P&S Modern SSDs

Research Session 1: Data Sanitization and Read-Retry in Modern NAND Flash-Based SSDs

Dr. Jisung Park

Prof. Onur Mutlu

ETH Zürich

Spring 2022

15 July 2022

Outline

- NAND Flash Basics

- Read-Retry in Modern NAND Flash-Based SSDs

- Data Sanitization in Modern NAND Flash-Based SSDs

NAND Flash-Based SSDs

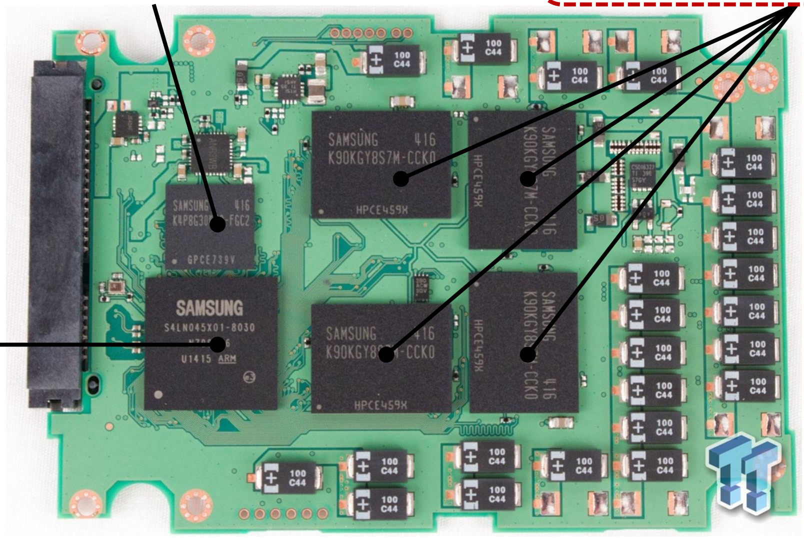
- A complicated embedded system
  - Multiple cores, HW controllers, DRAM, and NAND flash chips

SSD Controller

- ARM Cores
  Running firmware
- HW Flash Controllers
  Chip control, ECC

LPDDR DRAM
Metadata store

NAND Flash Chips
Persistent data store



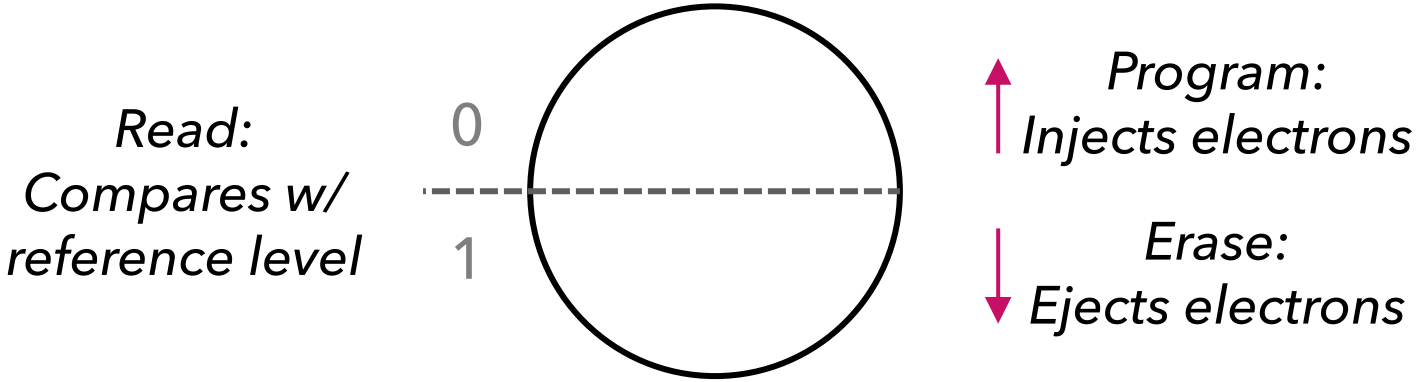
Unique characteristics

- Erase-before-write
- Limited endurance
- High error rates
- …

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

Flash Cell

- Stores data using its threshold voltage (V_{TH}) level
  - Dictated by the amount of electrons (i.e., charge) in the cell
  - The more the electrons, the higher the V_{TH} level



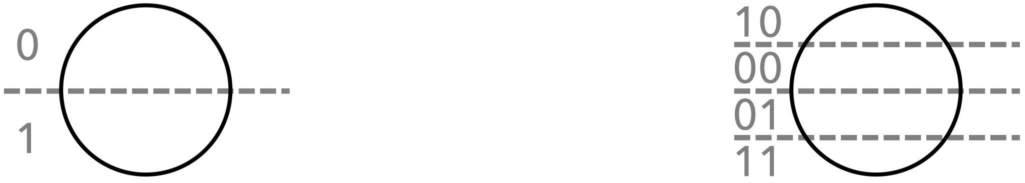
Flash Cell

1. Can change V_{TH} (=charge) in a non-volatile manner

2. Encodes bit data with different V_{TH} ranges

Flash Cell Characteristics

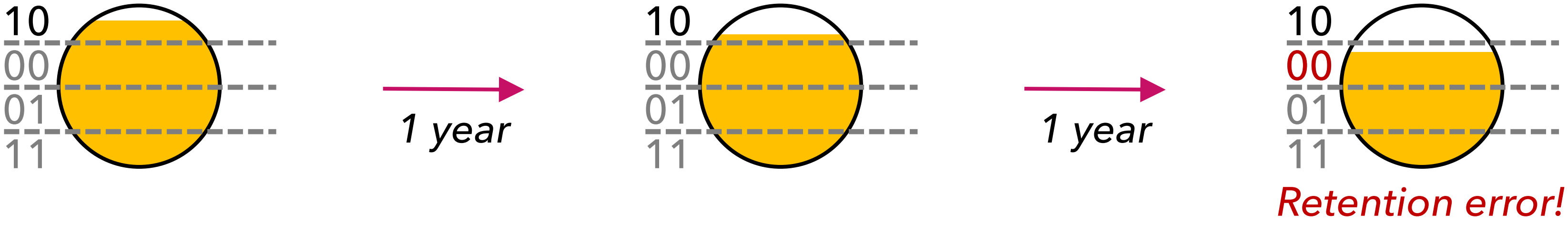
- Multi-leveling: A single flash cell can store multiple bits



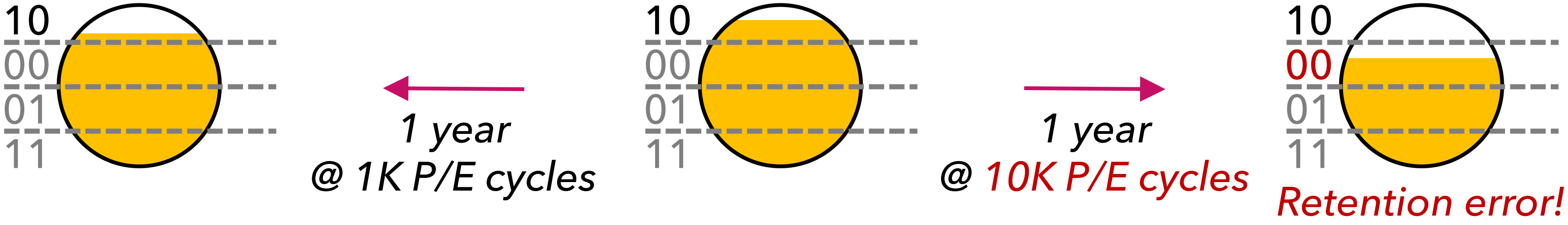
Single-Level Cell (SLC)

2-Bit Multi-Level Cell (MLC)

- Retention loss: A cell leaks electrons over time



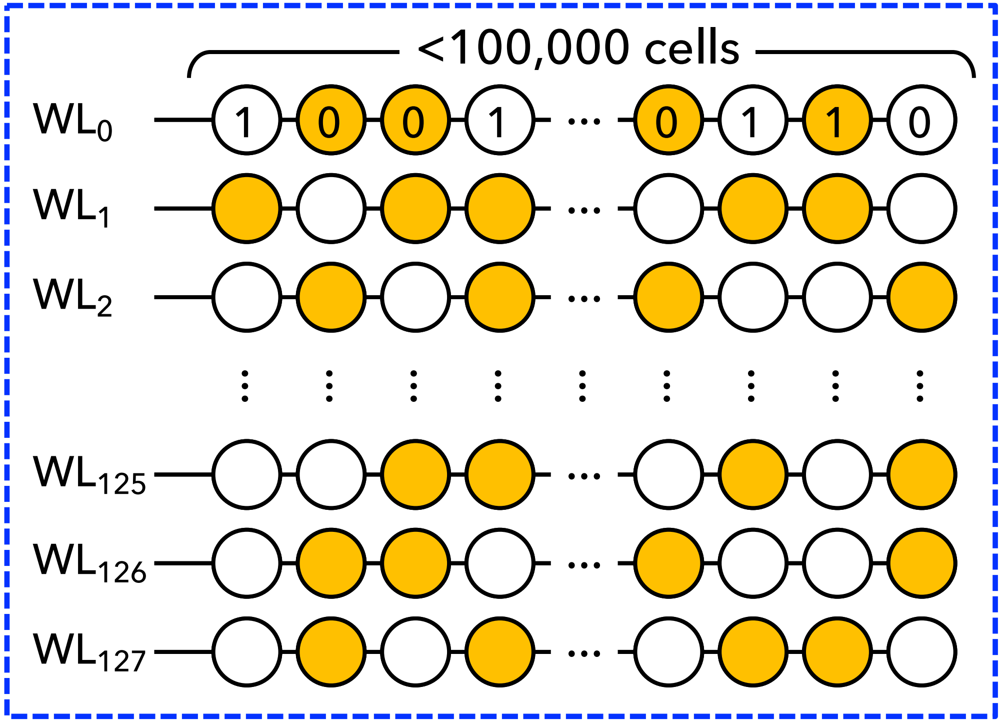
- Limited lifetime: A cell wears out after P/E cycling



Page and Block

- A number of flash cells operate in parallel

Cells in the same wordline (WL) are concurrently read/programmed



Block (128 WLs)

Cells in the same block are concurrently erased

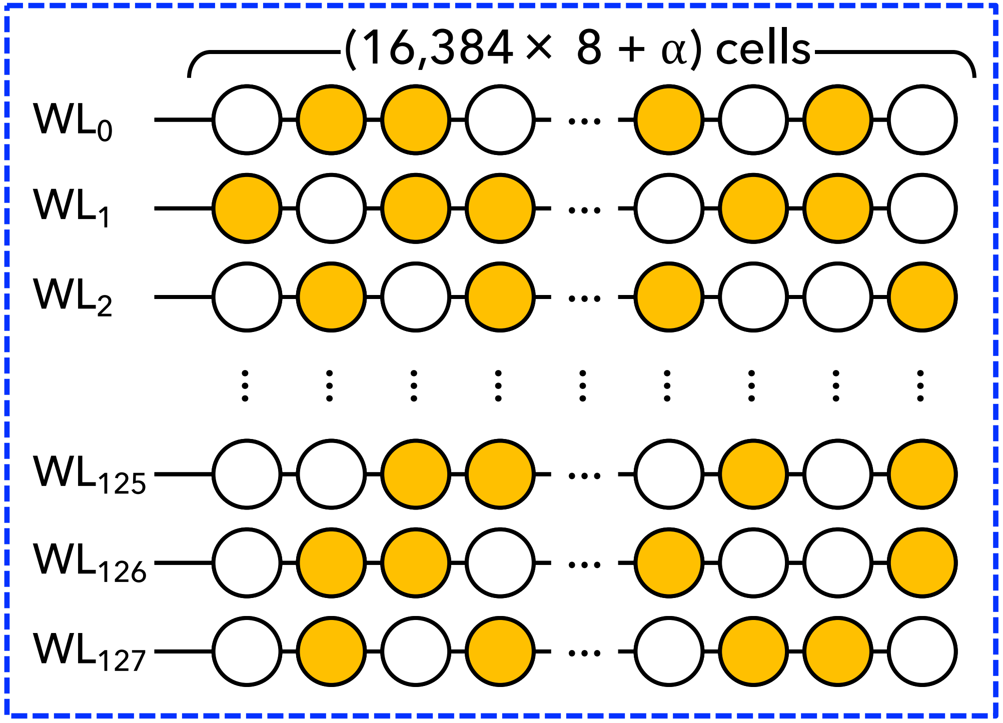
Long latency, but high bandwidth

No overwrite of WLs before erasing the block: *Erase-before-write*

Page and Block

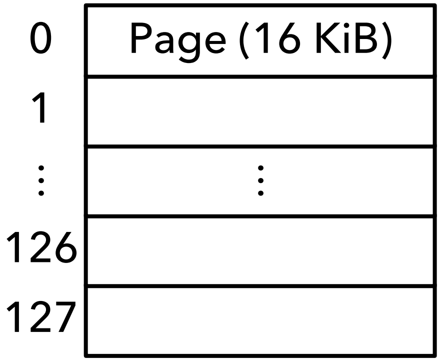
- A number of flash cells operate in parallel

Cells in the same wordline (WL) are concurrently read/programmed



Block (128 WLs)

Cells in the same block are concurrently erased

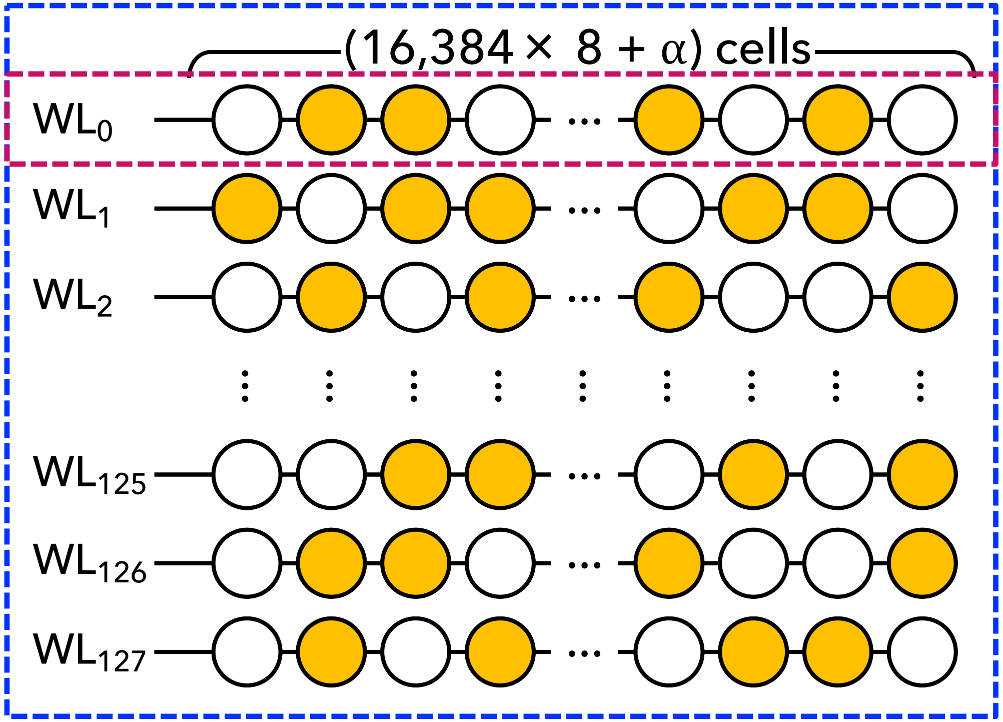


Block (127 pages)

Page and Block

- A number of flash cells operate in parallel

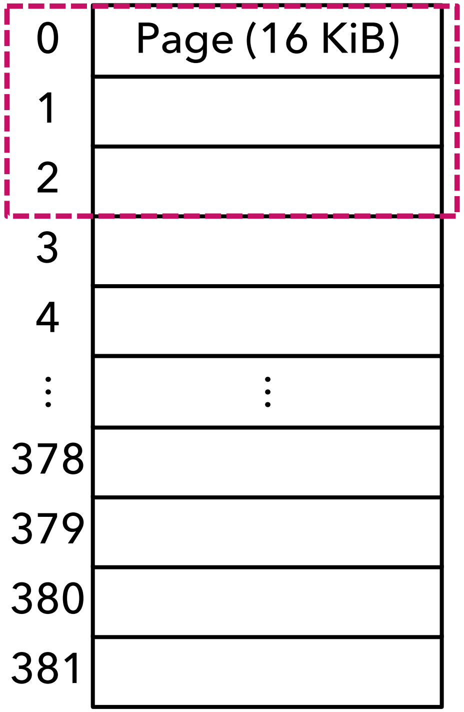
Cells in the same wordline (WL)
are concurrently read/programmed



Block (128 WLs)

Cells in the same block
are concurrently erased

Share the same cells (WL)



Triple-Level Cell (TLC)
Block (127 × 3 pages)

Pipelined & Adaptive Read-Retry

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

Jisung Park[1] Myungsuk Kim[2,3] Myoungjun Chun[2] Lois Orosa[1] Jihong Kim[2] Onur Mutlu[1]

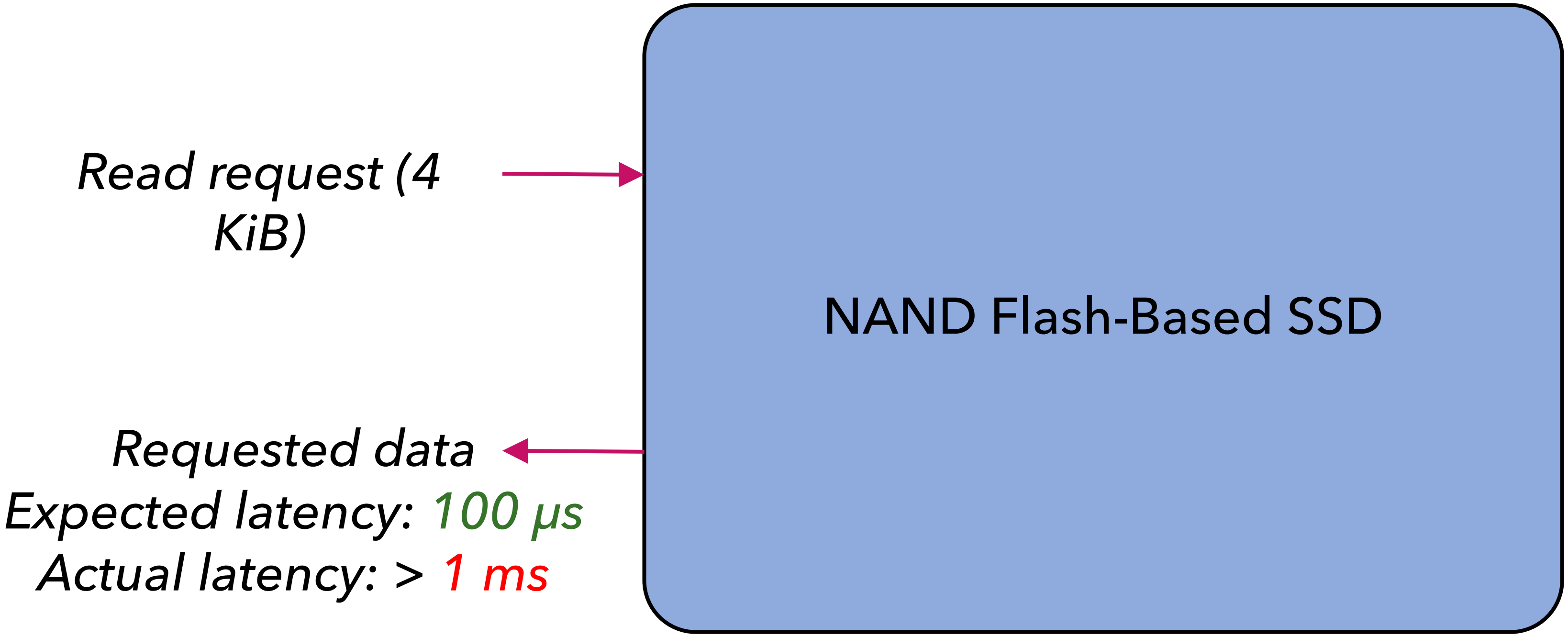
[1]ETH Zürich
Switzerland

[2]Seoul National University
Republic of Korea

[3]Kyungpook National University
Republic of Korea

The 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)

Problem: Long, Non-Deterministic Latency



Read request (4 KiB)

NAND Flash-Based SSD

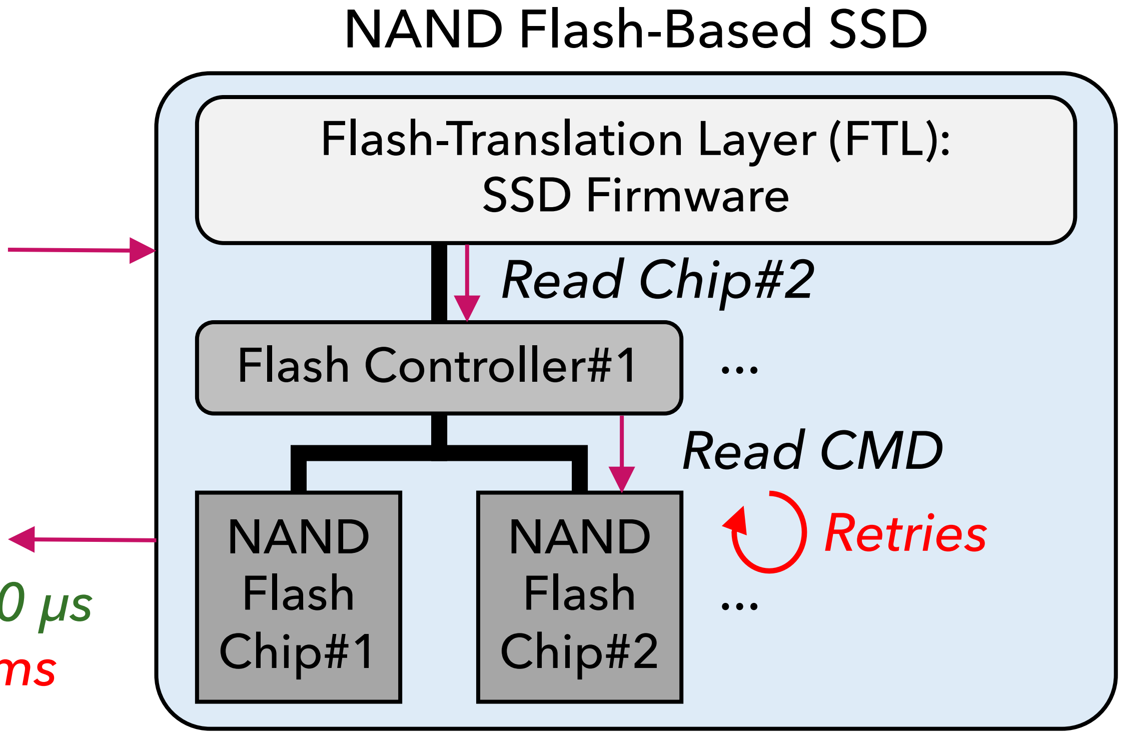
Requested data

Expected latency: 100 µs

Actual latency: > 1 ms

Degrades the quality of service
of read-intensive, latency-sensitive applications

Problem: Long, Non-Deterministic Latency



Read request (4 KiB)

