FAST'25
FEBRUARY 25–27, 2025
SANTA CLARA, CA, USA

OPIMQ: Order Preserving IO Stack for Multi-Queue Block Device

Jieun Kim Joontaek Oh Juwon Kim SeungWon Yoo and Youjip Won

OSLab
Operating Systems Laboratory

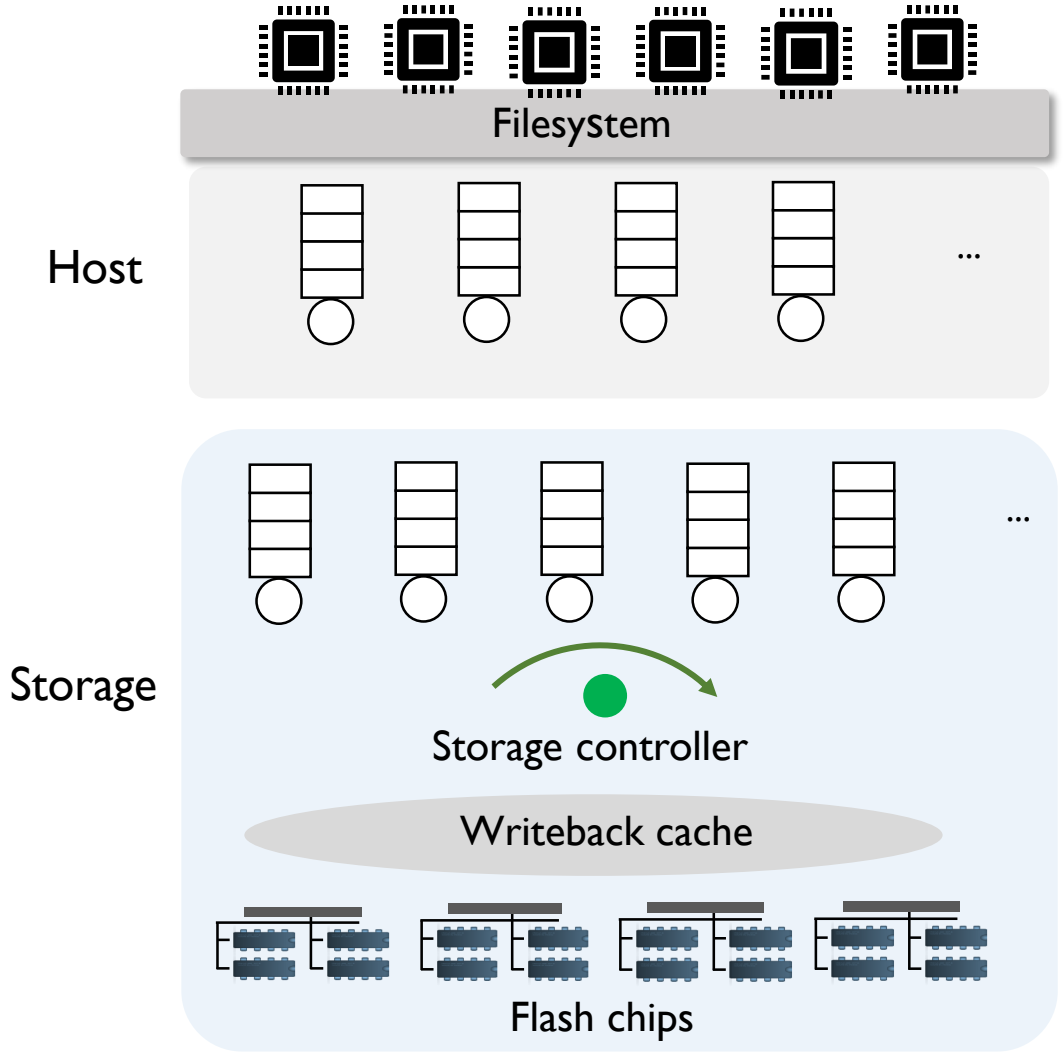
Outlines

- Background & Motivation
- Problem Formulation
- Design
- Evaluation
- Conclusion

Outlines

- Background & Motivation
- Problem Formulation
- Design
- Evaluation
- Conclusion

