NAND Flash Errors: A Modern Survey

Proceedings of the IEEE, Sept. 2017

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD’s reliability and lifetime.

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixing Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642

Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery

YU CAI, SAUGATA GHOSE
Carnegie Mellon University

ERICH F. HARATSCH
Seagate Technology

YIXIN LUO
Carnegie Mellon University

ONUR MUTLU
ETH Zürich and Carnegie Mellon University
Agenda

- Background, Motivation and Approach
- Experimental Characterization Methodology
- **Error Analysis and Management**
  - Main Characterization Results
  - Retention-Aware Error Management
  - Threshold Voltage and Program Interference Analysis
  - Read Reference Voltage Prediction
  - Neighbor-Assisted Error Correction
  - Read Disturb Error Handling
  - Retention Error Handling
  - Large Scale Field Analysis
  - 3D NAND Flash Memory Reliability
- Summary
3D NAND Flash Memory
**HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness**

Yixin Luo†, Saugata Ghose†, Yu Cai‡, Erich F. Haratsch‡, and Onur Mutlu§

†Carnegie Mellon University ‡Seagate Technology §ETH Zürich
Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu, "Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation"

Abstract

POMACS Journal Version (same content, different format)

Slides (pptx) (pdf)
Flash lifetime decreases in each generation despite increased ECC strength.
Planar vs. 3D NAND Flash Memory

**Scaling**
- **Planar NAND Flash Memory**
  - Reduce flash cell size,
  - Reduce distance b/w cells
- **3D NAND Flash Memory**
  - Increase # of layers

**Reliability**
- **Planar NAND Flash Memory**
  - Scaling hurts reliability
- **3D NAND Flash Memory**
  - Not well studied!
Charge Trap Based 3D Flash Cell

- Cross-section of a charge trap transistor
2D vs. 3D Flash Cell Design

2D Floating-Gate Cell

3D Charge-Trap Cell
3D NAND Flash Memory Organization

Fig. 43. Organization of flash cells in an $M$-layer 3D charge trap NAND flash memory chip, where each block consists of $M$ wordlines and $N$ bitlines.
More Background and State-of-the-Art


Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery

YU CAI, SAUGATA GHOSE
Carnegie Mellon University

ERICH F. HARATSCH
Seagate Technology

YIXIN LUO
Carnegie Mellon University

ONUR MUTLU
ETH Zürich and Carnegie Mellon University
3D vs. Planar NAND Errors: Comparison

Table 4. Changes in behavior of different types of errors in 3D NAND flash memory, compared to planar (i.e., two-dimensional) NAND flash memory. See Section 6.2 for a detailed discussion.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Change in 3D vs. Planar</th>
</tr>
</thead>
<tbody>
<tr>
<td>P/E Cycling</td>
<td>3D is less susceptible, due to current use of charge trap</td>
</tr>
<tr>
<td>(Section 3.1)</td>
<td>transistors for flash cells</td>
</tr>
<tr>
<td>Program</td>
<td>3D is less susceptible for now, due to use of one-shot</td>
</tr>
<tr>
<td>(Section 3.2)</td>
<td>programming (see Section 2.4)</td>
</tr>
<tr>
<td>Cell-to-Cell Interference</td>
<td>3D is less susceptible for now, due to larger manufacturing</td>
</tr>
<tr>
<td>(Section 3.3)</td>
<td>process technology</td>
</tr>
<tr>
<td>Data Retention</td>
<td>3D is more susceptible, due to early retention loss</td>
</tr>
<tr>
<td>(Section 3.4)</td>
<td></td>
</tr>
<tr>
<td>Read Disturb</td>
<td>3D is less susceptible for now, due to larger manufacturing</td>
</tr>
<tr>
<td>(Section 3.5)</td>
<td>process technology</td>
</tr>
</tbody>
</table>
Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation

Yixin Luo    Saugata Ghose    Yu Cai    Erich F. Haratsch    Onur Mutlu
Executive Summary

• Problem: 3D NAND error characteristics are **not well studied**
• Goal: *Understand & mitigate* 3D NAND errors to improve lifetime

• **Contribution 1: Characterize** real 3D NAND flash chips
  • *Process variation:* $21 \times$ error rate difference across layers
  • *Early retention loss:* Error rate increases by $10 \times$ after 3 hours
  • *Retention interference:* Not observed before in planar NAND

• **Contribution 2: Model** RBER and threshold voltage
  • *RBER (raw bit error rate) variation model*
  • *Retention loss model*

• **Contribution 3: Mitigate** 3D NAND flash errors
  • *LaVAR:* Layer Variation Aware Reading
  • *LI-RAID:* Layer-Interleaved RAID
  • *ReMAR:* Retention Model Aware Reading
  • Improve flash lifetime by $1.85 \times$ or reduce ECC overhead by $78.9\%$
Agenda

• Background & Introduction

• Contribution 1: Characterize real 3D NAND flash chips

• Contribution 2: Model RBER and threshold voltage

• Contribution 3: Mitigate 3D NAND flash errors

• Conclusion
Agenda

• Background & Introduction

• Contribution 1: Characterize real 3D NAND flash chips
  • Process variation
  • Early retention loss
  • Retention interference

• Contribution 2: Model RBER and threshold voltage

• Contribution 3: Mitigate 3D NAND flash errors

• Conclusion
Flash cells on different layers may have different error characteristics.
Characterization Methodology

- Modified firmware version in the flash controller
  - Controls the read reference voltage of the flash chip
  - Bypasses ECC to get raw data (with raw bit errors)
- Analysis and post-processing of the data on the server
Layer-to-Layer Process Variation

The diagram illustrates the variation in RBER (Radiation Bit Error Rate) across different normalized layer numbers. The graph shows two curves: one for MSB (Most Significant Bit) and another for LSB (Least Significant Bit). The y-axis represents RBER ranging from $10^{-4}$ to $4 	imes 10^{-4}$, while the x-axis represents the normalized layer number from 0 to 100.

Max RBER is indicated by the peak in the graph, with a significant difference of 21 times between the MSB and LSB curves. Additionally, there is a reduced RBER by a factor of 2.4 times, as highlighted by the green arrow.
Layer-to-Layer Process Variation

Large RBER variation across layers and LSB-MSB pages
Retention Loss Phenomenon

Planar NAND Cell

- Control Gate
- Gate Oxide
- Floating Gate (Conductor)
- Tunnel Oxide
- Source
- Drain
- Substrate

3D NAND Cell

- Control Gate
- Charge Trap (Insulator)
- Gate Oxide
- Tunnel Oxide
- Source
- Drain
- Substrate

Most dominant type of error in planar NAND. Is this true for 3D NAND as well?
Early Retention Loss

Retention errors increase quickly immediately after programming
Characterization Summary

• **Layer-to-layer process variation**
  • Large RBER variation across layers and LSB-MSB pages
  • → Need new mechanisms to tolerate RBER variation!

• **Early retention loss**
  • RBER increases quickly after programming
  • → Need new mechanisms to tolerate retention errors!

• **Retention interference**
  • Amount of retention loss correlated with neighbor cells’ states
  • → Need new mechanisms to tolerate retention interference!

• **More threshold voltage and RBER results in the paper:**
  3D NAND P/E cycling, program interference, read disturb, read variation, bitline-to-bitline process variation

• **Our approach** based on insights developed via our experimental characterization: Develop **error models**, and build online **error mitigation mechanisms** using the models
• Background & Introduction

• Contribution 1: Characterize real 3D NAND flash chips

• Contribution 2: Model RBER and threshold voltage
  • Retention loss model
  • RBER variation model

• Contribution 3: Mitigate 3D NAND flash errors

• Conclusion
What Do We Model?

Read Reference Voltages

Threshold Voltage Distribution

Raw Bit Errors

MSB  LSB

\[ V_a \, V_b \, V_c \]

Threshold Voltage \( (V_{th}) \)

Probability
Optimal Read Reference Voltage

Probability

Threshold Voltage ($V_{th}$)

$V_a$  $V_b$  $V_c$

Raw Bit Errors
Retention Loss Model

Early retention loss can be modeled as a simple linear function of log(retention time)
Retention Loss Model

• Goal: Develop a simple linear model that can be used online

• Models
  • Optimal read reference voltage ($V_b$ and $V_c$)
  • Raw bit error rate ($\log(RBER)$)
  • Mean and standard deviation of threshold voltage distribution ($\mu$ and $\sigma$)

• As a function of
  • Retention time ($\log(t)$)
  • P/E cycle count (PEC)

• e.g., $V_{opt} = (\alpha \times PEC + \beta) \times \log(t) + \gamma \times PEC + \delta$

• Model error <1 step for $V_b$ and $V_c$
• Adjusted $R^2 > 89\%$
Variation-agnostic $V_{\text{opt}}$
- Same $V_{\text{ref}}$ for all layers optimized for the entire block

RBER distribution follows gamma distribution

KL-divergence error = 0.09
Agenda

- Background & Introduction
- Contribution 1: Characterize real 3D NAND flash chips
- Contribution 2: Model RBER and threshold voltage
- Contribution 3: Mitigate 3D NAND flash errors
  - LaVAR: Layer Variation Aware Reading
  - Li-RAID: Layer-Interleaved RAID
  - ReMAR: Retention Model Aware Reading
- Conclusion
LaVAR: Layer Variation Aware Reading

- **Layer-to-layer process variation**
  - Error characteristics are different in each layer
- **Goal**: Adjust read reference voltage for each layer

- **Key Idea**: Learn a *voltage offset (Offset)* for each layer
  - \[ V_{\text{Layer aware}}^{\text{opt}} = V_{\text{Layer agnostic}}^{\text{opt}} + \text{Offset} \]

- **Mechanism**
  - **Offset**: Learned once for each chip & stored in a table
    - Uses \((2 \times \text{Layers})\) Bytes memory per chip
  - \( V_{\text{Layer agnostic}}^{\text{opt}} \): Predicted by any existing \( V_{\text{opt}} \) model
    - *E.g.*, ReMAR [Luo+Sigmetrics’18], HeatWatch [Luo+HPCA’18], OFCM [Luo+JSAC’16], ARVT [Papandreou+GLSVLSI’14]

- Reduces RBER on average by **43%**
  (based on our characterization data)
LI-RAID: Layer-Interleaved RAID

- **Layer-to-layer process variation**
  - Worst-case RBER much higher than average RBER
- **Goal:** Significantly reduce worst-case RBER

- **Key Idea**
  - Group flash pages on *less reliable layers* with pages on *more reliable layers*
  - Group *MSB pages* with *LSB pages*

- **Mechanism**
  - Reorganize RAID layout to eliminate worst-case RBER
  - *<0.8%* storage overhead
Conventional RAID

<table>
<thead>
<tr>
<th>Wordline #</th>
<th>Layer #</th>
<th>Page</th>
<th>Chip 0</th>
<th>Chip 1</th>
<th>Chip 2</th>
<th>Chip 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>MSB</td>
<td>Group 0</td>
<td>Group 0</td>
<td>Group 0</td>
<td>Group 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>LSB</td>
<td>Group 1</td>
<td>Group 1</td>
<td>Group 1</td>
<td>Group 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>MSB</td>
<td>Group 2</td>
<td>Group 2</td>
<td>Group 2</td>
<td>Group 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>LSB</td>
<td>Group 3</td>
<td>Group 3</td>
<td>Group 3</td>
<td>Group 3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>MSB</td>
<td>Group 4</td>
<td>Group 4</td>
<td>Group 4</td>
<td>Group 4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>LSB</td>
<td>Group 5</td>
<td>Group 5</td>
<td>Group 5</td>
<td>Group 5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>MSB</td>
<td>Group 6</td>
<td>Group 6</td>
<td>Group 6</td>
<td>Group 6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>LSB</td>
<td>Group 7</td>
<td>Group 7</td>
<td>Group 7</td>
<td>Group 7</td>
</tr>
</tbody>
</table>

Worst-case RBER in any layer limits the lifetime of conventional RAID
LI-RAID: Layer-Interleaved RAID

<table>
<thead>
<tr>
<th>Wordline #</th>
<th>Layer #</th>
<th>Page</th>
<th>Chip 0</th>
<th>Chip 1</th>
<th>Chip 2</th>
<th>Chip 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>MSB</td>
<td>Group 0</td>
<td>Blank</td>
<td>Group 4</td>
<td>Group 3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>LSB</td>
<td>Group 1</td>
<td>Blank</td>
<td>Group 5</td>
<td>Group 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>MSB</td>
<td>Group 2</td>
<td>Group 1</td>
<td>Blank</td>
<td>Group 5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>LSB</td>
<td>Group 3</td>
<td>Group 0</td>
<td>Blank</td>
<td>Group 4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>MSB</td>
<td>Group 4</td>
<td>Group 3</td>
<td>Group 0</td>
<td>Blank</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>LSB</td>
<td>Group 5</td>
<td>Group 2</td>
<td>Group 1</td>
<td>Blank</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>MSB</td>
<td>Blank</td>
<td>Group 5</td>
<td>Group 2</td>
<td>Group 1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>LSB</td>
<td>Blank</td>
<td>Group 4</td>
<td>Group 3</td>
<td>Group 0</td>
</tr>
</tbody>
</table>

Any page with worst-case RBER can be corrected by other reliable pages in the RAID group.
LI-RAID: Layer-Interleaved RAID

