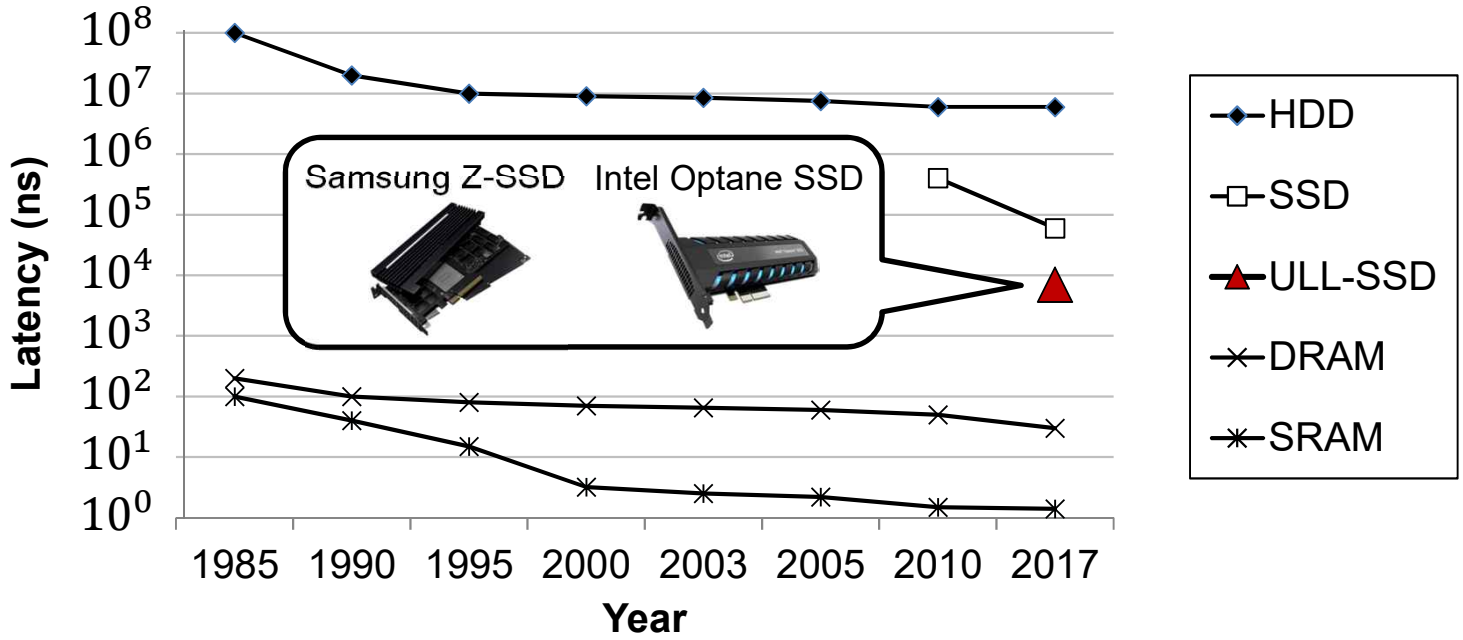
Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs

Jinkyu Jeong
Sungkyunkwan University (SKKU)

Source: Gyusun Lee et al., Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs, USENIX ATC 19

Storage Performance Trends

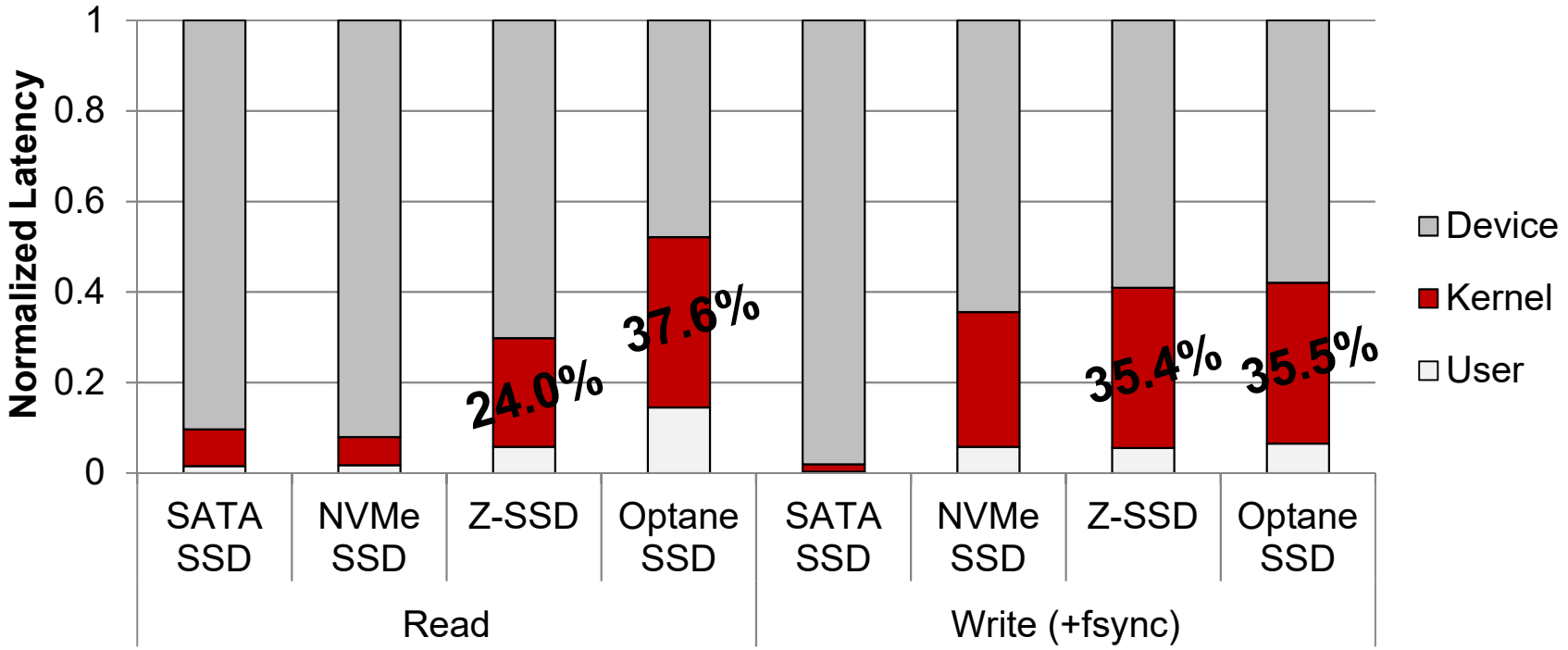
- **Emerging ultra-low latency SSDs deliver I/Os in <u>a few *µs*</u>**



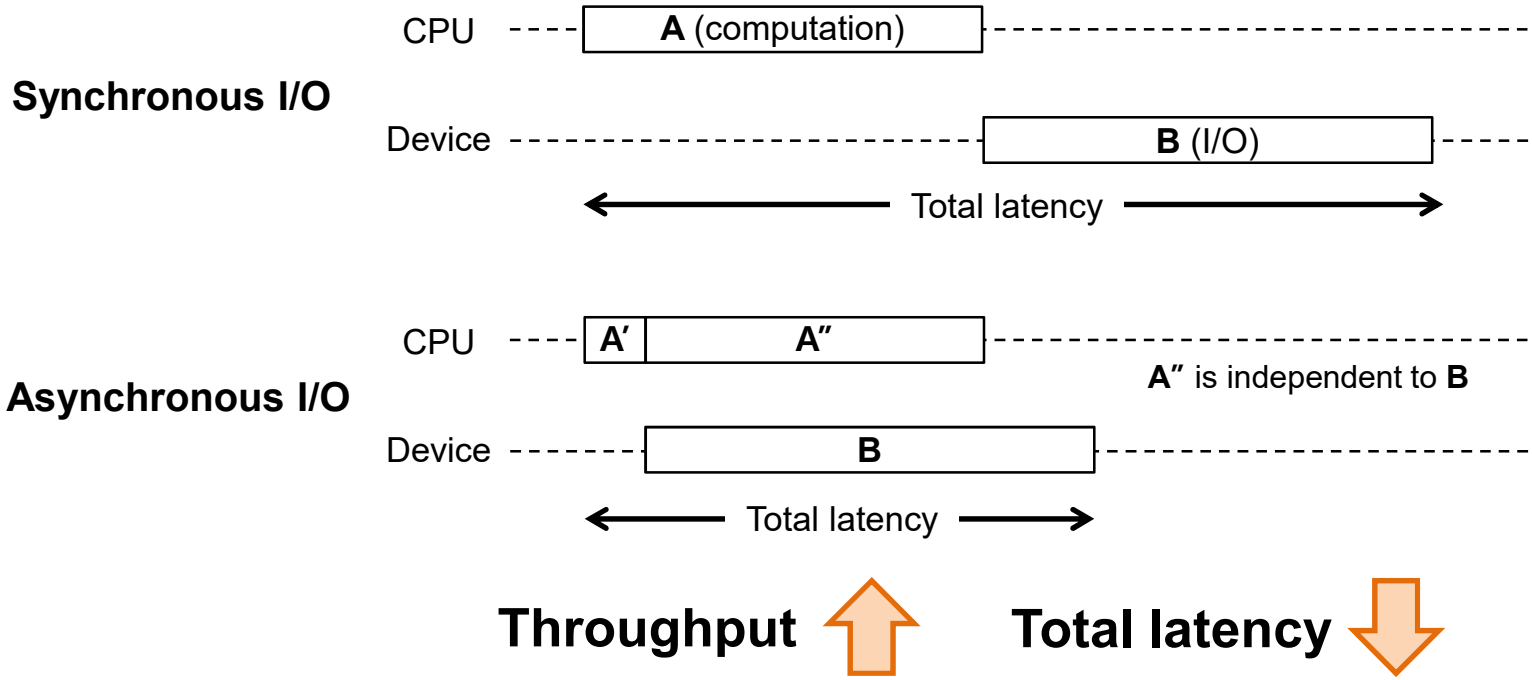
Source: R. E. Bryant and D. R. O'Hallaron, Computer Systems: A Programmer's Perspective, Second Edition, Pearson Education, Inc., 2015

Overhead of Kernel I/O Stack

- **Low-latency SSDs expose the overhead of kernel I/O stack**

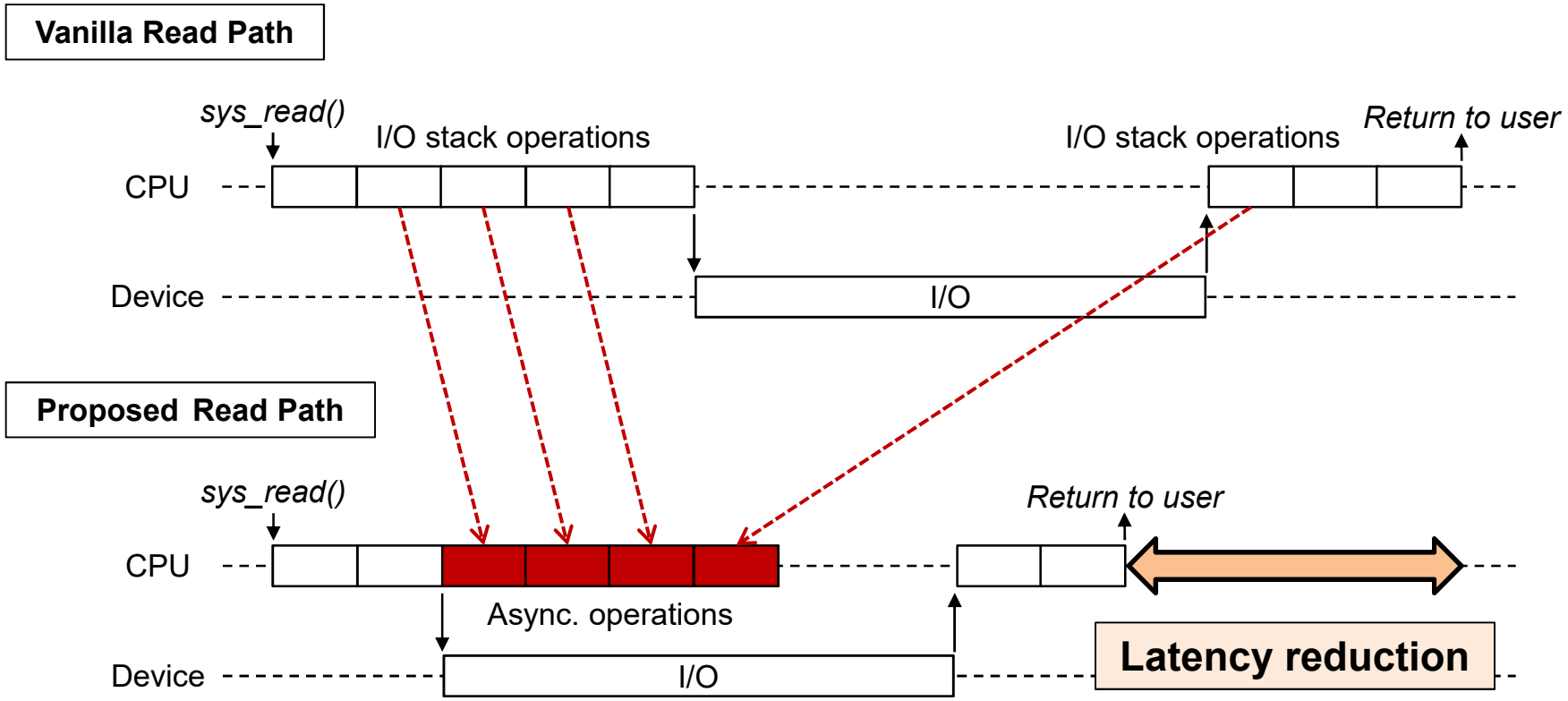


Synchronous I/O vs. Asynchronous I/O



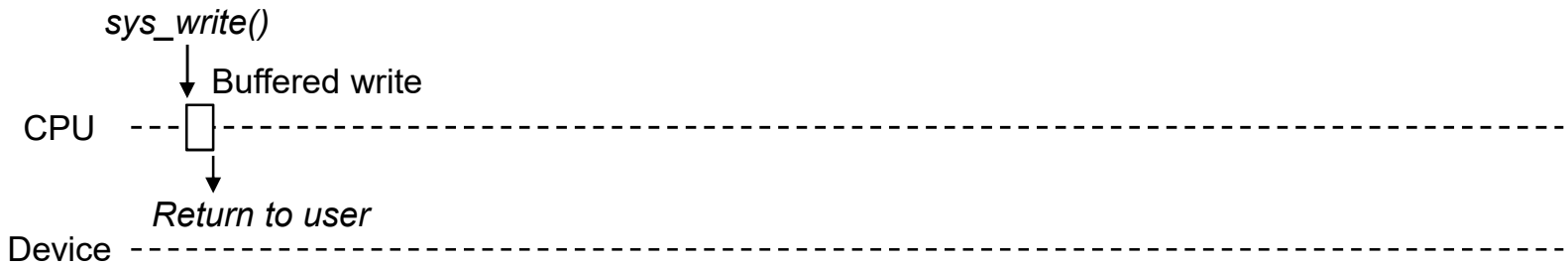
Our Idea: apply asynchronous I/O concept to the I/O stack itself

