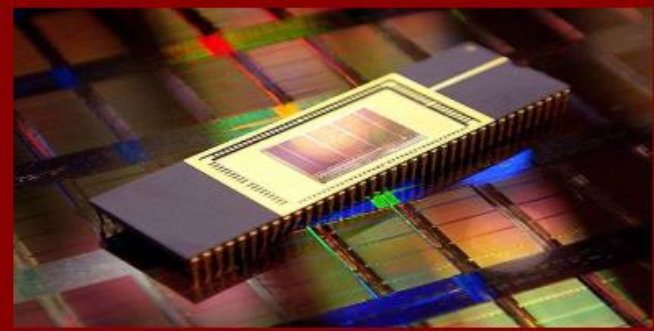


NVRAMOS 2018

Operating System Support for
Next Generation Large Scale NVRAM

Organized by KIISE SIGFAST & Computer Systems Society,
Oct. 25 - Oct. 27, 2018, Jeju, Korea



SQL Statement Logging for Making SQLite Truly Lite

Jong-Hyeok Park, Gihwan Oh, Sang-Won Lee



Outline

▶ About SQLite

▶ Motivation

- Problem Definition
- Why Logical Logging?

▶ SQLite/SSL

- Architecture and Implementation

▶ Performance Evaluation

▶ Conclusion



De-facto standard mobile DBMS

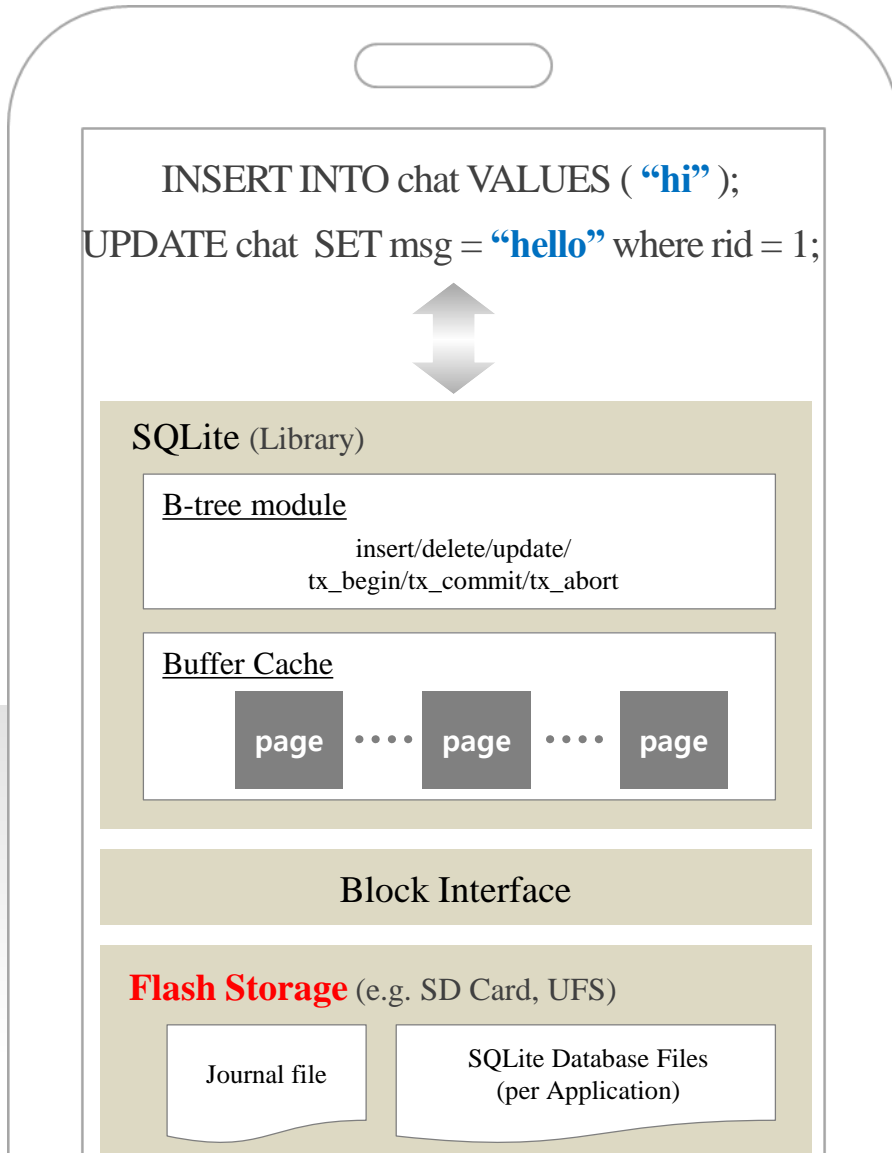
Productivity

Solid transactional support

Lightweight codebase



SQLite is **NOT LITE**



Huge Write Amplification

