Crash Consistent Non-Volatile Memory Express

Xiaojian Liao, Youyou Lu, Zhe Yang, Jiwu Shu

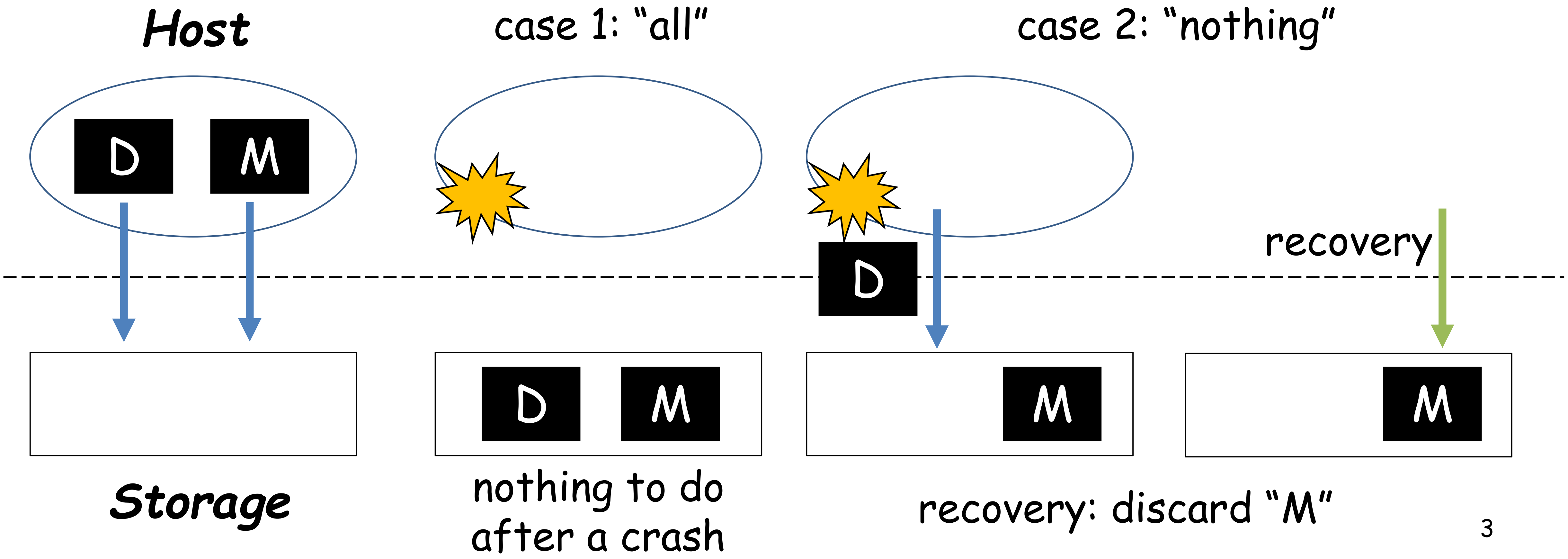
Tsinghua University

Agenda

- Background and Motivation
- ccNVMe Design and Implementation
- Evaluation
- Conclusion

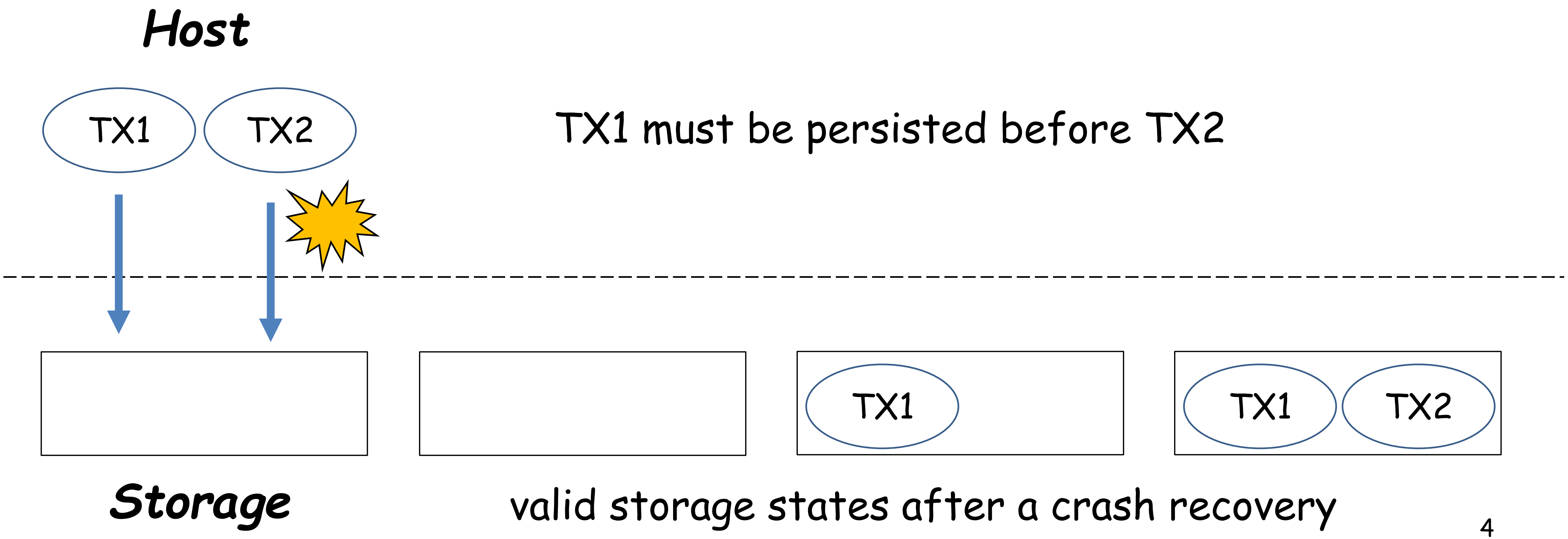
Background: crash consistency

Atomicity ("all" or "nothing") of a single operation that updates multiple blocks despite a sudden system crash



