Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs

Gyusun Lee†, Seokha Shin†, Wonsuk Song†, Tae Jun Ham§,

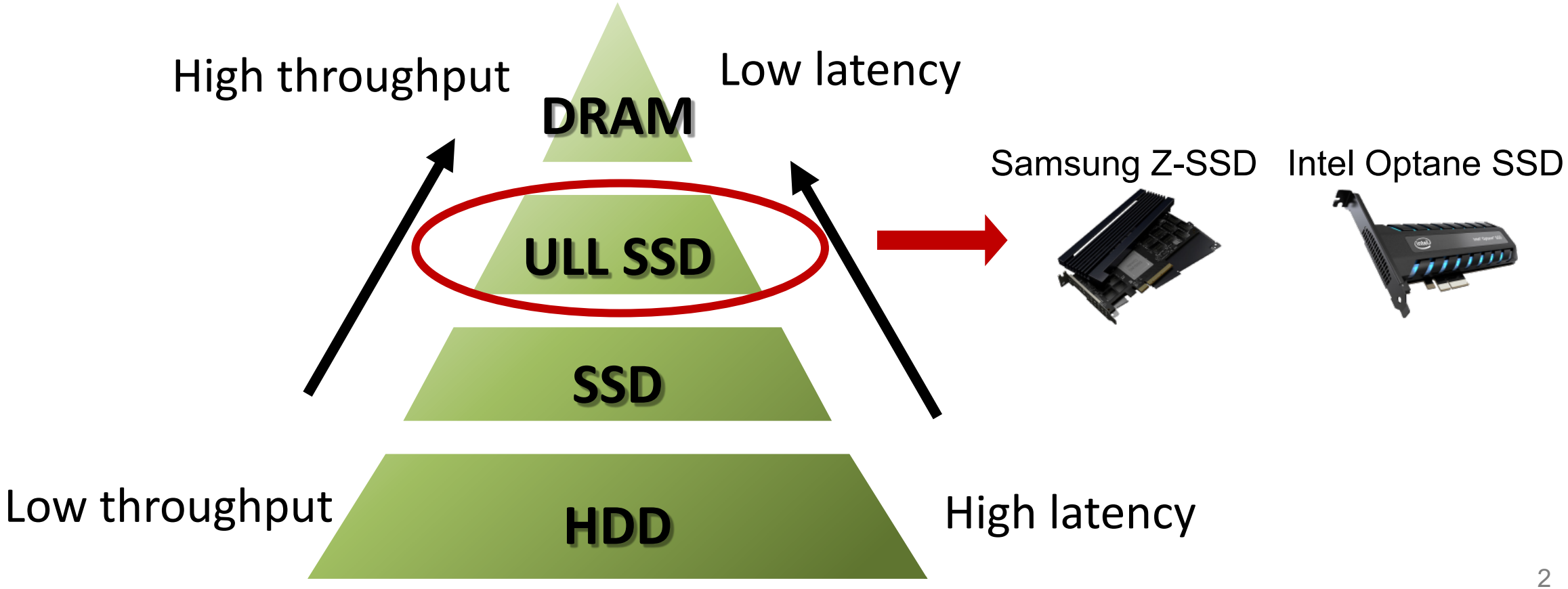
Jae W. Lee§, Jinkyu Jeong†

†Sungkyunkwan University, §Seoul National University

USENIX ATC 2019

