F2FS: A New File System for Flash Storage

Changman Lee, Dongho Sim, Joo-Young Hwang, and Sangyeun Cho

S/W Development Team Memory Business
Samsung Electronics Co., Ltd.

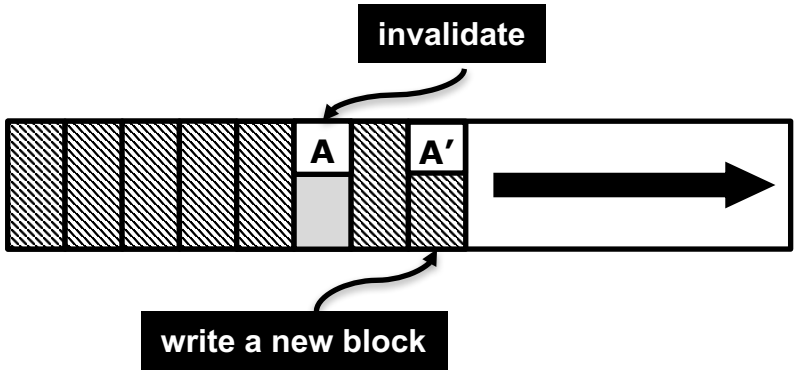
Presenter: Jonggyu Park

Contents

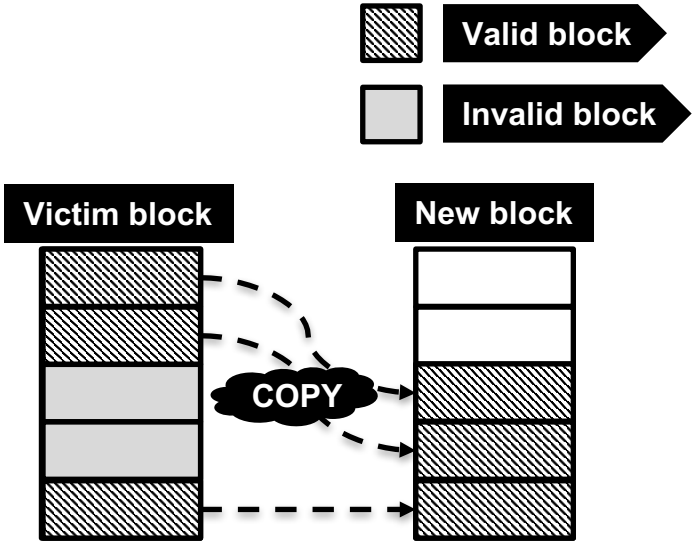
Why LFS is good for Flash?

- **Flash memory**
  - **Out-place update (No in-place update)**
  - **Random write is harmful (increasing GC costs)**
  - **Sequential I/O is faster than Random I/O**

- **Log-structured File system**
  - **Out-place update**
  - **Mostly sequential write**



<Updating 'A' block on LFS>



<Garbage Collection in Flash Memory>

Drawbacks of Conventional LFS

- **HDD-optimized Layout**
- **Wandering Tree Problem (index structure)**
- **No data classification**
- **High cleaning costs under high utilization**
- **High ‘fsync’ overhead (checkpoint per a single fsync)**

F2FS (Flash-Friendly File System)

- **HDD-optimized Layout**
  → Flash-aware Layout

- **Wandering Tree Problem (index structure)**
  → Efficient index structure

- **No data classification**
  → Multi-head logging and data hot/cold separation

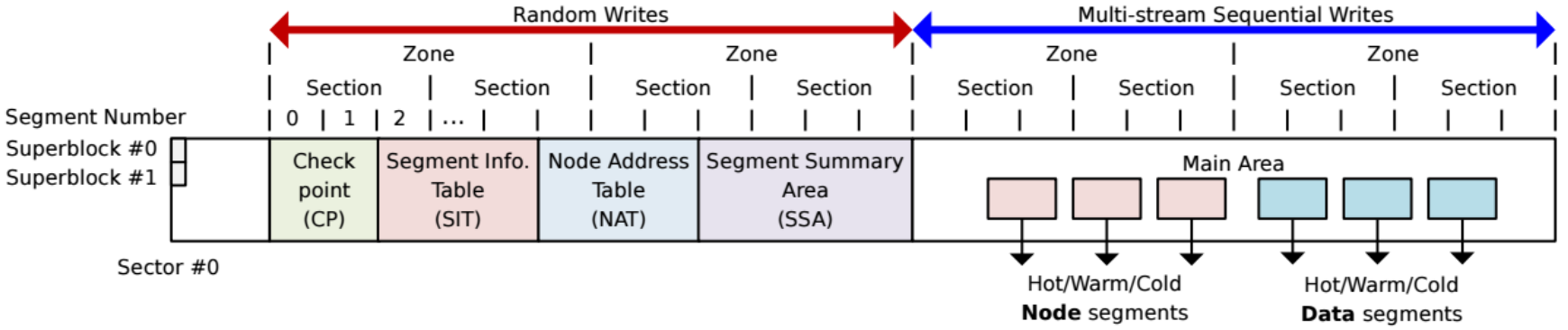
- **High cleaning costs under high utilization**
  → Adaptive logging

- **High 'fsync' overhead (checkpoint per a single fsync)**
  → fsync acceleration

Flash Friendly On-disk Layout

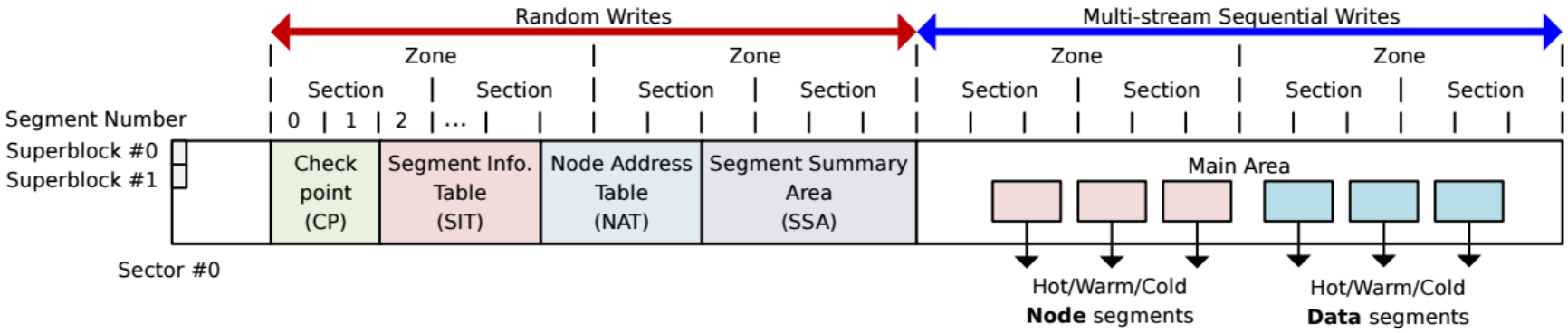
- **Flash-aware on disk layout**
  - **FS metadata are in the random write zone**
  - **Main area is aligned to the zone size**
  - **Cleaning is performed in a unit of section (FTL's GC unit)**

