USENIX ATC '22
JULY 11-13, 2022
CARLSBAD, CA

IPLFS: Log-Structured File System without Garbage Collection

Juwon Kim, Minsu Jang, Muhammad Danish Tehseen, Joontaek Oh, Youjip Won

KAIST

Outline

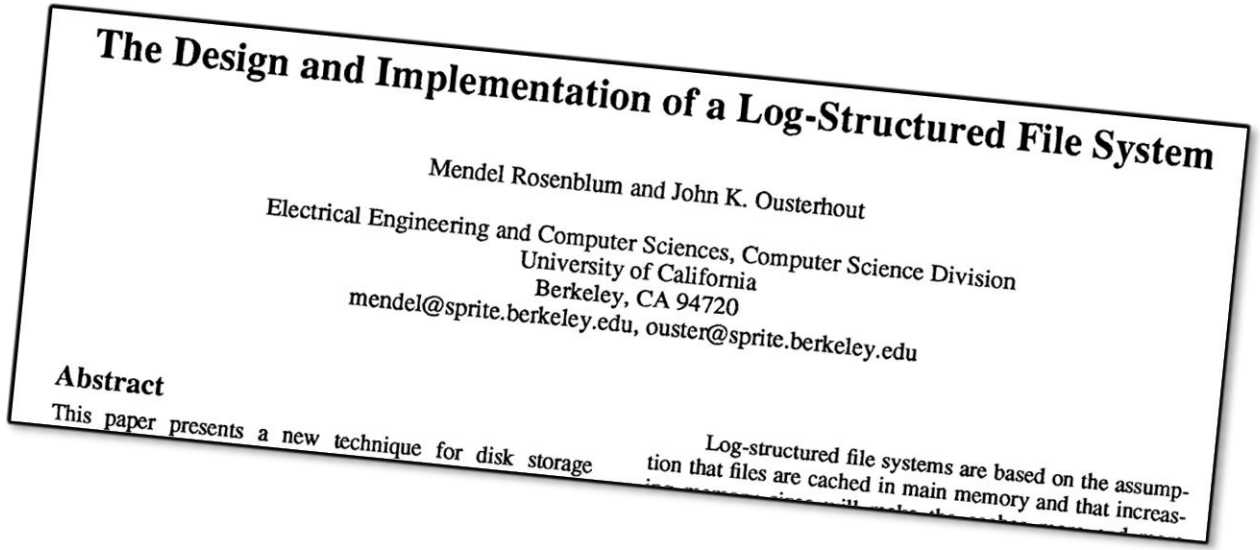
- ✓ Background & Motivation
- ✓ Problem Formulation
- ✓ IPLFS: Log-structured Filesystem for Infinite Partition
- ✓ Interval Mapping
- ✓ Evaluation
- ✓ Conclusion

Background & Motivation

Log-Structured Filesystem

Rosemblum et al. (SOSP '91)

- Write optimized filesystem.
- Treats the filesystem partition as a single log.
- Append only



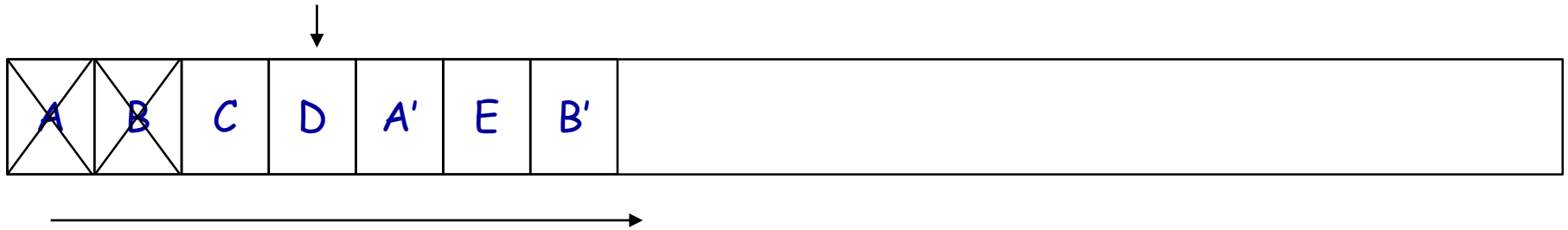
The Design and Implementation of a Log-Structured File System

Mendel Rosenblum and John K. Ousterhout

Electrical Engineering and Computer Sciences, Computer Science Division
University of California
Berkeley, CA 94720
mendel@sprite.berkeley.edu, ouster@sprite.berkeley.edu

Abstract

This paper presents a new technique for disk storage ... Log-structured file systems are based on the assumption that files are cached in main memory and that increas-

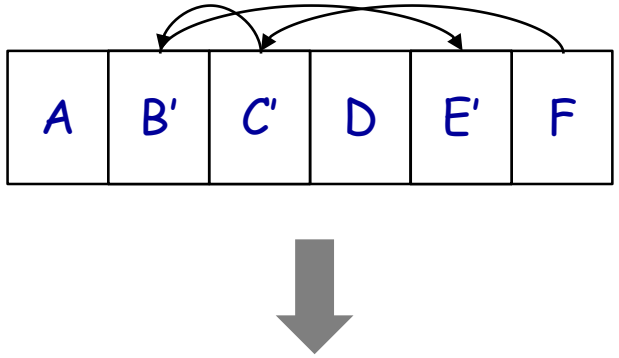



```
write (D) // new block
update (A') ; // A → A'
write (E) ; // new block
update (B) ; // B → B'
```

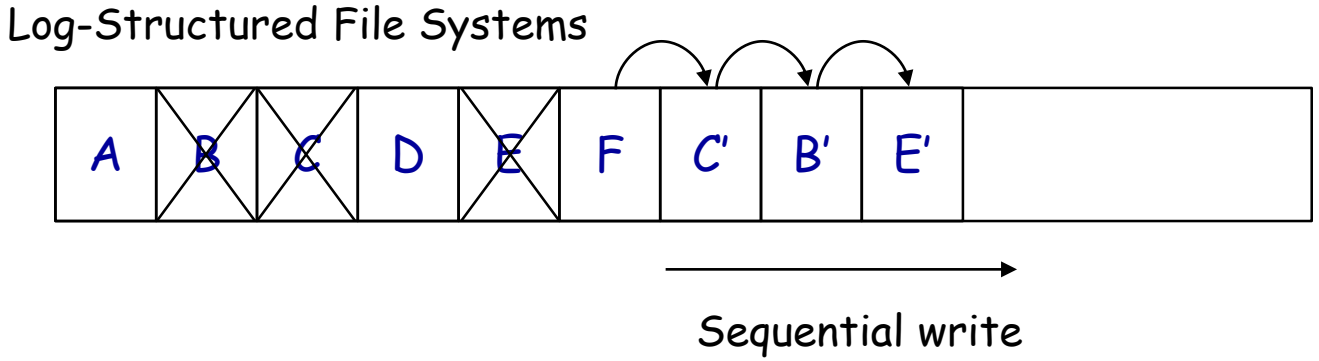
Log-Structured Filesystem

- Random Writes on a file.

```
update (C') ; // C → C'
update (B') ; // B → B'
update (E) ; // E → E'
```



- Sequential Write on LFS



Log-Structured Filesystem

Brilliant idea

- Efficiently handle random write workload.
- Perfect fit for write-dominant workload.
- Perfect fit for the slow write device. Such as Flash storage and SMR Disk.

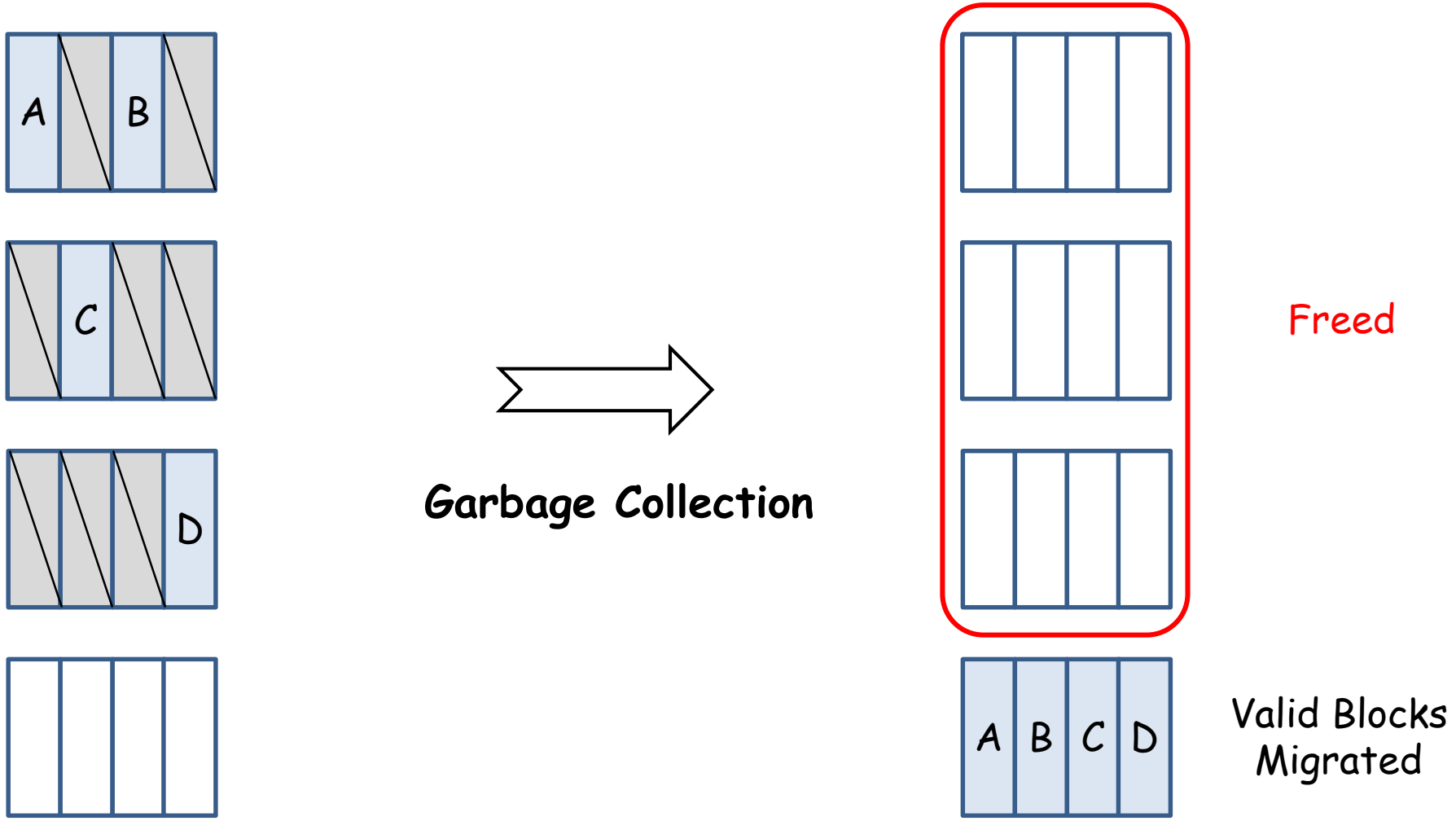
But, it has not been well received (till very recently.)

- Debut at SOSP '91.
- Default filesystem at Google Android smartphone. (F2FS, 2018).
- Few deployment at data center/server (NILFS, ACM SIGOPS '06)

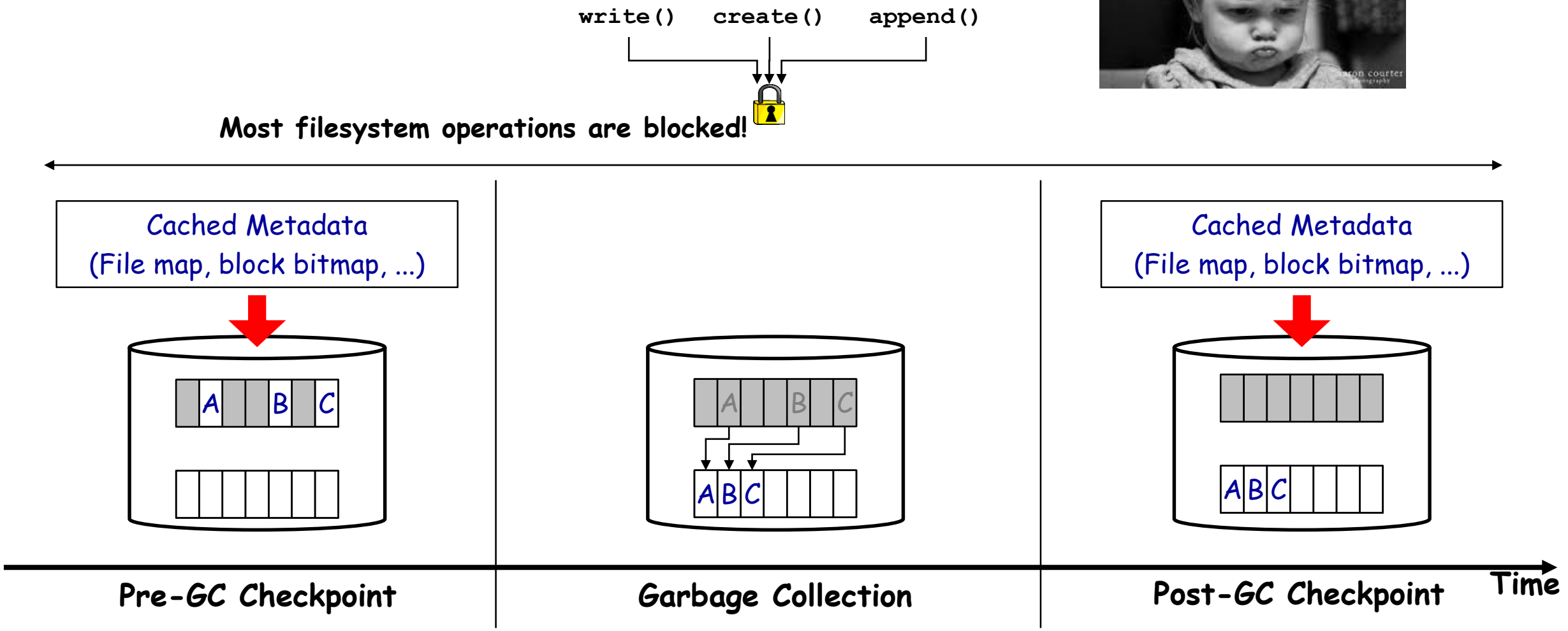
The Garbage Collection

Garbage Collection?

When there lack of free blocks in the filesystem partition, LFS reclaims the invalid blocks.

