2013 USENIX Annual Technical Conference (ATC'13)
SAN JOSE, CA, USA, JUNE 26~28, 2013

I/O Stack Optimization for Smartphones

**Sooman Jeong[1], Kisung Lee[2], Seongjin Lee[1],
Seoungbum Son[2], and Youjip Won[1]**

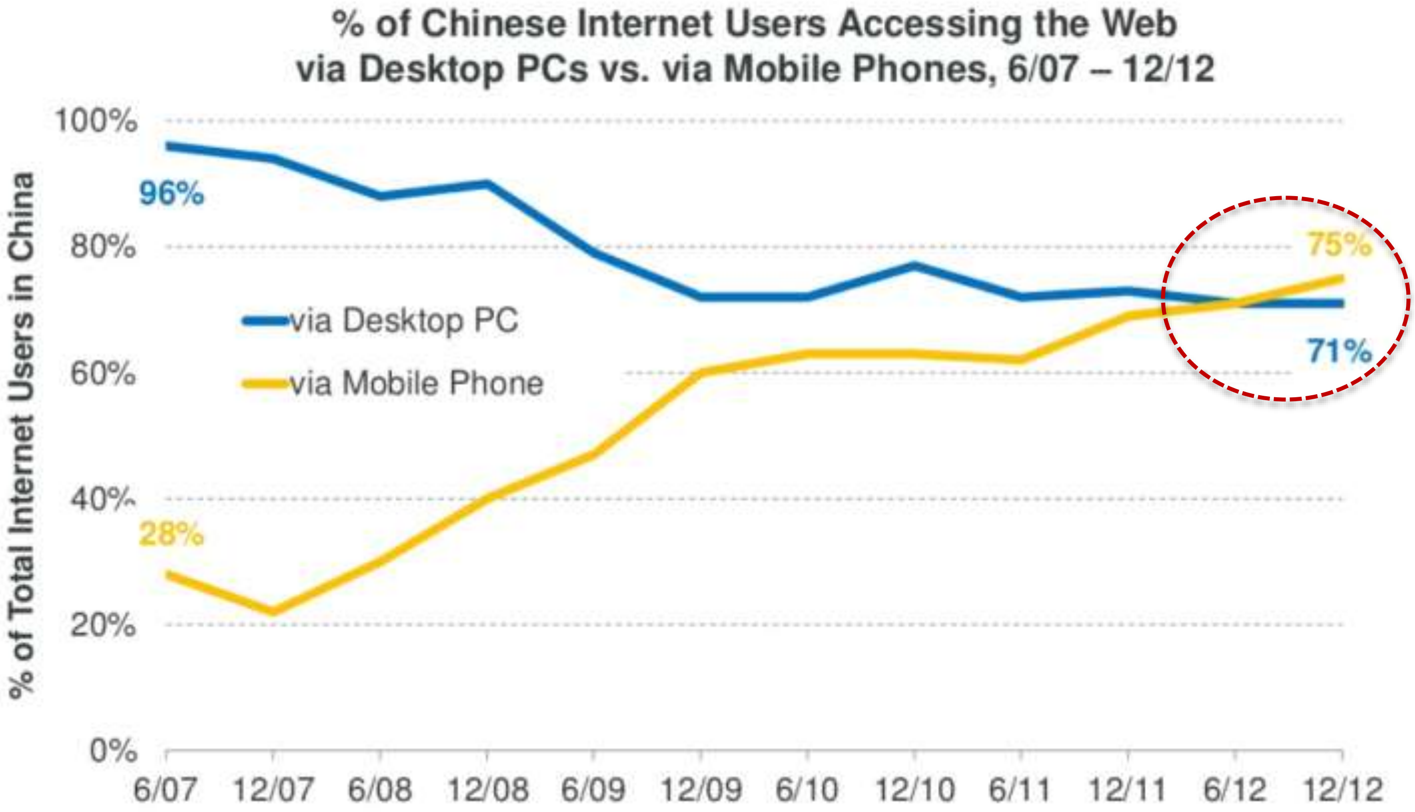
[1] Dept. of Electronics and Computer Engineering, Hanyang University
[2] Samsung Electronics

Outline

- Motivation
- Background
- Analysis of the Android I/O Stack
- Optimizations of the Android I/O Stack
    - Using the optimal journaling mode in SQLite
    - Alternative Filesystems
    - Eliminating unnecessary metadata flushes
    - External journaling
    - Using polling based I/O
- Evaluations
- Demo

Motivation

Smartphone is everywhere!

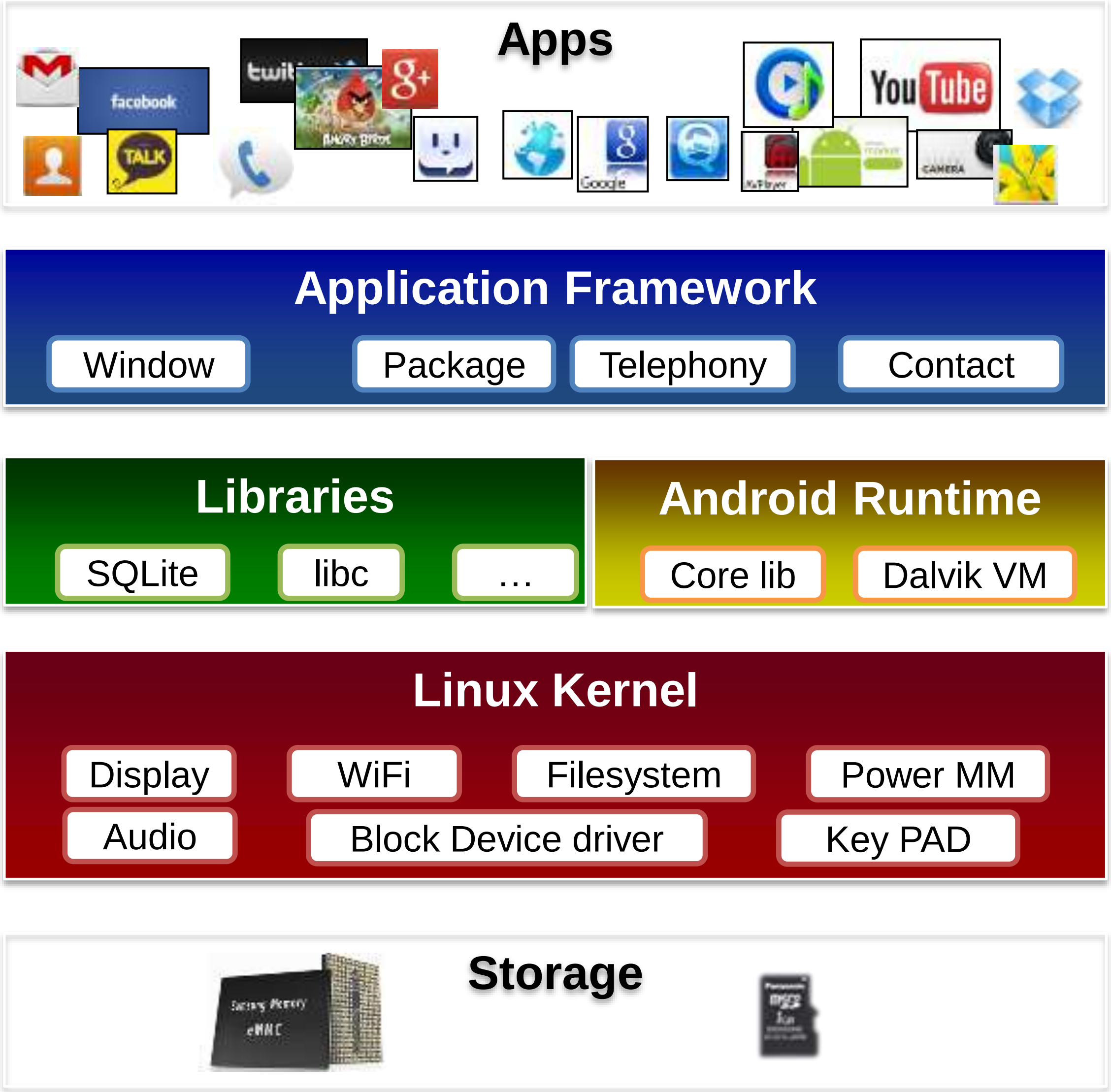


[KPCB Internet Trends 2013]

Motivation

Storage I/O is the performance bottleneck in Android.

Android Platform



Apps

Application Framework

Window

Package

Telephony

Contact

Libraries

SQLite

libc

…

Android Runtime

Core lib

Dalvik VM

Linux Kernel

Display

WiFi

Filesystem

Power MM

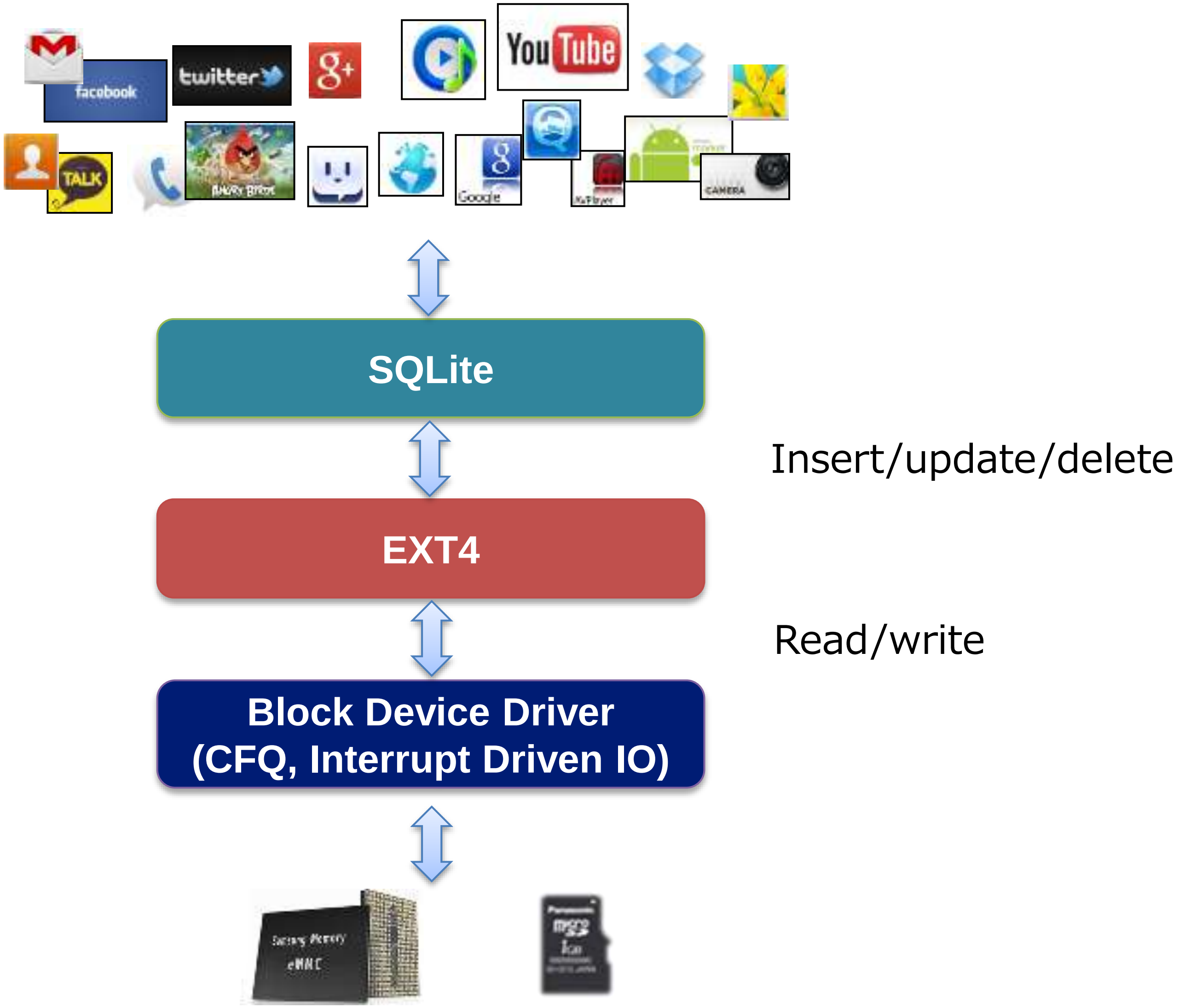
Audio

Block Device driver

Key PAD

Storage

I/O stack of Android Platform



SQLite

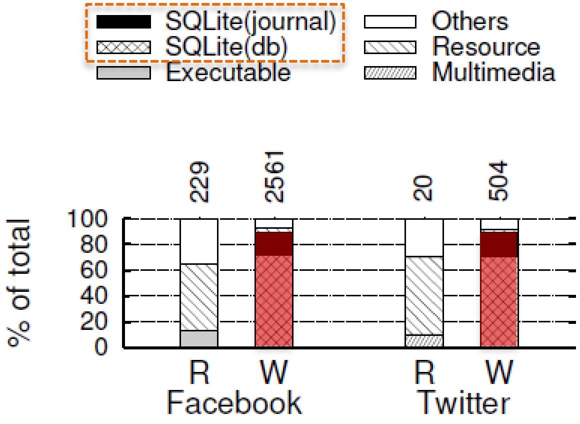
EXT4

Block Device Driver
(CFQ, Interrupt Driven IO)

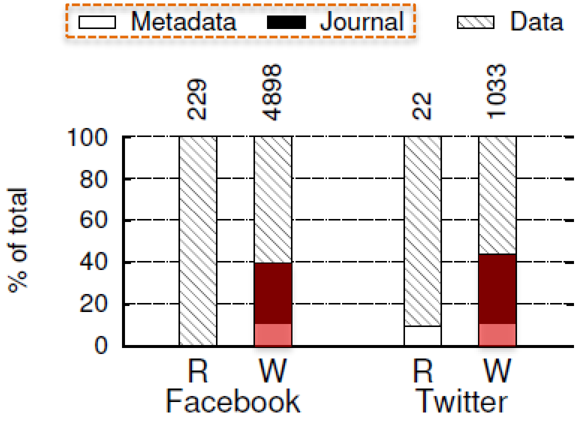
Insert/update/delete

Read/write

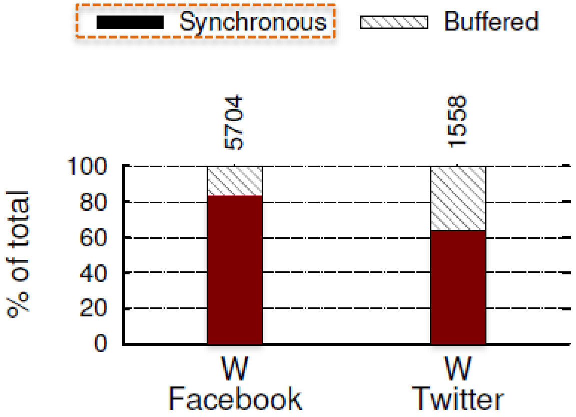
I/O characteristics of Android Apps (GS3, ICS)



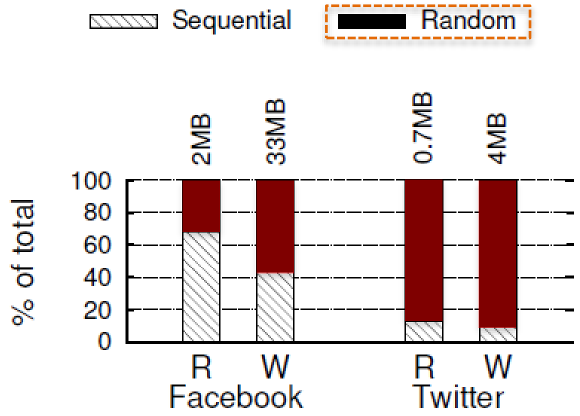
File Types



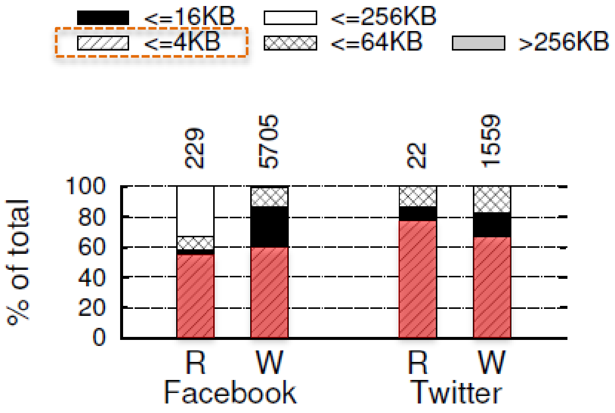
Block Types



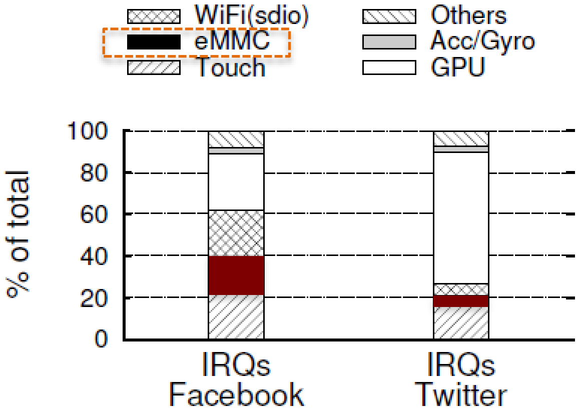
I/O Modes



Locality



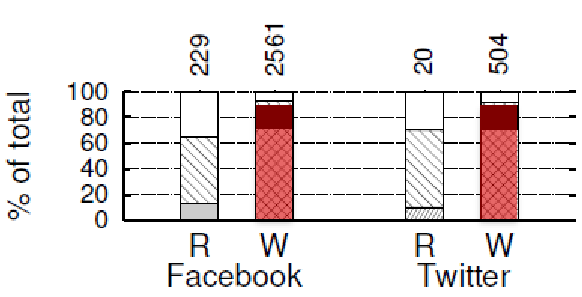
I/O Size



IRQs

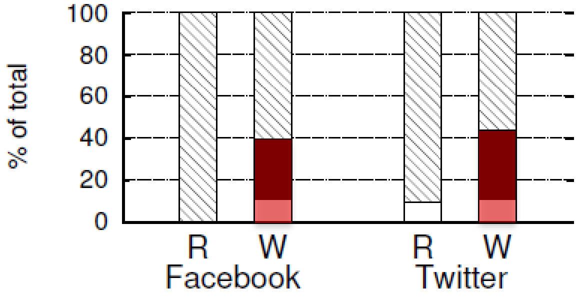
I/O characteristics of Android Apps (GS3, ICS)

