07. Flash File Systems

Special Topics in Computer Systems:
Modern Storage Systems
(SE820-01)

Instructor:

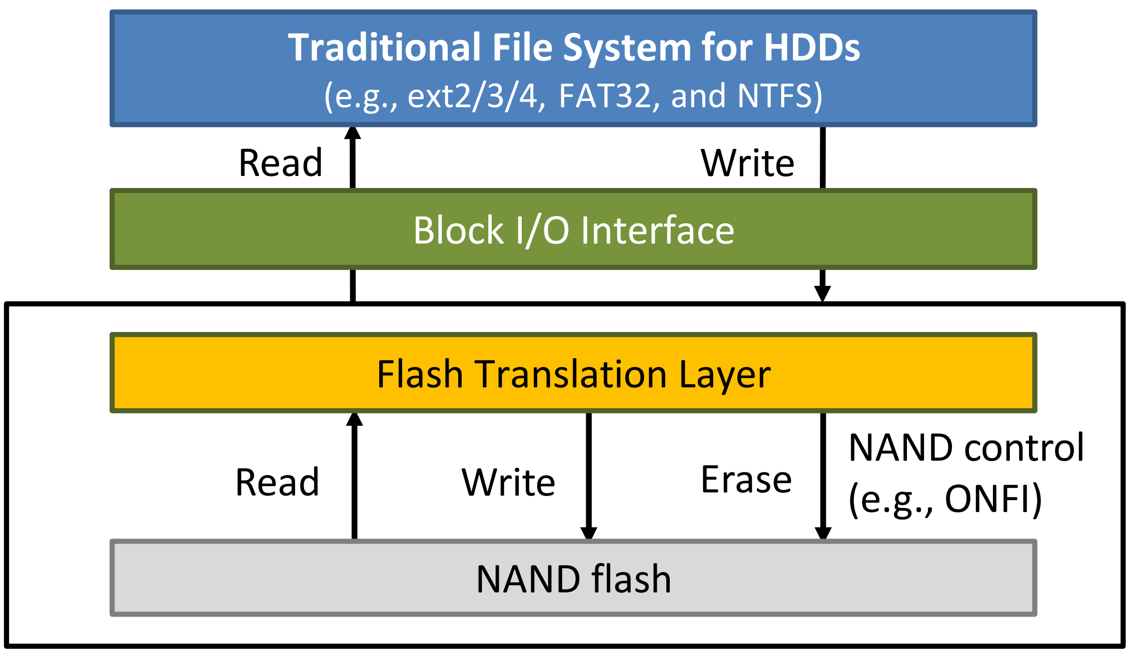
Prof. Sungjin Lee (sungjin.lee@dgist.ac.kr)

Outline

- **Traditional Flash File Systems**
- **SSD-Friendly Flash File Systems**
  - F2FS: Flash-friendly File System
- **Reference**

Traditional File Systems for Flash

- **Originally designed for block devices like HDDs**
  - e.g., ext2/3/4, FAT32, and NTFS
- **But, NAND flash memory is not a block device**
  - The FTL provides block-device views outside, hiding the unique properties of NAND flash memory

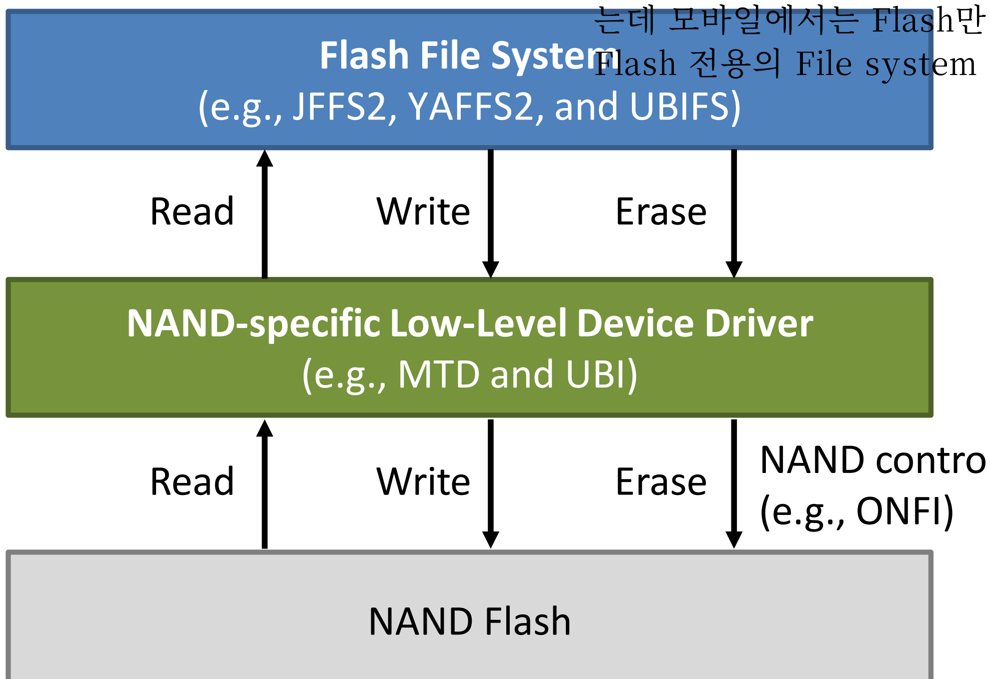


Flash-based SSDs

Flash File Systems

- **Directly manage raw NAND flash memory**
  - Internally performing address mapping, garbage collection, and wear-leveling by itself
- **Representative flash file systems**
  - JFFS2, YAFFS2, and UBIFS

FFS는 FS 단계에서 바로 FTL의 기능 + 그에 맞춘 block
interface를 사용함으로써
역할을 할 수 있는만큼 더 좋다.
(기존 방식은 HDD용으로 만든 것을 호환성을 유지하면서
Flash를 위해 사용할 수 있도록 block interface를 사용하
는데 모바일에서는 Flash만 쓰니까 호환성 고려 없이
Flash 전용의 File system + interface를 사용하는 것.

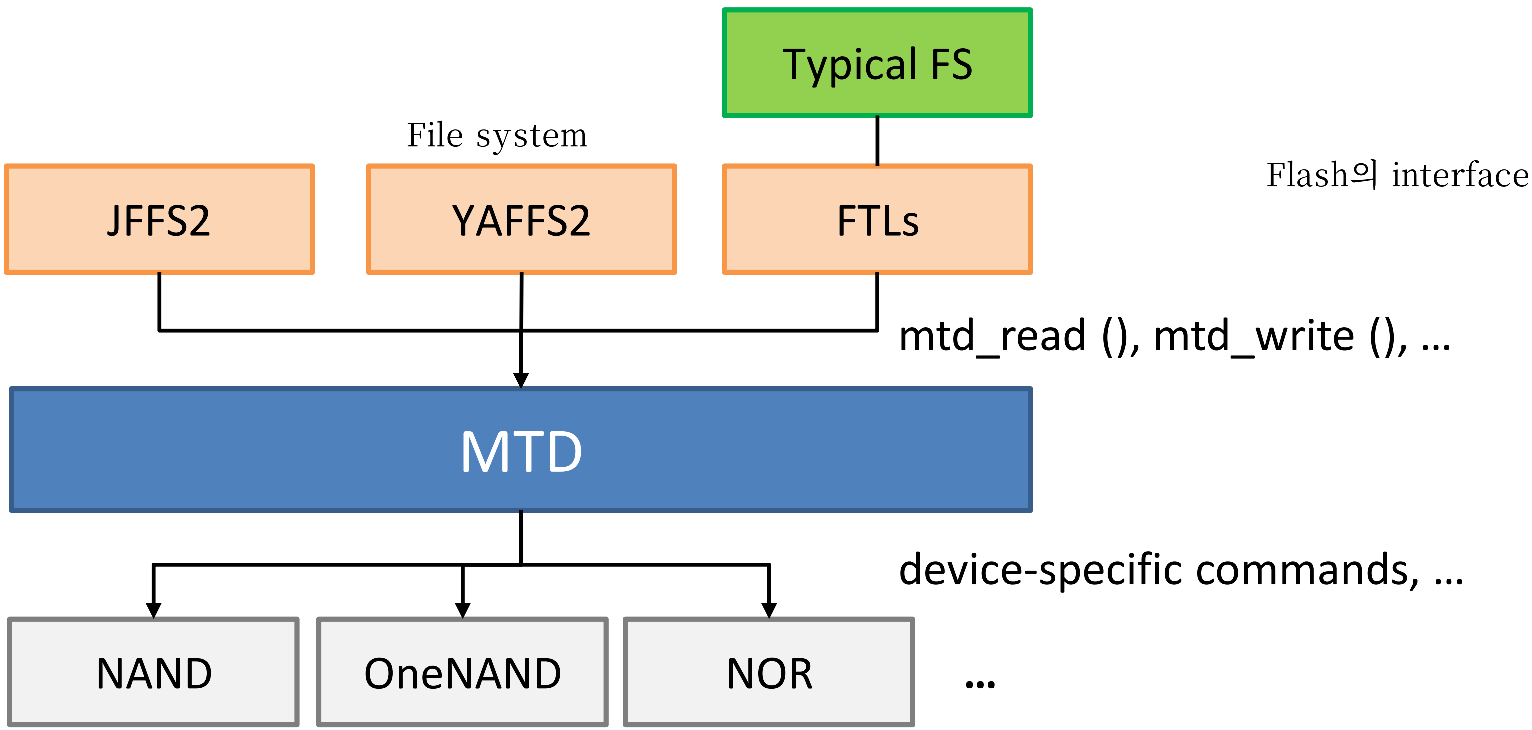


Memory Technology Device (MTD)

- **MTD is the lowest level for accessing flash chips**
  - Offer the same APIs for different flash types and technologies
    - e.g., NAND, OneNAND, and NOR

block interface를 대체하는 interface.

- **JFFS2 and YAFFS2 run on top of MTD**



Traditional File Systems vs. Flash File Systems

Flash 자체는 계속 성질이 바뀌어서..
안드로이드 초반까지는 FFS를 썼다.

	File System + FTL	Flash File System
Method	- Access a flash device via FTL	- Access a flash device directly
Pros	- High interoperability - No difficulties in managing recent NAND flash with new constraints	- High-level optimization with system-level information - Flash-aware storage management
Cons	- Lack of system-level information - Flash-unaware storage management	- Low interoperability - Must be redesigned to handle new NAND constraints

단점 해결을 위해
추가적인
interface
추가

상업적으로 FFS는 성공하지 못했는데,
Flash chip에 대해 약간만 바뀌어도 완전히 새로 짜야 한다..

Flash file systems now become obsolete because of difficulties for the adoption to new types of NAND devices

다만, 그래도 기존의 FS를 Flash와 같이 쓰는 건
비효율적인 면이 많으므로
F2FS같이 Flash 전용 File system
(이면서 FFS처럼 chip이 바뀌어도 괜찮은_
이 나왔다.