• **Layer-to-layer process variation**
  • Worst-case RBER much higher than average RBER

• **Goal:** Significantly reduce worst-case RBER

• **Key Idea**
  • Group flash pages on *less reliable layers* with pages on *more reliable layers*
  • Group *MSB pages* with *LSB pages*

• **Mechanism**
  • Reorganize RAID layout to eliminate worst-case RBER
  • *<0.8%* storage overhead

• Reduces worst-case RBER by *66.9%* (based on our characterization data)
ReMAR: Retention Model Aware Reading

- **Early retention loss**
  - Threshold voltage shifts quickly after programming
- **Goal:** Adjust read reference voltages based on retention loss

- **Key Idea:** Learn and use a retention loss model online

- **Mechanism**
  - Periodically characterize and learn retention loss model online
  - Retention time = Read timestamp - Write timestamp
    - Uses **800 KB memory to store program time of each block**
  - Predict retention-aware $V_{opt}$ using the model

- Reduces RBER on average by **51.9%** (based on our characterization data)
Impact on System Reliability

LaVAR, LI-RAID, and ReMAR improve flash lifetime or reduce ECC overhead significantly
Error Mitigation Techniques Summary

- **LaVAR: Layer Variation Aware Reading**
  - Learn a $V_{\text{opt}}$ offset for each layer and apply *layer-aware $V_{\text{opt}}$*

- **LI-RAID: Layer-Interleaved RAID**
  - Group flash pages on *less reliable layers* with pages on *more reliable layers*
  - Group *MSB pages* with *LSB pages*

- **ReMAR: Retention Model Aware Reading**
  - Learn retention loss model and apply *retention-aware $V_{\text{opt}}$*

- **Benefits:**
  - Improve flash lifetime by $1.85 \times$ or reduce ECC overhead by $78.9\%$

- **ReNAC (in paper):** Rread a failed page using $V_{\text{opt}}$ based on the *retention interference* induced by neighbor cell
Agenda

• Background & Introduction

• Contribution 1: Characterize real 3D NAND flash chips

• Contribution 2: Model RBER and threshold voltage

• Contribution 3: Mitigate 3D NAND flash errors

• Conclusion
Conclusion

• Problem: 3D NAND error characteristics are **not well studied**
• Goal: *Understand* & *mitigate* 3D NAND errors to improve lifetime

• **Contribution 1: Characterize** real 3D NAND flash chips
  • *Process variation:* $21 \times \text{error rate difference across layers}$
  • *Early retention loss:* Error rate increases by $10 \times$ after 3 hours
  • *Retention interference:* *Not observed before* in planar NAND

• **Contribution 2: Model** RBER and threshold voltage
  • *RBER (raw bit error rate) variation model*
  • *Retention loss model*

• **Contribution 3: Mitigate** 3D NAND flash errors
  • *LaVAR*: Layer Variation Aware Reading
  • *LI-RAID*: Layer-Interleaved RAID
  • *ReMAR*: Retention Model Aware Reading
  • Improve flash lifetime by $1.85 \times$ or reduce ECC overhead by **78.9%**
Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation

Yixin Luo  Saugata Ghose  Yu Cai  Erich F. Haratsch  Onur Mutlu

Carnegie Mellon  SK hynix  ETH Zürich  SAFARI  SEAGATE
Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu, "Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation"

[Abstract]
[POMACS Journal Version (same content, different format)]
[Slides (pptx) (pdf)]
One More Idea
Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management

Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi*, Onur Mutlu
Carnegie Mellon University, *Dankook University
Executive Summary

• Flash memory can achieve **50x endurance improvement by relaxing retention time using refresh** [Cai+ ICCD ’12]

• **Problem**: Frequent refresh consumes the majority of endurance improvement

• **Goal**: Reduce refresh overhead to increase flash memory lifetime

• **Key Observation**: Refresh is unnecessary for **write-hot data**

• **Key Ideas of Write-hotness Aware Retention Management (WARM)**
  - Physically partition **write-hot pages and write-cold pages** within the flash drive
  - **Apply different policies** (garbage collection, wear-leveling, refresh) to each group

• **Key Results**
  - WARM w/o refresh **improves lifetime by 3.24x**
  - WARM w/ adaptive refresh **improves lifetime by 12.9x** (1.21x over refresh only)
Conventional Write-Hotness Oblivious Management

Unable to relax retention time for blocks with write-hot and cold pages
Key Idea: Write-Hotness Aware Management

Can relax retention time for blocks with write-hot pages only

```
<table>
<thead>
<tr>
<th>Flash Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hot Page 1</td>
</tr>
<tr>
<td>Hot Page 1</td>
</tr>
<tr>
<td>Hot Page 4</td>
</tr>
<tr>
<td>Hot Page 4</td>
</tr>
<tr>
<td>Hot Page 1</td>
</tr>
<tr>
<td>Hot Page 4</td>
</tr>
<tr>
<td>Hot Page 1</td>
</tr>
</tbody>
</table>
```
Write-Hotness Aware Retention Management

- Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu, "WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management"
  

---

WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management

Yixin Luo  
yixinluo@cs.cmu.edu

Yu Cai  
yucaicai@gmail.com

Saugata Ghose  
ghose@cmu.edu

Jongmoo Choi†  
choijm@dankook.ac.kr

Onur Mutlu  
onur@cmu.edu
HeatWatch

Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness

Yixin Luo  Saugata Ghose  Yu Cai  Erich F. Haratsch  Onur Mutlu

Carnegie Mellon  SK hynix  ETH Zürich  SAFARI  SEAGATE
Storage Technology Drivers - 2018

3D NAND Flash Memory

Stacked layers
Executive Summary

• 3D NAND flash memory susceptible to retention errors
  • Charge leaks out of flash cell
  • Two unreported factors: self-recovery and temperature

• We study self-recovery and temperature effects

• Experimental characterization of real 3D NAND chips

• Unified Self-Recovery and Temperature (URT) Model
  • Predicts impact of retention loss, wearout, self-recovery, temperature on flash cell voltage
  • Low prediction error rate: 4.9%

• We develop a new technique to improve flash reliability

• HeatWatch
  • Uses URT model to find optimal read voltages for 3D NAND flash
  • Improves flash lifetime by 3.85x
Agenda

- Background, Motivation and Approach
- Experimental Characterization Methodology
- Error Analysis and Management
  - Main Characterization Results
  - Retention-Aware Error Management
  - Threshold Voltage and Program Interference Analysis
  - Read Reference Voltage Prediction
  - Neighbor-Assisted Error Correction
  - Read Disturb Error Handling
  - Retention Error Handling
  - Large Scale Field Analysis
  - 3D NAND Flash Memory Reliability
- Summary
Summary of Key Works


Meza+, “A Large-Scale Study of Flash Memory Errors in the Field,” SIGMETRICS 2015.


NAND Flash Vulnerabilities [HPCA’17]

Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques

Yu Cai†, Saugata Ghose†, Yixin Luo††, Ken Mai†, Onur Mutlu§†, Erich F. Haratsch‡
†Carnegie Mellon University ‡Seagate Technology §ETH Zürich

Modern NAND flash memory chips provide high density by storing two bits of data in each flash cell, called a multi-level cell (MLC). An MLC partitions the threshold voltage range of a flash cell into four voltage states. When a flash cell is programmed, a high voltage is applied to the cell. Due to parasitic capacitance coupling between flash cells that are physically close to each other, flash cell programming can lead to cell-to-cell program interference, which introduces errors into neighboring flash cells. In order to reduce the impact of cell-to-cell interference on the reliability of MLC NAND flash memory, flash manufacturers adopt a two-step programming method, which programs the MLC in two separate steps. First, the flash memory partially programs the least significant bit of the MLC to some intermediate threshold voltage. Second, it programs the most significant bit to bring the MLC up to its full voltage state.

In this paper, we demonstrate that two-step programming exposes new reliability and security vulnerabilities. We expe-

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD’s reliability and lifetime.

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642
Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery

YU CAI, SAUGATA GHOSE
Carnegie Mellon University

ERICH F. HARATSCH
Seagate Technology

YIXIN LUO
Carnegie Mellon University

ONUR MUTLU
ETH Zürich and Carnegie Mellon University
Modern SSD Architecture
MQSim [FAST 2018]

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,

"MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices"


[Slides (pptx) (pdf)]
[Source Code]
FLIN [ISCA 2018]

- Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan G. Luna and Onur Mutlu,

"FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives"


[Slides (pptx) (pdf)] [Lightning Talk Slides (pptx) (pdf)] [Lightning Talk Video]

---

**FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives**

Arash Tavakkol† Mohammad Sadrosadati† Saugata Ghose† Jeremie S. Kim†† Yixin Luo†
Yaohua Wang†§ Nika Mansouri Ghiasi† Lois Orosa†* Juan Gómez-Luna† Onur Mutlu††
†ETH Zürich ††Carnegie Mellon University §NUDT *Unicamp
Efficient Data Sanitization [ASPLOS 2020]

- Myungsuk Kim, Jisung Park, Geonhee Cho, Yoona Kim, Lois Orosa, Onur Mutlu, and Jihong Kim,

"Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems"


[Slides (pptx) (pdf)]
[Talk Video (20 mins)]

Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems

Myungsuk Kim*
morssola75@davinci.snu.ac.kr
Seoul National University

Jisung Park*
jisung.park@inf.ethz.ch
ETH Zürich & Seoul National University

Geonhee Cho
ghcho@davinci.snu.ac.kr
Seoul National University

Yoona Kim
yoonakim@davinci.snu.ac.kr
Seoul National University

Lois Orosa
lois.orosa@inf.ethz.ch
ETH Zürich

Onur Mutlu
omutlu@gmail.com
ETH Zürich

Jihong Kim
jihong@davinci.snu.ac.kr
Seoul National University
Optimizing Read-Retry [ASPLOS 2021]

- Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu,

"Reducing Solid-State Drive Read Latency by Optimizing Read-Retry"
[2-page Extended Abstract]
[Short Talk Slides (pptx) (pdf)]
[Full Talk Slides (pptx) (pdf)]
[Short Talk Video (5 mins)]
[Full Talk Video (19 mins)]

Reducing Solid-State Drive Read Latency by Optimizing Read-Retry

Jisung Park\textsuperscript{1}  Myungsuk Kim\textsuperscript{2,3}  Myoungjun Chun\textsuperscript{2}  Lois Orosa\textsuperscript{1}  Jihong Kim\textsuperscript{2}  Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich  Switzerland  \textsuperscript{2}Seoul National University  Republic of Korea  \textsuperscript{3}Kyungpook National University  Republic of Korea
DeepSketch: A New Machine Learning-Based Reference Search Technique for Post-Deduplication Delta Compression

Jisung Park¹* Jeonggyun Kim²* Yeseong Kim² Sungjin Lee² Onur Mutlu¹

¹ETH Zürich ²DGIST


[Slides (pptx) (pdf)]
[Talk Video (15 minutes)]
In-Storage Genome Filtering [ASPLOS 2022]

- Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu,

"GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis"


[Talk Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Lightning Talk Video (90 seconds)]
[Talk Video (17 minutes)]

GenStore: A High-Performance In-Storage Processing System for Genome Sequence Analysis

Nika Mansouri Ghiasi\textsuperscript{1} Jisung Park\textsuperscript{1} Harun Mustafa\textsuperscript{1} Jeremie Kim\textsuperscript{1} Ataberk Olgun\textsuperscript{1} Arvid Gollwitzer\textsuperscript{1} Damla Senol Cali\textsuperscript{2} Can Firtina\textsuperscript{1} Haiyu Mao\textsuperscript{1} Nour Almadhoun Alserr\textsuperscript{1} Rachata Ausavarungnirun\textsuperscript{3} Nandita Vijaykumar\textsuperscript{4} Mohammed Alser\textsuperscript{1} Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich \textsuperscript{2}Bionano Genomics \textsuperscript{3}KMUTNB \textsuperscript{4}University of Toronto
ML-Based Hybrid Storage Design [ISCA 2022]

- Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gomez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu,
  "Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning"
  [Slides (pptx) (pdf)]
  [arXiv version]
  [Sibyl Source Code]
  [Talk Video (16 minutes)]

Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning

Gagandeep Singh\textsuperscript{1} Rakesh Nadig\textsuperscript{1} Jisung Park\textsuperscript{1} Rahul Bera\textsuperscript{1} Nastaran Hajinazar\textsuperscript{1}
David Novo\textsuperscript{3} Juan Gómez-Luna\textsuperscript{1} Sander Stuijk\textsuperscript{2} Henk Corporaal\textsuperscript{2} Onur Mutlu\textsuperscript{1}
\textsuperscript{1}ETH Zürich \textsuperscript{2}Eindhoven University of Technology \textsuperscript{3}LIRMM, Univ. Montpellier, CNRS
In-Flash Bulk Bitwise Execution [MICRO 2022]

- Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira, Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez-Luna, Myungsuk Kim, and Onur Mutlu,

"Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory"

Proceedings of the 55th International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, October 2022.

