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 Diagram](image)

Unique characteristics
• Erase-before-write
• Limited endurance
• High error rates
• …

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

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

- **Retention loss**: A cell leaks electrons over time

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

![Diagram of Single-Level Cell (SLC) and 2-Bit Multi-Level Cell (MLC)]

- **Retention error!** at 1 year @ 1K P/E cycles
- **Retention error!** at 1 year @ 10K P/E cycles
Page and Block

• A number of flash cells operate in parallel

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

<table>
<thead>
<tr>
<th>WL0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>⋮</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WL1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⋮</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WL2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⋮</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WL125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⋮</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WL126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⋮</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WL127</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⋮</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Cells in the same block are concurrently erased

Block (128 WLs)

Page (16 KiB)

<table>
<thead>
<tr>
<th>0</th>
<th>Page (16 KiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td></td>
</tr>
</tbody>
</table>

Block (127 pages)
Page and Block

- A number of flash cells operate in parallel

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

Cells in the same block are concurrently erased

Share the same cells (WL)

<table>
<thead>
<tr>
<th>0</th>
<th>Page (16 KiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
</tr>
<tr>
<td>378</td>
<td></td>
</tr>
<tr>
<td>379</td>
<td></td>
</tr>
<tr>
<td>380</td>
<td></td>
</tr>
<tr>
<td>381</td>
<td></td>
</tr>
</tbody>
</table>

Triple-Level Cell (TLC)
Block (127 × 3 pages)
Pipelined & Adaptive Read-Retry

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

Jisung Park¹  Myungsuk Kim²,³  Myoungjun Chun²  Lois Orosa¹  Jihong Kim²  Onur Mutlu¹

¹ETH Zürich  Switzerland  ²Seoul National University  Republic of Korea ³Kyungpook 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)

Requested data

Expected latency: 100 µs
Actual latency: > 1 ms

NAND Flash-Based SSD

Degrades the quality of service of read-intensive, latency-sensitive applications
Problem: Long, Non-Deterministic Latency

Requested data
Expected latency: 100 $\mu$s
Actual latency: > 1 ms

Internal Read-Retry Operations
Errors in NAND Flash Memory

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

![Diagram showing $V_{REF}$ and $V_{TH}$ levels for erased and programmed states]

- $V_{REF}$ distribution for an SLC page

1. **Erased State**
   - $V_{TH}$ levels are below $V_{REF}$
   - Distribution is typical of an erased state

2. **Programmed State**
   - $V_{TH}$ levels are above $V_{REF}$
   - Distribution is typical of a programmed state
Errors in NAND Flash Memory

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

![Diagram showing $V_{TH}$ level distributions for erased and programmed states with interference and retention loss effects.](image-url)
Error-Correcting Codes (ECC)

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

![Diagram of NAND Flash Chip and Flash Controller with ECC Engine]
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

### SLC Page Distribution

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

### TLC Page Distribution

<table>
<thead>
<tr>
<th># of cells</th>
<th>VREF0</th>
<th>VREF1</th>
<th>VREF2</th>
<th>VREF3</th>
<th>VREF4</th>
<th>VREF5</th>
<th>VREF6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>E</td>
<td>111</td>
<td>110</td>
<td>100</td>
<td>000</td>
<td>010</td>
<td>011</td>
</tr>
<tr>
<td>LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$V_{TH}$ Margin

$V_{REF}$
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?
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

Erroneous cells

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

\[ N_{\text{ERR}} = 32 \ < \ \text{ECC capability} \ C_{\text{ECC}} = 72 \]
Read-Retry: Performance Overhead

Read-retry increases the read latency almost linearly with the number of retry steps.
P&AR²: Outline

• Read-Retry in Modern NAND Flash-Based SSDs

• PR²: Pipelined Read-Retry

• AR²: Adaptive Read-Retry

• Evaluation Results
Pipelined Read-Retry (PR²): Key Idea

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**

- **tR**
- **tECC**
- **tDMA**

**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**

**Speculative start**

**Latency Reduction**

\[
N_{ERR} = N \times (tR + tDMA + tECC)
\]

\[
N \times (tR + tDMA + tECC)
\]

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 (AR²): Key Idea

\[ t_{R} + t_{ECC} \]

\[ N_{ERR} = 232 > C_{ECC} = 72 \]

\[ N_{ERR} = 173 \]

\[ N_{ERR} = 118 \]

\[ N_{ERR} = 87 \]

\[ N_{ERR} = 23 \]

\[ t_{READ} = t_{RETRY} = N \times (t_{R} + t_{DMA} + t_{ECC}) \]
Adaptive Read-Retry (AR²): 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 (AR²): Key Idea