Read Path Overview

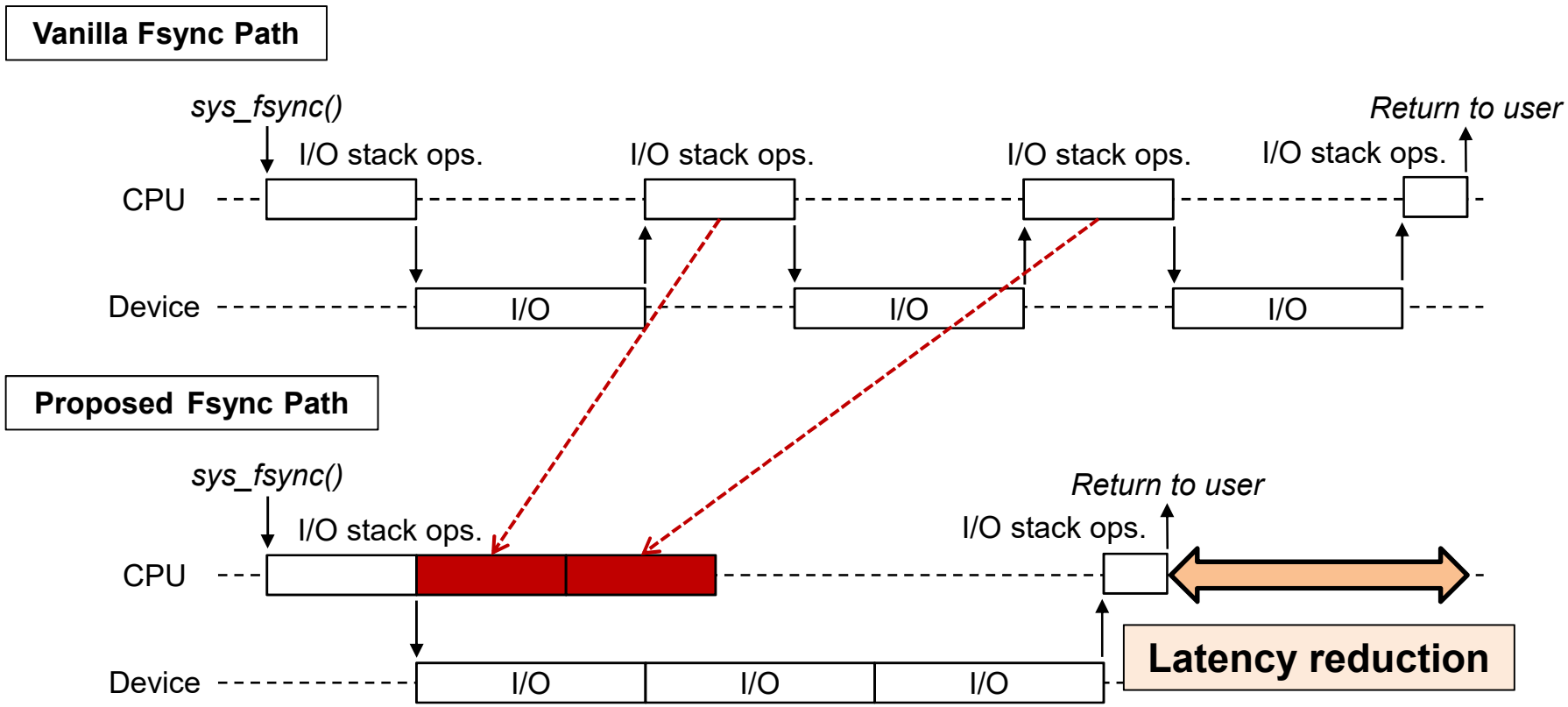


Write Path Overview

Vanilla Write Path



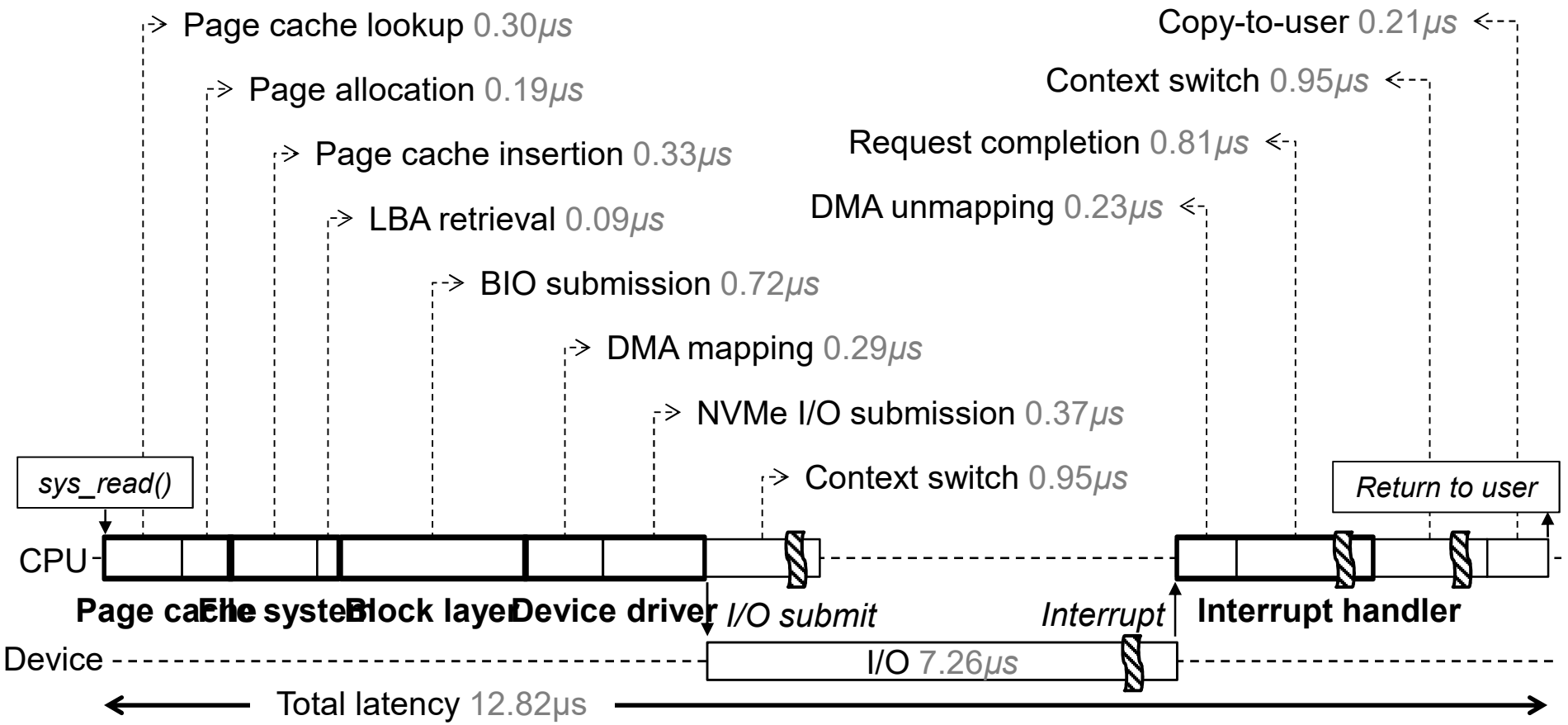
Write Path Overview



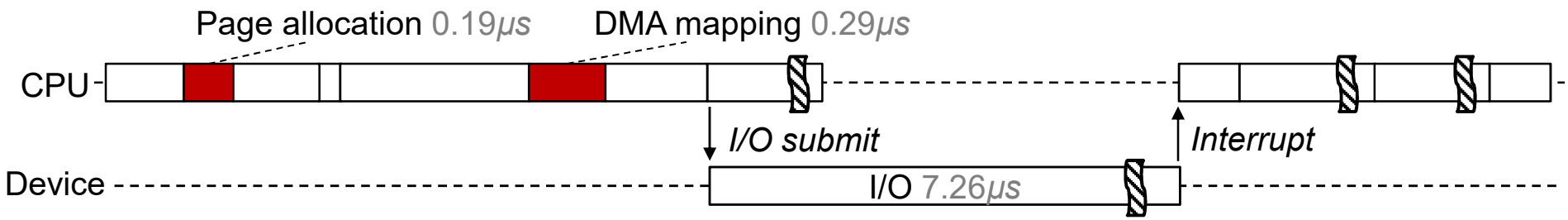
Agenda

- **Read path**
  - Analysis of vanilla read path
  - Proposed read path
- **Light-weight block I/O layer**
- **Write path**
  - Analysis of vanilla write path
  - Proposed write path
- **Evaluation**
- **Conclusion**

