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

Systems Software &
Architecture Lab.

Seoul National University

Fall 2021

Modern SSDs

The Unwritten Contract

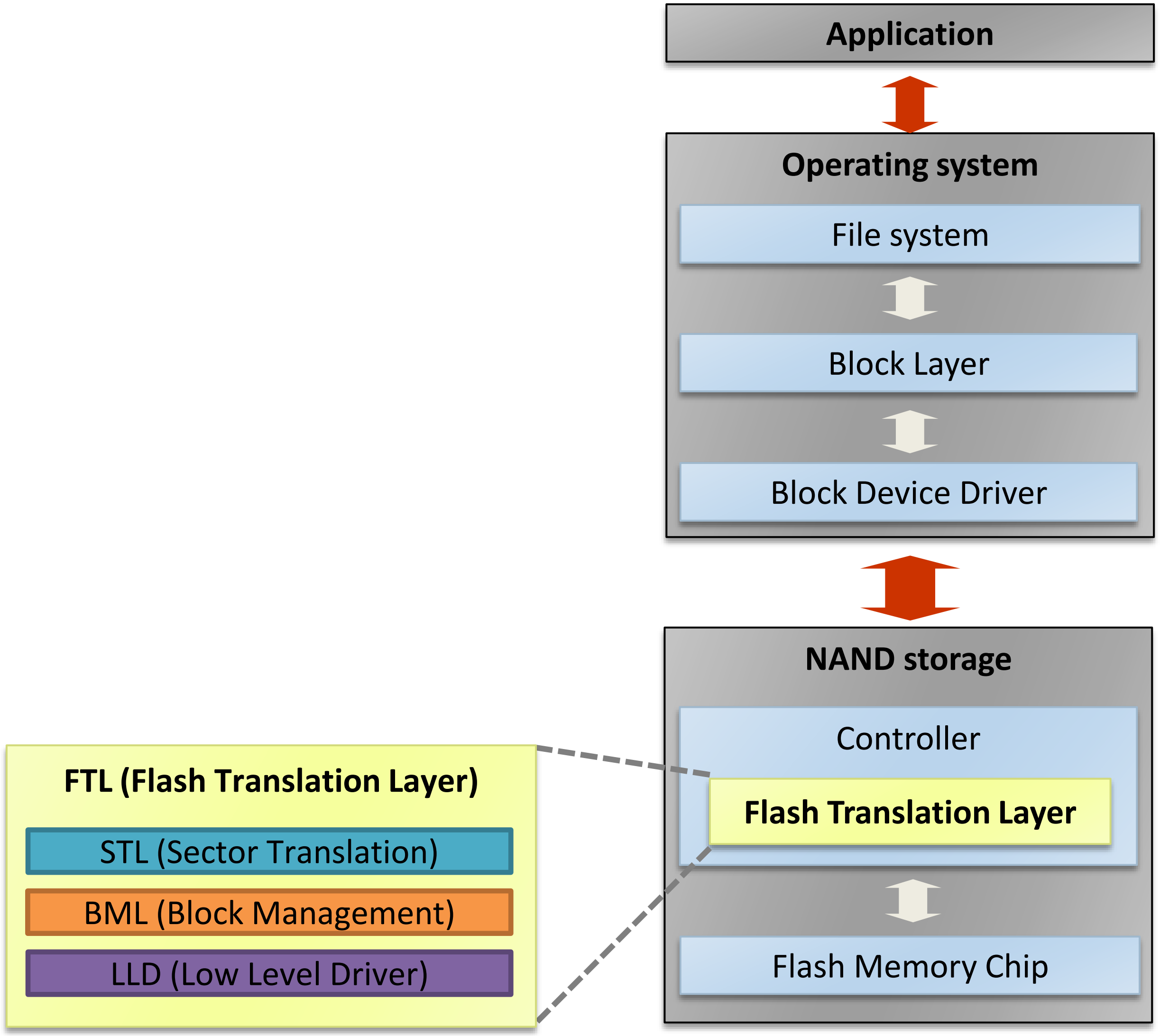
- Several assumptions are no longer valid

Assumptions	Disks	SSDs
Sequential accesses much faster than random	✔	✘
No write amplification	✔	✘
Little background activity	✔	✘
Media does not wear down	✔	✘
Distant LBNs lead to longer access time	✔	✘

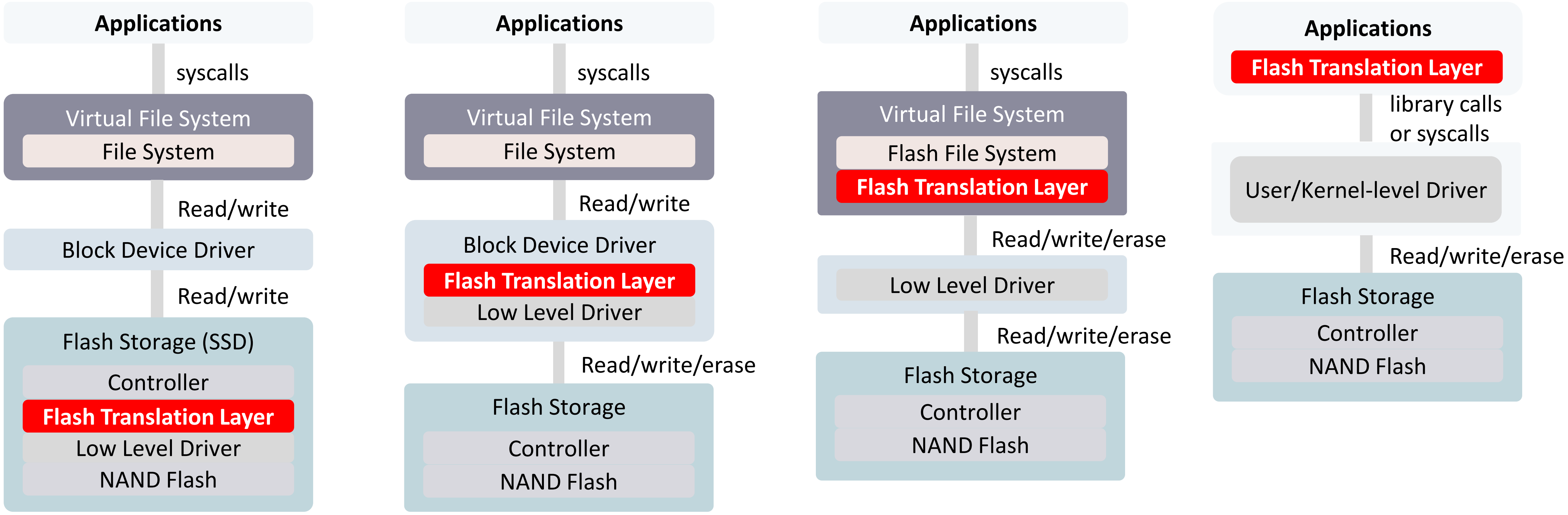
Source: A. Rajimwale et al., "Block Management in Solid-State Devices," USENIX ATC, 2009.

FTL Architecture

- **Sector Translation Layer**
  - Address mapping
  - Garbage collection
  - Wear leveling
- **Block Management Layer**
  - Bad block management
  - Error handling
- **Low Level Driver**
  - Flash interface



Approaches



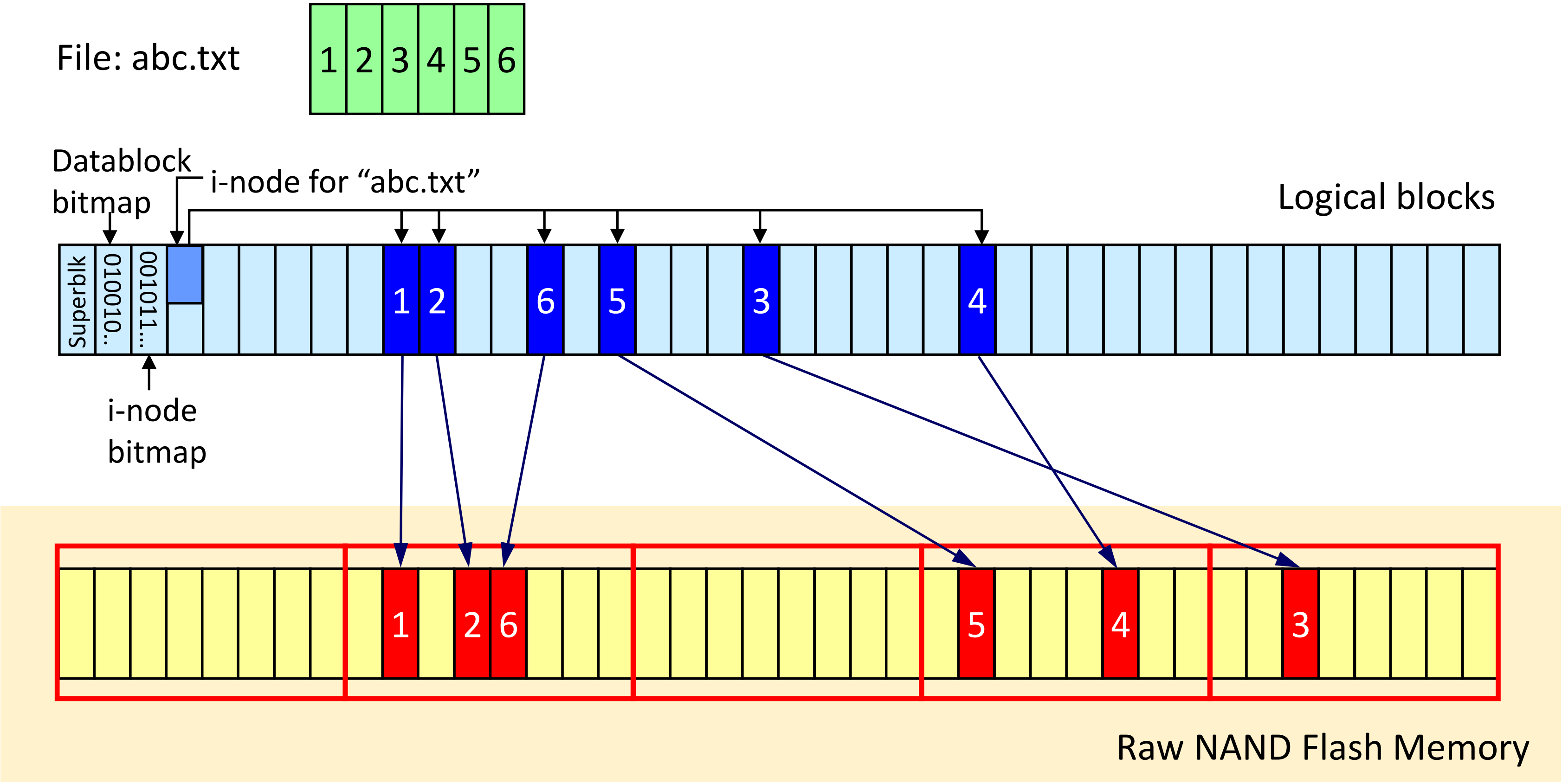
FTL on Device | **FTL on Host** | **Flash File System** | **Flash-aware App.**

FTL on Device

- Flash cards or flash SSDs are already equipped with FTL
- ***Benefits?***
- ***Limitations?***
- Hints from file systems or applications can be useful
  - TRIM, Stream, …

FTL on Device: Example

- What happens on file deletion?



