LATTE: A Native Table Engine on NVMe Storage

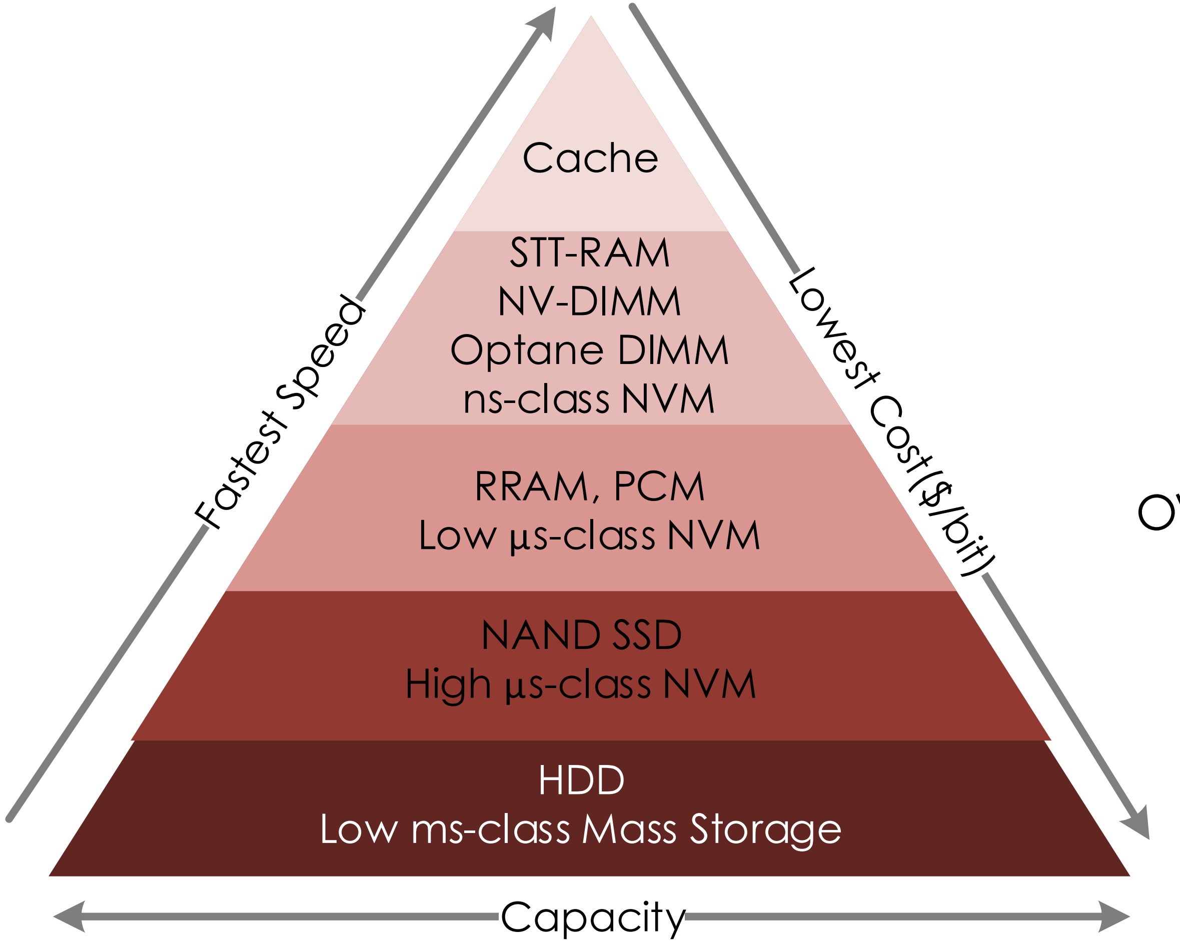
Jiajia Chu, Yunshan Tu, Yao Zhang, and Chuliang Weng*