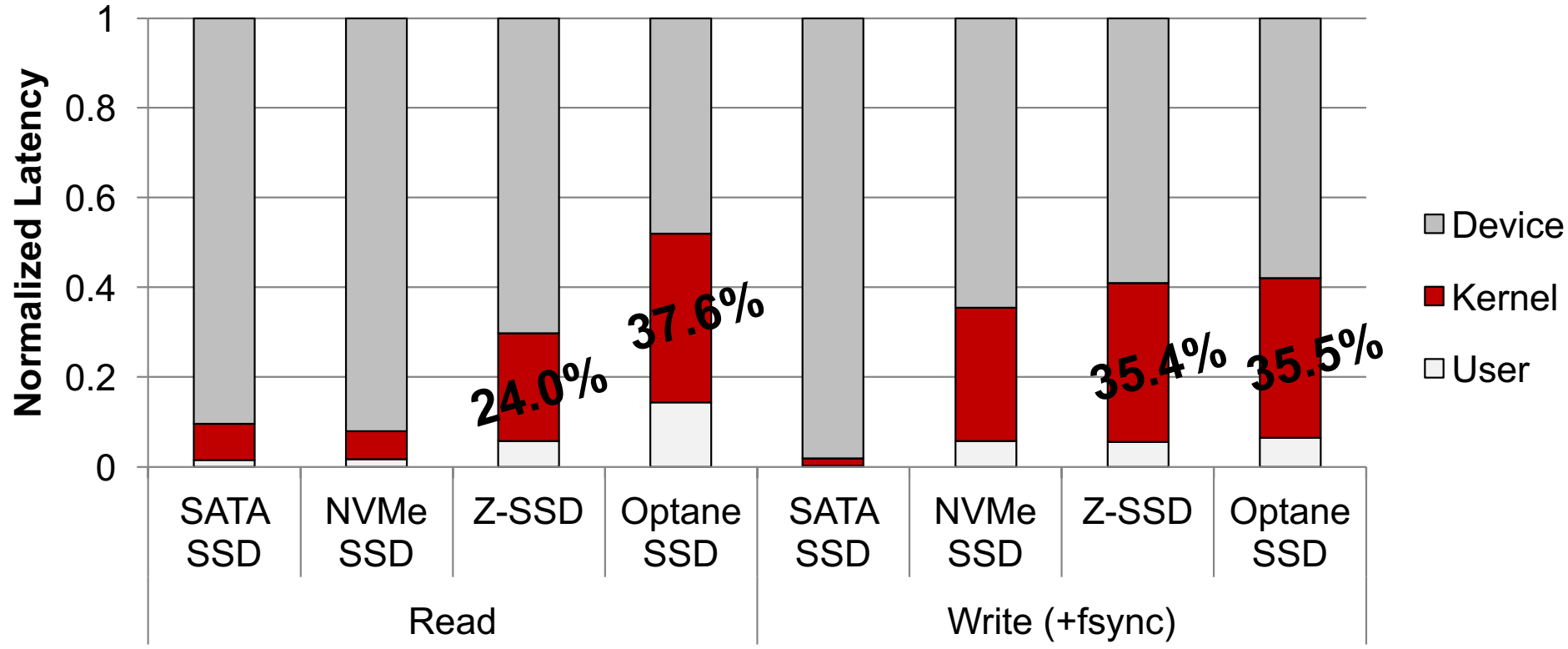
Background

- Ultra-low latency SSDs emerges

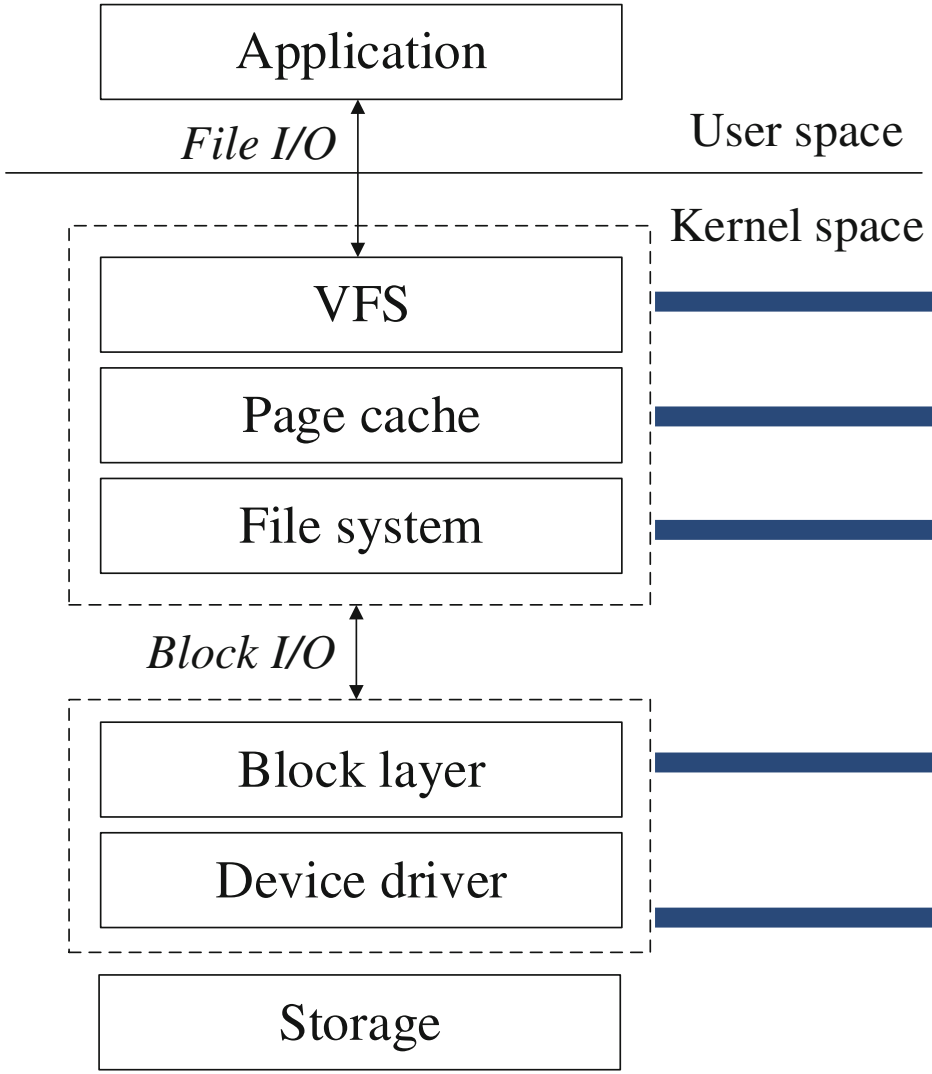


Background

- Kernel I/O stack accounts for a large fraction in total I/O latency, with ultra-low latency SSD