Modern I/O Stack: Multi-Queued Architecture



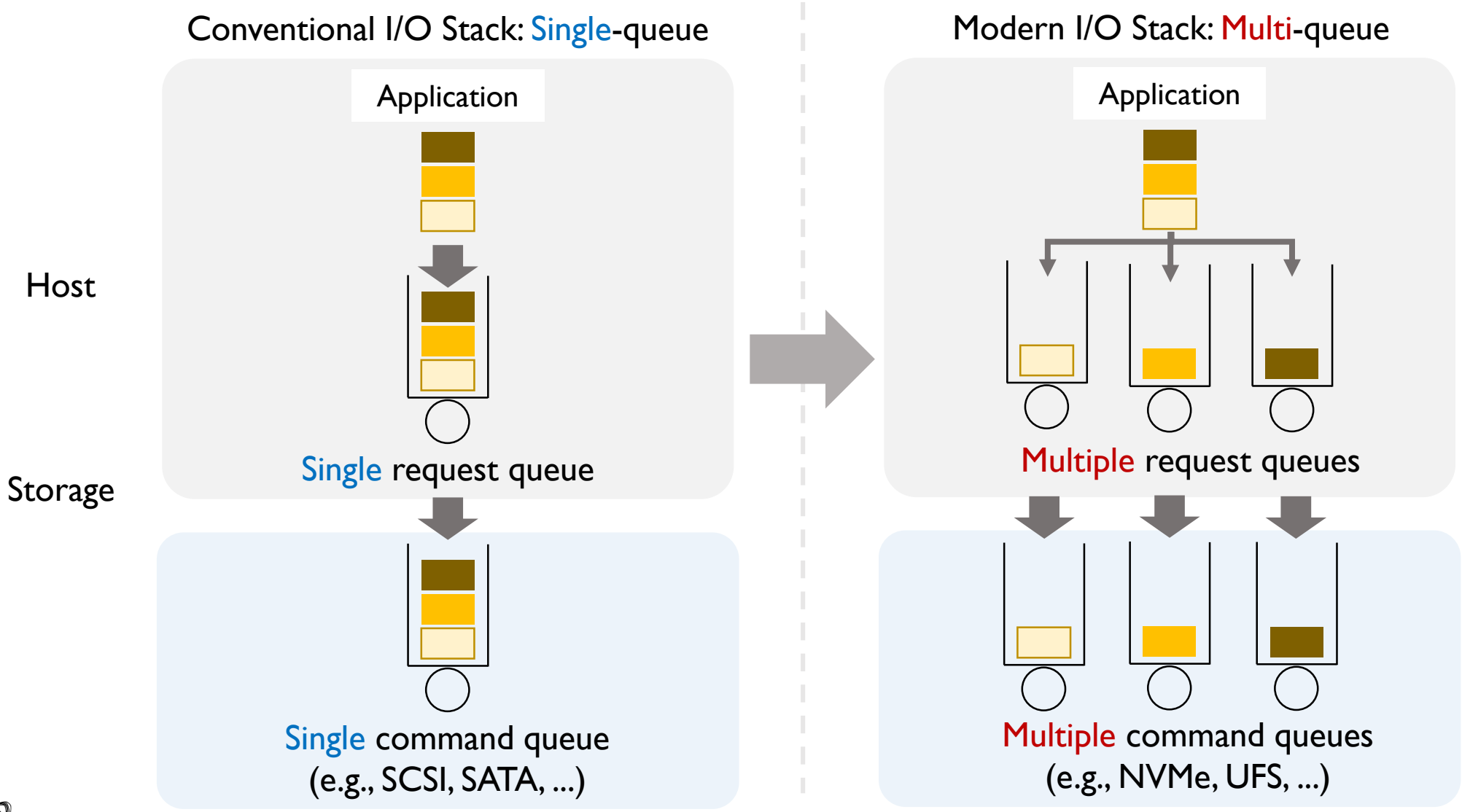
Multiple Request queues

- To avoid contention on request queue

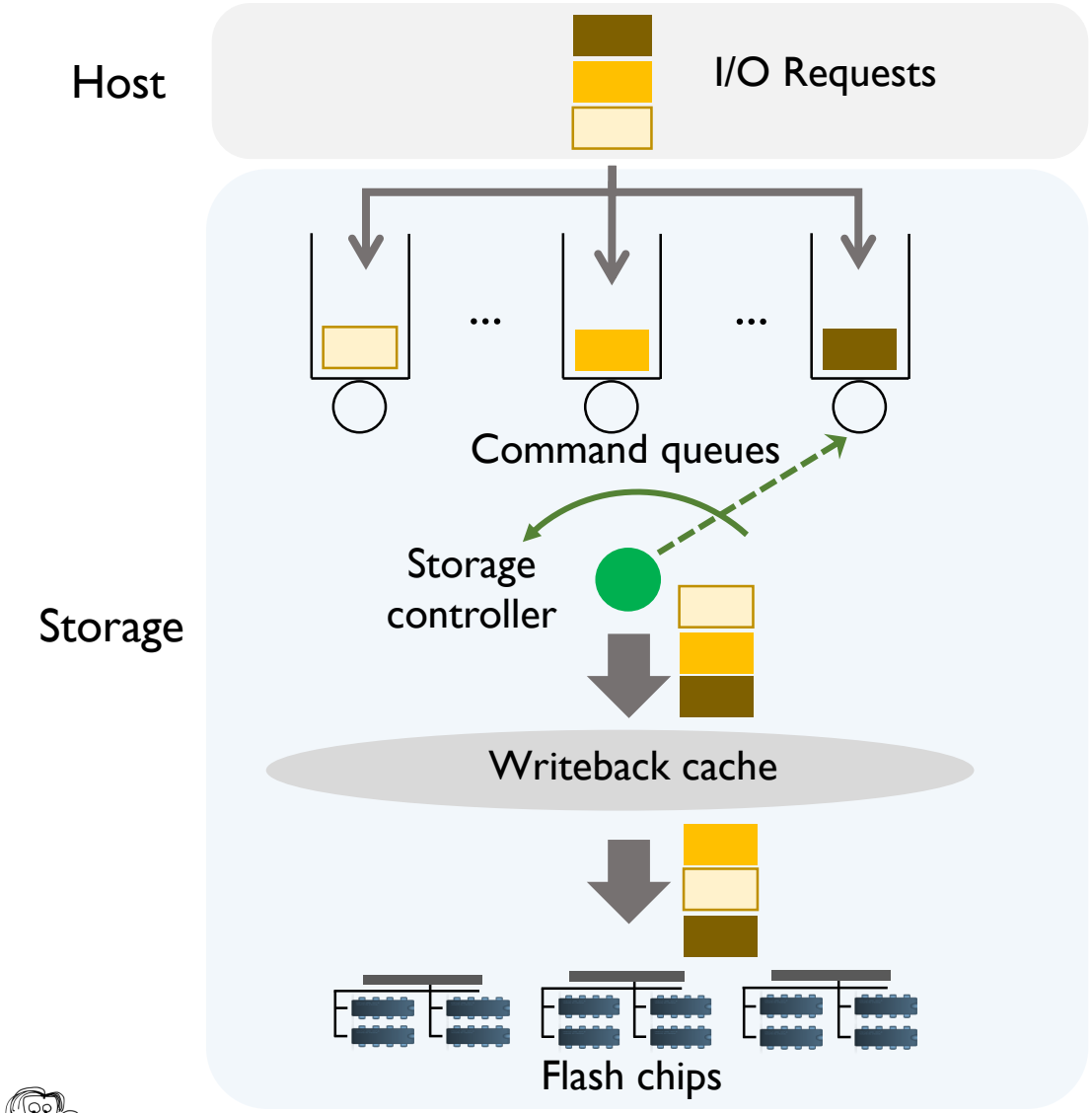
Multiple Command queues

- To exploit rich internal bandwidth
- Ex. NVMe, UFS

I/O Request Flow



Multi-Queue I/O Stack: Orderless

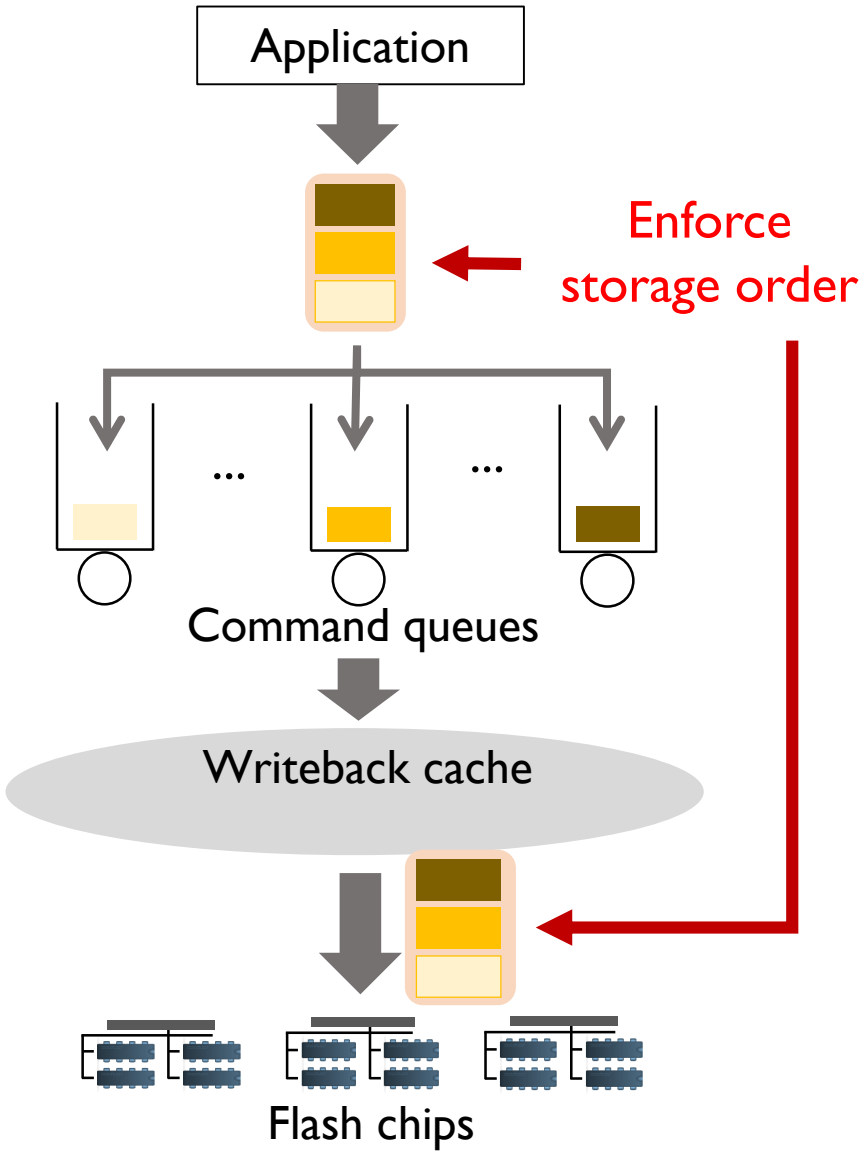


Modern I/O Stack is *orderless*.

1) I/O requests can be distributed across multiple queues

2) I/O requests can be reordered at
   - Storage controller
   - FTL (Flash Translation Layer)

Storage Order



Storage Order: The order in which a set of data blocks are made durable.

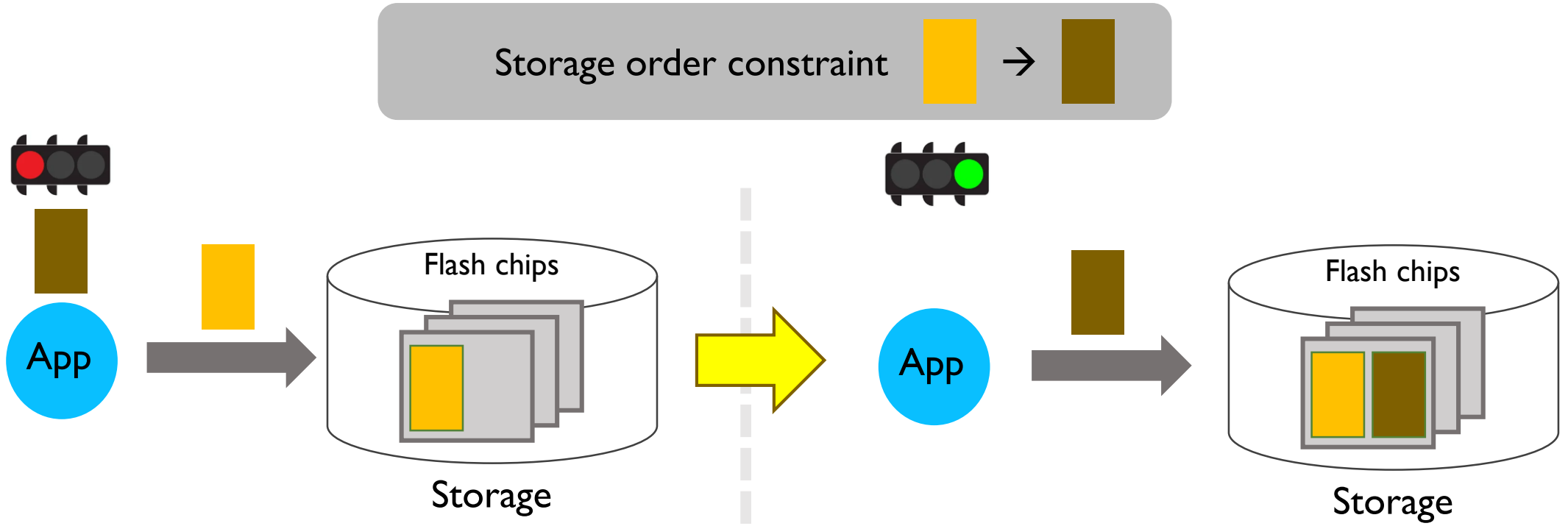
Storage order constraint:
W1 → W2
W1 should be made ***durable*** **before W2**.

Applications need to enforce the storage order.

- To ensure durability and correctness in crash recovery.
- e.g, database logging, Filesystem journaling, ...

Storage order guarantee

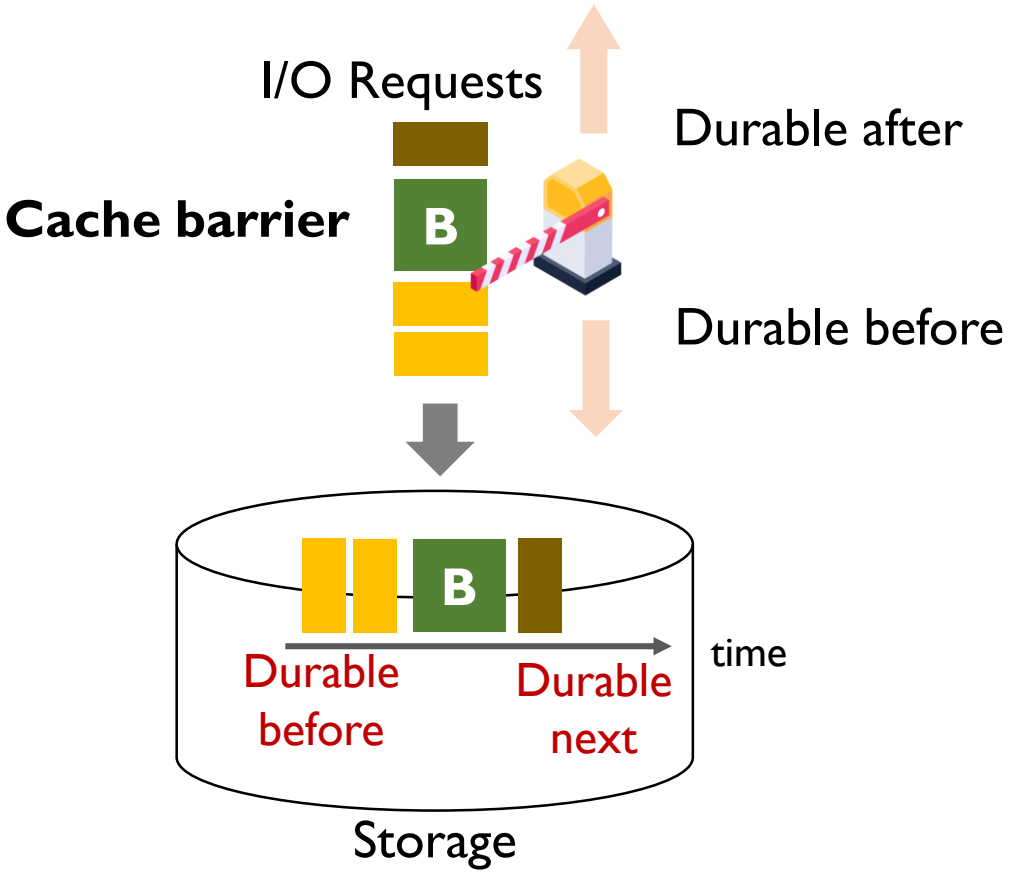
Storage order guarantee: Ensure durability is the issued order