JFFS2: Journaling Flash File System

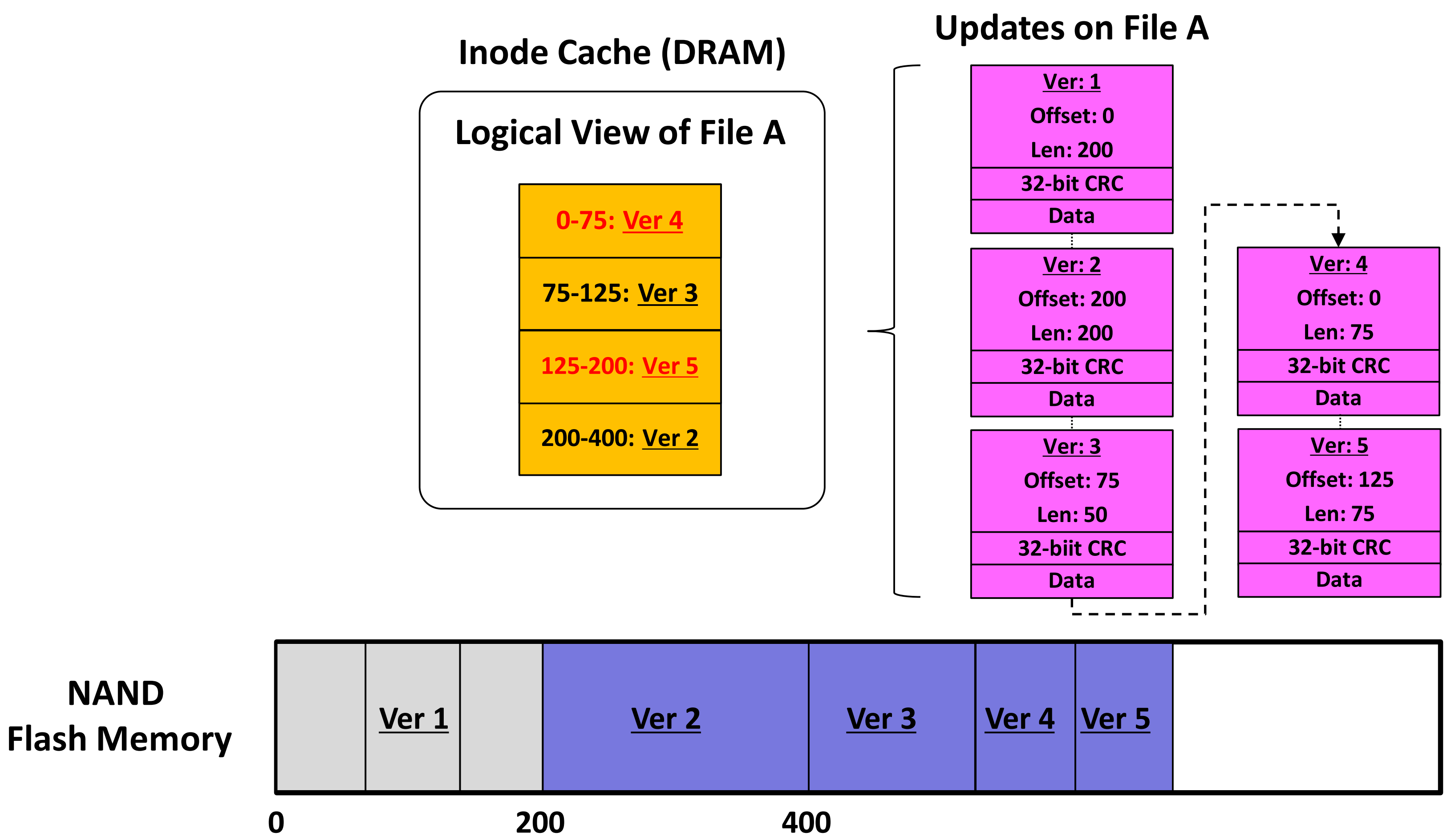
- **A log-structured file system (LFS) for use with NAND flash**
  - Unlike LFS, however, it *does not allow* any in-place updates!!!

- **Main features of JFFS2**
  - *File data* and *metadata* stored as nodes in NAND flash memory
  - Keep an inode cache holding the information of nodes in DRAM
  - A greedy garbage collection algorithm
    - Select cheapest blocks as a victim for garbage collection
  - A simple wear-leveling algorithm combined with GC
    - Consider the wearing rate of flash blocks when choosing a victim block for GC
  - Optional data compression

victim block을 고를 때
100번 중에 99번은 cost 적은걸로 고르다가
마지막 한번은 random하게 해서 wear - leveling
하는 건데, 대충 구현되있다.

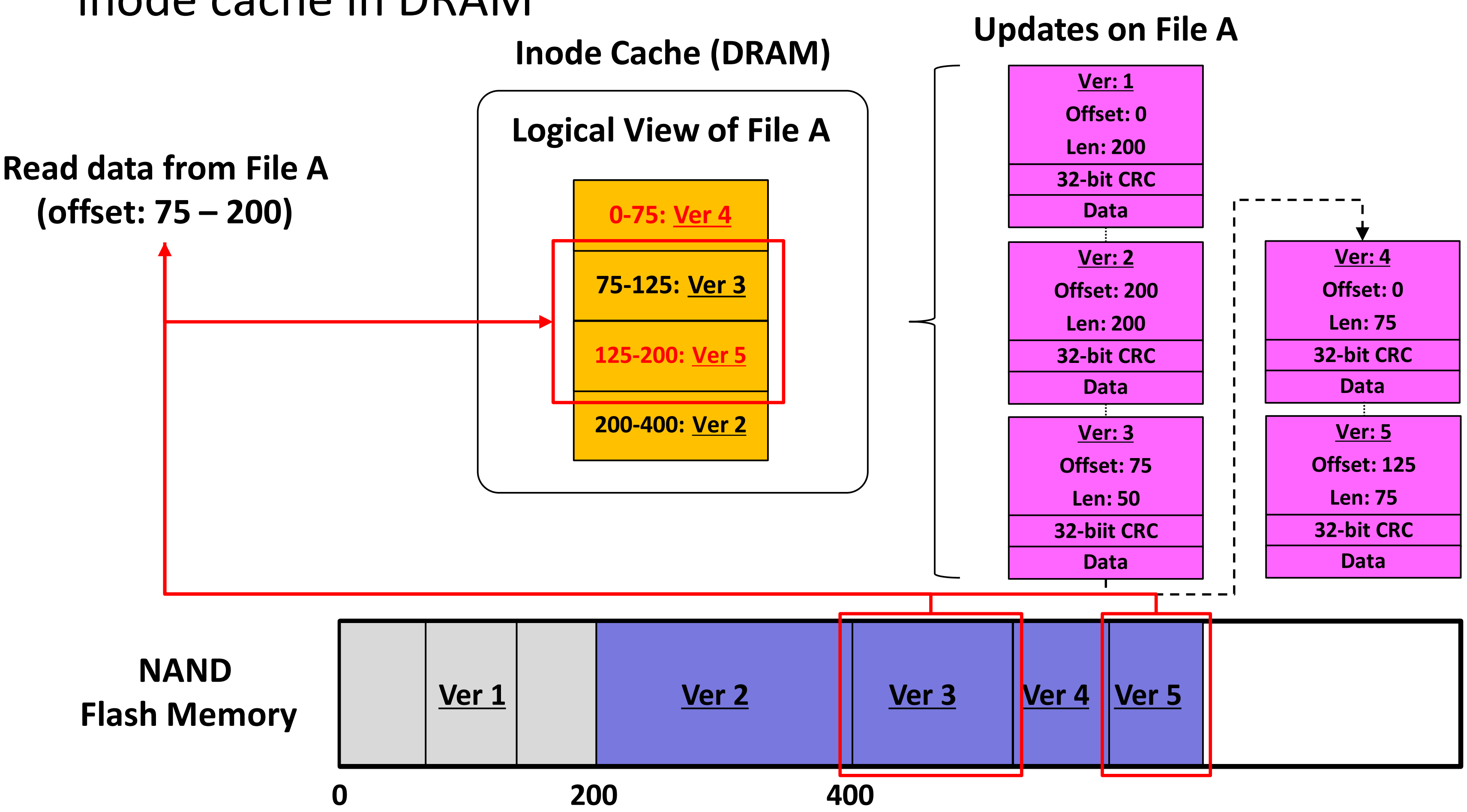
JFFS2: Write Operation

- All data are written sequentially to a log which records all changes



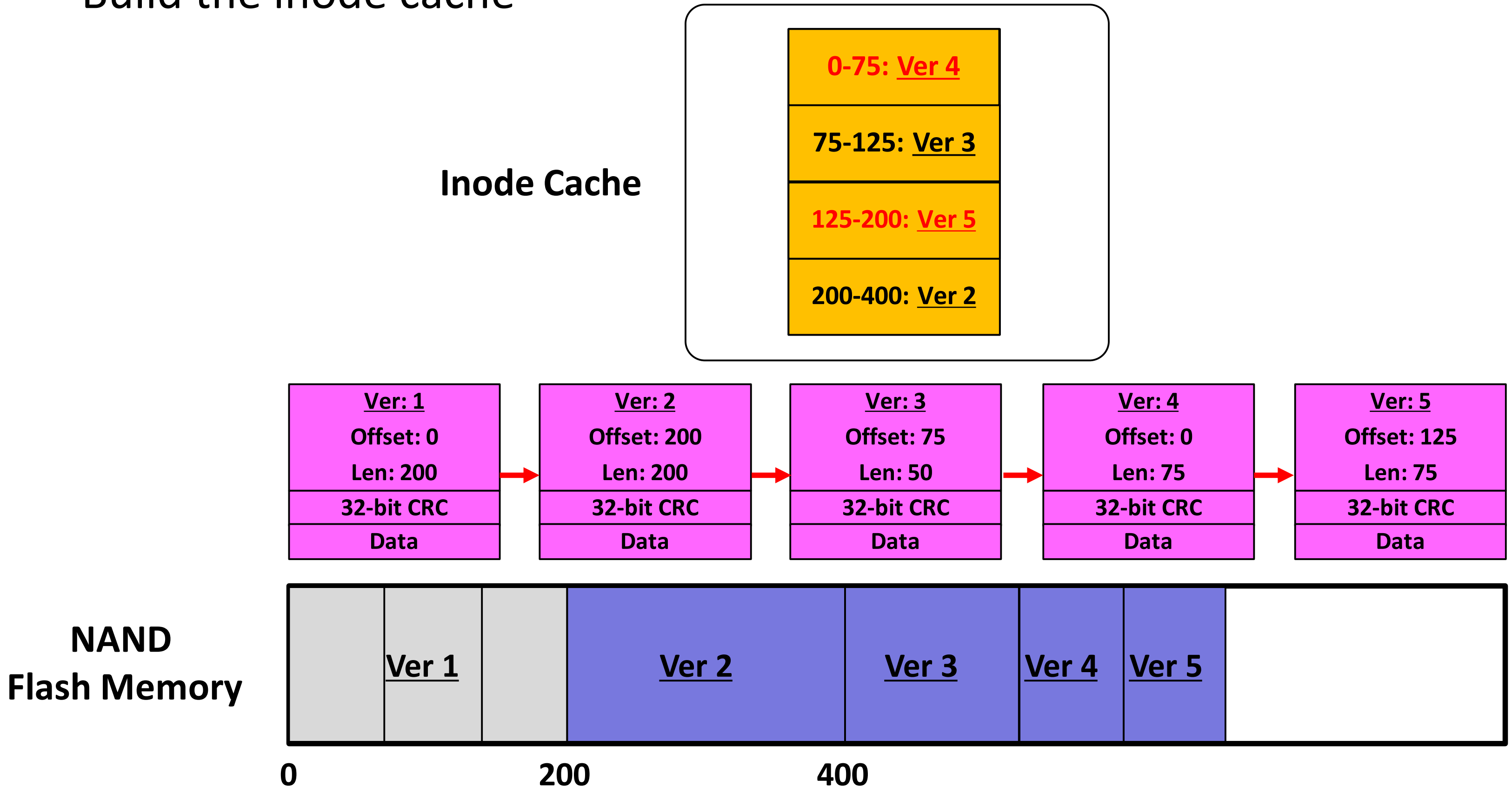
JFFS2: Read Operation

- The latest data can be read from NAND flash by referring to the inode cache in DRAM



JFFS2: Mount

- **Scan the flash memory medium after rebooting**
  - Check the CRC for written data and mark the obsolete data
  - Build the inode cache



JFFS2: Problems

- **Slow mount time**

  inode cache를 복구하려면 모든 node를 scan해야 한다.