Analysis of Vanilla Read Path

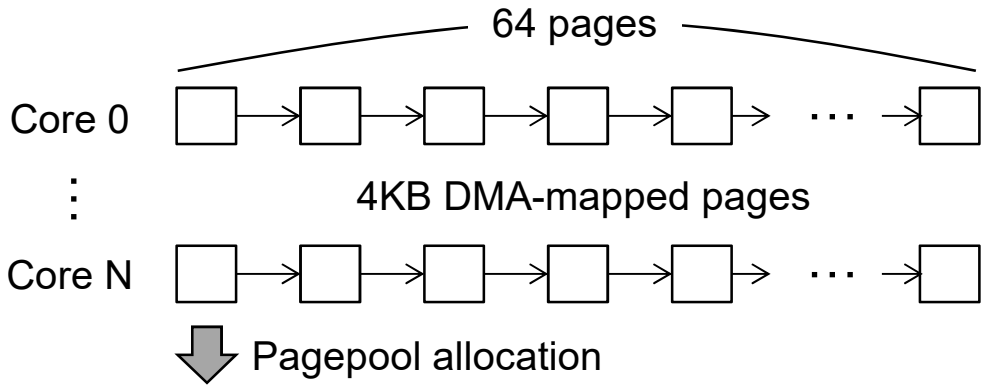


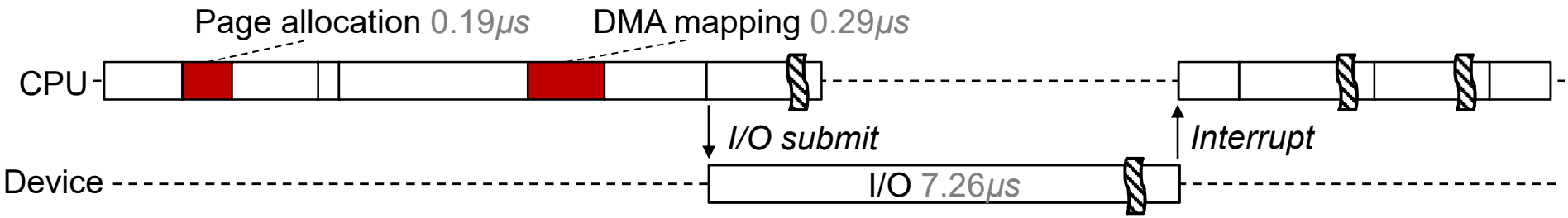
Page Allocation / DMA Mapping



Asynchronous Page Allocation / DMA Mapping

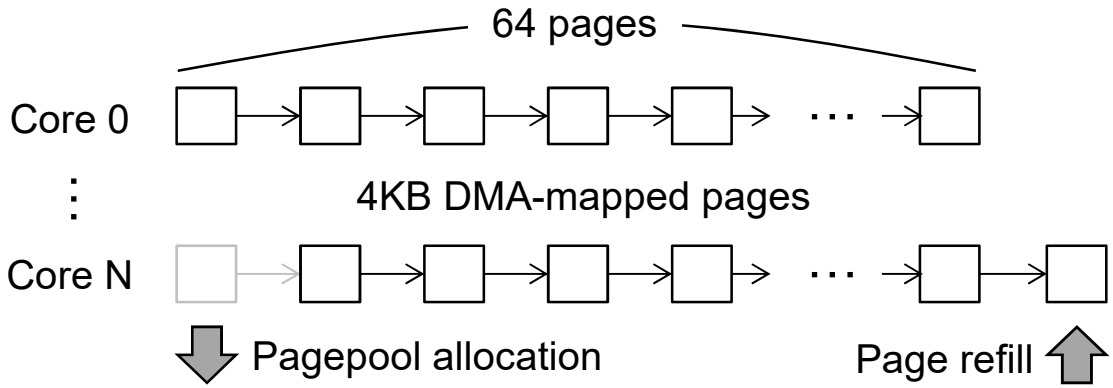
- **DMA-mapped page pool**

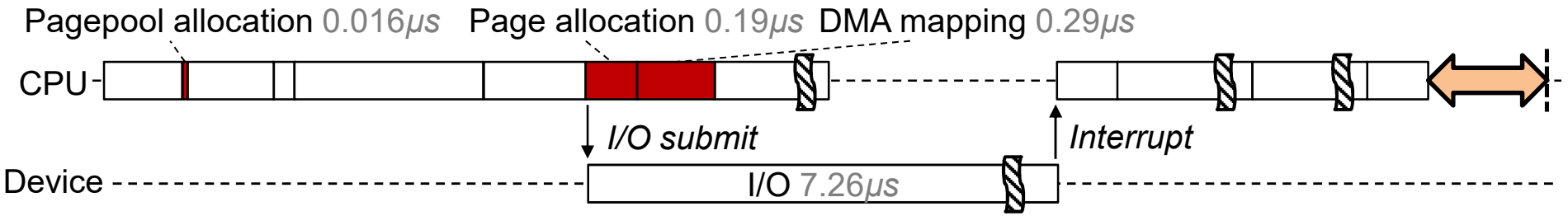




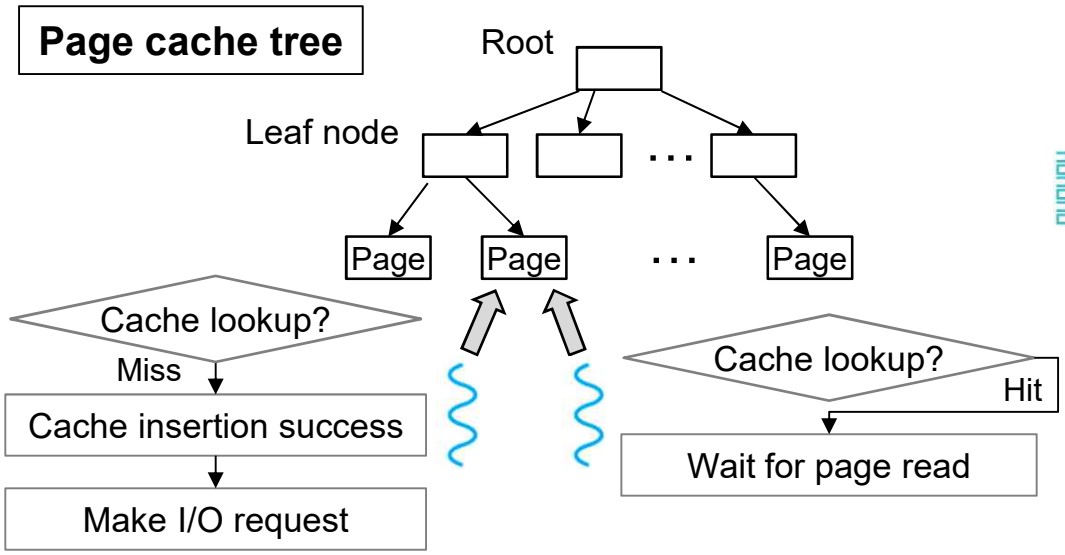
Asynchronous Page Allocation / DMA Mapping

- **DMA-mapped page pool**





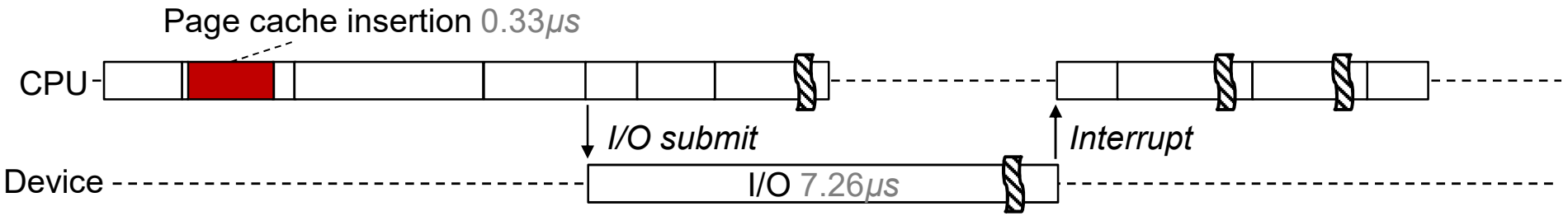
Page Cache Insertion



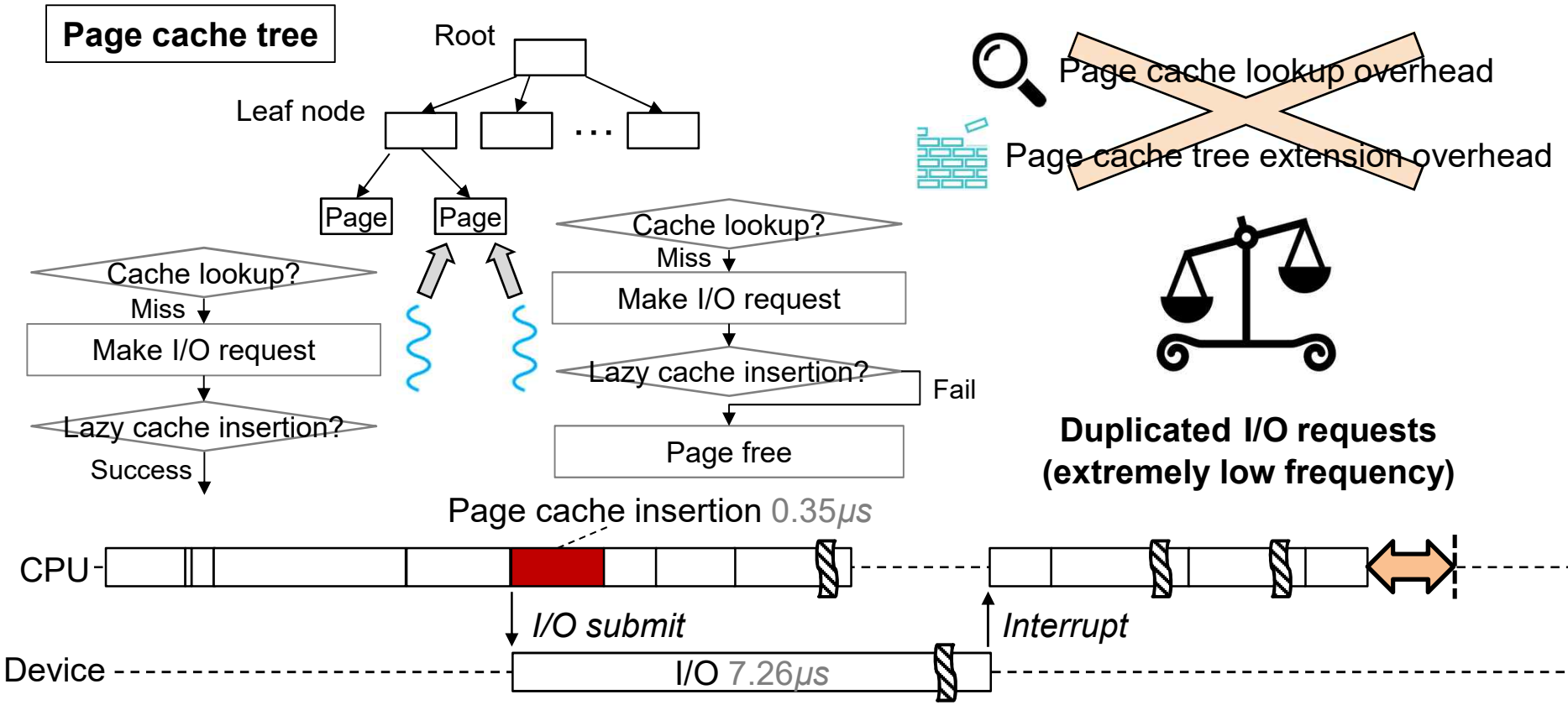
Page cache lookup overhead

Page cache tree extension overhead

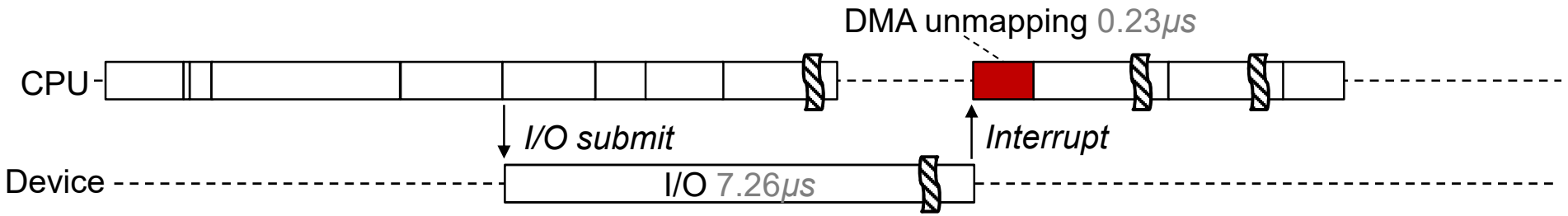
Prevention from duplicated I/O requests for the same file index



Lazy Page Cache Insertion

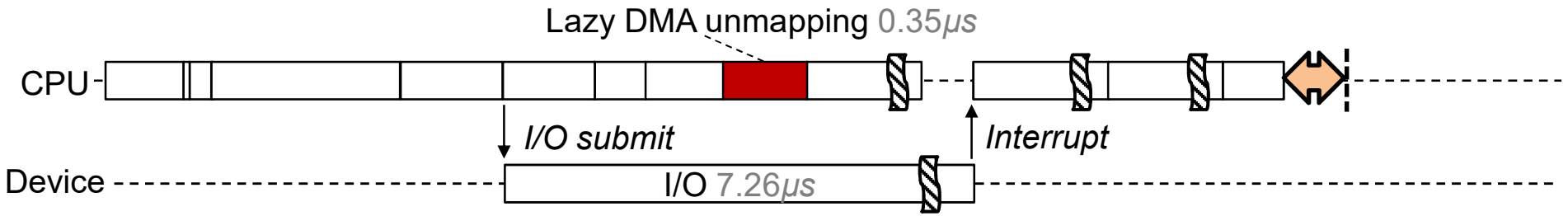


DMA Unmapping

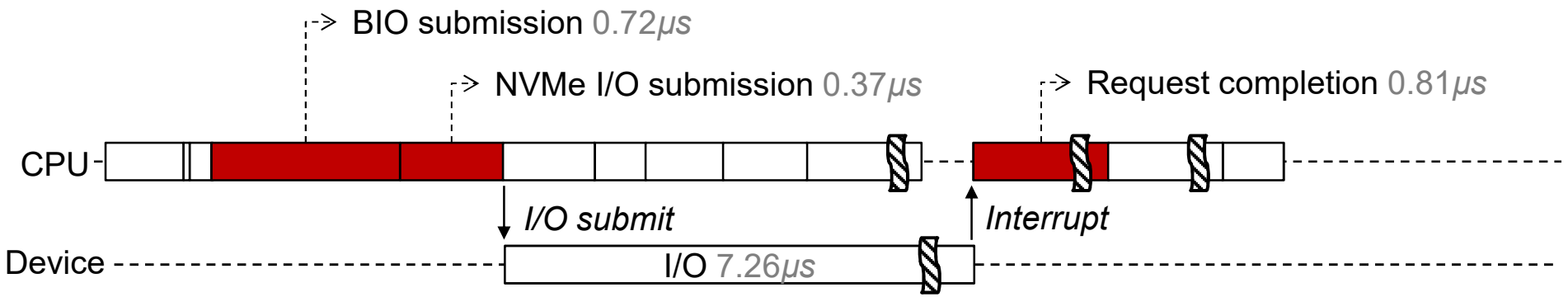


Lazy DMA Unmapping

- **Implementation**
  - Delays DMA unmapping to when a system is idle or waiting for another I/O requests
  - Extended version of the deferred protection scheme in Linux [*ASPLOS'16*]
  - Optionally disabled for safety



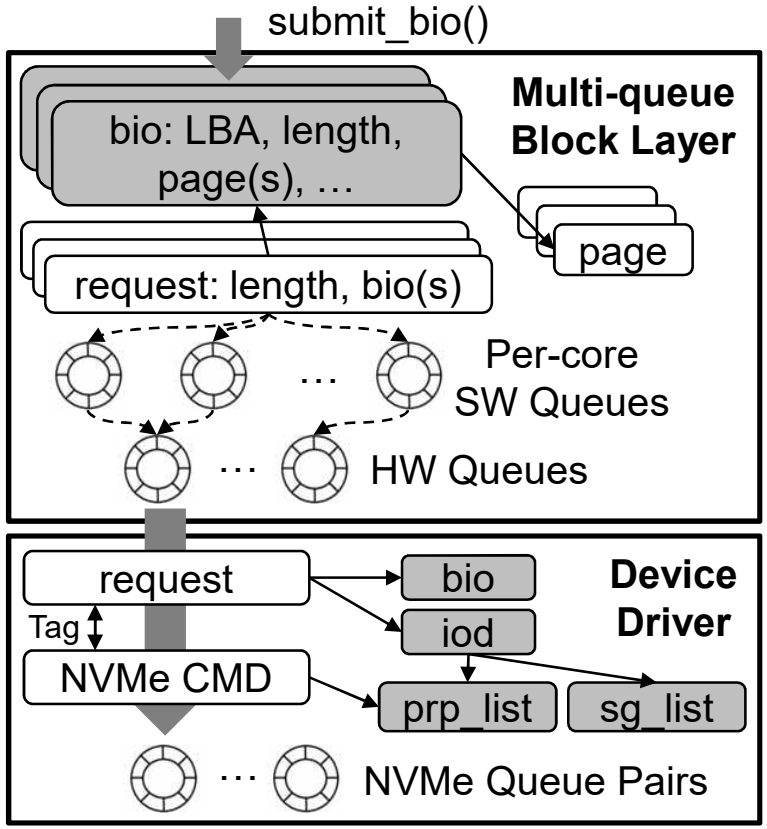
Remaining Overheads in the Proposed Read Path



