공학박사학위논문

3차원 낸드 플래시 저장 장치의 보안성과 실시간성 보장을 위한 계층 교차 최적화 기법

Cross-layer Optimization Techniques for Secure Real-time 3D NAND Flash-Based Storage Systems

2020년 8월

서울대학교 대학원
컴퓨터공학부
김명석

3차원 낸드 플래시 저장 장치의 보안성과 실시간성 보장을 위한 계층 교차 최적화 기법

Cross-layer Optimization Techniques for Secure Real-time 3D NAND Flash-Based Storage Systems

지도교수 김 지 홍

이 논문을 공학박사 학위논문으로 제출함
2020년 6월

서울대학교 대학원
컴퓨터공학부
김명석

김명석의 공학박사 학위논문을 인준함
2020년 7월

Abstract

In recent years, NAND flash-based storage systems have explosively grown in popularity owing to their unique advantages and the successful reduction in the cost-per-bit value of NAND flash memory. As NAND flash memory is widely adopted in various emerging data-driven applications, modern storage systems are required to satisfy new requirements such as high data security or real-time processing. Therefore, it becomes crucial to develop new optimization techniques that can properly address new challenges.

In this dissertation, we propose various optimization techniques that enable flash storage systems to support high data security and real-time processing without compromising the performance and the reliability of NAND flash memory. Our techniques are motivated by key findings that are based on comprehensive NAND characterization studies. We found that the read service time of flash storage systems can be significantly fluctuated and delayed by two major factors; One is iterative read operations to search optimal read reference voltages, called *read retries* and the other is a command conflict between read tasks and preceding I/O requests. Moreover, we also founded that the existing data sanitization techniques cannot avoid large performance or reliability penalties because they physically destroy the stored data. Therefore, to protect sensitive data reliably, a new approach is needed that provides equivalent security guarantees without physically changing the data. Based on our findings, we proposed optimization techniques to satisfy the new requirements of emerging storage markets.

We first introduce a practical read retry mitigation technique based on a new NAND aging marker that accurately represents the wear status of NAND blocks. Since the read retry operation is closely related to the wear of NAND flash memory, knowing the exact aging characteristics of individual NAND block is a prerequisite to minimize the read retries. From our evaluation study, we first show that the existing P/E cycle-based aging marker (`PeWear`) is inadequate to estimate the actual aging status of NAND blocks, thus losing opportunities for further optimizations. To overcome the limitations of `PeWear`, we propose a new NAND aging marker, `RealWear`, based on extensive characterization studies using real 3D TLC flash chips. By considering multiple variables that can affect the NAND cell wear, `RealWear` can accurately indicate the actual wear status of NAND blocks during run time. Based on the value of `RealWear`, we construct optimal read reference voltage tables and implement a `RealWear`-aware FTL, `rFTL`, that employs `RealWear`-specific module. Since an optimal read reference voltages can be directly provided without the time-consuming iterative searching, `rFTL` can significantly mitigate read latency fluctuations, guaranteeing that the read latency can be bounded with at most 2 read retry operations even in the worst-case scenario.

Secondly, we propose a new suspend/resume command for 3D NAND flash memory, called priority-aware Suspend/Resume (`pSR`). Unlike the existing preemption techniques, `pSR` limits the number of suspend/resume operations by permitting the preemption command only at predefined *safe* points, thus efficiently avoiding large reliability penalties due to excessive preemptions. Based on the accurate NAND error model, pSR partially allows read tasks to immediately suspend the ongoing flash operations with carefully managing the flash reliability. Since pSR efficiently eliminating the conflicts between flash operations, a high-priority read request can be served in a timely fashion without an unpredictable long delay.

Finally, we propose evanesco, a new data sanitization technique specifically designed for high-density 3D NAND flash memory. Unlike existing techniques that physically destroy stored data, evanesco provides data sanitization by blocking access to stored data. By exploiting existing spare flash cells in the flash memory chip, evanesco efficiently supports two new flash *lock* commands (pLock and bLock) that disable access to deleted data at both page and block granularities. Since the locked page (or block) can be unlocked only after its data is erased, evanesco provides a strong security guarantee even against an advanced threat model. To evaluate our technique, we build SecureSSD, an evanesco-enabled emulated flash storage system. Our experimental results show that SecureSSD can effectively support data sanitization with a small performance overhead and no reliability degradation.

In order to evaluate the effectiveness of the proposed techniques, we conducted a set of experiments with various benchmark tools and I/O traces collected from real-world applications. The experimental results showed that our proposed techniques allow a flash storage system to be managed in a timely fashion and to reliably protect sensitive data from a malicious attack, thus enabling NAND flash-based storage systems to better meet the various requirements of modern computing systems. Furthermore, our techniques can also improve both the performance and lifetime by fully leveraging the exact status of the individual NAND block.

Keywords: 3D NAND Flash Memory, Flash Translation Layer, NAND Flash-Based Storage Systems, Embedded Systems, Security, Real-time, Performance Optimization, Lifetime Optimization
Student Number: 2015-31049

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Over the past decade, NAND flash-based storage systems have been the key enablers of the modern digital revolution thanks to their superior properties compared to magnetic disk drives (e.g., HDDs) such as low power consumption, high shock resistance, fast random access, and small form factor. Especially, with the continuous effort of NAND manufacturers, the cost-per-bit value of NAND flash memory has been dramatically reduced due to various innovative technologies such as 3D process integration [1, 2] and multi-level cell technology [3, 4]. Therefore, NAND flash-based storage systems are expected to more widely adopted in various storage markets, and play a more critical role in new emerging storage market areas (e.g., real-time analytics, cloud storage, machine learning, and self-driving cars).

While the new application offers a great opportunity for NAND flash-based storage systems, it also presents other new challenges to be solved due to the variety of missions to be performed under more harsh operating conditions compared to existing applications. For example, consider the NAND flash memory for future self-driving cars. Autonomous vehicles should properly manage tera-bytes of data every hour under widely varying operating temperatures in a timely fashion [5]. Furthermore, since such a vehicle is expected to run for long times without failures, its on-board storage system should guarantee a long lifetime. Cloud storage service, where large amounts of data should be collected and processed, continuously struggle to guarantee a user experience, called QoS (Quality of Service), by eliminating the unpredictable variations in their service time. On the other hand, it is increasingly becoming important to protect sensitive personal data such as private photos, social-networking-service messages, or confidential data (e.g., medical records or proprietary documents) in storage devices from malicious attackers. Therefore, to successfully implement a modern storage system, NAND flash memory should satisfy various requirements not only *high-performance* and *long-lifetime* (key requirements of traditional applications) but also new requirements such as *high-security* and *real-time processing.*

Unfortunately, the current NAND flash-based storage systems are ill-suited to meet new flash requirements of emerging applications because most existing flash solutions have focused on resolving the limitations of large-capacity NAND flash memory (e.g., poor reliability or slow program latency). Therefore, to overcome new challenges, significant improvements are needed across the key design abstractions of a NAND flash-based storage system. Moreover, since it is hard to develop a single ultimate solution that can properly address all requirements, multiple optimization techniques should be developed and effectively integrated.

1.2 Dissertation Goals

Before exploring proper solutions, we analyzed the root causes of NAND flash-based storage systems making it difficult to support high-security and real-time processing based on a comprehensive evaluation using real 3D TLC flash chips. From NAND characterization studies, we identified the limitations of the existing techniques and obtained various optimization hints to be exploited in our study. Based on these hints, we developed device-level optimization techniques by revisiting (or re-designing) the conventional flash operations, and implemented system-level optimization techniques to take a fully advantage of the device-level techniques.

For example, to acquire the real-time properties of flash storage systems, we analyzed the root causes of non-deterministic read service time due to a long read tail-latency. This unpredictable and unfavorable phenomenon is not caused by single factor, but is caused by a combination of various factors in multiple layers constituting a storage system. In summary, as shown in Figure 1, the causes can be found in three different layers in a NAND flash-based storage system:

- **1st Layer -** *NAND flash memory*: Multiple read retries as NAND flash memory wears
- **2nd Layer -** *Flash Transition Layer (FTL)*: Command conflict by preceding I/O requests or management tasks in FTL
- **3rd Layer -** *Host System*: Priority-unaware Scheduling Policy

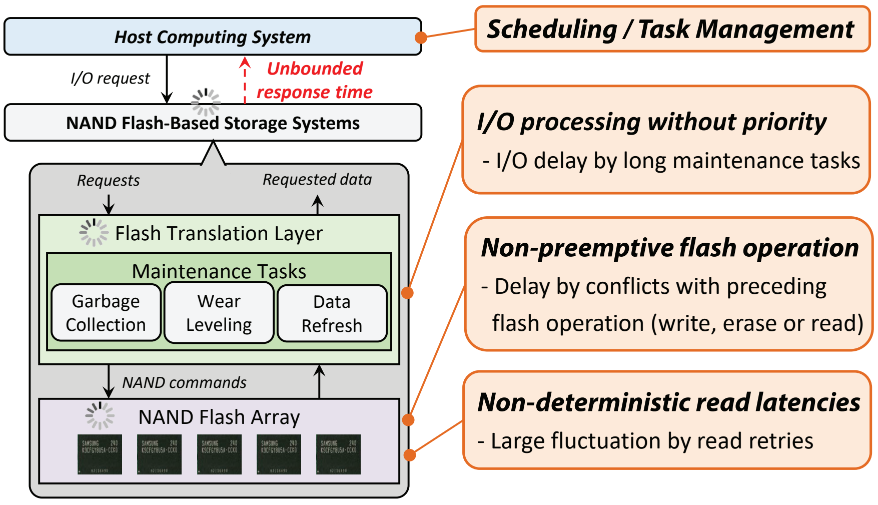

Host Computing System
I/O request
Unbounded response time
NAND Flash-Based Storage Systems
Requests
Requested data
Flash Translation Layer
Maintenance Tasks
Garbage Collection
Wear Leveling
Data Refresh
NAND commands
NAND Flash Array
Scheduling / Task Management
I/O processing without priority
- I/O delay by long maintenance tasks
Non-preemptive flash operation
- Delay by conflicts with preceding flash operation (write, erase or read)
Non-deterministic read latencies
- Large fluctuation by read retries

Figure 1: The root causes of non-deterministic read service time.

Based on the analysis results, we developed new optimization techniques to tackle the problem of non-deterministic read service time in NAND flash-based storage systems. First, to mitigate the number of read retries (`NumRetry`), since `NumRetry` is significantly varied as a NAND flash memory wears out, we attempted to accurately estimate the exact wear status of NAND flash memory by a new NAND aging marker (`RealWear`). Second, based on `RealWear`, we proposed a read retry mitigation technique (BoudedRead) that an optimal read reference voltages can be directly given without an iterative time-consuming search process. Third, we also proposed a new preemption technique to efficiently resolve the command conflicts between flash operations. By properly utilizing these optimization techniques, a host system is expected to develop a new scheduling (or task management) policy to guarantee the real-time properties. (This is not included in our paper, and remained as our future work.)

For the high-security in flash storage systems, we developed new data sanitization techniques through similar procedures to the above. We first analyzed the limitation of existing solutions which sanitize by physically destroying the data in NAND flash memory. And then, we proposed a new flash interface to sanitize security-sensitive data by disabling access to the sanitized data. Finally, we also introduced a system-level optimization technique that supports device-level techniques at low performance overhead.

In this dissertation, we propose new cross-layer techniques that aim at satisfying high-security and real-time processing of NAND flash-based storage systems. Our primary goal of this dissertation is as follows:

- Enabling a system software to vertically integrate the optimization techniques so as to maximize improvement gains.
- Developing system-level techniques to guarantee the requirement of real-time processing based a new NAND aging marker and preemption policy.
- Providing data sanitization techniques to reliably protect sensitive data without performance and reliability overheads.

1.3 Contributions

In this dissertation, we introduce three optimization techniques to add new solutions to the data security and real-time processing problem of NAND flash-based storage systems. The contributions of our work can be summarized as follows:

- We present a read retry mitigation technique using a new NAND aging marker, called `RealWear`, which can indicate the actual aging status of an individual NAND block in a much more accurate fashion over the existing P/E cycle-based aging marker. Based on the comprehensive device-level measurements, we quantitatively show the key variables related to the wear status of a NAND block, and design `RealWear` following a general model construction. Using real state-of-the-art 3D TLC NAND flash chips, we experimentally confirmed the validation of our model by comparing it with other existing NAND aging markers. To demonstrate the benefit of `RealWear` in flash storage systems, we present three case studies, LongLive, FastCopy, and BoudedRead. BoudedRead set an optimal reference voltage based on the value of `RealWear` and operating conditions. Our evaluation result of BoudedRead shows that no read can experience more than two read retries under all possible operating conditions, thus significantly reducing the tail latency of the flash storage system. To the best of our knowledge, BoudedRead is the first technique that supports the bounded read latency. BoudedRead can open new application areas for flash memory where timing constraints are important. In addition, over their counterpart techniques based on the P/E cycle-based aging marker, LongLive can extend the lifetime of individual NAND blocks by an average of 63% and FastCopy can improve the performance up to 21% by reducing GC overhead.

- We propose a novel priority-aware suspend/resume technique, called pSR, that handles all the cases where a read task conflicts with other flash commands (including read). Unlike the existing preemption techniques, pSR limits the number of suspend/resume operations by permitting the preemption command only at predefined safe points (that is experimentally determined), thus efficiently avoiding large reliability penalties due to excessive preemptions. Based on the accurate NAND error model, pSR partially allows read tasks to immediately suspend the ongoing flash operations with properly modifying the wear status of the target NAND block. Based on the proposed pSR command, we have developed an priority-aware FTL, called paFTL. When a read request arrives, the paFTL decides if an ongoing operation would be preempted or not depending on its priority, thus enabling a high-priority read task to be served in a timely manner without an unexpectedly long delay.

- We propose evanesco, a new low-cost data-sanitization technique for modern 3D NAND flash memory. Evanesco effectively sanitizes security-sensitive data by disabling access to the sanitized data until the corresponding block is physically erased. By using spare flash cells, evanesco reliably manages the data-access permission inside a flash chip without negative effect on the performance and reliability of a storage system. To take a full advantage of evanesco, We introduce SecureSSD, an evanesco-enabled storage system that implements evanesco at low performance overhead. By allowing a user to set the security requirements of written data with extended I/O interfaces, SecureSSD selectively sanitizes *only* security-sensitive data. We experimentally evaluate the reliability, performance, and lifetime of SecureSSD using 160 state-of-the art 3D TLC NAND flash chips. Our evaluations show that SecureSSD quickly sanitizes a page or a block without negatively affecting the reliability of the storage system. We compare SecureSSD to existing physical-sanitization techniques and show that SecureSSD significantly reduces the performance and lifetime overheads of data sanitization. Over existing reprogram-based techniques, SecureSSD increases the input/output operations per second (IOPS) performance by up to 4.8× (2.9× on average) and reduces the number of block erasures by up to 79% (62% on average).

1.4 Dissertation Structure

This dissertation is composed of six chapters including the introduction and conclusions which are at the first and the last, respectively. The four intermediate chapters are organized as follows:

Chapter 2 briefly explains the background for our work, including basics of NAND flash memory and overall architecture of NAND flash-based storage systems. We also examine the wear mechanism of NAND flash memory and its impact on flash operations.

Chapter 3 introduces a NAND aging marker, called `RealWear`, which accurately predicts the wear status of individual NAND blocks. We evaluate the inadequacy of the existing P/E cycle-based aging marker and discuss the root causes of the weakness of P/E cycle-based aging marker. We construct `RealWear` based on multiple variables that are closely related to the wear of NAND flash memory, and confirm its validations using the state-of-the-art 3D NAND devices. To take a full advantage of its benefits, we also propose a `RealWear`-aware FTL, `rFTL`, which manages NAND flash-based storage systems based on the value of `RealWear`. `rFTL` can further improves the existing optimization techniques, thus providing better reliability and performance of a flash storage system. Moreover, `rFTL` also enables a new optimization technique that reduces read latency fluctuations due to read retries, thus achieving *time-bounded* reads for a flash storage system.

In Chapter 4, we introduce a simple yet effective co-design of NAND devices and flash controllers to improve the read performance of NAND flash-based storage systems.. We first explain the limitations of the existing suspend/resume techniques and figure out the solution based on comprehensive NAND characterization studies. Then we propose design and implementation of a new suspend/resume command that can guarantee the read service time by properly handling the conflicts between high-priority read tasks and other flash operations.

Lastly, Chapter 5 presents a new integrated data sanitization technique, called Evanesco, for reliably protecting the sensitive data in NAND flash memory. Before introducing our techniques, we reveal the performance and reliability issues of the existing sanitization techniques. Then, we propose two new flash *lock* commands (`pLock` and `bLock`) that disable access to deleted data at both page and block granularity. After explaining the key mechanism of the two locking commands and their implementation details, to demonstrate the effectiveness of evanesco, we build SecureSSD, an emulation-based prototype solid-state drive (SSD) that implements an extended flash memory model with `pLock` and `bLock`. By using `pLock` and `bLock` only for security-sensitive data, SecureSSD keeps the performance overhead of data sanitization at minimum.

Chapter 2

Background

In this section, we first explain the basics of NAND flash-based storage systems, starting from the physical characteristics of underlying NAND cells and flash operations to the overall architecture of modern NAND flash-based storage systems. We also briefly describe the physical/structural characteristics of 3D NAND flash memory. We then review the wear mechanism of NAND flash memory and its impact on the performance and lifetime of storage systems. Finally, previous studies highly related to our work are summarized.

2.1 NAND Flash Organization

NAND flash memory consists of NAND cells, which store data, and peripheral circuits, which support flash commands such as read and write. Figure 2 illustrates a typical NAND block organization. Although the NAND cell, shown in Figure 2, is structurally similar to a normal MOS transistor, it is unique in that its threshold voltage V_{th} can be adjusted by injecting (ejecting) electrons into (out of) the floating gate[1]. Depending on the number of electrons in the floating

[1]In 2D NAND flash memory, floating-gate cell structures were typically used, while many 3D NAND devices adopt charge trap (CT)-type cell structures. The details will be explained in Section 2.4.

gate, the NAND cell works as an *off* or *on* switch under a given control gate voltage, thereby effectively storing data (encoded by V_{th}). For example, in single-level cell (SLC) flash memory [8], we can assign '0' state (i.e., data value '0') to high V_{th} in the NAND cell, and '1' state (i.e., data value '1') to low V_{th} in the flash cell.

As shown in Figure 2, a group of NAND cells (typically 8K-16K cells) form a wordline (WL) and multiple WLs (typically 128-256 WLs) form a block. In this block, there are n WLs and each WL consists of p NAND cells. For example, if there are 200 WLs in a block and each WL has 16-KB NAND cells, a NAND block has 3.2 MB of size. The WL can be logically divided into multiple pages which is the unit of read and program operations. A WL stores as many pages as the number of bits represented by a single NAND cell (up to four pages with state-of-the-art QLC technology). In single-level cell

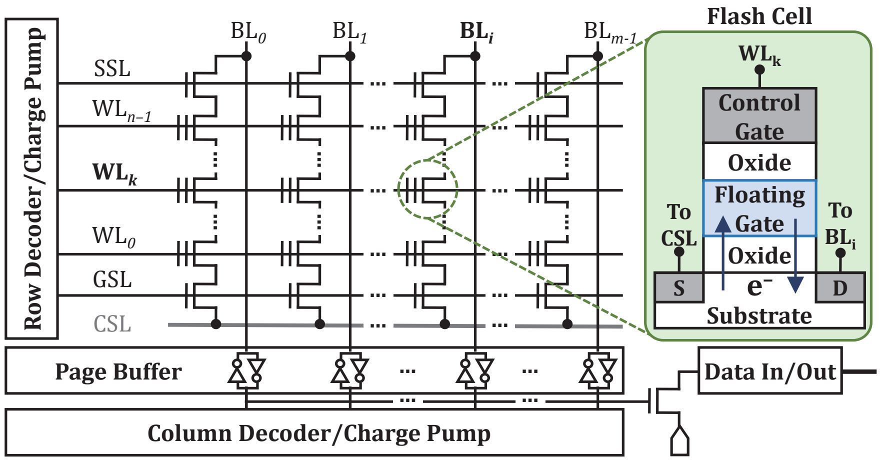


Figure 2: Organizational overview of a NAND flash block.

(SLC) flash memory, 1 WL corresponds to 1 page, while m-bit cell flash memory has m pages per a single WL. Since the NAND cells on the same WL$_k$ share the common WL$_k$ signal from the row decoder module, read and program operations are performed together at the same time, thus achieving high bandwidth. BLs, which are shared by all the blocks in a flash chip, are connected to the page buffer, which is used for off-chip data transfer through the data-in/out circuitry. Two select transistors at the top and bottom of a BL comprise the source select line (SSL) and the ground select line (GSL) of a block, respectively. By applying proper voltages to the SSL and GSL of a NAND block, we can activate (or de-activate) the block to perform flash operations.

2.2 Flash Operation

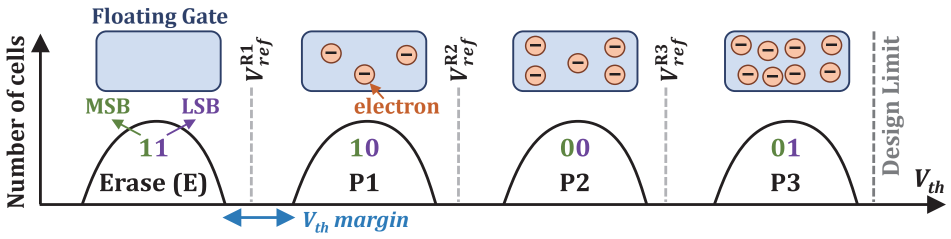
The *program operation* changes the data value of a NAND cell by increasing the cell's V_{th} (i.e., program operation can only change the data value from '1' to '0'). To increase the V_{th} of the selected NAND cells, *program* operation transfers electrons from the channel (i.e., substrate) into the floating gate of the selected NAND cells using FN tunneling [9] by applying a high voltage (i.e., $Vcg > 20$V) to the control gate in Figure 2. The FN tunneling current density J_{FN} is modeled as:

$$J_{FN} = A \cdot E_{ox} \cdot exp\frac{-k_B}{E_{ox}} \tag{2.1}$$

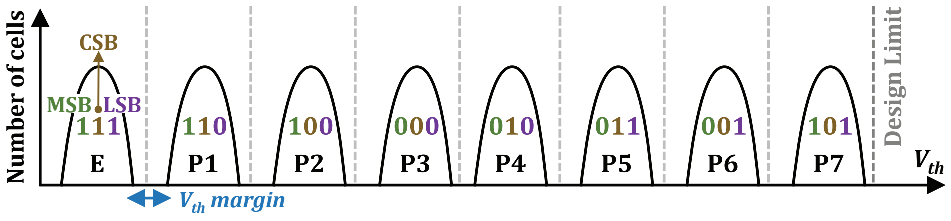
where A is a material/structure-specific constant, k_B is the Boltzmann constant, and E_{ox} is the electric field intensity across the T_{ox} layer. E_{ox} is given by (V_{fg}/d_{ox}) where V_{fg} is the floating gate voltage coupled by V_{cg}, and d_{ox} is the thickness of T_{ox} layer. As shown in Equation (2.1), the density of FN tunneling current is proportional to the electric field intensity across tunnel oxide (T_{ox}) layer. As a higher gate voltage is applied, more FN tunnel current flows, which means that more electrons are injected into the floating gate. On the other hand, the *erase operation* sets the data value of a NAND cell back to '1'. The erase operation decreases V_{th} of the NAND cells by applying a high voltage to the substrate (i.e., Vsub $>$ 20V) to remove electrons from the floating gate. Since the program operation can change the data value of a NAND cell only from '1' to '0', all the NAND cells of a page should be erased first to program data on the page (called *erase-before-program*). The erase operation works at block granularity because a high voltage is applied to the substrate that underlies the entire block.

To read the stored data from a page, the V_{th} levels of the NAND cells on the WL are probed using a read reference voltage V_{ref}. In Figure 2, when WL$_k$ is selected for read,[2] if the V_{th} of the i-th NAND cell in WL$_k$ is higher than V_{ref}, the i-th NAND cell turns *off*, so the cell current of BL$_i$ is blocked (i.e., the NAND cell is identified as '0'). On the other hand, if the V_{th} of the i-th NAND cell is lower than V_{ref},

[2]Since no other WL in the same block (e.g., WL$_{k-1}$ or WL$_{k+1}$) should affect the read operation on WL$_k$, the NAND cells in all other WLs should behave like pass transistors. Their gate voltages are set to V_{READ} ($>$ 6V), which is much higher than the highest V_{th} value of any NAND cell [10].



(a) m=2 : MLC flash memory.



(b) m=3 : TLC flash memory.

Figure 3: V_{th} distributions of 2^m-state NAND flash memory.

the i-th NAND cell turns *on*, so the cell current can flow through BL$_i$ (i.e., the NAND cell is identified as '1'). By sensing BLs from the selected WL$_k$, the stored data in the entire WL is read into the page buffer. Note that, if we can prevent the page buffer from buffering the data of a NAND cell, or prevent the data in the page buffer from being transferred out of the flash chip via the data-out path, we would prevent access to the stored data in any WL.

2.3 Multi-level Cell Flash Memory

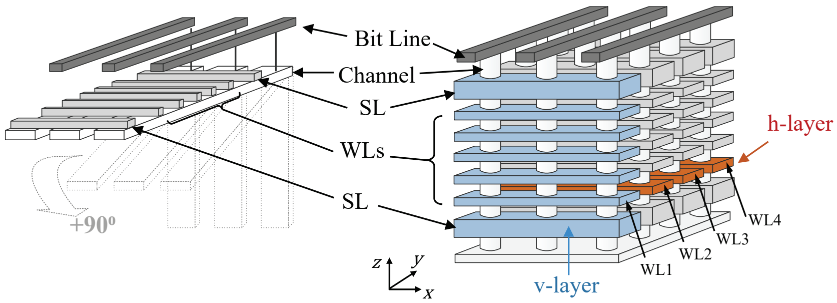
To improve the storage capacity of NAND flash memory, the multi-level cell (MLC) technique has been developed by NAND manufacturers. MLC technology was initially developed to store 2 bits per cell [11, 12], then extended to support 3 bits/cell (called TLC) [13, 14] and 4 bits/cell (called QLC) [3, 4]. Figure 3 illustrates V_{th} distributions for 2^m-state NAND flash memory that stores m bits within a single flash cell by using 2^m distinct V_{th} states, for $m = 2$ (MLC) and $m = 3$ (TLC). As m increases to store more bits in a flash cell, more V_{th} states should be squeezed into a limited V_{th} window, which is fixed at flash design time. Therefore, more careful management (e.g., smaller ISPP voltage steps [15]) is required for multi-level flash memory to form finer V_{th} states. Small ISPP voltage step (V_{ispp}) can keep V_{th} distributions of adjacent program states as distant as possible so that the interference between neighboring states can be minimized. Although narrowing V_{th} distributions for program states makes program states more error-resistant, it inevitably slows down the program speed ($tPROG$).

Furthermore, as m increases, the V_{th} margin (i.e., the gap between two neighboring V_{th} states) inevitably becomes smaller, as shown in Figures 3(a) and 3(b). When the V_{th} margin is smaller, the V_{th} distributions of two neighboring states are more likely to overlap under various noise conditions (e.g., long retention times [16, 17], cell-to-cell interference [18, 19], and read disturbance [20, 10]), which degrades the reliability of NAND flash memory. For example, MLC NAND flash memory can tolerate up to 3,000 program and erase (P/E) cycles, while TLC NAND flash memory can tolerate only about 1,000 P/E cycles [21]. As a result, the higher the flash capacity by increasing m, the lower the flash reliability and performance.

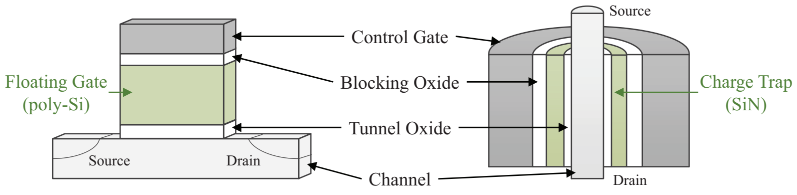
2.4 3D NAND Flash Memory

3D NAND flash memory [1, 2, 22], in which memory cells are vertically stacked, enabled the continuous growth in the flash capacity by overcoming various technical challenges in scaling 2D NAND flash memory. For example, 2D flash memory technologies had encountered the fundamental limits to scaling below the 10-nm process technology [23] because of the low device reliability (due to severe cell-to-cell interference) and high manufacturing complexity. By exploiting the vertical dimension for the capacity increase, 3D NAND flash memory has contributed to sustain the 50% per year growth rate in the NAND flash capacity.

Although an architecture of 3D NAND flash memory is conceptually described as if multiple 2D NAND flash layers are stacked in a vertical direction, the inner organization of 3D NAND flash memory is quite different from this logical explanation. Figure 4(a) illustrates an organizational difference in a NAND block between 2D flash and 3D flash. The 3D NAND block in Figure 4 (a) consists of five horizontal layers (h-layers), which are stacked along the z axis. Each horizontal layer consists of four word lines (WLs). Similarly, the 3D NAND block may be described to have four vertical layers (v-layers) in the y axis where each v-layer consists of five vertically stacked WLs that are separated by select-line (SL) transistors. As shown in Figure 4(a), when a 2D NAND block is rotated by 90 ° in a counterclockwise direction using the x axis as an axis of rotation, it corresponds to a single v-layer.