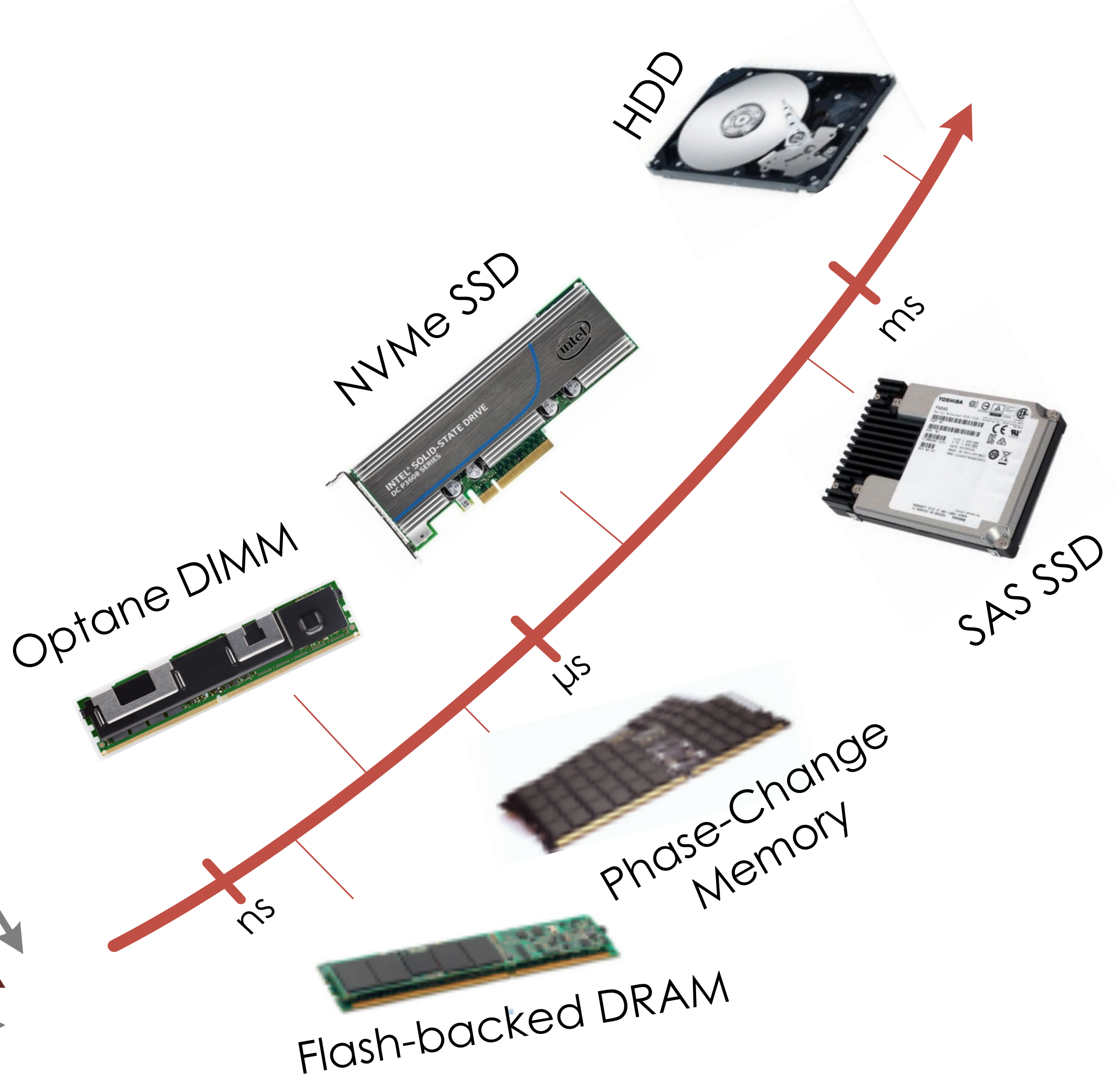
East China Normal University

Outline

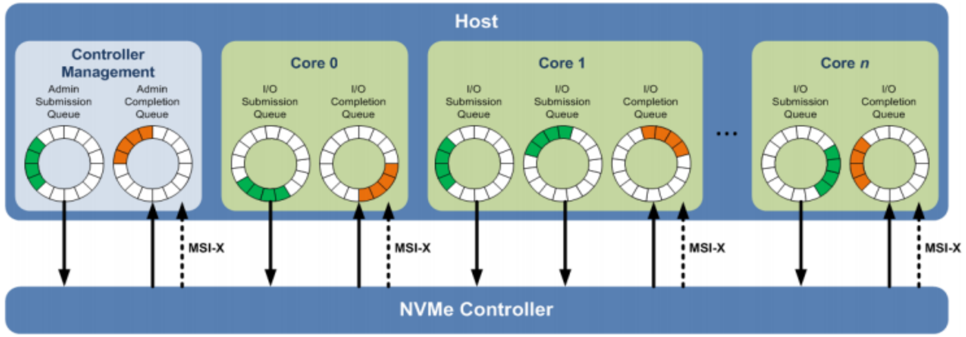
- Background
- Motivation
- Design
  - Architecture
  - Parallel Queues Scheduling
  - Undo Logging on Heterogeneous Storage
- Evaluation
- Conclusion

Background


Cache
STT-RAM
NV-DIMM
Optane DIMM
ns-class NVM
RRAM, PCM
Low µs-class NVM
NAND SSD
High µs-class NVM
HDD
Low ms-class Mass Storage
Fastest Speed
Lowest Cost($/bit)
Capacity


HDD
NVMe SSD
Optane DIMM
SAS SSD
Phase-Change Memory
Flash-backed DRAM
ns
µs
ms

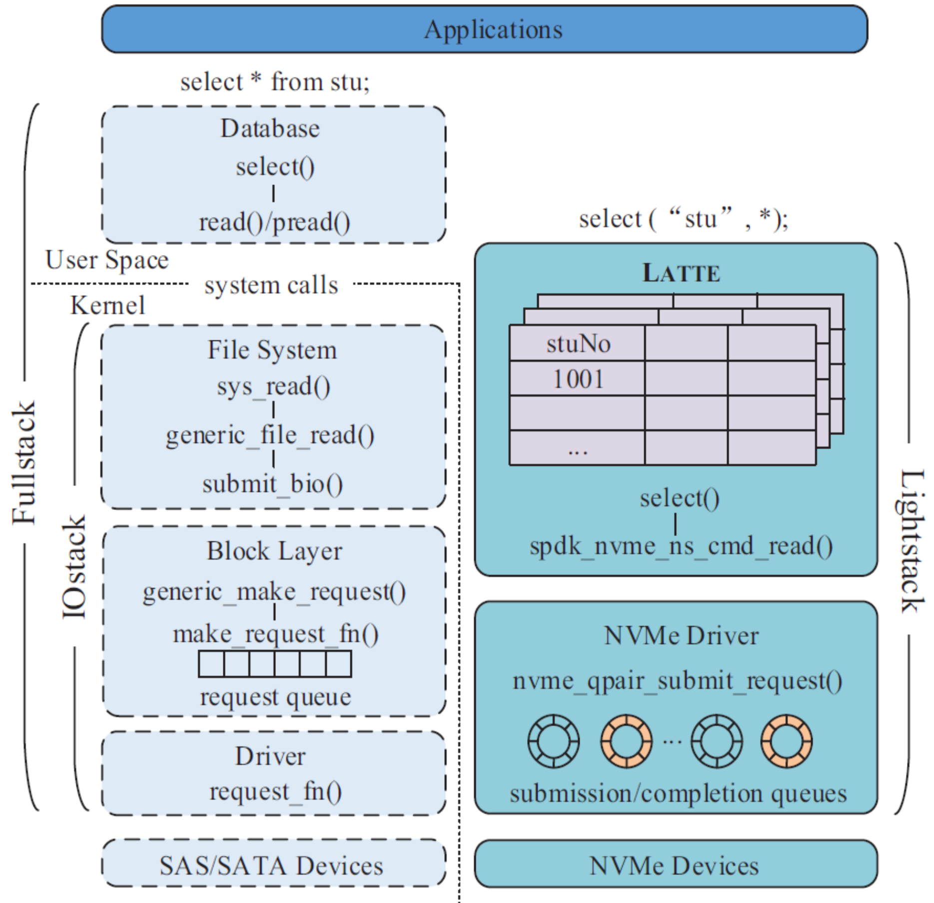
Background



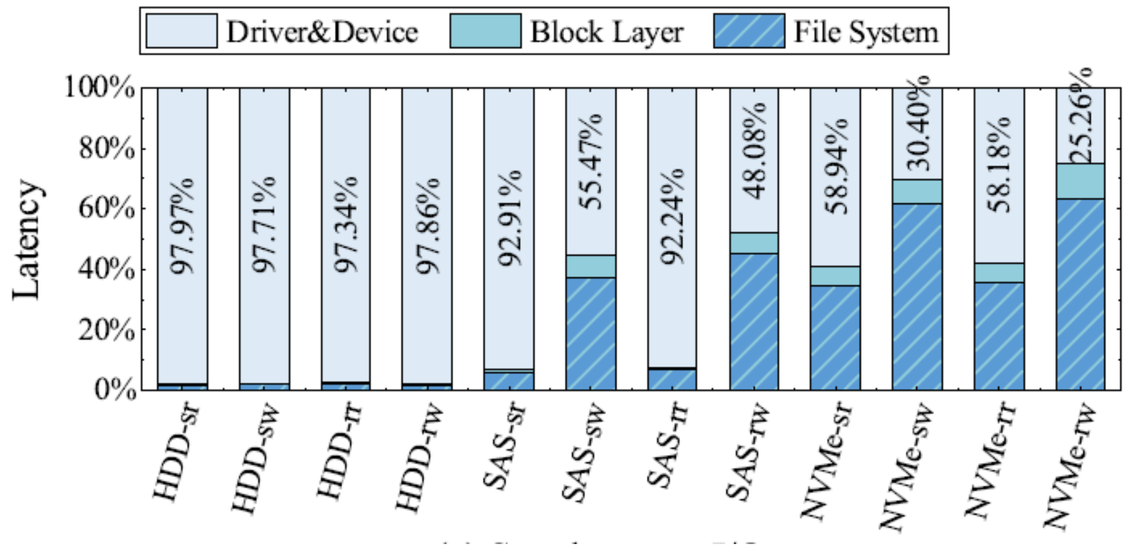
- NVMe is an extensible host controller interface protocol that provides efficient access to storage devices.
- Each NVMe controller corresponds to one pair of admin queues and multiple pairs of I/O queues.
- Multiple deep queues can execute I/O commands in parallel, considerably increasing the bandwidth of NVMe SSDs.

Type	Name	Write IOPS	Latency	Specification
SAS/SATA HDD	WDC WD40EZRX-75SPEB0	150 MB/s	2-10 ms	http://products.wdc.com/library/SpecSheet/ENG/2879-771438.pdf
SAS/SATA SSD	PX04SRB048	750 MB/s 22,000 IOPS	~200 μs	https://business.kioxia.com/content/dam/kioxia/shared/business/ssd/doc/eSSD-PX04SRB_PX04SRQ_Series.pdf
NVMe SSD	Intel P3608	3,000 MB/s 150,000 IOPS	20 μs	https://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-dc-p3608-spec.pdf

Motivation



(a) (b)



The software on the NVMe SSD accounts for a larger proportion. The overhead of writing even reaches 70%. The larger proportion of software overhead cannot be overlooked and motivates us to rethink the traditional layered storage stack.

