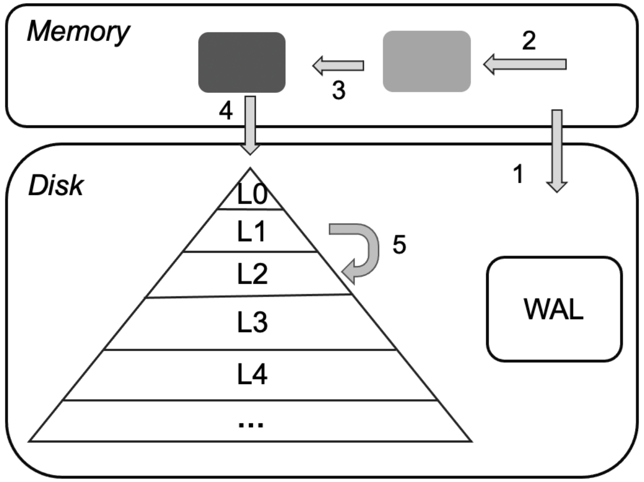
SpanDB: A Fast, Cost-Effective LSM-tree Based KV Store on Hybrid Storage

Hao Chen[1, 2], Chaoyi Ruan[1], Cheng Li[1], Xiaosong Ma[2], Yinlong Xu[1, 3]

[1] University of Science and Technology of China
[2] Qatar Computing Research Institute, HBKU
[3]Anhui Province Key Laboratory of High Performance Computing

Rise of Key-Value Stores

- Persistent key-value (KV) stores popular and important
  - Storing semi-structured data for enterprise services
    - E.g., LevelDB by Google, **RocksDB** by Facebook
  - Being backend storage engine for
    - Ceph, MyRocks, TiDB, Cassandra
  - **LSM-tree** based KV stores are popular

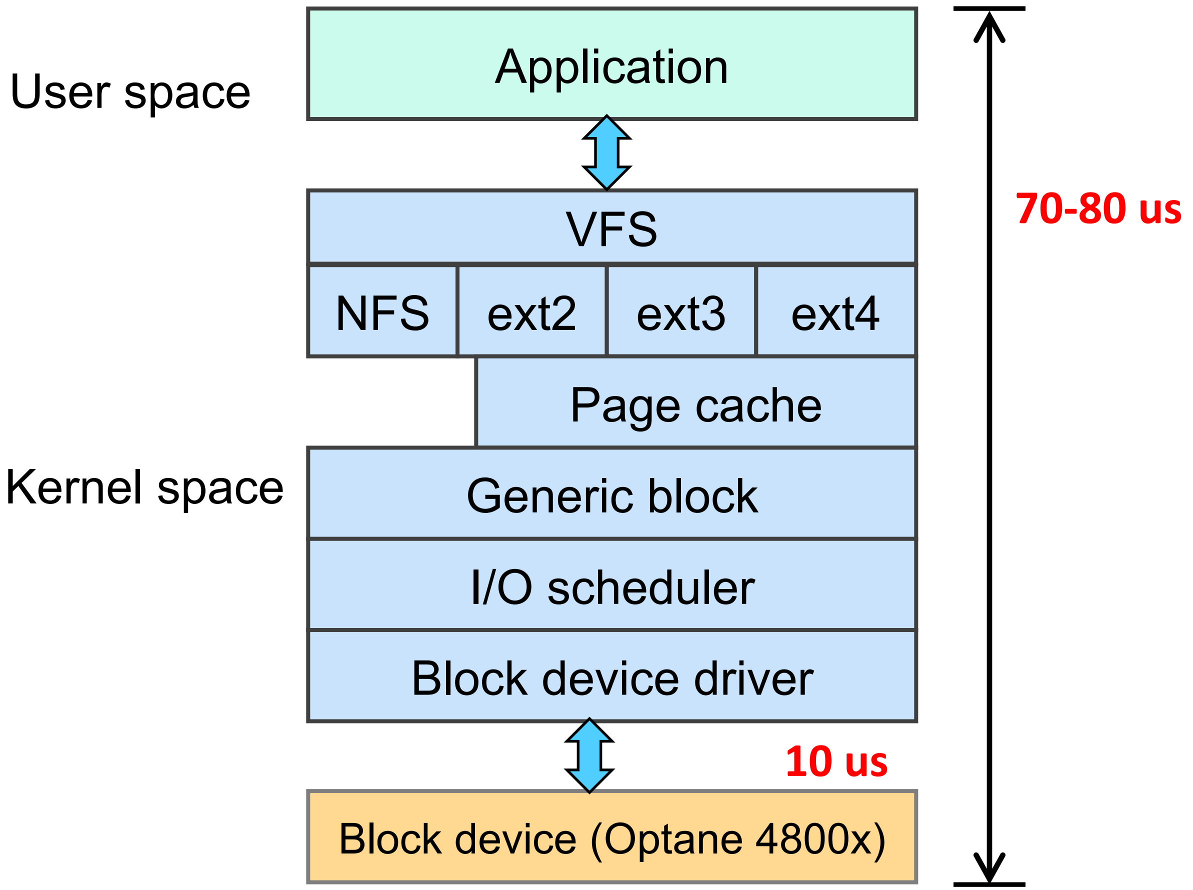


- Opportunities for performance enhancement brought by high-end **NVMe** storage devices

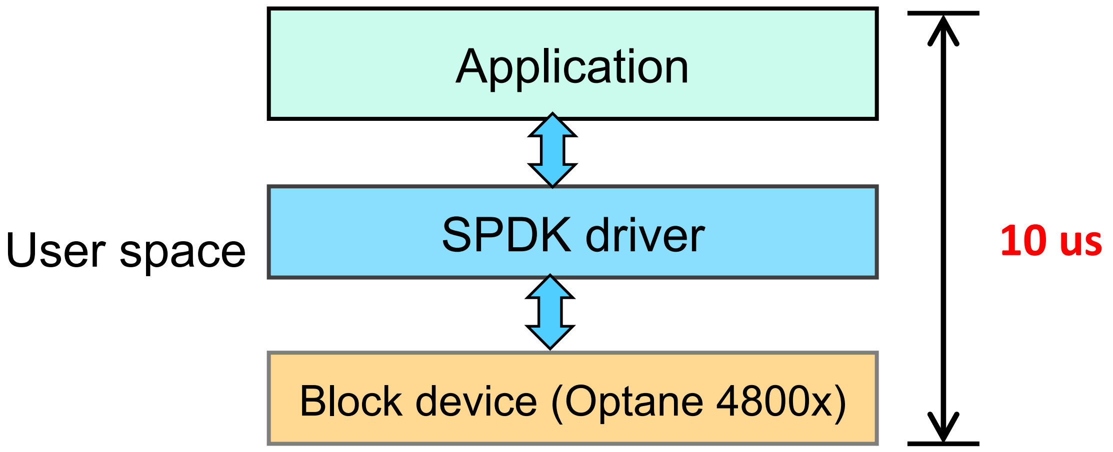
HDD (~ms, 100MB/s) → High-end SSD (~10us, 2000MB/s)

- **Unfortunately, their potential not fully exploited by modern LSM-tree based KV stores**

