IPLFS: Log-Structured File System without Garbage Collection

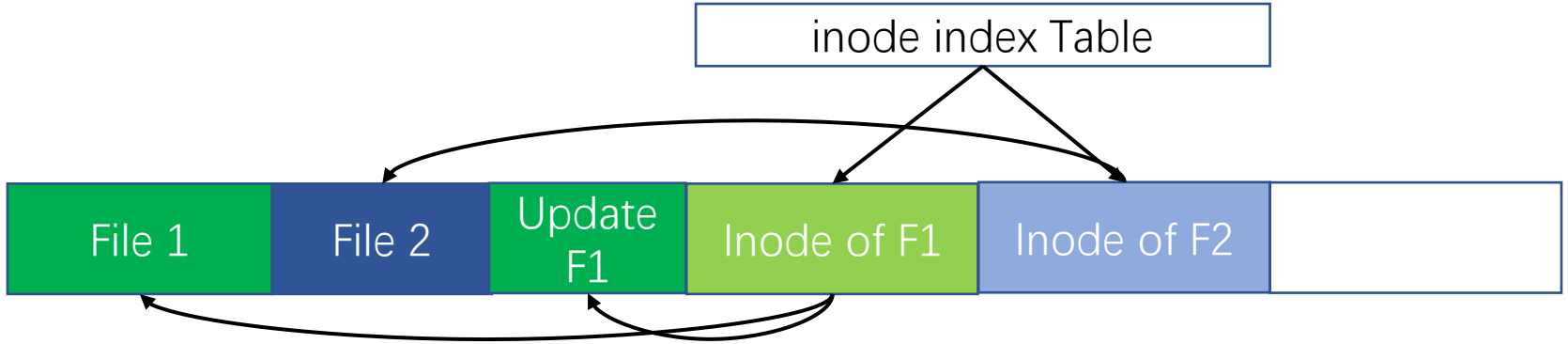
Juwon Kim, Minsu Jang, Muhammad Danish Tehseen, Joontaek Oh, and YouJip Won, KAIST

ATC'22

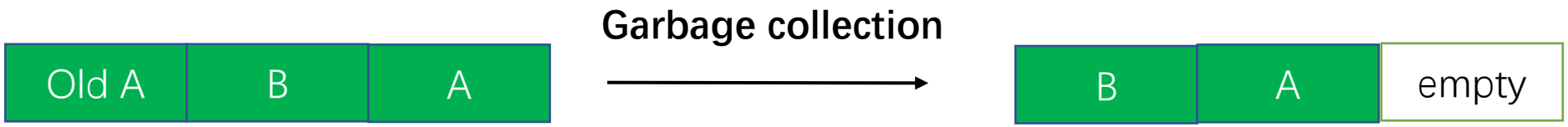
Background

➢Log-Structured System

- Sequential writes are faster than random writes
- Flash devices (SSD, etc) are better for sequential writes(life span and performance)

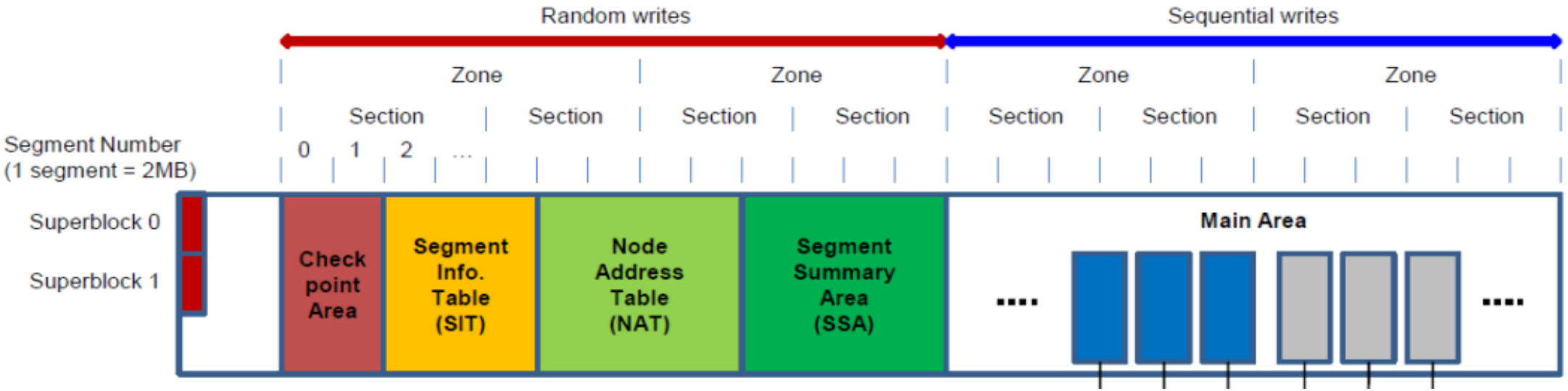


➢What to do if the LFS is full ?



