清華大學
Tsinghua University

Scalio: Scaling up DPU-based JBOF Key-value Store with NVMe-oF Target Offload

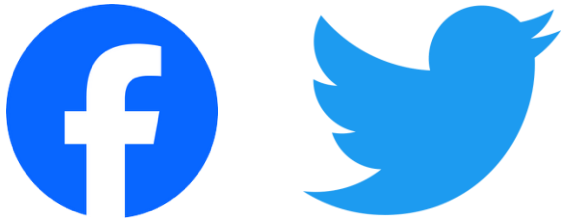
Xun Sun*[1], Mingxing Zhang*[1,2], Yingdi Shan*, Kang Chen*, Jinlei Jiang*, Yongwei Wu*†[2]

**Tsinghua University* †*Quan Cheng Laboratory*

[1] Equal contribution [2] Corresponding authors

July 8

Demand for High Throughput and Large Capacity Key-value Stores

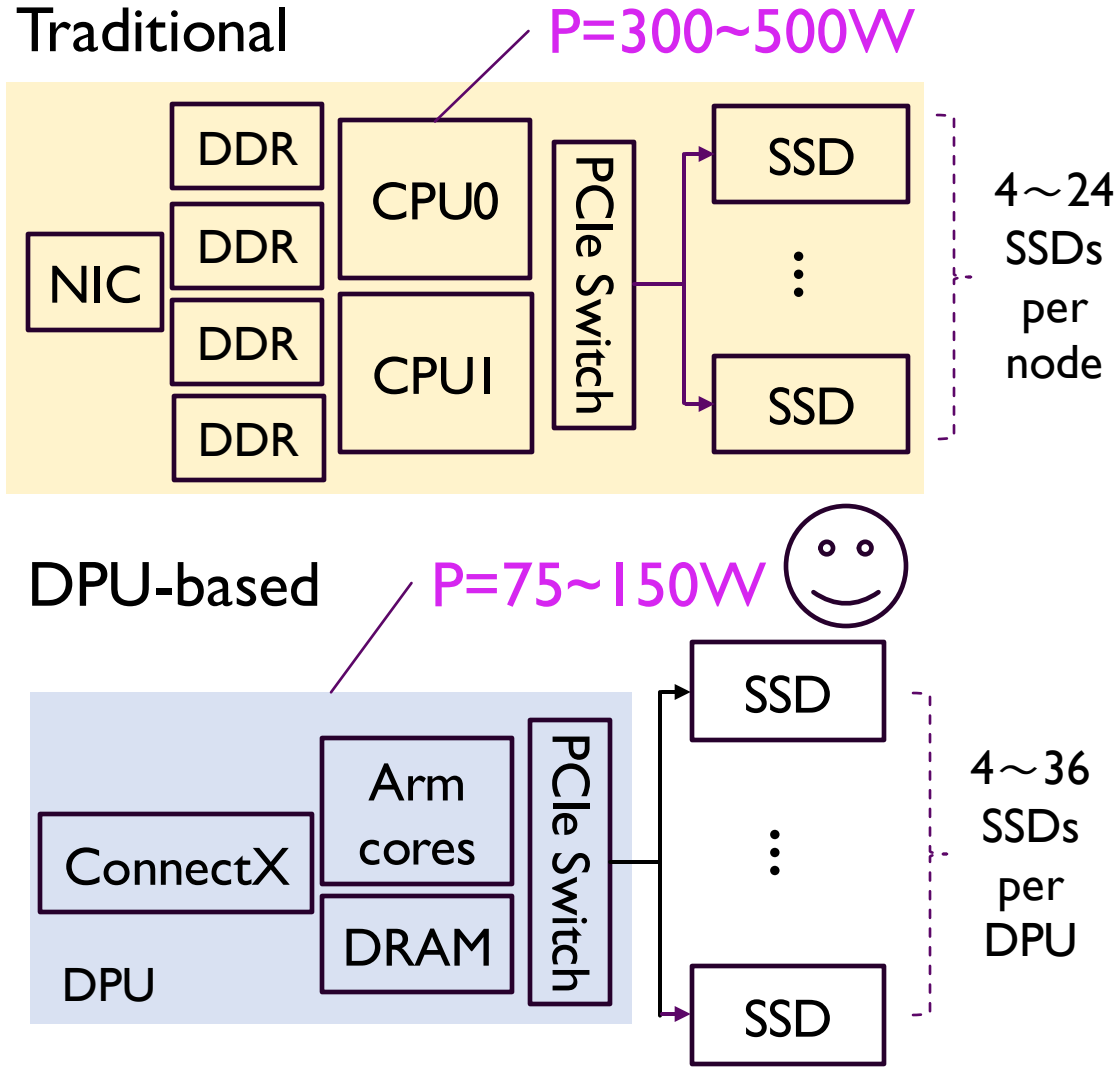


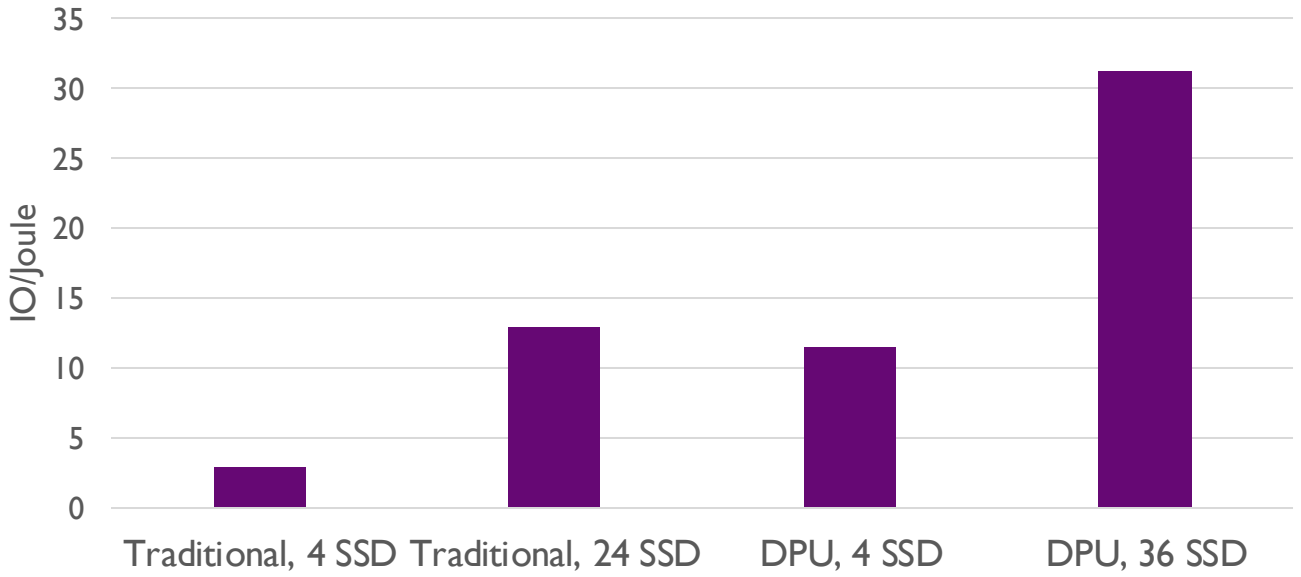
Billions of users' profiles, posts, and likes

Millions of real-time transactions

Vast amount of data

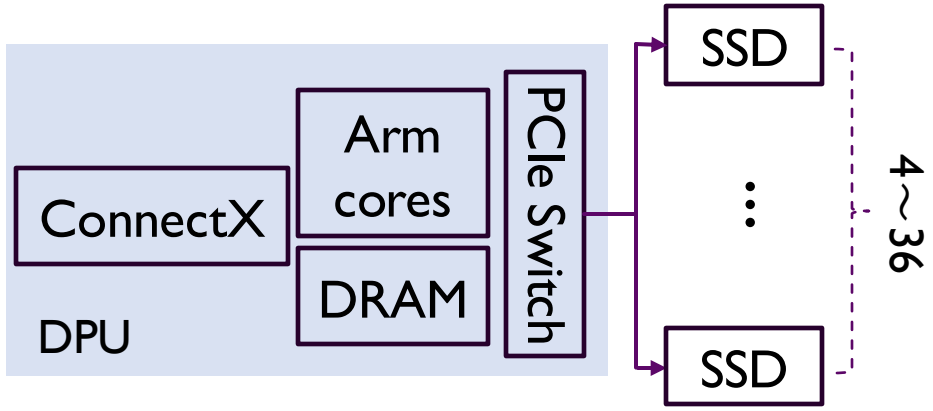
Traditional Storage Servers vs. DPU-based JBOF Storage Servers





DPU-based JBOF storage servers offer high energy efficiency.

