

Computer Architecture

Lecture 17: Flash Memory and Solid-State Drives II

Dr. Mohammad Sadrosadati

Prof. Onur Mutlu

ETH Zürich

Fall 2023

23 November 2023

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, YIXIN LUO, AND ONUR MUTLU

<https://arxiv.org/pdf/1706.08642>



More Up-to-date Version

- Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,
**"Errors in Flash-Memory-Based Solid-State Drives: Analysis,
Mitigation, and Recovery"**
Invited Book Chapter in Inside Solid State Drives, 2018.
[Preliminary arxiv.org version]

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

3D NAND Flash Reliability I [HPCA'18]

- Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu,
"HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature-Awareness"

Proceedings of the 24th International Symposium on High-Performance Computer Architecture (HPCA), Vienna, Austria, February 2018.

[[Lightning Talk Video](#)]

[[Slides \(pptx\)](#) ([pdf](#))] [[Lightning Session Slides \(pptx\)](#) ([pdf](#))]

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 University* [‡]*Seagate Technology* [§]*ETH Zürich*

3D NAND Flash Reliability II [SIGMETRICS'18]

- 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"

Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Irvine, CA, USA, June 2018.

[[Abstract](#)]

[[POMACS Journal Version \(same content, different format\)](#)]

[[Slides \(pptx\)](#) ([pdf](#))]

Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation

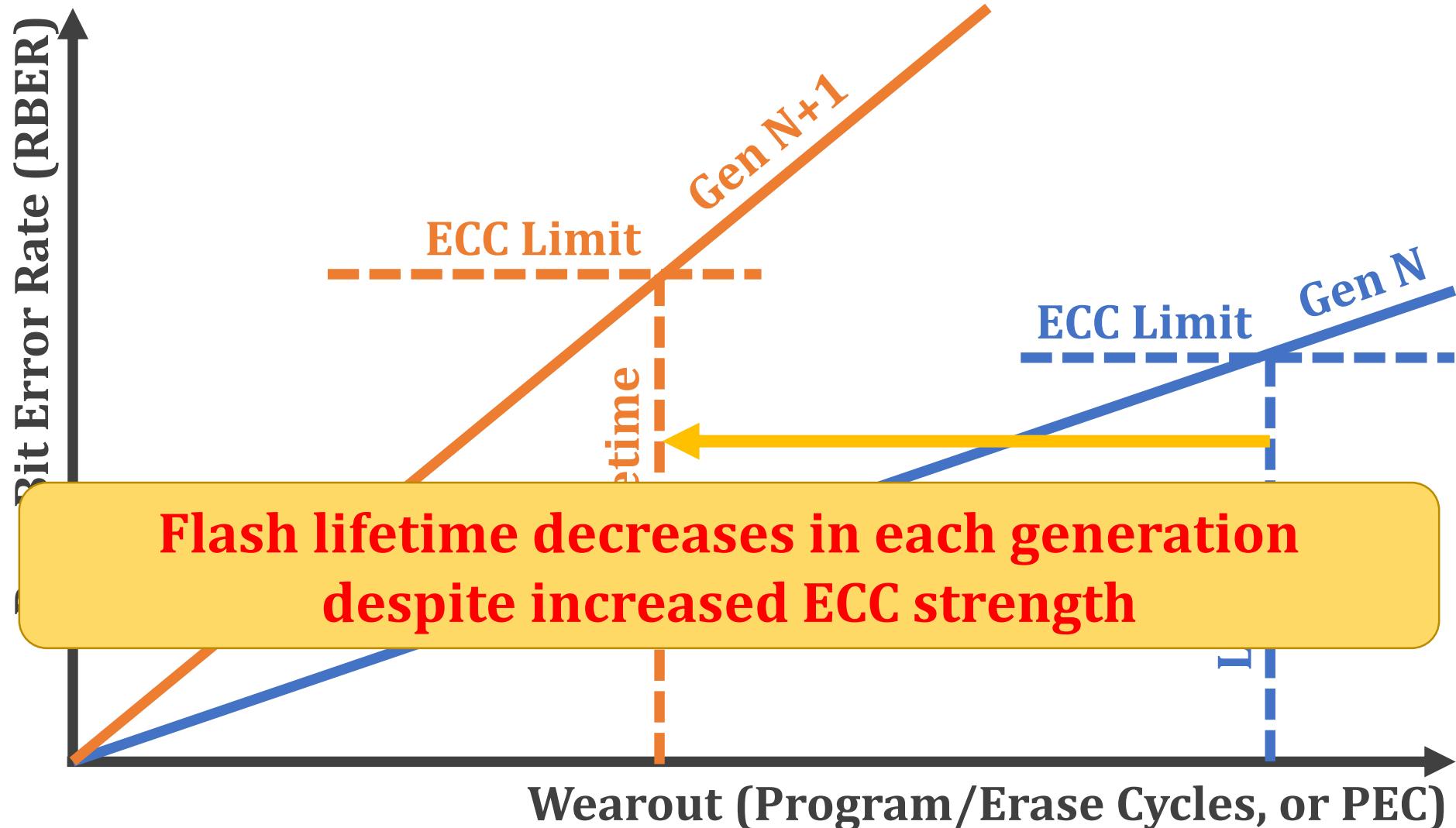
Yixin Luo[†] Saugata Ghose[†] Yu Cai[†] Erich F. Haratsch[‡] Onur Mutlu^{§†}

[†]Carnegie Mellon University

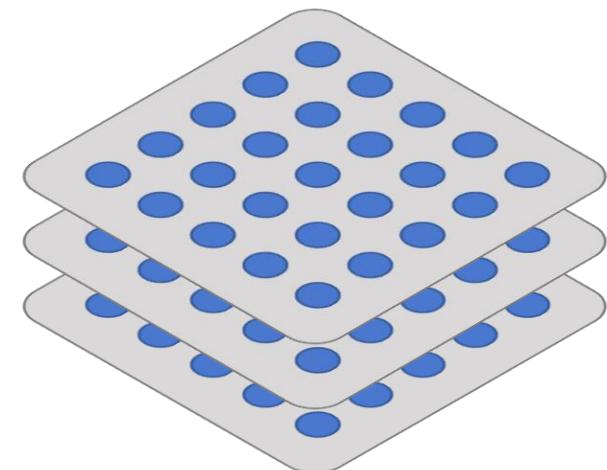
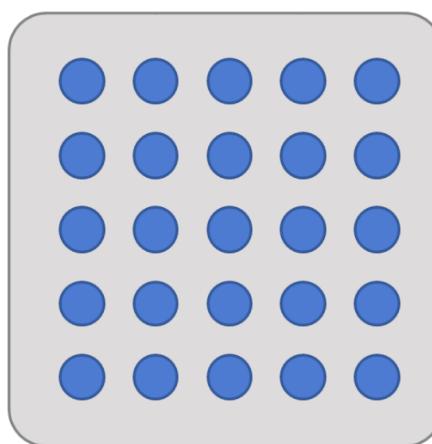
[‡]Seagate Technology

[§]ETH Zürich

NAND Flash Memory Lifetime Problem



Planar vs. 3D NAND Flash Memory



**Planar NAND
Flash Memory**

Scaling

Reduce flash cell size,
Reduce distance b/w cells

**3D NAND
Flash Memory**

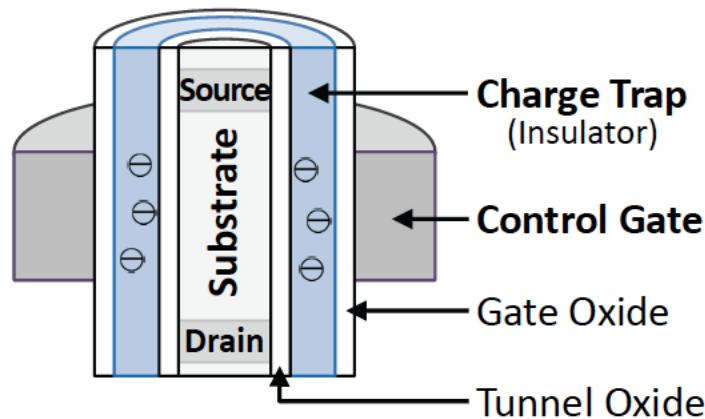
Reliability

Scaling hurts reliability

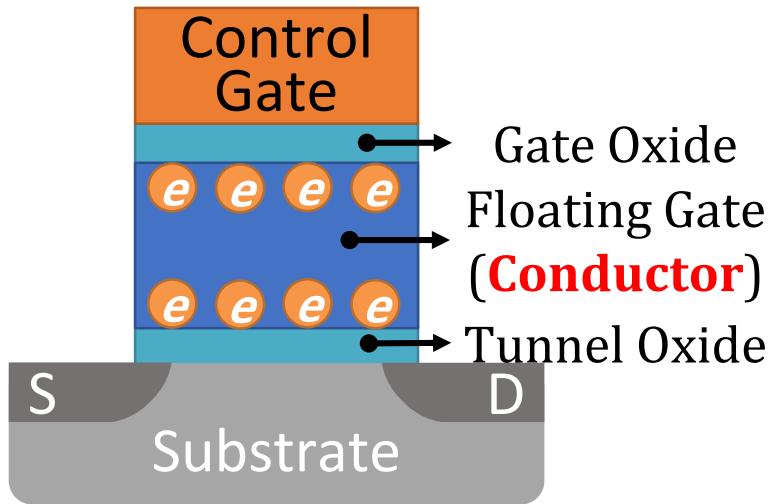
Not well studied!

Charge Trap Based 3D Flash Cell

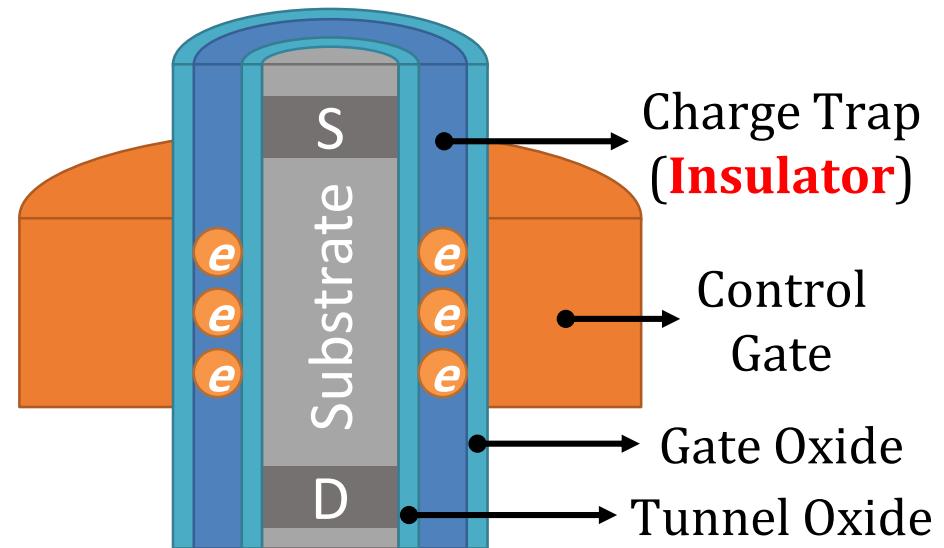
■ Cross-section of a charge trap transistor



2D vs. 3D Flash Cell Design



2D Floating-Gate Cell



3D Charge-Trap Cell

3D NAND Flash Memory Organization

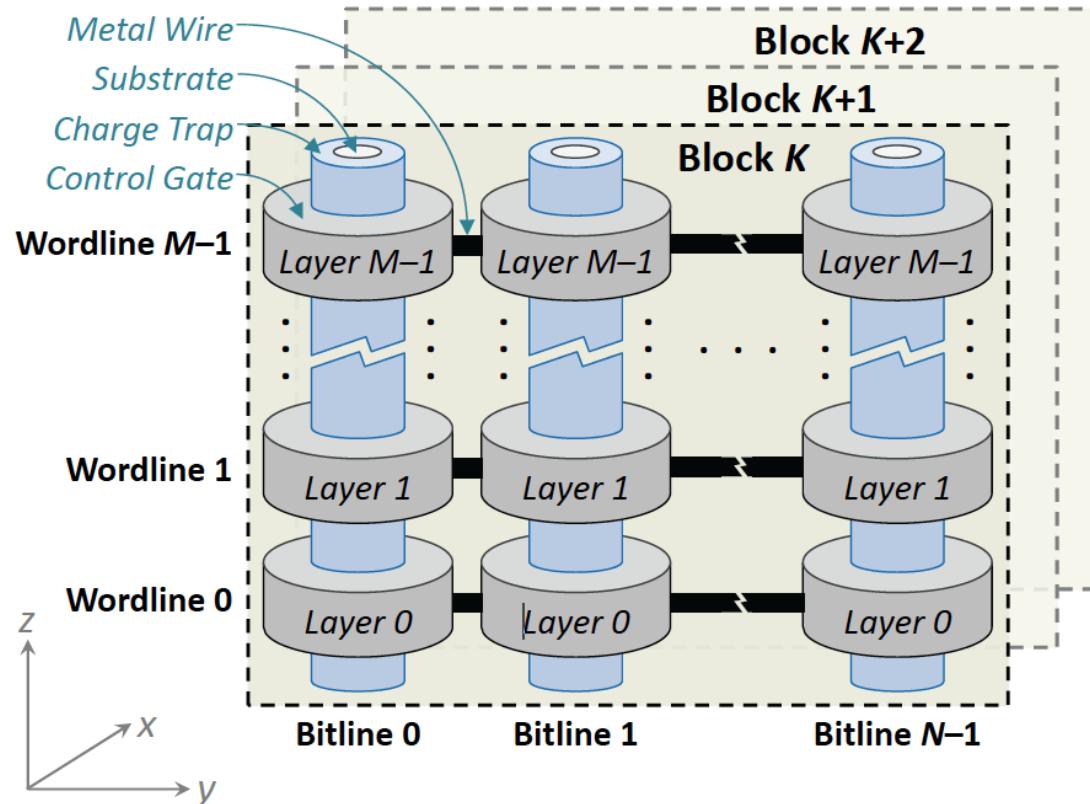


Fig. 43. Organization of flash cells in an M -layer 3D charge trap NAND flash memory chip, where each block consists of M wordlines and N bitlines.

More Background and State-of-the-Art

- Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,
"Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery"
Invited Book Chapter in Inside Solid State Drives, 2018.
[Preliminary arxiv.org version]

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.

Error Type	Change in 3D vs. Planar
P/E Cycling (Section 3.1)	3D is <i>less susceptible</i> , due to current use of charge trap transistors for flash cells
Program (Section 3.2)	3D is <i>less susceptible for now</i> , due to use of one-shot programming (see Section 2.4)
Cell-to-Cell Interference (Section 3.3)	3D is <i>less susceptible for now</i> , due to larger manufacturing process technology
Data Retention (Section 3.4)	3D is <i>more susceptible</i> , due to early retention loss
Read Disturb (Section 3.5)	3D is <i>less susceptible for now</i> , due to larger manufacturing process technology

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

SAFARI



ETH Zürich



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%*

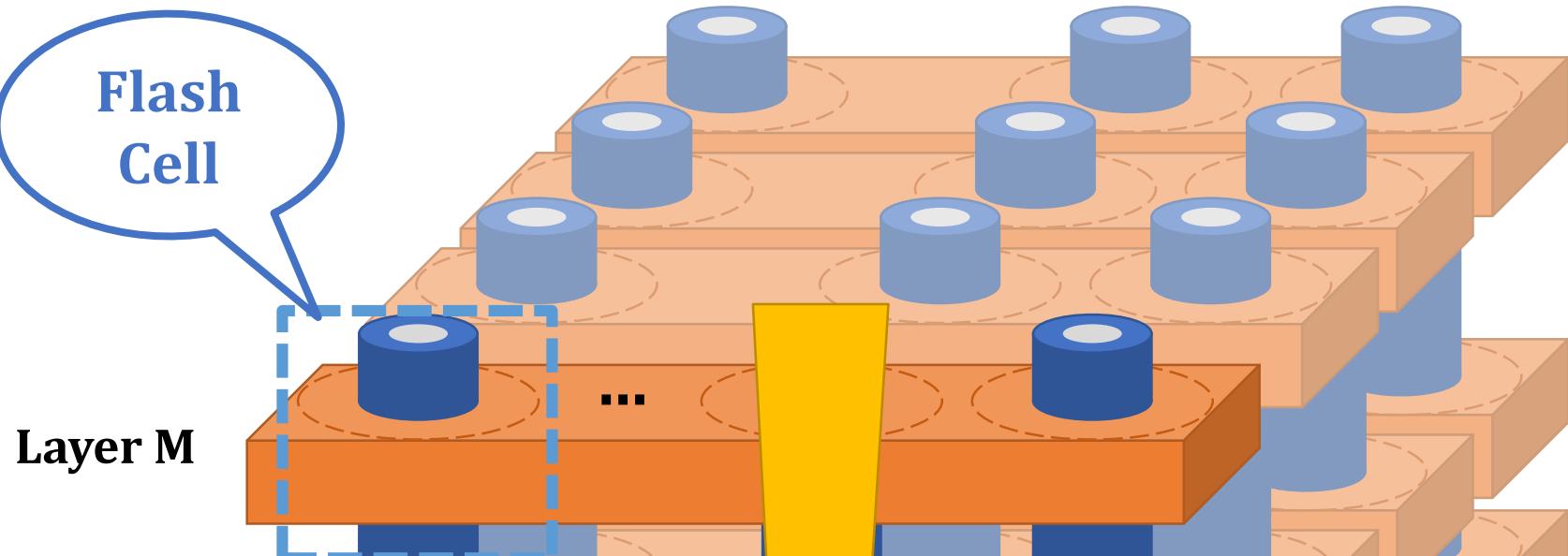
Agenda

- 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

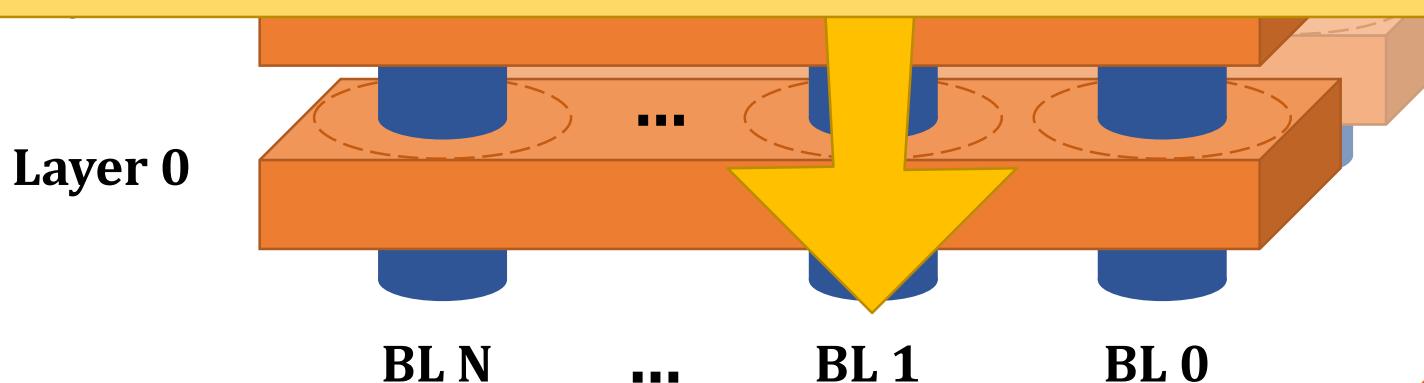
Agenda

- 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

Process Variation Across Layers



Flash cells on different layers may have different error characteristics

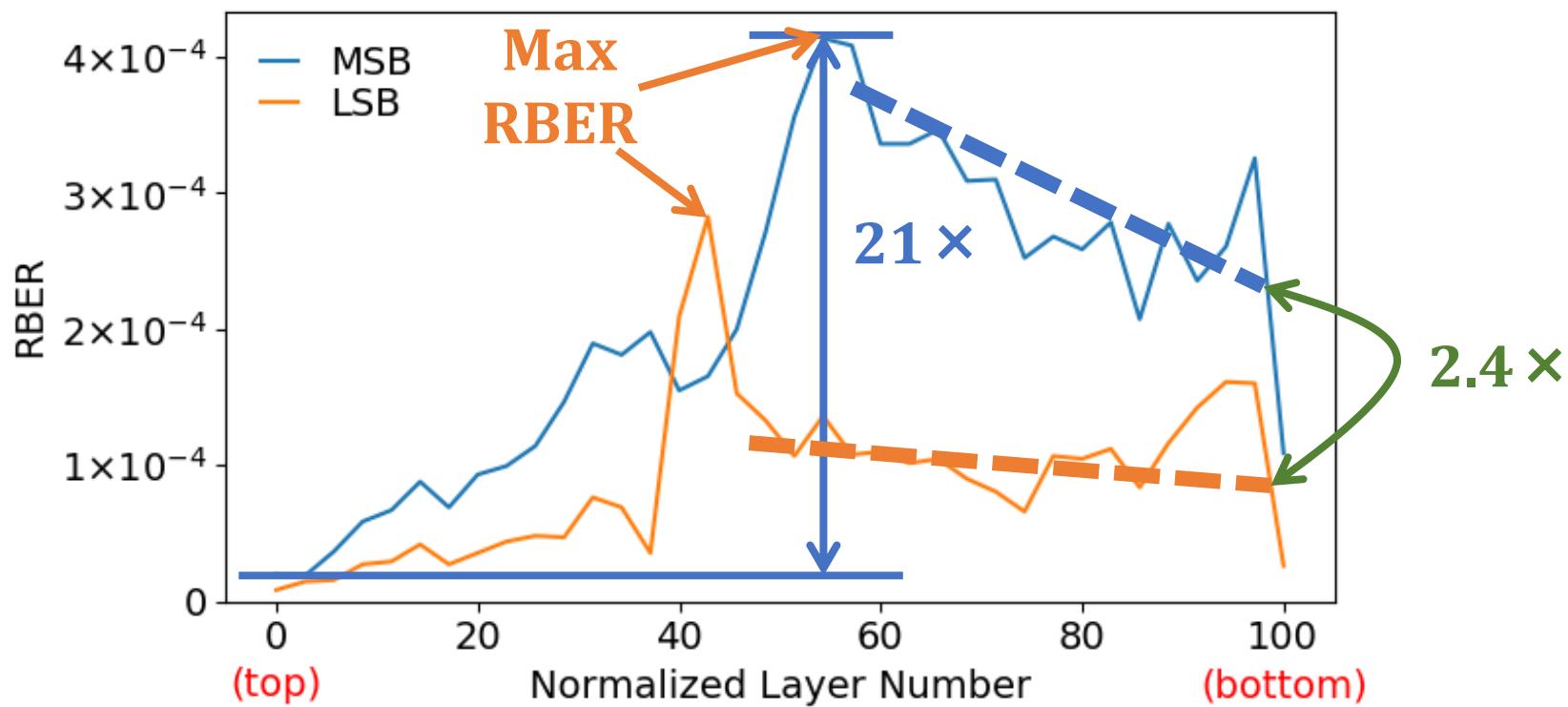


Characterization Methodology

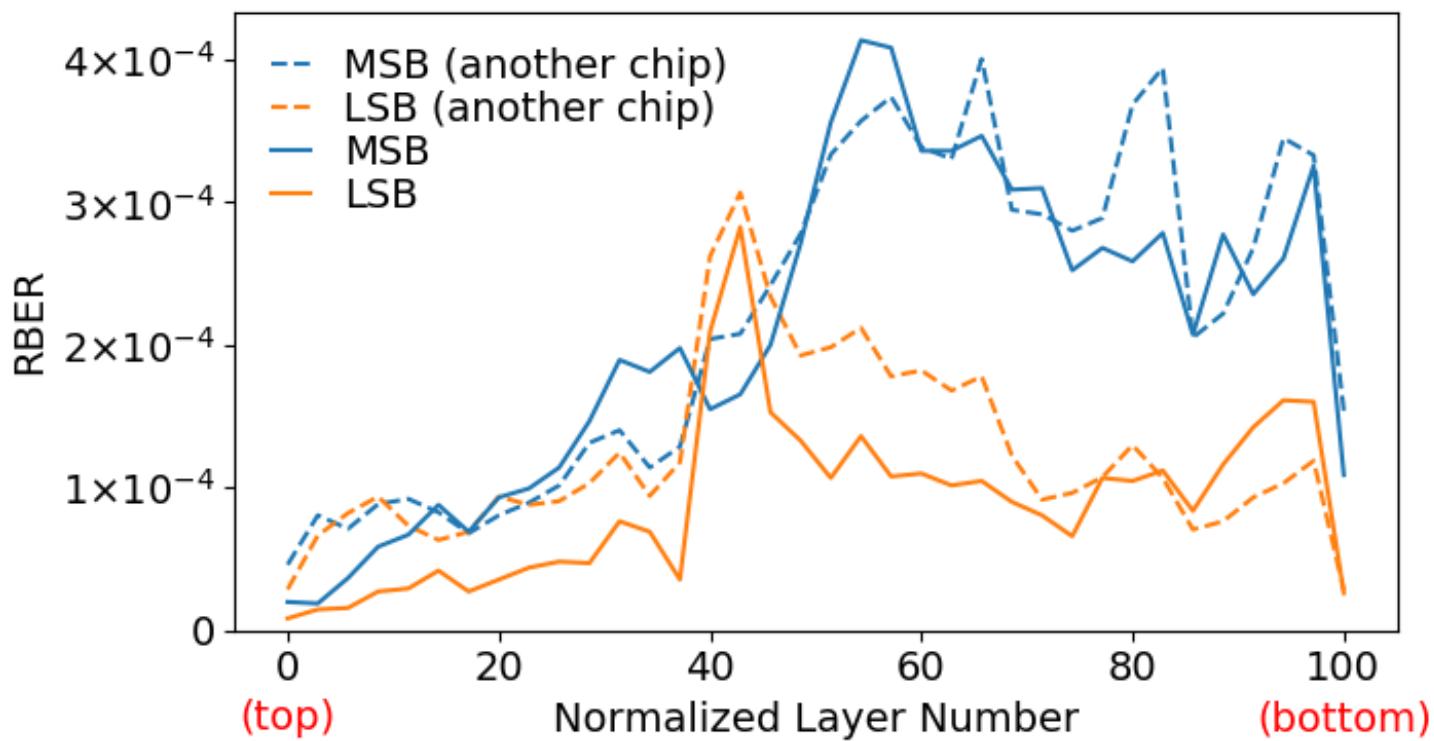
- Modified firmware version in the flash controller
 - Controls the read reference voltage of the flash chip
 - Bypasses ECC to get raw data (with raw bit errors)
- Analysis and post-processing of the data on the server



Layer-to-Layer Process Variation



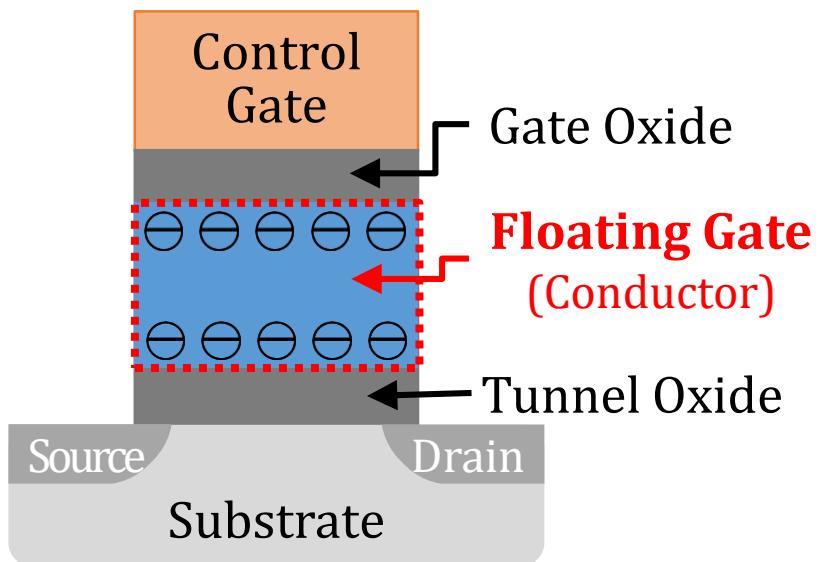
Layer-to-Layer Process Variation



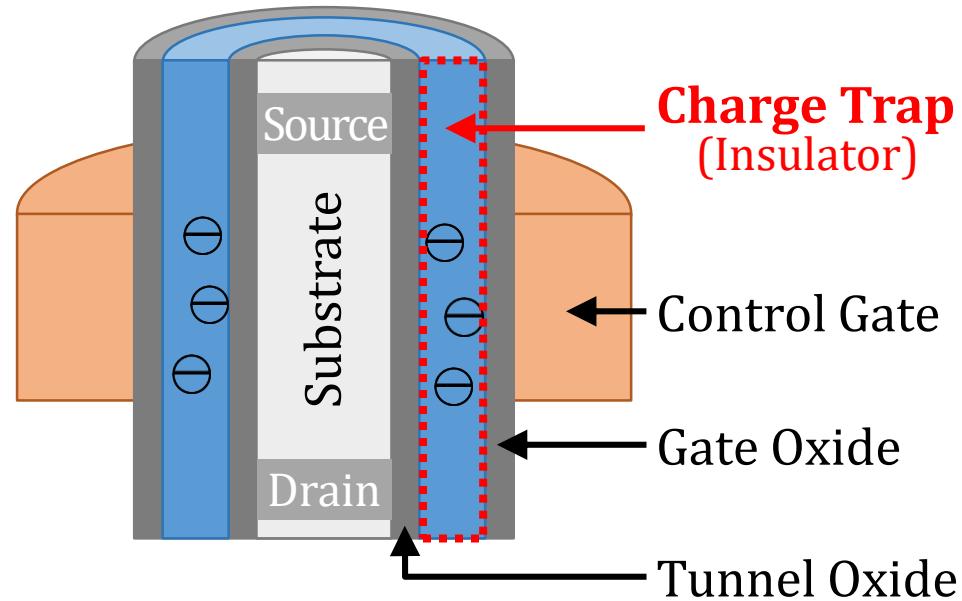
**Large RBER variation
across layers and LSB-MSB pages**

Retention Loss Phenomenon

Planar NAND Cell

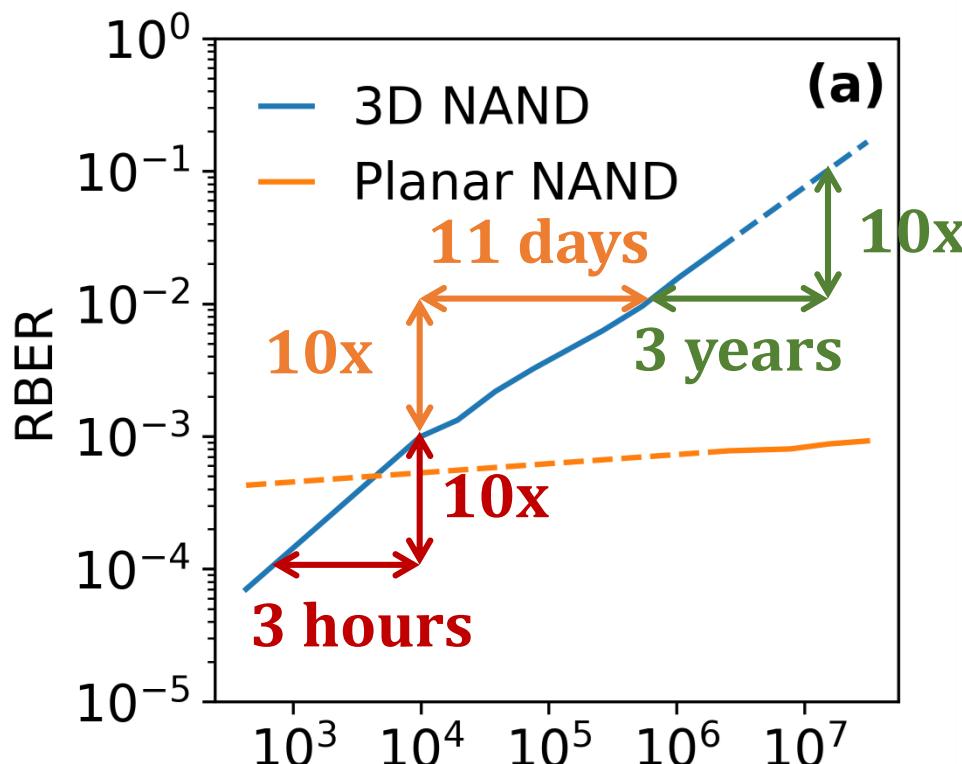


3D NAND Cell



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

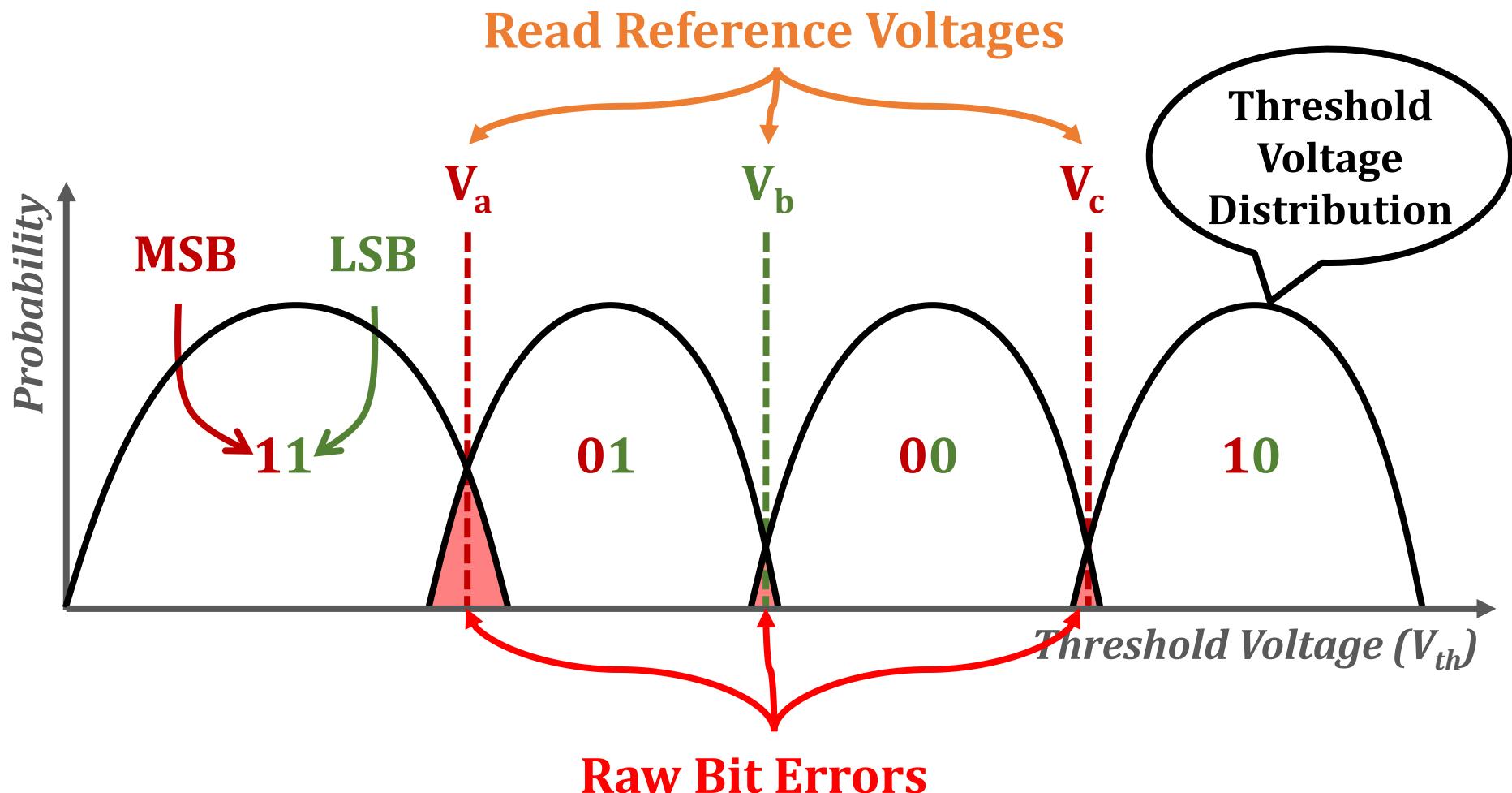
Characterization Summary

- **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

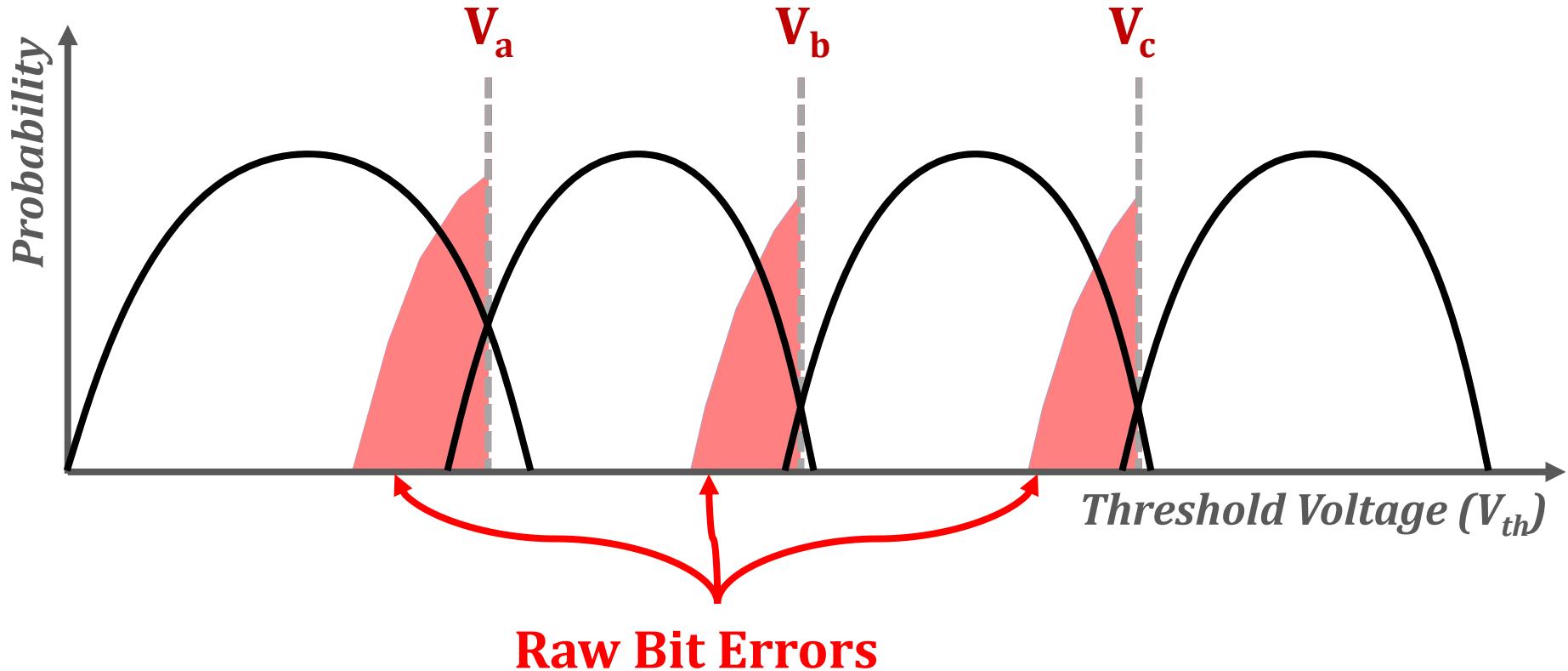
Agenda

- 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

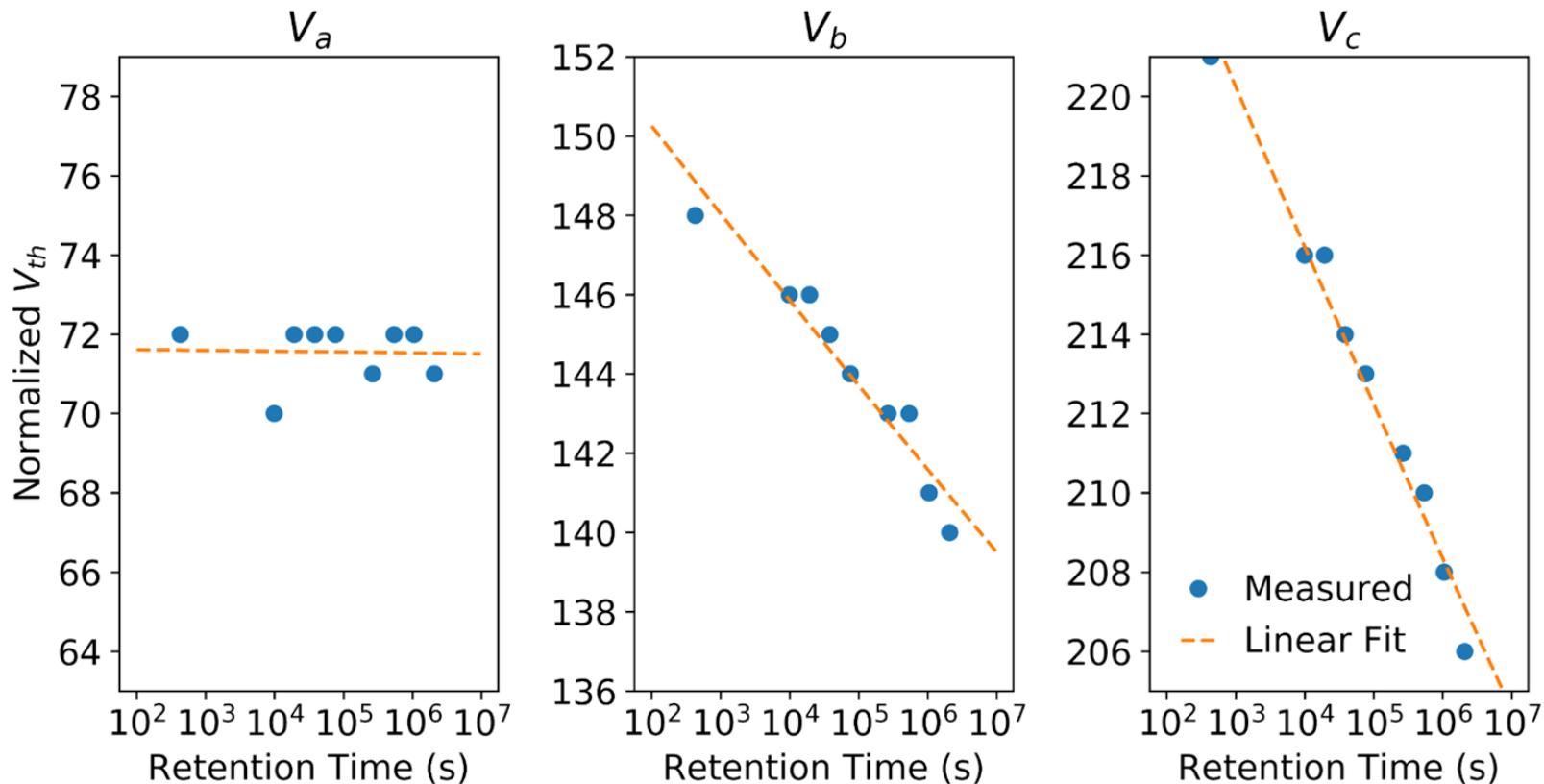
What Do We Model?



Optimal Read Reference Voltage



Retention Loss Model

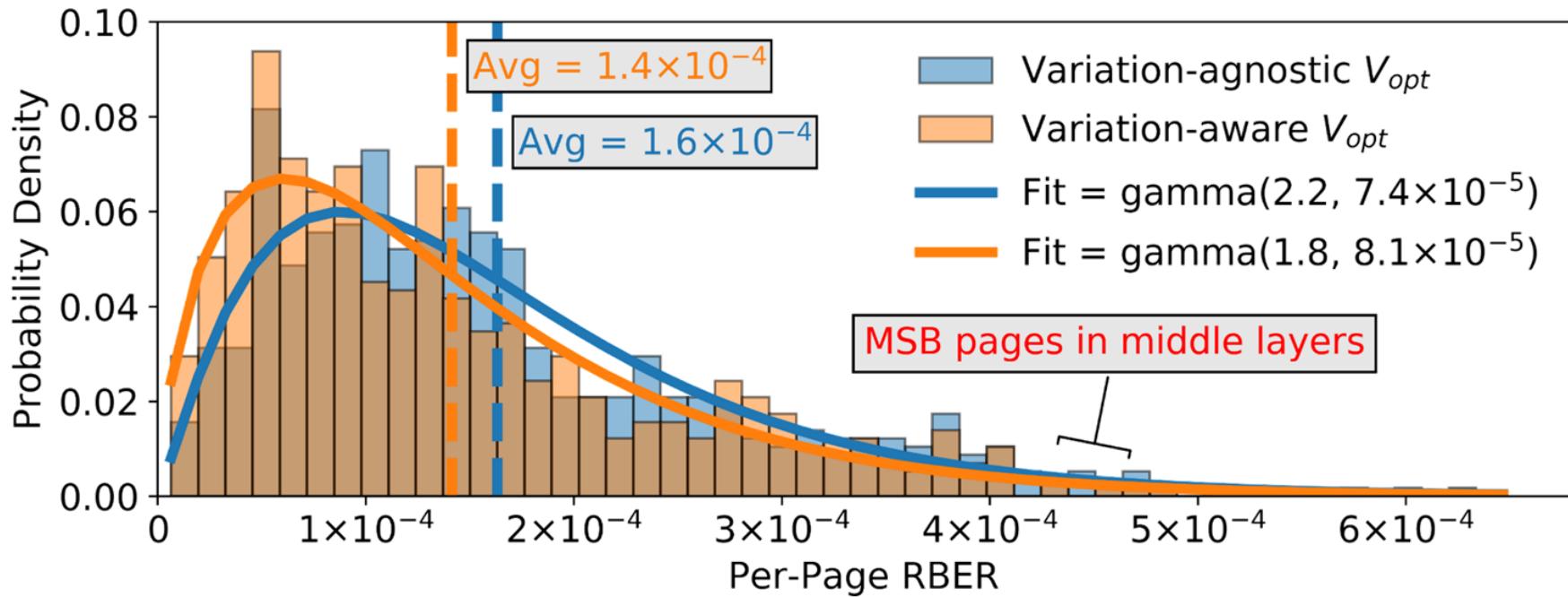


Early retention loss can be modeled as a simple linear function of $\log(\text{retention time})$

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 (μ and σ)
- 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 R² > 89%

RBER Variation Model



Variation-agnostic V_{opt}

- Same V_{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
 - $V_{opt}^{Layer\ aware} = V_{opt}^{Layer\ agnostic} + Offset$
- **Mechanism**
 - **Offset:** Learned once for each chip & stored in a table
 - *Uses $(2 \times Layers)$ Bytes memory per chip*
 - $V_{opt}^{Layer\ agnostic}$: Predicted by any existing V_{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

<i>Wordline #</i>	<i>Layer #</i>	<i>Page</i>	Chip 0	Chip 1	Chip 2	Chip 3
<i>0</i>	<i>0</i>	<i>MSB</i>	Group 0	Group 0	Group 0	Group 0
<i>0</i>	<i>0</i>	<i>LSB</i>	Group 1	Group 1	Group 1	Group 1
<i>1</i>	<i>1</i>	<i>MSB</i>	Group 2	Group 2	Group 2	Group 2
<i>1</i>	<i>1</i>	<i>LSB</i>	Group 3	Group 3	Group 3	Group 3
<i>2</i>	<i>2</i>	<i>MSB</i>	Group 4	Group 4	Group 4	Group 4
<i>2</i>	<i>2</i>	<i>LSB</i>	Group 5	Group 5	Group 5	Group 5
<i>3</i>	<i>3</i>	<i>MSB</i>	Group 6	Group 6	Group 6	Group 6
<i>3</i>	<i>3</i>	<i>LSB</i>	Group 7	Group 7	Group 7	Group 7

**Worst-case RBER in any layer
limits the lifetime of conventional RAID**

LI-RAID: Layer-Interleaved RAID

<i>Wordline #</i>	<i>Layer #</i>	<i>Page</i>	Chip 0	Chip 1	Chip 2	Chip 3
0	0	MSB	Group 0	Blank	Group 4	Group 3
0	0	LSB	Group 1	Blank	Group 5	Group 2
1	1	MSB	Group 2	Group 1	Blank	Group 5
1	1	LSB	Group 3	Group 0	Blank	Group 4
2	2	MSB	Group 4	Group 3	Group 0	Blank
2	2	LSB	Group 5	Group 2	Group 1	Blank
3	3	MSB	Blank	Group 5	Group 2	Group 1
3	3	LSB	Blank	Group 4	Group 3	Group 0

Any page with worst-case RBER can be corrected by other reliable pages in the RAID group

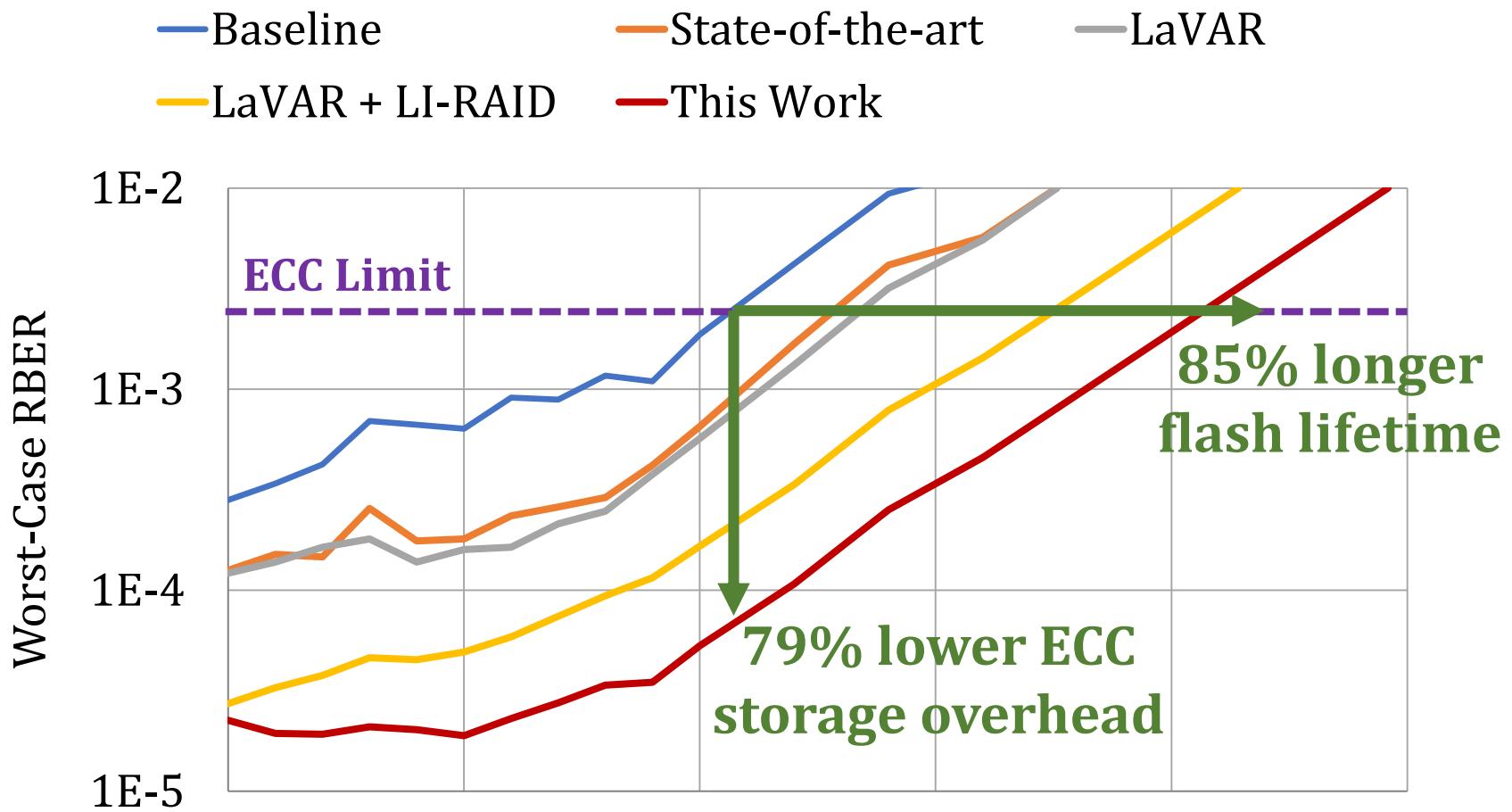
LI-RAID: Layer-Interleaved RAID

- **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_{opt} offset for each layer and apply *layer-aware V_{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_{opt}*
- **Benefits:**
 - Improve flash lifetime by **1.85×** or reduce ECC overhead by **78.9%**
- **ReNAC (in paper):** Reread a failed page using V_{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:* $21\times$ 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

SAFARI



ETH Zürich



3D NAND Flash Reliability II [SIGMETRICS'18]

- 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"

Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Irvine, CA, USA, June 2018.

[[Abstract](#)]

[[POMACS Journal Version \(same content, different format\)](#)]

[[Slides \(pptx\)](#) ([pdf](#))]

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 University

[‡]Seagate Technology

[§]ETH Zürich

One More Idea

WARM

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*

SAFARI

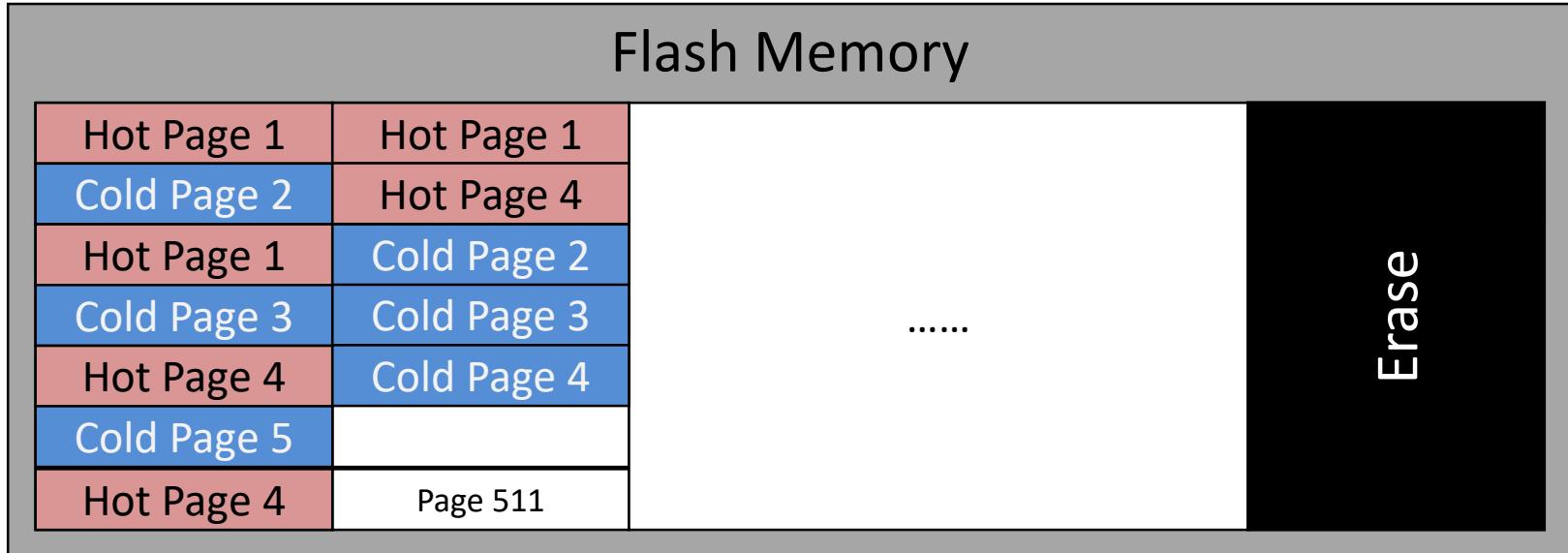
Carnegie Mellon



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

Flash Memory				
Hot Page 1	Cold Page 2	Hot Page 4		Page M
Hot Page 1	Cold Page 3	Hot Page 1		Page M+1
Hot Page 4	Cold Page 5			Page M+2
Hot Page 4			
Hot Page 1
Hot Page 4				
Hot Page 1	Page 511			Page M+255

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



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"
Proceedings of the 31st International Conference on Massive Storage Systems and Technologies (MSST), Santa Clara, CA, June 2015.
[Slides (pptx) (pdf)] [Poster (pdf)]

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

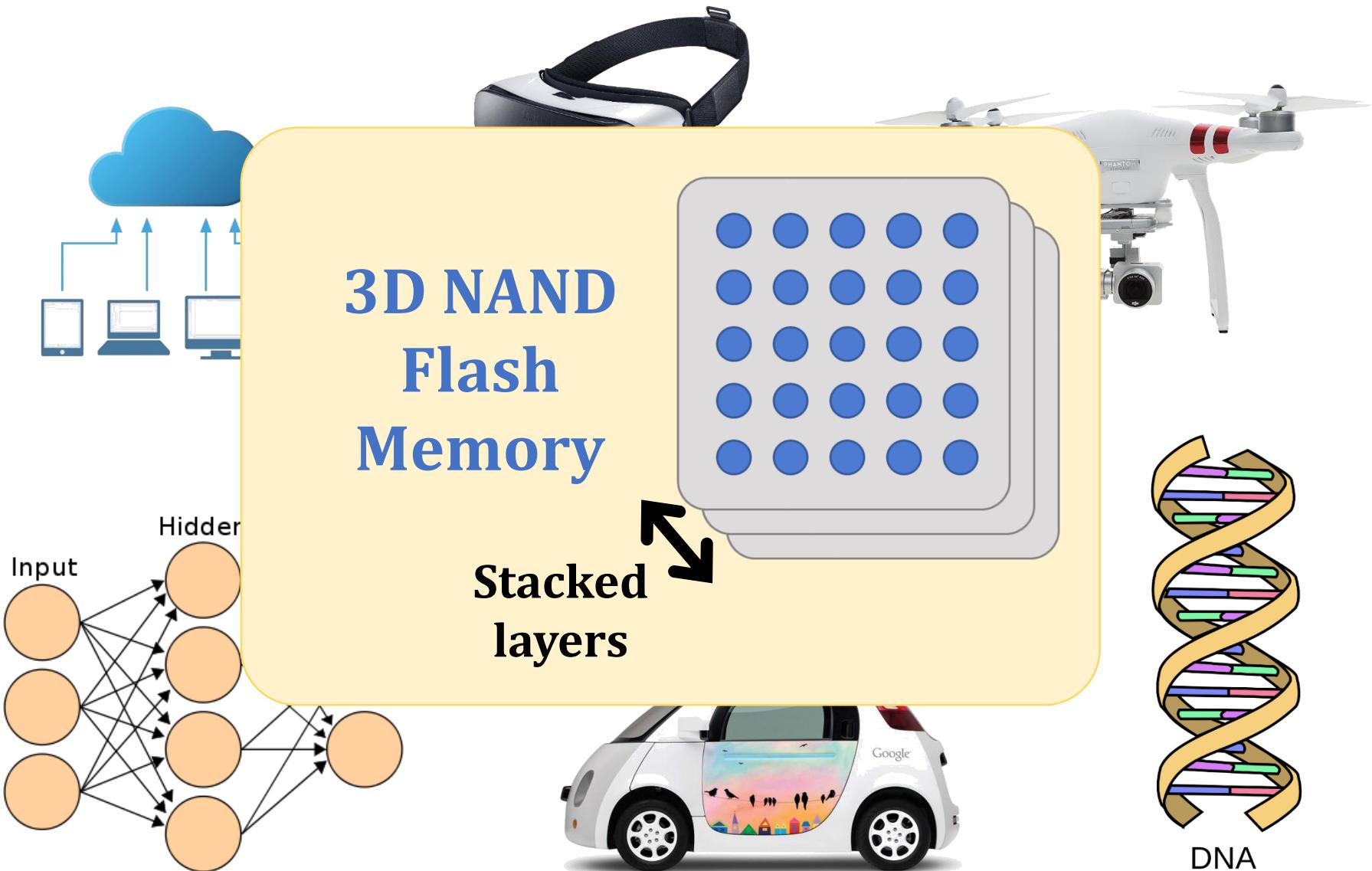
Carnegie Mellon

SAFARI



ETH Zürich

Storage Technology Drivers - 2018



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

- Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,
"Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid State Drives"
Proceedings of the IEEE, September 2017.

- Cai+, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," DATE 2012.
- Cai+, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," ICCD 2012.
- Cai+, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling," DATE 2013.
- Cai+, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," Intel Technology Journal 2013.
- Cai+, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," ICCD 2013.
- Cai+, "Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories," SIGMETRICS 2014.
- Cai+, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery," HPCA 2015.
- Cai+, "Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation," DSN 2015.
- Luo+, "WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management," MSST 2015.
- Meza+, "A Large-Scale Study of Flash Memory Errors in the Field," SIGMETRICS 2015.
- Luo+, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," IEEE JSAC 2016.
- Cai+, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," HPCA 2017.
- Fukami+, "Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices," DFRWS EU 2017.
- Luo+, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature-Awareness," HPCA 2018.
- Luo+, "Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation," SIGMETRICS 2018.
- Cai+, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid State Drives," Proc. IEEE 2017.

NAND Flash Vulnerabilities [HPCA'17]

HPCA, Feb. 2017

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-

belongs to a different flash memory *page* (the unit of data programmed and read at the same time), which we refer to, respectively, as the least significant bit (LSB) page and the most significant bit (MSB) page [5].

A flash cell is programmed by applying a large voltage on the control gate of the transistor, which triggers charge transfer into the floating gate, thereby increasing the threshold voltage. To precisely control the threshold voltage of the cell, the flash memory uses *incremental step pulse programming* (ISPP) [12, 21, 25, 41]. ISPP applies multiple short pulses of the programming voltage to the control gate, in order to increase the cell threshold voltage by some small voltage amount (V_{step}) after each step. Initial MLC designs programmed the threshold voltage in *one shot*, issuing all of the pulses back-to-back to program *both* bits of data at the same time. However, as flash memory scales down, the distance between neighboring flash cells decreases, which

https://people.inf.ethz.ch/omutlu/pub/flash-memory-programming-vulnerabilities_hPCA17.pdf

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, YIXIN LUO, AND ONUR MUTLU

<https://arxiv.org/pdf/1706.08642>



More Up-to-date Version

- Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,
"Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery"
Invited Book Chapter in Inside Solid State Drives, 2018.
[Preliminary arxiv.org version]

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"

Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST), Oakland, CA, USA, February 2018.

[Slides (pptx) (pdf)]

[Source Code]

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

Arash Tavakkol[†], Juan Gómez-Luna[†], Mohammad Sadrosadati[†], Saugata Ghose[‡], Onur Mutlu^{†‡}

[†]*ETH Zürich*

[‡]*Carnegie Mellon University*

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"

Proceedings of the 45th International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, June 2018.

[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"

Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Lausanne, Switzerland, March 2020.

[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"

Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Virtual, March-April 2021.

[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 Park¹ Myungsuk Kim^{2,3} Myoungjun Chun² Lois Orosa¹ Jihong Kim² Onur Mutlu¹

¹ETH Zürich
Switzerland

²Seoul National University
Republic of Korea

³Kyungpook National University
Republic of Korea

DeepSketch [FAST 2022]

- Jisung Park, Jeonggyun Kim, Yeseong Kim, Sungjin Lee, and Onur Mutlu,

"DeepSketch: A New Machine Learning-Based Reference Search Technique for Post-Deduplication Delta Compression"

Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, February 2022.

[Slides (pptx) (pdf)]

[Talk Video (15 minutes)]

DeepSketch: A New Machine Learning-Based Reference Search Technique for Post-Deduplication Delta Compression

Jisung Park^{1*} Jeonggyun Kim^{2*} Yeseong Kim² Sungjin Lee² Onur Mutlu¹

¹*ETH Zürich* ²*DGIST*

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"

Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLoS), Virtual, February-March 2022.

[[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 Ghiasi¹ Jisung Park¹ Harun Mustafa¹ Jeremie Kim¹ Ataberk Olgun¹
Arvid Gollwitzer¹ Damla Senol Cali² Can Firtina¹ Haiyu Mao¹ Nour Almadhoun Alserr¹
Rachata Ausavarungnirun³ Nandita Vijaykumar⁴ Mohammed Alser¹ Onur Mutlu¹

¹ETH Zürich ²Bionano Genomics ³KMUTNB ⁴University 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"
Proceedings of the 49th International Symposium on Computer Architecture (ISCA), New York, June 2022.
[[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 Singh¹ Rakesh Nadig¹ Jisung Park¹
David Novo³ Juan Gómez-Luna¹

¹ETH Zürich

Rahul Bera¹ Nastaran Hajinazar¹
Henk Corporaal² Onur Mutlu¹

²Eindhoven University of Technology ³LIRMM, 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^{§▽} Roknoddin Azizi[§] Geraldo F. Oliveira[§] Mohammad Sadrosadati[§]
Rakesh Nadig[§] David Novo[†] Juan Gómez-Luna[§] Myungsuk Kim[‡] Onur Mutlu[§]

[§]*ETH Zürich*

[▽]*POSTECH*

[†]*LIRMM, Univ. Montpellier, CNRS*

[‡]*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"

Proceedings of the 50th International Symposium on Computer Architecture (ISCA), Orlando, FL, USA, June 2023.