Agenda

- Read path
  - Analysis of vanilla read path
  - Proposed read path
- **Light-weight block I/O layer**
- Write path
  - Analysis of vanilla write path
  - Proposed write path
- Evaluation
- Conclusion

Linux Multi-queue Block I/O Layer

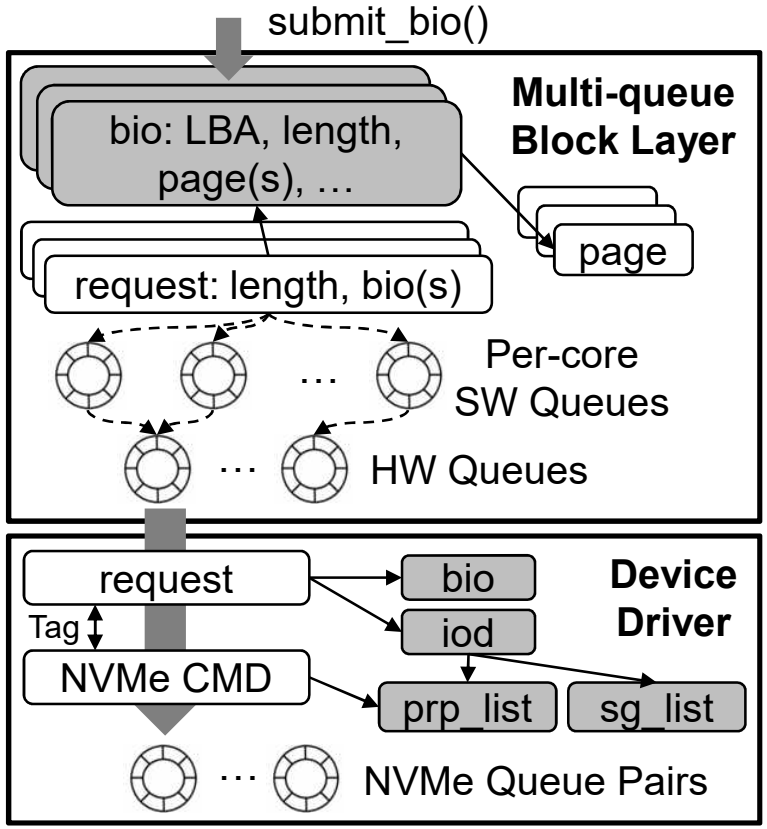


Linux multi-queue block layer

- **Structure conversion**
  - 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
- **Multiple dynamic memory allocations**
  - Bio (block layer)
  - NVMe iod, scatter/gather list, NVMe PRP* list (device driver)

*PRP: physical region page

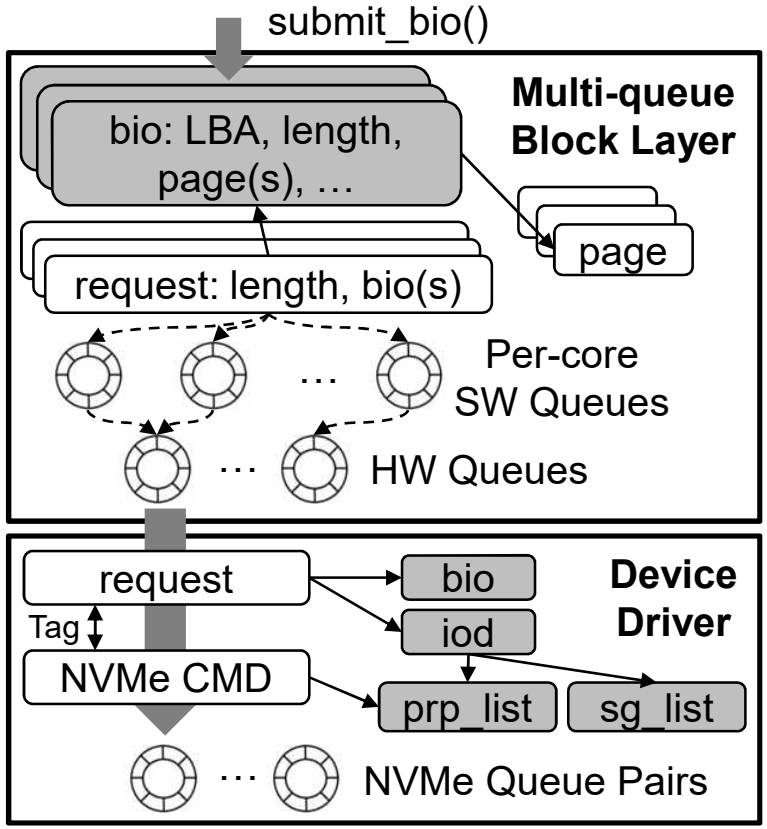
Linux Multi-queue Block I/O Layer



Linux multi-queue block layer

- **Structure conversion**
  - Inefficiency of I/O merging [Zhang,*OSDI'18*]
    - ✓ Useful feature for low-performance storage device
- **Multi-queue structure**
- **Multiple dynamic memory allocations**

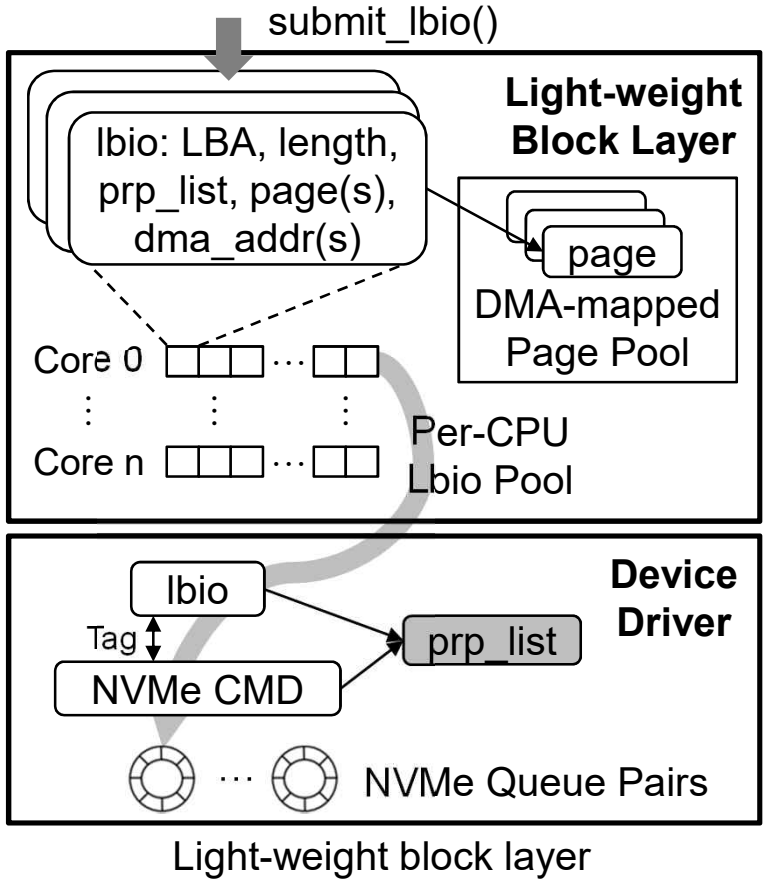
Linux Multi-queue Block I/O Layer



Linux multi-queue block layer

- **Structure conversion**
  - Inefficiency of I/O merging [Zhang,*OSDI'18*]
    - ✓ Useful feature for low-performance storage device
- **Multi-queue structure**
  - Inefficiency of I/O scheduling for low-latency SSDs [Saxena,*ATC'10*] [Xu,*SYSTOR'15*]
    - ✓ Default configuration is noop scheduler
  - Bypass multi-queue structure [Zhang,*OSDI'18*]
  - Device-side I/O scheduling [Peter,*OSDI'14*] [Joshi,*HotStorage'17*]
- **Multiple dynamic memory allocations**

Light-weight Block I/O Layer

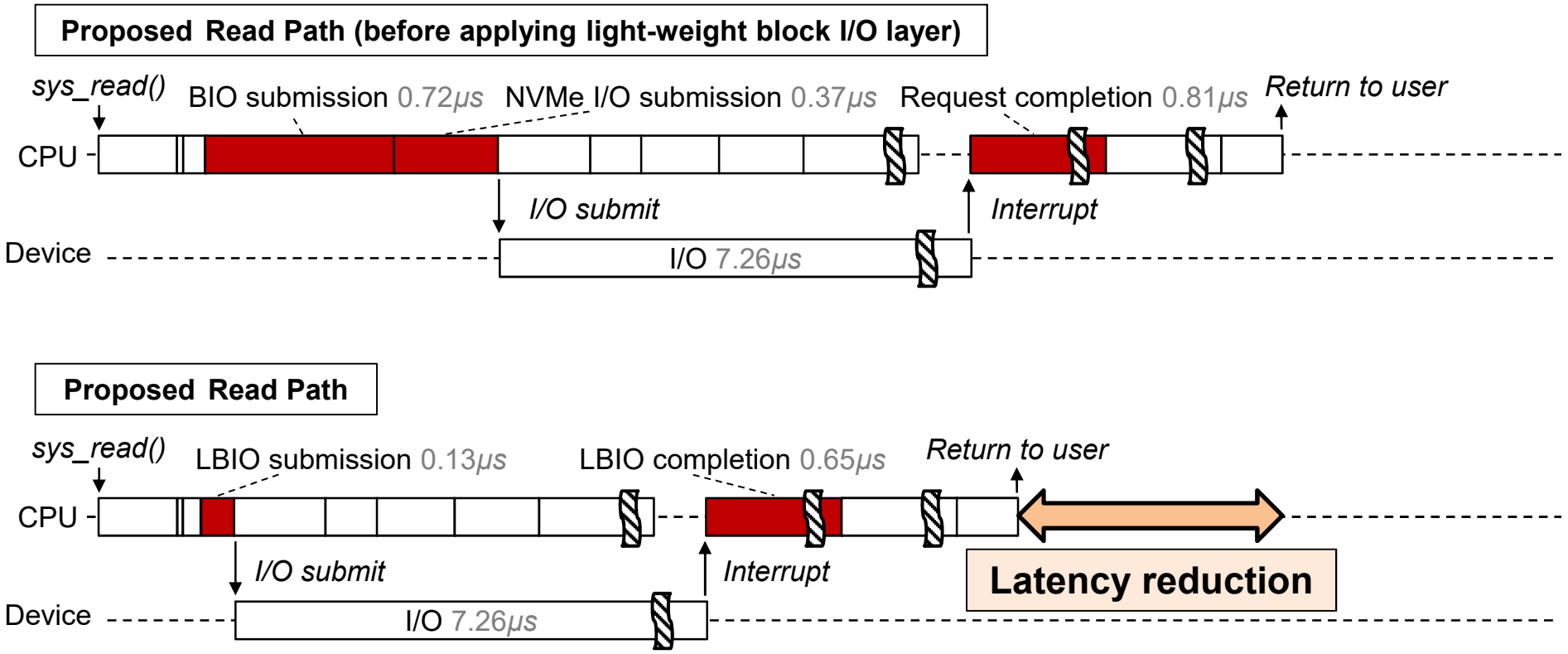


Light-weight block layer

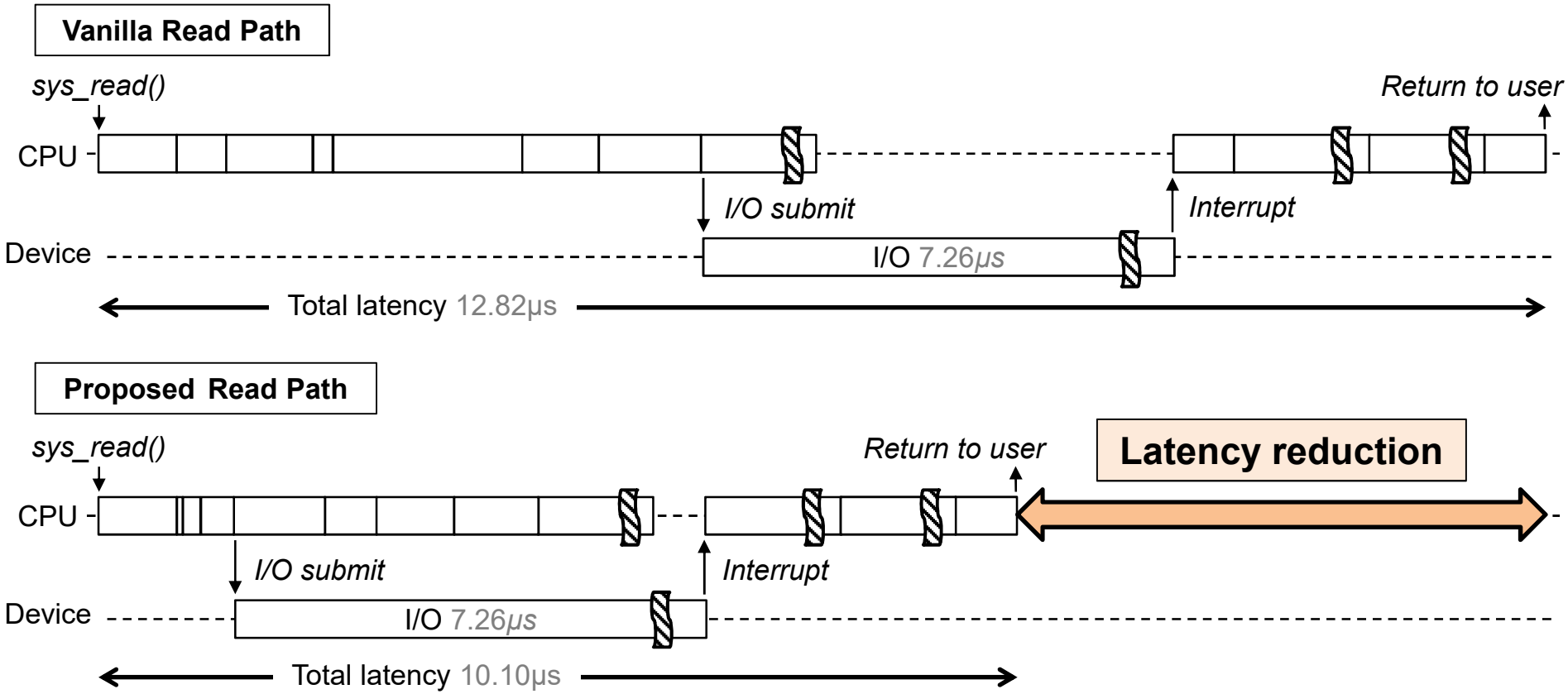
- **Light-weight bio (lbio) structure**
  - Contains only essential arguments for to make NVMe I/O request
  - Eliminates unnecessary structure conversions and allocations
- **Per-CPU lbio pool**
  - Supports lockless lbio object allocation
  - Supports tagging function
- **Single dynamic memory allocation**
  - NVMe PRP* list (device driver)