Cache barrier command

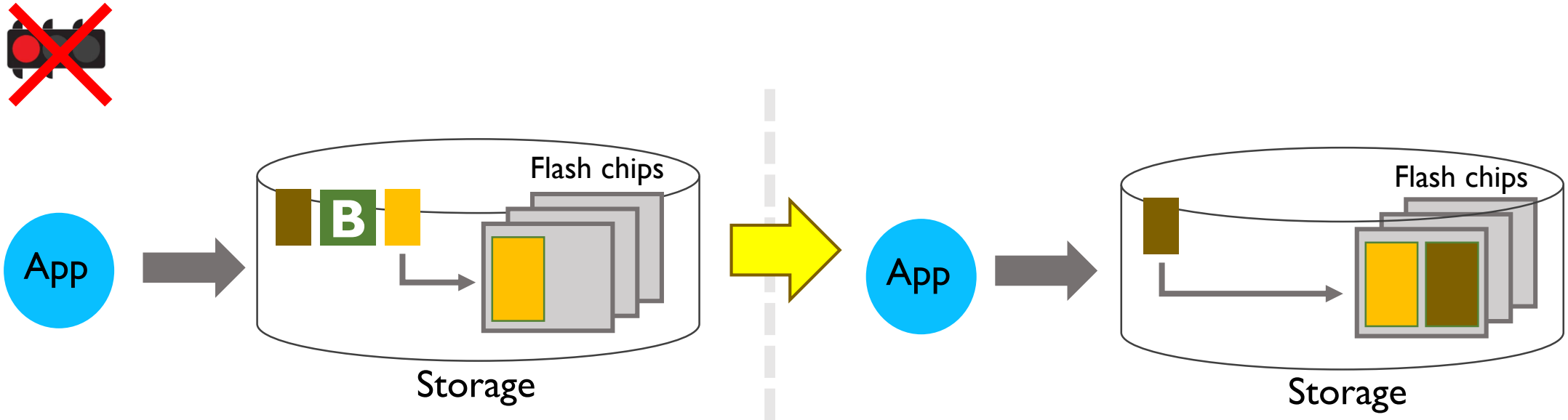
Controlling storage order

Cache barrier command (2005, eMMC)



Storage order guarantee in Order-preserving I/O Stack

Using *cache barrier* command



Ensuring storage order *without waiting* for the previous request's durability

Exisitng Works

In-Order Write with cache barrier

- Barrier-Enabled I/O Stack [FAST'18]
- LazyBarrier [ASPLOS'24]

Cache barrier

Out-of-Order Write with ordered recovery

- OPTR [ATC'19]

Global write identifier

- HORAE [OSDI'20]
- ccNVMe [SOSP'21]
- RIO [Eurosys'23]

PMR in NVMe

Limitations of Exisitng Works

In-Order Write with cache barrier

• Barrier-Enabled I/O Stack [FAST'18]	Only works in **single-queue** device
• LazyBarrier [ASPLOS'24]	Only support **UFS**

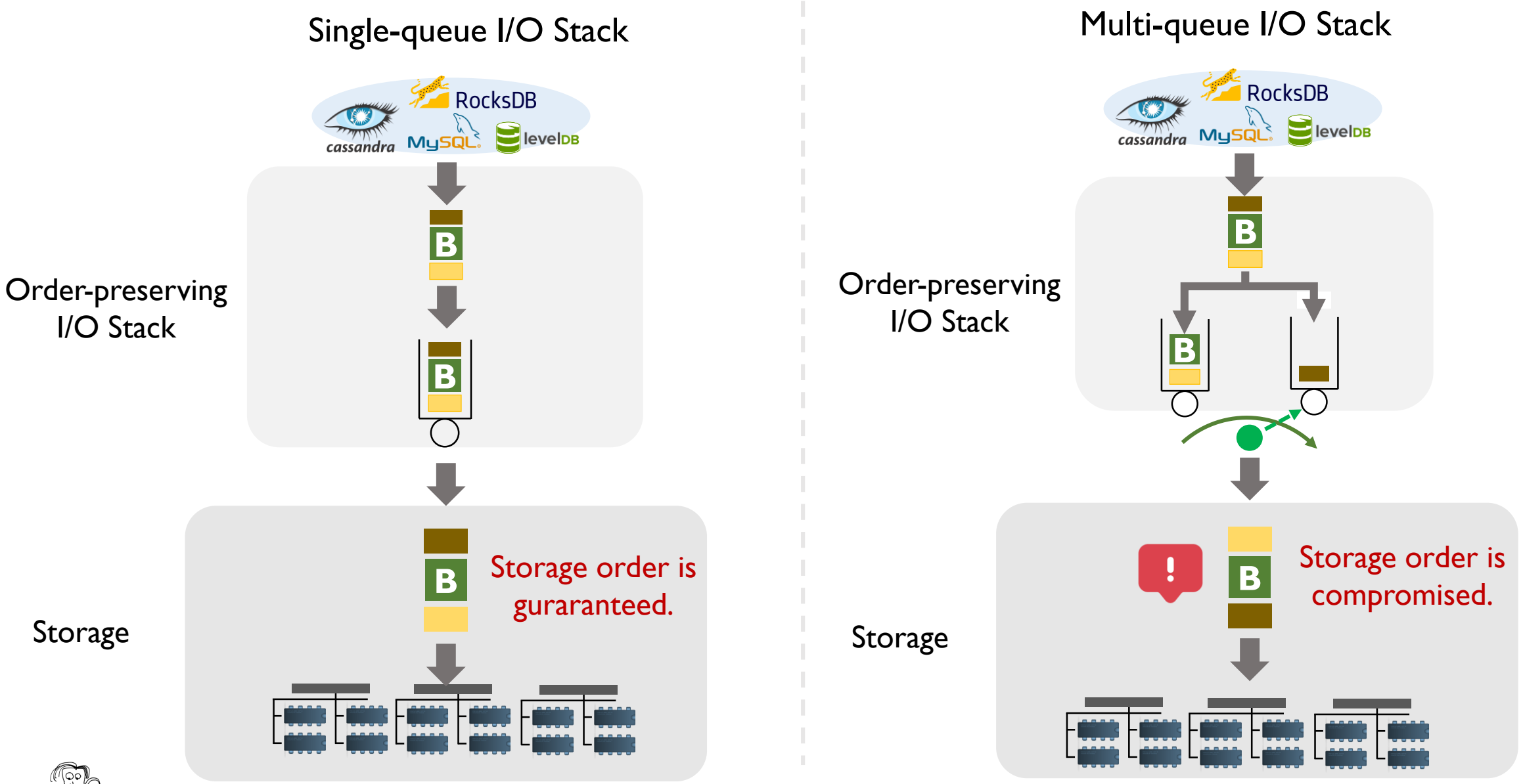
Out-of-Order Write with ordered recovery

• OPTR [ATC'19]	Only support single write stream
• HORAE [OSDI'20] • ccNVMe [SOSP'21] • RIO [Eurosys'23]	Require specific hardware Inefficient ordering guarantee

Outlines

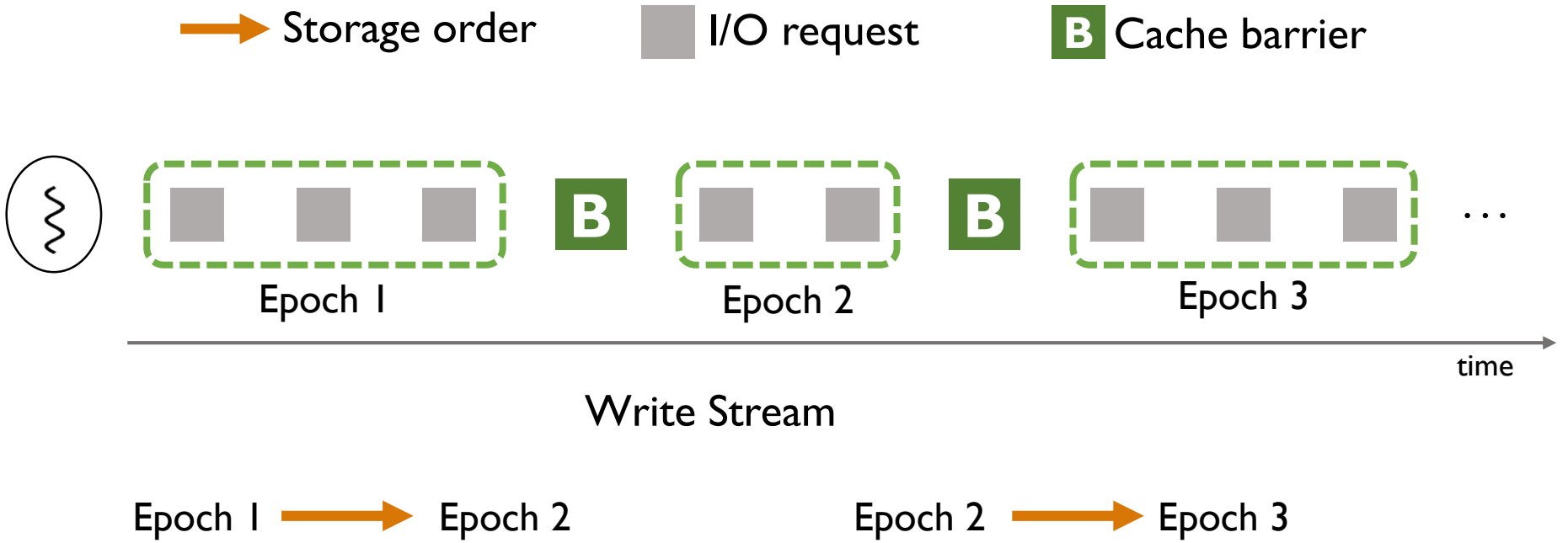
- Background & Motivation
- **Problem Formulation**
- Design
- Evaluation
- Conclusion

Problem Formulation



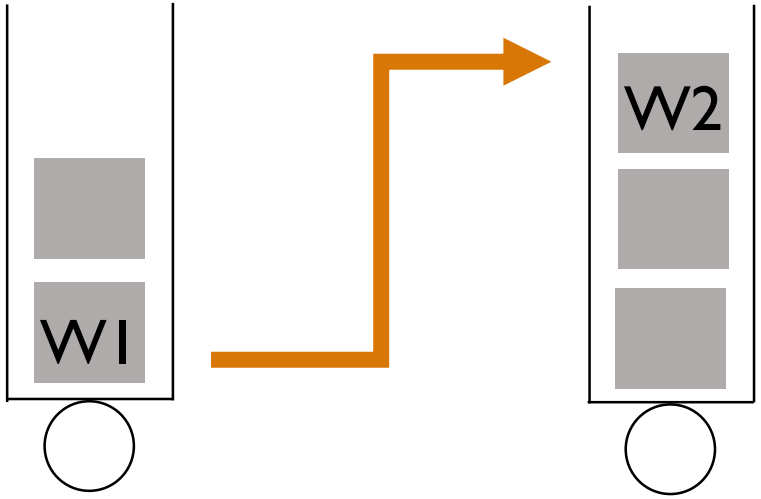
Model

- Stream: a set of requests from a thread
- Epoch: a set of write requests in a stream that can be reordered with each other