[[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^{§▼} Hamid Sarbazi-Azad^{†‡} Juan Gómez Luna[§] Onur Mutlu[§]

[§]*ETH Zürich*

[▼]*POSTECH*

[†]*Sharif University of Technology*

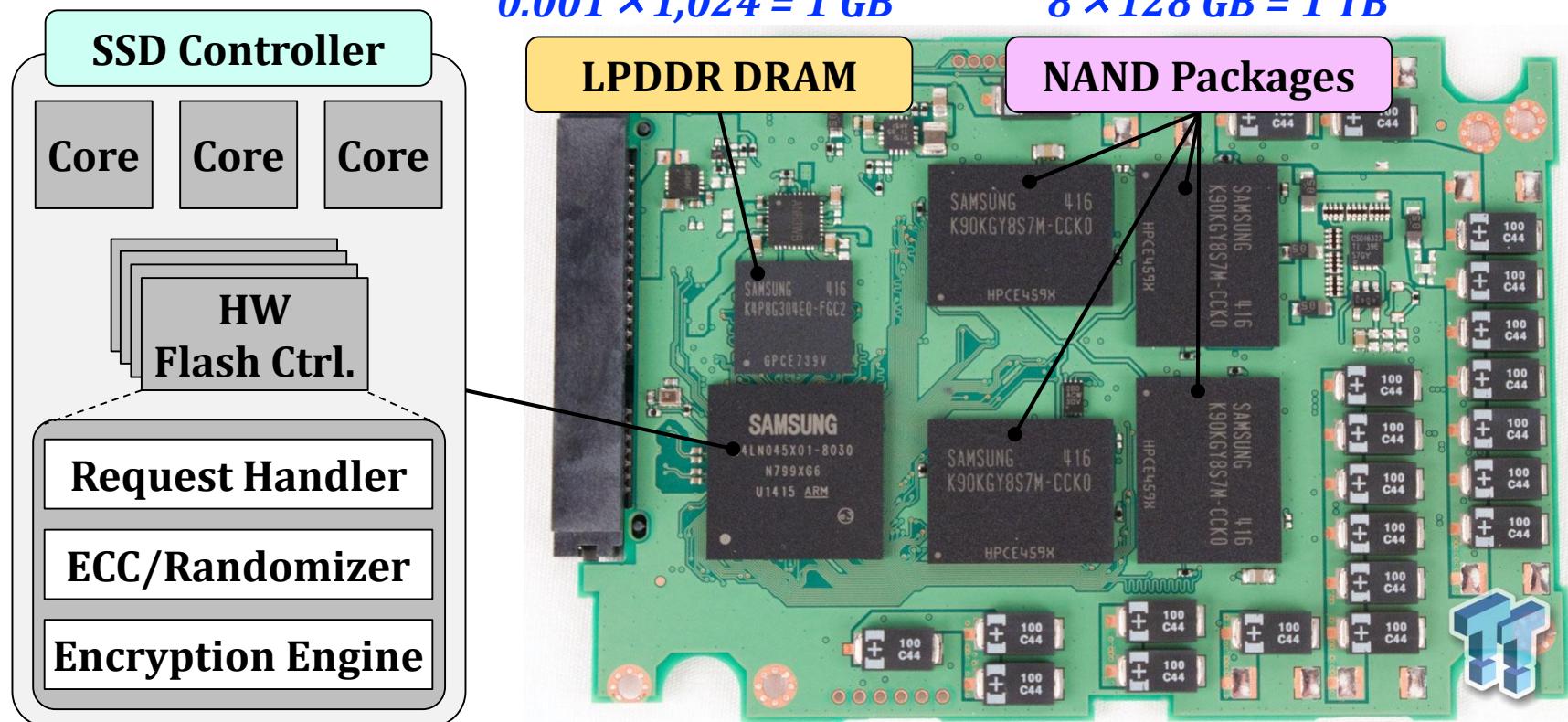
[‡]*IPM*

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

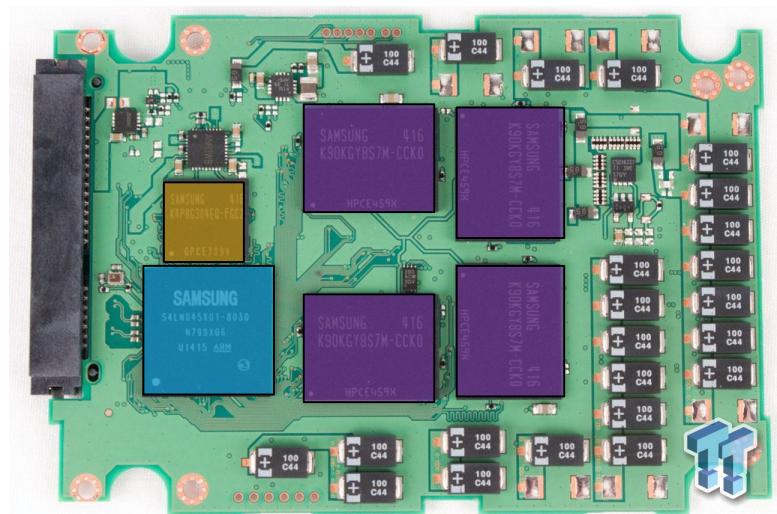
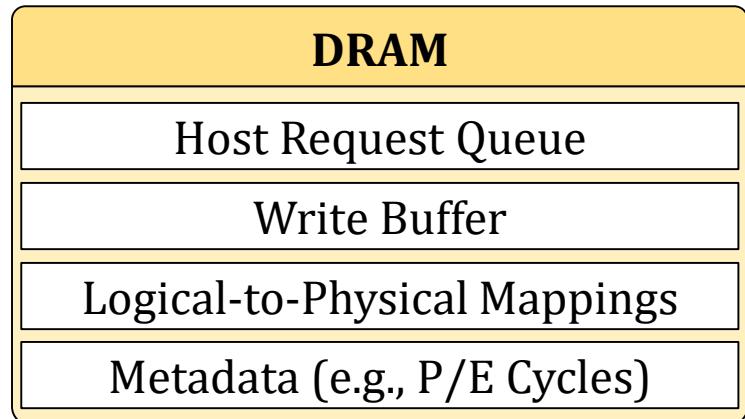
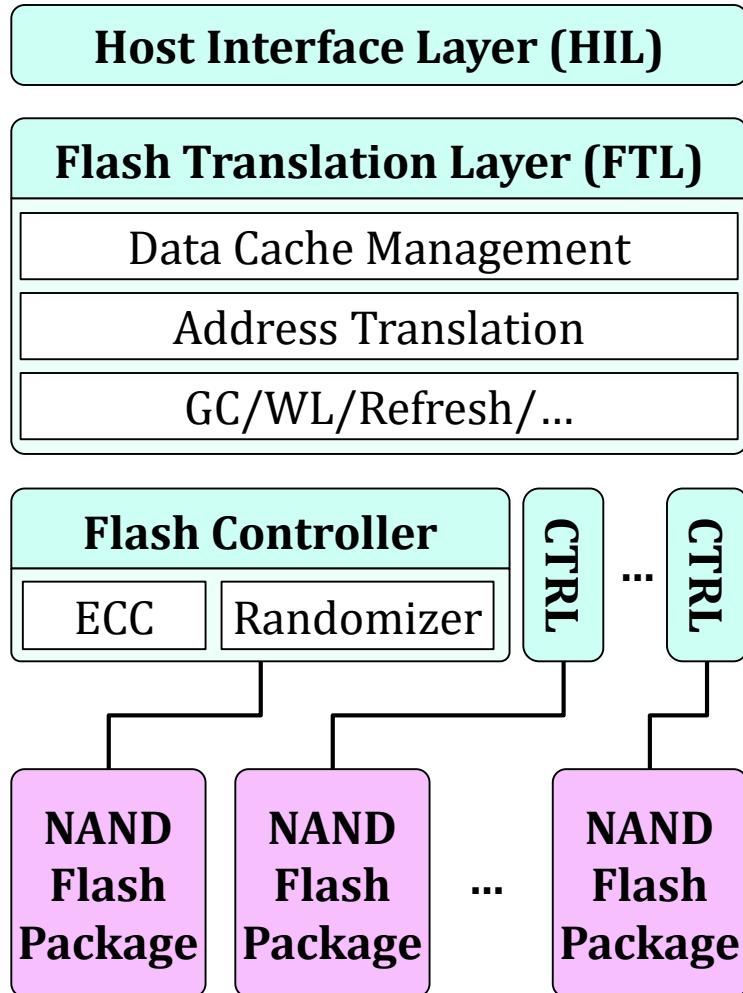
Modern SSD Architecture

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

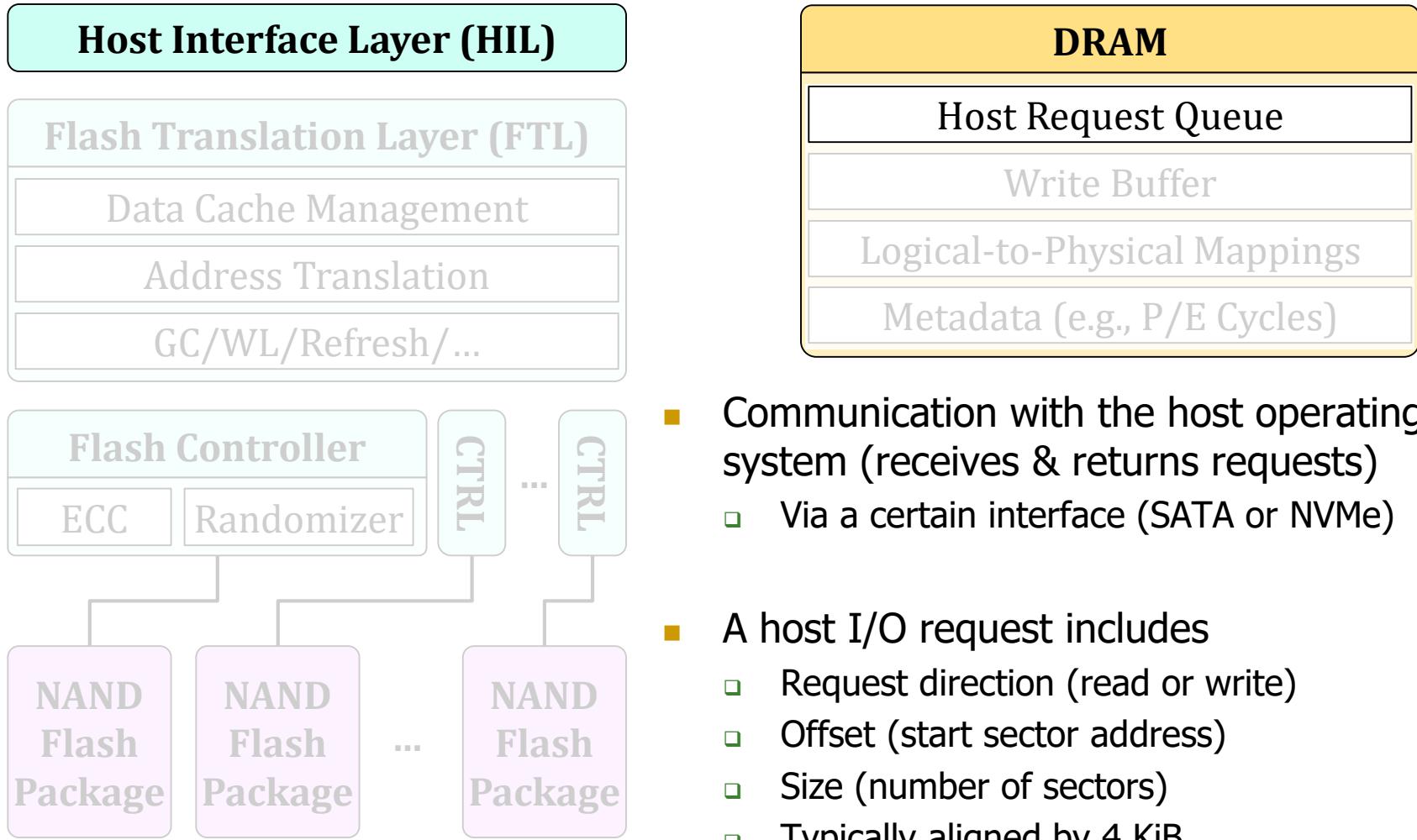


Samsung PM853T 960GB Enterprise SSD (from <https://www.tweaktown.com/reviews/6695/samsung-pm853t-960gb-enterprise-ssd-review/index.html>)

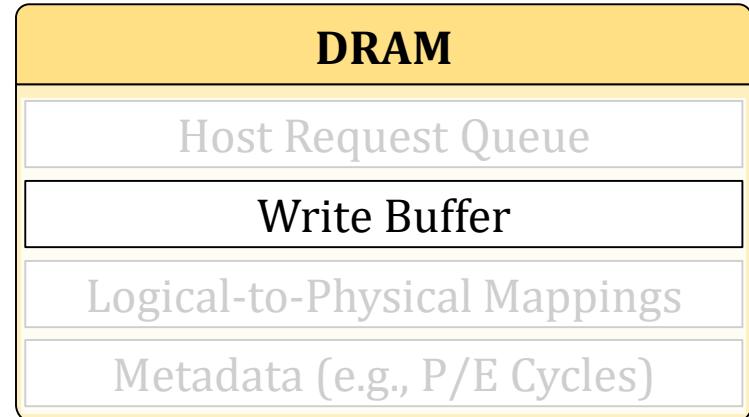
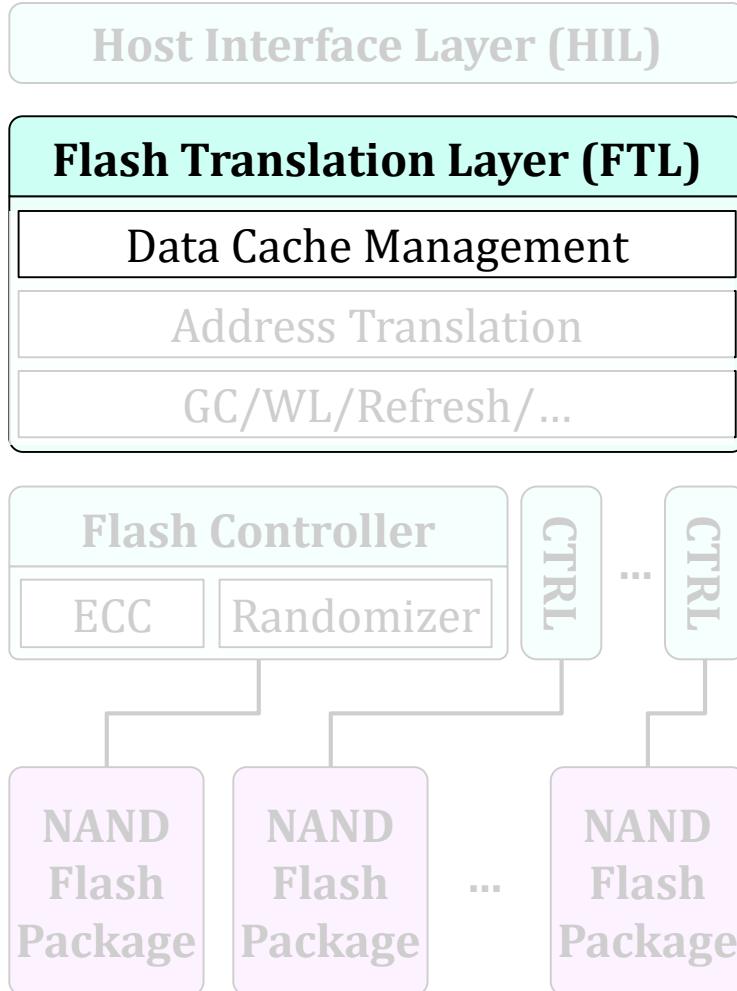
Another Overview



Request Handling: Write

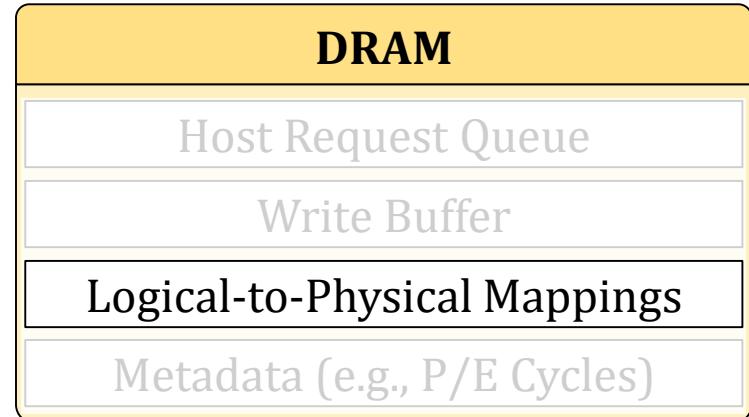
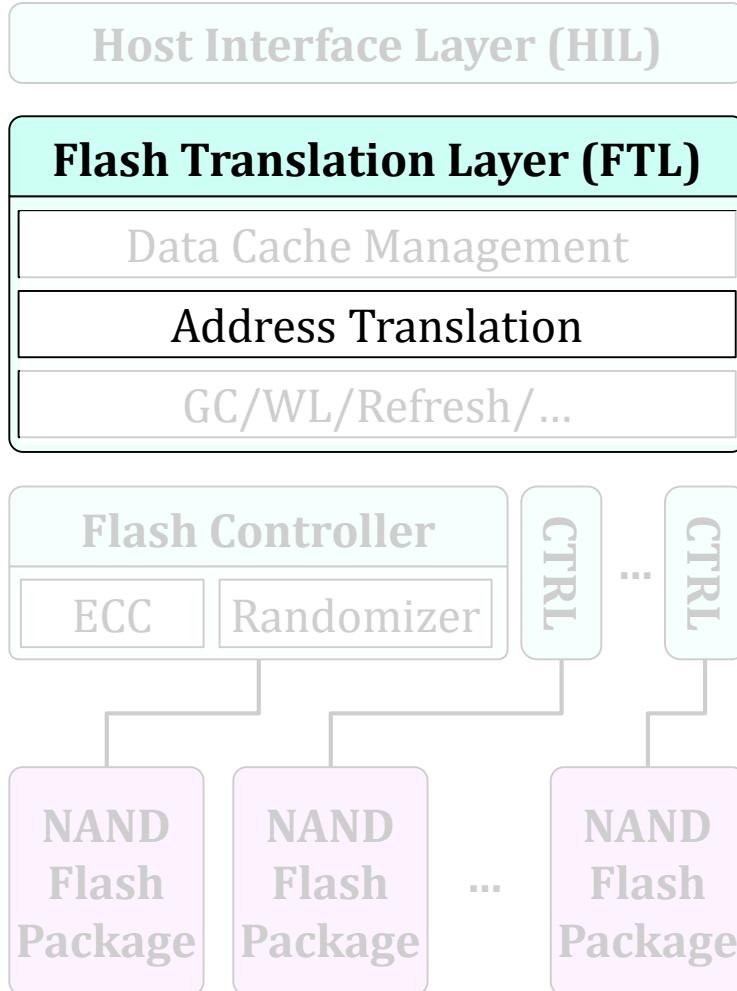


Request Handling: Write



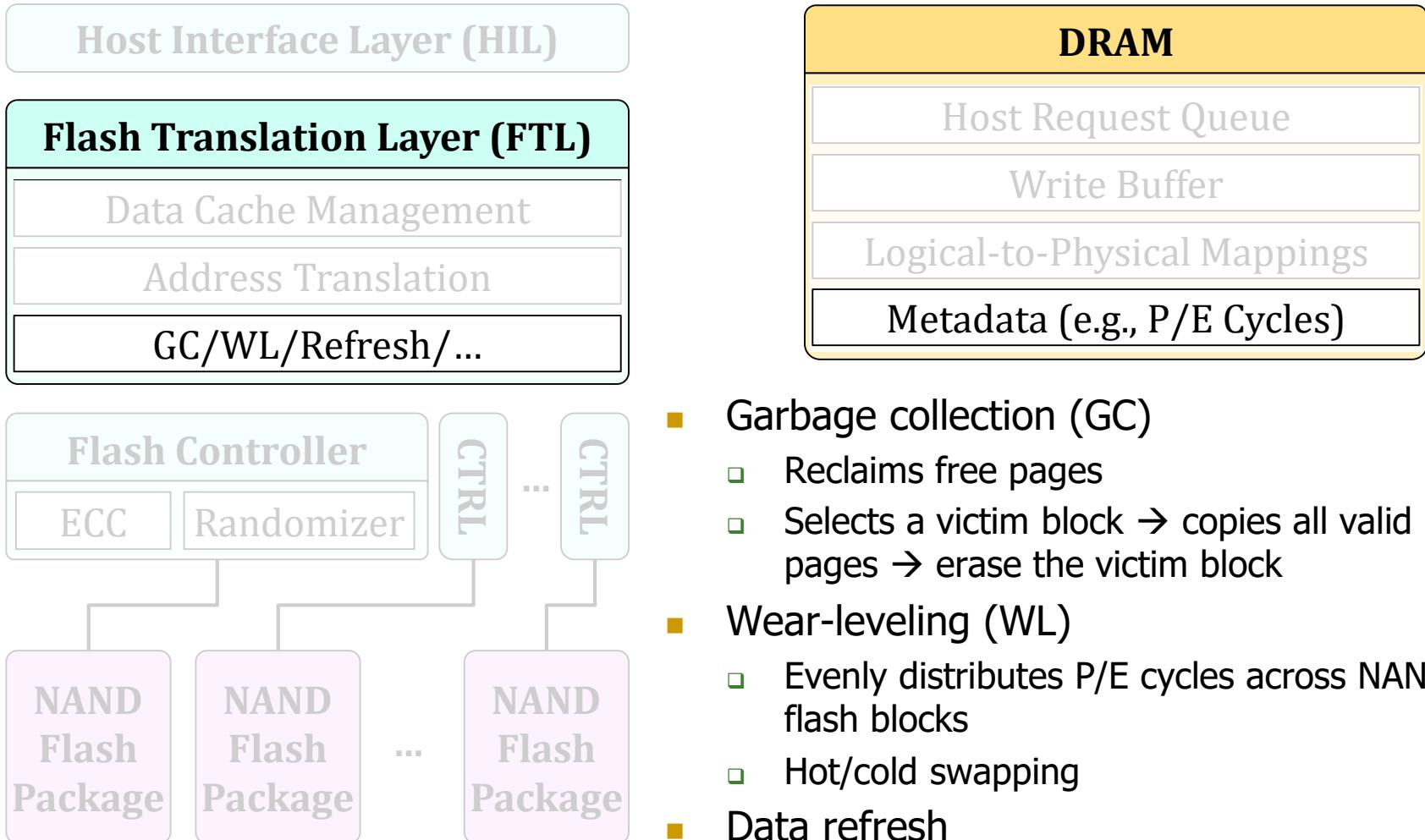
- Buffering data to write (read from NAND flash memory)
 - Essential to reducing write latency
 - Enables flexible I/O scheduling
 - Helpful for improving lifetime (not so likely)
- Limited size (e.g., tens of MBs)
 - Needs to ensure data integrity even under sudden power-off
 - Most DRAM capacity is used for L2P mappings

Request Handling: Write

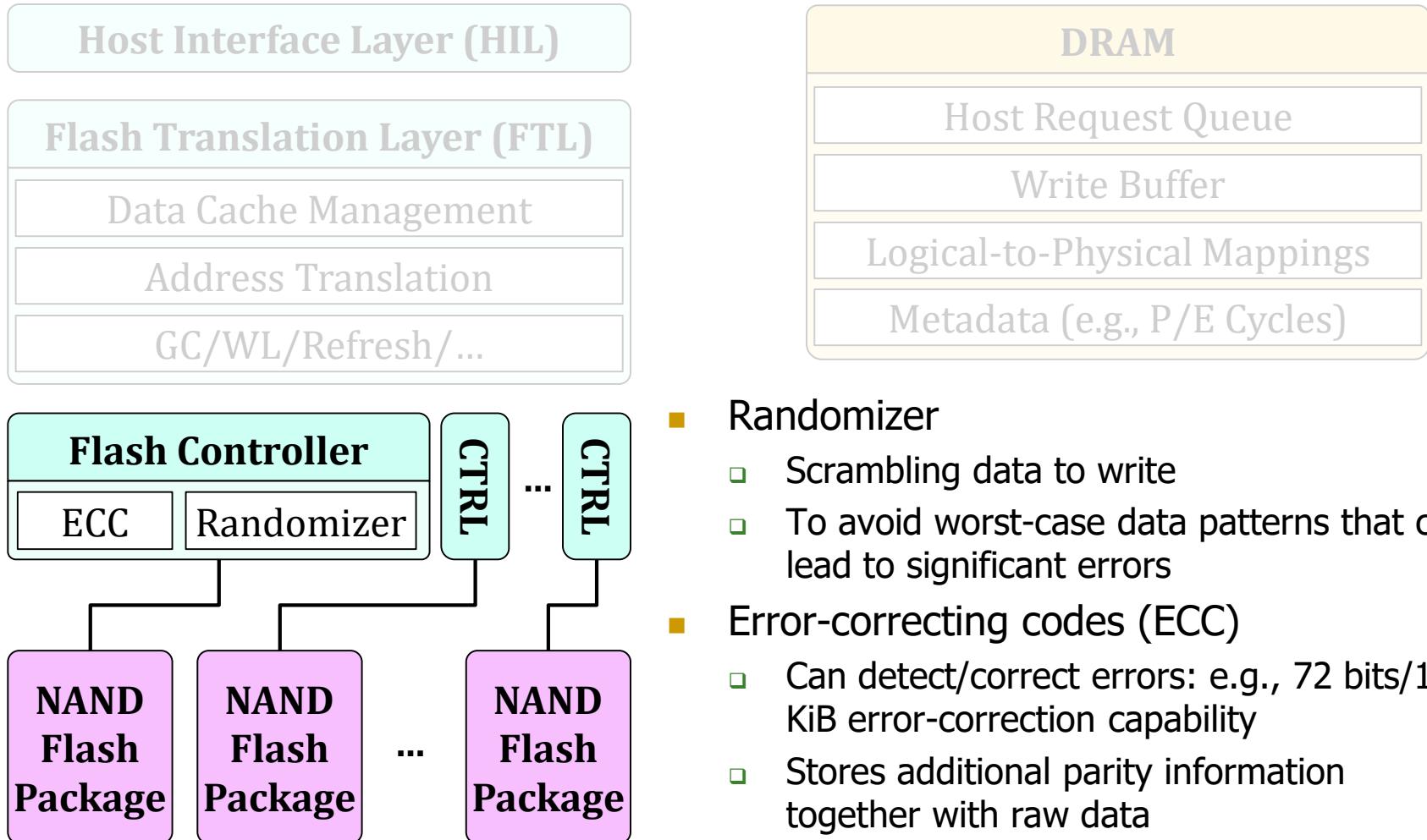


- Core functionality for out-of-place writes
 - To hide the erase-before-write property
- Needs to maintain L2P mappings
 - Logical Page Address (LPA) → Physical Page Address (PPA)
- Mapping granularity: 4 KiB
 - 4 Bytes for 4 KiB → 0.1% of SSD capacity

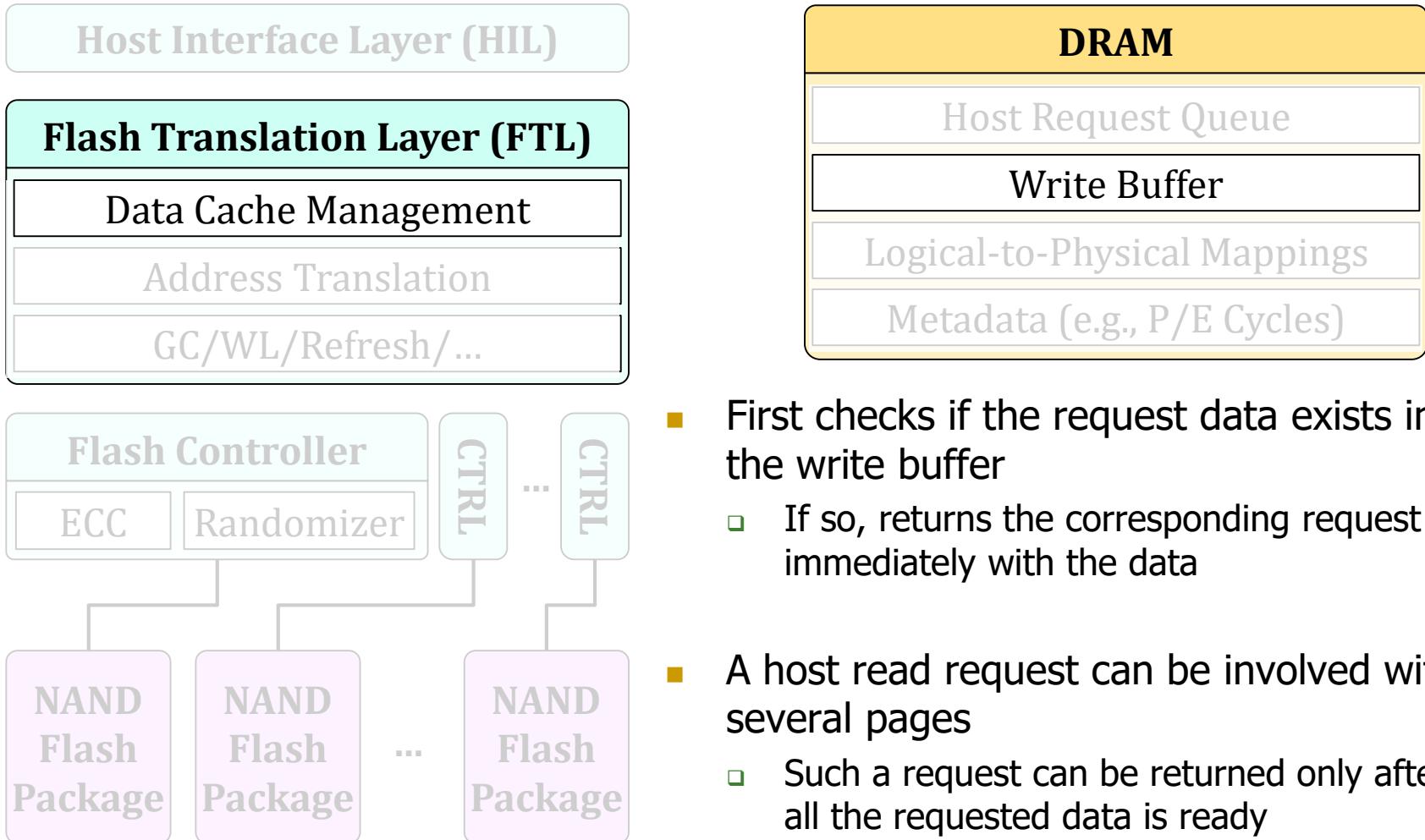
Request Handling: Write



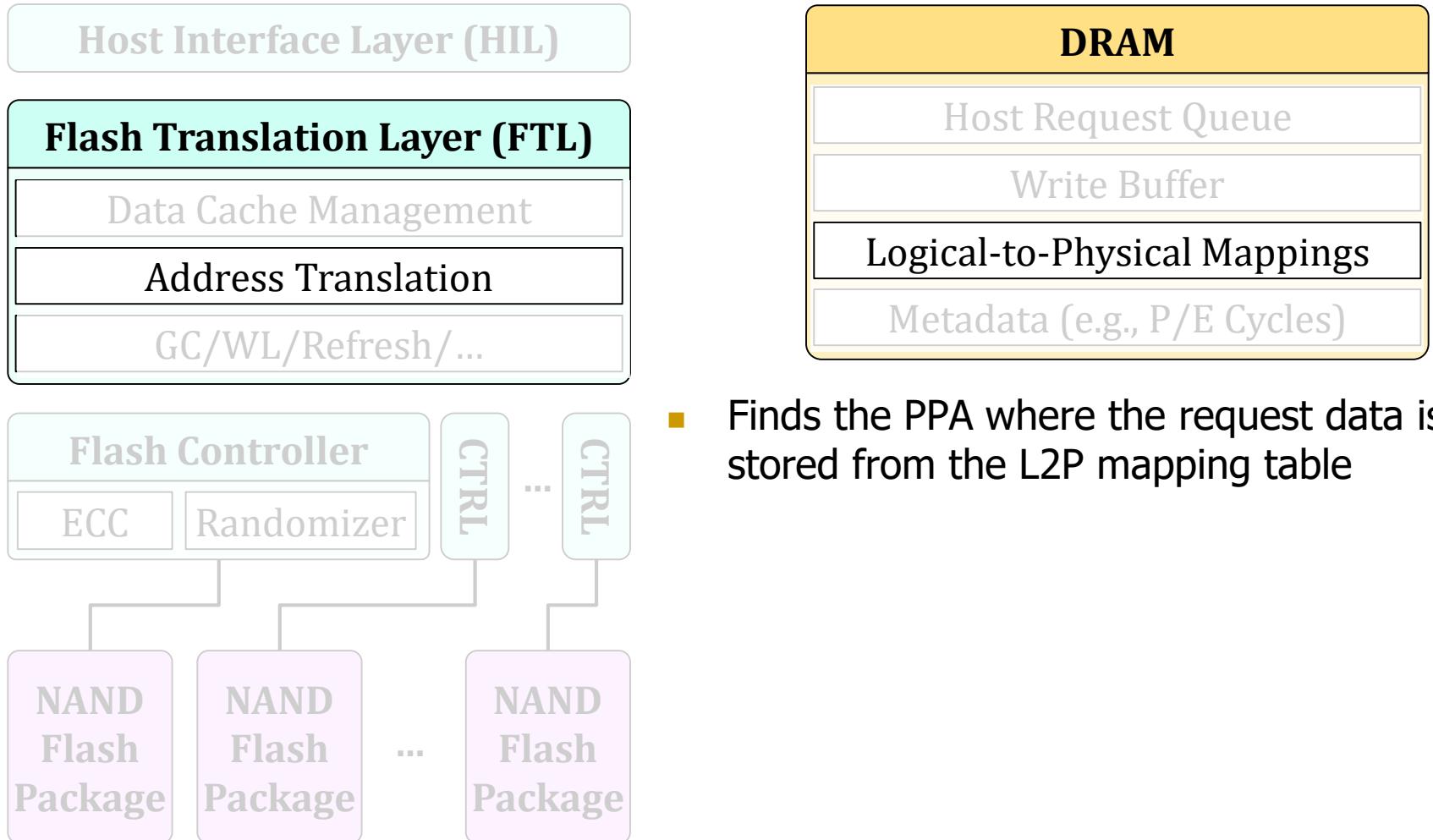
Request Handling: Write



Request Handling: Read

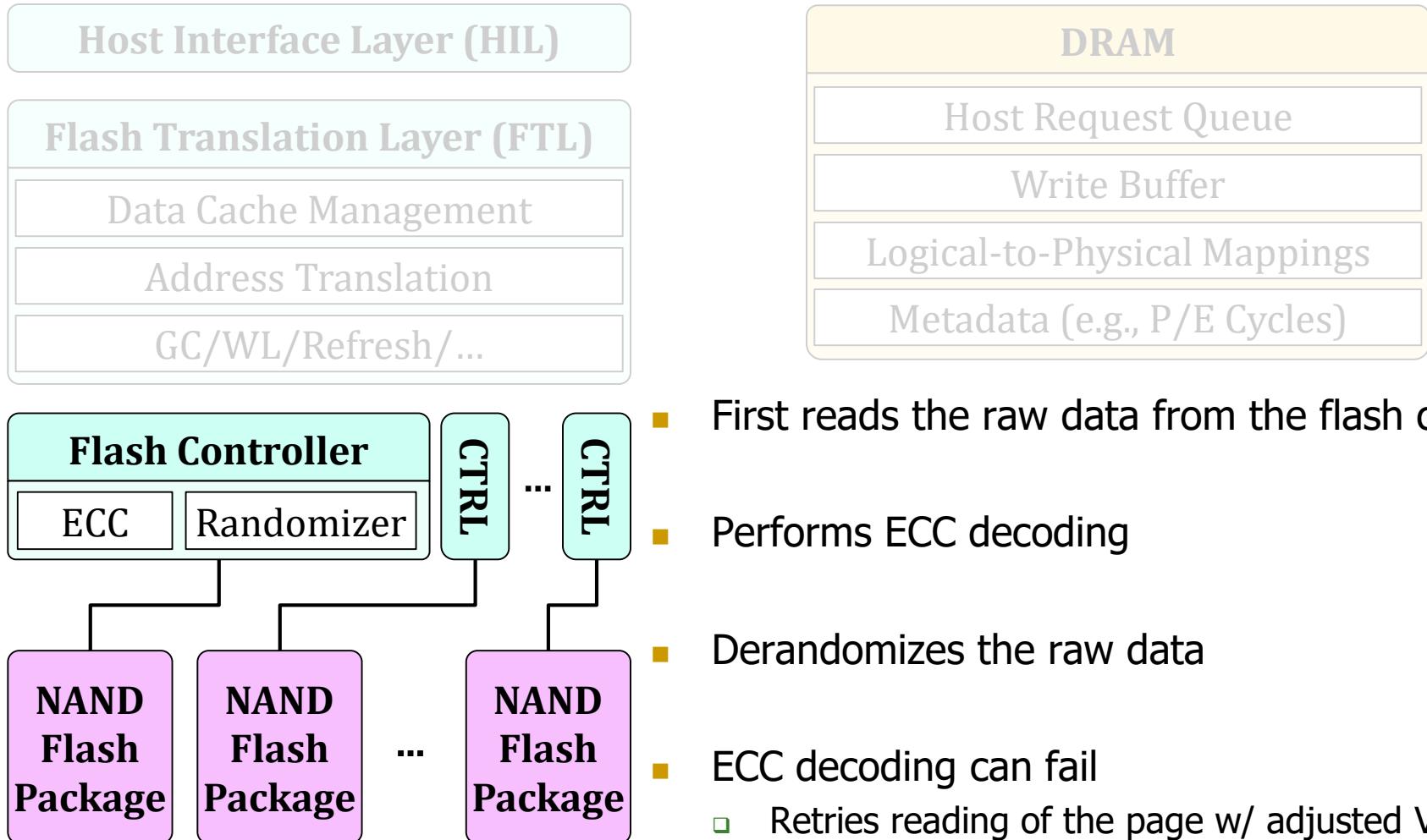


Request Handling: Read



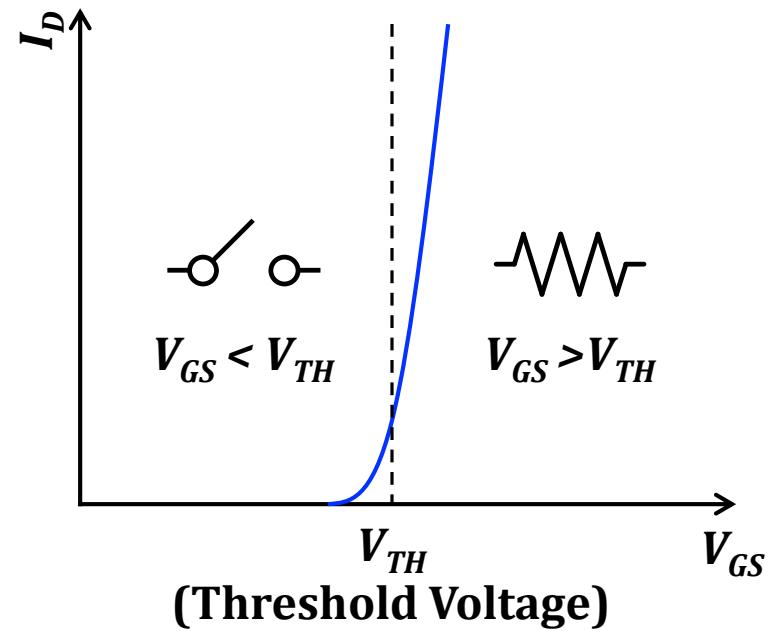
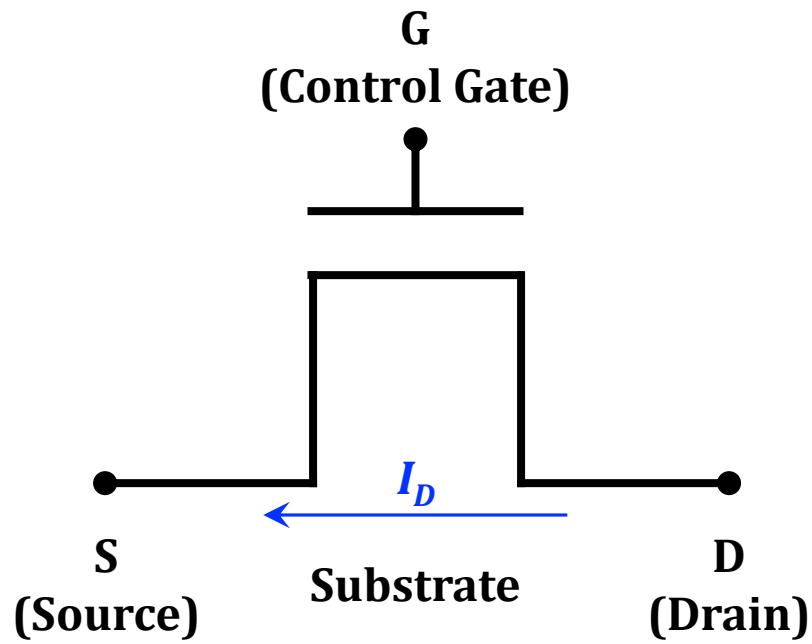
- Finds the PPA where the request data is stored from the L2P mapping table

Request Handling: Read



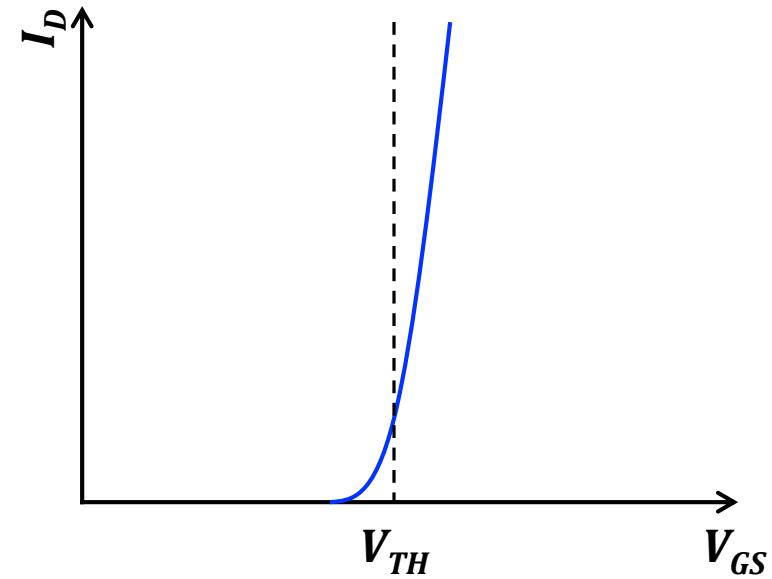
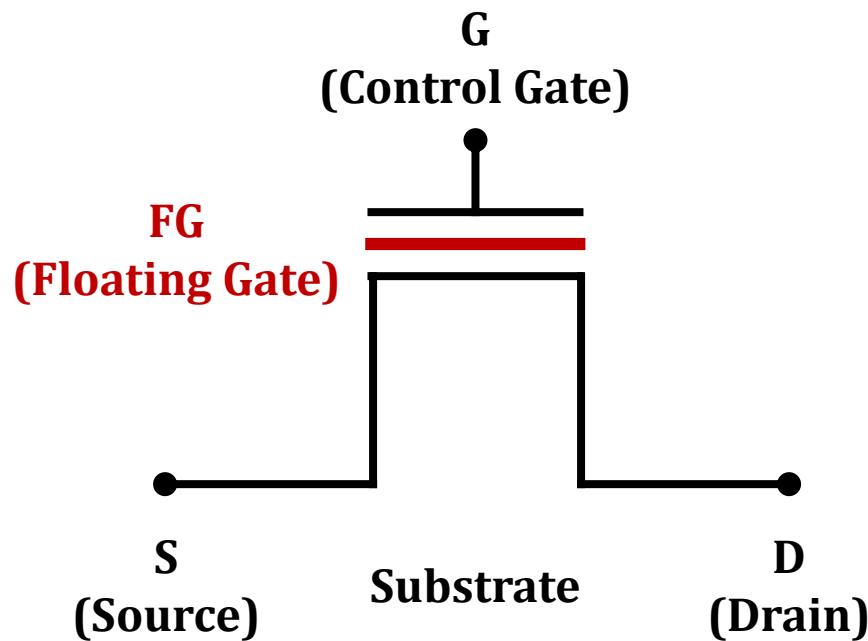
A Flash Cell

- Basically, it is a transistor



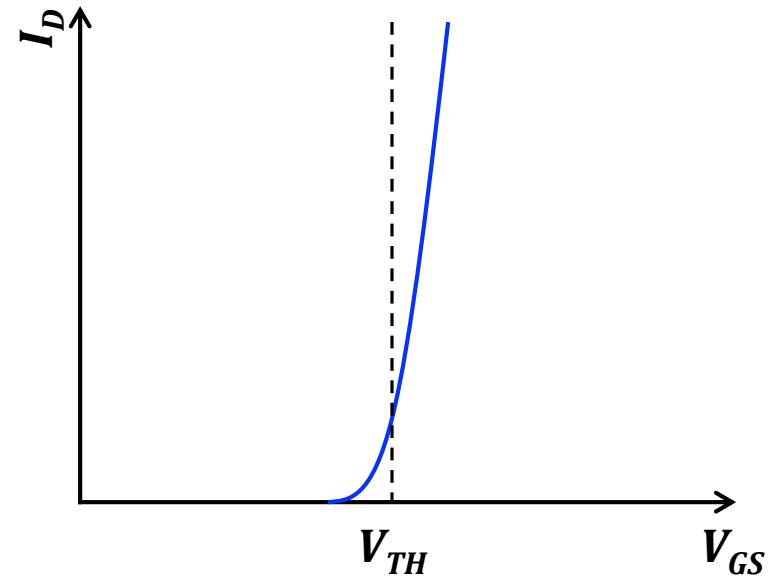
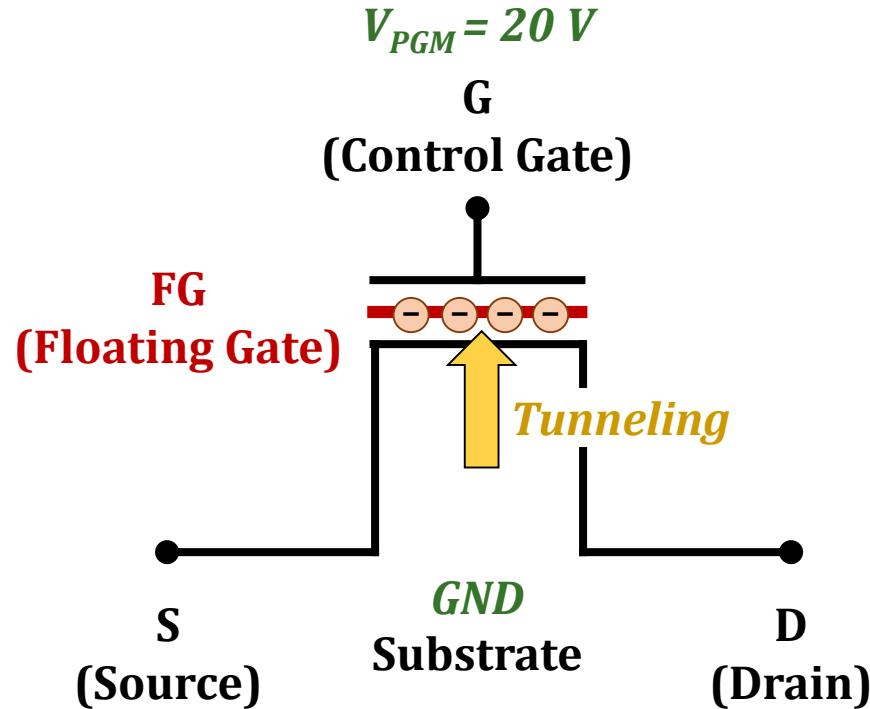
A Flash Cell

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



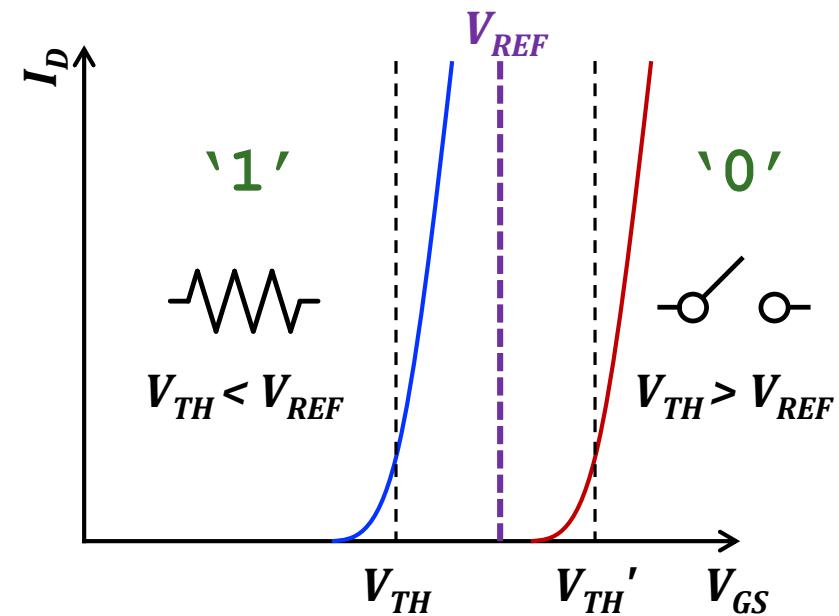
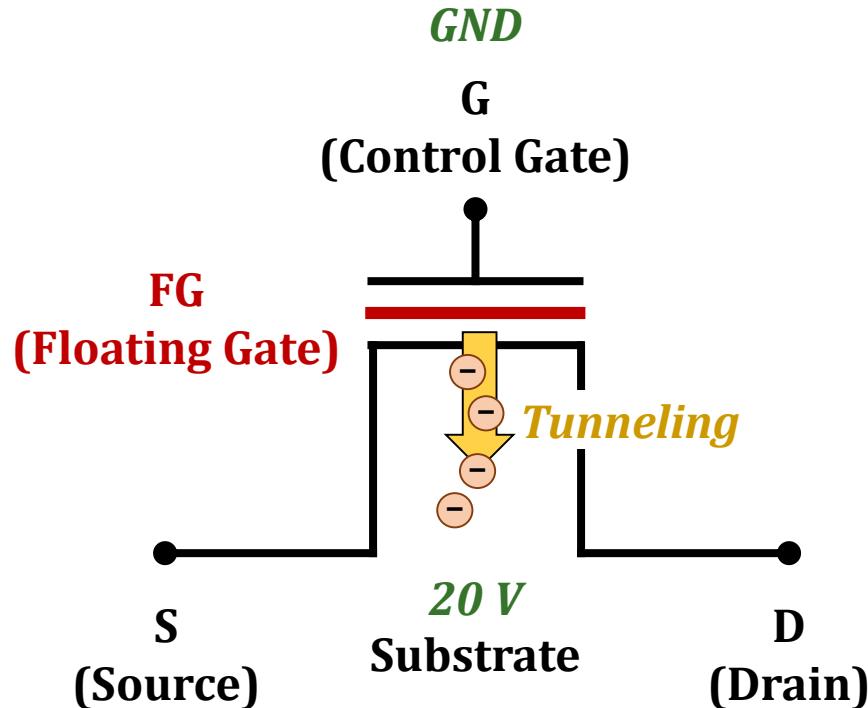
A Flash Cell

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



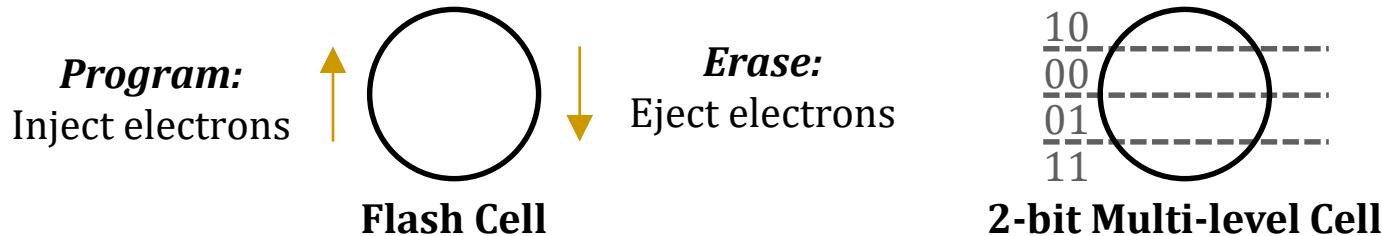
A Flash Cell

- 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})

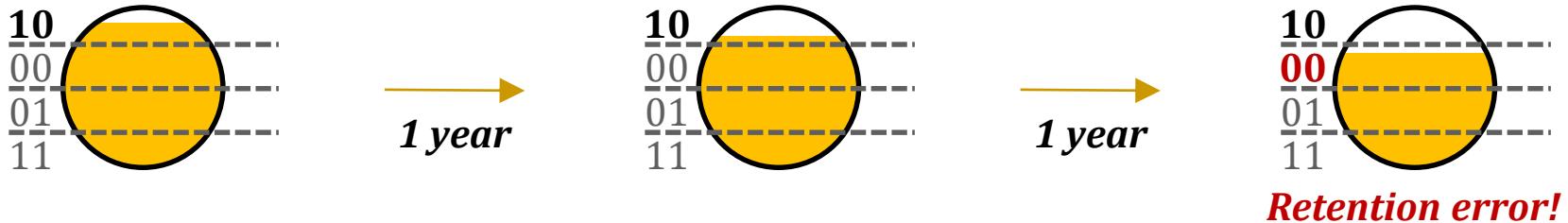


Flash Cell Characteristics

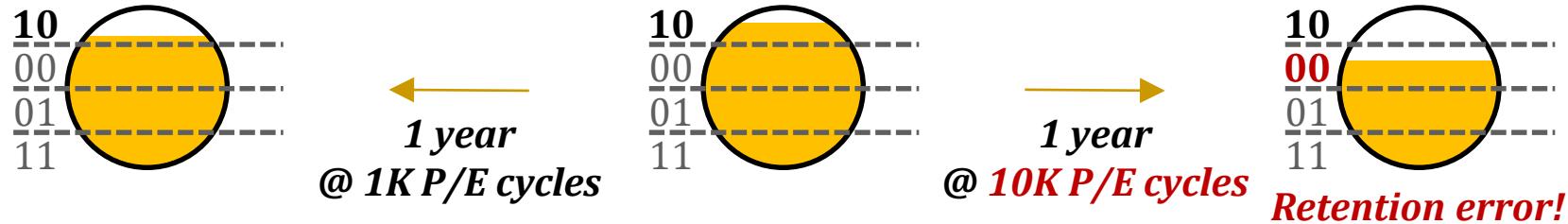
- Multi-leveling: A flash cell can store multiple bits



- Retention loss: A cell leaks electrons over time

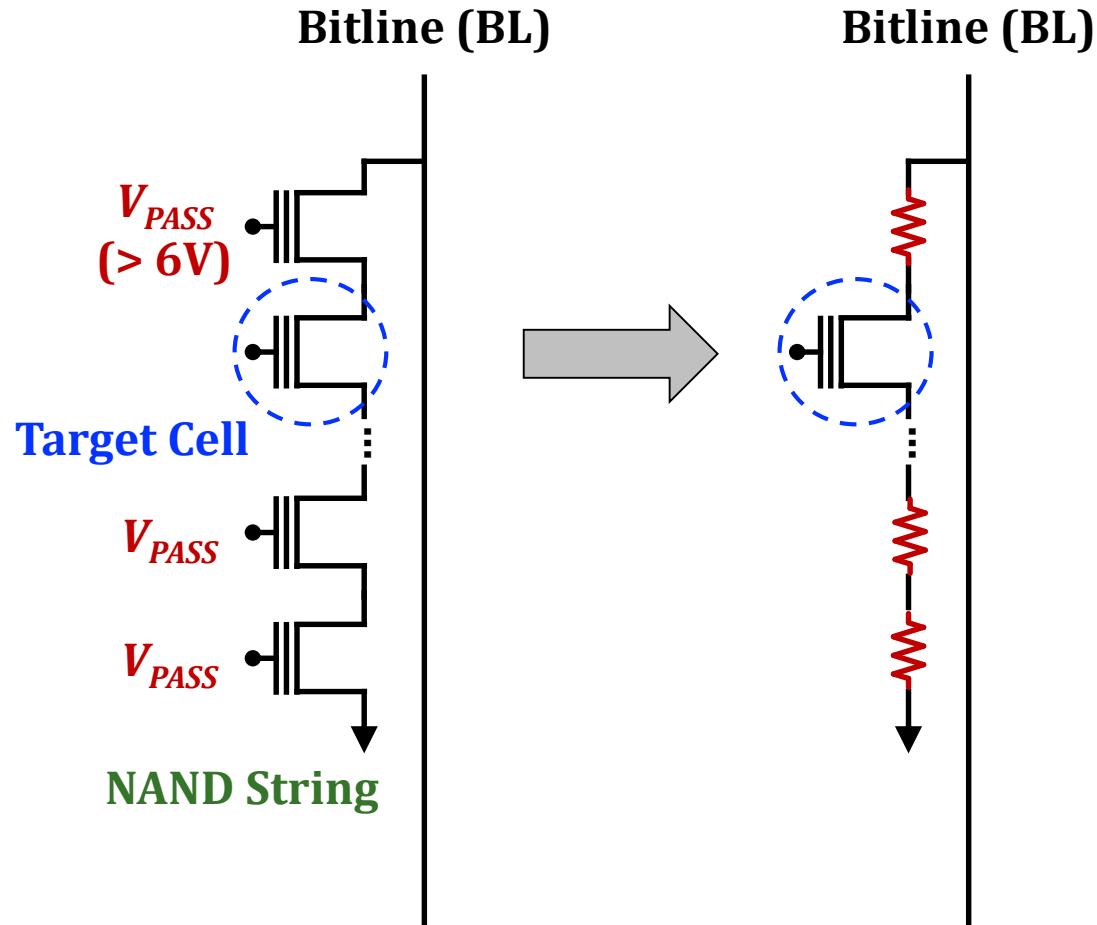


- Limited lifetime: A cell wears out after P/E cycling



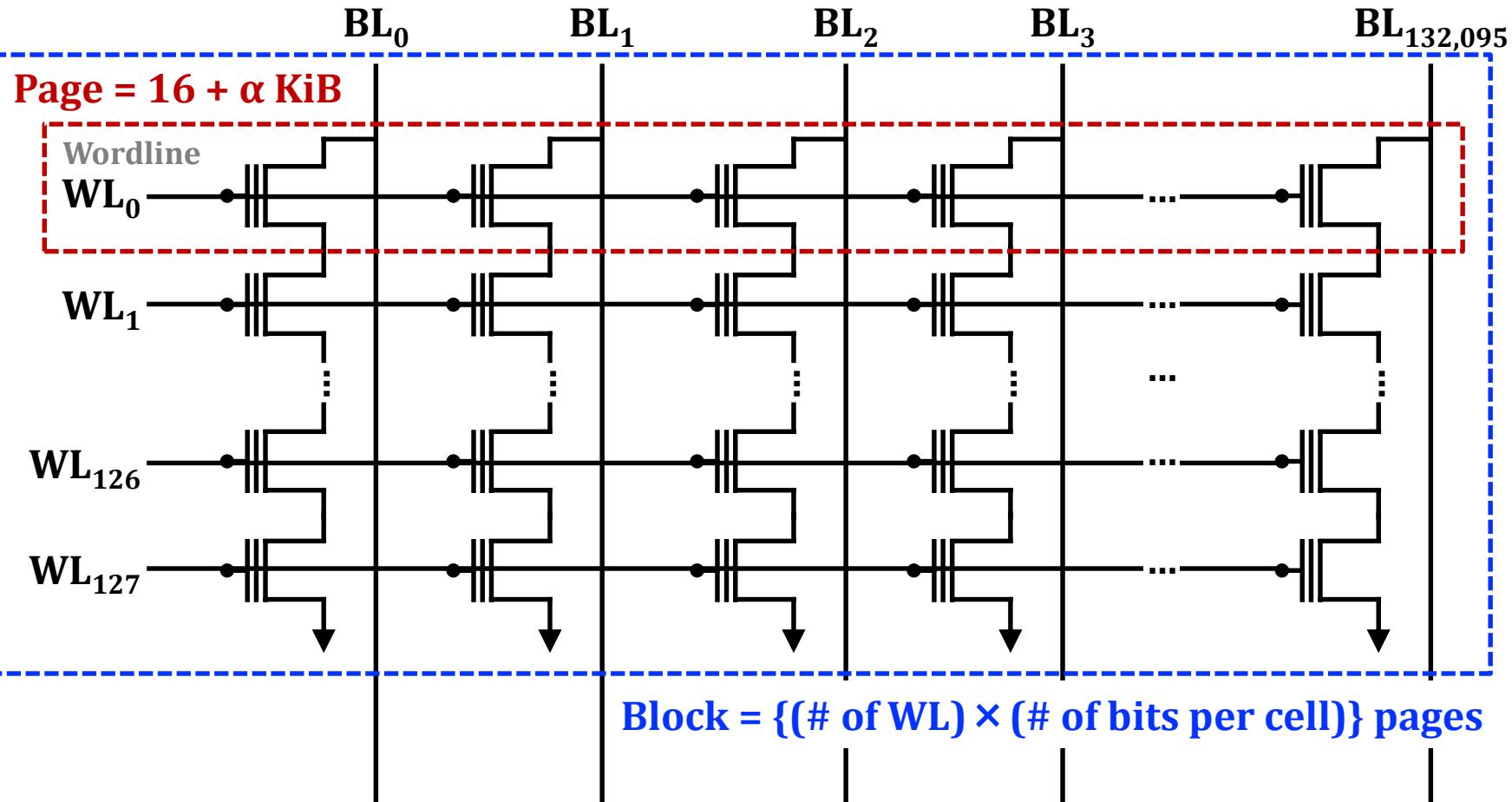
A NAND String

- Multiple (e.g., 128) flash cells are serially connected



Pages and Blocks

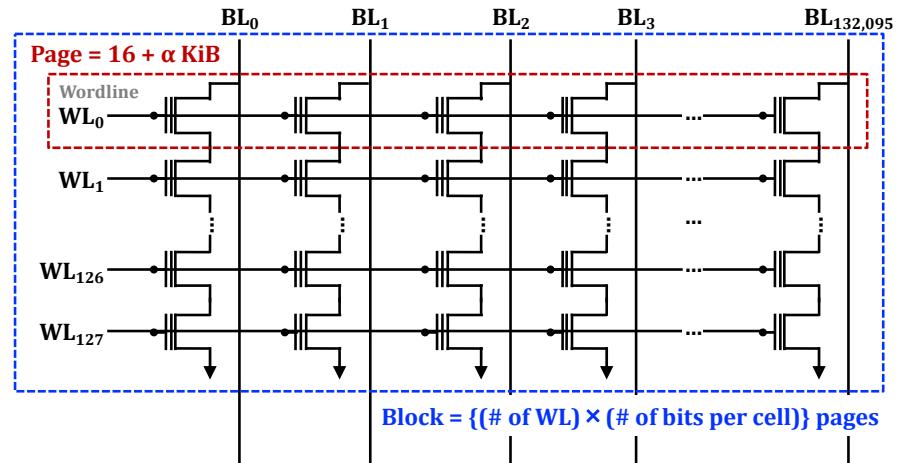
- A large number ($> 100,000$) of cells operate concurrently



Pages and Blocks (Continued)

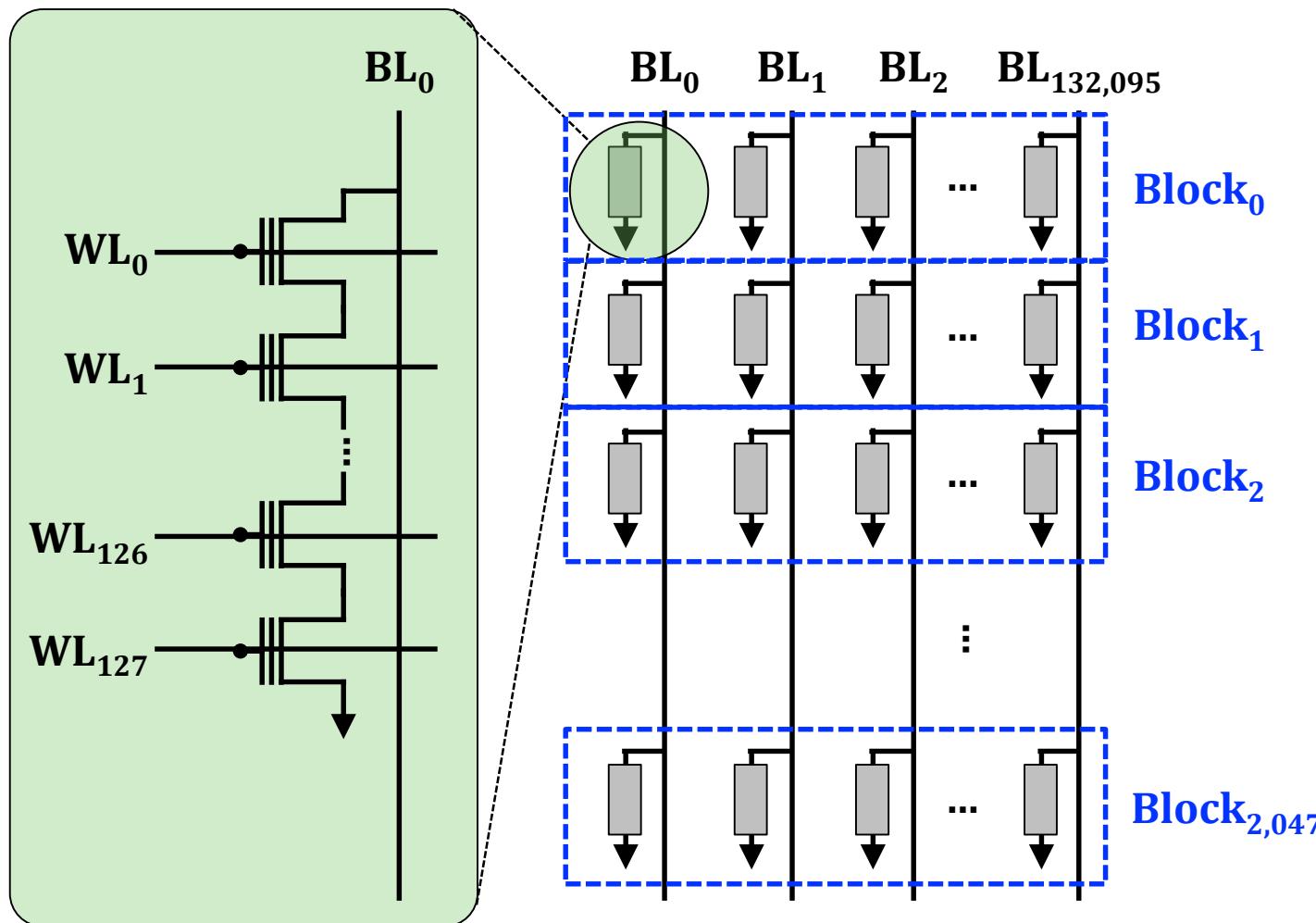
- Program and erase: Unidirectional
 - Programming a cell → Increasing the cell's V_{TH}
 - Erasing a cell → Decreasing the cell's V_{TH}
- Programming a page cannot change '0' cells to '1' cells
→ **Erase-before-write property**

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



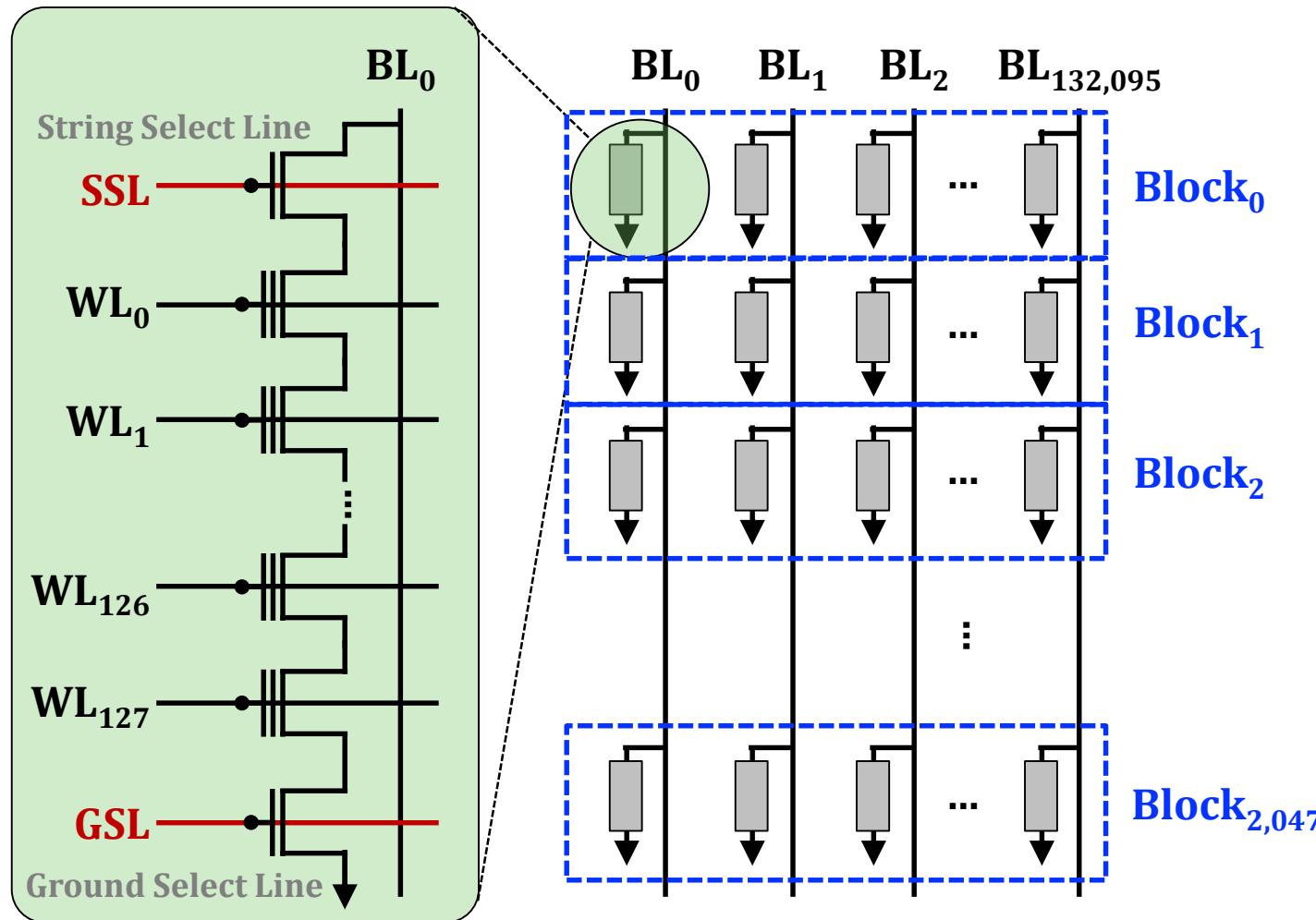
Planes

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



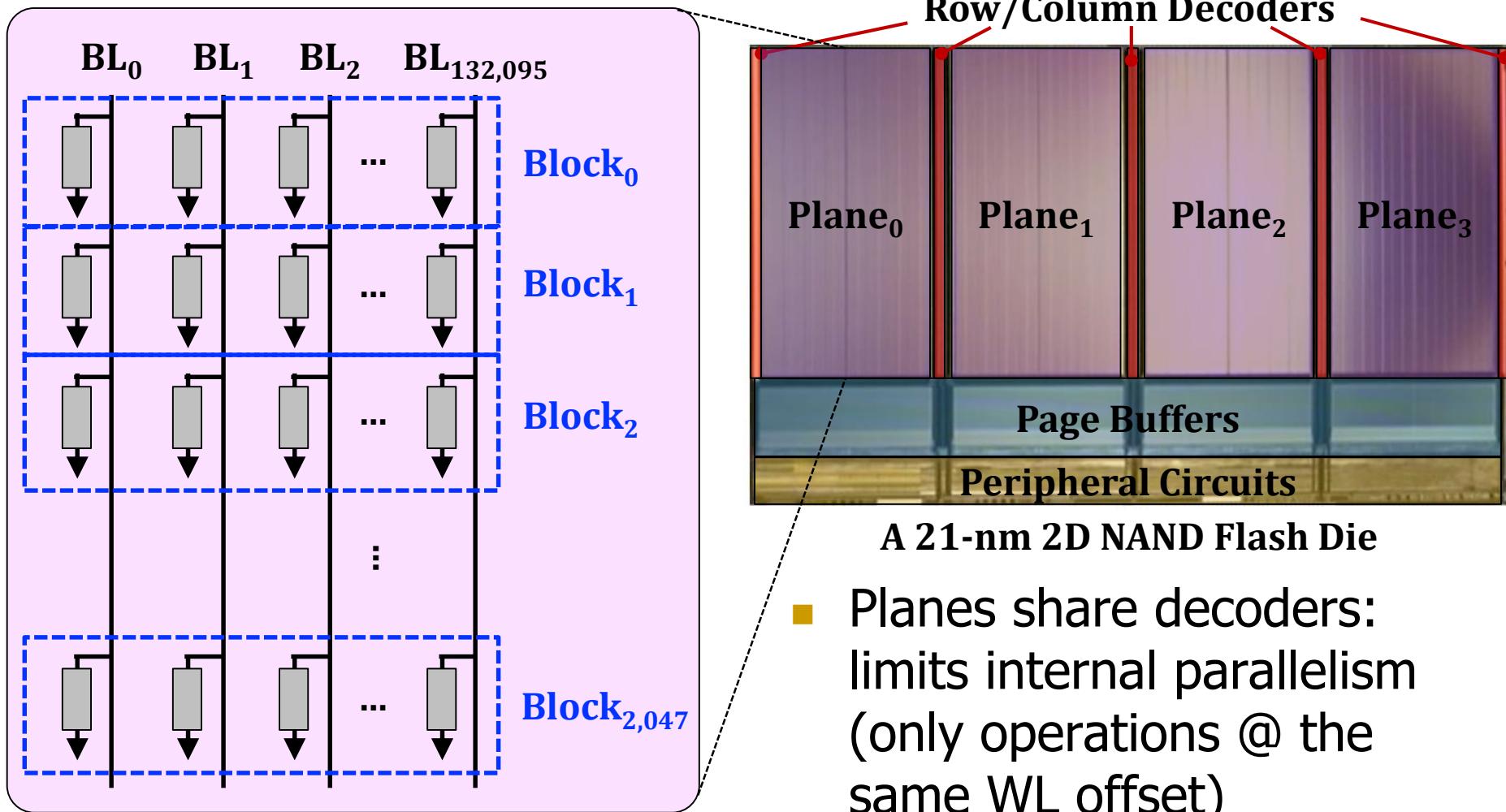
Planes

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



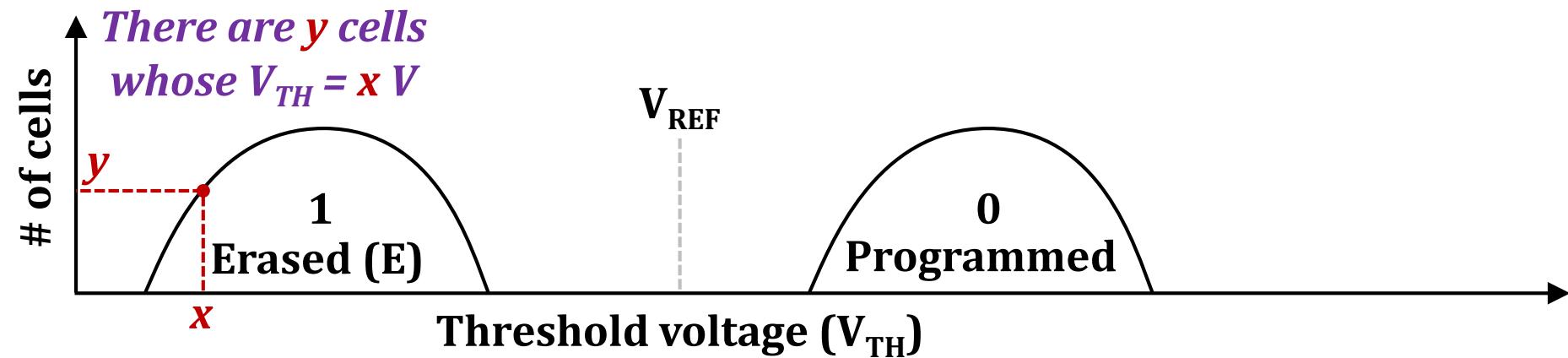
Planes and Dies

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



Threshold Voltage Distribution

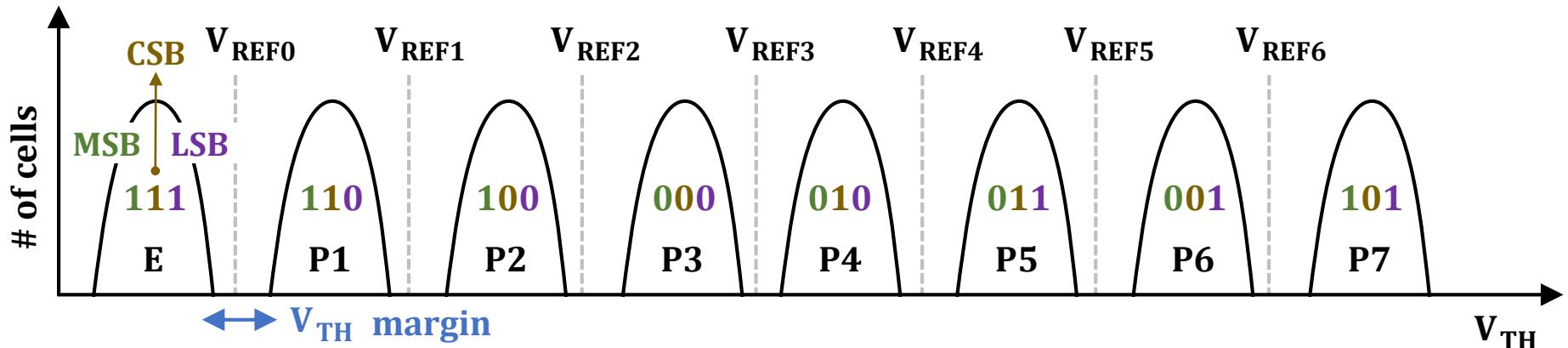
- V_{TH} distribution of cells in a programmed page/block/chip



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

V_{TH} Distribution of MLC NAND Flash

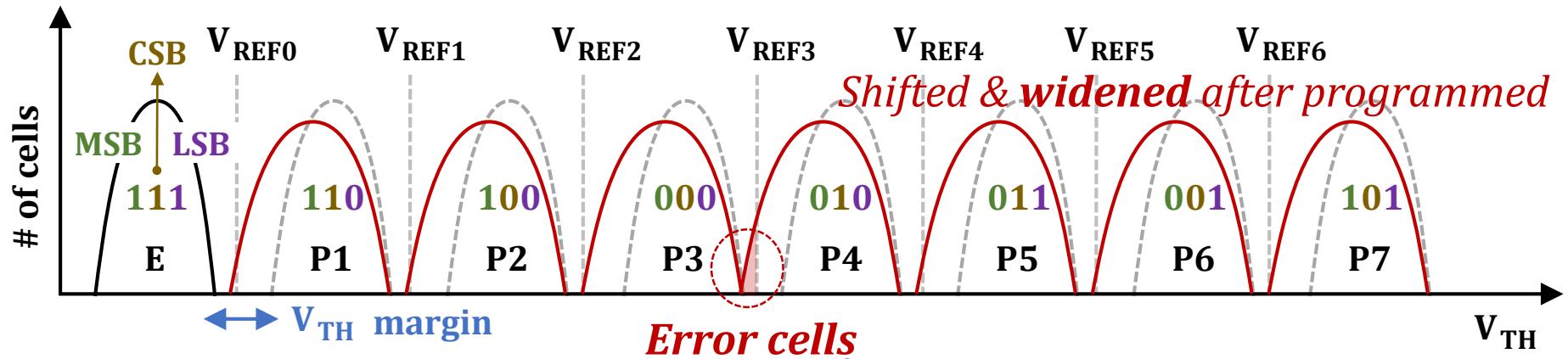
- Multi-level cell (MLC) technique
 - $2^m V_{TH}$ states required to store m bits in a single flash cell



- Limited width of the V_{TH} window: Need to
 - Make each V_{TH} state narrow
 - Guarantee sufficient margins b/w adjacent V_{TH} states

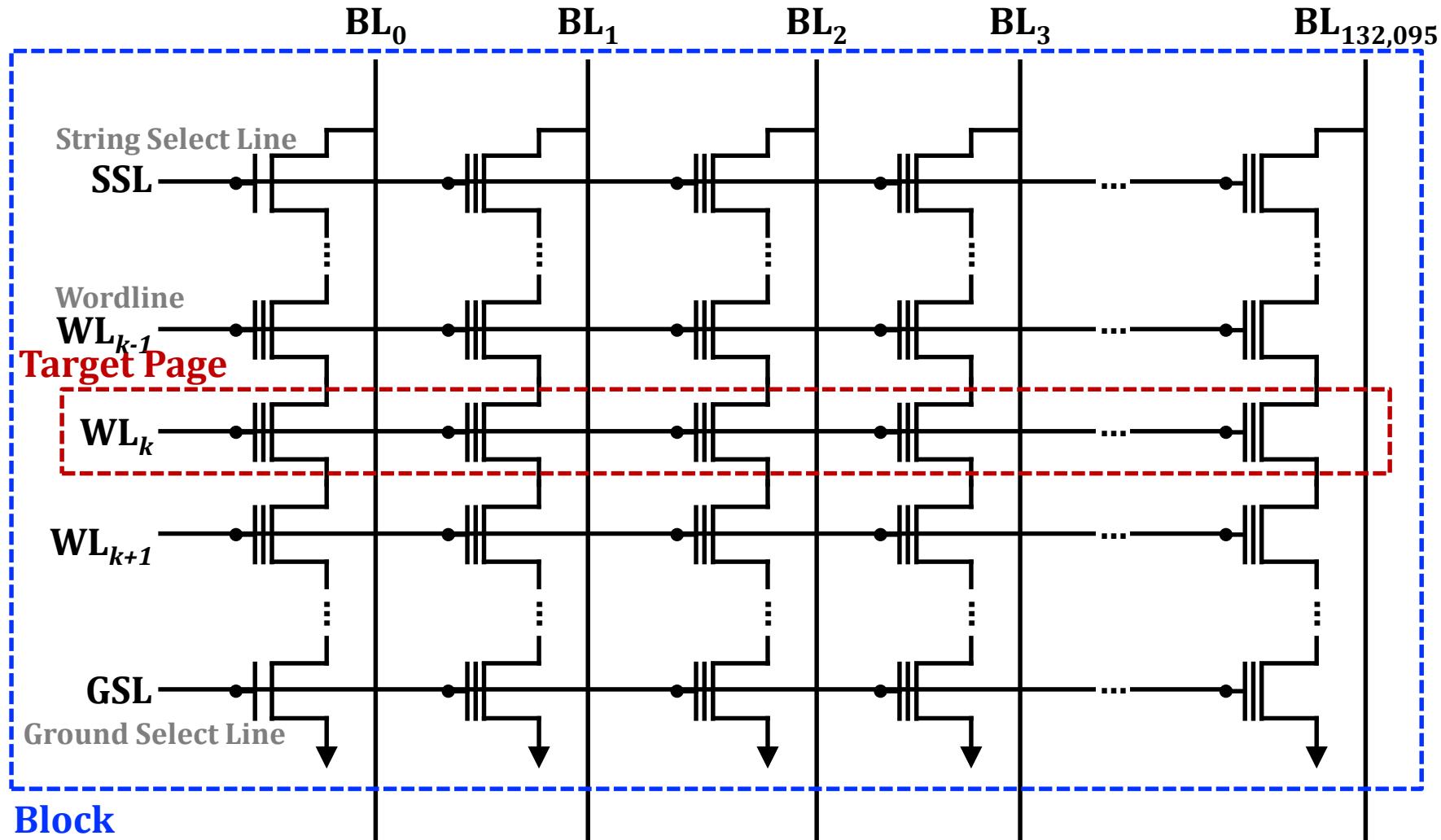
V_{TH} Distribution of MLC NAND Flash

- Multi-level cell (MLC) technique
 - $2^m V_{TH}$ states required to store m bits in a single flash cell



