SSDs

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Spring 2026

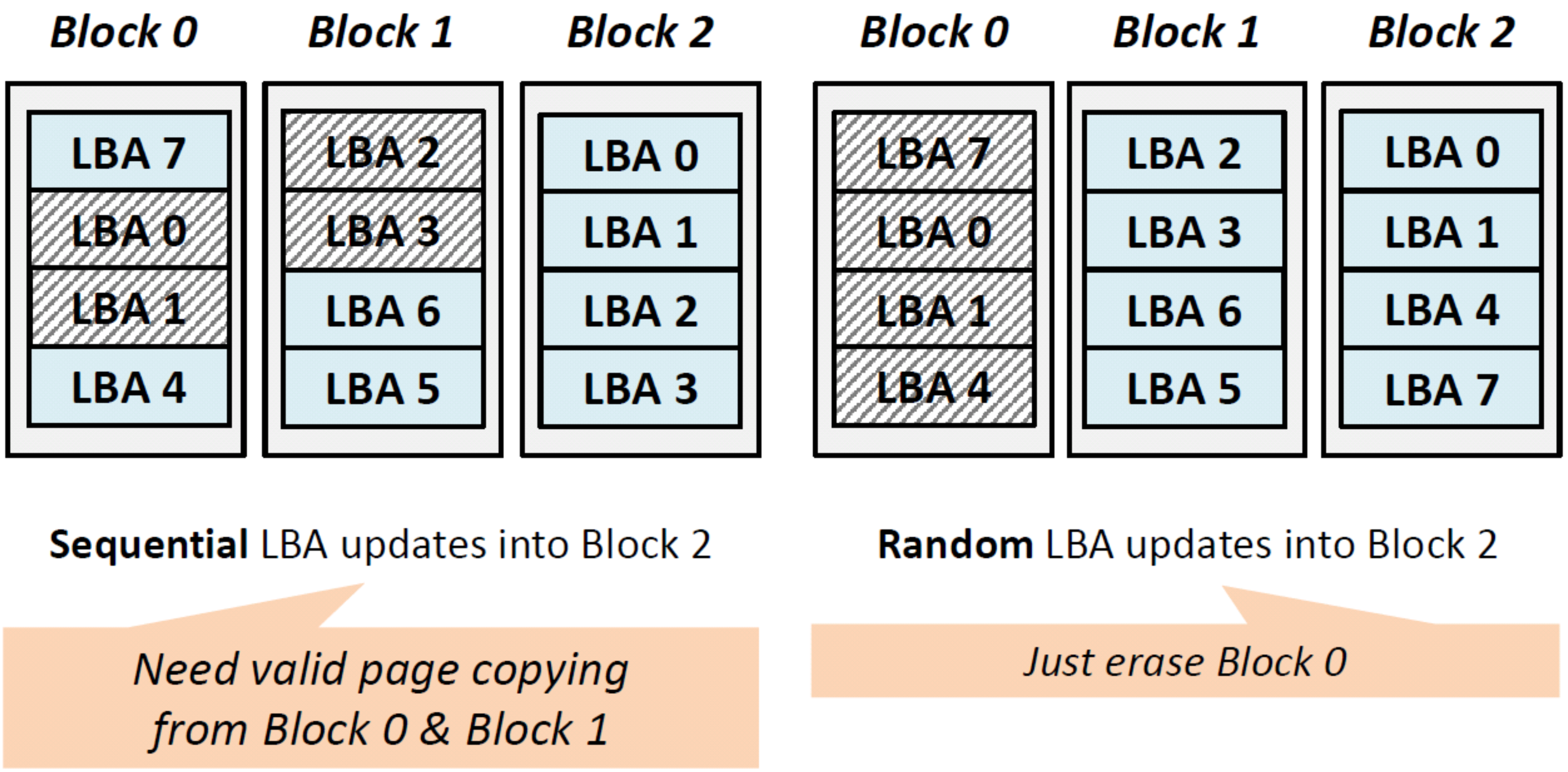
The Multi-streamed Solid-State Drive

(J.-U. Kang et al., HotStorage, 2014)

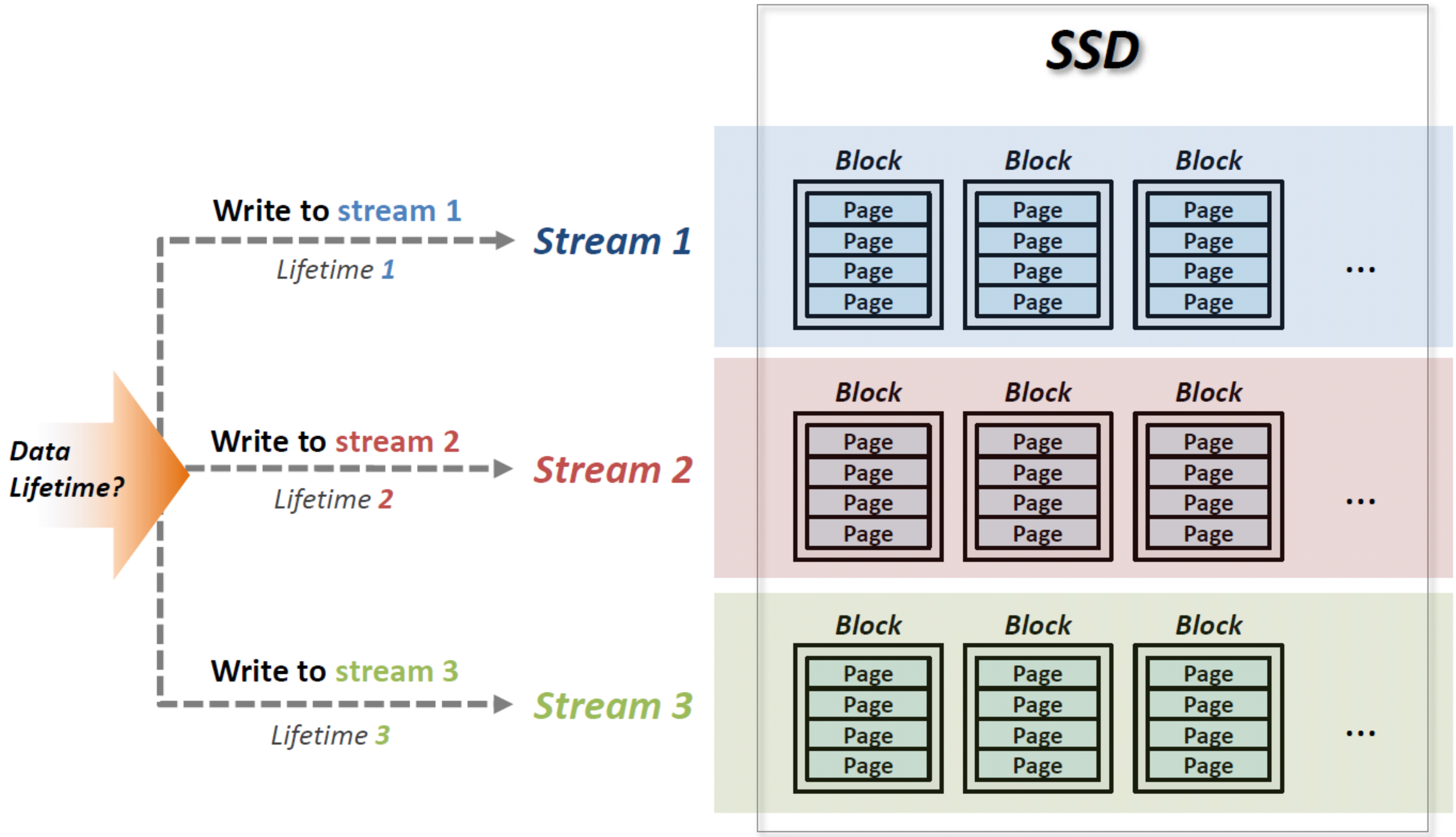
Some of slides are borrowed from the authors' presentation.

Effects of Write Patterns

- Previous write patterns (= current state) matter



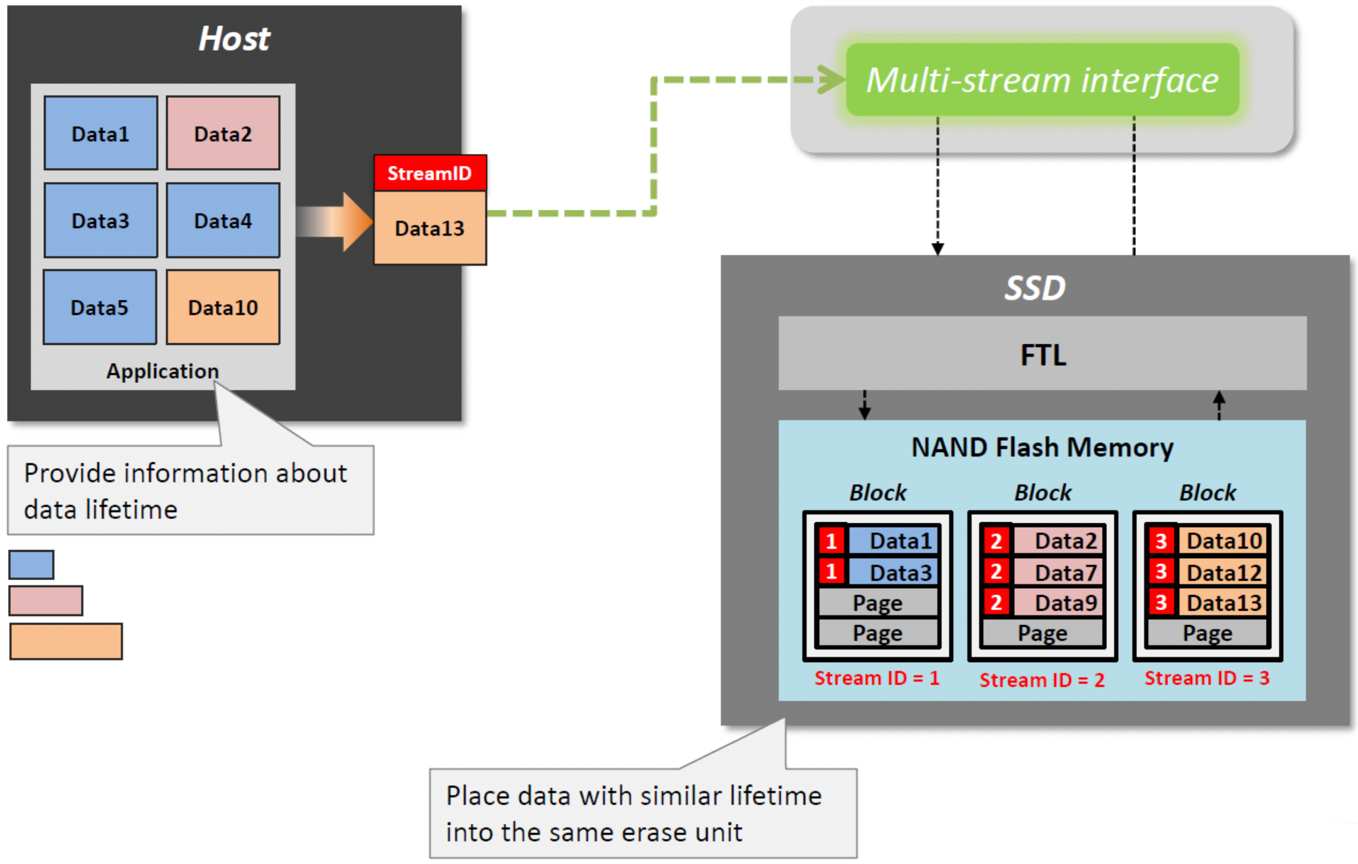
Stream



Data
Lifetime?
Write to stream 1
Lifetime 1
Stream 1
Write to stream 2
Lifetime 2
Stream 2
Write to stream 3
Lifetime 3
Stream 3
SSD
Block
Page
...

