Advanced I/O Stack for Zone-based Mobile Flash Storage

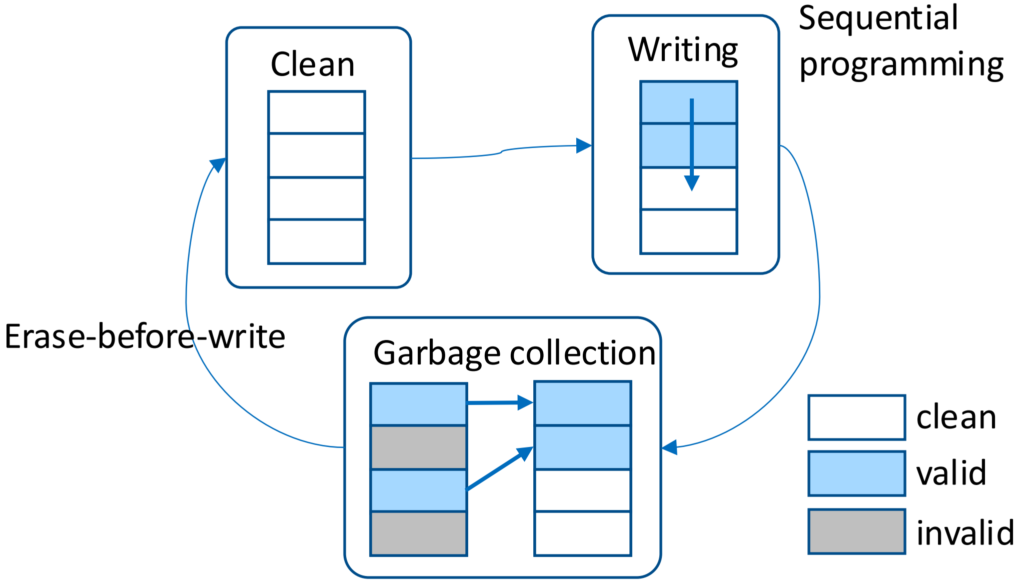
2024/10/25

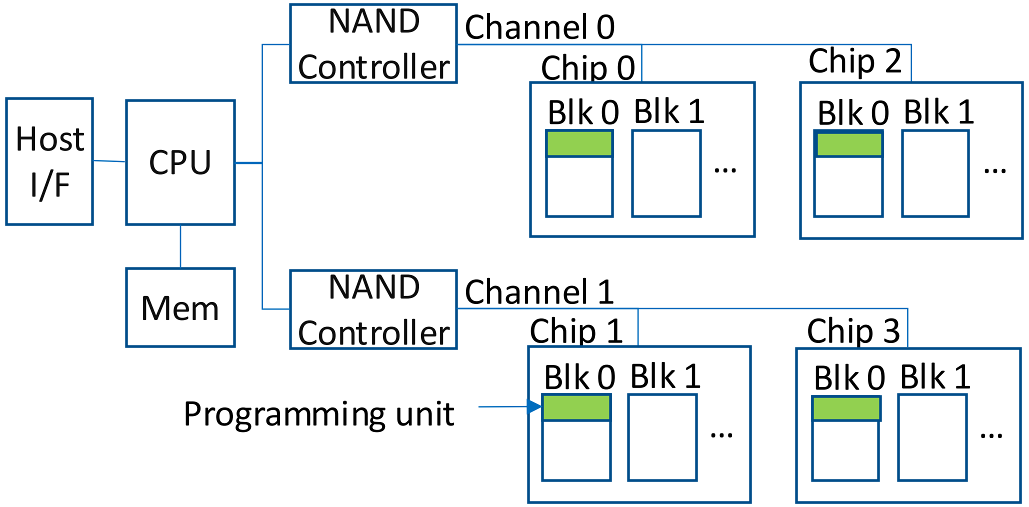
Joo-Young Hwang, Seokhwan Kim, Daejun Park, Yong-gil Song,
Junyoung Han, Seunghyun Choi, Sangyeun Cho (Samsung Electronics), Youjip Won (KAIST)

Background

Conventional SSD (Block Interface)

- Flash memory has sequential programming and “erase-before-write” characteristics.
- Logical-to-physical (L2P) mapping and garbage collection provides block interface for flash memory.
- Data striping over multiple chips for parallel operation.



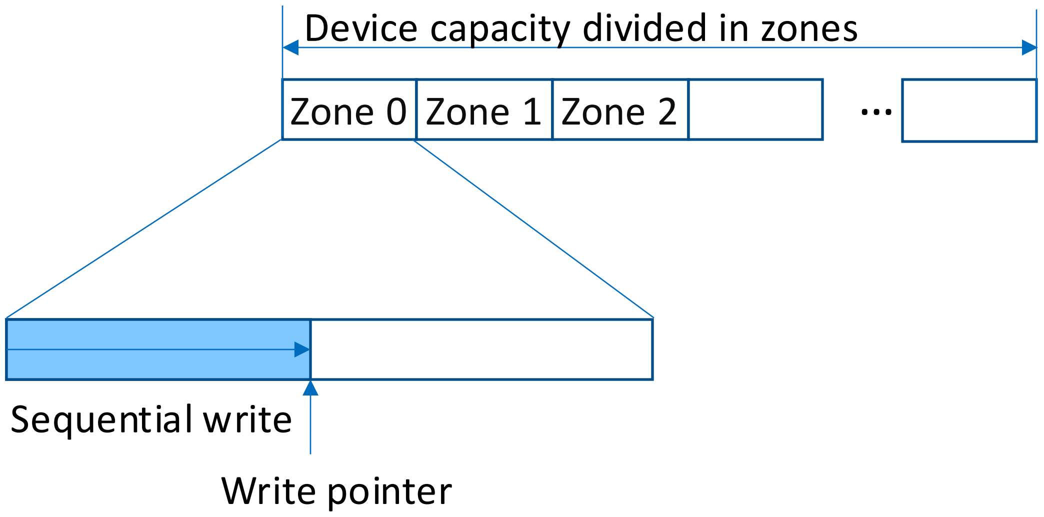


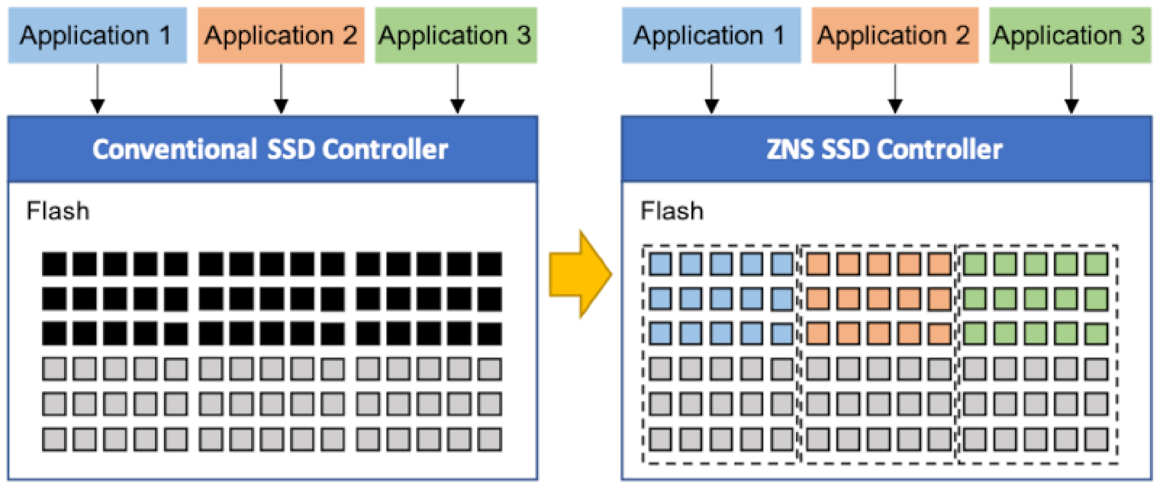
- **Superblock**: the set of flash blocks where data is striped
- **Superpage**: the set of programming units on the same offset of the superblock

- Issues
  - Memory cost for L2P mapping table (← page mapping)
  - Write amplification (← data with different lifetimes are mixed)

Zoned Namespace SSD

- Sequential write constraint → coarse-grain (zone) mapping → reduced L2P map
- Host controls data placement → removes device garbage collection
  - Host is responsible for data separation to reduce write amplification.





