香港中文大學
The Chinese University of Hong Kong

CSCI5550 Advanced File and Storage Systems

Lecture 06: Flash Memory

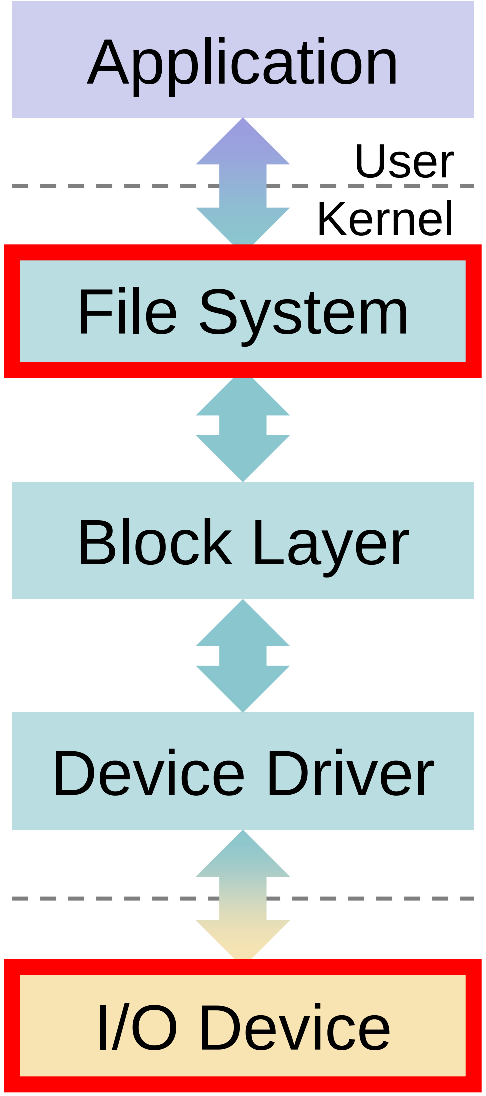
Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Outline

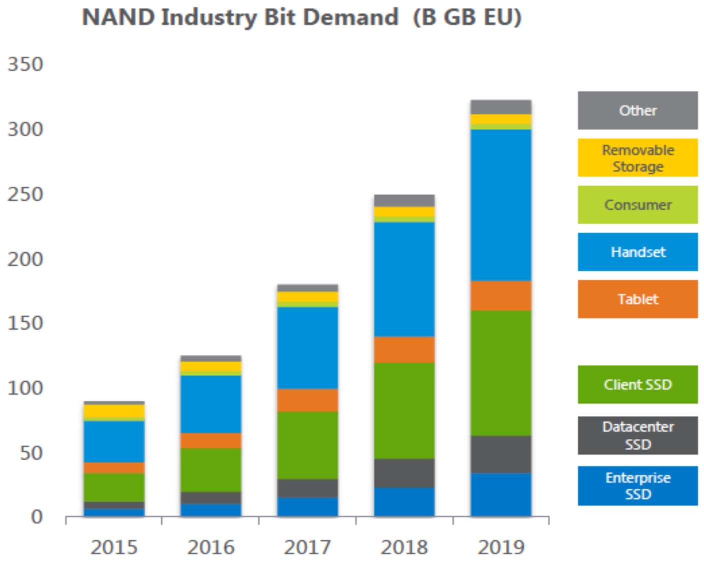
- Flash Memory: Why and How
  - NAND Flash Technology
  - Inherent Challenges
- System Architecture
- Flash Translation Layer
  - Address Mapping
  - Garbage Collection
  - Wear Leveling
  - Multilevel I/O Parallelism
- Flash-aware File System
  - Flash-Friendly File System (F2FS)

I/O Stack

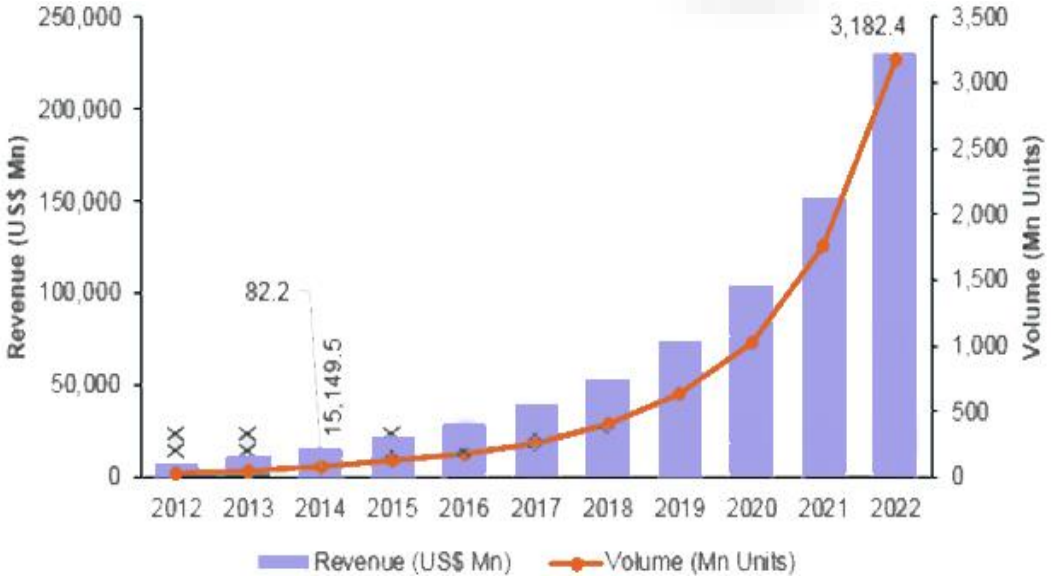


Why Flash Memory

- Flash memory is a widely used memory/storage technology in today's products.
  - – It is a type of non-volatile memory.
    - Data can be **persisted** under power loss.
  - – **Revenue Growth**: 40% every year!
  - – **Volume Price**: ~40% annual reduction!



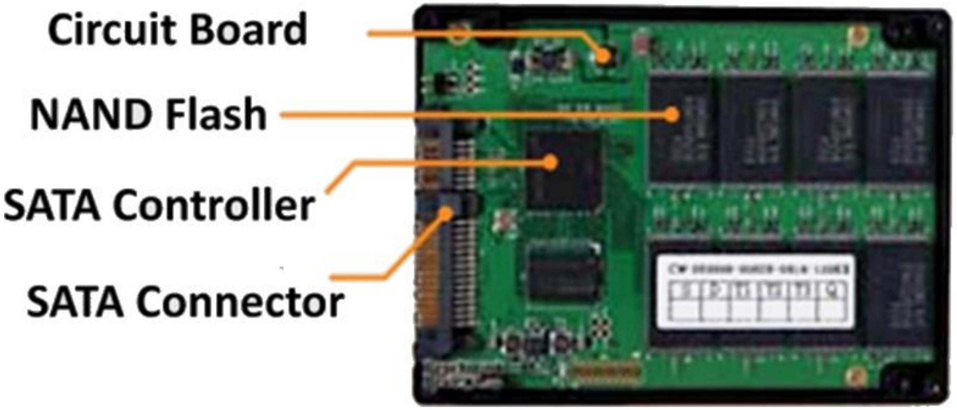
Sources : Micron, Intel And 3D NAND Post [link]



Sources : Global SSD Market to Post CAGR of 41% Until 2022, OriginStorage

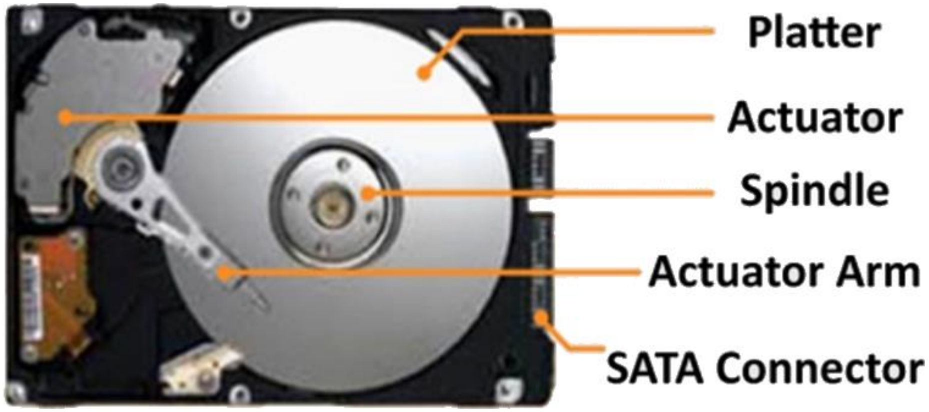
Solid-State Drive vs. Hard Disk Drive

Solid-State Drive (SSD)



- ✓ Faster performance
- ✓ No vibrations or noise
- ✓ Shock resistance
- ✓ More energy efficient
- ✓ Lighter and smaller

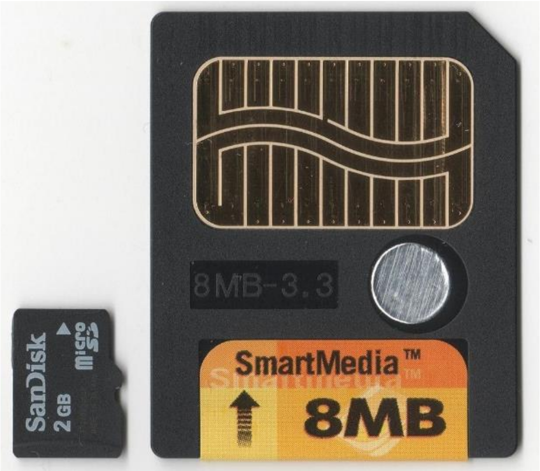
Hard Disk Drive (HDD)



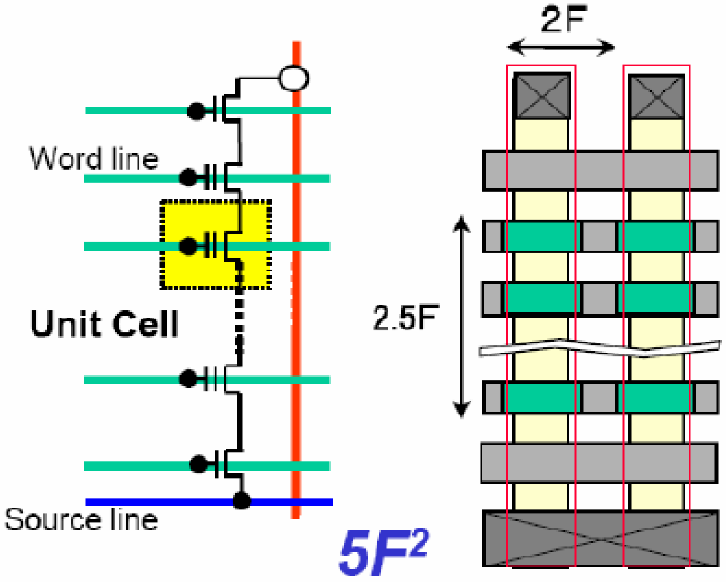
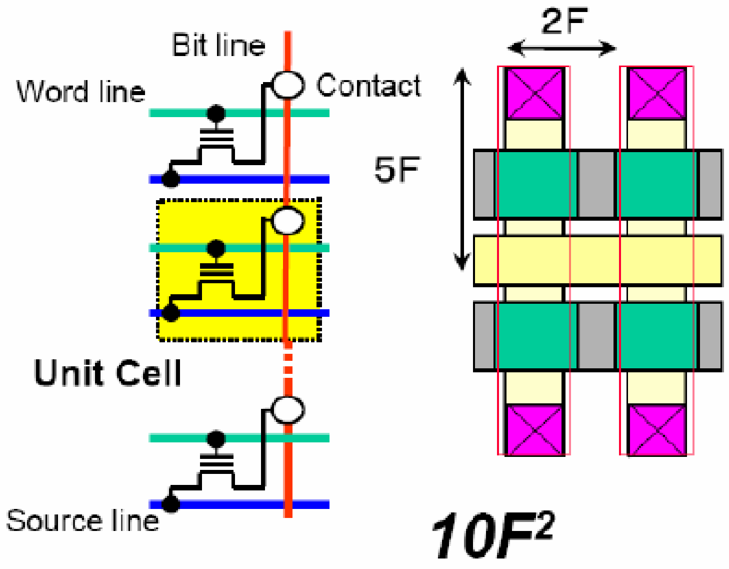
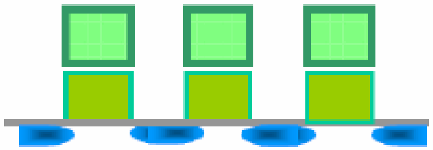
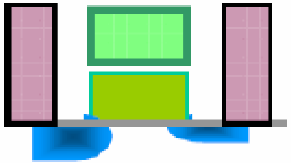
- ✓ Cheaper per GB

The Greatest Invention in 1990s

- Invented by **Dr. Fujio Masuoka (舛岡　富士雄)** born in 1943, while working for Toshiba around 1980.
- First presented in *IEEE International Electron Devices Meeting*, 1984.
- First **NAND flash** first introduced as **SmartMedia storage** in 1995.
- First **NOR flash** commercialized by Intel in 1998 to replace the read-only memory (ROM) to store **BIOS** and **firmware**.



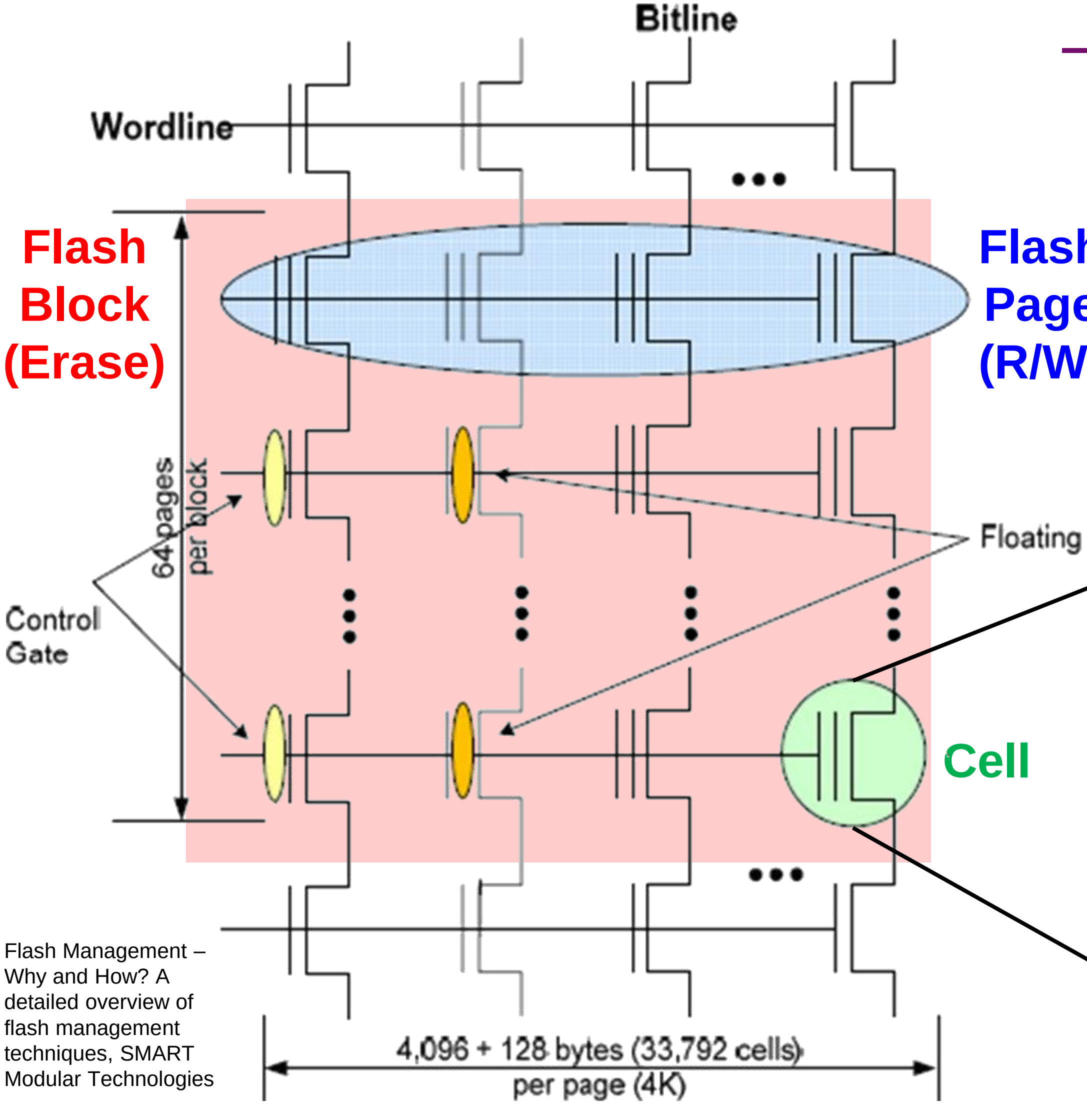
NAND Flash vs. NOR Flash

	NAND	NOR
Cell Array & Size	 2F Word line Unit Cell 2.5F Source line $5F^2$	 Bit line Word line Contact 2F 5F Unit Cell Source line $10F^2$
Cross-section		
Features	**Small Cell Size, High Density** **Low Power & Good Endurance** ➔ ***Mass Storage***	**Large Cell Current,** **Fast Random Access** ➔ ***Code Storage***

https://user.eng.umd.edu/~blj/CS-590.26/nand-presentation-2010.pdf

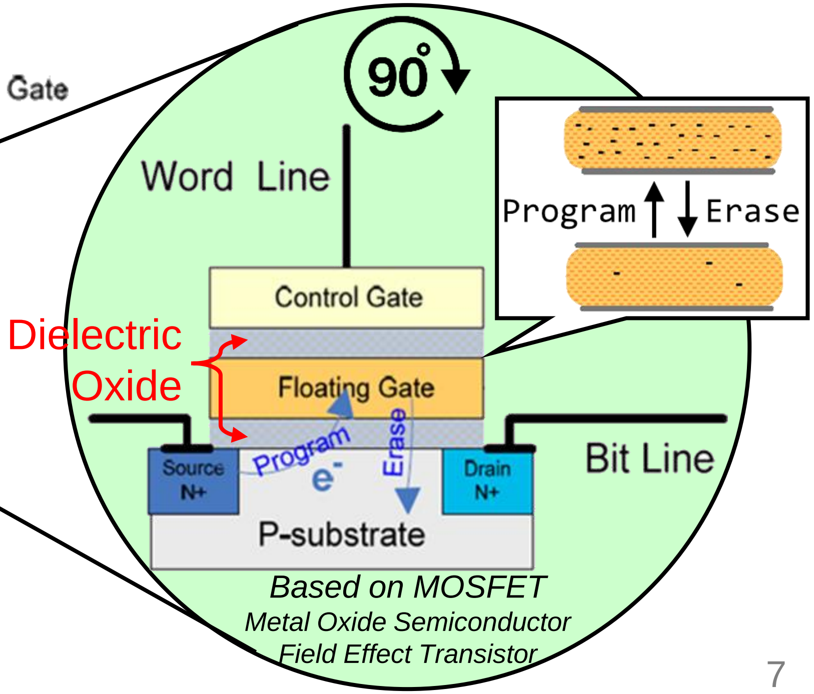
NAND Flash Technology

- NAND Flash Array



Flash Management – Why and How? A detailed overview of flash management techniques, SMART Modular Technologies