Block: 4KB
Segment: 2MB
Section: n segments
Zone: m sections



Flash Friendly On-disk Layout

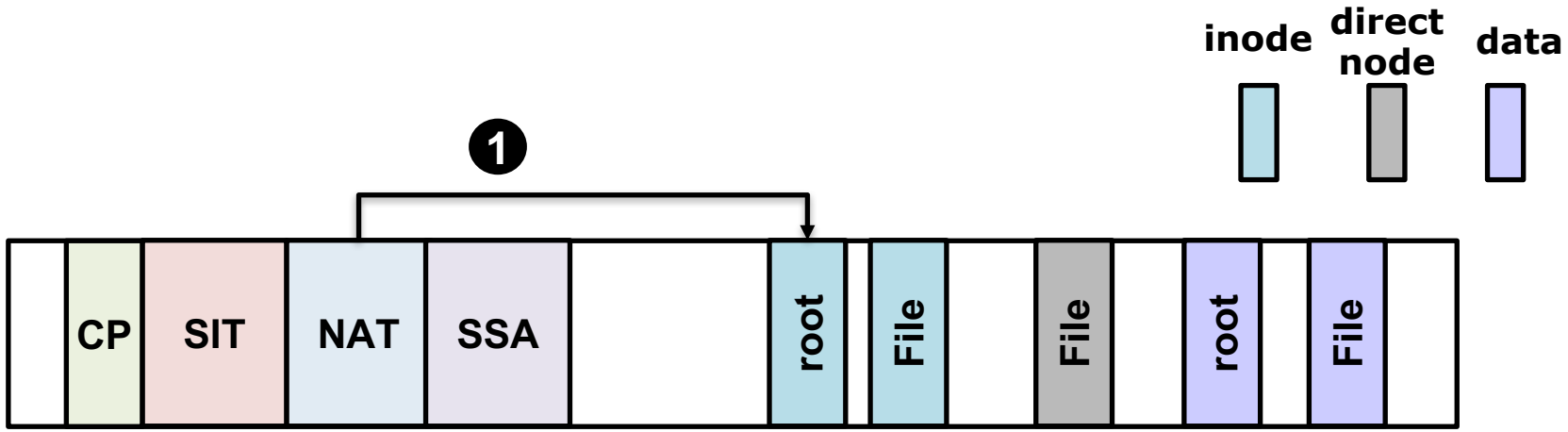
Type	Description
CP	File system info., bitmaps for valid NAT/SIT sets, and summary of current active segments
SIT	Segment info. such as valid block count and bitmap for the validity of all the blocks
NAT	Block address table for all the node blocks stored in the main area
SSA	Summary entries which contains the owner info. Of all the data and node blocks



How to read data on F2FS

Reading /file

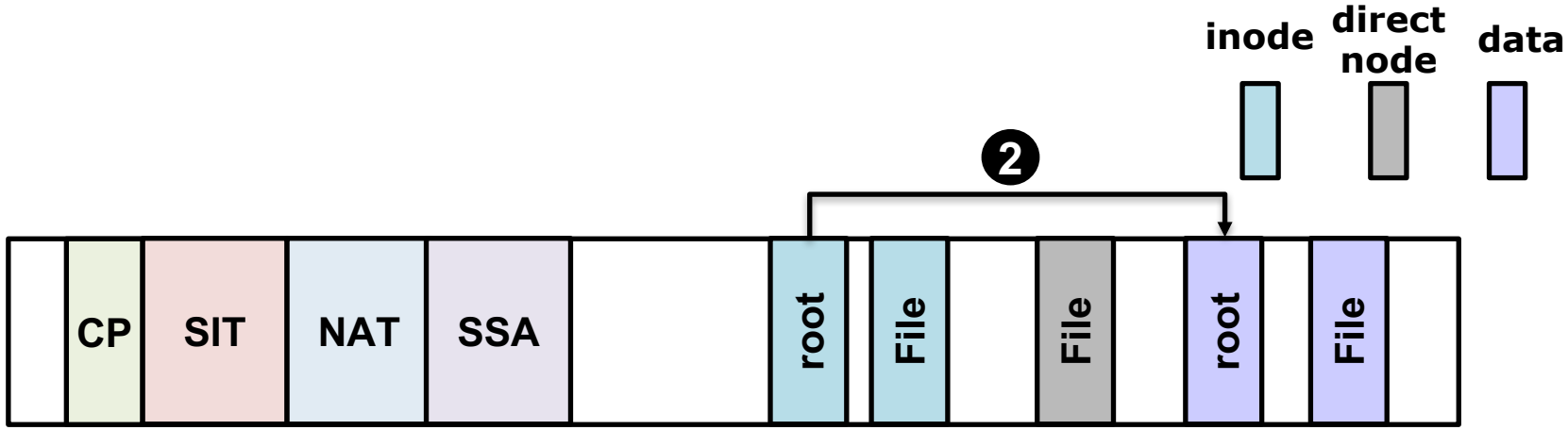
1) Obtain the 'root' inode through NAT



How to read data on F2FS

Reading /file

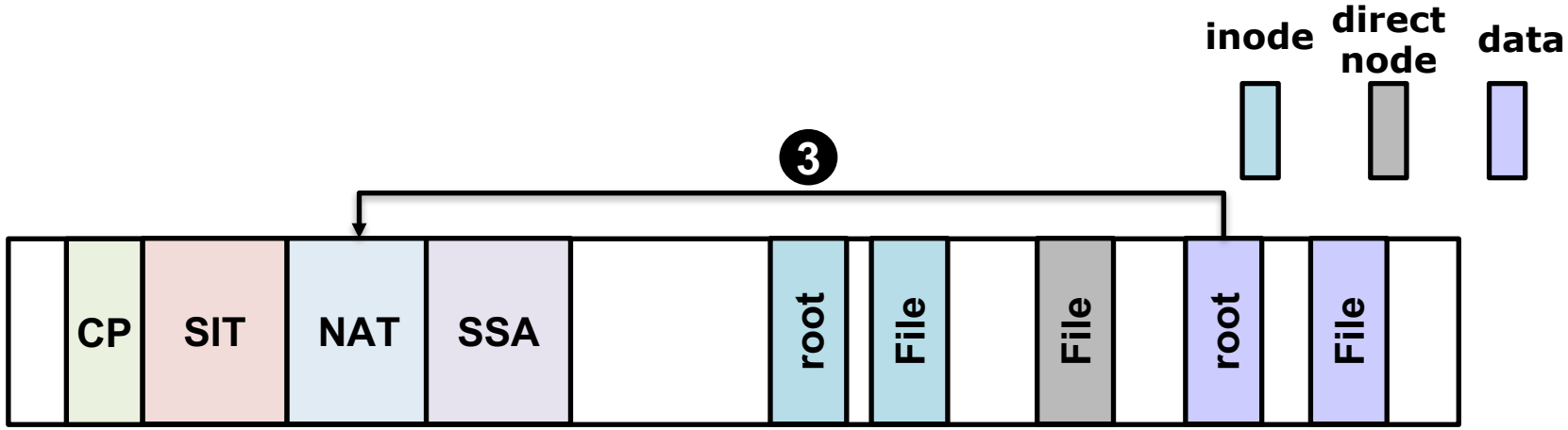
1) Obtain the 'root' inode through NAT
2) Search a directory entry named 'file' from its data block



How to read data on F2FS

Reading /file

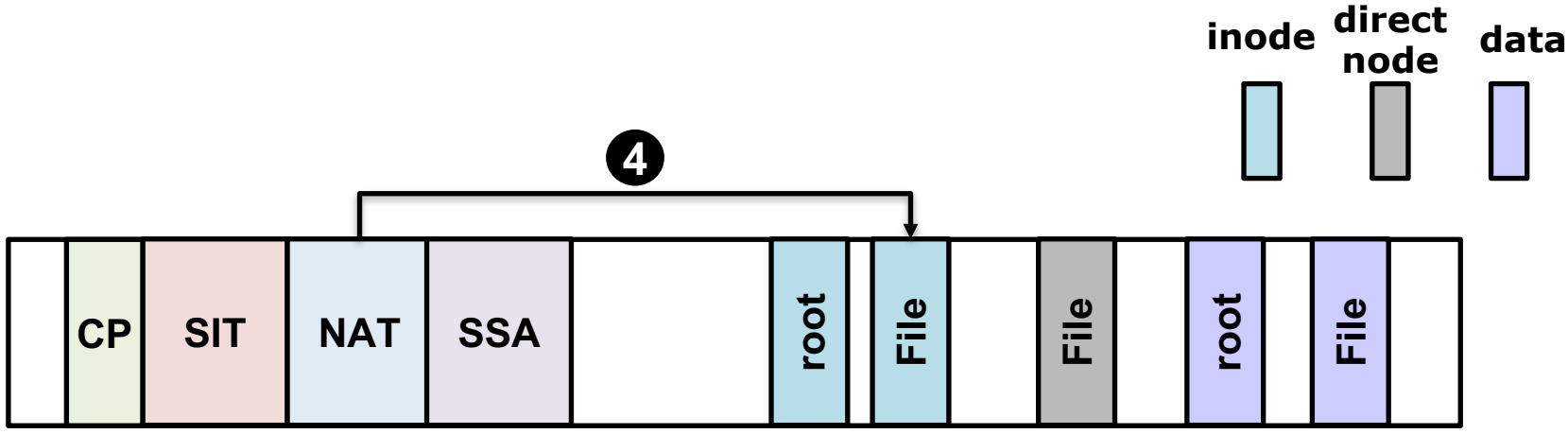
1) Obtain the 'root' inode through NAT
2) Search a directory entry named 'file' from its data block
3) Translate the inode number to the address through NAT