Motivation

lower latency

higher bandwidth

efficient random write

multiple I/O queues

NVMe SSD

Motivation

lower latency

higher bandwidth

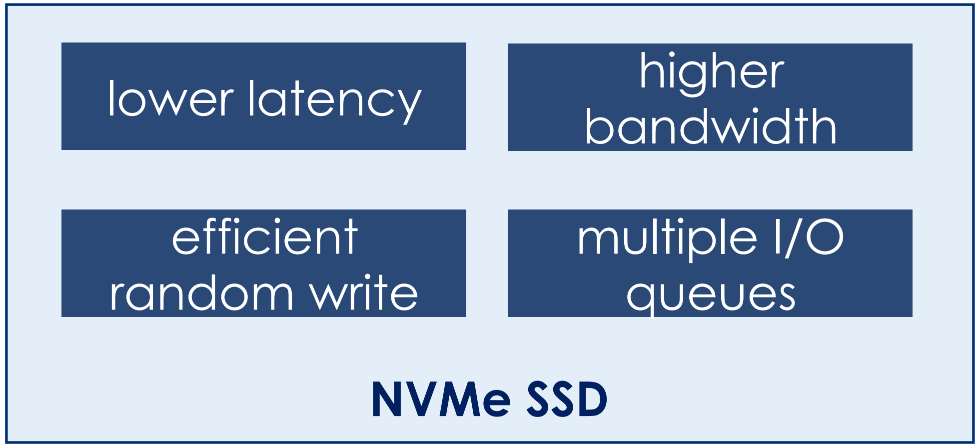
efficient random write

multiple I/O queues

NVMe SSD

- How to lighten the native storage stack
- How to use efficient random write to reduce redundant data writing and storage
- How to leverage multiple I/O queues to achieve higher performance

Motivation



- How to lighten the native storage stack
- How to use efficient random write to reduce redundant data writing and storage
- How to leverage multiple I/O queues to achieve higher performance

Design

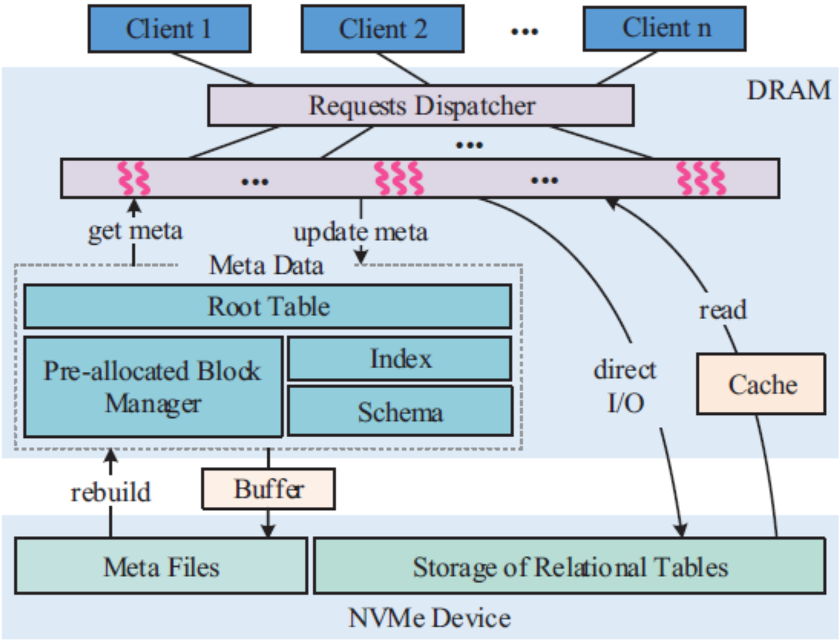
LATTE is a heterogeneous and efficient table storage engine designed for NVMe devices. It has three design goals:

Lower Latency—Short-Path Direct I/O. Following the Lightstack, LATTE bypasses the traditional Fullstack and implements a shorter path for data storage in user space.

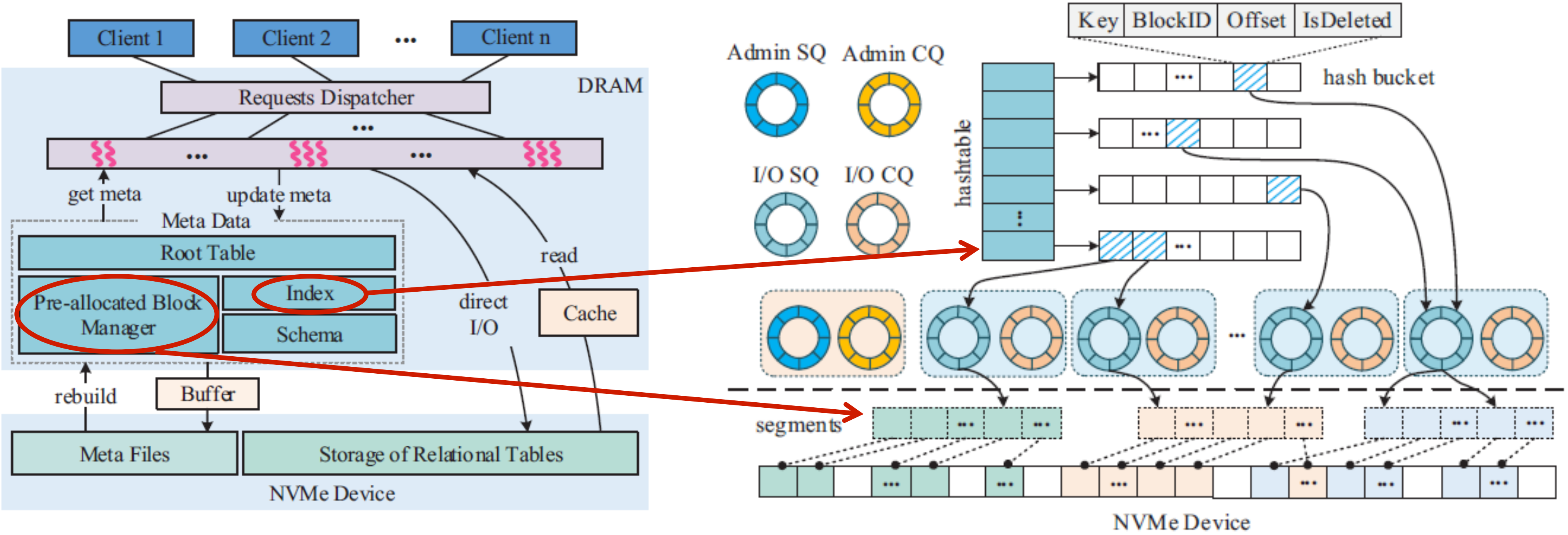
Higher IOPS—Parallel Queues Scheduling. LATTE allows different I/O queues to manipulate different tables in parallel to leverage the hardware parallelism fully. It also minimizes the write conflicts by pre-allocating physical storage space for each table.

Reduced Storage Footprint—Undo Logging. LATTE stores table data in NVMe SSDs while preserving the undo logs in NVDIMM to balance the performance, storage overhead, and data consistency.

