FAST'25
Feb. 25-27, 2025
SANTA CLARA, CA

D2FS: Device-Driven Filesystem Garbage Collection

Juwon Kim[†] Seungjae Lee[†] Joontaek Oh[†‡] Dongkun Shin[*] Youjip Won[†]

[†] KAIST
[‡] University of Wisconsin-Madison
[*] Sungkyunkwan University

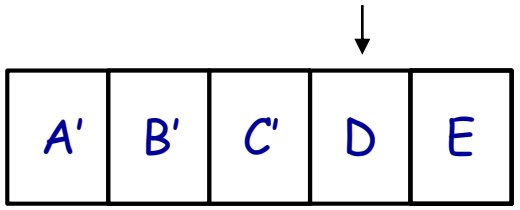
Background & Motivation

Log-Structured Filesystem

Rosenblum et al. (SOSP '91) and Margo Seltzer et al. (USENIX Winter '93)

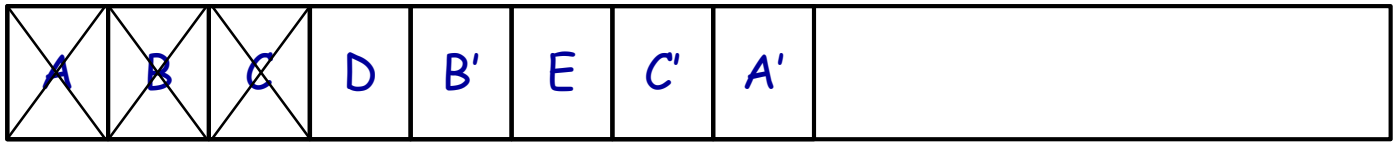
- Write Optimized Filesystem.
- Append only nature → Efficiently handle random write workload.

Random Writes on a file.



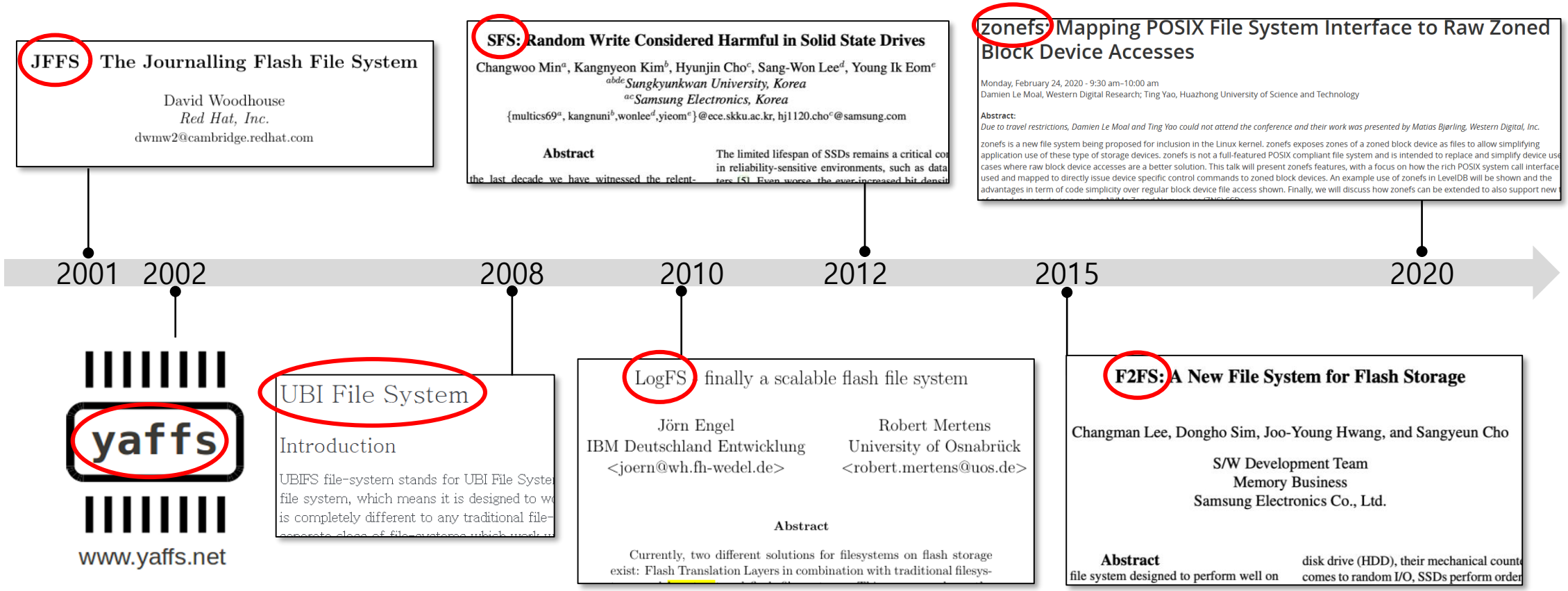

```
update (B') ; // B → B'
write (E) ;   // new block
update (C') ; // C → C'
update (A') ; // A → A'
```

Sequential Write
on Filesystem Partition.

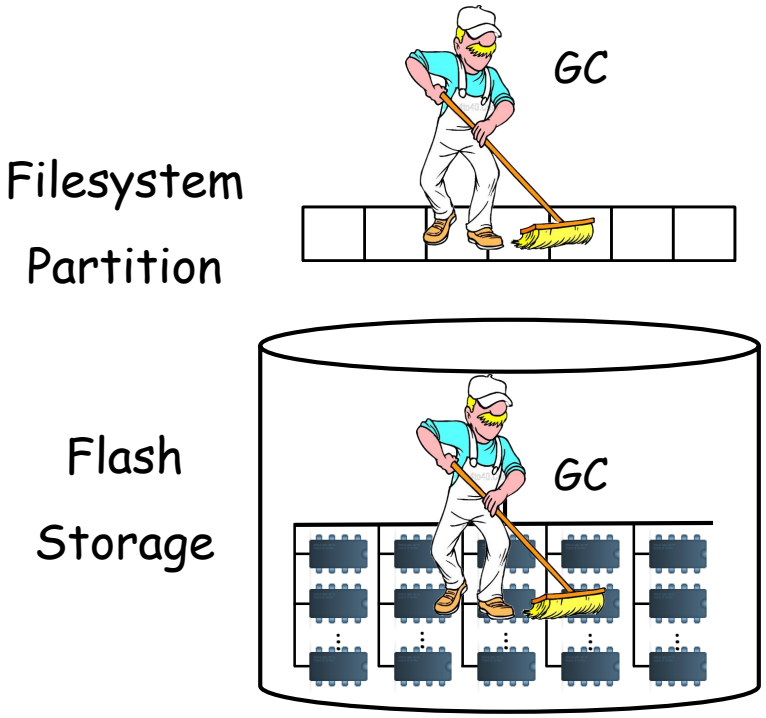


Log-Structured Filesystem for Flash Storage

- NAND flash requires sequential page writes. → fits append-only nature of LFS.



Duplicate Garbage Collection



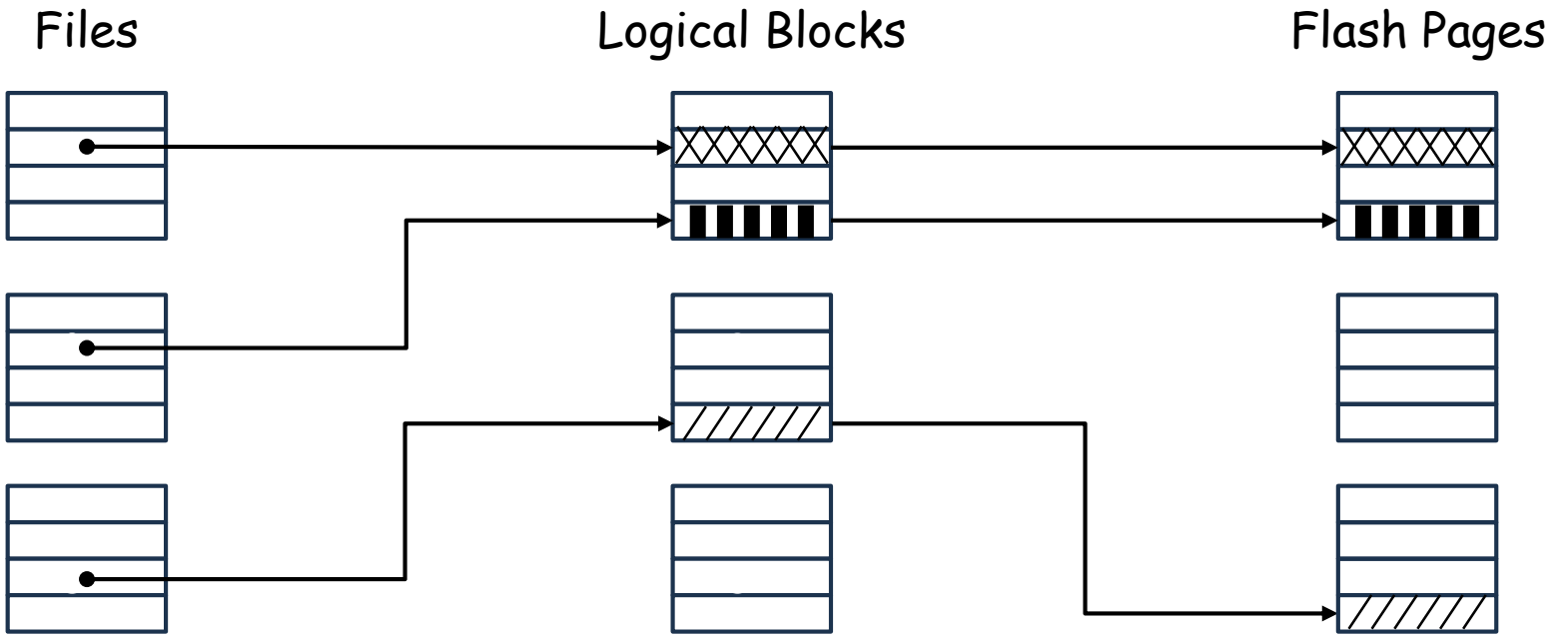
Reclaim the filesystem segments.

Reclaim the flash blocks.

Two garbage collections run redundantly.

WAF ↑ GC Overhead ↑

Mappings in I/O Stack

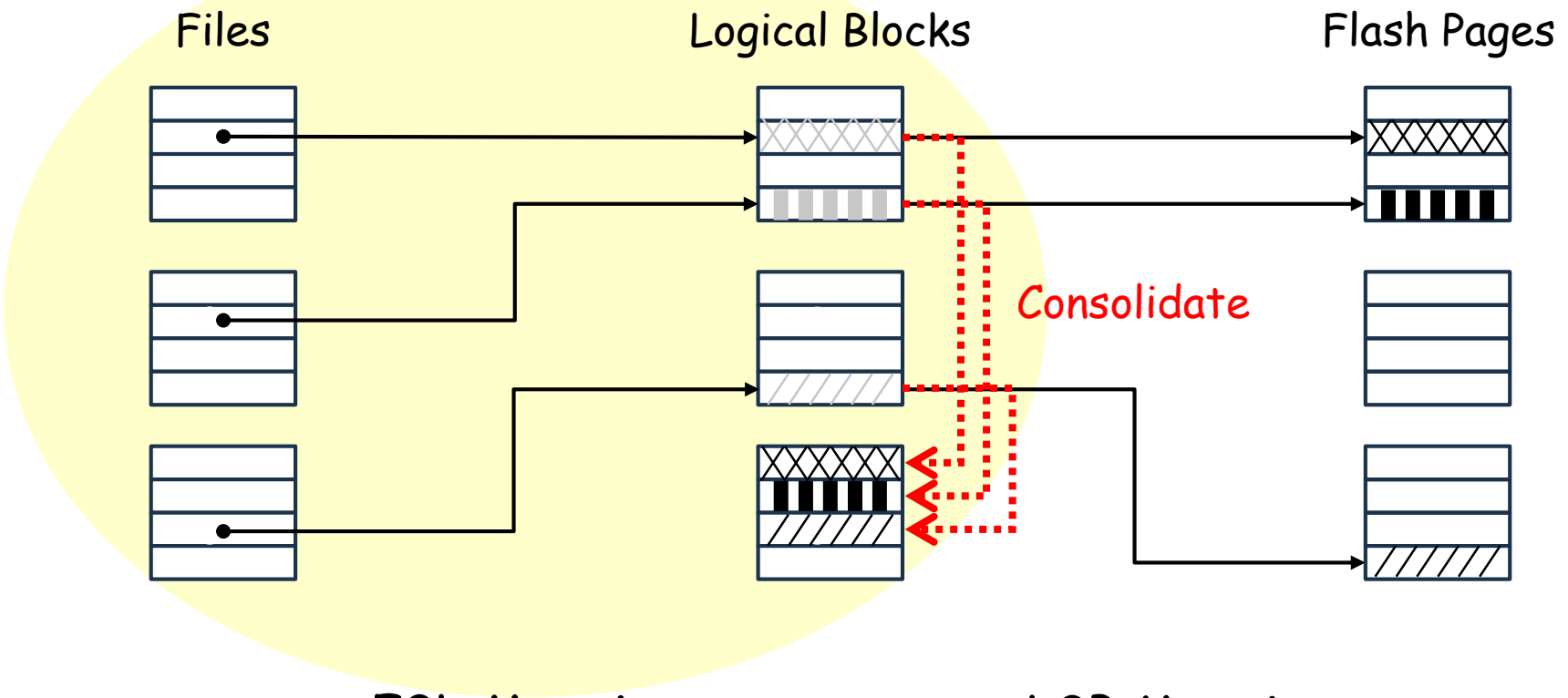


F2L Mapping
[fd,offset] → LBA

L2P Mapping
LBA → PPA

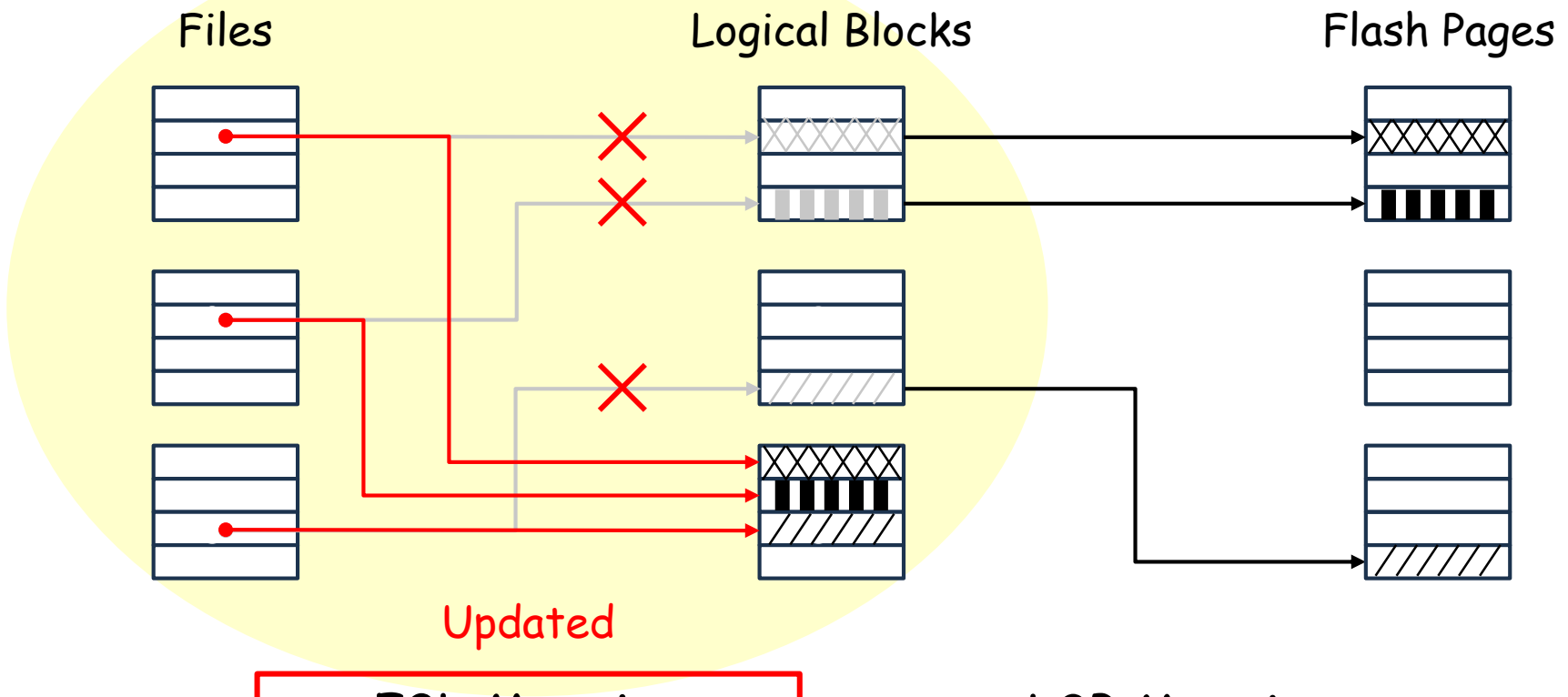
Garbage Collection in Filesystem

Reclaim invalid filesystem blocks.



Garbage Collection in Filesystem

Reclaim invalid filesystem blocks.

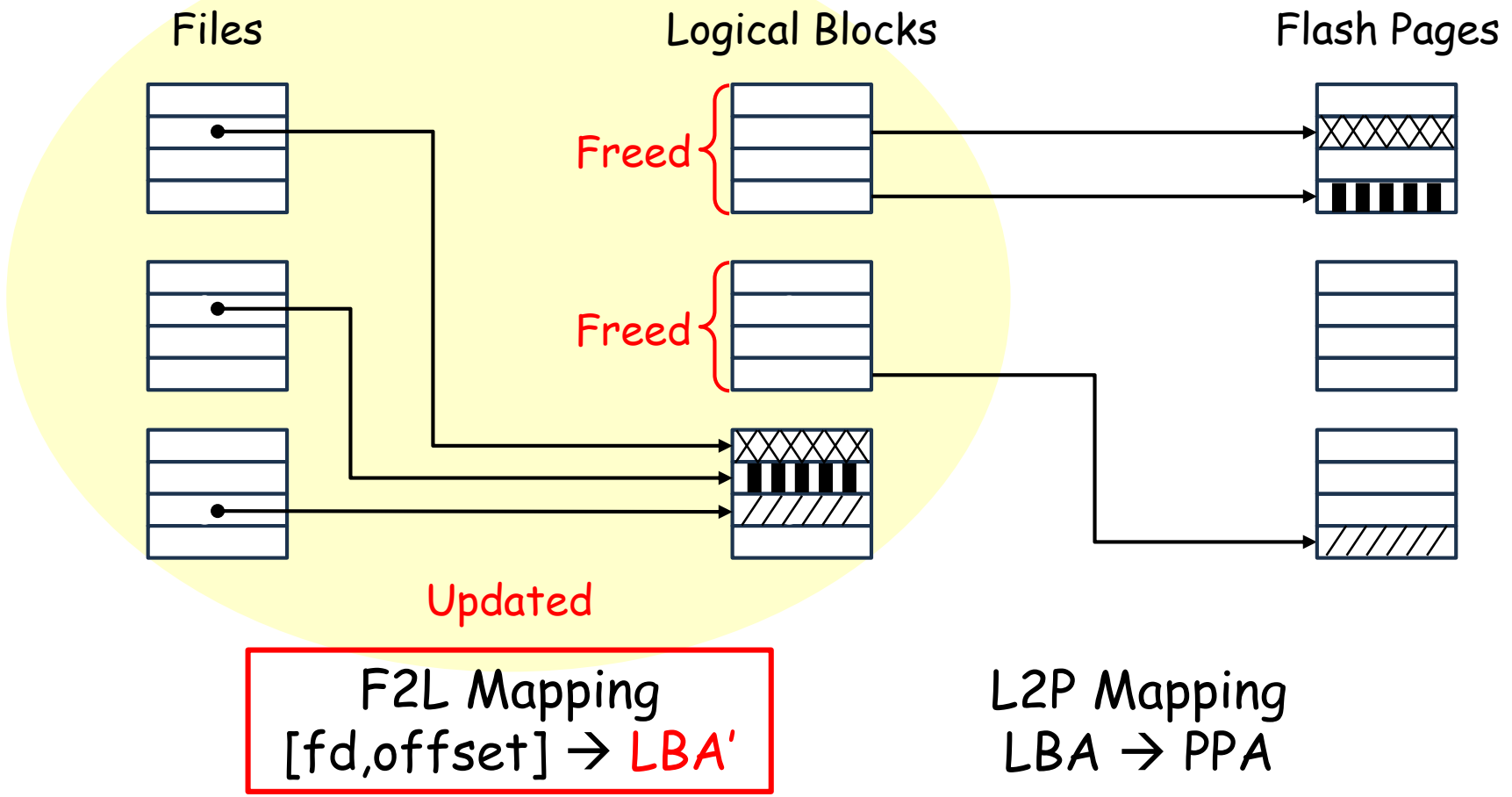


F2L Mapping
[fd,offset] → LBA'

L2P Mapping
LBA → PPA

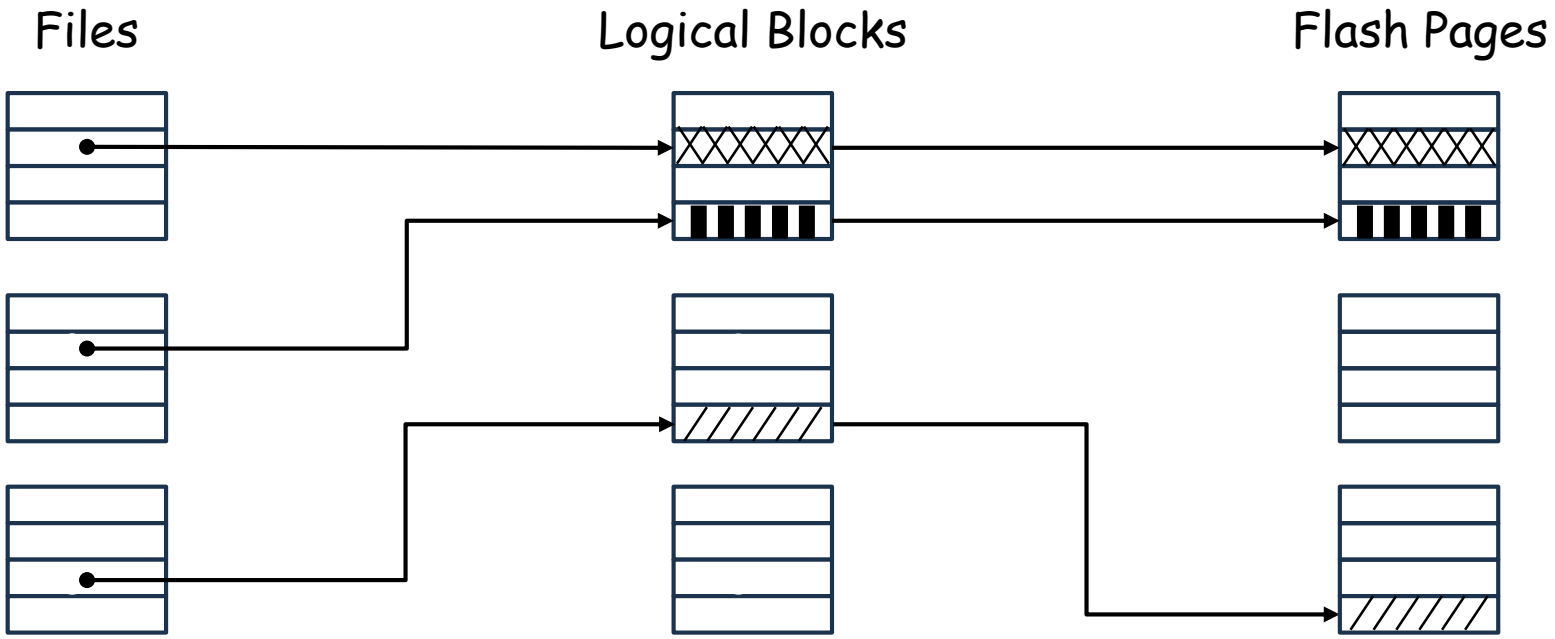
Garbage Collection in Filesystem

Reclaim invalid filesystem blocks.



Garbage Collection in Flash Storage

Reclaim invalid flash pages.