  - All nodes must be scanned at mount time
  - Mount time increases in proportion to flash size and file system contents

- **High memory consumption**

  모든 node 정보를 DRAM에 유지해야 하므로
  DRAM을 많이 차지한다.

  - All node information must be maintained in DRAM
  - Memory consumption linearly depends on file system contents

- **Low scalability**

  size가 커지면 앞의 2가지 이유로
  scalability도 낮다.

  - Infeasible for a large-scale flash device
  - Mount time and memory consumption increase according to a flash size

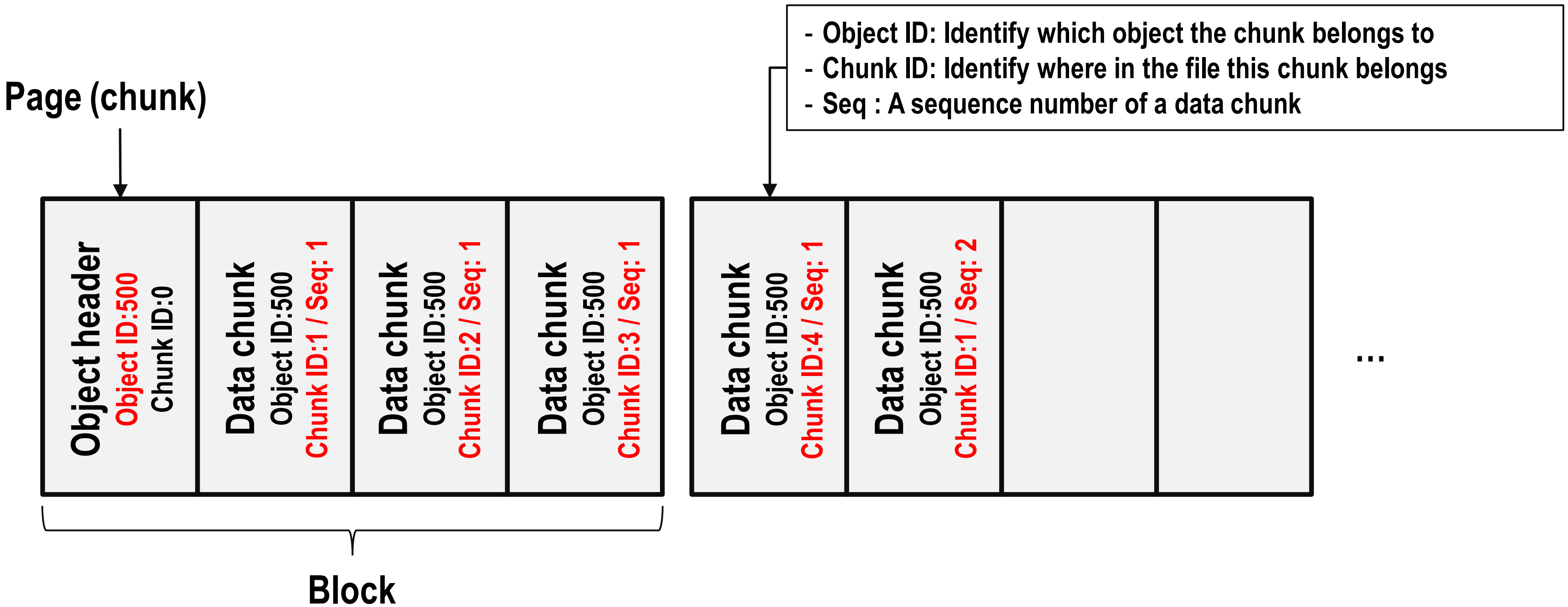
FS가 없어서 redhat에서 임시로 만든 건데
오픈소스 기반이고 다른 게 없어서 많이 사용했다.
애초에 임베디드 용이라 큰 문제는 없었다.

YAFFS2: Yet Another Flash File System

- **Another log-structured file system for flash memory**
  - Store data to flash memory like a log with a unique sequence number like JFFS2
  - Reads and writes are performed similar to JFFS2
- **Mitigate the problems raised by JFFS2**
  - Relatively frugal with memory resource
  - Checkpoints for a fast file system mount
  - Dynamic wear-leveling
- **Support multiple platforms**
  - Linux, WinCE, RTOSs, etc

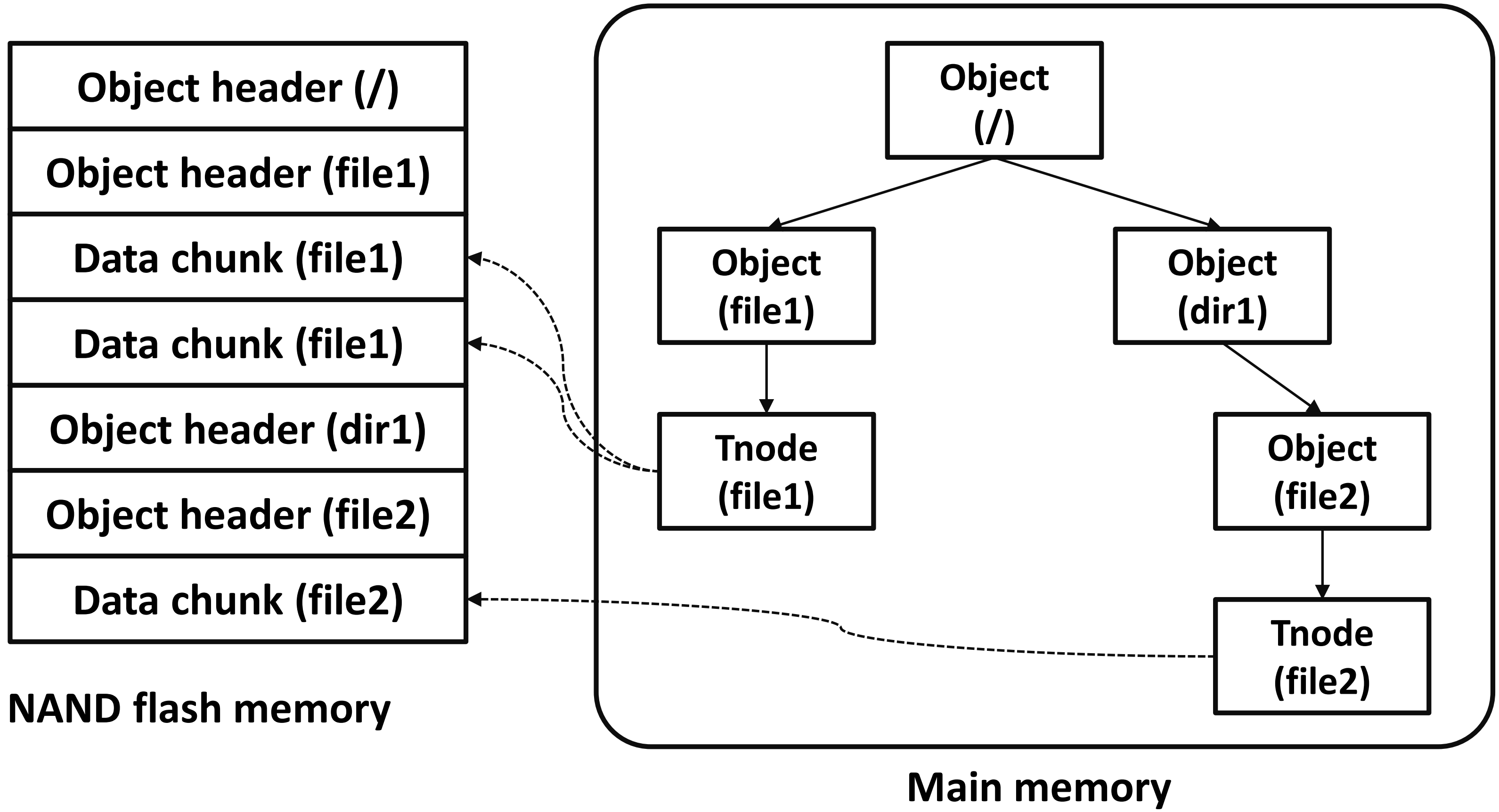
YAFFS2: Physical Layout

- **The entries in the log are all one chunk (one page) in size and can hold one of two types of chunk**
  - Data chunk: A chunk holding regular data file contents
  - Object header: A descriptor for an object, such as a directory, a regular file, etc; similar to `struct stat` but include dentry



YAFFS2: File System Layout

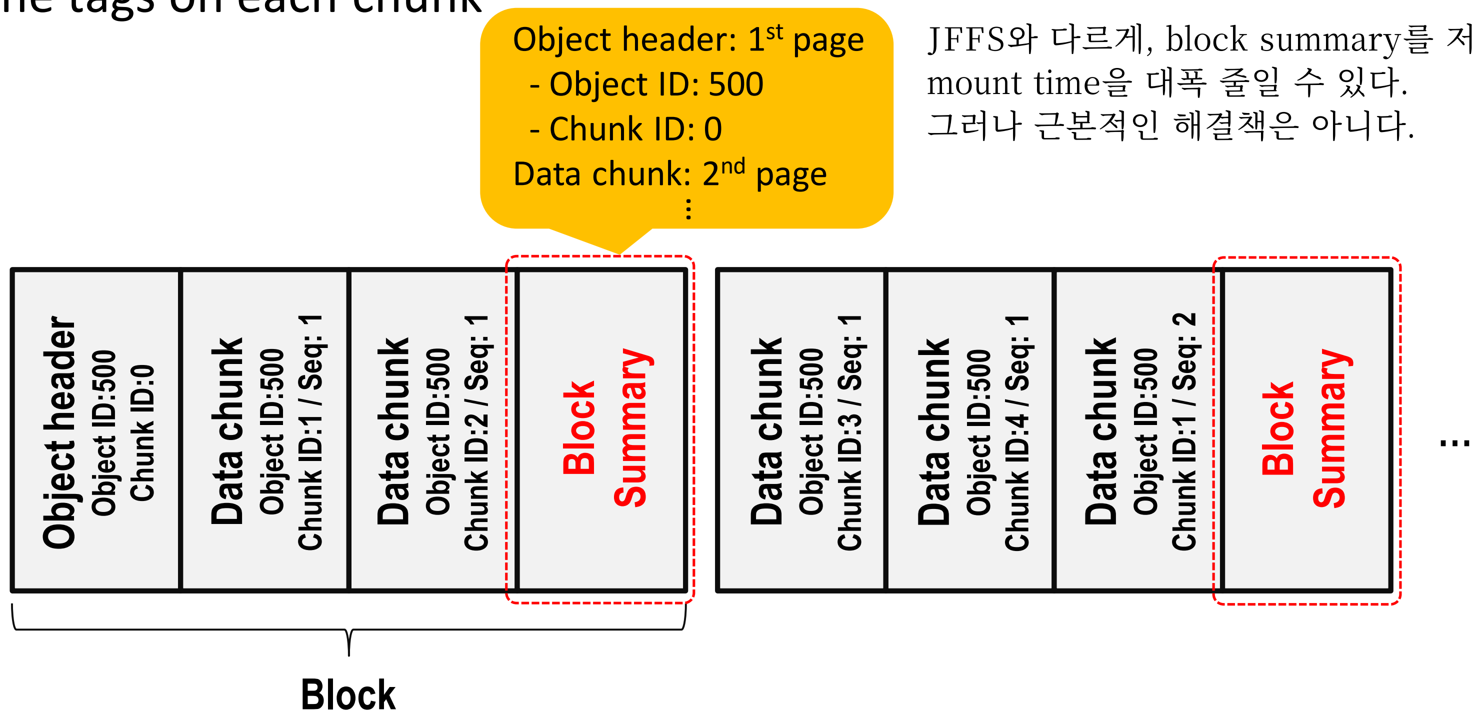
- Maintain the information about objects and chunks in DRAM



YAFFS2: Block Summary

- A block summary (including object id and chunk id) for chunks in the block are written to the last chunk
- This allows all the tags for chunks in that block to be read in one hit, avoiding full disk scan
- If a block summary is not available for a block, then a full scan is used to get the tags on each chunk

JFFS와 다르게, block summary를 저장하여 mount time을 대폭 줄일 수 있다.
그러나 근본적인 해결책은 아니다.