**Auto-Commit
Force-Write
Block Interface
Journaling
File system Metadata**

Durability & Atomicity







Durability

Performance

Life span



Alternative Logging Mechanisms [Gray 90]

	Physical	Aries-style Physiological	Logical
Method	Page-wise	Delta	SQL statement
Scheme	Vanilla SQLite	SQLite/PPL ^[VLDB 15]	SQLite/SSL
Log Size			
Recovery			

[Gray 90] J. Gray and A. Router. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

[VLDB 15] G. Oh, S. Kim, S.-W. Lee, and B. Moon. SQLite Optimization with Phase Change Memory for Mobile Applications. Proceedings of VLDB Endowment, 8(12), Aug. 2015.

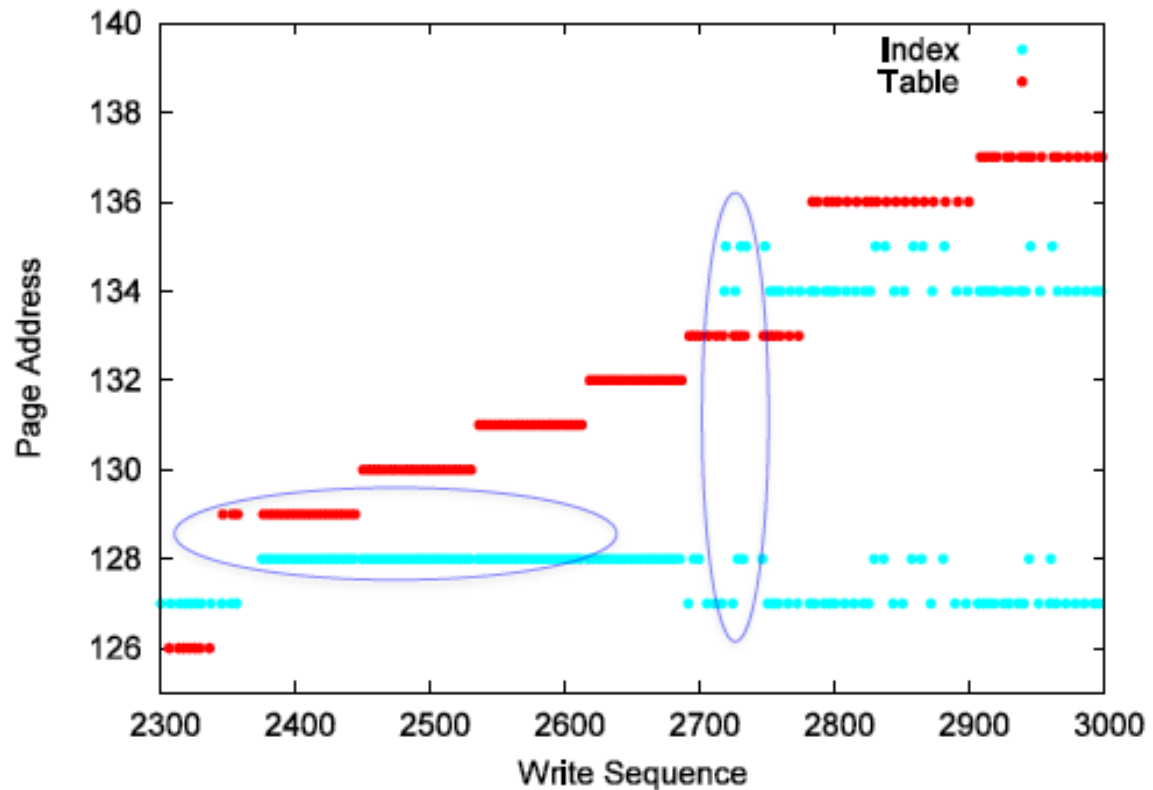
Why We Revisit Logical Logging?