SQLite > 90%



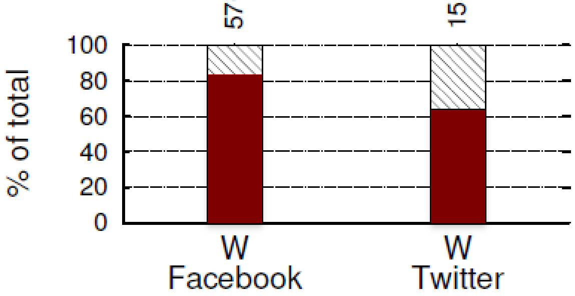
File Types

Metadata & Journal > 40%



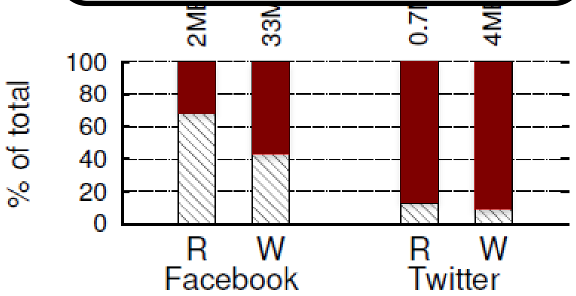
Block Types

Synchronous > 70%



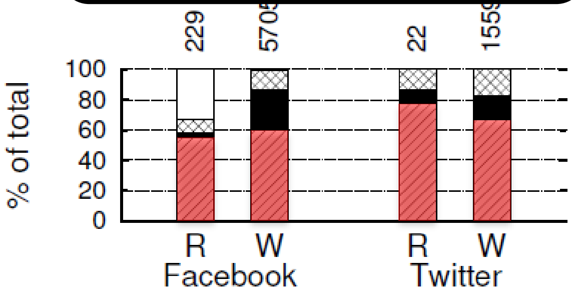
I/O Modes

Random > 80%



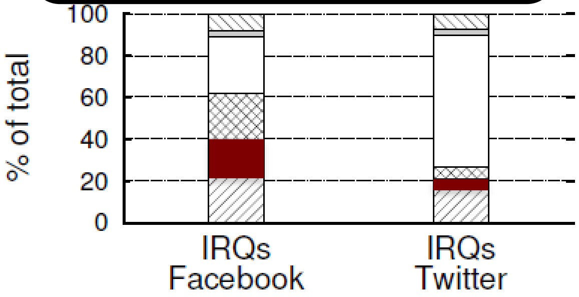
Locality

4KB I/O > 64%



I/O Size

IRQ for eMMC > 18%



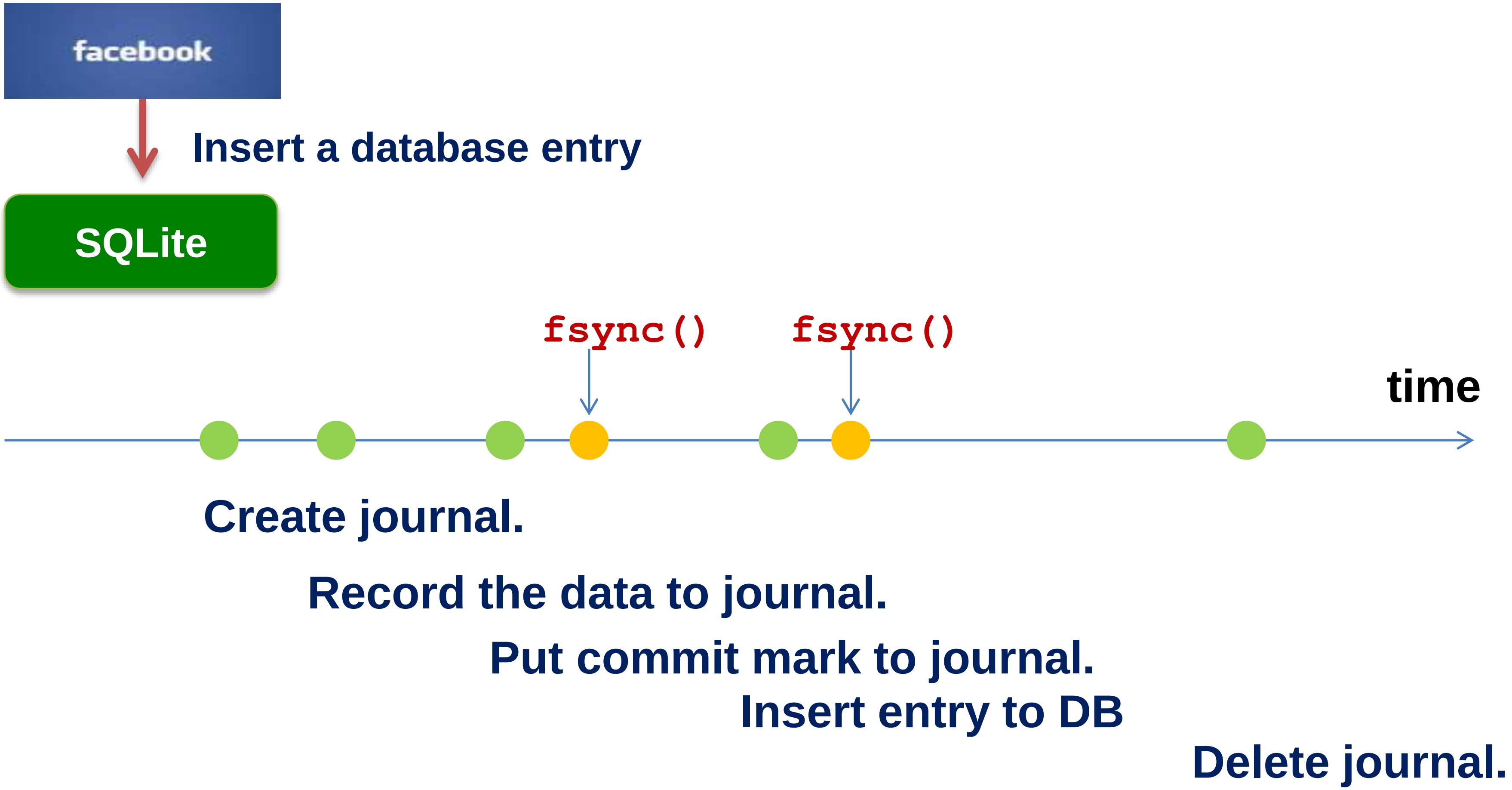
IRQs

Metadata & Journal > 40% !

SQLite > 90% !!!

Synchronous Write > 70% !!!

Journaling in SQLite (Delete Mode)



facebook

Insert a database entry

SQLite

fsync()

fsync()

time

Create journal.

Record the data to journal.

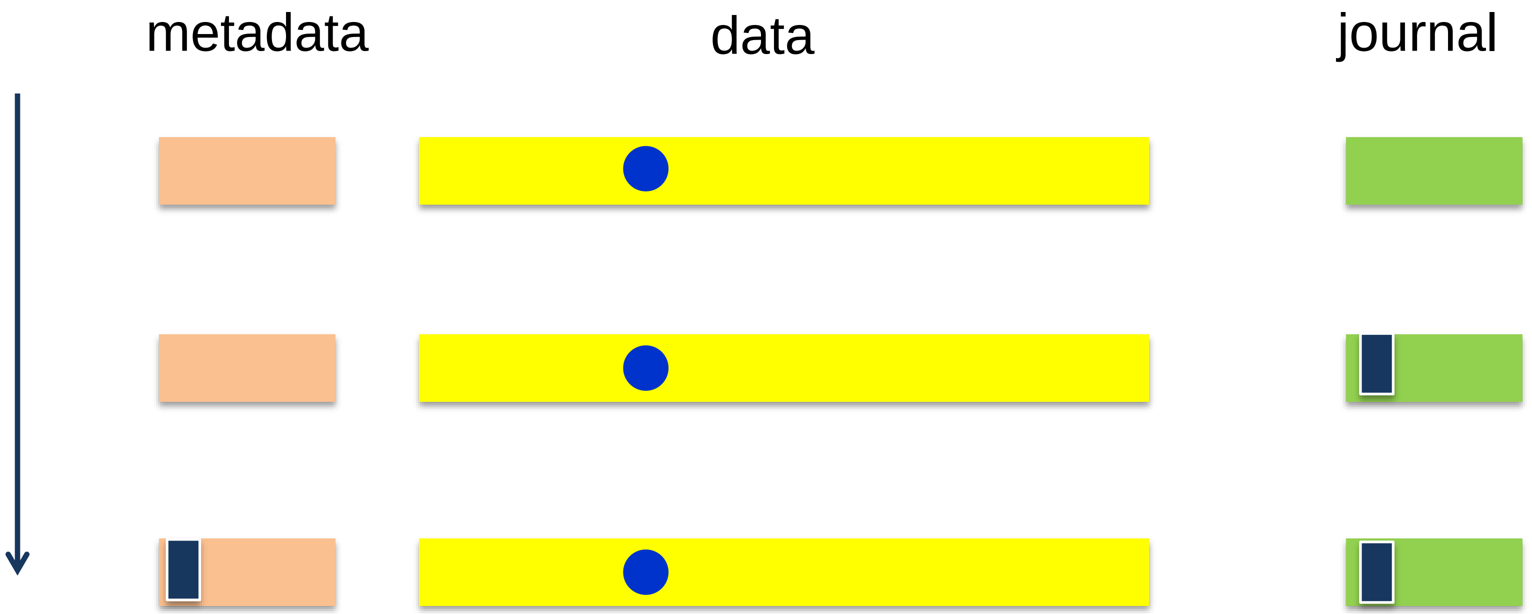
Put commit mark to journal.

Insert entry to DB

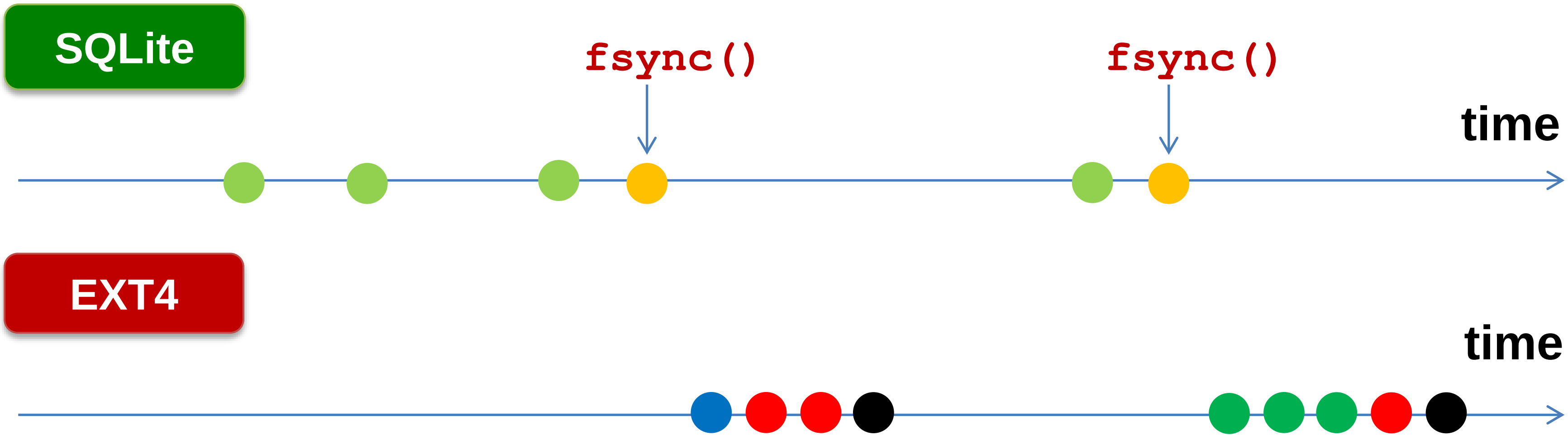
Delete journal.

Journaling in EXT4 (ordered mode)

```
write(fd, ● )
```



SQLite and EXT4



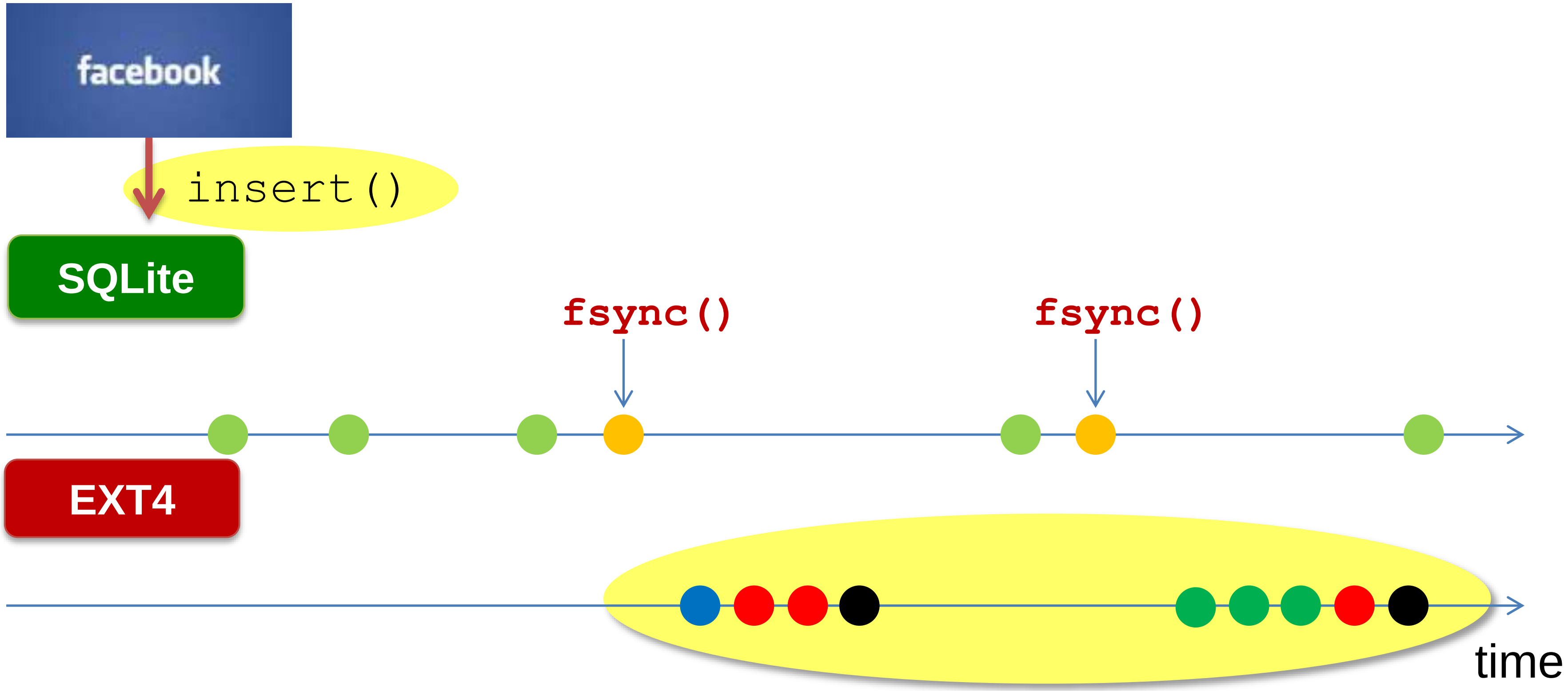
🔵 write SQLite journal to storage.

🟢 write SQLite DB to storage.

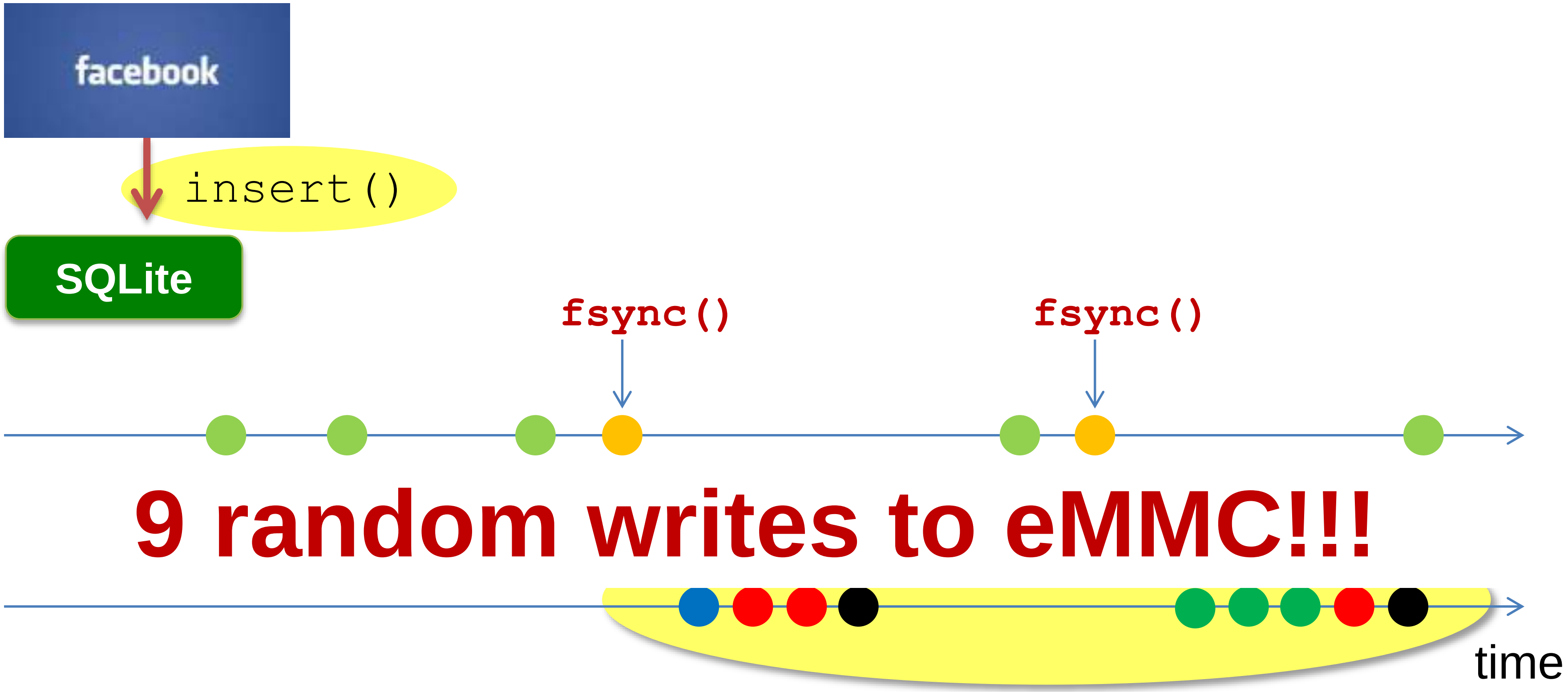
🔴 write EXT4 journal (descriptor, metadata) to storage.

