When Address Remapping Techniques Meet Consistency Guarantee Mechanisms

Dong Hyun Kang, Gihwan Oh, Dongki Kim†, In Hwan Doh†,
Changwoo Min‡, Sang-Won Lee, and Young Ik Eom
Sungkyunkwan University †Samsung Electronics ‡Virginia Tech

What Is Consistency, And Why Is It Important?

- What if you lose your precious data?

- How we can build a crash consistency system?
  - Turn on one of the consistency mechanisms
    - Journaling, copy-on-write, and logging

[Source: https://n2ws.com/blog/ebs-snapshot/transaction-logs-and-journaling]

Where To Handle Consistency Mechanism?

- **File system-level**
  - Journaling: ext3, ext4, and XFS
  - Copy-on-write: Btrfs and ZFS
  - Logging: F2FS
- **Application-level**
  - Database: MySQL, Oracle, and SQLite
  - Editor: Vim

Motivation

- Consistency mechanisms need extra writes to keep the file system to a consistent state
  - Redundant writes in journaling
  - Copy writes in copy-on-write
  - Additional writes in log-structured

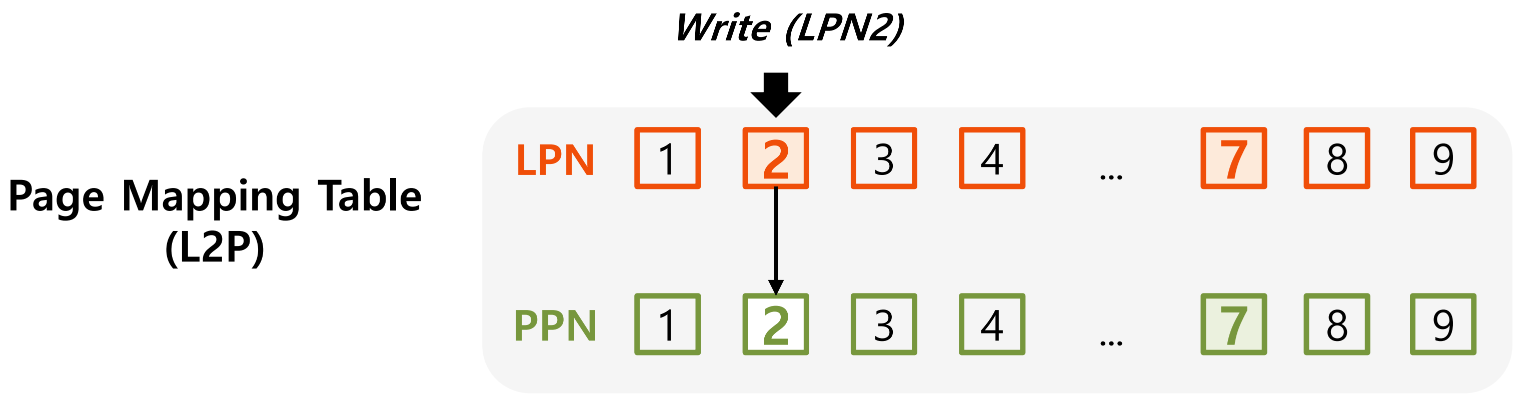
- Research question
  - **Can we guarantee crash consistency by writing the data only once?**

Outline

- Background
- Related work
- Case studies
- Implementation & Challenges
- Conclusion

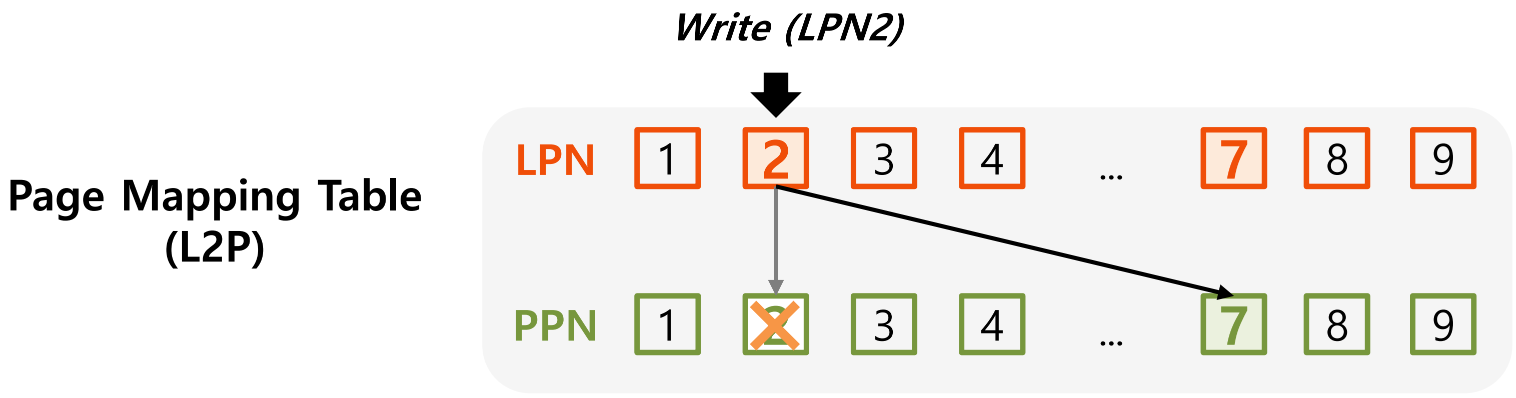
Flash Storage Device

- Flash storage device uses a special software inside the storage
  - **FTL** (flash translation layer): it emulates overwrite behavior by remapping its own mapping table



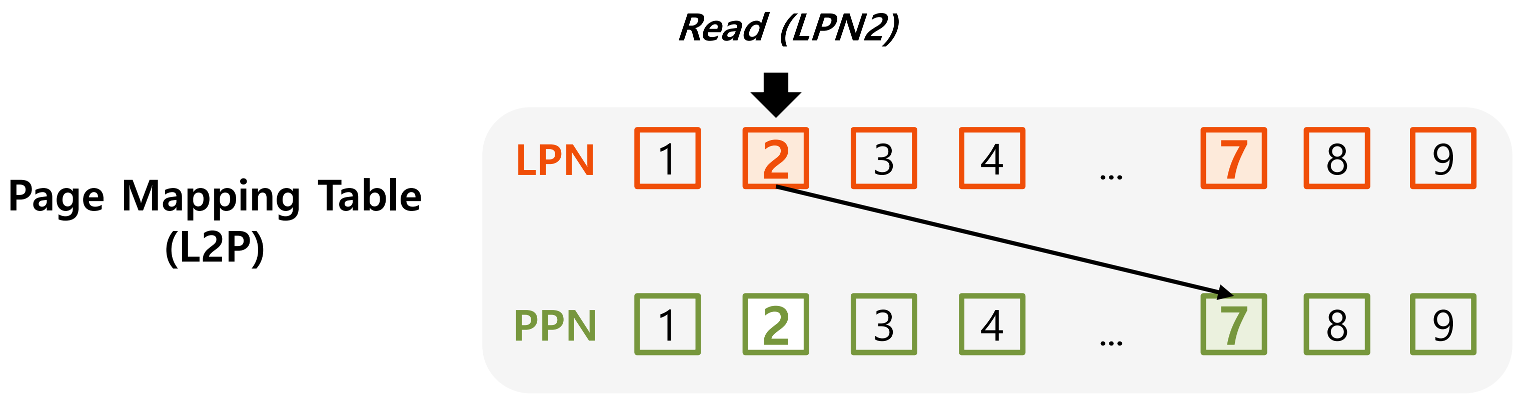
Flash Storage Device

- Flash storage device uses a special software inside the storage
  - **FTL** (flash translation layer): it emulates overwrite behavior by remapping its own mapping table