Emerging High-density DPU-based JBOF Storage Servers



Supermicro's JBOF platform

NVIDIA BlueField-3 DPU

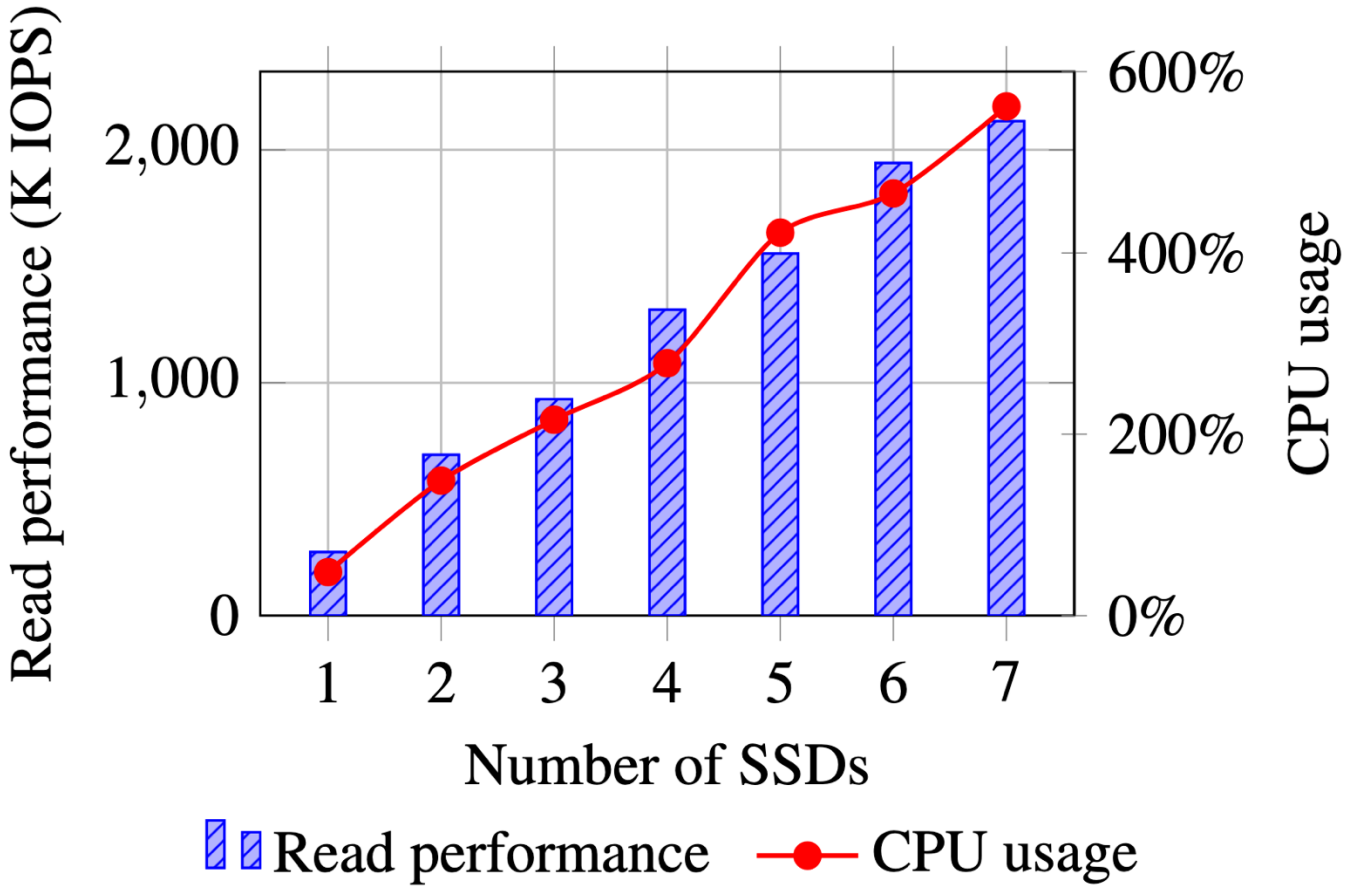
+ 36 NVMe SSDs

AIC's JBOF platform

NVIDIA BlueField-3 DPU

+ 26 NVMe SSDs

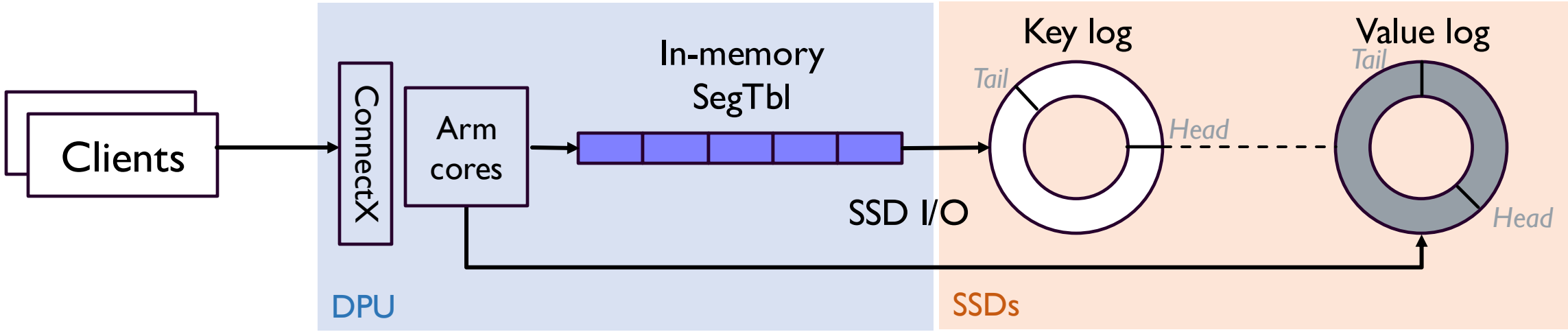
Handling Massive SSD I/O Requires High CPU Usage



FIO test by increasing the number of SSDs.

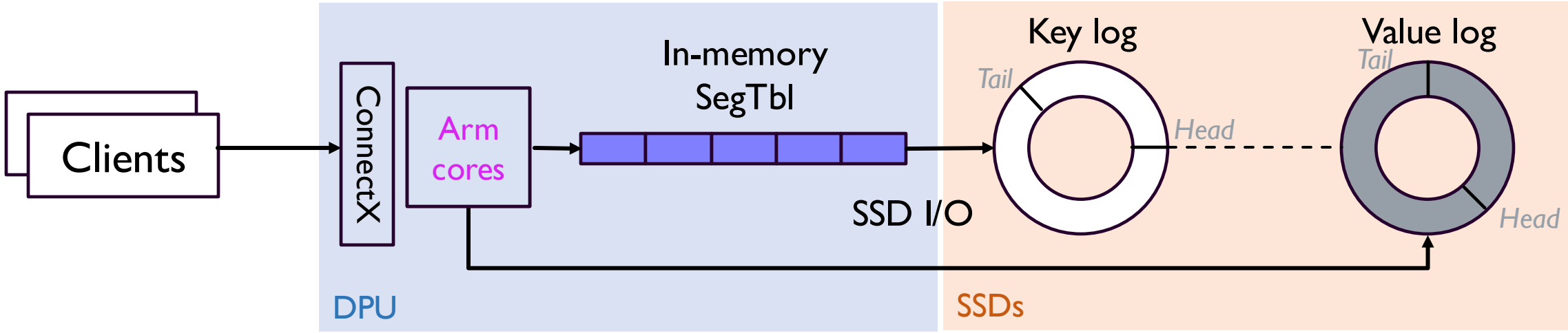
The CPU usage increases as the I/O rate goes up.

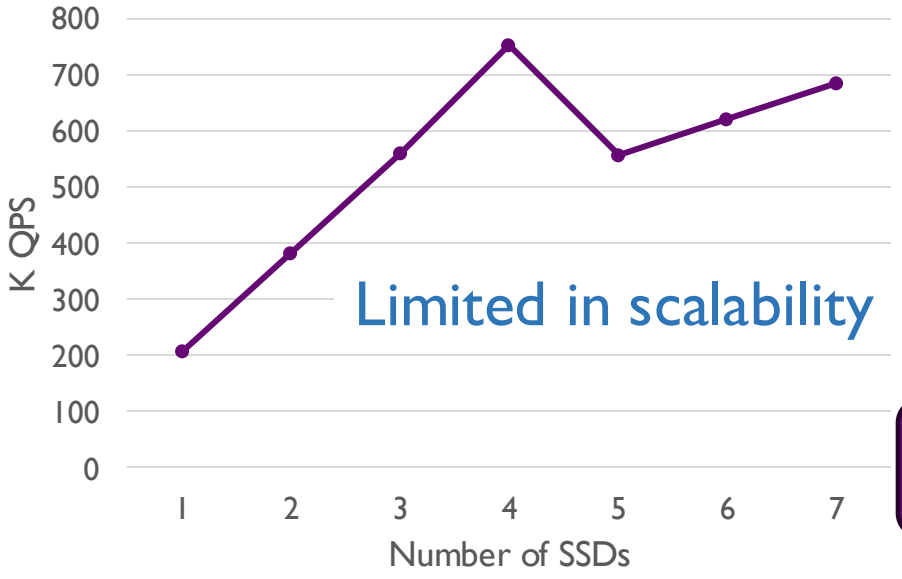
Prior Work: LEED Key-value Store [SIGCOMM '23]



- SOTA k-v store for DPU-based JBOF platforms
- Optimizes workflow to minimize CPU overhead
- Achieves 800 K QPS under a 4-SSD configuration (saturates SSD I/O)

