4. Storage Firmware (Part 1)

Special Topics in Computer Systems:
Modern Storage Systems
(IC820-01)

Instructor:

Prof. Sungjin Lee (sungjin.lee@dgist.ac.kr)

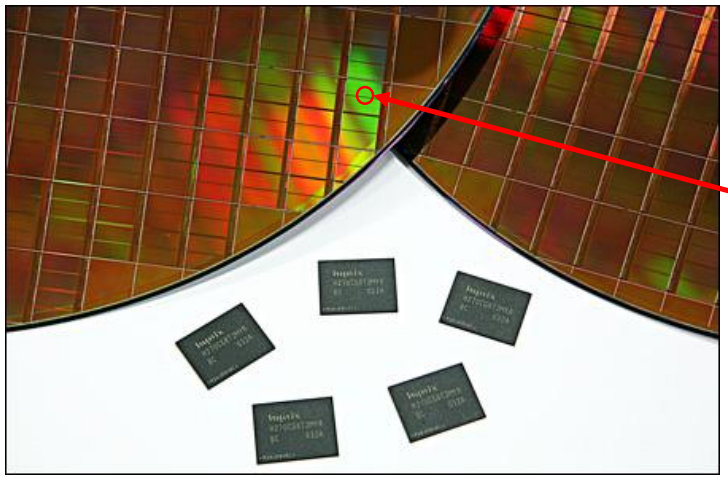
Outline

- **Review: NAND Cell Array**
- Flash Translation Layer
- Address Translation
- Garbage Collection

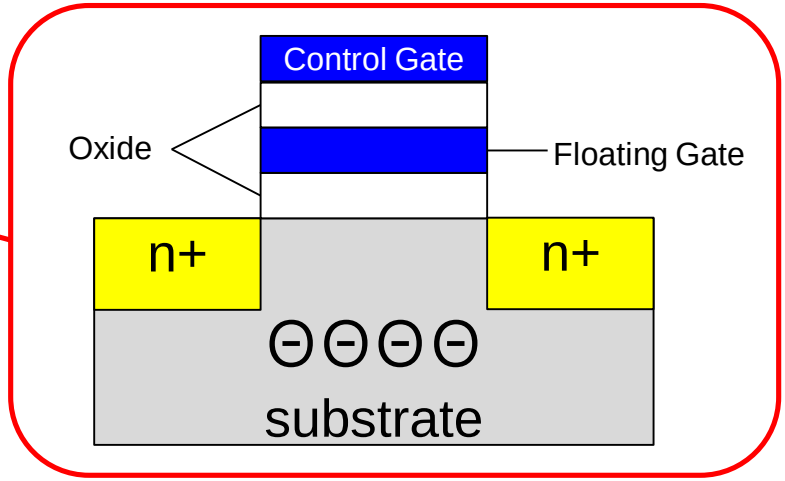
Review:

Flash Memory

- **Fundamentally different from HDDs – It is based on a transistor (a cell) that can be written and read using *electronic circuits***
  - No mechanical parts needed
- **Flash memory is "non-volatile"**
  - One (or more) bits are stored in a ***floating gate transistor*** (a cell) that holds a value *without power supply*



Wafer, Die, and Package

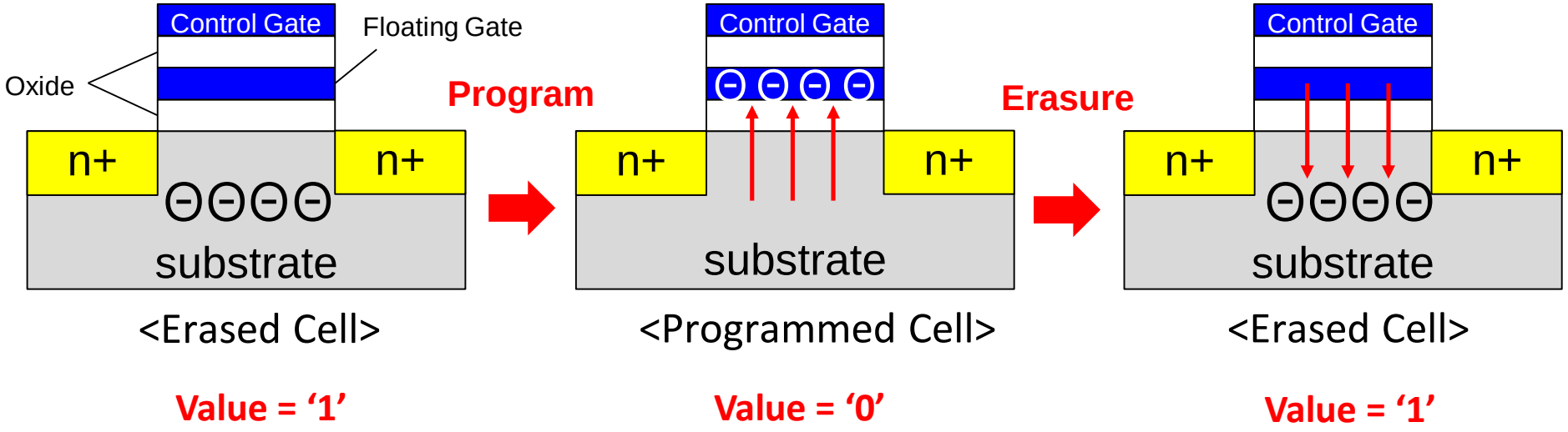


Floating Gate Transistor

Review:

How Value is Stored and Retrieved?

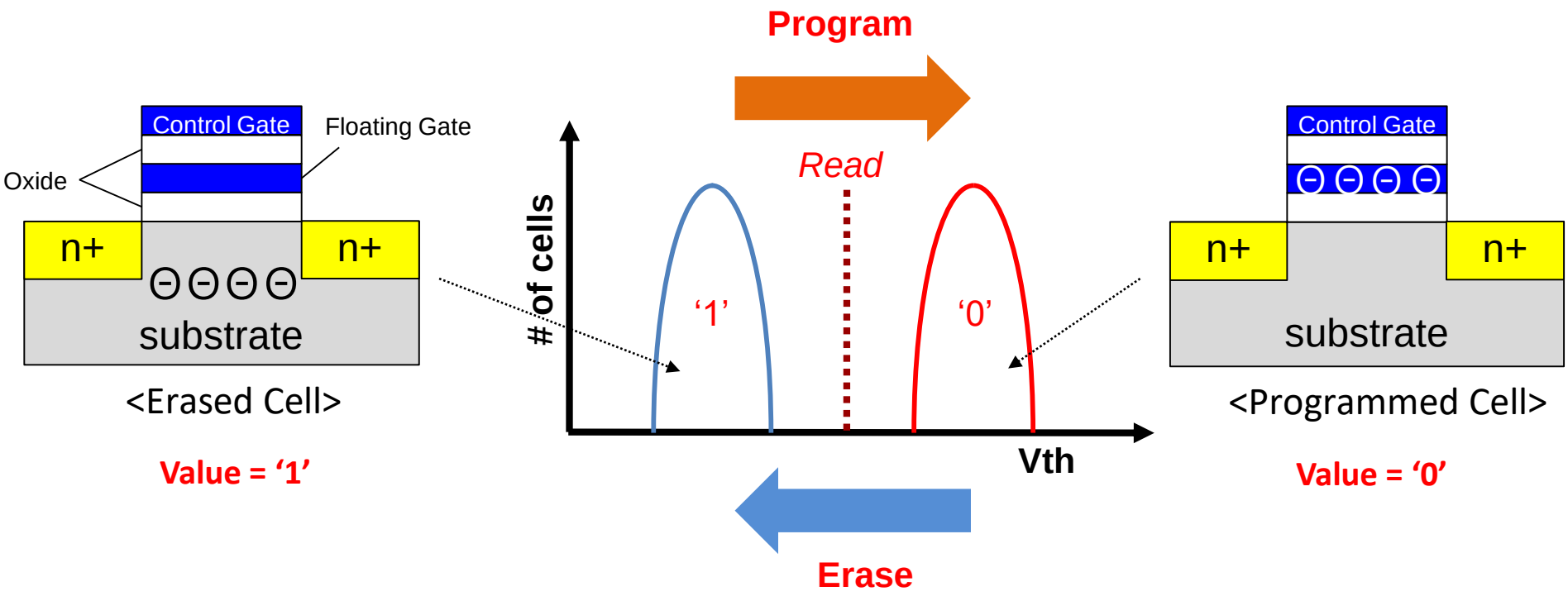
- **A bit value of each cell can be controlled by three operations**
  - Each cell is initially erased, and its value is represented as '1'
  - A value is changed to '0' by a ***program*** operation
  - A value is changed to '1' by an ***erasure*** operation
  - A value can be retrieved by a ***read*** operation



Review:

How Value is Stored and Retrieved? (Cont.)

- **Program & read binary data to a flash cell**
  - Data “0” → Program → Shift cell Vth to high → Off state → No current flow
  - Data “1” → Erase → Shift cell Vth to low → On State → Current flow

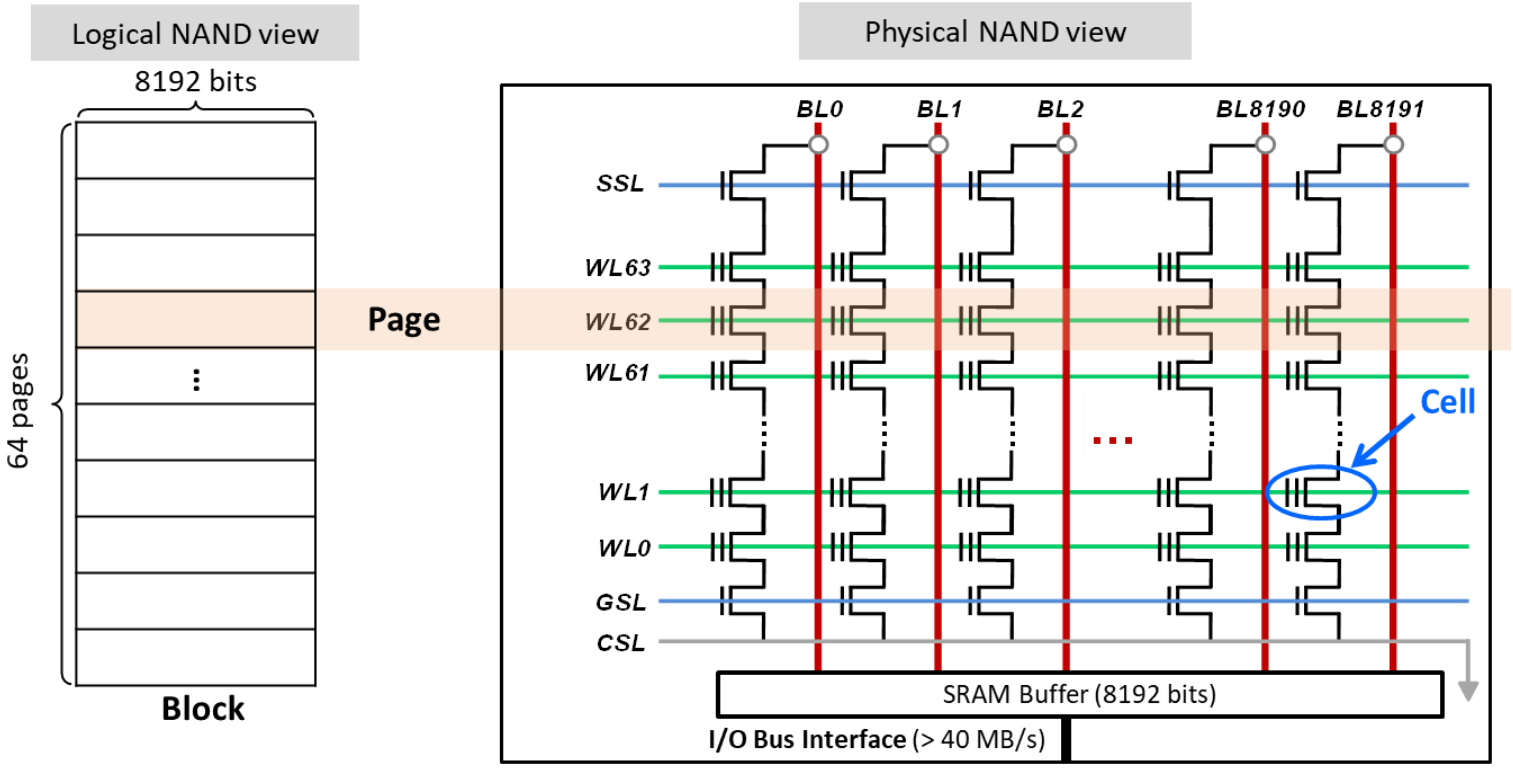


✓ *Read : Check the current flow*

Review:

NAND Cell Array

- **A group of NAND cells are written and read simultaneously**
  - This group of cells is called a *page* (4K – 16K cells)
- **A group of pages should be erased together**
  - This group of pages is called a *block* (128 – 256 pages)
- **This asymmetric I/O unit makes *random writes quite expensive***



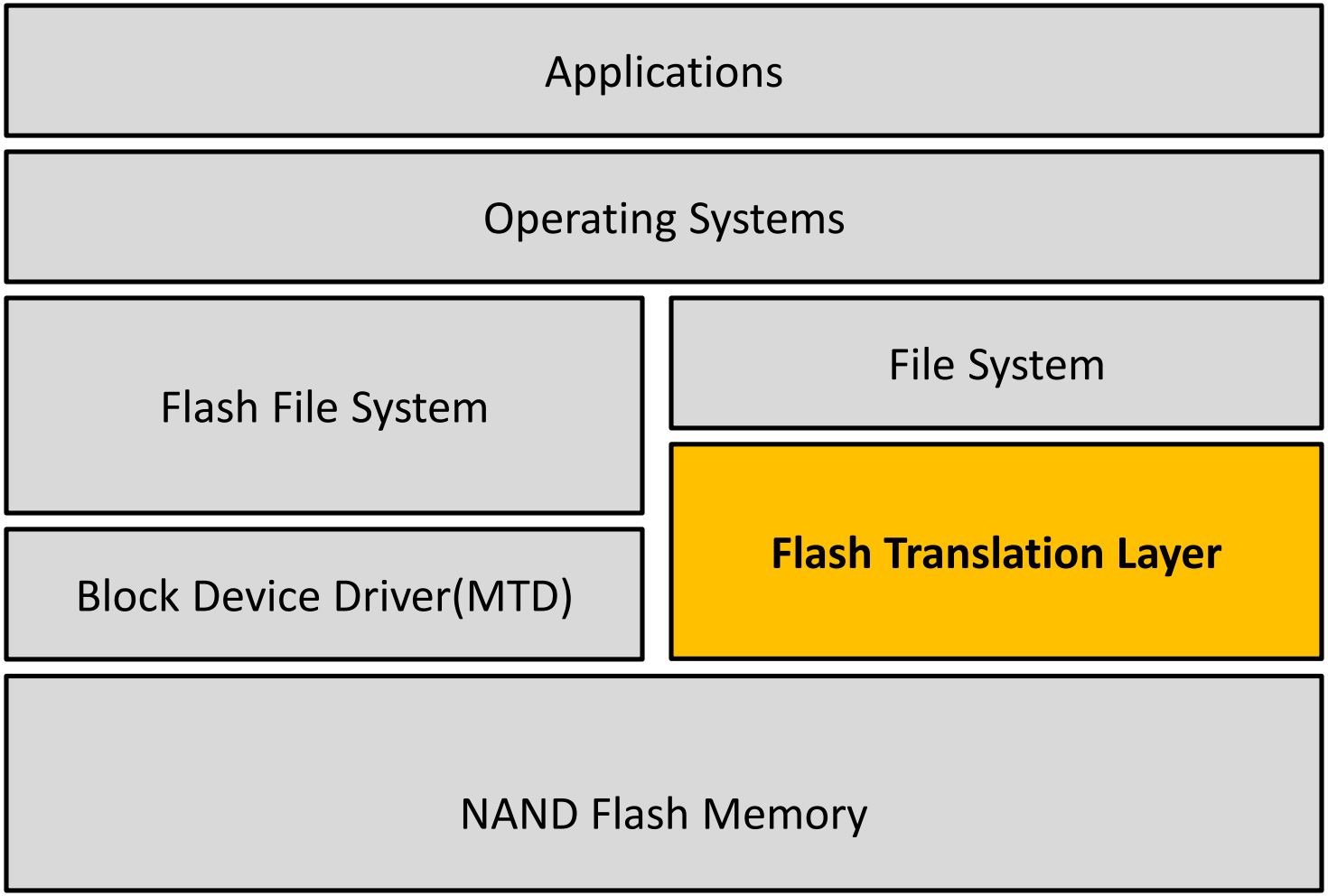
Outline

- Review: NAND Cell Array
- **Flash Translation Layer**
- Address Translation
- Garbage Collection

