P&S Modern SSDs

Fine-Grained Mapping &
Multi-Plane Operation-Aware Block Management

Dr. Jisung Park

Prof. Onur Mutlu

ETH Zürich

Spring 2022

15 April 2022

Recap: What We Have Discussed So Far

- **SSD Organization**

- **NAND Flash Organization and Operations**

- **Advanced NAND Flash Commands**

- **FTL: Address Translation & Garbage Collection**

Today's Agenda

- **Fine-Grained Mapping**

- Multi-plane Operation-Aware Blk. Mgmt.

I/O Mismatch b/w OS and NAND Flash

- The page size (i.e., minimum I/O unit) of NAND flash memory has continuously increased.
    - From 256 bytes to 16 KiB
    - Low area overhead and high bandwidth (size / latency)

- The logical block (or sector) size of file systems has also increased.
    - From 512 bytes to 4 KiB
    - To more efficiently work with NAND flash-based SSDs
    - Increasing the block size is not straightforward.
        - I/O handling is closely related OS memory management.
        - Memory page size = 4 KiB
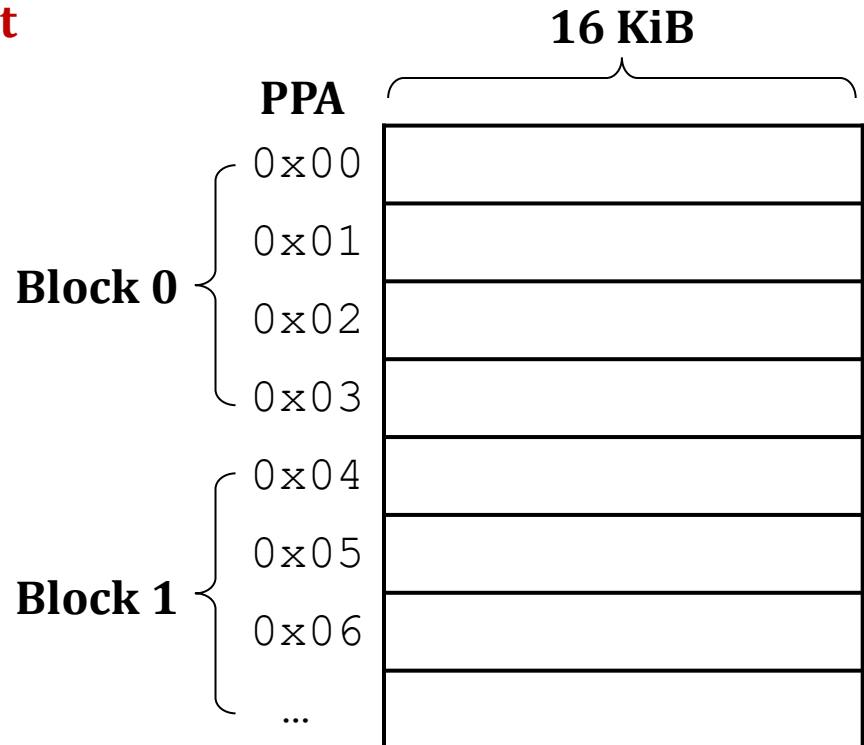   - Unnecessary fetch or eviction at the page cache

Small Write Requests

- Inefficiencies due to the erase-before-write property

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

`0b 0000 0000 0000 0100`

16-KiB Page Number   **4-KiB Offset**

LPA	PPA
0x00	–
0x01	–
0x02	–
0x03	–
0x04	–
0x05	–
…	…

16 KiB

PPA

Block 0
- 0x00
- 0x01
- 0x02
- 0x03

Block 1
- 0x04
- 0x05
- 0x06
- …

Small Write Requests

- Inefficiencies due to the erase-before-write property

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

`0b  0000 0000 0000 0100`

16-KiB Page Number   **4-KiB Offset**

LPA	PPA
0x00	–
0x01	–
0x02	–
0x03	–
0x04	–
0x05	–
…	…

16 KiB

PPA

Block 0
- 0x00 A
- 0x01
- 0x02
- 0x03

Block 1
- 0x04
- 0x05
- 0x06
- …

Small Write Requests

- Inefficiencies due to the erase-before-write property

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

`0b 0000 0000 0000 0100`

16-KiB Page Number   **4-KiB Offset**

LPA	PPA
0x00	–
0x01	**0x00**
0x02	–
0x03	–
0x04	–
0x05	–
…	…

16 KiB

PPA

Block 0
- 0x00 **A**
- 0x01
- 0x02
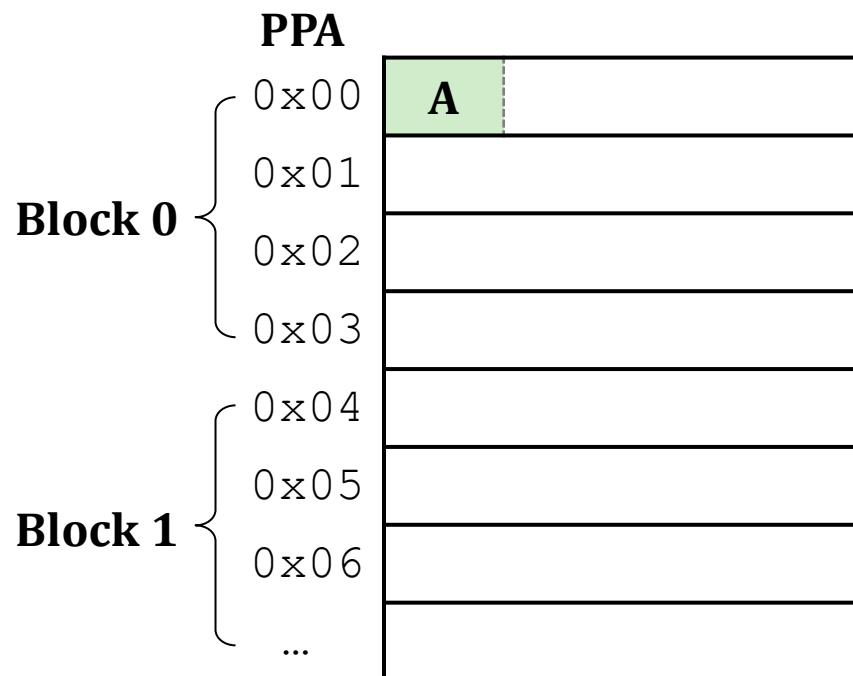- 0x03

Block 1
- 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**

LPA	PPA
0x00	–
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…

PPA

Block 0
- 0x00 A
- 0x01
- 0x02
- 0x03

Block 1
- 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**

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

PPA

Block 0
- 0x00   A
- 0x01   B   C
- 0x02
- 0x03

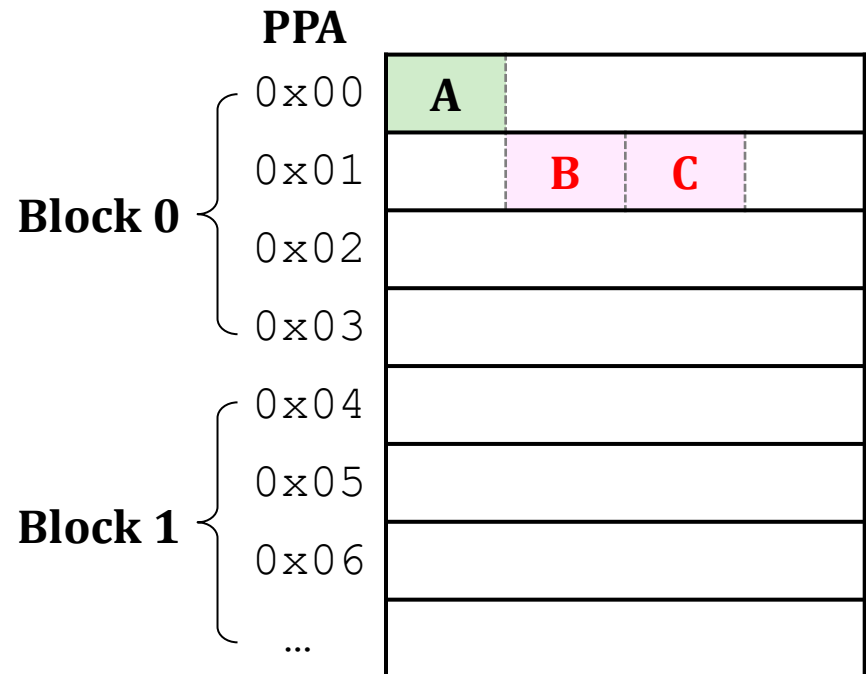Block 1
- 0x04
- 0x05
- 0x06
- ...

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**

LPA	PPA
0x00	**0x01**
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…

PPA

Block 0
- 0x00 A
- 0x01 B C
- 0x02
- 0x03

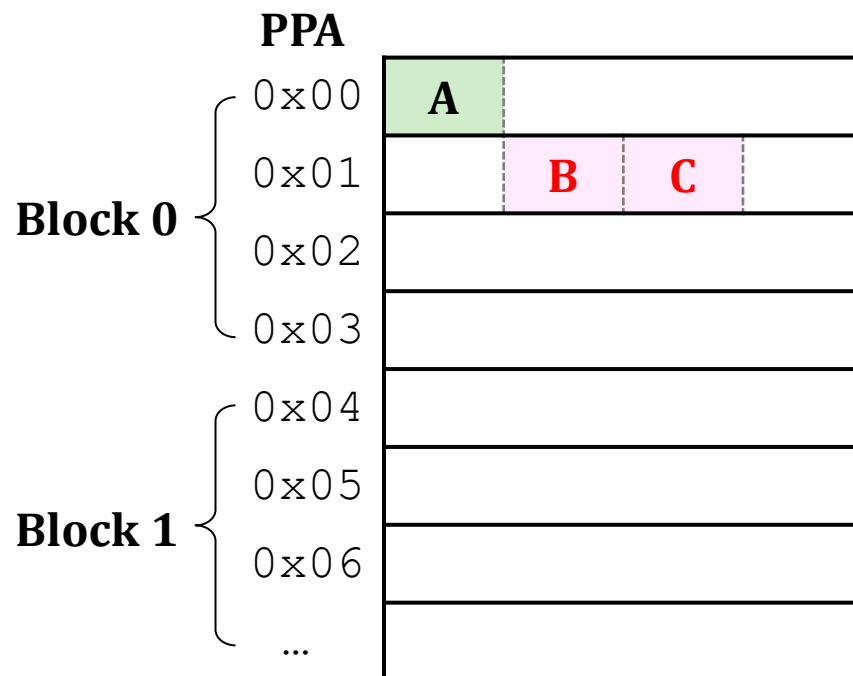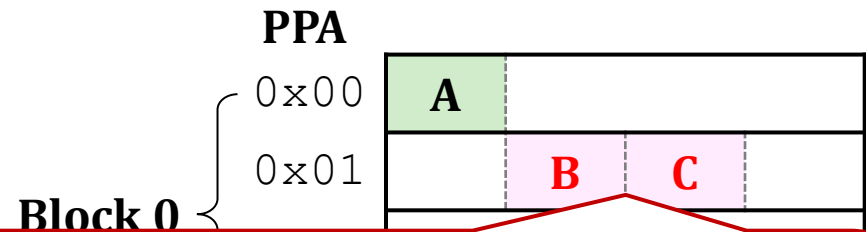Block 1
- 0x04
- 0x05
- 0x06
- …

