Z-Journal: Scalable Per-Core Journaling

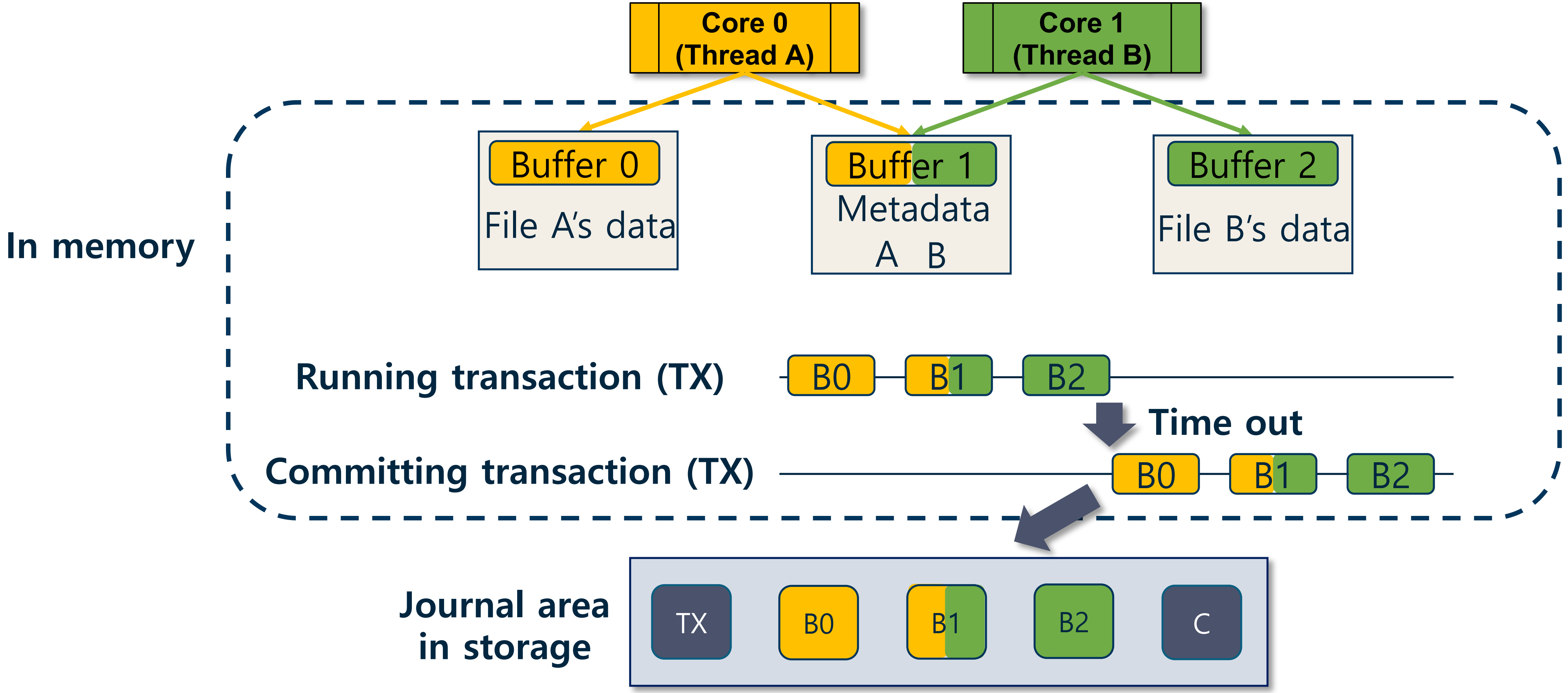
Jongseok Kim[1], Cassiano Campes[1],
Joo-Young Hwang[2], Jinkyu Jeong[1] and Euiseong Seo[1]

Sungkyunkwan University[1]

Samsung Electronics Co., Ltd.[2]

Conventional Journaling Scheme

- JBD2: Generic Journal Layer for Ext4 and Other FS



Core 0 (Thread A)

Core 1 (Thread B)

