NVRAMOS '14
10.30. 2014

Resolving Journaling of Journal Anomaly via Weaving Recovery Information into DB Page

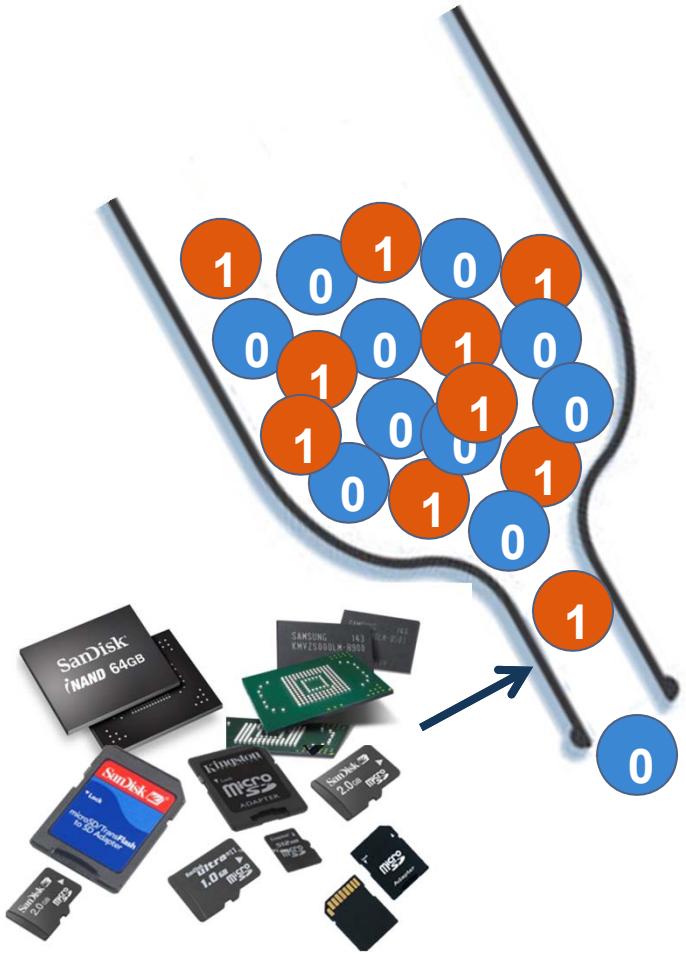
Beomseok Nam
UNIST

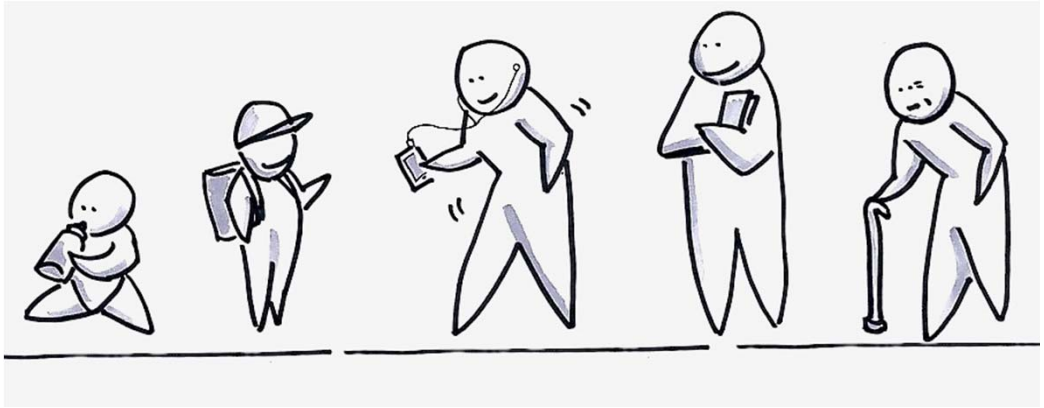
Outline

- Motivation
  - Journaling of Journal Anomaly
- How to resolve Journaling of Journal anomaly
  - Multi-Version B-Tree (MVBT)
- Optimizations of Multi-Version B-Tree
  - Lazy Split
  - Reserved Buffer Space
  - Lazy Garbage Collection
  - Metadata Embedding
  - Disabling Sibling Redistribution
- Evaluation
- Conclusion

Storage I/O Problems in Android

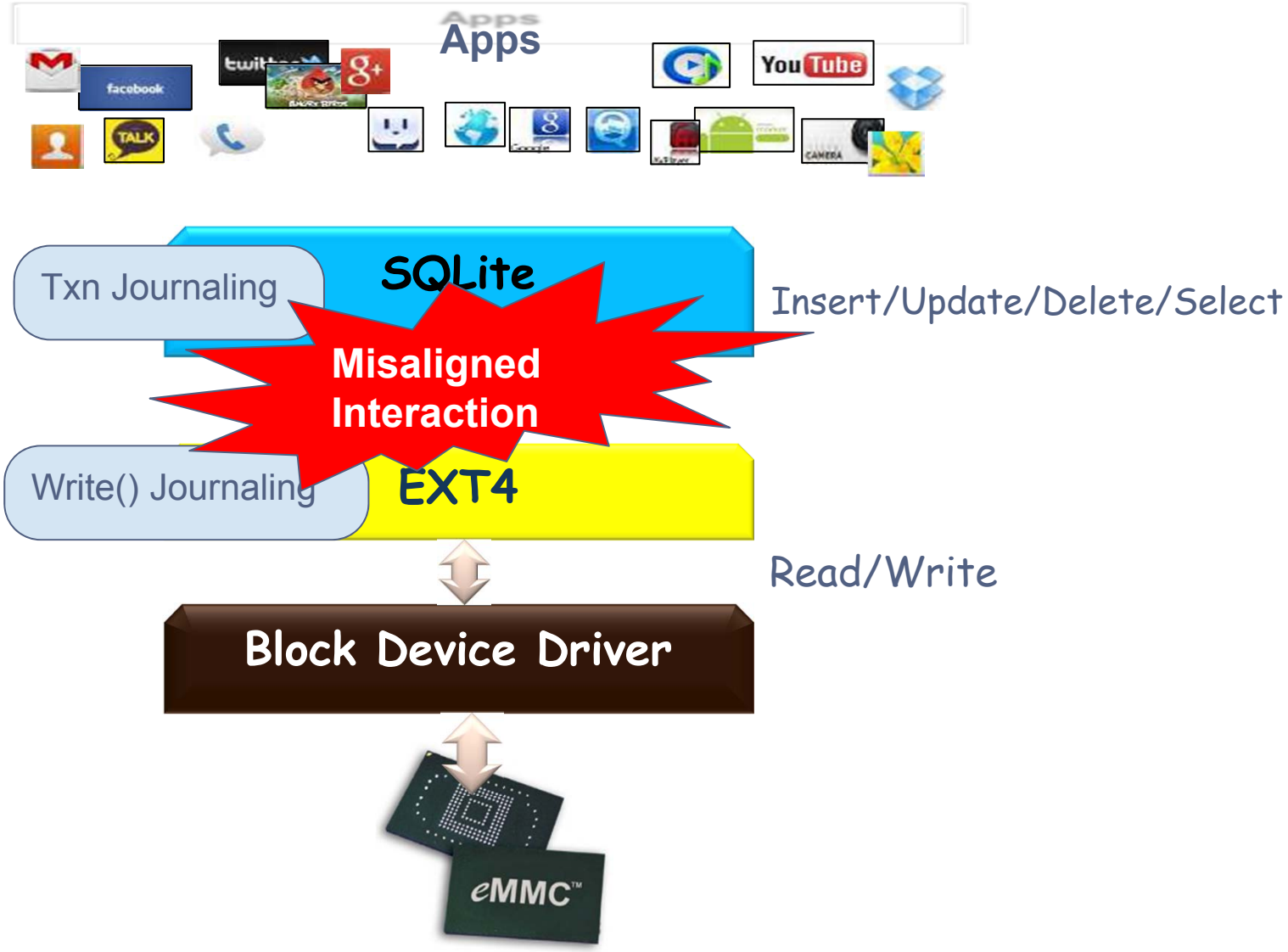
- Performance Bottleneck
- Lifetime of Storage





Cause = Excessive IO

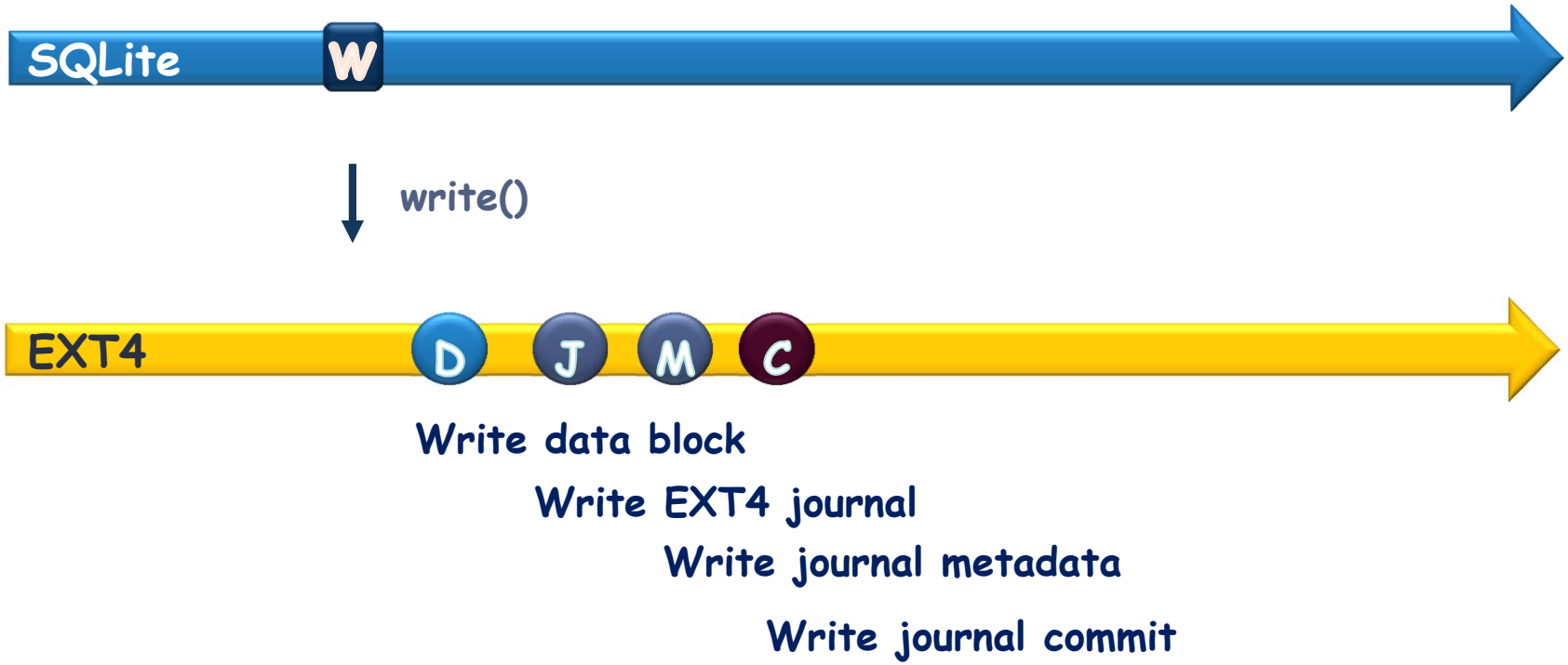
Android I/O Stack



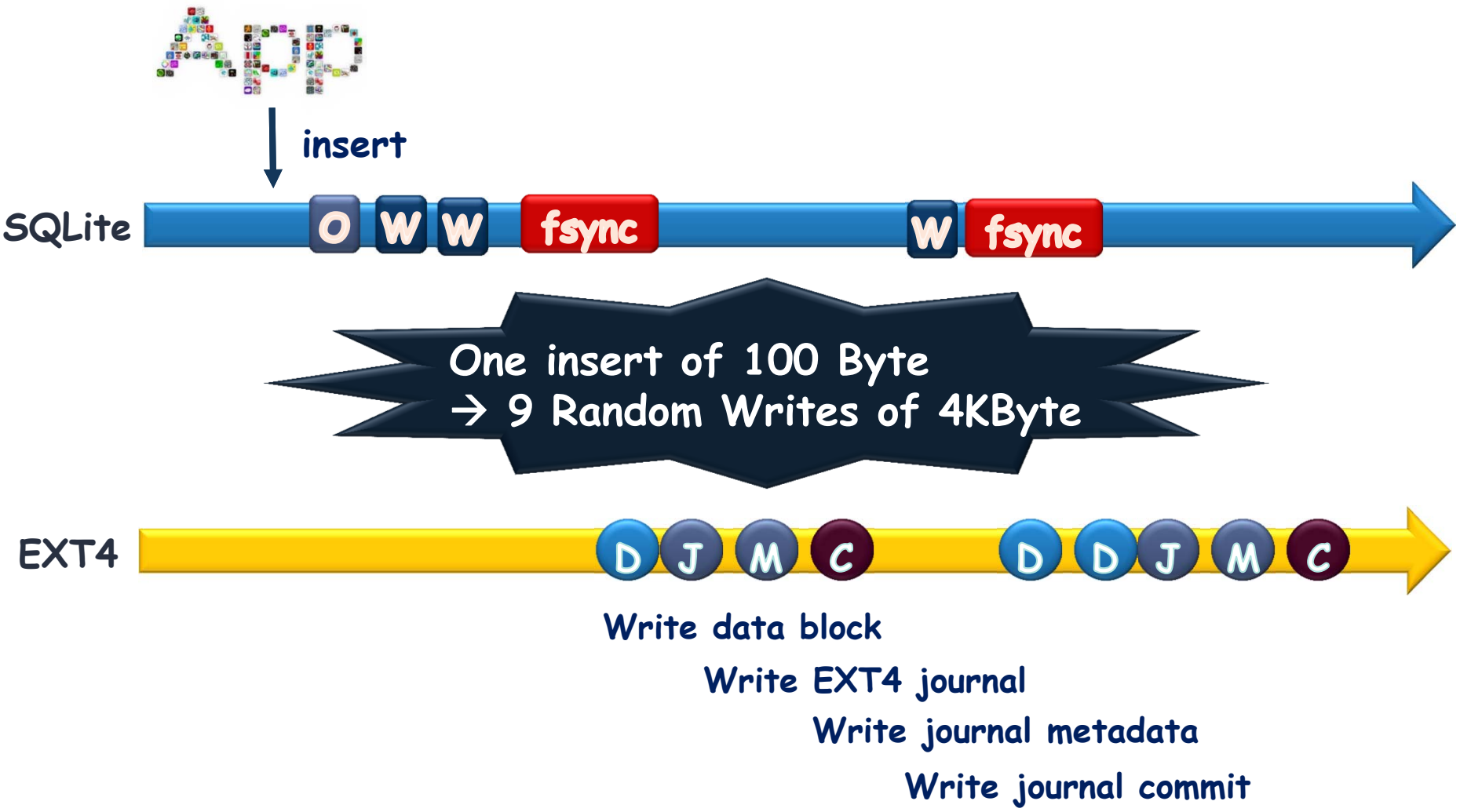
Journaling of Journal Anomaly

- Journaling in SQLite (TRUNCATE mode)

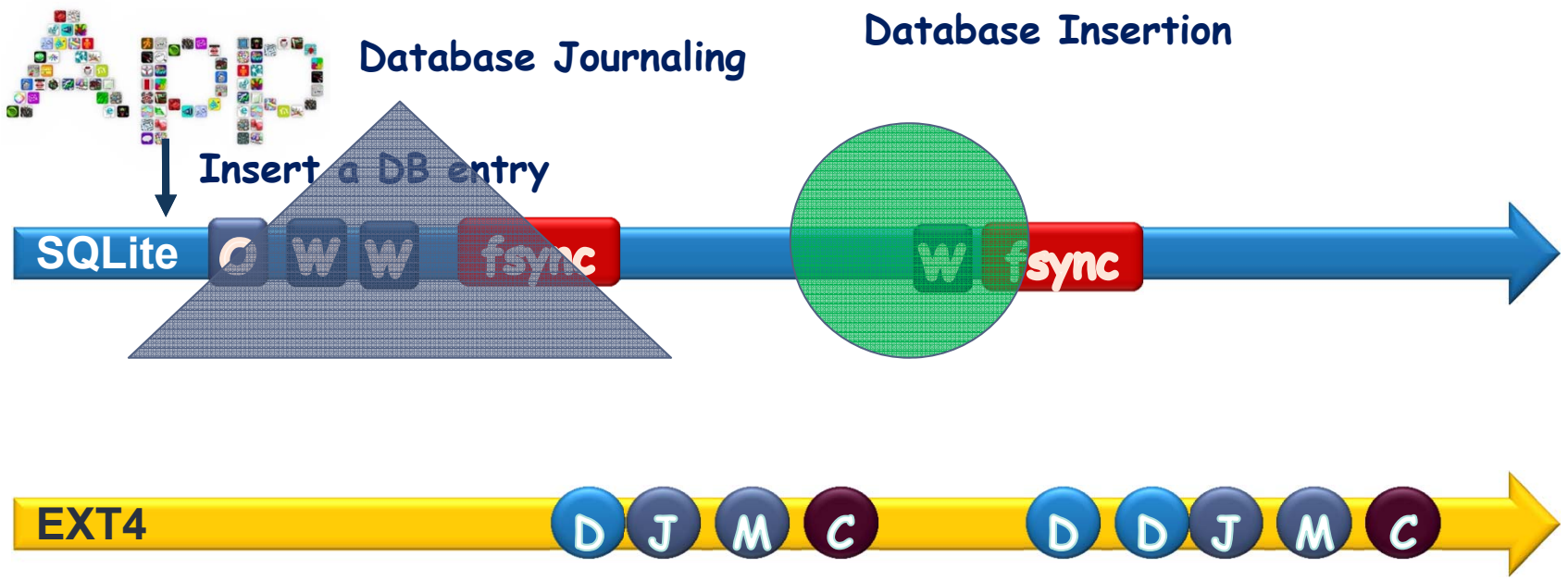
EXT4 Journaling (ordered mode)



Journaling of Journal Anomaly

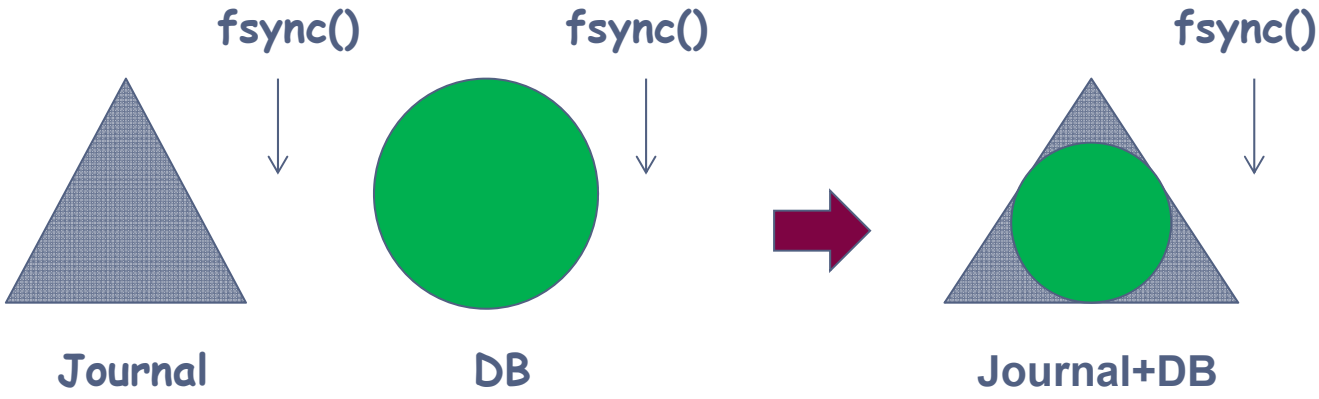


How to Resolve Journaling of Journal?