Technical Preconditions

- ✓ Single User
- ✓ Strong Update Locality
- ✓ Transaction Consistent Checkpoint Mechanism

[Gray 90, CSUR 83, ICDE 14]

< Sequence of page-write request in SQLite >



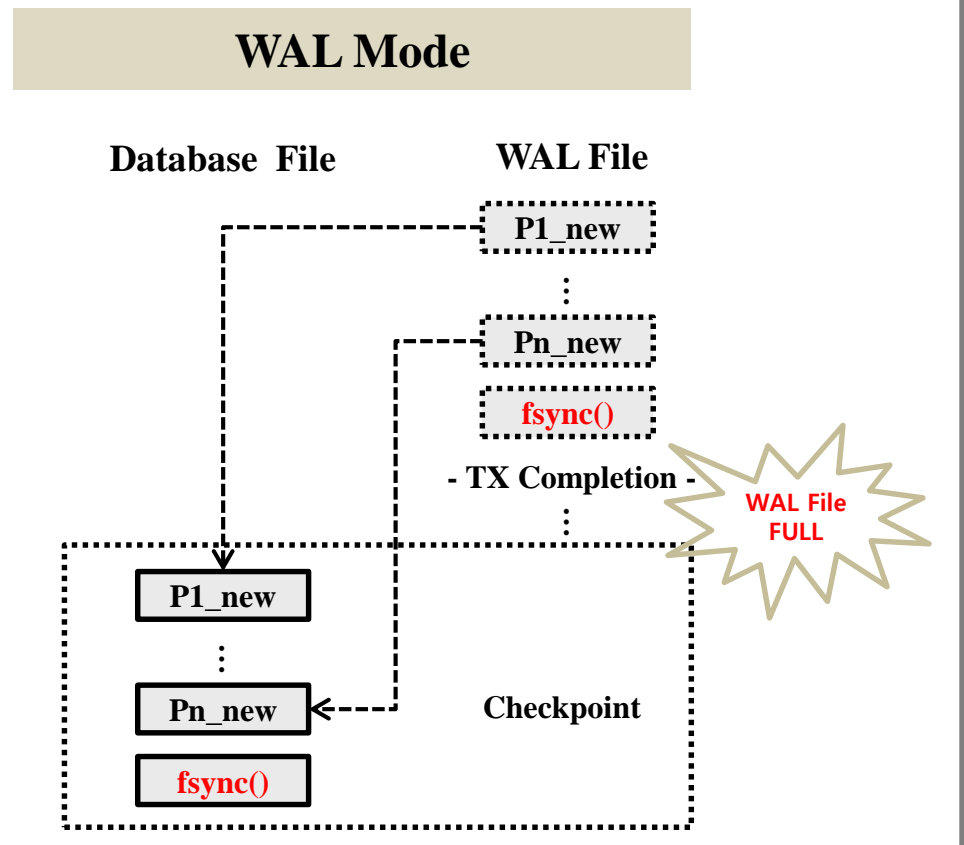
Why Logical Logging? (2)

Technical Preconditions

- ✓ Single User
- ✓ Strong Update Locality
- ✓ Transaction Consistent Checkpoint Mechanism

[Gray 90, CSUR 83, ICDE 14]