In memory

Buffer 0
File A's data

Buffer 1
Metadata
A B

Buffer 2
File B's data

Running transaction (TX)
B0 B1 B2

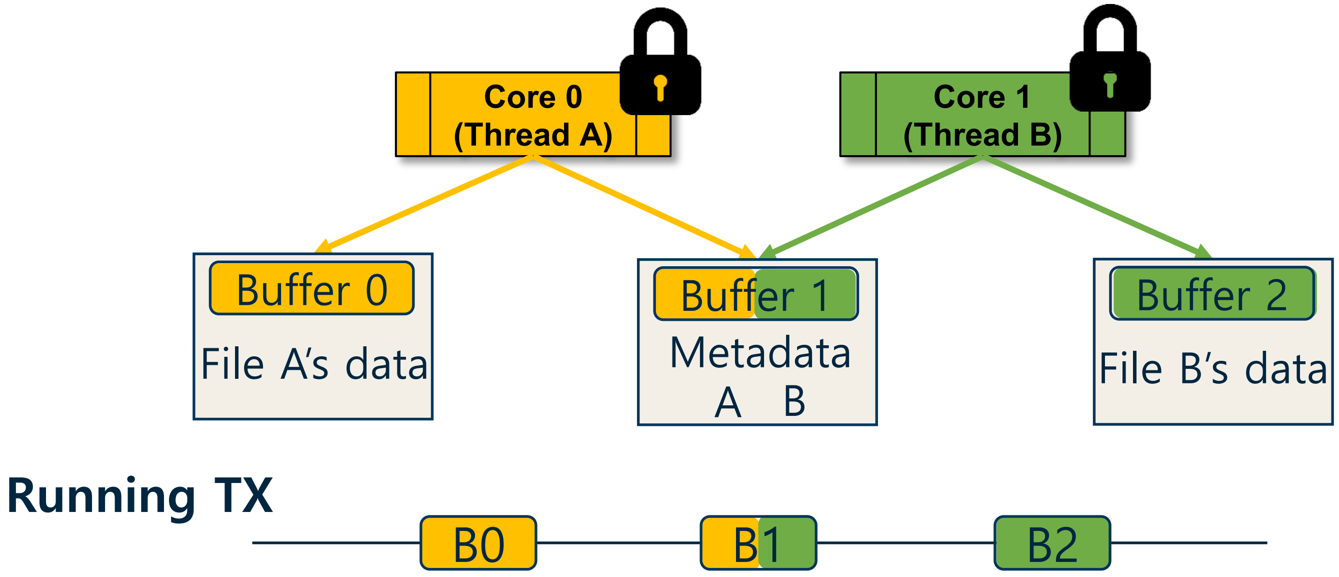
Time out

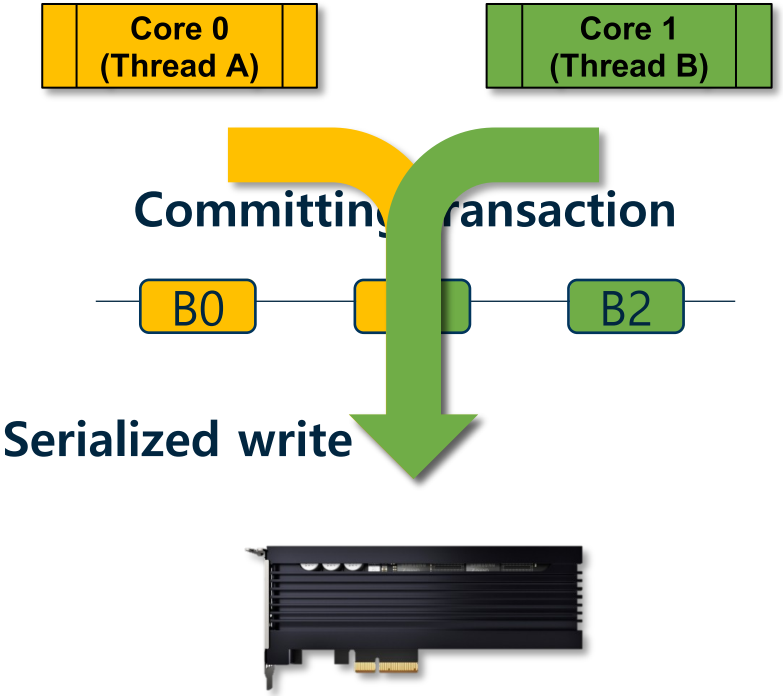
Committing transaction (TX)
B0 B1 B2

Journal area in storage
TX B0 B1 B2 C

Journal – A Scalability Bottleneck

- The lock acquisition for the running transaction
- Not to fully utilize the internal-parallelism provided by the modern NVMe SSDs





Related work

- **Parallelize a part of journal layer**
  - iJournaling [ATC '17]
  - High-performance transaction processing [FAST '18]