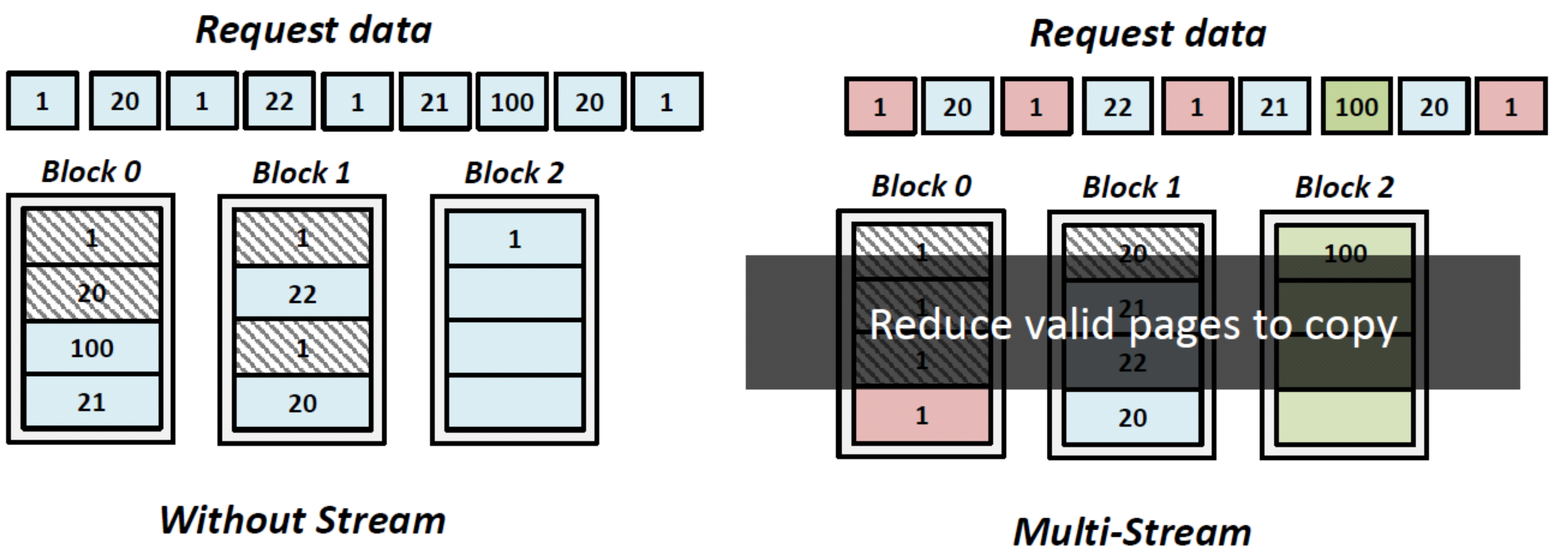
The Multi-streamed SSD

- Mapping data with different lifetime to different streams



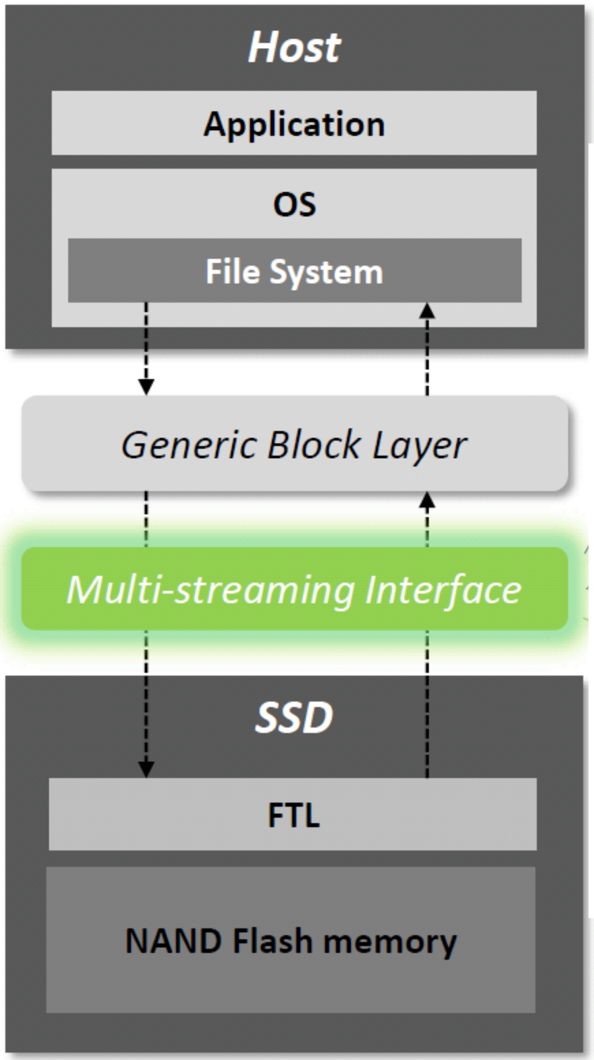
Working Example

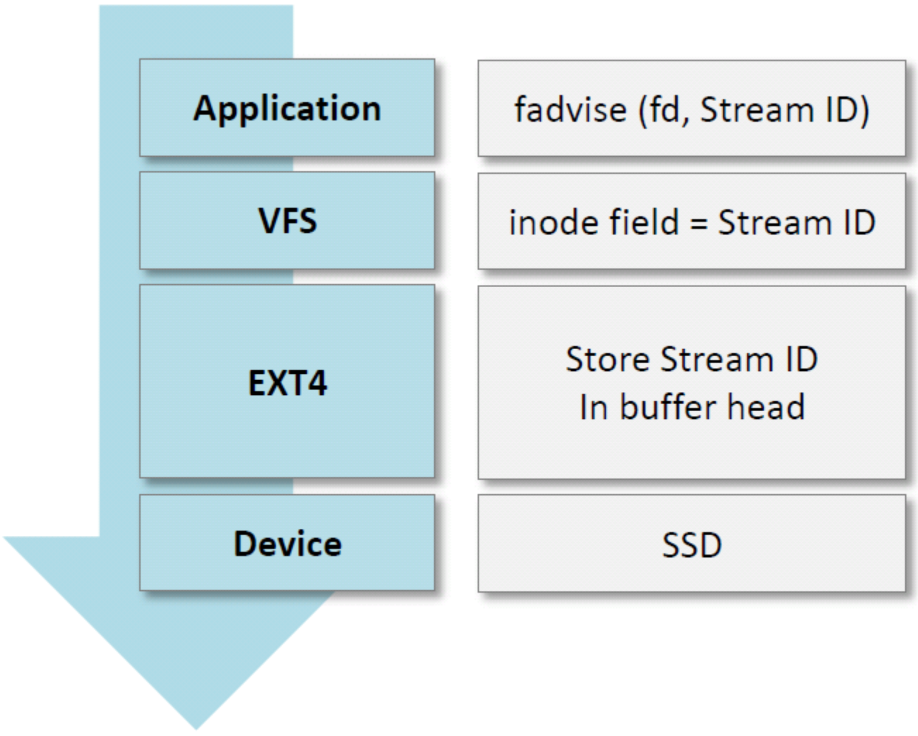
- High GC efficiency → Performance improvement



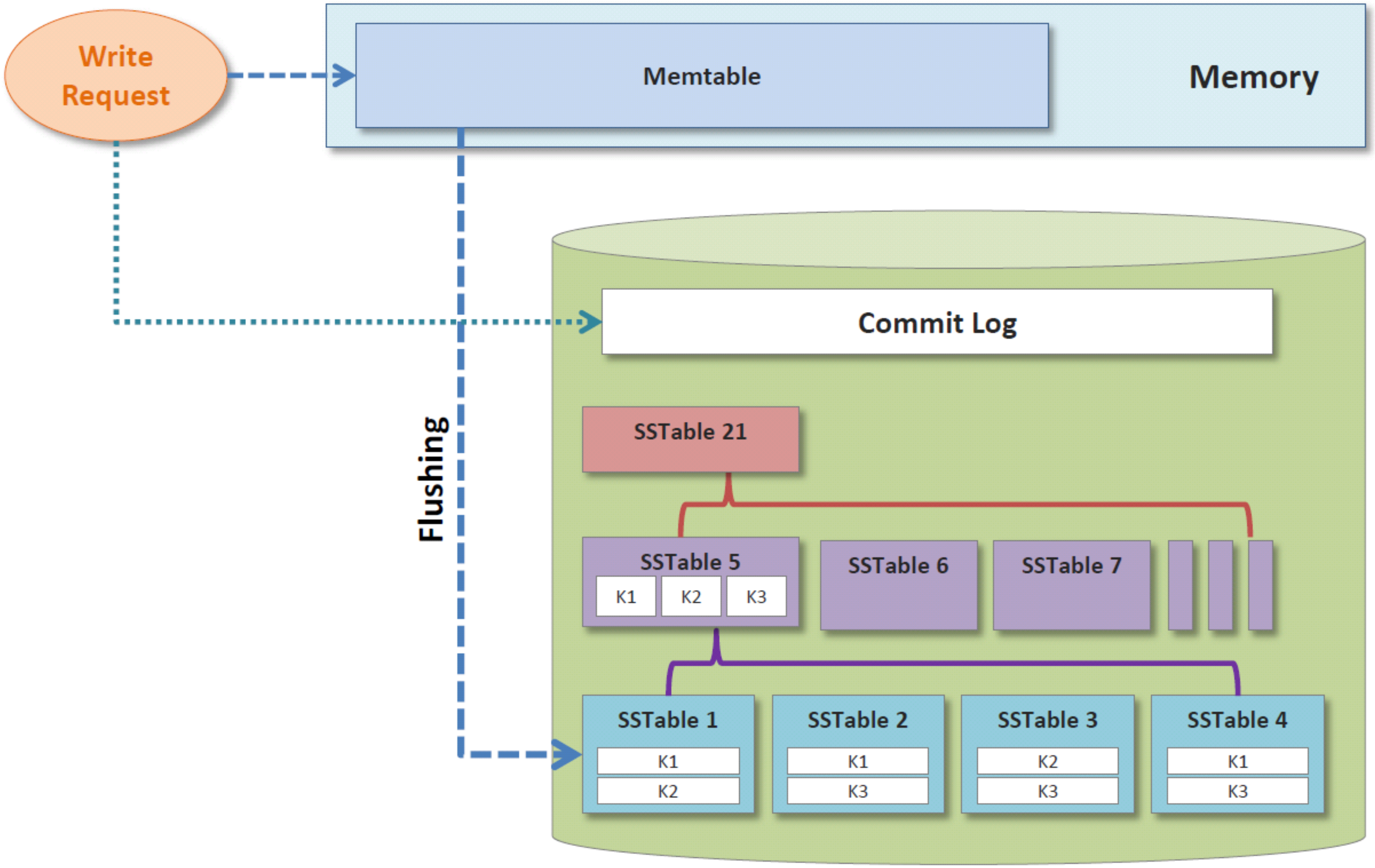
For effective multi-streaming,
proper mapping of data to streams is essential!