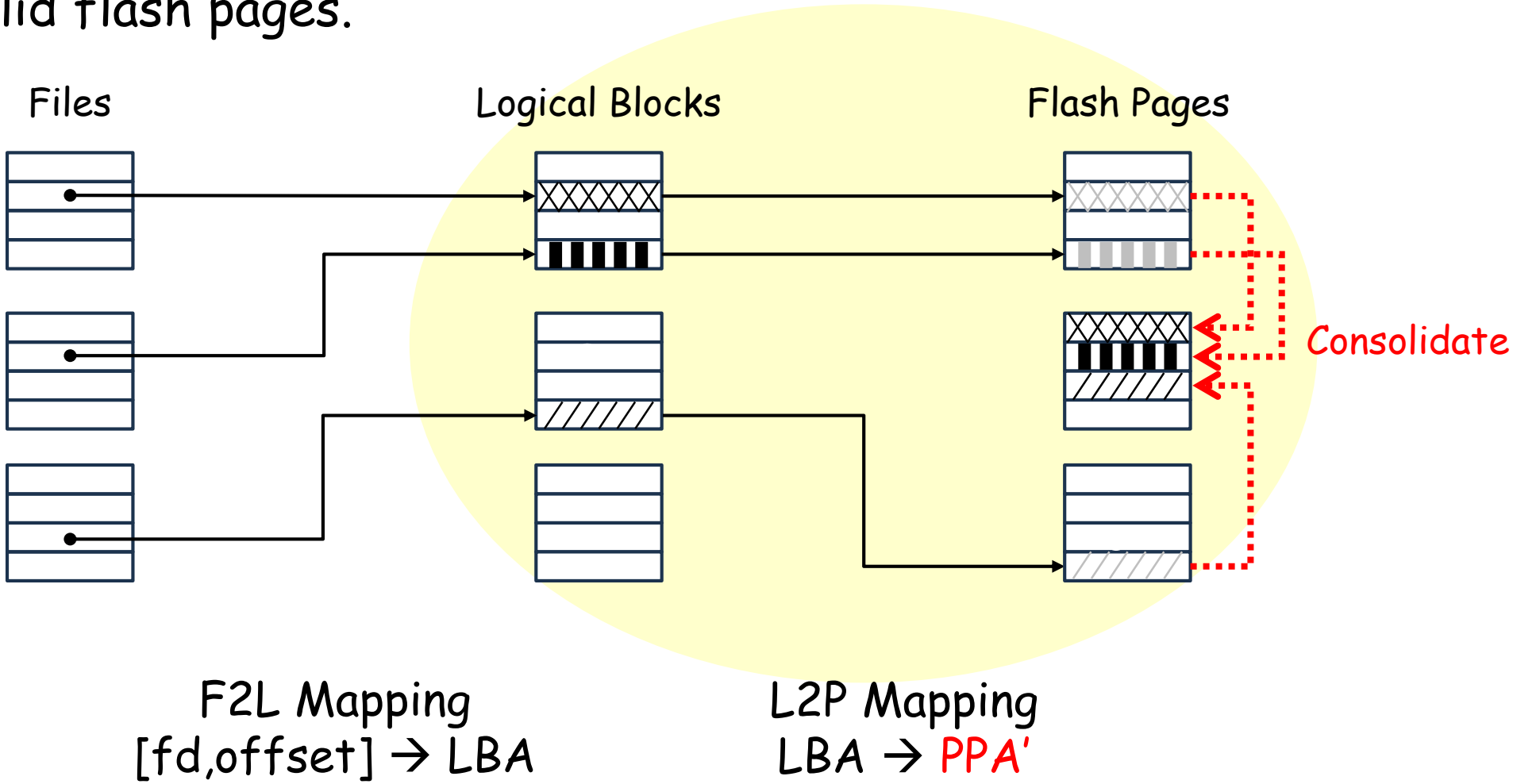


F2L Mapping
[fd,offset] → LBA

L2P Mapping
LBA → PPA

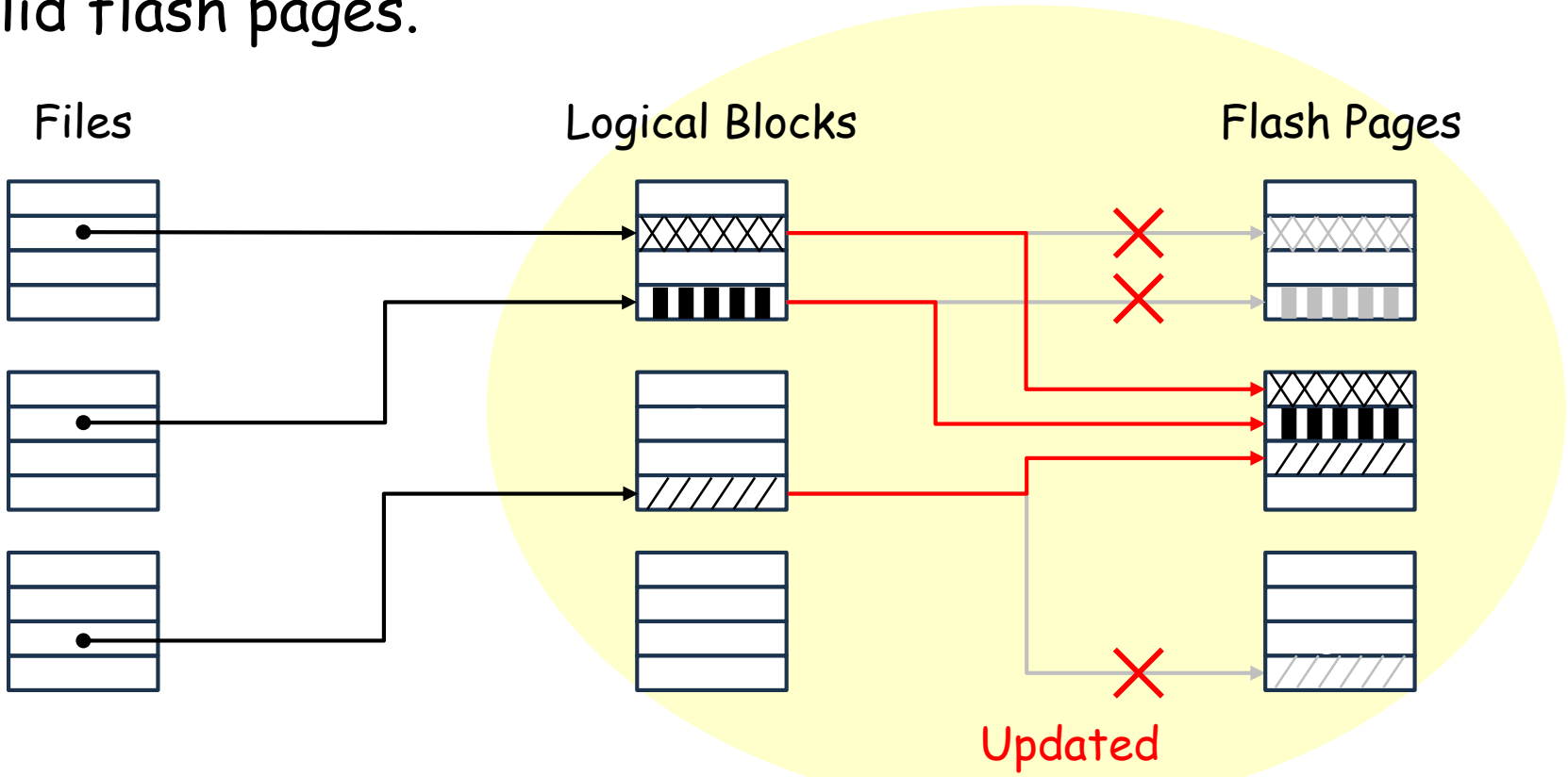
Garbage Collection in Flash Storage

Reclaim invalid flash pages.



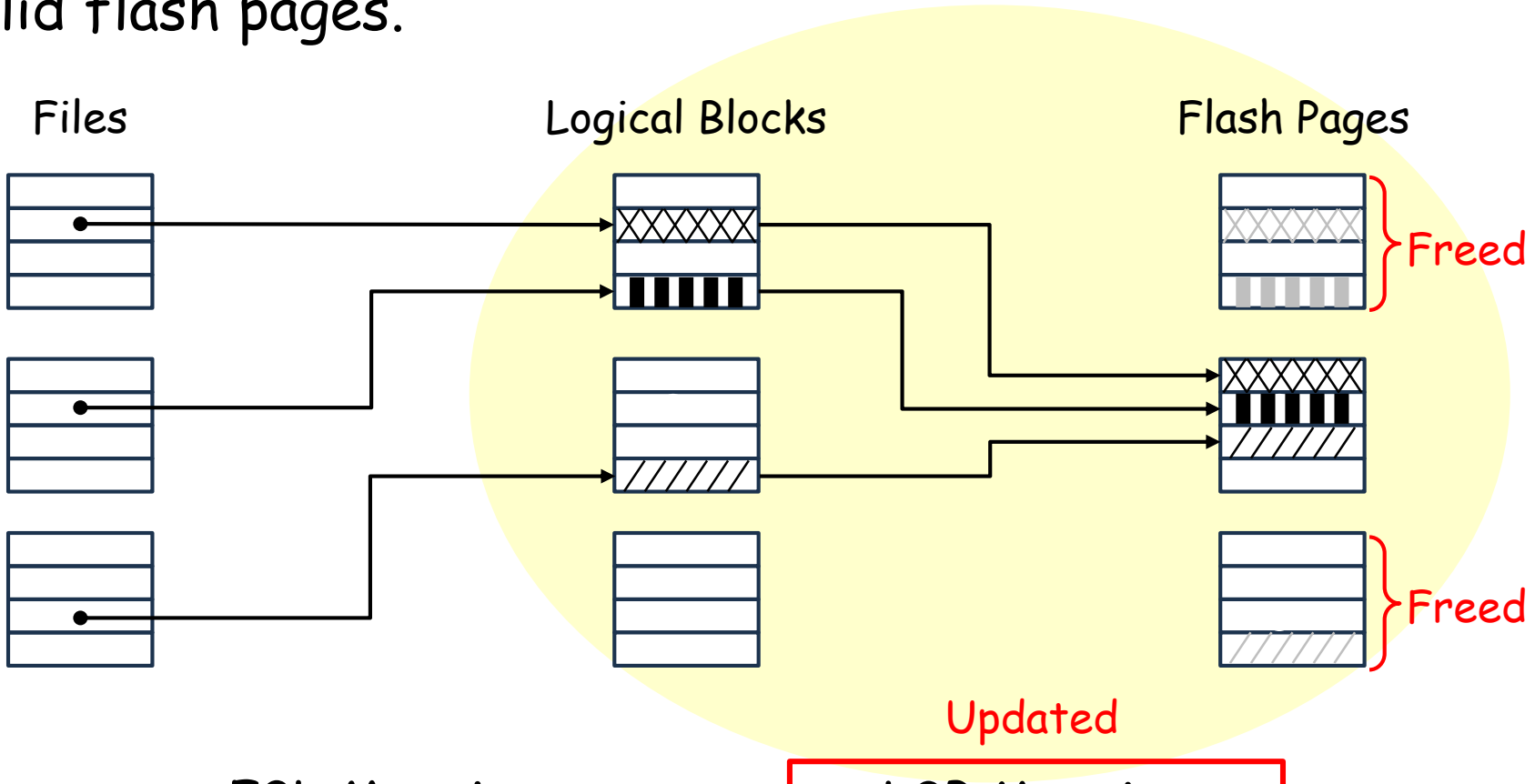
Garbage Collection in Flash Storage

Reclaim invalid flash pages.

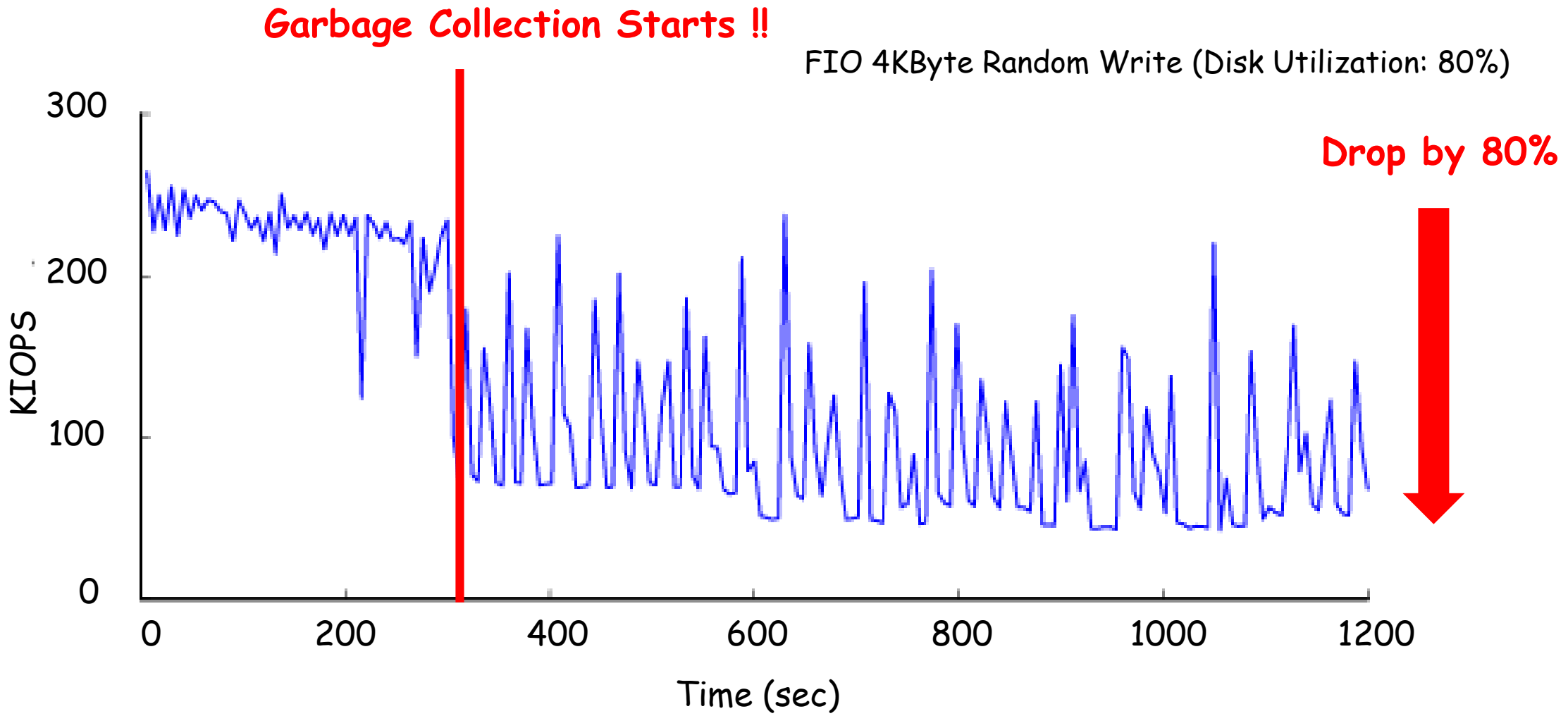


Garbage Collection in Flash Storage

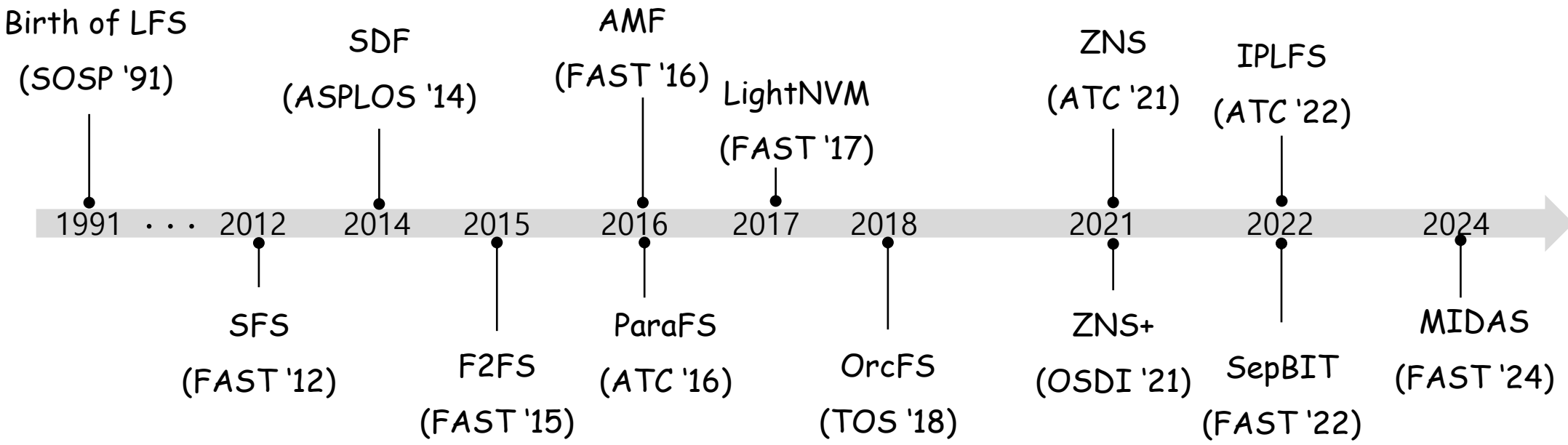
Reclaim invalid flash pages.



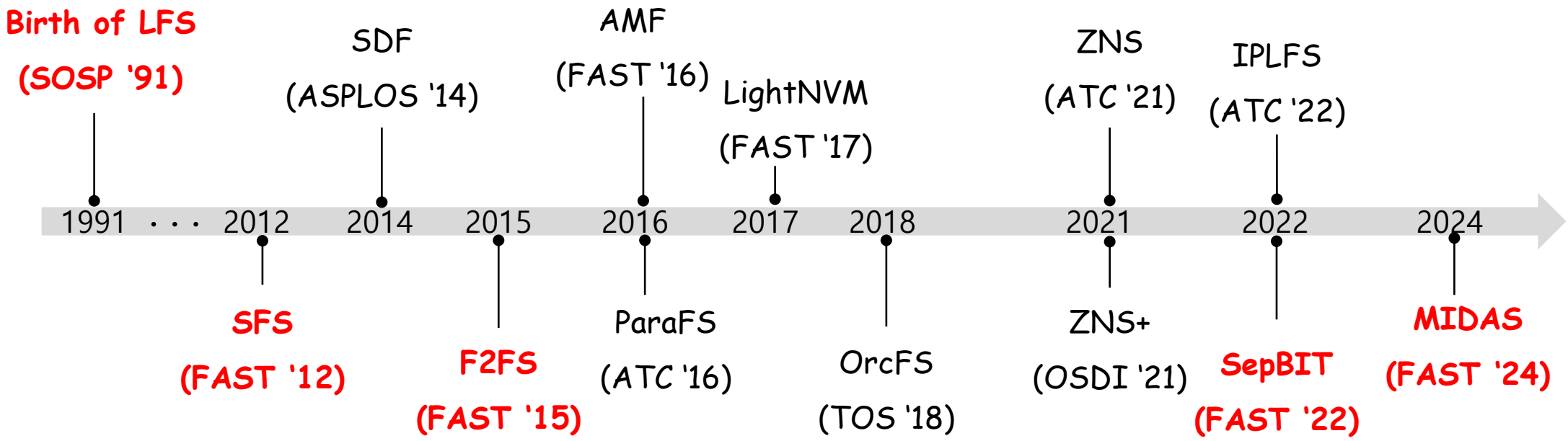
Overhead of Garbage Collection



Previous Approach to alleviate GC overhead



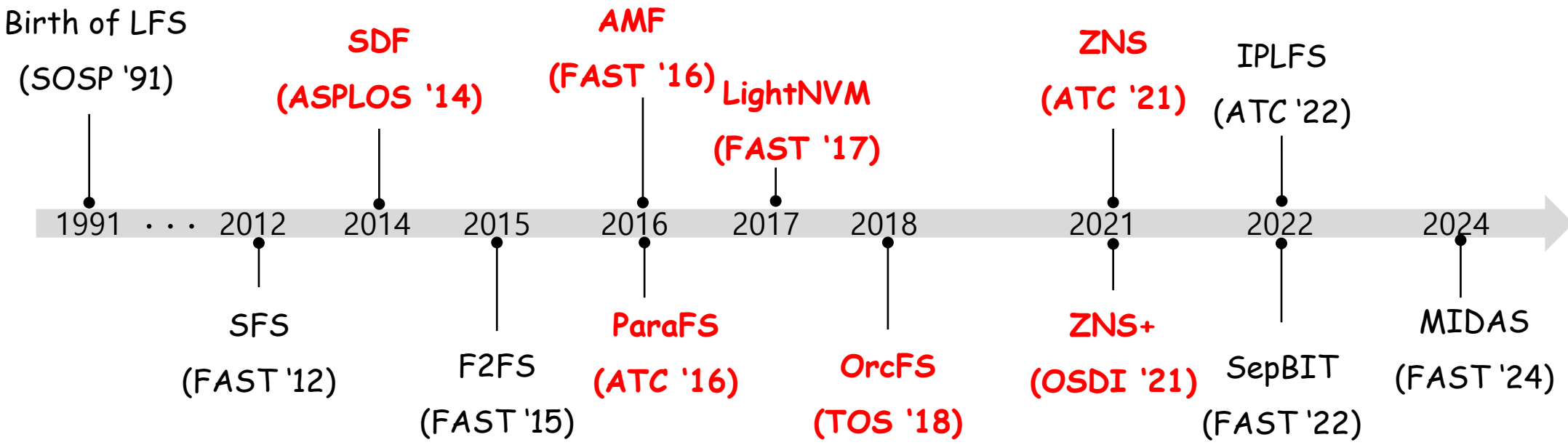
Previous Approach to alleviate GC overhead



Cluster the file blocks with similar lifespan together.

→ Reduce WAF.

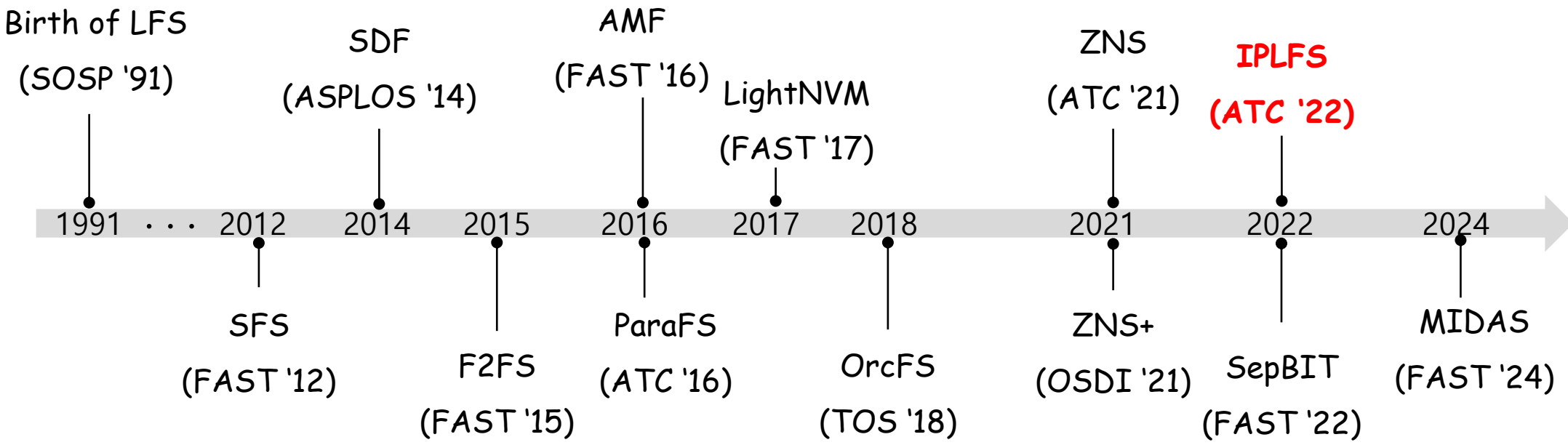
Previous Approach to alleviate GC overhead



The filesystem cleans the storage device.

→ Eliminate device GC.