➡ **Inherent serialization from committing to a centralized journal remains**

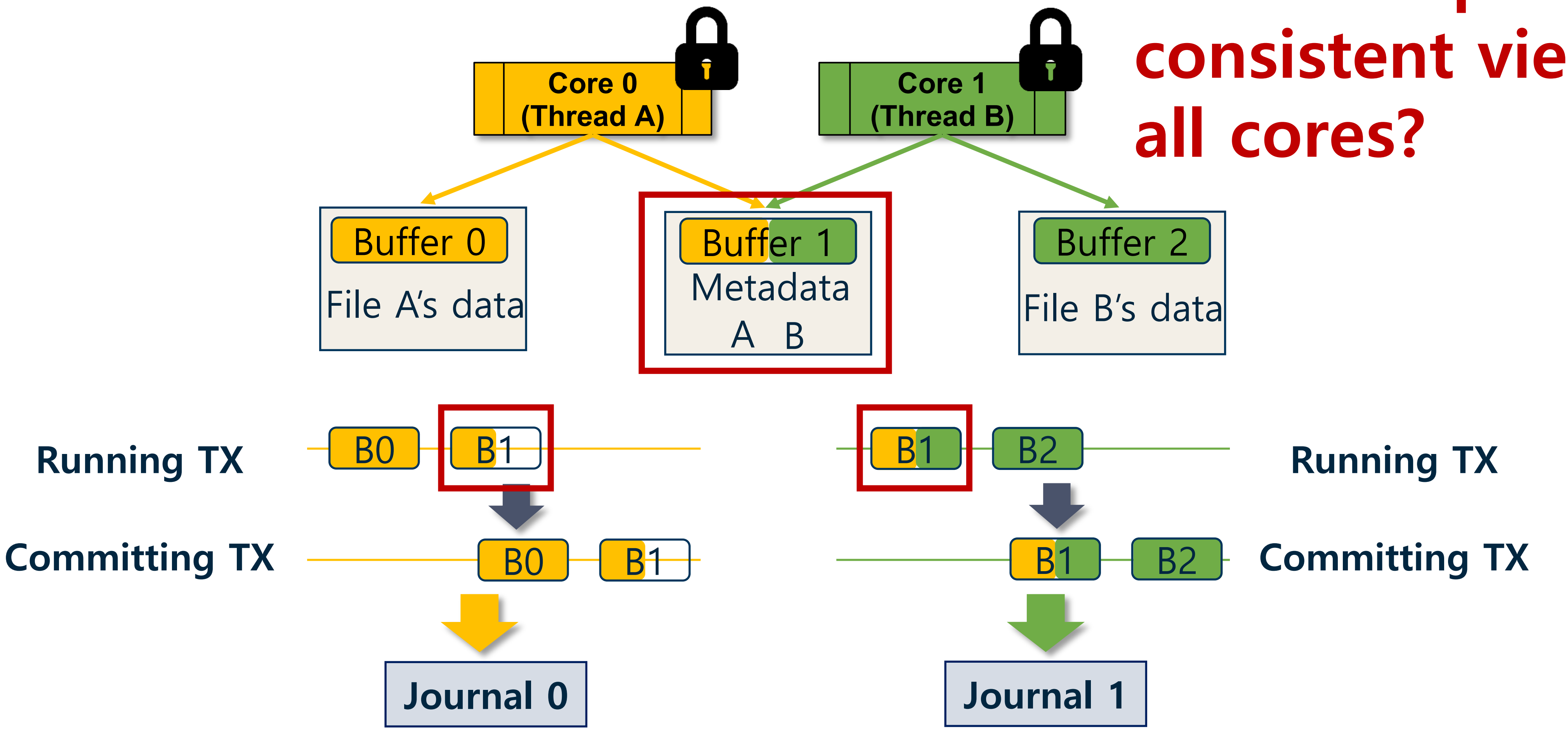
- **Fully-redesigned file systems for scalability**
  - SpanFS [ATC '15]
  - ScaleFS [SOSP '17]

➡ **Journal layer is tightly coupled with file system layer**
They cannot be directly applied to the existing file systems