Background: storage order

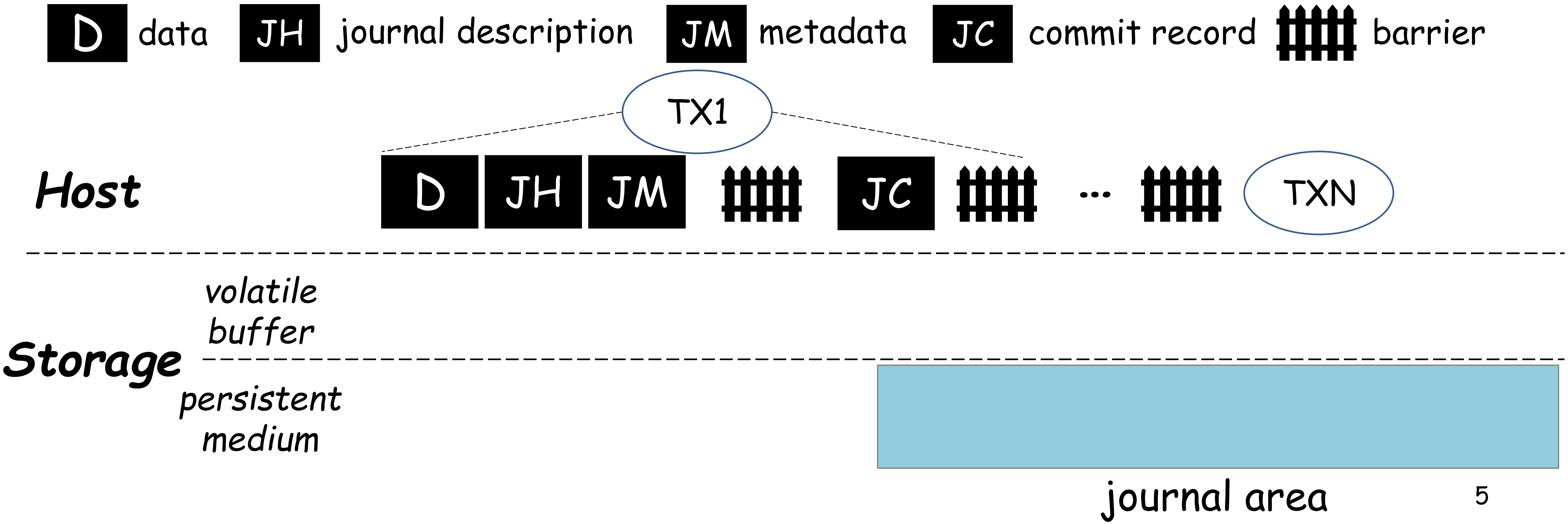
Persistence order of multiple individual operations (transactions) despite a sudden system crash



Transaction and journaling

Most existing storage systems use journaling (or write-ahead log) to achieve crash consistency and storage order.

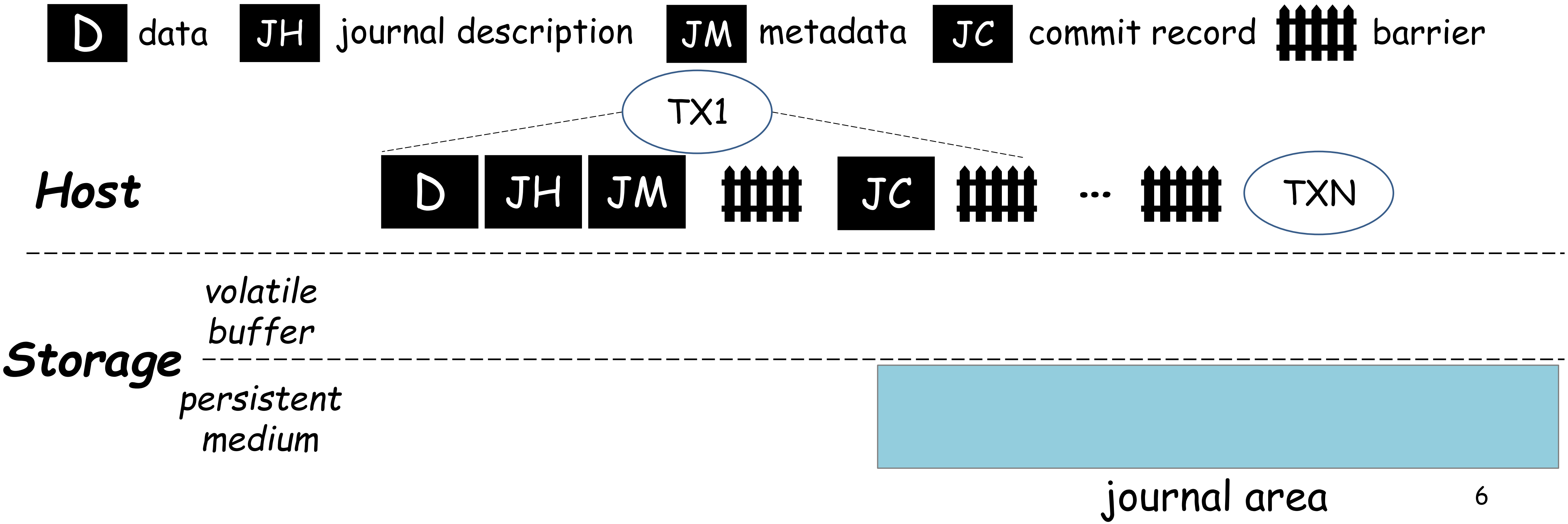
Workload: create and write a new file, data journaling mode



Transaction and journaling

Most existing storage systems use journaling (or write-ahead log) to achieve crash consistency and storage order.

Workload: create and write a new file, data journaling mode

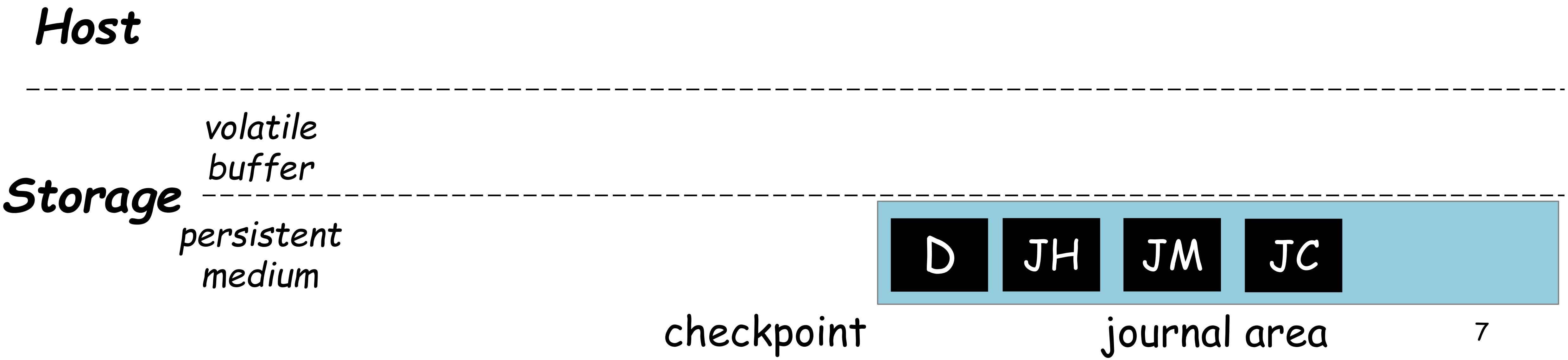


Transaction and journaling

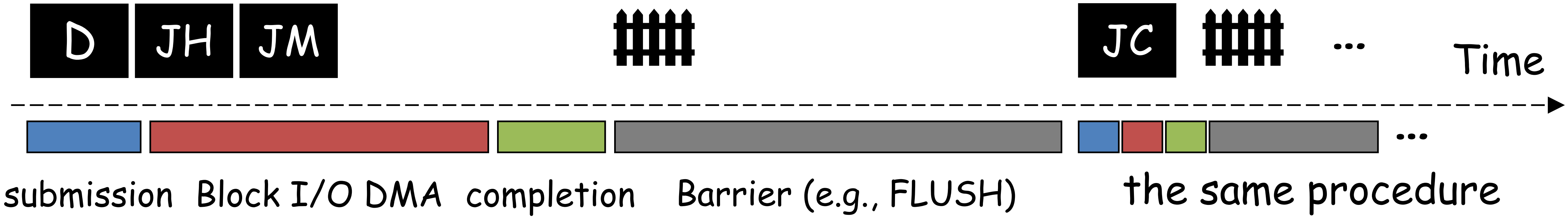
Most existing storage systems use journaling (or write-ahead log) to achieve crash consistency and storage order.