Background

➢ **F2FS**



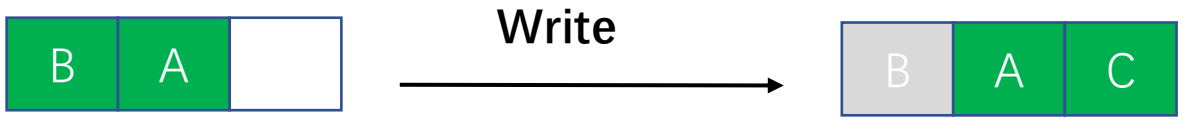
File system -> Zones -> Sections -> Segments -> Blocks

- Node Address Table: File name → inode address (**read and write**)
- Segment Summary Area: inode → father inode **(garbage collection)**
- Segment Info Table: Sgement valid info**(garbage collection)**
- Check Point Area: filesystem's consistency record **(crash recovery)**

Background

➢ Write

Append batch update to segment -> Append inode to segment -> update **NAT/SSA/SIT**



➢ Read

Read **NAT** to get inode address -> read data (find the latest version)

Background

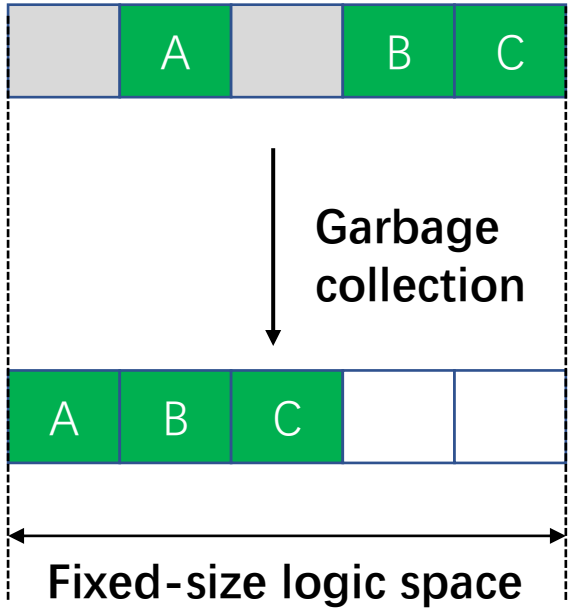
➢ Garbage collection

Happened when:

- There are **few free segments** in the file system
- Filesystem is idle

What to do:

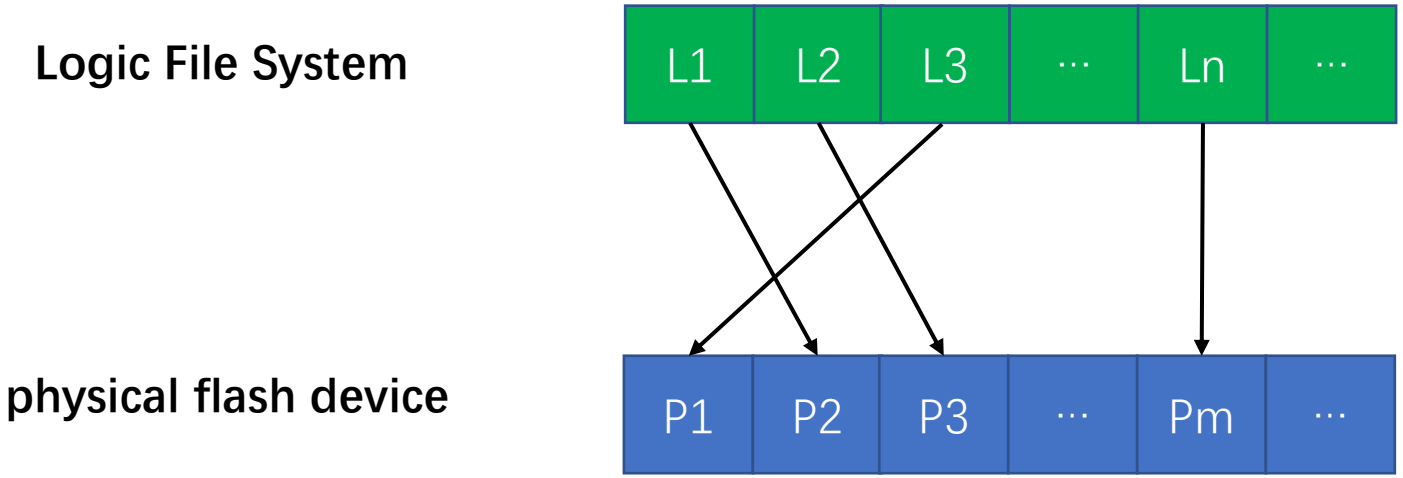
- Find the Segment with the most invalid blocks (Search **SIT**)
- **Remove invalid blocks & Compaction valid blocks**
- update metadata(**SSA** and **NAT**)