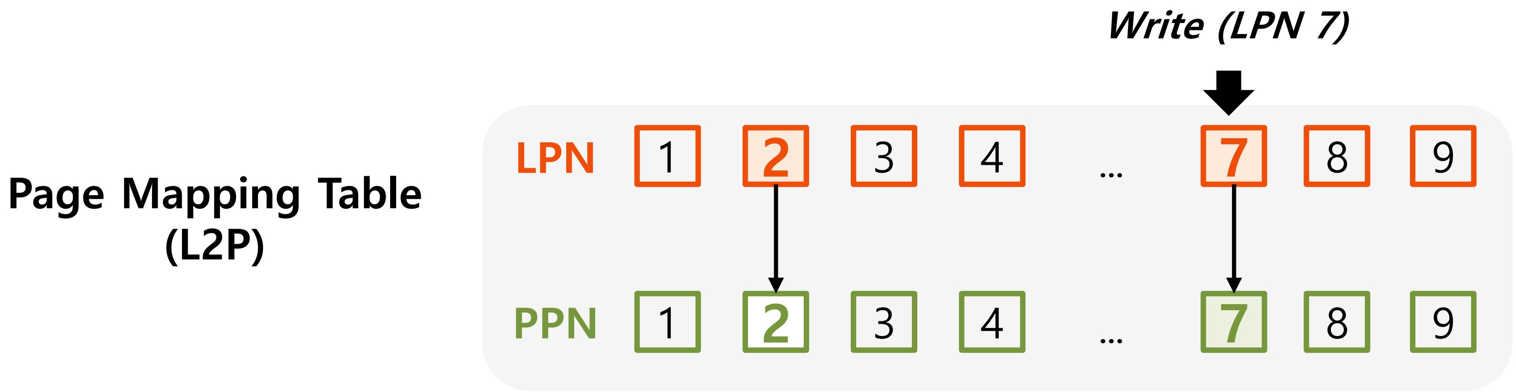
Flash Storage Device

- Flash storage device uses a special software inside the storage
  - **FTL** (flash translation layer): it emulates overwrite behavior by remapping its own mapping table



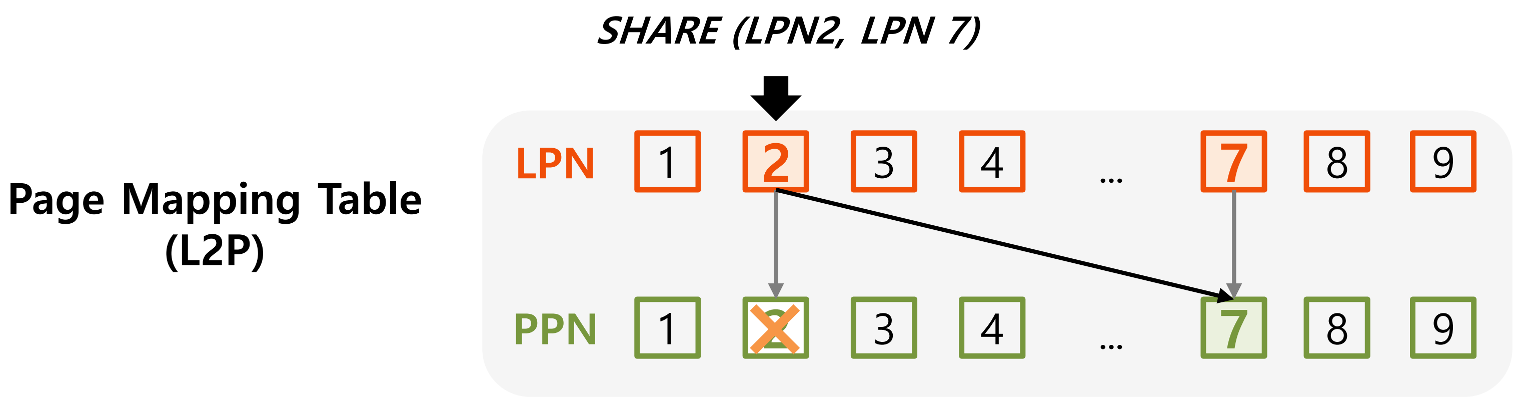
SHARE Interface

- **SHARE** interface [SIGMOD'16] allows host to explicitly ask FTL to change its internal address mapping table
  - Target PPN is shared via address remapping



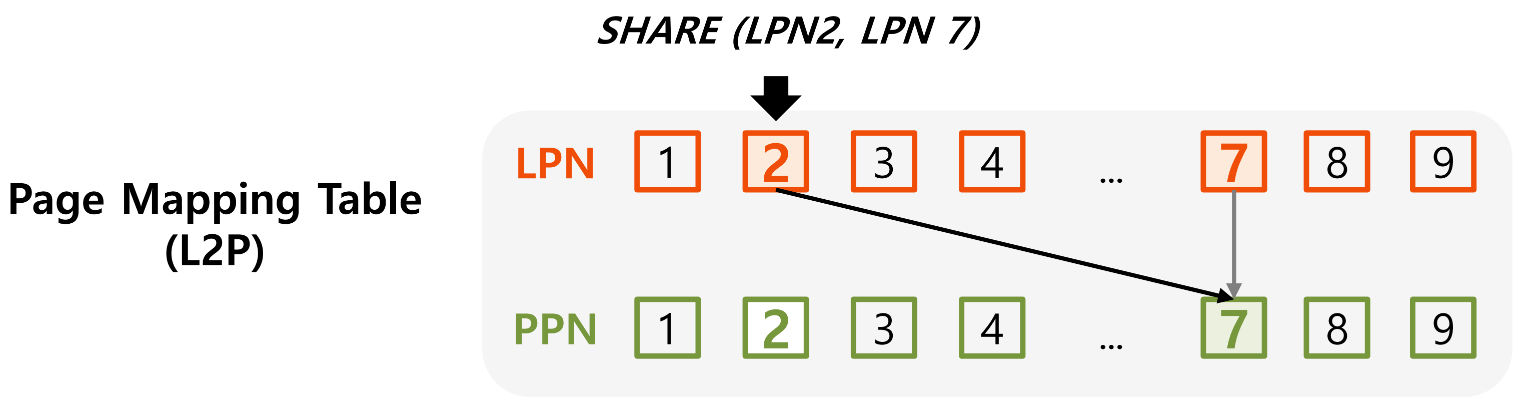
SHARE Interface

- **SHARE** interface [SIGMOD'16] allows host to explicitly ask FTL to change its internal address mapping table
  - Target PPN is shared via address remapping



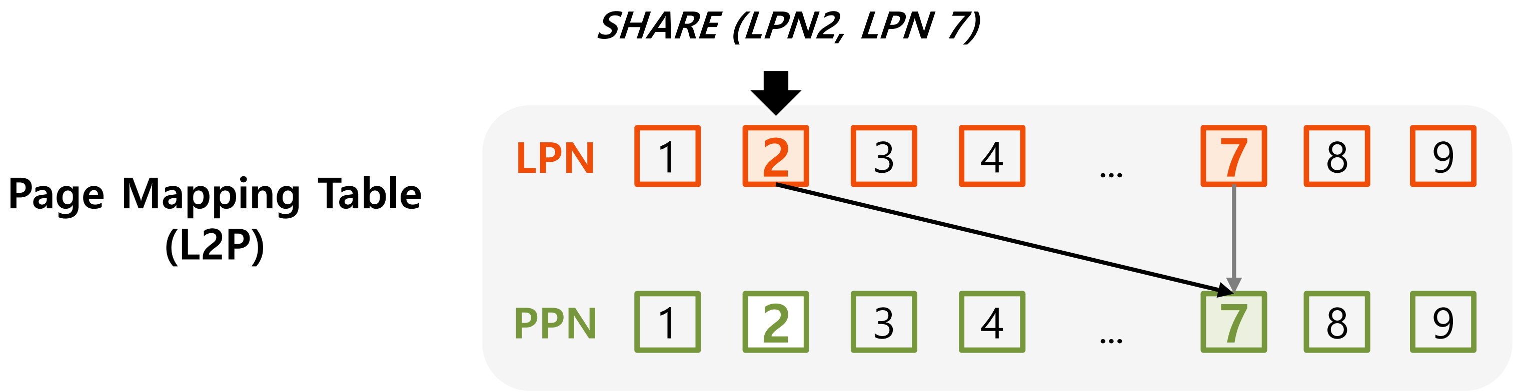
SHARE Interface

- **SHARE** interface [SIGMOD'16] allows host to explicitly ask FTL to change its internal address mapping table
  - Target PPN is shared via address remapping