Challenge 1: Fast Accesses to Fast Devices



Linux I/O Stack

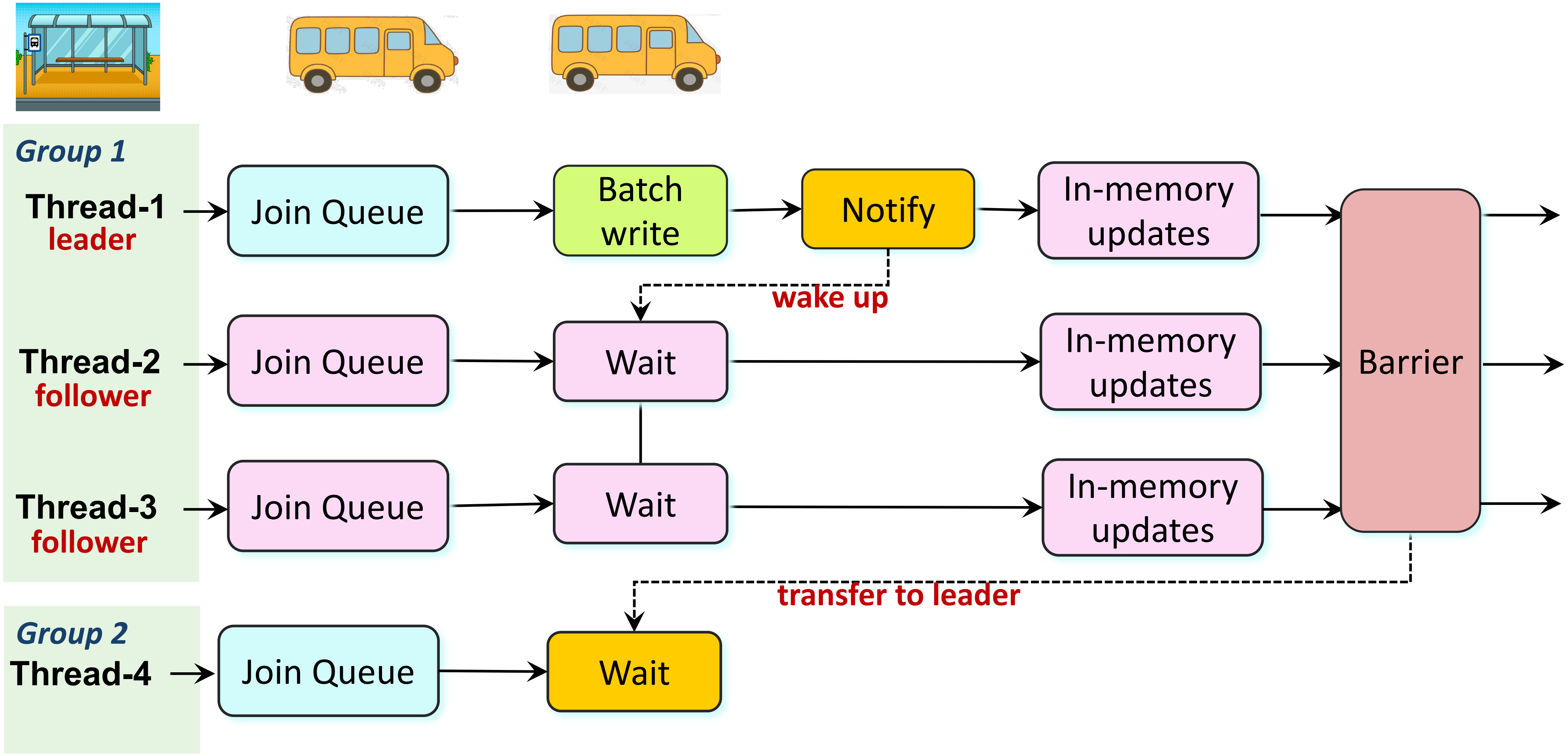


SPDK driver

- Lower latency stemming from
  - User-space driver, avoid syscall
  - Polling for completion rather than interruption
- However, benefits come at costs
  - Need to manage raw space with **no FS support**
  - Busy wait wastes CPU cycles

Challenge 2: Thread Sync Overhead in Group Logging

- ❑ **Group logging**: widely used to speed up write-ahead logging (WAL)
- ❑ All existing group logging implementations **sequential**