- NAND Flash Cell
  - Floating-Gate Transistor
    - **Program**: Inject electrons into FG to raise voltage
    - **Erase**: Remove electrons from FG to lower voltage
    - **Read**: Sense voltage of FG

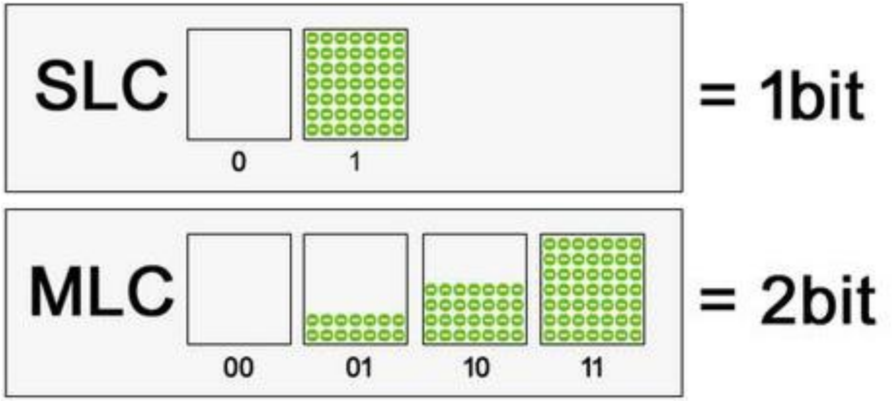


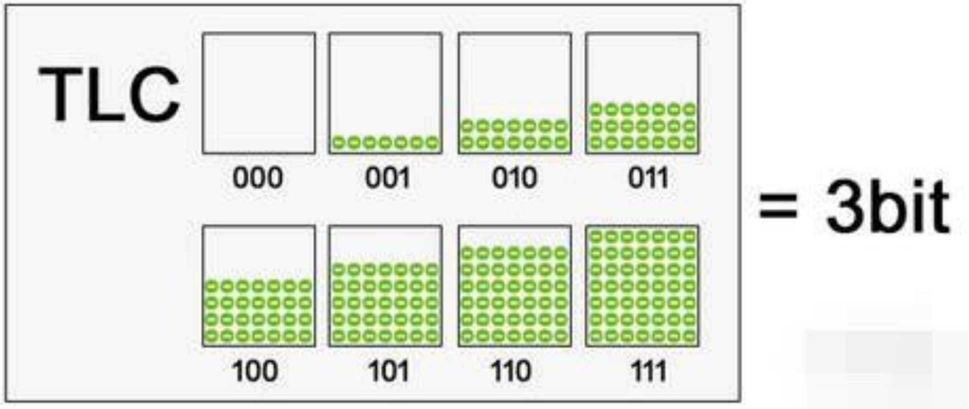
Based on MOSFET

Metal Oxide Semiconductor Field Effect Transistor

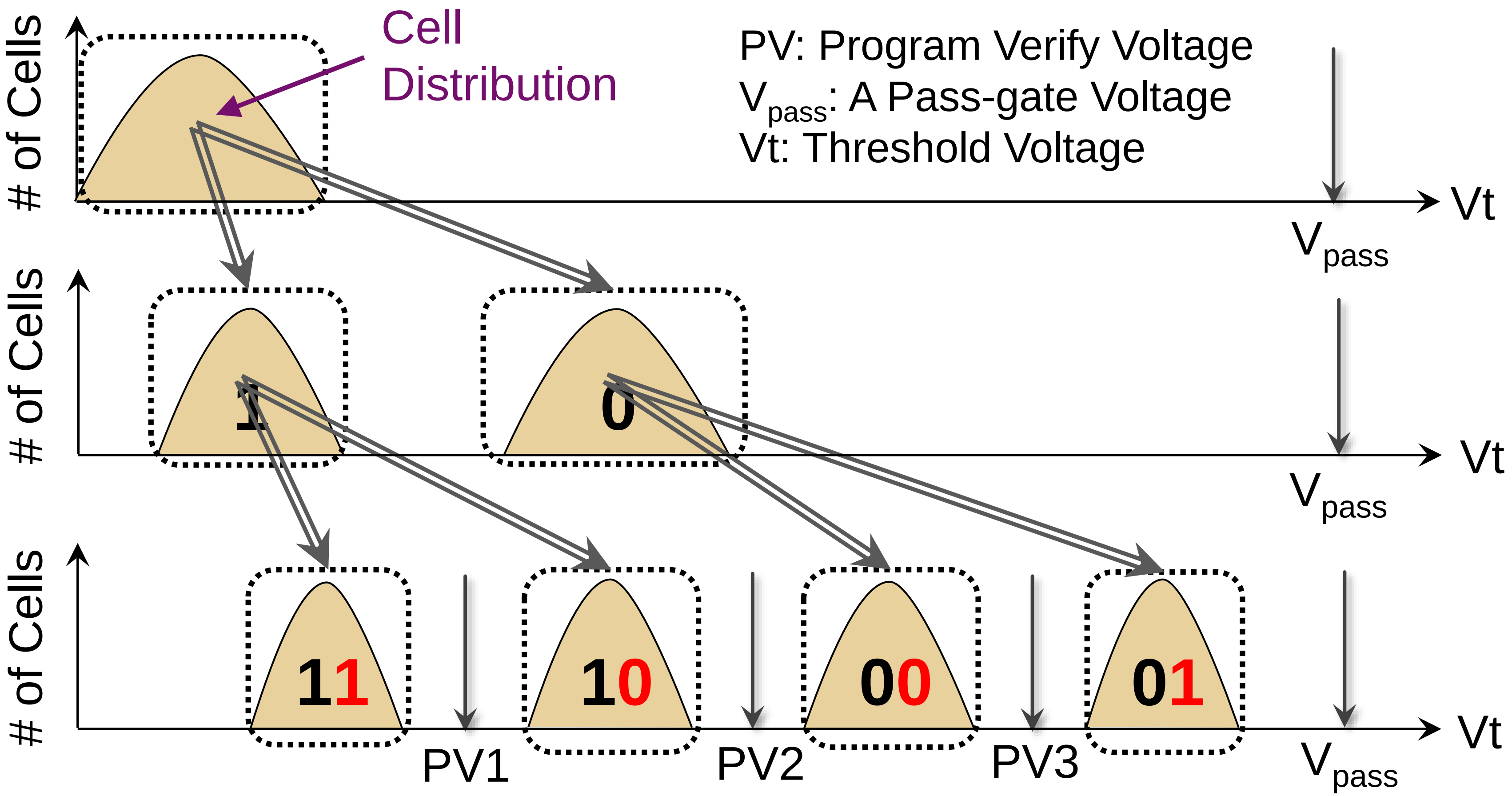
Single-Level Cell & Multi-Level Cell (1/2)

- **Single-Level Cell (SLC)**: *one bit per cell*
  - – SLC provides faster read/write speed, lower error rate and longer endurance with higher cost.
- **Multi-Level Cell (MLC)**: *multiple bits per cell*
  - – MLC allows each memory cell to store multiple bits of information with degraded performance and reliability.
    - Multi-Level Cell (MLC_{x2}): **2** bits per cell
    - Triple-Level Cell (TLC): **3** bits per cell
    - Quad-Level Cell (QLC): **4** bits per cell





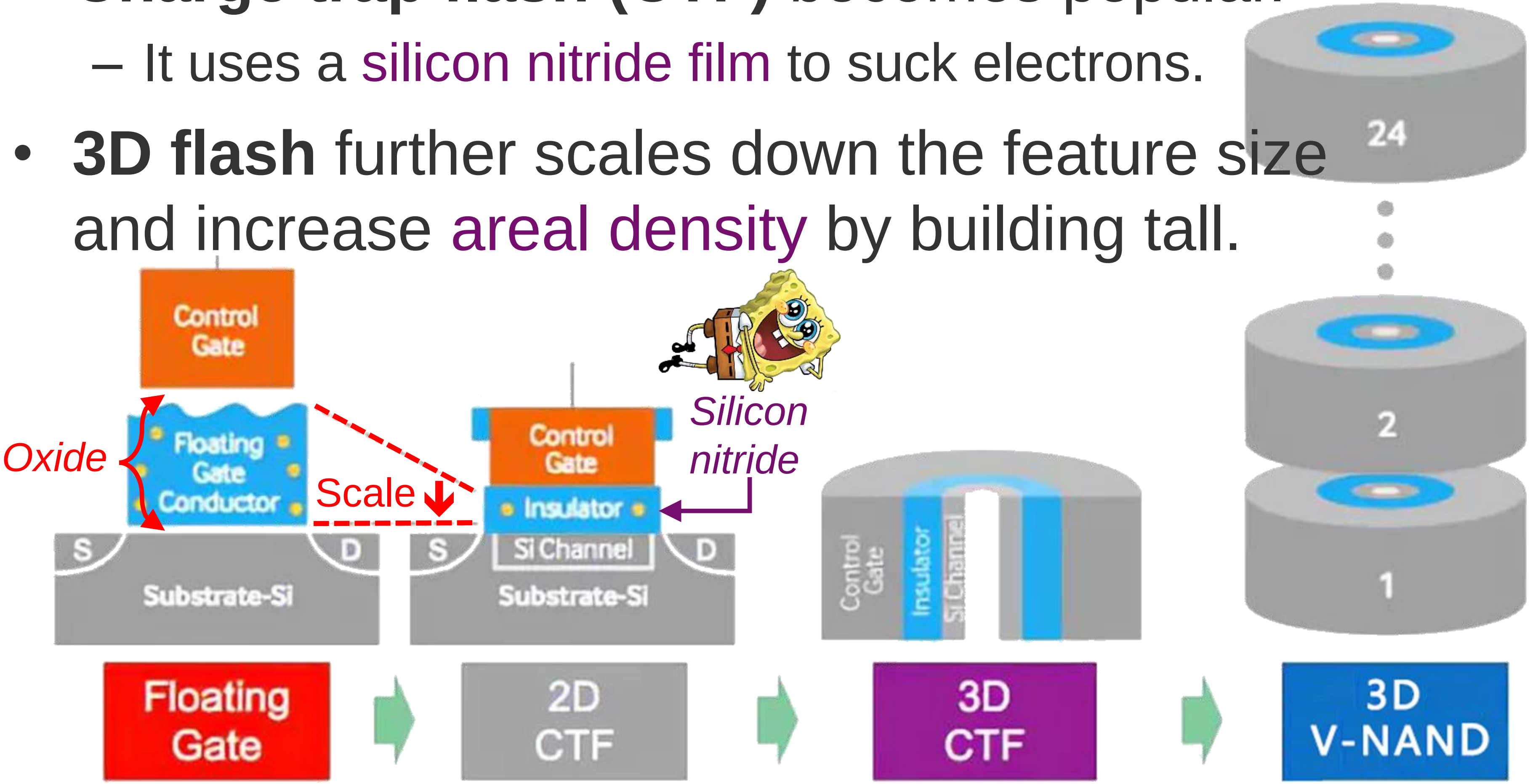
Single-Level Cell & Multi-Level Cell (2/2)



- Low efficiency in programming/verifying low bit(s)
- High bit error rate in low bit(s)

Evolution of NAND Flash

- Scaling down **floating-gate** cell is challenging.
  - The oxide thickness must be more than 6-nm.
- **Charge trap flash (CTF)** becomes popular.
  - It uses a silicon nitride film to suck electrons.
- **3D flash** further scales down the feature size and increase areal density by building tall.

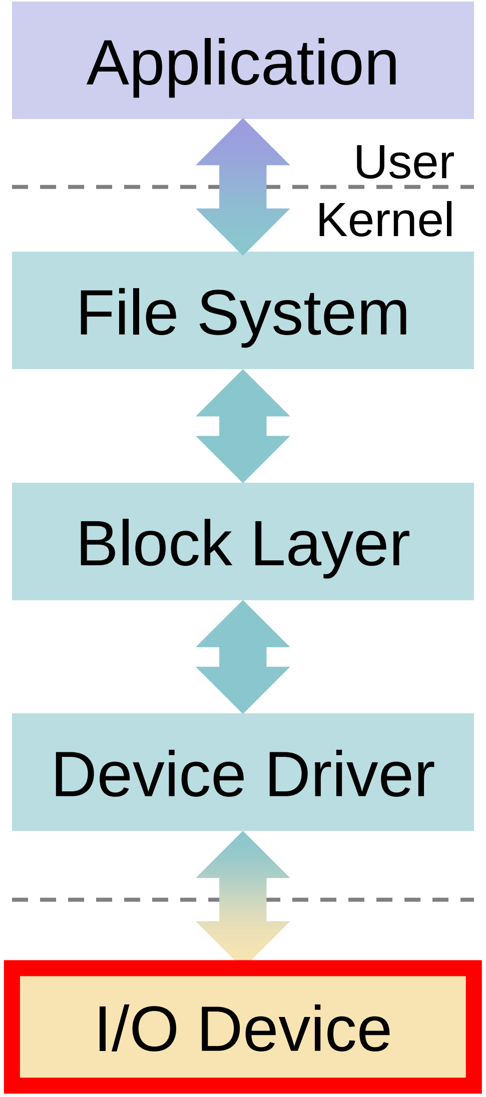


https://www.chinaflashmarket.com/Instructor/163856

Outline

- Flash Memory: Why and How
  - NAND Flash Technology
  - Inherent Challenges
- System Architecture
- Flash Translation Layer
  - Address Mapping
  - Garbage Collection
  - Wear Leveling
  - Multilevel I/O Parallelism
- Flash-aware File System
  - Flash-Friendly File System (F2FS)

I/O Stack



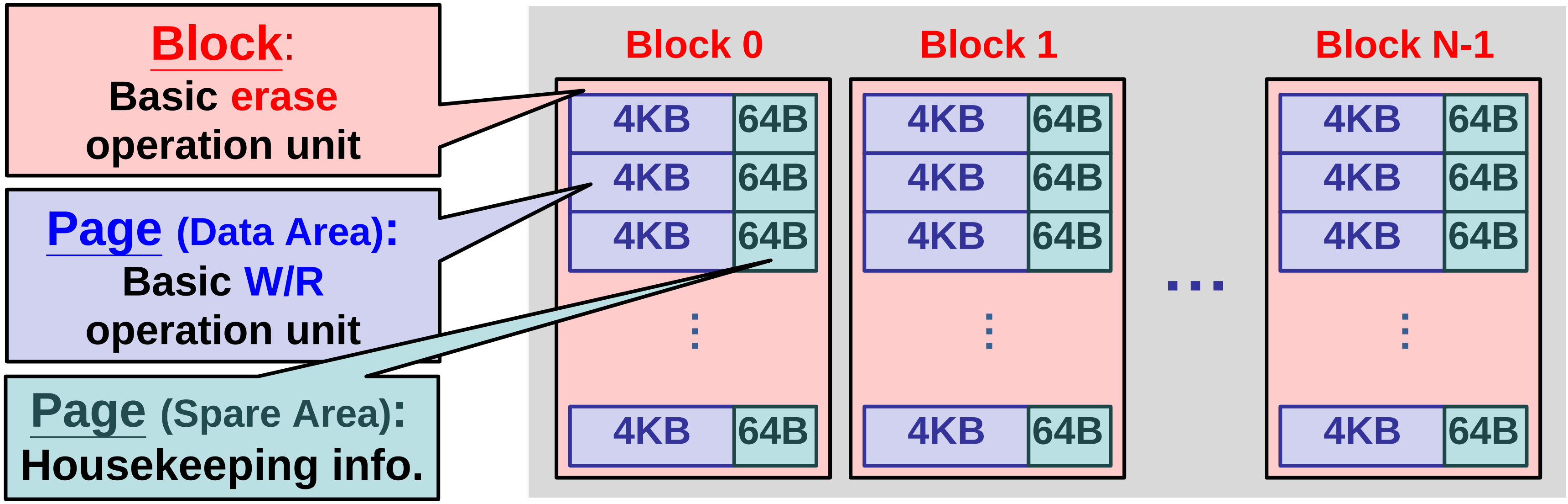
Inherent Challenges

- Common challenges of NAND flash:
  ① Asymmetric operation units between read/write and erase
  ② Erase before writing (a.k.a., write-once property)
  ③ Limited endurance
  ④ Data errors caused by write and read disturb
  ⑤ Data retention errors

- **Sophisticated management techniques** are needed to make flash become better.

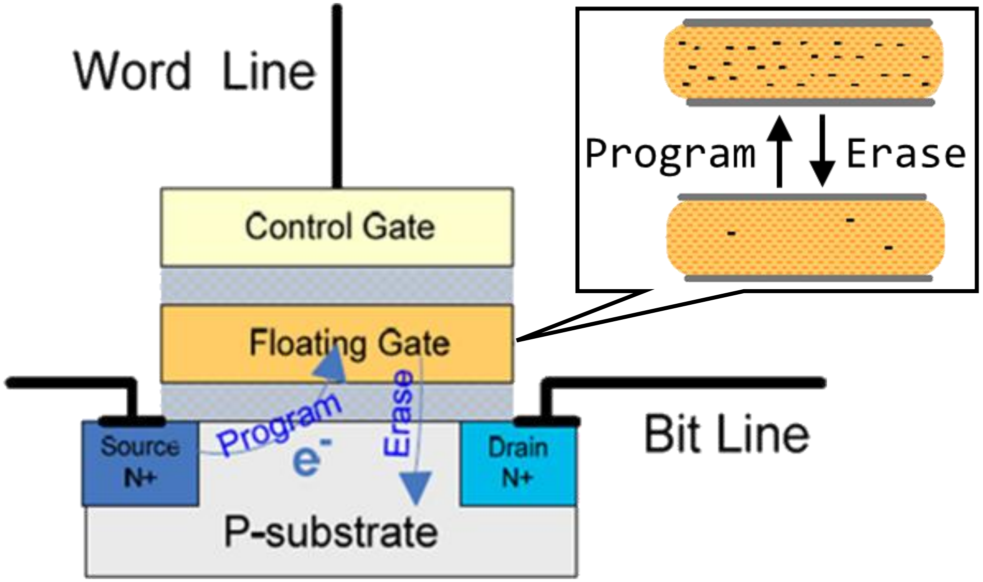
① Asymmetric Operation Units

- Flash cells are organized into pages, and hundreds of pages are grouped into a block.
  - A page is further divided into **data area** and **spare area**.
    - The **spare area** keeps redundancy for error correction or metadata.
- **Asymmetric Operation Units**: Flash cells can only be read or written in the unit of a page; while all pages of a block need to be erased at a time.