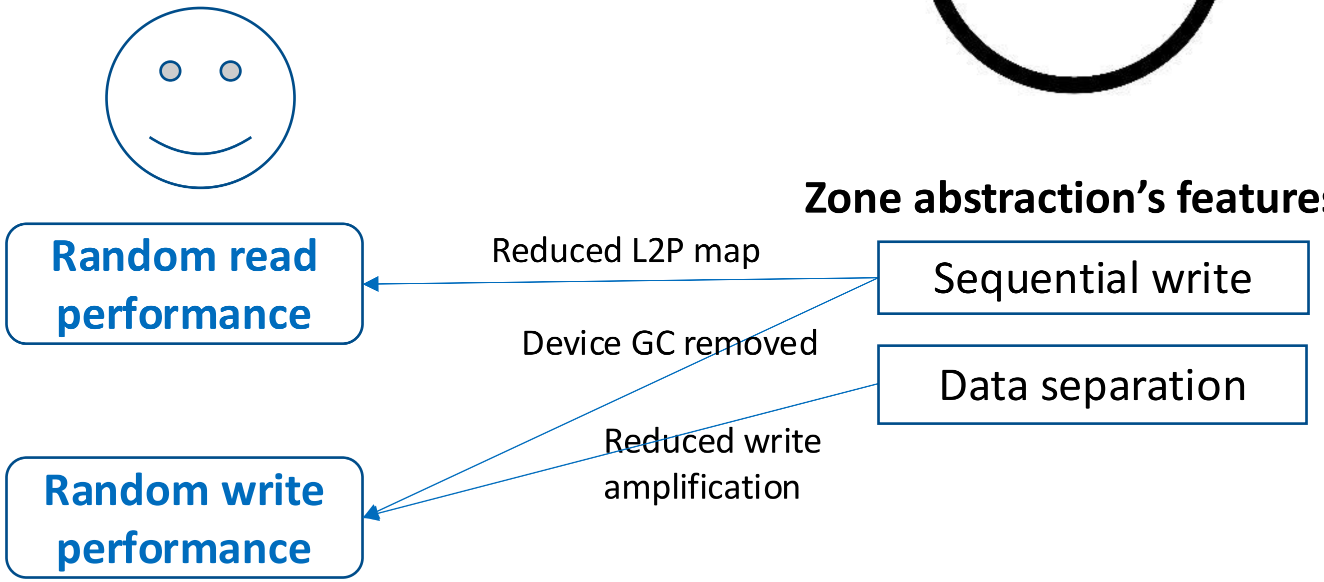
Regular SSD: Device controls data placement

ZNS SSD: Applications control data placement in zones

Does mobile storage benefit from zone abstraction?

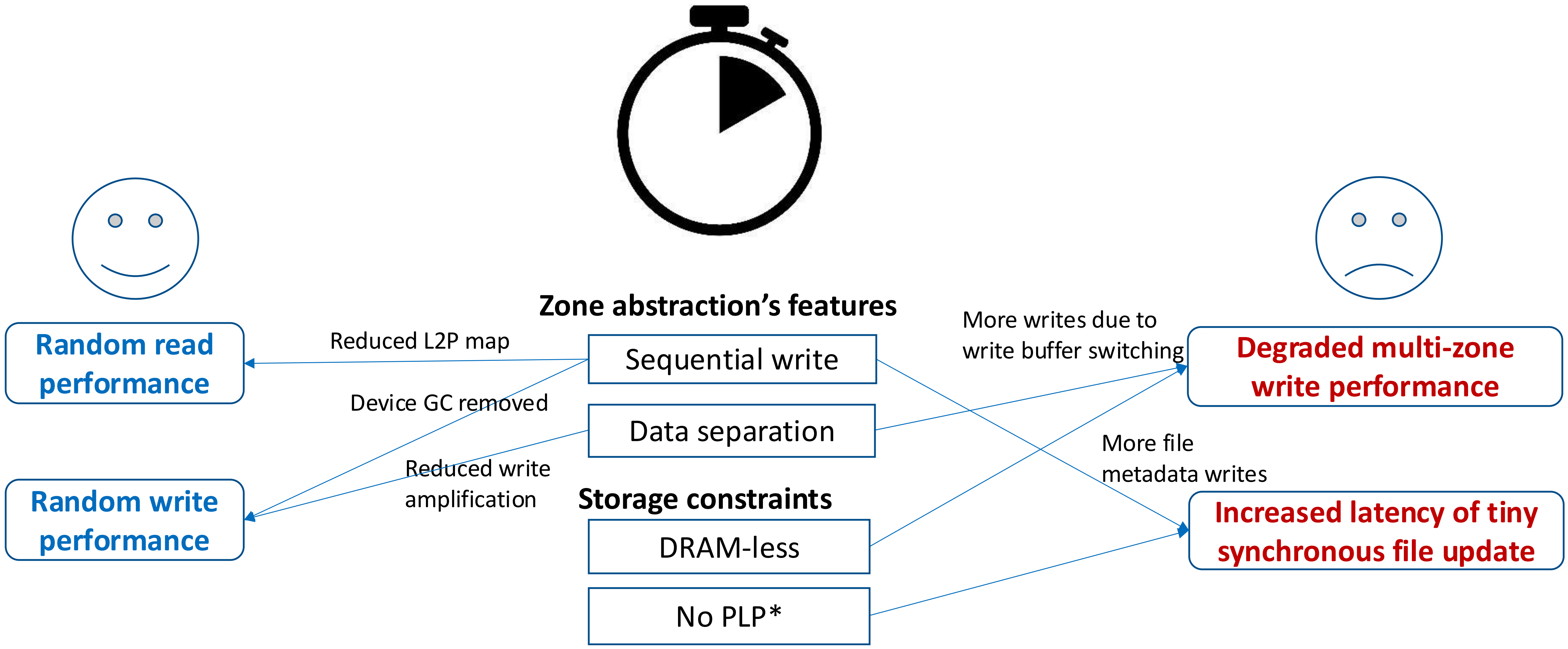
Responsiveness is critical in mobile devices.

Zone abstraction's features



Does mobile storage benefit from zone abstraction?

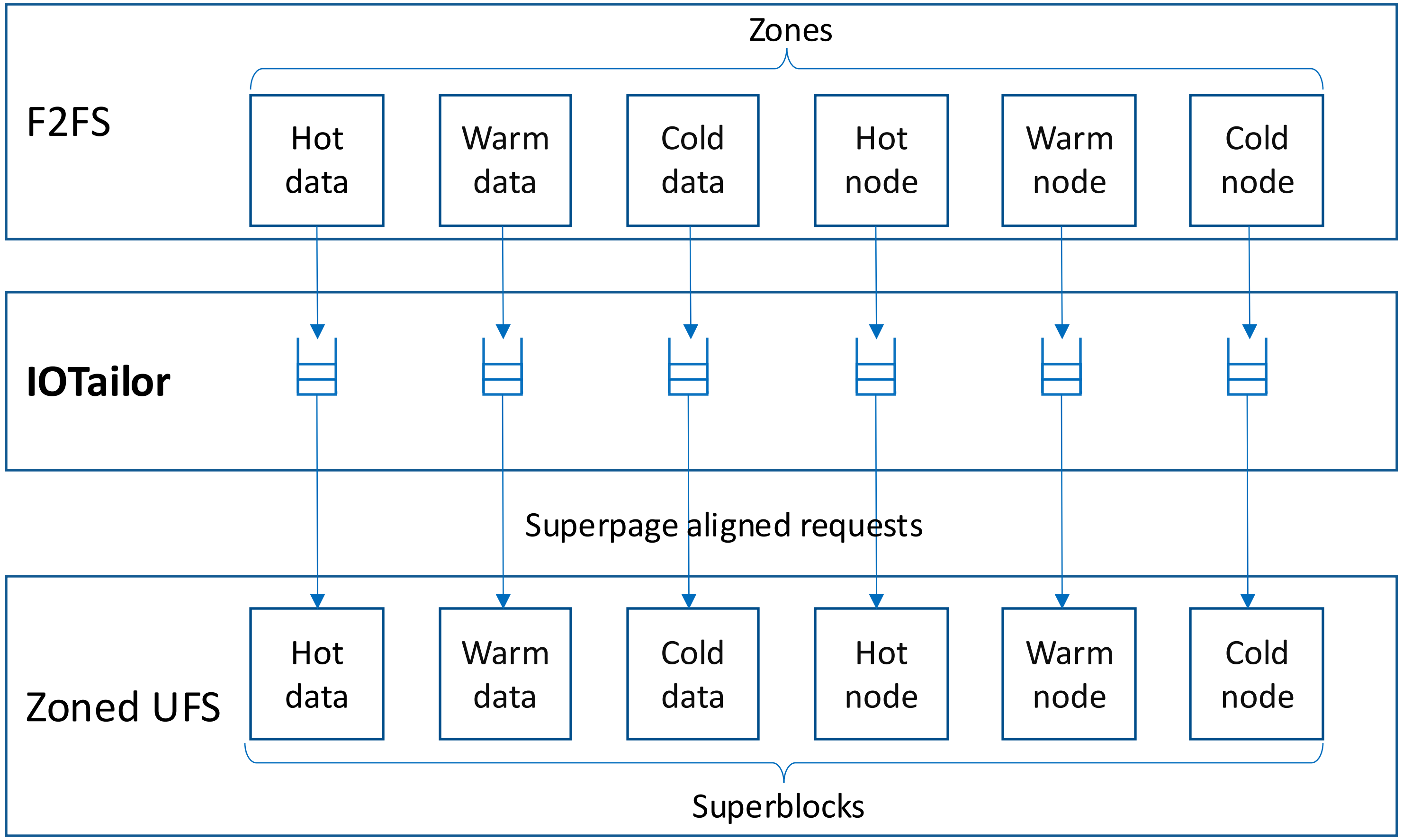
Responsiveness is critical in mobile devices.



