

Jin-Soo Kim  
([jinsoo.kim@snu.ac.kr](mailto:jinsoo.kim@snu.ac.kr))

Systems Software &  
Architecture Lab.  
Seoul National University

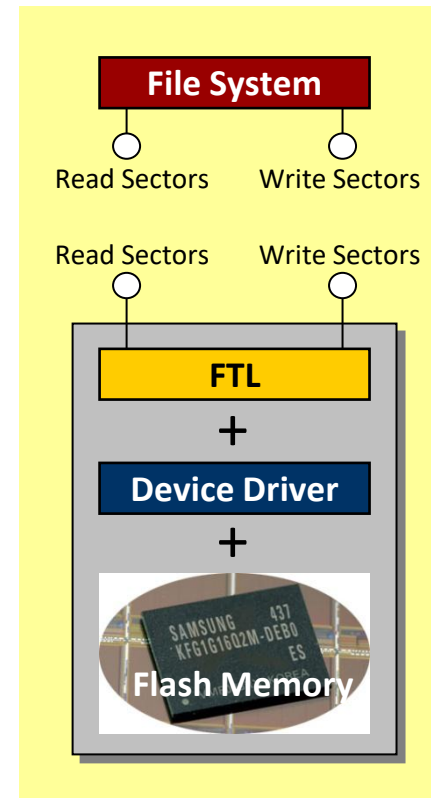
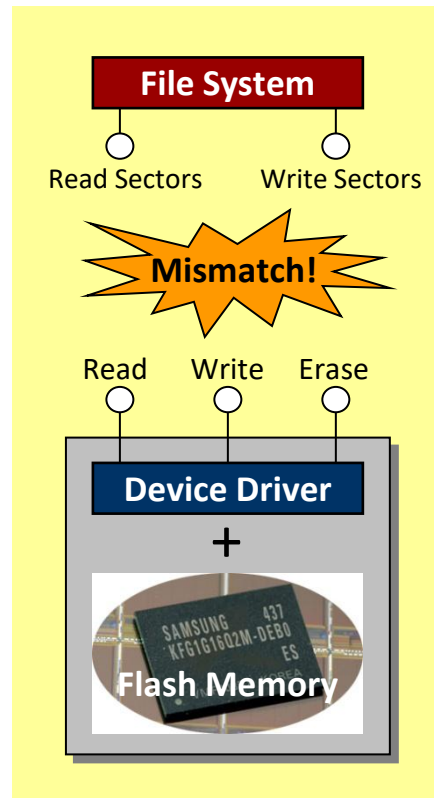
Spring 2026

# Flash Translation Layers

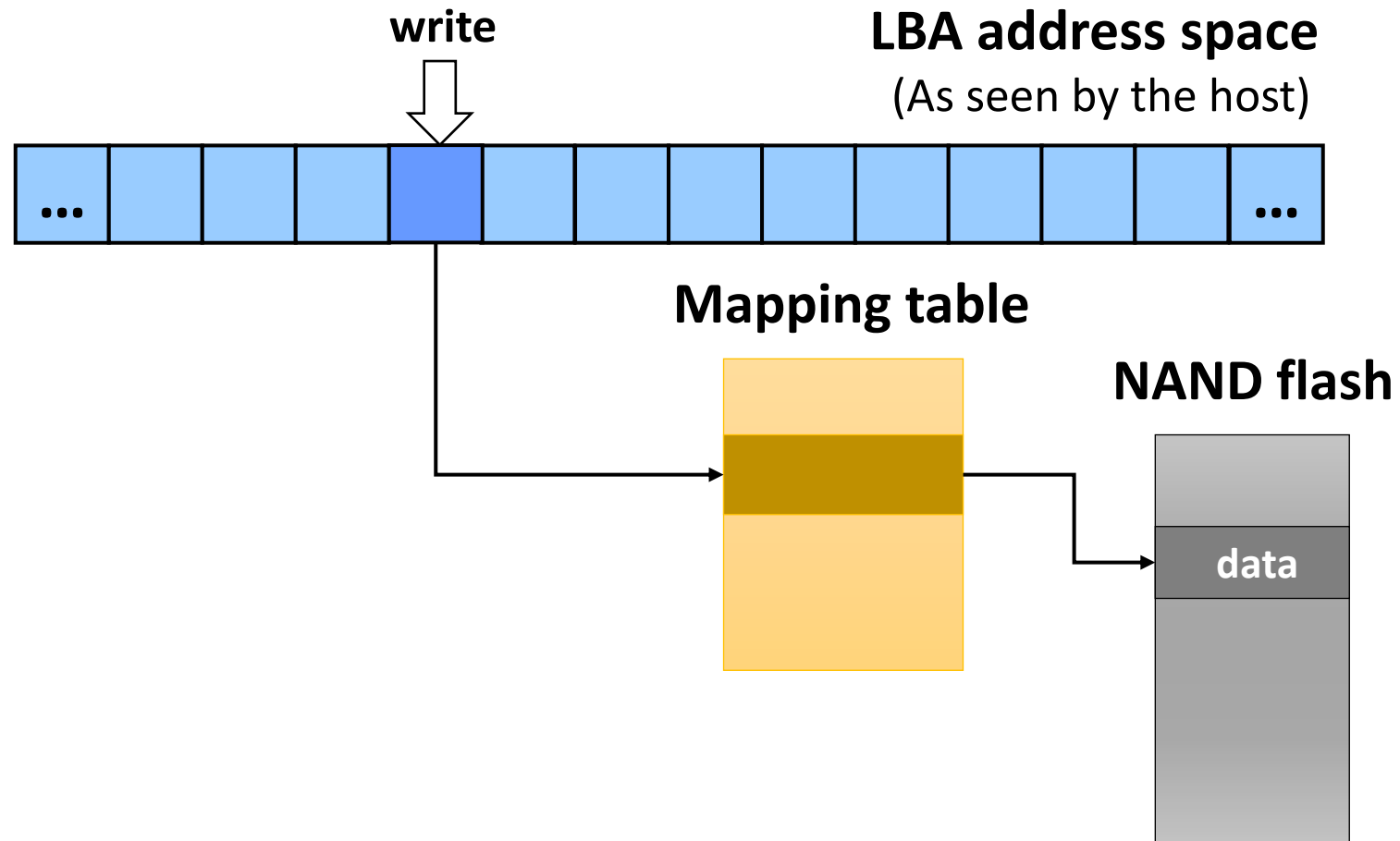


# What is FTL?

- A software layer to make NAND flash fully emulate traditional block devices (or disks)