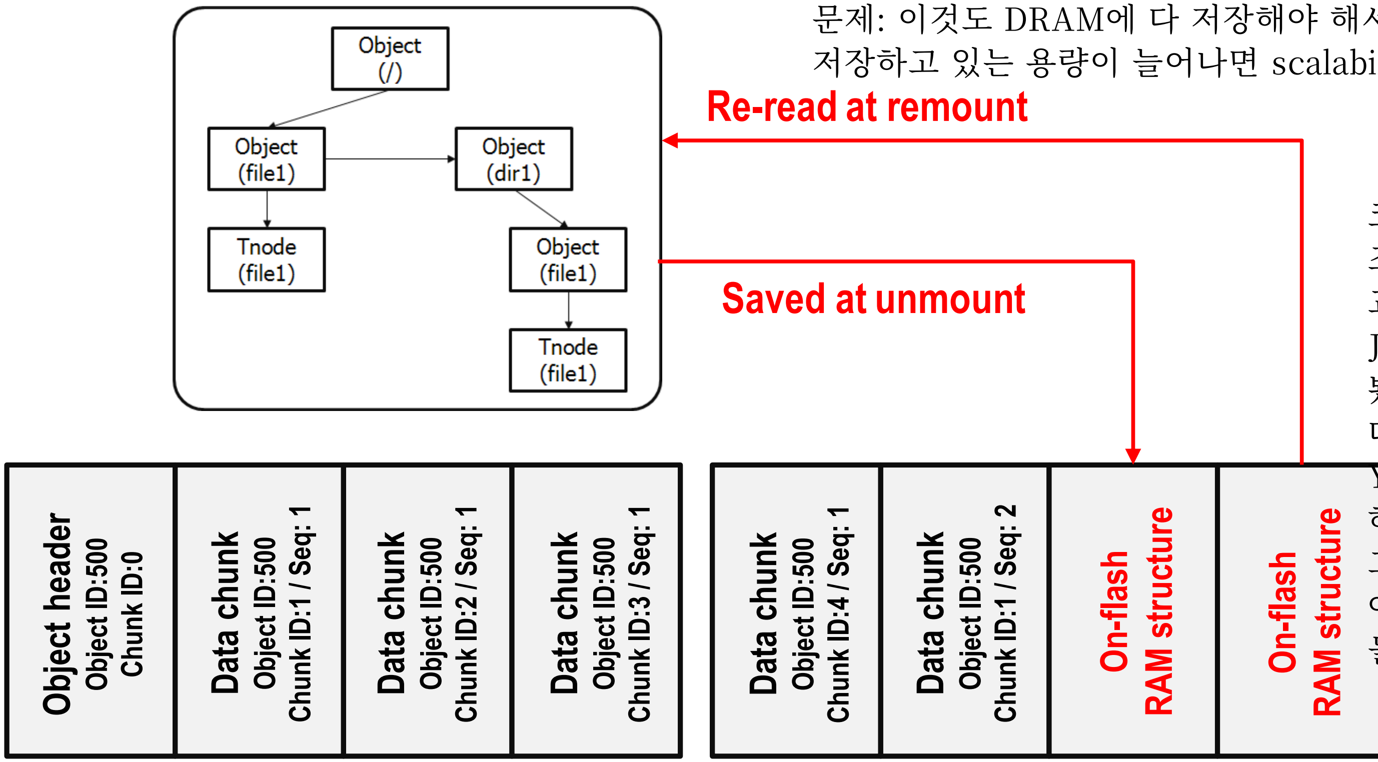
YAFFS2: Checkpoints

- DRAM structures are saved on flash at unmount
- Structures re-read, avoiding boot scan
- Lazy loading also reduces mount time

정상 종료시에는
DRAM 내용물을 Flash에 저장하여
boot scan 자체를 안해도 되도록.

그럼 Failure는? summary block 읽겠지.

문제: 이것도 DRAM에 다 저장해야 해서
저장하고 있는 용량이 늘어나면 scalability가 떨어진다.



코드가 JFFS보다
조금 더 깔끔하다.
교수님은 박사때
JFFS를 1년동안 수정해
봤는데 코드가 너무 더럽
다.
YAFFS는 코드를 깔끔
하게 짰다.
그래도 급하게 짠 것이고
아직 시스템적인 철학이
들어있지 않다.

UBIFS: Unsorted Bock Image File System

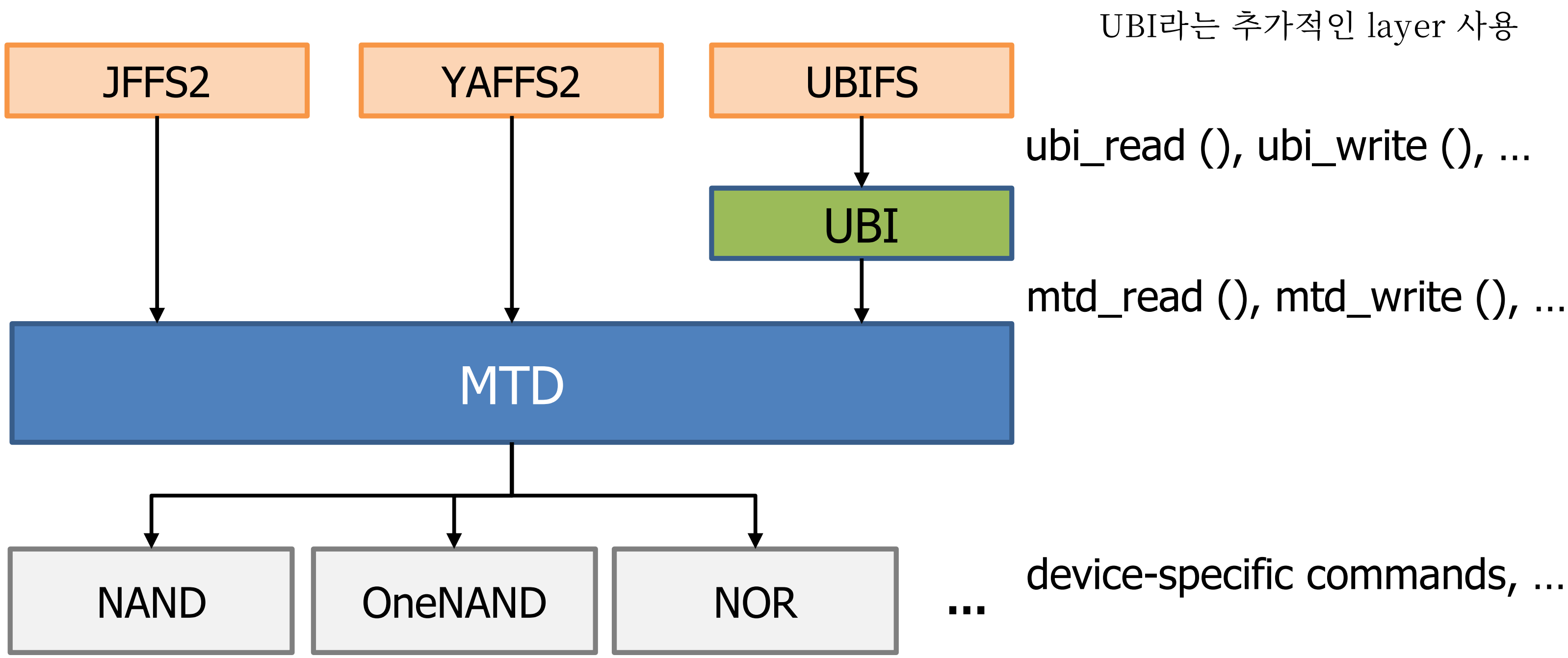
- **A new flash file system developed by Nokia**
  - Considered as the next generation of JFFS2

잘 만들었는데
FFS를 안 쓰게 되면서
쓸 일이 없어졌다.

- **New features of UBIFS**
  - Scalability
    - Scale well with respect to flash size
    - Memory size and mount time do not depend on flash size
  - Fast mount
    - Do not have to scan the whole media when mounting
  - Write-back support
    - Dramatically improve the throughput of the file system in many workloads
  - …

UBI: Unsorted Block Image

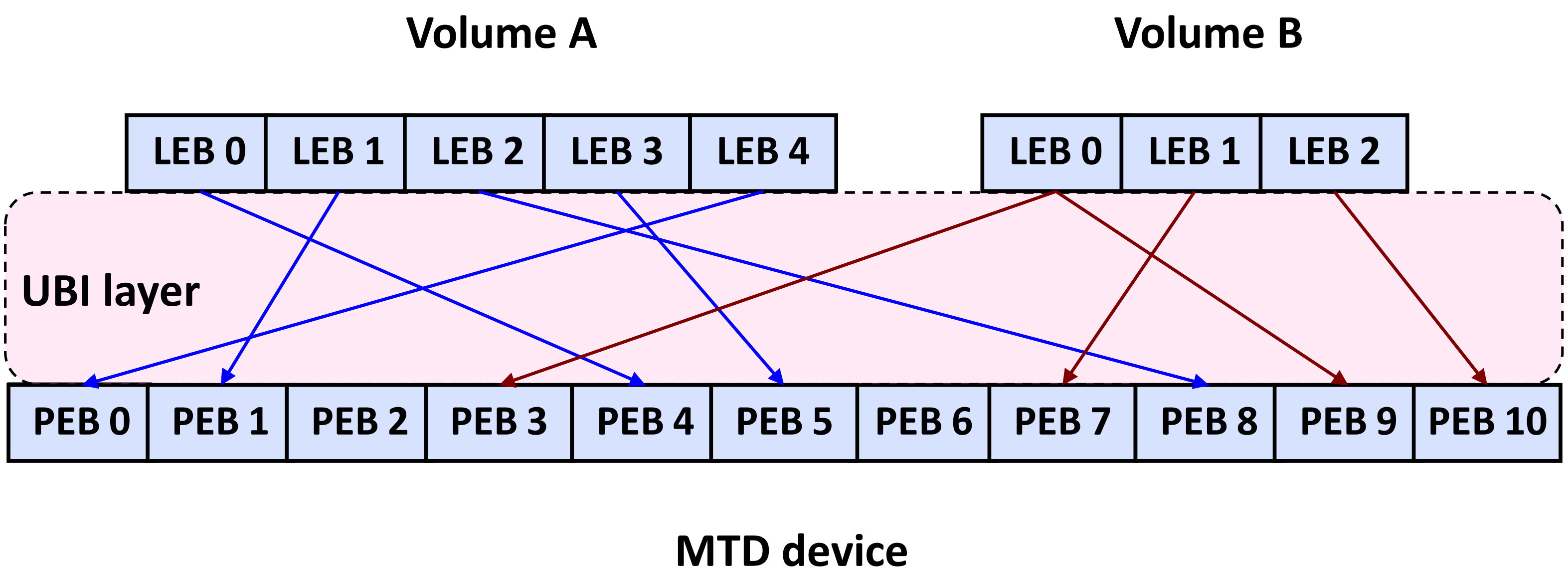
- **UBIFS runs on top of UBI volume**
  - UBI supports multiple volumes, bad block management, wear-leveling, and bit-flips error management
  - The upper level software can be simpler with UBI