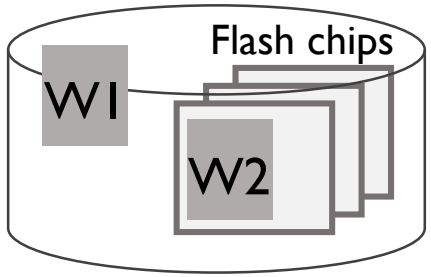
Problem: Inter-Queue Storage Order

Storage order between the multiple queues



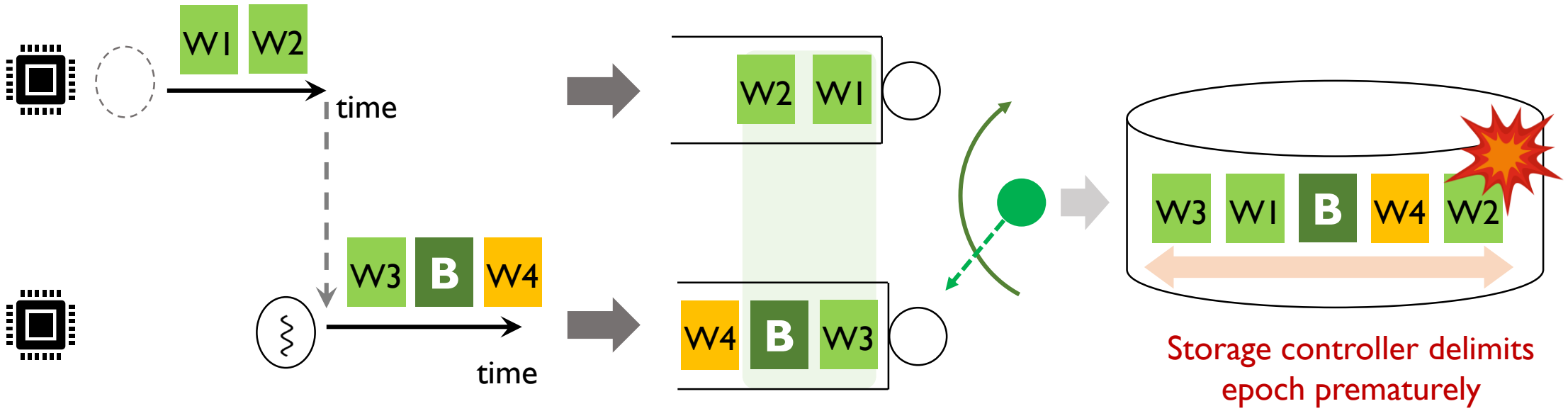
W1 and W2 can be either from the same stream or from different streams.

W1 → W2



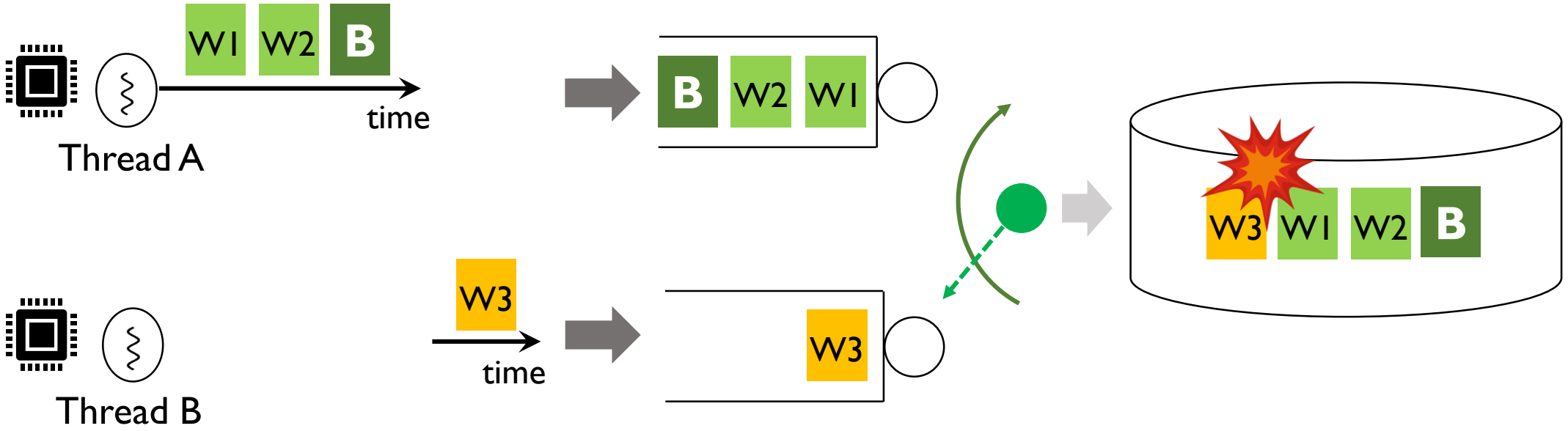
Challenge 1: Guarantee Intra-Stream Storage Order

- Due to core migration of thread by work-stealing (load balancing)
- *Epoch Split:* Write requests within an epoch are distributed across multiple queues



Challenge 2: Guarantee Inter-Stream Storage Order

- When requests with ordering dependency are from different threads
- When the threads run on different cores

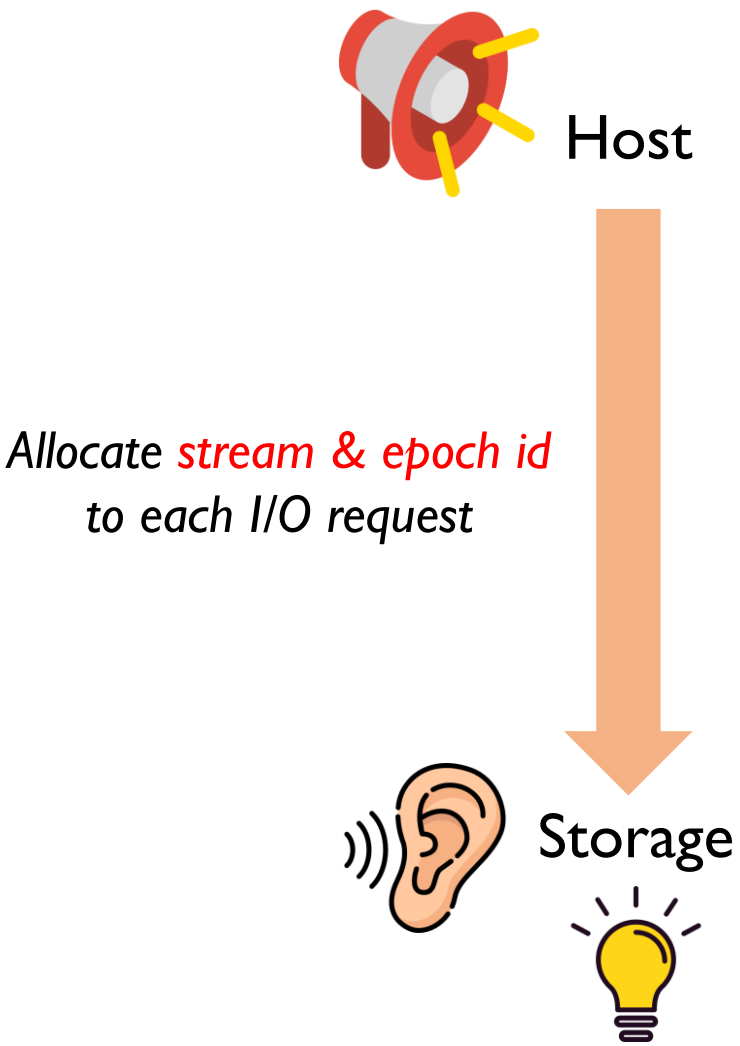


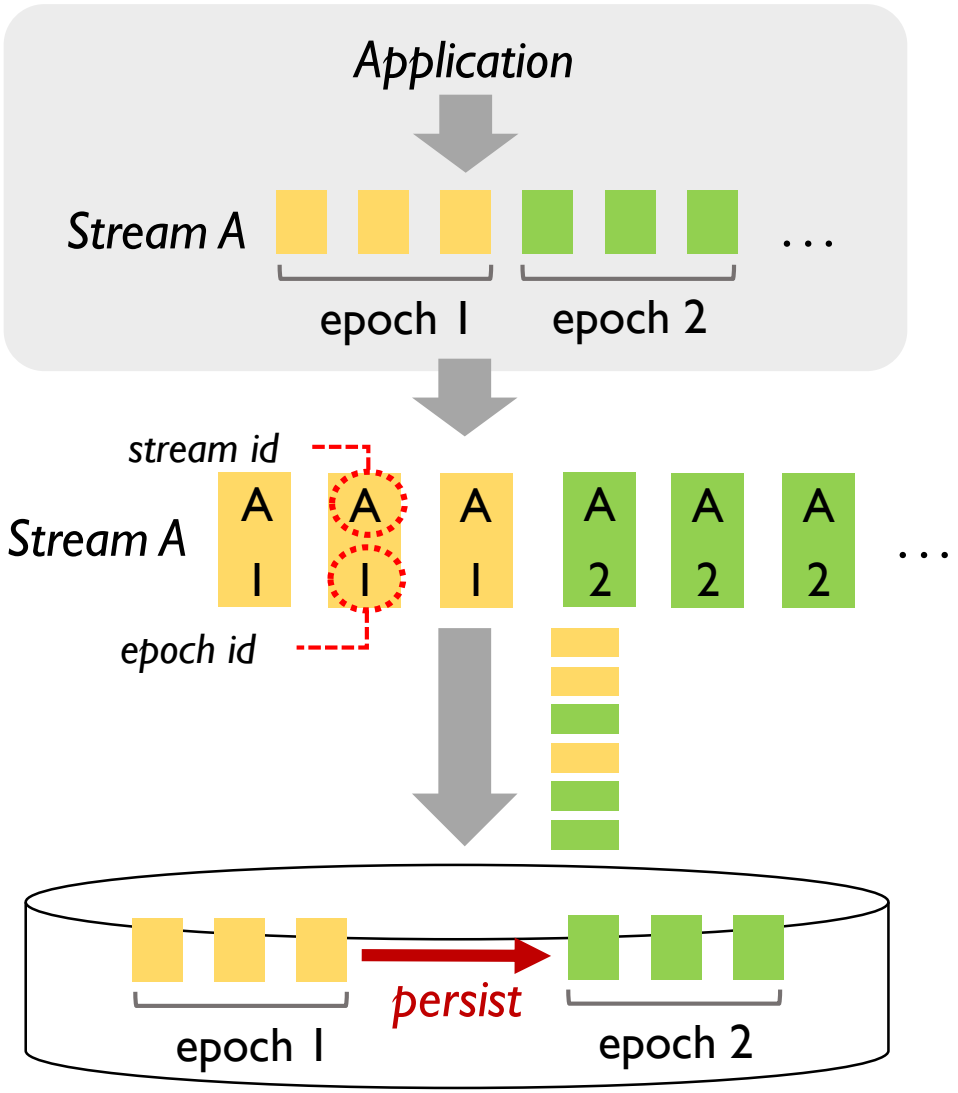
OPIMQ: Order-Preserving I/O Stack for Multi-queue Block Device