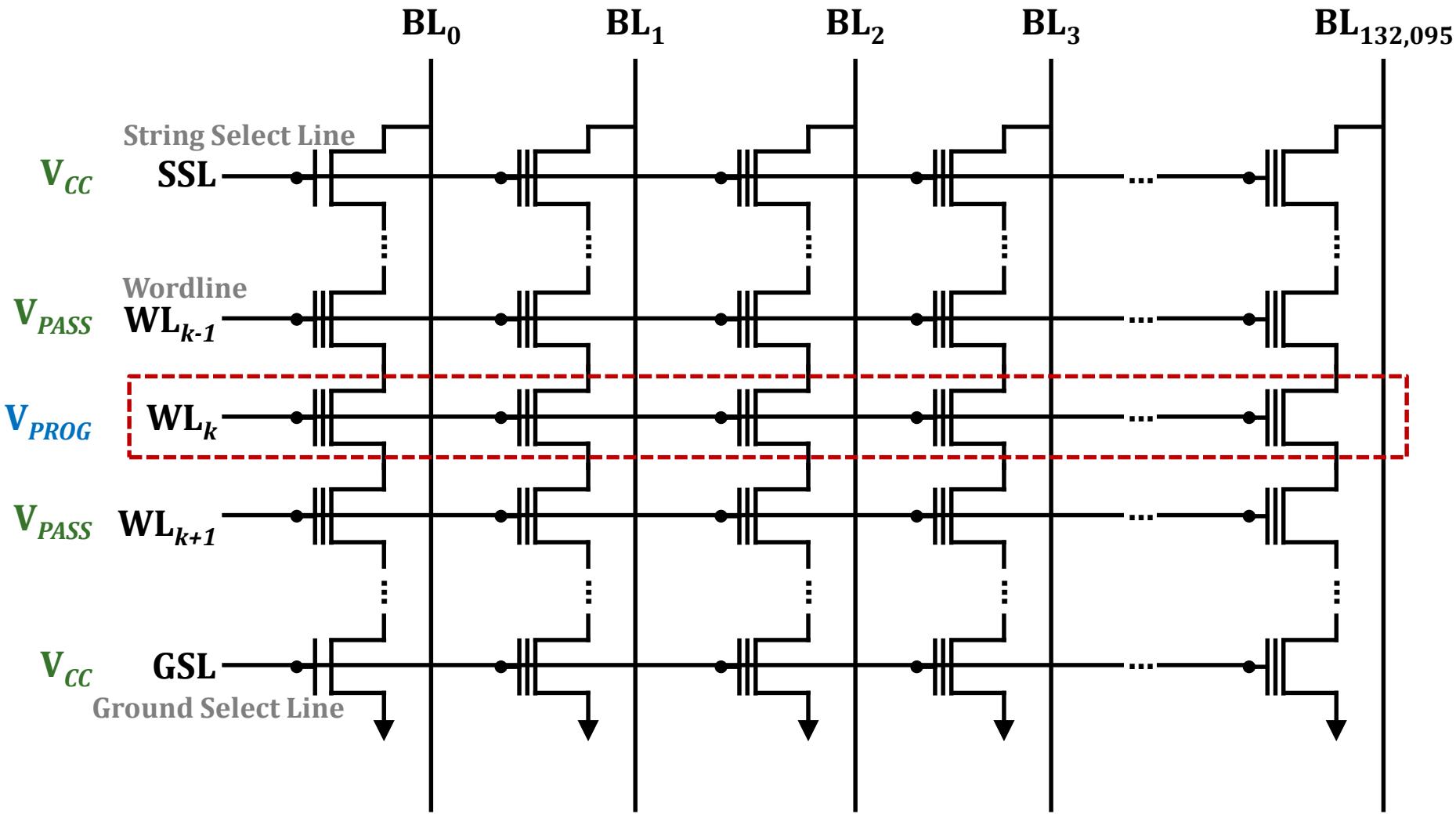
- Limited width of the V_{TH} window: Need to
 - Make each V_{TH} state narrow
 - Guarantee sufficient margins b/w adjacent V_{TH} states
 - V_{TH} changes over time after programmed
 - Narrower margins → Lower reliability
 - More bits per cell → higher density but lower reliability

Basic Operation: Page Program



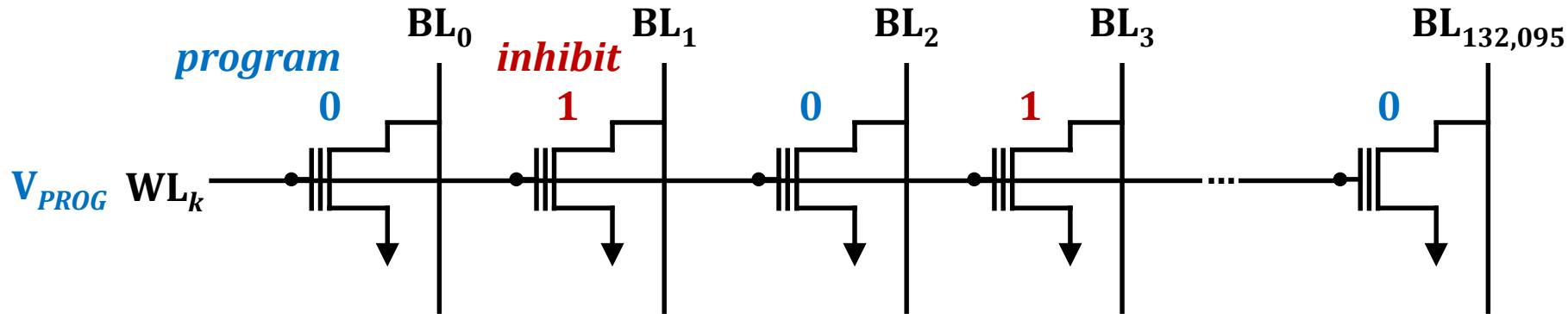
Basic Operation: Page Program

- WL control – All other cells operate as a resistance



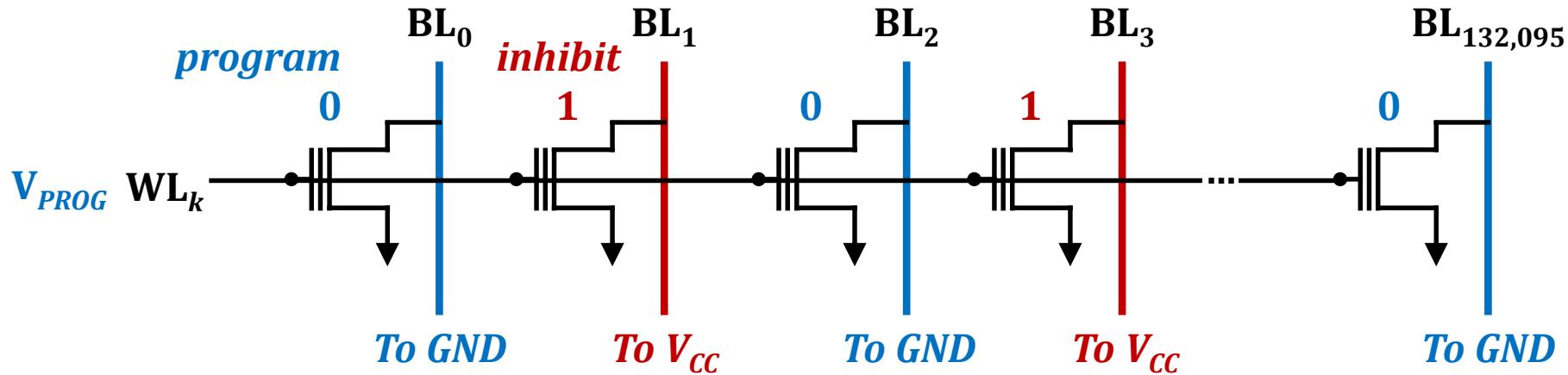
Basic Operation: Page Program

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

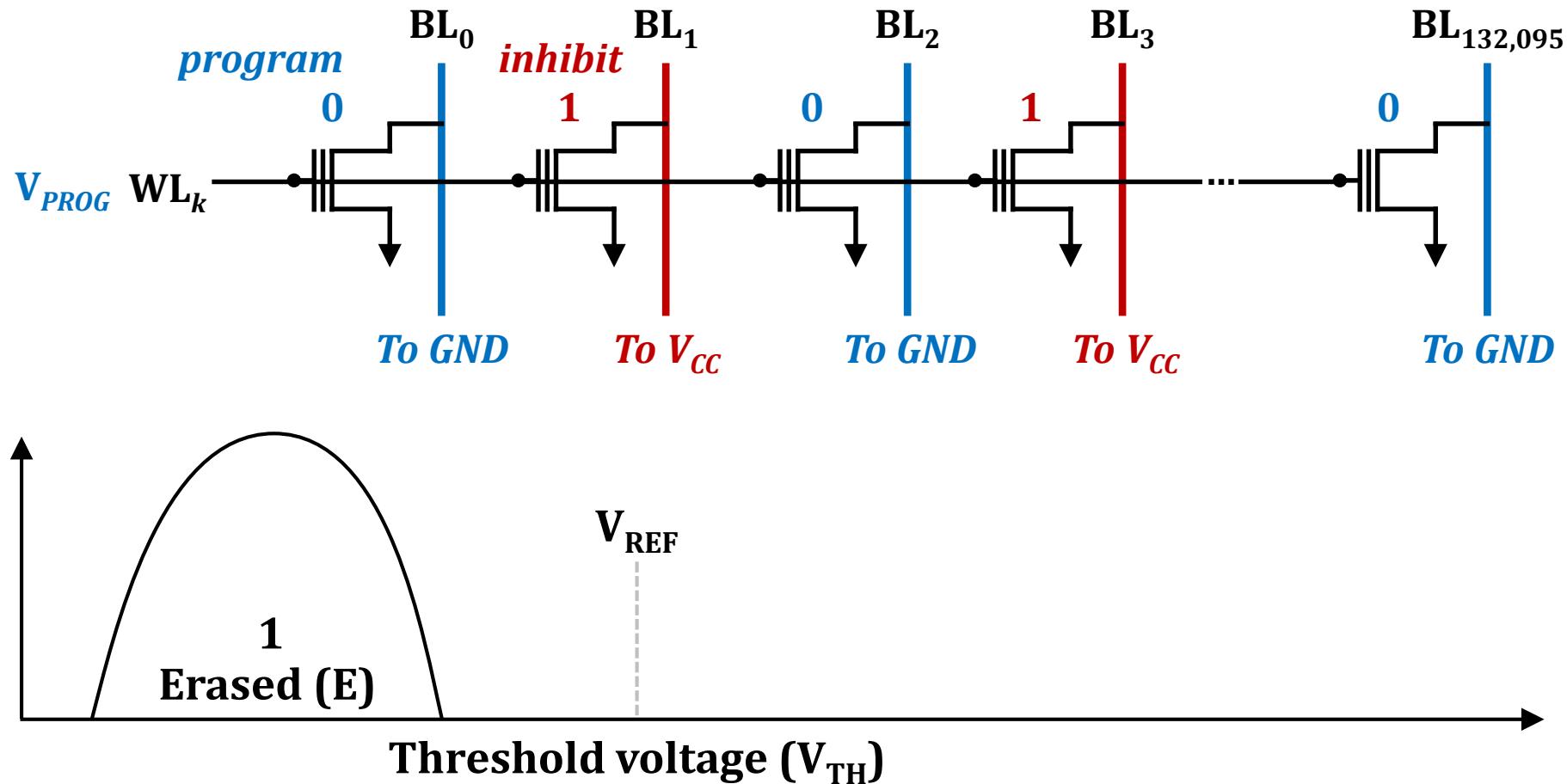


Basic Operation: Page Program

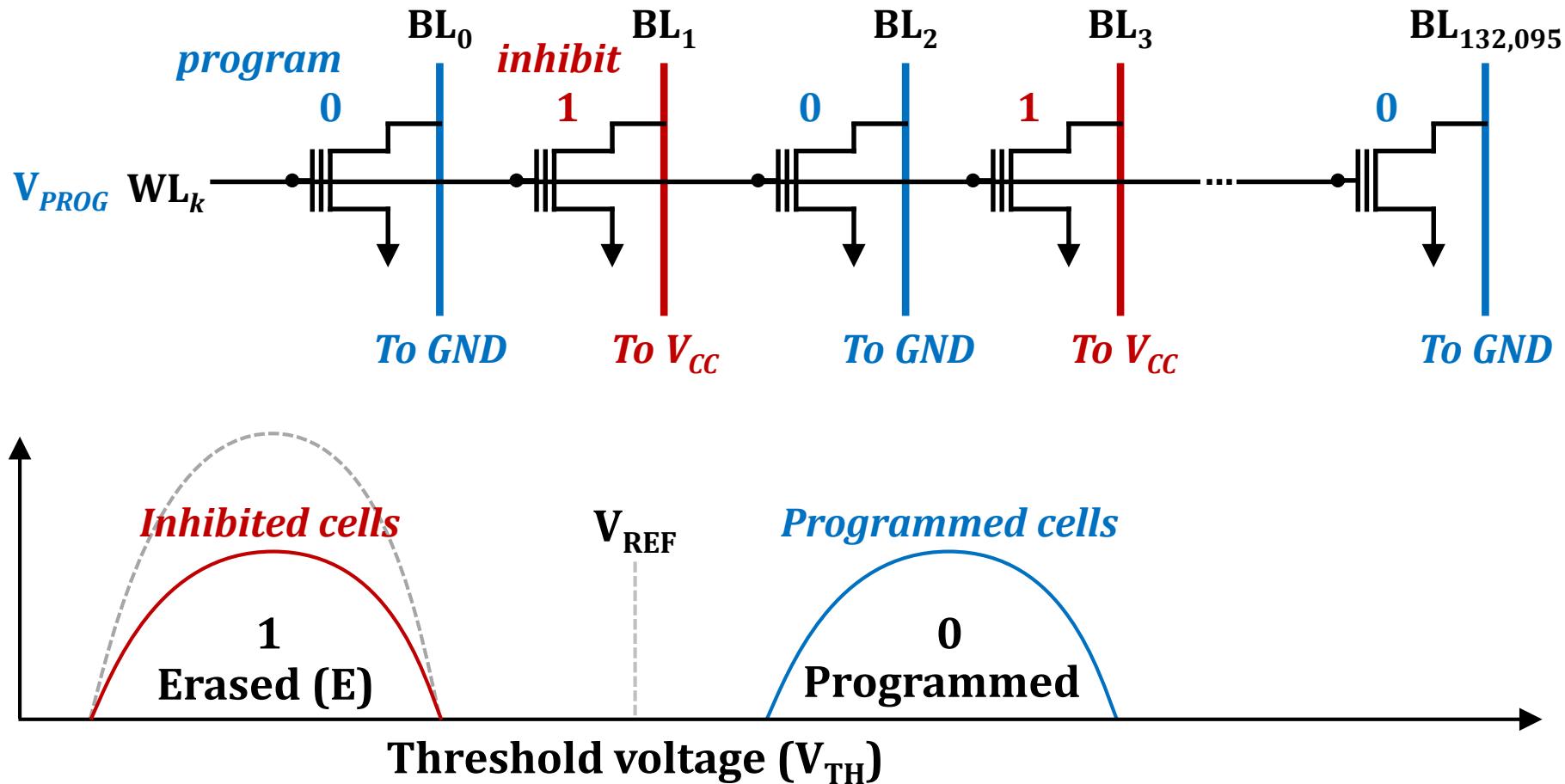
- BL control – **Inhibits cells** to not be programmed



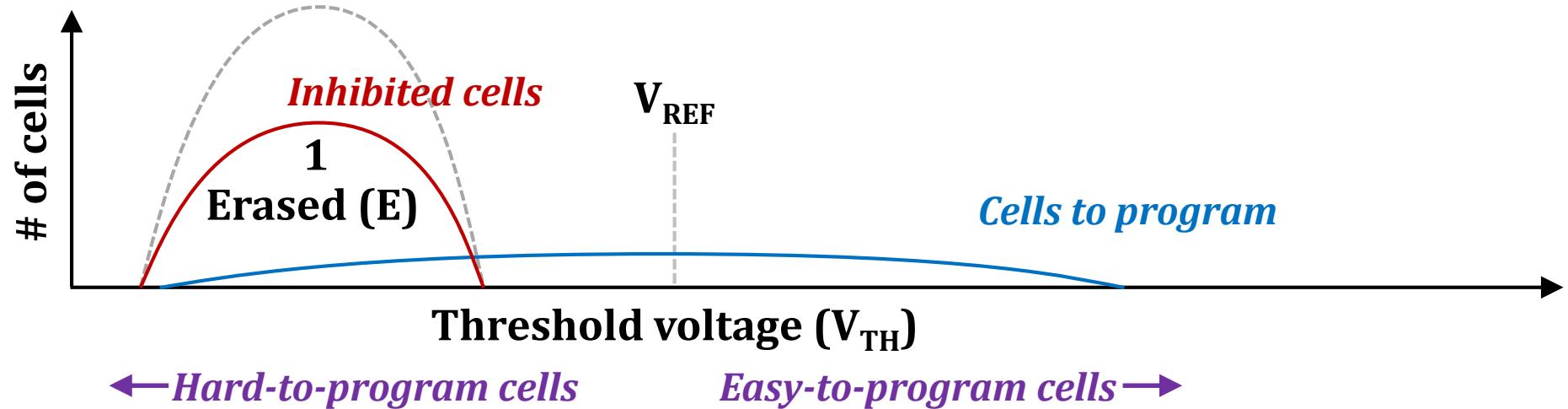
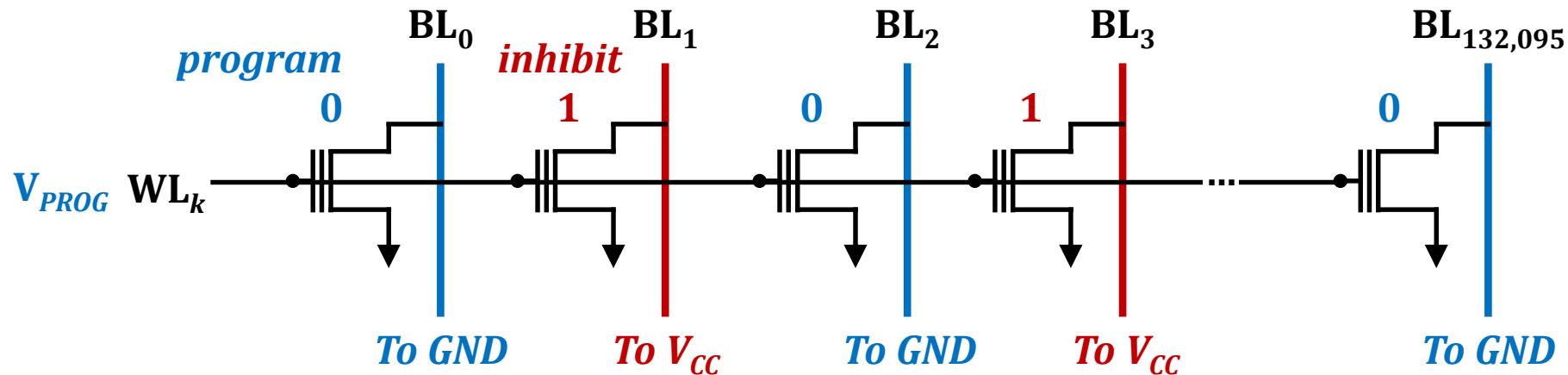
Basic Operation: Page Program



Basic Operation: Page Program

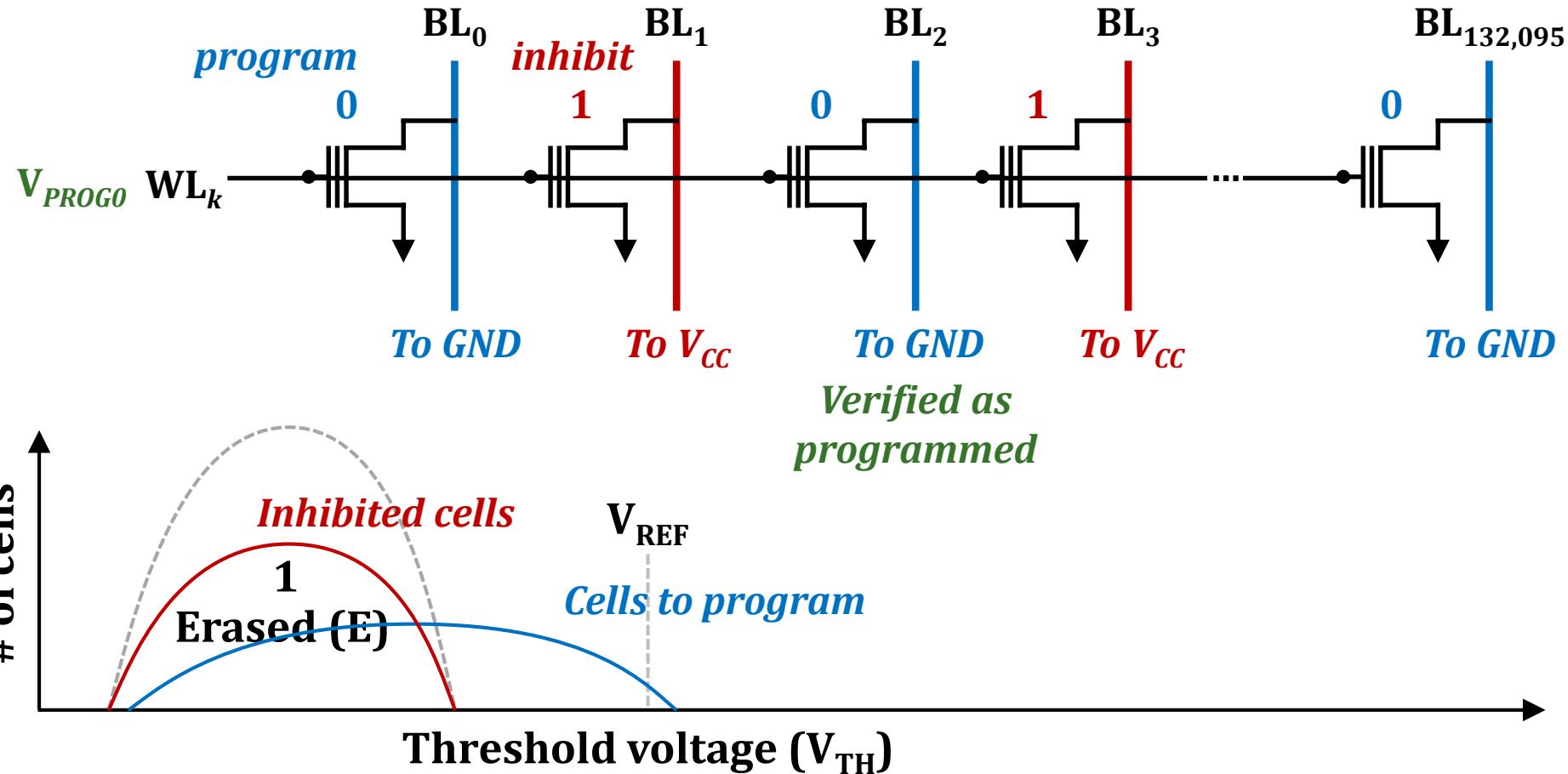


Basic Operation: Page Program



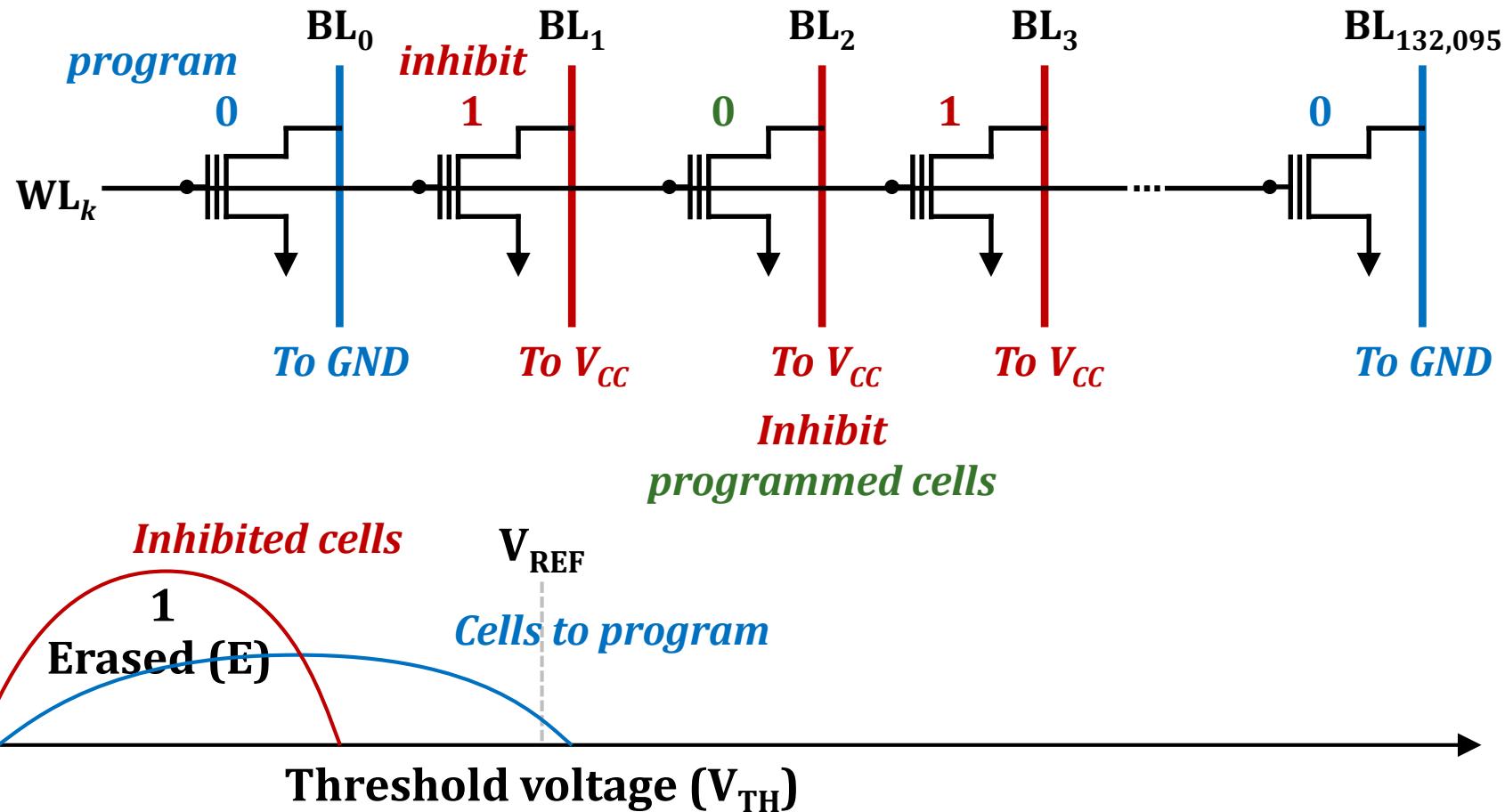
Basic Operation: Page Program

■ Incremental Step-Pulse Programming (ISPP)



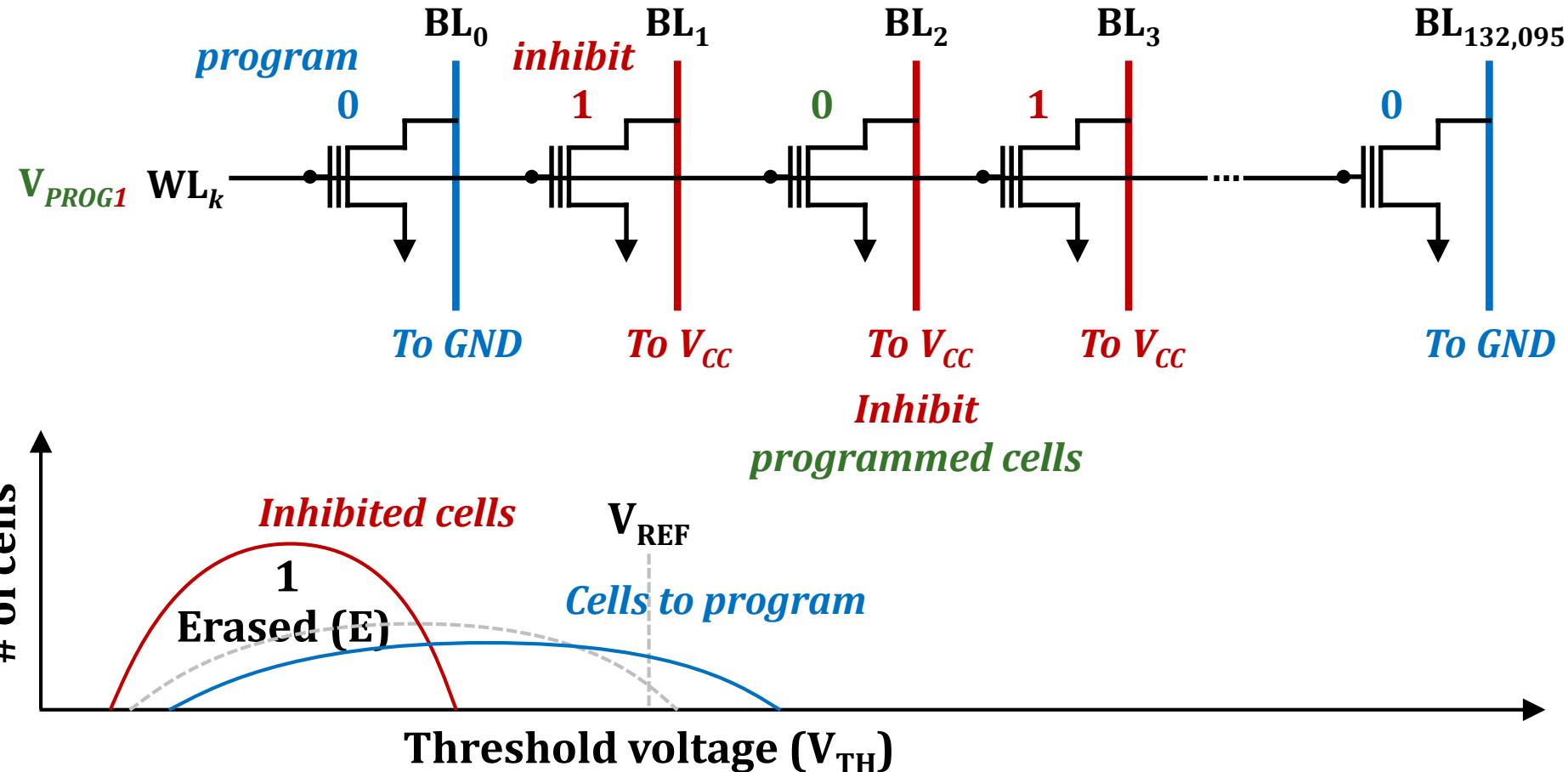
Basic Operation: Page Program

■ Incremental Step-Pulse Programming (ISPP)



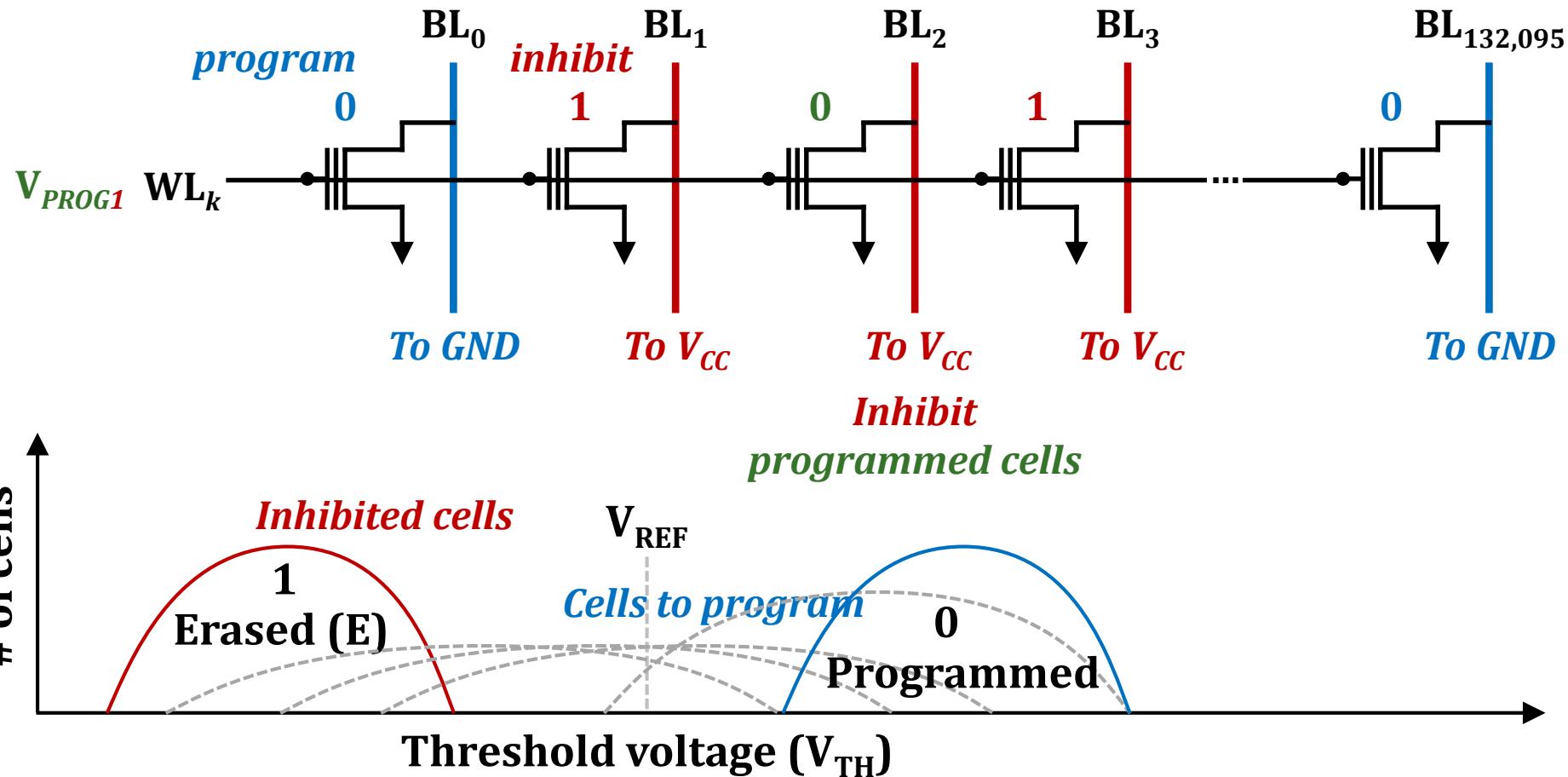
Basic Operation: Page Program

■ Incremental Step-Pulse Programming (ISPP)



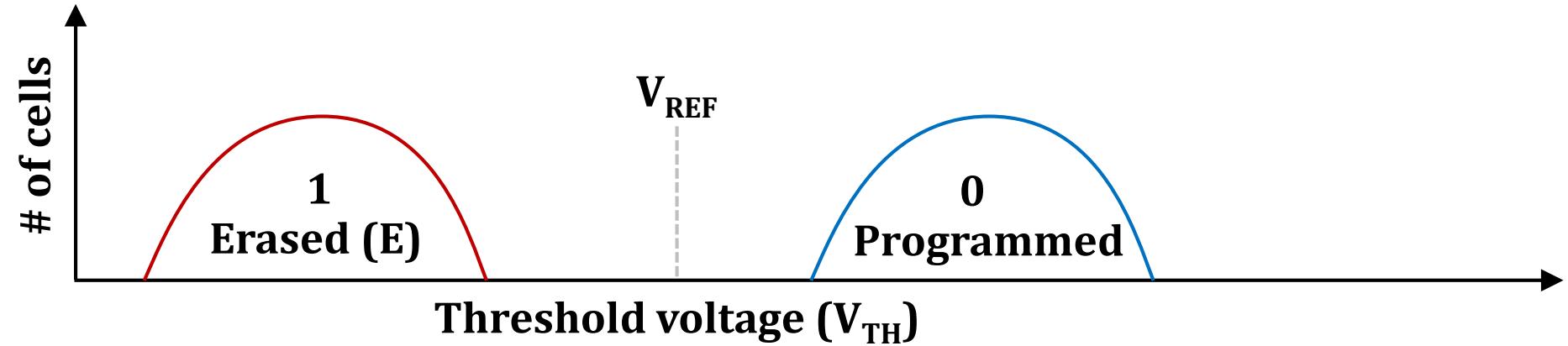
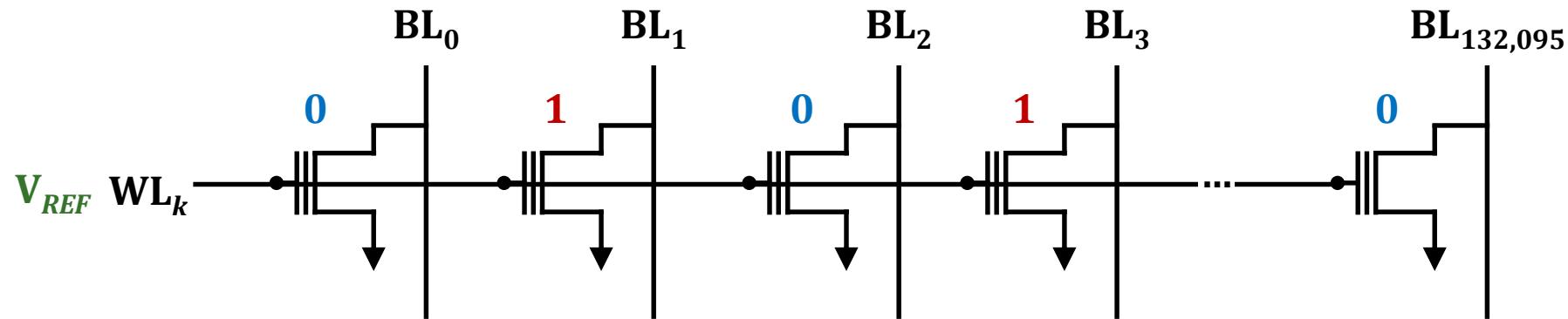
Basic Operation: Page Program

■ Incremental Step-Pulse Programming (ISPP)



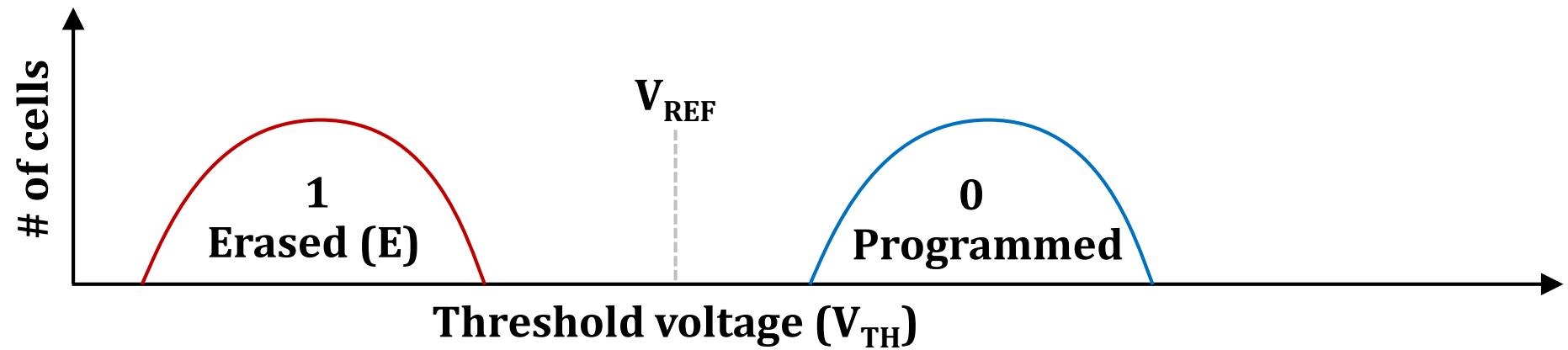
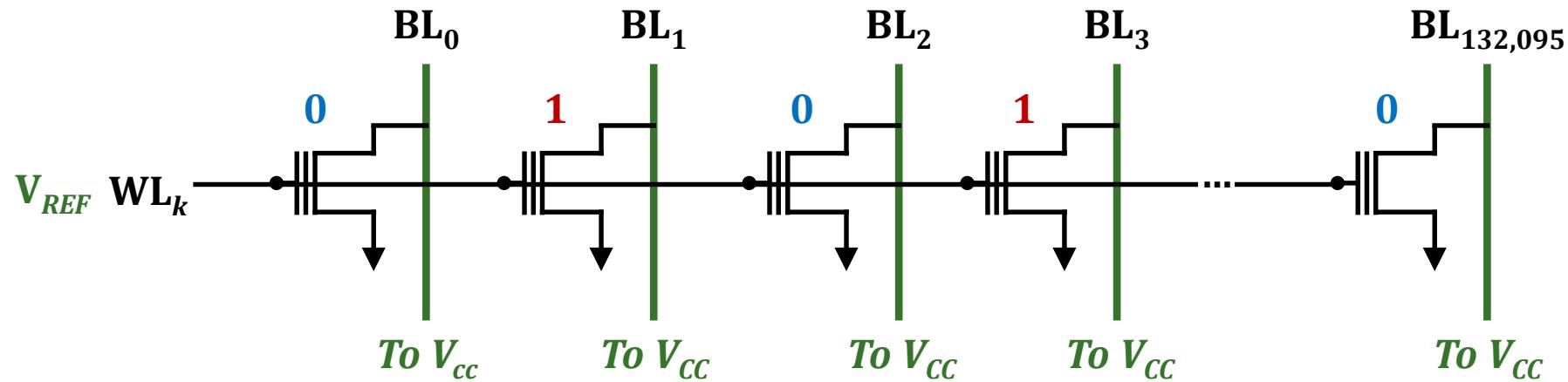
Basic Operation: Page Read

- WL control – All other cells operate as a resistance



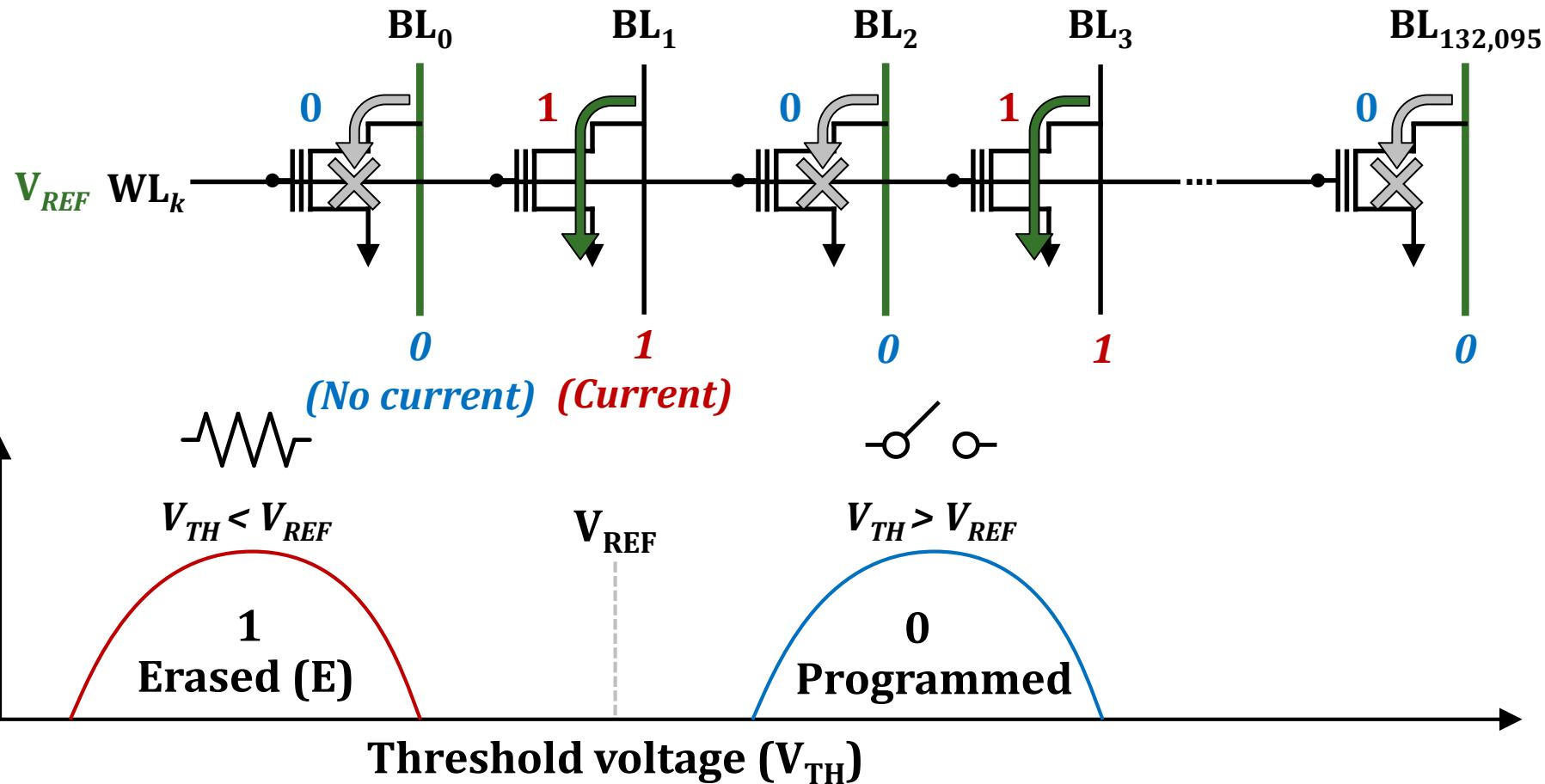
Basic Operation: Page Read

- BL control – Charge all BLs



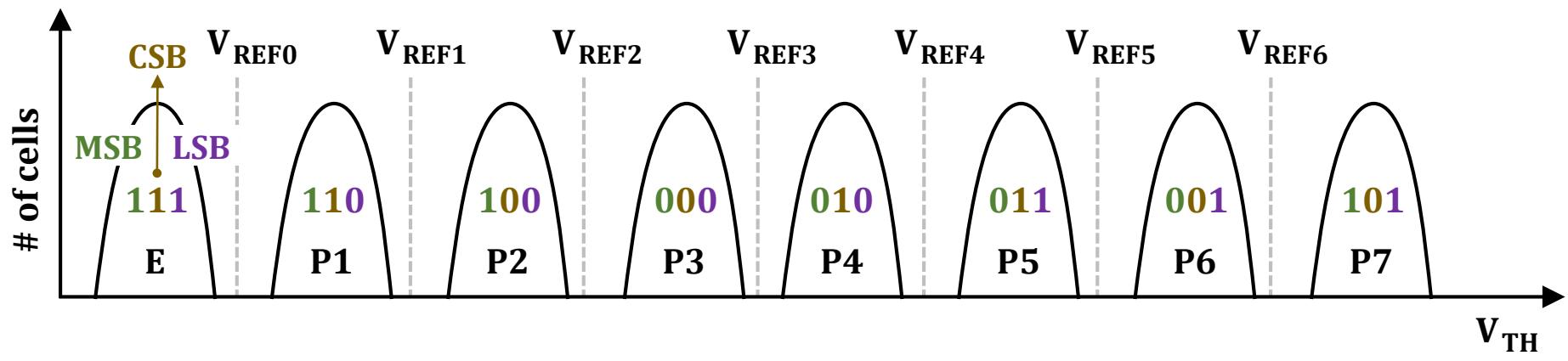
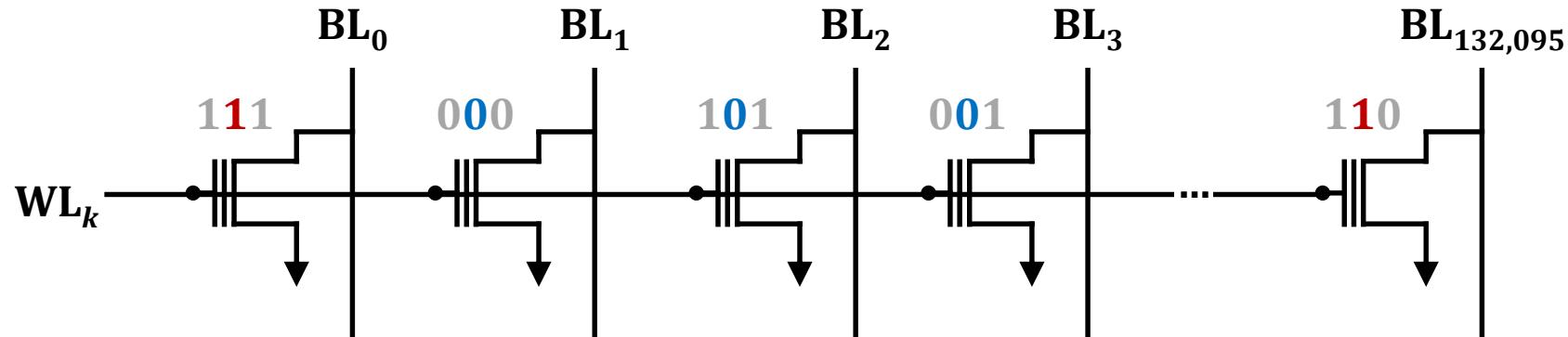
Basic Operation: Page Read

- Sensing the current through BLs



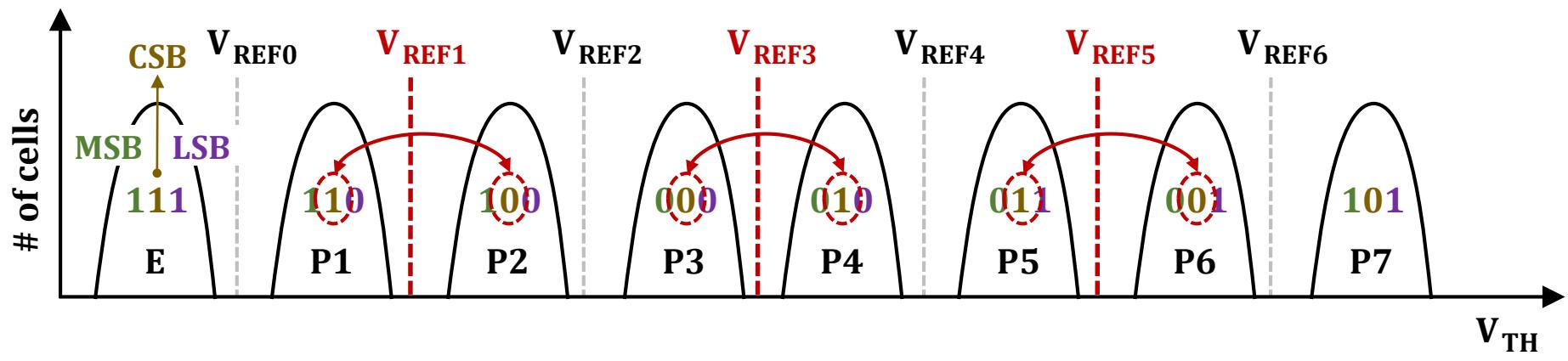
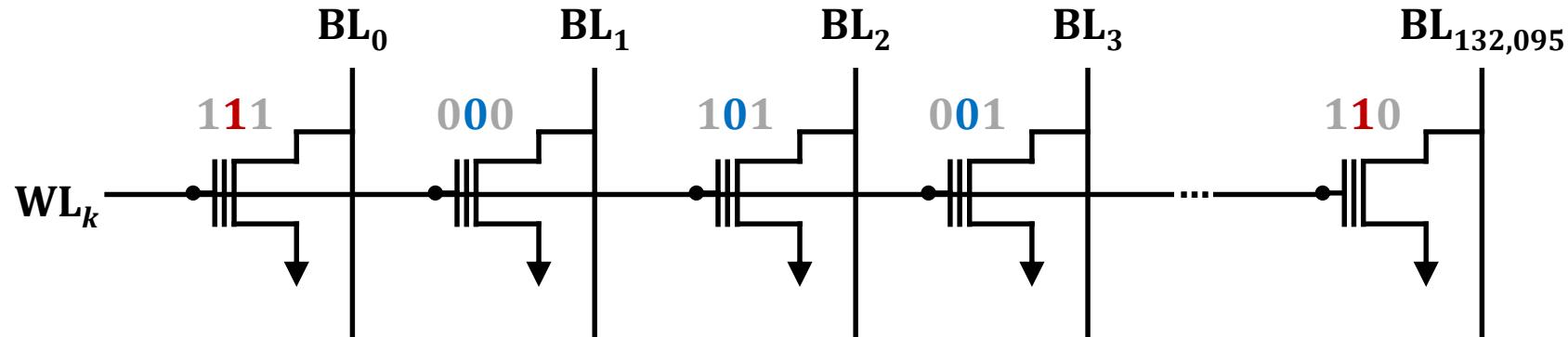
Basic Operation: Page Read - MLC

- Sensing the current through BLs



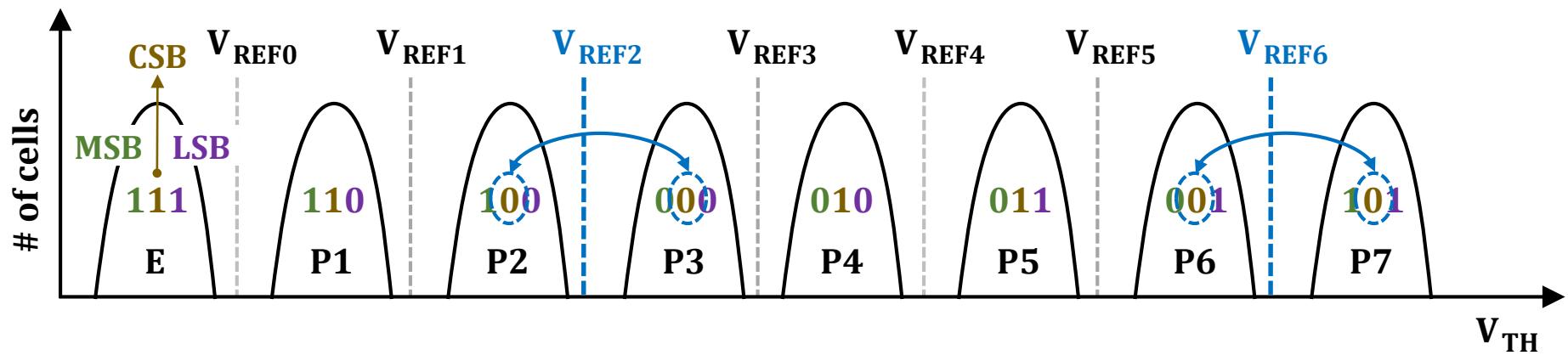
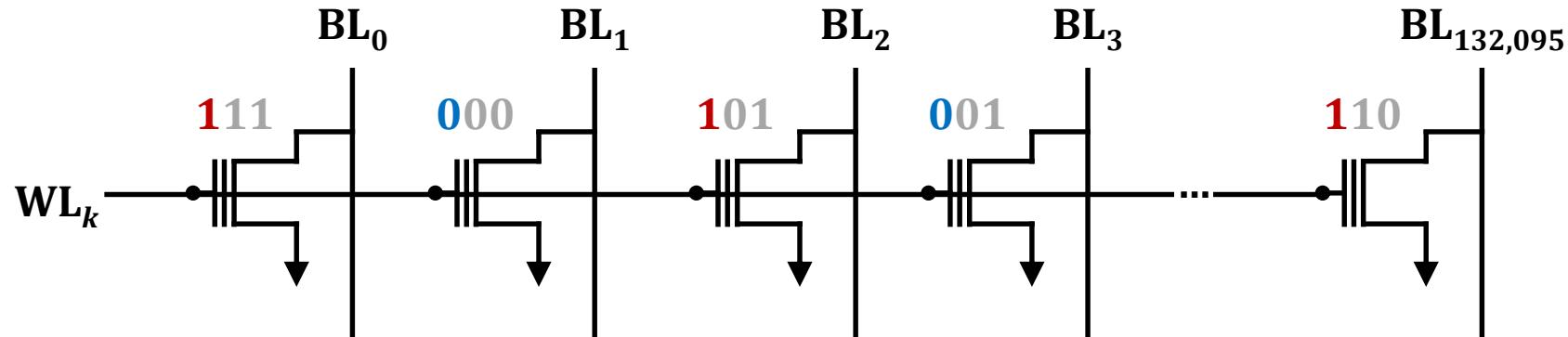
Basic Operation: Page Read - MLC

- Sensing the current through BLs



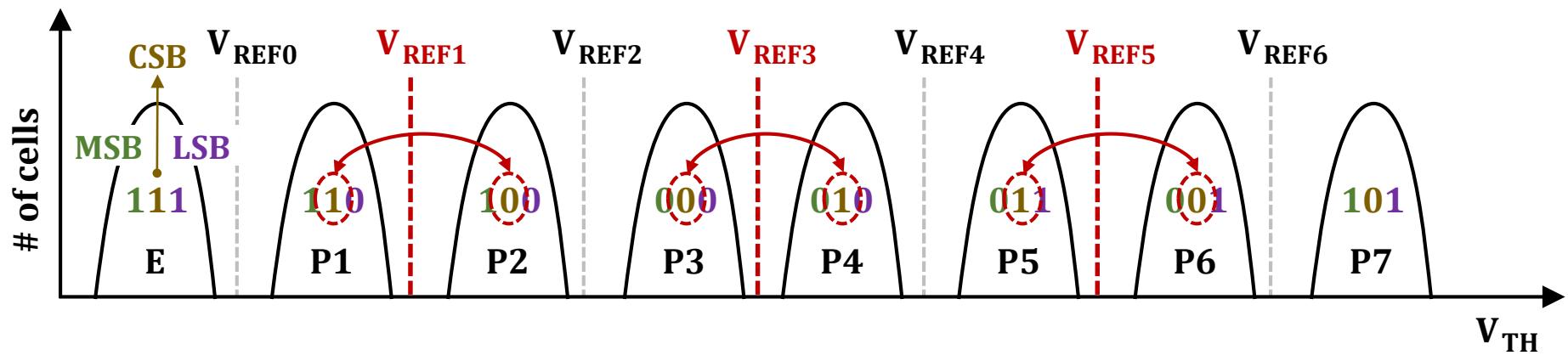
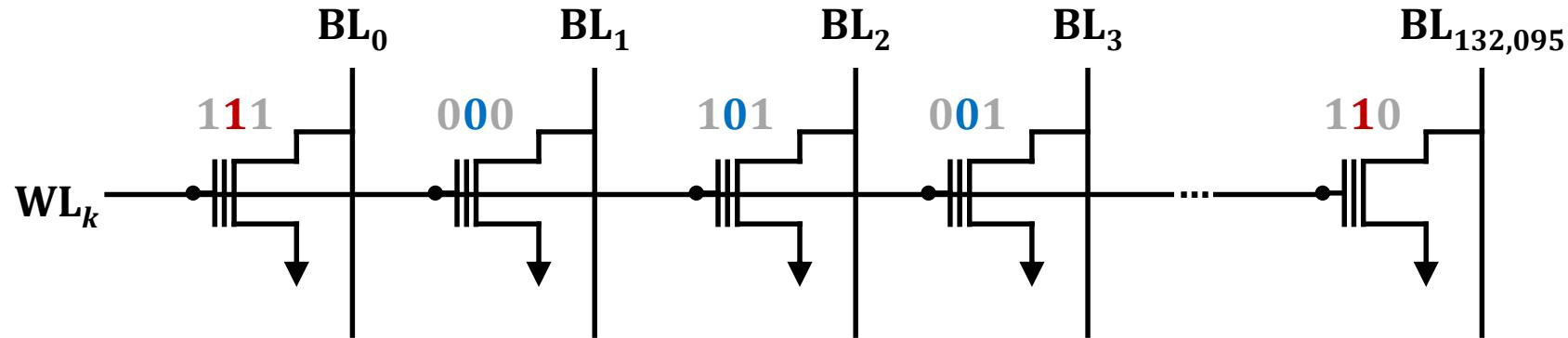
Basic Operation: Page Read - MLC

- Sensing the current through BLs



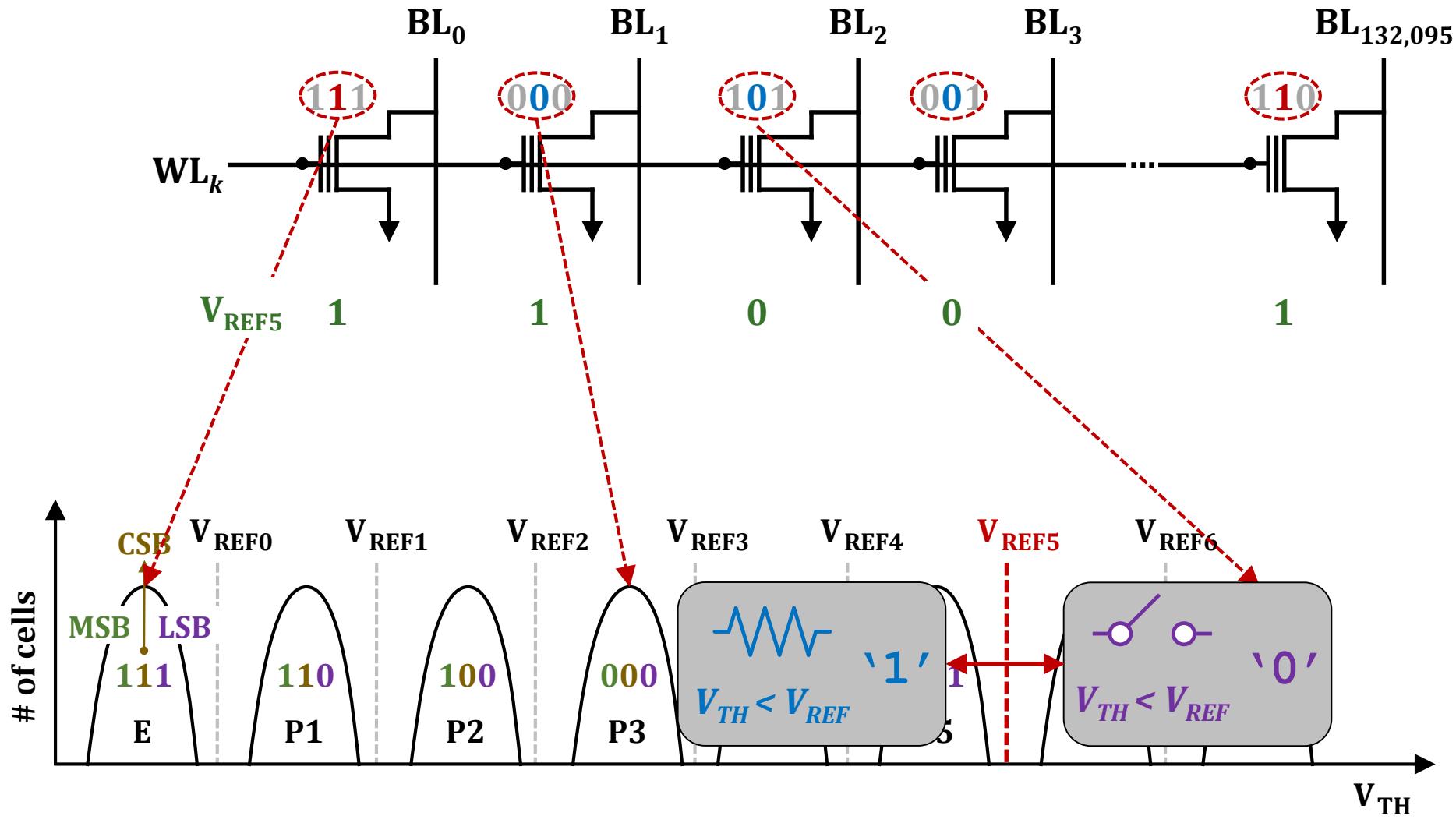
Basic Operation: Page Read - MLC

- Sensing the current through BLs



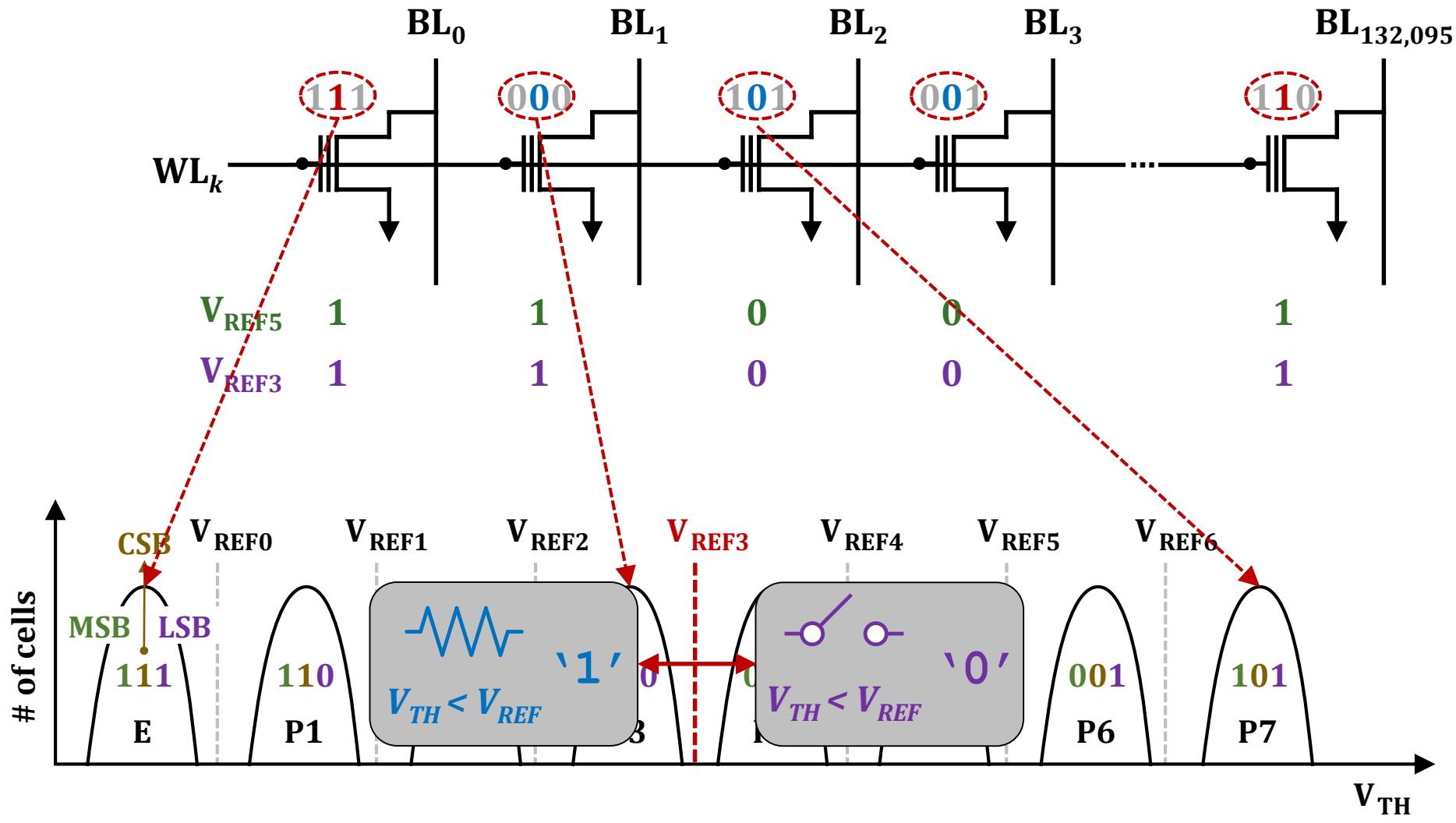
Basic Operation: Page Read - MLC

- Sensing the current through BLs



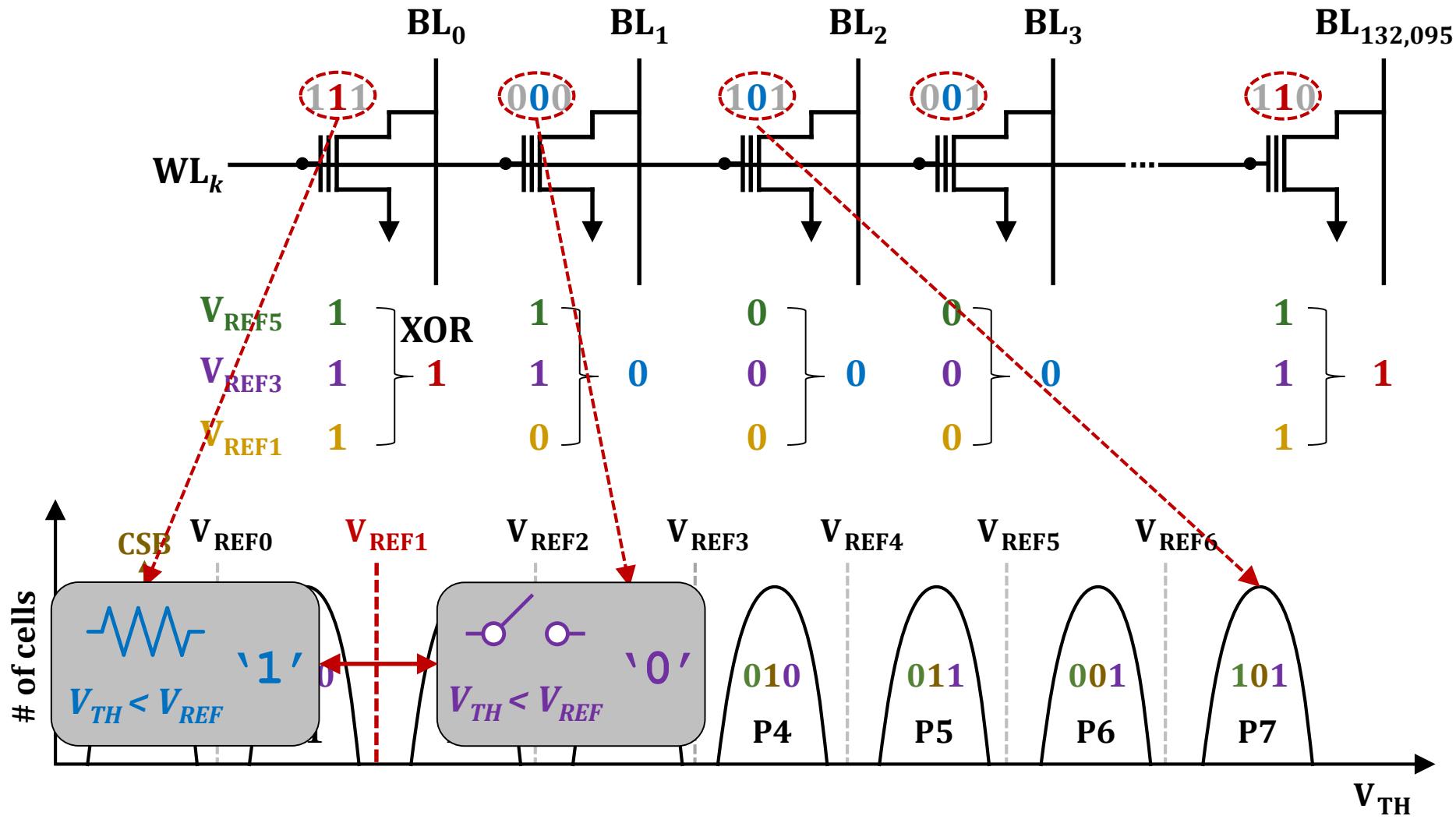
Basic Operation: Page Read - MLC

- Sensing the current through BLs



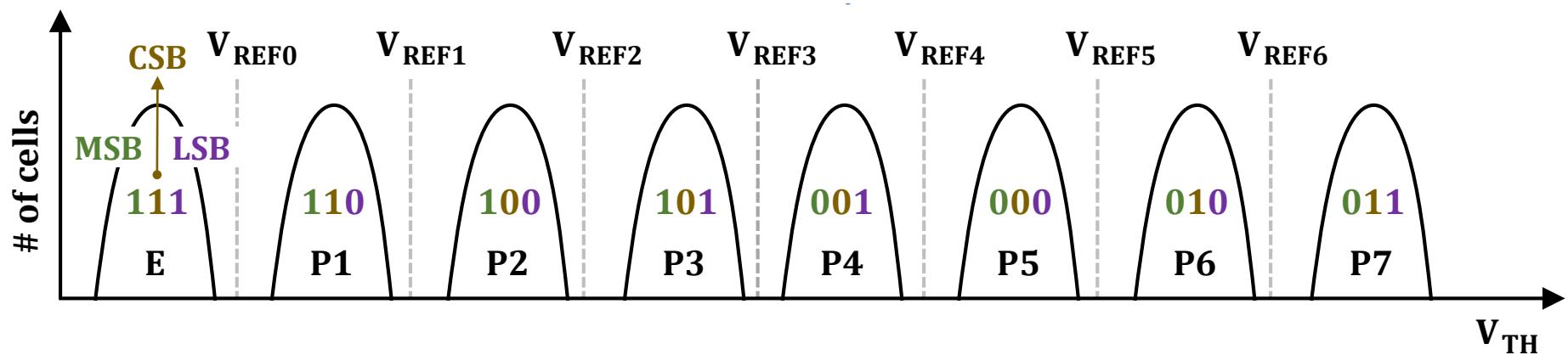
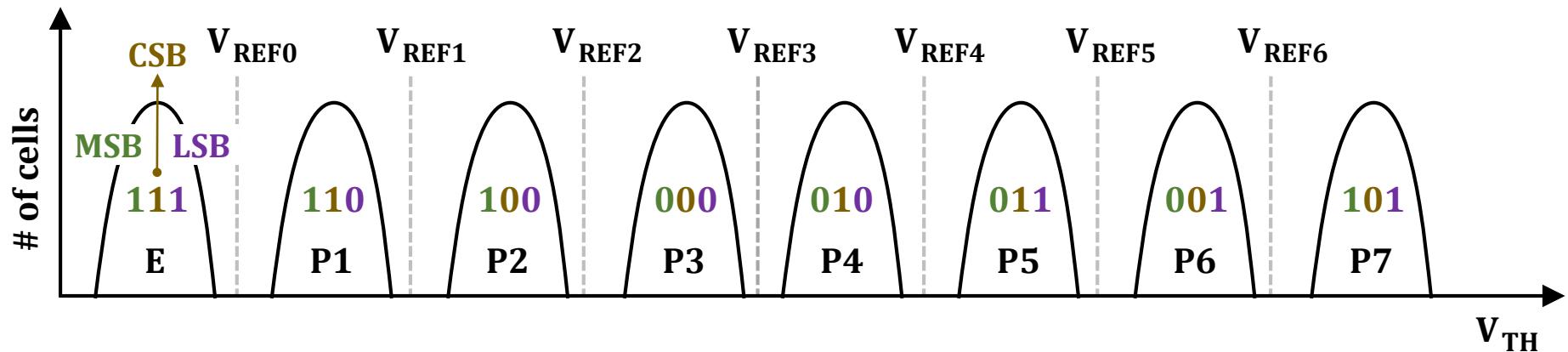
Basic Operation: Page Read - MLC