Journaling of journal anomaly.

How to Resolve Journaling of Journal?

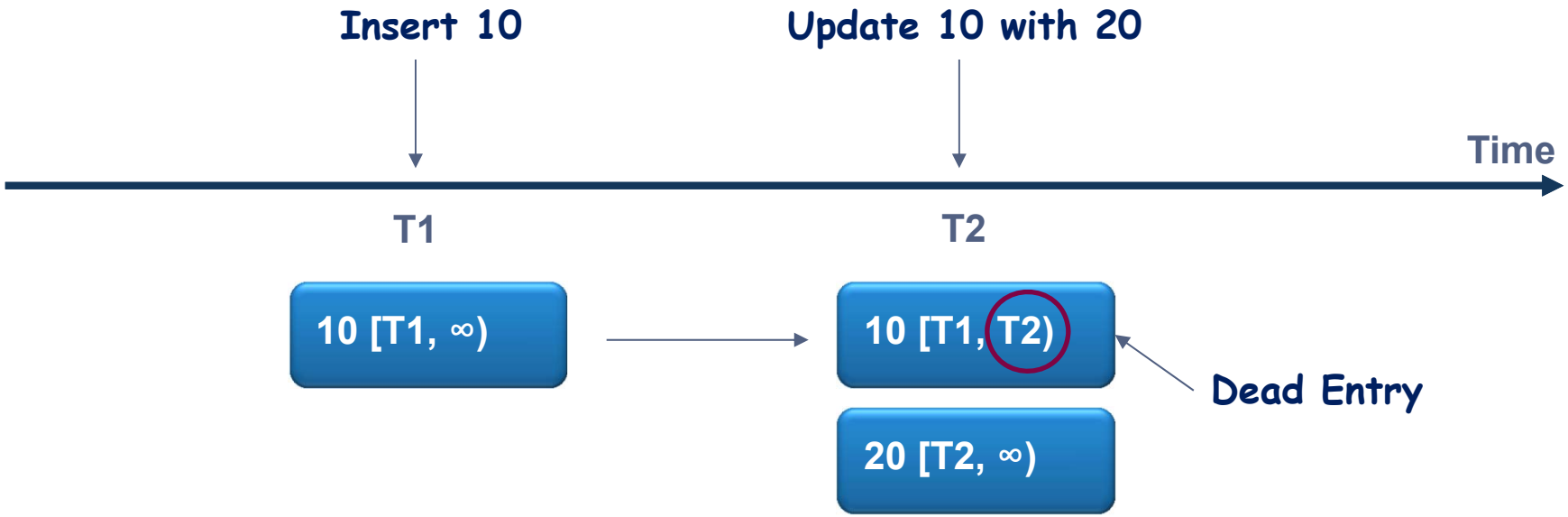


=

Version-based B-Tree (MVBT)

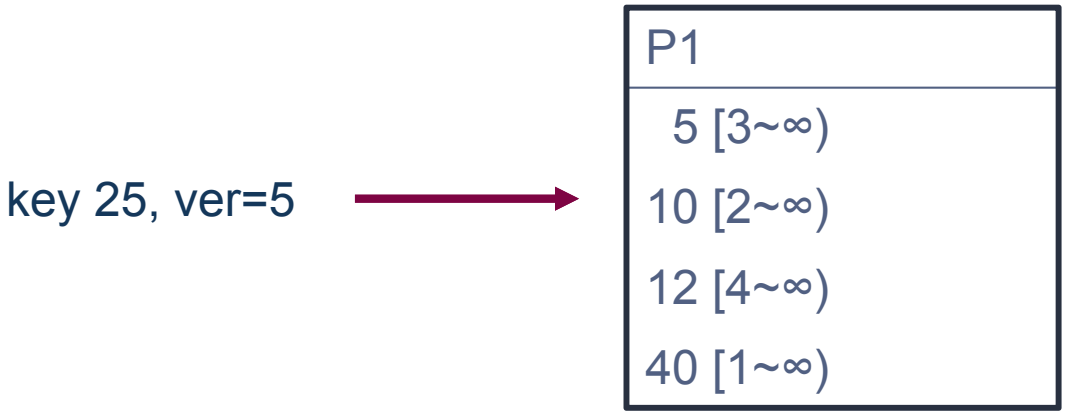
Version-based B-Tree (MVBT)