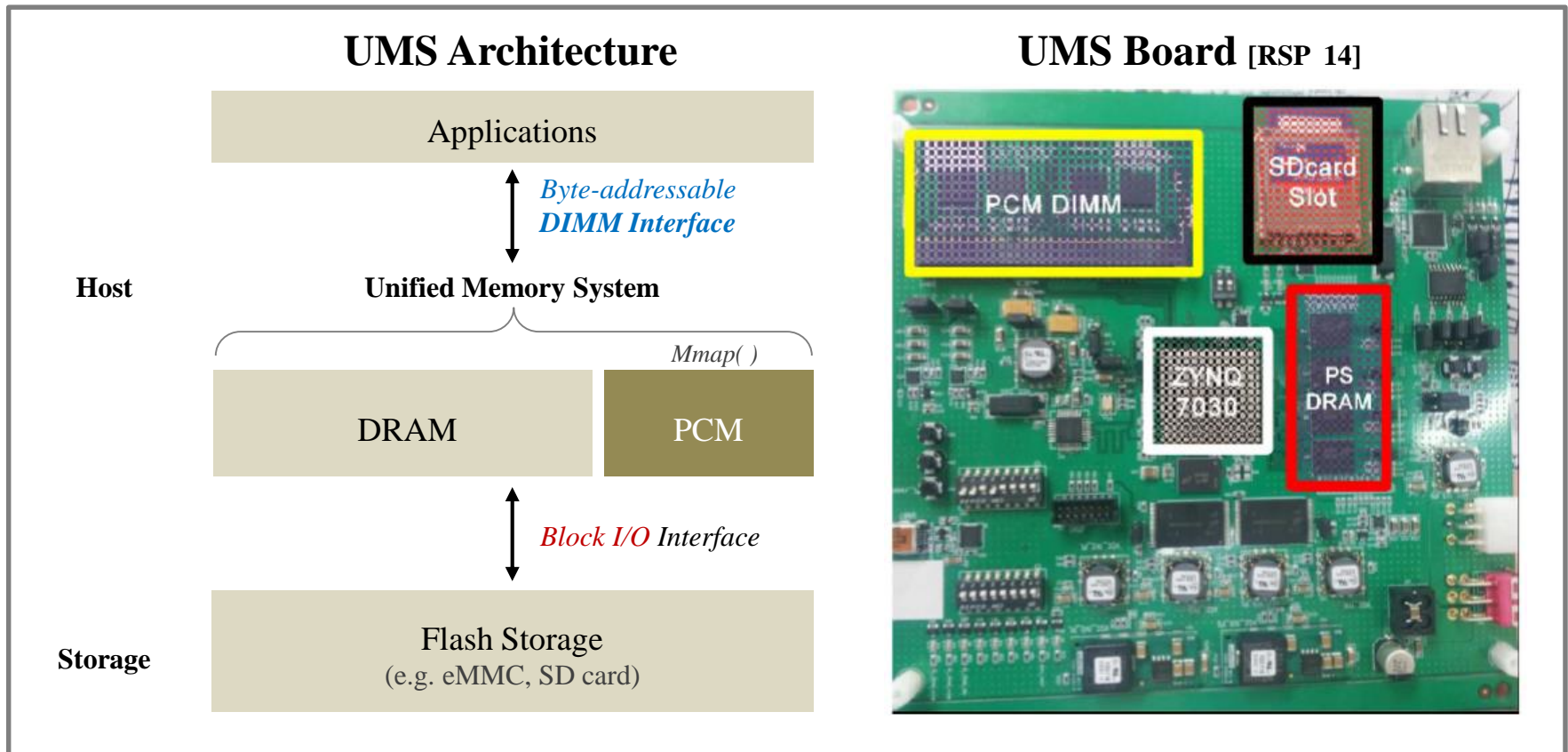
< SQLite WAL Mechanism >



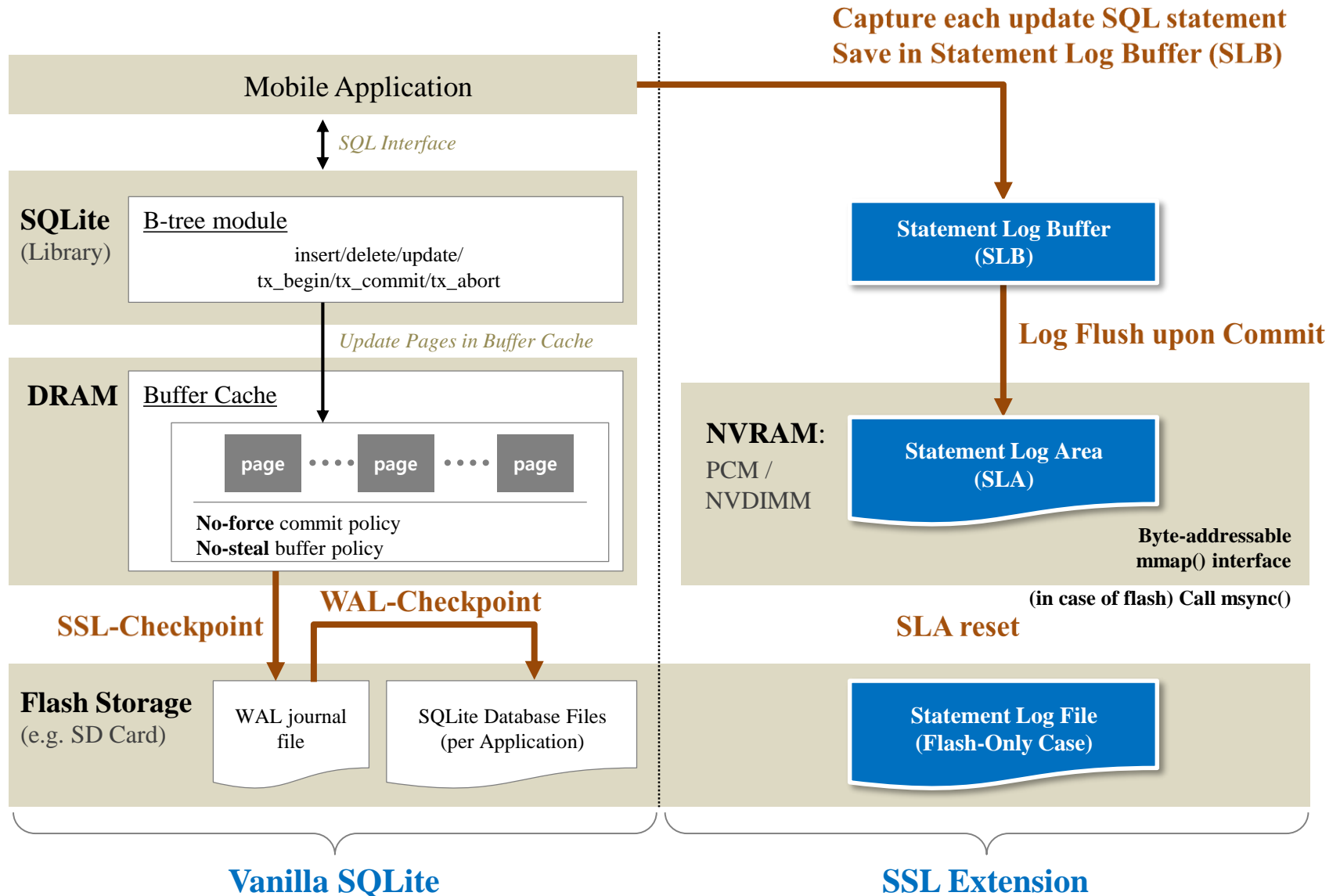
Why Logical Logging? (3)

Non Volatile Memory & Logical Logging

- ▶ Byte Addressable
- ▶ Avoid I/O Stack
- ▶ Enable to realize full potential of Logical Logging