② Erase before Write (Write-once)

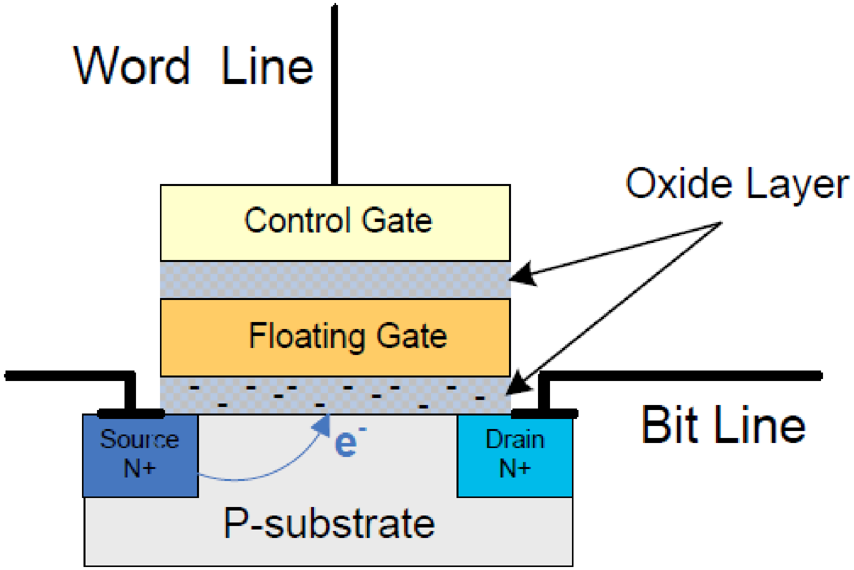
- **Erase before Write**: Once written to “`0`”, the only way to reset a flash cell to “`1`” is by erasing.
  - The **`erase`** operation sets all bits in a **block** to “`1`”s at a time.



- **Write-once Property**: A flash **page** cannot be overwritten until the residing **block** is erased first.

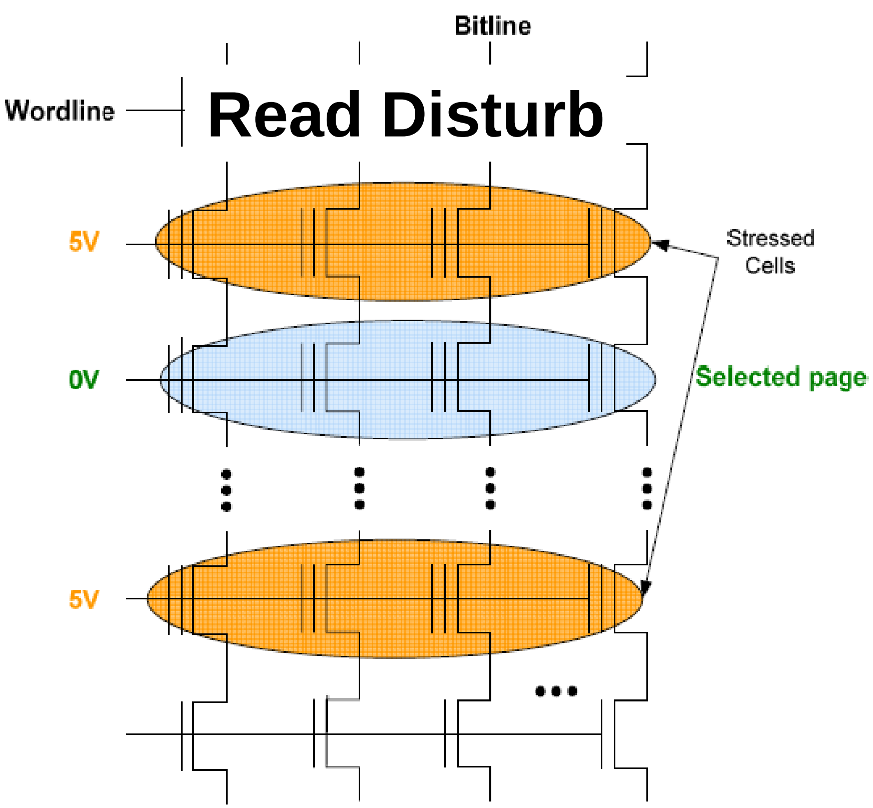
③ Limited Endurance

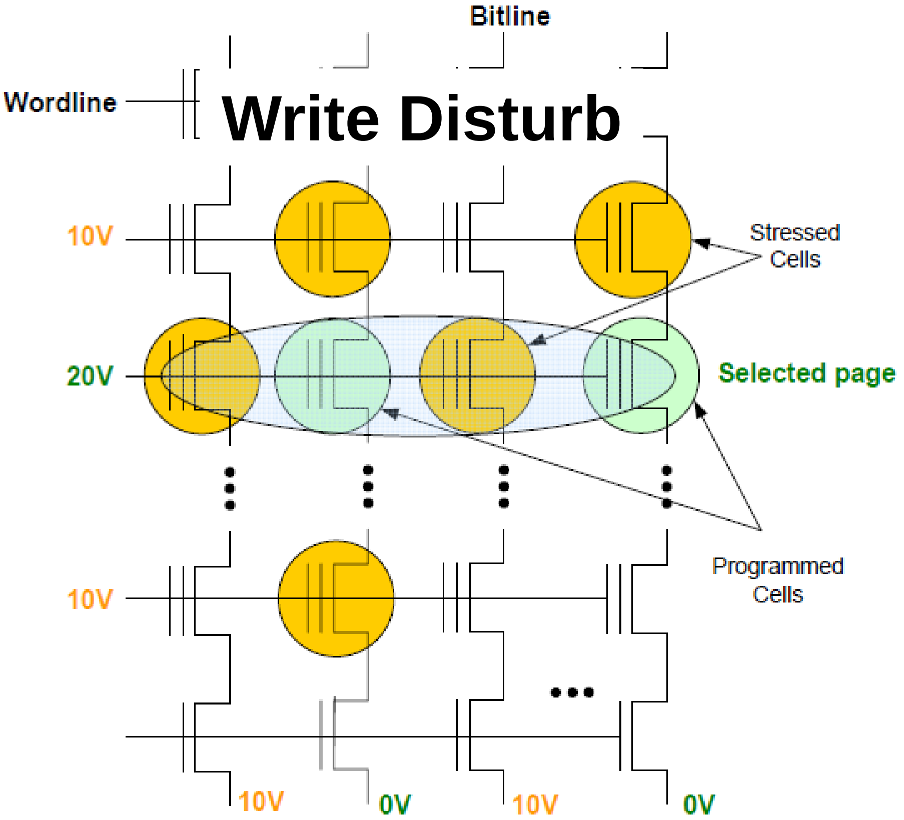
- **Limited Endurance**: A flash block can only endure a limited number of program/erase (P/E) cycles.
  - SLC: $60K \sim 100K$ P/E cycles
  - MLC: $1K \sim 10K$ P/E cycles
  - TLC: $<\ 1K$ P/E cycles
- Reason: The oxide layer may be “worn out”.



④ Read/Write Disturb

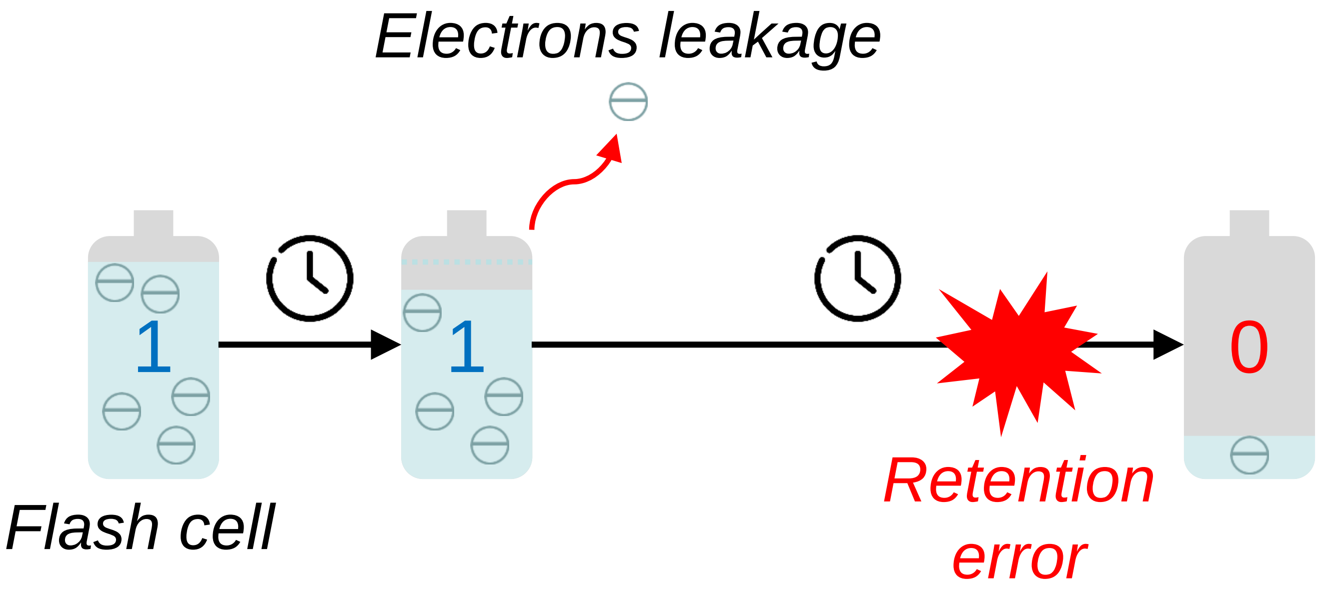
- **Read/Write Disturb**: Reading or writing a page may result in the “weak programming” on its neighbors.
  - **Solution to Write Disturb**: Programming pages of a block “in a sequential order” (a.k.a., **sequential write constraint**).





⑤ Data Retention Errors

- **Data retention time** defines how long the written data remains valid within a flash cell.
  - – It is inversely related to the number of P/E cycles.
- Electrons leak over time and result in retention errors.
  - – **Solution**: "**Correct-and-refresh pages**" from time to time.
    - This solution can also reset the **read/write disturb**.

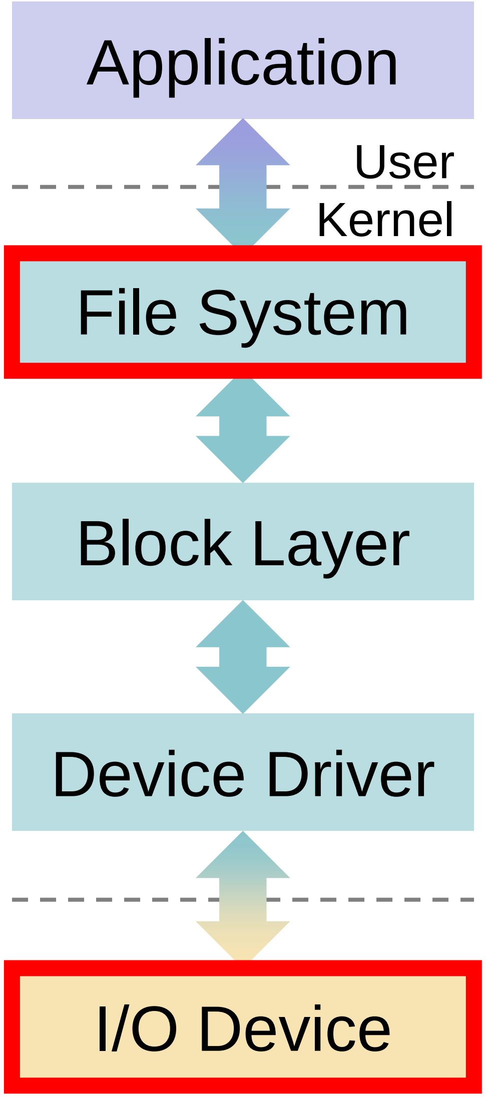


Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime (ICCD'12)

Outline

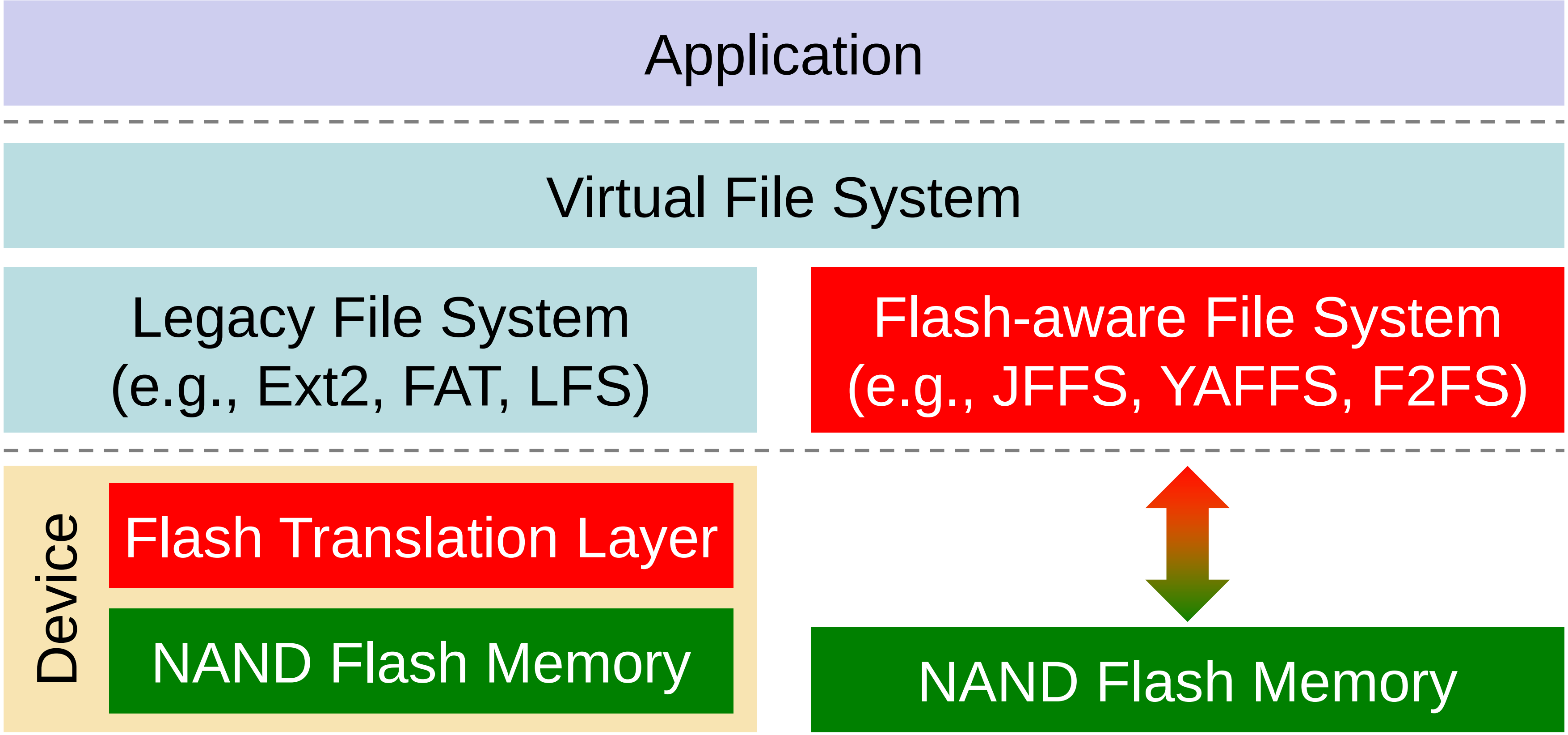
- Flash Memory: Why and How
  - NAND Flash Technology
  - Inherent Challenges
- System Architecture
- Flash Translation Layer
  - Address Mapping
  - Garbage Collection
  - Wear Leveling
  - Multilevel I/O Parallelism
- Flash-aware File System
  - Flash-Friendly File System (F2FS)

I/O Stack



System Architecture

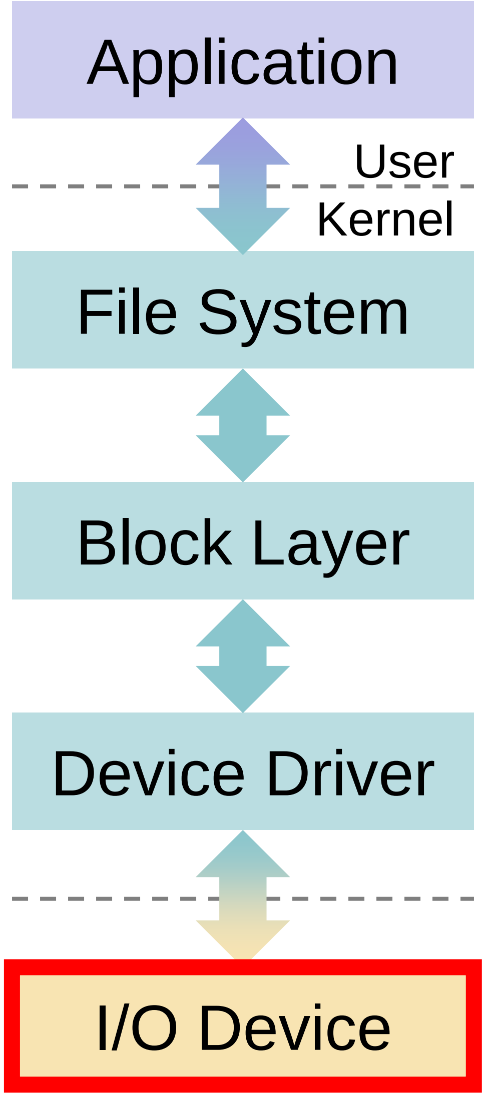
- There are two typical ways to address the inherent challenges of flash memory:
  ① Implementing a **Flash Translation Layer** in the **device**.
  ② Designing a **Flash-aware File System** in the **host**.