- Versioning

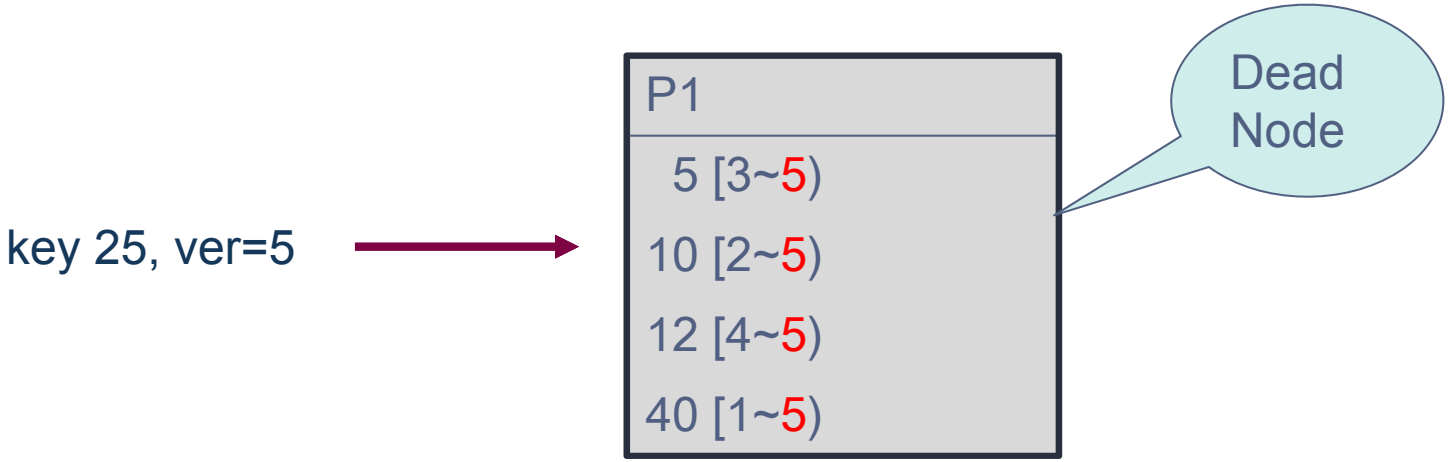


Do not overwrite old version data
Do not need a rollback journal

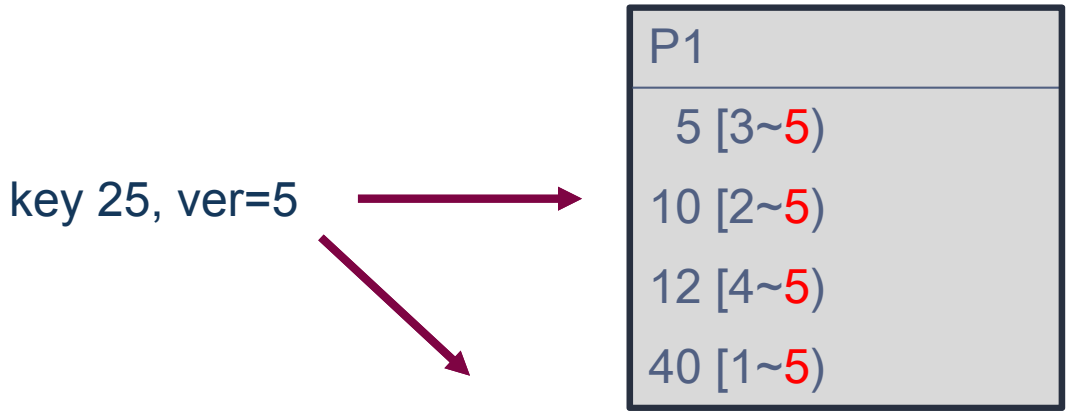
Node Split in Multi-Version B-Tree



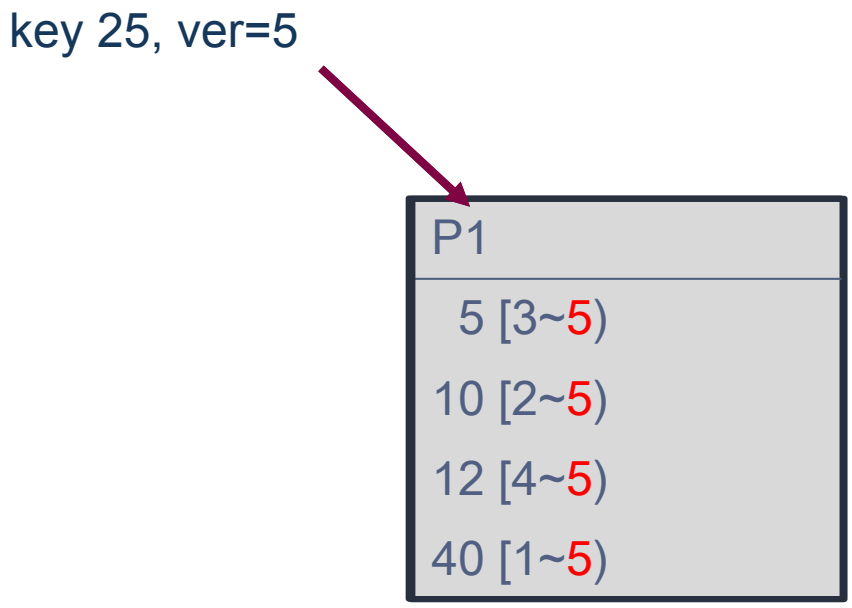
Node Split in Multi-Version B-Tree



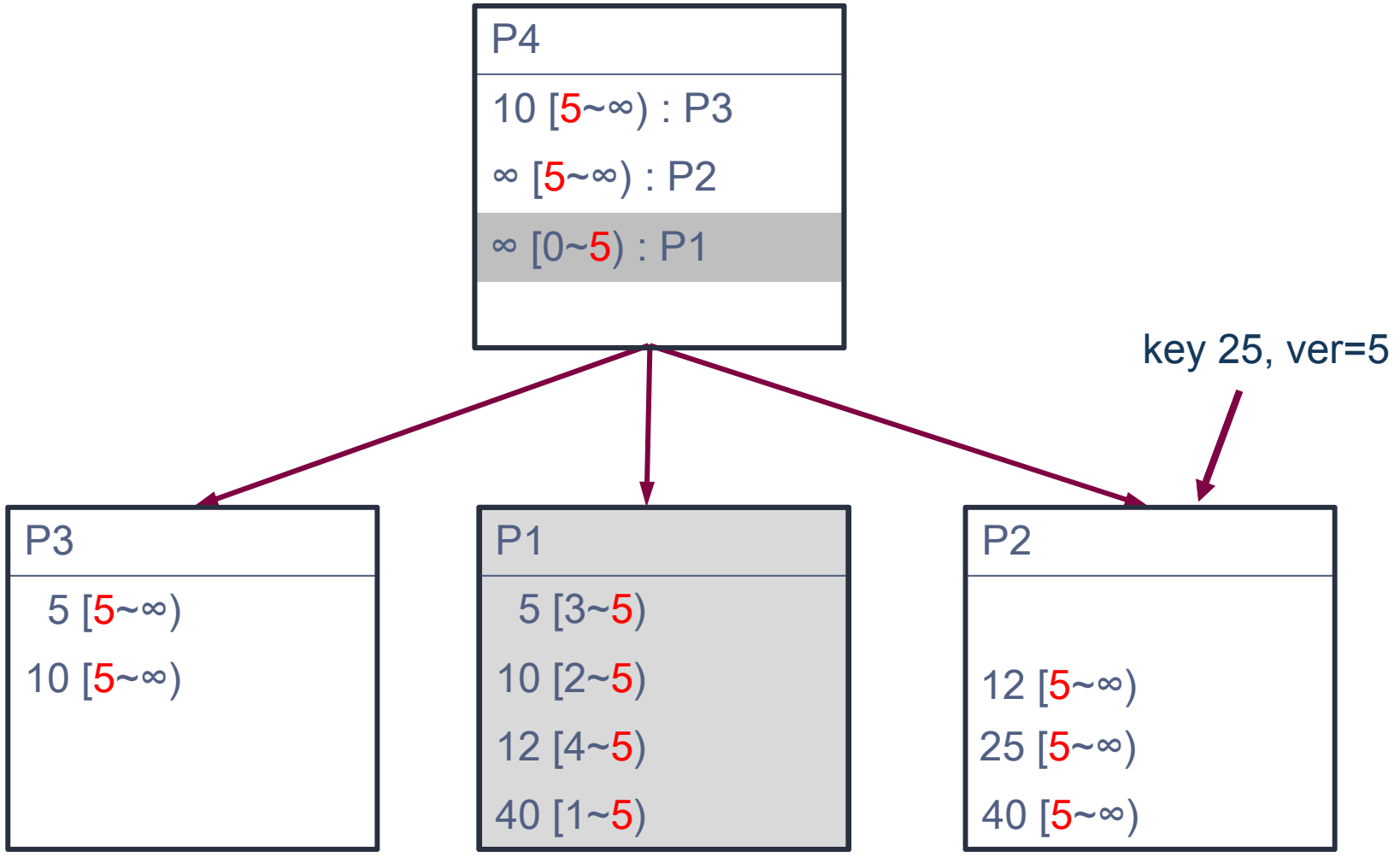
Node Split in Multi-Version B-Tree



Node Split in Multi-Version B-Tree

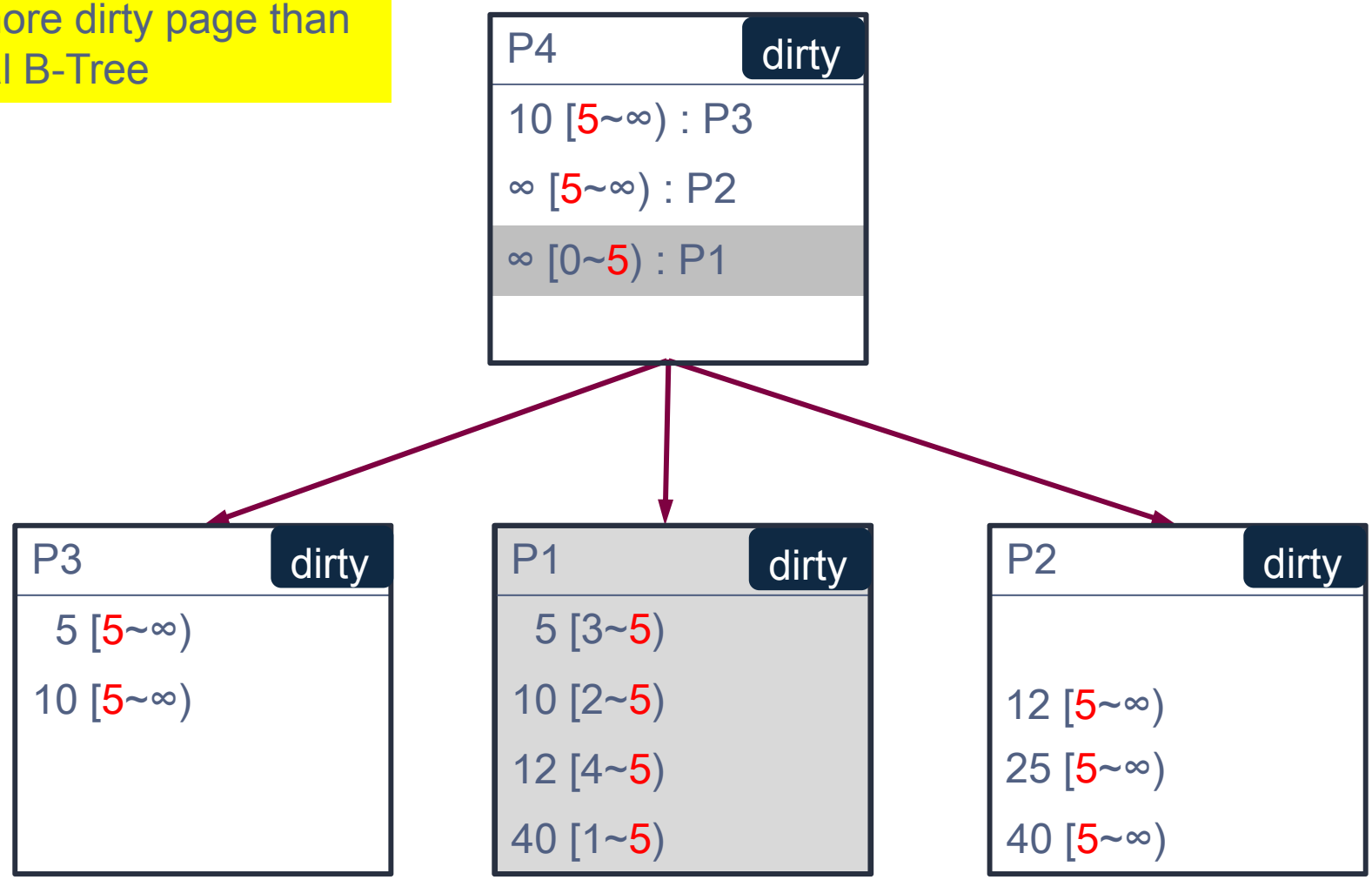


Node Split in Multi-Version B-Tree

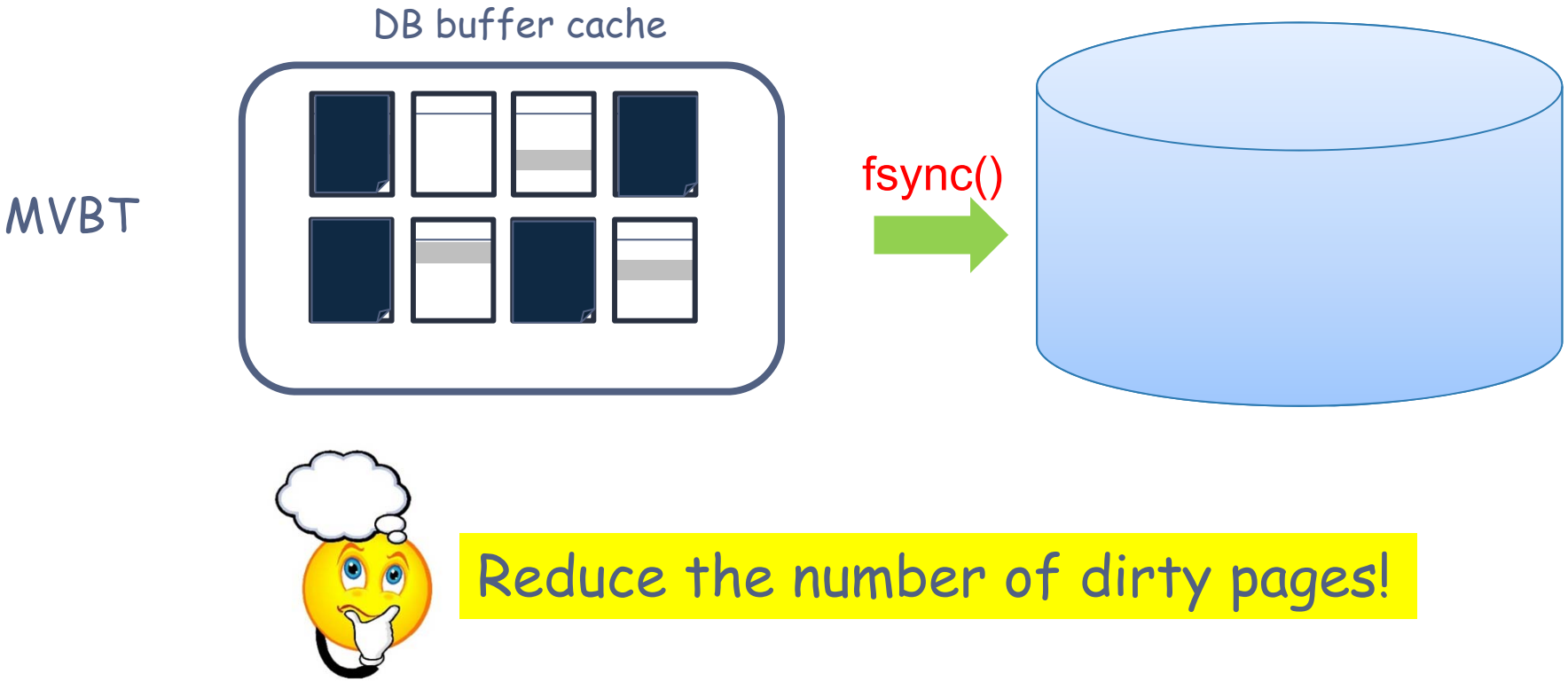


Node Split in Multi-Version B-Tree

One more dirty page than original B-Tree

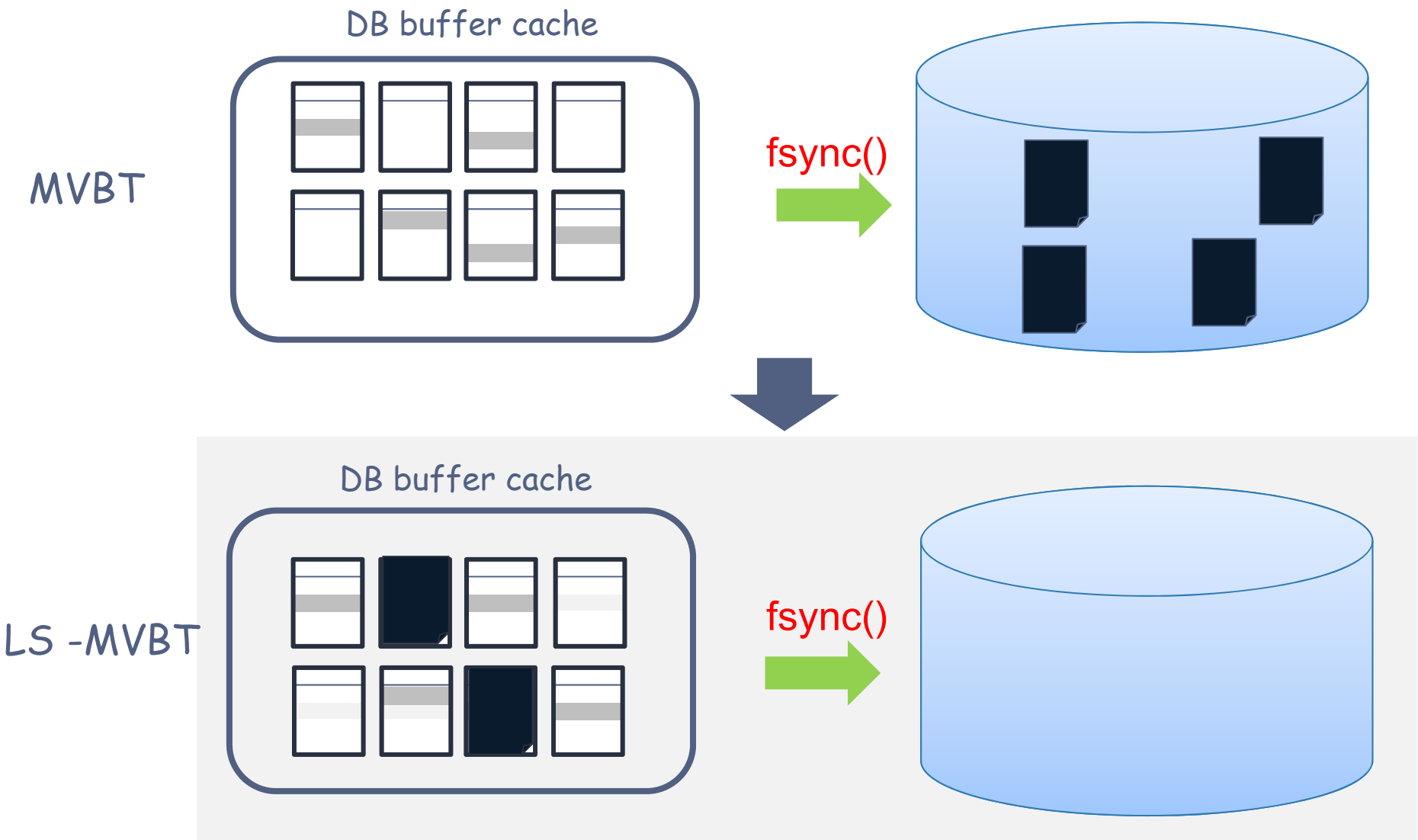


I/O Traffic in MVBT

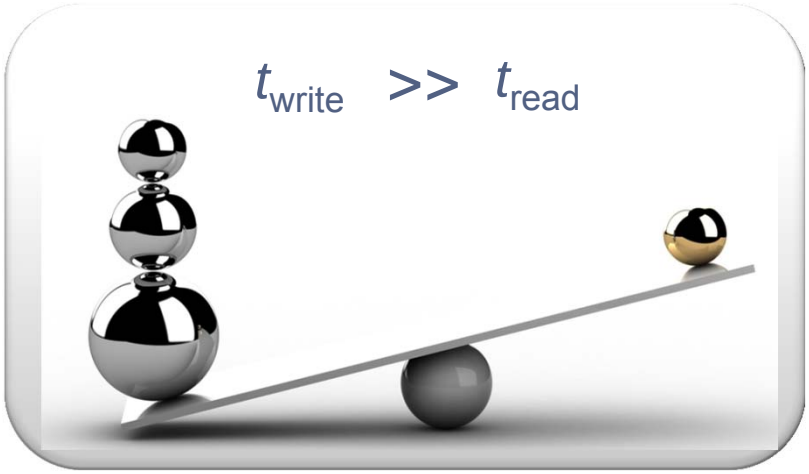


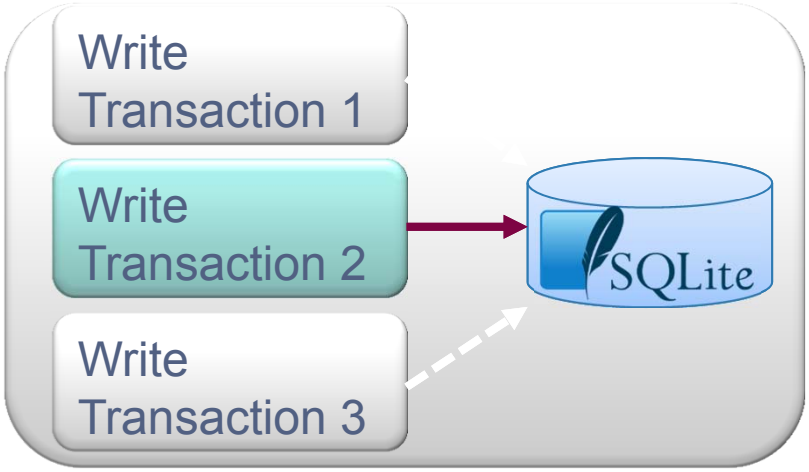
Reduce the number of dirty pages!

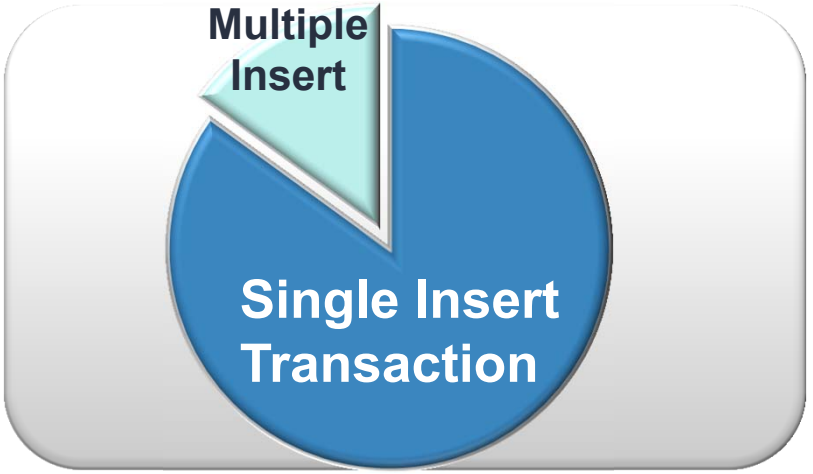
I/O Traffic in MVBT



Optimizations in Android I/O

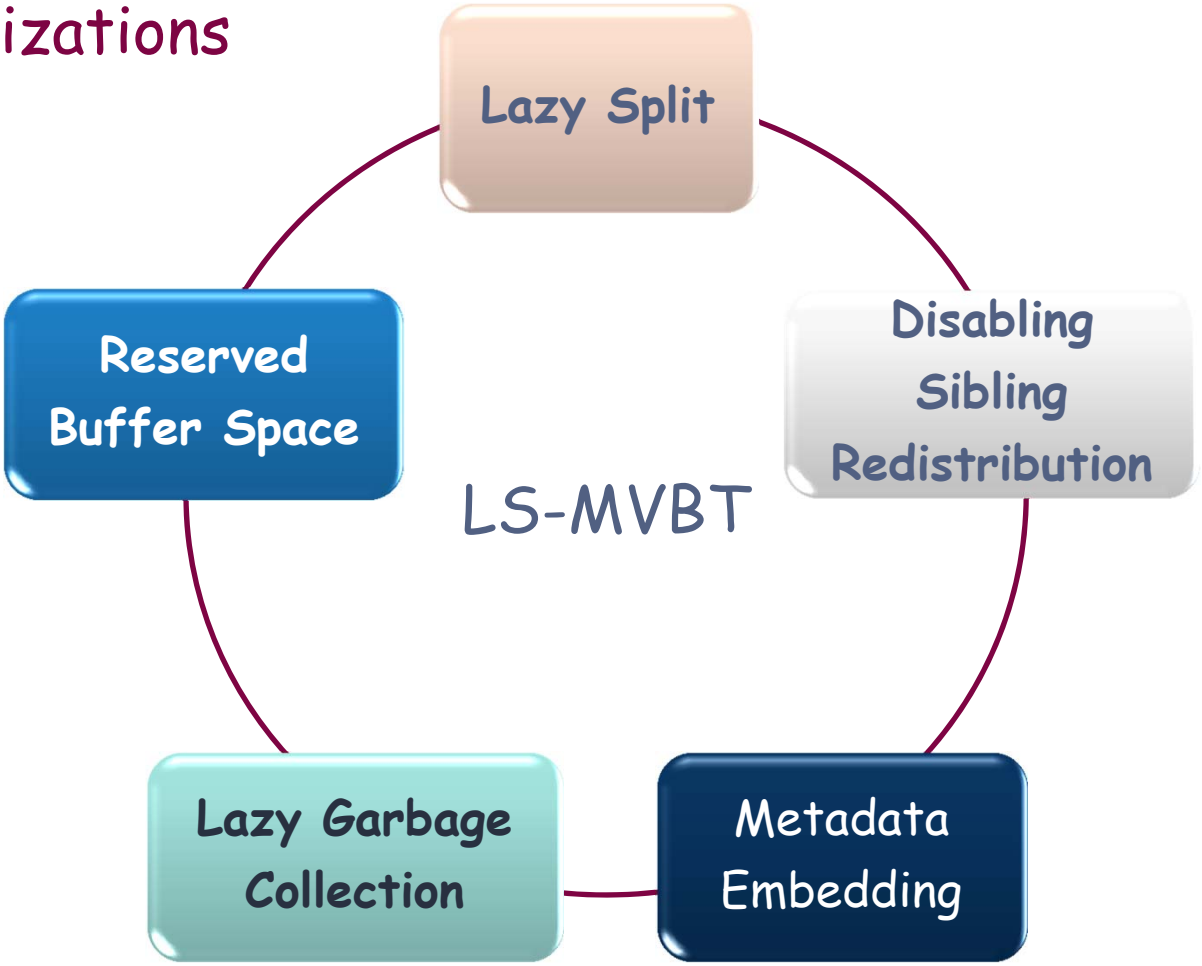






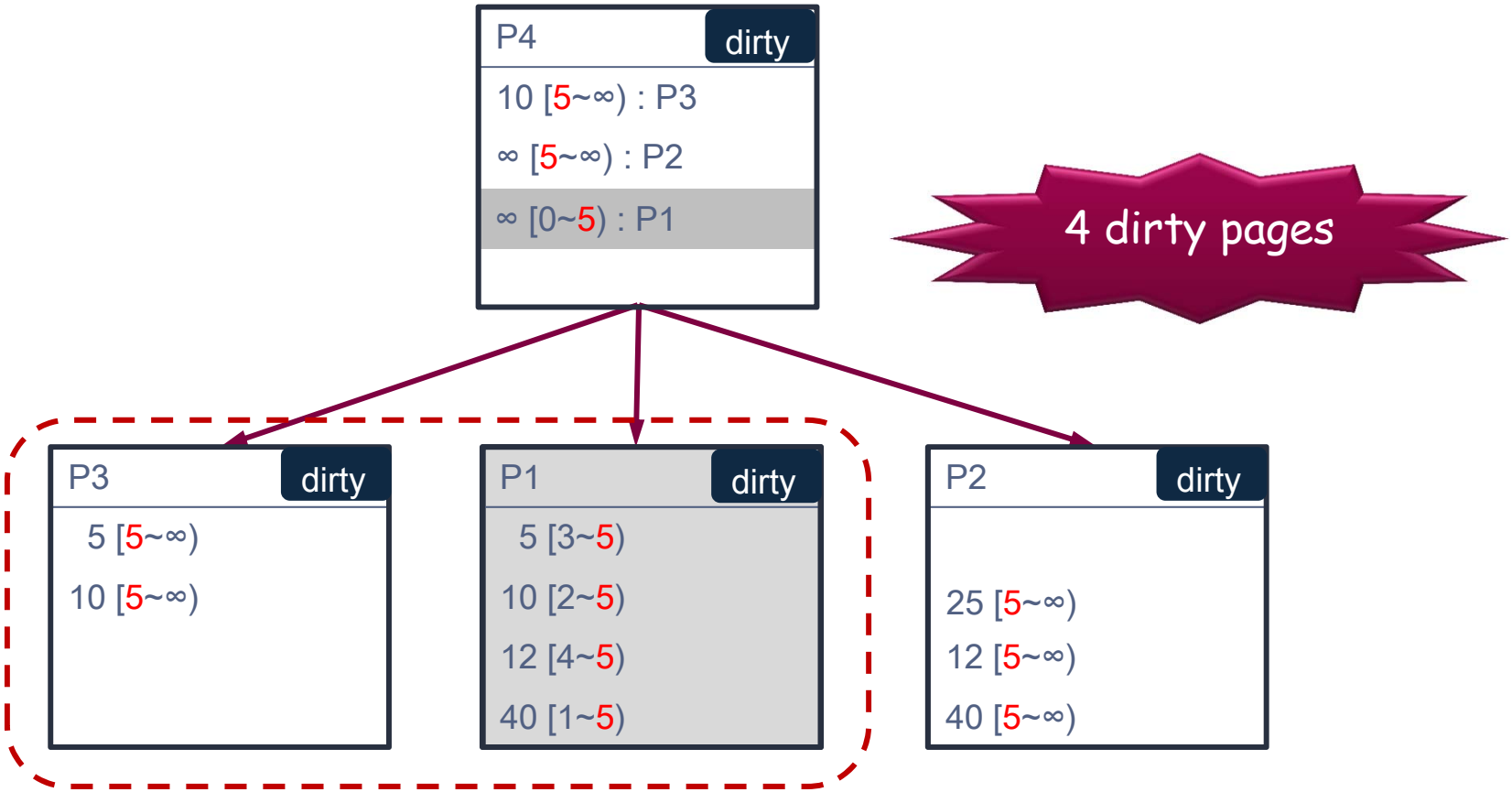
Lazy Split Multi-Version B-Tree (LS-MVBT)