Previous Approach to alleviate GC overhead

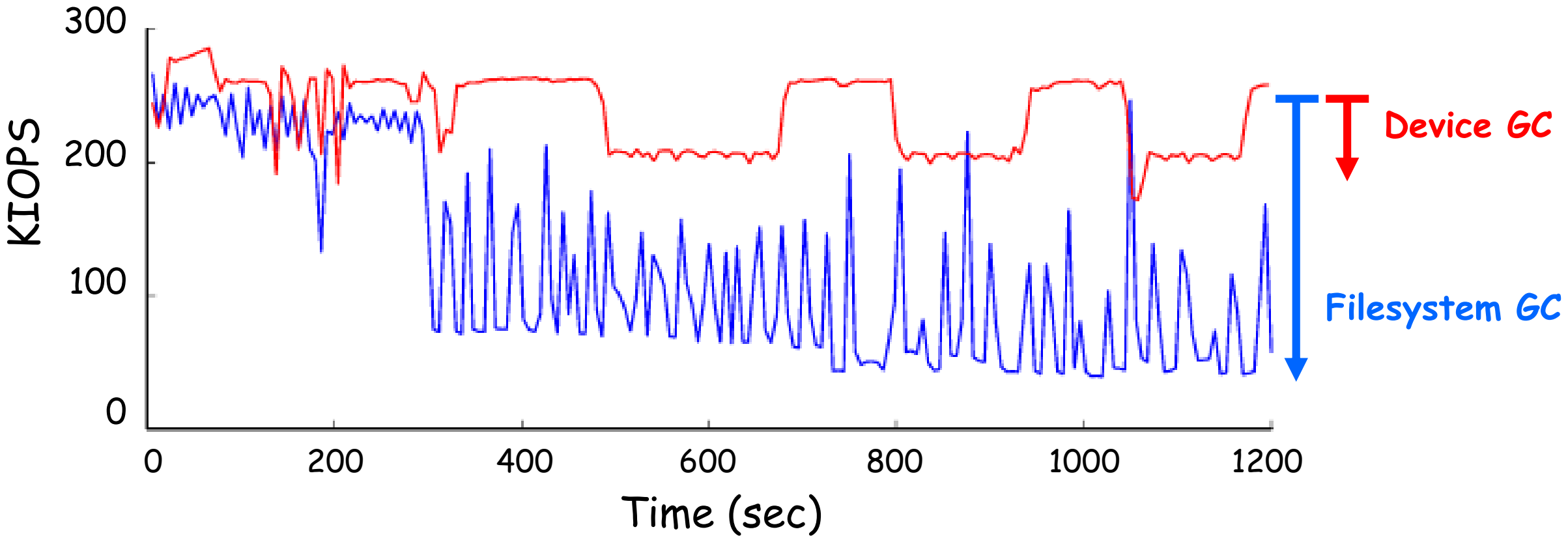


Eliminate the filesystem GC.

→ L2P mapping structure becomes huge and complex.

→ **Hard to deploy in the real world.**

Filesystem GC is much more costly than the Device GC !



Filesystem GC is more costly than Device GC.

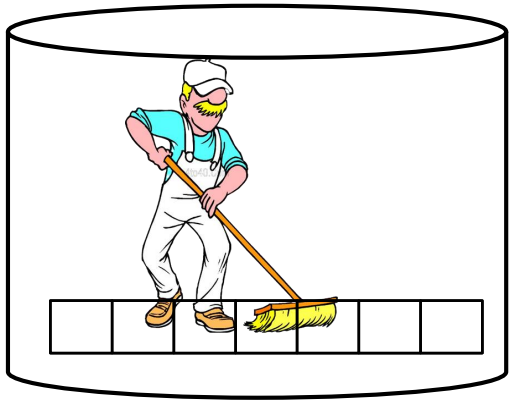
	Filesystem GC	Device GC
Filesystem Semantic-Aware	👍	👎
Checkpoint Overhead	👎	👍
Filemap Traversal Overhead	👎	👍
Metadata Update Overhead	👎	👍
Host-Device Data Transfer Overhead	👎	👍
Page Allocation Overhead	👎	👍

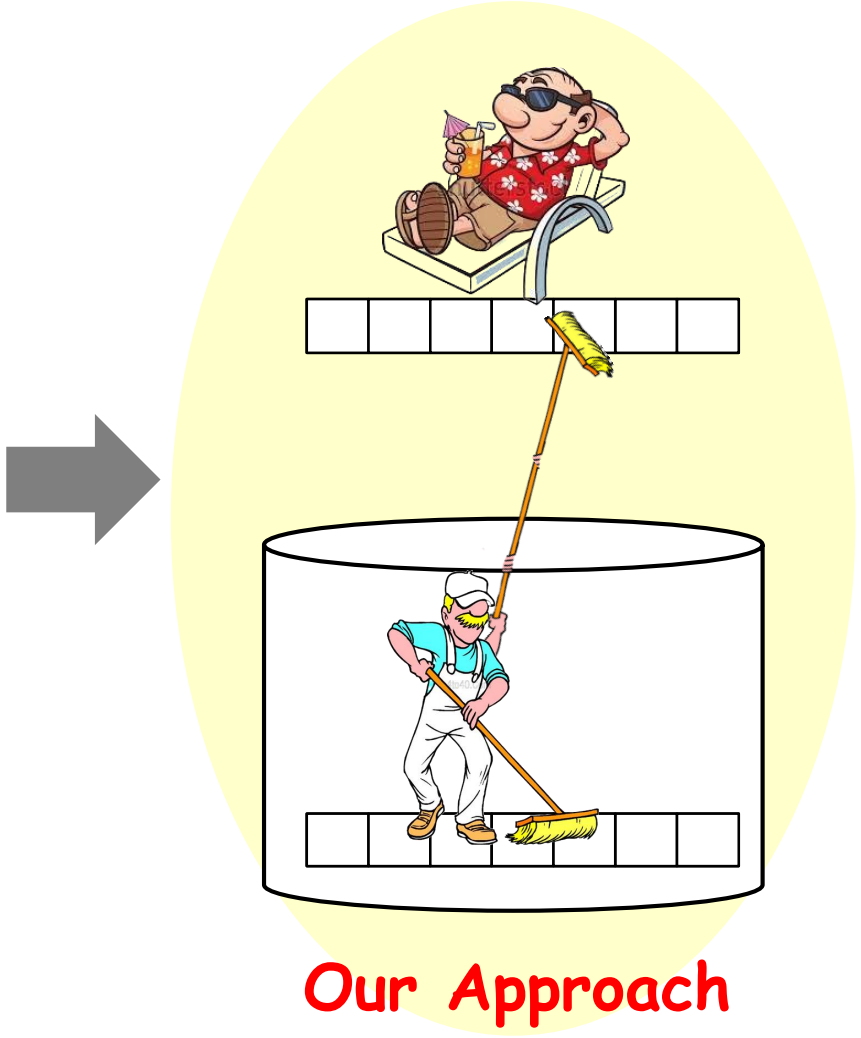
Filesystem GC involves too many sub-operations compared to Device GC !

Let the device clean the filesystem partition.

Filesystem Partition

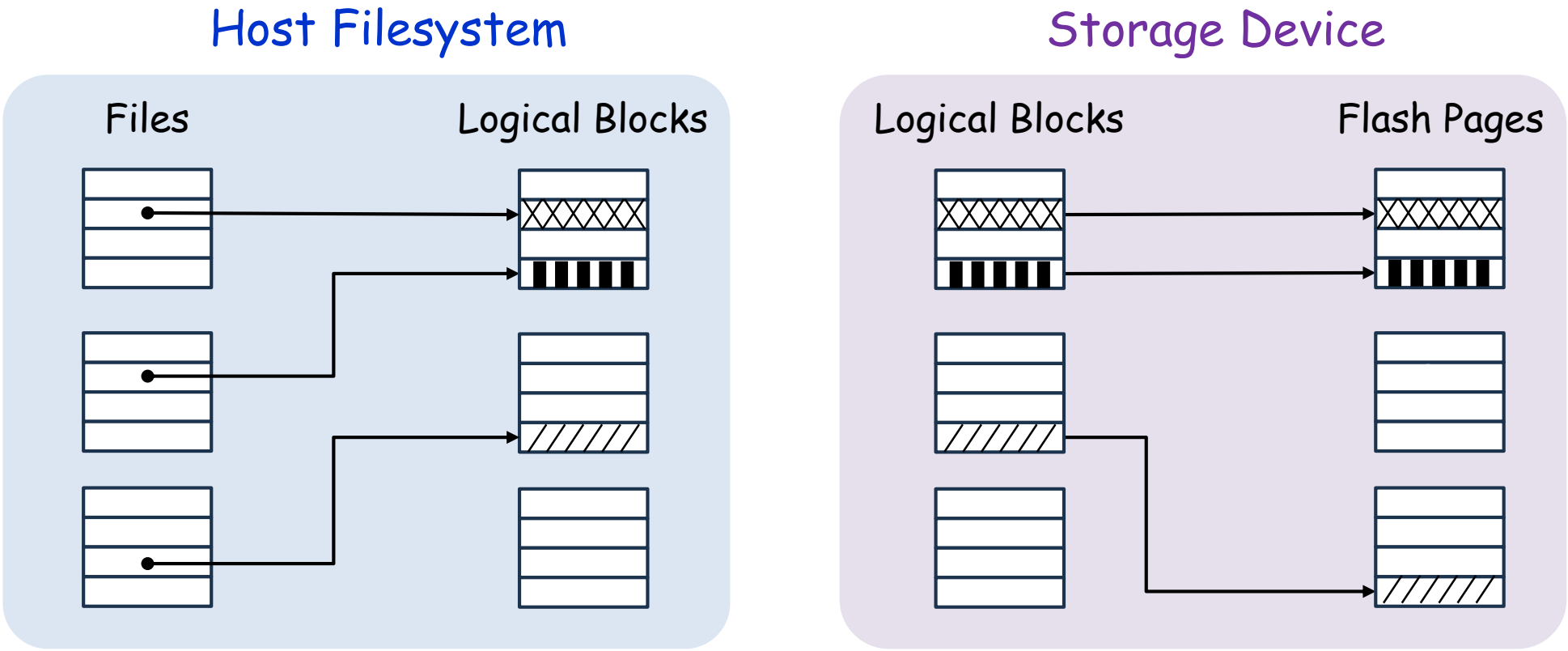
Flash Storage





Our Approach

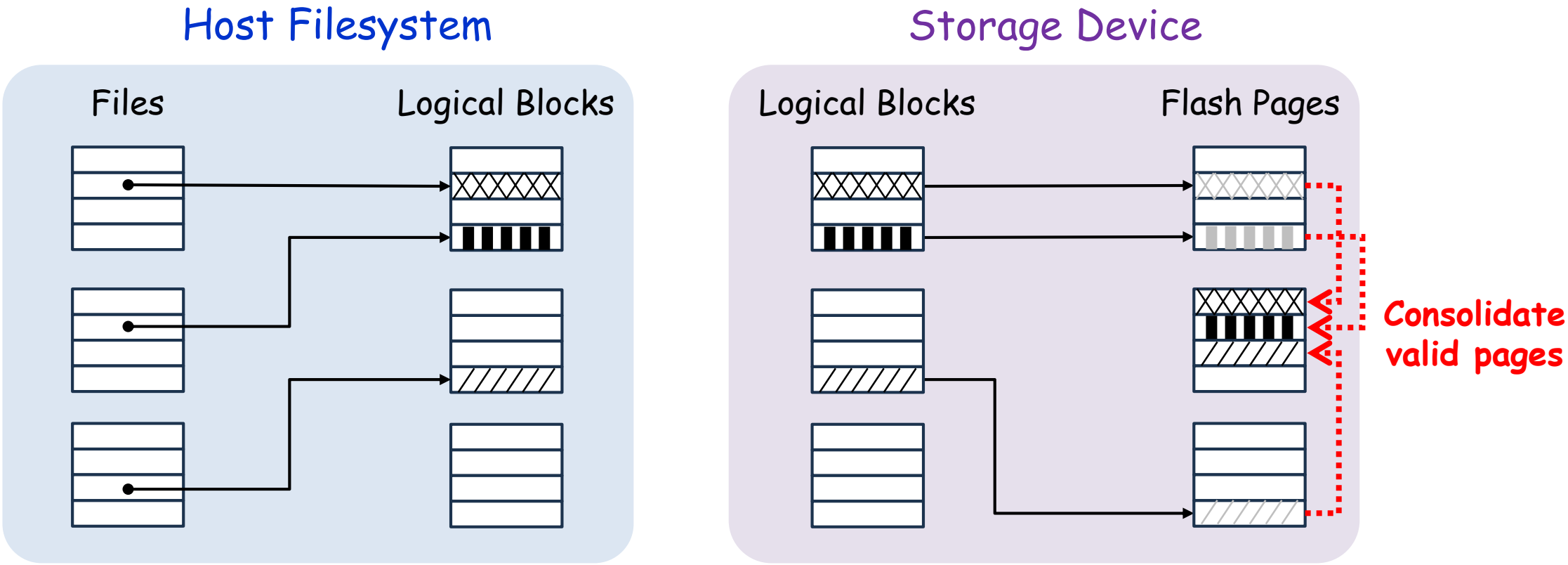
Device GC to clean Filesystem Partition



F2L Mapping
[fd, offset] → LBA

L2P Mapping
LBA → PPA

Device GC to clean Filesystem Partition

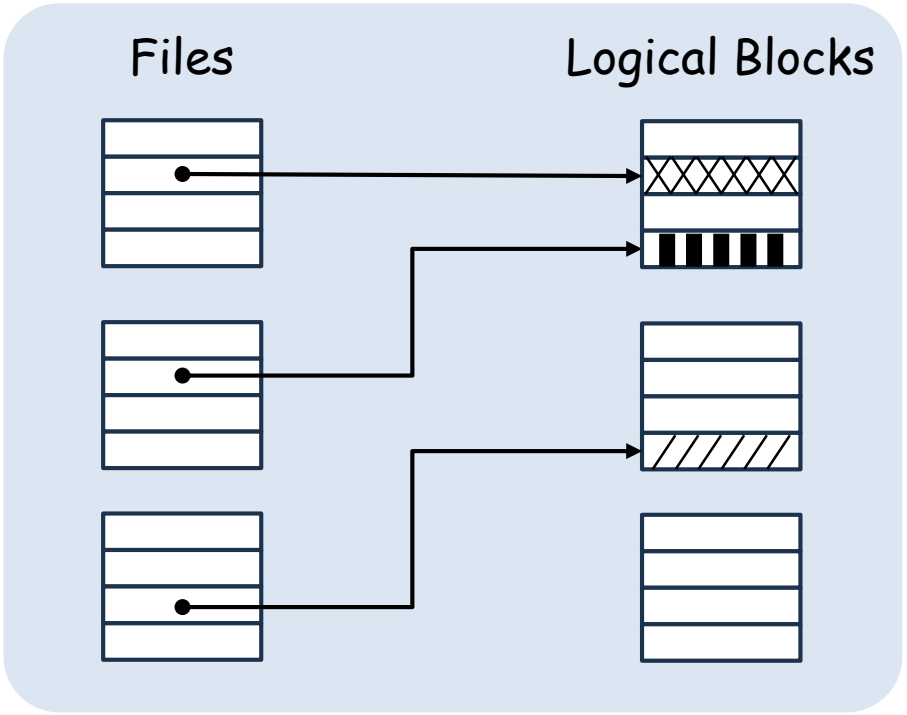


F2L Mapping
[fd, offset] → LBA

L2P Mapping
LBA → ~~PPA~~
PPA'

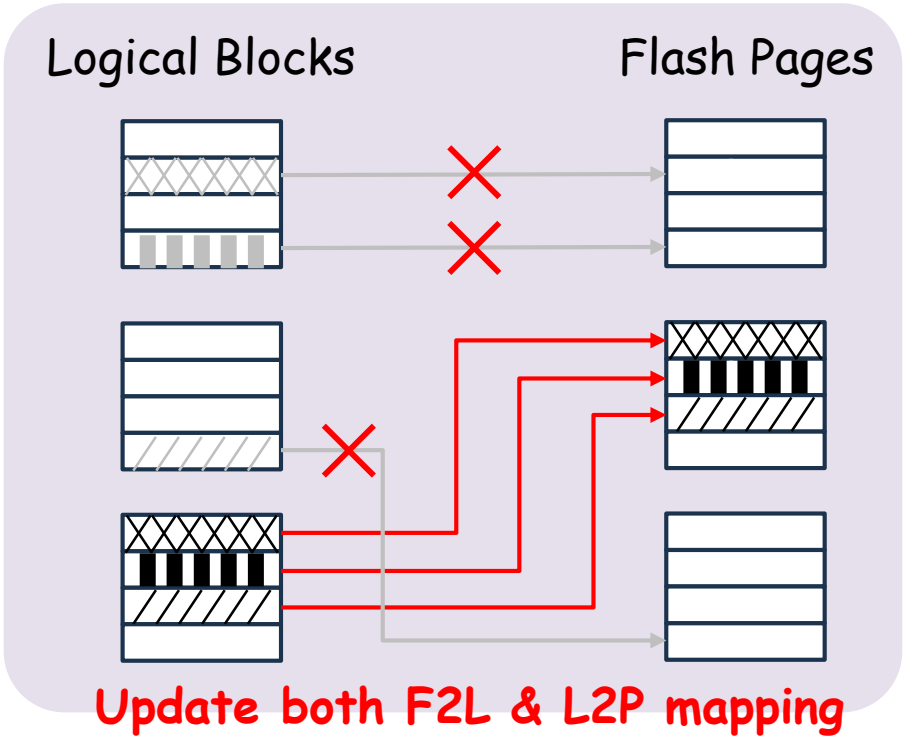
Device GC to clean Filesystem Partition

Host Filesystem



F2L Mapping
[fd, offset] → LBA

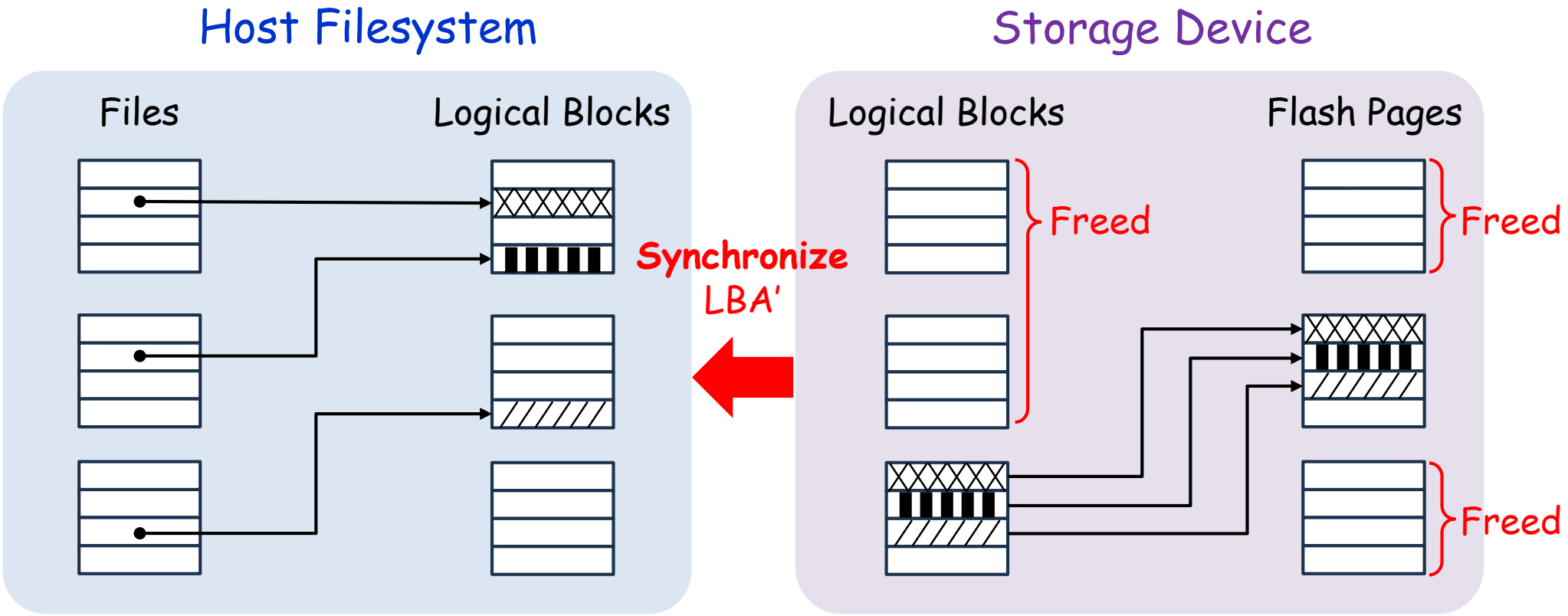
Storage Device



Update both F2L & L2P mapping

L2P Mapping
~~LBA → PPA~~
LBA' → PPA'

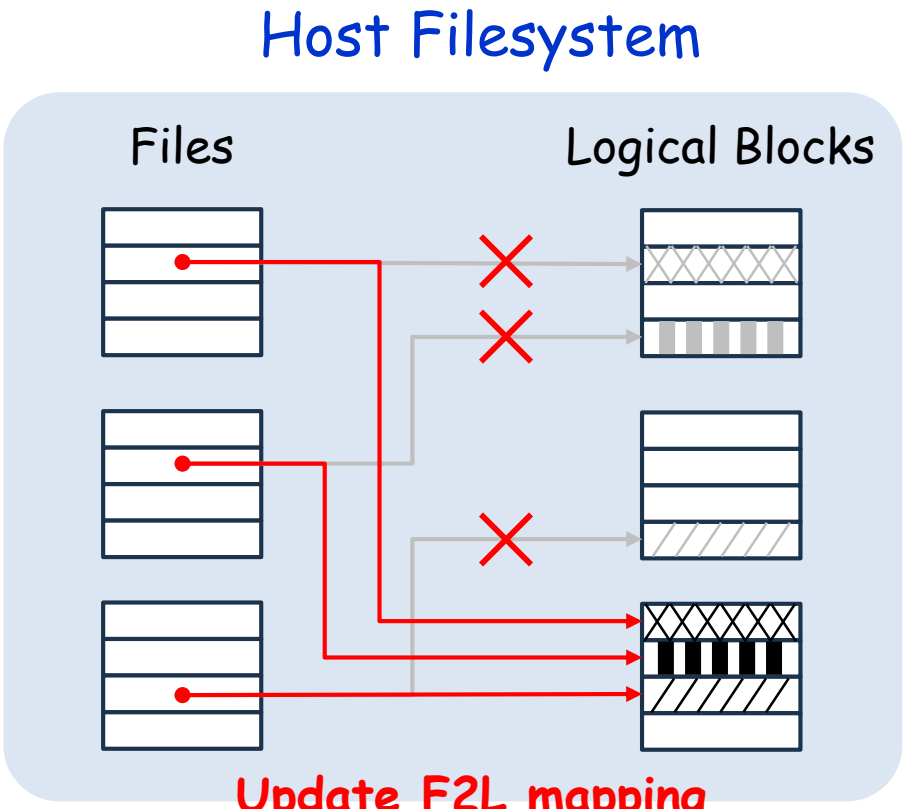
Device GC to clean Filesystem Partition

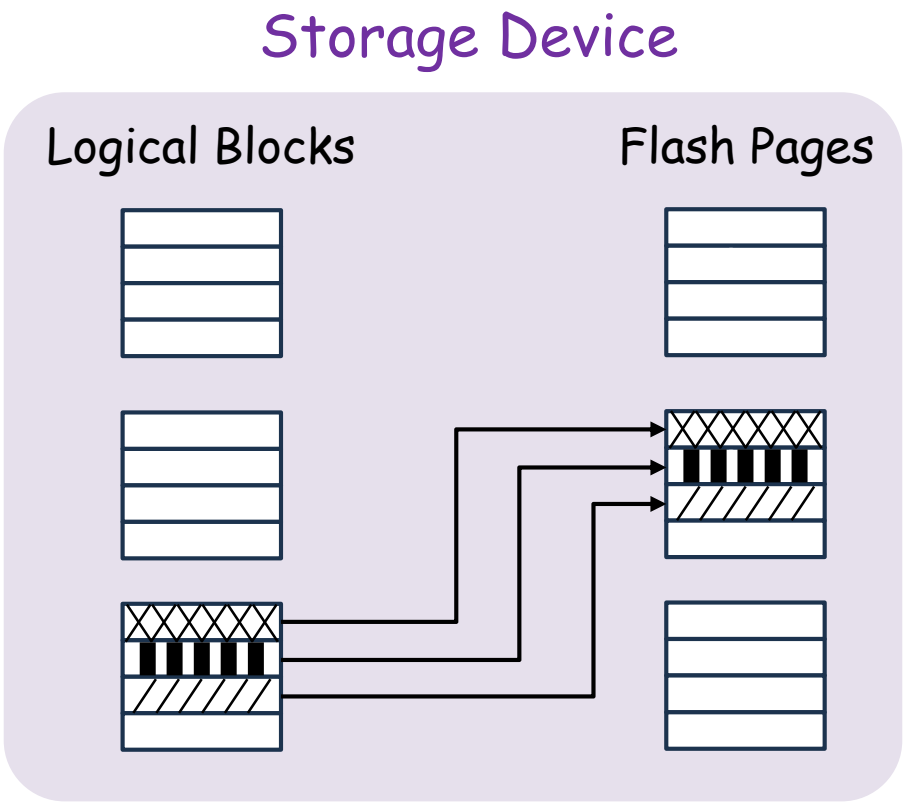


F2L Mapping
[fd, offset] → LBA

L2P Mapping
~~LBA → PPA~~
LBA' → PPA'