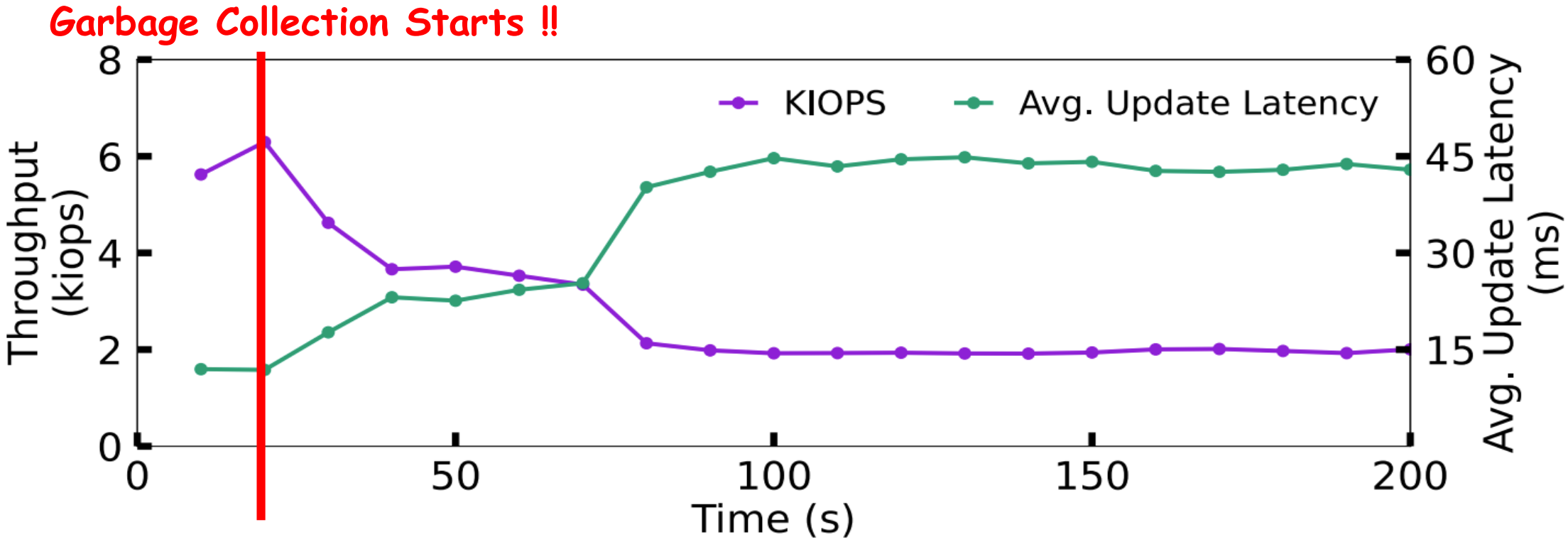


Garbage Collection is costly.



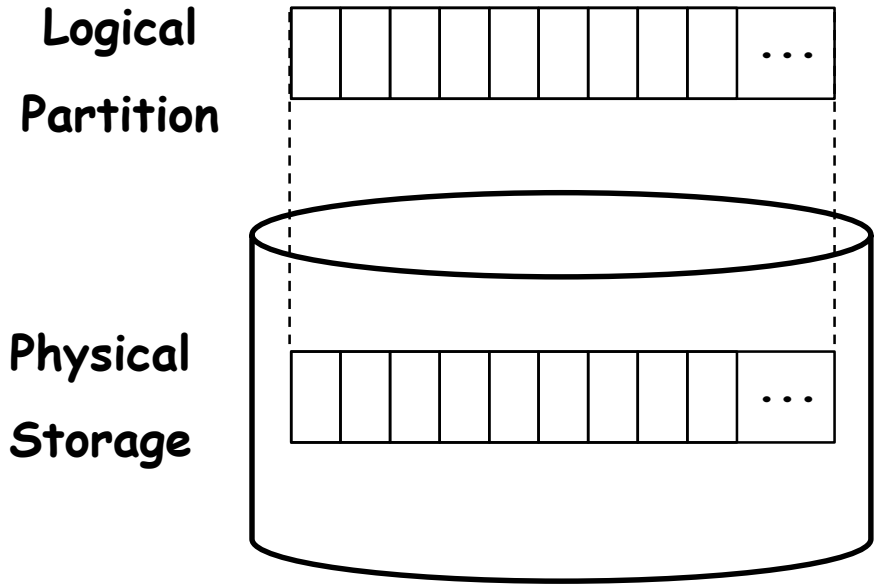
Overhead of Garbage Collection

- Unpredictable !
- No worst case bound!



Benchmark: MySQL YCSB Workload A (Disk Utilization: 90%)

Why does Garbage Collection exist?



The logical partition size is limited

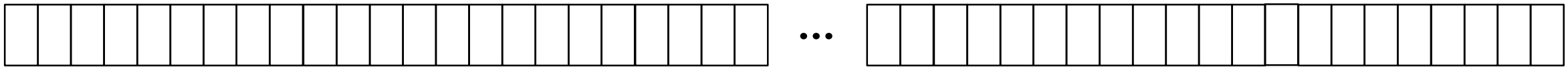
(by the size of the physical storage).

What if …

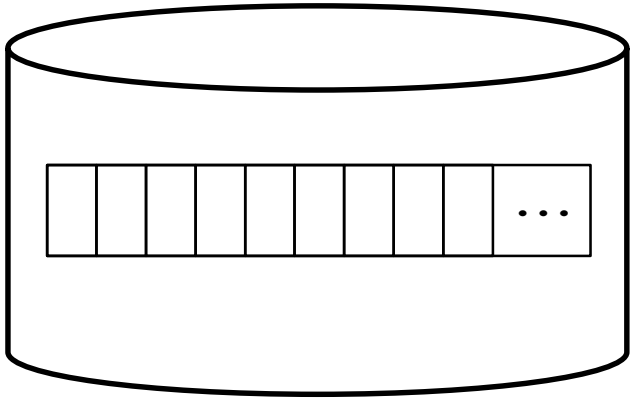
Separate the file system partition size from the disk size.

Make the filesystem partition size very large.

Virtually **Infinite Logical Partition, 8 Zbyte (2^{64} sectors)**



Physical Storage 10 TByte

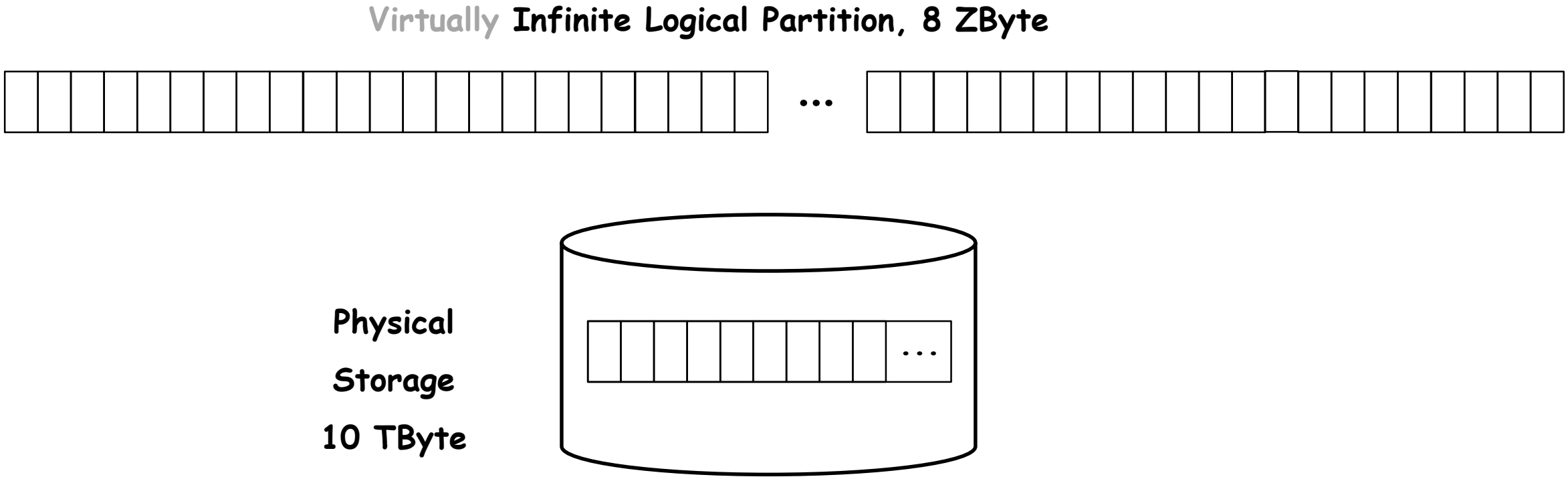


Before,

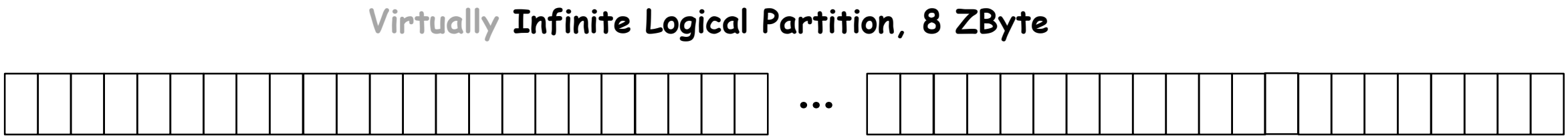
valid blocks + invalid blocks + free blocks ≤ storage size

Now,

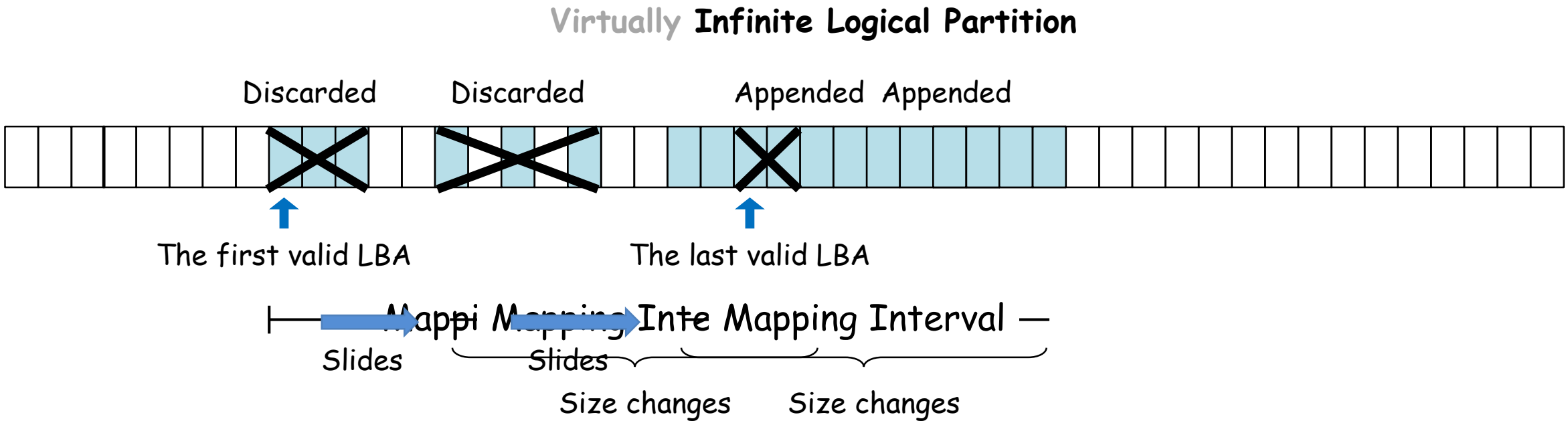
valid blocks ≤ storage size



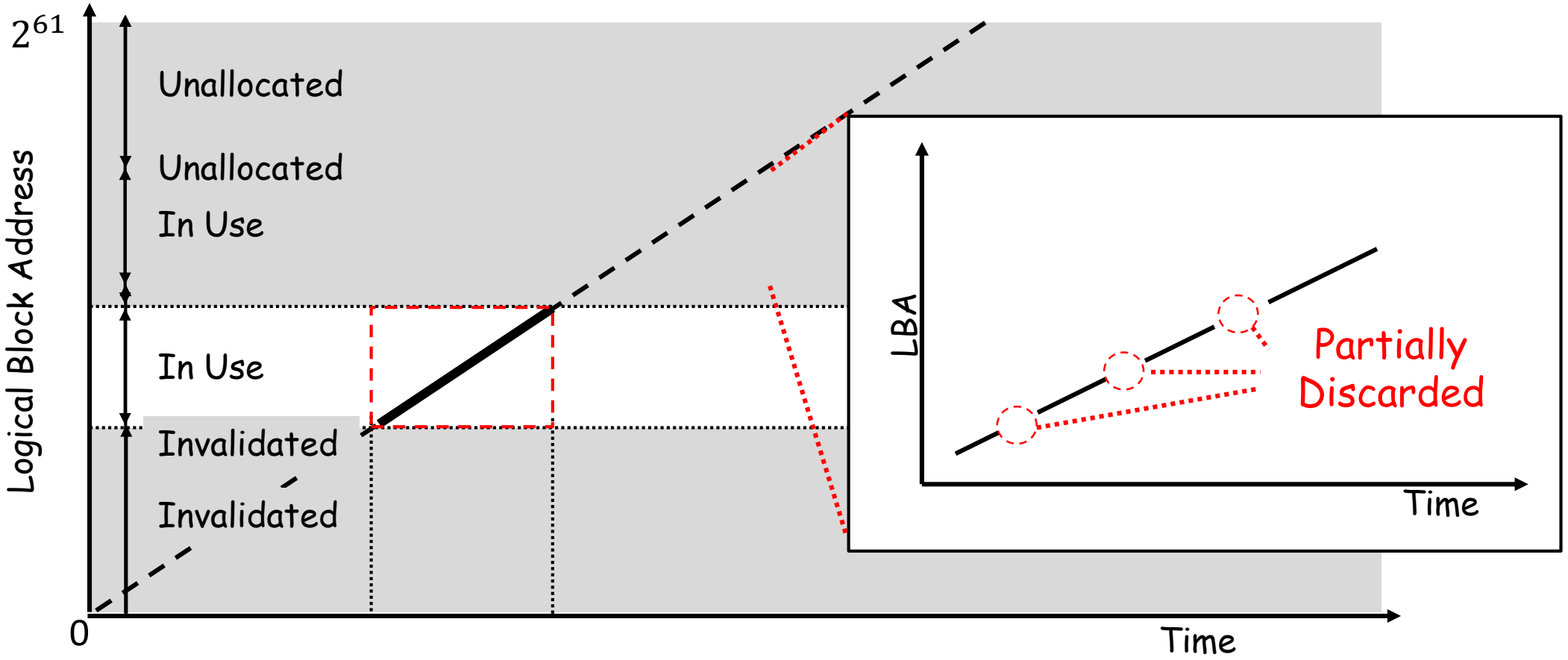
Flash cell can be erased or programmed for only limited number of times, e.g. 10K for SLC. The Flash storage lifespan expires long before it fully uses up the very large logical partition size.