- Optimizations

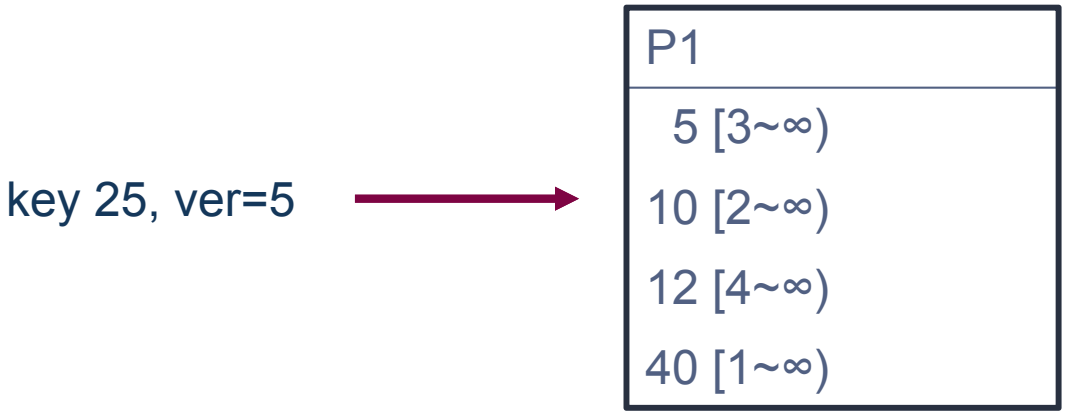


Optimization1: Lazy Split

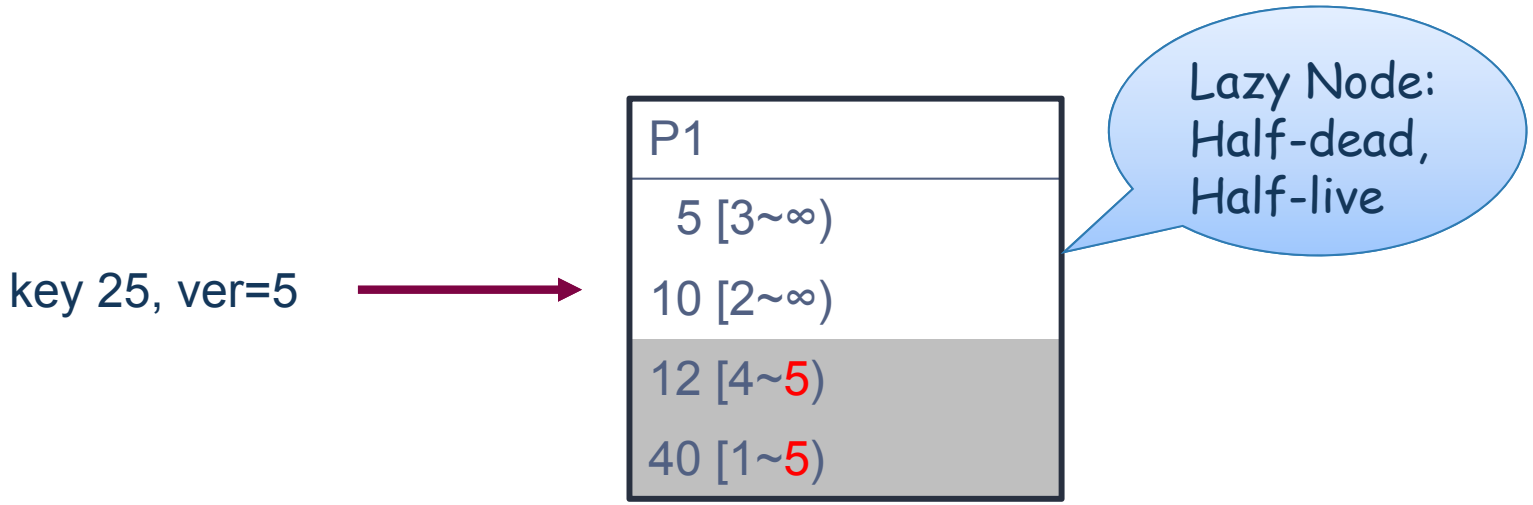
- Legacy Split in MVBT



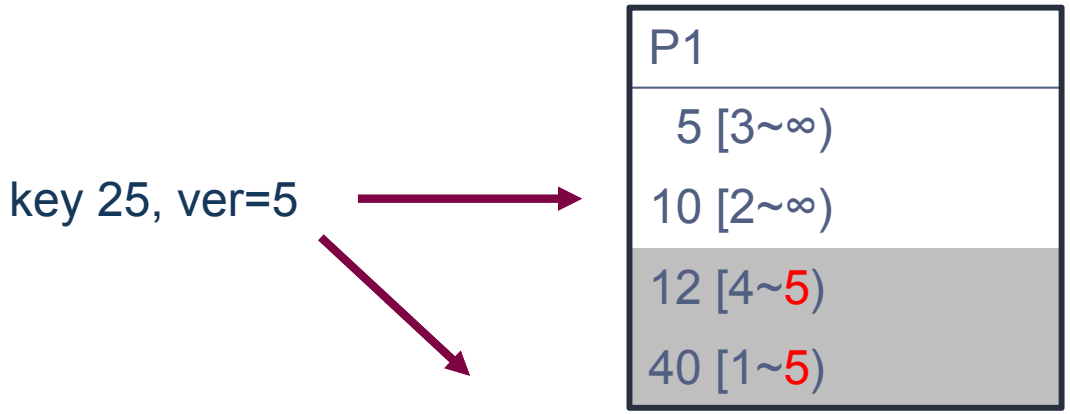
Optimization1: Lazy Split



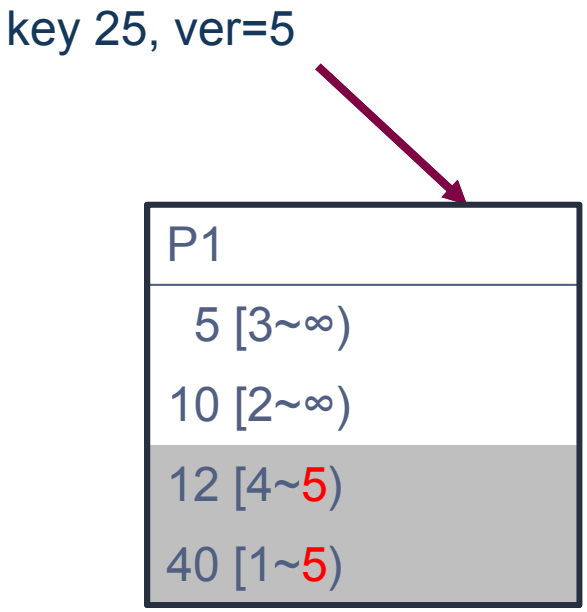
Optimization1: Lazy Split



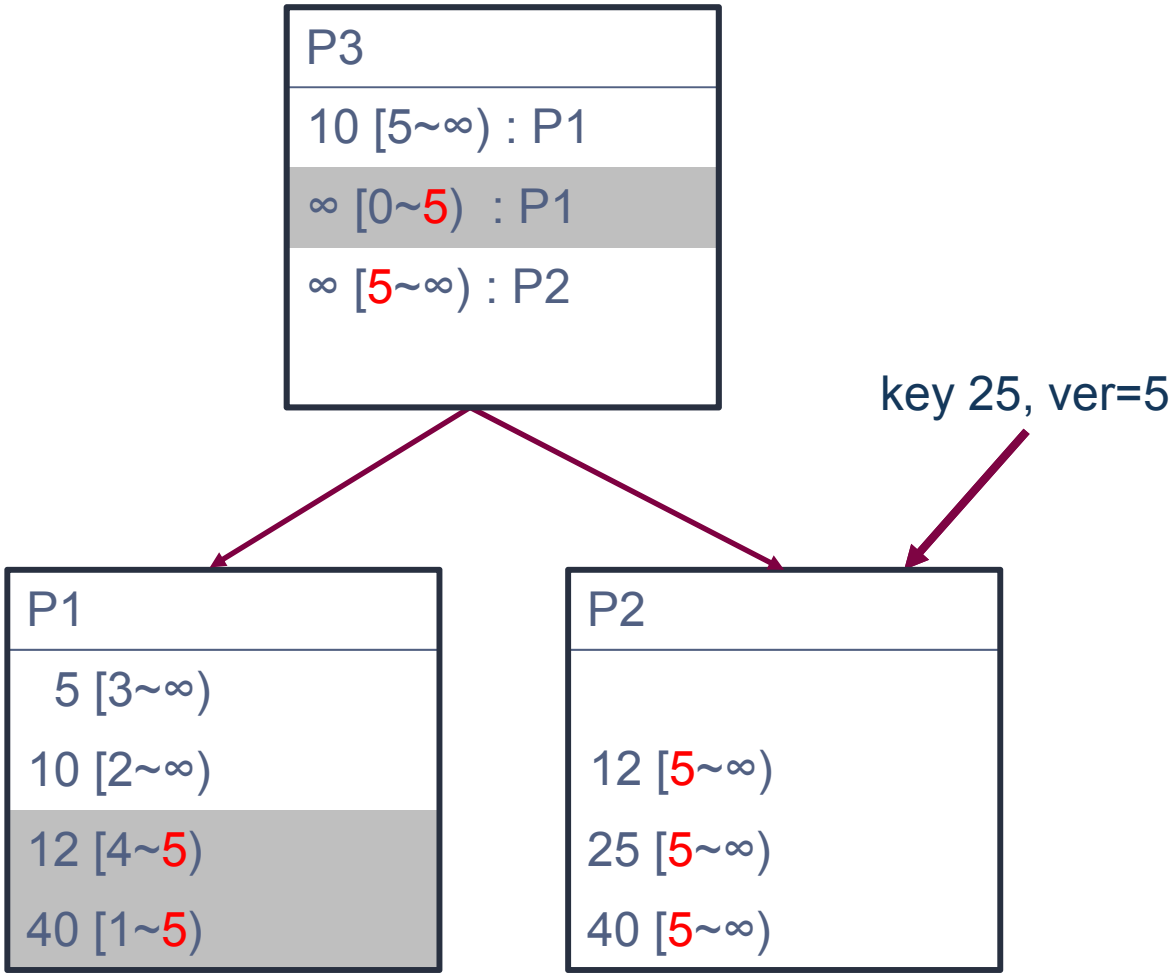
Optimization1: Lazy Split



Optimization1: Lazy Split

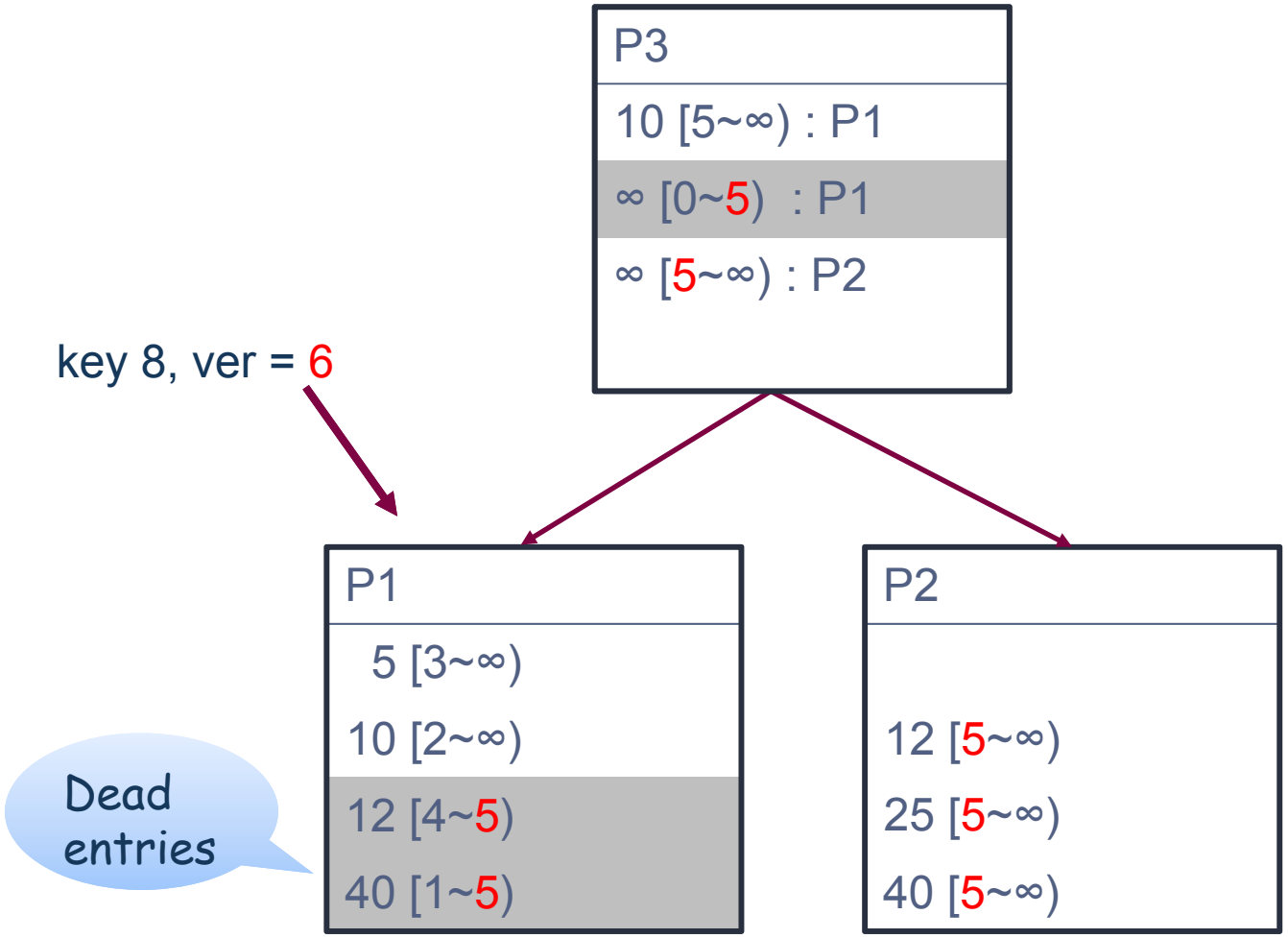


Optimization1: Lazy Split



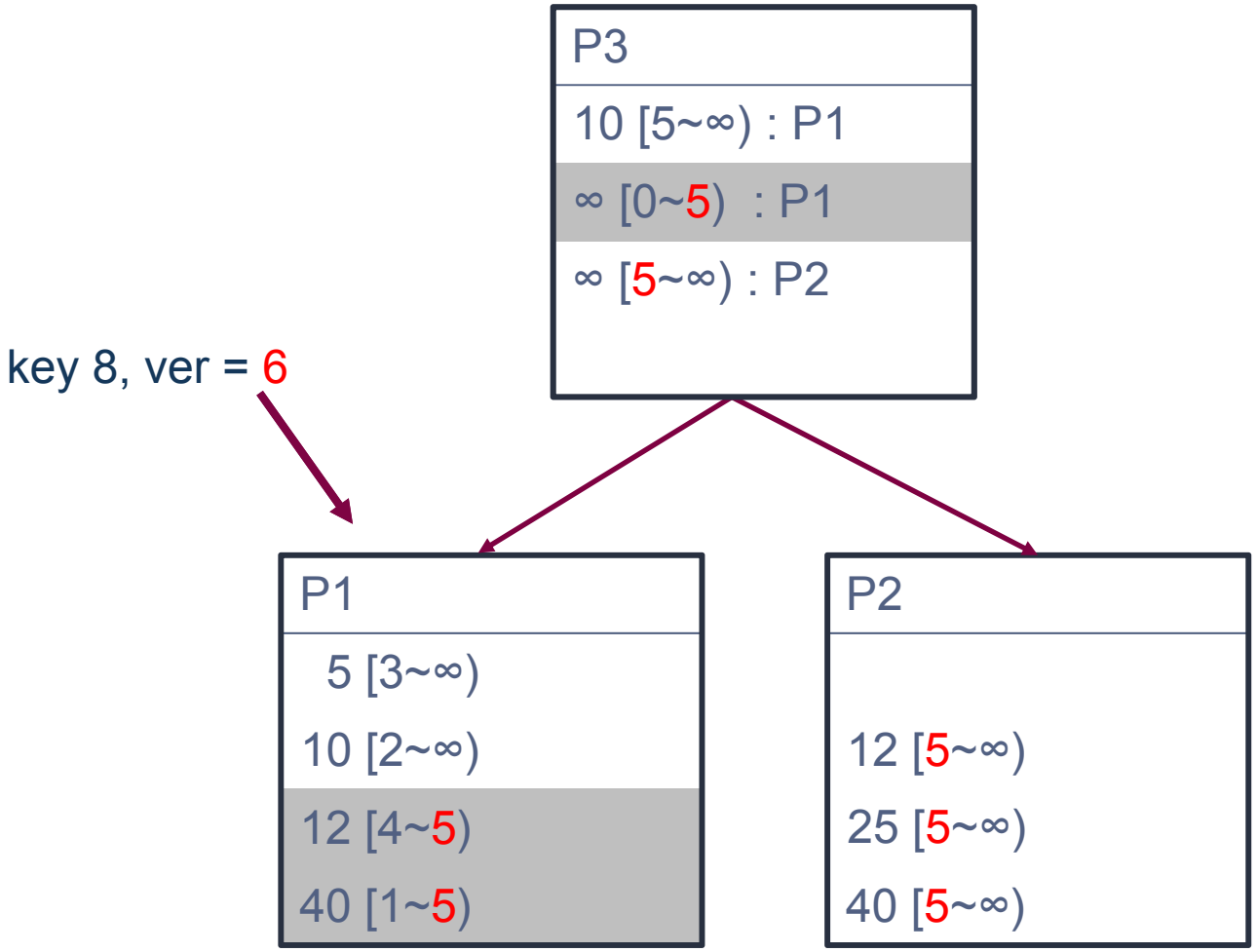
Optimization1: Lazy Split

- Lazy Node Overflow



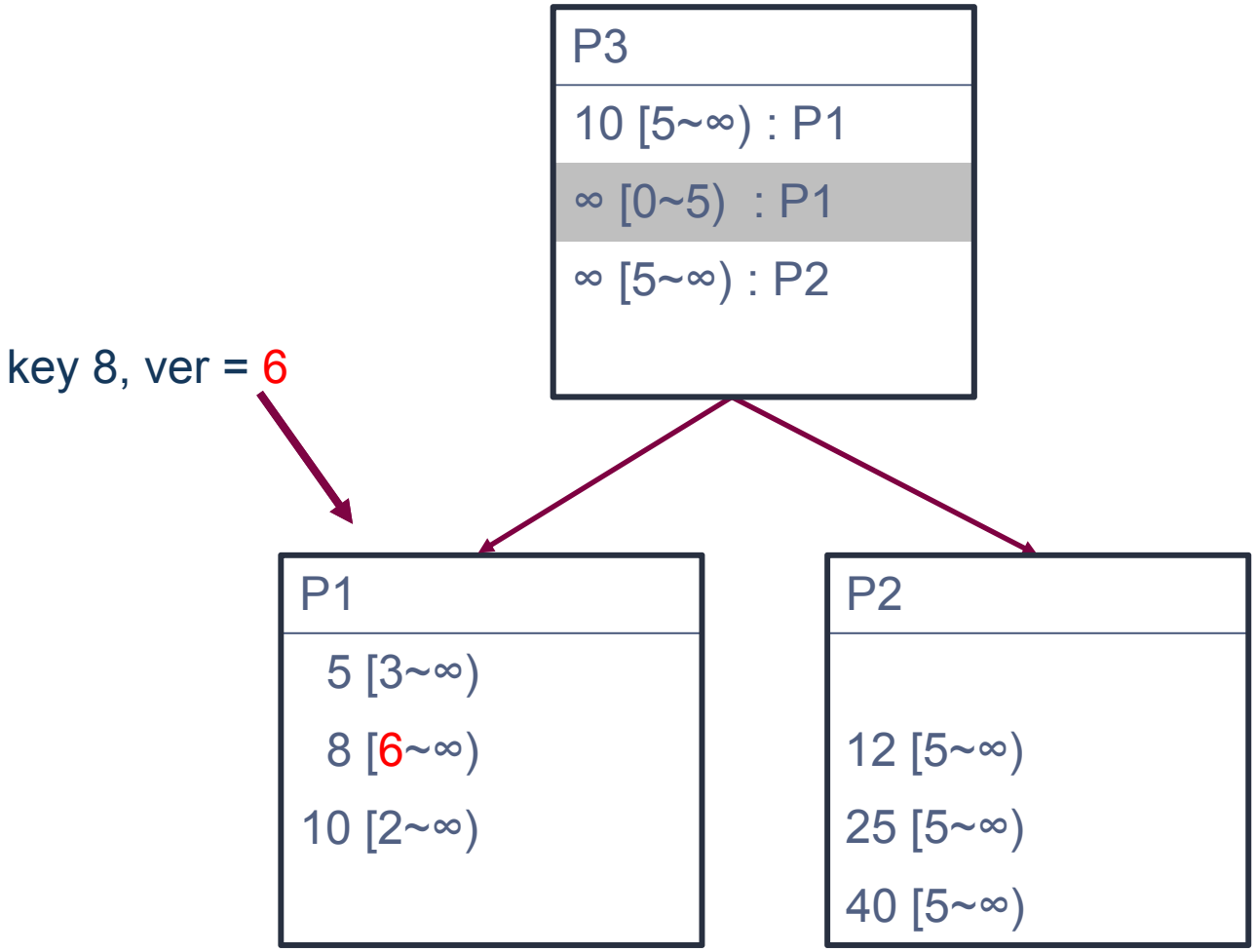
Optimization1: Lazy Split

- Lazy Node Overflow → Garbage collect dead entries



Optimization1: Lazy Split

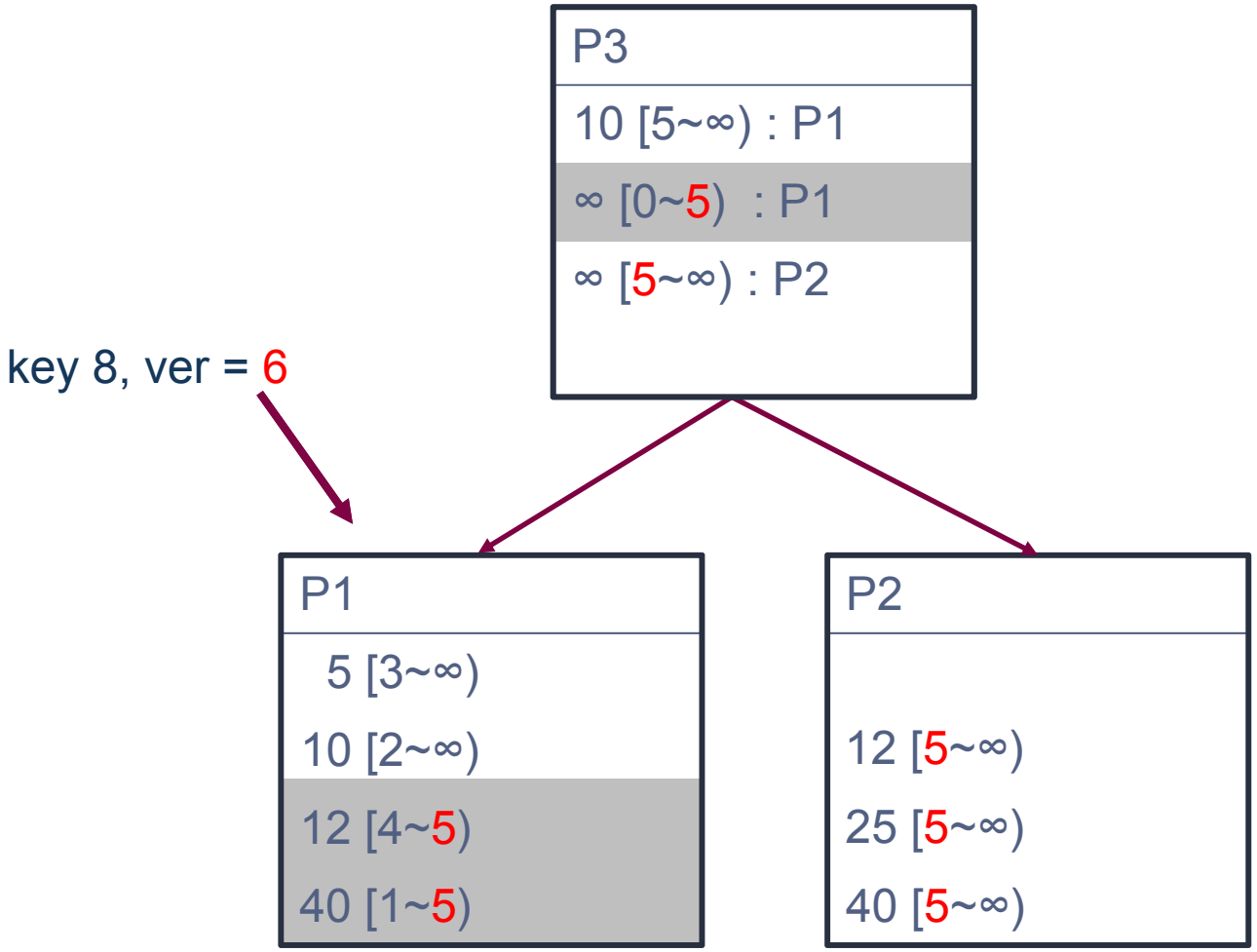
- Lazy Node Overflow → Garbage collect dead entries



But, what if the dead entries are being accessed by other transactions?