Outline

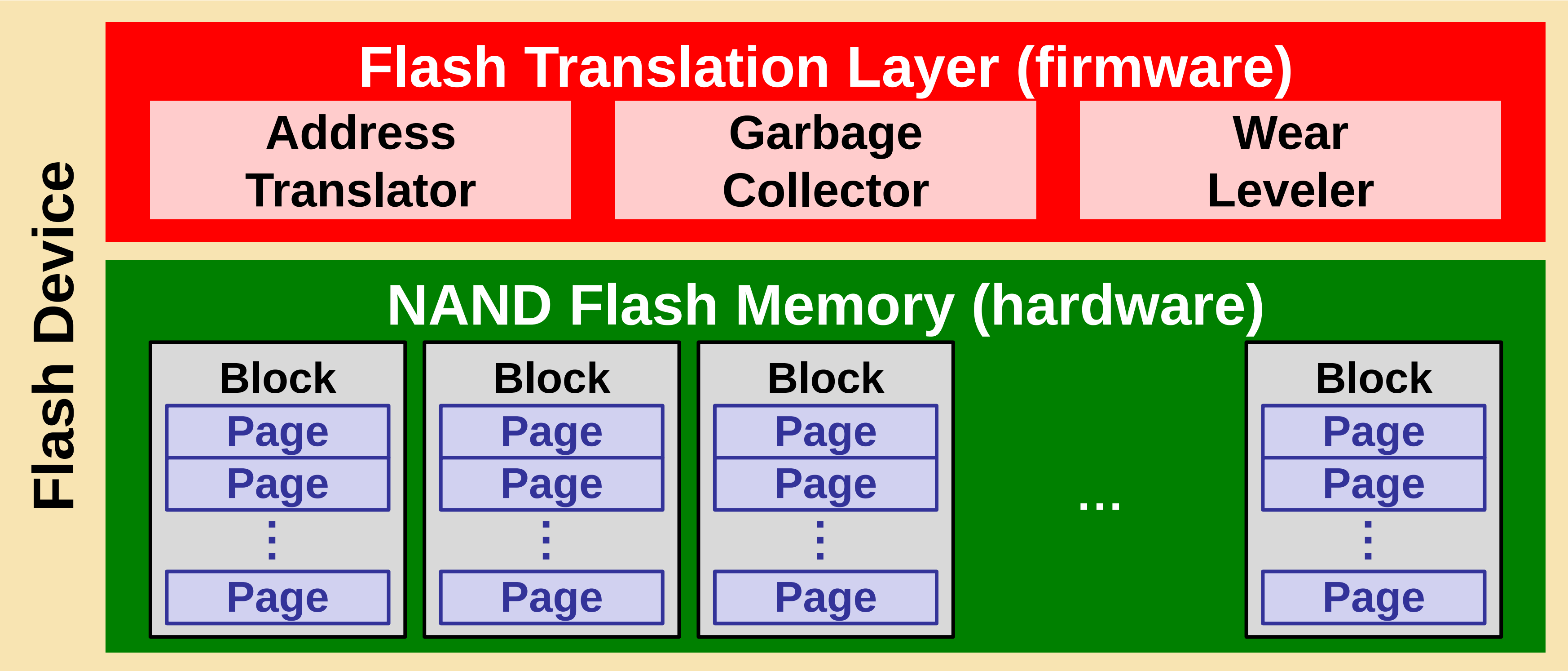
- Flash Memory: Why and How
  - NAND Flash Technology
  - Inherent Challenges
- System Architecture
- Flash Translation Layer
  - Address Mapping
  - Garbage Collection
  - Wear Leveling
  - Multilevel I/O Parallelism
- Flash-aware File System
  - Flash-Friendly File System (F2FS)

I/O Stack



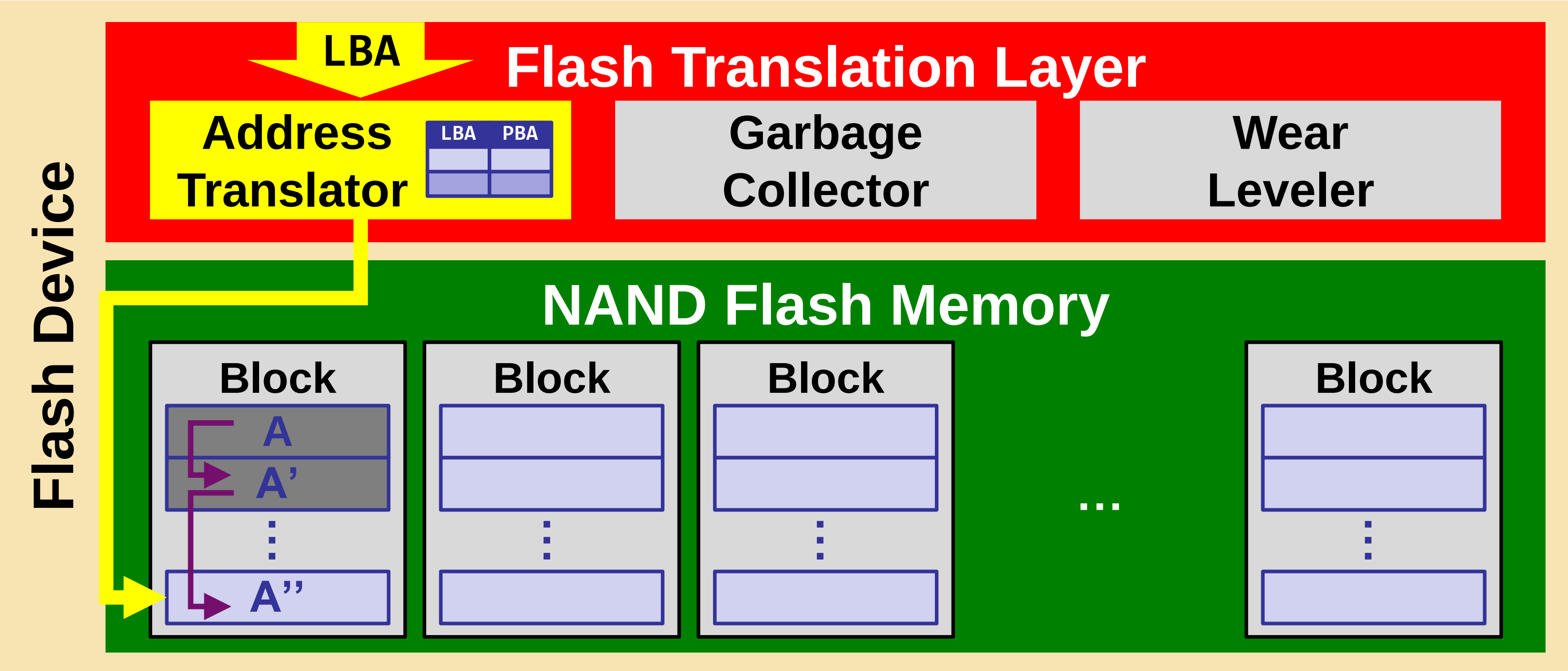
Flash Translation Layer

- **Flash Translation Layer (FTL)** is a firmware inside the flash device to make the NAND flash memory "appear as" a block device to the host.
  - It consists of three major components: ① address translator, ② garbage collector, and ③ wear leveler.



① Address Translator

- Due to the write-once property, the **out-place update** is adopted to write the updated data to free pages.
  - **Address translator maps** *logical block addresses (**LBAs**)* from the host to *physical page addresses (**PPAs**)* in flash.
    - The mapping table is kept in the **memory space** of the flash device.



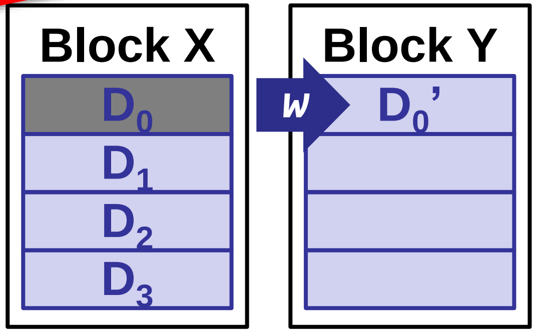
Page-Level, Block-Level, and Hybrid

- ## Page-Level (PL)

1-1 Page-Level Mapping Table

Big in table size!

LBA	PPA
0	(X, 0)→(Y, 0)
1	(X, 1)
2	(X, 2)
…	…

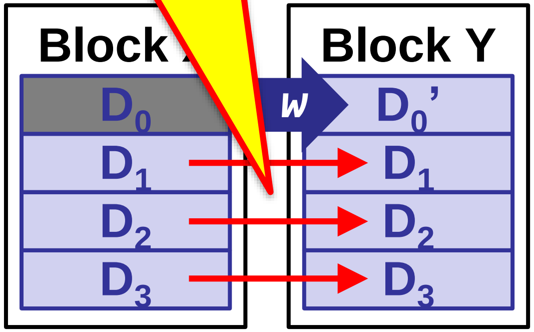


- ## Block-level (BL)

1-1 Block-Level Mapping Table

Slow in write!

LBA	PBA
0~3	X → Y
4~7	…
8~11	…
…	…



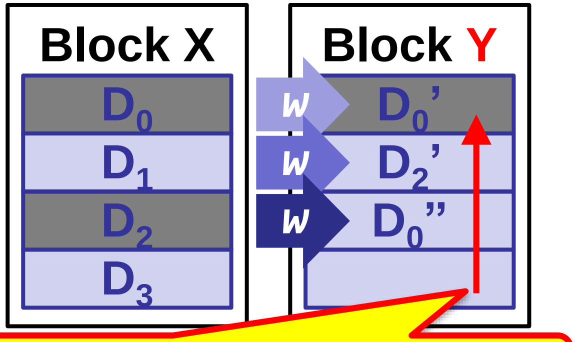
- ## Block-Level (BAST)

1-2 Block-Level Mapping Table

(Like BL) **Primary**

(Logging) **Secondary**

LBA	PBA
0~3	(X, Y)
4~7	…
8~11	…
…	…



Slow in read (must check spare area)!

- ## Hybrid (FAST)

Good R/W?

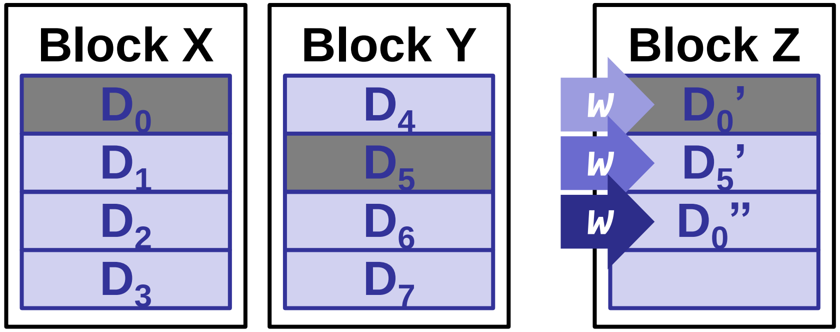
1-1 BL Map

LBA	PBA
0~3	X

+

PL Map

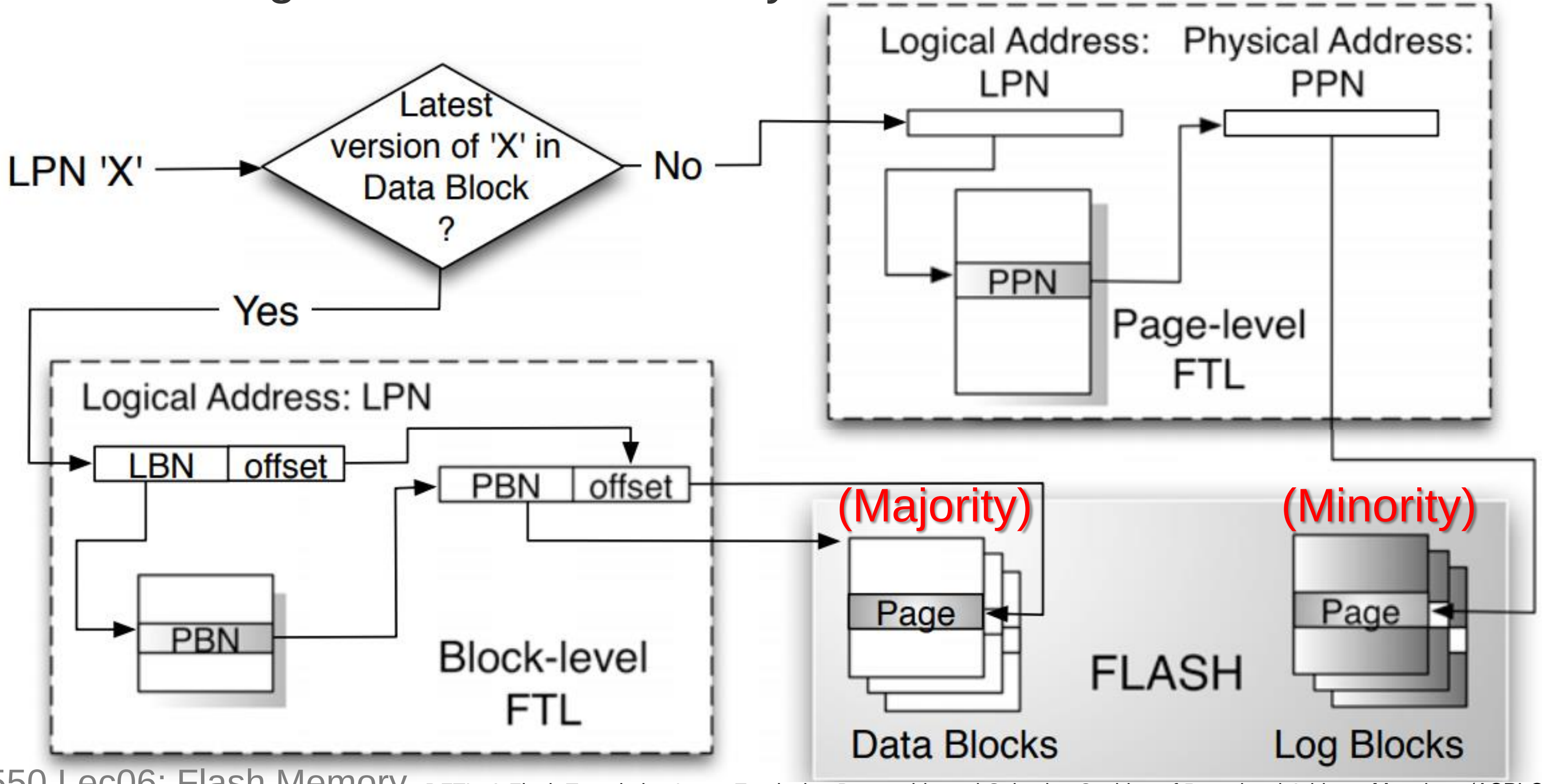
LBA	PPA
0	(Z, 2)



Data Block(s) *share* **Log Block(s)**

Hybrid FTL

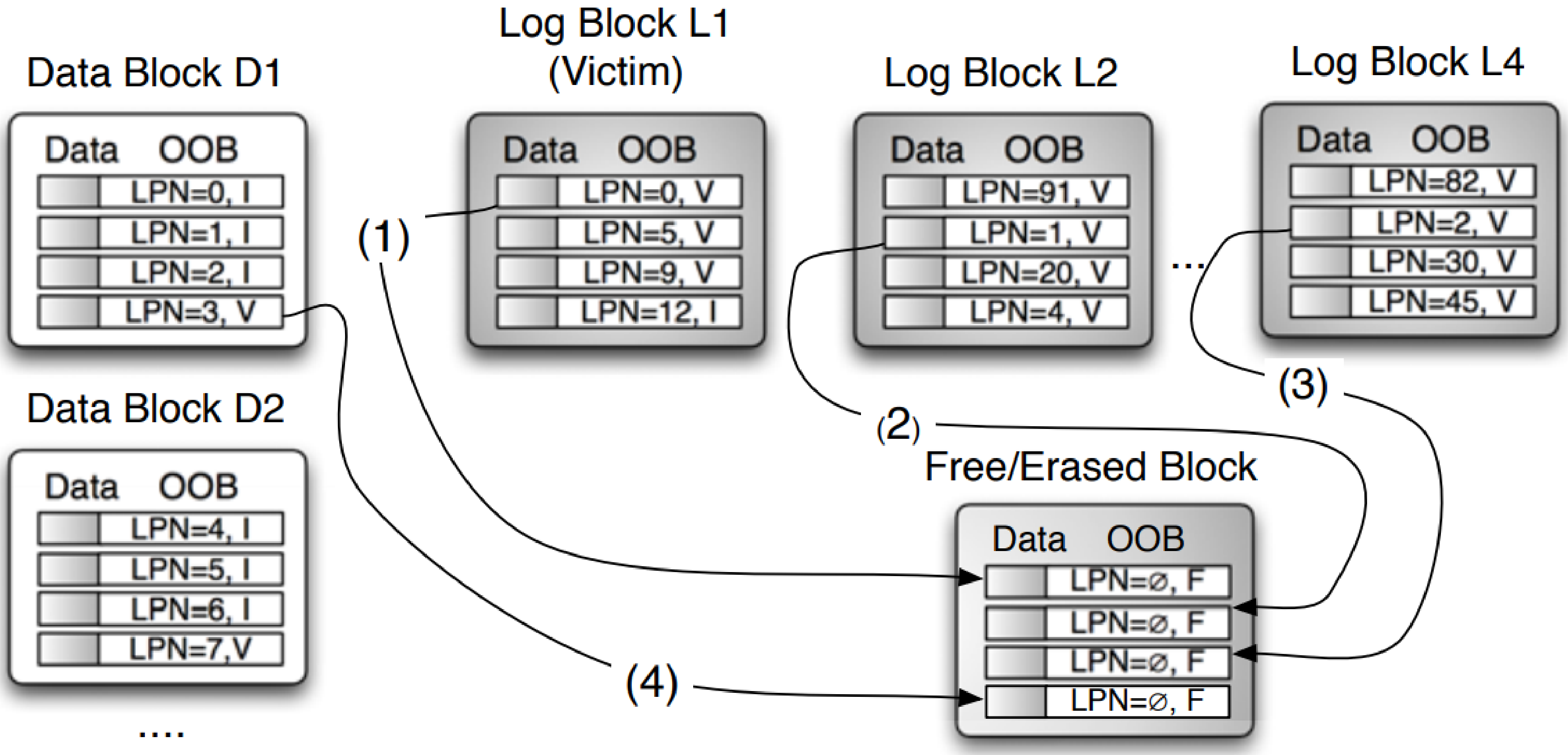
- Hybrid FTLs logically partition blocks into two groups:
  - **Data Blocks** are mapped via the block-level mapping.
  - **Log/Update Blocks** are mapped via a page-level mapping.
    - Any update on data blocks are performed by writes to the log blocks.
    - Few log blocks are shared by all data blocks.



DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings (ASPLOS'09)

Expensive Merge of Hybrid FTL

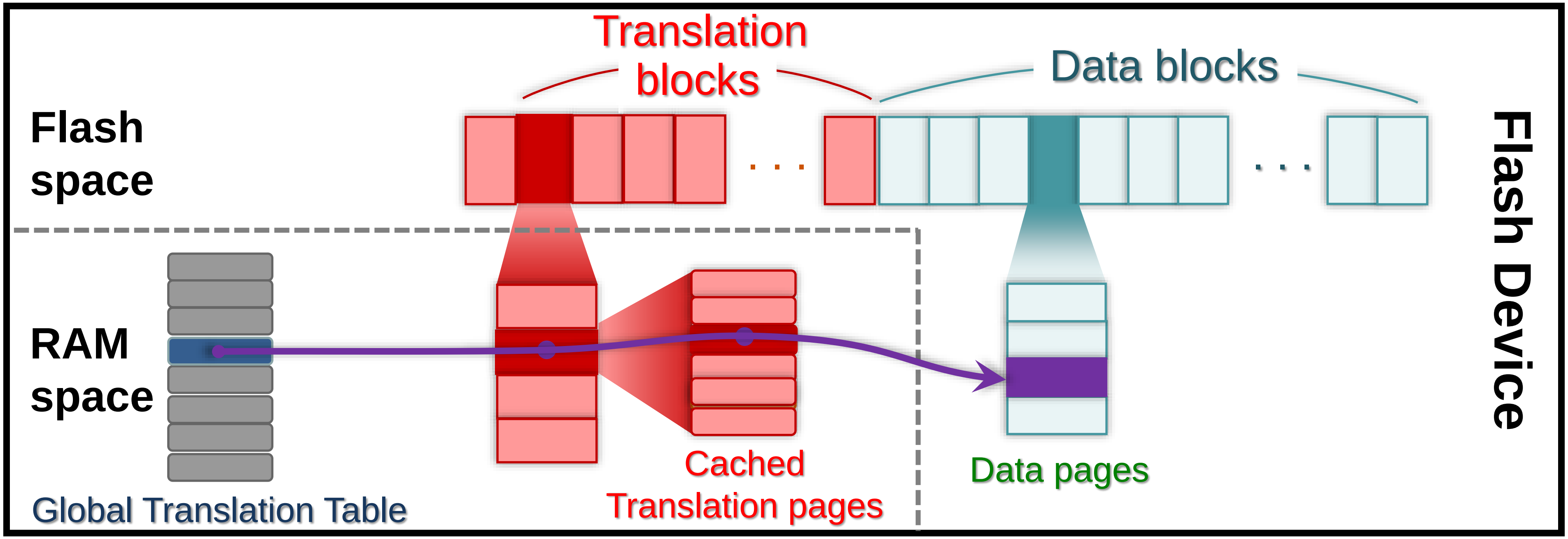
- Hybrid FTLs induce costly garbage collection.



DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings (ASPLOS'09)

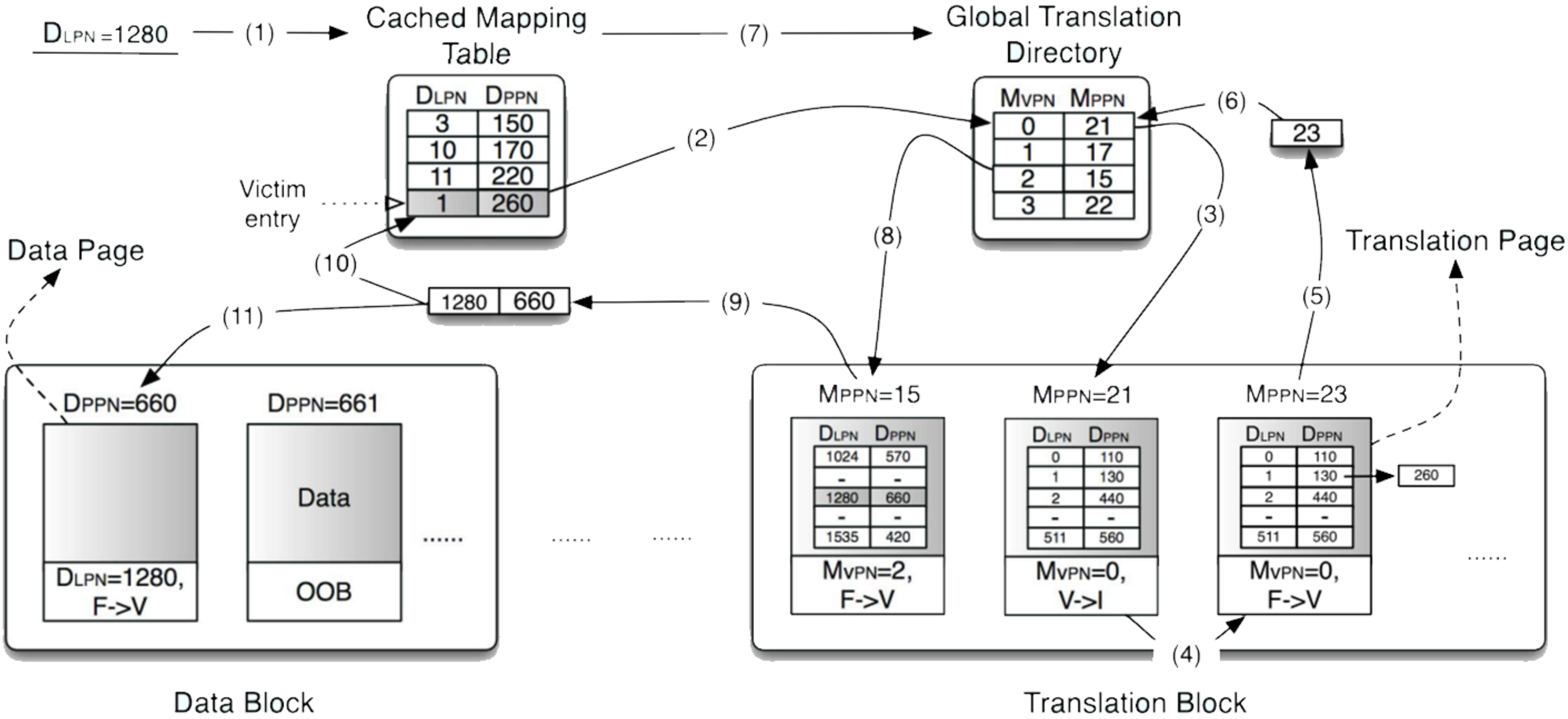
Demand-based Address Translation

- Keeping all mapping tables in RAM is ineffective.
- **Page-level** translation for all data with **limited RAM**.
  - **Map** of data pages are stored in translation pages on **flash**.
  - Translation pages are cached in **RAM** on demand.
  - **Map** of translation pages (i.e., global translation table) are kept in **RAM** for efficient lookup.



DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings (ASPLOS'09)

DFTL: An Working Example



Cached Mapping Table

DLPN	DPPN
3	150
10	170
11	220
1	260

Global Translation Directory

MVPN	MPPN
0	21
1	17
2	15
3	22

MPPN=15

DLPN	DPPN
1024	570
-	-
1280	660
-	-
1535	420

MPPN=21

DLPN	DPPN
0	110
1	130
2	440
-	-
511	560

MPPN=23

DLPN	DPPN
0	110
1	130
2	440
-	-
511	560

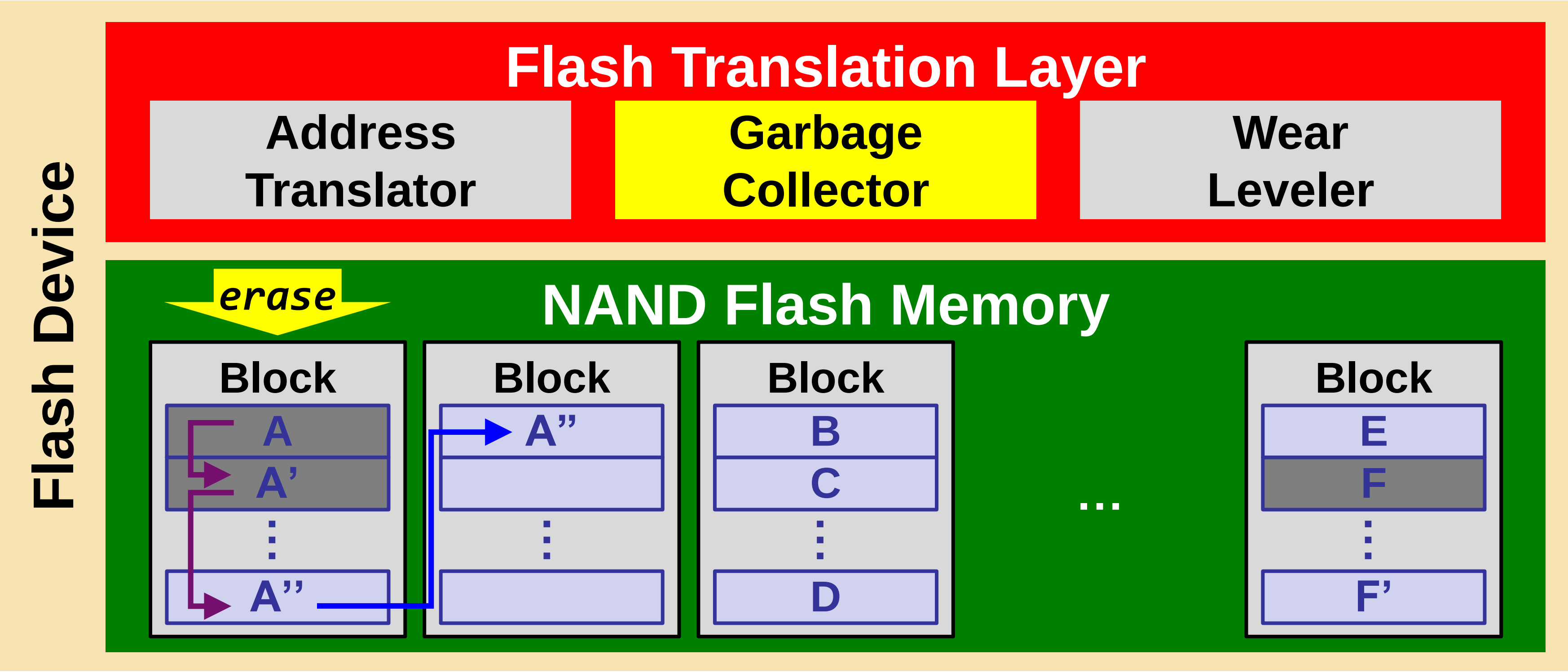
(1) Request to DLP N 1280 incurs a miss in Cached Mapping Table (CMT), **(2)** Victim entry DLP N 1 is selected, its corresponding translation page MPPN 21 is located using Global Translation Directory (GTD), **(3)-(4)** MPPN 21 is read, updated (DPPN 130 → DPPN 260) and written to a free translation page (MPPN 23), **(5)-(6)** GTD is updated (MPPN 21 → MPPN 23) and DLP N 1 entry is erased from CMT. **(7)-(11)** The original request's (DLP N 1280) translation page is located on flash (MPPN 15). The mapping entry is loaded into CMT and the request is serviced.

Note that each GTD entry maps 512 logically consecutive mappings.

DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings (ASPLOS'09)

② Garbage Collector

- Since the out-place update leaves multiple versions of data, the **garbage collector** is to reclaim pages occupied by stale data by **erasing** its residing blocks.
  - The **live-page copying** is needed to migrate pages of the latest versions of data before the erase operation.

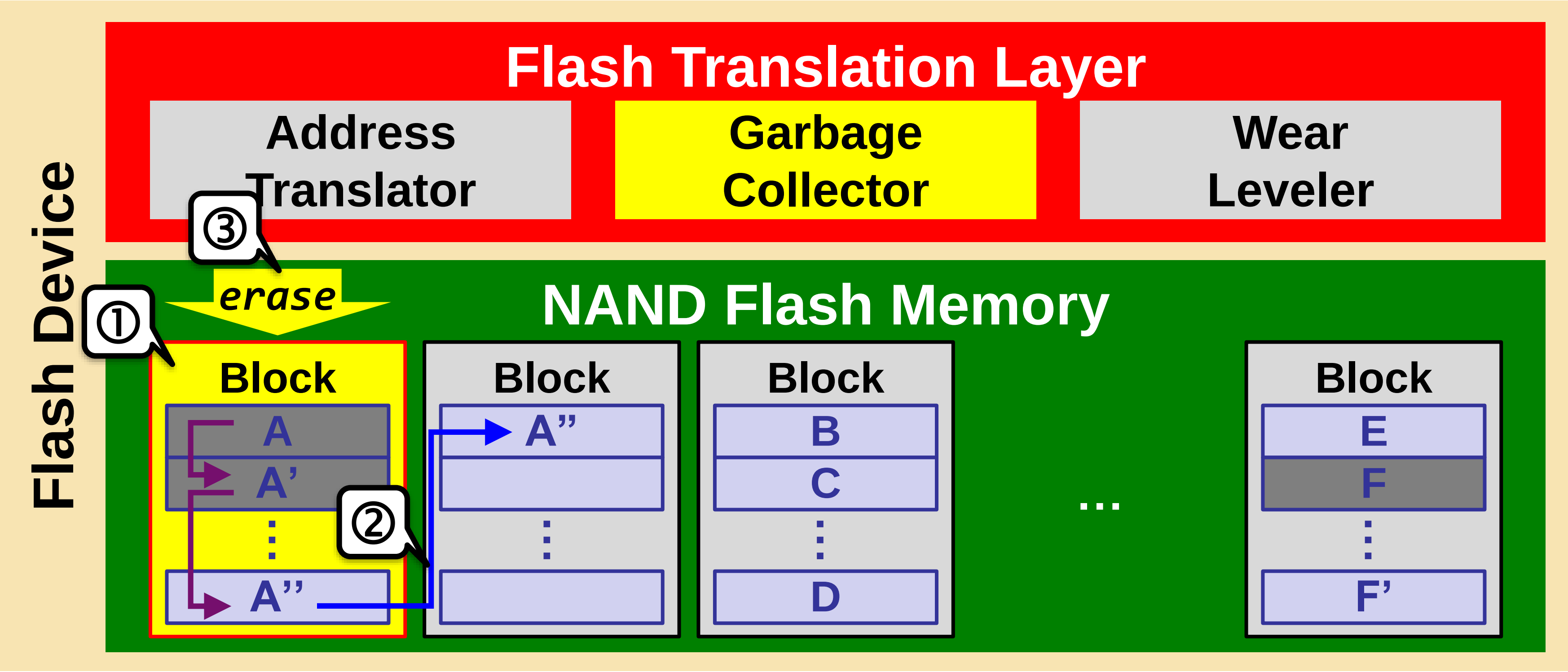


Typical GC Procedure

① **Victim Block Selection**: Pick one (or more) block that is “most worthy” to be reclaimed

② **Live-Page Copying**: Migrate all live pages out

③ **Victim Block Erasing**: Reclaim the space occupied by dead pages



Typical Victim Block Selection Policies

- Random selects the victim block in a uniformly-random manner to yield a long-term average use.
- FIFO (or RR) cleans blocks in a round-robin manner for minimized P/E cycle difference.
- Greedy cleans the block with the largest number of dead pages for minimized live-page copying.
- Cost-Benefit (CB) (and its variants) cleans the block with the largest benefit-cost for wear leveling:

$$\frac{benefit}{cost} = \frac{age \times (1 - u)}{2u}$$

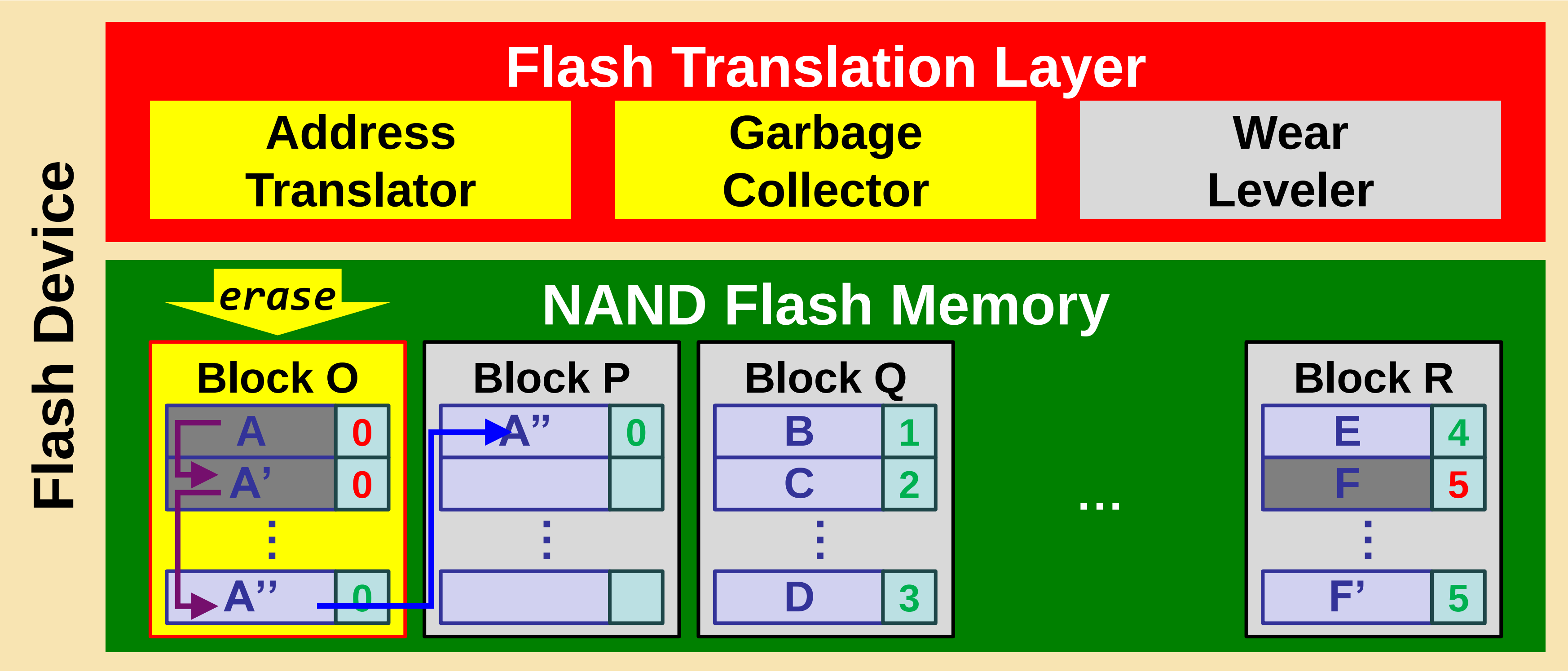
  - ***age***: invalidated time period = current time – the time when a page of the block was lastly invalidated
  - ***u***: percentage of live pages of the block

Garbage Collection and Wear Leveling for Flash Memory: Past and Future (SmartComp'14)

Page Liveness

- Keep the **LBA** into the spare area.
- Check both the spare area of pages and the mapping table during GC.
  - Live: LBA **matches** the mapping.
  - Dead: LBA **mismatches** the mapping.