Parallel Queues Scheduling

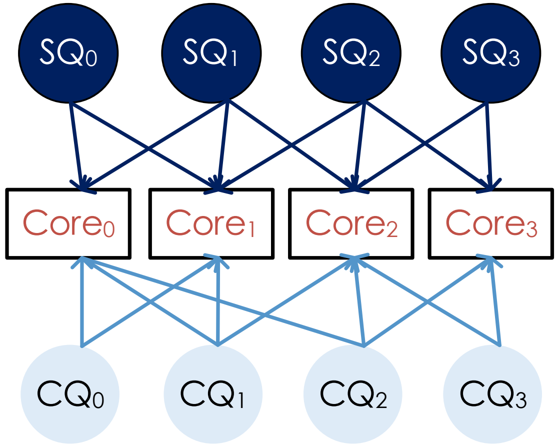


Parallel Queues Scheduling



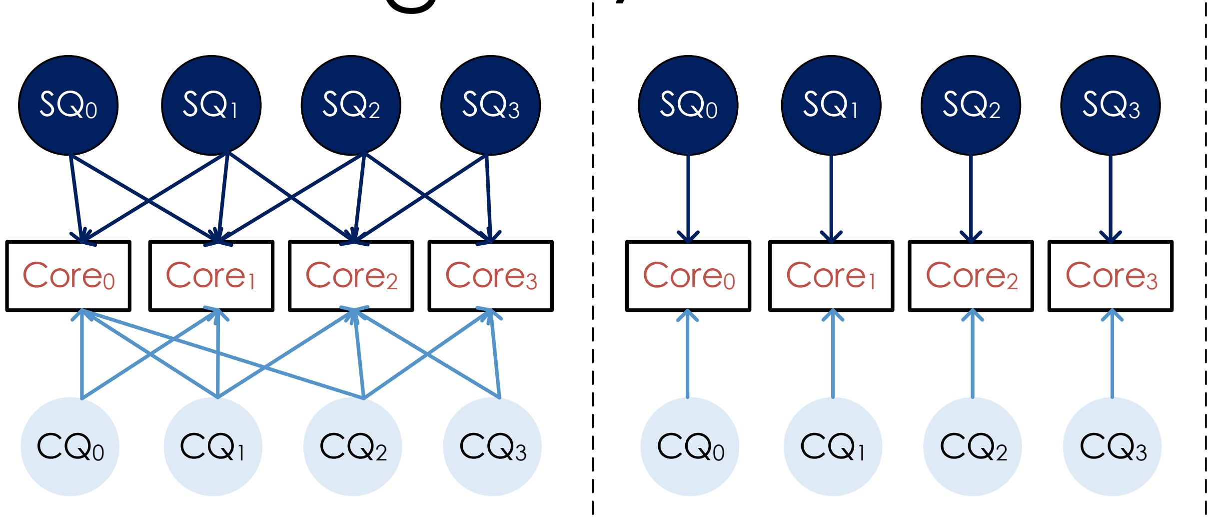
LATTE divides the NVMe device into massive data blocks, which are the smallest read and write units. We define a certain number of physical data blocks as a segment. Each segment belongs to only one table. When LATTE allocates data blocks for insert operations, it needs to access the segment manager to obtain available data blocks.

Binding of I/O Queues and CPU Cores



(a) Naïve

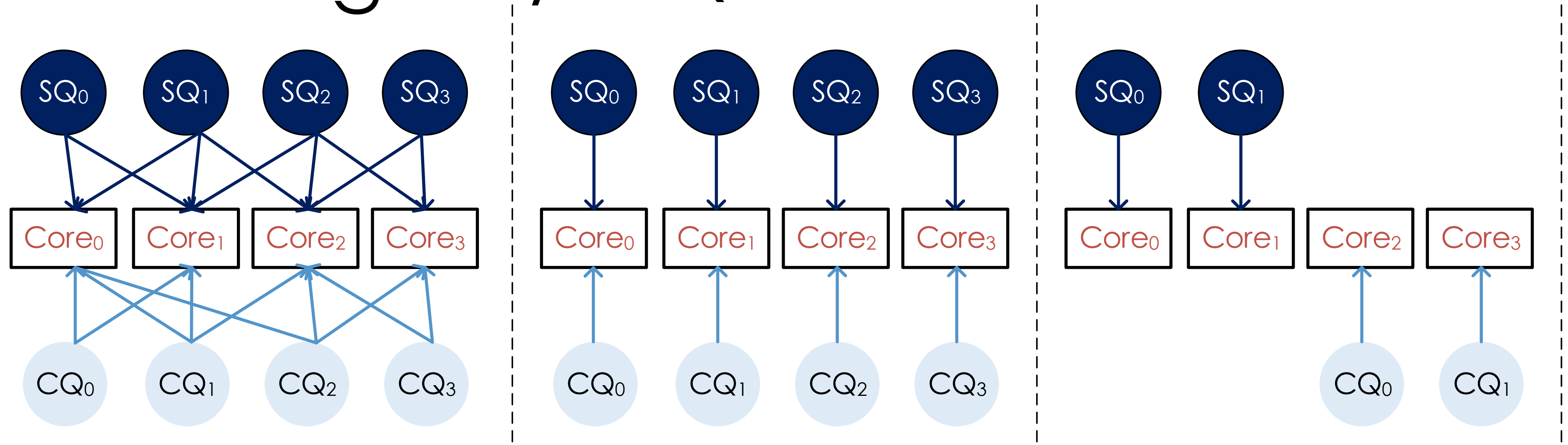
Binding of I/O Queues and CPU Cores



(a) Naïve

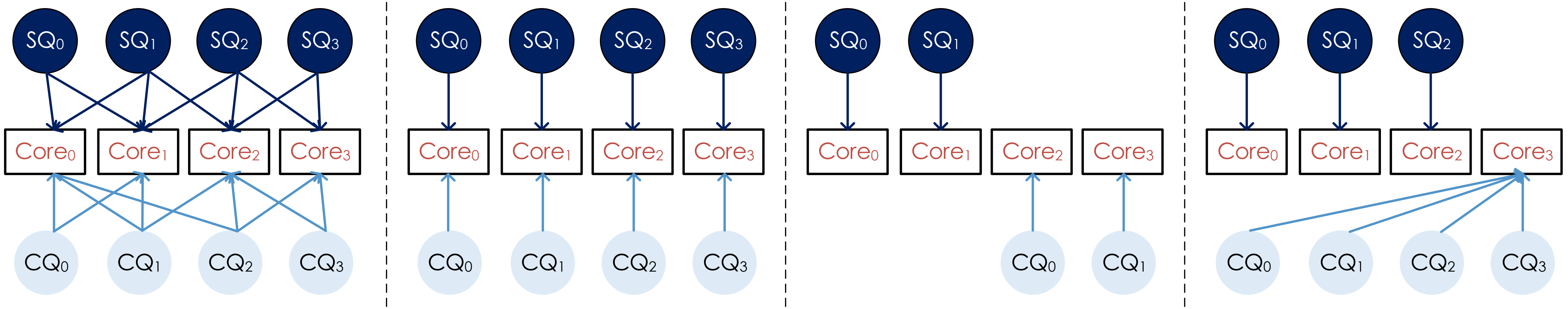
(b) Pair-Binding

Binding of I/O Queues and CPU Cores



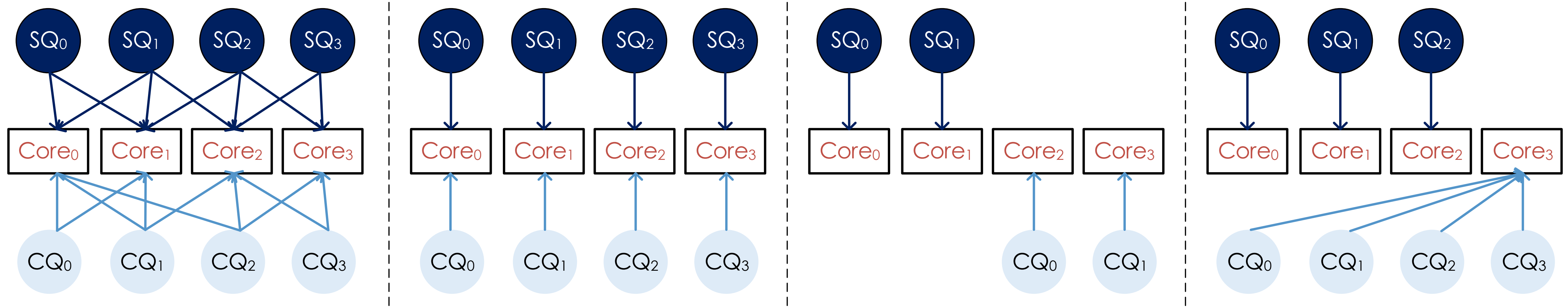
(a) Naïve (b) Pair-Binding (c) Separated-Binding