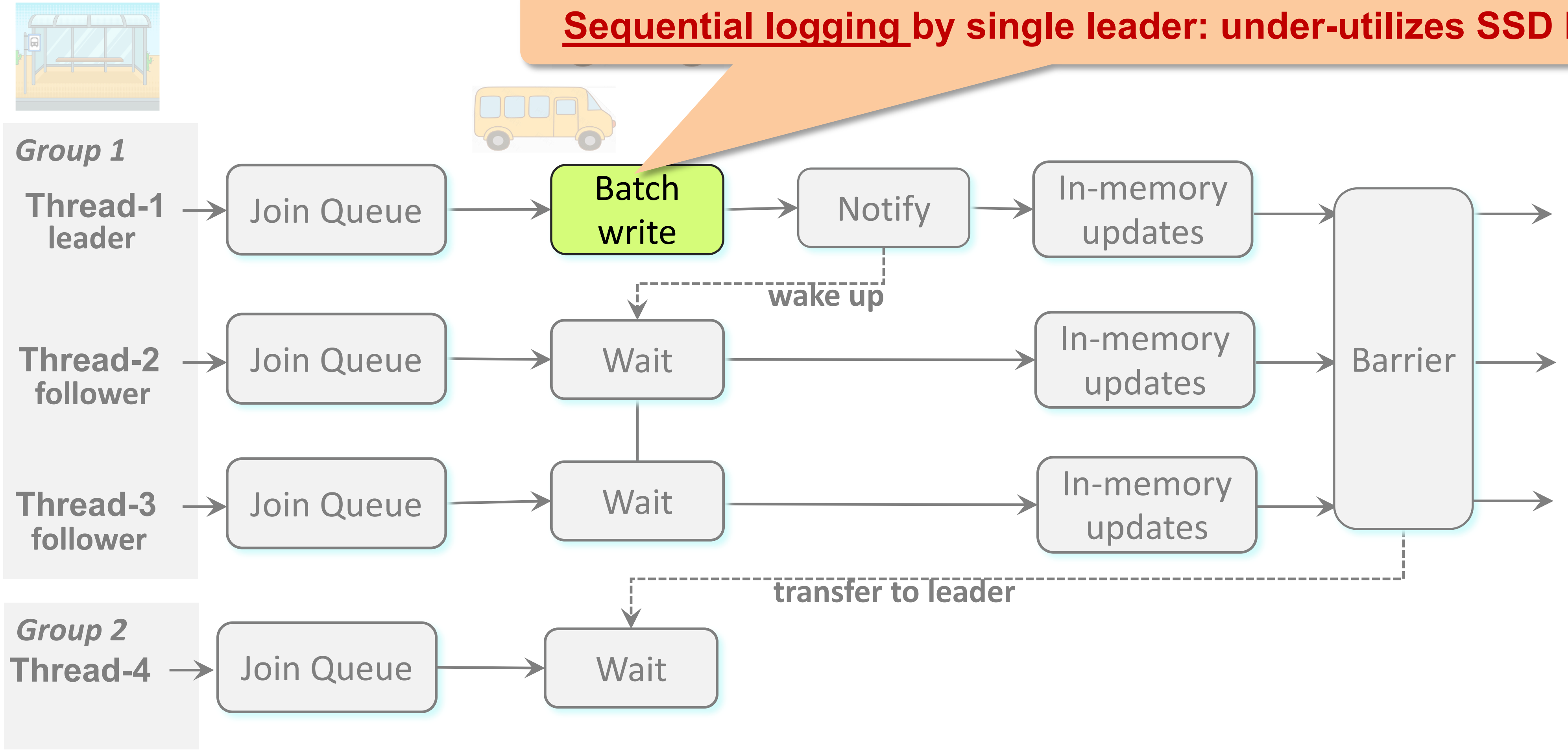


RocksDB/LevelDB group logging process

Challenge 2: Thread Sync Overhead in Group Logging

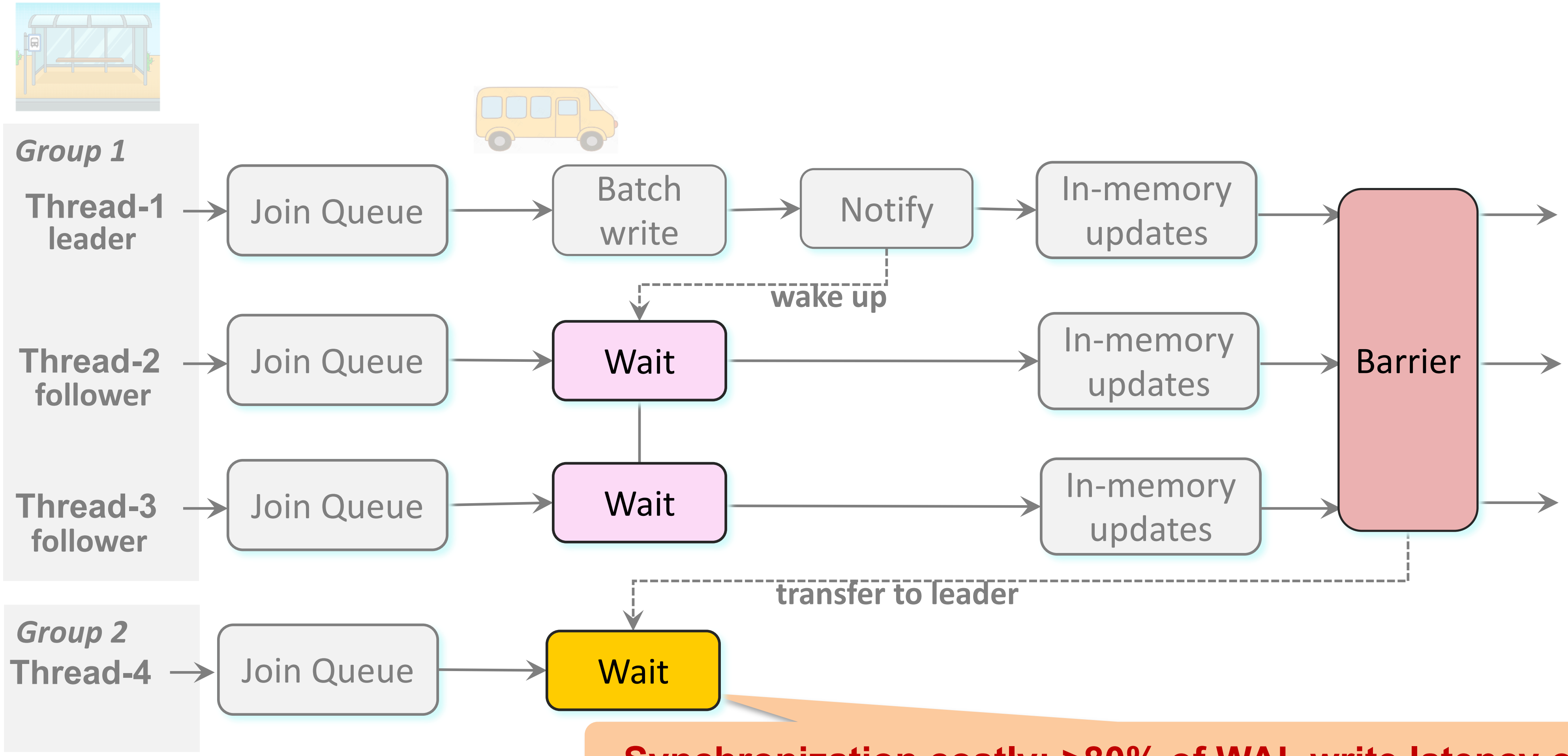
- ❑ **Group logging**: widely used to speed up write-ahead logging (WAL) performance
- ❑ All existing group logging implementations **sequential**

Sequential logging by single leader: under-utilizes SSD bandwidth



Challenge 2: Thread Sync Overhead in Group Logging

- ❑ **Group logging**: widely used to speed up write-ahead logging (WAL) performance
- ❑ All existing group logging implementations **sequential**



<u>Synchronization</u> costly: >80% of WAL write latency on Optane SSD

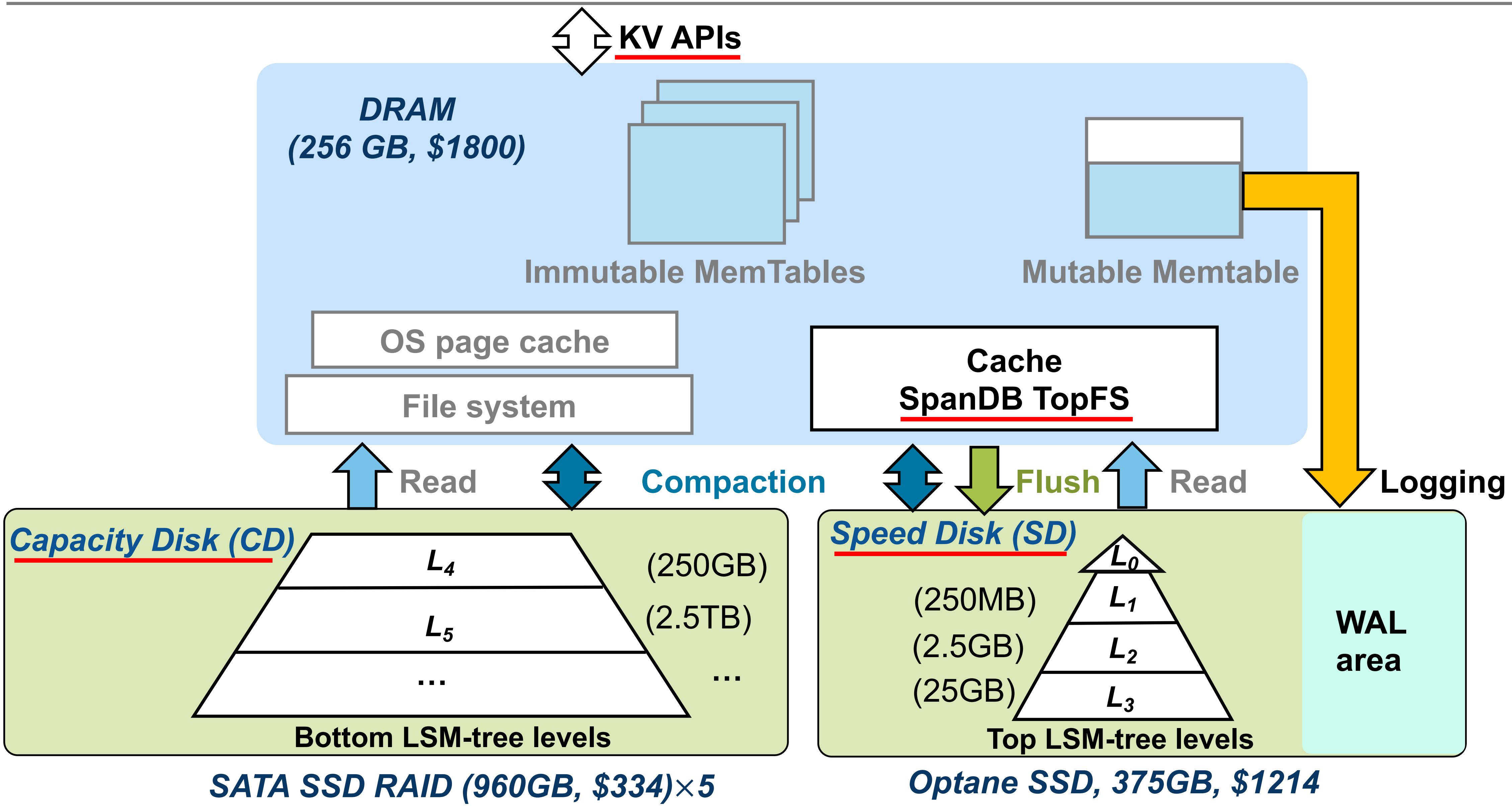
Related Work and Our Approach

- ❑ Optimized KV stores based on LSM-tree
  - ❖ PebblesDB [SOSP'17], SILK [ATC'19], ElasticBF [ATC'19], SplinterDB [ATC'20]
  - ❖ Limitations: data structure changes, using conventional Linux I/O stack