Workload: create and write a new file, data journaling mode

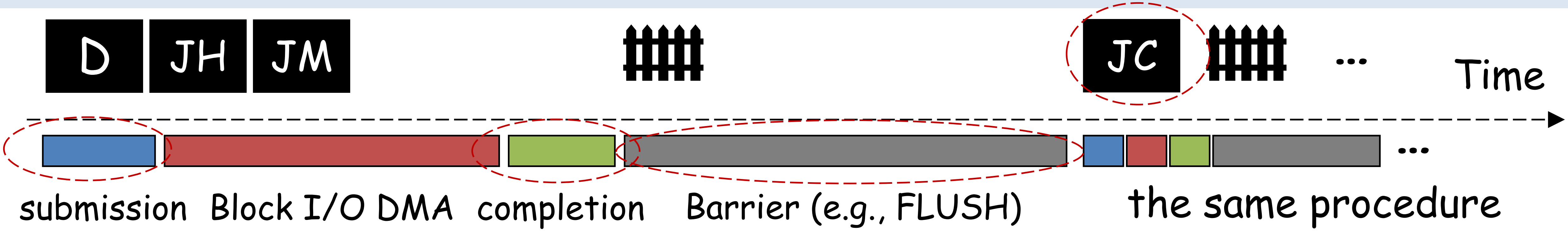
D data **JH** journal description **JM** metadata **JC** commit record [barrier] barrier



Motivation: issues of journaling

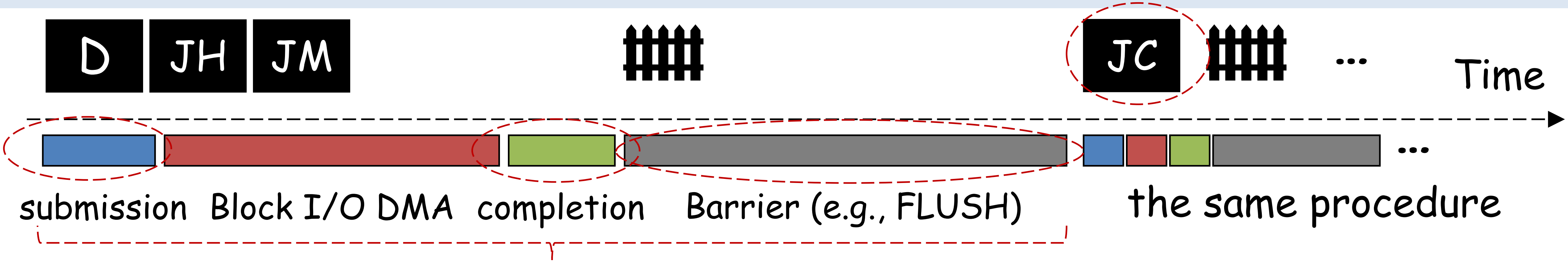


Motivation: issues of journaling



- Issue 1: extra storage/PCIe traffic
  - extra MMIOs of submission and completion due to per-request doorbells
  - irrelevant blocks incurred by the device-wide FLUSH
  - extra commit record (JC) generated by journaling to ensure atomicity

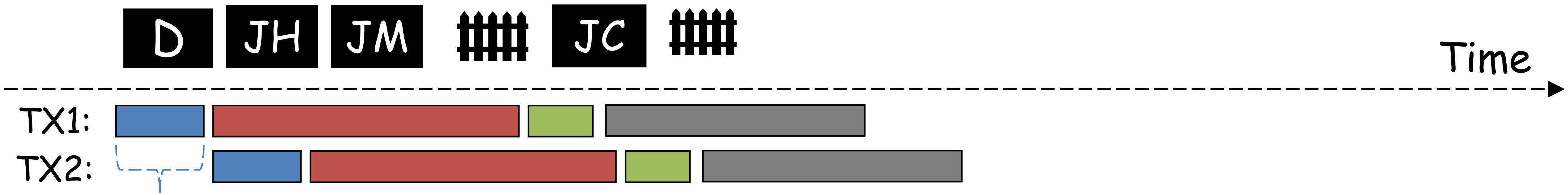
Motivation: issues of journaling



- Issue 1: extra storage/PCIe traffic
  - extra MMIOs of submission and completion due to per-request doorbells
  - irrelevant blocks incurred by the device-wide FLUSH
  - extra commit record (JC) generated by journaling to ensure atomicity
- Issue 2: serialization of ordered transactions
  - pose long latency to each transaction, worsening issue 1
  - conflate atomicity and storage order with durability

Our solution: ccNVMe

Generic storage protocol that provides crash consistency, per-hardware-queue storage order and high performance.



fatomic: crash consistency and ordering with only two MMIOs over PCIe!

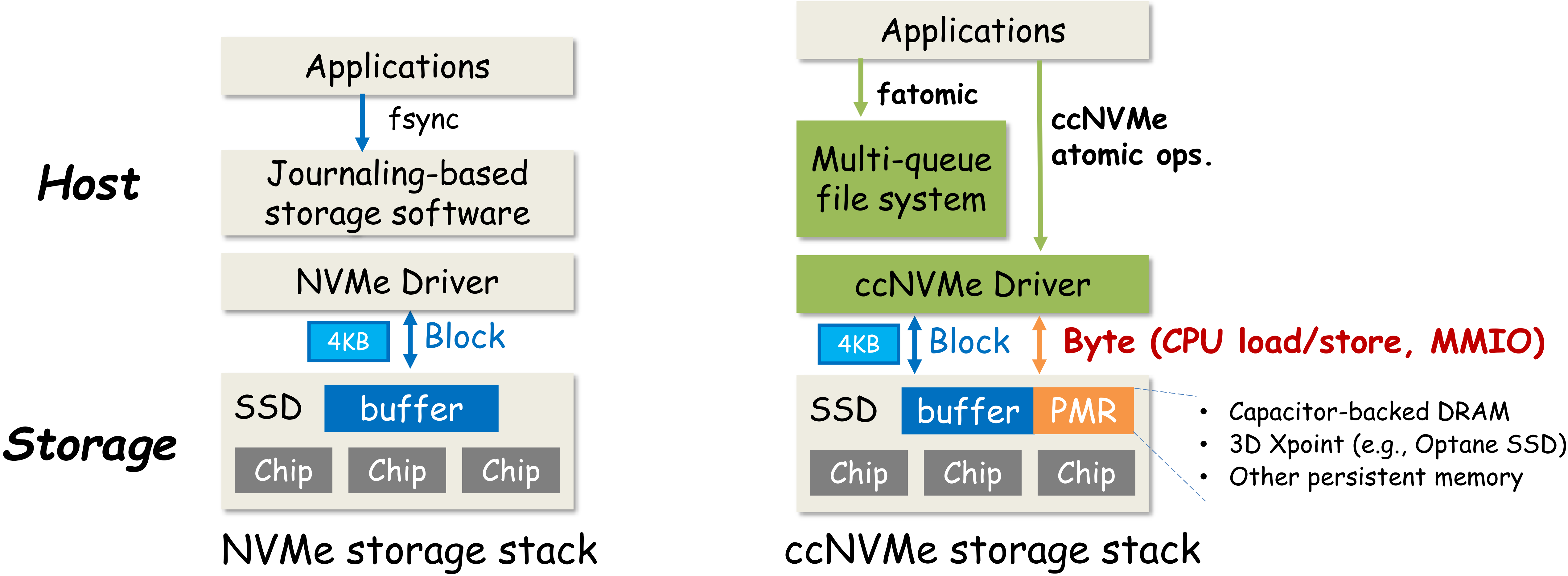
- Advantage 1: reduce unnecessary storage/PCIe traffic
  - remove commit record (JC)
  - remove one expensive device-wide FLUSH
  - reduce MMIOs via transaction-aware MMIOs and doorbells
- Advantage 2: parallelize atomic and ordered transactions
  - separate atomicity from durability

