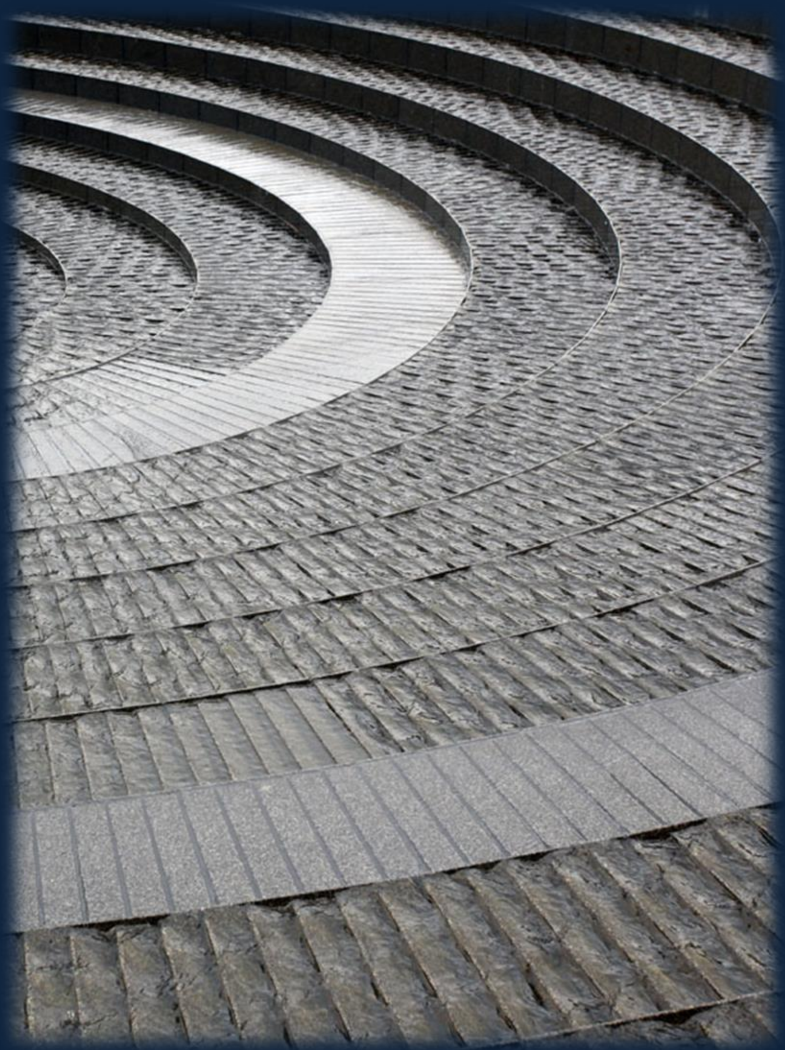


Flash Translation Layers I

Jin-Soo Kim (*jinsookim@skku.edu*)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Storage Abstraction

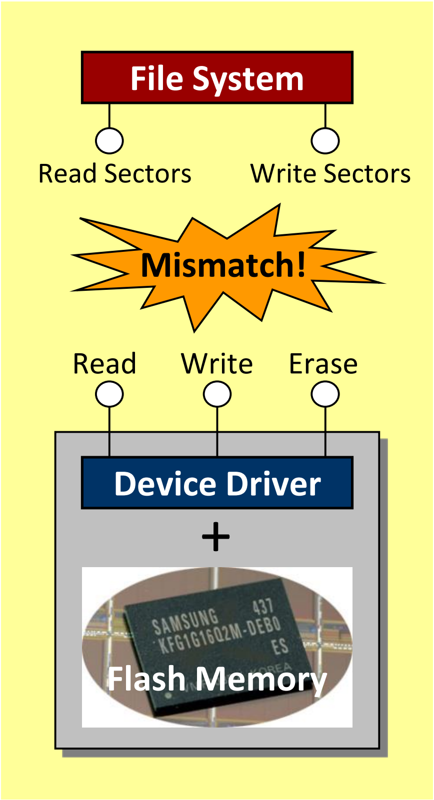
- Abstraction given by block device drivers:

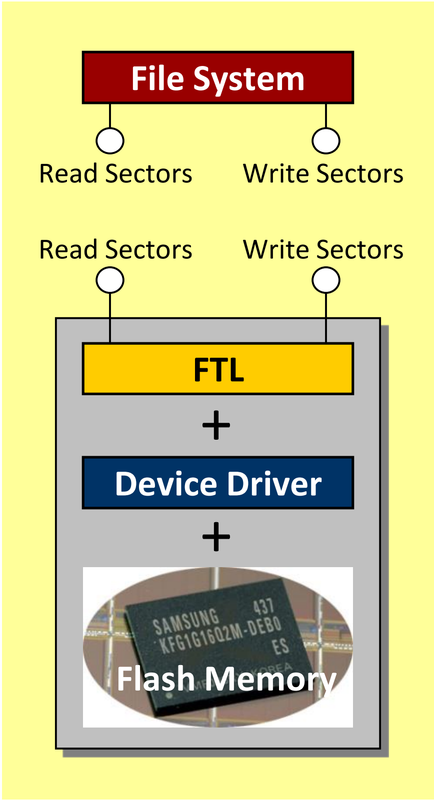
- Operations
  - Identify(): returns N
  - Read (start sector #, # of sectors)
  - Write (start sector #, # of sectors, data)

Source: Sang Lyul Min (Seoul National University)

What is FTL?

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





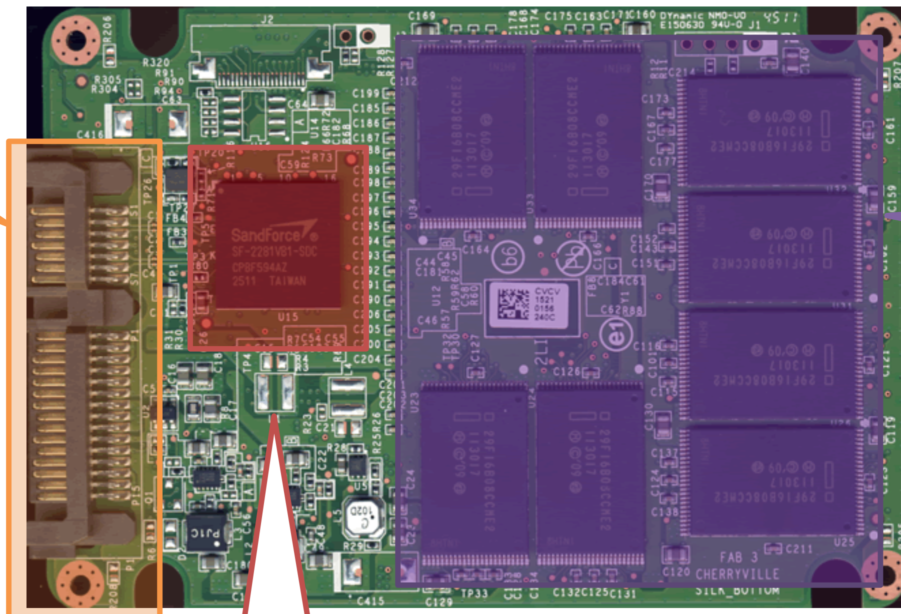
Source: Zeen Info. Tech.

Major Components in SSD

- Similar in most NAND storage systems

1. Host Interface

SATA,
PCIe,
UFS,
eMMC,
etc.