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

LPA	PPA
0x00	**0x01**
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…

PPA

0x00   A

0x01   **B**   **C**

Block 0

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).
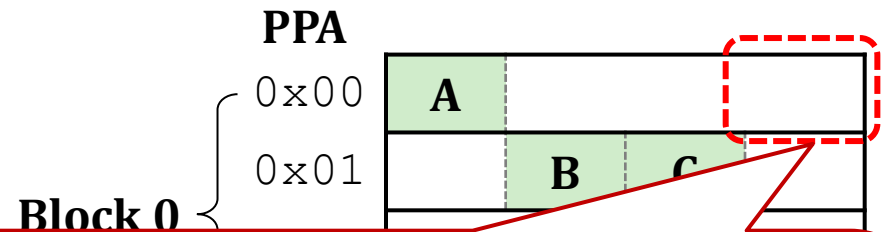
Small Write Requests

- 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**

LPA	PPA
0x00	0x01
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…

PPA

Block 0 { 0x00 [A] [ ] , 0x01 [B] [C] }

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

Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

LPA	PPA
0x00	0x01
0x01	0x00

PPA

Unused yet discarded

PPA		
0x00	A	
0x01		B	C
0x02		

Block 0

…	…

| … | |
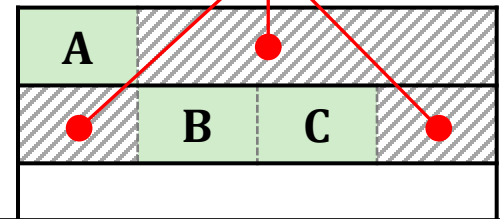
Small writes cause waste of P/E cycles:
More frequent garbage collections
→ Performance and lifetime degradation

Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

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

Read

PPA

Block 0 { 0x00, 0x01, 0x02, 0x03

Block 1 { 0x04, 0x05, 0x06, ...

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**

Modify

Read

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

PPA

Block 0
- 0x00 A
- 0x01 B C
- 0x02
- 0x03

Block 1
- 0x04
- 0x05
- 0x06
- ...

Small Write Requests

- Inefficiencies due to the erase-before-write property

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

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

LPA	PPA
0x00	0x01
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…

Modify

A **D**

Read   *Write*

PPA

0x00 **A**
0x01   **B** **C**
Block 0 0x02 **A**   **D**
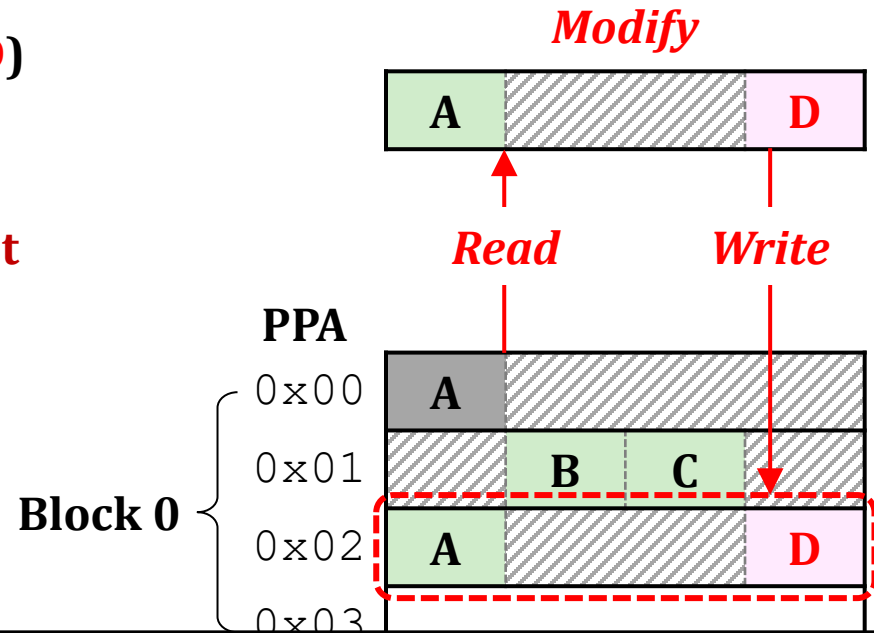0x03

0x04
Block 1 0x05
0x06
…

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**

Modify

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

Read   *Write*

PPA

0x00	**A**
0x01	**B** **C**
0x02	**A** **D**
0x03	

Block 0

Small writes cause read-modify-writes:
Waste of P/E cycles + additional read operations
→ Performance and lifetime degradation

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

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

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

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

Page Buffer

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

Block 0

0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Block 1

0x10	0x11	0x12	0x13
0x14	0x15	0x16	0x17
0x18	0x19	0x1A	0x0B
...			

PPA

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

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

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

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

Page Buffer

A			

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

Block 0

0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Block 1

0x10	0x11	0x12	0x13
0x14	0x15	0x16	0x17
0x18	0x19	0x1A	0x0B
...			

PPA

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

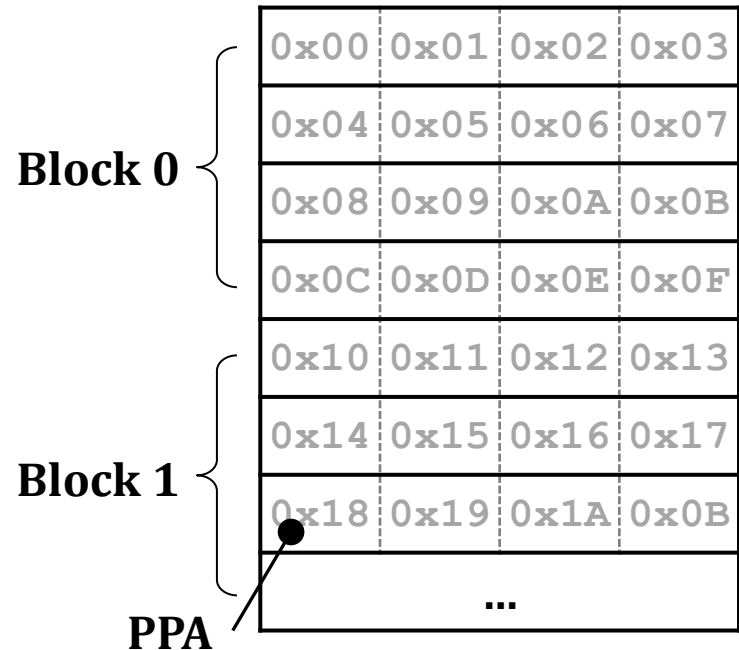**Req** (**LBA:** `0x04`, **Size:** `1`, **DIR:** `W`, **Data:** A)

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

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

Page Buffer

A			

LPA	PPA
0x00	–
0x01	–
...	–
0x04	**0x00**
...	–
0x07	–
...	...

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

Block 0

Block 1

PPA

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks
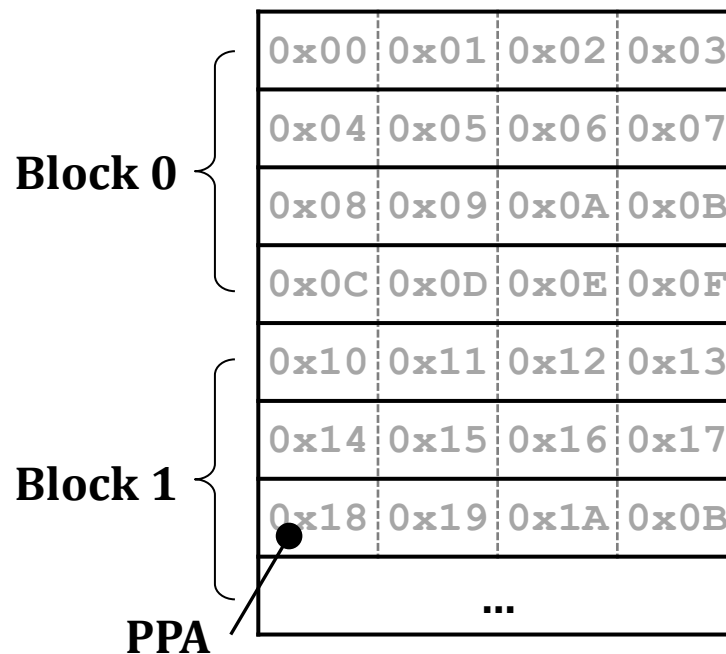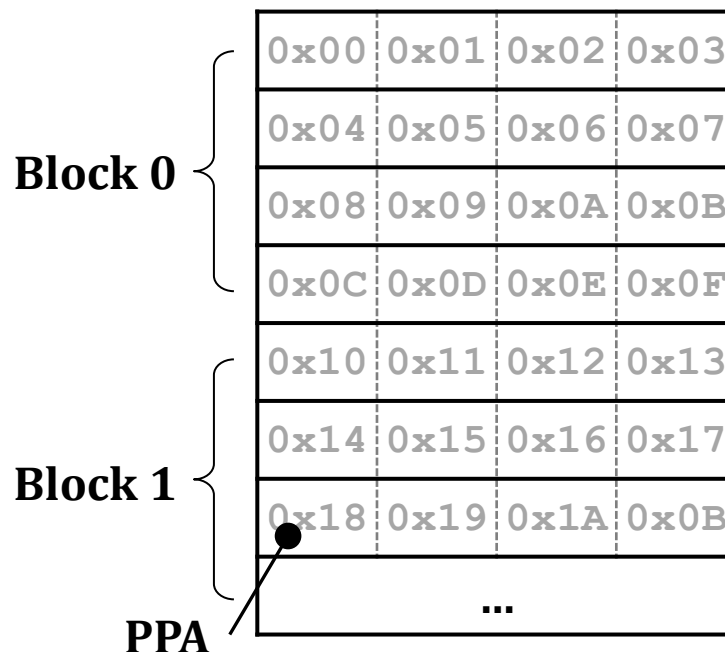
Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

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

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

Page Buffer

A			

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

Block 0

0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Block 1

0x10	0x11	0x12	0x13
0x14	0x15	0x16	0x17
0x18	0x19	0x1A	0x0B
...			