Actively Used File System Partition



Actively Used File System Partition

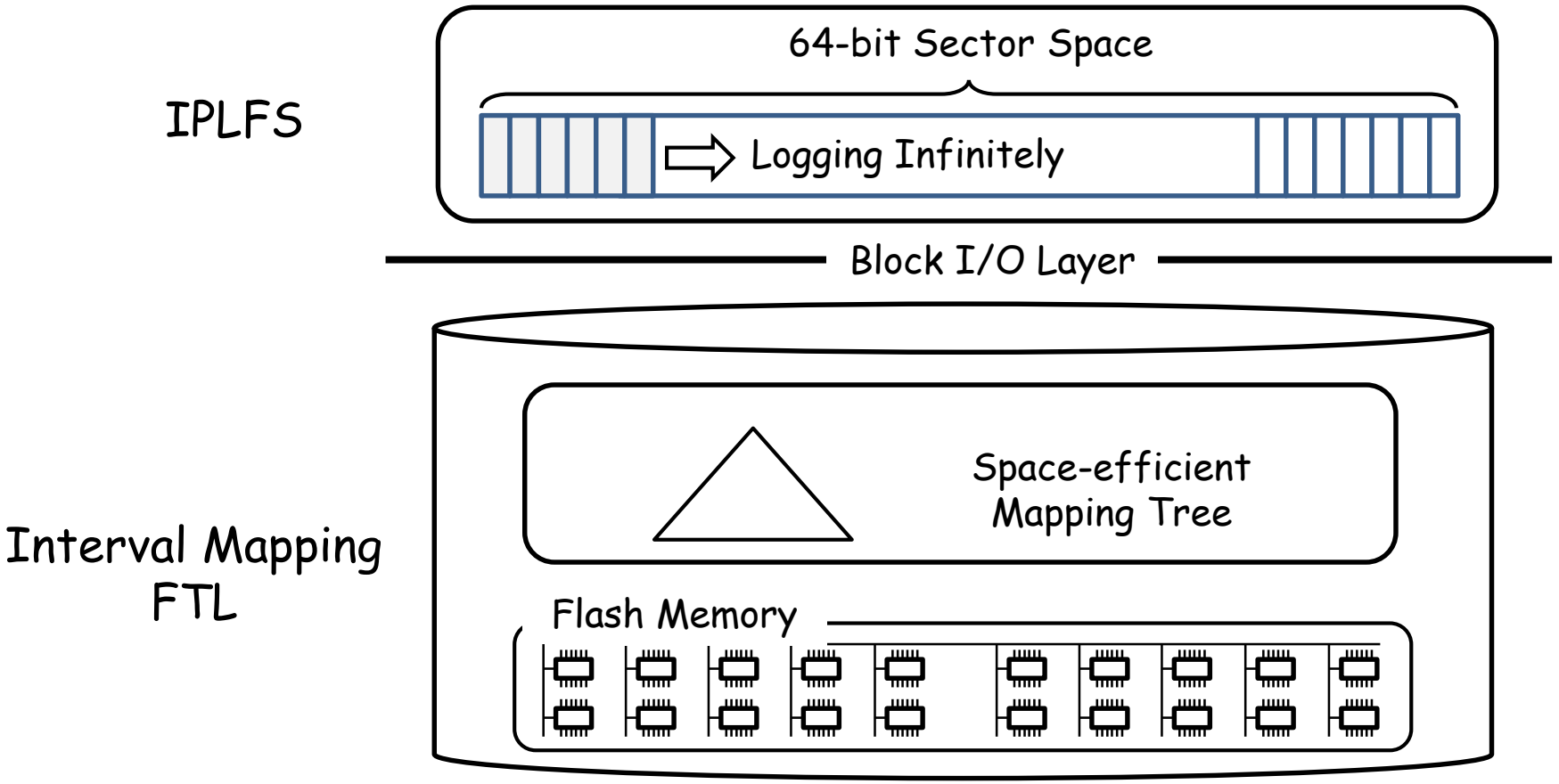


Problem Formulation

Handling very large file system partition

- ✓ File system
  - ➢ Metadata Overhead
- ✓ FTL
  - ➢ Mapping Table Overhead

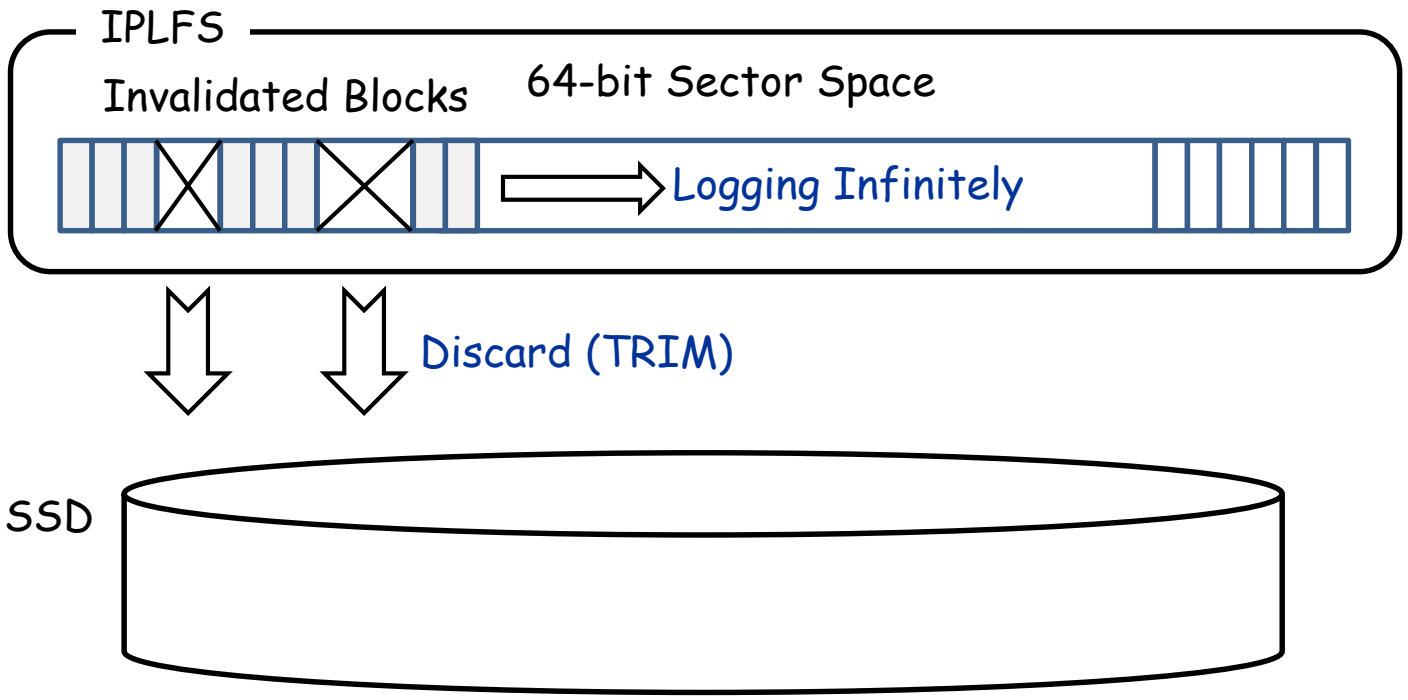
IO Stack for GC-less filesystem



IPLFS: Log-Structured File System for Infinite Partition

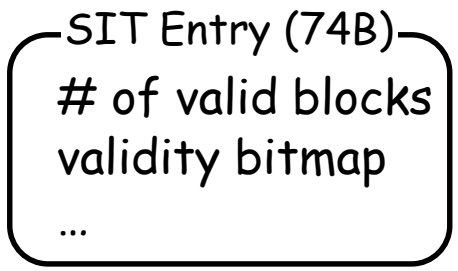
IPLFS

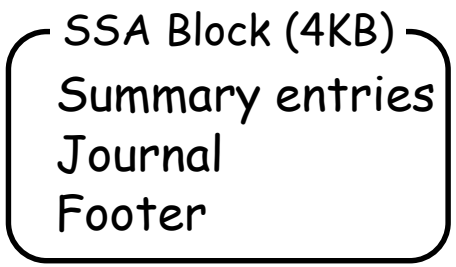
- ✓ Logging Infinitely on 64-bit Sector Space, 8 ZByte
- ✓ Discard Logging



Metadata for Log-structured filesystem

- Block bitmap: O(logical partition size)
- Reverse mapping: O(logical partition size)
- File index: the number of files in the partition: O(storage size)





Super Block	Checkpoint Area	Block Bitmap (SIT in F2FS)	Reverse Map (SSA in F2FS)	File Index (NAT in F2FS)	
4KB	4MB	512PB in IPLFS	8EB in IPLFS		

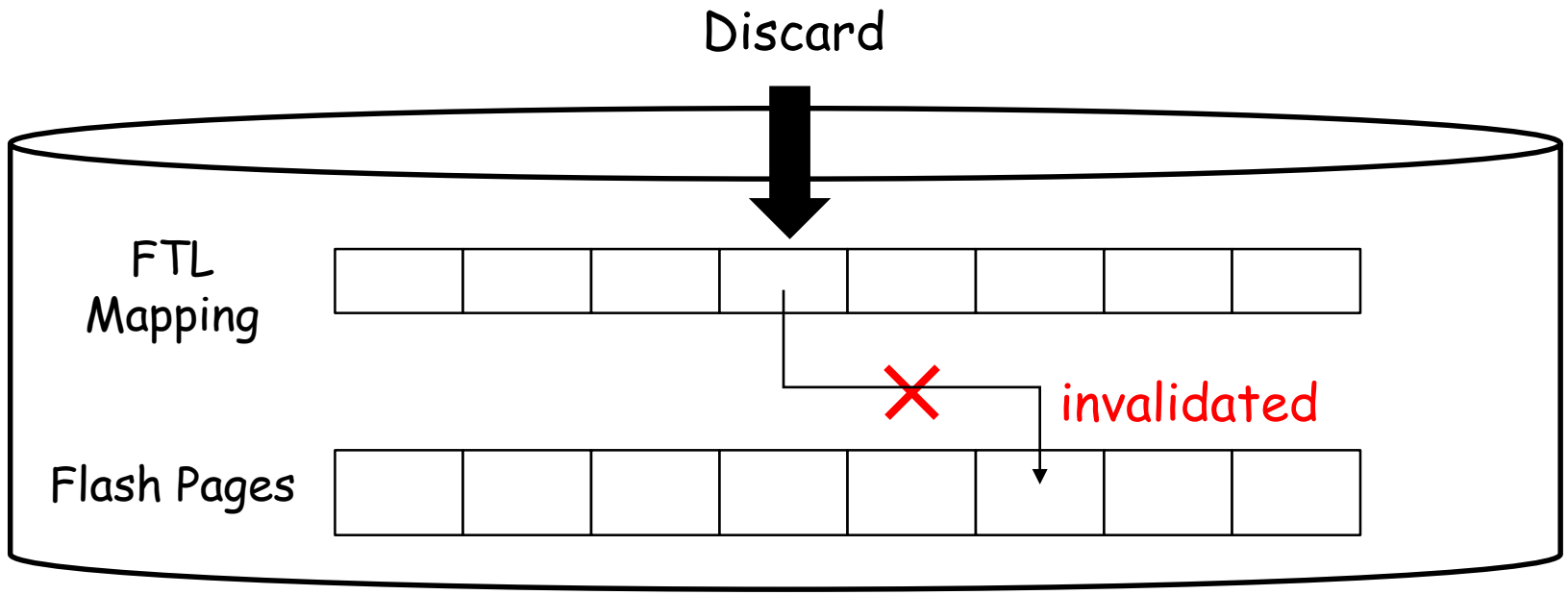
Metadata for IPLFS

- ~~Block bitmap~~
- ~~Reverse mapping~~
- File Index: the number of files in the partition: O(storage size)

Discard Command

```
discard [begin_LBA, length]
```

- **Inform SSD of the filesystem blocks that are no longer used by the filesystem.**
- **Prohibit FTL *GC* module from migrating invalid filesystem blocks.**

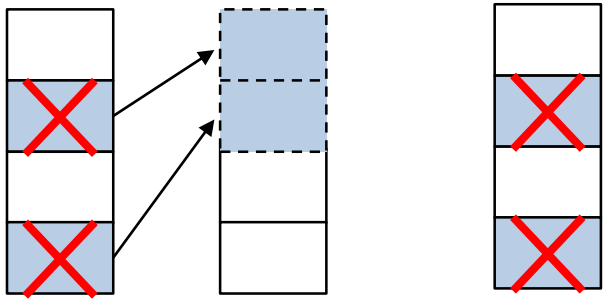


Discard the file system blocks (F2FS)

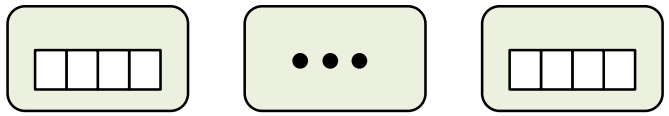
At file system operation

Garbage Collection

truncate, unlink



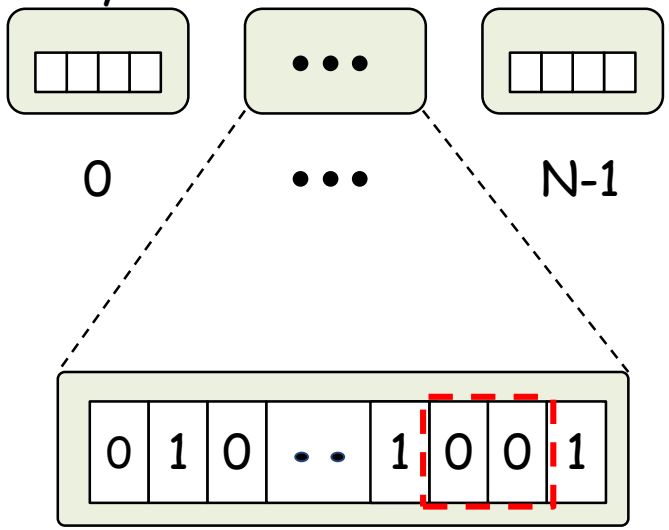
Invalidate the block bitmap.



At checkpoint

Scan the dirty Block Bitmap
&
Create discard commands

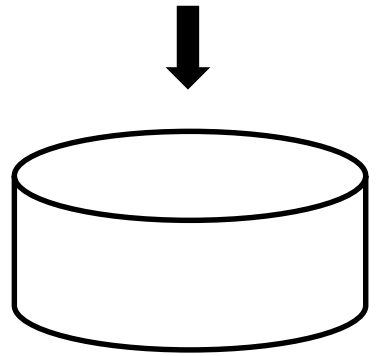
Dirty SIT Entries



Dispatch the discards!

Discard Command

[Start LBA, Size]



At 50 msec interval
at most 8 discards at a time
Only when IO is idle.