LEED Fails to Scale up



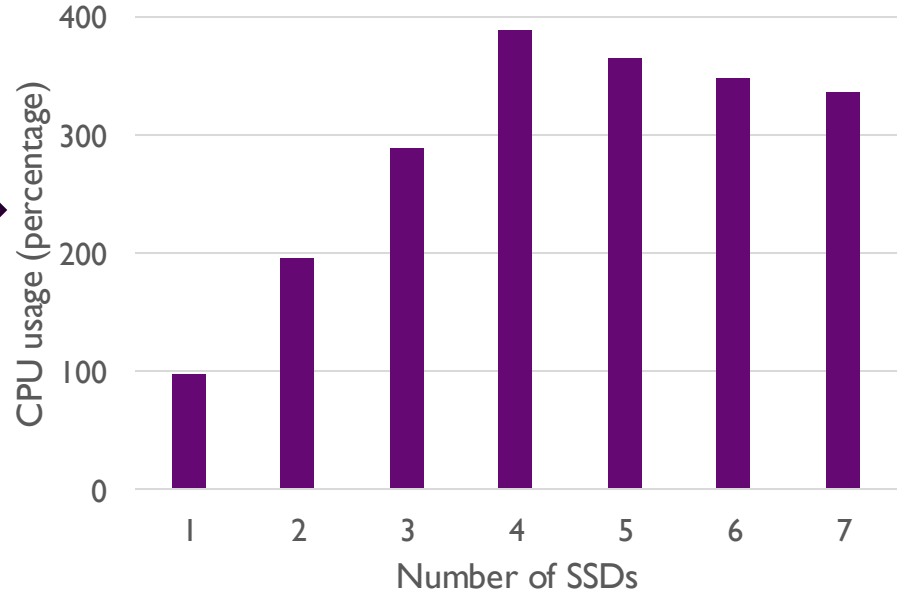


Limited in scalability

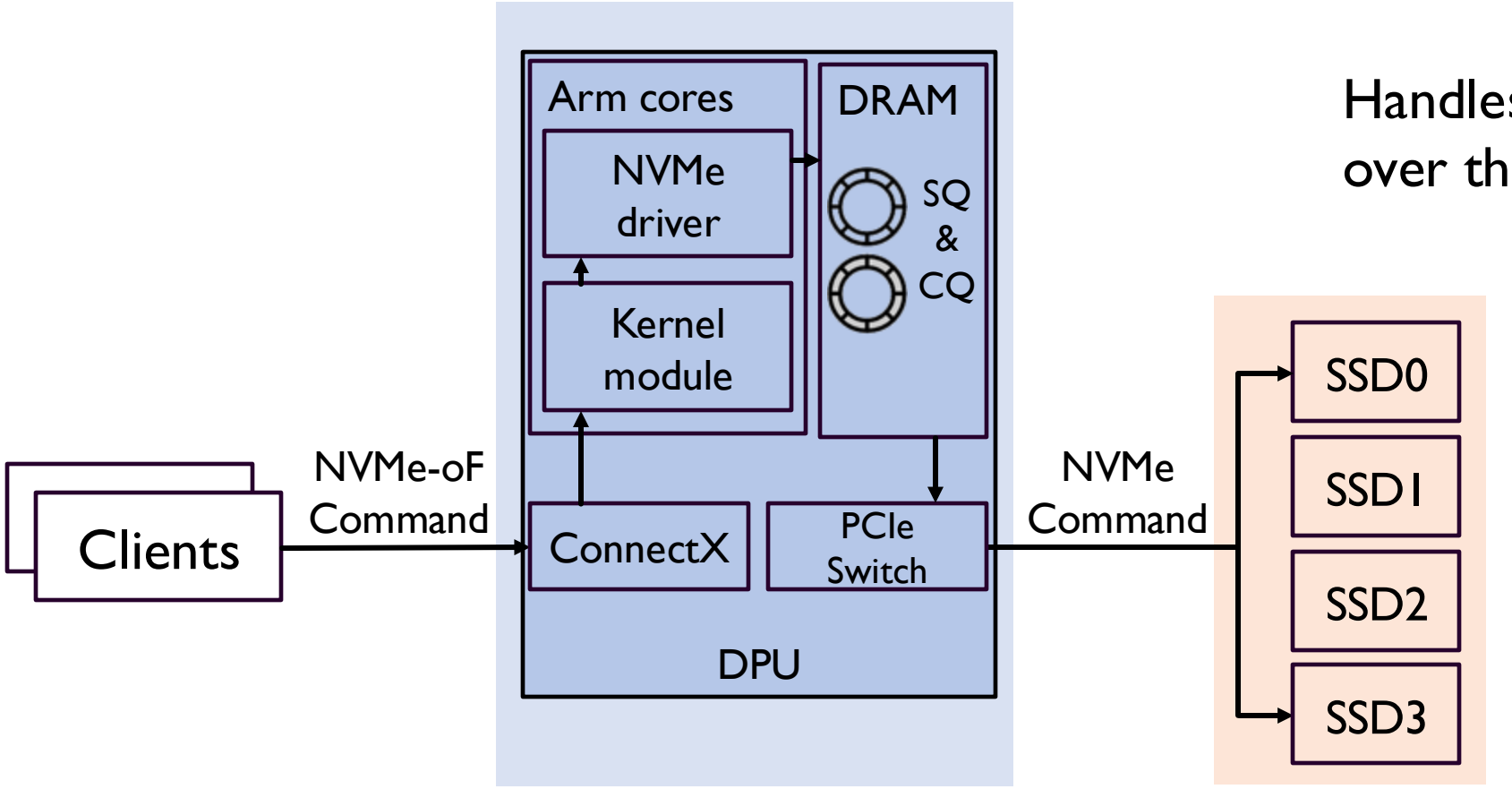
Reason

Overloaded Arm cores!