- Sensing the current through BLs



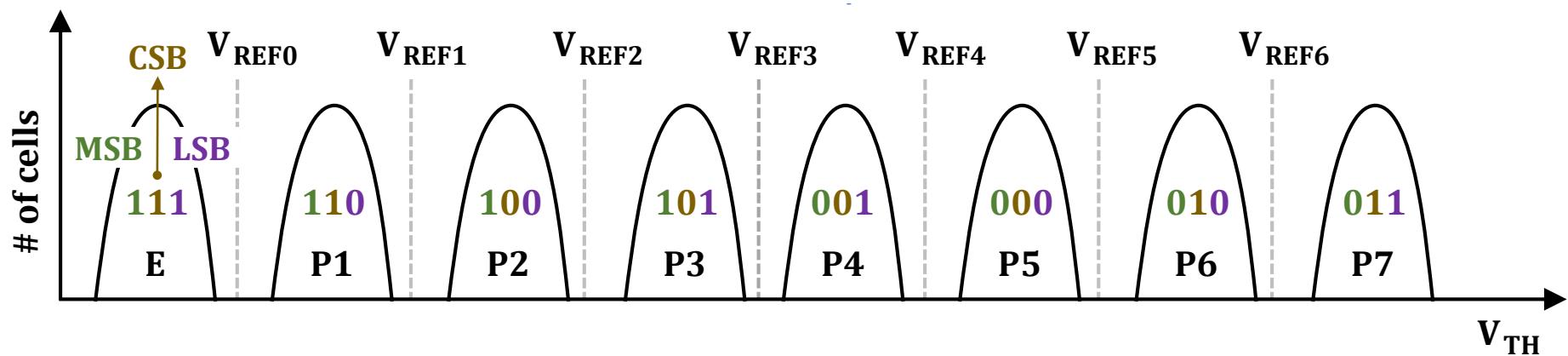
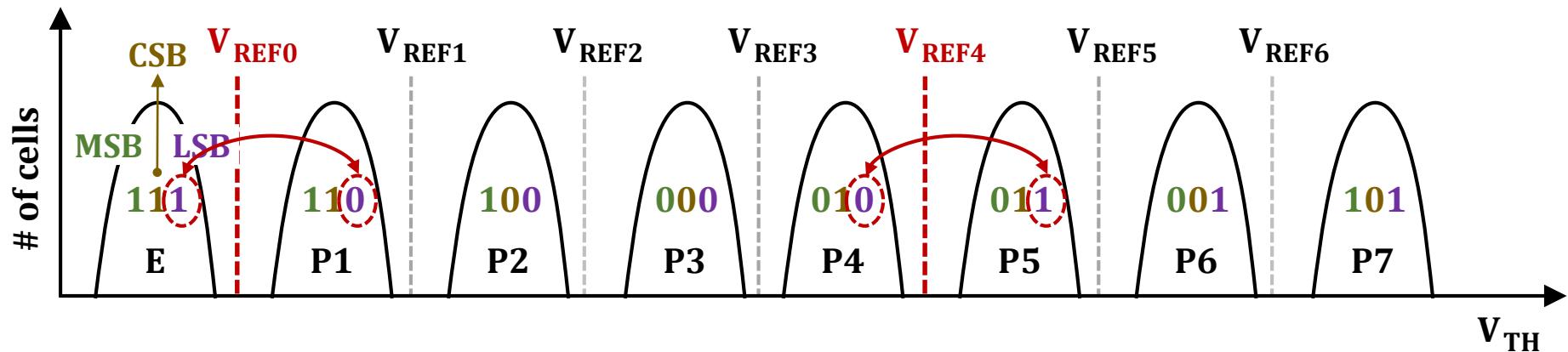
Basic Operation: Page Read – Takeaways

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



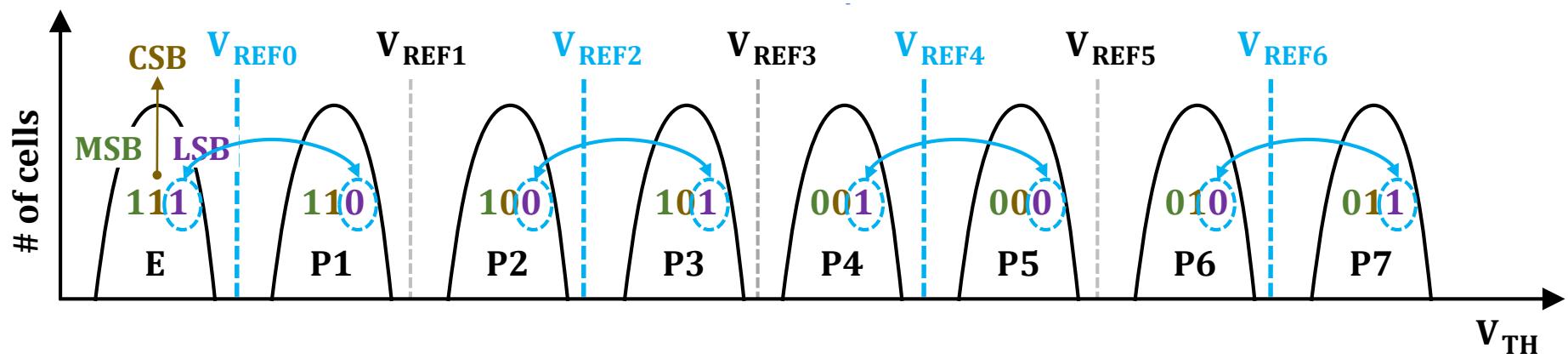
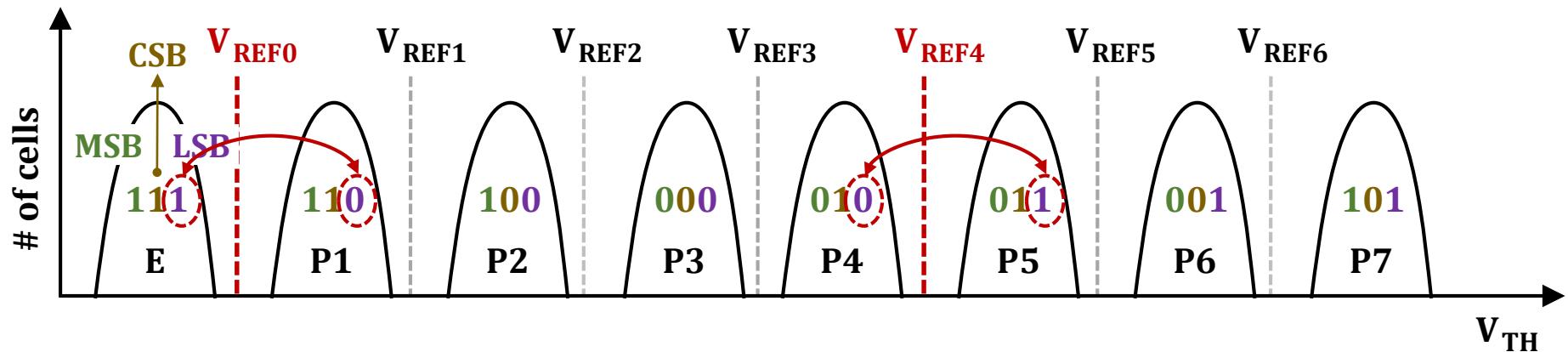
Basic Operation: Page Read – Takeaways

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



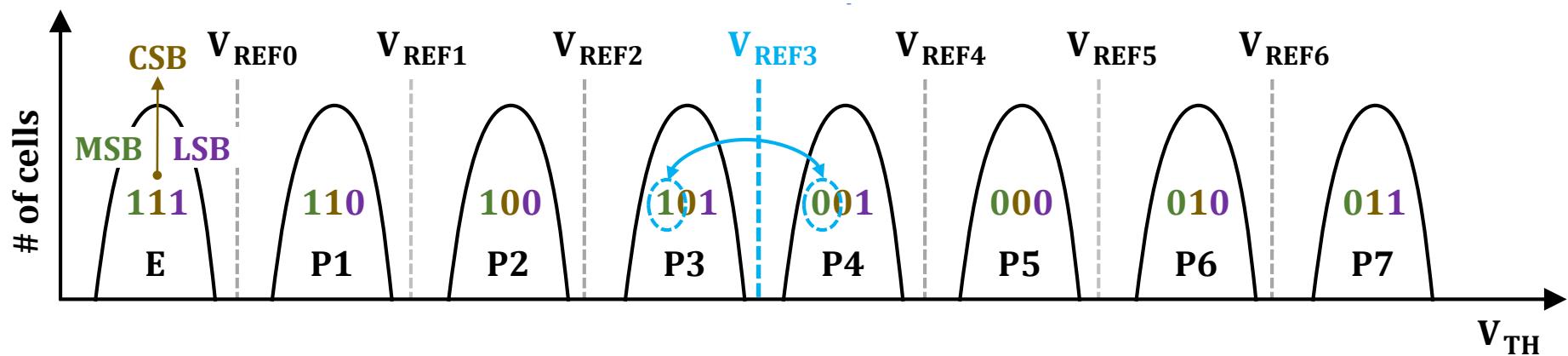
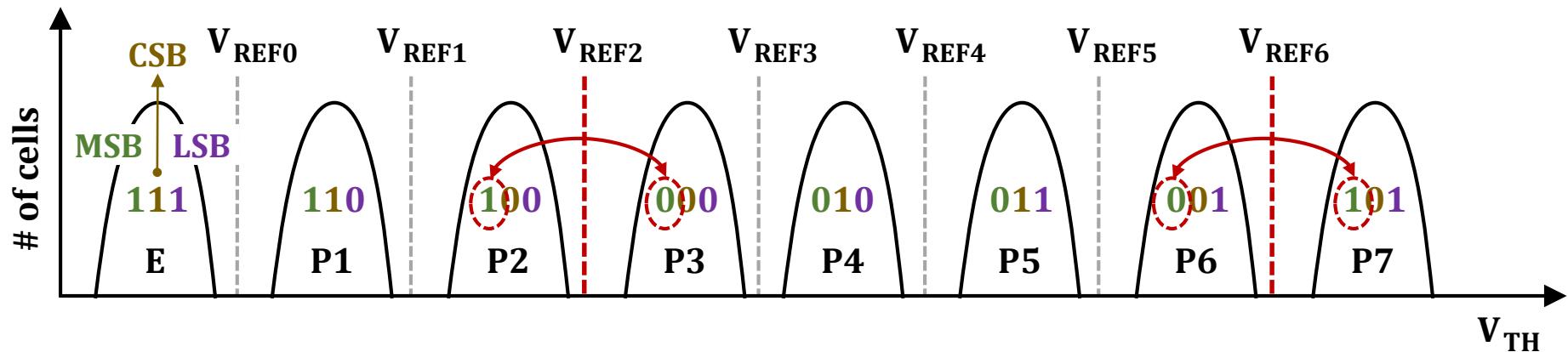
Basic Operation: Page Read – Takeaways

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



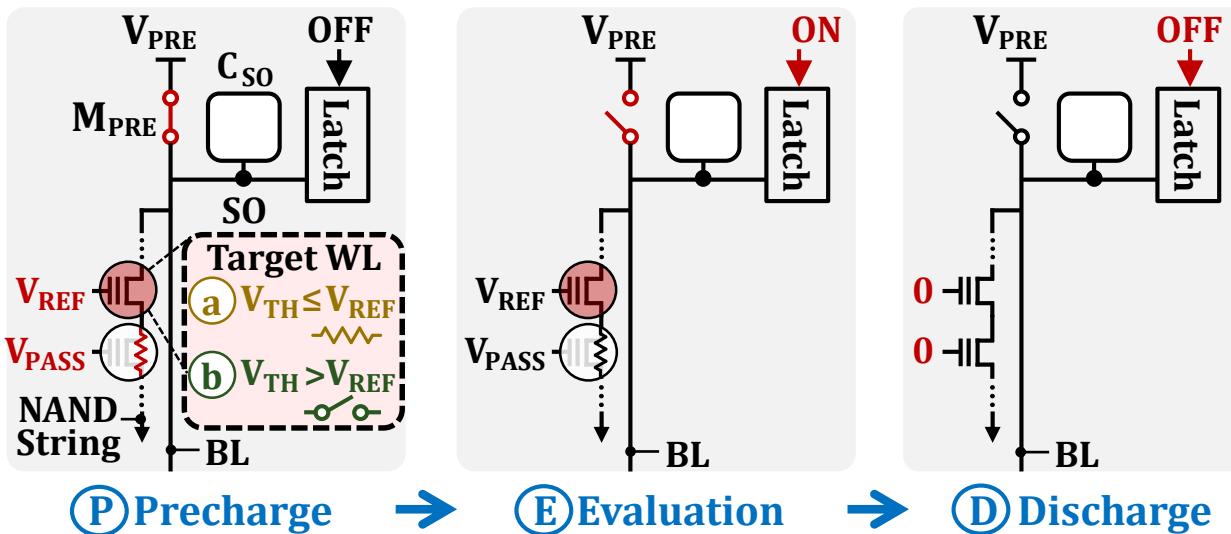
Basic Operation: Page Read – Takeaways

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

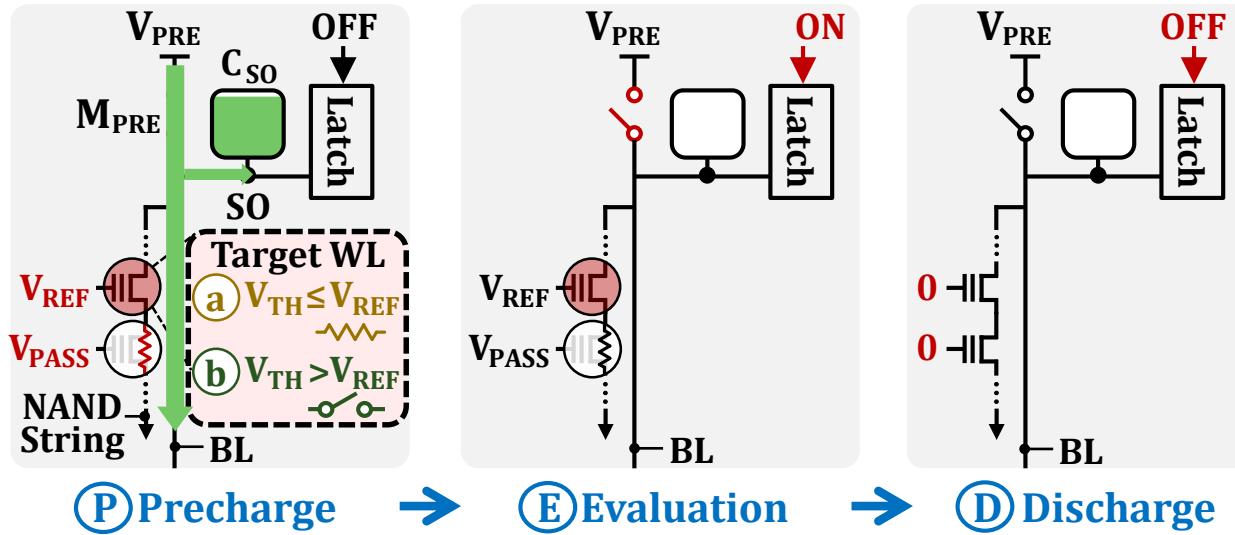


Read Mechanism

- NAND flash read mechanism consists of three steps:
 - 1) Precharge
 - 2) Evaluation
 - 3) Discharge

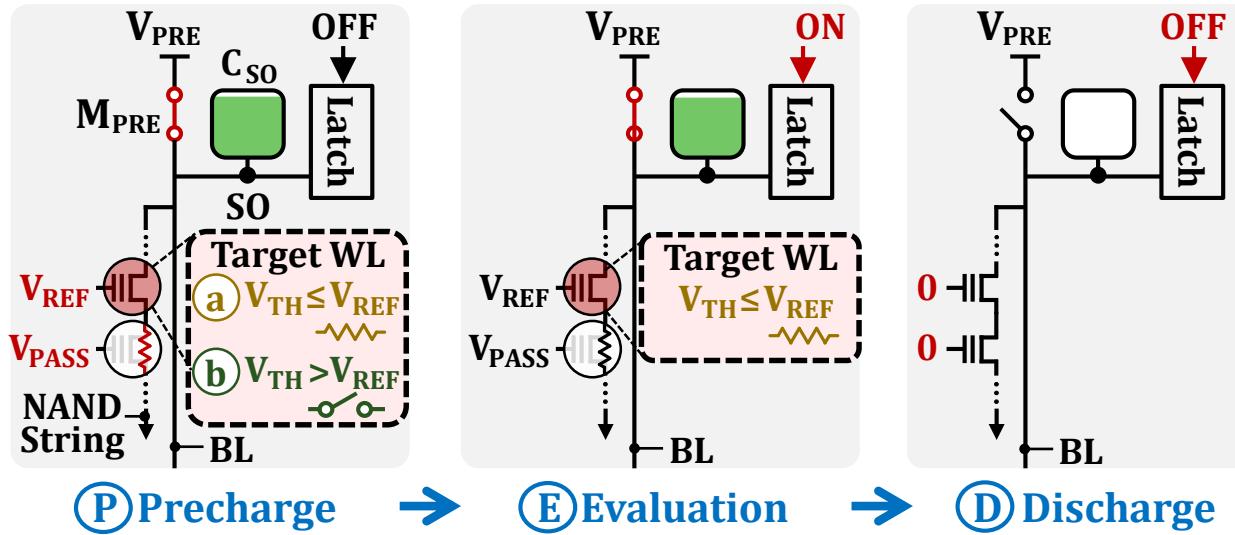


Read Mechanism: Precharge

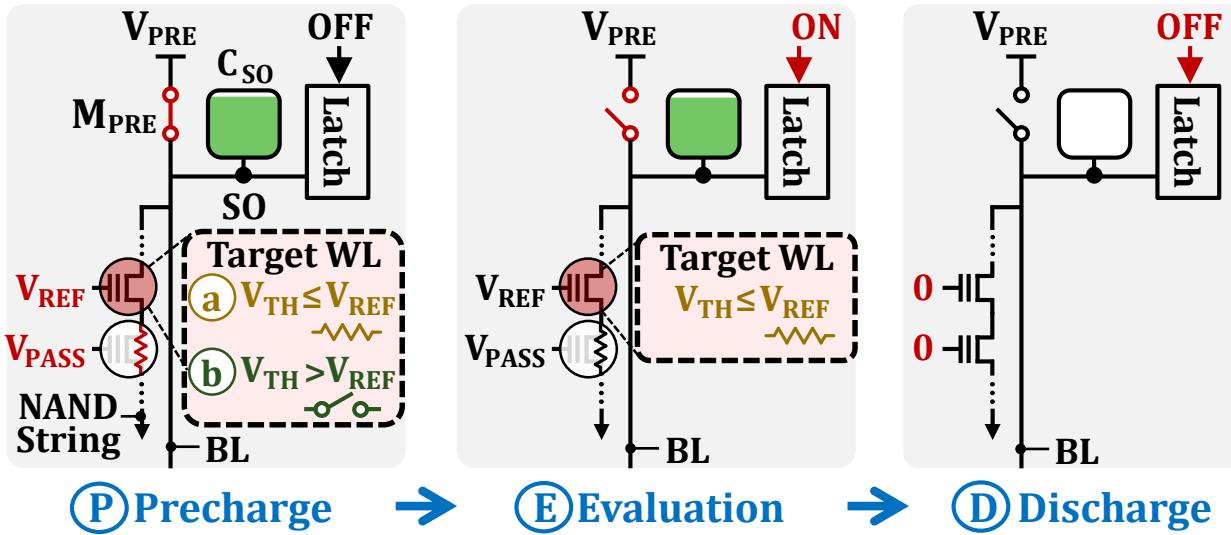


Enable precharge transistor M_{PRE} to charge all target BLs and their sense-out capacitors (C_{SO}) to V_{PRE}

Read Mechanism: Evaluation

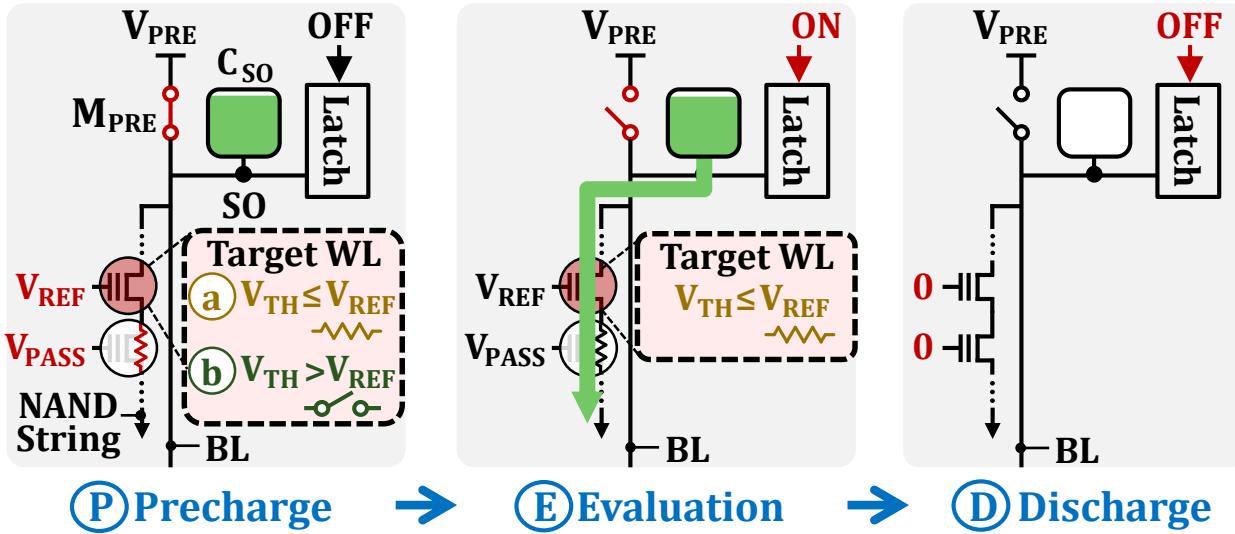


Read Mechanism: Evaluation



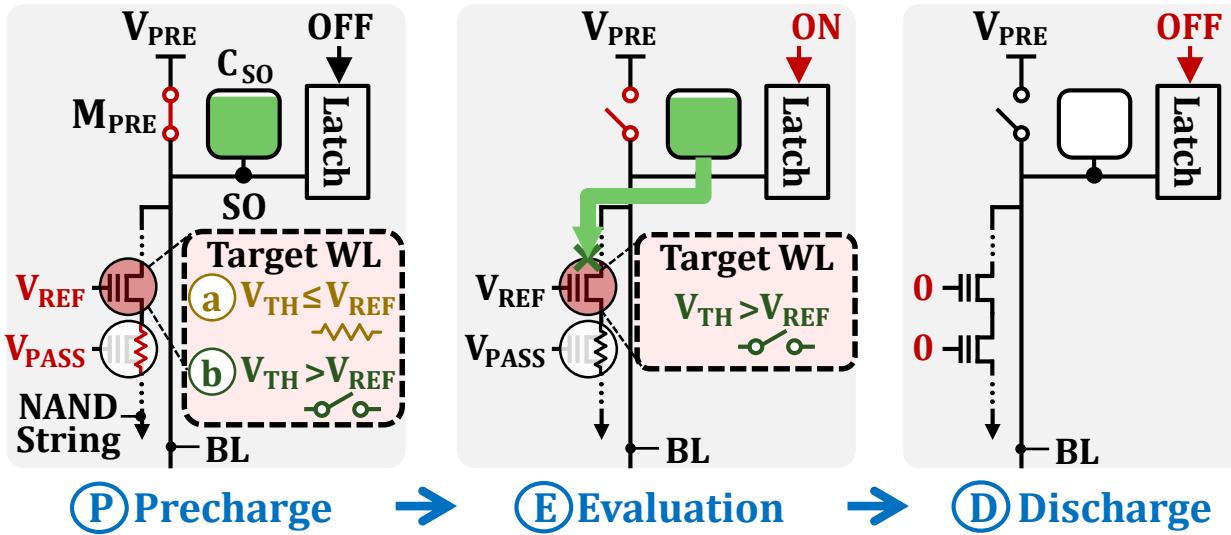
Disconnect the BLs from V_{PRE} and enable the latching circuit

Read Mechanism: Evaluation



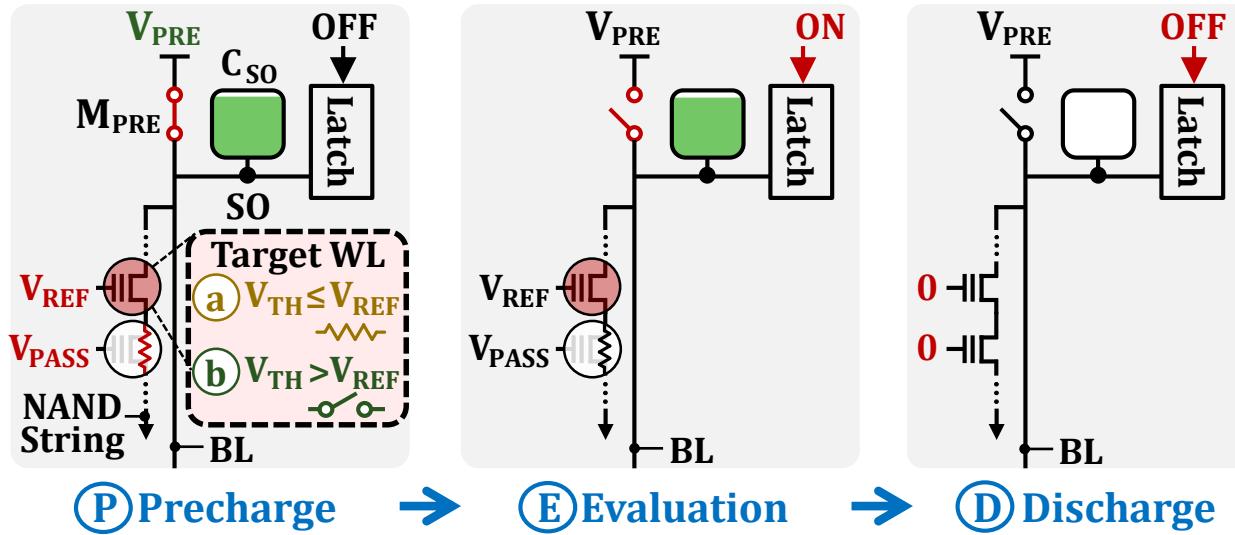
If $V_{TH} \leq V_{REF}$, the charge in C_{SO} quickly flows through the NAND string (Sensed as 1)

Read Mechanism: Evaluation



If $V_{TH} > V_{REF}$, the target cell blocks the BL discharge current (Sensed as 0)

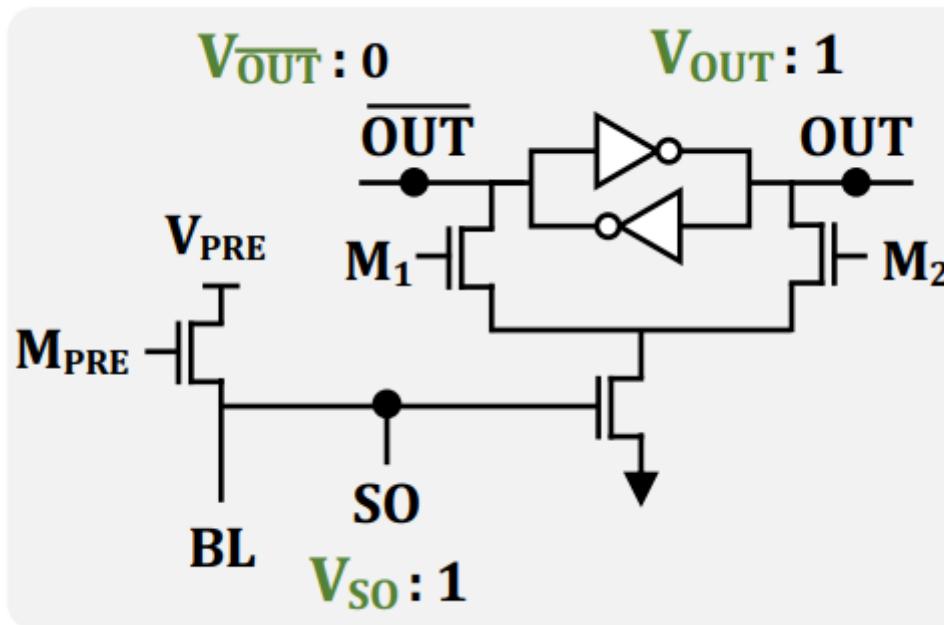
Read Mechanism: Discharge



Bitlines are discharged to return the NAND string to its initial state for future operations

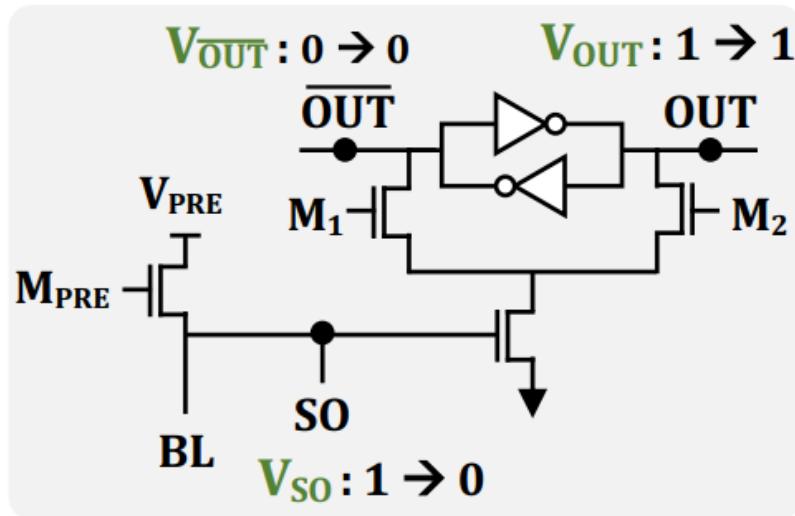
Latching Circuit

- Before the evaluation step, the chip initializes the latching circuit
 - Activating transistor M_1
 - $V_{\overline{OUT}} = 0$
 - $V_{out} = 1$

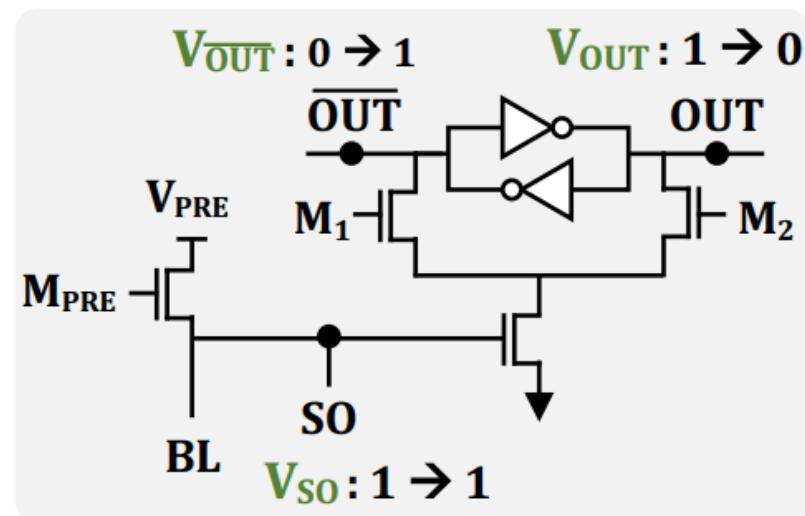


Latching Circuit

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



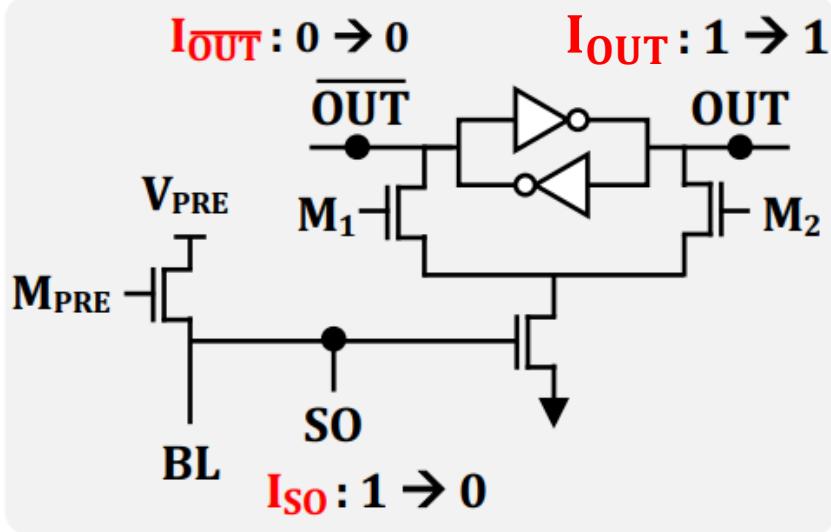
(a) $V_{TH} \leq V_{REF}$



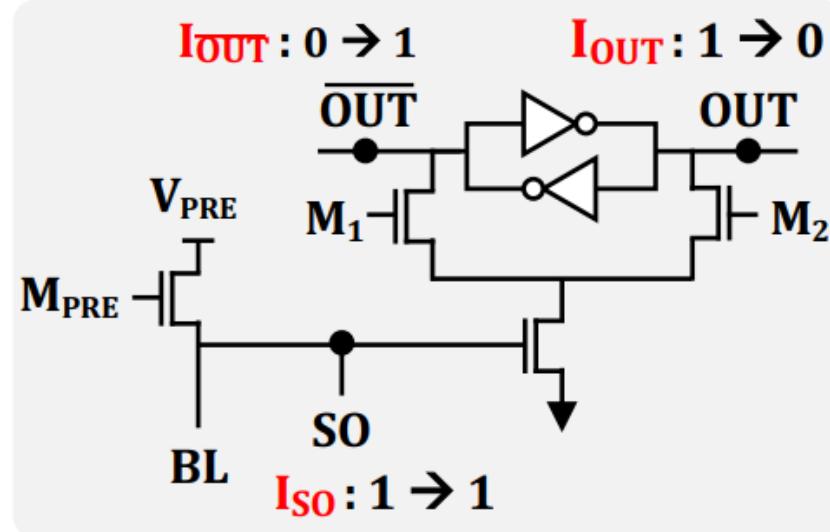
(b) $V_{TH} > V_{REF}$

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

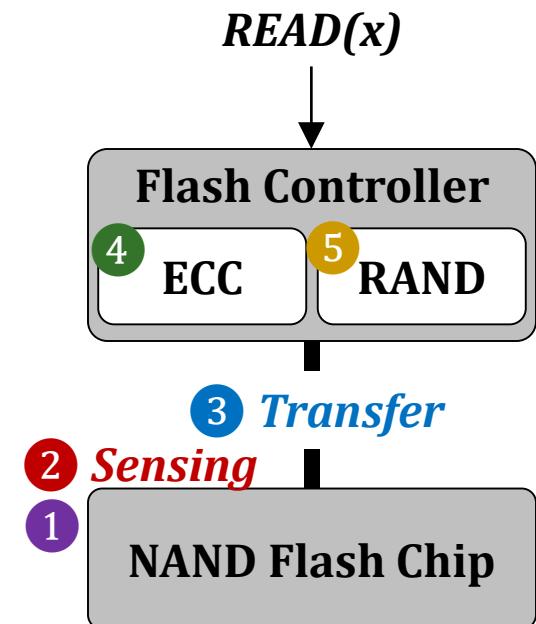
Source: <https://www.anandtech.com/show/16504/the-samsung-ssd-980-500gb-1tb-review>

NAND Flash Chip Performance

- 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 100us/700us/3ms
- 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 tDMA = **16 us**

NAND Flash Chip Performance (Cont.)

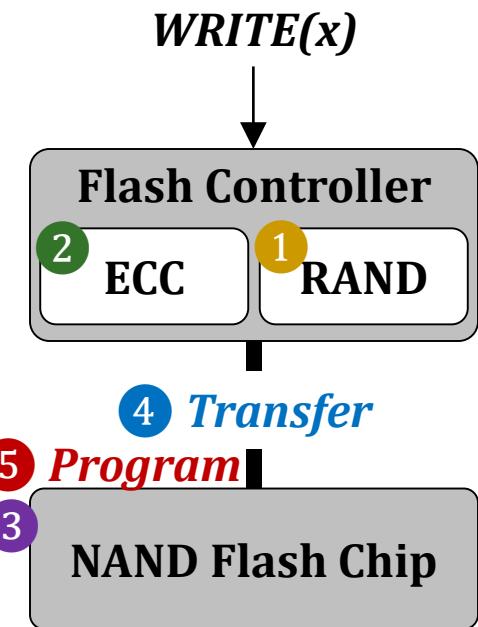
- tR, tPROG, and tBERS
 - Latencies for chip-level read/program/erase operations
 - tR: 50~100 us
 - tPROG: 700us~1000 us
 - tBERS: 3ms~5ms
- Flash-controller level latency
 - 1-Gb I/O rate and 16-KiB page size
 - Read
 - $(t_{CMD}) + tR + t_{DMA} + t_{ECC_{DEC}} + (t_{RND})$
 - e.g., $100 + 16 + 20 = 136$ 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
 - $(t_{CMD}) + tR + t_{DMA} + t_{ECC_{DEC}} + (t_{RND})$
 - e.g., $100 + 16 + 20 = 136$ us
 - Program
 - $(t_{RND}) + t_{ECC_{ENC}} + (t_{CMD}) + t_{DMA} + t_{PROG}$
 - e.g., $20 + 16 + 700 = 736$ us



NAND Flash Chip Performance (Cont.)

- How about bandwidth?

- Read

- $16 \text{ KiB} / 136 \text{ us} \approx 120 \text{ MB/s}$

- Write

- $16 \text{ KiB} / 736 \text{ us} \approx 22 \text{ MB/s}$

WAIT!

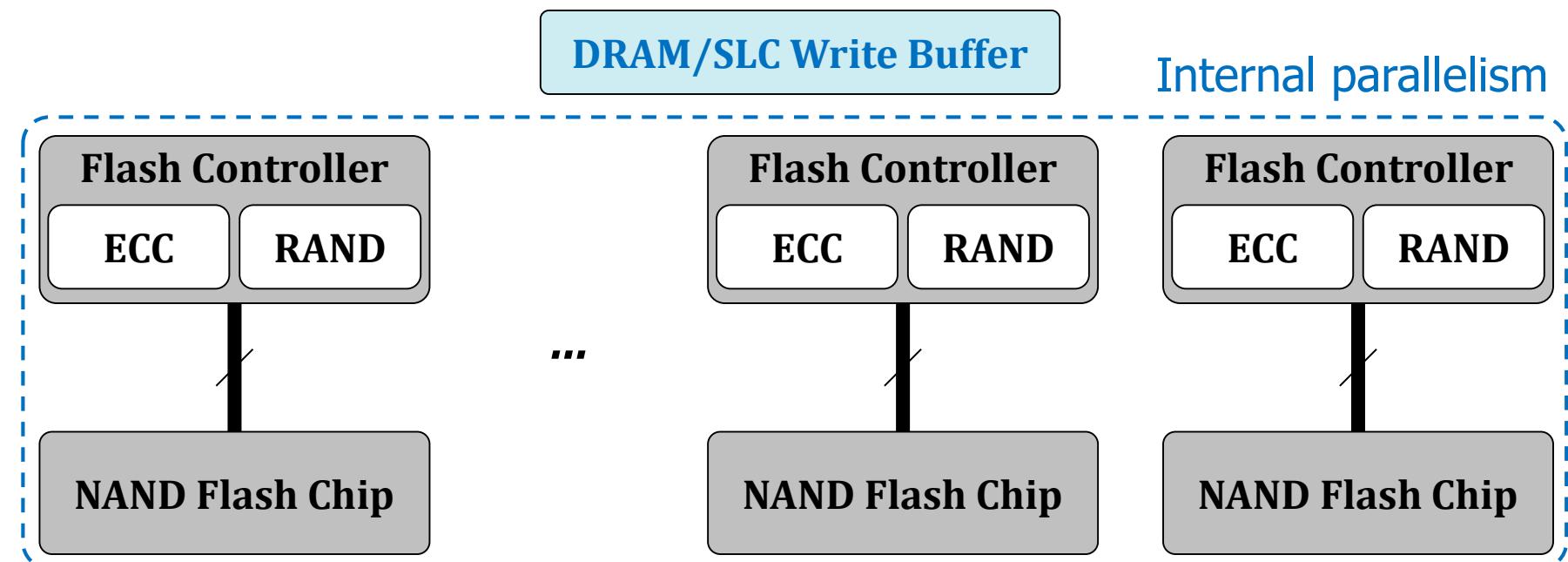
SSD read latency: **67 us**

SSD read bandwidth: **3.5 GB/s**

SSD write latency: **47 us**

SSD write bandwidth: **3 GB/s**

Optimizations w/ advanced commands



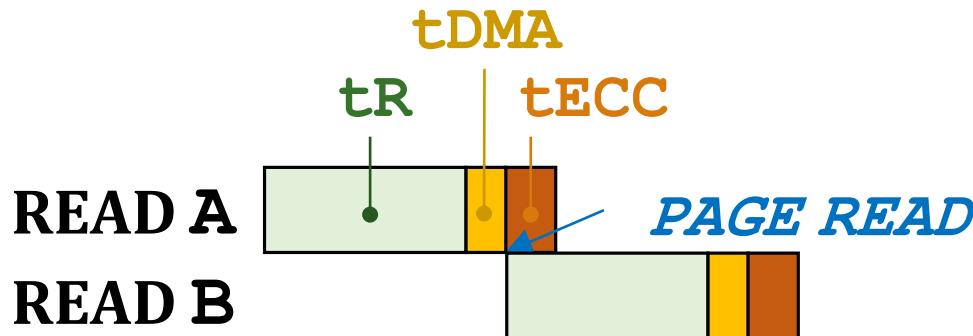
Advanced Commands for Small Reads

- 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 operation¹
 - Microarchitecture-level optimization – **directly reduces tR**
- Optimization 2: **Random Data Out (RDO)**
 - Data transfer with an arbitrary offset and size
 - Reduce **tDMA** and **tECC_{DEC}**

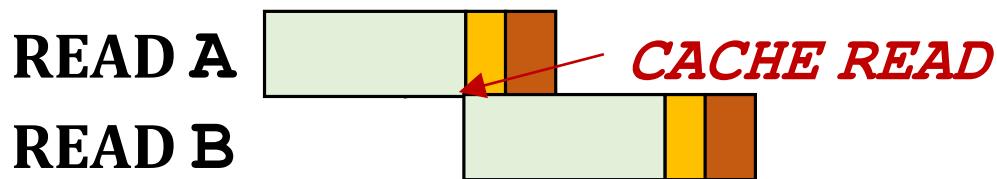
¹https://media-www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn_2993_snap_read.pdf

CACHE READ Command

- Performs consecutive reads in a pipelined manner



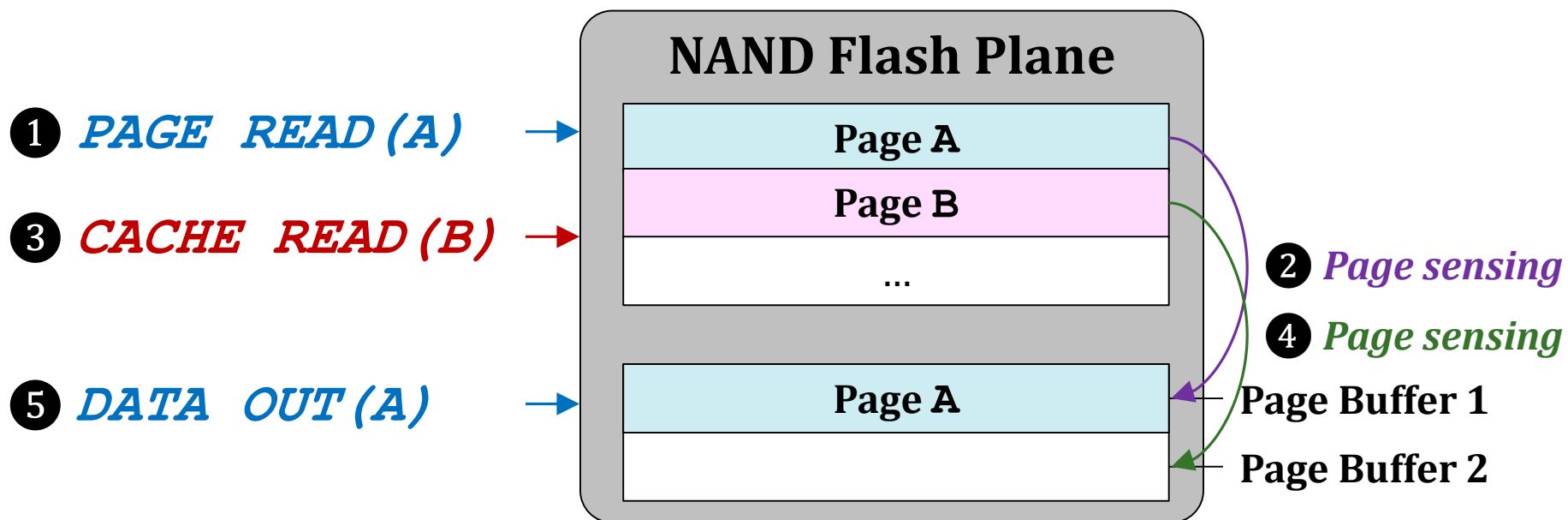
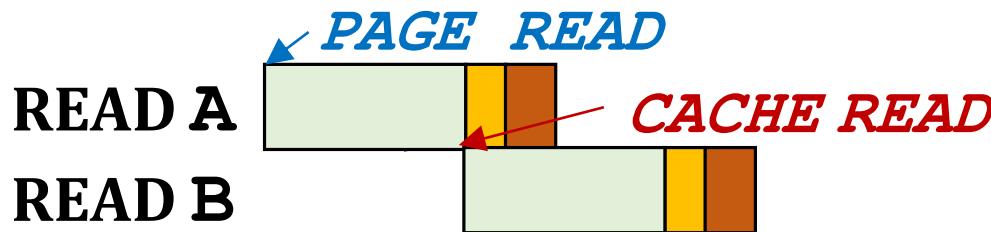
Regular PAGE READ:
Overlaps **only** tECC with tR



CACHE READ:
Overlaps tDMA & tECC
with tR

Enabling the CACHE READ Command

- Needs additional on-chip page buffer

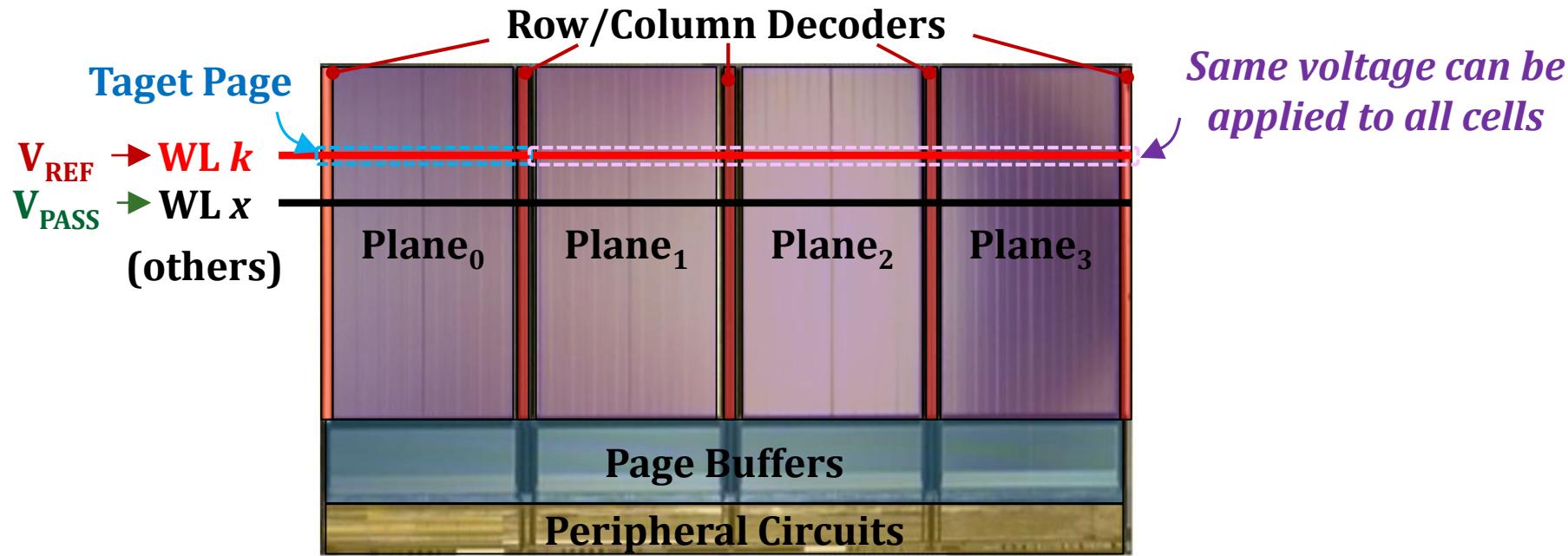


CACHE READ Command: Benefit

- 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

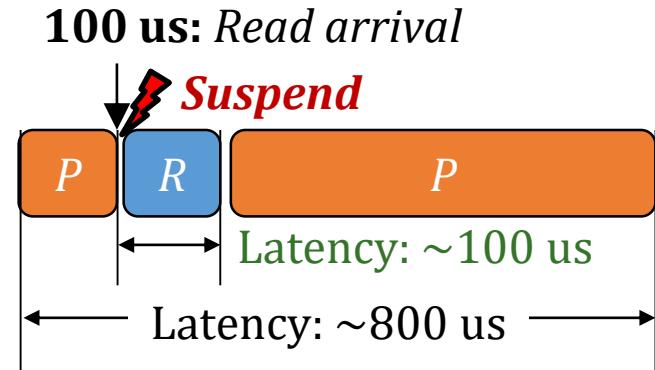
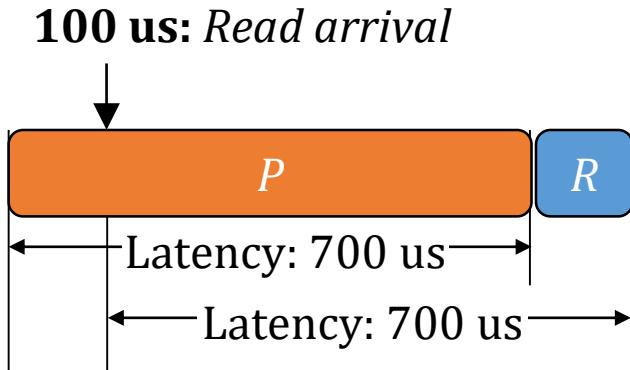
- Increase the throughput/bandwidth almost linearly with # of planes that concurrently operate
 - Bandwidth with regular page programs:
16 KiB / 736 us \approx 22 MB/s
 - Bandwidth with multi-plane page programs (2 planes):
32 KiB / 736 + 16 (tDMA) + 20 (tECC) us \approx 41.5 MB/s
- Per-operation latency increases
 - Regular page program: tECC_{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

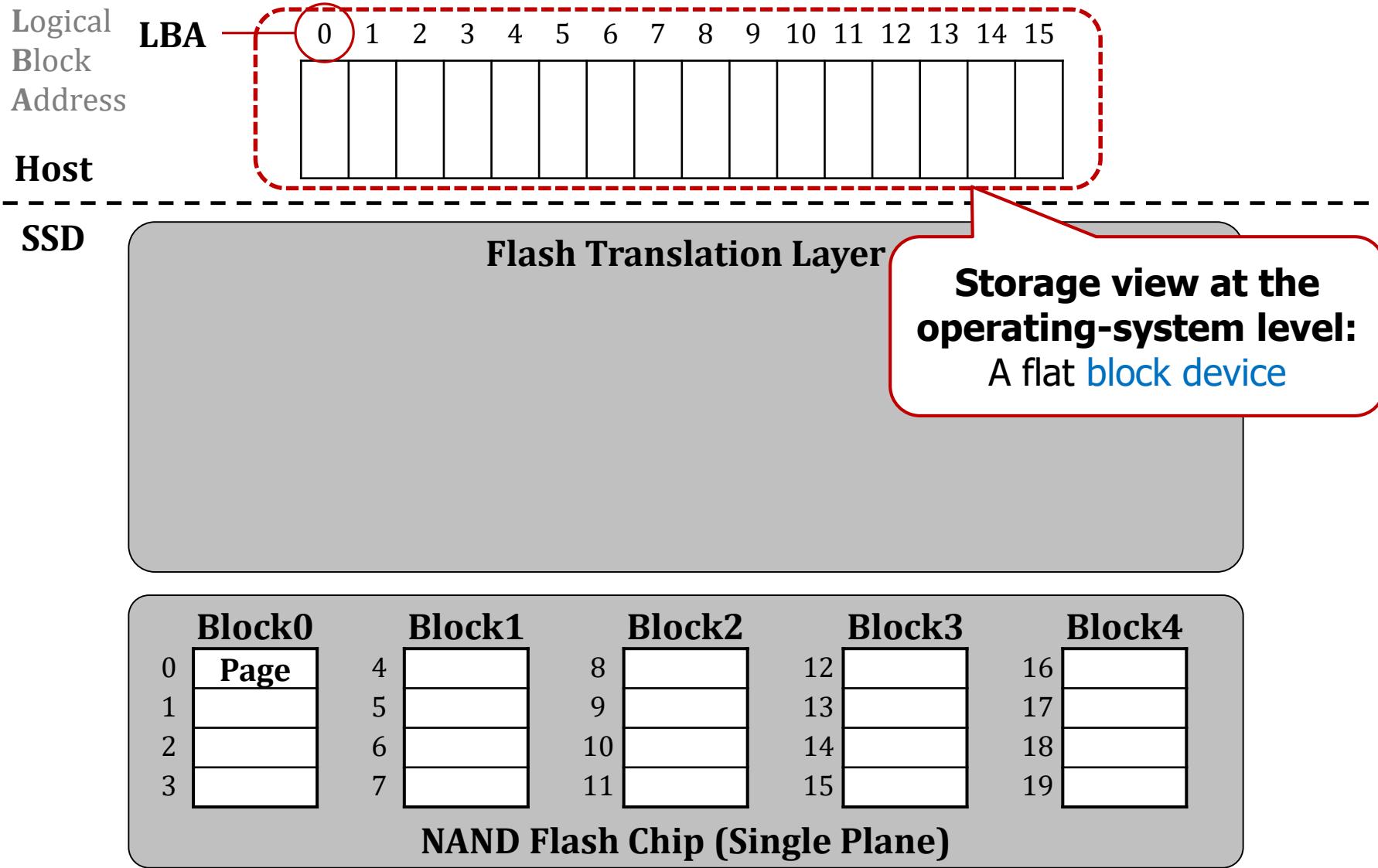
Flash Translation Layer: Overview

- 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**

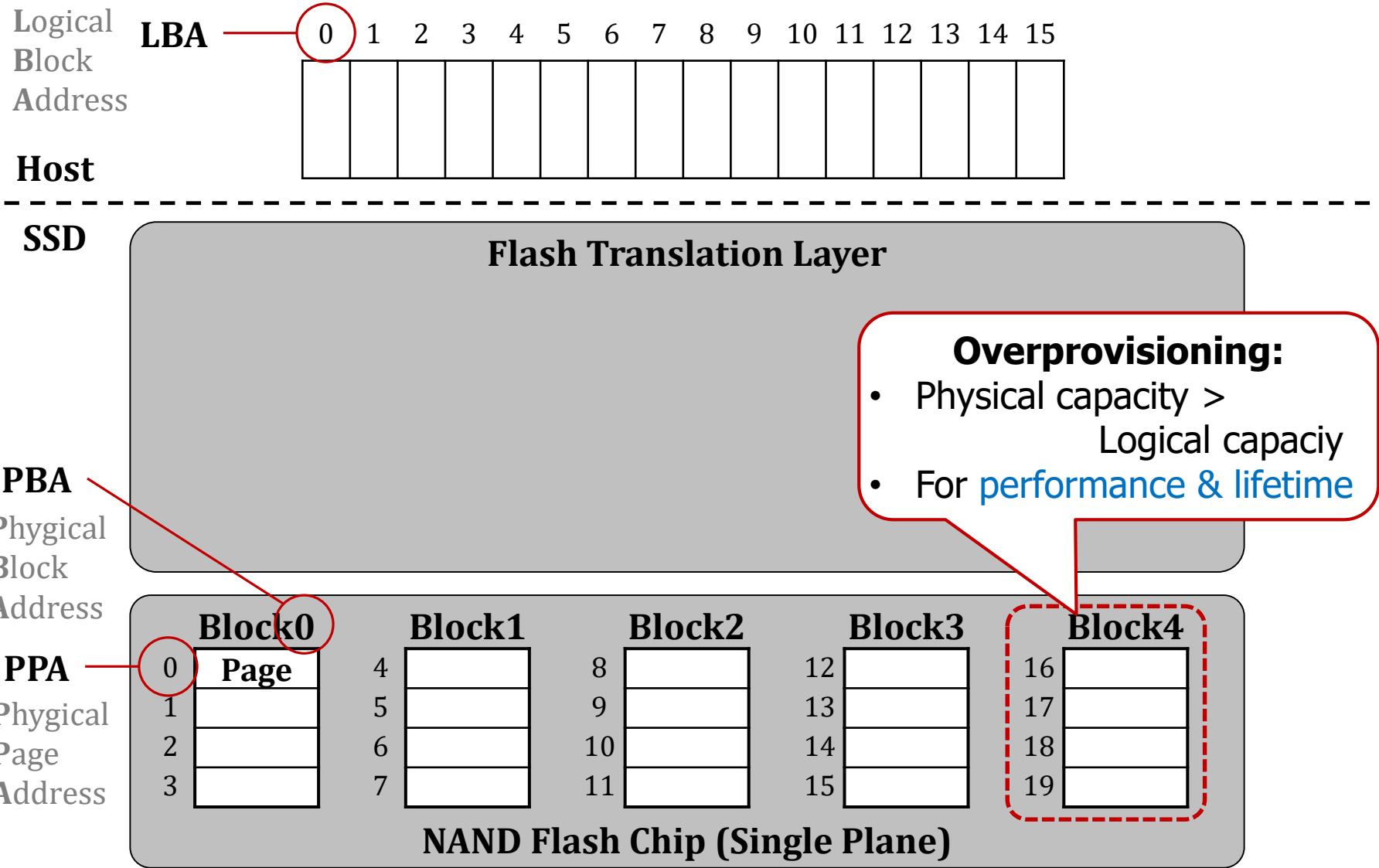
Flash Translation Layer: Overview

- 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**

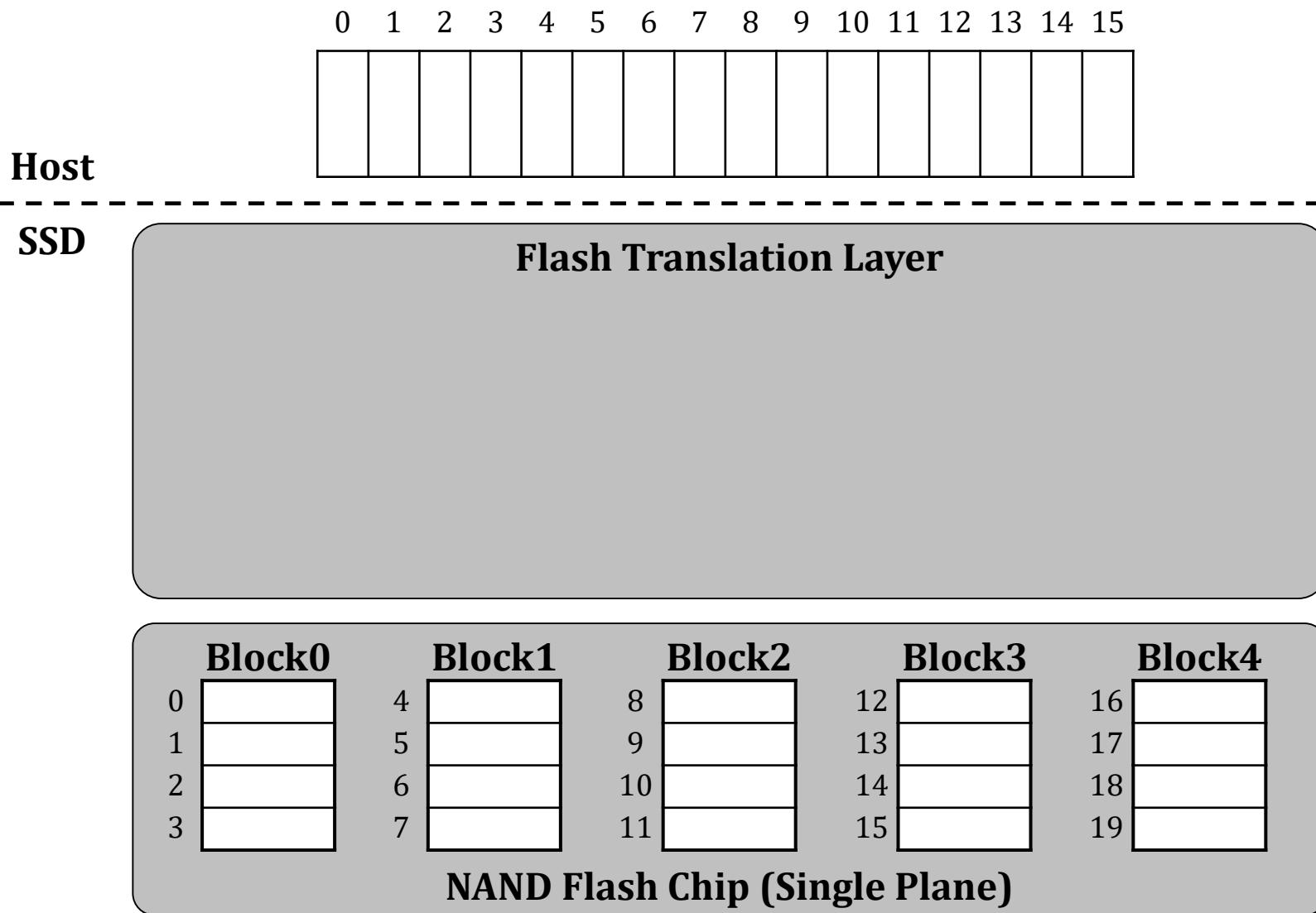
Simple SSD Architecture



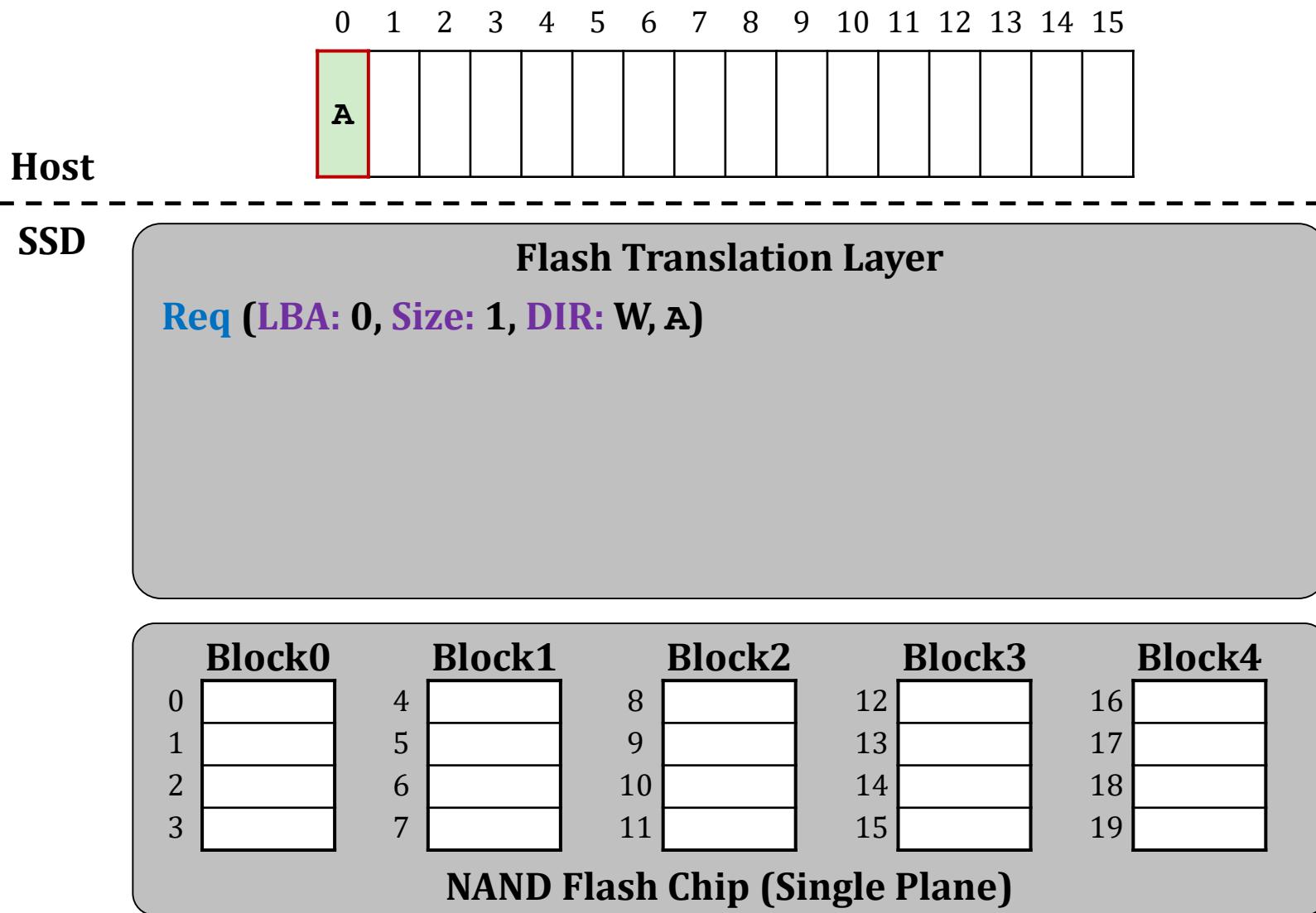
Simple SSD Architecture



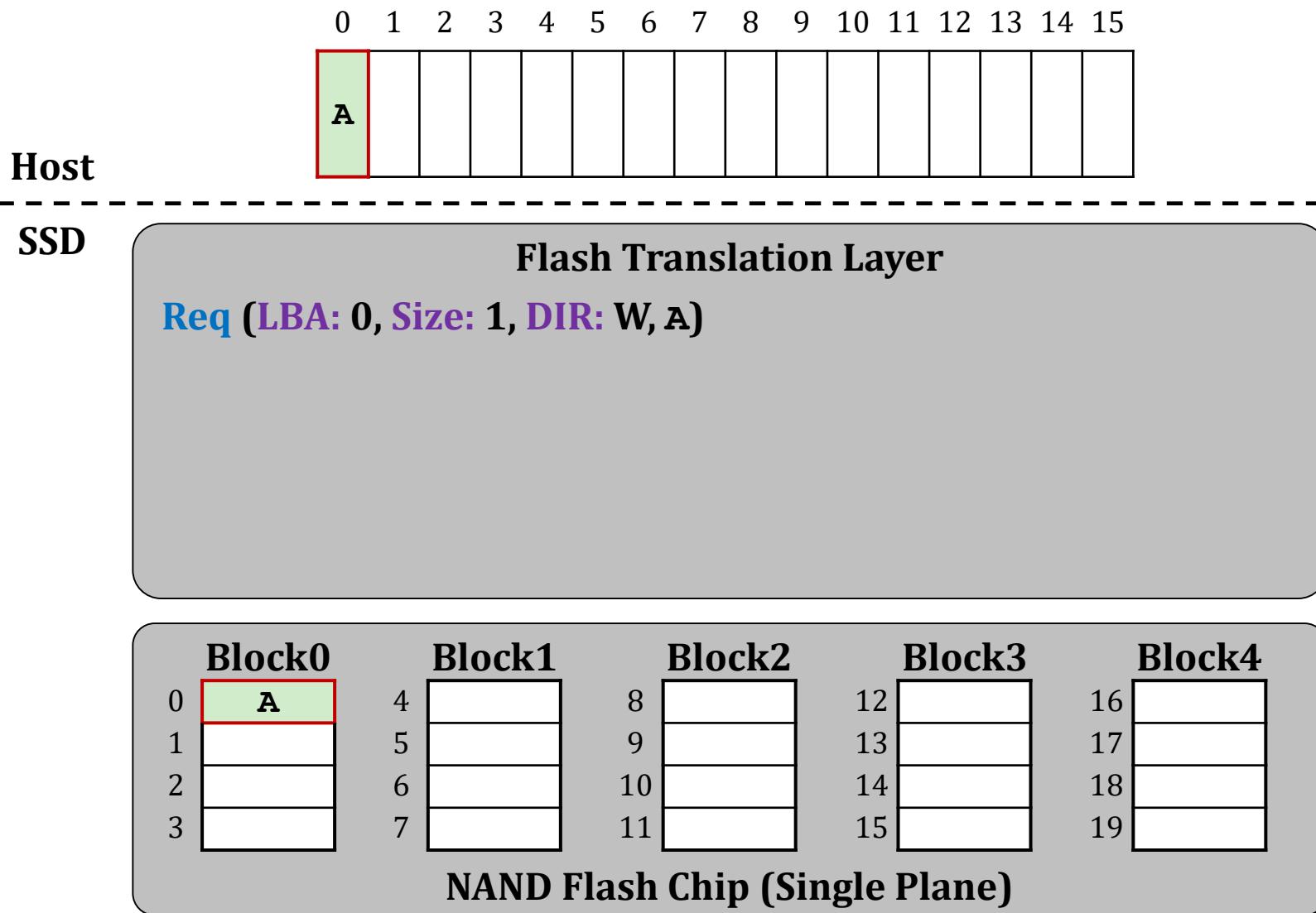
Write Request Handling: Page Write



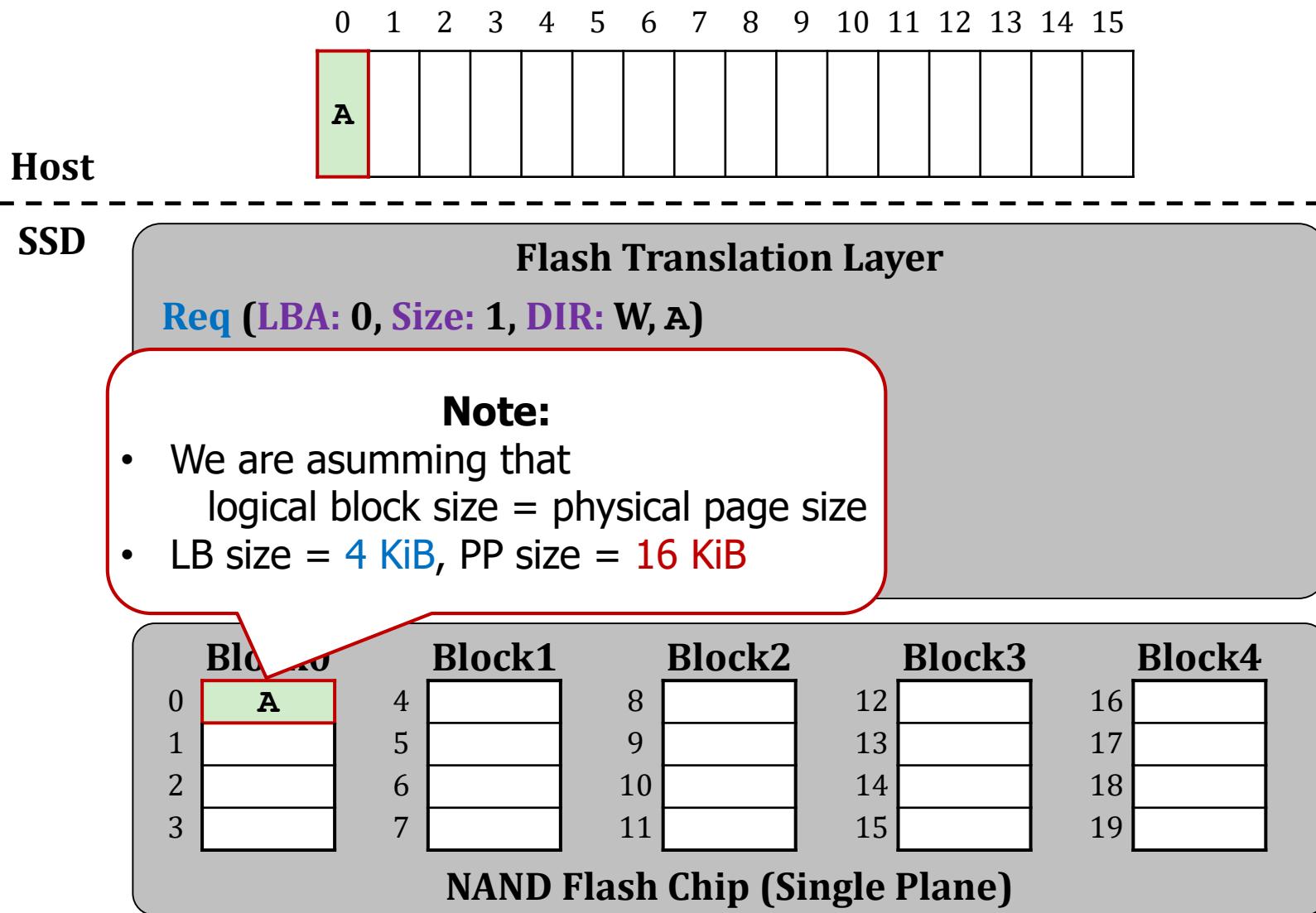
Write Request Handling: Page Write



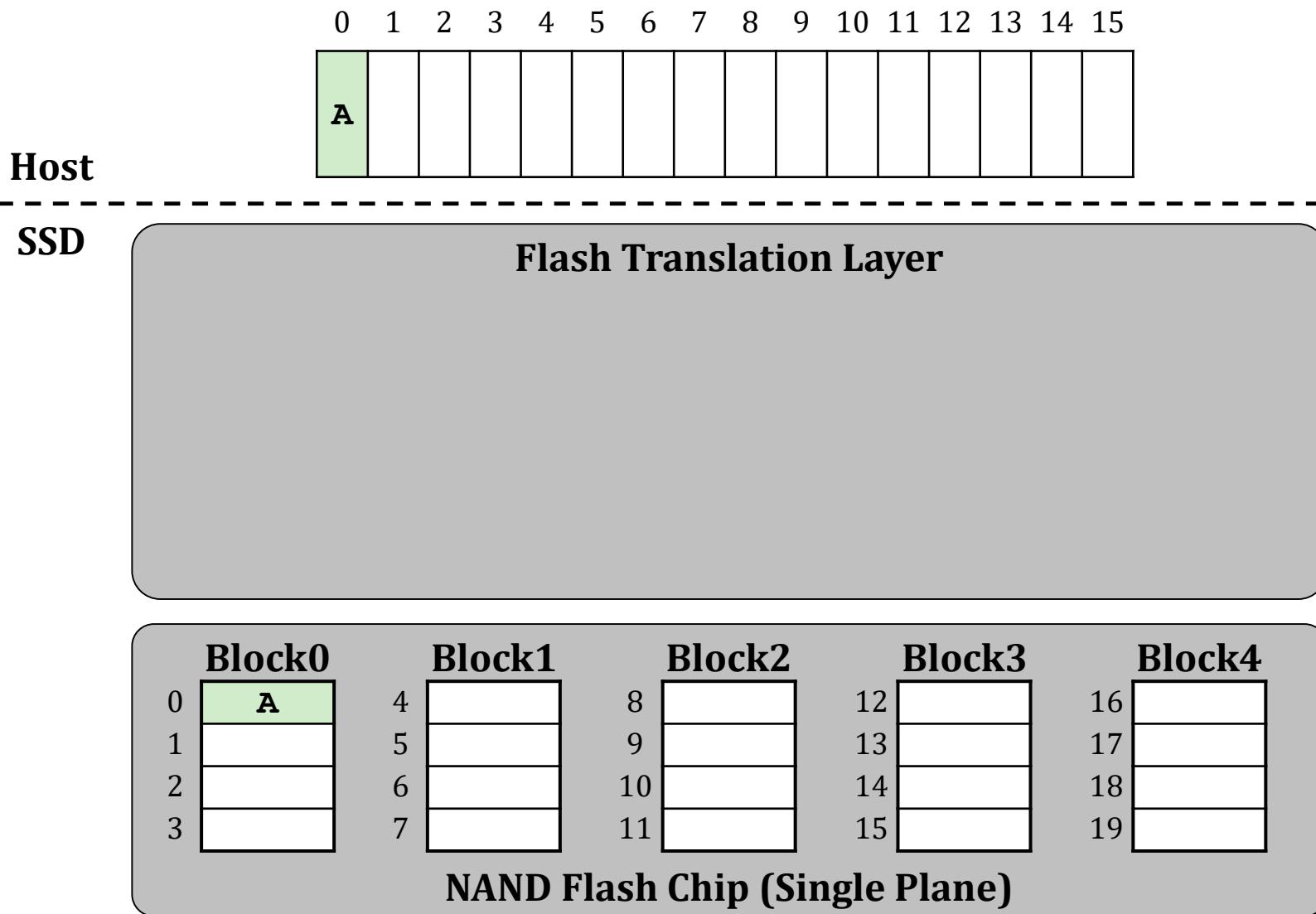
Write Request Handling: Page Write



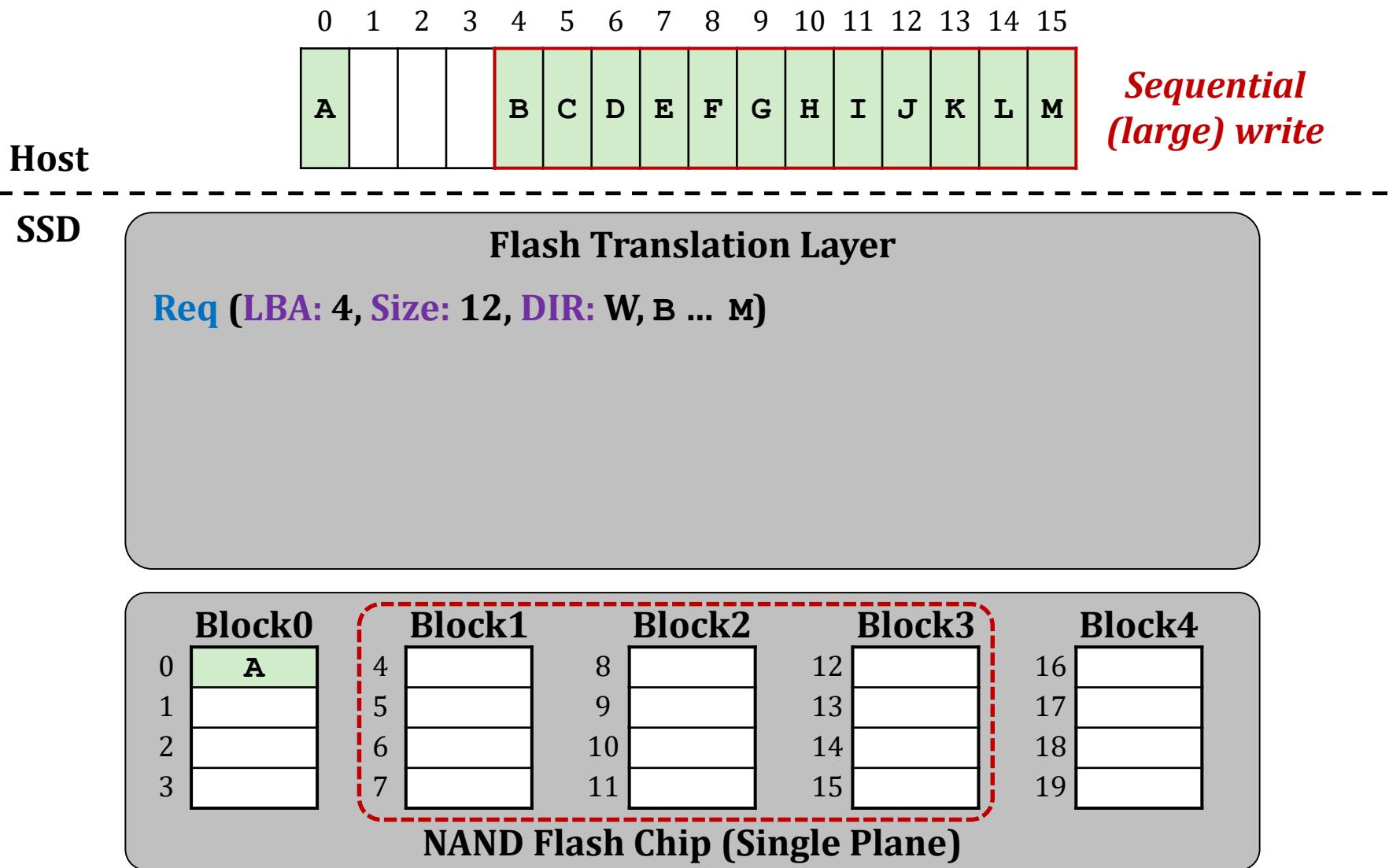
Write Request Handling: Page Write



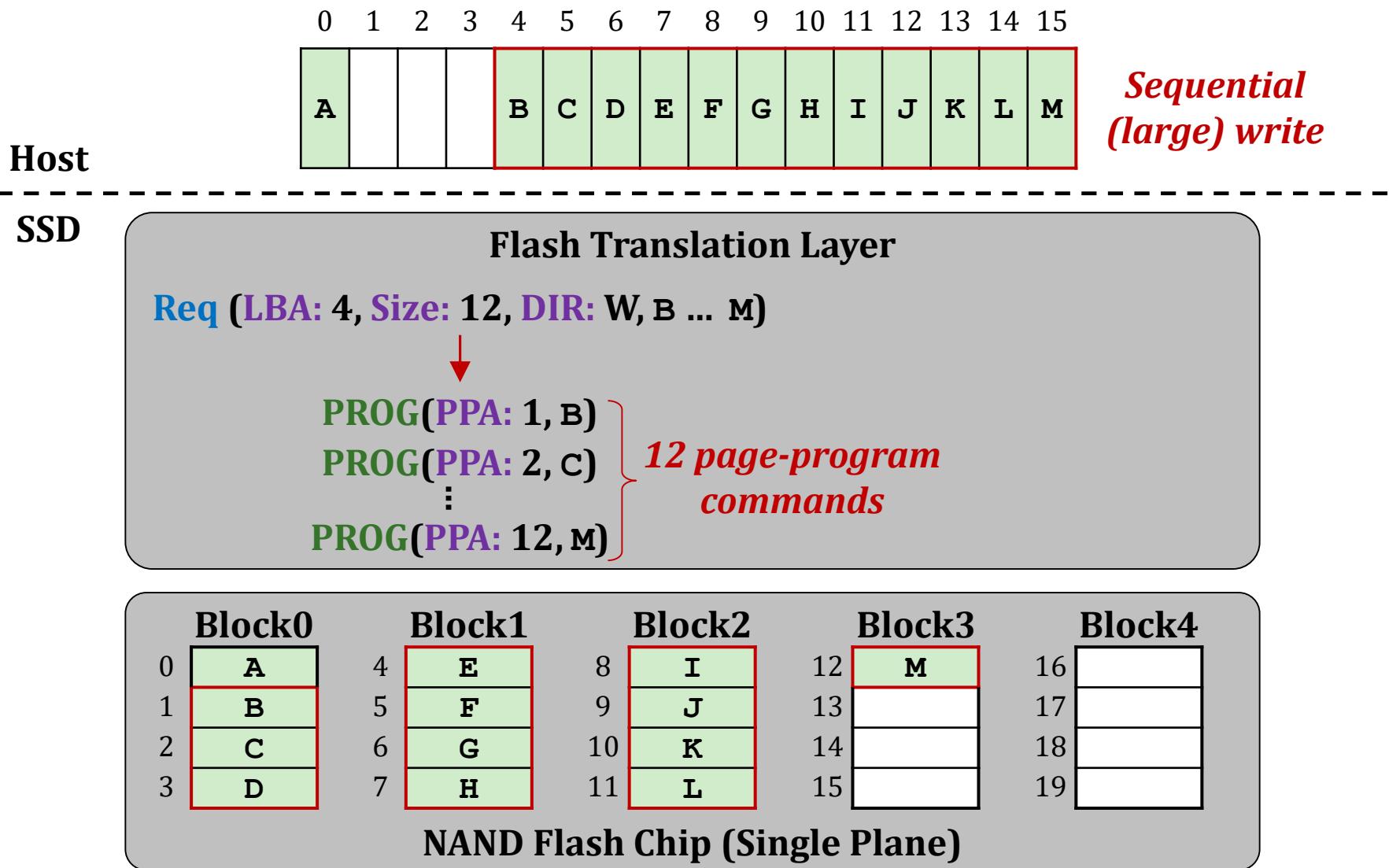
Write Request Handling: Sequential Write