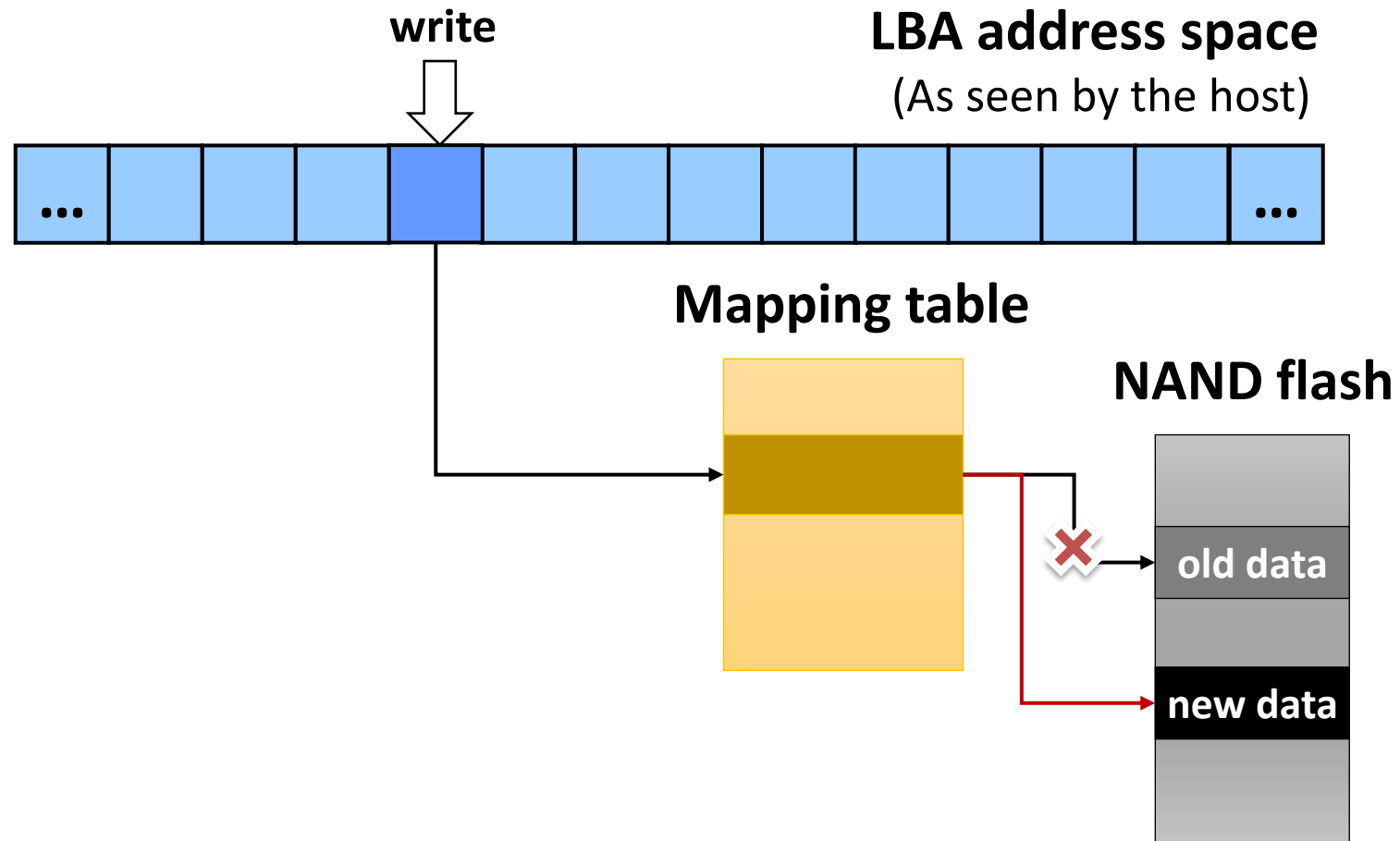


# Address Mapping

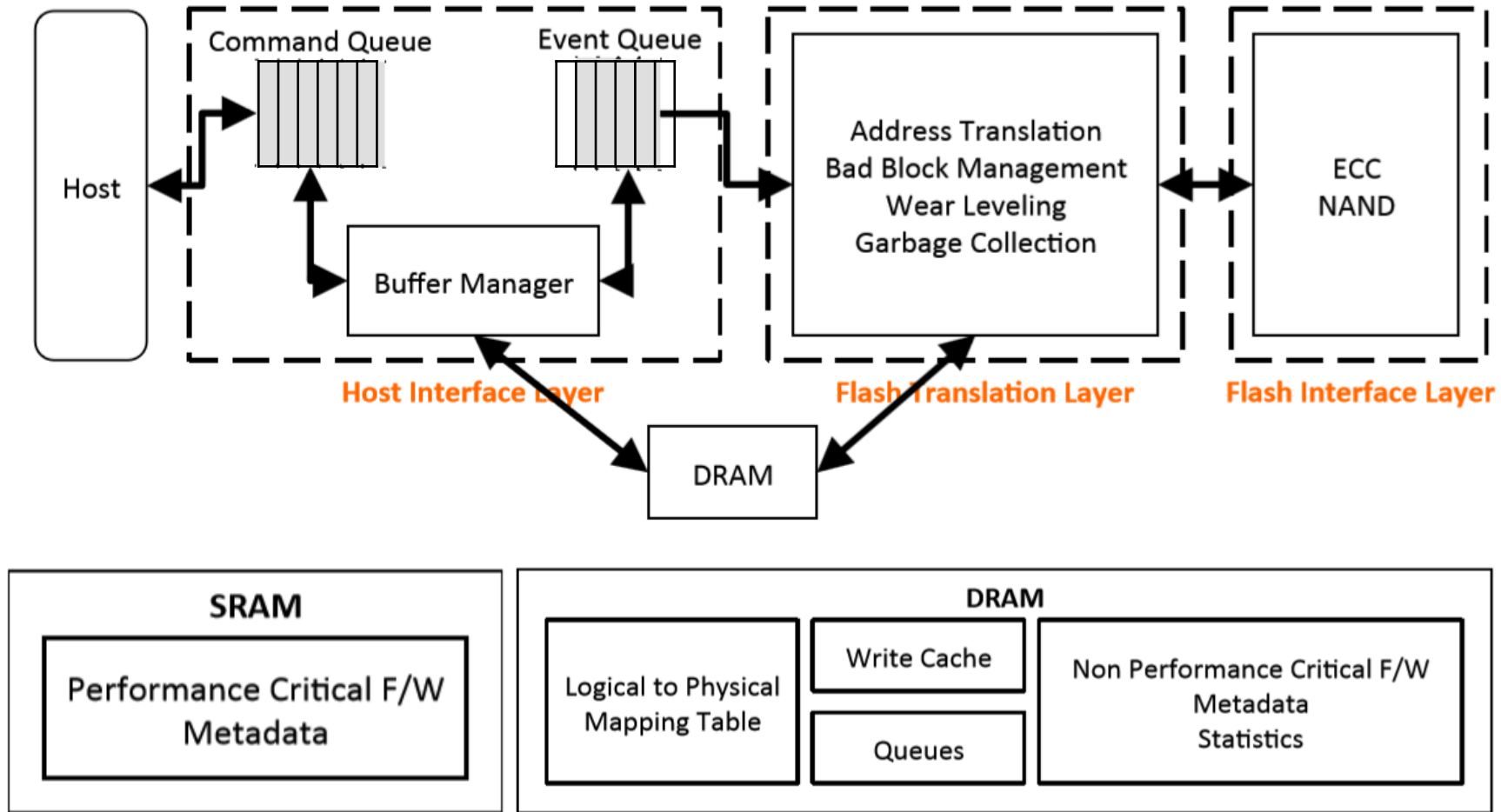


# Address Mapping

- Required due to “no overwrite” characteristic



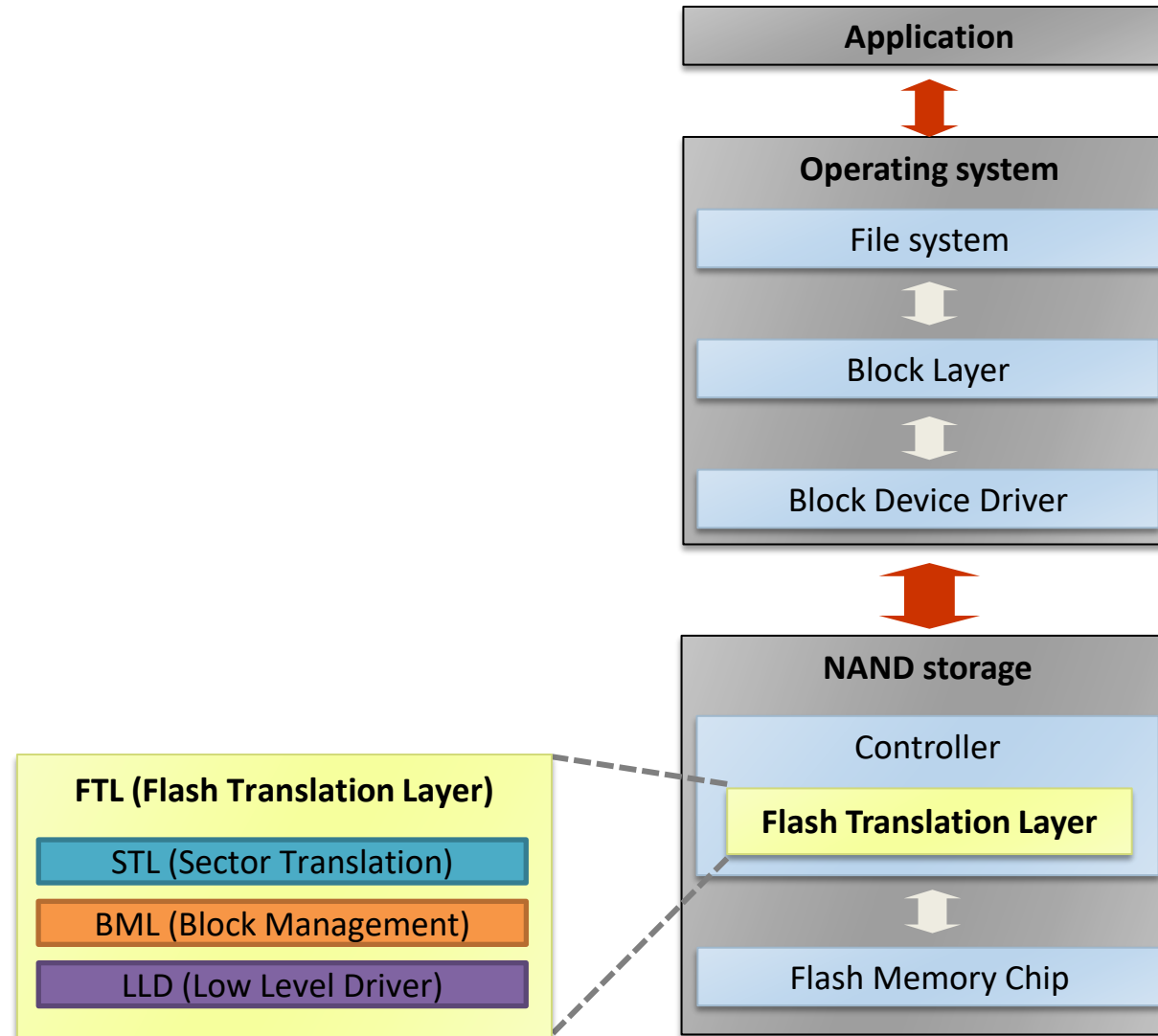
# Basic Firmware Architecture



S. H. Noh and Y.-S. Kee, *Flash Memory and Its By-product: A to Z in a Flash*, FAST Tutorial, 2015.

# FTL Architecture

- **Sector Translation Layer**
  - Address mapping
  - Garbage collection
  - Wear leveling
- **Block Management Layer**
  - Bad block management
  - Error handling
- **Low Level Driver**
  - Flash interface



# Performance Features

- Indirect mapping (address translation)
- Garbage collection
- Over-provisioning
- Hot/cold data identification/separation
- Exploiting parallelism over multiple channels/flash chips/planes
- Request scheduling of multiple commands
- Buffer management
- ...

# Reliability Features

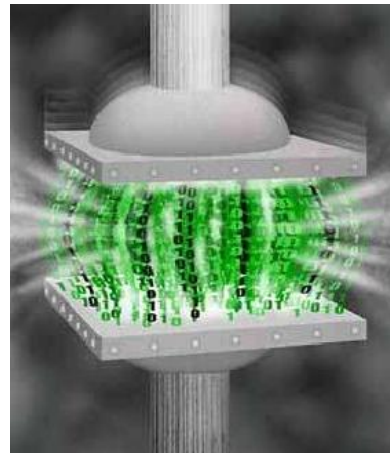
- Bad block management
- Wear leveling
- Power-off recovery
- Error detection and correction
- Countermeasures for cell characteristics
- ...