How to read data on F2FS

Reading /file

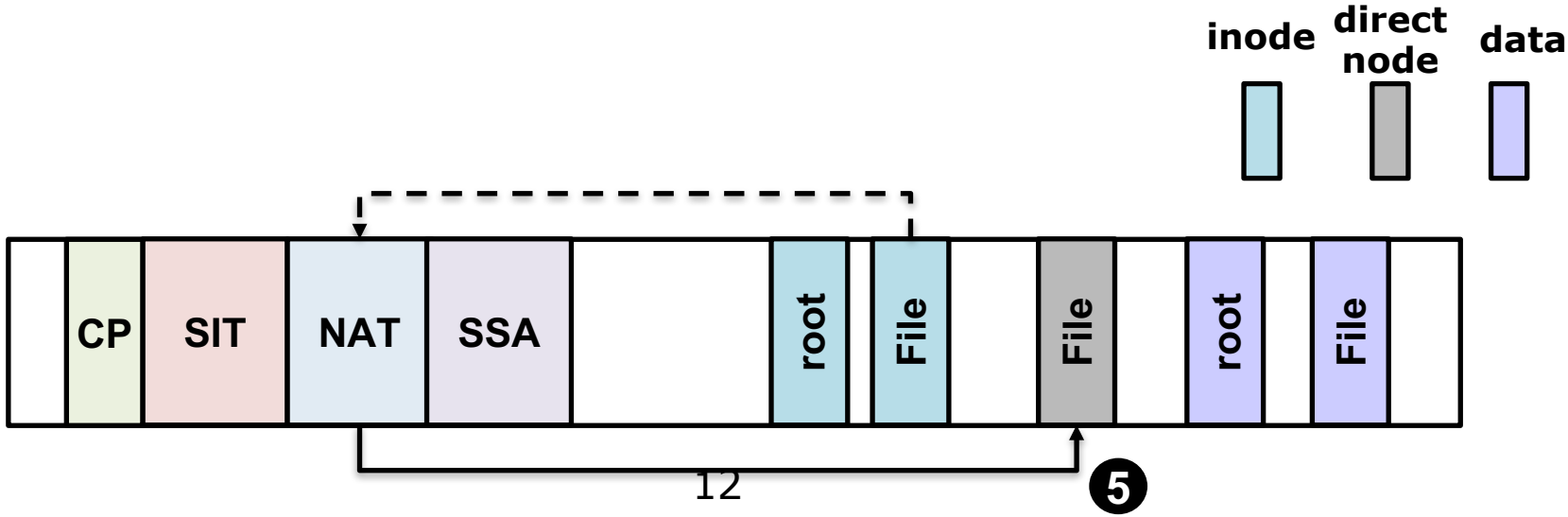
1) Obtain the 'root' inode through NAT
2) Search a directory entry named 'file' from its data block
3) Translate the inode number to the address through NAT
4) Obtain the 'file' inode by reading the corresponding block



How to read data on F2FS

Reading /file

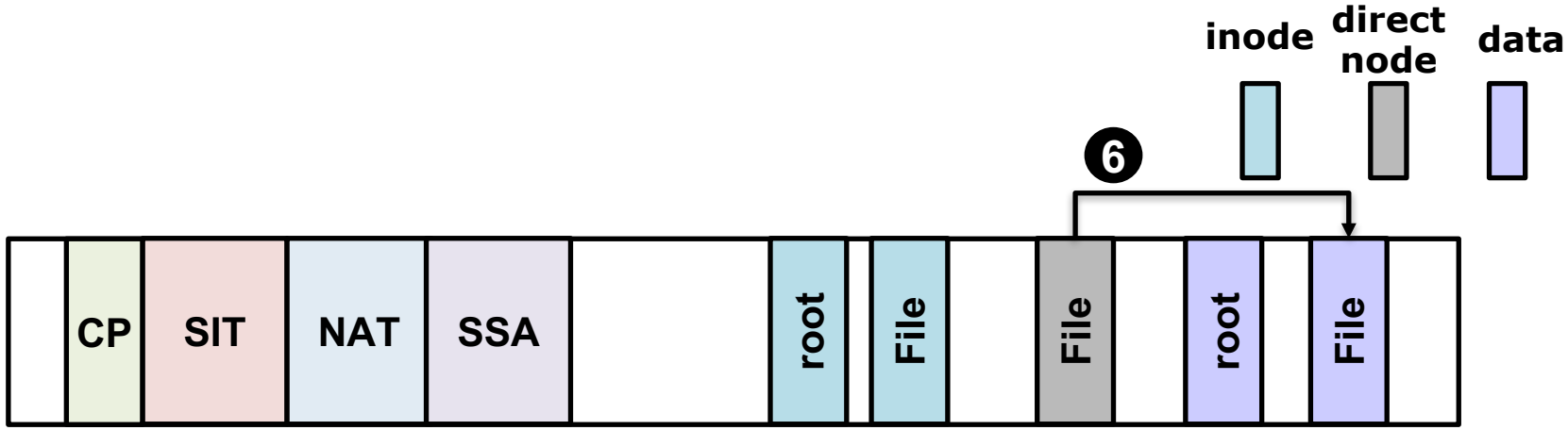
1) Obtain the 'root' inode through NAT
2) Search a directory entry named 'file' from its data block
3) Translate the inode number to the address through NAT
4) Obtain the 'file' inode by reading the corresponding block
5) Obtain a direct node block address translated by NAT