(a) An organizational difference.



(b) A difference in cell structures.

Figure 4: Illustrations of differences between 2D and 3D NAND.

Furthermore, most 3D NAND devices (e.g., TCAT [1], p-BICs [22] and SMArT [24]) adopt cylindrical charge trap (CT)-type cell structures. This CT-type cell uses a non-conductive layer of silicon nitride (SiN) that traps electrical charges to store bit information, while 2D NAND devices use floating gate cell structures which store bits in a conductor (e.g., poly-Si). As shown in Figure 4(b), this SiN layer has been modified into a three dimensional form that wraps around the channel, acting as an insulator that holds charges.

2.5 Wear Mechanism of Flash Memory

Figure 5 shows a schematic illustration of 3D NAND flash cell. The NAND cell is surrounded by dielectric materials (the tunnel oxide

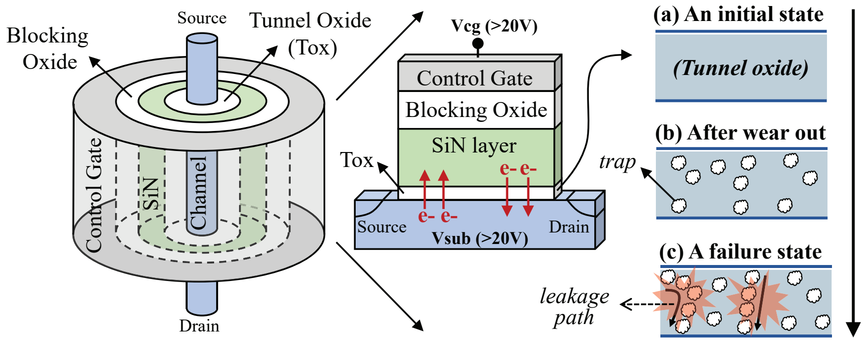
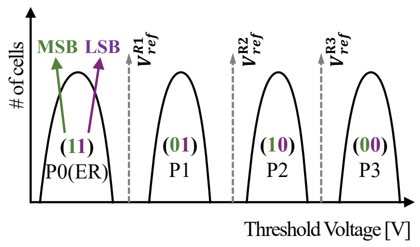


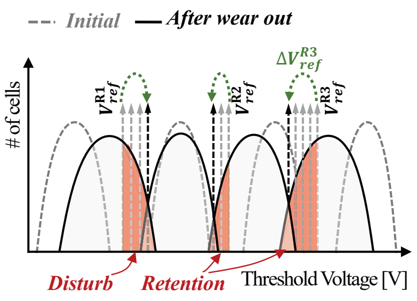
Figure 5: A schematic diagram of a 3D NAND flash cell.

and blocking oxide) so that electrons in the SiN layer are electrically insulated. This cell organization gives NAND flash memory a non-volatile characteristic so that the electrons trapped into the NAND cell do not leak even after power off. Ideally, the T_{ox} layer works as a perfect insulator with no impurities when the NAND cell is initially manufactured (i.e. an initial state), as shown in Figure 5(a). However, as NAND flash memory experiences repetitive program and erase operations, T_{ox} layer is progressively damaged by stress from a high program/erase voltage. As the damage to T_{ox} layer is accumulated, as illustrated in Figure 5(b), unexpected traps (i.e., defects) are generated and accumulated in T_{ox} layer. These traps make T_{ox} layer an inefficient insulator, making NAND cells *wear out*.

As shown in Figure 5(c), if the total amount of accumulated traps exceed a certain level by repetitive stress, the traps tend to be aligned and make a leakage path that electrons can move, so T_{ox} layer cannot longer acts as an insulator. Once such a path is formed in T_{ox} layer,



(a) An initial state of MLC flash memory.



(b) After wear out state of MLC flash memory.

Figure 6: Changes in V_{th} distributions of the MLC flash.

the NAND cell cannot reliably store its data any more, making T_{ox} layer a poor insulator.

2.6 Impact of NAND Cell Wear Out

Since the reliability of NAND flash memory depends on he reliability of the T_{ox} layer (as an insulator) in a NAND cell, the cell wear-out status directly affects all the aspects of the flash operations. In this section, we describe how bit errors increase with the wear status of the NAND cell and how this affects flash operations with focus on the read latency.

2.6.1 Impact on NAND Bit Errors

Figure 6 illustrates how V_{th} distributions of NAND states changes as NAND cells wear out. In our example, to simplify description, we use MLC flash memory (where $m = 2$) that stores two bits (LSB and

MSB bits) in a single NAND cell. At the initial (i.e., fresh) state, LSB and MSB bits, which are stored in the same NAND cell, can be read without any problem by identifying the V_{th} state of the NAND cell using the reference voltages V_{ref}^{Ri}'s. Note that V_{ref}^{Ri} is the read reference voltage used to distinguish Pi state from P$(i-1)$ state. For example, V_{ref}^{R3} distinguishes P3 state from P2 state.

However, as the NAND cells wear out, the V_{th} distribution of the NAND cells gets widened and shifted under various noise conditions, so that V_{th} states are more likely to be overlapped with V_{ref}^{Ri}'s. When NAND cells are programmed, neighboring cells belonging to the ER-state (erased cells) are softly programmed, so that their V_{th} move to the right (We call this type of errors *program disturbance* [25].). The repeated read operations also shift V_{th} of P0-state cells to the right. When NAND cells are read, V_{READ} (6∼7V) is applied to the all WLs in the same block except for the target WL (See Section 2.2). When pages in the same block are frequently read, ER-state cells in the block are softly programmed, thus shifting their V_{th} to the right (We call this type of errors *read disturbance* [10].). If NAND cells are left for a long time after a program operation, a charge loss occurs by the stress-induced leakage current (SILC) through traps in the tunnel oxide layer. These errors, as shown in Figure 6(b), are called retention errors [17]. The charge loss shifts the V_{th} of program states to the left. These various noise conditions shift NAND states, thus making NAND states overlapped each other.

Once the V_{th} states are overlapped, the original data cannot be correctly read with V_{ref}^{Ri}'s (i.e., cannot identify whether '1' or '0'). For example, when P3 is shifted to left due to a long data retention time, it may overlap with V_{ref}^{R3}, thus introducing a large number of bit errors. The shaded regions in Figure 6(b) indicate such overlapped errors, and the larger the overlapped region, the greater the number of bit errors. Since a flash page should be reliably read within its lifespan, the lifetime of a flash storage system is directly affected by the number of bit errors in the flash page which, in turn, is directly related to the wear status of NAND cells in the flash page.

2.6.2 Impact on the Read Latency

When a flash page is read with V_{ref}^{Ri}, a flash controller monitors if bit errors are beyond the ECC (Error Correction Code) engine's correction capability. If bit errors cannot be corrected by the ECC engine, the flash controller *retries* a read operation with slightly shifted V_{ref}^{Ri} values until the page can be successfully read without uncorrectable errors. For example, as shown in Figure 6(b), when P3 is shifted due to a long data retention time, the initial V_{ref}^{R3} should be adjusted with an offset ΔV_{ref}^{R3} to minimize bit errors. We call this new reference voltage as an *optimal* reference voltage (e.g., $V_{ref}^{R3} - \Delta V_{ref}^{R3}$). If V_{th} states of NAND flash memory are substantially shifted from their initial distributions, the number `NumRetry` of read retries can be significantly increased.

In general, the read latency $tREAD$ of a NAND flash memory can be expressed as;

$$tREAD = (tR + tDMA + tECC) \times (\texttt{NumRetry} + 1) \tag{2.2}$$

where tR is the flash page access time, $tDMA$ is the data transfer time from a flash chip to a flash controller, and $tECC$ is the error correction time by the ECC engine. As shown in Equation (2.2), $tREAD$ of a NAND flash memory depends linearly on `NumRetry`. Therefore, to minimize the read latency, `NumRetry` should be minimized. For example, in Figure 6(b), if ΔV_{ref}^{R3} could be identified without four successive read retries, $tREAD$ can be substantially reduced. Since the wear status of NAND cells is one of the key factors that affect `NumRetry`, knowing the exact wear status of NAND cells is an important prerequisite of minimizing $tREAD$.

2.7 NAND Flash-Based Storage Systems

Figure 7 depicts an overall architecture of modern NAND flash-based storage systems. To achieve high performance, NAND flash-based storage systems employ several NAND devices. The read and write bandwidth of a single NAND device is far limited even over the traditional hard disk drives (HDDs). For example, a state-of-the-art 3D NAND device takes about 50 μs to read 4-KB page, thus providing only 80 MB/s of read bandwidth. NAND flash-based storage systems overcome this limited performance of a single NAND device by allowing multiple NAND devices to be operate in parallel. In order

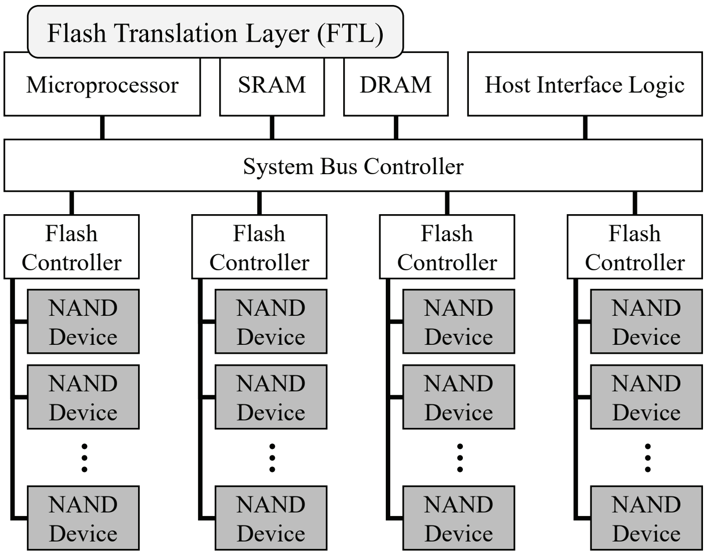


Figure 7: An overall architecture of typical NAND flash-based storage systems.

to minimize performance interference between NAND devices, NAND flash-based storage systems are design to have as many buses as possible (i.e., as less NAND devices per bus as possible) with dedicated hardware controllers which can handle multiple NAND commands on their NAND device simultaneously.

Most modern NAND flash-based systems run a special storage firmware, called a *flash translation layer* (FTL). Due to the unique features of NAND flash memory, such as the erase-before-write nature and operation-unit asymmetry, a NAND flash-based storage system should deal with underlying NAND devices in different ways over other storage medium such as DRAM and magnetic disks. An FTL is responsible for providing backwards compatibility to the block I/O interface, hiding the unique features of NAND flash memory.

FTLs support two main functionalities: the address translation and garbage collection (GC). NAND flash memory does not support *in-place update*, so an FTL should handle host writes in an append-only manner, writing incoming data to free (i.e., erased) pages. As a result, when a host system updates data at the same location in the host system view, the physical location of data are changed. In order to handle future reads on the data, an FTL have to keep track of the physical location (i.e., physical address) in the storage system corresponding the location in the host system view (i.e., logical address). Since this logical-to-physical (L2P) mapping is accessed in every read/write requests, the effectiveness of managing L2P mappings is critical to the performance of NAND flash-based storage systems.

In order to maintain free pages for future writes, an FTL has to perform GC. GC procedure reclaims *invalid* pages whose data are not necessary any longer as host system deleted the data or update them with new data at their corresponding logical address. It requires a block erasure, so all the valid pages in a victim block to be erased have to be moved to other free pages. These copy overheads can significantly degrade the performance and lifetime of NAND flash-based storage systems, so typical garbage collectors choose the target block which has a largest number of invalid pages.

2.8 Related Work

2.8.1 NAND Aging Markers

There have been many attempts to design a new NAND aging marker that can better represent the exact aging status of NAND flash memory. However, few investigations have successfully converted the theoretical findings on the NAND aging process from the NAND device physics to practical NAND aging markers that can be utilized at the FTL level. For example, Woo *et al.* [26] proposed a block aging metric based on $tPROG$ as well a $N_{P/E}$. However, as shown in Figure 12(b), $tPROG$ is not a meaningful input to the wear status of the NAND flash memory. It strongly correlates with the NAND wear only when blocks are young. When blocks become old, it is difficult to relate $tPROG$ with the NAND wear. Furthermore, this technique did not account for other key factors affecting the wear status of NAND flash cells (e.g., self-recovery effect) so its classification accuracy is questionable. Similarly, Pelato *et al.* [27] suggested another block aging metric exploiting bit-error rates (BERs) of a NAND block in addition to $tPROG$ and $N_{P/E}$. Wang [28] also proposed the super-block management technique, called *WAS*, by wear-leveling NAND blocks based on BERs. Although it may work better over [26], it shares the same limitation of [26]. Unlike existing FTL-level aging markers, `RealWear` is unique in that it comprehensively considers all the key factors that affect the NAND flash wear.

`RealWear` does not make any novel new contribution to the NAND wear model from the device physics perspective. However, unlike many existing theoretical results on the NAND wear model that were not successfully exploited at the FTL level, `RealWear` takes full advantages of theoretical findings at the practical NAND aging marker level. For example, Mielke *et al.* [29] performed extensive studies about the wear mechanism of flash memory with particular attention to the tunnel oxide degradation. They theoretically demonstrated how oxide degradation, which is caused by trap generation in the tunnel oxide, affects flash operations and reliability (i.e., BERs) and proved it by extensive evaluations at the device level. By quantifying the relationship between the trap density and the NAND cell wear, they successfully modeled the behavior of individual NAND cells under various conditions. Spinelli *et al.* [30] also presented comprehensive research to understand the wear of NAND flash memory and its root cause. Based on the experimental observations on device physics, they offered a new model to predict the V_{th} of NAND cells and a new wear mechanism for the state-of-the-art 3D NAND flash memory. Although many device-level characterization studies on NAND cell wear have been conducted, unfortunately, the results are rarely utilized as a NAND aging marker because of a huge overhead (e.g., cost), thus limiting their practical applications in real storage systems.

2.8.2 Preemption Command

There have been several previous studies that attempted to resolve the long read-tail latency. We briefly discuss related prior works focusing on command preemption and read retry mitigation.

Wu *et al.* [32] have suggested a command-preemption technique that immediately suspends the ongoing program/erase operations for reads. However, since it does not consider the side effect of excessive suspensions, it may incur serious reliability issues. Kim *et al.* [33] have proposed a hybrid erase suspension technique that allows immediate erase suspensions until timeout happens. Although this technique attempts to avoid the reliability degradation by deferring the erase suspension after a timeout, it is not designed based on precise modeling using real flash device. Furthermore, since none of them supports read preemption, they are limited in fully differentiating the read performance according to the I/O priority of each task.

Many studies have also tackled the read retry problem which significantly affects user-experienced performance of NAND flash-based storage systems. Cai *et al.* [17] proposed *ROR* (Retention Optimized Read) scheme to reduce read retries by adjusting read reference voltages periodically. Although *ROR* works well in low-density 2D flash memory, it could be challenging to apply *ROR* in modern flash storage systems using high-density 3D flash memory. For 48-layer 256-Gb 3D TLC flash memory, when *ROR* updates V_{ref}^{Ri}'s for each block once a day, it can take about 2.7 hours to monitor the entire flash storage once, resulting in a huge monitoring overhead. Li *et al.* [31] reduced the number of read retries by accurately estimating the optimal read voltage based on error patterns in a small set of reserved cells that store predefined data. The proposed scheme allows most read requests can be served with a single retry, but lowers the ECC capability since sentinel cells occupy the spare area reserved for the ECC parity. This drawback would render the approach difficult to be applied in modern TLC/QLC NAND flash memory where the reliability is a serious concern.

2.8.3 Data Sanitization

The most fundamental approach to sanitizing data is to physically destroy the data in the storage medium. Diesburg *et al.* [34] propose a framework that enables a file system to notify an SSD to immediately erase blocks containing security-sensitive data. However, such an erase-based approach can significantly degrade the performance and lifetime of SSDs, introducing a large number of data copies. To sanitize data without requiring block erasure, several studies [35, 36, 37, 38] propose techniques based on scrubbing, which destroys the V_{th} values of cells in a target page in a WL. Scrubbing a WL in SLC flash memory is relatively simple as there is only one page in a WL. However, for MLC flash memory, scrubbing techniques need to copy other valid pages in the same WL to some other WLs before destroying the V_{th} values of the target page in a WL. To mitigate the performance overhead of scrubbing in MLC flash memory, Lin *et al.* [39] propose the one-shot reprogramming technique that enables the destruction of the V_{th} values of individual pages in a WL separately from each other. However, as shown in Section 5.1.2, the one-shot reprogramming technique is not easily applicable to modern 3D NAND flash storage systems, since it cannot meet the reliability requirements due to significant over-programming errors.

Multiple works use data encryption to implement low-overhead data sanitization techniques for flash-based storage systems [40, 41, 42, 43]. The schemes encrypt security-sensitive data with a cryptographic algorithm, such as AES [44], and delete the encryption key when the data needs to be sanitized. Since it is almost impossible to obtain original data without the encryption key, the encrypted data can be effectively destroyed just by deleting the encryption key. However, encryption may not always be desirable due to performance overheads or resource constraints, and it also requires a complicated key management to satisfy stringent security requirements. If the encryption key is compromised [45, 46], this solution becomes ineffective.

Chapter 3

Read Retry Mitigation Using a New NAND Aging Marker

In this chapter, we introduce new *read retry mitigation techniques* that are applicable to modern storage systems. As shown in Section 2.6.2, read retries can linearly increase the read latency ($tREAD$), and the number of read retries (`NumRetry`) is closely related to the wear status of a NAND block. For example, the data is read from multiple NAND blocks that have a different wear status, flash storage systems can suffer from large fluctuations in the read latency due to largely different `NumRetry` per NAND blocks. If we know the exact wear status of NAND flash memory, an optimal read reference voltages can be directly given without an iterative time-consuming search process, so the number of read retries (`NumRetry`) can be ideally eliminated. This chapter consists of two parts; One is to design a new NAND aging marker (`RealWear`) to accurately represent the individual wear status of NAND blocks, and the other is to develop the read retry mitigation technique (BoudedRead) using the optimal read reference table that is constructed based on `RealWear`.

3.1 Motivation: Inadequacy of P/E-cycle based aging markers

In theory, the wear status of a NAND cell is determined by the amount of damage accumulated in the tunnel oxide layer of the NAND cell from flash operations (e.g., program or erase). However, because of practical difficulties, the defect in the tunnel oxide layer cannot be directly measured during run time. Instead, the wearing degree of NAND cells is approximately estimated using a wear predictor that is based on the known causes of degrading the tunnel oxide layer.

The most common wear indicator is to count the chronological age of a NAND cell based on the number of program/erase (P/E) cycles (i.e., the number of program and erase operations the NAND cell has experienced), in a similar fashion as the chronological age of a human being. Since a high electrical voltage (¿ 20V) is known to damage the tunnel oxide layer during the program and erase operations, the number of P/E cycles has been regarded as an effective and practical proxy indicating the wear status of NAND cells.

Intuitively, the role of a P/E-cycle based aging marker is similar to that of the chronological age of a human being. (In this paper, we call a P/E-cycle based aging marker `PeWear`.) For example, individuals with the same chronological age could have widely different biological ages due to their genetic differences such as the length of telomere [47]. Even twins with the same genetic characteristics may have different biological ages because of differences in their life styles and living environments [48]. Similarly, when two NAND blocks experience the same number of P/E cycles, their wearing degree could be significantly different. This difference in aging characteristics between NAND blocks is caused by various factors such as (1) process variations occurred while they are manufactured, (2) I/O workload variations in different NAND blocks, and (3) variations in their operating environment. Although these factors affect the wearing degree of the NAND block in both directions (i.e., less wearing or more wearing), the P/E cycle-based aging marker cannot reflect these factors at all.

In this section, we report our evaluation results on the inadequacy of the existing P/E cycle-based aging marker and discuss the root causes of the weakness of P/E cycle-based aging marker.

3.1.1 Evaluation Methodology

Evaluation Metric Before we present our evaluation results on a P/E cycle-based aging marker, we describe a general methodology for evaluating a NAND aging marker `M`. As shown in Section 2.5, the NAND cell wear is closely related to the number of traps (i.e., a trap density) accumulated in the tunnel oxide layer during flash operations (i.e., P/E cycles). However, since the true trap density in T_{ox} can only be measured by a destructive way using a highly-sophisticated tools (e.g., SEM [49]), it is practically impossible to measure trap density during run time.

As an alternative metric to represent the wear status, we use the number $N(t)$ of retention errors after t-month retention time. Many previous studies about NAND physics have shown that the number of retention errors has a near-linear relationship with the trap density [50, 51]. Furthermore, for recent multi-level NAND flash memory, retention errors are dominant in determining the flash reliability, especially when the NAND flash memory is aged [6, 17].

Following the common industry practice (i.e., the JEDEC standard [52]), $N(t)$ is measured by using an *accelerated lifetime test.* For example, to emulate a 12-month retention time condition, we baked the flash chips at 85°for 13 hours (which is equivalent to one year at 30°based on the Arrhenius's law[1] [53]). We measured $N(t)$ while varying both the number of P/E cycles (from 0 to 20K) and the data retention time (from 0 to 12 months)[2]. In our evaluations, a NAND block is regarded as dead (or failed) when $N(12)$ of the NAND block outnumbers the maximum number N_{ECC} of bit errors that can be corrected by an ECC module. (Because of the page limit, we only discuss the key results under the 12-month retention time requirement which

[1]The Arrhenius's Law is given as follows:

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

where AF is the acceleration factor between t_1 and t_2. t_1 is the dwell time under temperature T_1, and t_2 is the dwell time under temperature T_2. k_B is the Boltzmann constant (8.62×10^{-5} eV/K), and the activation energy (E_a) is a process-dependent constant, which is typically 1.1 eV for NAND flash memory [52].

[2]Data center applications usually require 2∼3 DWPD (Drive Writes Per Day) [54, 55] with the warranty period of 3∼5 years. Since host writes are considerably amplified, NAND flash memory needs to support at least 7K P/E cycles for such data center applications. Since most NAND blocks are still usable after 7K P/E cycles, we measured $N(12)$ while increasing the number of P/E cycles up to 20K P/E cycles for a comprehensive characterization study.

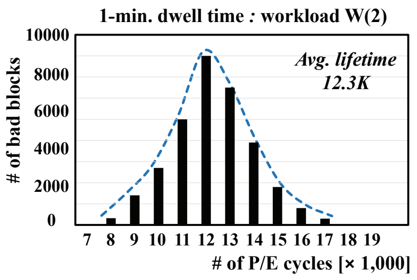
is often the worst-case reliability condition.)

Evaluation Workload We used simple synthetic I/O benchmark programs in our evaluations. To reflect wear-related key factors, a synthetic benchmark program repeats a sequence $S(n)$ of three flash operations where $S(n)$ is given by [s1: erase a block B, s2: program all the pages in B, and s3: read $1000 \times n$ pages from B]. By varying n in s3, we effectively simulate the dwell time[3] which is known to affect NAND cell wear. (The dwell time, D_t^B of a NAND block B, is defined as the average interval length between consecutive erase operations to the NAND block B [56].) We report our evaluation results for two benchmark programs, $W(2)$ and $W(10)$, where $S(2)$ and $S(10)$ were used, respectively. $W(2)$ is used to mimic an update-intensive workload with short dwell time (e.g., 1 minute) while $W(10)$ simulates a read-intensive workload with longer dwell time (e.g., 5 minutes). Furthermore, to avoid pattern dependent reliability problems of written data [57], 8 different pseudo-randomized patterns were used when pages are programmed.

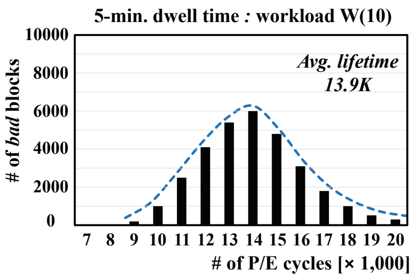
3.1.2 Evaluation Results

Bad Block Classification We first evaluated `PeWear` for deciding if a NAND block is bad or not. When `PeWear` is used to classify NAND blocks, a bad-block classifier decides that a NAND block B is bad when the number $N_{P/E}^B$ of P/E cycles exceeds the maximum allowed

[3]As will be explained in Section 3.1.3, the T_{ox} layer of NAND cells may be cured during the dwell time, thus improving the wear status of NAND cells.

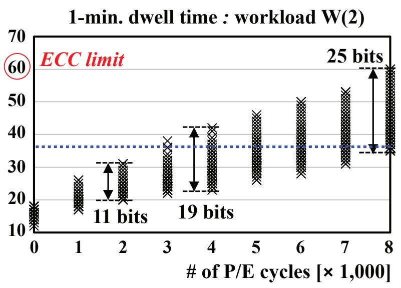


(a) A bad block distribution in W(2).

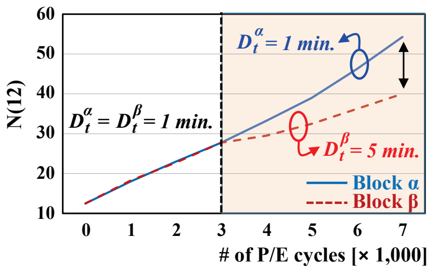


(b) A bad block distribution in W(10).

Figure 8: `PeWear` evaluation results: Bad block classification.



(a) Per-sector error distributions.



(b) Error variation over D_t.

Figure 9: `PeWear` evaluation results with varying $N^B_{P/E}$ values and dwell times.

number $N^{Max}_{P/E}$ of P/E cycles. If `PeWear` were an ideal NAND aging marker, any read request to a block B' with $N^{B'}_{P/E} > N^{Max}_{P/E}$ would fail while all the reads to a block with $N^{B'}_{P/E} \leq N^{Max}_{P/E}$ would be successful. To evaluate the effectiveness of `PeWear` in a bad-block classifier, we collected $N^B_{P/E}$ values when blocks became physically bad, not logically bad by a heuristic criterion such as `PeWear`. We use $N^{Max(B)}_{P/E}$ to denote $N^B_{P/E}$ value of the block B when read to the block B fails for the first time.

Figures 8(a) and 8(b) show how (physically) bad blocks are distributed over different $N_{P/E}^{B}$ values. If `PeWear` is an ideal NAND aging marker, all blocks would reach its maximum lifetime at the same number of P/E cycles. However, wide distributions over varying P/E cycles clearly indicate that `PeWear` does not meet the requirement of an ideal aging marker. Our evaluation result reveals that some blocks are able to withstand a larger number of P/E cycles than others before becoming unusable. For example, some NAND blocks showed uncorrectable errors at 8K P/E cycles while others were still usable after 16K P/E cycles. Obviously, `PeWear` cannot accurately represent the exact wear status of NAND blocks. Furthermore, when we use `PeWear` as a NAND aging marker, to avoid data loss, we must set $N_{P/E}^{Max}$ less than the minimum value of all $N_{P/E}^{Max(B)}$ values. Therefore, the lifetime of most NAND blocks are wasted. For example, 7K P/E cycles may be used as $N_{P/E}^{Max}$ in Figure 8(a), thus wasting the lifetime of all the NAND blocks.

Figures 8(a) and 8(b) also show the impact of the dwell time on the flash reliability. For the $W(2)$ workload with the 1-minute dwell time, an average NAND block becomes bad around after 12.3K P/E cycles. On the other hand, as the length of dwell time increases to 5 minutes in $W(10)$, an average NAND block becomes bad around after 13.9K P/E cycles. Since `PeWear` does not consider the dwell time, it is insufficient to be an ideal aging marker.

Per-Sector Error Variations Since $N(12)$ is known to have a strong correlation with the wear status of a NAND block (as explained in

Section 3.1.1), we evaluated how closely `PeWear` is related to $N(12)$. We measured the per-sector $N(12)$ values because the ECC engine in a flash controller recovers the errors in the unit of sector. Figure 9(a) shows that $N(12)$ values can be significantly different even at the blocks with the same P/E cycles. When $W(2)$ was used, for example, at 4K P/E cycles, a sector from the worst NAND block experienced 19 more bit errors than that from the best NAND block. Error variations between NAND blocks get wider as well with the number of P/E cycles. At 7K P/E cycles, the difference between the maximum and the minimum number of bit errors increases to 25 bits from 19 bits at 4K P/E cycles. Note that Figure 9(a) clearly indicates that `PeWear` should be revisited to be an ideal aging marker. For example, as shown in a dotted line in Figure 9(a), when the wear status of a block B is about 37 (as represented by its $N(12)$ value), the number of P/E cycles of the block B can be any value between 3K and 8K.

3.1.3 Root Causes of Inadequacy in `PeWear`

From the device physics perspective, there are three key variations among NAND blocks that were not reflected by the P/E-cycle based aging marker. We explain three variations in this section which help to understand why `PeWear` is inadequate as a NAND aging marker. (Since the key motivation for developing `RealWear` is to better reflect three variations in a NAND aging marker, this section also serves as a motivation of `RealWear`.)

Process Variations The difference in aging characteristics among

NAND blocks is mainly caused by process variability that occurs while they are manufactured. As shown in Section 2.5, the wear of NAND cells is resulted from the T_{ox} degradation due to high voltage stress applied during program and erase operations. Equation (2.1) for J_{FN} in Section 2.2 shows that once V_{SiN} is fixed, the voltage stress on the T_{ox} is determined by the thickness of T_{ox} (i.e., d_{ox}).