[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Lecture Video (44 minutes)]
[arXiv version]

Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park$\text{§}^\uparrow$ Roknoddin Azizi$\text{§}$ Geraldo F. Oliveira$\text{§}$ Mohammad Sadrosadati$\text{§}$
Rakesh Nadig$\text{§}$ David Novo$\uparrow$ Juan Gómez-Luna$\text{§}$ Myungsuk Kim$\text{‡}$ Onur Mutlu$\text{§}$

$\text{§}$ETH Zürich $\uparrow$ POSTECH $\uparrow$ LIRMM, Univ. Montpellier, CNRS $\text{‡}$Kyungpook National University

Venice [ISCA 2023]

- Rakesh Nadig, Mohammad Sadrosadati, Haiyu Mao, Nika Mansouri Ghiasi, Arash Tavakkol, Jisung Park, Hamid Sarbazi-Azad, Juan Gómez Luna, and Onur Mutlu,

"Venice: Improving Solid-State Drive Parallelism at Low Cost via Conflict-Free Accesses"


[arXiv version]
[Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Lightning Talk Video (3 minutes)]
[Talk Video (14 minutes, including Q&A)]

Venice: Improving Solid-State Drive Parallelism at Low Cost via Conflict-Free Accesses

*Rakesh Nadig$  *Mohammad Sadrosadati$  Haiyu Mao$  Nika Mansouri Ghiasi$
Arash Tavakkol$  Jisung Park$\triangledown$  Hamid Sarbazi-Azad$\dagger$  Juan Gómez Luna$  Onur Mutlu$

$\triangledown$ETH Zürich  $\dagger$Sharif University of Technology  $\dagger$IPM

Agenda

- Overview on SSD Organization
  - Storage Controller & Request Handling
  - NAND Flash Hierarchy
  - NAND Flash Read/Write Operations

- Address Mapping and Garbage Collection

- I/O Scheduling
Modern SSD Architecture

- A modern SSD is a complicated system that consists of multiple cores, HW controllers, DRAM, and NAND flash memory packages.

![SSD Architecture Diagram]

- **SSD Controller**
  - Core
  - HW Flash Ctrl.
  - Request Handler
  - ECC/Randomizer
  - Encryption Engine

- **LPDDR DRAM**: \(0.001 \times 1,024 = 1 \text{ GB}\)

- **NAND Packages**: \(8 \times 128 \text{ GB} = 1 \text{ TB}\)

Another Overview

Host Interface Layer (HIL)

Flash Translation Layer (FTL)
- Data Cache Management
- Address Translation
- GC/WL/Refresh/...

Flash Controller
- ECC
- Randomizer

DRAM
- Host Request Queue
- Write Buffer
- Logical-to-Physical Mappings
- Metadata (e.g., P/E Cycles)

NAND Flash Package

NAND Flash Package

NAND Flash Package
Request Handling: Write

Host Interface Layer (HIL)

Flash Translation Layer (FTL)
- Data Cache Management
- Address Translation
- GC/WL/Refresh/...

Flash Controller
- ECC
- Randomizer

NAND Flash Package

DRAM

Host Request Queue
- Write Buffer
- Logical-to-Physical Mappings
- Metadata (e.g., P/E Cycles)

- Communication with the host operating system (receives & returns requests)
  - Via a certain interface (SATA or NVMe)

- A host I/O request includes
  - Request direction (read or write)
  - Offset (start sector address)
  - Size (number of sectors)
  - Typically aligned by 4 KiB
Request Handling: Write

- **Host Interface Layer (HIL)**
- **Flash Translation Layer (FTL)**
  - Data Cache Management
  - Address Translation
  - GC/WL/Refresh/...

**Flash Controller**
- ECC
- Randomizer

**DRAM**
- Host Request Queue
- Write Buffer
- Logical-to-Physical Mappings
- Metadata (e.g., P/E Cycles)

- **Buffering data to write (read from NAND flash memory)**
  - Essential to reducing write latency
  - Enables flexible I/O scheduling
  - Helpful for improving lifetime (not so likely)

- **Limited size (e.g., tens of MBs)**
  - Needs to ensure data integrity even under sudden power-off
  - Most DRAM capacity is used for L2P mappings
Request Handling: Write

**Host Interface Layer (HIL)**

**Flash Translation Layer (FTL)**
- Data Cache Management
- Address Translation
- GC/WL/Refresh/…

**Flash Controller**
- ECC
- Randomizer

**DRAM**
- Host Request Queue
- Write Buffer
- Logical-to-Physical Mappings
- Metadata (e.g., P/E Cycles)

**Core functionality for out-of-place writes**
- To hide the erase-before-write property

**Needs to maintain L2P mappings**
- Logical Page Address (LPA) → Physical Page Address (PPA)

**Mapping granularity: 4 KiB**
- 4 Bytes for 4 KiB → 0.1% of SSD capacity
Request Handling: Write

- **Garbage collection (GC)**
  - Reclaims free pages
  - Selects a victim block → copies all valid pages → erase the victim block

- **Wear-leveling (WL)**
  - Evenly distributes P/E cycles across NAND flash blocks
  - Hot/cold swapping

- **Data refresh**
  - Refresh pages with long retention ages
Request Handling: Write

- **Randomizer**
  - Scrambling data to write
  - To avoid worst-case data patterns that can lead to significant errors

- **Error-correcting codes (ECC)**
  - Can detect/correct errors: e.g., 72 bits/1 KiB error-correction capability
  - Stores additional parity information together with raw data

- **Issues NAND flash commands**
Request Handling: Read

- Host Interface Layer (HIL)
  - Request Handling: Read
  - Flash Controller
    - ECC
    - Randomizer
    - NAND Flash Packages
  - Flash Translation Layer (FTL)
    - Data Cache Management
    - Address Translation
    - GC/WL/Refresh/…
  - Logical-to-Physical Mappings
  - Metadata (e.g., P/E Cycles)

- DRAM
  - Host Request Queue
  - Write Buffer

- First checks if the request data exists in the write buffer
  - If so, returns the corresponding request immediately with the data

- A host read request can be involved with several pages
  - Such a request can be returned only after all the requested data is ready
Request Handling: Read

- Host Interface Layer (HIL)
- Flash Translation Layer (FTL)
  - Data Cache Management
  - Address Translation
  - GC/WL/Refresh/...

- Flash Controller
  - ECC
  - Randomizer

- NAND Flash Package
- NAND Flash Package
- NAND Flash Package

- DRAM
  - Host Request Queue
  - Write Buffer
  - Logical-to-Physical Mappings
  - Metadata (e.g., P/E Cycles)

- Finds the PPA where the request data is stored from the L2P mapping table
Request Handling: Read

- First reads the raw data from the flash chip
- Performs ECC decoding
- Derandomizes the raw data
- ECC decoding can fail
  - Retries reading of the page w/ adjusted $V_{REF}$
A Flash Cell

- Basically, it is a transistor

![Diagram of a Flash Cell with labels for source (S), substrate, drain (D), control gate (G), and threshold voltage (V_{TH}). The graph shows the relationship between I_D and V_{GS} with two regions: V_{GS} < V_{TH} and V_{GS} > V_{TH}.](image-url)
A Flash Cell

- Basically, it is a transistor
  - w/ a special material: Floating gate (2D) or Charge trap (3D)
A Flash Cell

- Basically, it is a transistor
  - w/ a special material: Floating gate (2D) or Charge trap (3D)
  - Can hold electrons in a non-volatile manner

![Flash Cell Diagram]

- $V_{PGM} = 20\, V$
- $G$ (Control Gate)
- $FG$ (Floating Gate)
- $GND$ (Substrate)
- $S$ (Source)
- $D$ (Drain)
- $I_D$ vs $V_{GS}$
- $V_{TH}$
A Flash Cell

- Basically, it is a transistor
  - w/ a special material: Floating gate (2D) or Charge trap (3D)
  - Can hold electrons in a non-volatile manner
  - Changes the cell’s threshold voltage \((V_{TH})\)

\[ \text{Source} \quad \text{Drain} \quad \text{Substrate} \quad \text{Control Gate} \quad \text{Floating Gate} \quad \text{Tunneling} \quad \text{GND} \]

\[ V_{TH} < V_{REF} \quad V_{TH} > V_{REF} \]

\[ '1' \quad '0' \]
Flash Cell Characteristics

- Multi-leveling: A flash cell can store multiple bits

  ![Flash Cell Programming and Erasing]

  - Program: Inject electrons
  - Erase: Eject electrons

- Retention loss: A cell leaks electrons over time

  ![Retention Loss Example]

- Limited lifetime: A cell wears out after P/E cycling

  ![Limited Lifetime Example]
Multiple (e.g., 128) flash cells are serially connected
Pages and Blocks

- A large number (> 100,000) of cells operate concurrently.

Page = 16 + α KiB

Block = {(# of WL) × (# of bits per cell)} pages
Pages and Blocks (Continued)

- Program and erase: Unidirectional
  - Programming a cell → **Increasing** the cell’s $V_{TH}$
  - Erasing a cell → **Decreasing** the cell’s $V_{TH}$

- Programming a page cannot change ‘0’ cells to ‘1’ cells → **Erase-before-write property**

- Erase unit: Block
  - Increase erase bandwidth
  - Makes in-place write on a page very inefficient → **Out-of-place write**
Planes

- A large number (> 1,000) of blocks share bitlines in a plane.
Planes

- A large number (> 1,000) of blocks share bitlines in a plane.
Planes and Dies

- A die contains multiple (e.g., 2 – 4) planes

![Diagram of a NAND Flash Die showing planes, blocks, and decoders.]

- Planes share decoders: limits internal parallelism (only operations @ the same WL offset)
Threshold Voltage Distribution

- $V_{TH}$ distribution of **cells** in a programmed page/block/chip

- **Why distribution?** Variations across the cells
  - Some cells are more easily programmed or erased

There are $y$ cells whose $V_{TH} = x \cdot V$

![Diagram showing threshold voltage distribution]

- Threshold voltage ($V_{TH}$)
- # of cells
- $V_{REF}$
- Erased (E)
- Programmed

$y$ cells at $V_{TH} = x \cdot V$
Multi-level cell (MLC) technique

- $2^m V_{TH}$ states required to store $m$ bits in a single flash cell

Limited width of the $V_{TH}$ window: Need to

- Make each $V_{TH}$ state narrow
- Guarantee sufficient margins b/w adjacent $V_{TH}$ states
Multi-level cell (MLC) technique

- $2^m \text{V}_{\text{TH}}$ states required to store $m$ bits in a single flash cell

Limited width of the V\textsubscript{TH} window: Need to

- Make each V\textsubscript{TH} state narrow
- Guarantee sufficient margins b/w adjacent V\textsubscript{TH} states
  - V\textsubscript{TH} changes over time after programmed
  - Narrower margins $\rightarrow$ Lower reliability
  - More bits per cell $\rightarrow$ higher density but lower reliability

Error cells

Shifted & widened after programmed
Basic Operation: Page Program

- **String Select Line (SSL)**
- **Wordline**
  - $WL_{k-1}$
  - $WL_k$
  - $WL_{k+1}$

**Target Page**

- **Ground Select Line (GSL)**

**Block**

- $BL_0$
- $BL_1$
- $BL_2$
- $BL_3$
- $BL_{132,095}$
Basic Operation: Page Program

- WL control – All other cells operate as a resistance
Basic Operation: Page Program

- BL control – *Inhibits cells* to not be programmed

![Diagram showing BL control with program and inhibit signals]
Basic Operation: Page Program

- BL control – **Inhibits cells** to not be programmed

![Diagram showing BL control](image_url)

**V\text{PROG}**  **WL\_k**

- **BL\_0** (program, 0)
- **BL\_1** (inhibit, 1)
- **BL\_2** (program, 0)
- **BL\_3** (program, 1)
- **BL\_132,095** (program, 0)

- **To GND**
- **To V\text{CC}**
Basic Operation: Page Program

- **V_{PROG}**
- **WL_k**
- **BL_0**
- **BL_1**
- **BL_2**
- **BL_3**
- **BL_{132,095}

**program**

**inhibit**

To **GND**
To **V_{CC}**
To **GND**
To **V_{CC}**
To **GND**

1 Erased (E)

Threshold voltage (**V_{TH}**)
Basic Operation: Page Program

- **Program (P)**: When **V\textsubscript{PROG}** is applied to **WL\textsubscript{k}**, the cells are programmed.
- **Inhibit (I)**: When an **1** is applied to a BL (e.g., BL\textsubscript{1}), the cell is inhibited.

### Threshold Voltage (V\textsubscript{TH})
- **Erased (E)**: 1
- **Programmed**: 0

### Inhibited Cells
- **V\textsubscript{REF}

### Programmed Cells

# of cells vs. Threshold voltage (V\textsubscript{TH})

---

99
Basic Operation: Page Program

program  inhibit

$V_{\text{PROG}}$ $WL_k$

To $GND$  To $V_{CC}$  To $GND$  To $V_{CC}$  To $GND$

# of cells

$V_{\text{REF}}$

Inhibited cells

Erased (E)

Cells to program

Hard-to-program cells  Easy-to-program cells

Threshold voltage ($V_{TH}$)
Basic Operation: Page Program

- Incremental Step-Pulse Programming (ISPP)

---

**Diagram:**
- **V<sub>PROG</sub>:** Bias voltage applied to the word line (WL) to select cells.
- **BL<sub>0</sub>** to **BL<sub>132,095</sub>:** Bit lines for program, inhibit, and verify operations.
- **0** and **1** states indicated for cells.
- **To GND** and **To V<sub>CC</sub>** connections for bit lines.