⚫ write EXT4 journal (commit) to storage.

Summary



Summary



9 random writes to eMMC!!!

Summary

facebook

insert(

SQLite

nc()

9 ran

MMC!!!

time

Journaling of Journal

SQLite maintains DB journal. 🙂

+

EXT4 maintains filesystem journal. 🙂

=

EXT4 journals SQLite journal file.

Journaling of Journal

SQLite maintains DB journal. ☺

+

EXT4 journals SQLite journaling activity.

70% of the writes are purely for managerial purpose!

EXT4 journals SQLite journal file.

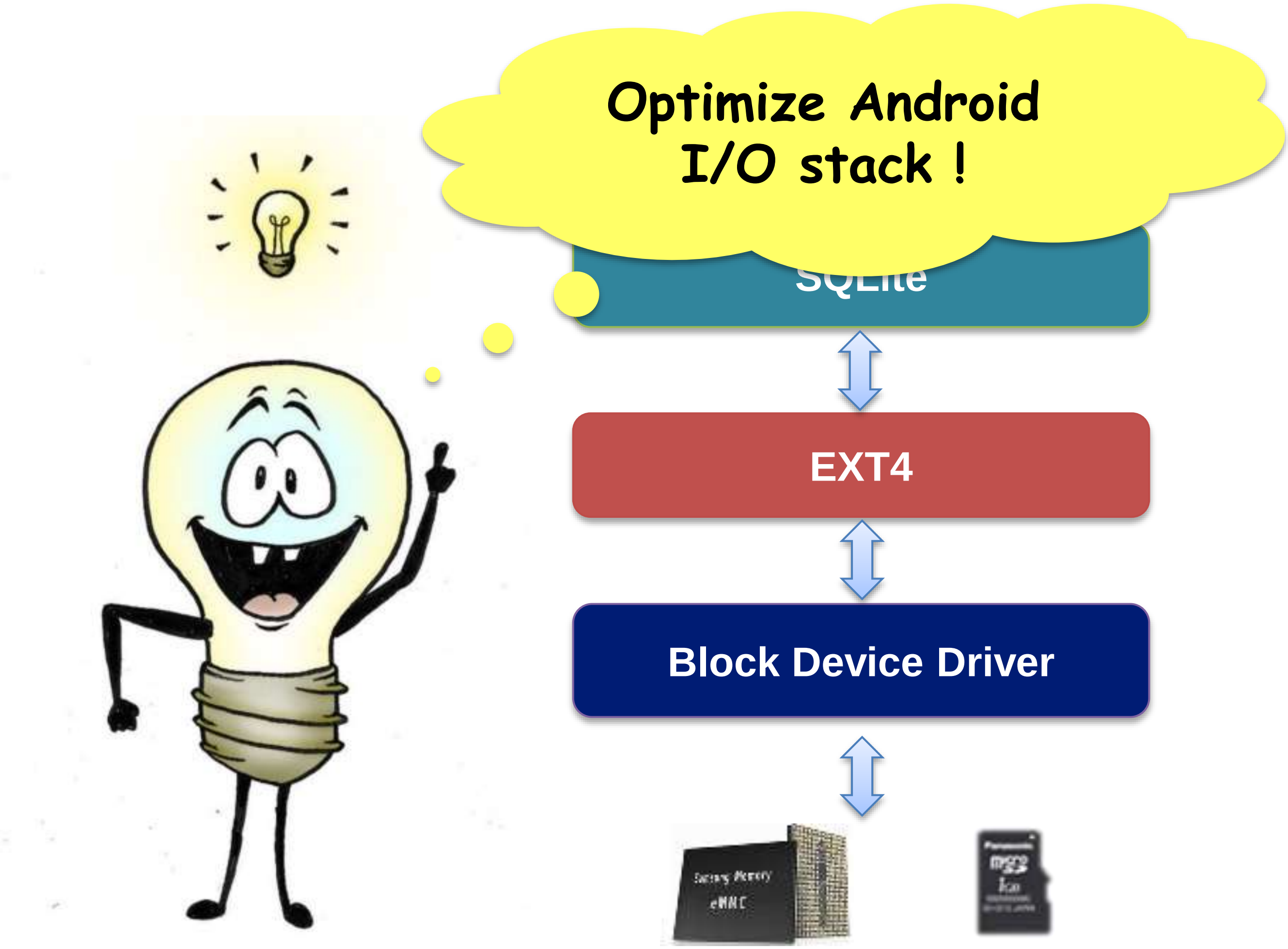
Journaling of Journal

SQLi

EXT4 jo ng activity.

70% of the wri agerial purpose!

EXT4 jo .



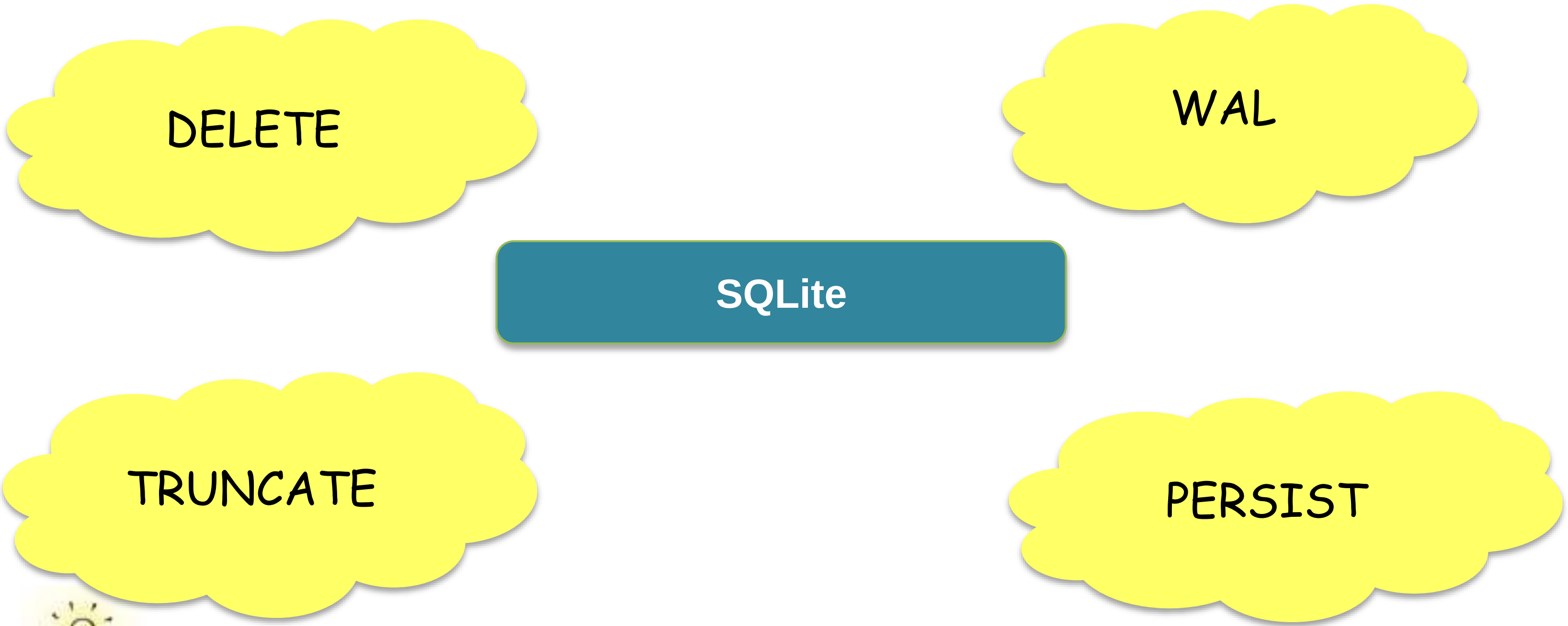
Optimize Android I/O stack !

SQLite

EXT4

Block Device Driver

SQLite Journaling mode



DELETE

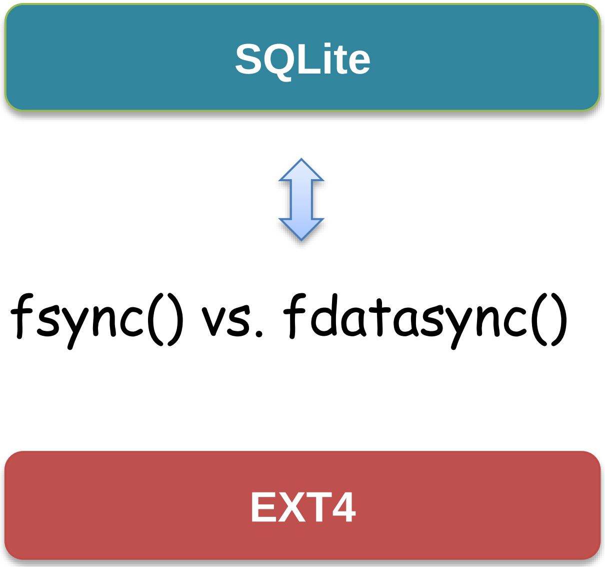
WAL

SQLite

TRUNCATE

PERSIST

Eliminating unnecessary metadata flushes



SQLite

fsync() vs. fdatasync()

EXT4

Alternative Filesystems

XFS

F2FS

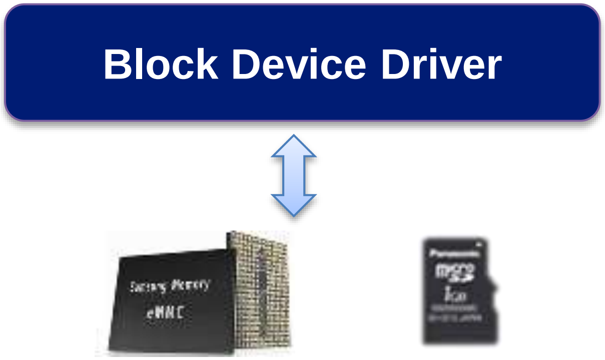
EXT4

NILFS2

BTRFS

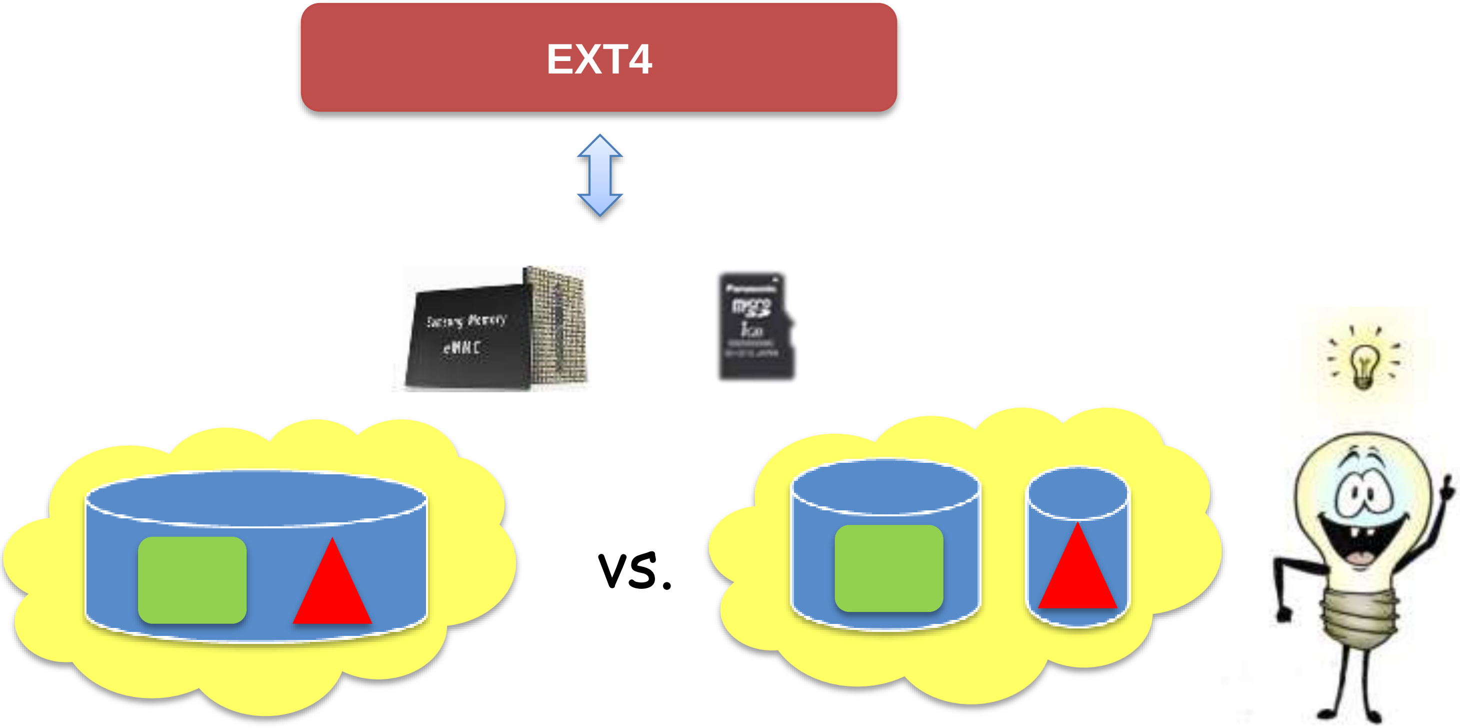
Interrupt vs. Polling

interrupt vs. polling



Block Device Driver

External Journaling

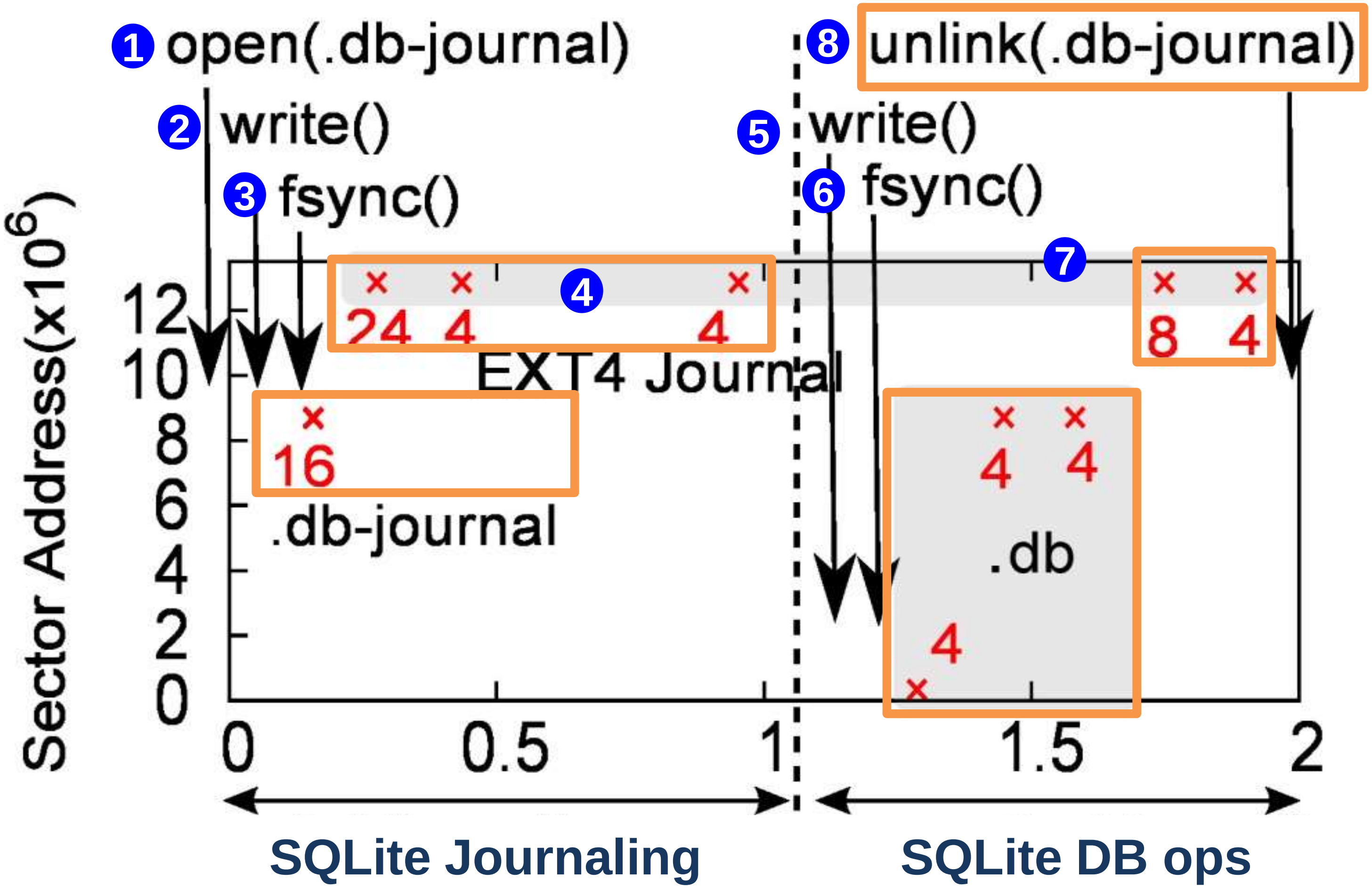


EXT4

vs.