Flash Translation Layer (FTL)

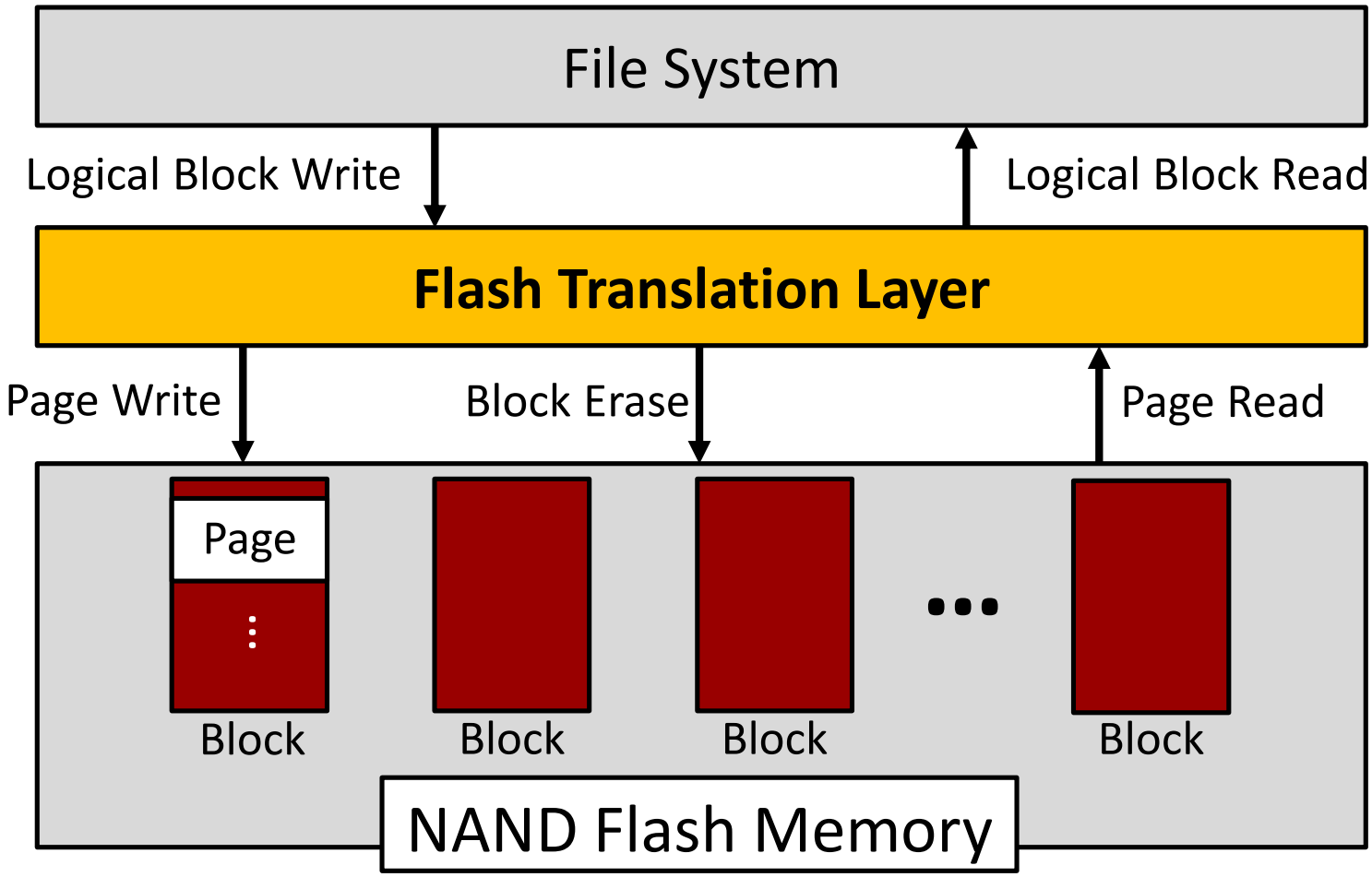
- **Flash-based SSDs have quite different characteristics from conventional hard disk drives (HDDs)**
  - Different IO primitives: read, write, and erasure
  - Asymmetric I/O units: 4-16KB for reading and writing, 2-4MB for erasure
  - No in-place update
  - Multi-channels/ways
  - Limited lifetime
  - …
- **The storage firmware, a flash translation layer (FTL), is responsible for addressing all the above issues!**

Software Architecture



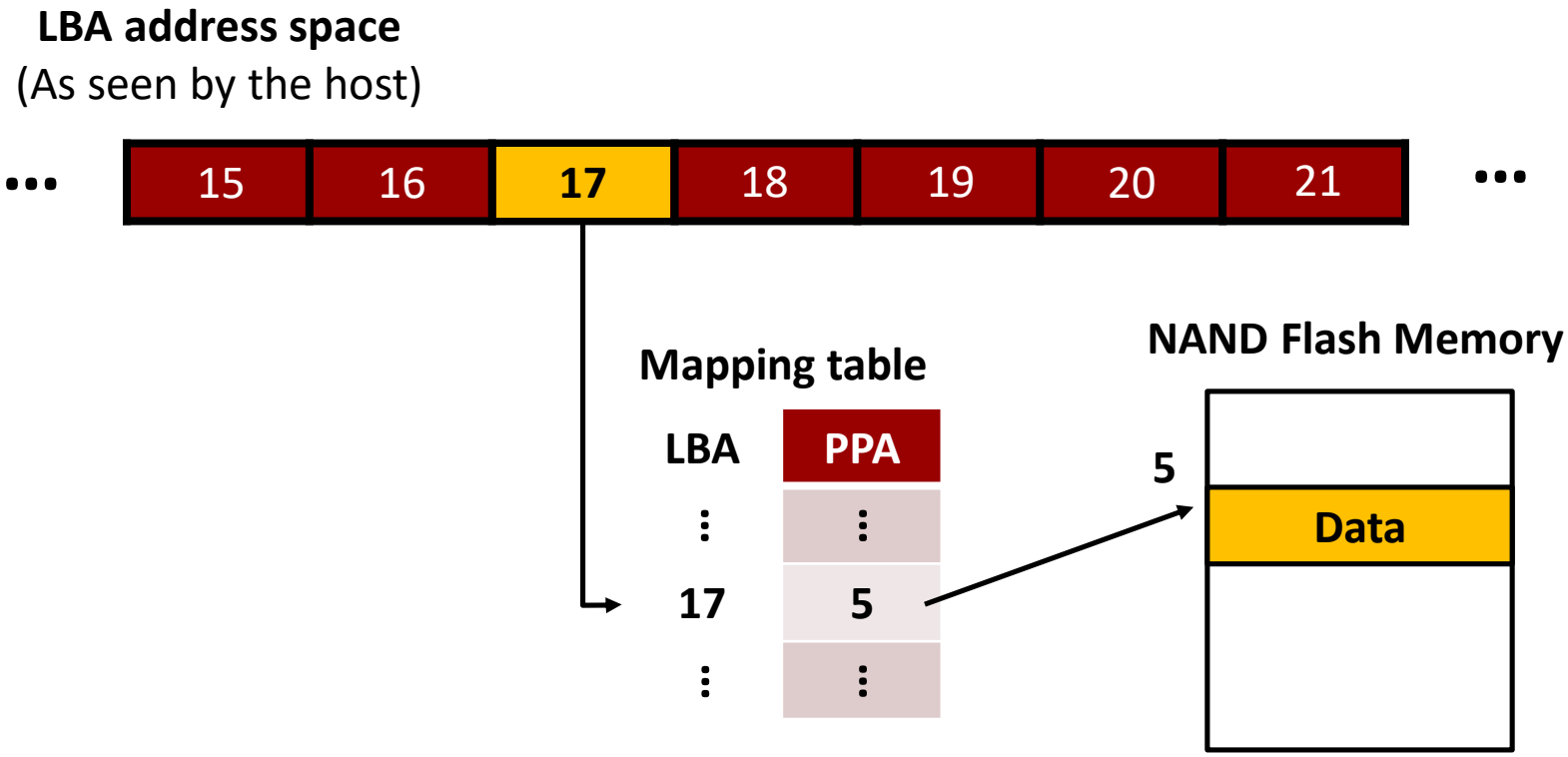
Flash Translation Layer

- **A software layer to make NAND Flash emulate traditional Block devices (or disks)**



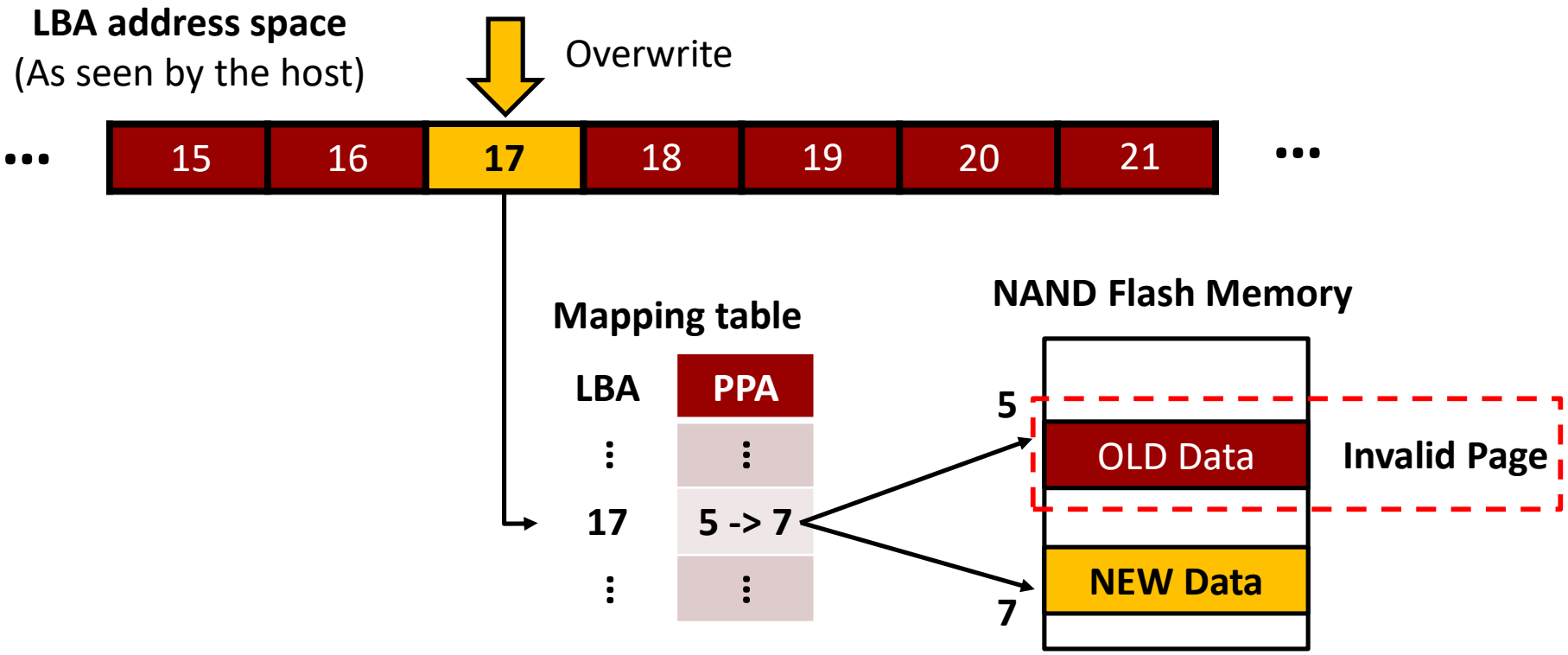
Out-Place Update

- NAND flash memory does not support overwrite operations
- FTL uses an out-place update policy, generating invalid pages



Out-Place Update

- NAND flash memory does not support overwrite operations
- FTL uses an out-place update policy, generating invalid pages



Detailed Roles of FTL

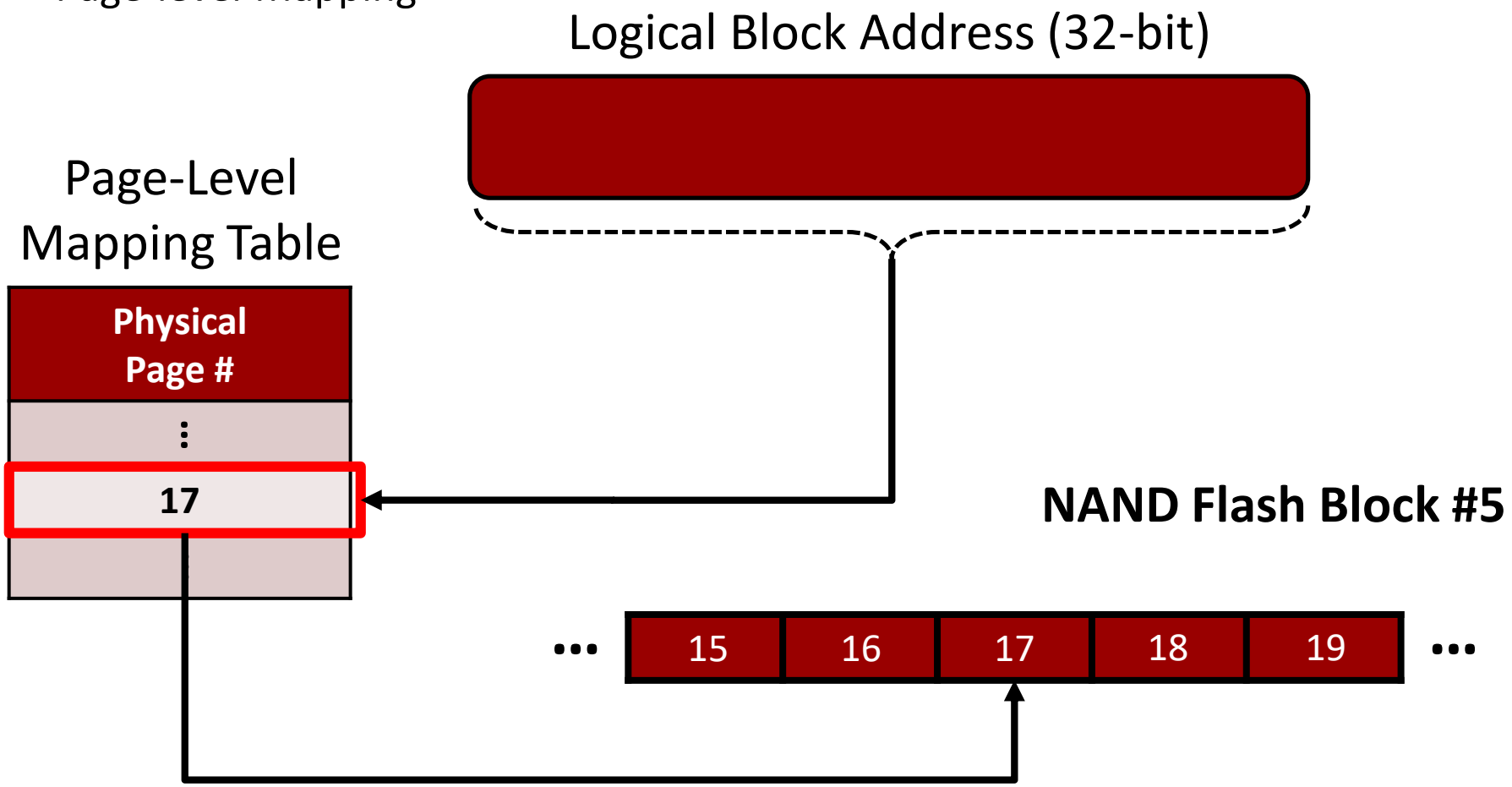
- **For performance**
  - Indirect mapping (Address Translation)
  - Garbage Collection
  - Over-provisioning
  - etc.
- **For Reliability**
  - Bad Block management
  - Wear-leveling
  - Error Correction Code (ECC)
  - etc.
- **Other Features**
  - Encryption
  - Compression
  - De-duplication
  - etc.