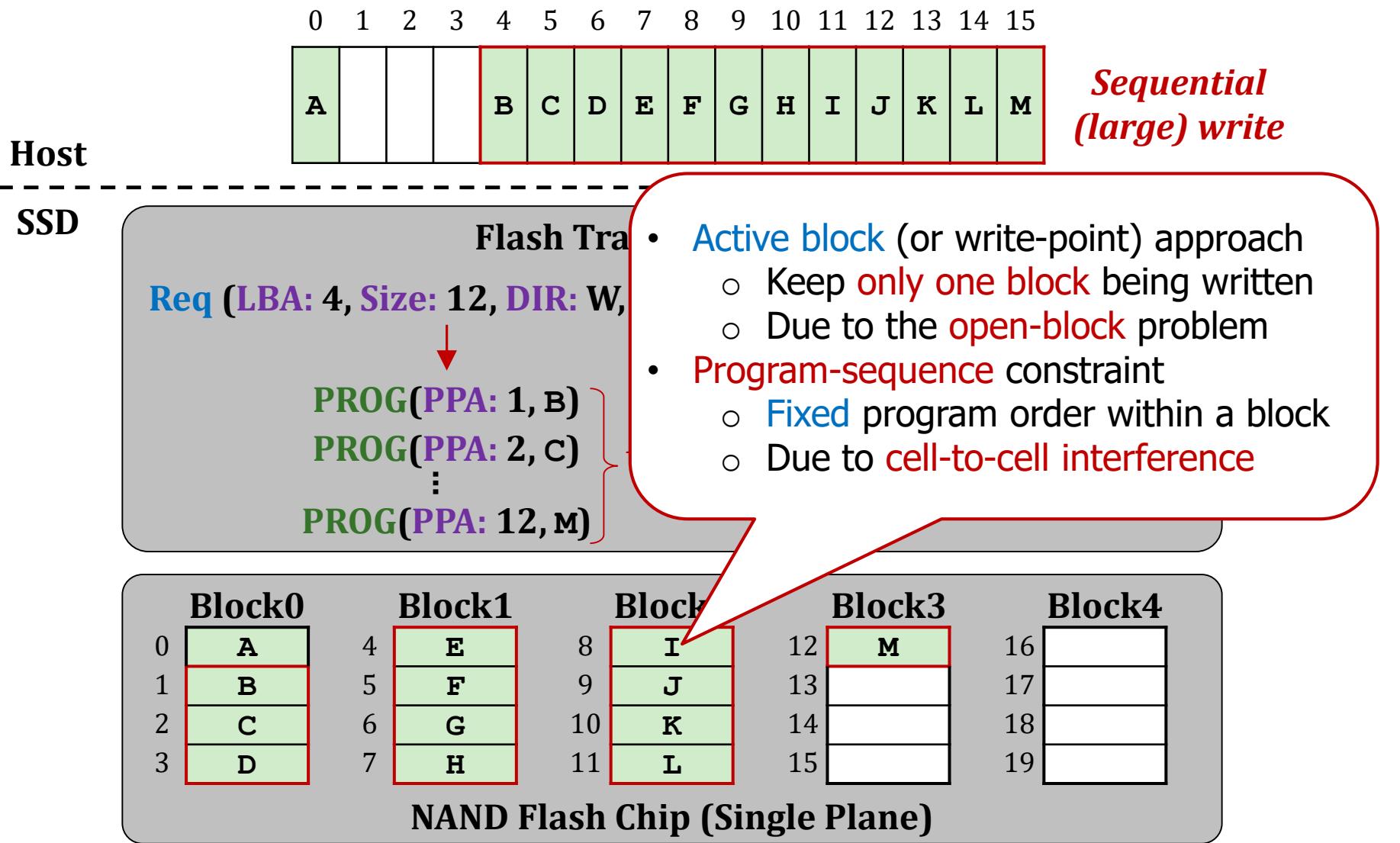
Write Request Handling: Sequential Write



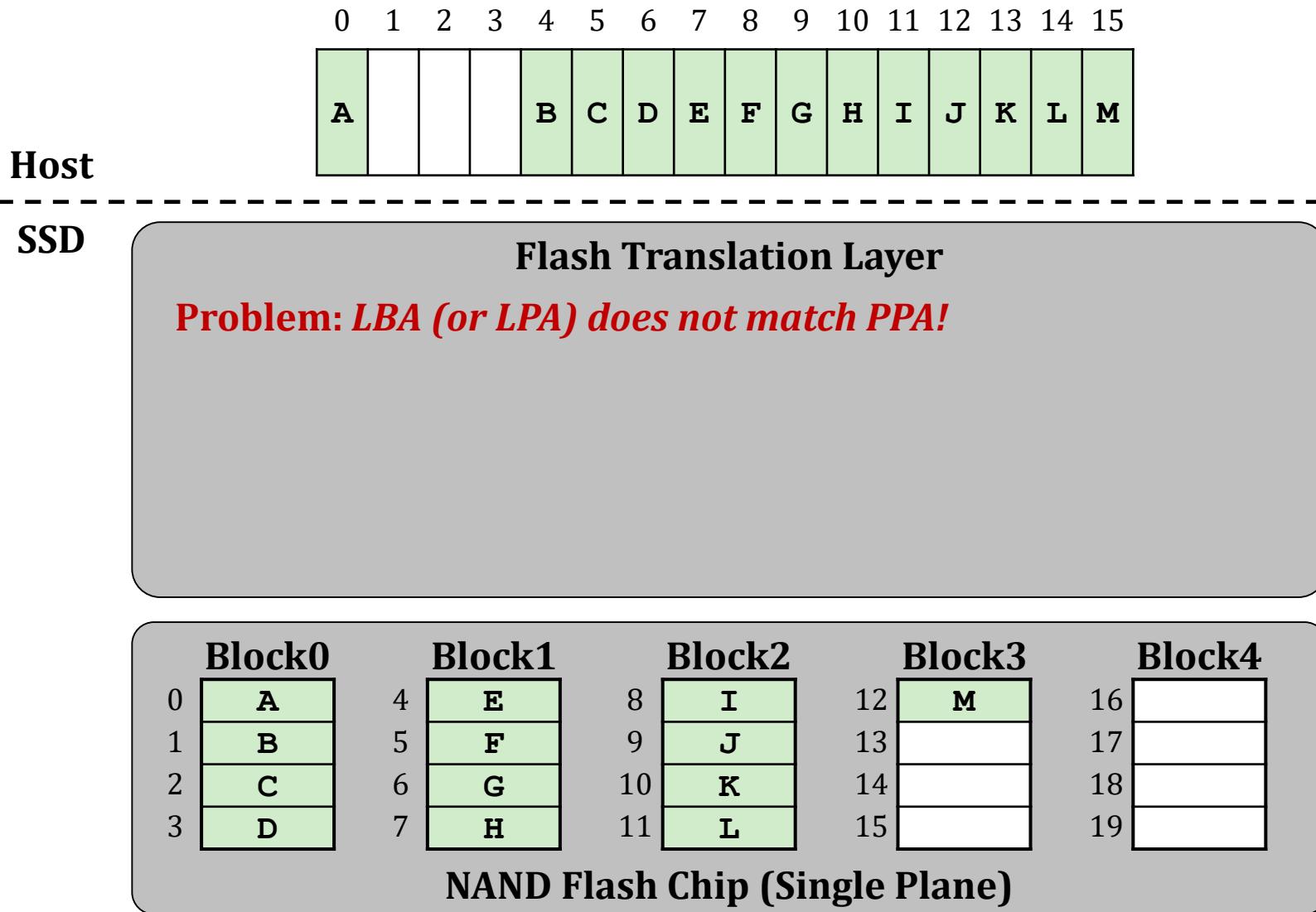
Write Request Handling: Sequential Write



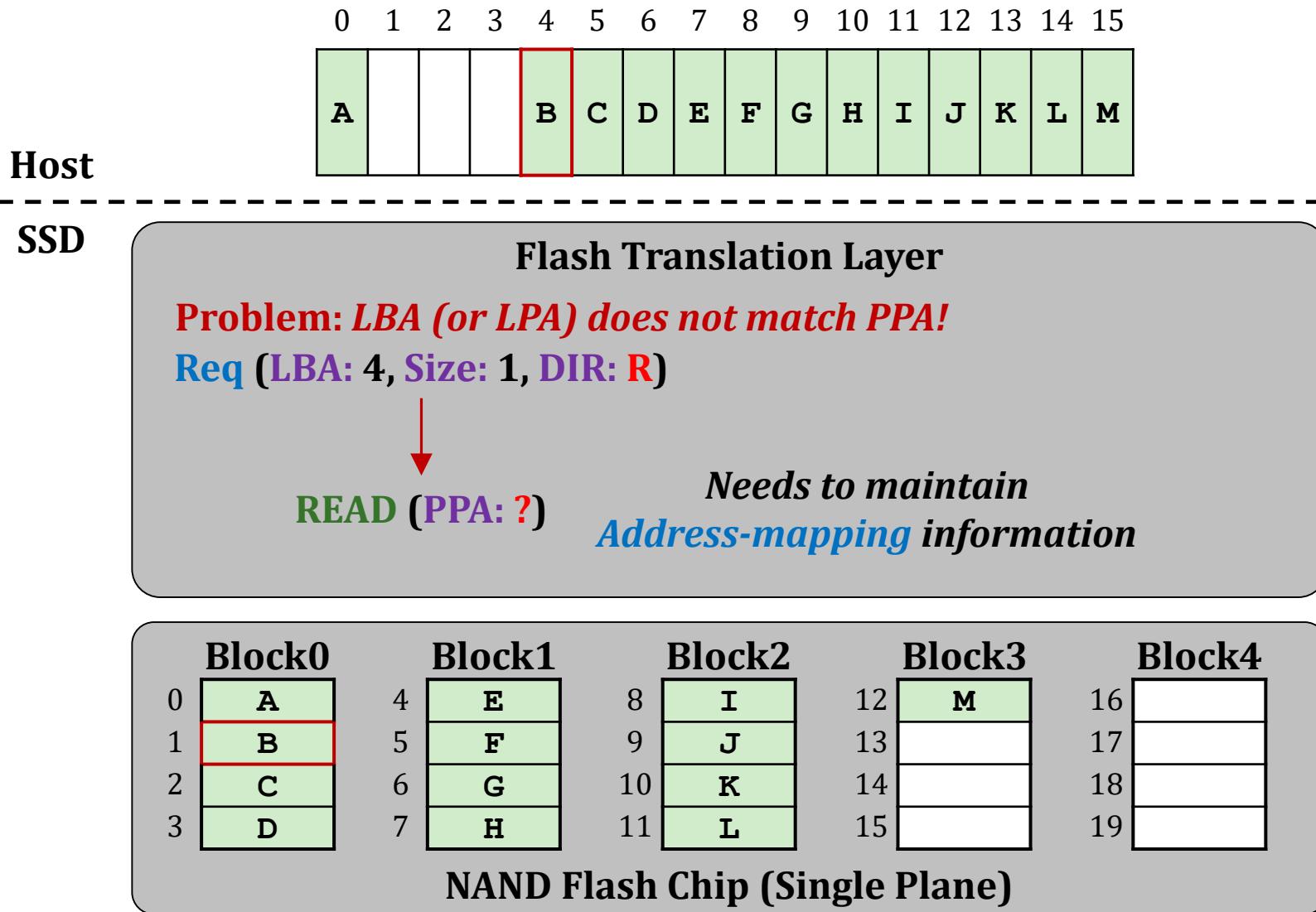
Write Request Handling: Sequential Write



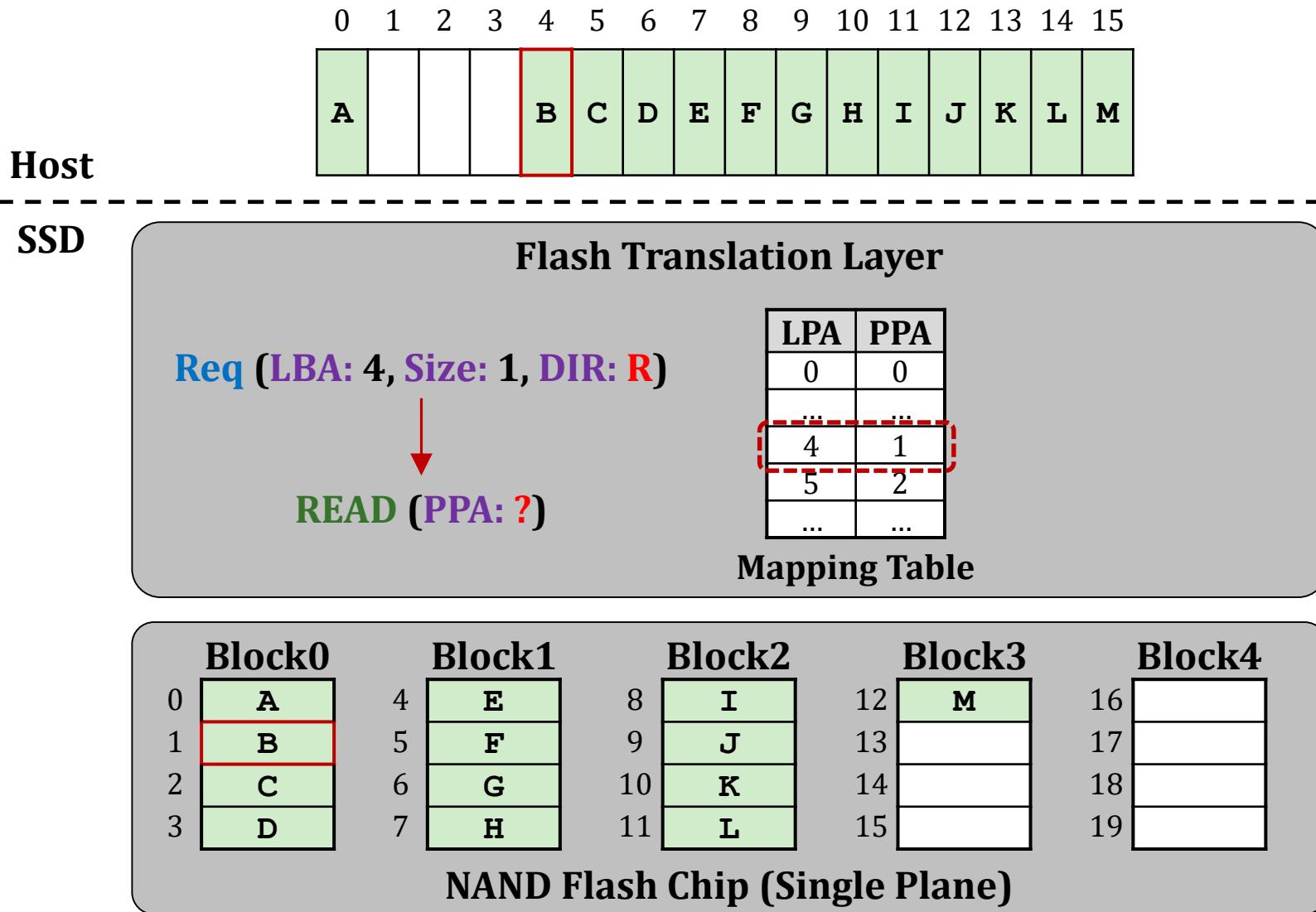
Write Request Handling: Address Mapping



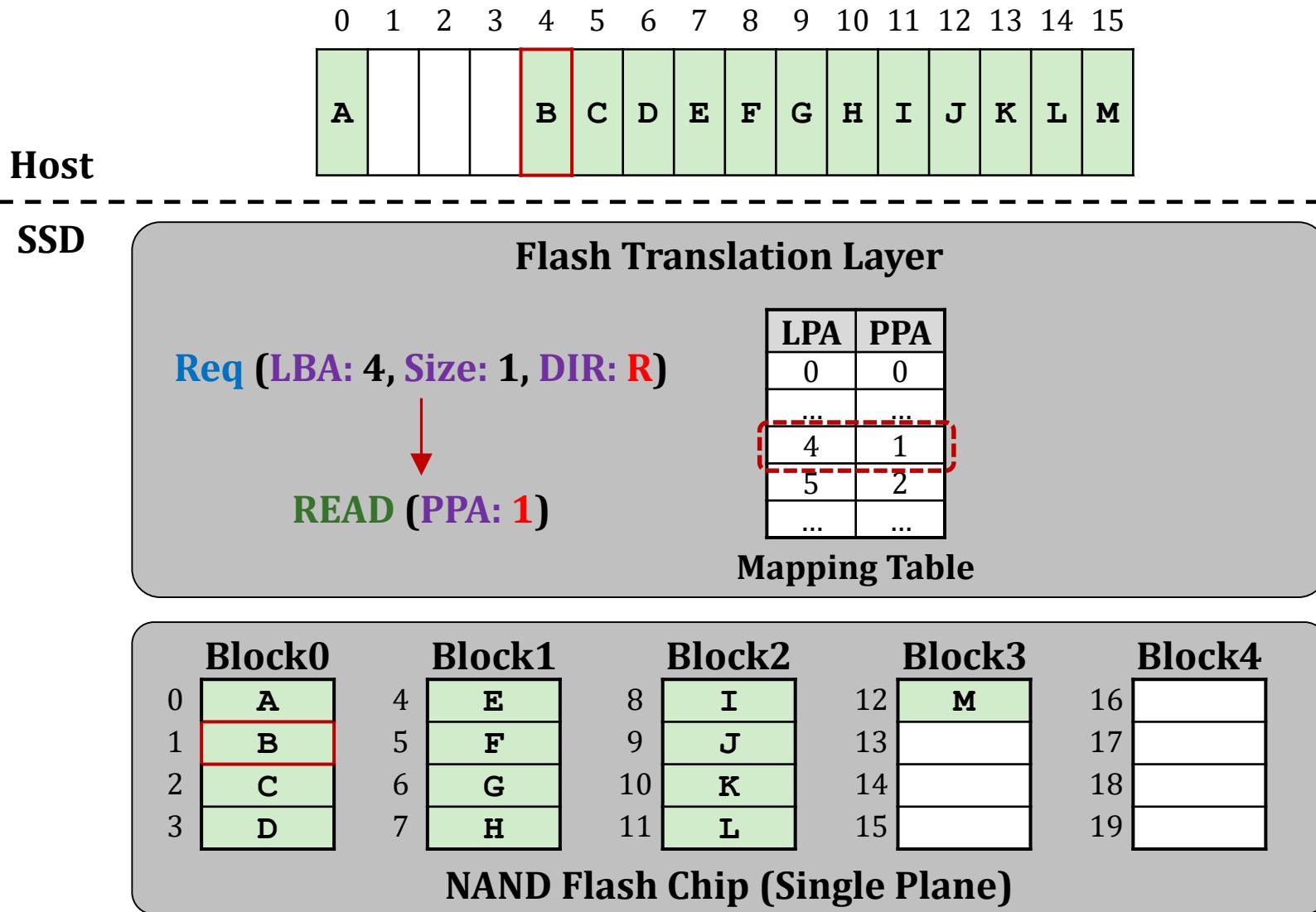
Write Request Handling: Address Mapping



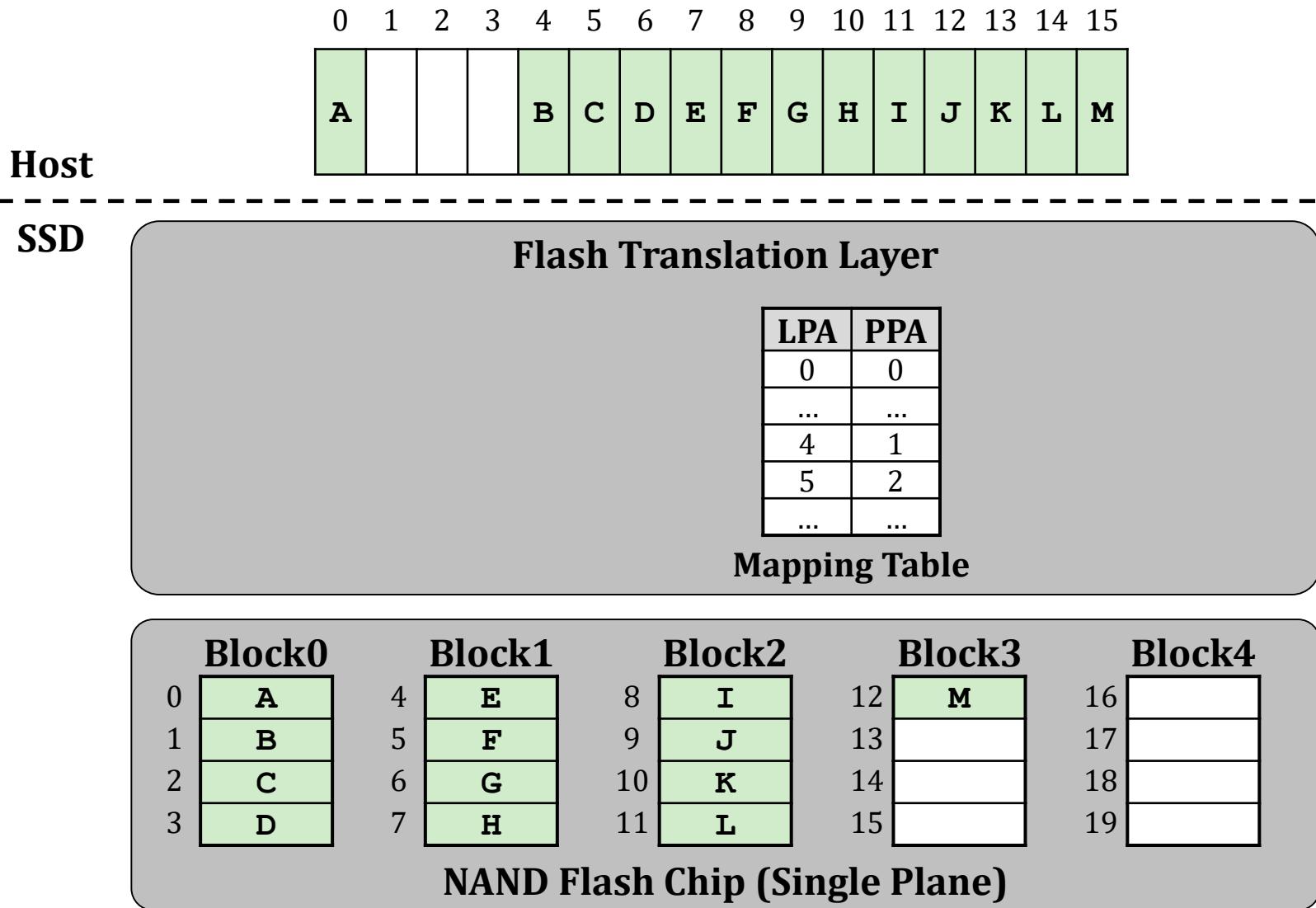
Write Request Handling: Address Mapping



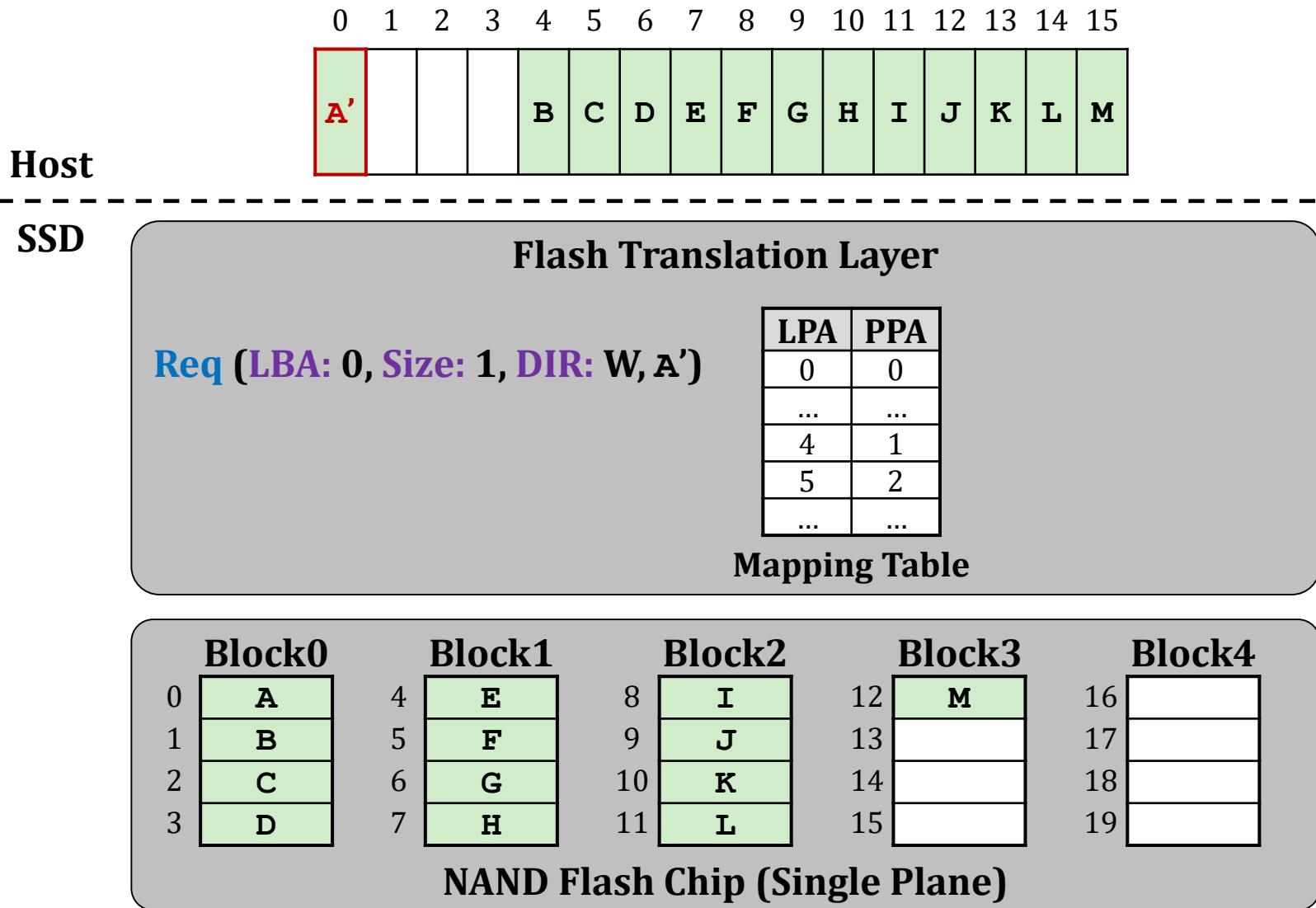
Write Request Handling: Address Mapping



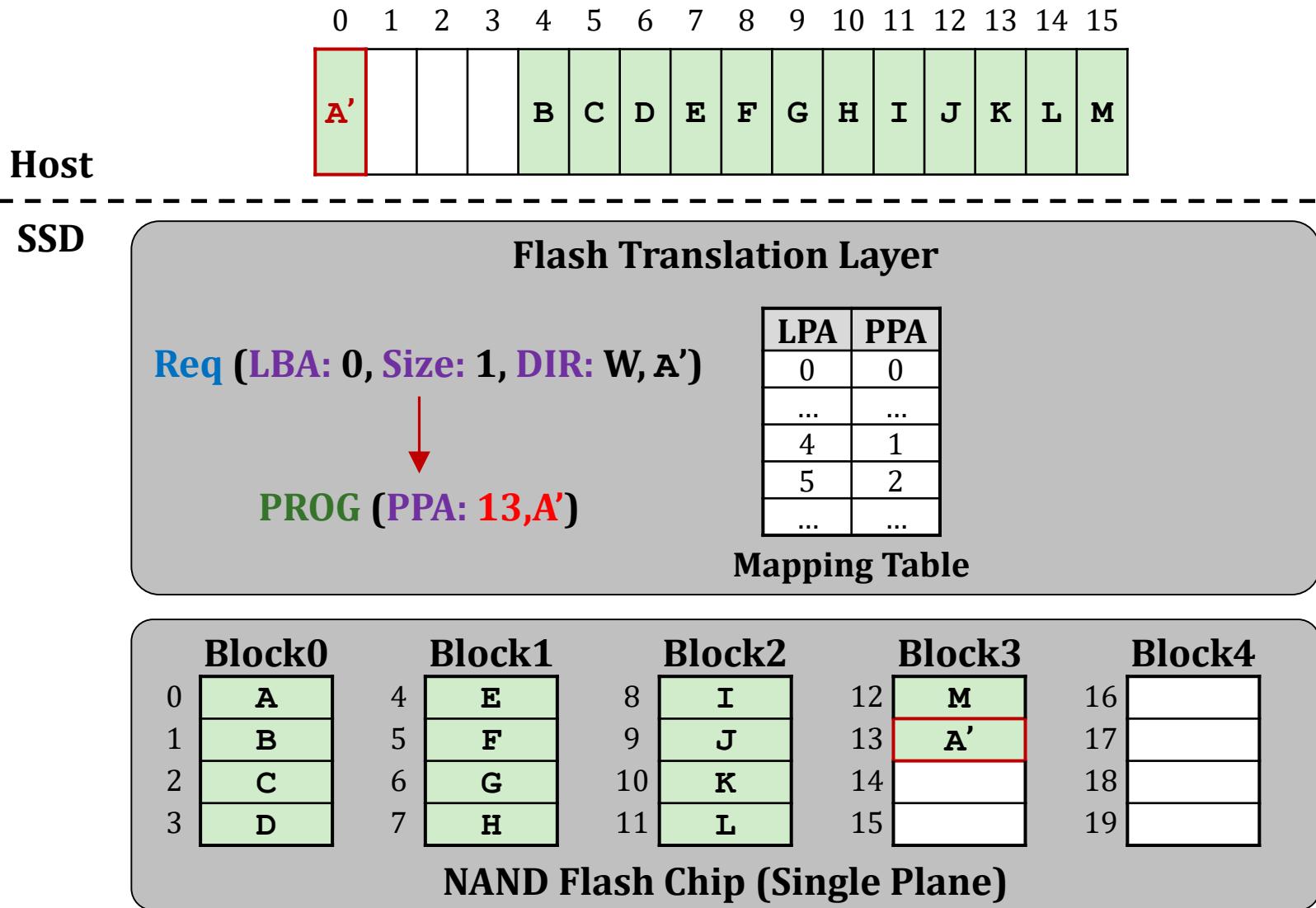
Write Request Handling: Update



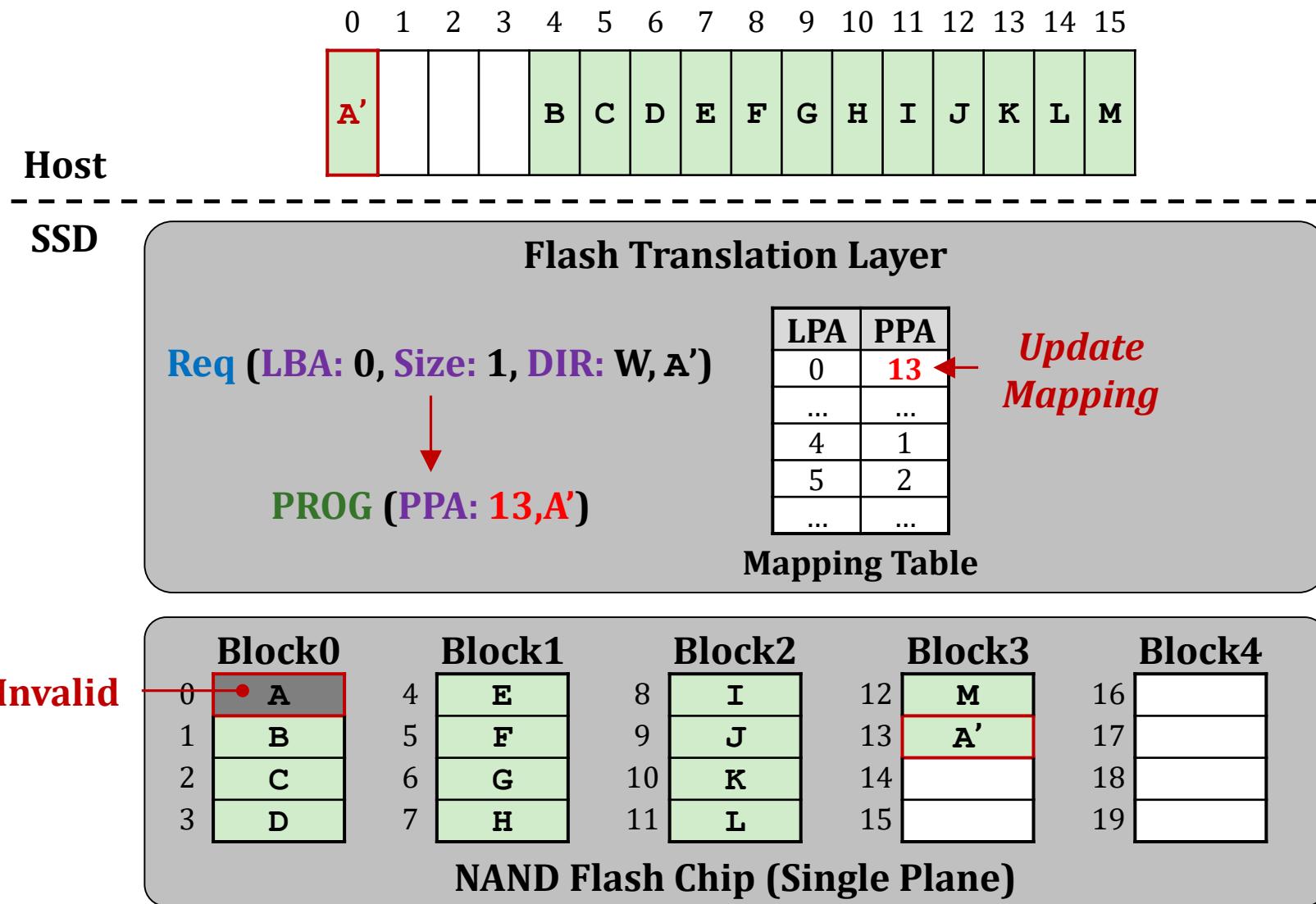
Write Request Handling: Update



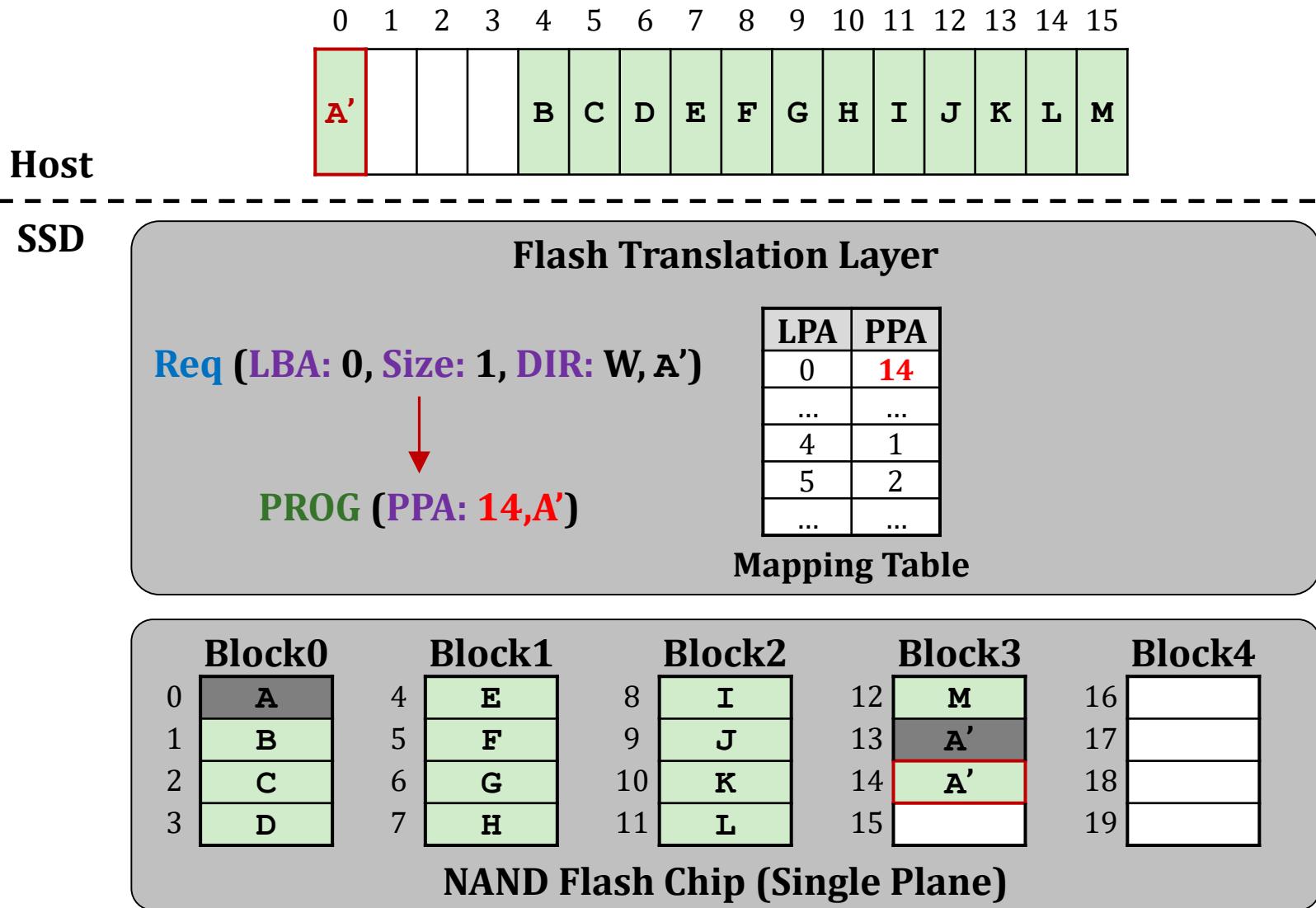
Write Request Handling: Update



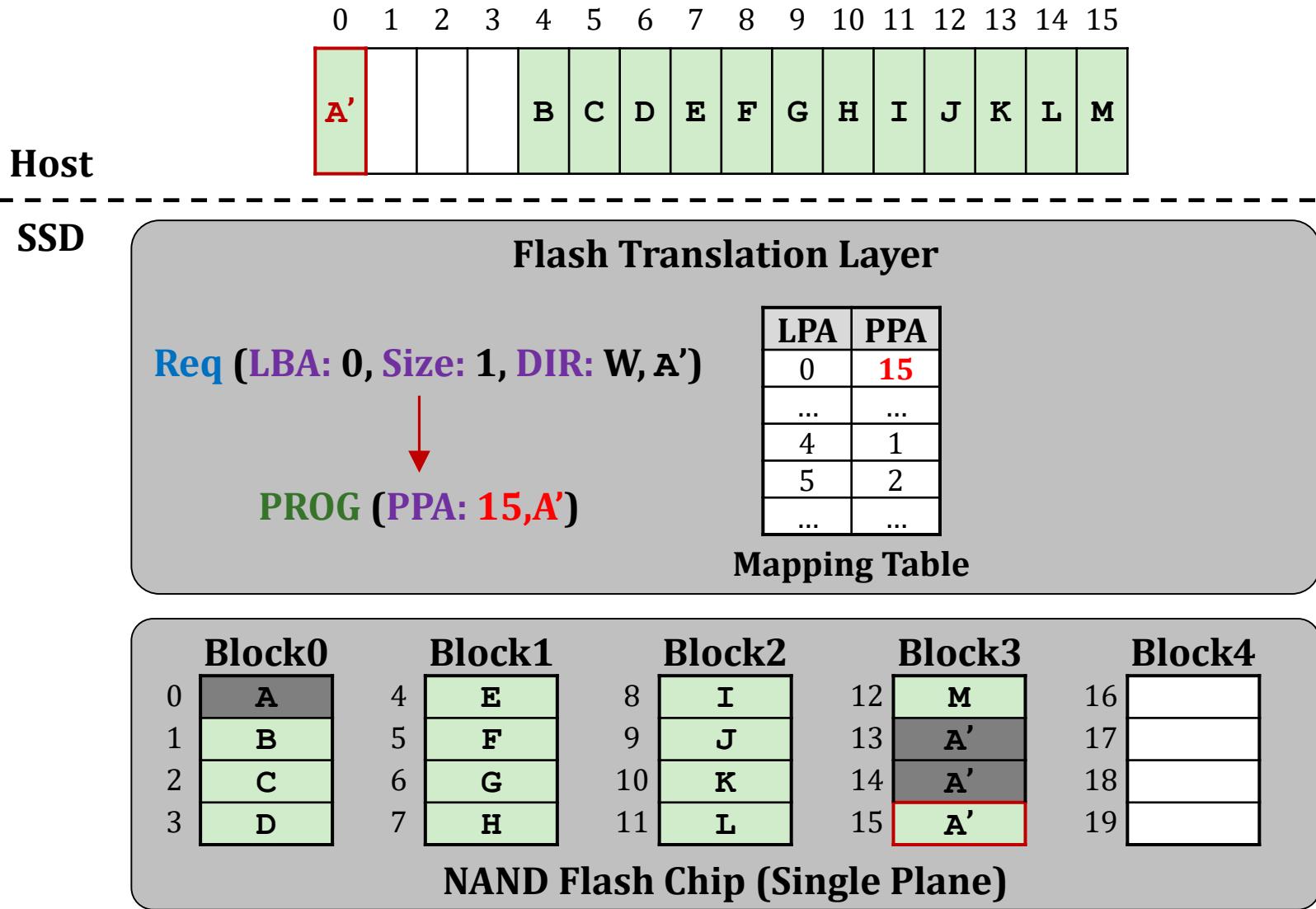
Write Request Handling: Update



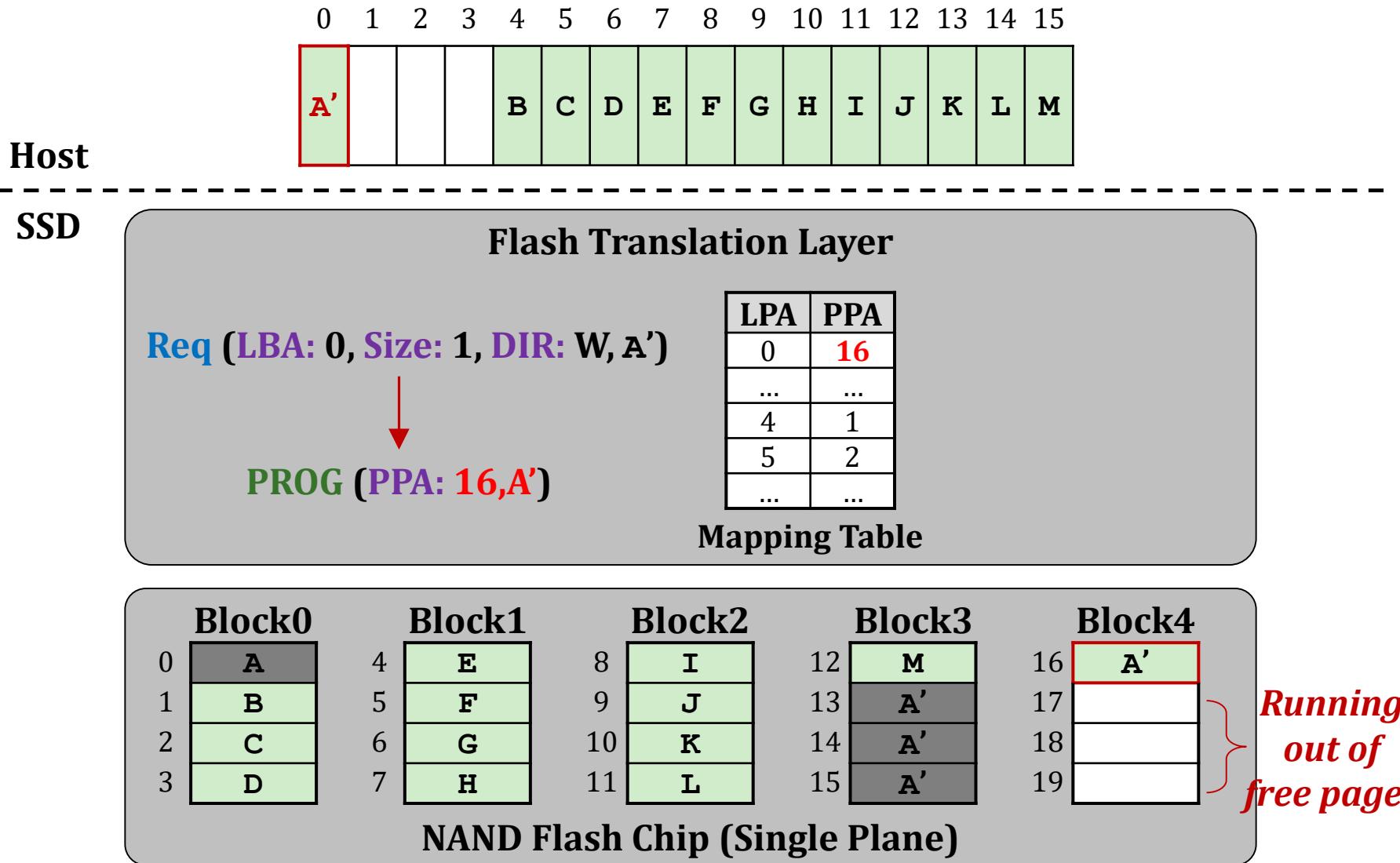
Write Request Handling: Update



Write Request Handling: Update



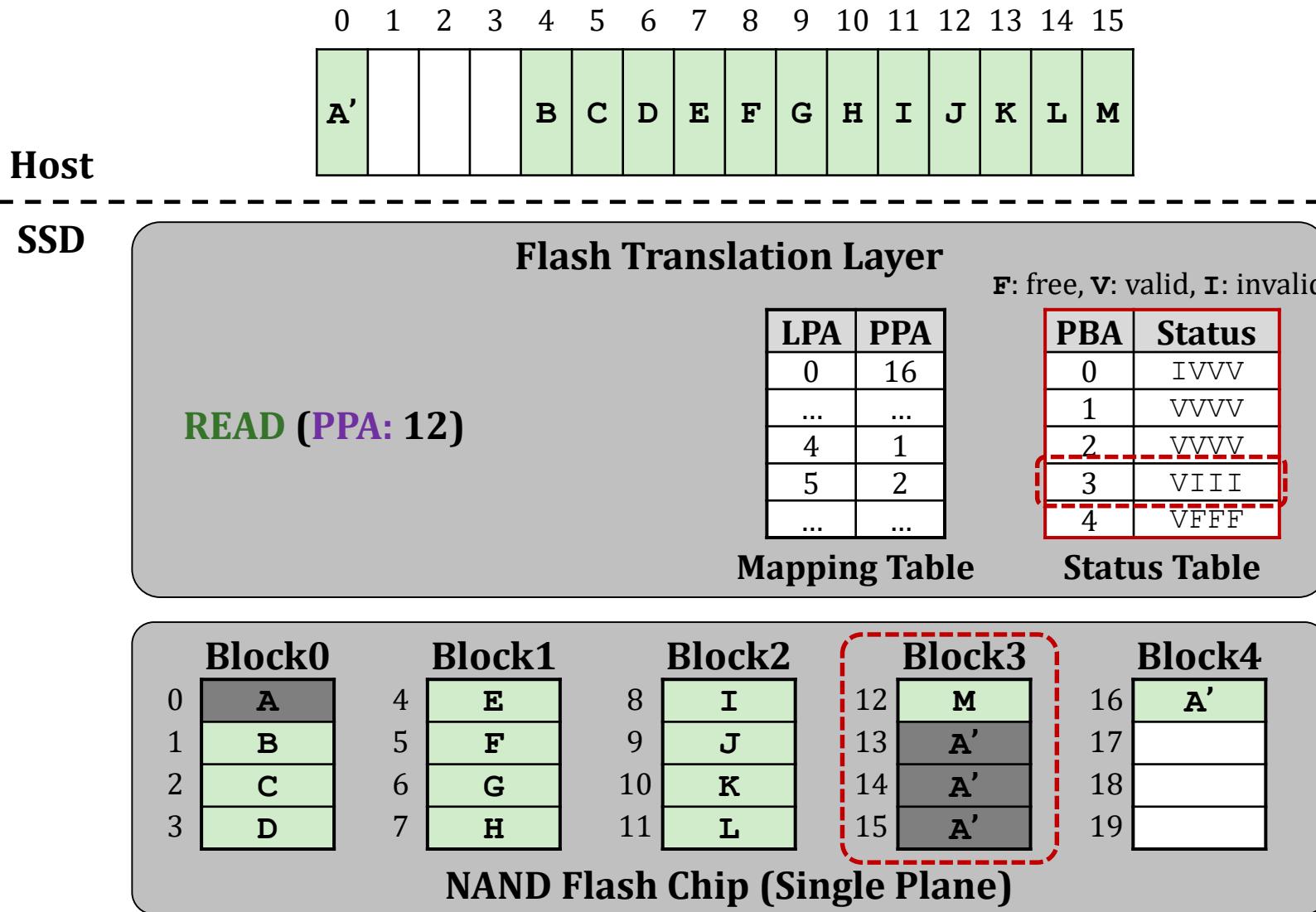
Write Request Handling: Update



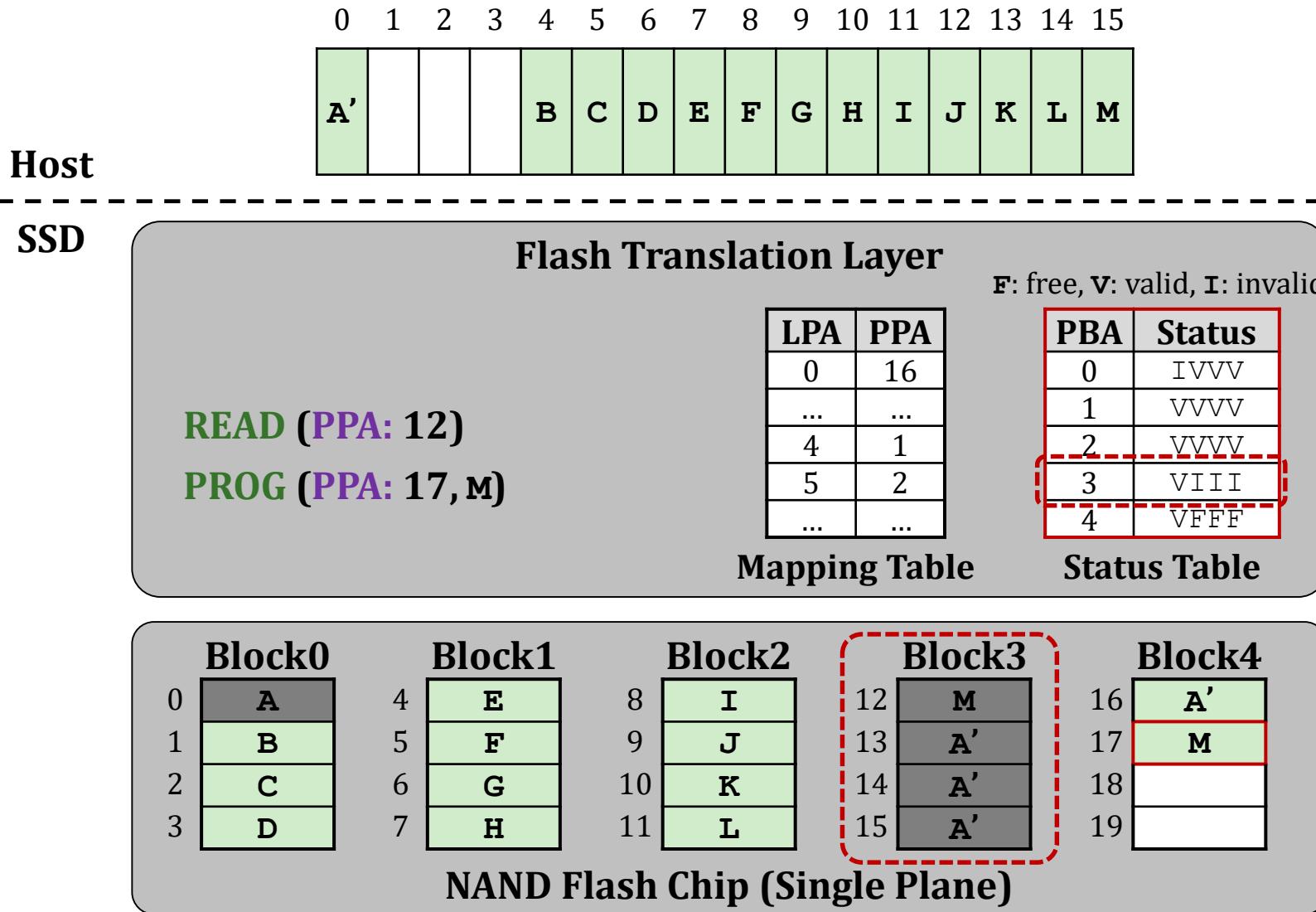
Garbage Collection

- 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 → 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

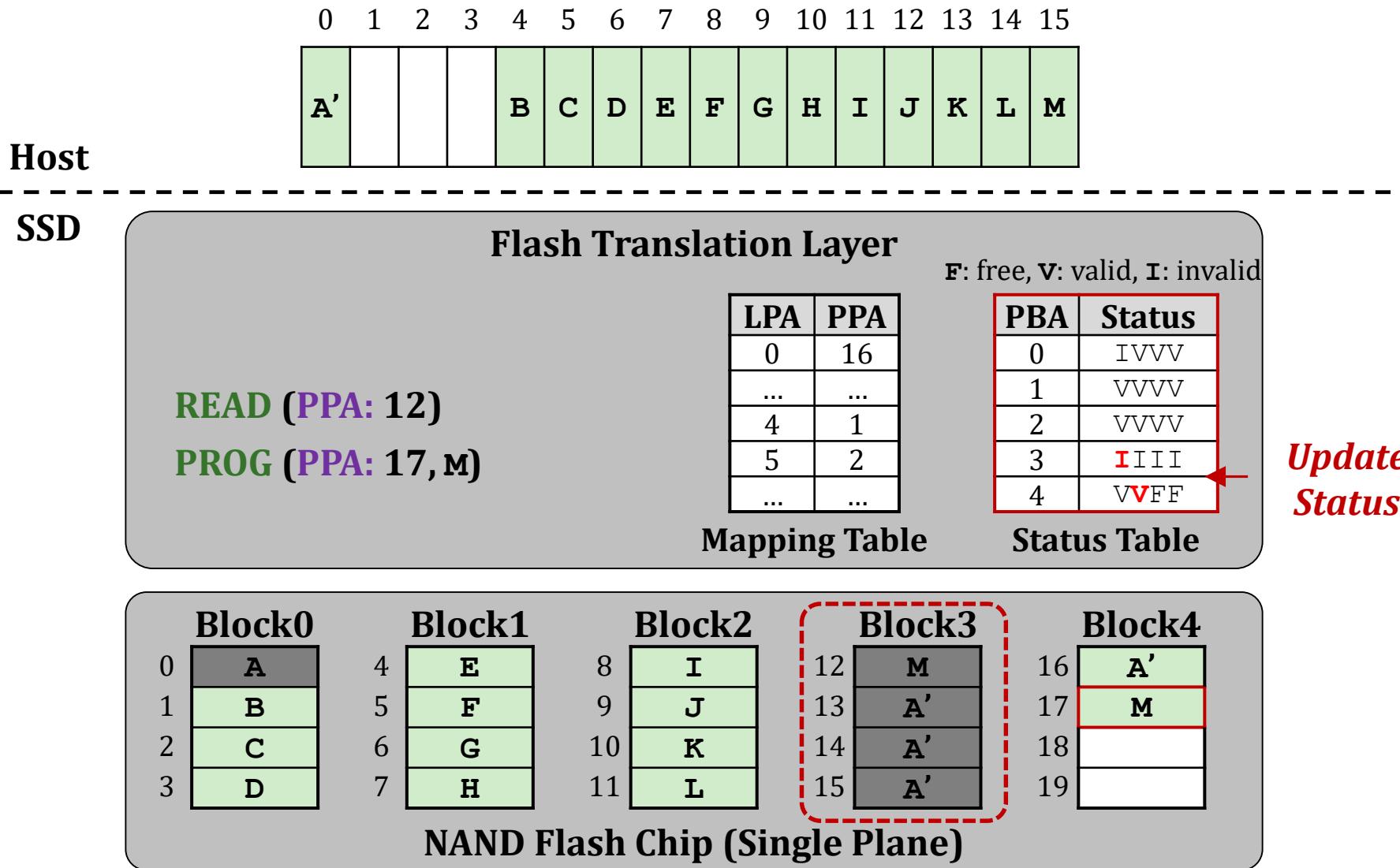
Write Request Handling: Garbage Collection



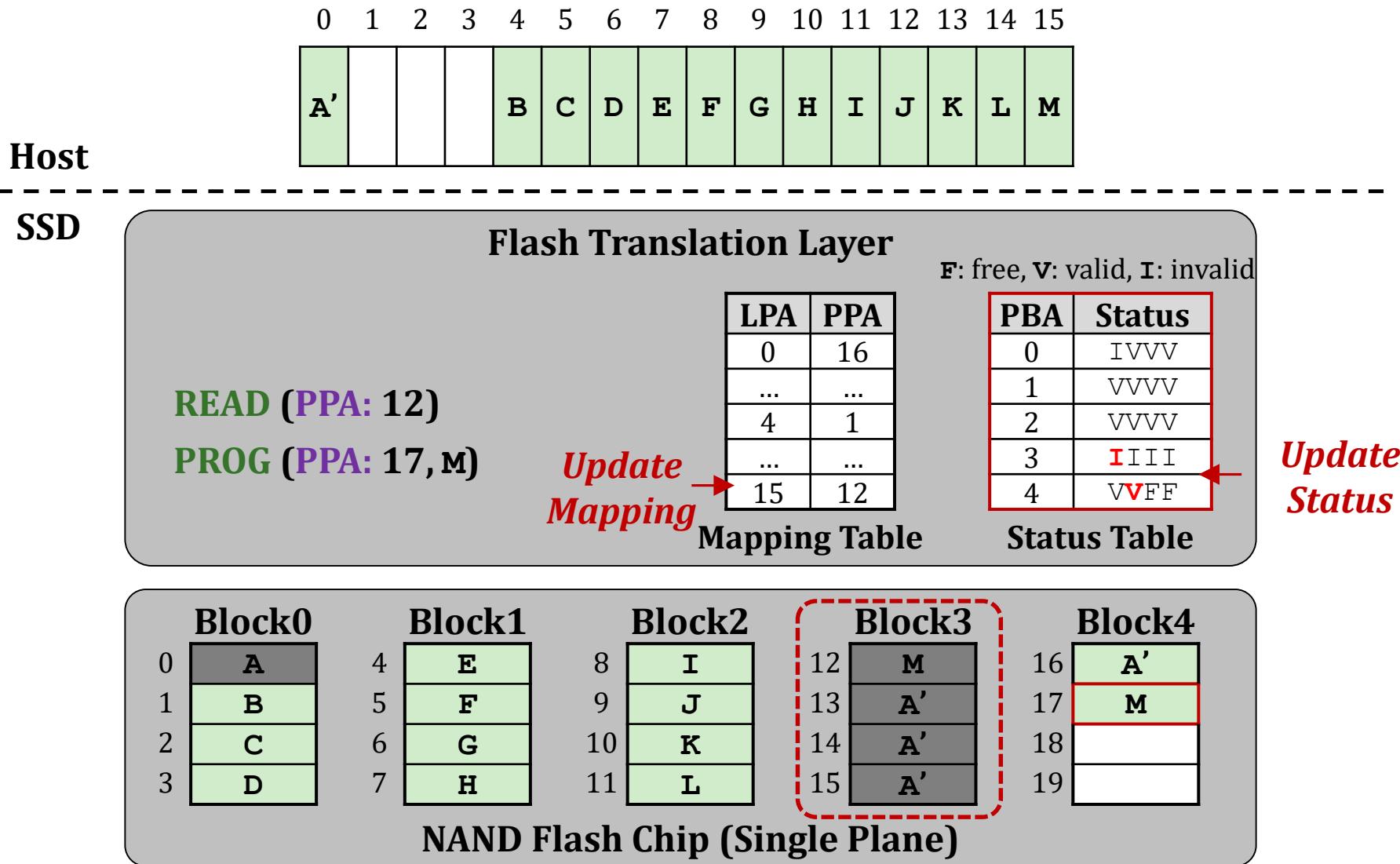
Write Request Handling: Garbage Collection



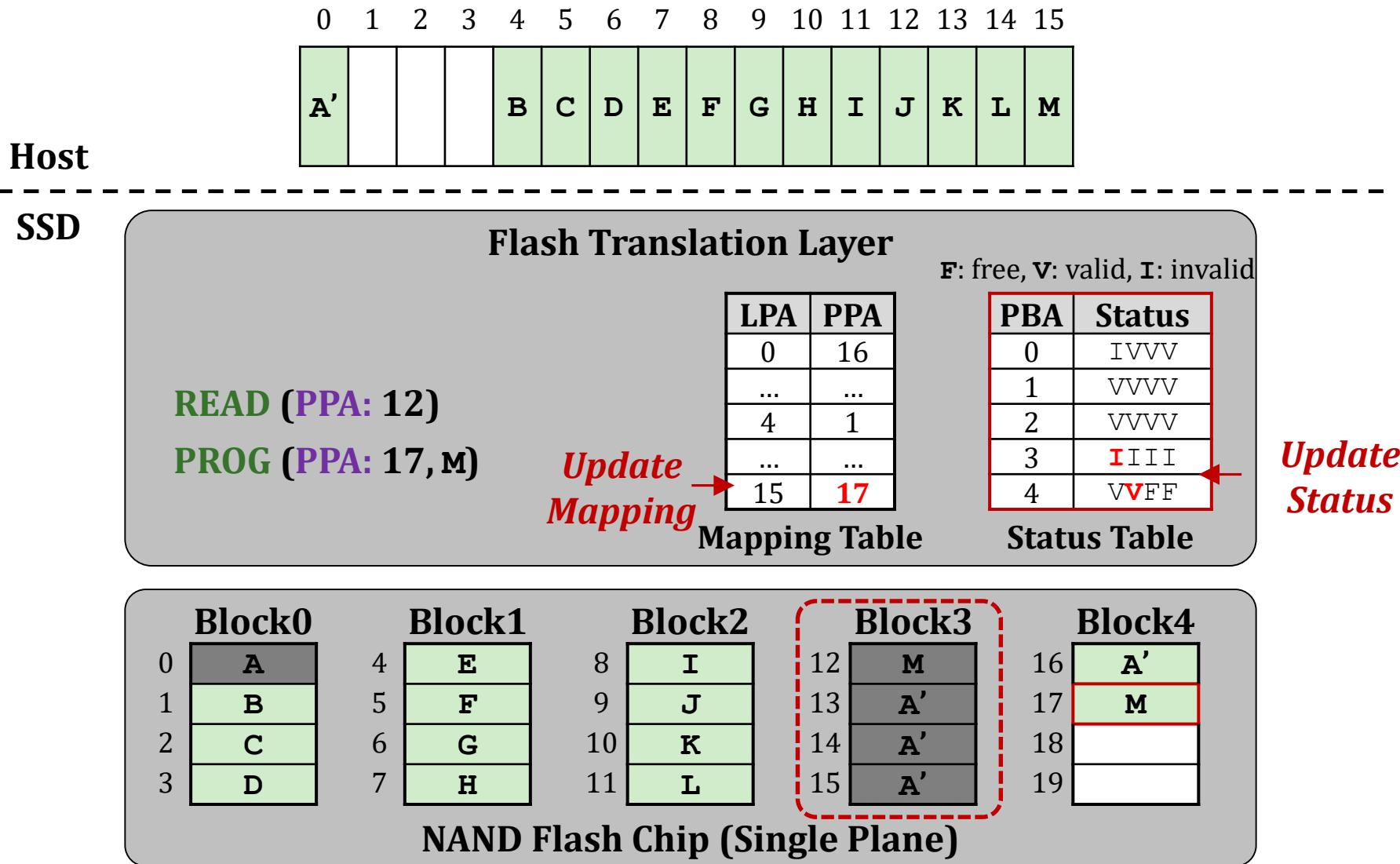
Write Request Handling: Garbage Collection