SHARE Interface

- **SHARE** interface [SIGMOD'16] allows host to explicitly ask FTL to change its internal address mapping table
  - Target PPN is shared via address remapping



- **SHARE** atomically supports multi-address remapping

Outline

- Background
- Related work
- Case studies
- Implementation & Challenges
- Conclusion

Remapping Approaches in Various Cases

- Which layer
  - JFTL [ACM TOS'09] -> FTL layer
  - ANViL [USENIX FAST'15] -> Virtual storage layer
  - SHARE [ACM SIGMOD'16] -> FTL layer
  - Janus [USENIX ATC'17] -> FTL with File system layer
  - SHRD [USENIX FAST'17] -> FTL with Block layer
  - Ext4-lazy [USENIX FAST'17] -> File system layer

Remapping Approaches in Various Cases

- What purposes
  - JFTL [ACM TOS'09] -> File system-level consistency
  - ANViL [USENIX FAST'15] -> File system-level consistency
  - SHARE [ACM SIGMOD'16] -> Application-level consistency
  - Janus [USENIX ATC'17] -> Defragmentation
  - SHRD [USENIX FAST'17] -> Sequential writes
  - Ext4-lazy [USENIX FAST'17] -> Sequential writes

Remapping Approaches in Various Cases

- What purposes
  - JFTL [ACM TOS’09] -> File system-level consistency
  - ANViL [USENIX FAST’15] -> File system-level consistency
  - SHARE [ACM SIGMOD’16] -> Application-level consistency
  - Janus [USENIX ATC’17] -> Defragmentation
  - SHRD [USENIX FAST’17] -> Sequential writes
  - Ext4-lazy [USENIX FAST’17] -> Sequential writes

Remapping Approaches in Various Cases

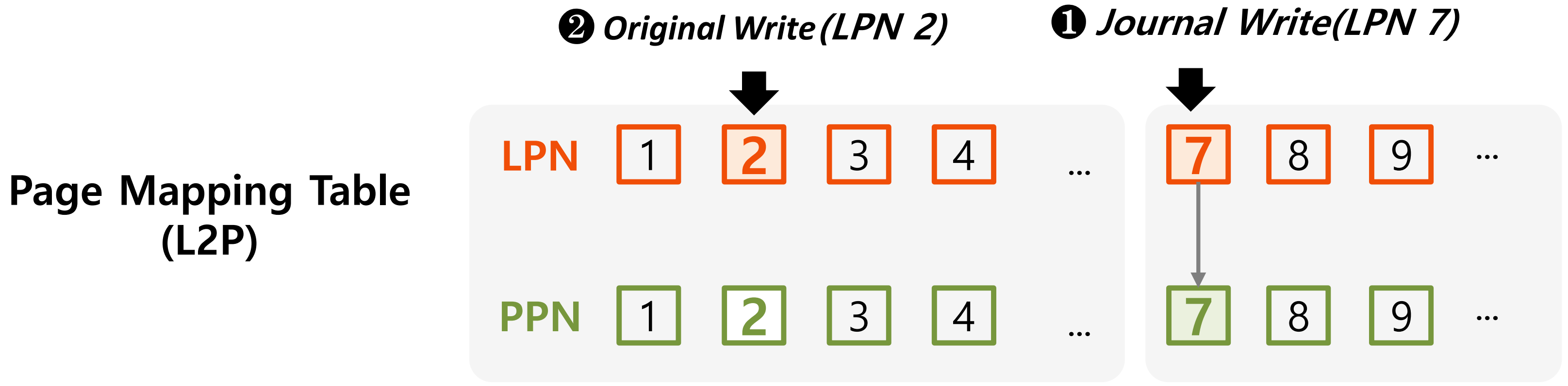
- What purposes
  - JFTL [ACM TOS'09] -> File system-level consistency
  - ANViL [USENIX FAST'15] -> File system-level consistency
  - SHARE [ACM SIGMOD'16] -> Application-level consistency
  - Janus [USENIX ATC'17] -> Defragmentation
  - SHRD [USENIX FAST'17] -> Sequential writes
  - Ext4-lazy [USENIX FAST'17] -> Sequential writes

Outline

- Background
- Related work
- Case studies
- Implementation & Challenges
- Conclusion

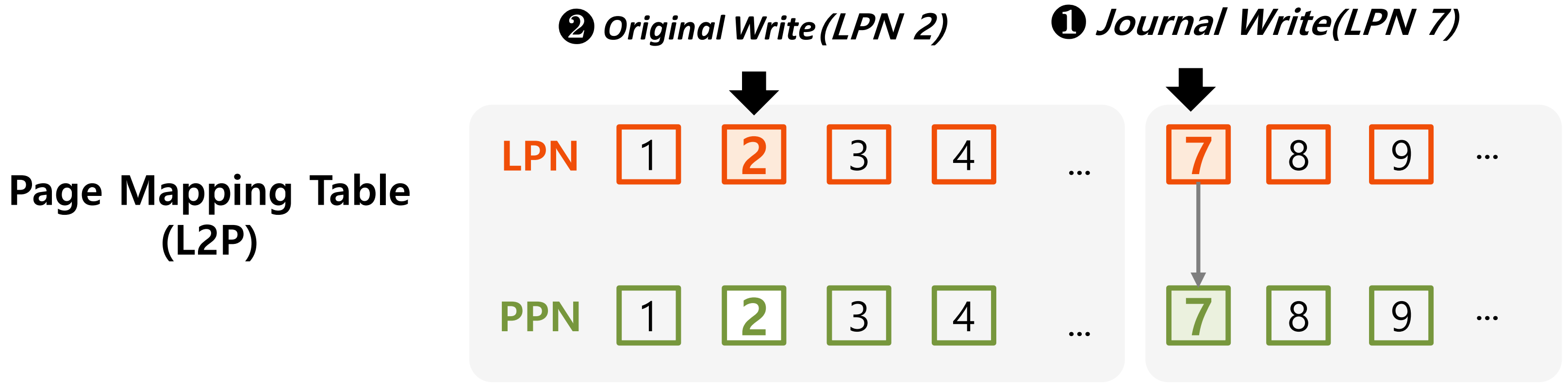
Case Study 1: Ext4

- Traditional Ext4 file system writes same data twice to guarantee crash consistency