Background

LBA	PBA
L1	P2
L2	P3
…	…
Ln	Pm

➢File systems mapping to physical flash device



Logic block address

Physical block address

➢Flash Translation Layer

- **Address translation**(LBA to PBA)
- **Garbage collection** (erase blocks given by top-level filesystem)
- **Wear-leveling**(Improve life span)

Problem & IDEA

- **Problem in Garbage Collection**

  Garbage collection has high peak overhead

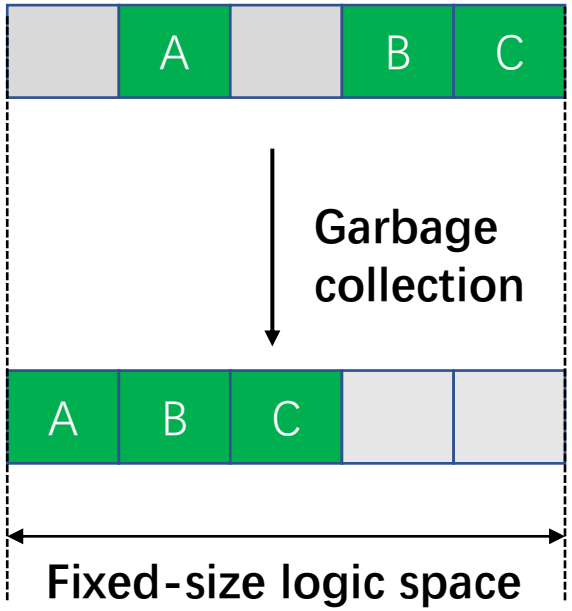
- **Idea**

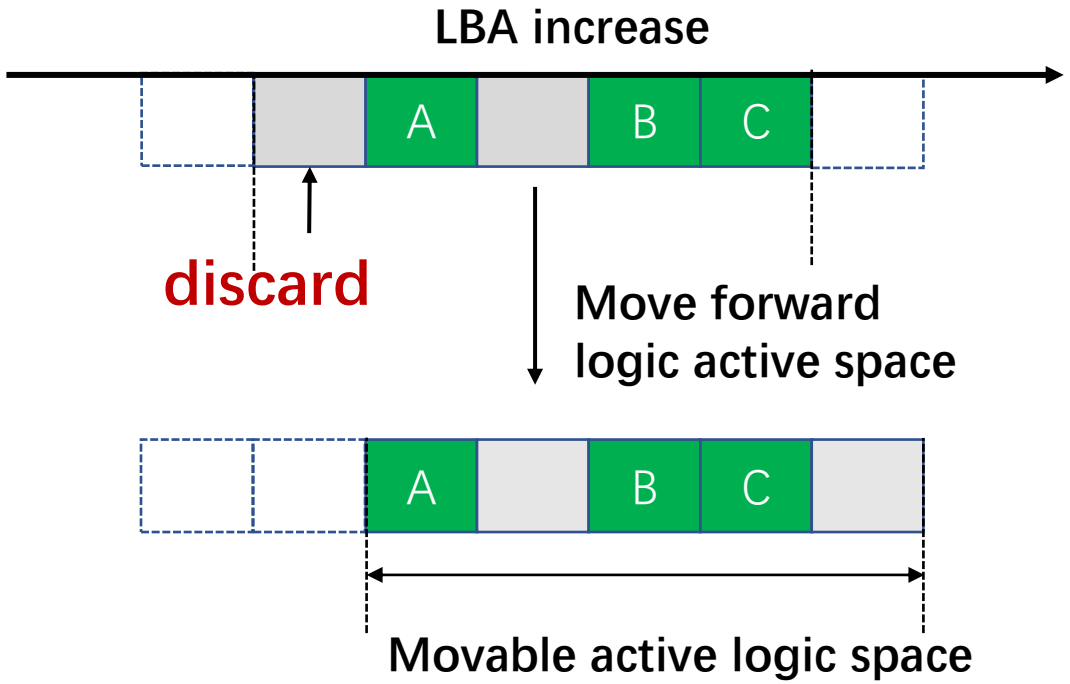
  -- Why exists garbage collection?
  -- Logic size of filesystem is limited (same as physical flash device)