LBA	PPA
0	(O, 3)
1	(Q, 0)
2	(Q, 1)
3	(Q, 3)
…	…



Flash Device

Flash Translation Layer

Address Translator

Garbage Collector

Wear Leveler

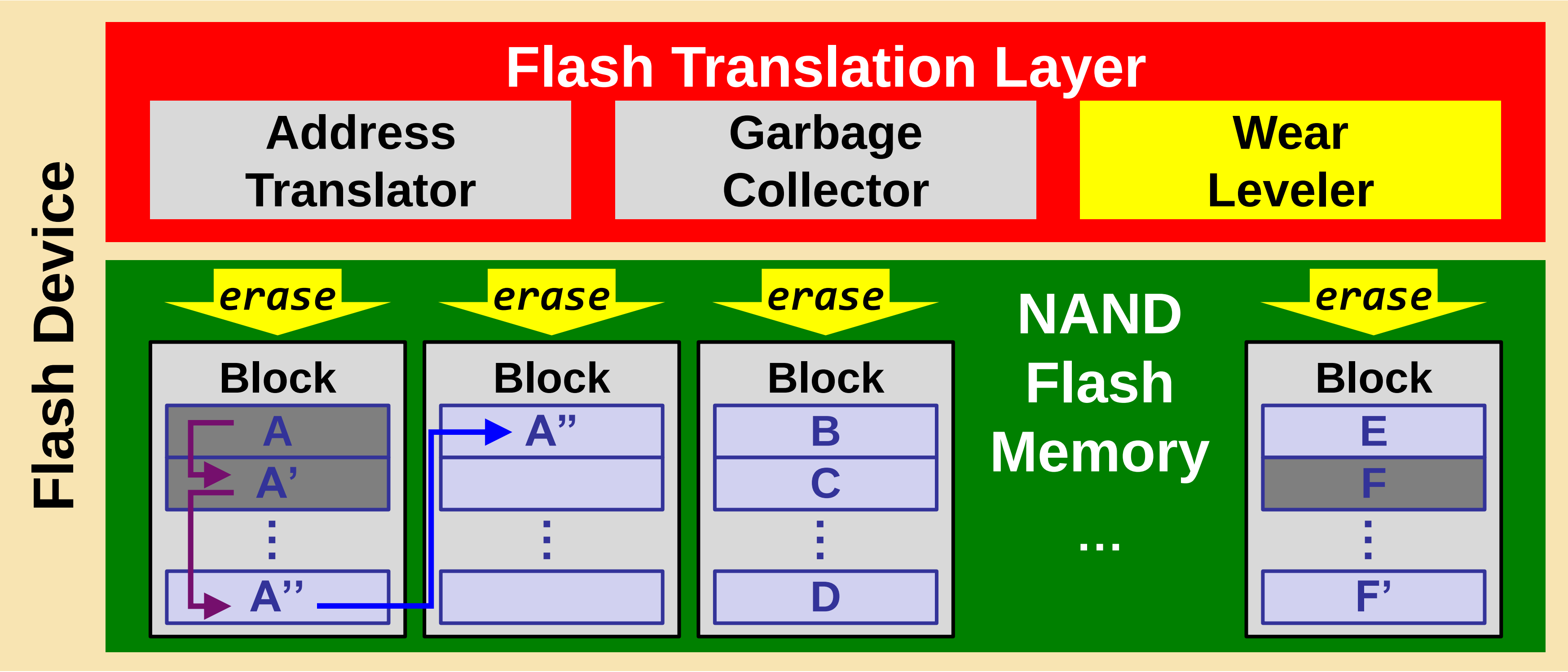
NAND Flash Memory

erase

Block O		Block P		Block Q		…	Block R	
A	0	A''	0	B	1		E	4
A'	0			C	2		F	5
⋮		⋮		⋮			⋮	
A''	0			D	3		F'	5

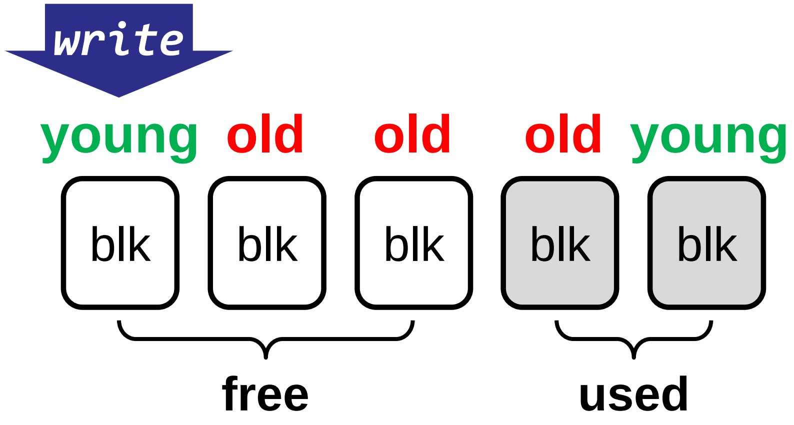
③ Wear Leveler

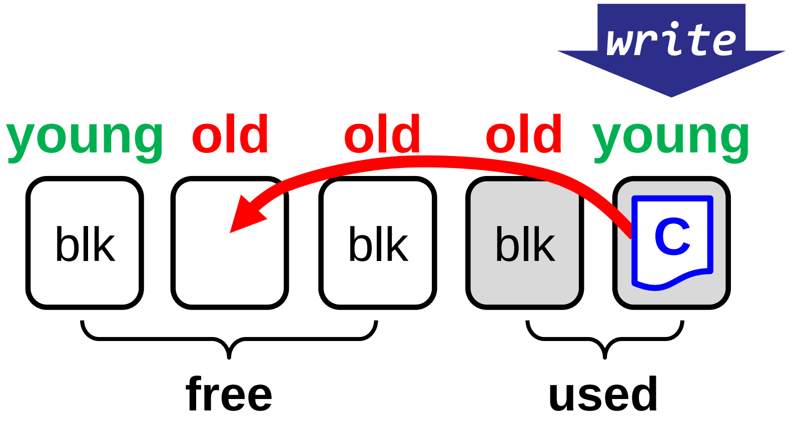
- Since each block can only endure a limited number of P/E cycles, the **wear leveler** is to prolong the overall lifetime by evenly distributing erases.
  - The main objective is to prevent any flash block from being worn out "prematurely" than others.



Dynamic vs. Static Wear Leveling

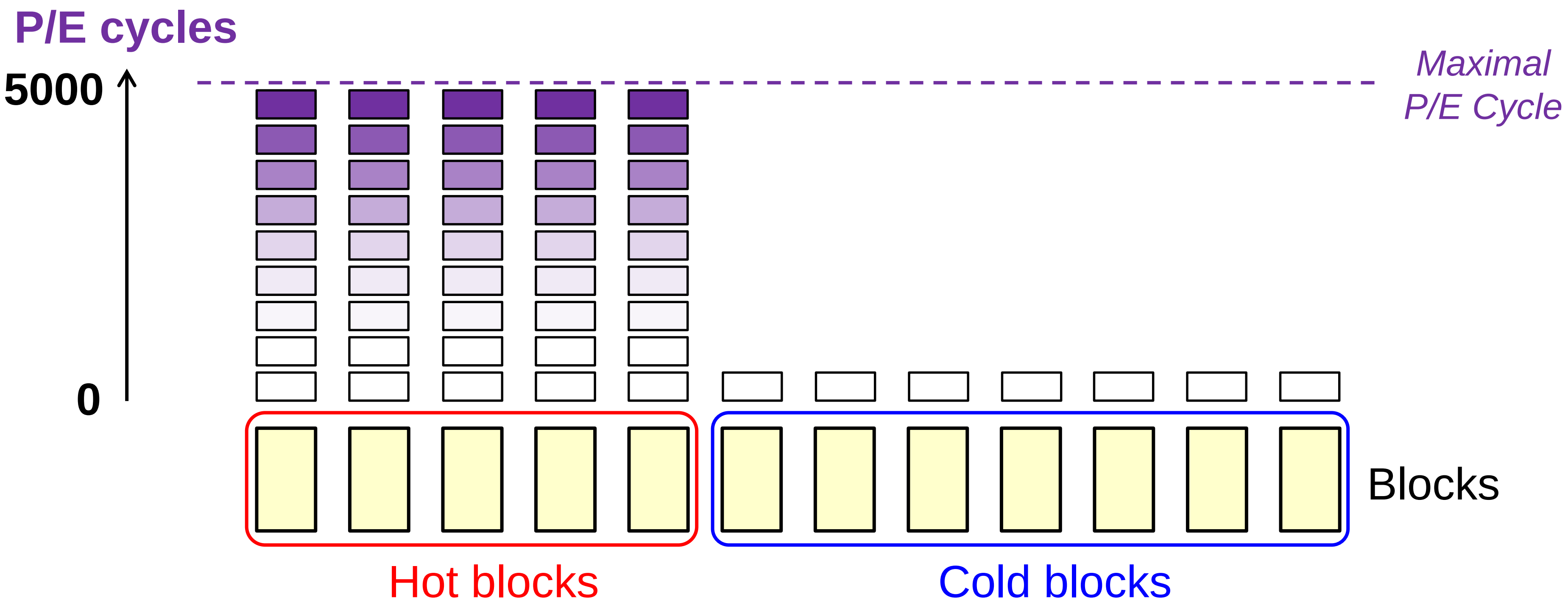
- Wear leveling is classified into **static** or **dynamic** based on the type of blocks involved in WL:
  - **Dynamic Wear Leveling**
    - Use free block with lower erase count (i.e., young block) to service writes.
  - **Static Wear Leveling**
    - Let blocks have **even erase counts** by actively moving cold data to elder blocks.
    - Then use young block to service writes as DWL.





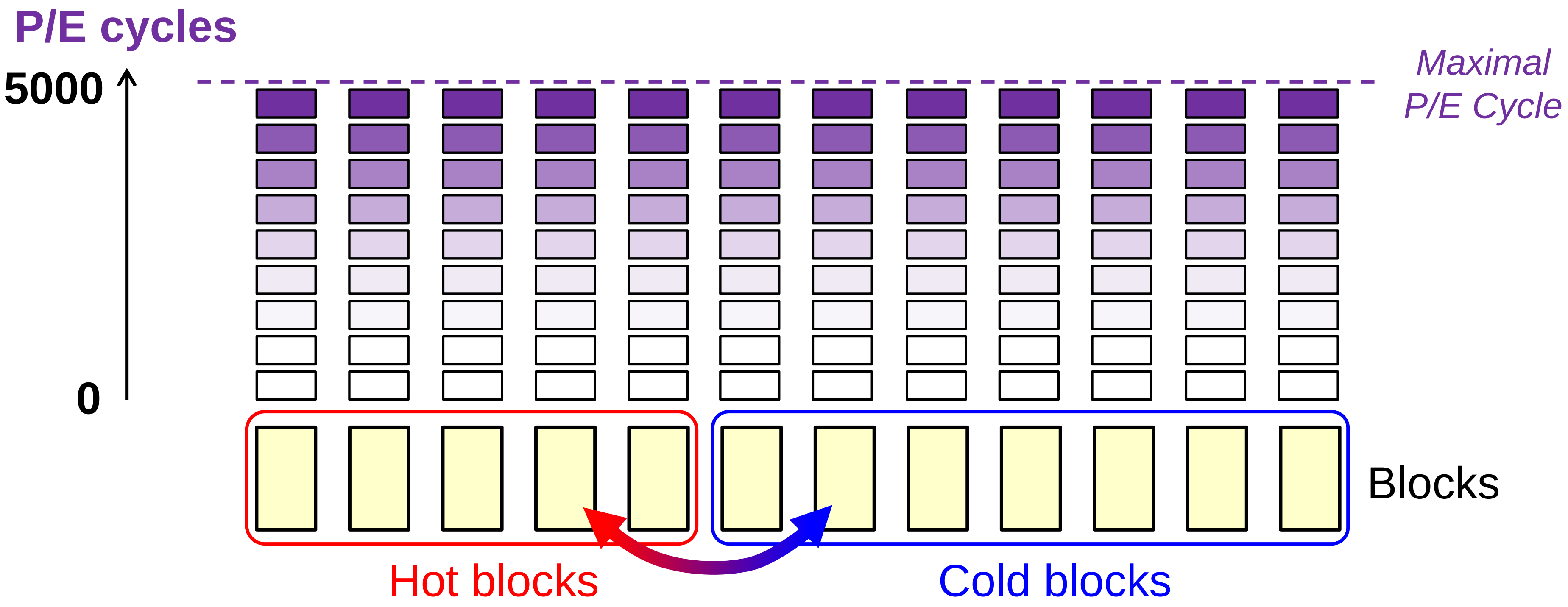
Dynamic Wear Leveling (DWL)

- DWL achieves wear leveling **only for hot blocks**.
  - **Hot Block**: a block mainly containing hot data
  - **Cold Block**: a block mainly containing cold data
- Hot blocks will be **worn out earlier** than cold blocks.



Static Wear Leveling (SWL)

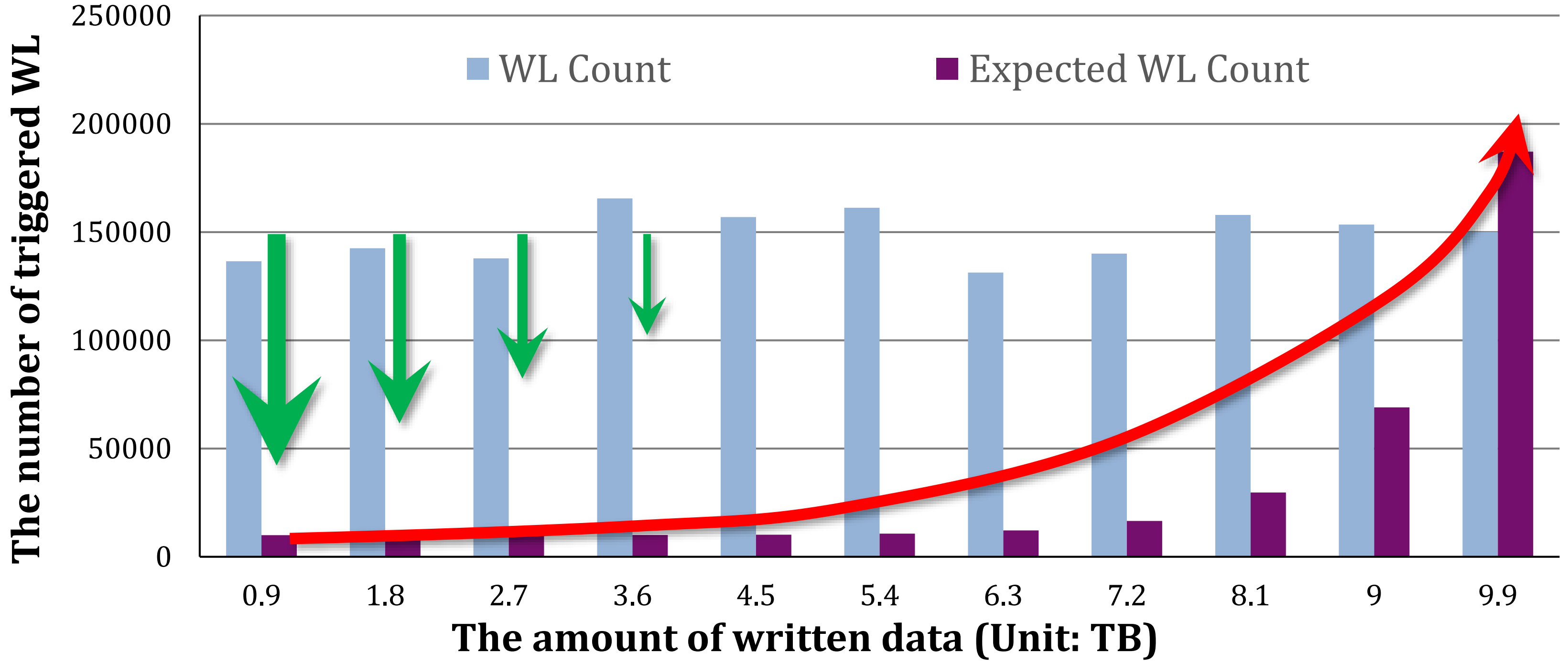
- SWL achieves wear leveling **for all blocks** by **pro-actively moving** cold data to young blocks.
  - – Extra data migrations are introduced.
  - – Performance is traded for lifetime/endurance.



Improving Flash Wear-Leveling by Proactively Moving Static Data (TC'10)
Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead (MSST'11)

Progressive Wear Leveling (PWL)

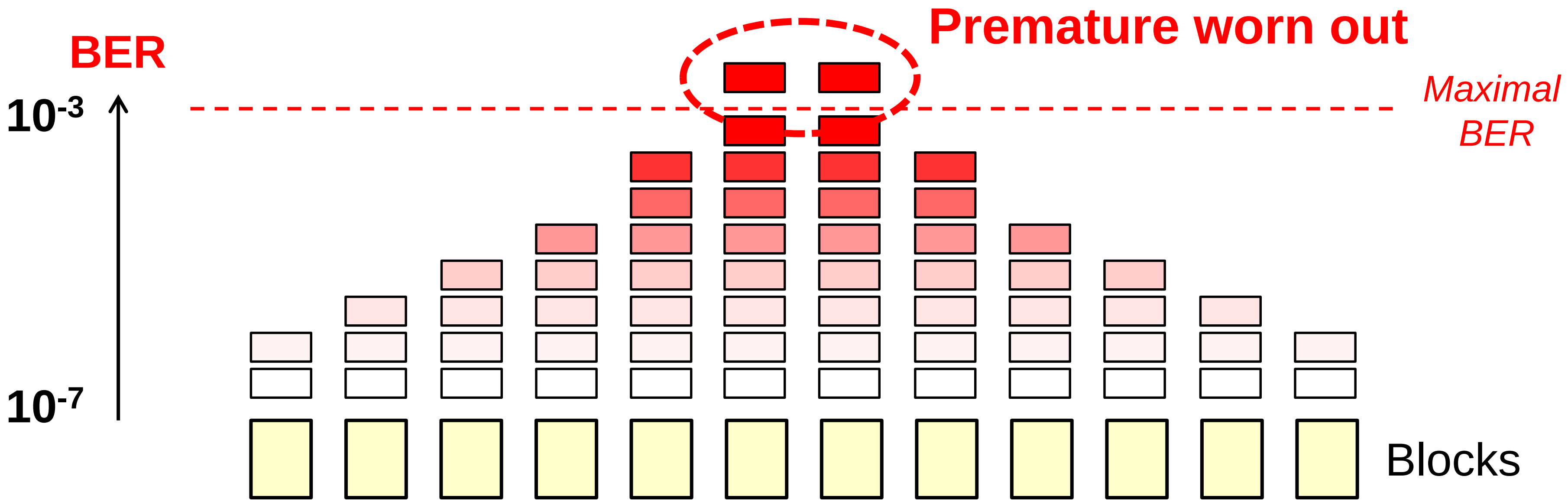
- When, and how often WL should be performed?
- WL should be performed in a **progressive way**:
  - – Prevent WL in the early stages for better performance.
  - – Progressively trigger WL to prolong lifetime.
    - More and more WLs are performed **over time**.