Device GC to clean Filesystem Partition

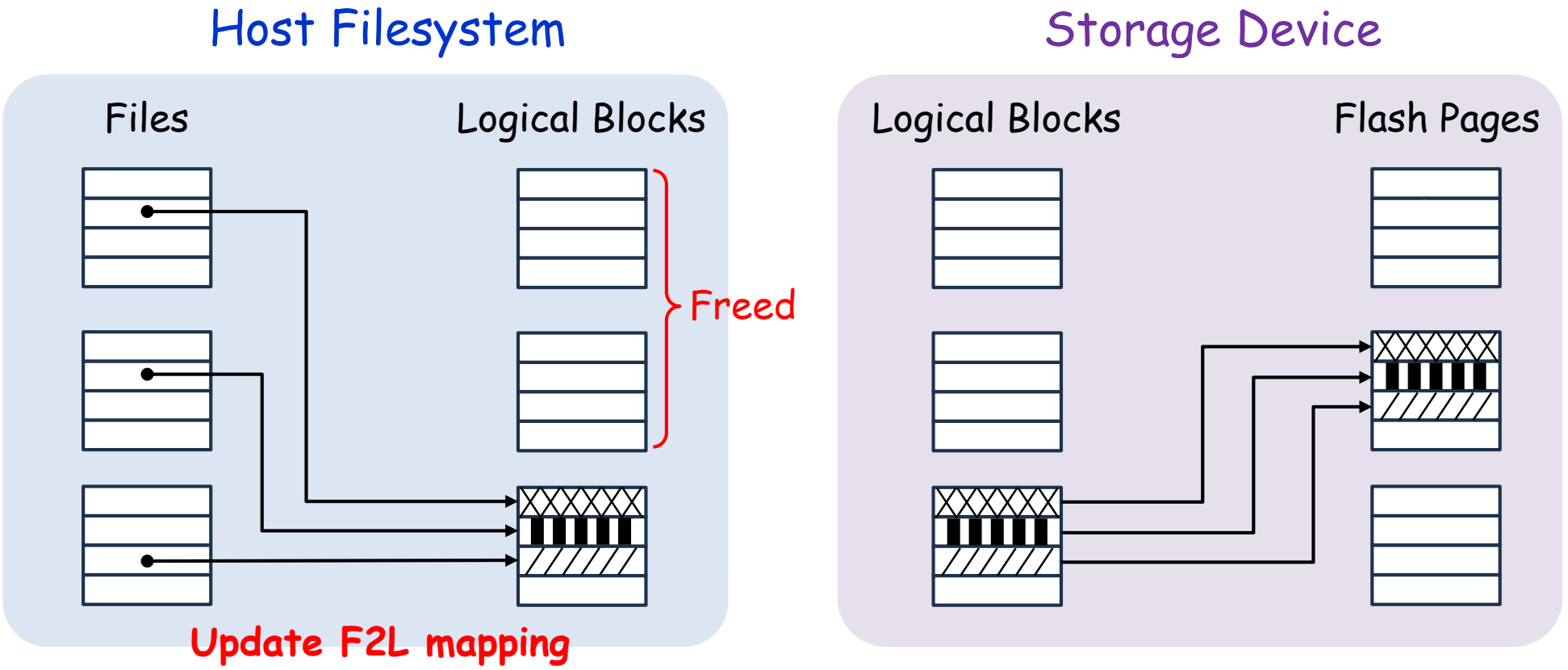




F2L Mapping
[fd, offset] → ~~LBA~~
LBA'

L2P Mapping
~~LBA → PPA~~
LBA' → PPA'

Device GC to clean Filesystem Partition

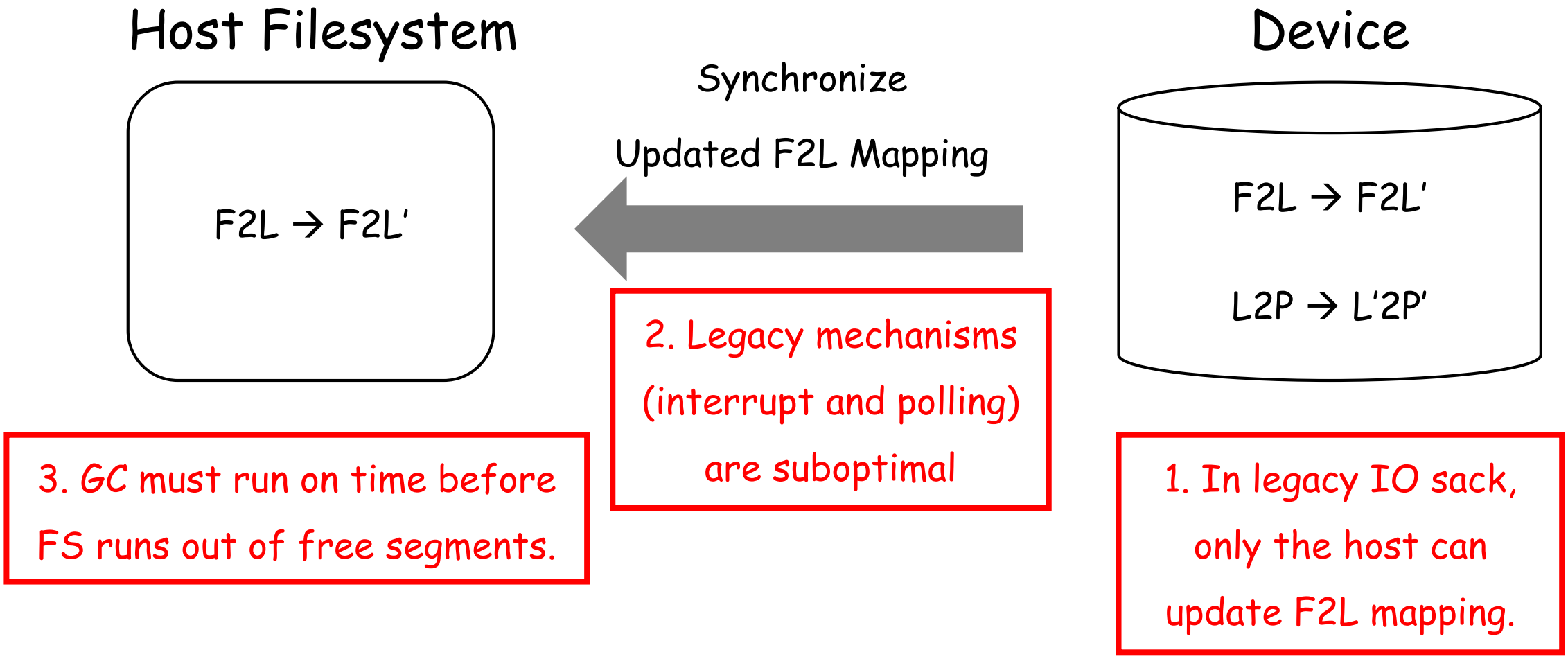


F2L Mapping
[fd, offset] → ~~LBA~~
LBA'

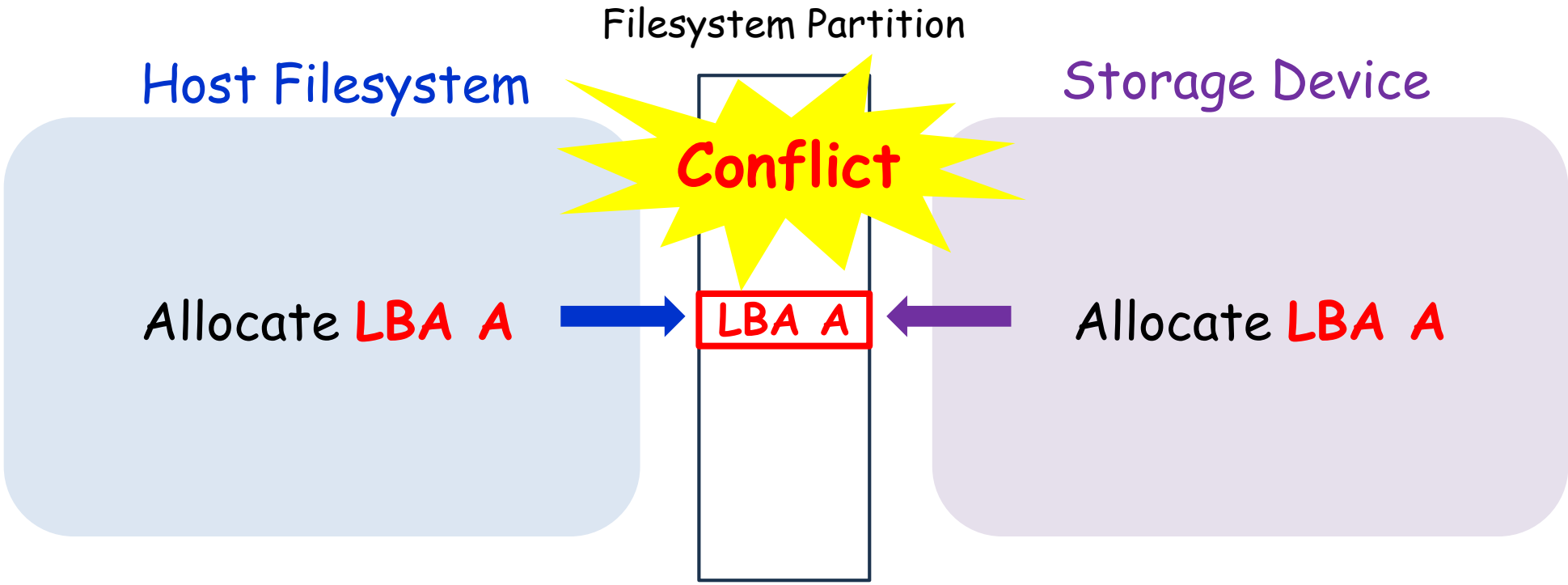
L2P Mapping
~~LBA → PPA~~
LBA' → PPA'

Problem Formulation

Challenges


Host Filesystem
F2L → F2L'
Synchronize
Updated F2L Mapping
Device
F2L → F2L'
L2P → L'2P'
2. Legacy mechanisms (interrupt and polling) are suboptimal
3. GC must run on time before FS runs out of free segments.
1. In legacy IO sack, only the host can update F2L mapping.

Challenge 1. Device should be able to update F2L mapping without interfering the filesystem activity.



Challenge 2. Optimal Interface for device-host synchronization.

	Saving the host CPU cycles	Seamless integration with the existing I/O stack
Polling	X	O
Interrupt	O	X
???	O	O

Polling wastes the host CPU cycles.

Defining a new interrupt requires significant changes in existing I/O stack…

- A new interrupt, a new interrupt vector table, and a new interrupt handler.

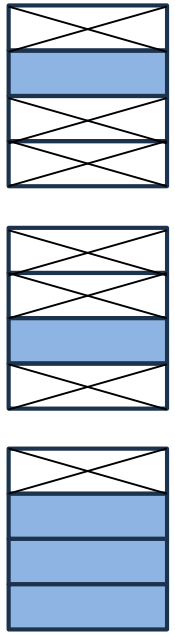
Challenge 3. GC must run on time before FS runs out of free segments.

Filesystem Partition

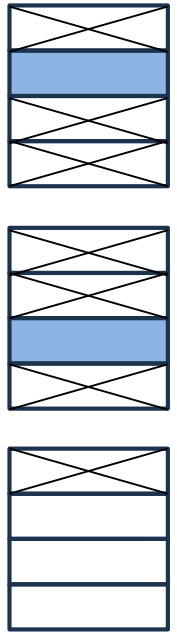
Waiting for GC

to reclaim invalid

file blocks !



Storage Space

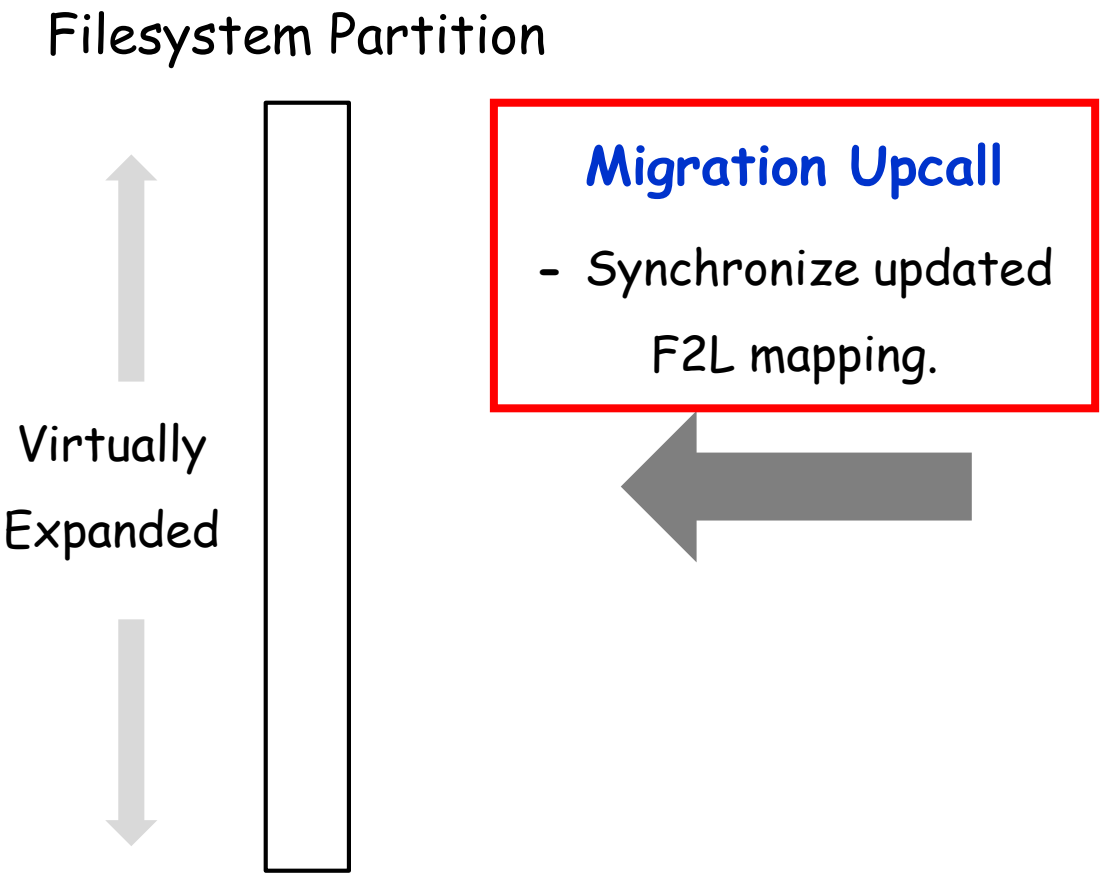


Device runs GC on

its own schedule.

D2FS: Device-Driven Filesystem Garbage Collection

Design Overview of D2FS

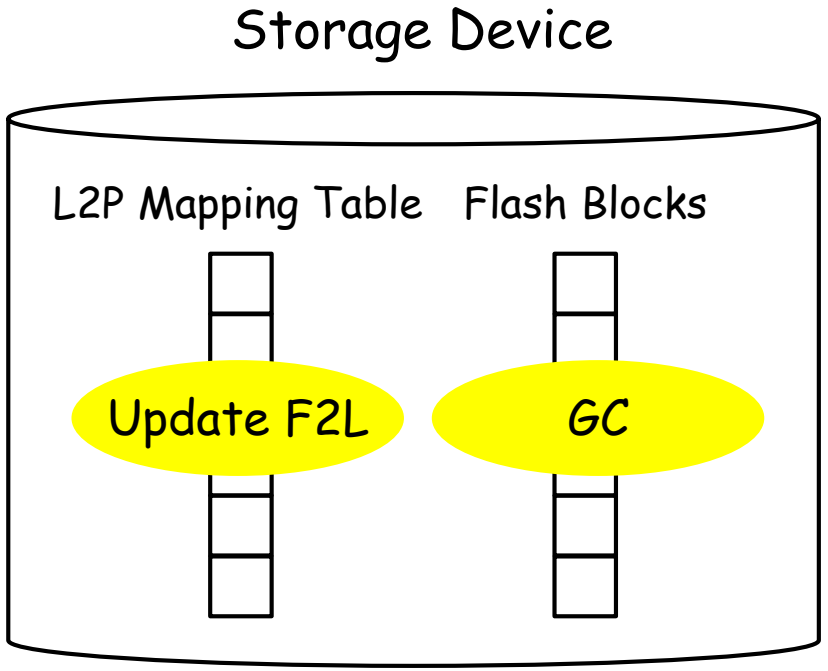


Migration Upcall

- Synchronize updated F2L mapping.

Virtual Overprovisioning

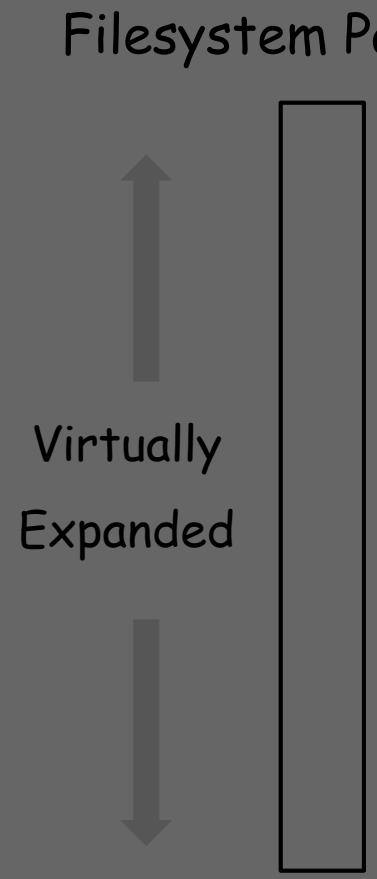
- Prevent FS running out of free blocks.



Coupled Garbage Collection

- Device updates F2L mapping.

Design Overview of D2FS

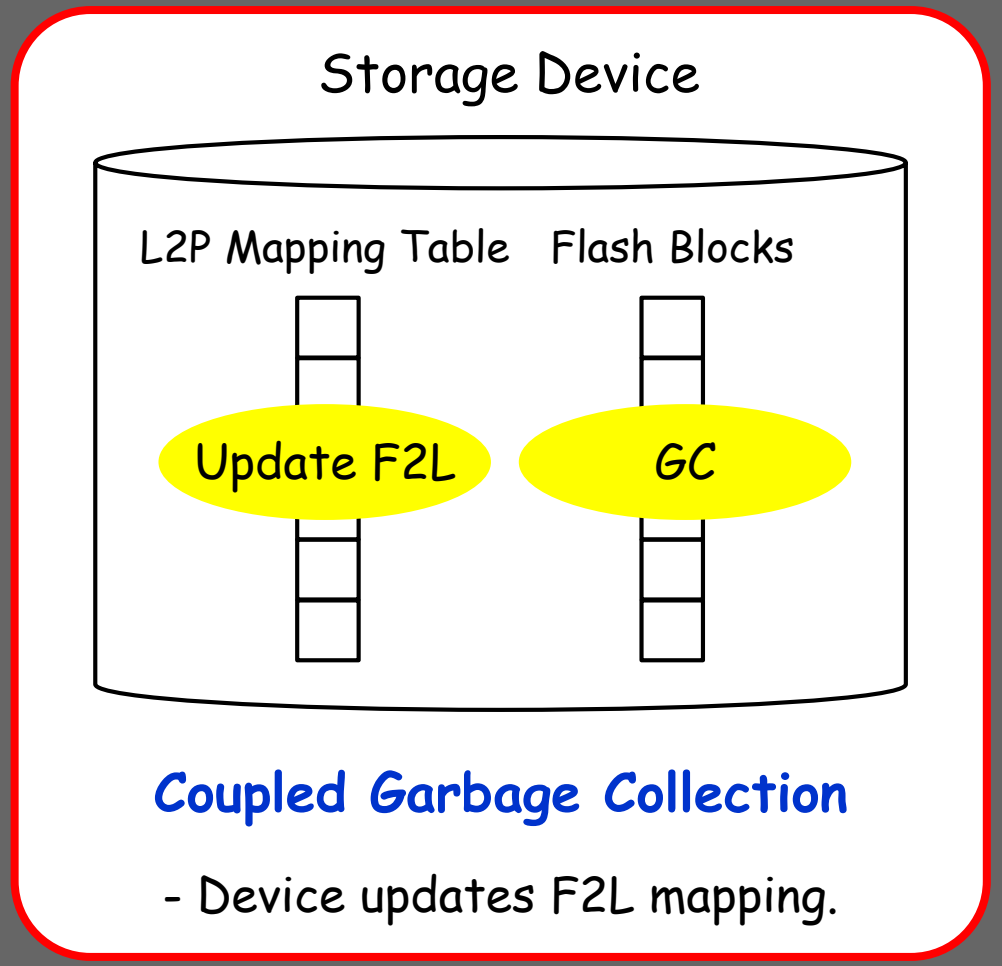


Virtual Overprovisioning

- Prevent FS running out of free blocks

Migration Upcall

- Synchronize updated F2L mapping.

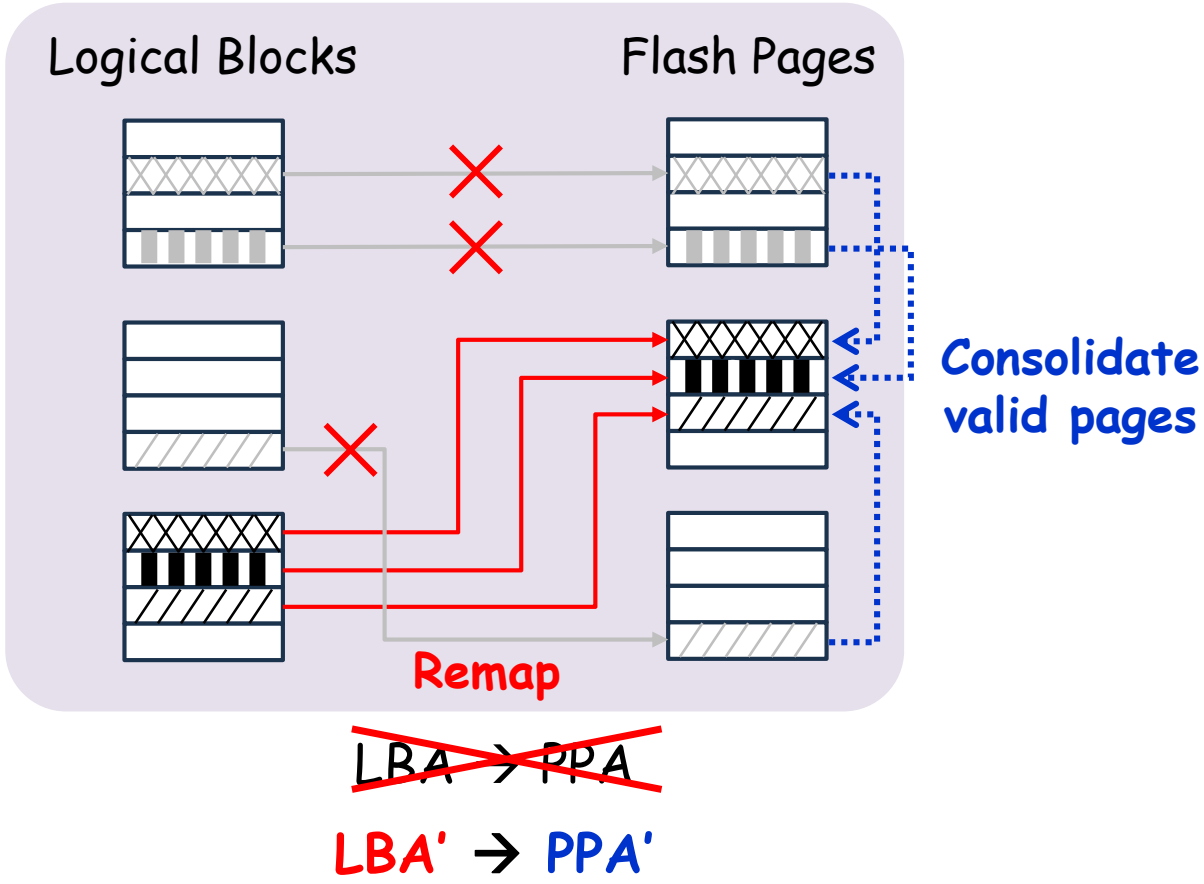


Coupled Garbage Collection

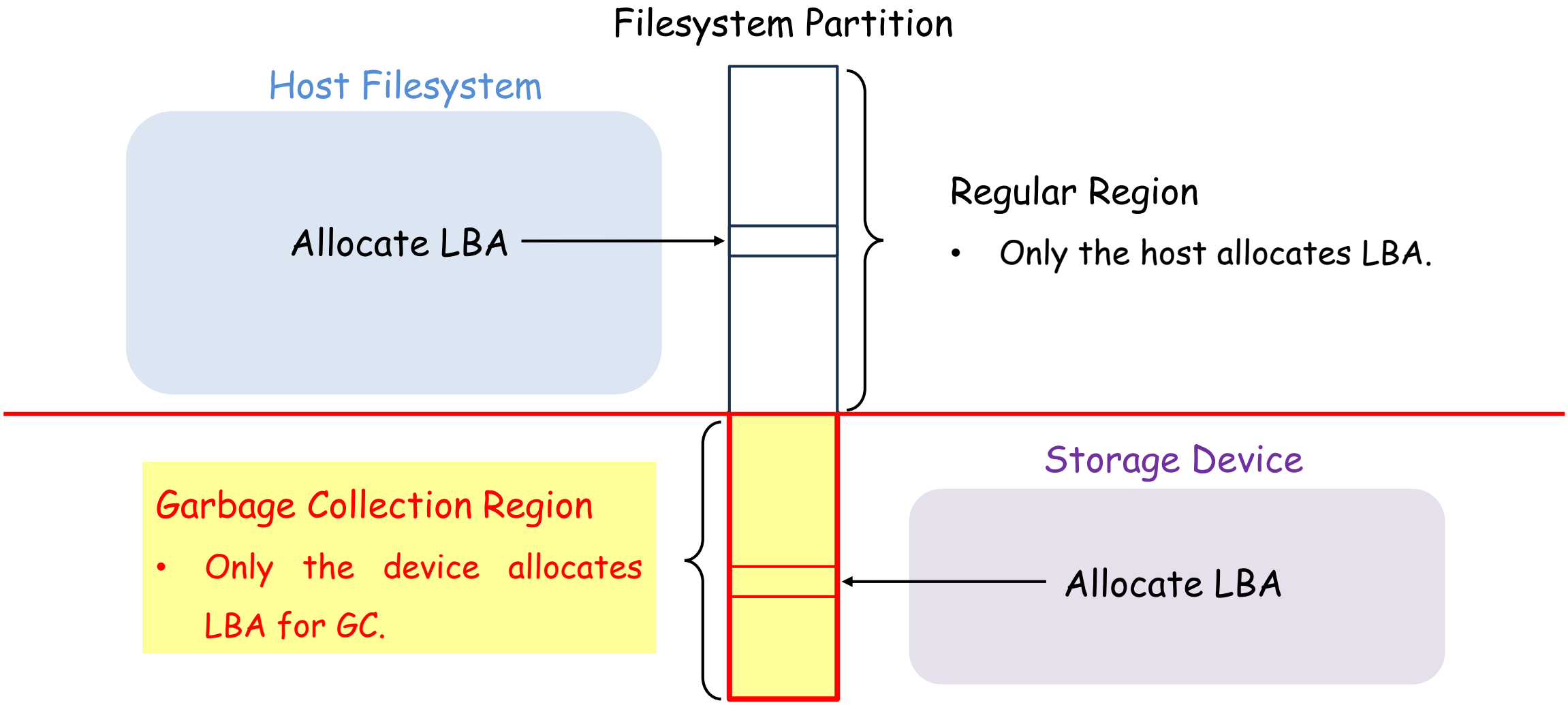
- Device updates F2L mapping.

