

NVRAMOS 2015
Jeju, Korea, Oct. 23, 2015

WALDIO: ELIMINATING THE FILESYSTEM JOURNALING IN RESOLVING THE JOURNALING OF JOURNAL ANOMALY



Youjip Won

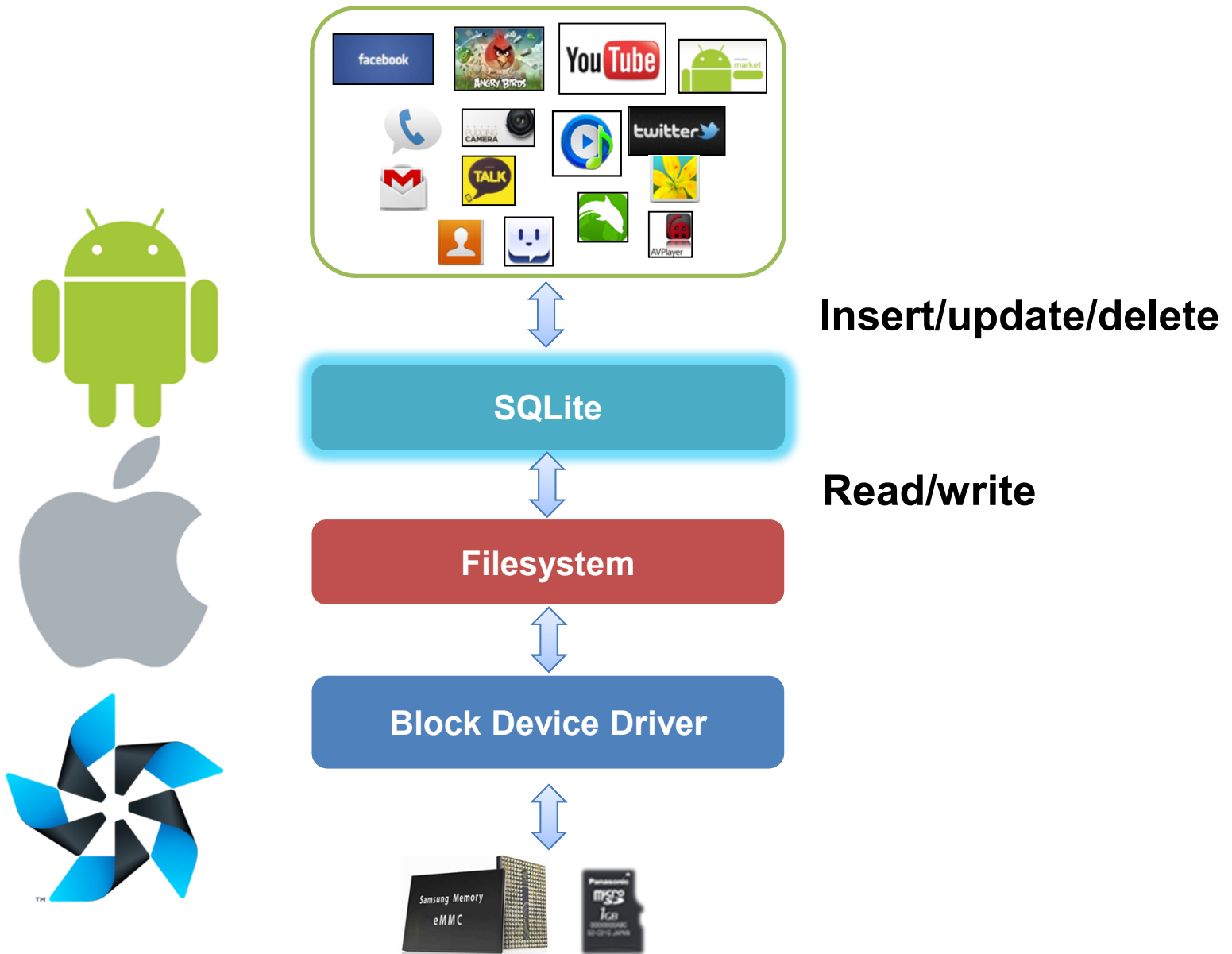
Hanyang University

OUTLINE

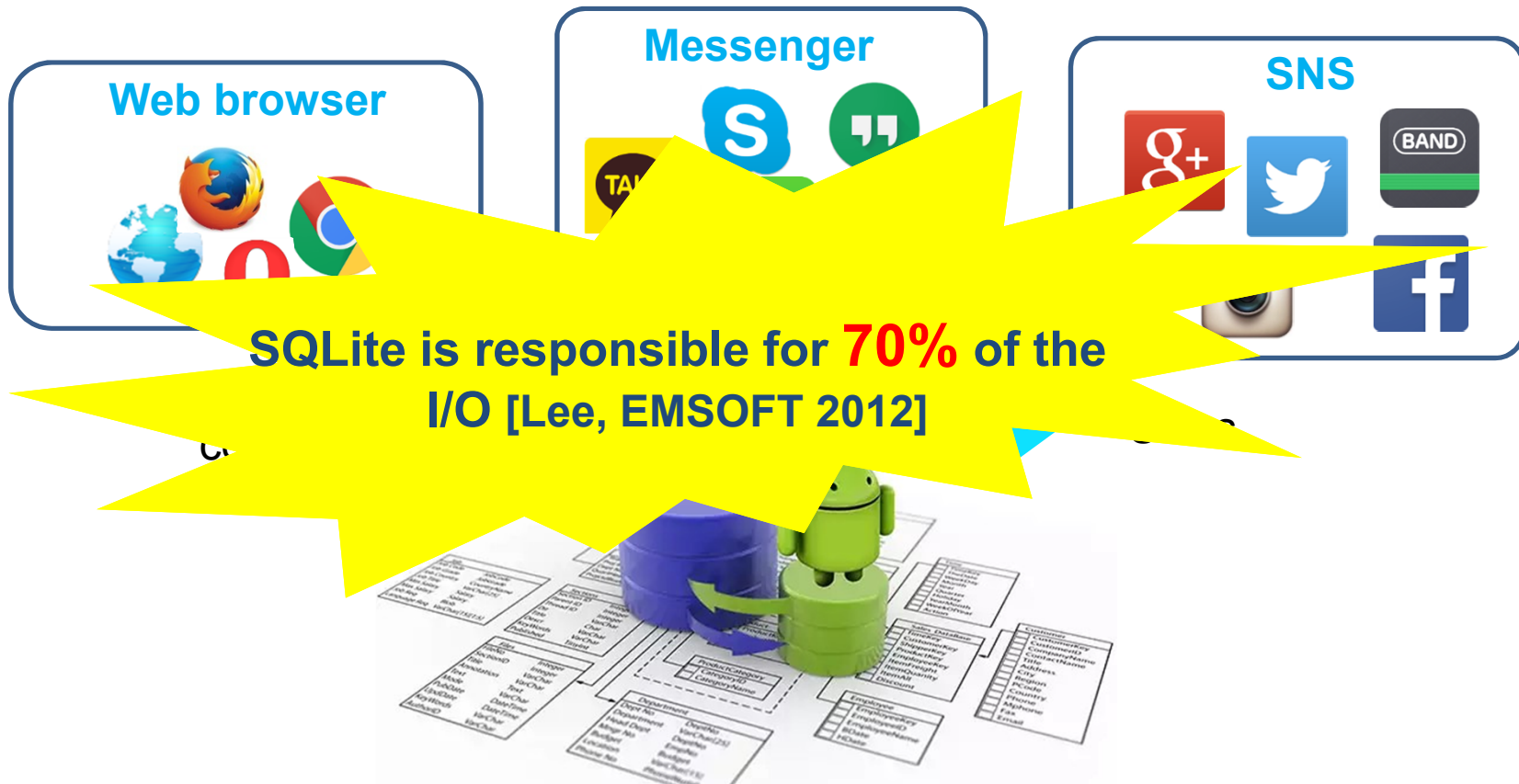
- **Motivation**
- **Background**
- **Eliminating the JOJ Anomaly**
- **Problem Assessment**
- **Proposal**
 - Preallocation in with Explicit Journaling
 - Header Embedding
 - Group Sync
- **Durability**
- **Experiment**
- **Summary**

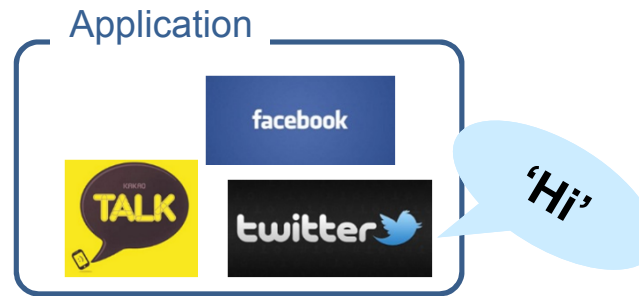
INTRO

000. **Motivation**



In Smartphone, SQLite maintains everything!





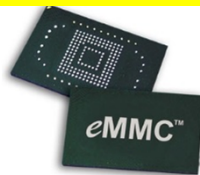
EXT4 filesystem journals SQLite journal file.

“Journaling of Journal Anomaly” [Jeong USENIX ATC 13]

f

1

Eliminate EXT4 Journaling from SQLite Journal IO!



No	comments
0	'Hello'
1	'What...'
2	'Hi'

Background

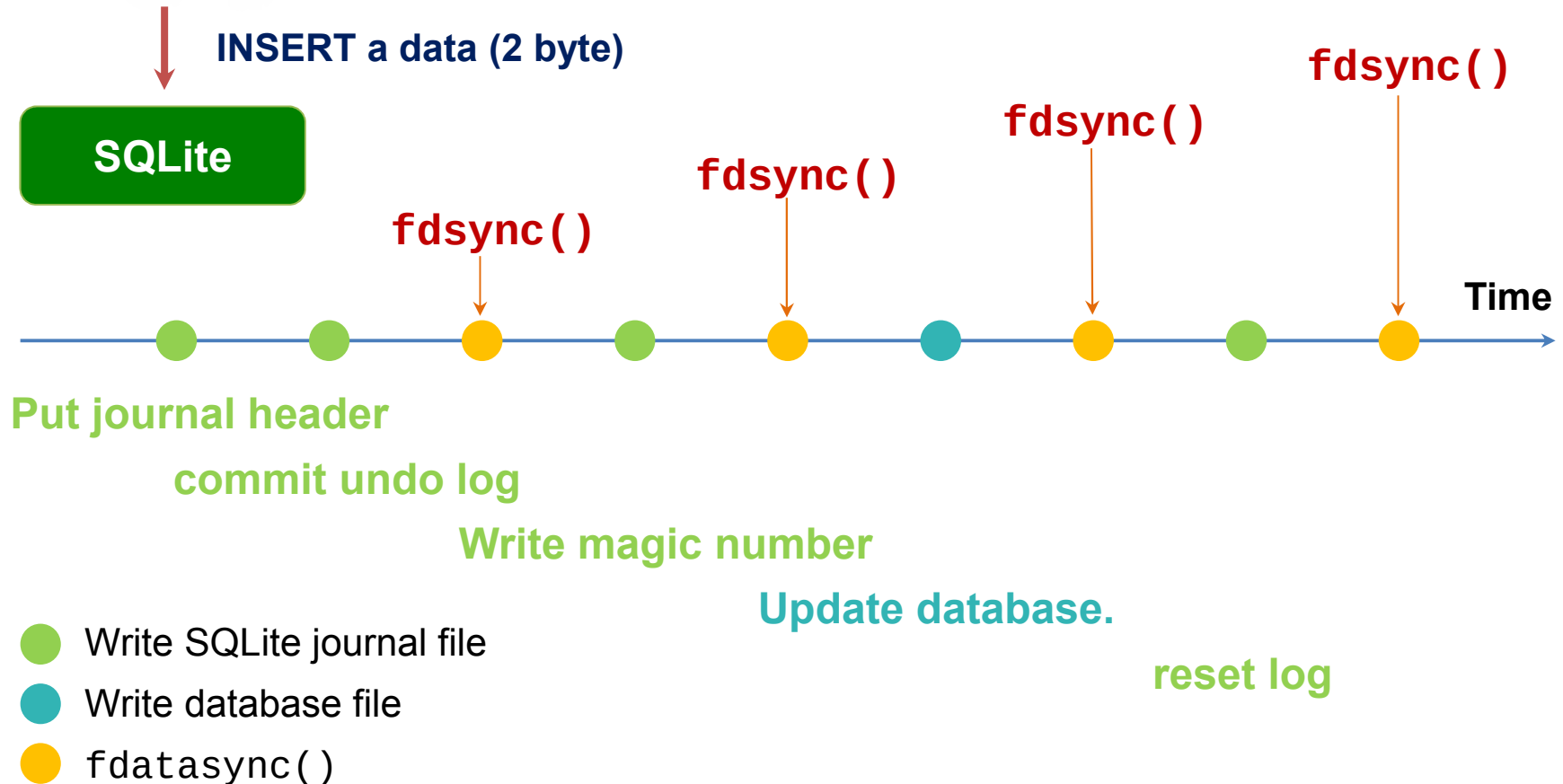
001. **Journaling of Journal Anomaly**

INSERT (PERSIST MODE)

