distrib. Syst.*, Oct. 2016. 2.1.2

[307] Lavanya Subramanian, Vivek Seshadri, Yoongu Kim, Ben Jaiyen, and Onur Mutlu. MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems. In *HPCA*, 2013. 2.1.2

[308] Kang-Deog Suh, Byung-Hoom Suh, Young-Ho Lim, Jin-Ki Kim, Young-Joon Choi, Yong-Nam Koh, Sung-Soo Lee, Suk-Chon Suk-Chon, Byung-Soon Choi, Jin-Sun Yum, et al. A 3.3 V 32 Mb NAND Flash Memory With Incremental Step Pulse Programming Scheme. *Solid-State Circuits, IEEE Journal of*, 1995. 2.2.4

[309] Haleh Tabrizi, Borja Peleato, Rajiv Agarwal, and Jeffrey Ferreira. Improving NAND Flash Read Performance Through Learning. In *ICC*, 2015. 5.5.2, 5.6

[310] K. Takeuchi, S. Satoh, T. Tanaka, K. Imamiya, and K. Sakui. A Negative Vth Cell Architecture for Highly Scalable, Excellently Noise-Immune, and Highly Reliable NAND Flash Memories. *IEEE Journal of Solid-State Circuits*, 1999. 3.1.5

[311] H. Tanaka, M. Kido, K. Yahashi, M. Oomura, R. Katsumata, M. Kito, Y. Fukuzumi, M. Sato, Y. Nagata, Y. Matsuoka, Y. Iwata, H. Aochi, and A. Nitayama. Bit Cost Scalable Technology with Punch and Plug Process for Ultra High Density Flash Memory. In *VLSIT*, 2007. 3.4.1

[312] S. Tanakamaru, C. Hung, A. Esumi, M. Ito, K. Li, and K. Takeuchi. 95%-Lower-BER 43%-Lower-Power Intelligent Solid-State Drive (SSD) With Asymmetric Coding and Stripe Pattern Elimination Algorithm. In *ISSCC*, 2011. 3.1.4, 3.2.3

[313] R. Tanner. A Recursive Approach to Low Complexity Codes. *IEEE Trans. Inf. Theory*, Sep. 1981. 3.3.1

[314] Toru Tanzawa. Method and Apparatus for Generating Temperature-Compensated Read and Verify Operations in Flash Memories, 2007. US Patent 7,277,355. 10.1

[315] Veeresh Taranalli, Hironori Uchikawa, and Paul H Siegel. Channel Models For Multi-Level Cell Flash Memories Based on Empirical Error Analysis. arXiv:1602.07743 [cs.AR], 2016. 5.6

[316] Arash Tavakkol, Juan Gómez Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices. In *FAST*, 2018. 2.1.3

[317] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan G. Luna, and Onur Mutlu. FLIN:

Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives. In *ISCA*, 2018. 2.1.3, 10.3.3

[318] Techman Electronics Co. Techman XC100 NVMe SSD. White Paper v1.0, 2016. 2.1.3

[319] Toshiba Corp. 3D Flash Memory: Scalable, High Density Storage for Large Capacity Applications. `http://www.toshiba.com/taec/adinfo/technologymoves/3d-flash.jsp`, 2017. 3.4.1

[320] A. N. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. P. Jouppi. LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems. In *ISCA*, 2012. 3.5.6

[321] Univ. of Massachusetts. Storage: UMass Trace Repository. `http://tinyurl.com/k6golon`, 2002. 4.3, 4.2

[322] V. van der Veen, Y. Fratanonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Guiffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *CCS*, 2016. 3.5.3

[323] Ned Varnica. LDPC Decoding: VLSI Architectures and Implementations — Module 1: LDPC Decoding. In *Flash Memory Summit*, 2013. 3.3.1

[324] R. K. Venkatesan, S. Herr, and E. Rotenberg. Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM. In *HPCA*, 2006. 3.5.2, 3.5.2