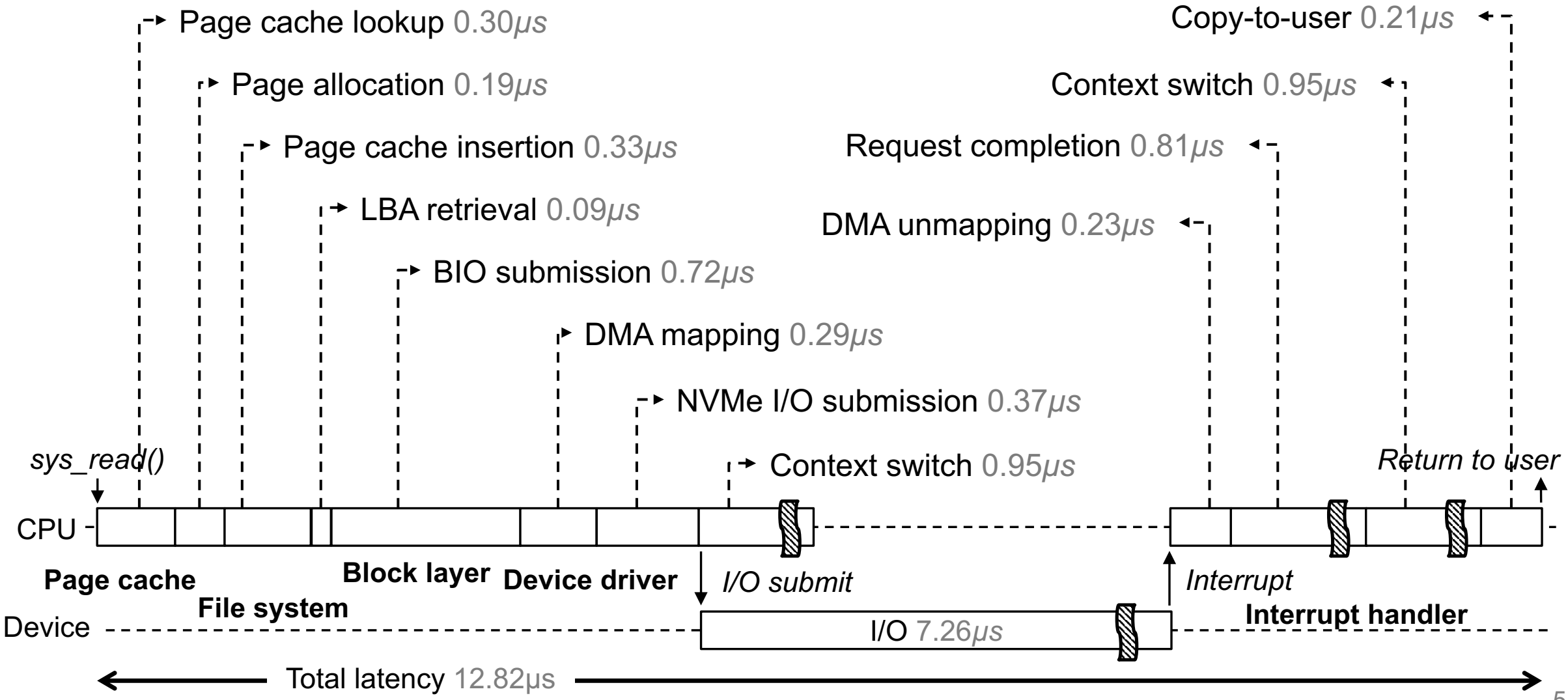


Background

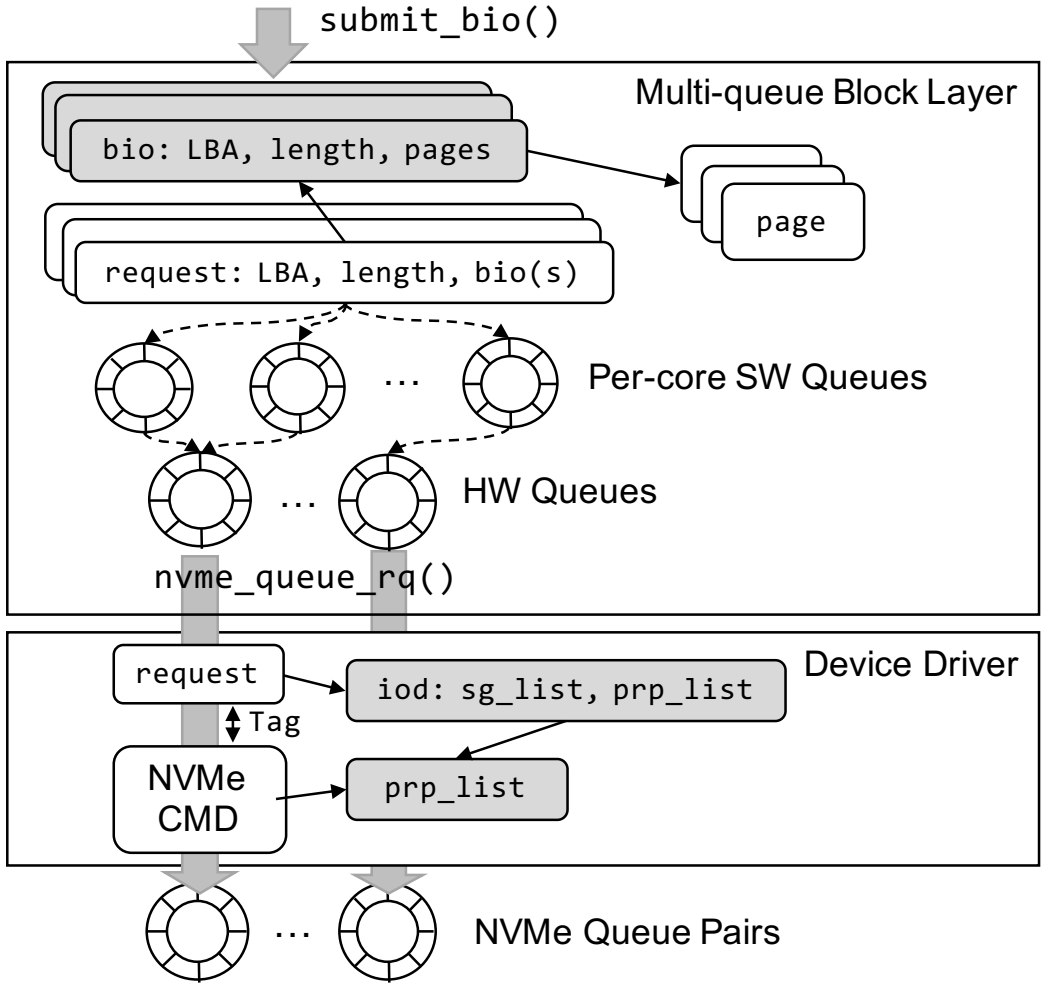


- The layers of I/O stack
  - VFS → Offer an abstraction of underlying file system
  - Page cache → Cache file data
  - File system → File system-specific implementations on top of the block storage
  - Block layer → OS-level block request/response management and block I/O scheduling
  - Device driver → Handle device-specific I/O command submission and completion