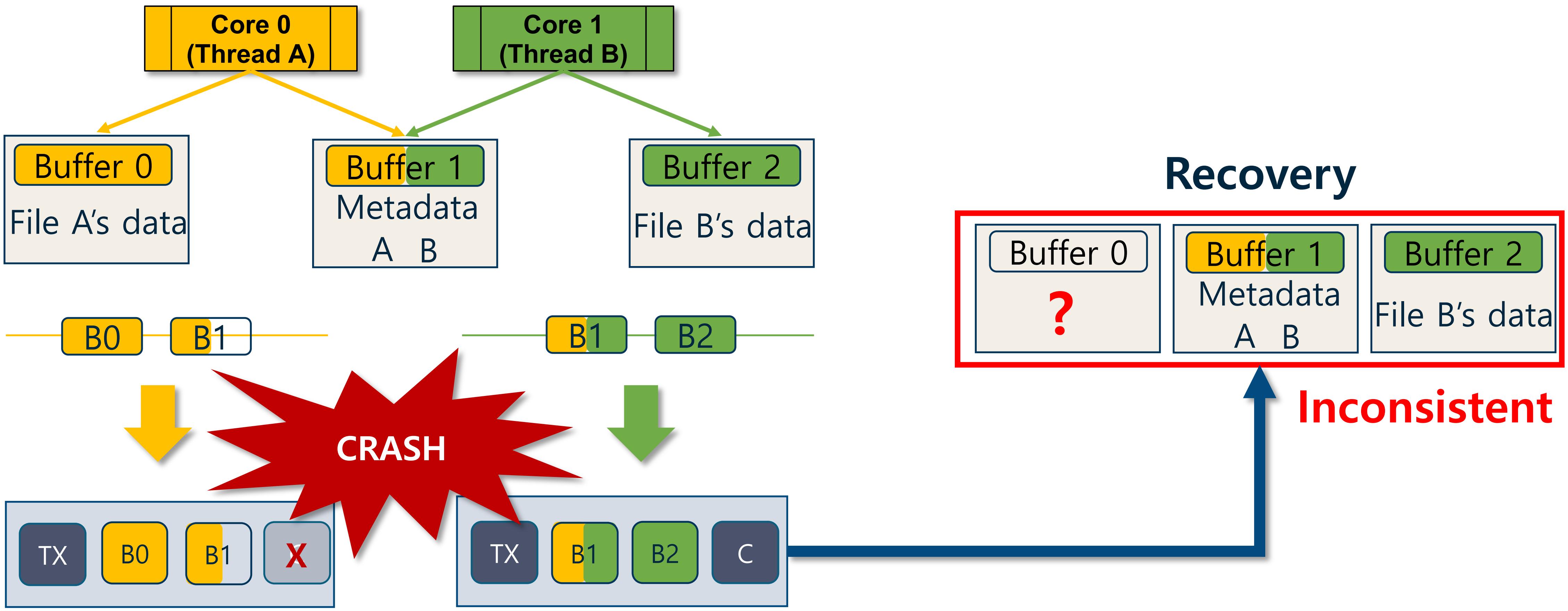
Intuitive Solution

- Per-core Journal

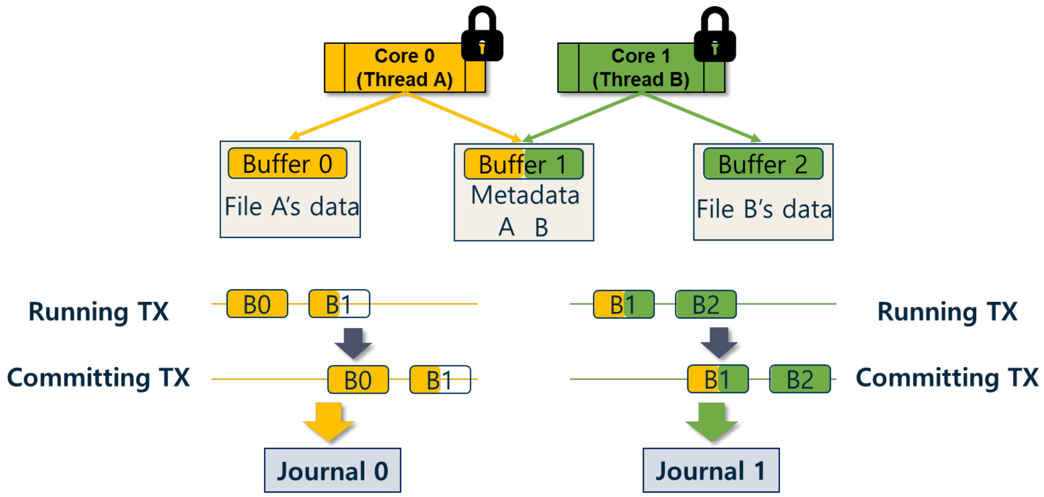
Does this provide consistent view to all cores?



Journal Coherence Problem



Our Approach: Z-Journal



Per-core Journaling

- Journal coherence commit
  - Order preserving transaction chaining
  - Proactive frozen copy
- Core [illegible] allocation
- Journal coherence checkpointing
- Recovery

Please refer to our paper

Journal coherence mechanism

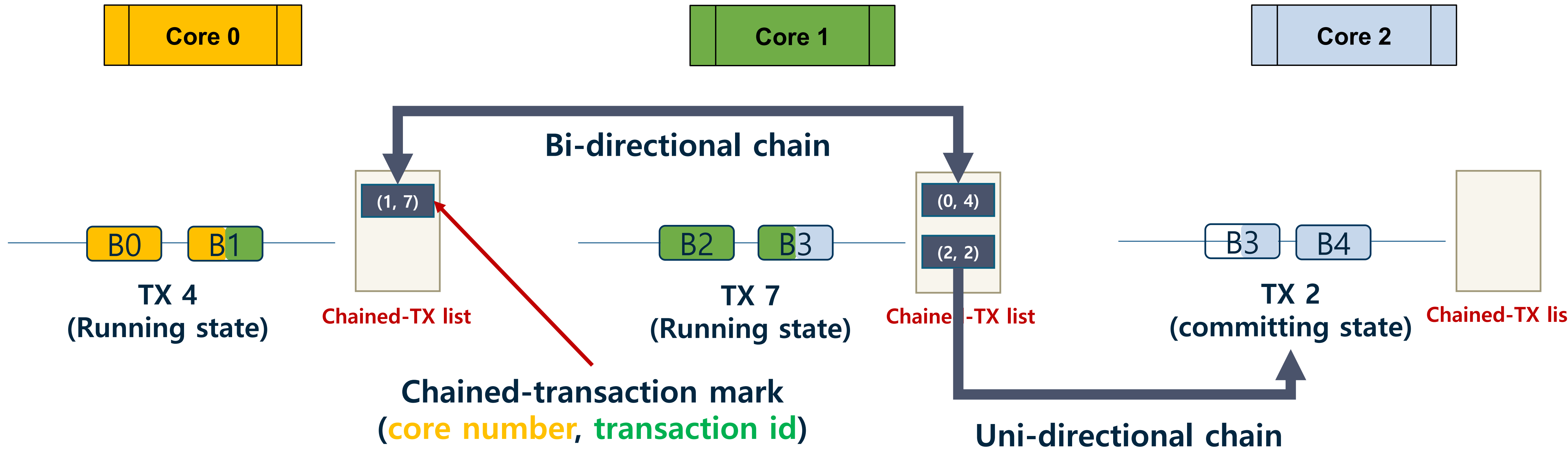
Order-preserving Transaction Chaining

- **We can checkpoint a modification of a buffer only after its preceding modifications become persistent**
- **We need to record write orders to a shared block**
- **The transaction chain graph**
  - Imposing order-constrains over transactions
  - Putting off the enforcement of write-order constraints to the checkpoint and recovery time