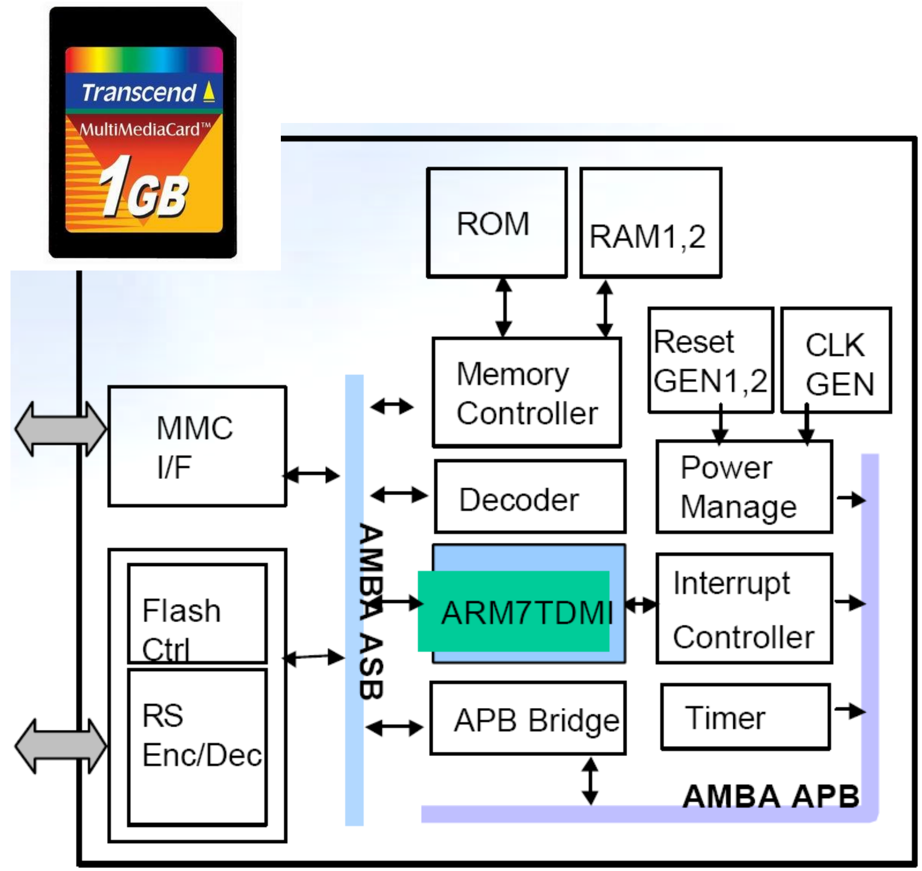
2. Storage Controller

Running FTL

3. NAND Array

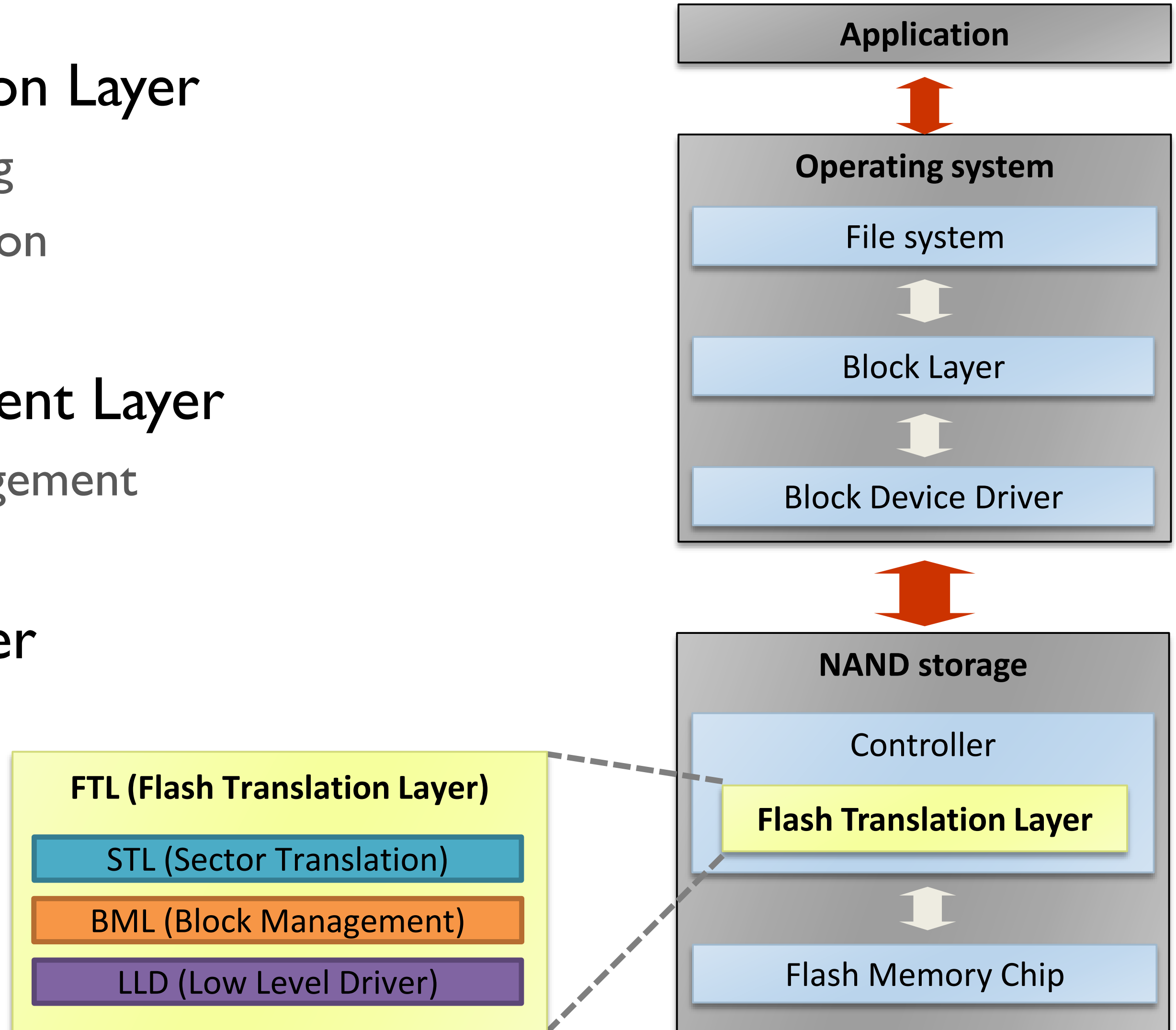
Data Storage

Flash Cards Internals

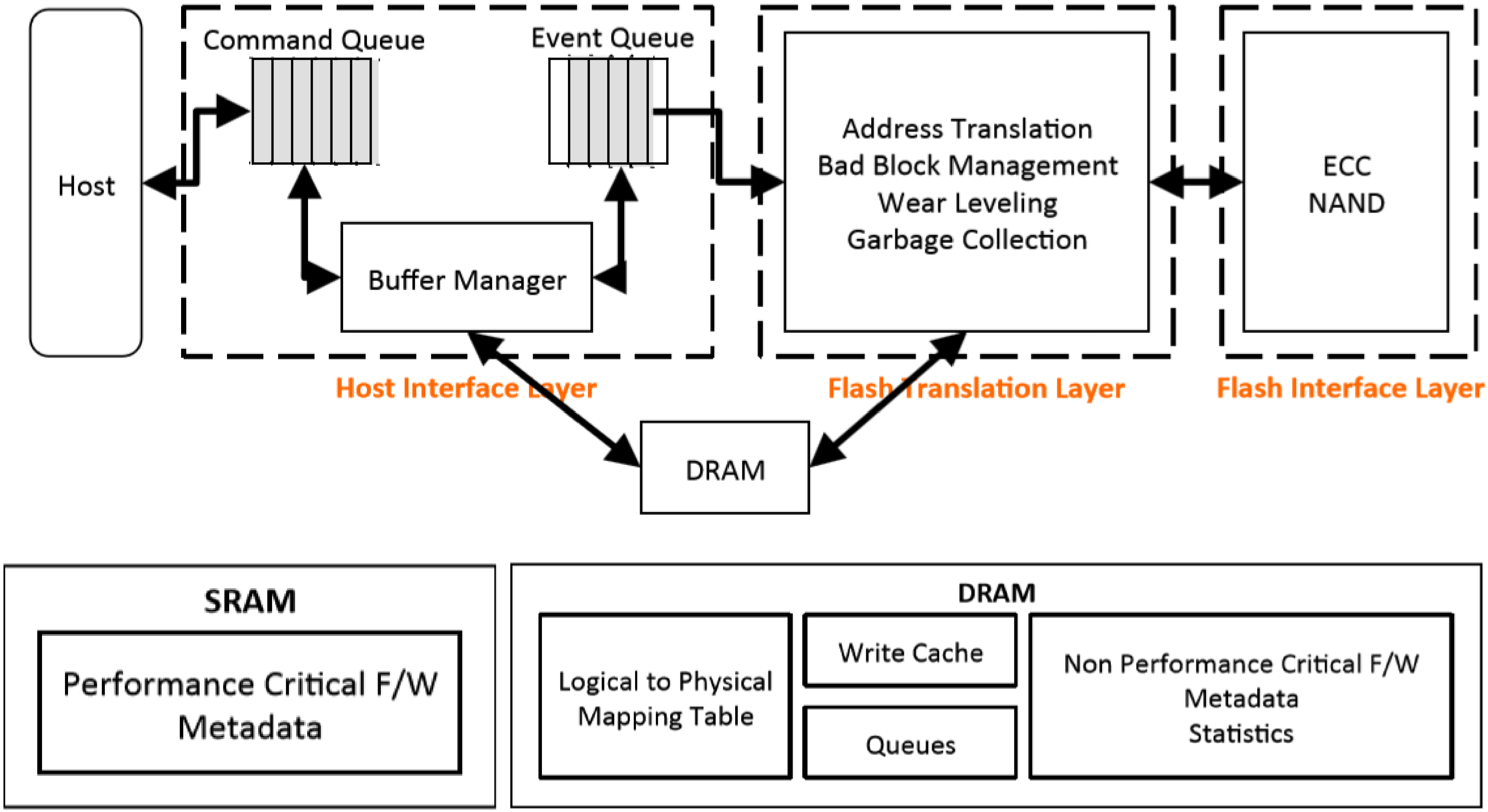


FTL Architecture

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

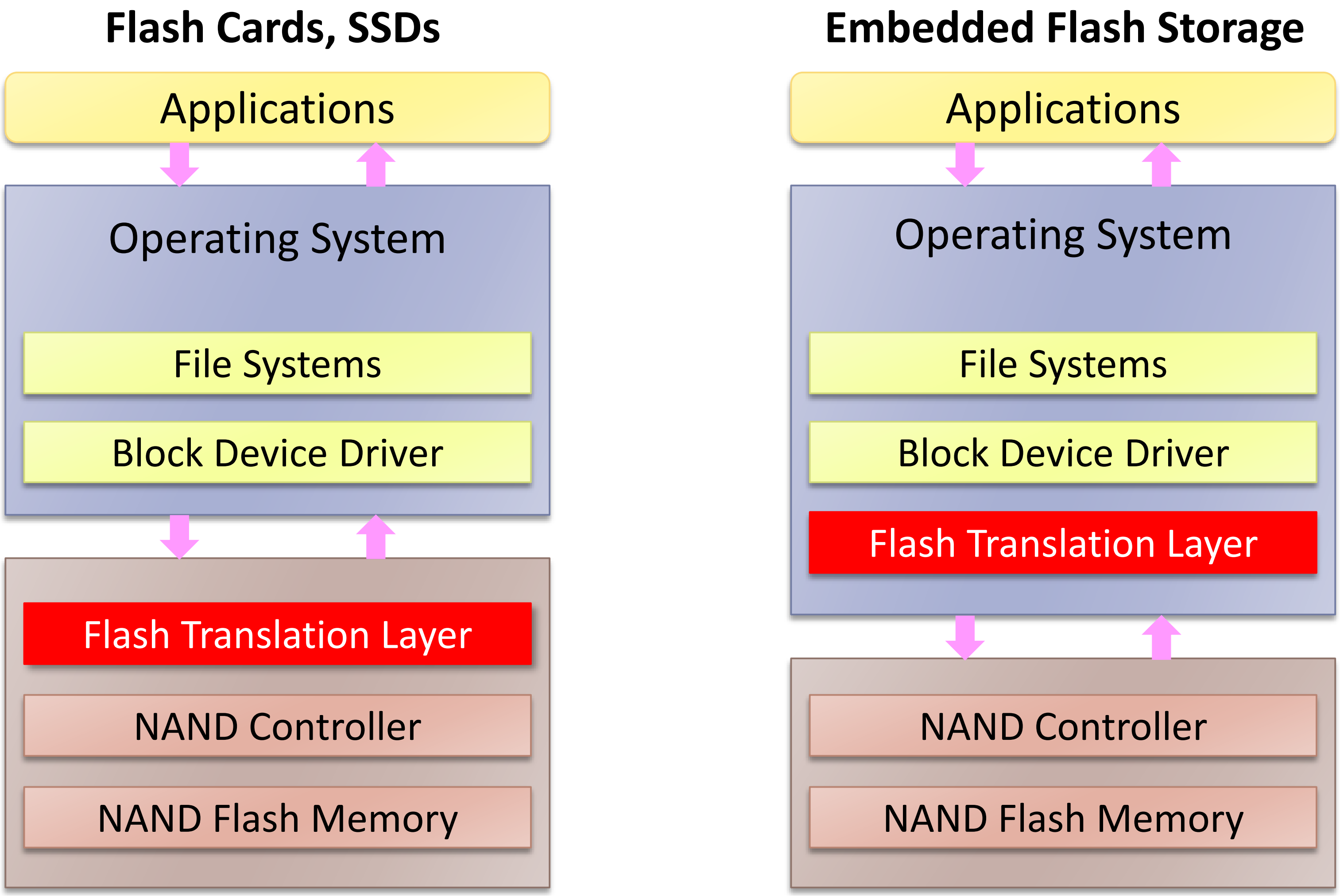


Basic Firmware Architecture



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

Implementing FTLs



Plethora of FTLs

HFTL
SAST
SFTL
MS FTL
BPLRU
BFTL
AFTL
FAST
LazyFTL
KAST
Chameleon
CNFTL
DFTL
LAST
MNFTL
super-block scheme
CFTL
Log block scheme
GFTL
μ-FTL
JFTL
zFTL
Hydra FTL
Vanilla FTL
Replacement block scheme
YanusFTL
Reconfigurable FTL
..........and so on
WAFTL
UFTL