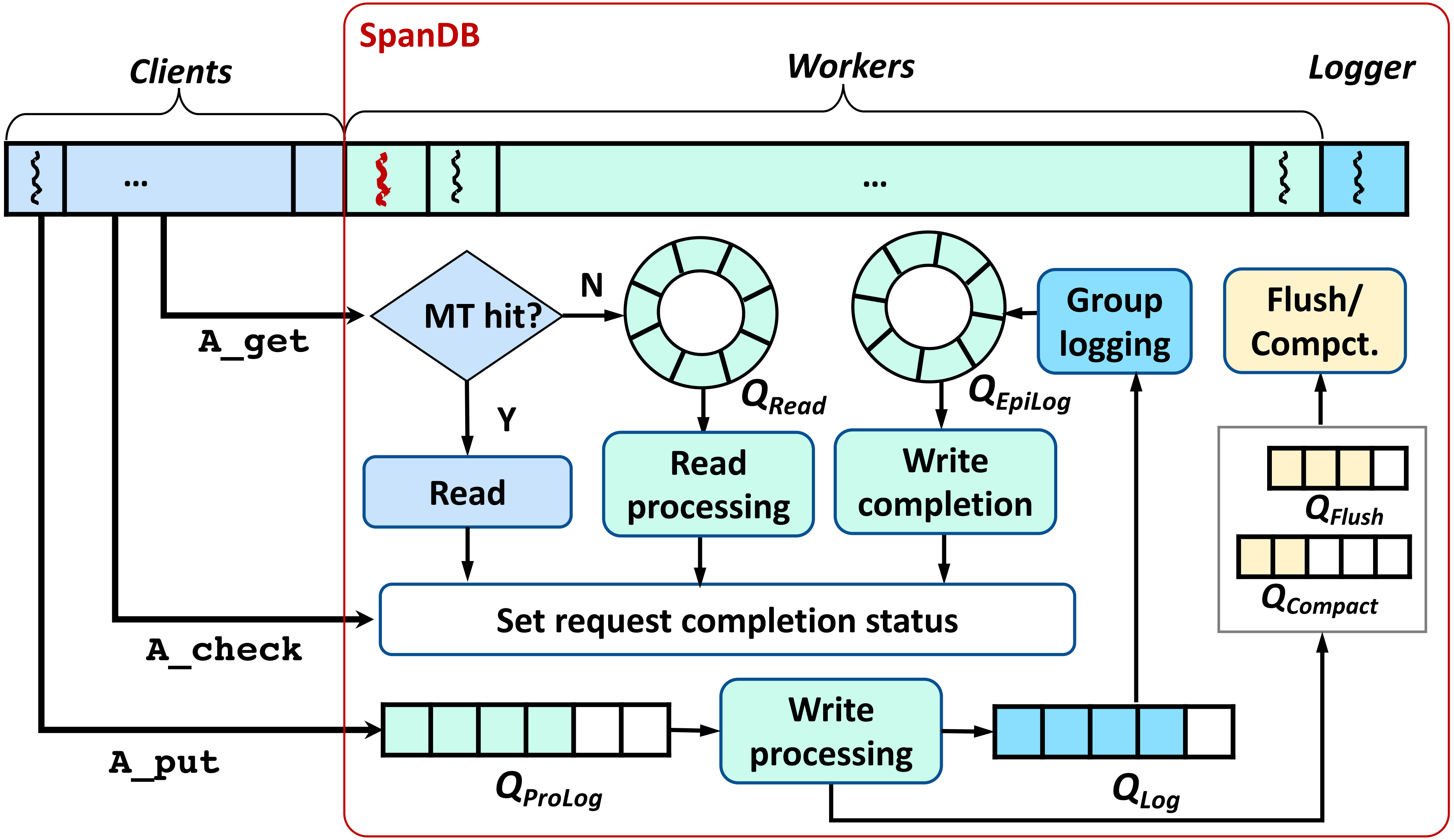
- ❑ Develop KV stores on NVMe SSDs
  - ❖ KVSSD [SYSTOR'19], KVell [SOSP'19], FlatStore [ASPLOS'20]
  - ❖ Limitations: High hardware cost, loss of transaction support in some cases

- ❑ **Our focus and major approach**
  - ❖ **Cost-effectiveness**: coupling small, fast devices with larger, slower ones
  - ❖ **Full utilization of fast device**: latency, bandwidth, and capacity
  - ❖ **Compatibility**: enhancing widely used RocksDB, no new data structures