Requested data

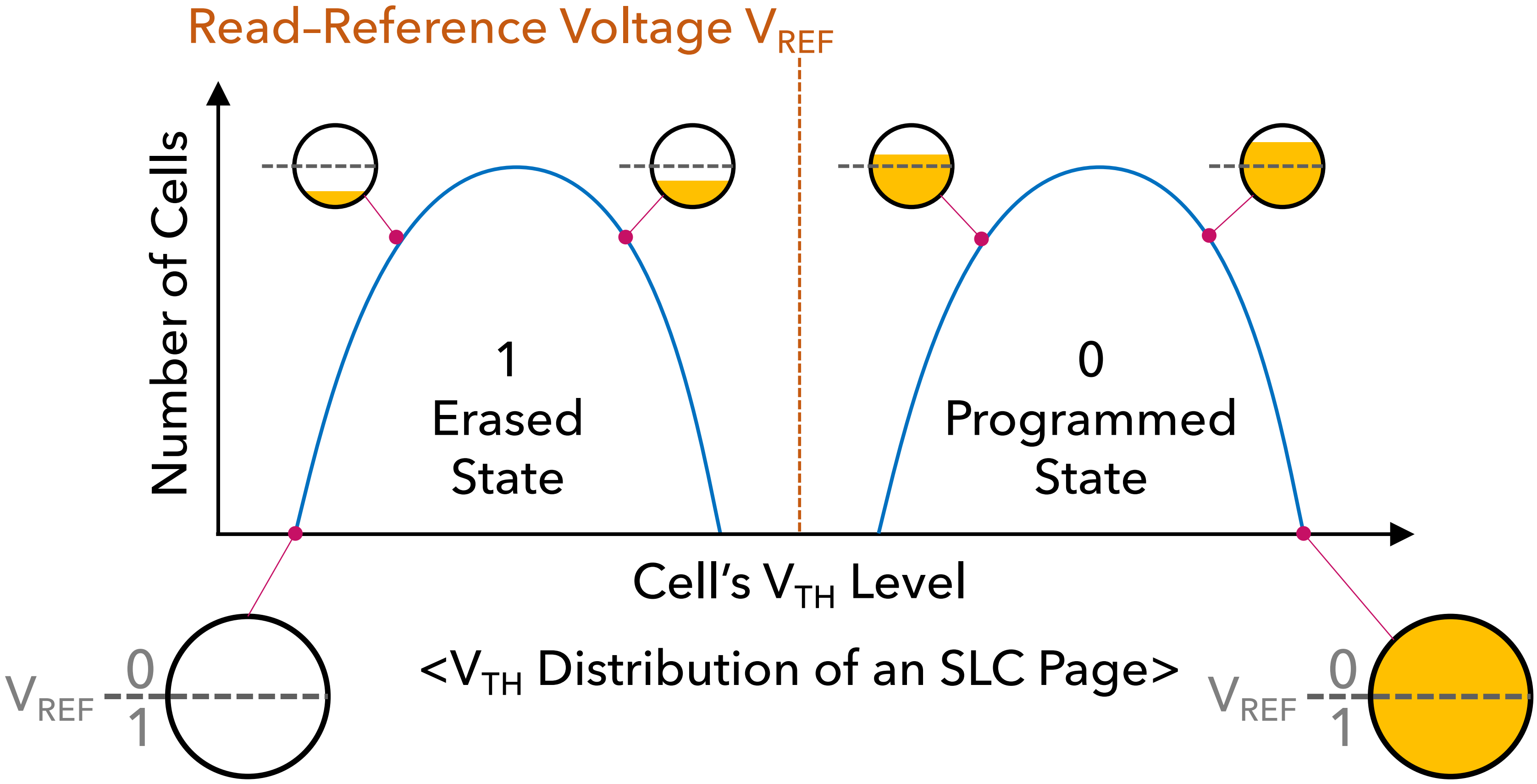
Expected latency: 100 µs

Actual latency: > 1 ms

Internal Read-Retry Operations

Errors in NAND Flash Memory

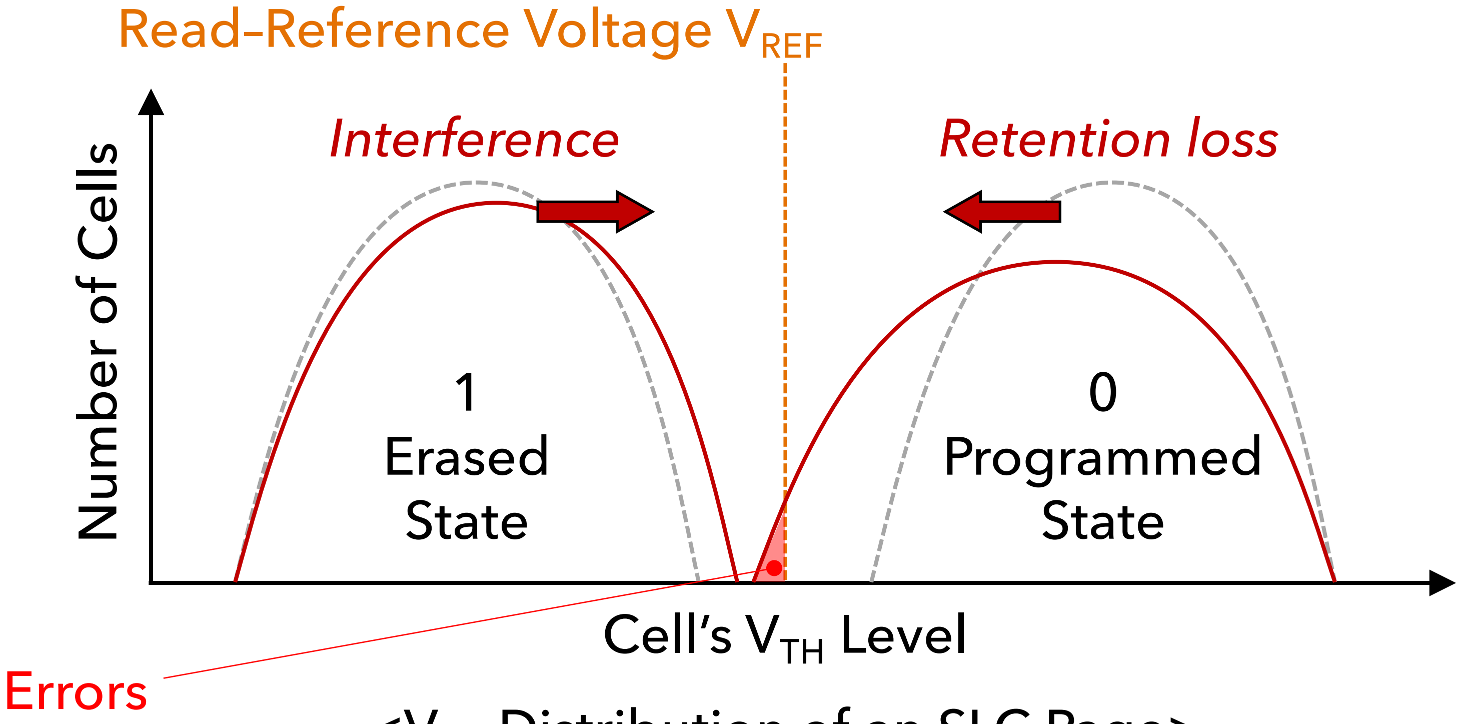
- NAND flash memory stores data by using cells' V_{TH} levels



<V_{TH} Distribution of an SLC Page>

Errors in NAND Flash Memory

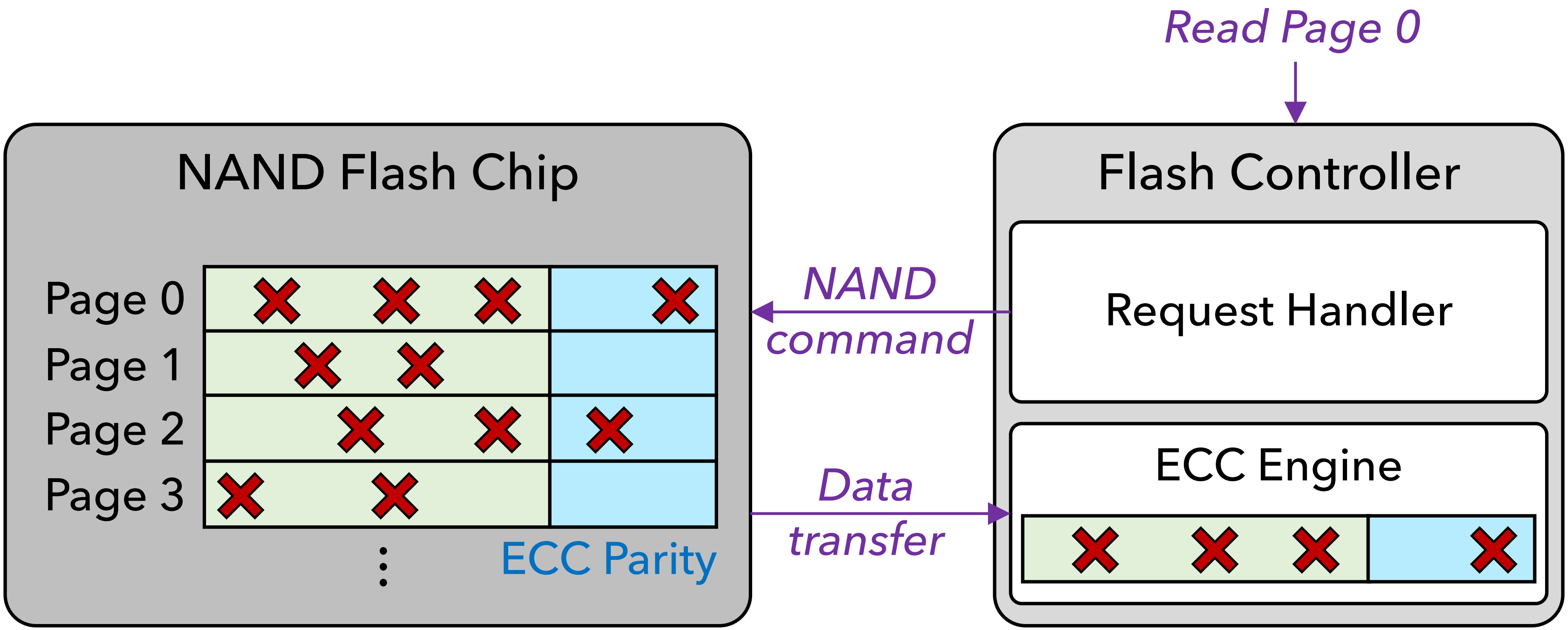
- Various sources shift and widen programmed V_{TH} states
  - Retention loss, program interference, read disturbance, etc.



<V_{TH} Distribution of an SLC Page>

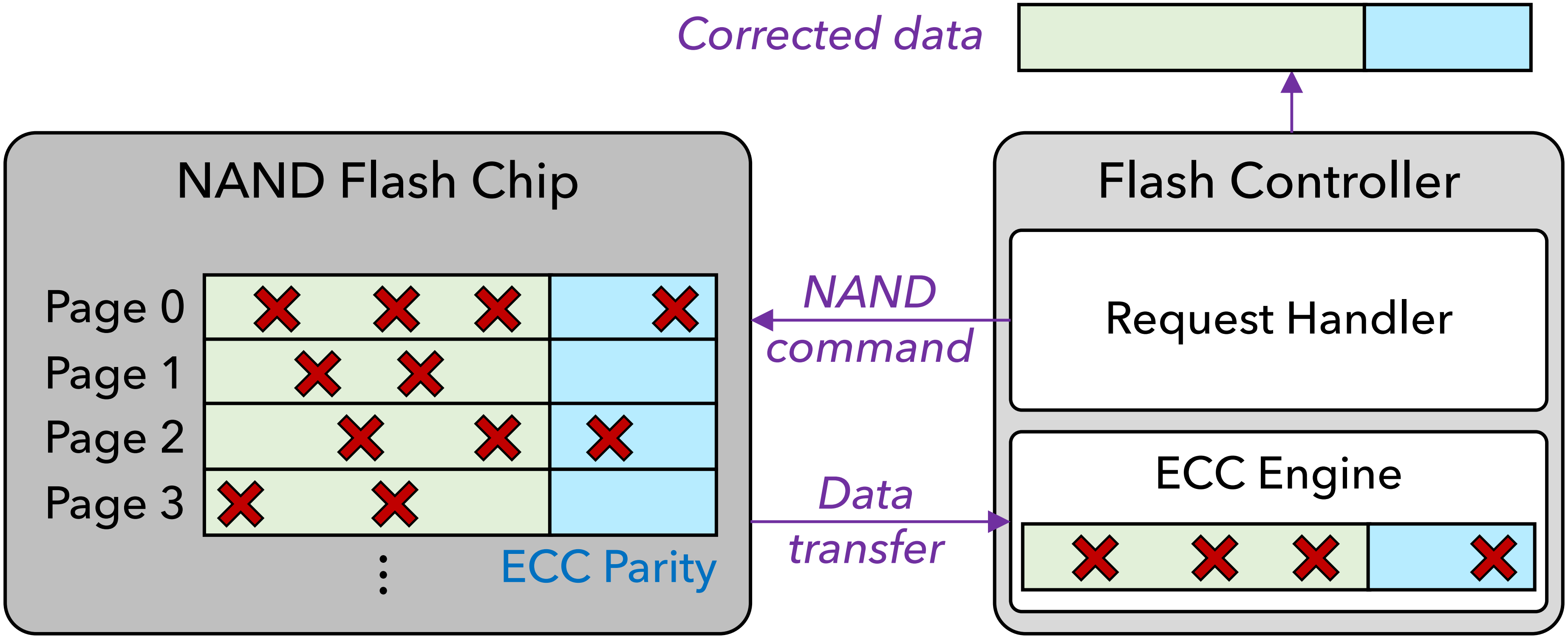
Error-Correcting Codes (ECC)

- Store redundant information (ECC parity)
  - To detect and correct row bit errors



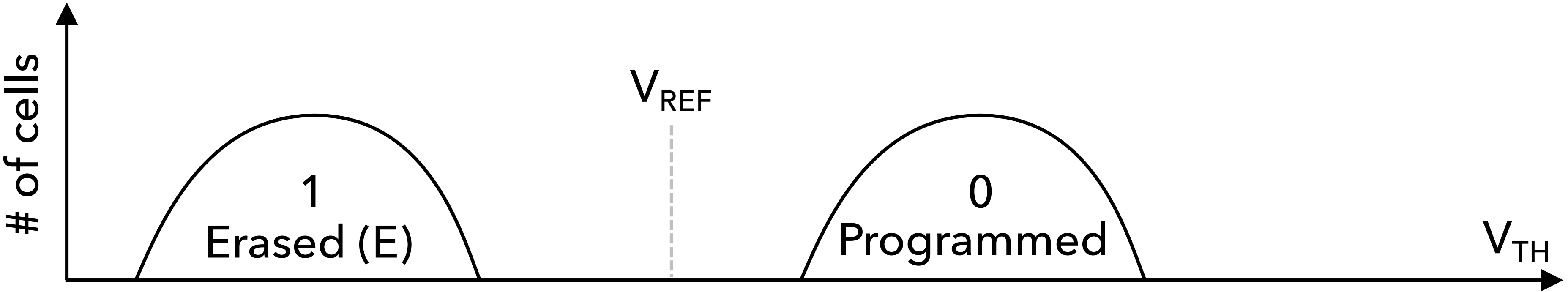
Error-Correcting Codes (ECC)

- Store redundant information (ECC parity)
  - To detect and correct row bit errors

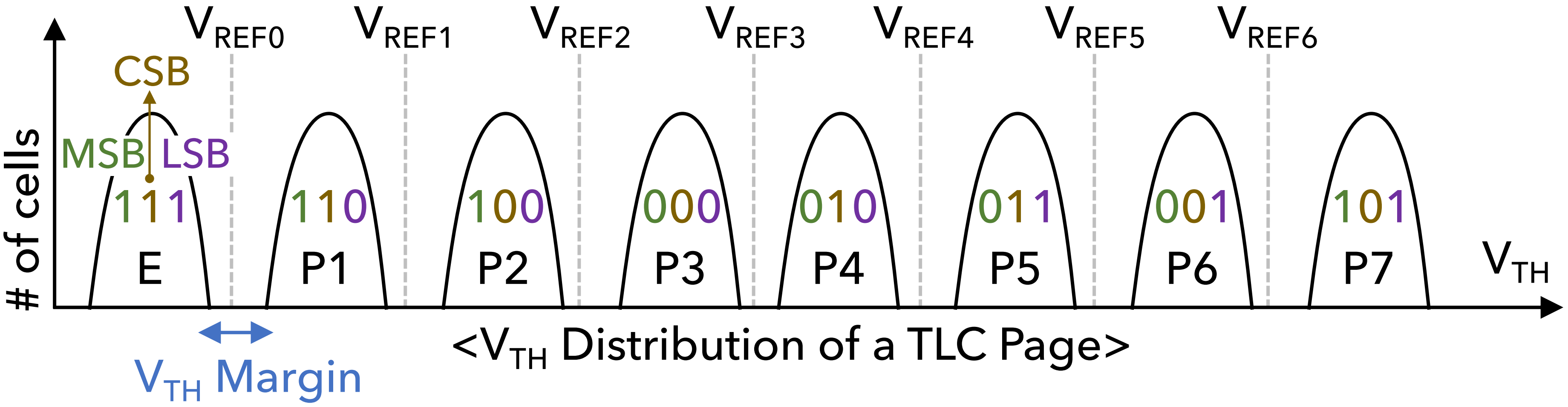


Errors in Modern NAND Flash Memory

- High row bit-error rates (RBER) in MLC NAND flash memory
  - Narrow margin b/w adjacent V_{TH} states



<V_{TH} Distribution of an SLC Page>



<V_{TH} Distribution of a TLC Page>

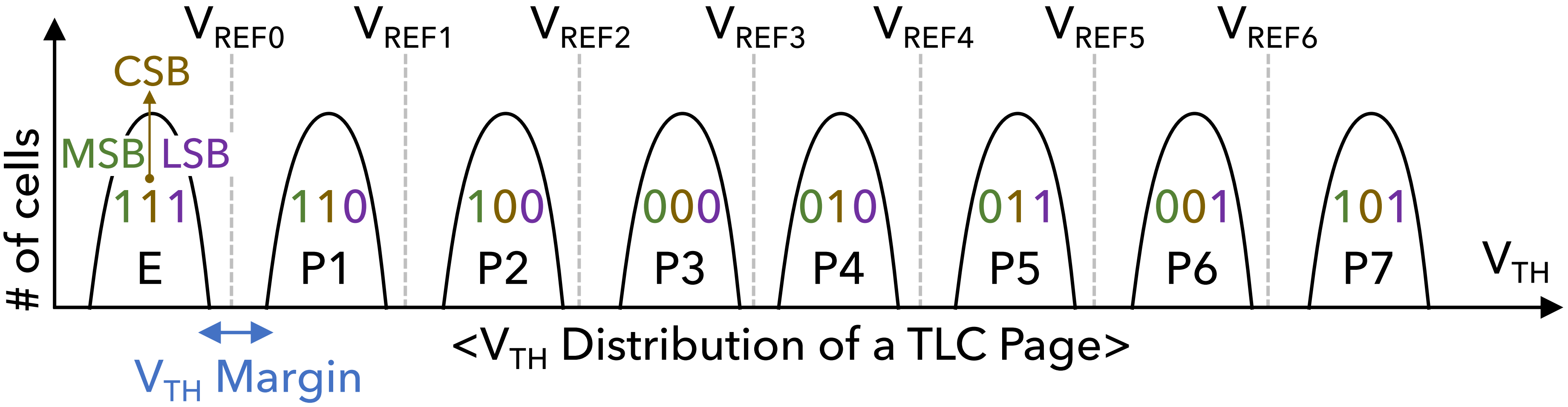
Errors in Modern NAND Flash Memory

- High row bit-error rates (RBER) in MLC NAND flash memory
    - Narrow margin b/w adjacent V_{TH} states

Strong ECC: Corrects ~80 bit errors per 1-KiB data

Not Scalable: Area, power, latency, …

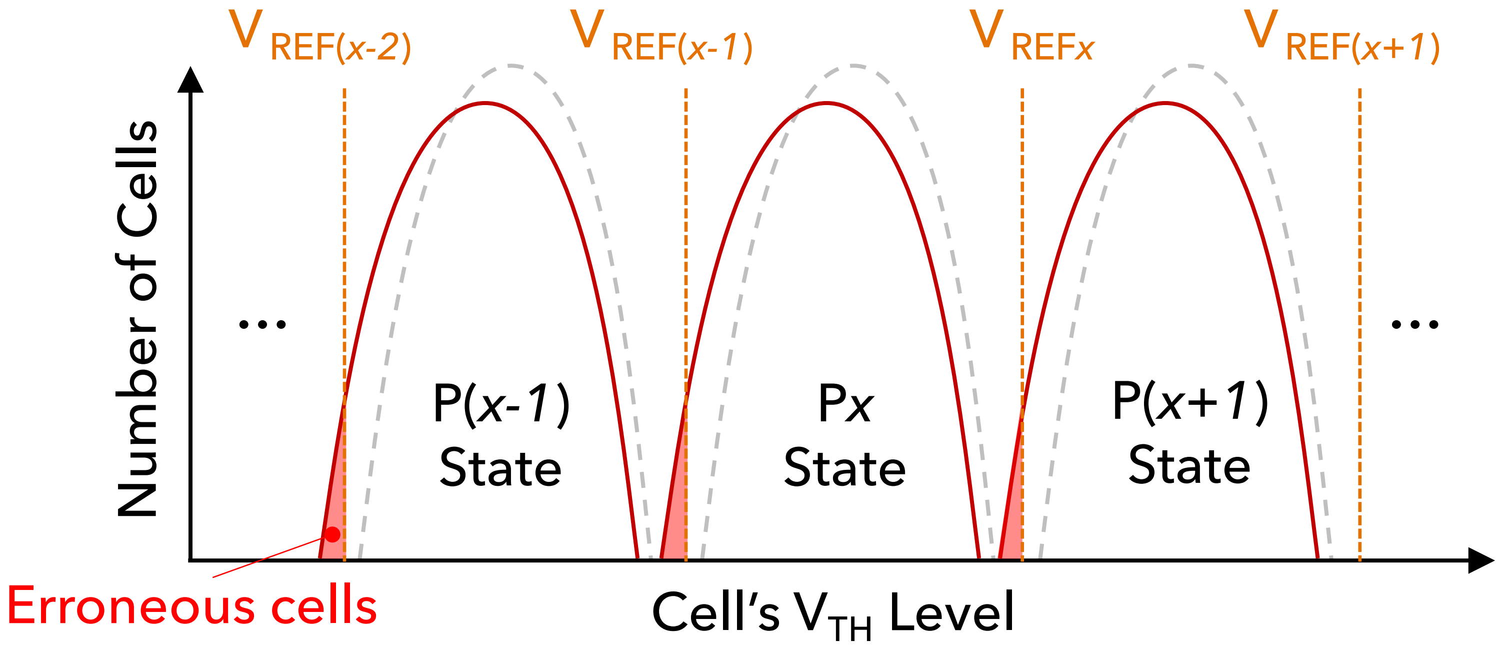
What if RBER > ECC Capability?