Motivation: bypass CPU

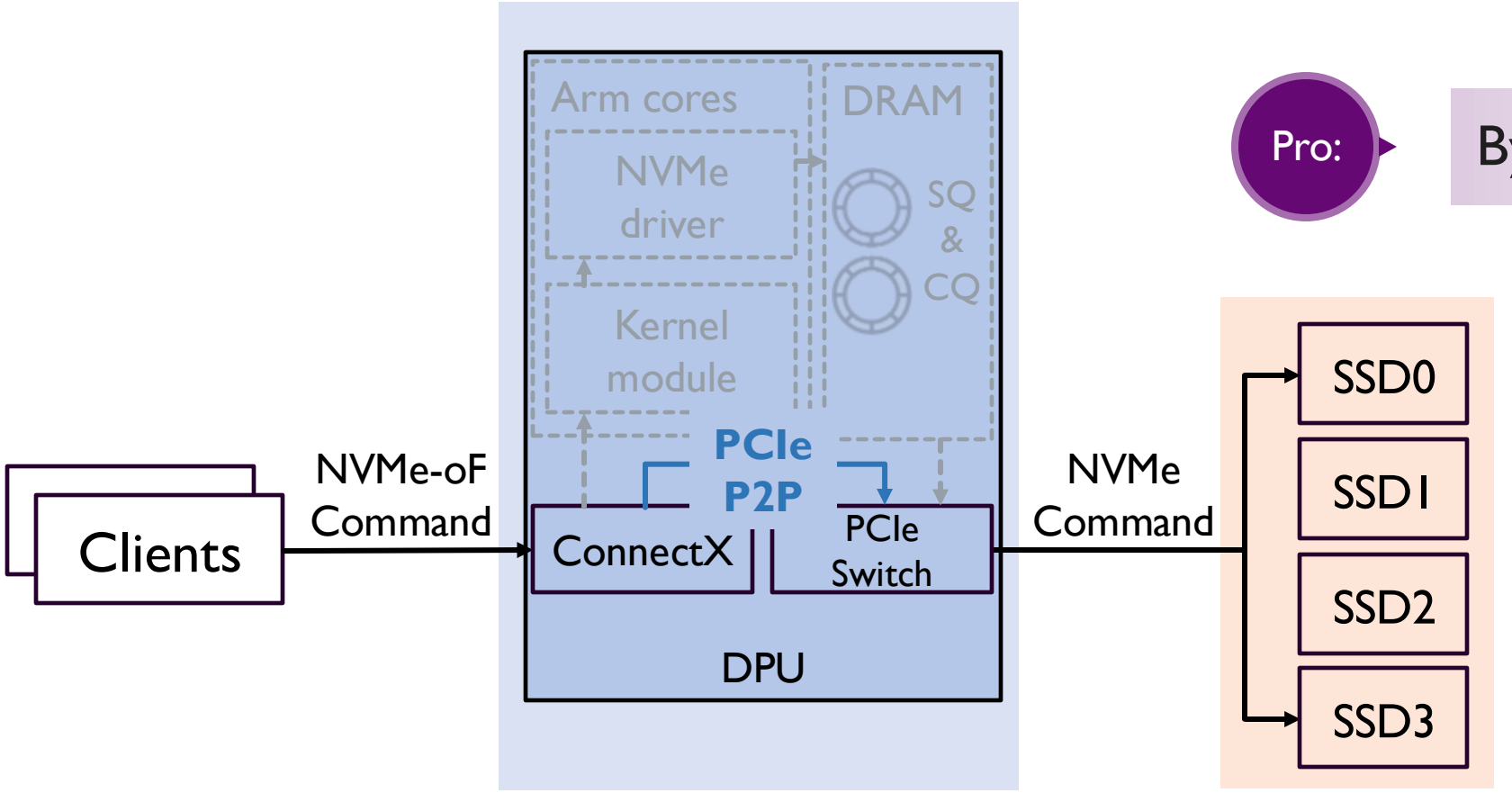


NVMe over Fabrics Target Offload Frees CPU Usage



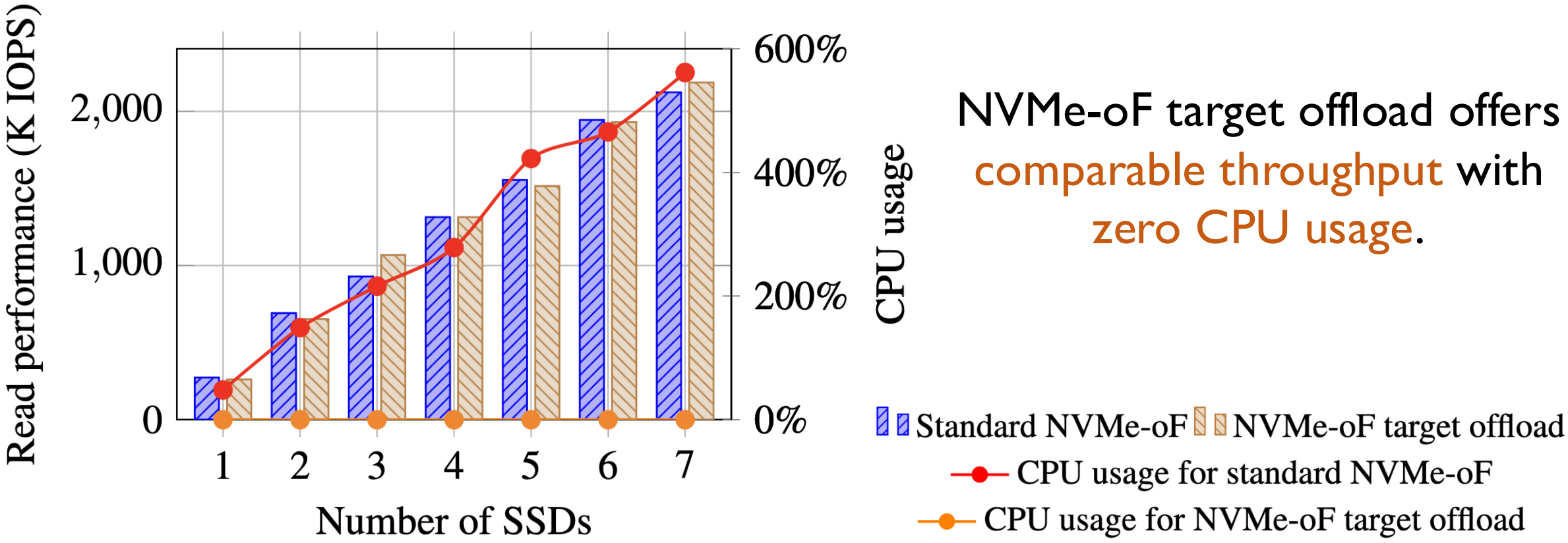
Handles NVMe commands over the network

NVMe over Fabrics Target Offload Frees CPU Usage



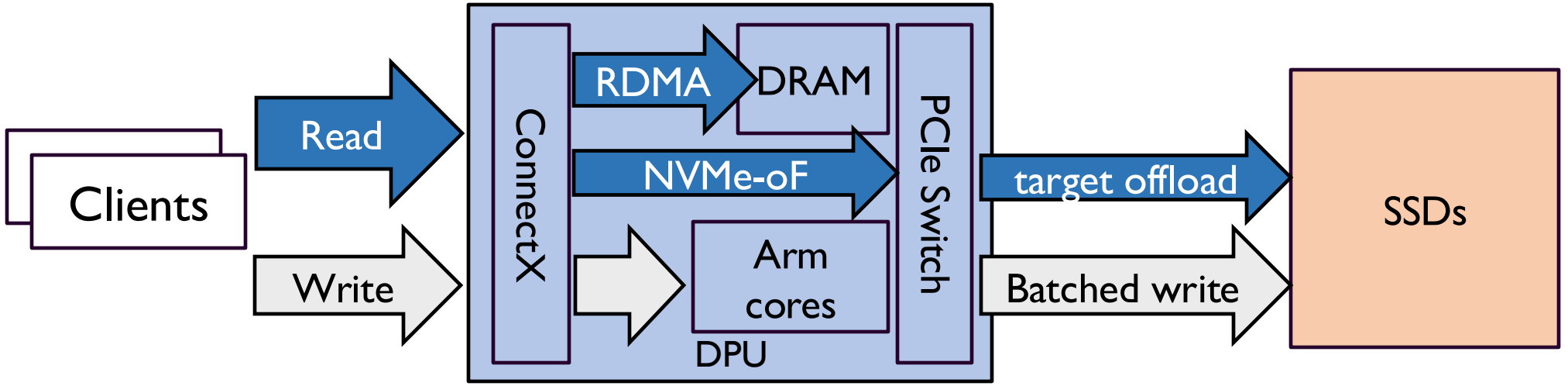
Pro: Bypasses Arm cores

NVMe over Fabrics Target Offload Frees CPU Usage



NVMe-oF target offload offers comparable throughput with zero CPU usage.

Scalio's Design Focuses on Reducing CPU Usage



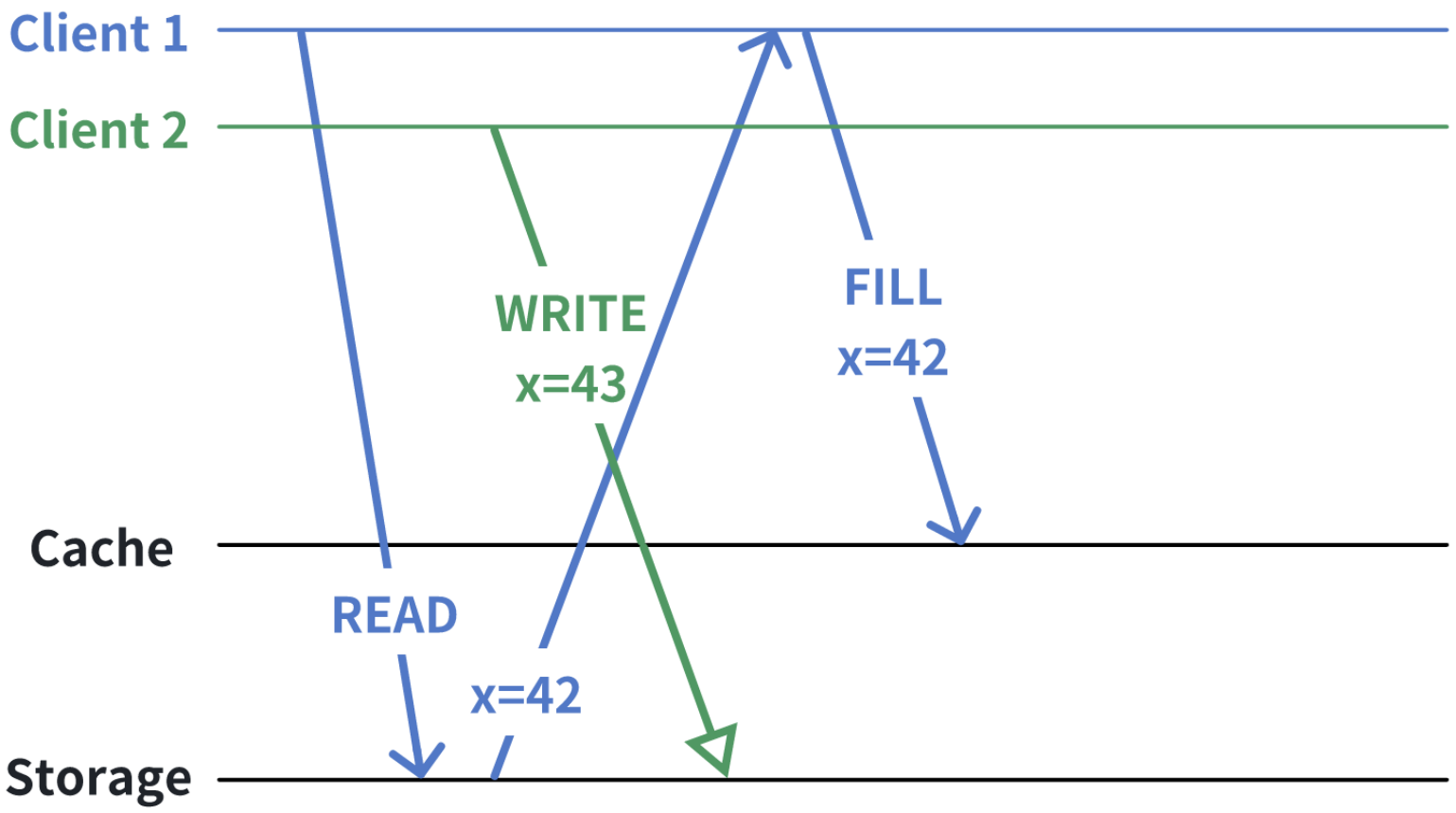
Free DPU's Arm cores with RDMA and NVMe-oF target offload.

Reduce CPU usage with batched write.