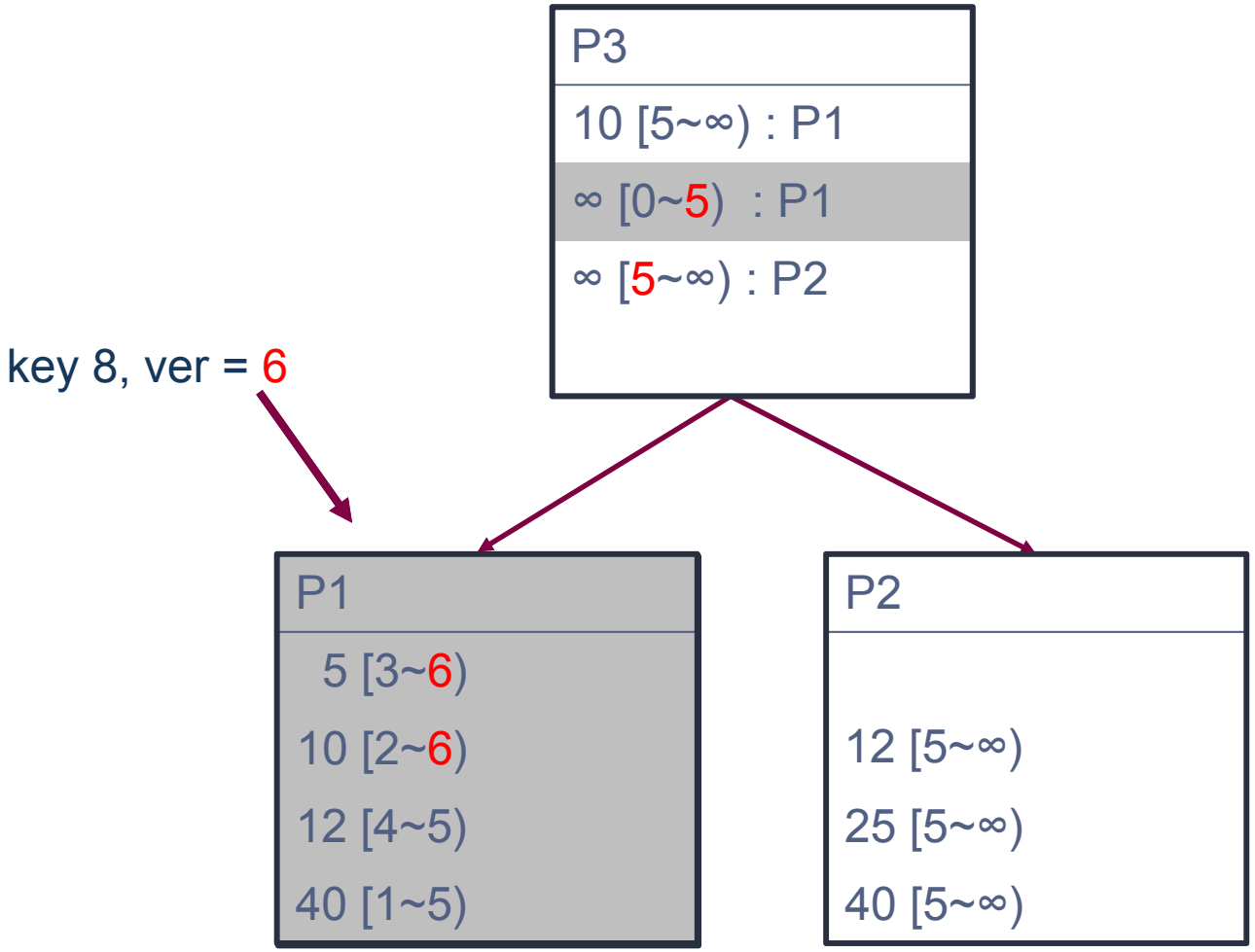
Optimization2: Reserved Buffer Space

- Option 1. Wait for read transactions to finish



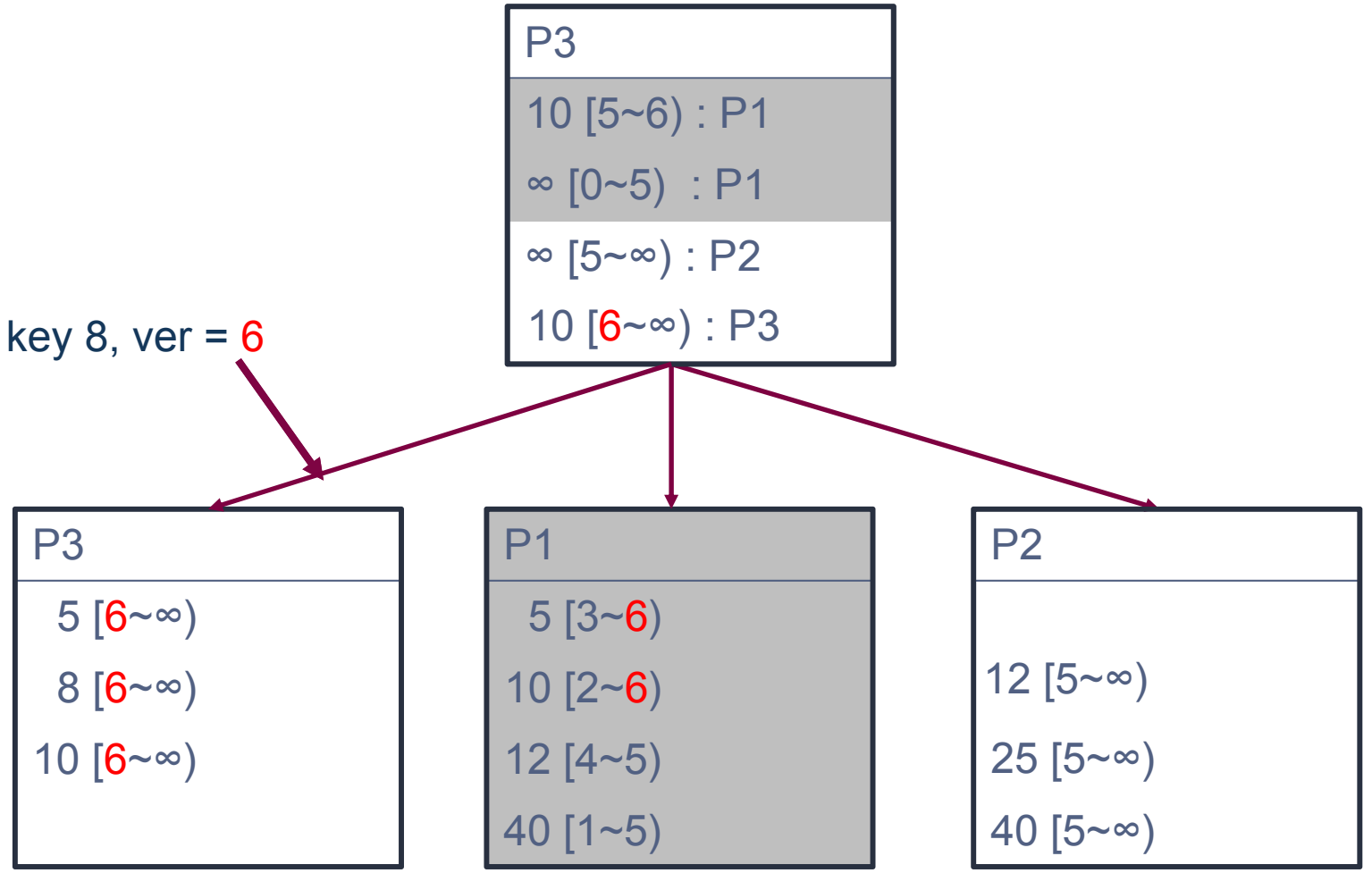
Optimization2: Reserved Buffer Space

- Option 2. Split as in legacy MVBT split



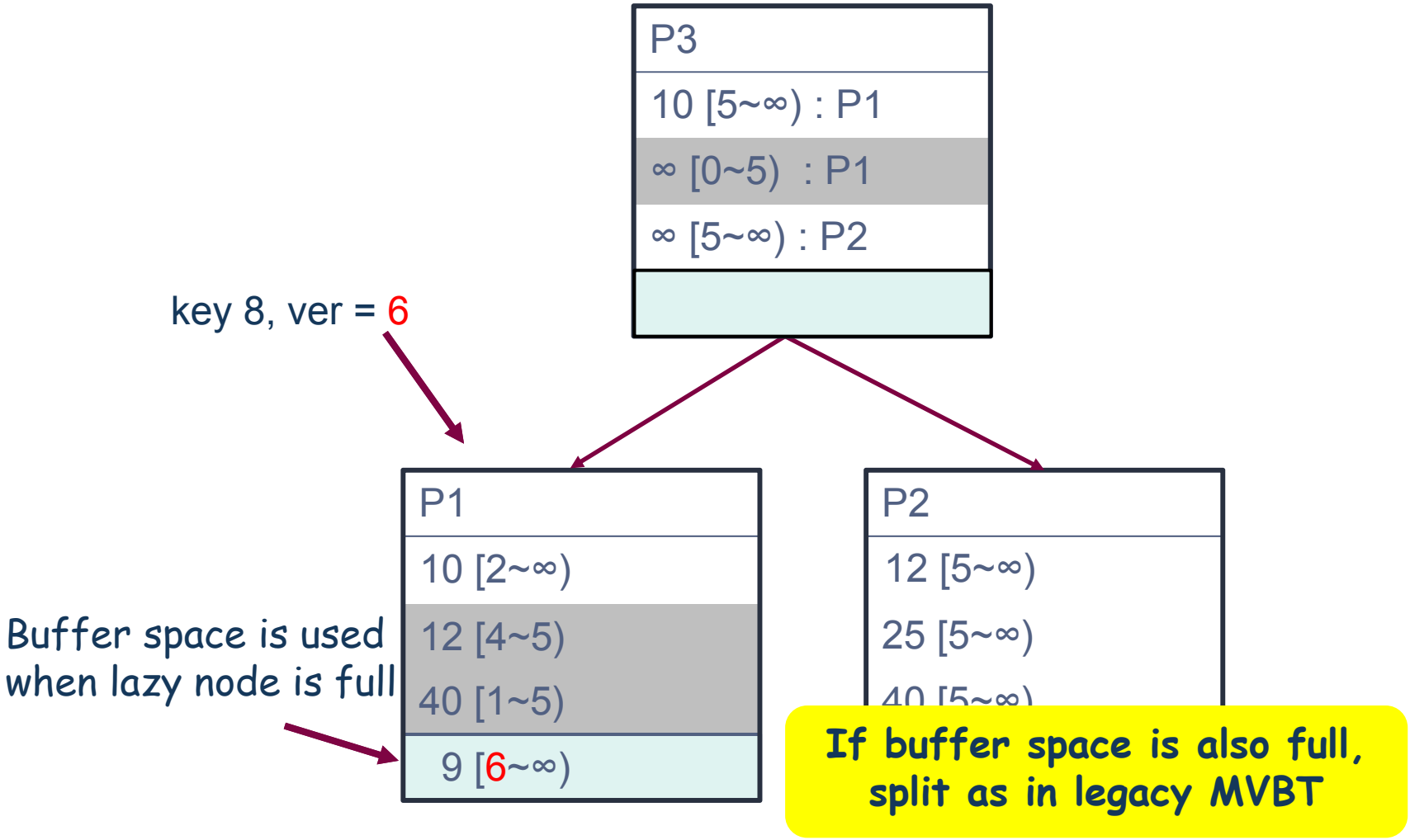
Optimization2: Reserved Buffer Space

- Option 2. Split as in legacy MVBT split



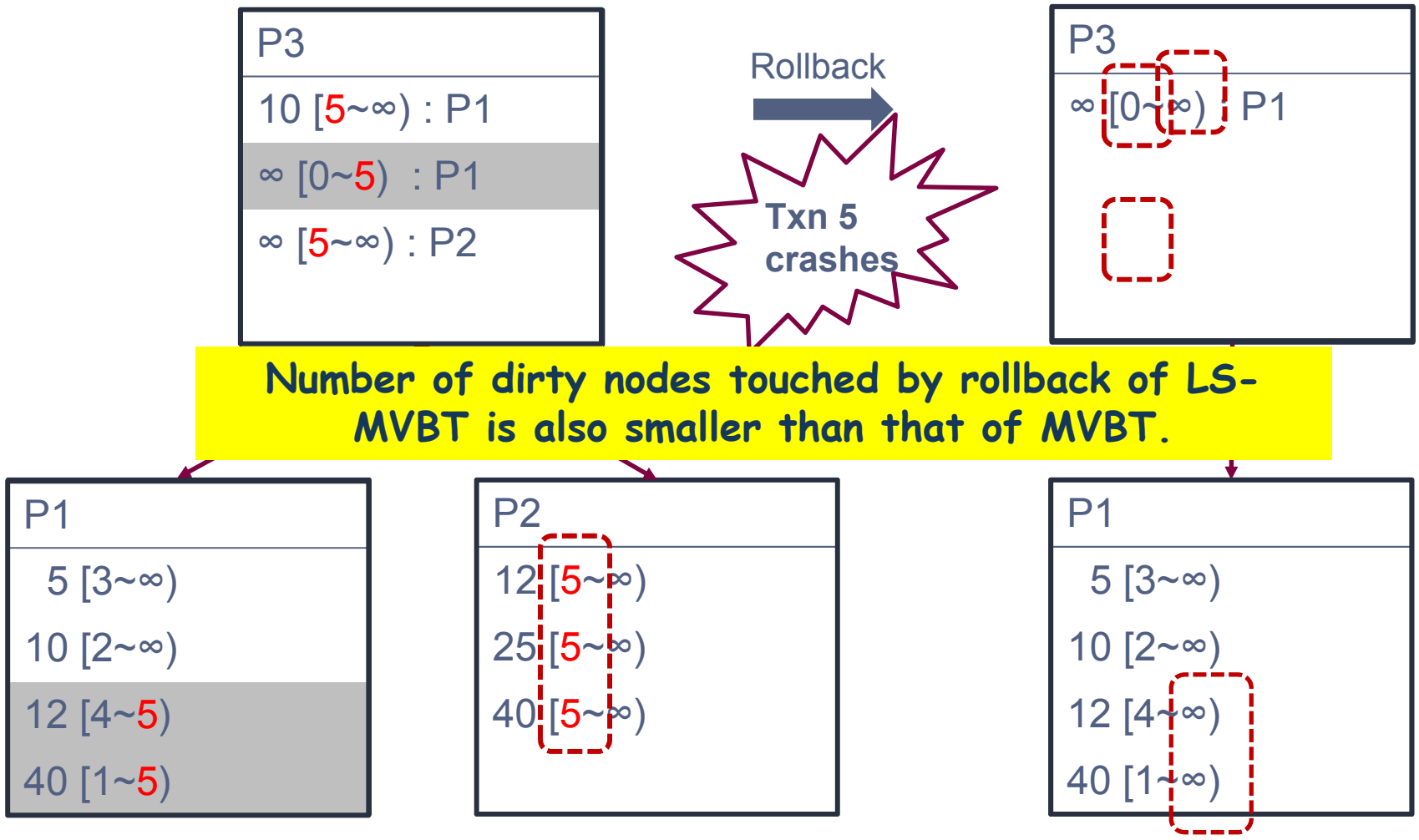
Optimization2: Reserved Buffer Space

- Option 3. Pad some buffer space in tree nodes



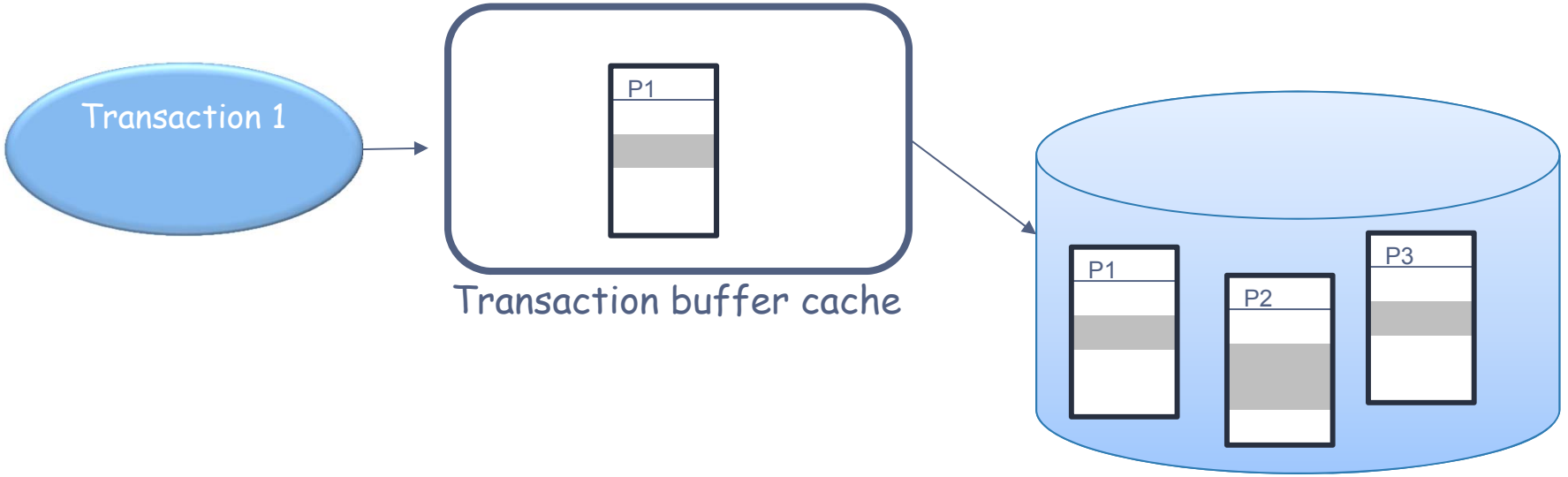
Rollback in LS-MVBT

- Similar to rollback in MVBT



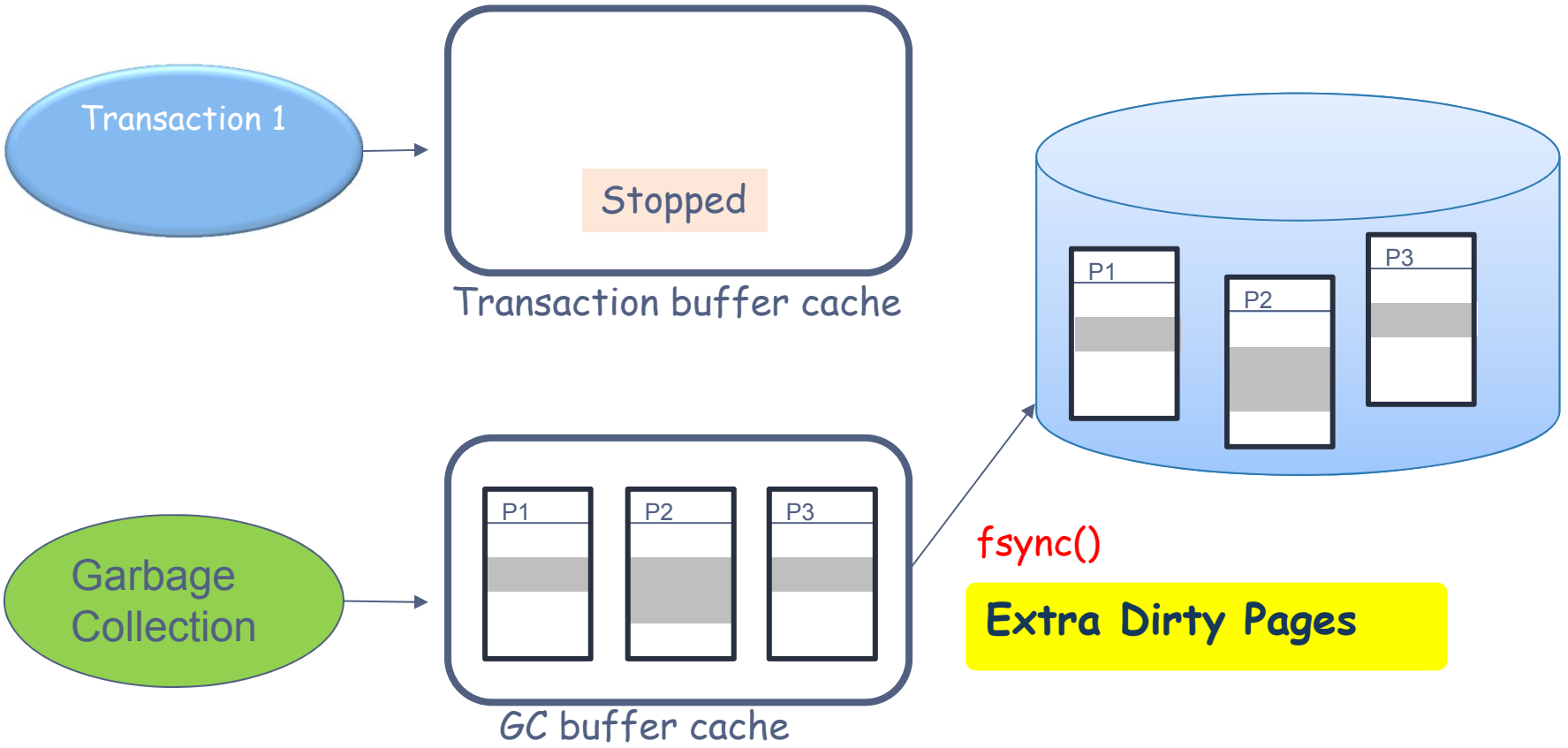
Optimization3: Lazy Garbage Collection

- Periodic Garbage Collection



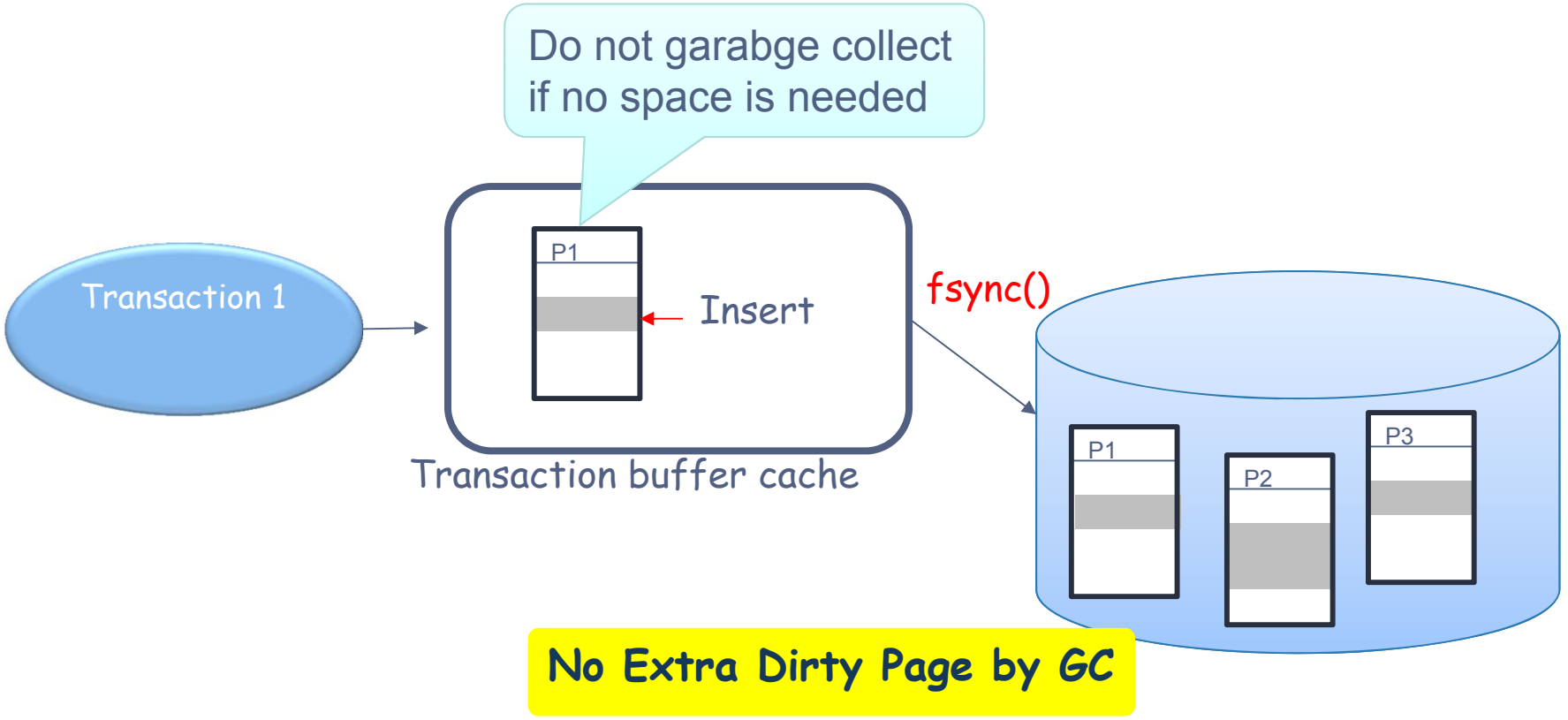