# Other Features

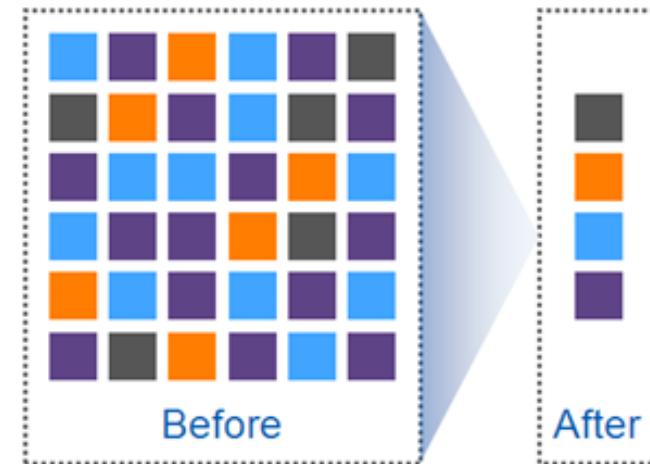
## Encryption



## Compression



## Deduplication



# Plethora of FTLs

SAST                      HFTL  
SFTL                      MS FTL                      BPLRU

BFTL                      AFTL                      FAST                      LazyFTL                      KAST

Chameleon                      CNFTL                      DFTL                      LAST                      MNFTL

super-block scheme                      CFTL                      Log block scheme

GFTL                       $\mu$ -FTL                      JFTL                      zFTL

Hydra FTL                      Vanilla FTL                      Replacement block scheme

Reconfigurable FTL                      YanusFTL

WAFTL                      UFTL                      .....and so on



# Mapping Schemes

- **Page mapping**
  - Fine-granularity page-level map table
  - High amount of memory space required for the map table
- **Block mapping**
  - Coarse-granularity block-level map table
  - Small amount of memory space required for the map table
- **\_\_\_\_\_ mapping**
  - Use both page-level and block-level map tables
  - Higher algorithm complexity

# Block Mapping FTL

# Block Mapping

## ■ Mapping in block-level

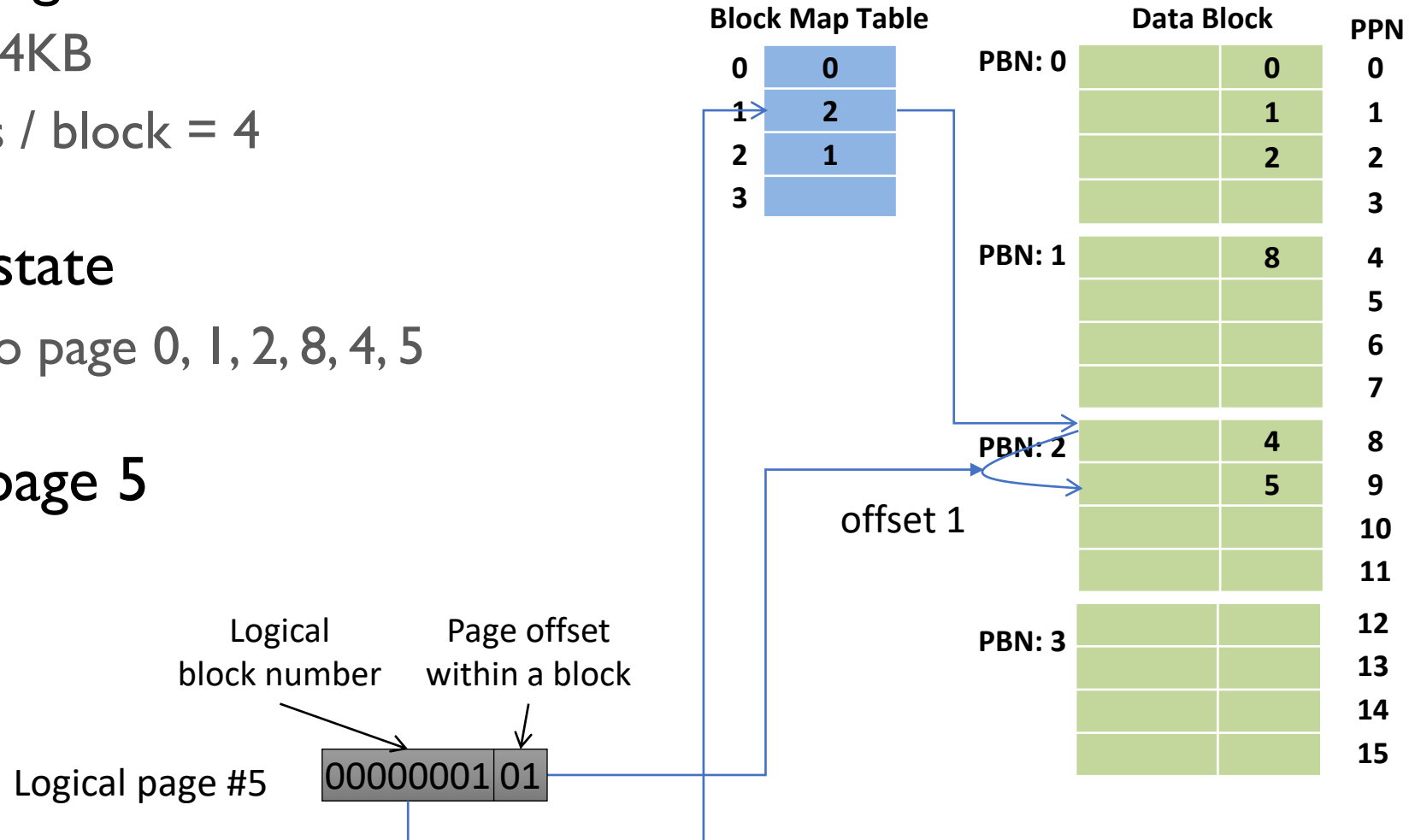
- Logical block number  $\rightarrow$  physical block number
- Block mapping table (BMT) required
- Page offset remains the same

## ■ Translation

- Step 1: logical sector number  $\rightarrow$  logical page number (LPN)
- Step 2: LPN  $\rightarrow$  logical block number (LBN) + page offset
  - $LBN = LPN / N$ , where  $N$ : # of pages in a block
  - Page offset =  $LPN \% N$
- Step 3: LBN  $\rightarrow$  physical block number (PBN) via BMT
  - Use the same page offset

# Example: Block Mapping

- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- Reading page 5



# Example: Block Mapping

- Flash configuration

- Page size: 4KB
- # of pages / block = 4

- Current state

- Written to page 0, 1, 2, 8, 4, 5

- New requests (in order)

- Write to page 9
- Write to page 3
- Write to page 5 (update)

Block Map Table

0	0
1	2
2	1
3	

	Data Block	PPN
PBN: 0	0	0
	1	1
	2	2
		3
PBN: 1	8	4
		5
		6
		7
PBN: 2	4	8
	5	9
		10
		11
PBN: 3		12
		13
		14
		15

# Example: Block Mapping

- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- New requests (in order)
  - Write to page 9
  - Write to page 3
  - Write to page 5 (update)

Block Map Table

0	0
1	2
2	1
3	

	Data Block	PPN
PBN: 0	0	0
	1	1
	2	2
		3
PBN: 1	8	4
	9	5
		6
		7
PBN: 2	4	8
	5	9
		10
		11
PBN: 3		12
		13
		14
		15

# Example: Block Mapping

- Flash configuration

- Page size: 4KB
- # of pages / block = 4

- Current state

- Written to page 0, 1, 2, 8, 4, 5

- New requests (in order)

- Write to page 9
- Write to page 3**
- Write to page 5 (update)

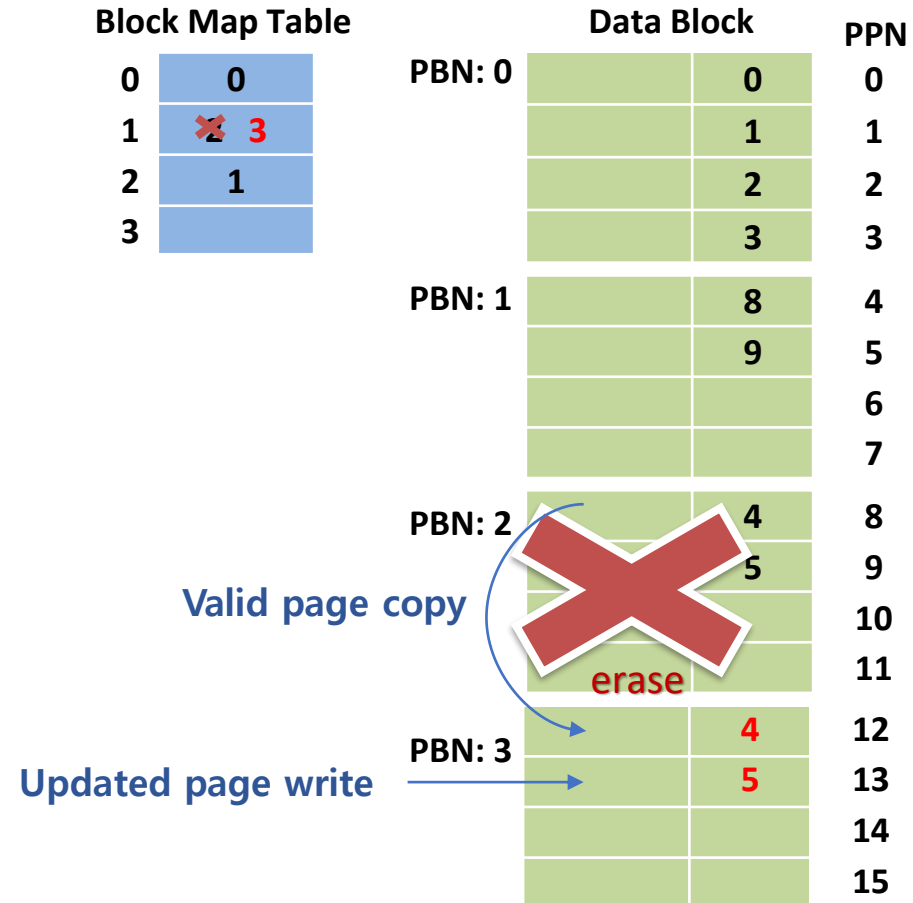
Block Map Table

0	0
1	2
2	1
3	

	Data Block	PPN
PBN: 0	0	0
	1	1
	2	2
	3	3
PBN: 1	8	4
	9	5
		6
		7
PBN: 2	4	8
	5	9
		10
		11
PBN: 3		12
		13
		14
		15

# Example: Block Mapping

- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- New requests (in order)
  - Write to page 9
  - Write to page 3
  - Write to page 5 (update)**



# Example: Block Mapping

- Current state
  - Written to page 0, 1, 2, 8, 4, 5
  - Written to page 9, 3, 5
- New requests (in order)
  - Write to page <0, 1, 2, 3>

Block Map Table

0	0
1	3
2	1
3	

	Data Block	PPN
PBN: 0	0	0
	1	1
	2	2
	3	3
PBN: 1	8	4
	9	5
		6
		7
PBN: 2		8
		9
		10
		11
PBN: 3	4	12
	5	13
		14
		15

# Example: Block Mapping

- Current state
  - Written to page 0, 1, 2, 8, 4, 5
  - Written to page 9, 3, 5
- New requests (in order)
  - Write to pages <0, 1, 2, 3>