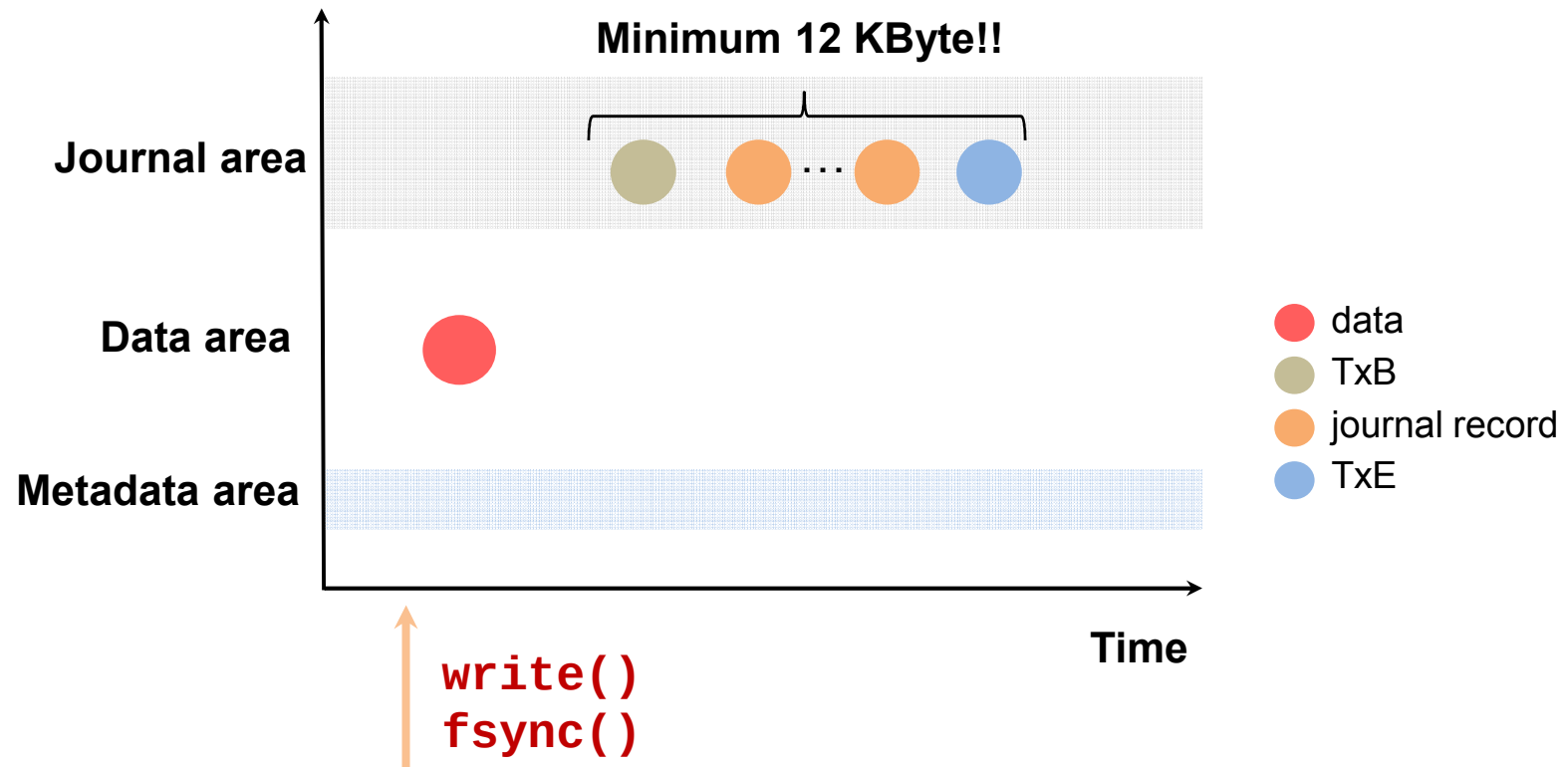


INSERT a data (2 byte)

SQLite



EXT4 JOURNALING (ORDERED MODE)

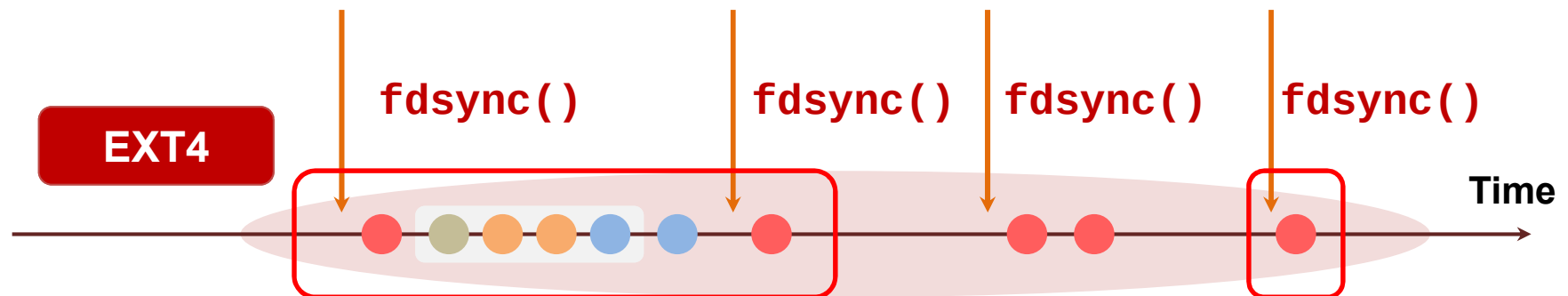


JOURNALING OF JOURNAL ANOMALY (PERSIST MODE)

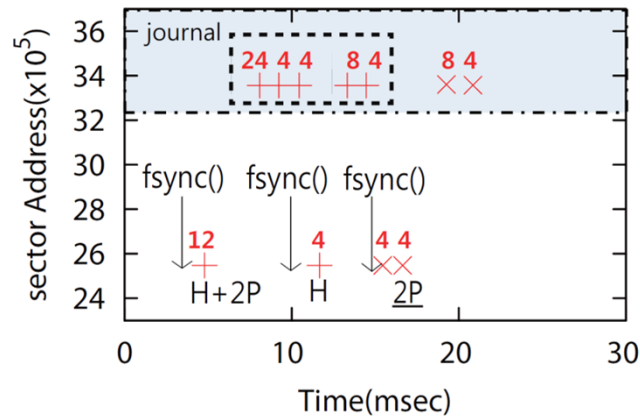


- Write journal file
- Write database file

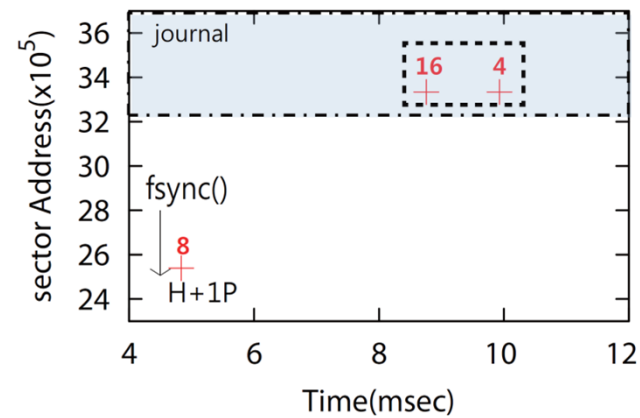
**Inserting 'Hi' (2 Byte)
→ 40 KByte to storage !!!**



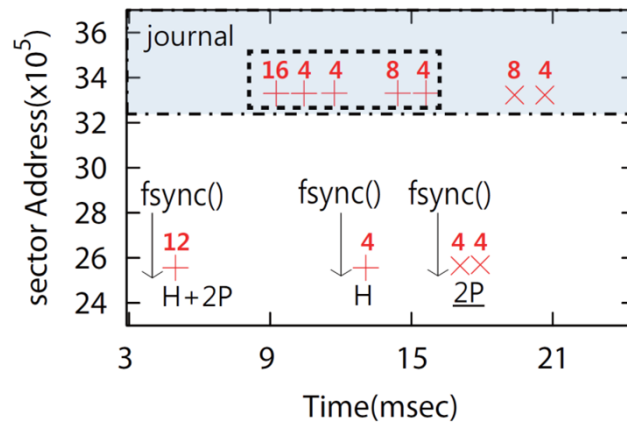
IO ACCESSES IN SQLITE FOR AN INSERT



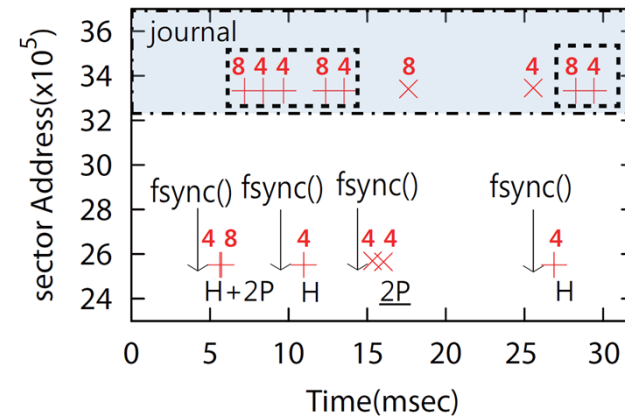
DELETE mode: INSERT



WAL mode : INSERT



TRUNCATE mode : INSERT

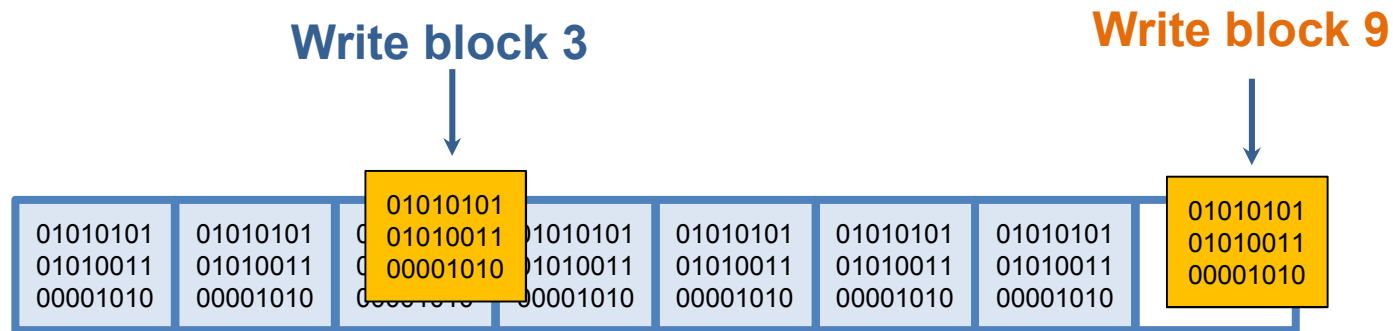


PERSIST mode : INSERT

Approach

002. **Resolving the Journaling
of Journal Anomaly**

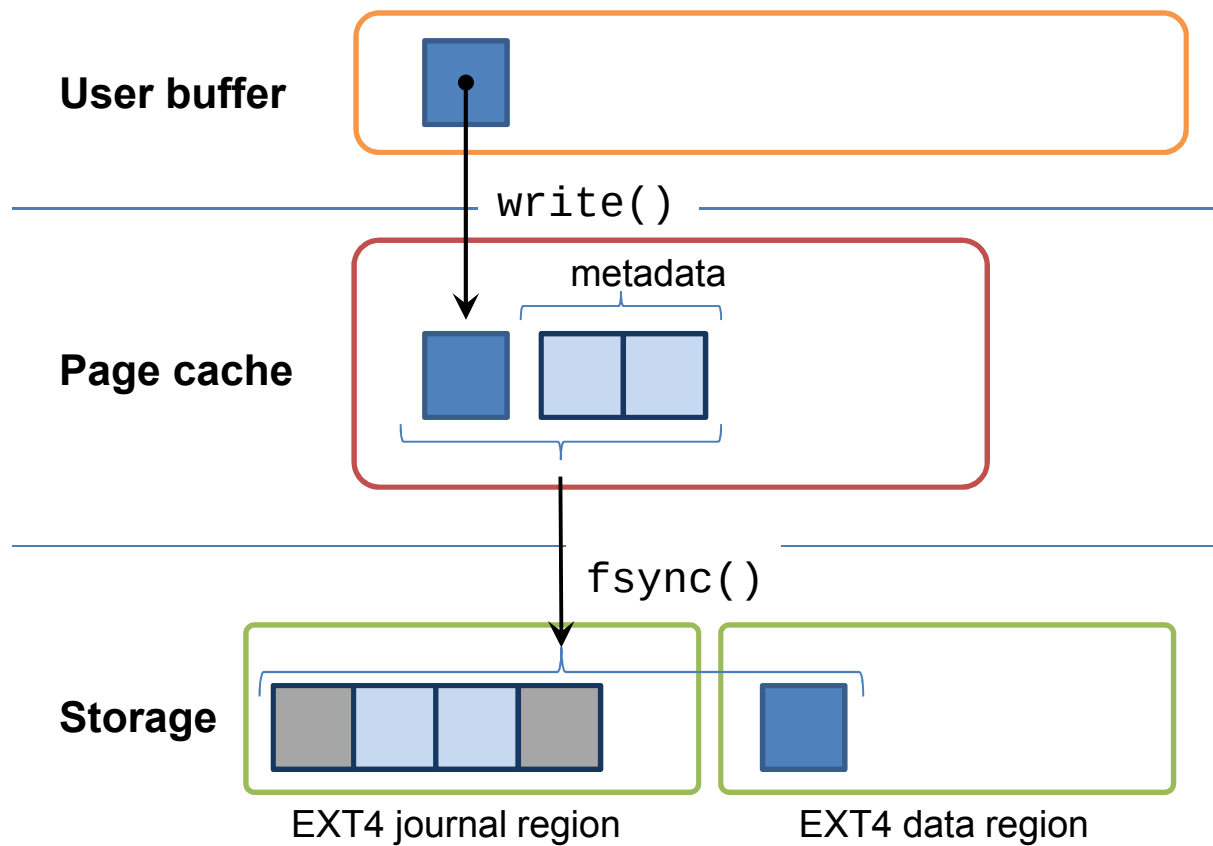
ALLOCATING A BLOCK IN WRITE



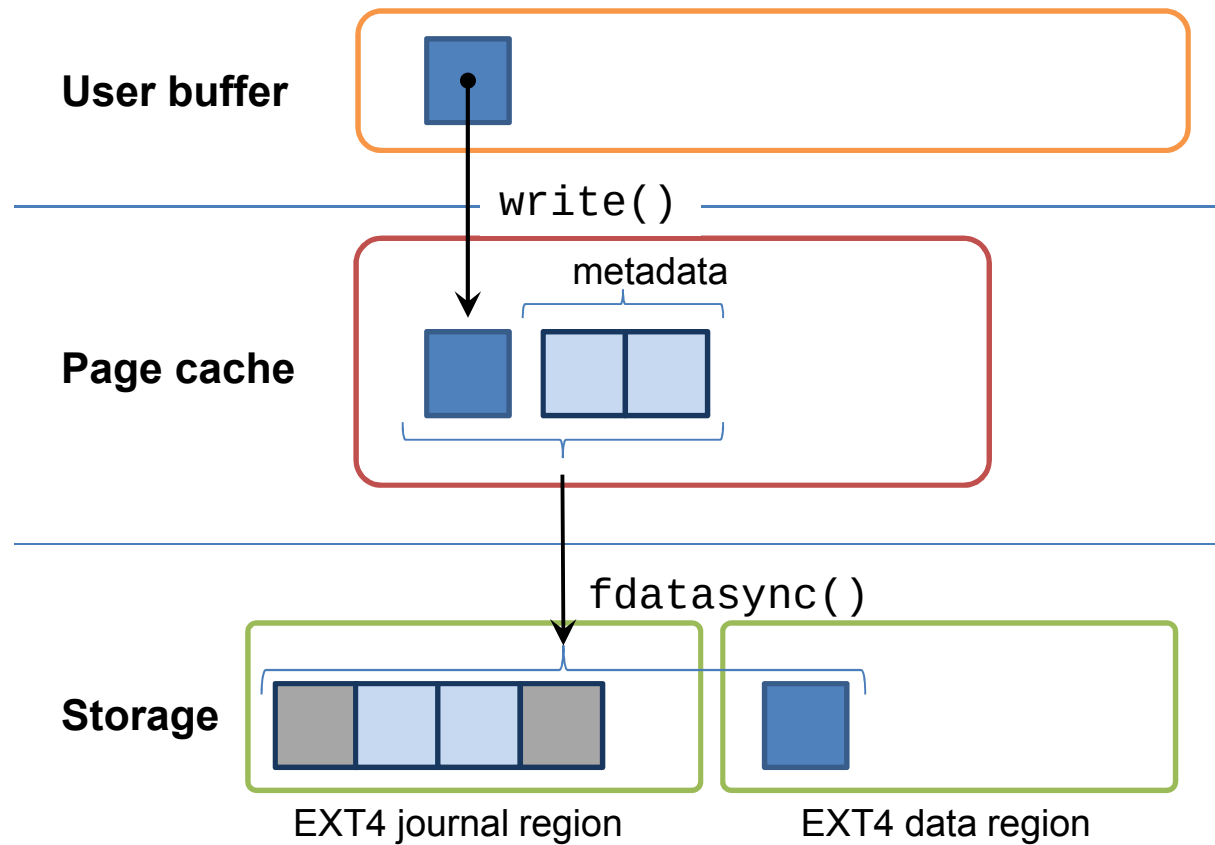
Write block 3: Non-Allocating write, No metadata update