Device GC to clean Filesystem Partition

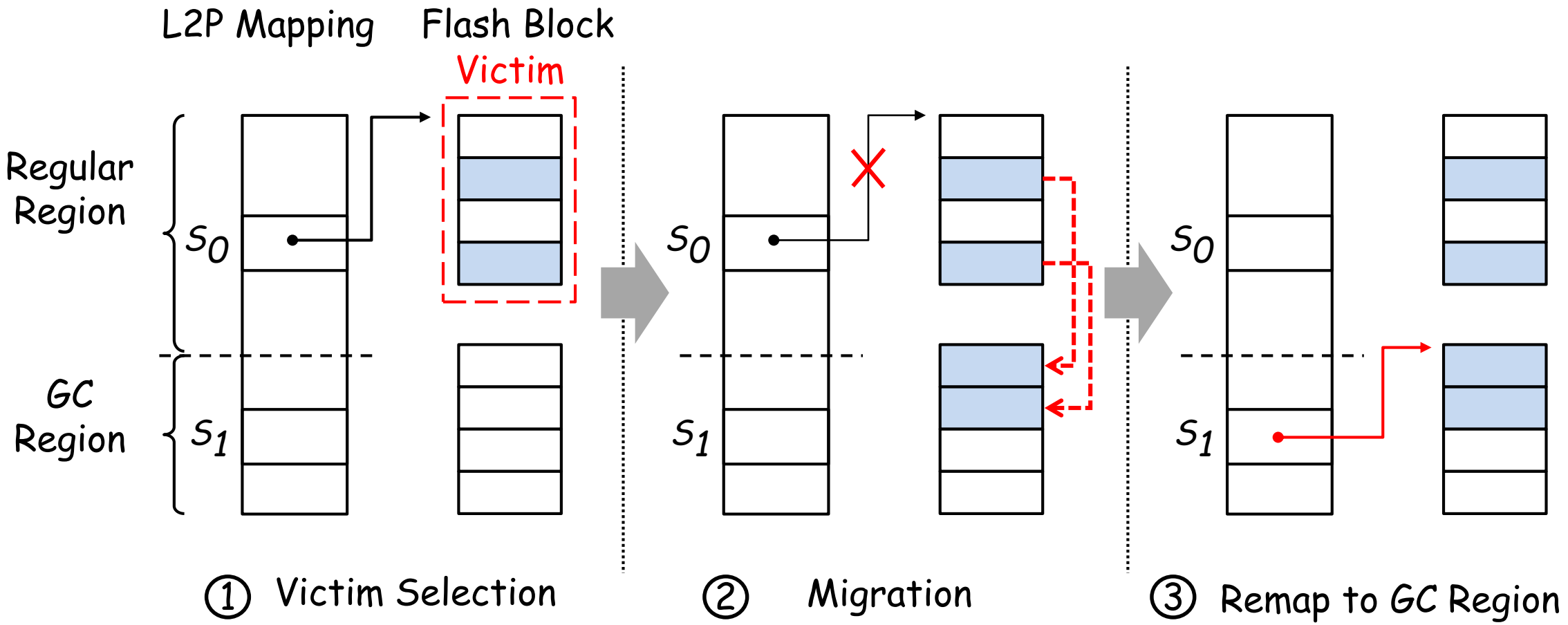
- Device garbage collection
- Update both **PPA** and **LBA** → Reclaim both **flash block** and **filesystem segment** together.



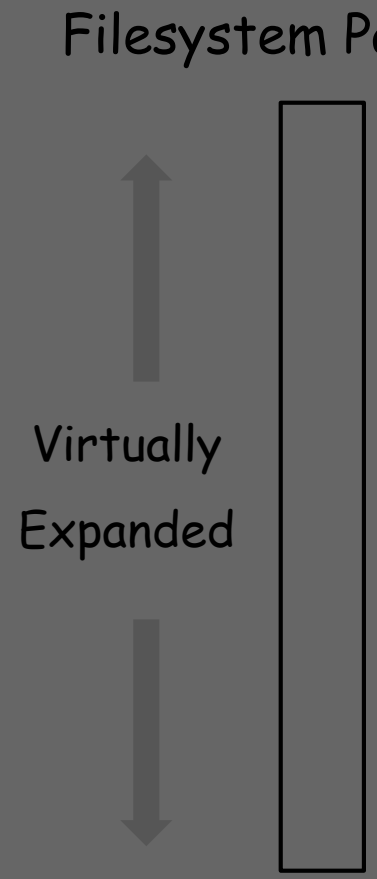
Dedicated Filesystem Region for Device GC



Coupled Garbage Collection



Design Overview of D2FS

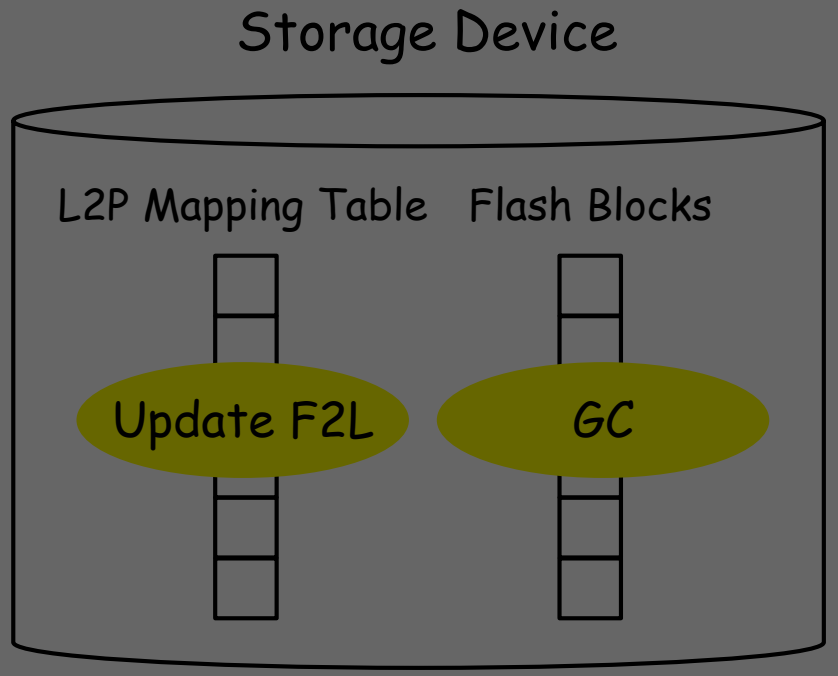


Virtual Overprovisioning

- Prevent FS running out of free blocks

Migration Upcall

- Synchronize updated F2L mapping



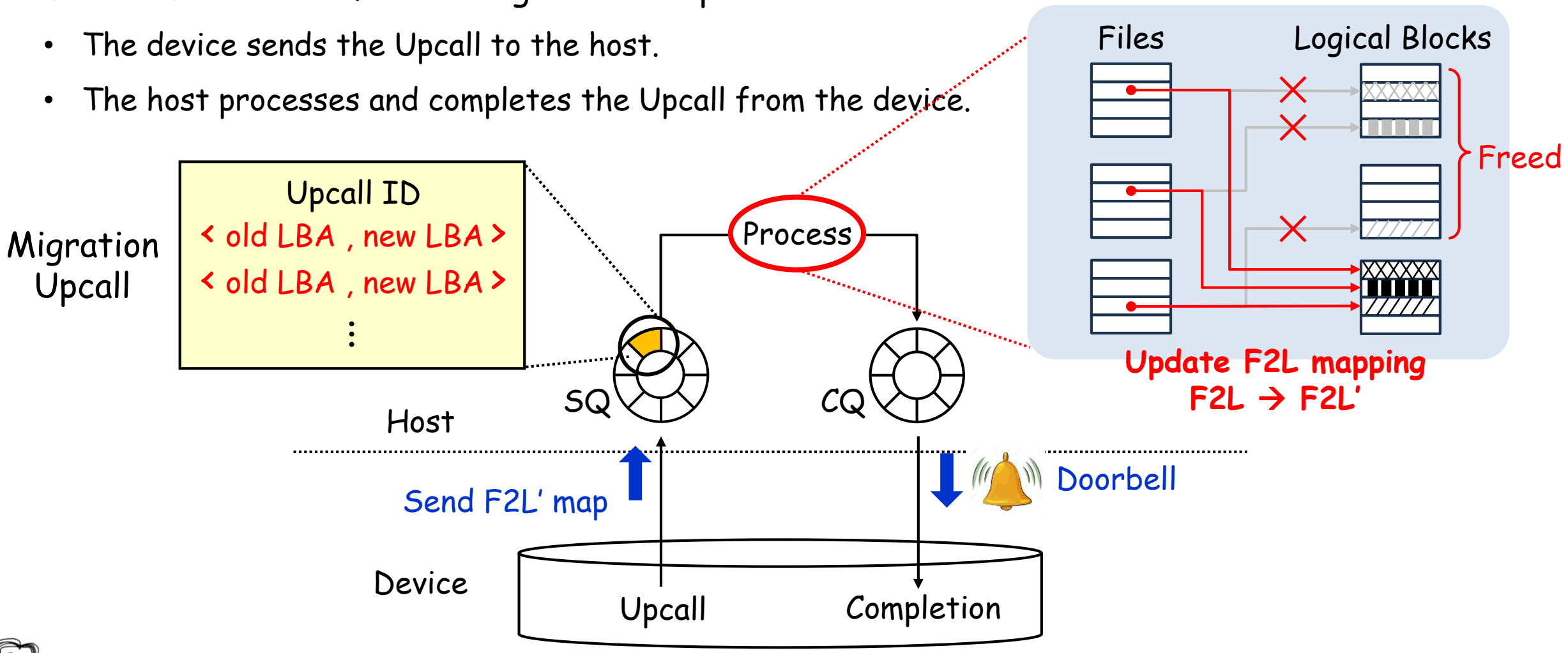
Coupled Garbage Collection

- Device updates F2L mapping

Migration Upcall

Device-Driven I/O for sending the LBA updates.

- The device sends the Upcall to the host.
- The host processes and completes the Upcall from the device.

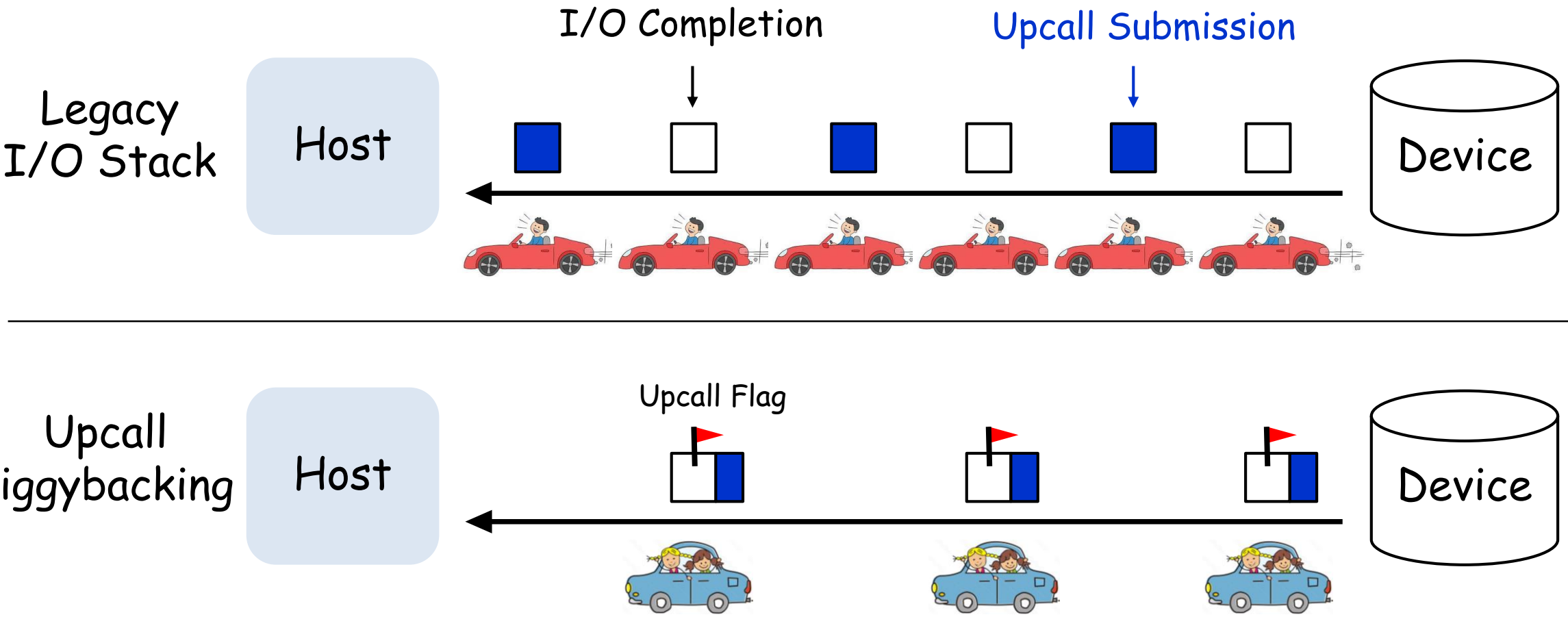


I/O Signaling Mechanism for Migration Upcall

	Saving the host CPU cycles	Seamless integration with the existing I/O stack
Polling	X	O
Interrupt	O	X
Upcall Piggybacking	O	O

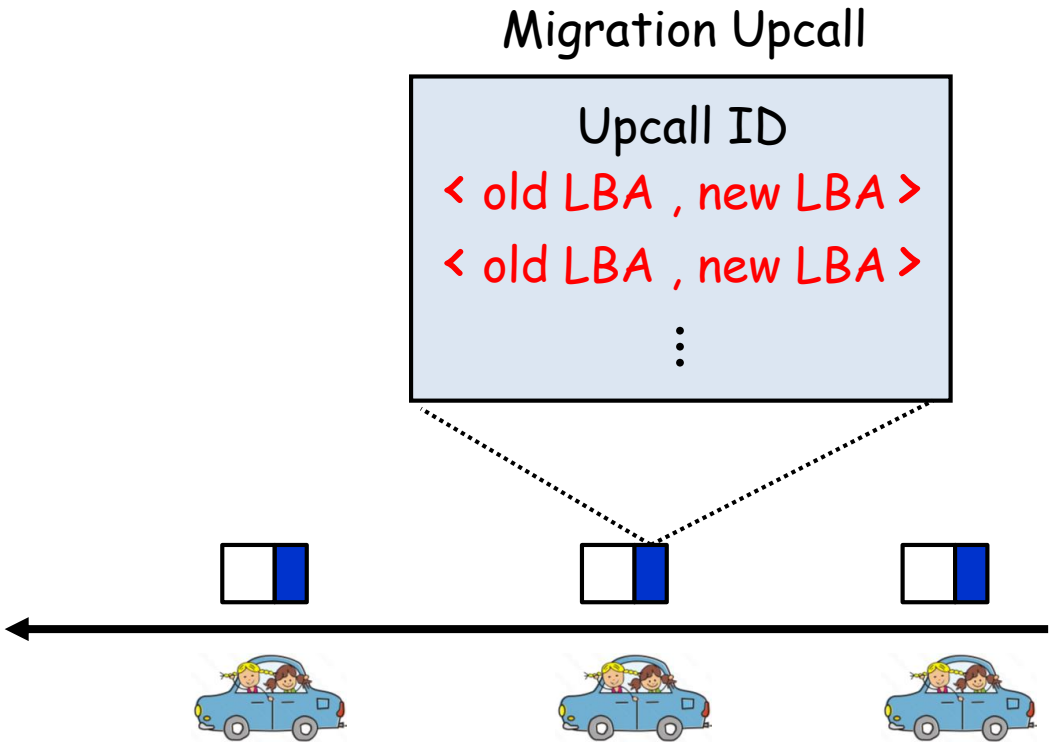
- Piggyback the upcall notification **on the completion signal of the other command.**

Upcall Piggybacking

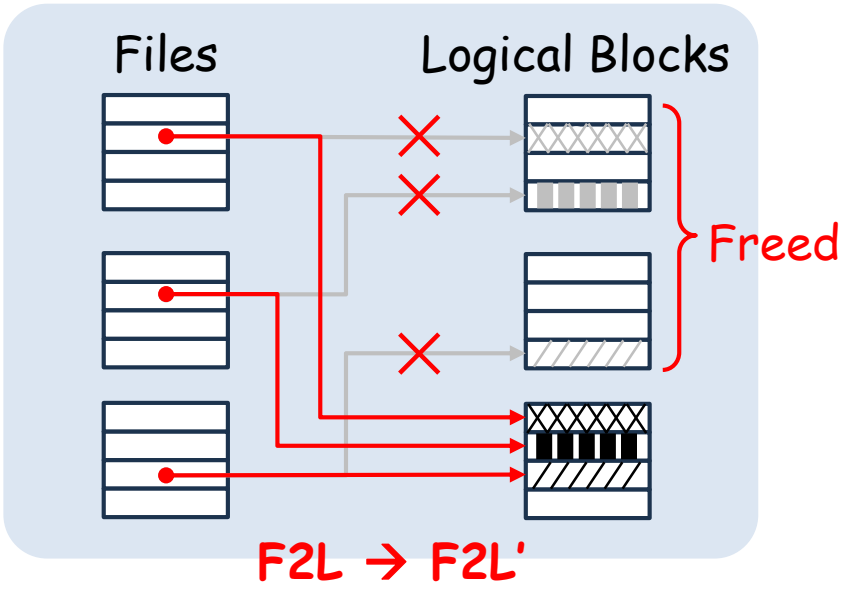


Processing Migration Upcall

1. Extract updated F2L mapping from Upcall.

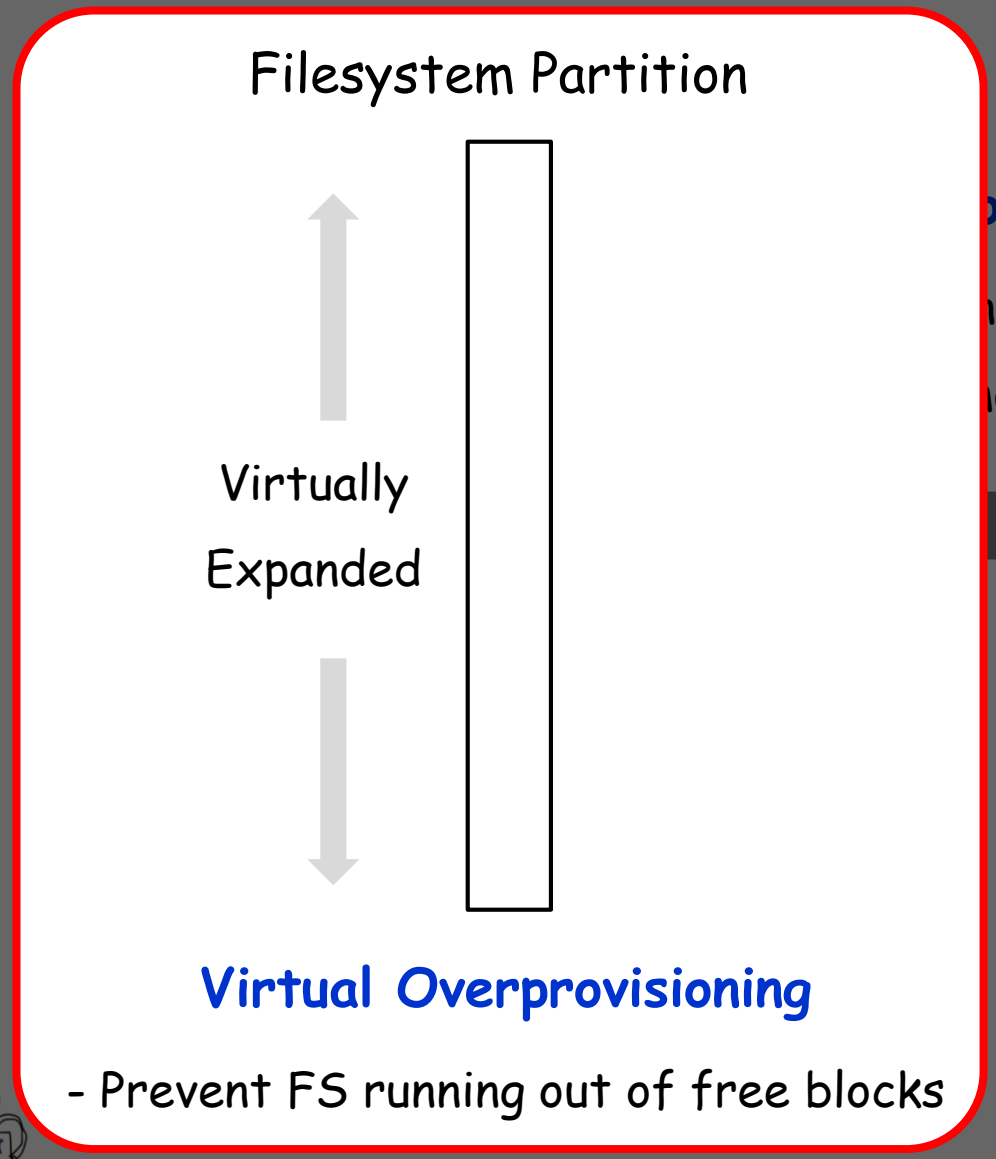


2. Update filesystem state w.r.t. updated F2L mapping.



→ create free segments.