Architecture



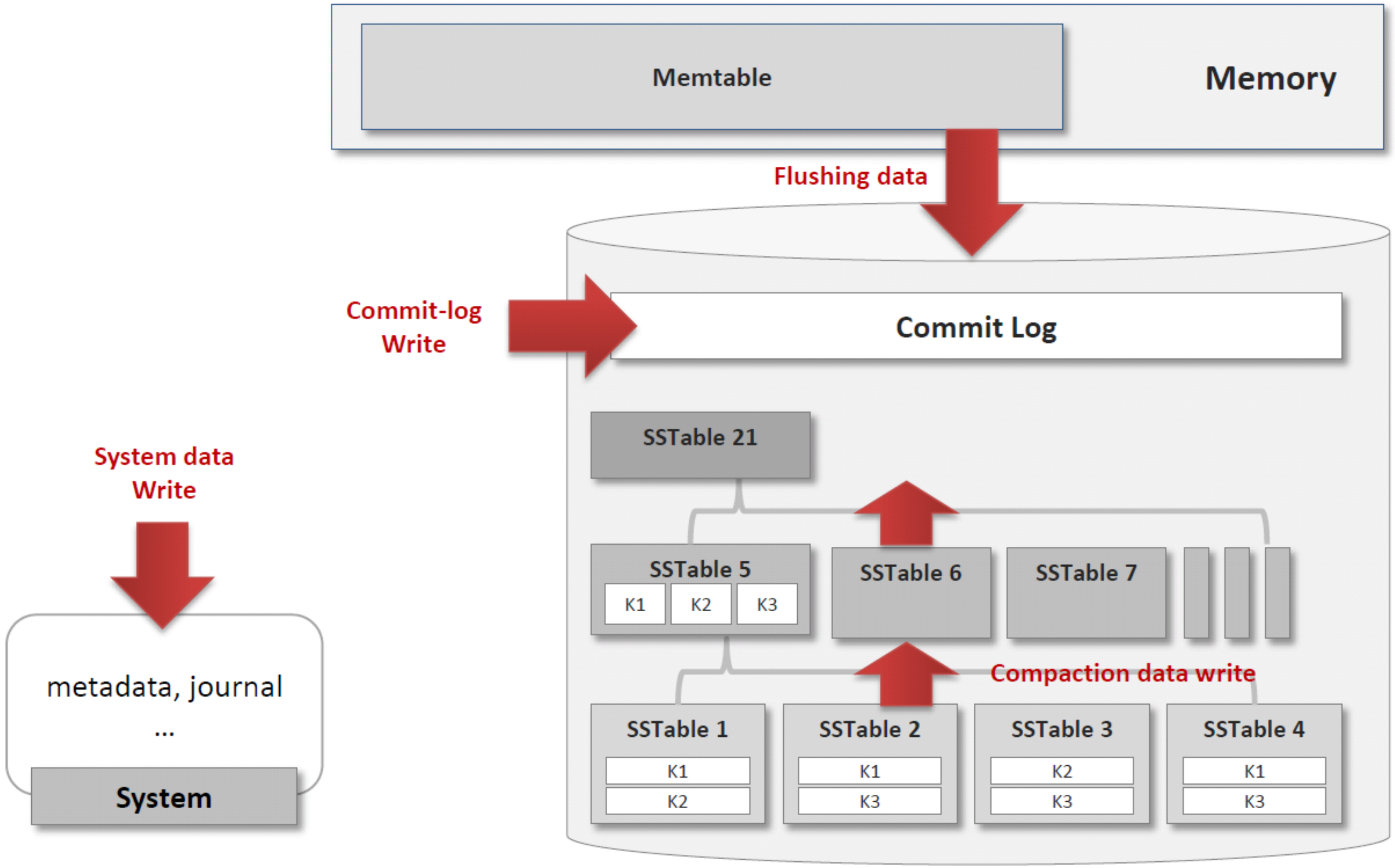


Case Study: Cassandra



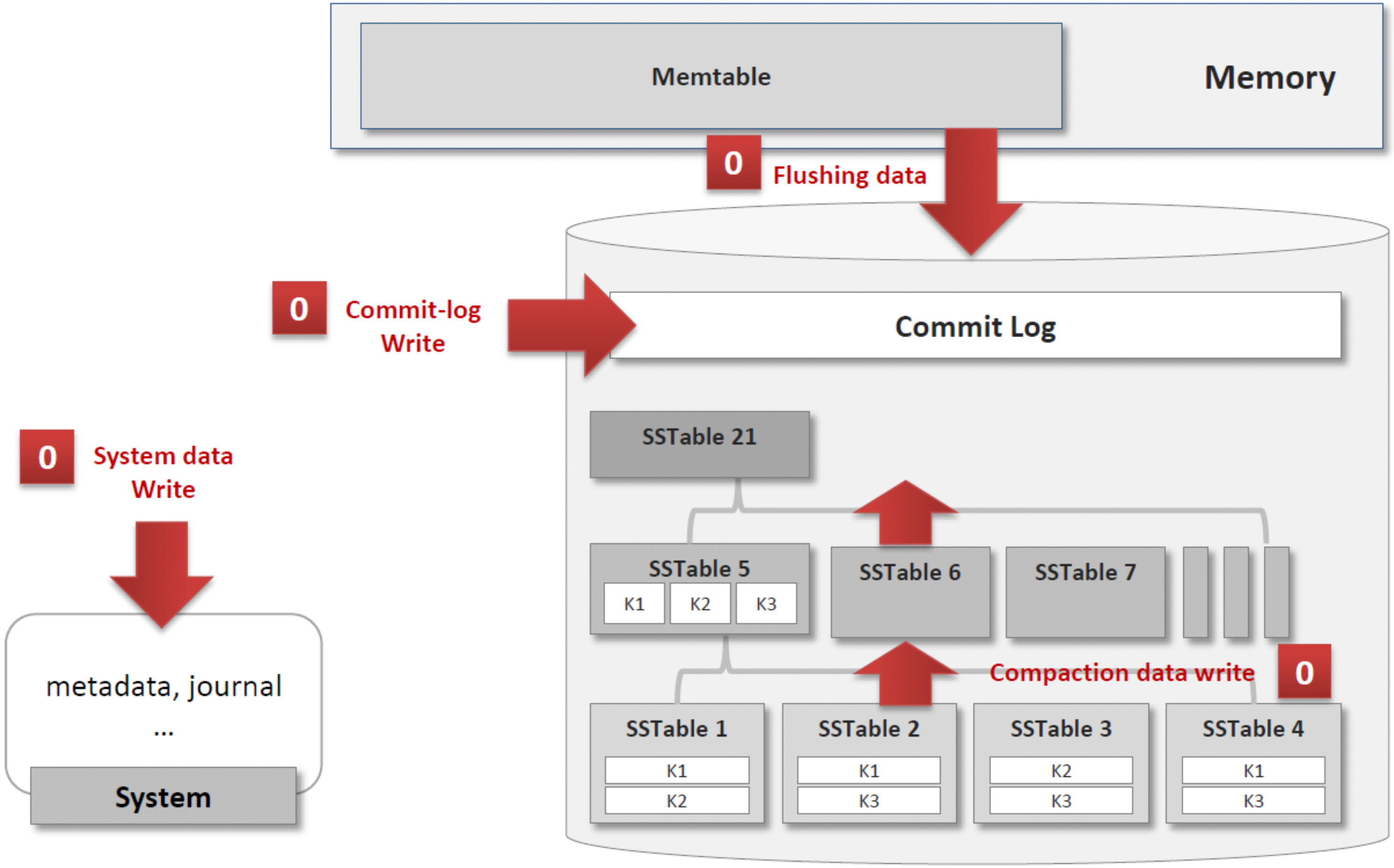
Cassandra's Write Patterns

- Write operations when Cassandra runs



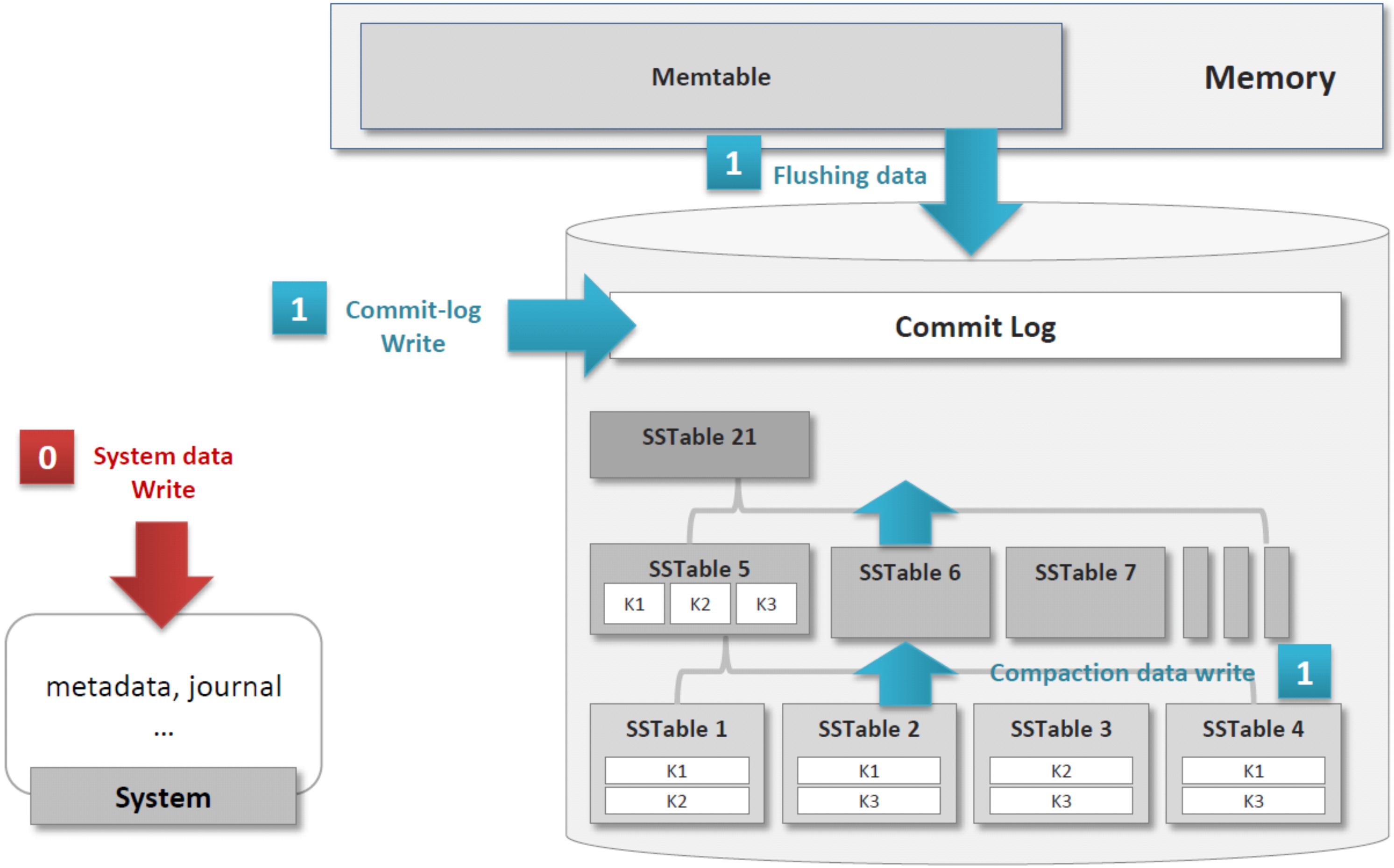
Mapping #1: Conventional

- Just one stream ID (= conventional SSD)



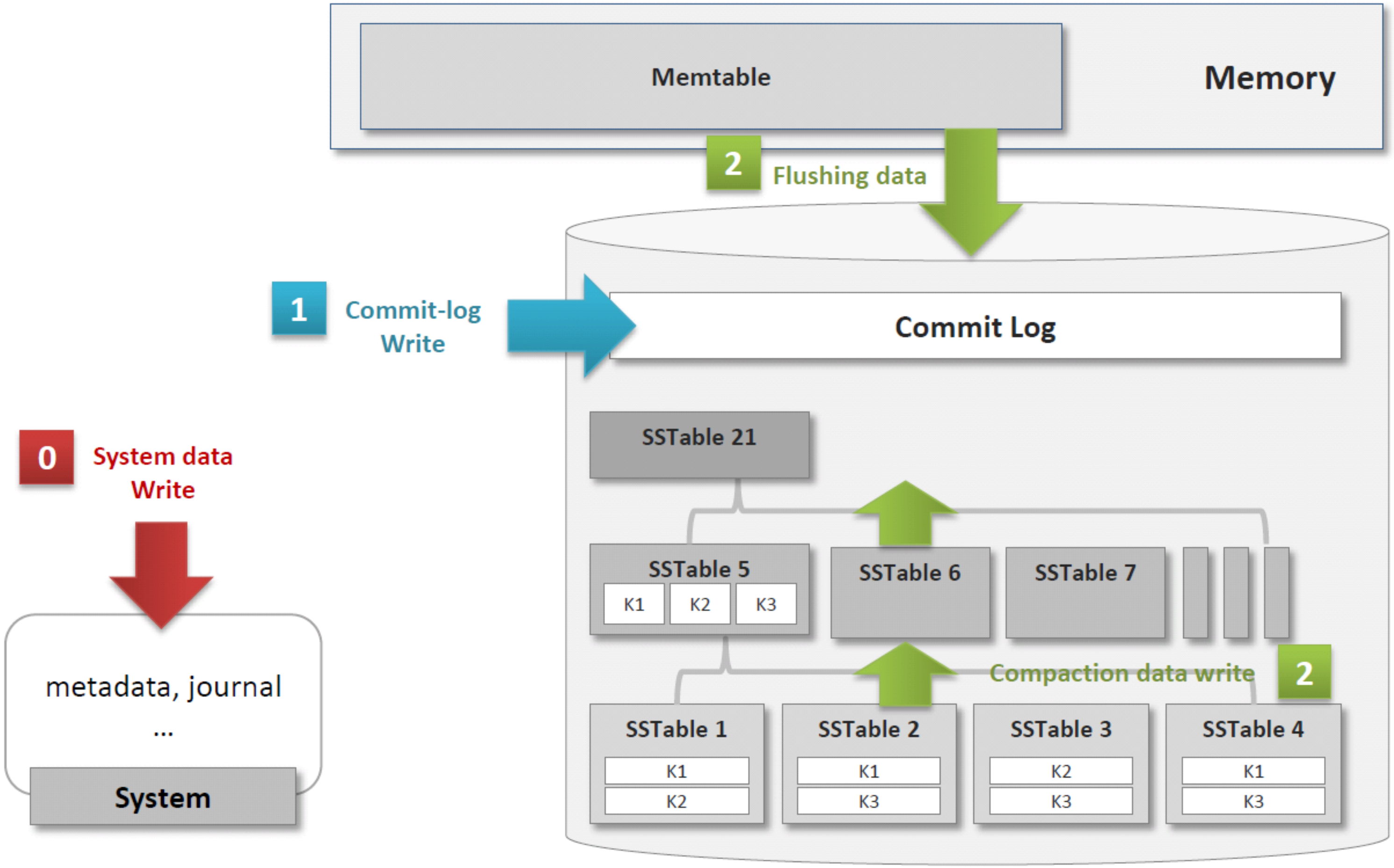
Mapping #2: Multi-App

- Separate application writes (ID 1) from system traffic (ID 0)