[325] C. Wang and W.-F. Wong. Extending the Lifetime of NAND Flash Memory by Salvaging Bad Blocks. In *DATE*, 2012. 3.2.6

[326] Jiadong Wang, Kasra Vakilinia, Tsung-Yi Chen, Thomas Courtade, Guiqiang Dong, Tong Zhang, Hari Shankar, and Richard Wesel. Enhanced Precision Through Multiple Reads for LDPC Decoding in Flash Memories. *Selected Areas in Communications, IEEE Journal on*, 2014. 3.2.5, 3.3.1, 3.3.1, 3.3.1, 3.3.2, 5.5.4, 5.6

[327] W. Wang, T. Xie, and D. Zhou. Understanding the Impact of Threshold Voltage on MLC Flash Memory Performance and Reliability. In *ICS*, 2014. 2.2.4

[328] Yi Wang, Lisha Dong, and Rui Mao. P-Alloc: Process-Variation Tolerant Reliability Management for 3D Charge-Trapping Flash Memory. *TECS*, 16(5s):142, 2017. 6.2.1

[329] J. Werner. A Look Under the Hood at Some Unique SSD Features. In *Flash Memory Summit*, 2010. 3.2.3

[330] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu. Reducing Cache Power With Low-Cost, Multi-Bit Error-Correcting Codes. In *ISCA*, 2010. 3.5.6

[331] M. Willett. Encrypted SSDs: Self-Encryption Versus Software Solutions. In *Flash Memory Summit*, 2015. 2.1.3

[332] E. H. Wilson, M. Jung, and M. T. Kandemir. Zombie NAND: Resurrecting Dead NAND Flash for Improved SSD Longevity. In *MASCOTS*, 2014. 3.1, 3.2.7, 7.1.3

[333] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi,

and K. E. Goodson. Phase Change Memory. *Proc. IEEE*, Dec. 2010. 3.5.7

[334] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai. Metal-Oxide RRAM. *Proc. IEEE*, Jun. 2012. 3.5.7

[335] Guanying Wu and Xubin He. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In *FAST*, 2012. 2.1.3

[336] Guanying Wu, Xubin He, Ningde Xie, and Tong Zhang. DiffECC: Improving SSD Read Performance Using Differentiated Error Correction Coding Schemes. In *MASCOTS*, 2010. 3.1, 3.2.7, 5.1, 5.5.5

[337] Qi Wu, Guiqiang Dong, and Tong Zhang. Exploiting Heat-Accelerated Flash Memory Wear-Out Recovery to Enable Self-Healing SSDs. In *HotStorage*, 2011. 1.3.4, 3.1.6, 3.1.6, 7.1.2, 7.1.2, 7.1.4

[338] Qi Wu, Guiqiang Dong, and Tong Zhang. A First Study on Self-Healing Solid-State Drives. In *IMW*, 2011. 3.1.6, 7.1.2, 7.1.4

[339] Y. Wu and E. T. Cohen. Optimization of Read Thresholds for Non-Volatile Memory. U.S. Patent 9,595,320, 2015. 3.2.5

[340] Y. Wu, Z. Chen, Y. Cai, and E. F. Haratsch. Method of Erase State Handling in Flash Channel Tracking. U.S. Patent 9,213,599, 2015. 3.3.1

[341] Y. Wu, Y. Cai, and E. F. Haratsch. Systems and Methods for Soft Data Utilization in a Solid State Memory System. U.S. Patent 9,201,729, 2017. 3.3.1

[342] Y. Wu, Y. Cai, and E. F. Haratsch. Fixed Point Conversion of LLR Values Based on Correlation. U.S. Patent 9,582,361, 2017. 3.3.1

[343] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *USENIX Security*, 2016. 3.5.3

[344] M. Xu, M. Li, and C. Tan. Extended Arrhenius Law of Time-to-Breakdown of Ultrathin Gate Oxides. *Appl. Phys. Lett.*, 2003. 3.2.3

[345] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Manu Awasthi, Tameesh Suri, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. Performance Characterization of Hyperscale Applications on NVMe SSDs. In *SIGMETRICS*, 2015. 2.1.3