Write block 9: Allocating write, metadata updated

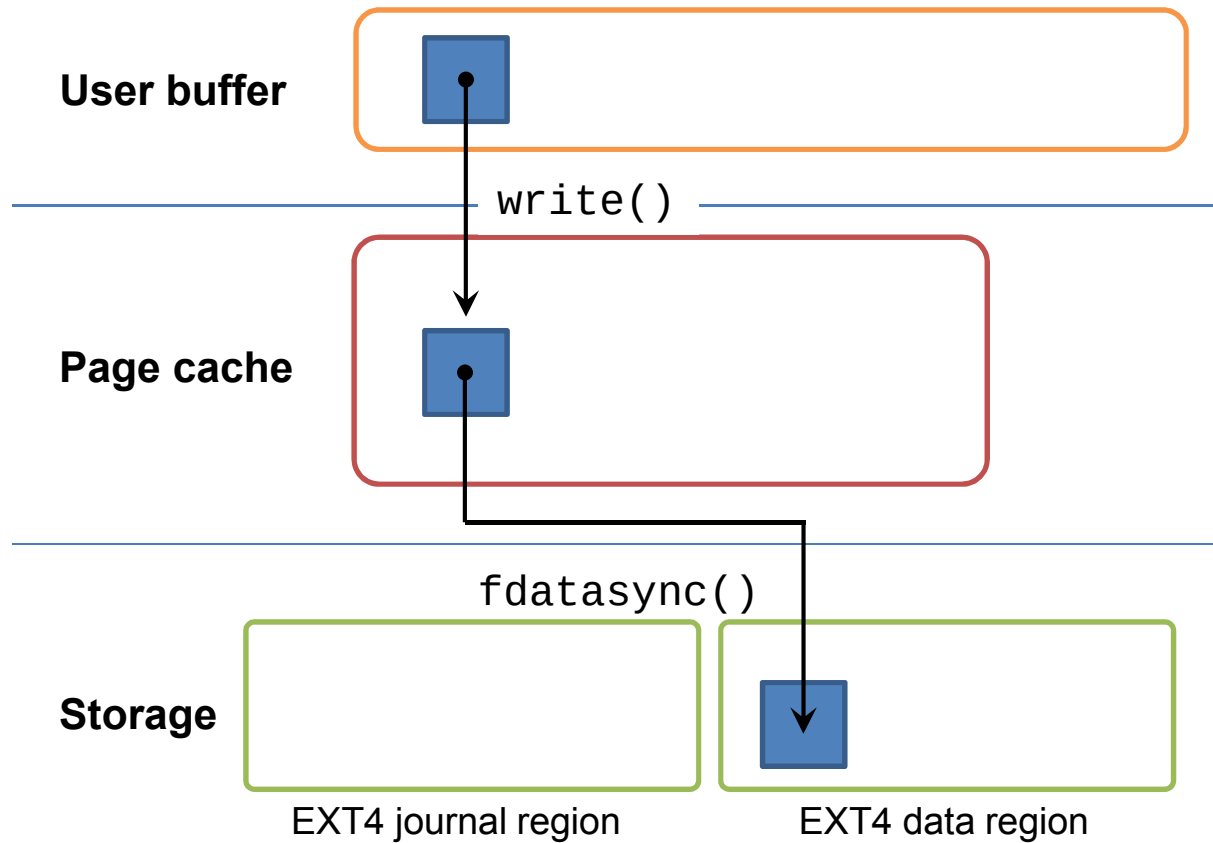
WRITING A BLOCK TO THE STORAGE: VIA FSYNC ()



WRITING A BLOCK TO THE STORAGE: VIA FDATASYNC(): ALLOCATING WRITE

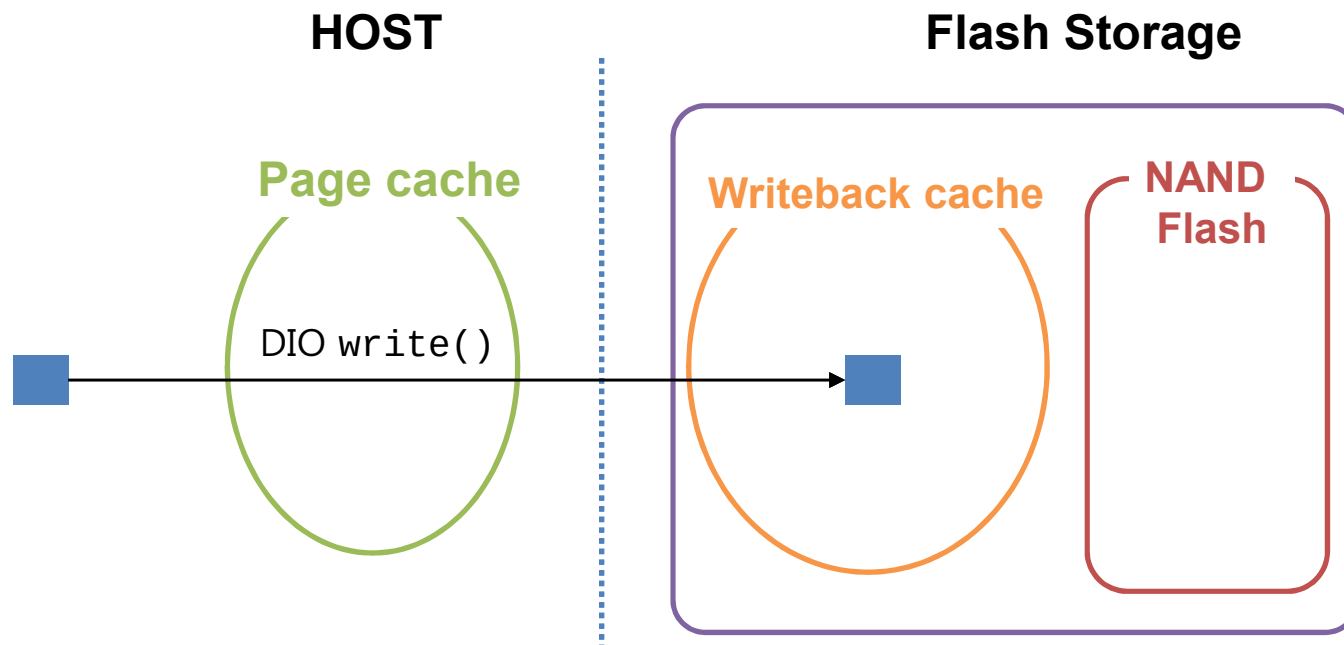


WRITING A BLOCK TO THE STORAGE: VIA FDATASYNC(): NON ALLOCATING WRITE

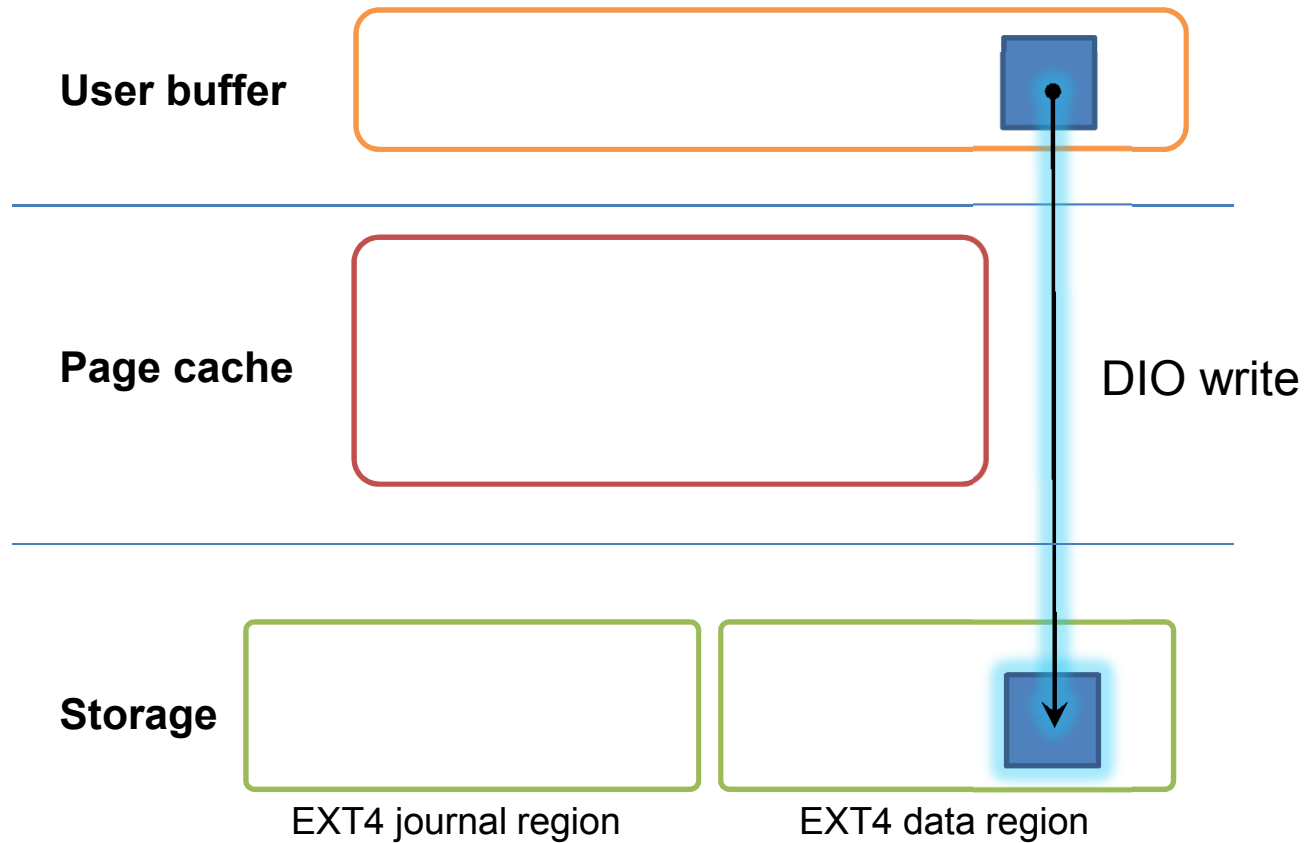


DIRECT IO

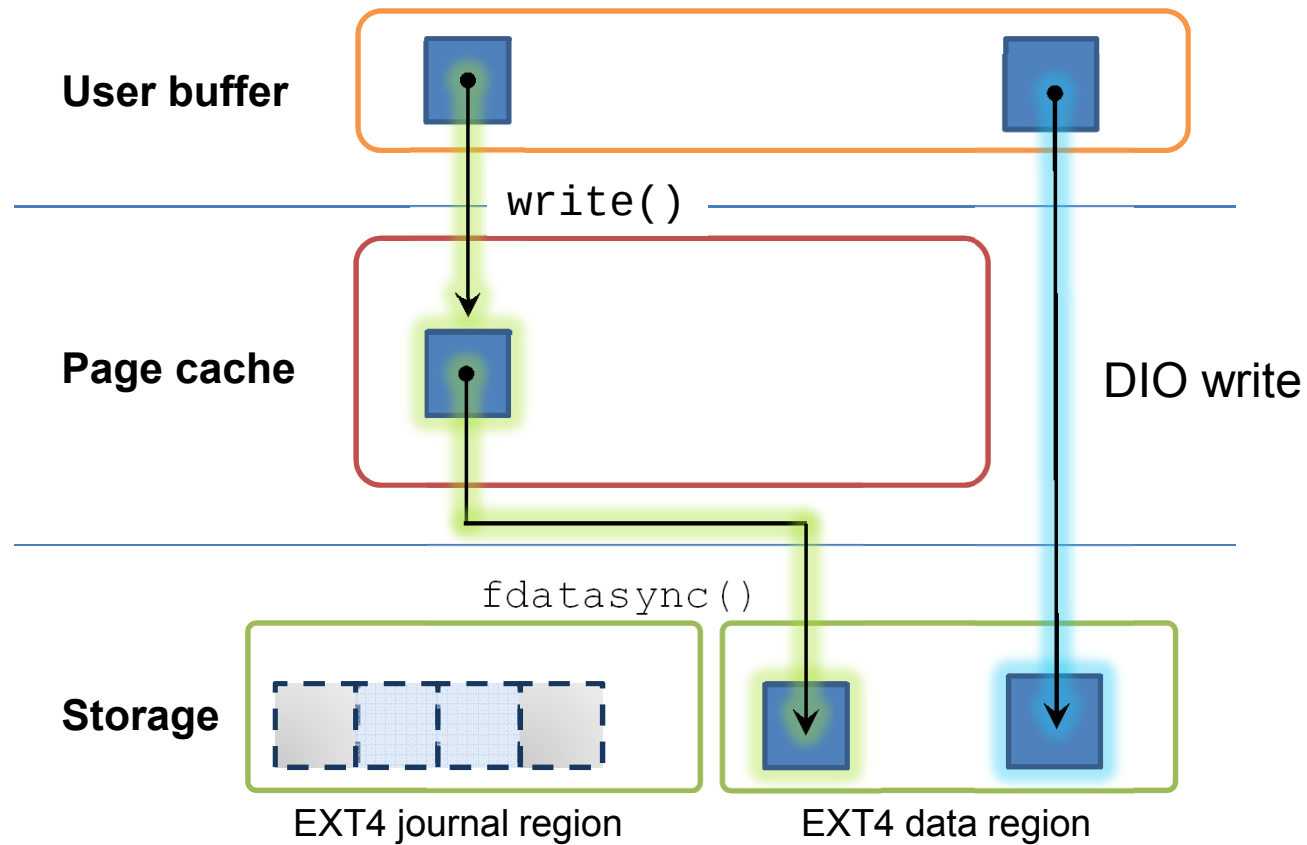
- Bypass the page cache.
- Write directly to the storage
- Write to writeback cache at the storage.
- IO should be sector aligned.