Block Map Table

0	<del>0</del> 2
1	3
2	1
3	

	Data Block	PPN
PBN: 0	<del>0</del>	0
	<del>1</del>	1
	<del>2</del>	2
	<del>3</del> erase	3
PBN: 1	8	4
	9	5
		6
		7
PBN: 2	0	8
	1	9
	2	10
	3	11
PBN: 3	4	12
	5	13
		14
		15

# Block Mapping

## ■ Pros

- Small RAM usage
  - One mapping entry per block
- Good performance for sequential writes

## ■ Cons

- Inefficient handling of small random writes
  - Even a single page update requires a block copy & erase

# Page Mapping FTL

# Page Mapping

- Mapping in page-level
  - Logical page number  $\rightarrow$  physical page number
  - Page mapping table (PMT) required
  - # entries in PMT == # pages visible to OS
- Translation
  - Step 1: logical sector number (LSN)  $\rightarrow$  logical page number (LPN)
  - Step 2: LPN  $\rightarrow$  physical page number (PPN) via PMT

# Example: Page Mapping

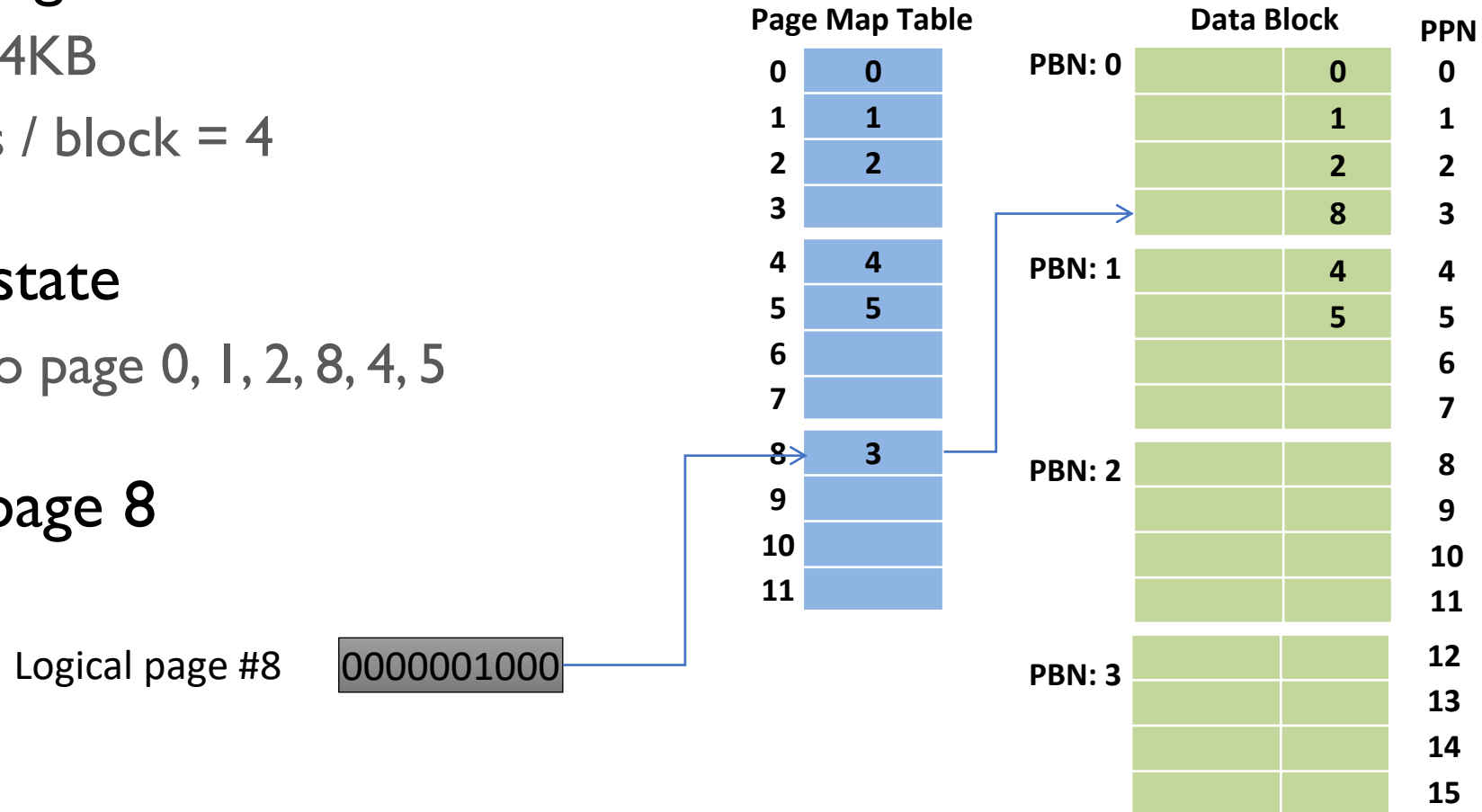
- Flash configuration

- Page size: 4KB
- # of pages / block = 4

- Current state

- Written to page 0, 1, 2, 8, 4, 5

- Reading page 8



# Example: Page Mapping

- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- New requests (in order)
  - Write to page 9
  - Write to page 3
  - Write to page 5

Page Map Table		Data Block		PPN
0	0	PBN: 0	0	0
1	1		1	1
2	2		2	2
3			8	3
4	4	PBN: 1	4	4
5	5		5	5
6				6
7				7
8	3	PBN: 2		8
9				9
10				10
11				11
		PBN: 3		12
				13
				14
				15

# Example: Page Mapping

- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- New requests (in order)
  - Write to page 9
  - Write to page 3
  - Write to page 5

Page Map Table		Data Block		PPN
0	0	PBN: 0	0	0
1	1		1	1
2	2		2	2
3			8	3
4	4	PBN: 1	4	4
5	5		5	5
6			9	6
7				7
8	3	PBN: 2		8
9	6			9
10				10
11				11
		PBN: 3		12
				13
				14
				15

# Example: Page Mapping

- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- New requests (in order)
  - Write to page 9
  - Write to page 3**
  - Write to page 5

Page Map Table

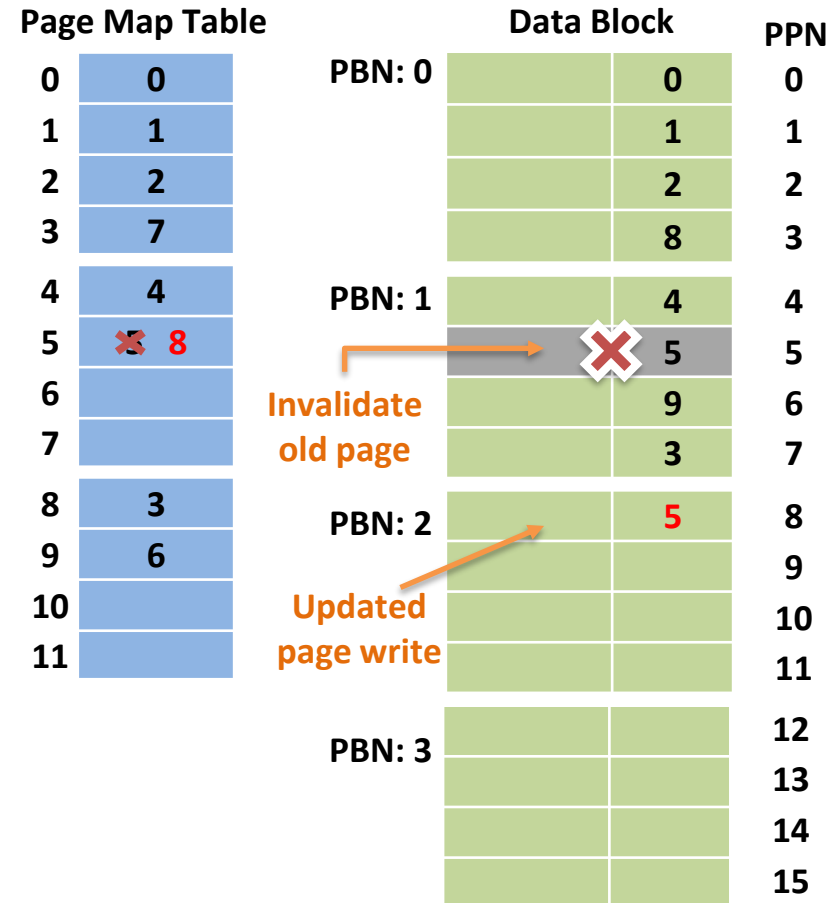
0	0
1	1
2	2
3	7
4	4
5	5
6	
7	
8	3
9	6
10	
11	

Data Block

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

# Example: Page Mapping

- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- New requests (in order)
  - Write to page 9
  - Write to page 3
  - Write to page 5**