How to read data on F2FS

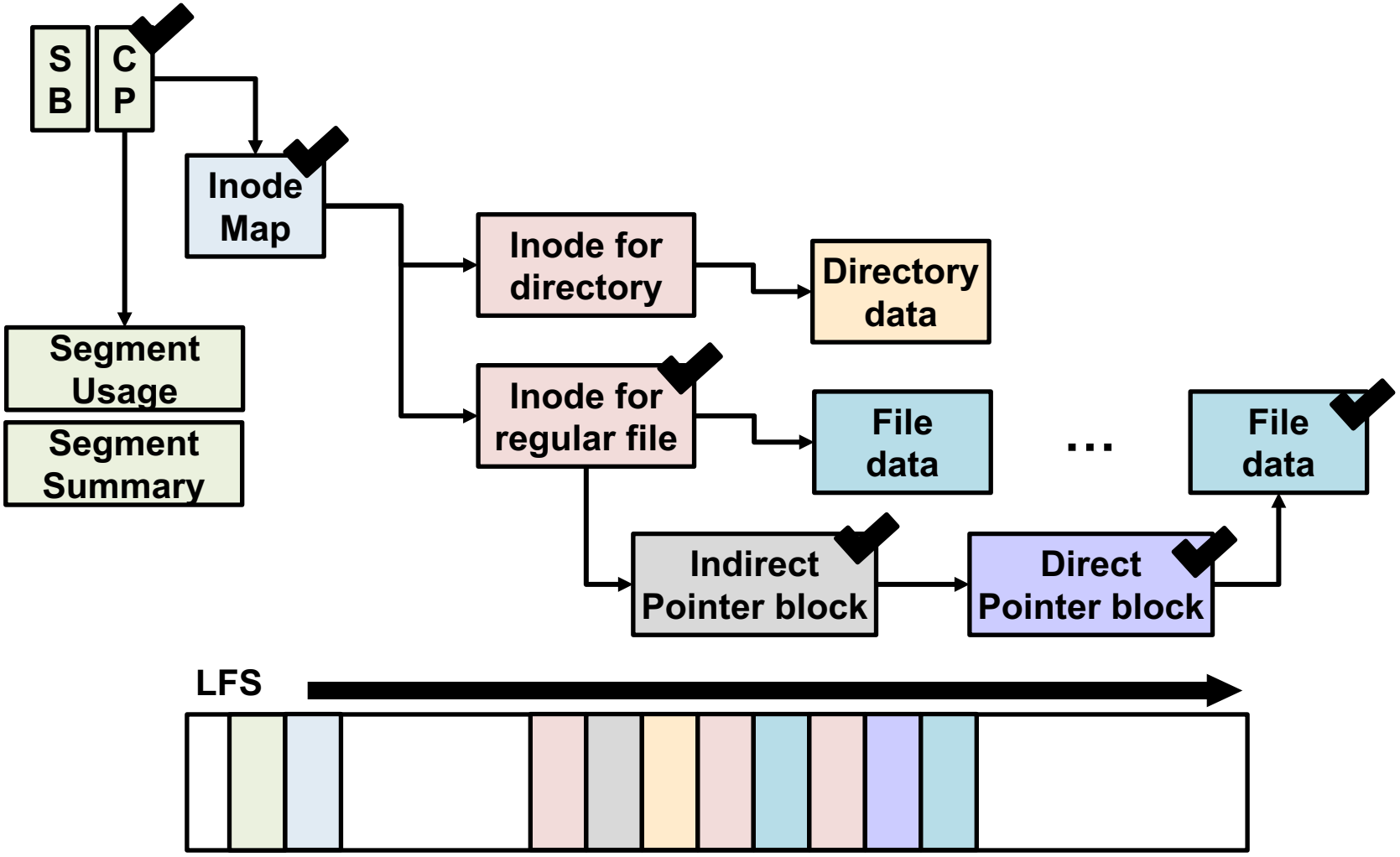
Reading /file

1) Obtain the 'root' inode through NAT
2) Search a directory entry named 'file' from its data block
3) Translate the inode number to the address through NAT
4) Obtain the 'file' inode by reading the corresponding block
5) Obtain a direct node block address translated by NAT
6) Access the data block using the direct node block



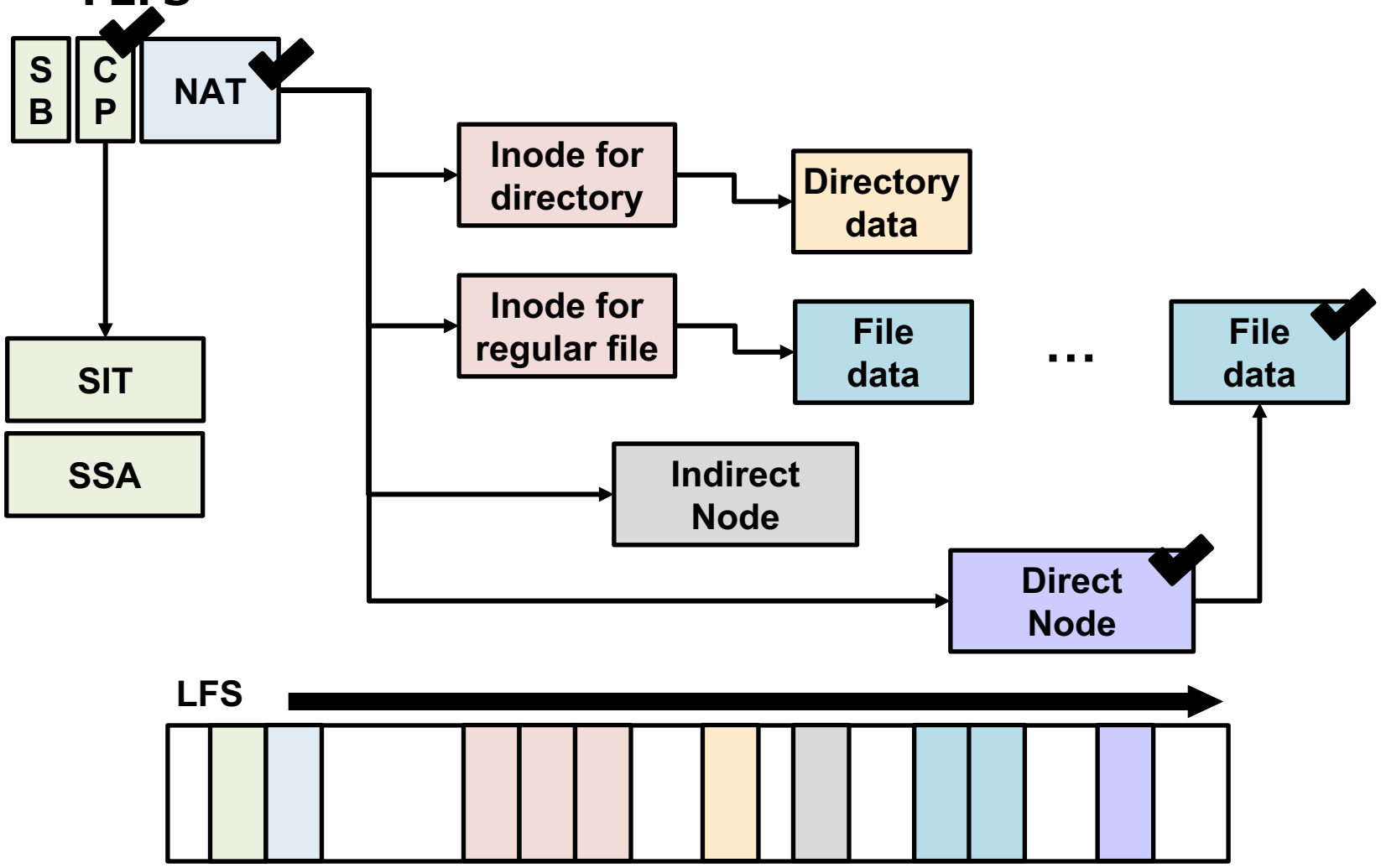
Efficient Index Structure

- **Conventional LFS**



Efficient Index Structure

- F2FS



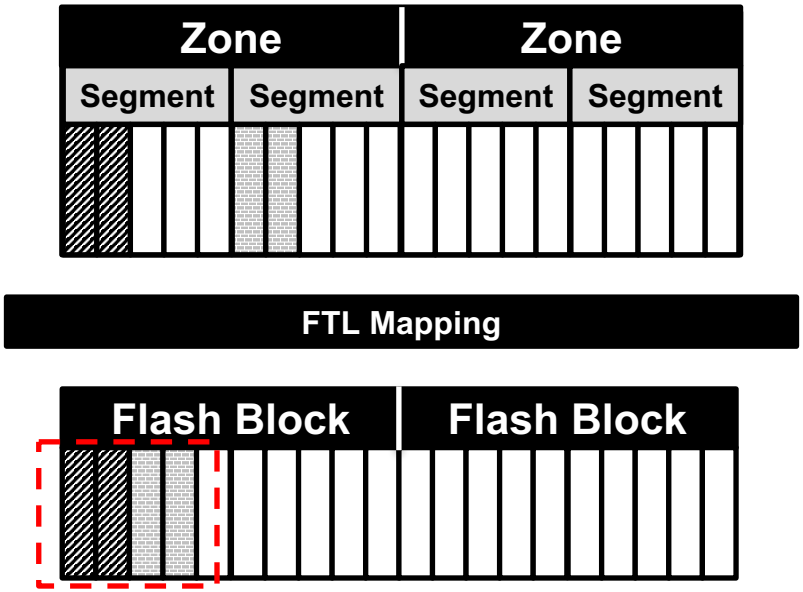
Multi-head Logging

- **Hot/Cold separation**
  - **Node > Data**
  - **Direct node > Indirect Node**
  - **Directory > Regular File**