  IDEA:
  - Design a file system (LFS) with infinite logical size
  - Spread out the peak overhead

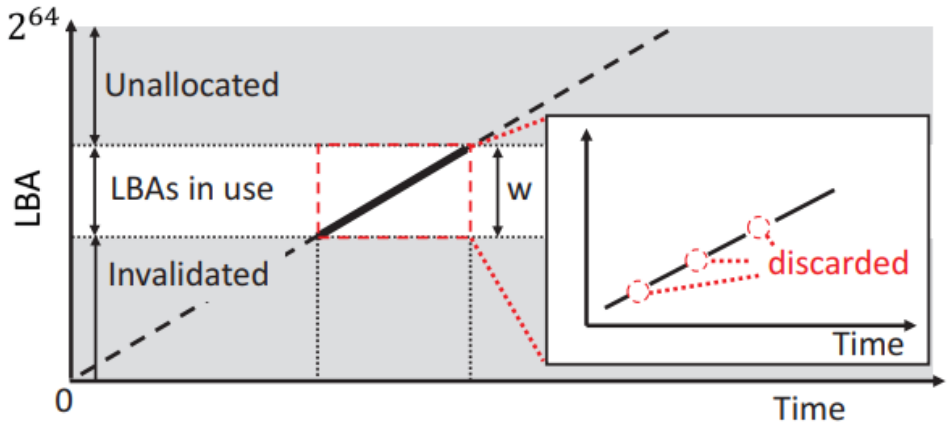
IDEA

- **Infinite logical size**





F2FS



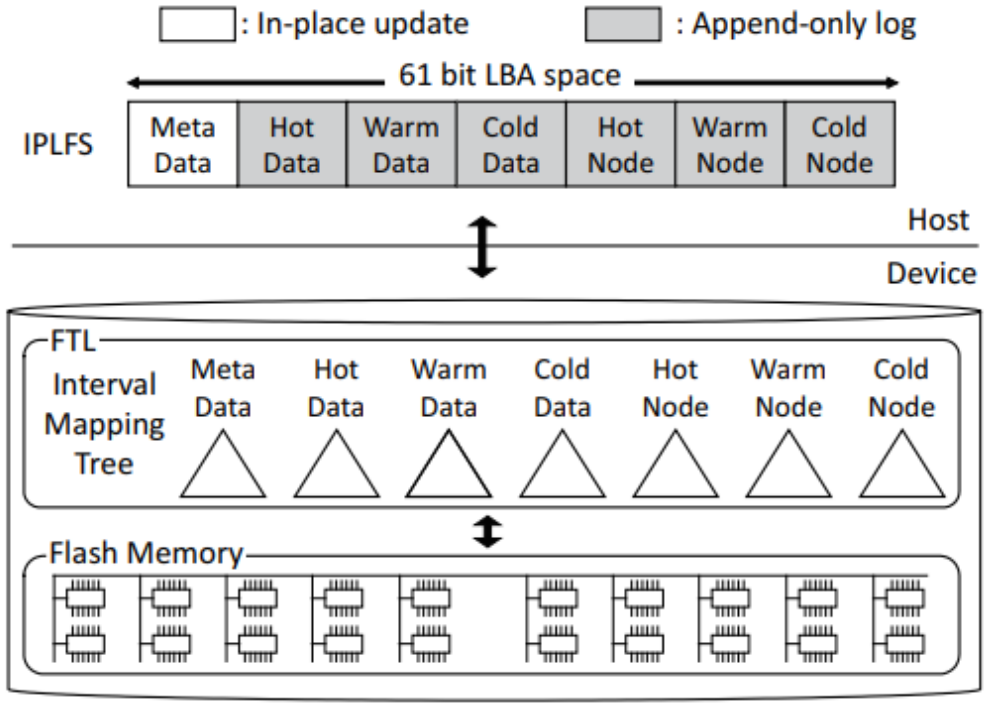
IPLFS

LBA increases with time going

Overview

➢ Overview

- **IPLFS**
  re-designed LFS base on F2FS
- **Interval Mapping**
  re-designed Flash translation Layer