---

**Table:**
- **# of cells**
- **Erased (E)**
- **V<sub>REF</sub>**
- **Threshold voltage (V<sub>TH</sub>)**

**Legend:**
- **Inhibited cells**
- **Cells to program**
Basic Operation: Page Program

- Incremental Step-Pulse Programming (ISPP)

![Diagram showing page programming process](attachment:diagram.png)

- Program
- Inhibit

**Threshold voltage** ($V_{TH}$)

**# of cells**

**Erased (E)**

**Inhibited cells**

**Cells to program**

**Inhibit programmed cells**

**V_{REF}**
Basic Operation: Page Program

- Incremental Step-Pulse Programming (ISPP)

![Diagram showing ISPP process]

- Program
- Inhibit

V_{PROG1} WL_k

Programmed cells

Inhibit programmed cells

Erased (E)

Cells to program

Inhibited cells

Threshold voltage (V_{TH})

# of cells

V_{REF}
Basic Operation: Page Program

- Incremental Step-Pulse Programming (ISPP)

### Diagram

- **Program**: BL<sub>0</sub>, BL<sub>1</sub>, BL<sub>2</sub>, BL<sub>3</sub>, BL<sub>132,095</sub>
- **Inhibit**: WL<sub>k</sub>, V<sub>PROG</sub>

#### Connections
- **To GND**: BL<sub>0</sub>, BL<sub>1</sub>, BL<sub>2</sub>, BL<sub>132,095</sub>
- **To V<sub>CC</sub>**: BL<sub>1</sub>, BL<sub>2</sub>, BL<sub>3</sub>

#### Voltage Levels
- **V<sub>REF</sub>**
- **V<sub>PROG</sub>**

#### States
- **Erased (E)**: 1
- **Inhibited cells**: 0
- **Programmed**: 0

#### Threshold Voltage (V<sub>TH</sub>)

### Table

<table>
<thead>
<tr>
<th>Erased (E)</th>
<th>Inhibited cells</th>
<th>Programmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Basic Operation: Page Read

- WL control – All other cells operate as a resistance

![-diagram showing WL control and different states of cells-]

- # of cells
- Erased (E)
- Programmed
- Threshold voltage ($V_{TH}$)
- $V_{REF}$
Basic Operation: Page Read

- **BL control** – Charge all BLs

![Diagram of BL control and voltage levels](image_url)

- \( V_{REF} \) and \( WL_k \) signals
- BLs connected to various voltage levels: To \( V_{cc} \) or 0
- Erased (E) and Programmed states

**Threshold voltage (V_{TH})**

- Erased (E): 1
- Programmed: 0

<table>
<thead>
<tr>
<th># of cells</th>
<th>Erased (E)</th>
<th>Programmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_{REF}</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Basic Operation: Page Read

- Sensing the current through BLs

![Diagram showing sensing the current through BLs]

**V_{REF}**  **WL_k**

- **0** (No current)
- **1** (Current)

<table>
<thead>
<tr>
<th># of cells</th>
<th>Threshold voltage (V_{TH})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Erased (E)</td>
</tr>
<tr>
<td>1</td>
<td>Programmed</td>
</tr>
</tbody>
</table>

- V_{TH} < V_{REF}
- V_{TH} > V_{REF}

107
Basic Operation: Page Read - MLC

- Sensing the current through BLs

![Diagram showing sensing the current through BLs]
Basic Operation: Page Read - MLC

- Sensing the current through BLs

![Diagram showing the sensing of currents through BLs with WLk, BL0, BL1, BL2, BL3, and BL132,095.](image)

![Diagram showing the voltage reference points and cell states.](image)
Basic Operation: Page Read - MLC

- Sensing the current through BLs

![Diagram showing the sensing of current through BLs with values 111, 000, 101, 001, and 110]
Basic Operation: Page Read - MLC

- Sensing the current through BLs

```
WL_k

BL_0  BL_1  BL_2  BL_3  BL_132,095

111   000   101   001   110
```

```
V_{TH}

# of cells

MSB  LSB

V_{REF0}  V_{REF1}  V_{REF2}  V_{REF3}  V_{REF4}  V_{REF5}  V_{REF6}

CSB

111

E

P1  P2  P3  P4  P5  P6  P7

100  010  010  001  011  001  101
```
Basic Operation: Page Read - MLC

- Sensing the current through BLs
Basic Operation: Page Read - MLC

- Sensing the current through BLs

Schematic diagram showing the sensing process through bit lines (BLs) with corresponding voltage reference points (VREF0 to VREF6) and threshold voltages (V_TH). The MSB and LSB are highlighted with specific values for each cell.
Basic Operation: Page Read - MLC

- Sensing the current through BLs

![Diagram showing BLs and WLs with sensing and XOR operations]
Basic Operation: Page Read – Takeaways

- MLC NAND flash memory requires an on-chip XOR logic
- Bit-encoding affects the read latency!
  - Compare # of sensing for LSB

![Diagram showing V_TH and # of cells with various bit values and sensing points.]
Basic Operation: Page Read – Takeaways

- MLC NAND flash memory requires an on-chip XOR logic
- Bit-encoding affects the read latency!
  - Compare # of sensing for LSB
Basic Operation: Page Read – Takeaways

- MLC NAND flash memory requires an on-chip XOR logic
- Bit-encoding affects the read latency!
  - Compare # of sensing for LSB

![Diagram showing the comparison of sensing for LSB]
Basic Operation: Page Read – Takeaways

- MLC NAND flash memory requires an on-chip XOR logic
- Bit-encoding affects the read latency!
  - Compare # of sensing for LSB

![Diagram showing the comparison of sensing for LSB]
Read Mechanism

- NAND flash read mechanism consists of three steps:
  1) Precharge
  2) Evaluation
  3) Discharge
Enable precharge transistor $M_{\text{PRE}}$ to charge all target BLs and their sense-out capacitors ($C_{\text{SO}}$) to $V_{\text{PRE}}$. 

Precharge Mechanism: Precharge
Read Mechanism: Evaluation

1. **Precharge**
   - $V_{PRE}$ applied to $M_{PRE}$ and $C_{SO}$
   - $V_{PASS}$ maintains $V_{REF}$
   - Target WL $V_{TH} \leq V_{REF}$

2. **Evaluation**
   - $V_{PRE}$ applied to $M_{PRE}$ and $C_{SO}$
   - $V_{PASS}$ maintains $V_{REF}$
   - Target WL $V_{TH} \leq V_{REF}$ and $V_{TH} > V_{REF}$

3. **Discharge**
   - $V_{PRE}$ applied to $M_{PRE}$ and $C_{SO}$
   - $V_{PASS}$ maintains $V_{REF}$
   - Target WL $V_{TH} \leq V_{REF}$
   - $V_{PRE}$ and $V_{REF}$ discharged
Disconnect the BLs from \( V_{\text{PRE}} \) and enable the latching circuit.
If $V_{TH} \leq V_{REF}$, the charge in $C_{SO}$ quickly flows through the NAND string (Sensed as 1)
If $V_{TH} > V_{REF}$, the target cell blocks the BL discharge current (Sensed as 0)
Bitlines are discharged to return the NAND string to its initial state for future operations.
Latching Circuit

Before the evaluation step, the chip initializes the latching circuit

- Activating transistor $M_1$
- $V_{OUT} = 0$
- $V_{out} = 1$
Latching Circuit

- The evaluation step
  - Disables $M_{PRE}$ and $M_1$
  - Enables $M_2$

(a) $V_{TH} \leq V_{REF}$

(b) $V_{TH} > V_{REF}$
Inverse Read

- Performing an inverse read by simply changing the activation sequence of $M_1$ and $M_2$
  - The precharge step activates $M_2$
  - The evaluation step disables $M_2$ and activates $M_1$

(a) $V_{TH} \leq V_{REF}$

(b) $V_{TH} > V_{REF}$
SSD Performance

- **Latency (or response time)**
  - The time delay until the request is returned
  - Average read latency (4 KiB): 67 us
  - Average write latency (4 KiB): 47 us

- **Throughput**
  - The number of requests that can be serviced per unit time
    - **IOPS**: Input/output Operations Per Second
    - Random read throughput: up to 500K IOPS
    - Random write throughput: up to 480K IOPS

- **Bandwidth**
  - The amount of data that can be accessed per unit time
    - Sequential read bandwidth: up to 3,500 MB/s
    - Sequential write bandwidth: up to 3,000 MB/s

Source: https://www.anandtech.com/show/16504/the-samsung-ssd-980-500gb-1tb-review
Chip operation latency
- **tR**: Latency of reading (sensing) data from the cells into the on-chip page buffer
- **tPROG**: Latency of programming the cells with data in the page buffer
- **tBERS**: Latency of erasing the cells (block)
- Varies depending on the MLC technology, processing node, and microarchitecture
  - In 3D TLC NAND flash, $tR/tPROG/tBERS \approx 100\text{us}/700\text{us}/3\text{ms}$

I/O rate
- **Number of bits** transferred via a single I/O pin per unit time
- A typical flash chip transfers data in a byte granularity (i.e., via 8 I/O pins)
- e.g., 1-Gb I/O rate & 16-KiB page size $\rightarrow t\text{DMA} = 16\text{us}$
NAND Flash Chip Performance (Cont.)

- **tR, tPROG, and tBERS**
  - Latencies for chip-level read/program/erase operations
  - tR: 50~100 us
  - tPROG: 700us~1000 us
  - tBERS: 3ms~5ms

- **Flash-controller level latency**
  - 1-Gb I/O rate and 16-KiB page size
  - Read
    - \((t\text{CMD}) + tR + t\text{DMA} + t\text{ECC}_{\text{DEC}} + (t\text{RND})\)
    - e.g., 100 + 16 + 20 = 136 us
NAND Flash Chip Performance (Cont.)

- **tR, tPROG, and tBERS**
  - Latencies for chip-level read/program/erase operations
  - tR: 50~100 us
  - tPROG: 700us~1000 us
  - tBERS: 3ms~5ms

- Flash-controller level latency
  - 1-Gb I/O rate and 16-KiB page size
  - **Read**
    - \((tCMD) + tR + tDMA + tECC_{DEC} + (tRND)\)
    - e.g., 100 + 16 + 20 = 136 us
  - **Program**
    - \((tRND) + tECC_{ENC} + (tCMD) + tDMA + tPROG\)
    - e.g., 20 + 16 + 700 = 736 us
NAND Flash Chip Performance (Cont.)

- How about bandwidth?
  - **Read**
    - 16 KiB / 136 us \(\approx\) 120 MB/s
  - **Write**
    - 16 KiB / 736 us \(\approx\) 22 MB/s

**WAIT!**

- SSD read latency: 67 us
- SSD read bandwidth: 3.5 GB/s
- SSD write latency: 47 us
- SSD write bandwidth: 3 GB/s

Optimizations w/ advanced commands

---

**DRAM/SLC Write Buffer**

---

**Internal parallelism**

---

**NAND Flash Chip**

---

**ECC**

---

**RAND**

---

**Flash Controller**

---

...
Advanced Commands for Small Reads

- **Minimum I/O units in modern file systems**: 4 KiB
  - Latency & bandwidth waste due to I/O-unit mismatch
  - e.g., A page read unnecessarily reads/transfers 12-KiB data

- **Optimization 1**: Sub-page sensing
  - e.g., Micron SNAP READ operation
  - Microarchitecture-level optimization – directly reduces $t_R$

- **Optimization 2**: Random Data Out (RDO)
  - Data transfer with an arbitrary offset and size
  - Reduce $t_{DMA}$ and $t_{ECC_{DEC}}$

---

CACHE READ Command

- Performs consecutive reads in a pipelined manner

**Regular PAGE READ:**
Overlaps only tECC with tR

**CACHE READ:**
Overlaps tDMA & tECC with tR
Enabling the CACHE READ Command

- Needs additional on-chip page buffer

1. **PAGE READ (A)**

2. **Page sensing**

3. **CACHE READ (B)**

4. **Page sensing**

5. **DATA OUT (A)**

NAND Flash Plane

- Page A
- Page B
- ...

Page Buffer 1
Page Buffer 2
CACHE READ Command: Benefit

- Removes tDMA from the critical path
  - Increases throughput/bandwidth
  - Reduces effective latency
    - By reducing the time delay for a request being blocked by the previous request
Multi-Plane Operations

- Concurrent operations on different planes
  - Recall: Planes share WLs and row/column decoders
  - Opportunity: Planes can **concurrently** operate
  - Constraints: Only for the same operations on the same page offset
Multi-Plane Operations: Benefit

- Increase the throughput/bandwidth almost linearly with 
  # of planes that concurrently operate
  - Bandwidth with regular page programs:  
    16 KiB / 736 us ≈ 22 MB/s
  - Bandwidth with multi-plane page programs (2 planes):  
    32 KiB / 736 + 16 (tDMA) + 20 (tECC) us ≈ 41.5 MB/s

- Per-operation latency increases
  - Regular page program: tECC\textsubscript{ENC} + tDMA + tPROG
  - Multi-plane page program: \( N_{\text{Plane}} \times (t\text{ECC}_{\text{ENC}} + t\text{DMA}) + t\text{PROG} \)

- The benefits highly depend on the access pattern and FTL’s data placement
Program & Erase Suspensions

- **Read performance is often more important**
  - Writes can be done in an asynchronous manner using buffers
    - e.g., return a write request immediately after receiving the data (and storing it to the write buffer)
  - A read request can be returned only when the requested data is ready (after reading the data from the chip)

- **Significant latency asymmetry**
  - tR: 100 us, tPROG: 700 us, tBERS: 5 ms (TLC NAND flash)
    - If the chip is designed to program all the pages in the same WL at once, the actual program latency is 2,100 us
  - The worst-case chip-level read latency can be 50x longer than the best-case latency
Program & Erase Suspensions (Cont.)