NAND cells with a thinner d_{ox} are more susceptible to stress by a high program or erase voltage, which makes them easier to wear out than those with a thicker d_{ox}. For example, when a program or erase voltage of 30V is applied to a NAND cell, the NAND cell with d_{ox} of 6 nm is exposed to the stress of 50 MV/cm. On the other hand, the NAND cell with d_{ox} of 5 nm is exposed to the stress of 60 MV/cm. As the number of P/E cycles increases, each NAND cell accumulates a different amount of stress due to the process variability[4], resulting in heterogeneous aging characteristics between NAND blocks even if they experience the same P/E cycles as shown in Figure 9(a).

I/O Workload Variations Although `PeWear` faithfully tracks the number of erase/program operations, it does not account for the dwell time D_t^B of a NAND block B, which significantly affects the NAND aging process because of its self-recovery effect on the cell wear. When a NAND block is left idle without any active program/erase operation, the wear status of the NAND block may be partially improved because the damage to the T_{ox} layer can be cured during the dwell time.

[4]For example, d_{ox} of flash cells at the edge of a wafer can be thinner than that at the center of the wafer due to the *loading effect* [58].

This phenomenon, often called as the detrapping process, is resulted from degenerating traps in the T_{ox} layer by thermal energy. The self-recovery effect is known to be logarithmically proportional to D_t^B [59, 60].

Since D_t is decided by how I/O operations are performed, the I/O characteristics of an I/O workload directly affects the wear status of NAND blocks. For example, in an update-intensive workload, D_t^B tends to be short because NAND blocks should be frequently erased and programmed. Figure 9(b) illustrates why an ideal NAND aging marker should consider the effect of D_t^B. To minimize the impact of the process variation on this evaluation, we selected physically adjacent two NAND blocks (say, the block α and the block β). As shown in Figure 9(b), two blocks exhibit the same $N(12)$ values when their D_t^B values are equal. However, when NAND blocks experience different dwell times, their $N(12)$ values significantly differ. The block β, which has 5 times longer D_t^B over the block α, wears more slowly than the block α.

Operating Environment Variations Various environmental conditions (such as operating temperatures) can also affect the wear status of a NAND block. For example, the self-recovery effect is proportional to the ambient temperature of a storage system because the detrapping process can be accelerated by thermal energy [61]. Therefore, although the initial aging status of two NAND blocks were the same, when they experience different ambient temperatures, the aging rate of two blocks can be significantly different.

As commonly done, the impact of operating temperature on the self-recovery effect can be understood by converting the impact of operating temperature to that of the dwell time at a baseline temperature, say, 30°. The converted dwell time at the baseline temperature is called as the effective dwell time D_t^{eff}. The dwell time at x°can be converted to the effective dwell time at 30°using the Arrhenius's Law [53]. For example, the dwell time of 1 hour at 50°corresponds to the effective dwell time of 13 hours at 30°. Since the impact of operation temperature can be converted to that of the dwell time, if operating temperature is known (e.g., from an on-board thermal sensor), the dwell time can handle both variations from operating environment and an I/O workload.

3.1.4 Extended `PeWear` Markers

To improve the accuracy of `PeWear`, several NAND aging markers have been proposed by extending `PeWear` with additional factors that were related to the wear status of a NAND block. For example, Woo *et al.* [26] proposed a block aging metric by combining the program latency $tPROG$ and $N_{P/E}$. (We call this scheme as `PeWear+`.) Pelato *et al.* [27] suggested an extended version of `PeWear+` by exploiting the bit error rate of a NAND block. (We call this scheme as `PeWear++`.) Although both `PeWear+` and `PeWear++` work better over `PeWear` as will be shown in Section 3.3, they do not fully overcome the fundamental limitations of `PeWear` because they cannot adequately handle three variations explained above.

3.2 New NAND Aging Marker: `RealWear`

To overcome the limitations of the P/E cycle-based aging marker (as described in Section 3.1), `RealWear` is proposed based on a new flash wear model that was built from our extensive flash characterization study. As in Section 3.1, we used the number $N(12)$ of errors as a metric for determining the wear status of NAND cells. For a given block B, we define its age, denoted by $age(B)$, as $N(12)/N_{ECC}$. Since we assume that $N(12) \leq N_{ECC}$, $age(B)$ ranges from 0 to 1. Since $N(12)$ of the block B should be based on the number of bit errors after the 12-month retention time, it cannot be measured on-line when it is needed to decide $age(B)$. Therefore, the key to designing a NAND aging marker lies in estimating $N(12)$ accurately without waiting for 12 months.

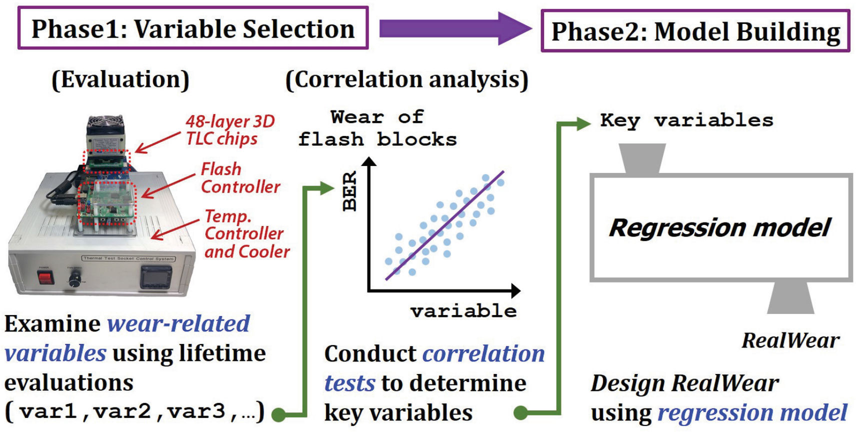

Phase1: Variable Selection
Phase2: Model Building
(Evaluation)
(Correlation analysis)
48-layer 3D TLC chips
Flash Controller
Temp. Controller and Cooler
Examine wear-related variables using lifetime evaluations (var1,var2,var3,...)
Wear of flash blocks
BER
variable
Conduct correlation tests to determine key variables
Key variables
Regression model
RealWear
Design RealWear using regression model

Figure 10: An overview of model building for `RealWear`.

Table 1: A summary of candidate variables.

Type		Description	Inclusion
Latency	$tBERS$	Erase latency	O
	$tREAD$	flash page access time	X
	$tPROG$	Program latency	X
Bit errors	$N(0)$	No. of bit errors right after program	O
Noisy Factors	ICI	Inter-cell interference	X
	RTN	Random telegraph noise	X
	BPD	Back Pattern Dependency	X
Dynamic Factors	$N_{P/E}$	P/E cycles	O
	D_t	Dwell time	O
	K_{amb}	Ambient temperature	O

Figure 10 shows an overall procedure that we used in building a model for `RealWear`. We followed a general model construction approach which consists of two key phases: *variable selection* and *model-building*. In the variable selection phase, we first collected wear-related candidate variables based on their known relationship with the wear status of NAND blocks. We empirically evaluated each candidate variable to check if it was sufficiently correlated with bit errors (i.e, $N(12)$). In the model building phase, we constructed a model for estimating $age(B)$ of a NAND block B using regression analysis of the selected variables.

3.2.1 Variable Selection

We initially collected 10 candidate variables, which are summarized in Table 1, that are known to be related to the wear status of NAND flash memory. Since our goal was to build a practical aging marker that can be used by a flash/SSD controller, we considered a wear-related variable as a candidate variable only if they can be accessible using an open NAND interface (e.g., ONFI [62].) The candidate variables are grouped into four types. The variables in the latency type are known to change their latency depending on the wear status of a NAND block [26, 27, 6], thus making them good indicators for representing the NAND aging characteristics. $N(0)$, which represents the number of errors right after programming a block with no data retention time, was selected as a candidate variable because it can indicate the status of T_{ox} layer before errors from other factors (such as retention errors or read-disturb errors) are included in $N(t)$ [27]. Furthermore, $N(0)$ can be directly measured with little overhead. Since the worst $N(0)$ value would be needed for `RealWear`, for a given block, we measure only once the number of bit errors from the LSB page of the most vulnerable WL (e.g., an edge WL) in the block, which takes less than 20μs. The variables, ICI[5] [19], RTN[6] [63] and BPD[7] [20], in the noisy factors category were included because they distort V_{th} distributions of NAND cells as NAND flash memory gets aged, thus potentially affecting $N(12)$. D_t and K_{amb} (the ambient temperature) were included because they are known to affect the wear status of a NAND block by the self-recovery effect.

To select a proper subset of variables from 10 candidate variables,

[5]Inter-Cell-Interference (ICI) can be decided by measuring how much the initial V_{th} value of a victim NAND cell is affected by adjacent NAND cells.

[6]Random Telegraph Noise (RTN) can be measured by the error variance during repetitive read operations (e.g., 10 reads in our evaluation).

[7]Back Pattern Dependency (BPD) is the read error variance depending on the data pattern written in a NAND block.

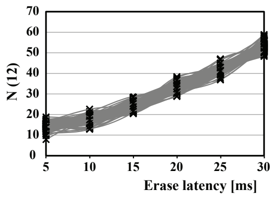
we performed comprehensive evaluations using 160 state-of-the-art (48-layer) 3D TLC flash chips. To minimize the potential distortions in the evaluation results, for each test scenario, we evenly selected 120 test blocks from each chip at different physical block locations, and tested all the WLs in each selected block. We tested a total of 3,686,400 WLs (11,059,200 pages) to obtain statistically significant experimental results. By using an in-house custom test board (equipped with a flash controller and a thermal controller), we evaluated 10 candidate variables using $N(12)$ values under various operating conditions.

As shown in the last column of Table 1, we finally selected 5 variables ($tBERS$, $N(0)$, $N_{P/E}$, D_t, and K_{amb}) in estimating $N(12)$. $tBERS$ and $N(0)$ are useful in representing the effect of process variability in our model. Two dynamic factors (D_t and K_{amb}), which reflect variations in I/O workload and operating environment, can be combined into the effective dwell time (D_t^{eff}) at 30°because the effect of K_{amb} can be converted to additional dwell time, as explained in Section 3.1.3. Therefore, we do not include K_{amb} in our model. In summary, the estimator $\widehat{N}(12)$ of $N(12)$ can be computed as follow:

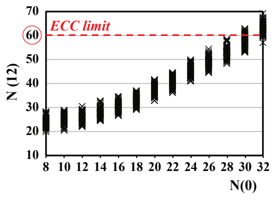
$$\widehat{N}(12) = f(N_{P/E}\ ,\ t_{BERS}\ ,\ N(0)\ ,\ D_t^{eff}) + C_0 \tag{3.1}$$

where the constant term C_0 reflects inborn defects of a manufacturing process.

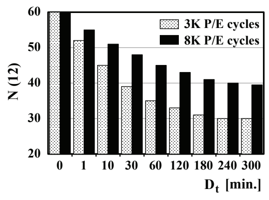
Figures 11, 12 and 13 summarize the key correlation analysis results of 10 candidate variables for $\widehat{N}(12)$. Figures 11(a) and 11(b)



(a) $tBERS$ vs. $N(12)$.

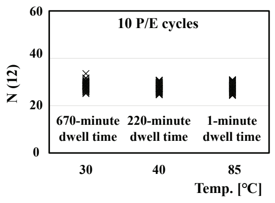


(b) $N(0)$ vs. $N(12)$.

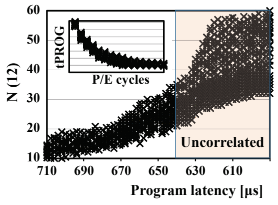


(c) D_t vs. $N(12)$.

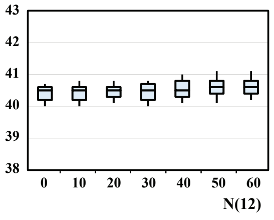
Figure 11: Correlation analysis of candidate variables for $N(12)$.



(a) D_t vs. $N(12)$.

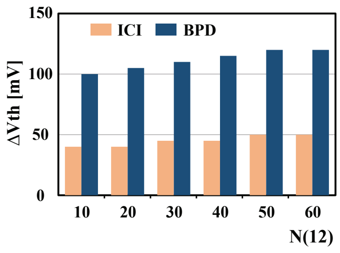


(b) $tPROG$ vs. $N(12)$.

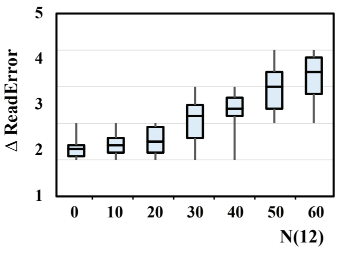


(c) $N(12)$ vs. $tREAD$.

Figure 12: Correlation analysis of candidate variables for $N(12)$.



(a) $N(12)$ vs. BPD & ICI.



(b) $N(12)$ vs. RTN.

Figure 13: Correlation analysis of candidate variables for $N(12)$.

show that $tBERS$ and $N(0)$ have a strong and positive *quasi-linear* correlation with $N(12)$. $tBERS$ and $N(0)$, therefore, are included in Equation (3) as a quasi-linear relationship. Figure 11(c) shows the effect of D_t on $N(12)$. As D_t increases, the wear of a NAND block is improved (i.e., the wear is recovered) by the self-recovery effect. However, its effect was almost saturated when the NAND block experienced the effective dwell time more than 180 minutes regardless of the aging status of the NAND block. Our observations match well with a known fact that the self-recovery effect has a *logarithmic* relationship with the dwell time because it is resulted from degenerating traps (i.e., the detrapping process) due to thermal energy, as explained in Section 3.1.3.

Since we represent the impact of operating temperature by the converted effective dwell time, we validated if the effective dwell time accurately reflects the impact of operating temperature on the wear status of NAND cells. Figure 12(a) compares $N(12)$ values of 10 P/E-cycled NAND blocks under three different (dwell time, temperature) combinations that are converted to the same effective dwell time. As shown in Figure 12(a), $N(12)$ values indicate that three combinations equally affect the NAND cell wear as expected by the Arrhenius's Law.

We have not included $tPROG$, $tREAD$, ICI, BPD, and RTN in Equation (3.1). Figure 12(b) explains why $tPROG$ was not included. Although $tPROG$ is known to decrease as blocks get aged [6], the decreasing rate becomes very small once NAND cells were moderately worn (as shown in the inserted graph). Therefore, unlike in previous studies (e.g., [26]), we find that $tPROG$ is not an efficient indicator for the wear status of a NAND block. For example, when $tPROG$ is 610 us, the corresponding $N(12)$ interval ranges 30 to 58. $tREAD$ was also excluded from Equation (3.1) because it did not change with the NAND cell wear, as shown in Figure 12(c). Figures 13(a) and 13(b) show the impact of noisy factors on $N(12)$. As shown in Figure 13(a), the amount of the V_{th} shift (ΔV_{th}) due to ICI and BPD, which can cause additional bit errors, increases as a NAND block gets aged. However, ICI and BPD factors were excluded from Equation (3) because their effect on $N(12)$ was too small to be practically detected during run time. Figure 13(b) shows the effect of RTN using ΔReadError values which indicate changes in the number of bit errors between successive read operations to the same NAND page. As shown in Figure 13(b), ΔReadError tends to increase as a NAND block gets aged. Although RTN can moderately affect $N(12)$ (e.g., 4 additional error bits), we did not include RTN in our model because computing ΔReadError during run time may incur a substantial performance overhead.

3.2.2 Building Model

Based on the evaluation results, we refine Equation (3) as follows:

$$\widehat{N}(12) = C_0 + \beta_1 \cdot N_{P/E} + \beta_2 \cdot tBERS + \beta_3 \cdot N(0) + \beta_4 \cdot ln(D_t^{eff})$$

Five coefficients, β_1, β_2, β_3, β_4, and C_0, were estimated by the least squares approximation method [64, 27]. Specifically,

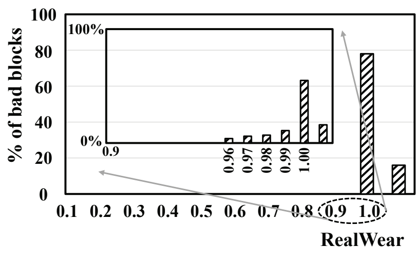
$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ C_0 \end{bmatrix} = (A^T A)^{-1} A^T \begin{bmatrix} \vdots \\ N(12) \\ \vdots \end{bmatrix}$$

$$where \quad A = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ N_{P/E} & t_{BERS} & N(0) & ln\ (D_t^{eff}) & C_0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}.$$

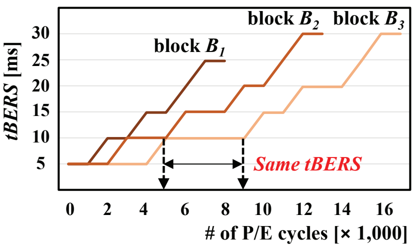
$tBERS$ and $N(0)$, which are almost equally contribute to $N(12)$, have the largest influence on $N(12)$. $N_{P/E}$ complements changes in the wear status when $tBERS$ and $N(0)$ do not change. Furthermore, the impact of D_t^{eff} on $N(12)$ was rather limited up to 30% because the self-recovery effect has a logarithmic relationship with the dwell time as shown in Section 3.2.1.

3.2.3 Model Calibration

From early model validation results, we observed that the constructed model in Equation (4) needs some changes to achieve the high prediction accuracy. Figure 14(a) shows how accurately the $\widehat{N}(12)$ model of Equation (4) can decide if a block is bad or not. First, 28% of test blocks became dead before their ages reach 1.0, which is the false-negative case. This error is critical because data may be stored to a dead block, thus losing data. Second, 16% of test blocks were still alive even after their ages reach 1.0, which is the false-positive



(a) A bad block distribution before calibration.

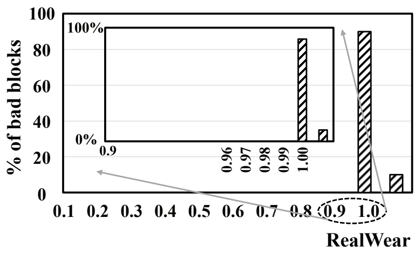


(b) $N_{P/E}$ vs. $tBERS$.

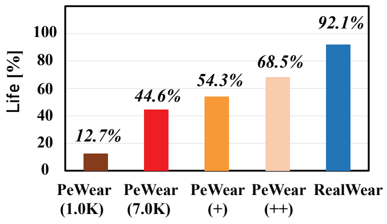
Figure 14: Early model evaluation results.

case. Although less critical, this error wastes the lifetime of mis-aged blocks.

Since it is a critical requirement for a bad-block classifier not to have any false-negative case, we made two modifications to Equation (4) as the final step of building `RealWear`. The first change was required because the measurement resolution of $tBERS$ was too coarse when a block is erased. When a block is erased, an incremental step pulse erase (ISPE) scheme [20] is used. Under the ISPE scheme, a sequence of erase substeps are applied until a block is successfully erased. Since each erase substep typically takes 5 ms, $tBERS$ values can only take multiples of 5 ms. Unfortunately, 5 ms was too coarse to distinguish different wear status of a NAND block. For example, as shown in Figure 14(b), $tBERS$ of the block B_3 does not change at 10 ms until its P/E cycles reach 10K. Although the wear status of the block B_3 was getting worse as its $N_{P/E}^{B_3}$ is increasing from 5K and 10K, $tBERS$ could not properly indicate the changing wear status because of the large timing resolution of a single erase substep. The second



(a) A bad block distribution after calibration.



(b) A comparison of `life`.

Figure 15: Lifetime validation results of `RealWear`.

change was needed because of the RTN factor, which was excluded in the proposed $N(12)$ model. As shown in Figure 13(d), the RTN factor can introduce up to 4 bit errors. When such additional error bits were added when N(0) is measured, the accuracy of $\widehat{N}(12)$ was degraded.

3.3 Validation of `RealWear`

To validate the accuracy of `RealWear`, we performed an extensive validation studies under different operating conditions. We report the key results only in this section, focusing on three validation cases as in Section 3.1: (1) bad block classification, (2) per-sector error variation, and (3) support for temporal/operational variations.

3.3.1 Bad Block Classification

To determine how effectively `RealWear` predicts the lifetime of individual NAND blocks, we evaluated if `RealWear` can accurately decide if a block is live or dead. In our model, when the age of a NAND block reaches 1.0, the block is considered to be dead. Figure 15(a) shows how (physically) bad blocks are distributed over different $age(B)$ values. Compared to the evaluation results of `PeWear`, most of the bad blocks are distributed within a narrow $age(B)$ range. This result is clearly different from that of Figure 8(a) where bad blocks are widely distributed over different $N_{P/E}^{B}$ values. It suggests that `RealWear` can effectively represent the different aging characteristics among NAND blocks and eventually prevent NAND blocks from being wasted.

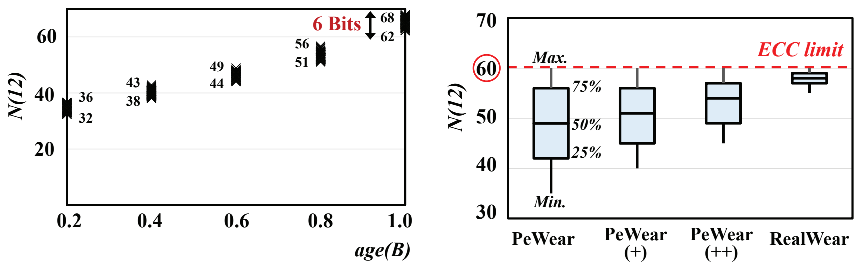
Especially, as shown in the box graph of Figure 15(a), no test blocks become dead before their ages reach 1.0. That is, as long as the block age is less than 1, it can reliably store its data. On the other hand, 8% of the test blocks outlived their maximum age (i.e., 1.0) partly because we over-estimate the block age to account for the RTN factor. Although these blocks could have been used longer, unused lifetimes were short without causing any reliability problem.

To compare the lifetime prediction accuracy of a NAND aging marker `M`, we defined a new metric, the lifetime efficiency (`life`) of `M` as follows:

$$\texttt{life}(\texttt{M}) = \frac{\textit{Predicted total amount of writes}}{\textit{Actual total amount of writes}} \times 100. \tag{3.2}$$

Figure 15(b) compares `life` values of different NAND aging markers. The lifetime efficiency of `RealWear` is close to 92%. That is, `RealWear` utilizes more than 90% of the NAND block lifetime. On the other hand, `life(PeWear)` is very poor. For example, when $N_{P/E}^{Max}$ was set to 7K, `life(PeWear)` was only 44.6%. If $N_{P/E}^{Max}$ was set to 1K (which is a typical $N_{P/E}^{Max}$ value for consumer flash products), `life(PeWear)` significantly drops to 12.7%. When `PeWear` is used, most blocks were actually alive when `PeWear` identified them as dead, thus wasting a significant portion of their lifetimes.

We also examined the `life` values of proposed NAND aging markers in previous studies, `PeWear+` [26] and `PeWear++` [27](that we described in Section 2.8.1). As shown in Figure 15(b), the `life` value of `PeWear+` and `PeWear++` are 54.3% and 68.5%, respectively, which are much lower than that of `RealWear`. Their prediction accuracy is quite limited compared to `RealWear` because both `PeWear+` and `PeWear++` exploit $tPROG$ as a model variable to indicate the wear status of a NAND block. As shown in Section 3.2.1, when a NAND block becomes old (i.e., experiences a large number of P/E cycles), $tPROG$ is not correlated to the wear status of a NAND block. So, `PeWear+` and `PeWear++` cannot avoid the poor prediction accuracy in the high-endurance region. In contrast, `RealWear` can maintain the high prediction accuracy in the entire endurance region because it uses $tBERS$, which maintains a high correlation with the wear status of a NAND block even when the NAND block becomes very old, as a model variable.



(a) $age(B)$ vs. $N(12)$. (b) Per-sector error distributions.

Figure 16: Error variation evaluation results of `RealWear`.

3.3.2 Per-Sector Error Variations

We also evaluated how accurately `RealWear` estimates true $N(12)$ values. As shown in Figure 16(a), blocks with the same age values exhibited almost the same number of bit errors (per 1-KB sector). For example, at the 12-month retention time, error variations between blocks having the same age values were less than 7 bits. This result is quite contrasting to one shown in Figure 9(b) where blocks with the same $N_{P/E}$ showed differences in the number of bit errors up to 25 bits at 8K P/E cycles. Unlike in `PeWear` (as shown in Figure 9(b)), blocks with different block ages are unlikely to map to the same $N(12)$ value. In other words, `RealWear` can distinguish blocks with different $N(12)$ values without mis-classification.

To further validate the accuracy of `RealWear`, we compared the number of bit errors between blocks in different NAND aging markers. Among the blocks with the same `PeWear`, per-sector errors differ by up to 25 bits. On the other hand, when `RealWear` is used, the maxi-

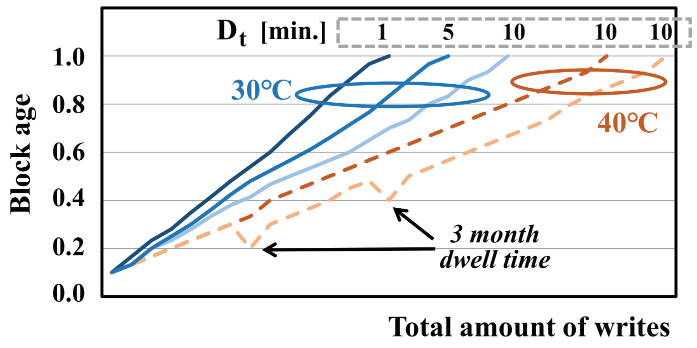


Figure 17: Changes in the $age(B)$ by the self-recovery effect.

mum difference in the per-sector error bits was only 6 bits. Although `PeWear+` and `PeWear++` show smaller per-sector error variations over `PeWear`, their prediction accuracy is significantly worse than that of `RealWear`. As shown in Figure 16(b), error variations between blocks with the same `PeWear+` and `PeWear++` values were up to 20 bits and 15 bits, respectively, which indicate that `PeWear+` and `PeWear++` are not sufficient to exactly distinguish the wear status of individual NAND blocks.

3.3.3 Self-Recovery Effect

Since one of the key differences between `RealWear` and `PeWear` is whether the self-recovery effect is modeled or not, we validated if `RealWear` properly reflects the impact of I/O workload variation and temperature variation on block ages. Figure 17 shows how the block age changes under different variations. As expected, the longer the dwell time, the slower the aging rate. At the same dwell time of 10 minutes, a higher operating temperature significantly slows the aging process. For example, when the operating temperature is changed from 30°to 40°, the lifetime of a block is increased by 20%.

To maximize the self-recovery effect, we emulated the power-off setting of an SSD by applying the 3-month dwell time at 40°. As shown in the bottom dotted line of Figure 17, `RealWear` can reflect this situation as well. Because of the self-recovery effect, blocks got younger as indicated by two drops in the graph.

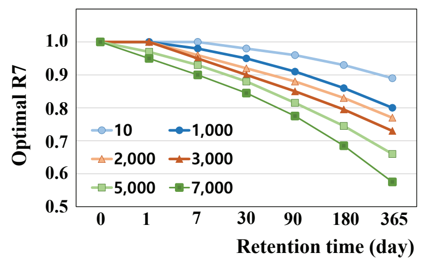
3.4 BoudedRead: Read Retry Mitigation

Based on `RealWear`, we develop a new system-level optimization technique, BoudedRead, which aims at mitigating the number of read retries by directly searching optimal read reference voltages. As described in Section 2.6.2, as a NAND block wears, *tREAD* significantly fluctuates because a failed read should be retried until an optimal reference voltage is found. Since the required number of read retries cannot be known *a priori*, it is generally not possible to put an upper bound on *tREAD*. BoudedRead, on the other hand, exploits the accurate wear status of a NAND block from `RealWear` so that the number of read retries can be bounded under a fixed maximum number. First, with varying retention times, read cycles, and operating temperature, we define an optimal read reference voltages table (ORT) based on extensive NAND characterization studies. Second, we confirm the validity of ORT by investigating `NumRetry` distributions of real 3D TLC flash chips under various operating conditions. Finally, we identify the benefit of BoudedRead by system-level evaluations using various real workload. Furthermore, to demonstrate additional benefits of `RealWear`, we also present two case studies, LongLive and FastCopy, that can improve the lifetime and performance of flash storage systems.

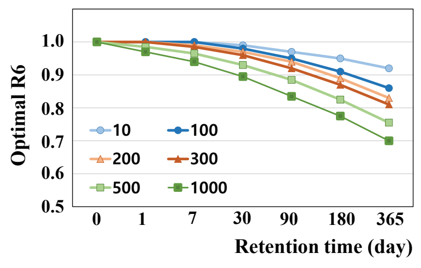
3.4.1 Optimal Read Reference Voltages Table (ORT)

In order to define the optimal read reference voltages table (ORT), we tracked the change in V_{th} distributions and bit errors of 3D TLC flash chips with varying operating conditions such as P/E cycles, retention time at 30°C[8], read cycles, and write/read temperature. Figures 18(a) $\sim$ 18(f) shows representative evaluation results of the optimal V_{ref}'s under various operating conditions. As shown in Section 2.5, since V_{th} distributions of each NAND state is affected and exhibits a different behavior by various factors, individual V_{ref}'s changes a different rate depending on the wear status of NAND flash memory or operating conditions. When the number of P/E cycles increases, the optimal V_{ref} for the lower states (R1 or R2) should be raised upward because V_{th} of the lower states (E or P1) easily moves to the right due to program disturbance. For example, at 1,000 P/E cycles, R1 and R2 should be raised +300mV and +100mV, respectively, to

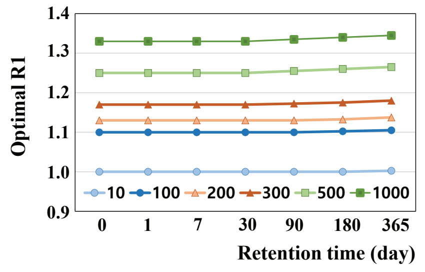
[8]The retention temperature is not considered because it can be converted to 30°C by Arrhenius's relationship.