E. H. Nam, HIL: FTL Design Framework with Provably-correct Crash Recovery, NVRAMOS, 2013.

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
- …

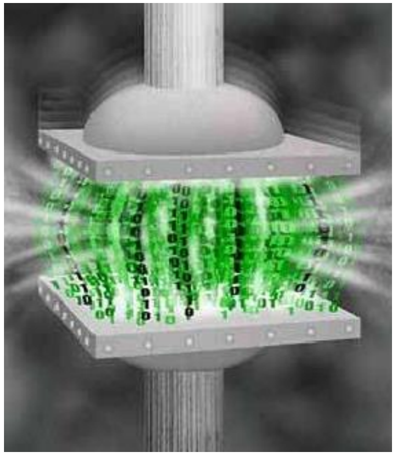
Reliability Features

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

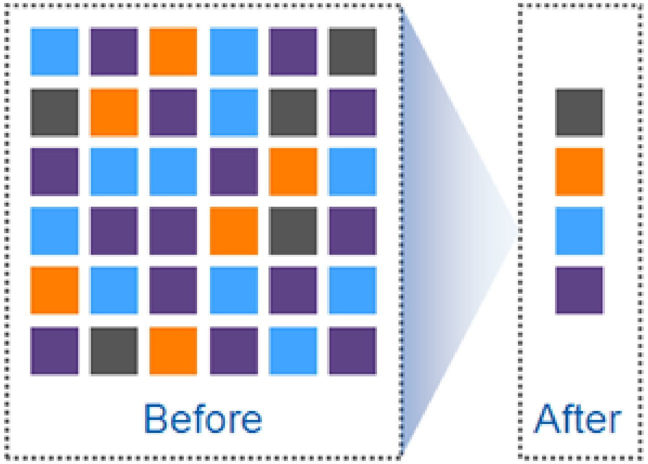
Other Features

Encryption

Compression

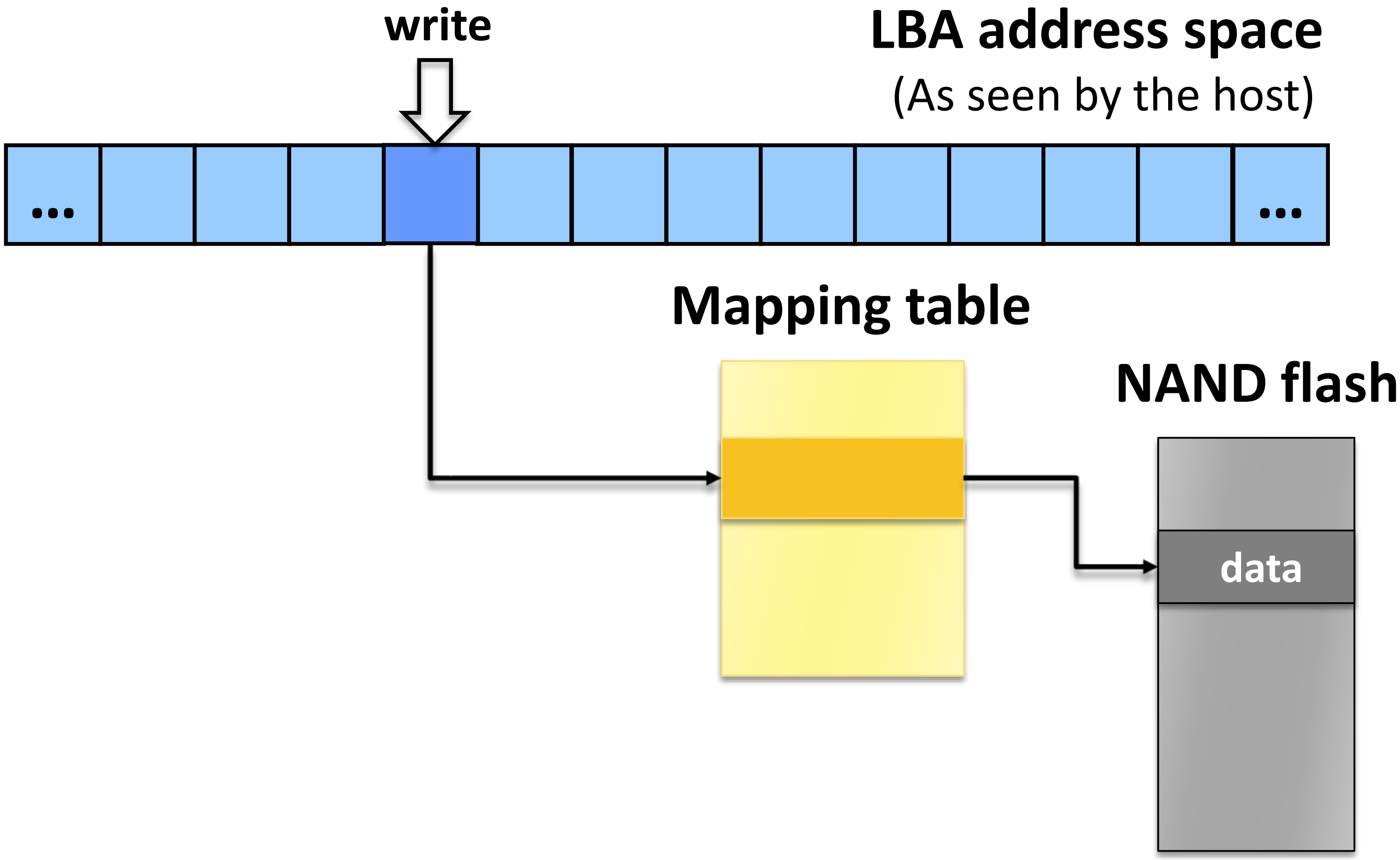


Deduplication



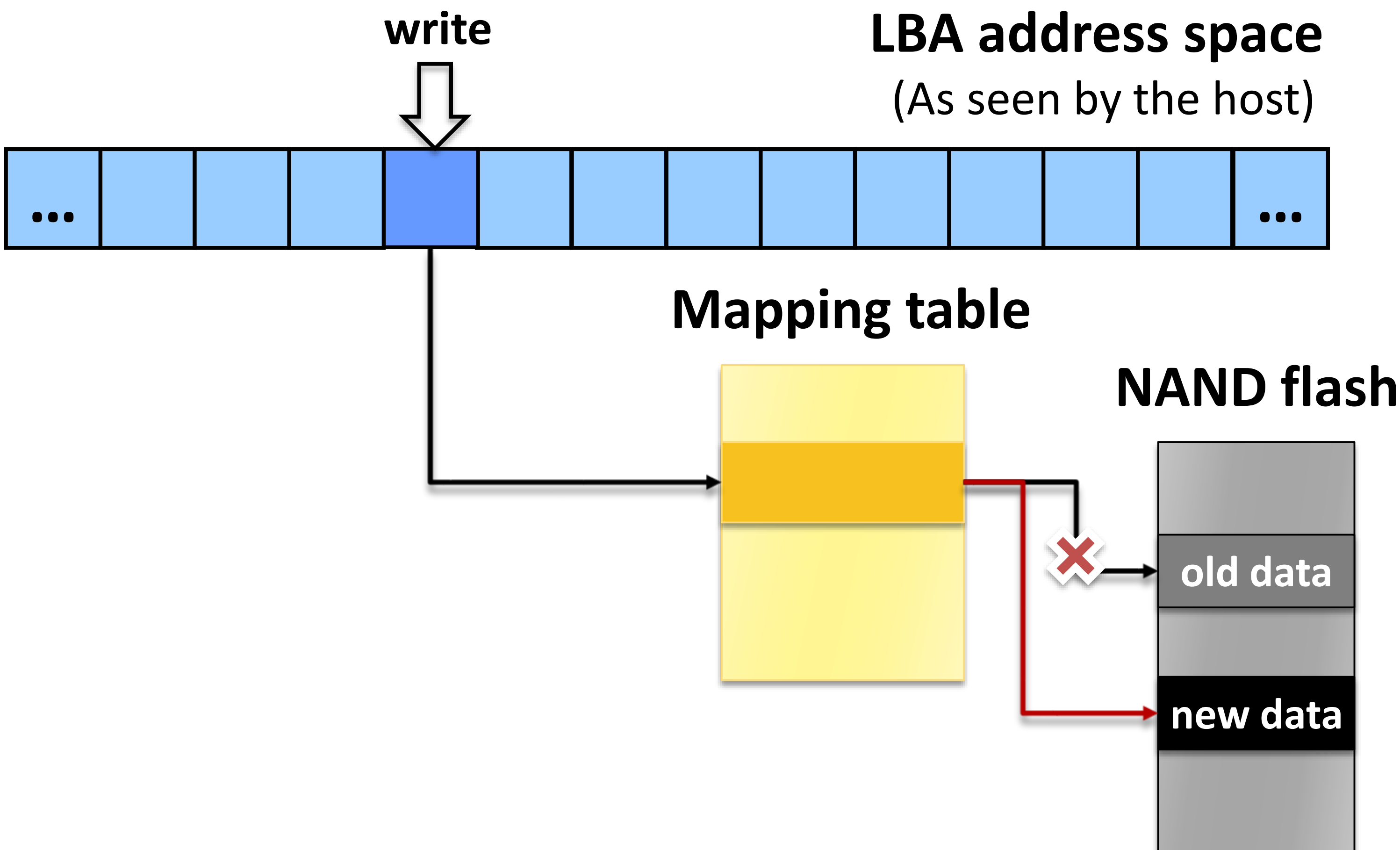
Page Mapping

Address Mapping



Address Mapping

- Required due to "no overwrite"



Mapping Schemes

- Page mapping
  - Fine-granularity page-level map table
  - Hugh 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