<V_{TH} Distribution of a TLC Page>

Read-Retry Operation

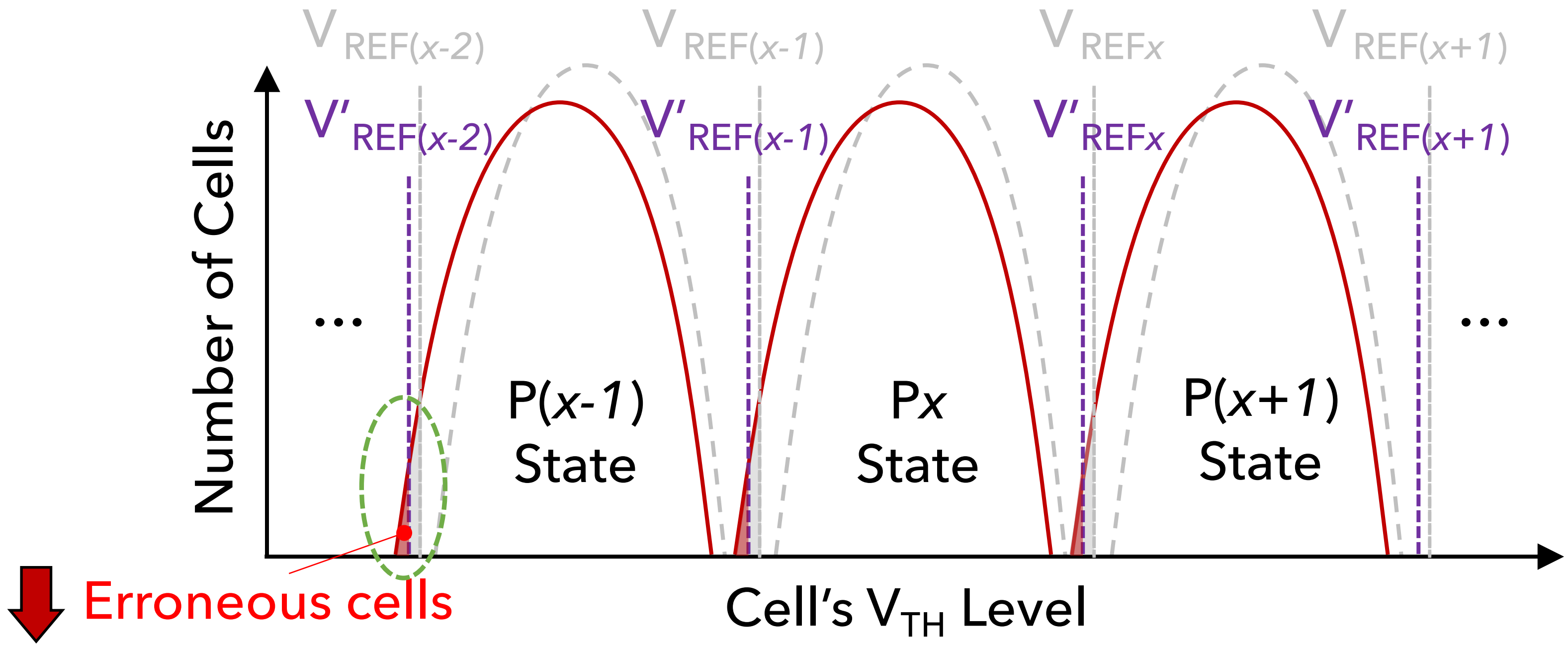
- Reads the page again with adjusted V_{REF} values



Read-Retry Operation

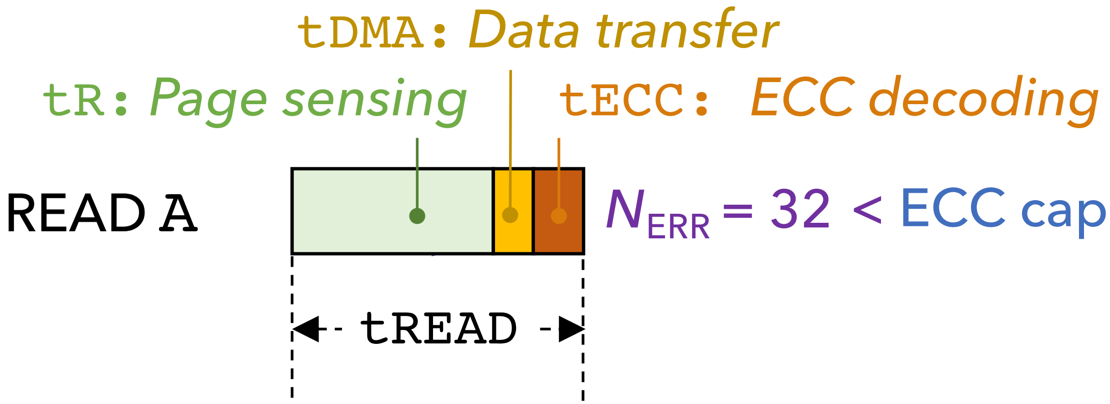
- Reads the page again with adjusted V_{REF} values

Read-retry: Adjusting V_{REF} values



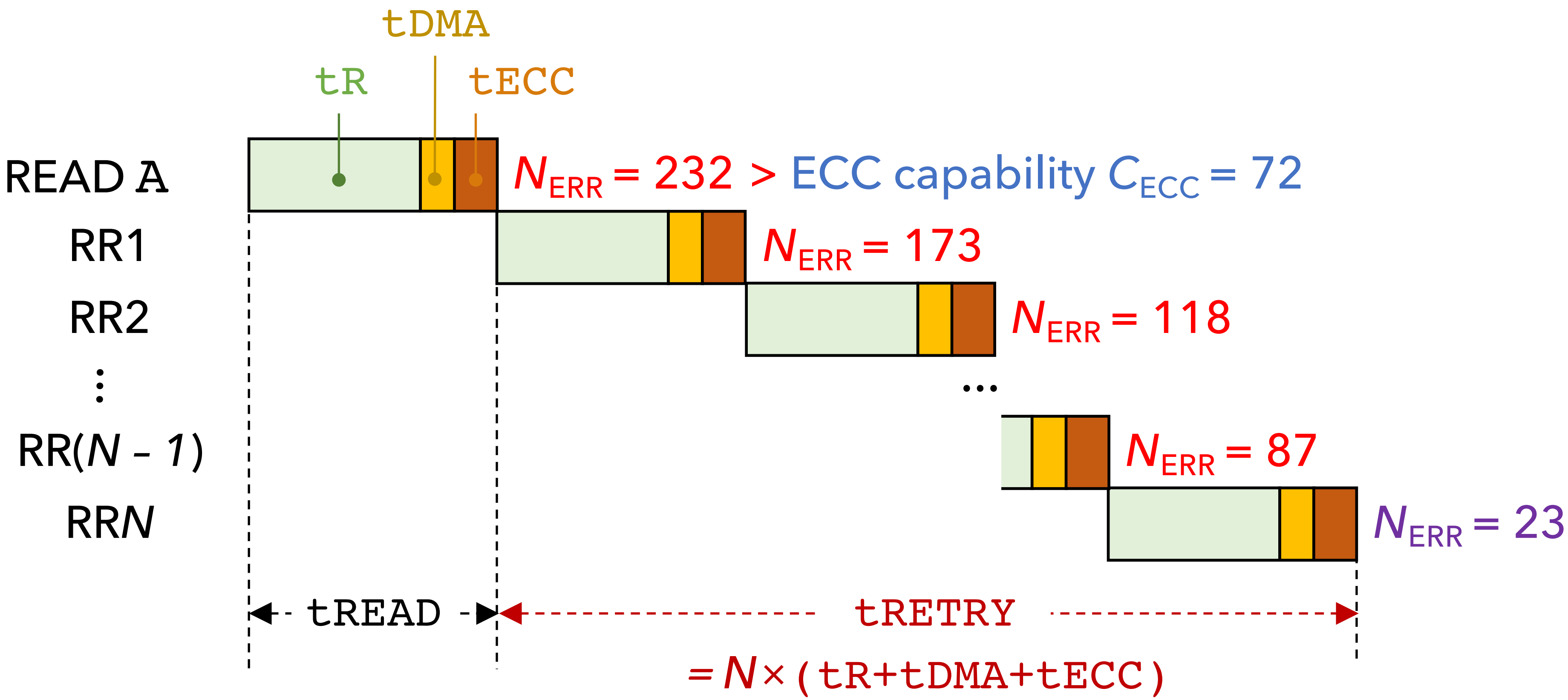
Read using properly-adjusted V_{REF} values
→ Decreases # of raw bit errors
to be lower than the ECC capability

Read-Retry: Performance Overhead



$N_{ERR} = 32 <$ ECC capability $C_{ECC} = 72$

Read-Retry: Performance Overhead



Read-retry increases the read latency almost linearly with the number of retry steps

P&AR^2: Outline

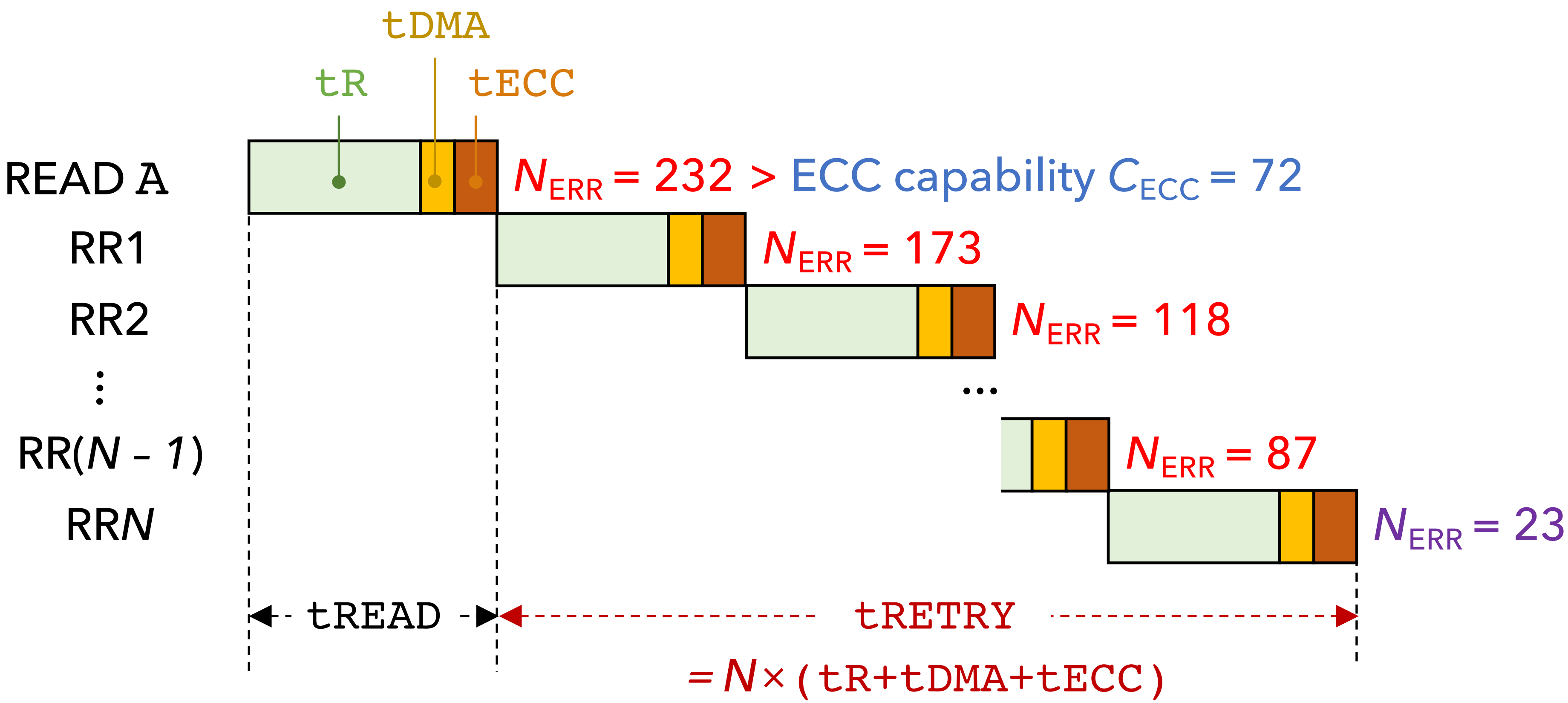
- Read-Retry in Modern NAND Flash-Based SSDs

- PR^2: Pipelined Read-Retry

- AR^2: Adaptive Read-Retry

- Evaluation Results

Pipelined Read-Retry (PR^2): Key Idea



tDMA

tR

tECC

READ A

RR1

RR2

⋮

RR(N – 1)

RRN

N_{ERR} = 232 > ECC capability C_{ECC} = 72

N_{ERR} = 173

N_{ERR} = 118

…

N_{ERR} = 87

N_{ERR} = 23

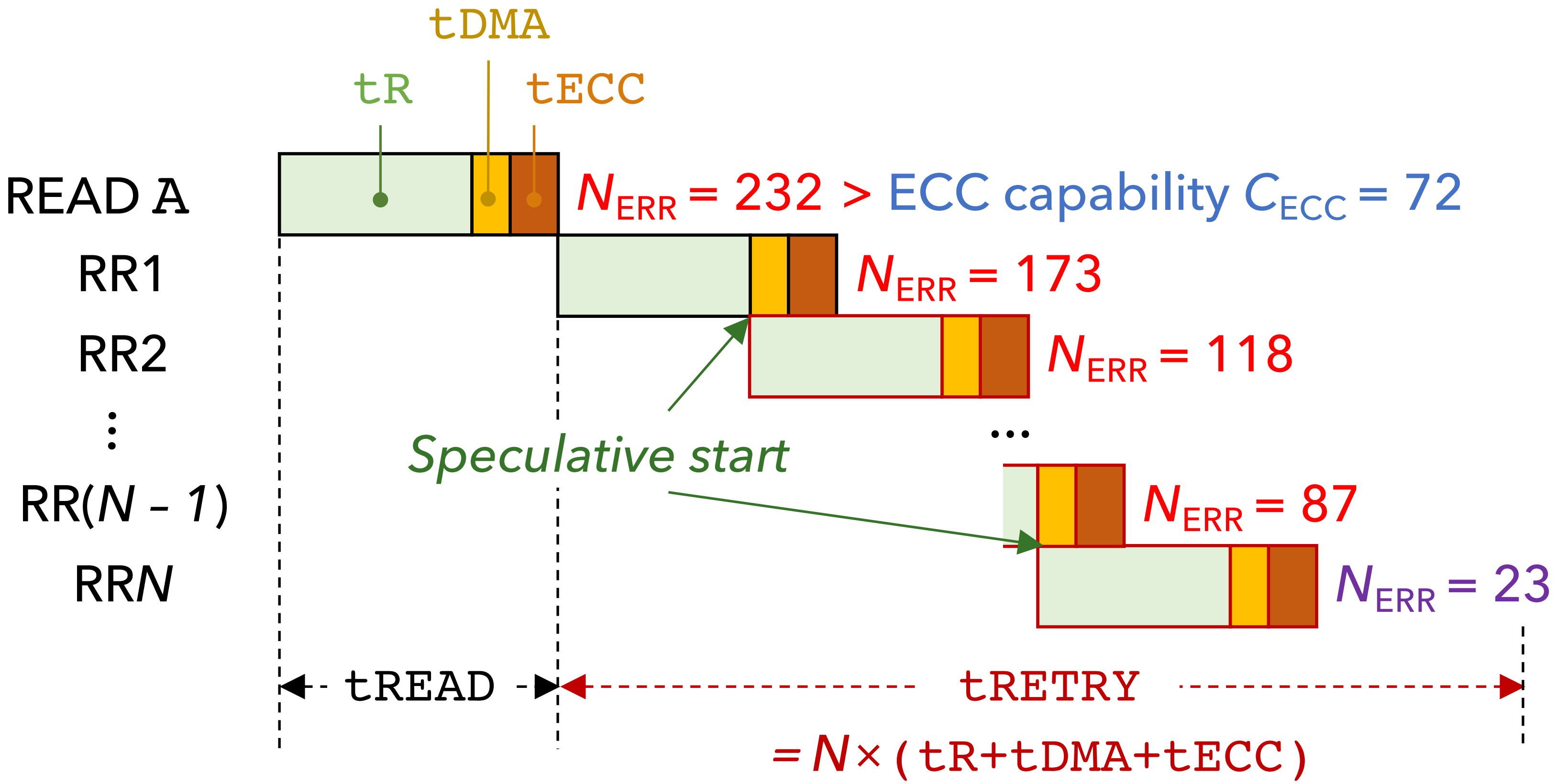
tREAD

tRETRY

= N × (tR+tDMA+tECC)

In common cases, multiple (up to 25) retry steps occur

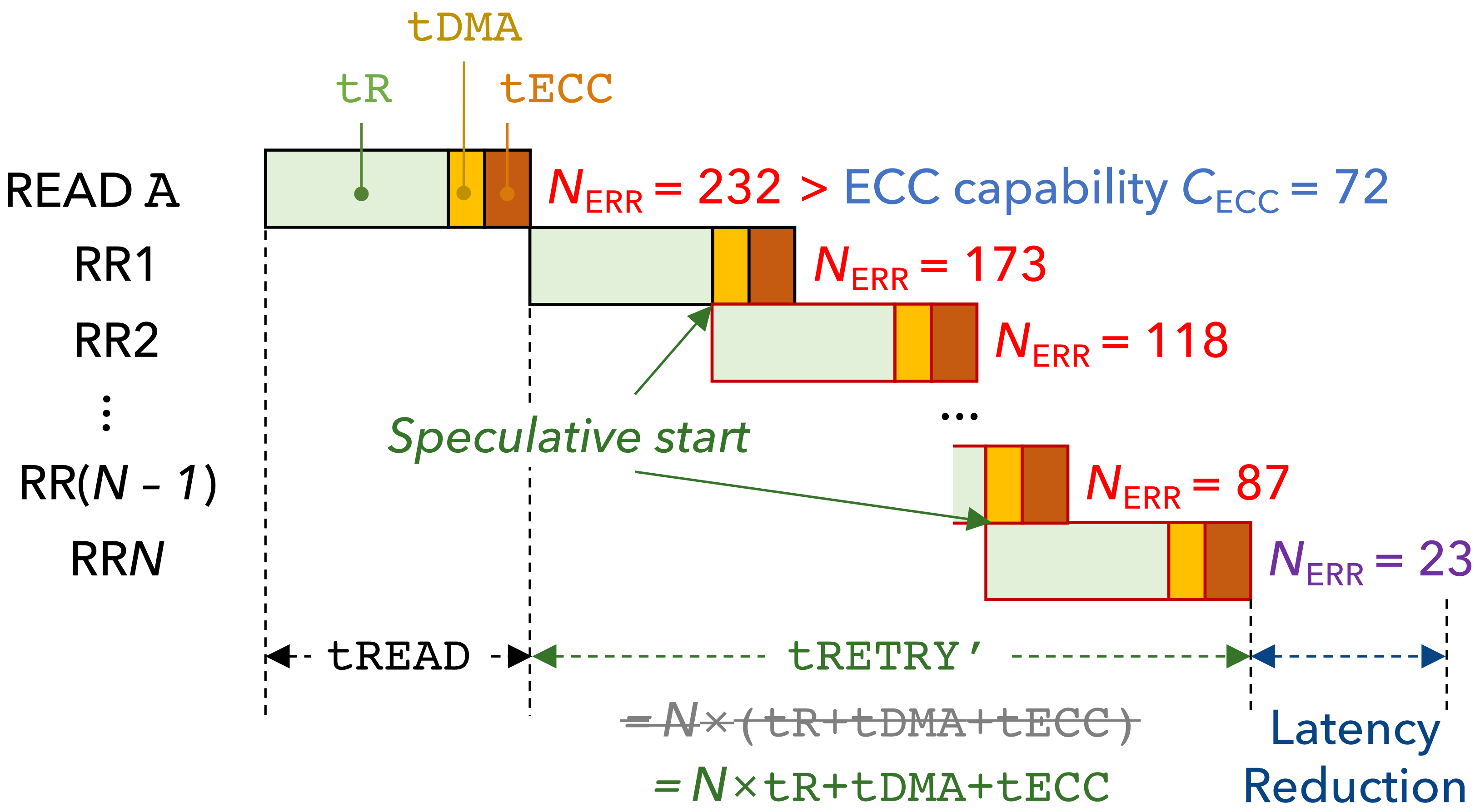
Pipelined Read-Retry (PR^2): Key Idea



In common cases, multiple (up to 25) retry steps occur

→ Speculatively starts the next retry step

Pipelined Read-Retry (PR^2): Key Idea



Removes tDMA & tECC

(~30% of each retry step) from the critical path

P&AR^2: Outline

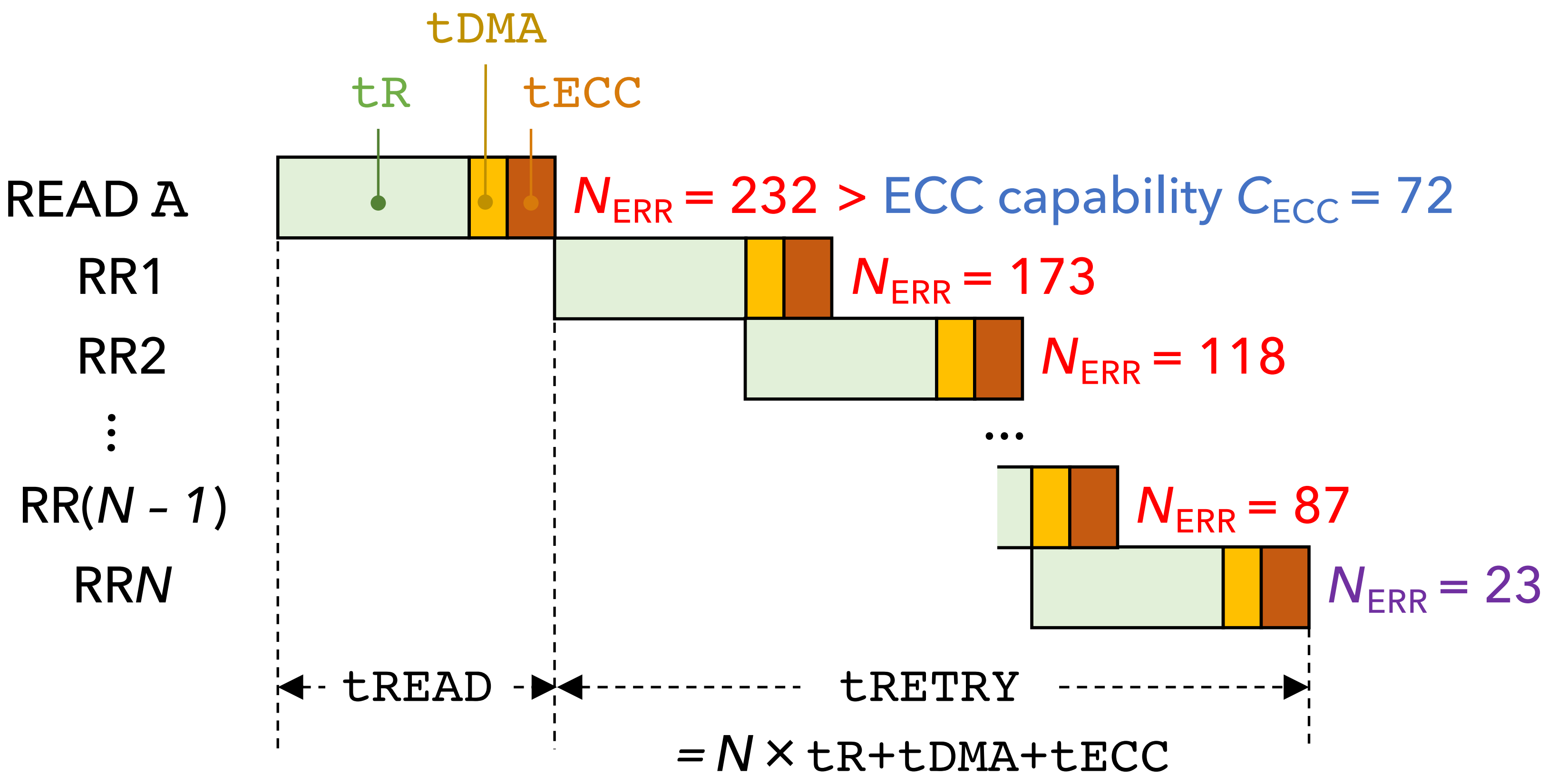
- Read-Retry in Modern NAND Flash-Based SSDs