How UBI works

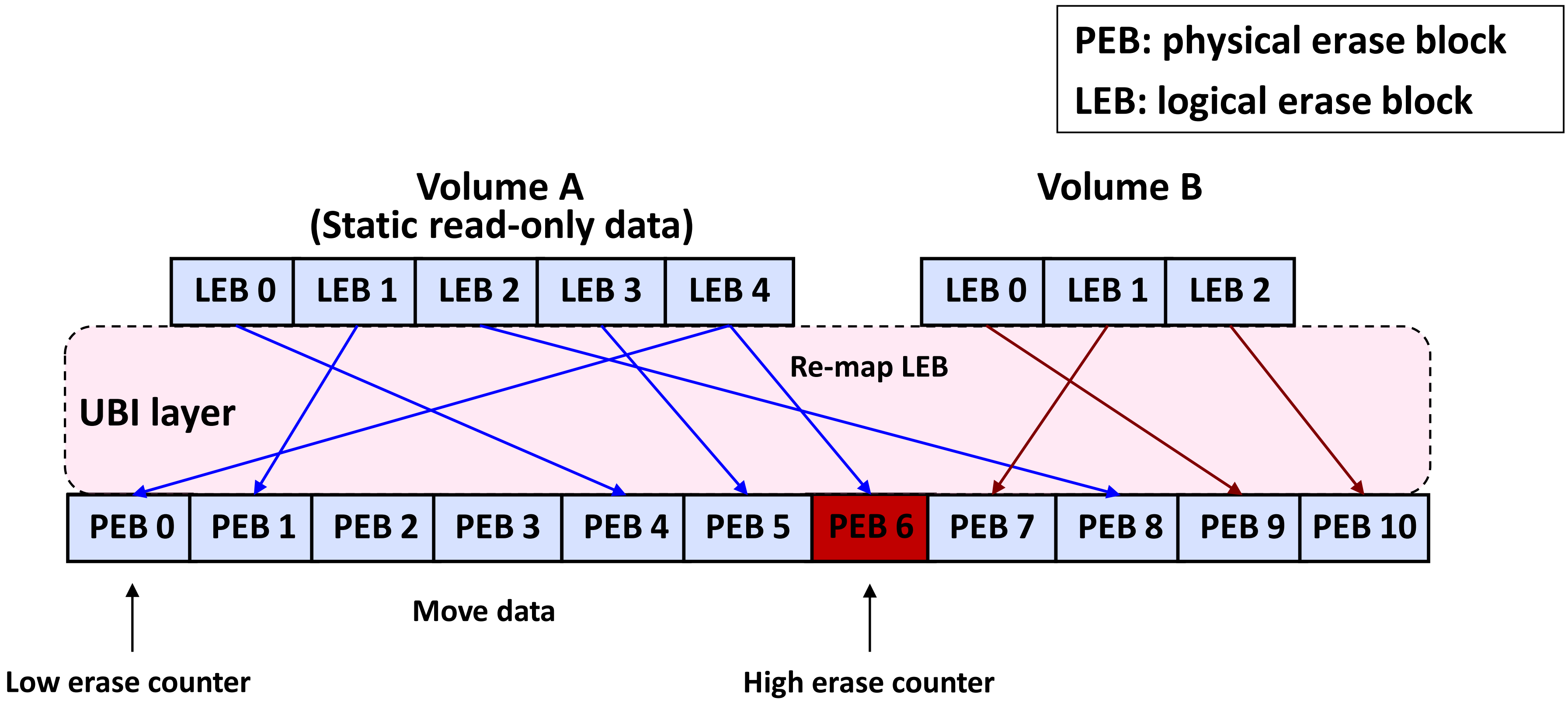
- **Logical erase blocks (LEBs) are mapped to physical erase blocks (PEBs)**
  - Any LEB can be mapped to any PEB

PEB: physical erase block
LEB: logical erase block



How UBI works

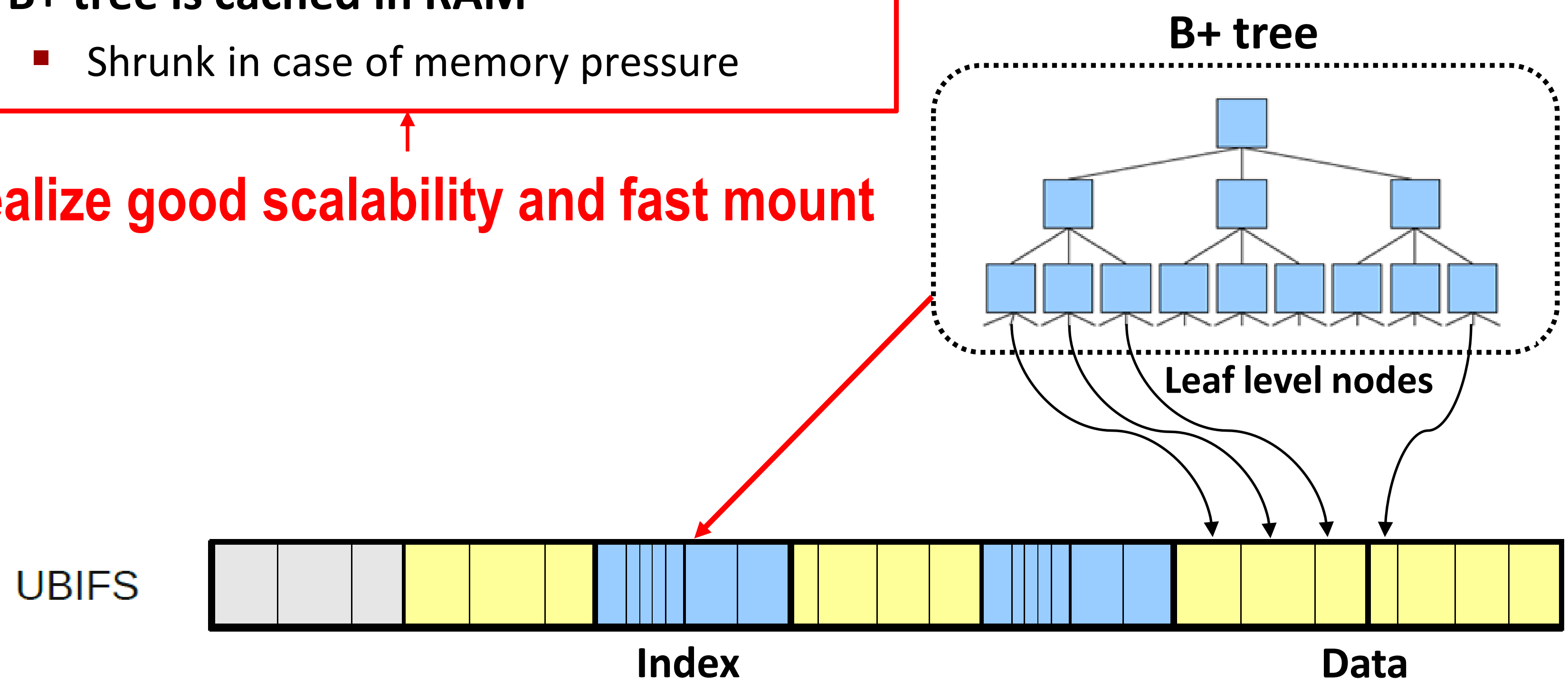
- UBI has its own wear-leveling algorithm that moves the data kept in highly erased blocks to lower one



UBIFS: Indexing with B+ Tree

- **UBIFS index is a B+ tree and is stored on NAND flash**
  - c.f., JFFS2 does not store the index on flash
- **Leaf level contains data**
- **Full scanning is not needed**
- **B+ tree is cached in RAM**
  - Shrunk in case of memory pressure

Realize good scalability and fast mount



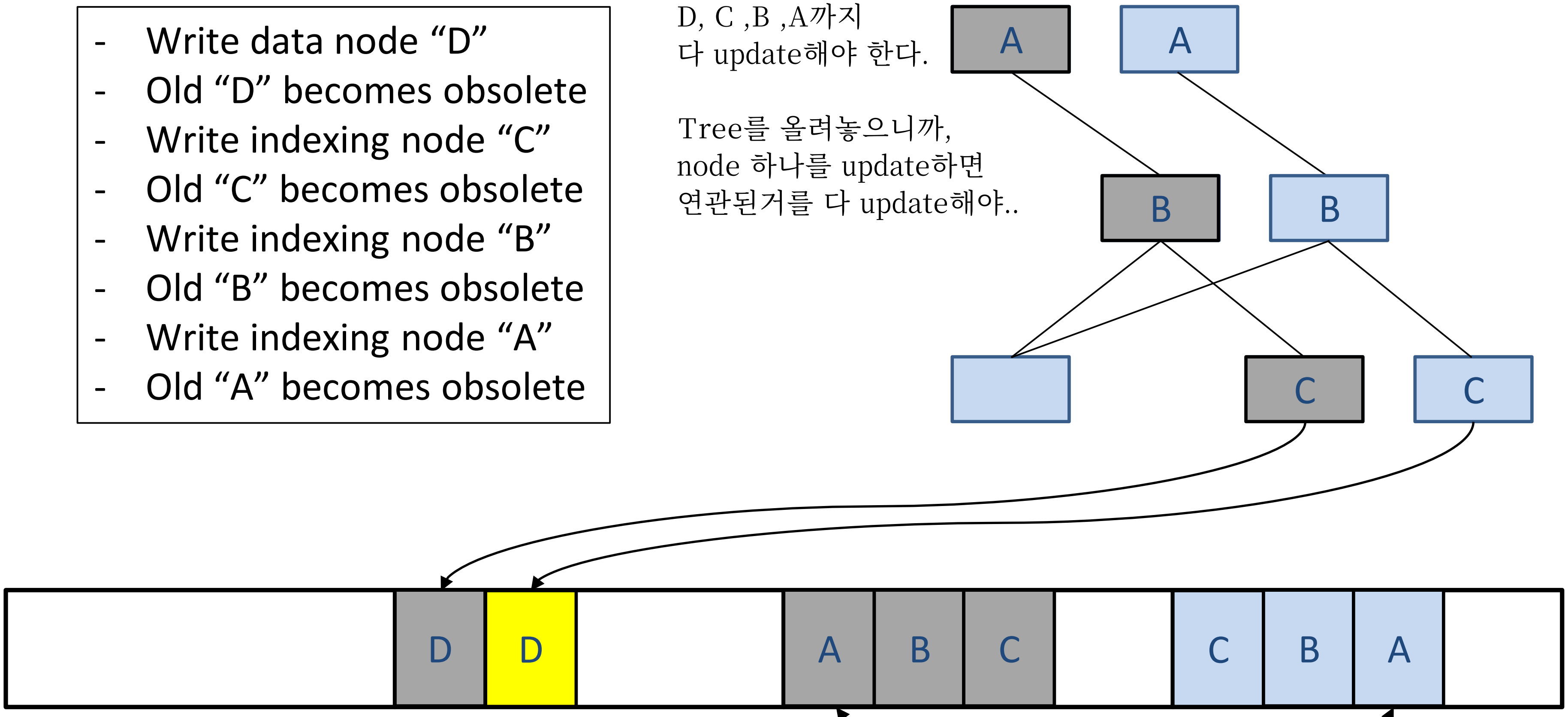
UBIFS: Wandering Tree

- How to find the root of the tree?