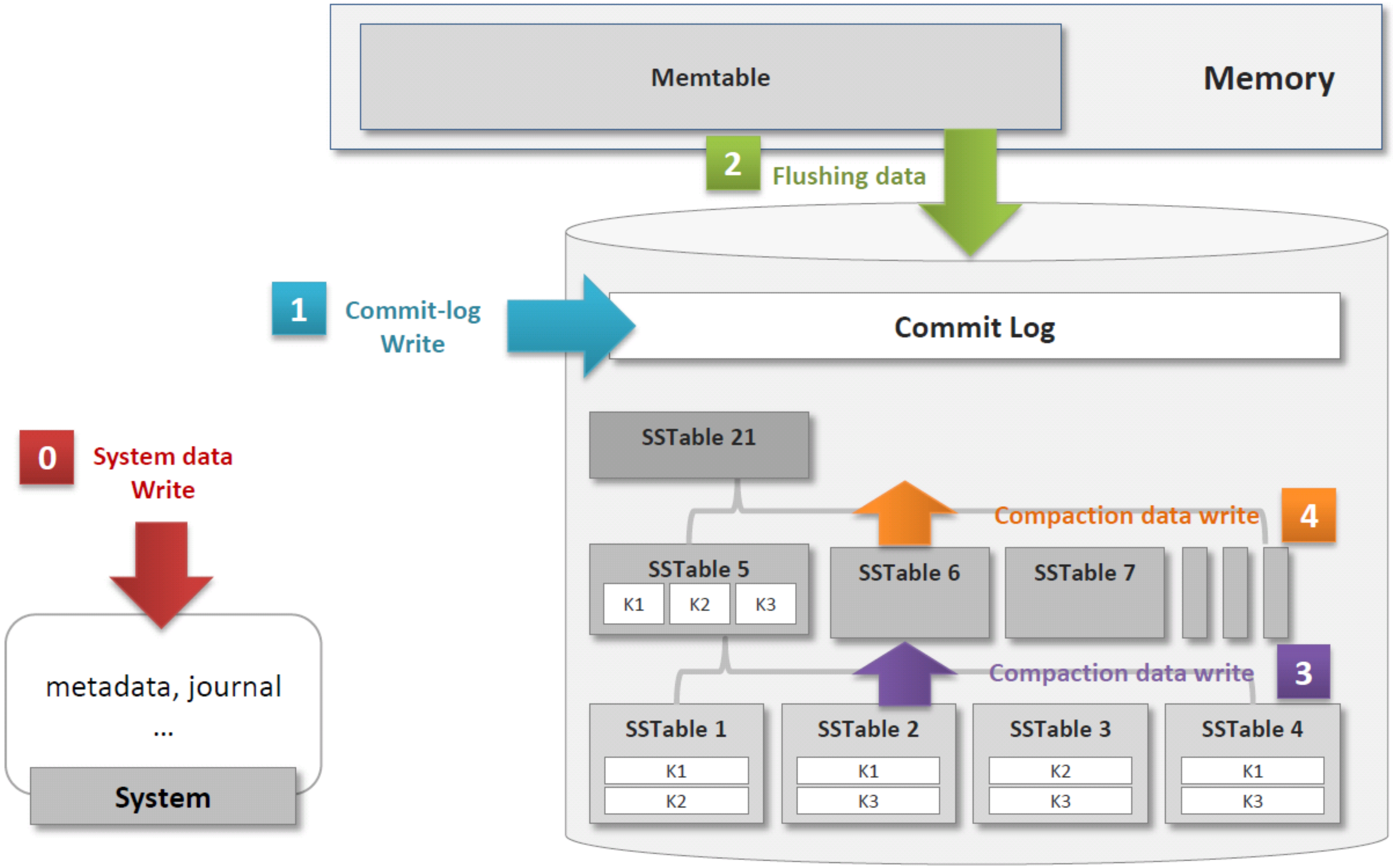
Mapping #3: Multi-Log

- Use three streams; further separate Commit Log



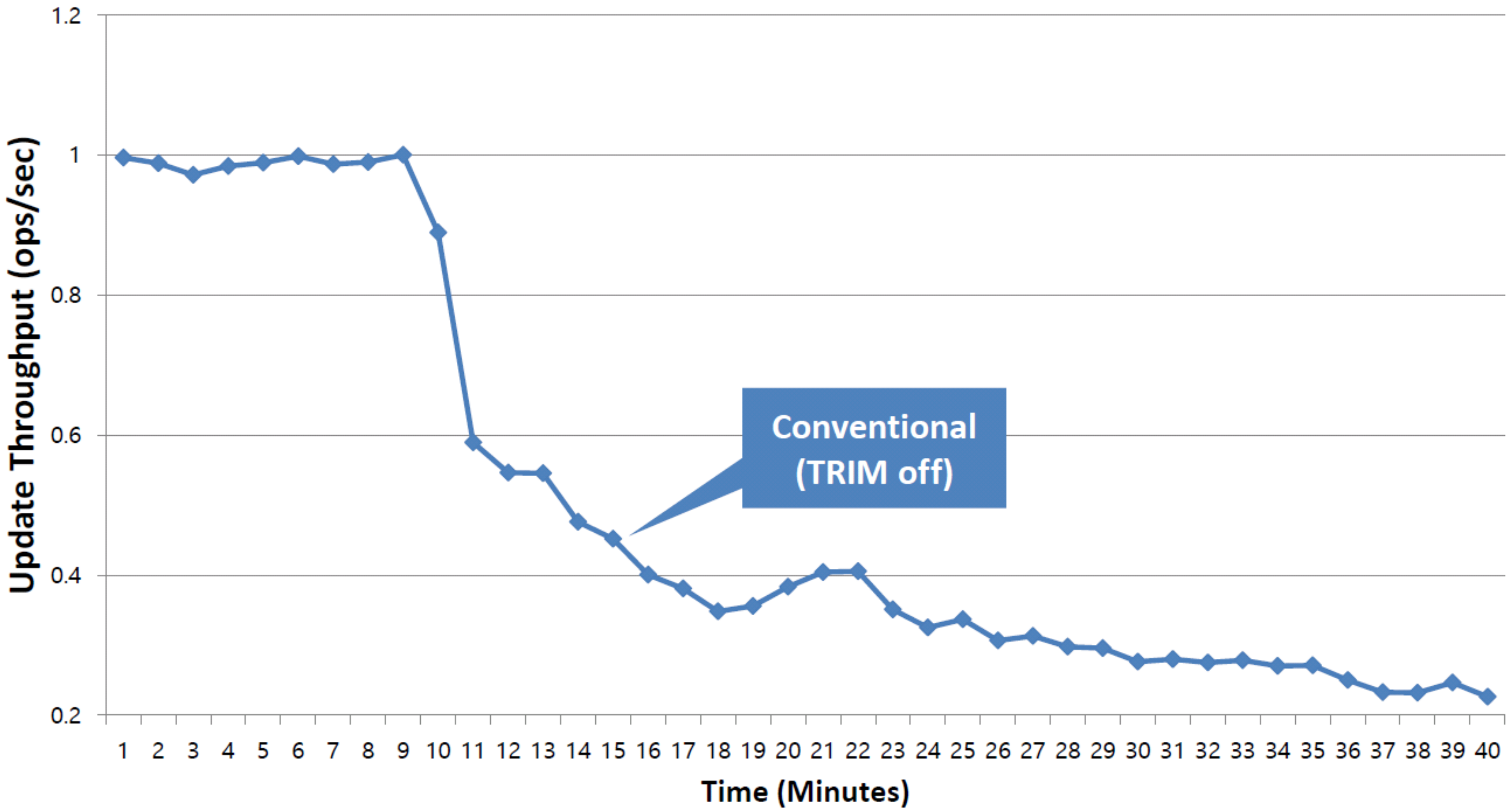
Mapping #4: Multi-Data

- Give distinct streams to different tiers of SSTables



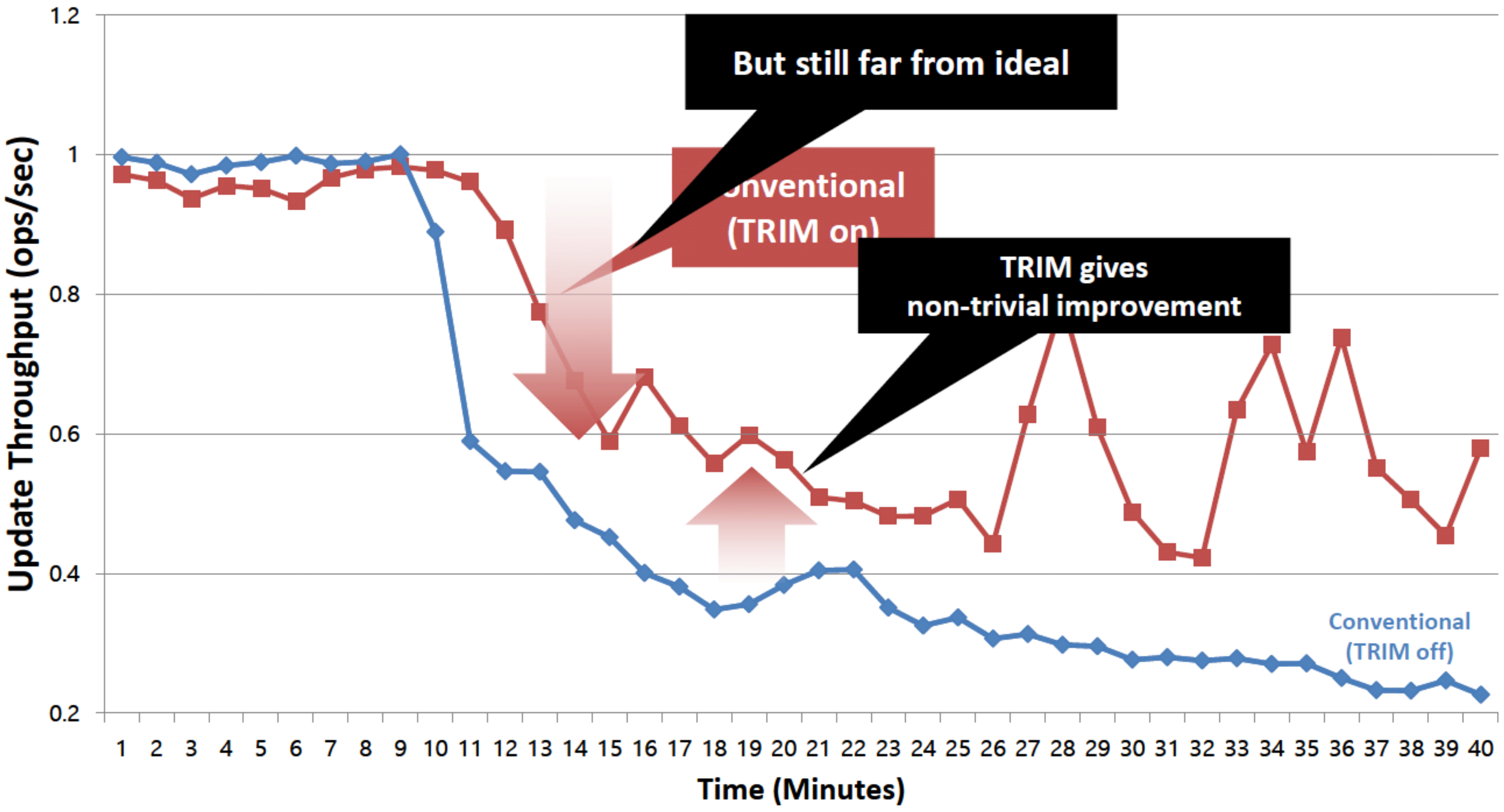
Results: Conventional

- Cassandra's normalized update throughput
  - Conventional "TRIM off"



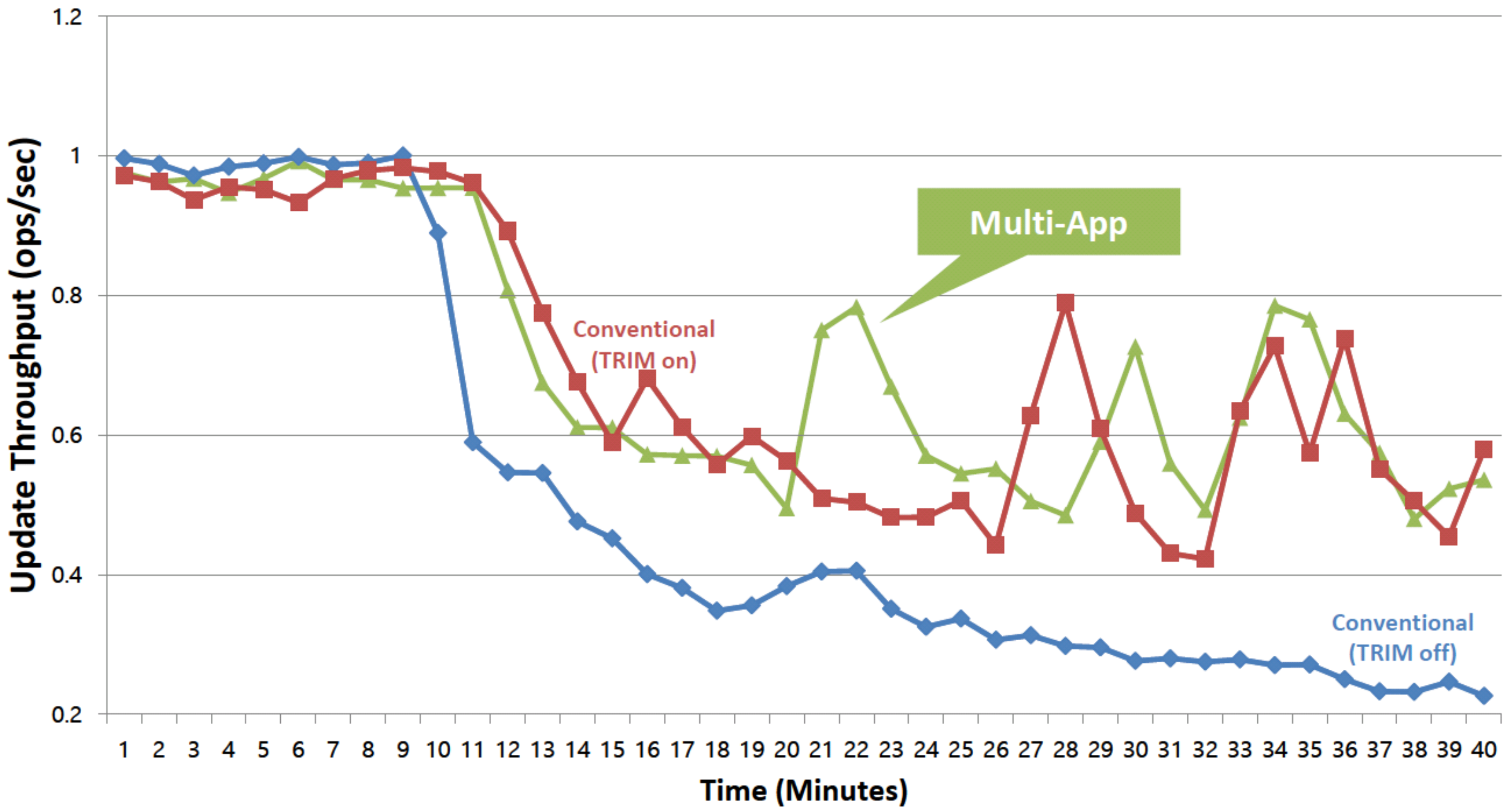
Results: Conventional with TRIM

- Cassandra's normalized update throughput
  - Conventional "TRIM on"



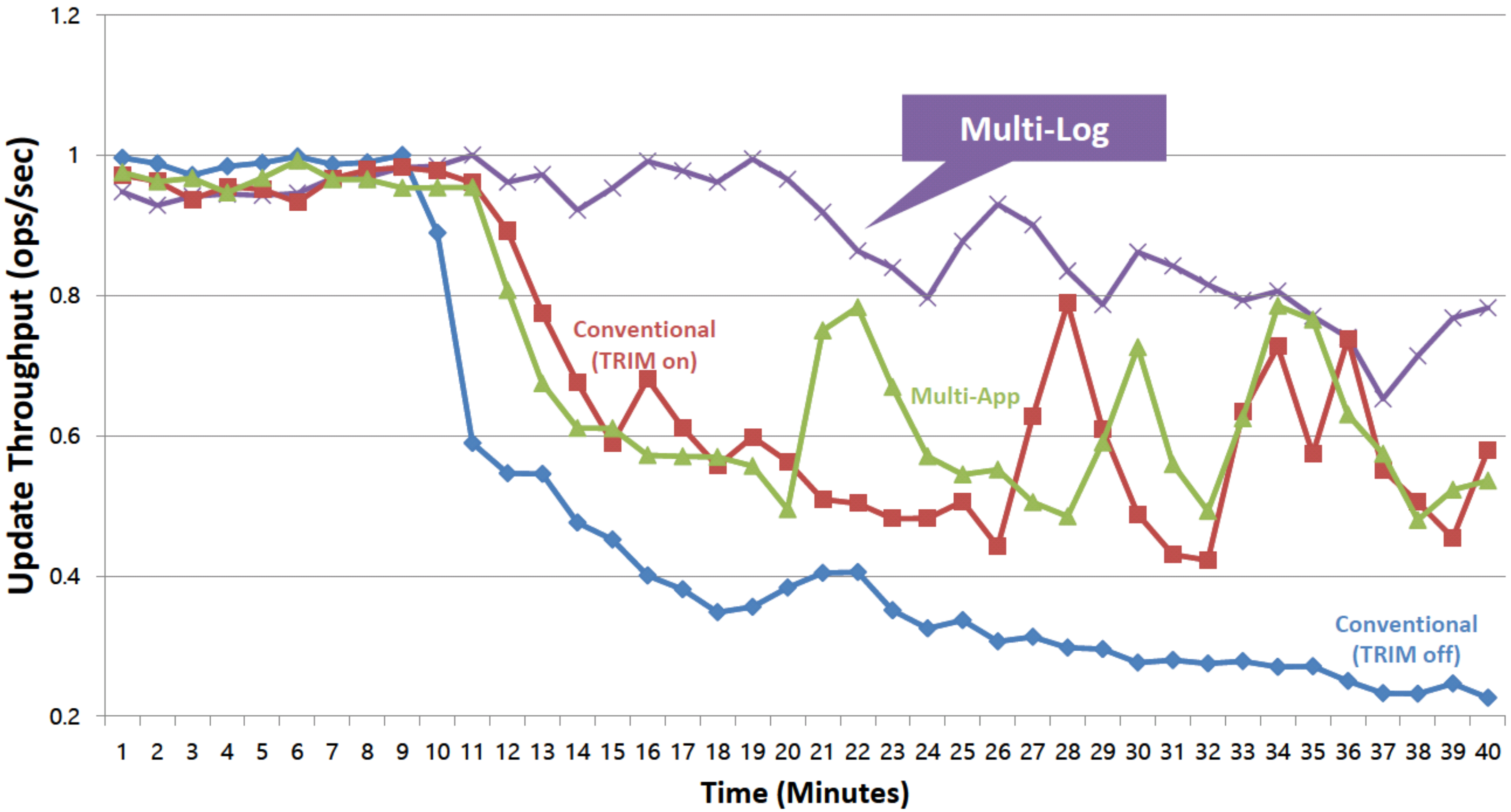
Results: Multi-App

- Cassandra's normalized update throughput
  - "Multi-App" (System data vs. Cassandra data)



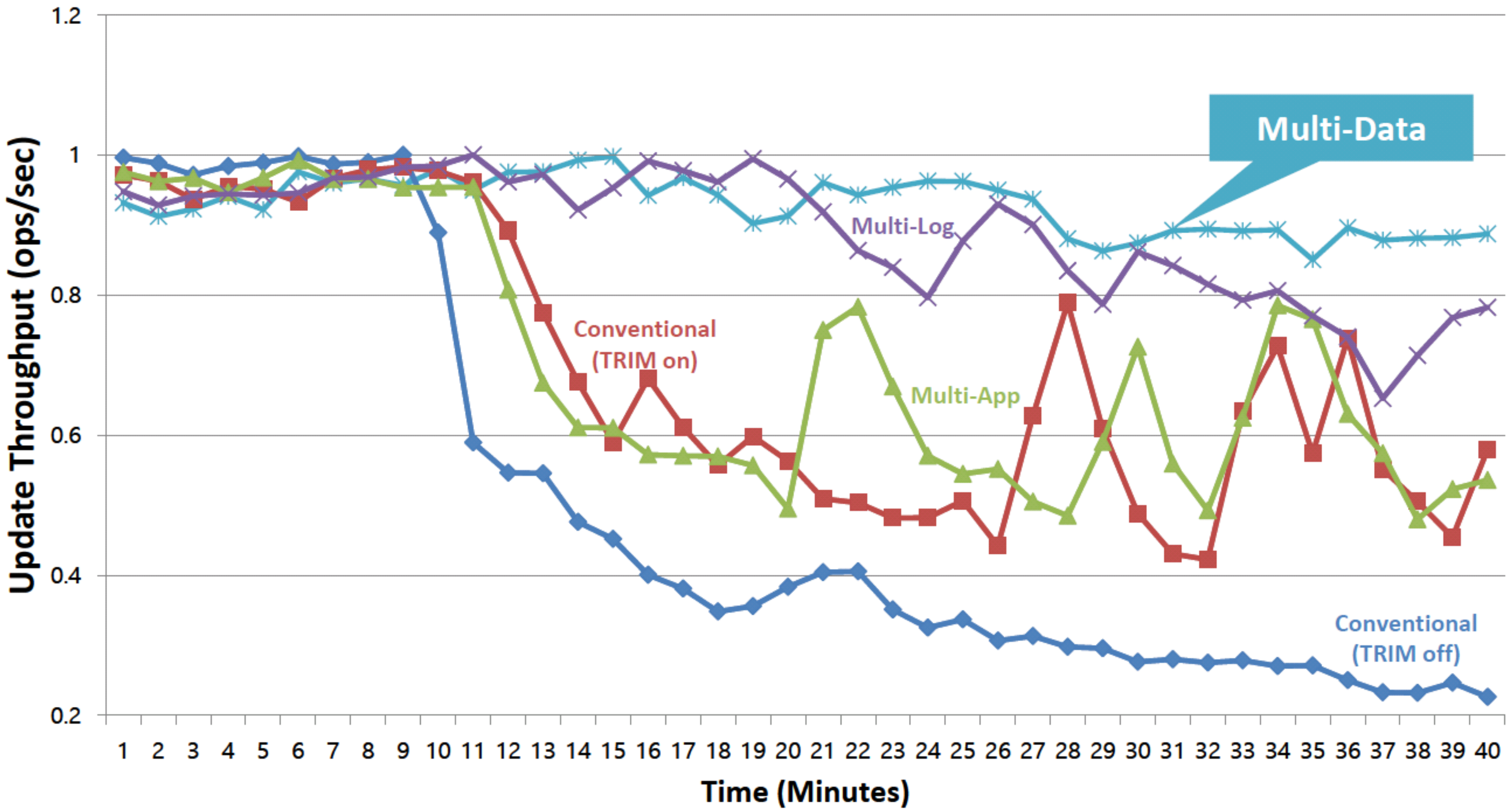
Results: Multi-Log

- Cassandra's normalized update throughput
  - "Multi-Log" (System data vs. Commit-Log vs. Flushed data)