*PRP: physical region page

Read Path Comparison



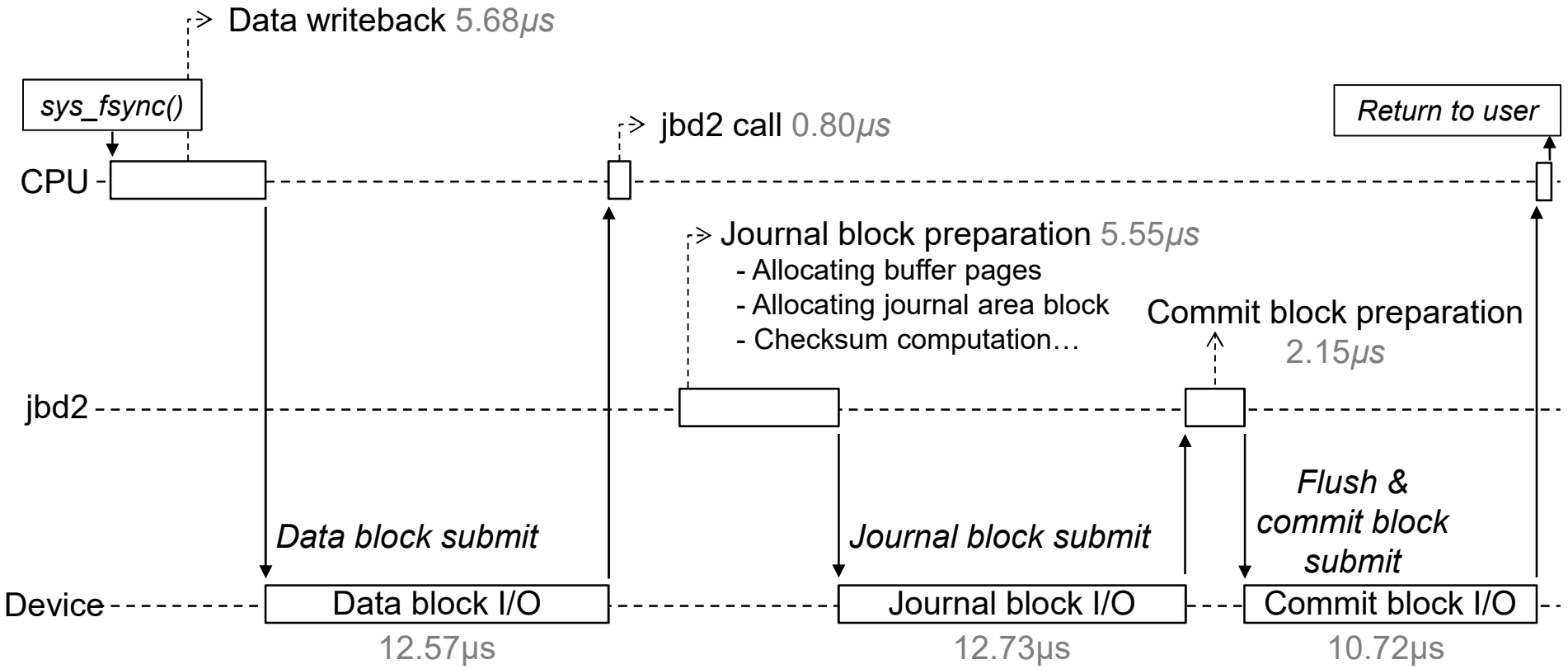
Read Path Comparison



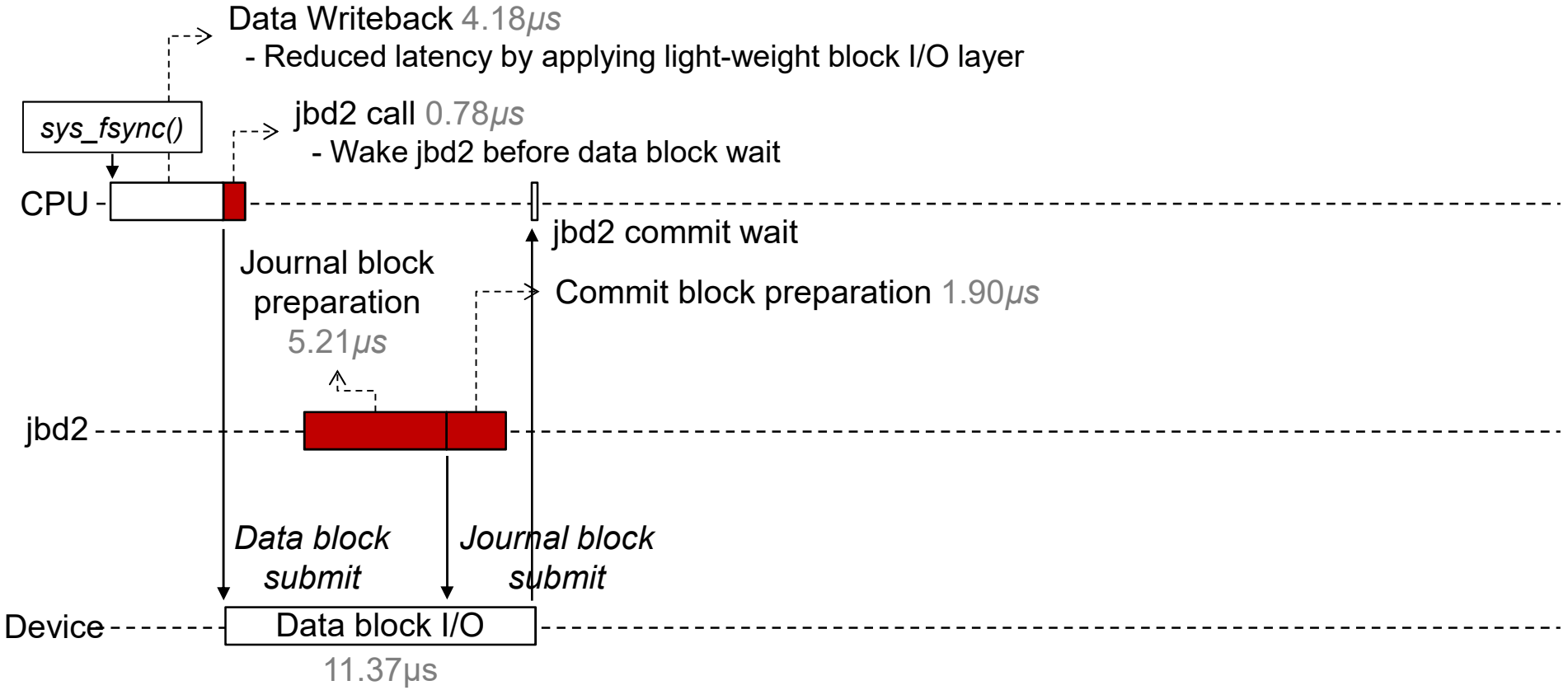
Agenda

- Read path
  - Analysis of vanilla read path
  - Proposed read path
- Light-weight block I/O layer
- **Write path**
  - Analysis of vanilla write path
  - Proposed write path
- Evaluation
- Conclusion

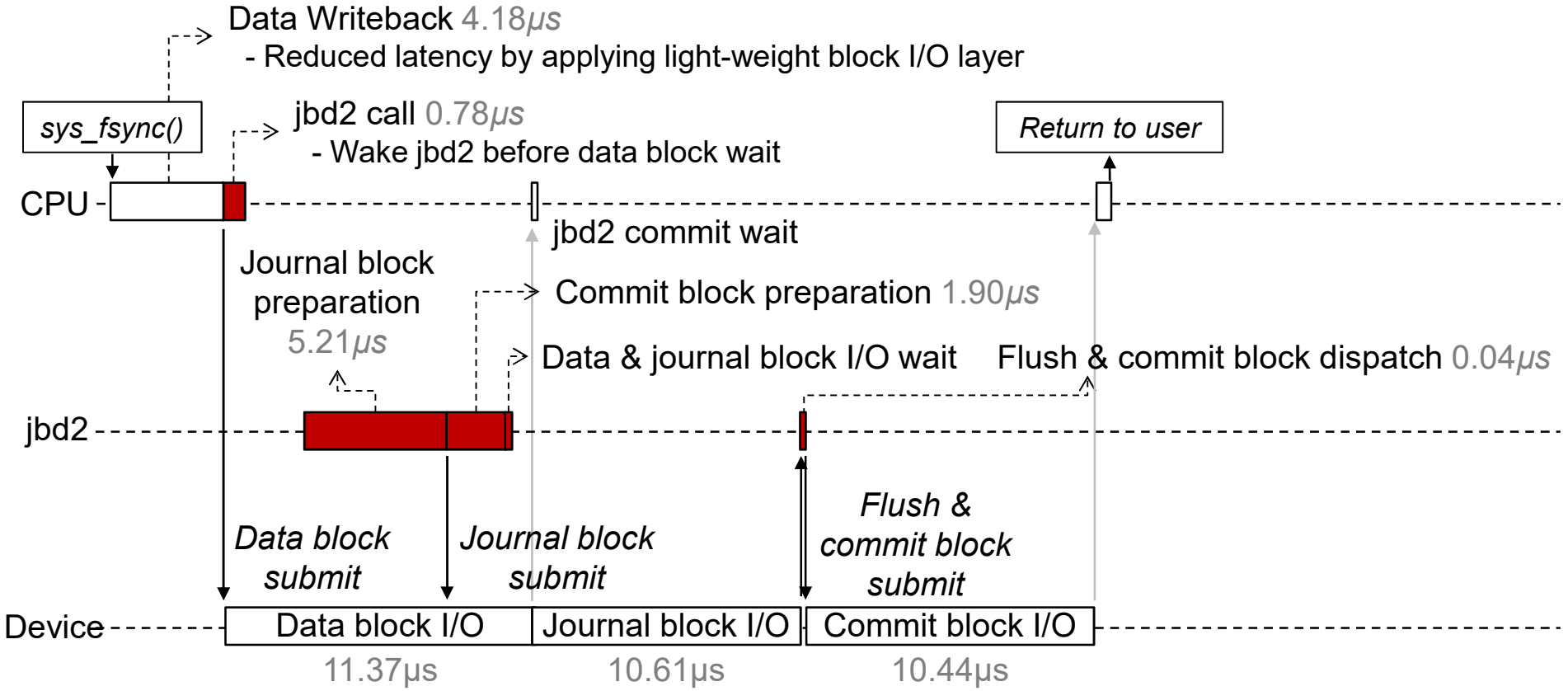
Analysis of Vanilla Fsync Path (Ext4 Ordered Mode)