- Write data node "D"
- Old "D" becomes obsolete
- Write indexing node "C"
- Old "C" becomes obsolete
- Write indexing node "B"
- Old "B" becomes obsolete
- Write indexing node "A"
- Old "A" becomes obsolete

data node D 하나를 update하면
D, C ,B ,A까지
다 update해야 한다.

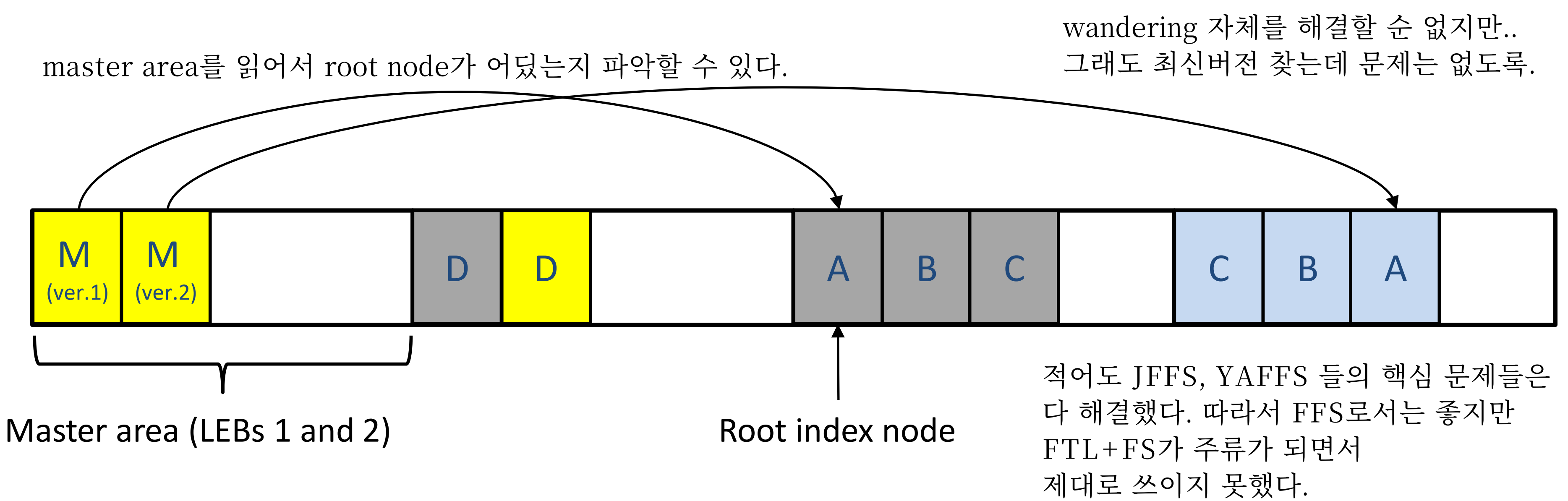
Tree를 올려놓으니까,
node 하나를 update하면
연관된거를 다 update해야..



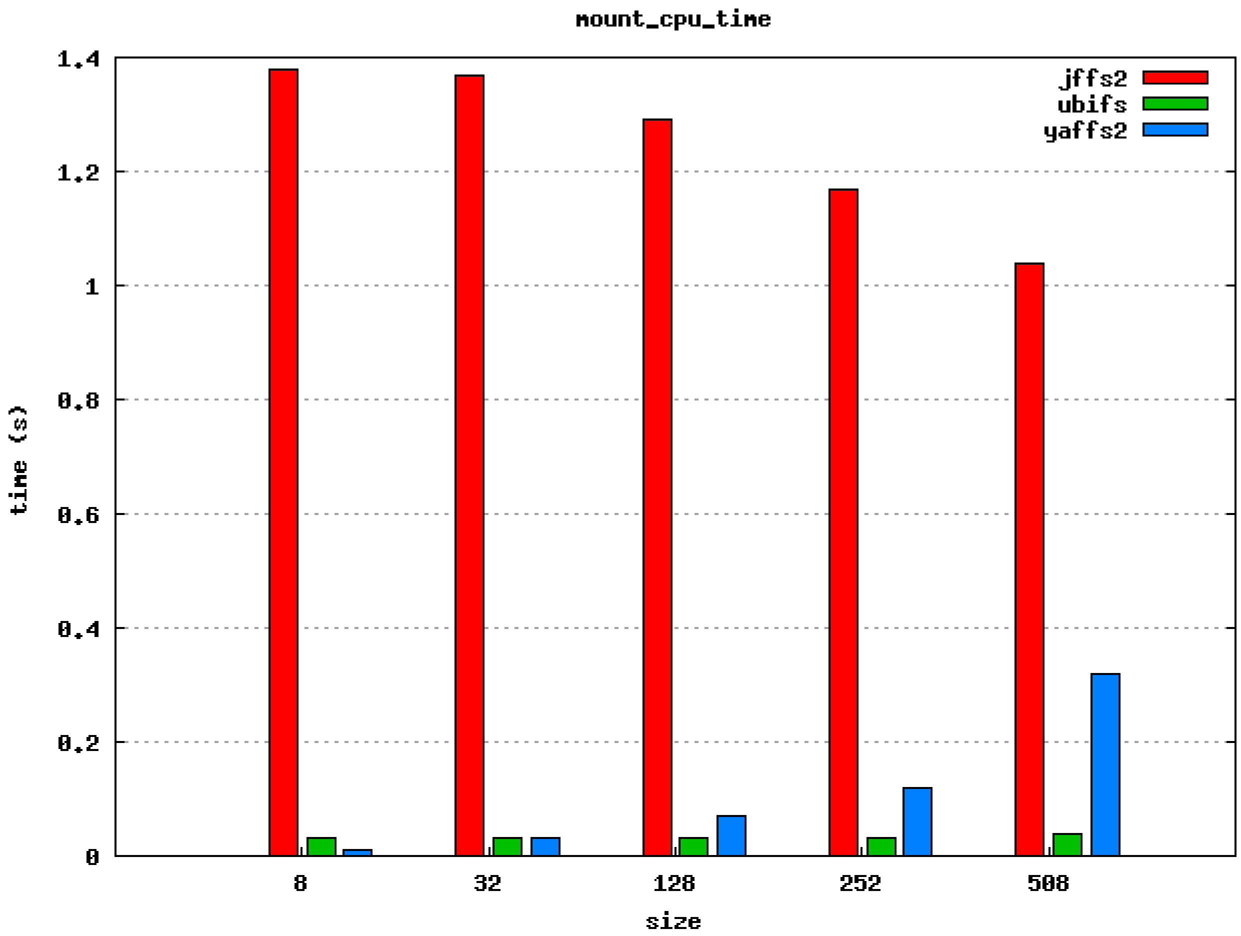
The position of the tree in flash is changed

UBIFS: Master Area

- LEBs 1/2 are reserved for *a master area* pointing to *a root index node*
  - The master area can be quickly found on mount since its location is fixed (LEBs 1 and 2)
  - Using the root index node, B+ tree can be quickly constructed
- A mater area could have multiple nodes
  - A valid master node is found by scanning master area



Mount Time Comparison



Summary

- **JFFS2: Journaling Flash File System version 2**
  - Commonly used for low volume flash devices
  - Compression is supported
  - Long mount time & High memory consumption
- **YAFFS2: Yet Another Flash File System version 2**
  - Fast mount time with check-pointing
- **UBIFS: Unsorted Block Image File System**
  - Fast mount time and low memory consumption by adopting B+ tree indexing

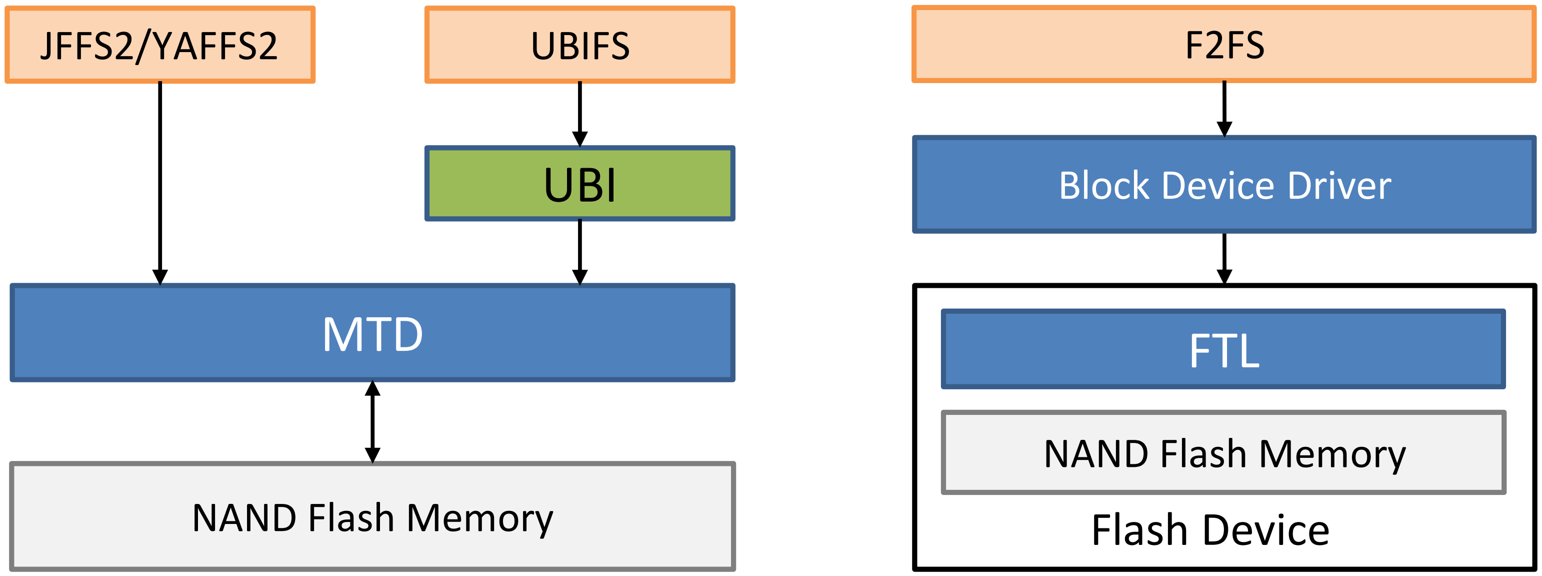
Outline

- Traditional Flash File Systems
- **SSD-Friendly Flash File Systems**
  - F2FS: Flash-friendly File System
- Reference

F2FS: Flash-friendly File System

기존의 FFS와 다르게, LFS임에도 block i/o subsystem 기반으로 되어있다.

