P&S Modern SSDs
Advanced NAND Flash Commands & Address Translation

Dr. Mohammad Sadrosadati
Prof. Onur Mutlu
ETH Zürich
Spring 2023
31 March 2023
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
Today’s Agenda

- Advanced NAND Flash Commands
- Address Translation & Garbage Collection
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
  - $t_R$: Latency of reading (sensing) data from the cells into the on-chip page buffer
  - $t_{PROG}$: Latency of programming the cells with data in the page buffer
  - $t_{BERS}$: Latency of erasing the cells (block)
  - Varies depending on the MLC technology, processing node, and microarchitecture
    - In 3D TLC NAND flash, $t_R/t_{PROG}/t_{BERS} \approx 100\text{us}/700\text{us}/3\text{ms}$

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

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

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

```
Flash Controller

1. NAND Flash Chip
2. Sensing
3. Transfer
4. ECC
5. RAND

READ(x)
```
NAND Flash Chip Performance (Cont.)

- **tR, tPROG, and tBERS**
  - Latencies for chip-level read/program/erase operations
  - \( tR: 50 \sim 100 \text{ us} \)
  - \( tPROG: 700\text{us} \sim 1000 \text{ us} \)
  - \( tBERS: 3\text{ms} \sim 5\text{ms} \)

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

- **Read**
  - 16 KiB / 136 us \(\approx\) 120 MB/s

- **Write**
  - 16 KiB / 736 us \(\approx\) 22 MB/s

**WAIT!**

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

Optimizations w/ advanced commands

---

**DRAM/SLC Write Buffer**

**Internal parallelism**

- **Flash Controller**
  - ECC
  - RAND

- **NAND Flash Chip**

...
Advanced Commands for Small Reads

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

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

- Optimization 2: Random Data Out (RDO)
  - Data transfer with an arbitrary offset and size
  - Reduce tDMA and tECC_{DEC}

---

**CACHE READ Command**

- Performs consecutive reads in a pipelined manner

Regular PAGE READ:
Overlaps only tECC with tR

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

- Needs additional on-chip page buffer

1. PAGE READ (A)
2. Page sensing
3. CACHE READ (B)
4. Page sensing
5. DATA OUT (A)

NAND Flash Plane

Page A
Page B
...

Page Buffer 1
Page Buffer 2
CACHE READ Command: Benefit

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

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

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

![Diagram showing multiple planes with shared decoders and buffers]

**Same voltage can be applied to all cells**
Multi-Plane Operations: Benefit

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

- Per-operation latency increases
  - Regular page program: tECC_{ENC} + tDMA + tPROG
  - Multi-plane page program: \( N_{\text{Plane}} \times (tECC_{\text{ENC}} + tDMA) + tPROG \)

- 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

  100 us: Read arrival


- 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

100 us: Read arrival

Latency: 700 us

Latency: 700 us

Suspend

Latency: ~700 us

Latency: ~800 us
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
Today’s Agenda

- Advanced NAND Flash Commands
- Address Translation & Garbage Collection
Flash Translation Layer: Overview

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

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

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

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

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

 Host

 SSD

 Flash Translation Layer

 NAND Flash Chip (Single Plane)

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

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

Flash Translation Layer

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

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

**Flash Translation Layer**

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

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

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

NAND Flash Chip (Single Plane)

Flash Translation Layer

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

**Flash Translation Layer**

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

**Sequential (large) write**

#### NAND Flash Chip (Single Plane)

<table>
<thead>
<tr>
<th>Block0</th>
<th>Block1</th>
<th>Block2</th>
<th>Block3</th>
<th>Block4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4 A</td>
<td>4 8</td>
<td>8 12</td>
<td>12 16</td>
<td>16 19</td>
</tr>
<tr>
<td>1 5</td>
<td>5 9</td>
<td>9 13</td>
<td>13 17</td>
<td>17</td>
</tr>
<tr>
<td>2 6</td>
<td>6 10</td>
<td>10 14</td>
<td>14 18</td>
<td>18</td>
</tr>
<tr>
<td>3 7</td>
<td>7 11</td>
<td>11 15</td>
<td>15 19</td>
<td>19</td>
</tr>
</tbody>
</table>
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)
- **PROG**(PPA: 12, M)

12 page-program commands

**NAND Flash Chip (Single Plane)**

<table>
<thead>
<tr>
<th>Host</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Block0</td>
<td>Block1</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>

**Sequential (large) write**
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

```
<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>B</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>5</td>
<td>C</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>6</td>
<td>D</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>7</td>
<td>E</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>G</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td></td>
</tr>
</tbody>
</table>
```

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

Flash Translation Layer

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

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

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

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

READ (PPA: ?)

