P&S Modern SSDs

Garbage Collection

Dr. Mohammad Sadrosadati
Prof. Onur Mutlu

ETH Zürich
Spring 2023
19 May 2023
Recap: SSD & NAND Flash Memory

- SSD organization
  - SSD controller: Multicore CPU + per-channel flash controllers
  - DRAM: Metadata store, 0.1% of SSD capacity
  - NAND flash chips
    - Channel (Package(s)) – Die – Plane – Block – Page

- NAND flash characteristics
  - Erase-before-write, asymmetry in operation units (read/program: page, erase: block), limited endurance, retention loss...

- Basic NAND flash operations
  - Read/program/erase
Recap: Advanced Commands

- **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
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

NAND Flash Chip (Single Plane)

Simple SSD Architecture

Logical Block Address

LBA

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Host

SSD

Flash Translation Layer

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

NAND Flash Chip (Single Plane)
Simple SSD Architecture

Logical Block Address

LBA 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Host

SSD

Flash Translation Layer

Overprovisioning:
- Physical capacity > Logical capacity
- For performance & lifetime

PPA

Physical Block Address

PBA

Physical Page Address

NAND Flash Chip (Single Plane)

Block0

0

Page

Block1

4

8

12

16

Block2

5

9

13

17

Block3

6

10

14

18

Block4

7

11

15

19
Write Request Handling: Page Write

Flash Translation Layer

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

Flash Translation Layer

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

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

Host

SSD

Flash Translation Layer

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

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

**Note:**

- We are assuming that logical block size = physical page size
- LB size = 4 KiB, PP size = 16 KiB
Write Request Handling: Sequential Write
Write Request Handling: Sequential Write

Flash Translation Layer

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

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

**Flash Translation Layer**

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

PROG(PPA: 1, B)

PROG(PPA: 2, C)

: 12 page-program commands

PROG(PPA: 12, M)

**Sequential (large) write**

Host

SSD

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

- **Active block** (or write-point) approach
  - Keep only one block being written
  - Due to the open-block problem

- **Program-sequence** constraint
  - Fixed program order within a block
  - Due to cell-to-cell interference
Write Request Handling: Address Mapping

### Flash Translation Layer

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

<table>
<thead>
<tr>
<th>Host</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E F G H I J K L M</td>
<td>A B C D E F G H I J K L M</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Block0</th>
<th>Block1</th>
<th>Block2</th>
<th>Block3</th>
<th>Block4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>J</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>K</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

Problem: LBA (or LPA) does not match PPA!
Req (LBA: 4, Size: 1, DIR: R)

Needs to maintain Address-mapping information

Flash Translation Layer

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

Host

SSD

Flash Translation Layer

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

READ (PPA: ?)

Mapping Table

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

Host

SSD

Flash Translation Layer

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

READ (PPA: 1)

Mapping Table

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

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

Flash Translation Layer

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

Mapping Table

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

Host SSD

Flash Translation Layer

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

Mapping Table

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

Flash Translation Layer

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

PROG (PPA: 13, A')

Mapping Table

Block0 | Block1 | Block2 | Block3 | Block4
---|---|---|---|---
0 | A | E | I | M | 16
1 | B | F | J | 13 | 17
2 | C | G | K | A' | 18
3 | D | H | L | | 19
Write Request Handling: Update

**Flash Translation Layer**

- **Req** (LBA: 0, Size: 1, DIR: W, A')
  - **PROG** (PPA: 13, A')

**Mapping Table**

- **LPA** | **PPA**
  - 0 | 13
  - ... | ...
  - 4 | 1
  - 5 | 2
  - ... | ...