- PR^2: Pipelined Read-Retry

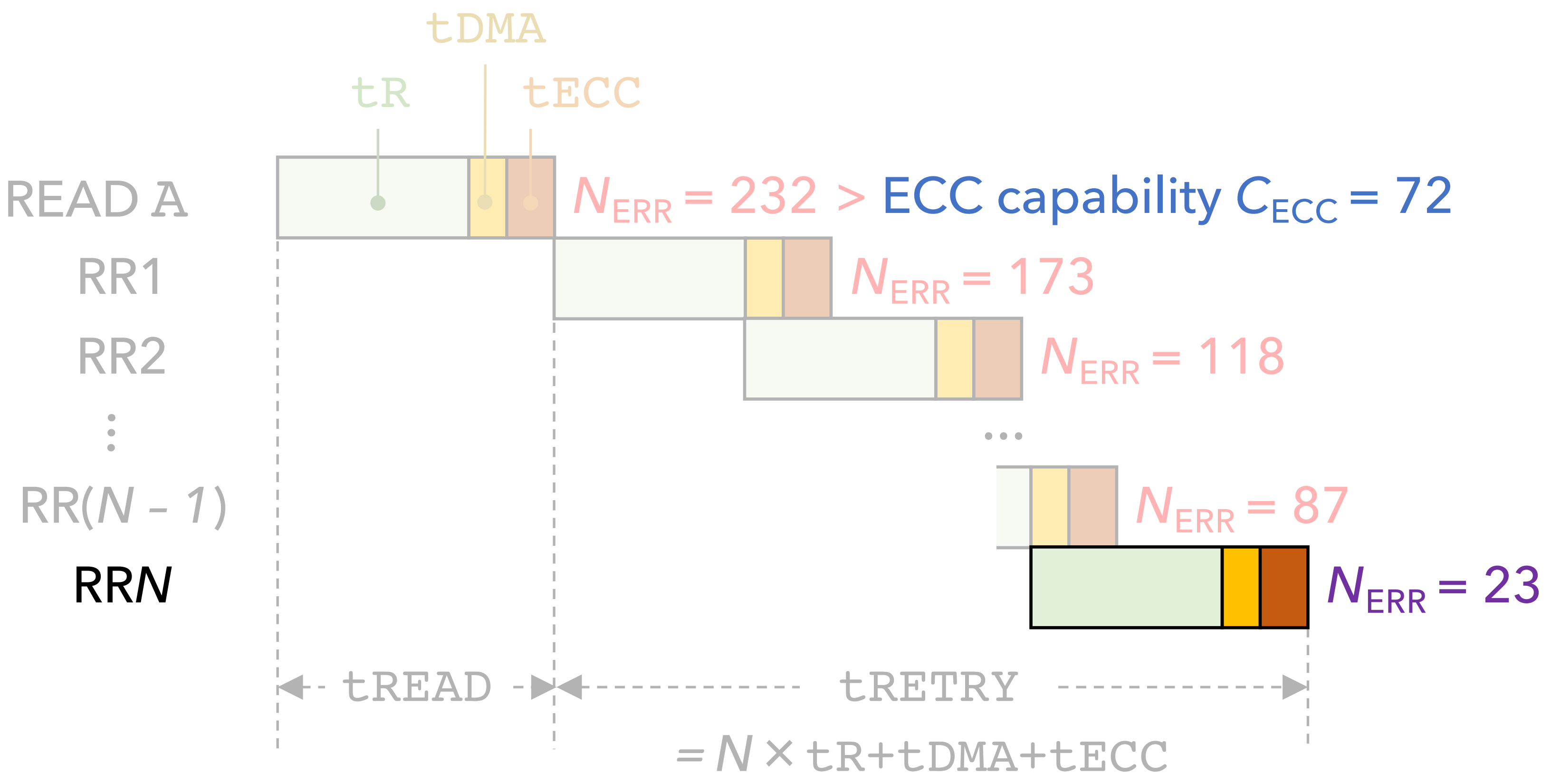
- AR^2: Adaptive Read-Retry

- Evaluation Results

Adaptive Read-Retry (AR^2): Key Idea

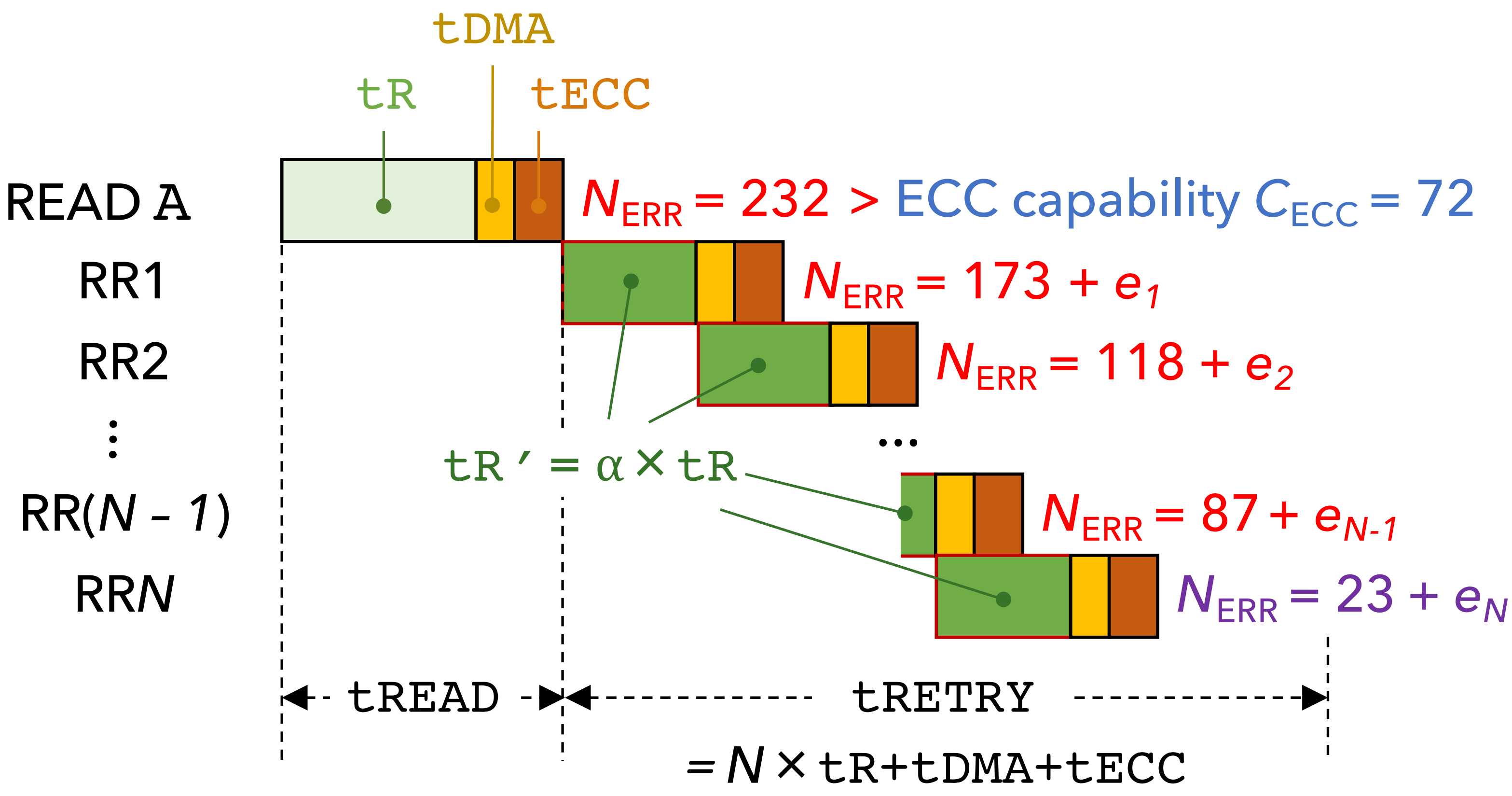


Adaptive Read-Retry (AR^2): Key Idea



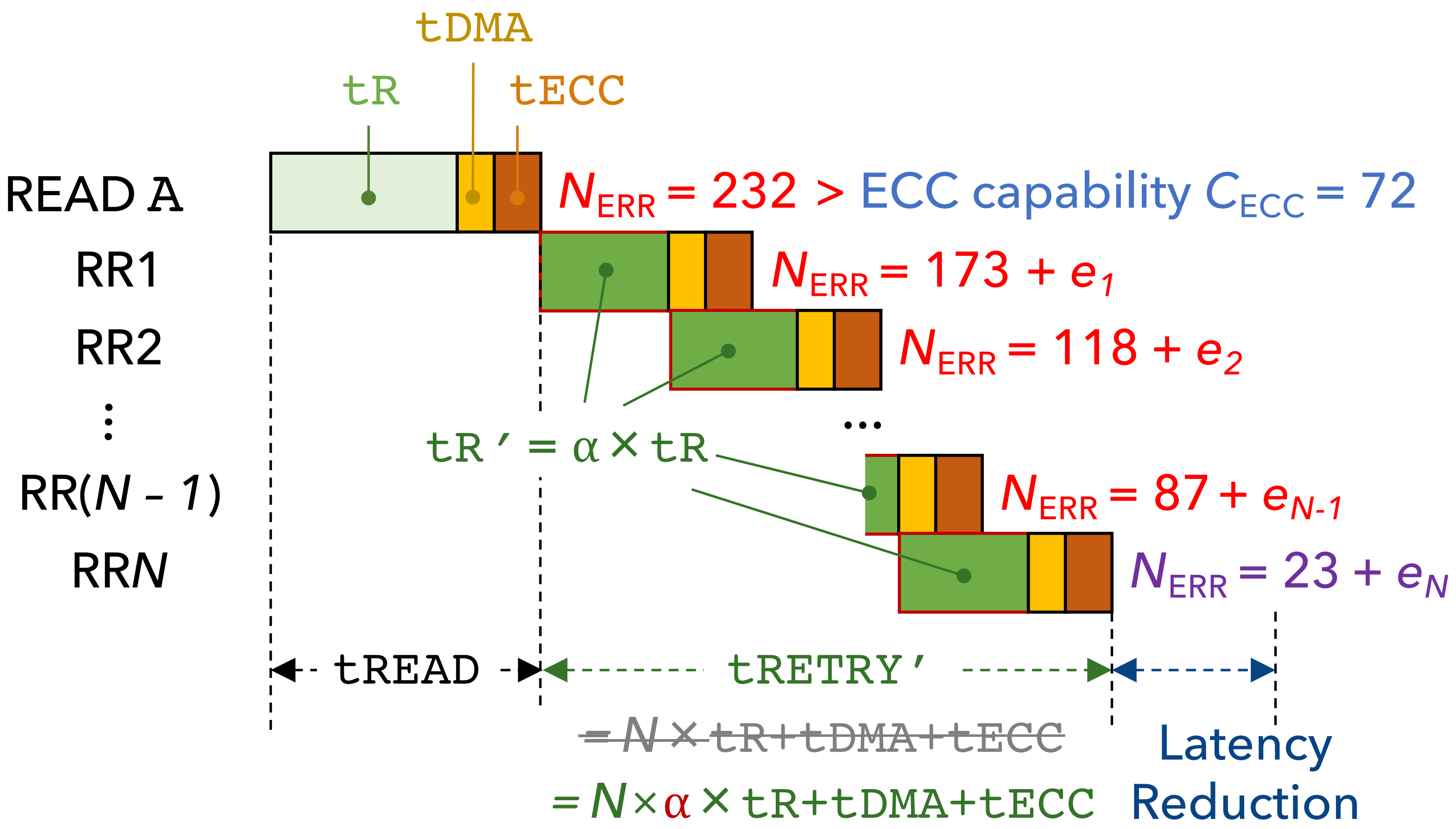
A large ECC margin
in the final retry step when read-retry succeeds
→ Can we leverage this ECC (reliability) margin?

Adaptive Read-Retry (AR^2): Key Idea



Trading the large ECC margin to reduce tR

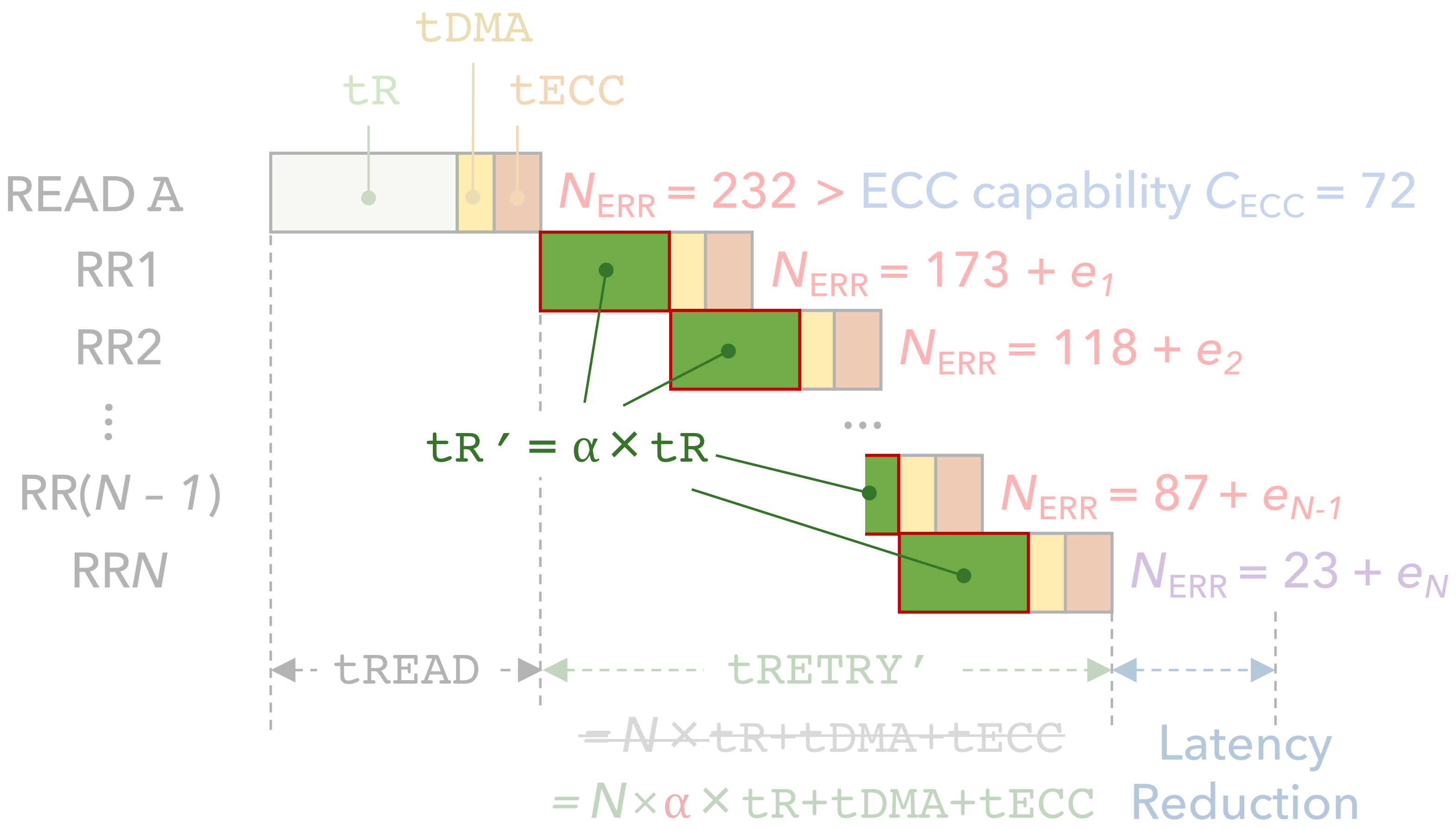
Adaptive Read-Retry (AR^2): Key Idea



Trading the large ECC margin to reduce tR

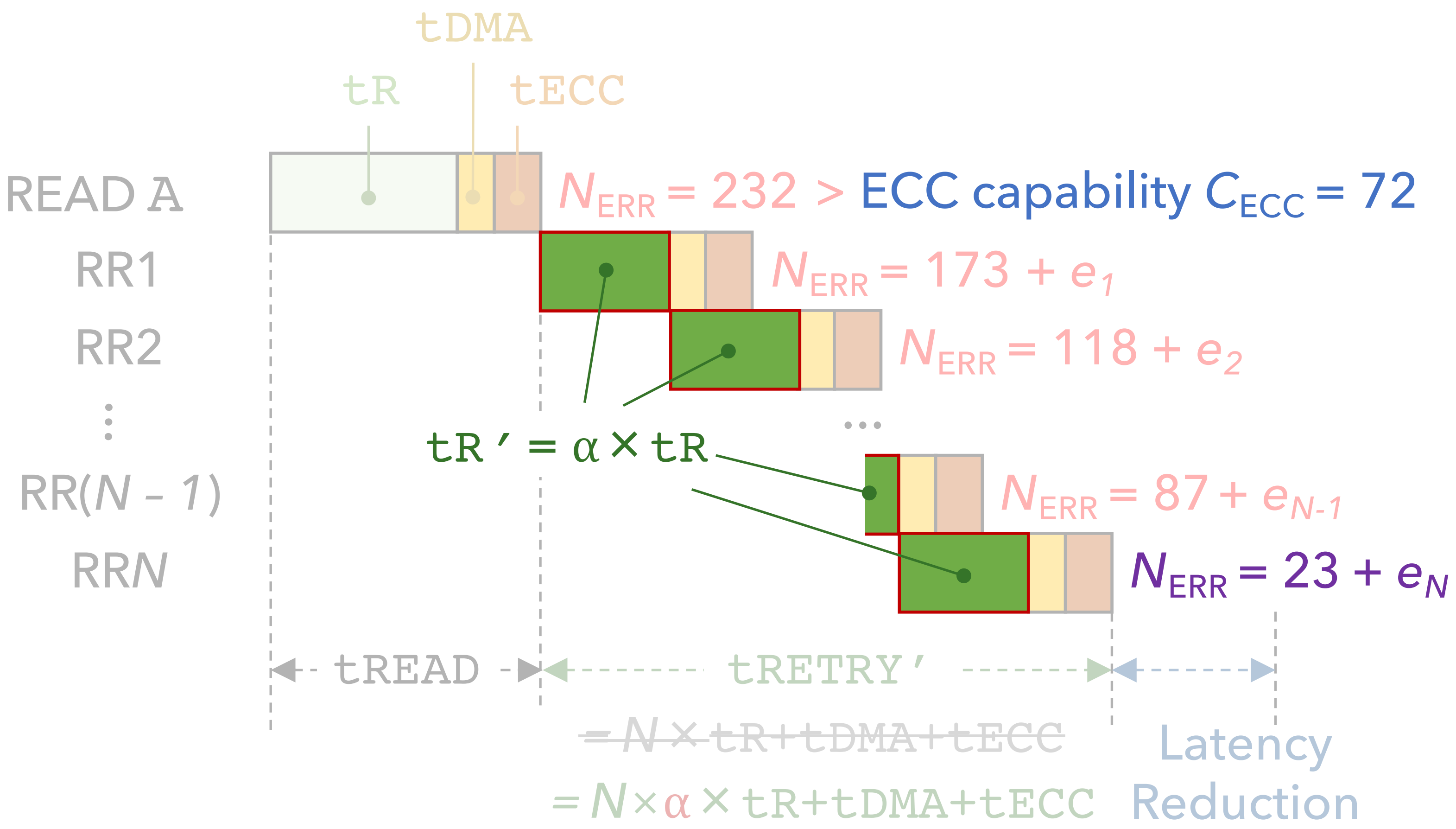
→ Further reduction in the read-retry latency

Adaptive Read-Retry (AR^2): Key Idea



AR^2 reduces tR in every retry step

Adaptive Read-Retry (AR^2): Key Idea



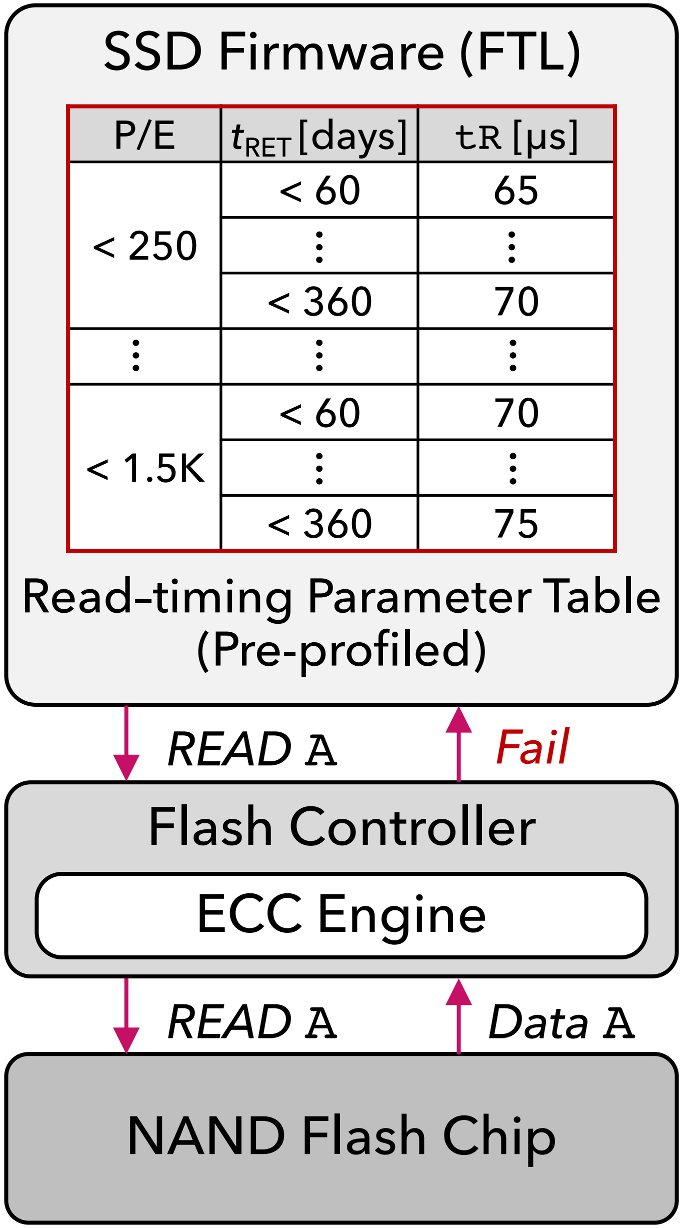
AR^2 reduces tR in every retry step ensuring $N_{ERR} < C_{ECC}$ in the final retry step

Real-Device Characterization

- 160 real 48-layer TLC NAND flash chips
- Observation 1: A large ECC margin in the final retry step even under worst-case operating conditions
  - At most 40 errors per KiB under 1–year retention time @ 2K program and erase (P/E) cycles
  - Use of near-optimal V_{REF} in the final retry step
- Observation 2: A large reliability margin incorporated in read-timing parameters
  - 25% `tR` reduction → At most 23 additional errors
  - Worst-case-based design due to process variations

AR^2 can easily work in commodity NAND flash chips w/ at least 25% `tR` reduction

P&AR²: Design



SSD Firmware (FTL)