Analysis of Vanilla Read Path



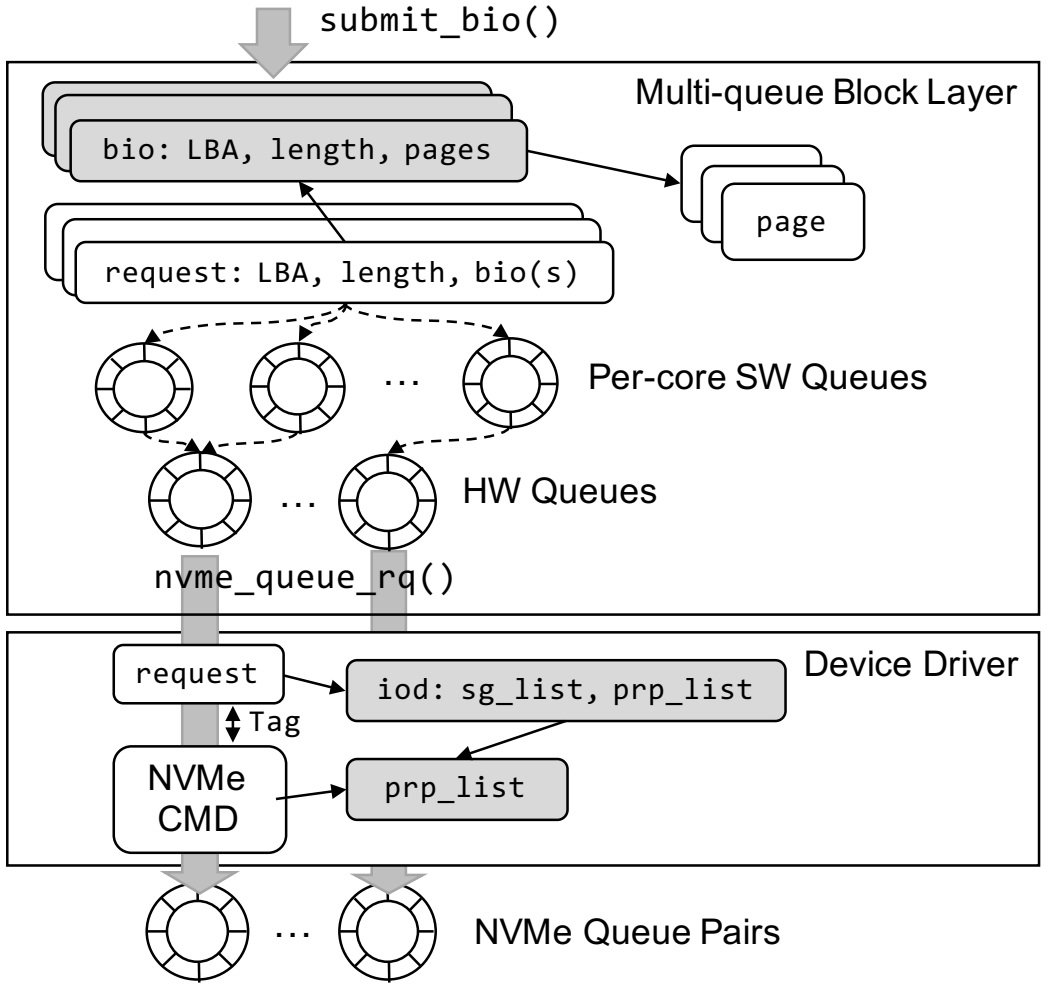
Linux Multi-queue Block Layer



- Block layer: OS-level block request/response management and block I/O scheduling
  - Merge bio with pending request via I/O merging
  - Assign new tag & request and convert from bio
- Multi-queue structure
  - Software Staging queue(SW queue)
    - ✓ Support I/O scheduling and reordering
  - Hardware dispatch queue (HW queue)
    - ✓ Deliver the I/O request to the device driver
- Dynamic memory allocations
  - bio(block layer)
  - iod, prp_list, sg_list(device layer)

- iod: used to perform DMA mapping
- Physical region pages(PRP): filled with allocated DMA addresses