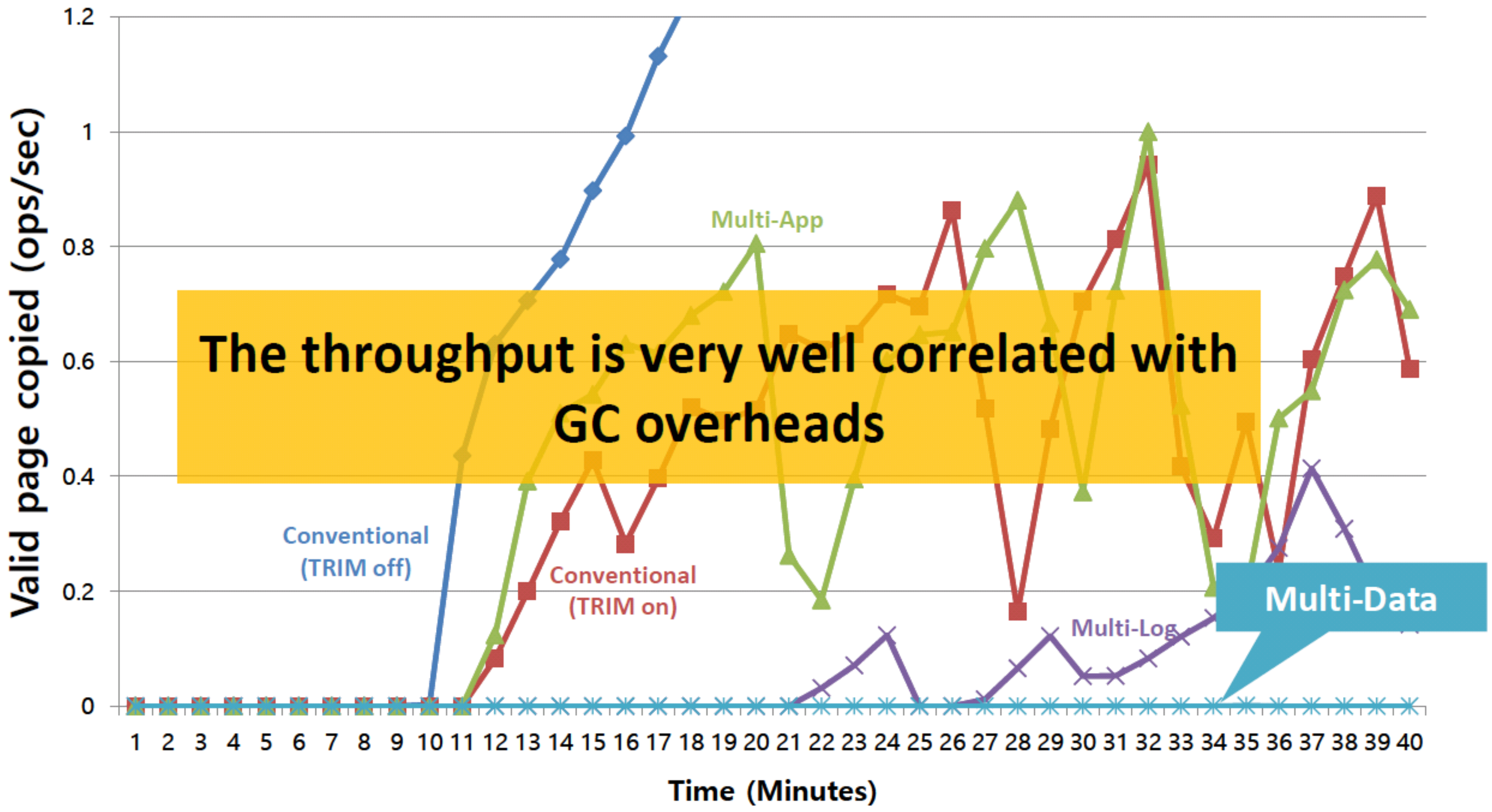
Results: Multi-Data

- Cassandra's normalized update throughput
  - "Multi-Data" (System data vs. Commit-Log vs. Flushed data vs. Compaction Data)



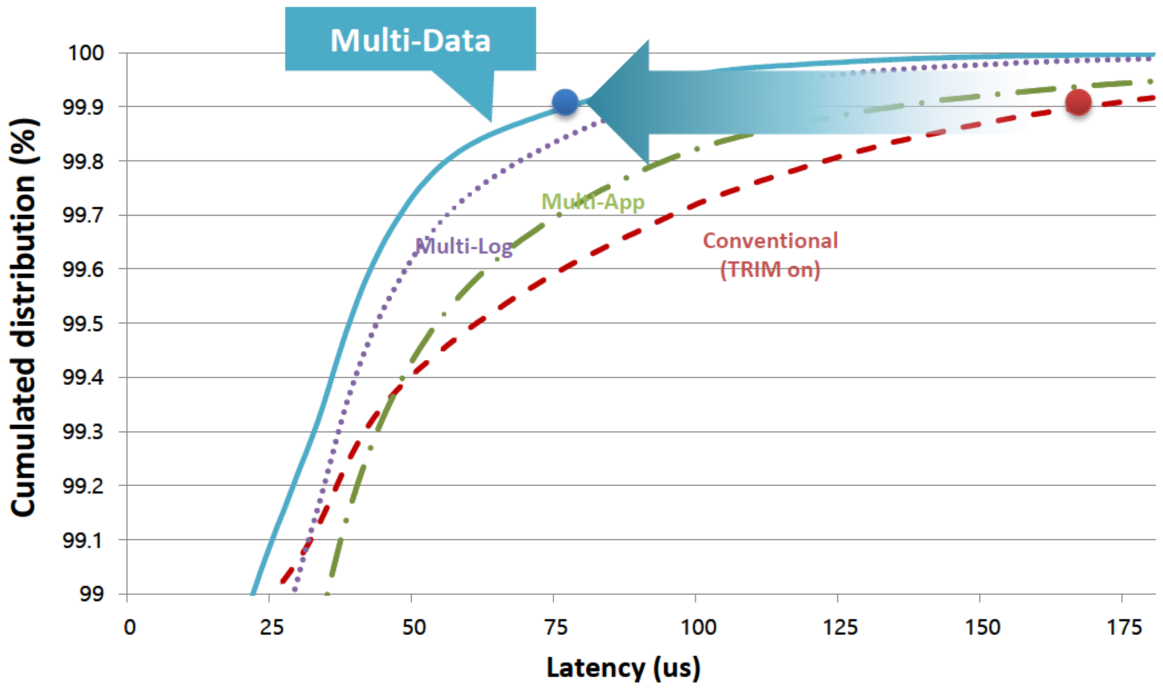
Results: GC Overheads

- Cassandra's GC overheads



Results: Latency

- Cassandra's cumulated latency distribution
  - Multi-streaming improves write latency
  - At 99.9%, Multi-Data lowers the latency by 53% compared to Normal



Summary

- Mapping application and system data with different lifetimes to SSD streams
  - Higher GC efficiency, lower latency
- Multi-streaming can be supported on a state-of-the-art SSD and co-exist with the traditional block interface

- Standardized in T10 SCSI (SAS SSDs) in 2015
- Standardized in NVMe 1.3 in 2017

ZNS SSDs

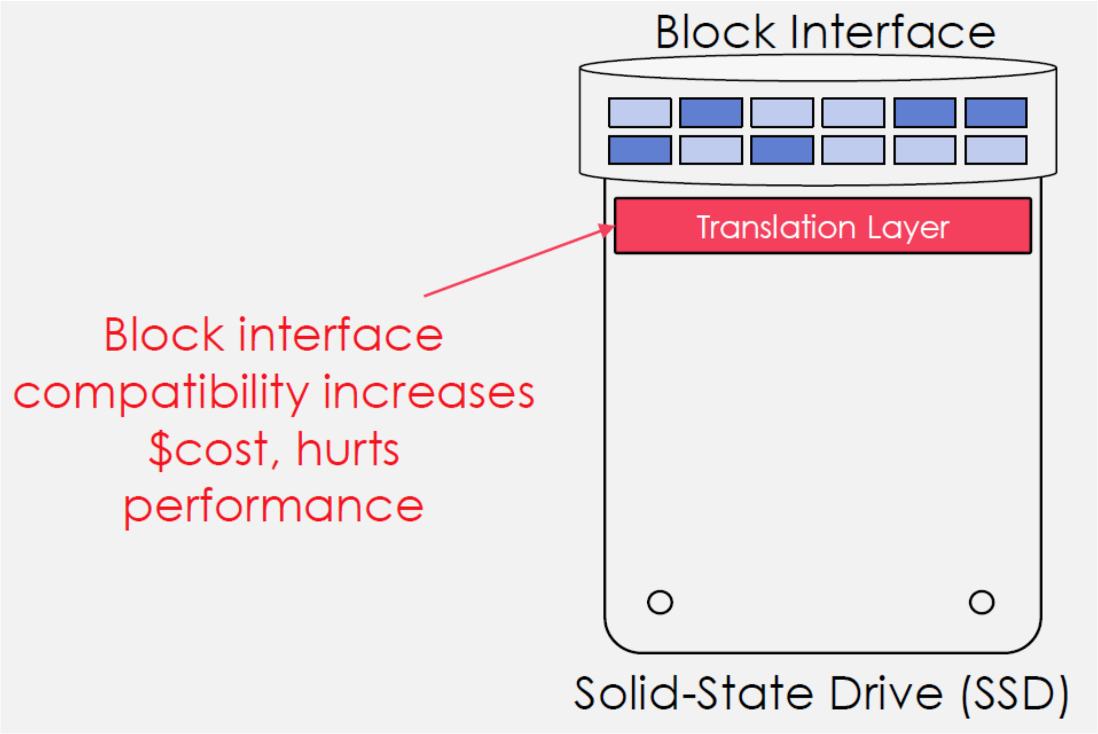
(Matias Bjørling et al., USENIX ATC, 2021)

Some of slides are borrowed from the authors' presentation.

The Block Interface Tax

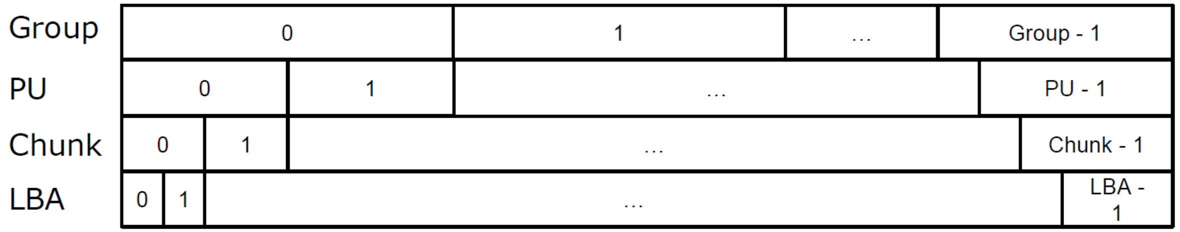
- For flash-based SSDs, the block interface is a poor fit
  - SSDs append pages to erase blocks, need to erase whole block before rewriting

- Data placement overhead
  - Media over-provisioning (7 ~ 28%)
  - Higher cost
  - Write amplification
  - Unpredictable latency
  - No isolation



History

- Baidu's Software Defined Flash (SDF) [ASPLOS '14, EuroSys '14]
  - Expose a channel as an independent device
- OCSSD 1.2
  - Physical Page Addressing: Channel, LUN (die), Plane, Block, Page, Sector
  - Exposes flash read/program/erase timings and MLC page pairing information
  - Everything in the host
- OCSSD 2.0 [FAST '17]
  - Physical Page Addressing: Group (channel), LUN (PU), Chunk, Logical block
  - Read/write/reset commands
  - Write sequentially within a chunk
  - Media management in the drive

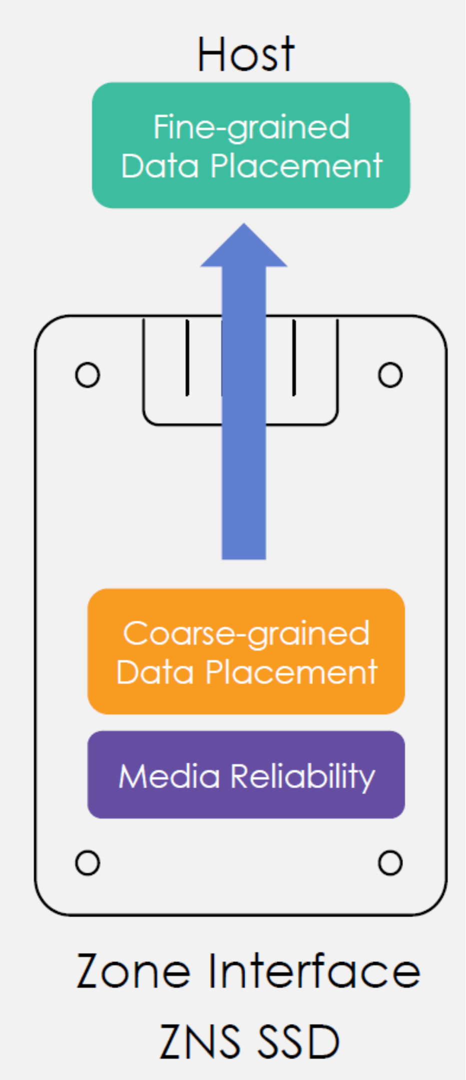


ZNS SSDs

- Sequential zone writes onto distinct erase blocks
  - Random writes are disallowed
  - Zones must be explicitly reset by the host
  - Data placement occurs at the coarse-grained level of zones