TRIM

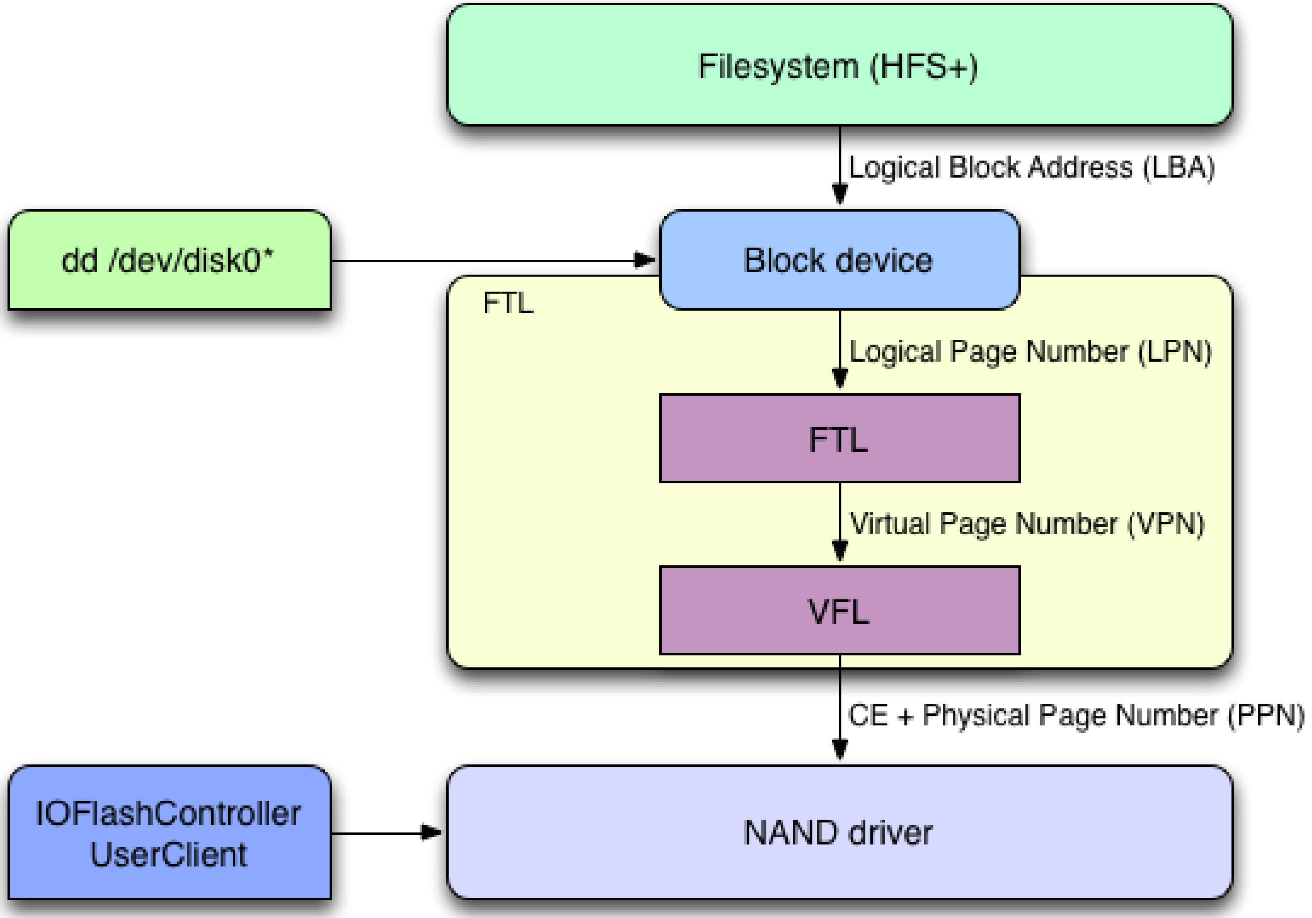
- ATA interface standard (T13 technical committee)
  - "The data in the specified sectors is no longer needed"
  - Originally proposed as a non-queued command, but SATA 3.1 introduces the queued TRIM command
  - UNMAP, WRITE SAME with unmap flag in SCSI, DEALLOCATE in NVMe
- Types
  - Non-deterministic Trim: reads may return different data
  - Deterministic Trim: reads return the same data
  - Deterministic Read Zero after Trim: all reads shall return zero
- TRIM commands can be automatically issued on file deletion or format
- fstrim: discard unused blocks on a mounted file system

FTL on Host

- FusionIO DFS, Apple APFS / HFS+

- ***Benefits?***

- ***Limitations?***



Flash File Systems

- Kernel manages raw flash memory directly
- Cross-layer optimization possible
  - Example: file data indexing
    - Legacy file system: <inode #, block #> → <LBA> → <flash block #, flash page #>
    - Flash file system: <inode #, block #> → <flash block #, flash page #>
  - ***What else?***
- Used in old embedded systems, but not so successful
  - JFFS2, YAFFS, UBIFS, …
  - ***Why?***

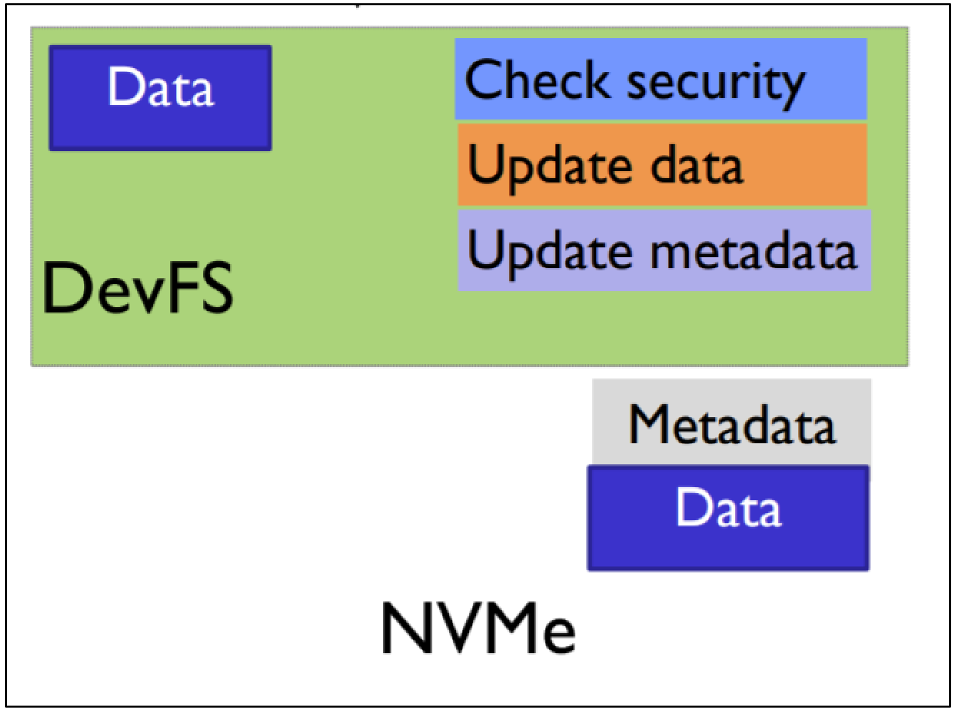
Flash-aware Applications

- Datacenter applications want to manage the underlying flash directly
  - ***Why?***

- Diverse proposals on flash interface
  - Software Defined Flash [ASPLOS '14]
  - Application-managed Flash [FAST '16]
  - Open-Channel SSD [FAST '17]
  - ZNS(Zoned-NameSpace) SSD [ATC '21]

Another Extreme: DevFS

- Device-level File System [FAST '18]
  - Move file system into the device hardware
  - Use device-level CPU and memory for DevFS
  - Apps. bypass OS for control and data plane
  - DevFS handles integrity, concurrency, crash-consistency, and security
  - Achieves true direct-access



- Challenges
  - Limited memory inside the device
  - DevFS lack visibility to OS state