Agenda

- Background and Motivation
- ccNVMe Design and Implementation
- Evaluation
- Conclusion

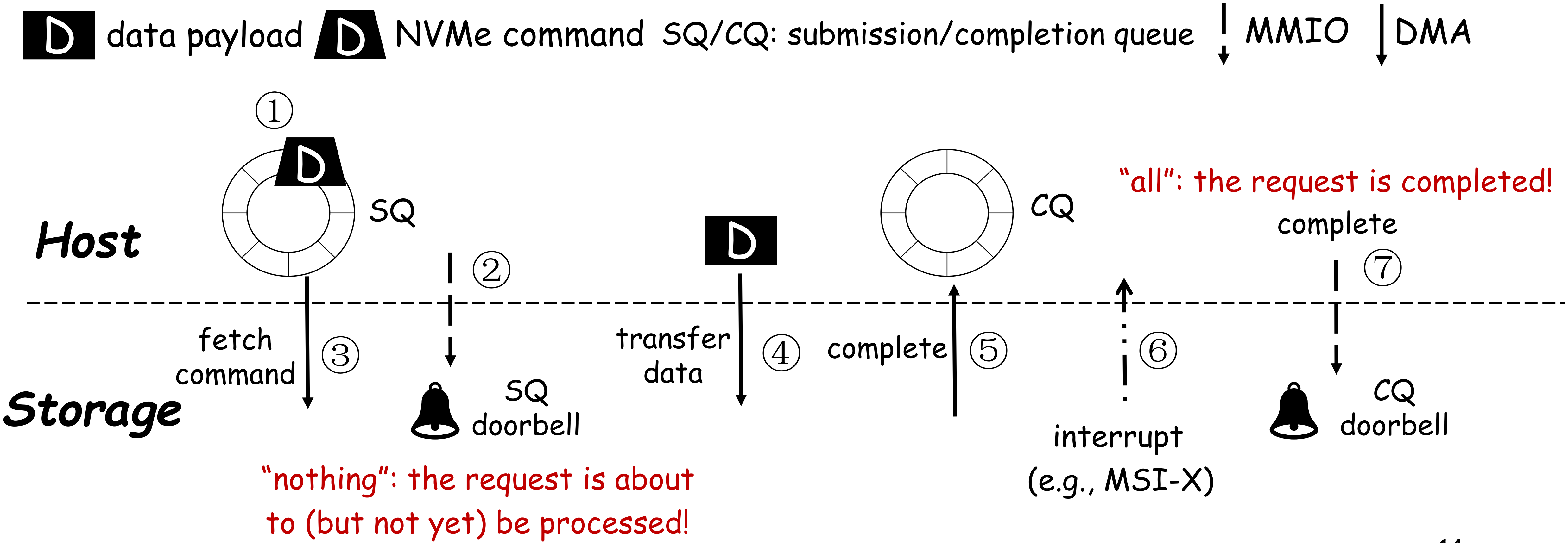
ccNVMe design overview

ccNVMe is designed as an extension of NVMe (Non-Volatile Memory Express) atop PMR (persistent memory reigon)-enabled SSDs.



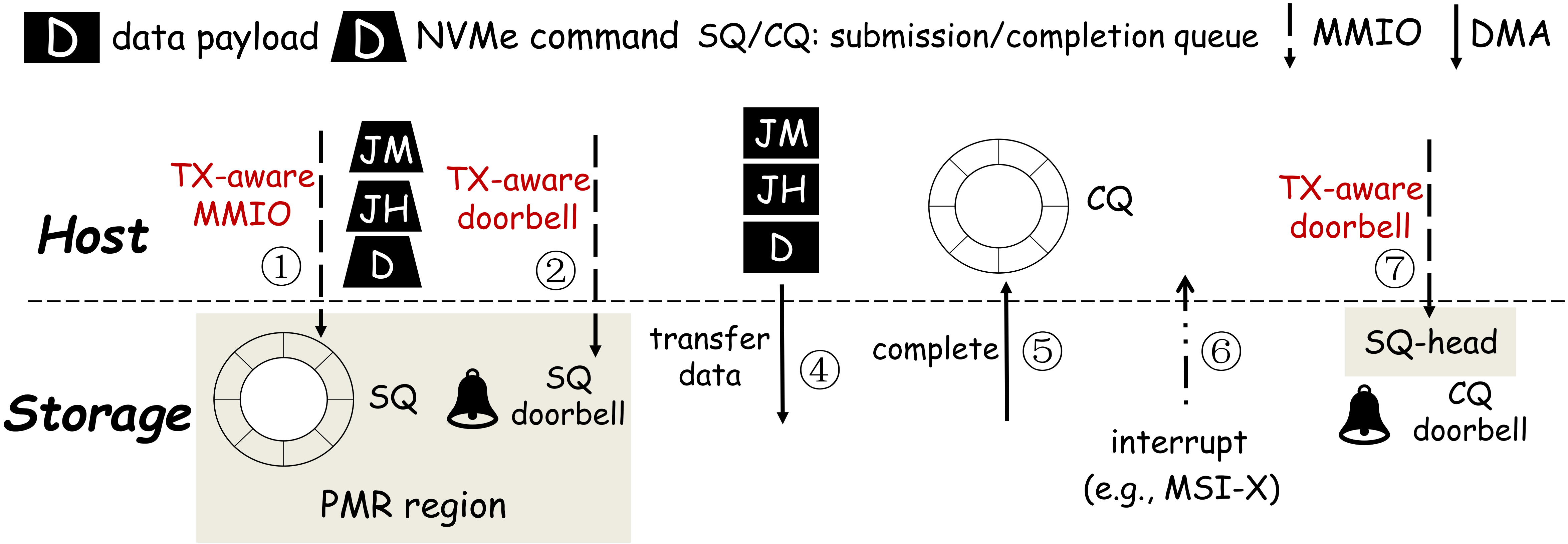
ccNVMe key insights from NVMe

Key observation: the SQ and doorbells naturally track the life cycle (e.g., submitted or completed) of each request!