PPA

Fine-Grained Mapping + Page Buffer

■ Write a page only when there are sufficient data blocks

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

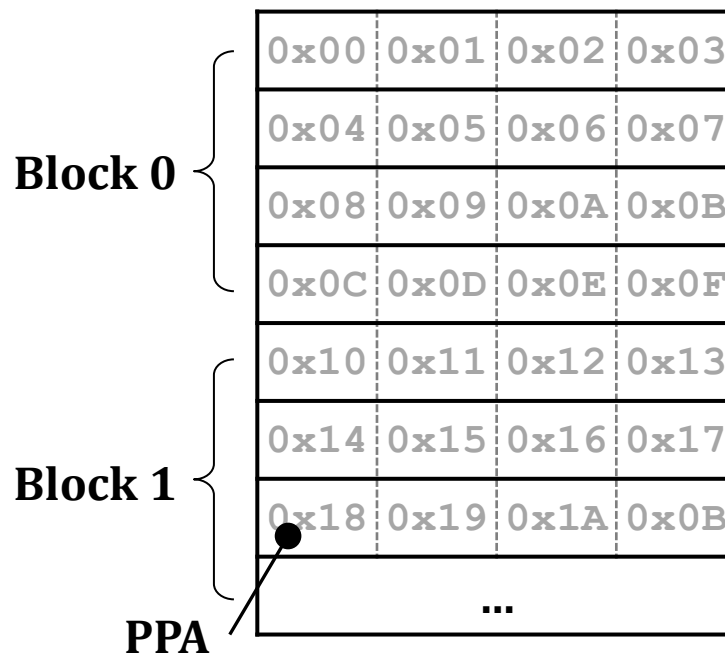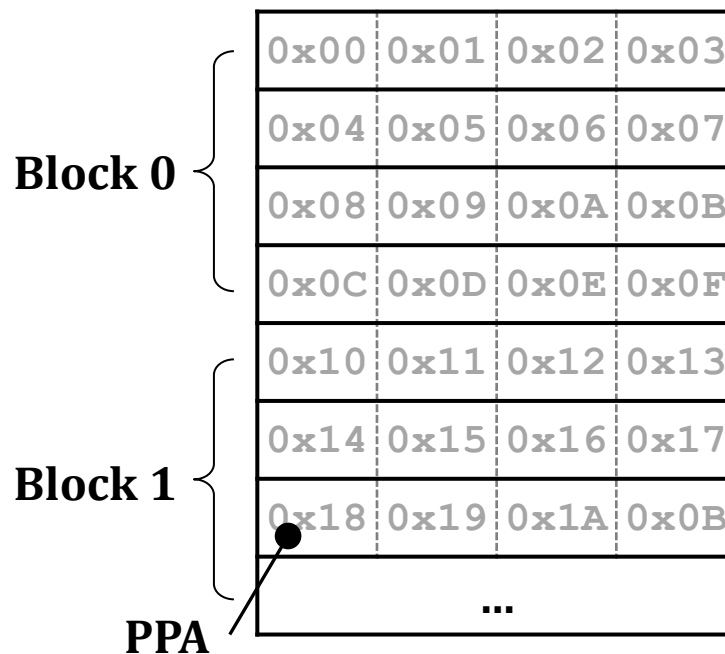**Req (LBA: 0x01, Size: 2, DIR: w, Data: B, C)**

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

Page Buffer

A	B	C	

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

Block 0

0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Block 1

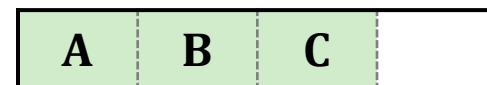| 0x10	0x11	0x12	0x13
0x14	0x15	0x16	0x17
0x18	0x19	0x1A	0x0B
...			

PPA

Fine-Grained Mapping + Page Buffer

■ Write a page only when there are sufficient data blocks
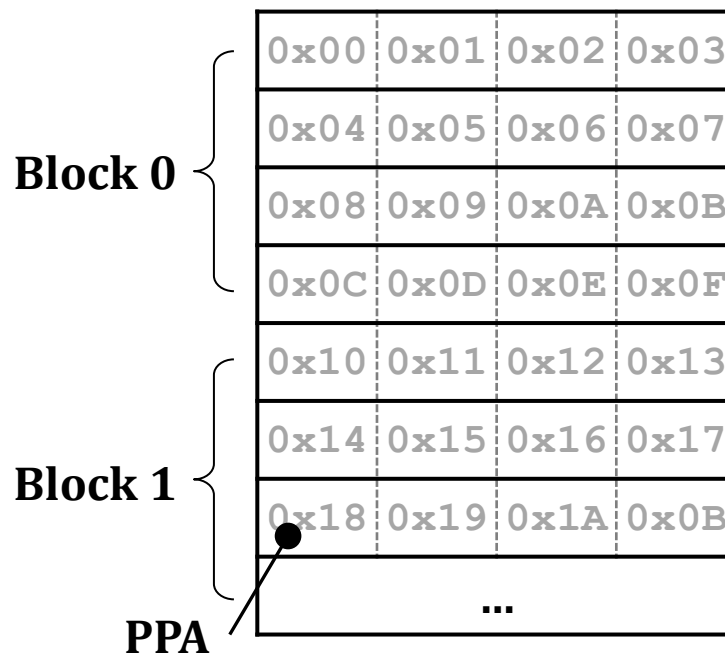
Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

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

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

Page Buffer

A	B	C	

LPA	PPA
0x00	–
0x01	**0x01**
...	–
0x04	0x00
...	–
0x07	–
...	...

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

Block 0

Block 1

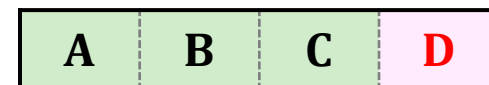**PPA**

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

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

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

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

Page Buffer

A	B	C	

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

Block 0

0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Block 1

0x10	0x11	0x12	0x13
0x14	0x15	0x16	0x17
0x18	0x19	0x1A	0x0B
...			

PPA

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks
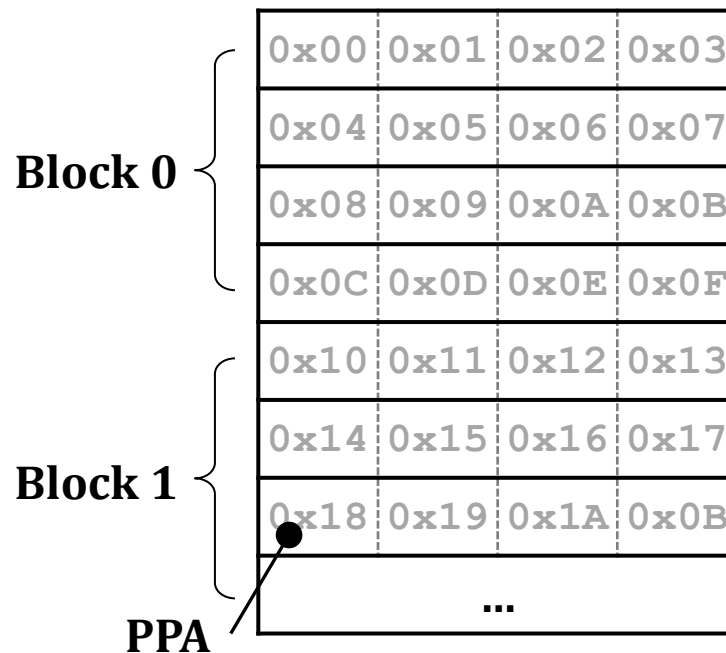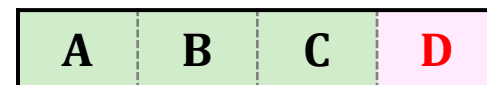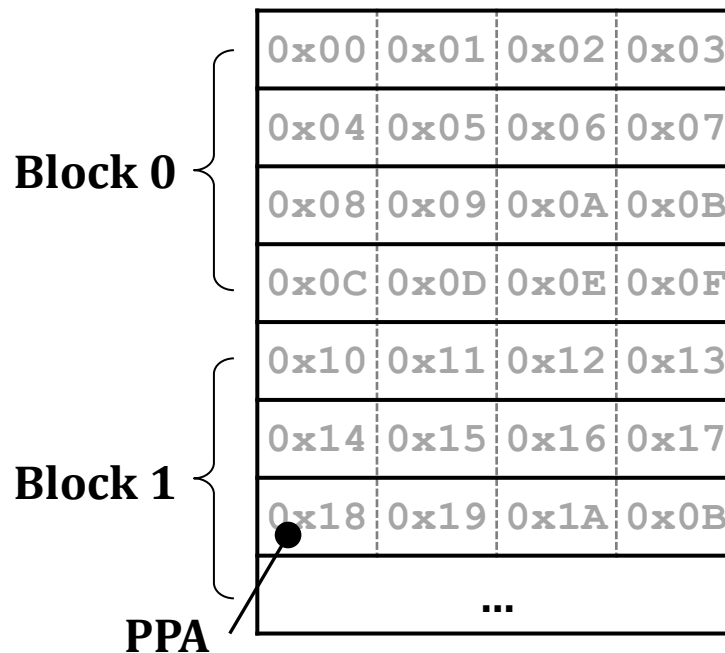
Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

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

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

Page Buffer

A	B	C	D

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

Block 0

0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Block 1

0x10	0x11	0x12	0x13
0x14	0x15	0x16	0x17
0x18	0x19	0x1A	0x0B
...			

PPA

Fine-Grained Mapping + Page Buffer
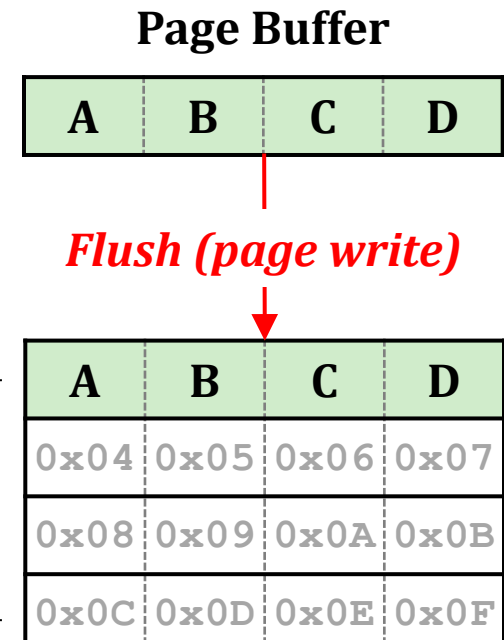
- Write a page only when there are sufficient data blocks

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

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

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

Page Buffer

A	B	C	D

LPA	PPA
0x00	–
0x01	0x01
...	–
0x04	0x00
...	–
0x07	**0x03**
...	...

Block 0
0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Block 1
0x10	0x11	0x12	0x13
0x14	0x15	0x16	0x17
0x18	0x19	0x1A	0x0B
...			