Outline

- Review: NAND Cell Array
- Flash Translation Layer
- **Address Translation**
- Garbage Collection

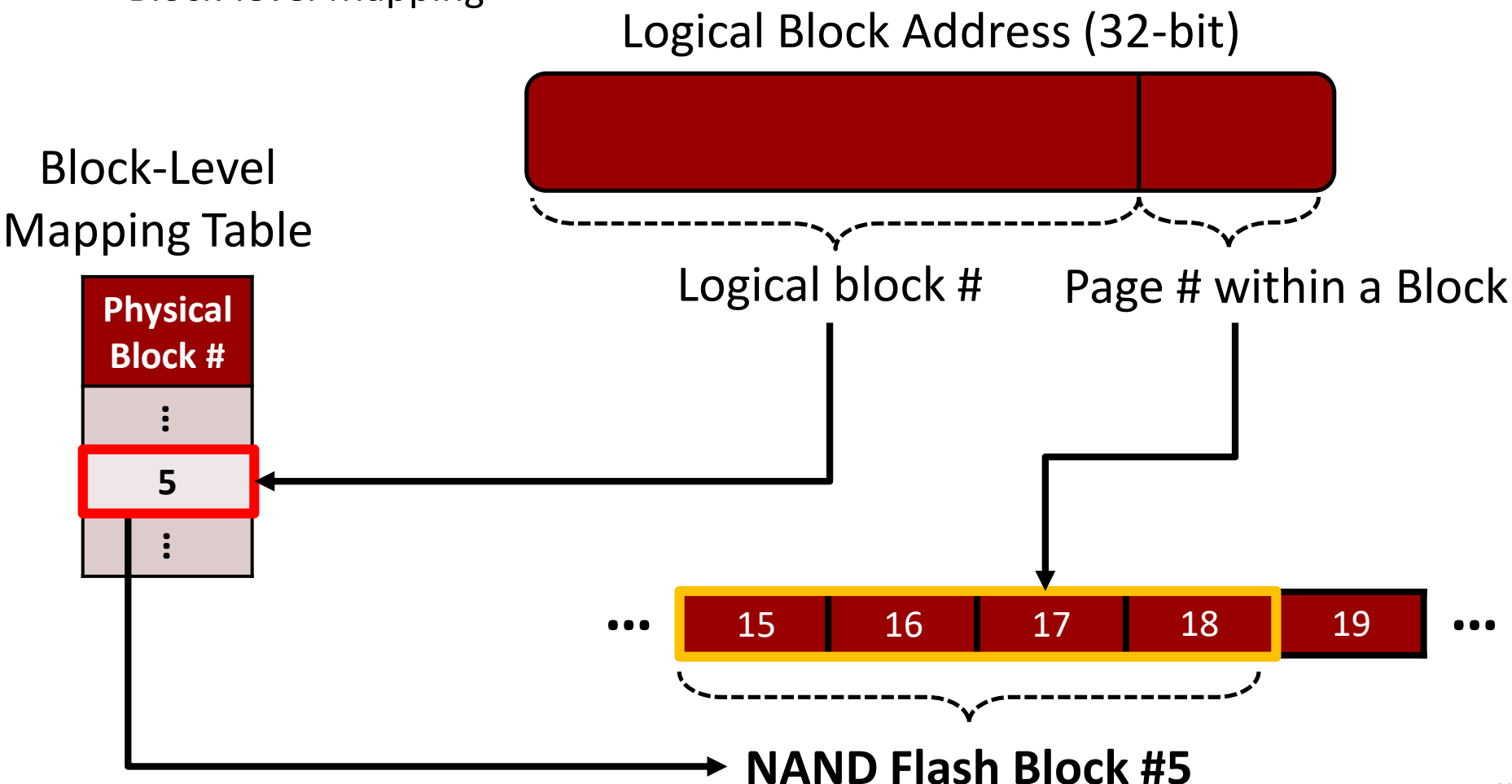
Mapping

- **Mapping Granularity**
  - Page-level Mapping



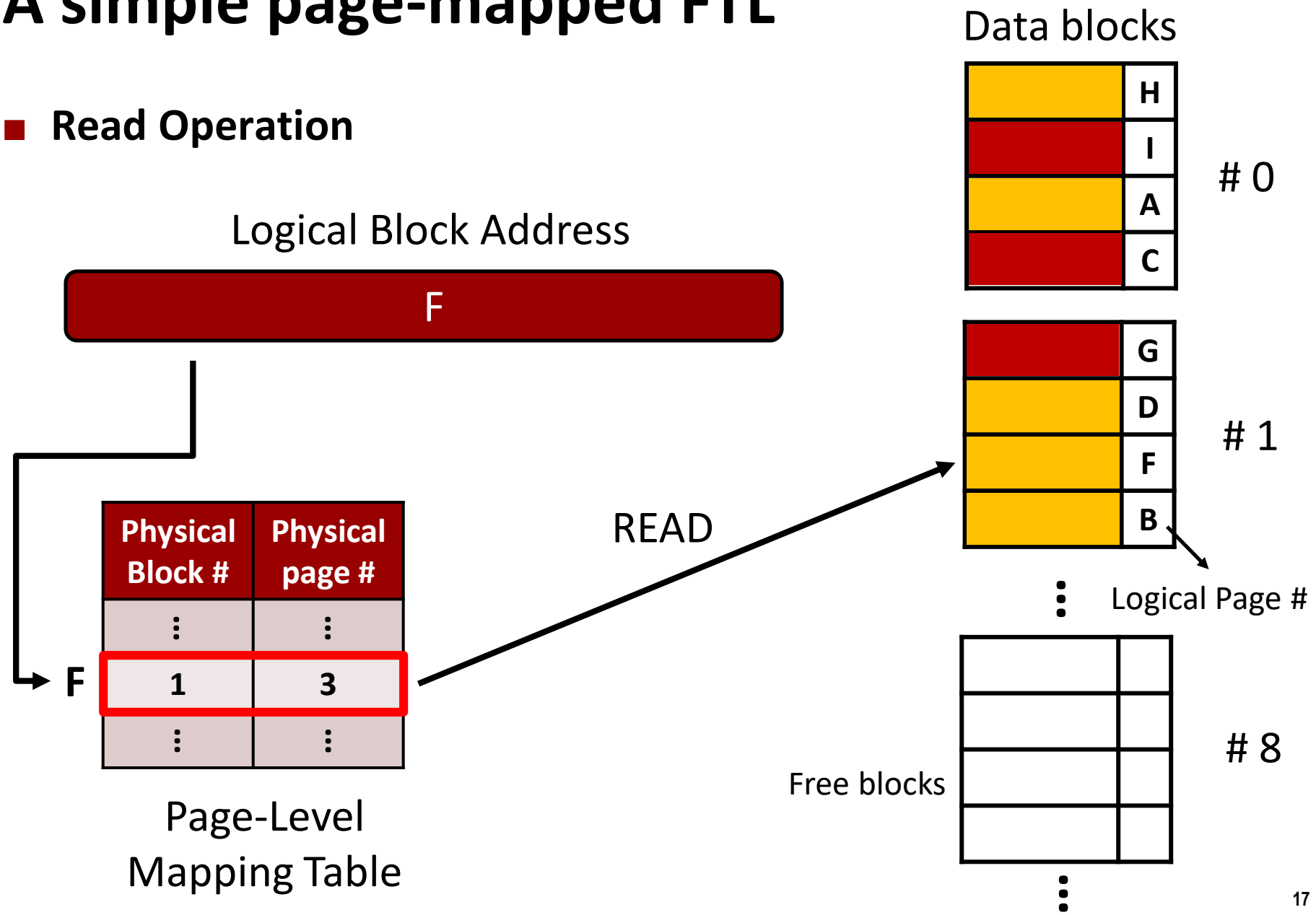
Mapping

- **Mapping Granularity**
  - Block-level Mapping



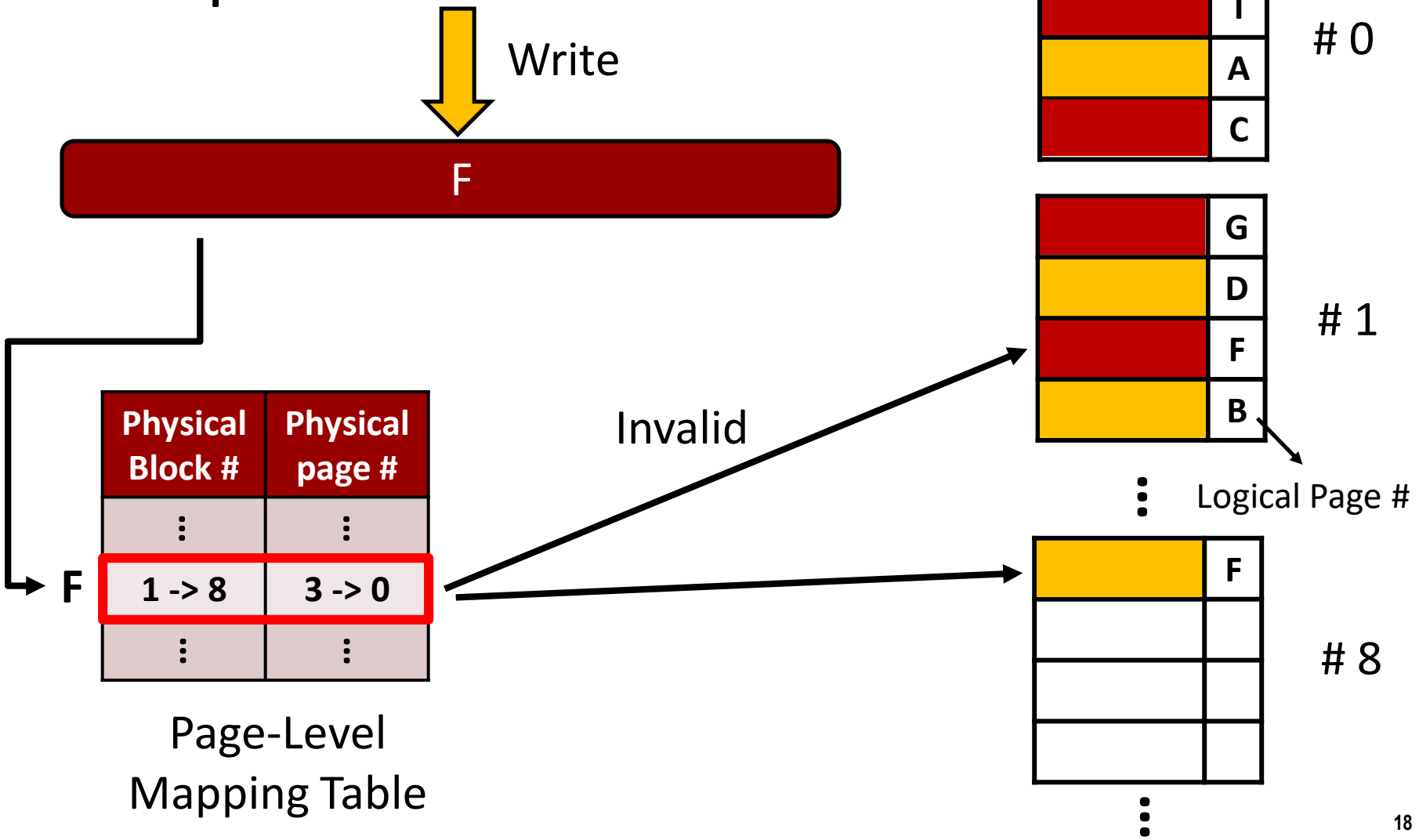
A simple page-mapped FTL

- **Read Operation**



A simple page-mapped FTL

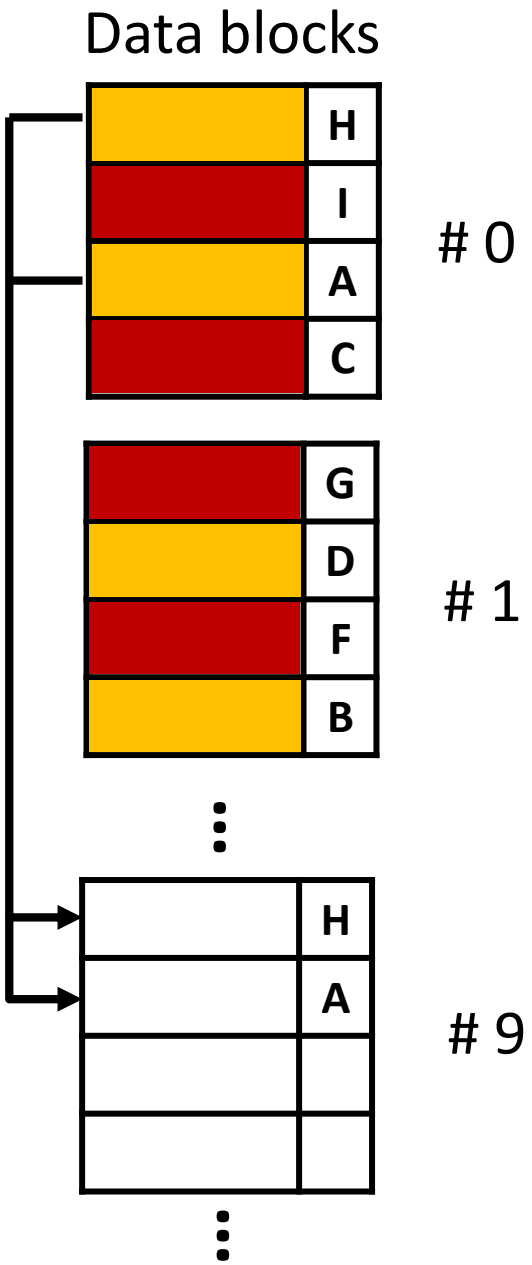
- **Write Operation**



A simple page-mapped FTL

- **Garbage Collection**
  - 1. Select the Victim block and free block
  - 2. Copy all the valid data to free
  - 3. Update mapping table
  - 4. Erase victim block

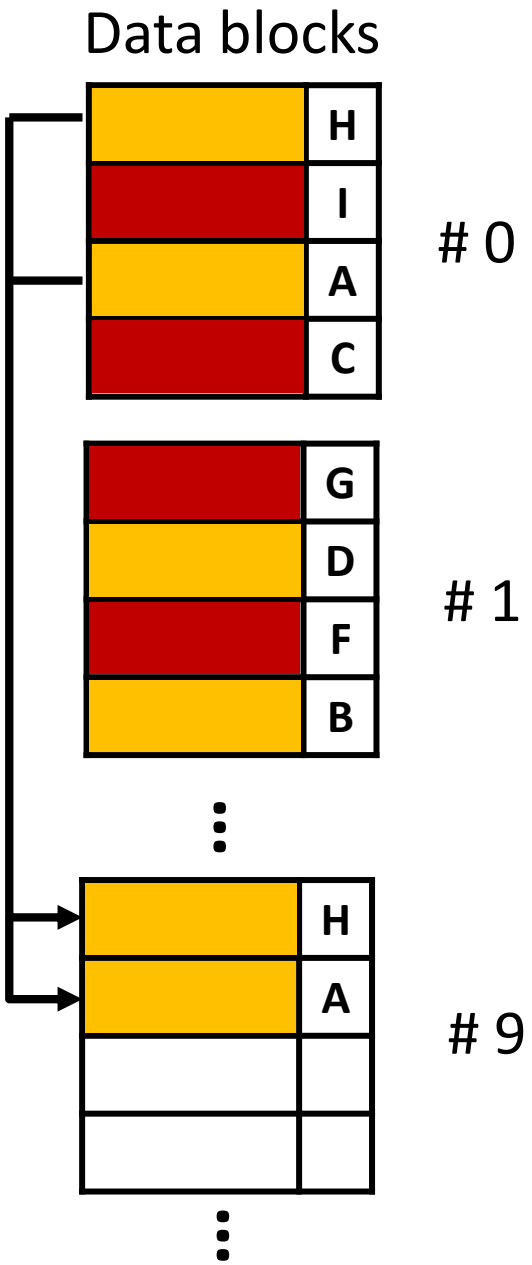
	Physical Block #	Physical page #
H	0	0
A	0	2



A simple page-mapped FTL

- **Garbage Collection**
  - 1. Select the Victim block and free block
  - 2. Copy all the valid data to free
  - 3. Update mapping table
  - 4. Erase victim block

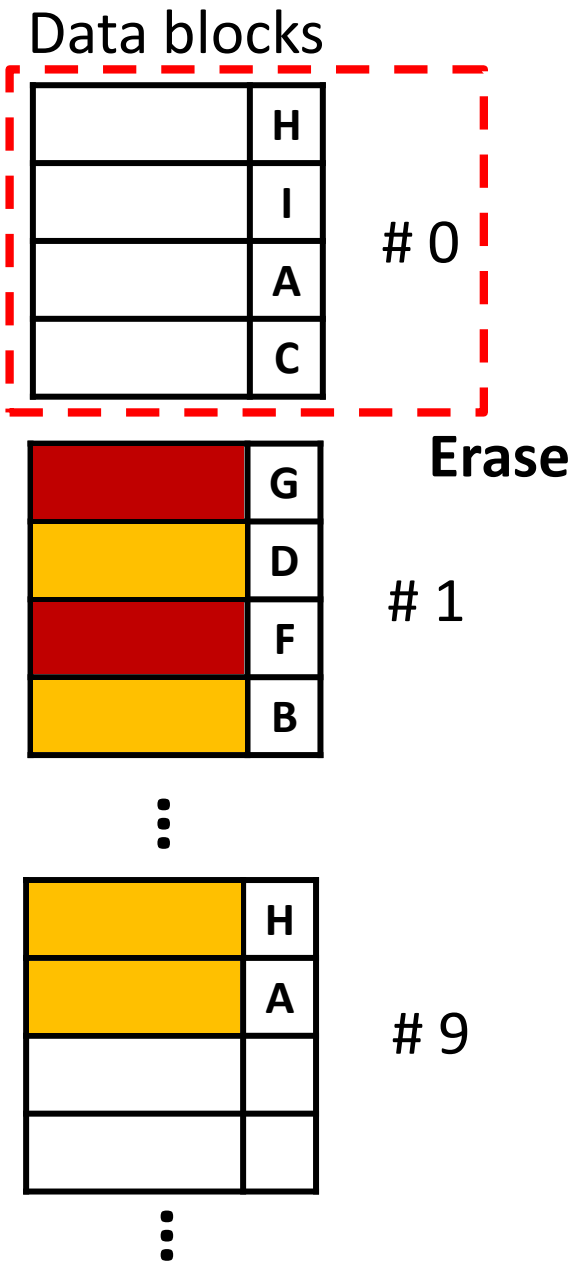
	Physical Block #	Physical page #
H	0→9	0→0
A	0→9	2→1