- $t_{READ} = N \times t_R + t_{DMA} + t_{ECC}$

Trading the large ECC margin to reduce $t_R$
Adaptive Read-Retry (AR²): Key Idea

Trading the large ECC margin to reduce $t_R$

$→$ Further reduction in the read-retry latency
Adaptive Read-Retry (AR²): Key Idea

\[ N_{ERR} = 232 > ECC \text{ capability } C_{ECC} = 72 \]

\[ N_{ERR} = 173 + e_1 \]

\[ N_{ERR} = 118 + e_2 \]

\[ N_{ERR} = 87 + e_{N-1} \]

\[ N_{ERR} = 23 + e_N \]

**AR² reduces t_R in every retry step**
Adaptive Read-Retry (AR$^2$): Key Idea

AR$^2$ reduces $t_R$ 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 $V_{\text{REF}}$ in the final retry step

• Observation 2: A large reliability margin incorporated in read-timing parameters
  ◦ 25% $t_R$ reduction $\rightarrow$ 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% $t_R$ reduction
### P&AR²: Design

#### SSD Firmware (FTL)

<table>
<thead>
<tr>
<th>P/E</th>
<th>$t_{\text{RET}}$ [days]</th>
<th>$t_R$ [μs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 250</td>
<td>&lt; 60</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 360</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 1.5K</td>
<td>&lt; 60</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 360</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

#### Read-timing Parameter Table (Pre-profiled)

- **Read A**
- **Flash Controller**
- **ECC Engine**
- **NAND Flash Chip**

- **Data A**
- **Fail**

- **ECC fail**

- **tR**
- **tDMA**
- **tECC**
## P&AR²: Design

### SSD Firmware (FTL)

<table>
<thead>
<tr>
<th>P/E</th>
<th>(t_{\text{RET}}) [days]</th>
<th>(t_R) [μs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 250</td>
<td>&lt; 60</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; 360</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; 360</td>
<td>75</td>
</tr>
<tr>
<td>&lt; 1.5K</td>
<td>&lt; 60</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Read-timing Parameter Table (Pre-profiled)**

- **Best \(t_R\) for the current operating conditions**

![Diagram showing SSD Firmware (FTL) components and read-timing parameter table](image)

- **Flash Controller**
- **ECC Engine**

- **NAND Flash Chip**

- **ECC fail**

- **Best \(t_R\)**

- **Fail**

- **tR**, **tDMA**, **tECC**
P&AR²: Design

SSD Firmware (FTL)

<table>
<thead>
<tr>
<th>P/E</th>
<th>( t_{RET} ) [days]</th>
<th>( t_R ) [( \mu s )]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 250</td>
<td>&lt; 60</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 360</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 360</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 1.5K</td>
<td>&lt; 60</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Read-timing Parameter Table (Pre-profiled)

1. **Best \( t_R \) for the current operating conditions**

Flash Controller

ECC Engine

NAND Flash Chip

Page A

1. **Best \( t_R \)**

2. **Read-retry w/ best \( t_R \) + speculative start**

ECC fail

ECC success

\[ t_R \quad t_{DMA} \quad t_{ECC} \quad \text{Best } t_R \]
P&AR²: Design

SSD Firmware (FTL)

<table>
<thead>
<tr>
<th>PEC</th>
<th>( t_{\text{RET}} ) [days]</th>
<th>( t_R ) [( \mu \text{s} )]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 60</td>
<td>65</td>
<td></td>
</tr>
</tbody>
</table>

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

NAND Flash Chip

Read-retry w/ best \( t_R \) + speculative start
P&AR²: Outline

• Read-Retry in Modern NAND Flash-Based SSDs

• PR²: Pipelined Read-Retry

• AR²: 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

File System

Update

No overwrite → Out-of-place update

NAND Flash-Based SSD
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
4. Run forensic tools
Existing Solutions

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

![Diagram showing target block, target page, free block, and copy of valid pages with timing calculation for copy operation.](image)
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

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]
  - **Problem 1**: MLC NAND flash stores multiple pages in a WL

---

**Target Block**

- Free
- Valid
- Invalid
- Scrubbed

**Target Page**

<table>
<thead>
<tr>
<th>Page 0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**WL0**

<table>
<thead>
<tr>
<th>WL0</th>
<th>V_{REF0}</th>
<th>V_{REF1}</th>
<th>V_{REF2}</th>
<th>V_{REF3}</th>
<th>V_{REF4}</th>
<th>V_{REF5}</th>
<th>V_{REF6}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Page 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Page 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**V_{TH}**