*PLP: Power loss protection

ZMS (Zoned Mobile I/O Stack)

- Utilizing F2FS*, data is separated according to six temperature types.
- Techniques to address the challenges: **IOTailor, budget-based in-place update**
- Optimization techniques: copy offloading, multi-granularity mapping (not covered in this talk)



- **Budget-based in-place update**
- Copy offloading to reduce GC cost
- Multi-granularity mapping

*F2FS: A new file system for flash storage, Lee et al. USENIX FAST '15

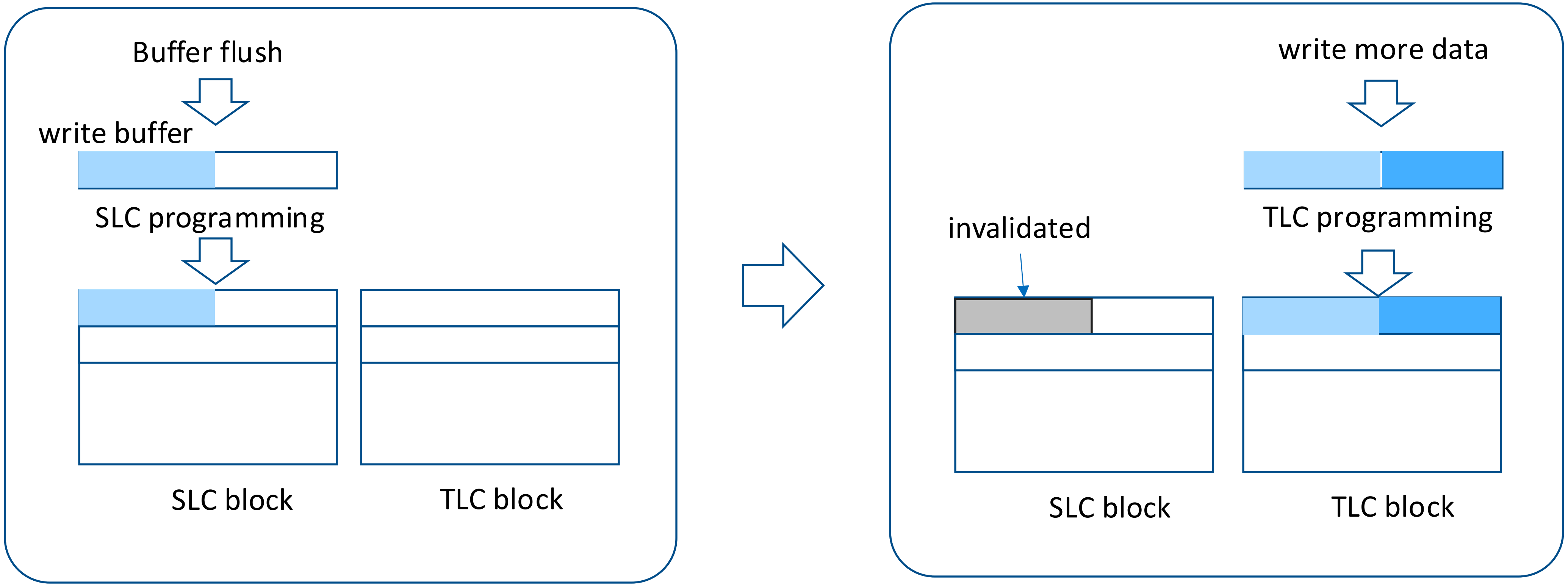
Talk Outline

- Challenge #1: Multi-zone write performance
- Challenge #2: Latency of Tiny Synchronous File Update
- Evaluation
- Conclusion

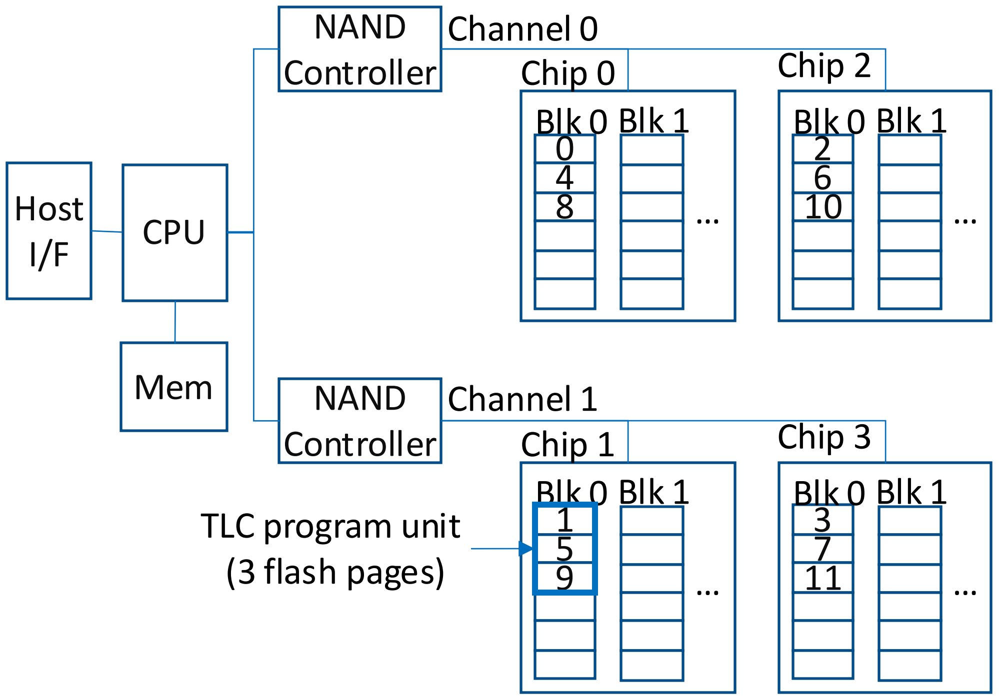
Challenge #1:
Multi-zone Write Performance

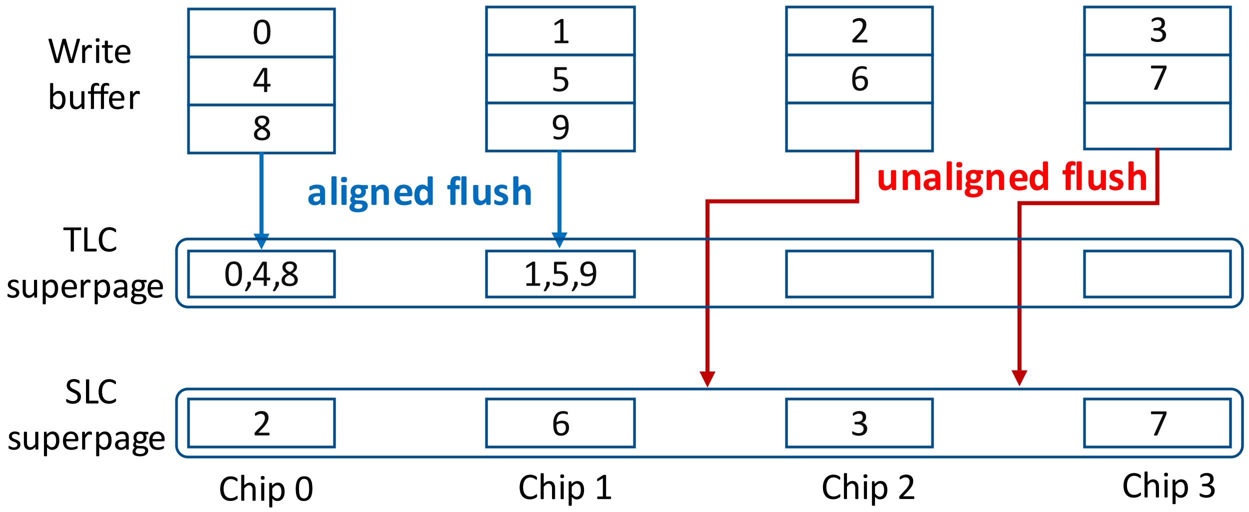
SLC Buffering to Handle Unaligned Buffer Flushes

- Unaligned buffer flush: flush data that is smaller than TLC programming unit.
- Backup data to SLC, later migrate data to TLC
- Side-effect: double writes