WRITING A BLOCK TO THE STORAGE VIA DIRECT IO: NON-ALLOCATING WRITE



For Non-Allocating write,
DIO write == buffered write() + fdatasync()





INSERT

SQLite

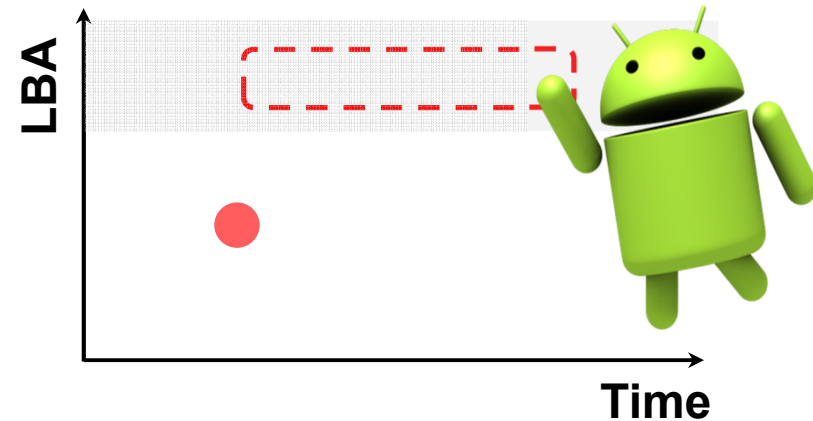
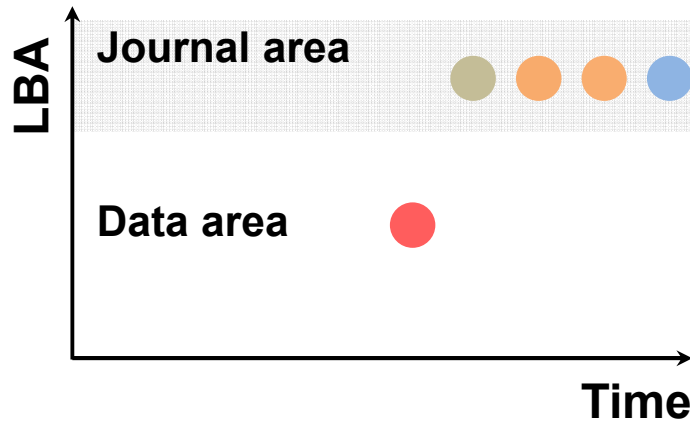
Buffered write
to journal

VS

Direct write
to journal



**We use Direct IO for
log commit in SQLite!!**



Problem

003. **Problem Assessment**

ISSUES IN USING DIO WRITE FOR SQLITE JOURNALING

- **Issue 1**

Avoid metadata update in SQLite log commit.

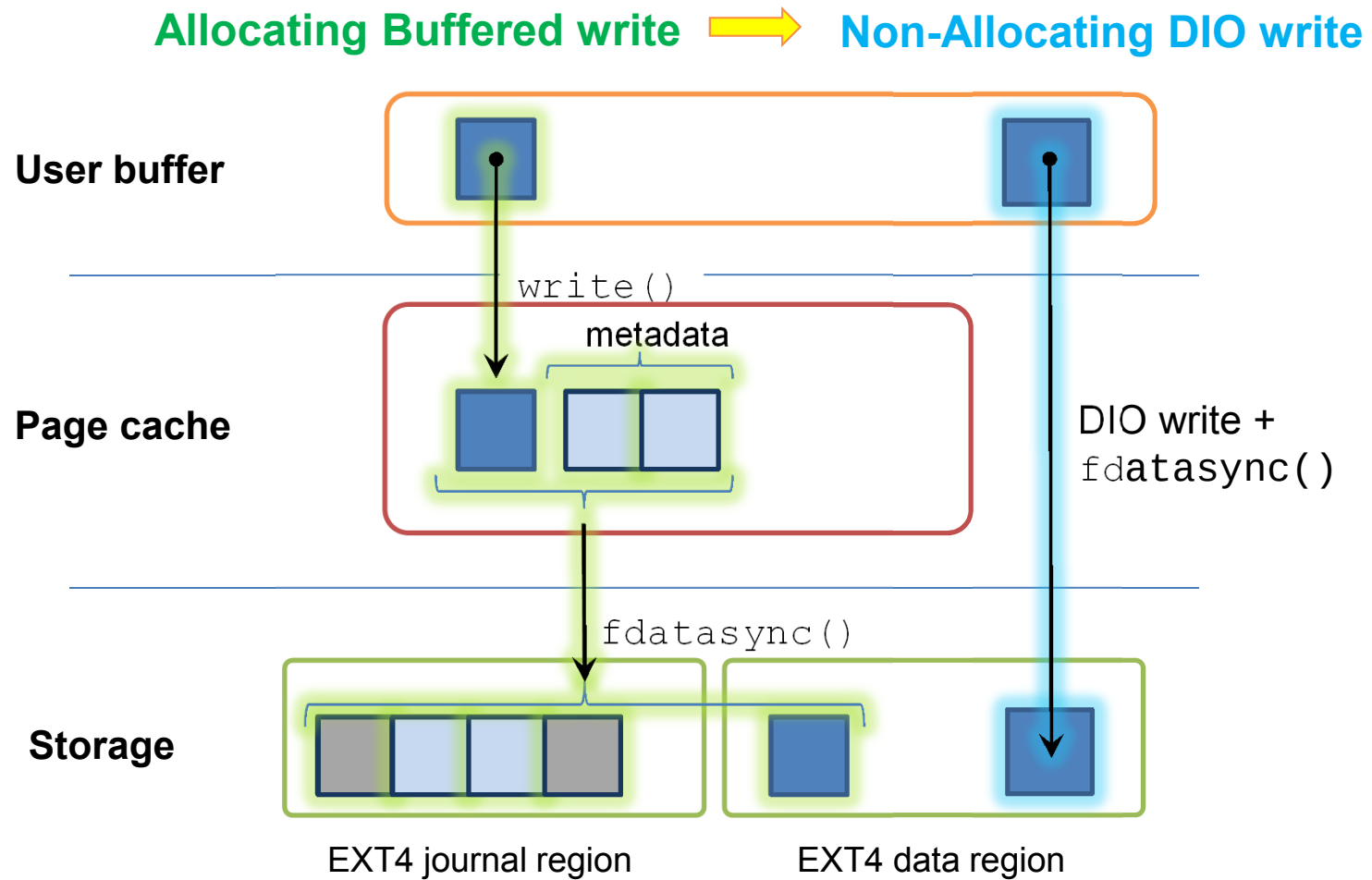
- **Issue 2**

Make the SQLite IO size direct IO compatible and block device friendly.

- **Issue 3**

Make the DIO write efficient and flash memory friendly.

ISSUE 1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT



ISSUES IN USING DIO WRITE FOR SQLITE JOURNALING

- **Issue 1**

Avoid metadata update in SQLite log commit.

- **Issue 2**

Make the SQLite IO size direct IO compatible and block device friendly.

- **Issue 3**

Make the DIO write efficient and flash memory friendly.

ISSUES IN USING DIO WRITE FOR SQLITE JOURNALING

- **Issue 1**

Avoid metadata update in SQLite log commit.

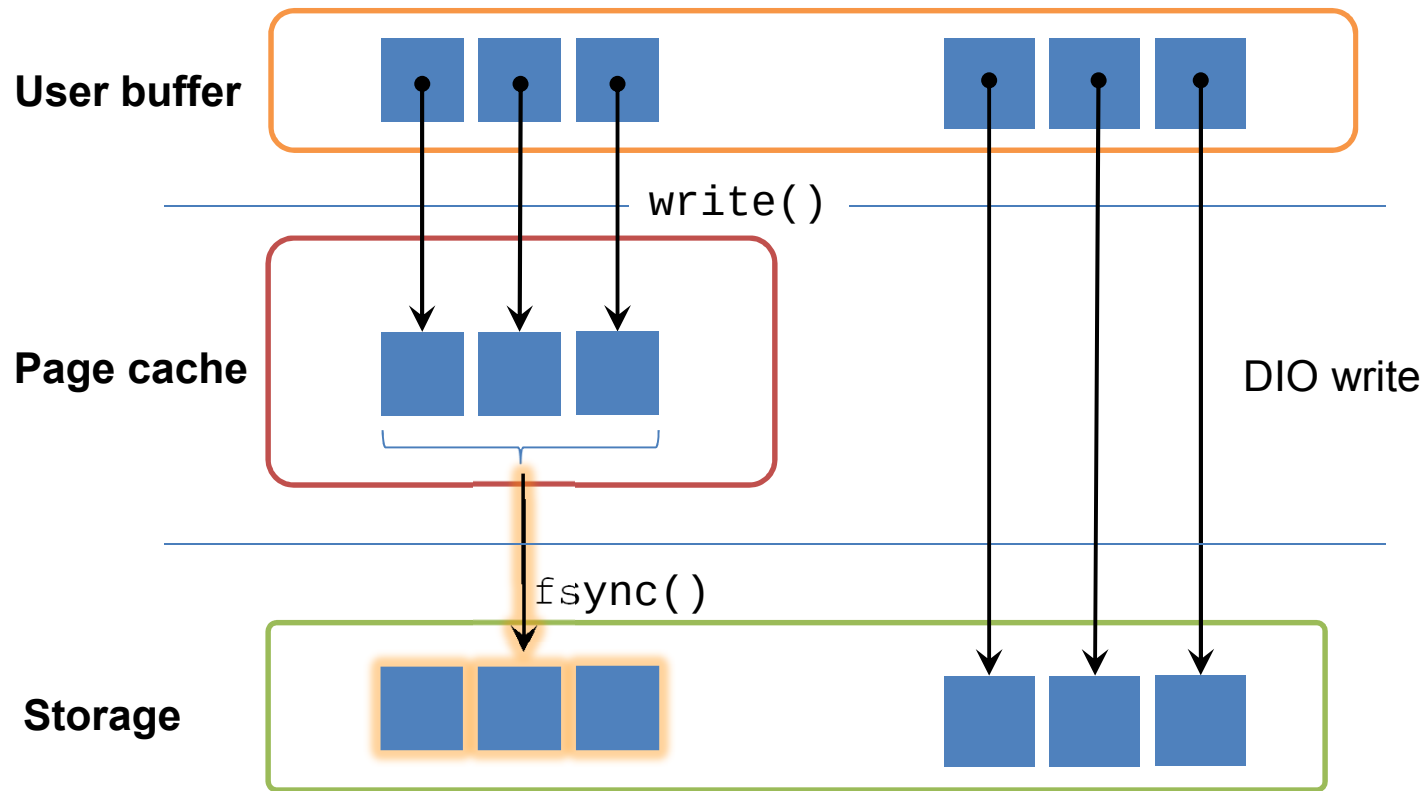
- **Issue 2**

Make the SQLite IO size direct IO compatible and block device friendly.

- **Issue 3**

Make the DIO write efficient and flash memory friendly.