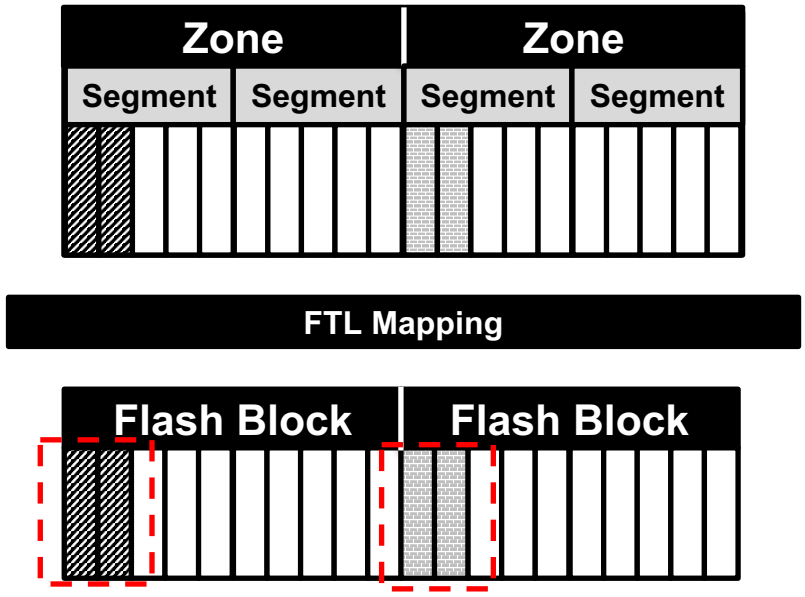
Type	Temp.	Objects
Node	Hot	Direct node blocks for directories
	Warm	Direct node blocks for regular files
	Cold	Indirect node blocks
Data	Hot	Directory entry blocks
	Warm	Data blocks made by users
	Cold	Data blocks moved by cleaning; Cold data blocks specified by users; Multimedia file data

- **Zone-aware Allocation**

<Zone-blind allocation>

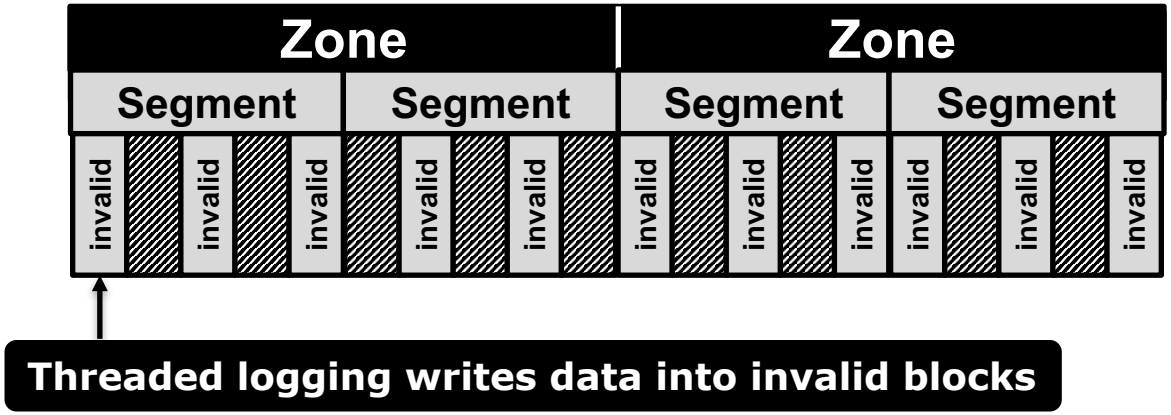


<Zone-aware allocation>



Adaptive Logging

- **To reduce cleaning costs at high utilization, F2FS utilize adaptive logging**
  - **Append logging (logging to clean segments)**
    - **Need cleaning operations if no free segments**
    - **Cleaning causes mostly rand. read and seq. write**
  - **Threaded logging (logging to dirty segments)**
    - **Reuse invalid blocks in dirty segments**
    - **No need cleaning**
    - **Cause random writes**



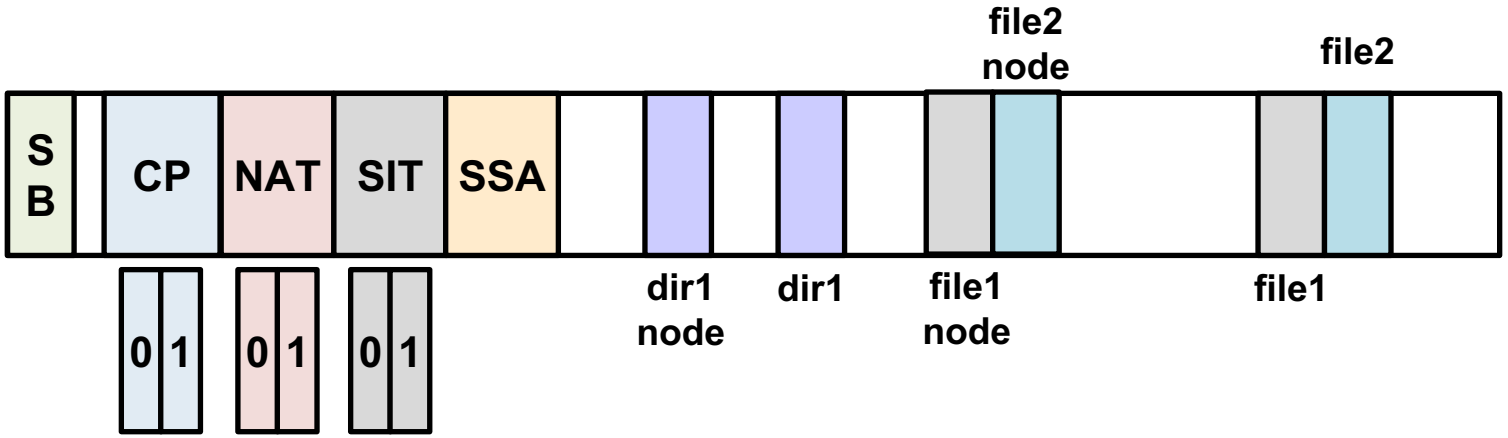
Recovery and ‘fsync’ Acceleration

- **Recovery**
  - **Checkpoint and rollback**

- **‘fsync’ Acceleration**
  - **When fsync, Direct node blocks are written with fsync mark**
  - **No need to create a checkpoint**
  - **When crash, compare fsynced blocks with old blocks**