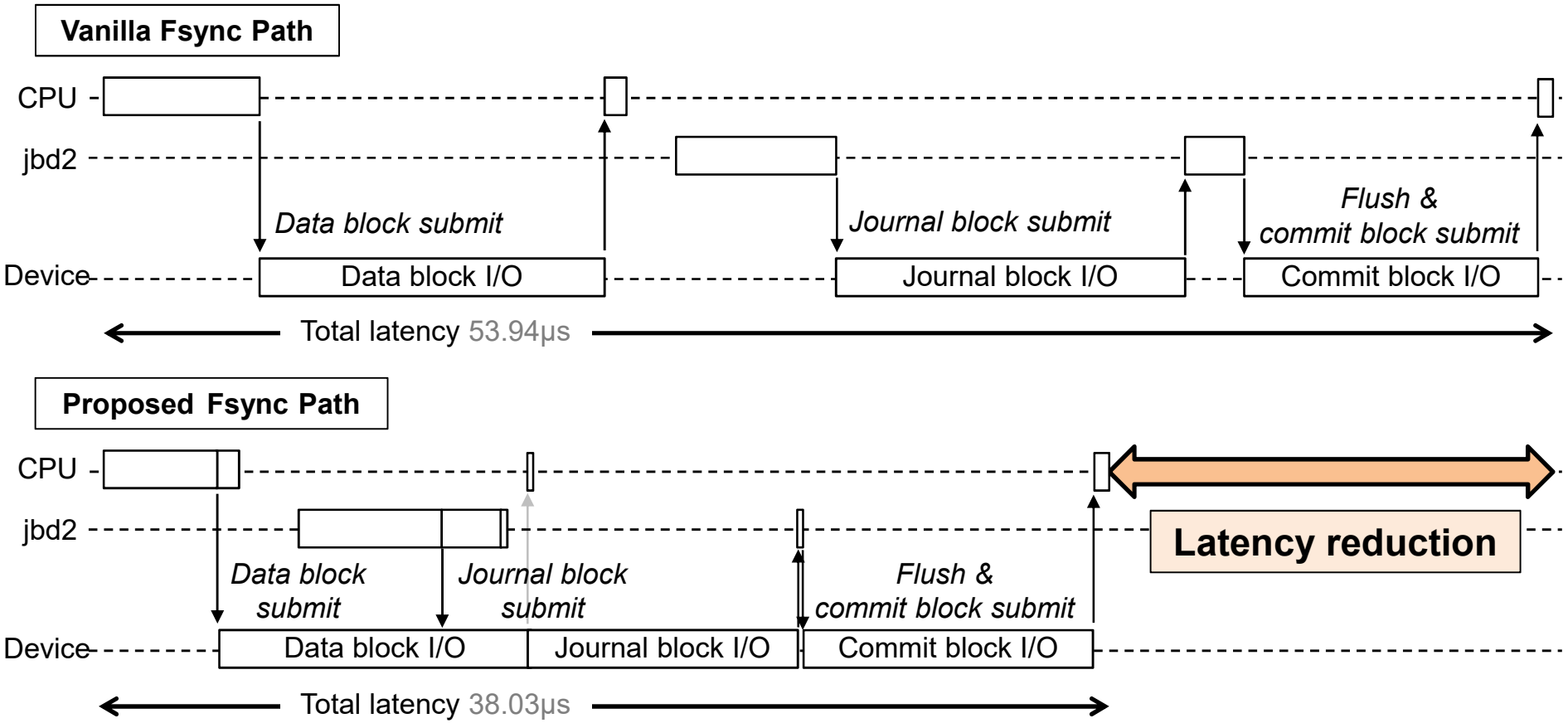
Proposed Fsync Path (Ext4 Ordered Mode)



Proposed Fsync Path (Ext4 Ordered Mode)



Fsync Path Comparison (Ext4 Ordered Mode)



Agenda

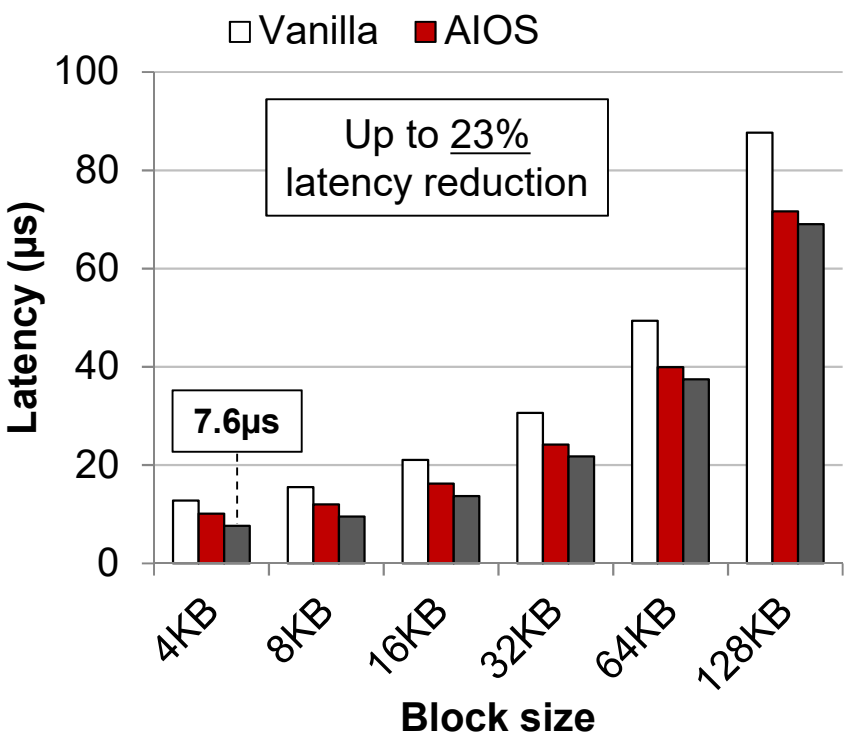
- Read path
  - Analysis of vanilla read path
  - Proposed read path
- Light-weight block I/O layer
- Write path
  - Analysis of vanilla write path
  - Proposed write path
- **Evaluation**
- Conclusion

Experimental Setup

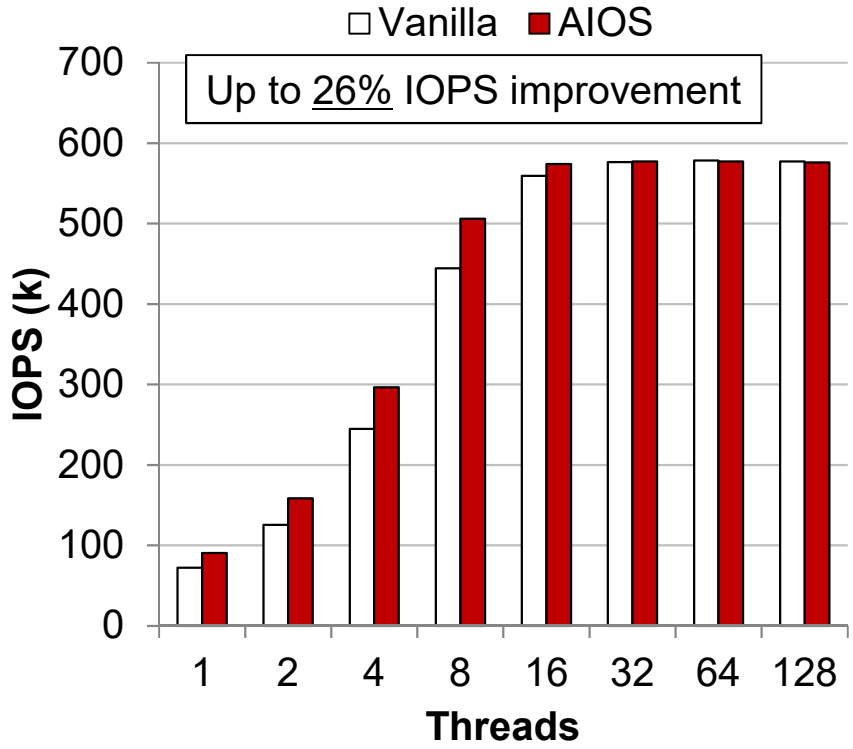
Server	Dell R730
OS	Ubuntu 16.04.4
Base kernel	Linux 5.0.5
CPU	Intel Xeon E5-2640v3 2.6GHz 8-cores
Memory	DDR4 32GB
Storage devices	Z-SSD: Samsung SZ985 800GB Optane SSD: Intel Optane 905P 960GB
Workloads	Synthetic micro-benchmark: FIO Real-world workload: RocksDB DBbench

FIO Performance (Random Read)

- **Single thread**

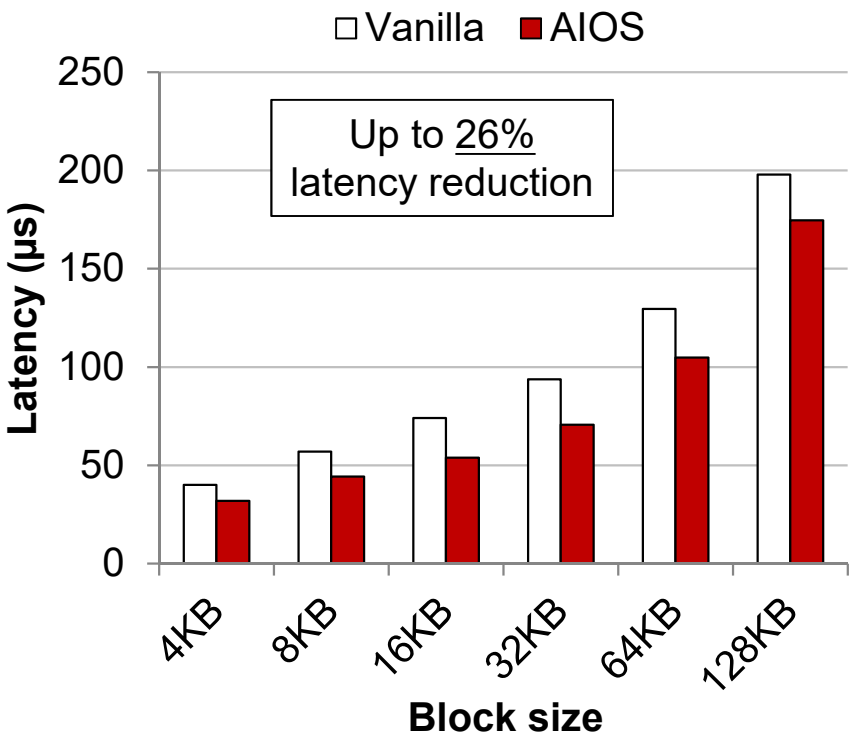


- **4KB block size**

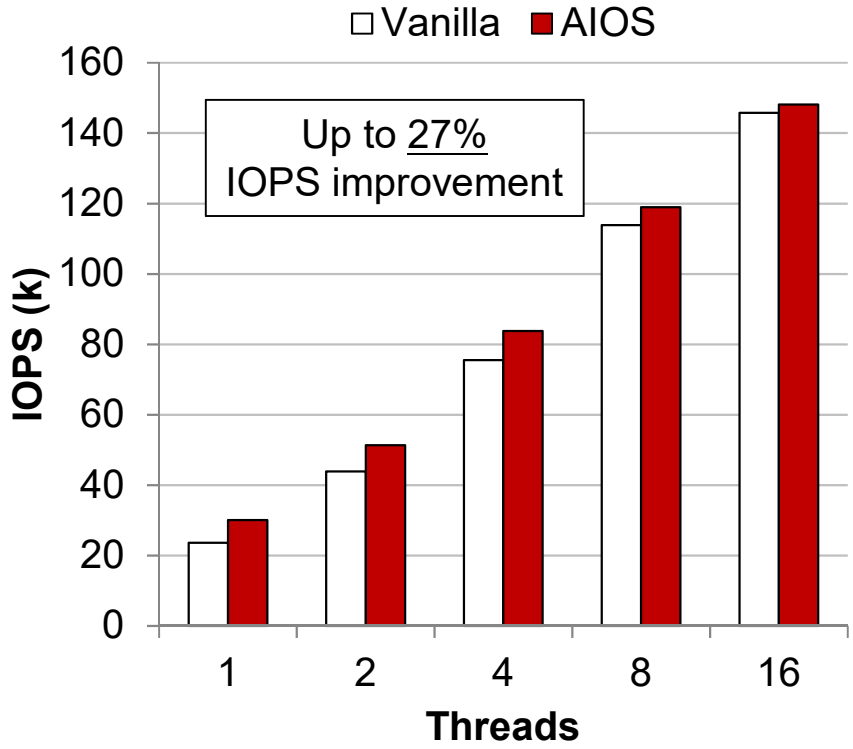


FIO Performance (Random Write+Fsync, Ext4 Ordered)