P/E	t_{RET} [days]	tR [μs]
< 250	< 60	65
	⋮	⋮
	< 360	70
⋮	⋮	⋮
< 1.5K	< 60	70
	⋮	⋮
	< 360	75

Read-timing Parameter Table (Pre-profiled)

READ A

Fail

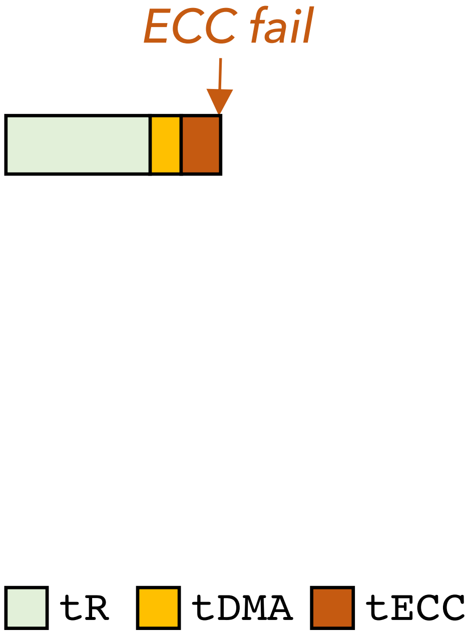
Flash Controller

ECC Engine

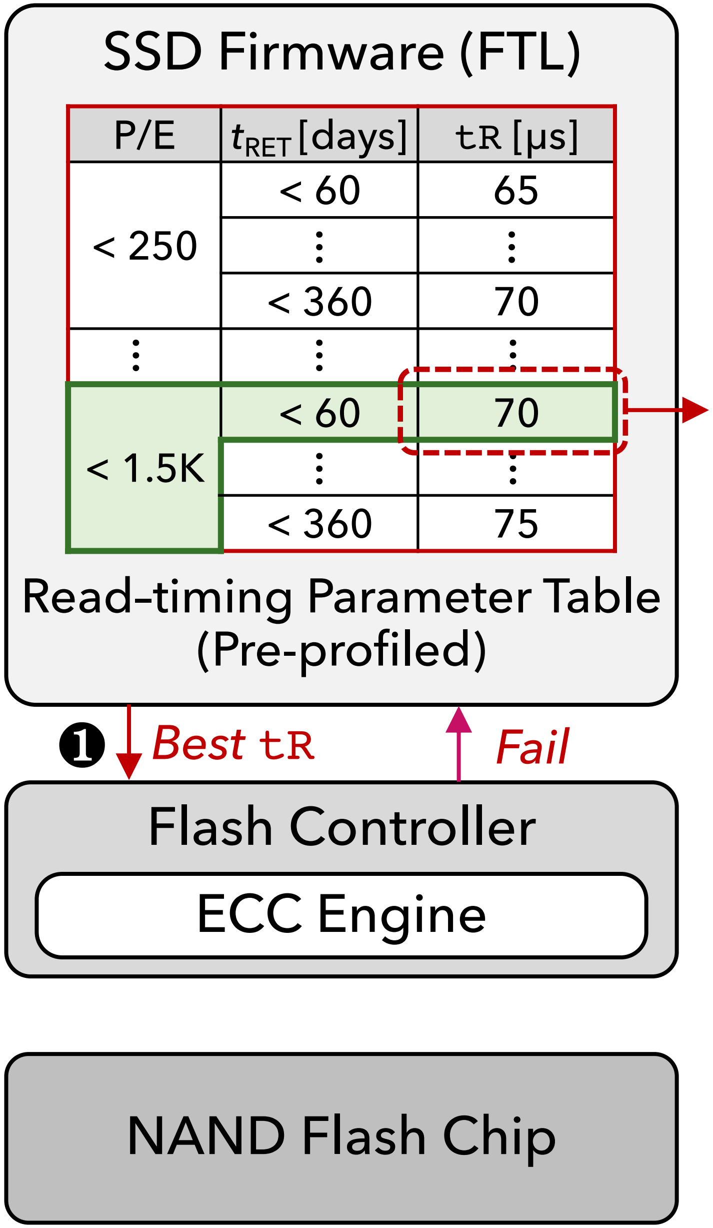
READ A

Data A

NAND Flash Chip



P&AR^2: Design

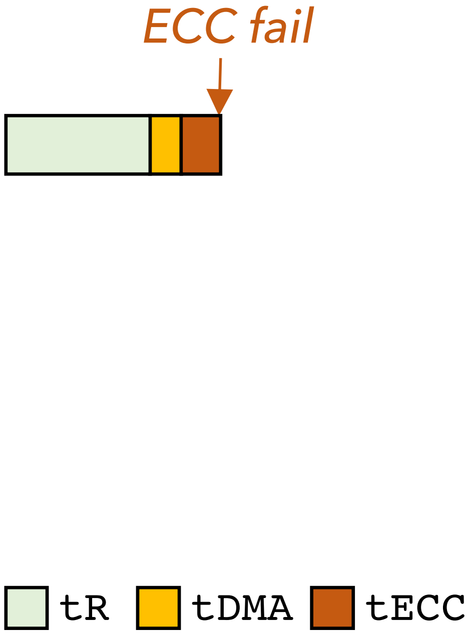


SSD Firmware (FTL)

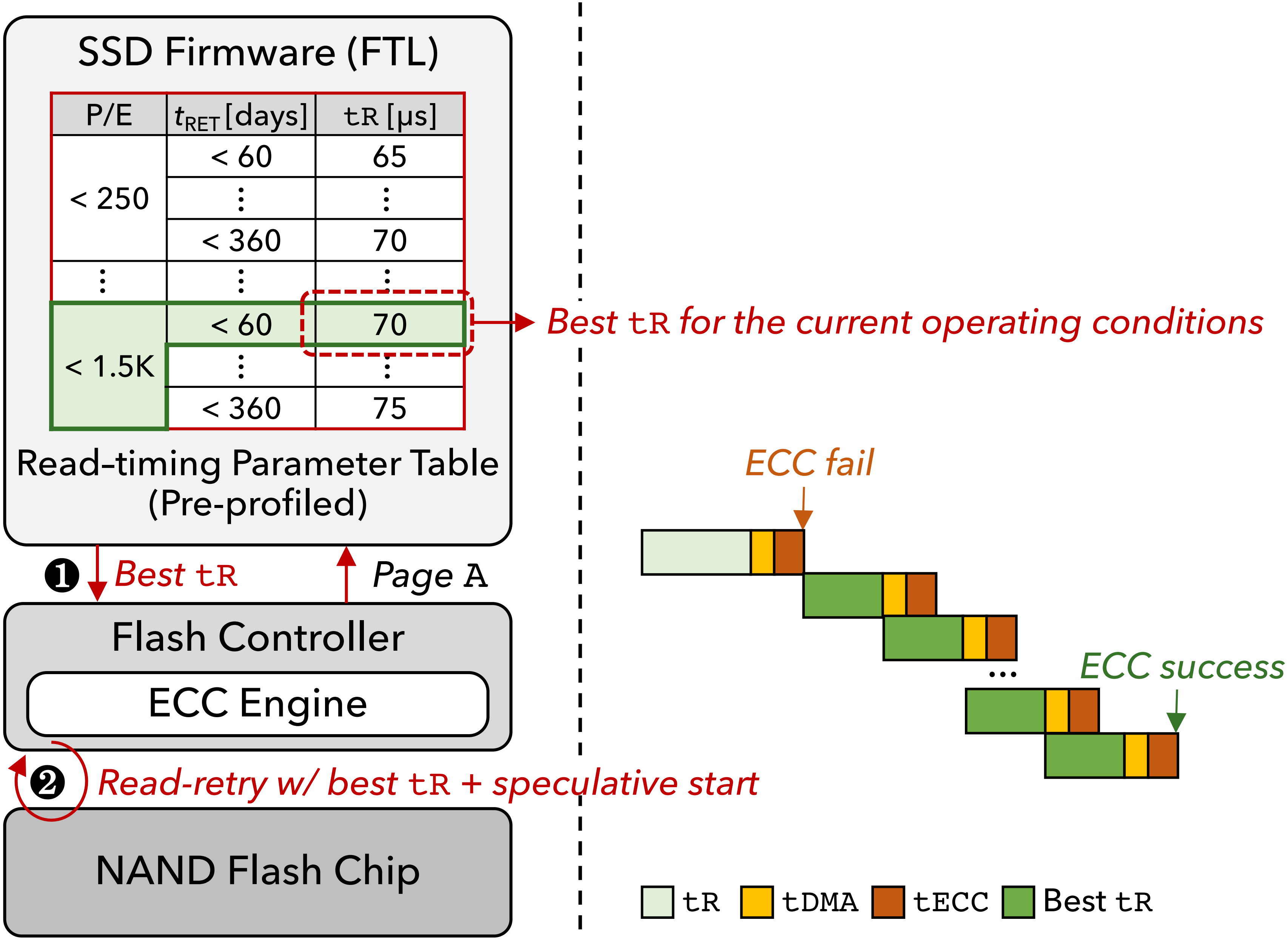
P/E	t_{RET} [days]	tR [μs]
< 250	< 60	65
	⋮	⋮
	< 360	70
⋮	⋮	⋮
< 1.5K	< 60	70
	⋮	⋮
	< 360	75

Read-timing Parameter Table
(Pre-profiled)

Best tR for the current operating conditions



P&AR^2: Design



P/E	t_{RET} [days]	tR [μs]
< 250	< 60	65
	⋮	⋮
	< 360	70
⋮	⋮	⋮
< 1.5K	< 60	70
	⋮	⋮
	< 360	75

Read-timing Parameter Table (Pre-profiled)

Best tR *for the current operating conditions*

❶ *Best* tR

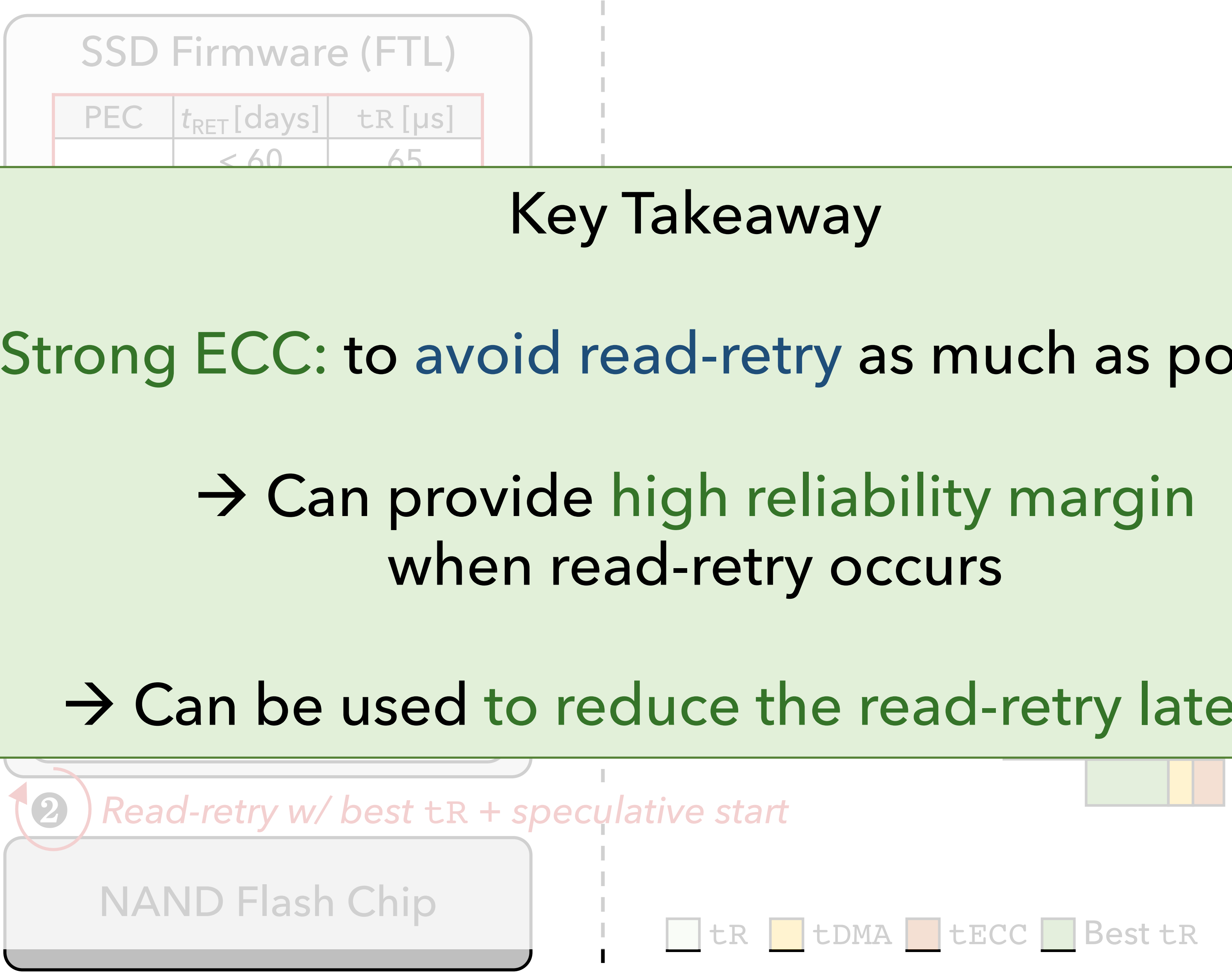
Page A

❷ *Read-retry w/ best* tR *+ speculative start*

ECC fail

ECC success

P&AR^2: Design



PEC	t_{RET} [days]	tR [μs]
	< 60	65

Key Takeaway

Strong ECC: to avoid read-retry as much as possible

→ Can provide high reliability margin
when read-retry occurs

→ Can be used to reduce the read-retry latency

P&AR^2: Outline

- Read-Retry in Modern NAND Flash-Based SSDs

- PR^2: Pipelined Read-Retry

- AR^2: Adaptive Read-Retry

- Evaluation Results

Evaluation Results

- Simulation using MQSim [Tavakkol+, FAST18] and 12 real-world workloads

- Our proposal improves SSD response time by
  - Up to 51% (35% on average) compared to a high-end SSD w/o read-retry mitigation
  - Up to 32% (17% on average) compared to a state-of-the-art read-retry mitigation technique [Shim+, MICRO19]

Outline

- NAND Flash Basics

- Read-Retry in Modern NAND Flash-Based SSDs

- Data Sanitization in Modern NAND Flash-Based SSDs

Access Control-Based Data Sanitization

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

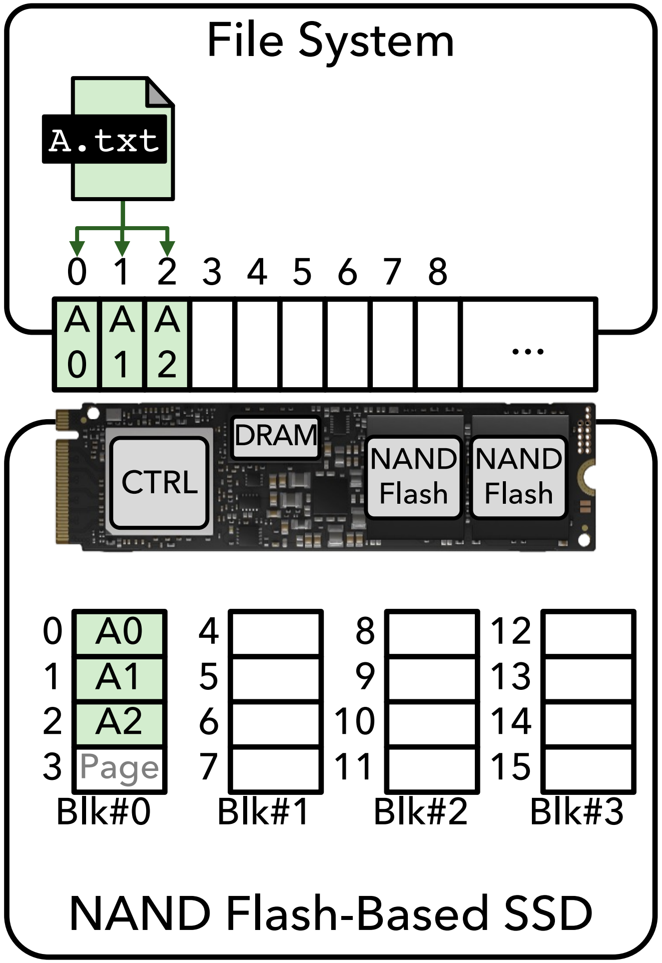
Jihong Kim
jihong@davinci.snu.ac.kr
Seoul National University

*The first two authors contributed equally to this research.

The 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)

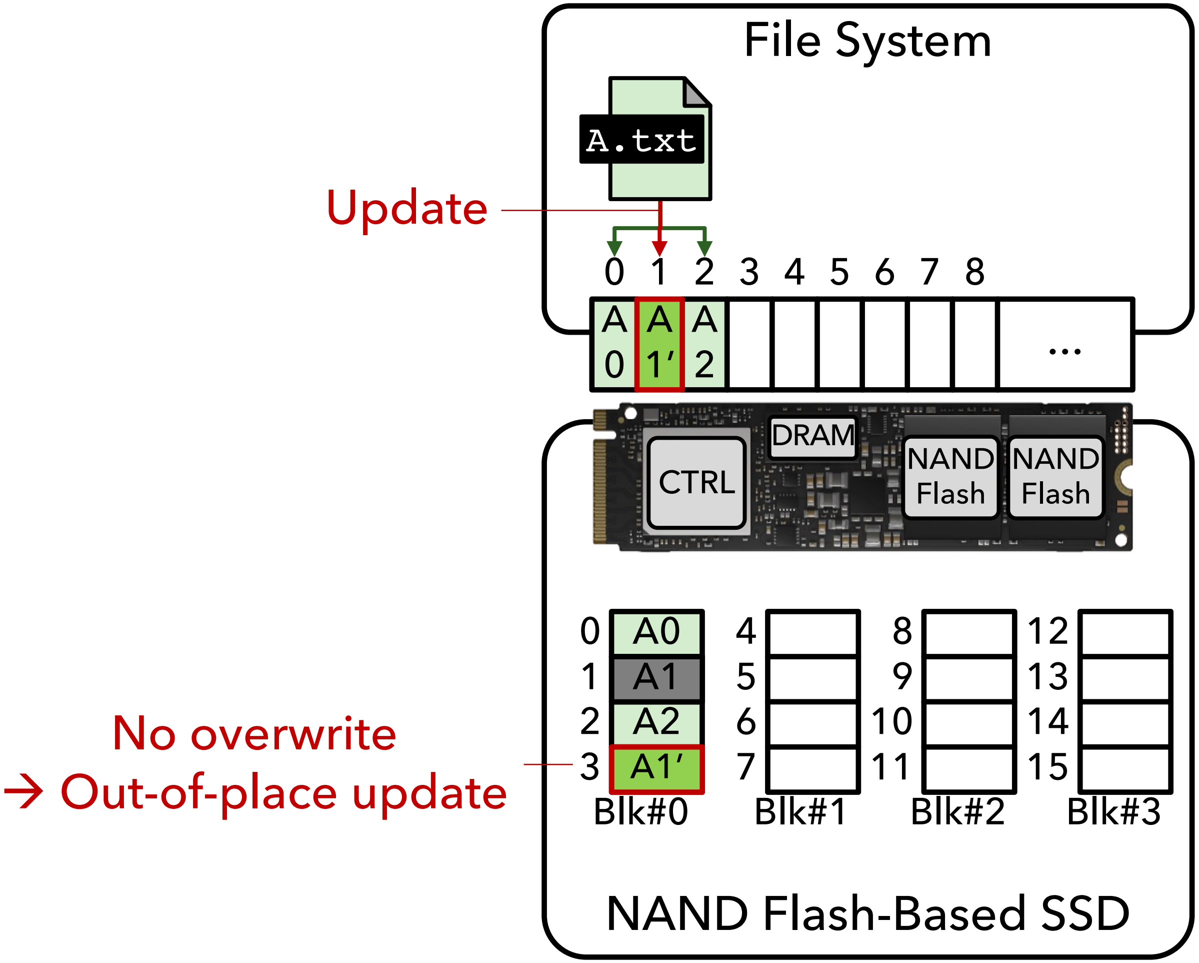
Data-Remanence Problem in SSDs

- Deleted data can remain in SSDs for indefinite time



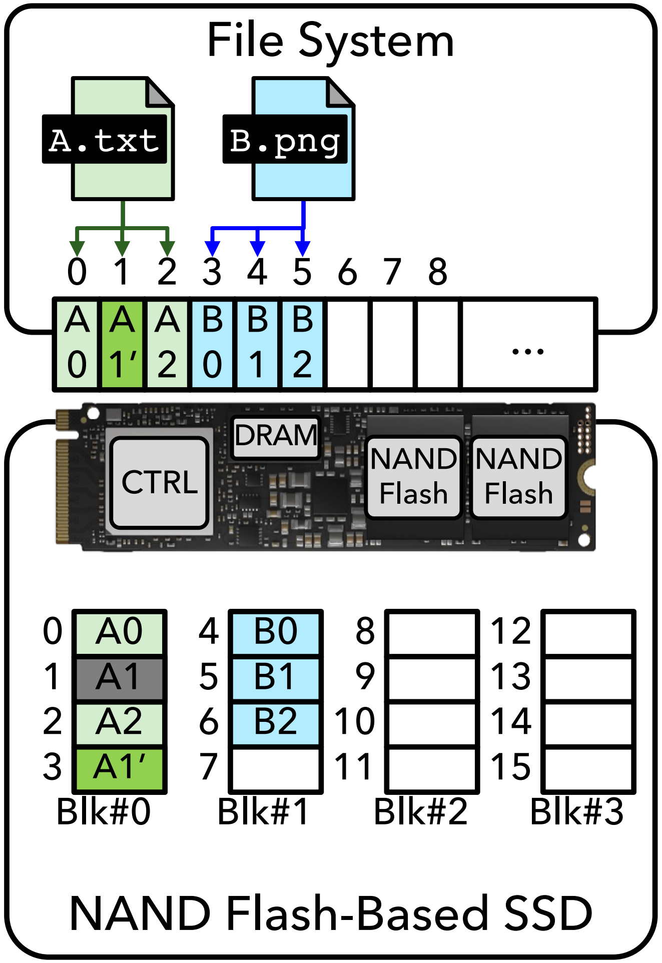
Data-Remanence Problem in SSDs

- Deleted data can remain in SSDs for indefinite time



Data-Remanence Problem in SSDs

- Deleted data can remain in SSDs for indefinite time

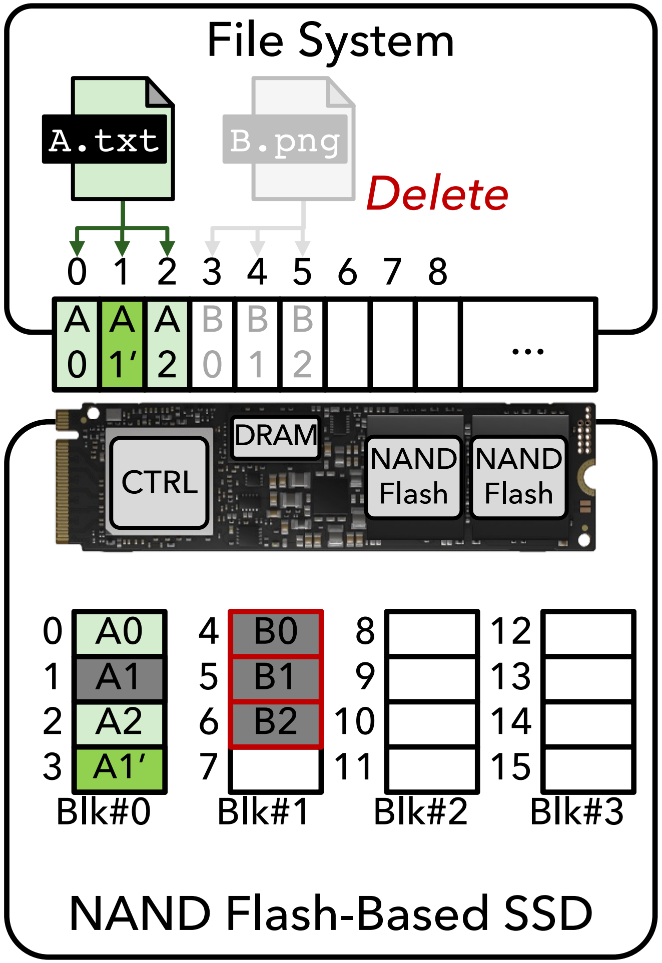


Data-Remanence Problem in SSDs

- Deleted data can remain in SSDs for indefinite time

Q: When is a page erased?