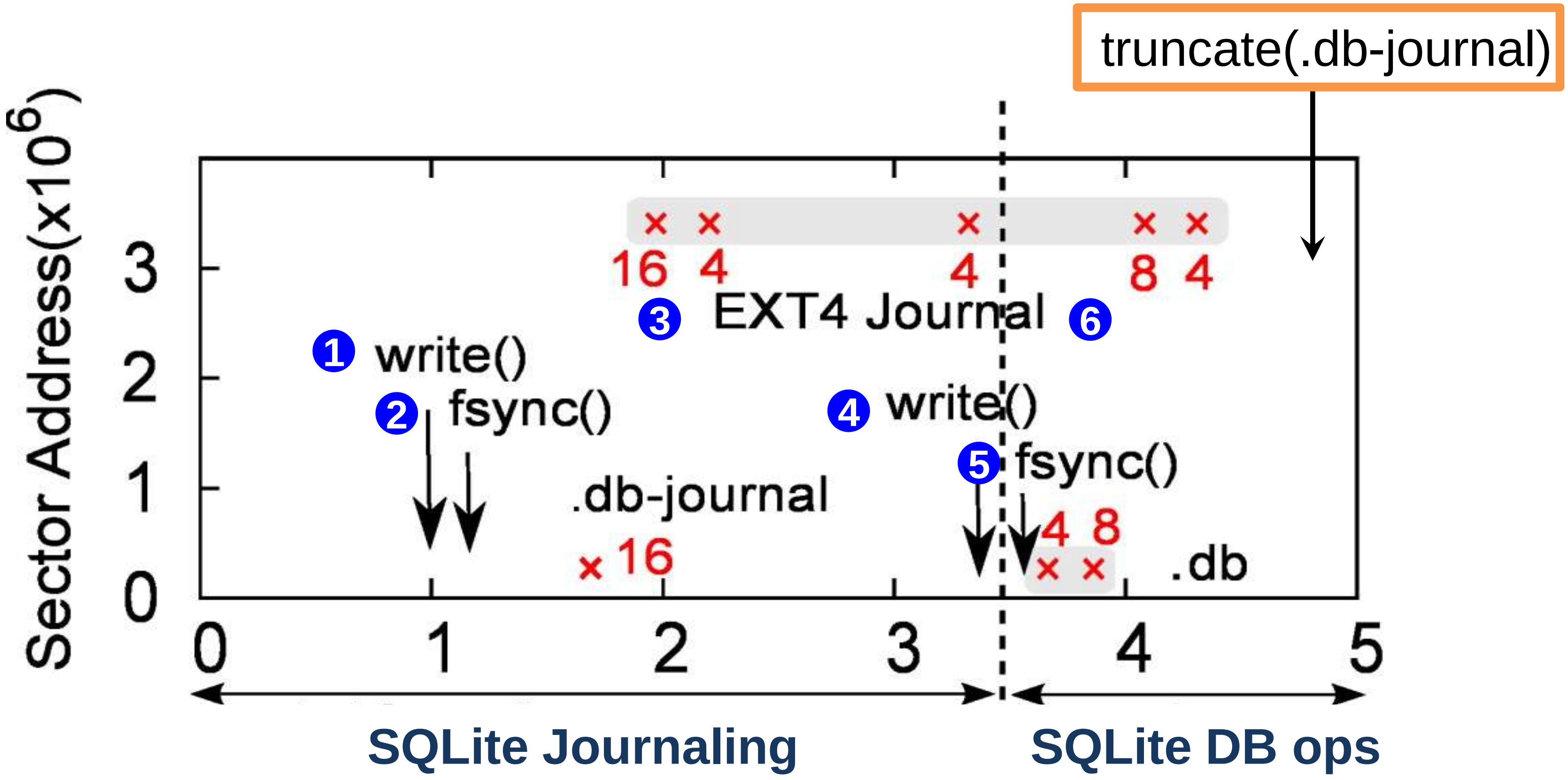
SQLite Journaling Modes

Delete (GS3, ICS)



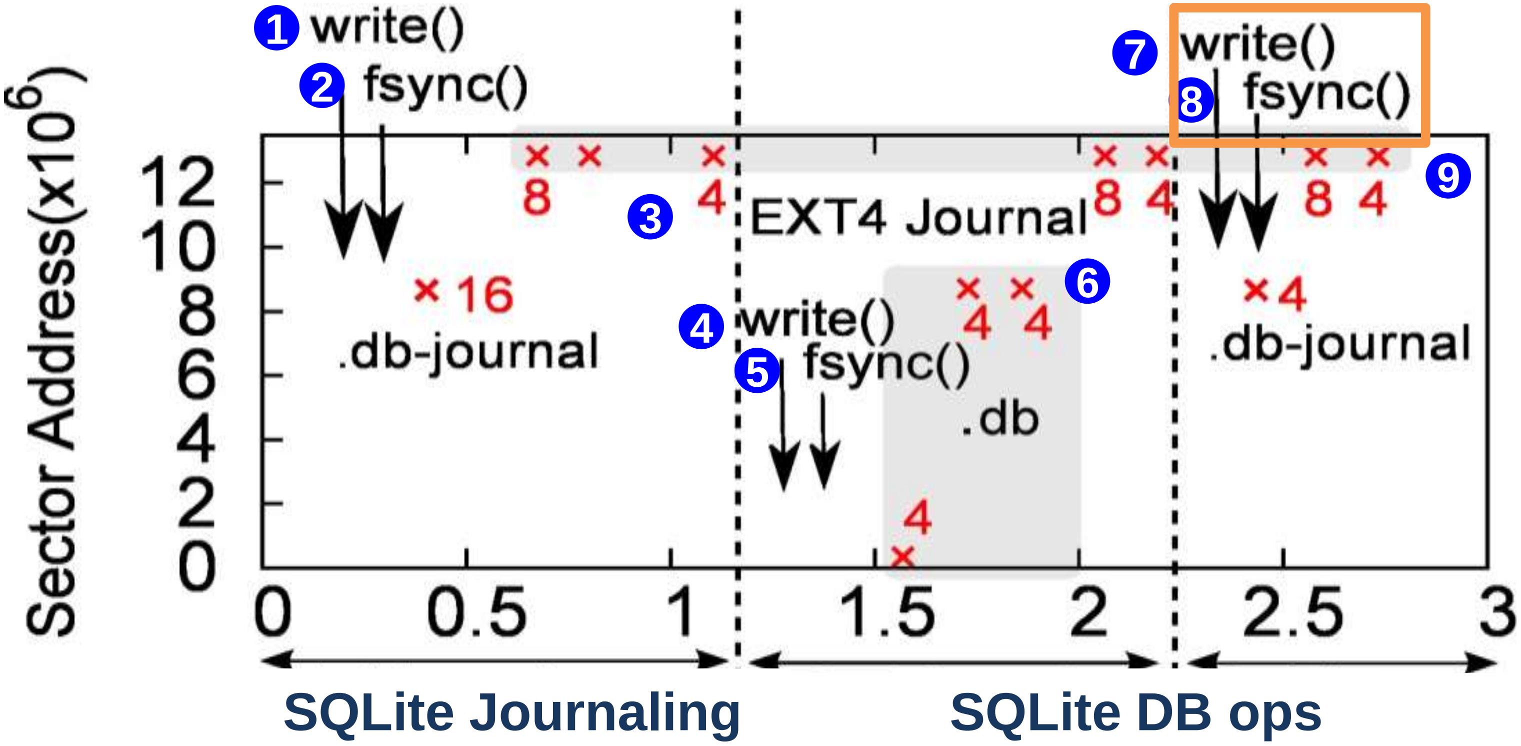
2 `fsync()` and 9 `writes` for one insert() !

Truncate (GS3, ICS)



2 `fsync()` and 8 `writes`.

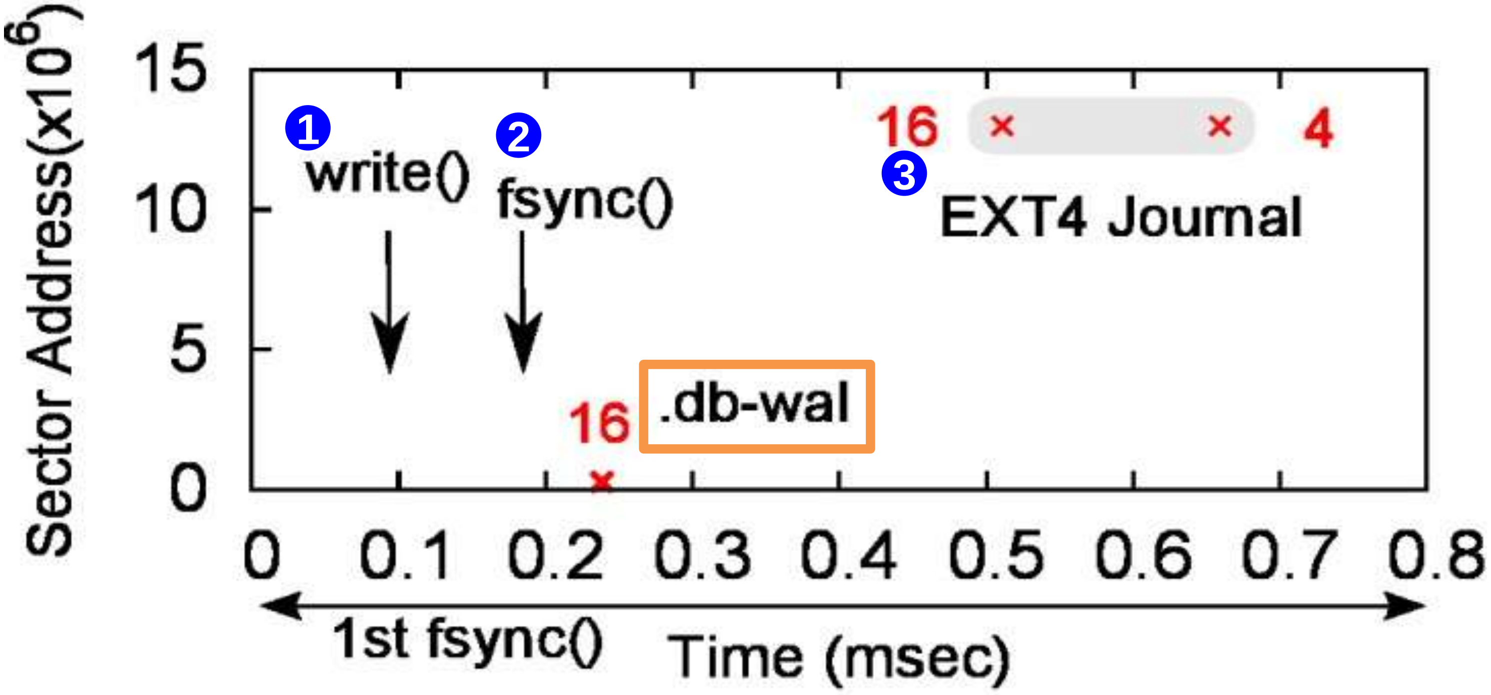
Persist (GS3, ICS)



3 `fsync()` and 12 `writes`.

The worst mode!

WAL Mode (GS3, ICS)



Only **1 `fsync()`** and **3 `writes`**.

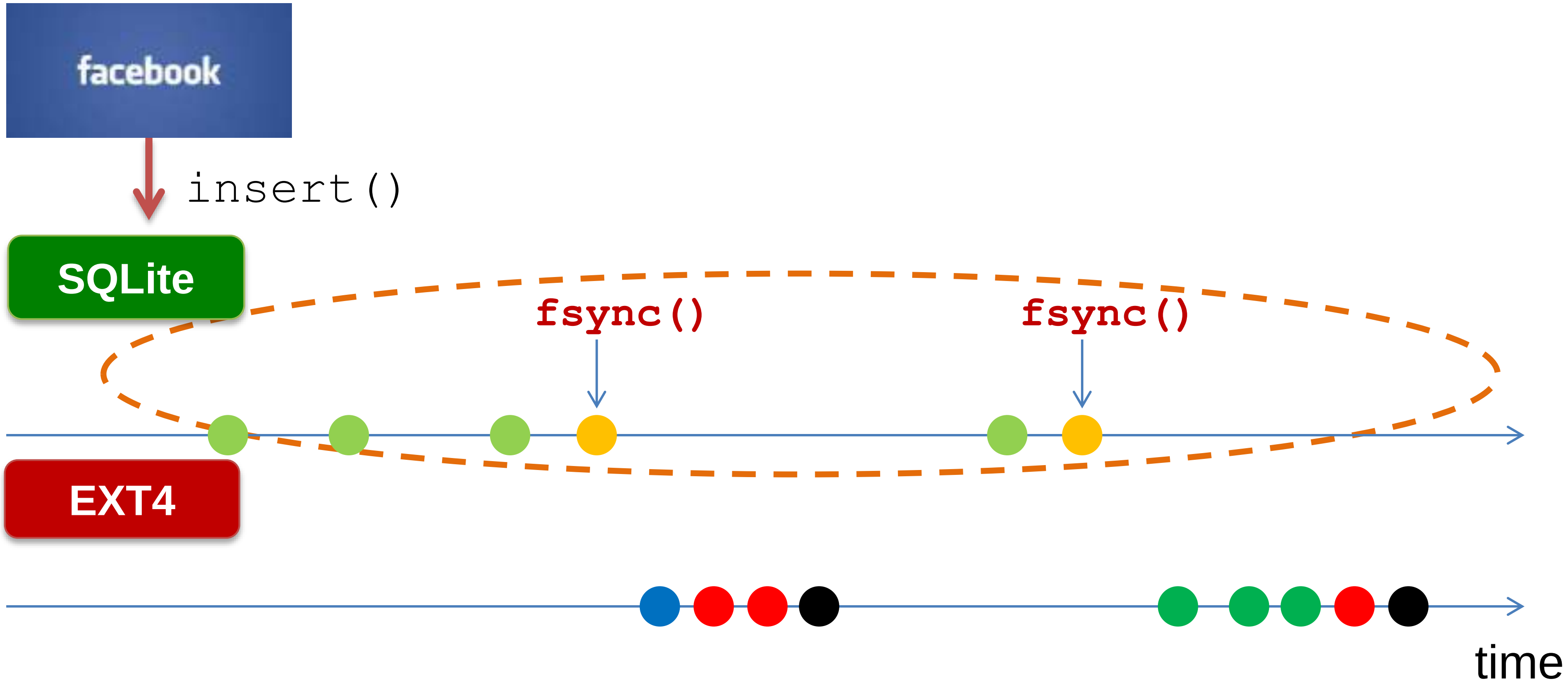
The best mode!

SQLite Journaling Mode

Summary

SQLite Journaling Mode	DELETE	TRUNCATE	PERSIST	WAL
Number of fsync() calls	2	2	3	1
Number of IOs	9	8	12	3
EXT4 Journal size (metadata)	24 KB	16 KB	8 KB	16 KB
Total IO Volume	72 KB	64 KB	72 KB	36 KB

Filesystems

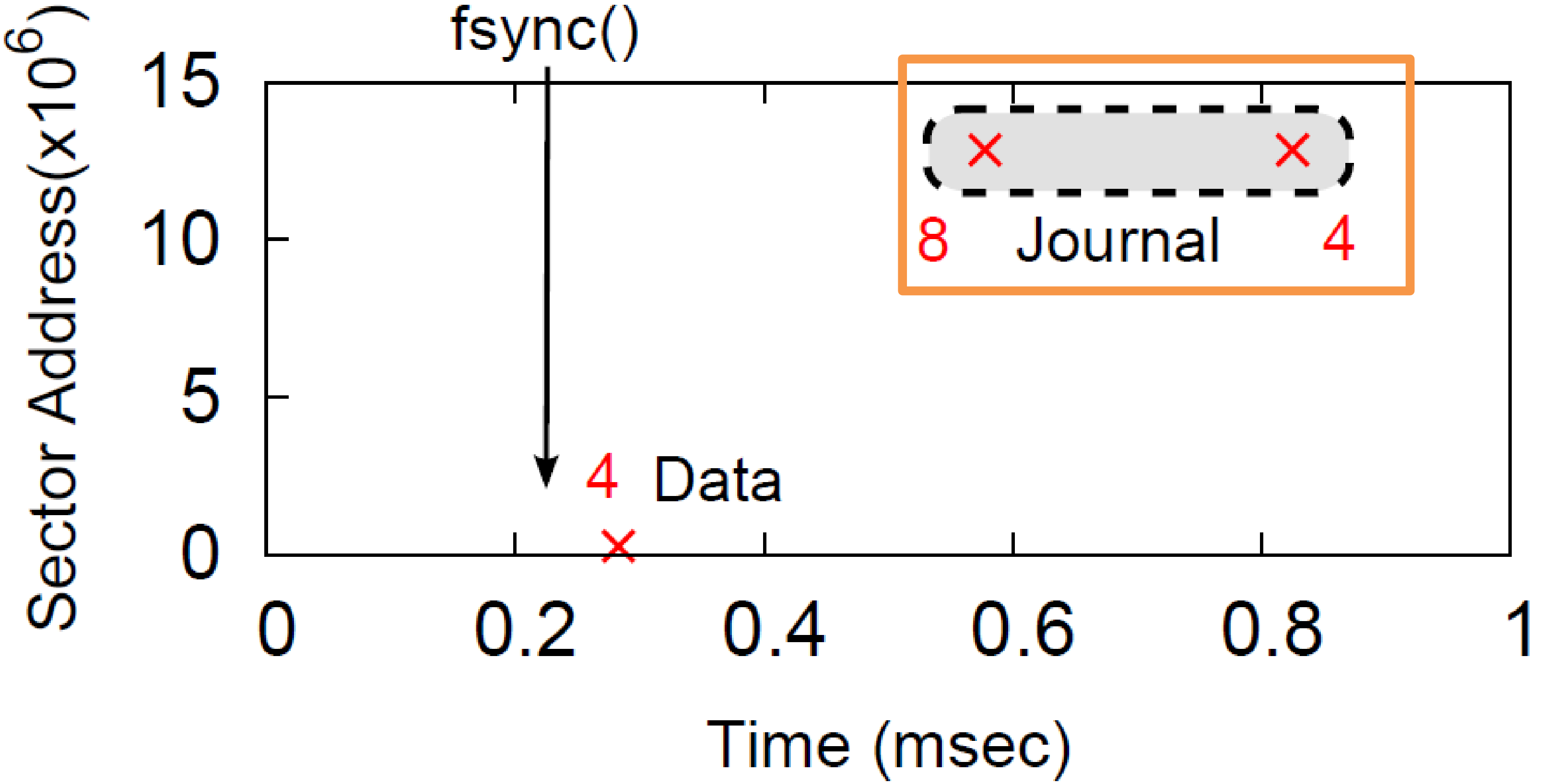


“`write()` followed by `fsync()`”