Metadata For IPLFS

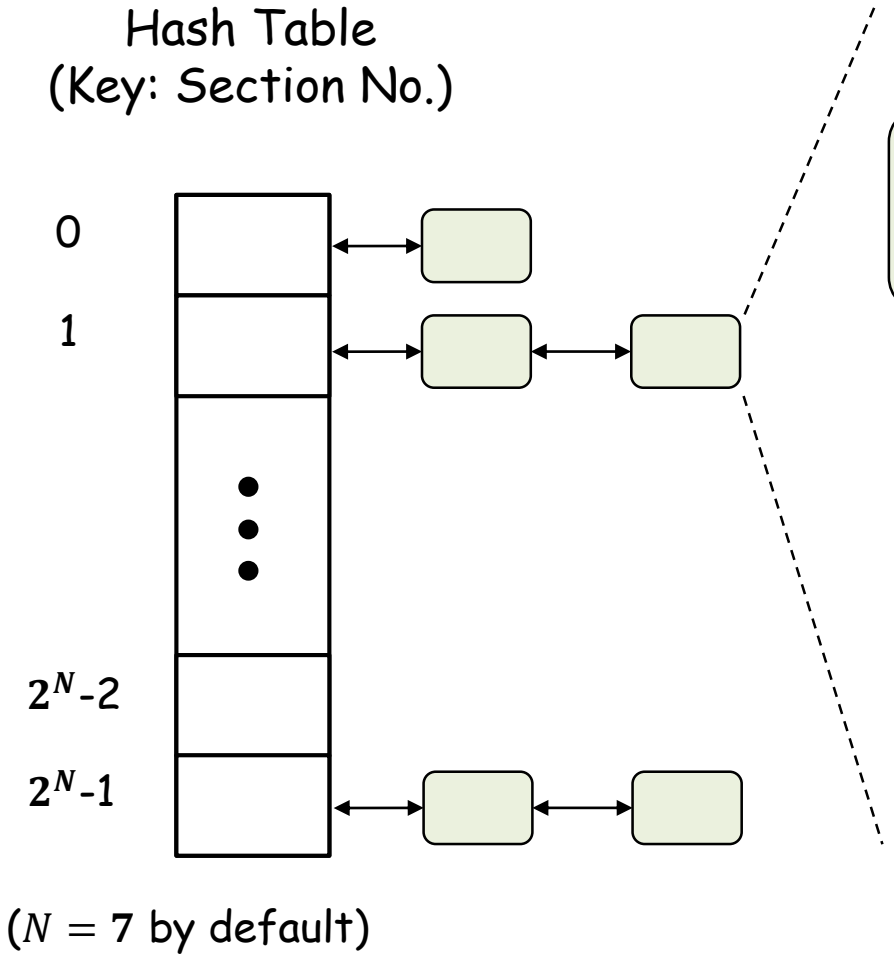
	Garbage Collection	File Mapping	Discard
~~Reverse Mapping~~	O		
File Index		O	
~~Block Bitmap~~	O		O

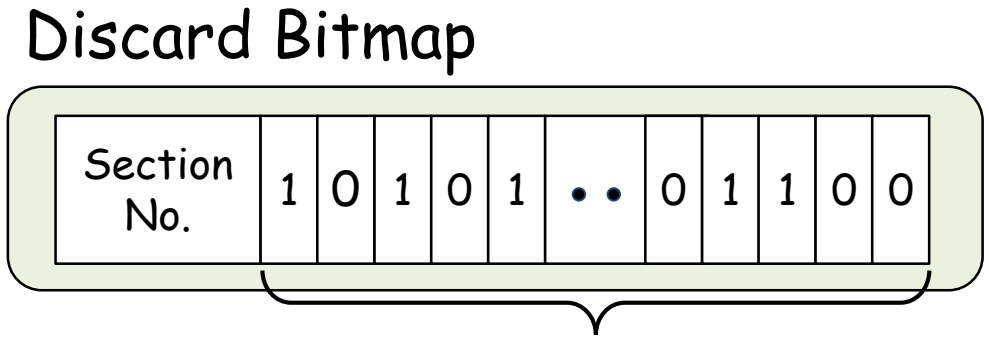
We need Metadata for discarding the filesystem blocks.

For creating the discard commands **Discard Bitmap** In-Memory.

For crash recovery of discard commands, **Discard Log** On-Disk.

Discard Bitmap





Discard Block Bitmap of a Section
(Section Size = 1 GByte by default)

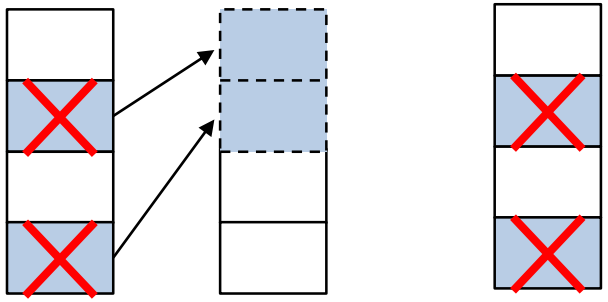
- ✓ Created if there's no discard bitmap for the incoming invalidated block.
- ✓ Deleted after transformed into the discard commands at the checkpoint.

Discard Mechanism of IPLFS

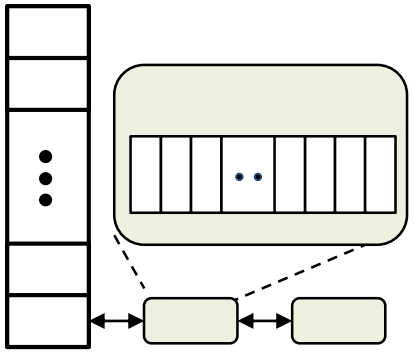
At file system operation

Garbage Collection

truncate, unlink

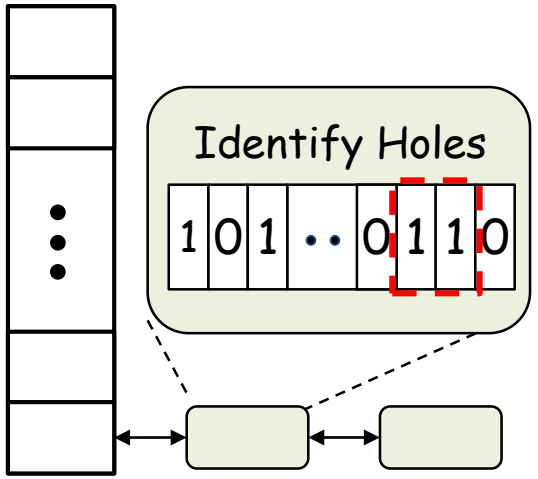


Invalidate the discard bitmap.



At checkpoint

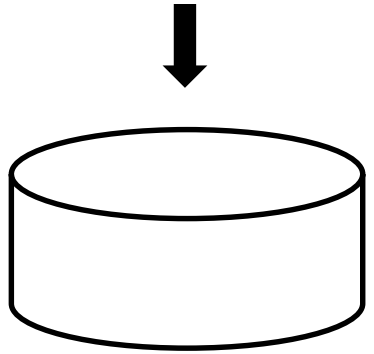
Scan the discard Bitmap
&
Create discard commands



Dispatch the discards!

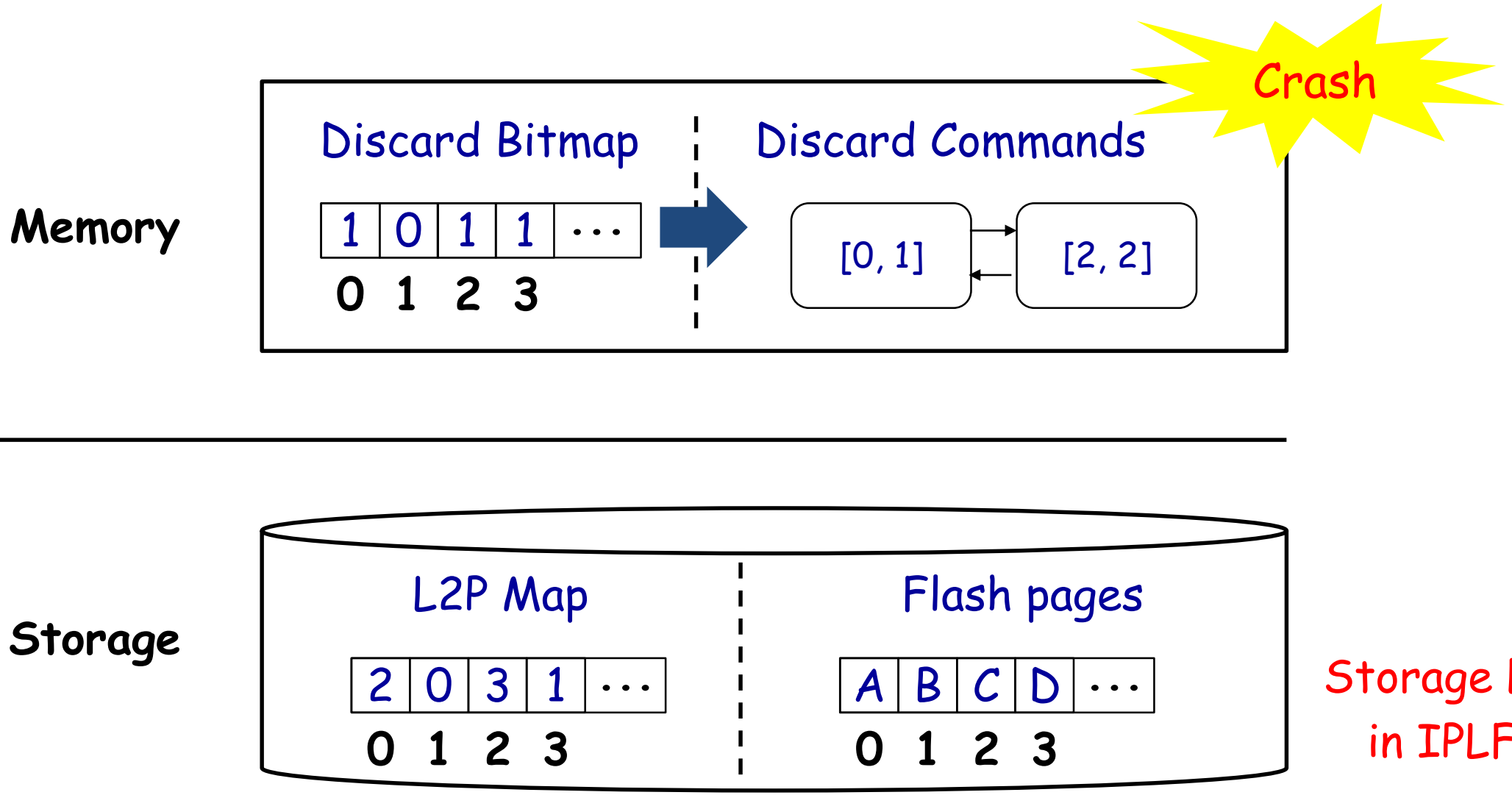
Discard Command

[Start LBA, Size]



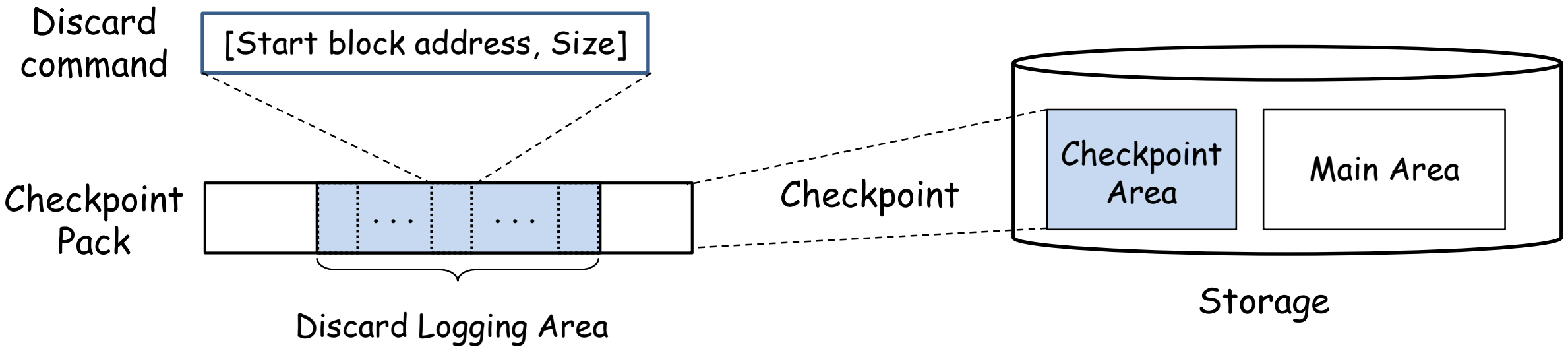
At 50 msec interval
at most 16 discards at a time
~~Only when IO is idle.~~