- Hybrid mapping
  - Use both page-level and block-level map tables
  - Higher algorithm complexity

Page Mapping

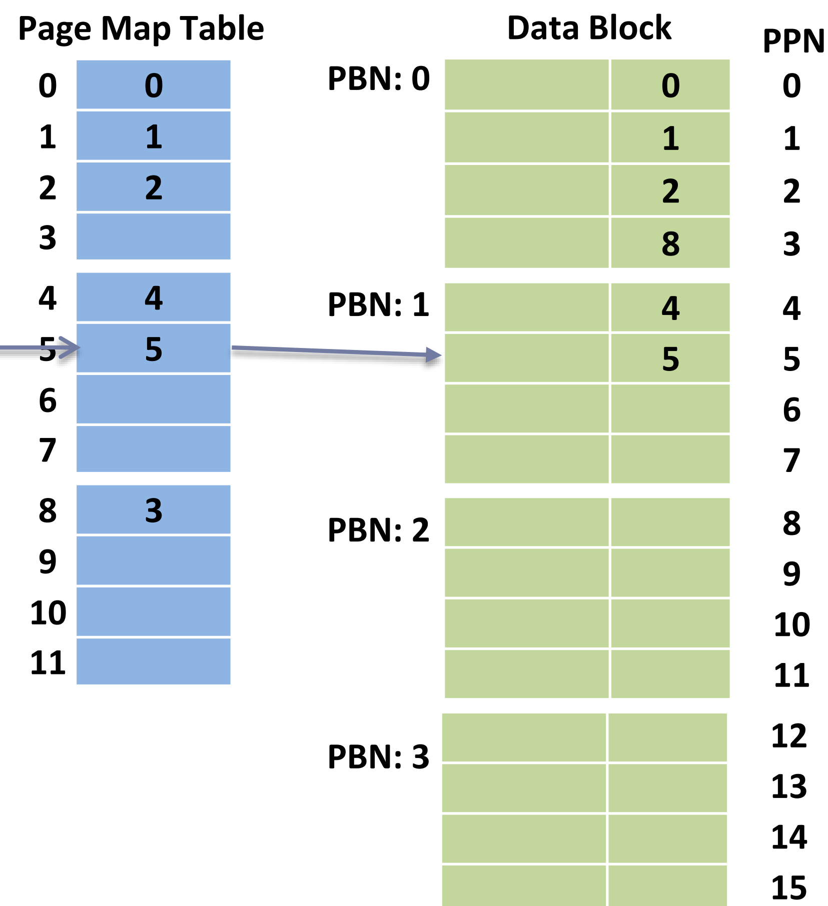
- Mapping in page-level
  - Logical page number → physical page number
  - Page mapping table (PMT) required
  - # entries in PMT == # pages visible to OS

- Translation
  - Step 1: logical sector number → logical page number (LPN)
  - Step 2: LPN → physical page number (PPN) via PMT

Example: Page Mapping

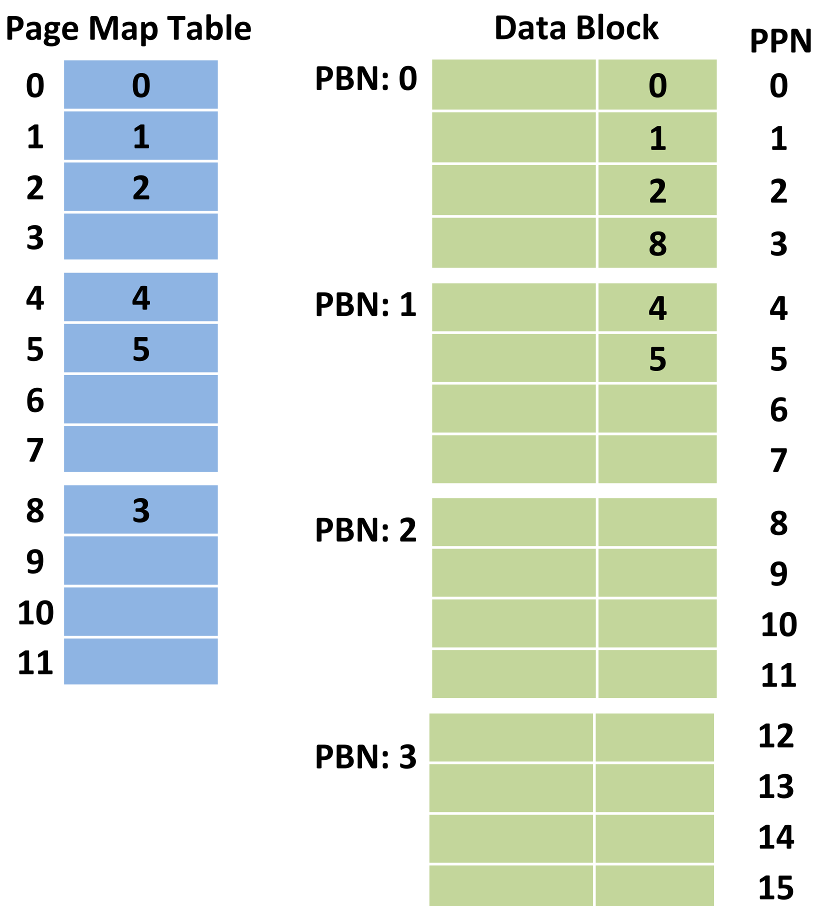
- Flash configuration
  - Page size: 4KB
  - # of pages / block = 4
- Current state
  - Written to page 0, 1, 2, 8, 4, 5
- Reading page 5