Binding of I/O Queues and CPU Cores



(a) Naïve (b) Pair-Binding (c) Separated-Binding (d) Skew-Binding

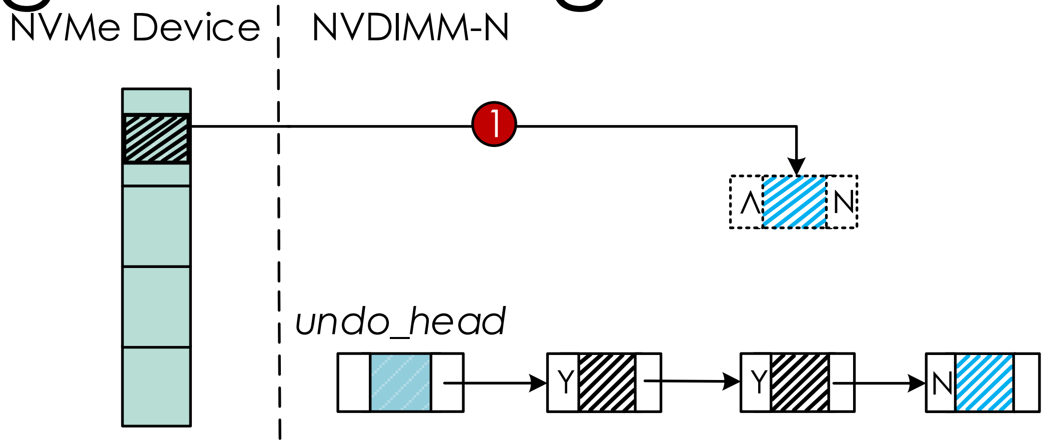
Binding of I/O Queues and CPU Cores



(a) Naïve (b) Pair-Binding (c) Separated-Binding (d) Skew-Binding

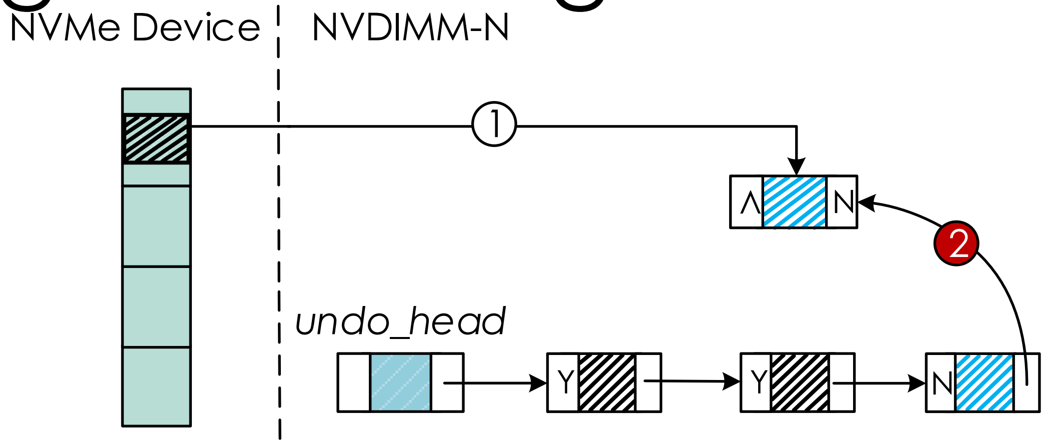
<table>
<tr><th></th><th colspan="3">Strengths</th><th colspan="2">Weaknesses</th></tr>
<tr><td>Naïve</td><td colspan="3">Providing more balanced use of CPU resources.</td><td colspan="2">Increasing the cache miss and thread migration.</td></tr>
<tr><td>Pair-Binding</td><td rowspan="3">Reducing the cache miss and thread migration.</td><td>\</td><td>\</td><td rowspan="3">Some CPU cores are unbound and idle.</td><td>SQ and CQ bound to the same CPU core compete for CPU resources.</td></tr>
<tr><td>Separated-Binding</td><td rowspan="2">Reducing CPU competition between SQ and CQ.</td><td>\</td><td>Consuming more CPU resources for polling.</td></tr>
<tr><td>Skew-Binding</td><td>Saving more CPU resource for calculations and data processing.</td><td>All CQs occupy only one CPU core for polling, so the check for each I/O task completion is slightly slower than that of Separated-Binding.</td></tr>
</table>

Undo Logging on Heterogeneous Storage



(1) When a write request arrives, an undo log will be written into the non-volatile buffer by the CLWB and MFENCE instructions.

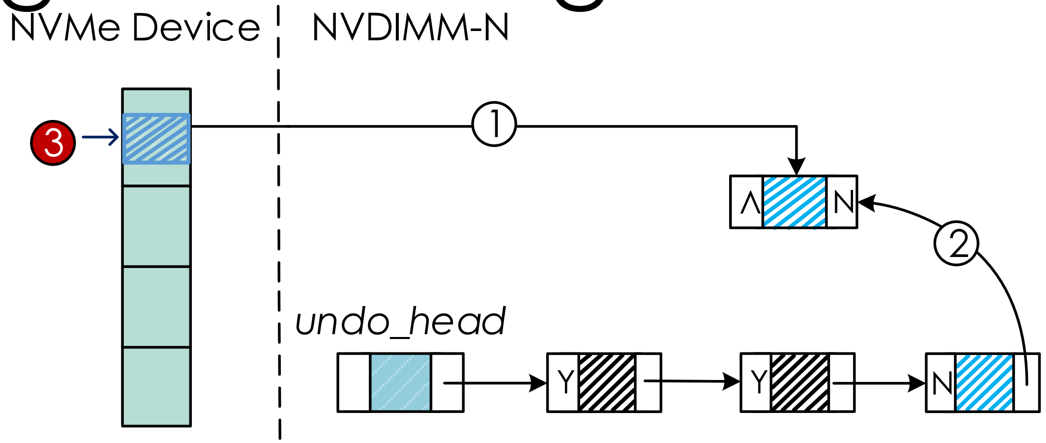
Undo Logging on Heterogeneous Storage



(1) When a write request arrives, an undo log will be written into the non-volatile buffer by the CLWB and MFENCE instructions.

(2) Next, the undo log will be appended to the end of the log list.