Discard bitmap with Power failure

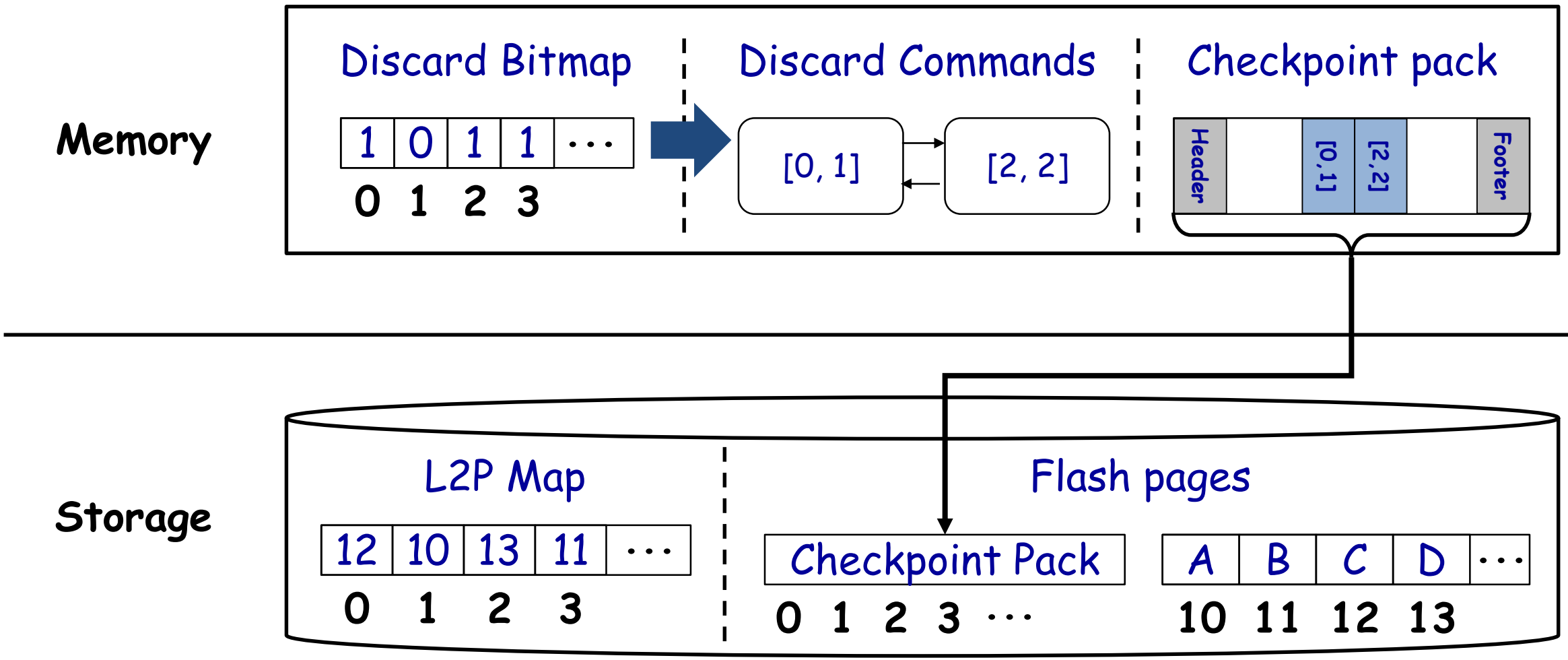


Discard Logging

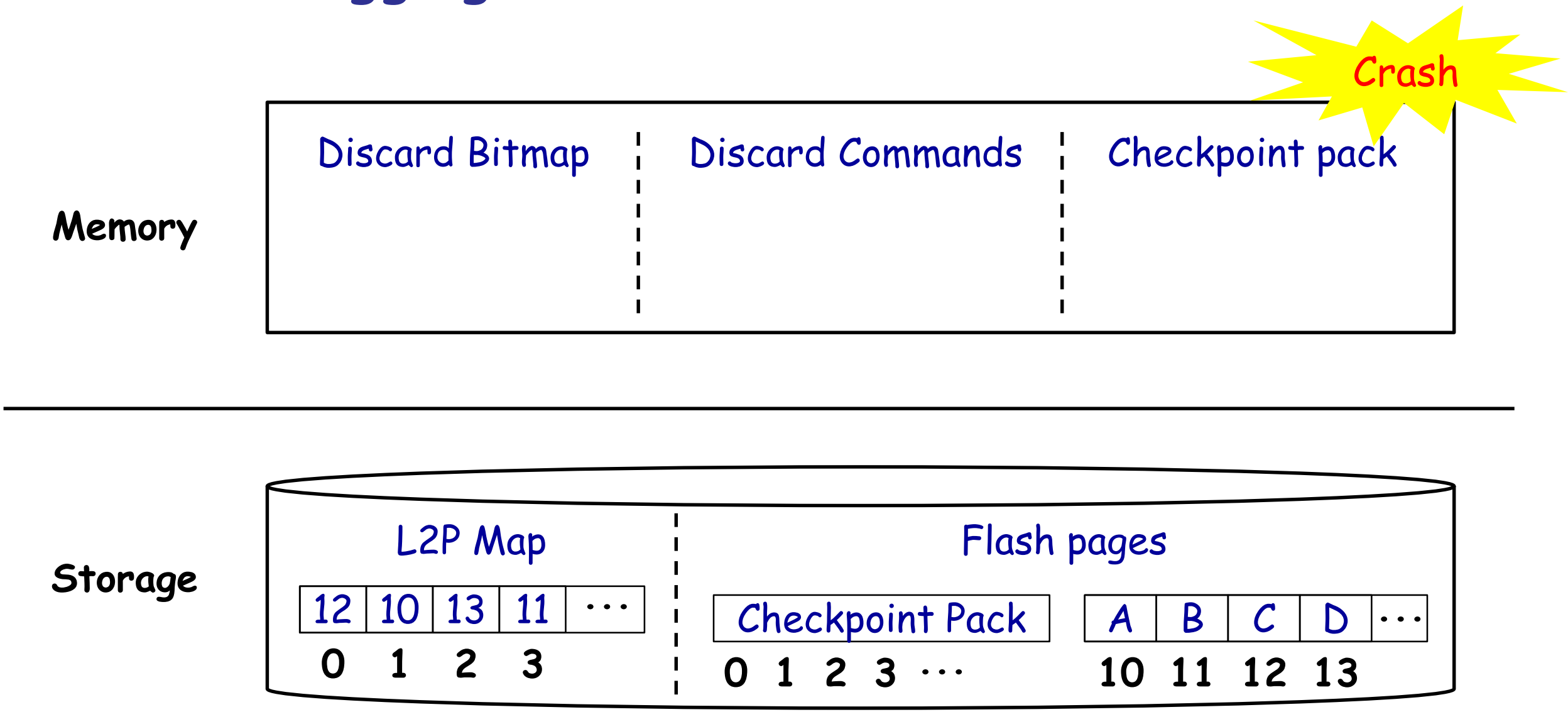
- ✓ Log the Discard commands into a Checkpoint Pack.
- ✓ Recover the Discard Commands from Checkpoint Pack



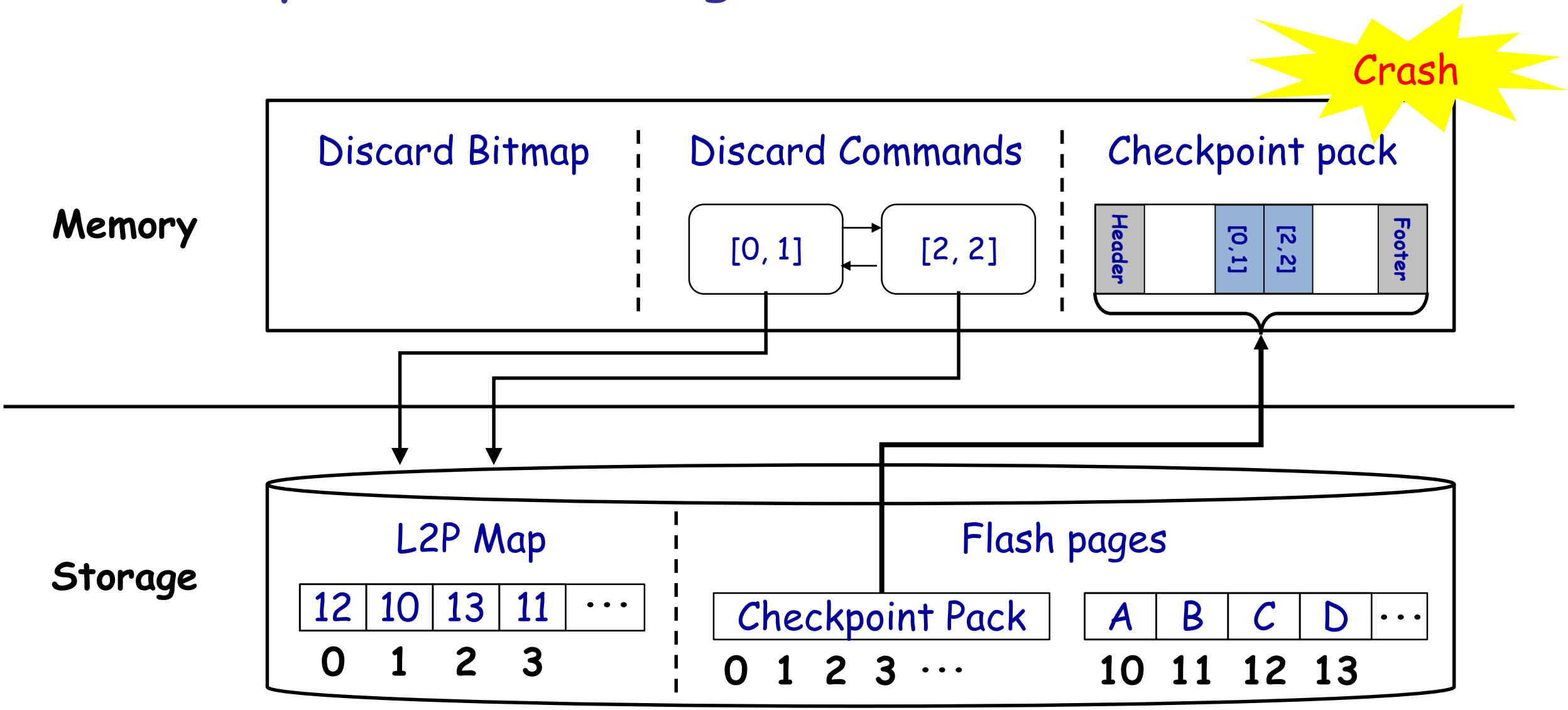
Discard Logging Mechanism



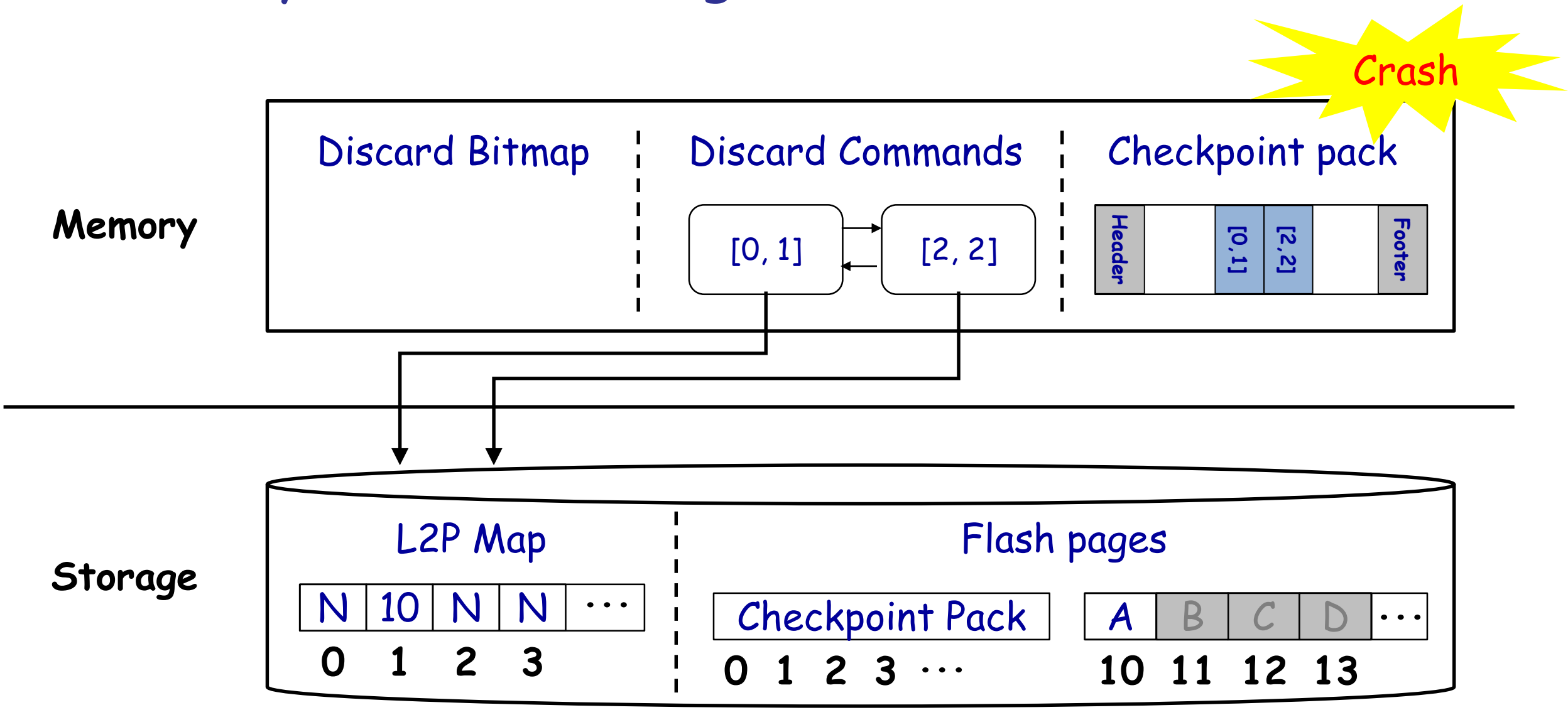
Discard Logging Mechanism with Power failure