A simple page-mapped FTL

- **Garbage Collection**
  - 1. Select the Victim block and free block
  - 2. Copy all the valid data to free
  - 3. Update mapping table
  - 4. Erase victim block

	Physical Block #	Physical page #
H	0→9	0→0
A	0→9	2→1



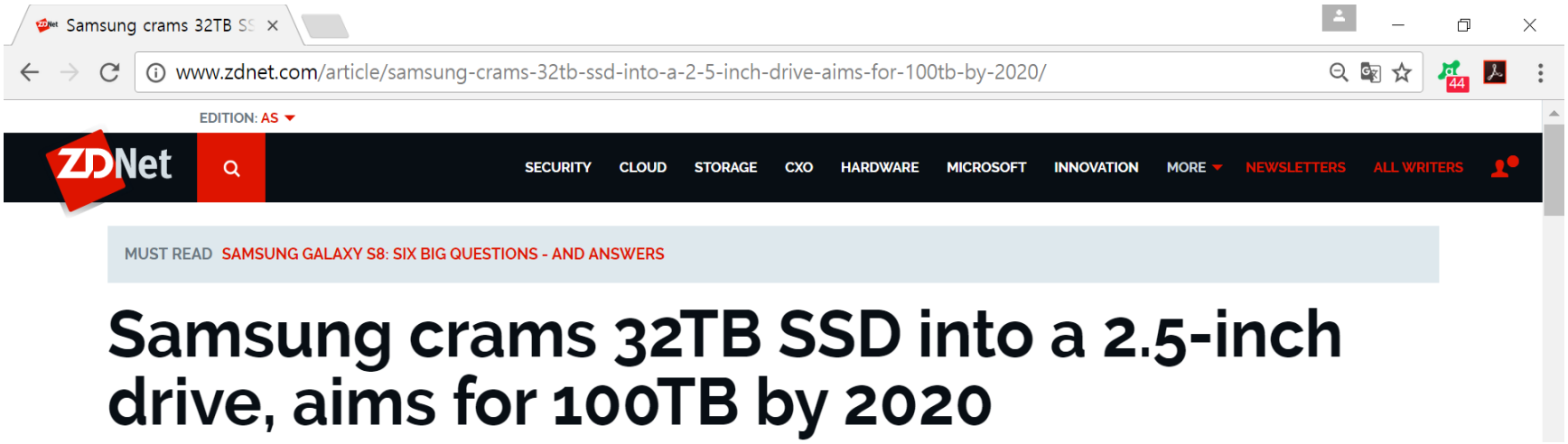
Page-Level Mapping

- Can map any logical page to any physical page
- Efficient Flash page utilization
- Small Garbage Collection overhead

Mapping Table Size Problem

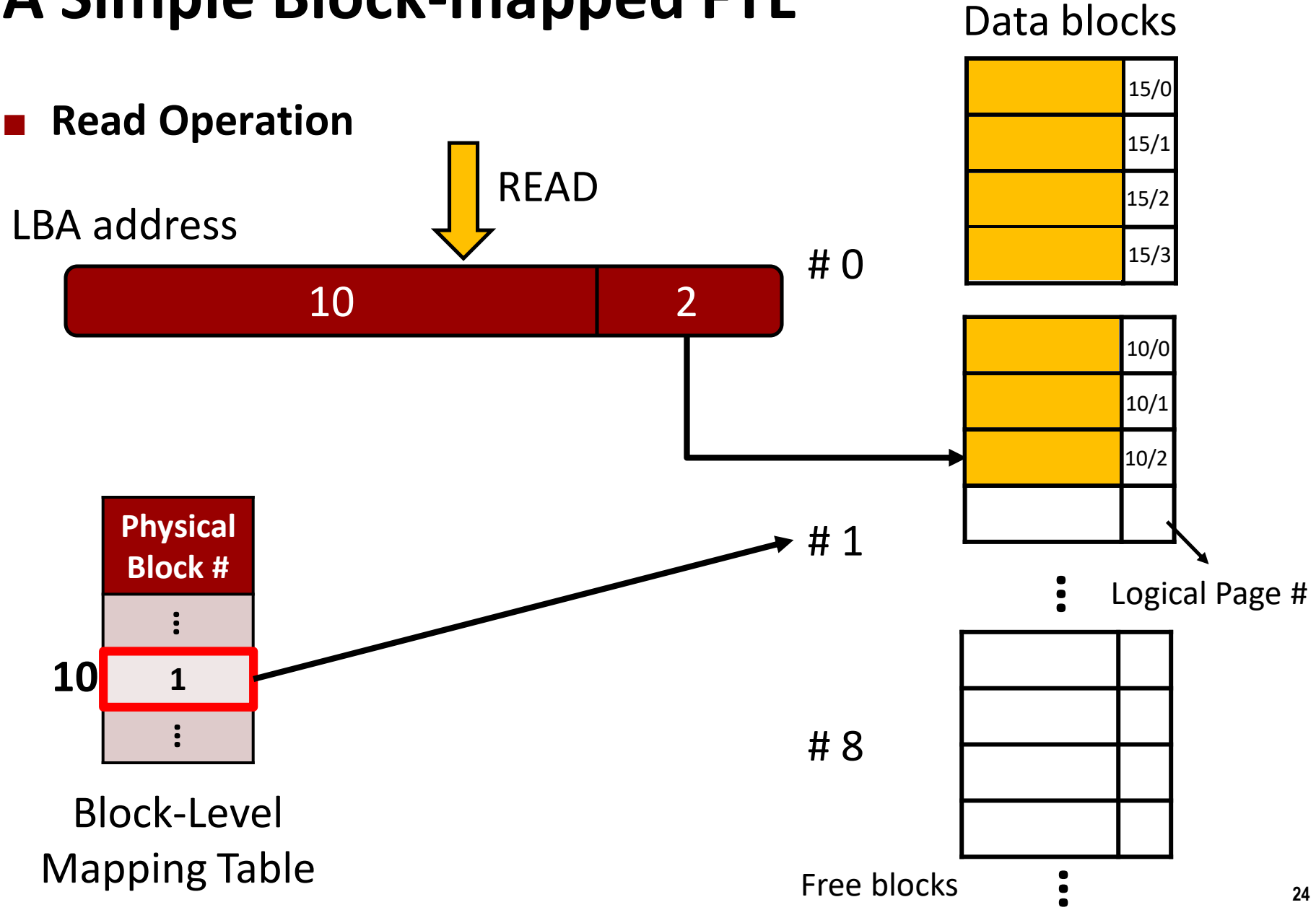
- **With a 8-K page size:**
  - 32TB → 8G (=2^45/2^12) address entries * 4B per entry = 32GB