Construction of Transaction Chain Graphs

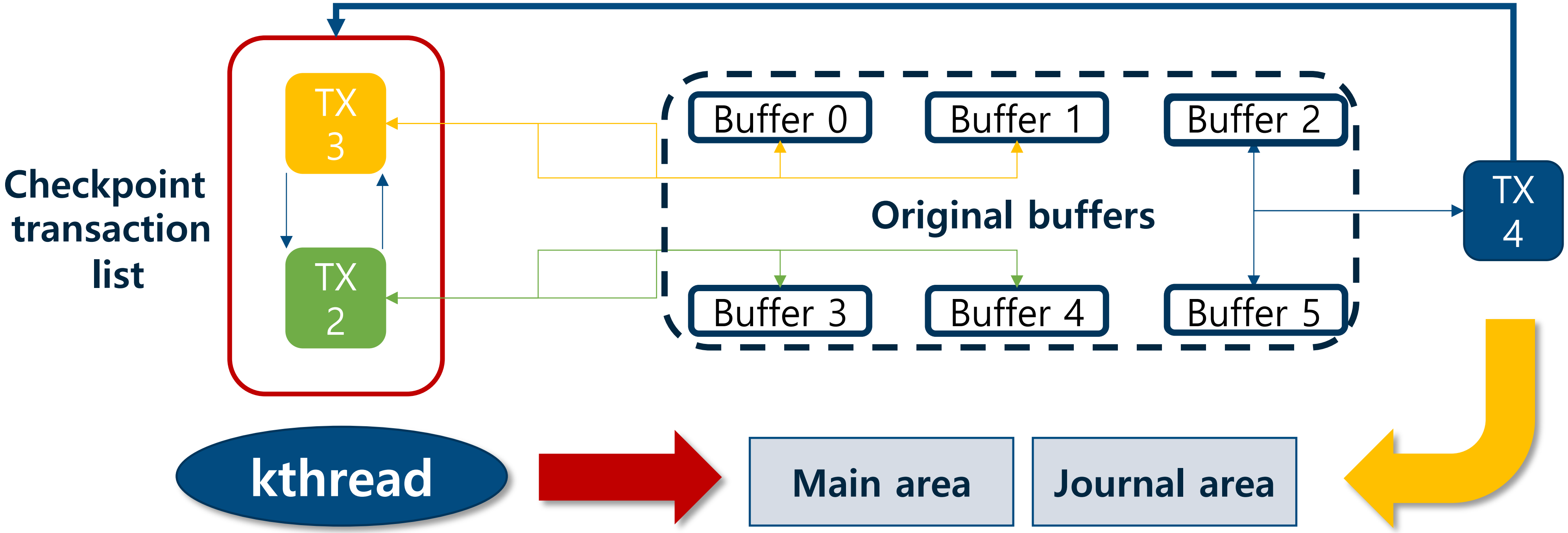
- Chained-transaction list
- Chained-transaction mark

Allowing each core to **commit** transactions to its journal **independently** to other journals



Conventional Checkpoint Process

- **When the commit is finished**
    - The transaction is inserted at the checkpoint transaction list
    - The buffers are marked as dirty