Design Overview of D2FS



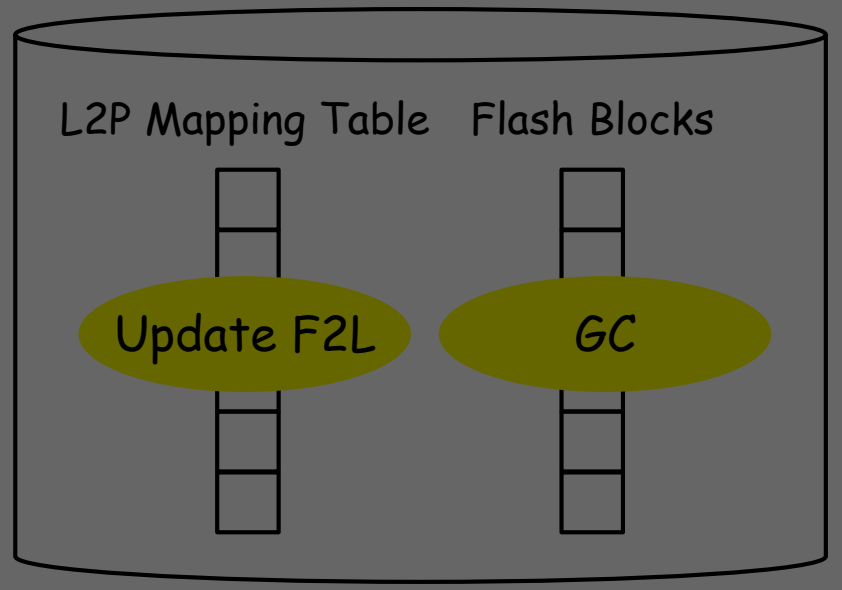
Virtual Overprovisioning

- Prevent FS running out of free blocks

on Upcall

ize updated

apping

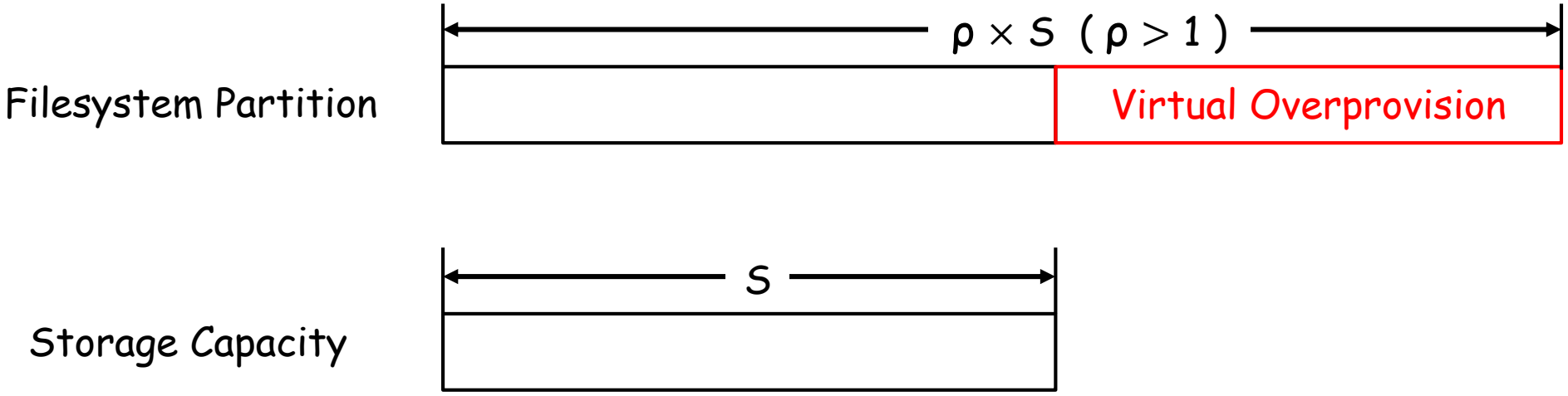


Coupled Garbage Collection

- Device updates F2L mapping

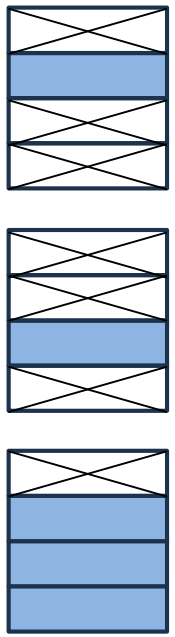
Virtual Overprovisioning

- Separate the filesystem partition size from the storage capacity.
- Virtually expand the filesystem partition size.



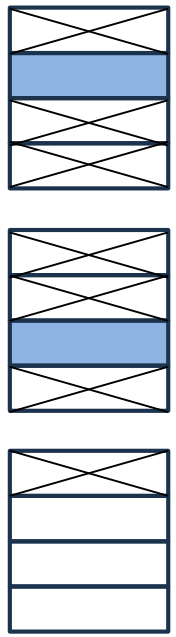
In Legacy I/O Stack

Filesystem Partition



Require device to run GC immediately !

Storage Space



GC must run on time before the filesystem lacks of free blocks.

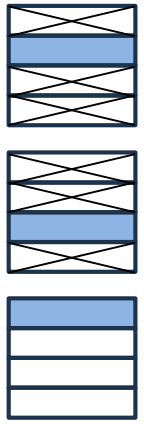
Virtual Overprovisioning

Filesystem Partition

Enough free virtual segments.

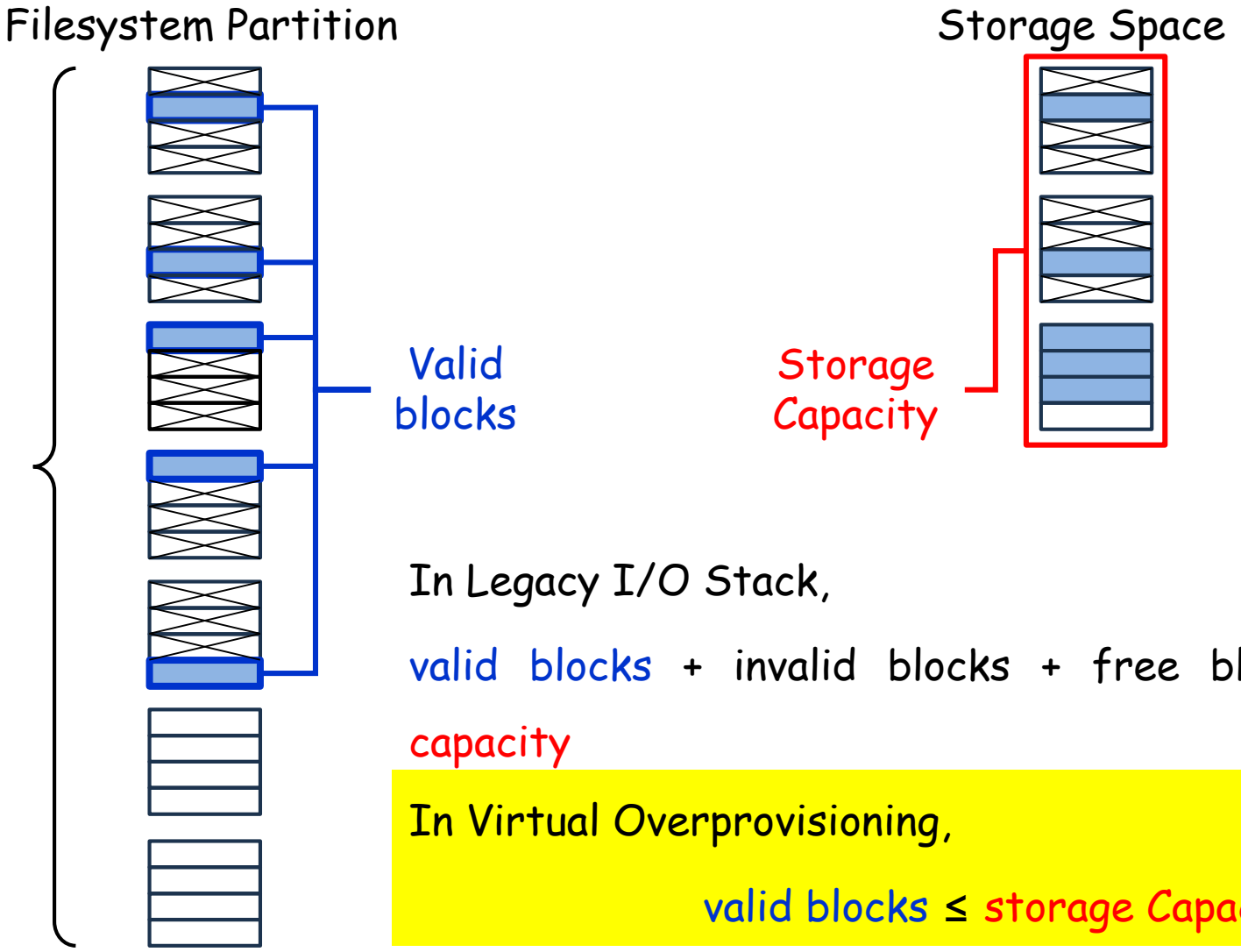
Virtual Overprovision

Storage Space



Device runs GC on its own schedule.

Virtual Overprovisioning



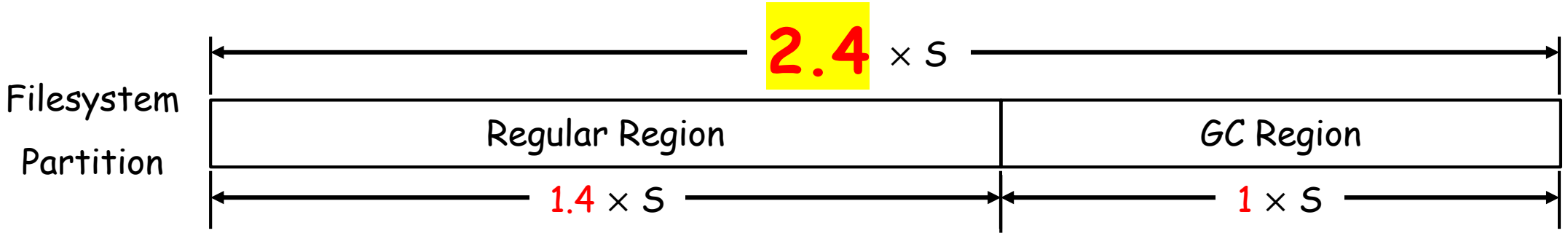
In Legacy I/O Stack,

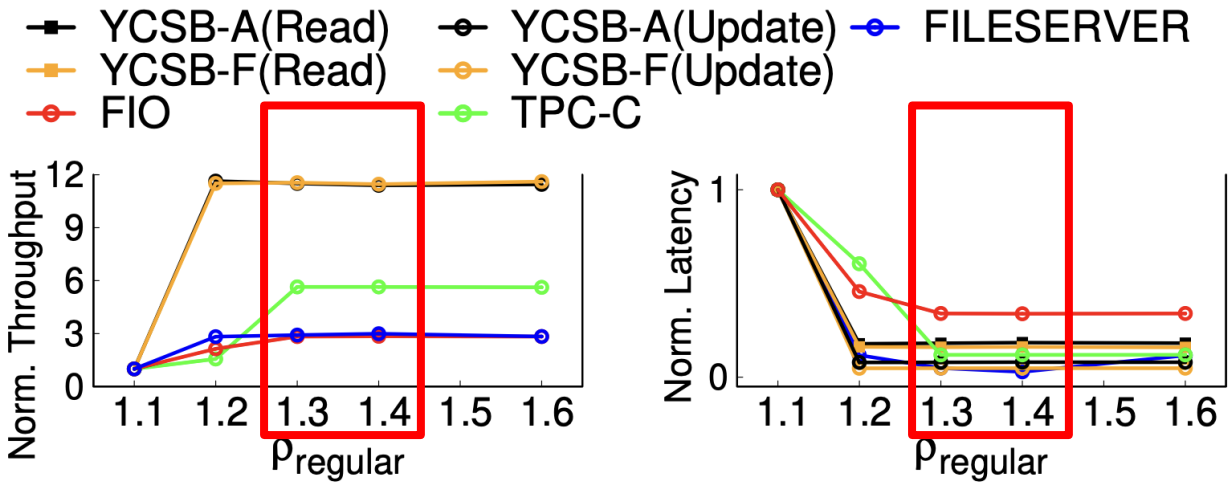
valid blocks + invalid blocks + free blocks ≤ storage capacity

In Virtual Overprovisioning,

valid blocks ≤ storage Capacity

Optimal Virtual Overprovisioning Degree





Evaluation

Evaluation

HW Setup

- CPU: Intel Xeon, 2.10 GHz, 40 cores
- Memory: 64 GByte
- Storage: 256 GByte emulated SSD (NVMeVirt, FAST '23)
  - Modeling Samsung 970 Pro (8 channels, 32 MByte superblock)
- OS: Linux 5.11.0

Workloads

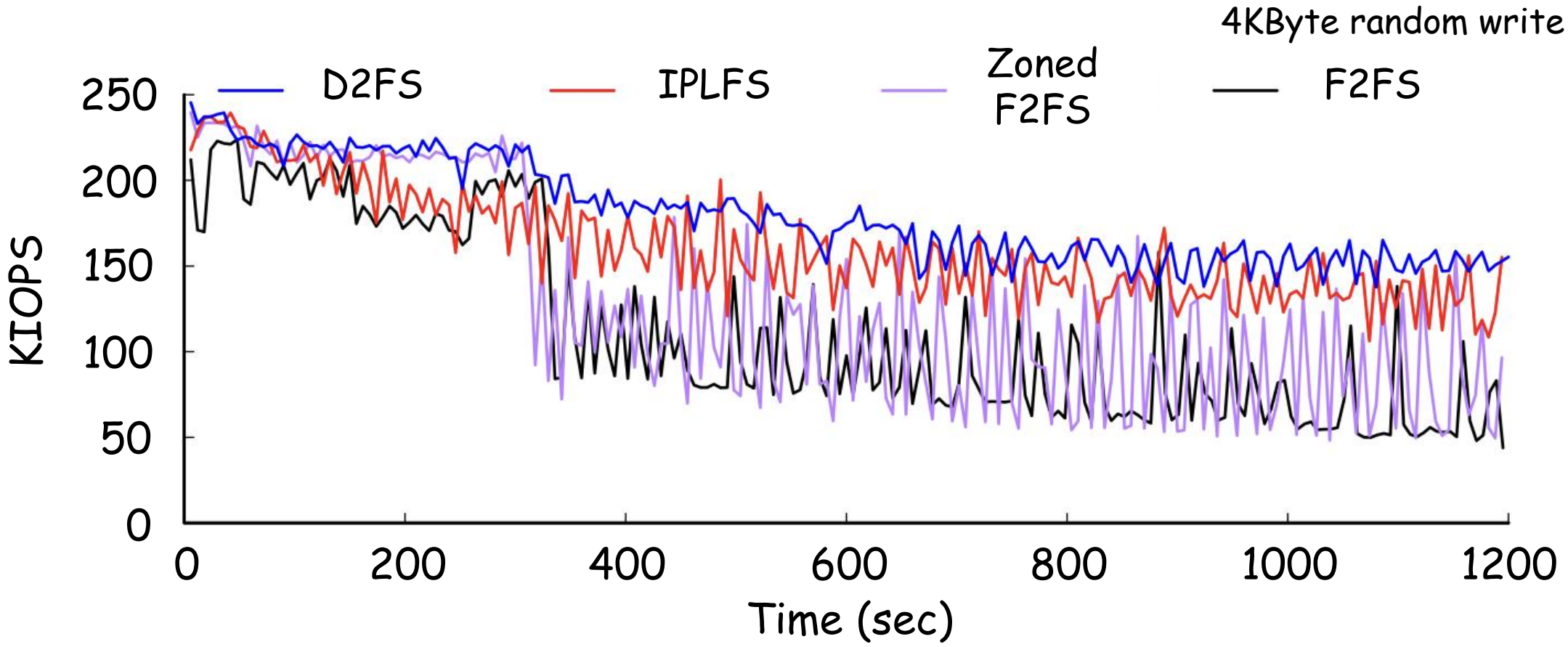
- FIO, TPC-C, YCSB-A, YCSB-F, Fileserver

Evaluation

Compared Filesystems

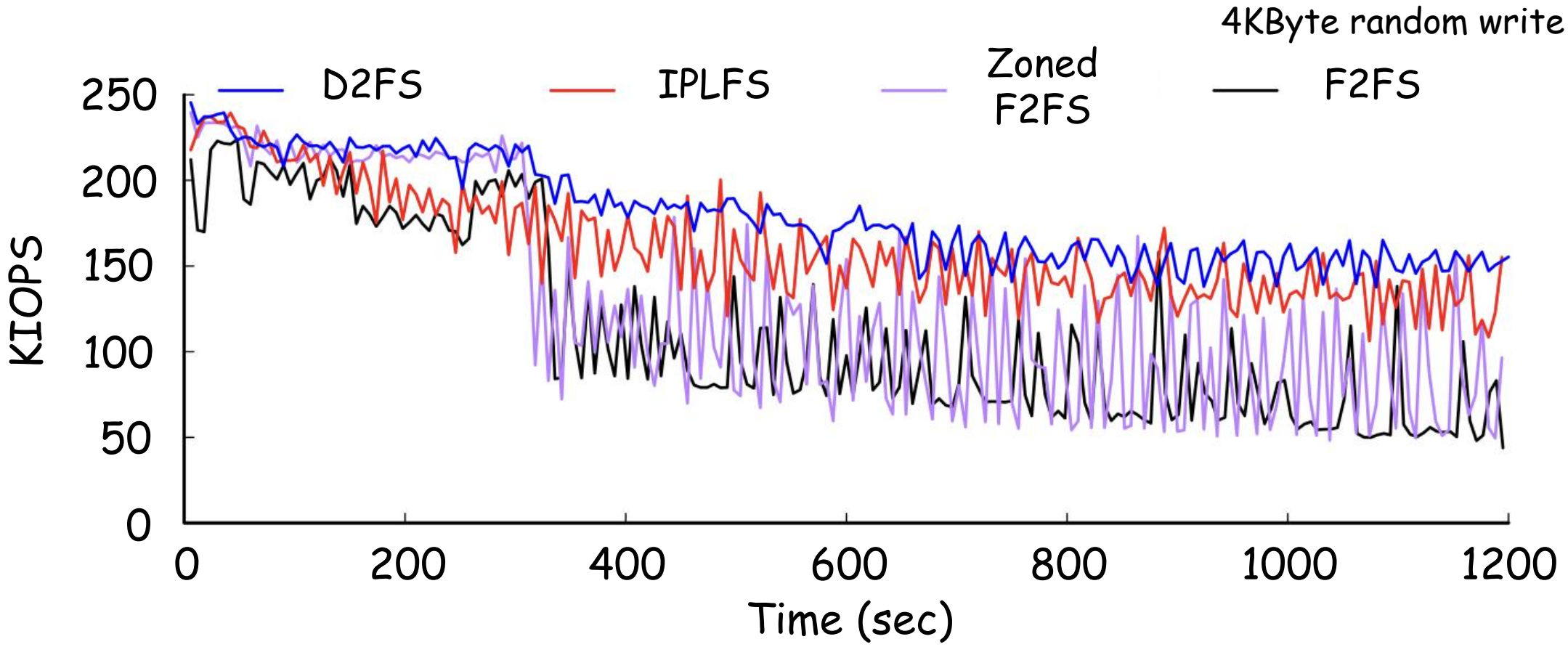
	Filesystem GC	Device GC
F2FS	O	O
Zoned F2FS (F2FS with ZNS support)	O	X
IPLFS	X	O
D2FS	X	O

Eliminating Filesystem GC



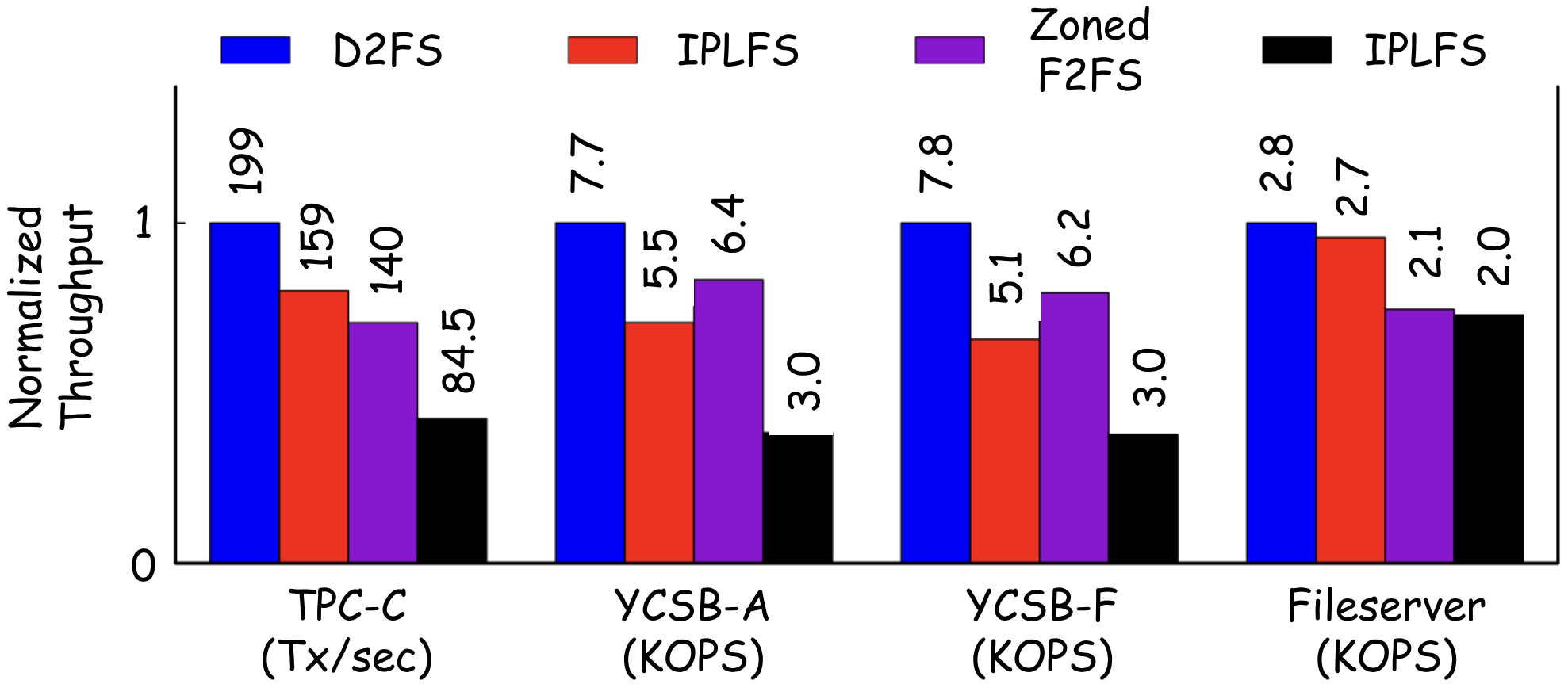
D2FS outperforms F2FS and zoned F2FS by 3× and 1.7×

Eliminating Filesystem GC



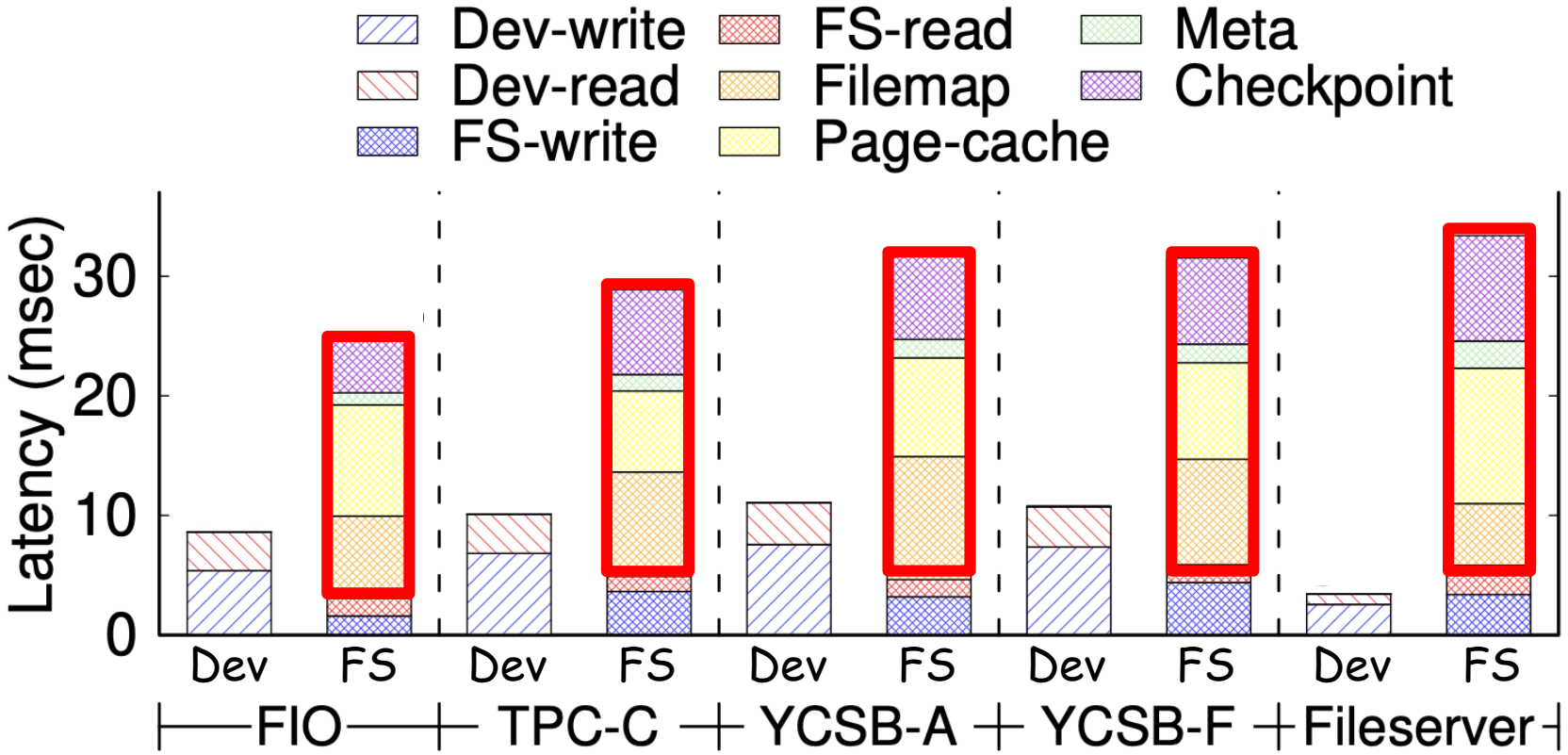
D2FS outperforms IPLFS by 15%,
since IPLFS suffers from the L2P mapping compaction overhead.