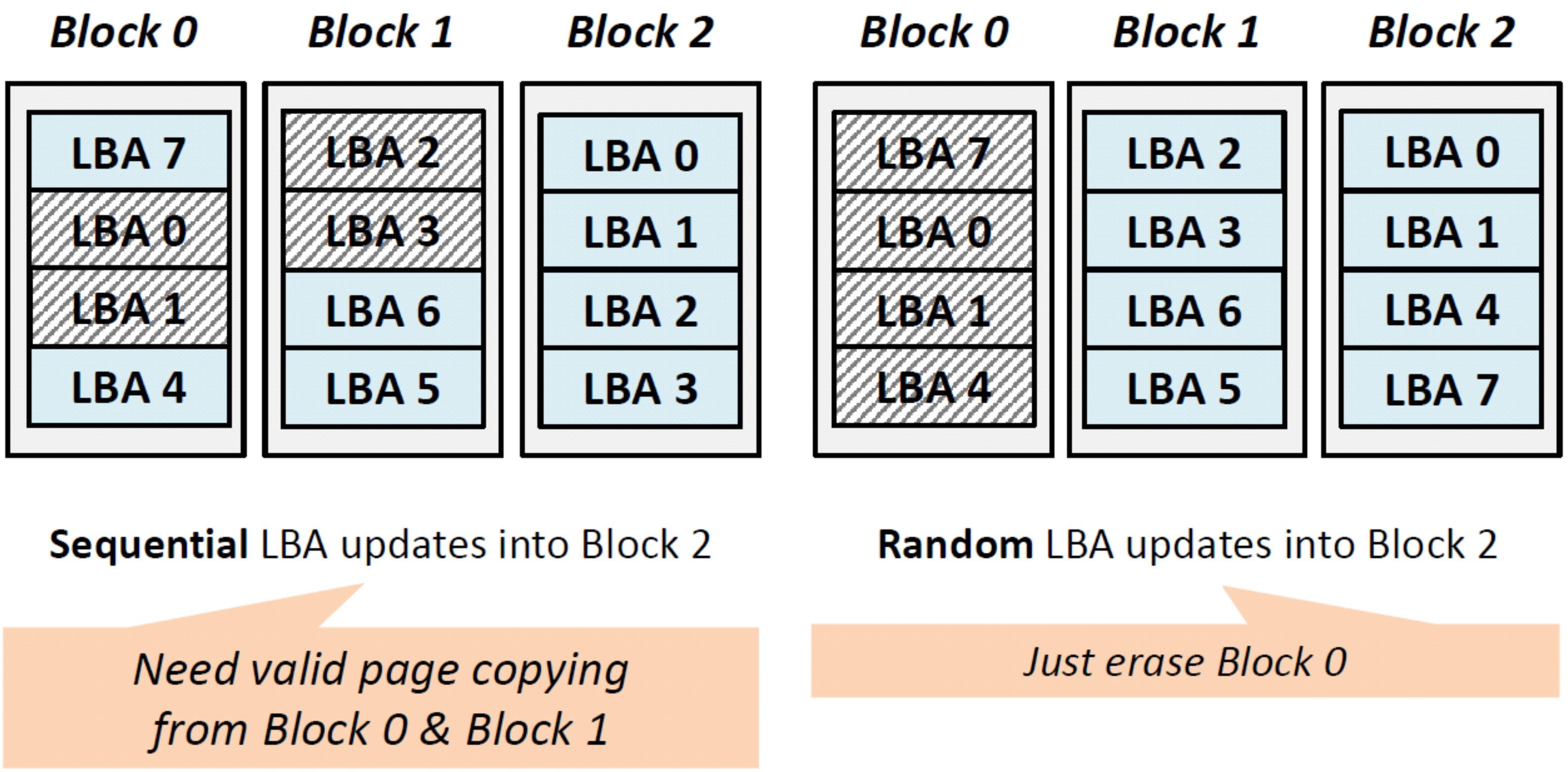
The Multi-streamed Solid-State Drive

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

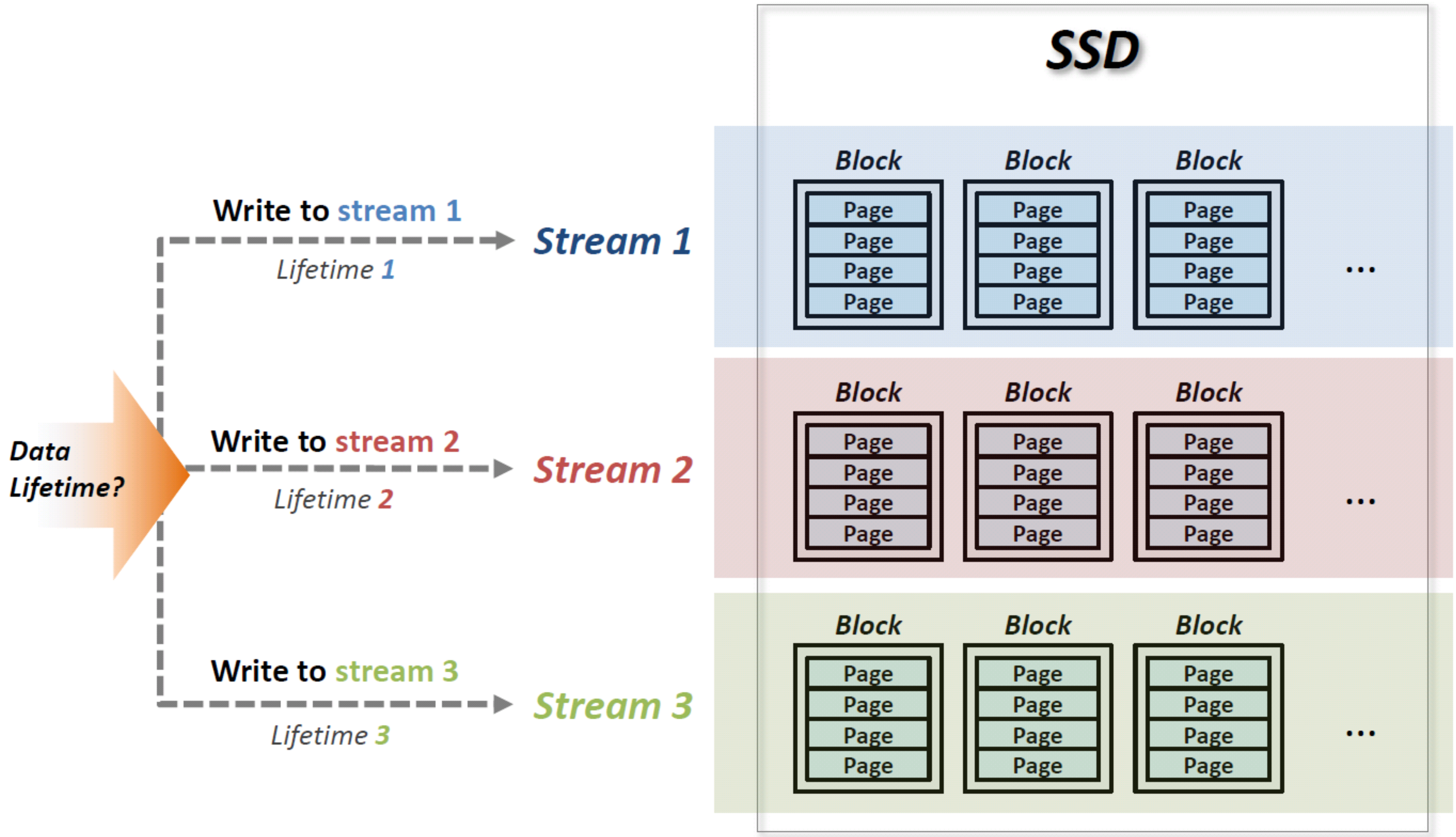
Some of slides are borrowed from the authors' presentation.

Effects of Write Patterns

- Previous write patterns (= current state) matter

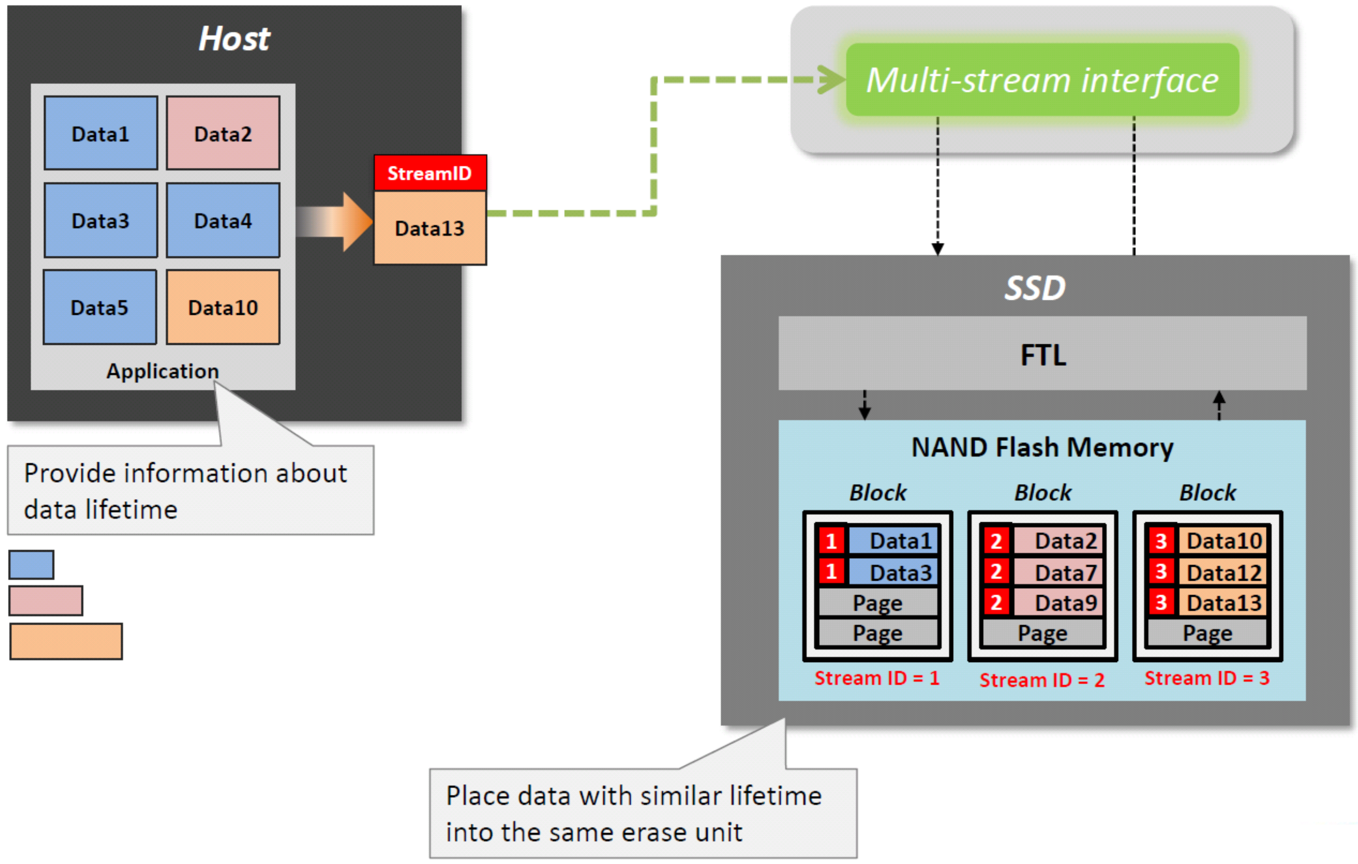


Stream


SSD
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
Block
Page
Page
Page
Page
...