Case Study 1: Ext4

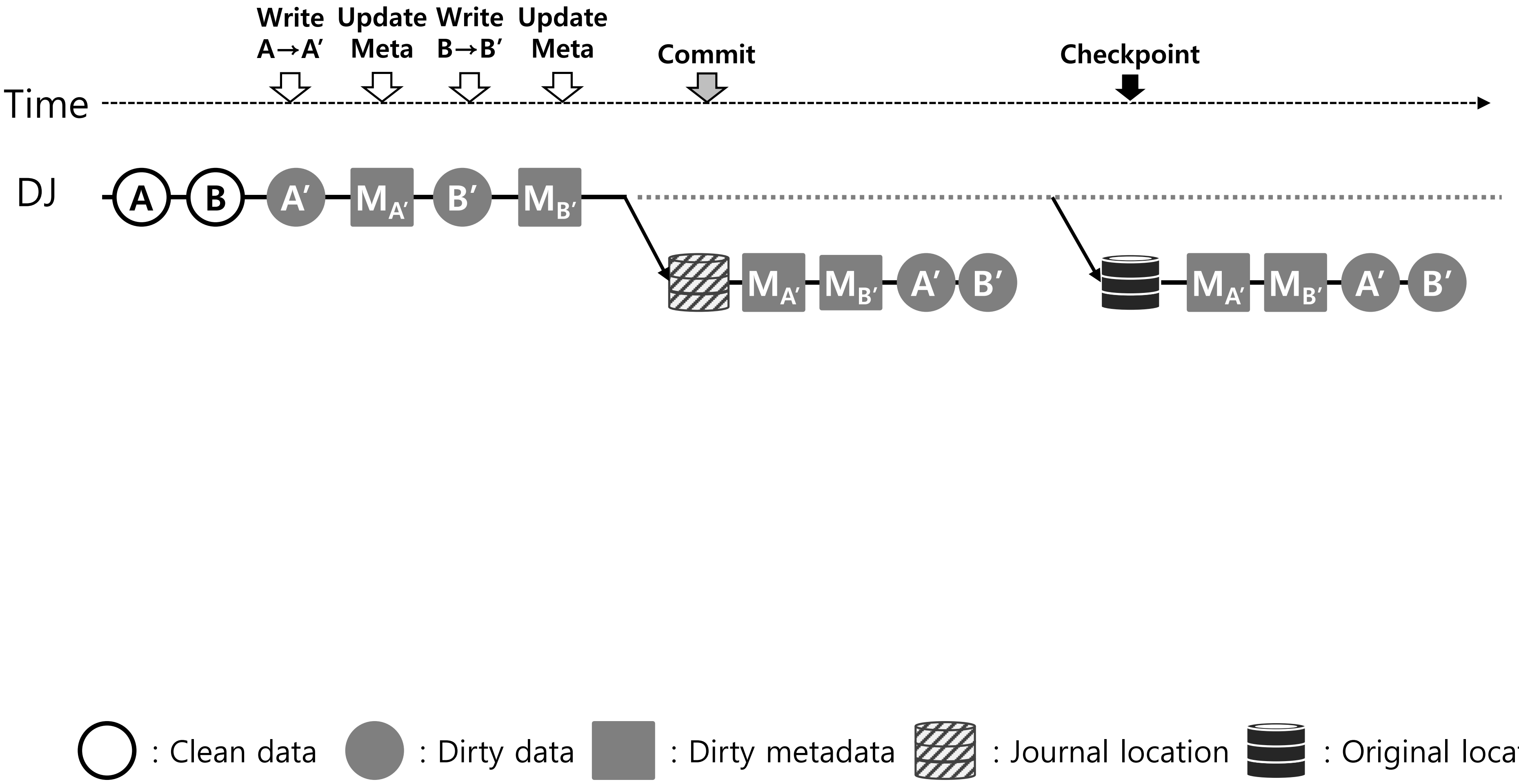
- Traditional Ext4 file system writes same data twice to guarantee crash consistency



- **SHARE-aware Ext4 can remove the second write by delegating it to SHARE**
  - SHARE-aware ordered journaling (SOJ) mode
  - SHARE-aware data journaling (SDJ) mode

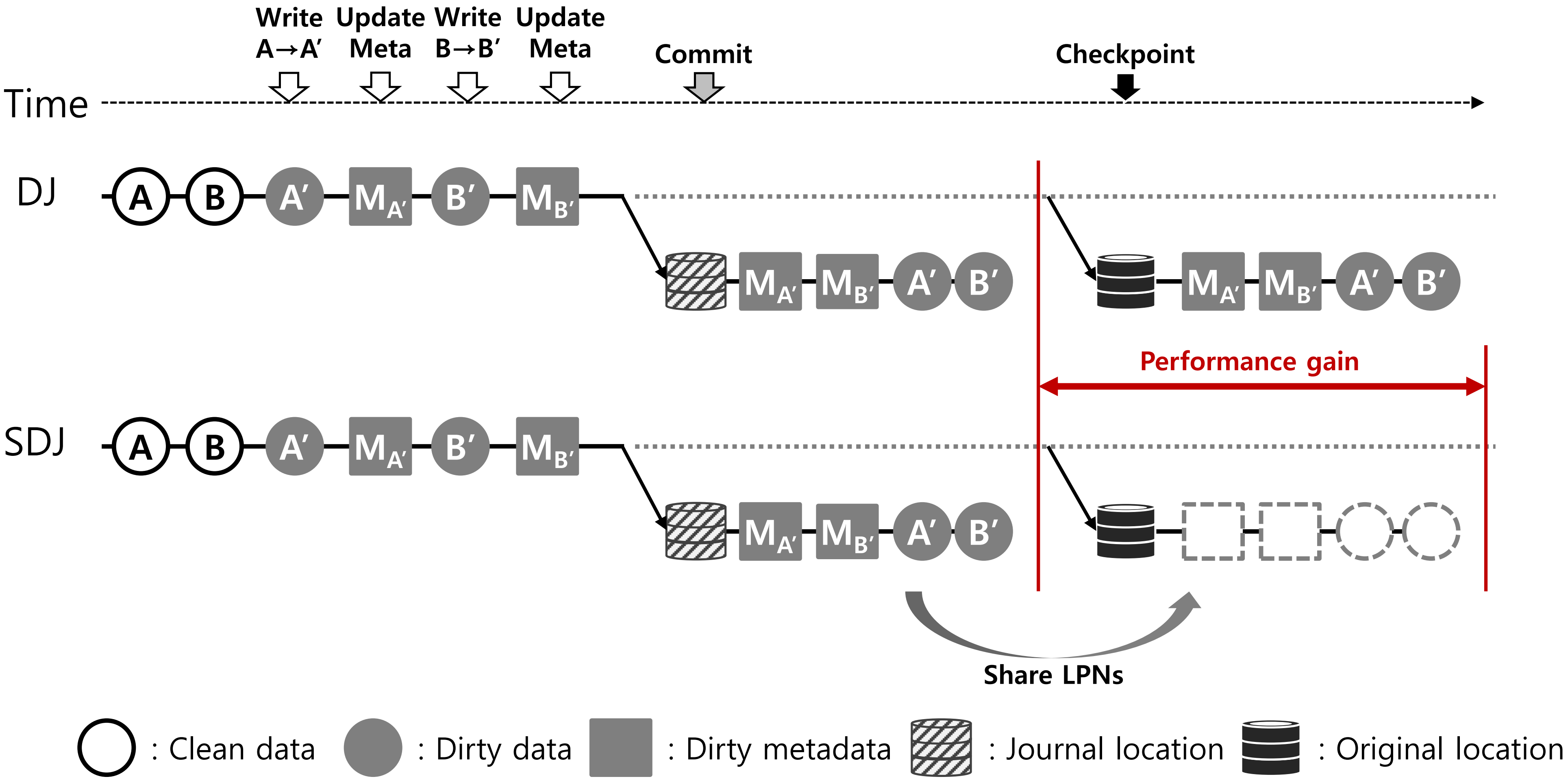
Case Study 1: Ext4

- For example (Data journaling mode)