Optimization3: Lazy Garbage Collection

- Periodic Garbage Collection



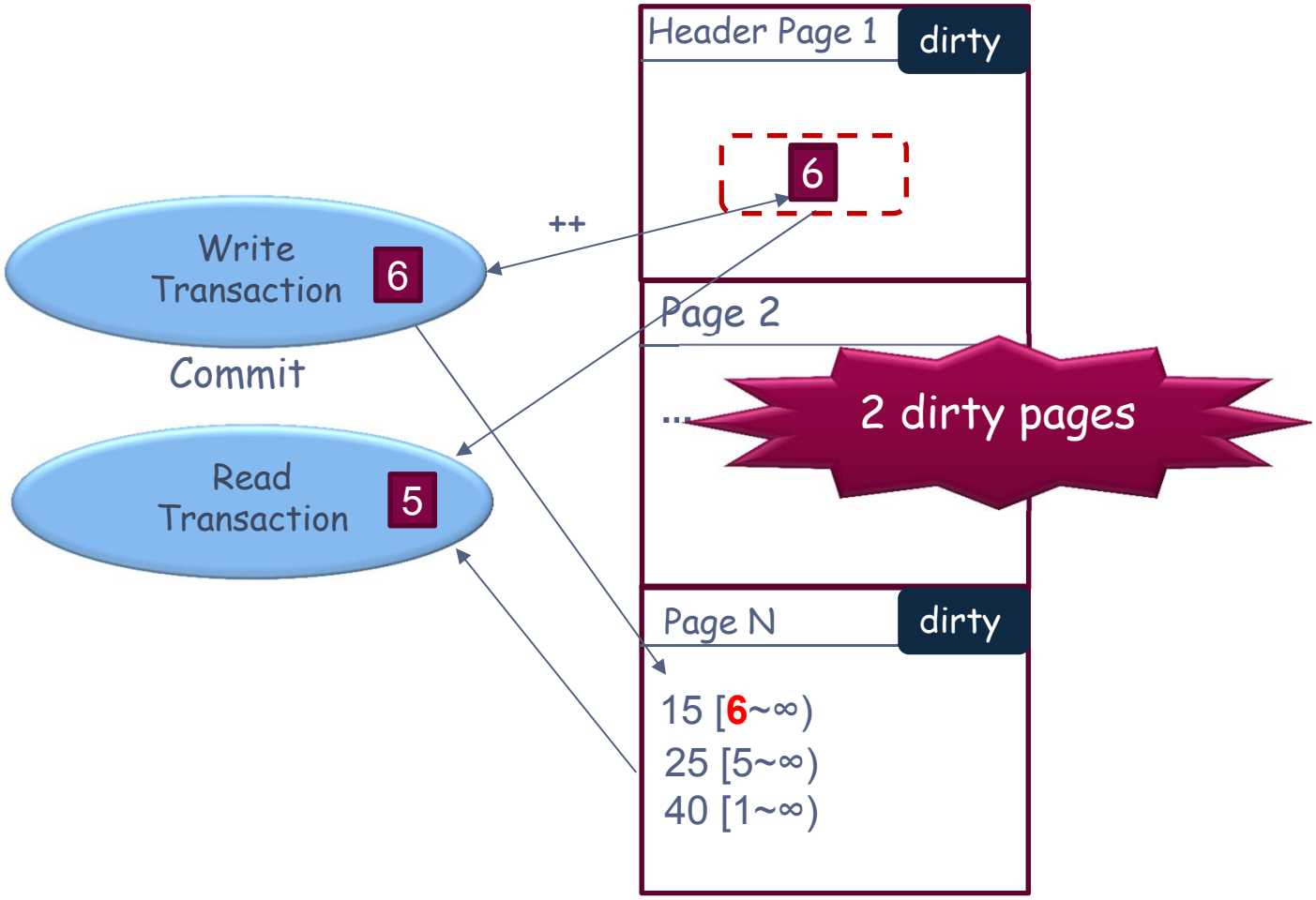
Optimization3: Lazy Garbage Collection

- Lazy Garbage Collection



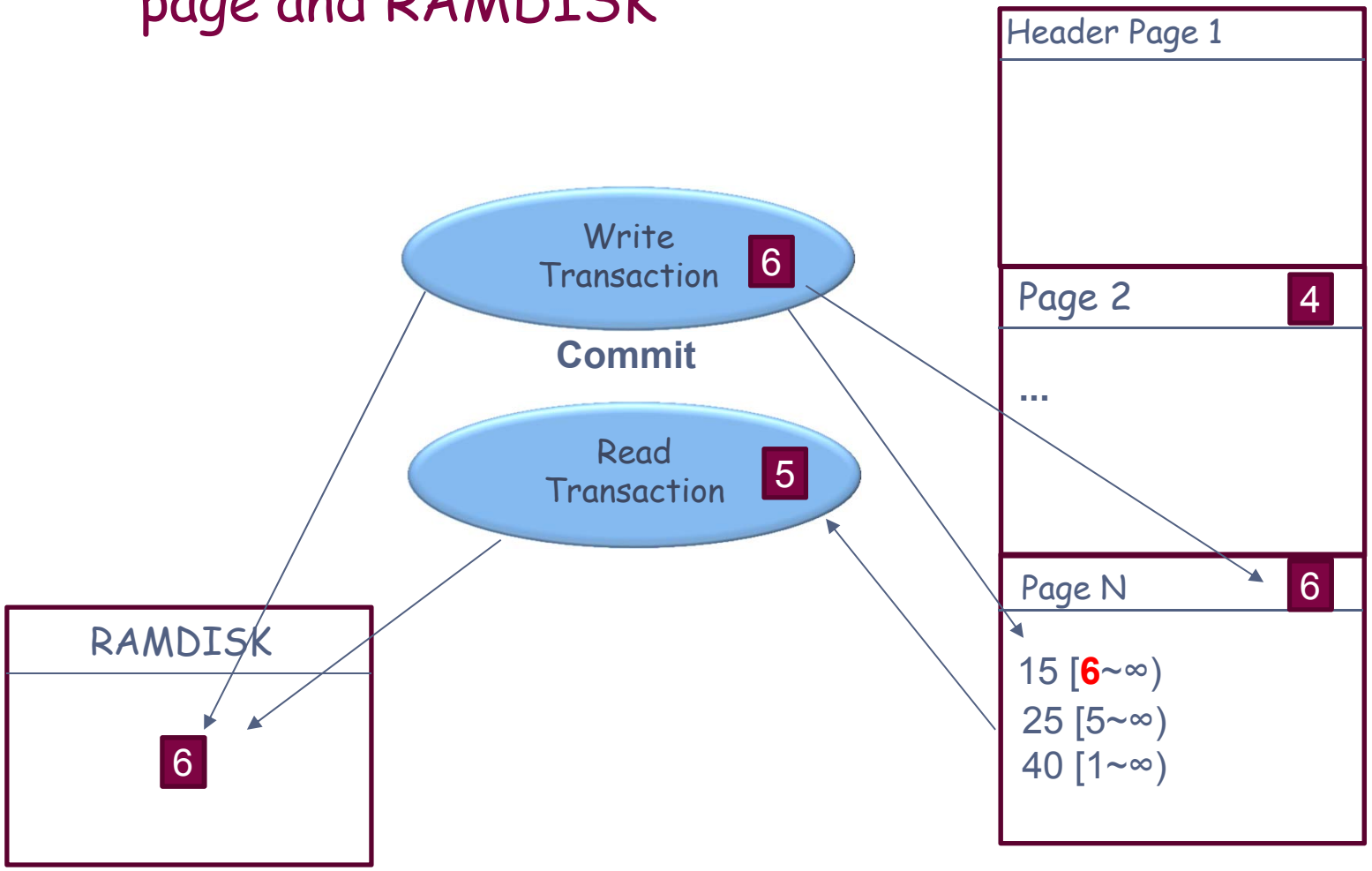
Optimization4: Metadata embedding

- Version = "File Change Counter" in DB header page



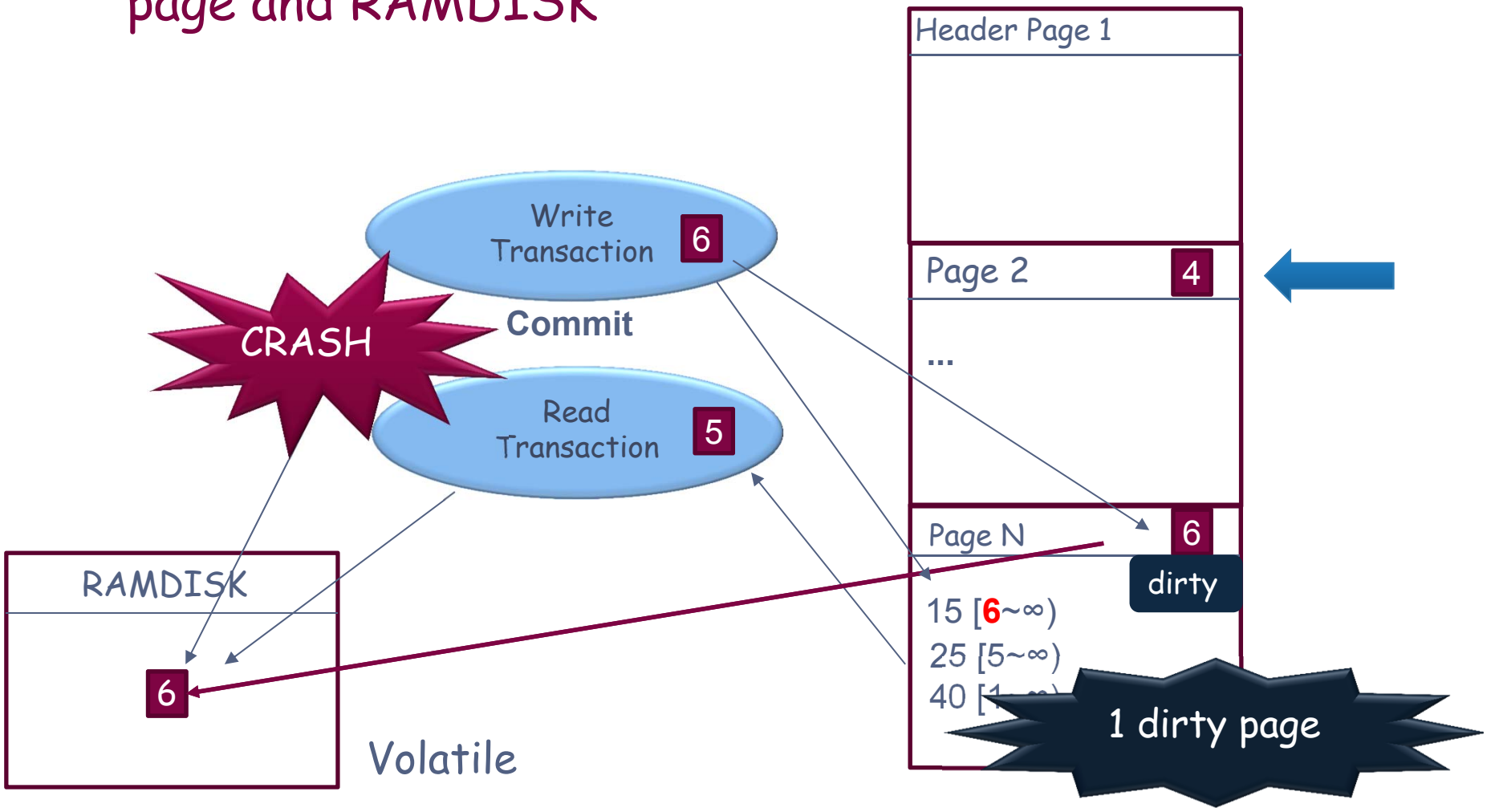
Optimization4: Metadata embedding

- Flush "File Change Counter" to the last modified page and RAMDISK



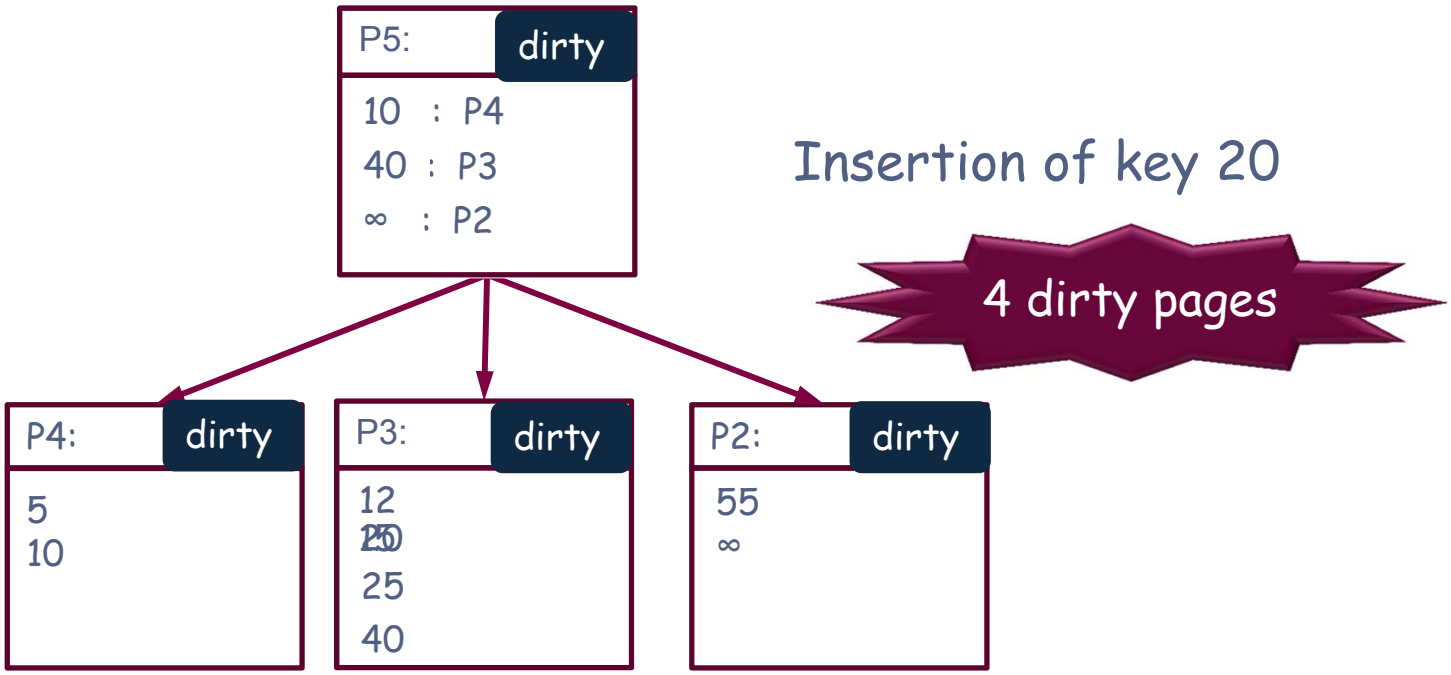
Optimization4: Metadata embedding

- Flush "File Change Counter" to the last modified page and RAMDISK



Optimization5: Disable Sibling Redistribution

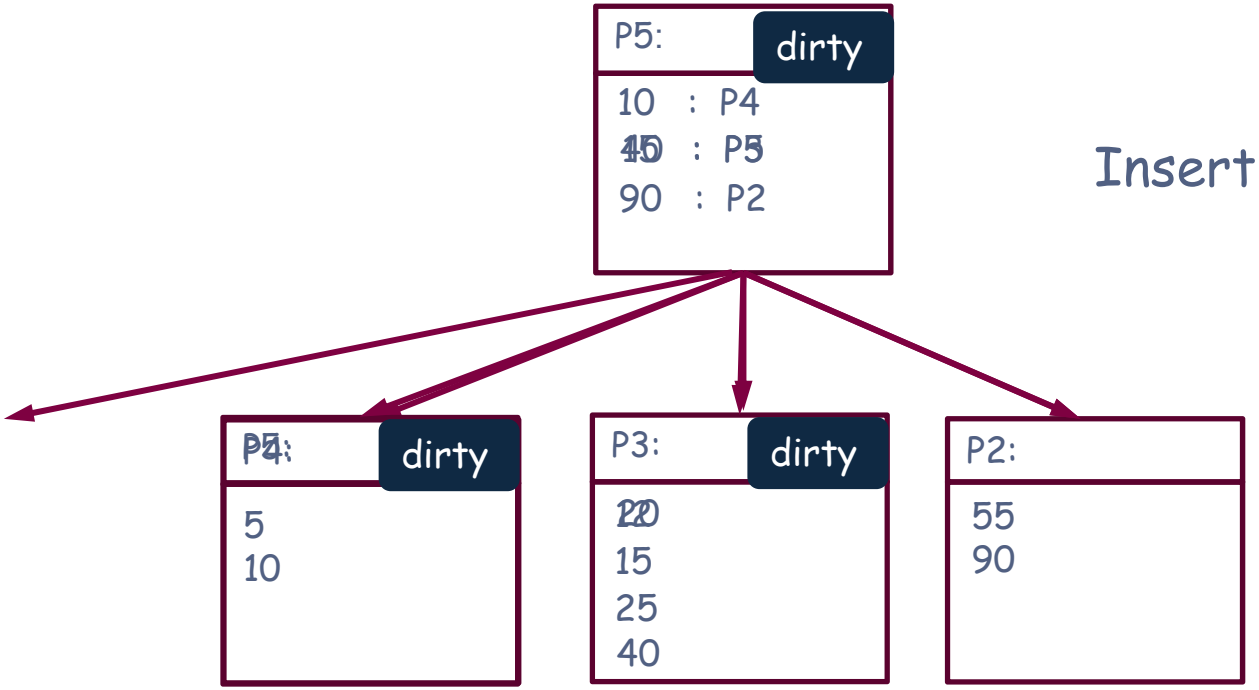
- Sibling redistribution hurts insertion performance