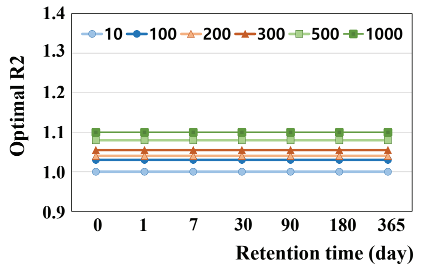
(a) Change in optimal R7.



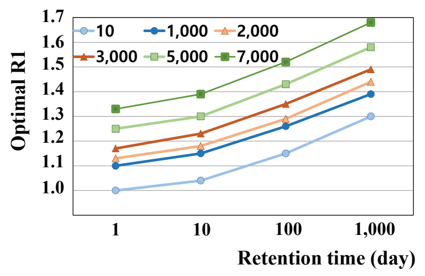
(b) Change in optimal R6.



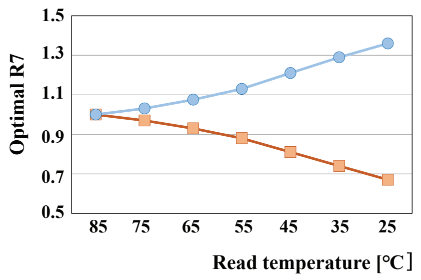
(c) Change in optimal R1.



(d) Change in optimal R2.



(e) Change in optimal R1



(f) Change in optimal R7.

Figure 18: Change in V_{ref} under various operating conditions.

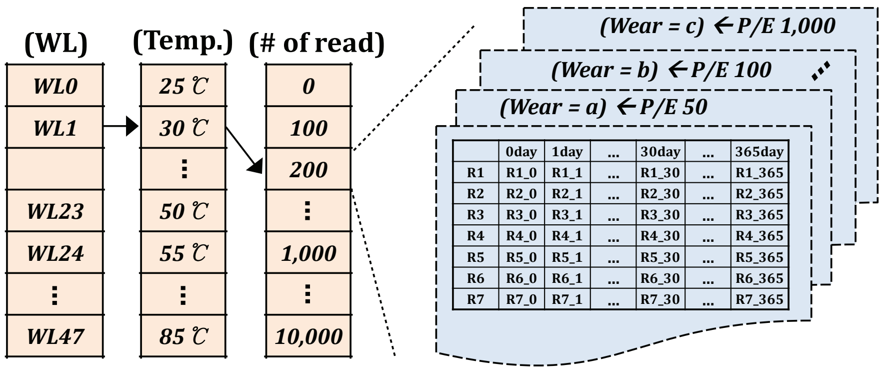
minimize bit errors. On the other hand, the optimal V_{ref} for the higher states (R7 or R6) is hardly affected by P/E cycles. However, when the retention time becomes longer, V_{th} of the higher states (P7 or P6) is susceptible to move to the left due to charge loss, so that the optimal V_{ref} should be shifted downward. As the number of P/E cycles increase, the amount of shift of the optimal V_{ref} is accelerated. Furthermore, higher program states experience a greater amount of the charge loss, and hence show a larger shift compared to lower program states [17]. In additions, when pages in the same block are frequently read, ER-state is likely to be softly programmed by V_{READ} and shifted to the right [10], therefore the optimal value R1 is also raised upward, as shown in Figure 18(e). Figure 18(f) shows that the temperature difference between write and read operations can move the V_{th} distributions of NAND states. Since the current flowing along BL (or channel) and the swing characteristics of select line transistors (SSL transistor in Figure 2) greatly depend on the temperature, read and verify operations are affected by the operating temperature. Therefore, the optimal V_{ref} should be changed considering the temperature when write and read operations.

Based on evaluation results, we constructed optimal read reference table (ORT) by considering the wearing degree of a NAND block, retention-times, read cycles, and operating temperature. Figure 19 shows shows the organizational concept of the optimal read reference voltages table which is a look-up table that stores the optimal optimal V_{ref}.

For the comparison its effect, our ORT was constructed by two versions; One is based on `RealWear` and the other is based on `PeWear`. In addition, to reflect the unique characteristics of 3D NAND flash memory, the WL address entry was added to ORT. Each WL in 3D NAND flash memory has different aging characteristics and bit errors depending on its physical location. For example, the edge WL shows a larger V_{th} shift than the center WL even in the same retention time, so optimal read reference voltages of the edge WL should be quite different from that of the center WL.

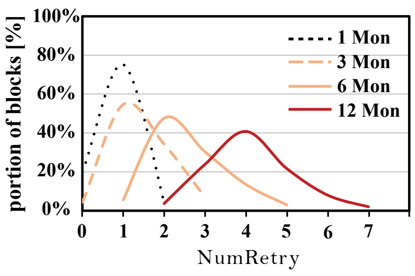
3.4.2 Device-level Evaluation of BoudedRead

BoudedRead is based on an existing read-latency optimization technique [70] that uses a lookup table for finding an optimal read reference voltage. For a fixed retention time requirement (e.g., 12 months), each entry of the ORT table, which is indexed by the number of P/E cycles, stores a pre-defined best sequence of read reference voltages that were decided from an off-line device-level characterization study. When a read reference voltage is needed for a block with x P/E cycles, read reference voltages stored in ORT[x] is tried one by one until a read is successful. Unlike the existing read-latency optimization technique that indexes the ORT table by the number of P/E

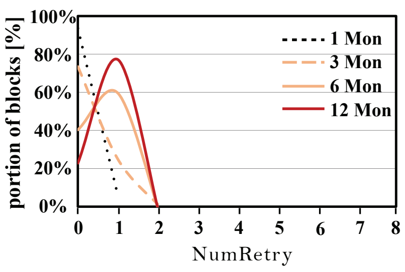


	0day	1day	...	30day	...	365day
R1	R1_0	R1_1	...	R1_30	...	R1_365
R2	R2_0	R2_1	...	R2_30	...	R2_365
R3	R3_0	R3_1	...	R3_30	...	R3_365
R4	R4_0	R4_1	...	R4_30	...	R4_365
R5	R5_0	R5_1	...	R5_30	...	R5_365
R6	R6_0	R6_1	...	R6_30	...	R6_365
R7	R7_0	R7_1	...	R7_30	...	R7_365

Figure 19: Organizational overview of ORT.



(a) Indexed by $N_{P/E}$.



(b) Indexed by `RealWear`.

Figure 20: `NumRetry` distributions.

cycles, BoudedRead uses the block age for indexing the ORT table.

Figure 20 compares how `NumRetry` changes when $N_{P/E}$ and $age(B)$ are used for locating the ORT entry, respectively. As shown in Figure 20(a), when $N_{P/E}$ is used, there are large variations in `NumRetry` because $N_{P/E}$ cannot accurately represent the wear status that strongly affects `NumRetry`. Under the 12-month data retention requirement, up to 7 read retries were needed. Since the minimum number of read retries was 2, $tREAD$ values can significantly fluctuate during run time. On the other hand, as shown in Figure 20(b), BoudedRead exhibits a narrow distribution of `NumRetry` values. Furthermore, no blocks need more than 2 read retries regardless of the data retention requirement.

3.4.3 System-level Evaluation Results

In order to evaluate the benefit of BoudedRead at the storage system level, we have implemented `rFTL`, a `RealWear`-aware FTL, using an open emulation platform for storage systems [65]. Figure 21 shows an overall organization of `rFTL`. `rFTL`, which is based on an existing

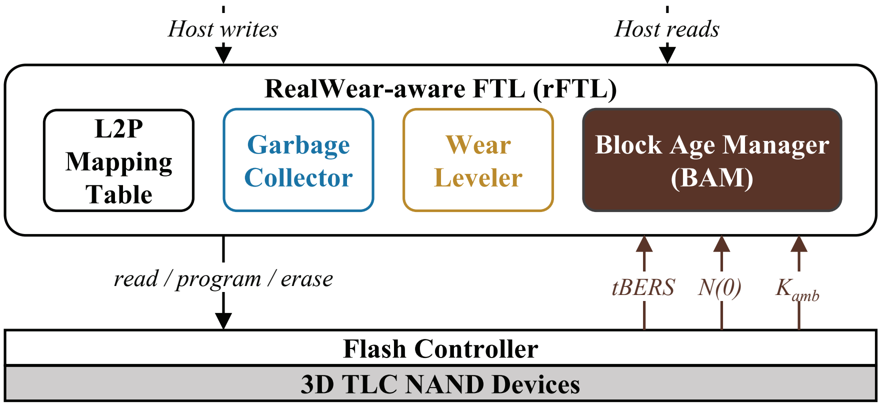


Figure 21: An organizational overview of a `rFTL`.

page-level FTL, employs one `RealWear`-specific module, the block age manager (`BAM`), that is responsible for supporting `RealWear`. The `BAM` module, which is in charge of managing block ages of all the blocks in a flash storage system, keeps track of the following key parameters for each block: $N_{P/E}$, D_t, $tBERS$, $N(0)$ and K_{amb}. Whenever a block B is erased, `BAM` 1) increments $N_{P/E}$ by one, 2) measures the elapsed time $tBERS$ for the block erasure and 3) records the current time for computing D_t. For each block, $N(0)$ is measured only once for the first page of a block. `BAM` reads back the first page of the block B immediately after it is programmed. K_{amb} is periodically updated by an on-board temperature sensor inside an SSD. Based on these model variables of `RealWear`, `BAM` keeps $age(B)$ in the metadata area of the block B. Although `rFTL` needs extra parameters, their overhead is negligible because they are managed at the block granularity. Since a few bytes would be sufficient for each block, per-block parameters can

Table 2: A summary of six workloads.

	Varmail	Fileserver	Proxyserver	Webserver	OLTP	NTRX
Read:Write	40:60	40:60	55:45	85:15	70:30	5:95
WAF	2.7	5.4	1.9	1.2	2.0	2.3

be stored in the OOB area of a block without incurring an extra space overhead. Furthermore, since they are computed as parts of normal operation, no performance overhead exists for `RealWear`.

Since we used a DRAM-based emulated storage system in our experiments, it was not possible to obtain the actual $tBERS$, $N(0)$ and K_{amb} values during run time. Instead, we generated $tBERS$ and $N(0)$ on demand whenever a block is erased. To accurately reflect real $tBERS$ values that are measured during run time, we built $tBERS$ distributions over different $N_{P/E}$ values in advance from our measurement experiments. When a block is erased, $tBERS$ is randomly generated from a pre-constructed $tBERS$ distribution. For example, when $N_{P/E}$ is 4K, $tBERS$ is randomly selected from three values, 5 ms, 10 ms, and 15 ms. $N(0)$ is synthetically generated in a similar fashion as well. We set K_{amb} to 30°. We assumed that K_{amb} does not change in our experiments.

For fast evaluations, we limited the SSD capacity to 32 GB. Our target SSD consists of two buses, each of which has four 3D TLC flash chips. Each chip has 428 blocks and each block is composed of 576 16-KB pages. We set basic flash operation timing parameters for read, program and erase to 80 μs, 700 μs, and 3.5 ms, respectively [14]. For three case studies, we used six workloads: four from Filebench

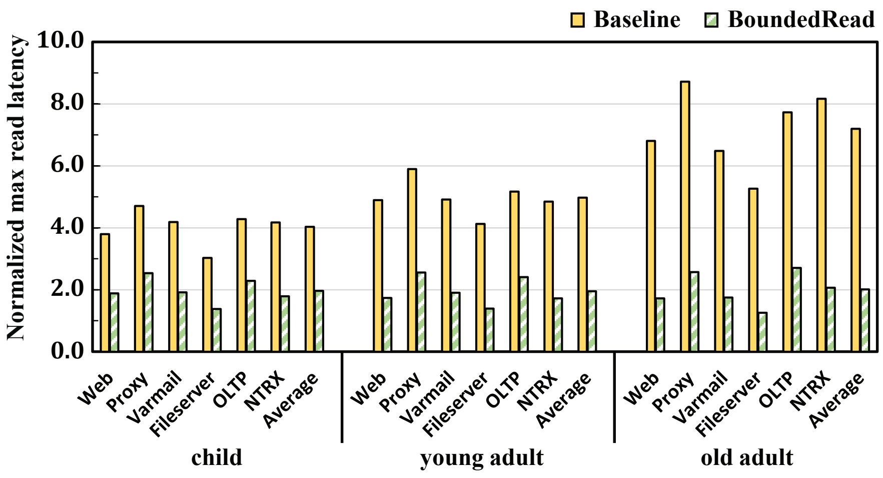


Figure 22: Read latency comparison results.

benchmark tool [79], Varmail, Fileserver, Proxyserver, and Webserver, and two from Sysbench [67], OLTP and NTRX. Table 2 summarizes the key I/O characteristics of each workload.

To better understand the benefit of BoudedRead, we compared the maximum read latency of each workload as shown in Figure 22. All the measurements were normalized to the minimum read latency with no read retry. Furthermore, we ignored queuing delay due to contention between other read requests. In order to simulate different wear status of NAND blocks, we pre-conditioned NAND blocks with three distinct states in the SSD lifetime: a child stage (when only 10% of the SSD lifetime is consumed), a young adult stage (when 20% of the SSD lifetime is passed) and an old adult stage (when 50% of the SSD lifetime is used). In the old adult stage, where more read retries are needed, the maximum read latency of the baseline technique can be 7 times longer on average over when no read retry is needed. In Proxyserver and OLTP, the maximum read latency increases by 9 times because many blocks in these workloads hold very cold pages with long retention time. On the other hand, the maximum read latency of BoudedRead does not increase more than 3 times over the minimum read latency (with no read retry) regardless of block stages or workloads.

3.4.4 Case Study 1: Lifetime Improvement

As a case study, we evaluated how much the storage lifetime can be extended by modifying the existing wear leveler to use `RealWear` instead of `PeWear` in selecting the oldest block. We call this version of `rFTL` by LongLive.

To evaluate the lifetime improvement of LongLive over a baseline FTL, which is based on a `PeWear`-based wear-leveler, we measured the total amount of writes until an SSD reaches the end of its lifetime. Since SSDs in the market use different $N_{P/E}^{Max}$ values, we evaluated LongLive over the baseline FTL with two $N_{P/E}^{Max}$ values: 1K and 8K. As shown in Figure 23, LongLive served on average 63% more writes over the baseline FTL. When $N_{P/E}^{Max}$ was set to 1K, the lifetime of LongLive increases, by 12 times. LongLive achieves a slightly higher improvement ratio on read-intensive workloads (e.g., Webserver). This is because, in read-intensive workloads, NAND blocks tend to have longer D_t's, which make more flash cells to be self recovered.

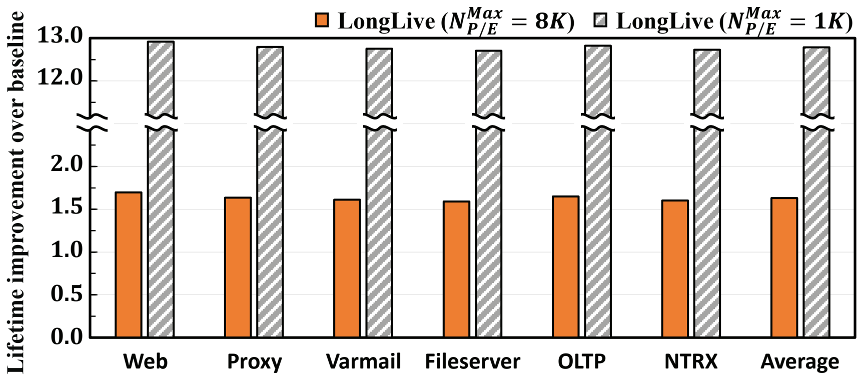


Figure 23: Lifetime Improvement of LongLive.

3.4.5 Case Study 2: GC Overhead Reduction

As another case study, we applied `RealWear` for reducing GC overhead. As well known, GC involves a large number of page copy operations, which may overlap with important host I/O requests. To avoid host-perceived performance degradation from GC, one efficient technique is to use the NAND copyback command. Since the copyback command allows a page to be moved to a different block inside a NAND die without performing off-chip DMA operations, it can significantly reduce the data migration overhead during GC. However, since an error correction operation is skipped during an internal data copy, bit errors are accumulated whenever copyback is used for data migration. If the number of accumulated errors by copyback operations exceeds N_{ECC}, data loss can occur. (In fact, this is the main reason why copyback is rarely used in modern SSDs [68])

To overcome the reliability issue from copyback operations, it is

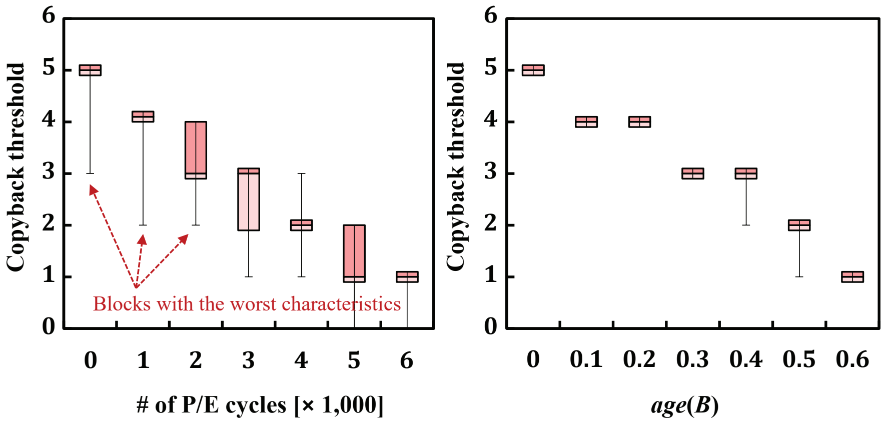


Figure 24: Copyback threshold variations.

important to find the *largest* number of consecutive copyback operations for a given block that does not cause data loss. We call this number the copyback threshold of the block. One recent technique [69] decides the copyback threshold of a block based on the block's P/E cycles (i.e., using `PeWear`). However, as expected, the copyback threshold defined by the number of P/E cycles is not effective, thus missing many opportunities for additional copybacks on many blocks. Figure 24(a) shows that blocks with the same P/E cycles may have quite different copyback threshold values using standard box plots with 25th and 75th percentiles. Since the copyback threshold value is decided by blocks with the worst reliability characteristics, as shown in Figure 24(a), most blocks under-utilize their copyback potentials.

On the other hand, FastCopy, our proposed technique, assigns the copyback threshold value of a block based on its block age, not

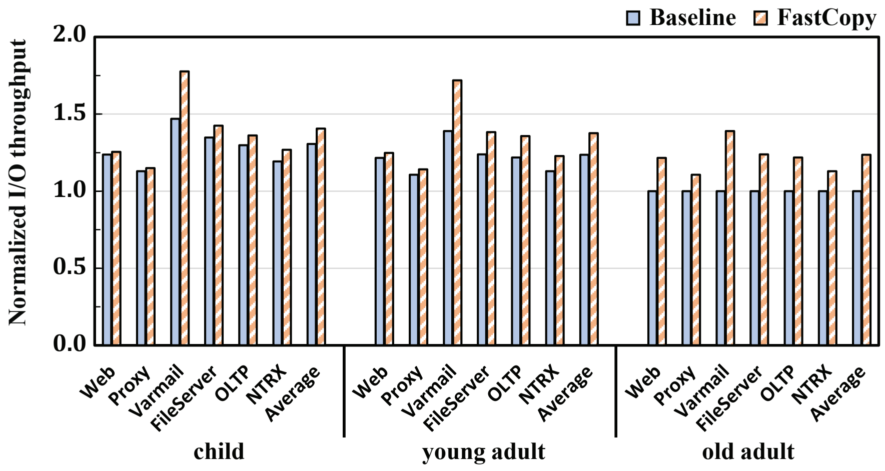


Figure 25: I/O throughput comparison results.

its P/E cycles. As shown in Figure 24(b), copyback thresholds are very narrowly distributed among the blocks with the same age. For example, except for 0.4-old blocks and 0.5-old blocks[9], all the blocks with the same age have a single copyback threshold value. Therefore, more copybacks are possible in FastCopy. For example, FastCopy can use copybacks five times in a row while the `PeWear`-based technique can apply copybacks at most three times.

To evaluate the GC performance improvement by FastCopy, we evaluated various workloads in three distinct block states, a child stage, a young adult stage, and an old adult stage. Figure 25 compares I/O throughput of FastCopy with the `PeWear`-based technique. The results in Figure 25 were normalized over the basic FTL which does not utilize a copyback operation. Overall, as NAND blocks get

[9]The wear status of these blocks were believed to be at the boundary condition between two copyback threshold values.

aged, FastCopy outperforms the baseline FTL by up to 21%. When blocks are in their old-adult stage, the baseline FTL cannot utilize copybacks at all. However, FastCopy continues to exploit copybacks thanks to the higher accuracy of `RealWear`.

Chapter 4

Priority-Aware Suspend/Resume Technique

Typically, the read latency of an SSD can significantly increase when reads and writes are mixed in a given workload. For example, as shown in Figure 26, the read latency can be increased by 23 times over the average read latency. Since many storage applications give a higher priority for reads over other flash operations (e.g., write and erase), such a dramatic increase in the read latency occurs because reads are blocked by writes and block erasures. Since both writes and erases are atomically executed, their long latencies (i.e., 2-3 orders of magnitude longer latencies) significantly increase the read latency.

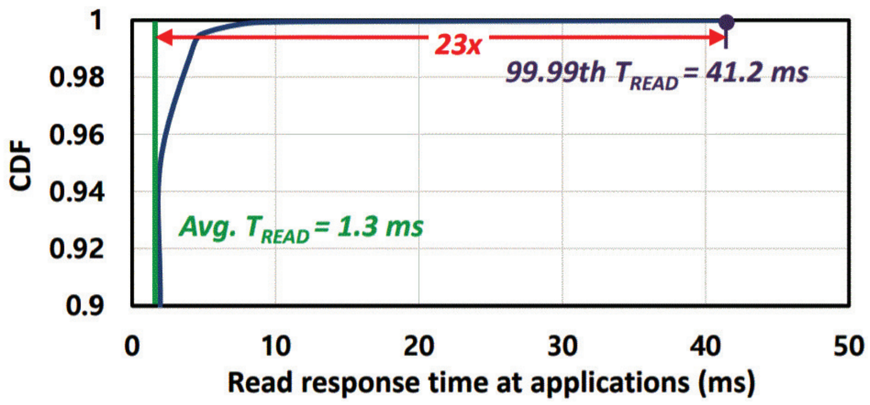


Figure 26: CDF of read latency.

When a read request r is issued from a host system to an SSD, its read latency λ_{READ} (the SSD-level read latency) significantly varies depending on the preceding I/O requests that were issued before r. If there are preceding higher-priority requests, r should wait until their services are finished, call a *conflict* problem.

Figure 27 shows a simple example of conflict problems. We assume that an SSD has 1 channel & 4way structure which can perform 4 flash chips simultaneously, and each flash chip has its own queue[1]. When 4 read requests ($\mathrm{r}_1 \sim \mathrm{r}_4$) are issued from a host, the host searches whether the target data is cached in DRAM or not (①). If the target data is found in DRAM, it is served to the host and read requests are successfully finished (i.e., cache hit). If the target data is not found in DRAM (i.e., cache miss), read requests are sent to the SSD to search the target data from NAND flash memory (②). Each request is transferred through channel (③) and put into the queue of flash chip where the destination block is located (④). After waiting for its own order in the queue, the read request activates a read operation at the destination block in a flash chip (⑤). The output data is transferred to an ECC (Error Correction Code) unit to guarantee the data integrity (⑥) and is returned to the host.

In the above scenario, λ_{READ} (r_i), can be affected by both $tREAD$ (r_i) (the NAND device read latency) and $T_{WAIT}(\mathrm{r}_i)$ (the wait time in

[1]The modern NAND flash-based storage system consists of multiple flash chips which are connected to independent micro controllers. By exploiting multi-channel architecture that can be operated independently and simultaneously, flash storage systems can secure high performance (e.g., high bandwidth).

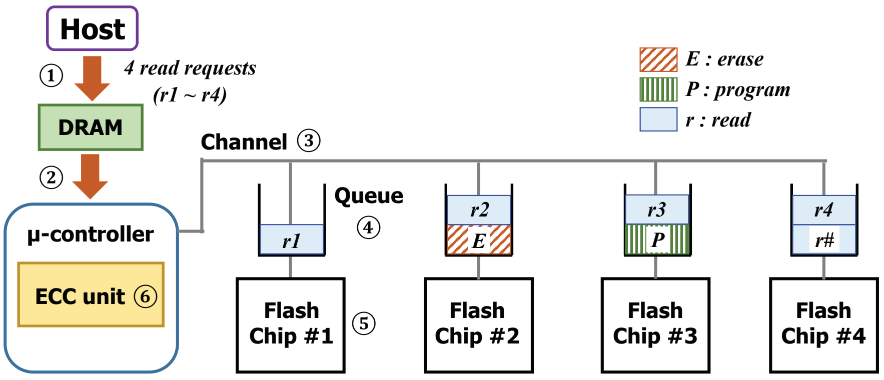


Figure 27: A conflict problem in the flash storage system.

a flash storage system). When r_i arrives at the destination block, the read task cannot be immediately activated because preceding flash operations can be performed in the target flash chip, so that it should wait until a precedent I/O request is finished. If a erase operation is already performed (the case of r_2 in Figure 27), the total waiting time, $T_{WAIT}(r_i)$ increases to $T^0_{WAIT}(r_i) + tBERS$ where $T^0_{WAIT}(r_i)$ is the waiting time without command conflict and $tBERS$ is a erase latency of NAND flash memory.

4.1 Limitation of the Existing Preemption Techniques

To handle read requests in a prioritized fashion over other commands, various preemption techniques are used in modern SSDs [32, 33, 71]. For example, if a read is blocked by a previous program/erase operation at the same flash chip, the existing techniques can suspend the ongoing operation to service the read first.

Unfortunately, the existing command preemption techniques are not sufficient to support read performance differentiation in an efficient fashion. First, the existing command preemption techniques focus on suspending slow ongoing flash commands such as program and erase operations (whose latency is 5.7× and 30.4× longer than that of a read, respectively [72, 73]). When a read operation conflicts with other read operations, no read preemption is supported in existing techniques. However, we observed that read-over-read conflicts among different tasks occur frequently. For example, in one of our test scenarios, 25% of high-priority reads are delayed by lower-priority reads. Furthermore, when such read-over-read conflicts happen at old SSDs, the waiting time for a high-priority read can be 6.1× higher than $tREAD$ due to read retries in servicing the ongoing (lower-priority) read.

Second, the existing techniques do not properly account for the impact of repeated erase preemption on the flash reliability. Figure 28

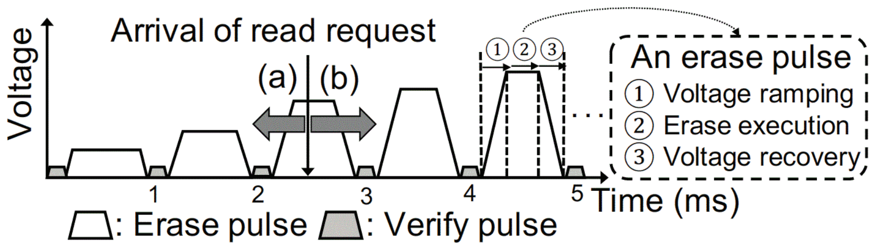
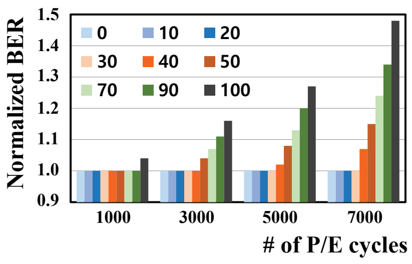


Figure 28: Erase operation in flash memory: Incremental step pulse erase.

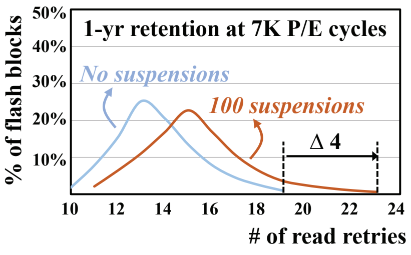
illustrates how an erase operation is performed in NAND flash memory. To perform a NAND block erase, the incremental step pulse erasing scheme is a standard feature in modern SSDs [89]. Instead of utilizing a single, very high voltage pulse (e.g., 14V) for an erase, which has negative impact on NAND lifetime, this scheme performs an erase operation with several, discrete pulses (typically 5 or fewer), and each pulse has a higher nominal voltage than the previous one. By verifying the set of erased cells between erase pulses and by applying higher voltage pulses to cells that are not erased yet, this scheme minimizes damage on NAND cells. A single erase pulse consists of the following 3 stages: ① voltage ramping stage in which the erase pulse reaches the desired voltage, ② erase execution stage during which the voltage is stabilized and maintained, and ③ voltage recovery stage in which the erase voltage is reset for the erase-verify operation. To suspend an ongoing erase (or resume a suspended erase), it is necessary to discharge (or charge) a high operating voltage (> 20V) during intervals represented as ① and ③ that is determined in the flash design stage. Therefore, NAND cells suffer from additional stress during the erase suspension and resumption [33] during charging and discharging a high erase voltage, thus accelerating the wear of the NAND block.