IPLFS Design

➢ What is the size of the logical file system

LBA range is set to **0~2^61** , represented by a 64-bit numbers

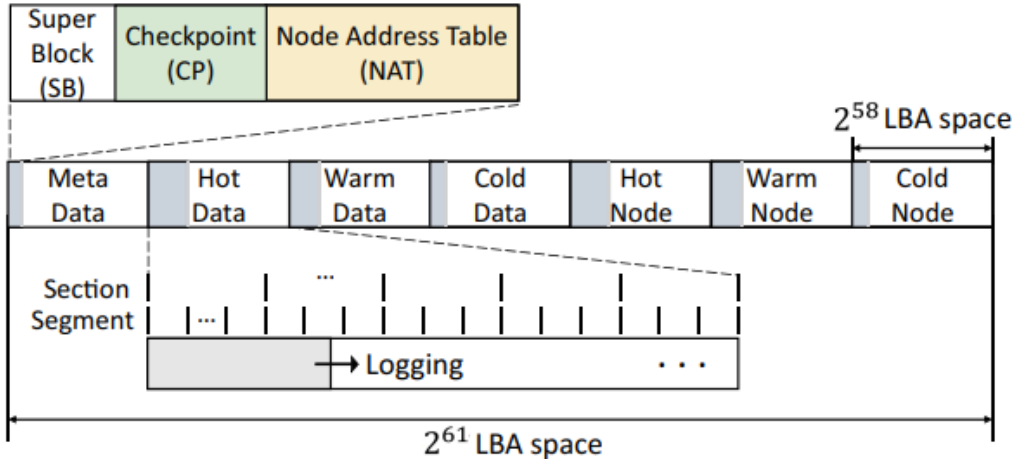
➢ Problem under huge LBA range

- **Current NAT is not suitable (bitmap will be too larger within this LBA range)**
- **Current SIT is not suitable (too many segment)**

➢ some other considerations

- Remove some data structures that only serve garbage collection

IPLFS Design



➢ Separate cold and hot data

- Total 6 kinds of block which are stored in 6 area

➢Adjust the data structure of metadata

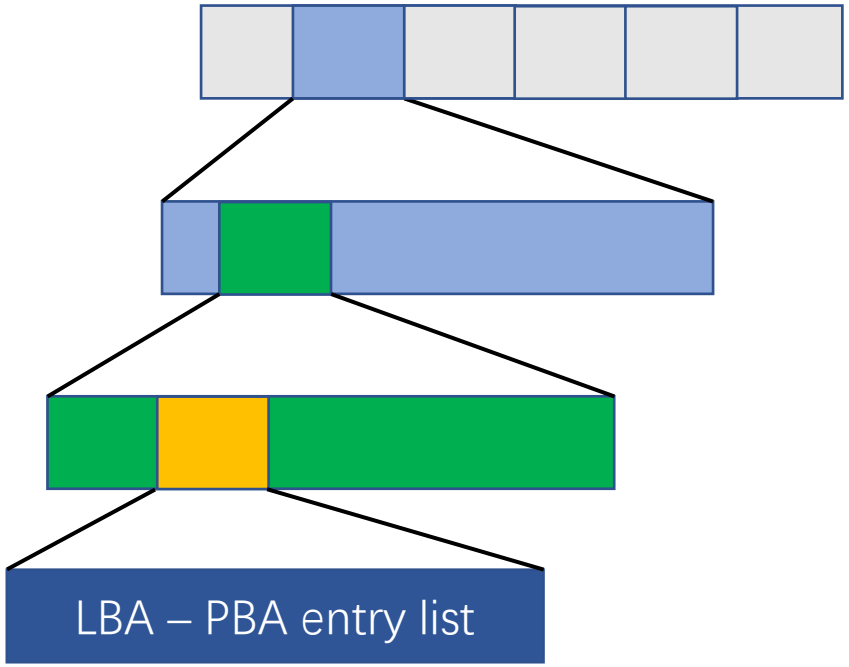
- **Limit** the size of NAT (max size is PBA space or not LBA space)
- **Remove** SIT and SSA
- **Add a new metadata Discard Bitmap** to support crash recovery

Metadata	Usage
CP	Crash recovery
NAT	Read/Write/GC
SIT	GC/Crash recovery
SSA	GC

Use a hash map to store invalid blocks

FTL Design

- **Problem**
  - Current LBA is not suitable (too large LBA-to-PBA mapping)
  - High memory overhead
- **Solution -- Interval Tree**



Root node, LBA range: [0 ~ X]

Zone node, LBA range: [y ~ y + 4G]

Mapping Segment node, LBA range: [z ~ y + 16MB]