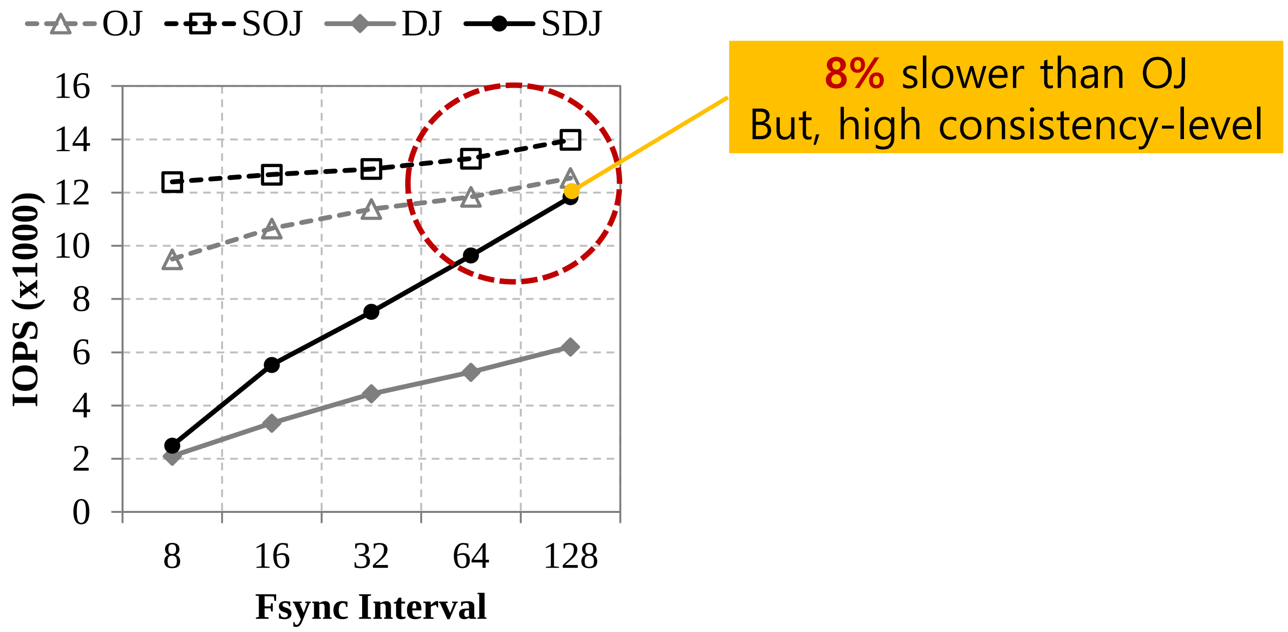
Case Study 1: Ext4

- For example (Data journaling mode)



Case Study 1: Ext4

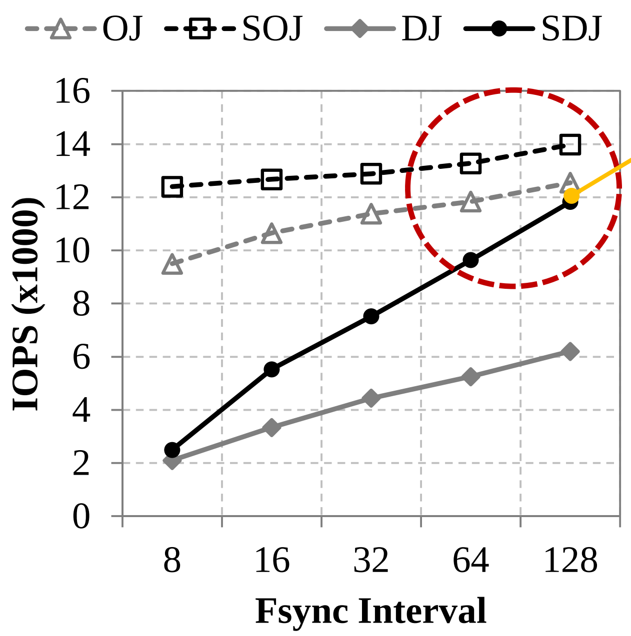
- Performance (FIO and Varmail)
  - SOJ shows better performance than traditional OJ
  - SDJ has significantly performance gain at large ***fsync*** interval



[FIO performance]

Case Study 1: Ext4

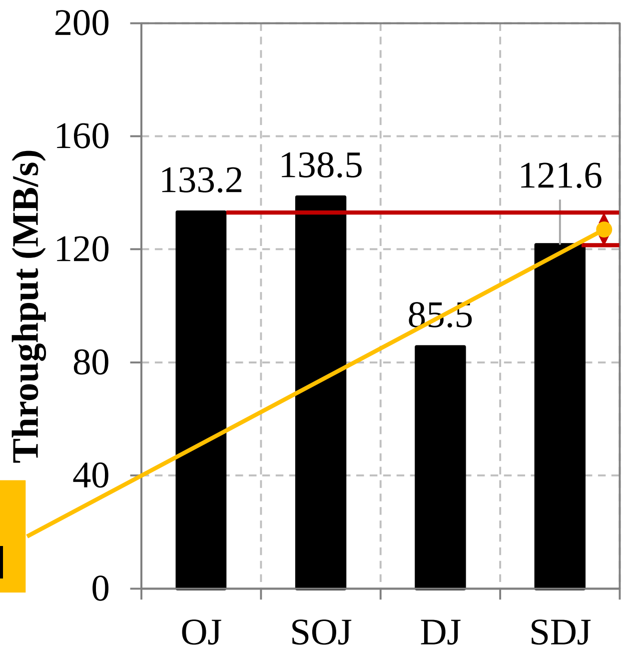
- Performance (FIO and Varmail)
  - SOJ shows better performance than traditional OJ
  - SDJ has significantly performance gain at large ***fsync*** interval



[FIO performance]

8% slower than OJ
But, high consistency-level

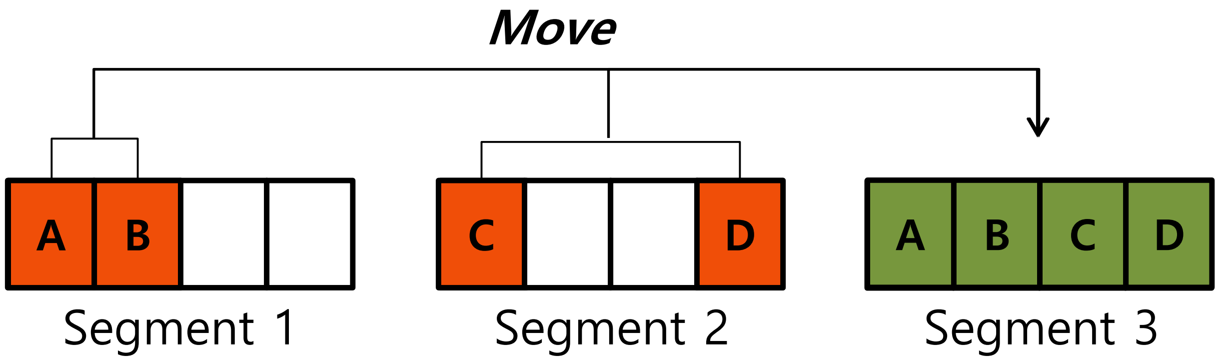
9% slower than OJ
But, high consistency-level



[Varmail performance]