To evaluate the impact of erase preemption on the wear of a NAND block, we measured the average bit-error rate (BER) of NAND blocks while changing the number of preemptions per erase from 0 to 100.[2] As shown in Figure 29(a), excessive erase preemptions signifi-

[2]When a read always preempts an ongoing erase, more than 100 preemptions



(a) Reliability impact of preemption.



(b) Changes in `NumRetry`.

Figure 29: Impact of an preemption on the block quality.

cantly degrade the wear of a flash block, increasing BER by up to 48% over the block with no suspension. When such reliability degradation occurs to a block, an FTL should be aware of it so that a proper remedy can be made for such blocks. However, no existing technique considers the reliability degradation due to erase preemption.

Furthermore, excessive suspensions can increase `NumRetry` and cause a wider `NumRetry` variance between flash blocks. As shown in Figure 29(b), when blocks are preempted 100 times by reads, the average and maximum values of `NumRetry` can increase by 2.8 and 4, respectively, over when there is no erase preemption. Note that the range of `NumRetry` is also increased from 9 to 12, which exacerbates the $tREAD$ fluctuation. Unless the reliability degradation from command preemption is properly managed, it becomes more difficult to differentiate read performance.

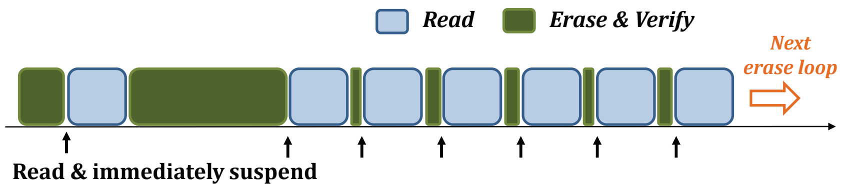
to a single erase can occur since 1) the erase latency is an order of magnitude longer than the read latency, and 2) multiple tasks continuously issue read requests.

4.2 Priority-Aware Preemption: `pSR`

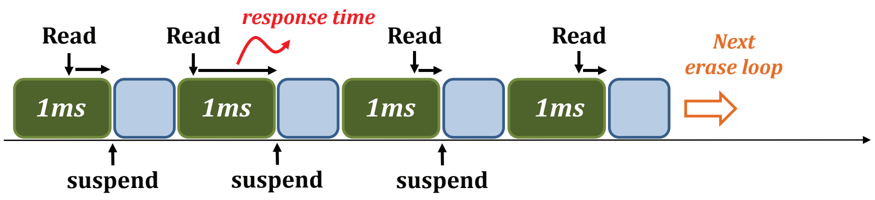
The goal of priority-aware preemption, `pSR`, is to ensure that higher-priority read commands preempt flash chip as much as possible, and to minimize side effects by excessive preemptions as described in Section 4.1 To achieve this goal, `pSR` module adaptively decides if an ongoing operation would be preempted or not when a read is requested. We consider three conflict cases between a read and an ongoing operation: 1) a read with an erase (rCe), 2) a read with a write (rCw), and 3) a read with a read (rCr).

4.2.1 Read-Erase Conflict: rCe

In the rCe case, `pSR` applies different decisions based on the read priority. When the read priority is high, the `pSR` immediately suspends the current erase step, which is similar to the existing preemption technique as shown in Figure 30 (a). The reliability degradation by excessive immediate suspensions is automatically tracked later when the `RealWear` updates $tBERS$ and $N(0)$. On the other hand, when the read priority is not high, the `pSR` in principle delays the erase suspension until the current erase sub-step (i.e., safe point) is finished as shown in Figure 30 (b). Since the preemption is deferred at a safe point, the number of preemption during erase operation is limited, so unfavorable reliability degradation can be efficiently avoided. Figure 29(a) indicates that up to 30 suspensions have no impact on the flash reliability. When a block B is erased, we set its suspension share,



(a) Immediate suspend/resume technique



(b) Deferred suspend/resume technique

Figure 30: Overview of preemption techniques.

$s(B)$, to 30 (i.e., the erase can be divided into 30 sub-steps.). Based on the workload requirement, we proportionally share 30 suspensions between medium- and low-priority tasks. In addition, to maximize the efficiency of the erase suspension, the pSR partially allows reads from medium/low-priority tasks to immediately suspend the ongoing erase.

4.2.2 Read-Program Conflict: rCw

In the rCw case, the pSR does not immediately suspend the current write. Instead, it waits until the current ISPP loop for the write is finished. Similar to the erase operation, the incremental step pulse programming (ISPP) scheme is common practice to control the V_{th} distribution in modern NAND flash memory, where a program operation is consists of multiple iterative sub-steps, called ISPP loop [15].

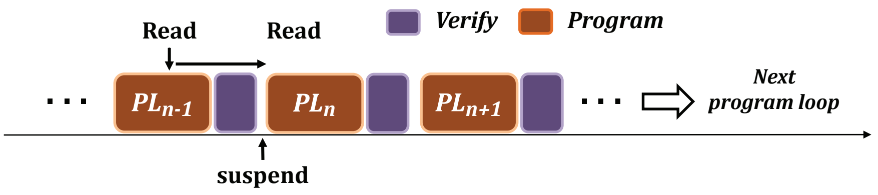


Figure 31: Suspend/resume during a program operation.

One ISPP loop takes 44 μs [74] while the 16-KB (1-page) $tREAD$ is 115 μs [73]. Since the length of one ISPP loop is shorter than the minimum $tREAD$ value (i.e., with no read retry), the benefit of immediate program suspension is minimal while its reliability impact on the block is not negligible. Therefore, as shown in Figure 31, when a read task is requested during program operation, it does not immediately suspend an ongoing program operation but waits until one program loop (i.e., one ISPP lope) is finished.

Furthermore, a read task can be conflicted with critical management tasks triggered by FTL (e.g., garbage collection (GC) to manage NAND flash memory reliably. Typically, the management tasks require a large amount of erase and program operations. When the garbage collection is once triggered, a large amount of program operations for copying valid data and many erase operations for obtaining free blocks. Since management tasks can be urgently required to avoid fatal system failure, the conflict between read requests and management tasks should be processed in a balanced fashion.

Figure 32 shows an example to resolve the conflict problem between a read task and a management task. Since read requests are

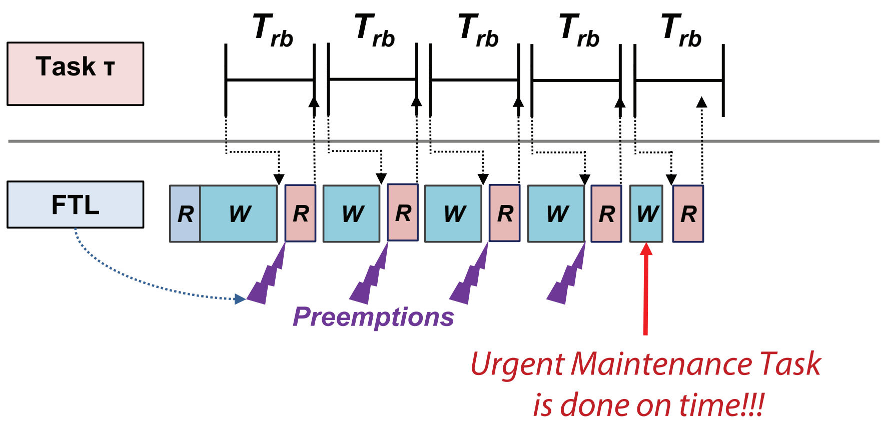


Figure 32: Conflict between read task and management task.

delayed until the atomic inner loop of erase or program operation in a management task is completed, the management task can be completed on time.

4.2.3 Read-Read Conflict: rCr

In the rCr case, the `pSR` takes a similar approach as in the rCe case. When the read priority is high, the `pSR` immediately suspends the current read (unless the priority of the current read is also high). To reflect the impact of read suspension on the block quality, the `pSR` increments the read counts[3] of the target block whenever an immediate read suspension occurs. For reads from the medium-priority task, the `pSR` suspends the low-priority read only when the ongoing read needs a read retry. By delaying the read suspension until a read retry

[3]As the read counts increases, the target NAND block suffers from read disturbance, therefore bit errors of the target block increase [10].

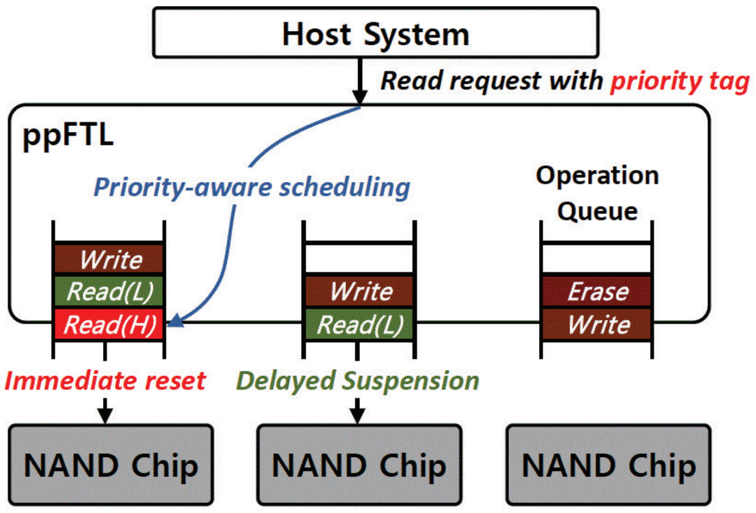


Figure 33: Overview of `paFTL`.

step is finished, the read counts of target block is not affected.

4.3 Evaluation Results

In order to take full advantage of new suspend/resume technique and its policy, we designed a new FTL, `paFTL`, which adaptively selects either the immediate preemption or the delayed suspension/resume depending on the priority of a read request. As illustrated in Figure 33, for an incoming read, `paFTL` checks if the read request has a high priority or not. For a high-priority read, `paFTL` immediately stops the current program (or erase) if the read request needs to access the same NAND plane as the current operation. For a normal-priority read, `paFTL` waits for the completion of the current program (or erase) loop. If the quota (i.e., the number of suspend/resume that does not

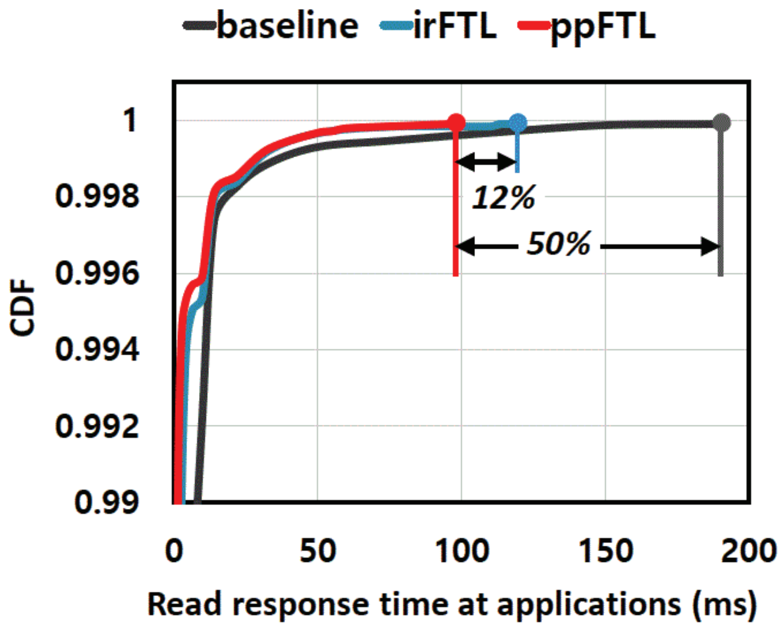


Figure 34: Evaluation results of `paFTL`.

cause the degradation of flash reliability) is enough when considering future workload, medium-priority or low-priority read tasks can have a chance to immediately preempt, thus improving the read performance.

To evaluate the effectiveness of the proposed technique, we compared our `paFTL` with two different FTLs, baseline and `irFTL`. Baseline, which assumes that both program and erase are atomic operations, includes many FTL features for minimizing the read latency such as preemptive garbage collection [90] and out-of-order I/O scheduling [91]. `irFTL`, which is optimized for the read latency, always use the immediate reset for read requests. As shown in Figure 34 `paFTL` reduces 99.99th read latency by 50% over baseline. Moreover, `paFTL` improves the read performance by 12% even over `irFTL` with a negligible P/E increase while avoiding unnecessary immediate suspensions.

Chapter 5

Lock Based Data Sanitization Technique

5.1 Motivation

In this section, we explore the data versioning problem in NAND flash-based storage systems. Furthermore, we also examine why the existing data sanitization techniques cannot be practical in real storage systems.

5.1.1 Data Versioning Problem

Due to the unique nature of NAND flash memory called out-of-place update that is different from magnetic disk storage, when a file system deletes or updates a file, multiple versions of old data of the file can remain in the storage system as the FTL always writes new data of the file in new physical pages. In this paper, we call this issue the *data versioning problem*. To better understand the data versioning problem in a flash-based storage system, we empirically measure how many invalid versions of a file exist in a flash-based storage system throughout the lifetime of the file under varying storage workload characteristics.

Version Trace Tool. We use a custom I/O tracing environment, VerTrace, that integrates an existing I/O profiling tool, IOPro [75], and an open storage emulation platform, FlashBench [76]. VerTrace annotates each physical page with 1) the name of the file to which it belongs and 2) the creation time of the file. VerTrace uses the MD5 hash function [77] to efficiently manage per-page annotation information, which is passed to the emulated storage model of FlashBench via an extended block I/O interface. We extend the emulated storage model of FlashBench to support a logger module that keeps track of the number $N_{valid}^{page}(f, t)$ of valid pages and the number $N_{invalid}^{page}(f, t)$ of invalid pages for a file f at time t. VerTrace works with the ext4 file system [78].

Benchmark Traces and Settings. We use three benchmark traces, Mobile, MailServer, and DBServer, each of which mimics the I/O activity in an Android smartphone, a mail server, and a database server, respectively. For faster evaluation, we limit the maximum capacity of the emulated storage to 16 GiB. To avoid potential start-up bias of simulations and focus on steady-state behavior, each evaluation runs until the total written data size exceeds 64 GiB after we initially fill 75% of the storage capacity.

Metrics. The main goal of our evaluation is to identify 1) how many invalid versions of a file exist, and 2) for how long these invalid versions remain inside the storage device. To evaluate the impact of the file access pattern on data versioning, we classify files into two types depending on their write patterns. We call a file f a *uni-version (UV)* file if the snapshot (i.e., contents) of f at time t is a subset of the snapshot of f at time $(t+1)$. For example, if f is an append-only file or a write-once file, f is a UV file. If f is not a UV file, we call f a *multi-version (MV)* file. For example, if the file system deletes or overwrites f, f is not a UV file.

To quantize the data versioning behavior of a file, we use two metrics. First, we define the *version amplification factor (VAF)* of a file f as follows:

$$VAF(f) = \max_{t \in I}\{N^{page}_{invalid}(f,t)\} / \max_{t \in I}\{N^{page}_{valid}(f,t)\}$$

where I represents the entire execution time of the workload. The higher the $VAF(f)$ of a file f, the higher the number of invalid versions are present in the flash chips, which makes the storage system more vulnerable to malicious access. Intuitively, the $VAF(f)$ of a UV file f would be '0'. If f is an MV file, its $VAF(f)$ significantly varies depending on the access pattern of f, which reveals the amount of updated or deleted data of f remaining stale inside flash chips.

Second, we measure the total length $T_{insecure}(f)$ of insecure time intervals of a file f. We define that a file f is insecure at time t, if $N^{page}_{invalid}(f,t) > 0$. The longer the $T_{insecure}(f)$ of a file f, the more likely an adversary can recover an old version of file f. In a real system, $T_{insecure}(f)$ highly depends not only on the access pattern of f, but also on the system idle time. For example, $T_{insecure}(f)$ may be extremely long if there is a huge time gap between when a user deletes file f and when the user issues a sufficient number of writes to invoke GC (which physically erases the deleted data). Since the system idle time significantly varies depending on the user, which is difficult to model, we use *logical* time that increments by 1 for each 4-KiB host write.

Analysis Results. Table 3 summarizes three interesting observations about the data versioning behavior of the three benchmark traces. We calculate the average and maximum VAF and $T_{insecure}$ values of all the created files in each trace execution. $T_{insecure}$ values are normalized to the total number of writes needed to fill the entire capacity of an SSD.[1]

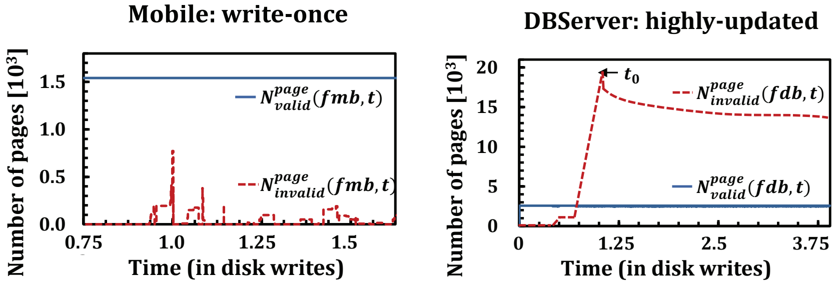
First, the $VAF(f)$ of a file f can be quite high (e.g., 7.8) when file f is heavily updated, as seen for MV files in DBServer. Even if a file is not deleted, such files with high VAF values can pose security vulnerabilities unless their old versions are properly sanitized.

Second, even uni-version files with no updates can have a large number of invalid versions as seen for Mobile (1.5 VAF value) and MailServer (1.0 VAF value). Since UV files do not update their own data, these invalid versions are the result of the extra copy operations

Table 3: A summary of our data versioning evaluations.

Workload	Uni-version (UV) files				Multi-version (MV) files			
	$VAF(f)$		$T_{insecure}(f)$		$VAF(f)$		$T_{insecure}(f)$	
	avg.	max.	avg.	max.	avg.	max.	avg.	max
Mobile	0.24	1.5	2.0×10^{-2}	0.43	1.0	2.0	0.41	2.3
MailServer	0.22	1.0	2.1×10^{-2}	1.7	0.93	2.4	0.50	2.5
DBServer	4.8×10^{-3}	0.24	0.52	2.6	3.2	7.8	3.5	3.5

[1]If $T_{insecure}(f) = 1.0$, it indicates that invalid pages of a file f exist while the total capacity of a disk is written.



(a) f_{mb}: a UV file in Mobile. (b) f_{db}: an MV file in DBServer.

Figure 35: Data versioning under different write patterns.

during GC invocations.[2]

Third, $T_{insecure}$ values are quite large in UV files as well as MV files. For example, the average and maximum $T_{insecure}$ values of MV files in DBServer are 3.5, which indicates that most of the MV files have one or more invalid versions for a very long time (while the host system performs 3.5 disk writes). Note that even UV files are insecure for a significant amount of time (e.g., in MailServer and DBServer) because GC victim blocks are erased lazily as explained in Section 5.2.4.

To highlight different data versioning patterns, we select two files, f_{mb} (from Mobile) and f_{db} (from DBServer), and compare their $N^{page}_{valid}(f,t)$ and $N^{page}_{invalid}(f,t)$ timeplots. Figure 35(a) shows the timeplot for the append-only file f_{mb}. Even though f_{mb} is a UV file with no updates, there are a fair number of invalid pages (up to 800 pages) due

[2]As explained in Section 2.7, the GC process invalidates all the valid pages in a victim block, after copying these valid pages' data to other free pages. Until the victim block is erased, a UV file can have invalid pages stored in the victim block. Since a block is erased lazily (to minimize the negative reliability impact of erase operations; see Section 5.2.4), it may take a long time for invalid pages of the victim block to be physically erased.

to GC invocations. Figure 35(b) shows the data versioning pattern of the heavily-updated MV file f_{db}. Before the GC process is invoked at time t_0, $N^{page}_{invalid}(f_{db}, t)$ rapidly increases due to frequent updates while $N^{page}_{valid}(f_{db}, t)$ remains constant. Although $N^{page}_{invalid}(f_{db}, t)$ tends to decrease after t_0 because invalid pages are erased by subsequent GC invocations, the rate of decrease in $N^{page}_{invalid}(f_{db}, t)$ is quite slow because 1) invalid pages of f_{db} are scattered to many blocks and 2) more invalid pages are generated from continuous updates to f_{db}.

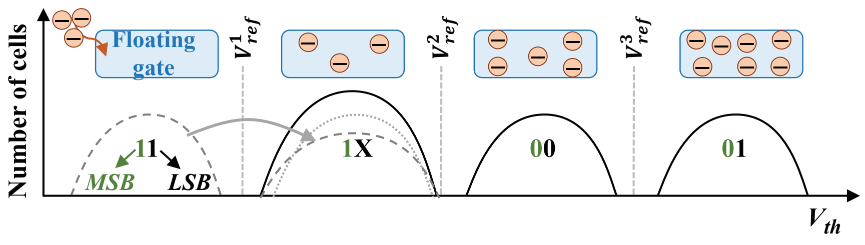
Based on our empirical study, we identify two key requirements that a desired data sanitization technique should meet. First, the technique should support *per-page sanitization.* As mentioned above, the invalid pages of a file (e.g., f_{db} of DBServer) can be stored in the same block with other files' valid pages. If it is not possible to individually sanitize an invalid page, all the valid pages stored in the same block should be copied to other block(s) to sanitize the invalid page, which incurs significant performance and lifetime overheads.

Second, the effect of the page-level sanitization technique should be *immediate.* Our study shows that a single file may have a large number of invalid pages for a long time, even when the host system does not delete the file. If the storage system does not support immediate sanitization of invalid data (i.e., if it sanitizes a file only when the host system deletes the file), the storage system should keep track of all the physical pages used for each file because the FTL uses multiple physical pages to store the data of a single logical page when the file system updates the logical page or the GC process moves the data. Doing so not only requires additional I/O interfaces to send file-system information, but also significantly increases the metadata maintained inside the storage system.

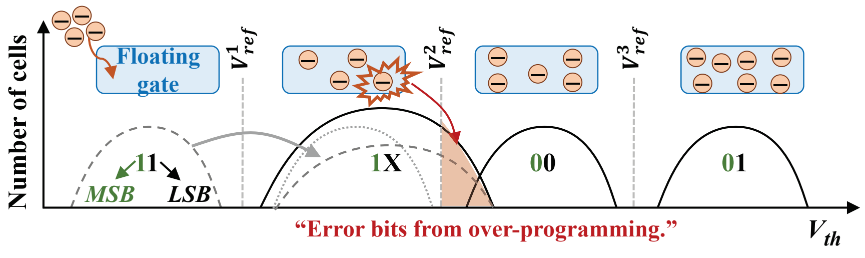
5.1.2 Reprogram-Based Data Sanitization

Most existing data-sanitization techniques (e.g., [34, 35, 37, 39, 36, 38]) destroy the stored data by intentionally changing the V_{th} of flash cells. For example, the scrubbing technique [37] increases the V_{th} of all flash cells in a WL so that the V_{th} distributions of different states are mixed together, which makes it impossible to identify the original data. However, in MLC NAND flash memory, this technique is not easy to adopt because it incurs a significant performance overhead to move valid pages out of the WL to be scrubbed. For example, consider TLC NAND flash memory that stores three pages (i.e., LSB (least-significant bit), CSB (central-significant bit), and MSB (most-significant bit) pages) in each WL. To sanitize one of the three pages, other valid page(s) should be moved to other free page(s). To do so, two extra read operations and two extra write operations may be needed.

To overcome the performance overhead of the scrubbing technique, prior work proposes a more efficient reprogram-based sanitization technique for MLC NAND flash memory [39]. Unlike scrubbing, this technique uses the one-shot programming scheme. The key claim of the one-shot reprogramming (OSR) technique is that, even in MLC NAND flash memory, a page can be safely destroyed by using a low



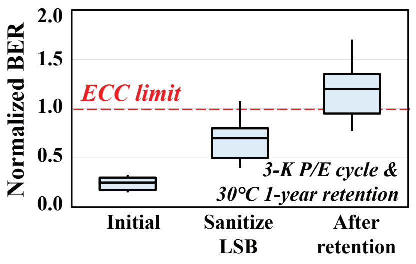
(a) Intended (normal) program operation.



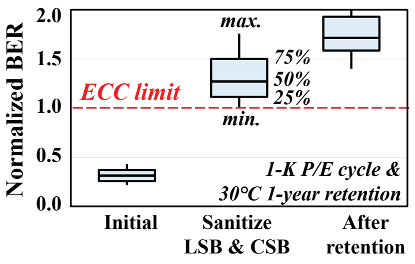
(b) Abnormal over-programming.

Figure 36: V_{th} distributions of 2^m-state NAND flash memory.

program voltage, while the other page in the same WL does not suffer a critical reliability damage. Since no page copy is required during the reprogram process, OSR can achieve zero-copy overhead. Figure 36(a) illustrates an example case where OSR works as intended. In this example, the LSB page is to be sanitized while the MSB page is to remain as a valid page. To destroy the LSB page, OSR moves the V_{th} levels of the E-state cells (i.e., '11') to the right so that they are overlapped with the V_{th} levels of the P1-state cells (i.e., '10'). By doing so, the original LSB page cannot be correctly read with V_{ref}^{R1}, which effectively sanitizes the LSB page. Note that, in this example, the MSB page is not affected at all, and can be reliably read with V_{ref}^{R2}.



(a) A bad block distribution in W(2).



(b) A bad block distribution in W(10).

Figure 37: Changes in RBER of flash pages under OSR.

Although many flash cells would behave as shown in Figure 36(a), a significant number of flash cells may misbehave under OSR due to over-programming [80, 19]. Figure 36(b) shows such a case where OSR moves the V_{th} levels of the E-state cells too far (to the right) such that some of them overlap with the P2-state cells (i.e., '00'). When such over-programming errors occur, the MSB page cannot be reliably read with V_{ref}^{R2}, because the MSB values of over-programmed cells (which should be '1') are recognized as '0'.

To identify how the reliability of multi-level cell NAND flash memory is affected by over-programming errors, we measure how *raw bit-error rate (RBER)* changes under OSR in real 3D MLC and TLC flash memory chips. Figures 37(a) and 37(b) show RBER of flash pages in MLC and TLC NAND flash memory, respectively, under three different conditions: 1) right after (i.e., zero retention time) programming all the pages on a WL (left-most box plot), 2) right after sanitizing the other page(s) on the same WL using OSR (middle box plot), and 3) after a 1-year retention time (right-most box plot).[3] In MLC NAND flash memory, as shown in Figure 37(a), after the LSB page is sanitized by the OSR, 7.4% of the RBER values in MSB pages exceed the ECC limit, making these *valid* MSB pages *unreadable*. As explained in Figure 36(b), a large portion of the extra bit errors on MSB pages are over-programming errors due to an excessive V_{th} shift during OSR. To minimize over-programming errors, one solution might be to fine-tune the OSR parameters separately for each WL. However, since the exact amount of V_{th} shift under OSR can significantly vary depending on each WL's process-variation related characteristics (e.g., the physical location of each WL on the chip), customizing the OSR parameters separately for each WL is extremely difficult. Figure 37(b) shows that the impact of OSR on the reliability of TLC NAND flash memory is even higher than that in MLC because TLC NAND flash memory has a narrower V_{th} margin between adjacent V_{th} states. For example, when both the LSB page and the CSB page are sanitized, *all* of the MSB pages become unreadable due to their high RBER values.

We also observe that the RBER value of a valid page after sanitizing other pages on the same WL greatly increases if the valid page experiences a long retention time. When we measure RBER with the industry standard requirement (i.e., 1-year retention requirement at 30°C [52]), most of the MSB pages in 3D MLC NAND flash memory and all of the MSB pages in 3D TLC NAND flash memory, cannot be

[3]All measurements are normalized to the maximum RBER value below which an ECC module can correct errors.

reliably read. As shown in the right-most box plots in Figures 37(a) and (b), such valid pages' RBER values can be more than 1.5 times over the ECC limit (correction capability). Our evaluation results, therefore, clearly show that OSR is not a reliable solution for data sanitization in modern flash-based storage systems, since it leads to destruction of valid data that is not supposed to be sanitized.

5.2 Evanesco: Lock-Based Sanitization

We propose `evanesco`, a new technique to sanitize a physical page *immediately* without negatively affecting the reliability of other stored data. In this section, we present our threat model and introduce two new flash commands, `pLock` and `bLock`, which enable `evanesco` to sanitize a page and a block, respectively, at low cost.

5.2.1 Threat Model

We assume a very capable attacker who possesses all the required skills to recover deleted information from a modern flash-based storage system by reading flash cells through the interfaces to the NAND flash chip. The attacker can gain physical access to a full system, including a processor, DRAM, and a flash-based storage device. The attacker can deconstruct the storage device (e.g., de-soldering flash chips) without any damage on stored data, and directly access the raw flash chips through all known flash interface commands while bypassing the file system and the FTL.