Write Request Handling: Garbage Collection

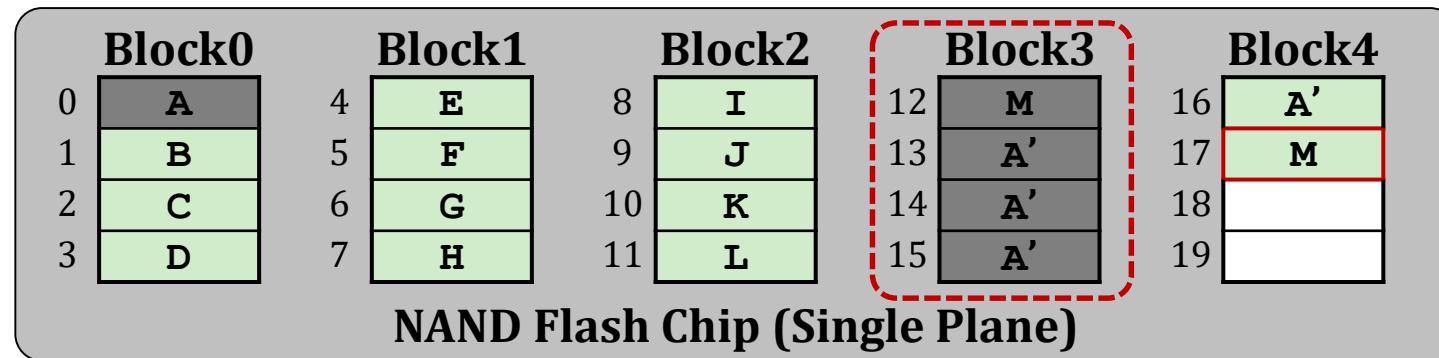
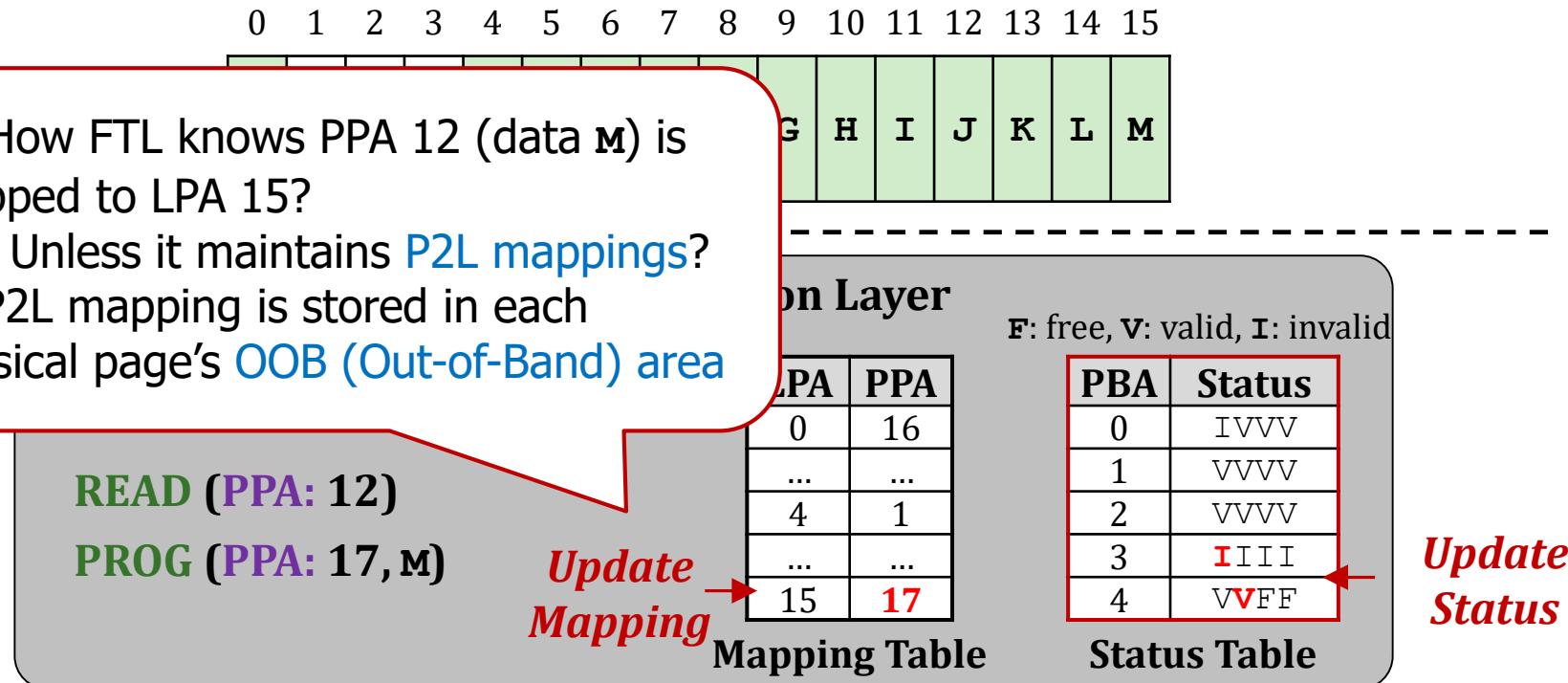


Write Request Handling: Garbage Collection

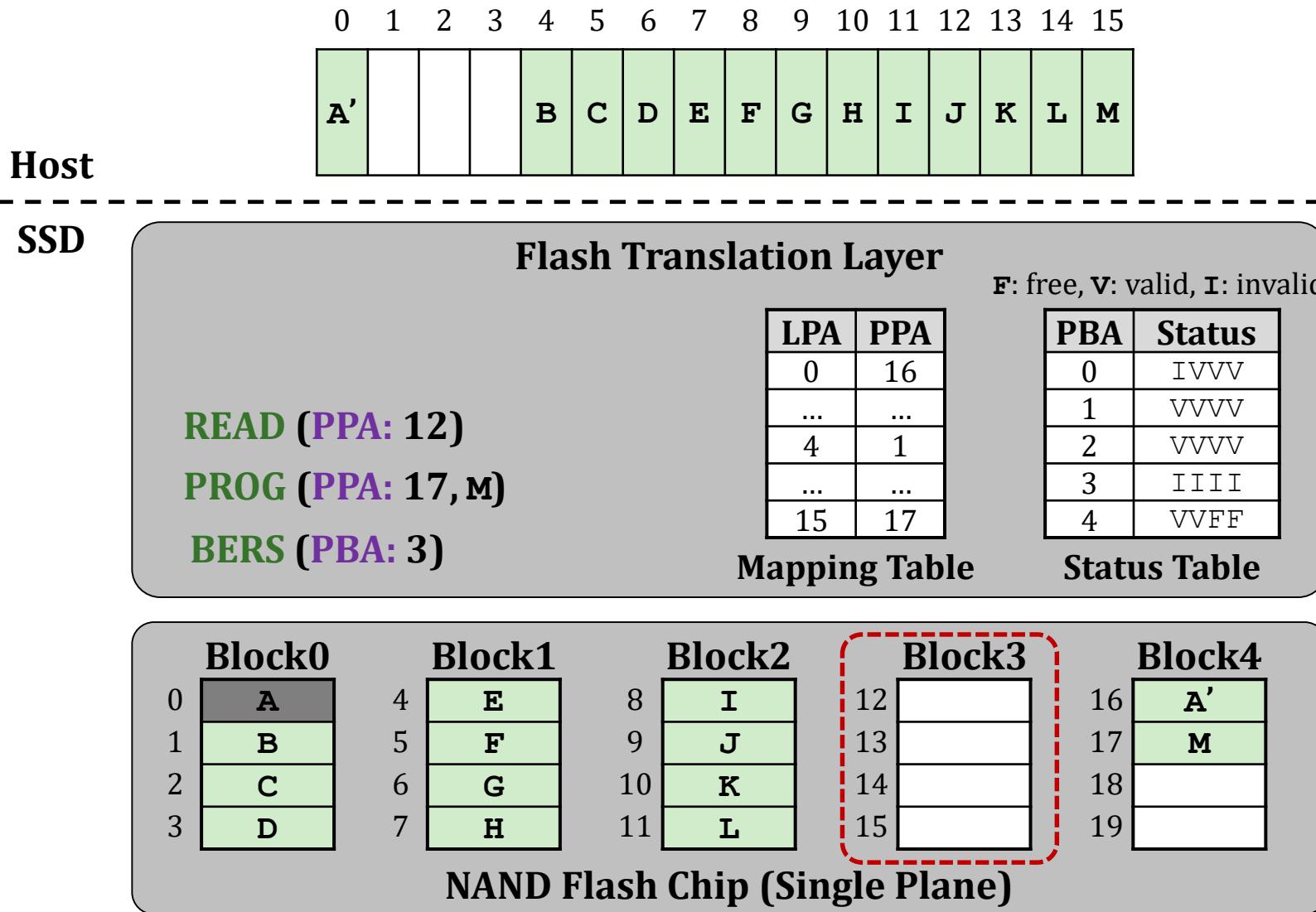


Write Request Handling: Garbage Collection

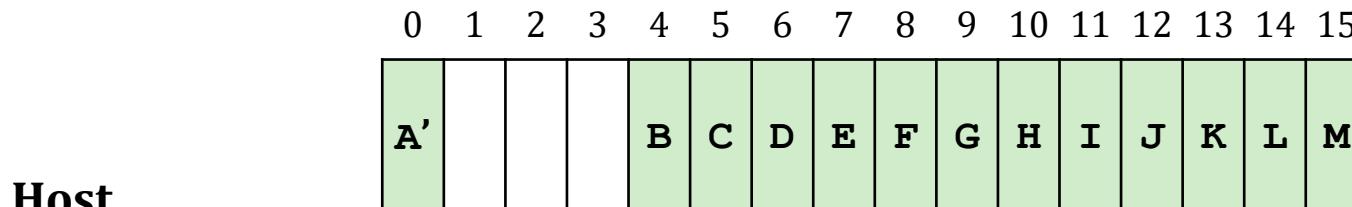
- Q: How FTL knows PPA 12 (data M) is mapped to LPA 15?
 - Unless it maintains P2L mappings?
- A: P2L mapping is stored in each physical page's OOB (Out-of-Band) area



Write Request Handling: Garbage Collection



Write Request Handling: Garbage Collection



SSD

Flash Translation Layer

F: free, V: valid, I: invalid

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

LPA	PPA
0	16
...	...
4	1
...	...
15	17

Mapping Table

PBA	Status
0	IVVV
1	VVVV
2	VVVV
3	FFFF
4	VVFF

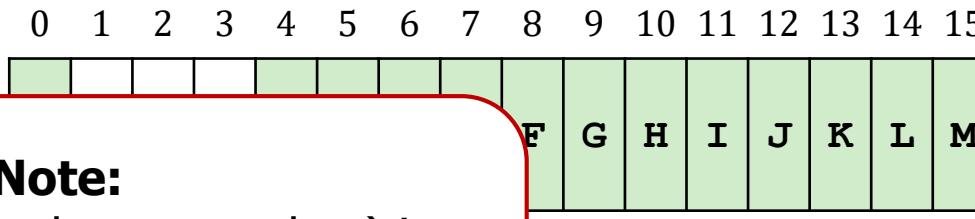
Status Table

Update Status

Block0		Block1		Block2		Block3		Block4	
0	A	4	E	8	I	12		16	A'
1	B	5	F	9	J	13		17	M
2	C	6	G	10	K	14		18	
3	D	7	H	11	L	15		19	

NAND Flash Chip (Single Plane)

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

~~PAGE (PPA: 12)~~
~~PG (PPA: 17, M)~~
BERS (PBA: 3)

Translation Layer

F: free, V: valid, I: invalid

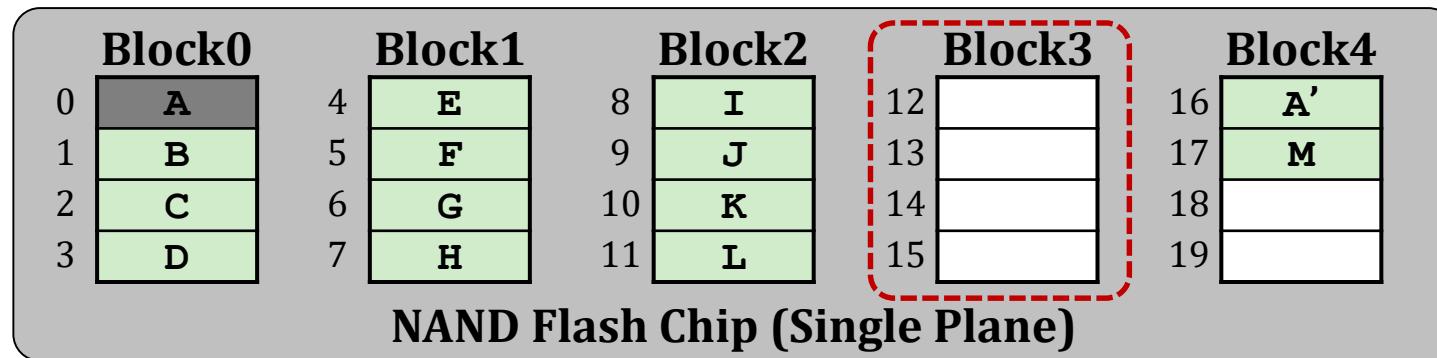
LPA	PPA
0	16
...	...
4	1
...	...
15	17

Mapping Table

PBA	Status
0	I VVV
1	V VVV
2	V VVV
3	FFFF
4	V VFF

Status Table

Update Status



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) $tR = 100 \text{ us}$, $tPROG = 700 \text{ us}$, $tBERS = 5 \text{ ms}$
 - # of pages to copy = $576 \times 0.05 = 28.8 \rightarrow 28 \text{ pages}$
 - GC latency > $28 \times (tR + tPROG) + tBERS = 27,400 \text{ us}$
 - **Order(s) of magnitude larger** latency than tR and $tPROG$
 - Copy operations are **the major contributor** (rather than $tBERS$)
 - 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

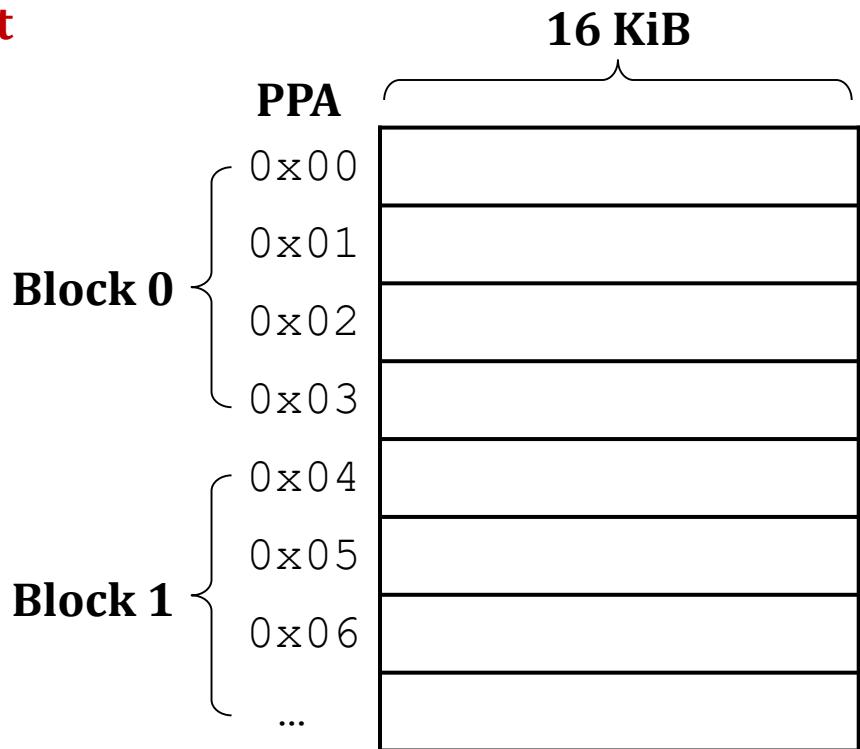
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

0b 0000 0000 0000 0100
16-KiB Page Number 4-KiB Offset

LPA	PPA
0x00	-
0x01	-
0x02	-
0x03	-
0x04	-
0x05	-
...	...



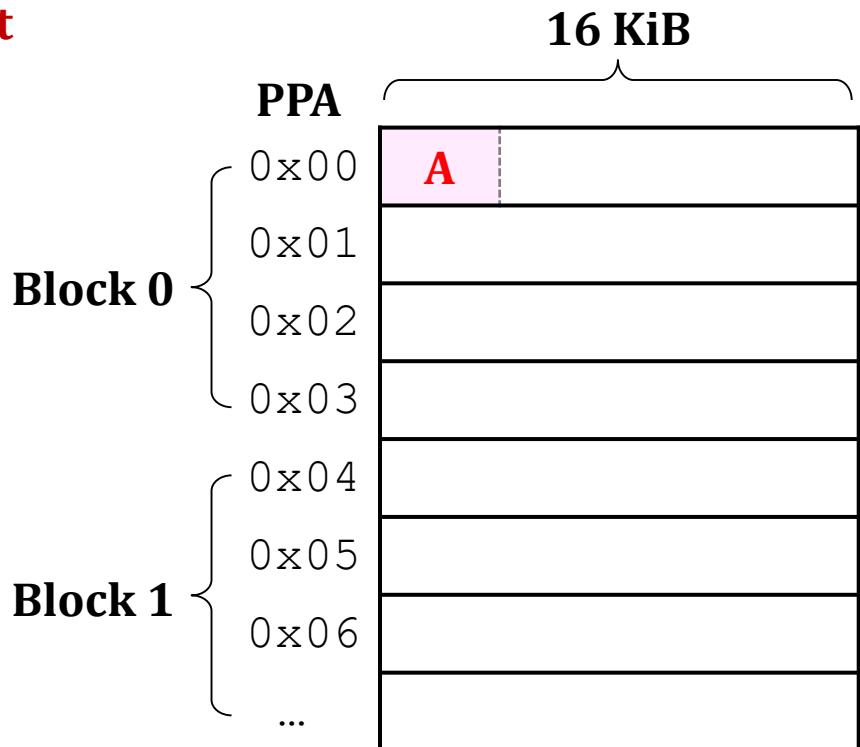
Small Write Requests

- 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

LPA	PPA
0x00	-
0x01	-
0x02	-
0x03	-
0x04	-
0x05	-
...	...



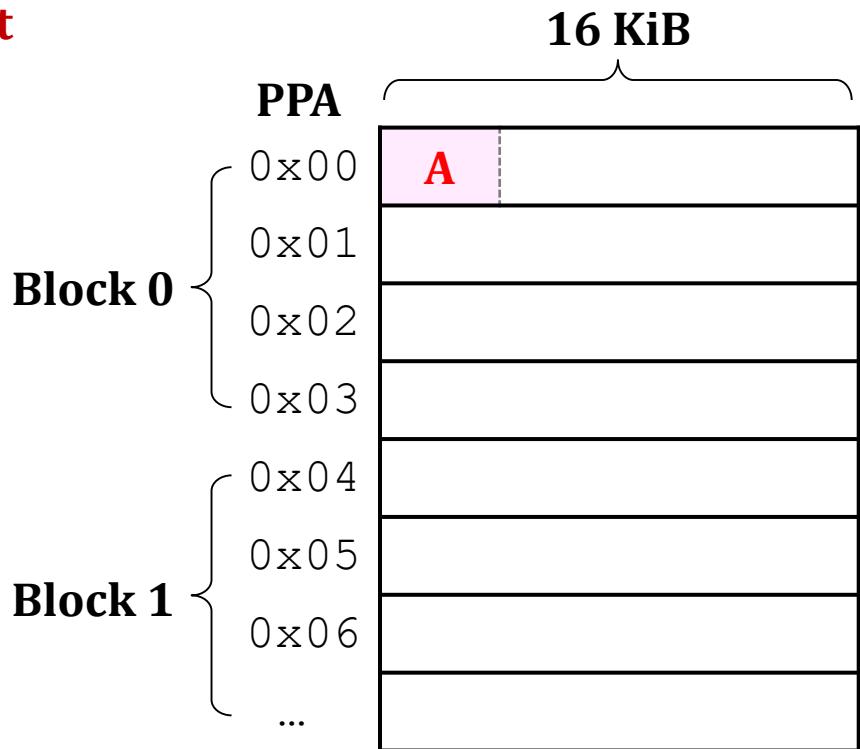
Small Write Requests

- 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

LPA	PPA
0x00	-
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



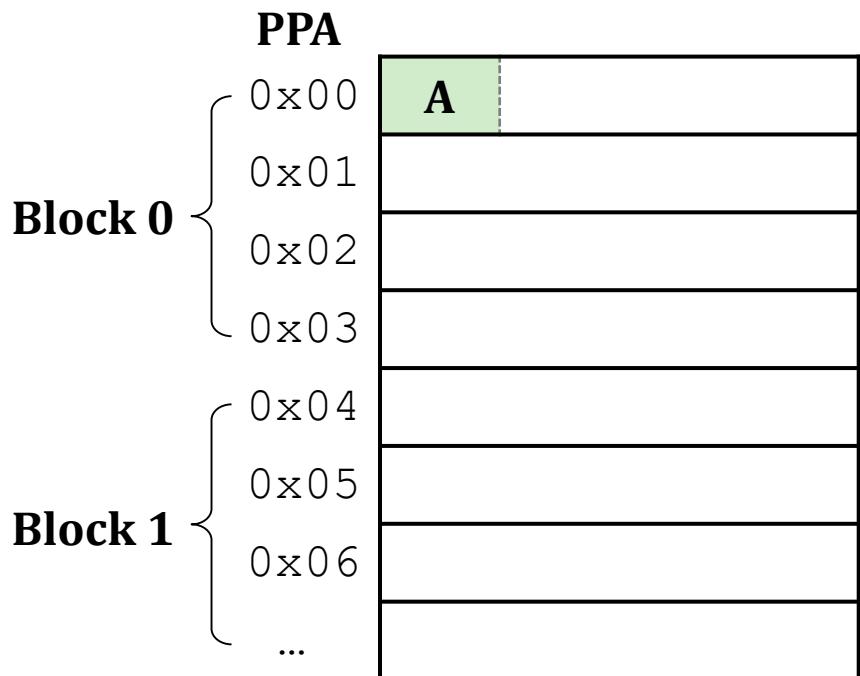
Small Write Requests

- 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

LPA	PPA
0x00	-
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



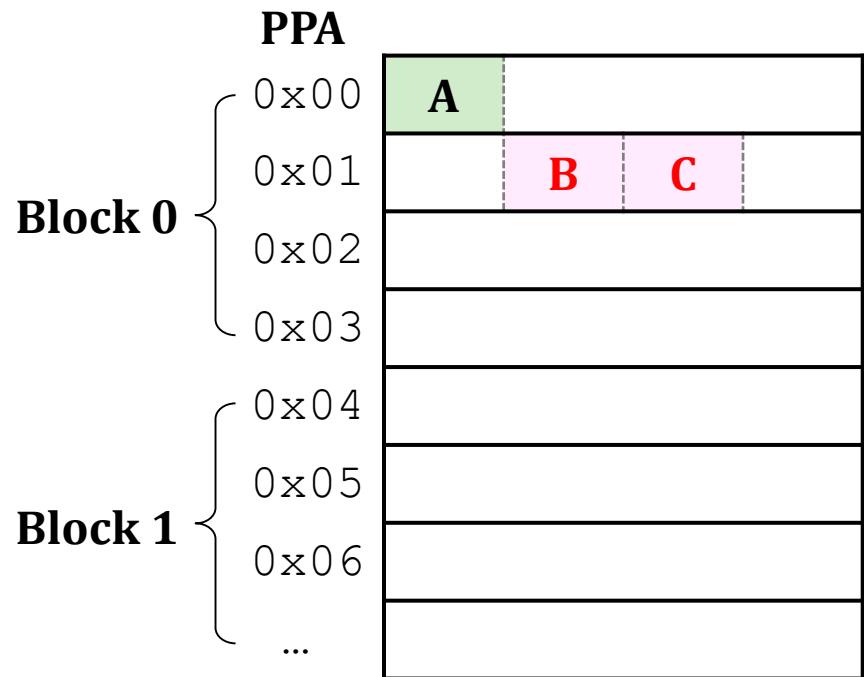
Small Write Requests

- 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

LPA	PPA
0x00	-
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



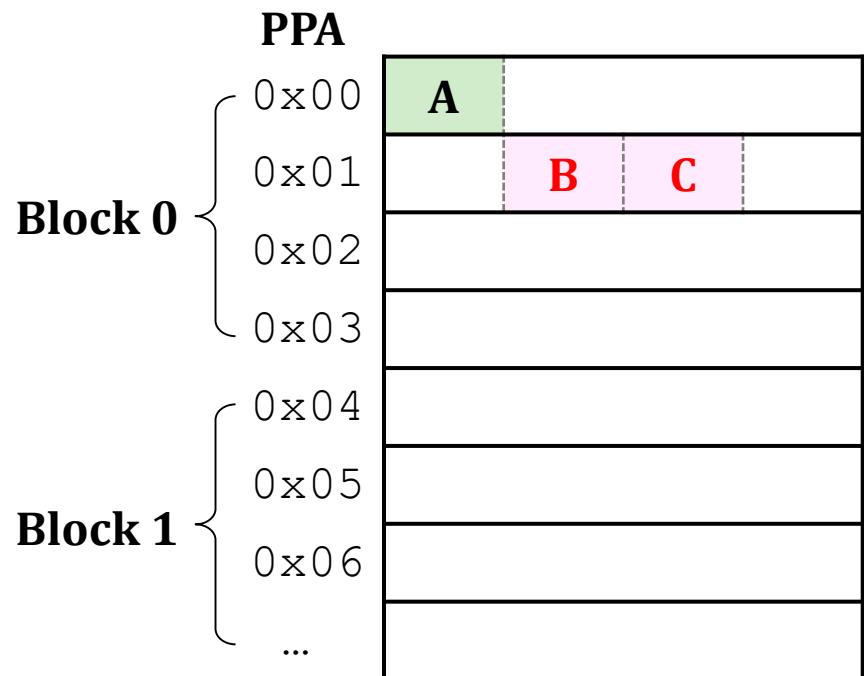
Small Write Requests

- 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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



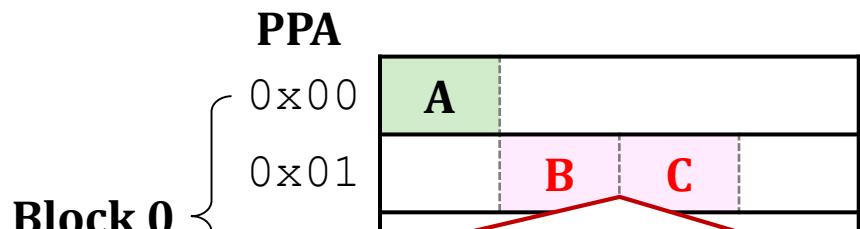
Small Write Requests

- 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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



- Why at the middle of the page?**
 - To keep the 4-KiB offset: mapping table stores only the index of the 16-KiB page!
- Why not using the unused space in physical page 0x00?**
 - That space is already mapped to logical pages 0x05~0x07 (not written yet).

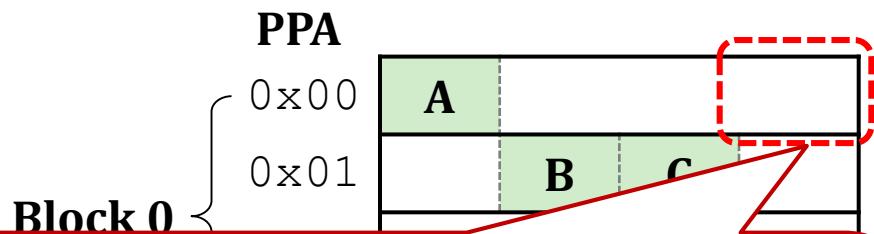
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



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.

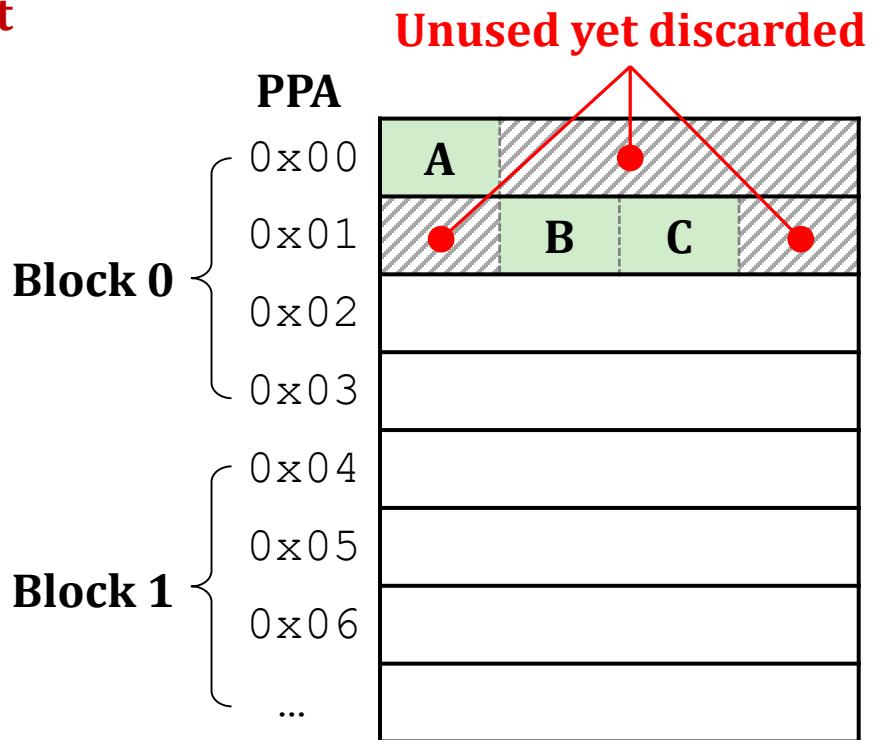
Small Write Requests

- 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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



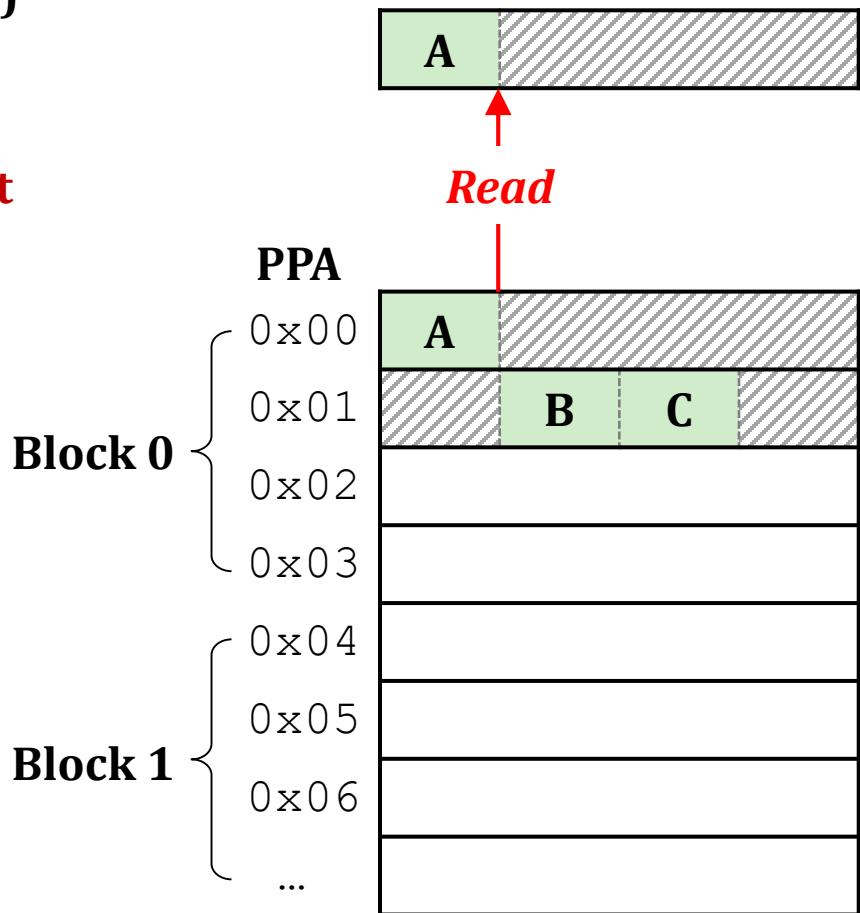
Small Write Requests

- 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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



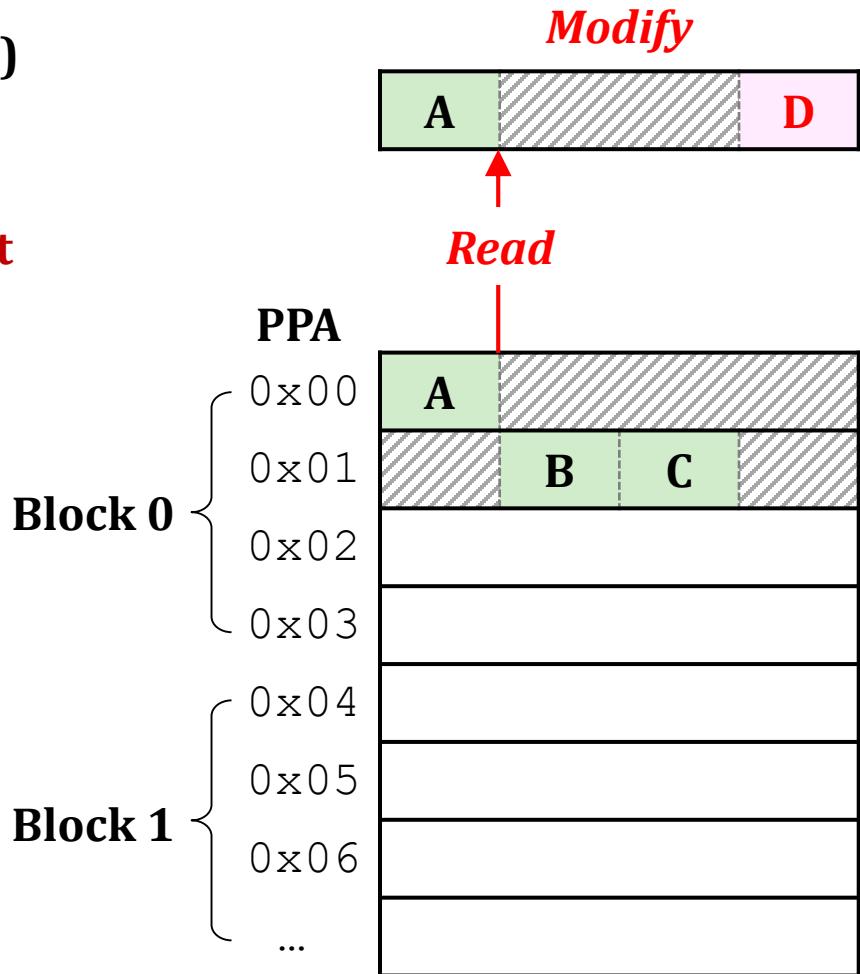
Small Write Requests

- 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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



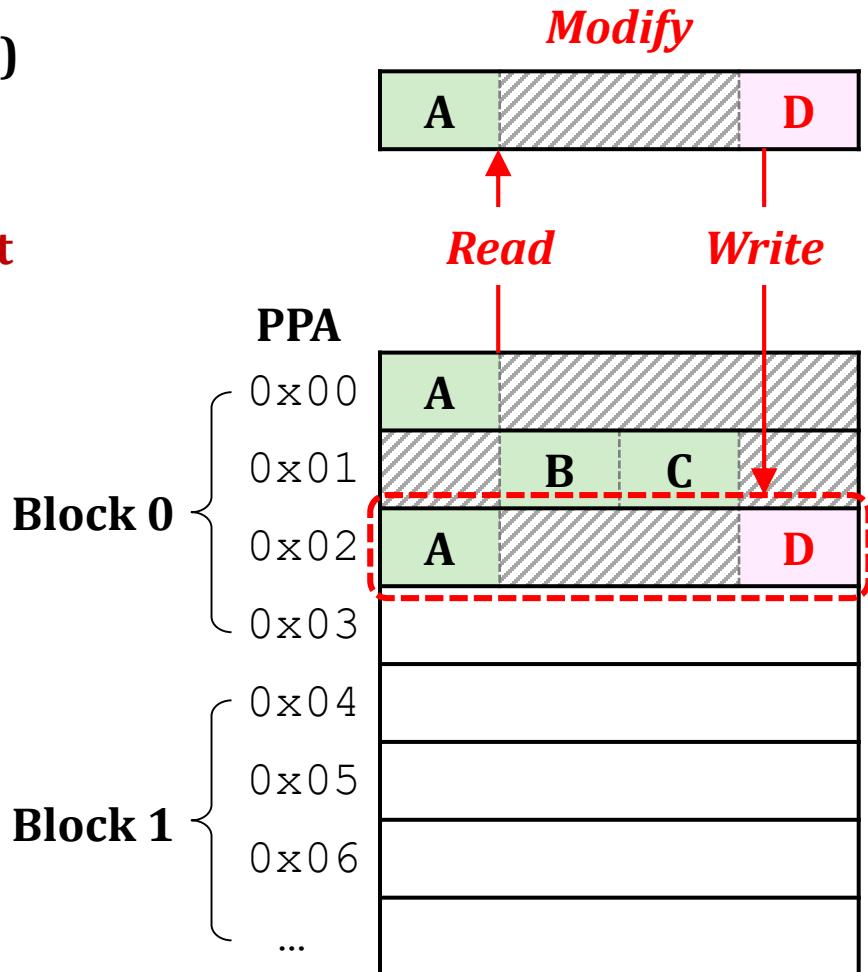
Small Write Requests

- 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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	-
0x03	-
0x04	-
0x05	-
...	...



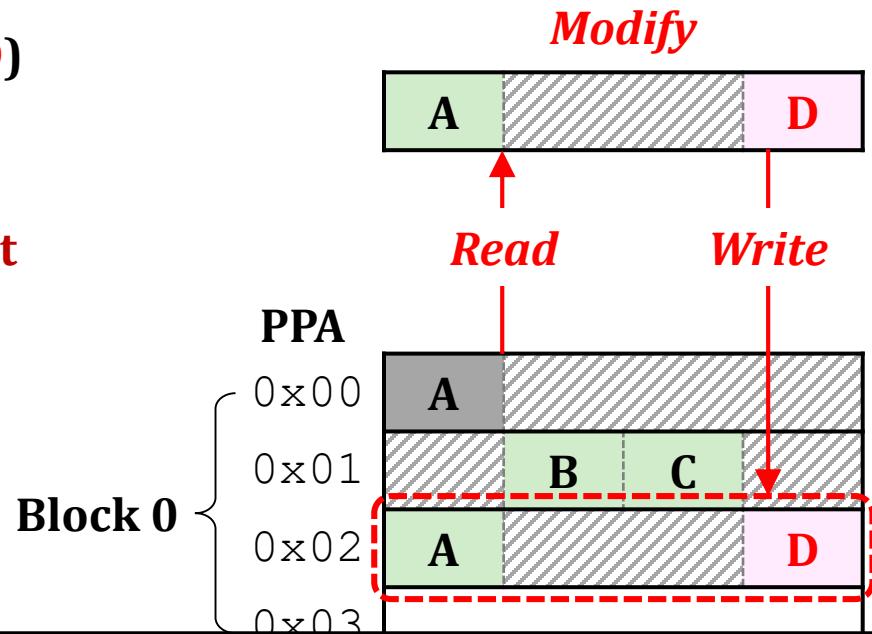
Small Write Requests

- 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

LPA	PPA
0x00	0x01
0x01	0x02
0x02	-



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)

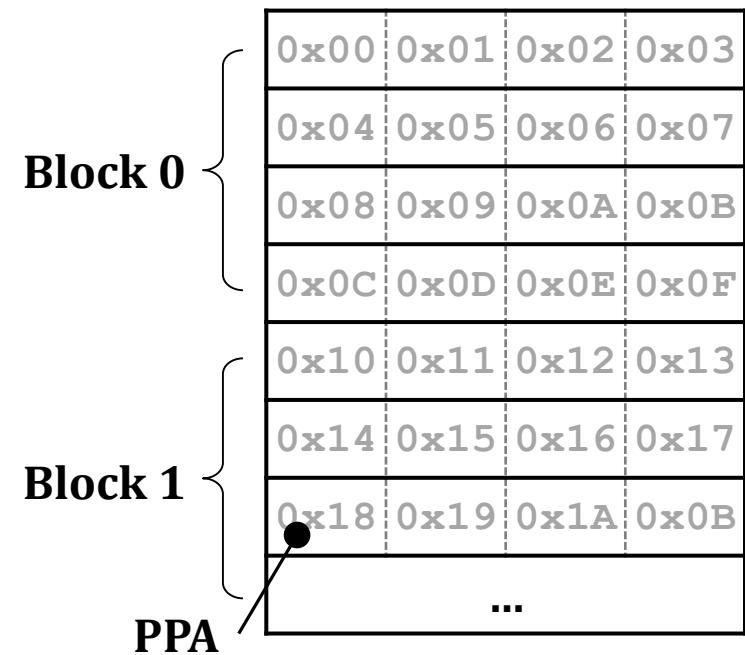
Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

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

Page Buffer



LPA	PPA
0x00	-
0x01	-
...	-
0x04	-
...	-
0x07	-
...	...



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)

Page Buffer

A			
---	--	--	--

LPA	PPA
0x00	-
0x01	-
...	-
0x04	-
...	-
0x07	-
...	...

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	0x0B
			...

Block 0

Block 1

PPA

The diagram illustrates the fine-grained mapping and page buffering mechanism. The LPA table shows that LPA 0x04 is mapped to none (-), while other LPAs like 0x01, 0x07, and others have valid PPA entries. The page buffer contains four pages (A, B, C, D) in a single block. The PPA table shows two blocks of four pages each. A pointer from the PPA entry for LPA 0x04 points to the start of Block 1, indicating that data for LPA 0x04 is buffered in Block 1.

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

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

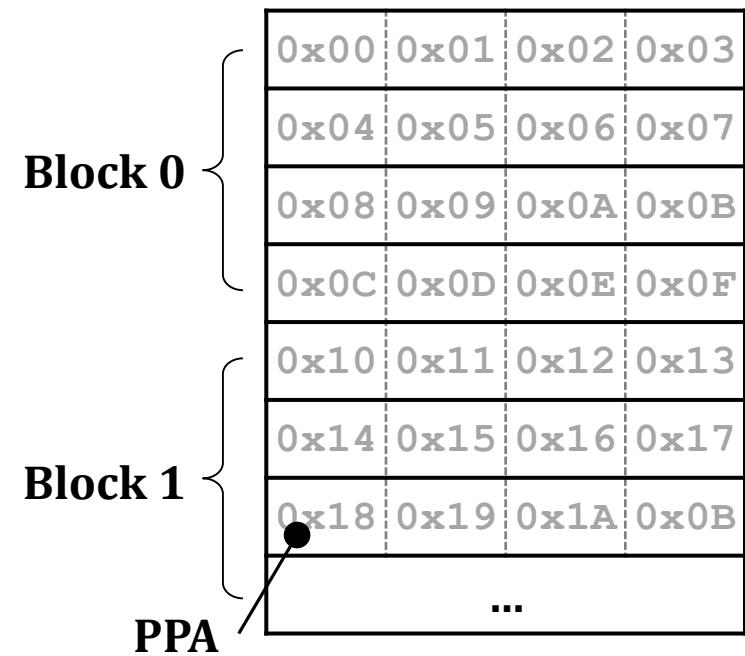
Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

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

Page Buffer



LPA	PPA
0x00	-
0x01	-
...	-
0x04	0x00
...	-
0x07	-
...	...



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)

Page Buffer

A			
---	--	--	--

LPA	PPA
0x00	-
0x01	-
...	-
0x04	0x00
...	-
0x07	-
...	...

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	0x0B
			...

Block 0

Block 1

PPA

Fine-Grained Mapping + Page Buffer

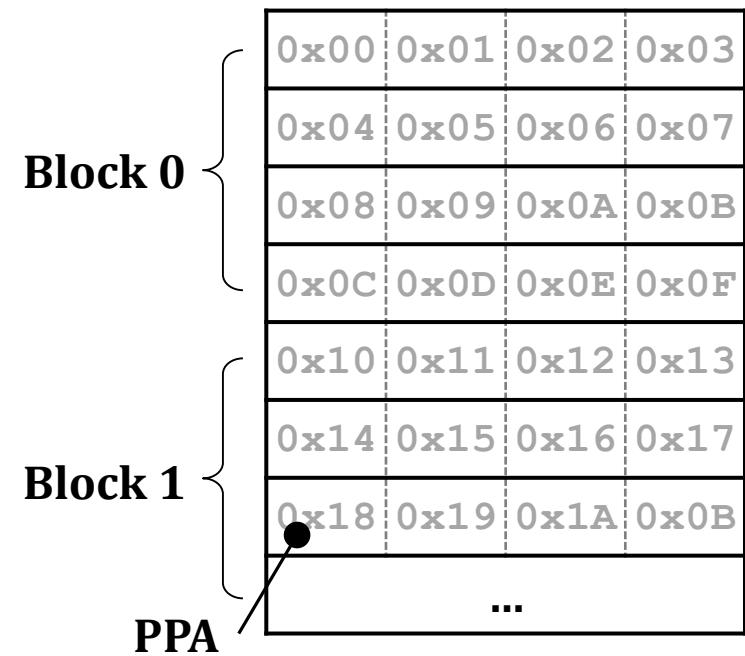
- Write a page only when there are sufficient data blocks

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

Page Buffer

A	B	C
---	---	---

LPA	PPA
0x00	-
0x01	-
...	-
0x04	0x00
...	-
0x07	-
...	...



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)

Page Buffer

A	B	C	

LPA	PPA
0x00	-
0x01	0x01
0x04	0x00
...	-
0x07	-
...	...

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	0x0B
			...

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)

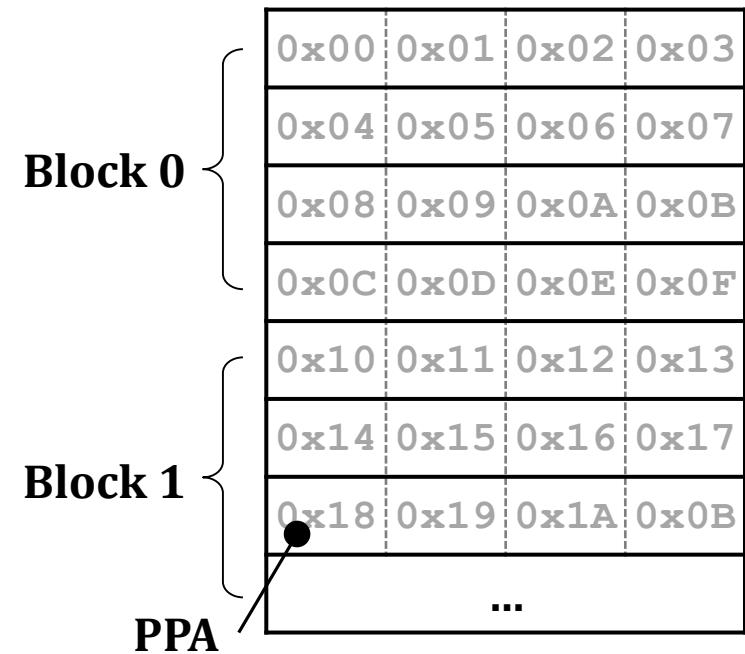
Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

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

Page Buffer

A	B	C	

LPA	PPA
0x00	-
0x01	0x01
...	-
0x04	0x00
...	-
0x07	-
...	...



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

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

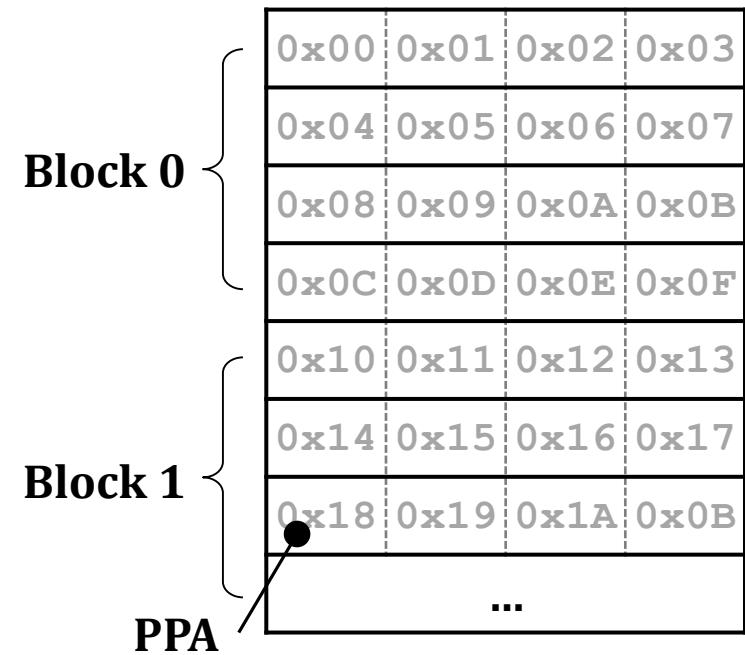
Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)

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

Page Buffer

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

LPA	PPA
0x00	-
0x01	0x01
...	-
0x04	0x00
...	-
0x07	-
...	...



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)

Page Buffer

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

LPA	PPA
0x00	-
0x01	0x01
...	-
0x04	0x00
...	-
0x07	0x03
...	...

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	0x0B
...			

Block 0

Block 1

PPA

The diagram illustrates the fine-grained mapping and page buffering mechanism. The LPA table shows the logical page addresses and their corresponding physical page addresses. The Page Buffer is organized into blocks, where each block contains four pages. The 0x07 LPA entry in the LPA table maps to PPA 0x03, which is located in Block 1 of the Page Buffer. The Page Buffer is shown with four blocks, each containing four pages, and the 0x03 page is highlighted.

Fine-Grained Mapping + Page Buffer

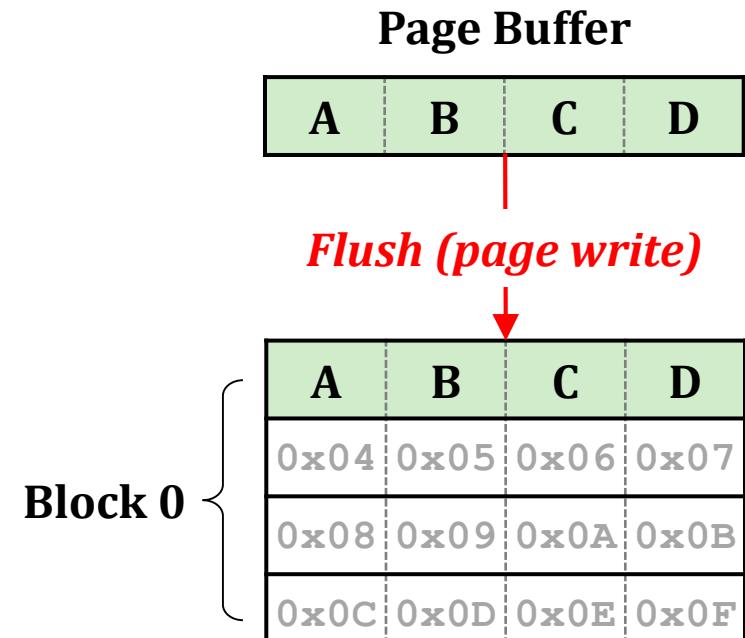
- Write a page only when there are sufficient data blocks

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)

LPA	PPA
0x00	-
0x01	0x01
...	-



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

3 writes (+1 read) → 1 write

Drawbacks of Fine-Grained Mapping

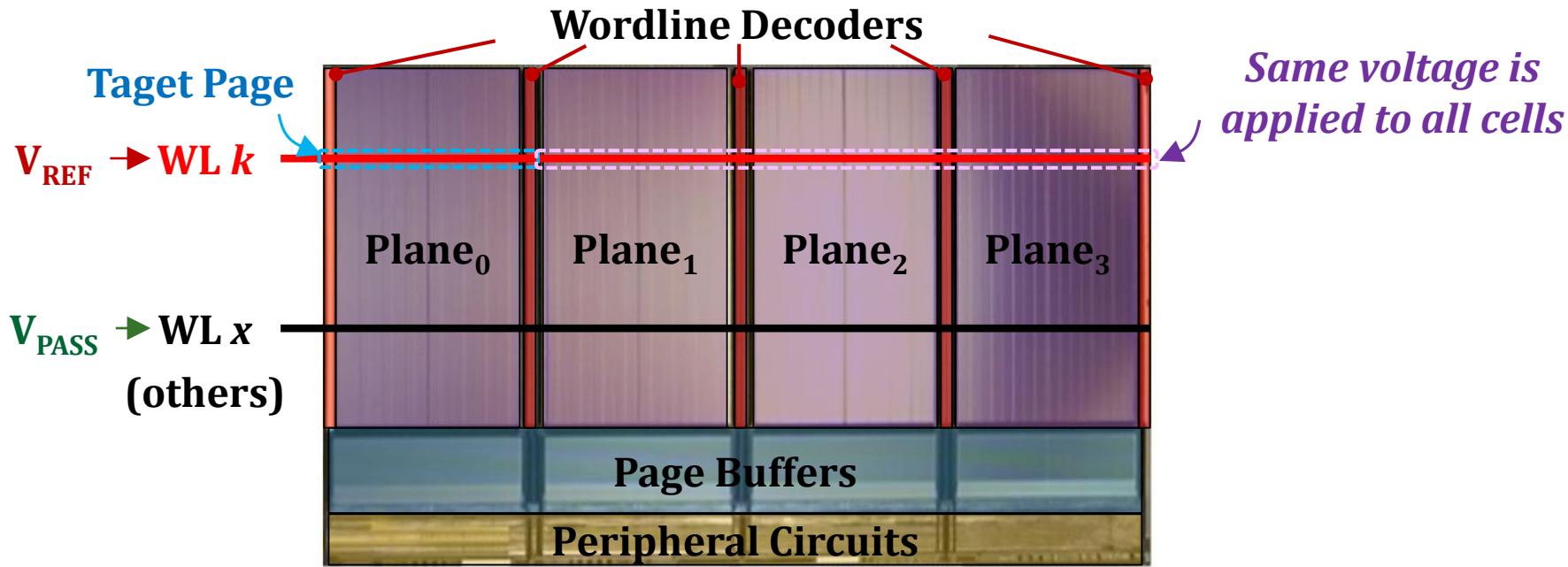
- Larger mapping table
 - 16-KiB mapping → 4 bytes per 16-KiB page = 0.025%
 - 4-KiB mapping → 4 bytes per 4-KiB page = 0.1%
 - For a 2-TB SSD, 2-GB DRAM is required.
 - Increases the SSD's price and power/energy consumption
- Data durability of written data
 - Page buffers are implemented by using volatile memory (e.g., SRAM or DRAM).

Despite non-negligible drawbacks,
fine-grained mapping is widely used
in modern SSDs due to its high benefits

Multi-plane Operation-Aware Block Management

Recap: Multi-Plane Operations

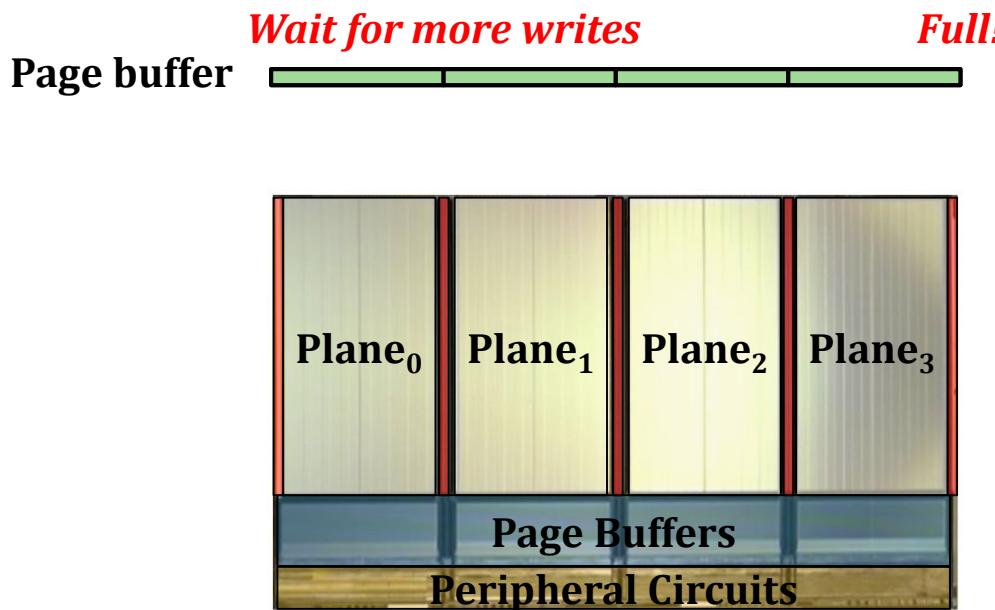
- Concurrent operations on different planes
 - Recall: Planes share WLs and row/column decoders



- Opportunity: Planes can **concurrently** operate
- Constraints: Only for **the same operations on the same page offset**

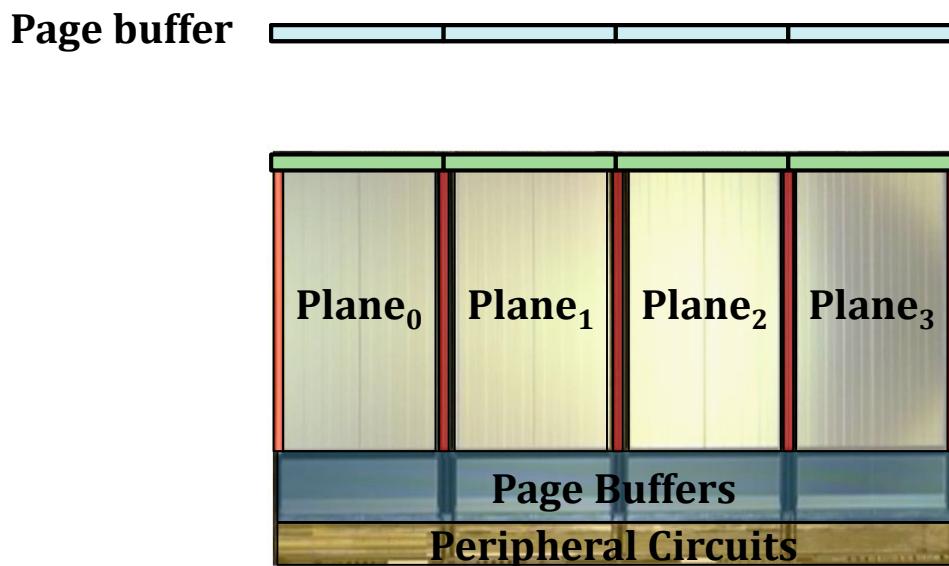
Multi-Plane-Aware Data Placement

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



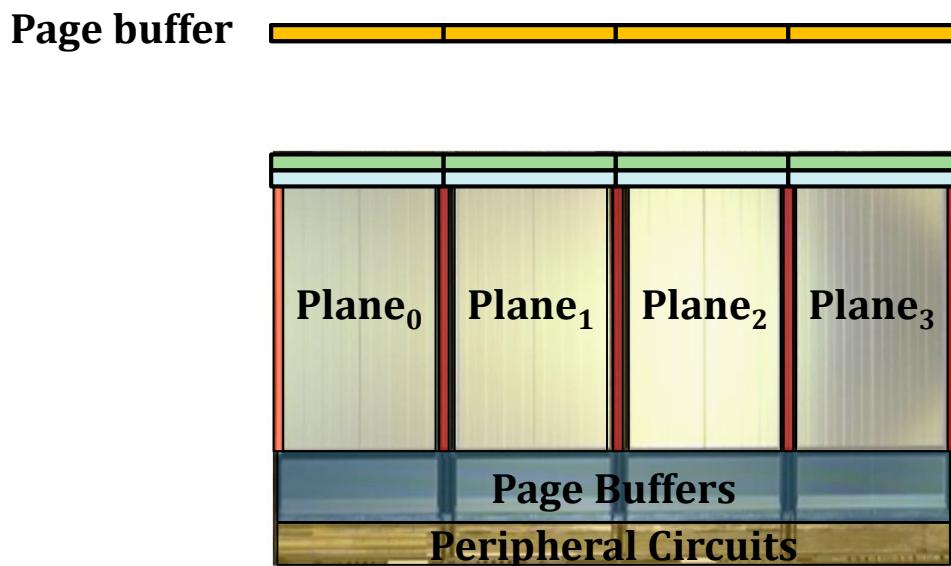
Multi-Plane-Aware Data Placement

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



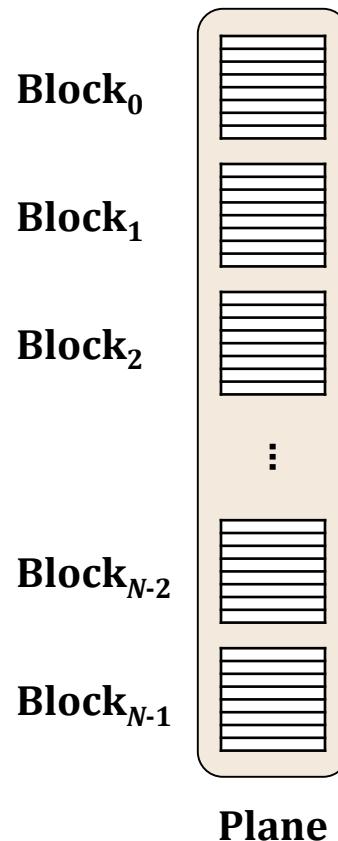
Multi-Plane-Aware Data Placement

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



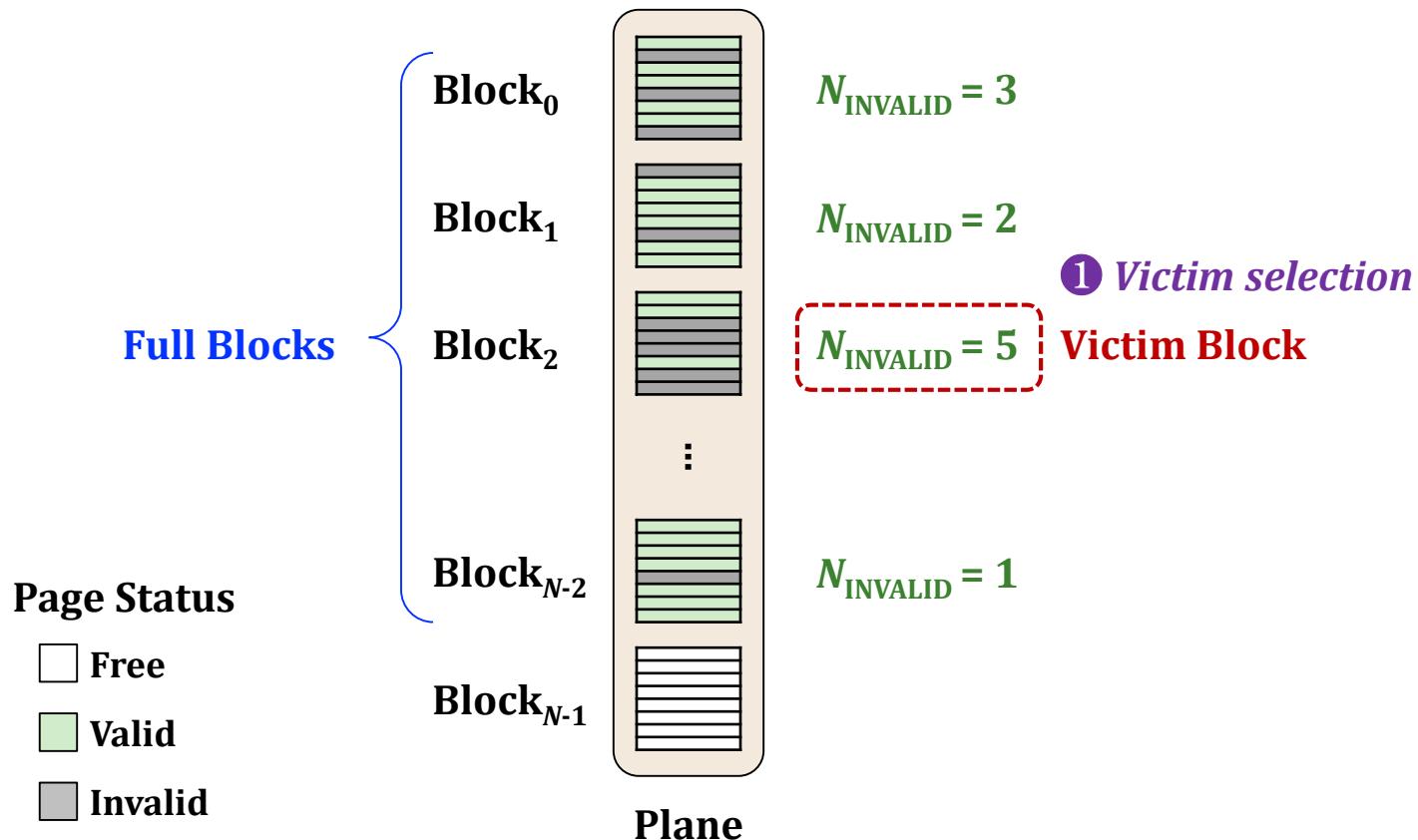
Multi-Plane-Aware Block Management

- 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**).



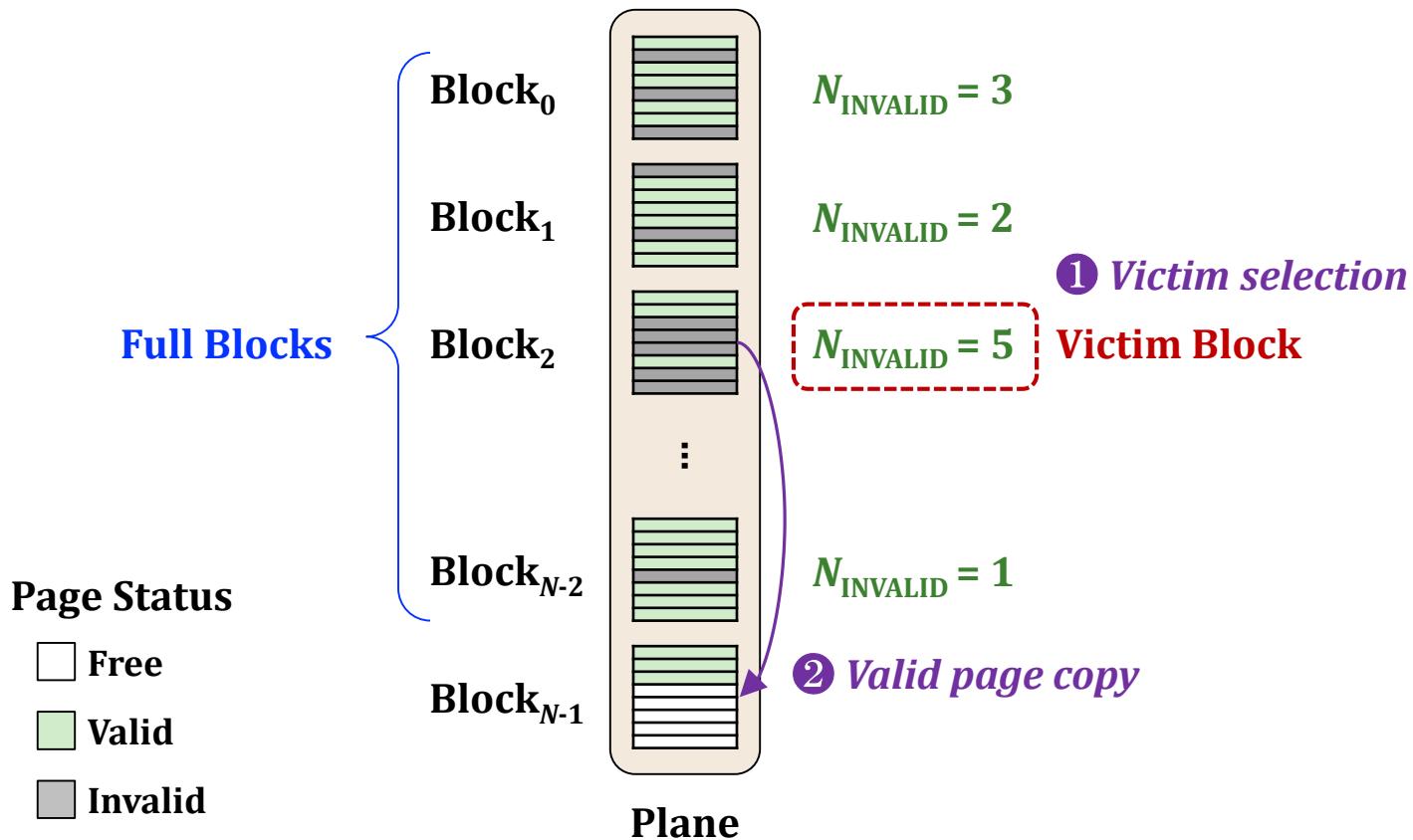
Multi-Plane-Aware Block Management

- 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**).



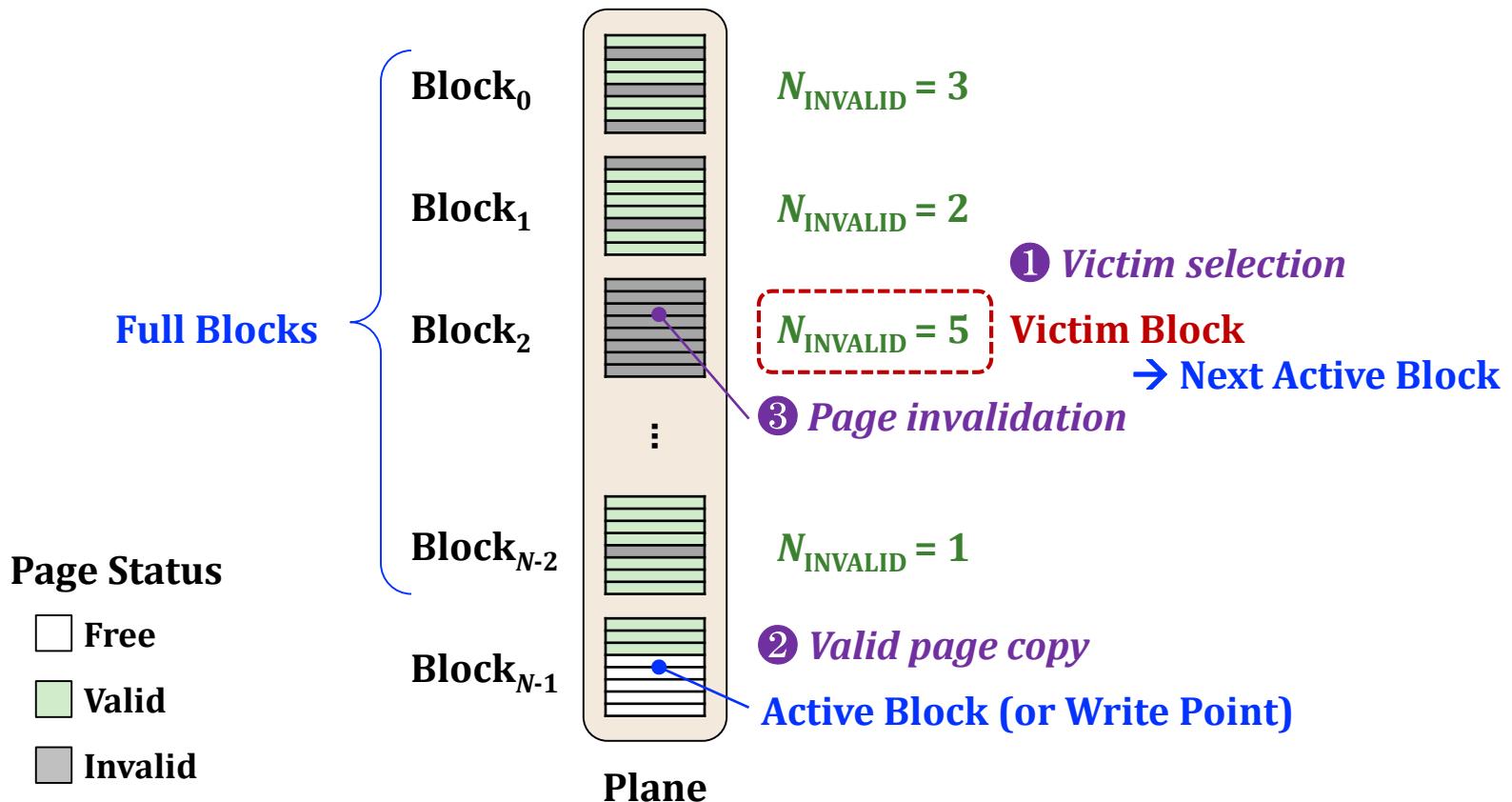
Multi-Plane-Aware Block Management

- 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**).



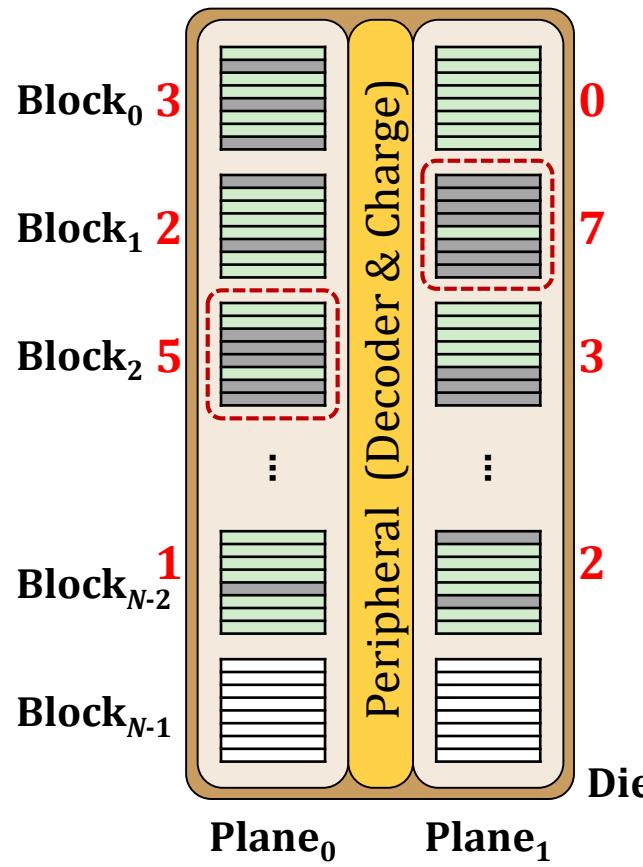
Multi-Plane-Aware Block Management

- 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**).