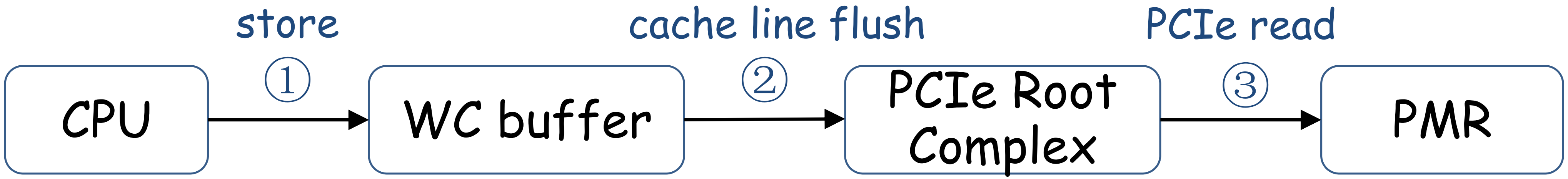
ccNVMe work flow

Key idea: let crash consistency and storage order take the free rides of data dissemination mechanism of the original NVMe.

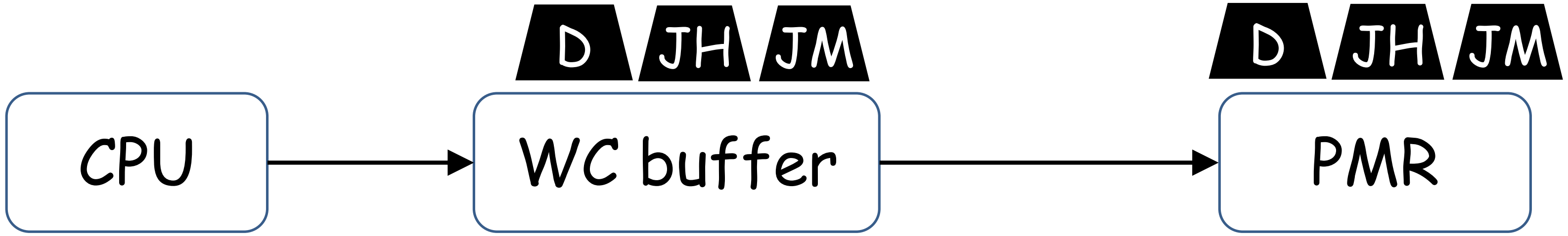


TX-aware MMIO

Persistent MMIO write to PMR



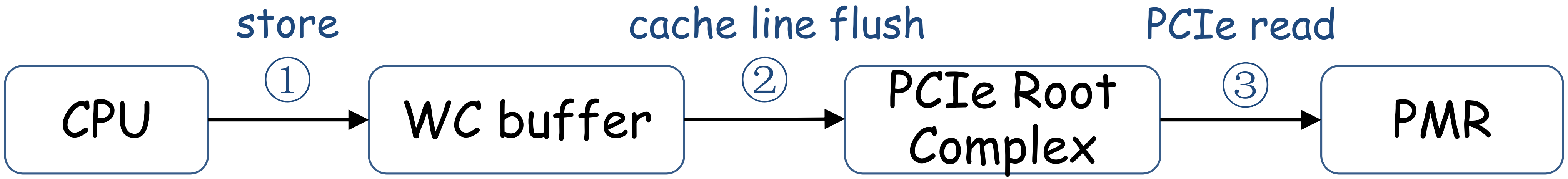
TX-aware MMIO: batching MMIOs of each transaction



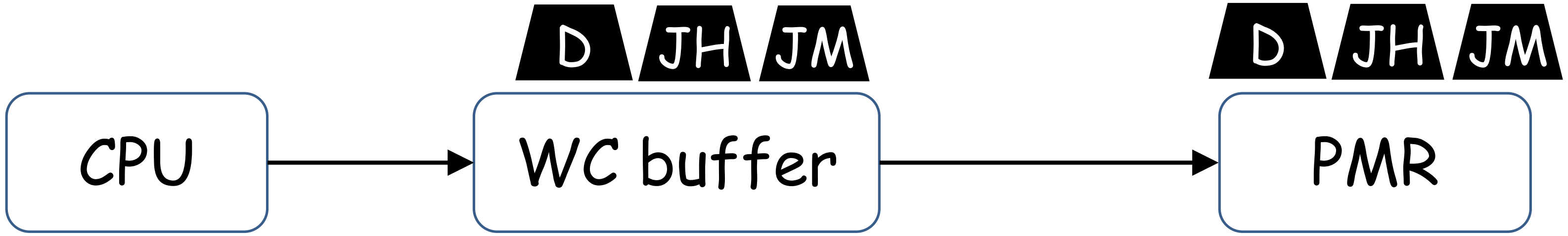
step 1. store D; store JH; store JM step 2. flush (D, length of (D+JH+JM)); PCIe read

TX-aware MMIO

Persistent MMIO write to PMR

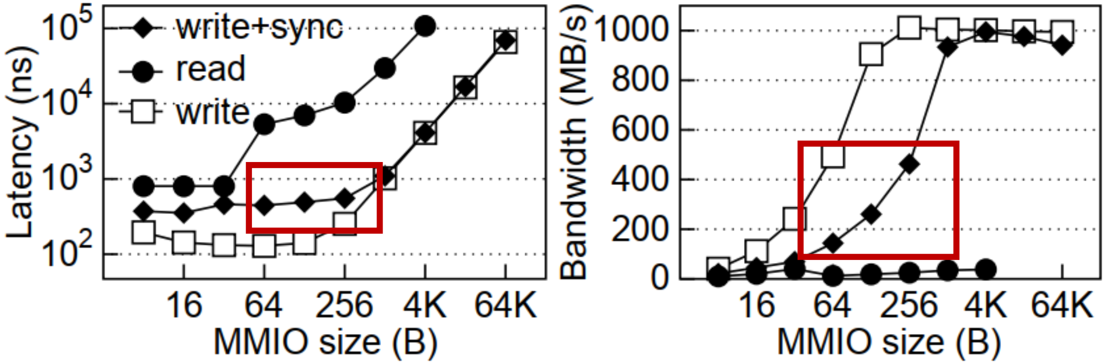


TX-aware MMIO: batching MMIOs of each transaction



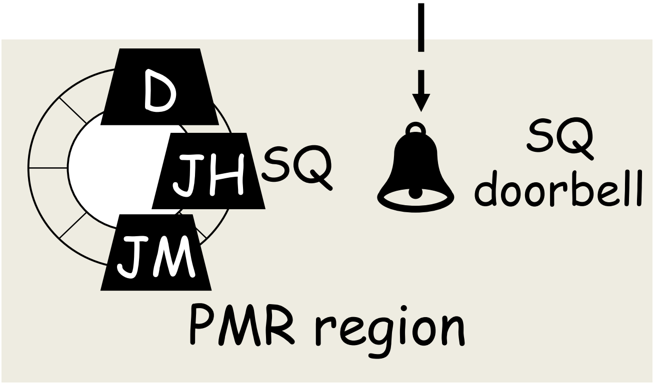
step 1. store D; store JH; st … +JM)); PCIe read

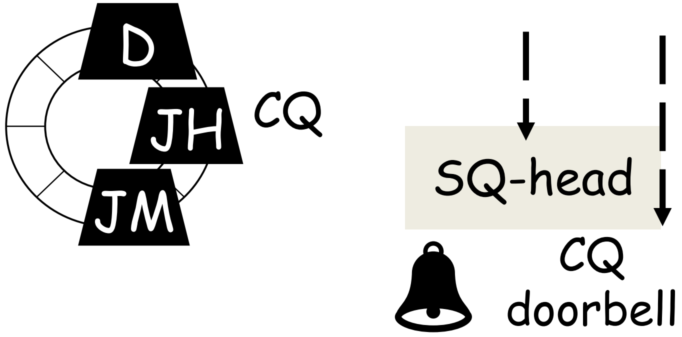
larger persistent MMIO
higher access efficiency
(details in paper)



TX-aware doorbell

One SQ doorbell and CQ doorbell for each transaction; let the requests of each transaction reach the same state.