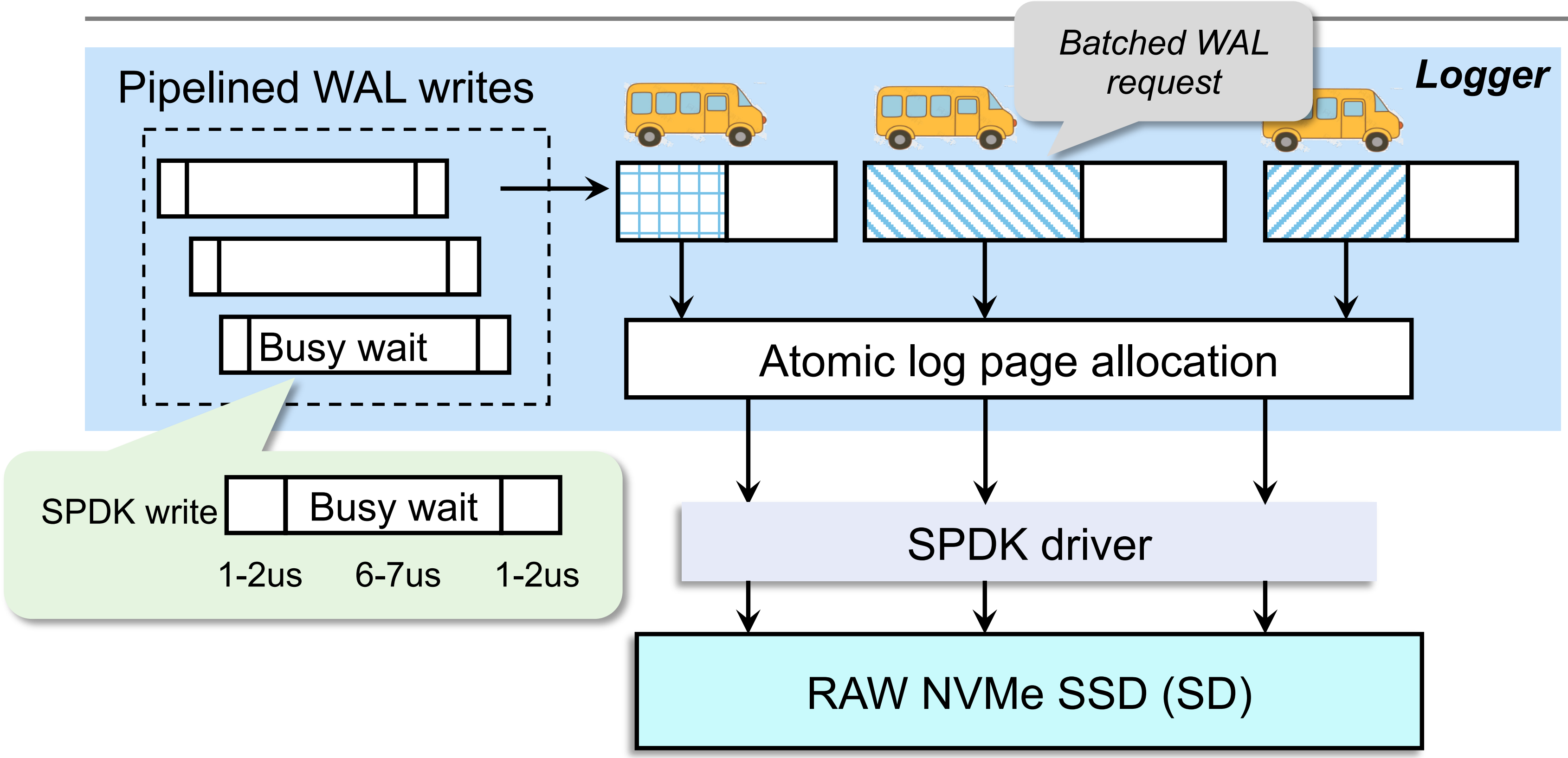
SpanDB Overview

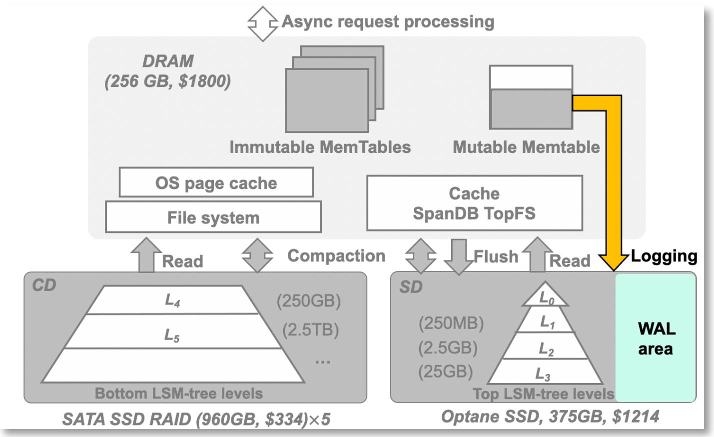


Async. KV Request Processing



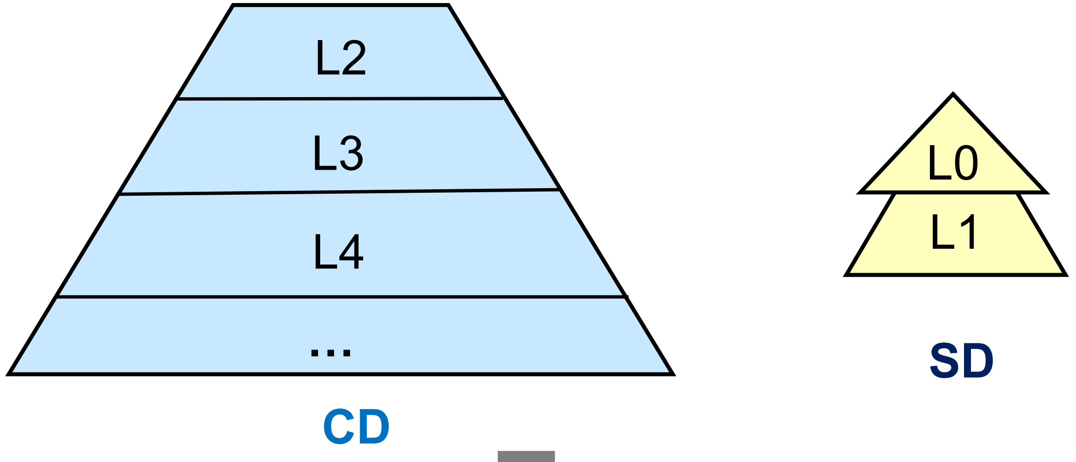
Parallel Logging via SPDK

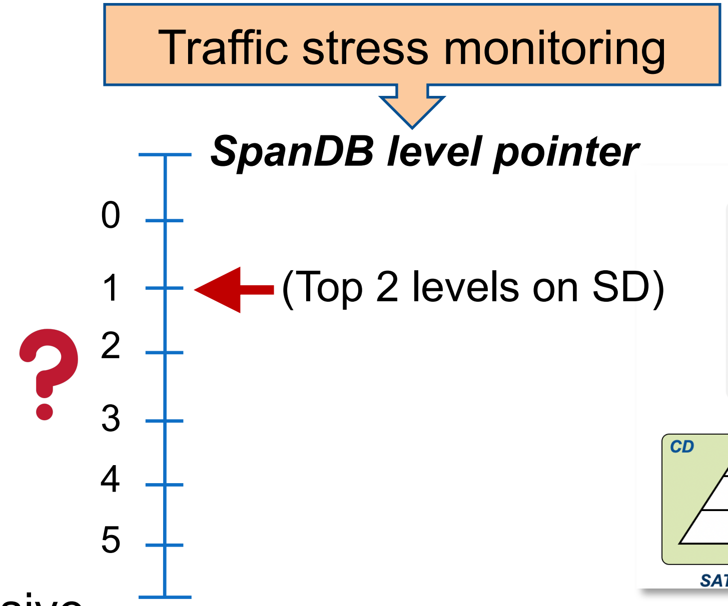


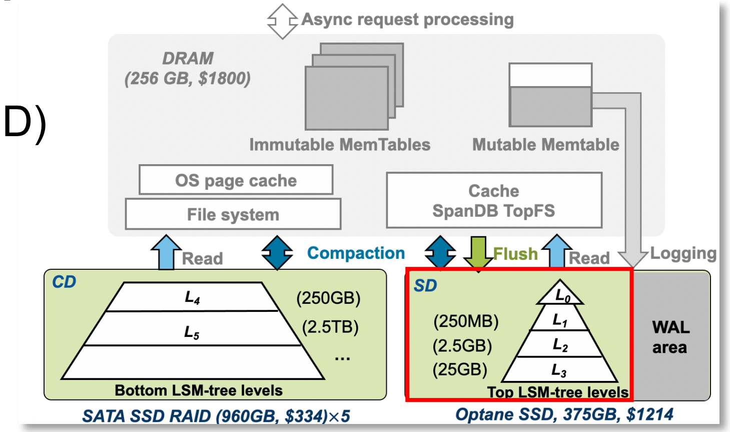


- ❑ WAL writes to log area on raw device via SPDK
  - ❖ **Concurrent**: for better NVMe device utilization
  - ❖ **Pipelined**: for better CPU time utilization busy
  - ❖ 1-2 dedicated loggers able to saturate Optane
  - ❖ Additional metadata management for consistency without FS