Linux Multi-queue Block Layer



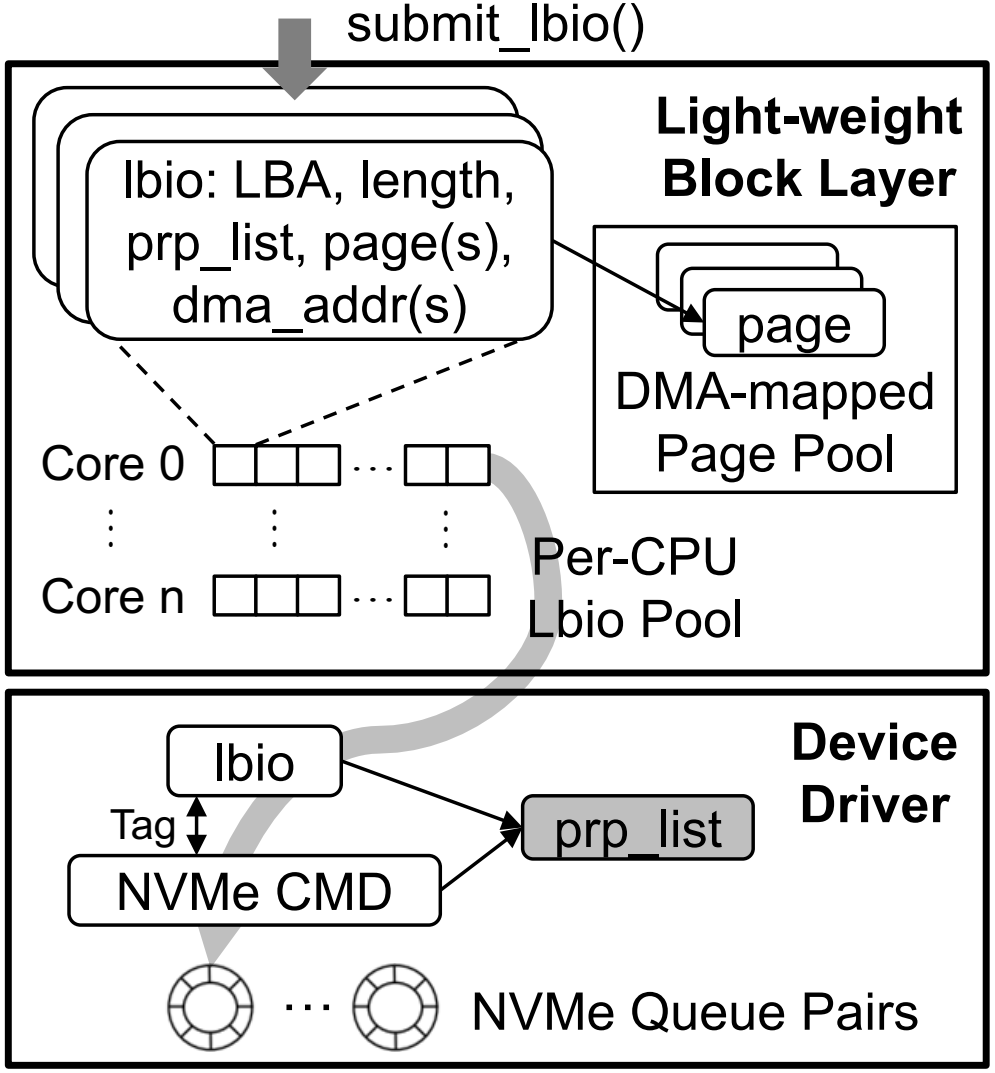
- Inefficiency of I/O merging for low-latency in block layer
- Inefficiency of multi-queue structure for low-latency
- Device-side I/O scheduling
- Heavy dynamic memory allocations
  - bio(block layer)
  - iod, prp_list, sg_list(device layer)

- iod: used to perform DMA mapping
- Physical region pages(PRP): filled with allocated DMA addresses

Motivation

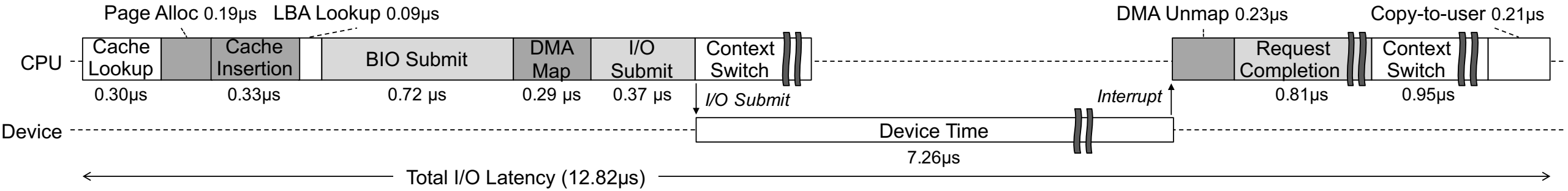
- **Ultra-low latency SSDs expose the overhead of kernel I/O stack (*up to 37.6%*)**
- **Inefficient and overweight multi-queue block layer**
- **Unexplored opportunities to further optimize the I/O latency**
  - **Synchronous I/O operations in vanilla kernel**
  - **Overlapping the synchronous I/O operation in the critical path**

Light-weight Block Layer

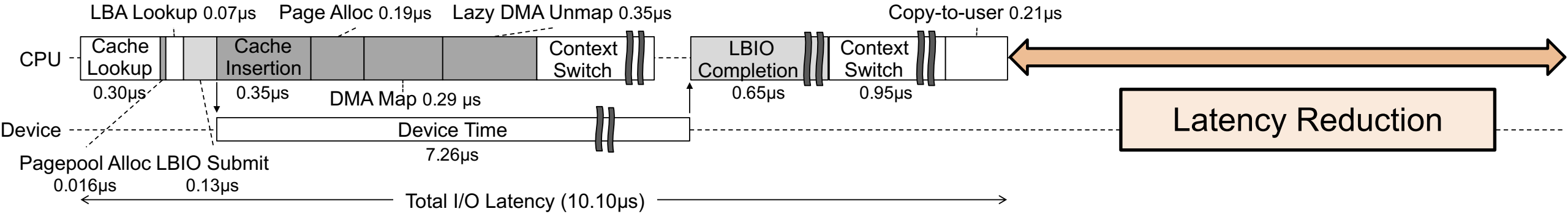


- Light-weight bio(*lbio*)structure
  - Eliminate unnecessary structure conversions and allocations
  - Enable the asynchronous DMA mapping feature with DMA address
- DMA-mapped page pool
  - A linked list of 4 KB DMA-mapped pages for each core
  - Hide time for page allocation and DMA mapping time
- Per-CPU lbio pool
  - Lockless lbio object allocation
  - Support tagging function(the index of lbio in the global array)
- Dynamic memory allocation
  - NVMe prp_list(device driver)