Undo Logging on Heterogeneous Storage

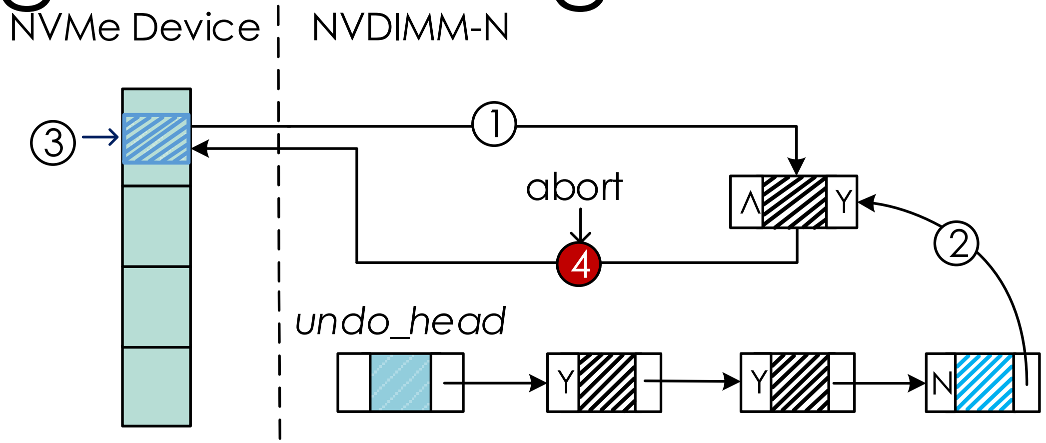


(1) When a write request arrives, an undo log will be written into the non-volatile buffer by the CLWB and MFENCE instructions.

(2) Next, the undo log will be appended to the end of the log list.

(3) The associated tuple data in the NVMe devices will then be updated.

Undo Logging on Heterogeneous Storage



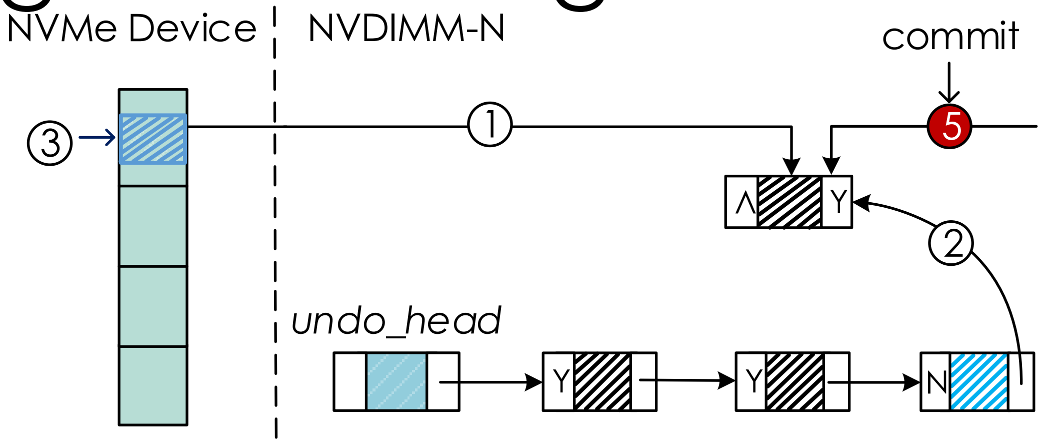
(1) When a write request arrives, an undo log will be written into the non-volatile buffer by the CLWB and MFENCE instructions.

(2) Next, the undo log will be appended to the end of the log list.

(3) The associated tuple data in the NVMe devices will then be updated.

(4) If the update fails, the modification will be aborted and the associated tuple data will be rolled back using the undo log in the buffer. After that, the undo log is marked as expired ('Y').

Undo Logging on Heterogeneous Storage



(1) When a write request arrives, an undo log will be written into the non-volatile buffer by the CLWB and MFENCE instructions.

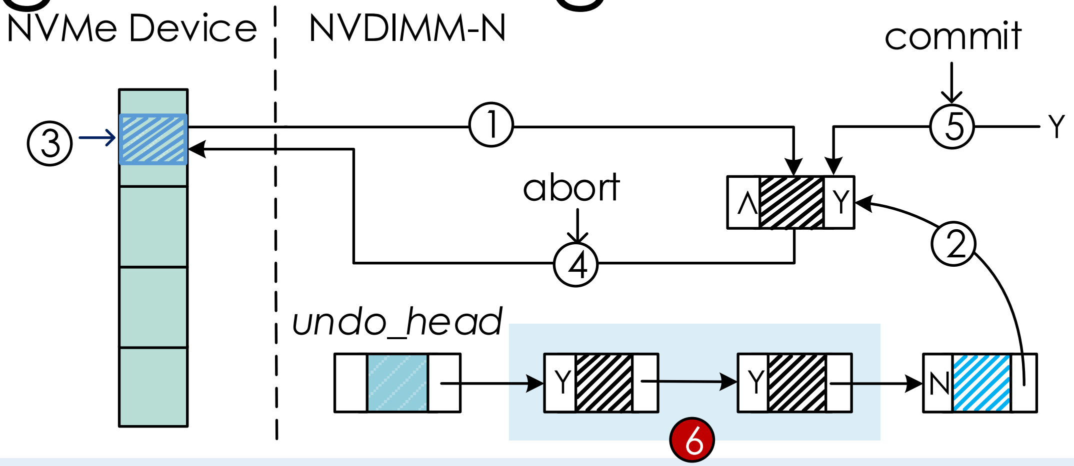
(2) Next, the undo log will be appended to the end of the log list.

(3) The associated tuple data in the NVMe devices will then be updated.

(4) If the update fails, the modification will be aborted and the associated tuple data will be rolled back using the undo log in the buffer. After that, the undo log is marked as expired (‘Y’).

(5) If the updated data has been committed, the corresponding undo log will also be marked as expired.

Undo Logging on Heterogeneous Storage



(1) When a write request arrives, an undo log will be written into the non-volatile buffer by the CLWB and MFENCE instructions.

(2) Next, the undo log will be appended to the end of the log list.

(3) The associated tuple data in the NVMe devices will then be updated.

(4) If the update fails, the modification will be aborted and the associated tuple data will be rolled back using the undo log in the buffer. After that, the undo log is marked as expired ('Y').

(5) If the updated data has been committed, the corresponding undo log will also be marked as expired.

(6) Finally, the garbage collection thread will release the expired undo logs in batches.