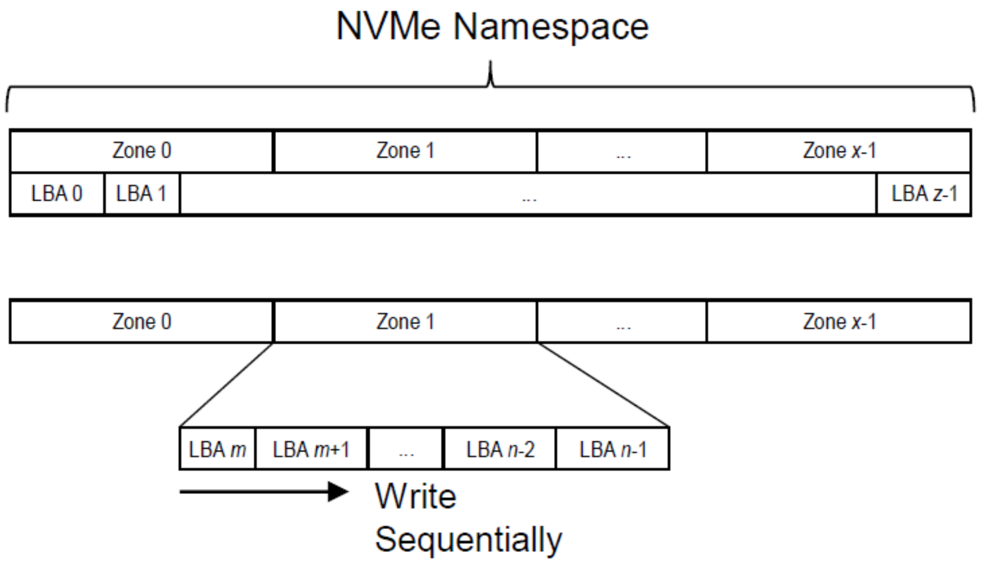
- ZNS SSDs relinquish GC responsibilities
  - GC of zones becomes the responsibility of the host

- Media reliability continues to be the full responsibility of the SSD



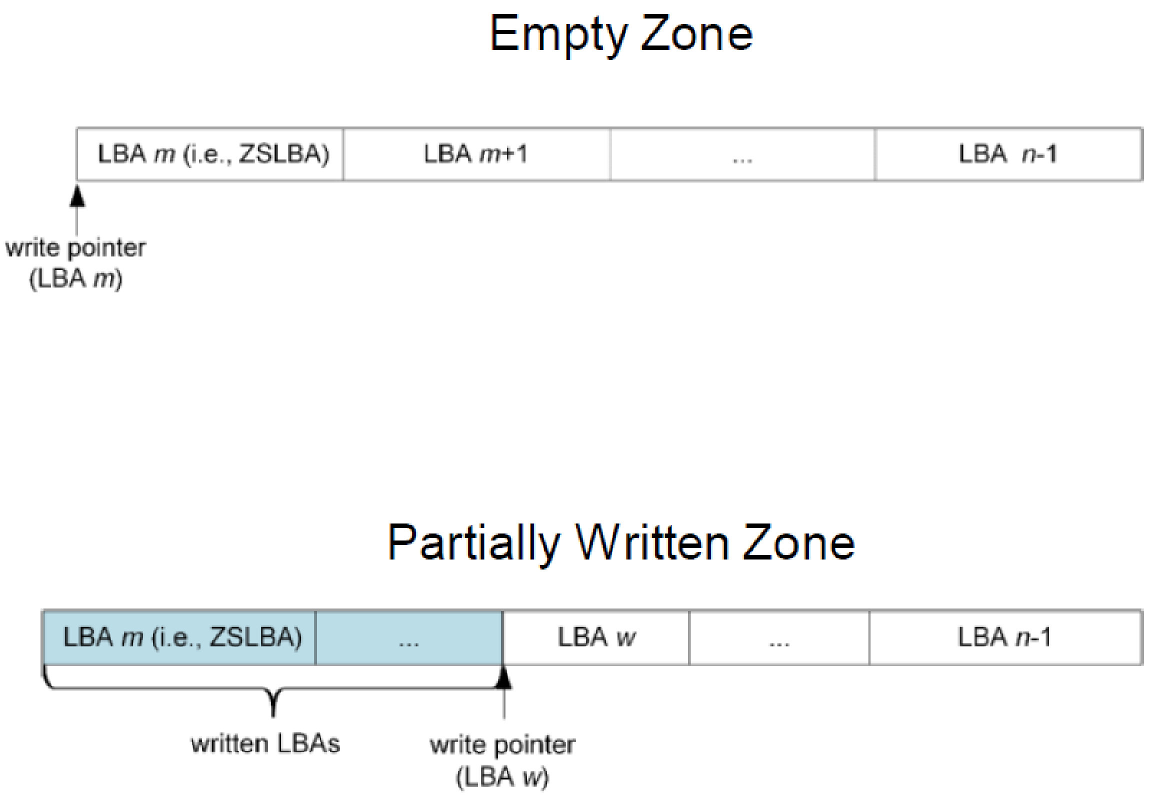
Zoned Storage Model

- Zones are laid out sequentially in an NVMe namespace
- The zone size is fixed and applies to all zones in the namespace
  - e.g., 512 MiB
- The command set inherits the NVMe Command Set
  - Built upon the conventional block interface (Read, Write, Flush and other commands)
  - Adds rules to collaborate on host and device data placement



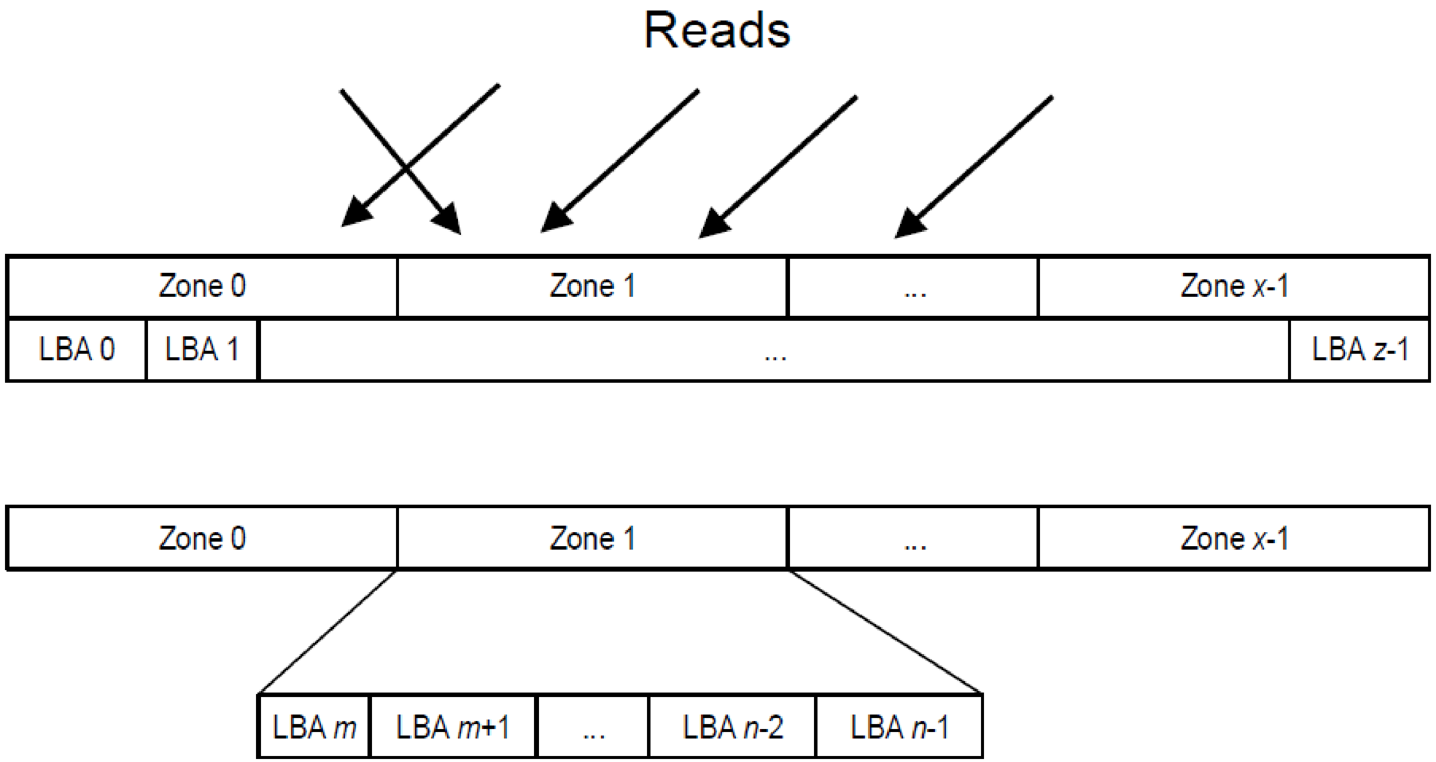
Writing to a Zone

- "Sequential Write Required"
  - Must be written sequentially
  - Must be reset if written to again
- Each zone has a set of associated attributes:
  - Write pointer
  - Zone starting LBA
  - Zone capacity
  - Zone state
- Very similar to writing zones with host-managed SMR HDDs



Reading from a Zone

- Writes are required to be sequential within a zone
- Reads may be issued to any LBA within a zone and in any order

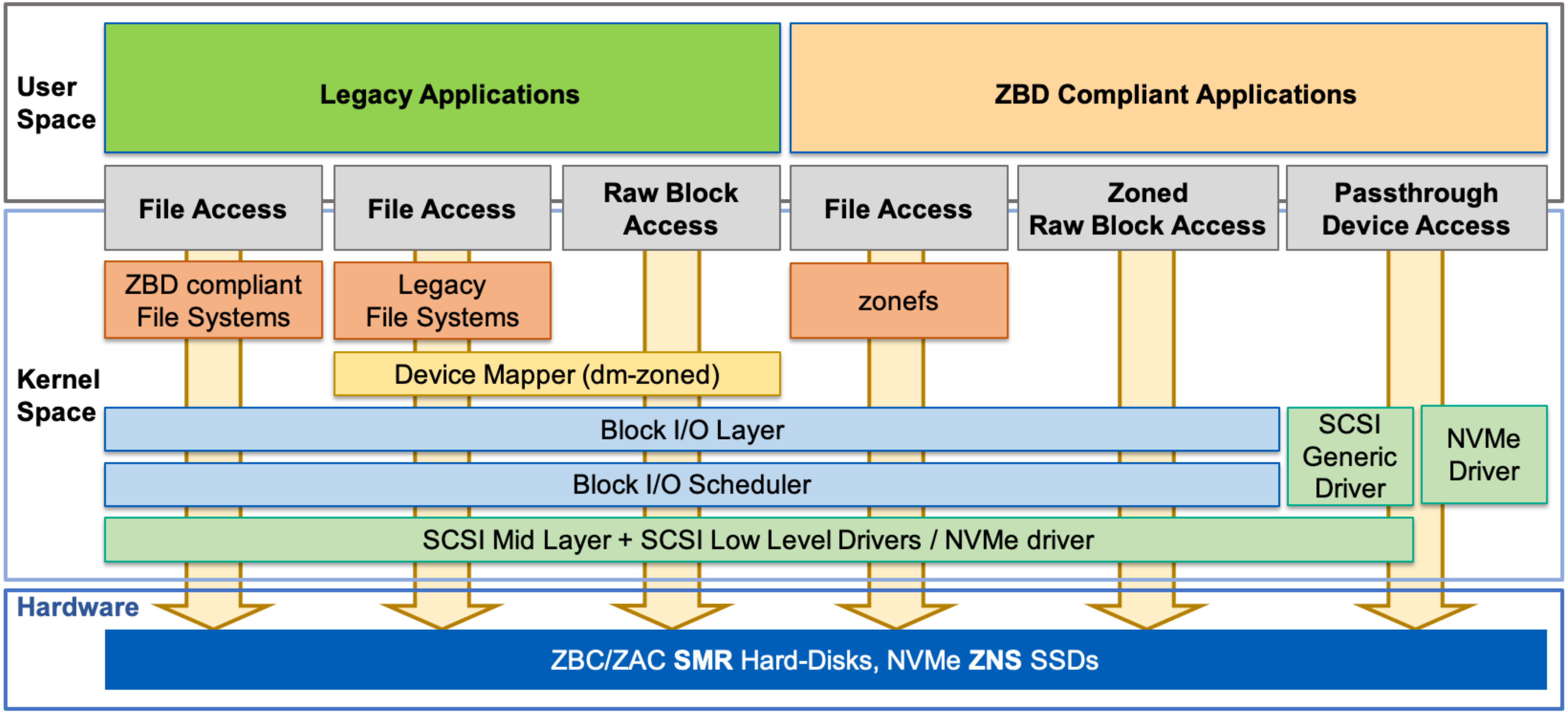


SMR HDDs and ZNS SSDs

- Host-managed SMR HDDs
  - Implements the SMR (ZAC/ZBC) specifications
  - ZAC: Zoned Device ATA Command Set in T13/SATA
  - ZBC: Zoned Block Commands in T10/SAS
- NVMe ZNS SSDs
  - Implement the Zoned NameSpace Command Set specification
  - Aligned with ZAC/ZBC to allow interoperability
- A single unified software stack support both storage types
  - Utilizes the already mature Linux storage stack built for SMR HDDs

Image from https://blog.westerndigital.com/what-is-zoned-storage-initiative/

Linux Zoned Block Device Support



https://zonedstorage.io/linux/overview/

Use Cases

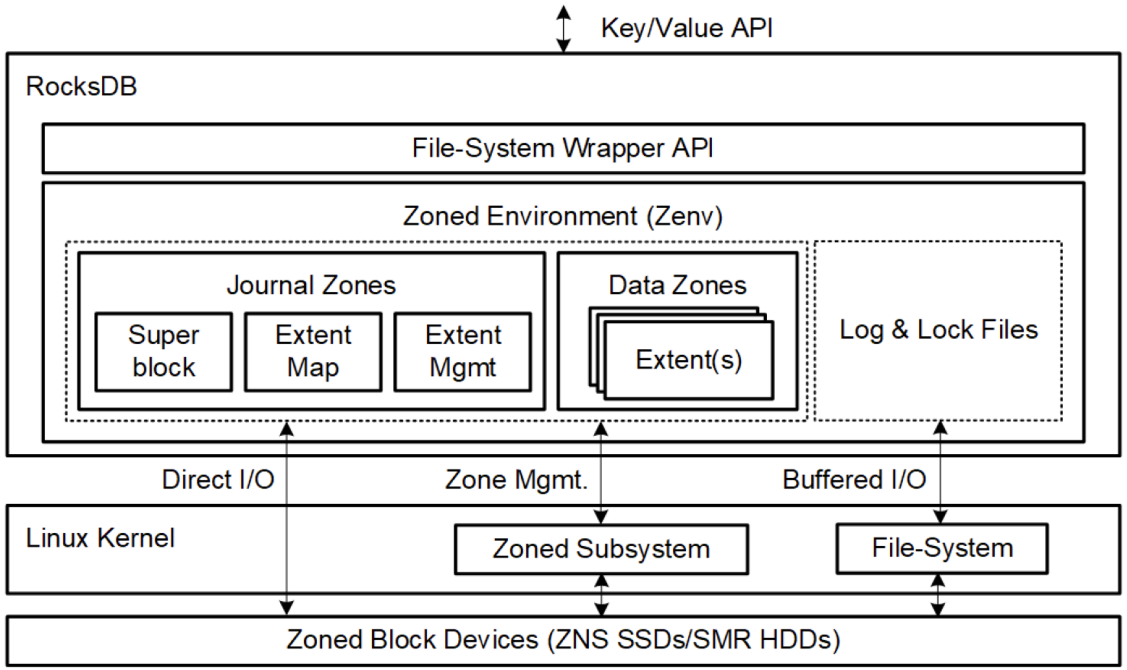
- **Host-side FTL**
  - Exposes the ZNS SSD as a conventional block SSD
  - High system overhead (DRAM and CPU)
  - For workloads with random write characteristics