- **Single thread**

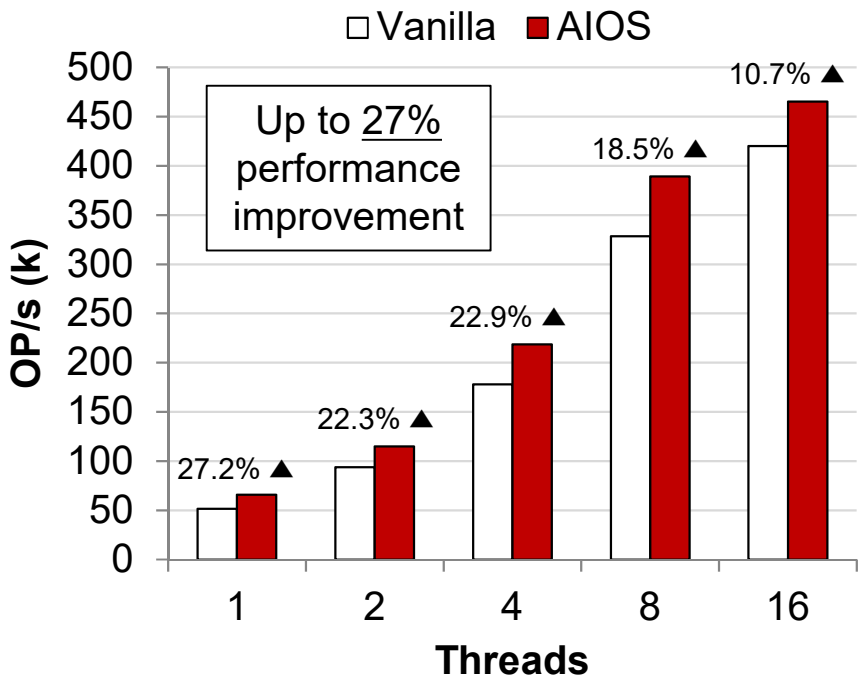


- **4KB block size**

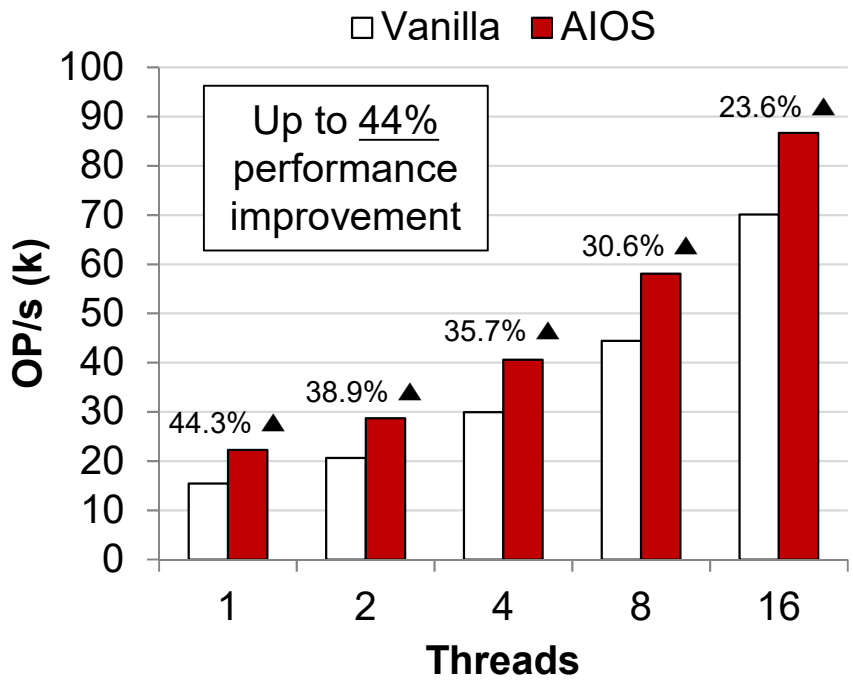


RocksDB Performance

- **DBbench readrandom**
  - 64GB dataset



- **DBbench fillsync**
  - 16GB dataset

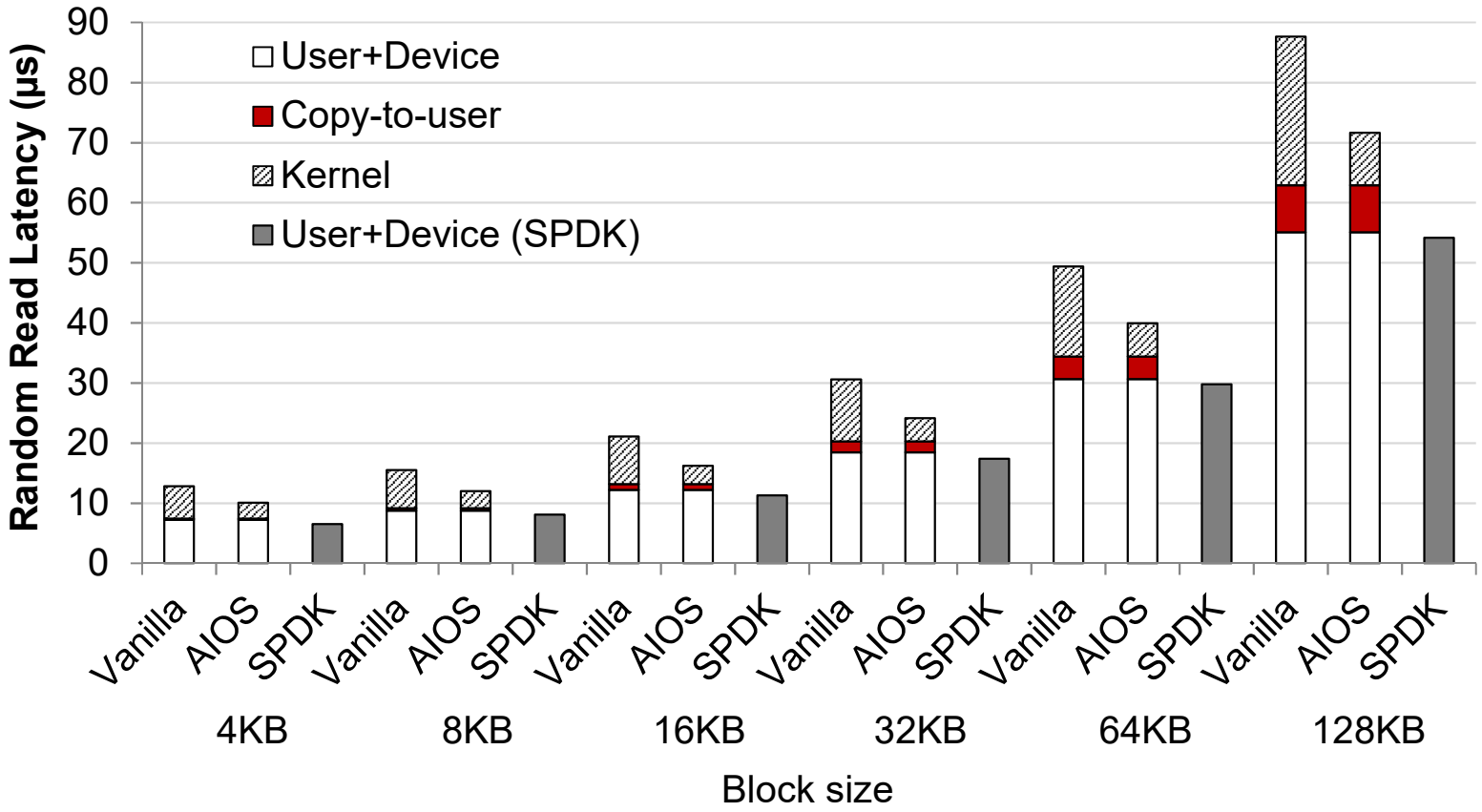


Conclusion

- **Asynchronous I/O stack**
  - Applies asynchronous I/O concept to the kernel I/O stack itself
  - Overlaps computation with I/O to reduce total I/O latency
- **Light-weight block I/O layer**
  - Provides low-latency block I/O services for low-latency NVMe SSDs
- **Performance evaluation**
  - Achieves a single-digit microsecond I/O latency on Optane SSD
  - Achieves significant latency reduction and performance improvement on real-world workloads

Source code: https://github.com/skkucsl/aios

Comparison with User-level Direct Access Approach



Q&A

- **Thank you**

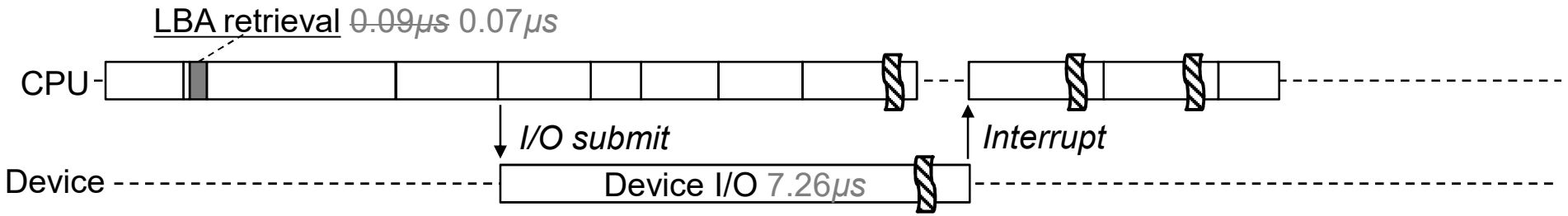
Acknowledgment

- **This talk is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2017R1C1B2007273).**

Extra Slides

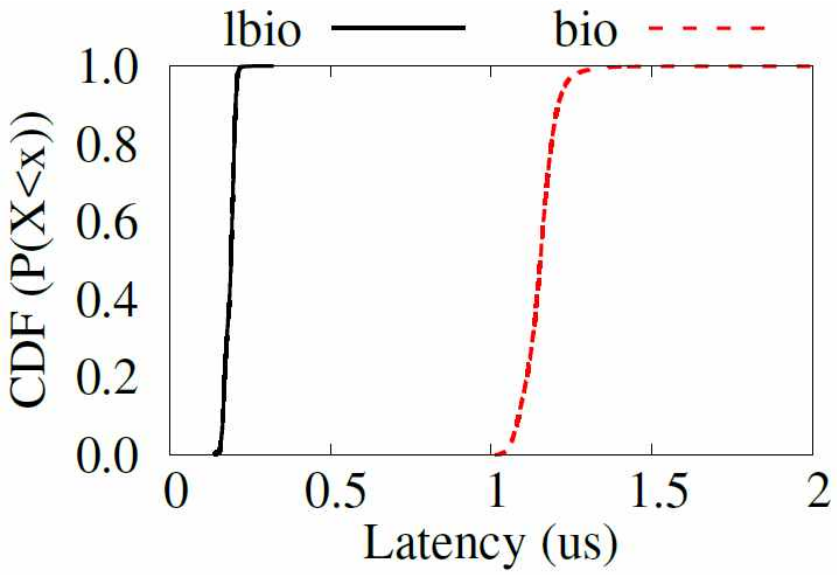
Preloading Extent Tree

- **Extent status tree (in-memory structure)**
  - Contains LBA mapping info. for file blocks (composed of extent objects)
  - Occurs additional I/O request to read mapping block in case of extent lookup miss
- **Control & data plane approach [*OSDI'14*]**
  - Preloads the entire mapping info. into the memory in the control plane (e.g., file open)
  - Selectively applies to files requiring low latency access

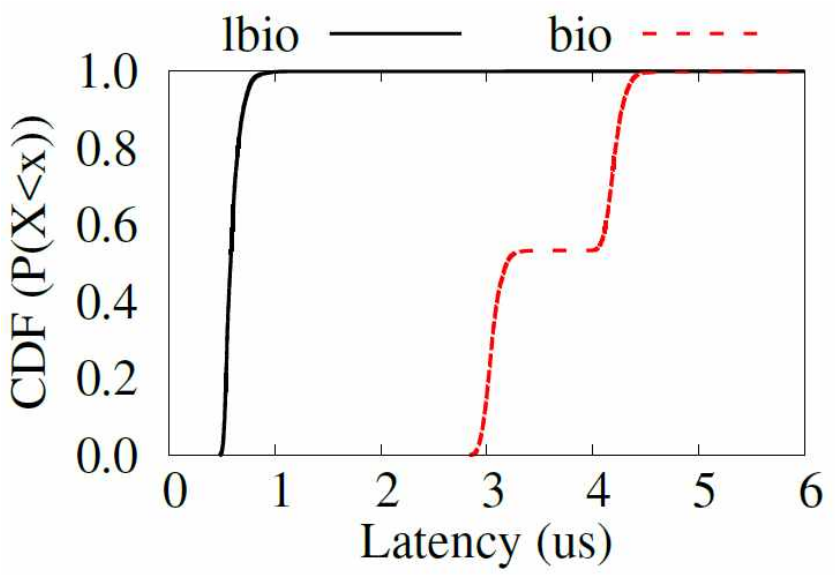


CDF of the Block I/O Submission Latency

- **Block I/O submission latency**
  - Time from allocating a object (bio or lbio) to dispatching an I/O command to a device