Needs to maintain Address-mapping information

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

**Flash Translation Layer**

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

**READ (PPA: ?)**

**Mapping Table**

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

**NAND Flash Chip (Single Plane)**

<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>5</td>
<td>9</td>
<td>J</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
<td>K</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>11</td>
<td>L</td>
<td>15</td>
</tr>
</tbody>
</table>
Write Request Handling: Address Mapping

Flash Translation Layer

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

READ (PPA: 1)

Mapping Table

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

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

Mapping Table

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

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

**Flash Translation Layer**

| Req (LBA: 0, Size: 1, DIR: W, A') | \[\begin{array}{|c|c|} \hline
LPA & PPA \\
0 & 0 \\
\vdots & \vdots \\
4 & 1 \\
5 & 2 \\
\vdots & \vdots \\
\hline
\end{array}\] |

**Mapping Table**

**NAND Flash Chip (Single Plane)**

**Host**

**SSD**
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)**

```
Block0  Block1  Block2  Block3  Block4
0       A       E       I       M       16
1       B       F       J       13
2       C       G       K       14
3       D       H       L       15
```

*Invalid*
Write Request Handling: Update

**Flash Translation Layer**

**Req** (LBA: 0, Size: 1, DIR: W, A’)

**PROG** (PPA: 14, A’)

**Mapping Table**

**NAND Flash Chip (Single Plane)**

<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>8</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LPA</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
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** (LBA: 0, Size: 1, DIR: W, A')
- **PROG** (PPA: 16, A')

**Mapping Table**

<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>

**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>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>VFFFF</td>
</tr>
</tbody>
</table>

Mapping Table

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

NAND Flash Chip (Single Plane)

READ (PPA: 12)
Write Request Handling: Garbage Collection

- **Flash Translation Layer**
  - **LPA**
  - **PPA**
  - **Status Table**
    - F: free, V: valid, I: invalid
  - **Mapping Table**

- **NAND Flash Chip (Single Plane)**
  - **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'
    - 14: A'
    - 15: A'
  - **Block4**
    - 16: A'
    - 17: M
    - 18: 
    - 19: 

**Host**

- **SSD**

**Flash Translation Layer**

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

**Flash Translation Layer**

- **LPA**
- **PPA**
  - 0: 16
  - ...: ...
  - 4: 1
  - 5: 2
  - ...: ...

**Mapping Table**

- **PBA**
- **Status**
  - 0: IVVV
  - 1: VVVV
  - 2: VVVV
  - 3: IIII
  - 4: VVFF

**NAND Flash Chip (Single Plane)**

**Host**

**SSD**

- **READ (PPA: 12)**
- **PROG (PPA: 17, M)**

**Update Status**
Write Request Handling: Garbage Collection

Host

SSD

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>12</td>
</tr>
</tbody>
</table>

Mapping Table

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

NAND Flash Chip (Single Plane)

F: free, V: valid, I: invalid

READ (PPA: 12)

PROG (PPA: 17, M)

Update Mapping

Update Status
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>

**Status Table**

- PBA: 0 IVVVV
- PBA: 1 VVVV
- PBA: 2 VVVV
- PBA: 3 IIIII
- PBA: 4 VVFF

**Mapping Table**

- Update Mapping
- Update Status

NAND Flash Chip (Single Plane)

**Host SSD**

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

• **Q:** How FTL knows PPA 12 (data M) is mapped to LPA 15?
  o 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

Host

SSD

Flash Translation Layer

Mapping Table

Status Table

F: free, V: valid, I: invalid

LPA | PPA
---|---
0  | 16
4  | 1
15 | 17

PBA | Status
---|---
0  | IVVVV
1  | VVVVV
2  | VVVVV
3  | IIIII
4  | VVFF

Block0 | Block1 | Block2 | Block3 | Block4
---|---|---|---|---
A | E | I |  | A’
B | F | J |  | M
C | G | K |  | 
D | H | L |  | 

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

Flash Translation Layer

F: free, V: valid, I: invalid

LPA | PPA | Status Table
---|---|---
0  | 16  | IVVVV
... | ... | VVVVV
4  | 1   | VVVVV
... | ... | FFFFF
15 | 17  | VVFFF

Update Status

Mapping Table

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

- If FTL performs GC in an atomic manner, it delays user requests for a significantly long time
  - Long tail latency (performance fluctuation)
  - Noisy neighbor: a read-dominant workload’s performance would be significantly affected when running with a write-intensive workload (+ performance fairness problem)
Performance Issues: Mitigation

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

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

- **Progressive GC:** Divide GC process into subtasks
  - e.g., copying 28 pages $\rightarrow$ (copying 1 page + servicing user request) $\times$ 28
  - Effective at decreasing tail latency
Required Materials

- Address Mapping
Recommend Materials

- **Cache read & Read-retry**
  - Jisung Park, Myungsuk Kim, Lois Orosa, Jihong Kim, and Onur Mutlu, “Reducing Solid-State Drive Read Latency by Optimizing Read-Retry,” In ASPLOS 2021.

- **Program & Erase Suspension**
  - Guanying Wu and Xunbin He, “Reducing SSD Read Latency via NAND Flash Program and Erase Suspension,” In USENIX FAST 2012.
P&S Modern SSDs

Advanced NAND Flash Commands & Address Translation

Dr. Mohammad Sadrosadati
Prof. Onur Mutlu

ETH Zürich
Spring 2023
31 March 2023