Case Study 2: LFS

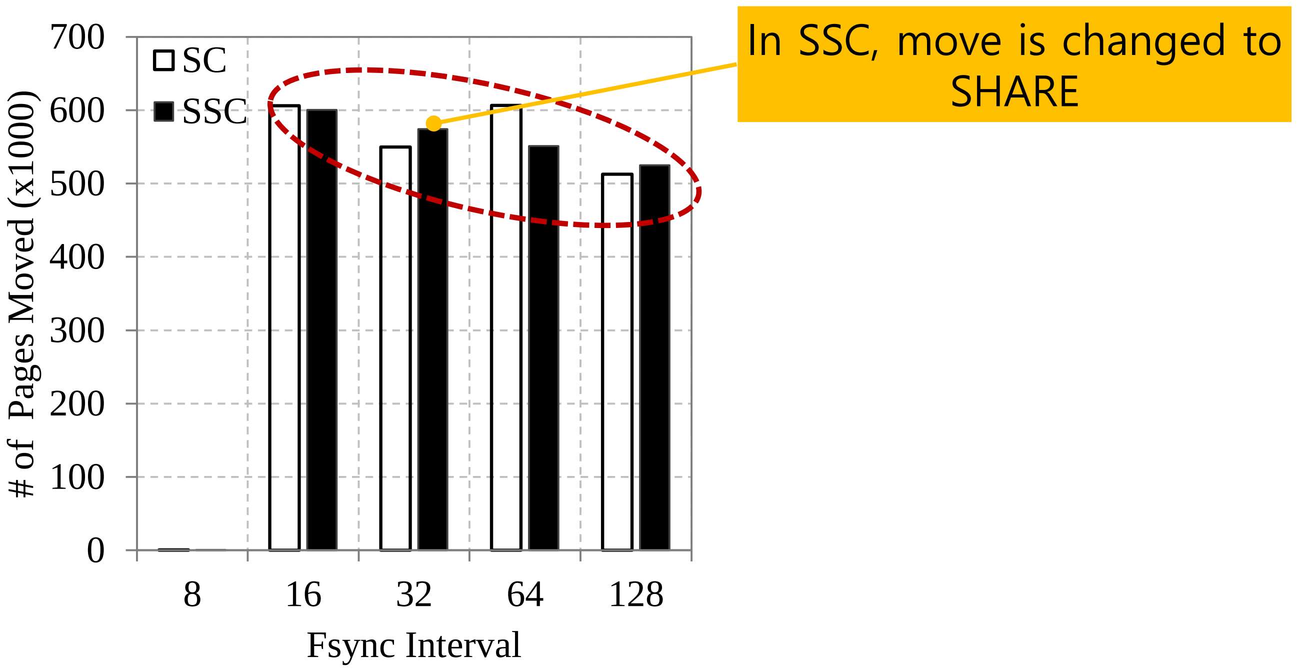
- Existing LFS basically requires the segment cleaning operation to reclaim free space



- **SHARE-aware LFS can remove the move operation by delegating it to SHARE**
  - SHARE-aware segment cleaning (SSC)

Case Study 2: LFS

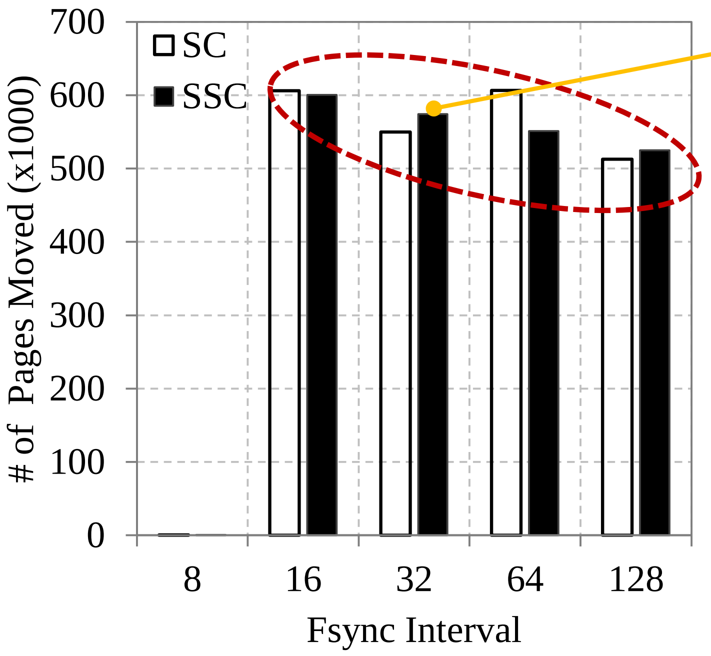
- Performance (FIO)
  - The number of total moved pages is similar to that of SC
  - But, SSC shows better performance than default SC



[The number of total moved pages]

Case Study 2: LFS

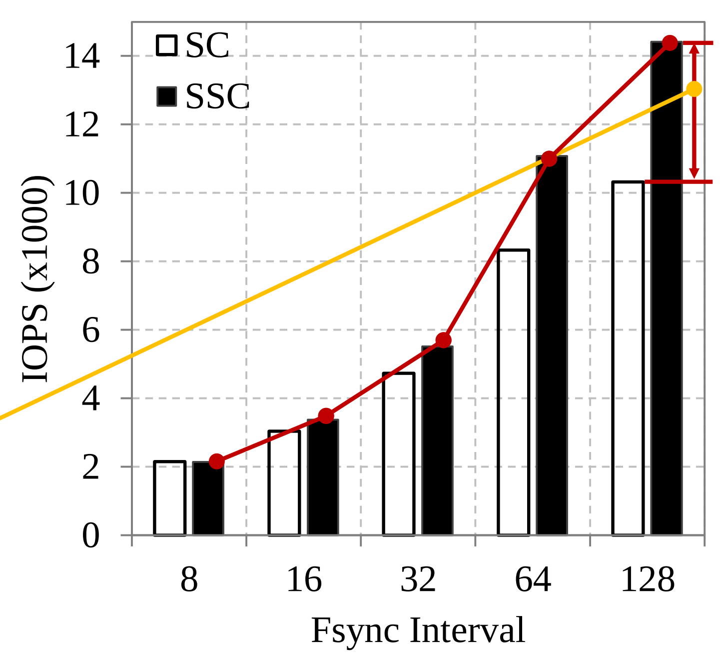
- Performance (FIO)
  - The number of total moved pages is similar to that of SC
  - But, SSC shows better performance than default SC



[The number of total moved pages]

In SSC, move is changed to SHARE

39% faster than SC

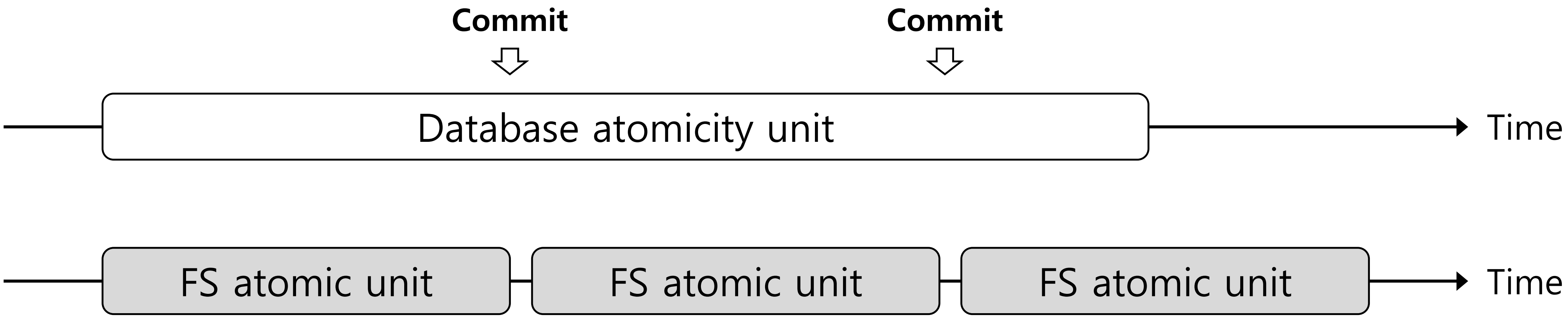


[Performance]