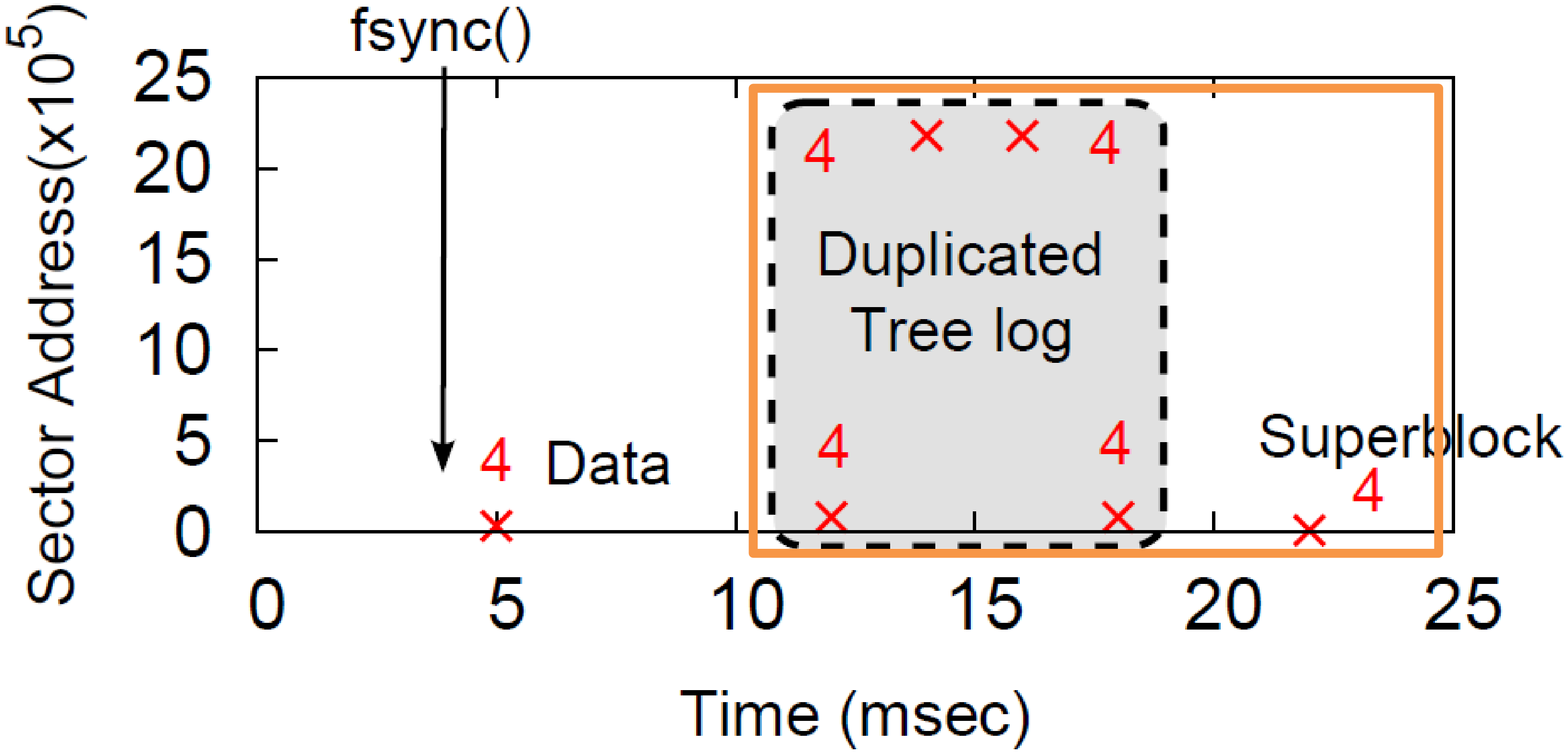
is the essence of the Android I/O.

EXT4

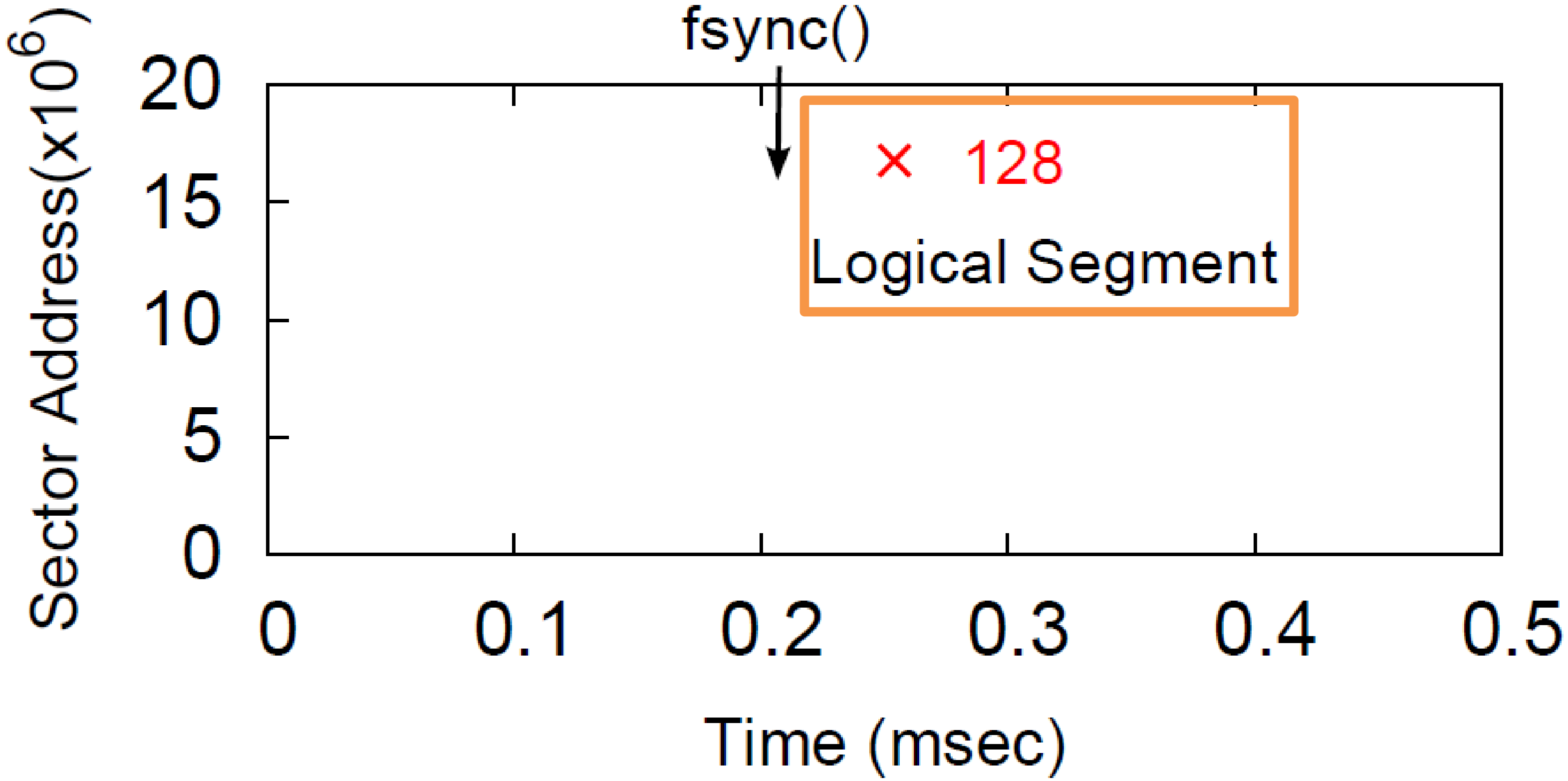
4 KB `write()` followed by `fsync()`



BTRFS

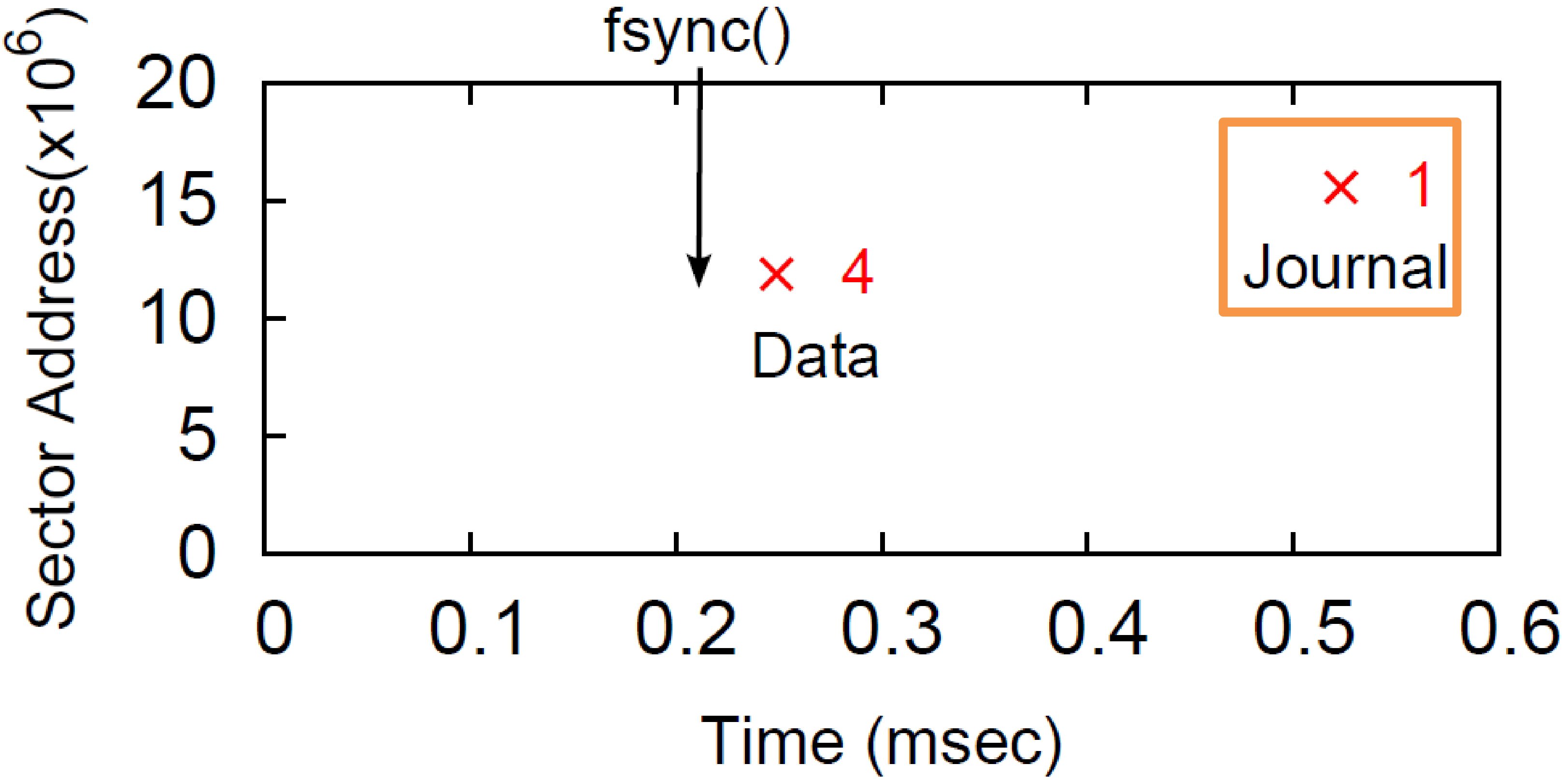

fsync()
4
Data
4
4
Duplicated
Tree log
4
4
Superblock
4
Sector Address(x10^5)
25
20
15
10
5
0
0
5
10
15
20
25
Time (msec)

NILFS2

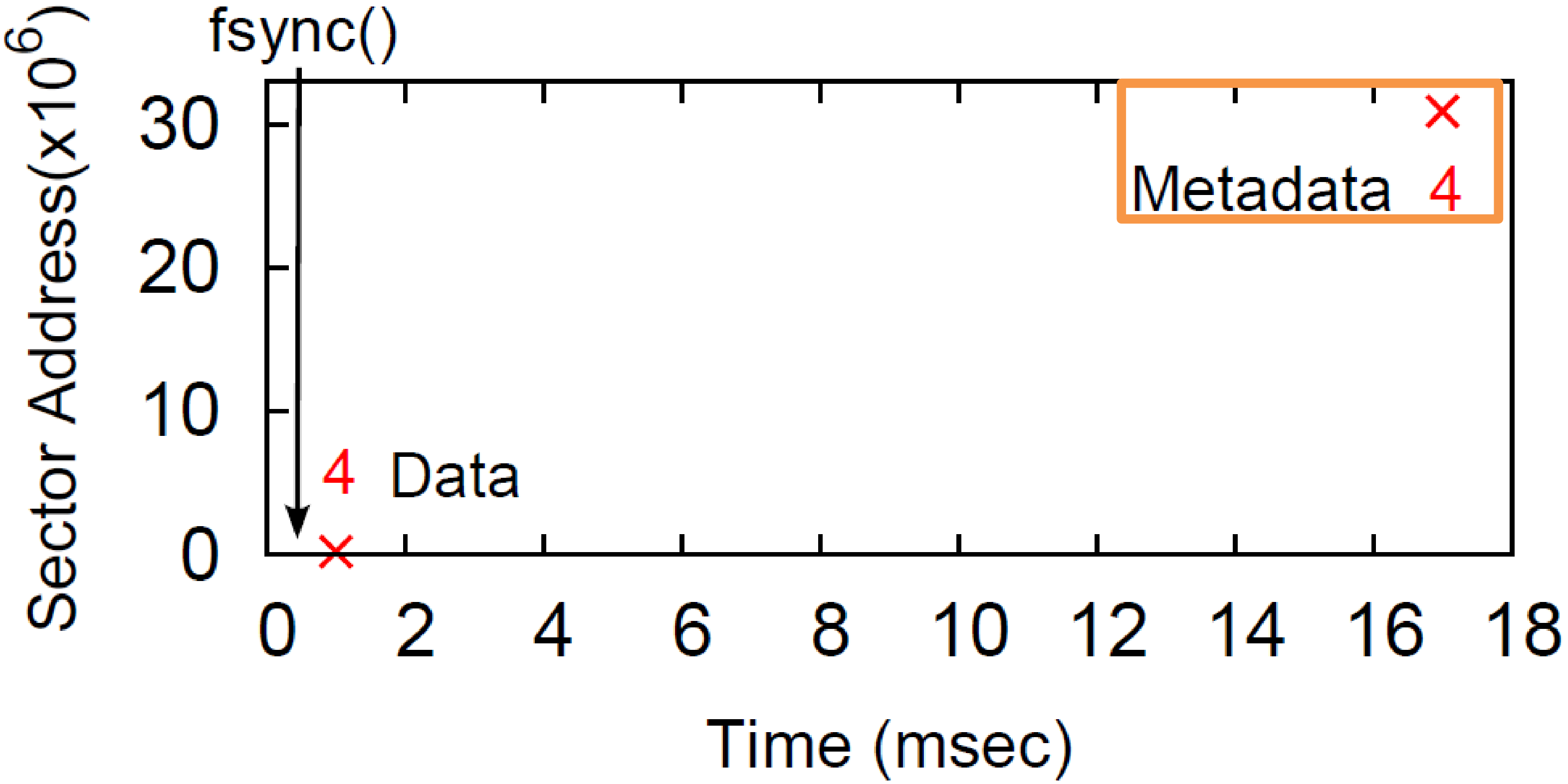


fsync()
× 128
Logical Segment
Sector Address(x10^6)
20
15
10
5
0
0
0.1
0.2
0.3
0.4
0.5
Time (msec)

XFS

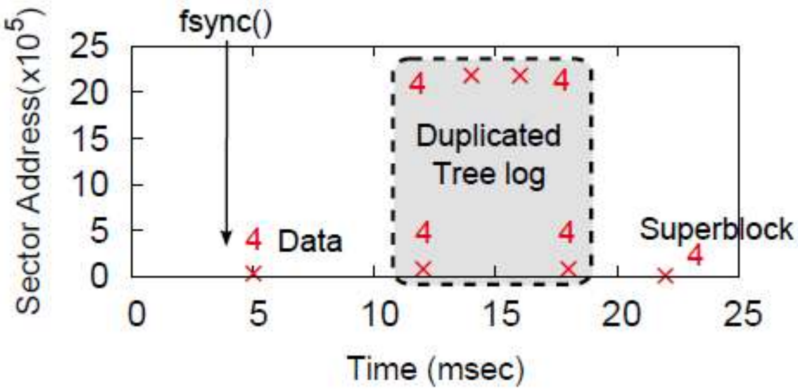

fsync()
× 4
Data
× 1
Journal
Sector Address(x10^6)
0
5
10
15
20
0
0.1
0.2
0.3
0.4
0.5
0.6
Time (msec)

F2FS (Flash Friendly Filesystem)

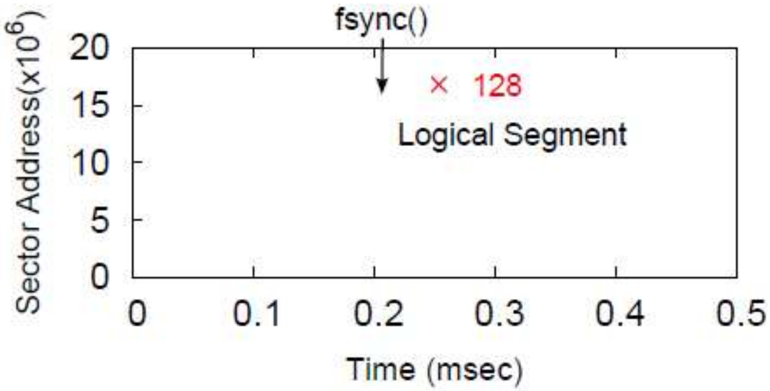


write() followed by fsync()