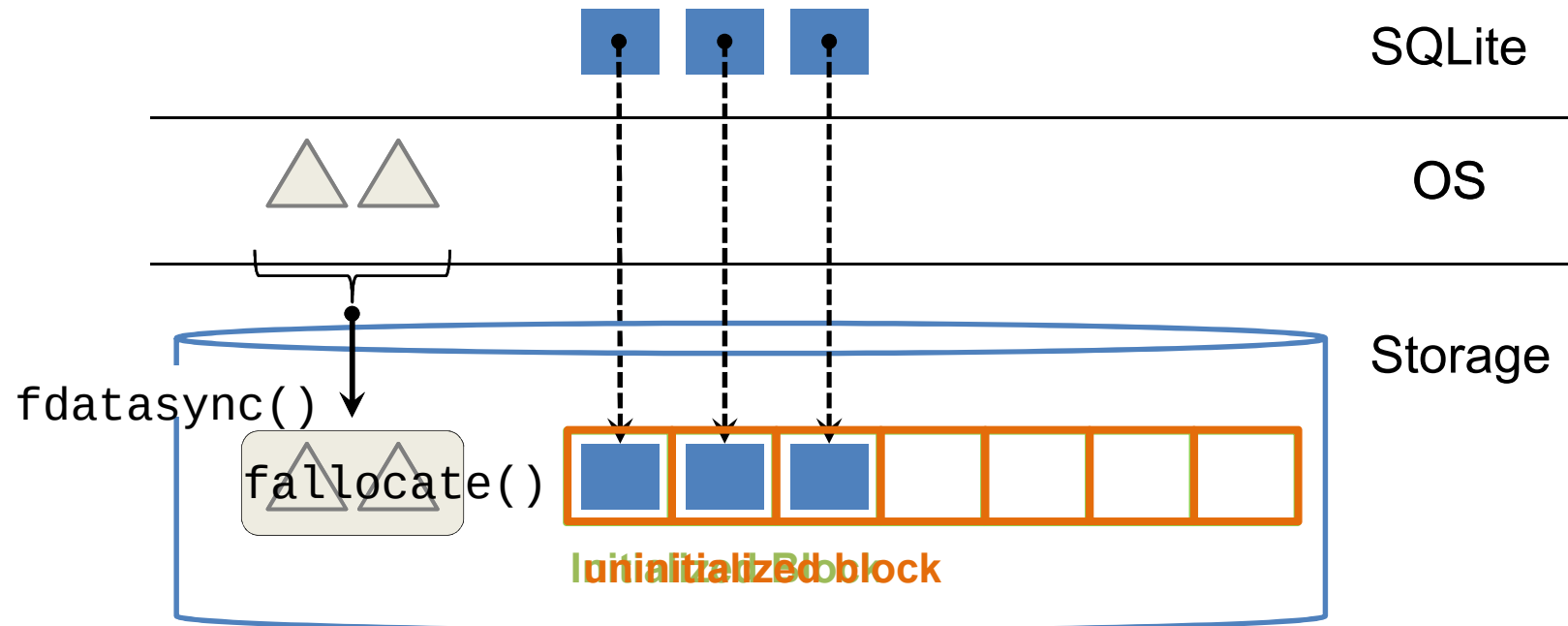
ISSUE 3. MAKE THE DIO WRITE MORE EFFICIENT



004. Preallocation with Explicit Journaling

ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

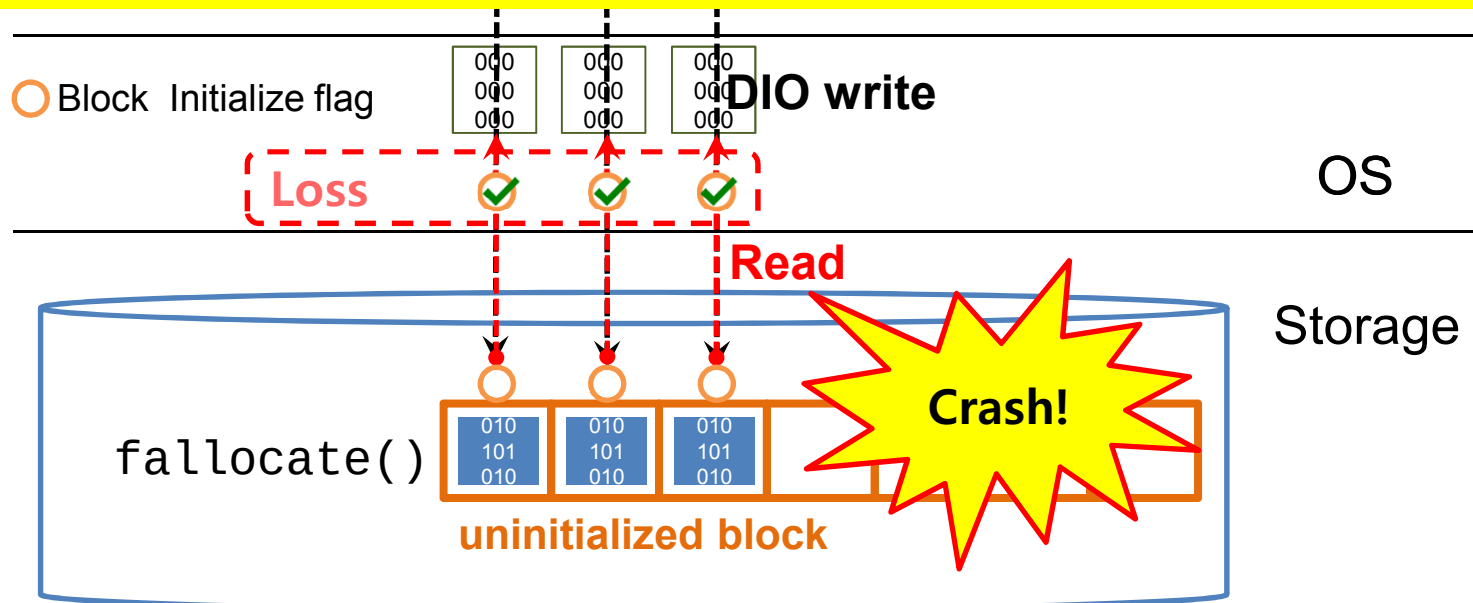
PREALLOCATION WITH EXPLICIT JOURNALING



ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

INITIALIZING THE ALLOCATED BLOCKS

Preallocation with Explicit Journaling should return the initialized blocks without exposing the stale content.



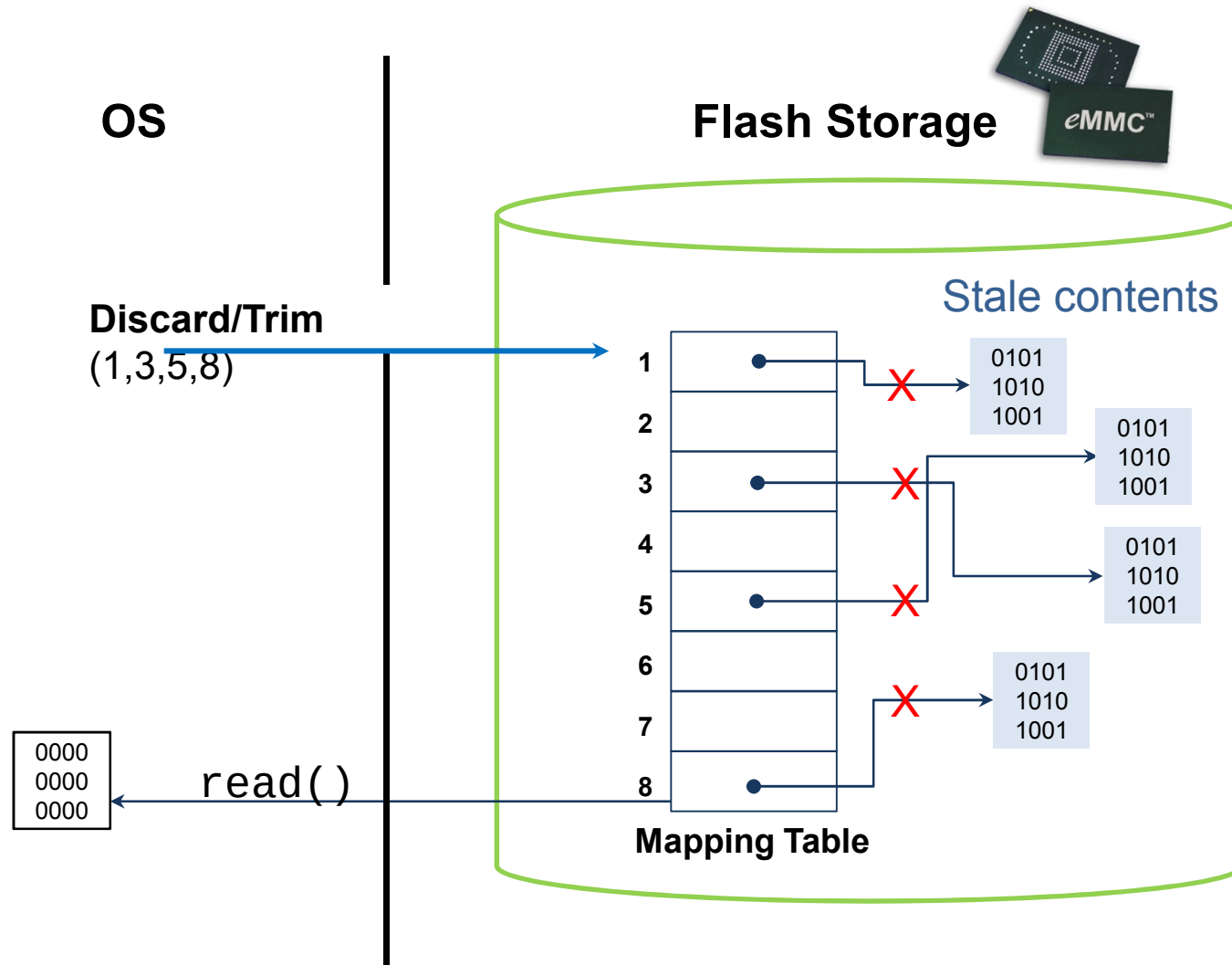
ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

INITIALIZING THE ALLOCATED BLOCKS

- **Zero-Fill the allocated blocks.**
- **Allocating the blocks with “initialized” flag set.**
- **We use discard/trim to avoid the exposure of the stale content**
 - unmap the Flash pages when files are deleted (DISCARD mount option).
 - unmap the Flash pages when the blocks are allocated (discard pages within `fallocate()`).

ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

DISCARD / TRIM COMMAND



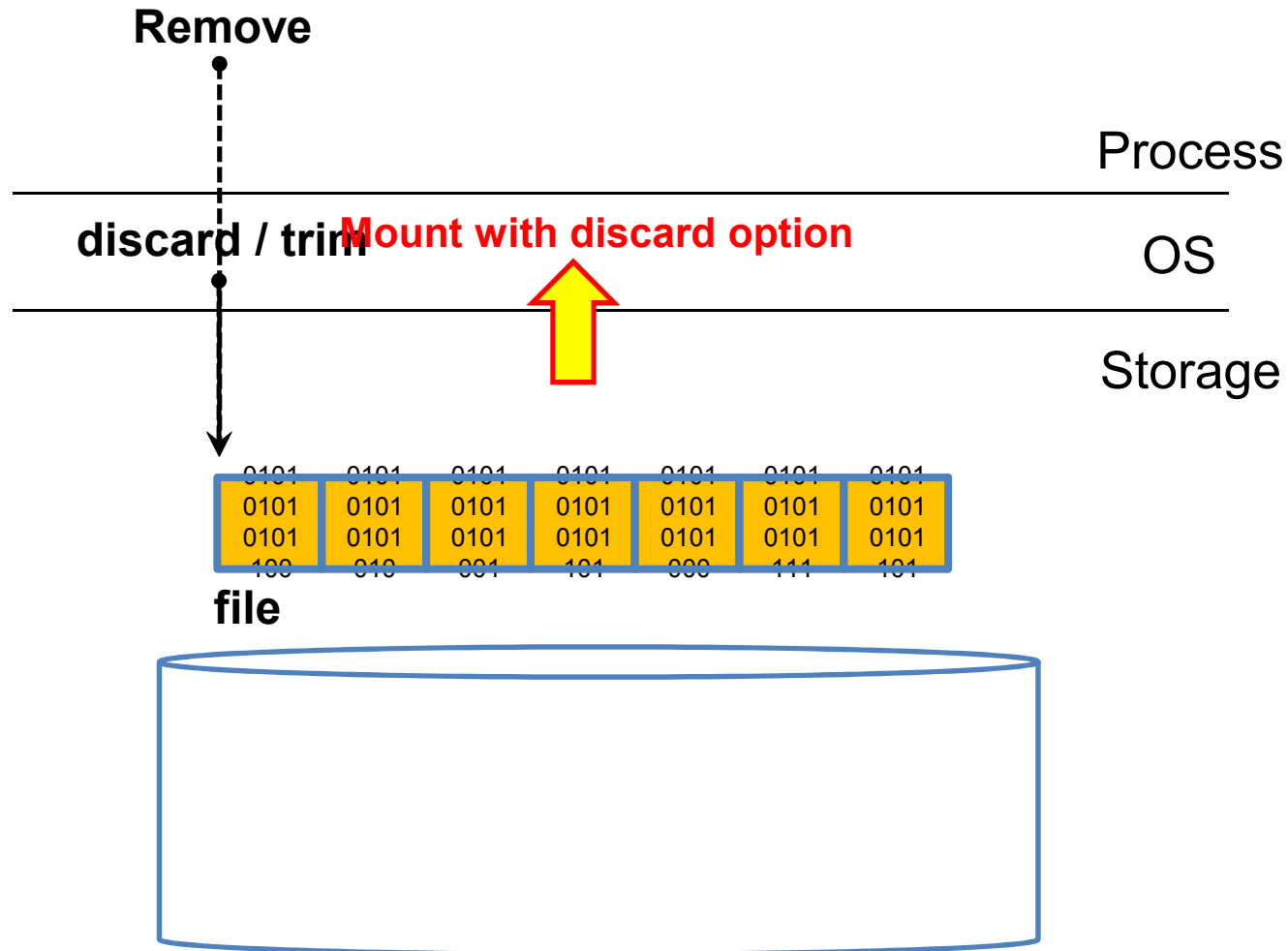
ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

INITIALIZING THE ALLOCATED BLOCKS

- **Zero-Fill the allocated blocks.**
- **Allocating the blocks with “initialized” flag set.**
- **We use discard/trim to avoid the exposure of the stale content**
 - unmap the Flash pages when files are deleted (DISCARD mount option).
 - unmap the Flash pages when the blocks are allocated (discard pages within `fallocate()`).

ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

APPROACH 2. ALLOCATING INITIALIZED BLOCK WITH DISCARD MOUNT



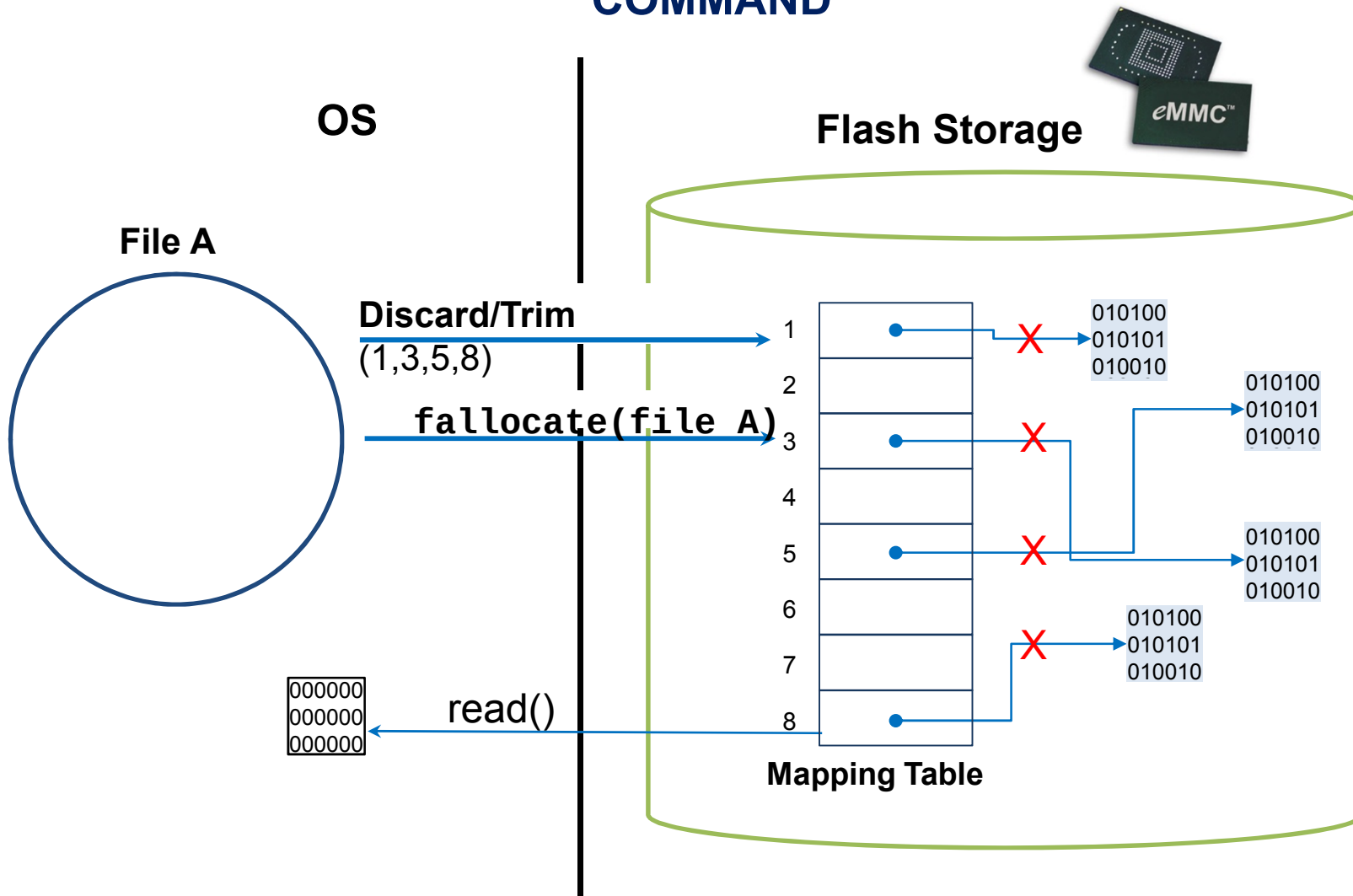
ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

INITIALIZING THE ALLOCATED BLOCKS

- **Zero-Fill the allocated blocks.**
- **Allocating the blocks with “initialized” flag set.**
- **We use discard/trim to avoid the exposure of the stale content**
 - unmap the Flash pages when files are deleted (DISCARD mount option).
 - unmap the Flash pages when the blocks are allocated (discard pages within `fallocate()`).