Proposed Read Path

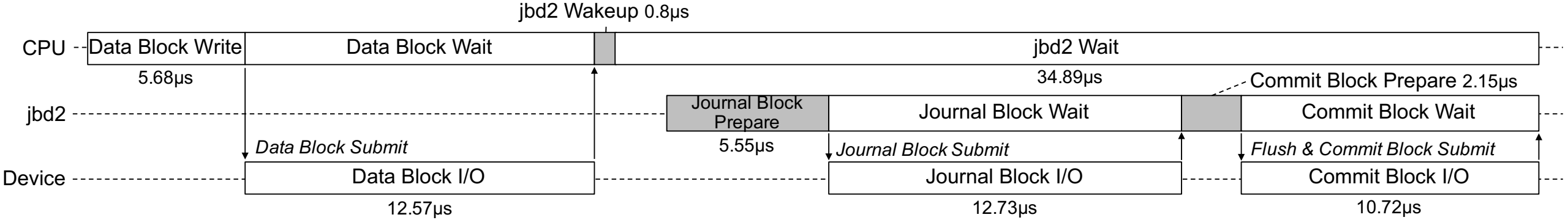


(a) Vanilla read path



(b) Proposed read path

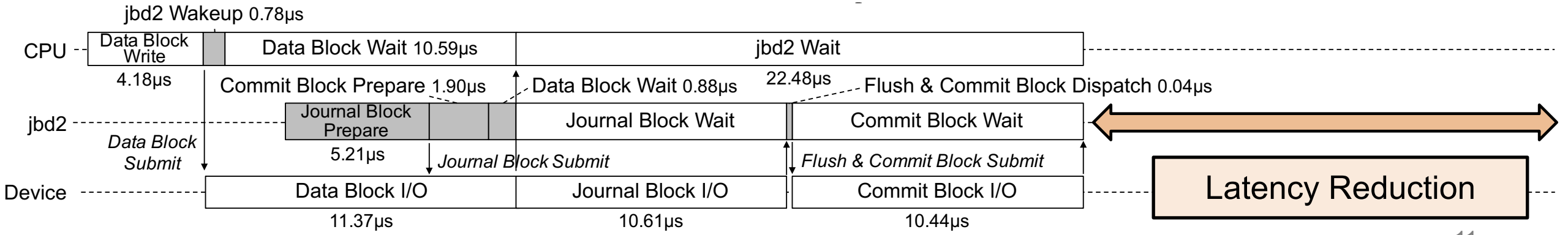
Proposed Write Path



(a) Vanilla `fsync` path

Write in memory + fsync
Jbd2: journal block device thread

Journal Block Prepare:
- Allocating buffer pages
- Allocating journal area block
- Checksum computation



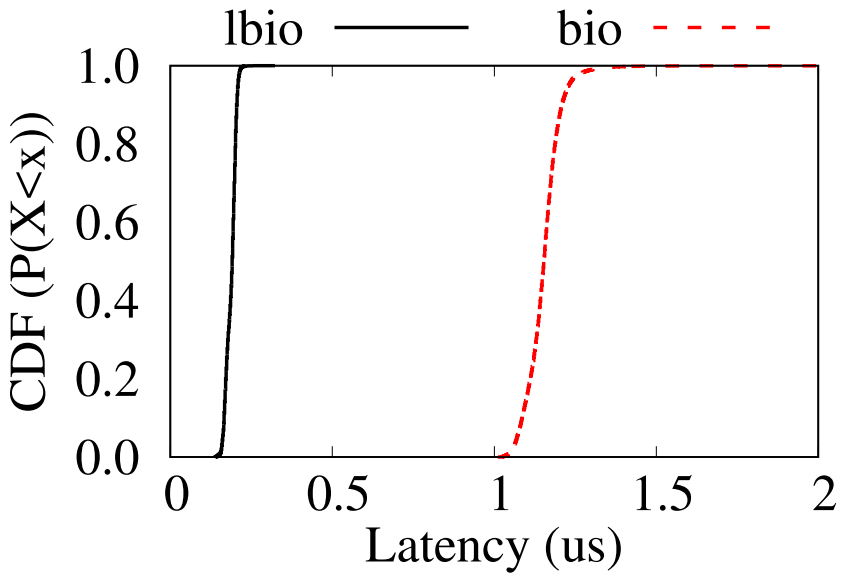
(b) Proposed `fsync` path

Experimental Setup

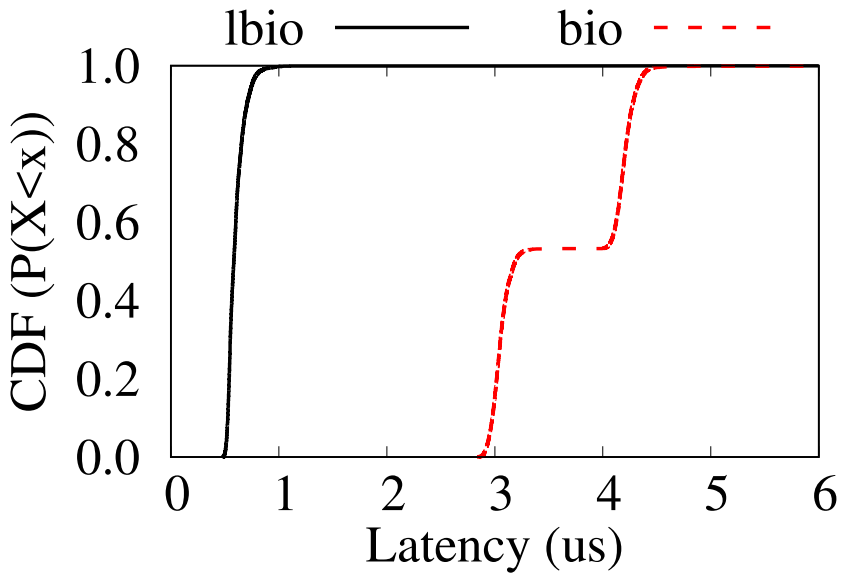
Server	Dell R730
OS	Ubuntu 16.04.4
Base kernel	Linux 5.0.5
CPU	Intel Xeon E5-2640v3 2.6GHz 8-cores
Memory	DDR4 32GB
Storage devices	Z-SSD: Samsung SZ985 800GB Optane SSD: Intel Optane 905P 960GB
Workloads	Synthetic micro-benchmark: FIO Real-world workload: RocksDB DBbench