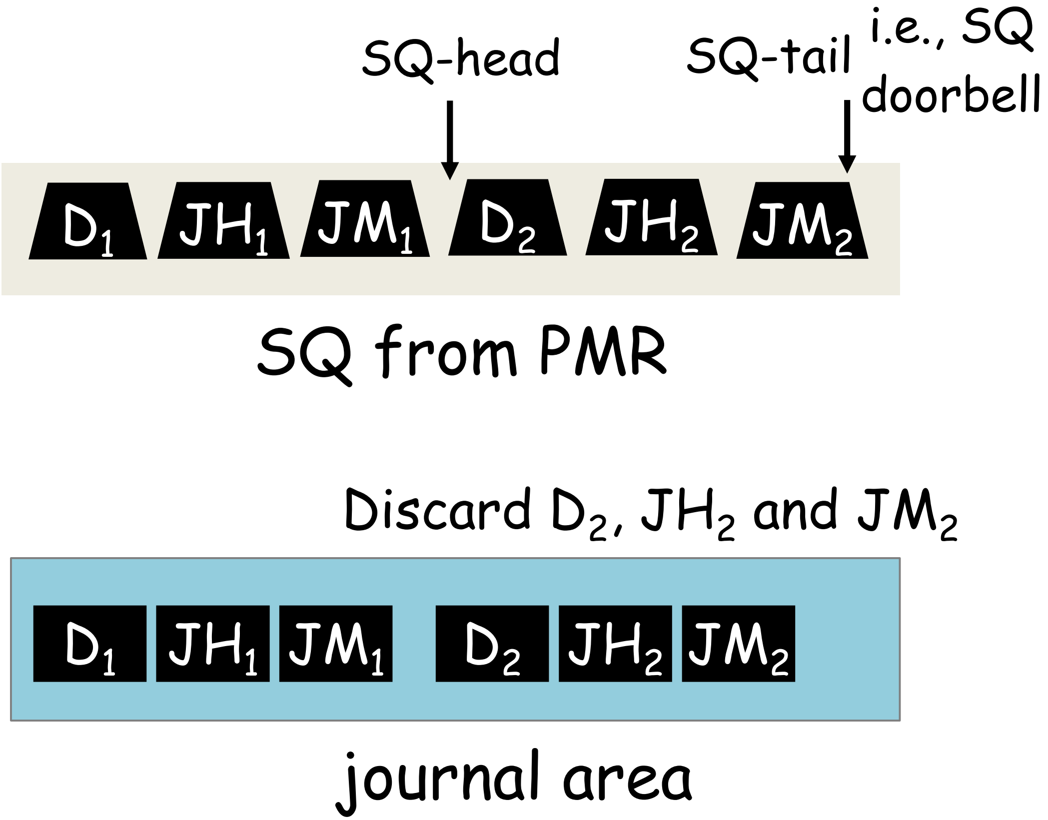


- ✓ All requests (D, JH, JM) are about to be processed! ("nothing")
- ✓ Reduce the SQ doorbell MMIO
- ✓ Remove the commit record; SQ doorbell as a commit point

- ✓ All requests (D, JH, JM) are completed! ("all")
- ✓ Record the SQ head value
- ✓ Reduce the CQ doorbell MMIO

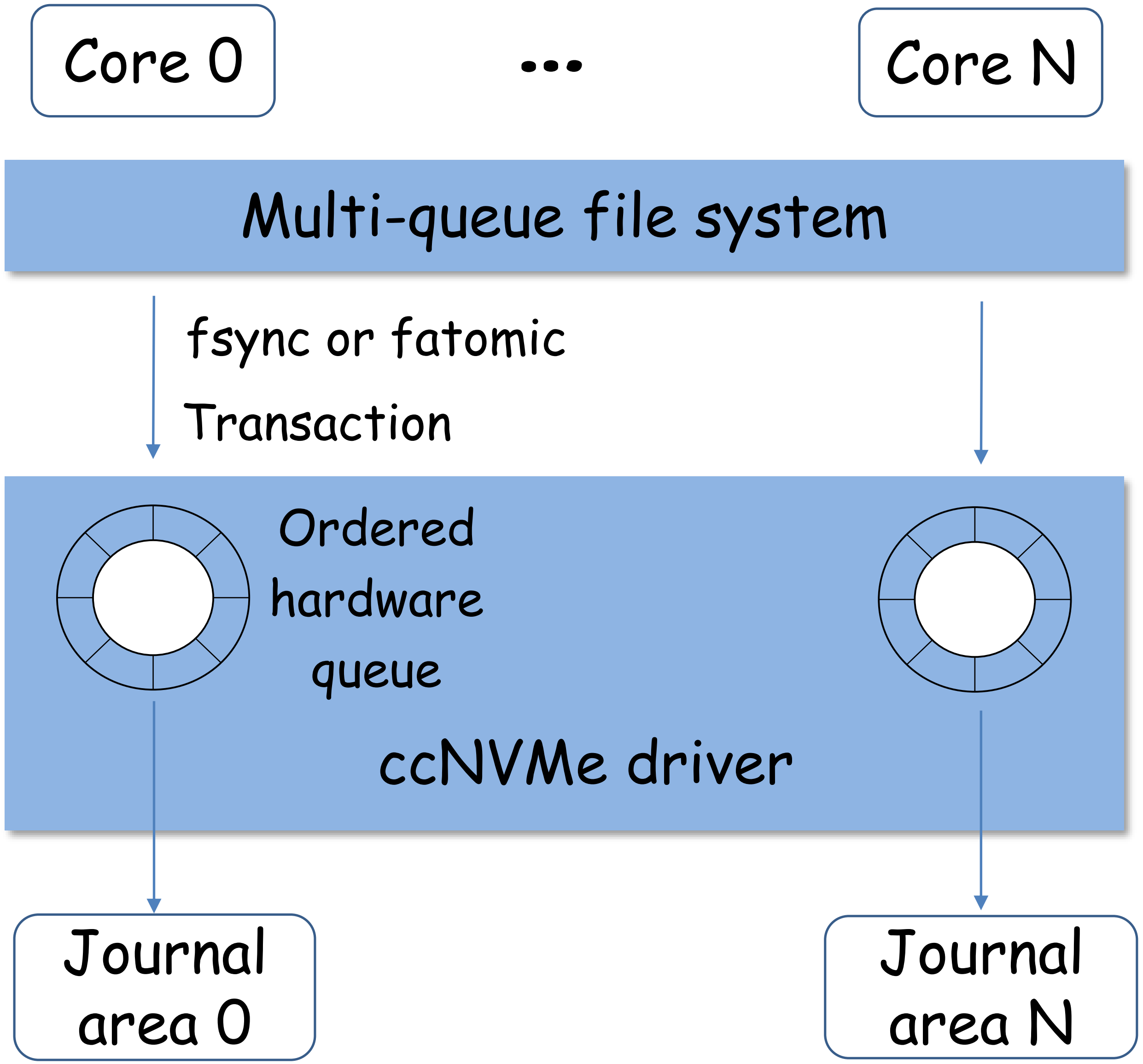
Crash recovery

ccNVMe provides non-atomic and out-of-order transactions to upper layer systems; upper layer systems handle these unfinished transactions, e.g., discard all for data journaling.