- Suspends an on-going program (erase) operation once a read arrives

- **Pros:** Significantly decreases the read latency
- **Cons**
  - Additional page buffer (for data to program)
  - Complicated I/O scheduling (Until when can we suspend on-going program requests?)
  - Negative impact on the endurance
Summary

- **Subpage Sensing & Random Data Out (RDO)**
  - For *I/O-unit mismatch* b/w OS and NAND flash memory

- **Cache Read Command**
  - For improving a chip’s *read throughput*
  - By overlapping data transfer and page sensing

- **Multi-Plane Operations**
  - For improving a chip’s *throughput*
  - By enabling *concurrently operation of multiple planes*

- **Program & Erase Suspensions**
  - For improving the *read latency* (*operation latency asymmetry*)
  - By prioritizing latency-sensitive reads over writes/erases
Agenda

- Overview on SSD Organization
  - Storage Controller & Request Handling
  - NAND Flash Hierarchy
  - NAND Flash Read/Write Operations

- Address Mapping and Garbage Collection

- I/O Scheduling
**Flash Translation Layer: Overview**

- SSD firmware (often referred to as SSD controller)
  - Provides *backward compatibility* with traditional HDDs
  - By hiding *unique characteristics* of NAND flash memory

- Responsible for many important **SSD-management tasks**
  - Address translation + garbage collection
    - Performs *out-of-place writes* due to erase-before-write property
  - Wear leveling
    - To prolong SSD lifetime by *evenly distributing* P/E cycles
  - Data refresh
    - Resets transient errors by *copying data* to a new page(s)
  - I/O scheduling
    - To take full advantage of **SSD internal parallelism**
Flash Translation Layer: Overview

- SSD firmware (often referred to as SSD controller)
  - Provides *backward compatibility* with traditional HDDs
  - By *hiding* unique characteristics of NAND flash memory

- Responsible for many important *SSD-management tasks*
  - Address translation + garbage collection
    - Performs *out-of-place writes* due to erase-before-write property
  - Wear leveling
    - To prolong SSD lifetime by *evenly distributing* P/E cycles
  - Data refresh
    - Resets transient errors by *copying data* to a new page(s)
  - I/O scheduling
    - To take full advantage of *SSD internal parallelism*
Simple SSD Architecture

Logical Block Address

Host

SSD

Flash Translation Layer

Storage view at the operating-system level: A flat block device

NAND Flash Chip (Single Plane)
Simple SSD Architecture

Overprovisioning:
- Physical capacity > Logical capacity
- For performance & lifetime
Write Request Handling: Page Write

Host

SSD

Flash Translation Layer

NAND Flash Chip (Single Plane)
Write Request Handling: Page Write

Flash Translation Layer

Req (LBA: 0, Size: 1, DIR: W, A)
Write Request Handling: Page Write

Flash Translation Layer

Req (LBA: 0, Size: 1, DIR: W, A)
Write Request Handling: Page Write

Flash Translation Layer

Req (LBA: 0, Size: 1, DIR: W, A)

Note:
- We are assuming that logical block size = physical page size
- LB size = 4 KiB, PP size = 16 KiB

Host

SSD

NAND Flash Chip (Single Plane)
Write Request Handling: Sequential Write

Flash Translation Layer

NAND Flash Chip (Single Plane)
Write Request Handling: Sequential Write

**Flash Translation Layer**

**Req** *(LBA: 4, Size: 12, DIR: W, B ... M)*

**Sequential (large) write**

**NAND Flash Chip (Single Plane)**
Write Request Handling: Sequential Write

**Host**

**SSD**

**Flash Translation Layer**

**Req** (LBA: 4, Size: 12, DIR: W, B ... M)

PROG (PPA: 1, B)
PROG (PPA: 2, C)
PROG (PPA: 12, M)

12 page-program commands

**NAND Flash Chip (Single Plane)**
Write Request Handling: Sequential Write

Host

SSD

NAND Flash Chip (Single Plane)

Flash Translation Layer

Req \(\text{LBA: 4, Size: 12, DIR: W,} \)\n
\[\text{PROG(PPA: 1, B)}\]
\[\text{PROG(PPA: 2, C)}\]
\[\vdots\]
\[\text{PROG(PPA: 12, M)}\]

Sequential (large) write

- **Active block** (or write-point) approach
  - Keep only one block being written
  - Due to the open-block problem
- **Program-sequence** constraint
  - Fixed program order within a block
  - Due to cell-to-cell interference

- Sequential (large) write

<table>
<thead>
<tr>
<th>Block0</th>
<th>Block1</th>
<th>Block2</th>
<th>Block3</th>
<th>Block4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>4</td>
<td>E</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>5</td>
<td>F</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>6</td>
<td>G</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>7</td>
<td>H</td>
<td>11</td>
</tr>
</tbody>
</table>

NAND Flash Chip (Single Plane)
Write Request Handling: Address Mapping

Flash Translation Layer

Problem: LBA (or LPA) does not match PPA!

NAND Flash Chip (Single Plane)
Write Request Handling: Address Mapping

Flash Translation Layer

Problem: LBA (or LPA) does not match PPA!

Req (LBA: 4, Size: 1, DIR: R)

 Needs to maintain Address-mapping information

NAND Flash Chip (Single Plane)
Write Request Handling: Address Mapping

Host

SSD

Flash Translation Layer

Req (LBA: 4, Size: 1, DIR: R)

READ (PPA: ?)

Mapping Table

NAND Flash Chip (Single Plane)
Write Request Handling: Address Mapping

<table>
<thead>
<tr>
<th>Block0</th>
<th>Block1</th>
<th>Block2</th>
<th>Block3</th>
<th>Block4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>E</td>
<td>I</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>J</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>G</td>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>H</td>
<td>L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NAND Flash Chip (Single Plane)

Flash Translation Layer

Req (LBA: 4, Size: 1, DIR: R)

READ (PPA: 1)

Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Host

SSD
Write Request Handling: Update

Flash Translation Layer

Mapping Table

NAND Flash Chip (Single Plane)
Write Request Handling: Update

Host SSD

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>A’</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Flash Translation Layer

Req (LBA: 0, Size: 1, DIR: W, A’)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Mapping Table

<table>
<thead>
<tr>
<th>Block0</th>
<th>Block1</th>
<th>Block2</th>
<th>Block3</th>
<th>Block4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>4</td>
<td>E</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>5</td>
<td>F</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>6</td>
<td>G</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>7</td>
<td>H</td>
<td>11</td>
</tr>
</tbody>
</table>

NAND Flash Chip (Single Plane)
Write Request Handling: Update

**Flash Translation Layer**

**Req** (LBA: 0, Size: 1, DIR: W, A')

**PROG** (PPA: 13, A')

**Mapping Table**

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**NAND Flash Chip (Single Plane)**

[Diagram of Flash Translation Layer and NAND Flash Chip]
Write Request Handling: Update

Host

SSD

Flash Translation Layer

Req (LBA: 0, Size: 1, DIR: W, A')

PROG (PPA: 13, A')

Mapping Table

NAND Flash Chip (Single Plane)
Write Request Handling: Update

Flash Translation Layer

Req (LBA: 0, Size: 1, DIR: W, A')

PROG (PPA: 14, A')

Mapping Table

NAND Flash Chip (Single Plane)
Write Request Handling: Update

**Flash Translation Layer**

**Req (LBA: 0, Size: 1, DIR: W, A')**

**PROG (PPA: 15, A')**

**Mapping Table**

**NAND Flash Chip (Single Plane)**
Write Request Handling: Update

![Diagram showing Flash Translation Layer and mapping table]

**Req (LBA: 0, Size: 1, DIR: W, A')**

**PROG (PPA: 16, A')**

**Mapping Table**

**NAND Flash Chip (Single Plane)**

Running out of free pages
Garbage Collection

- Reclaims **free pages** by erasing **invalid** pages
  - Erase unit: **block**
  - If a victim block (to erase) has **valid pages**, all the valid pages **need to be copied** to other free pages
    - **Performance overhead**: \((t_{\text{READ}} + t_{\text{PROG}}) \times \# \text{ of valid pages}\)
    - **Lifetime overhead**: additional writes $\rightarrow$ P/E-cycle increase

- **Greedy** victim-selection policy:
  - Erases the block with the **largest number** of invalid pages
  - Needs to maintain **# of invalid (or valid) pages** for each block
Write Request Handling: Garbage Collection

Host

SSD

Flash Translation Layer

F: free, V: valid, I: invalid

Mapping Table

Status Table

NAND Flash Chip (Single Plane)
Write Request Handling: Garbage Collection

Flash Translation Layer

Mapping Table

Status Table

NAND Flash Chip (Single Plane)
Write Request Handling: Garbage Collection

Flash Translation Layer

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Mapping Table

<table>
<thead>
<tr>
<th>PBA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IVVVV</td>
</tr>
<tr>
<td>1</td>
<td>VVVV</td>
</tr>
<tr>
<td>2</td>
<td>VVVV</td>
</tr>
<tr>
<td>3</td>
<td>IIII</td>
</tr>
<tr>
<td>4</td>
<td>VVFF</td>
</tr>
</tbody>
</table>

Status Table

READ (PPA: 12)

PROG (PPA: 17, M)

NAND Flash Chip (Single Plane)

Host

SSD

Update Status

F: free, V: valid, I: invalid
Write Request Handling: Garbage Collection

Flash Translation Layer

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
</tr>
</tbody>
</table>

Status Table

<table>
<thead>
<tr>
<th>PBA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IVVVV</td>
</tr>
<tr>
<td>1</td>
<td>VVVV</td>
</tr>
<tr>
<td>2</td>
<td>VVVV</td>
</tr>
<tr>
<td>3</td>
<td>IIII</td>
</tr>
<tr>
<td>4</td>
<td>VVFF</td>
</tr>
</tbody>
</table>

Update Status

Mapping Table

NAND Flash Chip (Single Plane)

Host SSD

READ (PPA: 12)
PROG (PPA: 17, M)
Update Mapping

F: free, V: valid, I: invalid
### Write Request Handling: Garbage Collection

#### Flash Translation Layer

**READ** (PPA: 12)

**PROG** (PPA: 17, M)

**Update Mapping**

**Update Status**

#### NAND Flash Chip (Single Plane)

<table>
<thead>
<tr>
<th>Host</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
</tr>
</tbody>
</table>

#### Status Table

<table>
<thead>
<tr>
<th>PBA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IVVVV</td>
</tr>
<tr>
<td>1</td>
<td>VVVVV</td>
</tr>
<tr>
<td>2</td>
<td>VVVVV</td>
</tr>
<tr>
<td>3</td>
<td>IIIII</td>
</tr>
<tr>
<td>4</td>
<td>VVFFF</td>
</tr>
</tbody>
</table>

#### NAND Flash Chip (Single Plane)

<table>
<thead>
<tr>
<th>Block0</th>
<th>Block1</th>
<th>Block2</th>
<th>Block3</th>
<th>Block4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
<td>19</td>
</tr>
</tbody>
</table>
Write Request Handling: Garbage Collection

- **Q:** How FTL knows PPA 12 (data M) is mapped to LPA 15?
  - Unless it maintains P2L mappings?
- **A:** P2L mapping is stored in each physical page’s OOB (Out-of-Band) area

**READ (PPA: 12)**

**PROG (PPA: 17, M)**

**Update Mapping**

**Update Status**

- **F:** free, **V:** valid, **I:** invalid

**Mapping Table**

<table>
<thead>
<tr>
<th>PBA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IVVVV</td>
</tr>
<tr>
<td>1</td>
<td>VVVV</td>
</tr>
<tr>
<td>2</td>
<td>VVVV</td>
</tr>
<tr>
<td>3</td>
<td>IIII</td>
</tr>
<tr>
<td>4</td>
<td>VVFF</td>
</tr>
</tbody>
</table>

**Status Table**

**NAND Flash Chip (Single Plane)**
Write Request Handling: Garbage Collection

Flash Translation Layer

**Flash Translation Layer**

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
</tr>
</tbody>
</table>

**Status Table**

<table>
<thead>
<tr>
<th>PBA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IVVVV</td>
</tr>
<tr>
<td>1</td>
<td>VVVV</td>
</tr>
<tr>
<td>2</td>
<td>VVVV</td>
</tr>
<tr>
<td>3</td>
<td>I III</td>
</tr>
<tr>
<td>4</td>
<td>V V FF</td>
</tr>
</tbody>
</table>

**Mapping Table**

<table>
<thead>
<tr>
<th></th>
<th>PBA</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Host SSD NAND Flash Chip (Single Plane)**