PPA

Fine-Grained Mapping + Page Buffer

■ Write a page only when there are sufficient data blocks

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

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

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

Page Buffer

A	B	C	D

Flush (page write)

LPA	PPA
0x00	–
0x01	0x01
…	–

Block 0

A	B	C	D
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Fine-grained mapping significantly reduces the number of NAND flash operations:
3 writes (+1 read) → 1 writes

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
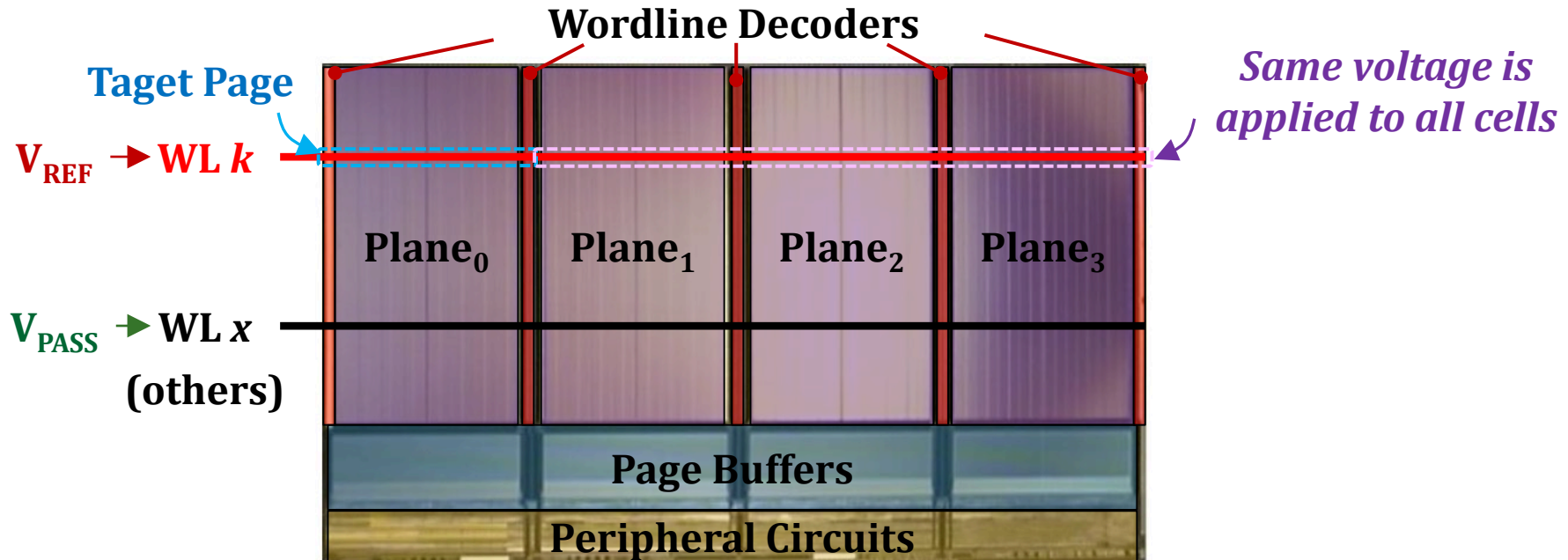 - Page buffers are implemented by using volatile memory (e.g., SRAM or DRAM).

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

Today's Agenda

- Fine-Grained Mapping

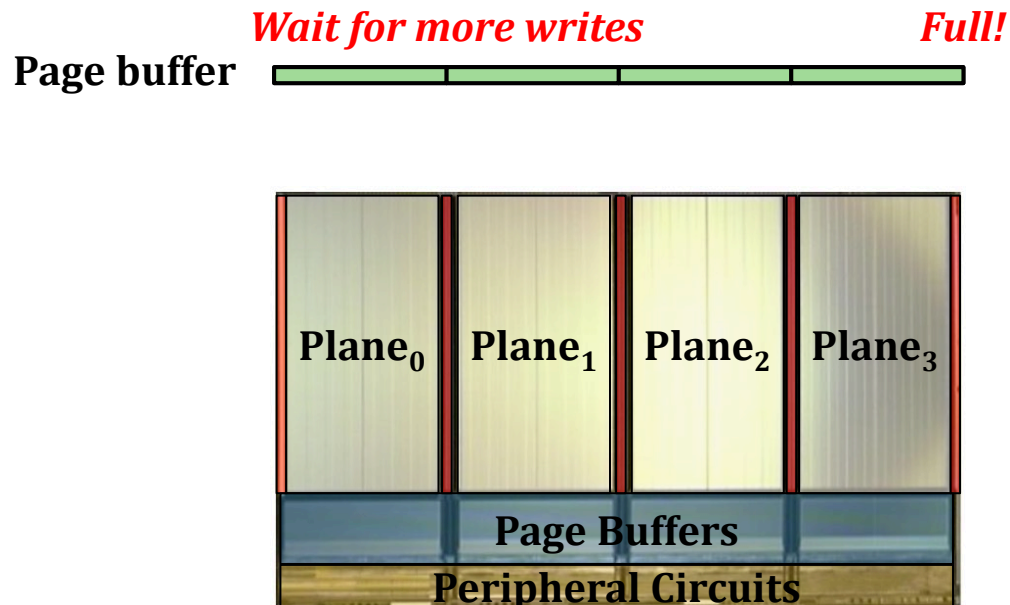- **Multi-plane Operation-Aware Blk. Mgmt.**

Recap: Multi-Plane Operations

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

Wordline Decoders

Taget Page

Same voltage is applied to all cells

V_{REF} → **WL k**

V_{PASS} → **WL x (others)**

Plane$_0$ **Plane$_1$** **Plane$_2$** **Plane$_3$**

Page Buffers

Peripheral Circuits

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

Multi-Plane-Aware Data Placement

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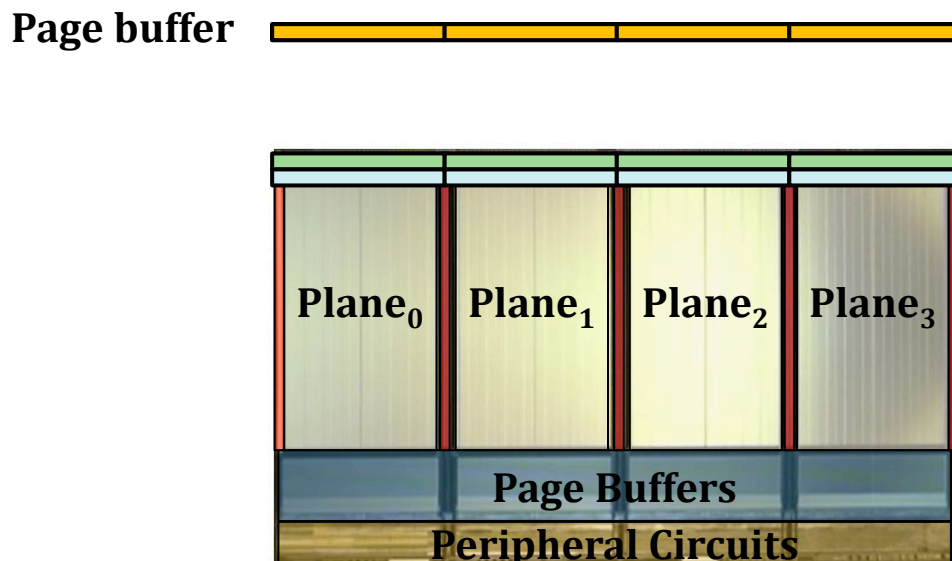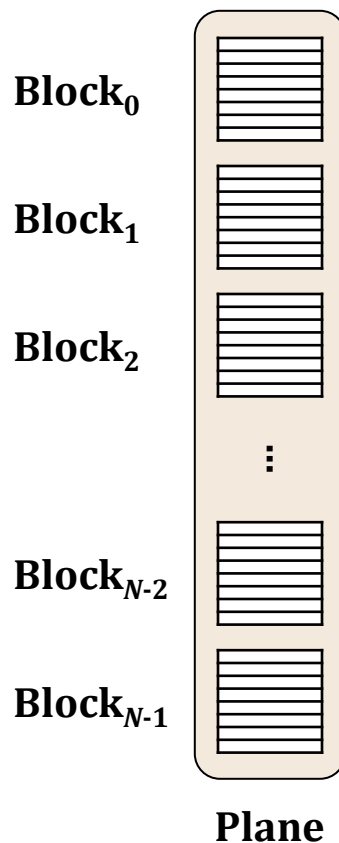 - Flush N_{plane} pages at once after buffering them

Wait for more writes   *Full!*

Page buffer

Plane$_0$ **Plane$_1$** **Plane$_2$** **Plane$_3$**

Page Buffers

Peripheral Circuits

Multi-Plane-Aware Data Placement

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

Page buffer ▭▭▭▭

Plane$_0$ Plane$_1$ Plane$_2$ Plane$_3$

Page Buffers

Peripheral Circuits
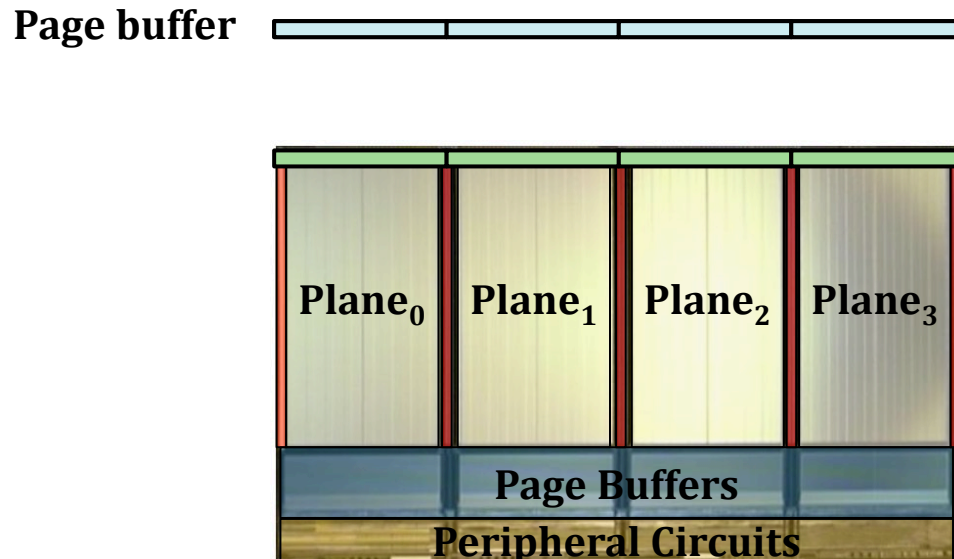
Multi-Plane-Aware Data Placement