- **Q: Can we do with a lot less than 16GB?**



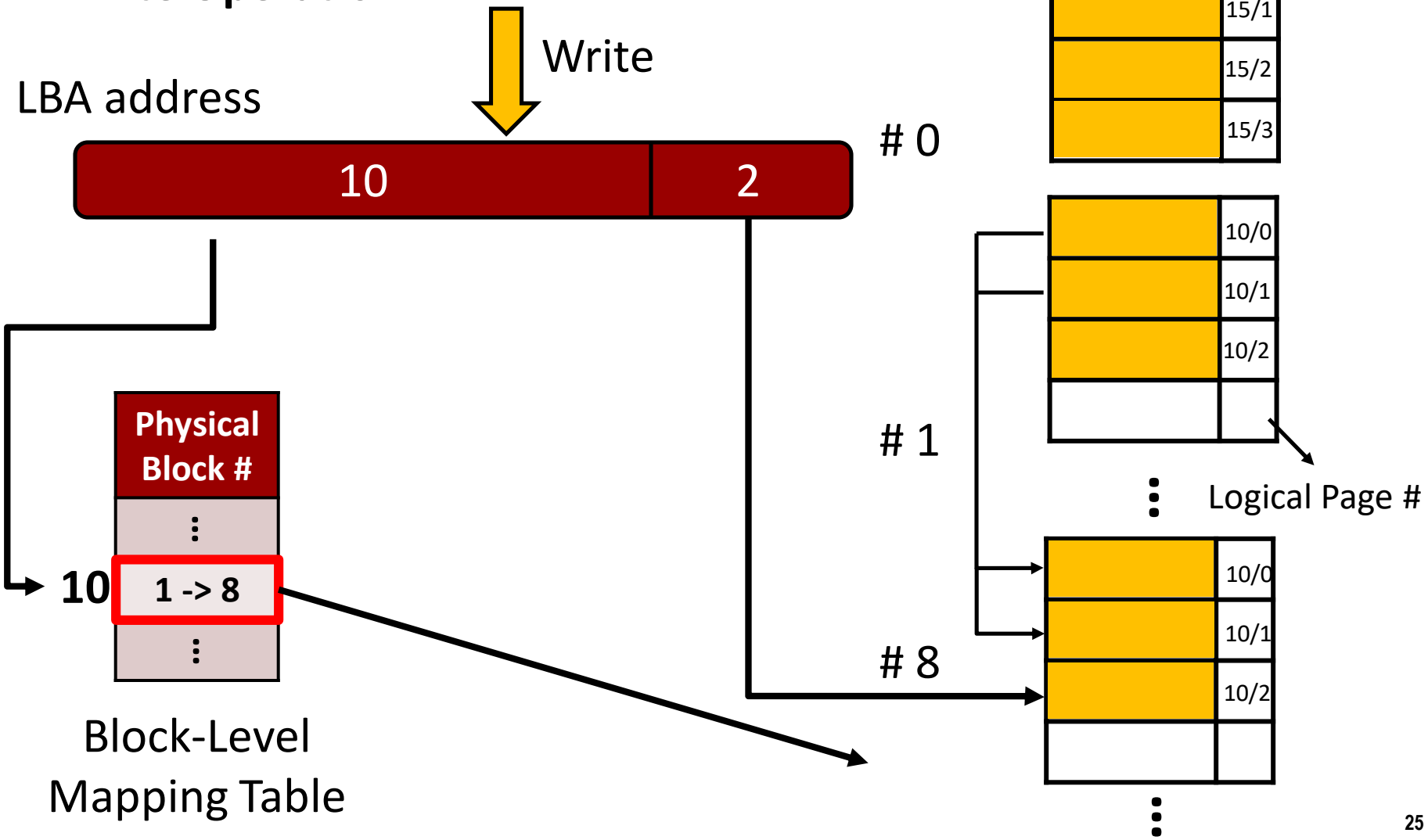
A Simple Block-mapped FTL

- **Read Operation**



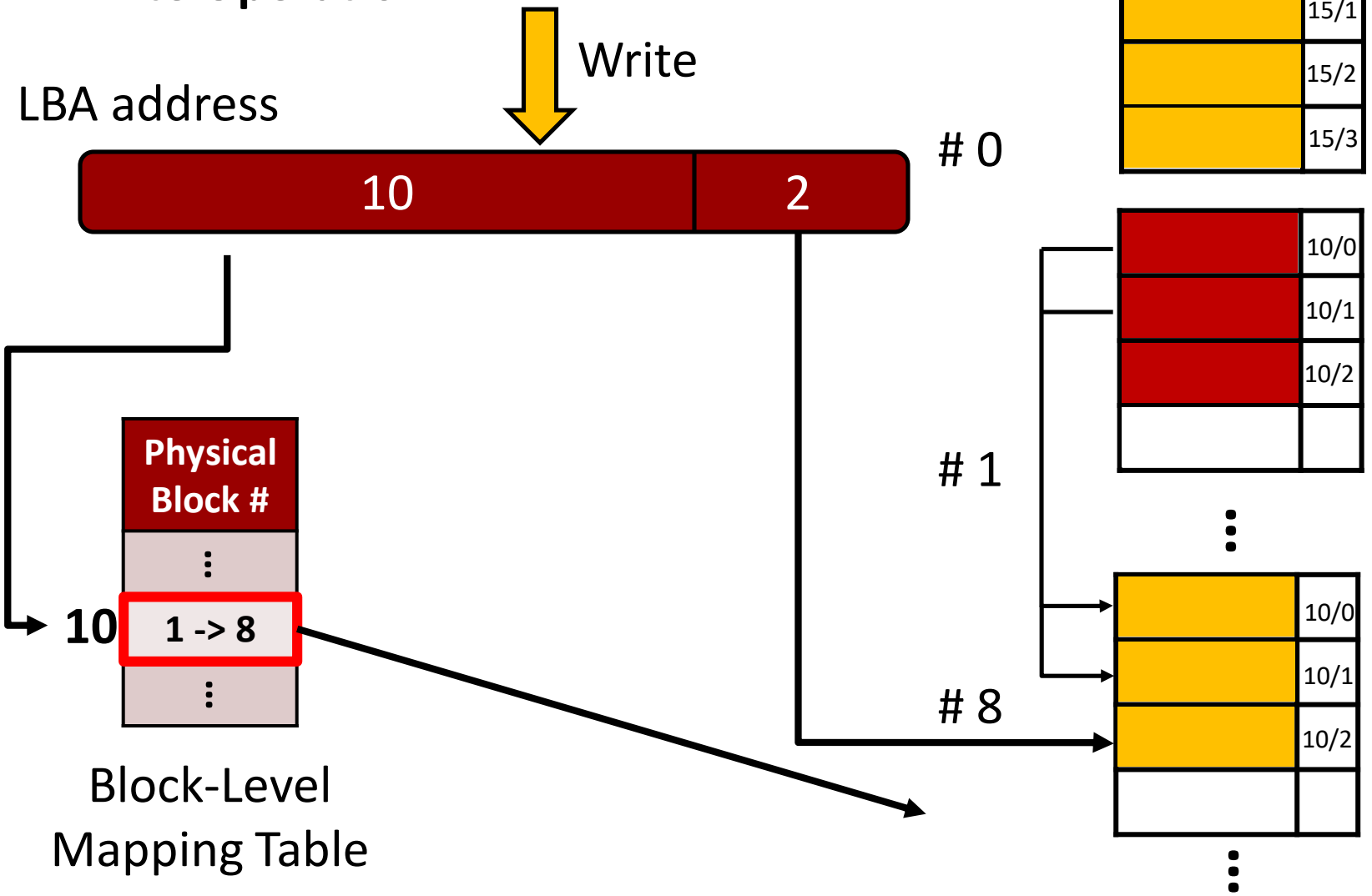
A Simple Block-mapped FTL

- **Write Operation**



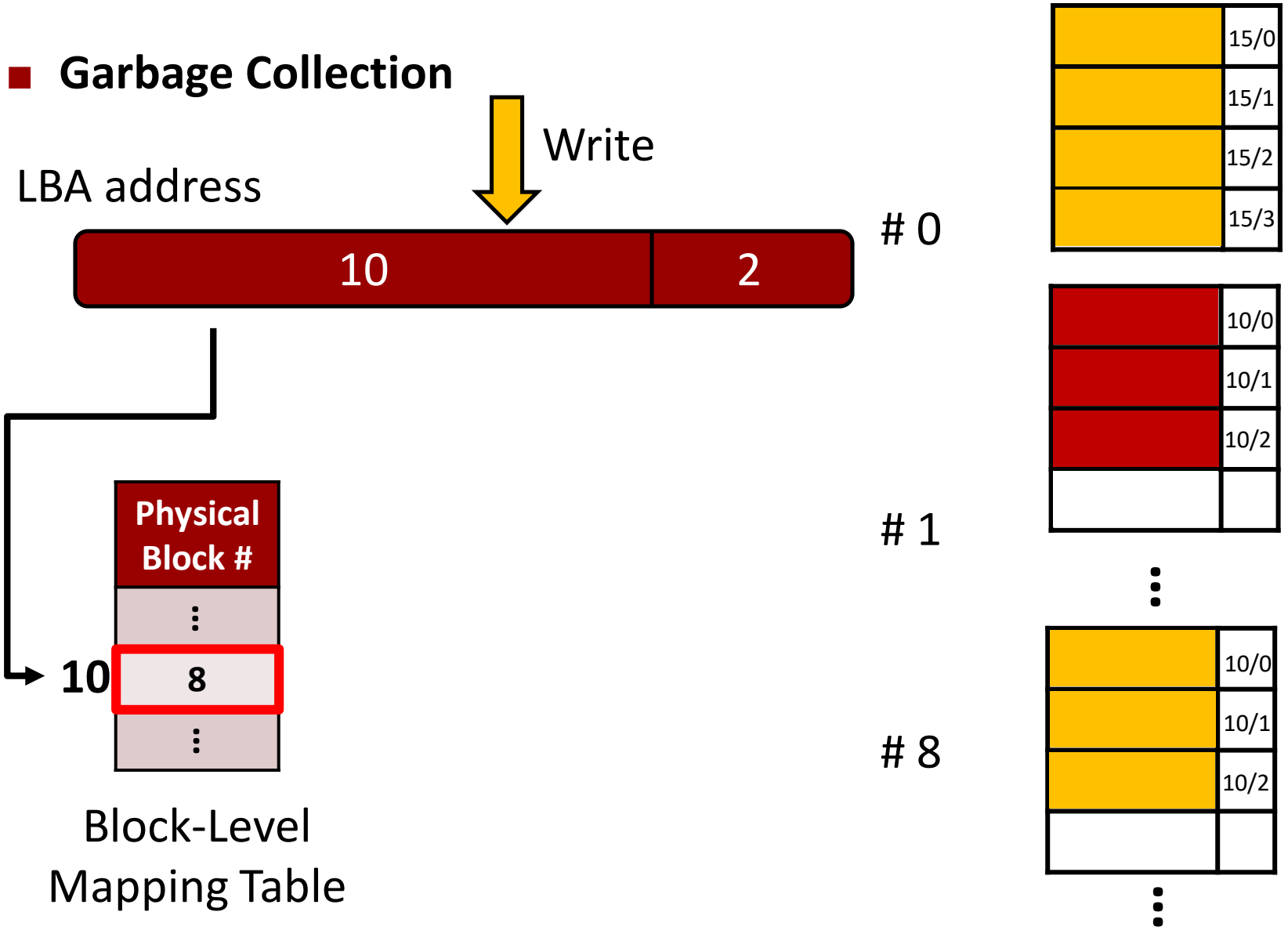
A Simple Block-mapped FTL

- **Write Operation**



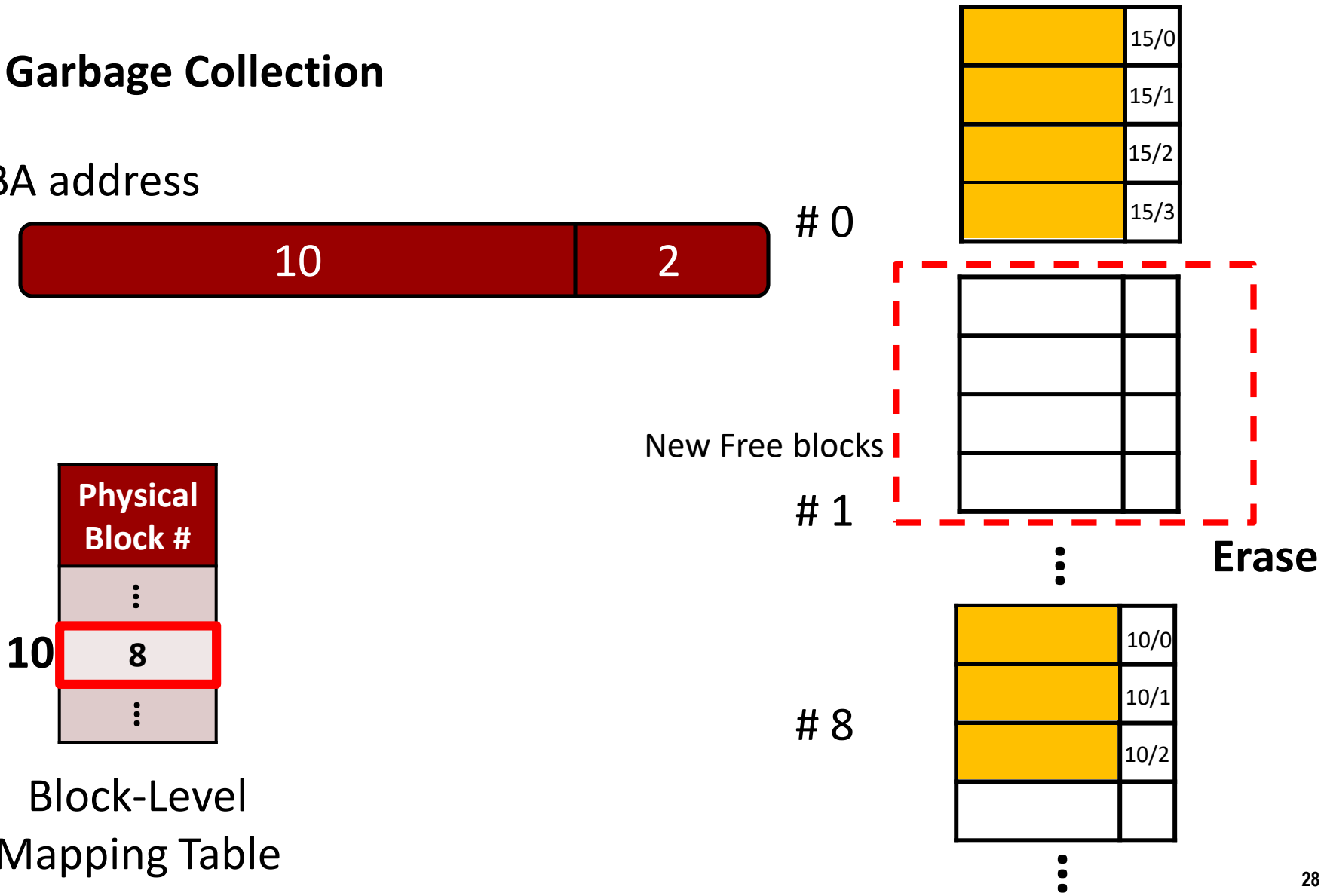
A Simple Block-mapped FTL

- **Garbage Collection**



A Simple Block-mapped FTL

- **Garbage Collection**



Block-Level Mapping

- Requires a much smaller mapping table
- Page offset is fixed
- Low utilization

Challenge in Block-Level Mapping FTLs

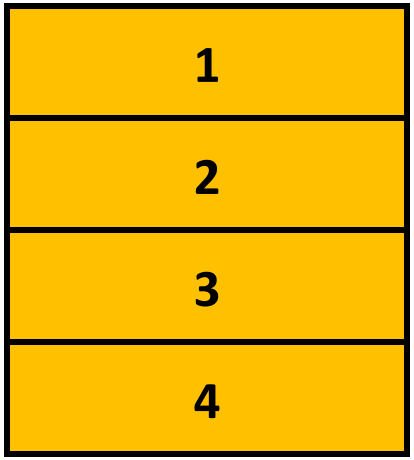
- **Poor small-random write performance**
  - Due to expensive copy operation when only a part of block is modified
- **Various schemes have been introduced**
  - Replacement block scheme
  - Log block scheme
  - Super block scheme
  - FAST and LAST
  - etc.

Replacement Block Scheme

- **Idea**
  - A data block has a chain of write buffer blocks called replacement blocks
  - Mapping within a replacement block is managed in block-level

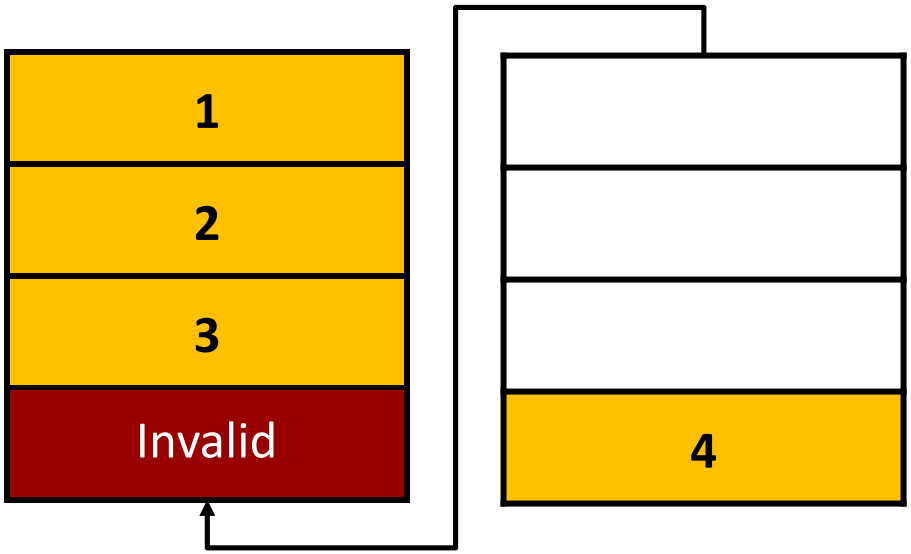
Replacement Block Scheme

- **Replacement-block scheme**
  - Maintain write history between an original block and an updated block
  - E.g., Write trace : 1, 2, 3, 4



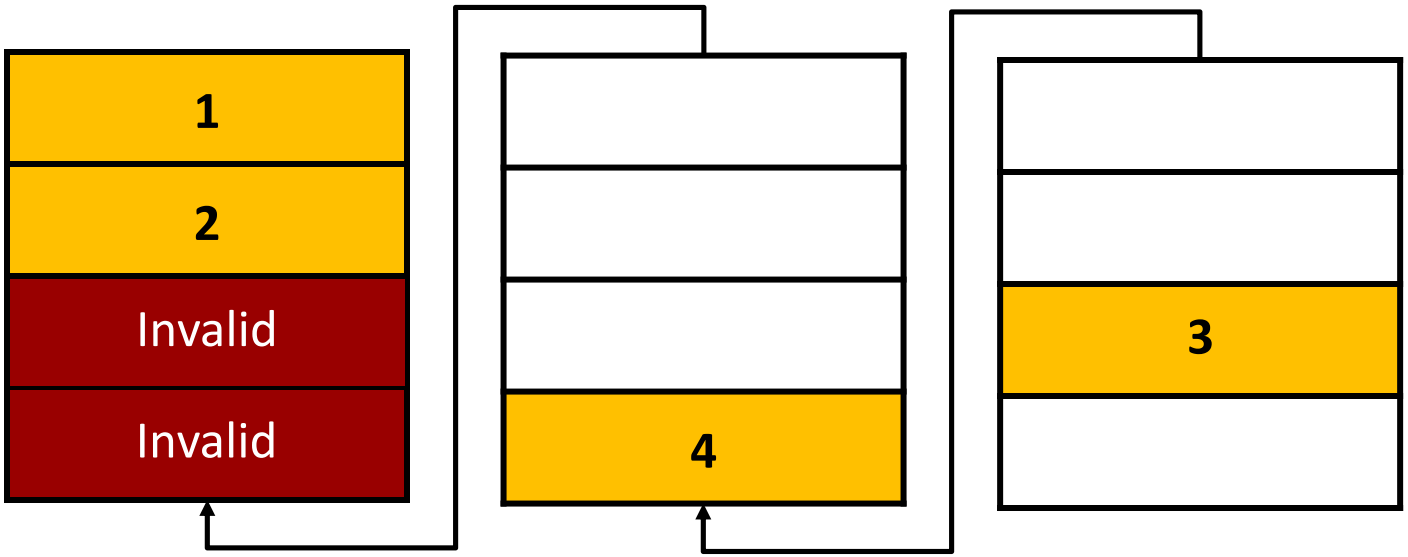
Replacement Block Scheme

- **Replacement-block scheme**
  - Maintain write history between an original block and an updated block
  - E.g., Write trace : 1, 2, 3, 4, 4



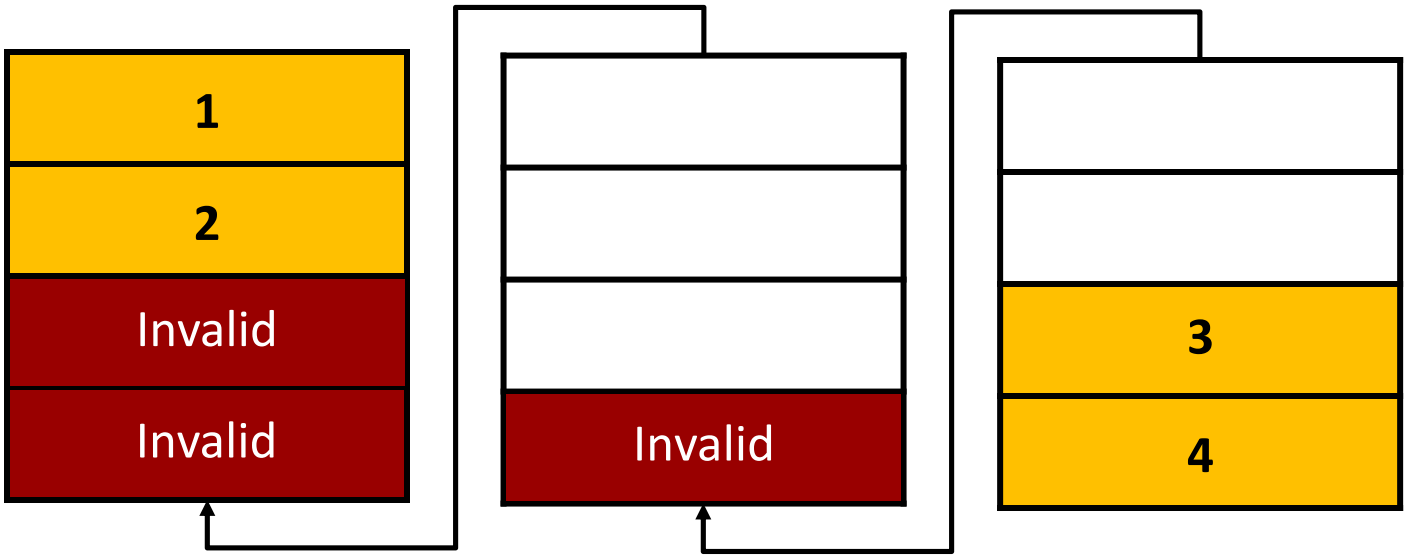
Replacement Block Scheme

- **Replacement-block scheme**
  - Maintain write history between an original block and an updated block
  - E.g., Write trace : 1, 2, 3, 4, 4, 3



Replacement Block Scheme

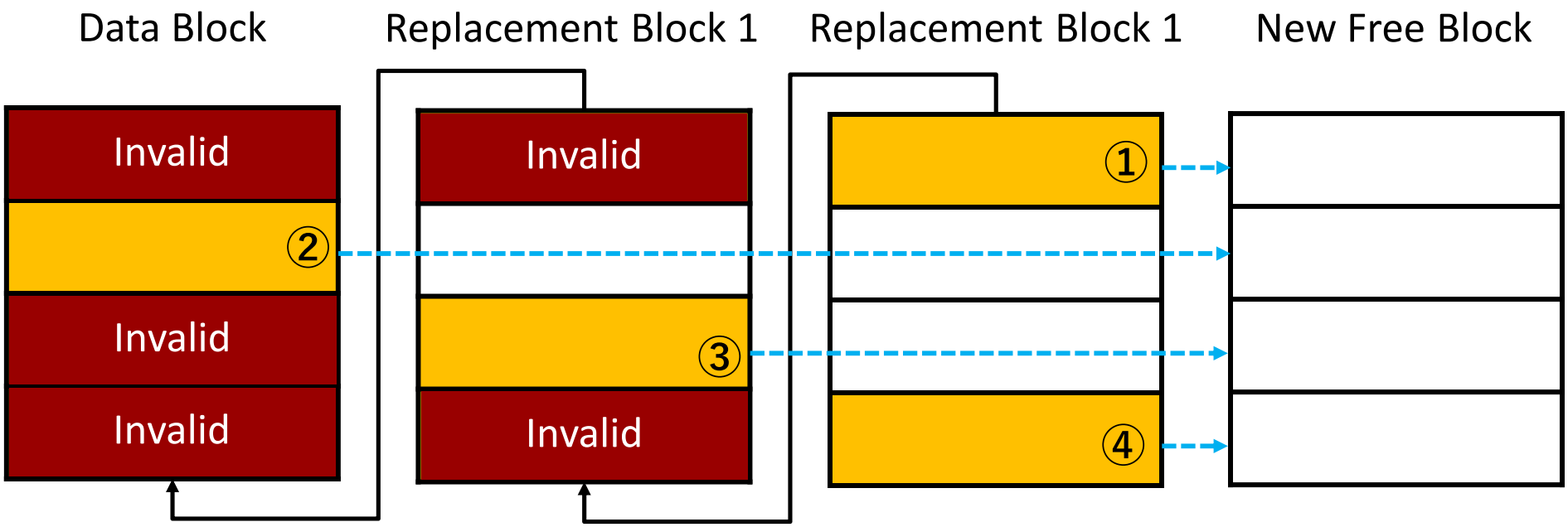
- **Replacement-block scheme**
  - Maintain write history between an original block and an updated block
  - E.g., Write trace : 1, 2, 3, 4, 4, 3, 4