A: Only in garbage collection
= when running out of free pages



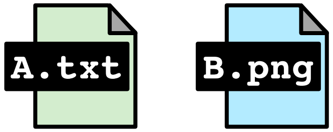
Data-Recovery Attack

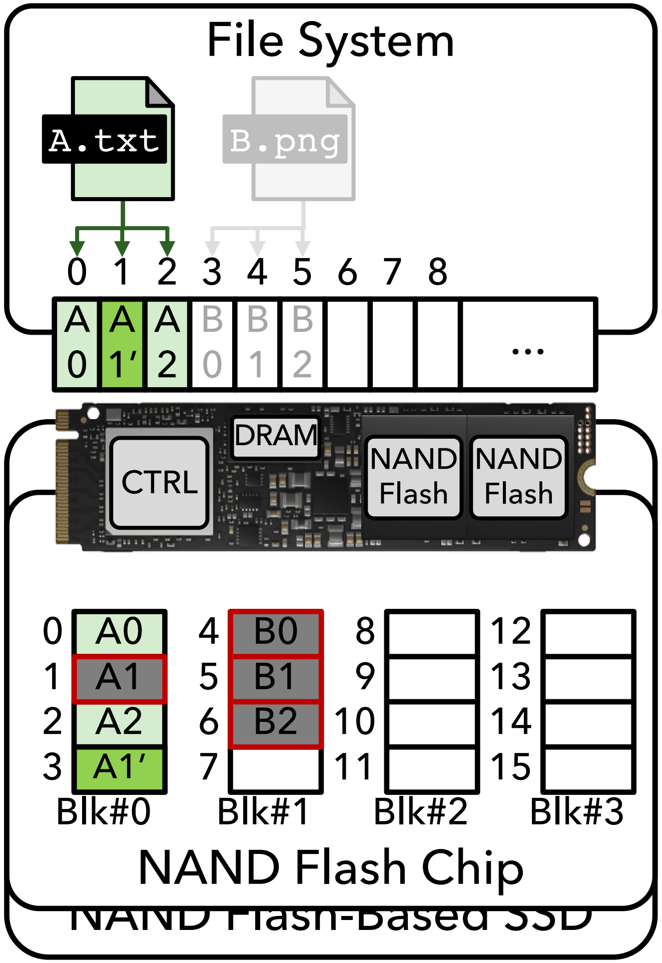
- System requirement: Obsolete data must be inaccessible

1. Detach the SSD
2. Detach the chip
3. Direct access to the chip

A0	A1	A2	B1

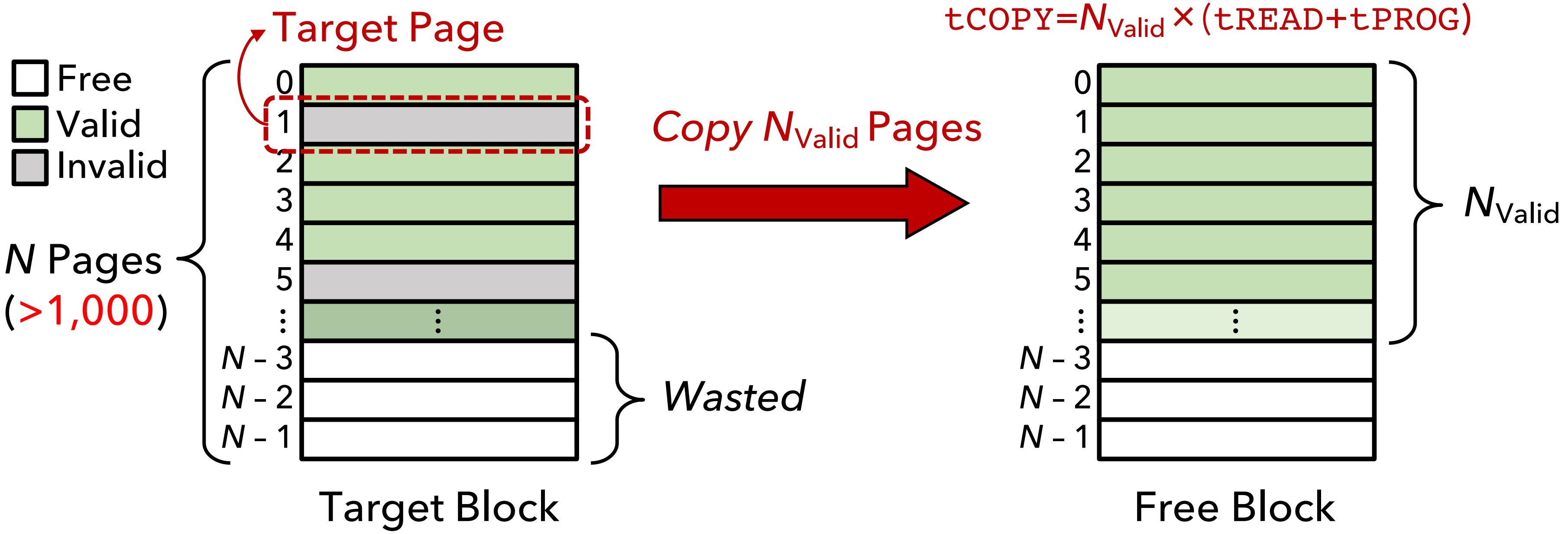
4. Run forensic tools





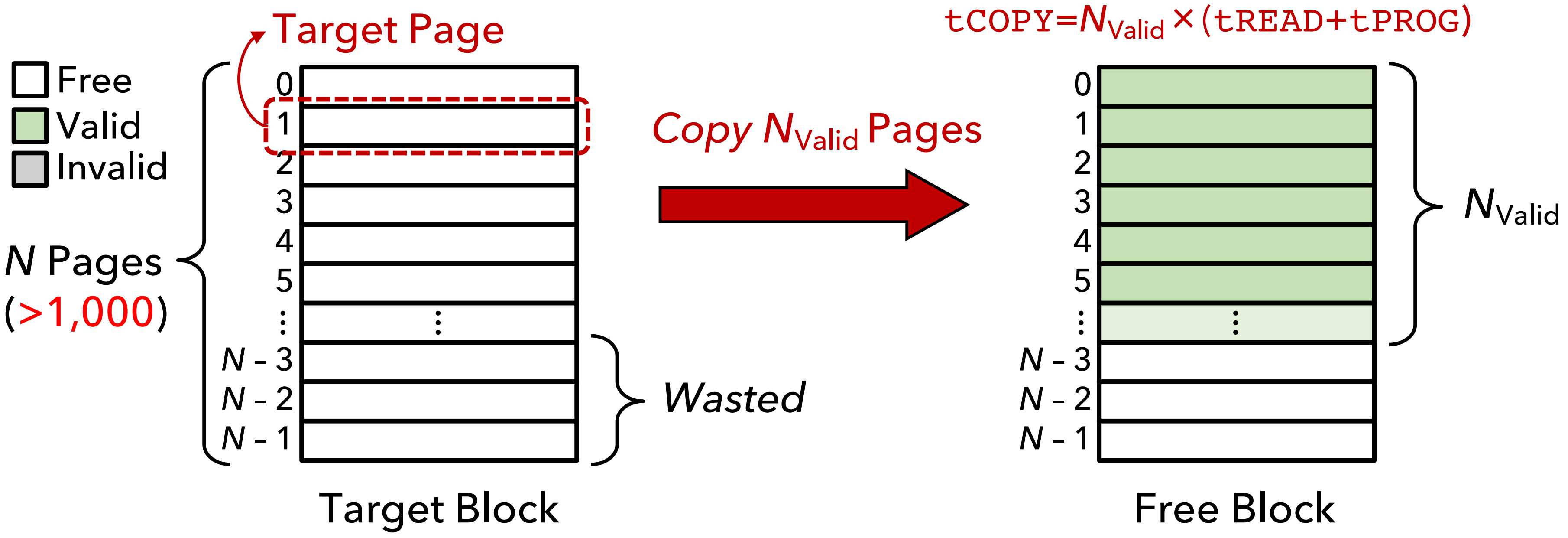
Existing Solutions

- Why not immediately erase an invalid page?
  - Erase unit: a block (> 1,000 pages)



Existing Solutions

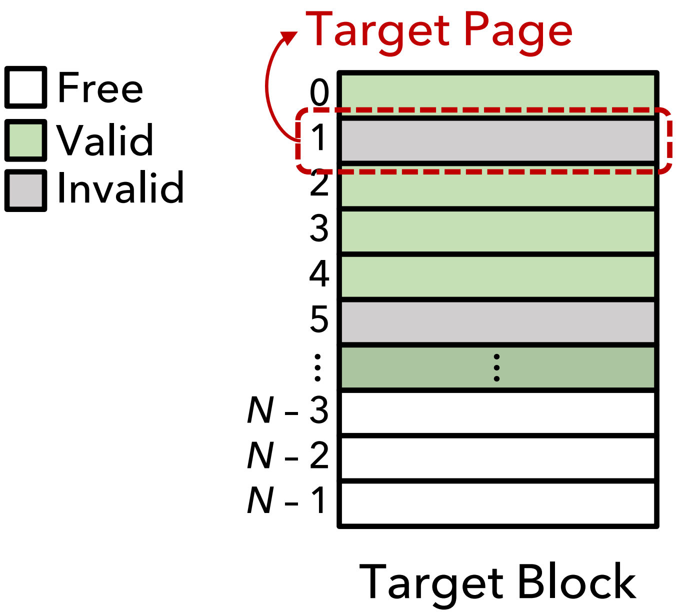
- Why not immediately erase an invalid page?
  - Erase unit: a block (> 1,000 pages)

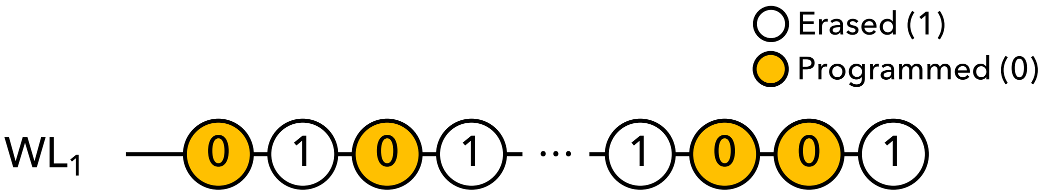


Immediate block erasure causes prohibitive performance and lifetime overhead

Existing Solutions

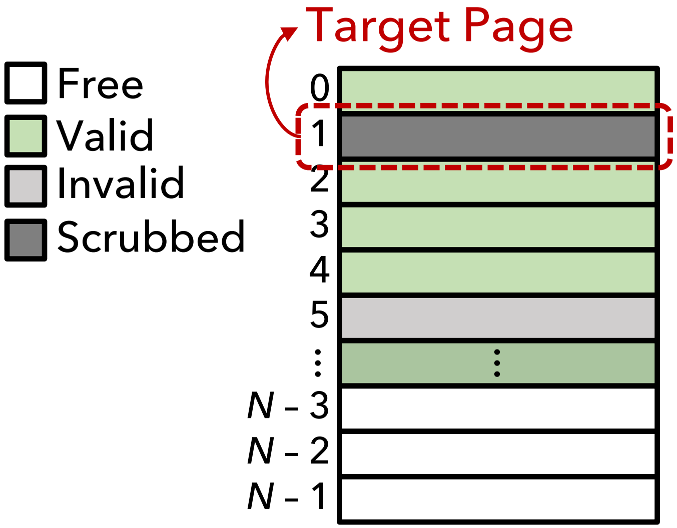
- Scrubbing [Wei+, FAST'11]
  - Reprograms all the flash cells storing an invalid page



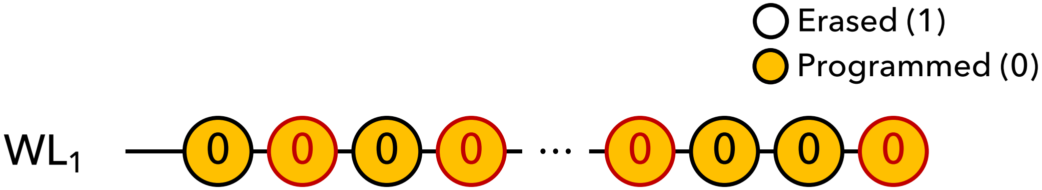


Existing Solutions

- Scrubbing [Wei+, FAST'11]
  - Reprograms all the flash cells storing an invalid page



Target Block



Scrubbing: Reprogram the WL (to all '0's)

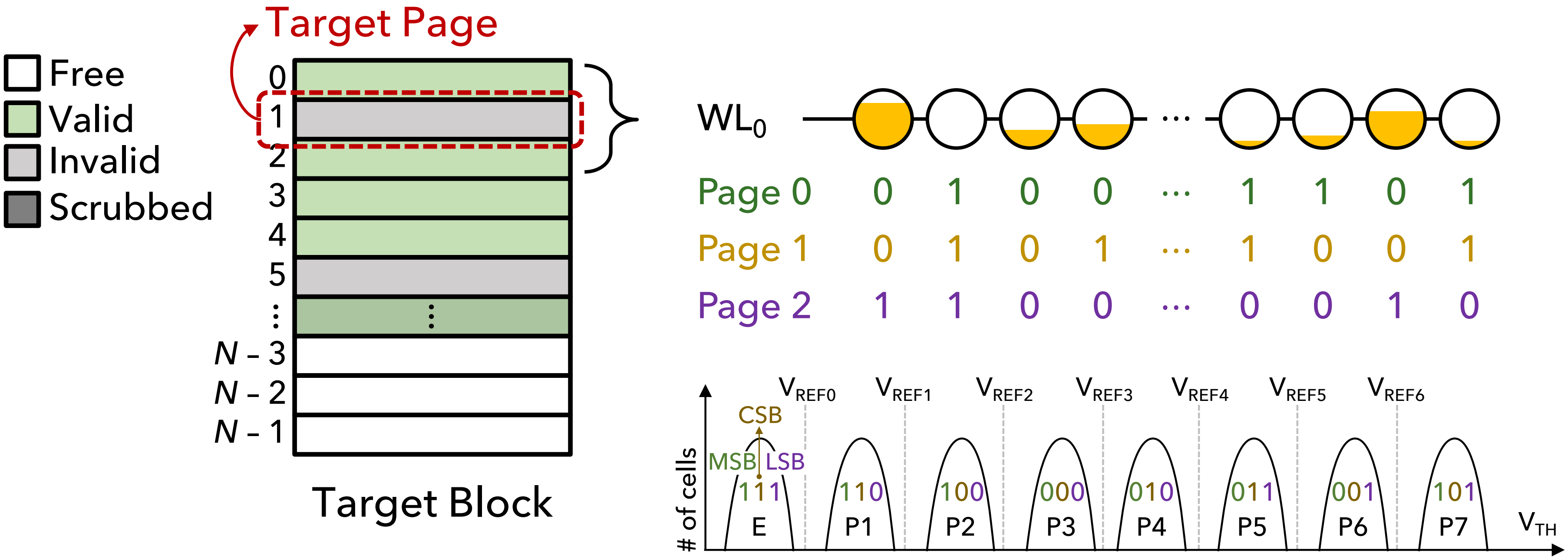
Overwriting is infeasible usually, except when the data is all '0's

Destroys the page data w/o block erasure

Problem solved?

Existing Solutions

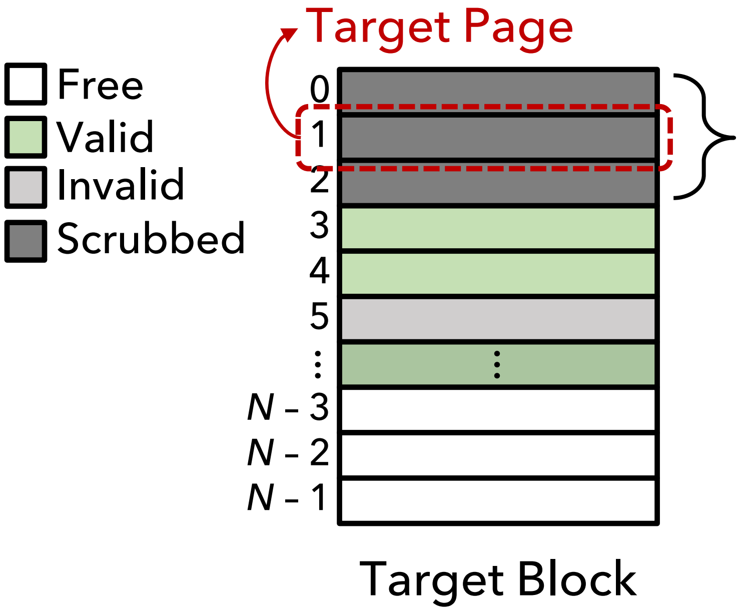
- Scrubbing [Wei+, FAST'11]
  - Problem1: MLC NAND flash stores multiple pages in a WL

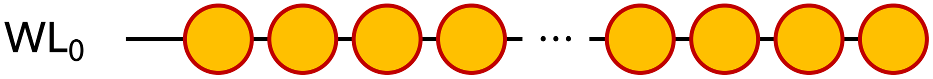


Page 0	0	1	0	0	⋯	1	1	0	1
Page 1	0	1	0	1	⋯	1	0	0	1
Page 2	1	1	0	0	⋯	0	0	1	0

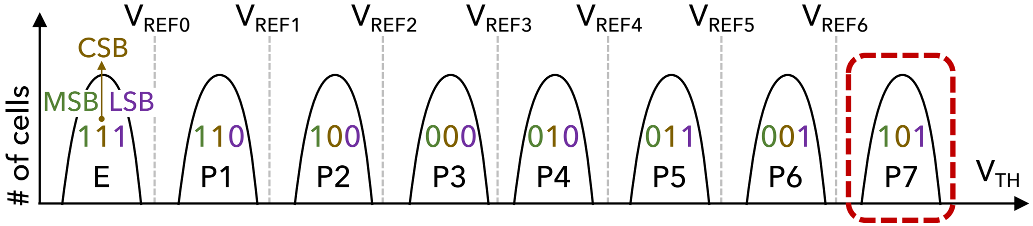
Existing Solutions

- Scrubbing [Wei+, FAST'11]
  - Problem1: MLC NAND flash stores multiple pages in a WL





Page 0	1	1	1	1	···	1	1	1	1
Page 1	0	0	0	0	···	0	0	0	0
Page 2	1	1	1	1	···	1	1	1	1

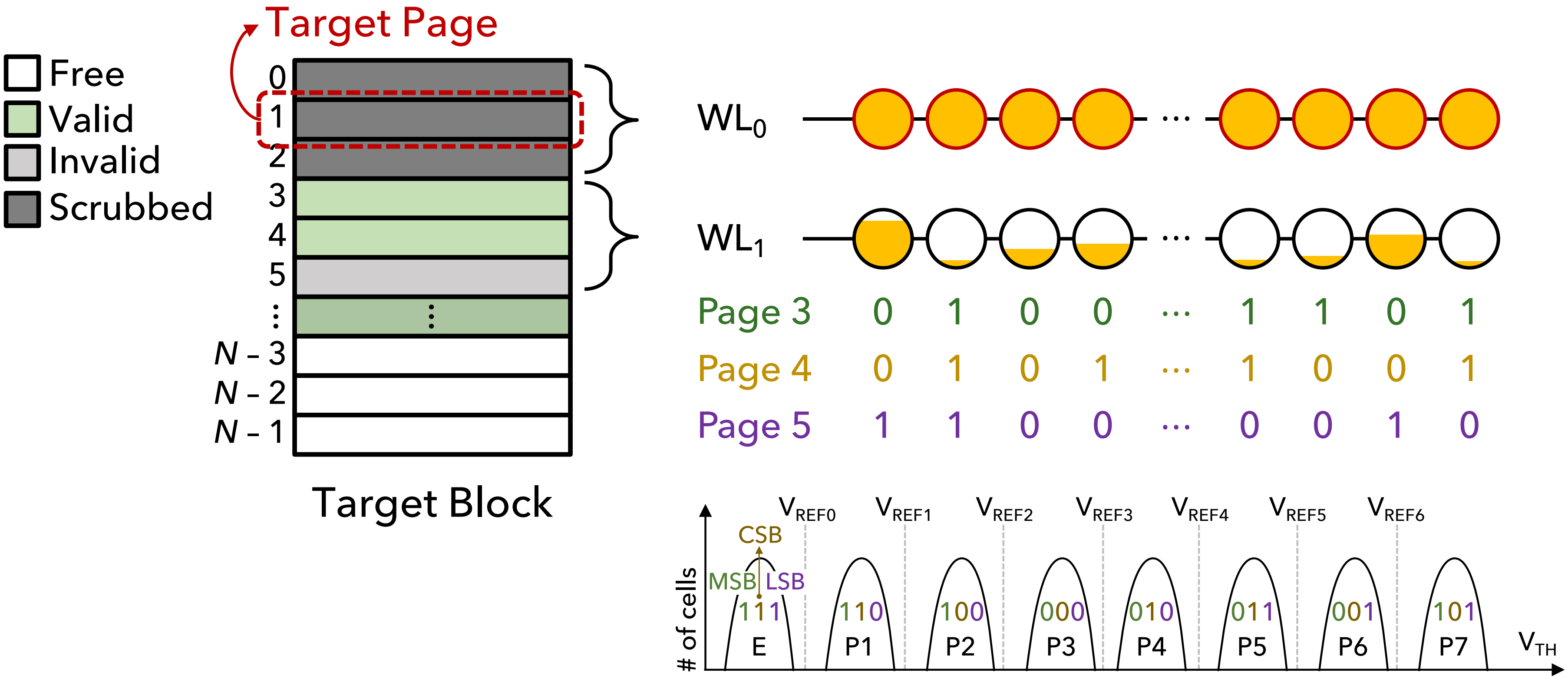


Scrubbing in MLC NAND flash memory

→ Destroys other valid pages → Copy overheads

Existing Solutions

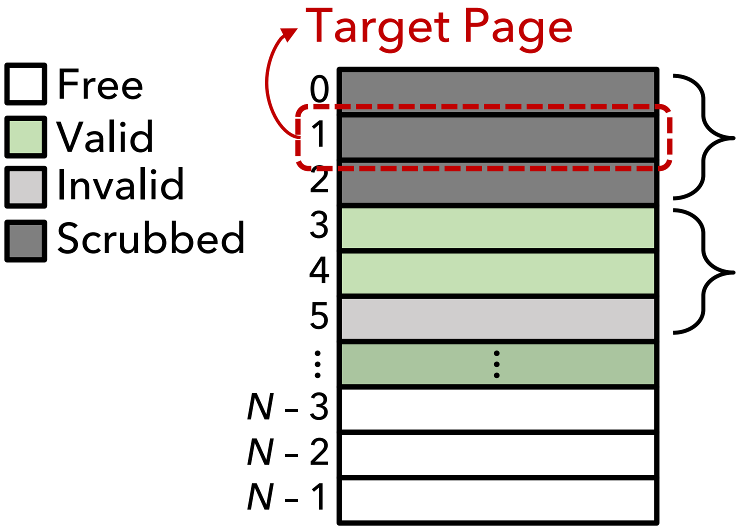
- Scrubbing [Wei+, FAST'11]
  - Problem 1: MLC NAND flash stores multiple pages in a WL
  - Problem 2: Program interference