- crash consistency: tx-aware doorbell, transactions are submitted and completed atomically.
- storage order: in-order doorbells, doorbells of each hardware queue are set in order.

Multi-Queue File System



Global storage order among hardware queues

Per-hardware-queue storage order

Crash consistency

Other details in paper

- Details of ccNVMe commands, compatible with the original NVMe commands using reserved fields
- Metadata shadow paging to persist shard blocks in parallel
- Selective revocation to handle block reuse across multi-queue
- Implementation details

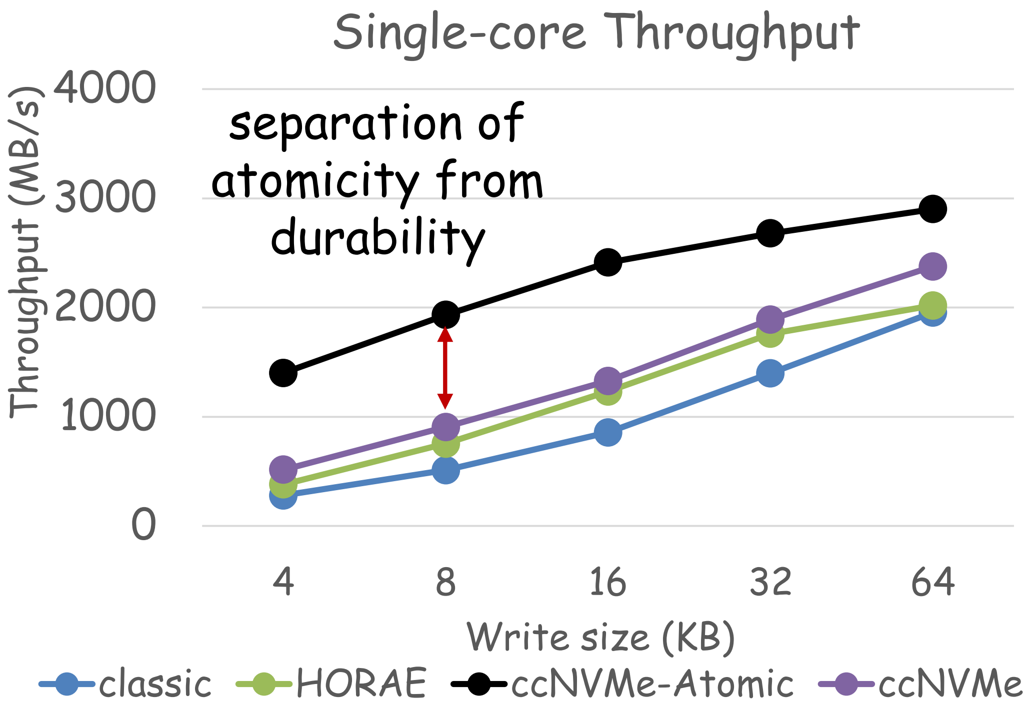
Evaluation

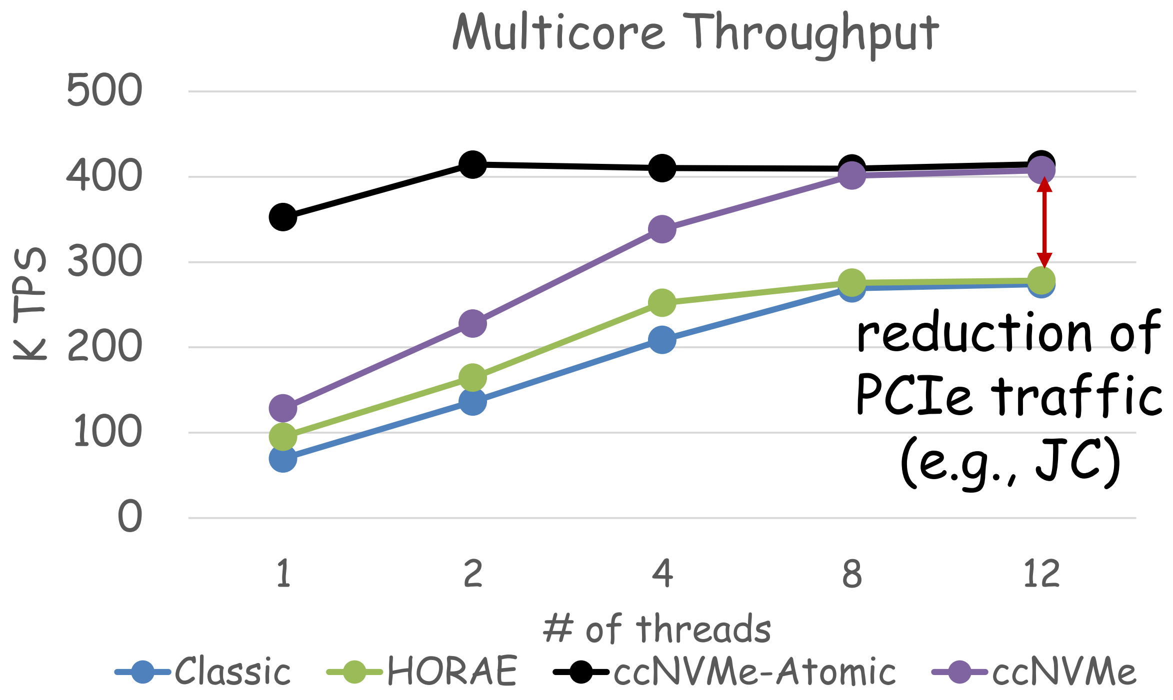
CPU	2 Intel E5-2680 V3 CPUs, each with 12 cores, totally 24 physical CPU cores
SSD	Intel 905P Optane, Intel P5800X Optane
Compared system	Linux vanilla kernel 4.18.20; classic journaling, HORAE[OSDI'20]; Ext4, HORAEFS[OSDI'20], Ext4-NJ (no-journal setup of Ext4)
Workloads	• Transaction performance; • File system performance; (see paper) • Application performance; • Understanding the performance; (see paper) • Crash consistency; (see paper)