ISSUE1. AVOID METADATA UPDATE IN SQLITE LOG COMMIT

APPROACH 3. ALLOCATING INITIALIZED BLOCK WITH DISCARD COMMAND

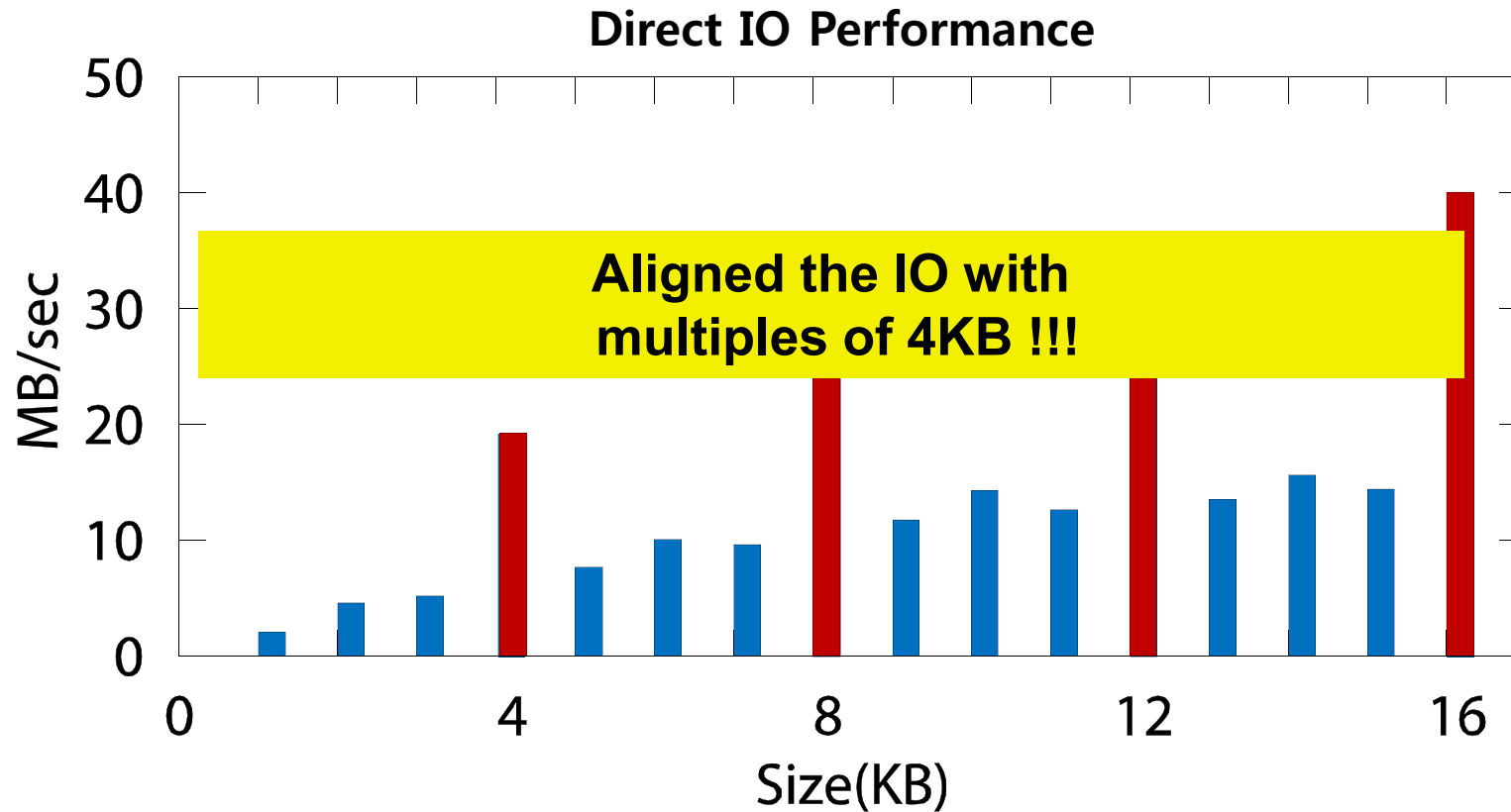


HE

005.**Header Embedding**

ISSUE2. MAKE THE SQLITE IO SIZE DIRECT IO COMPATIBLE

IO SIZE VS. IO PERFORMANCE



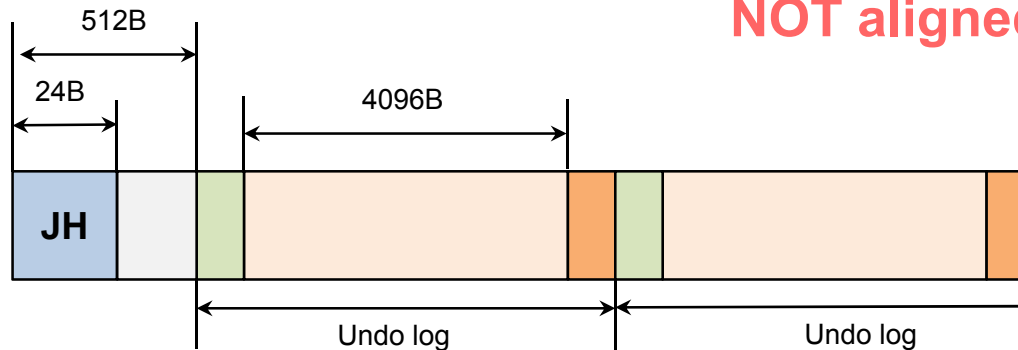
ISSUE2. MAKE THE SQLITE IO SIZE DIRECT IO COMPATIBLE

SQLITE JOURNAL FILE STRUCTURE

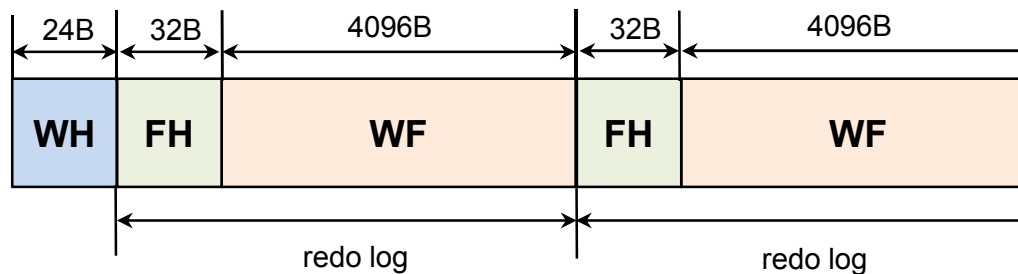
- Undo log file

NOT compatible with direct IO

NOT aligned sector size !!

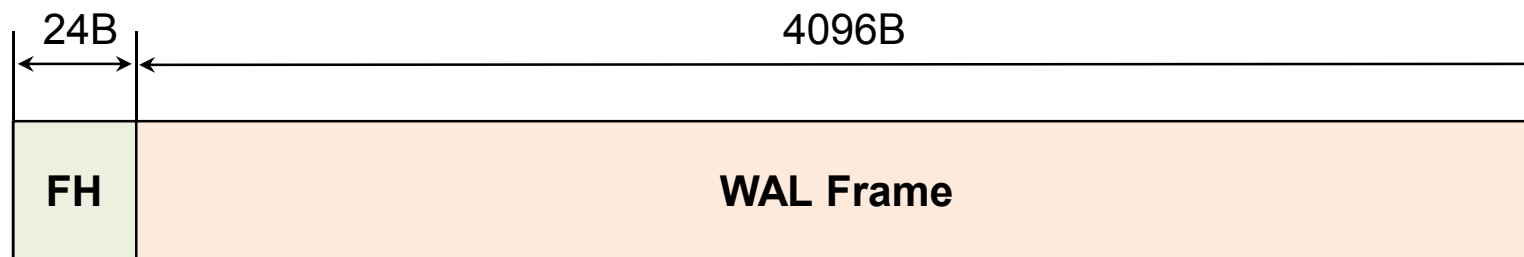


- Redo log file

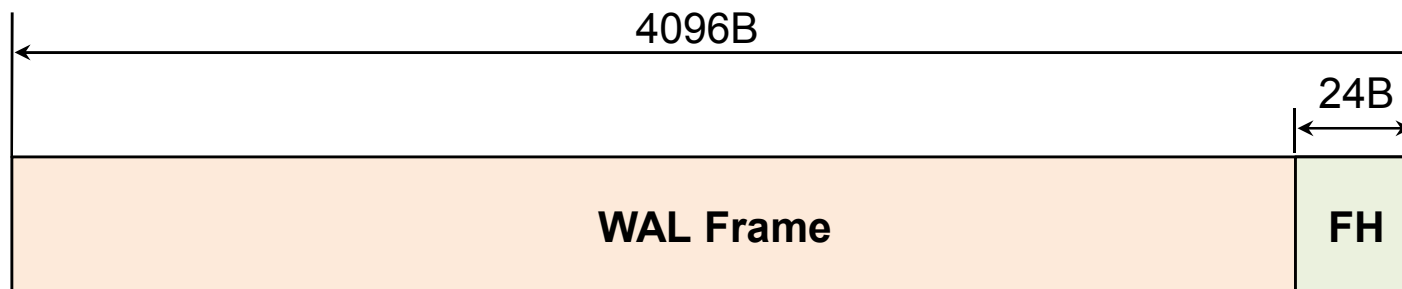


ISSUE2. MAKE THE SQLITE IO SIZE DIRECT IO COMPATIBLE

HEADER EMBEDDING



HEADER EMBEDDING



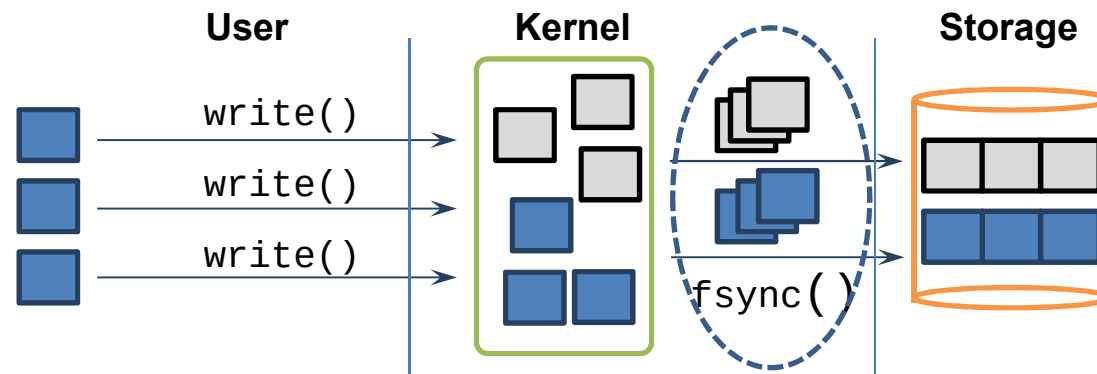
GS

006.**Group Synchronization**

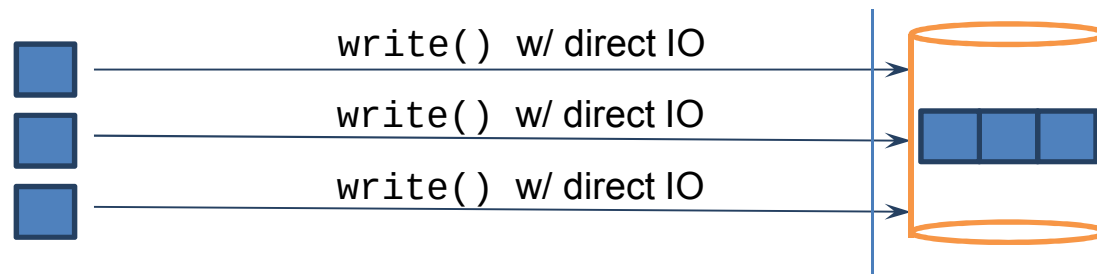
ISSUE3. MAKE THE DIO WRITE EFFICIENT

DIO WRITE IS INEFFICIENT

OS merges the IO in buffered IO.
IO.

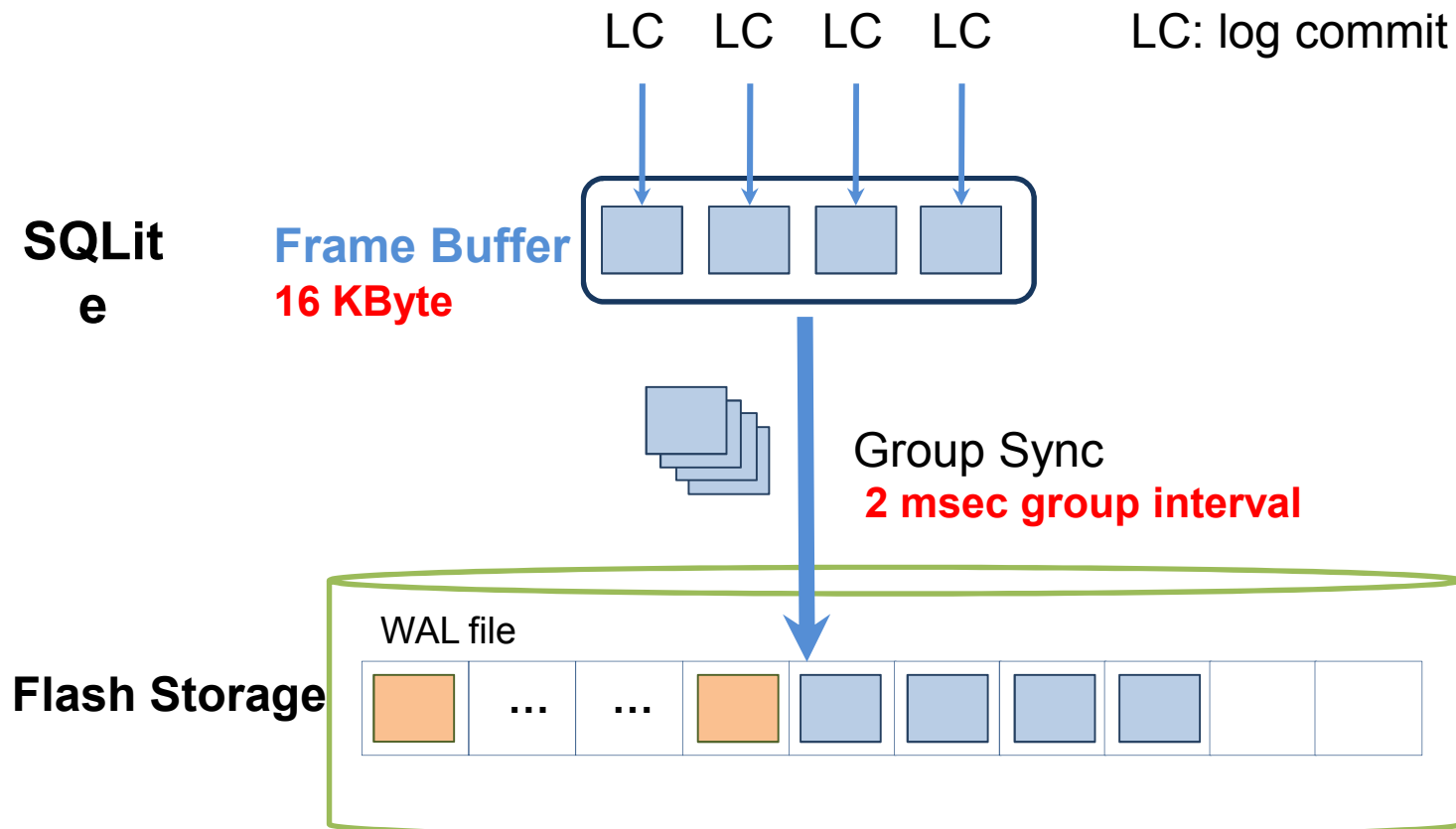


Direct IO write cannot be merged.