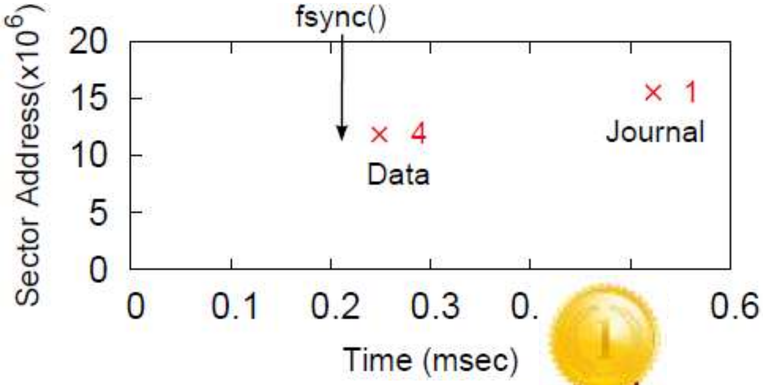
Summary



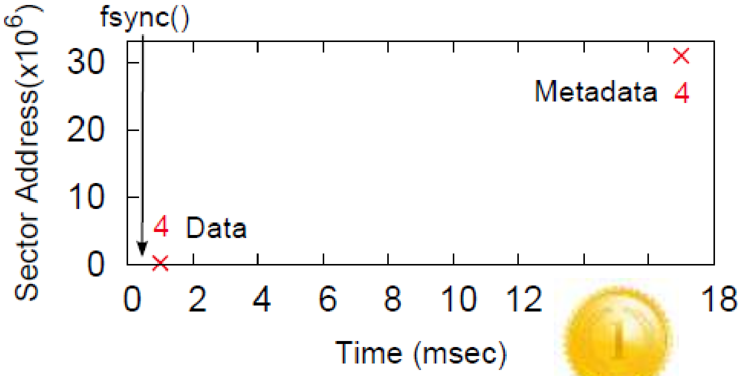
BTRFS



NILFS2



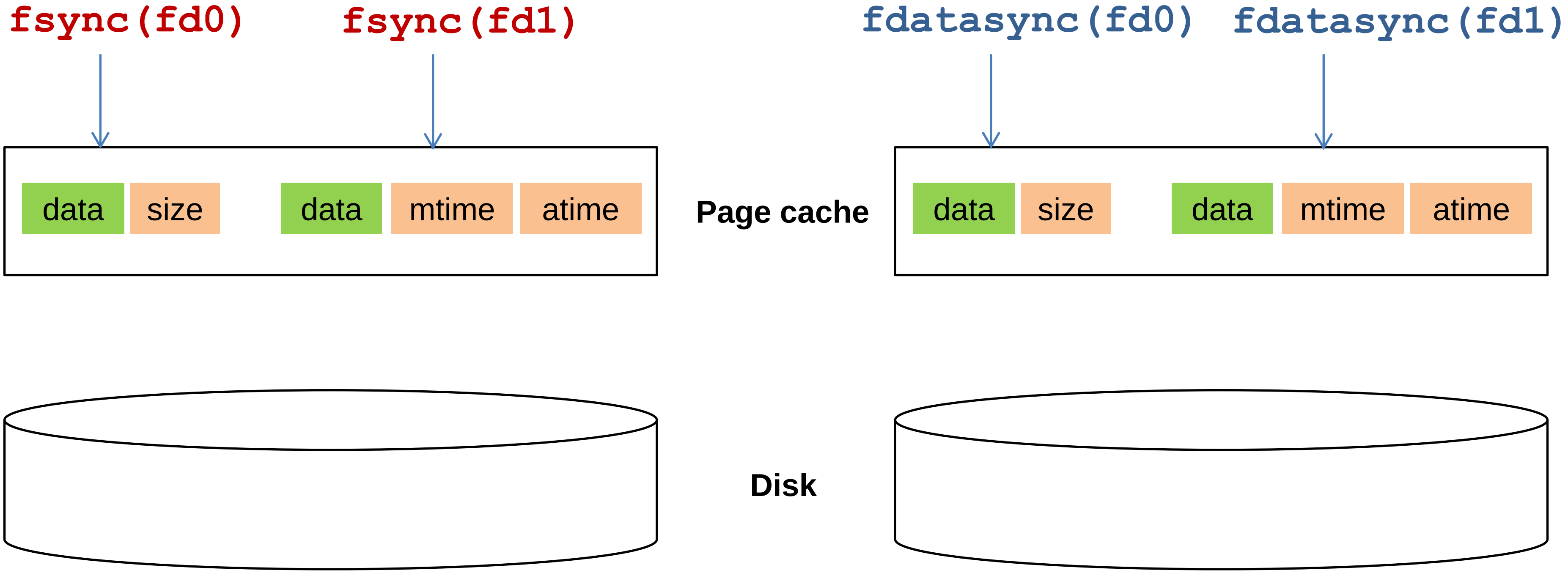
XFS



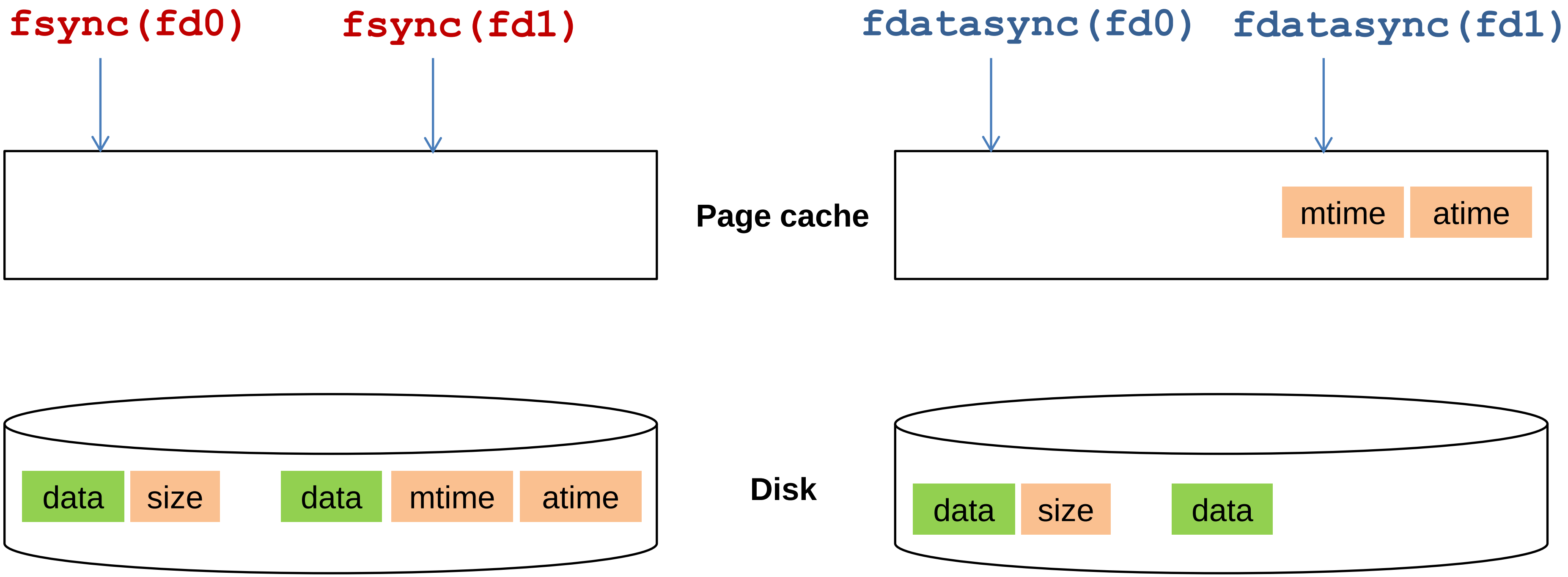
F2FS

`fsync()` vs. `fdatasync()`

Eliminating Unnecessary Metadata Flushes

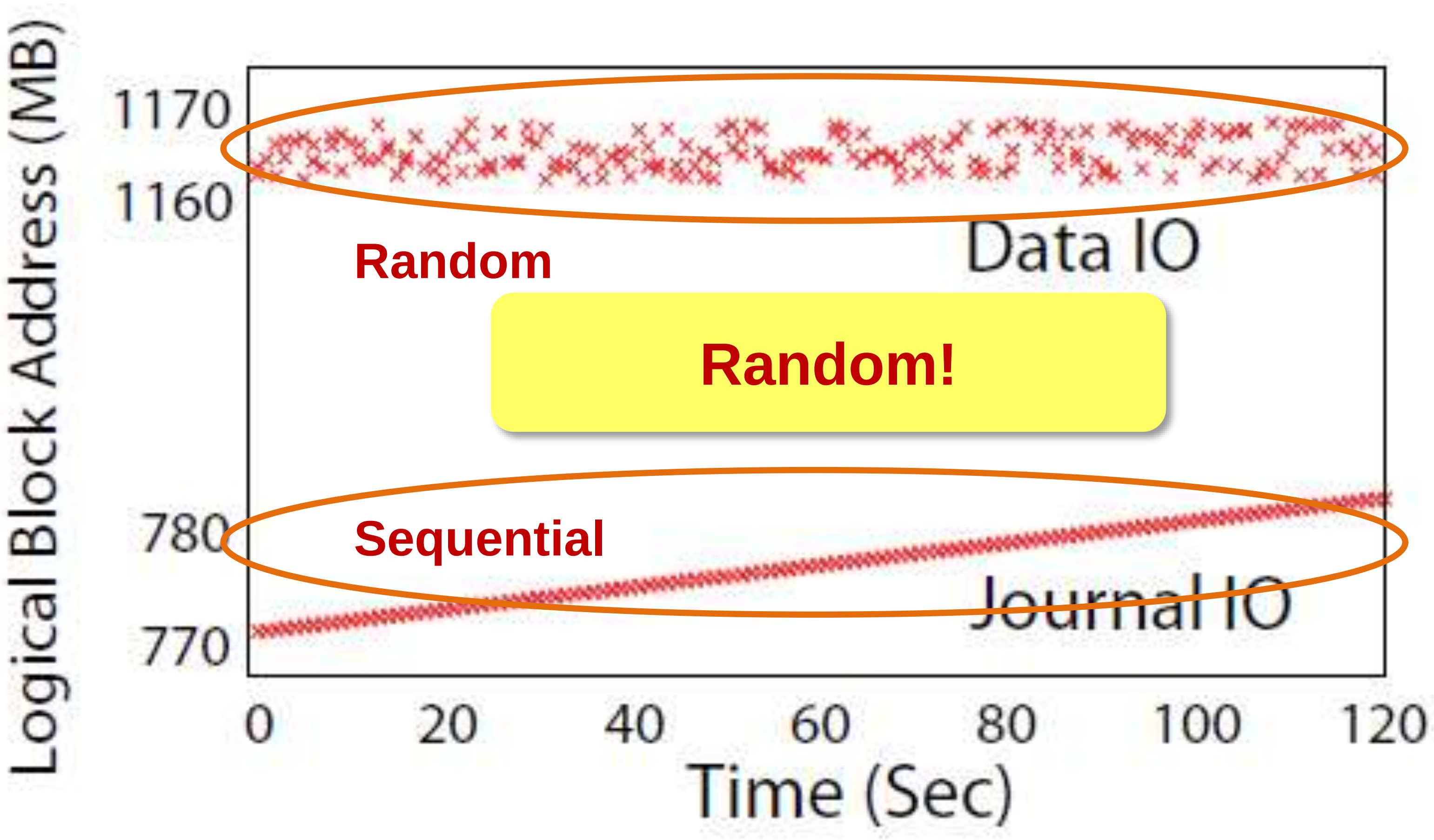


Eliminating Unnecessary Metadata Flushes



External Journaling

4K random write() followed by fsync()



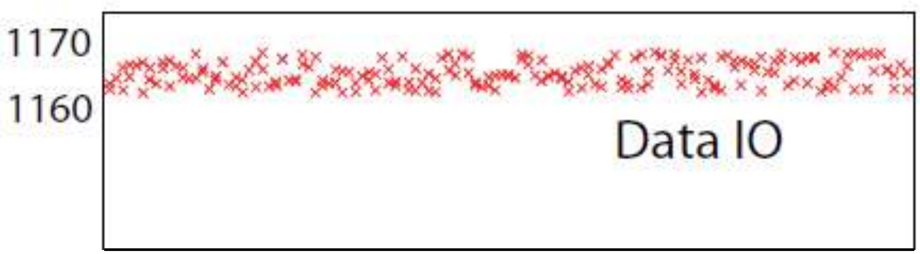
Random

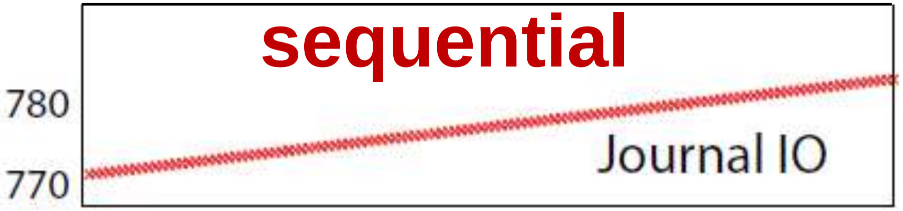
Random!

Sequential

External journaling

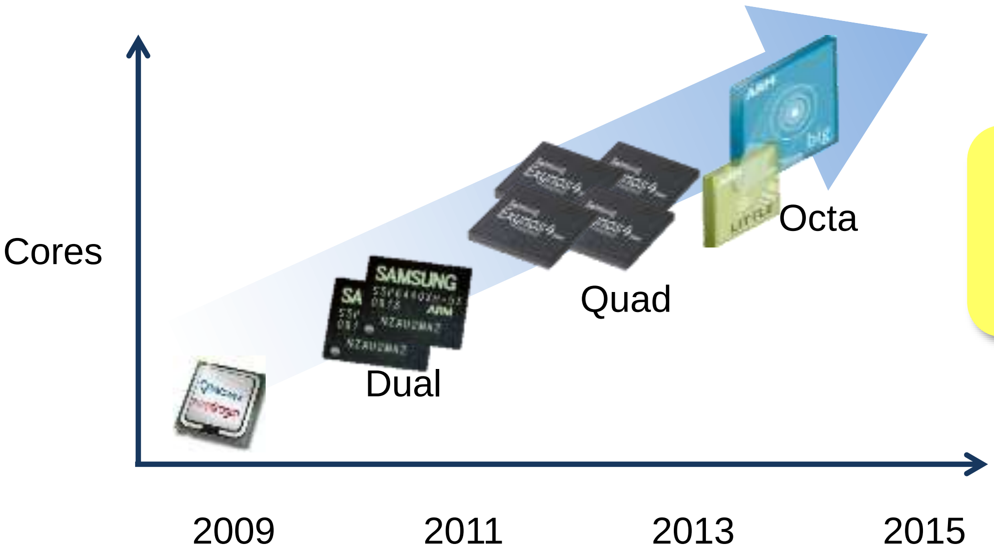
Journal on separate partition
→ FTL can exploit the locality of I/O!