Example of Unaligned Buffer Flush Handling

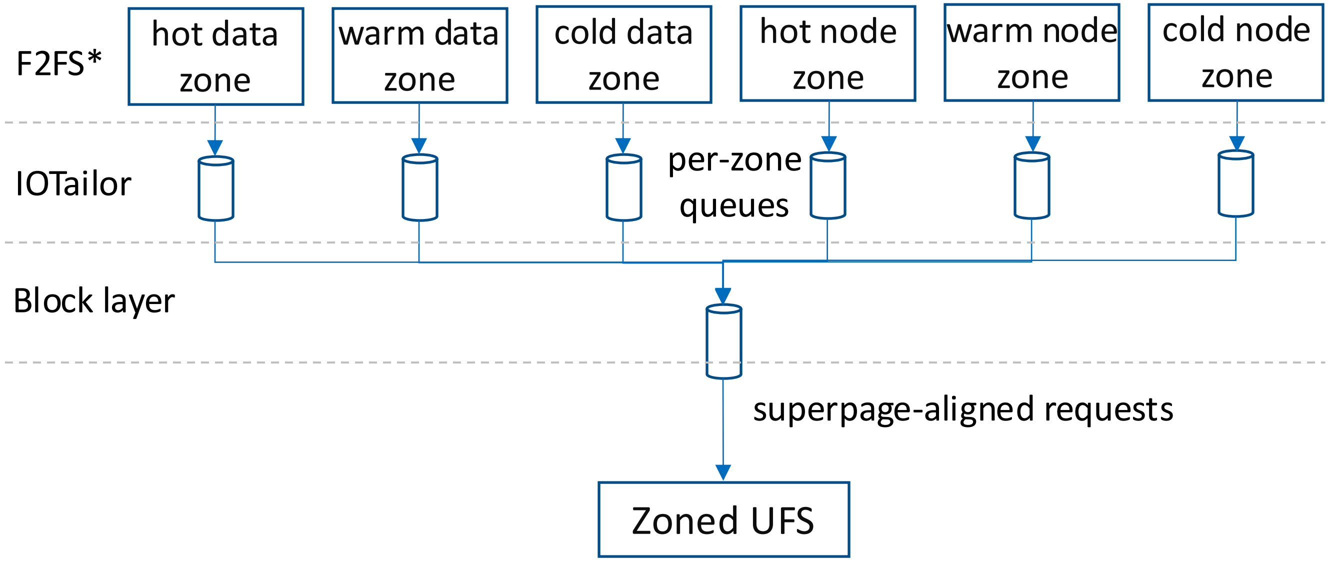


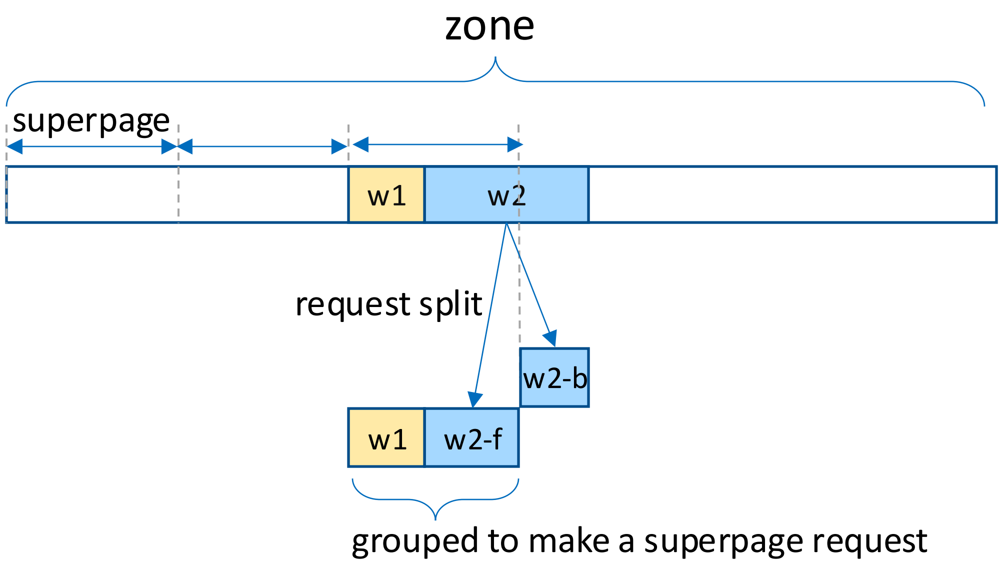


Stripe unit: 32KiB (16 KiB page x 2 planes)
Superpage = 12 stripe units (384 KiB)

IOTailor

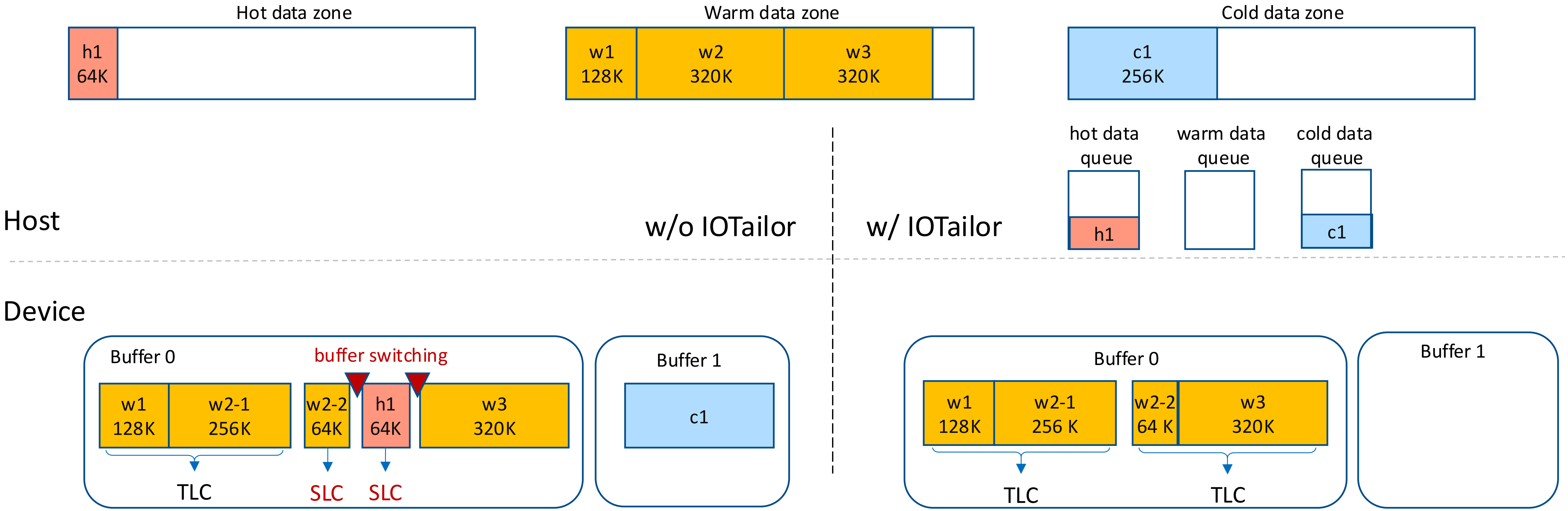
- Superpage-aligned request is good for parallelism and avoids unaligned buffer flushes.
- For each zone, IOTailor transforms requests to superpage-aligned requests
- Request split & request grouping in the per-zone queues





Example of Writing to Multiple Zones

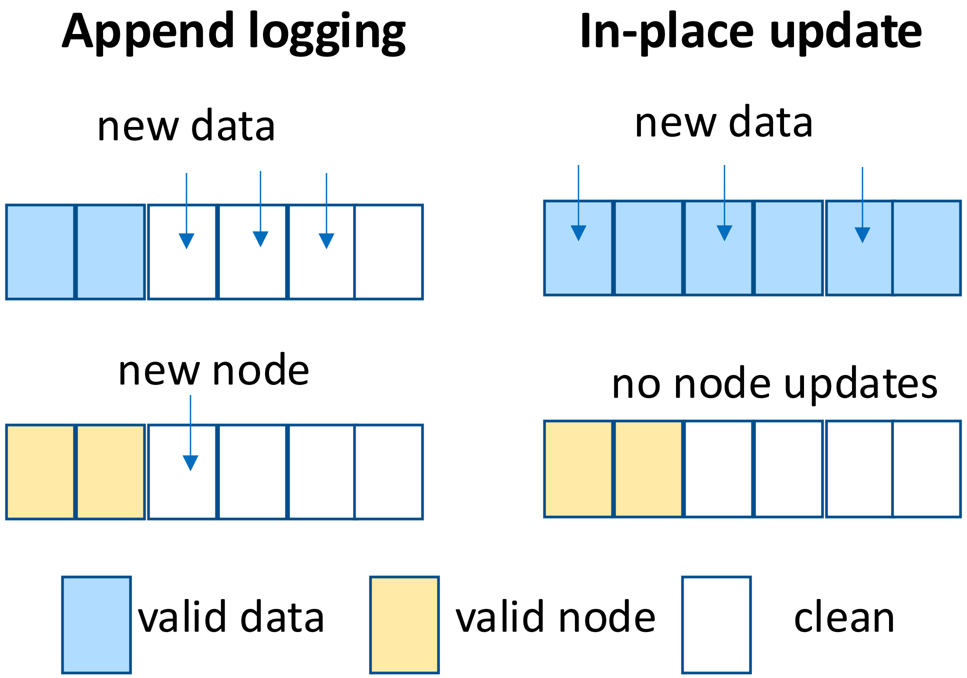
Command order: w1 – c1 – w2 – h1 – w3



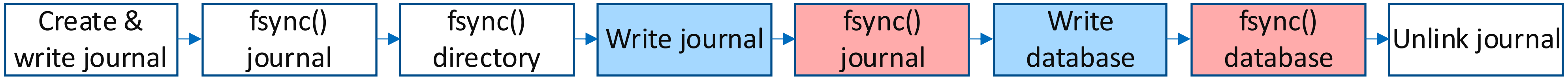
Challenge #2:
Latency of Tiny Synchronous File Update