# Page Mapping

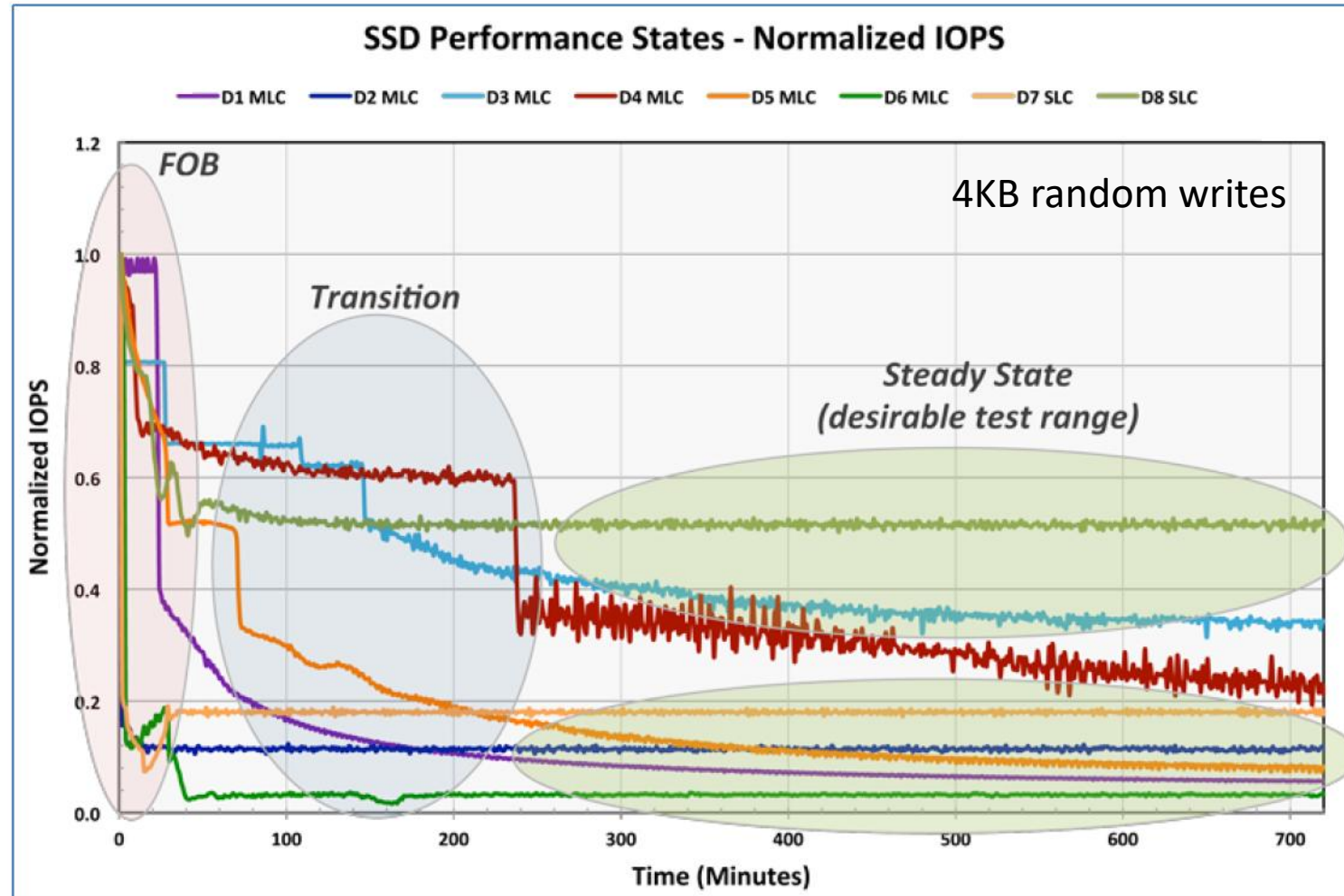
## ■ Pros

- Most flexible
- Efficient handling of small random writes
  - A logical page can be located anywhere within the flash storage
  - Updated page can be written to any free page

## ■ Cons

- Large memory footprint
  - One page mapping entry per page
  - 32MB for 32GB (4KB page)
- Sensitive to the amount of reserved blocks
- Performance affected as the system ages

# Why?



# Garbage Collection

- **Garbage collection (GC)**
  - Eventually, FTL will run out of blocks to write to
  - GC must be performed to reclaim free space
  - Actual GC procedure depends on the mapping scheme
  
- **GC in page-mapping FTL**
  - Select victim block(s)
  - Copy all valid pages of victim block(s) to free block
  - Erase victim block(s)
  - Note: At least one free block should be reserved for GC

# Example: GC in Page Mapping

## ■ Current state

- Written to page 0, 1, 2, 8, 4, 5
- Written to page 9, 3, 5

## ■ New requests (in order)

- Write to page 8
- Write to page 9
- Write to page 3
- Write to page 1
- Write to page 4

Page Map Table		Data Block		PPN
0	0	PBN: 0	0	0
1	1		1	1
2	2		2	2
3	7		8	3
4	4	PBN: 1	4	4
5	8		5	5
6			9	6
7			3	7
8	3	PBN: 2	5	8
9	6			9
10				10
11				11
		PBN: 3		12
				13
			Spare block	14
				15

# Example: GC in Page Mapping

## ■ Current state

- Written to page 0, 1, 2, 8, 4, 5
- Written to page 9, 3, 5

## ■ New requests (in order)

- Write to page 8
- Write to page 9
- Write to page 3
- Write to page 1
- Write to page 4

Page Map Table		Data Block		PPN
0	0	PBN: 0	0	0
1	1		1	1
2	2		2	2
3	7		X 8	3
4	4	PBN: 1	4	4
5	8		X 5	5
6			9	6
7			3	7
8	9	PBN: 2	5	8
9	6		8	9
10				10
11				11
		PBN: 3		12
			Spare block	13
				14
				15

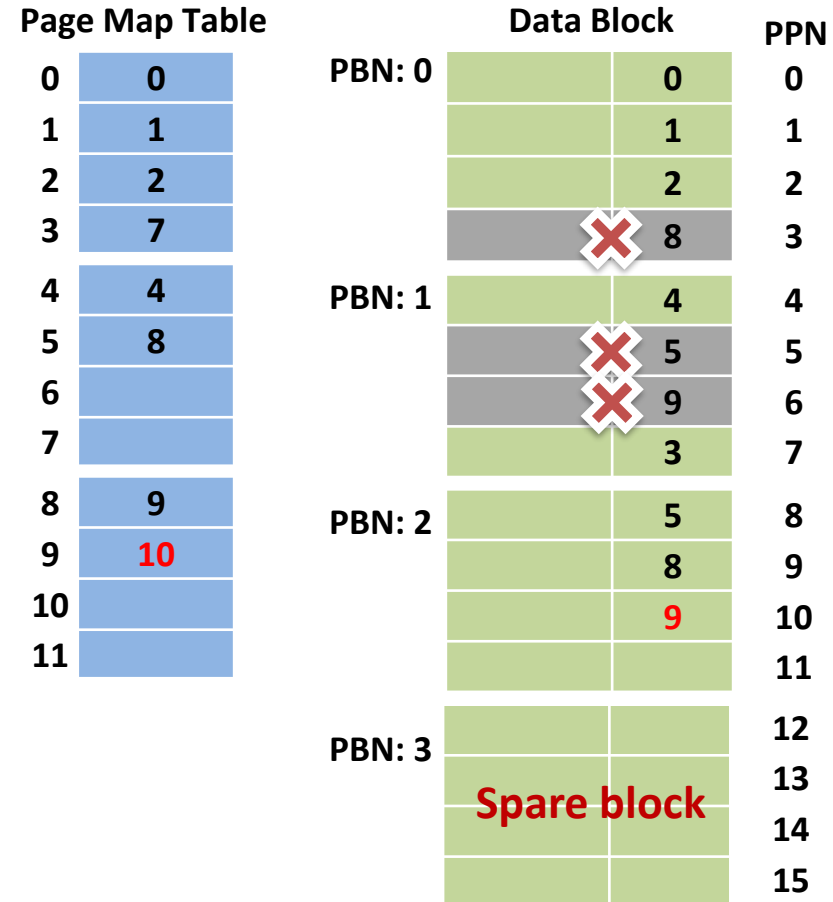
# Example: GC in Page Mapping

## ■ Current state

- Written to page 0, 1, 2, 8, 4, 5
- Written to page 9, 3, 5

## ■ New requests (in order)

- Write to page 8
- **Write to page 9**
- Write to page 3
- Write to page 1
- Write to page 4



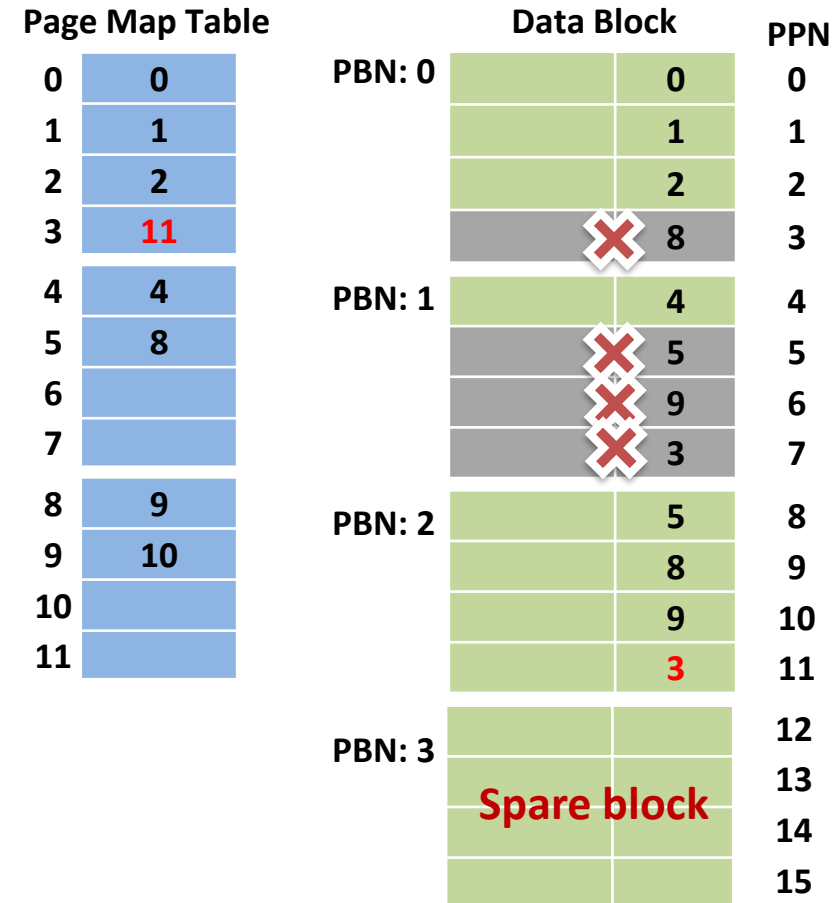
# Example: GC in Page Mapping

## ■ Current state

- Written to page 0, 1, 2, 8, 4, 5
- Written to page 9, 3, 5

## ■ New requests (in order)

- Write to page 8
- Write to page 9
- **Write to page 3**
- Write to page 1
- Write to page 4