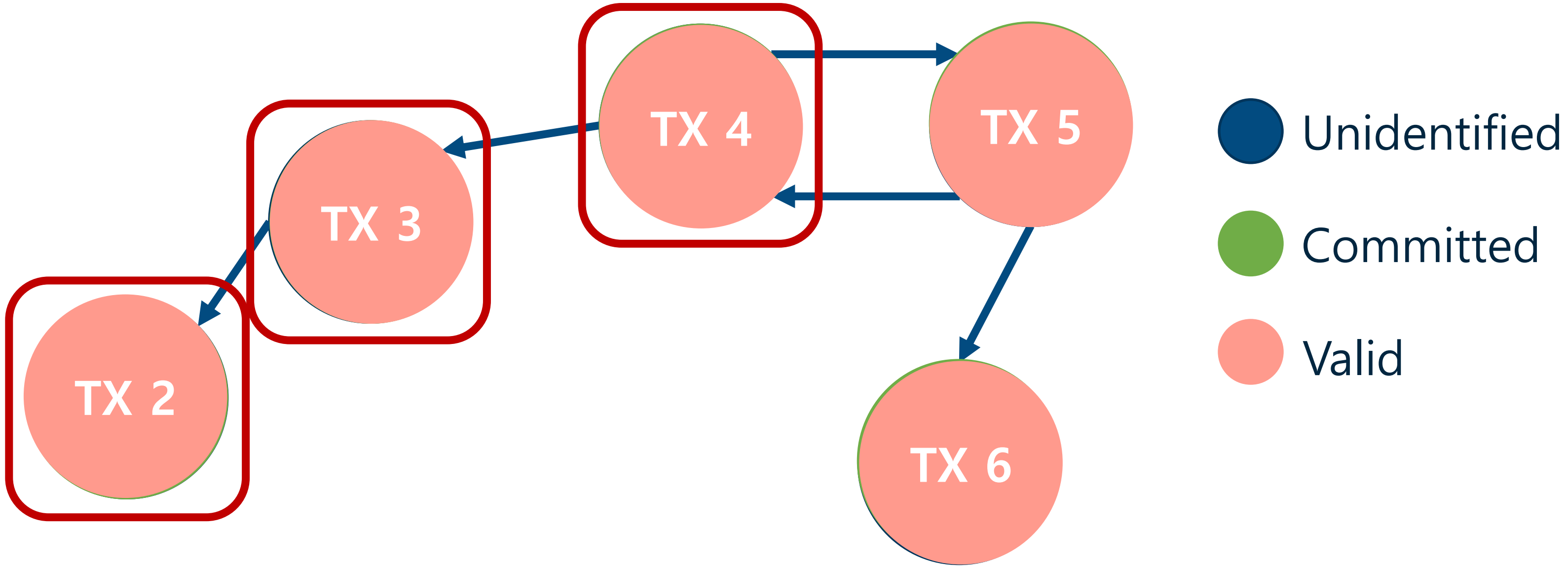


Journal Coherence Checkpoint

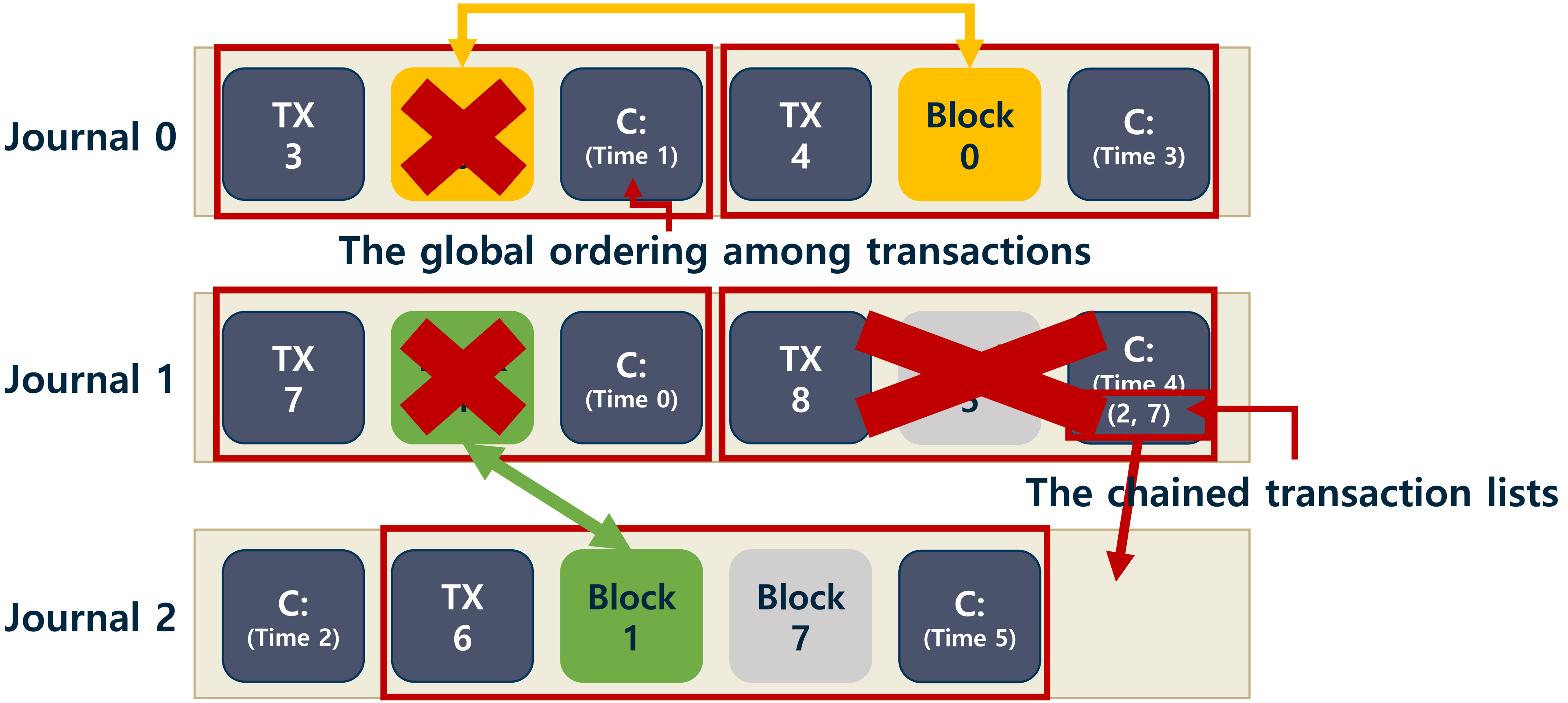
- **When the commit is finished**
  - The transaction will be skipped over setting dirty flags of transaction's buffers
  - As stated earlier, not all committed transactions become objects of checkpointing in Z-Journal
- **A transaction is *valid***
  - A transaction without a preceding transaction is *valid*
  - When all of its preceding transactions are *valid*
- **Z-Journal checkpoints only the valid transactions**
  - If a transaction turns out to be valid, its buffers will be marked as dirty

Transaction Chain Graph Traversing

- **To check the validity of a committed transaction, traversing the transaction chain graph is required**
- **Examples**
  - Check TX 2
  - Check TX 3
  - Check TX 4



Recovery



- Traversing the graphs to find valid transactions
- Comparing the timestamps of both transactions to determine the latest buffer image to restore

Evaluation

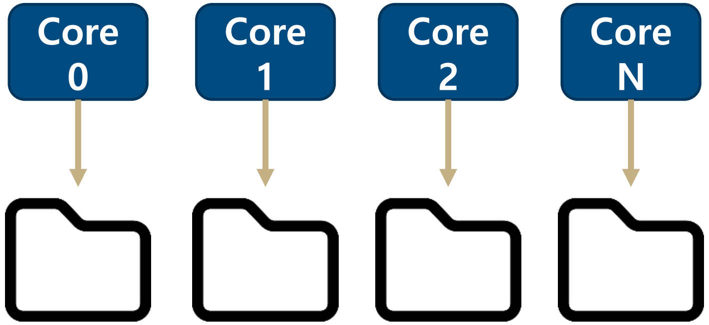
- **Environment**

	Specification
Processor	Intel Xeon Gold 6138 x 4 sockets
Memory	DDR4 2666 MHz 32GB x 16
Sotrage	Samsung SZ985 NVMe SSD 800GB
OS	Linux kernel 4.14.78 (Journal=data mode journaling)