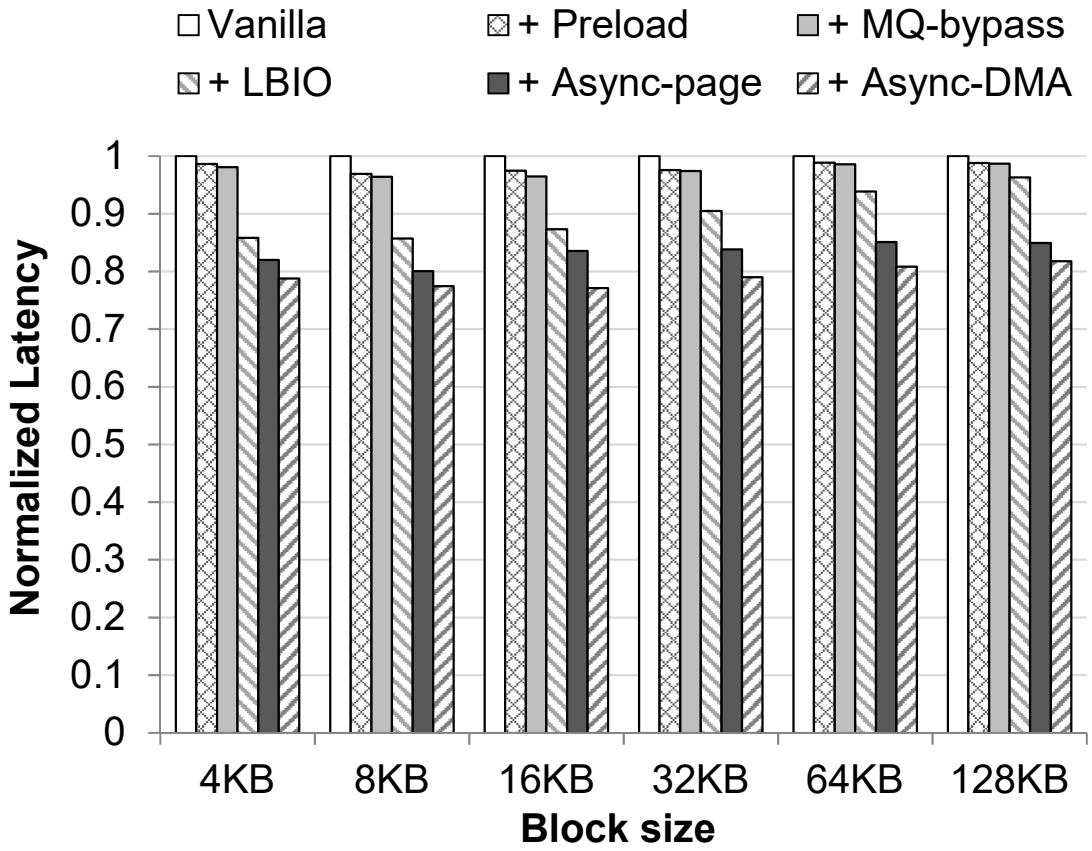


4KB random read



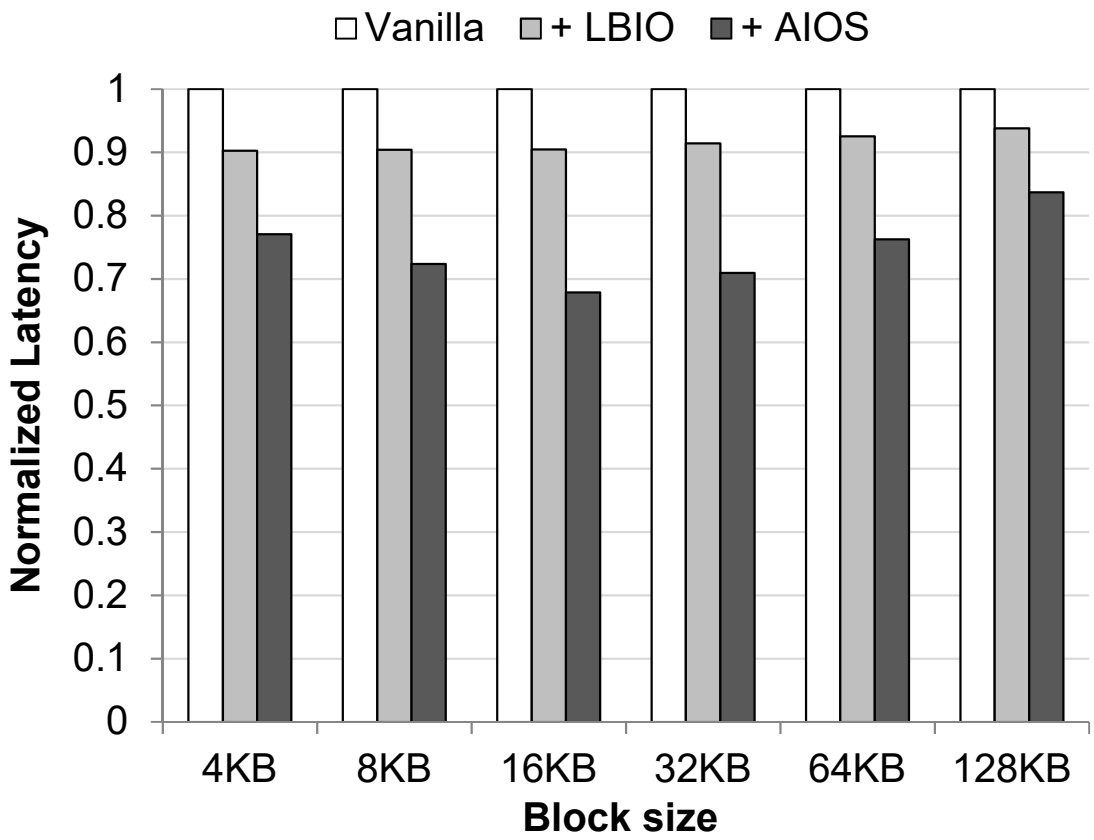
32KB random read

Read Performance Analysis with Opt. Levels



- **Preload**
  - Preloading extent tree
- **MQ-bypass**
  - Bypassing SW, HW queue
- **LBIO**
  - Light-weight block I/O layer
- **Async-page**
  - Asynchronous page allocation
  - Lazy page cache insertion
- **Async-DMA**
  - Asynchronous DMA mapping
  - Lazy DMA unmapping

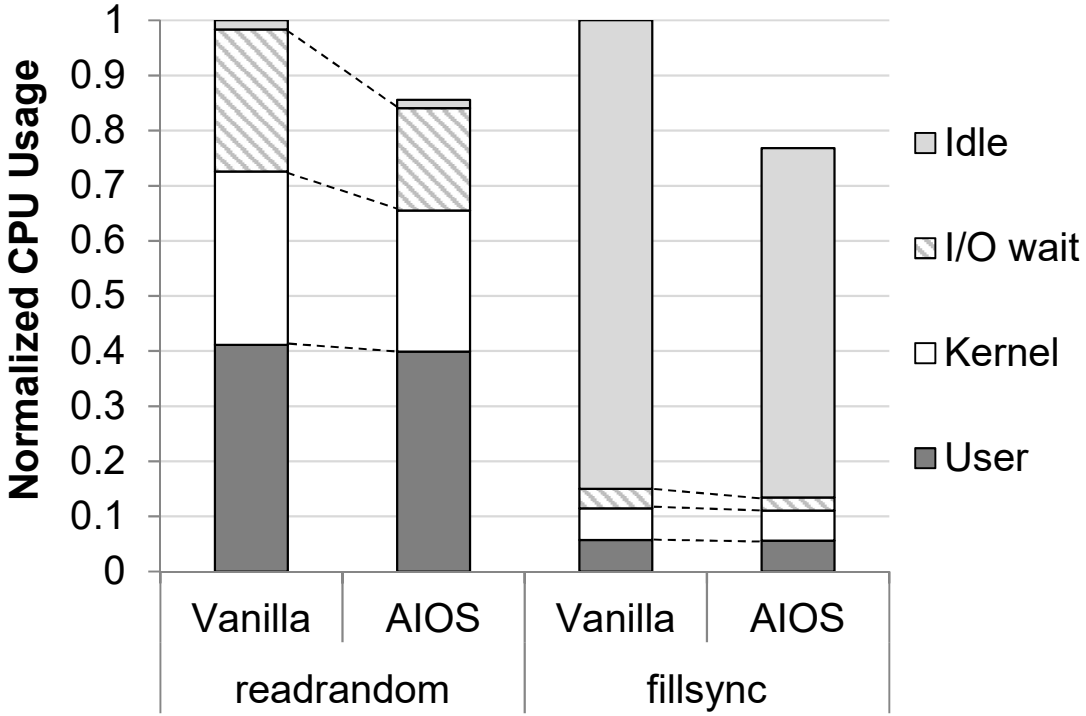
Write(+fsync) Performance Analysis with Opt. Levels



- **LBIO**
  - Light-weight block I/O layer
  - Synchronous page allocation / DMA mapping
- **AIOS**
  - Proposed fsync path

CPU Usage Breakdown

- **RocksDB DBbench readrandom / fillsync using 8 threads**

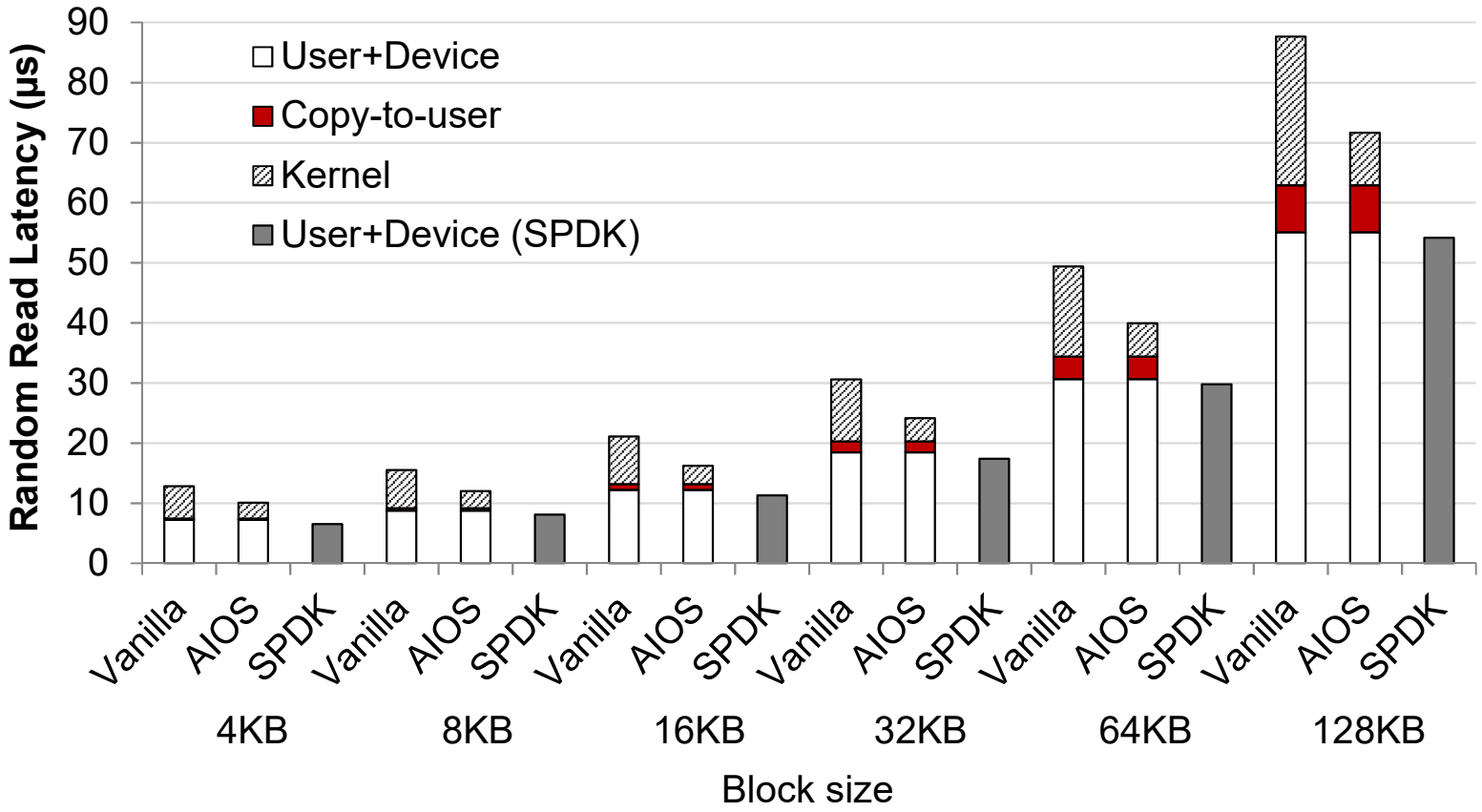


Reduced overall runtime

Reduced I/O wait & kernel CPU time

Effective overlapping between I/O and kernel operations through **AIOS**

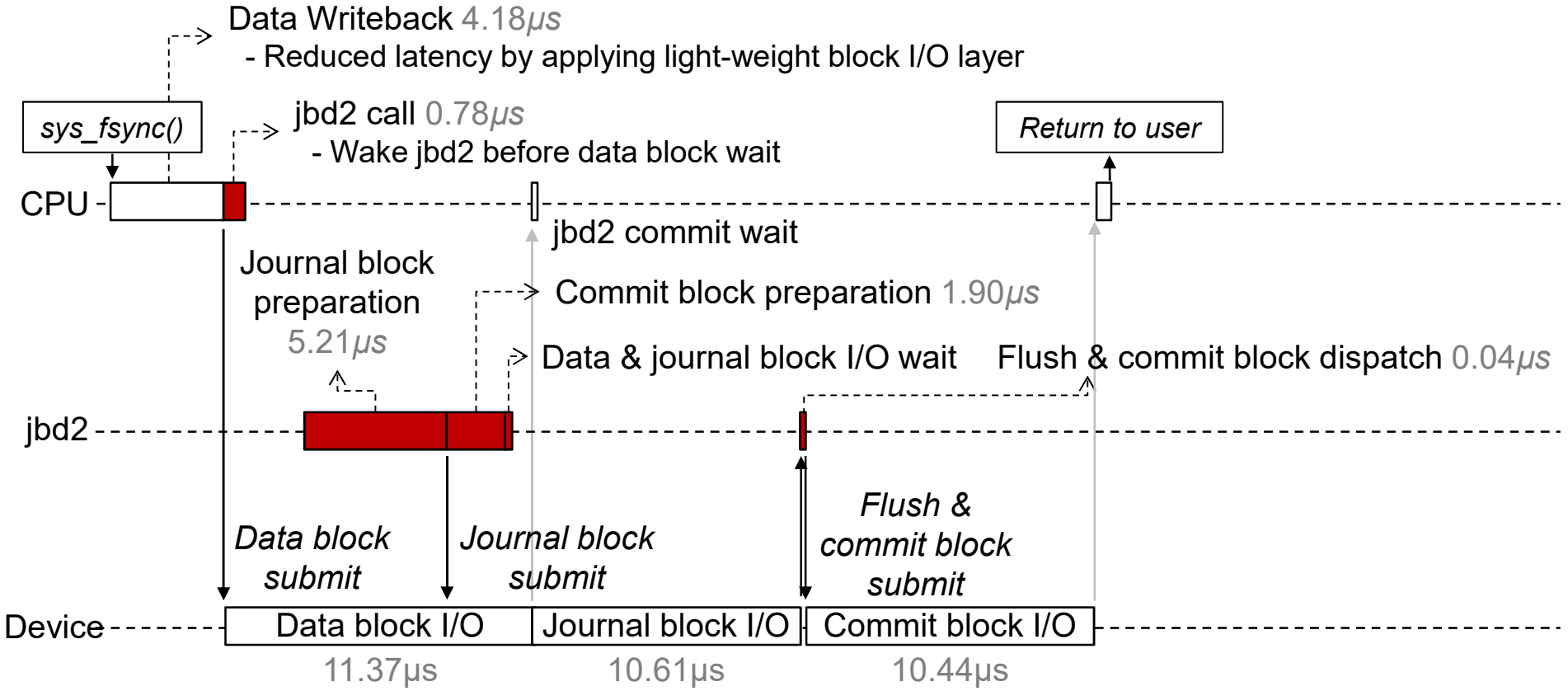
Comparison with User-level Direct Access Approach



Memory Overheads on Asynchronous I/O Stack

	Object	Linux block layer	Light-weight block layer
Statically allocated objects (per-core)	request pool	412KB	-
	lbio array	-	192KB
	Free page pool	-	256KB
128KB block request	(l)bio + (l)bio_vec	648B	704B
	request	384B	-
	iod	974B	-
	prp_list	4096B	4096B
	Total	6104B	4800B

Proposed Fsync Path (Ext4 Ordered Mode)



Lazy DMA Unmapping

- **Implementation**
  - Delays DMA unmapping to when a system is idle or waiting for another I/O requests
  - Extended version of the deferred protection scheme in Linux [*ASPLOS'16*]
  - Optionally disabled for safety