Macrobenchmark



D2FS outperforms F2FS by 2.6 ×, zoned F2FS by 1.4 × and IPLFS by 1.5 ×.

Device GC vs. Filesystem GC



Source of FS GC Overhead
- Checkpoint
- Page Allocation
- Filemap-Traversing

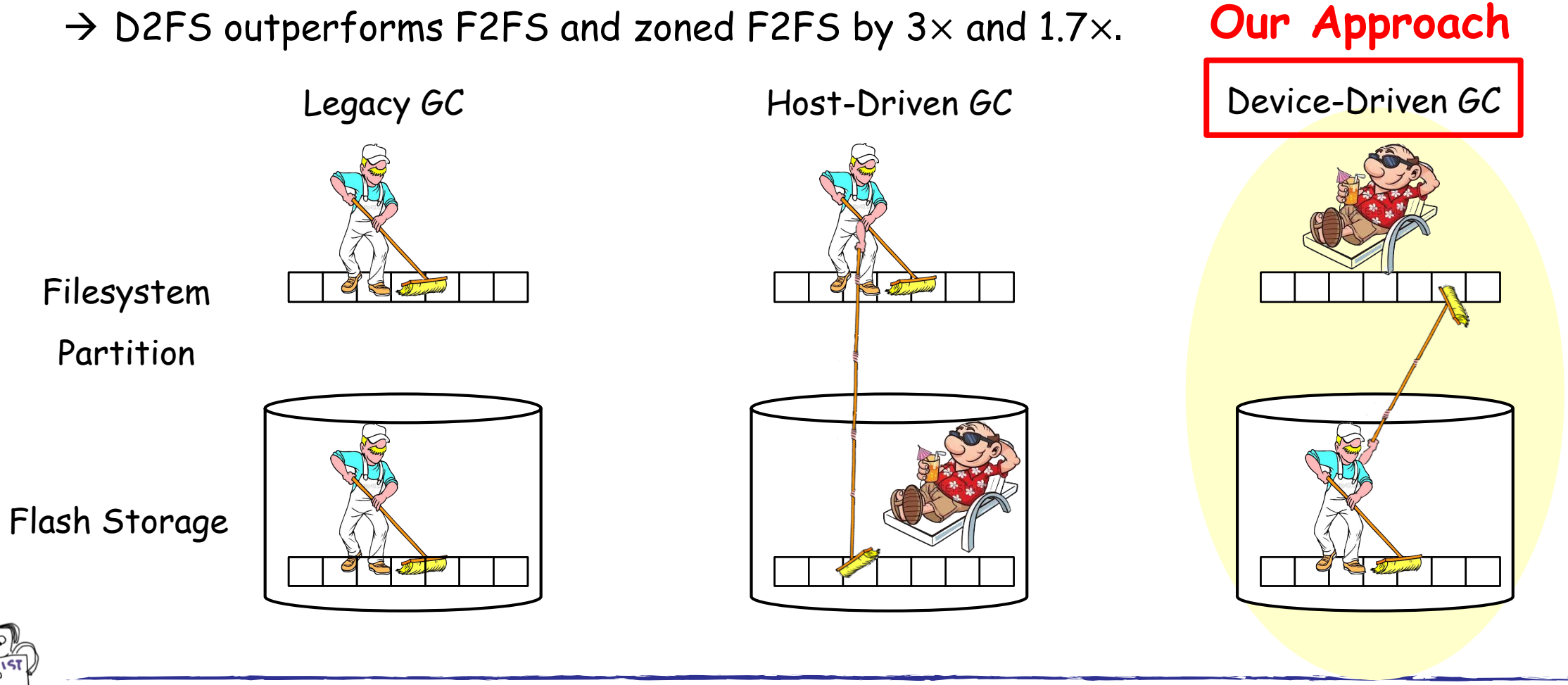
Filesystem GC takes 3 × longer than Device GC.

Conclusion

The filesystem GC is more expensive than the device GC.

→ We make the device GC clean the filesystem partition.

→ D2FS outperforms F2FS and zoned F2FS by 3× and 1.7×.