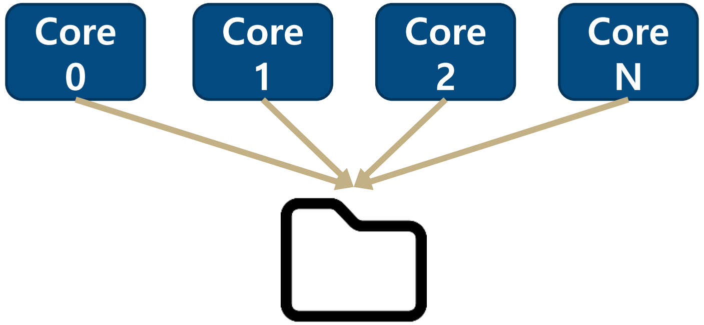
- **Implementation**
  - Z-Journal is modified based on JBD2
  - The ext4 file system has minimal modifications to recognize multiple journals
- **Workloads**
  - FxMark[1], Filebench, SysBench

[1] Changwoo Min, Sanidhya Kashyap, Steffen Maass, and Taesoo Kim. *Understanding manycore scalability of file systems*. In 2016 USENIX Annual Technical Conference (ATC), 2016.

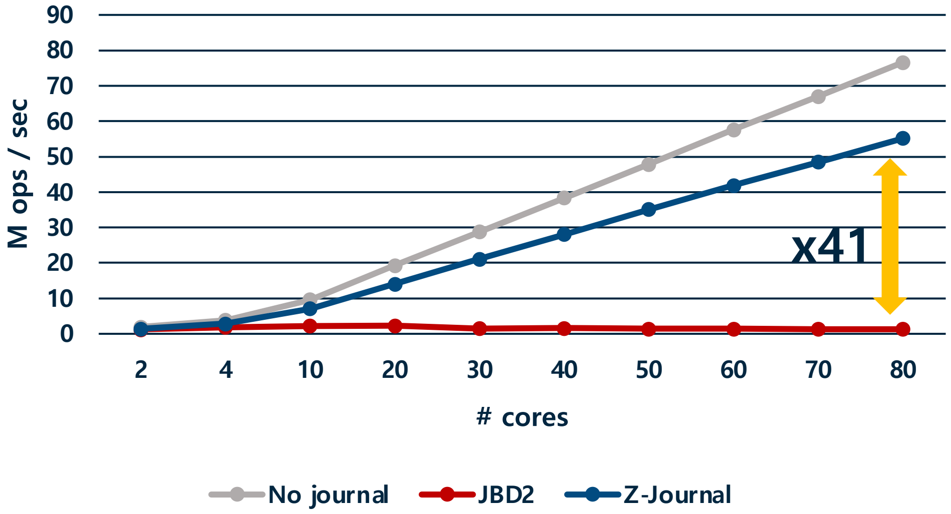
Under Various Sharing Conditions


Core 0
Core 1
Core 2
Core N

Low sharing

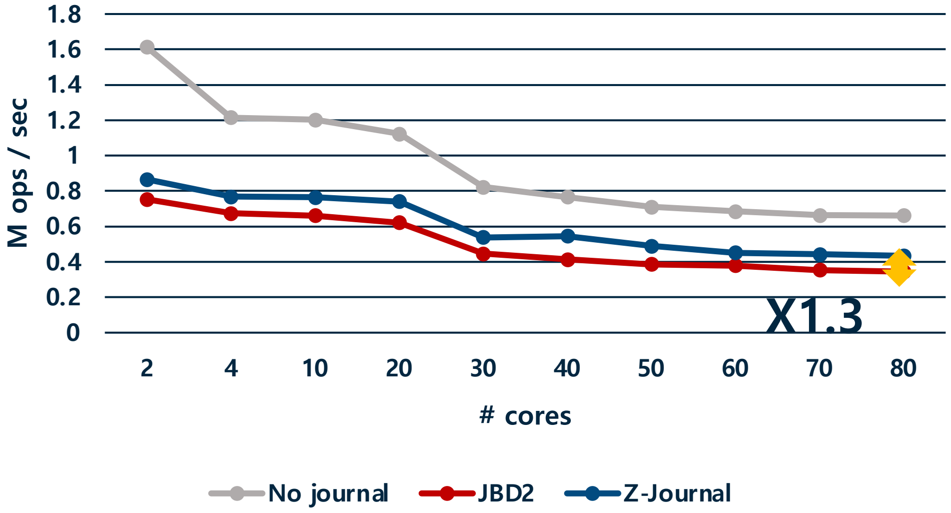

Core 0
Core 1
Core 2
Core N

Medium sharing