**Host SSD NAND Flash Chip (Single Plane)**

**Block0**

- **A**
- **B**
- **C**
- **D**

**Block1**

- **E**
- **F**
- **G**
- **H**

**Block2**

- **I**
- **J**
- **K**
- **L**

**Block3**

- **M**
- **N**
- **O**
- **P**

**Block4**

- **A’**
- **B’**
- **C’**
- **D’**
Write Request Handling: Garbage Collection

**Flash Translation Layer**

- **LPA**
  - 0: 16
  - 4: 1
- **PPA**
  - 12
  - 17
  - 3

**Mapping Table**

- **Block0**
  - A
  - B
  - C
  - D
- **Block1**
  - E
  - F
  - G
  - H
- **Block2**
  - I
  - J
  - K
  - L
- **Block3**
  - 12
  - 13
  - 14
  - 15
- **Block4**
  - A'
  - B

**Status Table**

- **PBA**
  - 0
  - 1
  - 2
  - 3
  - 4

**Status**

- **F**: free
- **V**: valid
- **I**: invalid

**Update Status**

- **READ (PPA: 12)**
- **PROG (PPA: 17, M)**
- **BERS (PBA: 3)**
Write Request Handling: Garbage Collection

**Note:**
- Block erasure (and status update) is done just before programming a new page to the block (i.e., lazy erase)
  - Due to the open-block problem

(PPA: 12)
- PROG (PPA: 17, M)
- BERS (PBA: 3)

Update Status

<table>
<thead>
<tr>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>4</td>
<td>E</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>5</td>
<td>F</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>6</td>
<td>G</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>7</td>
<td>H</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>A'</td>
<td>17</td>
<td>M</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance Issues

- Garbage collection significantly affects SSD performance
  - High latency: Large block size of modern NAND flash memory
    - Assume 1) a block contains 576 pages,
      2) only 5% of the pages in the victim block are valid
      3) $t_R = 100$ us, $t_{PROG} = 700$ us, $t_{BERS} = 5$ ms
    - # of pages to copy = $576 \times 0.05 = 28.8 \rightarrow 28$ pages
    - GC latency > $28 \times (t_R + t_{PROG}) + t_{BERS} = 27,400$ us
    - Order(s) of magnitude larger latency than $t_R$ and $t_{PROG}$
    - Copy operations are the major contributor (rather than $t_{BERS}$)
  - If FTL performs GC in an atomic manner, it delays user requests for a significantly long time
    - Long tail latency (performance fluctuation)
    - Noisy neighbor: a read-dominant workload’s performance would be significantly affected when running with a write-intensive workload (+ performance fairness problem)
Performance Issues: Mitigation

- **TRIM (UNMAP or discard) command**
  - Informs FTL of deletion/deallocation of a logical block
  - Allows FTL to skip copy of obsolete (i.e., invalid) data

- **Background GC:** Exploits SSD idle time
  - Challenge: how to accurately predict SSD idle time
  - Premature GC: copied pages could have been invalidated by the host system

- **Progressive GC:** Divide GC process into subtasks
  - e.g., copying 28 pages → (copying 1 page + servicing user request) × 28
  - Effective at decreasing tail latency
Fine-Grained Mapping
I/O Mismatch b/w OS and NAND Flash

- The page size (i.e., minimum I/O unit) of NAND flash memory has continuously increased
  - From 256 bytes to 16 KiB
  - Low area overhead and high bandwidth (size / latency)

- The logical block (or sector) size of file systems has also increased
  - From 512 bytes to 4 KiB
  - Increasing the block size is not straightforward
    - I/O handling is closely related to OS memory management
    - Memory page size = 4 KiB
    - Unnecessary fetch or eviction at the page cache
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: 0x04, Size: 1, DIR: w, Data: A)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>-</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

16-KiB Page Number 4-KiB Offset
Small Write Requests

- Inefficiencies due to the erase-before-write property

**Req** (LBA: \(0x04\), Size: 1, DIR: \(w\), Data: \(A\))

\[
\begin{array}{c}
0b \ 0000 \ 0000 \ 0000 \ 0100 \\
\text{16-KiB Page Number} \ \ \ \ \ \ \ \ \text{4-KiB Offset}
\end{array}
\]

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>-</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Block 0

- PPA 0x00
  - A

Block 1

- PPA 0x01
- PPA 0x02
- PPA 0x03
- PPA 0x04
- PPA 0x05
- PPA 0x06
- ...
Small Write Requests

- Inefficiencies due to the erase-before-write property

**Req** (LBA: 0x04, Size: 1, DIR: w, Data: A)

```
0b 0000 0000 0000 0100
```

16-KiB Page Number  4-KiB Offset

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Block 0**

```
0x00
0x01
0x02
0x03
```

**Block 1**

```
0x04
0x05
0x06
...```

A
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

0b 0000 0000 0000 0001
16-KiB Page Number 4-KiB Offset

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PPA</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td></td>
</tr>
<tr>
<td>0x04</td>
<td></td>
</tr>
<tr>
<td>0x05</td>
<td></td>
</tr>
<tr>
<td>0x06</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
0b 0000 0000 0000 0001
16-KiB Page Number  4-KiB Offset
```

Block 0

<table>
<thead>
<tr>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
</tr>
<tr>
<td>0x01</td>
</tr>
<tr>
<td>0x02</td>
</tr>
<tr>
<td>0x03</td>
</tr>
<tr>
<td>0x04</td>
</tr>
<tr>
<td>0x05</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Block 1

<table>
<thead>
<tr>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x06</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

0b 0000 0000 0000 0001
16-KiB Page Number 4-KiB Offset

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Block 0

PPA
0x00
0x01
0x02
0x03
0x04
0x05
0x06

A

B

C

...
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

\[ \text{0b 0000 0000 0000 0001} \]

16-KiB Page Number 4-KiB Offset

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

1. Why at the middle of the page?
   - To keep the 4-KiB offset: mapping table stores only the index of the 16-KiB page!

2. Why not using the unused space in physical page 0x00?
   - That space is already mapped to logical pages 0x05~0x07 (not written yet).
Small Write Requests

- Inefficiencies due to the erase-before-write property

**Req (LBA: 0x07, Size: 1, DIR: w, Data: D)**

\[ 0b \quad 0000 \quad 0000 \quad 0000 \quad 0111 \]

16-KiB Page Number 4-KiB Offset

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Q: Can we use the unused space?**

**A: Not likely, because**

- Data randomization – Cells in the unused space have been already programmed.
- Program-order constraint – Re-programming physical page 0x00 can affect the reliability of the data stored in physical page 0x01.
Small Write Requests

- Inefficiencies due to the erase-before-write property

**Req** (LBA: 0x07, Size: 1, DIR: w, Data: D)

```
0b 0000 0000 0000 0111
16-KiB Page Number  4-KiB Offset
```

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Block 0**

```
0x00
0x01
0x02
0x03
```

**Block 1**

```
0x04
0x05
0x06
...
```

Unused yet discarded
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: 0x07, Size: 1, DIR: w, Data: D)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

0b 0000 0000 0000 0111
16-KiB Page Number  4-KiB Offset

Read

Block 0

PPA
0x00
0x01
0x02
0x03
0x04
0x05
0x06
...

Block 1

PPA
0x00
0x01
0x02
0x03
0x04
0x05
0x06
...

A

B

C
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: \texttt{0x07}, Size: 1, DIR: w, Data: D)

\[\begin{array}{c}
\text{Block 0} \\
A \quad \text{PPA} \quad 0x00 \\
B \quad 0x01 \\
C \quad 0x02 \\
\ldots \\
\text{Block 1} \\
0x04 \\
0x05 \\
0x06 \\
\ldots
\end{array}\]

16-KiB Page Number 4-KiB Offset

\[\begin{array}{c}
\text{LPA} \\
0x00 \\
0x01 \\
0x02 \\
0x03 \\
0x04 \\
0x05 \\
\ldots
\end{array}, \begin{array}{c}
\text{PPA} \\
0x00 \\
0x01 \\
0x02 \\
0x03 \\
0x04 \\
0x05 \\
\ldots
\end{array}\]
Small Write Requests

- Inefficiencies due to the erase-before-write property

**Req (LBA: 0x07, Size: 1, DIR: w, Data: D)**

```
0b 0000 0000 0000 0111
16-KiB Page Number  4-KiB Offset
```

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
</tr>
<tr>
<td>0x01</td>
<td>0x00</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>0x03</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>0x05</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Modify**

**Read**

**Write**

Block 0

- PPA 0x00
- PPA 0x01
- PPA 0x02
- PPA 0x03

Block 1

- PPA 0x04
- PPA 0x05
- PPA 0x06
Small Write Requests

- Inefficiencies due to the erase-before-write property

**Req** (LBA: 0x07, Size: 1, DIR: W, Data: D)

```
0b 0000 0000 0000 0111
16-KiB Page Number 4-KiB Offset
```

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
</tr>
<tr>
<td>0x01</td>
<td>0x02</td>
</tr>
<tr>
<td>0x02</td>
<td>-</td>
</tr>
</tbody>
</table>

**Small writes cause read-modify-writes:**
Waste of P/E cycles + additional read operations
→ Performance and lifetime degradation
**Fine-Grained Mapping + Page Buffer**

- Write a page only when there are sufficient data blocks

**Req** (LBA: 0x04, Size: 1, DIR: w, Data: A)

**Req** (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

**Req** (LBA: 0x07, Size: 1, DIR: w, Data: D)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x07</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Page Buffer**

![Page Buffer Diagram]

**Block 0**

<table>
<thead>
<tr>
<th>0x00</th>
<th>0x01</th>
<th>0x02</th>
<th>0x03</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04</td>
<td>0x05</td>
<td>0x06</td>
<td>0x07</td>
</tr>
<tr>
<td>0x08</td>
<td>0x09</td>
<td>0x0A</td>
<td>0x0B</td>
</tr>
<tr>
<td>0x0C</td>
<td>0x0D</td>
<td>0x0E</td>
<td>0x0F</td>
</tr>
<tr>
<td>0x10</td>
<td>0x11</td>
<td>0x12</td>
<td>0x13</td>
</tr>
<tr>
<td>0x14</td>
<td>0x15</td>
<td>0x16</td>
<td>0x17</td>
</tr>
<tr>
<td>0x18</td>
<td>0x19</td>
<td>0x1A</td>
<td>0x0B</td>
</tr>
</tbody>
</table>

**Block 1**

<table>
<thead>
<tr>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x18</td>
</tr>
<tr>
<td>0x19</td>
</tr>
<tr>
<td>0x1A</td>
</tr>
<tr>
<td>0x0B</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
## Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

**Req** (LBA: 0x04, Size: 1, DIR: w, Data: A)

**Req** (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

**Req** (LBA: 0x07, Size: 1, DIR: w, Data: D)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x07</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

![Page Buffer Diagram]

![Block Diagram]
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

**Req** (LBA: 0x04, Size: 1, DIR: w, Data: A)
**Req** (LBA: 0x01, Size: 2, DIR: w, Data: B, C)
**Req** (LBA: 0x07, Size: 1, DIR: w, Data: D)
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

**Req (LBA: 0x04, Size: 1, DIR: w, Data: A)**

**Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)**

**Req (LBA: 0x07, Size: 1, DIR: w, Data: D)**

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0x04</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0x07</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page Buffer

A

Block 0

0x00 0x01 0x02 0x03
0x04 0x05 0x06 0x07
0x08 0x09 0x0A 0x0B
0x0C 0x0D 0x0E 0x0F
0x10 0x11 0x12 0x13
0x14 0x15 0x16 0x17
0x18 0x19 0x1A 0x1B

Block 1

PPA

...
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: w, Data: A)
Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)
Req (LBA: 0x07, Size: 1, DIR: w, Data: D)
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: w, Data: A)
Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)
Req (LBA: 0x07, Size: 1, DIR: w, Data: D)
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

**Request 1**
- LBA: 0x04
- Size: 1
- DIR: w
- Data: A

**Request 2**
- LBA: 0x01
- Size: 2
- DIR: w
- Data: B, C

**Request 3**
- LBA: 0x07
- Size: 1
- DIR: w
- Data: D

### Page Buffer

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### LPA to PPA Mapping

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>0x01</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>0x00</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x07</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Block 0

```
0x00 | 0x01 | 0x02 | 0x03
0x04 | 0x05 | 0x06 | 0x07
0x08 | 0x09 | 0x0A | 0x0B
0x0C | 0x0D | 0x0E | 0x0F
0x10 | 0x11 | 0x12 | 0x13
0x14 | 0x15 | 0x16 | 0x17
0x18 | 0x19 | 0x1A | 0x1B
```

### Block 1

```
0x10 | 0x11 | 0x12 | 0x13
0x14 | 0x15 | 0x16 | 0x17
0x18 | 0x19 | 0x1A | 0x1B
```

**PPA**
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

**Req** (LBA: 0x04, Size: 1, DIR: w, Data: A)

**Req** (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

**Req** (LBA: 0x07, Size: 1, DIR: w, Data: D)
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: w, Data: A)
Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)
Req (LBA: 0x07, Size: 1, DIR: w, Data: D)

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>0x01</td>
<td>0x01</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x04</td>
<td>0x00</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>0x07</td>
<td>0x03</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Page Buffer

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x01</td>
<td>0x02</td>
<td>0x03</td>
<td></td>
</tr>
<tr>
<td>0x04</td>
<td>0x05</td>
<td>0x06</td>
<td>0x07</td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>0x09</td>
<td>0x0A</td>
<td>0x0B</td>
<td></td>
</tr>
<tr>
<td>0x0C</td>
<td>0x0D</td>
<td>0x0E</td>
<td>0x0F</td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>0x11</td>
<td>0x12</td>
<td>0x13</td>
<td></td>
</tr>
<tr>
<td>0x14</td>
<td>0x15</td>
<td>0x16</td>
<td>0x17</td>
<td></td>
</tr>
<tr>
<td>0x18</td>
<td>0x19</td>
<td>0x1A</td>
<td>0x0B</td>
<td></td>
</tr>
</tbody>
</table>