F2FS write optimization does not work for zoned device

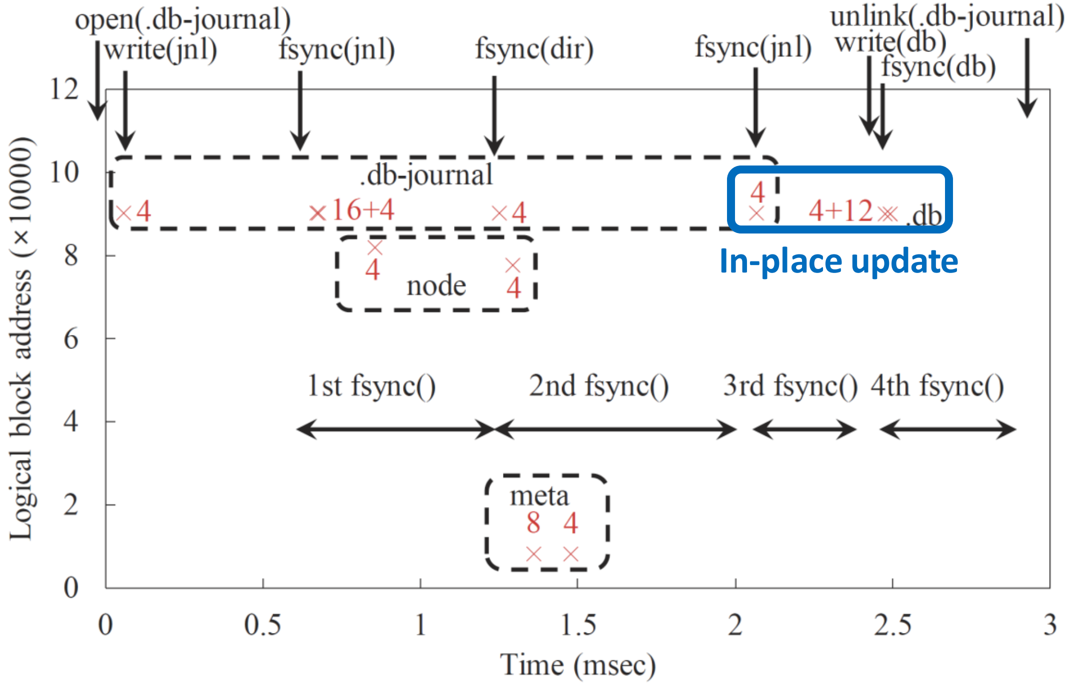
- Tiny synchronous file update is latency critical.
- For conventional block device, F2FS uses in-place update policy for tiny (< 32KiB) synchronous file update.
- In-place update policy cannot be used on zoned devices.



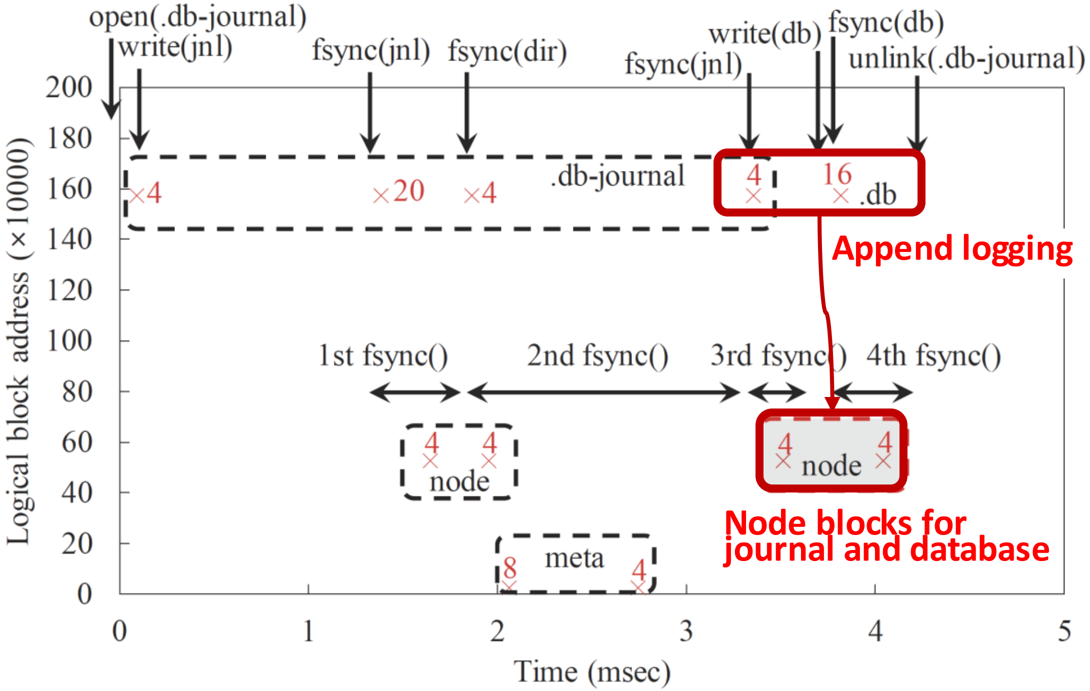
I/O Pattern of SQLite insert() Transaction