Outlines

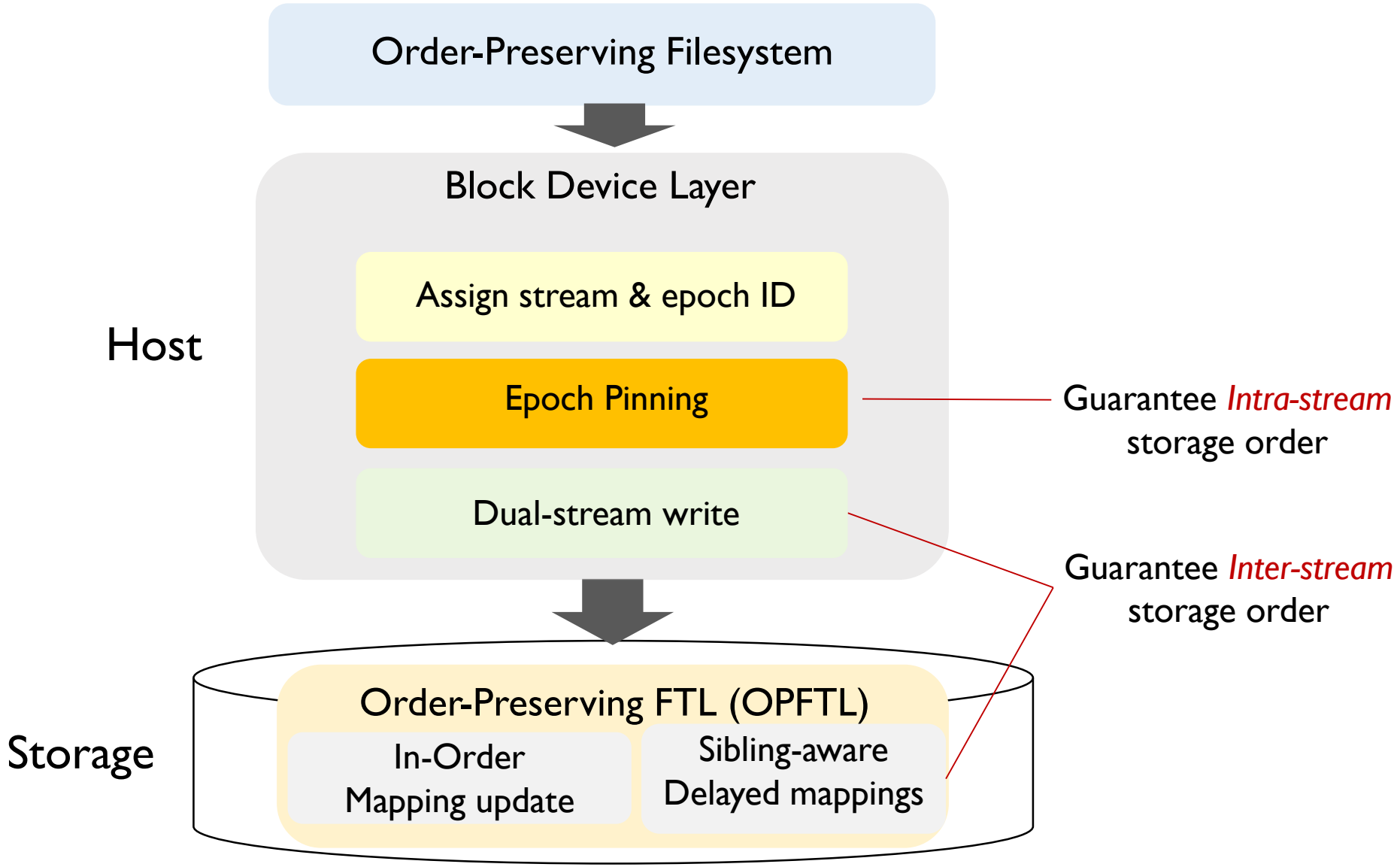
- Background & Motivation
- Problem Formulation
- **Design**
- Evaluation
- Conclusion

Concept


Host
Allocate stream & epoch id
to each I/O request
Storage

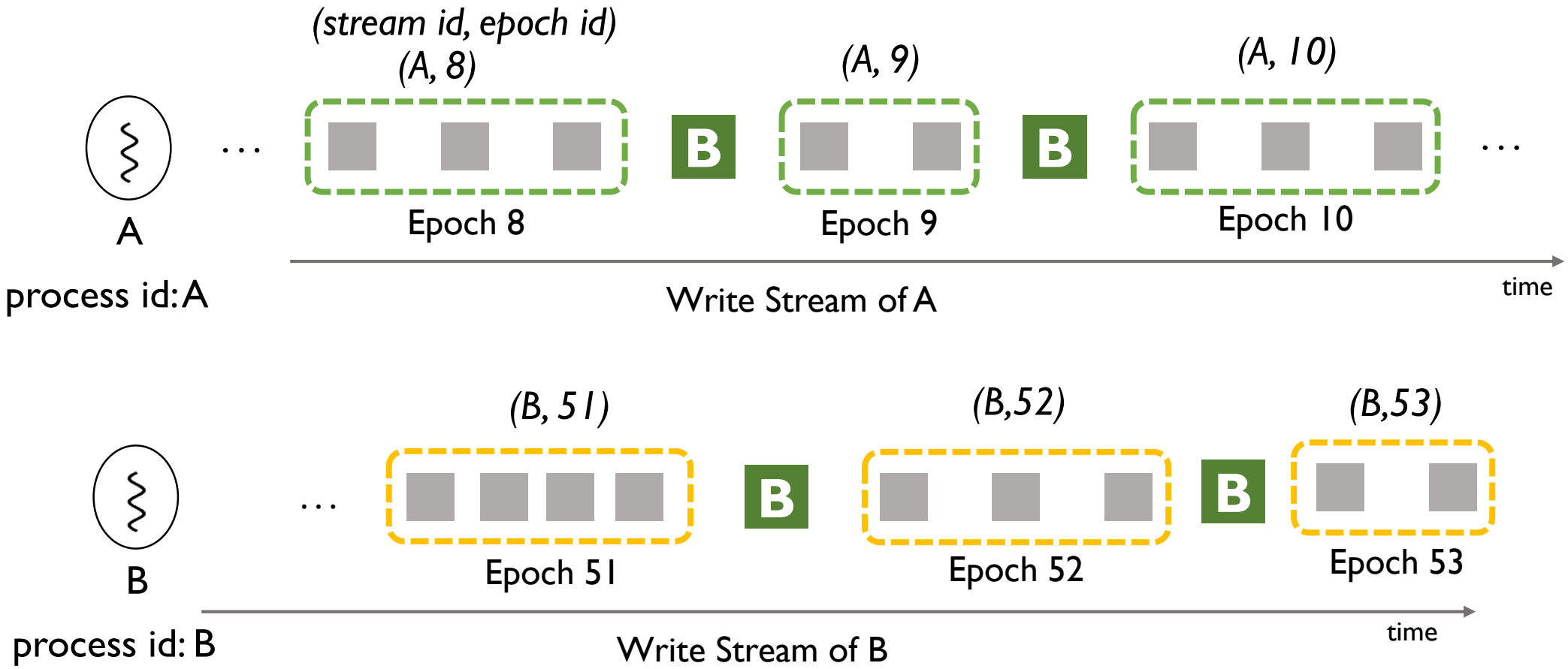

Application
Stream A
epoch 1
epoch 2
stream id
Stream A
A 1
A 1
A 1
A 2
A 2
A 2
epoch id
persist
epoch 1
epoch 2

Design


Order-Preserving Filesystem
Block Device Layer
Assign stream & epoch ID
Epoch Pinning
Dual-stream write
Host
Guarantee *Intra-stream* storage order
Guarantee *Inter-stream* storage order
Order-Preserving FTL (OPFTL)
In-Order Mapping update
Sibling-aware Delayed mappings
Storage

Assign stream and epoch ID

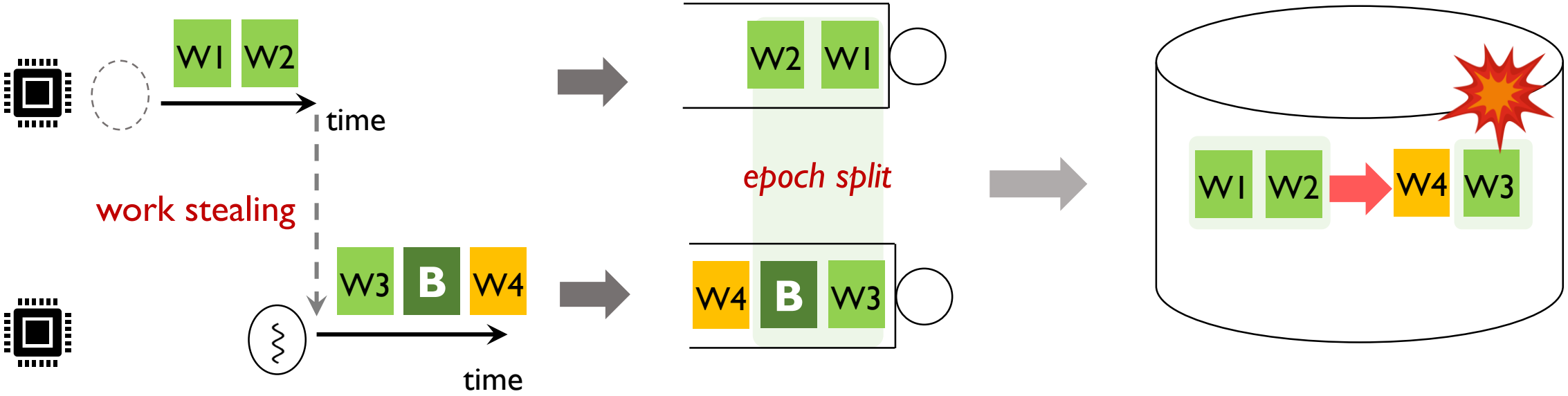
- Stream ID: process ID
- Epoch ID is managed independently for each stream.