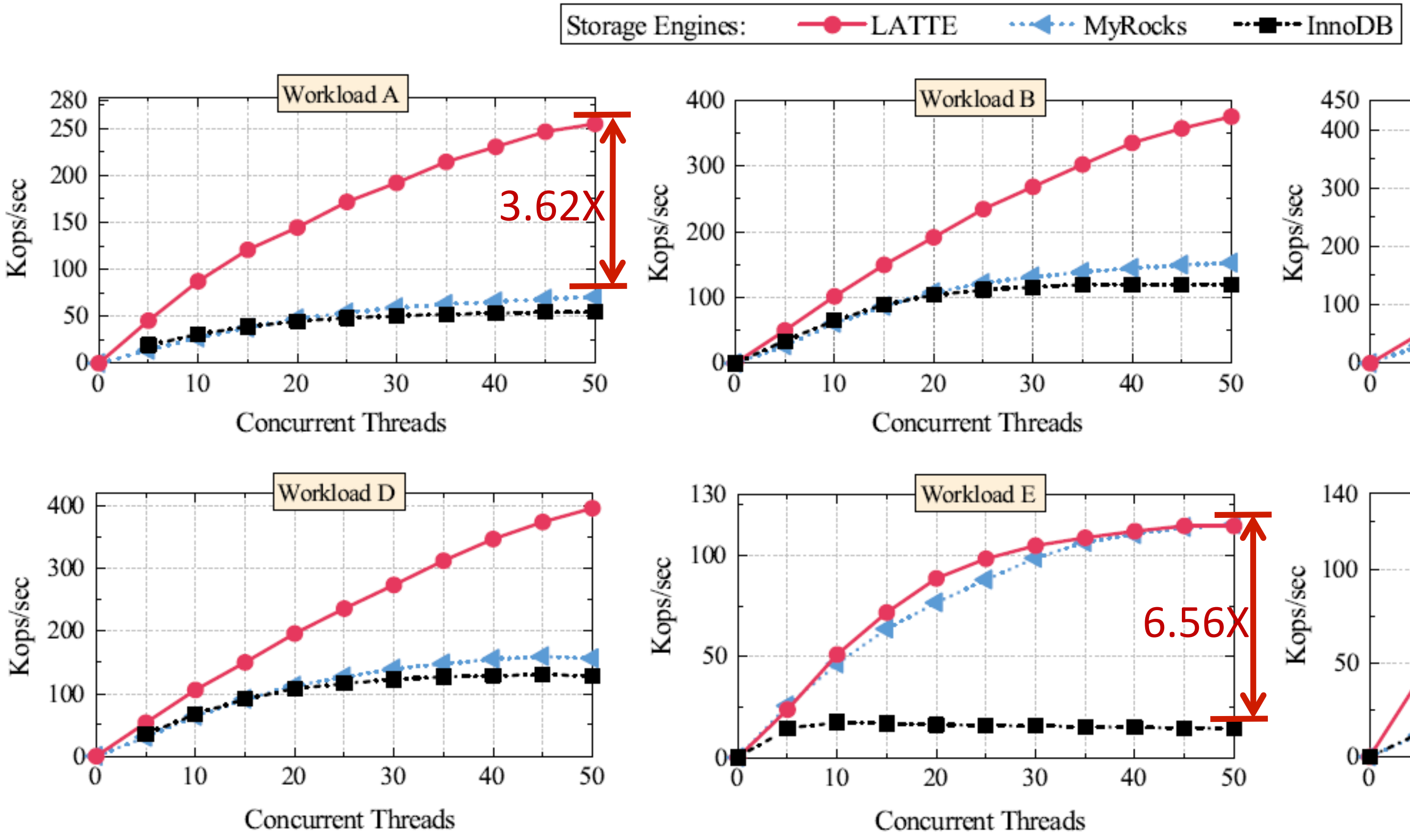
Evaluation

- Dual ten-core Xeon E5-2630 v4 CPUs
- A 3.2 TB Intel DC P3608 Series NVMe SSD
- A 375 GB Intel Optane DC P4800X NVMe SSD
- 224 GB DRAM
- CentOS 7 with Linux 3.10.0 kernel

We used the C++ programming language to implement LATTE with 7,200 lines of code, and utilized the NVMe driver library of the Storage Performance Development Kit (SPDK) during the implementation.

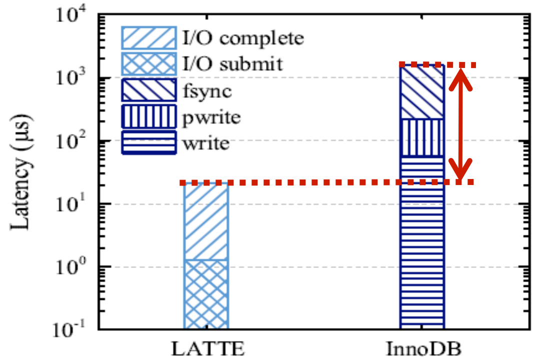
Evaluation

Throughput

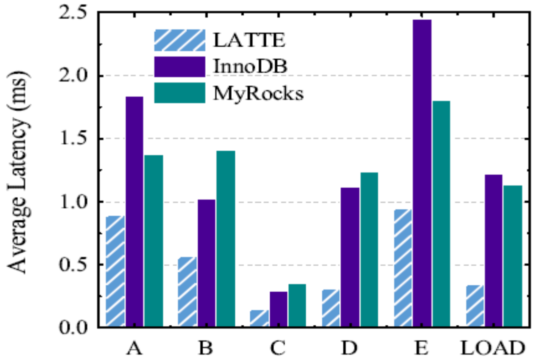


Evaluation

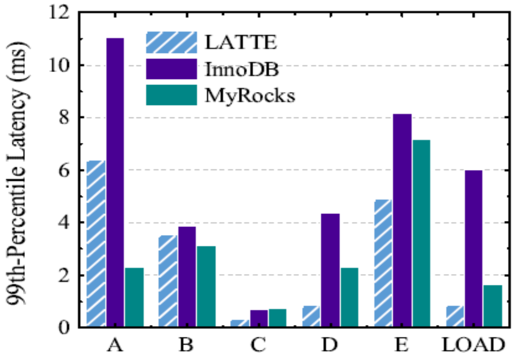
Latency



(a) Break-Down of an Insert.

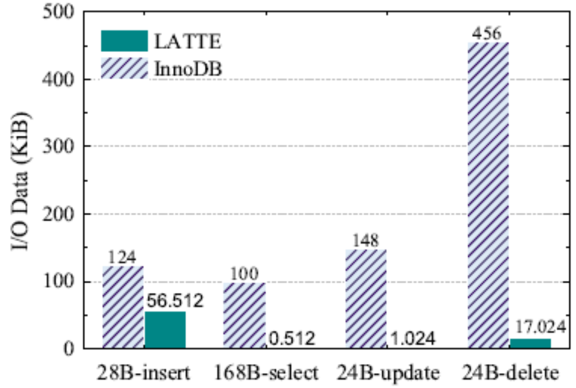


(b) YCSB Average Latency.

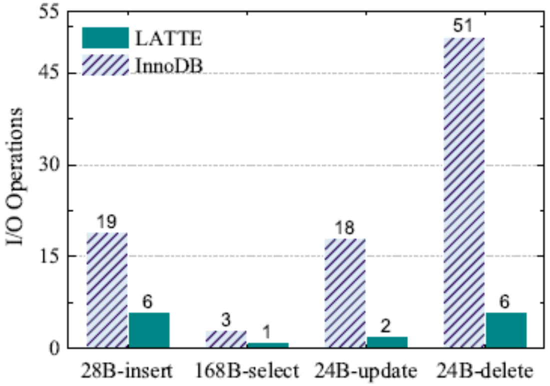


(c) YCSB Tail Latency.

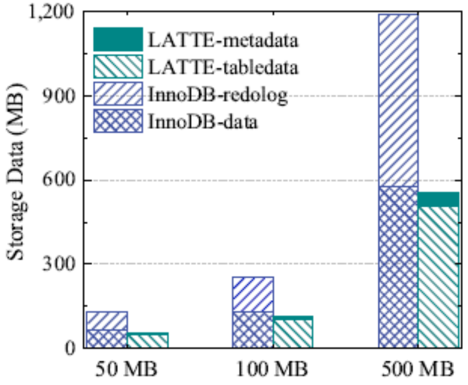
Storage Overhead



(a) I/O Data.