- Disable sibling redistribution → avoid dirtying 4 pages

Optimization5: Disable Sibling Redistribution

- Disabled sibling redistribution

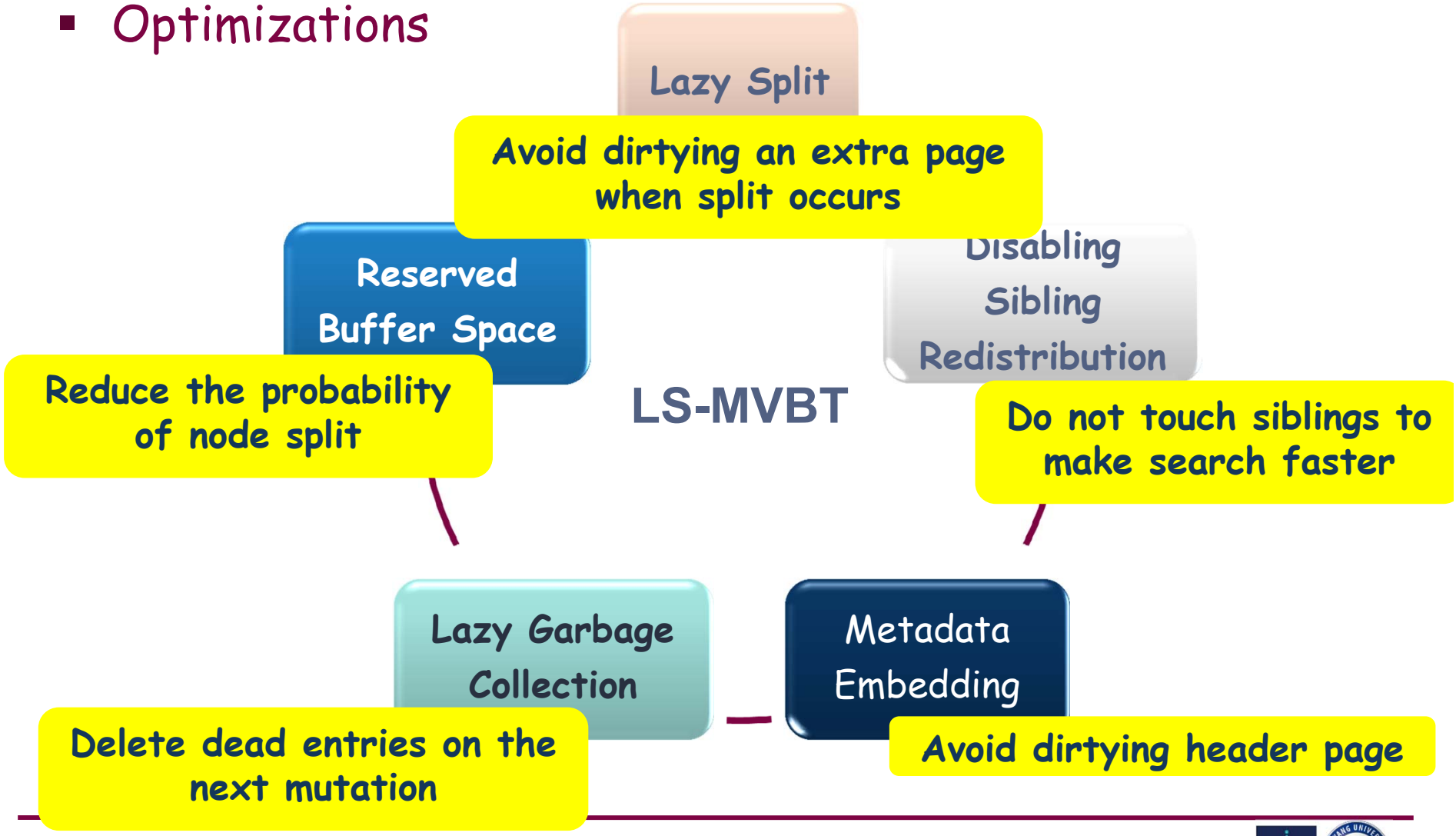


Insertion of key 20

3 dirty pages

Summary

- Optimizations



Performance Evaluation

- Implemented LS-MVBT in SQLite 3.7.12
- Testbed: Samsung Galaxy-S4 (JellyBean 4.3)
  - Exynos 5 Octa 5410 Quad Core 1.6GHz
  - 2GB Memory
  - 16GB eMMC

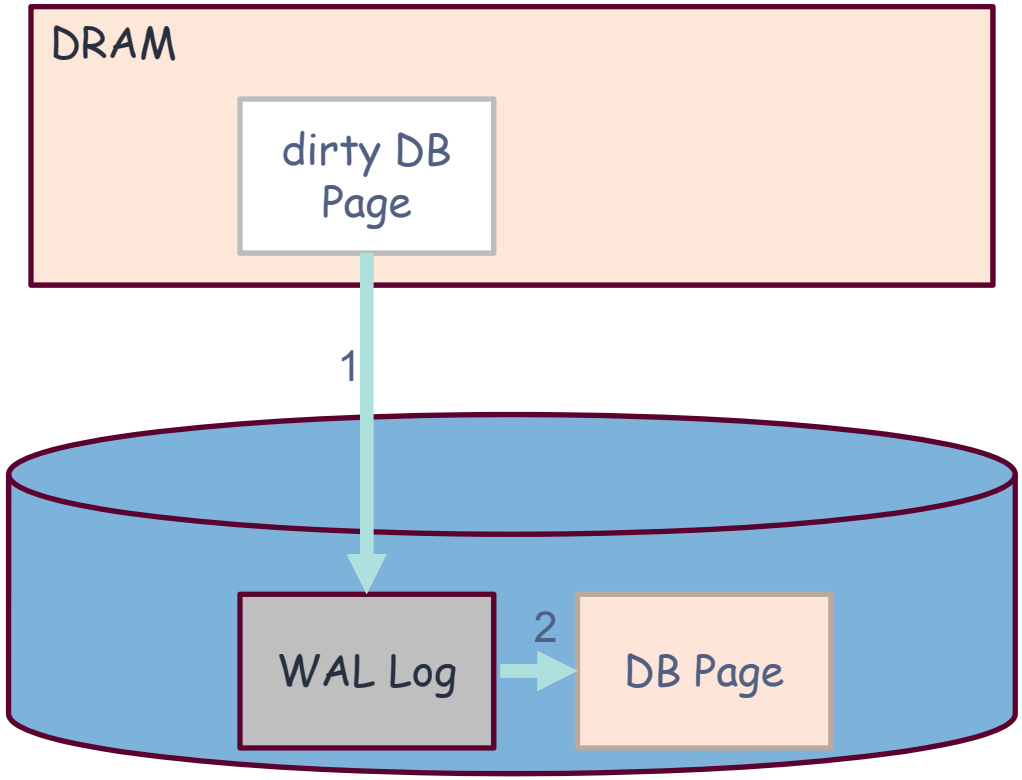
Samsung GALAXY S4

- Performance Analysis Tools
  - Mobibench
  - MOST

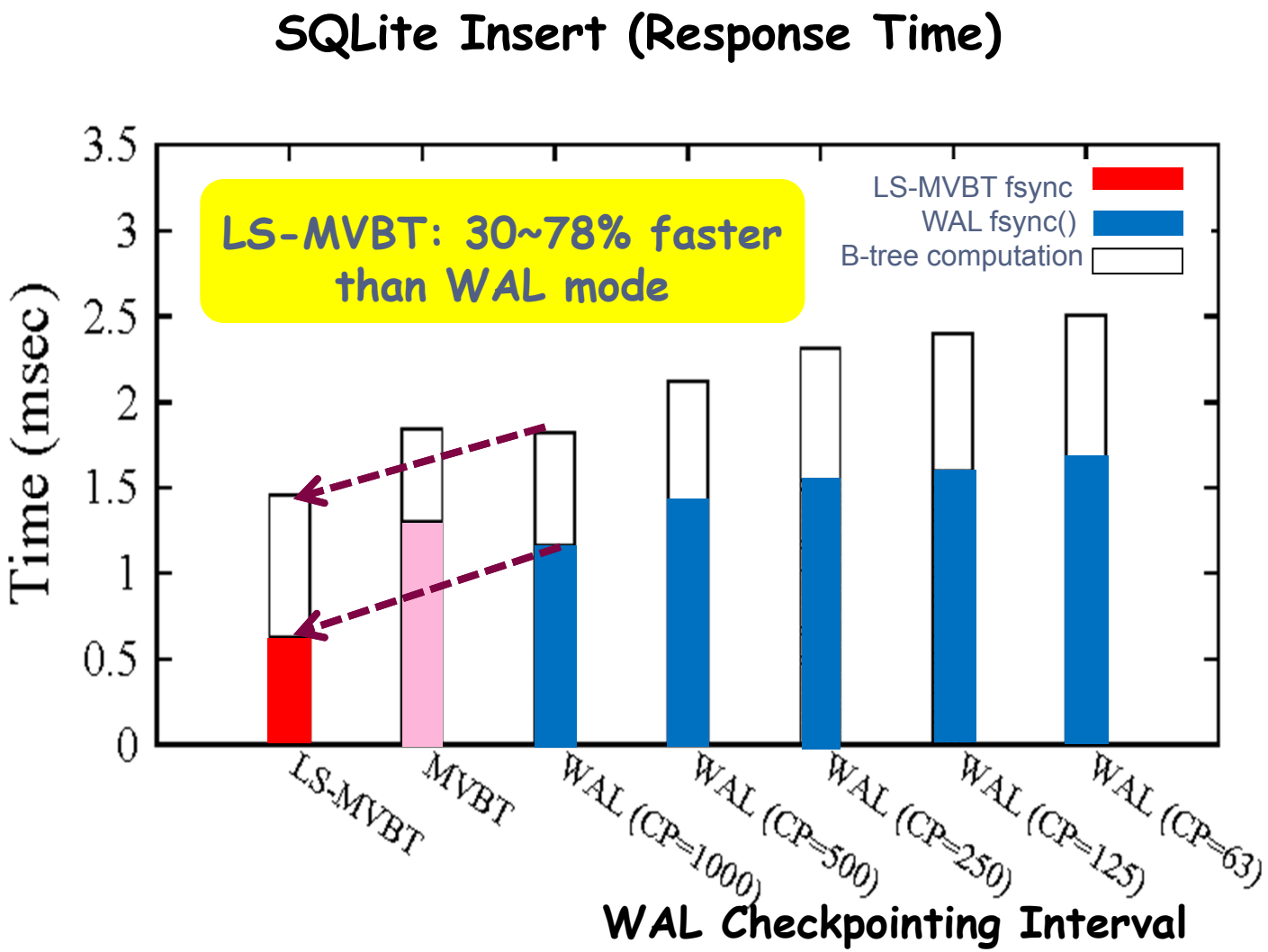
android

WAL (Write-Ahead-Logging) mode

- Default mode in Jelly Bean and KitKat
- Commit -> Write to a log file
- Checkpointing -> Flush logs to DB pages