ISSUE3. MAKE THE DIO WRITE EFFICIENT

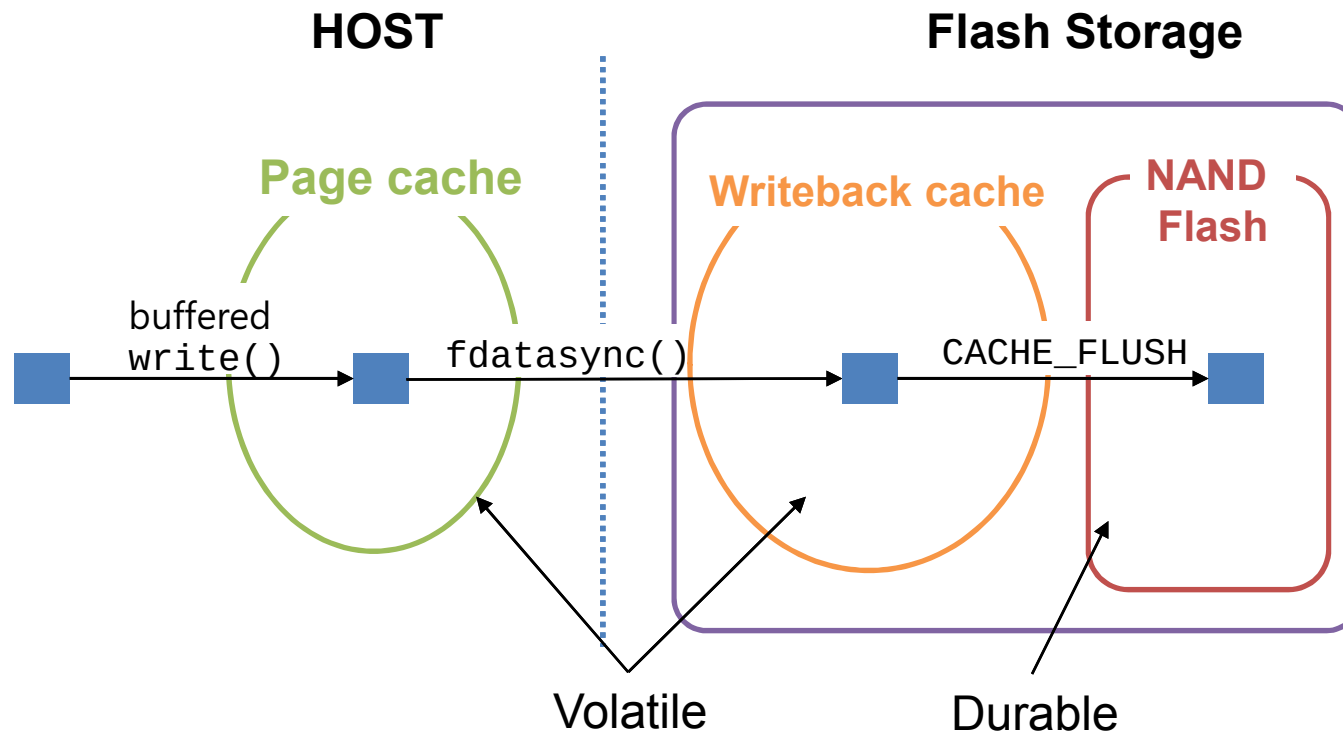
GROUP SYNCHRONIZATION



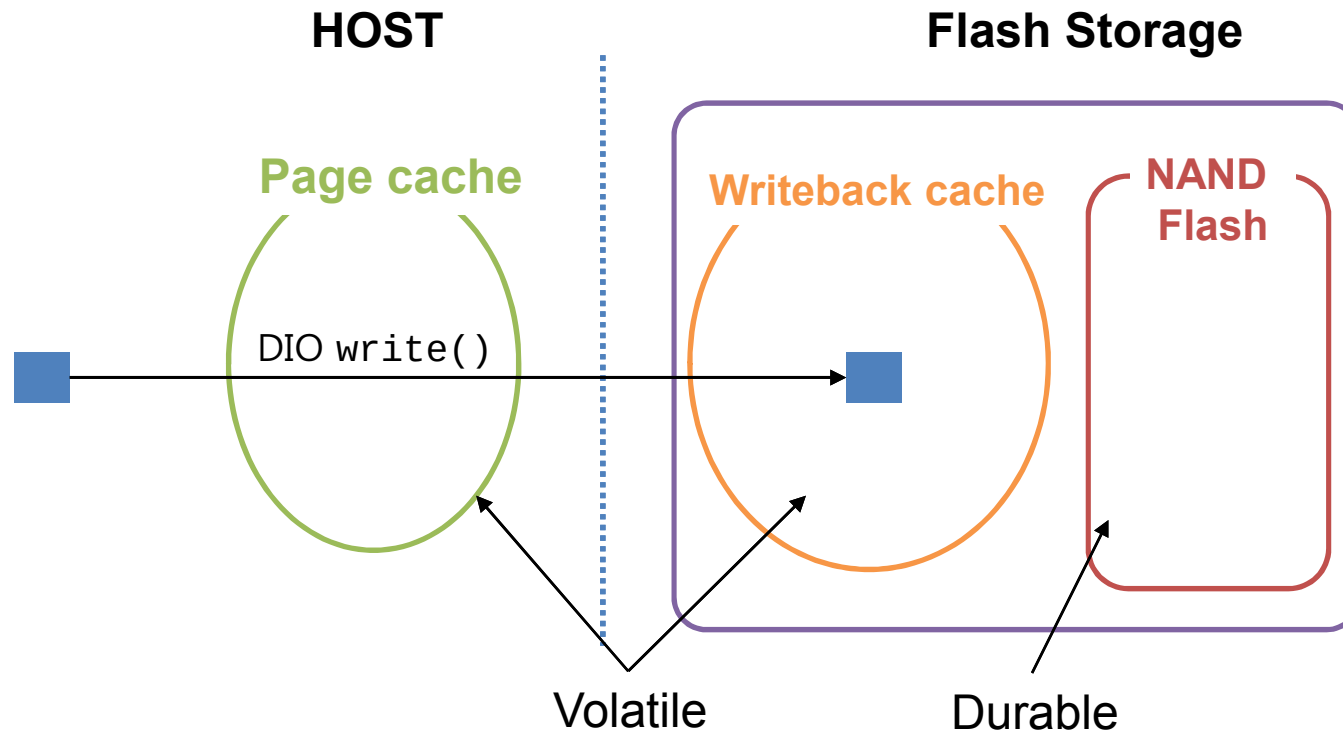
DB

007. **Durability**

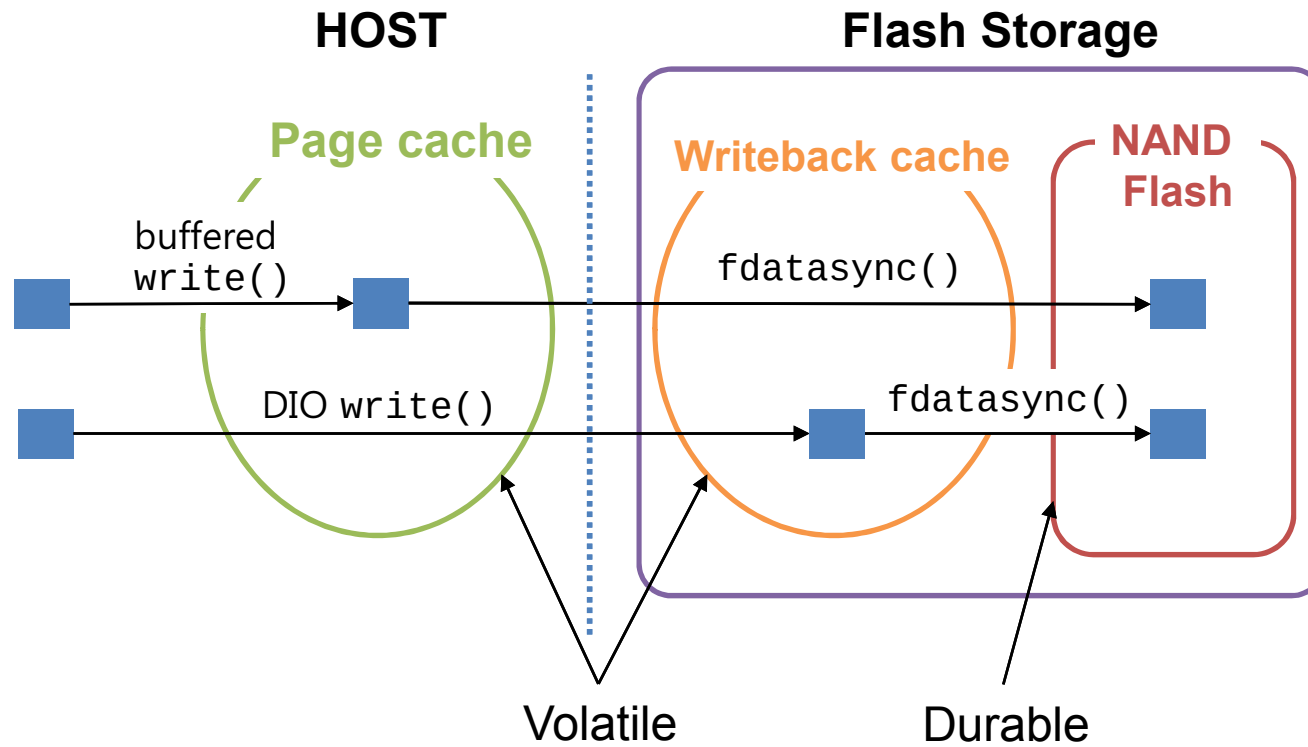
DURABILITY GUARANTEE OF BUFFERED IO



DURABILITY OF DIRECT IO

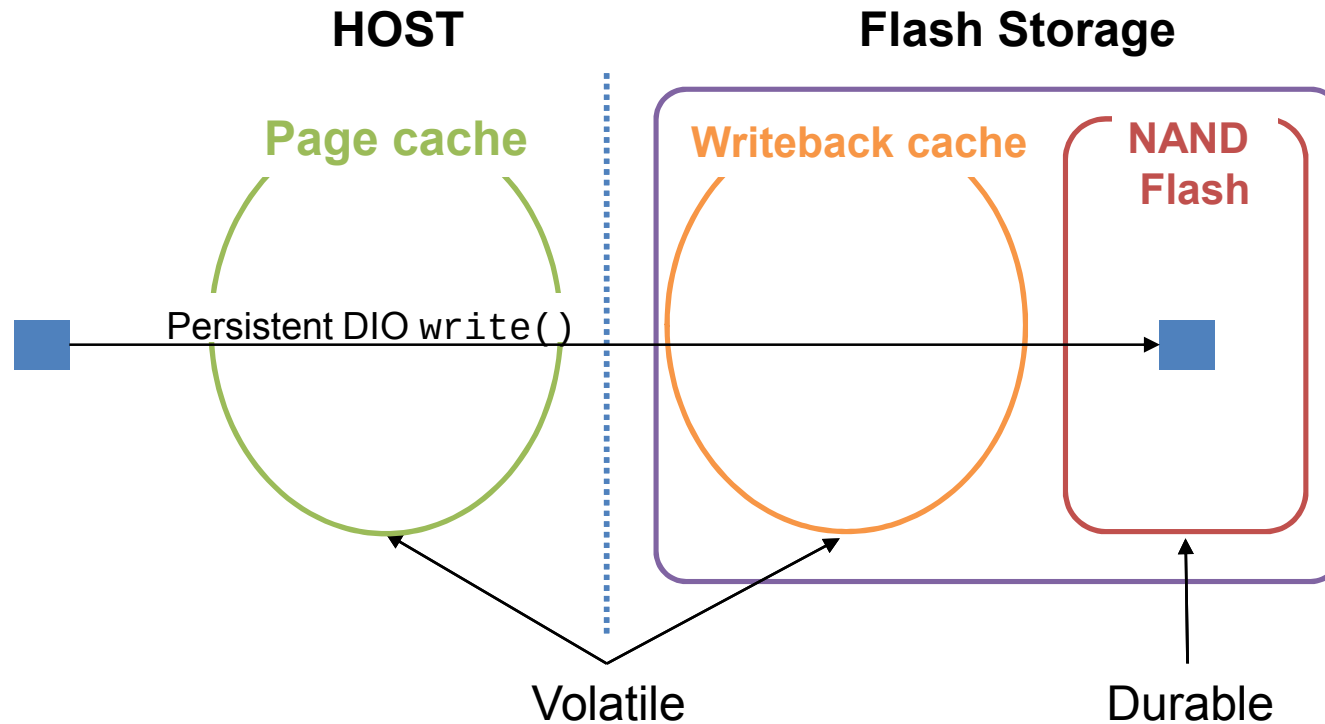


TYPE 1. USE OF FULL SYNC



TYPE 2. PERSISTENT DIRECT IO

- We develop “Persistent Direct IO” using “reliable write” command in eMMC5.1
- Eliminating the writeback cache flush overhead.