Latency Saving

Note: Time from allocating a (l)*bio* object to dispatching an I/O command to a device on Optane SSD



(a) 4 KB random read

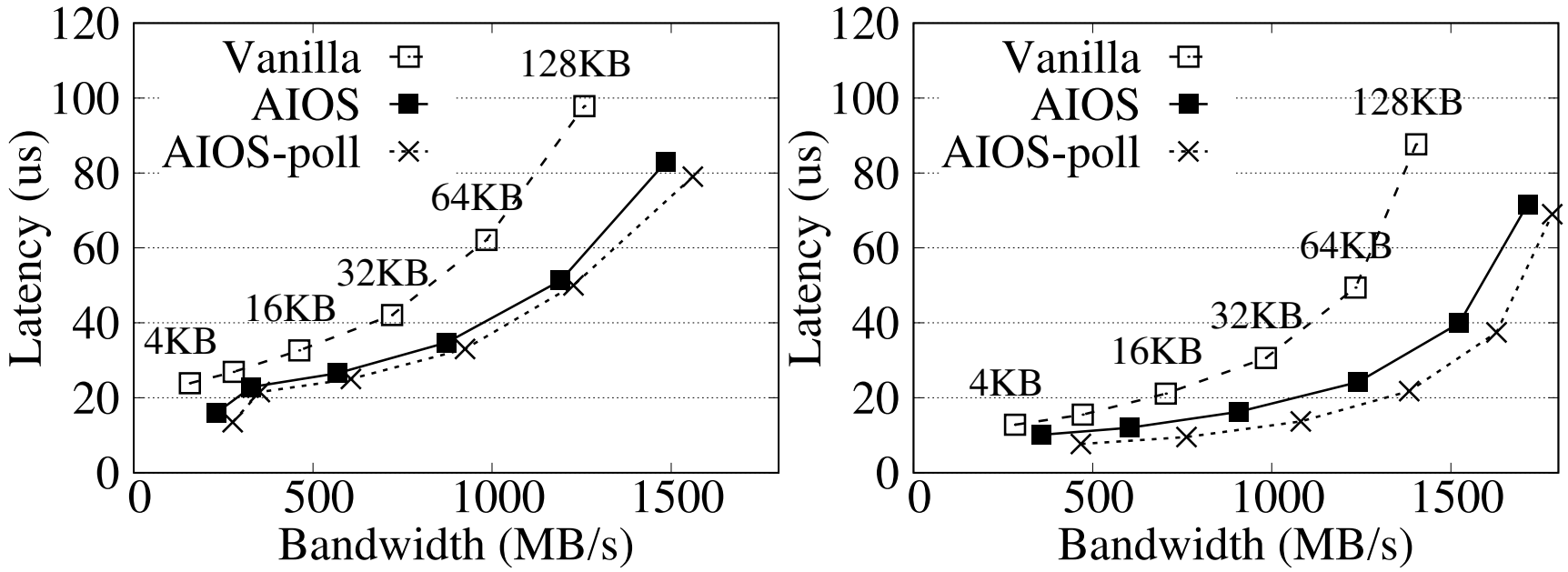


(b) 32 KB random read

- 4 KB → minimum dynamic memory allocations
- 32 KB → maximum dynamic memory allocations

- On average, a block I/O request in *LBIO* is *83.4%–84.4%* shorter latency compared with *bio*

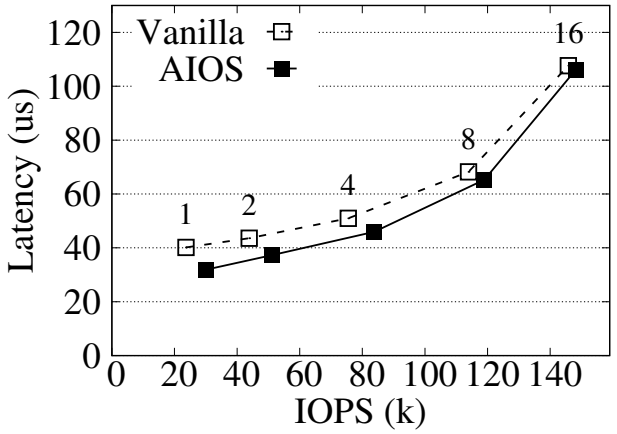
FIO Random Read Latency



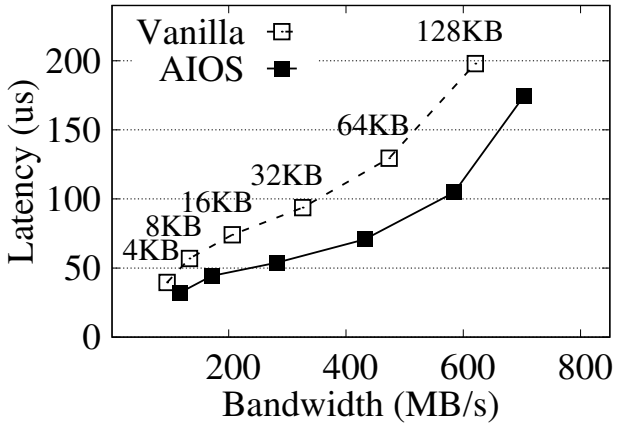
(a) Z-SSD (b) Optane SSD

- AIOS achieves latency reduction about *15-33%* compared with Vanilla
- Larger I/O block size gets more latency reduction because larger portion of read-related kernel computation gets overlapped