Case Study 3: Application

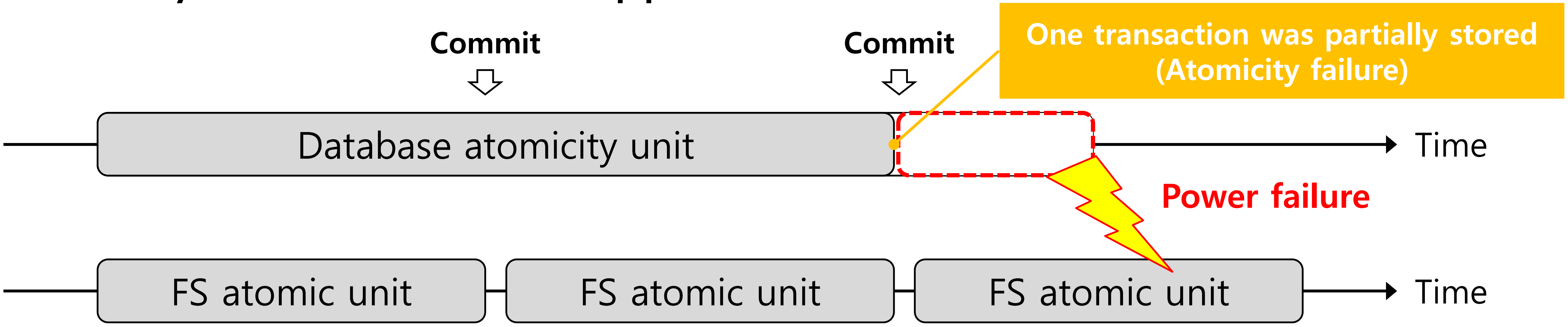
- Some applications (e.g., databases and key-value stores) have their own consistency mechanisms even with Ext4 DJ mode
  - Double write buffer in MySQL

- In DJ mode, the transaction of file system may break the atomicity of the database application



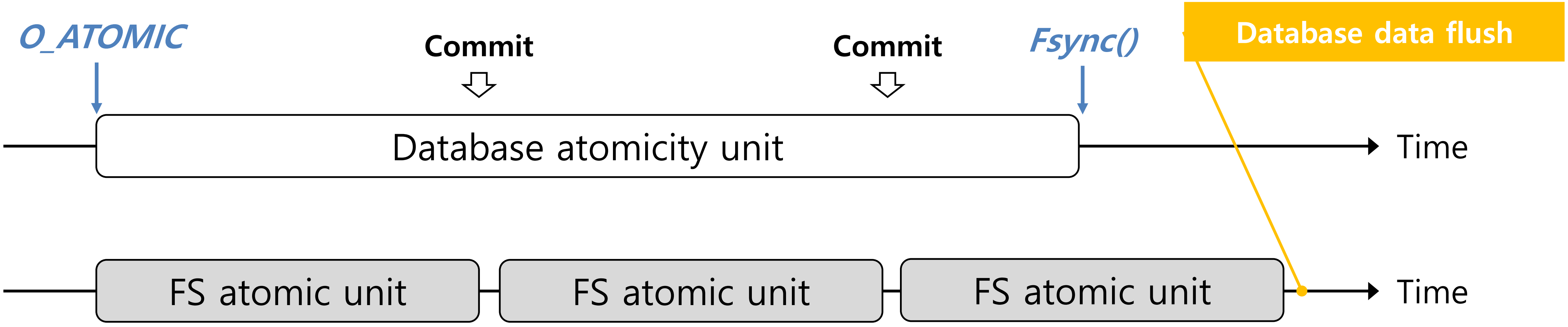
Case Study 3: Application

- Some applications (e.g., databases and key-value stores) have their own consistency mechanisms even with Ext4 DJ mode
  - Double write buffer in MySQL
- In DJ mode, the transaction of file system may break the atomicity of the database application



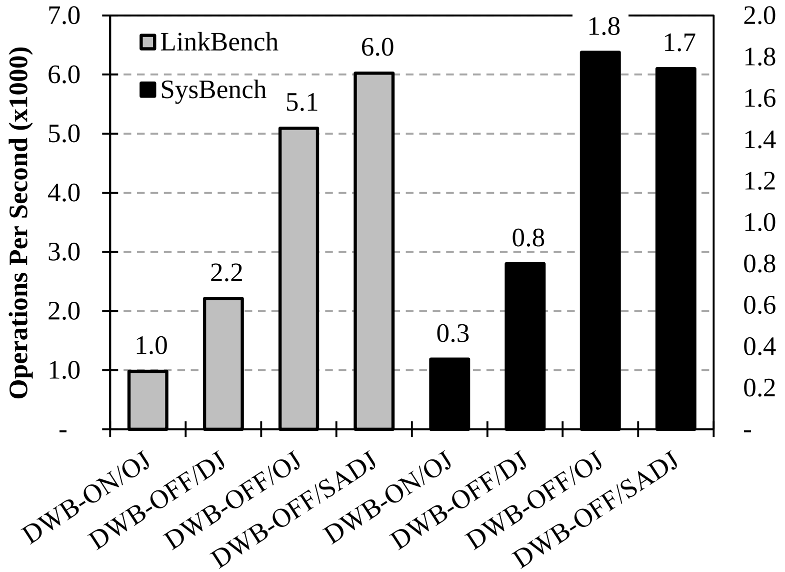
Case Study 3: Application

- **The ACID semantics of database transactions can be successfully guaranteed via SHARE**
  - SHARE-aware application-level data journaling (SADJ) mode
    - It utilizes the failure-atomic update APIs [EUROSYS'13]
      - O_ATOMIC flag, failure-atomic msync(), and syncv() interface



Case Study 3: Application

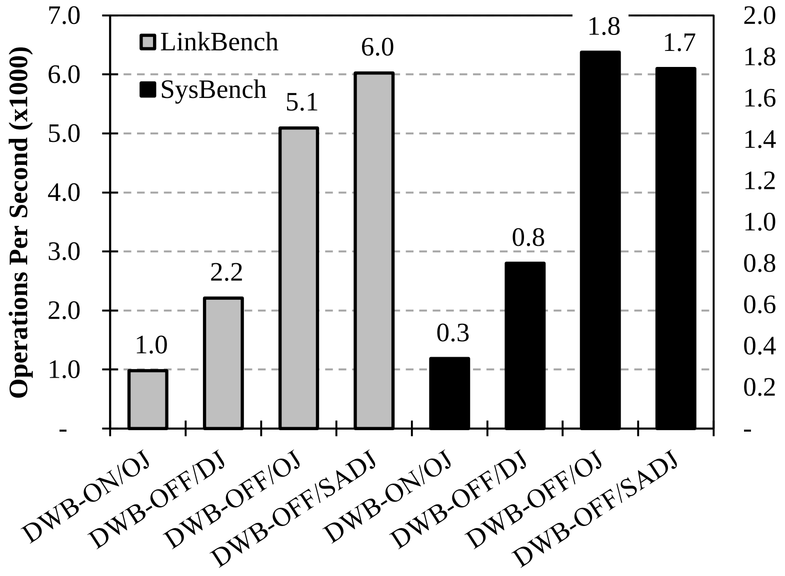
- Performance (MySQL/InnoDB)
  - DWB-OFF/SADJ outperforms the DWB-ON/OJ by 6.16 times and the DWB-OFF/DJ by 2.73 times
  - DWB-OFF/SADJ invokes 16.4x less disk cache FLUSH operations