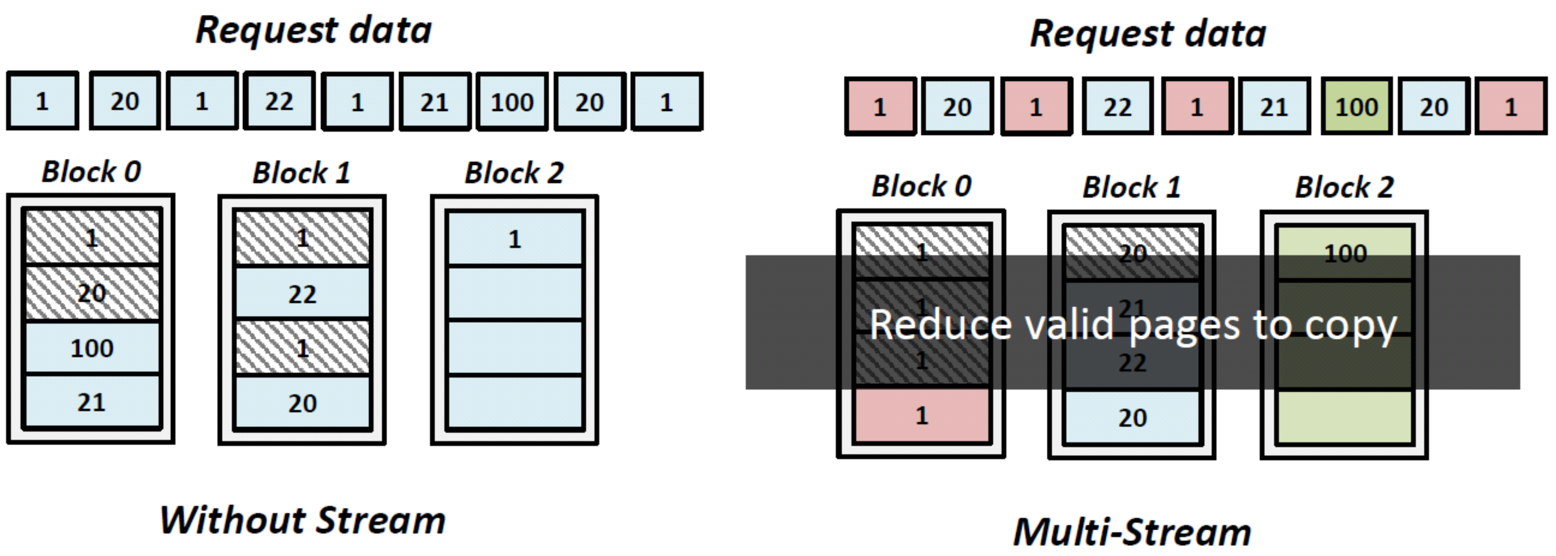
The Multi-streamed SSD

- Mapping data with different lifetime to different streams



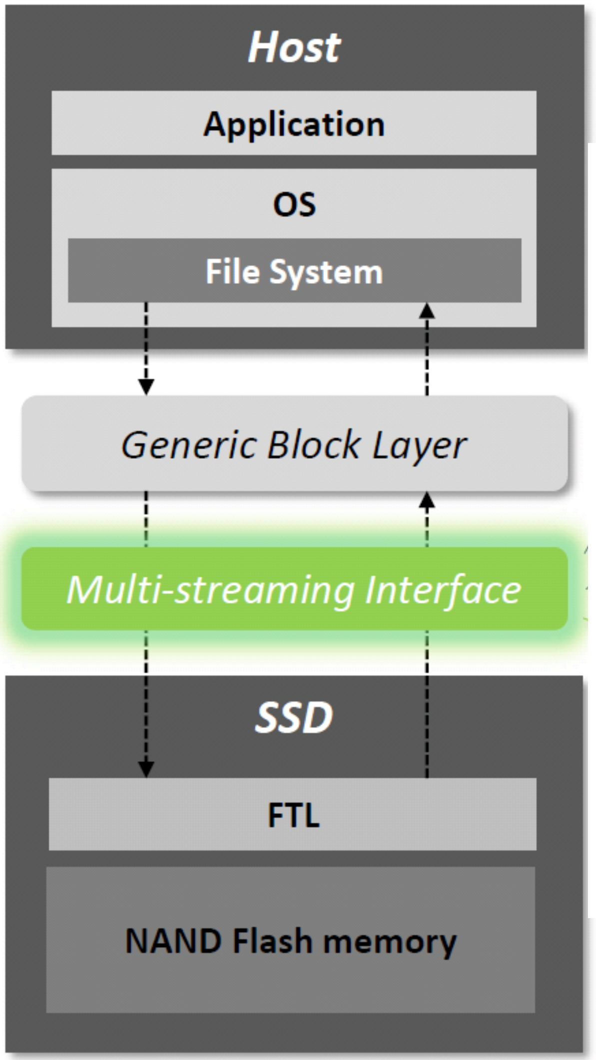
Working Example

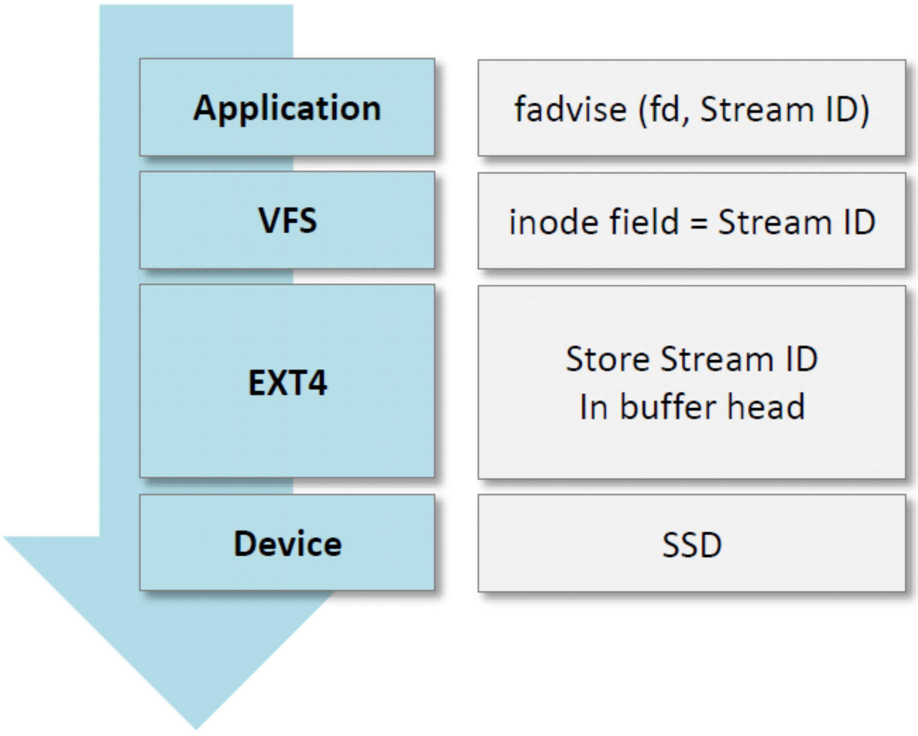
- High GC efficiency → Performance improvement



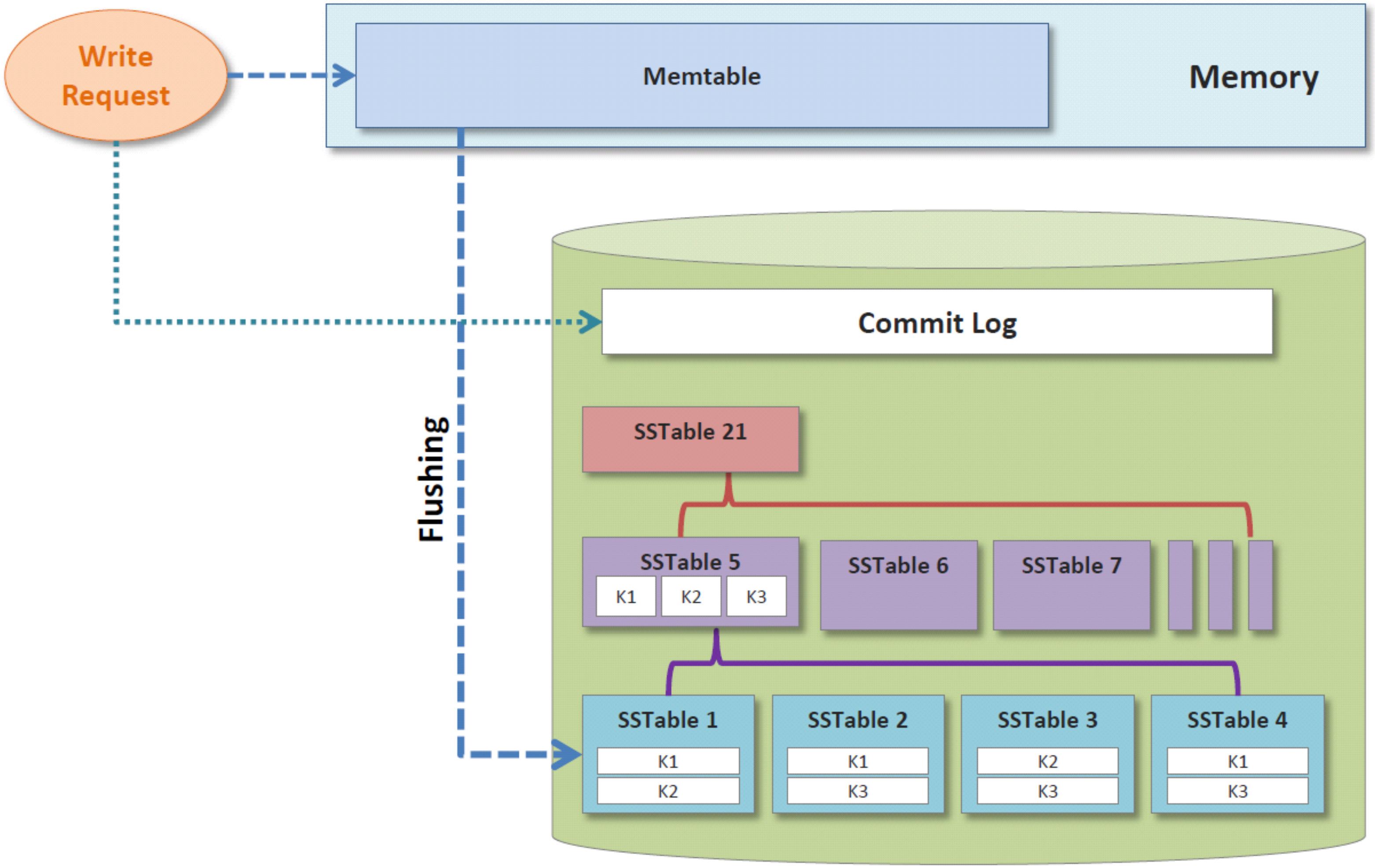
For effective multi-streaming,
proper mapping of data to streams is essential!