Challenge: Ensure Linearizability among Components

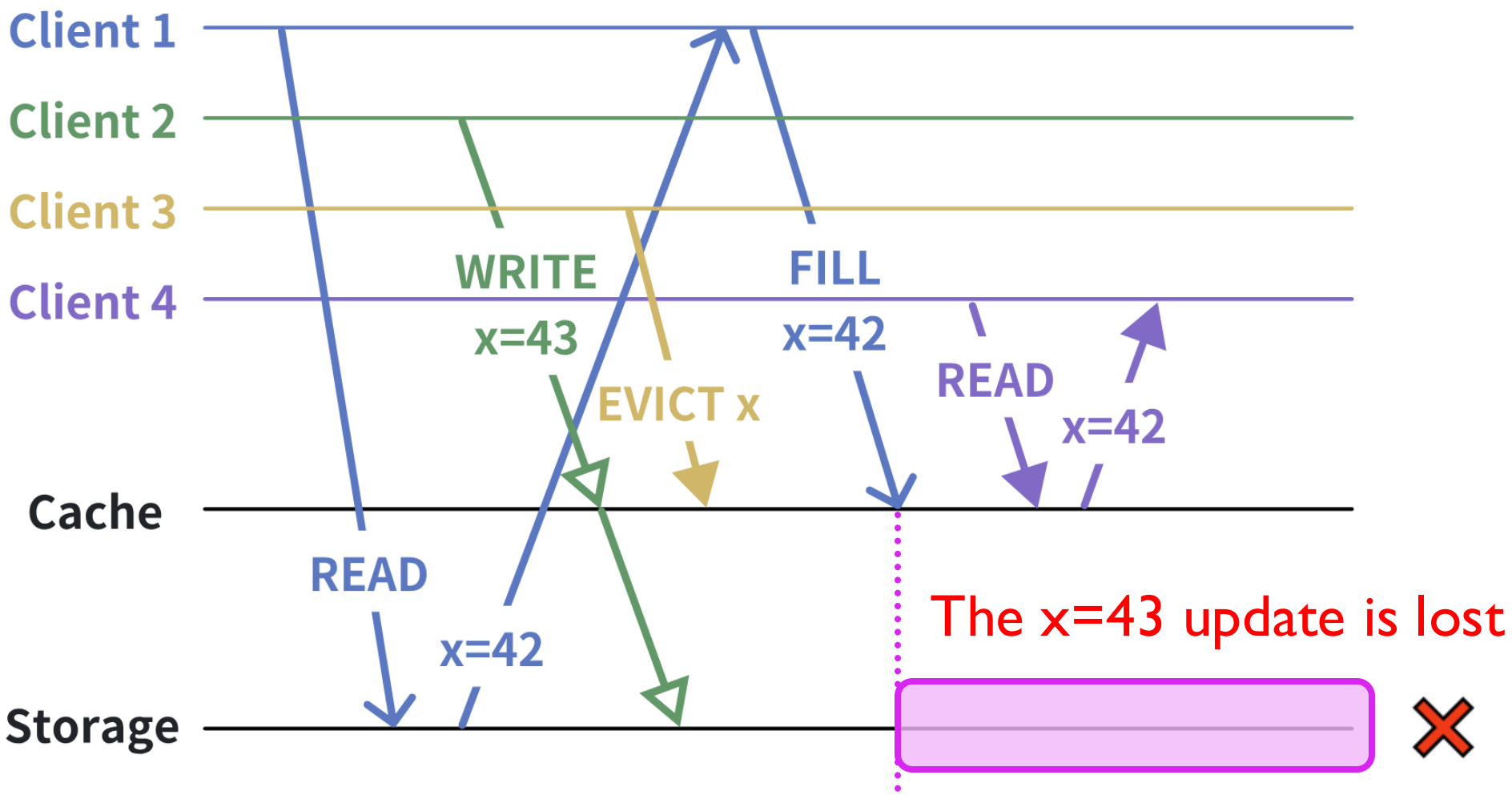
- To reduce server CPU usage, RDMA and NVMe-oF operations are issued on the clients' side independently.
- Linearizability may be violated without careful coordination, especially with the involvement of cache.

Counterexamples: naïve approaches violate linearizability

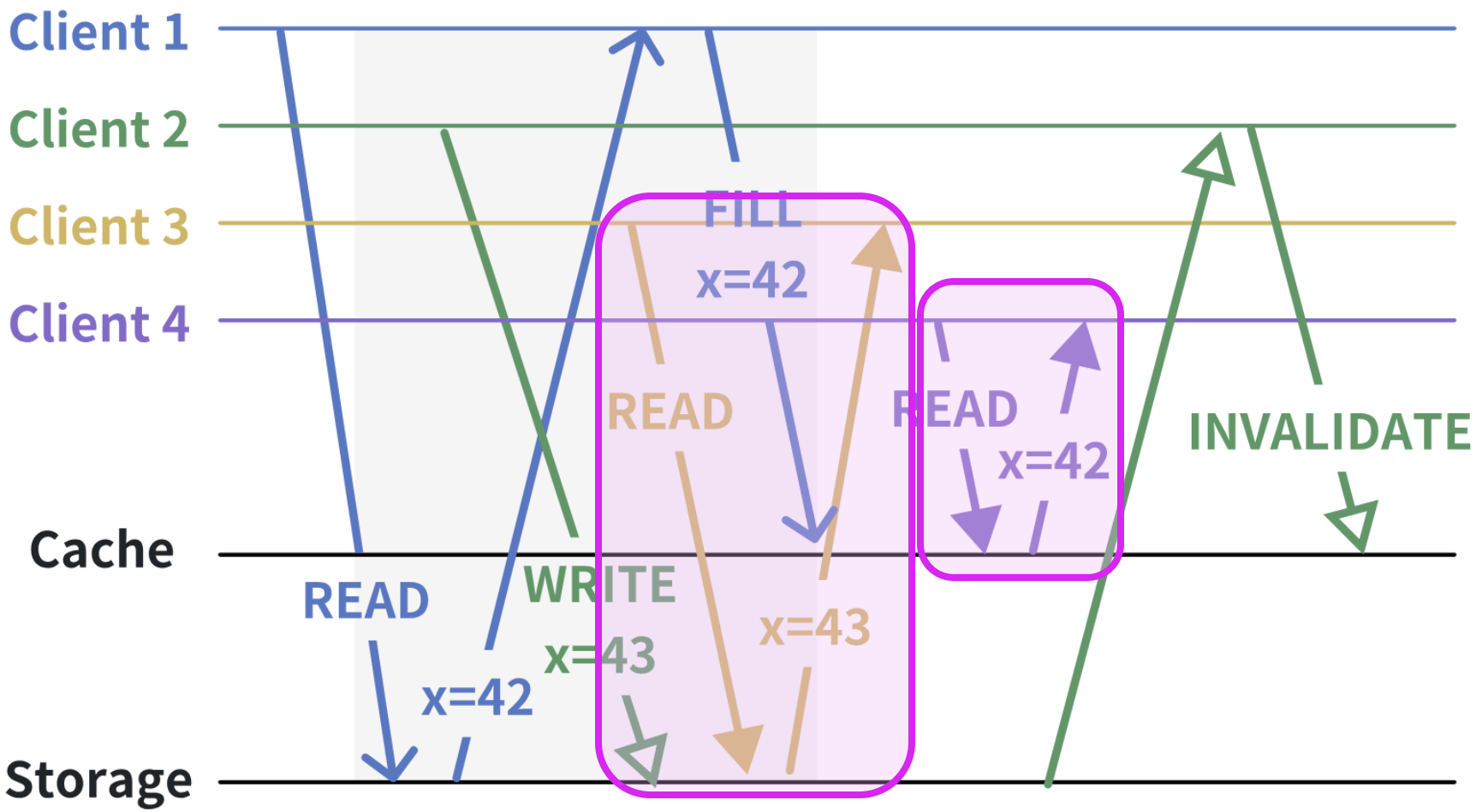


- Consider the case:
- Initially x missing in cache
- Client 1 reads x
- Client 2 updates x
- When to update cache?