Guarantee Intra-Stream Storage Order

{W1 W2 W3 } → {W4}

Epoch 1 Epoch 2

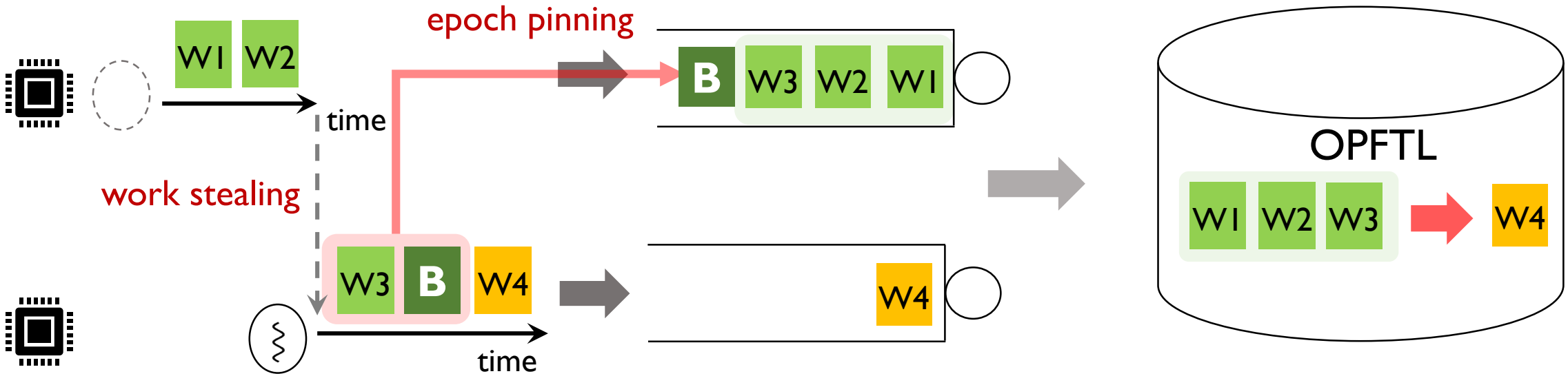


Epoch Pinning

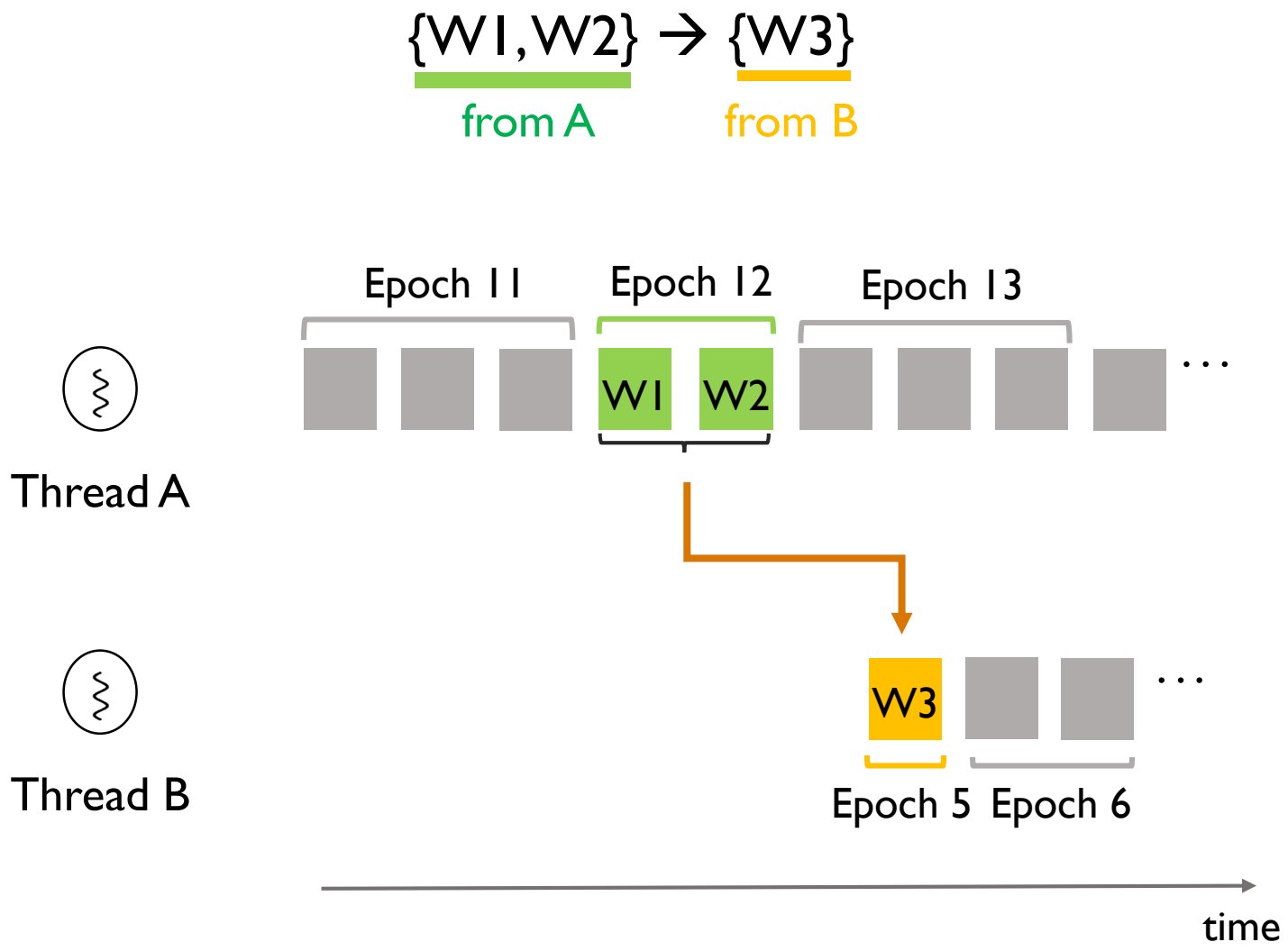
Guarantee the write requests at the same epoch to be placed in the same request queue

{W1 W2 W3 } → {W4}

Epoch 1 Epoch 2

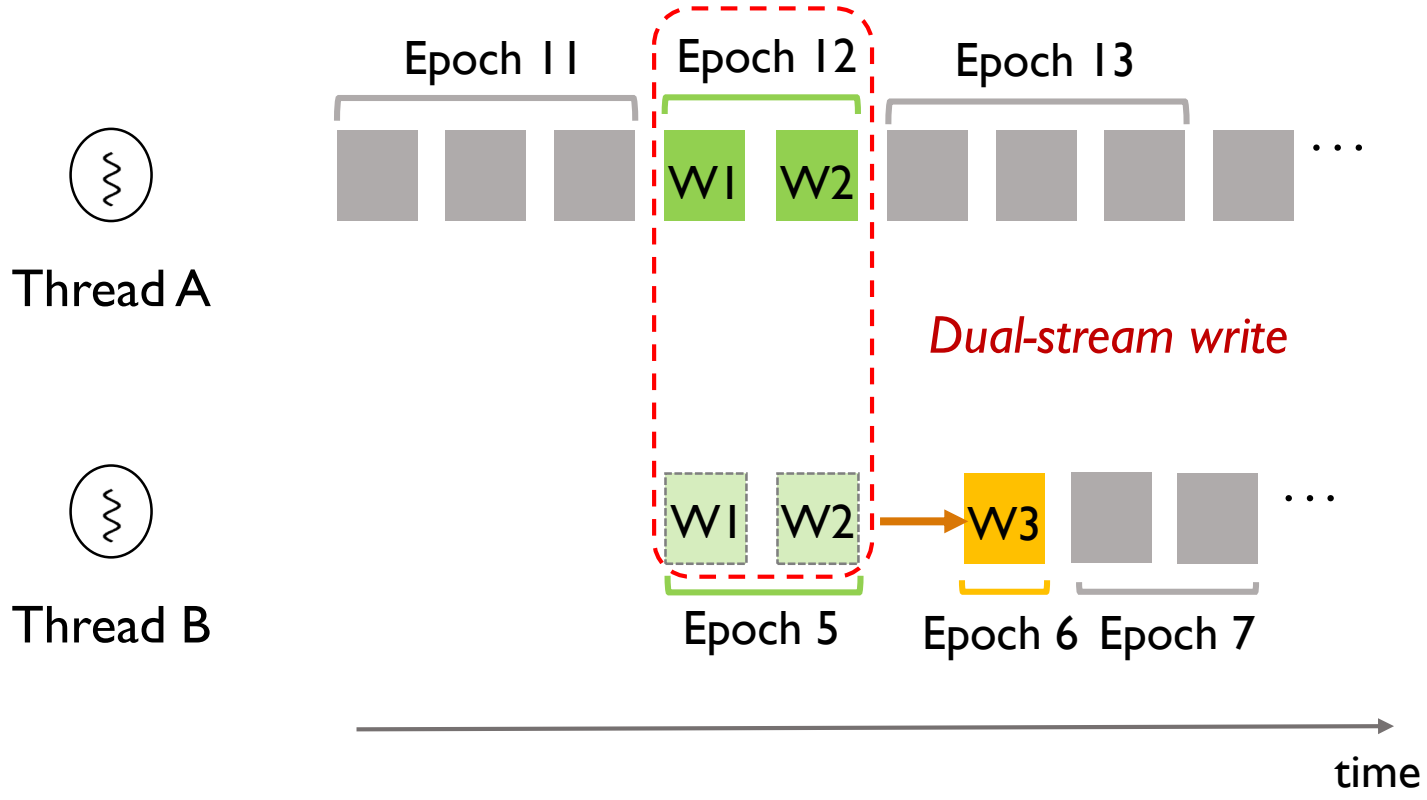


Guarantee Inter-Stream Storage Order



Dual-Stream Write

Dual-stream write: belongs to the two streams simultaneously.

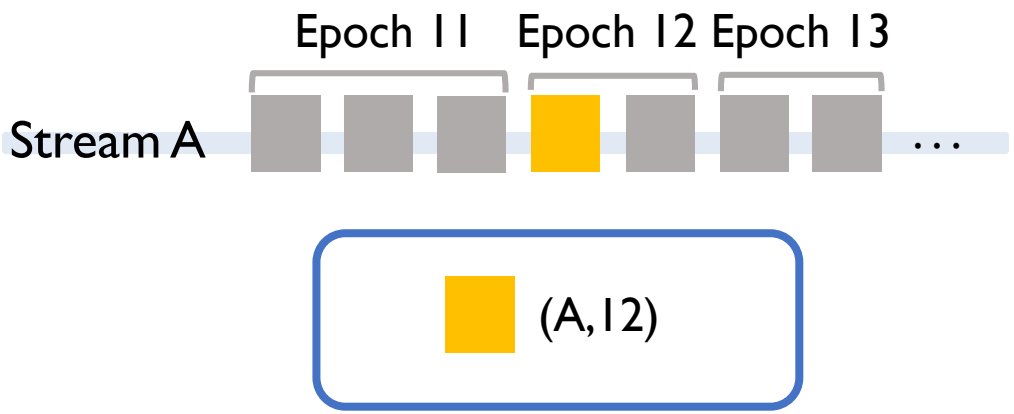


Dual-Stream Write (Cont.)