Architecture



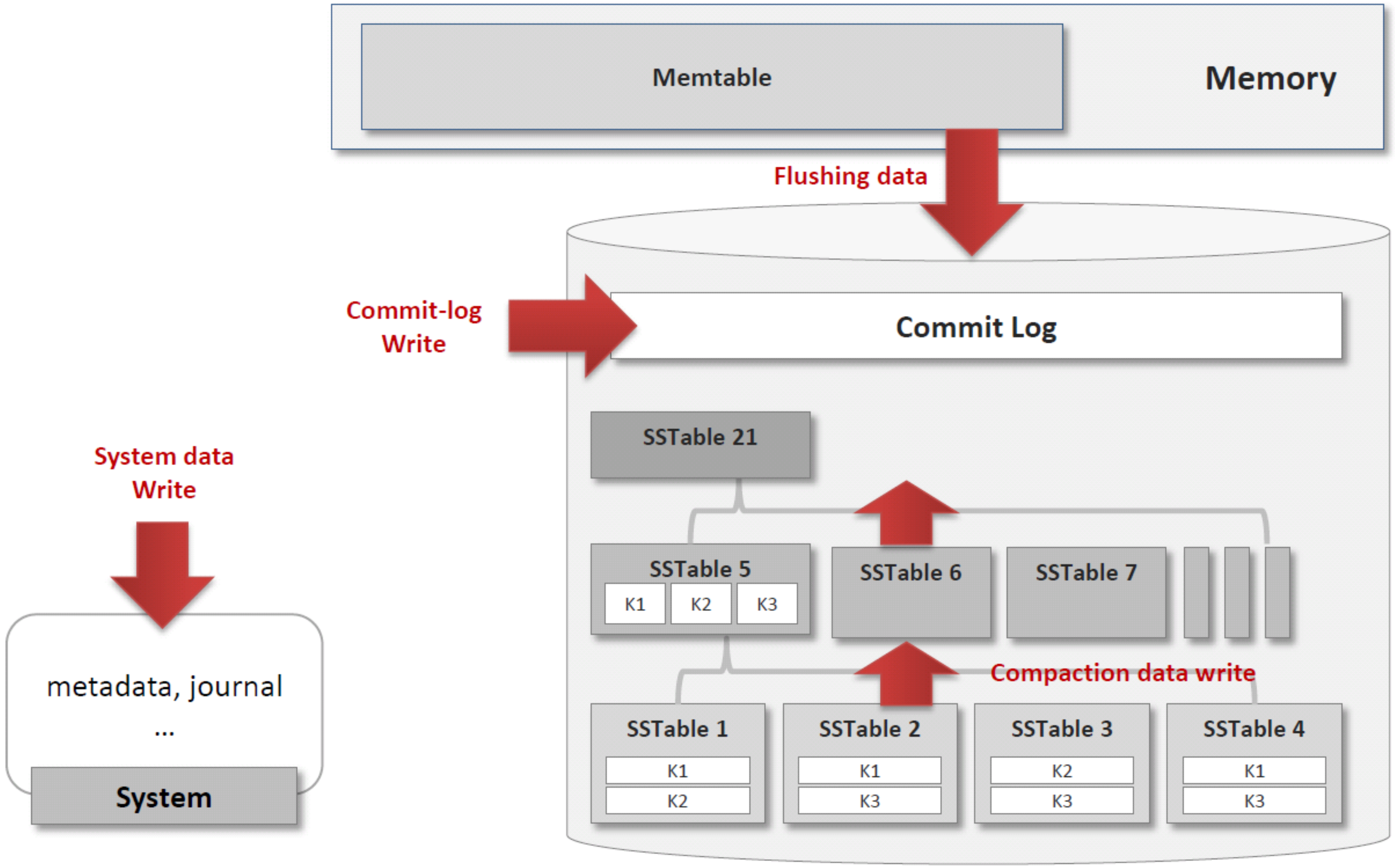


Case Study: Cassandra



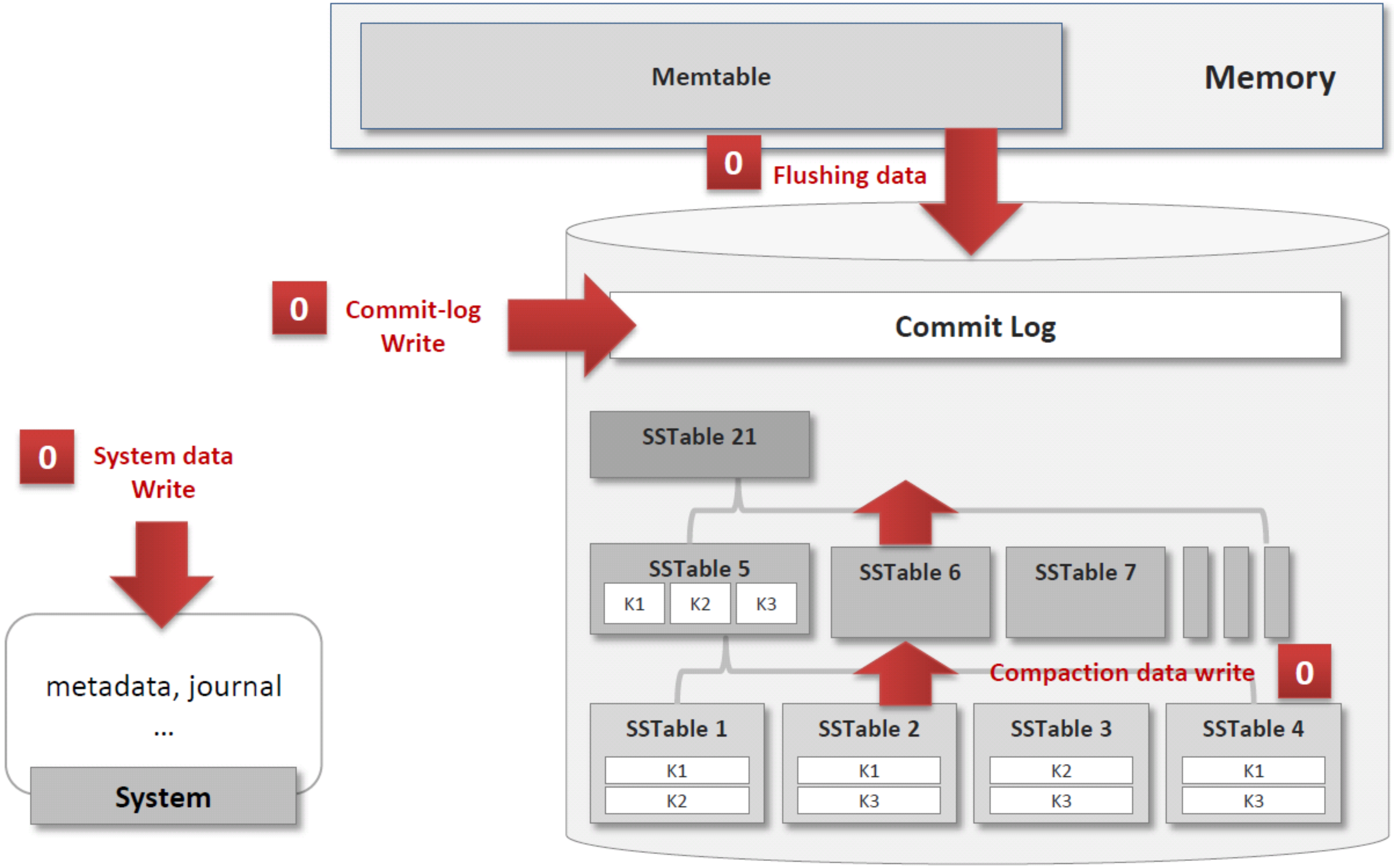
Cassandra's Write Patterns

- Write operations when Cassandra runs



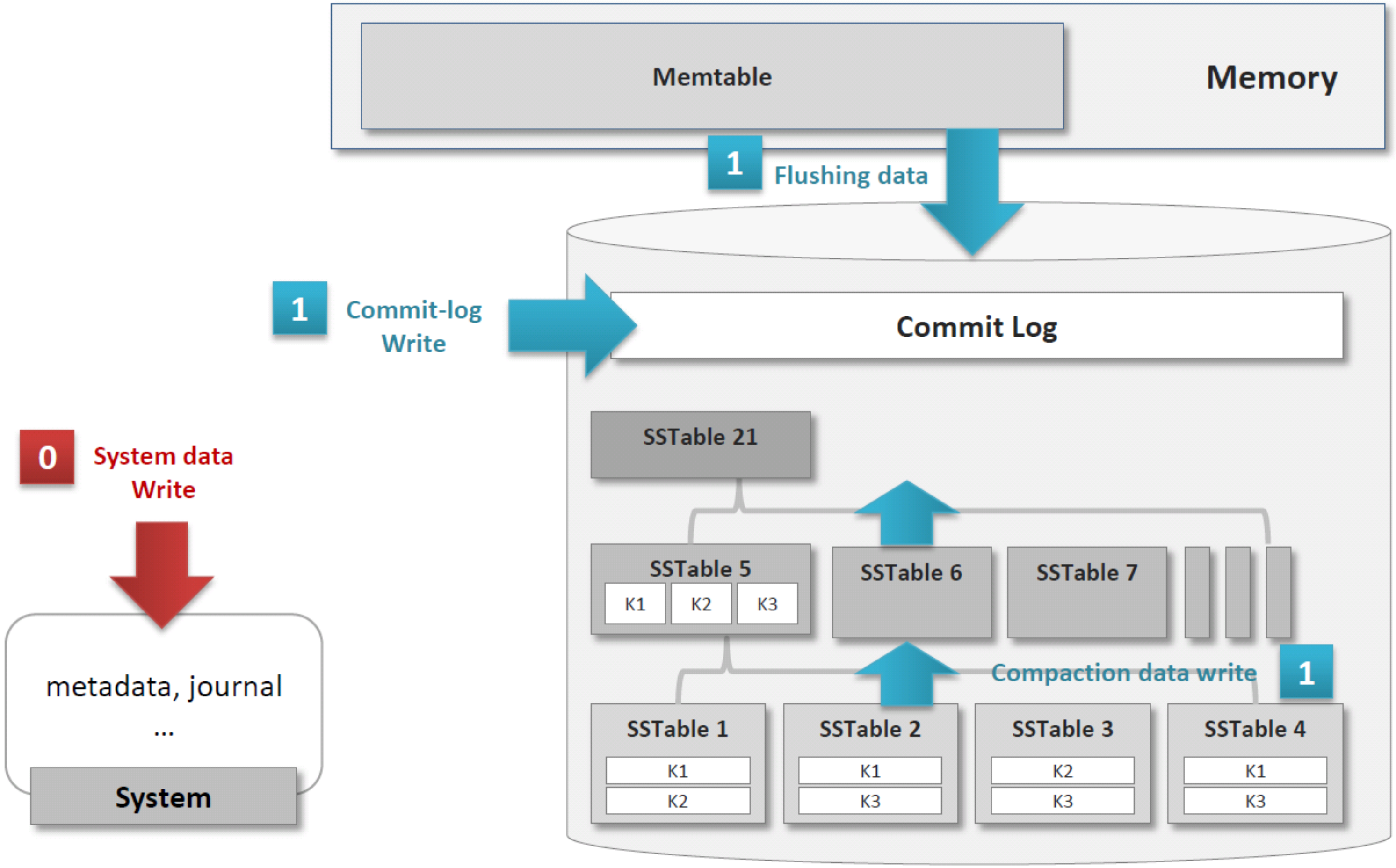
Mapping #1: Conventional

- Just one stream ID (= conventional SSD)