Dynamic LSM-Tree Level Placement

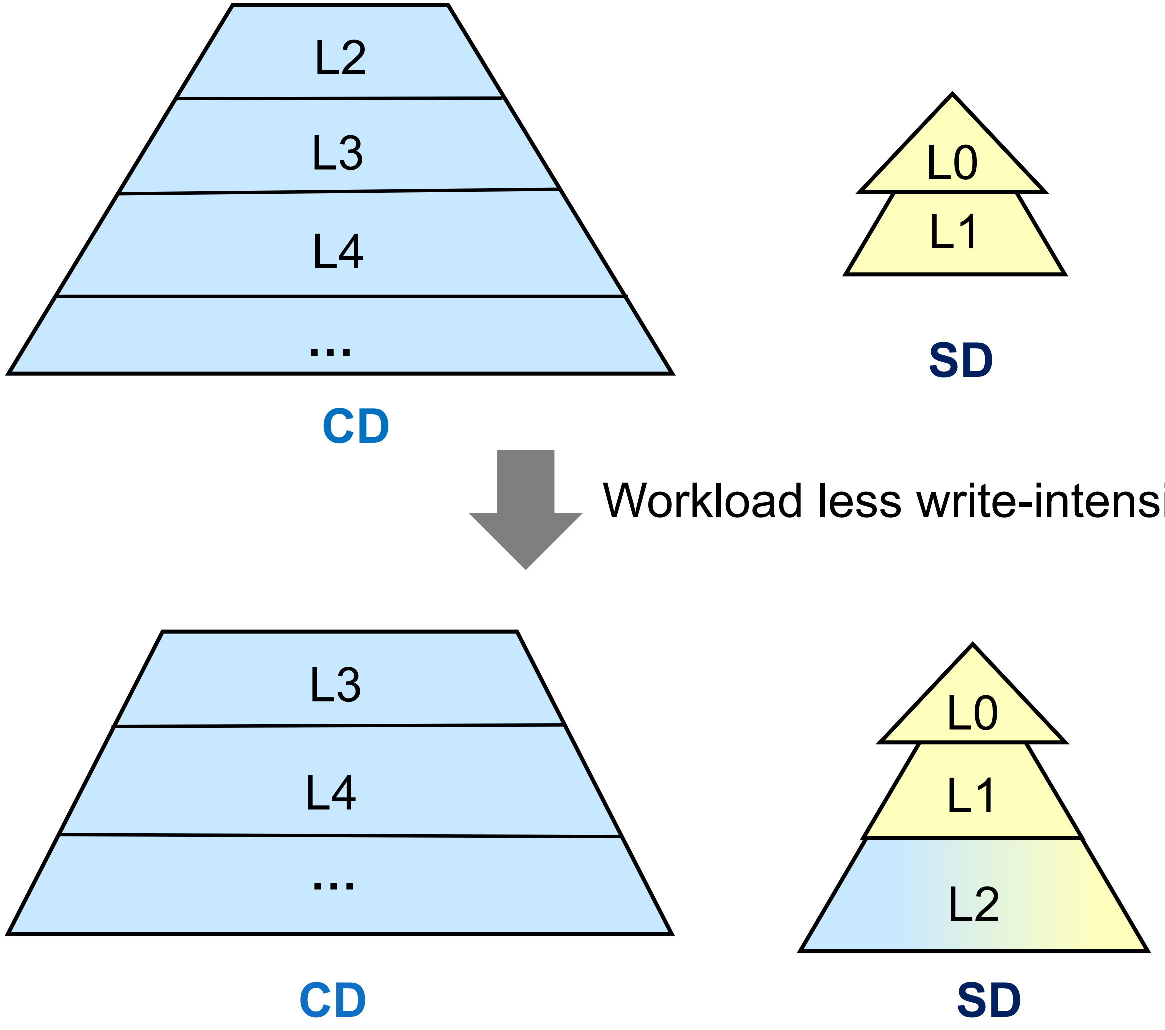


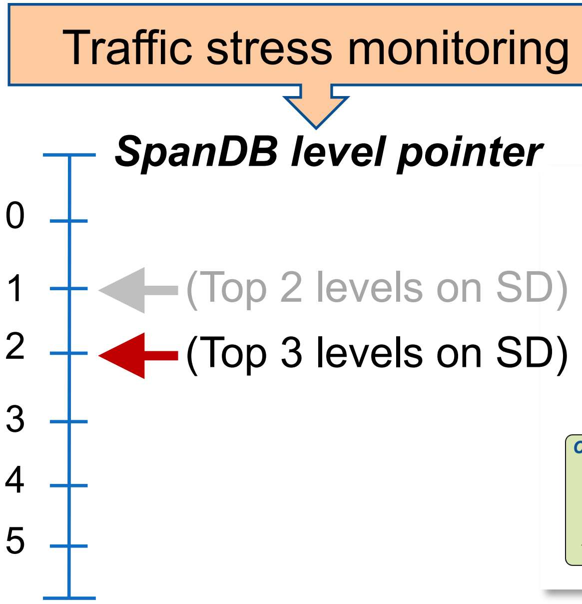


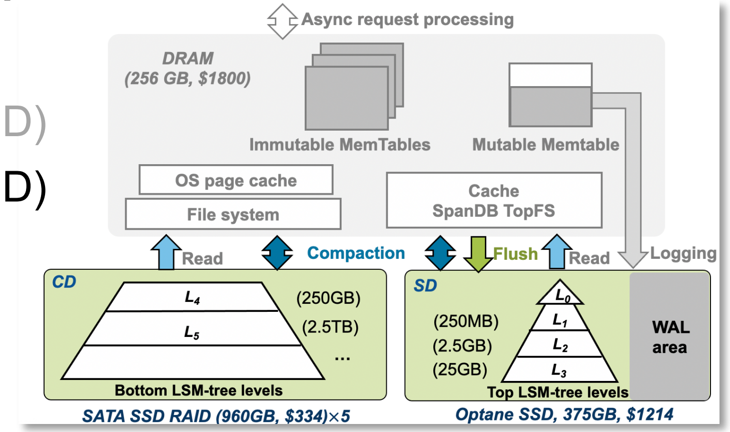


Workload less write-intensive

Dynamic LSM-Tree Level Placement







- ❑ All **new** writes to L2 will go to SD
- ❑ Lazy placement update, no active data migration
- ❑ Data cached at SD when level move to CD

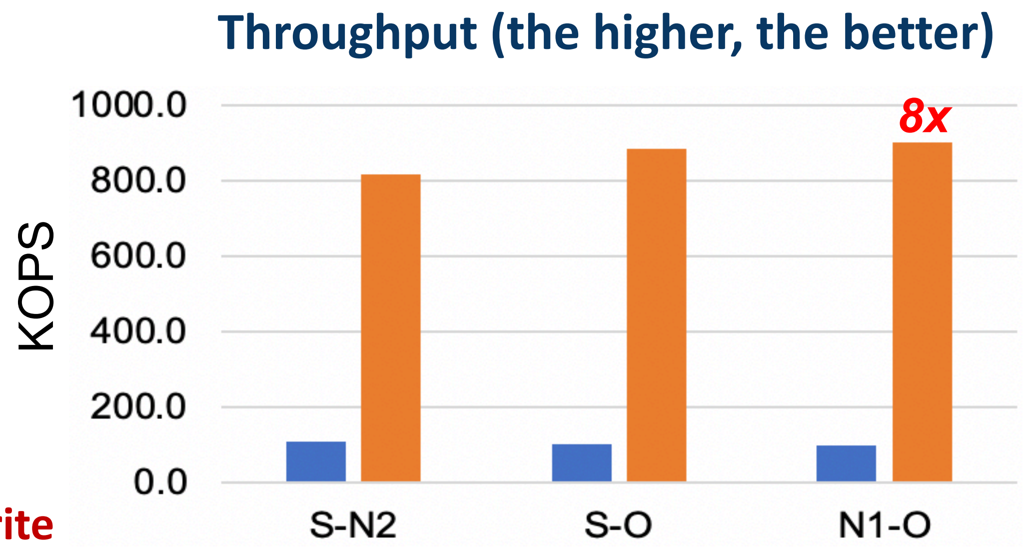
Experimental Setup

- Hardware
  - 2 20-core CPUs, 256GB memory
  - 4 types of **data center** storage devices

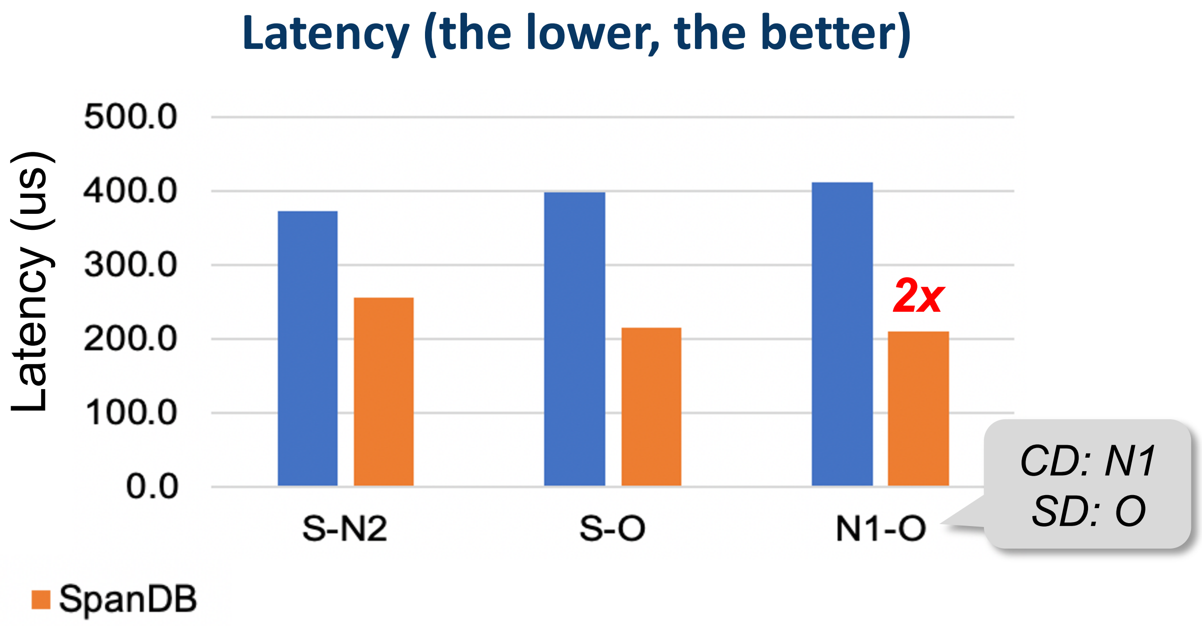
ID	Model	Price	Seq. write bandwidth	Write latency
S	Intel S4510 (SATA)	0.26 $/GB	510 MB/s	**37 us**
N1	Intel P4510 (NVMe)	0.25 $/GB	2900 MB/s	**18 us**
N2	Intel P4610 (NVMe)	0.40 $/GB	2080 MB/s	**18 us**
O	Intel Optane P4800X (NVMe)	3.25 $/GB	2000 MB/s	**10 us**