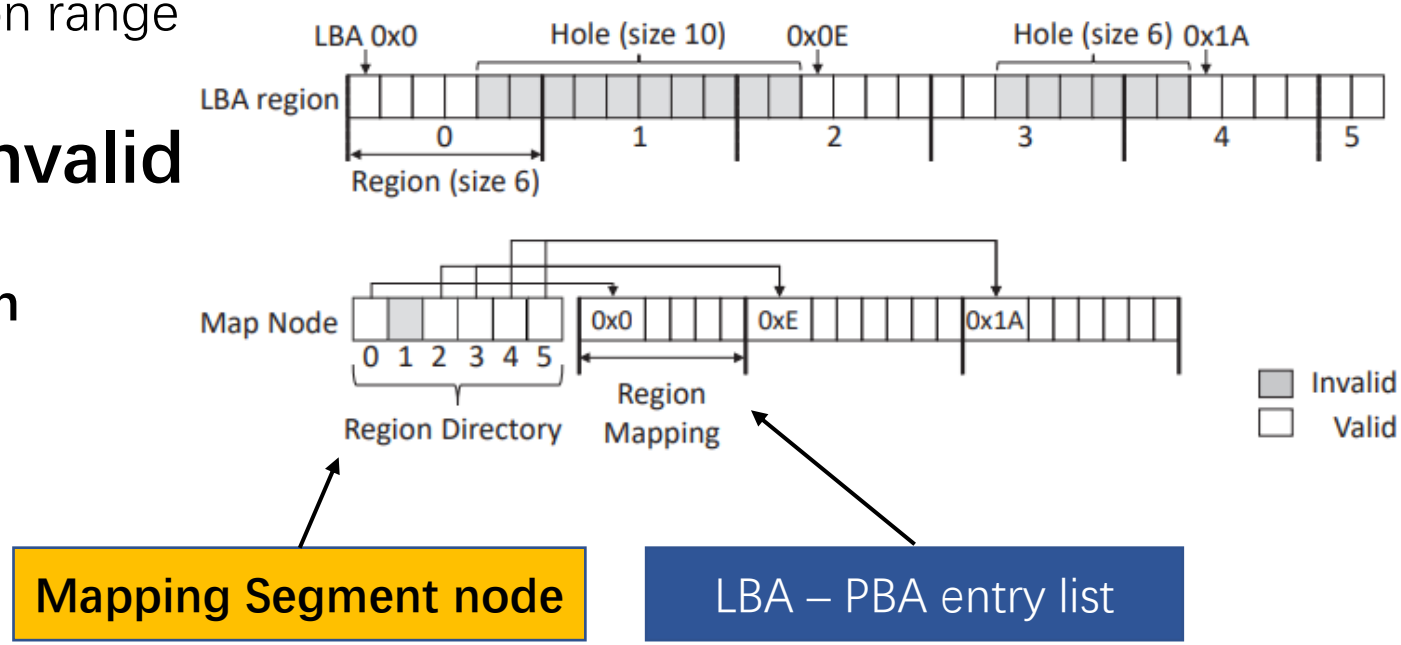
FTL design

➢Mapping Segment node

- Divide the segment range into **multiple regions**
- The blocks in each region are either **all valid** or **all invalid**
- Only Invalid region will be stored
- Use background thread maintain region range

➢Problem Blocks becoming invalid

- Use a background thread for **compaction**



FTL Design

➢Dynamic update active zones

- Trigger when: Too few free zones
- What to do: assign some new empty zones
- Discard invalid blocks ?