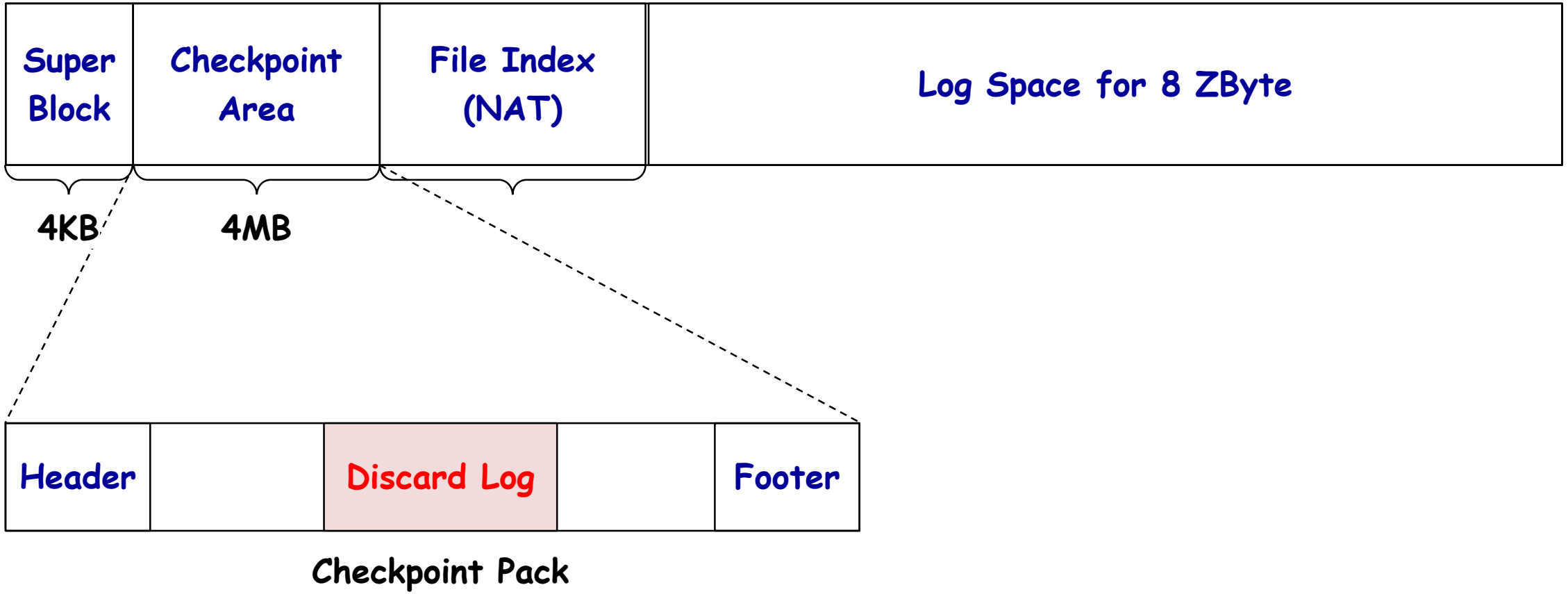
Recovery of Discard Log



Recovery of Discard Log



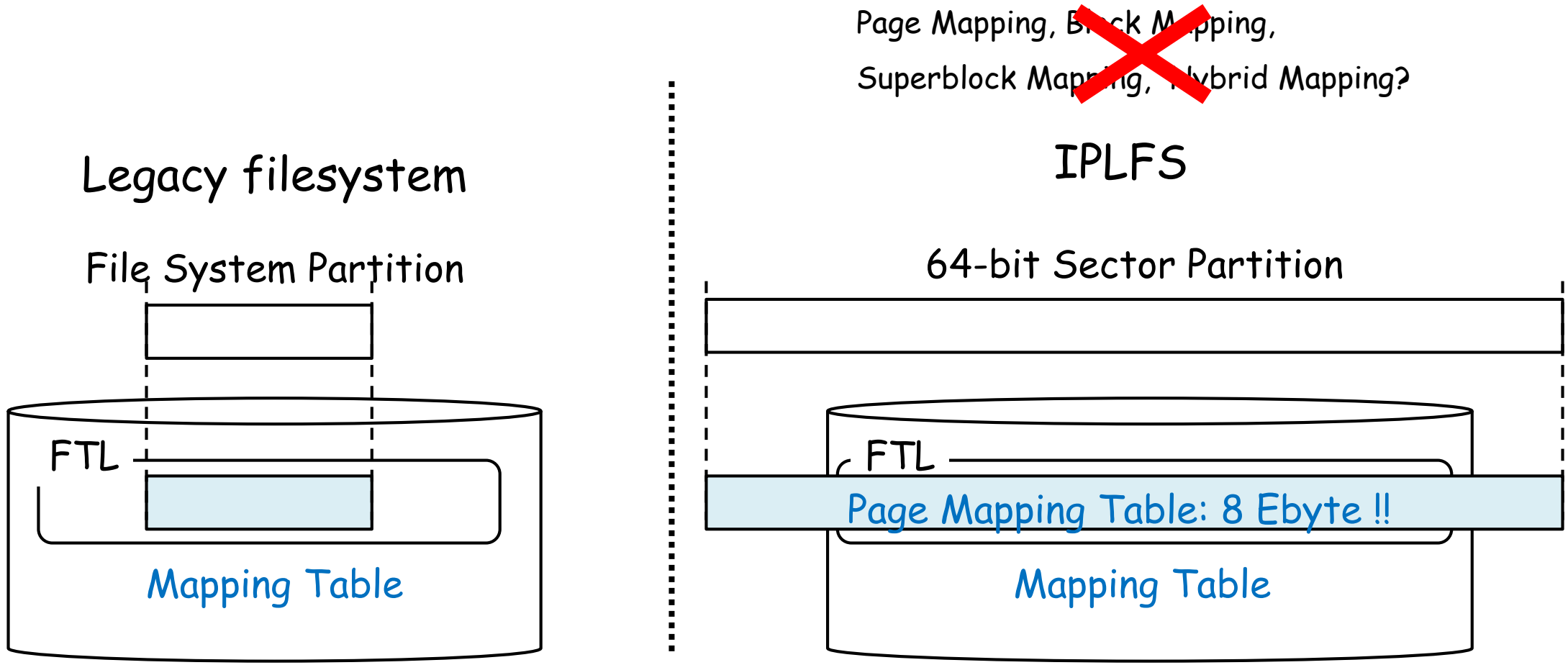
File system Layout of IPLFS



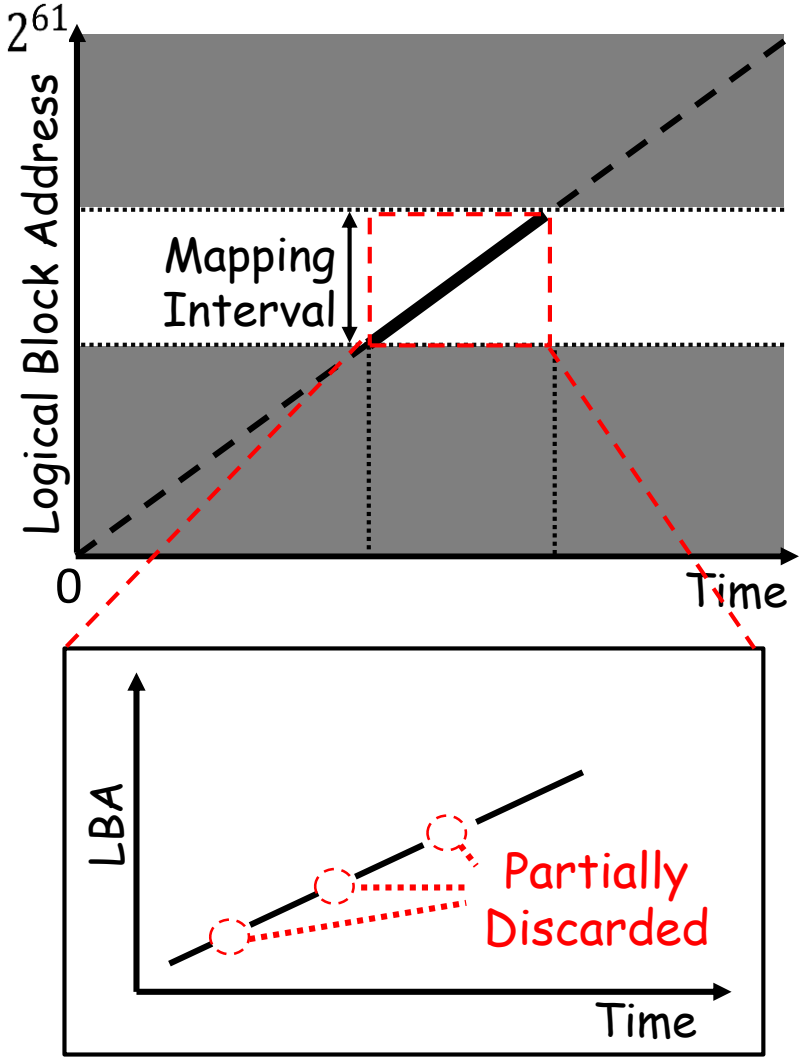
Interval Mapping

FTL Mapping Table For Infinite Partition?

Legacy Mapping Table of FTL can not support 64-bit Sector Partition.

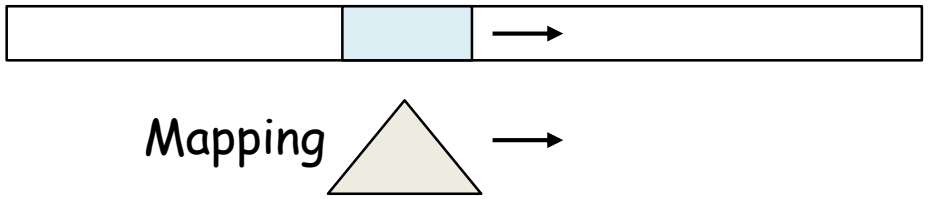


Interval Mapping



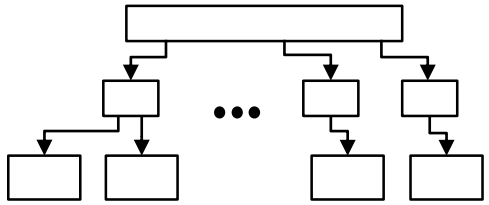
- Maintain mapping only for actively used region in the logical partition.

Mapping Interval

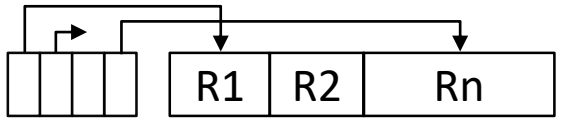


- Within the actively used region, exclude the mappings for invalidated LBA's as much as possible.

Tree Based Mapping Structure

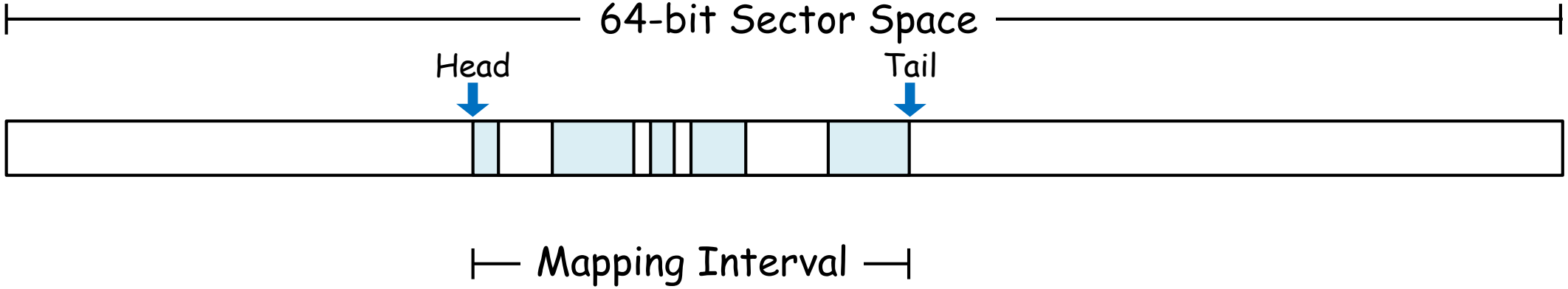


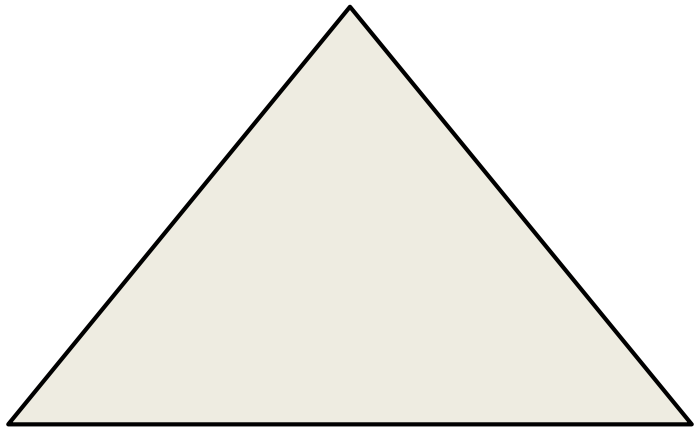
Fixed Region Mapping for leaf node



Moving Mapping Interval

- ✓ Mapping Interval: From the First Valid Mapping To the Last Valid Mapping.

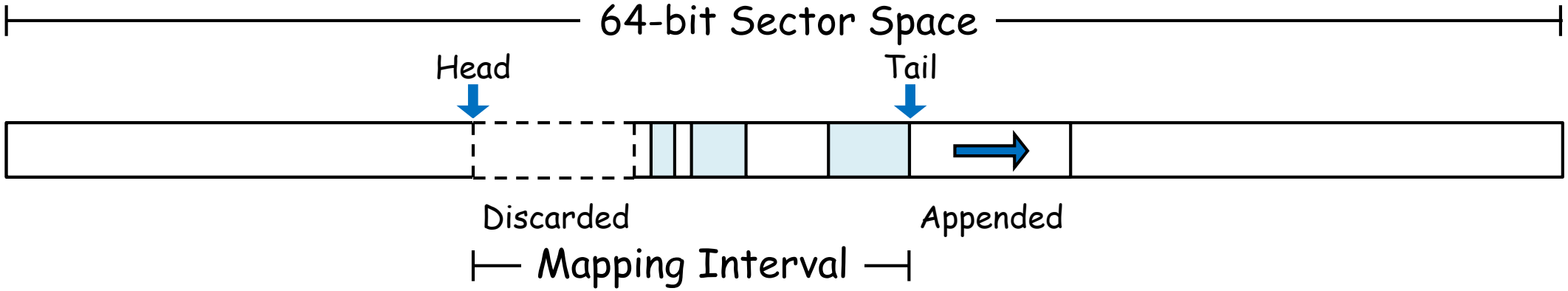


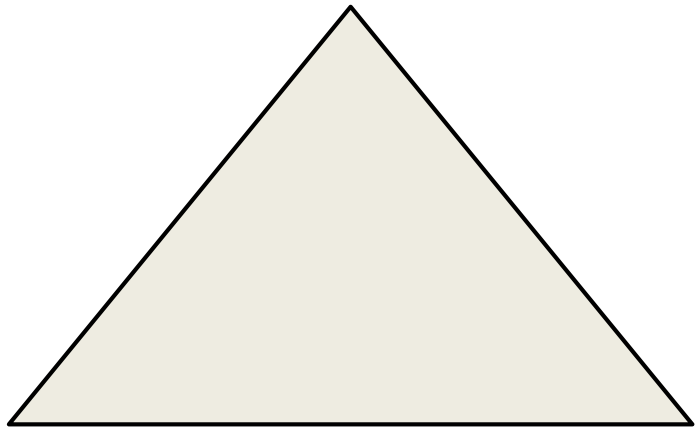


Interval Mapping Tree
- Map Only Actively Used Region in the logical Partition

Moving Mapping Interval

✓ Discard & Append change the Head and the Tail of Mapping Interval

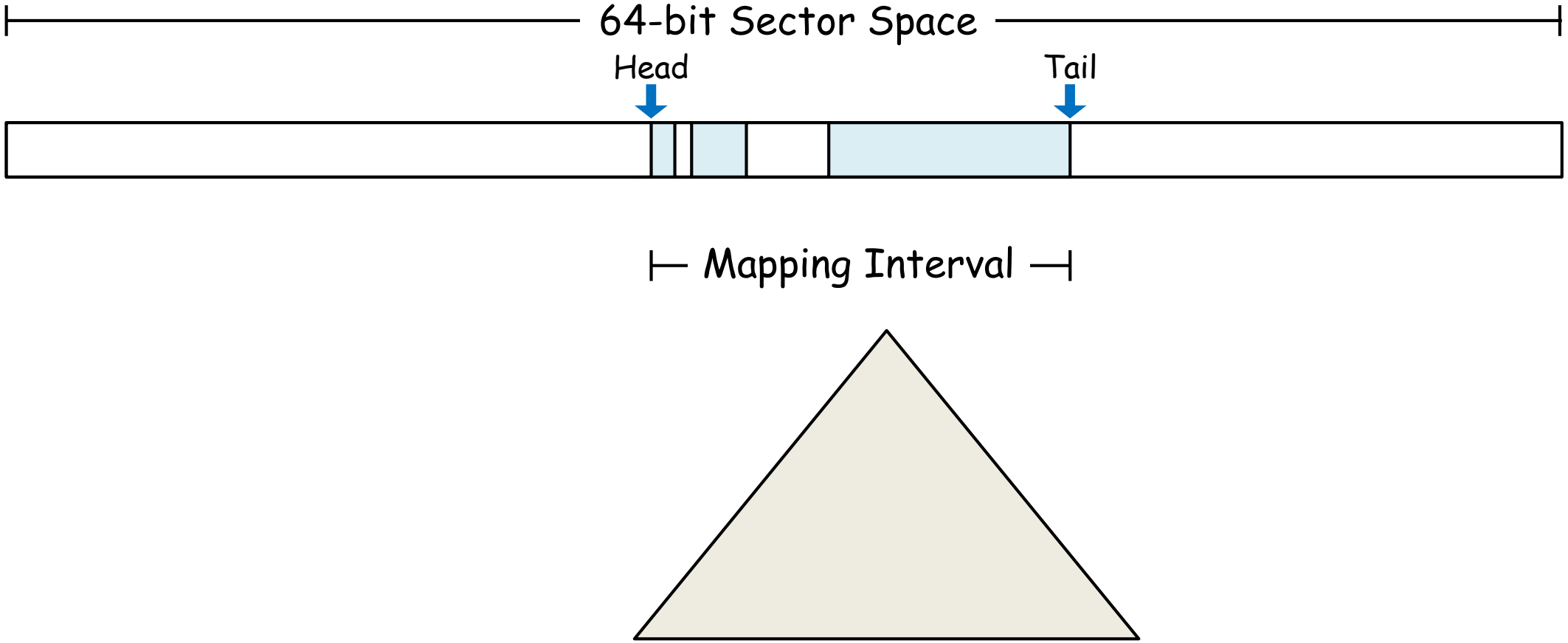




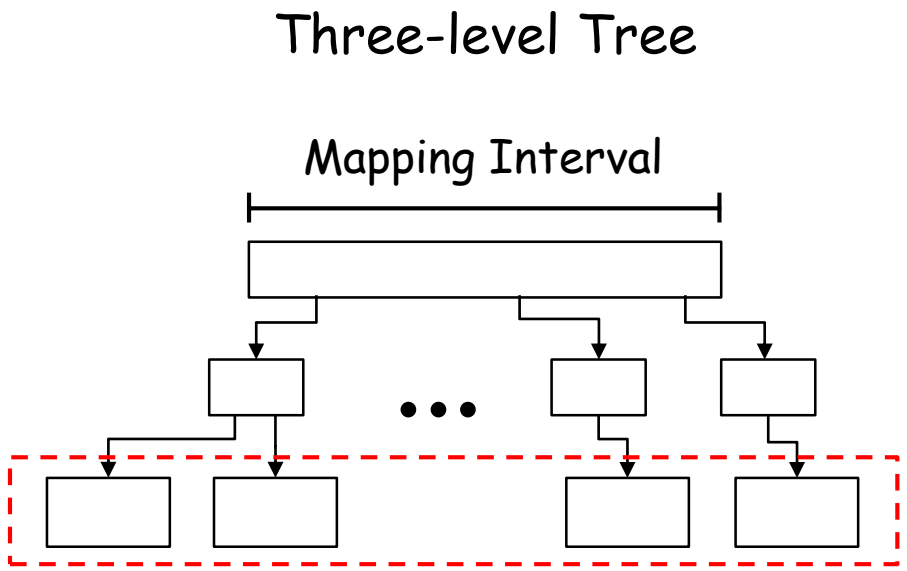
Interval Mapping Tree
- Map Only Actively Used Region in the logical Partition

Moving Mapping Interval

- ✓ Mapping Interval moves to the higher Logical Block Address.