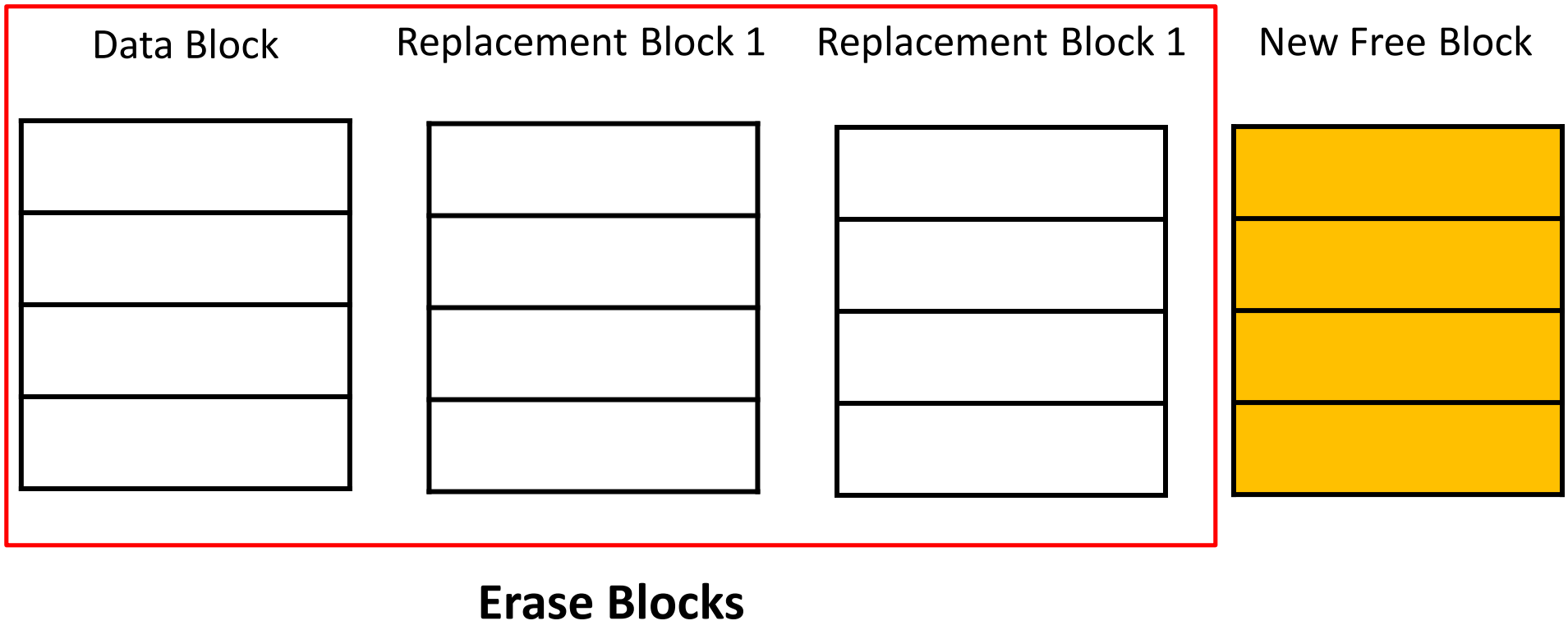
Replacement Block Scheme

- **Merge Operation**
  - Is triggered when there is no free block for a replacement block
  - Gathers valid pages in a data block and write buffer blocks (replacement blocks) to form a single complete data block

Merge Operation

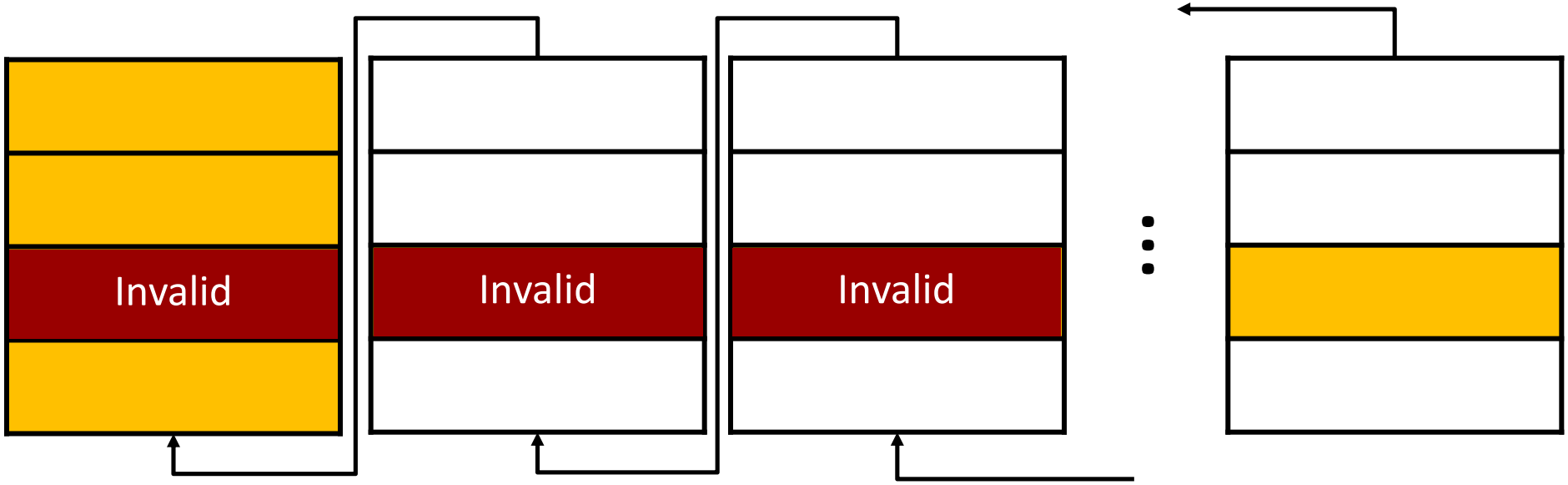


Merge Operation



Replacement Block Scheme

- **Problems**
  - Low utilization of replacement blocks
  - Sequential traverse over replacement blocks during reads and writes
  - No consideration for sequential programming constraint



Hybrid Mapping

- **The main difficulties the FTL faces in giving high performance is the severely constrained size of SRAM**
  - Coarse-grained mapping (Block-level mapping)
    - Small SRAM size / Poor garbage collection efficiency
  - Fine-grained mapping (Page-level mapping)
    - Efficient garbage collection / Lage SRAM size

Hybrid Mapping

	Page Level Mapping	Block level mapping
Characteristic	- Logical page is mapped to physical page - Large mapping table	- Logical block is mapped to physical block - Small mapping table
Pros	- Efficient in handling small size writes	- Small management overhead for maintaining translation information
Cons	- Large management overhead for maintaining translation information	- Less efficient in handling small size writes

Combination of the two different granularities for the better performance

Hybrid Mapping FTLs

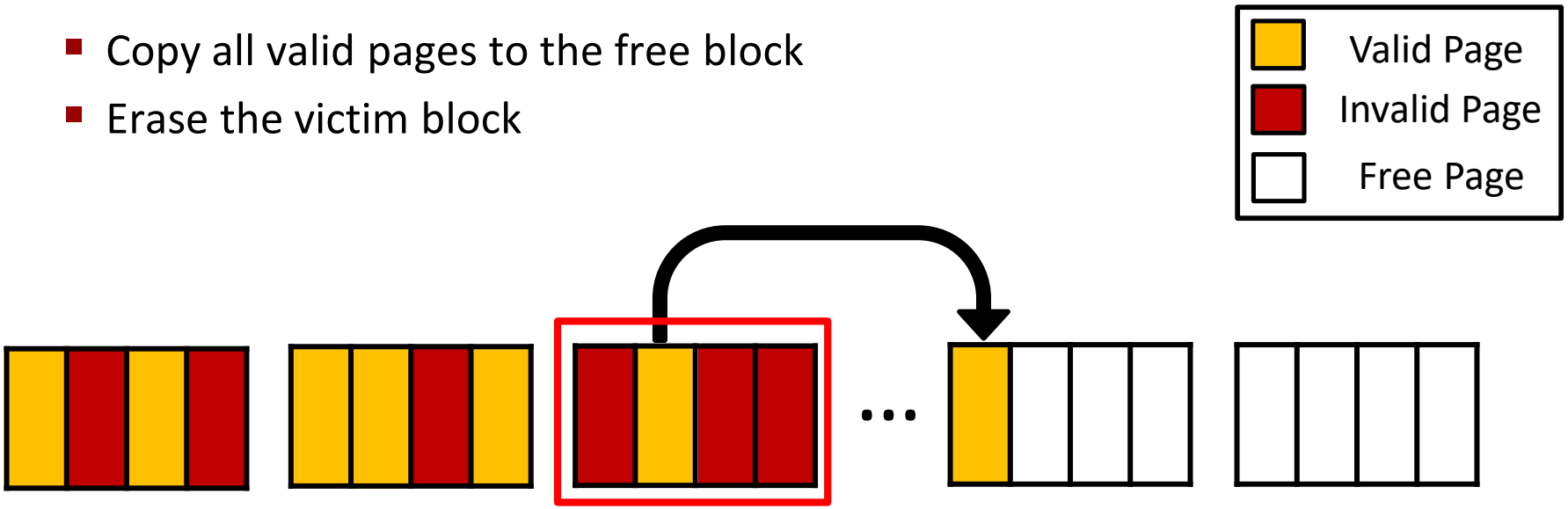
- **Exploits both mapping schemes**
  - Page mapping
    - Update blocks / Log blocks
  - Block mapping
    - Data blocks
- **Garbage Collection Frequency**
  - Page Level << Hybrid <<< Block Level
- **Memory Requirements**
  - Page Level >>> Hybrid > Block Level

Outline

- Review: NAND Cell Array
- Flash Translation Layer
- Address Translation
- **Garbage Collection**

Garbage Collection

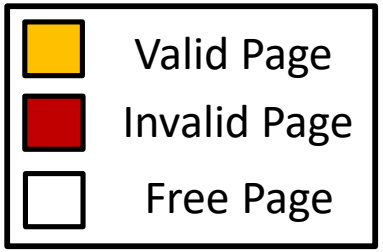
- **The free space is completely exhausted with invalid pages**
- **Need to reclaim the space wasted by invalid data**
  - Select the victim block
  - Copy all valid pages to the free block
  - Erase the victim block

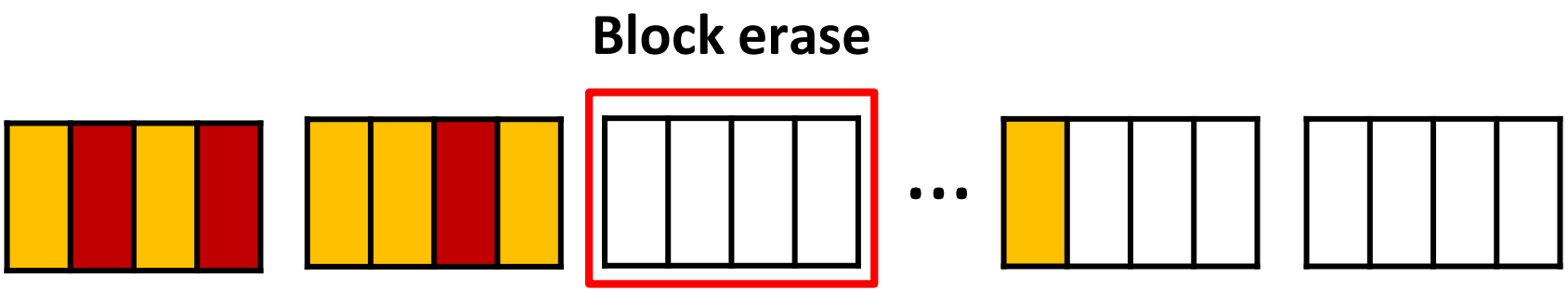


- **Garbage collection overhead = valid page copy + block erase**

Garbage Collection

- **The free space is completely exhausted with invalid pages**
- **Need to reclaim the space wasted by invalid data**
  - Select the victim block
  - Copy all valid pages to the free block
  - Erase the victim block





- **Garbage collection overhead = valid page copy + block erase**

Garbage Collection Overhead

- **Garbage collection incurs many valid page copies and block erasures**
  - Increase the overall response time of user I/O requests
  - Increase the number of P/E (program/erasure) cycles

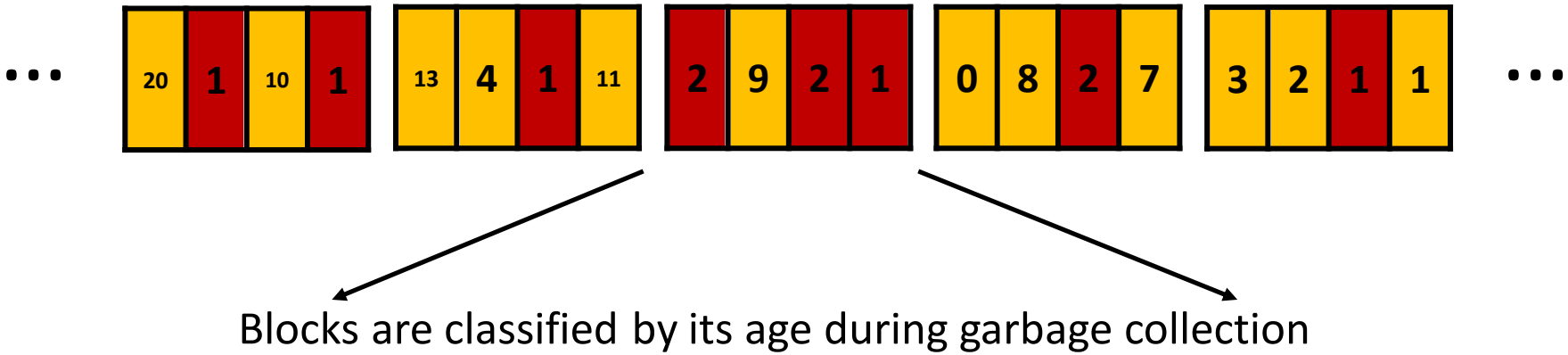
- **Our goal is to reduce the extra operations caused by garbage collection**

Technical Issues in Garbage Collection

- **How to organize valid data**
  - Where the user data is written → **Hot and cold separation policy**
- **Which block to reclaim**
  - Which block is preferred for garbage collection → **Victim block selection policy**
- **When to begin**
  - When there are no free blocks → **On-demand garbage collection**
  - When there are sufficient idle times → **Background garbage collection**