- **Log-structured file system for FTL devices**
  - Unlike other flash file systems, it runs atop FTL-based flash storage and is optimized for it
  - Exploit system-level information for better performance and reliability (e.g., better hot-cold separation, background GC, …)



Traditional flash file system

F2FS

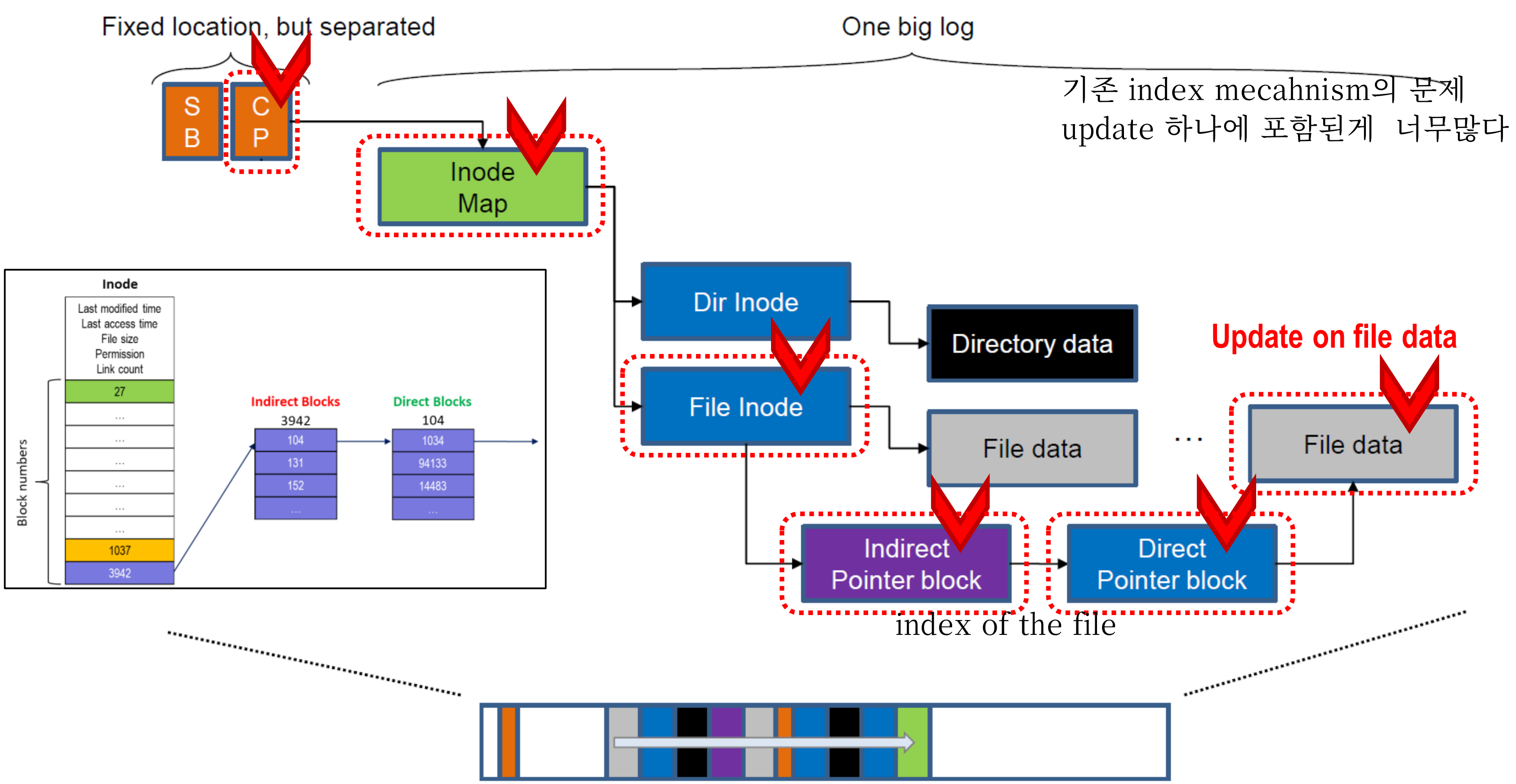
Design Concept of F2FS

- **Designed to take advantage of both of two approaches**
  - Exploit high-level system information
  - Flash-aware storage management
  - Handle new NAND flash without redesign

	File System + FTL	Flash File System
Method	- Access a flash device via FTL	- Access a flash device directly
Pros	- High interoperability - No difficulties in managing recent NAND flash with new constraints	- High-level optimization with system-level information - Flash-aware storage management
Cons	- Lack of system-level information - Flash-unaware storage management	- Low interoperability - Must be redesigned to handle new constraints

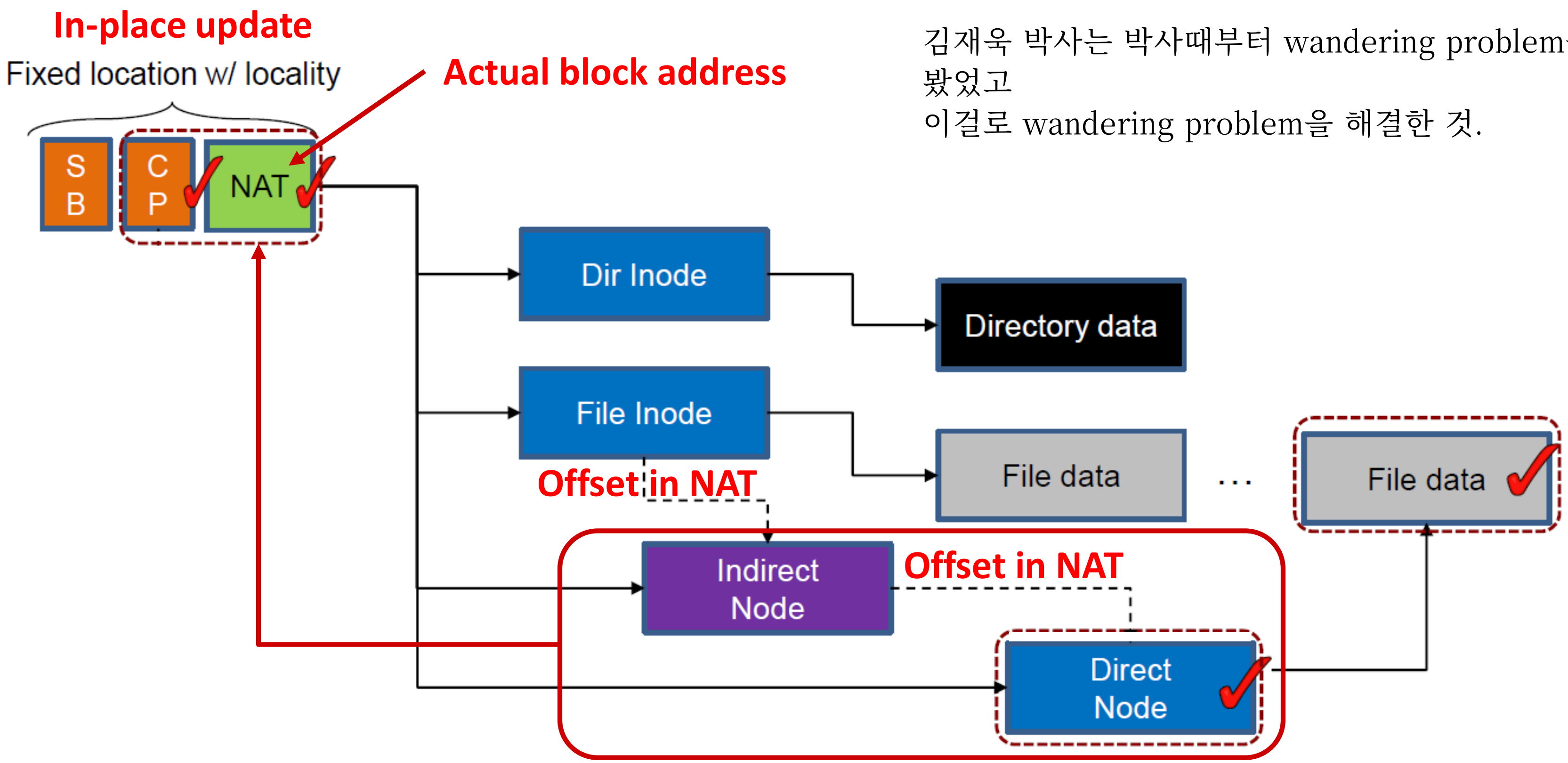
Index Structure of LFS

- **LFS and UBIFS suffer from the wandering tree problem**
  - Update on a file causes several extra writes



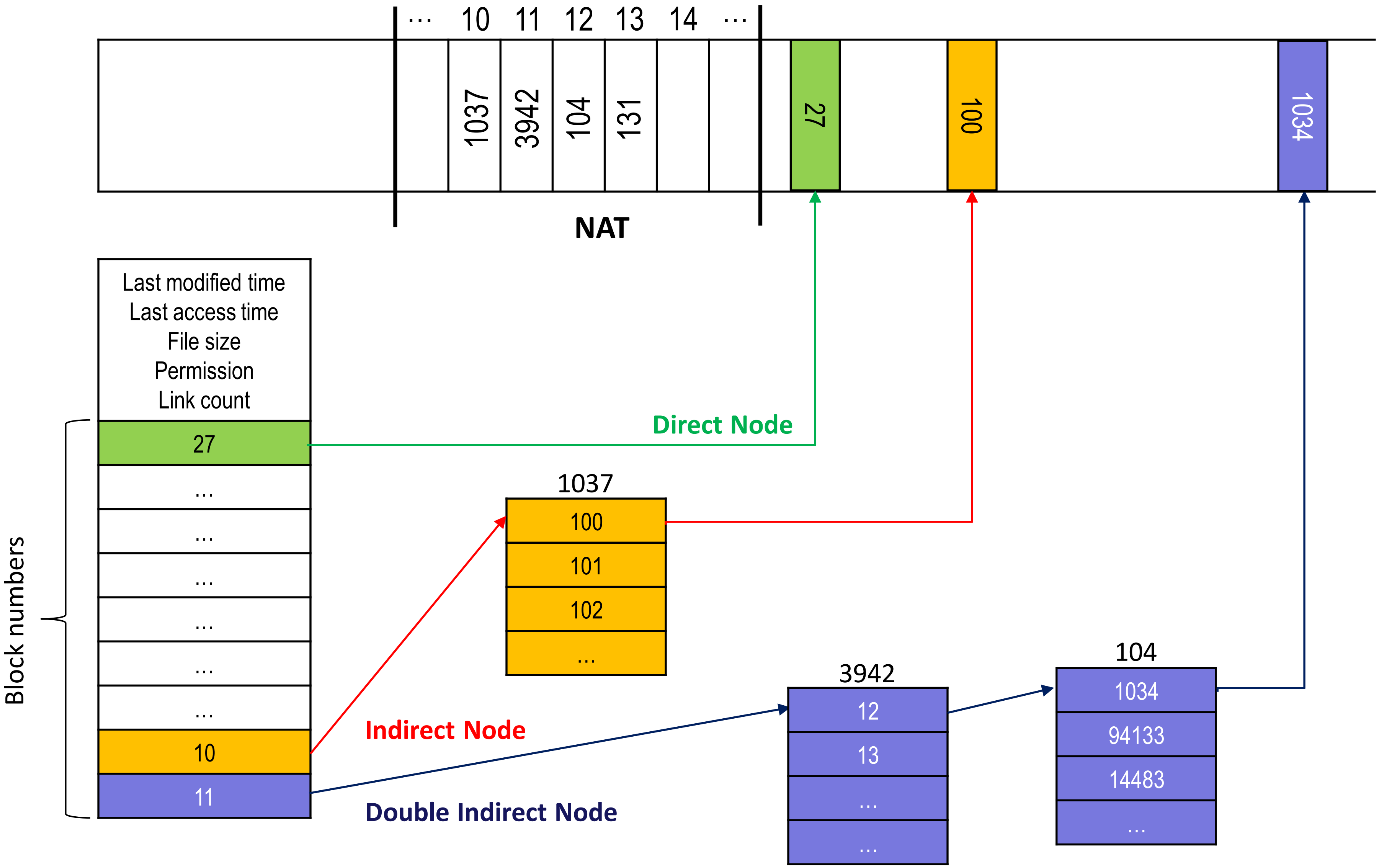
Index Structure of F2FS

- Introduce Node Address Table (NAT) containing the locations of all the node blocks, including indirect and direct nodes
- NAT allows us to eliminate the wandering tree problem