If the storage system is encrypted, the attacker can obtain any necessary passwords and encryption keys to decrypt stored data. The attacker has comprehensive knowledge about the implemented encryption scheme, and can perform sophisticated attacks (e.g., a cold boot attack [45]) to retrieve the secret keys. The attacker can issue a court order or legal subpoena that obliges a user to reveal the used password.

We assume that the attacker cannot directly probe raw flash memory cells to retrieve stored data using highly sophisticated tools such as a scanning electron microscope (SEM) [49]. Although such attempts were successful in very early 2D SLC NAND flash memory (with 350-nm technology node) [81], to our knowledge, they are very difficult in practice for modern 3D NAND flash memory due to several reasons. First, in order to have visual access to flash memory cells, the attacker needs to deprocess several tens of layers of flash chips. Since memory cells are organized in a cubic form in 3D NAND flash memory, it likely requires extreme effort to expose individual cells without damaging their electrical status (i.e., stored data). Second, the technology node of modern 3D NAND flash memory already has reached 20 nm [82], and manufacturers employ aggressive multi-level cell techniques (e.g., TLC and QLC NAND flash memory) to maximize storage density. To directly read stored data from such memory cells, an SEM should support an extremely high resolution to distinguish the contrast between cells with different values. We are not aware of any demonstration that allows an attacker to directly probe raw memory cells in modern 3D NAND flash memory. We speculate that, if such techniques exist, they require extremely expensive equipment and infrastructure.

5.2.2 Approach Overview

The key insight of `evanesco` is that if we could block access to a flash page by controlling the on-chip access permission (AP) flag of the page, we could achieve the same effect of data sanitization without physically destroying the data stored in the page. In `evanesco`, we support two types of AP flags inside a flash chip, a page-level AP (pAP) flag and a block-level AP (bAP) flag, controlled by two new flash commands, `pageLock` (`pLock`) and `blockLock` (`bLock`), respectively. The `pLock <ppn>` command locks physical page number `ppn` by setting the pAP flag of `ppn` to the *disabled* state. The `bLock <pbn>` command locks physical block number `pbn` by setting the bAP flag of `pbn` to the *disabled* state. When a page or a block of a flash chip is locked by `pLock` or `bLock`, respectively, the flash chip blocks any access to it. Since the pAP flag (or bAP flag) of the locked page (or locked block) can be reset to its default *enabled* state only after the block with the locked page (or the locked block itself) is erased (i.e., no *unlock* command exists for locked pages or blocks), once a page or a block is locked, its data becomes permanently inaccessible. When the locked page or block is re-enabled, its data already has been destroyed by an erase operation.

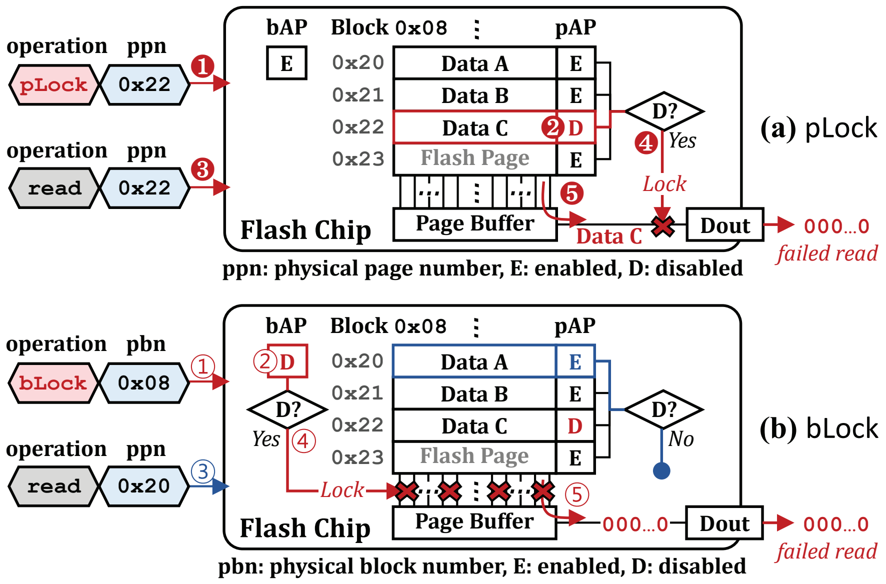


Figure 38: Operational overview of `pLock` and `bLock`.

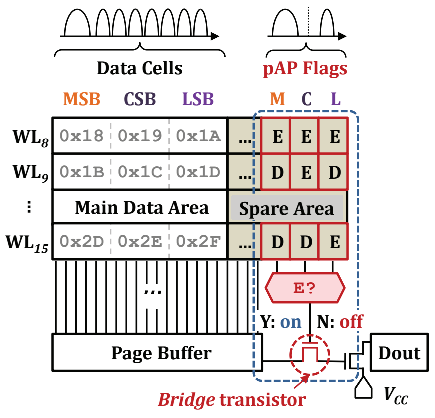
Figures 38(a) and 38(b) illustrate an operational overview of `pLock` and `bLock`, respectively. To sanitize physical page `0x22` (denoted as `PP#0x22`), the `pLock <0x22>` command (❶) sets the pAP flag of `PP#0x22` to `disabled` ('D' in ❷). Future reads to `PP#0x22` (❸) fail because the evanesco-enhanced logic inside the flash chip checks if the pAP flag is enabled (❹) before transferring the page data out from a flash chip (❺). If a target page's pAP flag is enabled (e.g., as for `PP#0x20`), a read request to the page operates as a normal read operation. Similarly, as shown in Figure 38(b), when physical block `0x08` (denoted as `PB#0x08`) needs to be sanitized, the `bLock <0x08>` command (①) sets the bAP flag of `PB#0x08` to `disabled` ('D' in ②). When the bAP flag is disabled, a read to any of the pages in `PB#0x08` fails, including a read to `PP#0x20` (③), because the `evanesco`-enhanced logic first checks the bAP flag (④) before the pAP flag and prevents reading out page data from a disabled block (⑤), regardless of the pAP flag of the target page.

5.2.3 PLock: Page-Level Data Sanitization

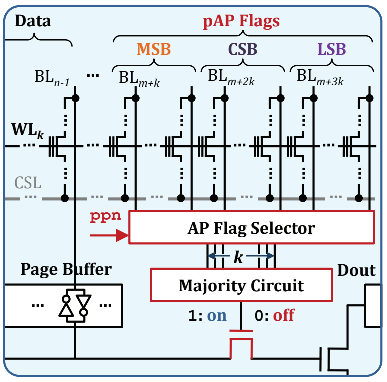
Organizational Overview. Figure 39(a) shows an organizational overview of our `pLock` implementation. In order to exploit the existing flash organization as much as possible, per-page pAP flags are implemented using flash cells available in the spare area[4] (i.e., the OOB (out-of-band) area) of each WL. For example, in the TLC flash memory illustrated in Figure 39(a), three pAP flags are placed in the spare area for the LSB, CSB, and MSB pages of each WL, respectively. Since the spare area is read concurrently with the main data area (i.e., a page), no special command is needed to read a pAP flag. If the pAP flag of a page is set to `disabled` , the bridge transistor, which connects the page buffer to data-out pins as shown in Figure 39(a), is turned off so that the flash chip outputs all-zero data. Otherwise, the flash chip outputs the requested data from the page.

To implement the pAP flags with spare flash cells, it should be possible to *selectively* program flash cells on the same WL because 1) the pAP flag of a page is set to `disabled` *after* the page is pro-

[4]A flash page consists of a main data area for storing data and a spare area for storing page-specific information such as the logical page address and error-correcting code (ECC) [20] values. A typical 16-KiB page has up to 1 KiB as spare area.



(a) An organizational overview of `pLock`.



(b) An implementation of `pLock`.

Figure 39: `pLock` implementation.

grammed on the main data area and 2) the pAP flag of each page on the same WL is set to disabled at different times. To support such selective cell programming, we exploit the SBPI (self-boost program inhibit) technique [15], which allows flash cells on the same WL to be selectively programmed by choosing different voltage settings for different BLs.[5] For example, when programming `PP#0x19`, we inhibit all three pAP flags on WL_8 with the SBPI technique so that the pAP flags are not programmed, and thus they stay in the default `enabled` state. When `PP#0x19` needs to be sanitized, `pLock` sets only the pAP flag of `PP#0x19` to `disabled` by inhibiting all the data cells on WL_8 as well as the pAP flags of `PP#0x18` and `PP#0x1A`.

[5]When a page in WL_k is programmed, its i-th cell within the page is selectively programmed depending on the value of BL_i. If BL_i is set to '0' (i.e., 0V), the i-th cell is programmed. If BL_i is set to '1' (i.e., V_{CC}), the i-th cell is not programmed (i.e., it is inhibited) because there is an insufficient voltage difference between the cell's floating gate and the substrate due to the channel boosting effect [20, 80, 15, 83]

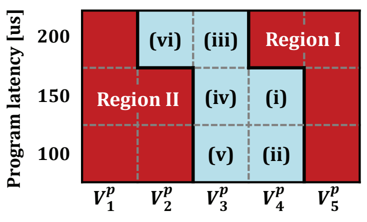
Implementation. To support `pLock` with the organization shown in Figure 39(a), there are two main implementation challenges. First, programming a pAP flag should not cause reliability issues (e.g., due to interference) in the main data area and the spare area of the WL. Although the SBPI technique supports selective programming of flash cells on a WL, inhibited cells might be affected by a high program voltage applied to the WL. Second, a pAP flag should be programmed fast and read reliably. In particular, it should be guaranteed that there is no error in pAP flags under all flash operating conditions (e.g., long retention times, high P/E cycles, and process variability [16]). For example, if a disabled pAP flag value is mistakenly re-enabled after a long retention time, the associated locked page can be accessed again, which is unacceptable.

In our current design, we meet the first requirement by programming a pAP flag using the one-shot programming scheme with a lower program voltage, in addition to the SBPI scheme. Since flash cells for pAP flags need to distinguish between only two discrete states `enabled` and `disabled`, we treat flash cells for pAP flags as SLC cells. Unlike typical TLC data cells which need to store eight different states, SLC flash cells with two states can be reliably programmed with a low program voltage, avoiding a reliability degradation due to the over-programming problem (Section 5.1.2). Furthermore, the one-shot programming scheme reduces the frequency and duration of applying a program voltage to a WL. Therefore, using the SBPI scheme with a low-voltage one-shot programming scheme minimizes the impact of programming a pAP flag on the reliability of the inhibited flash cells. Note that using the one-shot programming scheme also has the benefit of having a relatively short latency.

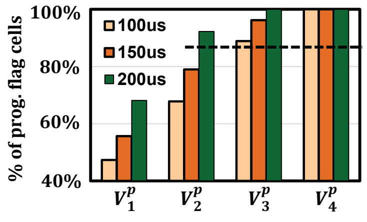
To guarantee error-free management of pAP flags, we employ a simple k-modular redundancy scheme that allocates k flash cells for each pAP flag. As shown in Figure 39(b), the k-bit majority circuit computes the pAP flag value of each page from k flash cells. As we show in the following subsection, with a sufficiently high k, we can manage pAP flags reliably without requiring a complicated ECC module.

Design Space Exploration. To determine good design parameters for our proposed `pLock` implementation in Figure 39, we conduct comprehensive reliability and performance evaluations using 160 state-of-the-art (48-layer) 3D NAND flash chips. To minimize the potential distortions in the evaluation results, for each test scenario, we evenly select 120 test blocks from each chip at different physical block locations, and test all the WLs in each selected block. We test a total of 3,686,400 WLs (11,059,200 pages) to obtain statistically significant experimental results. Using an in-house custom test board, we evaluate various performance and reliability metrics while varying the number of P/E cycles (from 0 to 1,000) and retention-time requirements (from 0 to 5 year). Due to the page limit, we discuss only the key results under worst-case reliability conditions.[6]

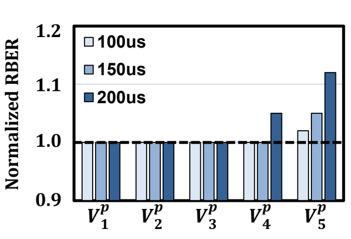
[6]Our test procedure follows the JEDEC standard [52] recommended for commercial-grade flash products.



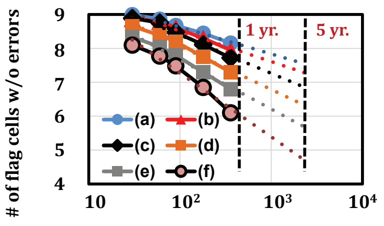
(a) Initial design space $\Psi \times T$.



(b) RBER variations in $\Psi \times T$.



(c) Success rate of programming flag cells.



(d) Retention errors in flag cells when $k = 9$.

Figure 40: Design space exploration results for `pLock`.

In the design shown in Figure 39, there are three key design parameters that we need to decide: 1) program voltage V_{prog}^{pLock} and 2) program latency t_{pLock} used for programming the flag cells, and 3) k, the number of flash cells per pAP flag. To find the best combination of $(V_{prog}^{pLock}, t_{pLock})$, as shown in Figure 40(a), we start from an initial design space $\Psi \times T$ where $\Psi = \{V_1^p, V_2^p, V_3^p, V_4^p, V_5^p\}$ $(V_{i+1}^p - V_i^p = 0.5V)$ and $T = \{100\mu\text{s}, 150\mu\text{s}, 200\mu\text{s}\}$. We define the initial design space $\Psi \times T$ via a preliminary evaluation on the performance and reliability of the `pLock` implementation in Figure 39.

First, we evaluate how the reliability of data cells is affected by a different combination of $(V_{prog}^{pLock}, t_{pLock}) \in \Psi \times T$ for the `pLock` implementation. Although the SBPI technique enables `pLock` to inhibit data cells while programming flag cells, the RBER of data cells can increase due to *program disturbance*.[7] As shown in Figure 40(b), the higher the program voltage or the longer the program latency, the higher the RBER of data cells due to program disturbance during `pLock`. Based on the result shown in Figure 40(b), we exclude four combinations in Region I (Figure 40(a)) from further consideration because they increase the RBER of data cells.

Second, we evaluate if a flag cell can be reliably programmed for a (V_{prog}^{pLock}, t_{pLock}) combination of the remaining design space (i.e., $\Psi \times T-$ Region I). As shown in Figure 40(c), several combinations cannot reliably program a flag flash cell due to low program voltage or short program time. For example, with combination (V_1^p, 100μs), `pLock` can program only 47.3% of flag cells successfully. Based on the result in Figure 40(c), we exclude five combinations in Region II, leaving six candidate combinations, (i) $\sim$ (vi), as shown in Figure 40(a).

As the last step in our design space exploration, we evaluate how the number of retention errors changes when k flag cells are grouped to represent a single pAP flag. We test two retention-time requirements at 30 °C after 1K P/E cycles, 1-year and 5-year retention times, while varying k from 5 to 11. Figure 40(d) shows the evaluation results when $k = 9$ (which we use as the final k value). The number of retention errors is significantly affected by which (V_{prog}^{pLock}, t_{pLock}) combination we use. For example, for the 5-year retention-

[7]Even if all the data cells in a WL are inhibited during `pLock`, the high program voltage applied to the WL can affect the V_{th} levels of the data cells. This undesired phenomenon is called program disturbance [20, 80, 25].

time requirement, combination (vi), (V_2^p, 200 μs), leads to 5 retention errors in 9 flag cells, while combination (i), (V_4^p, 150 μs), leads to at most 2 errors. When combined with the 9-bit majority circuit, combination (vi) cannot guarantee that a pAP flag is correctly managed throughout the required retention time. Out of the six candidate combinations, we select combination (ii), (V_4^p, 100 μs), which meets a high retention-time requirement with the shortest t_{pLock}, as the final design parameter along with 9 flag cells to represent a pAP flag.

5.2.4 BLock: Block-Level Data Sanitization

Need for Block-Level Sanitization. The `pLock` command enables per-page sanitization at low cost, but its performance overhead may become nontrivial if a large number of pages need to be sanitized at the same time. For example, if a user wants to securely delete a 1-GiB file (e.g., a video file) from a flash-based storage system with 16-KiB page size, 65,536 consecutive `pLock` commands are needed, which can introduce significant delay in the flash-based storage system. A block-level data sanitization mechanism could mitigate the performance overhead of a large number of `pLock` commands: a single `bLock` command can sanitize all the pages in a block at once with low latency.

There is an even more fundamental reason to support such a `bLock` command in modern 3D NAND flash memory. In recent 3D NAND flash memory, due to structural characteristics, the reliability of data stored in a block strongly depends on the time gap be-

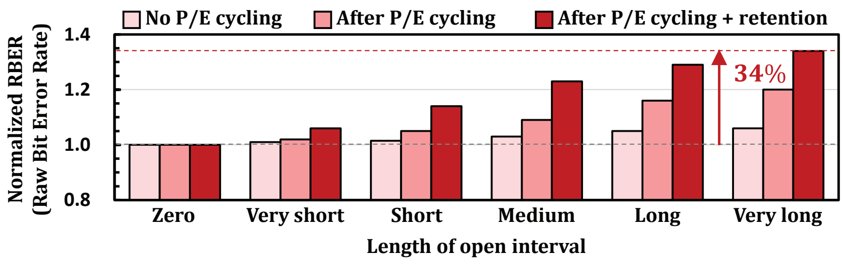


Figure 41: RBER vs. open interval length.

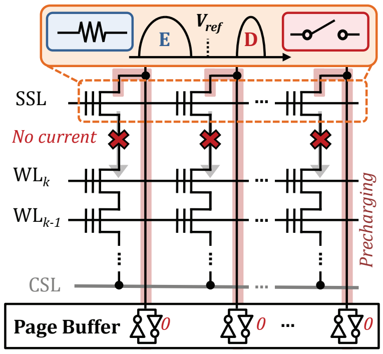
tween when the block is erased and when data is programmed to the block [84]. This time gap is called an *open interval*. The shorter the open interval, the more reliable the storage of data. Figure 41 illustrates how RBER increases as the length of an open interval increases. When the open interval is the largest we tracked, RBER is 30% larger than when the open interval is zero. To avoid the reliability problem of an open interval, a block should be erased *lazily*, i.e., the erase of the block should happen just before programming data on the block. Therefore, `bLock` is essential to effectively sanitize data in an entire block without reliability issues.

Implementation. We implement `bLock` by leveraging a new feature of 3D flash organization. We allocate per-block bAP flags in the SSL of each flash block. As explained in Section 2.1, there is an SSL at the top of each block, which is used to select the active block during flash operations. Unlike 2D flash memory where normal transistors are used for SSL transistors, 3D flash memory uses normal flash cells as SSL transistors [85], which allows us to program (and erase) the SSL of a block as a normal WL.[8] Therefore, by sufficiently increasing the V_{th} levels of SSL *cells* (i.e., programming the SSL just like programming a normal WL), we can turn the SSL cells of a block into *off switches*, which effectively inhibit all read requests to the block. Since there is no way to erase only SSL cells using the standard flash interfaces, `bLock` can efficiently sanitize an entire block.

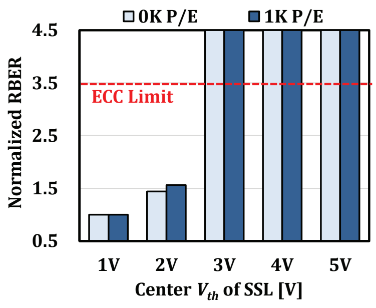
Figure 42(a) shows the operational overview of our `bLock` implementation. To disable a block (i.e., to set its bAP to `disabled`), `bLock` shifts the V_{th} levels of SSL cells to higher than V_{READ}, which effectively disconnects all the flash cells below such SSL cells from the page buffer. Since no current can flow through BLs, the page buffer data for all the flash pages in a block is fixed to '0' irrespective of the actual page data. As shown in Figure 42(b), we find that when the center V_{th} level of an SSL exceeds 3V, a read operation to any of the pages in the corresponding block fails due to the introduction of enough bit errors beyond the correction capability of ECC.

Design Space Exploration. To implement `bLock` in practice, two requirements should be satisfied. First, `bLock` should move the V_{th} levels of SSL cells sufficiently so that all SSL cells are completely turned off during a read operation. Second, before physically erasing a flash block, SSL cells should reliably keep their high V_{th} levels during the entire lifetime. Note that, in `bLock`, we do not need to consider the interference between an SSL and other WLs in the same block

[8]Using a normal flash cell to implement an SSL transistor is inevitable in 3D NAND flash memory because inserting a normal transistor in a vertically-stacked 3D flash organization is more difficult than inserting a flash cell.



(a) An operational overview.



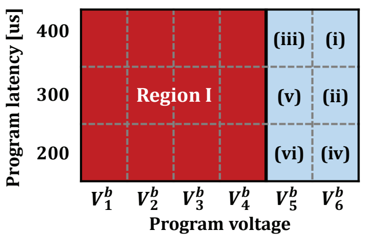
(b) RBER vs. center V_{th} of SSL.

Figure 42: `bLock` implementation.

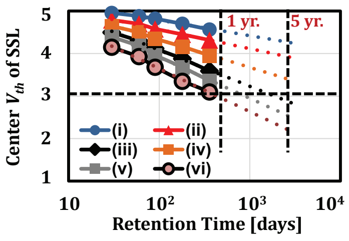
because an SSL is already physically separated from other WLs in the same block via a dummy WL (which is inserted between an SSL and WLs) to prevent potential unintentional programming of an SSL during a normal flash operation.

In our `bLock` implementation, we consider two design parameters related to bAP flags: program voltage V_{prog}^{bLock} and program latency t_{bLock} which are used for programming bAP flags (i.e., SSL cells). To minimize latency, we use the one-shot program scheme for `bLock`. To find a good combination of (V_{prog}^{bLock}, t_{bLock}), we start from an initial design space $\Psi \times T$ where $\Psi = \{V_1^b, V_2^b, ..., V_6^b\}$ ($V_{i+1}^b - V_i^b = 1.0V$) and $T = \{200\mu\text{s}, 300\mu\text{s}, 400\mu\text{s}\}$, as shown in Figure 43(a).

First, we evaluate if each combination can reliably program an SSL so that the center V_{th} level of the SSL is maintained above 3V with the one-shot programming technique. Based on our evaluation, we exclude the candidate combinations in Region I (in Figure 43(a)) from further consideration because they cannot move the center V_{th}



(a) Initial design space $\Psi \times T$.



(b) Impact of retention time on center V_{th} of SSL.

Figure 43: Design space exploration results for `bLock`.

level of SSL transistors to higher than 3V with a desired latency.

Second, we evaluate how the V_{th} levels of SSL cells change under a given retention time requirement. We test two retention-time requirements at 30 °C after 1K P/E cycles, 1-year and 5- year retention times. Figure 43(b) shows that the center V_{th} level of an SSL significantly vary depending on different (V_{prog}^{pLock}, t_{pLock}) combinations. For example, the center V_{th} level of an SSL programmed with combination (i), (V_6^b, 400μs), is predicted to be more than 4V even after 5 years, while the center V_{th} level of an SSL programmed with combination (vi), (V_5^b, 200μs), is predicted to be lower than 3V before 1 year. Thus, combination (vi) is not reliable (and neither are combinations (iv) and (v)). Considering both t_{bLock} and retention reliability, we select combination (ii), (V_6^b, 300μs), as our final design parameters.

5.2.5 Implementation Overhead

Area Overhead. The proposed `pLock` implementation requires one 9-bit majority circuit per flash chip in addition to 27 flag cells per each

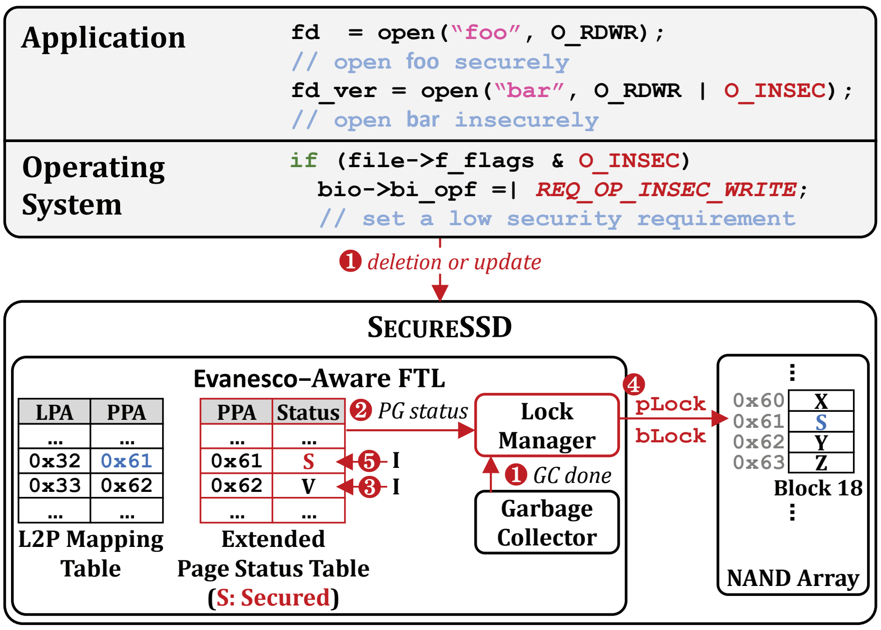
WL. Since we implement the flag cells using the unused flash cells in the spare area of a WL, no space overhead exists for supporting the pAP flags. For the 9-bit majority circuit, approximately 200 transistors are needed [86]. Considering the size of the typical peripheral circuit area in modern flash memory, the area impact of this majority circuit is insignificant. The area overhead of the bridge transistors is also negligible because only one bridge transistor is needed for each data-out path. For example, only 8 bridge transistors are needed for a typical $\times 8$ I/O NAND flash chip.

Latency Overhead. In our implementation, t_{pLock} and t_{bLock} are 100μs and 300μs, respectively. Compared to the page-program latency ($tPROG$) and block-erasure latency ($tBERS$), the latency overhead of `pLock` and `bLock` is very small. For 3D TLC NAND flash memory, t_{pLock} is less than 14.3% of $tPROG$ (700μs), and t_{bLock} is less than 8.6% of $tBERS$ (3.5ms) [14].

5.3 SecureSSD: System Integration

In order to take full advantage of `pLock` and `bLock` at the system level, we design an evanesco-enabled flash-based storage system, called SecureSSD, which efficiently supports data sanitization at low cost by interacting with a host computing system. Although `pLock` and `bLock` provide low-cost data sanitization at the flash-chip level, it would unnecessarily degrade both SSD and system performance if they are used for sanitizing security-insensitive data. To avoid this, SecureSSD allows the user to specify the *security requirements* of written data through an extended I/O interface, so that the evanesco-aware FTL in SecureSSD uses pLock and bLock only when invalidating security-sensitive data.

Figure 44 shows how SecureSSD manages written data according to the data's security requirements. To support data sanitization for evanesco-*unaware* systems in a backward compatible manner, SecureSSD, by default, treats all written data as security sensitive. When an evanesco-aware application does *not* require high security for a file (e.g., bar in Figure 44), it opens the file with a new access mode flag O_INSEC. Opening a file with O_INSEC indicates that the file data can have multiple versions in the SSD and deletion is not



```
fd     = open("foo", O_RDWR);
// open foo securely
fd_ver = open("bar", O_RDWR | O_INSEC);
// open bar insecurely
```

```
if (file->f_flags & O_INSEC)
  bio->bi_opf =| REQ_OP_INSEC_WRITE;
  // set a low security requirement
```

Figure 44: Operational overview of SecureSSD.

secure. For a write request to a file opened with the `O_INSEC` flag, a block I/O request to SecureSSD is flagged with a new operation flag `REQ_OP_INSEC_WRITE` so that SecureSSD is aware that the written data is security insensitive.

To keep track of the security requirement of each page, the evanesco-aware FTL in SecureSSD employs an extended page status table and a lock manager. A page in SecureSSD can be in one of four states: free, valid, invalid, or *secured*. For a default write (e.g., a write to LPA `0x32` of `foo` in Figure 44), the FTL updates the L2P mapping for the requested LPA with a free PPA (e.g., `0x61`), and sets the page status to secured. In contrast, for a security-insensitive write (e.g., a write to LPA `0x33` of `bar`), the FTL sets the page status of the corresponding PPA (e.g., `0x62`) to valid instead of secured.

When a PPA needs to be invalidated, e.g., due to a file update/deletion from the host or a copy operation in the GC process (❶ in Figure 44), the lock manager first retrieves the status of the PPA from the extended page status table (❷). If the status of the PPA is not secured, the FTL only updates the status to invalid (❸) as a regular evanesco-unaware FTL would do. If the status of the PPA is secured, the lock manager *immediately* invokes a `pLock` or `bLock` command depending on the status of the other pages in the same block (❹). For example, when sanitizing a single secured page, the lock manager issues a `pLock` command. On the other hand, when 1) all the remaining pages in a block need to be sanitized (e.g., during GC or due to a trim request to contiguous secured pages) and 2) the estimated latency for sanitizing the pages with pLock is longer than t_{bLock}, the lock manager issues a bLock command to minimize the performance overhead due to data sanitization. After that, the FTL updates the status of the securely-invalidated page(s) to invalid (❺).