sequential

Interrupt driven I/O vs. Polling based I/O

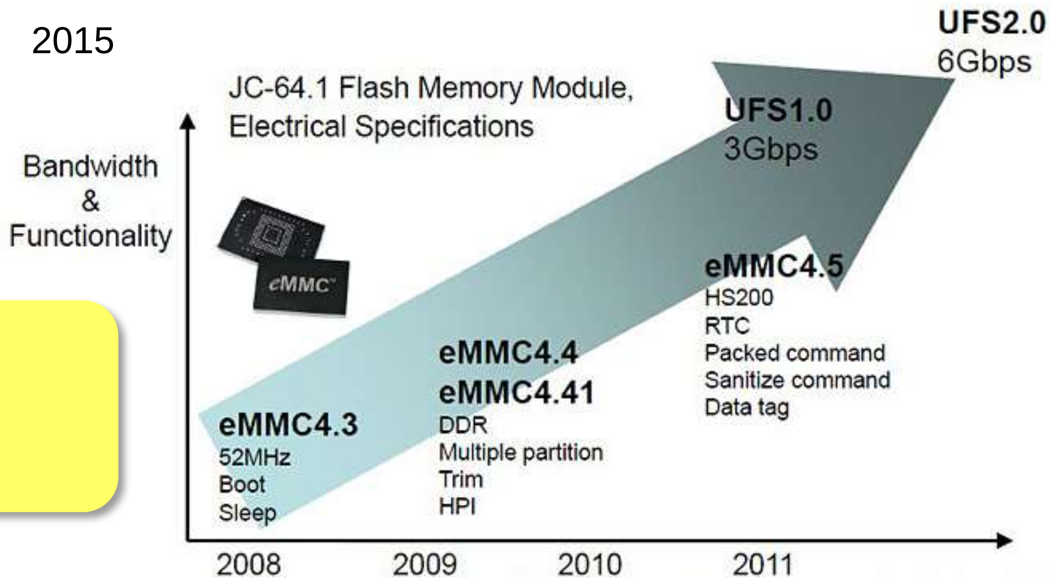
Hardware trend



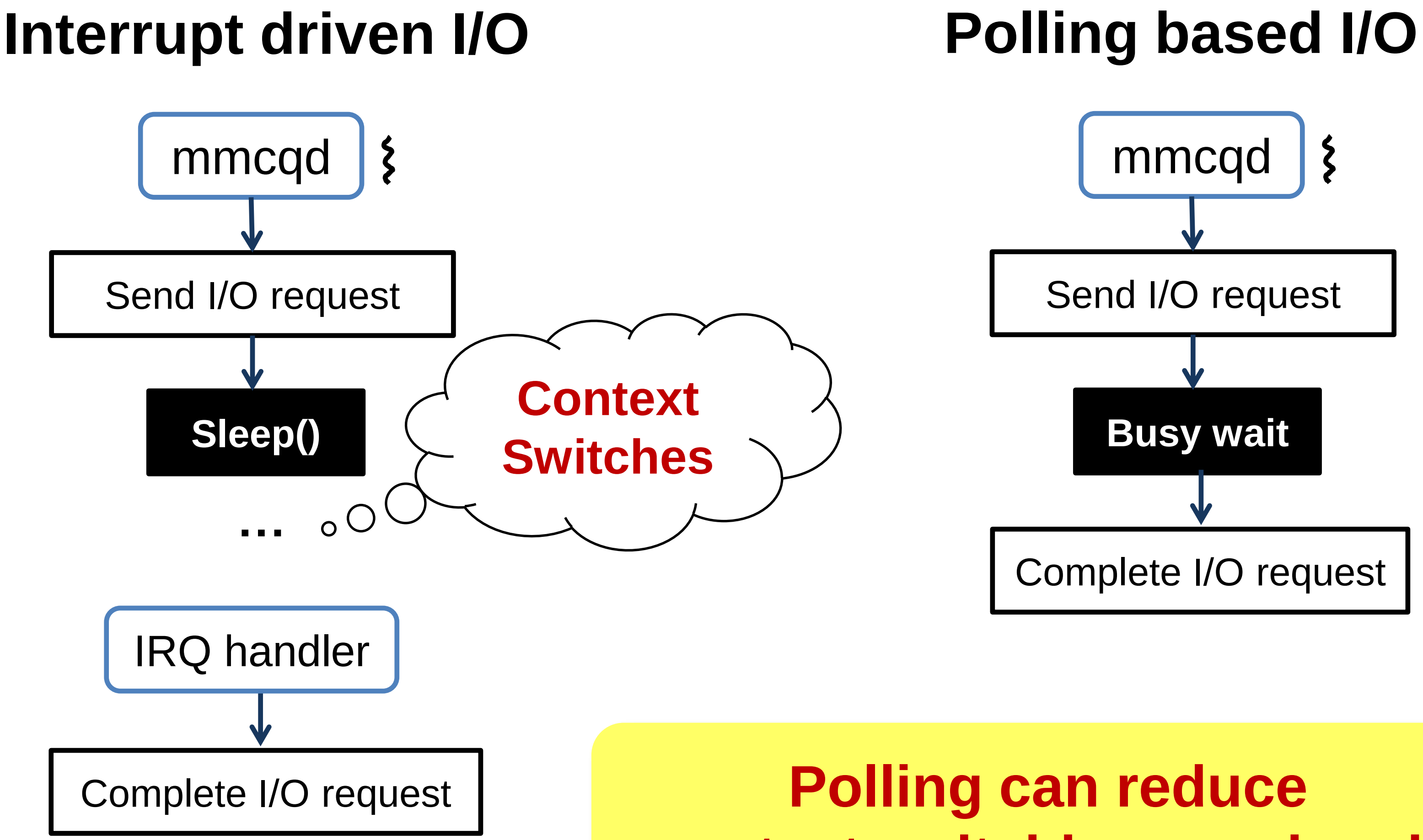
Multi-core on smartphones

The number of CPU cores ↑

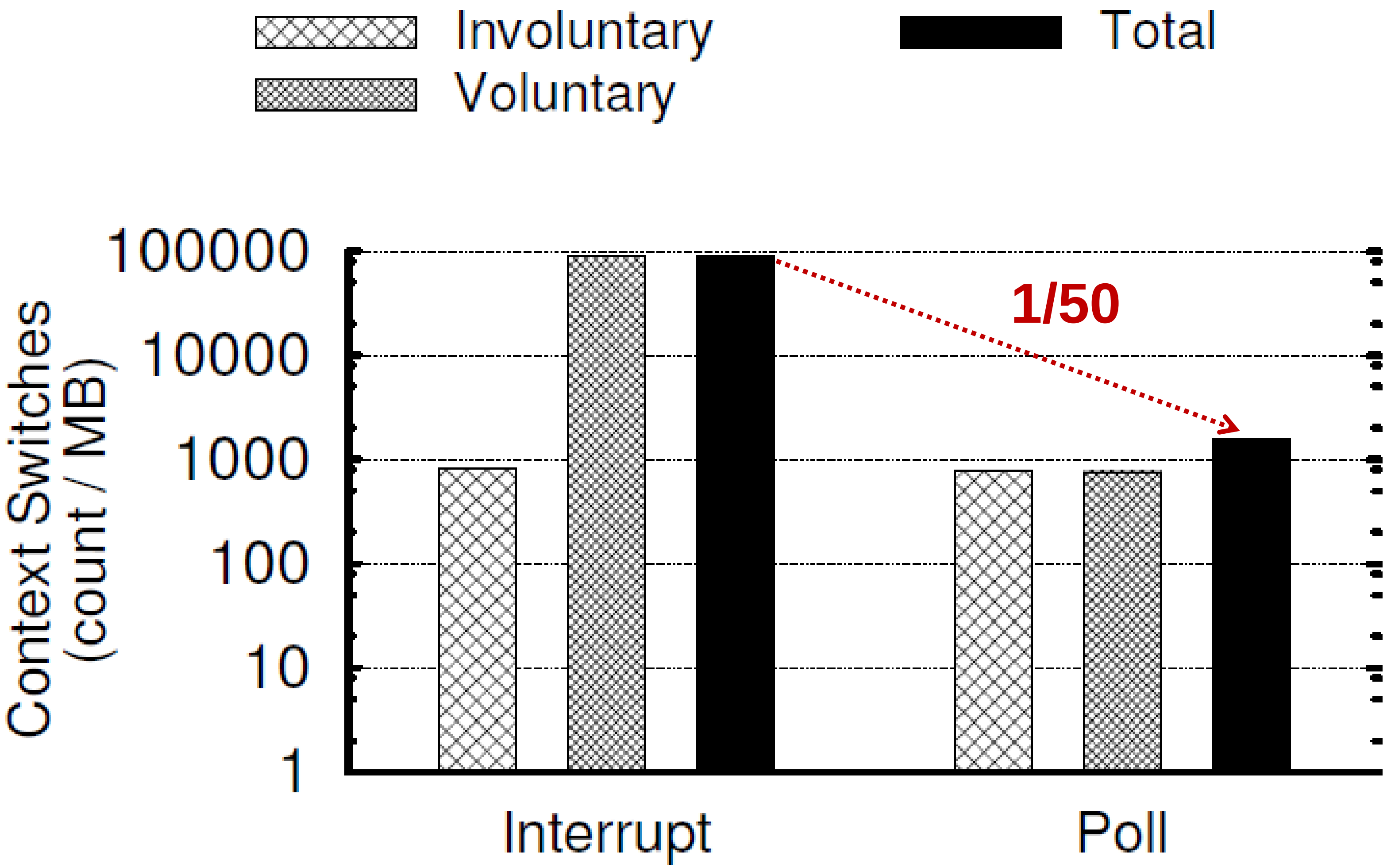
I/O latency of eMMC ↓



Performance of mobile flash storage



Polling can reduce context switching overhead!


Involuntary
Voluntary
Total
Context Switches (count / MB)
100000
10000
1000
100
10
1
Interrupt
Poll

1/50

Experiment

Implementation

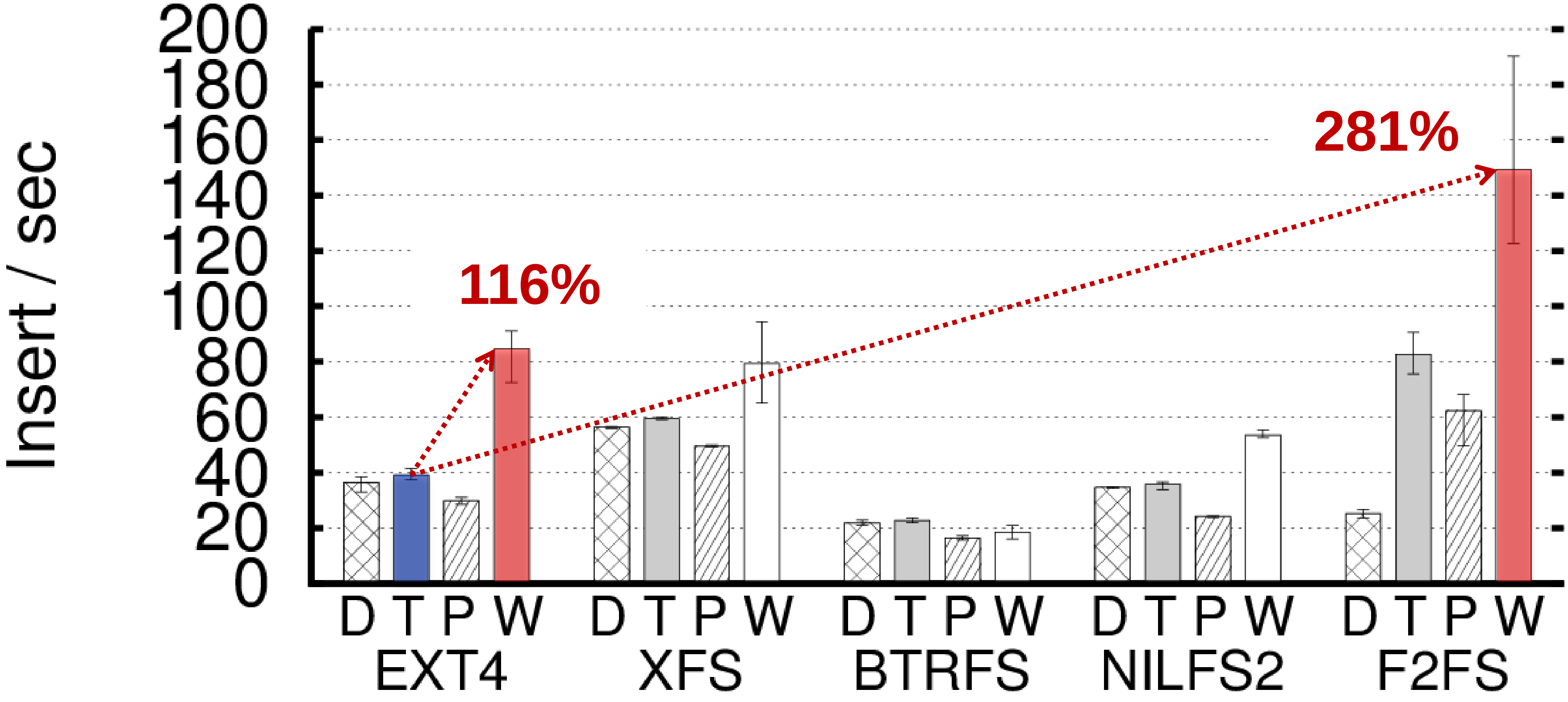
Galaxy S3(ICS 4.0.4, Linux 3.0.15)

Component	Specification
CPU	Exynos 4412 1.4 GHz Quad-core
RAM	2 GB
Internal Storage	32 GB eMMC
External Storage	16 GB Transcend u-SD Card

SQLite performance: journaling modes

SQLite Insert

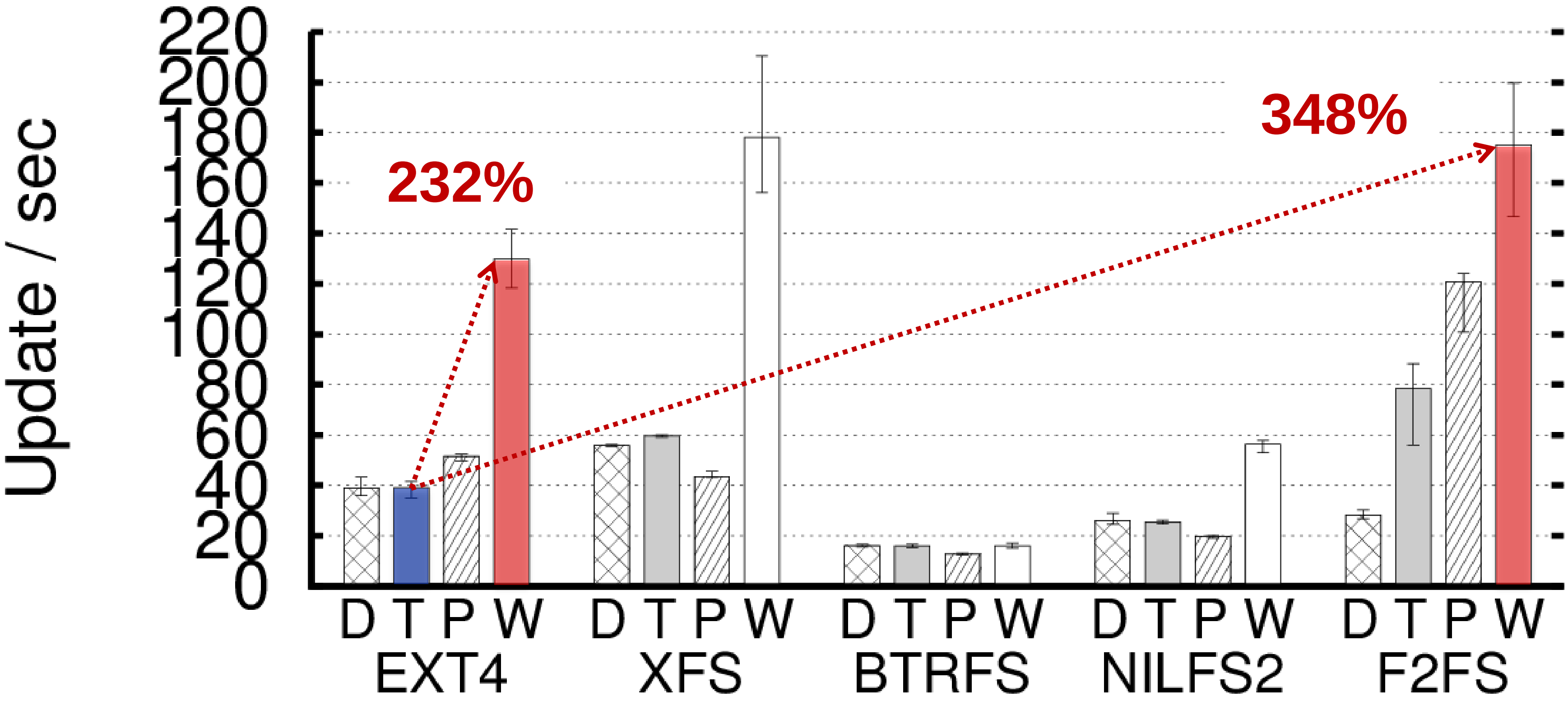
- TRUNCATE(default) → WAL : 116% up
- TRUNCATE, EXT4(default) → WAL,F2FS: 281% up



SQLite performance: journaling modes

SQLite Update

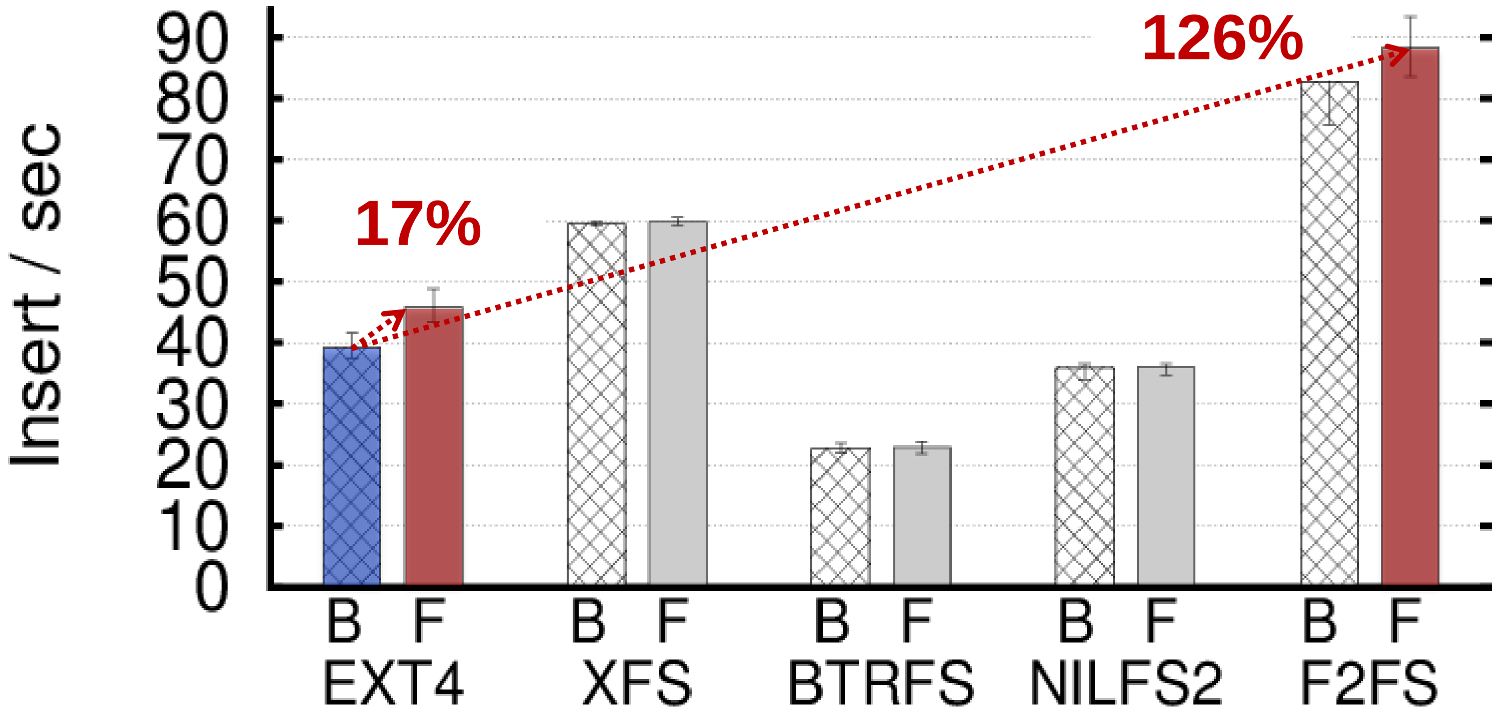
- TRUNCATE(default) → WAL : 232% up
- TRUNCATE, EXT4(default) → WAL,F2FS: 348% up



fsync() vs. fdatasync()