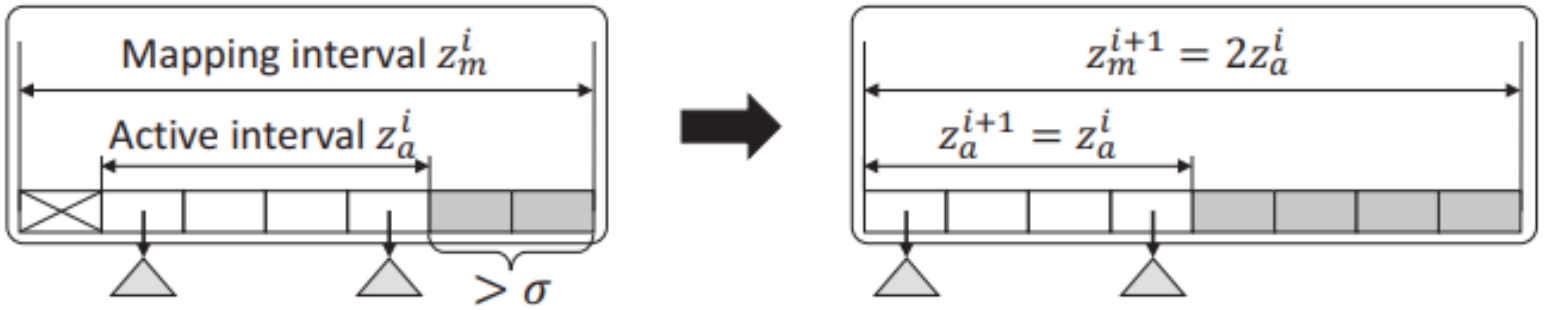


Figure 9: Updating the Mapping Interval

Evaluation

➢Environment

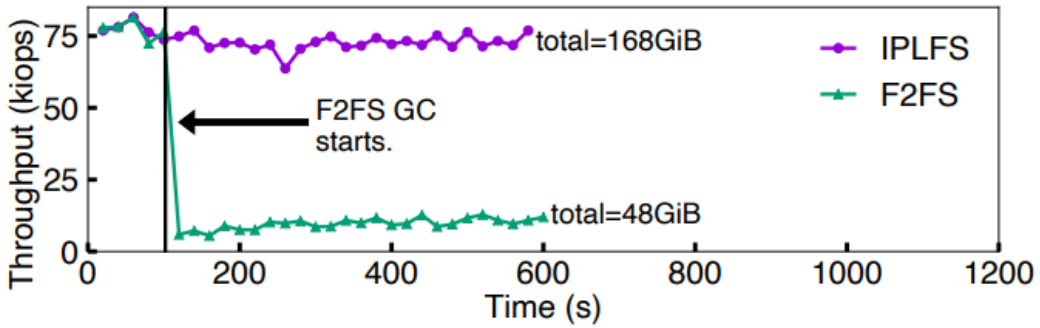
- CPU and DRAM: I7-7770K and 8GB DRAM
- Storage: Open SSD (230GB)
- Filesystem: IPLFS base on F2FS

➢Testing object

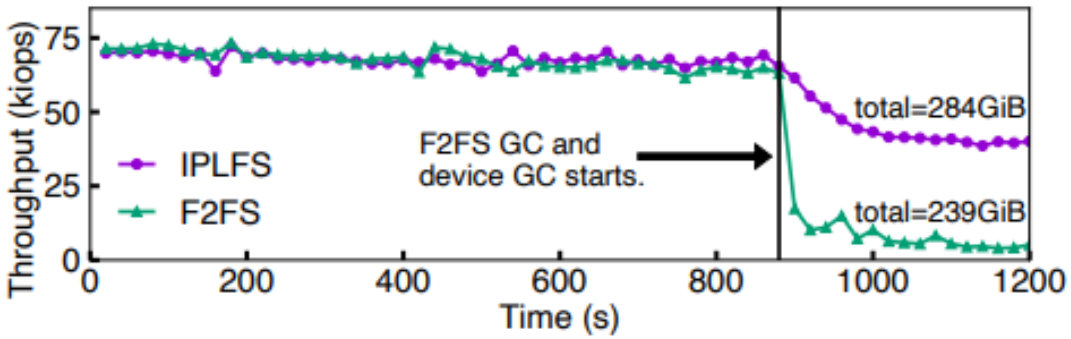
- IPLFS
- F2FS (baseline)

Evaluation

➢Throughtput



(a) Only with F2FS garbage collection. Time: 600s, file size: 28GByte, partition size: 30GByte



(b) F2FS garbage collection and FTL garbage collection. Time: 1200s, file size: 210GByte, partition size: 230GByte.