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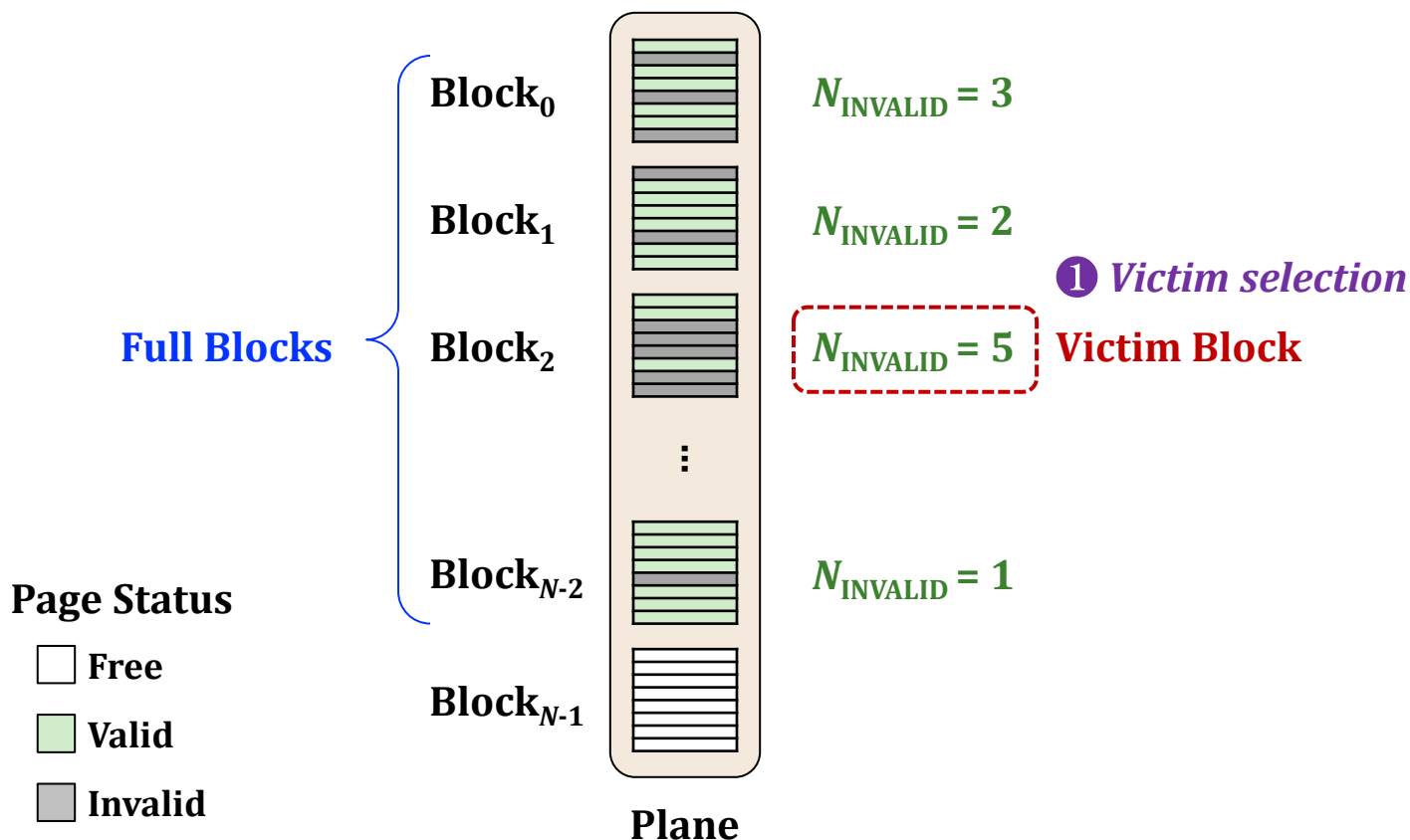   - Superblock-based block management

Page buffer

| Plane$_0$ | Plane$_1$ | Plane$_2$ | Plane$_3$ |

Page Buffers

Peripheral Circuits

Multi-Plane-Aware Block Management

■ Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).
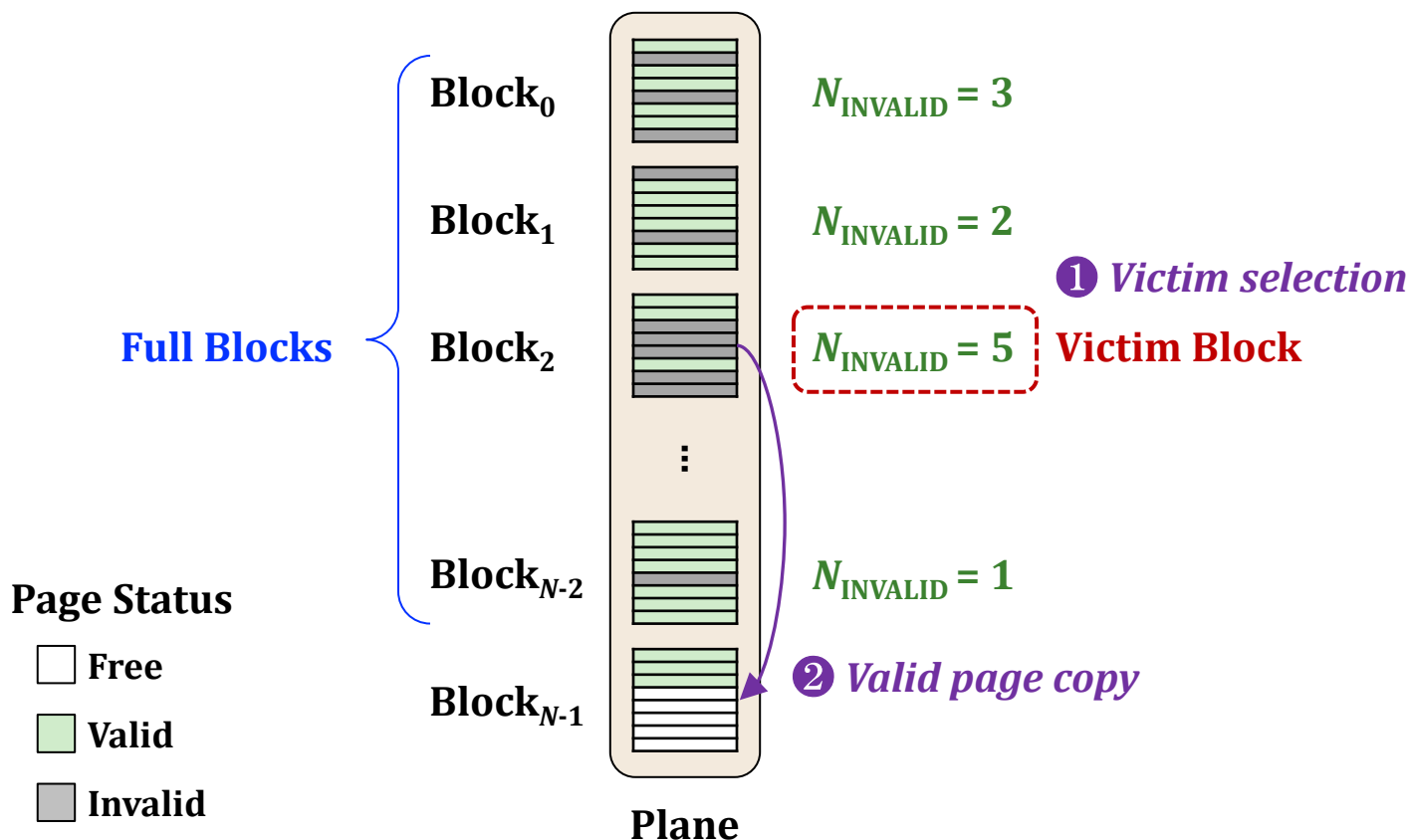
Block$_0$

Block$_1$

Block$_2$

⋮

Block$_{N-2}$
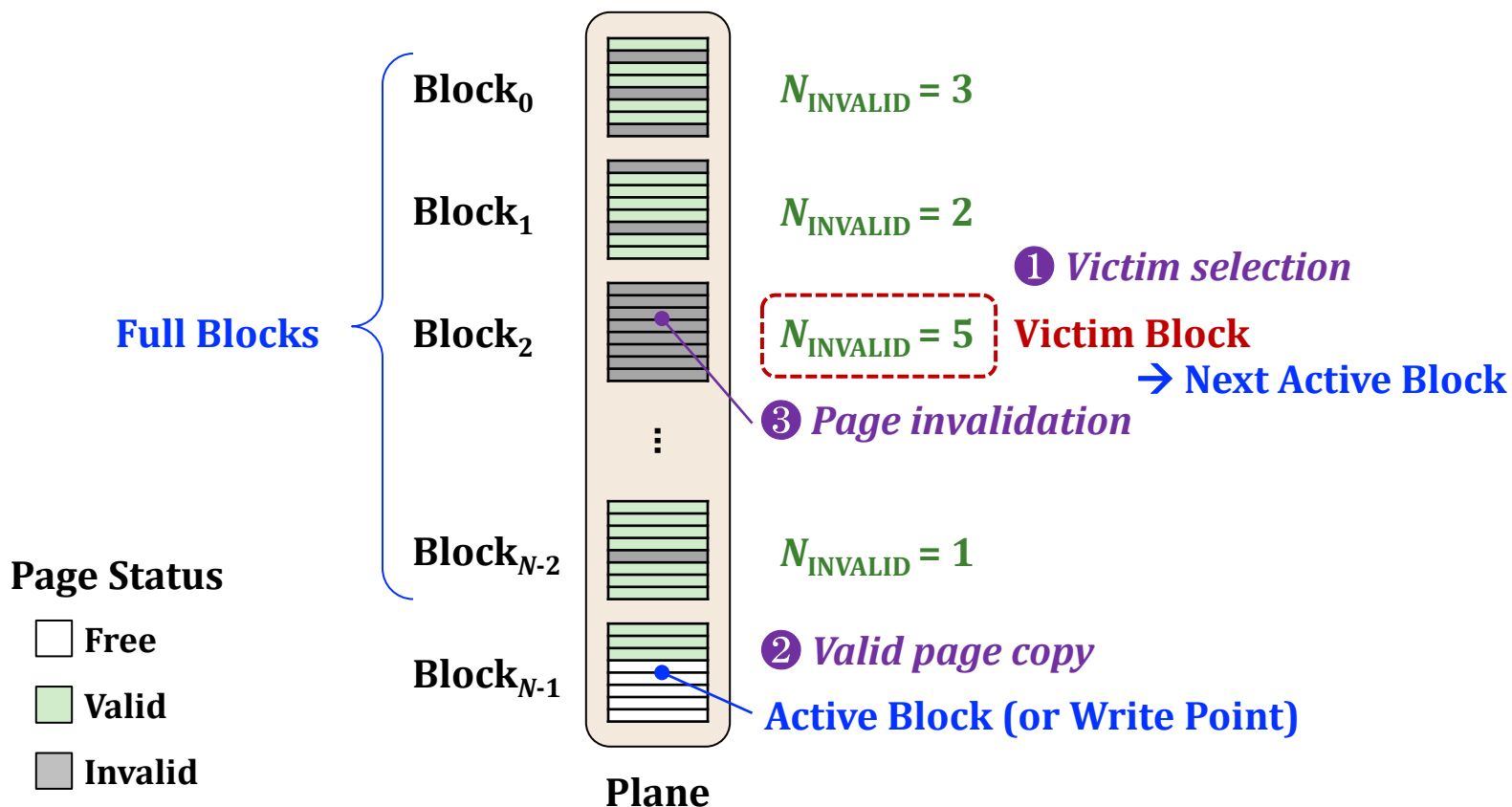
Block$_{N-1}$

Plane

Multi-Plane-Aware Block Management

- Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).