Counterexample 1: write-to-cache-first violates linearizability

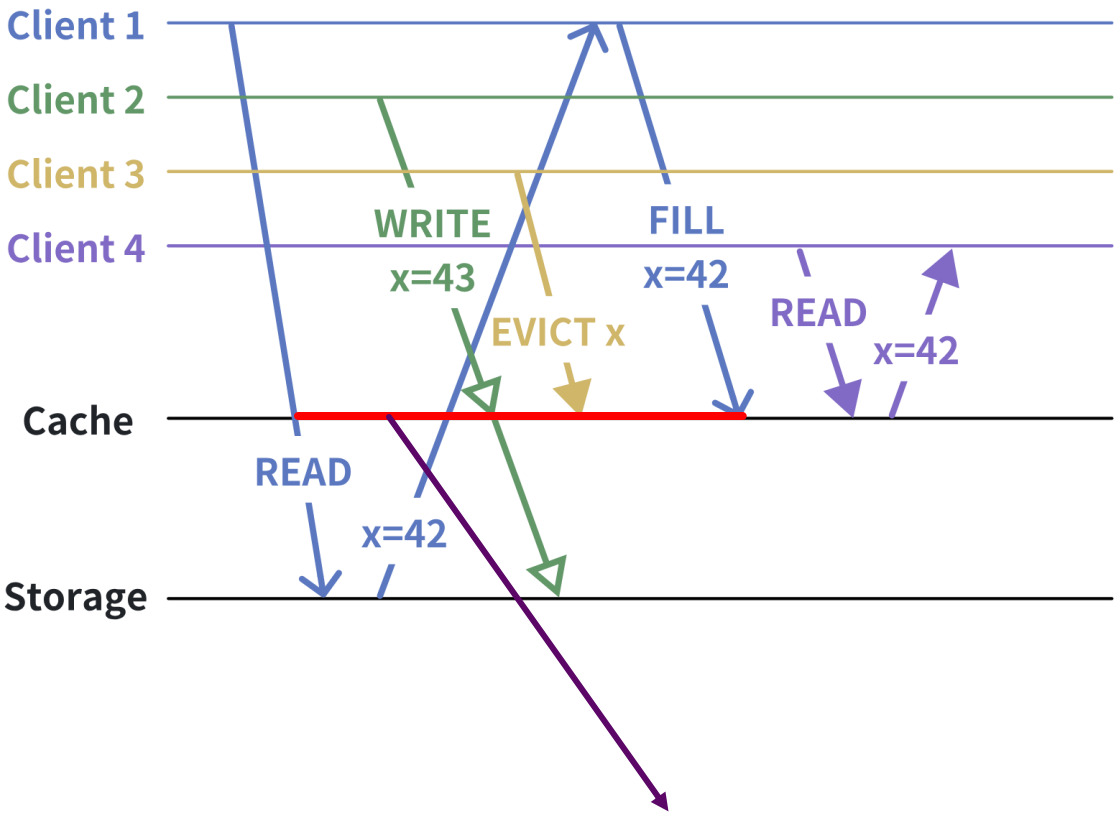


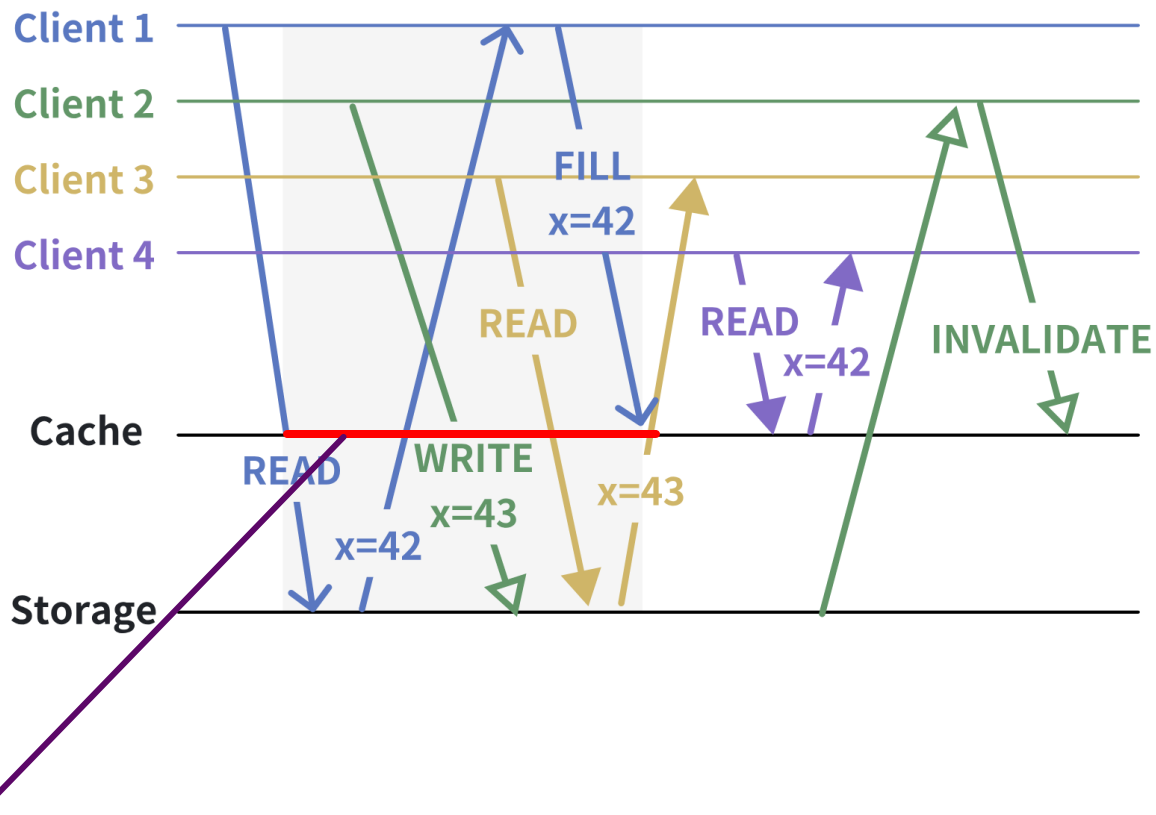
Counterexample 2: write-to-SSD-first violates linearizability



✗ The order of these reads is wrong

Key to Linearizability: DRAM be Aware of the Filling Process



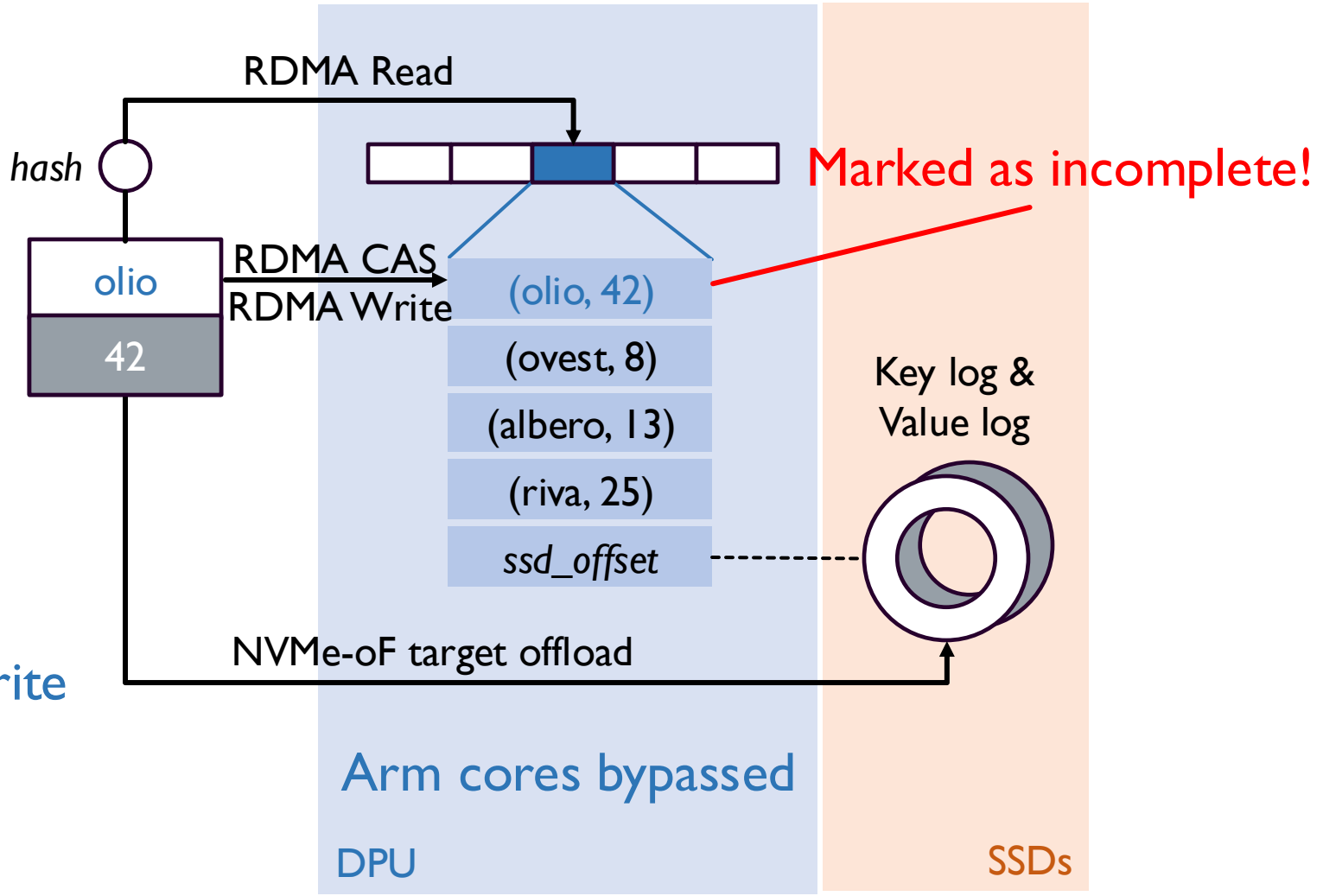


On read miss: record incomplete states in the DRAM

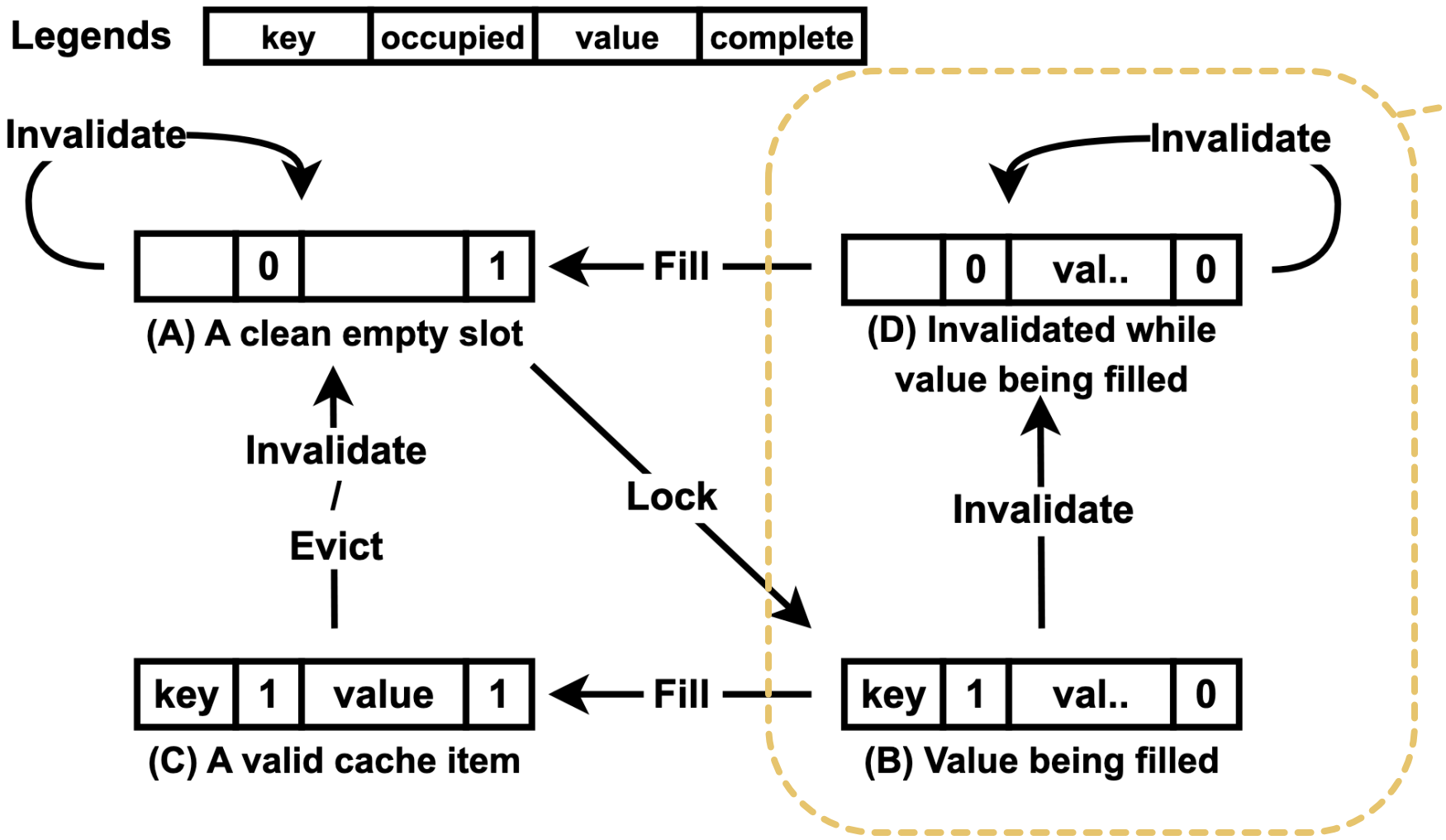
Scalio's Data Flow on Cache Miss

1. Fetch a block with RDMA Read
2. No matching slot; pick a victim
3. Lock the slot with RDMA CAS
4. Read on-disk data with NVMe-oF target offload
5. Fill back the cache with RDMA Write

Key: mark as incomplete and then fill.