SQLite Insert

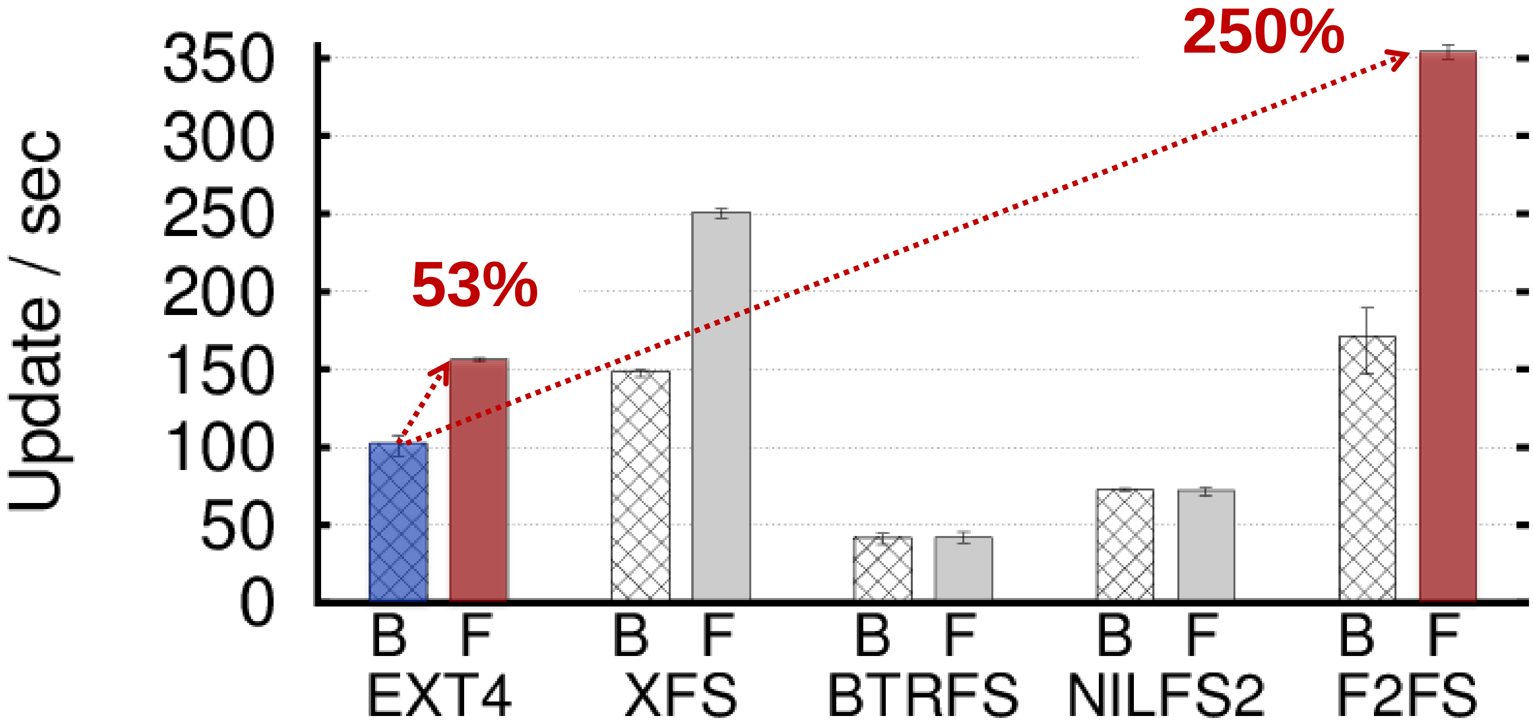
- fsync() → fdatasync() : 17% up
- fsync() → fdatasync() and F2FS : 126% up



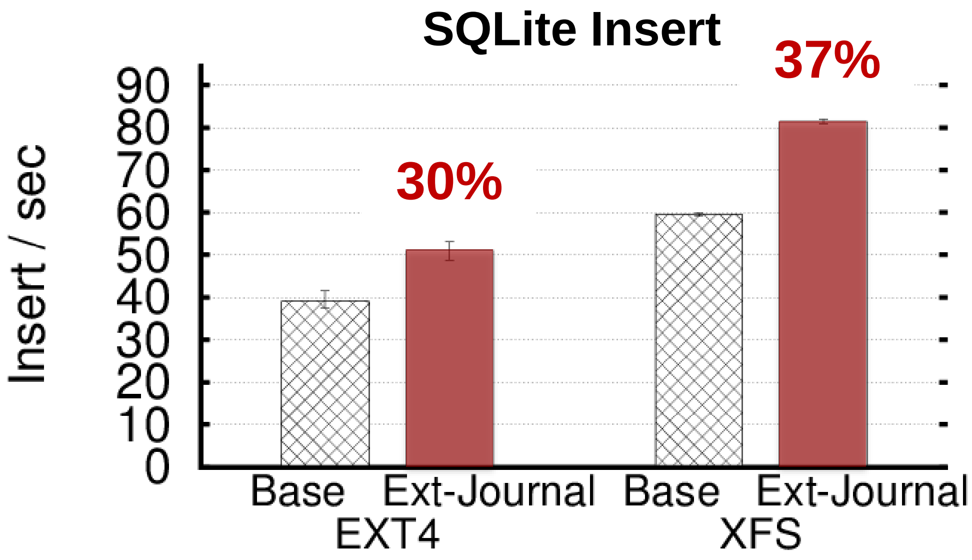
fsync() vs. fdatasync()

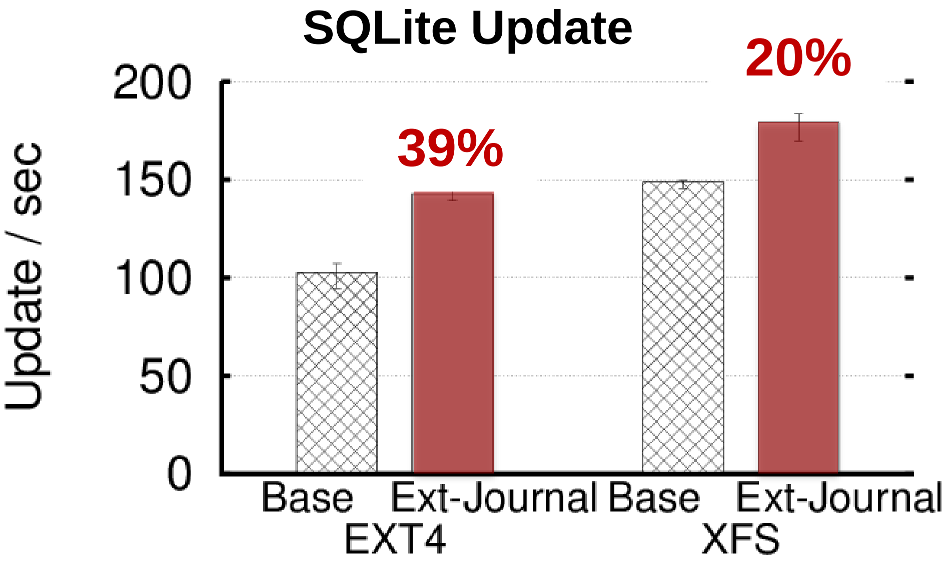
SQLite Update

- `fsync()` → `fdatasync()` : 53% up
- `fsync()` → `fdatasync()` and F2FS : 250% up



External journaling





Polling

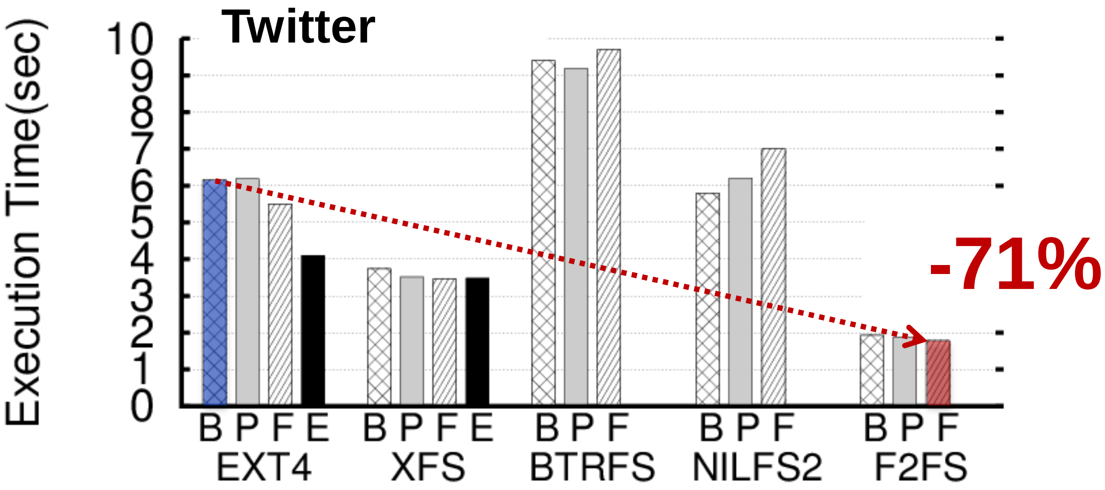
4 KB random write+fsync()

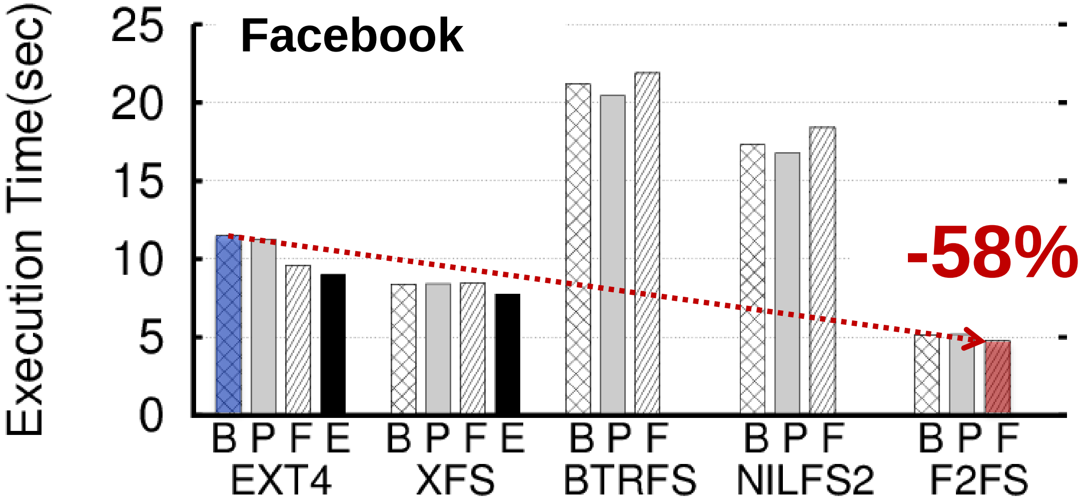
# of thread	Scenario	Idle		HD Record	
		base	poll	base	poll
1	KIOPS	1002	981	667	756
	CPU (%)	7.5	10.9	26.4	30.2
10	KIOPS	2609	2705	2136	2351
	CPU (%)	11.1	12.9	30.1	33.1

- Marginal gain (1~2%) when CPU is IDLE.
- 13% gain when we record HD video in background.

Real Workload

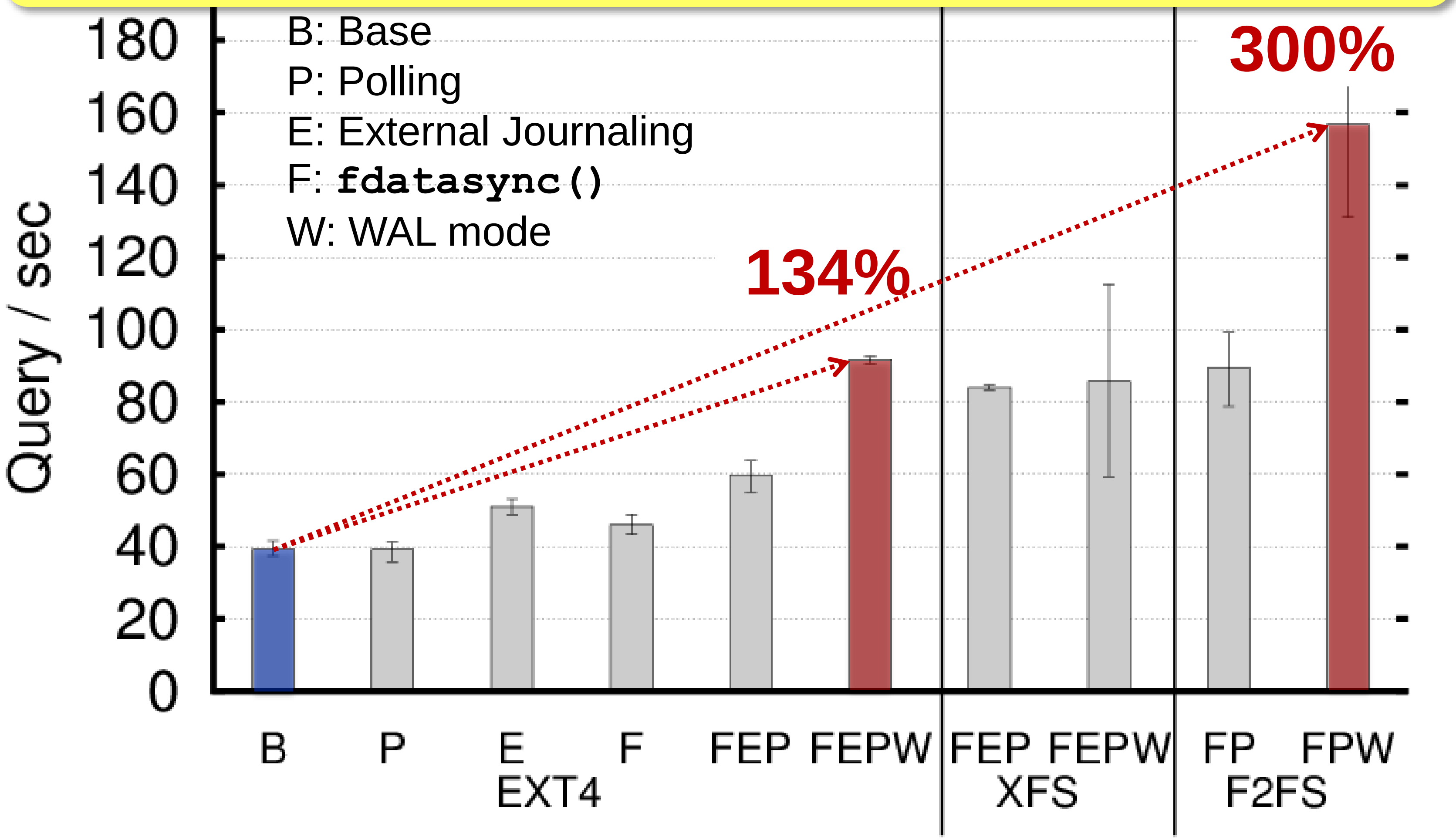
Replay Twitter and Facebook by Mobigen





Combining All the Improvements

fdatasync(), Ext. J, Polling, WAL: 134% up

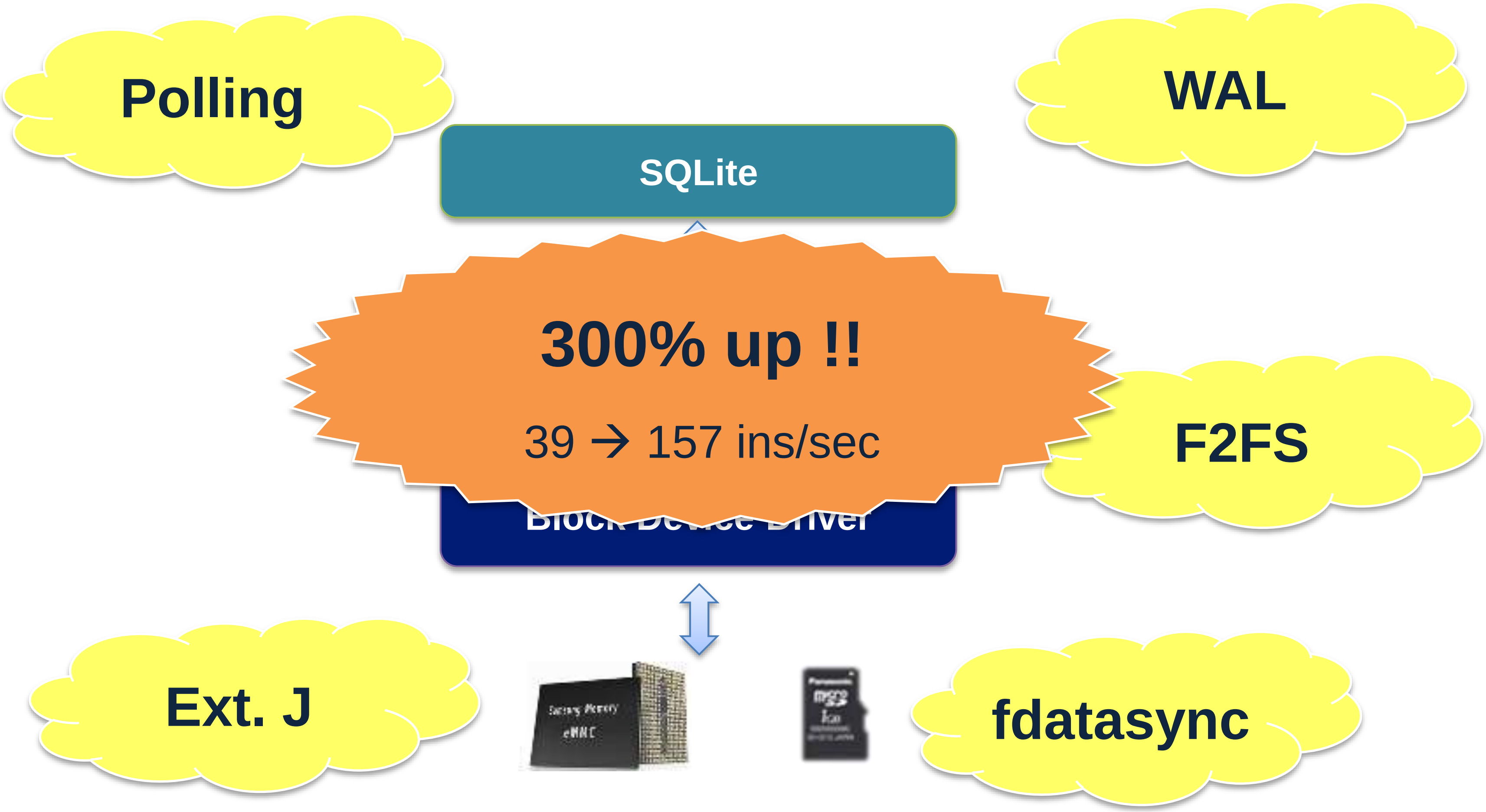


B: Base
P: Polling
E: External Journaling
F: `fdatasync()`
W: WAL mode

134%

300%

Finally,



Polling

WAL

SQLite

300% up !!

39 → 157 ins/sec

F2FS

Block Device Driver

Ext. J

fdatasync

Finally,

Polling

WAL

F2FS

Ext. J

atasync

Conclusion

- Android IO stack is collection of unorchestrated layers.
- Journaling of Journal(JOJ) lies at the core of the problem.
- We optimize Android I/O stack with WAL mode in SQLite, F2FS, `fdatasync(),` External journaling, polling based I/O.

What we achieved is…

- With legacy EXT4, SQLite performance improves by 134%.
- With F2FS, SQLite performance improves by 300%

solely via software modification on existing smartphone!

Thank you…

77smart@hanyang.ac.kr