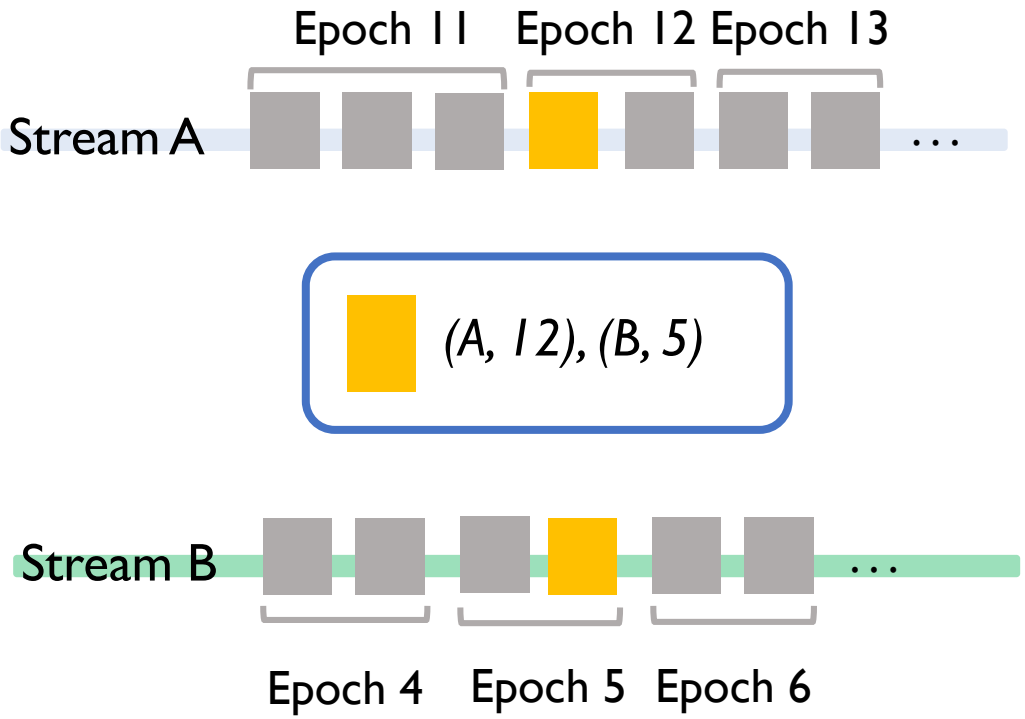
Dual-stream write has *two pairs of* stream id, epoch id

- Assign secondary (stream id, epoch id)

Single-stream write

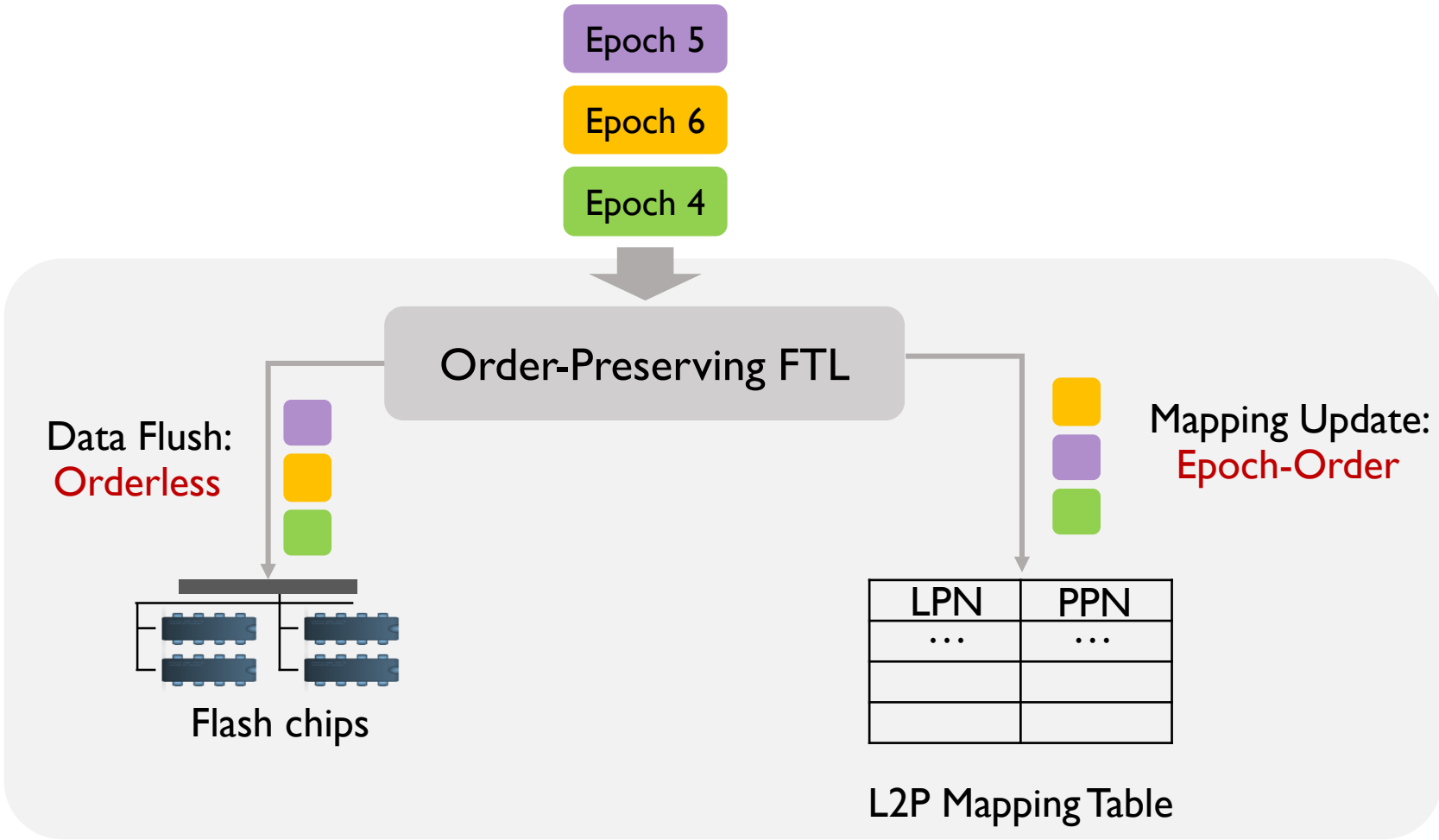


Dual-stream write



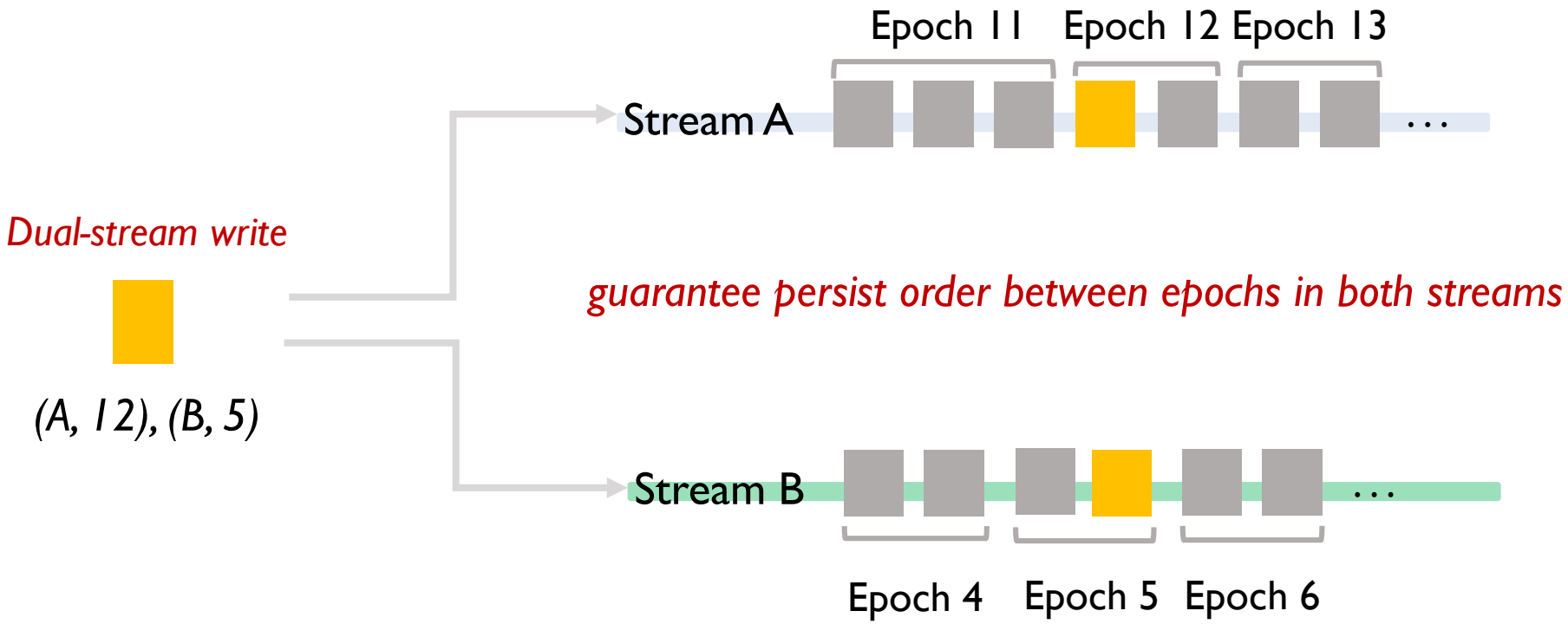
Order-Preserving FTL

- Ensure persist order between epochs within each stream

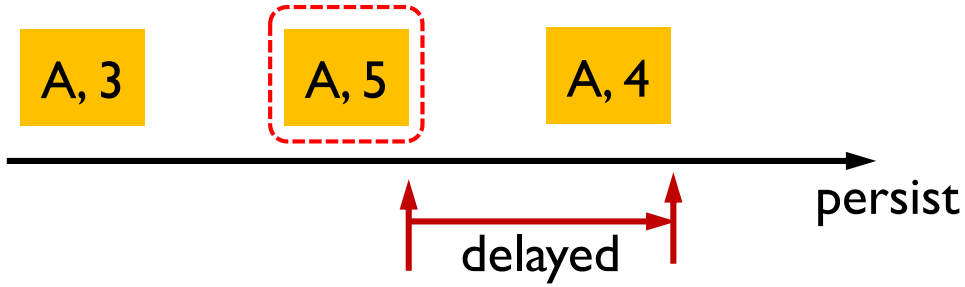


Storage Order in two streams

- Ensure inter-stream storage order along with dual-stream writes.
- For dual-stream writes, update the mapping table only if both streams meet ordering constraints.



Sibling-Aware Delayed Mapping

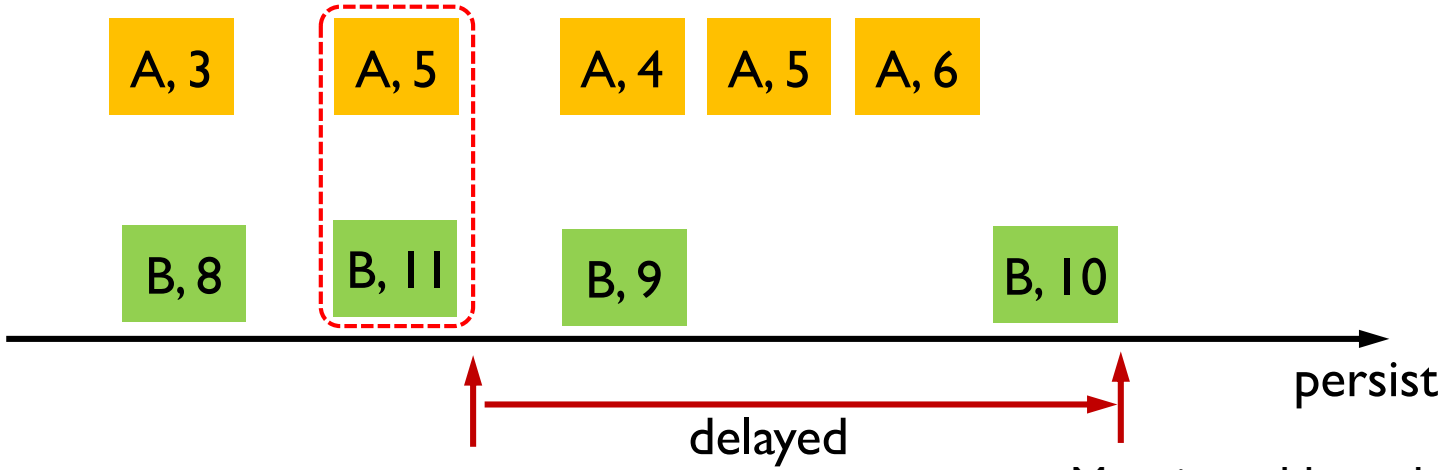


LPN	PPN
...	...