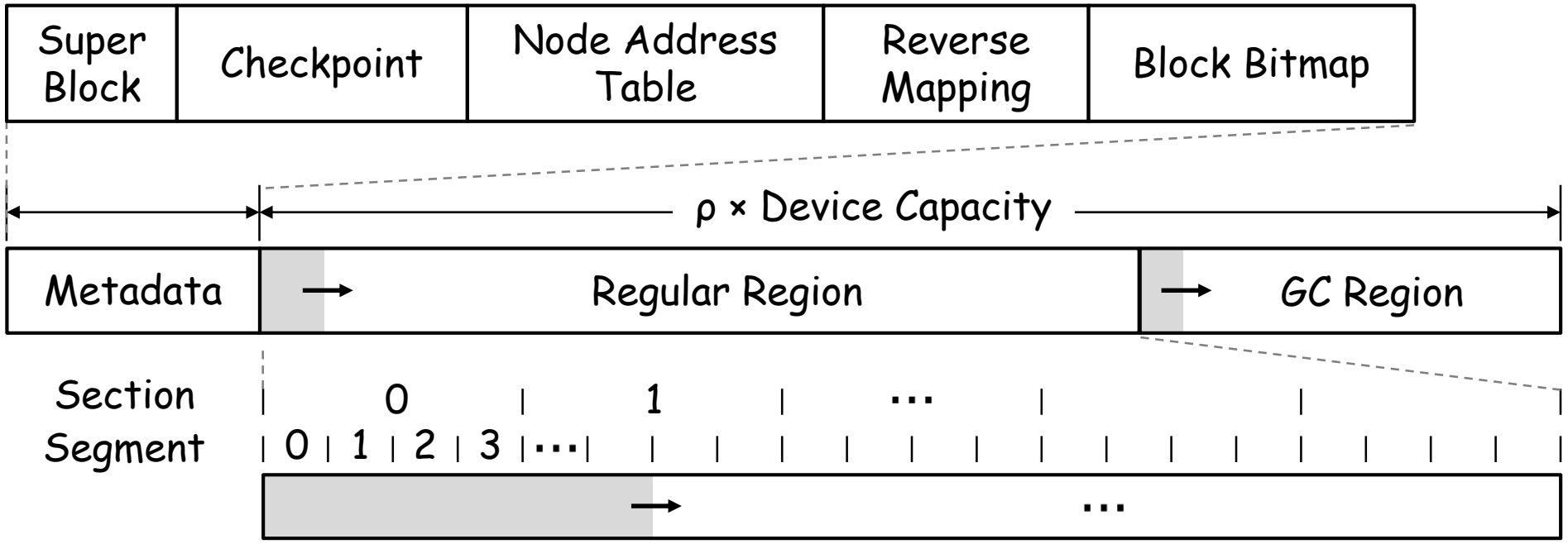


Question & Answer

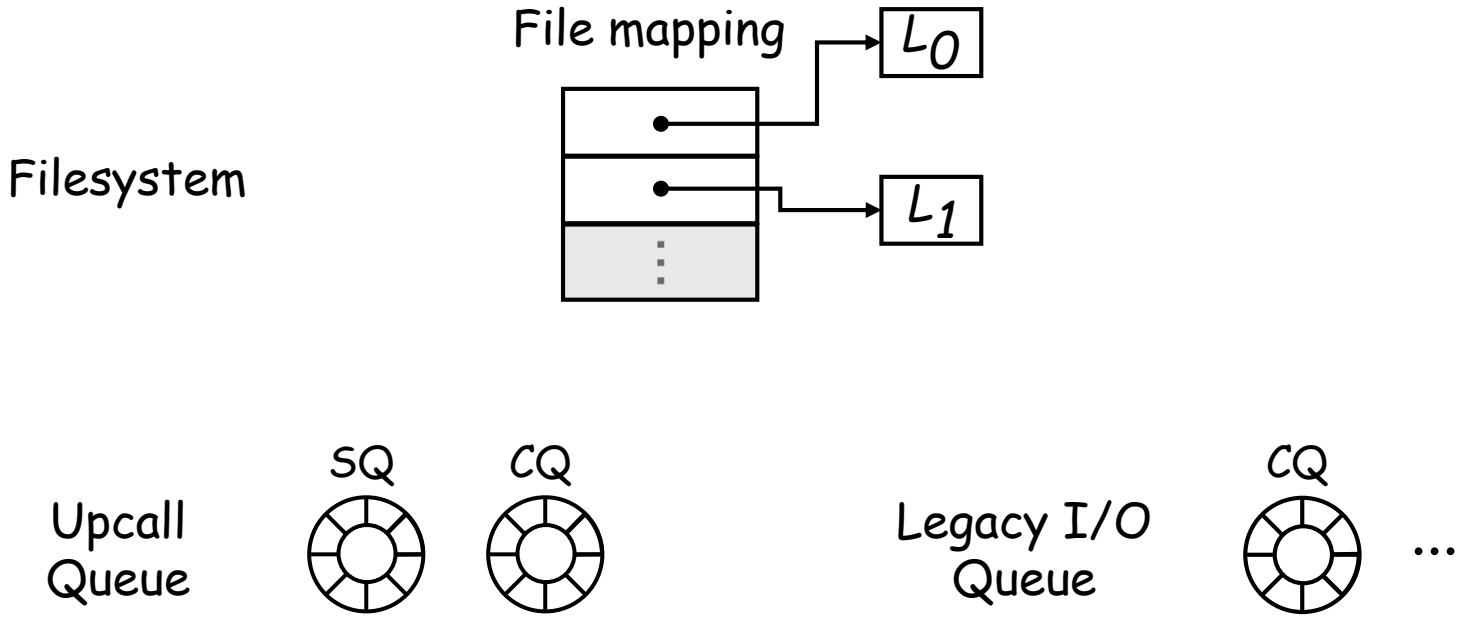
Appendix

Filesystem Partition Layout

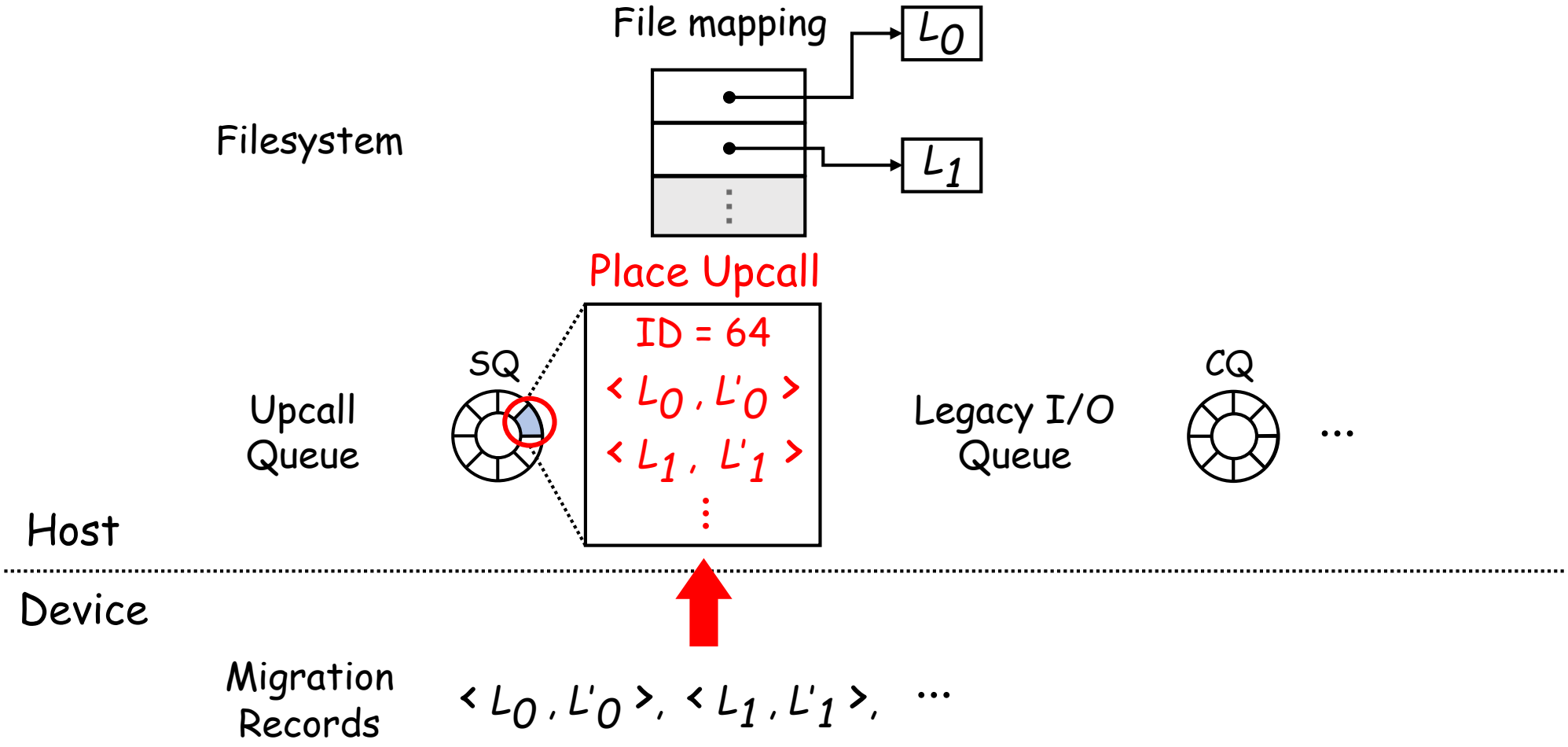
ρ : Virtual Overprovisioning Degree



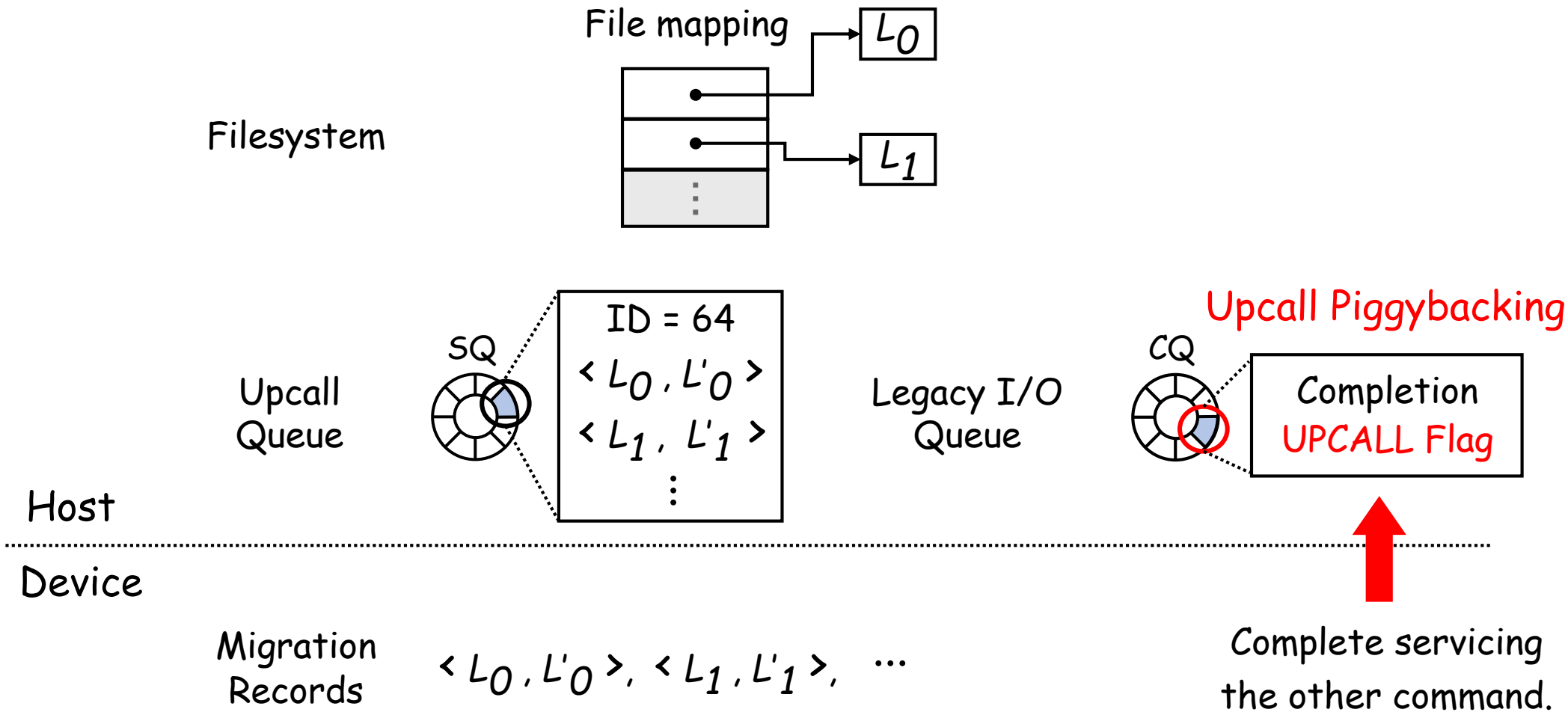
Processing the Migration Upcall



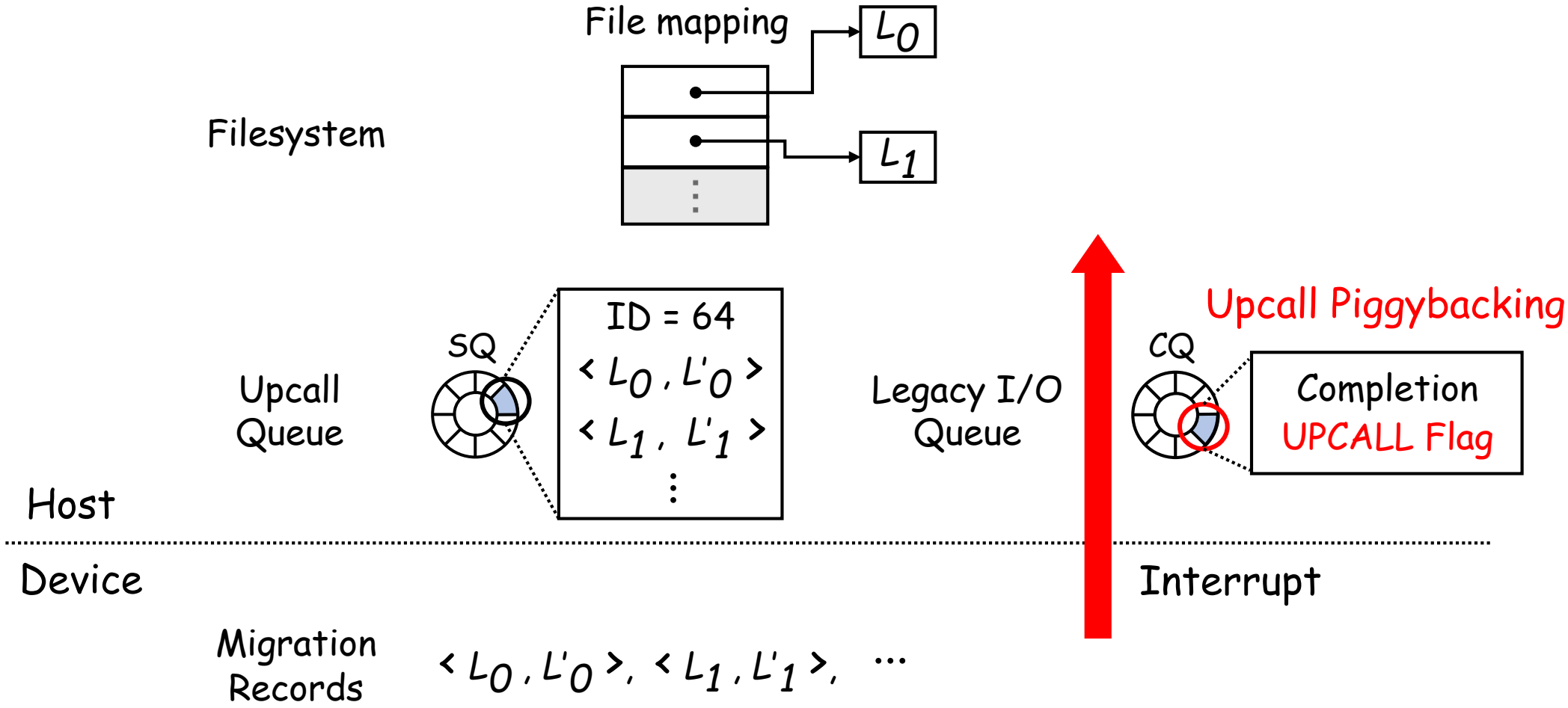
Processing the Migration Upcall



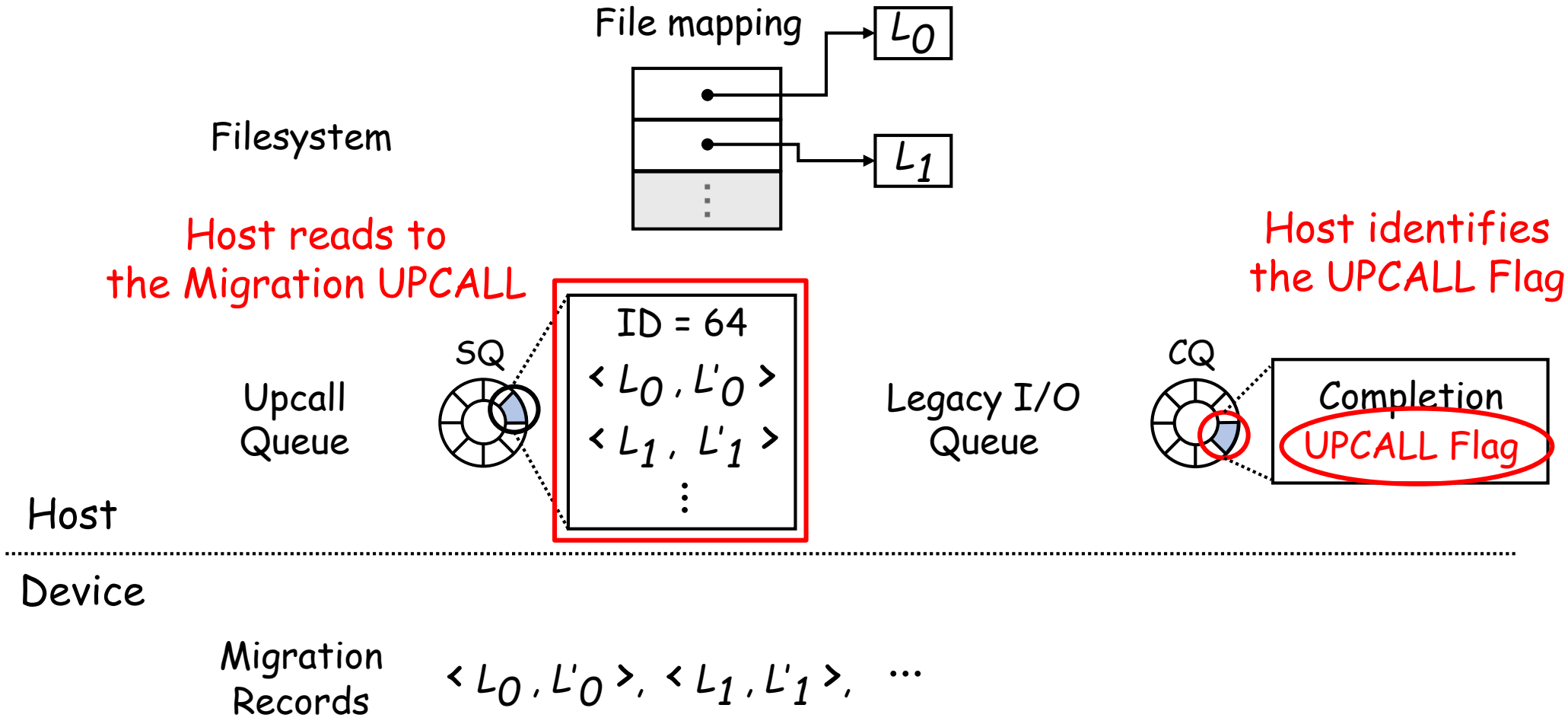
Processing the Migration Upcall



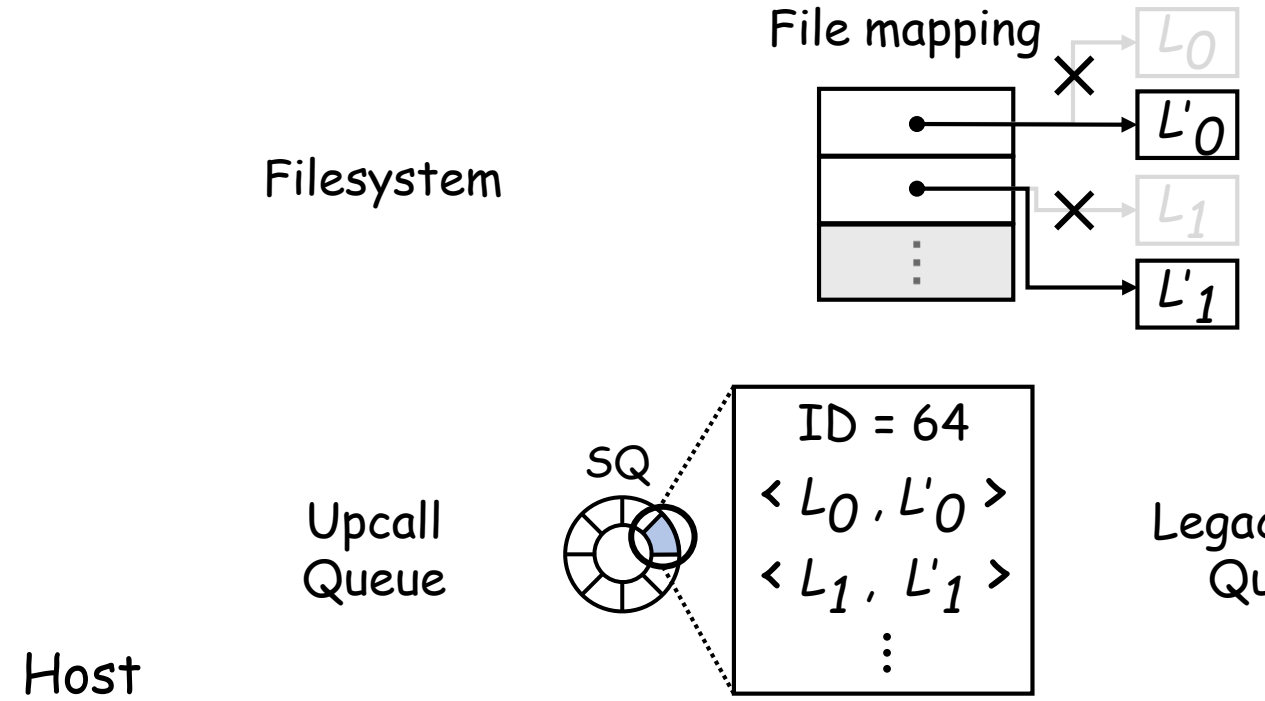
Processing the Migration Upcall



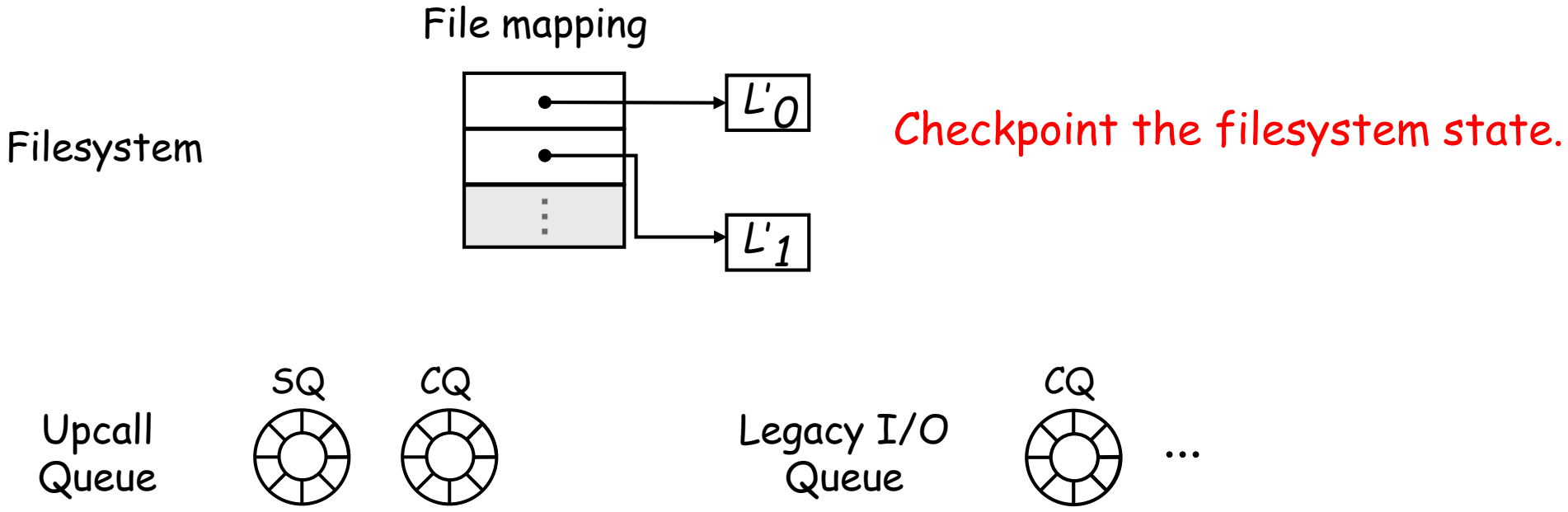
Processing the Migration Upcall



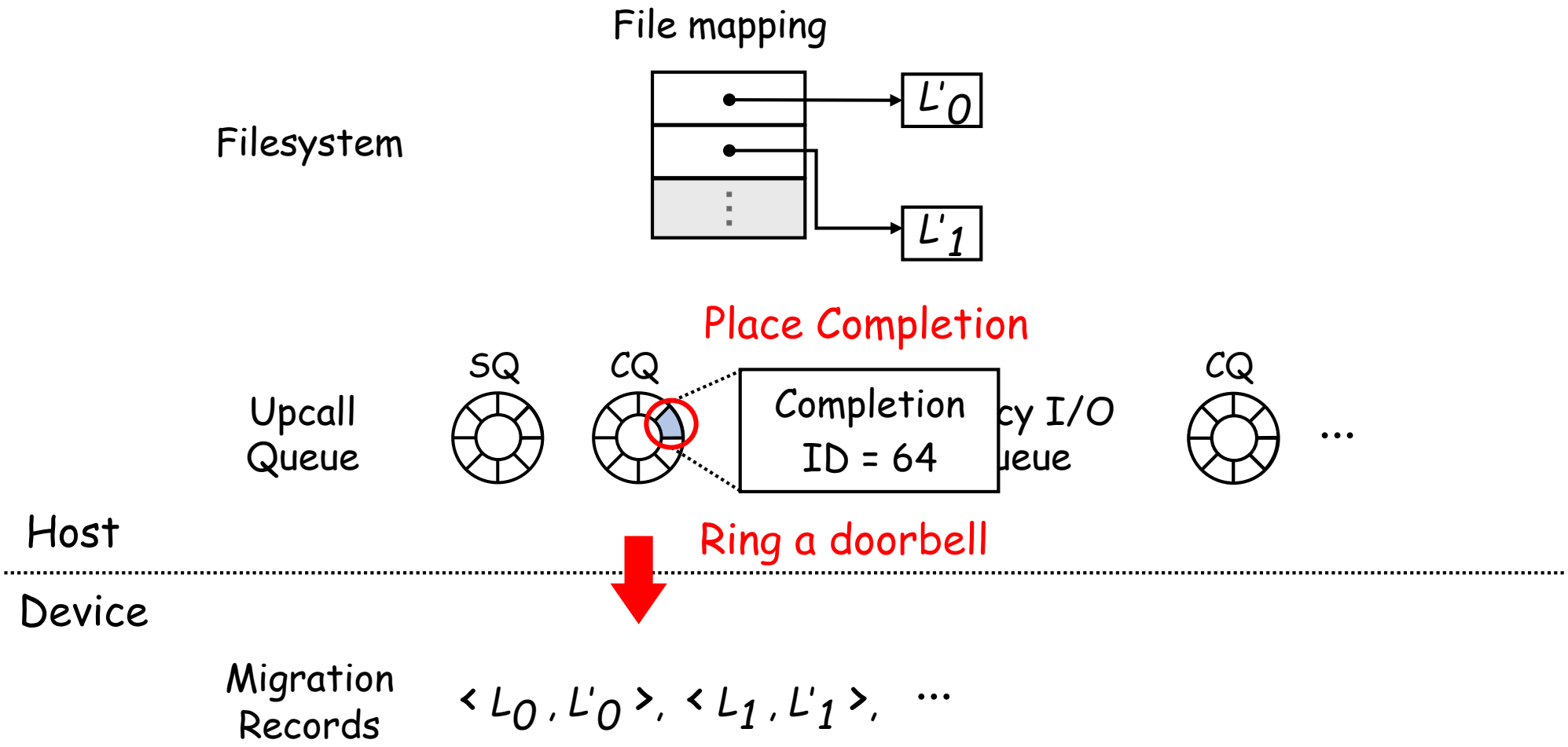
Processing the Migration Upcall



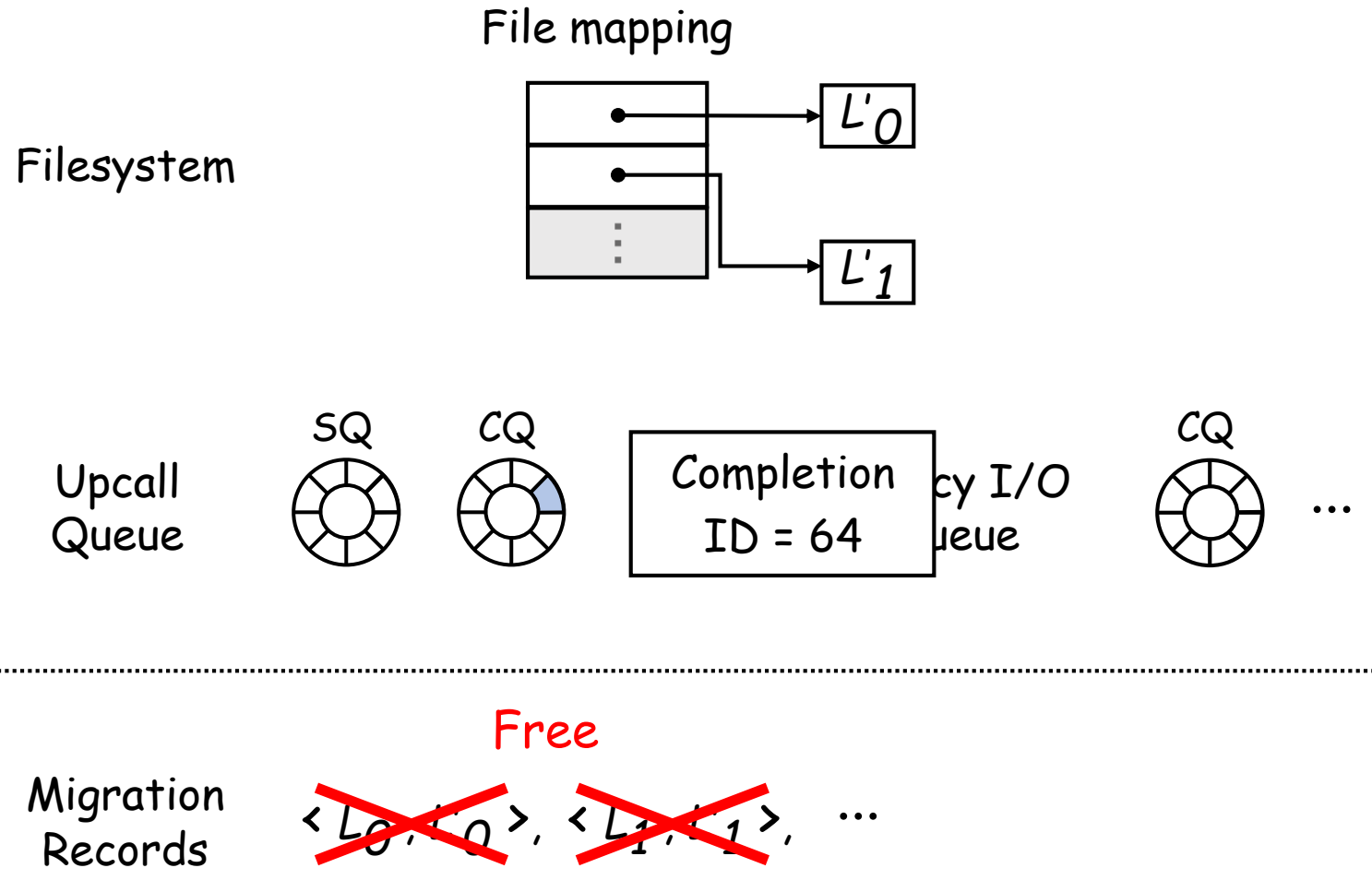
Processing the Migration Upcall



Processing the Migration Upcall



Processing the Migration Upcall



Recording Updated LBAs

Device Memory

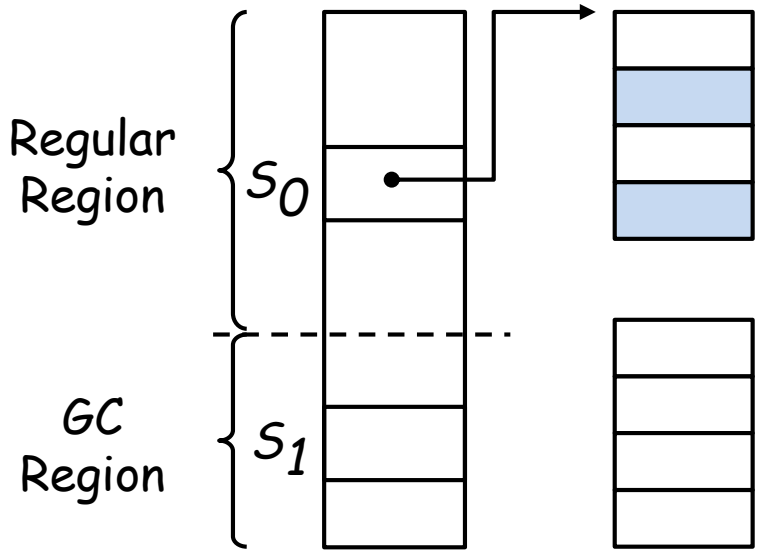
L2P Mapping

Regular Region

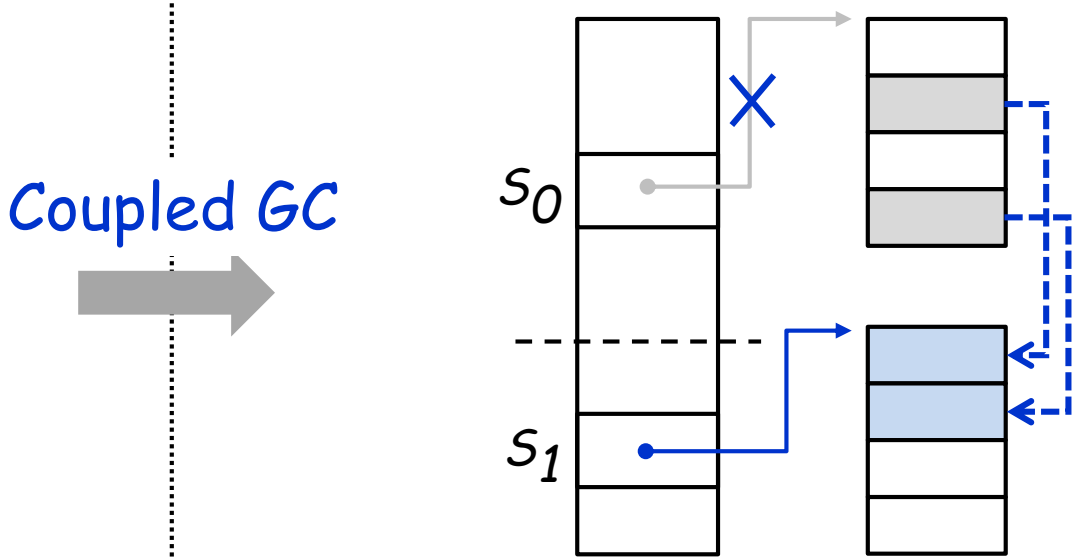
S_0

GC Region

S_1



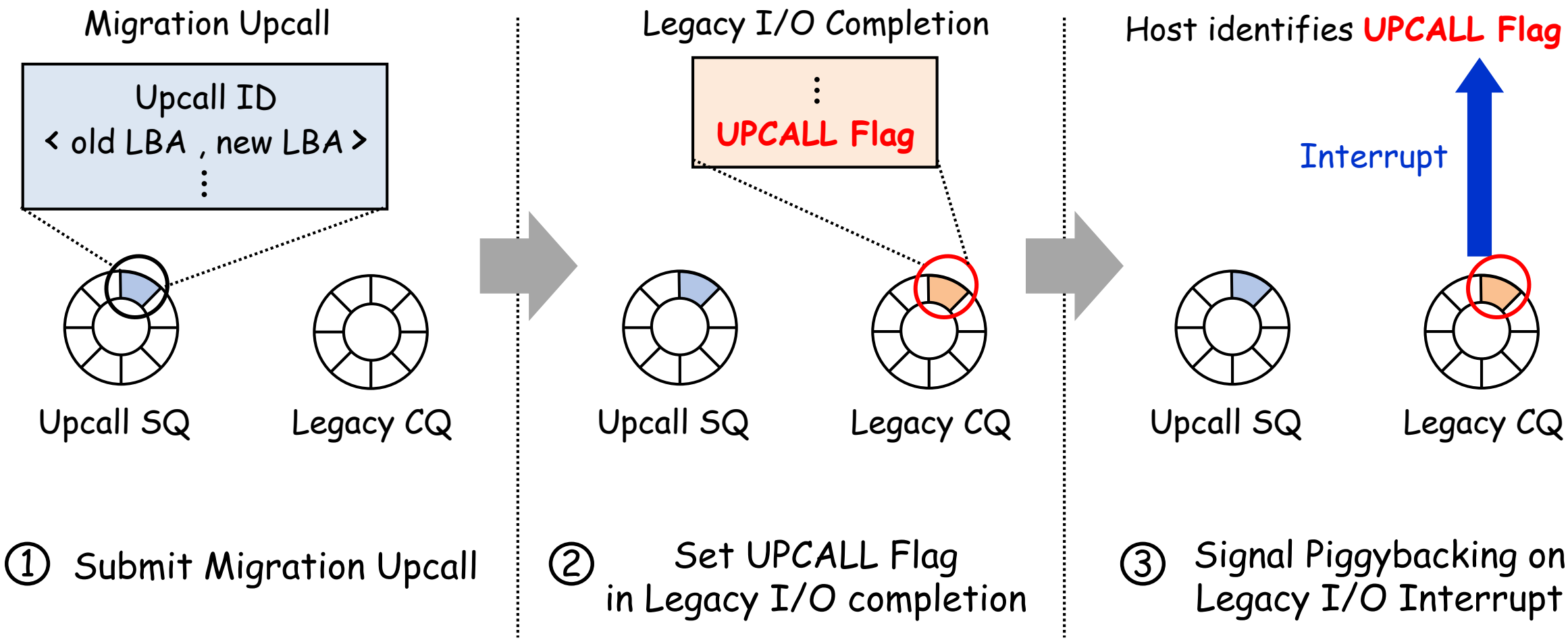
Coupled GC



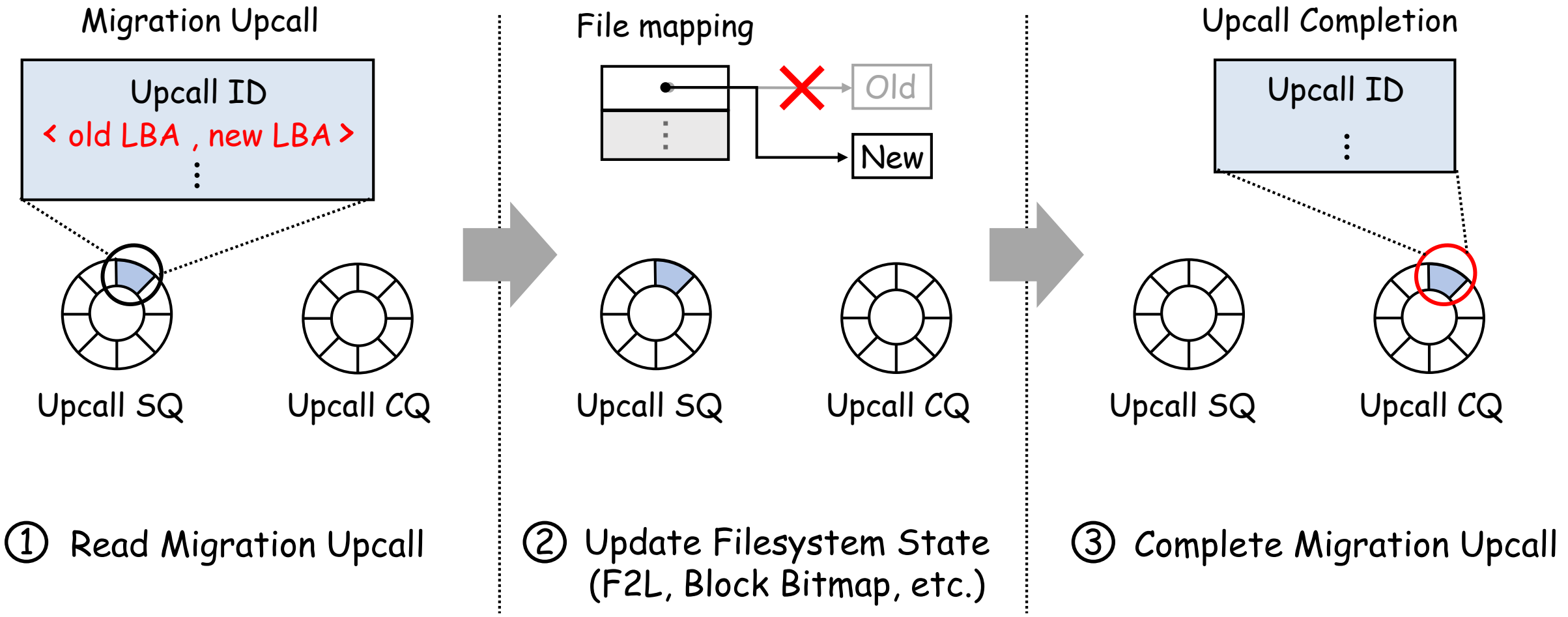
Recording < old LBA, new LBA>

$\langle S_0 + 1, S_1 + 0 \rangle$
$\langle S_0 + 3, S_1 + 1 \rangle$

Upcall Piggybacking



Processing Migration Upcall



FTL Memory Overhead

1.5 × larger than conventional page mapping FTL

Filesystem	F2FS	IPLFS	zF2FS	D2FS
Memory (MByte)	256.0	392.0	10.4	27.2

IPLFS is hard to be deploy in the real world with limited power and hardware budget.

D2FS uses the superblock-granularity L2P mapping. → Less FTL memory overhead.