Design of SQLite/SSL



Recovery

SLA = reset & No WAL file

- Crashed during Normal shutdown

→ **Create WAL journal file**

SLA = reset & WAL = reset

- Crashed during Initialization
- Crashed after WAL-Checkpoint

→ **No need to recovery**

SLA = reset & WAL = in-use

- Crashed after SSL-Checkpoint

→ **Copy latest pages in WAL to DB file**

SLA = in-use

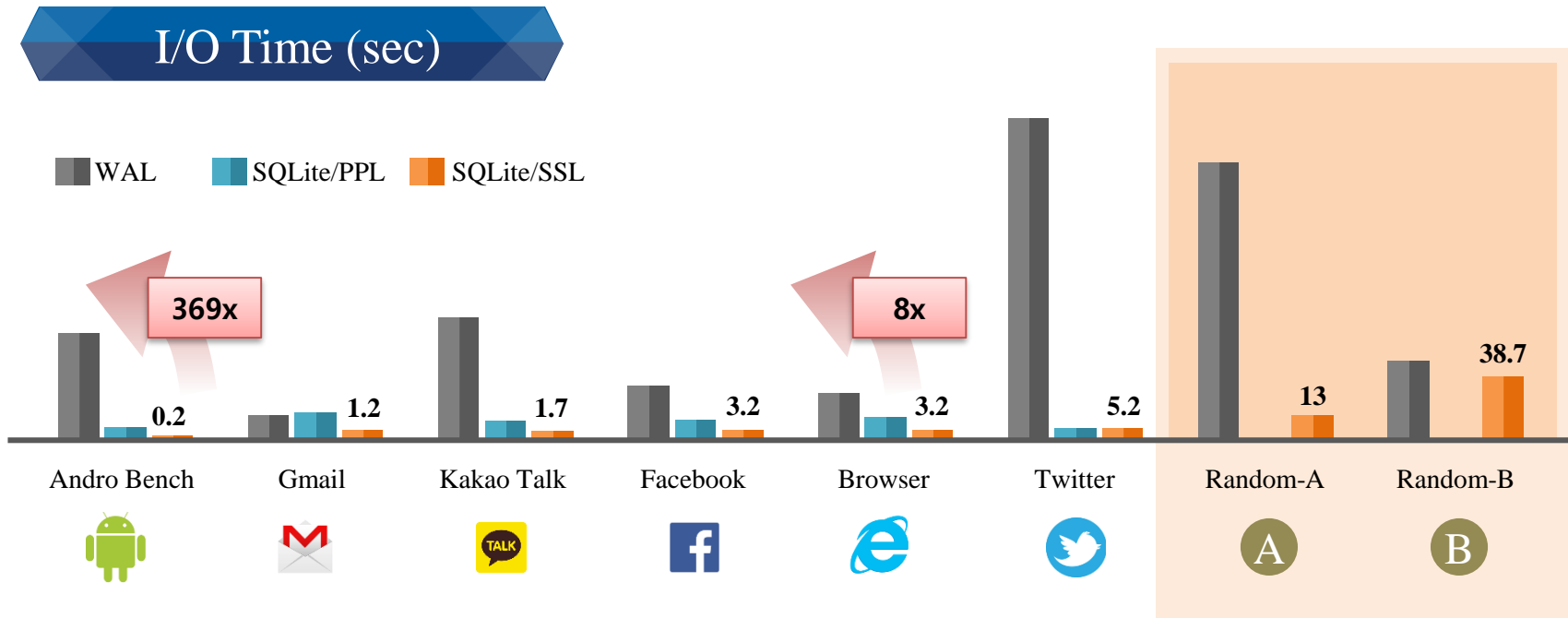
- Crashed prior to SSL-Checkpoint
- Crashed during SSL-Checkpoint