Reducing Data Migration Overheads of Flash Wear Leveling in a Progressive Way (TCAD'16)

Error-Rate-Aware Wear Leveling

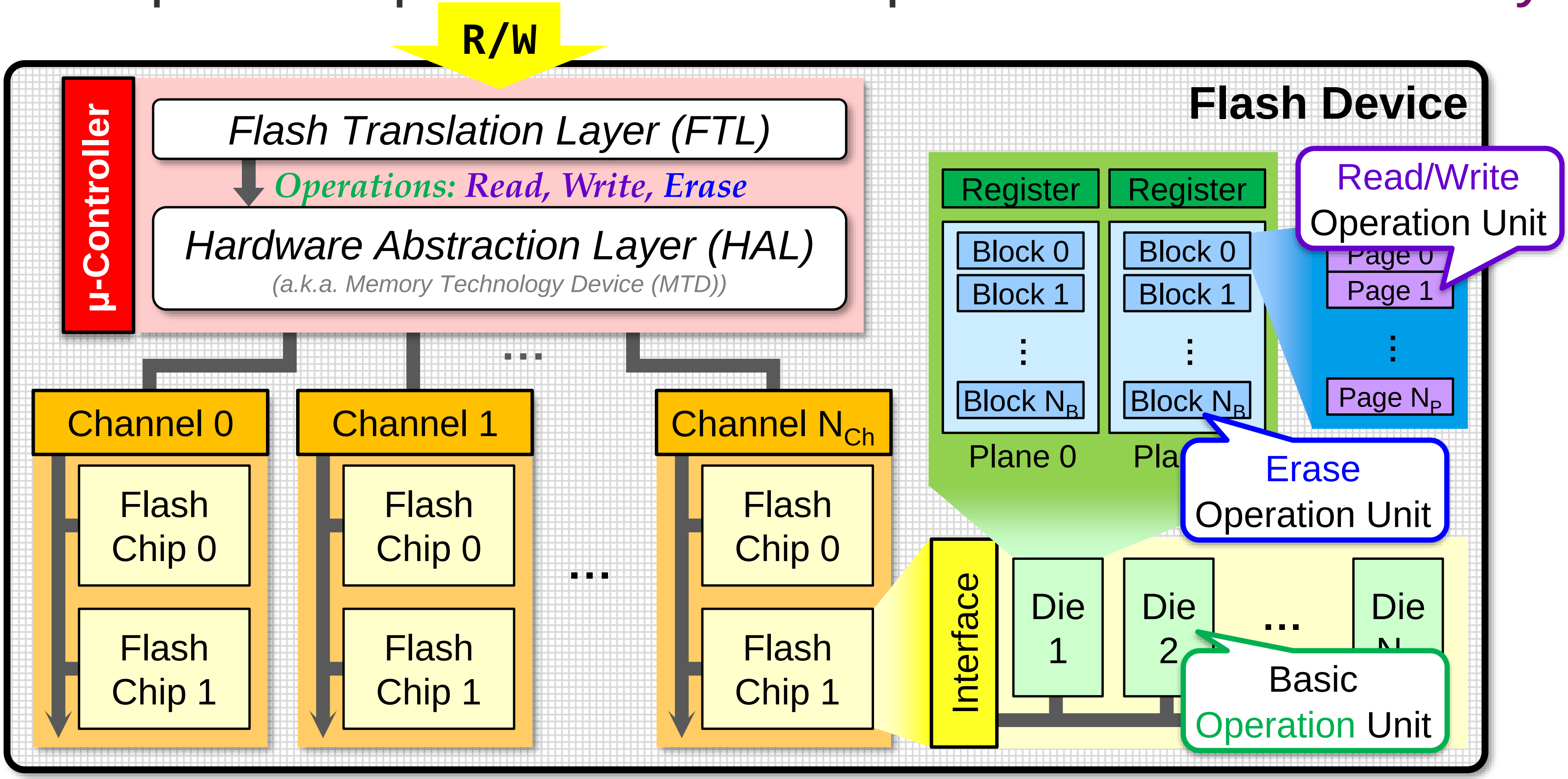
- **Key Observations: Flash is not perfect!**
  - P/E cycles might inaccurately reflect the reliability of flash.
  - Blocks might have different **bit-error rates (BERs)** when enduring the same P/E cycles due to **process variation**.
- **Idea: BER** could be a **better metric** to WL designs.
  - The error correction hardware can report BER to FTL.



New ERA: New Efficient Reliability-Aware Wear Leveling for Endurance Enhancement of Flash Storage Devices (DAC'13)

Multilevel I/O Parallelism (1/2)

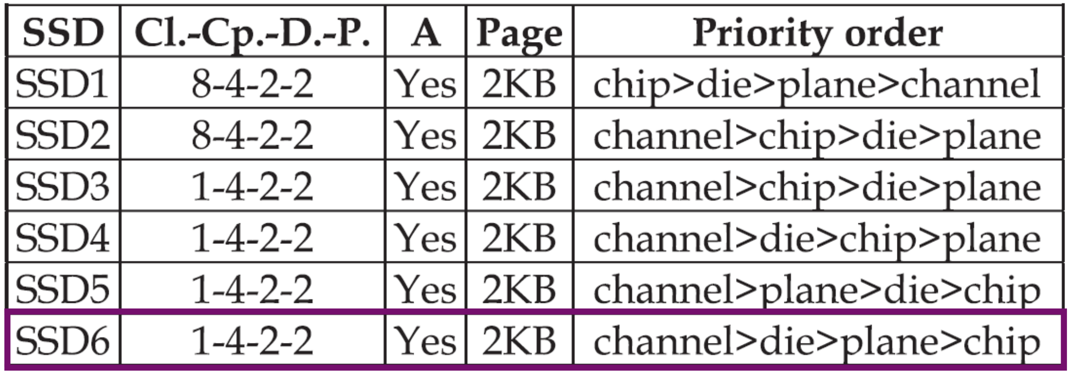
- The internal of flash devices is **highly hierarchical**:
  - **Channel → Chip → Die → Plane** → Block → Page
- Multiple I/O operations can be performed concurrently.



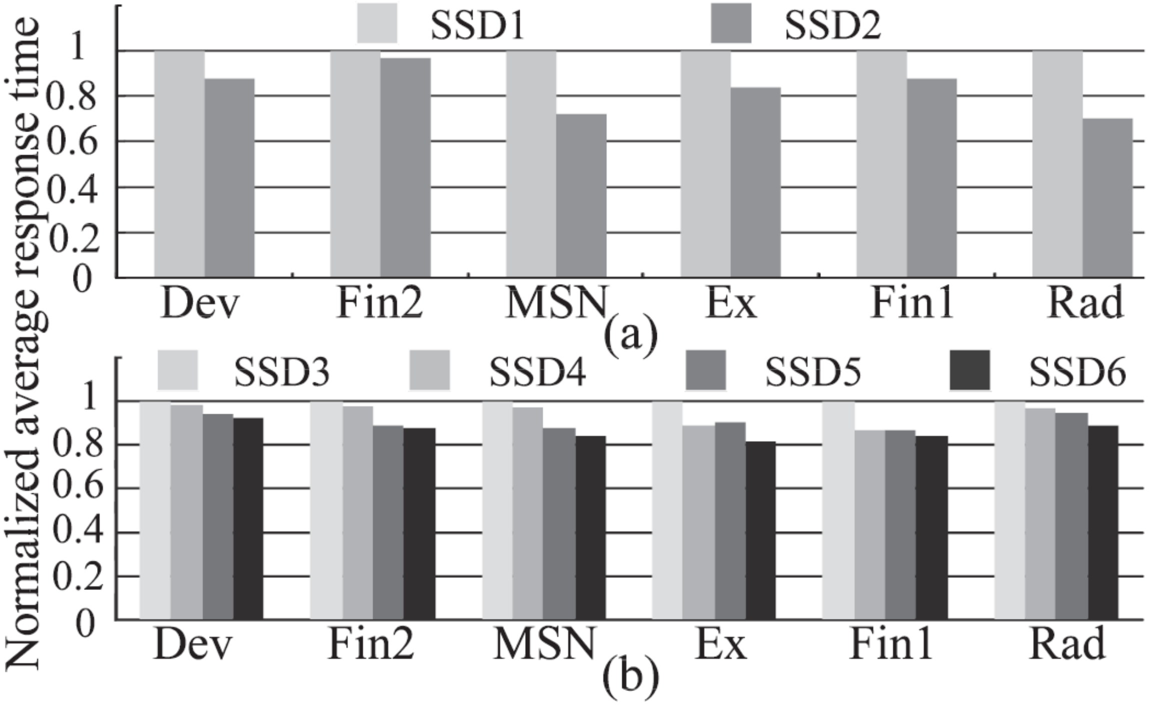
Exploring and Exploiting the Multilevel Parallelism Inside SSDs for Improved Performance and Endurance (TC'13)

Multilevel I/O Parallelism (2/2)

- The optimal priority order of parallelism should be:
  - ① Channel-level
  - ② Die-level
  - ③ Plane-level
  - ④ Chip-level



SSD	Cl.-Cp.-D.-P.	A	Page	Priority order
SSD1	8-4-2-2	Yes	2KB	chip>die>plane>channel
SSD2	8-4-2-2	Yes	2KB	channel>chip>die>plane
SSD3	1-4-2-2	Yes	2KB	channel>chip>die>plane
SSD4	1-4-2-2	Yes	2KB	channel>die>chip>plane
SSD5	1-4-2-2	Yes	2KB	channel>plane>die>chip
SSD6	1-4-2-2	Yes	2KB	channel>die>plane>chip

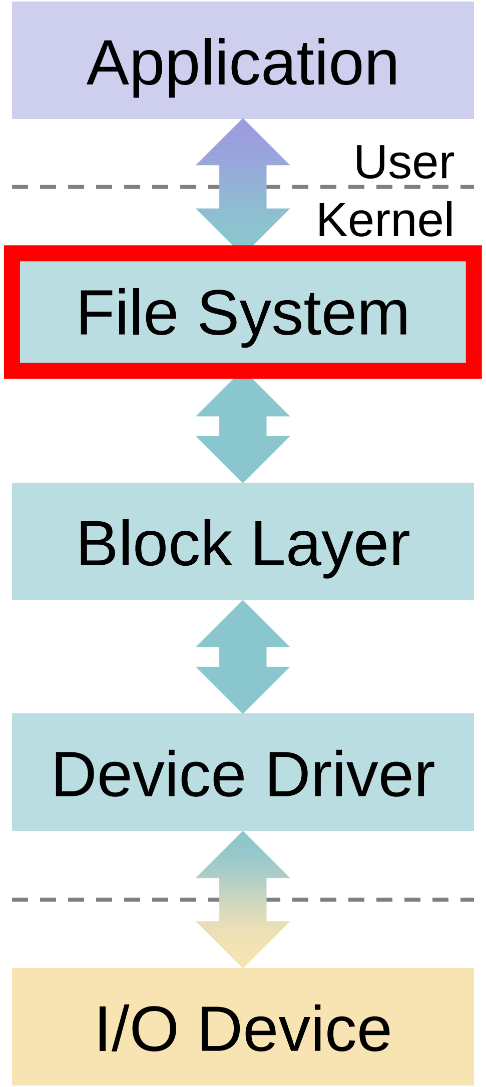


Exploring and Exploiting the Multilevel Parallelism Inside SSDs for Improved Performance and Endurance (TC'13)

Outline

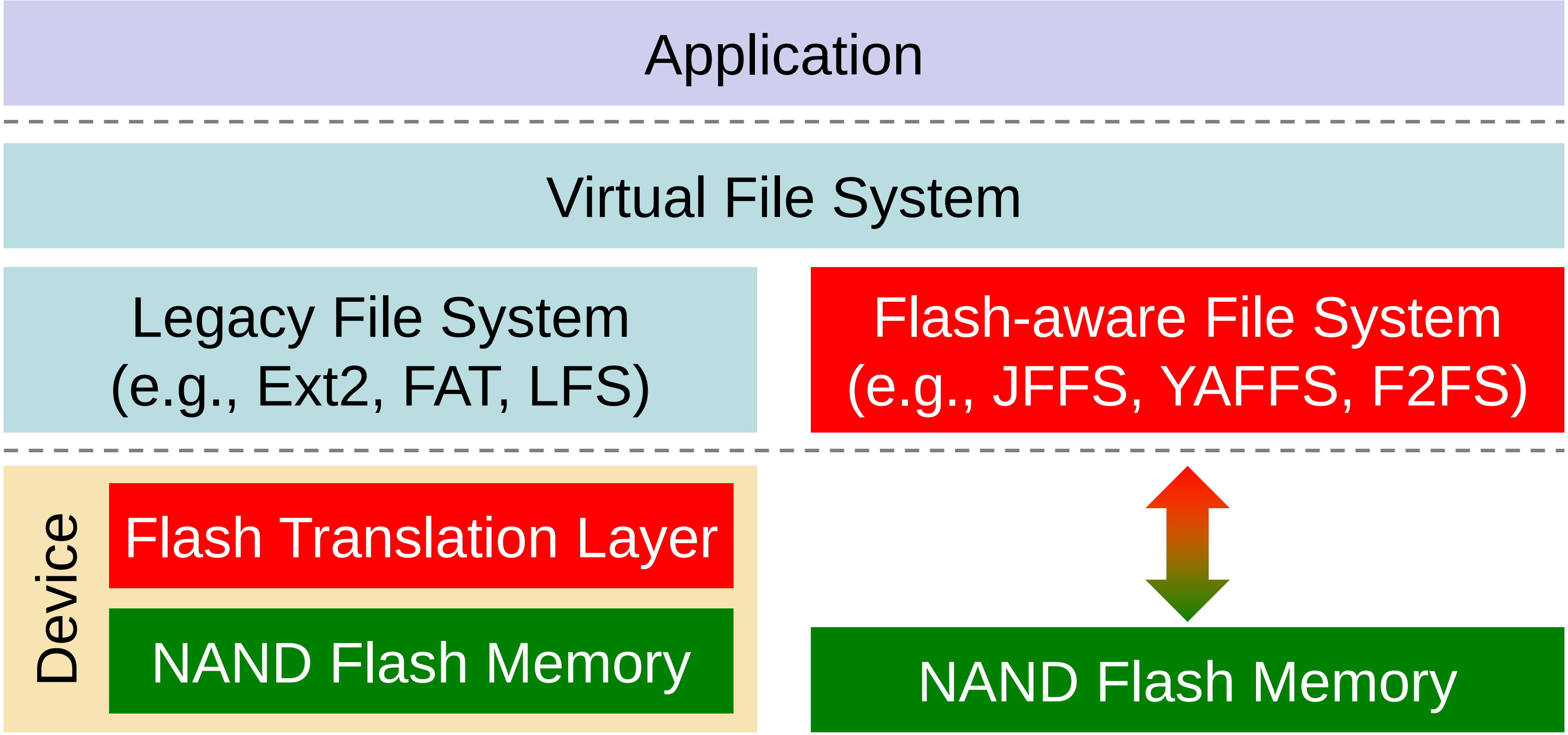
- Flash Memory: Why and How
  - NAND Flash Technology
  - Inherent Challenges
- System Architecture
- Flash Translation Layer
  - Address Mapping
  - Garbage Collection
  - Wear Leveling
  - Multilevel I/O Parallelism
- Flash-aware File System
  - Flash-Friendly File System (F2FS)

I/O Stack



Recall: System Architecture

- There are two typical ways to address the inherent challenges of flash memory:
  - ① Implementing a **Flash Translation Layer** in the **device**.
  - ② Designing a **Flash-aware File System** in the **host**.

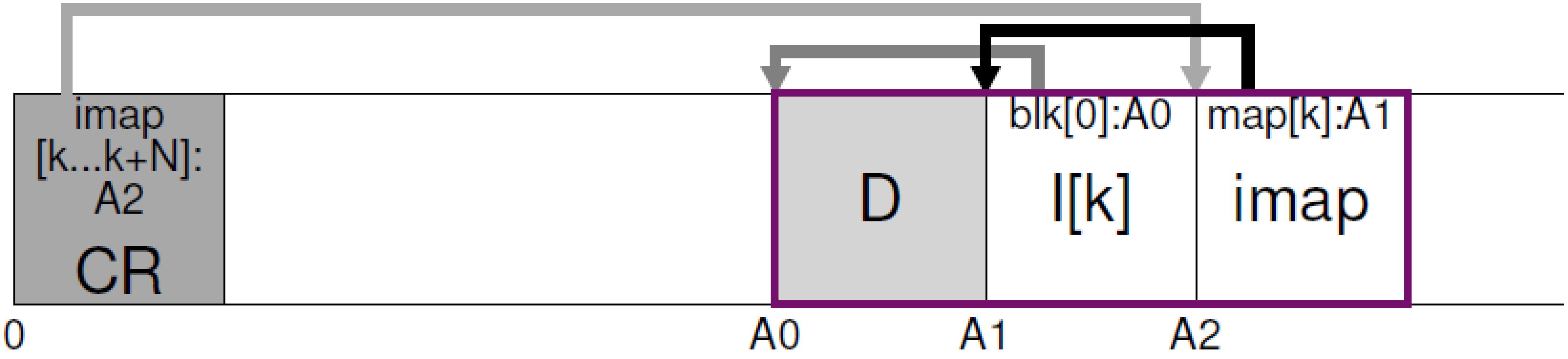


Flash-aware File System

- Random writes are bad to flash devices.
  - Free space fragmentation
  - Degraded performance (due to GC)
  - Reduced lifetime (due to GC)
- Writes must be reshaped into sequential writes.
  - Same as Log-structured file system (LFS) for HDD!
- Most flash-aware file systems are derived from LFS:
  - Journaling Flash File System (JFFS)
  - Yet Another Flash File System (YAFFS)
  - Flash-Friendly File System (F2FS)
    - Publicly available, included in Linux mainline kernel since Linux 3.8.

Recall: Log-structured File System

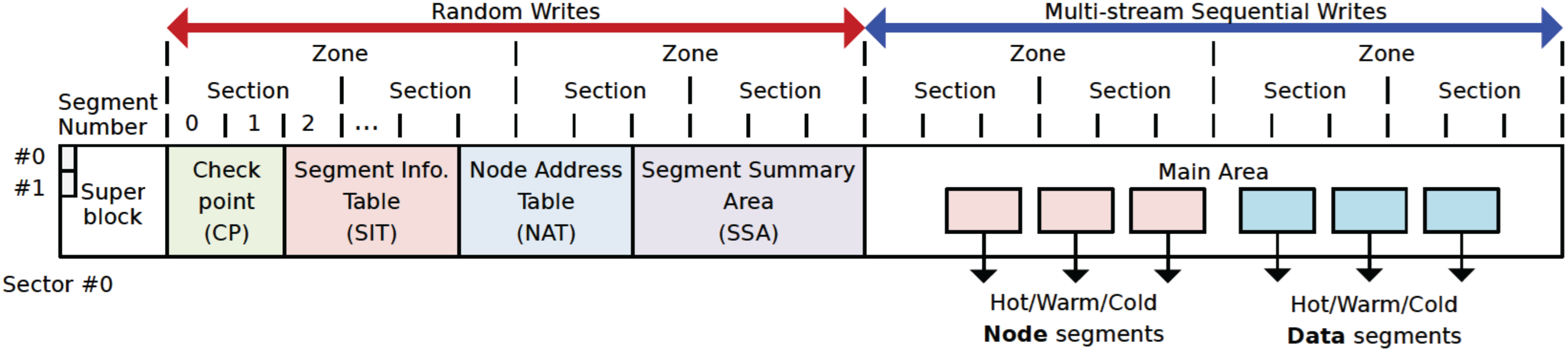
- LFS first buffers all writes in an in-memory **segment** and commits the segment to disk sequentially.
  - The Inode Map (**`imap`**)
    - Maps from an `inode-number` to the `disk-address` of the *most recent version* of the inode (i.e., one more mapping!).
    - Updated whenever an inode is written to disk.
    - Placed right next to where data block (**`D`**) and inode (**`I[k]`**) reside.
  - The Checkpoint Region (**`CR`**):
    - Records disk pointers to all latest pieces of **`imap`**.
    - Flushed to disk periodically (e.g., every 30 seconds).