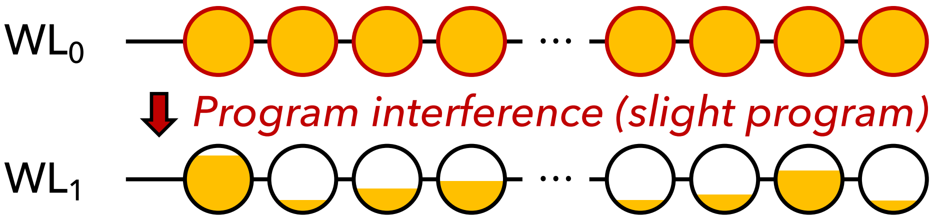
Page 3	0	1	0	0	···	1	1	0	1
Page 4	0	1	0	1	···	1	0	0	1
Page 5	1	1	0	0	···	0	0	1	0

Existing Solutions

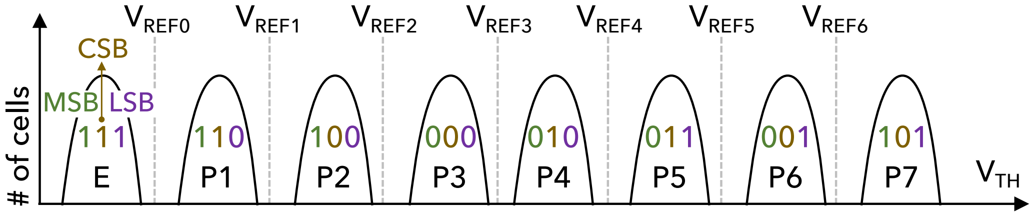
- Scrubbing [Wei+, FAST'11]
  - Problem 1: MLC NAND flash stores multiple pages in a WL
  - Problem 2: Program interference



Target Block

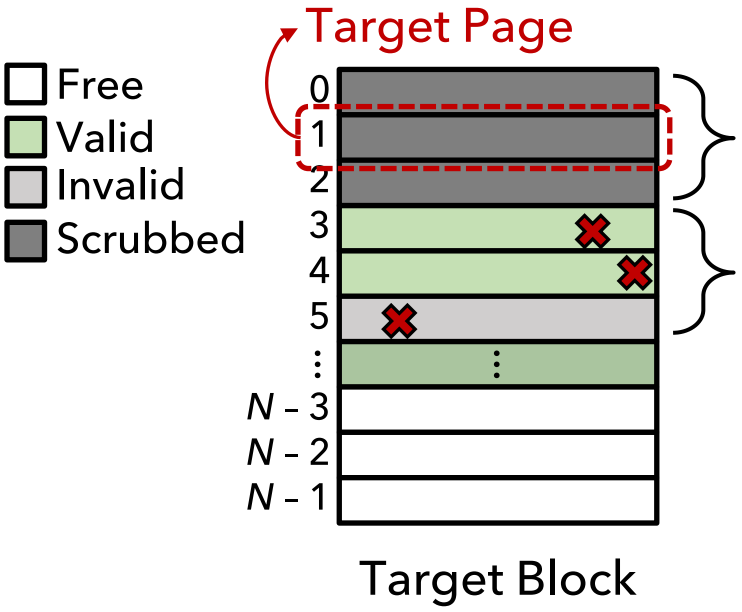


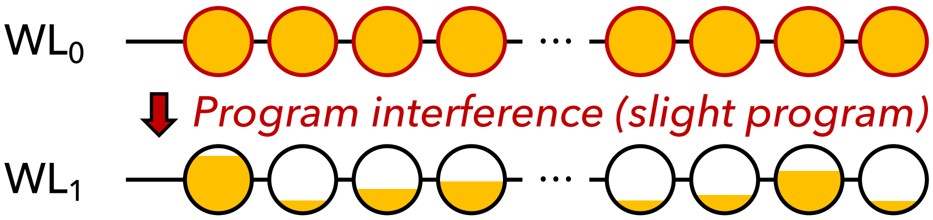
Page 3	0	1	0	0	⋯	1	1	0	1
Page 4	0	1	0	1	⋯	1	0	0	1
Page 5	1	1	0	0	⋯	0	0	1	0



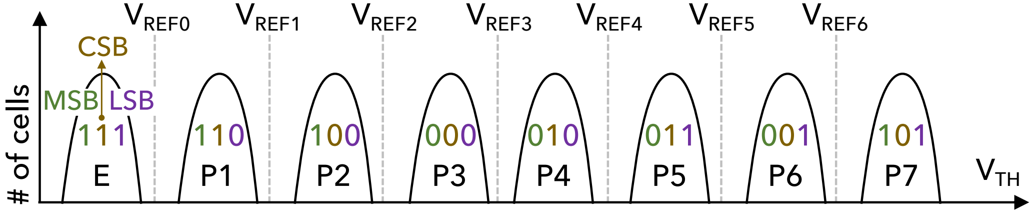
Existing Solutions

- Scrubbing [Wei+, FAST'11]
  - Problem 1: MLC NAND flash stores multiple pages in a WL
  - Problem 2: Program interference



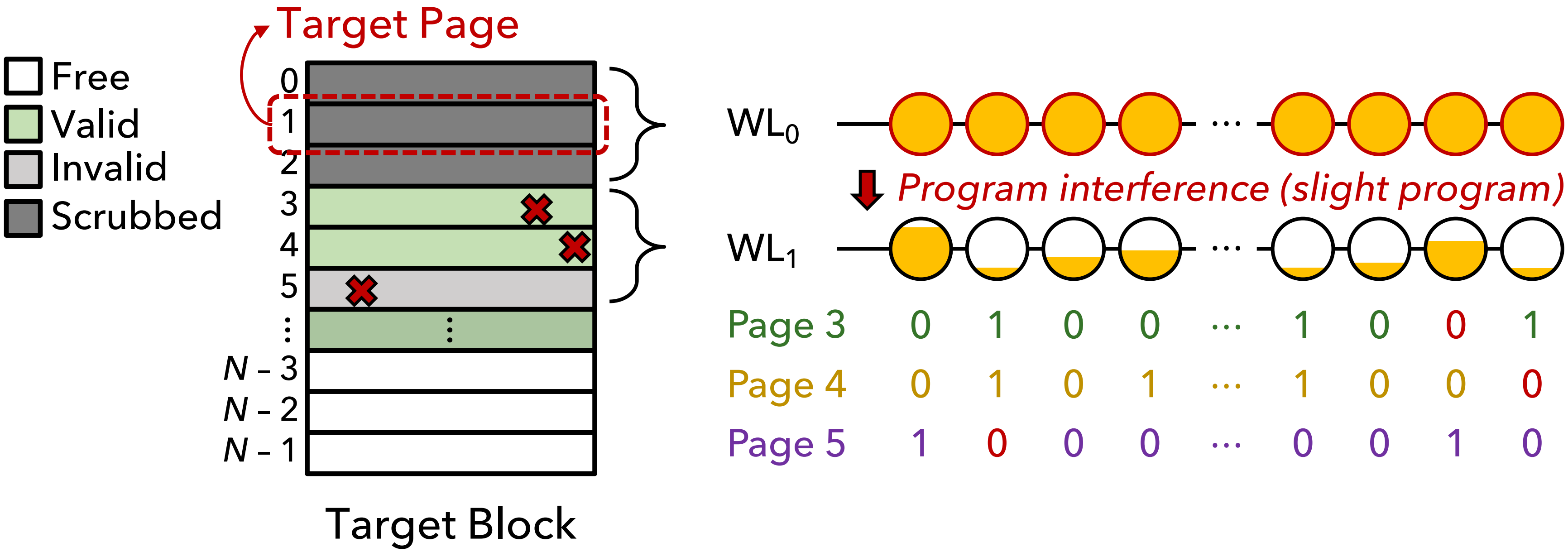


Page 3	0	1	0	0	⋯	1	0	0	1
Page 4	0	1	0	1	⋯	1	0	0	0
Page 5	1	0	0	0	⋯	0	0	1	0



Existing Solutions

- Scrubbing [Wei+, FAST'11]
  - Problem 1: MLC NAND flash stores multiple pages in a WL
  - Problem 2: Program interference



Page 3	0	1	0	0	⋯	1	0	0	1
Page 4	0	1	0	1	⋯	1	0	0	0
Page 5	1	0	0	0	⋯	0	0	1	0

Existing solutions incur performance, lifetime, and reliability problems in modern NAND flash memory

Evanesco: Outline

- Data Remanace in NAND Flash-Based SSDs

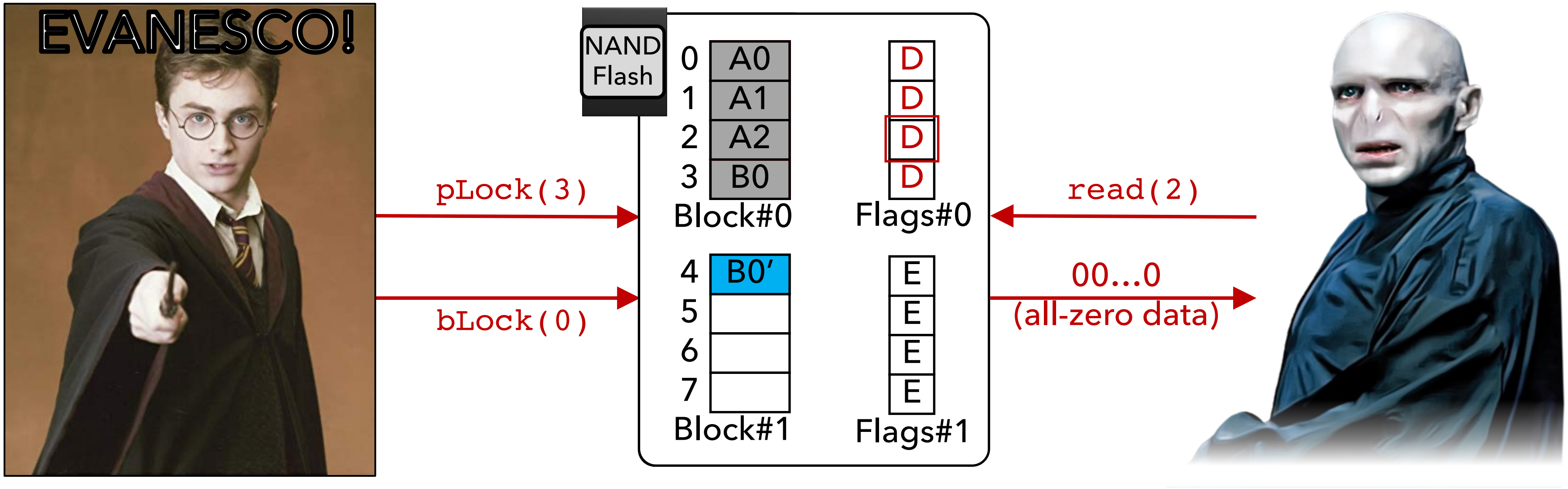
- Evanesco: Access Control–Based Sanitization

- Evaluation Results

Our Solution: Evanesco

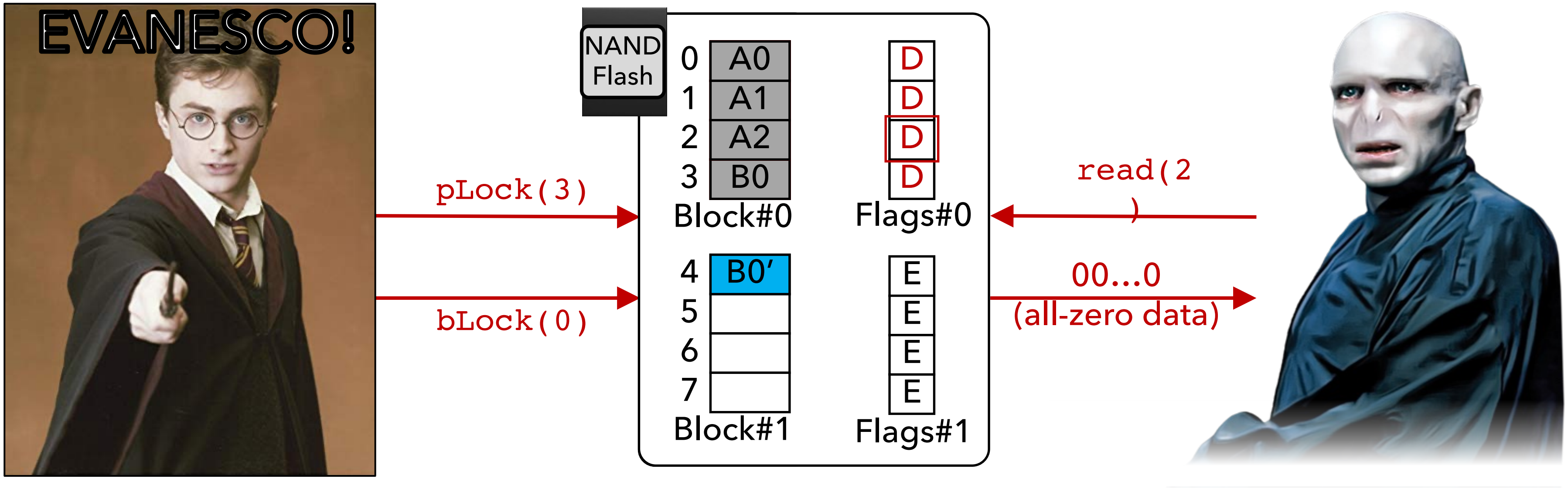
- Allows a NAND flash chip to be aware of data validity
  - On-chip access control to avoid access to invalid data
  - Low overhead: No copy operations
  - High reliability: No program interference

- Two new NAND commands: `pageLock` and `blockLock`



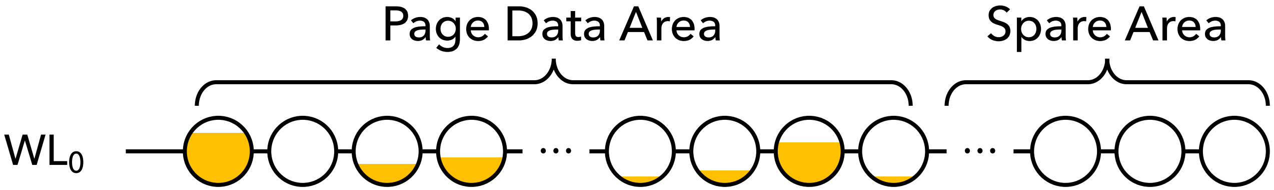
Evanesco: Requirements

- Keep access-permission flags in a non-volatile manner
- Access-control logic inside a NAND flash chip
- Minimal area overhead → High chip densitity is paramount



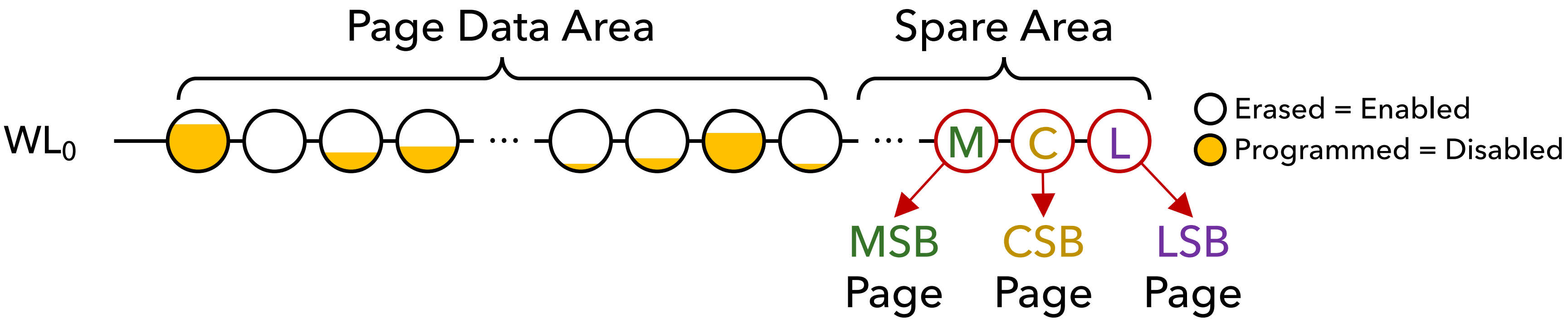
pLock: Page-Level Data Sanitization

- On-chip access-permission flags: Spare cells in each WL



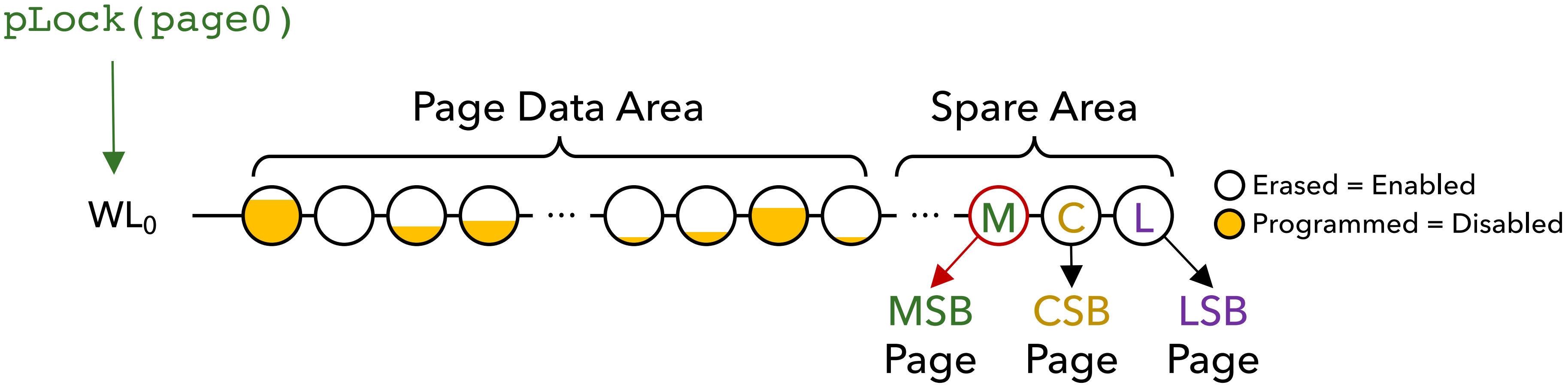
pLock: Page-Level Data Sanitization

- On-chip access-permission flags: Spare cells in each WL



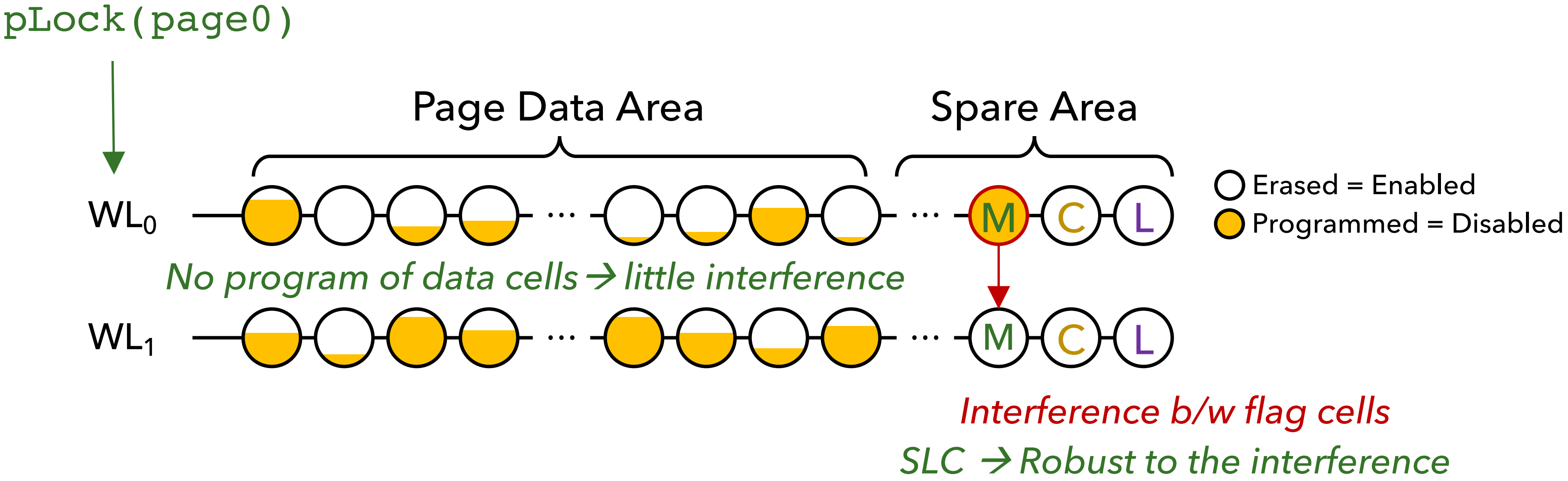
pLock: Page-Level Data Sanitization

- On-chip access-permission flags: Spare cells in each WL



pLock: Page-Level Data Sanitization

- On-chip access-permission flags: Spare cells in each WL



pLock(page0)

Page Data Area

Spare Area

WL0

WL1

○ Erased = Enabled

● Programmed = Disabled

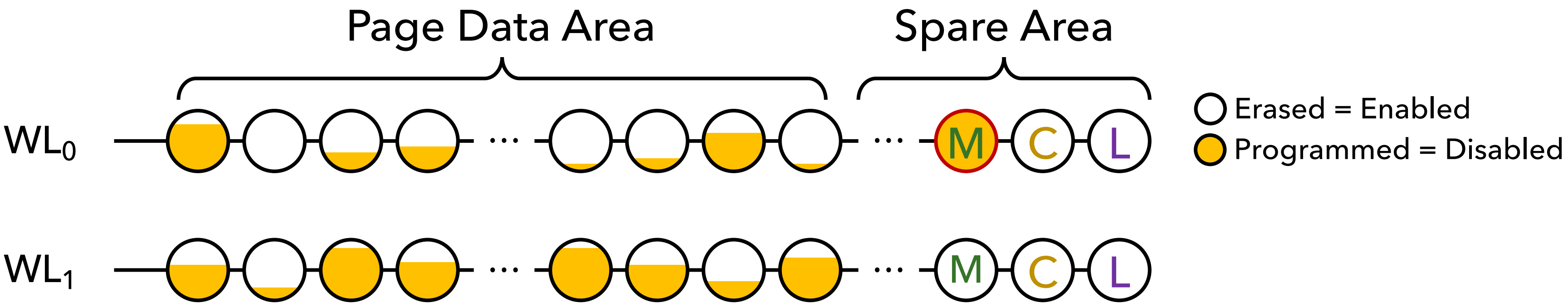
No program of data cells → little interference