Structure of Interval Mapping



1st level: Root node

2nd level: Zone node

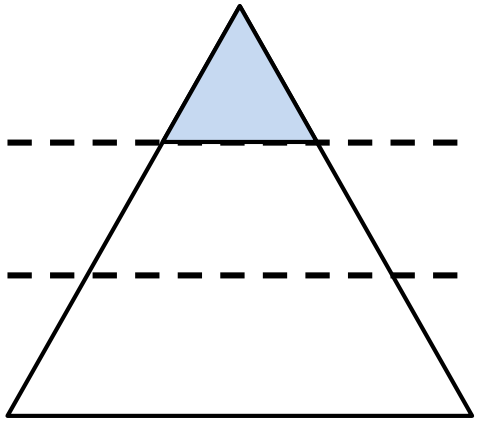
3rd level: Map node

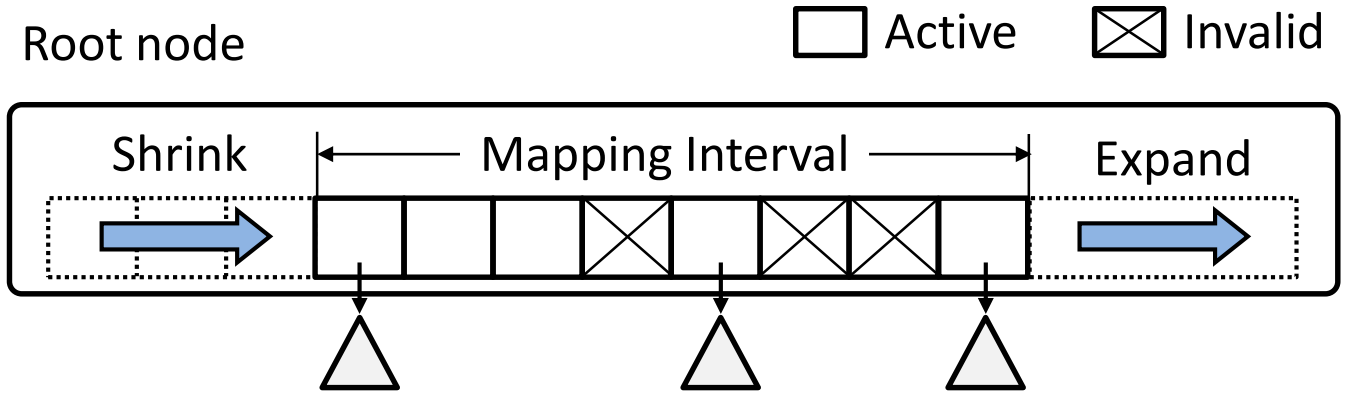
Total size of map nodes accounts for 99.8% of Tree Size

Root Node

- Shrinks and Expands to represent the Mapping Interval.
- Maintains the child node pointers in the Mapping Interval. (Child Node = Zone Node)

Interval Mapping Tree

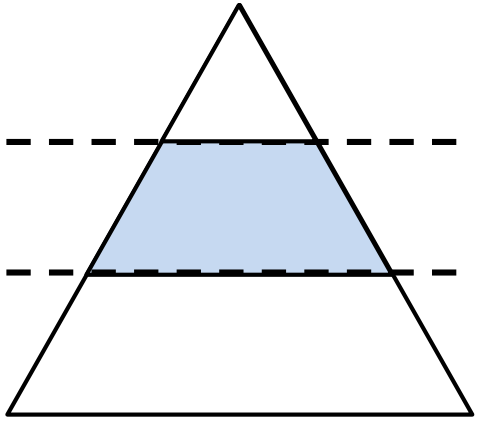


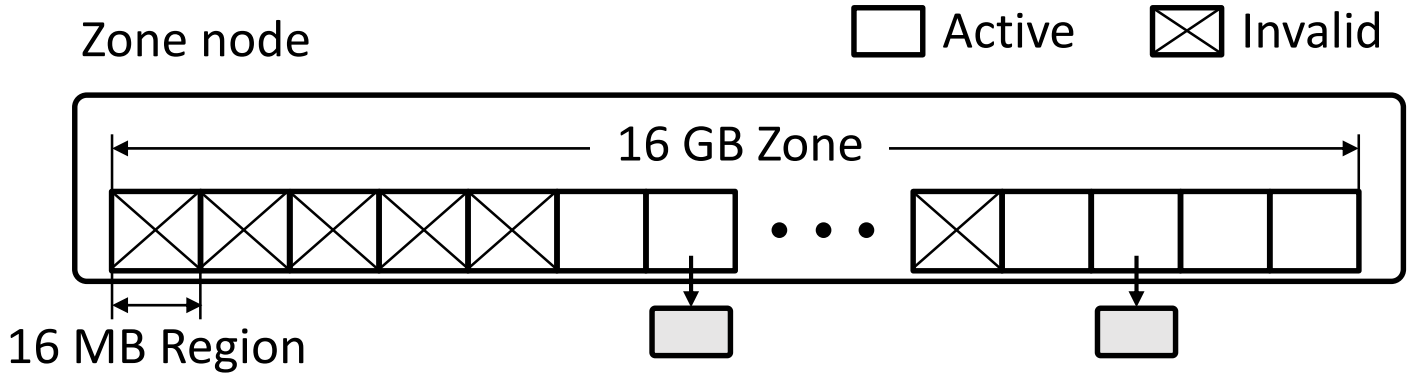


Zone Node

- Maintains the pointers to the map nodes. (Child Node = Map Node)
- Zone Size = 16 Gbyte

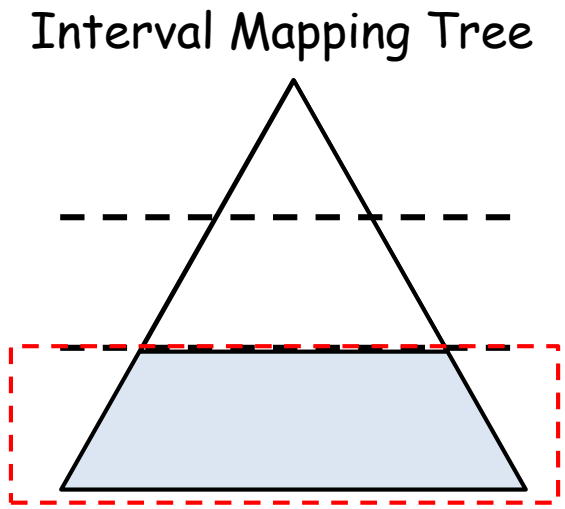
Interval Mapping Tree

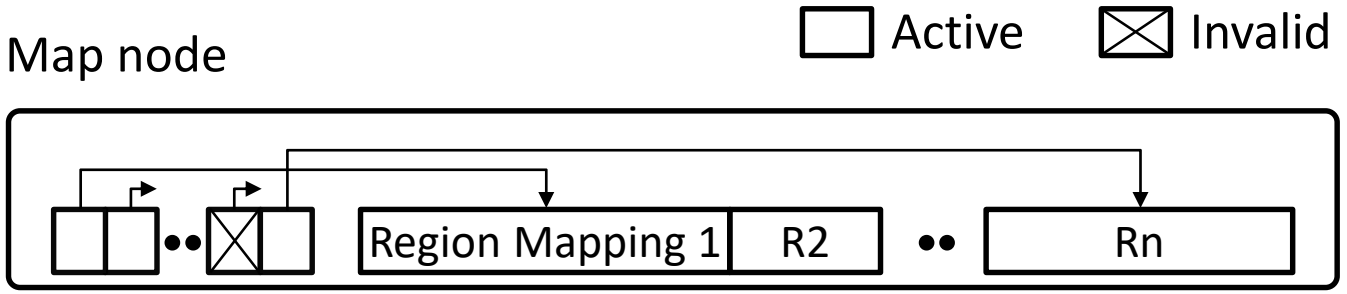




Map Node

- Maintains LBA-to-PBA Page Mappings for a Region (Region Size = 16 MByte)
- "Fixed-Region Mapping" to save Memory Footprint

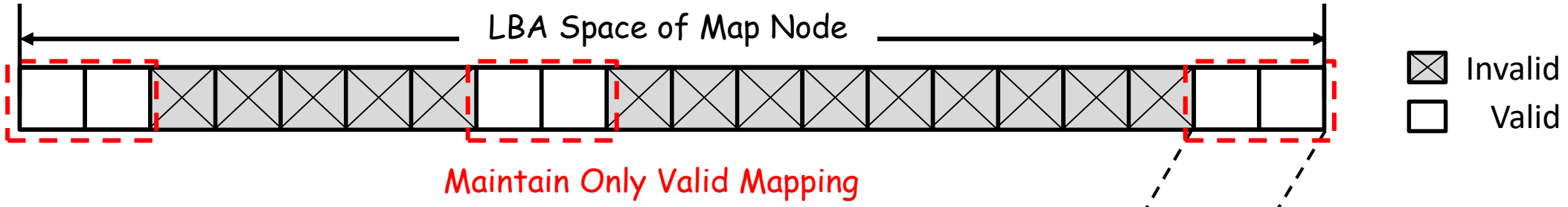




Total size of map nodes accounts for 99.8% of Tree Size

Fixed Region Mapping for Map node

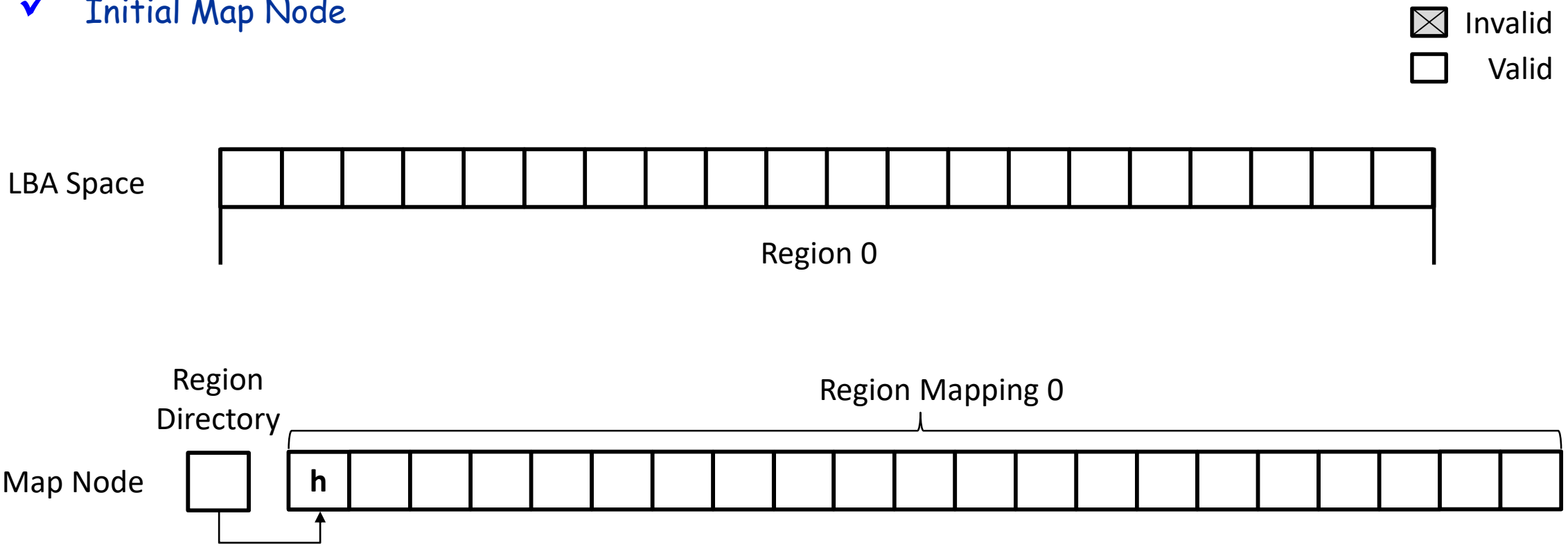
- ✓ How to map only valid LBA's?