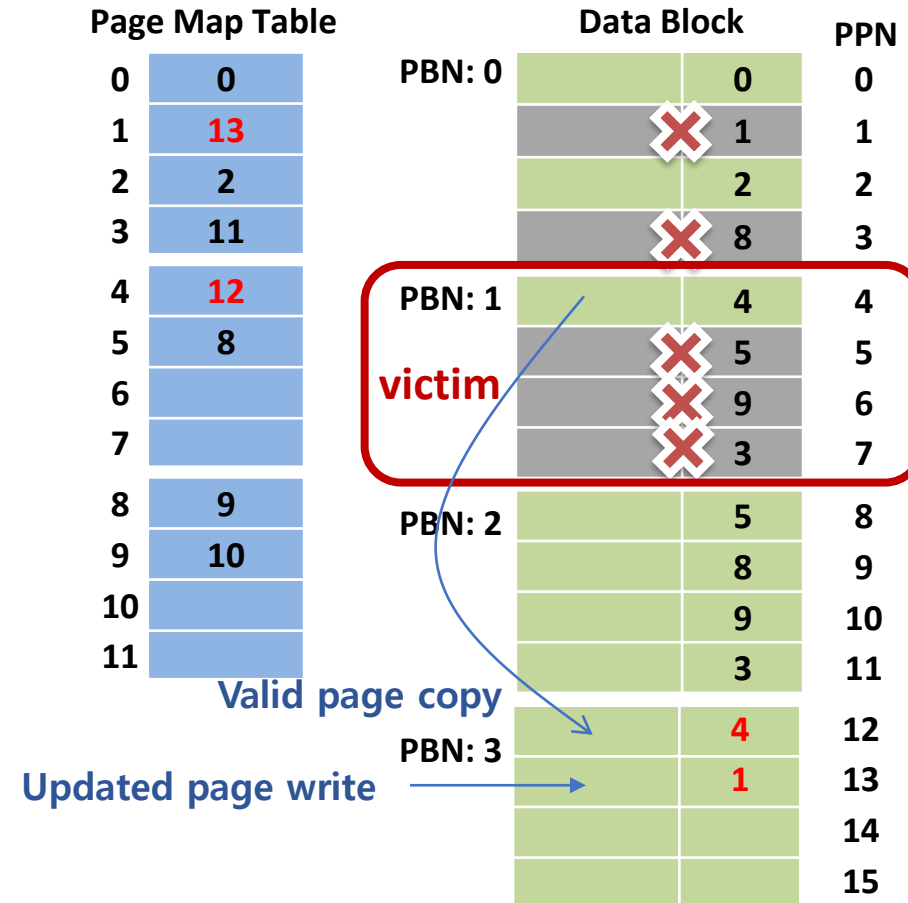
# Example: GC in Page Mapping

## ■ Current state

- Written to page 0, 1, 2, 8, 4, 5
- Written to page 9, 3, 5

## ■ New requests (in order)

- Write to page 8
- Write to page 9
- Write to page 3
- **Write to page 1**
- Write to page 4



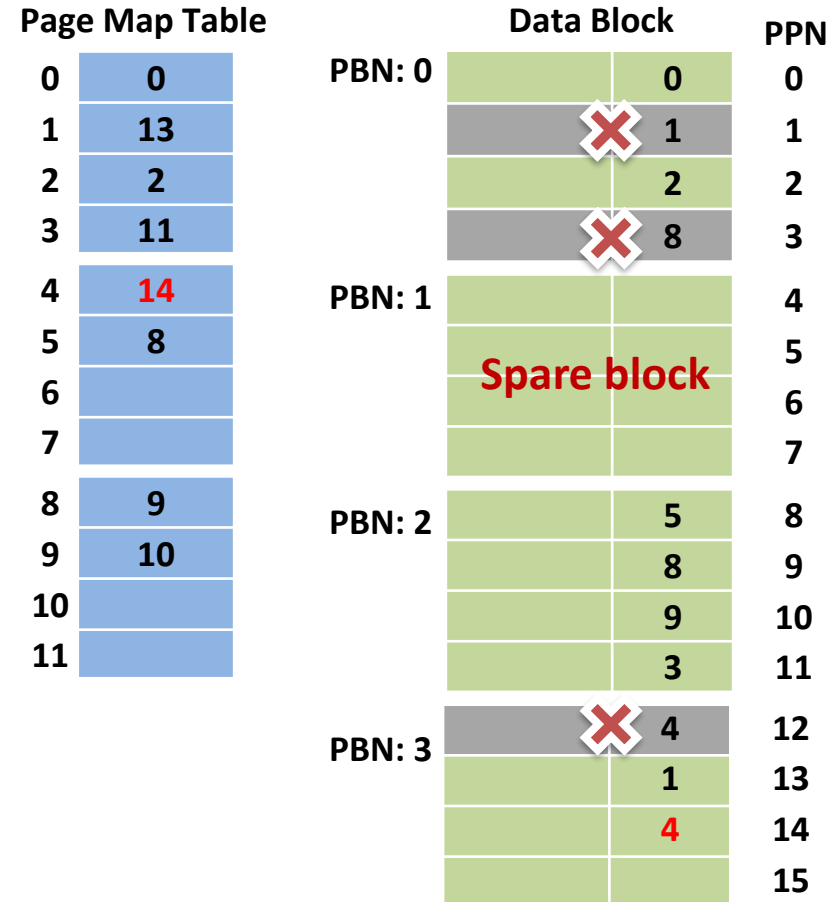
# Example: GC in Page Mapping

## ■ Current state

- Written to page 0, 1, 2, 8, 4, 5
- Written to page 9, 3, 5

## ■ New requests (in order)

- Write to page 8
- Write to page 9
- Write to page 3
- Write to page 1
- **Write to page 4**



# Write Amplification

- Ratio of data written to flash to data written from host

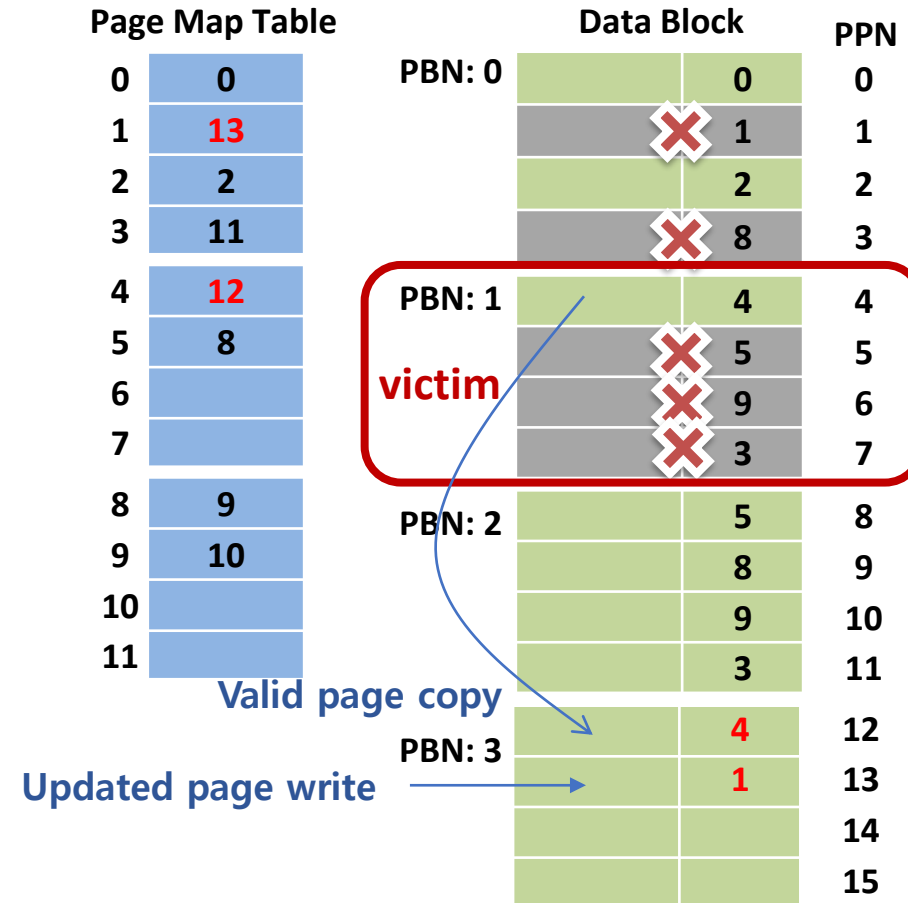
- Write Amplification Factor (WAF)

$$= \frac{\text{Bytes written to *Flash*}}{\text{Bytes written from *Host*}} = \frac{\text{Bytes written from *Host* + Bytes written during *GC*}}{\text{Bytes written from *Host*}}$$

- Generally, WAF is greater than one in flash storage
  - Due to valid page copies made from victim block to free block during GC
  - WAF is one of metrics that shows the efficiency of GC

# Example: Write Amplification

- Current state
  - Written to page 0, 1, 2, 8, 4, 5
  - Written to page 9, 3, 5
- New requests (in order)
  - Write to page 8, 9, 3, 1
- WAF = 1.08
  - Total host writes: 13
  - Total flash writes: 14



# Victim Selection Policy: Greedy

- Selects a block with the largest amount of invalid data
- A block with the \_\_\_\_\_ utilization  $u$

$$u = \frac{\text{Number of valid pages in a block}}{\text{Number of Pages in a block}}$$

- **Pros?**

- **Cons?**

# Victim Selection Policy: Cost-Benefit

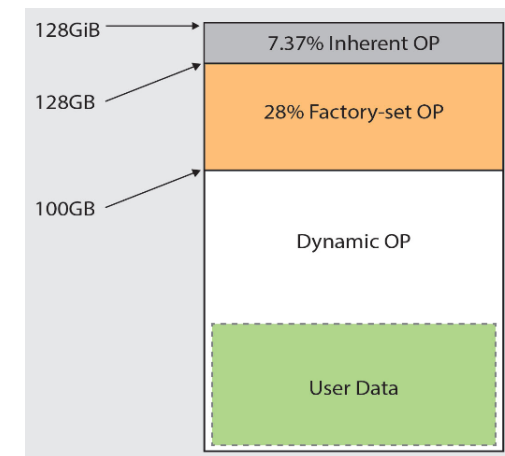
- Selects a block with the \_\_\_\_\_

$$\frac{Benefit}{Cost} = \frac{(1 - u)}{2u} \times age$$

- $u$ : utilization
  - $age$ : the time since the last modification
- **Pros?**
  - **Cons?**

# Over-Provisioning

- OP (Over-Provisioning) =  $\frac{\text{Physical Capacity}}{\text{Logical Capacity}} - 1$ 
  - Extra media space on an SSD that does not contain user data
- Typical SSDs have more space than is advertised
  - Consumer SSDs: ~ 7%
    - 1 Gigabyte (GB) =  $10^9$  bytes = 1,000,000,000 bytes
    - 1 Gibibyte (GiB) =  $2^{30}$  bytes = 1,073,741,824 bytes
  - Enterprise SSDs: > 25%
    - 100GB user space on 128GiB SSD: 37.4%