<Block device>

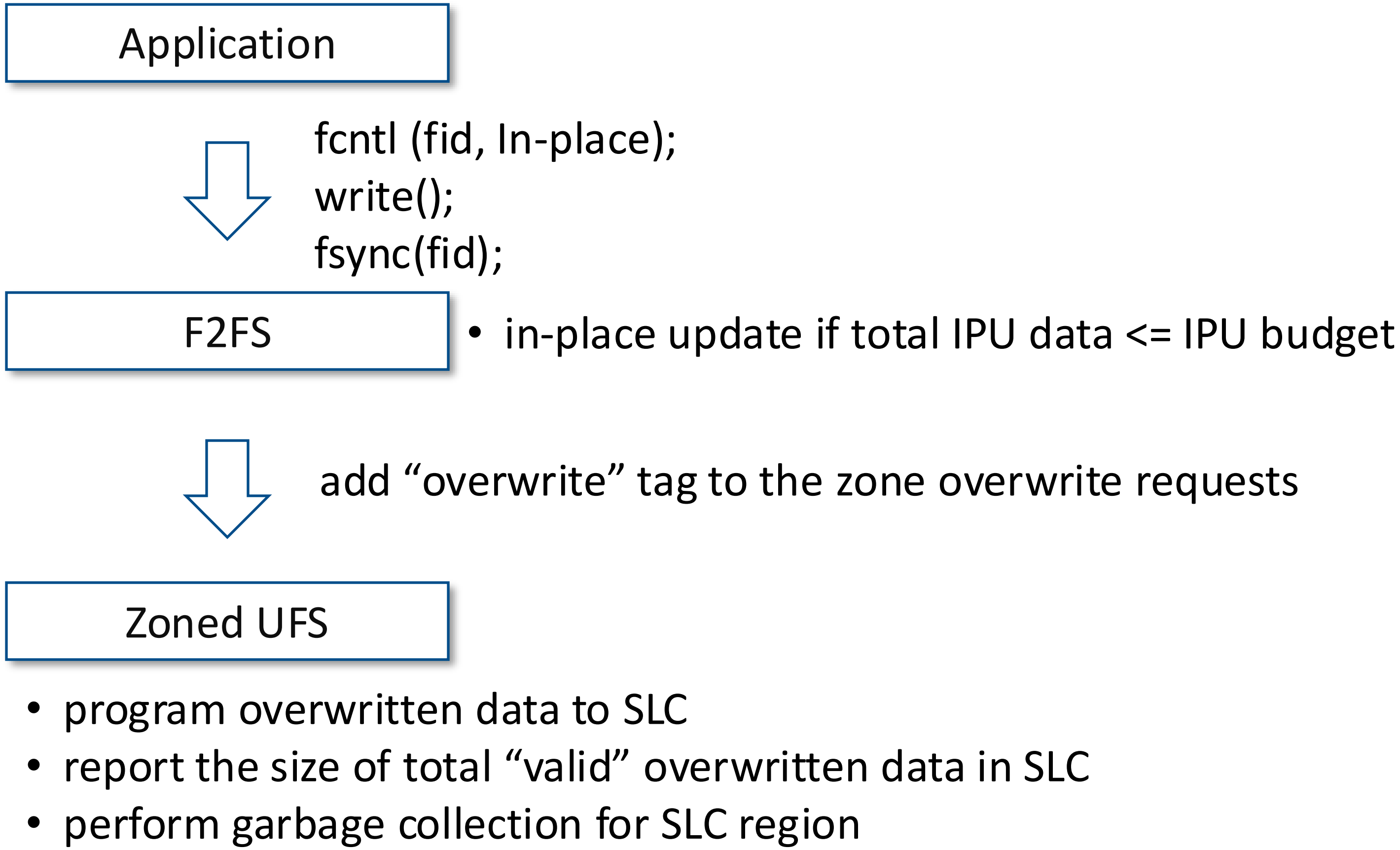


<Zoned device>



Budget-based In-Place Update

- Allow in-place update for files as per the application request
- Device writes the In-place updated data into SLC blocks.
- Cap total valid data size in SLC blocks for efficient garbage collection.

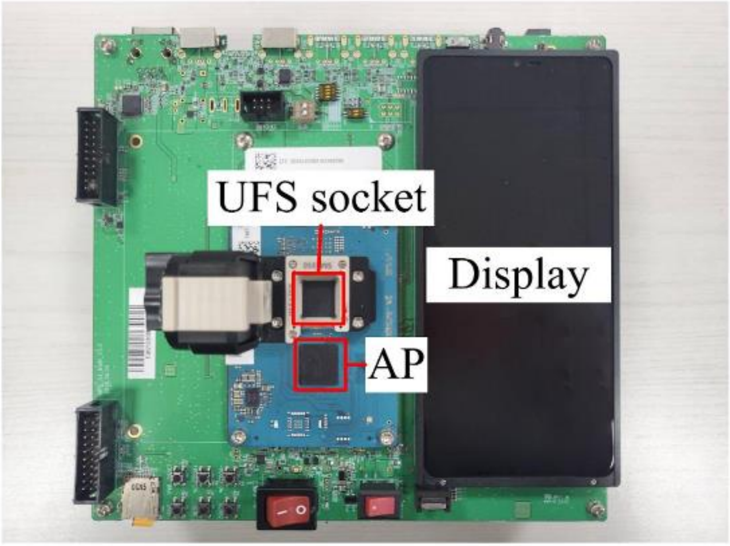


Evaluation

- How much are the benefits of zone abstraction for mobile storage?
- are the challenges addressed well?

Evaluation Setup

- Host platform: SM8350 (8 cores), 12GiB DRAM, Android 11, Linux 5.4
- Zoned UFS: 128GiB, UFS 2.1
  - zone size: 138MiB
  - a conventional logical unit for F2FS meta area
- Baseline: the same device with a firmware that supports legacy block interface

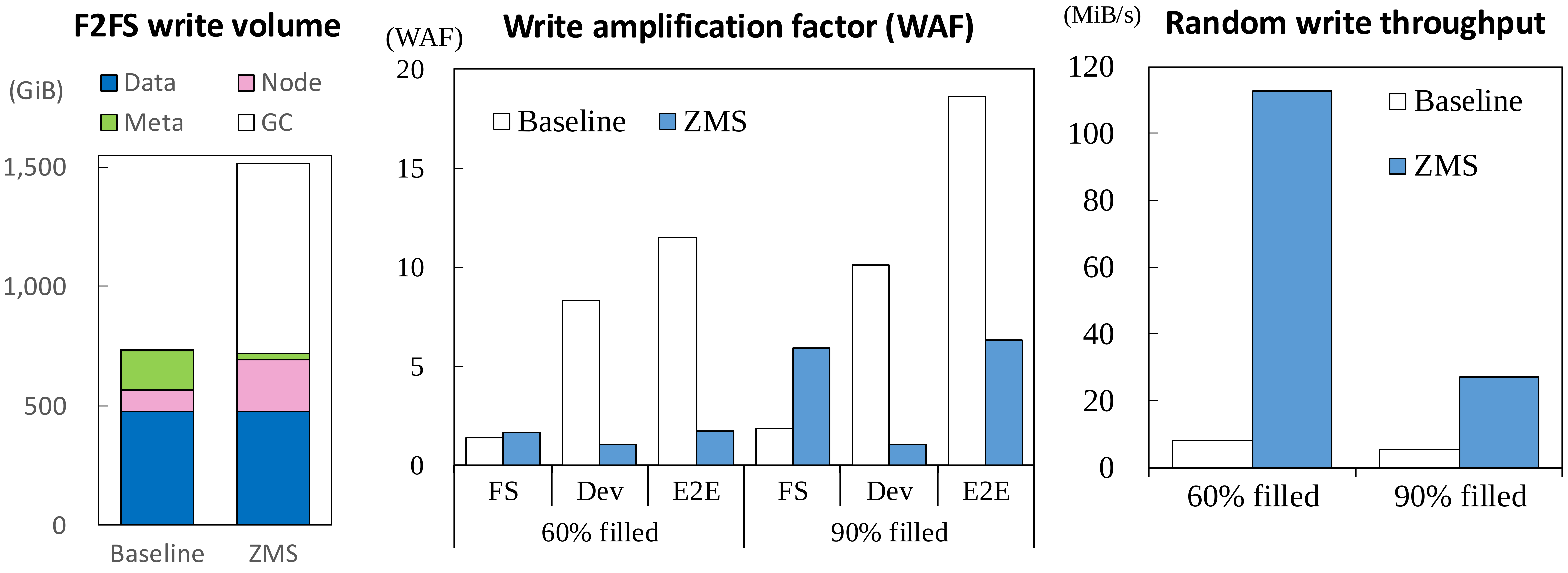


Workload	Configuration
Sequential read/write	Fio[1], 512 KiB IO size
Buffered random read/write	Fio, 4 KiB IO over 1 GiB file
Synchronous random write	Fio, 4 KiB write followed by fsync()
Wide range random read	Fio, 4 KiB read over 8 GiB file
Concurrent writing to multiple zones	Three concurrent Fio writing jobs, each writing to its own files
SQLite benchmark (Mobibench[2])	1M insert() transactions, 3.9 MiB WAL[3] file, 385 MiB database file
Application launch	Category (number of apps): basic (8), image (3), video (5), education (4), game (17)

[1]FIO: https://fio.readthedocs.io/en/latest/fio_doc.html
[2]Mobibench: https://github.com/ESOS-Lab/Mobibench
[3]WAL: Write ahead logging

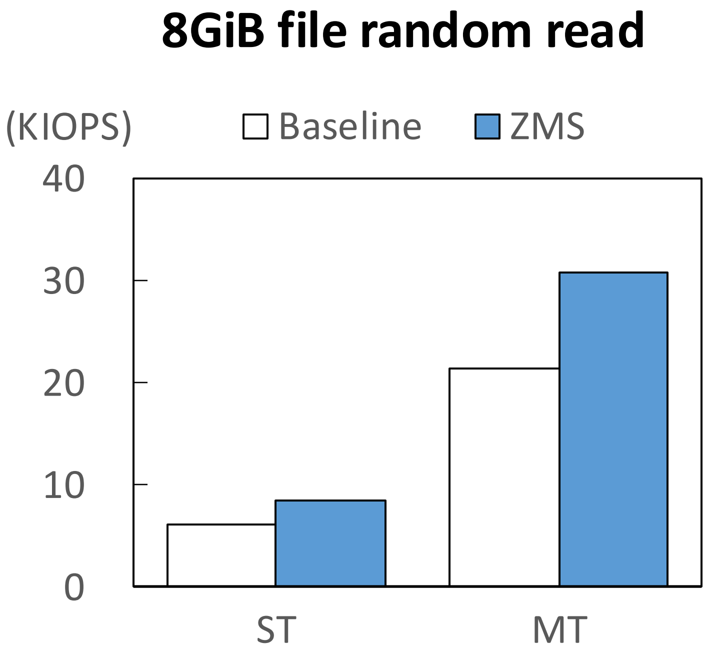
Random Write Performance & Write Amplification

- **2.85x ~ 6.4x** lower write amplification
- **5x ~ 13.6x** higher random write throughput