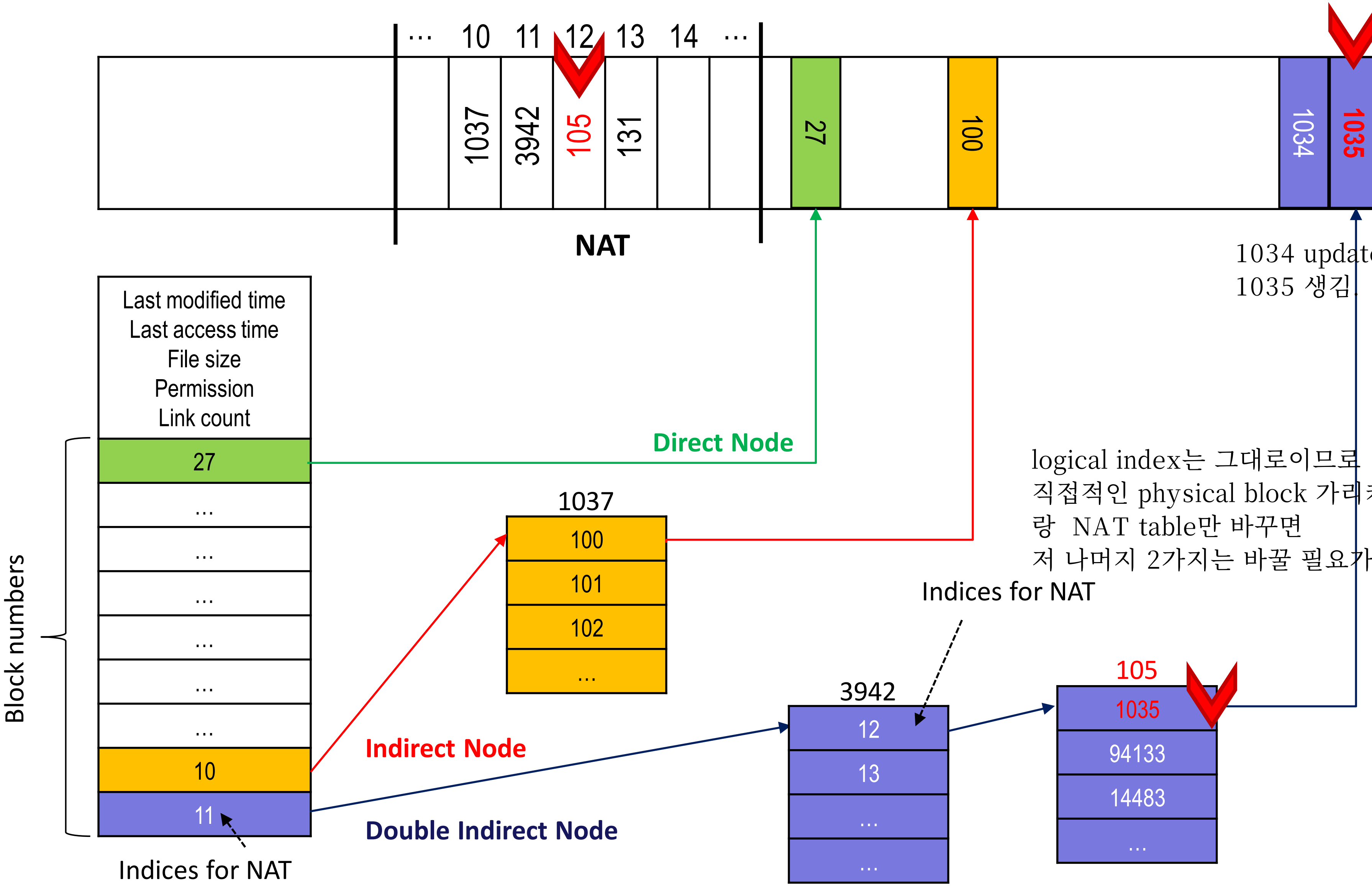


김재욱 박사는 박사때부터 wandering problem을 봤었고
이걸로 wandering problem을 해결한 것.

Index Structure of F2FS (Cont)



Index Structure of F2FS (Cont)

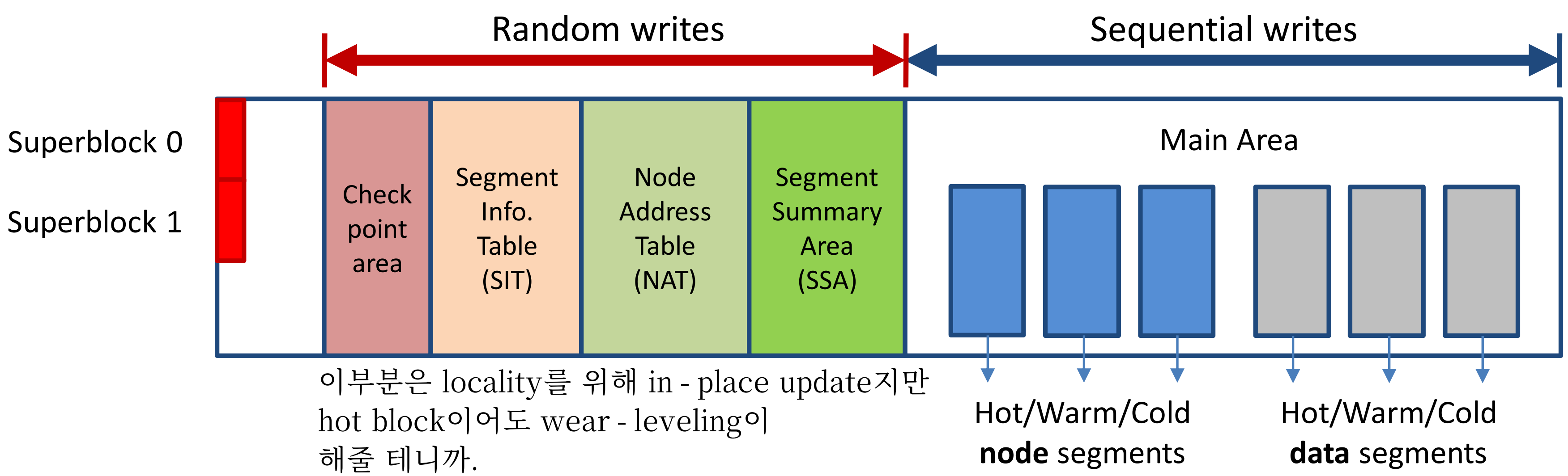


1034 update해서
1035 생김.

logical index는 그대로이므로
직접적인 physical block 가리키는거
랑 NAT table만 바꾸면
저 나머지 2가지는 바꿀 필요가 없다.

Logical Storage Layout

- **File-system's metadata is located together for locality**
  - Use an "in-place update" strategy for metadata
- **Files are written sequentially for performance**
  - Use an "out-of-place-update" strategy to exploit high throughput of multiple NAND chips
  - Six active logs for static hot and cold data separation



Cleaning Process

- **Background cleaning process**
  - A kernel thread doing the cleaning job periodically at idle time

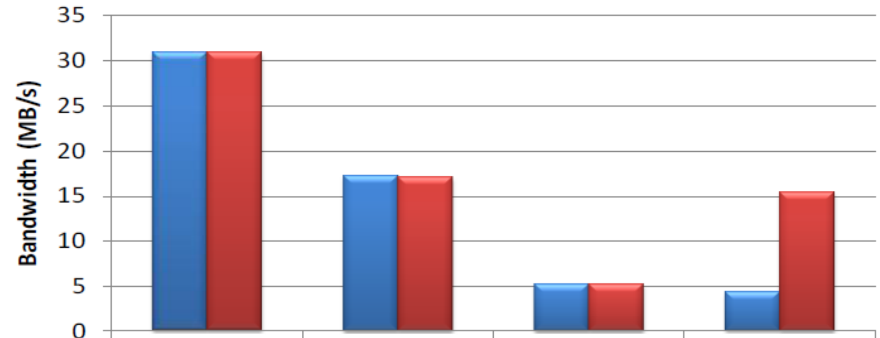
- **Victim selection policies**
  - Greedy algorithm for foreground cleaning job
    - Reduce the amount of data moved for cleaning
  - Cost-benefit algorithm for background cleaning job
    - Reclaim obsolete space in a file system
      - Improve the lifetime of a storage device
      - Improve the overall I/O performance

급할 때는 greedy로 GC,
여유로울 때는 cost - benefit으로.

F2FS Performance

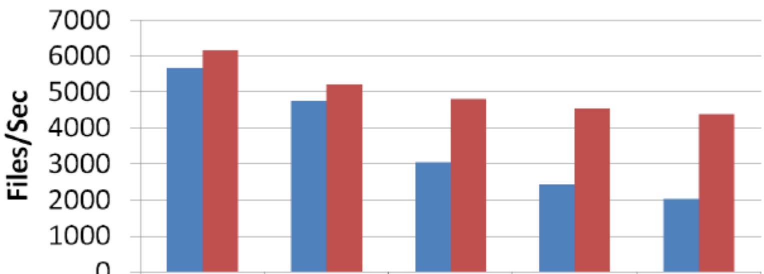
[System Specification]

CPU	ARM Coretex-A9 1.2GHz
DRAM	1GB
Storage	Samsung eMMC 64GB
Kernel	***Linux 3.3***
Partition Size	***12 GB***



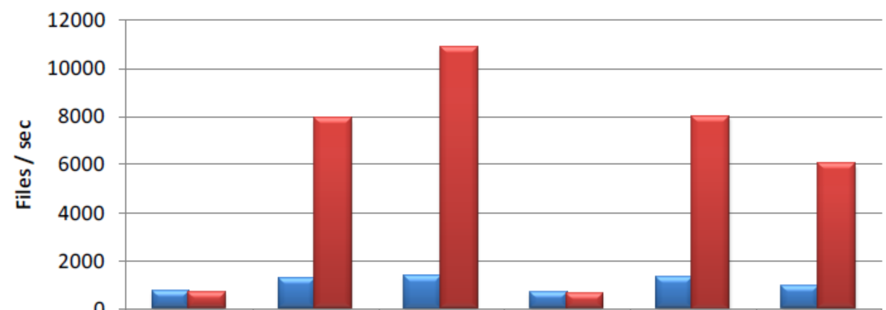
	seq. Read	seq. Write	rand. Read	rand. Write
EXT4	30.753	17.066	5.06	4.15
F2FS	30.71	16.906	5.073	15.204

[iozone]



	5000	10000	15000	20000	25000
ext4	5675.9	4741.6	3065	2439.5	2038.6
f2fs	6167.3	5207.9	4815.9	4544.4	4382.9

[fs_mark]



	seq.create	seq.stat	seq.delete	rand.create	rand.stat	rand.delete
EXT4	692	1238	1370	663	1250	915
F2FS	631	7871	10832	620	7962	5992

[bonnie++]

End of Chapter 7