Interference b/w flag cells

SLC → Robust to the interference

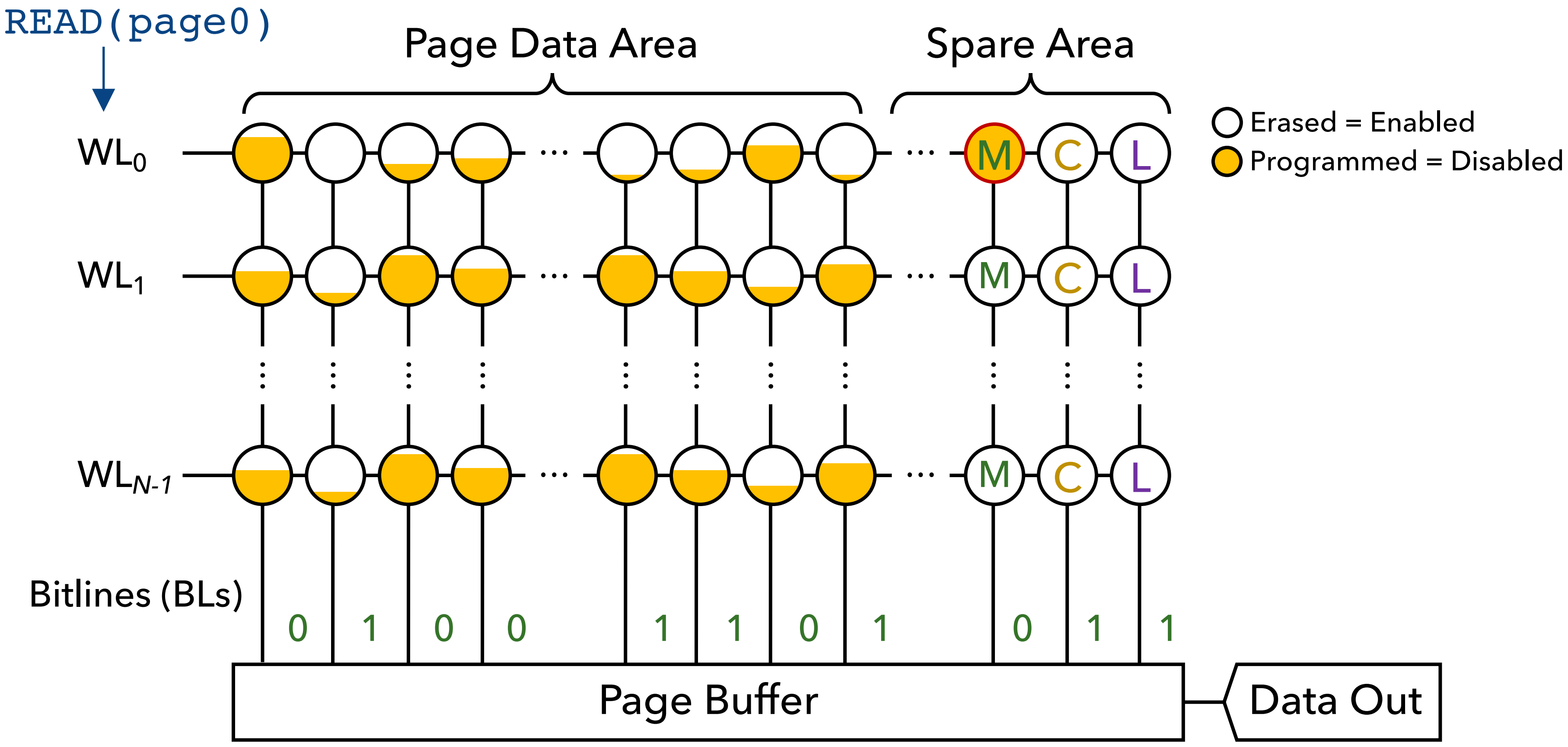
pLock: Page-Level Data Sanitization

- On-chip access-permission flags: Spare cells in each WL
- On-chip access control logic: Small changes to data path



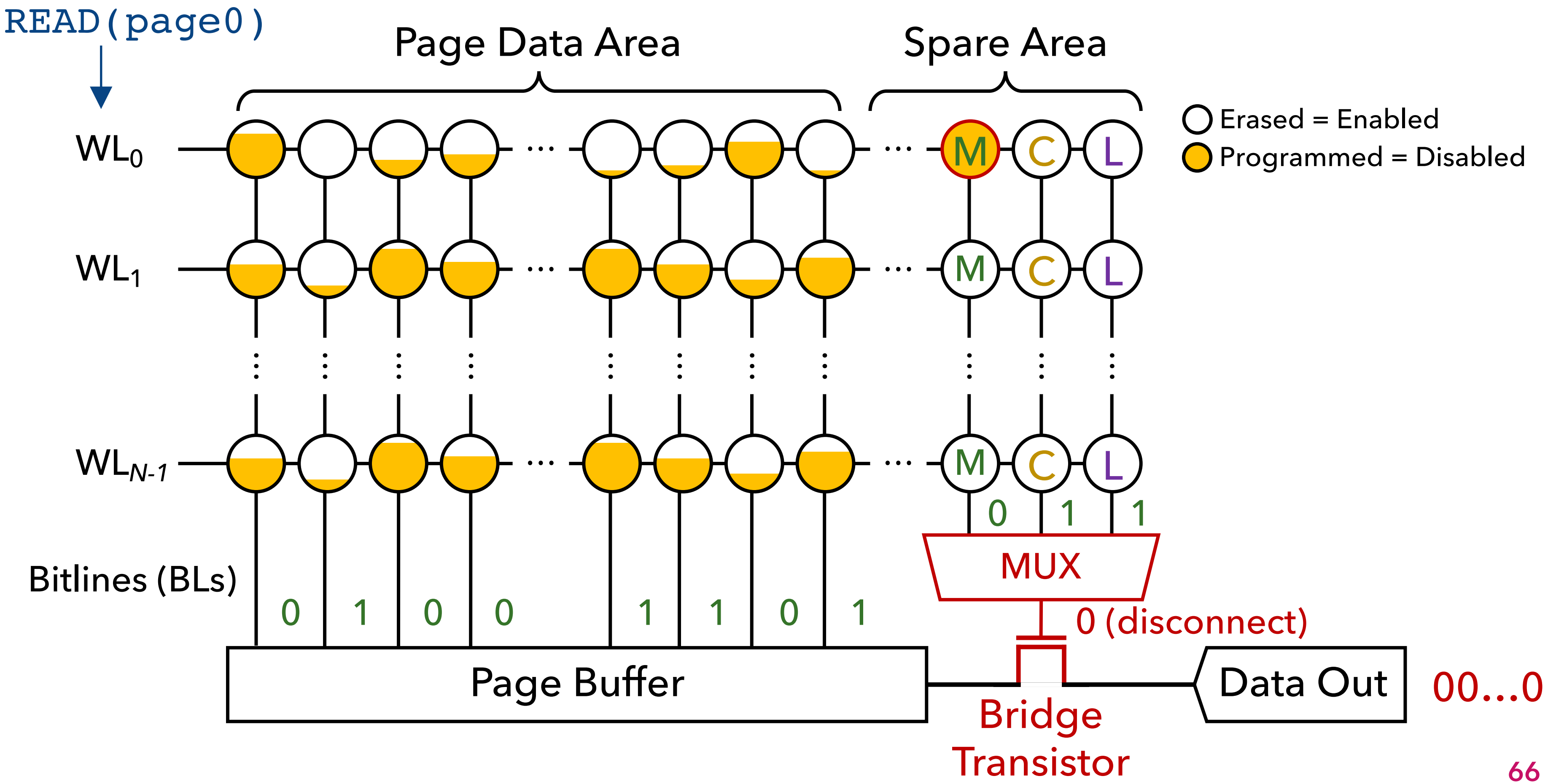
pLock: Page-Level Data Sanitization

- On-chip access-permision flags: Spare cells in each WL
- On-chip access control logic: Small changes to data path



pLock: Page-Level Data Sanitization

- On-chip access-permision flags: Spare cells in each WL
- On-chip access control logic: Small changes to data path`



Real-Device Characterization

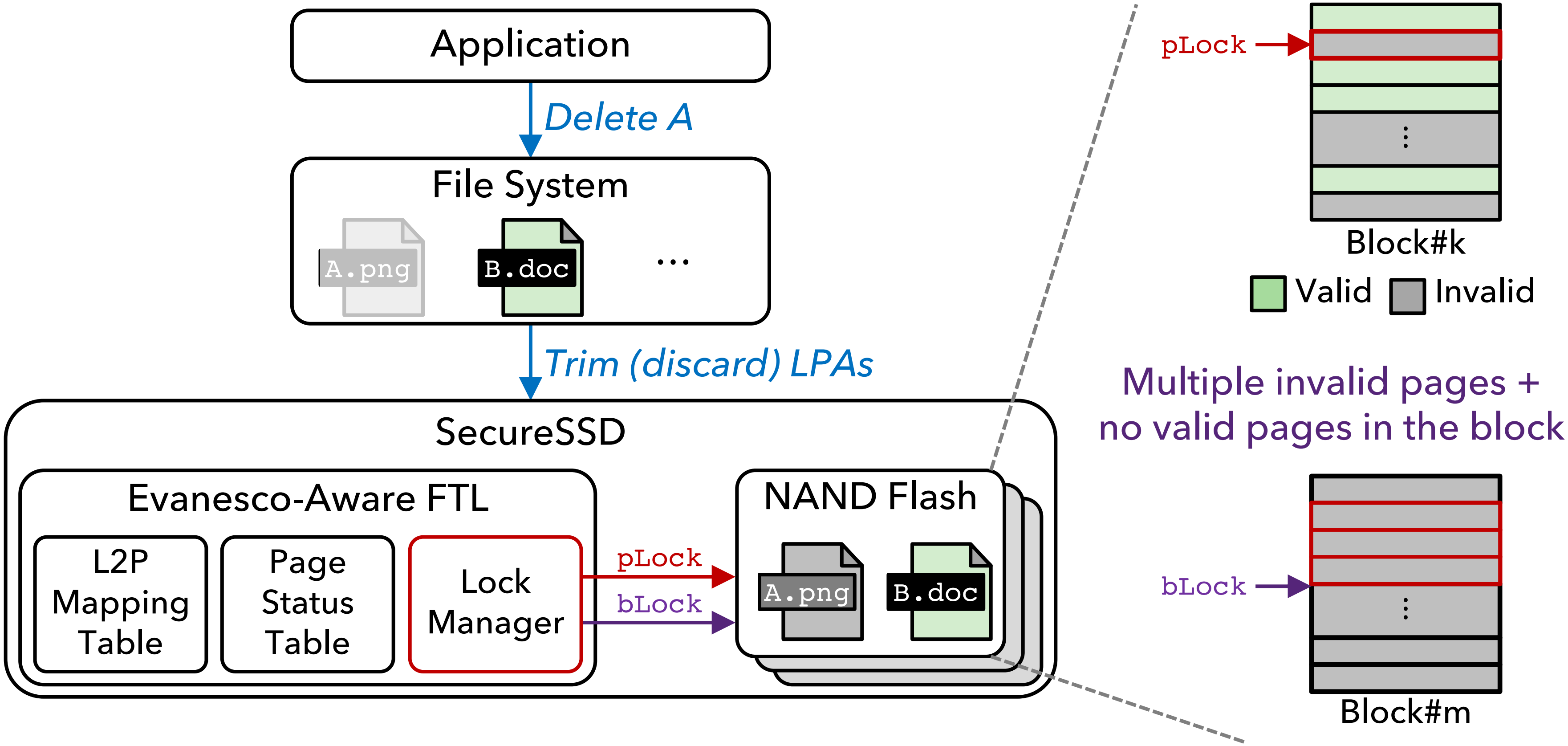
- Using 160 real 48-layer TLC NAND flash chips
- No reliability degradation for stored data
- `tPLOCK` = 100 μs, `tBLOCK` = 300 μs

Evanesco: No copy operation, no reliability issues w/ minimal changes to NAND flash chip designs

But the performance overhead is not negligible

SecureSSD: System-Level Optimization

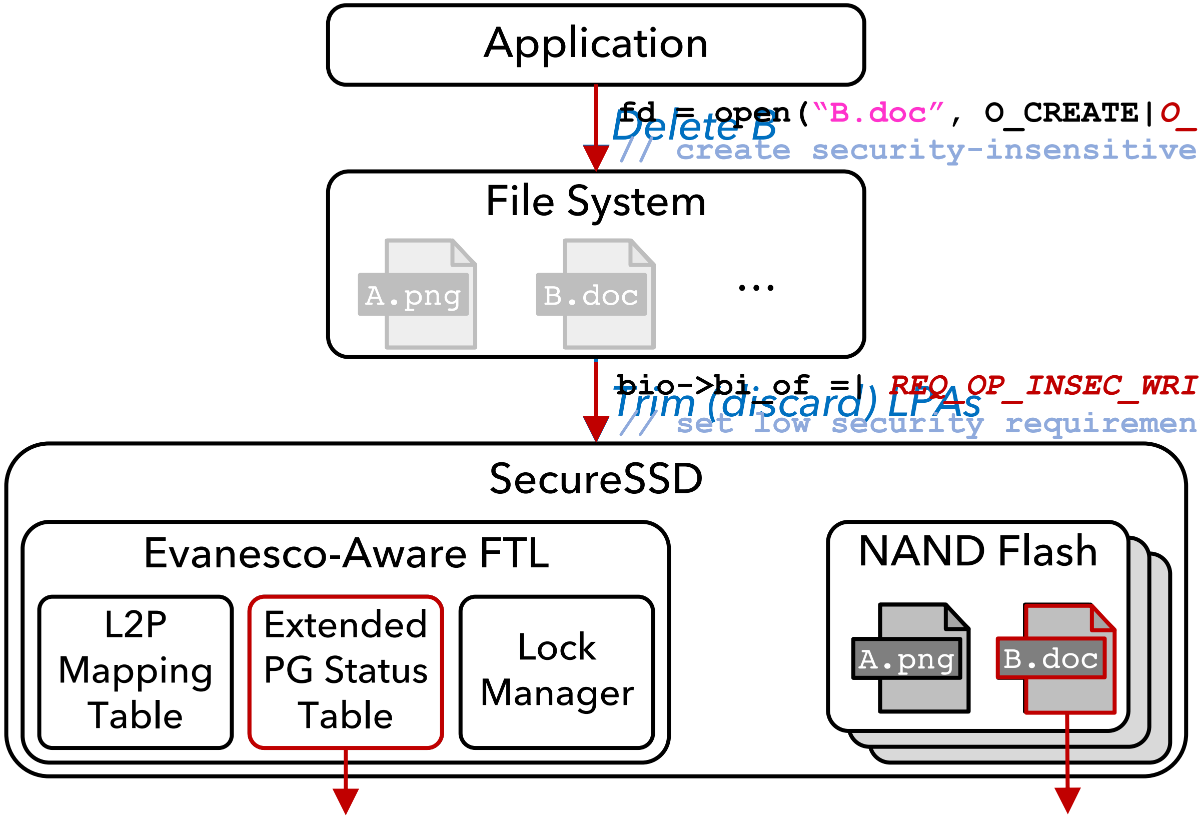
- To minimize the performance overhead of data sanitization
  - Issues pLock and bLock commands depending on the status of the target block



Multiple invalid pages + no valid pages in the block

SecureSSD: System-Level Optimization

- To minimize the performance overhead of data sanitization
  - Issues pLock and bLock commands depending on the status of the target block
  - Cross-layer interactions for selective data sanitization



```
fd = open("B.doc", O_CREATE|O_INSEC);
// create security-insensitive file B.doc
```

```
bio->bi_of =| REQ_OP_INSEC_WRITE;
// set low security requirement
```

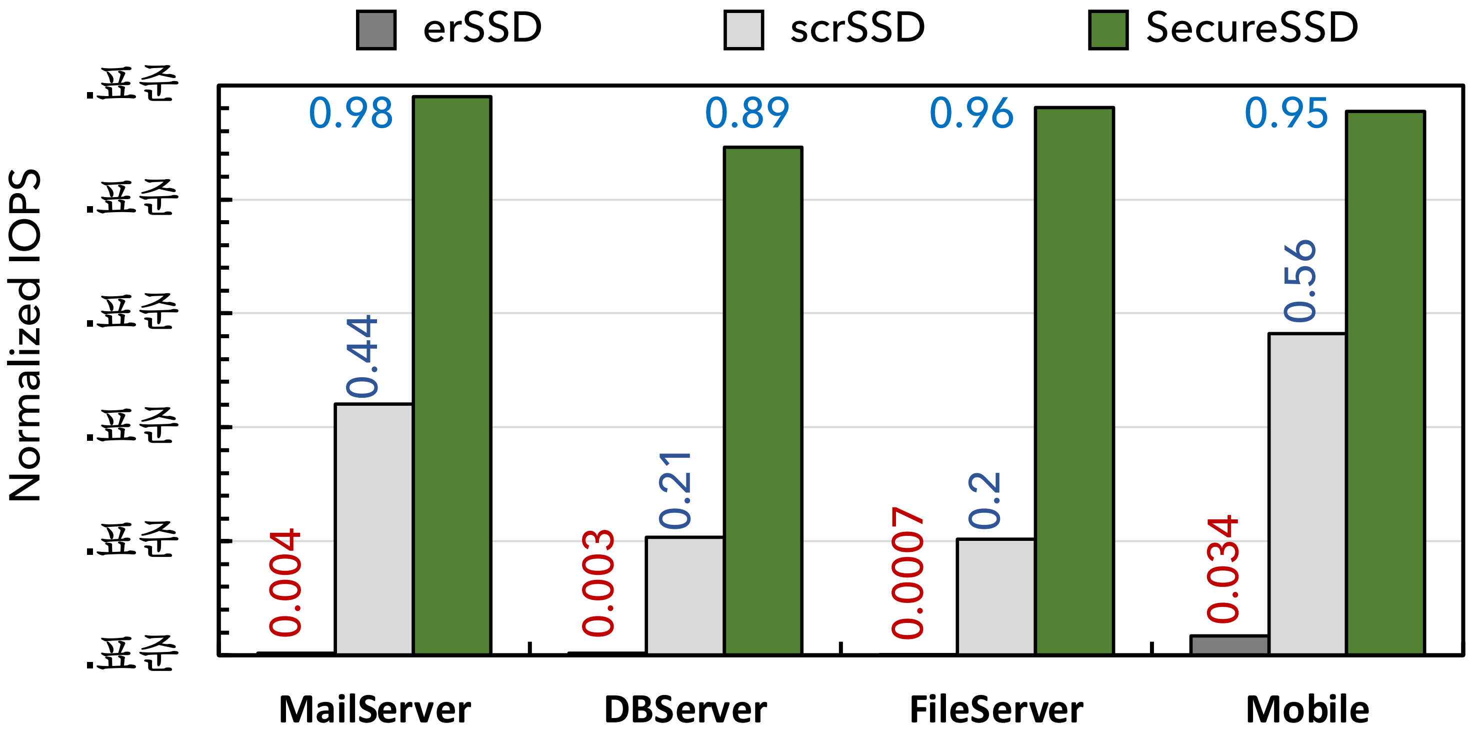
Set page status to `INSECURE`

Invalidation w/o `pLock` *or* `bLock`

Evanesco: Outline

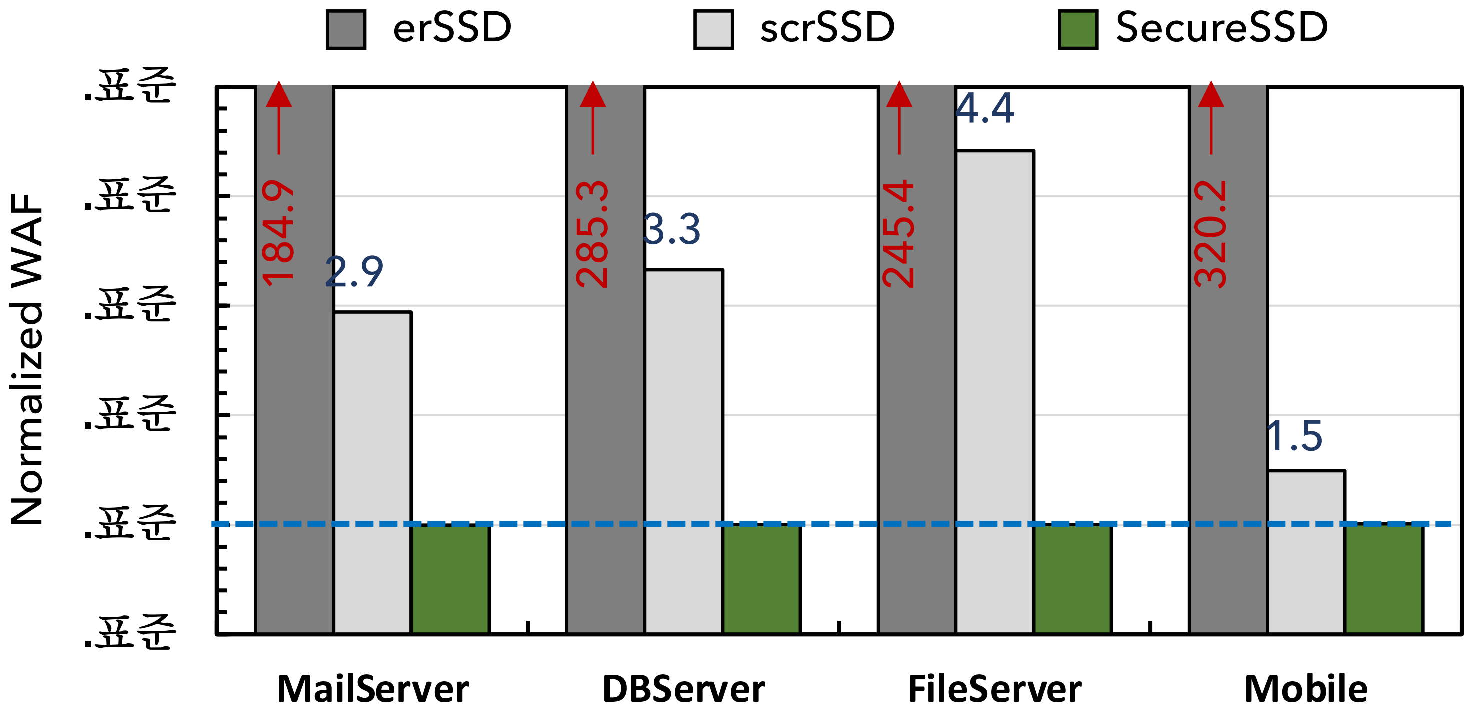
- Data Remanace in NAND Flash-Based SSDs
- Evanesco: Access Control-Based Sanitization
- Evaluation Results

Results: Performance



Evanesco significantly reduces performance overhead of data sanitization (11% slowdown at most)

Results: Lifetime



$$\textit{Write Amplification Factor } (WAF) = \frac{\#\ of\ physical\ pages\ written\ by\ the\ SSD}{\#\ of\ logical\ pages\ written\ by\ the\ host\ system}$$

No additional copy for data sanitization

→ No lifetime overhead

Summary of Contribution

- **Problem1:** Long, non-deterministic SSD read latency
  - Due to essential reliability management (read-retry)
  - Performance degradation of data-intensive applications

- **Our solution:** Pipelined & adaptive read-retry
  - Leveraging device characteristics and ECC margin
  - Reducing read-retry latency
    → easy to combine with other optimizations

- **Problem2:** Data remanence in NAND flash-based SSDs
  - Obsolete data remains intact in SSDs for an indefinite time
  - Physical data destrubtion: prohibitive performance overheads

- Our solution: Access control-based data sanitization
  - Avoids transfer of obsolete data from NAND flash chips
  - Minimal performance and reliability overheads

P&S Modern SSDs

Research Session 1:
Data Sanitization and Read-Retry
in Modern NAND Flash-Based SSDs

Dr. Jisung Park
Prof. Onur Mutlu
ETH Zürich

Spring 2022
15 July 2022