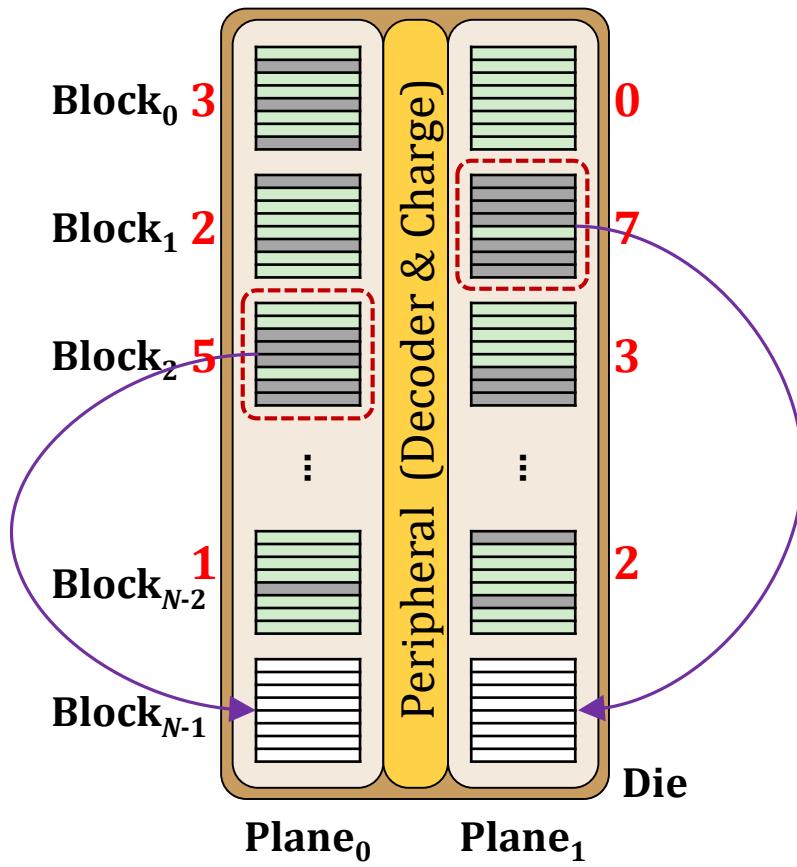
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



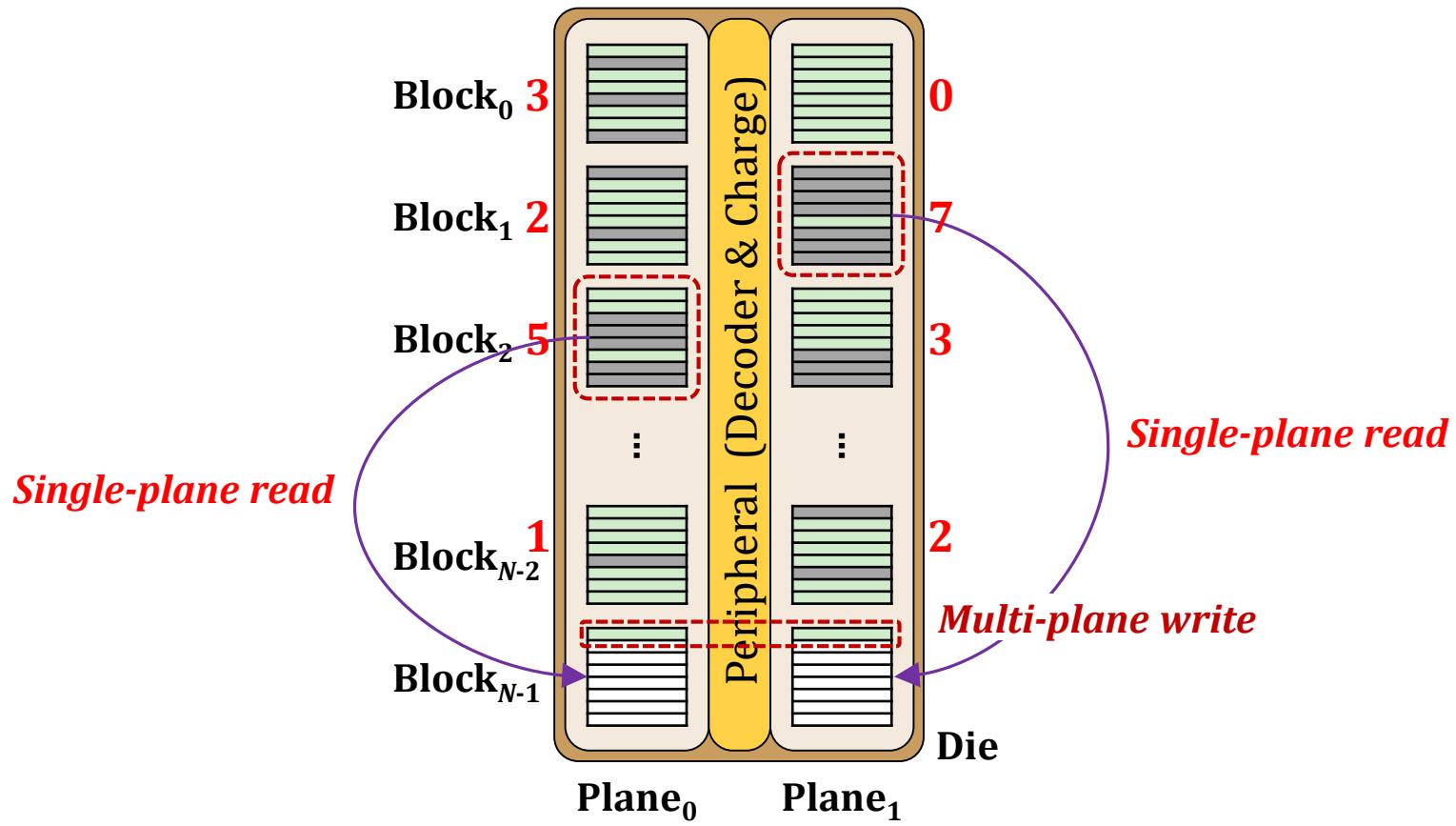
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



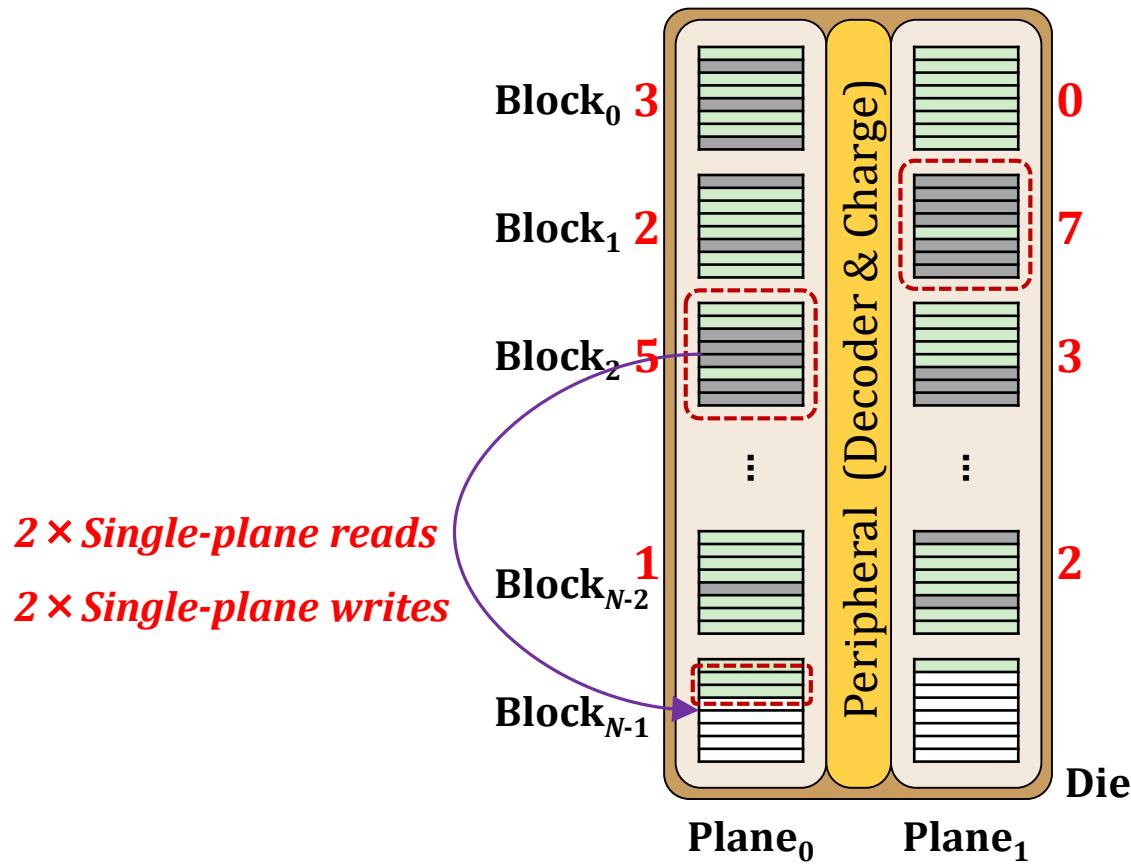
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



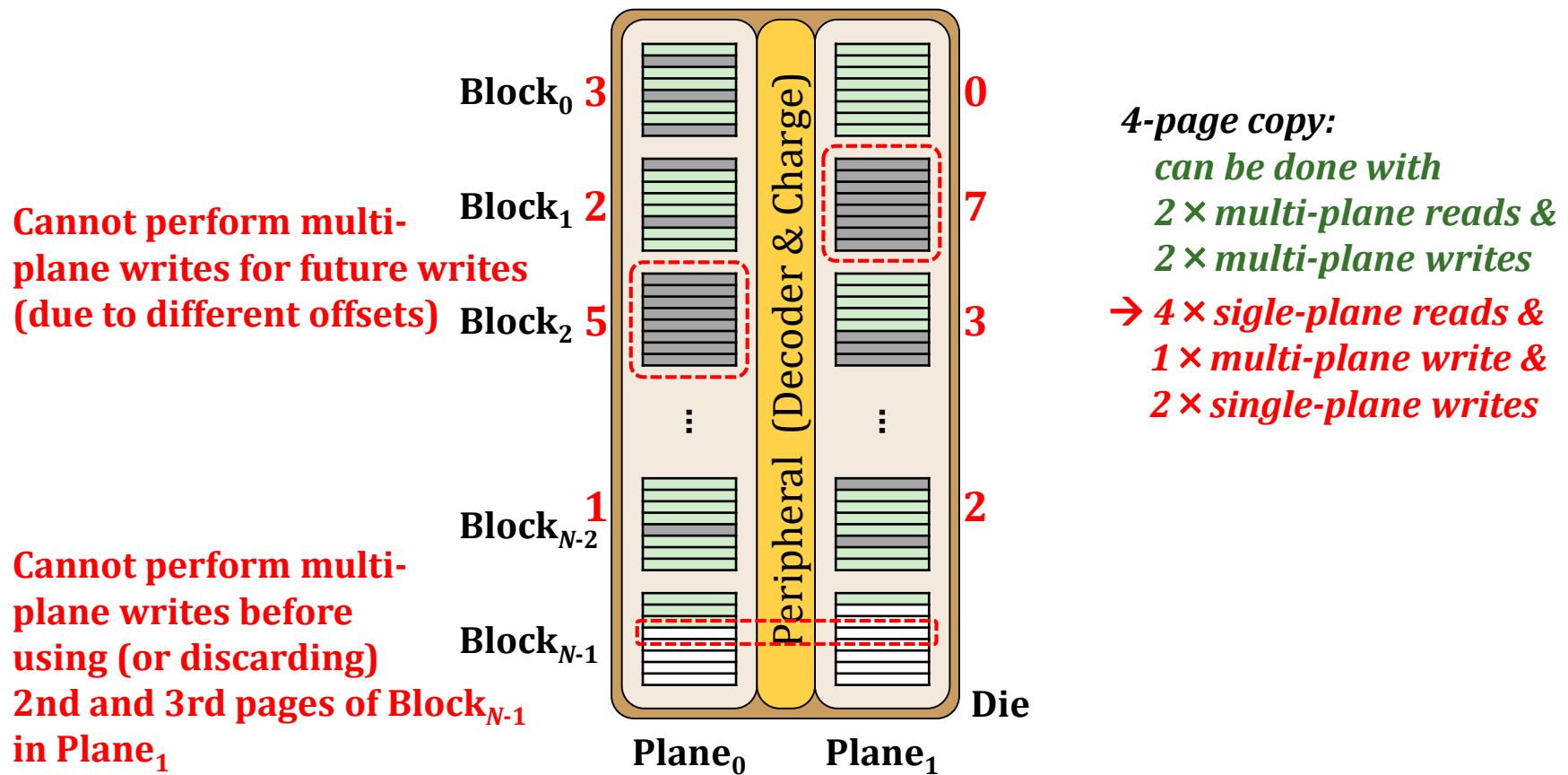
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



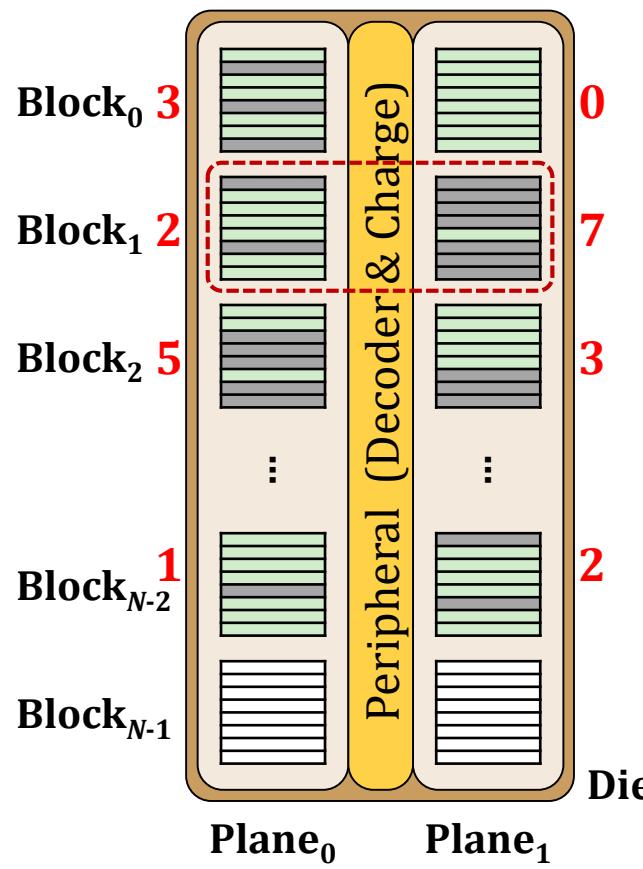
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



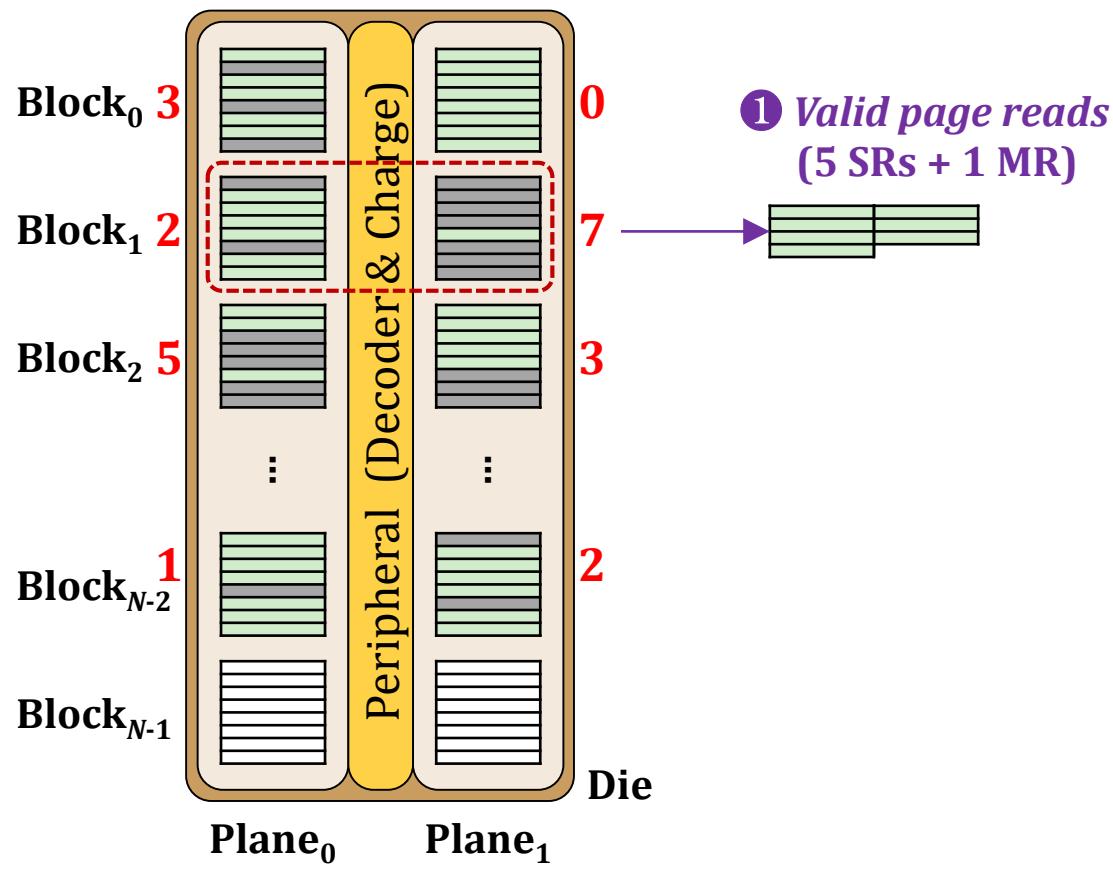
Multi-Plane-Aware Block Management

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



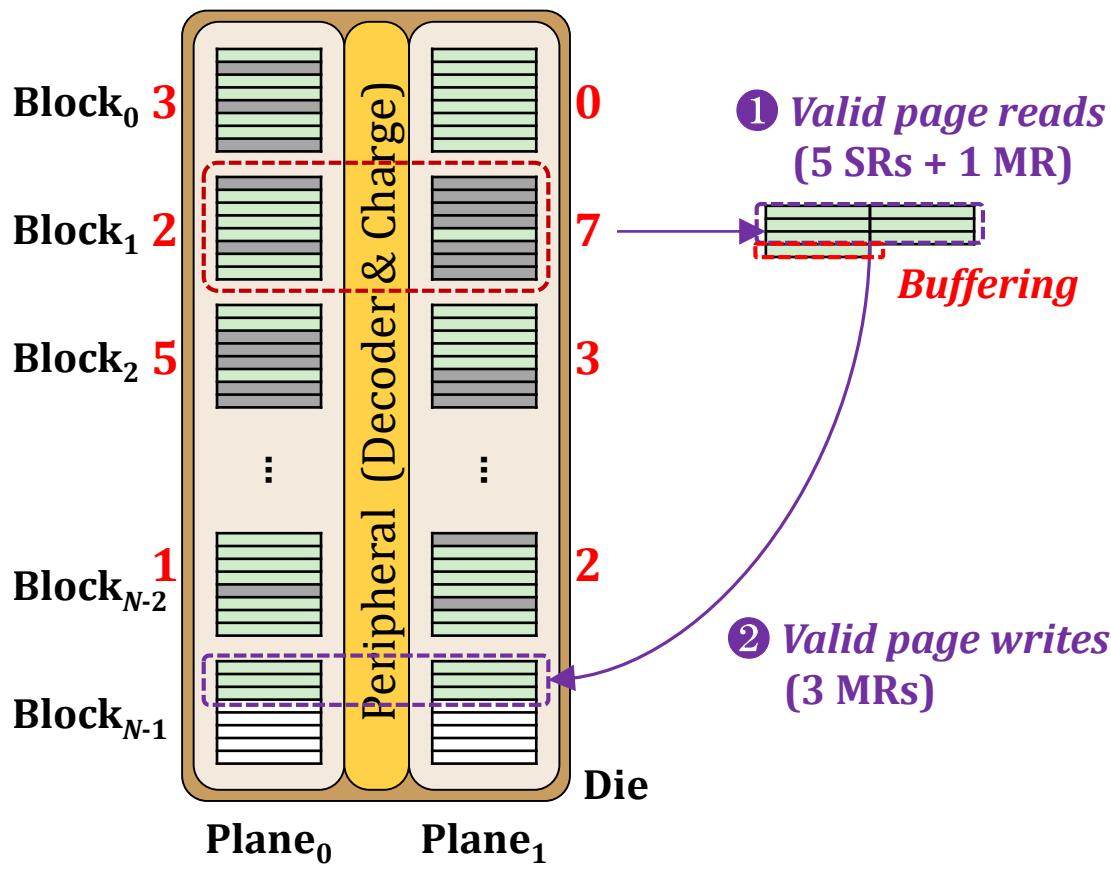
Multi-Plane-Aware Block Management

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



Multi-Plane-Aware Block Management

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

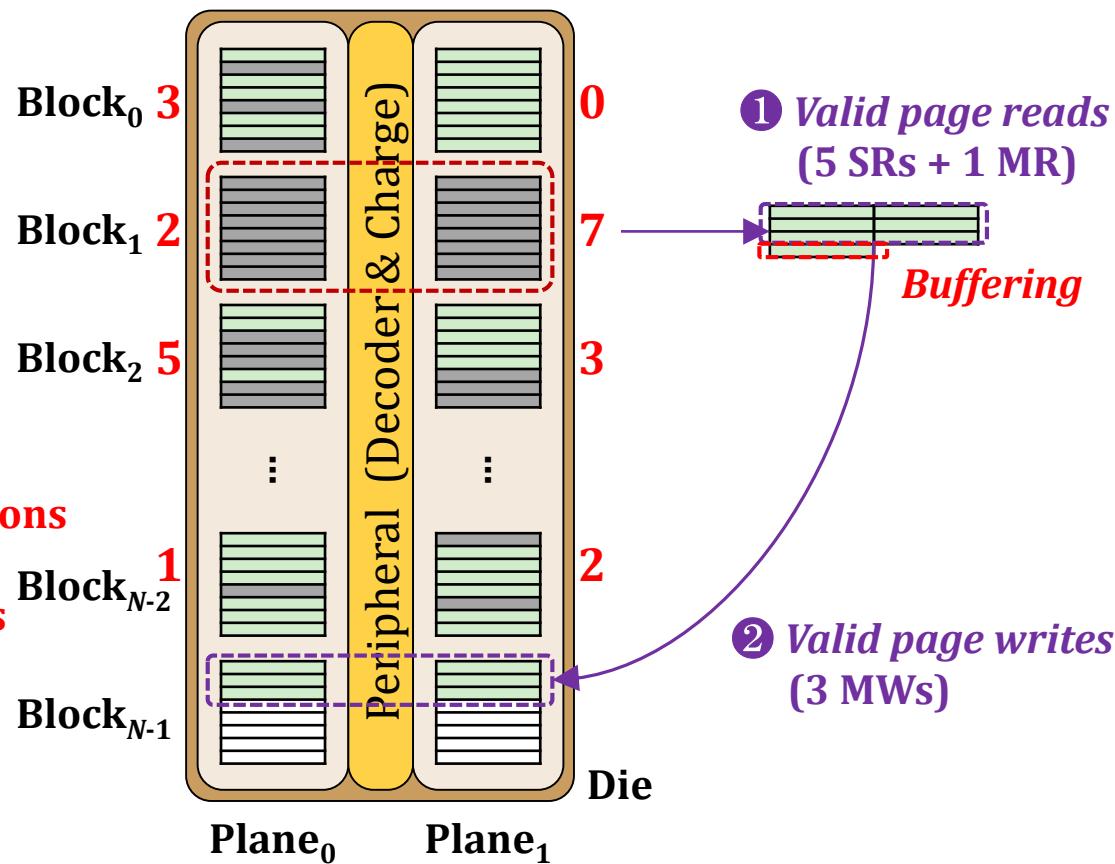


Multi-Plane-Aware Block Management

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

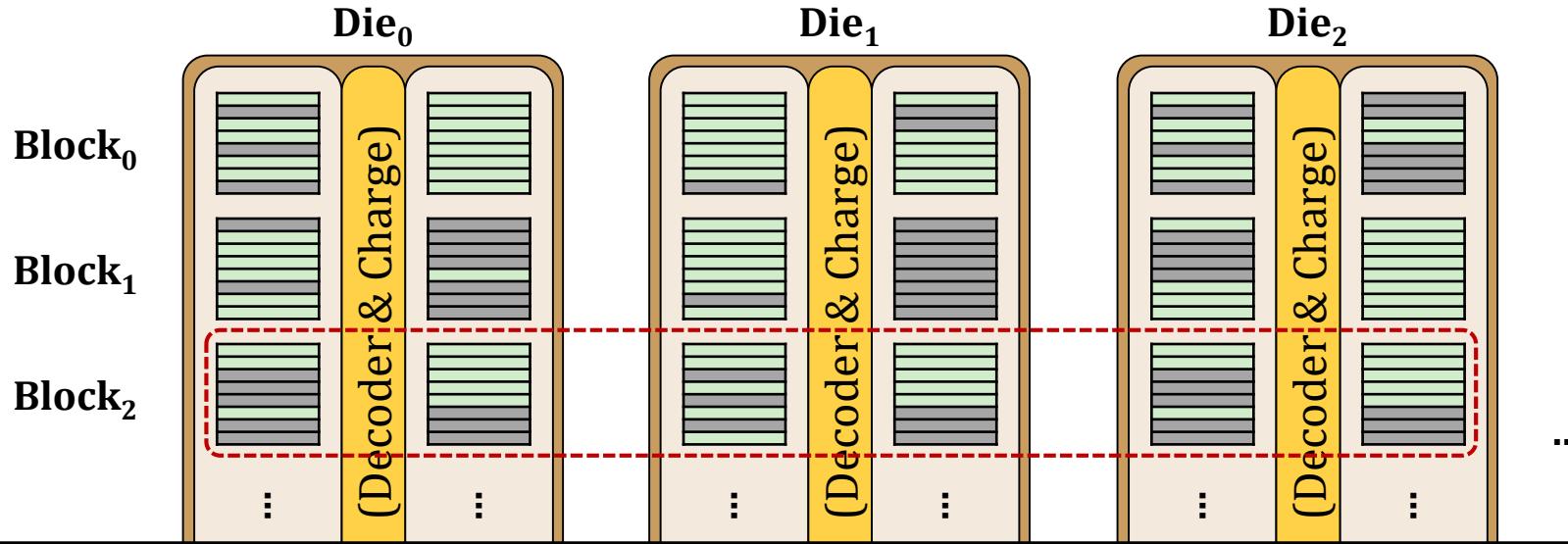
Pros:
Keep performing
multi-plane writes

Cons:
More read/write operations
→ 5 SRs + 1 MR + 3 MWs
vs. 4 SRs + 1 MW + 2 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

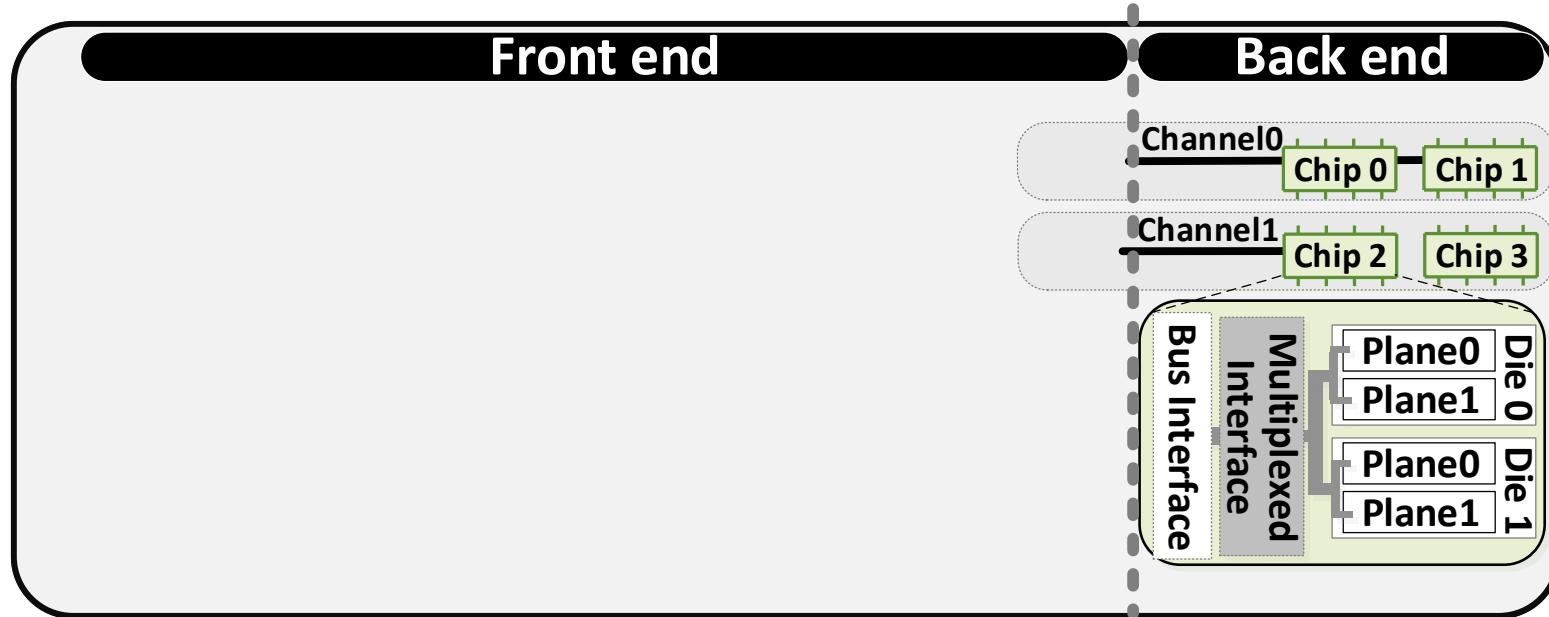
FLIN:
Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion

Internal Components of a Modern SSD

SAFARI

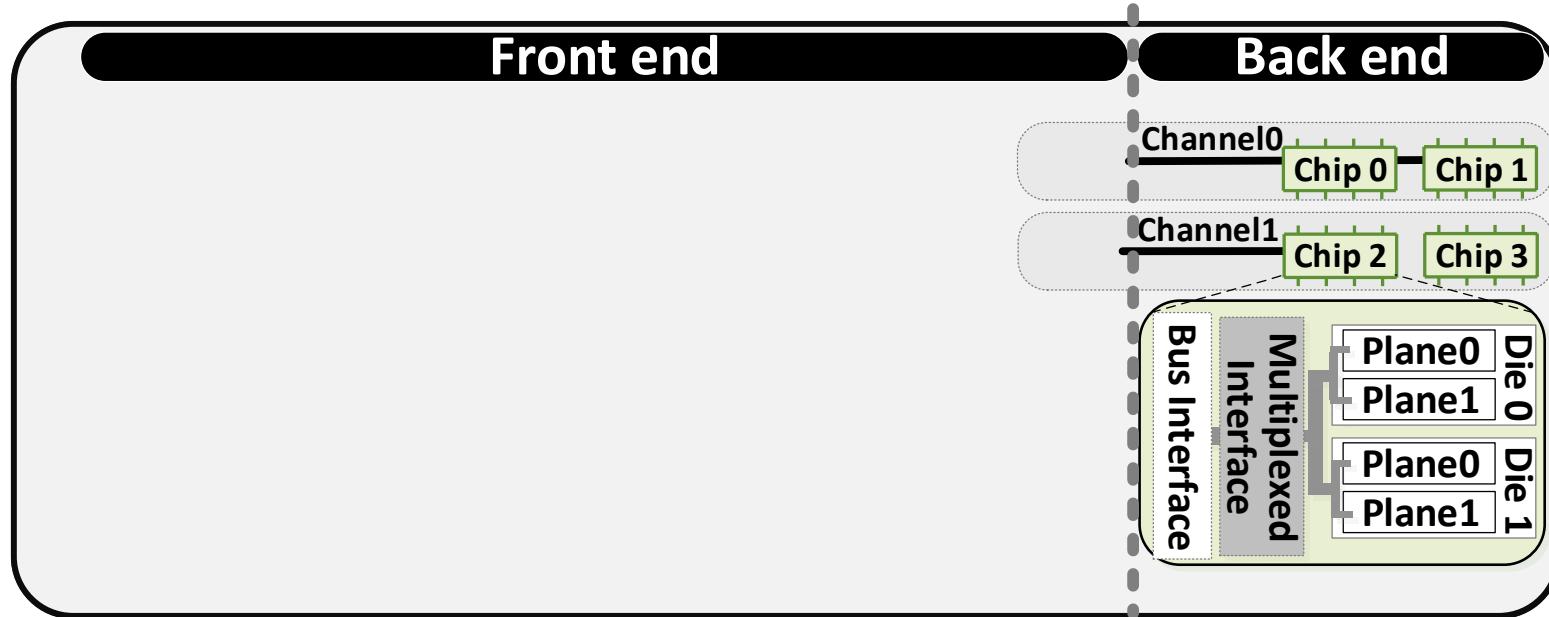


■ Back End: data storage

- Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)

Internal Components of a Modern SSD

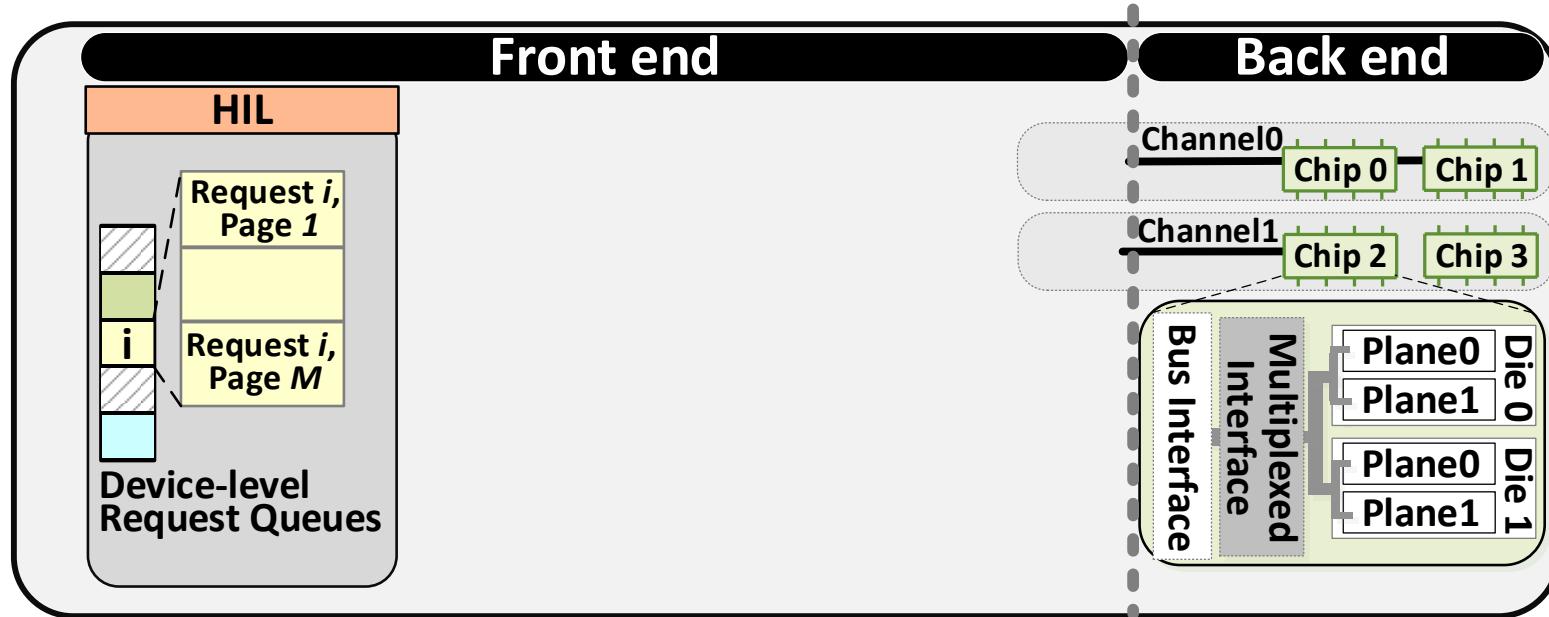
SAFARI



- **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

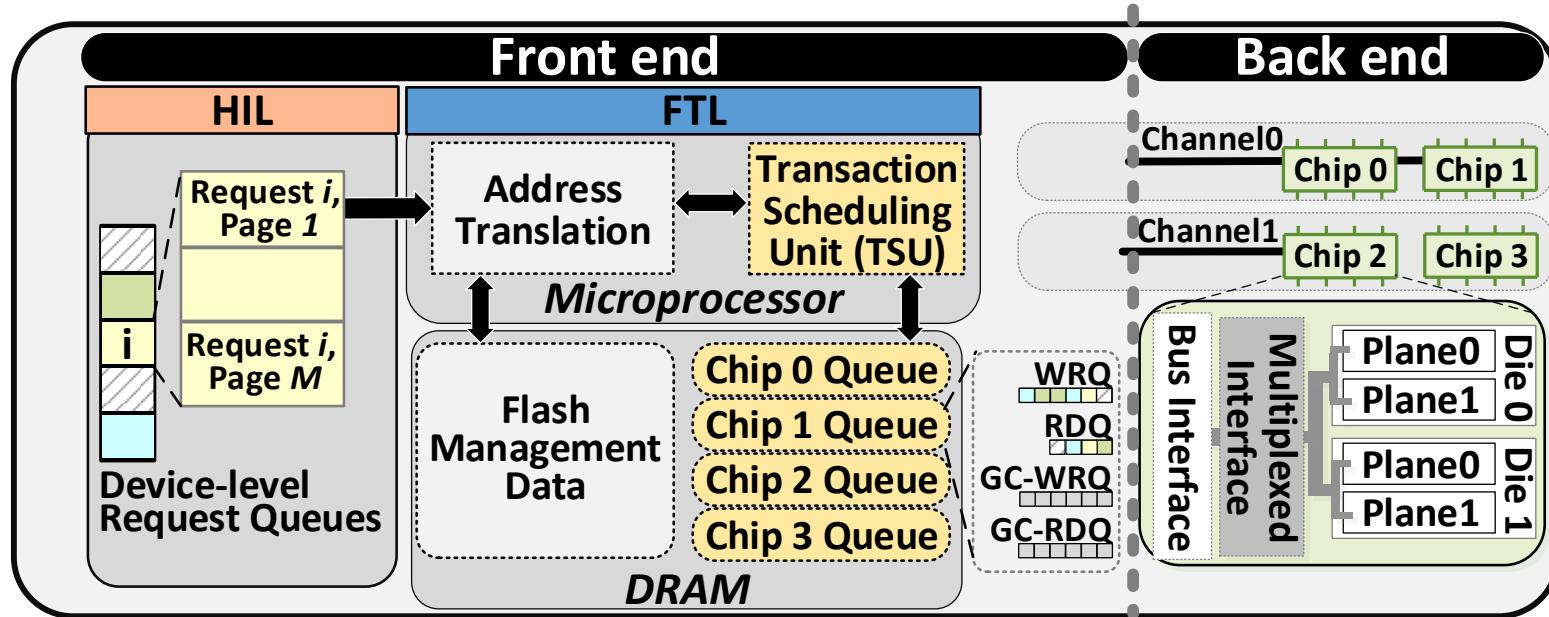
SAFARI



- **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

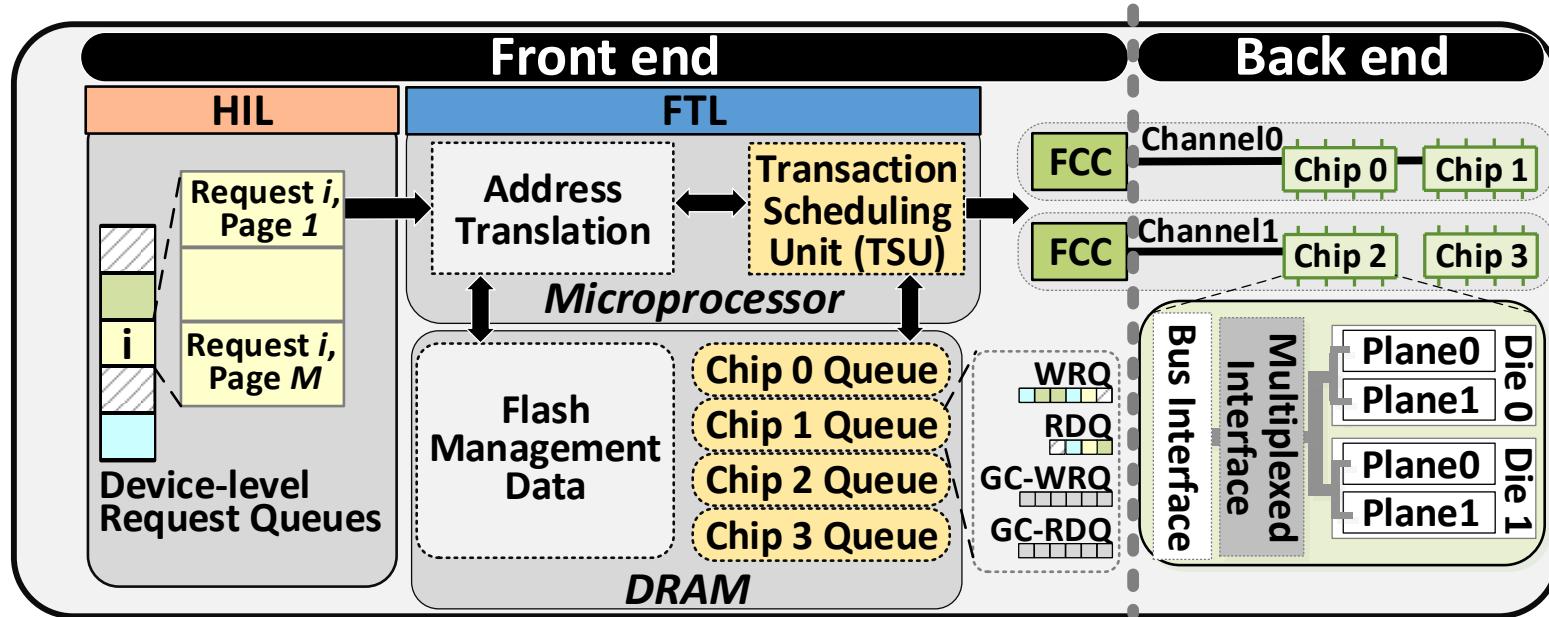
SAFARI



- **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

SAFARI

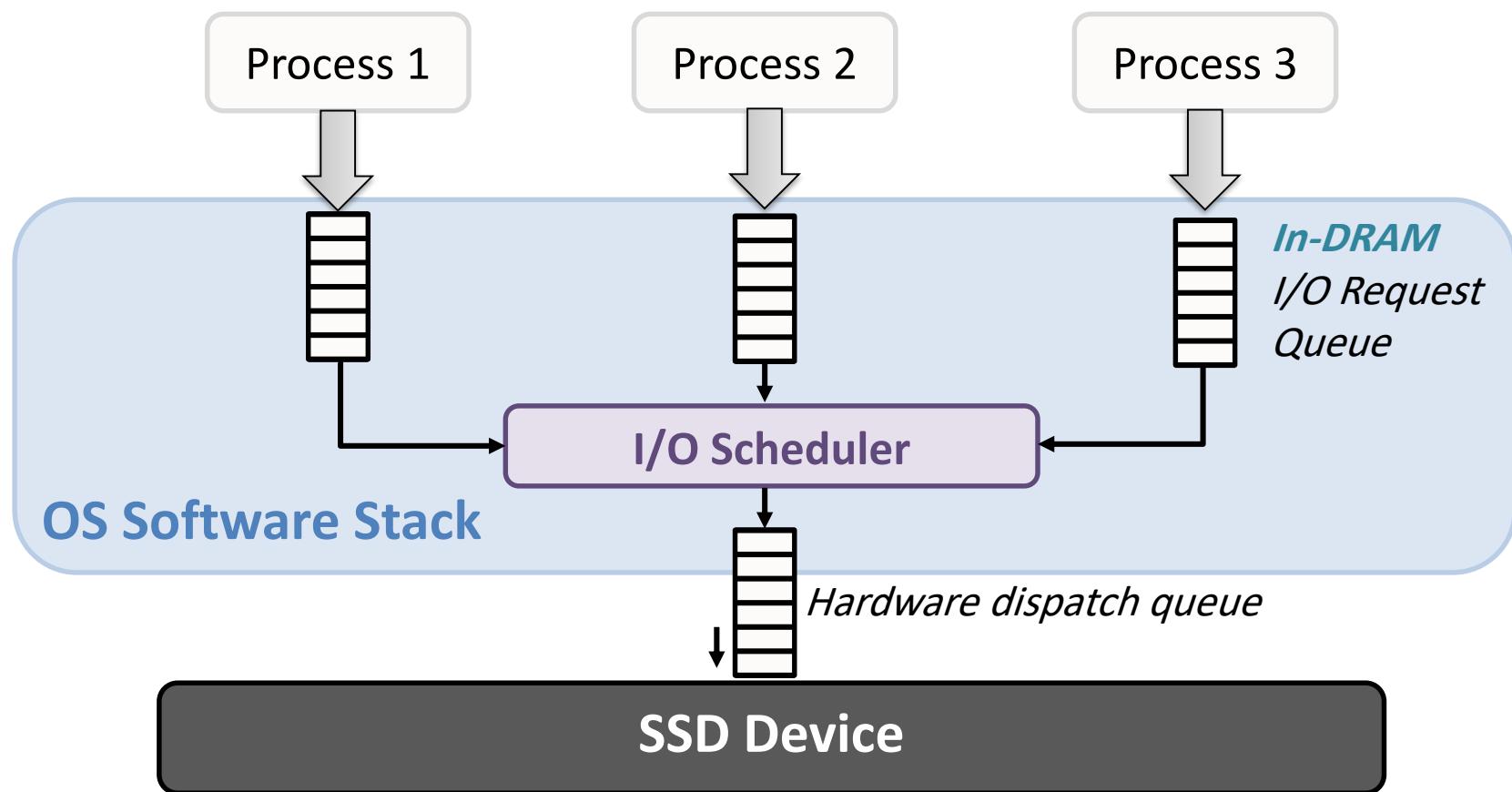


- **Back End:** data storage
 - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)
- **Front End:** management and control units
 - Host–Interface Logic (HIL): protocol used to communicate with host
 - Flash Translation Layer (FTL): manages resources, processes I/O requests
 - **Flash Channel Controllers (FCCs):** sends commands to, transfers data with memory chips in back end

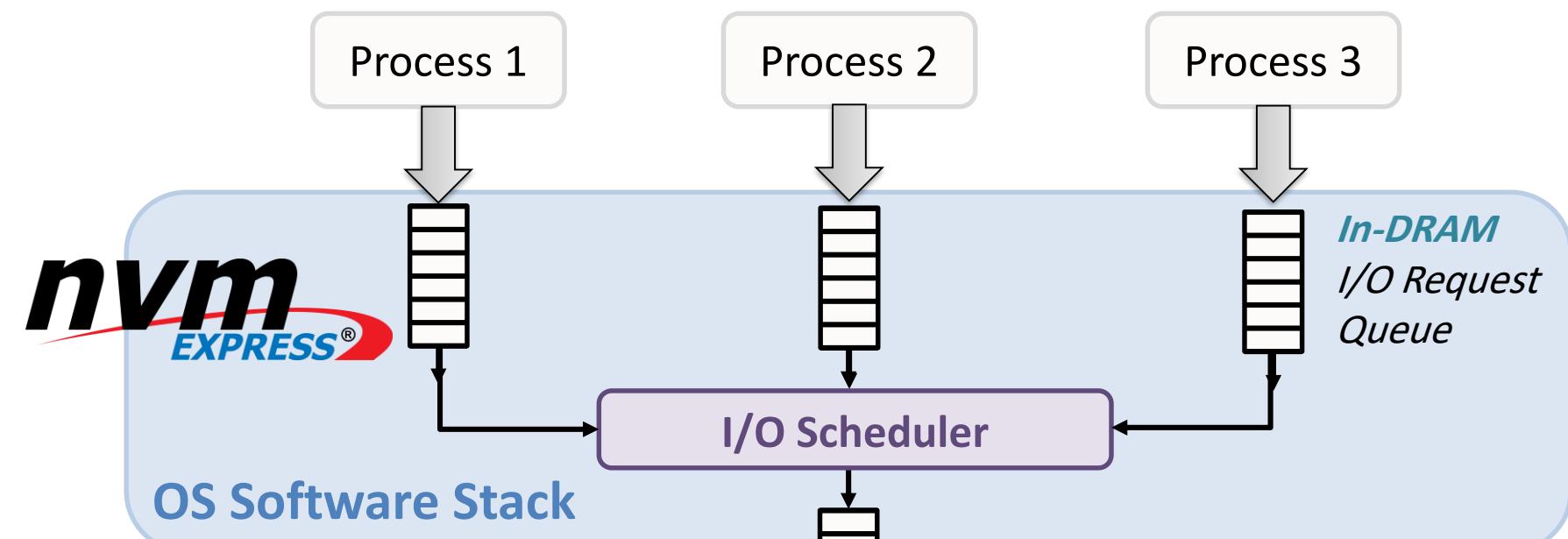
Conventional Host–Interface Protocols for SSDs

SAFARI

- 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?

Background: Modern SSD Design

Unfairness Across Multiple Applications in Modern SSDs

FLIN:

Flash-Level INterference-aware SSD Scheduler

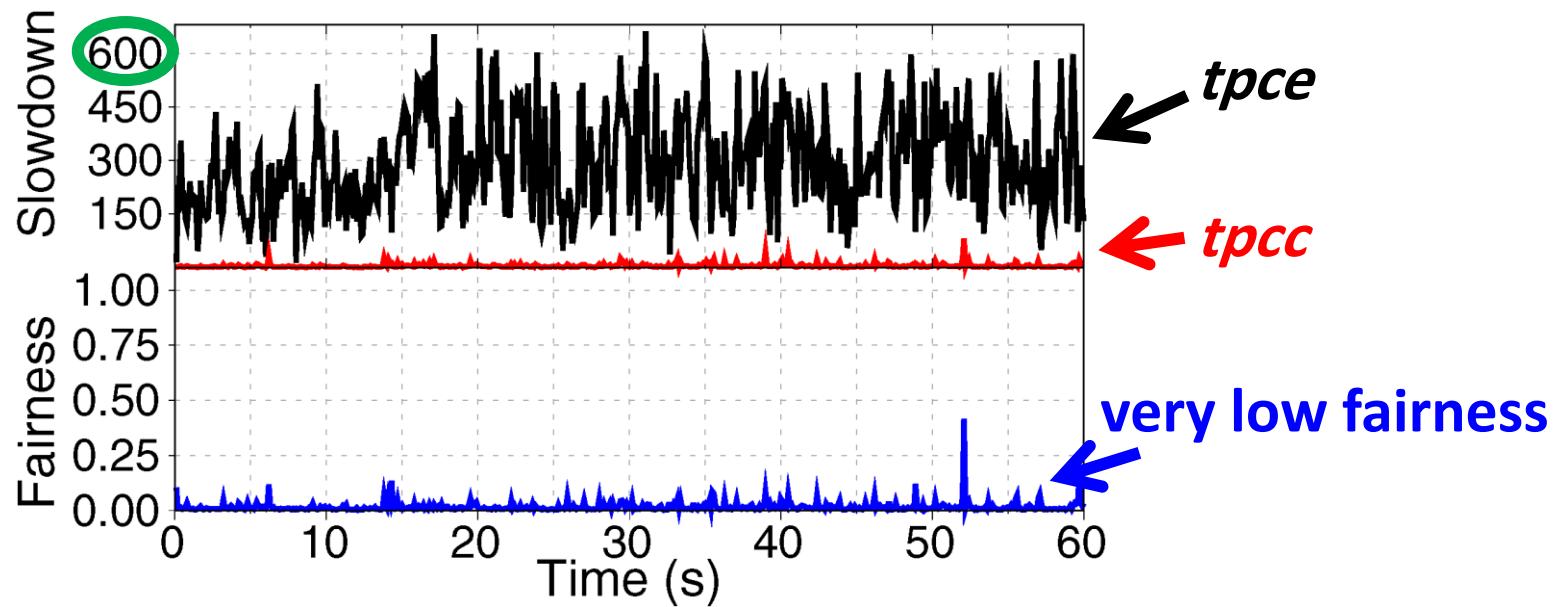
Experimental Evaluation

Conclusion

- 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
- **Slowdown** = $\frac{\text{shared flow response time}}{\text{alone flow response time}}$ *(lower is better)*
- **Unfairness** = $\frac{\text{max slowdown}}{\text{min slowdown}}$ *(lower is better)*
- **Fairness** = $\frac{1}{\text{unfairness}}$ *(higher is better)*

Representative Example: *tpcc* and *tpce*

SAFARI



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?

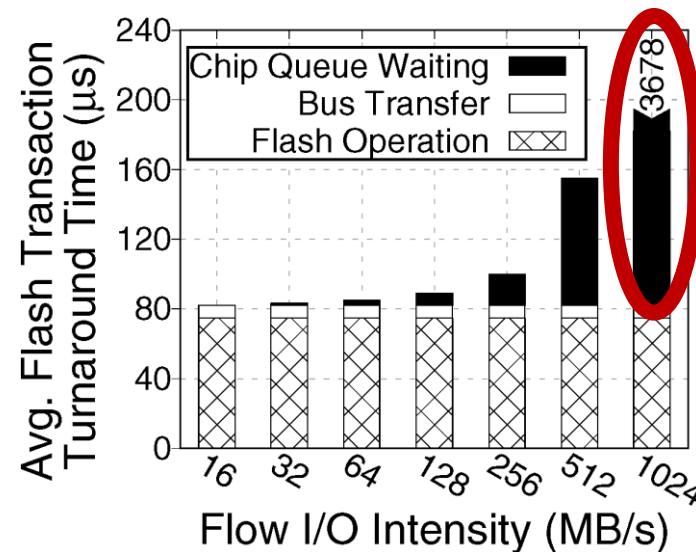
SAFARI

- 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

SAFARI

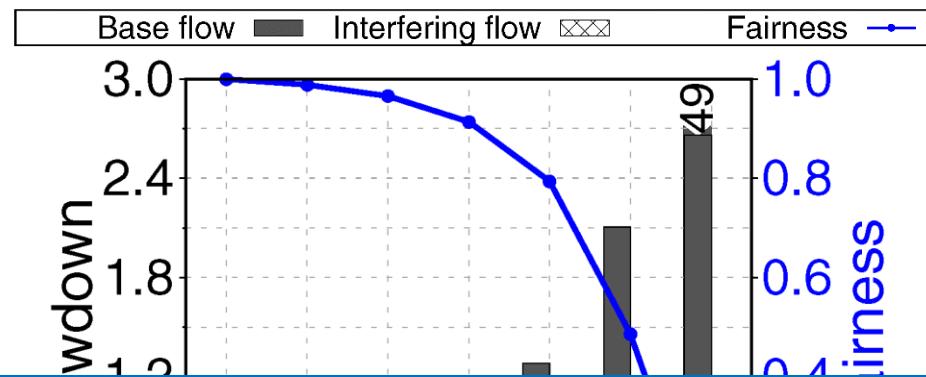
- The I/O intensity of a flow affects the average queue wait time of flash transactions



The queue wait time
highly increases with I/O intensity

Source 1: Different I/O Intensities

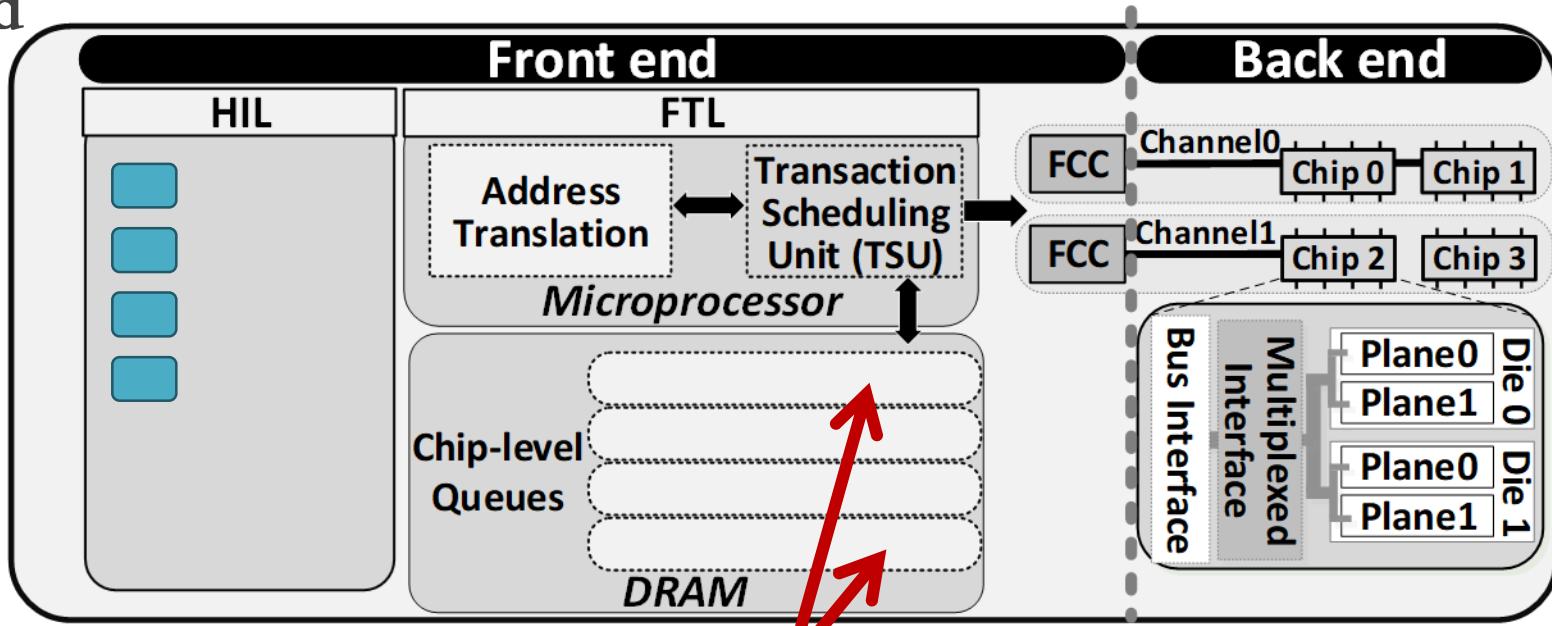
- An experiment to analyze the effect of concurrently executing two flows with **different I/O intensities** on fairness
 - Base flow: low intensity (16 MB/s) and **low** average chip-level queue length
 - Interfering flow: varying I/O intensities from **low** to very **high**



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



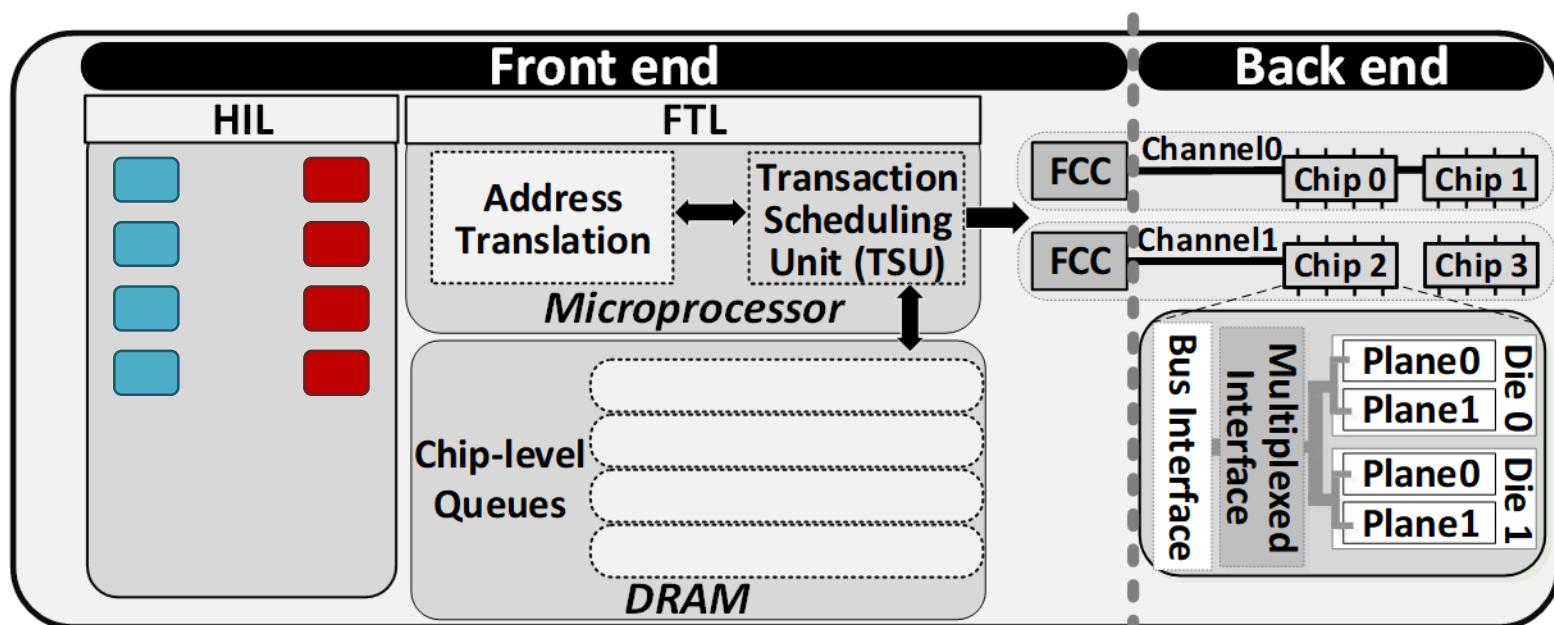
Even distribution of transactions in chip-level queues

- Leads to a **low queue wait time**

Source 2: Different Request Access Patterns

SAFARI

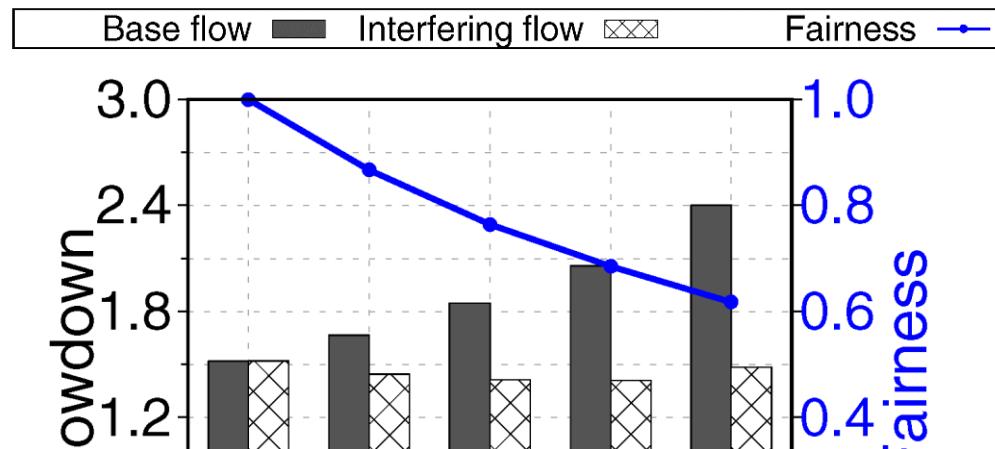
- Other flows have access patterns that **do not exploit parallelism**



Source 2: Different Request Access Patterns

SAFARI

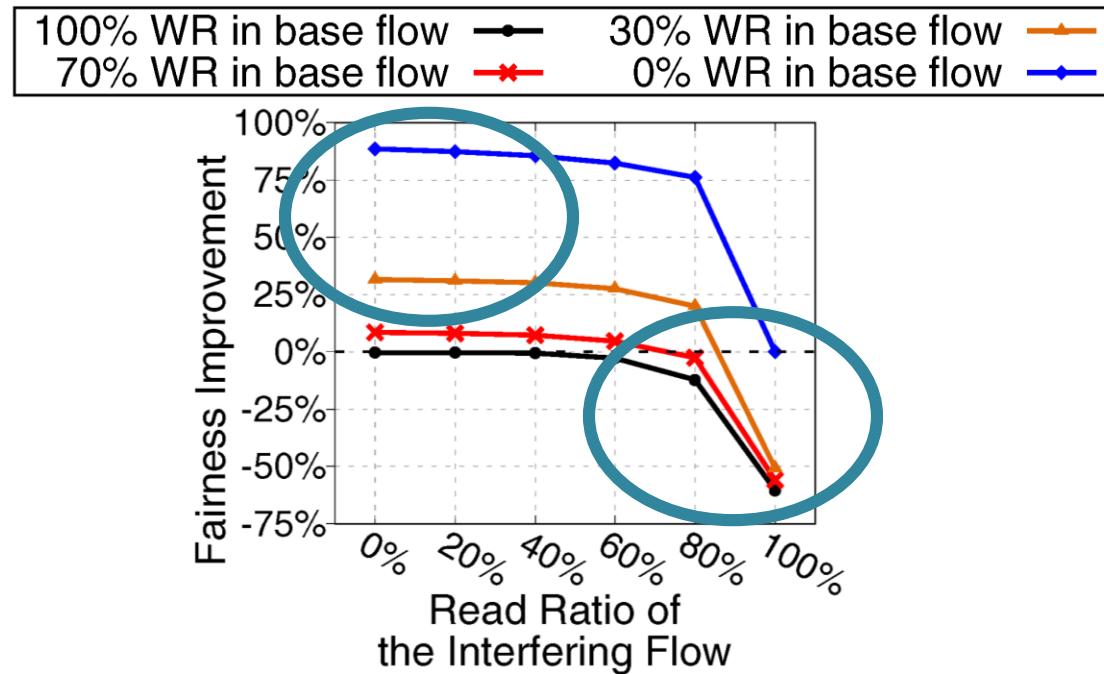
- An experiment to analyze the interference between concurrent flows with **different access patterns**
 - Base flow: **streaming** access pattern (parallelism friendly)
 - Interfering flow: **mixed** streaming and random access pattern



Flows with **parallelism-friendly** access patterns
are **susceptible to interference** from
flows whose access patterns do not exploit parallelism

Source 3: Different Read/Write Ratios

- State-of-the-art SSD I/O schedulers prioritize reads over writes



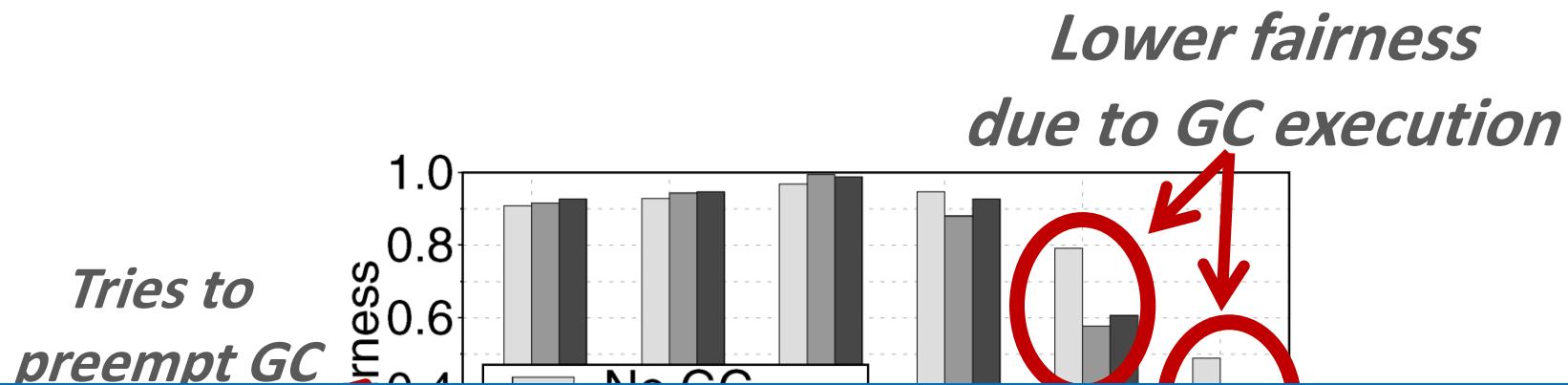
When flows have different read/write ratios,
existing schedulers do not effectively provide fairness

Source 4: Different Garbage Collection Demands **SAFARI**

- 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 SAFARI

- **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**



The GC activities of a **high-GC** flow can unfairly block flash transactions of a **low-GC** flow

- 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

Background: Modern SSD Design

Unfairness Across Multiple Applications in Modern SSDs

FLIN:

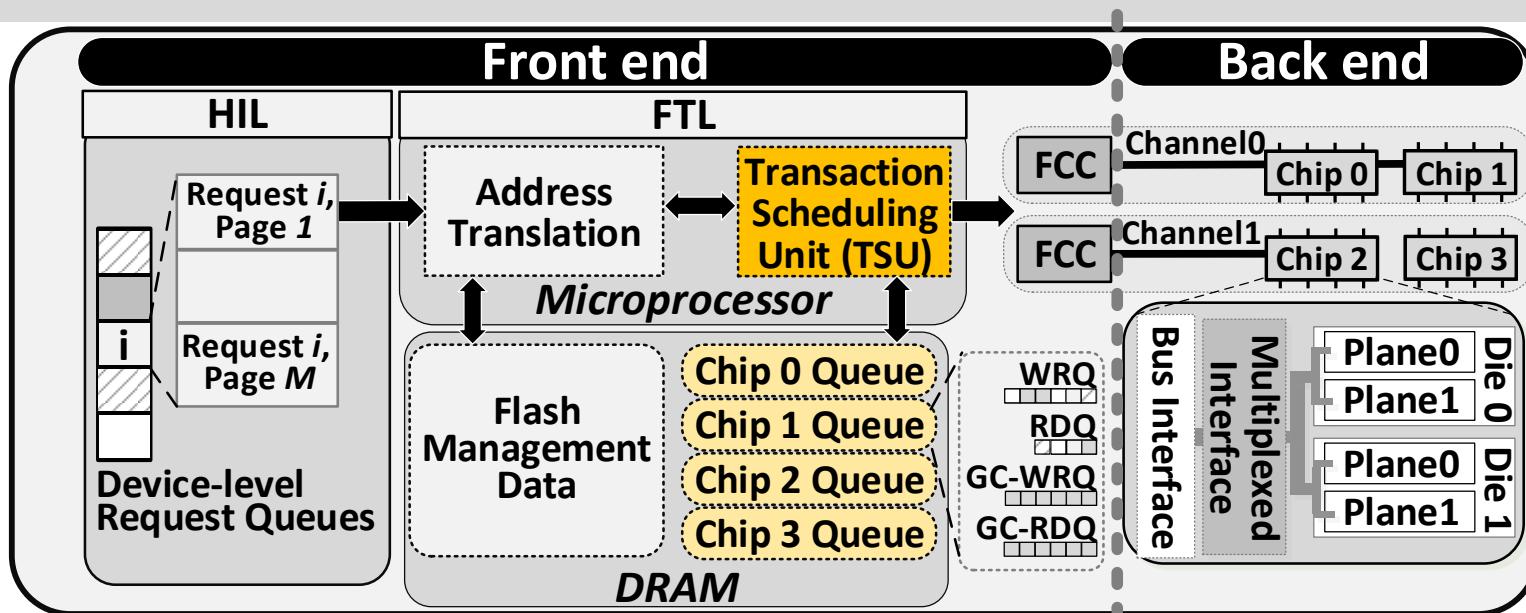
Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion

FLIN: Flash-Level INterference-aware Scheduler

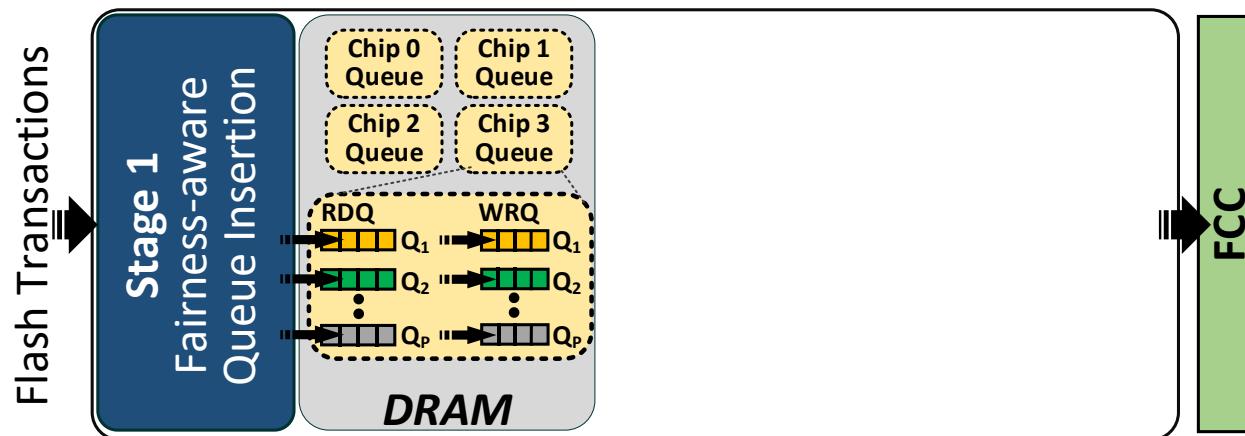
SAFARI



- 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

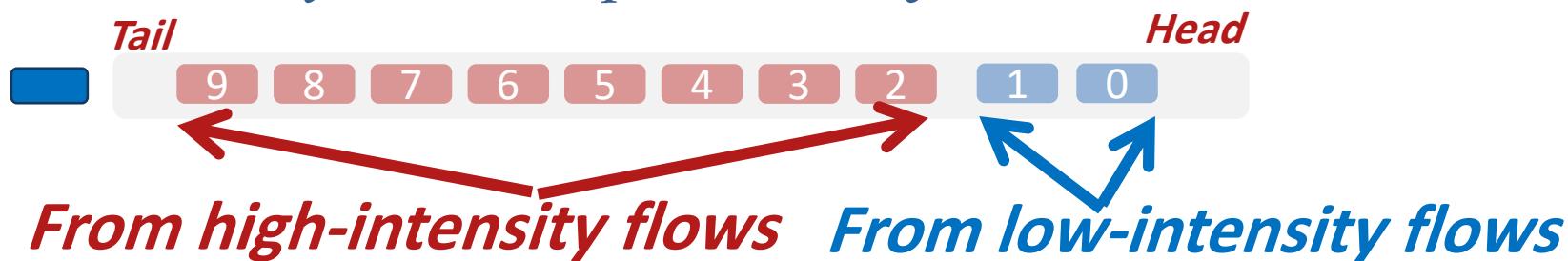
Three Stages of FLIN

SAFARI



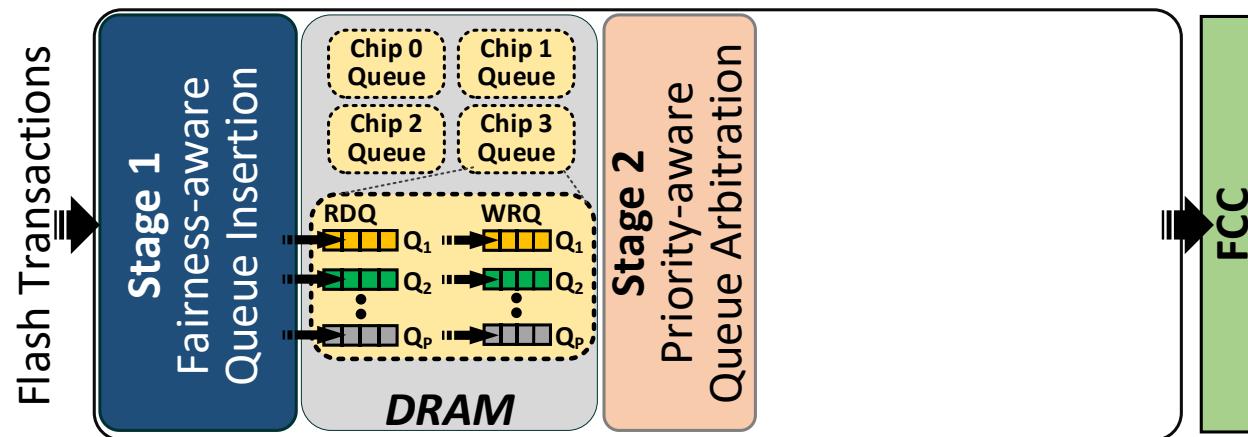
- Stage 1: Fairness-aware Queue Insertion

relieves I/O intensity and access pattern interference



Three Stages of FLIN

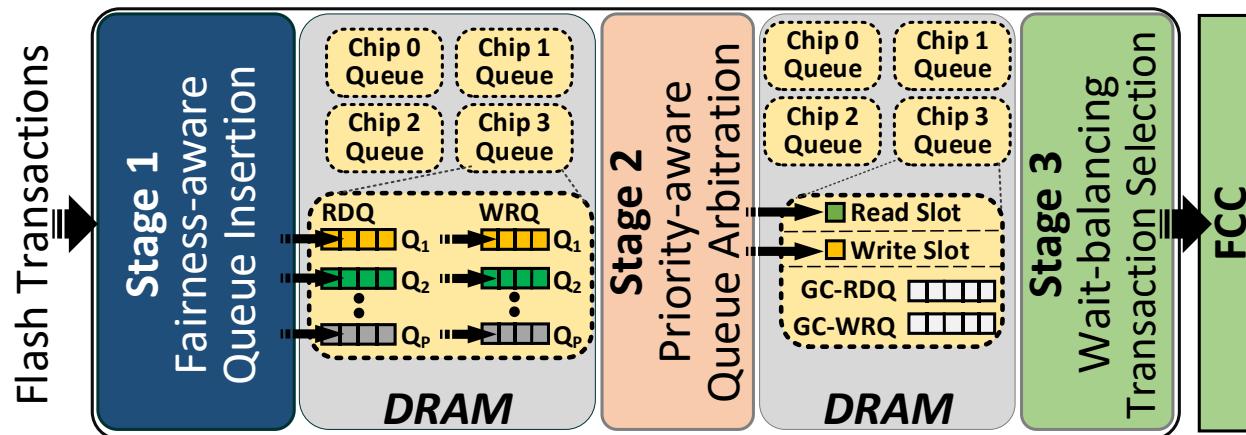
SAFARI



- 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

SAFARI



- **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

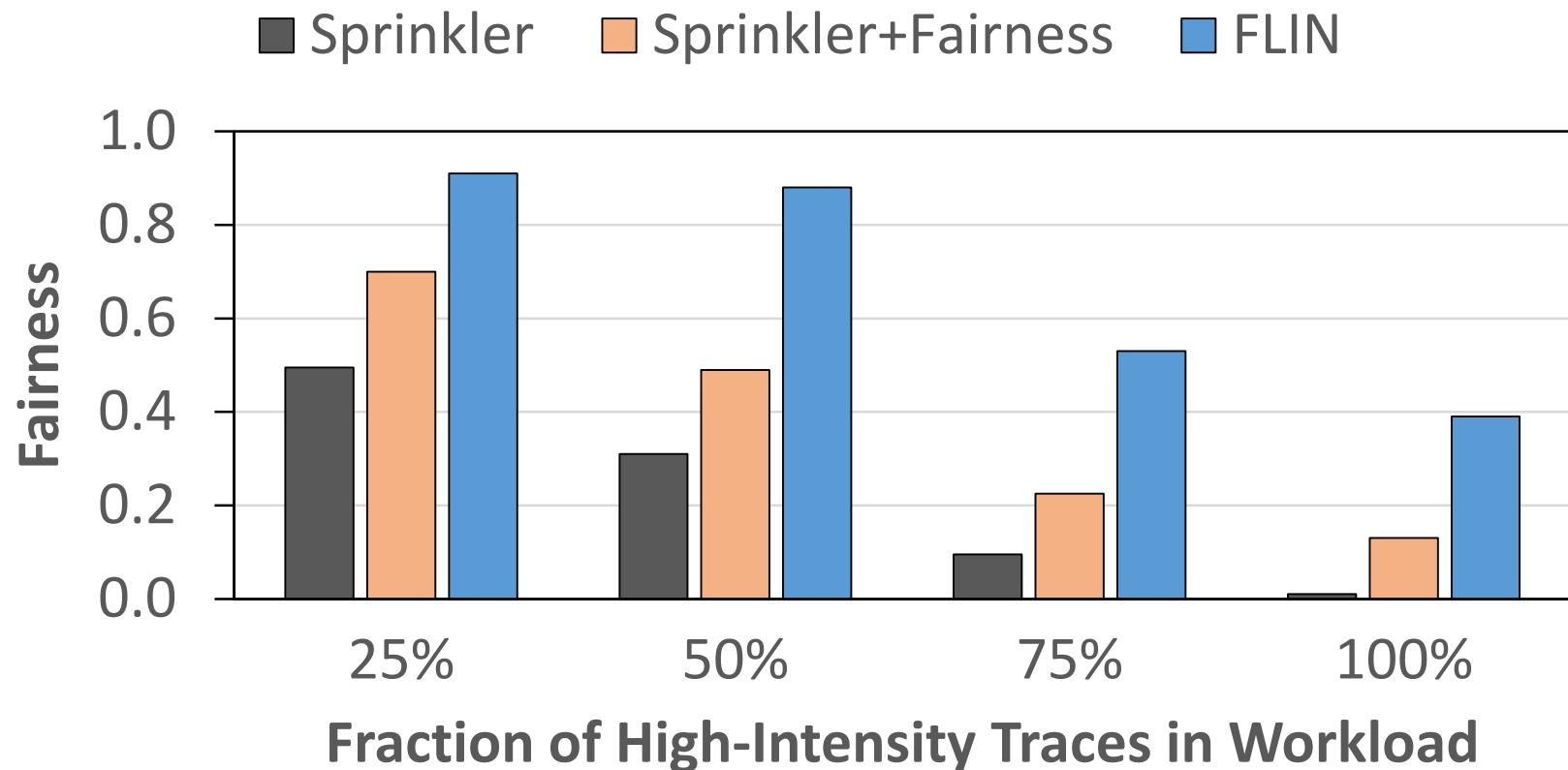
Conclusion

- 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

- **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 Over the Baselines

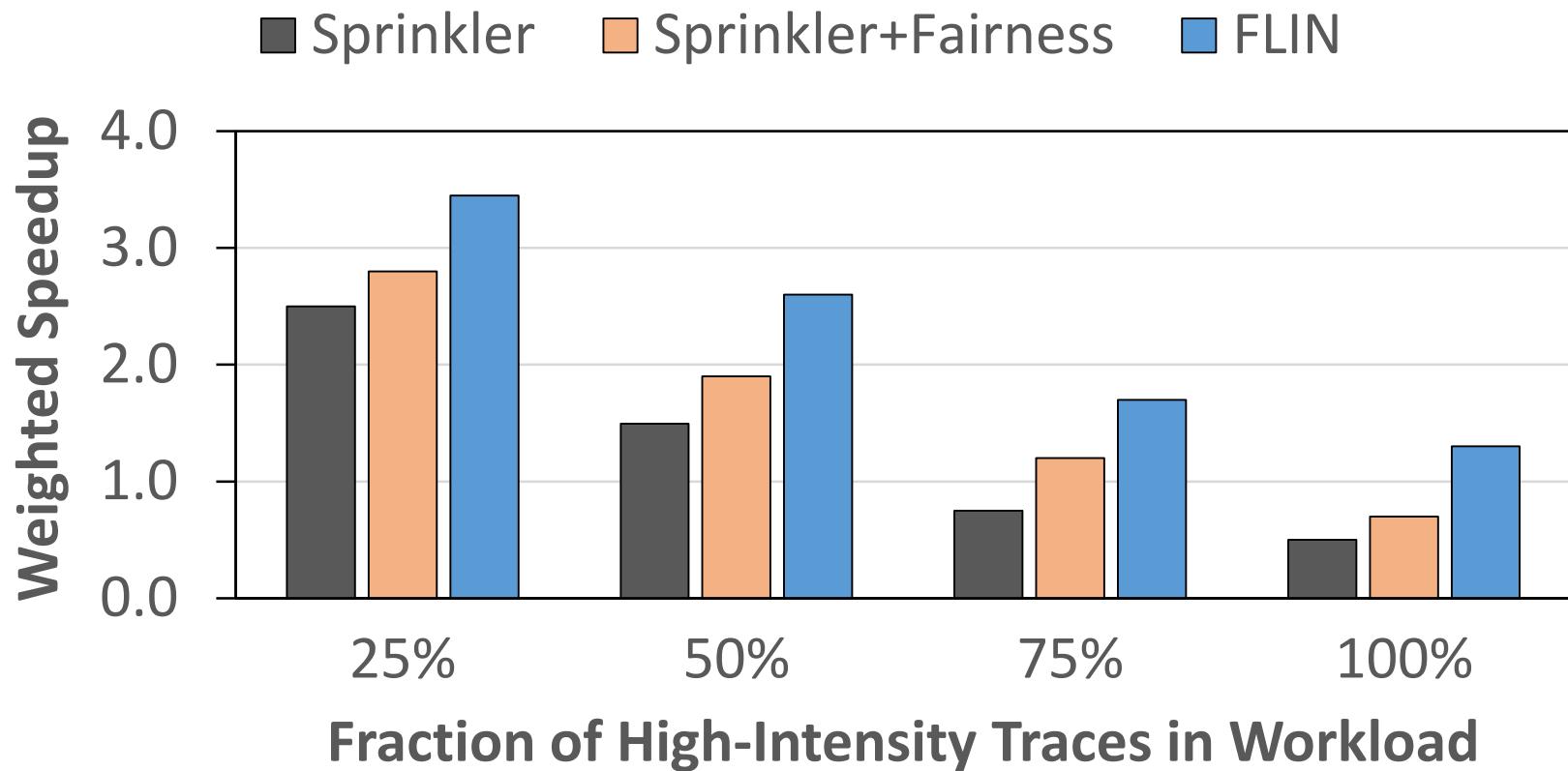
SAFARI



FLIN improves fairness by an average of 70%,
by mitigating *all* four major sources of interference

FLIN Improves Performance Over the Baselines

SAFARI



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

- 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

Computer Architecture

Lecture 17: Flash Memory and Solid-State Drives II

Dr. Mohammad Sadrosadati

Prof. Onur Mutlu

ETH Zürich

Fall 2023

23 November 2023

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

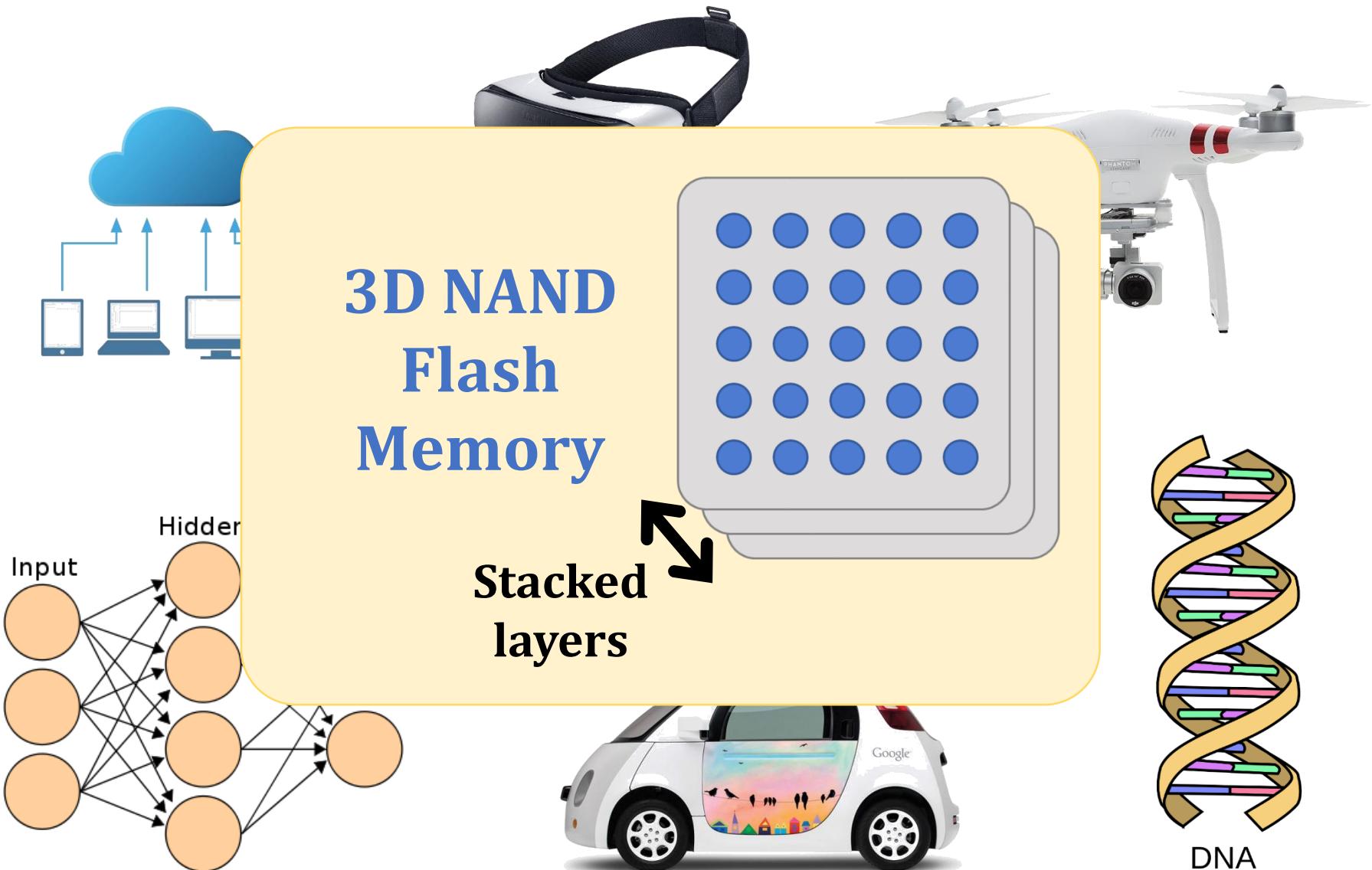
Carnegie Mellon

SAFARI



ETH Zürich

Storage Technology Drivers - 2018



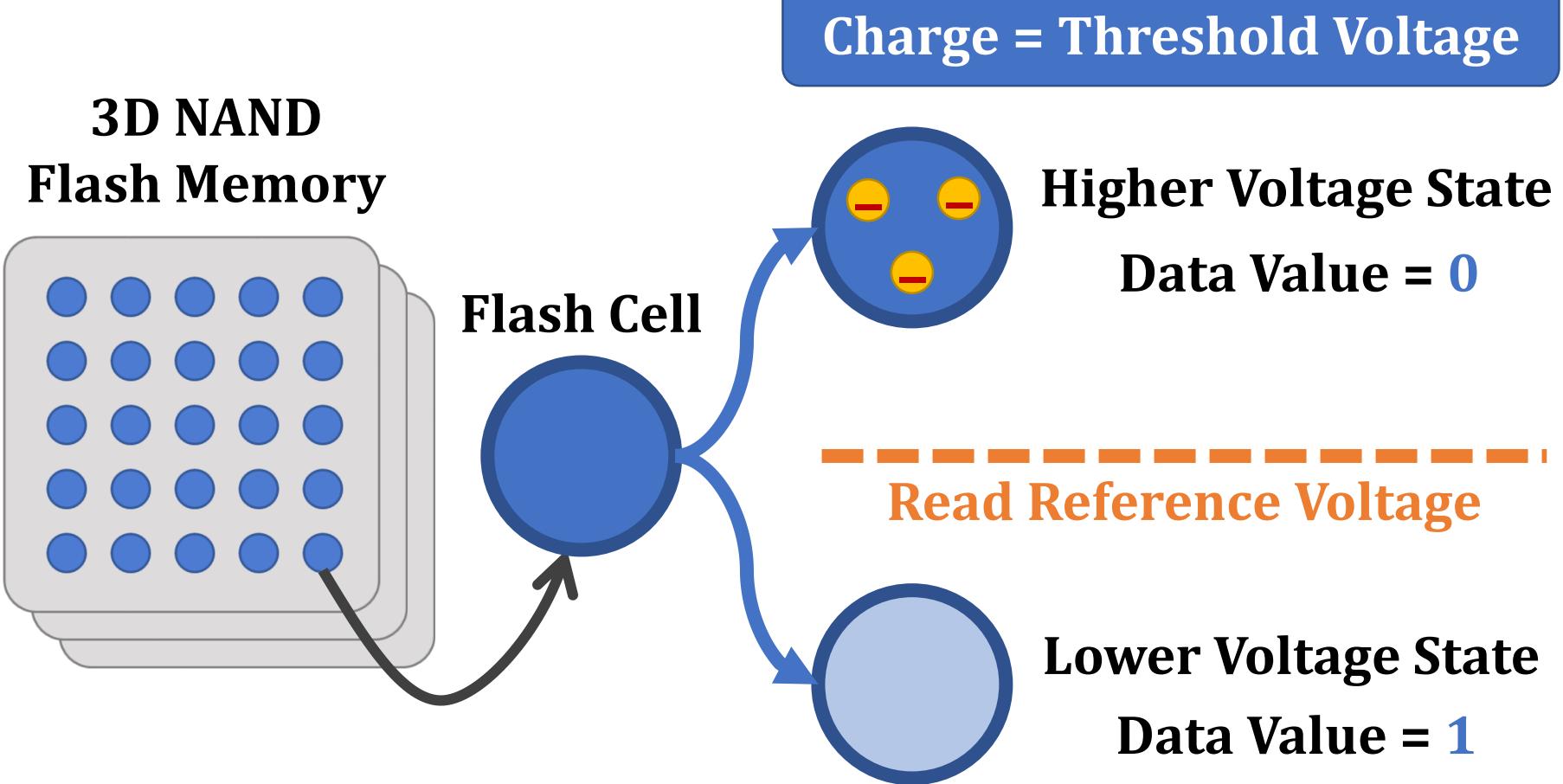
Executive Summary

- 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

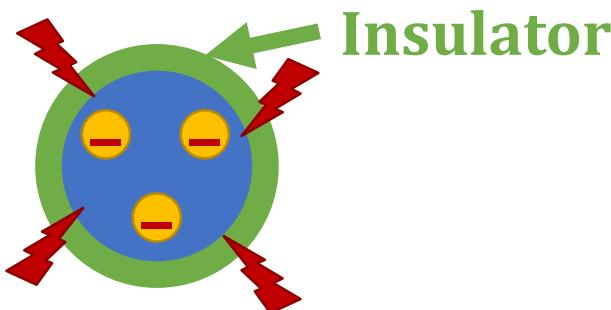
- Executive Summary
- **Background on NAND Flash Reliability**
- Characterization of Self-Recovery and Temperature Effect on Real 3D NAND Flash Memory Chips
- URT: Unified Self-Recovery and Temperature Model
- HeatWatch Mechanism
- Conclusion

3D NAND Flash Memory Background



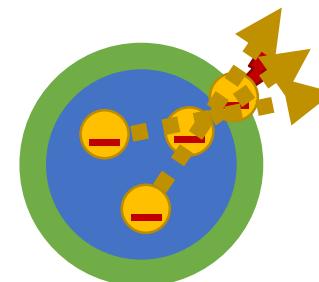
Flash Wearout

Program/Erase (P/E) → Wearout



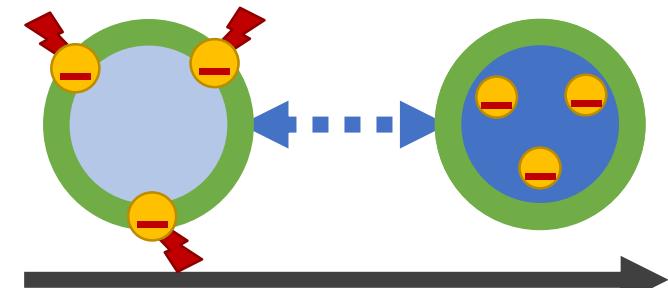
Wearout Effects:

1. **Retention Loss**
(voltage shift over time)



2. **Program Variation**
(init. voltage difference b/w states)

Wearout Introduces Errors



Voltage

Improving Flash Lifetime

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

**Two Ways to Improve
Flash Lifetime**

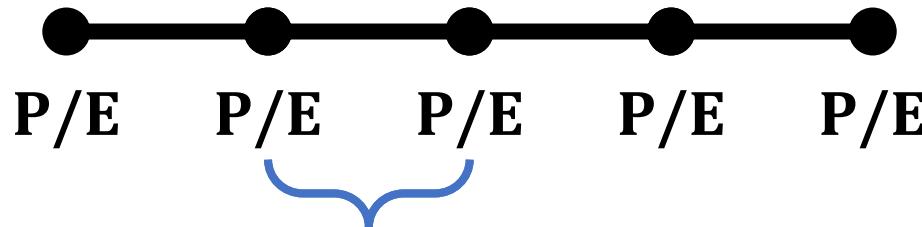


**Exploiting the
Self-Recovery Effect**

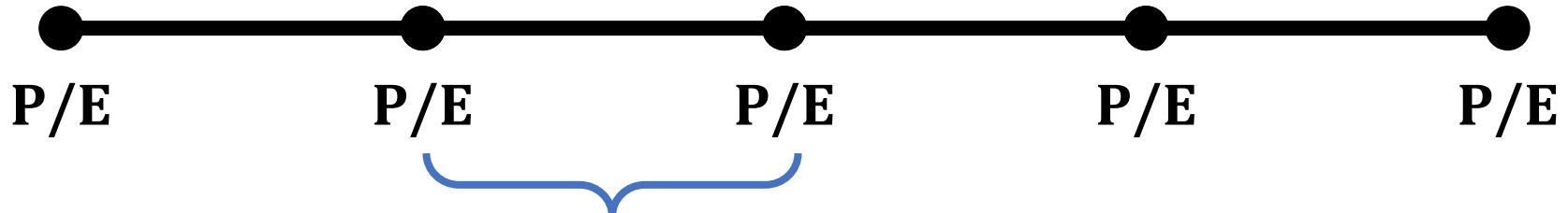
**Exploiting the
Temperature Effect**

Exploiting the Self-Recovery Effect

Partially repairs damage due to wearout



Dwell Time: Idle Time Between P/E Cycles

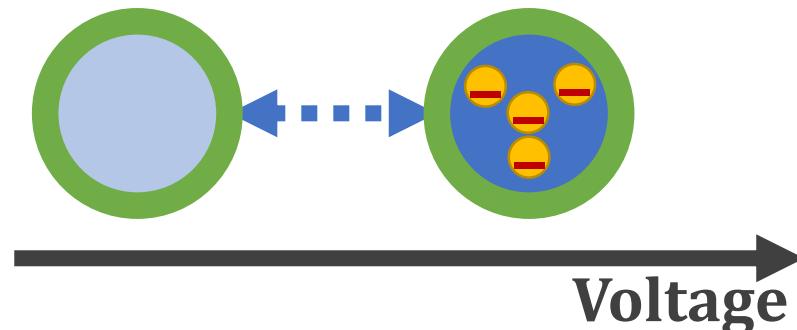


Longer Dwell Time: More Self-Recovery

Reduces Retention Loss

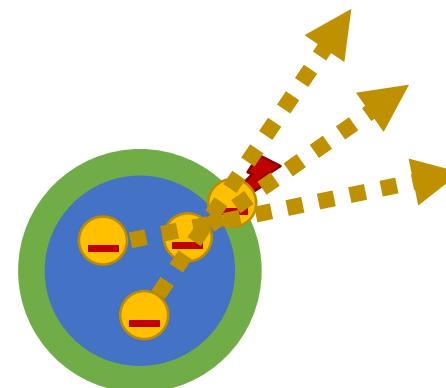
Exploiting the Temperature Effect

**High Program
Temperature**



Increases Program Variation

**High Storage
Temperature**



Accelerates Retention Loss

Prior Studies of Self-Recovery/Temperature

Self-Recovery Effect

Planar (2D) NAND



3D NAND



Temperature Effect

JEDEC 2010
(no characterization)



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

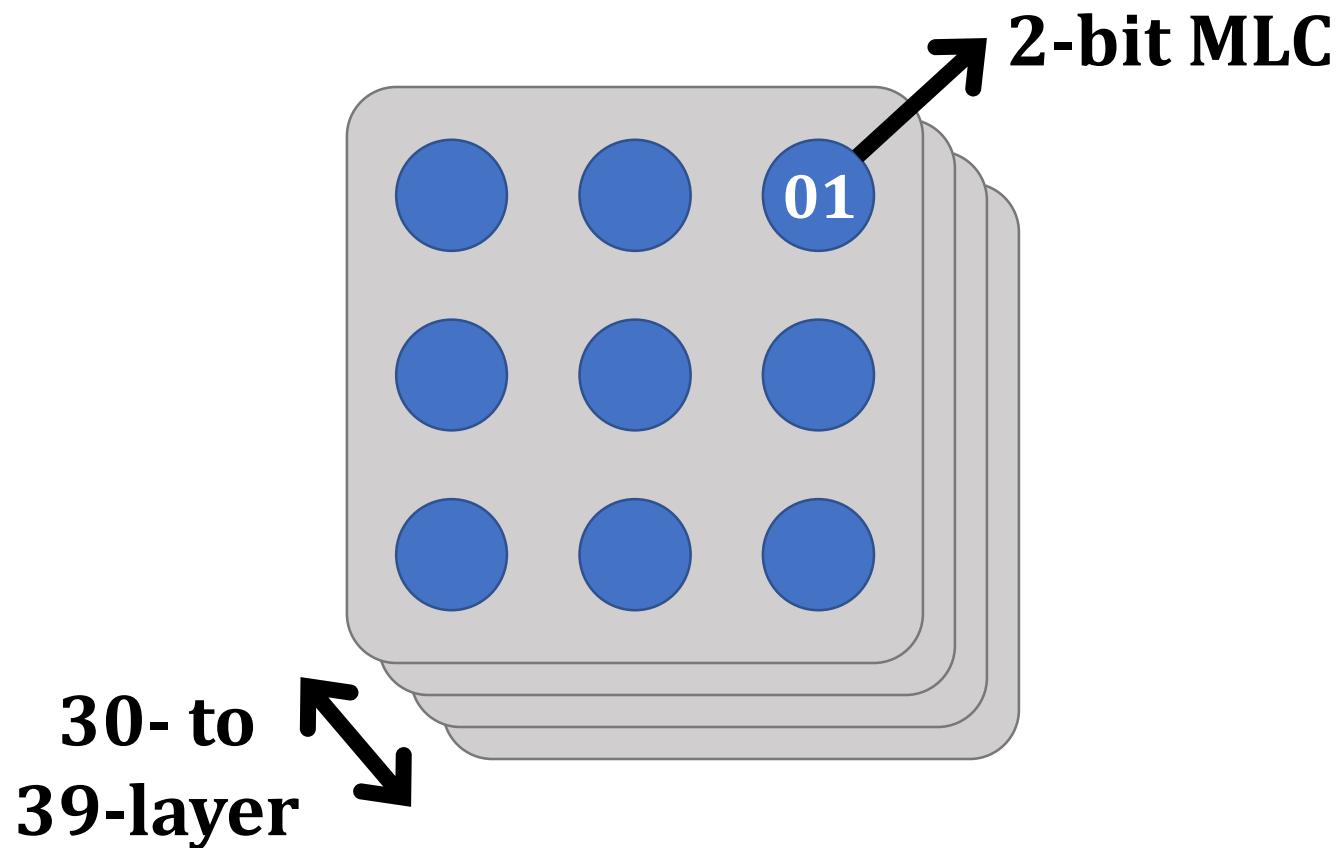
Characterization Methodology

- 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

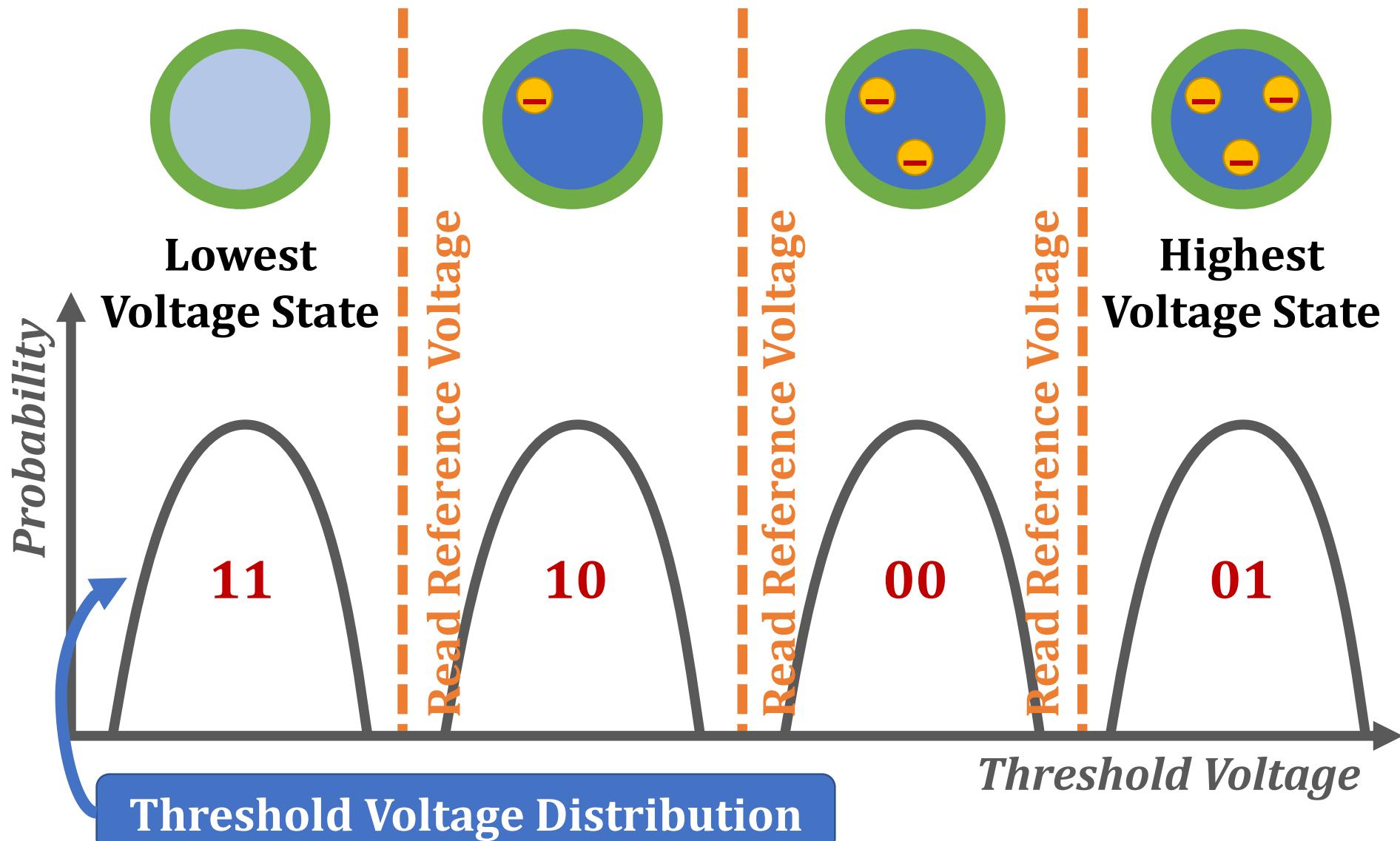


Characterized Devices

Real 30-39 Layer 3D MLC NAND Flash Chips



MLC Threshold Voltage Distribution Background



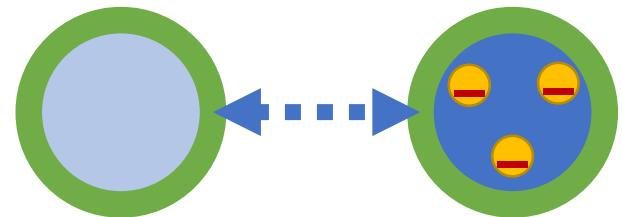
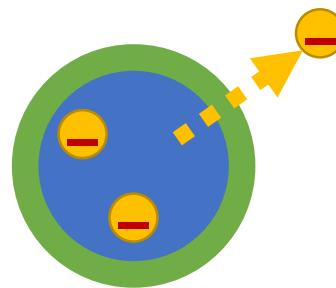
Characterization Goal

Characterized
Metrics

Retention Loss Speed
(how fast voltage shifts
over time)

Characterized
Phenomena

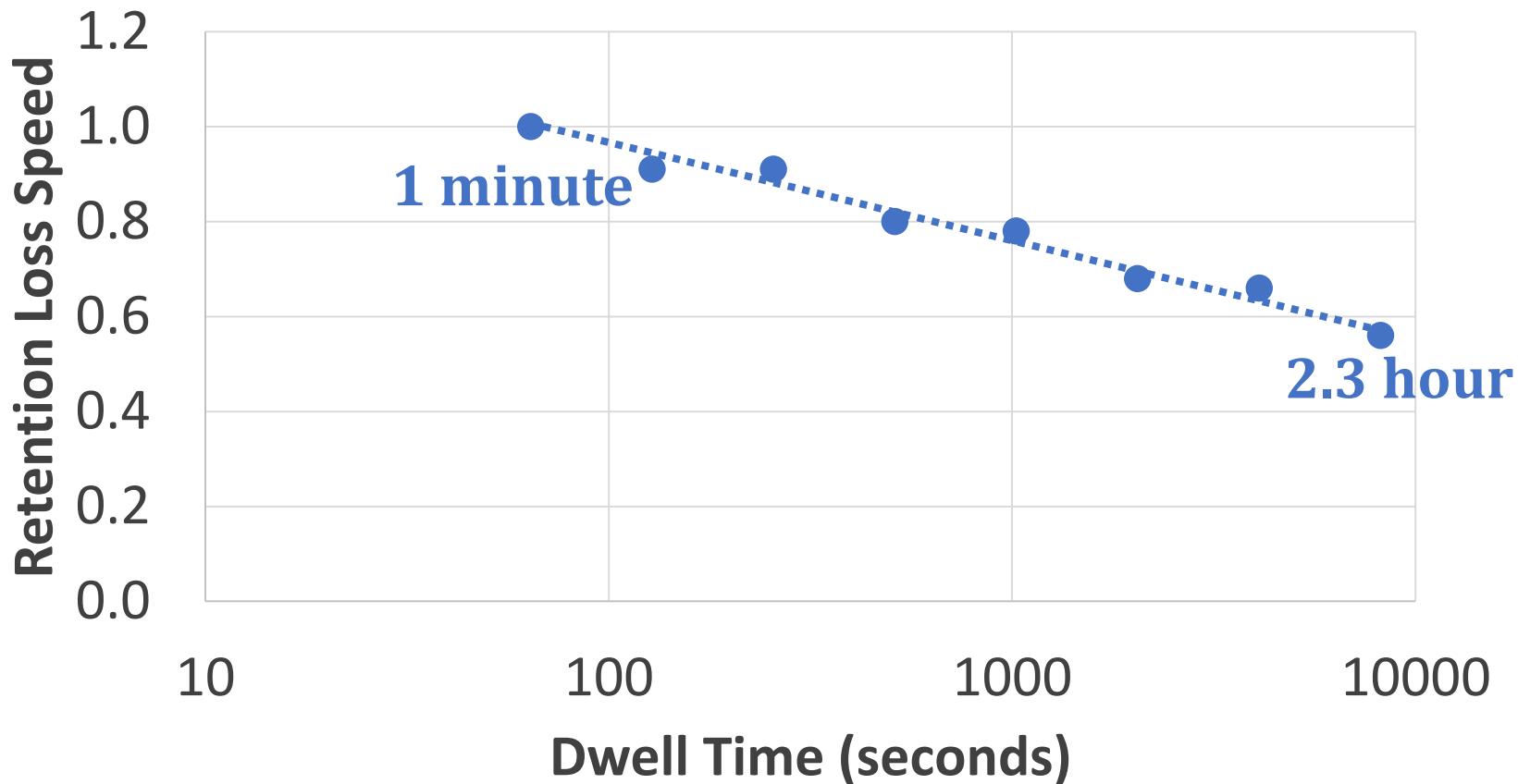
Self-Recovery
Effect



Program Variation
(initial voltage difference
between states)

Temperature
Effect

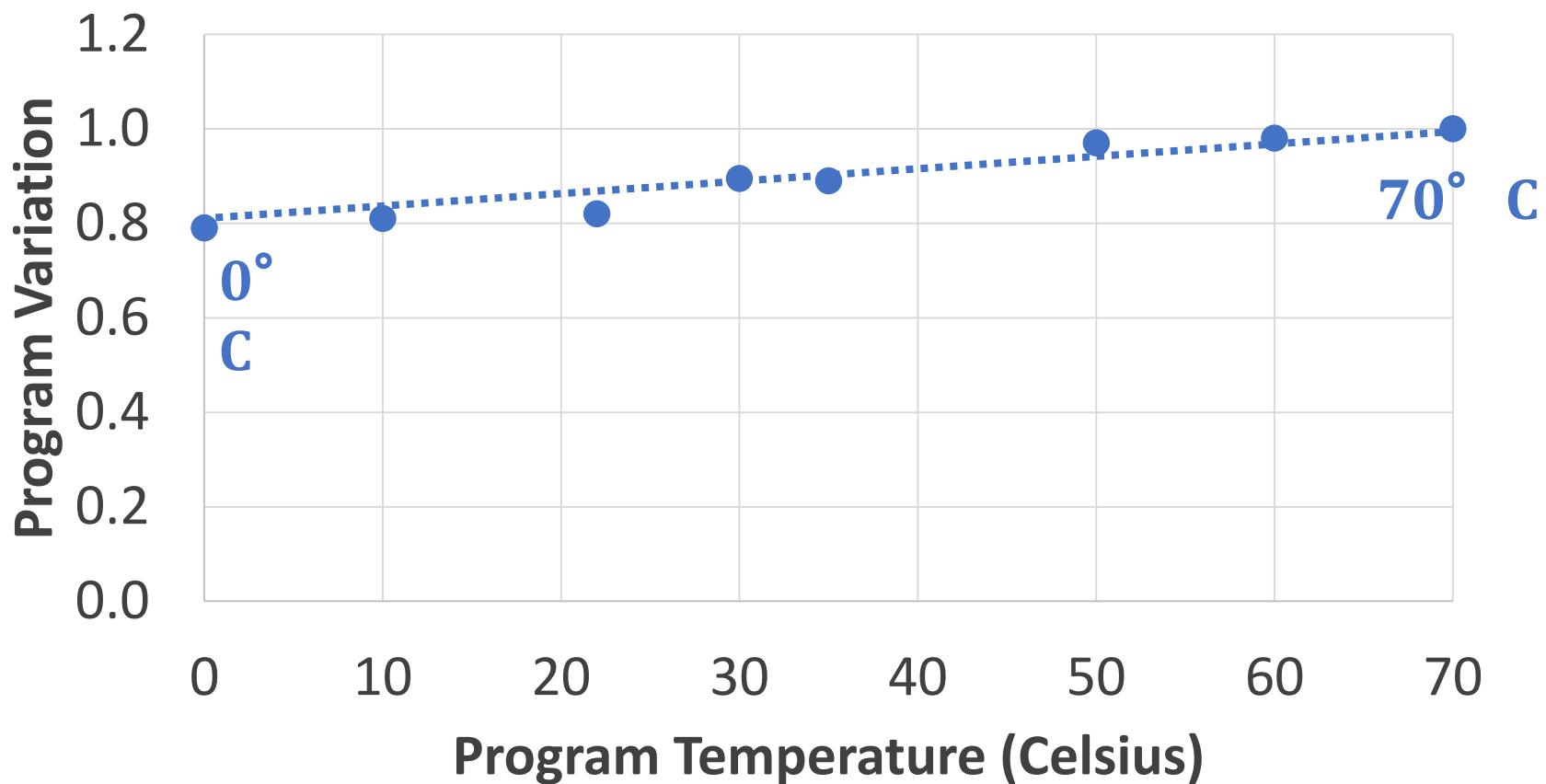
Self-Recovery Effect Characterization Results



Dwell time: Idle time between P/E cycles

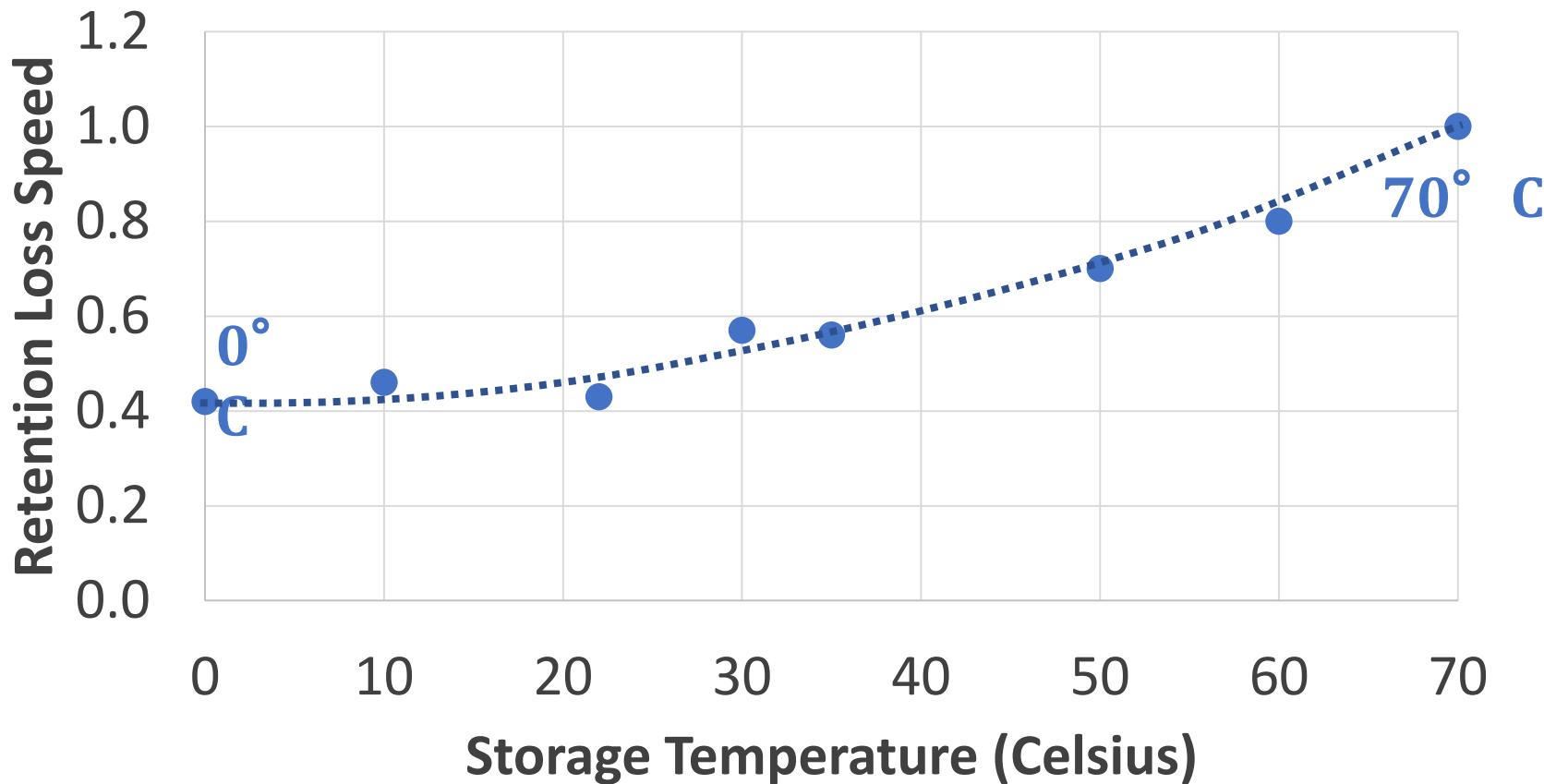
Increasing dwell time from 1 minute to 2.3 hours
slows down retention loss speed by 40%

Program Temperature Effect Characterization Results



Increasing program temperature from 0°C to 70°C improves program variation by 21%

Storage Temperature Effect Characterization Results



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

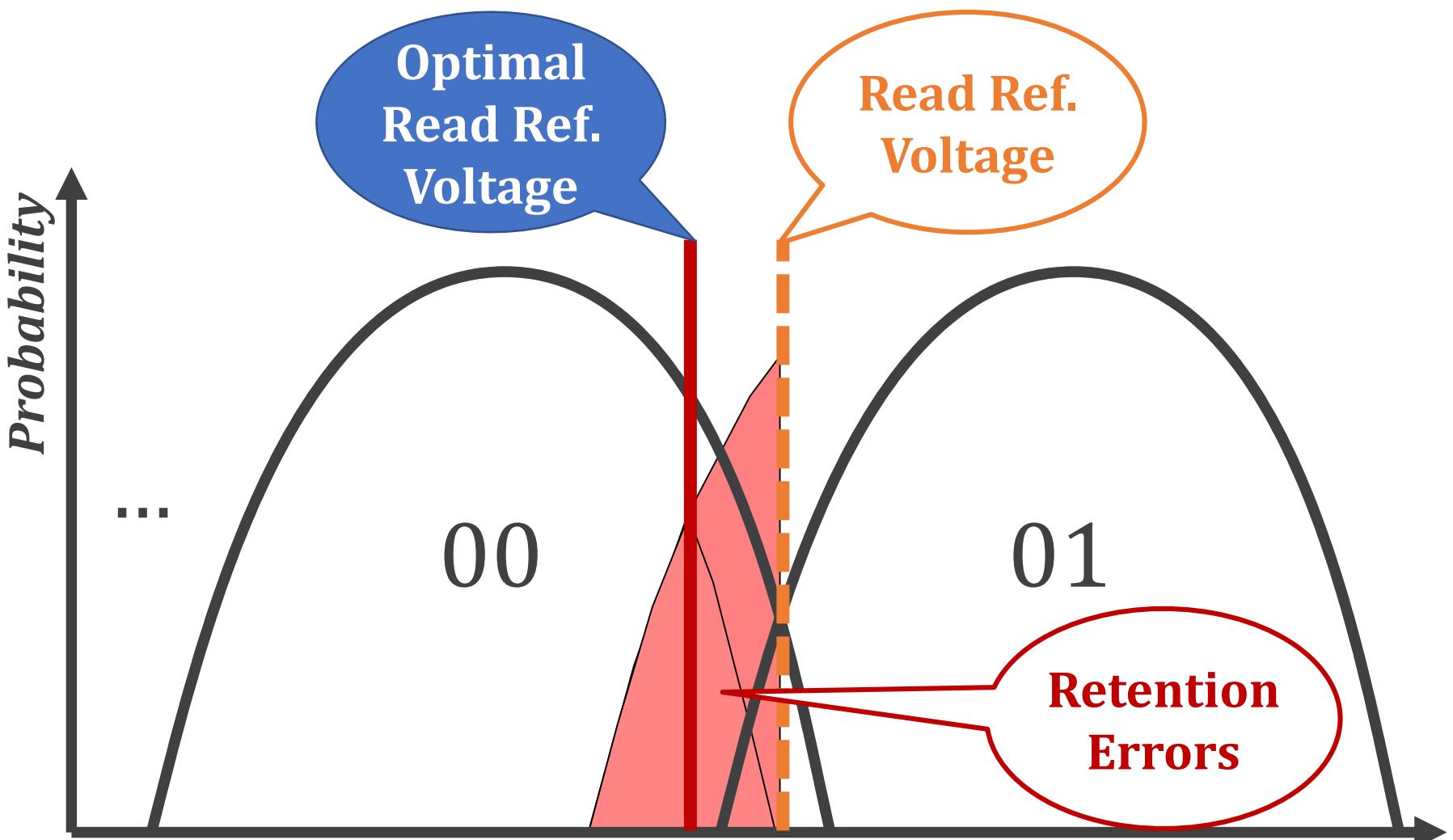
Other Characterizations Methods in the Paper:

- 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:

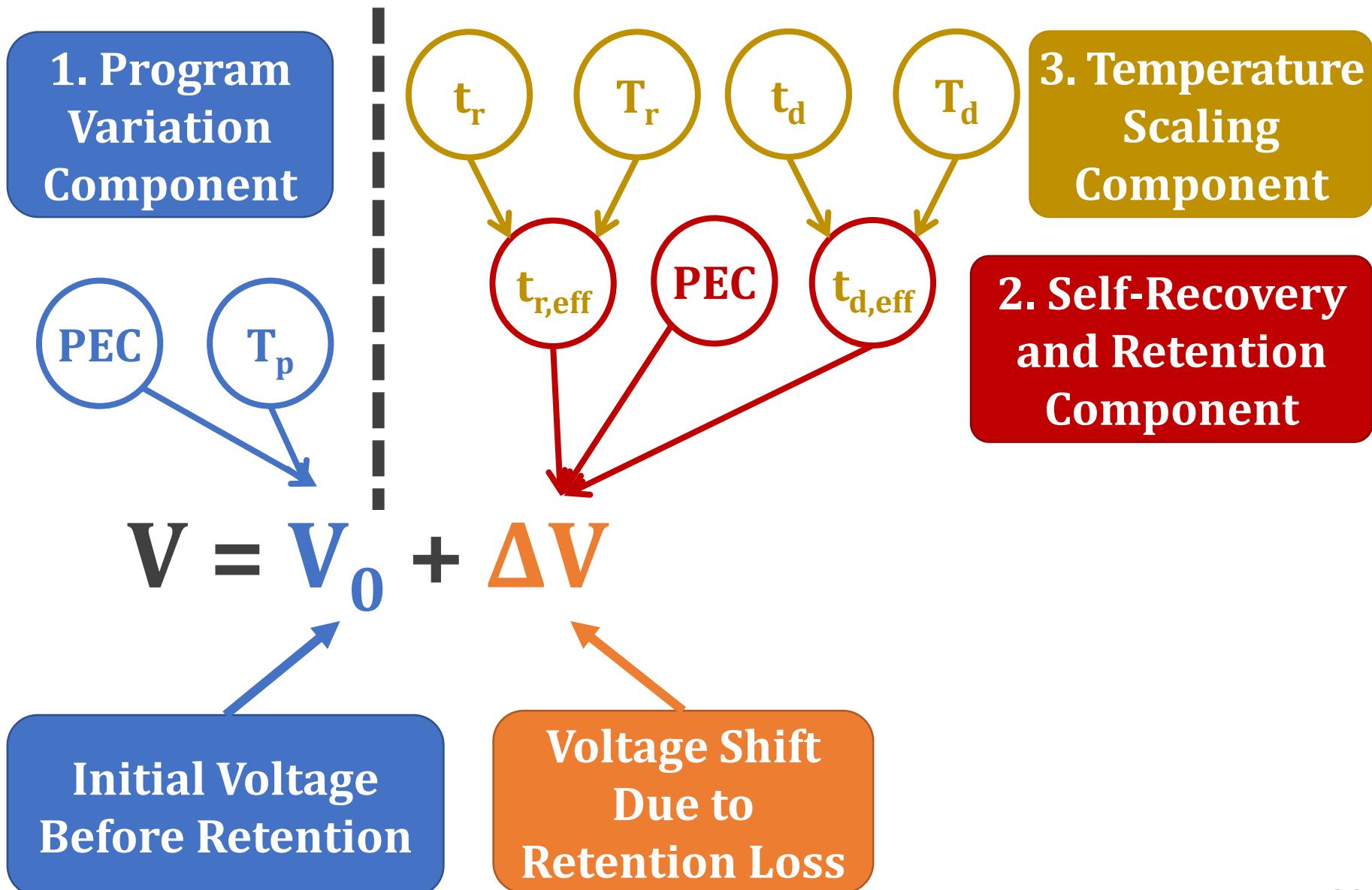
$$V = V_0 + \Delta V$$

Mean
Threshold
Voltage

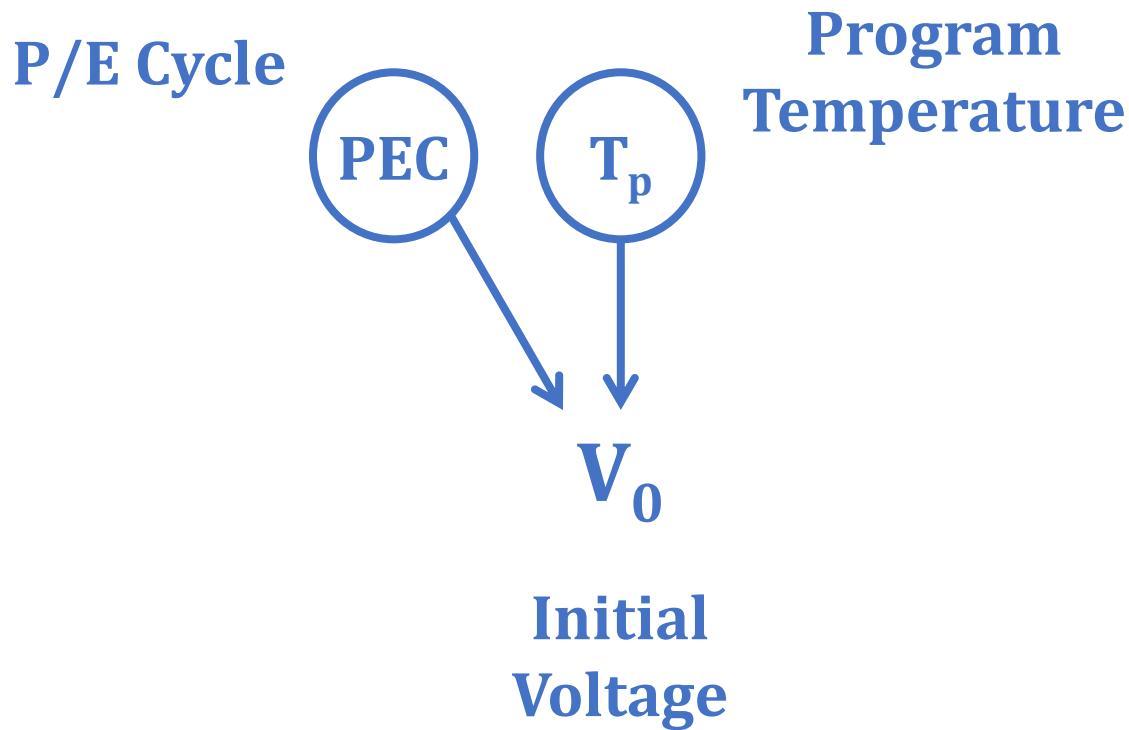
Initial Voltage
Before Retention
(Program Variation)

Voltage Shift
Due to
Retention Loss

URT Model Overview



1. Program Variation Component

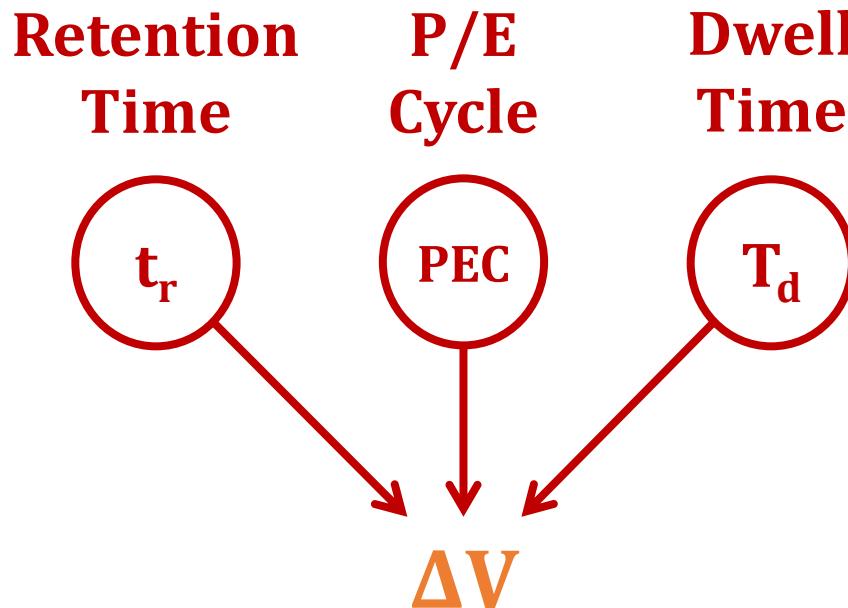


$$V_0 = A \cdot T_p \cdot PEC + B \cdot T_p + C \cdot PEC + D$$



Validation: $R^2 = 91.7\%$

2. Self-Recovery and Retention Component



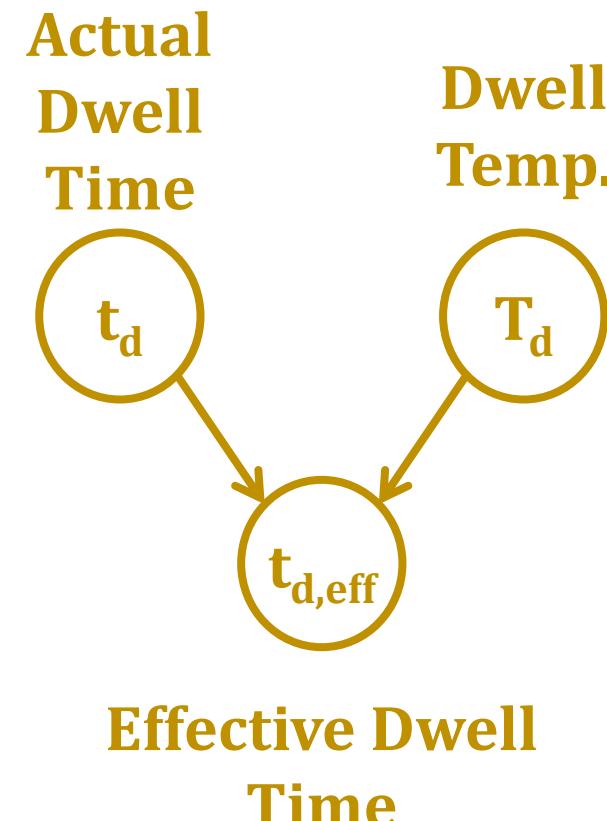
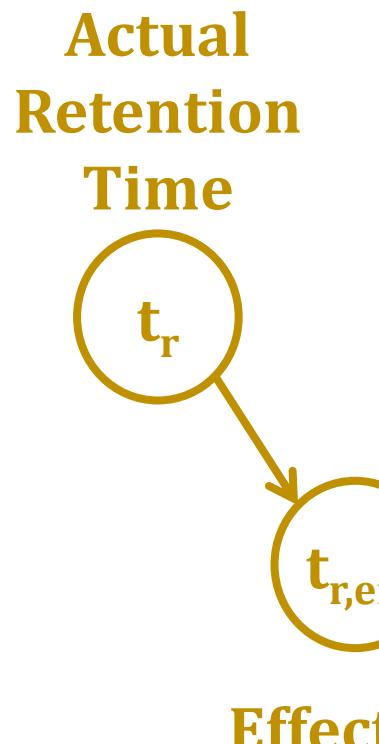
Retention Shift

$$\Delta V(t_{er}, t_{ed}, PEC) = b \cdot (PEC + c) \cdot \ln \left(1 + \frac{t_{er}}{t_0 + a \cdot t_{ed}} \right)$$



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

3. Temperature Scaling Component



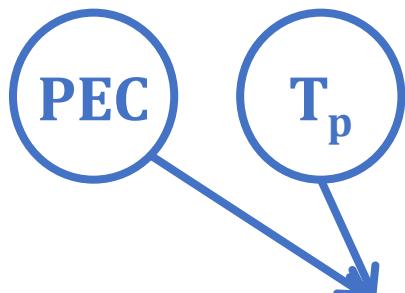
Arrhenius Equation: $AF = \frac{t_{real}}{t_{room}} = \exp\left(\frac{E_a}{k_B} \cdot \left(\frac{1}{T_{real}} - \frac{1}{T_{room}}\right)\right)$



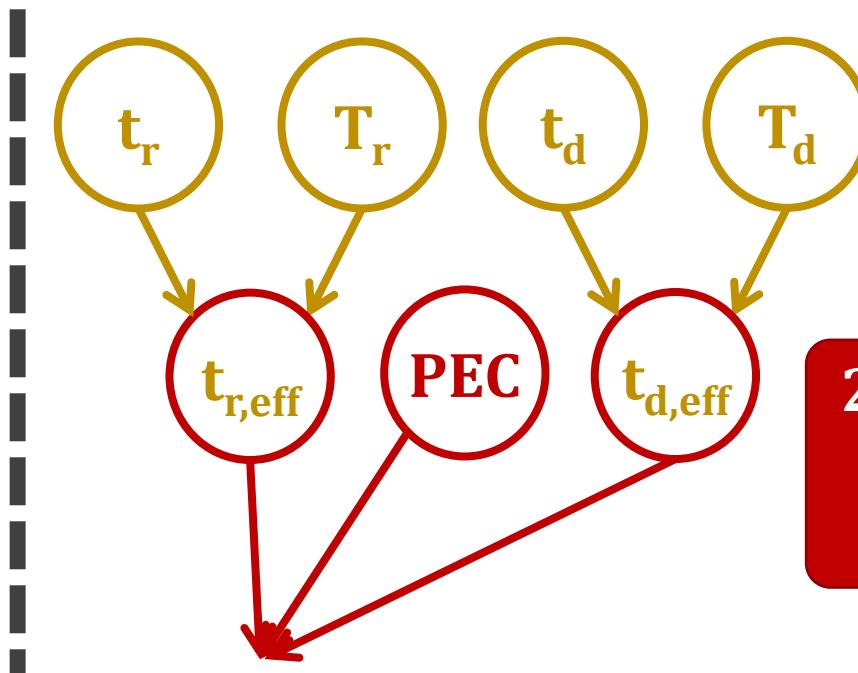
Validation: Adjust an important parameter, E_a , from 1.1 eV to 1.04 eV

URT Model Summary

1. Program Variation Component



$$V = V_0 + \Delta V$$



3. Temperature Scaling Component

2. Self-Recovery and Retention Component

Validation:

Prediction Error Rate = 4.9%



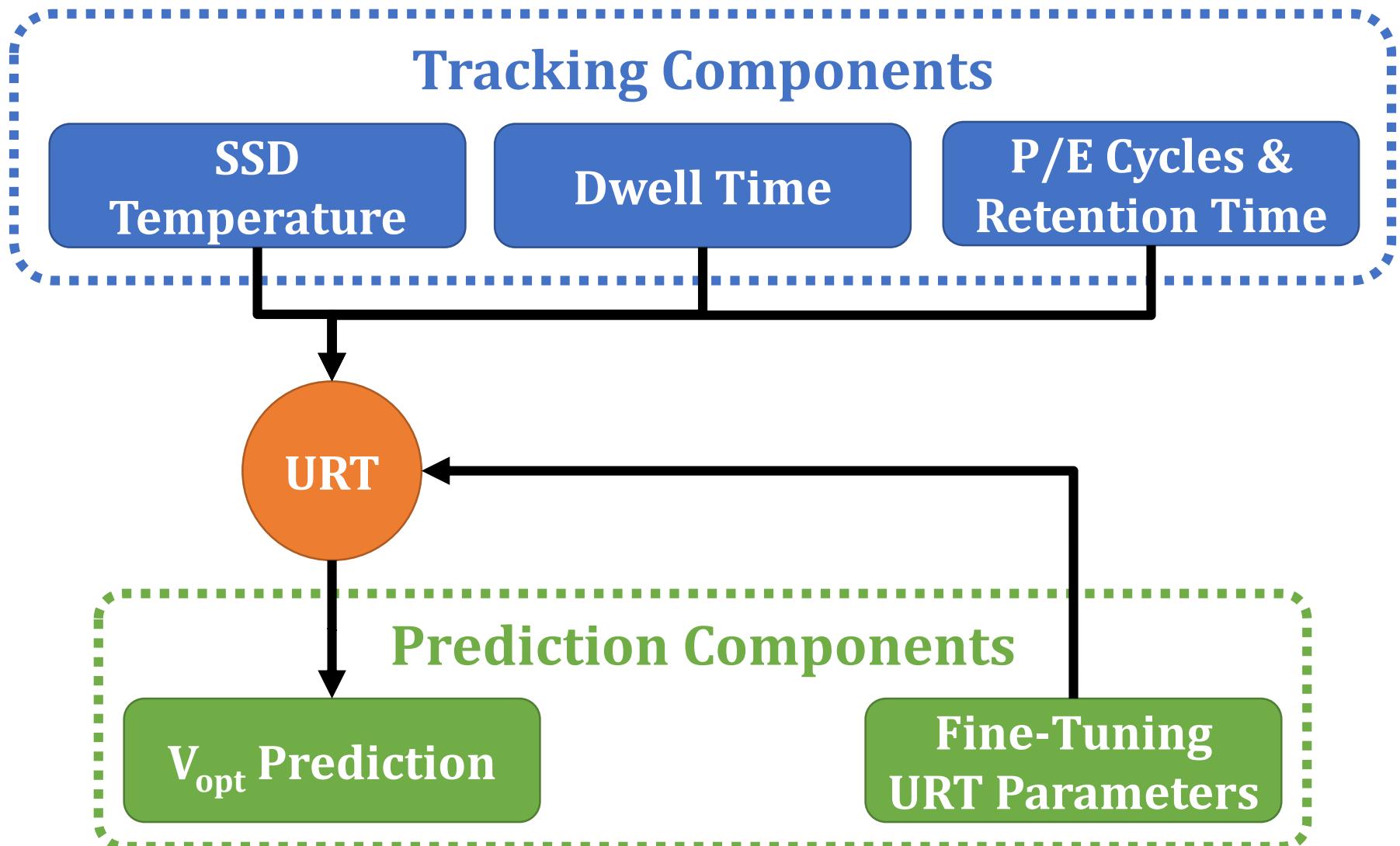
Outline

- 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 SSD Temperature

Tracking Components

SSD
Temperature

Dwell Time

P/E Cycles &
Retention Time

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

Prediction Components

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_{opt} Prediction

Fine-Tuning
URT Parameters

Predicting Optimal Read Reference Voltage

Tracking Components

SSD
Temperature

Dwell Time

P/E Cycles &
Retention Time

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

Prediction Components

V_{opt} Prediction

Fine-Tuning
URT Parameters

Fine-Tuning URT Parameters Online

Tracking Components

SSD
Temperature

Dwell Time

P/E Cycles &
Retention Time

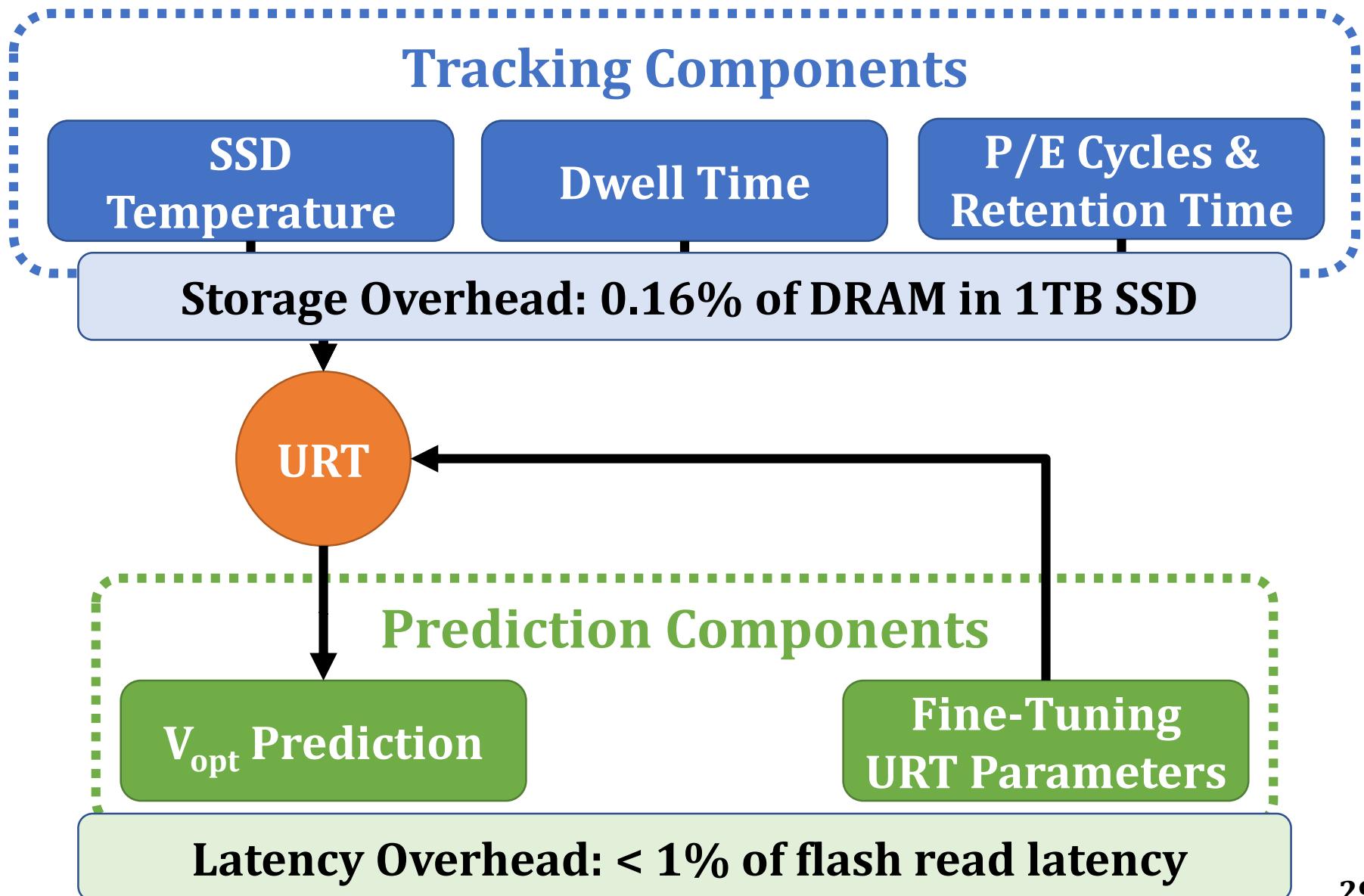
- Accommodates **chip-to-chip variation**
- Uses **periodic sampling**

Prediction Components

V_{opt} Prediction

Fine-Tuning
URT Parameters

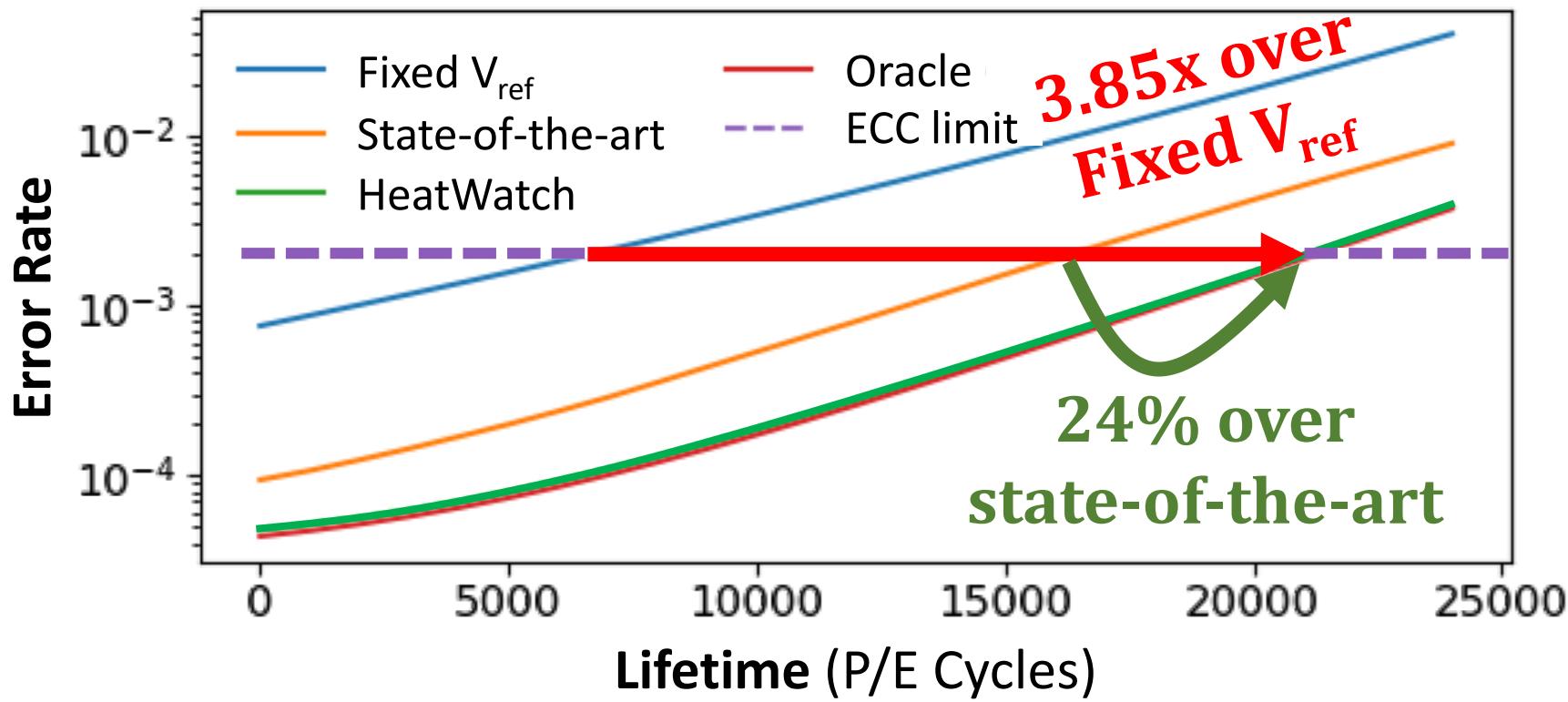
HeatWatch Mechanism Summary



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



HeatWatch improves lifetime by capturing the effect of retention, wearout, self-recovery, temperature

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

SAFARI



ETH Zürich