Full Blocks

Block$_0$ $N_{\text{INVALID}} = 3$

Block$_1$ $N_{\text{INVALID}} = 2$

❶ *Victim selection*

Block$_2$ $N_{\text{INVALID}} = 5$ **Victim Block**

⋮

Block$_{N-2}$ $N_{\text{INVALID}} = 1$

Block$_{N-1}$

Plane

Page Status

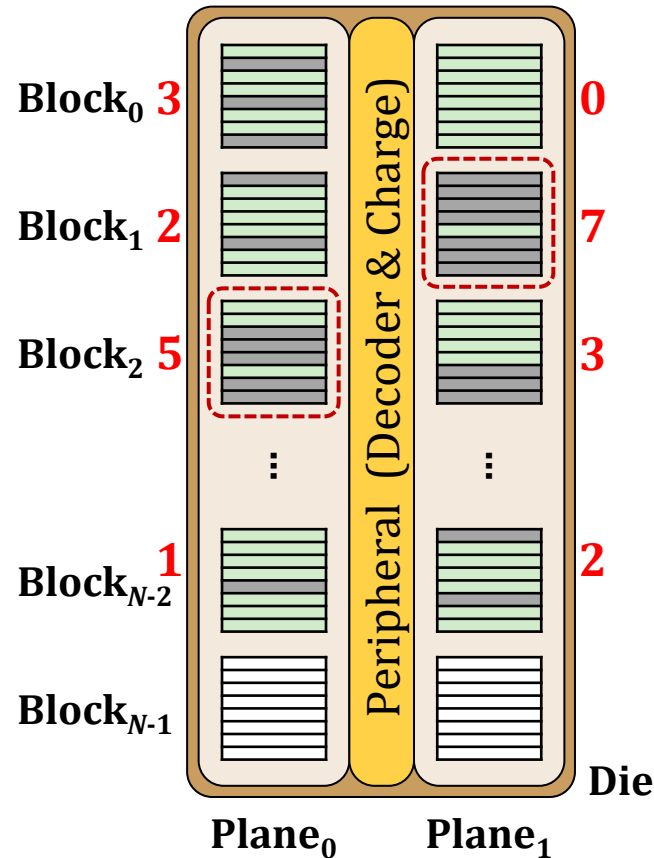☐ **Free**

🟩 **Valid**

⬜ **Invalid**

Multi-Plane-Aware Block Management

- Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).

Block$_0$ $N_{INVALID} = 3$

Block$_1$ $N_{INVALID} = 2$

❶ *Victim selection*

Full Blocks   **Block$_2$** $N_{INVALID} = 5$ **Victim Block**

Block$_{N-2}$ $N_{INVALID} = 1$

Page Status

❷ *Valid page copy*

☐ **Free**

Block$_{N-1}$

■ **Valid**

■ **Invalid**

Plane

Multi-Plane-Aware Block Management

- Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).
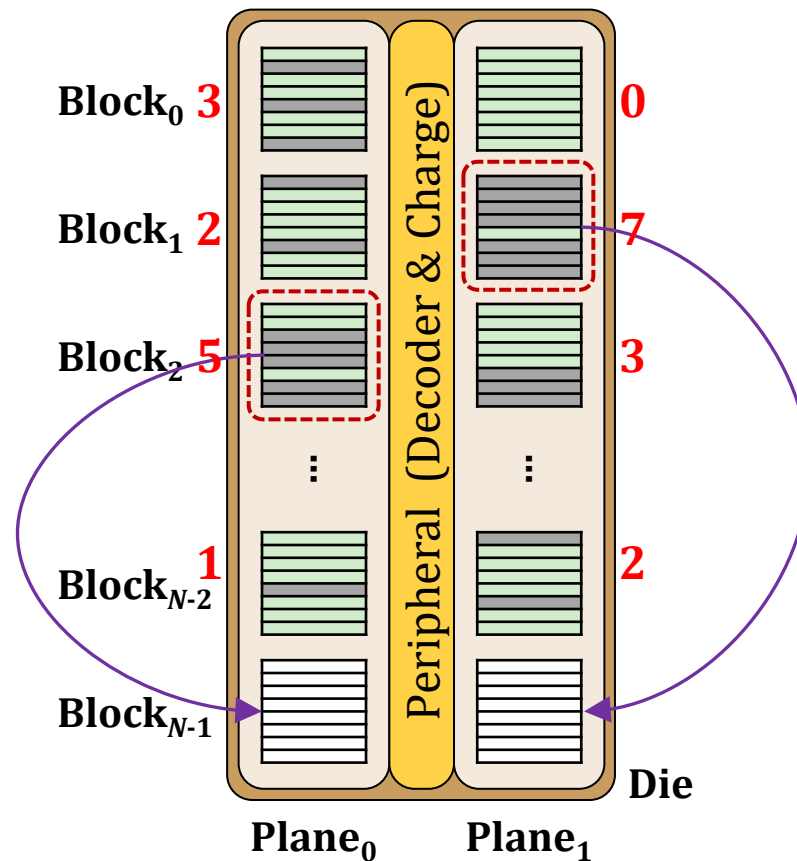
$Block_0$ — $N_{INVALID} = 3$

$Block_1$ — $N_{INVALID} = 2$

❶ *Victim selection*

$Block_2$ — $N_{INVALID} = 5$ Victim Block → Next Active Block

Full Blocks

❸ *Page invalidation*

$Block_{N-2}$ — $N_{INVALID} = 1$

❷ *Valid page copy*

$Block_{N-1}$ — Active Block (or Write Point)

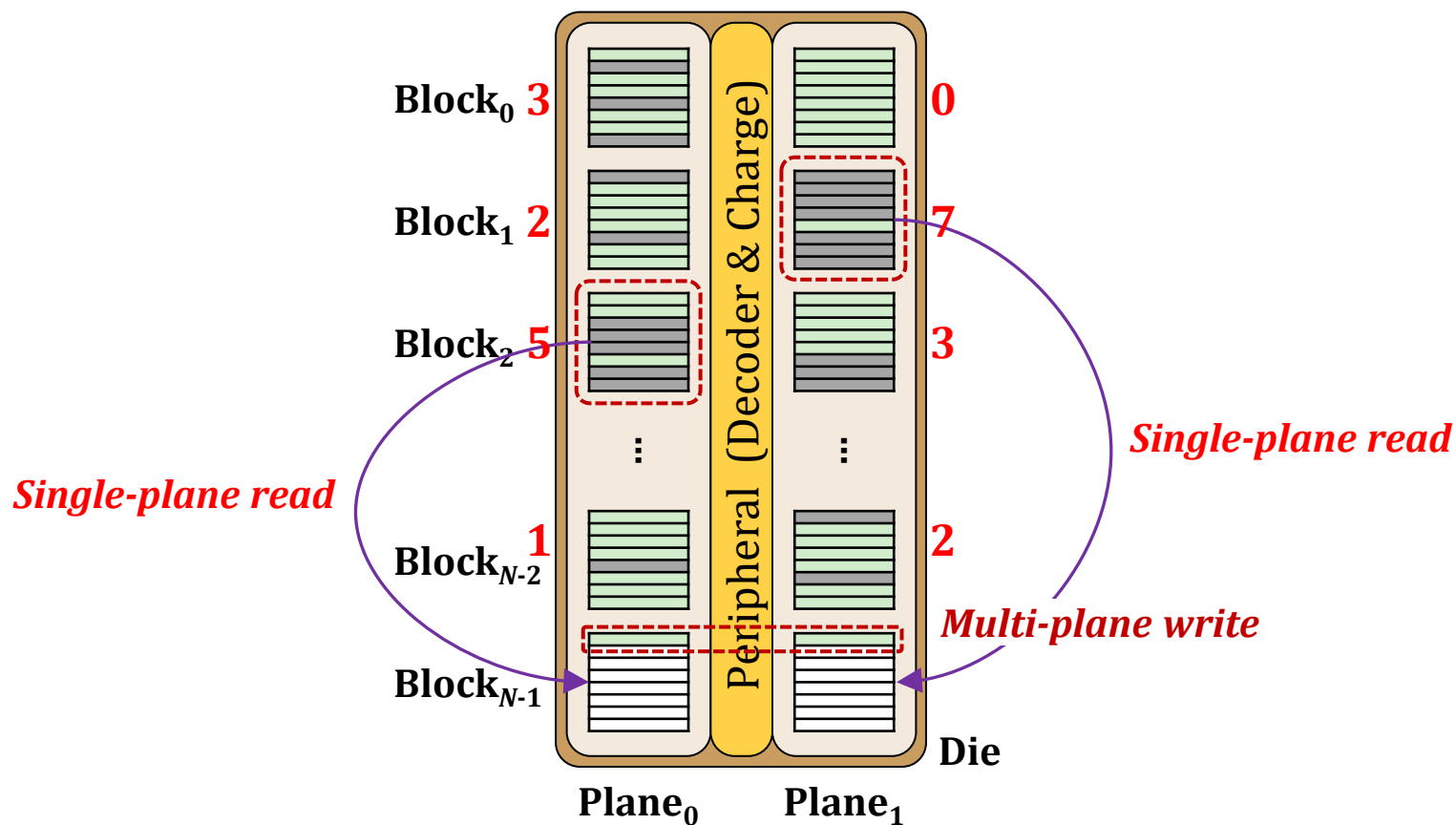**Page Status**
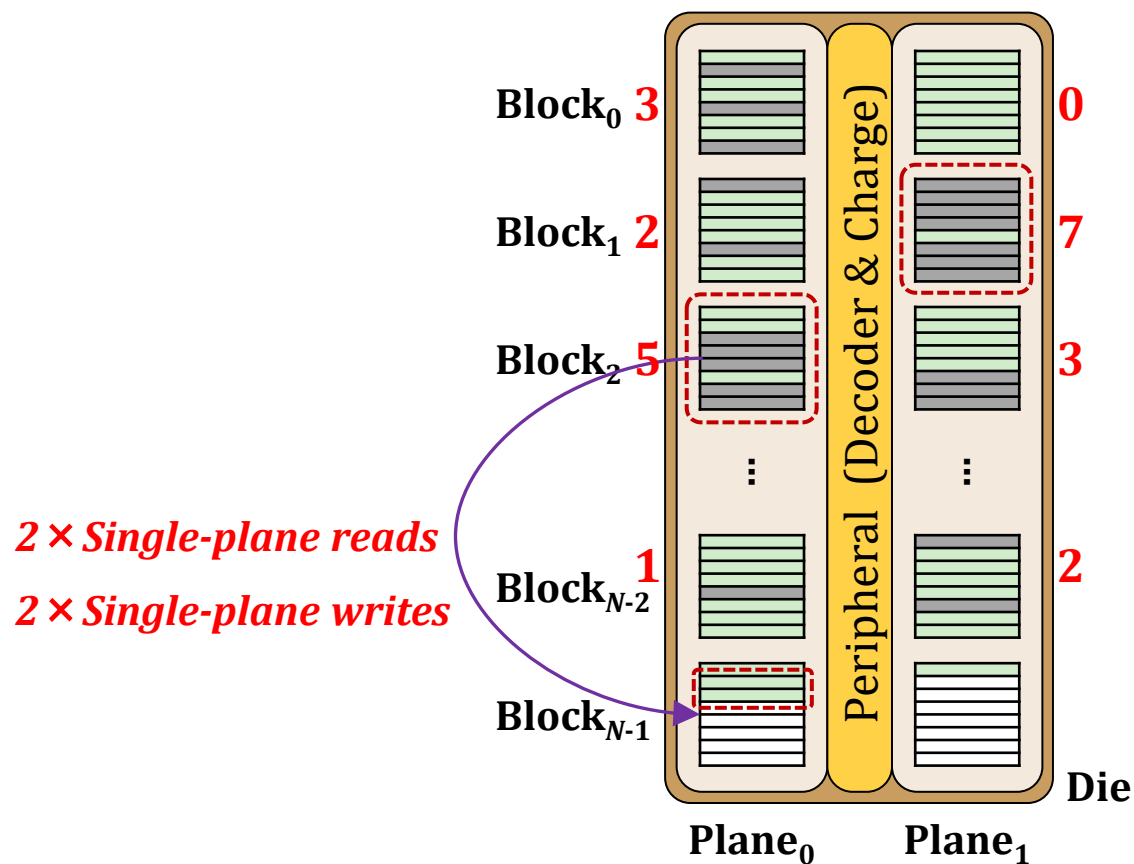- ☐ Free
- ☐ Valid
- ☐ Invalid