<table>
<thead>
<tr>
<th># of cells</th>
<th>CSB</th>
<th>MSB</th>
<th>E</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td></td>
<td></td>
<td></td>
<td>110</td>
<td>110</td>
<td>100</td>
<td>010</td>
<td>010</td>
<td>001</td>
<td>011</td>
</tr>
<tr>
<td>Page 1</td>
<td></td>
<td></td>
<td></td>
<td>000</td>
<td>111</td>
<td>110</td>
<td>000</td>
<td>010</td>
<td>001</td>
<td>111</td>
</tr>
<tr>
<td>Page 2</td>
<td></td>
<td></td>
<td></td>
<td>111</td>
<td>000</td>
<td>000</td>
<td>111</td>
<td>001</td>
<td>101</td>
<td>101</td>
</tr>
</tbody>
</table>
Existing Solutions

- **Scrubbing** [Wei+, FAST’11]
  - Problem 1: MLC NAND flash stores multiple pages in a WL

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
Existing Solutions

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

• Scrubbing [Wei+, FAST’11]
  ◦ Problem 1: MLC NAND flash stores multiple pages in a WL
  ◦ Problem 2: Program interference
Existing Solutions

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

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`

![Diagram showing the use of `pLock(3)` and `bLock(0)` commands with NAND Flash chip access and flags]

- `read(2)` results in all-zero data
- Flags indicate valid data and program interference
**Evanesco: Requirements**

- Keep access-permission flags in a non-volatile manner
- Access-control logic inside a NAND flash chip
- Minimal area overhead $\Rightarrow$ High chip density is paramount

<table>
<thead>
<tr>
<th>Block#0</th>
<th>Flags#0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>D</td>
</tr>
<tr>
<td>A1</td>
<td>D</td>
</tr>
<tr>
<td>A2</td>
<td>D</td>
</tr>
<tr>
<td>B0</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block#1</th>
<th>Flags#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0’</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>E</td>
</tr>
</tbody>
</table>

NAND Flash

- `pLock(3)`
- `bLock(0)`

Read(2)

`00...0` (all-zero data)
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

\[ p\text{Lock}(\text{page0}) \]
pLock: Page-Level Data Sanitization

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

\[ \text{pLock}\left(\text{page0}\right) \]

- WL\(_0\):
  - **Page Data Area**
  - **Spare Area**
  - No program of data cells $\Rightarrow$ little interference

- WL\(_1\):
  - **Page Data Area**
  - **Spare Area**
  - Interference b/w flag cells

**SLC $\Rightarrow$ 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-permission flags: **Spare cells in each WL**
- On-chip access control logic: **Small changes to data path**

```
READ(page0)

Page Data Area
WL0
WL1
WL_{N-1}

Spare Area

Bitlines (BLs) 0 1 0 0 1 1 0 1 0 1 1

Page Buffer Data Out

Erased = Enabled Programmed = Disabled
```
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**

**READ(page0)**

- **Page Data Area**
  - WL0
  - WL1
  - WL_{N-1}
- **Spare Area**
  - Erased = Enabled
  - Programmed = Disabled

**Bitlines (BLs)**

```plaintext
0 1 0 0 1 1 0 1
```

**Page Buffer**

**MUX**

- 0 (disconnect)

**Data Out**

```plaintext
00...0
```
Real-Device Characterization

• Using 160 real 48-layer TLC NAND flash chips

• No reliability degradation for stored data

• $t_{\text{PLOCK}} = 100 \, \mu\text{s}, \ t_{\text{BLOCK}} = 300 \, \mu\text{s}$

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

![Diagram showing SecureSSD architecture](image)

- Application
  - Delete A
  - Trim (discard) LPAs

- File System
  - A.png
  - B.doc

- SecureSSD
  - Evanesco-Aware FTL
    - L2P Mapping Table
    - Page Status Table
  - Lock Manager
  - NAND Flash
    - A.png
    - B.doc

- NAND Flash
  - Block #k
    - Valid
    - Invalid

- Multiple invalid pages + no valid pages in the block

- Block #m
  - pLock
  - 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

```c
fd = open("B.doc", O_CREAT | 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
Evanesco significantly reduces performance overhead of data sanitization (11% slowdown at most)
Results: Lifetime

**Write Amplification Factor (WAF)**

\[
WAF = \frac{\text{# of physical pages written by the SSD}}{\text{# of logical pages written by the host system}}
\]

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 destruction: 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