→ **Re-executes SQL statement in SLA**

Performance Evaluation

UMS-Board : PCM as SLA log device

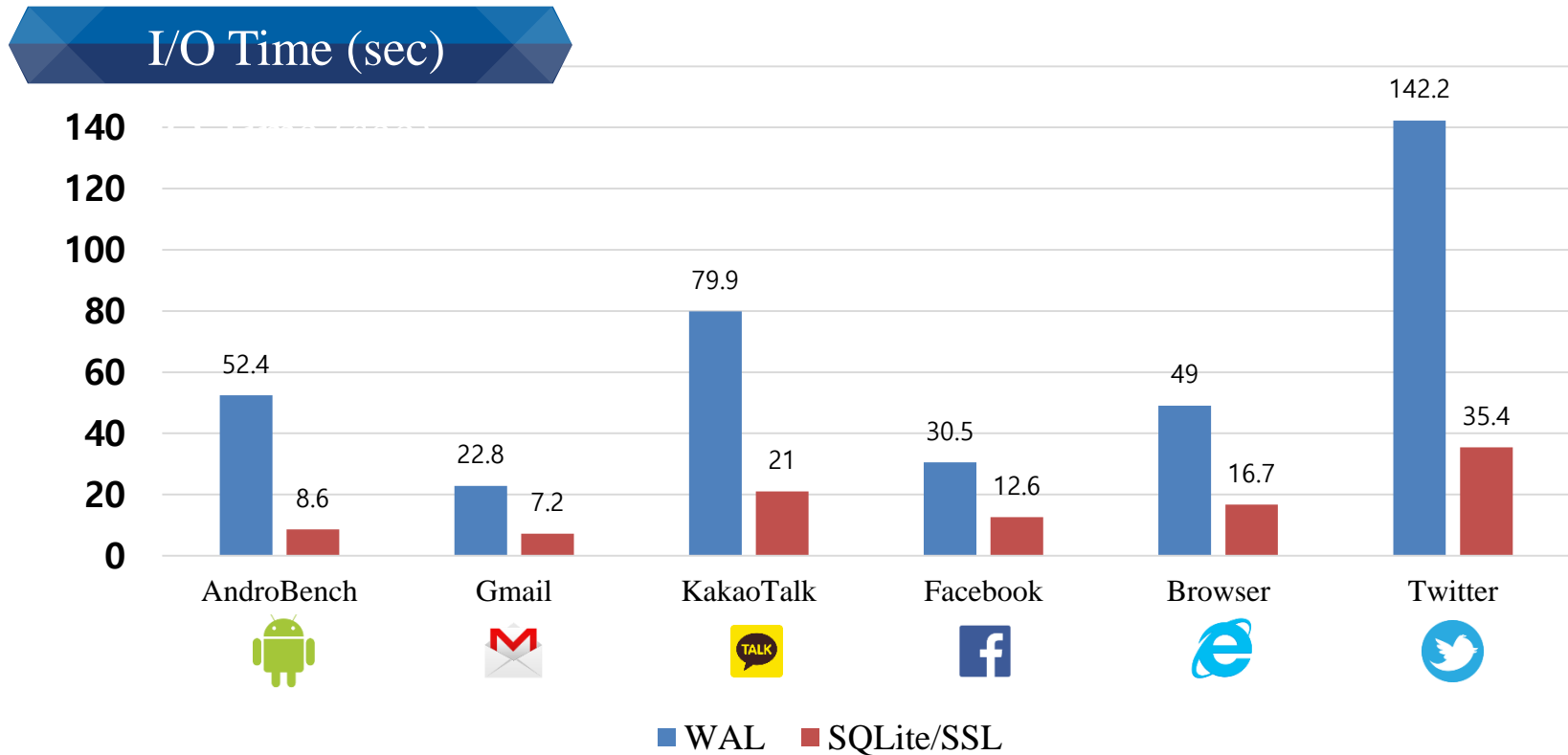
- Reduce # of Checkpoints
- Reduce # of Writes



- **No worse** than Vanilla SQLite even in fully random workloads
- In terms of recovery time, acceptable in practice (less than **1sec**)

Performance Evaluation (2)

PC : SD Card as SLA log device



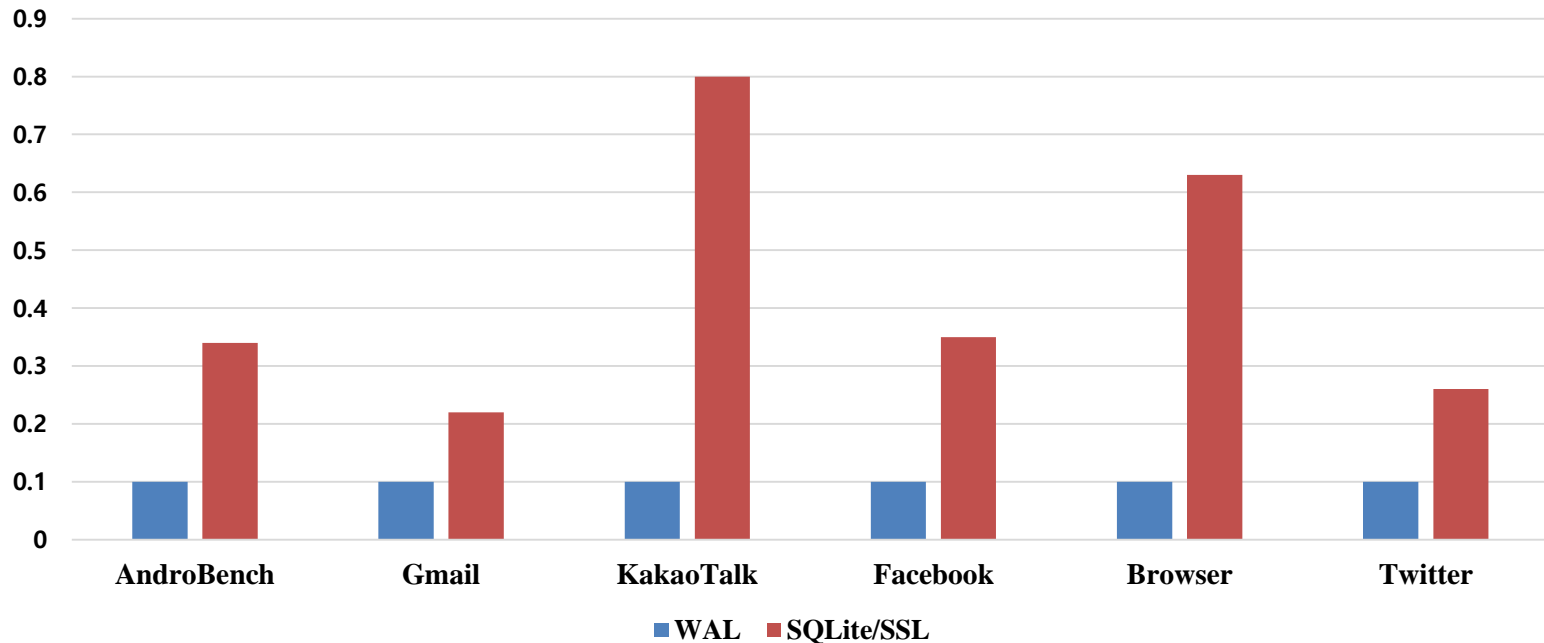
- In Flash-only, **2 ~ 6 times** better
- Demonstrate that SQLite/SSL is quite effective without NVM