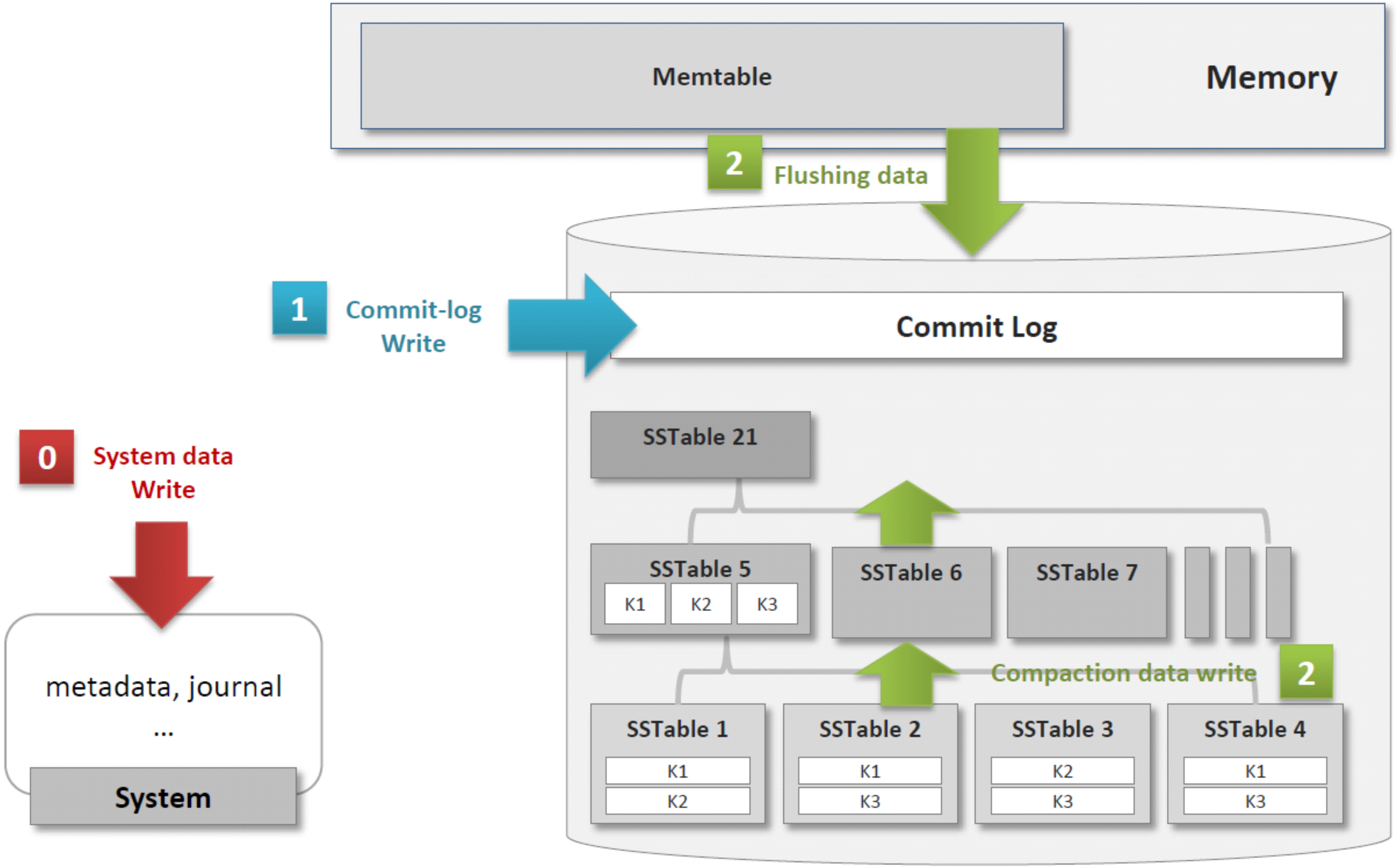
Mapping #2: Multi-App

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



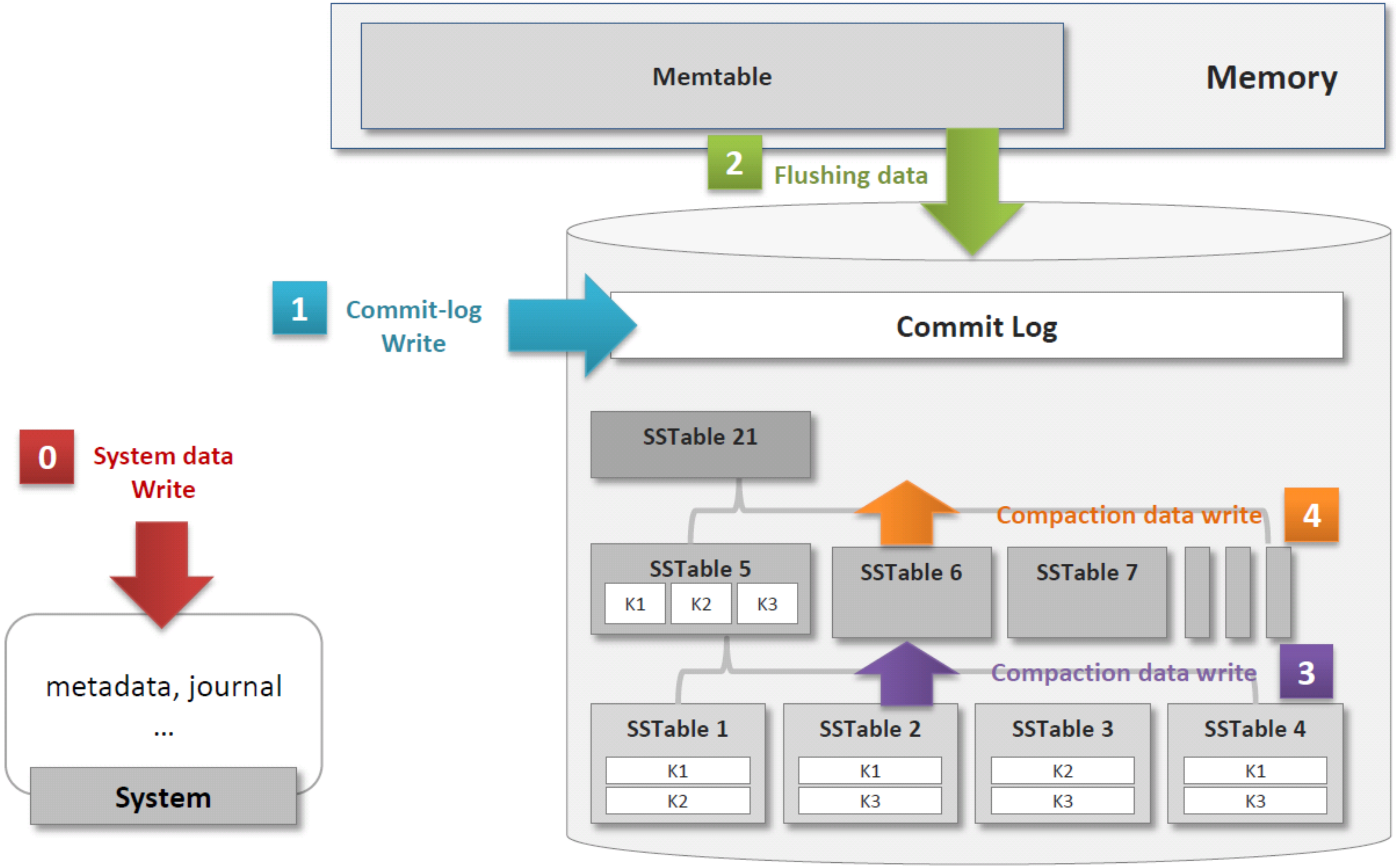
Mapping #3: Multi-Log

- Use three streams; further separate Commit Log



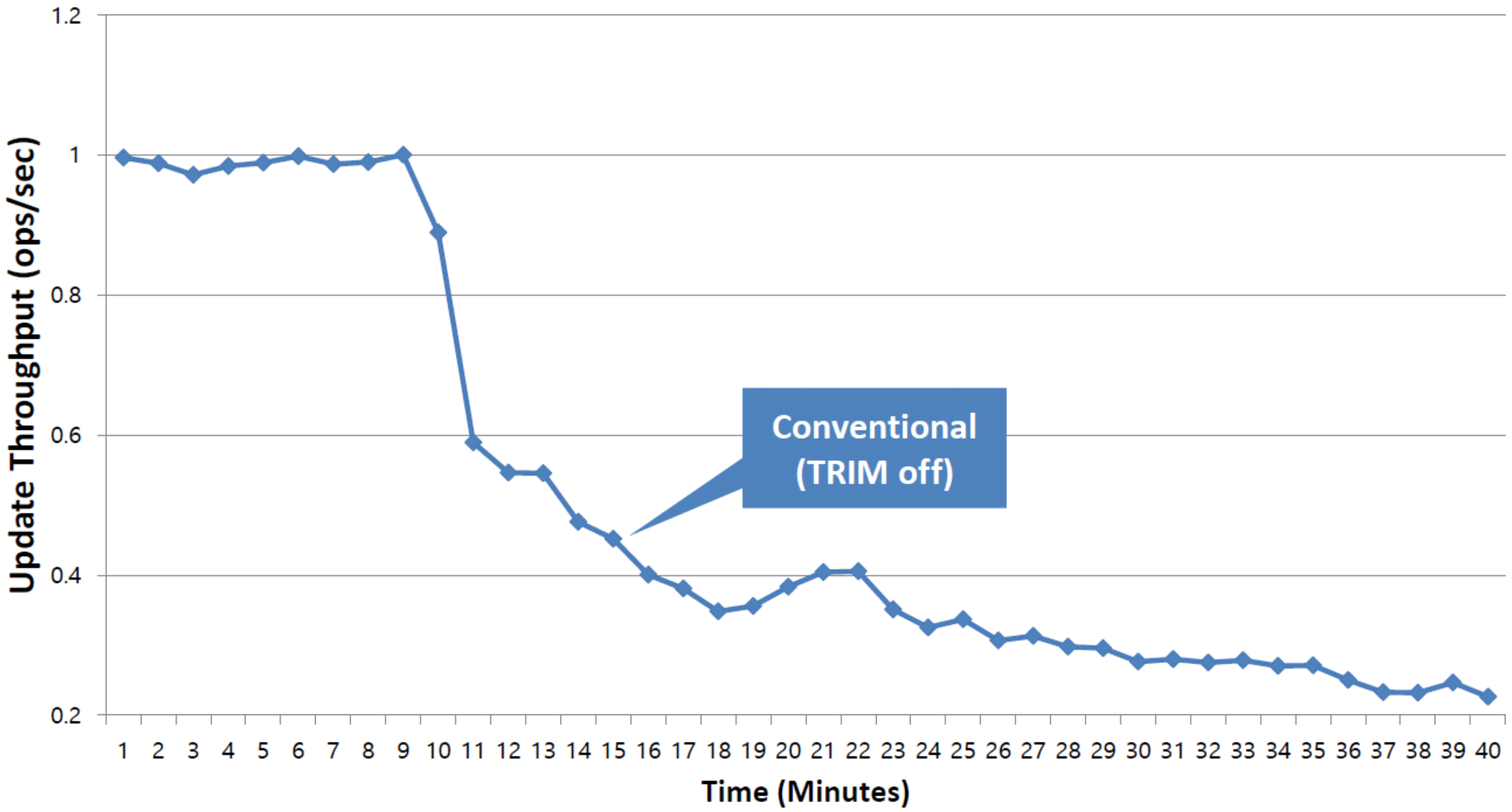
Mapping #4: Multi-Data

- Give distinct streams to different tiers of SSTables