Flash-friendly On-Disk Layout

- **Flash Awareness**
  - All the FS metadata are located together for locality.
  - Start address of main area is aligned to the **zone** size.
    - **block**=4KB; **segment**=2MB; **section**=n segments; **zone**=m sections.
  - File system cleaning (i.e., GC) is done in a unit of **section**.

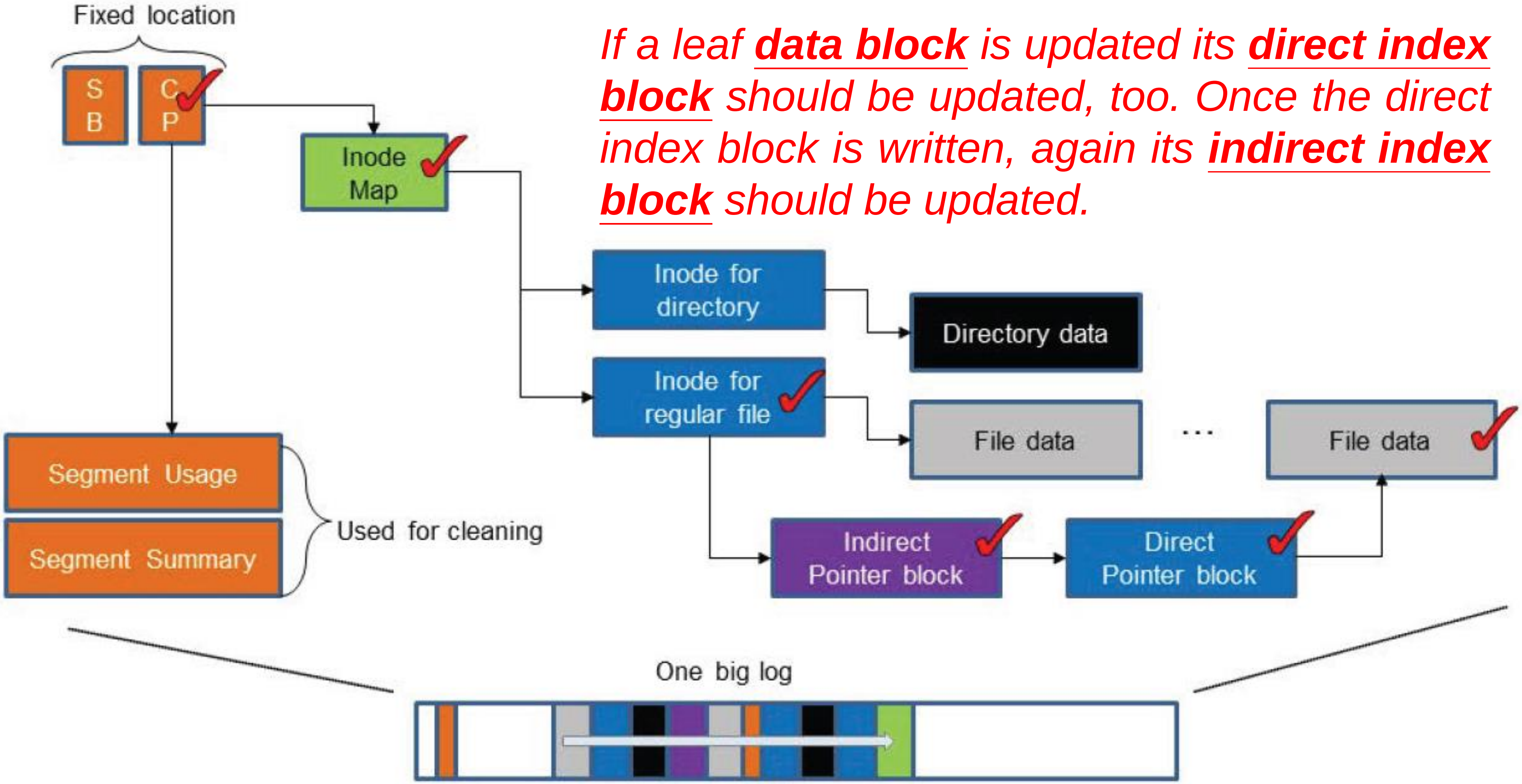
- **Cleaning Cost Reduction**
  - Multi-head logging for hot/cold data separation.



LFS Index Structure

- LFS manages the disk space as one big log.
- LFS has the update propagation problem.

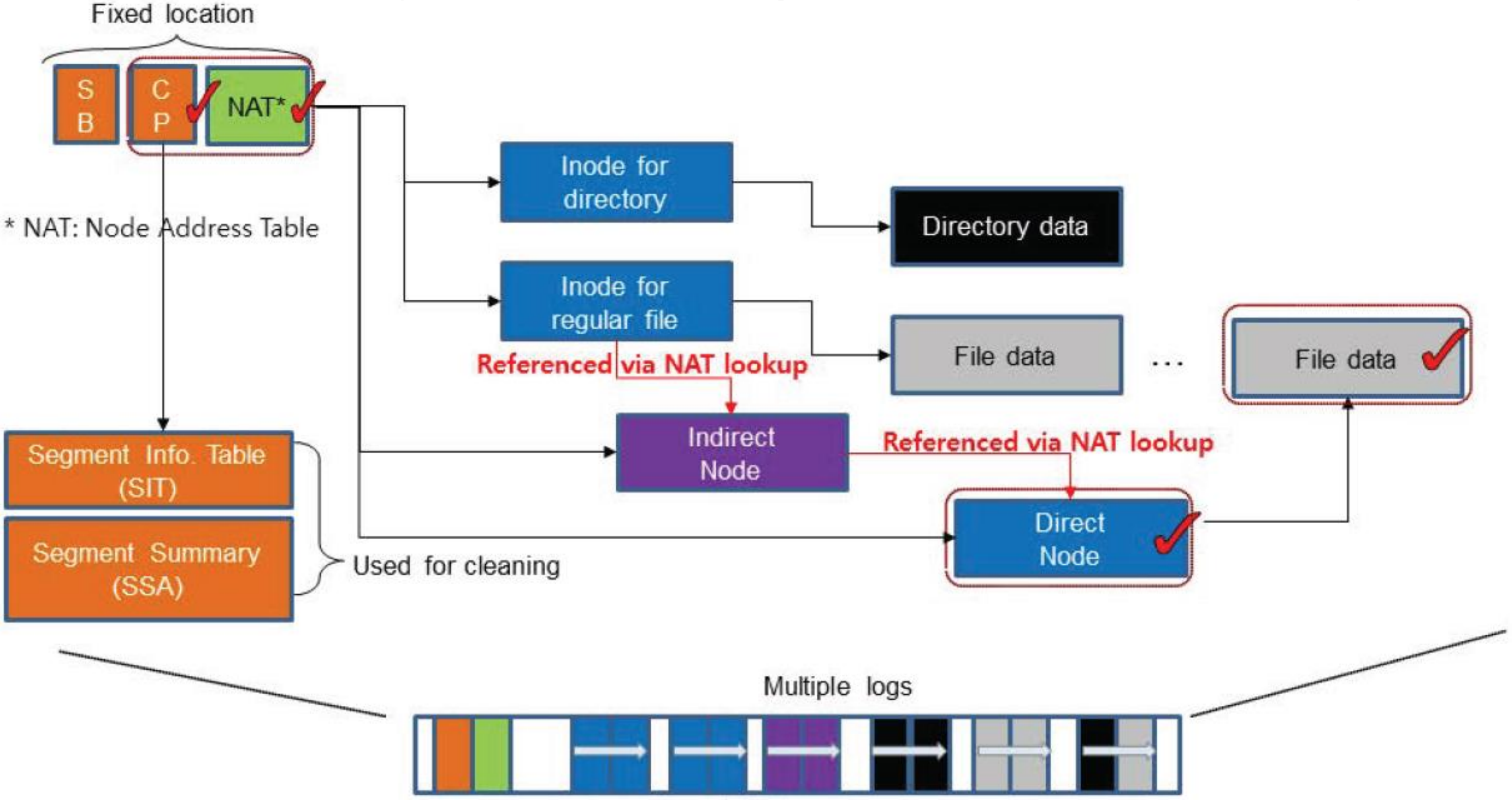
*If a leaf **data block** is updated its **direct index block** should be updated, too. Once the direct index block is written, again its **indirect index block** should be updated.*



https://www.usenix.org/sites/default/files/conference/protected-files/fast15_slides_lee.pdf

F2FS Index Structure

- Restrained update propagation by *node address table*.
- F2FS manages the flash space as *multi-head log*.



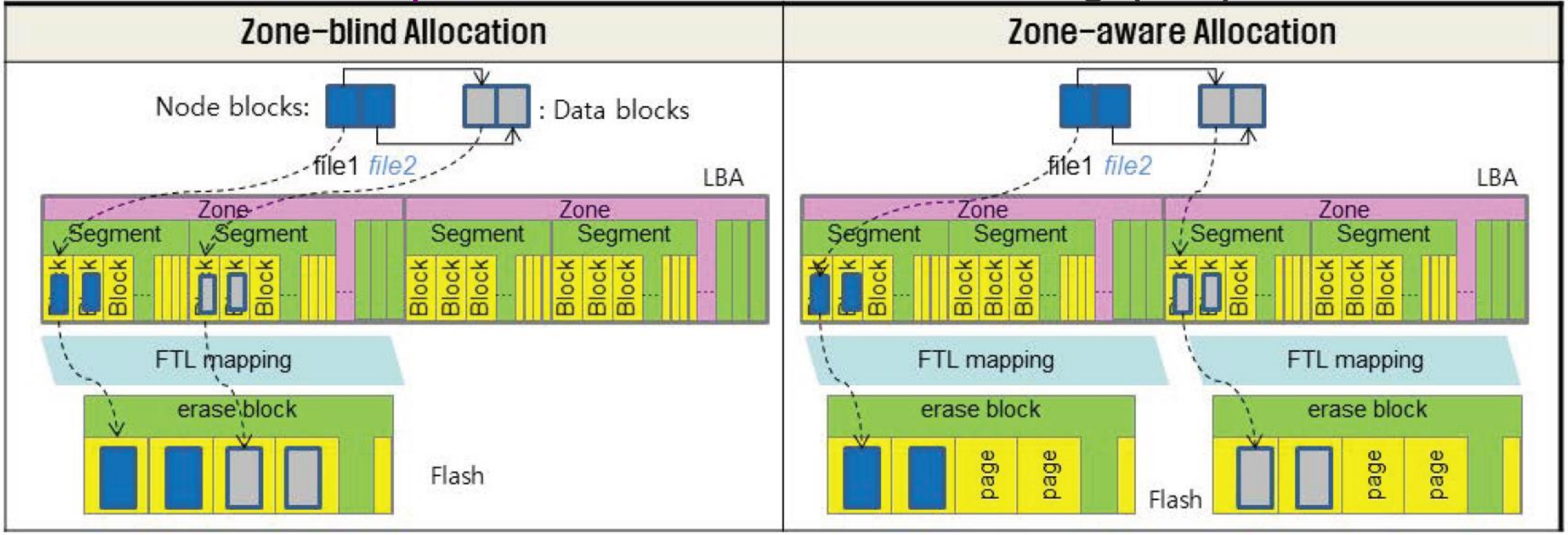
https://www.usenix.org/sites/default/files/conference/protected-files/fast15_slides_lee.pdf

Multi-head Logging

- **Data temperature**:
  - Node > Data
  - Direct Node > Indirect Node
  - Directory > User File

Type	Temp.	Objects
Node	Hot	Direct node blocks for directories
	Warm	Direct node blocks for regular files
	Cold	Indirect node blocks
Data	Hot	Directory entry blocks
	Warm	Data blocks made by users
	Cold	Data blocks moved by cleaning; Cold data blocks specified by users; Multimedia file data

- **Separation of multi-head logs** in NAND flash
  - Hot/cold separation reduces the cleaning (GC) overhead.



https://www.usenix.org/sites/default/files/conference/protected-files/fast15_slides_lee.pdf

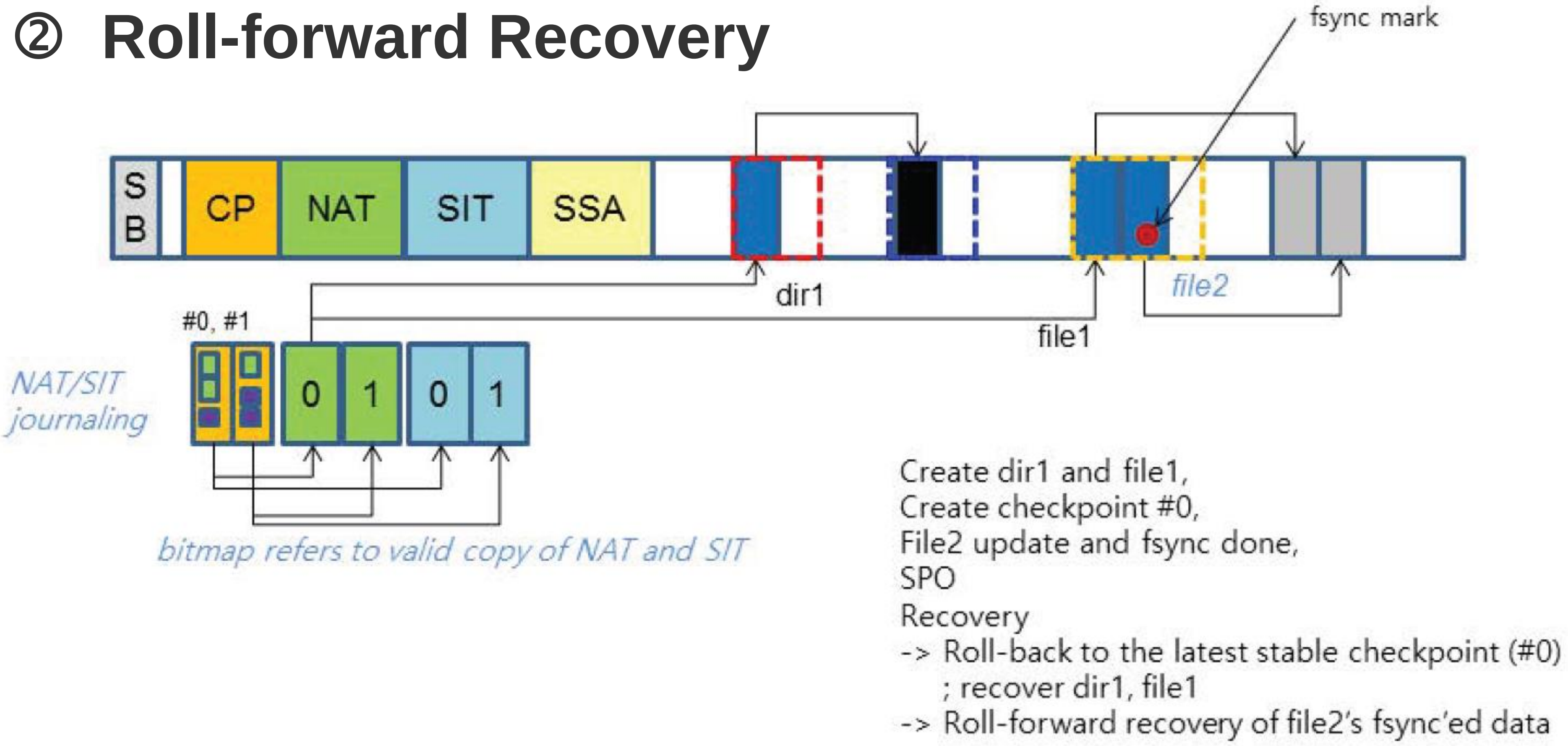
Crash Recovery

① Checkpoint

– Maintain shadow copy of checkpoint, NAT, SIT blocks

– Recovers the latest checkpoint

② Roll-forward Recovery



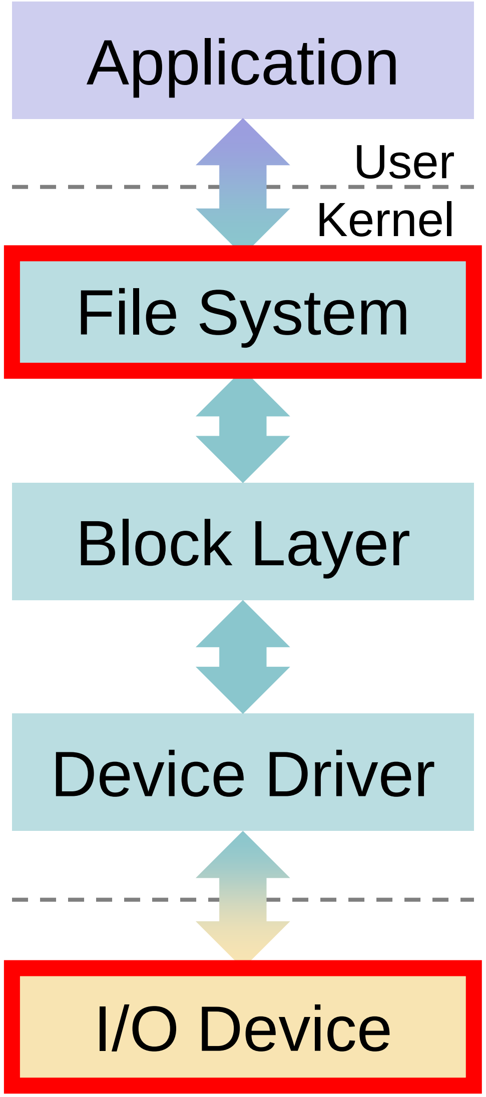
Create dir1 and file1,
Create checkpoint #0,
File2 update and fsync done,
SPO
Recovery
-> Roll-back to the latest stable checkpoint (#0)
; recover dir1, file1
-> Roll-forward recovery of file2's fsync'ed data

https://www.usenix.org/sites/default/files/conference/protected-files/fast15_slides_lee.pdf

Summary

- Flash Memory: Why and How
  - NAND Flash Technology
  - Inherent Challenges
- System Architecture
- Flash Translation Layer
  - Address Mapping
  - Garbage Collection
  - Wear Leveling
  - Multilevel I/O Parallelism
- Flash-aware File System
  - Flash-Friendly File System (F2FS)

I/O Stack


Application
User
Kernel
File System
Block Layer
Device Driver
I/O Device