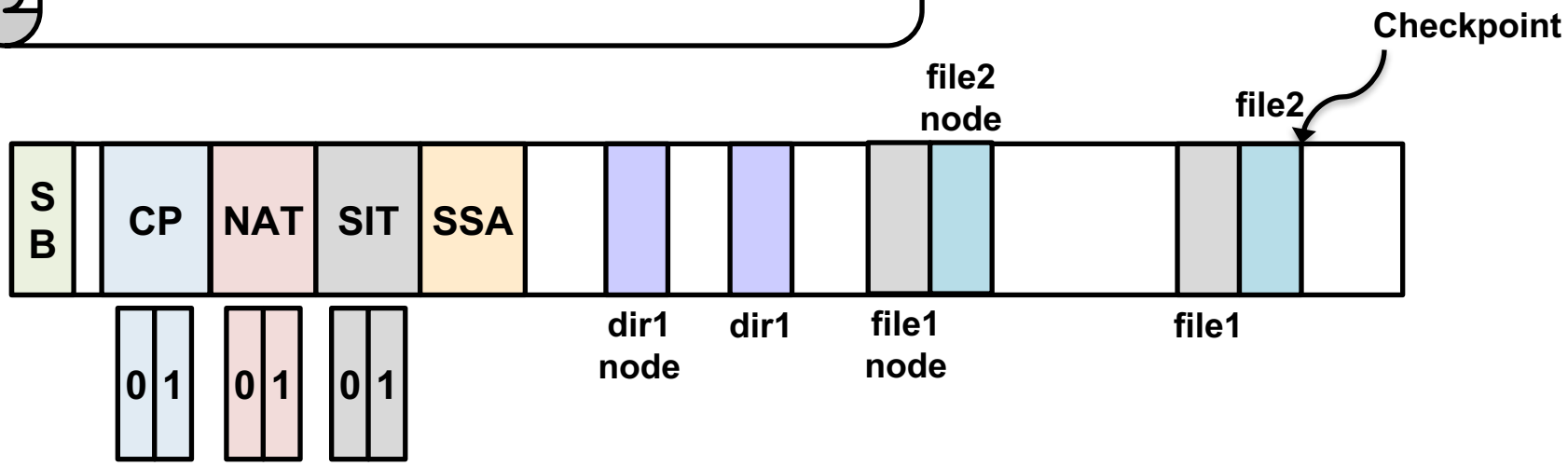
Recovery and ‘fsync’ Acceleration

1. Create dir1, file1, and file2



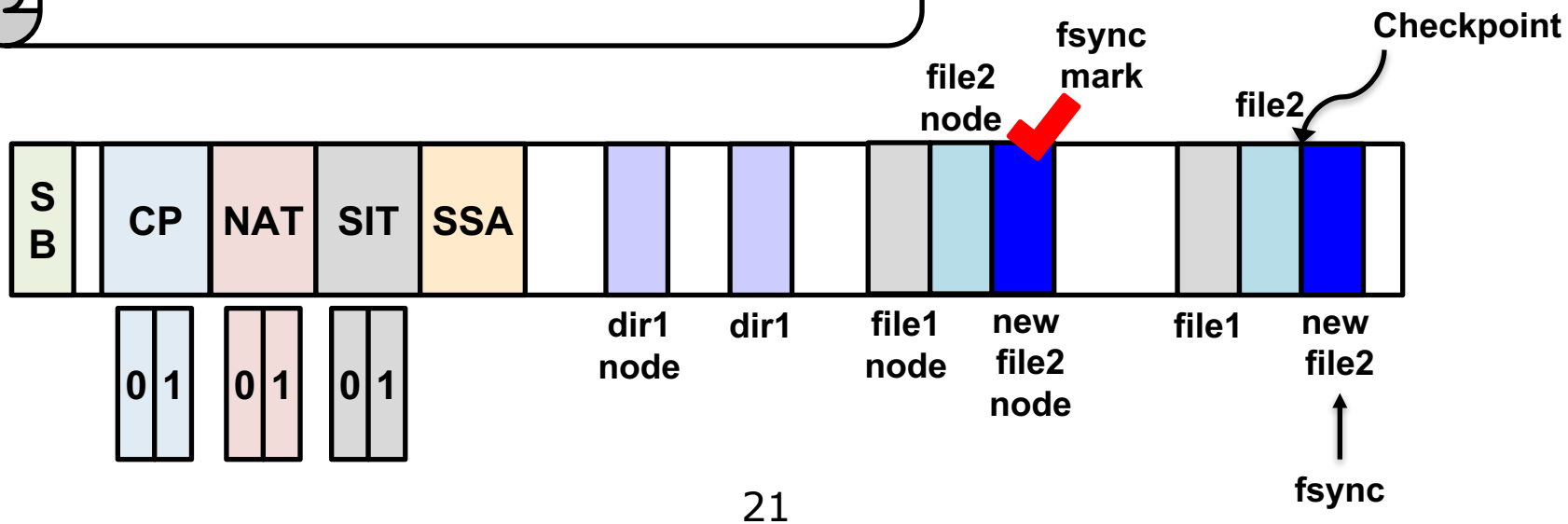
Recovery and ‘fsync’ Acceleration

1. Create dir1, file1, and file2
2. Create checkpoint



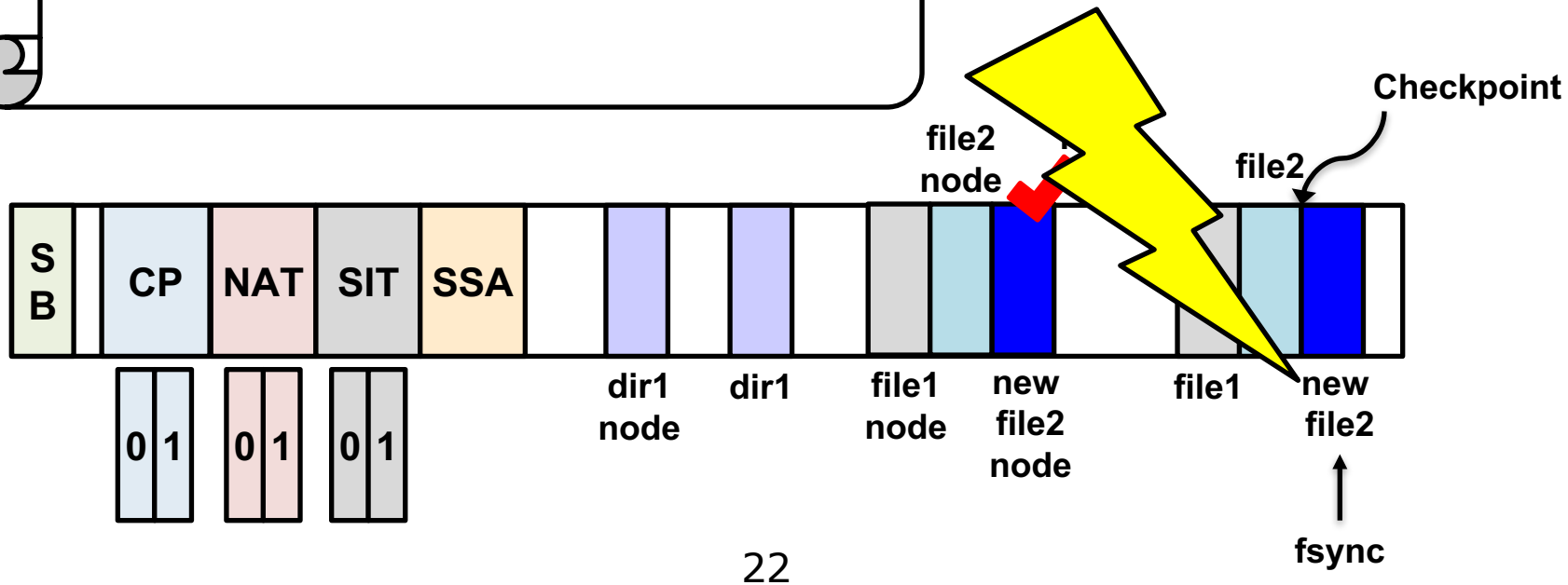
Recovery and 'fsync' Acceleration

1. Create dir1, file1, and file2
2. Create checkpoint
3. File2 update and fsync



Recovery and ‘fsync’ Acceleration

1. Create dir1, file1, and file2
2. Create checkpoint
3. File2 update and fsync
4. Sudden Power Off

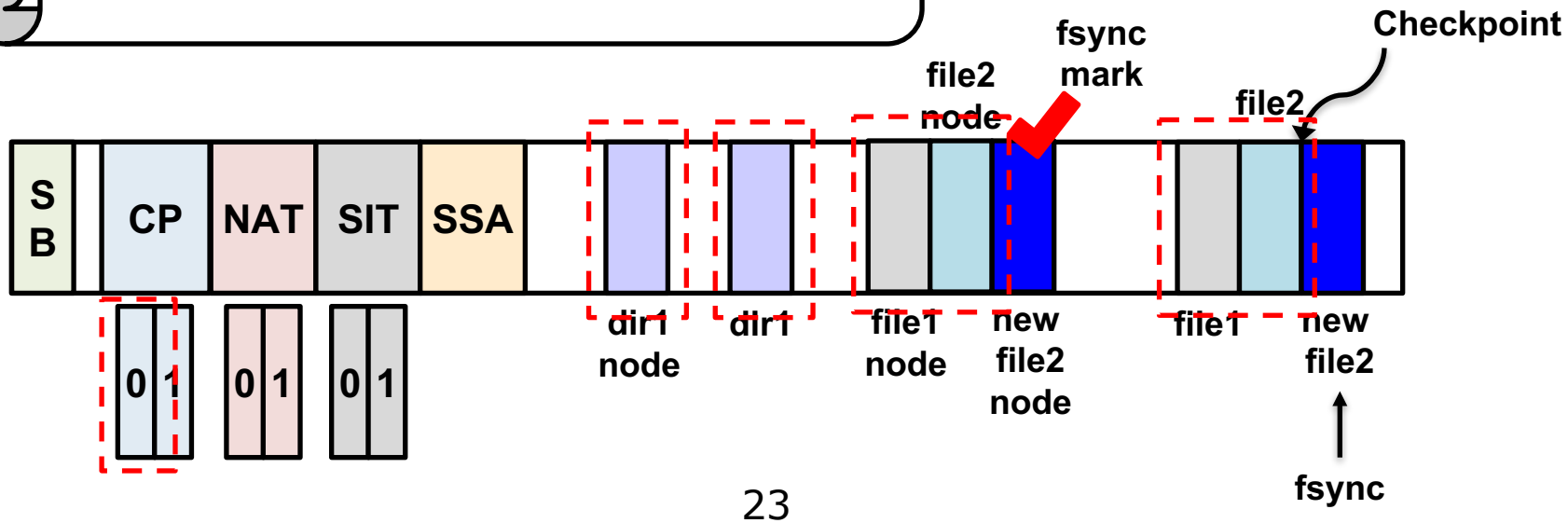


Recovery and 'fsync' Acceleration

1. Create dir1, file1, and file2
2. Create checkpoint
3. File2 update and fsync
4. Sudden Power Off

- Recovery

5. Roll-back to the latest stable checkpoint

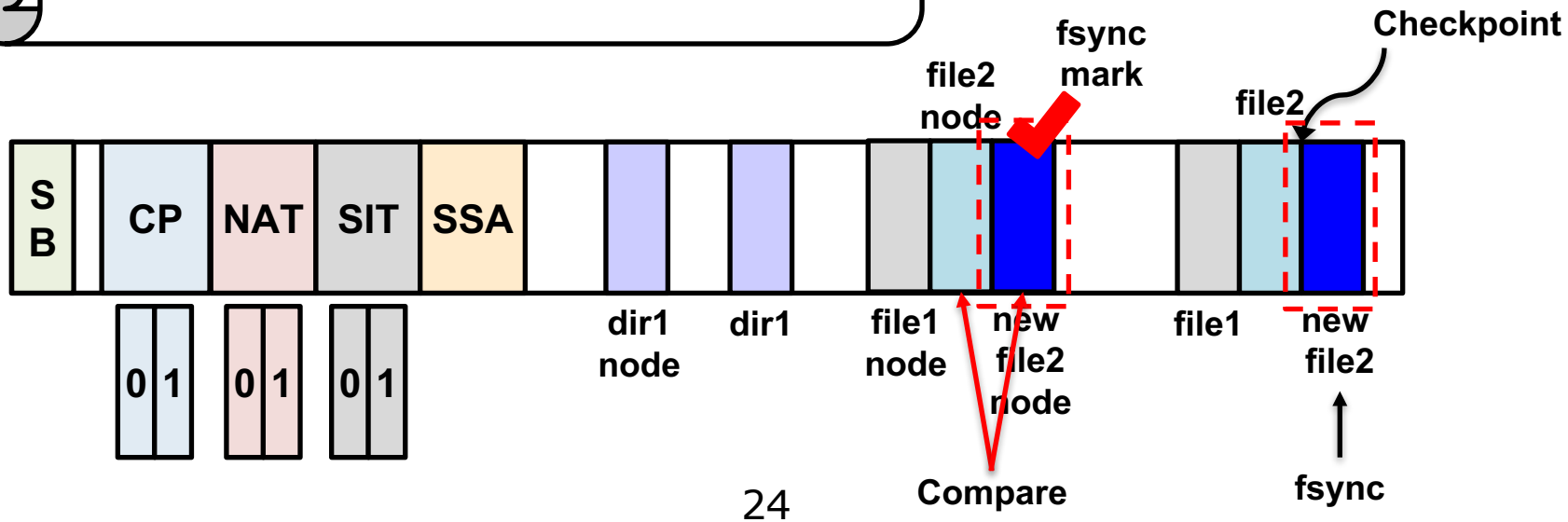


Recovery and ‘fsync’ Acceleration