Hot and Cold Separation Policy

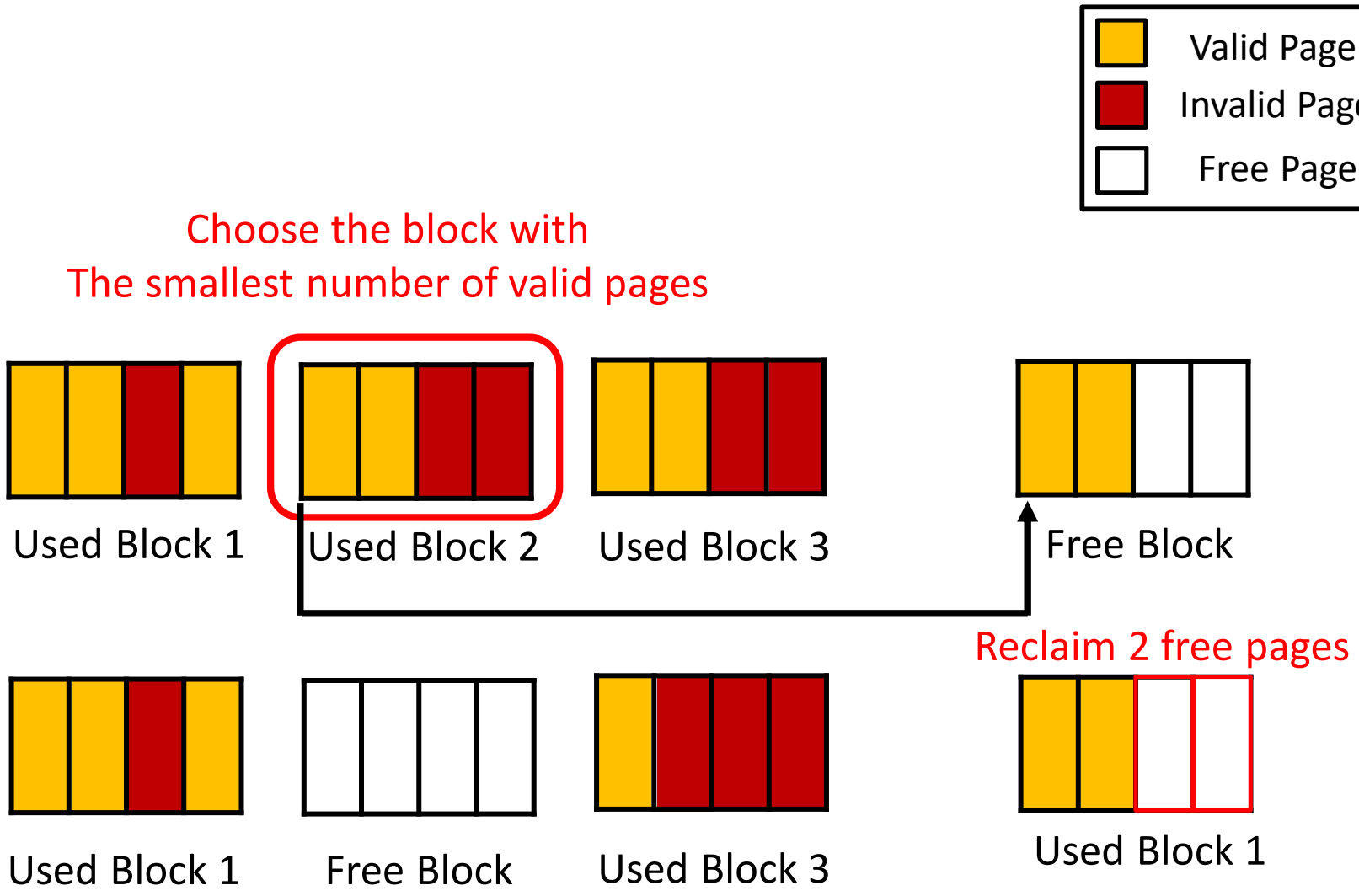
- **Basic Idea : Age-based Separation**
  - Consider the locality of reference
    - Blocks containing ‘hot’ data (e.g., 1 & 2) tend to be invalidated more rapidly



Victim Selection Policy

- **Greedy Policy**
  - Principle : choose the least utilized block to clean
  - Pros : work well under workloads with uniform access pattern
  - Cons : do not perform well when there's high locality of writes

Greedy Policy - Example



Victim Selection Policy

- **Cost-Benefit Policy**
  - Principle : chooses a block that minimizes the equation below
  - Pros : Perform well with update locality
  - Cons : Computation / Data overhead

$$\frac{Cost}{Benefit} = \frac{u}{(1-u) * Age}$$

- u : utilization of the block (# of valid pages)
- Age : the most recent modified time of any page in the block

Age Transformation Function

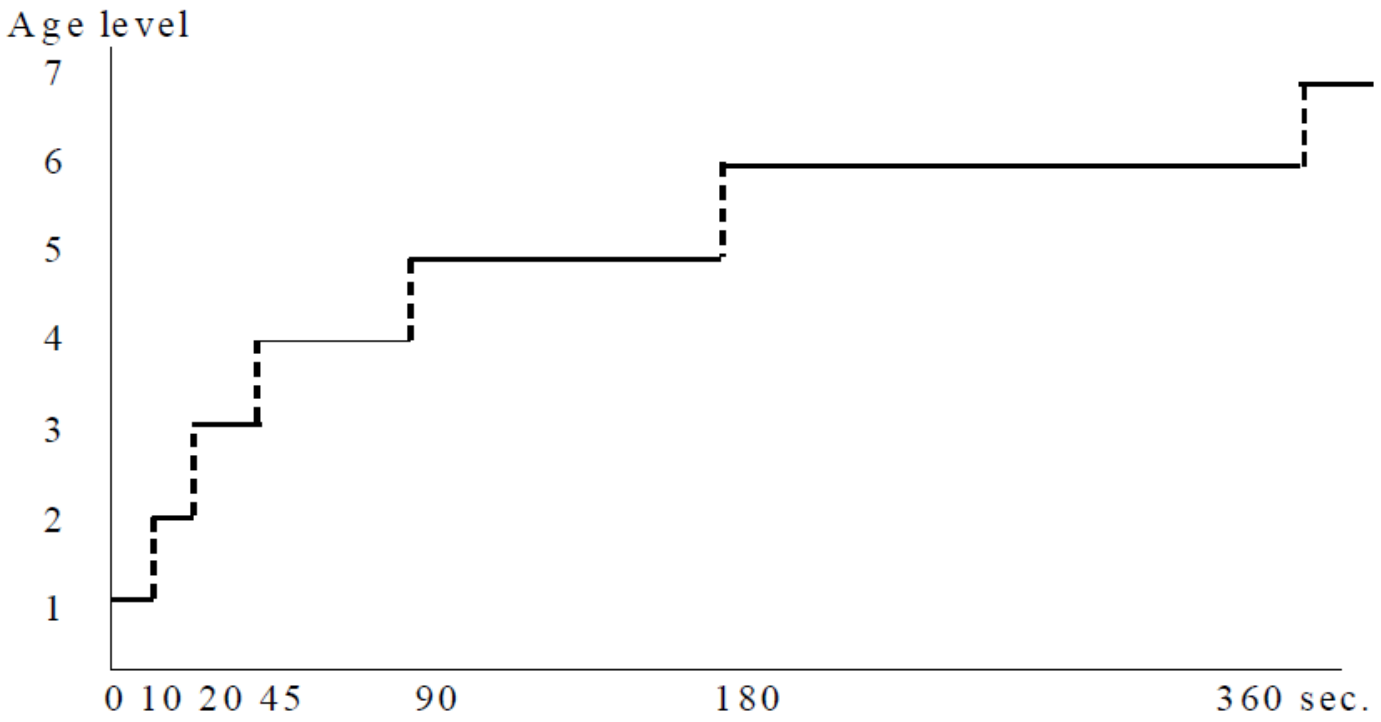
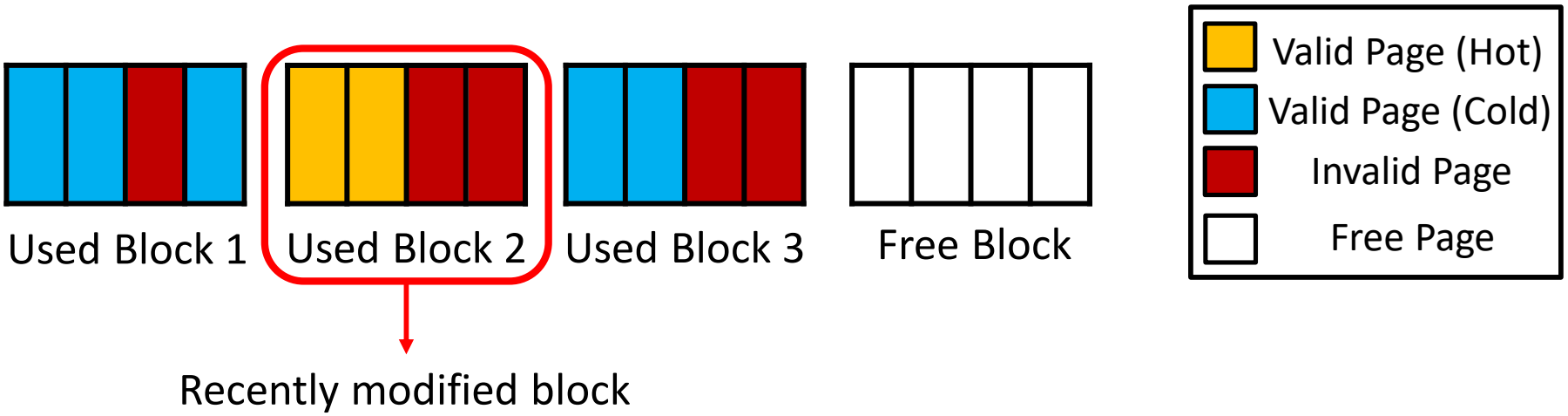


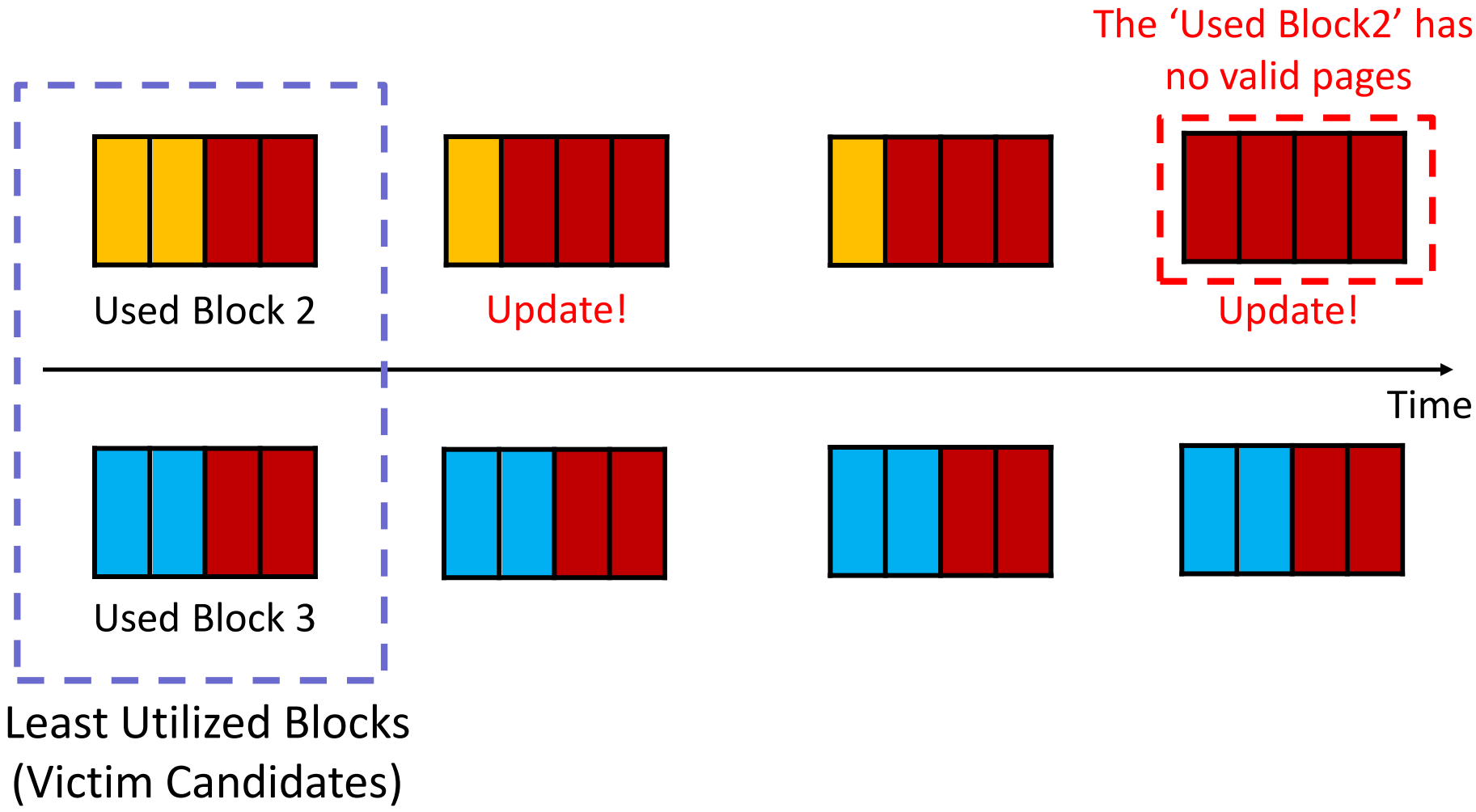
Figure 7: Age transformation function.

Cost-Benefit - Example

- **Used Blocks 2 and 3 have the least block utilization**
- **Chooses ‘ Used Block 3’ as a victim block because it holds many cold pages**