**NAND Flash Chip (Single Plane)**

- **Invalid**

- **Host**
  - SSD

- **Block0**
  - 0: A
  - 1: B
  - 2: C
  - 3: D

- **Block1**
  - 4: E
  - 5: F
  - 6: G
  - 7: H

- **Block2**
  - 8: I
  - 9: J
  - 10: K
  - 11: L

- **Block3**
  - 12: M
  - 13: A'

- **Block4**
  - 16: 
  - 17: 
  - 18: 
  - 19: 

- **Update Mapping**
Write Request Handling: Update

Flash Translation Layer

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

PROG (PPA: 14, A')

Mapping Table

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Host

SSD

Flash Translation Layer

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

PROG (PPA: 15, A')

Mapping Table

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

**Flash Translation Layer**

**Req** \(\text{LBA: 0, Size: 1, DIR: W, A'}\)

**PROG** \(\text{PPA: 16, A'}\)

**Mapping Table**

**NAND Flash Chip (Single Plane)**

Running out of free pages
Garbage Collection

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

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

Flash Translation Layer

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

Mapping Table

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

Host

SSD

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

Flash Translation Layer

Host

SSD

Mapping Table

Status Table

F: free, V: valid, I: invalid

READ (PPA: 12)

PROG (PPA: 17, M)

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

Flash Translation Layer

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

Mapping Table

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

Host SSD

READ (PPA: 12)

PROG (PPA: 17, M)

NAND Flash Chip (Single Plane)

Update Status

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

Flash Translation Layer

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

F: free, V: valid, I: invalid

Update Status

Update Mapping

Mapping Table

Status Table

NAND Flash Chip (Single Plane)

READ (PPA: 12)

PROG (PPA: 17, M)
Write Request Handling: Garbage Collection

Flash Translation Layer

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

PBA Status

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

**Update Status**

**Update Mapping**

**NAND Flash Chip (Single Plane)**

**Host SSD**

**Mapping Table**

**Status Table**

**Flash Translation Layer**

**READ (PPA: 12)**

**PROG (PPA: 17, M)**
Write Request Handling: Garbage Collection

Flash Translation Layer

- LPA: Location Pointer Address
- PPA: Page Pointer Address

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

Mapping Table

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

Status Table

NAND Flash Chip (Single Plane)

Host

SSD

READ (PPA: 12)

PROG (PPA: 17, M)

BERS (PBA: 3)
Write Request Handling: Garbage Collection

Flash Translation Layer

F: free, V: valid, I: invalid

Mapping Table

Status Table

Update Status

NAND Flash Chip (Single Plane)
**Note:**

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

(PPA: 12)

PROG (PPA: 17, M)

BERS (PBA: 3)
Performance Issues

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

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

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

- **Progressive GC**: Divide GC process into subtasks
  - e.g., copying 28 pages → (copying 1 page + servicing user request) × 28
  - Effective at decreasing tail latency
Fine-Grained Mapping
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 \( \text{LBA: 0x04, Size: 1, DIR: w, Data: A} \)

16-KiB Page Number \quad 4\text{-KiB Offset}

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

Block 0

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

Block 1

16 KiB
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

16 KiB

Block 0

<table>
<thead>
<tr>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
</tr>
<tr>
<td>0x01</td>
</tr>
<tr>
<td>0x02</td>
</tr>
<tr>
<td>0x03</td>
</tr>
<tr>
<td>0x04</td>
</tr>
<tr>
<td>0x05</td>
</tr>
<tr>
<td>0x06</td>
</tr>
</tbody>
</table>

Block 1

A
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

\[\begin{array}{cccc}
0b & 0000 & 0000 & 0000 & 0100 \\
\hline
16-KiB Page Number & 4-KiB Offset
\end{array}\]

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

16 KiB

Block 0

\[\begin{array}{cccc}
0x00 & A & 0x01 & 0x02 \\
0x03 & 0x04 & 0x05 & 0x06 \\
\end{array}\]

Block 1
Small Write Requests

- Inefficiencies due to the erase-before-write property

\[ \text{Req} (\text{LBA: } 0\times01, \text{ Size: } 2, \text{ DIR: w}, \text{ Data: B, C}) \]

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

16-KiB Page Number 4-KiB Offset

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

Block 0

Block 1

PPA

0x00
0x01
0x02
0x03
0x04
0x05
0x06
...
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

Block 0

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

Block 1

B

C
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

Block 0

Block 1

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

A

B

C
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

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

- Inefficiencies due to the erase-before-write property

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

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

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

Q: Can we use the unused space?
A: Not likely, because
- Data randomization – Cells in the unused space have been already programmed.
- Program-order constraint – Re-programming physical page 0x00 can affect the reliability of the data stored in physical page 0x01.
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