(A, 3)
(A, 4)
(A, 5)

Mapping table update for (A, 5)

dual-stream write



LPN	PPN
...	...

(A, 3)
(A, 4)
(B, 8)
(B, 9)
(B, 10)
(A, 5), (B,11)

Mapping table update for (A, 5), (B,11)

Outlines

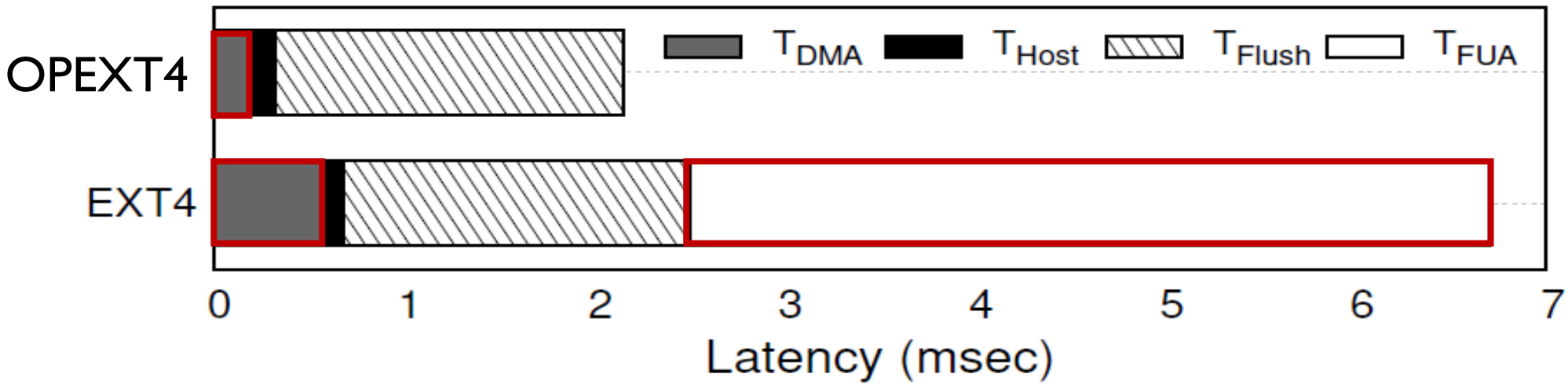
- Background & Motivation
- Problem Formulation
- Design
- Evaluation
- Conclusion

Evaluation

- CPU : Intel Xeon Gold 6320 (2.1 GHz, 2 Socket X 20 core = 40 core)
- Memory : 512GB DRAM
- Storage: Samsung 980 Pro (NVMe)
- OS (Kernel)
  - CentOS 7.4 (Linux Kernel 5.18.18)
- Filesystem: EXT4, BarrierFS with single command queue, OPIMQ w/ OPEXT4, MQFS
- Workloads: Filebench varmail, Sysbench OLTP insert, Dbench

Latency: fsync() system call

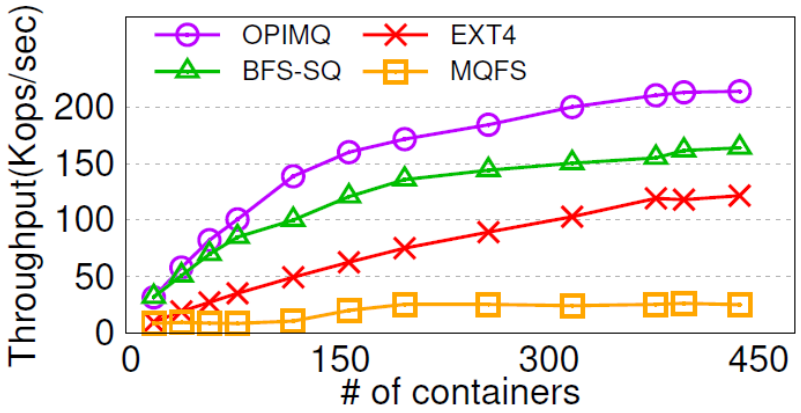
Workload: FIO 4KB write followed by fsync().



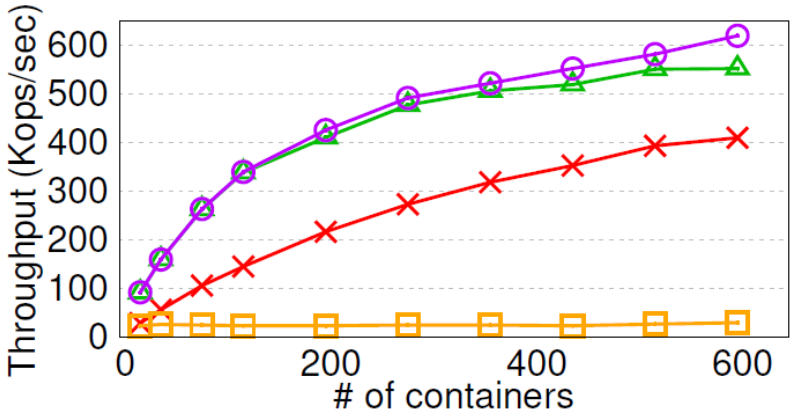
OPIMQ mitigates the overhead of storage order gurantee.

- Entire latency: $^1/_3$ of that EXT4
- DMA Latency: $^1/_2$ of that EXT4
- *FUA* accounts for 63% of the total.

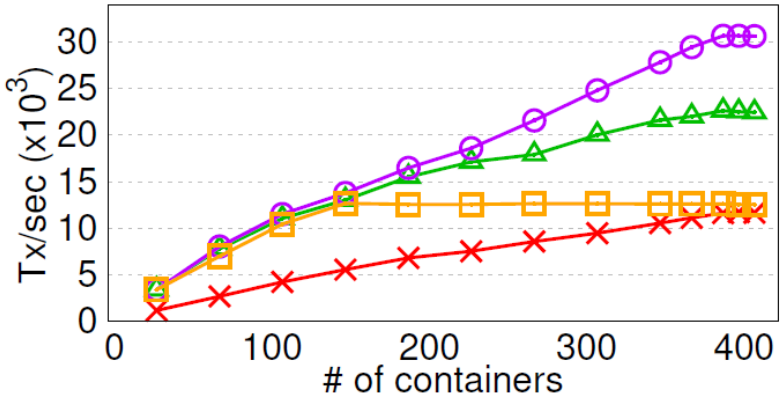
Macro Benchmarks



Filebench varmail



dbench

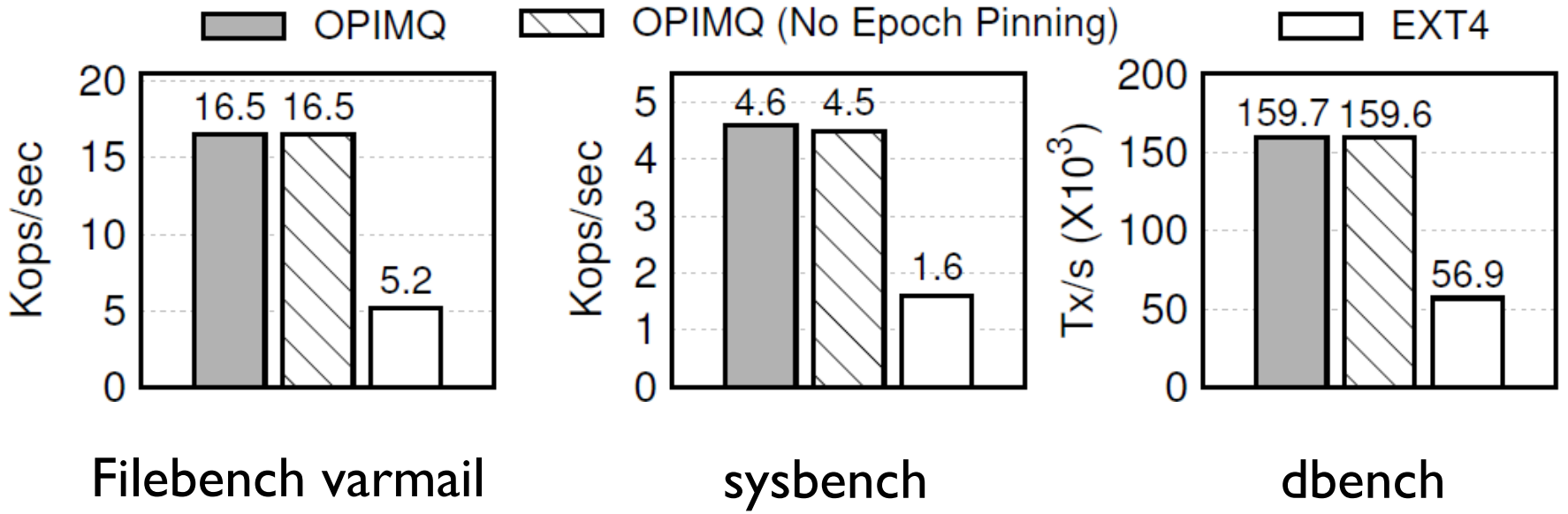


Sysbench-OLTP insert on MySQL

OPIMQ outperforms the vanilla Linux I/O stack by 2.9 x, 2.8 x, and 2.9 x under varmail, Dbench, and Sysbench.

Epoch Pinning Overhead

With 40 concurrent docker containers



Filebench varmail | sysbench | dbench

Epoch pinning does not cause any significant overhead

Overhead of OPFTL

- Cosmos OpensSSD (230 Gbyte, 8 channels, and 1 core)
- 4KByte random write() followed by fsync()

	Page Mapping FTL	OPFTL
Throughput (Kops/sec)	1.91	1.89
Latency (usec)	2.65	2.65

The overhead of ensuring the storage order in OPFTL is negligible.

Outlines

- Background & Motivation
- Problem Formulation
- Design
- Evaluation
- Conclusion

Conclusion

- We propose OPIMQ, Order-Preserving I/O stack for Multi-Queue Block Device and OPFTL, Order-Preserving Flash Translation Layer.
- OPIMQ opens up an new avenue for the cache barrier command to be employed in the storage products.
- Open-sourced at https://github.com/ESOS-Lab/OPIMQ.git

Thank you!

Any Questions?