Logical page #5 0000000101



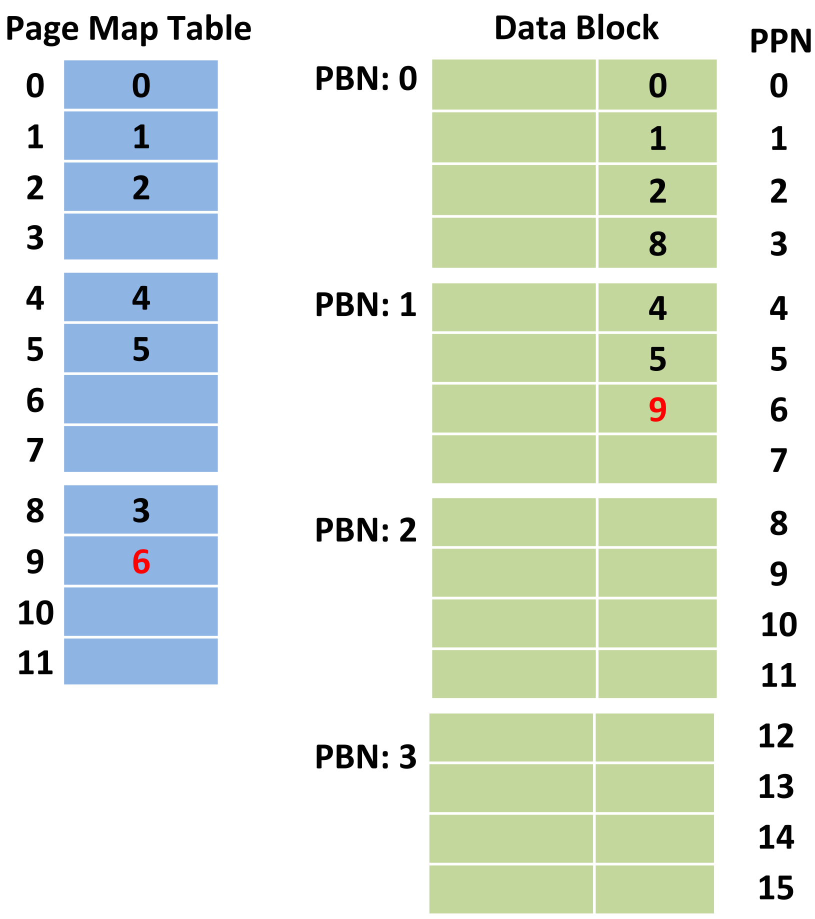
Example: Page Mapping

- 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



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



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

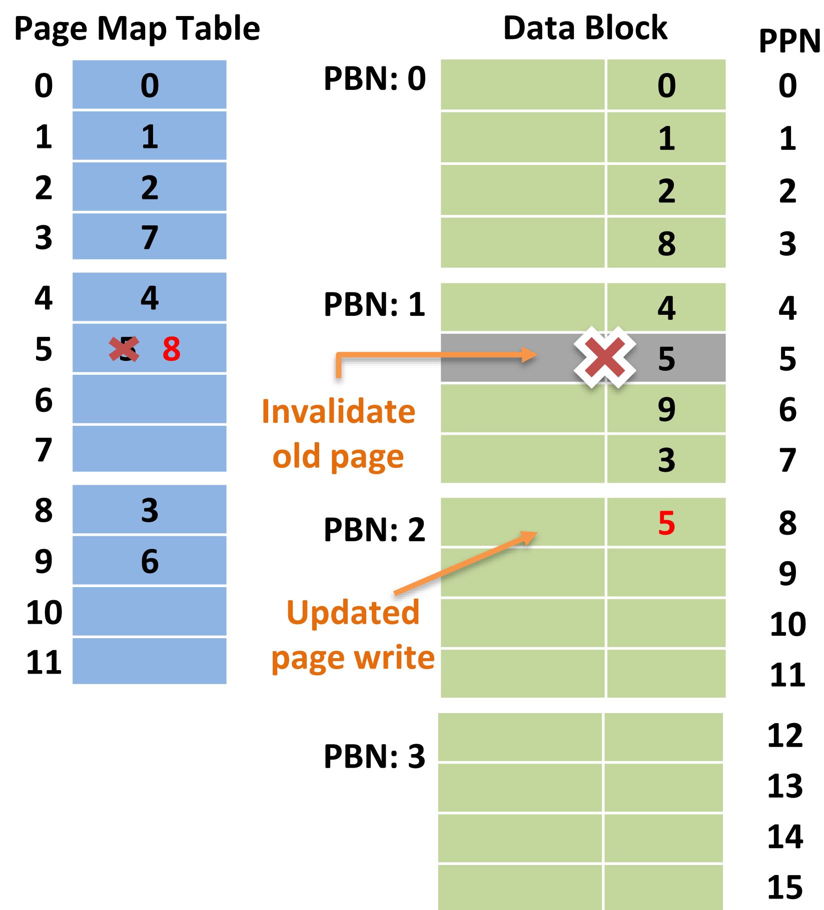
0	0
1	1
2	2
3	7
4	4
5	5
6	
7	
8	3
9	6
10	
11	

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

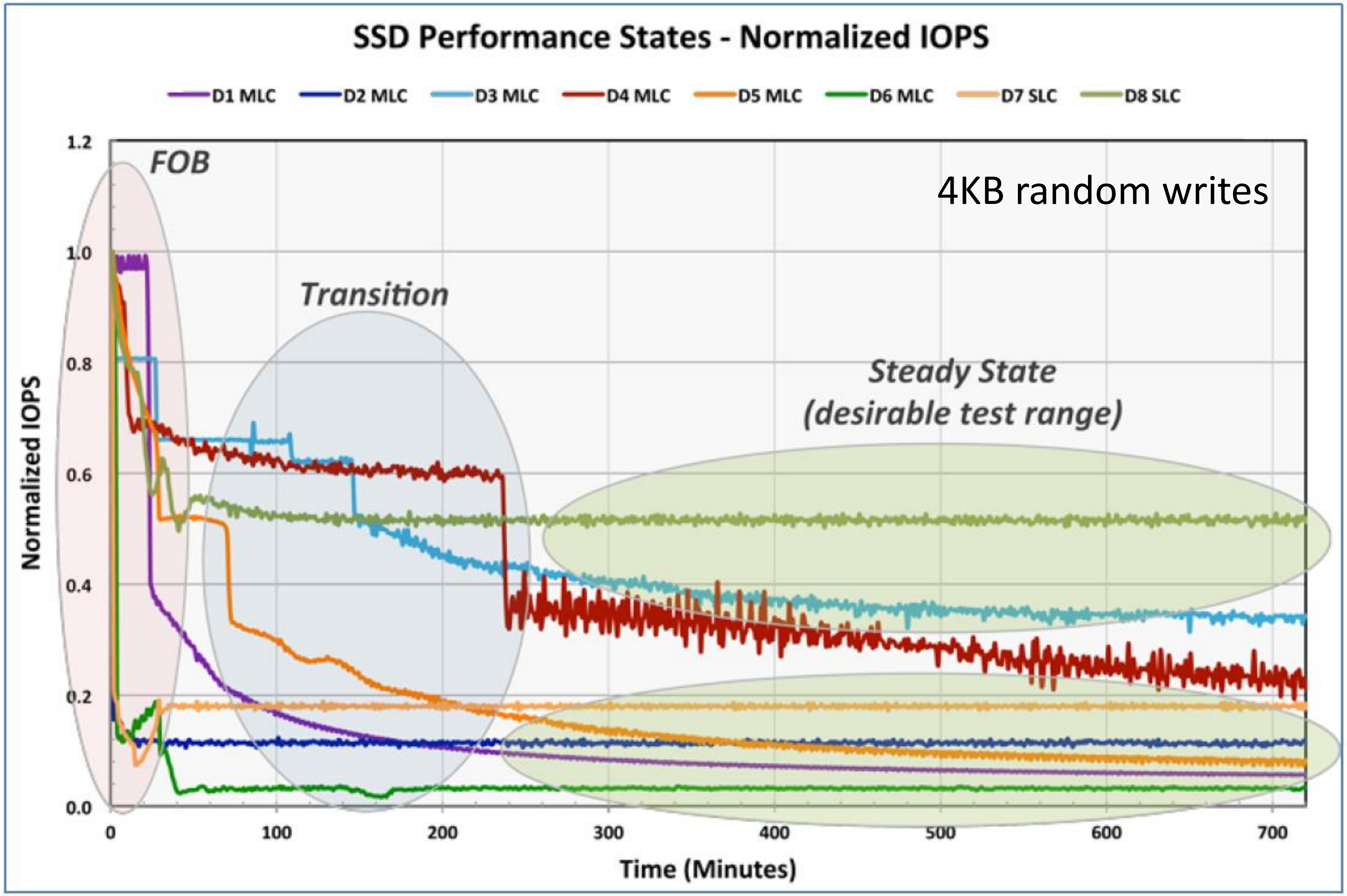


Page Mapping

- 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 (OP)
  - Performance affected as the system ages