[346] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. Performance Analysis of NVMe SSDs and Their Implication on Real World Databases. In *SYSTOR*, 2015. 2.1.3

[347] R.-I. Yamada, Y. Mori, Y. Okuyama, J. Yugami, T. Nishimoto, and H. Kume. Analysis of Detrap Current Due to Oxide Traps to Improve Flash Memory Retention. In *IRPS*, 2000. 3.1.4

[348] D. S. Yaney, C. Y. Lu, R. A. Kohler, M. J. Kelly, and J. T. Nelson. A Meta-Stable Leakage Phenomenon in DRAM Charge Storage — Variable Hold Time. In *IEDM*, 1987. 3.5.2

[349] J. Yang. High-Efficiency SSD for Reliable Data Storage Systems. In *Flash Memory Summit*, 2011. 3.1, 3.2.4

[350] Ming-Chang Yang, Yu-Ming Chang, Che-Wei Tsao, Po-Chun Huang, Yuan-Hao Chang,

and Tei-Wei Kuo. Garbage collection and wear leveling for flash memory: past and future. In *SMARTCOMP*, 2014. 2.1.3, 2.1.3, 2.1.4

[351] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu. Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories. *ACM TACO*, Dec. 2014. 3.5.7

[352] HanBin Yoon, J. Meza, R. Ausavarungnirun, R.A. Harding, and O. Mutlu. Row Buffer Locality Aware Caching Policies for Hybrid Memories. In *ICCD*, 2012. 3.5.7

[353] J. H. Yoon. 3D NAND Technology: Implications to Enterprise Storage Applications. In *Flash Memory Summit*, 2015. 3.1.6, 3.4, 3.4.1, 3.4.2

[354] J. H. Yoon and G. A. Tressler. Advanced Flash Technology Status, Scaling Trends & Implications to Enterprise SSD Technology Enablement. In *Flash Memory Summit*, 2012. 3.2

[355] J. H. Yoon and G. A. Tressler. Advanced Flash Technology Status, Scaling Trends & Implications to Enterprise SSD Technology Enablement. In *Flash Memory Summit*, 2012. 3.1, 3.1.4

[356] J. H. Yoon, R. Godse, G. Tressler, and H. Hunter. 3D-NAND Scaling and 3D-SCM — Implications to Enterprise Storage. In *Flash Memory Summit*, 2017. 3.4.2, 3.4.3

[357] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas. Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation. In *MICRO*, 2017. 3.5.7

[358] Liu Yuan, Huaida Liu, Pingui Jia, and Yiping Yang. An Adaptive ECC Scheme for Dynamic Protection of NAND Flash Memories. In *ICASSP*, 2015. 5.5.5

[359] Enze Zhang, Weiyi Wang, Cheng Zhang, Yibo Jin, Guodong Zhu, Qingqing Sun, David Wei Zhang, Peng Zhou, and Faxian Xiu. Tunable charge-trap memory based on few-layer mos2. *ACS NANO*, 2014. 6.2.2

[360] Wangyuan Zhang and Tao Li. Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures. In *PACT*, 2009. 3.5.7

[361] Z. Zhang, W. Xiao, N. Park, and D. J. Lilja. Memory Module-Level Testing and Error Behaviors for Phase Change Memory. In *ICCD*, 2012. 3.5.7

[362] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Tong Zhang, Xiaodong Zhang, and Nanning Zheng. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *FAST*, 2013. 2.1.3, 3.3.1, 3.3.1, 3.3.1, 3.3.1, 3.3.2, 5.5.4, 5.6

[363] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *ISCA*, 2009. 3.5.7

[364] A. Zuck, S. Toledo, D. Sotnikov, and D. Harnik. Compression and SSDs: Where and How? In *INFLOW*, 2014. 2.1.3

[365] Aviad Zuck, Yue Li, Jehoshua Bruck, Donald E Porter, and Dan Tsafrir. Stash in a Flash. 2018. 3.1.1