Scalio incorporates four states to ensure linearizability



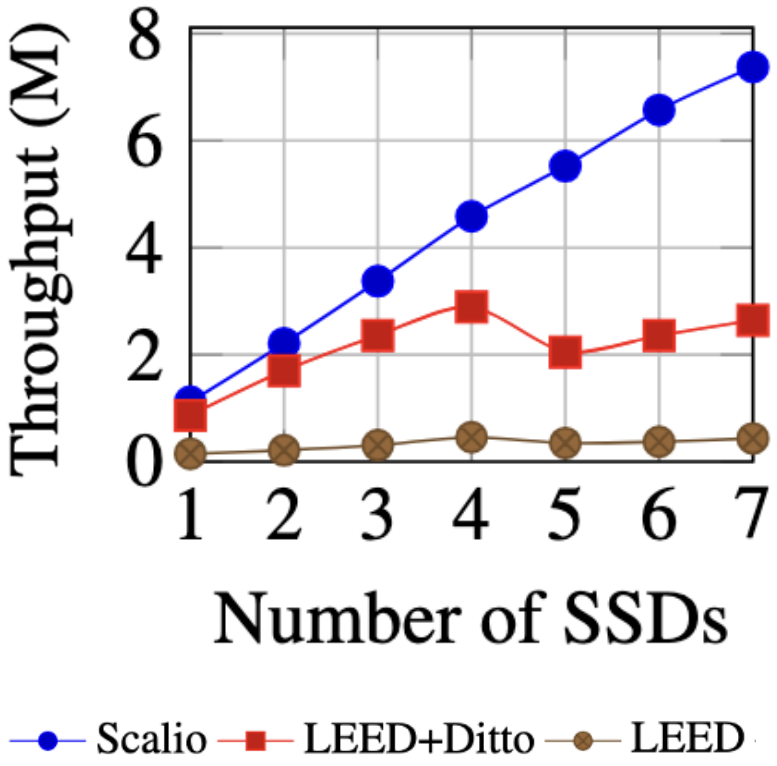
Incomplete states: Other readers should wait until filling is done.

Ensures linearizability without involvement of the server CPU.

Evaluation: Setup

- Up to **5** client nodes (up to **160** client threads)
- 1 server node & **7** Samsung 970 PRO SSDs (**500K** IOPS each)
- Limited to **8** CPU cores and **8** GB DRAM
- Baselines: LEED (SSD k-v store), LEED + Ditto (with DRAM k-v cache)
- YCSB workloads **A, B, C, D, F** with 16 B keys and 64 B values

Evaluation: Performance Improvements (YCSB Workload C)



Scalio scales up best!

Evaluation: Performance Improvements (All Workloads)

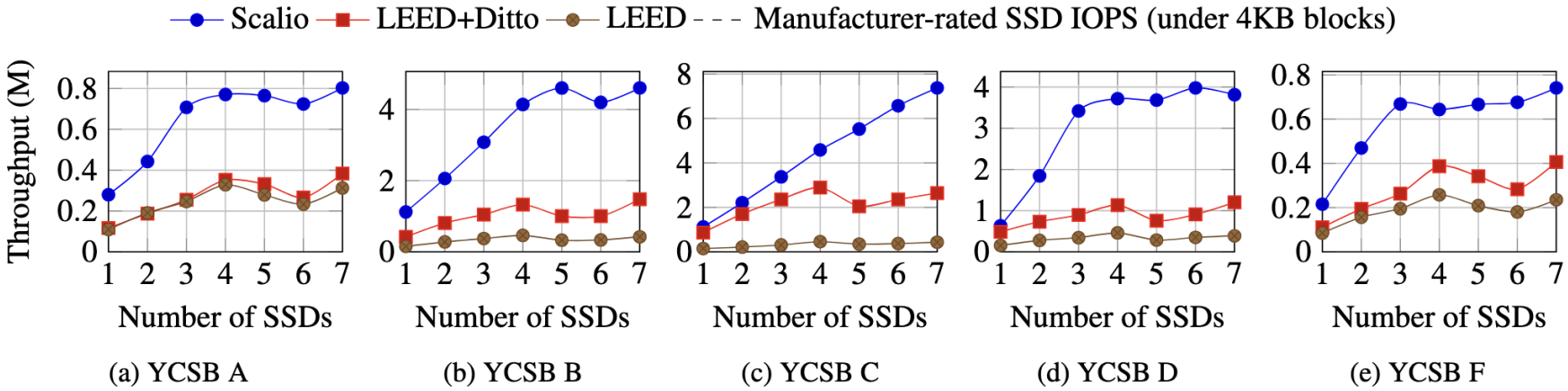


Figure 10: Throughput for YCSB workloads A,B,C,D and F as the number of SSDs scale up.

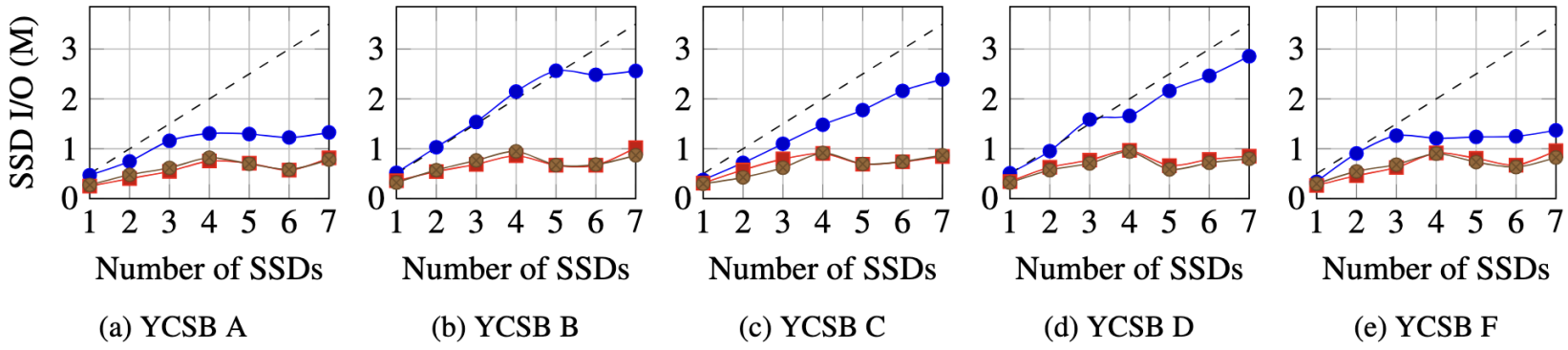
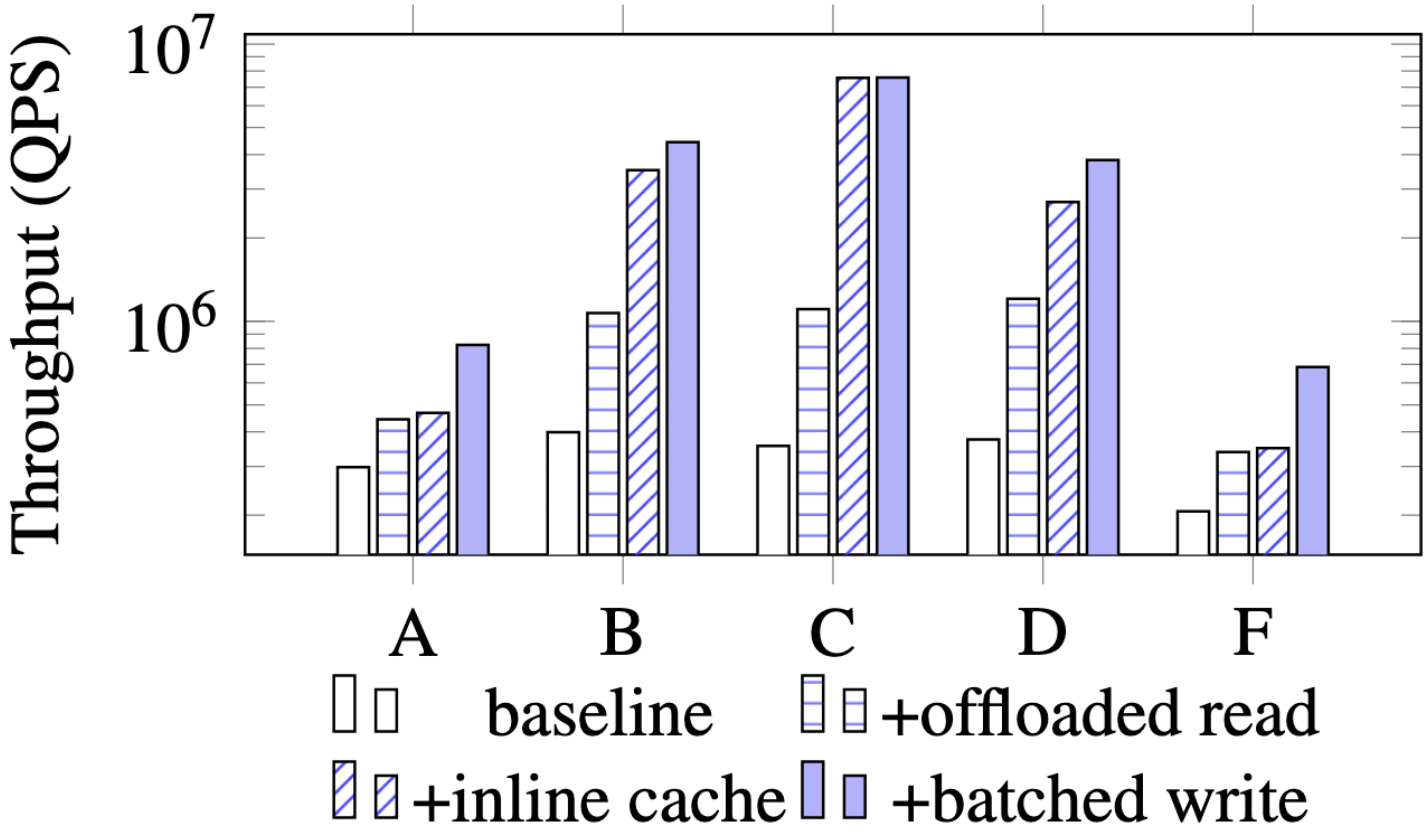


Figure 11: SSD I/O usage under YCSB workloads A,B,C,D and F as the number of SSDs scale up.