Plane

Multi-Plane-Aware Block Management

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

Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.
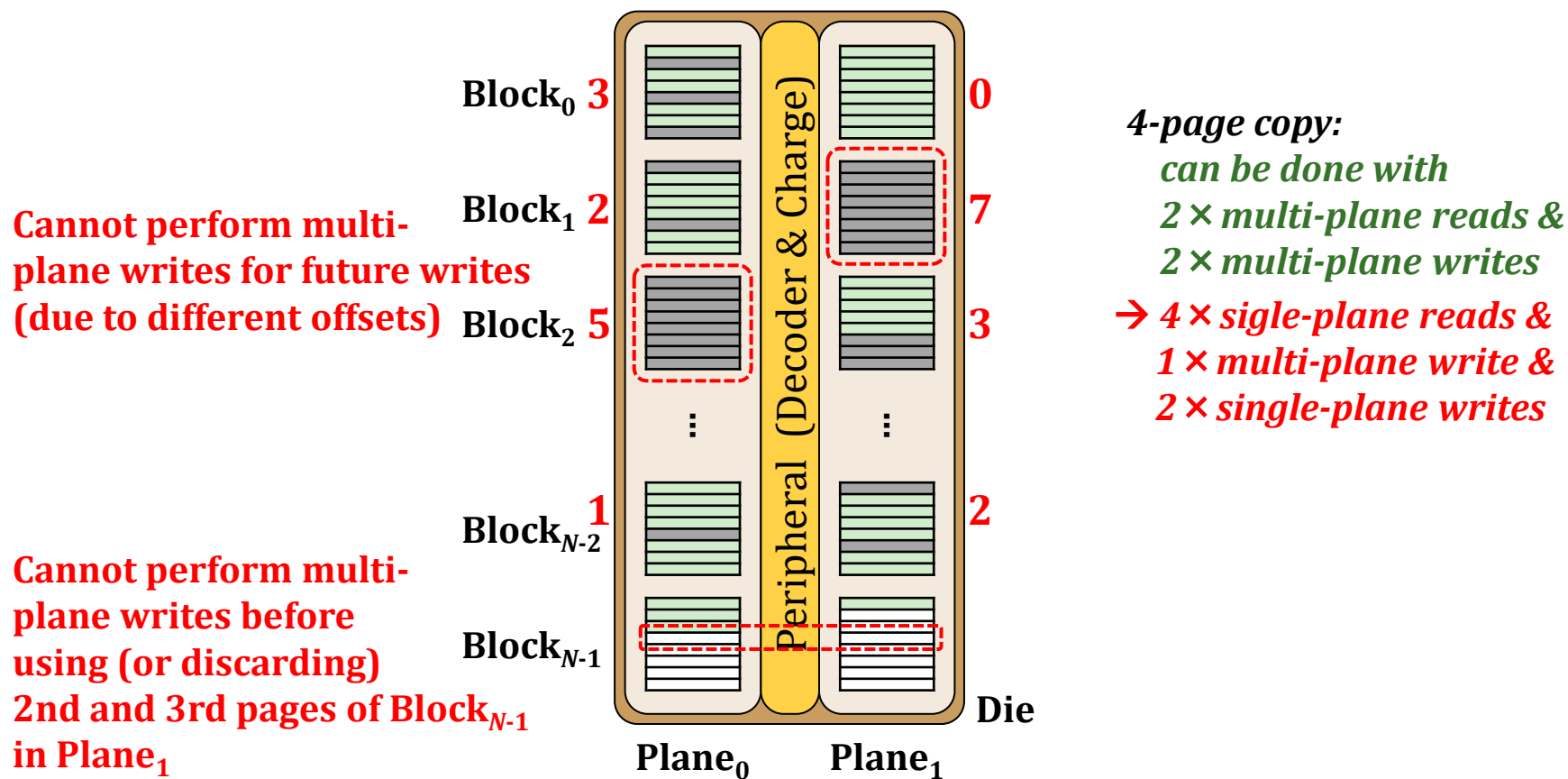
Multi-Plane-Aware Block Management

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

Multi-Plane-Aware Block Management

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

Block$_0$ 3   0
Block$_1$ 2   7
Block$_2$ 5   3

Peripheral (Decoder & Charge)

2 × Single-plane reads
2 × Single-plane writes

Block$_{N-2}$ 1   2
Block$_{N-1}$

Plane$_0$ Plane$_1$

Die

Multi-Plane-Aware Block Management

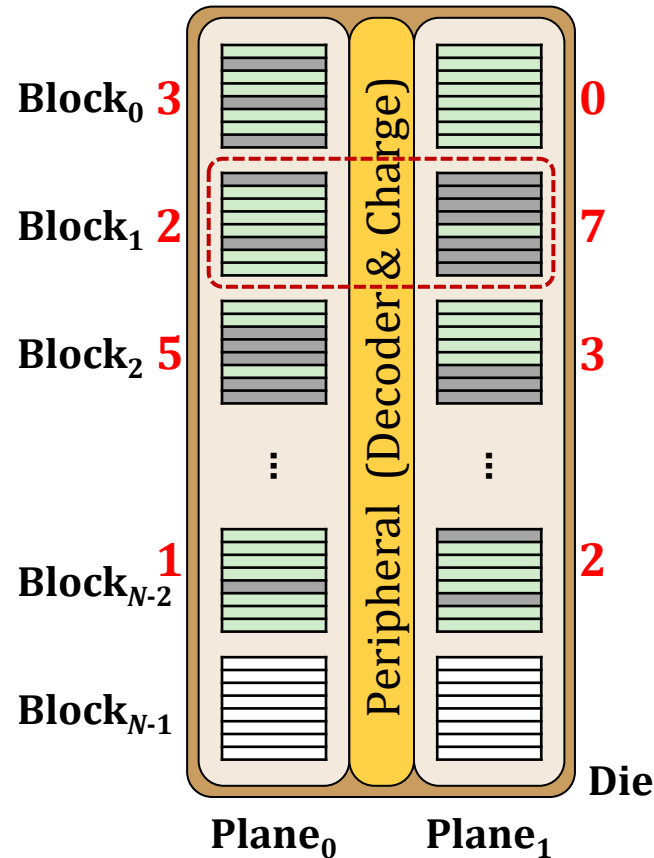- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.

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$_1$

Block$_0$ 3

Block$_1$ 2

Block$_2$ 5

Block$_{N-2}$ 1

Block$_{N-1}$

Peripheral (Decoder & Charge)