1. Create dir1, file1, and file2
2. Create checkpoint
3. File2 update and fsync
4. Sudden Power Off

- Recovery

5. Roll-back to the latest stable checkpoint
6. Roll-forward to file2’s fsynced data

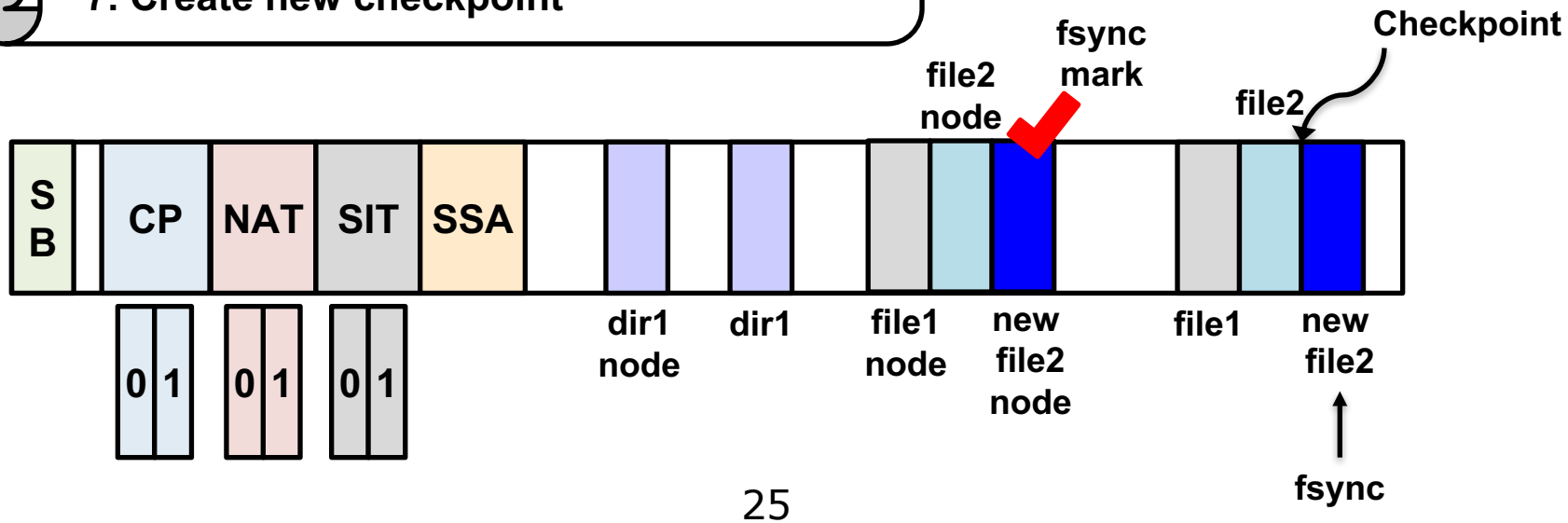


Recovery and ‘fsync’ Acceleration

1. Create dir1, file1, and file2
2. Create checkpoint
3. File2 update and fsync
4. Sudden Power Off

- Recovery

5. Roll-back to the latest stable checkpoint
6. Roll-forward to file2’s fsynced data
7. Create new checkpoint



Evaluation Setup

- **Hardware & Software Specs**

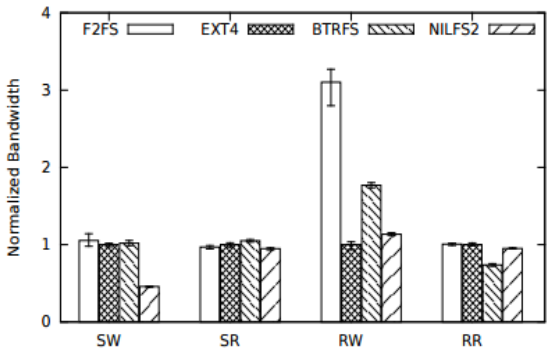
Target	System	Storage Devices
Mobile	**CPU: Exynos 5410** **Memory: 2GB** **OS: Linux 3.4.5** **Android: JB 4.2.2**	**eMMC 16GB** **(2GB partition)**
Server	**CPU: Intel i7-3770** **Memory: 4GB** **OS: Linux 3.14** **Ubuntu 12.10 server**	**SATA SSD 250GB** **NVMe SSD 960GB**