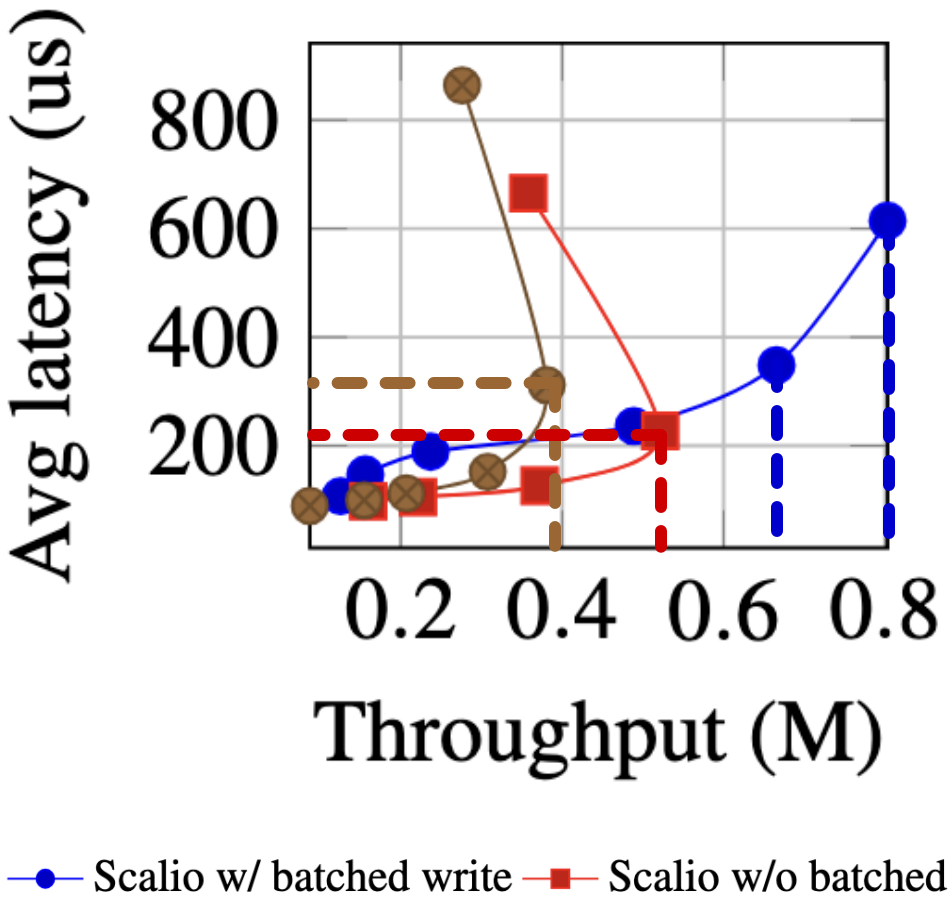
Scalio scales up best and utilizes SSD IOPS best

Evaluation: Improvements Breakdown



All optimizations contribute to Scalio's improvement

Evaluation: Latency (YCSB Workload A)



Scalio w/o batched write: high throughput & low latency

Scalio w/ batched write: even higher throughput

Tune the config according to the goal.

Evaluation: Latency (All Workloads)

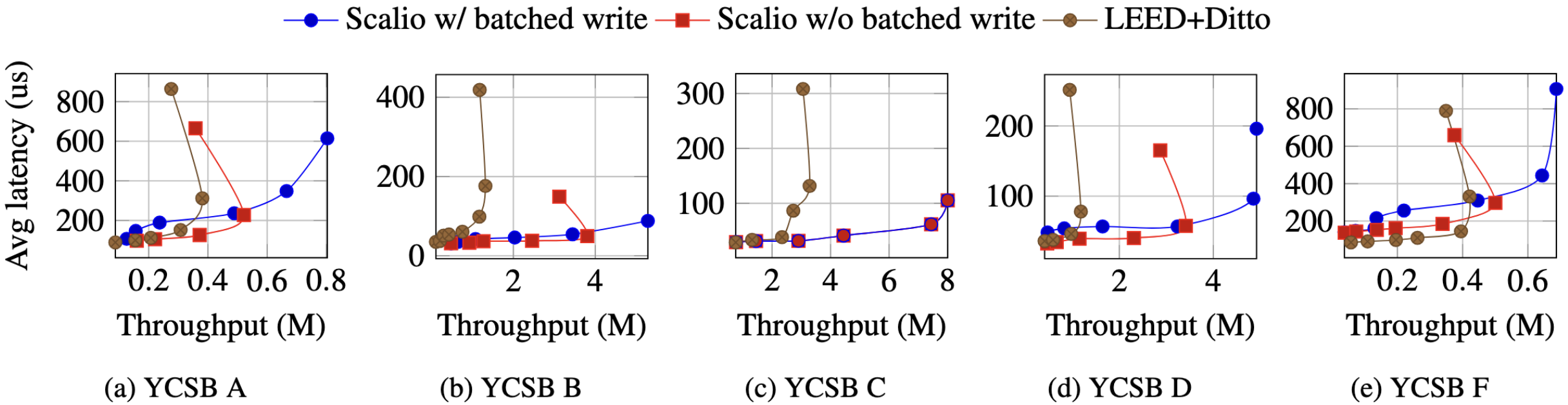
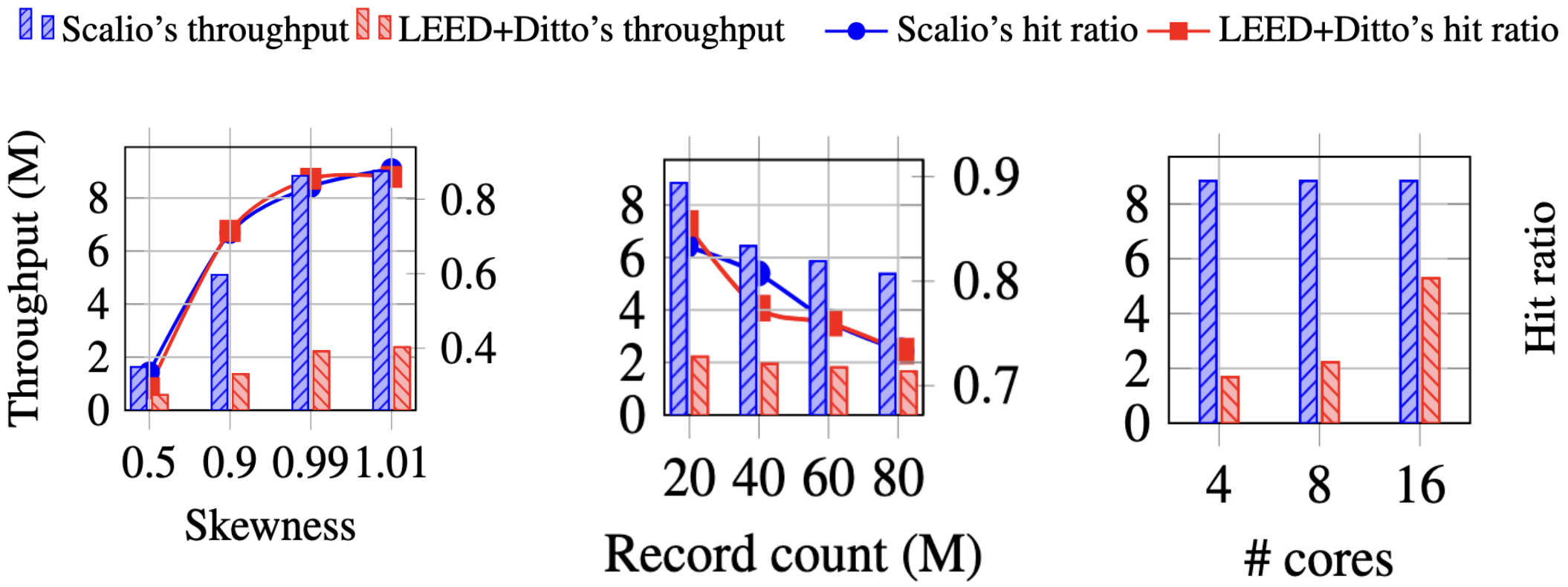


Figure 13: Latency to throughput by varying the number of concurrent I/O queues.

Evaluation: Sensitivity Analysis (YCSB Workload C)



Scalio outperforms the baseline under all configurations.

Thank you for your listening!

https://github.com/madsys-dev/scalio-osdi25-ae