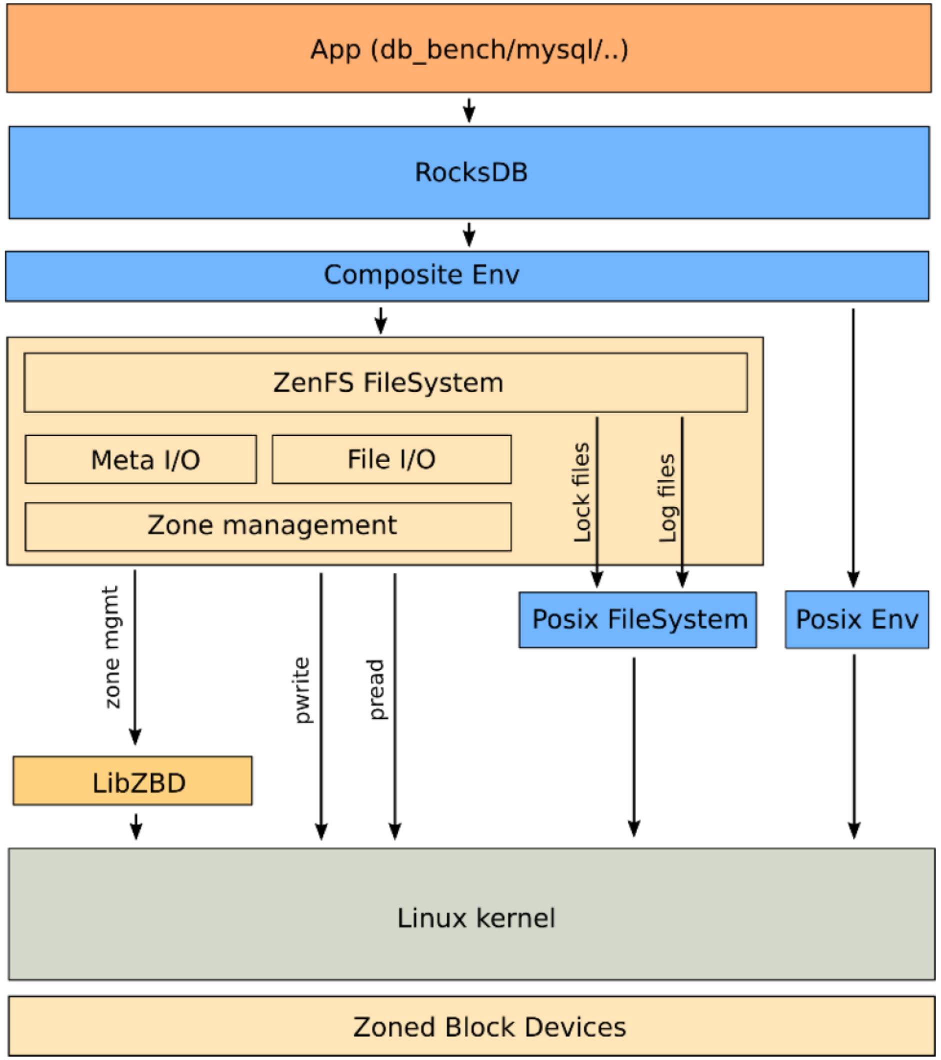
- **File systems**
  - ZNS SSD-aware file systems (e.g., f2fs + zones)
  - Efficient use of resources
  - Some inefficient data placement causes host GC

- **End-to-end data placement**
  - Application-specific data placement (e.g., RocksDB + ZenFS)
  - No indirection overhead caused by FTL nor file system
  - Highest performance, lowest write amplification

RocksDB on ZNS SSD

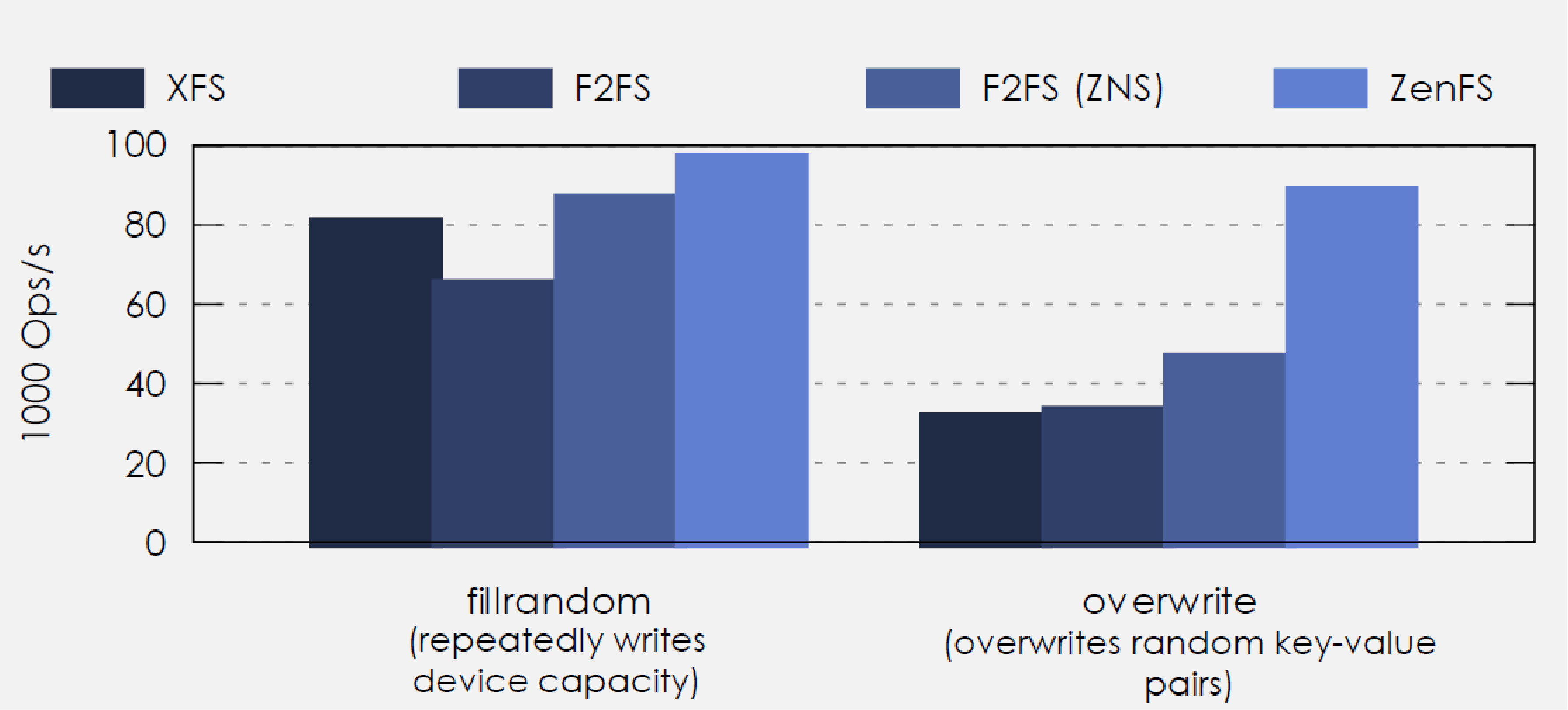
- ZenFS
  - A storage backed for RocksDB
  - Extent-based
  - No GC





RocksDB: Writes

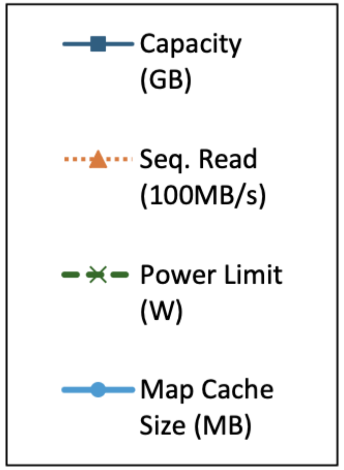
- Double the throughput over 28% OP SSDs
- Write amplification: ZNS 1.0x, XFS 2.0x, vanilla F2FS 2.4x

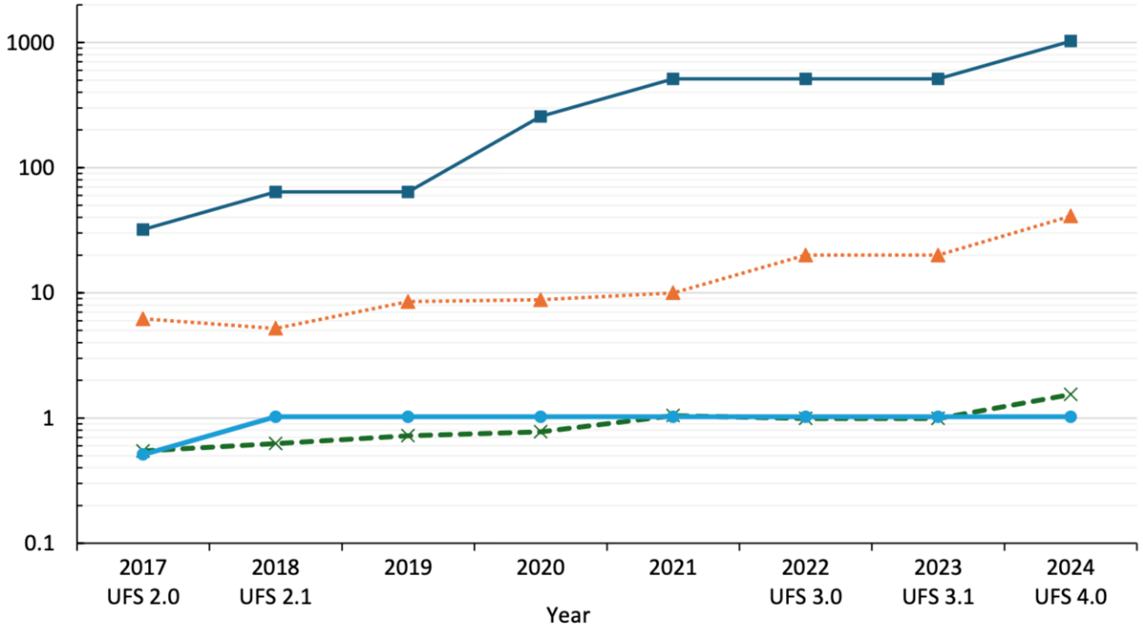


ZUFS (Zoned UFS) [FAST '26]

- Zone-level mapping (~ 8KB for 1TB)
- Efficient buffer management
- End-to-end write ordering guarantees
- F2FS with proactive GC

- Improved random reads
- Stable and predictable read latency
- Genshin game verification/loading: 35 sec → 30 sec
- Shipped in Google Pixel 10 Pro/XL





Summary: ZNS SSD

- ***What's good?***

- ***What's bad?***

FDP SSDs

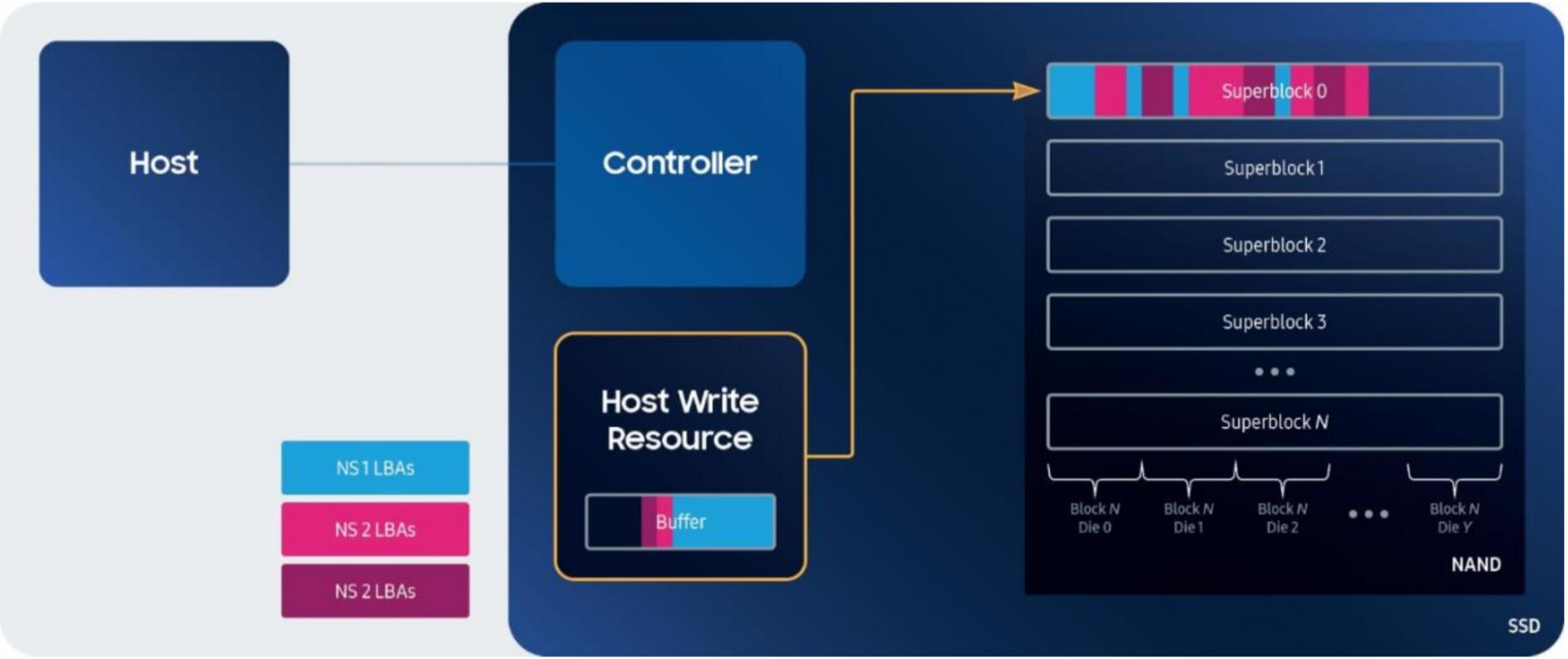
FDP (Flexible Data Placement) SSDs

- Motivation
  - Streams directives are not enforced (just hints)
  - ZNS is too strict; it requires a full rewrite of the storage stack