5.4 System-Level Evaluation

Methodology. We implement SecureSSD on FlashBench [76] with an evanesco-enabled emulated flash model. Although FlashBench supports up to 512-GiB capacity, we limit its SSD capacity to 32 GiB for fast evaluation. We configure SecureSSD with two channels, each of which has four 3D TLC NAND flash chips. Each chip has 428 blocks and each block has 576 16-KiB pages (i.e., 192 WLs). We set flash operation timing parameters for $tREAD$, $tPROG$, and $tBERS$ to 80μs, 700μs, and 3.5ms, respectively. Based on our design space exploration results, we set t_{pLock} and t_{bLock} to 100μs and 300μs, respectively.

We use four different benchmark traces. Three traces, MailServer, DBServer, and FileServer, are generated with the Filebench benchmark tool [79]. One trace, Mobile, is collected from an Android smartphone (Samsung Galaxy S2 [87]). Table 4 summarizes three main I/O characteristics of the evaluated benchmarks: 1) read to write ratio, 2) write pattern, and 3) write size. We use a custom trace replayer that sends each write request to an SSD with its security requirements and aligns the data boundary of each write request to multiples of 16 KiB (i.e., the physical page size).

Table 4: I/O characteristics of our four benchmarks.

Benchmark	read:write	File write pattern	Write size
MailServer	1:1	create/append/delete e-mails	16–32 KiB
DBServer	1:10	overwrite data files and log files	16–256 KiB
FileServer	3:4	create/append/delete files	32–128 KiB
Mobile	1:50	create/delete pictures	0.5–8 MiB

We compare SecureSSD (`secSSD`) with two baseline SSDs, `erSSD` and `scrSSD`, which exploit existing physical sanitization techniques to support *immediate* data sanitization.[9] As with `secSSD`, `erSSD` and `scrSSD` manage write requests in a secure fashion, only if the requests have a high security requirement. When a secured page needs to be invalidated, `erSSD` *erases* the entire block that contains the secured page[10] while `scrSSD` performs *scrubbing* on the WL that contains the secured page. When the target block or the target WL has other valid pages, `erSSD` and `scrSSD` copy the valid pages to other free pages. In `scrSSD`, we set the scrubbing latency to 100μs assuming that the one-shot programming scheme is used to minimize performance overhead. To understand the benefits of `pLock` and `bLock`, we also evaluate `secSSD`$_{nobLock}$ which works in the same fashion as `secSSD` but without `bLock`.

Evaluation Results. To evaluate the performance of SecureSSD, we measure input/output operations per second (IOPS) performance and

[9] All three SSDs we evaluate guarantee that $\forall t : N^{page}_{invalid}(f, t) = 0$, for a file f with a high security requirement.

[10] Since we are interested in comparing the I/O performance of `erSSD` and `secSSD`, we assume that there is no reliability issue due to the open block problem (see Section 5.2.4) in `erSSD`, i.e., `erSSD` can immediately erase a block without any reliability issue.

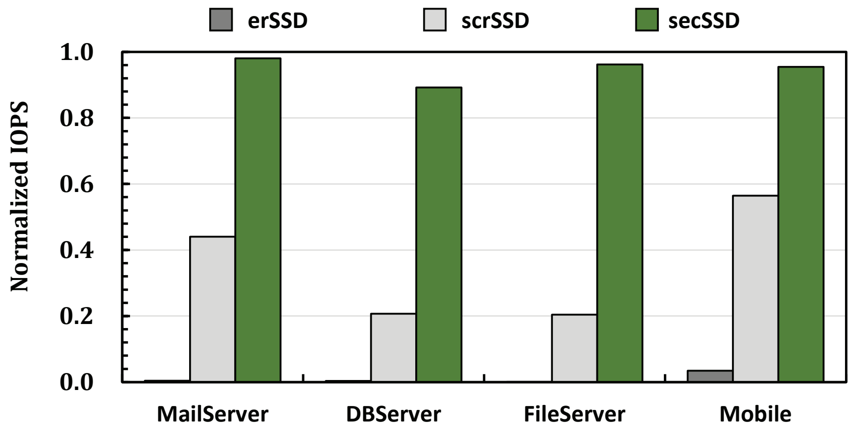


Figure 45: IOPS of different SSDs (higher is better).

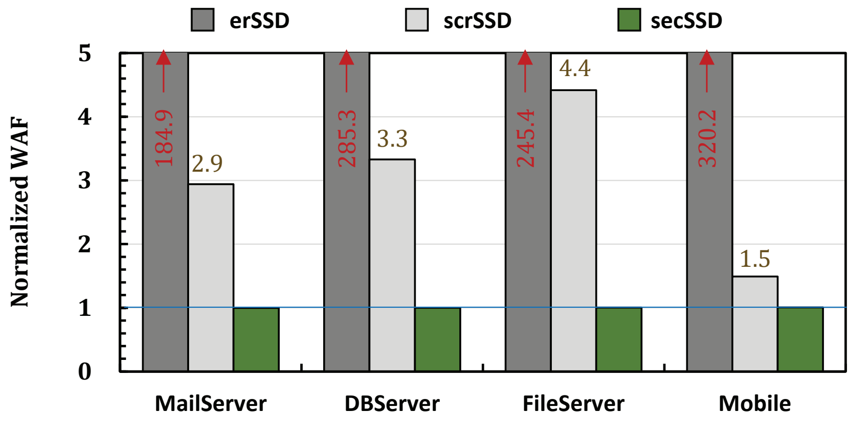


Figure 46: WAF of different SSDs (lower is better).

write amplification factor (WAF) values for each SSD. All values are normalized to ones from an SSD with no data sanitization support. Figure 45 compares IOPS values of different SSDs under each workload. SecSSD significantly outperforms erSSD and scrSSD under every workload. erSSD performs poorly, achieving less than 4% of the IOPS

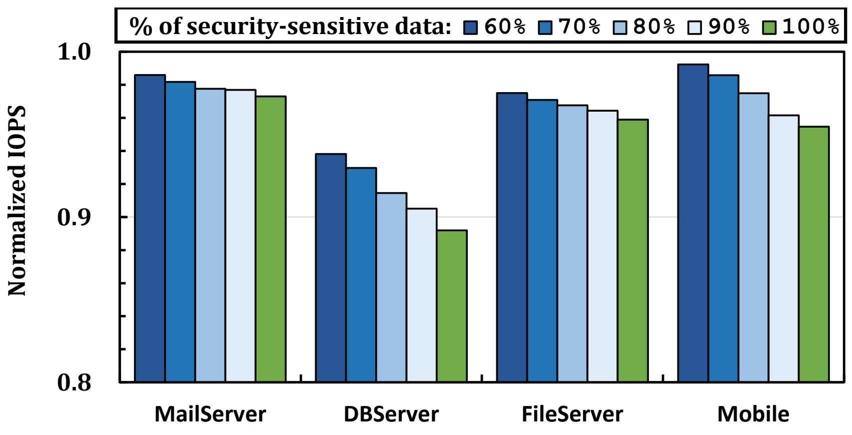


Figure 47: Performance of SecureSSD under four different workloads.

level of the baseline SSD. Although `scrSSD` significantly outperforms `erSSD`, it achieves only 34% of the performance of the baseline SSD. In contrast, `secSSD` achieves 94.5% of the performance of the baseline SSD.

The performance gap between `secSSD` versus `erSSD` / `scrSSD` is mainly due to the large number of additional copy operations present in `erSSD` and `scrSSD`. As shown in Figure 46, the WAF values of `erSSD` and `scrSSD` are substantially higher, by up to 320× and 4.41× (251× and 2.6× on average) over the baseline SSD, respectively. In contrast, `secSSD` achieves almost equivalent WAF as the baseline SSD. Note that the amplified writes in `erSSD` and `scrSSD` can greatly degrade the SSD lifetime as well as the IOPS performance due to more frequent GC invocations.

Even though both $\texttt{secSSD}_{nobLock}$ and `secSSD` can sanitize a page without copying other valid pages stored in the same WL, secSSD$_{nobLock}$ has lower performance than secSSD, in particular, under workloads with large-size writes (as in FileServer and Mobile). This is because the more pages are invalidated at the same time, the more opportunities for secSSD to sanitize an entire block by using bLock. We compare the number of pLock operations performed in secSSD and secSSD$_{nobLock}$ under each workload: the results show that the use of bLock operations reduces the number of pLock operations by up to 57% (28% on average) in secSSD. As a result, secSSD further improves the IOPS performance by up to 5.4% (3.1% on average) over secSSD$_{nobLock}$, as shown in Figure 45.

Finally, we measure the IOPS performance of secSSD under each workload while varying the fraction of securely-managed data, as shown in Figure 47. The fewer the secured pages, the higher the performance secSSD could achieve by using pLock and bLock only when invalidating security-sensitive data. When managing 60% of total written data in a secure fashion, the performance of secSSD is only up to 6.2% (2.8% on average) lower than that of the baseline SSD. Although selective sanitization has a higher performance impact under write-intensive workloads, DBServer and Mobile, secSSD exhibits the lowest performance in DBServer. This is because DBServer issues a large number of small updates to securely managed files such that secSSD has little opportunity to perform bLock and to exploit the internal parallelism of the SSD for pLock operations.

Chapter 6

Conclusions

6.1 Summary

NAND flash-based storage systems have been widely adopted in modern computing systems, owing to their superior properties compared to HDDs and the continuous reduction of the cost-per-bit value of NAND flash memory. However, due to the popularity of NAND flash memory in emerging storage market areas (e.g., data-driven applications), NAND flash-based storage systems are necessary to satisfy new requirements such as high security or better user experience. In order to address these new challenges properly, we should explore new optimization techniques based on revisiting existing flash solutions.

In this dissertation, we proposed several system-level techniques that aim at resolving new emerging requirements without sacrificing performance or lifetime. First, we presented `RealWear`, a new NAND aging marker, that represents the wear status of NAND blocks very accurately. Unlike the existing P/E cycle-based aging marker, `RealWear` systematically exploits the underlying physics of the NAND cell aging process in deriving a NAND flash wear model. Based on an extensive real flash characterization study using recent 3D flash chips, 5 key model parameters were selected and a novel flash wear model was developed using regression analysis. Our validation study showed that `RealWear` can accurately classify NAND blocks based on their real wear status. In order to demonstrate the practical benefit of `RealWear` at the flash storage system level, we reported three case studies that improve the lifetime and performance of flash storage systems. Our experimental results show that a `RealWear`-aware FTL can enhance the lifetime and the performance of a flash storage system by 63% and 21%, respectively over a P/E cycle-based conventional FTL. Furthermore, for the first time, we demonstrate that the worst-case flash read latency can be bounded by using `RealWear` in deciding an optimal read voltage level.

Second, we introduced a new suspend/resume technique, pSR, which allows us to process a high-priority read request first without a delay that caused by conflicts with other preceding flash operations. Based on the accurate NAND error model, we revisit the existing preemption command to avoid flash reliability degradation by excessive suspend/resume operations. In order to take advantages of pSR-enabled NAND devices at the storage level, we developed a priority-aware FTL, paFTL, which employs a novel priority-aware suspend/resume technique that comprehensively treats the read-first command preemption policy. Our experimental results using paFTL show that the proposed priority-aware flash management scheme is effective in differentiating read performance over tasks with different priorities.

Finally, we proposed evanesco, a new chip-level data sanitization technique for modern flash-based storage systems. Evanesco supports immediate per-page sanitization at low cost with two new flash commands, pLock and bLock, that disable access to a page or a block, respectively, using access control mechanisms implemented on the flash chip. By leveraging spare cells in existing flash memory organization, evanesco effectively makes sanitized data in a flash chip inaccessible, with only a small resource overhead. Using state-of-the-art 3D NAND flash chips, we validate that pLock and bLock can quickly disable a target page and block without compromising reliability of stored data. To fully exploit pLock and bLock, we design an evanesco-enabled flash storage system, SecureSSD, which efficiently and securely manages security-sensitive data by interacting with a host system using extended I/O interfaces. Our experimental results show that SecureSSD can delete security-sensitive files immediately and irrecoverably while providing a comparable performance to an SSD with no data sanitization support.

6.2 Future Work

6.2.1 Prediction Model for Sudden NAND flash Failure

Although we believe that RealWear can accurately represent the *normal* NAND aging process, it cannot be used to predict sudden NAND flash failures that are not related to the T_{ox} layer. For example, when some peripheral circuitry of a NAND flash chip suddenly malfunctions, RealWear cannot be used to prepare for such a sud-

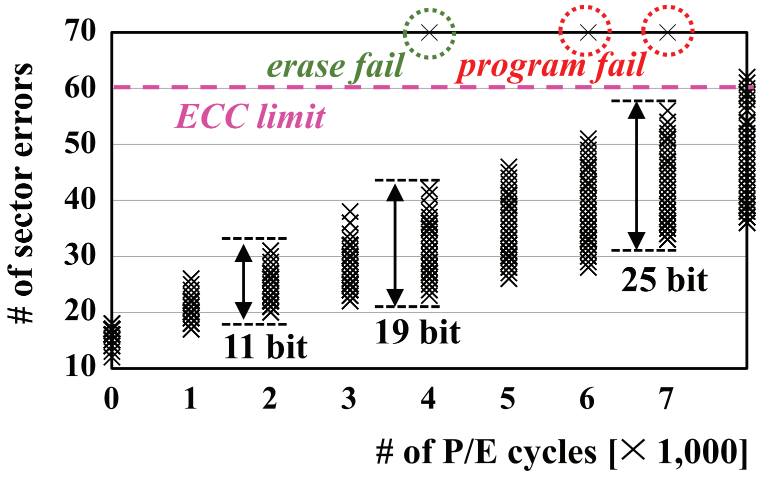


Figure 48: Sudden failure of NAND blocks.

den failure in advance. As shown in Figure 48, some NAND blocks suddenly become dead (i.e., no more store the data reliably) by program or erase status fail. These failures occur regardless of the wear of NAND flash memory and can spread to entire NAND blocks in a flash chip. One of our immediate directions is to develop a prediction model for such sudden failures. From our preliminary evaluation, we verified that there exist meaningful fore-warnings for some sudden chip failures. When such a prediction model is combined with `RealWear`, the reliability of a flash storage system is expected to be significantly improved.

6.2.2 Extensions of Read Retry Mitigation

In our BoudedRead, the number of read retries (`NumRetry`) is mitigated by pre-defined look-up table (ORT) containing the optimal

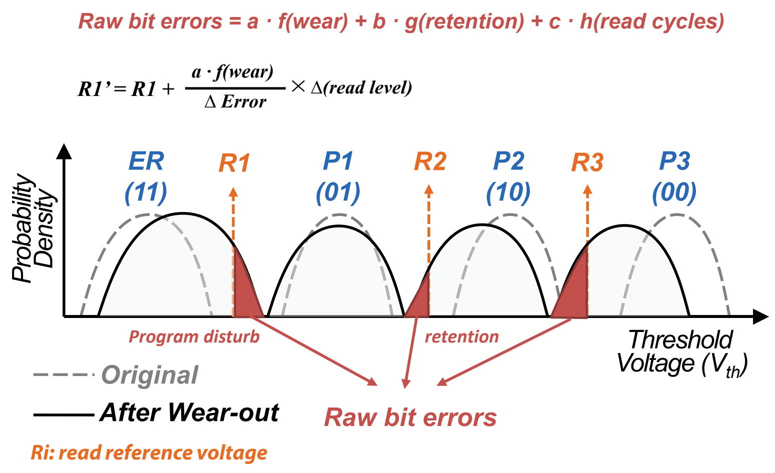

Raw bit errors = a · f(wear) + b · g(retention) + c · h(read cycles)
R1' = R1 + a · f(wear) / Δ Error × Δ(read level)
Probability Density
ER (11)
R1
P1 (01)
R2
P2 (10)
R3
P3 (00)
Program disturb
retention
Threshold Voltage (V_th)
Original
After Wear-out
Raw bit errors
Ri: read reference voltage

Figure 49: Estimation of optimal read reference voltage.

read reference voltages. Although the effect of BoudedRead is evident, it requires a great number of NAND characterization studies to define ORT. Moreover, flash storage systems should allocate a certain amount of memory space to maintain ORT.

To further optimize the read retry mitigation technique, we believe that the optimal read reference voltage can be obtained by a simple calculation based on accurate error prediction. Using a new NAND aging marker (`RealWear`) and noisy factors such as retention time or read cycles, the number of bit errors from a flash block can be precisely estimated. For example, as shown in Figure 49, when bit errors of block B is measured, we can calculate how many bit errors are generated by each factors. If we know how many bit errors can be reduced through one read retry (Δ Error), we can estimate how much the read reference voltage needs to be shifted. For example, if the number of bit errors caused by the wear of a flash block (i.e., program disturb error) is 20 and δ Error is 4, a total of 5 read retries is needed. Therefore, the new optimal R1 value (R1') can be calculated by R1 + 5 $\times$ δ(read level). (Δ(read level)) is the amount of read reference voltage that changes when one read retry is performed.)

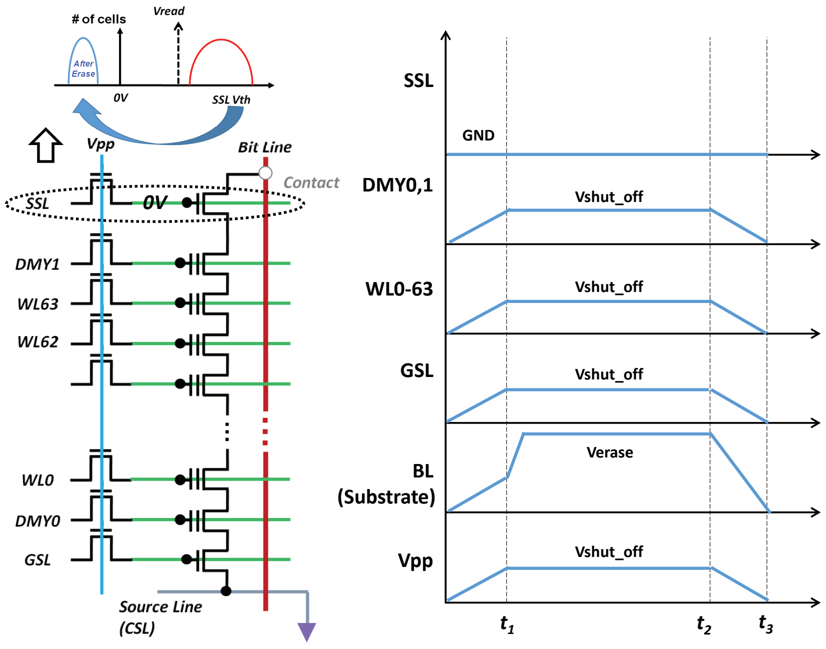
6.2.3 Extensions of evanesco

Our `bLock` efficiently sanitizes the data in NAND flash memory by disabling access to deleted data at block granularity, and access to sanitized block can only recovered by block erasure. In order to apply our sanitization technique to more various environments, we will attempt to a new data sanitization technique that can disable access to a flash block adaptively. By modifying the conventional NAND architecture, we can design two types of `bLock`; One is recoverable `bLock`, and the other is unrecoverable `bLock`.

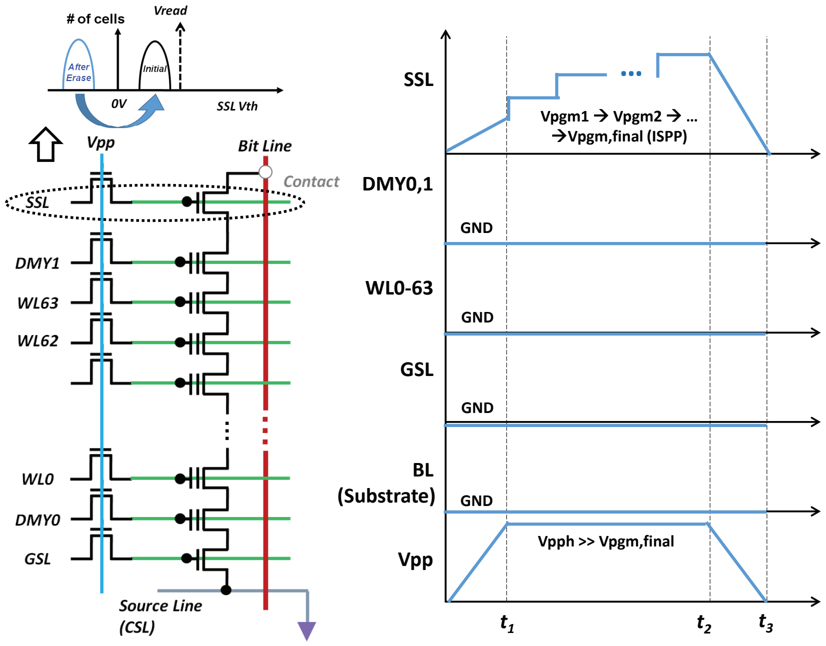
As shown in Figure 51, there need two key changes to implement the recoverable `bLock`. To set V_{th} of SSL transistors to an initial state, we should add new NAND interfaces; One is for erasing *only* SSL transistors, and the other is for programming *only* SSL transistors to raise their V_{th}. Figures 50(a) and 51(b) shows how to control only SSL transistors without affecting other WLs in a flash block for recoverable `bLock`. On the other hand, for unrecoverable `bLock`, we can introduce another new NAND interface to program both SSL and dummy WL at the same time as shown in Figure 51. By exploiting these new data sanitization techniques, we expect that a flash storage system can protect its data more efficiently under various environments.

6.2.4 Extensions for Advanced Memory Technology

Based on both chip-level and system-level evaluations, we conclude that our techniques are an effective for modern flash memory based SSDs with low overheads. We believe the basic ideas of our techniques are applicable to other memory technologies such as advanced 3D NAND flash memory or new emerging memories (e.g., PCM, STT-MRAM, or RRAM). However, depending on the electrical/physical characteristics of each memory technologies, optimization techniques should be modified to well-suit for the purpose. For 3D NAND flash memory, as the number of vertical layers increases, evanesco should be more carefully managed because successive programming flag cells can cause exacerbate program disturbance in main data. In addition, recent COP (Cell On Peri) technology, which can change the internal organization of NAND flash-based storage system, demands a totally different approach to optimize a storage system.



(a) The internal conditions for erasing SSL.



(b) The internal conditions for programming SSL.

Figure 50: Re-design of NAND architecture for recoverable `bLock`.

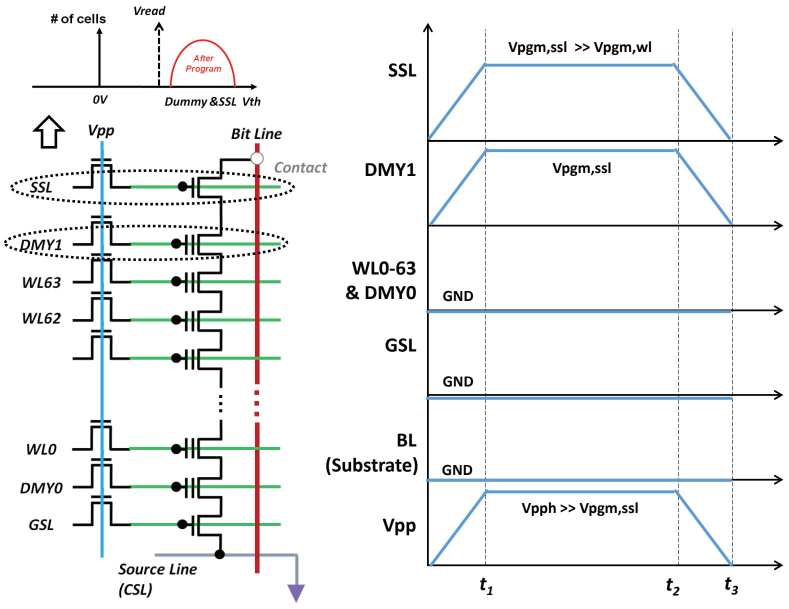


Figure 51: Re-design of NAND architecture for unrecoverable `bLock`.

Bibliography

[1] J. Jang, H. Kim, W. Cho, H. Cho, J. Kim, S. Shim, Y. Jang, J. Jeong, B. Son, D. Kim, K. Kim, J. Shim, J. Lim, K. Kim, S. Yi, J. Lim, D. Chung, H. Moon, S. Hwang, J. Lee, Y. Son, Y. Chung, Y. Lee, "Vertical cell array using TCAT (Terabit Cell Array Transistor) technology for ultra high density NAND flash memory," in *Proceedings of the IEEE Symposium on VLSI Technology* (VLSI), 2009.

[2] S. Lee, J.-Y. Lee, I.-H. Park, J. Park, S.-W. Yun, M.-S. Kim, J.-H. Lee, M. Kim, K. Lee, T. Kim, B. Cho, D. Cho, S. Yun, J.-N. Im, H. Yim, K.-H. Kang, S. Jeon, S.-Jo, Y.-L. Ahn, S.-M. Joe, S. Kim, D.-K. Woo, J. Park, H.-W. Park, Y. Kim, J. Park, Y. Choi, M. Hirano, J.-D. Ihm, B. Jeong, S.-K. Lee, M. Kim, H. Lee, S. Seo, H. Jeon, C.-H. Kim, H. Kim, J. Kim, Y. Yim, H. Kim, D.-S. Byeon, H.-J. Yang, K.-T. Park, K.-H. Kyung, and J.-H. Choi, "A 128Gb 2b/Cell NAND Flash Memory in 14nm Technology with tPROG=640μs and 800MB/s I/O Rate," in *Proceedings of the IEEE International Solid-State Circuits Conference* (ISSCC), 2016.

[3] N. Shibata *et al.*, "A 1.33 Tb 4-bit/Cell 3D-flash memory on a 96-Word-Line-Layer technology," IEEE International Solid-State Circuits Conference (ISSCC), pp. 210–212, 2019.

[4] S. Lee *et al.*, “A 1Tb 4b/cell 64-stacked-WL 3D NAND flash memory with 12MB/s program throughput,” IEEE International Solid-State Circuits Conference (ISSCC), pp. 340–342, 2018.

[5] AEC, “AEC-Q100 Qualification specifications from the Automotive Electronics Council (AEC),” http://www.aecouncil.com/AECDocuments.html.

[6] M. Kim, Y. Song, M. Jung, and J. Kim, “SARO: A State-Aware Reliability Optimization Technique for High Density NAND Flash Memory,” in *Proceedings of the ACM Great Lakes Symposium on VLSI* (GLSVLSI), 2018.

[7] Enterprise Storage, “SSD Lifespan: How Long Will Your SSD Work?,” https://www.enterprisestorageforum.com/storage-hardware/ssd-lifespan.html, 2019.

[8] F. Roohparvar, “Single Level Cell Programming in a Multiple Level Cell Non-Volatile Memory Device,” In Unite States Patent, Number 7,366,013, April 2008.

[9] J. Maserjian and N. Zamani, “Behavior of the Si/SiO2 interface observed by Fowler-Nordheim tunneling,” AIP Journal of Applied Physics, 1982.

[10] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, “Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery,” in *Proceedings of the 45th Annual IEEE/IFIP*

International Conference on Dependable Systems and Networks (DSN), 2015.

[11] K. Kanda, N. Shibata, T. Hisada, K. Isobe, M. Sato, Y. Shimizu, T. Shimizu, T. Sugimoto, T. Kobayashi, N. Kanagawa, Y. Kajitani, T. Ogawa, K. Iwasa, M. Kojima, T. Suzuki, Y. Suzuki, S. Sakai, T. Fujimura, Y. Utsunomiya, T. Hashimoto, N. Kobayashi, Y. Matsumoto, S. Inoue, Y. Suzuki, Y. Honda, Y. Kato, S. Zaitsu, H. Chibvongodze, M. Watanabe, H. Ding, N. Ookuma, and R. Yamashita, "A 19 nm 112.8 mm^2 64 Gb Multi-Level Flash Memory With 400 Mbit/sec/pin 1.8 V Toggle Mode Interface," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 159–167, 2012.

[12] K. T. Park, S. Nam, D. Kim, P. Kwak, D. Lee, Y. Choi, M. H. Choi, D. H. Kwak, D. H. Kim, M. S. Kim, H. W. Park, S. W. Shim, K. M. Kang, S. W. Park, K. Lee, H. J. Yoon, K. Ko, D. K. Shim, Y. L. Ahn, J. Ryu, D. Kim, K. Yun, J. Kwon, S. Shin, D. S. Byeon, K. Choi, J. M. Han, K. H. Kyung, J. H. Choi, and K. Kim, "Three-dimensional 128 Gb MLC vertical NAND flash memory with 24-WL stacked layers and 50 MB/s high-speed programming," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 204–213, 2014.

[13] H. Nitta, T. Kamigaichi, F. Arai, T. Futatsuyama, M. Endo, N. Nishihara, T. Murata, H. Takekida, T. Izumi, K. Uchida, T. Maruyama, I. Kawabata, Y. Suyama, A. Sato, K. Ueno,

H.Takeshita, Y. Joko, S. Watanabe, Y. Liu, H. Meguro, A. Kajita, Y. Ozawa, Y. Takeuchi, T. Hara, T. Watanabe, S. Sato, H. Tomiie, Y. Kanemaru, R. Shoji, C. H. Lai, M. Nakamichi, K. Owada, T. Ishigaki, G. Hemink, D. Dutta, Y. Dong, C. Chen, G. Liang, M. Higashitani, J. Lutze, "Three bits per cell floating gate NAND flash memory technology for 30nm and beyond," in *Proceedings of the IEEE International Reliability Physics Symposium* (IRPS), 2014.