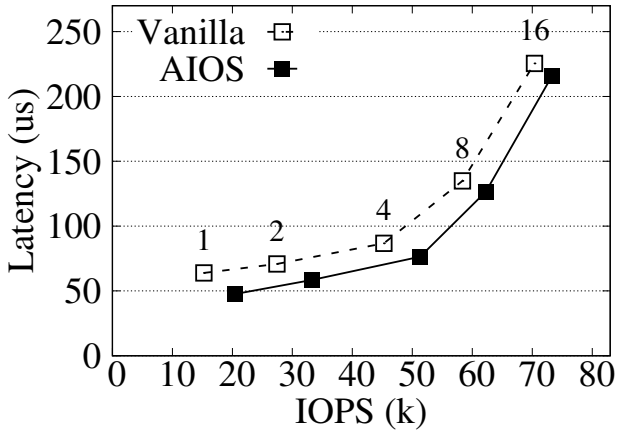
FIO Random Write Latency



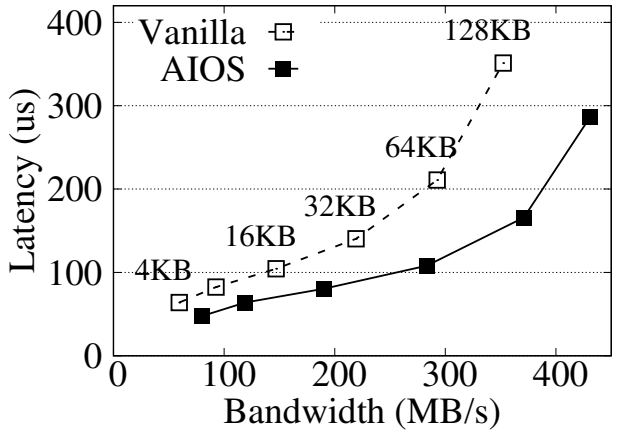
(a) Ordered, # of threads



(b) Ordered, block sizes



(c) Data-journaling, # of threads



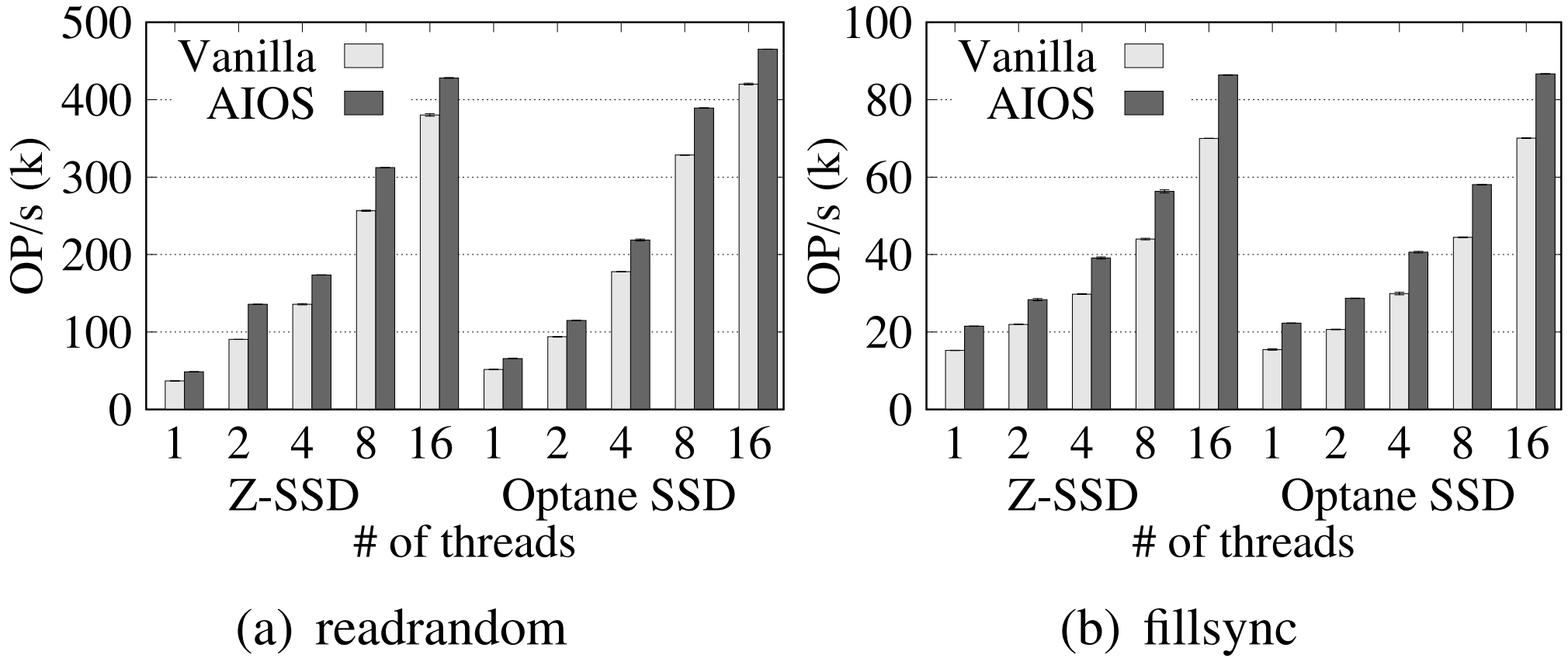
(d) Data-journaling, block sizes

Two journaling mode

- Ordered-mode
  - ✓ All data are directly out to file system prior to its metadata being committed to the journal
- Data-journaling mode
  - ✓ All data and metadata are written to the journal before being written to disk

- Latency reduction is about *21% and 26%* in two journaling mode
- Improvement by up to *27% and 34%*

RocksDB DBbench Performance



- Speedup on the *readrandom* workload by *11– 32%* and the *fillsync* workload by *22–44%*

Conclusion

- Light-weight block I/O layer
  - Eliminate unnecessary components to minimize the delay
  - Provide low-latency block I/O services for low-latency SSDs
- Asynchronous I/O stack
  - Replaces synchronous operations in the I/O path with asynchronous ones to overlap computation
  - Achieves single-digit microseconds I/O latency on Optane SSD
- Source code: https://github.com/skkucsl/aios