Block 0

PPA

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

Block 1

- Unused yet discarded

A

B

C
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

- **Block 0**
  - PPA: 0x00
  - 0x01
  - 0x02
  - 0x03
  - 0x04
  - 0x05
  - 0x06

- **Block 1**
  - PPA: 0x00
  - 0x01
  - 0x02
  - 0x03
  - 0x04
  - 0x05
  - 0x06

*Read*
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

Block 0

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

Block 1

- Modify
- Read
Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

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

- Write a page only when there are sufficient data blocks

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

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

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

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

**Page Buffer**

<table>
<thead>
<tr>
<th>Block 0</th>
<th>Block 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 0x01 0x02 0x03</td>
<td>0x10 0x11 0x12 0x13</td>
</tr>
<tr>
<td>0x04 0x05 0x06 0x07</td>
<td>0x14 0x15 0x16 0x17</td>
</tr>
<tr>
<td>0x08 0x09 0x0A 0x0B</td>
<td>0x18 0x19 0x1A 0x1B</td>
</tr>
<tr>
<td>0x0C 0x0D 0x0E 0x0F</td>
<td></td>
</tr>
<tr>
<td>0x14 0x15 0x16 0x17</td>
<td></td>
</tr>
</tbody>
</table>
### Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

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

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

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

---

#### Page Buffer

- A

---

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

---

**Block 0**

- \(0x00\) \(0x01\) \(0x02\) \(0x03\)
- \(0x04\) \(0x05\) \(0x06\) \(0x07\)
- \(0x08\) \(0x09\) \(0xA\) \(0xB\)
- \(0xC\) \(0xD\) \(0xE\) \(0xF\)
- \(0x10\) \(0x11\) \(0x12\) \(0x13\)
- \(0x14\) \(0x15\) \(0x16\) \(0x17\)
- \(0x18\) \(0x19\) \(0xA\) \(0xB\)
- ...
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

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

- Write a page only when there are sufficient data blocks

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

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

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

- Write a page only when there are sufficient data blocks

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

- Write a page only when there are sufficient data blocks

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

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

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

- Write a page only when there are sufficient data blocks

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

- Write a page only when there are sufficient data blocks

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

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

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

- Write a page only when there are sufficient data blocks

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

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

Page Buffer

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

Block 0

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

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)

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

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

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

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

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

Despites non-negligible drawbacks, fine-grained mapping is widely used in modern SSDs due to its high benefits.
Multi-plane Operation-Aware Block Management
Concurrent operations on different planes

- Recall: Planes share WLs and row/column decoders

- Opportunity: Planes can **concurrently** operate

- Constraints: Only for the **same operations on the same page offset**

Recap: Multi-Plane Operations
To perform as many multi-plane operations as possible

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

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

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

![Diagram of page buffer and planes]
Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).
Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).
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

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

2. **Valid page copy**

Page Status:
- Free
- Valid
- Invalid
Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.

Multi-Plane-Aware Block Management

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

Cannot perform multi-plane writes before using (or discarding) 2nd and 3rd pages of Block_{N-1} in Plane_1

4-page copy: can be done with
2 × multi-plane reads &
2 × multi-plane writes
→ 4 × single-plane reads &
1 × multi-plane write &
2 × single-plane writes
Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes.
**Multi-Plane-Aware Block Management**

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

![Diagram showing multi-plane-aware block management](image)

- **Valid page reads**: (5 SRs + 1 MR)
**Multi-Plane-Aware Block Management**

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

---

![Diagram showing multi-plane block management](image)

- **Valid page reads**: $(5 \text{ SRs} + 1 \text{ MR})$
- **Valid page writes**: $(3 \text{ MRs})$
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
  - $5 \text{ SRs} + 1 \text{ MR} + 3 \text{ MWs}$
  - vs. $4 \text{ SRs} + 1 \text{ MW} + 2 \text{ SWs}$

---

1. **Valid page reads**
   - $(5 \text{ SRs} + 1 \text{ MR})$
   - **Buffering**

2. **Valid page writes**
   - $(3 \text{ MWs})$
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.
P&S Modern SSDs

Garbage Collection

Dr. Mohammad Sadrosadati
Prof. Onur Mutlu

ETH Zürich
Spring 2023
19 May 2023