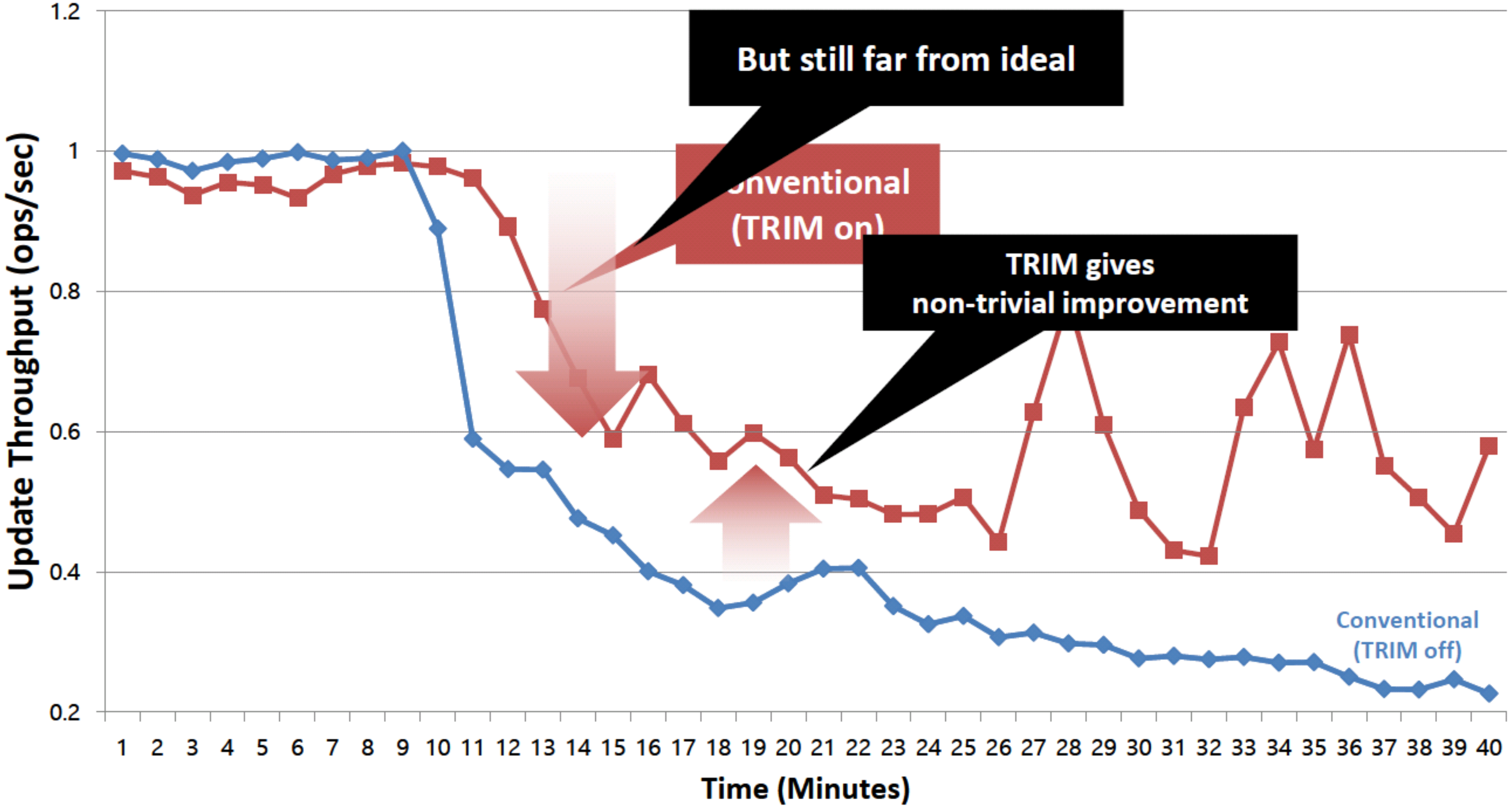
Results: Conventional

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



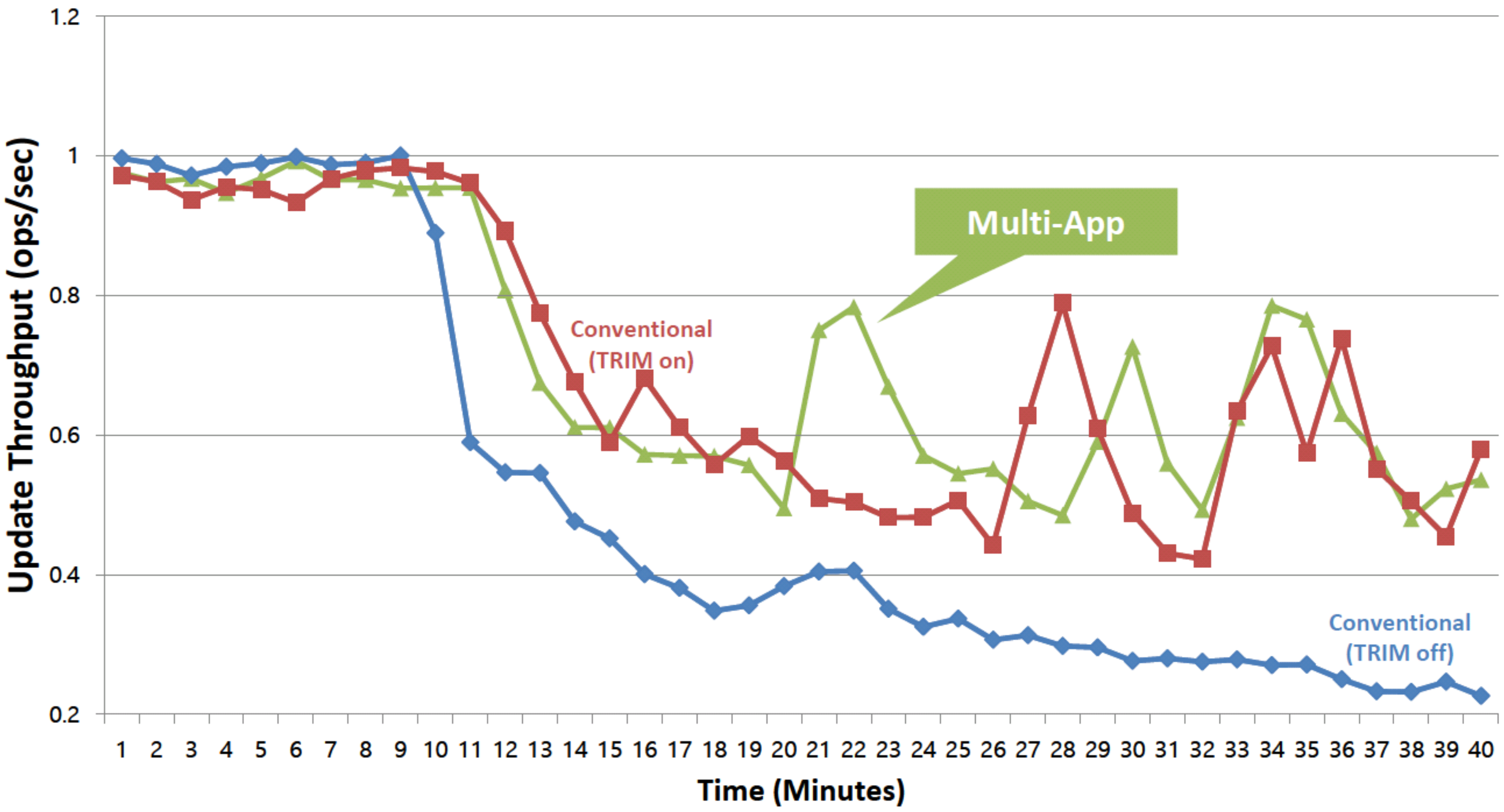
Results: Conventional with TRIM

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



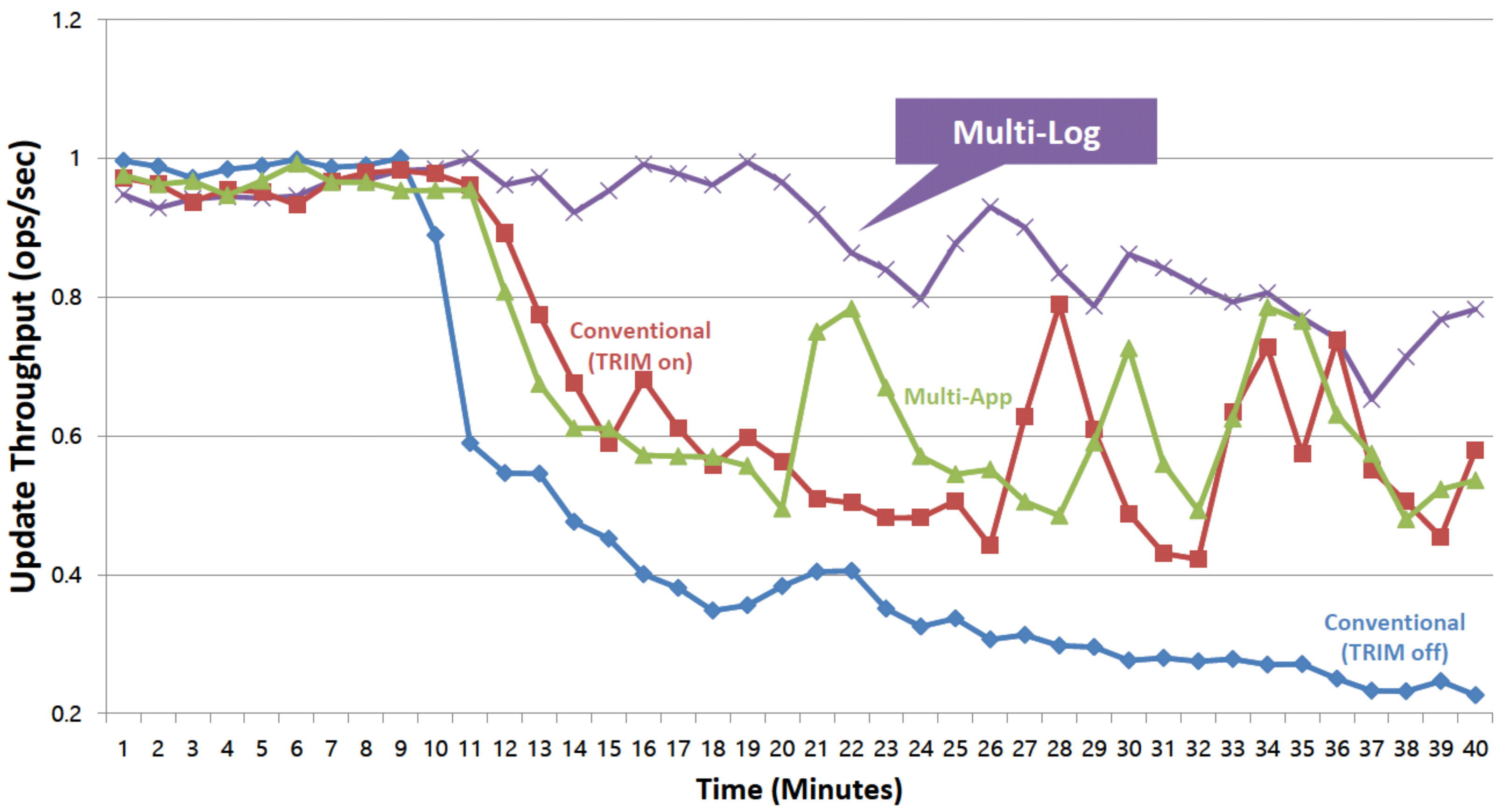
Results: Multi-App

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