Garbage Collection

Why?



http://tfindelkind.com/2015/08/20/

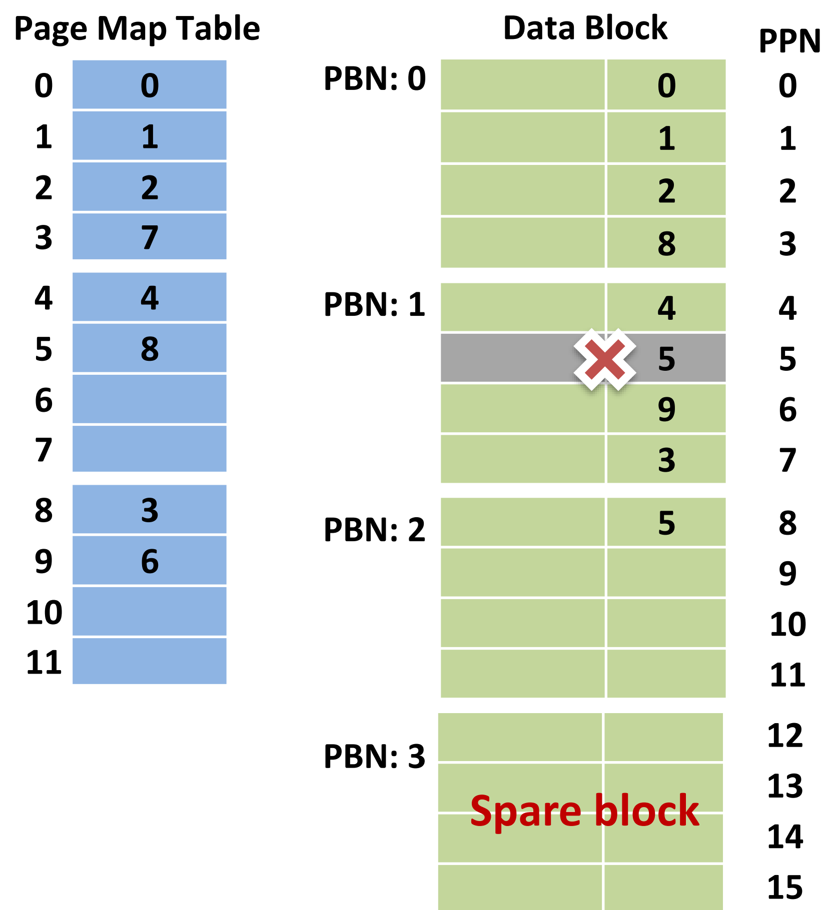
Garbage Collection

- 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

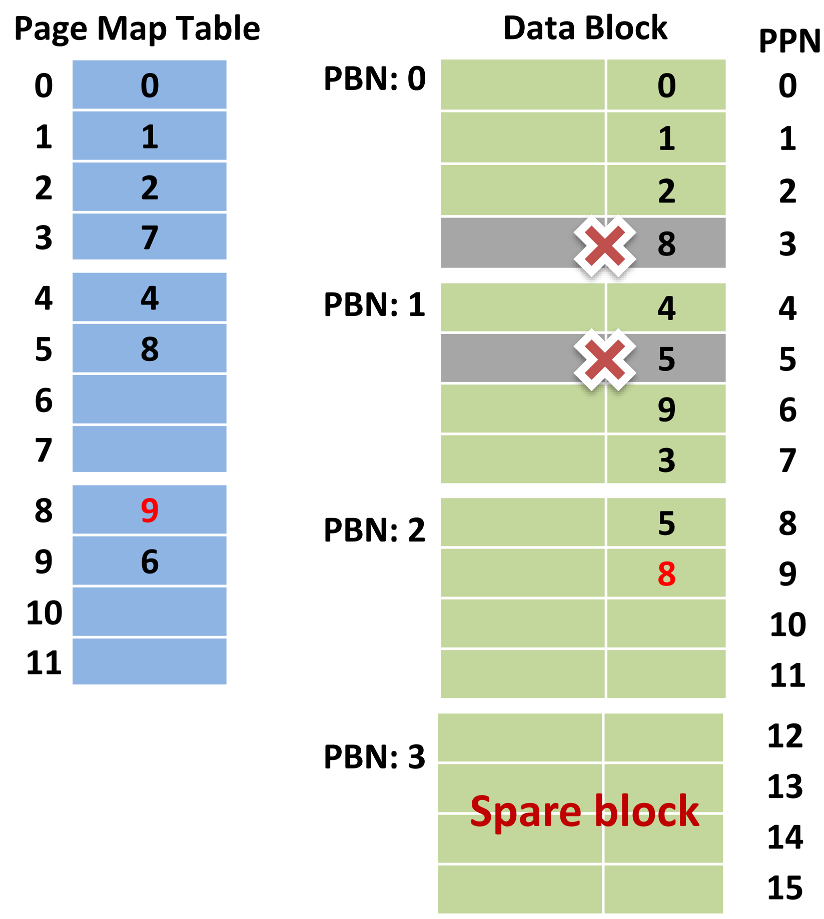
Example: GC in Page Mapping

- 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



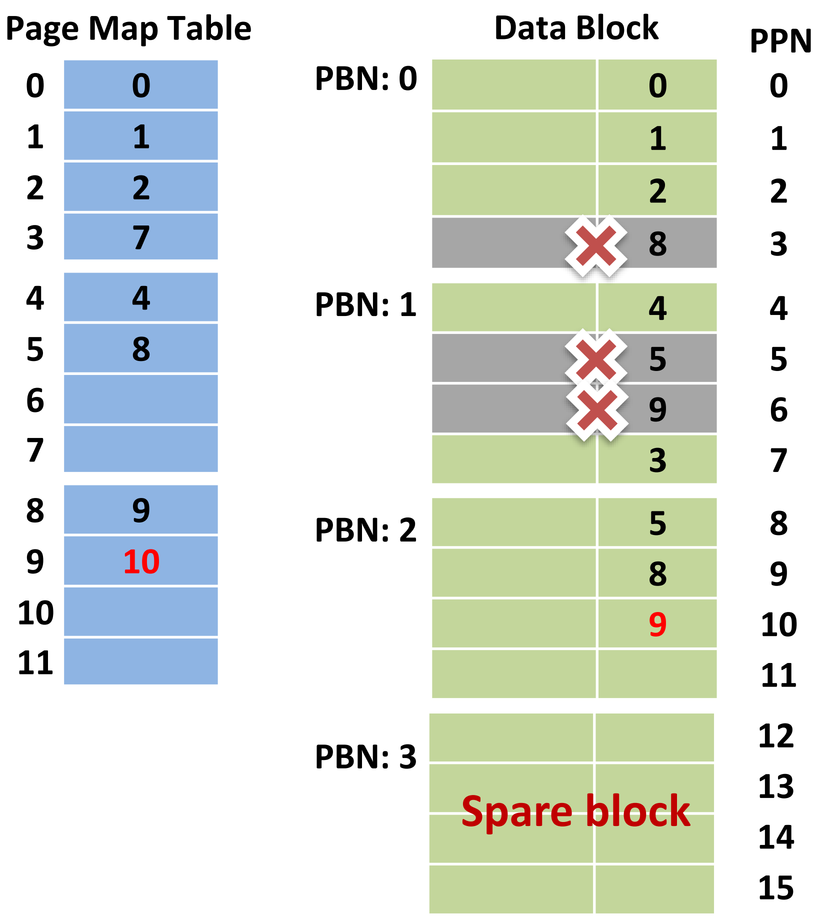
Example: GC in Page Mapping

- 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



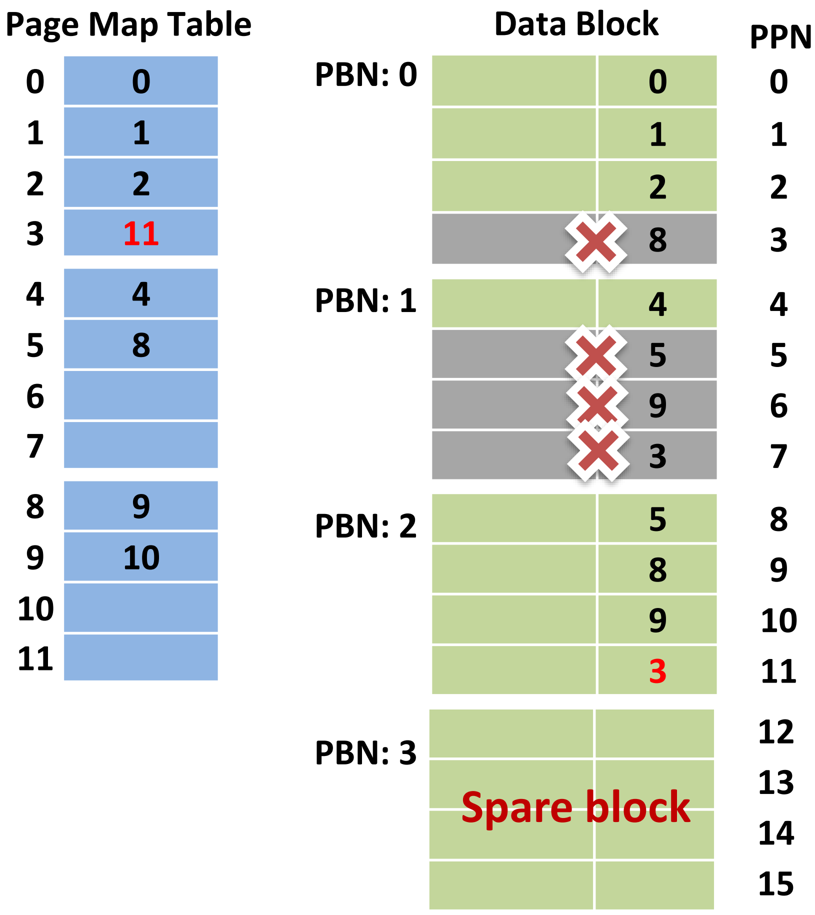
Example: GC in Page Mapping

- 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



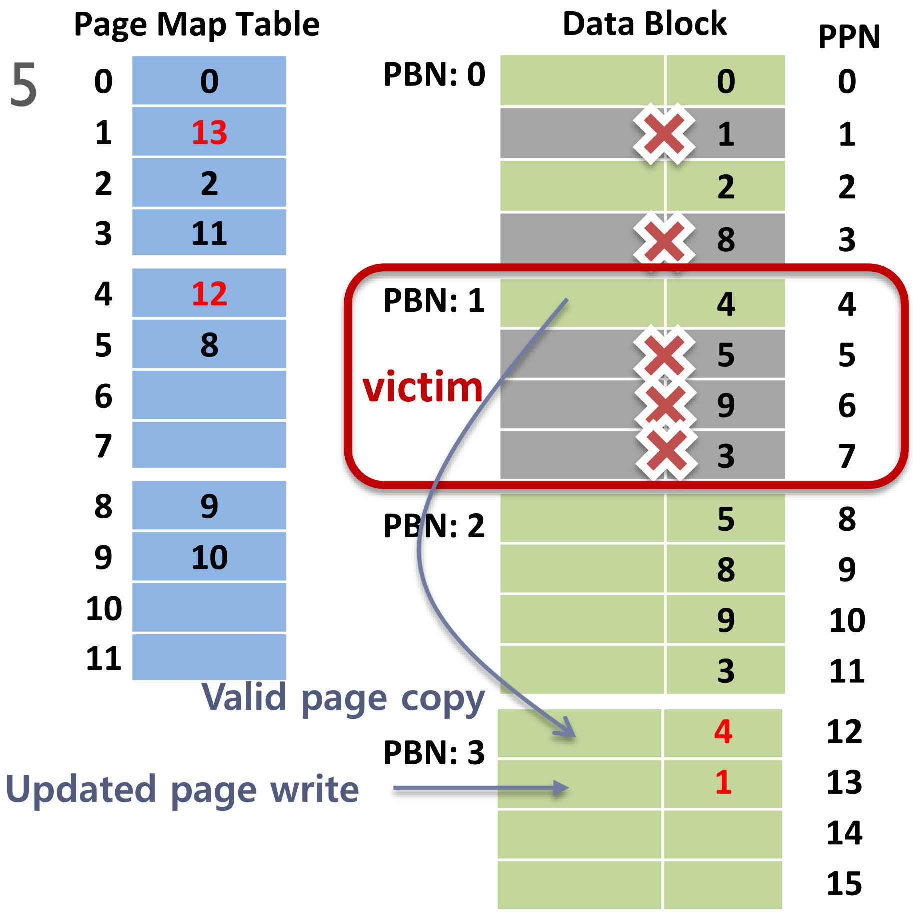
Example: GC in Page Mapping