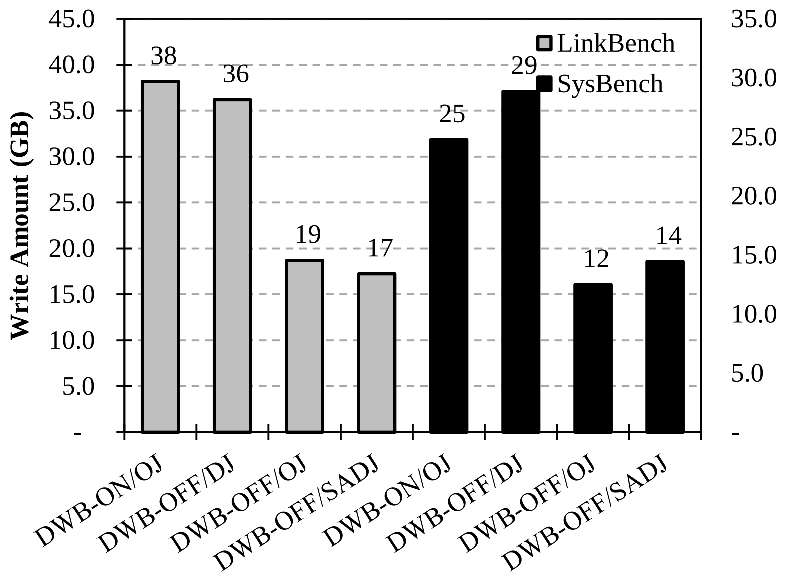
[Performance]

Case Study 3: Application

- Performance (MySQL/InnoDB)
  - DWB-OFF/SADJ outperforms the DWB-ON/OJ by 6.16 times and the DWB-OFF/DJ by 2.73 times
  - DWB-OFF/SADJ invokes 16.4x less disk cache FLUSH operations



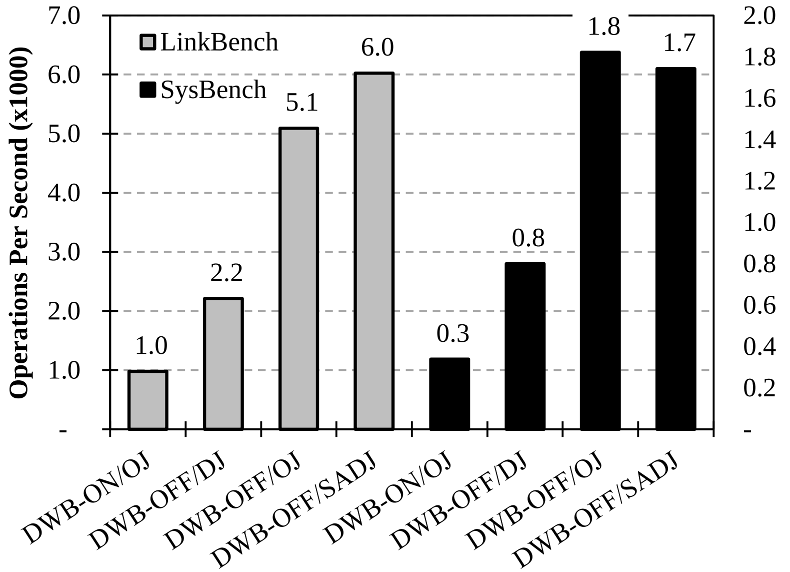
[Performance]



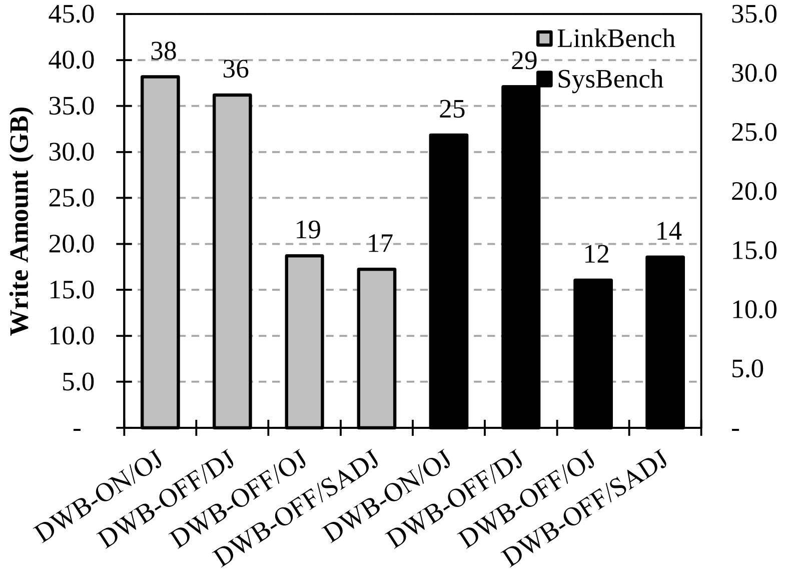
[Write amount]

Case Study 3: Application

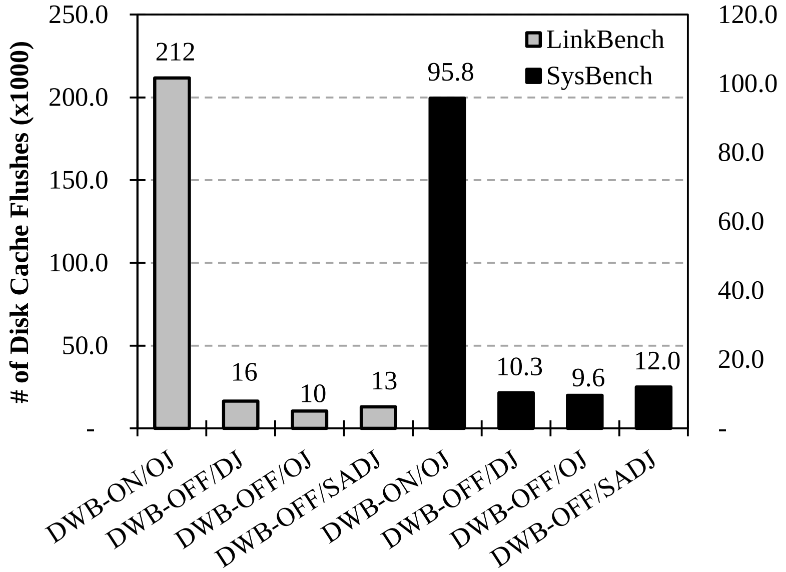
- Performance (MySQL/InnoDB)
  - DWB-OFF/SADJ outperforms the DWB-ON/OJ by 6.16 times and the DWB-OFF/DJ by 2.73 times
  - DWB-OFF/SADJ invokes 16.4x less disk cache FLUSH operations



[Performance]



[Write amount]



[# of Disk cash flush]

Outline

- Background
- Related work
- Case studies
- Implementation & Challenges
- Conclusion

Implementation & Challenges

- Implementation
  - Linux kernel 4.6.7
  - Quad-core processor (Intel i7-6700) and 8GB memory
  - SHARE interface
    - SHARE-enabled SSD by modifying an FTL firmware of a commercial high-end PCIeM.2 SSD
    - SHARE command has been added as a vendor unique command

- Challenges
  - the small-size journal area (i.e., 128 MB)

Outline

- Background
- Related work
- Case studies
- Implementation & Challenges
- Conclusion

Conclusion

- Tackled a problem in current consistency mechanisms
  - Double write overhead
  - Segment cleaning overhead

- Presented a comprehensive study with the address remapping technique

- Feature work
  - CoW-based B-tree file systems need to be explored

Thank you!

Questions?