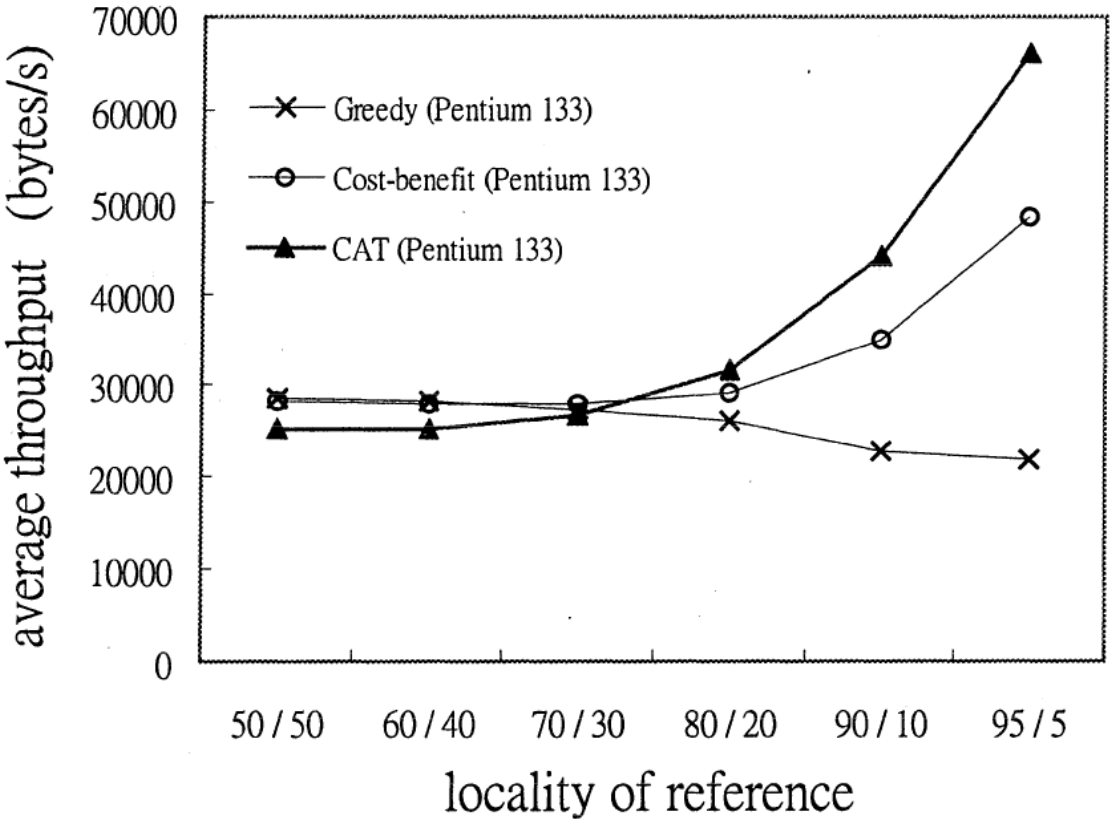


Cost-Benefit - Example



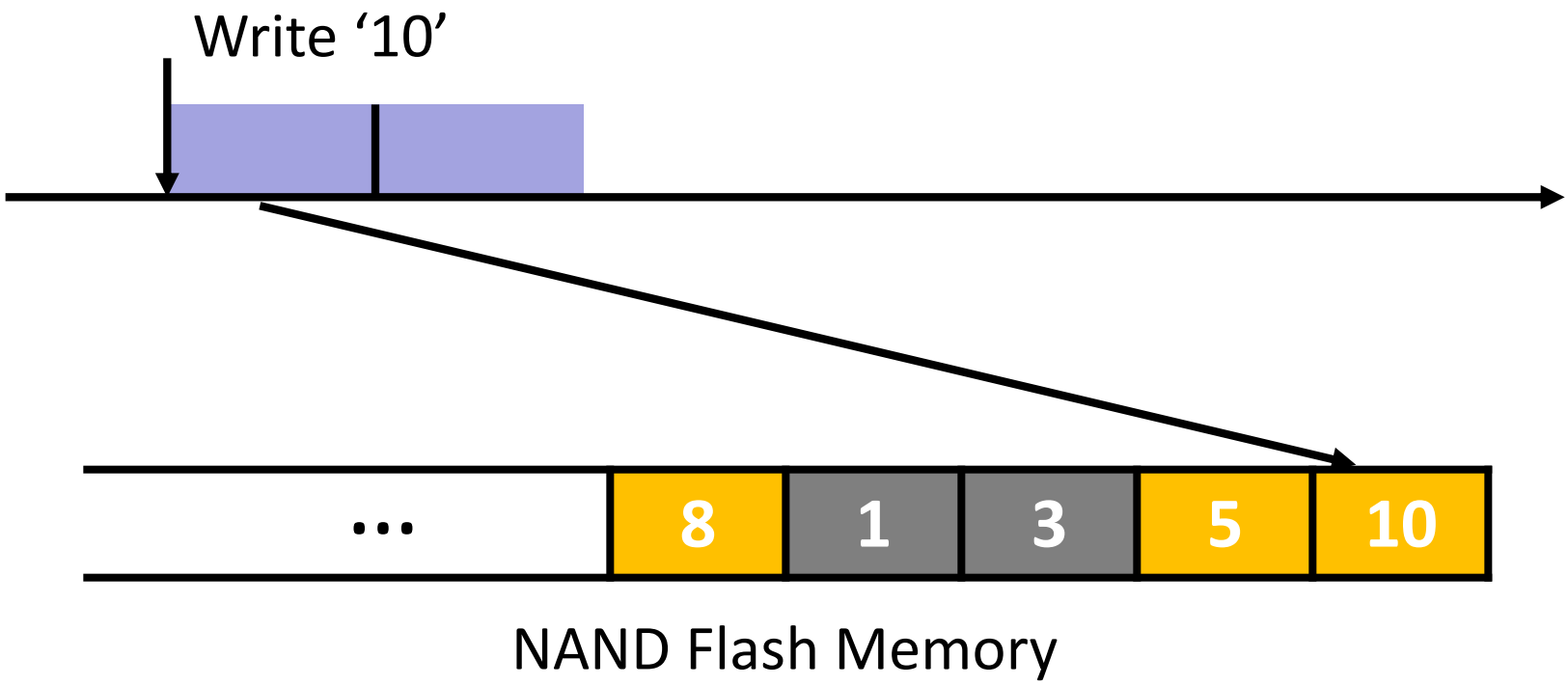
Experimental Results

- **Average throughput**



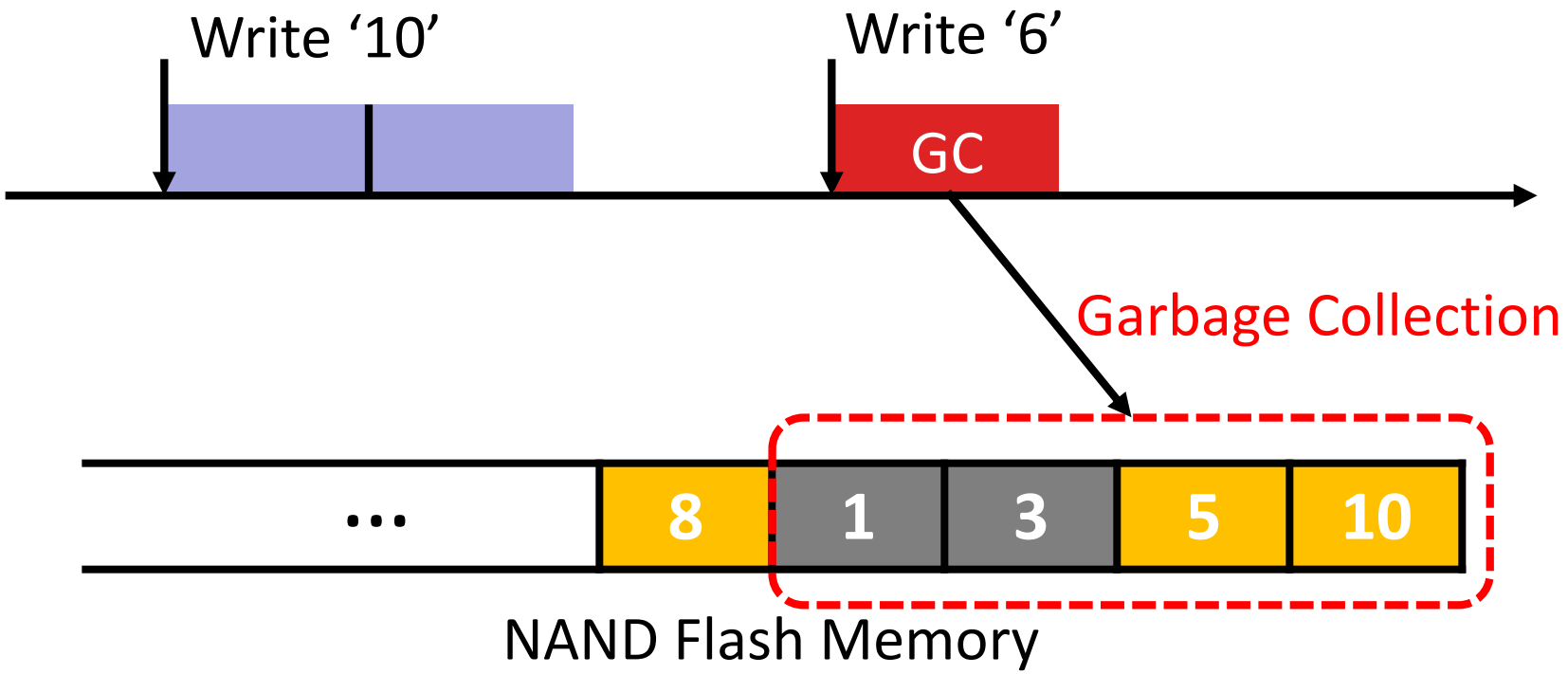
On-Demand Garbage Collection

- **Perform garbage collection when there are no free blocks in flash memory**



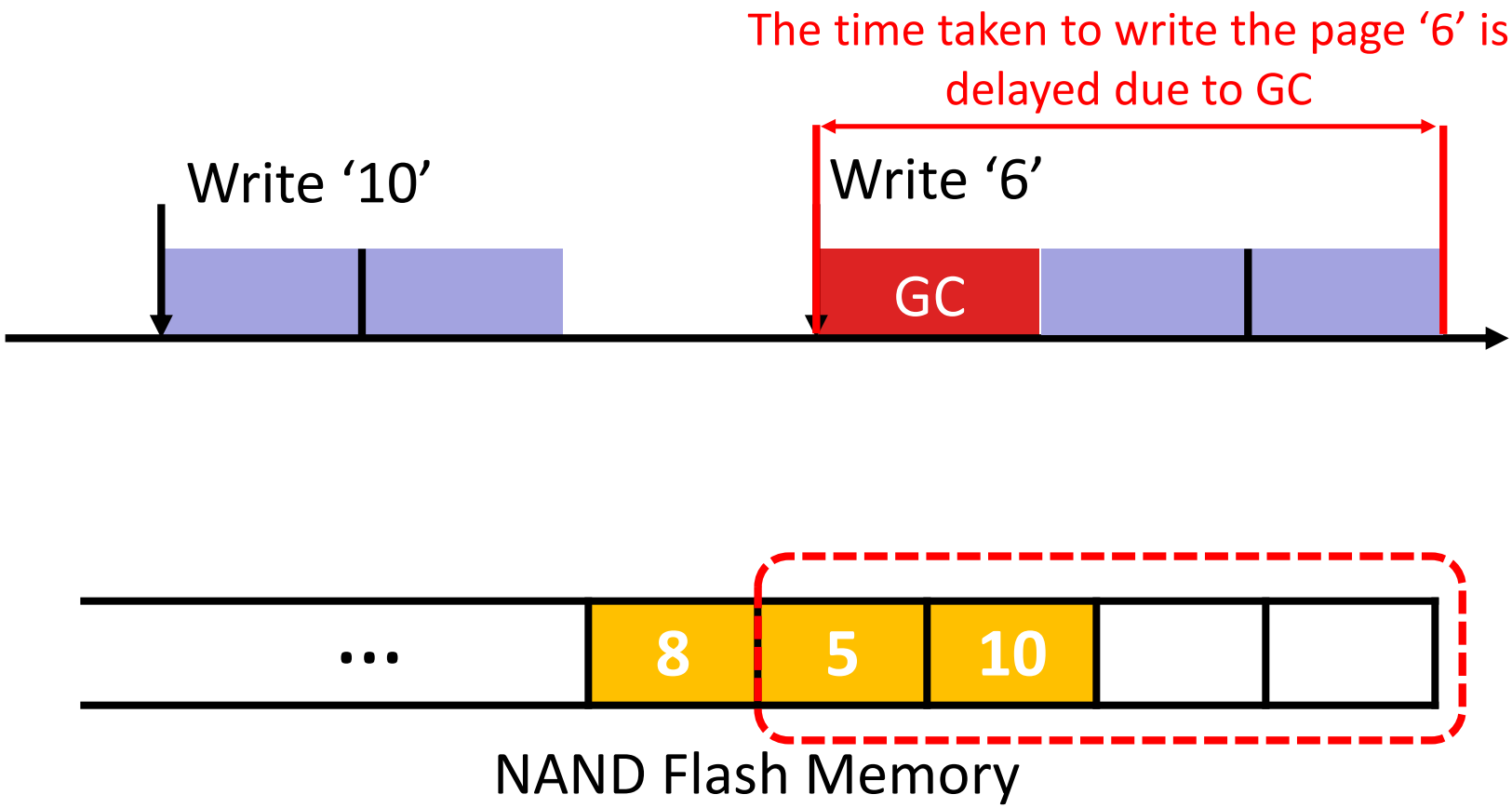
On-Demand Garbage Collection

- **Perform garbage collection when there are no free blocks in flash memory**



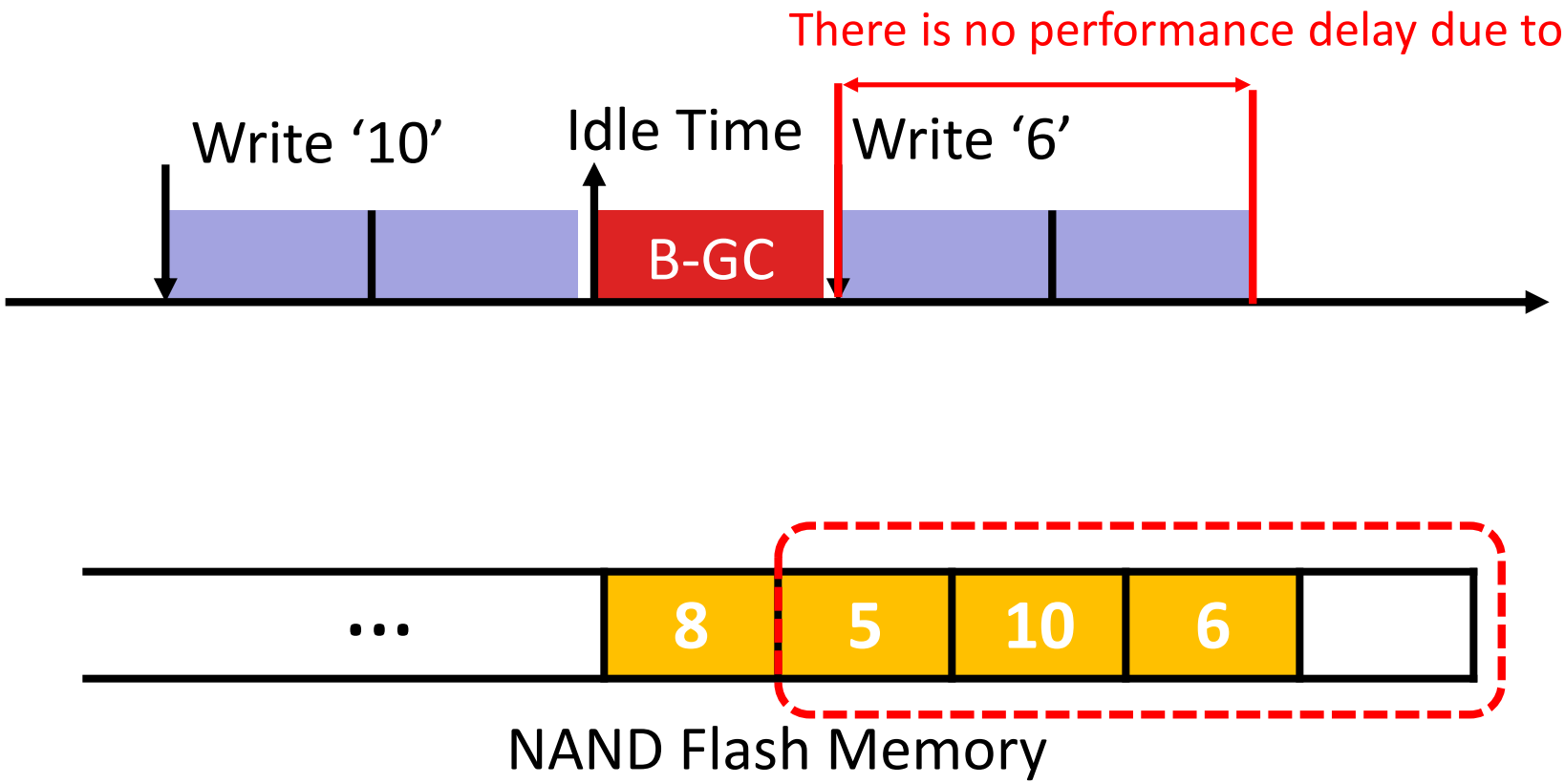
On-Demand Garbage Collection

- **Perform garbage collection when there are no free blocks in flash memory**



Background Garbage Collection (B-GC)

- **Perform garbage collection when there are available idle times**

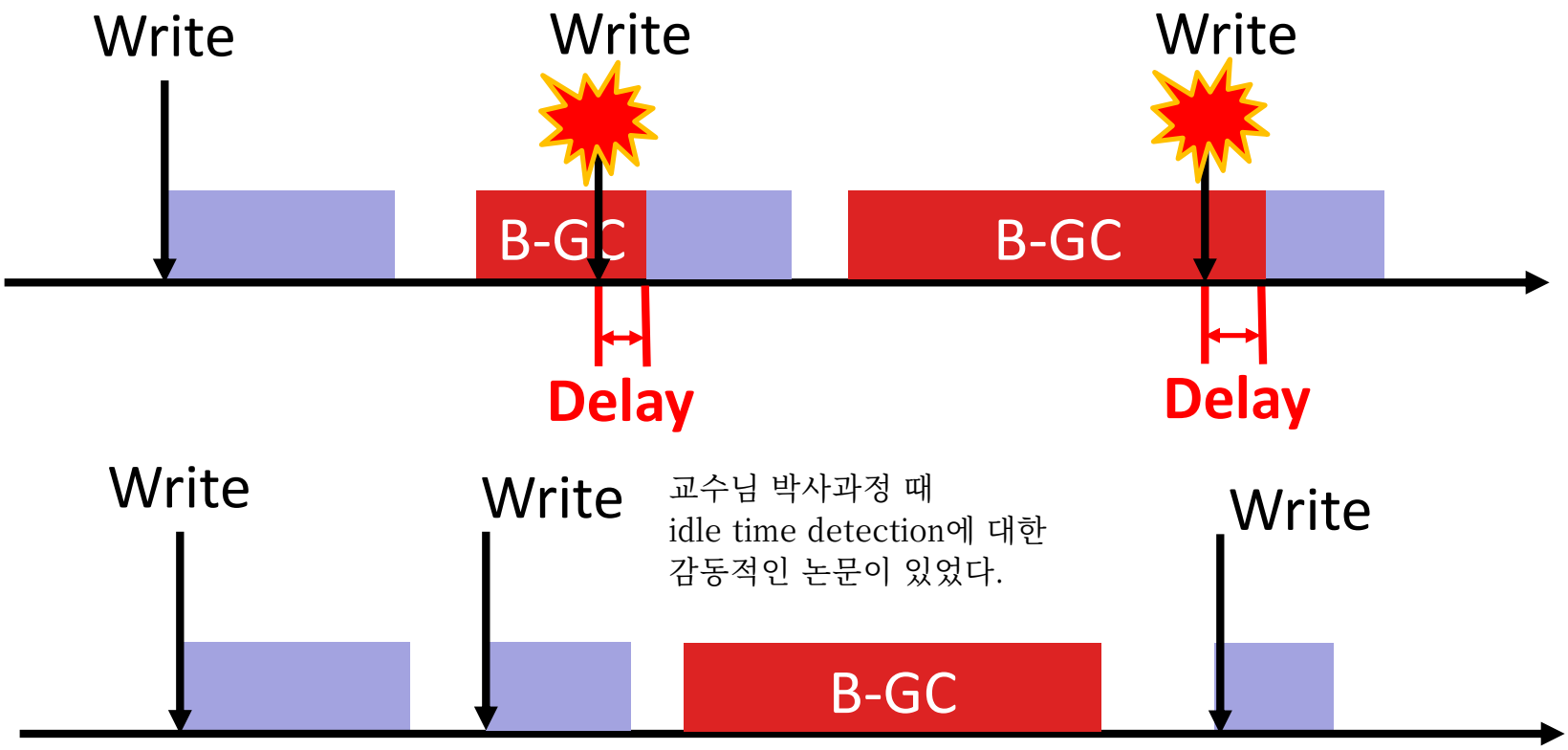


Challenges in B-GC

- **When a background garbage collector starts and stops**
  - Garbage Collection scheduling
- **How many over-provisioned pages are maintained**
  - Capacity over-provisioning
- **…**

Garbage Collection Scheduling

- **Garbage collection must be carefully started and stop**



End of Chapter 4