Performance Evaluation (3)

Recovery Performance

- Acceptable in practice
- Worst-case scenario : SLA = in-use (*FULL*)

Recovery Time (sec)



Conclusion

SQLite/SSL demonstrates that logical logging can be fully and effectively realized in mobile DBMSs

▶ **Key observation about mobile workloads**

: Short transactions with strong update locality

▶ **Transaction-consistent checkpoint in SQLite**

: WAL journaling can be naturally extended for TCC

▶ **Emerging NVMs**

: Byte-addressability makes SSL more attractive

Future Works

▶ **UFS with MRAM**

▶ **Add competitive edges to domestic storage devices and mobile platforms**

Q&A

Recovery

SLA = reset & No WAL file

- Crashed during Normal shutdown

➔ **Create WAL journal file**

SLA = reset & WAL = reset

- Crashed during Initialization
- Crashed after WAL-Checkpoint

➔ **No need to recovery**

SLA = reset & WAL = in-use

- Crashed after SSL-Checkpoint

➔ **Copy latest pages in WAL to DB file**

SLA = in-use

- Crashed prior to SSL-Checkpoint
- Crashed during SSL-Checkpoint

➔ **Re-executes SQL statement in SLA**

Performance Evaluation

Experimental Setup

	UMS Board	PC
Processor	Xilinx Zynq-7030 dual ARM Cotex-A9 1GHz	Intel Core i7-3770 3.40 GHz
DRAM	1GB	12 GB
PCM	512 MB	-
Storage	SD Card : MB-MSBGA	
File system	EXT4	
Linux Kernel	3.9.0 Xilinx kernel	4.6 kernel

Performance Evaluation

Mobile Workloads

Trace	Androbench	Gmail	KakaoTalk	Facebook	Browser	Twitter
# of Files	1	1	1	11	6	17
DB size (MB)	0.19	0.74	0.45	1.95	2.51	6.08
Total # of TXs (Batch + Auto)	3,081 (2+3,079)	984 (806+178)	4,342 (432+3,910)	1,281 (262+1,019)	1,522 (1,439+29)	2,022 (17+2,005)
Total # of SQLs (Batch + Auto)	3,082 (3+3,079)	10,579 (10,419+178)	8,469 (4,559+3,910)	3,082 (2,063+1,019)	4,493 (4,464+29)	10,291 (448+2,005)
Page writes / T X	3.38	8.58	3.58	3.11	3.88	1.40
Avg. size of update SQL stmt/TX (B)	215	1,913	1,094	1,094	8,304	506