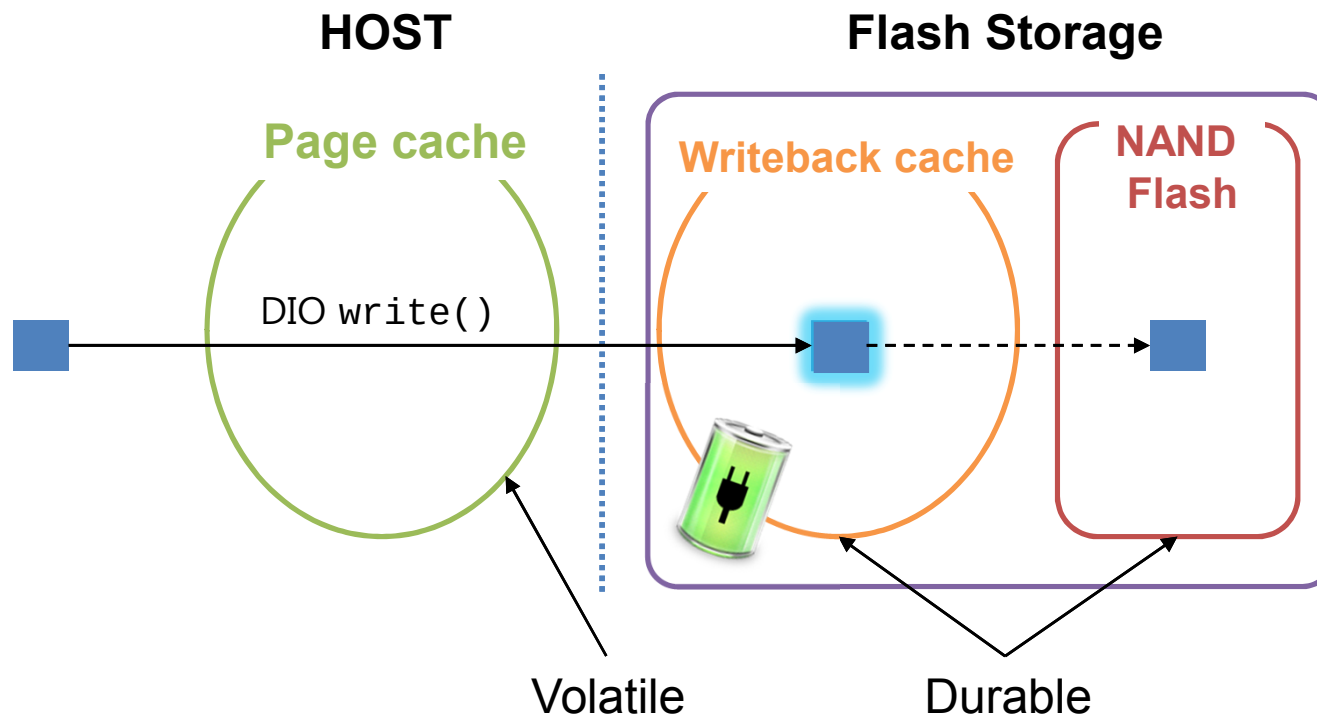


TYPE 3. SMARTPHONE WITH NON-REMOVABLE BATTERY

The writes to the writeback cache can become durable.

We do not need explicit cache flush!!!

We can use normal sync option



Evaluation

008. **Experiment**

IMPLEMENTATION

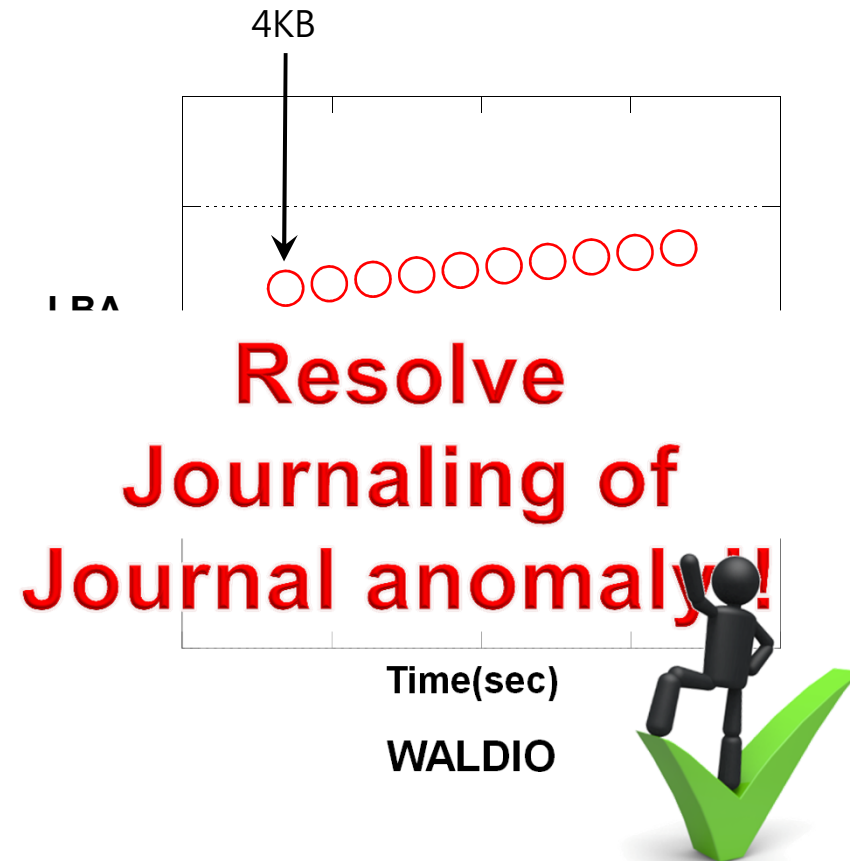
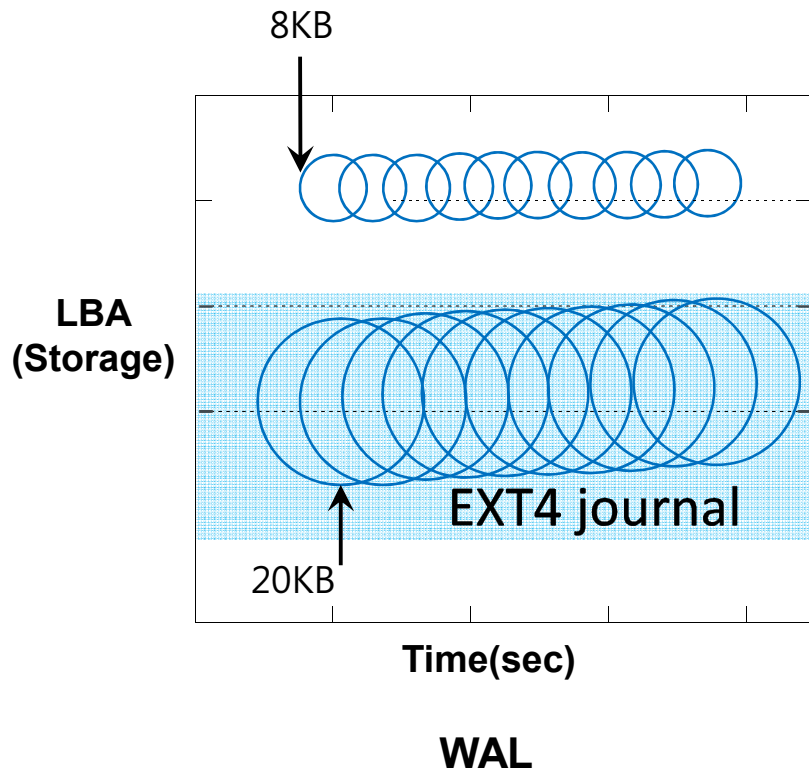
Galaxy S5 (KitKat 4.4.2, Linux 3.4)



Component	Specification
AP	Snapdragon 801 MSM8974AC
CPU	Krait 2.457 GHz Quad-core
RAM	2 GB / LPDDR3 SDRAM
Internal Storage	32 GB eMMC

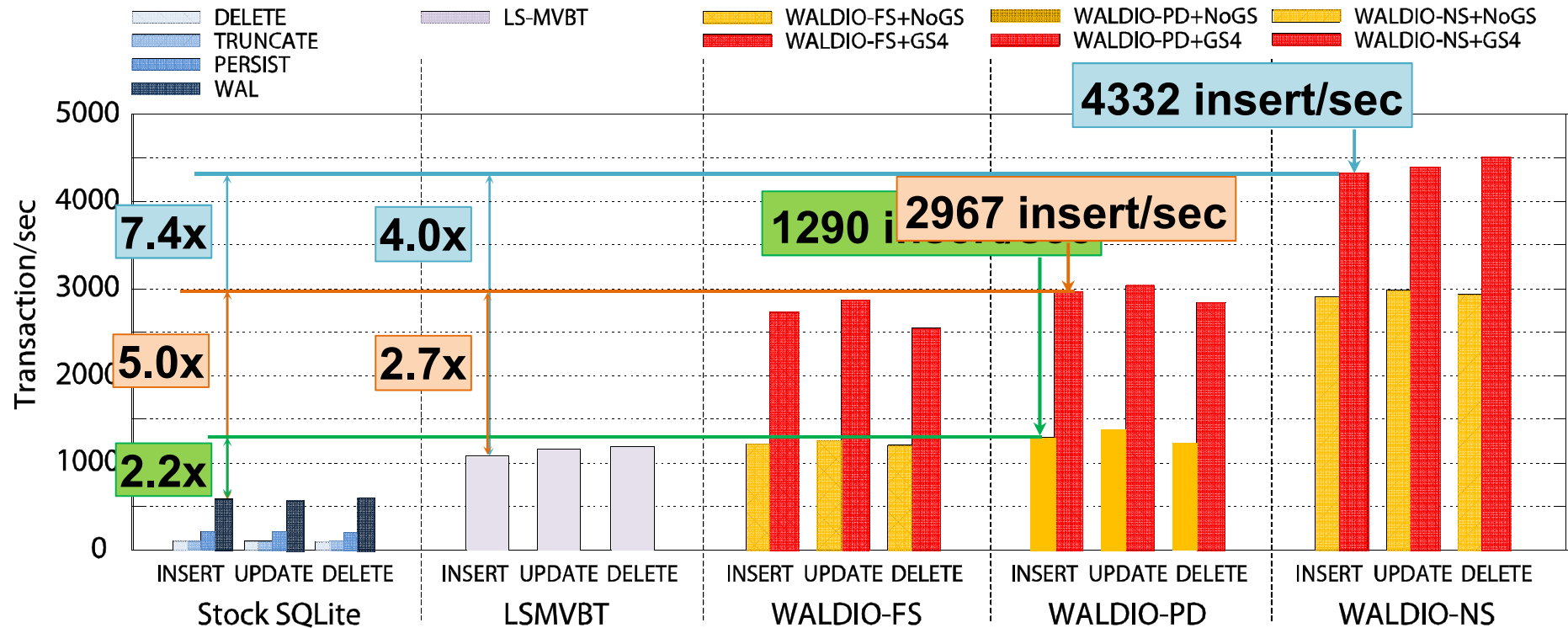
IO TRACE: WAL VS. WALDIO IO TRACE

Insert 100Byte , 10 times



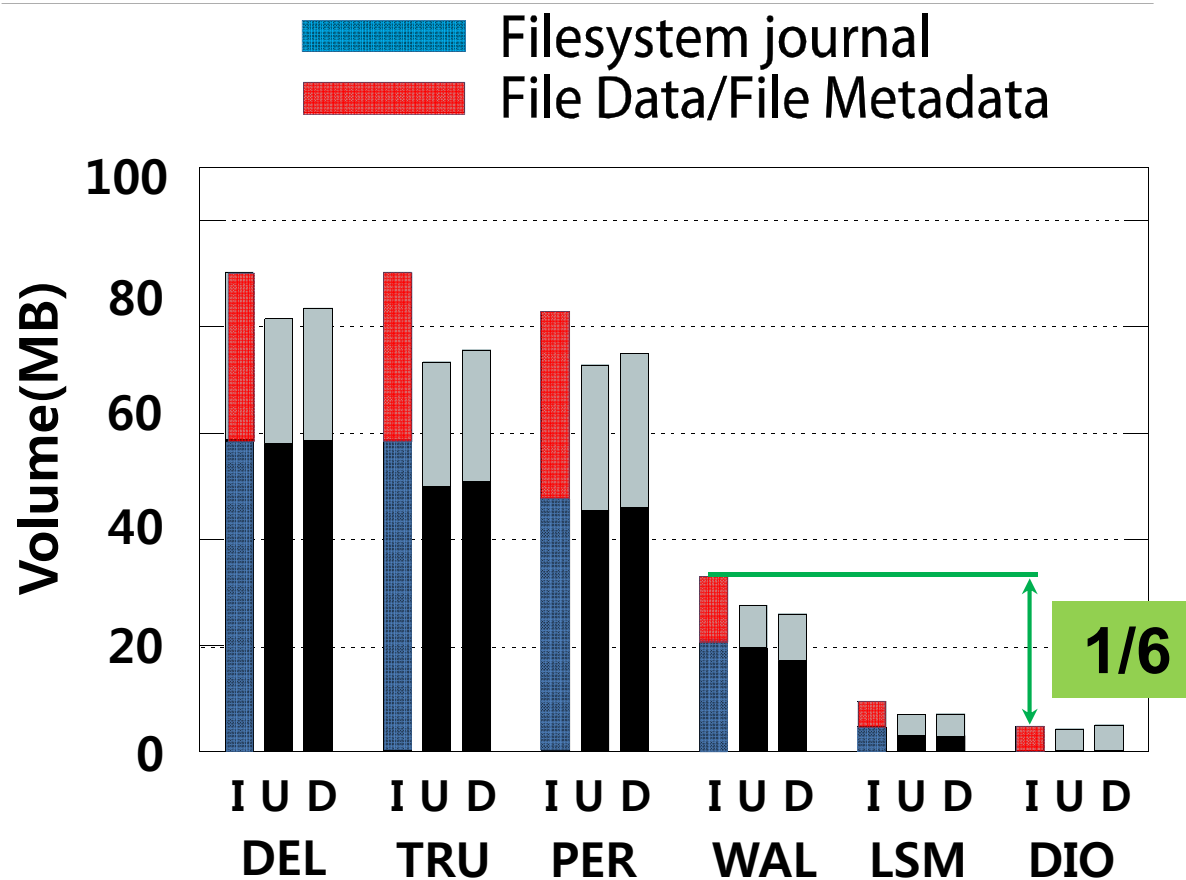
PERFORMANCE COMPARISON

Insert 100Byte , 10000 times



IO VOLUME AT SQLITE OPERATION

Insert 100Byte , 10000 times



SUMMARY

- We develop Preallocation with Explicit Journaling, Header Embedding and Group Synch.
- We use direct IO in SQLite log commit avoiding expensive EXT4 log commit.
- We develop new SQLite journal mode “WALDIO”.
- As much as 7.4x performance against WAL,
- The total IO volume decreases to 1/6 against WAL.
- We successfully resolve Journaling of Journal anomaly.

Thank you !

Questions ?

**Many more to do, please contact
yjwon@hanyang.ac.kr**