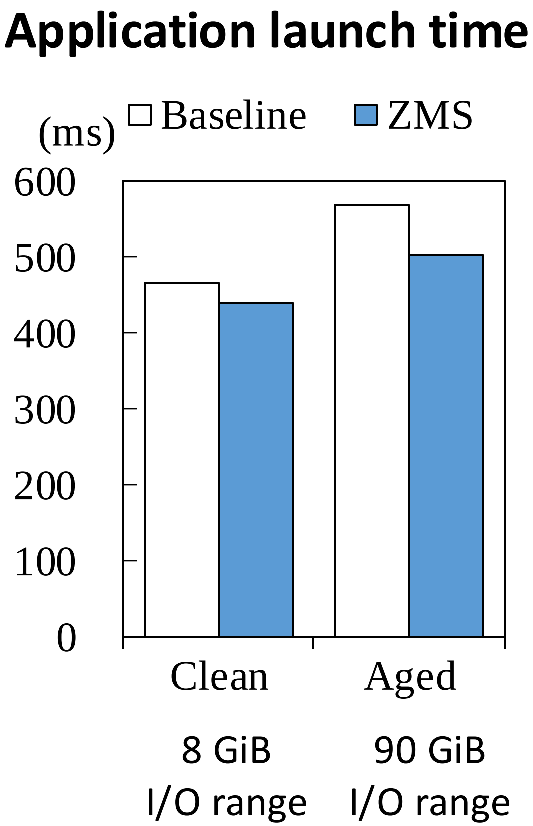


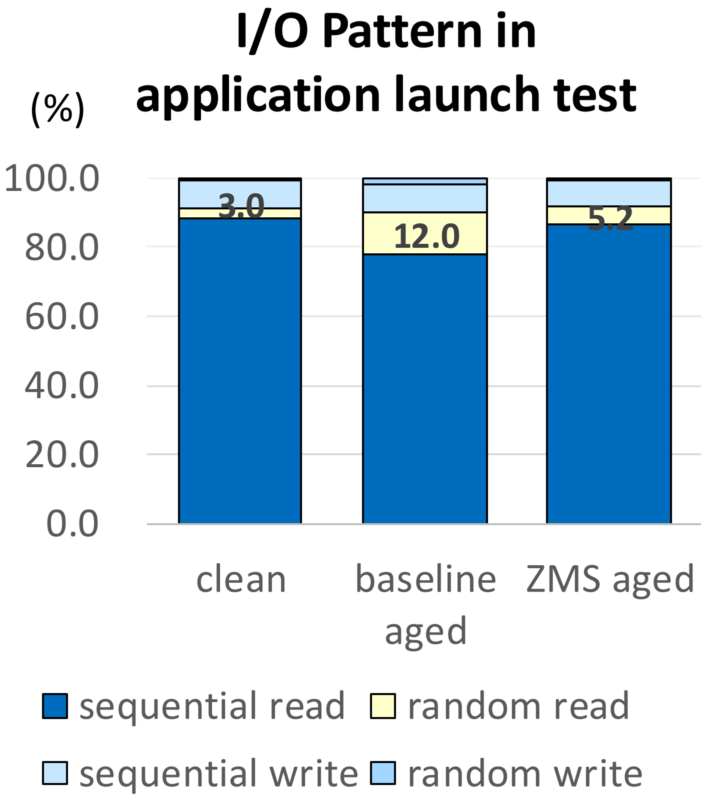
Random Read Performance

- **37~44%** better random read performance
- Application launch time: **5.8 ~ 11.6%** reduction



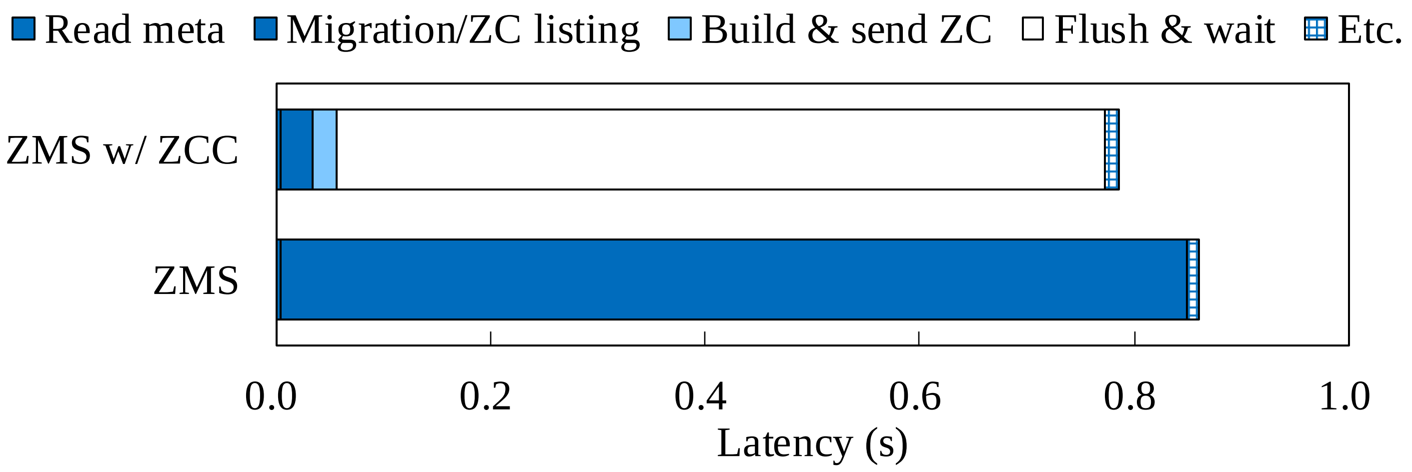
Baseline: map cache miss ratio: 27.1%
ZMS: no map cache miss



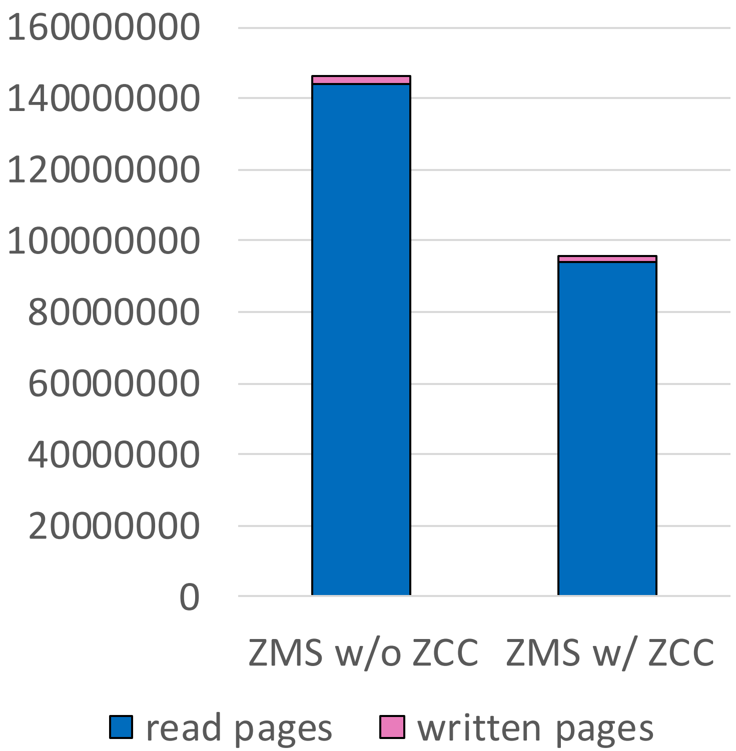


Copy Offloading Impact

- **8.7%** reduction of F2FS garbage collection latency
- **34%** reduction of I/O in application launch test (improved page cache efficiency)
- **7.3%** reduction of power in random write test (reduced CPU load)