- **Summary of benchmarks**

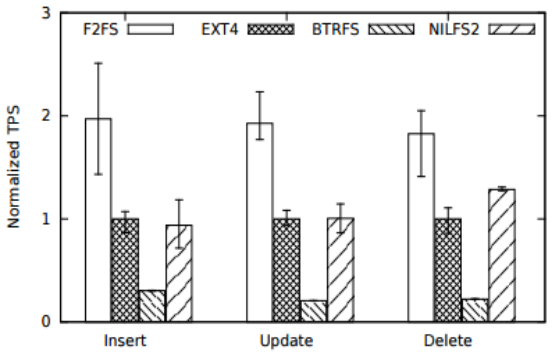
Target	Name	Workload	Files	File size	Threads	R/W	`fsync`
Mobile	iozone	Sequential and random read/write	1	1G	1	50/50	N
	SQLite	Random writes with frequent `fsync`	2	3.3MB	1	0/100	Y
	Facebook-app	Random writes with frequent `fsync`	579	852KB	1	1/99	Y
	Twitter-app	generated by the given system call traces	177	3.3MB	1	1/99	Y
Server	videoserver	Mostly sequential reads and writes	64	1GB	48	20/80	N
	fileserver	Many large files with random writes	80,000	128KB	50	70/30	N
	varmail	Many small files with frequent `fsync`	8,000	16KB	16	50/50	Y
	oltp	Large files with random writes and `fsync`	10	800MB	211	1/99	Y

Mobile Benchmark

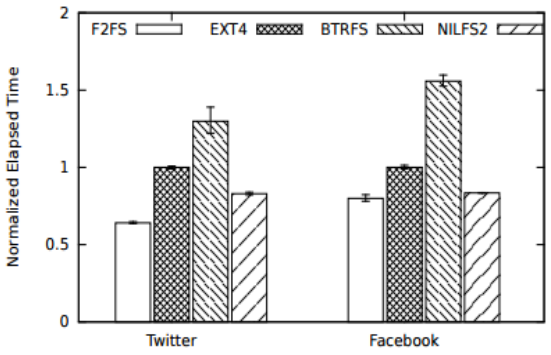
- **In iozone-RW, F2FS performs 3.1x better than ext4**
  - **In F2FS, more than 90% of writes are sequential**
- **F2FS reduces write amount per fsync at SQLite**
  - **F2FS reduces the amount of data writes by about 46% over Ext4**
- **F2FS reduces the elapsed time by 20% (facebook) and 40% (twitter) compared with Ext4**



(a) iozone



(b) SQLite

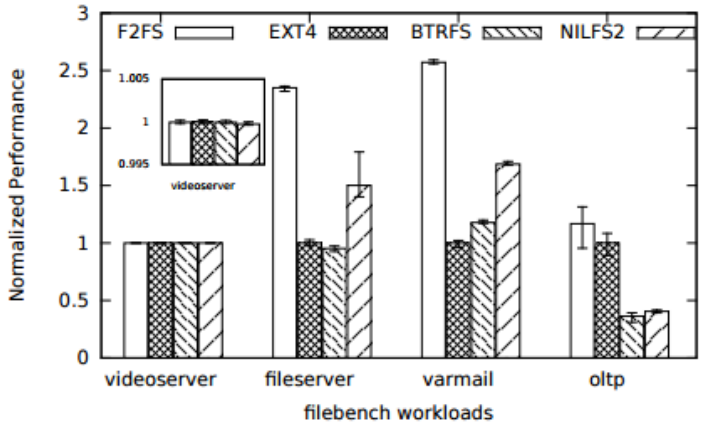


(c) Facebook-app and Twitter-app

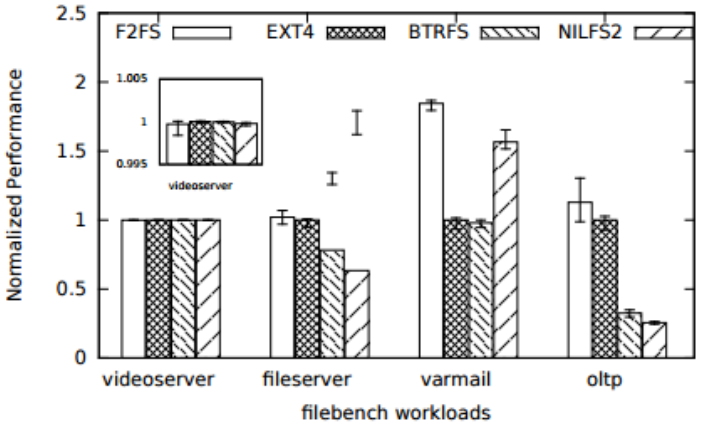
Server Benchmark

- **On SATA SSD, F2FS shows**
  - **2.5x better than ext4 on varmail benchmark**
  - **16% better than ext4 on Oltp benchmark**

- **On PCIe SSD, F2FS shows**
  - **1.8x better than ext4 on varmail benchmark**
  - **13% better than ext4 on Oltp benchmark**



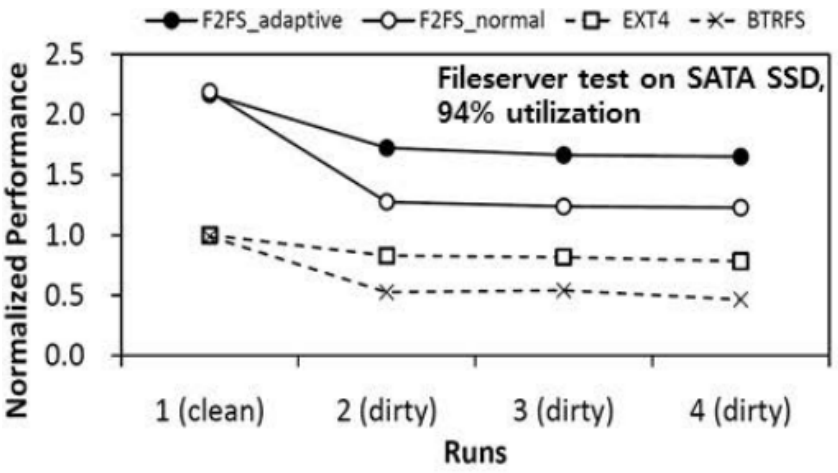
(a) SATA SSD

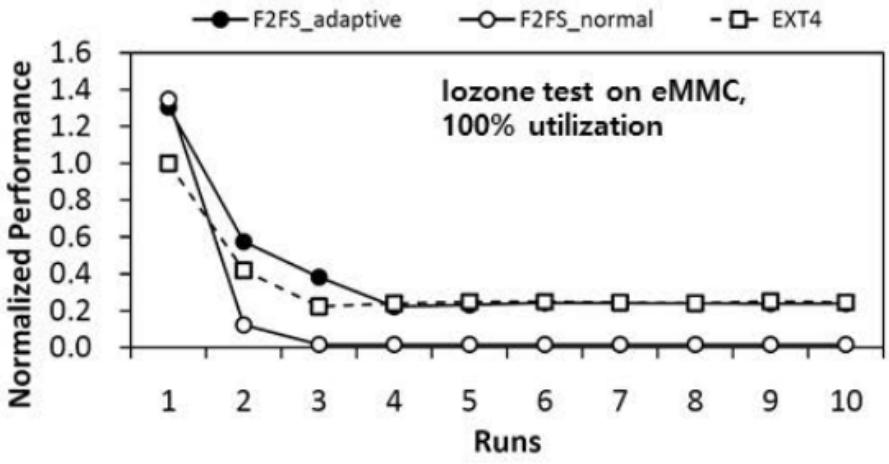


(b) PCIe SSD

Adaptive Logging Performance

- **Adaptive logging give graceful performance degradation under highly aged conditions**
  - **Fileserver test on SATA SSD (94% util.)**
    - **Performance improvement: 2x/3x over ext4/btrfs**
  - **IOzone test on eMMC (100% util.)**
    - **Performance is similar to ext4**





Conclusion

- **F2FS Contributions**
  - **Flash-Friendly on-disk layout**
  - **Efficient Index Structure**
  - **Multi-head logging**
  - **Adaptive logging**
  - **Recovery and fsync acceleration**

- **Evaluation**
  - **F2FS outperforms other FSs on various benchmarks**
  - **F2FS transforms random write to sequential write**
  - **F2FS reduce fsync overheads**
  - **Adaptive logging relieves cleaning overheads**

Thank you