Transaction performance

Tested SSD: Intel P5800X

Workload: each thread persists independent transactions



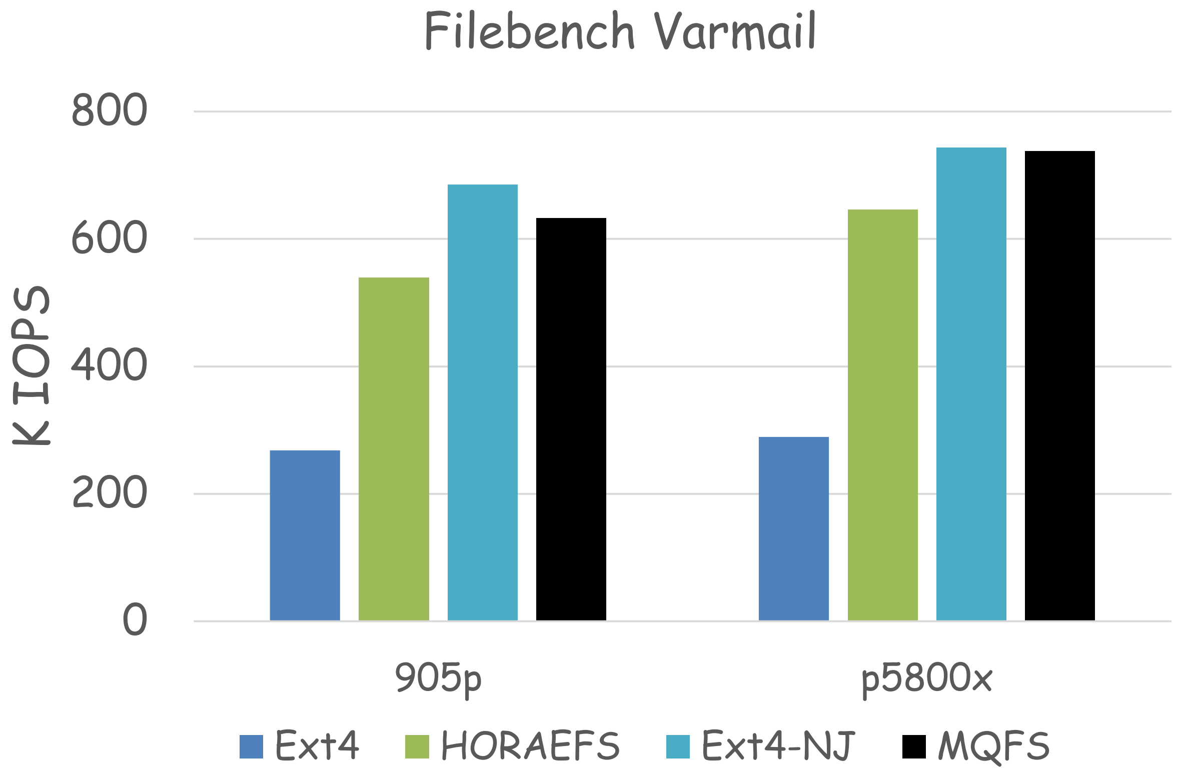


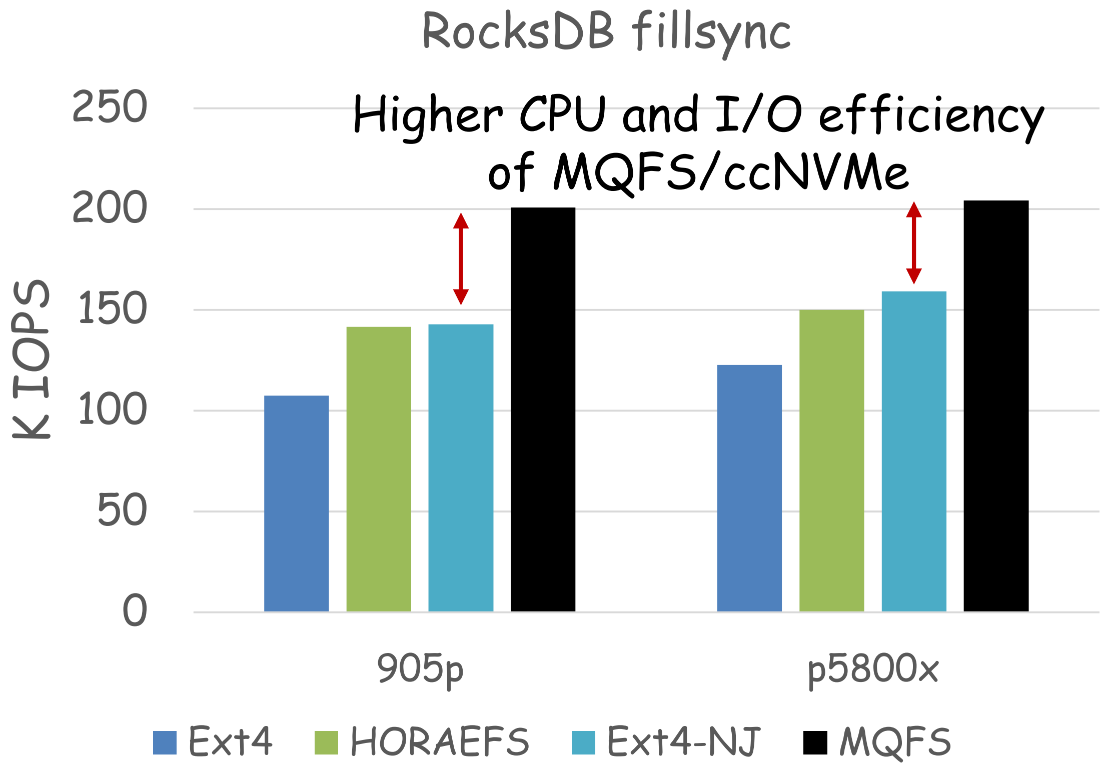
ccNVMe-Atomic = 2 x ccNVMe = 2.2 x HORAE = 3 x Classic

ccNVMe-Atomic = 1.6 x ccNVMe = 2.2 x HORAE = 2.6 x Classic

Application performance

MQFS: no-journal file system atop ccNVMe, this work, provide crash consistency and storage order.





MQFS ≈ Ext4-NJ = 1.2/1.1 x HORAEFS =2.4/2.6 x Ext4

MQFS = 1.4/1.3 x Ext4-NJ = 1.4/1.4 x HORAEFS = 1.9/1.7 x Ext4

Conclusion

- ccNVMe: Crash Consistent Non-Volatile Memory Express
  - Provide generic transaction abstraction, crash consistency and storage order inside the standard storage protocol
  - Separate atomicity from durability to fully exploit the high concurrency (e.g., multiple deep queues) of modern high-performance NVMe SSDs
- MQFS: Multi-Queue File System
  - Upper layer storage software should reduce the CPU overhead to embrace the fast crash consistency and storage order of ccNVMe

Source code: https://github.com/thustorage/ccnvme

Q&A

Thank You!

Crash Consistent Non-Volatile Memory Express

Xiaojian Liao, Youyou Lu, Zhe Yang, Jiwu Shu

Tsinghua University

liao-xj17@mails.tsinghua.edu.cn
greatliaoxiaojian@gmail.com