# Example: Over-Provisioning

- OP = 33%
- Logical capacity: 3 blocks
- Physical capacity: 4 blocks

Page Map Table

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

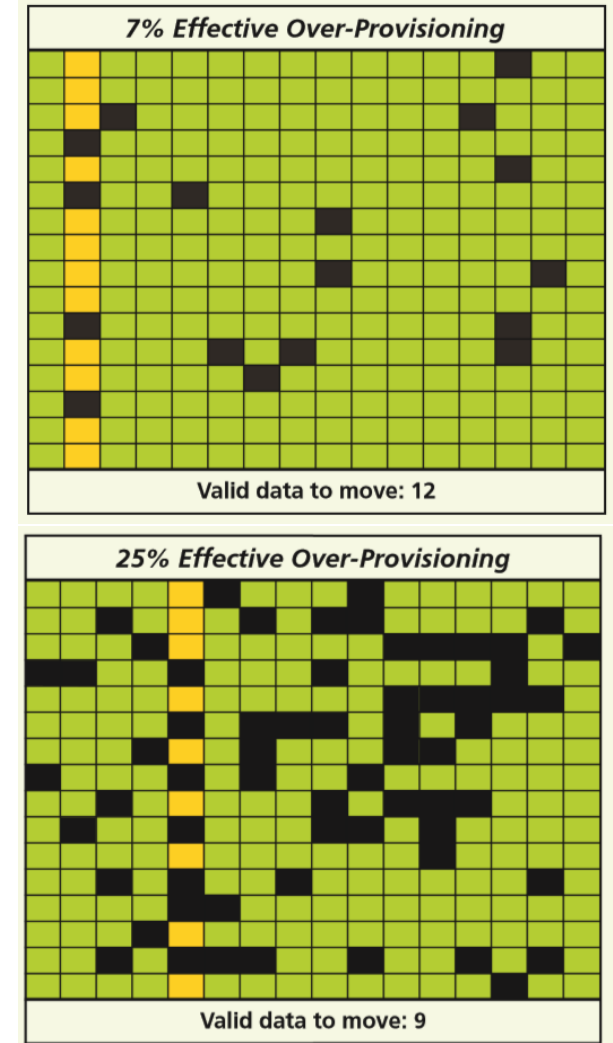
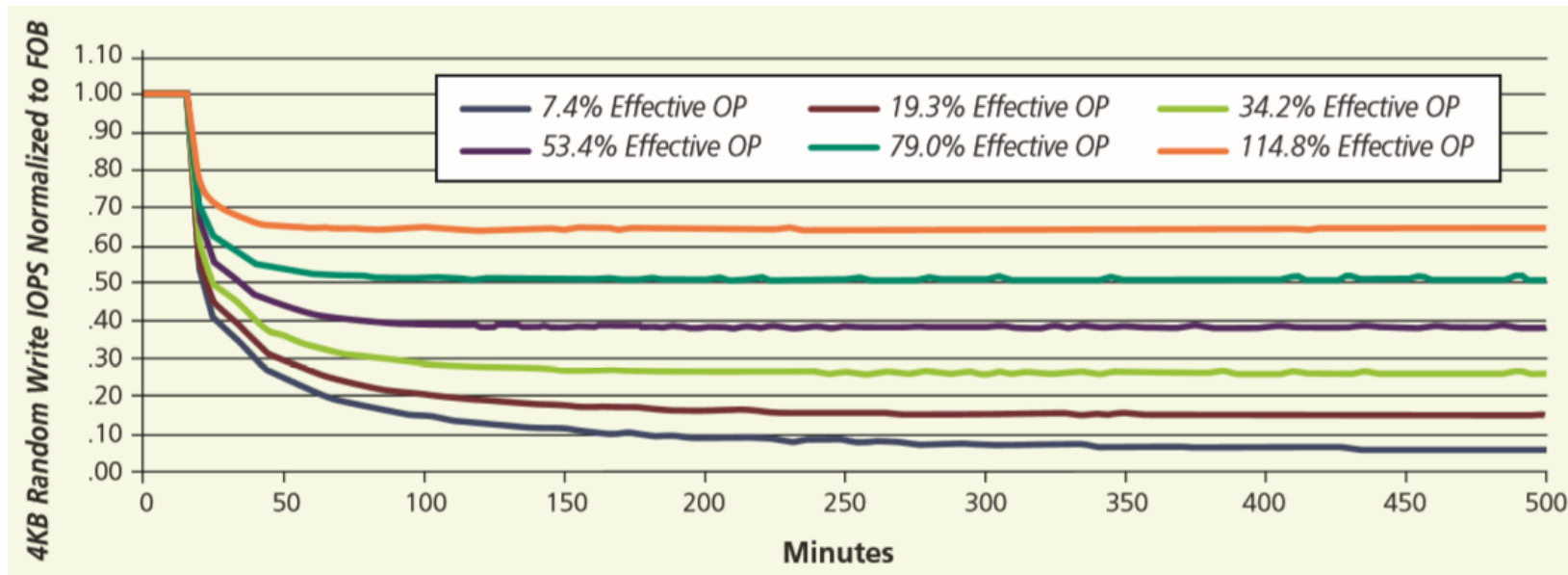
	Data Block	PPN
PBN: 0	0	0
	<del>1</del>	1
	2	2
	<del>8</del>	3
PBN: 1		4
	Spare block	5
		6
		7
PBN: 2	5	8
	8	9
	9	10
	3	11
PBN: 3	<del>4</del>	12
	1	13
	4	14
		15

# Why Over-Provisioning?

- Over-Provisioning Space (OPS) is used for
  - Write buffers
  - \_\_\_\_\_
  - \_\_\_\_\_
  - \_\_\_\_\_
- Garbage collection cost
  - Affected by utilization of SSD space and Over-Provisioning
  - \_\_\_\_\_ utilization → Better performance
  - \_\_\_\_\_ OP → Better performance

# Over-Provisioning and GC

- IOPS for random write workloads
  - What about for sequential write workloads?



# Hybrid Mapping FTL

# Hybrid Mapping

## ■ Goals

- Small DRAM footprint
- Efficient handling of small random writes

## ■ Basic idea

- Basic mapping scheme: block mapping
- A small number of page-mapped log blocks
  - Updated pages are just written to the log block to avoid erase-before-write
  - When there remains no free log block, victim log block is merged with data block to reclaim a log block
  - Log blocks can be regarded as buffers for updated (hot) pages

# Hybrid Mapping

- **Memory requirement**
  - Most of blocks are block-mapped (data blocks)
  - Only a small fixed number of blocks (log blocks) are page-mapped
- **Random write performance**
  - Random writes are accommodated in the page-mapped log blocks
- **Many schemes proposed so far**
  - BAST, FAST, Superblock, LAST, KAST, SAST, ...

# Design Considerations

- Relationship between data block and log block
  - 1 log block : 1 data block ( $\approx$  direct-mapped)
    - Only one log block is dedicated to a single data block
    - One log block is used to accommodate updates from the corresponding data block
  - 1 log block : N data blocks ( $\approx$  set-associative)
    - One log block is shared among a set of data blocks
  - M log blocks : N data blocks ( $\approx$  fully-associative)
    - All data blocks share all log blocks
    - One log block can accommodate updates from different data blocks
    - Updates from one data block can scatter over different log blocks

# A Space-efficient Flash Translation Layer for Compact Flash Systems

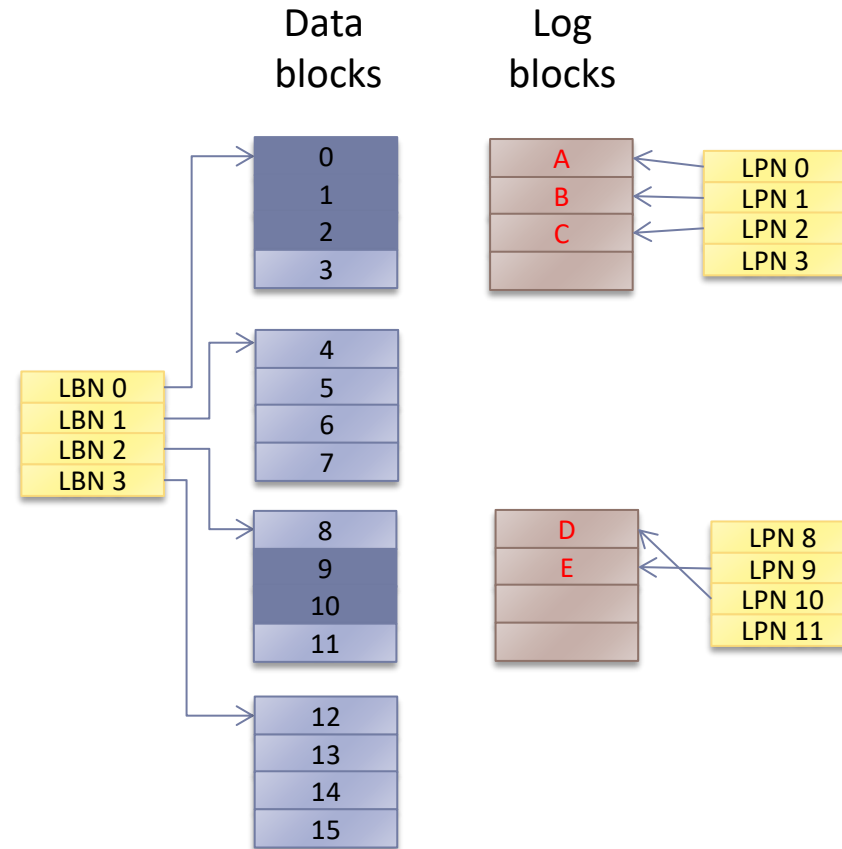
(J. Kim et al., IEEE Trans. on Consumer Electronics, 2002.)