- 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



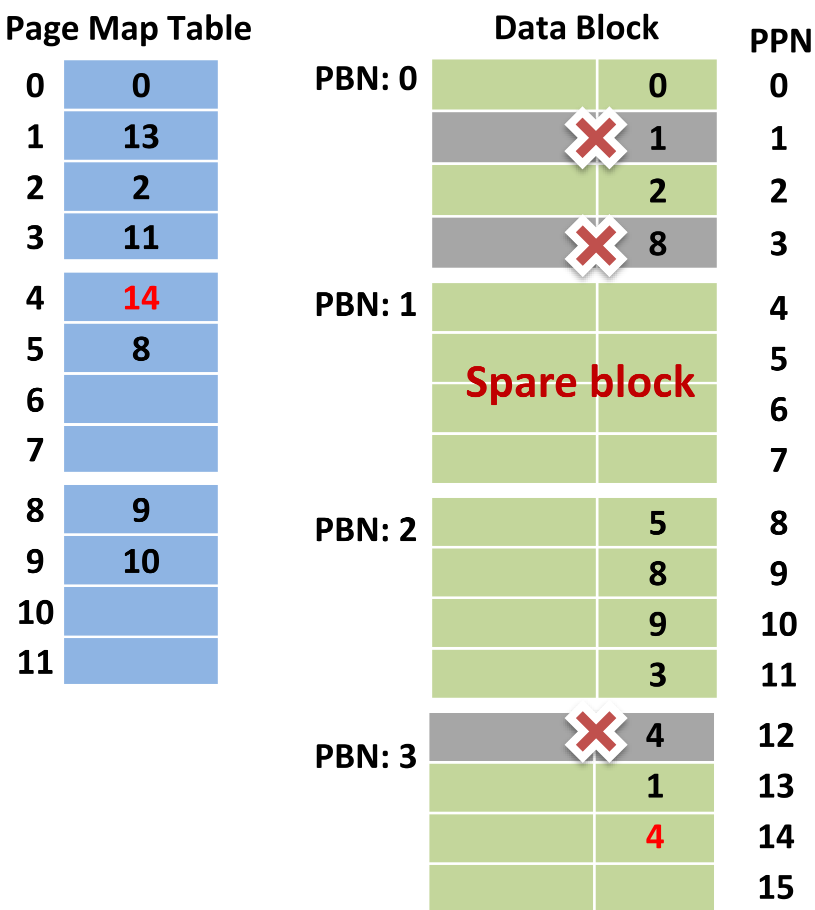
Example: GC in Page Mapping

- 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



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

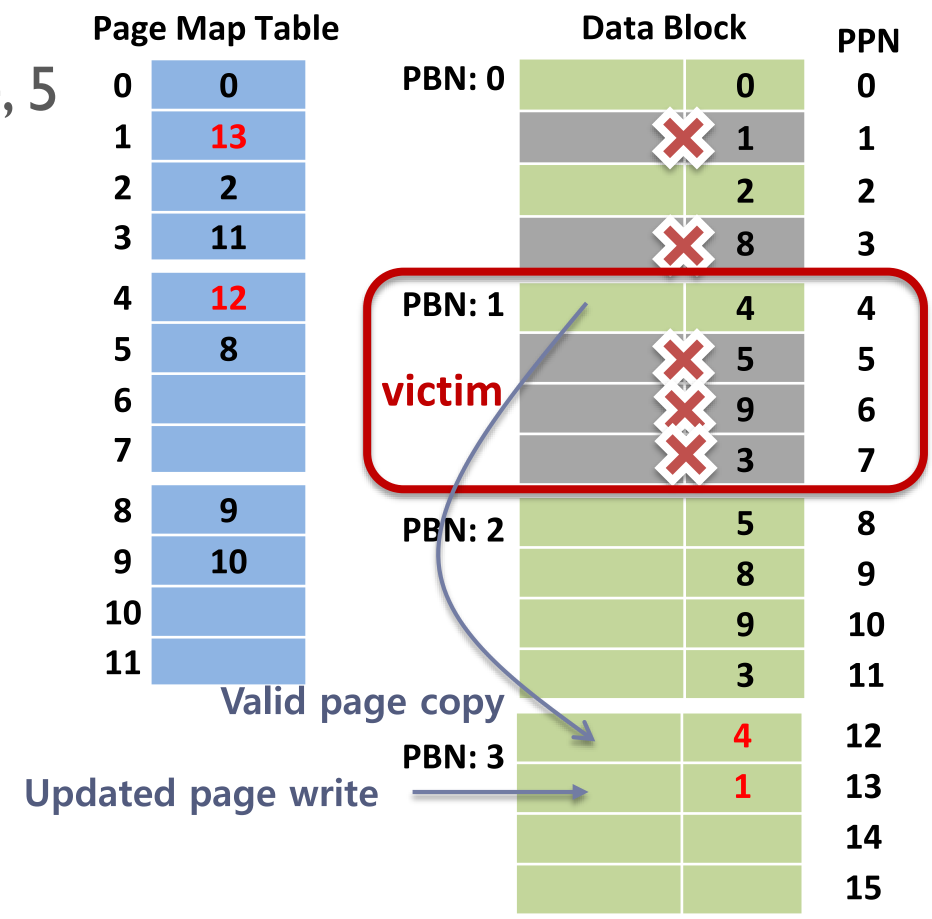
- Write Amplification Factor (WAF)

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

- 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 which 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 Policies

- Greedy policy
  - Selects a block with the largest amount of invalid data
  - A block with the minimum utilization *u*