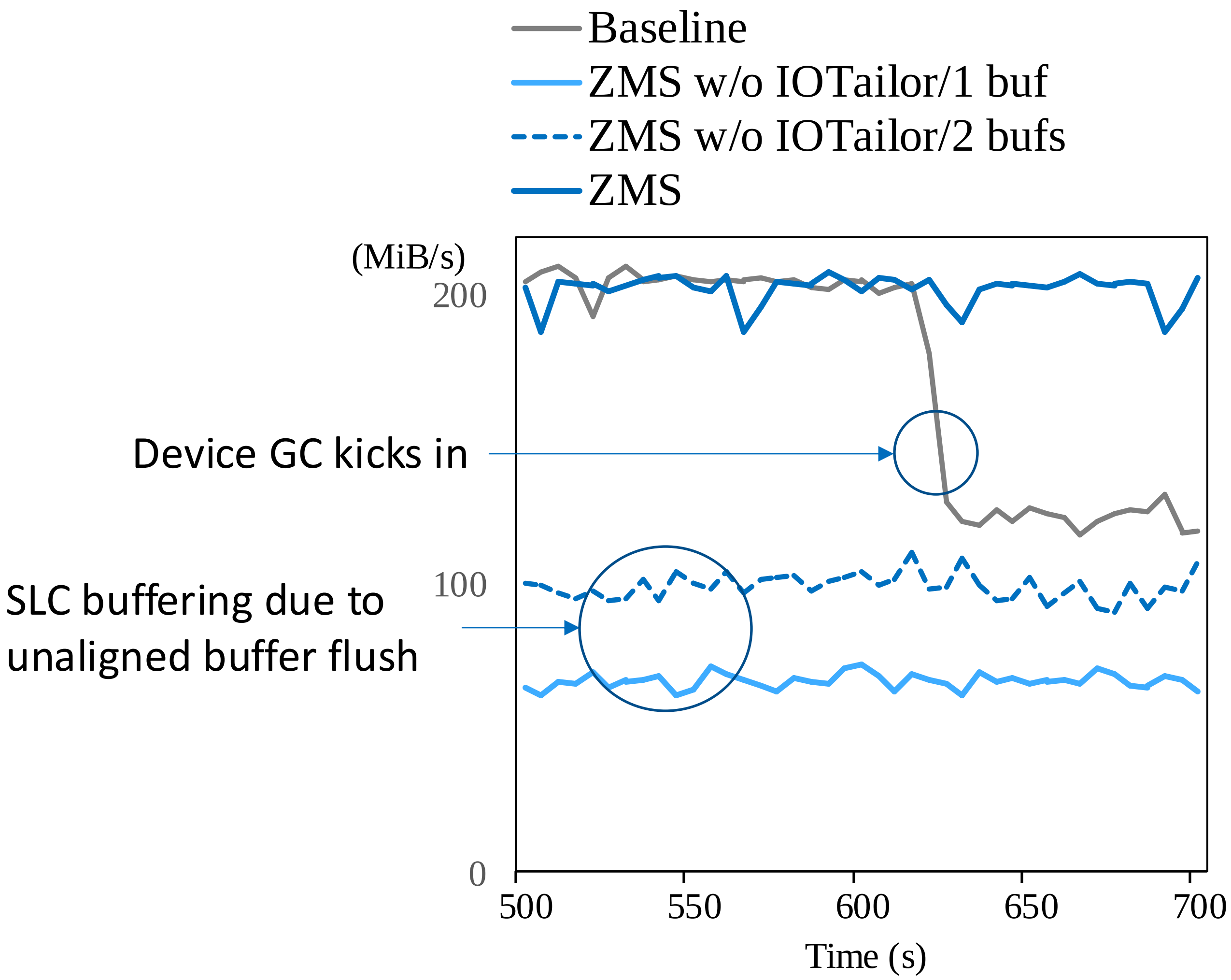


Fio random write test (76 GB file)	Baseline	ZMS w/o ZCC	ZMS w/ ZCC
Throughput (MiB/sec)	22.2	60.4	63.4
Test runtime (sec)	3461	1272	1210
Average CPU load	55.1	100.8	65.3
Average power during runtime (Wh)	2.76	1.24	1.15



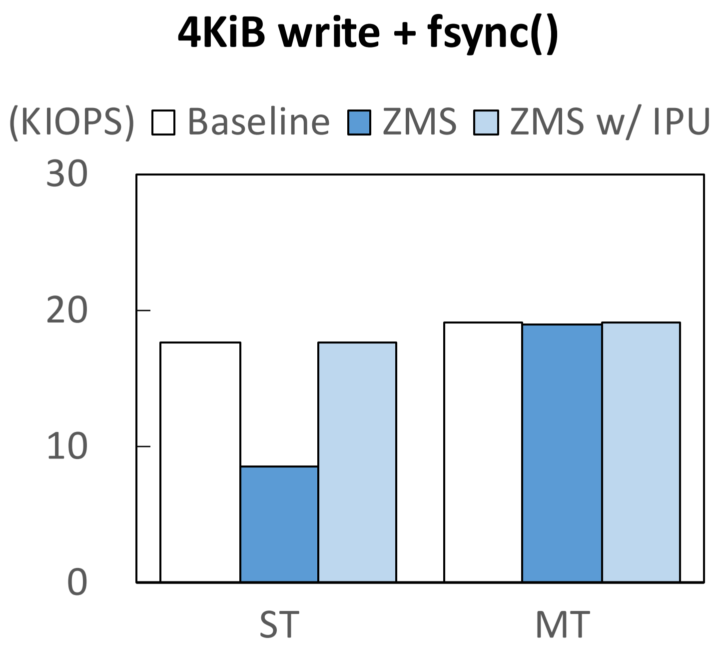
Performance of Writing to Multiple Zones

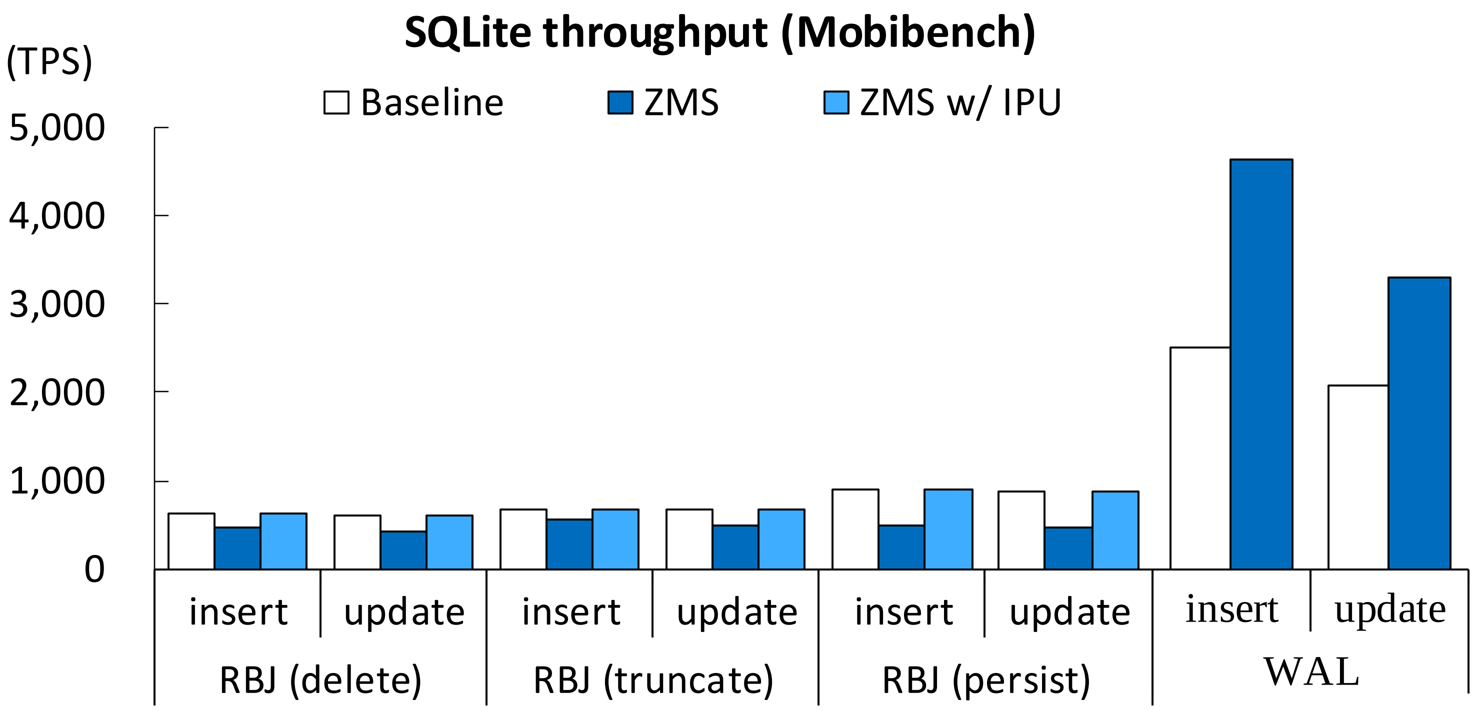
- IOTailor improves multi-zone write performance by reducing SLC buffering.



Synchronous Update Performance

- Using the budget-based in-place update, ZMS shows no performance degradation in tiny synchronous update and SQLite rollback journal mode.
- **60~100%** performance gain in write-ahead log (WAL) mode (append logging).





Baseline: In-place update
ZMS: append logging
ZMS w/ IPU: append logging with budget-based in-place update

Conclusion

- Zone abstraction is promising for enhancing responsiveness of mobile devices.
- Two challenges in zoned mobile storage
  - Degraded multi-zone write performance
  - Increased latency of tiny synchronous file update
- ZMS techniques address the challenges
  - IOTailor improves performance of writing to multiple zones by avoiding unaligned buffer flushes due to buffer switching.
  - Budget-based in-place update improves synchronous update performance.
- ZMS improves random read/write performance and write amplification significantly.

Thank You!