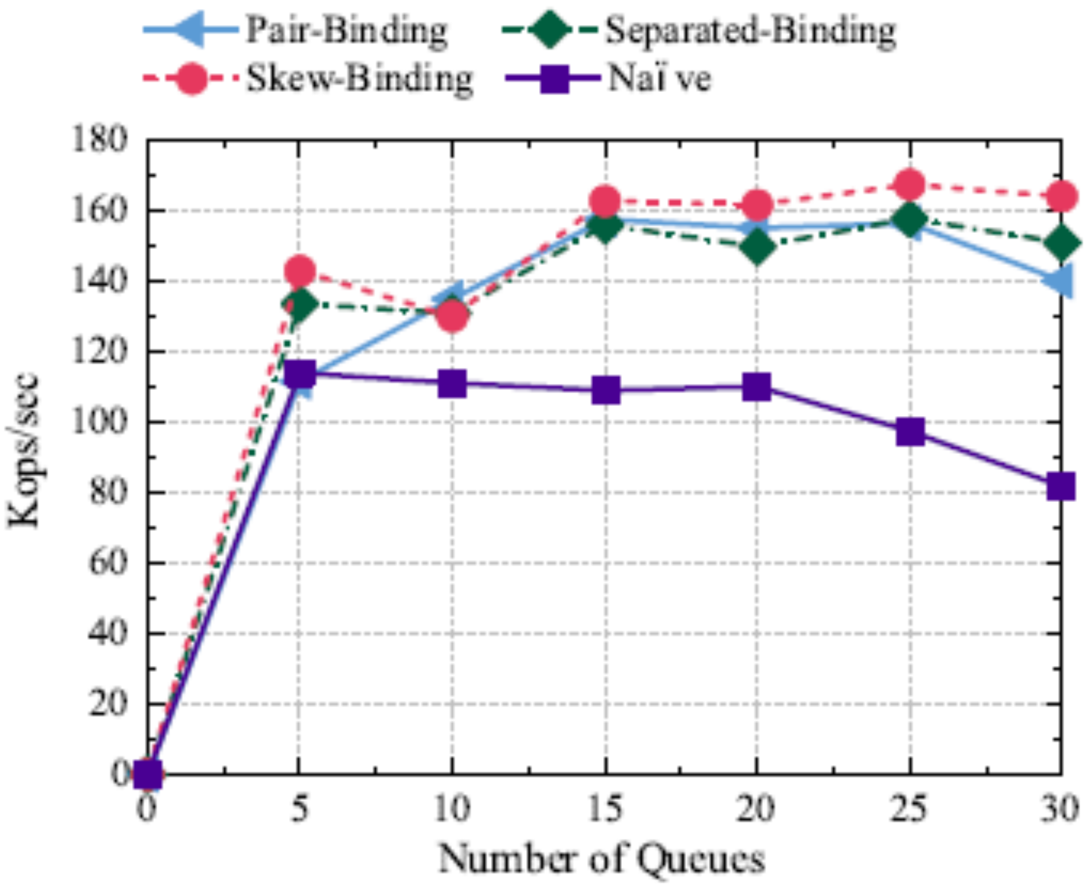


(b) I/O Operations.



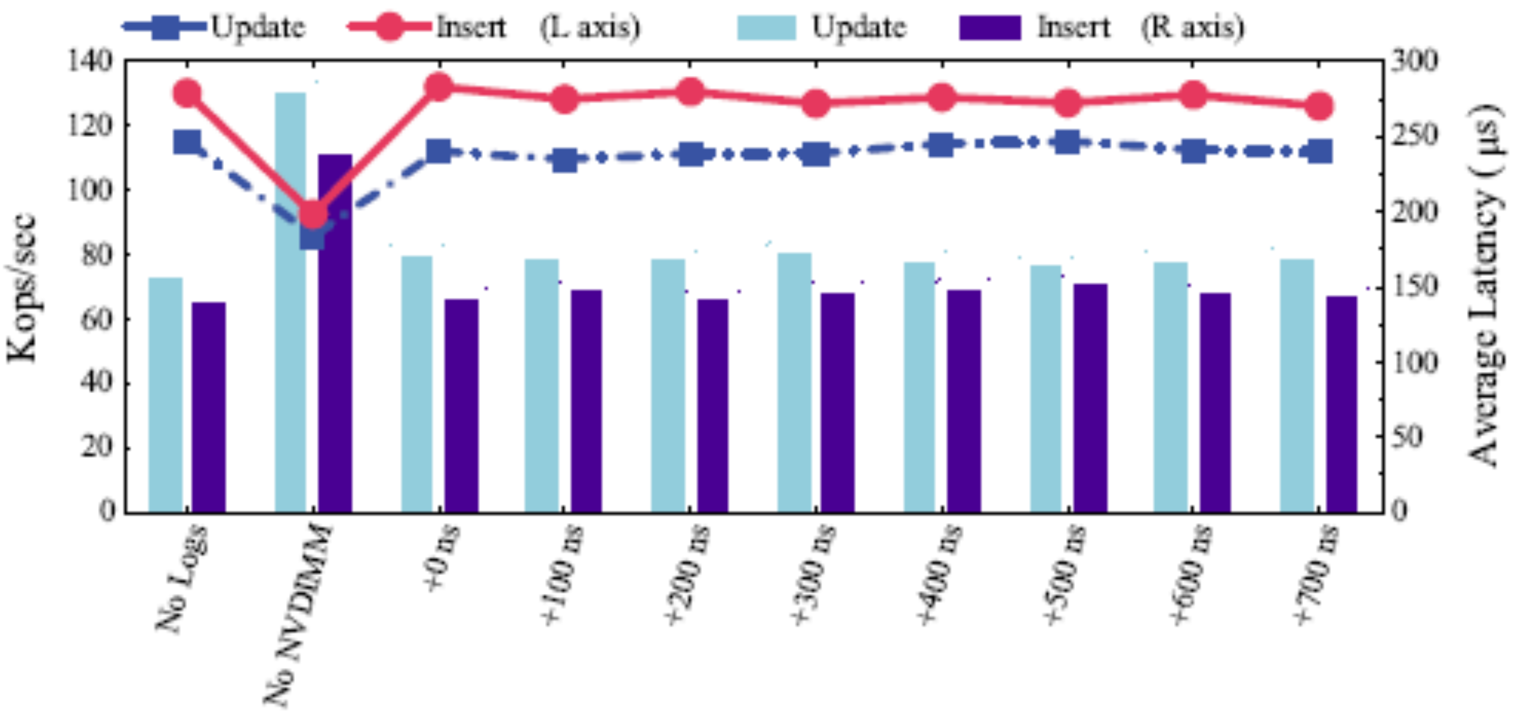
(c) Space Utilization.

Evaluation



- The Naïve mode does not fully combine the CPU cores and the I/O queues, thus introducing more thread migration and higher cache misses.
- The performance is highest in Skew-Binding mode because it binds all CQs to the same core, which mitigates the scramble for CPU resources so that more CPU resources are used to serve user requests.

Evaluation



- Different memory delays simulated different byte-addressable NVDIMM devices.
- No Logging. Writing without the undo logging method.
- No NVDIMM. Persisting undo logs into the NVMe SSD without resorting to the NVDIMM.

Conclusion

- **Lightweight Native NVMe-Based Storage Stack.** We propose a lightweight storage stack for NVMe devices. It integrates and simplifies the layers of the database, file system, and operating system to shorten the I/O path to achieve ultra-low latency.
- **Efficient Table Storage Engine.** Following the Lightstack, we build a table storage engine, LATTE, which first provides efficient data service in user space without data conversion between tables and files.
- **Parallel Scheduling for I/O Queues.** In LATTE, we combine multiple deep I/O queues and CPU cores to promote I/O parallelism.
- **Undo Logging on Heterogeneous Storage.** In LATTE, we also accelerate the undo logging with NVDIMM to mitigate the issue of write amplification without sacrificing performance and data consistency.

LATTE: A Native Table Engine on NVMe Storage

Jiajia Chu, Yunshan Tu, Yao Zhang, and Chuliang Weng*

East China Normal University

Thank you
Q&A

syslab@ECNU