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

  - Pros
    - Least valid data copying costs
    - Simple
  - Cons
    - Does not perform well when there is high locality among writes
    - Does not consider wear leveling

Victim Selection Policies

- Cost-benefit policy
  - Selects a block with the maximum

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

    - *u*: utilization
    - *age*: the time since the last modification
  - Pros
    - Performs well with locality
    - Somehow helps to achieve even wear
  - Cons
    - Computation/data overhead

Victim Selection Policies

- Cost-Age-Times (CAT) policy
  - Selects a block with the minimum

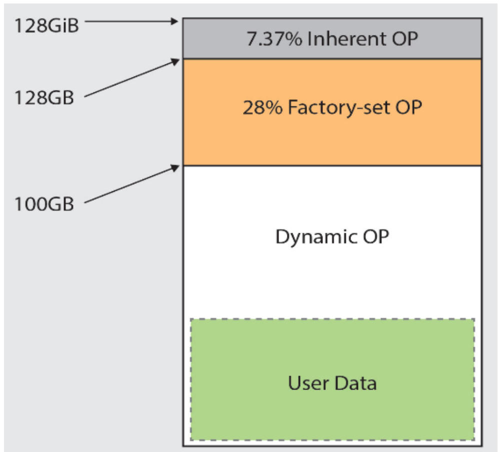
$$\frac{Cost}{Benefit} \times \frac{Times}{Age} = \frac{u}{(1-u) \times age} \times count$$

    - *u*: utilization
    - *age*: the time since the last modification
    - *count*: erase count for the block
  - Pros
    - Performs well with locality
    - Takes block wear counts into account
  - Cons
    - Computation/data overhead

Over-Provisioning

- OP (Over-Provisioning) $= \dfrac{Physical\ Capacity}{Logical\ Capacity} - 1$
  - 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%
    - e.g. 100GB user space on 128GiB SSD:
      ~ 28% + 7% = 35%



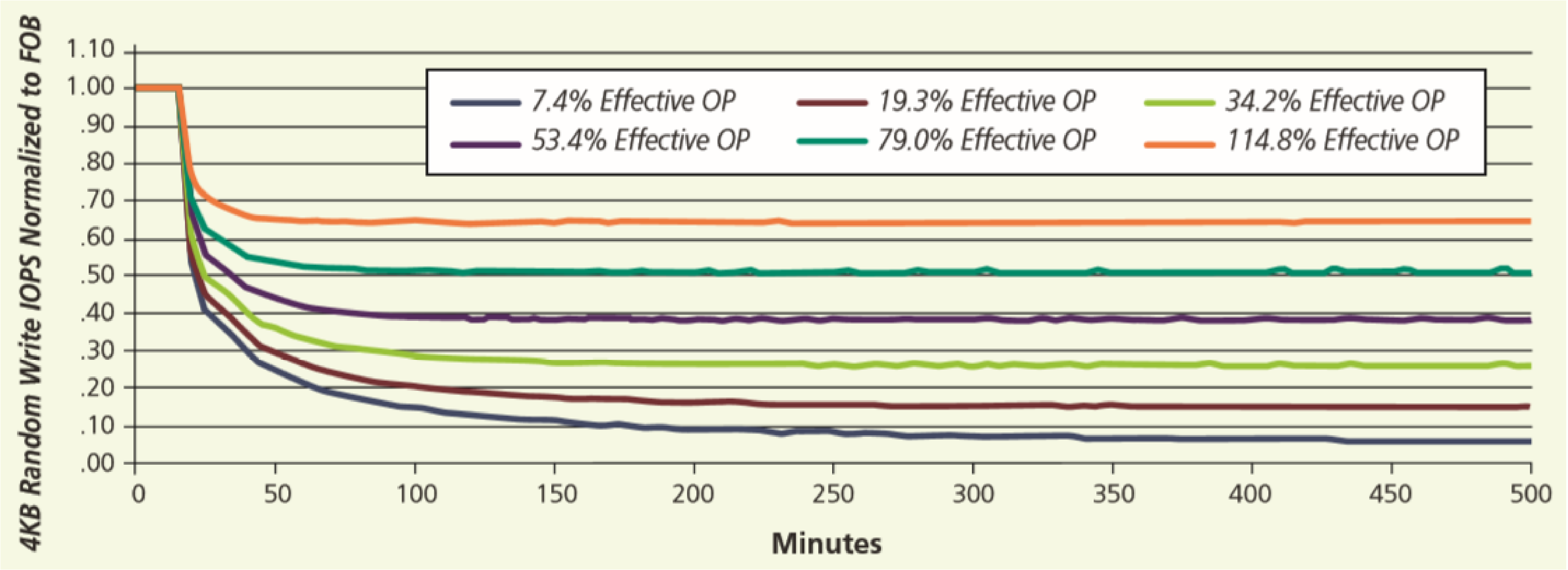
Over-Provisioning

- Over-Provisioning Space (OPS) is used for
  - Firmware images
  - FTL metadata
  - Bad block remapping
  - Write buffers

- Garbage collection cost
  - Affected by utilization of SSD space and Over-Provisioning
  - Lower utilization → Better performance
  - Larger OP → Better performance

Over-Provisioning

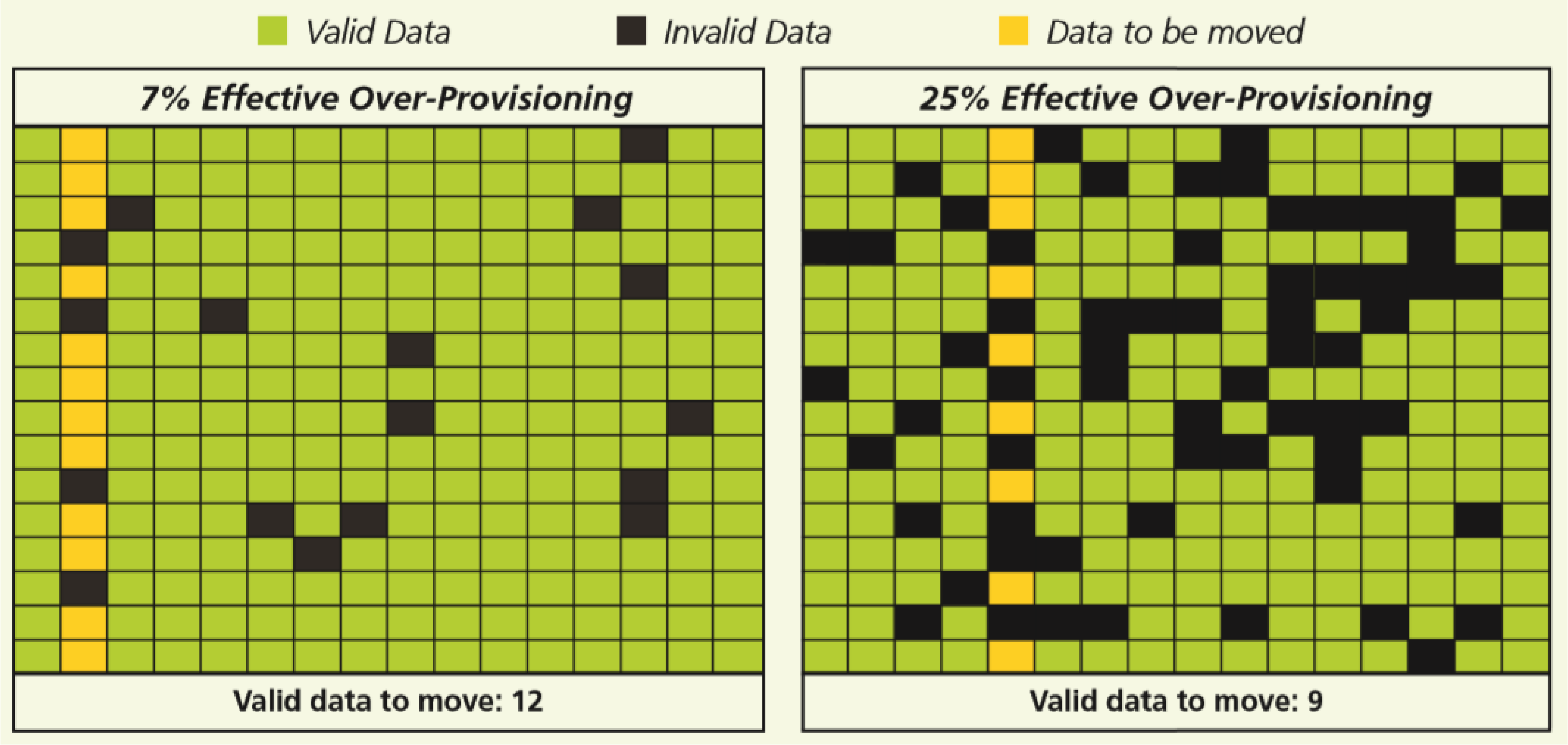
- Over-provisioning and random write workloads
  - What about for sequential write workloads?



D. Glen, Differences in Personal vs. Enterprise SSD Performance, Micron Technology, Inc.

Over-Provisioning

- Over-provisioning on GC
  - Larger OP results in lower WAF



D. Glen, Differences in Personal vs. Enterprise SSD Performance, Micron Technology, Inc.

Over-Provisioning

- Some manufacturers provide software tools to configure the amount of over-provisioning space