- Goals
  - Give host software enough control over NAND placement for WAF ≈ 1
  - Preserve backward compatibility
  - First-class host stack support: Linux io_uring passthru, SPDK, fio, nvme-cli, QEMU

- NVMe TP4161

Conventional SSD

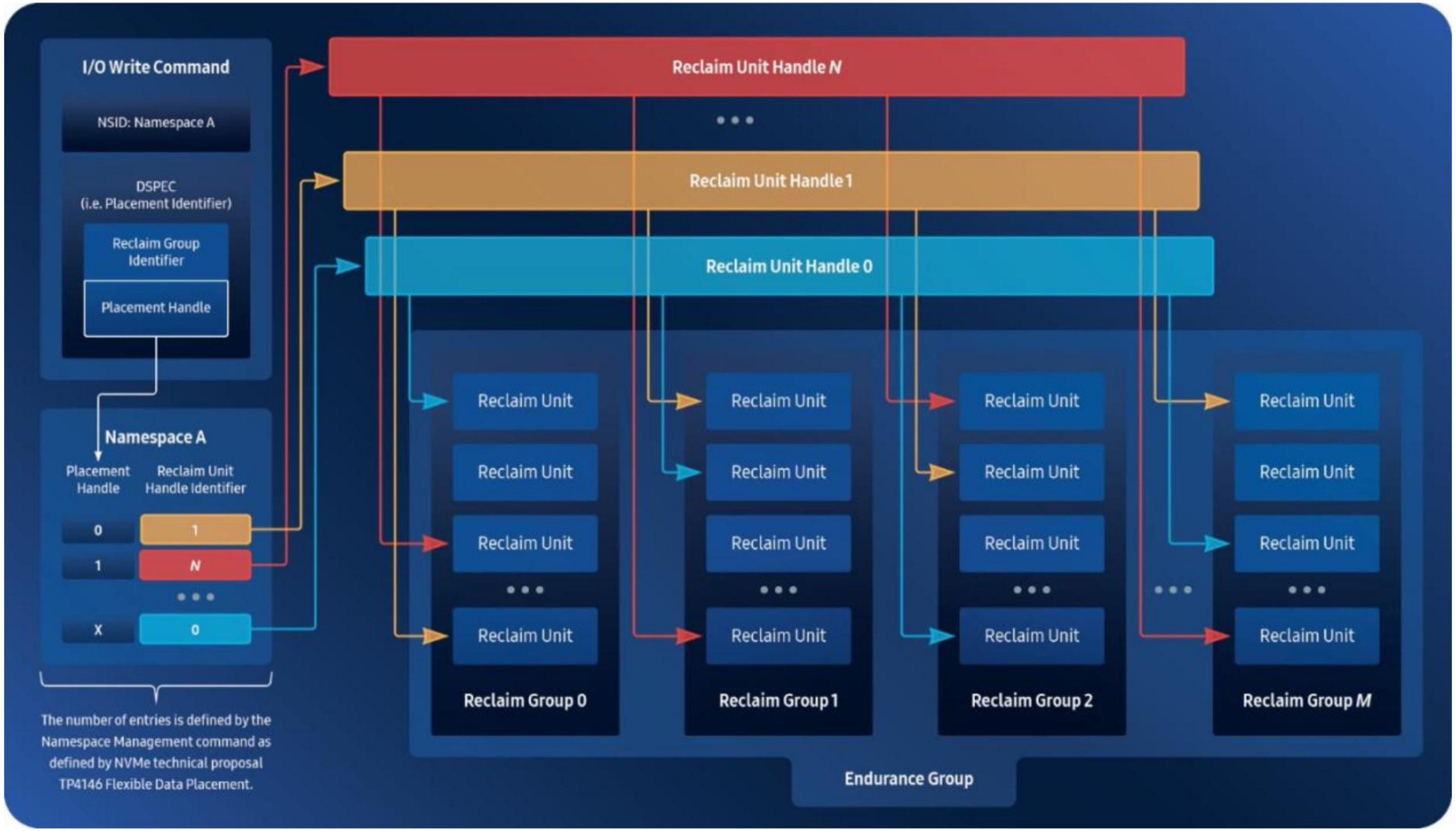


Introduction to Flexible Data Placement: A New Era of Optimized Data Management, Samsung Electronics, 2023.

FDP Architecture

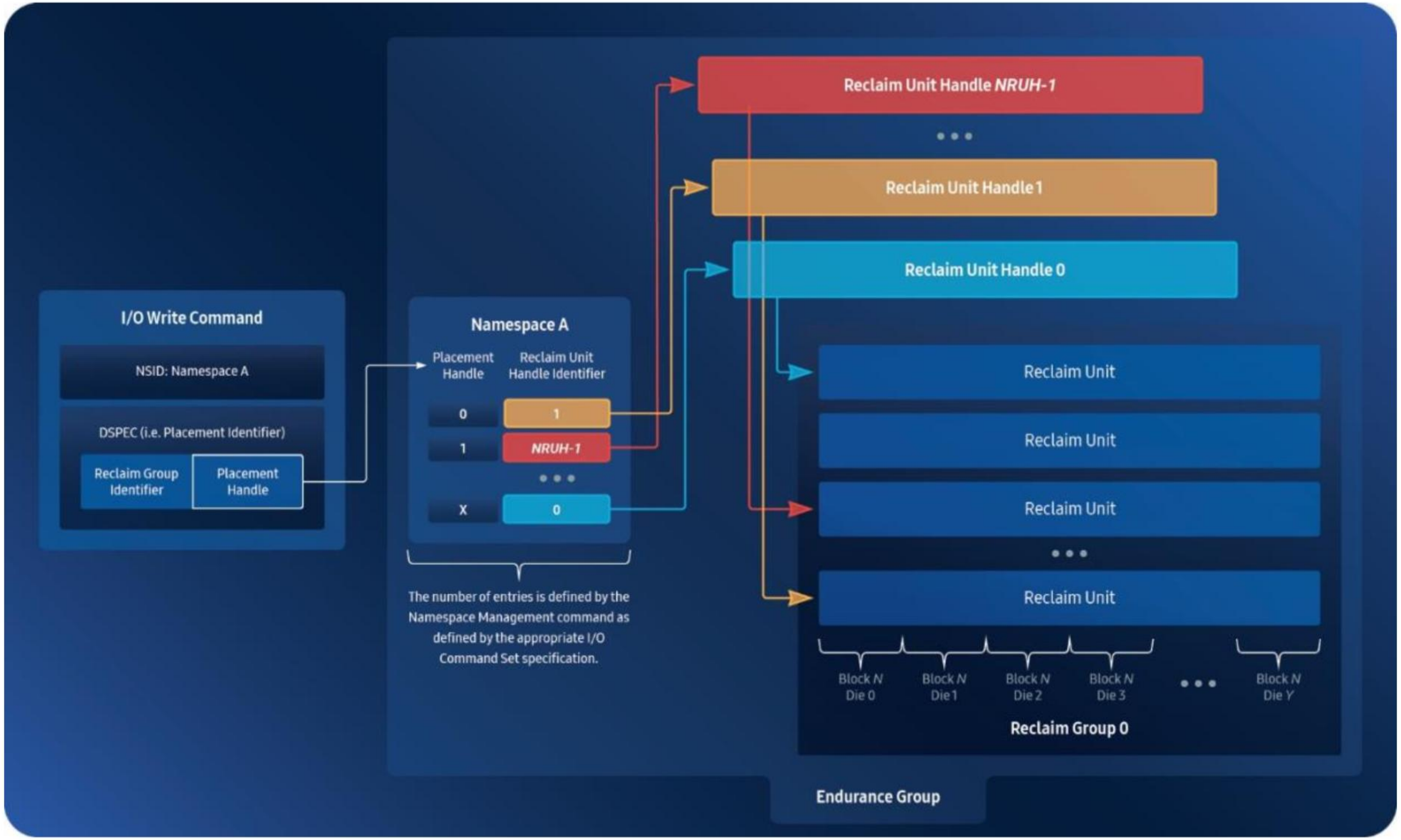
- Reclaim Unit (RU)
  - A set of NAND blocks that the host can write logical blocks into
- Reclaim Unit Handle (RUH)
  - A resource within the SSD to manage and buffer the logical blocks to write to a RU
  - A namespace may be attached to one or more RUHs
- Reclaim Group (RG)
  - A collection of Reclaim Units
  - Single RG: SSD manages parallelism
  - Multiple RG: Host controls per-die placement
- Placement handle
  - An index into the list of Reclaim Unit Handles

FDP with Multiple RG



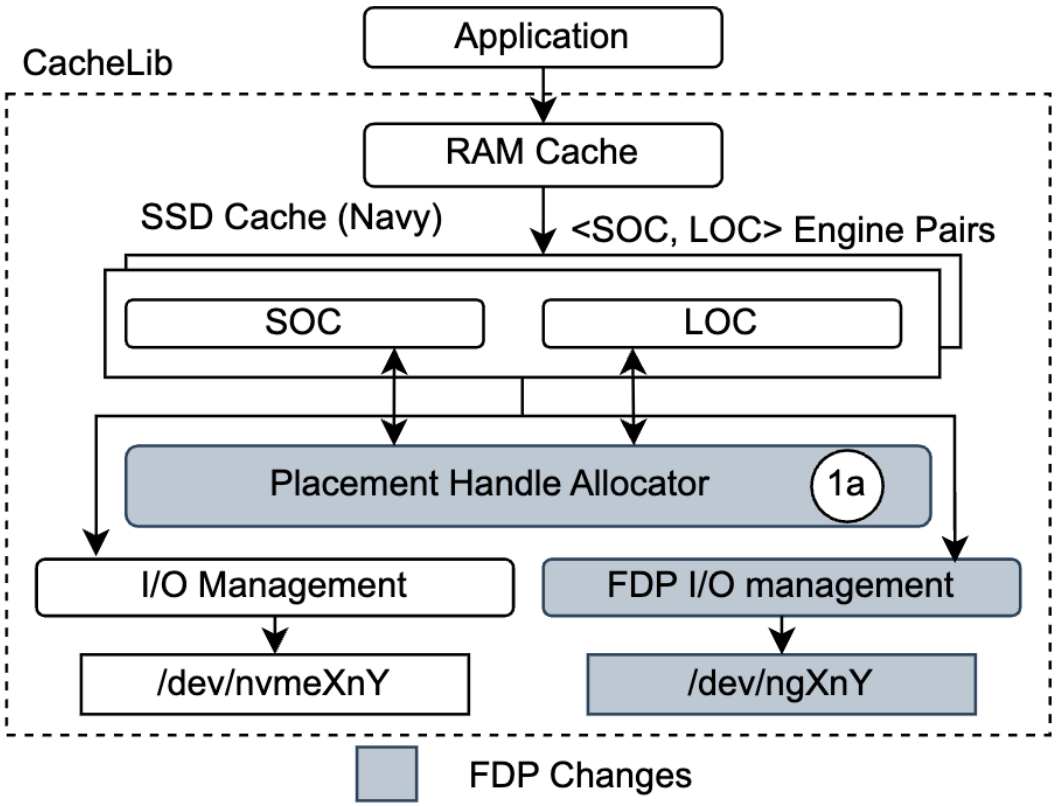
Introduction to Flexible Data Placement: A New Era of Optimized Data Management, Samsung Electronics, 2023.

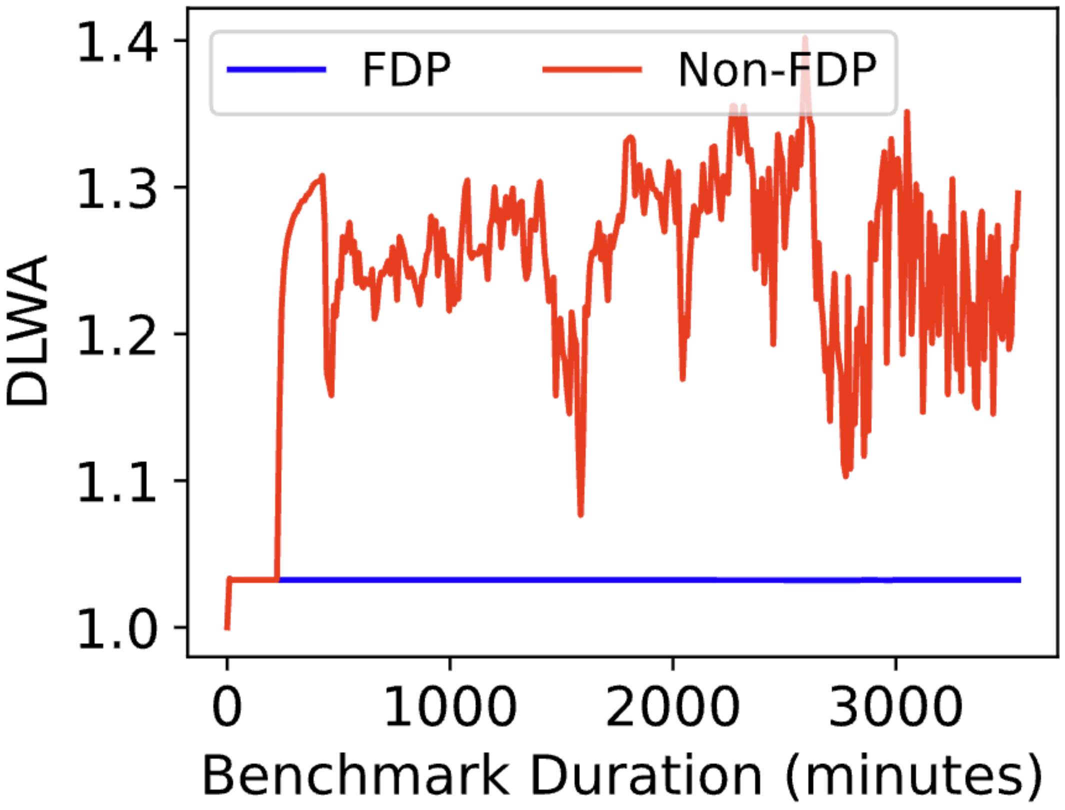
FDP with Single RG



Introduction to Flexible Data Placement: A New Era of Optimized Data Management, Samsung Electronics, 2023.

CacheLib Results





M Allison et al., "Towards Efficient Flash Caches with Emerging NVMe Flexible Data Placement SSDs," EuroSys, 2025.

Summary

Characteristic	**Streams** [40]	**Open-Channel** [25]	**ZNS** [1, 24]	**FDP** [19]
Supported write patterns	Random, Sequential	Random, Sequential	Sequential	Random, Sequential
Data placement primitive	Using stream identifiers	Using logical to physical address mapping by host	Using zones	Using reclaim unit handles
Control of garbage collection	SSD-based without feedback to host	Host-based	Host-based	SSD-based with feedback through logs
NAND media management by host	No	Yes	No	No
Can run applications unchanged	Yes	No	No	Yes

M Allison et al., "Towards Efficient Flash Caches with Emerging NVMe Flexible Data Placement SSDs," EuroSys, 2025.