- ✓ Fixed-Region Mapping

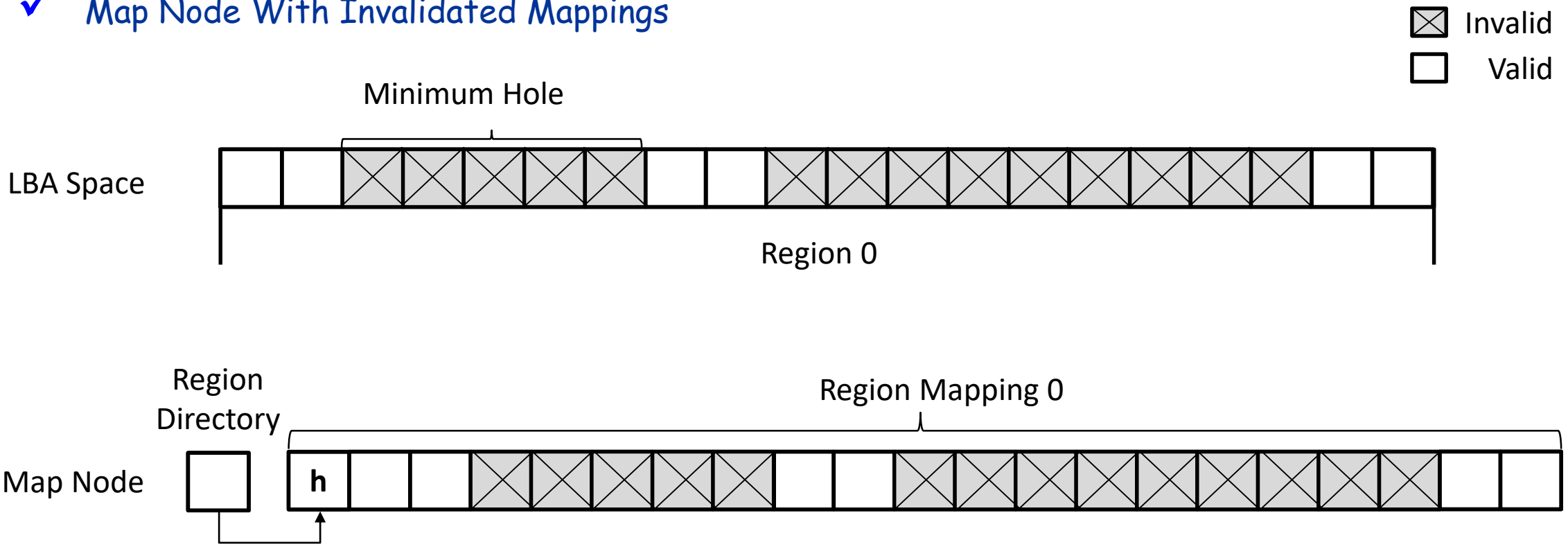
Map Node Compaction

✓ Initial Map Node



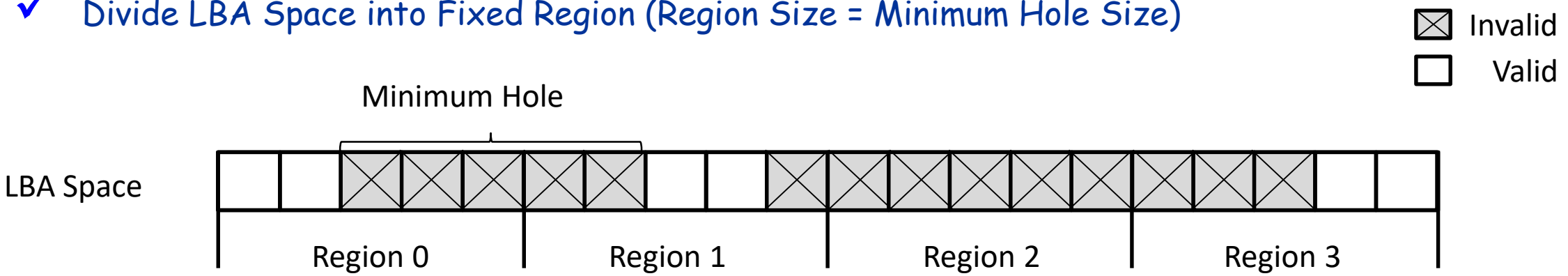
Map Node Compaction

✓ Map Node With Invalidated Mappings



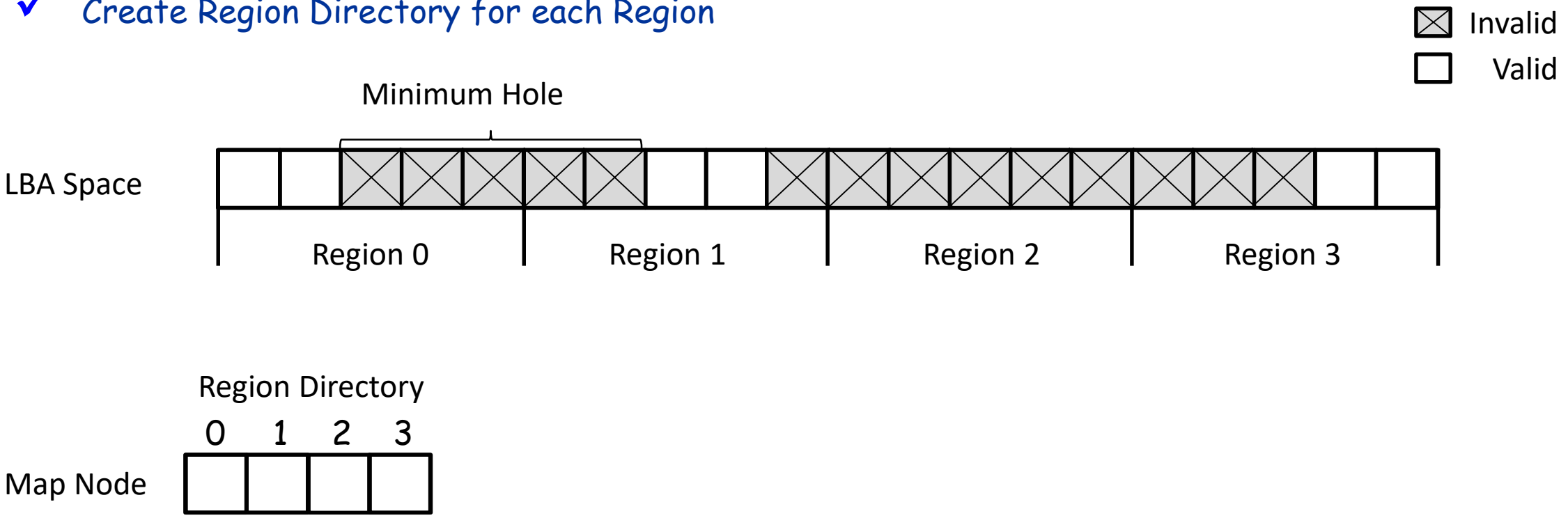
Map Node Compaction

- ✓ Divide LBA Space into Fixed Region (Region Size = Minimum Hole Size)



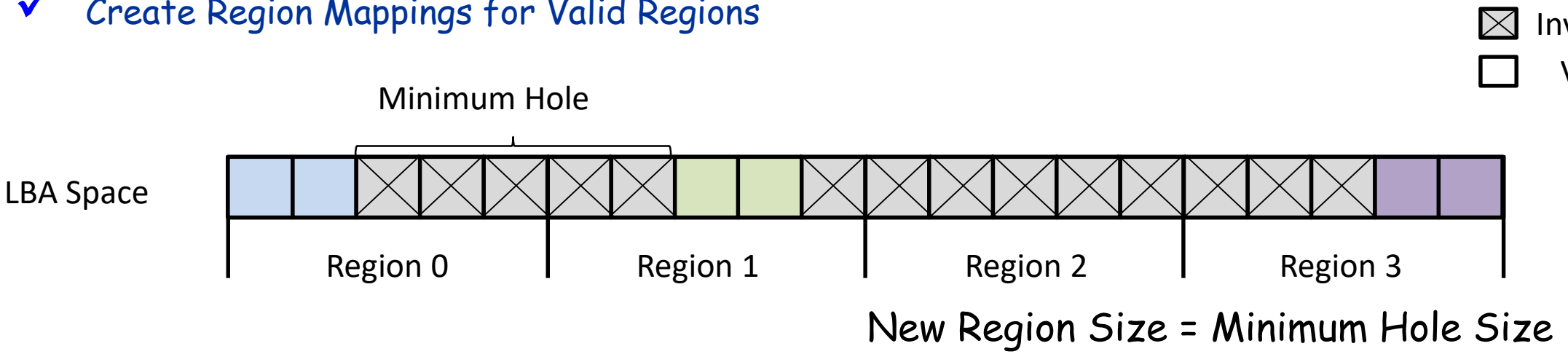
Map Node Compaction

- ✓ Create Region Directory for each Region



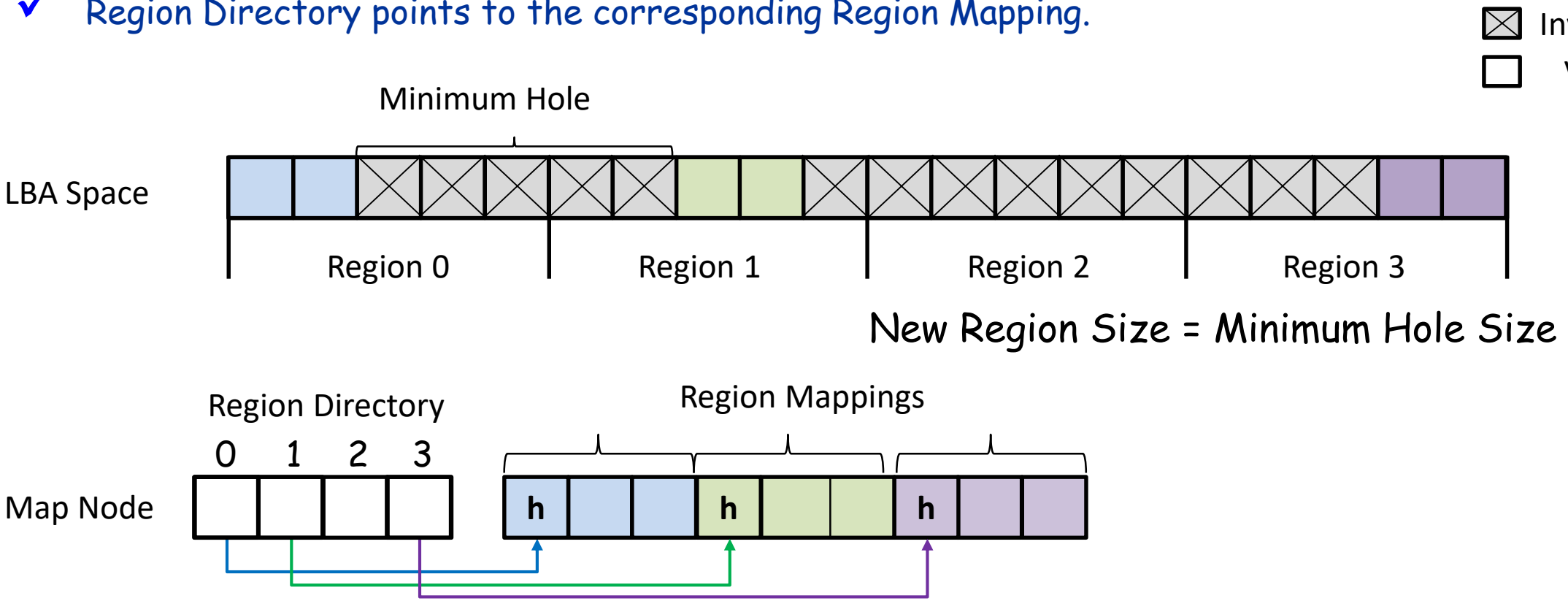
Map Node Compaction

✓ Create Region Mappings for Valid Regions



Map Node Compaction

✓ Region Directory points to the corresponding Region Mapping.



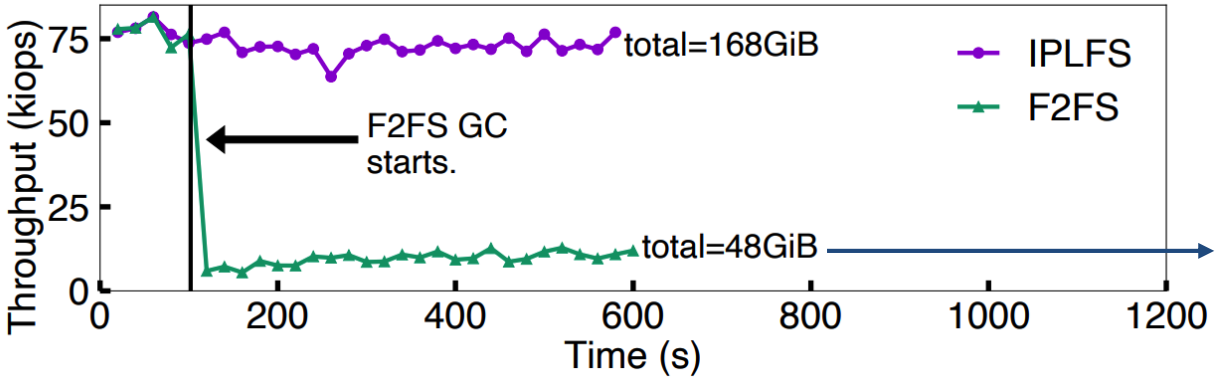
Evaluation

Evaluation

- ✓ HW Setup
  - CPU: Intel CPU i7-4770K (3.50GHz, 4core)
  - Memory: 8GB
  - OS: Linux 5.11.0
  - Storage: Cosmos+ OpenSSD 230GByte with 8 channels
- ✓ Workloads
  - [Fio] Garbage Collection Elimination
  - [Fileserver in Filebench] Map Node Compactoin

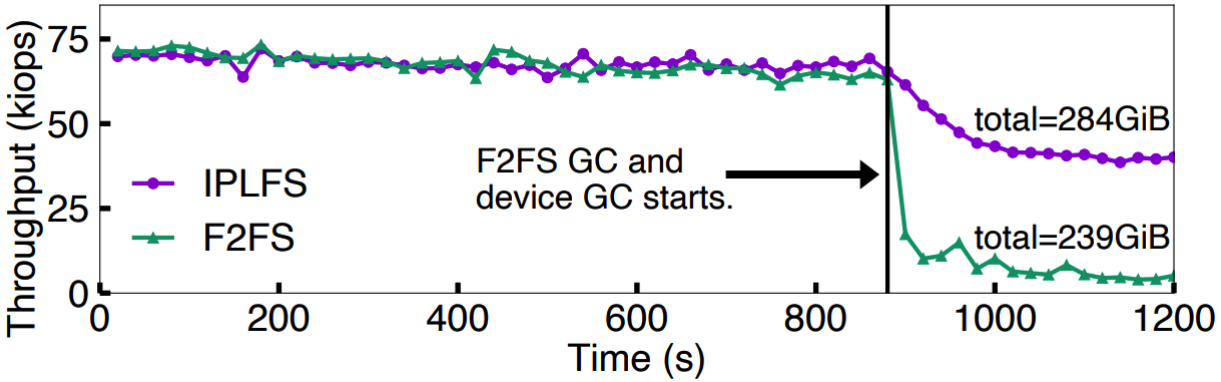
Garbage Collection Elimination

- ✓ Only Filesystem GC case: 28GB file random write on 30GB partition



Perf. drops to 1/10

- ✓ Filesystem & FTL GC case: 210GB file random write on 230GB partition

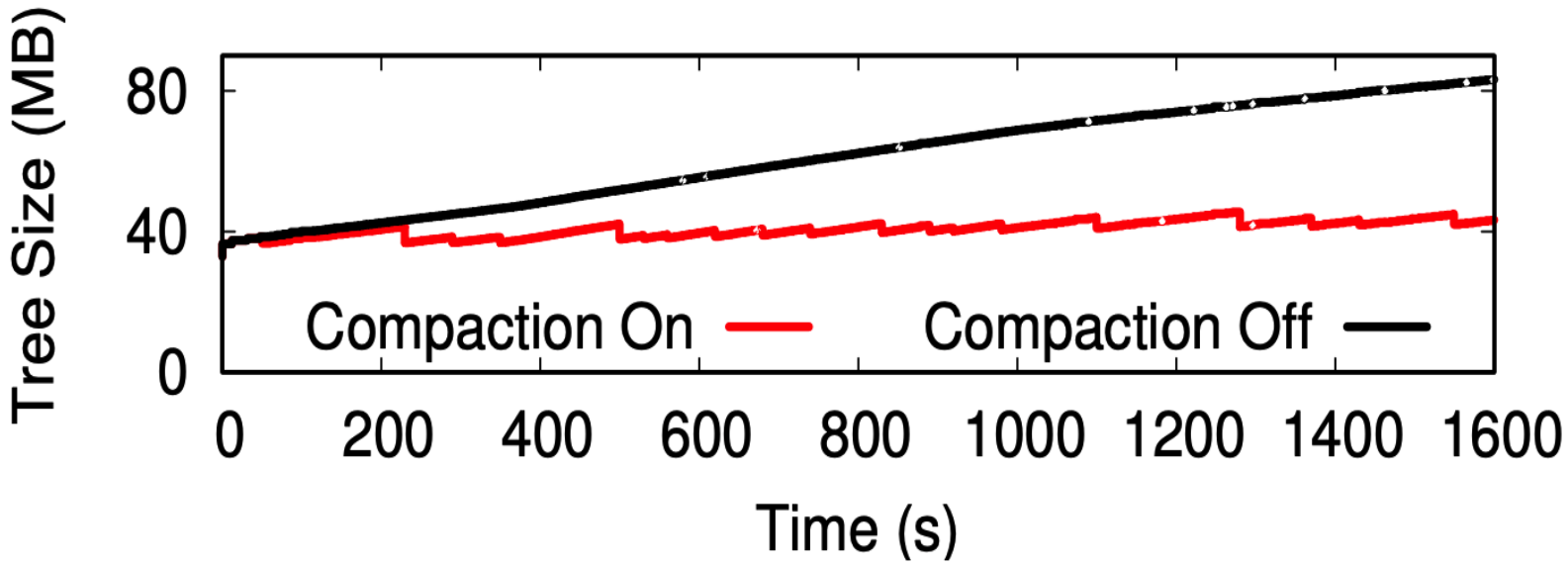


FTL GC: perf. drops to 40%

FS GC: perf. drops to 1/10

Map Node Compaction

- ✓ Mapping Tree Size vs. Time (File size 160GB, Compaction period 30s, threshold 0.7)



Steady around 40MB

Conclusion

- ✓ Disk-sized logical partition is the main cause of garbage collection in LFS
- ✓ We developed IPLFS, a log-structured filesystem for infinite partition
  - Filesystem is free from Garbage Collection
  - Interval Mapping maintains logical-to-physical mapping for actively used filesystem region.
- ✓ IPLFS outperforms F2FS by up to 12.8x (FIO).

Question & Answer