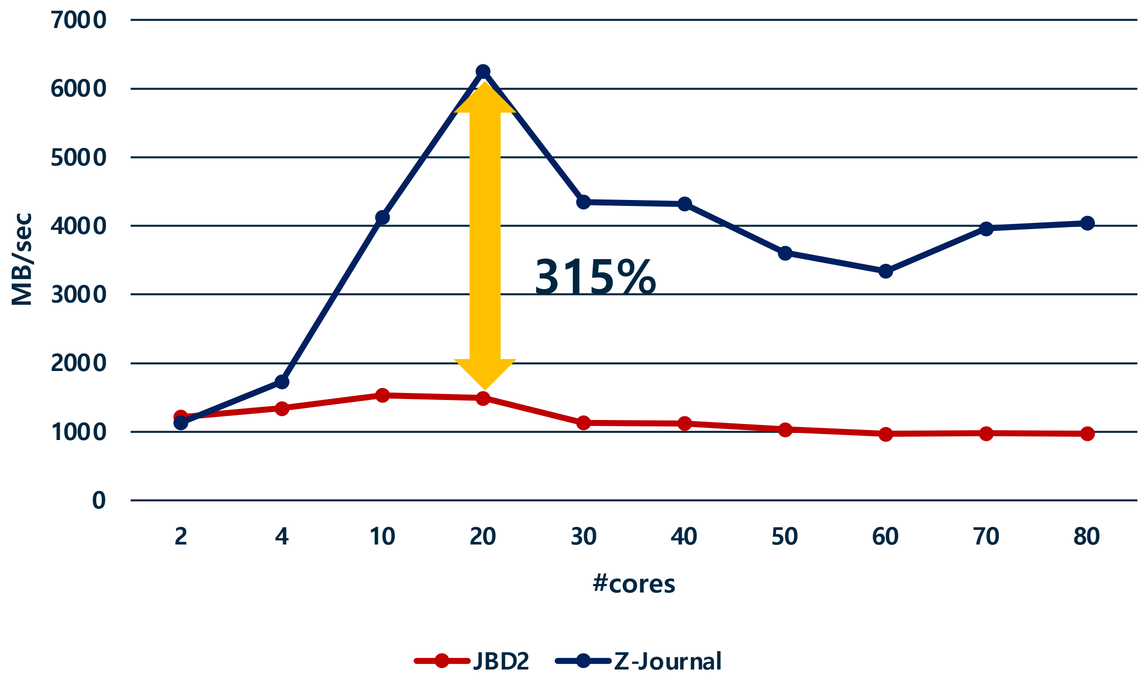
x41

Overwrite

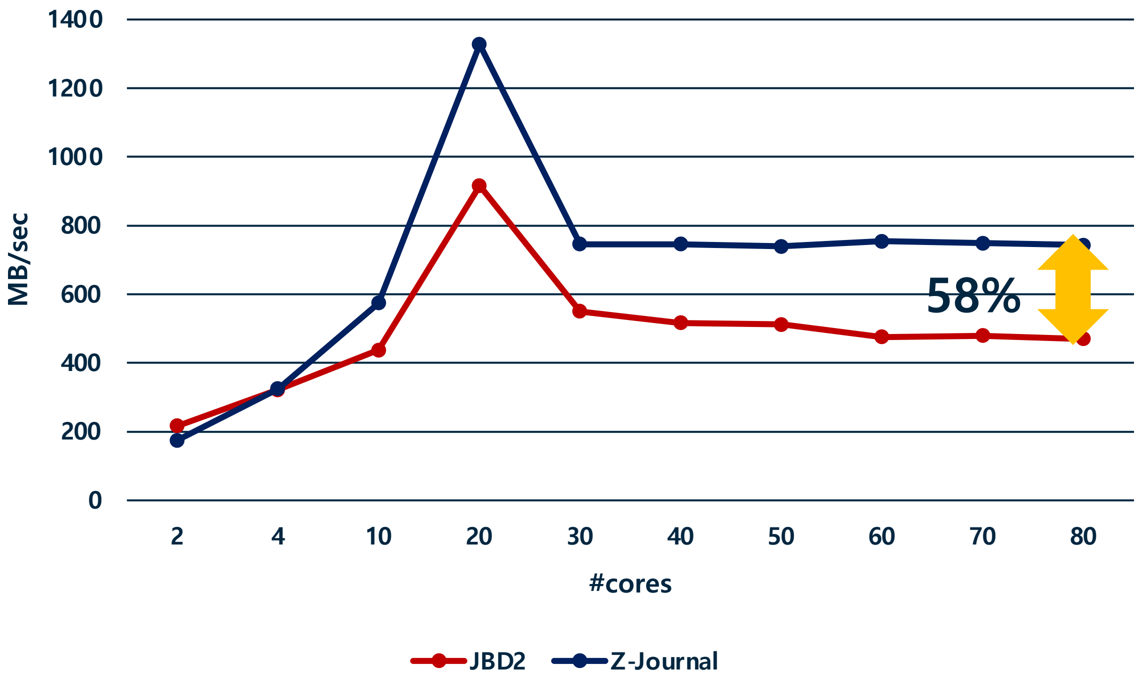


X1.3

Overall File System Performance



Fileserver



Varmail

Z-Journal Conclusion

- **Per-core journal**
  - The thread running on a core can write to the journal dedicated to the core independently to the other threads
- **Journal coherence mechanism**
  - This enables scalable per-core journaling while keeping crash consistency
- **Performance evaluation**
  - Our evaluation showed that, through its per-core journaling design, Z-Journal is faster and more scalable than the current JBD2

Q&A

Thank you

ks7sj@skku.edu