# BAST

- One of the early versions of hybrid mapping
- Key ideas
  - Provide log blocks for storing updates
    - Data blocks are managed in block mapping
    - Log blocks are managed in page mapping
  - 1-to-1 mapping between log block and data block
    - One log block is dedicated to a single data block
  - Mix fine- and coarse-grained management schemes
- Homogeneous log blocks
  - Each log block is the same in type and role (see FAST for a different example)

# Data vs. Log Blocks

- Data block
  - Block-level mapping
- Log block
  - Page-level mapping
  - Temporary block associated with the data block

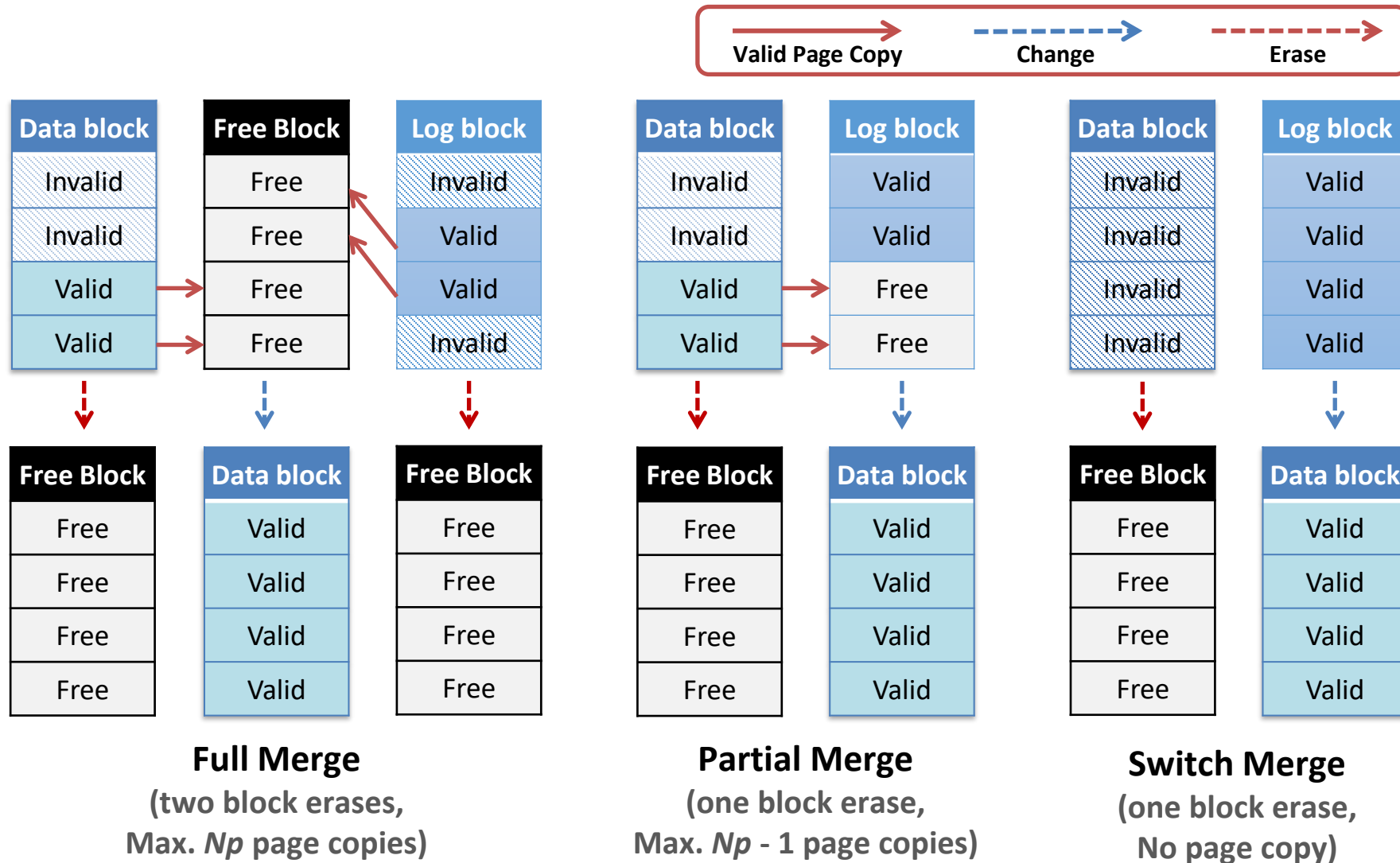


$$W = \langle \{0\}, \{1,2\}, \{10\}, \{9\} \rangle$$

# Block Merge

- Block merge happens when
  - All the pages of a log block are overwritten
  - No available log block to allocate to a data block
- Types of block merge
  - Switch / Partial / Full
  - Cost: Switch < Partial < Full
  - System maintains at least one free block for full merge operation

# Types of Block Merges



# Summary: BAST

## ■ Pros

- Better performance than pure block-mapping scheme
- Reduced resource requirements for page mapping management

## ■ Cons

- Low utilization of log blocks
  - Log block utilization =  $(\# \text{ of written pages}) / (\# \text{ of pages in log block})$
  - If the write pattern is random, log block is frequently merged despite its low utilization

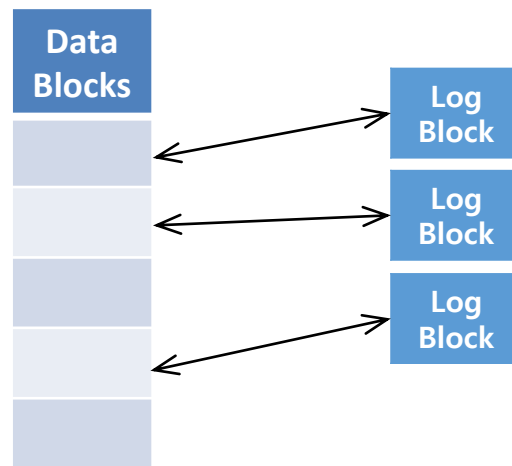
# A Log-buffer based Flash Translation Layer using Fully- Associative Sector Translation

(S.W. Lee et al., ACM Trans. on Embedded Computing Systems, 2007.)

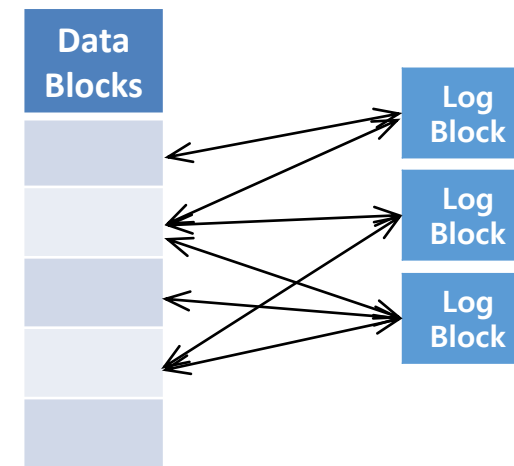
# FAST

## ■ Fully-Associative Sector Translation

- Overcome the problems of BAST
  - Alleviate log block thrashing
  - Avoid frequent merge operations
- N-to-M mapping between data blocks and log blocks
  - Fully associative approach in mapping logical pages onto log blocks



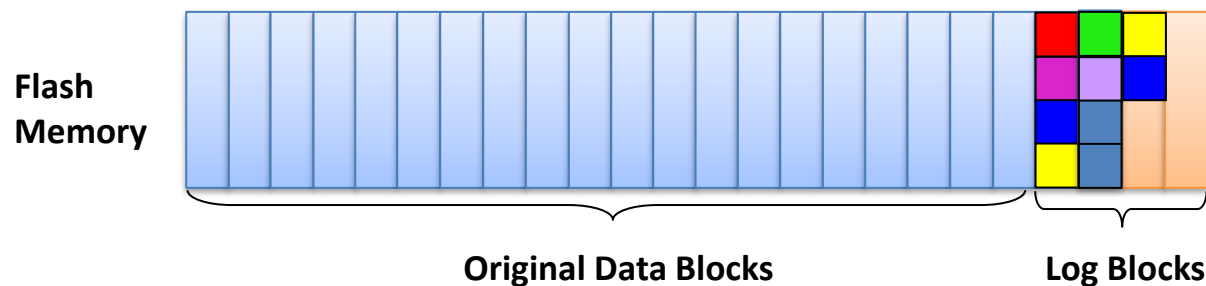
**BAST (1:1 mapping)**



**FAST (N:M mapping)**

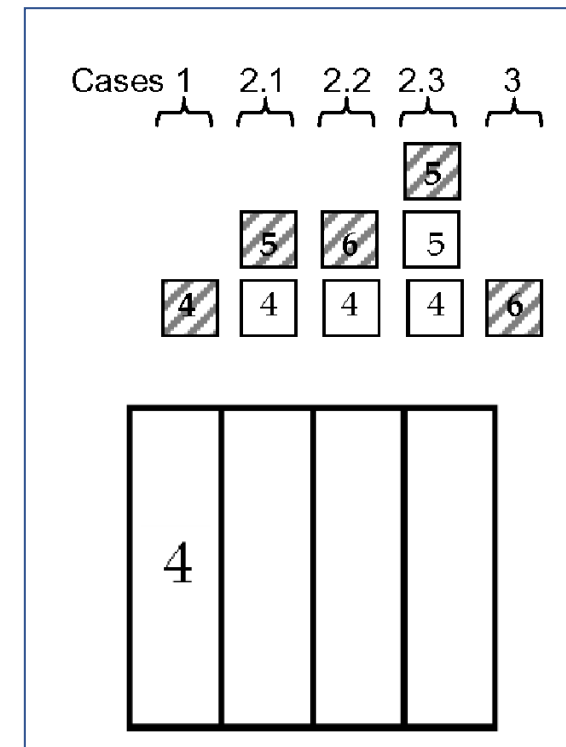
# FAST Architecture

- Two types of log blocks
  - SW: Sequential Write log block (just one)
    - To increase the chance of switch merge for sequential writes
  - RW: Random Write log block
- Mapping table
  - Block-level mapping table (for data blocks)
  - Page-level mapping table (for SW)
  - Page-level mapping table (for RW)



# Handling Writes

- Written to SW log block if
  - $LPN \bmod Np == 0$  (LPN is at the beginning of a block) or
  - LPN immediately follows the pages written in SW log block
- Cases 1 & 2.1:
  - Inserted into SW log block
- Cases 2.2 & 2.3:
  - Merge SW log block with its data block
- Case 3:
  - Inserted into RW log block
  - Select victim in round-robin fashion if needed



# Summary: FAST

## ■ Pros

- Improved log block utilization
- Avoid unnecessary merge

## ■ Cons

- Increased merge time
- Detecting sequential writes is hard
- Looking up the page-level mapping table for RW log blocks