Block 0

Block 1

PPA
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: w, Data: A)

Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: w, Data: D)

Fine-grained mapping significantly reduces the number of NAND flash operations:

3 writes (+1 read) $\rightarrow$ 1 write
Drawbacks of Fine-Grained Mapping

- Larger mapping table
  - 16-KiB mapping → 4 bytes per 16-KiB page = 0.025%
  - 4-KiB mapping → 4 bytes per 4-KiB page = 0.1%
  - For a 2-TB SSD, 2-GB DRAM is required.
    - Increases the SSD’s price and power/energy consumption

- Data durability of written data
  - Page buffers are implemented by using volatile memory (e.g., SRAM or DRAM).
  - Once data is written to an SSD, the SSD needs to guarantee the data’s integrity even under sudden power off.
  - Solution: power capacitors

*Despites non-negligible drawbacks, fine-grained mapping is widely used in modern SSDs due to its high benefits*
Multi-plane Operation-Aware Block Management
Concurrent operations on different planes
- Recall: Planes share WLs and row/column decoders
- Opportunity: Planes can **concurrently** operate
- Constraints: Only for the same operations on the same page offset

Recap: Multi-Plane Operations

- Same voltage is applied to all cells

![Diagram showing multi-plane operations with WLs and decoders]
Multi-Plane-Aware Data Placement

- To perform as many multi-plane operations as possible
  - Flush $N_{\text{plane}}$ pages at once after buffering them

![Diagram of page buffer and planes]

- Wait for more writes
- Full!
To perform as many multi-plane operations as possible

- Flush $N_{\text{plane}}$ pages at once after buffering them
Multi-Plane-Aware Data Placement

- To perform as many multi-plane operations as possible
  - Flush $N_{\text{plane}}$ pages at once after buffering them
  - Need to keep the write points of all planes to be the same
- Superblock-based block management

```
Page buffer

Plane_0  Plane_1  Plane_2  Plane_3

Page Buffers
Peripheral Circuits
```
Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).
Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).
Multi-Plane-Aware Block Management

Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).

[Diagram showing block management with various blocks and their invalid page counts.]

1. **Victim selection**
   - Block_0: $N_{INVALID} = 3$
   - Block_1: $N_{INVALID} = 2$
   - Block_2: $N_{INVALID} = 5$
   - Block_{N-1}: $N_{INVALID} = 1$

2. **Valid page copy**

Page Status:
- Free
- Valid
- Invalid
Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).

Multi-Plane-Aware Block Management

- Victim selection
- Valid page copy
- Page invalidation

Block status:
- Free
- Valid
- Invalid

Plane diagram:
- Block0: N_INVALID = 3
- Block1: N_INVALID = 2
- Block2: N_INVALID = 5
- BlockN-2: N_INVALID = 1
- BlockN-1: Active Block (or Write Point)
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.

Cannot perform multi-plane writes for future writes (due to different offsets)

Cannot perform multi-plane writes before using (or discarding) 2nd and 3rd pages of Block$_{N-1}$ in Plane$_1$

4-page copy: can be done with 
$2 \times$ multi-plane reads & 
$2 \times$ multi-plane writes

$4 \times$ single-plane reads & 
$1 \times$ multi-plane write & 
$2 \times$ single-plane writes
Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes
**Multi-Plane-Aware Block Management**

- **Superblock-based management**: groups each block with the same index (i.e., vertical position) in different planes.

![Diagram](image)

1. **Valid page reads** (5 SRs + 1 MR)

- Block_0
- Block_1
- Block_2
- Block_{N-2}
- Block_{N-1}

- Plane_0
- Plane_1

- Die
Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes.

1. Valid page reads (5 SRs + 1 MR)
2. Valid page writes (3 MRs)
Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes

Pros:
- Keep performing multi-plane writes

Cons:
- More read/write operations
  - $5 \text{ SRs} + 1 \text{ MR} + 3 \text{ MWs}$
- vs. $4 \text{ SRs} + 1 \text{ MW} + 2 \text{ SWs}$
Multi-Plane-Aware Block Management

- Offset management: Die level or SSD level?

Multi-plane operations can significantly improve SSD performance, but requires proper management in FTL
Agenda

- Overview on SSD Organization
  - Storage Controller & Request Handling
  - NAND Flash Hierarchy
  - NAND Flash Read/Write Operations

- Address Mapping and Garbage Collection

- I/O Scheduling
Flash Translation Layer: Overview

- SSD firmware (often referred to as SSD controller)
  - Provides **backward compatibility** with traditional HDDs
  - By hiding **unique characteristics** of NAND flash memory

- Responsible for many important **SSD-management tasks**
  - Address translation + garbage collection
    - Performs **out-of-place writes** due to erase-before-write property
  - Wear leveling
    - To prolong SSD lifetime by **evenly distributing** P/E cycles
  - Data refresh
    - Resets transient errors by **copying data** to a new page(s)
  - I/O scheduling
    - To take full advantage of **SSD internal parallelism**
FLIN:
Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives

Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie S. Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gómez-Luna, Onur Mutlu

ISCA 2018
Executive Summary

- Modern solid-state drives (SSDs) use new storage protocols (e.g., NVMe) that **eliminate the OS software stack**
  - I/O requests are now scheduled inside the SSD
  - Enables **high throughput**: millions of IOPS

- **OS software stack elimination removes existing fairness mechanisms**
  - We experimentally characterize fairness on four real state-of-the-art SSDs
  - Highly unfair slowdowns: large difference across concurrently-running applications

- We find and analyze **four sources of inter-application interference** that lead to slowdowns in state-of-the-art SSDs

- **FLIN**: a new I/O request scheduler for modern SSDs designed to provide both **fairness and high performance**
  - Mitigates all four sources of inter-application interference
  - Implemented fully in the SSD controller firmware, uses < 0.06% of DRAM space
  - FLIN improves **fairness by 70%** and **performance by 47%** compared to a state-of-the-art I/O scheduler
Background: Modern SSD Design

Unfairness Across Multiple Applications in Modern SSDs

FLIN: Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion
- **Back End**: data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)
**Internal Components of a Modern SSD**

- **Back End**: data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)
- **Front End**: management and control units
Internal Components of a Modern SSD

- **Back End**: data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)

- **Front End**: management and control units
  - **Host–Interface Logic (HIL)**: protocol used to communicate with host
Internal Components of a Modern SSD

- **Back End**: data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)

- **Front End**: management and control units
  - Host–Interface Logic (HIL): protocol used to communicate with host
  - Flash Translation Layer (FTL): manages resources, processes I/O requests
Internal Components of a Modern SSD

- **Back End**: data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)

- **Front End**: management and control units
  - Host–Interface Logic (HIL): protocol used to communicate with host
  - Flash Translation Layer (FTL): manages resources, processes I/O requests
  - Flash Channel Controllers (FCCs): sends commands to, transfers data with memory chips in back end
SSDs initially adopted conventional host–interface protocols (e.g., SATA)

- Designed for magnetic hard disk drives
- Maximum of only **thousands of IOPS** per device
Modern SSDs use high-performance host–interface protocols (e.g., NVMe)

- Bypass OS intervention: SSD must perform scheduling
- Take advantage of SSD throughput: enables millions of IOPS per device

Fairness mechanisms in OS software stack are also eliminated.

Do modern SSDs need to handle fairness control?
Outline

Background: Modern SSD Design

Unfairness Across Multiple Applications in Modern SSDs

FLIN:
Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion
Measuring Unfairness in Real, Modern SSDs

- We measure fairness using four **real state-of-the-art SSDs**
  - NVMe protocol
  - Designed for datacenters

- **Flow:** a series of I/O requests generated by an application

- \( \text{Slowdown} = \frac{\text{shared flow response time}}{\text{alone flow response time}} \) (lower is better)

- \( \text{Unfairness} = \frac{\max \text{ slowdown}}{\min \text{ slowdown}} \) (lower is better)

- \( \text{Fairness} = \frac{1}{\text{unfairness}} \) (higher is better)
Representative Example: *tpcc* and *tpce*

**Average slowdown of *tpce*:**

2x to 106x across our four real SSDs

SSDs do not provide fairness among concurrently-running flows
What Causes This Unfairness?

- Interference among concurrently-running flows
- We perform a detailed study of interference
  - MQSim: detailed, open-source modern SSD simulator [FAST 2018]  
    https://github.com/CMU-SAFARI/MQSim
  - Run flows that are designed to demonstrate each source of interference
  - Detailed experimental characterization results in the paper

- We uncover four sources of interference among flows
Source 1: Different I/O Intensities

- The **I/O intensity** of a flow affects the average **queue wait time** of flash transactions.

The queue wait time highly increases with **I/O intensity**.
An experiment to analyze the effect of concurrently executing two flows with different I/O intensities on fairness

- **Base flow**: low intensity (16 MB/s) and low average chip-level queue length
- **Interfering flow**: varying I/O intensities from low to very high

The average response time of a low-intensity flow substantially increases due to interference from a high-intensity flow
Source 2: Different Access Patterns

- Some flows take advantage of **chip-level parallelism** in back end

  - Even distribution of transactions in chip-level queues

- Leads to a **low queue wait time**
Other flows have access patterns that do not exploit parallelism.
An experiment to analyze the interference between concurrent flows with different access patterns

- Base flow: streaming access pattern (parallelism friendly)
- Interfering flow: mixed streaming and random access pattern

Flows with parallelism-friendly access patterns are susceptible to interference from flows whose access patterns do not exploit parallelism
State-of-the-art SSD I/O schedulers **prioritize reads over writes**

When flows have **different read/write ratios**, existing schedulers do not effectively provide fairness.
**Source 4: Different Garbage Collection Demands**

- **NAND flash memory performs writes out of place**
  - Erases can only happen on an entire flash block (hundreds of flash pages)
  - Pages marked invalid during write

- **Garbage collection (GC)**
  - Selects a block with mostly-invalid pages
  - Moves any remaining valid pages
  - Erases that block

- **High-GC flow**: flows with a higher write intensity induce more garbage collection activities
**Source 4: Different Garbage Collection Demands**

- **Garbage collection** may block user I/O requests
  - Primarily depends on the **write intensity** of the workload
- An experiment with two 100%-write flows with different intensities
  - Base flow: low intensity and **moderate** GC demand
  - Interfering flow: different write intensities from **low-GC** to **high-GC**

**Lower fairness due to GC execution**

- The GC activities of a **high-GC** flow can unfairly block flash transactions of a **low-GC** flow
Summary: Source of Unfairness in SSDs

- Four major sources of unfairness in modern SSDs
  1. I/O intensity
  2. Request access patterns
  3. Read/write ratio
  4. Garbage collection demands

OUR GOAL

Design an I/O request scheduler for SSDs that
(1) provides fairness among flows
by mitigating all four sources of interference, and
(2) maximizes performance and throughput
Outline

Background: Modern SSD Design

Unfairness Across Multiple Applications in Modern SSDs

**FLIN:** Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion
FLIN: Flash-Level INterference-aware Scheduler

- FLIN is a three-stage I/O request scheduler
  - Replaces existing transaction scheduling unit
  - Takes in flash transactions, reorders them, sends them to flash channel
- Identical throughput to state-of-the-art schedulers
- Fully implemented in the SSD controller firmware
  - No hardware modifications
  - Requires < 0.06% of the DRAM available within the SSD
Three Stages of FLIN

- **Stage 1: Fairness-aware Queue Insertion**
  relieves I/O intensity and access pattern interference

```
<table>
<thead>
<tr>
<th>Tail</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>
```

*From high-intensity flows*  *From low-intensity flows*
Three Stages of FLIN

- **Stage 1: Fairness-aware Queue Insertion**
  *relieves I/O intensity and access pattern interference*

- **Stage 2: Priority-aware Queue Arbitration**
  *enforces priority levels that are assigned to each flow by the host*
Three Stages of FLIN

- **Stage 1**: Fairness-aware Queue Insertion
  
  relieves I/O intensity and access pattern interference

- **Stage 2**: Priority-aware Queue Arbitration
  
  enforces priority levels that are assigned to each flow by the host

- **Stage 3**: Wait-balancing Transaction Selection
  
  relieves read/write ratio and garbage collection demand interference
Background: Modern SSD Design

Unfairness Across Multiple Applications in Modern SSDs

FLIN:
Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion
Evaluation Methodology

- **MQSim**: [https://github.com/CMU-SAFARI/MQSim](https://github.com/CMU-SAFARI/MQSim) [FAST 2018]
  - Protocol: NVMe 1.2 over PCIe
  - User capacity: 480GB
  - Organization: 8 channels, 2 planes per die, 4096 blocks per plane, 256 pages per block, 8kB page size

- 40 workloads containing four randomly-selected storage traces
  - Each storage trace is collected from real enterprise/datacenter applications: UMass, Microsoft production/enterprise
  - Each application classified as low-interference or high-interference
Two Baseline Schedulers

- **Sprinkler** [Jung+ HPCA 2014]
  a state-of-the-art device-level high-performance scheduler