[14] W. Jeong, J. W. Im, D. H. Kim, S. W. Nam, D. K. Shim, M. H. Choi, H. J. Kim, D. H. Kim, Y. S. Kim, H. W. Park, D. H. Kwak, S. W. Park, S. M. Yoon, W. G. Hahn, J. H. Ryu, S. W. Shim, K. T. Kang, J. D. Ihm, I. M. Kim, D. S. Lee, J. Cho, M. S. Kim, J. H. Jang, S. W. Hwang, D. S. Byeon, H. J. Yang, K. Park, K. H. Kyung, and J. H. Choi, "A 128 Gb 3b/cell V-NAND flash memory with 1 Gb/s I/O rate," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 204–212, 2015.

[15] K. Suh, B. Suh, Y. Lim, J. Kim, Y. Choi, Y. Koh, S. Lee, S. Kwon, B. Choi, J. Yum, J. Choi, J. Kim, and H. Lim, "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, 1995.

[16] Y. Cai, E. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, and

analysis," in *Proceedings of the the Conference on Design, Automation and Test in Europe* (DATE), 2012.

[17] Y. Cai, Y. Luo, E. Haratsch, K. Mai, O. Mutlu, "Data retention in MLC NAND flash memory: Characterization, optimization, and recovery," in *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture* (HPCA), 2015.

[18] K.-T. Park, M. Kang, D. Kim, S.-W. Hwang, B. Y. Choi, Y.-T. Lee, C. Kim, and K. Kim, "A Zeroing Cell-to-Cell Interference Page Architecture with Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 919–928, 2008.

[19] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. Haratsch, "Vulnerabilities in MLC NAND flash memory programming: Experimental analysis, exploits, and mitigation techniques," in *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture* (HPCA), 2017.

[20] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories*, Springer, 2010.

[21] Micron, "Micron 3D NAND flash memory," https://www.micron.com/-/media/client/global/documents/products /product-flyer/3d-nand-flyer.pdf, 2016.

[22] M. Ishiduki, Y. Fukuzumi, R. Katsumata, M. Kito, M. Kido, H. Tanaka, Y. Komori, Y. Nagata, T. Fujiwara, T. Maeda, Y. Mikajiri, S. Oota, M. Honda, Y. Iwata, R. Kirisawa, H. Aochi, and A. Nitayama, "Optimal device structure for pipe-shaped BiCS flash memory for ultra high density storage device with excellent performance and reliability," in *Proceedings of the IEEE International Electron Devices Meeting* (IEDM), 2009

[23] Y. Park, J. Lee, S. Cho, G. Jin, and E. Jung, "Scaling and reliability of NAND flash devices," in *Proceedings of the IEEE Symposium on Reliability Physics* (IRPS), 2014.

[24] E. Choi and S. Park, "Device considerations for high density and highly reliable 3D NAND flash cell in near future," in *Proceedings of the IEEE International Electron Devices Meeting* (IEDM), 2012.

[25] A. Torsi, Y. Zhao, H. Liu, T. Tanzawa, A. Goda, P. Kalavade, and K. Parat, "A program disturb model and channel leakage current study for sub-20 nm NAND flash cells," *IEEE Transactions on Electron Devices*, vol. 58, no. 1, pp. 11–16, 2010.

[26] Y. Woo and J. Kim "Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs," in *Proceedings of the International Conference on Embedded Software* (EMSOFT), 2013.

[27] B. Peleato, H. Tabrizi, R. Agarwal, and J. Ferreira, "BER-based wear leveling and bad block management for NAND flash," in *Proceedings of the IEEE International Conference on Communications* (ICC), 2015.

[28] S. Wang, F. Wu, C. Yang, Z. Zhou, C. Xie, and J. Wan, "WAS: Wear Aware Superblock Management for Prolonging SSD Lifetime," in *Proceedings of the Design Automation Conference* (DAC), 2019.

[29] M. Neal and J. Chen, "Reliability of flash nonvolatile memories," *Oxide Reliability: A Summary of Silicon Oxide Wearout, Breakdown, and Reliability*, pp. 103–134, 2002.

[30] A. Spinelli, C. Compagnoni, and A. Lacaita, "Reliability of NAND flash memories: Planar cells and emerging issues in 3D devices," *Computers*, vol. 6, no. 2, pp. 16, 2017.

[31] Q. Li, M. Ye, Y. Cui, L. Shi, X. Li, and C. Xue, "Sentinel Cells Enabled Fast Read for NAND Flash," in *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems* (HotStorage), 2019.

[32] G. Wu and X. He, "Reducing SSD Read Latency via NAND Flash Program and Erase Suspension," in *Proceedings of the USENIX Conference on File and Storage Technologies* (FAST), 2012.

[33] S. Kim, J. Bae, H. Jang, W. Jin, J. Gong, S. Lee, T. Ham, and J. Lee, "Practical Erase Suspension for Modern Low-latency

SSDs," in *Proceedings of the USENIX Annual Technical Conference* (ATC), 2019.

[34] S. Diesburg, C. Meyers, M. Stanovichi, M. Mitchell, J. Marshall, J. Gould, and A. Wang, "TrueErase: Per-File Secure Deletion for the Storage Data Path," in *Proceedings of the 28th Annual Computer Security Applications Conference* (ACSAC), 2012.

[35] K. Sun, J. Choi, and S. H. Noh, "Models and Design of an Adaptive Hybrid Scheme for Secure Deletion of Data in Consumer Electronics," *IEEE Transactions on Consumer Electronics* (TEC), vol. 54, no. 1, pp. 100–104, 2008.

[36] W. C. Wang, C. C. Ho, Y. H Chang, K. T. W. Kim, and P. H. Lin, "Scrubbing-Aware Secure Deletion for 3-D NAND Flash," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (TCAD), vol. 37, no. 11, pp. 2790–2081, 2018.

[37] M. Y. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, "Reliably Erasing Data from Flash-Based Solid State Drives," in *Proceedings of the USENIX Conference on File and Storage Technologies* (FAST), 2011.

[38] S. Jia, L.Xie, B. Chen, and P. Liu, "NFPS: Adding undetectable secure deletion to flash translation layer," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (ASIACCS), 2016.

[39] P. H. Lin, Y. M. Chang, Y. C. Li, C. C. Ho, and Y. H. Chang, "Achieving Fast Sanitization with Zero Live Data Copy for MLC Flash Memory," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (ICCAD), 2018.

[40] D. Boneh and R. J. Lipton, "A Revocable Backup System," in *Proceedings of the USENIX Security*, 1996.

[41] J. Lee, J. Heo, Y. Cho, J. Hong, and S. Y. Shin, "Secure Deletion for NAND Flash File System," in *Proceedings of the ACM Symposium on Applied Computing* (SAC), 2008.

[42] J. Lee, K. Ganesh, H. Lee, and Y. Kim "FeSSD: A Fast Encrypted SSD Employing On-Chip Access-Control Memory," *IEEE Computer Architecture Letters* (CAL), vol. 16, no. 2, pp. 115–118, 2017.

[43] J. Reardon, S. Capkun, and D. Basin, "Data Node Encrypted File System: Efficient Secure Deletion for Flash Memory," in *Proceedings of the USENIX Security*, 2012.

[44] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*, Springer, 2013.

[45] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and W. E. Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," in *Proceedings of the USENIX Security*, 2008.

[46] T. Müller, T. Latzo, and F. C. Freiling, "Self-Encrypting Disks Pose Self-Decrypting Risks: How to Break Hardware-Based Full Disk Encryption," *Technical Report Friedrich-Alexander University of Erlangen-Nuremberg*, 2012.

[47] C. Greider and E. Blackburn, "A telomeric sequence in the RNA of Tetrahymena telomerase required for telomere repeat synthesis," *Nature*, vol. 16, no. 6205, pp. 331–337, 1989.

[48] R. Khamsi, "Twins grow apart as they age," https://www.nature.com/articles/news050704-3, 2005.

[49] S. Swapp, "Scanning Electro Microscopy (SEM)," https://serc.carleton.edu/research_education/geochemsheets/techniques /SEM.html, 2017.

[50] A. Chou, K. Lai, K. Kumar, P. Chowdhury, and J. Lee, "Modeling of stress-induced leakage current in ultrathin oxides with the trap-assisted tunneling mechanism," *Applied physics letters*, vol. 70, no. 25, pp. 3407–3409, 1997.

[51] S. Kamohara, D. Park, and C. Hu "Deep-trap SILC (stress induced leakage current) model for nominal and weak oxides," in *Proceedings of the 36th IEEE International Reliability Physics Symposium* (IRPS), 1998.

[52] JEDEC, "JEDEC Solid State Technology Assn., Solid-State Drive (SSD) Requirements and Endurance Test Method [JESD218]," https://www.jedec.org, 2010.

[53] S. Arrhenius, “Über die Dissociationswärme und den Einfluss der Temperatur auf den Dissociationsgrad der Elektrolyte,” *Zeitschrift für physikalische Chemie*, vol. 4, pp. 96–116, 1889.

[54] Intel, “Intel SSD Data Center Family,” https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/data-center-ssds.html, 2019.

[55] Samsung, “Setting the standard in storage,” https://www.samsung.com/semiconductor/ssd/, 2019.

[56] Y. Luo, S. Ghose, Y. Cai, E. Haratsch, O. Mutlu, “HeatWatch: Improving 3D NAND flash memory device reliability by exploiting self-recovery and temperature awareness,” in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture* (HPCA), 2018.

[57] J. Cha, J. Kang, and S. Kang, “Data randomization scheme for endurance enhancement and interference mitigation of multilevel flash memory devices,” *Etri Journal* vol. 35, no. 1, pp. 166–169, 2013.

[58] J. Kattrunen, J. Kiihamaki, and S. Franssila, “Loading effects in deep silicon etching,” in *Proceedings of the International Society of Optical Engineering* (SPIE), 2000.

[59] M. Neal, H. P. Beljal, A. Fazio, Q. Meng, N. Righos, “Recovery effects in the distributed cycling of flash memories,” in *Proceedings*

of the IEEE International Reliability Physics Symposium (IRPS), 2006.

[60] S. Lee, T. Kim, K. Kim, and J. Kim, "Lifetime management of flash-based SSDs using recovery-aware dynamic throttling," in *Proceedings of the USENIX Conference on File and Storage Technologies* (FAST), 2012.

[61] Q. Wu, G. Dong, and T. Zhang, "Exploiting Heat-Accelerated Flash Memory Wear-Out Recovery to Enable Self-Healing SSDs," in *Proceedings of the USENIX Conference on Hot topics in Storage and File systems* (HotStorage), 2011.

[62] ONFI, "Open NAND Interface specification," http://www.onfi.org/specifications, 2020.

[63] S. Joe, J. Yi, S. Park, H. Shin, B. Park, Y. Park, and J. Lee, "Threshold voltage fluctuation by random telegraph noise in floating gate NAND flash memory string," *IEEE Transactions on Electron Devices* (TED), vol. 58, no. 1, pp. 67–73, 2011.

[64] C .Zuppa, "Error estimates for moving least square approximations," *Bulletin of the Brazilian Mathematical Society*, vol. 34, no. 2, pp. 231–249, 2003.

[65] S. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, "BlueDBM: An Appliance for Big Data Analytics," in *Proceedings of the International Symposium on Computer Architecture* (ISCA), 2015.

[66] Filebench, http://filebench.sourceforge.net.

[67] Sysbench, http://github.com/akopytov/sysbench.

[68] D. Hong, M. Kim, J. Park, M. Jung, and J. Kim, "Improving SSD Performance Using Adaptive Restricted-Copyback Operations'" in *Proceedings of the IEEE Non-Volatile Memory Systems and Applications Symposium* (NVMSA), 2019.

[69] F. Wu, J. Zhou, S. Wang, Y. Du, C. Yang, and C. Xie, "FastGC: Accelerate Garbage Collection Via an Efficient Copyback-based Data Migration in SSDs," in *Proceedings of the Design Automation Conference* (DAC), 2018.

[70] M. Doi, T. Tokutomi, S. Hachiya, A. Kobayashi, S. Tanakamaru, S. Ning, T. Iwasaki, and K. Takeuchi, "Quick-low-density parity check and dynamic threshold voltage optimization in 1X nm triple-level cell NAND flash memory with comprehensive analysis of endurance, retention-time, and temperature variation," *Japanese Journal of Applied Physics* (JJAP), vol. 55, no. 8, pp. 084201-1–084201-10, 2016.

[71] H. Maejima, K. Kanda, S. Fujimura, T. Takagiwa, S. Ozawa, J. Sato, Y. Shindo, M. Sato, N. Kanagawa, and J. Musha, "A 512Gb 3b/Cell 3D flash memory on a 96-word-line-layer technology," in *Proceedings of the IEEE International Solid-State Circuits Conference* (ISSCC), 2018.

[72] D. Kang, W. Jeong, C. Kim, D. Kim, Y. Cho, K. Kang, J. Ryu, K. Kang, S. Lee, W. Kim, H. Lee, J. Yu, N. Choi, D. Jang, J. Ihm, D. Kim, Y. Min, P. Kwak, B. Jung, D. Lee, H. Kim, H. Yang, D. Byeon, K. Park, K. Kyung, and J. Choi, "256Gb 3b/Cell V-NAND Flash Memory with 48 Stacked WL Layers," in *Proceedings of the IEEE International Solid-State Circuits Conference* (ISSCC), 2016.

[73] Micron, "Micron Announces 16 nm 128Gb MLC NAND, SSD in 2014," http://www.anandtech.com/show/7147/micron-announces-16nm-128gb-mlc-nand-ssds-in-2014, 2013.

[74] G. Wu, "Performance and Reliability Study and Exploration of NAND Flash-based Solid State Drives," Virginia Commonwealth University, 2013.

[75] S. S. Hahn, S. Kee, C. Ji, L. P. Chang, I. Yee, L. Shi, C. J. Xue, and J. Kim, "Improving file system performance of mobile storage systems using a decoupled defragmenter," in *Proceedings of the USENIX Annual Technical Conference* (ATC), 2017.

[76] S. Lee, J. Park, and J. Kim, "FlashBench: A workbench for a rapid development of flash-based storage devices," in *Proceedings of the 23rd IEEE International Symposium on Rapid System Prototyping* (RSP), 2012.

[77] R. Rivest, "The MD5 message-digest algorithm," https://tools.ietf.org/html/rfc1321, 1992.

[78] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, “The new ext4 filesystem: current status and future plans,” in *Proceedings of the Linux Symposium*, 2007.

[79] V. Tarasov, E. Zadok, and S. Shepler, “Filebench: A flexible framework for file system benchmarking,” *USENIX Login*, vol. 41, no. 1, pp. 6–12, 2016.

[80] S. Aritome, *NAND flash memory technologies*, Springer, 2015.

[81] F. Courbon, S. Skorobogatov, and C. Woods, title=Reverse engineering flash EEPROM memories using scanning electron microscopy, in *Proceedings of the International Conference on Smart Card Research and Advanced Applications* 2016.

[82] Tech insights, “Intel/Micron 64L 3D NAND Analysis,” https://www.techinsights.com/blog/intelmicron-64l-3d-nand-analysis, 2017.

[83] M. Kim J. Lee, S. Lee, J. Park, J. Kim, “Improving performance and lifetime of large-page NAND storages using erase-free sub-page programming,” in *Proceedings of the Design Automation Conference* (DAC), 2017.

[84] C. M. Compagnoni, A. Ghetti, M. Ghidotti, A. S. Spinelli, and A. Visconti, “Data retention and program/erase sensitivity to the array background pattern in deca-nanometer NAND Flash memories,” *IEEE Transactions on Electron Devices* (TED), vol. 57, no. 1, pp. 321–327, 2009.

[85] Samsung, “Samsung V-NAND technology: Yield more capacity, performance, endurance and power efficiency,” https://studylib.net/doc/8282074/samsung-v-nand-technology, 2014.

[86] Z. Gajda and L. Sekanina, “Reducing the number of transistors in digital circuits using gate-level evolutionary design,” in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007.

[87] Samsung Electronics, “Galaxy S2 Black,” https://www.samsung.com/uk/smartphones/galaxy-s2/GT-I9100LKAXEU/, 2011.

[88] K. Jeffay, D. Smith, A. Moorthy, and J. Anderson, “Proportional Share Scheduling of Operating System Services for Real-Time Applications,” in *Proceedings of the IEEE Real-Time Systems Symposium* (RTSS), 1998.

[89] D. W. Lee, S. Cho, B. W. Kang, S. Park, B. Park, M. K. Cho, K. Ahn, Y. S. Ye, and S. W. Park, “The operation algorithm for improving the reliability of TLC (triple level cell) NAND flash characteristics,” in *Proceedings of the 2011 3rd IEEE International Memory Workshop* (IMW), 2011.

[90] J. Lee, Y. Kim, G. M. Galen, S. Oral, F. Wang, and J. Kim, “A semi-preemptive garbage collector for solid state drives,” in *Pro-*

ceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2011.

[91] S. S. Hahn, S. Lee, and J. Kim, "SOS: Software-based out-of-order scheduling for high-performance NAND flash-based SSDs," in *Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies* (MSTT), 2013.

초 록

최근 몇 년 동안 낸드 플래시 기반 스토리지 시스템은 플래시 메모리이 가지고 있는 여러 고유한 장점과 비트 당 비용 가치의 성공적인 감소로 인해 저장 장치 시장에서 폭발적으로 성장하고 있다. 낸드 플래시 메모리가 다양한 새로운 데이터 중심 애플리케이션에 널리 채택됨에 따라 최신 스토리지 시스템은 높은 데이터 보안 또는 실시간 처리와 같은 새로운 요구 사항을 충족해야한다. 따라서 새로운 요구 사항을 해결하기 위해서는 기존 기법의 최적화 정도를 뛰어넘을 수 있는 새로운 기법의 개발이 필요하다.

본 논문에서는 플래시 스토리지 시스템이 낸드 플래시 메모리의 성능과 안정성을 손상시키지 않으면서도 높은 데이터 보안 및 실시간 처리를 지원할 수 있는 다양한 최적화 기술을 제안한다. 우리의 기술은 실제 낸드 소자를 활용한 포괄적인 특성 연구의 결과를 기반으로 고안되었다. 플래시 스토리지 시스템의 읽기 서비스 시간은 크게 두 가지 요인에 의해 변동되고 지연 될 수 있다. 하나는 최적의 읽기 기준 전압을 검색하기 위하여 반복적으로 읽기 작업을 수행하는 읽기 재시도 동작이고 다른 하나는 읽기 작업과 이전에 수행중인 I/O 요청 간의 충돌이다. 또한 기존 데이터 삭제 기술은 저장된 데이터를 물리적으로 파괴하기 때문에 성능이나 신뢰성의 열화를 피할 수 없다. 따라서 중요한 데이터를 안정적으로 보호하려면 물리적으로 데이터를 변경하지 않고도 동등한 보안성을 보장할 수 있는 새로운 접근 방식이 필요하다.

본 논문에서는 낸드 소자에 대한 평가 결과를 바탕으로 스토리지 시장의 새로운 요구 사항을 충족시키는 최적화 기술을 제안한다.

첫째로, 낸드 블록의 마모 상태를 정확하게 나타내는 새로운 낸드 노화 지수를 기반으로 하는 실제 읽기 재시도 완화 기술을 개발하였다. 읽기 재시도 작업은 낸드 플래시 메모리의 마모와 밀접한 관련이 있으므로 개별 낸드 블록의 정확한 노화 특성을 아는 것은 읽기 재 시도를 최소화하기위한 필수 전제 조건이다. 사전 평가를 통하여 우리는 기존의 P/E 사이클 기반 노화 지수(`PeWear`)가 낸드 블록의 실제 노화 상태를 측정하기에 부적절함을 확인하였다. `PeWear`

의 한계를 극복하기 위해 실제 3D TLC 플래시 칩을 사용한 광범위한 특성 평가를 기반으로 새로운 낸드 노화 지수 RealWear를 제안하였다. 낸드 셀의 마모에 영향을 줄 수있는 여러 변수를 고려하여 RealWear는 런타임 동안 낸드 블록의 실제 마모 상태를 정확하게 나타낼 수 있다. RealWear 의 값을 바탕으로 다양한 동작 조건을 고려한 최적의 읽기 기준 전압 테이블을 구성하고 이를 활용하기 위하여 RealWear 특정 모듈을 사용하는 RealWear-aware FTL, rFTL 을 구현하였다. 성능 열화를 유발하는 반복적인 검색 작업없이 최적의 읽기 기준 전압을 직접 제공 할 수 있기 때문에 rFTL 은 읽기 지연 시간 변동을 크게 완화하여 최악의 시나리오에서도 읽기 지연 시간을 최대 2 번의 읽기 재시도 작업으로 제한 할 수 있다.

둘째로, 3D 낸드 플래시 메모리의 특성을 기반으로 우선 순위에 따라 suspend/resume (일시 선점/재개) 동작을 지원하는 기법을 제안한다 (pSR). 기존의 선점 기술과 달리 pSR 은 사전에 정의된 *safe* 지점에서만 선점 명령을 허용하여 일시 선점/재개 작업의 수를 제한하므로 과도한 선점 작업으로 인한 신뢰성 열화를 효과적으로 피할 수 있다. 정확한 낸드 오류 모델을 기반으로 pSR 은 부분적으로 읽기 작업이 진행중인 I/O 작업을 즉시 중단할 수 있도록 허용한다. pSR 은 플래시 작업 간의 충돌을 효율적으로 제거하기 때문에 예측할 수 없는 긴 지연없이 우선 순위가 높은 읽기 요청을 적시에 제공 할 수 있습니다.

마지막으로, 고 용량의 3D 낸드 플래시 메모리를 위해 새로운 데이터 삭제 기술인 evanesco 를 제안한다. 저장된 데이터를 물리적으로 파괴하는 기존 기술과 달리 evanesco 는 저장된 데이터에 대한 접근을 차단함으로써 저장된 데이터의 보안을 보장한다. 플래시 메모리 칩에서 제공되는 여분의 플래시 셀을 활용하여 evanesco 는 페이지 및 블록 단위로 삭제된 (플래시 칩에는 남아있는) 데이터에 대한 접근을 비활성화하는 두 개의 새로운 플래시 명령 (pLock 및 bLock)을 효율적으로 지원한다. 잠긴 페이지 (또는 블록)는 물리적으로 데이터가 지워진 후에 만 잠금을 해제 할 수 있으므로 evanesco 는 다양한 위협 모델에 대해서도 강력한 보안을 보장한다. 이 기법의 효과를 평가하기 위해 evanesco 를 지원하는 에뮬레이트 플래시 스토리지 시스템인 SecureSSD 를 구현하였다.

실험 결과를 바탕으로 SecureSSD 가 작은 성능 오버 헤드와 신뢰성 저하없이 데이터 삭제를 효과적으로 지원할 수 있음을 확인하였다.

본 논문에서 제안한 기법들은 저장장치 프로토타입 및 공개 낸드 플래시 저장장치 개발/평가 환경에 구현되었으며, 실제 응용 프로그램에서 수집한 다양한 벤치 마크 도구 및 I/O 트레이스들을(traces) 사용하여 그 유용성을 검증하였다. 실험 결과에 따르면 제안된 기술을 통해 플래시 스토리지 시스템을 적시에 관리하고 악의적 인 공격으로부터 민감한 데이터를 안전하게 보호할 수 있음을 확인하였다. 또한, 당사의 기술은 개별 낸드 블록의 정확한 상태를 완전히 활용하여 성능과 수명을 모두 향상시킬 수 있었다.

키워드: 낸드 플래시 메모리, 플래시 변환 계층, 낸드 플래시 기반 저장장치, 내장형 시스템, 데이타 보안, 실시간성, 성능 최적화, 수명 최적화
학번: 2015-31049

감사의 글

지난 십여년 동안의 치열했던 회사 생활을 잠시 뒤로하고 늦은 나이에 뒤늦게 다시 학교로 돌아온지 벌써 5년의 시간이 흘렀습니다. 학교에서 보낸 시간들은 제 인생의 다른 어떤 기간과도 비교할 수 없을 만큼 소중한 시간이었고, 이곳에서의 경험들은 앞으로의 제 인생에 큰 힘이 될 것으로 생각합니다. 돌이켜보면 포기하고 싶으만큼 힘들 때도 많았지만 그럴때마다 항상 따뜻하게 격려해 주시고 도와 주셨던 고만운 분들 덕분에 지금의 좋은 성과를 얻을 수 있게 된 것 같습니다.

제 박사학위논문이 논문이 완성되기까지 정말 많은 분들의 도움이 있었습니다. 이 지면을 빌어 그분들께 감사의 인사를 드리고자 합니다.

먼저 지도교수님이신 김지홍 교수님께 진심으로 감사의 말씀을 드립니다. 교수님의 지도 하에 박사과정을 할 수 있었던 것은 제 인생에서 정말 큰 행운이었습니다. 시스템에 대한 지식이 일천한 저를 제자로 받아주시고 스스로 연구를 진행하고 그 결과를 논문으로 작성하여 발표까지 할 수 있는 한 명의 연구자가 될 수 있도록 저를 지켜봐주시고 지도해 주신 것 감사드립니다. 돌이켜보면 교수님께 전공 지식 뿐 만 아니라 인간과 삶을 대하는 진정한 자세도 함께 배운 것 같습니다. 앞으로도 소중한 가르침 잊지 않고 마음속 깊이 간직하며 삶 속에서 실천할 수 있도록 노력하겠습니다.

또한 부족한 저의 논문을 심사해주신 유승주 교수님, 이재욱 교수님, 김진수 교수님, 그리고 이성진 교수님께도 감사의 말을 드리고 싶습니다. 바쁘신 와중에도 귀중한 시간을 내어 논문 자격 심사부터 최종 학위 심사에 이르기까지 아낌없이 조언해주시고 지도해 주신 것이 저의 학위 논문 완성에 큰 힘이 되었습니다.

제가 학업을 시작하고 이어갈 수 있도록 도와 주신 김기남 사장님, 최정혁 부사장님, 이성수 전무님, 최기환 상무님, 이진엽 상무님, 그리고 송기환 상무님

께도 감사드립니다. 박사 학위 진학이라는 좋은 기회를 주시고 다른 어려움없이 학업에 전념할 수 있도록 전폭적으로 지원해주셔서 마음 깊이 감사드립니다. 제가 보고 배운 것들을 바탕으로 복귀하여 회사에 도움이 될 수 있도록 노력하겠습니다.

제가 학교 생활에 잘 적응하고 무사히 학업을 마칠 수 있었던 데에는 연구실의 여러 동료들의 도움이 컸습니다. 박지성 박사, 김태진 박사, 한상욱 박사, 그리고 저와 같이 학술연수를 하였던 송영선 책임, 심영섭 수석, 홍두원 수석에게 많은 도움을 받았습니다. 함께 연구하면서 도움을 준 이재훈, 유장석, 한승욱, 천명준, 신슬기, 조유현, 김윤아, 조건희, 이두솔, 정일보, 이재용, 이두솔, 신재민, 심준석, 이상구, 한보경에게도 고맙다는 말을 전하고 싶습니다. 여러분들의 앞길에 항상 행운이 함께 하길 기원하며, 꼭 인생의 목표를 이루길 바랍니다.

제 인생의 사표이신 연세대학교 윤일구 교수님. 항상 존경하고 사랑합니다. 기쁜일, 힘든일 함께했던 반도체공학 연구실의 동료 및 후배 여러분들에게도 감사의 말을 전합니다. 제 인생의 행운으로 다가온 소중한 친구들. 대학생때 만나 지금껏 함께한 성환이, 화경이, 신이, 장혁이, 30년 지기 원숙이, 사회에서 만났지만 가족과도 같은 미선이, 승엽이, 혜정이, 영현이와 은명이에게도 지면을 빌려 고마움을 표하고 싶습니다.

제가 학업을 마칠때까지 따뜻한 관심과 기대를 가지고 기다려주신 사랑하는 가족들과 지인들에게도 감사드립니다. 아들을 항상 자랑스러워하시고 묵묵히 지켜봐 주시는 아버님, 어머님. 두 분께 받은 사랑을 조금이라도 보답할 수 있도록 언제나 최선을 다하겠습니다. 감사합니다. 그리고 사랑합니다. 그리고 제가 마땅히 해야했지만 학업을 핑계로 하지 못했던 일들을 대신 도와주셨고, 역시 저를 아들 이상으로 자랑스러워 하셨던 장인어른, 장모님께도 감사드립니다. 회사에 있을 때나 학교에 있을 때나 늘 바빠서 많은 시간을 같이 보내주지 못하는 빵점짜리 아빠를 좋아해주는 세상에서 가장 사랑스러운 딸 서연이에게도 미안하고 고맙다는 말을 꼭 하고 싶습니다. 못난 오빠를 항상 응원해주는 동생 은경이와 그 가족들, 그리고 친 동생같은 처제, 처남들에게도 감사의 마음을 전하고 싶습니다. 마지막으로 어린 나이에 부족한 저를 만나 오랜 시간동안

항상 밝은 모습으로 저를 지지해주었고, 제가 여기까지 오는데 가장 큰 도움을 준 주영이에게 제가 드릴 수 있는 가장 큰 감사와 사랑의 마음을 전합니다.
감사합니다.

2020년 8월

김 명 석