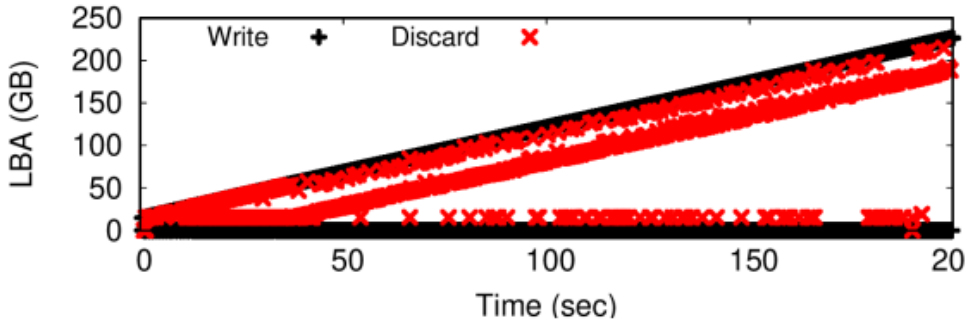
IPLFS: No GC
F2FS: Filesystem GC

IPLFS: Device GC (flash erase invalid block
F2FS: Filesystem GC + Device GC

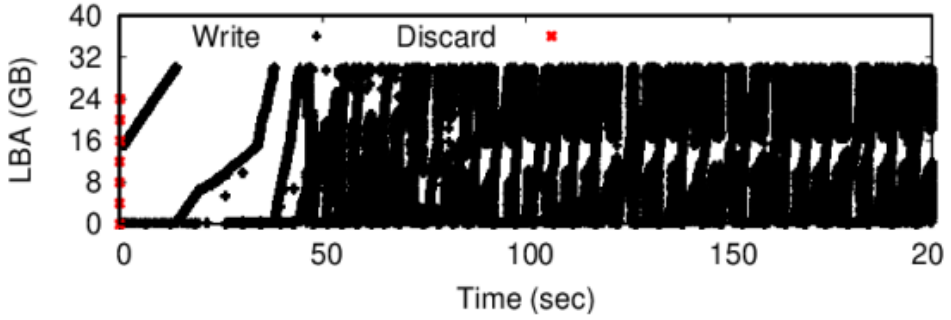
Evaluation

➢Discard Policy of IPLFS and F2FS

- **IPLFS**
  - Frequent discard
  - No GC and growing LBA
- **F2FS**
  - No frequent discard
  - Data block recycling(by GC)



(a) IPLFS



(b) F2FS

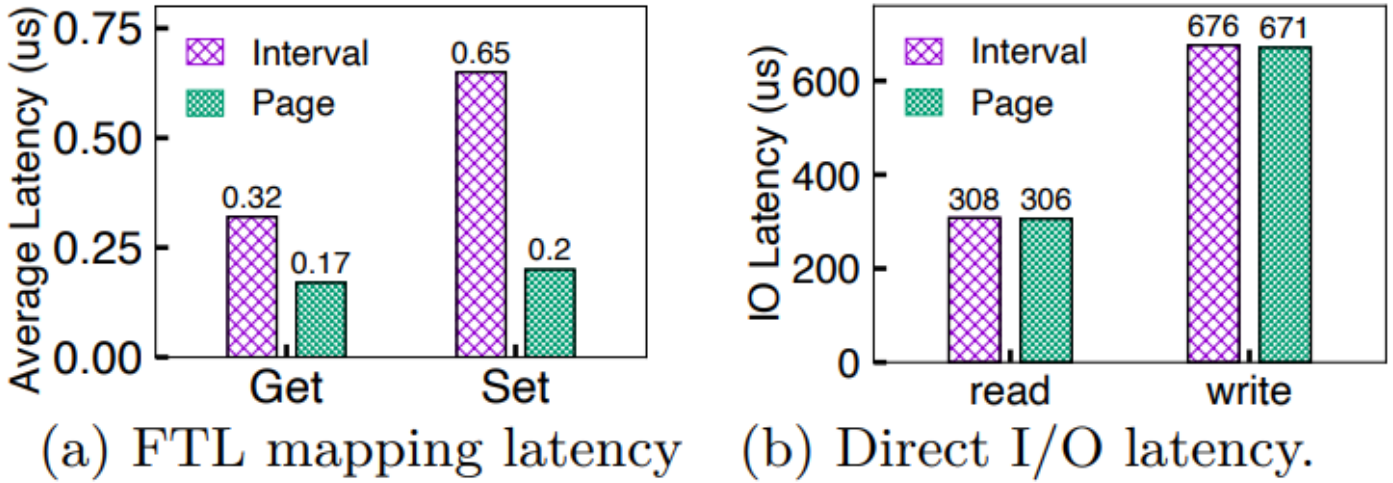
Data area

Metadata area

Partition size: 30GB

Evaluation

➢Latency Analysis



(a) FTL mapping latency (b) Direct I/O latency.

- IPLFS is 3.3x slower then F2FS in FTL mapping
- Both filesystems have almost the same IO latency

Conclusion

- **Problem**

  F2FS has high garbage collection overhead

- **Address the problem**

  Logic size of filesystem is limited

- **Idea**

  - Design a file system (LFS) with infinite logical size
  - Spread out the overhead (Remove GC and Use more frequent discard policy)

- **Challenge and Solution**

  Not suitable data structure in F2FS -> Redesign metadata (use hash map, etc..)

  Huge FTL table -> Redesign FTL (use interval tree)

疑问

- **关于文件系统的大小**

  在逻辑文件系统大于物理存储设备大小的前提下，如果我向文件系统存入数据实际大小大于物理存储设备大小怎么办？