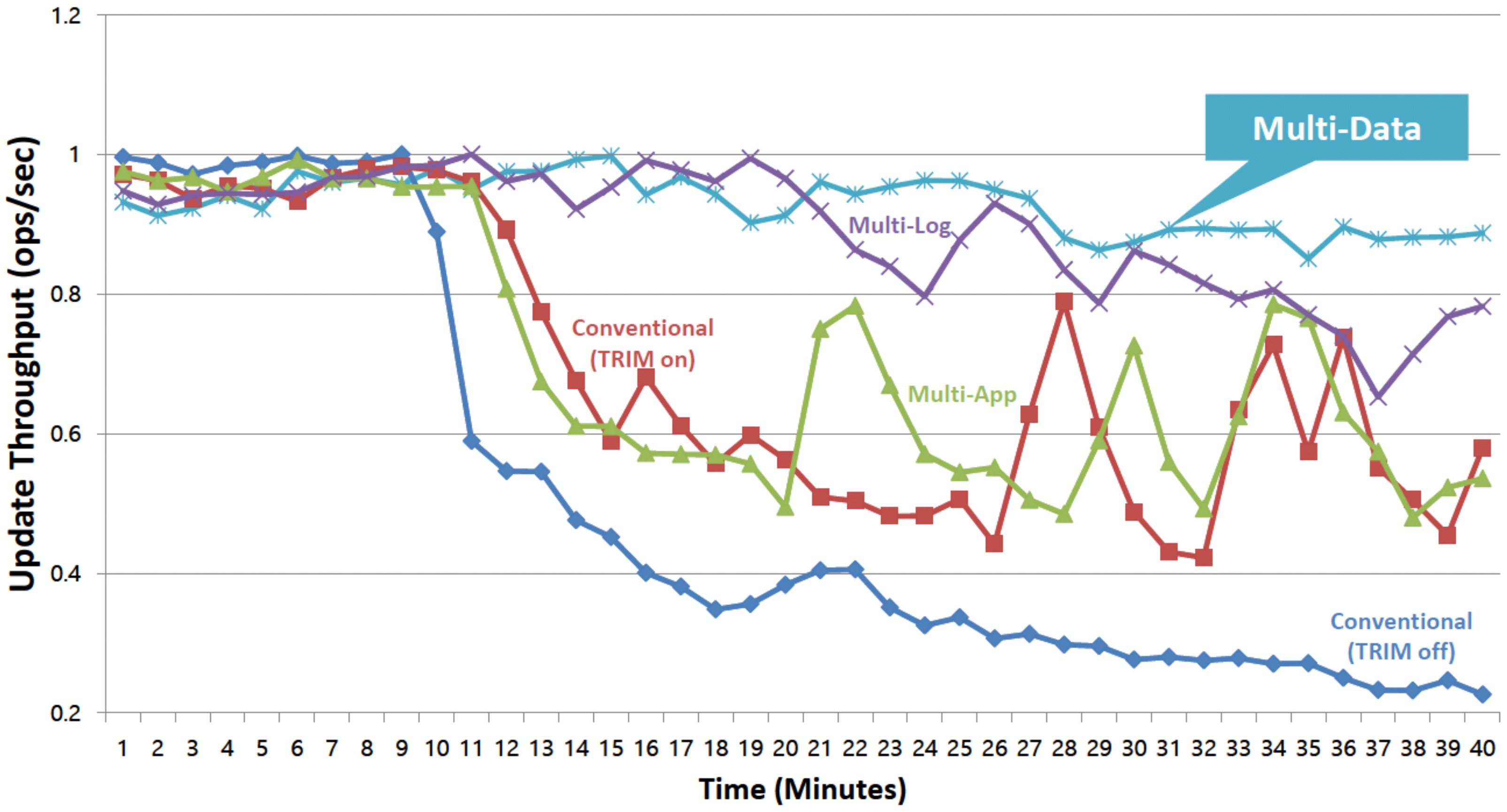
Results: Multi-Log

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



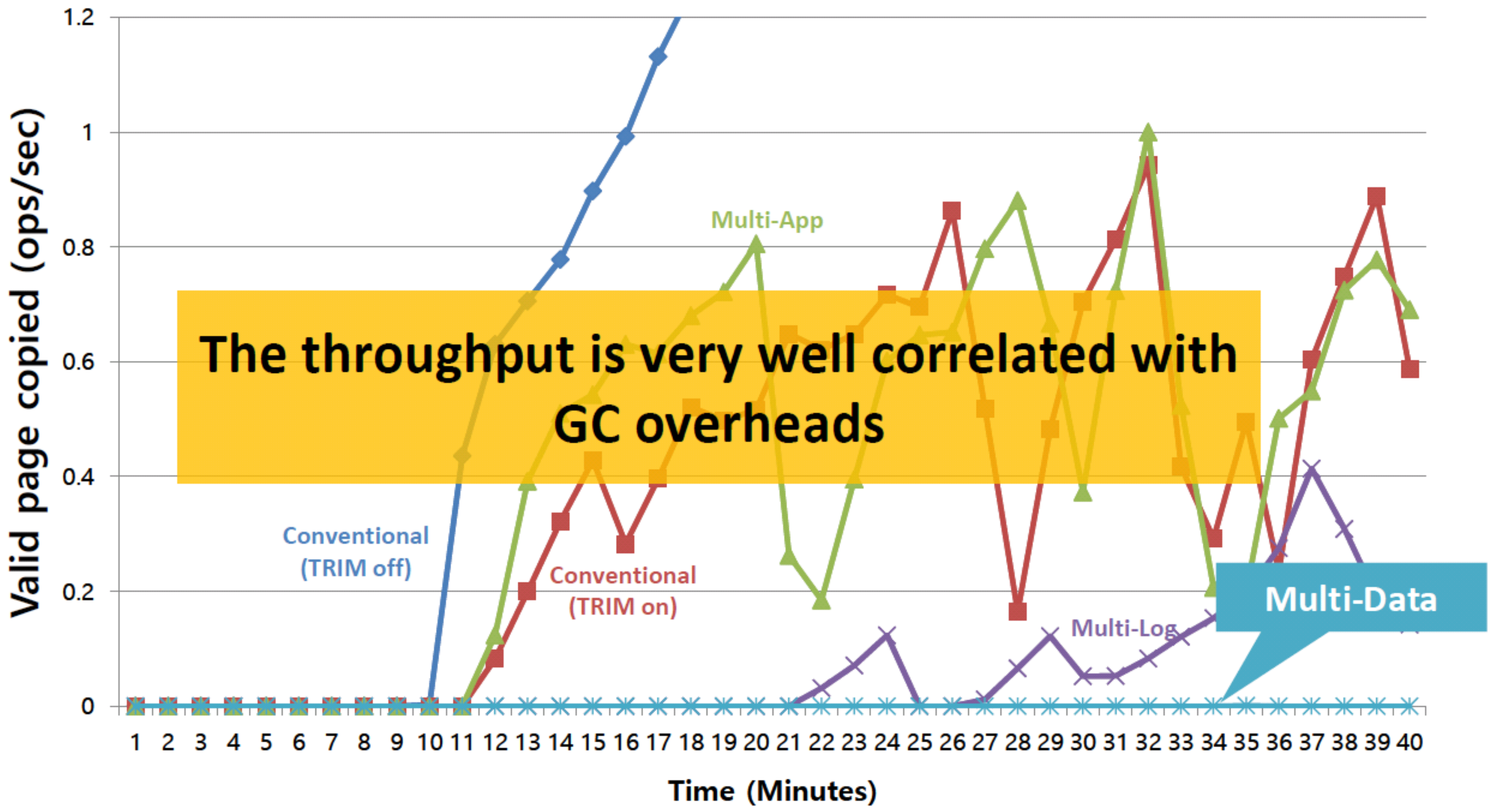
Results: Multi-Data

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



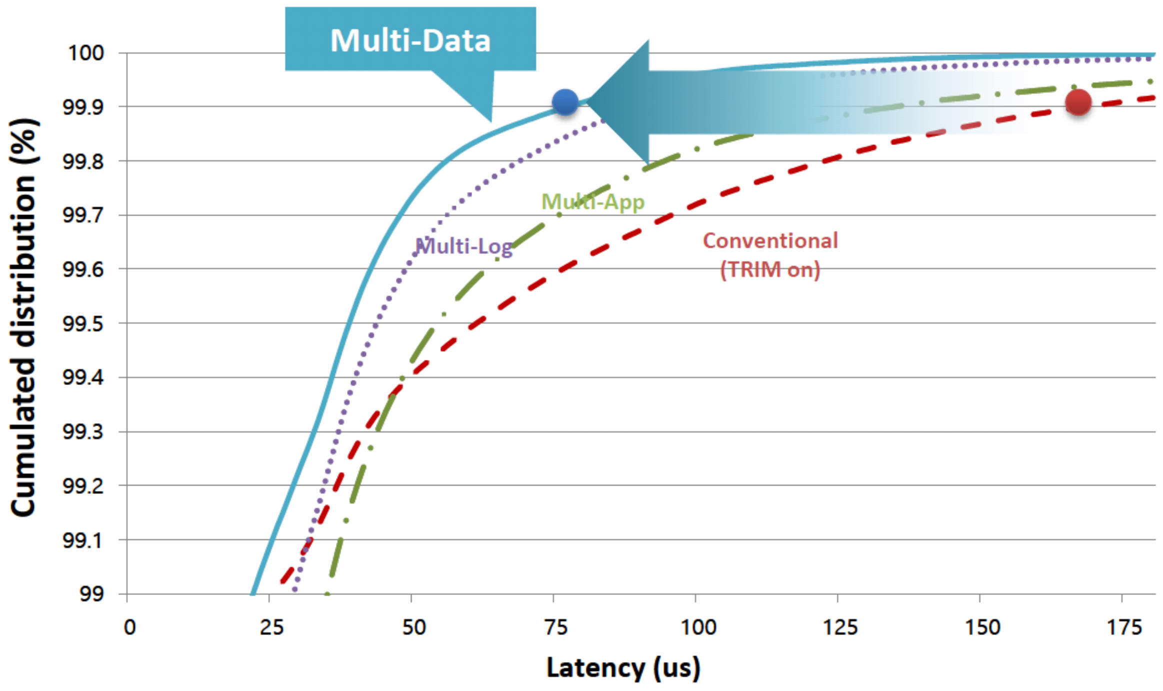
Results: GC Overheads

- 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