- Workloads: **YCSB** and **LinkBench**
- Baselines: **RocksDB** (v6.5.1), **KVell** [SOSP'19], and **RocksDB-BlobFS**
- Database size: primarily with **512GB**, up to **2TB**

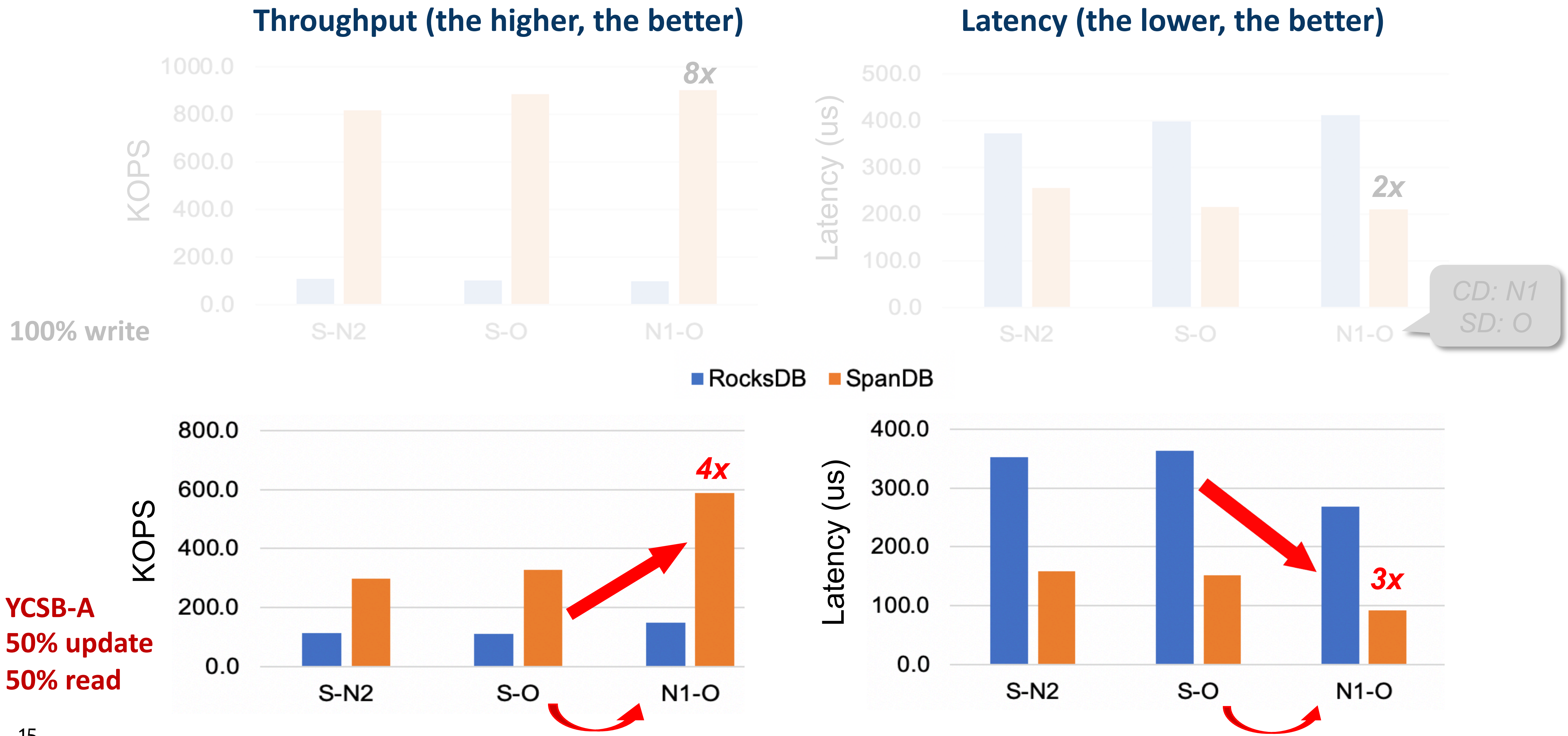
YCSB Results, Comparison w. RocksDB



100% write

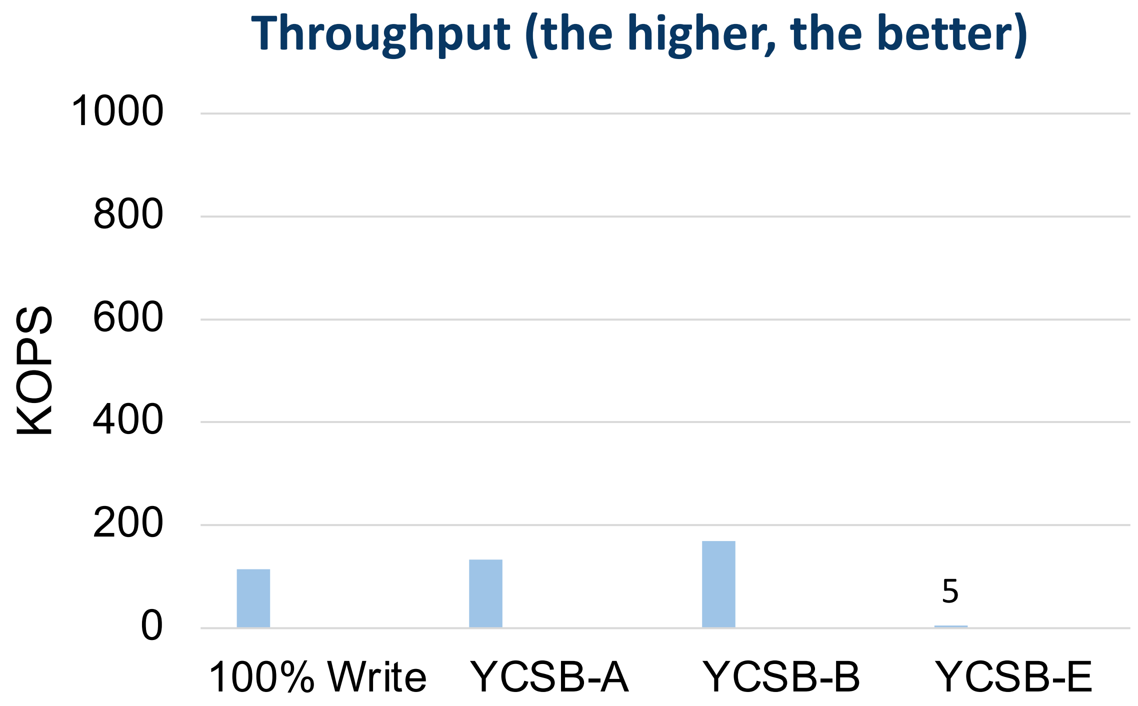


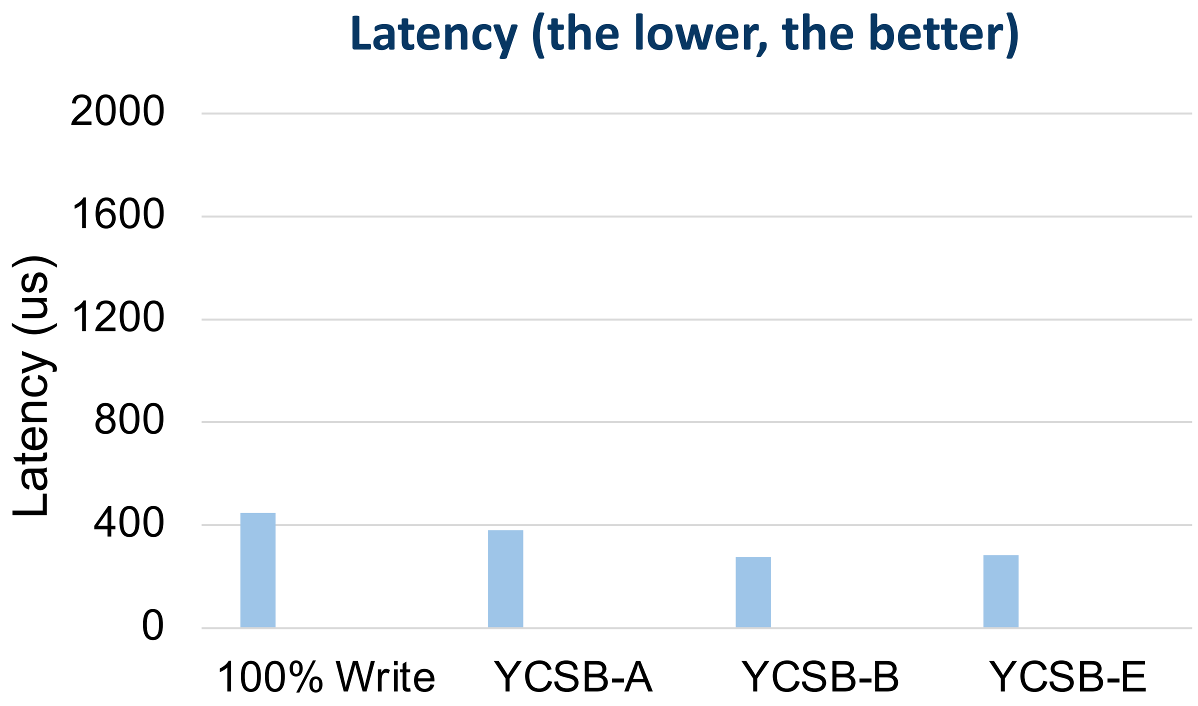
YCSB Results, Comparison w. RocksDB



YCSB Results, Comparison w. KVell

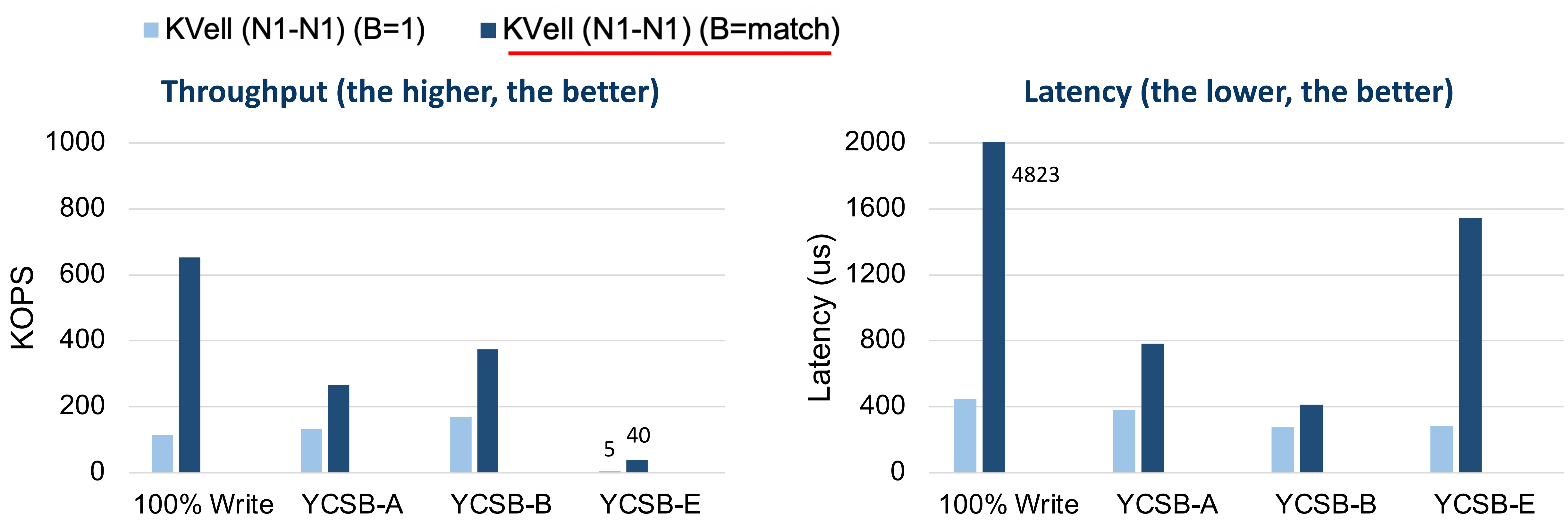
KVell (N1-N1) (B=1)





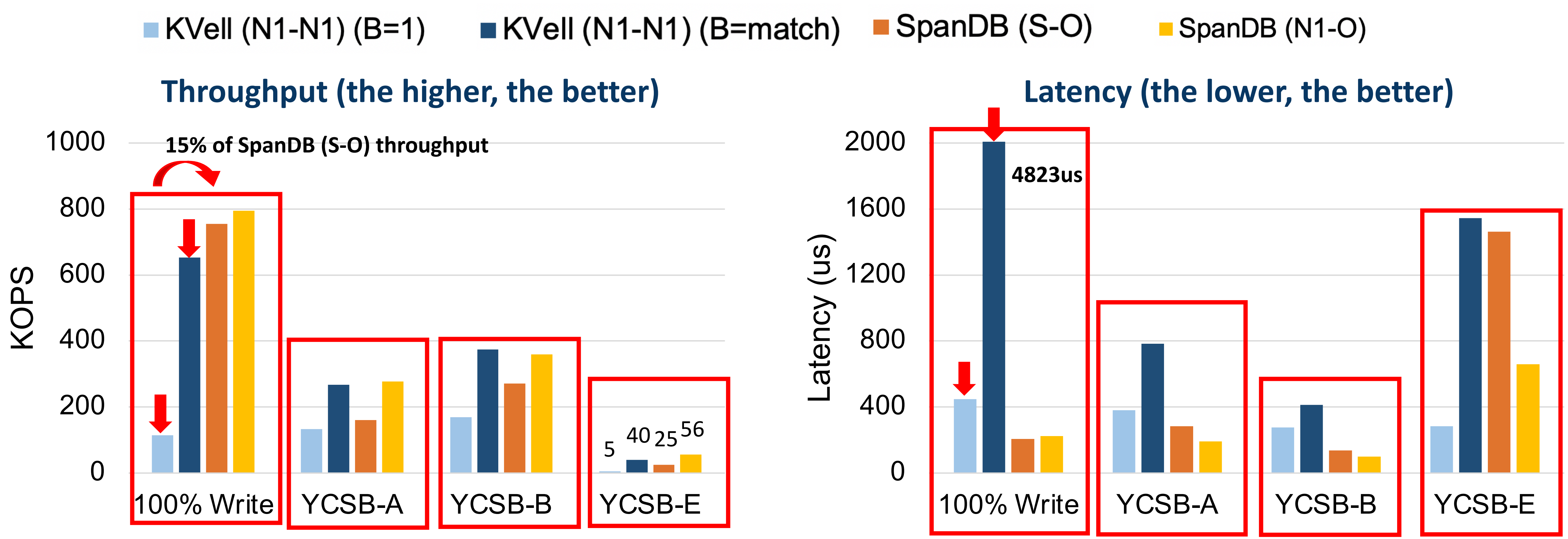
- 2TB database
- YCSB-A: 50% update and 50% read, YCSB-B: 5% update and 95% read, YCSB-E: 5% update and 95% scan
- KVell (B=1): batch size = 1 in KVell

YCSB Results, Comparison w. KVell



- 2TB database
- YCSB-A: 50% update and 50% read, YCSB-B: 5% update and 95% read, YCSB-E: 5% update and 95% scan
- KVell (B=1): batch size = 1 in KVell
- KVell (B=match): the smallest batch size that surpasses SpanDB's throughput

YCSB Results, Comparison w. KVell



- 2TB database
- YCSB-A: 50% update and 50% read, YCSB-B: 5% update and 95% read, YCSB-E: 5% update and 95% scan
- KVell (B=1): batch size = 1 in KVell
- KVell (B=match): the smallest batch size that surpasses SpanDB's throughput

SpanDB: A Fast, Cost-Effective LSM-tree Based KV Store on Hybrid Storage

Open-source: https://github.com/SpanDB/SpanDB

Thanks

{cighao, rcy}@mail.ustc.edu.cn, {chengli7, ylxu}@ustc.edu.cn, xma@hbku.edu.qa