0

7

3

2

Die

Plane$_0$ Plane$_1$

4-page copy:
can be done with
2 × multi-plane reads &
2 × multi-plane writes

→ *4 × sigle-plane reads &*
1 × multi-plane write &
2 × single-plane writes

Multi-Plane-Aware Block Management

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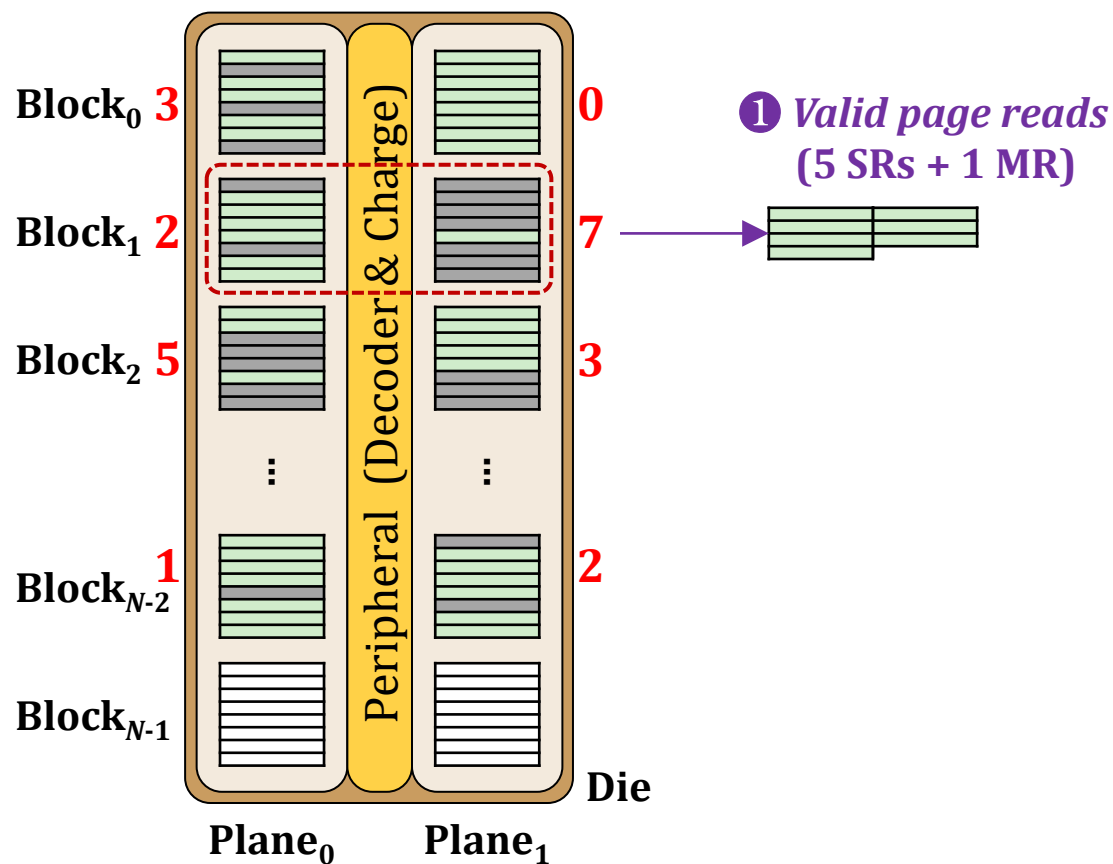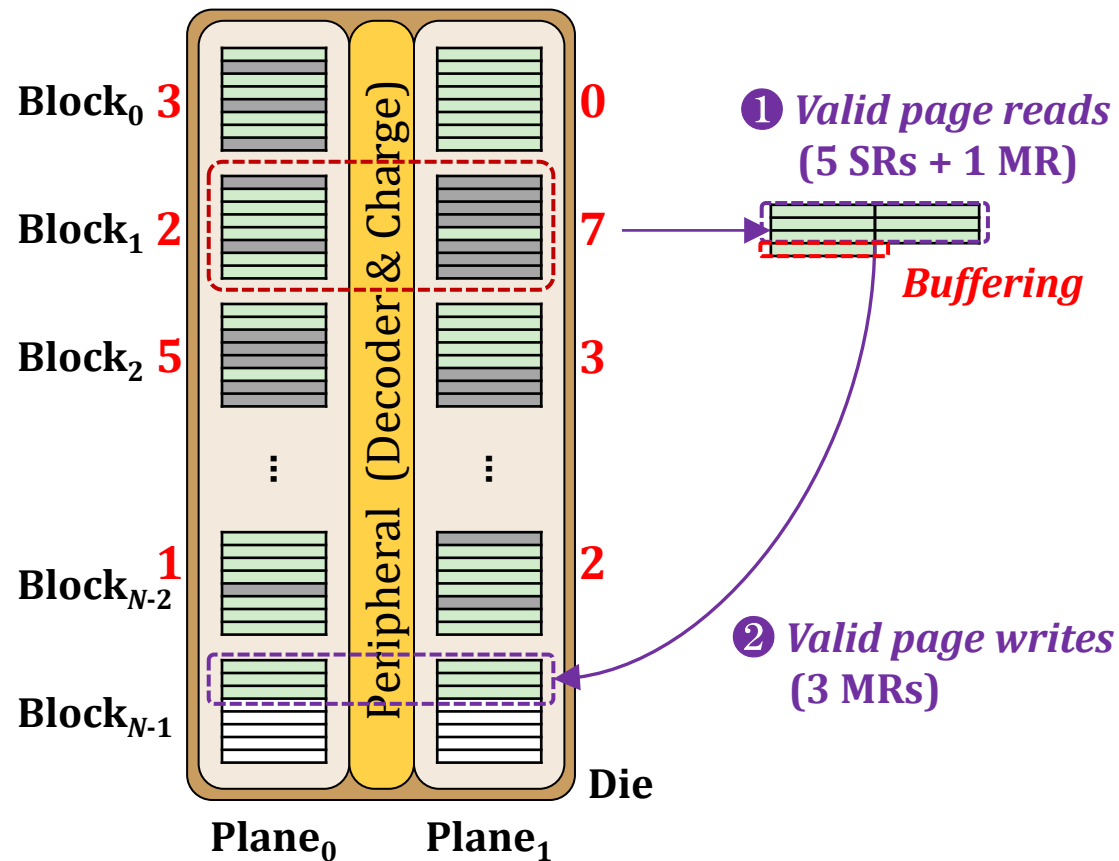
Multi-Plane-Aware Block Management

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

Block$_0$ **3**   **0**

Block$_1$ **2**   **7**

Block$_2$ **5**   **3**

Block$_{N-2}$ **1**   **2**

Block$_{N-1}$

Peripheral (Decoder & Charge)

Plane$_0$ **Plane$_1$**   **Die**
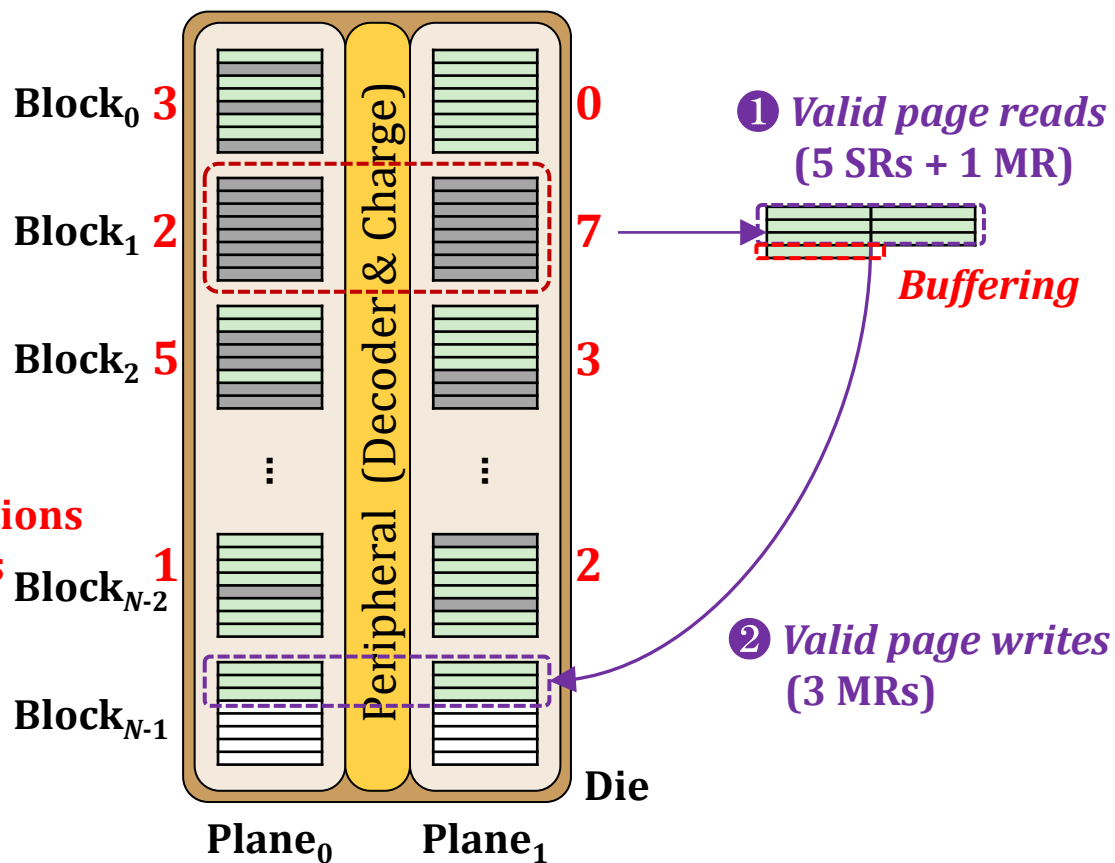
❶ *Valid page reads (5 SRs + 1 MR)*

Multi-Plane-Aware Block Management

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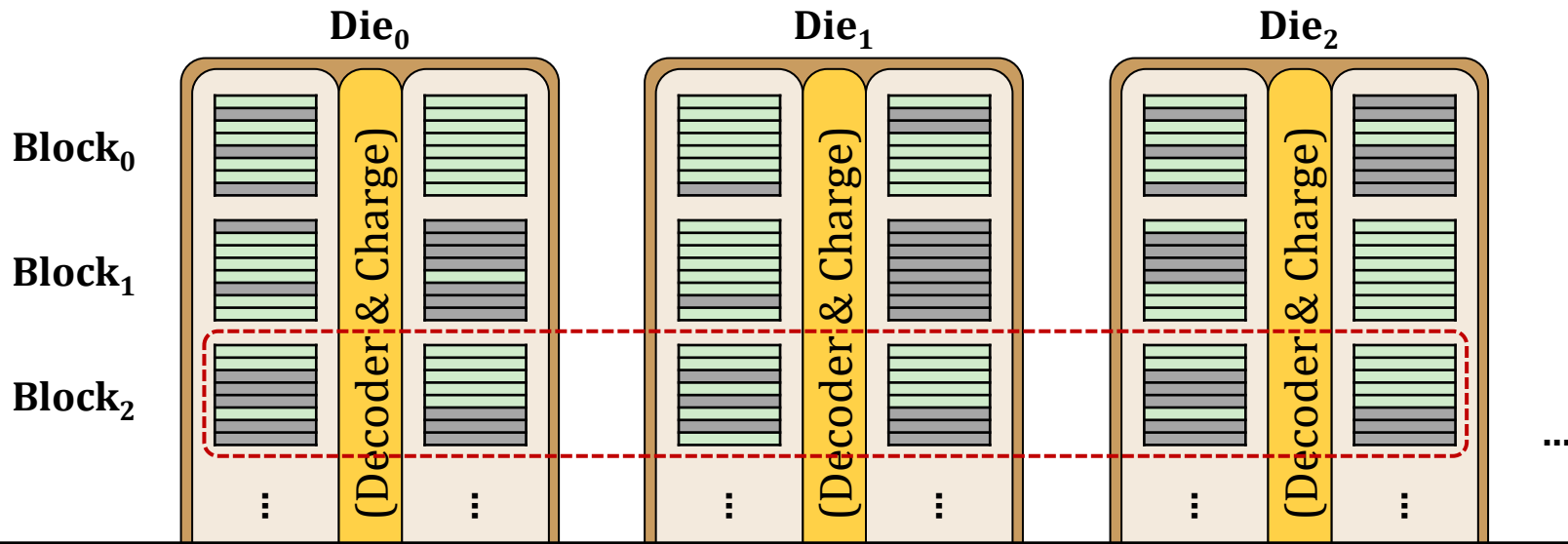
Multi-Plane-Aware Block Management

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

Pros:
Keep performing multi-plane writes

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

$Block_0$ **3**
$Block_1$ **2**
$Block_2$ **5**

⋮

$Block_{N-2}$ **1**
$Block_{N-1}$

Peripheral (Decoder & Charge)

0
7
3

⋮

2

Plane₀   **Plane₁**

Die

❶ *Valid page reads*
(5 SRs + 1 MR)

Buffering

❷ *Valid page writes*
(3 MRs)

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

Fine-Grained Mapping &
Multi-Plane Operation-Aware Block Management

Dr. Jisung Park

Prof. Onur Mutlu

ETH Zürich

Spring 2022

15 April 2022