- **Sprinkler+Fairness** [Jung+ HPCA 2014, Jun+ NVMSA 2015]
  we add a state-of-the-art fairness mechanism to Sprinkler that was previously proposed for OS-level I/O scheduling
  - Does not have direct information about the internal resources and mechanisms of the SSD
  - Does not mitigate all four sources of interference
FLIN improves fairness by an average of 70%, by mitigating *all* four major sources of interference.
FLIN Improves Performance Over the Baselines

**FLIN improves performance by an average of 47%, by making use of idle resources in the SSD and improving the performance of low-interference flows**
Other Results in the Paper

- Fairness and weighted speedup for each workload
  - FLIN improves fairness and performance for all workloads

- Maximum slowdown
  - Sprinkler/Sprinkler+Fairness: several applications with maximum slowdown over 500x
  - FLIN: no flow with a maximum slowdown over 80x

- Effect of each stage of FLIN on fairness and performance

- Sensitivity study to FLIN and SSD parameters

- Effect of write caching
Background: Modern SSD Design

Unfairness Across Multiple Applications in Modern SSDs

FLIN:
Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion
Conclusion

- Modern solid-state drives (SSDs) use new storage protocols (e.g., NVMe) that eliminate the OS software stack
  - Enables **high throughput**: millions of IOPS
  - OS software stack elimination **removes existing fairness mechanisms**
  - Highly unfair **slowdowns** on real state-of-the-art SSDs

- **FLIN**: a new I/O request scheduler for modern SSDs designed to provide both fairness and high performance
  - Mitigates all four sources of inter-application interference
    - Different I/O intensities
    - Different request access patterns
    - Different read/write ratios
    - Different garbage collection demands
  - Implemented fully in the SSD controller firmware, uses < 0.06% of DRAM
  - FLIN improves **fairness by 70%** and **performance by 47%** compared to a state-of-the-art I/O scheduler (Sprinkler+Fairness)
Agenda

- Overview on SSD Organization
  - Storage Controller & Request Handling
  - NAND Flash Hierarchy
  - NAND Flash Read/Write Operations

- Address Mapping and Garbage Collection

- I/O Scheduling
HeatWatch

Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness

Yixin Luo    Saugata Ghose    Yu Cai    Erich F. Haratsch    Onur Mutlu

Carnegie Mellon    SK hynix    ETH Zürich

SAFARI    SEAGATE
Storage Technology Drivers - 2018

3D NAND Flash Memory

Stacked layers
Executive Summary

• 3D NAND flash memory susceptible to retention errors
  • Charge leaks out of flash cell
  • Two unreported factors: self-recovery and temperature

• We study self-recovery and temperature effects

• Experimental characterization of real 3D NAND chips

• Unified Self-Recovery and Temperature (URT) Model
  • Predicts impact of retention loss, wearout, self-recovery, temperature on flash cell voltage
  • Low prediction error rate: 4.9%

• We develop a new technique to improve flash reliability

• HeatWatch
  • Uses URT model to find optimal read voltages for 3D NAND flash
  • Improves flash lifetime by 3.85x
Outline

• Executive Summary

**Background on NAND Flash Reliability**

• Characterization of Self-Recovery and Temperature Effect on Real 3D NAND Flash Memory Chips

• URT: Unified Self-Recovery and Temperature Model

• HeatWatch Mechanism

• Conclusion
3D NAND Flash Memory Background

3D NAND Flash Memory

Flash Cell

Charge = Threshold Voltage

Higher Voltage State
Data Value = 0

Read Reference Voltage

Lower Voltage State
Data Value = 1
Flash Wearout

Program/Erase (P/E) $\rightarrow$ Wearout

Wearout Introduces Errors

Wearout Effects:

1. Retention Loss (voltage shift over time)

2. Program Variation (init. voltage difference b/w states)
Improving Flash Lifetime

Errors introduced by wearout limit flash lifetime
(measured in P/E cycles)

Two Ways to Improve Flash Lifetime

 Exploiting the Self-Recovery Effect

 Exploiting the Temperature Effect
Exploiting the Self-Recovery Effect

Partially repairs damage due to wearout

Dwell Time: Idle Time Between P/E Cycles

Longer Dwell Time: More Self-Recovery

Reduces Retention Loss
Exploiting the Temperature Effect

High Program Temperature

High Storage Temperature

Voltage

Increases Program Variation

Accelerates Retention Loss
Prior Studies of Self-Recovery/Temperature

<table>
<thead>
<tr>
<th></th>
<th>Planar (2D) NAND</th>
<th>3D NAND</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Self-Recovery</strong></td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Effect</strong></td>
<td></td>
<td>Mielke 2006</td>
</tr>
<tr>
<td><strong>Temperature</strong></td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Effect</strong></td>
<td></td>
<td>JEDEC 2010 (no characterization)</td>
</tr>
</tbody>
</table>
Outline

• Executive Summary

• Background on NAND Flash Reliability

• Characterization of Self-Recovery and Temperature Effect on Real 3D NAND Flash Memory Chips

• URT: Unified Self-Recovery and Temperature Model

• HeatWatch Mechanism

• Conclusion
Characterization Methodology

- Modified firmware version in the flash controller
  - Control the read reference voltage of the flash chip
  - Bypass ECC to get raw NAND data (with raw bit errors)
- Control temperature with a heat chamber
Characterized Devices

Real 30-39 Layer 3D MLC NAND Flash Chips
MLC Threshold Voltage Distribution Background

![Diagram showing MLC Threshold Voltage Distribution]

- **Lowest Voltage State**: 11
- **Read Reference Voltage**: 10
- **Second Lowest State**: 00
- **Highest Voltage State**: 01

Threshold Voltage Distribution

```
Probability

Threshold Voltage
```

277
Characterization Goal

Characterized Metrics

Retention Loss Speed
(how fast voltage shifts over time)

Program Variation
(initial voltage difference between states)

Characterized Phenomena

Self-Recovery Effect

Temperature Effect

Self-Recovery Effect

Increasing dwell time from 1 minute to 2.3 hours slows down retention loss speed by 40%.
Increasing program temperature from 0°C to 70°C improves program variation by 21%
Lowering storage temperature from 70°C to 0°C slows down retention loss speed by 58%
Characterization Summary

Major Results:
• *Self-recovery* affects retention loss speed
• Program *temperature* affects program variation
• *Storage temperature* affects retention loss speed

Unified Model

Other Characterizations Methods in the Paper:
• More detailed results on self-recovery and temperature
  • Effects on error rate
  • Effects on threshold voltage distribution
• Effects of recovery cycle (P/E cycles with long dwell time) on retention loss speed
Outline

• Executive Summary

• Background on NAND Flash Reliability

• Characterization of Self-Recovery and Temperature Effect on Real 3D NAND Flash Memory Chips

• **URT: Unified Self-Recovery and Temperature Model**

• HeatWatch Mechanism

• Conclusion
Minimizing 3D NAND Errors

Optimal read reference voltage minimizes 3D NAND errors
Predicting the Mean Threshold Voltage

Our URT Model:

\[ V = V_0 + \Delta V \]

- **Mean Threshold Voltage**
- **Initial Voltage Before Retention (Program Variation)**
- **Voltage Shift Due to Retention Loss**
URT Model Overview

1. Program Variation Component
   - PEC
   - $T_p$

2. Self-Recovery and Retention Component
   - $t_{r,\text{eff}}$
   - PEC
   - $t_{d,\text{eff}}$

3. Temperature Scaling Component
   - $t_r$
   - $T_r$
   - $t_d$
   - $T_d$

Initial Voltage Before Retention

Voltage Shift Due to Retention Loss

$$V = V_0 + \Delta V$$
1. Program Variation Component

P/E Cycle

PEC

T_p

Program Temperature

V_0

Initial Voltage

\[ V_0 = A \cdot T_p \cdot PEC + B \cdot T_p + C \cdot PEC + D \]

Validation: \( R^2 = 91.7\% \)
2. Self-Recovery and Retention Component

Retention Time: $t_r$

P/E Cycle: PEC

Dwell Time: $T_d$

Retention Shift: $\Delta V$

Validation: 3x more accurate than state-of-the-art model

$$\Delta V(ter, ted, PEC) = b \cdot (PEC + c) \cdot \ln \left(1 + \frac{ter}{t_0 + a \cdot ted}\right)$$
3. Temperature Scaling Component

**Arrhenius Equation:**

\[ AF = \frac{t_{\text{real}}}{t_{\text{room}}} = \exp \left( \frac{E_a}{k_B} \cdot \left( \frac{1}{T_{\text{real}}} - \frac{1}{T_{\text{room}}} \right) \right) \]

**Validation:** Adjust an important parameter, \( E_a \), from 1.1 eV to 1.04 eV
Initial Voltage
Before Retention
Voltage Shift
Due to Retention Loss

\[ V = V_0 + \Delta V \]

1. Program Variation Component
   - PEC
   - \( t_p \)

2. Self-Recovery and Retention Component
   - \( t_{r,\text{eff}} \)
   - PEC
   - \( t_{d,\text{eff}} \)

3. Temperature Scaling Component
   - \( t_r \)
   - \( T_r \)
   - \( t_d \)
   - \( T_d \)

Validation:
Prediction Error Rate = 4.9%
Outline

• Executive Summary
• Background on NAND Flash Reliability
• Characterization of Self-Recovery and Temperature Effect on Real 3D NAND Flash Memory Chips
• URT: Unified Self-Recovery and Temperature Model
• HeatWatch Mechanism
• Conclusion
HeatWatch Mechanism

• Key Idea

• Predict change in threshold voltage distribution by using the URT model

• Adapt read reference voltage to near-optimal \( V_{opt} \) based on predicted change in voltage distribution
HeatWatch Mechanism Overview

Tracking Components

- SSD Temperature
- Dwell Time
- P/E Cycles & Retention Time

Prediction Components

- $V_{opt}$ Prediction
- Fine-Tuning URT Parameters
Tracking SSD Temperature

**Tracking Components**

- SSD Temperature
- Dwell Time
- P/E Cycles & Retention Time

**Prediction Components**

- Use existing sensors in the SSD
- **Precompute** temperature scaling factor at logarithmic time intervals

- $V_{opt}$ Prediction
- Fine-Tuning URT Parameters
Tracking Dwell Time

Tracking Components

SSD Temperature

Dwell Time

P/E Cycles & Retention Time

• Only need to log the timestamps of last 20 full drive writes
• Self-recovery effect diminishes after 20 P/E cycles

Prediction Components

$V_{opt}$ Prediction

Fine-Tuning URT Parameters
Tracking P/E Cycles and Retention Time

**Tracking Components**

- **SSD Temperature**
- **Dwell Time**
- **P/E Cycles & Retention Time**

  - P/E cycle count **already recorded** by SSD
  - **Log write timestamp** for each block
  - Retention time = read timestamp – write timestamp

**Prediction Components**

- $V_{\text{opt}}$ Prediction
- Fine-Tuning URT Parameters
Predicting Optimal Read Reference Voltage

Tracking Components

- SSD Temperature
- Dwell Time
- P/E Cycles & Retention Time

Prediction Components

- Calculate URT using tracked information
- Modeling error: 4.9%

$V_{opt}$ Prediction

Fine-Tuning URT Parameters
Fine-Tuning URT Parameters Online

Tracking Components

- SSD Temperature
- Dwell Time
- P/E Cycles & Retention Time

Prediction Components

- Accommodates chip-to-chip variation
- Uses periodic sampling
HeatWatch Mechanism Summary

Tracking Components

- SSD Temperature
- Dwell Time
- P/E Cycles & Retention Time

Storage Overhead: 0.16% of DRAM in 1TB SSD

URT

Prediction Components

- $V_{opt}$ Prediction
- Fine-Tuning URT Parameters

Latency Overhead: < 1% of flash read latency
HeatWatch Evaluation Methodology

• 28 real workload storage traces
  • MSR-Cambridge
  • We use real dwell time, retention time values obtained from traces

• Temperature Model:
  Trigonometric function + Gaussian noise
  • Represents periodic temperature variation in each day
  • Includes small transient temperature variation
HeatWatch greatly improves Flash lifetime by capturing the effect of retention, wearout, self-recovery, and temperature.

- HeatWatch improves lifetime by 3.85x over Fixed $V_{ref}$.
- HeatWatch improves lifetime by 24% over state-of-the-art.
Outline

• Executive Summary
• Background on NAND Flash Reliability
• Characterization of Self-Recovery and Temperature Effect on Real 3D NAND Flash Memory Chips
• URT: Unified Self-Recovery and Temperature Model
• HeatWatch Mechanism

• Conclusion
Conclusion

• 3D NAND flash memory susceptible to retention errors
  • Charge leaks out of flash cell
  • Two unreported factors: self-recovery and temperature

• We study self-recovery and temperature effects

  • Experimental characterization of real 3D NAND chips

• Unified Self-Recovery and Temperature (URT) Model
  • Predicts impact of retention loss, wearout, self-recovery, temperature on flash cell voltage
  • Low prediction error rate: 4.9%

• We develop a new technique to improve flash reliability

  • HeatWatch
    • Uses URT model to find optimal read voltages for 3D NAND flash
    • Improves flash lifetime by 3.85x
HeatWatch

Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness

Yixin Luo    Saugata Ghose    Yu Cai    Erich F. Haratsch    Onur Mutlu

Carnegie Mellon    SK hynix    ETH Zürich    SAFARI    SEAGATE