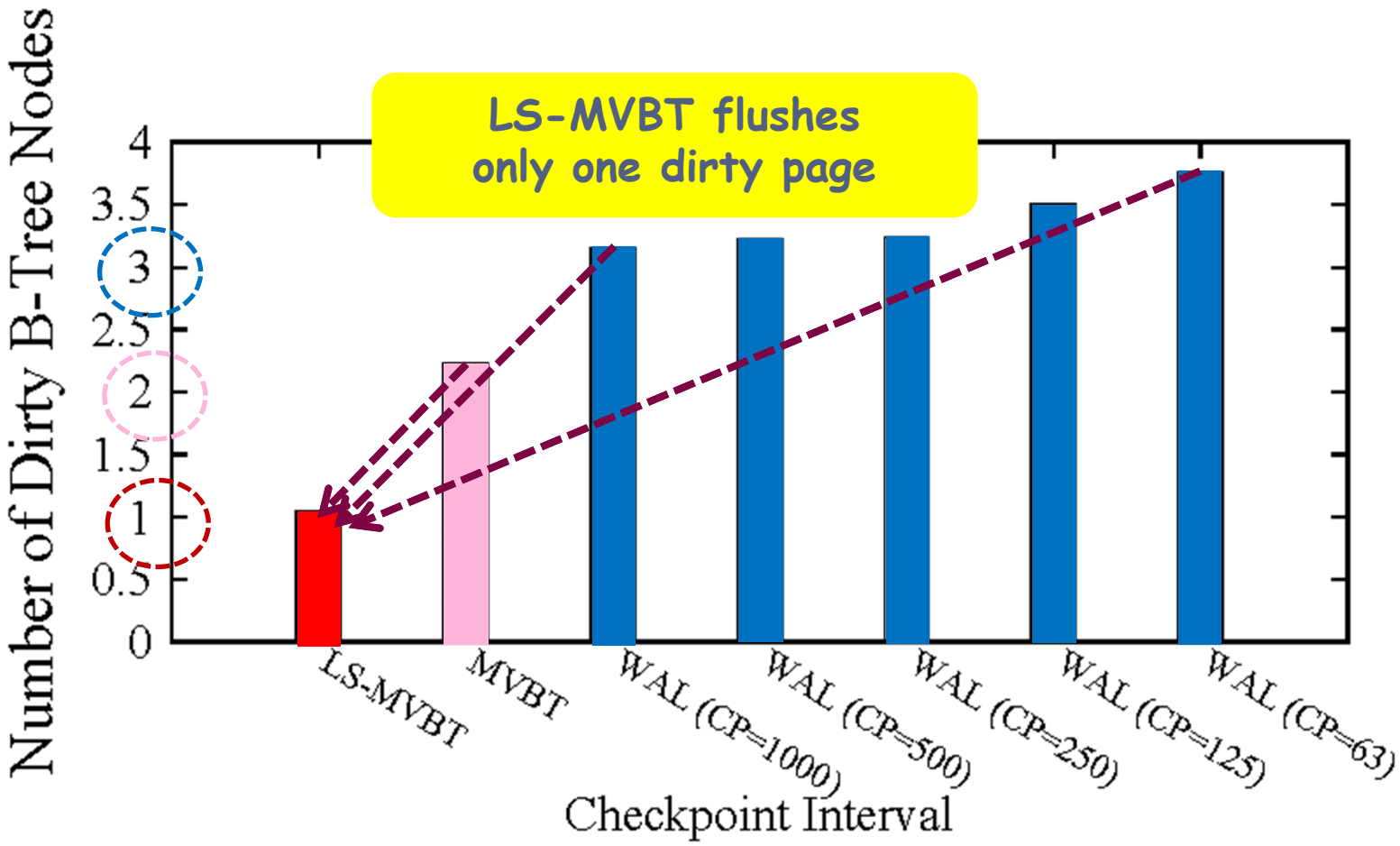


Analysis of insert Performance



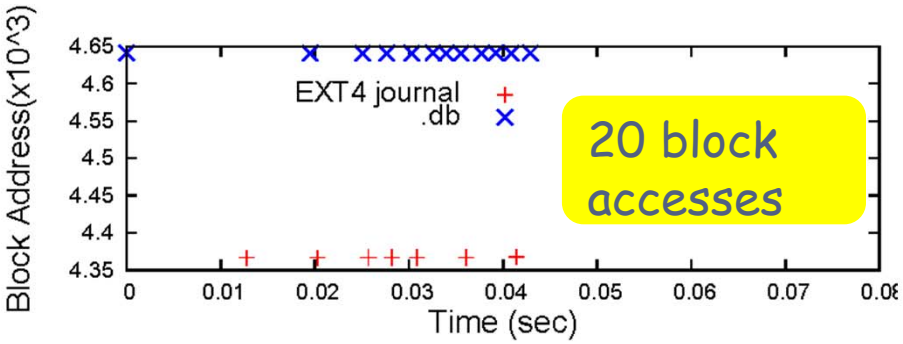
Analysis of insert Performance

SQLite Insert (# of Dirty Pages per Insert)

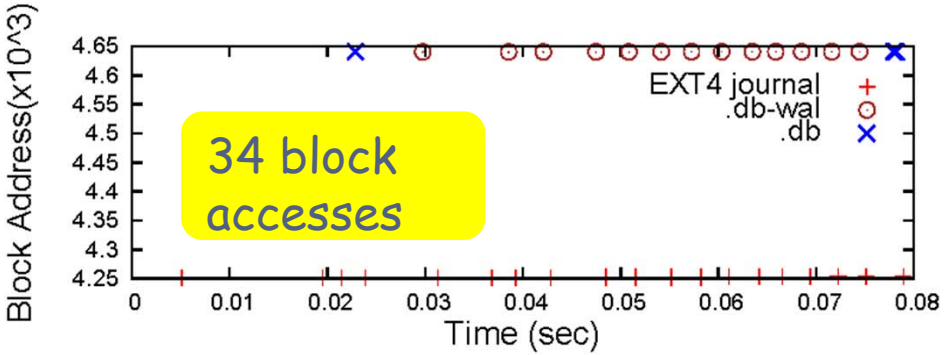


Analysis of Block I/O Behavior

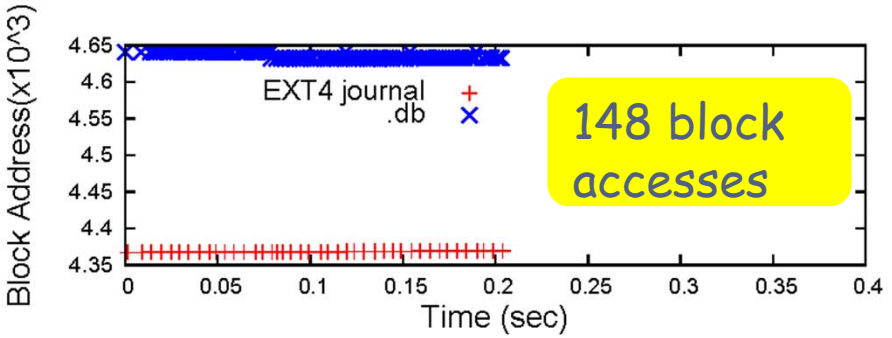
SQLite Insert (# of Accesses to Block Device)



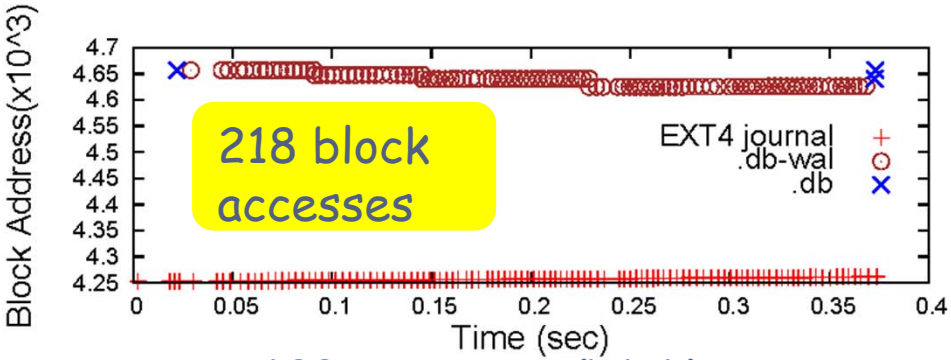
10 insertions (LS-MVBT)



10 insertions (WAL)



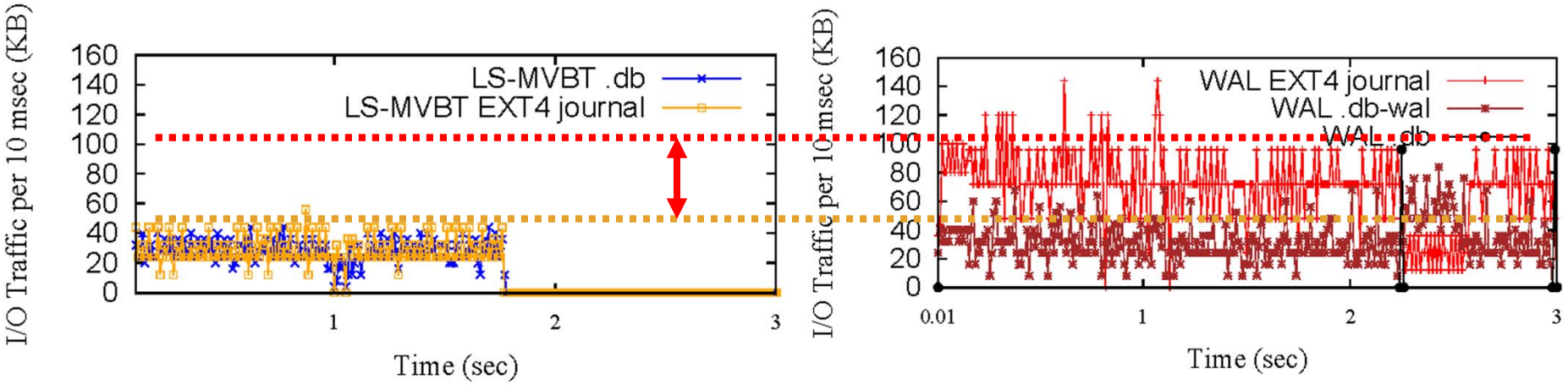
100 insertions (LS-MVBT)



100 insertions (WAL)

I/O Traffic at Block Device Level

SQLite Insert (I/O Traffic per 10 msec)



1000 insertions (LS-MVBT)

1000 insertions (WAL)

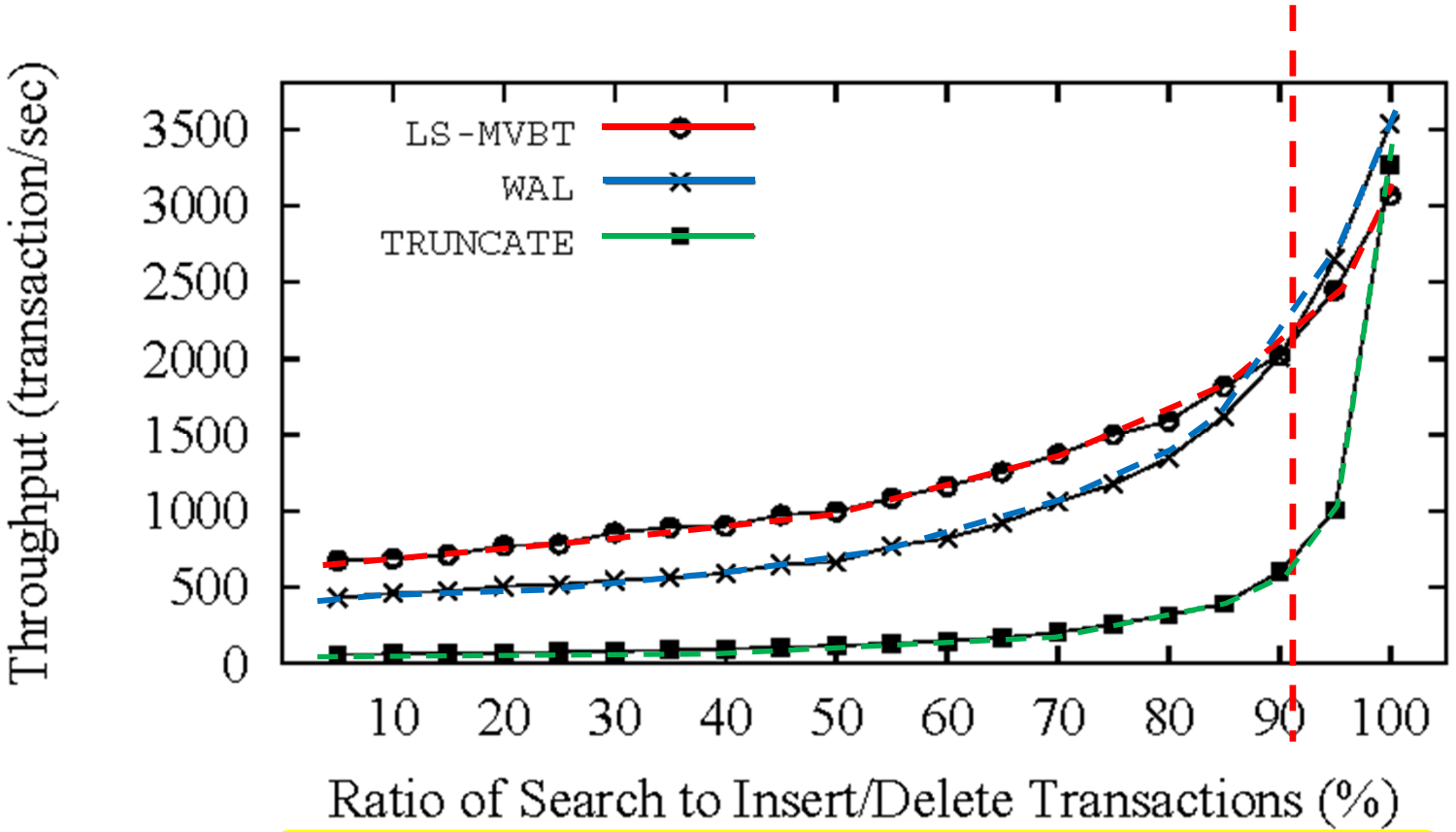
LS-MVBT saves 67% I/O traffic:
9.9 MB (LS-MVBT) vs 31 MB (WAL)

x3 longer life time of NAND flash

- How about Search performance?
  - LS-MVBT is optimized for write performance.

Search Performance

Transaction Throughput with varying Search/Insert ratio

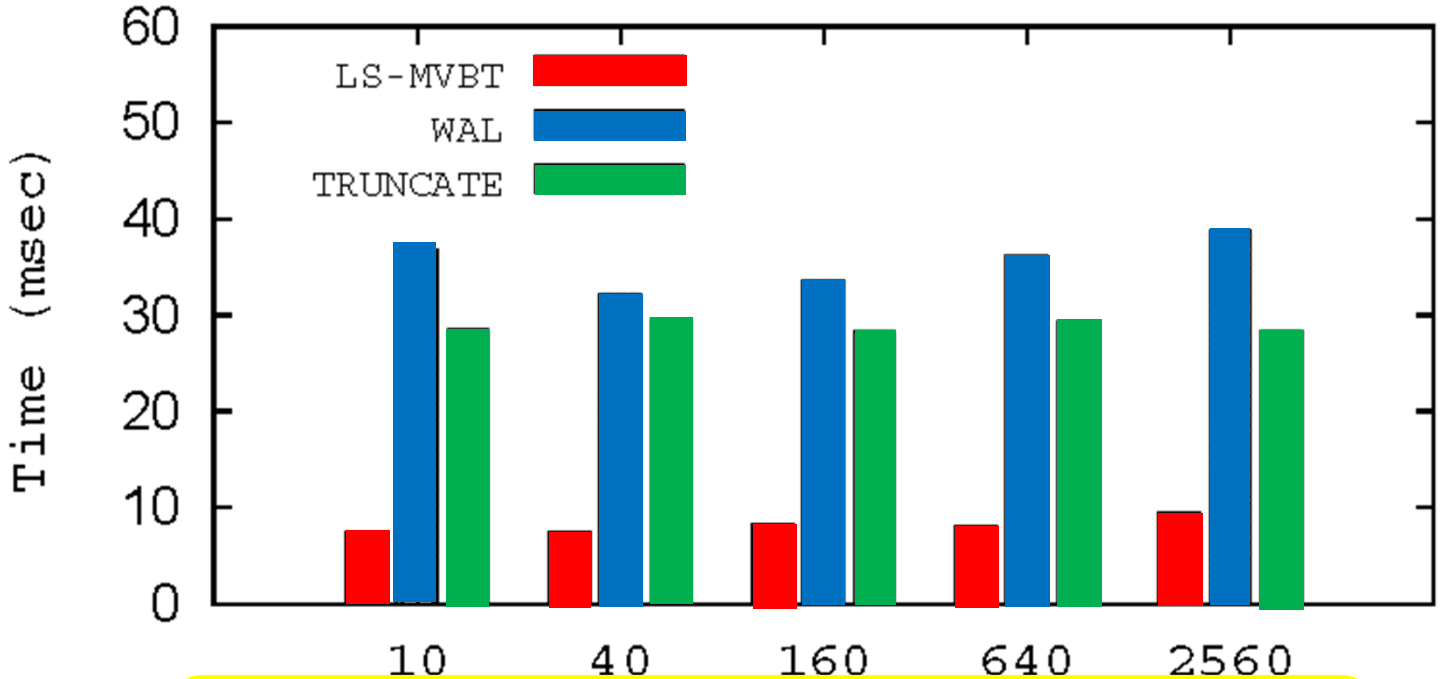


LS-MVBT wins unless more than 93% of transactions are search

- How about recovery latency?
  - WAL mode exhibits longer recovery latency due to log replay.
  - How about LS-MVBT?

Recovery

Recovery Time with varying # of insert operations to rollback

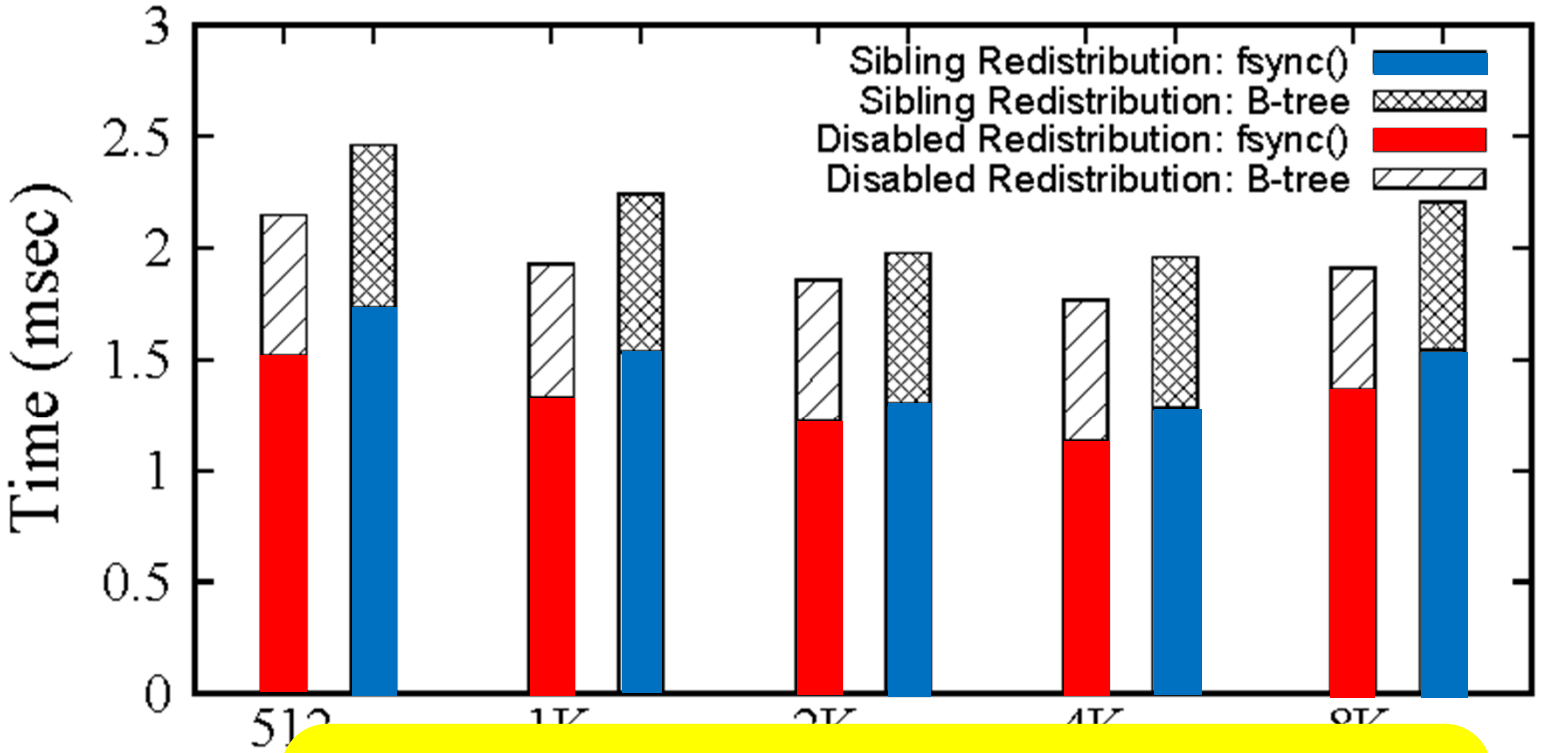


LS-MVBT is **x5~x6** faster than WAL.

- How much performance improvement for each optimization?

Effect of Disabling Sibling Redistribution

SQLite Insert: (Response Time and # of Dirty Pages)



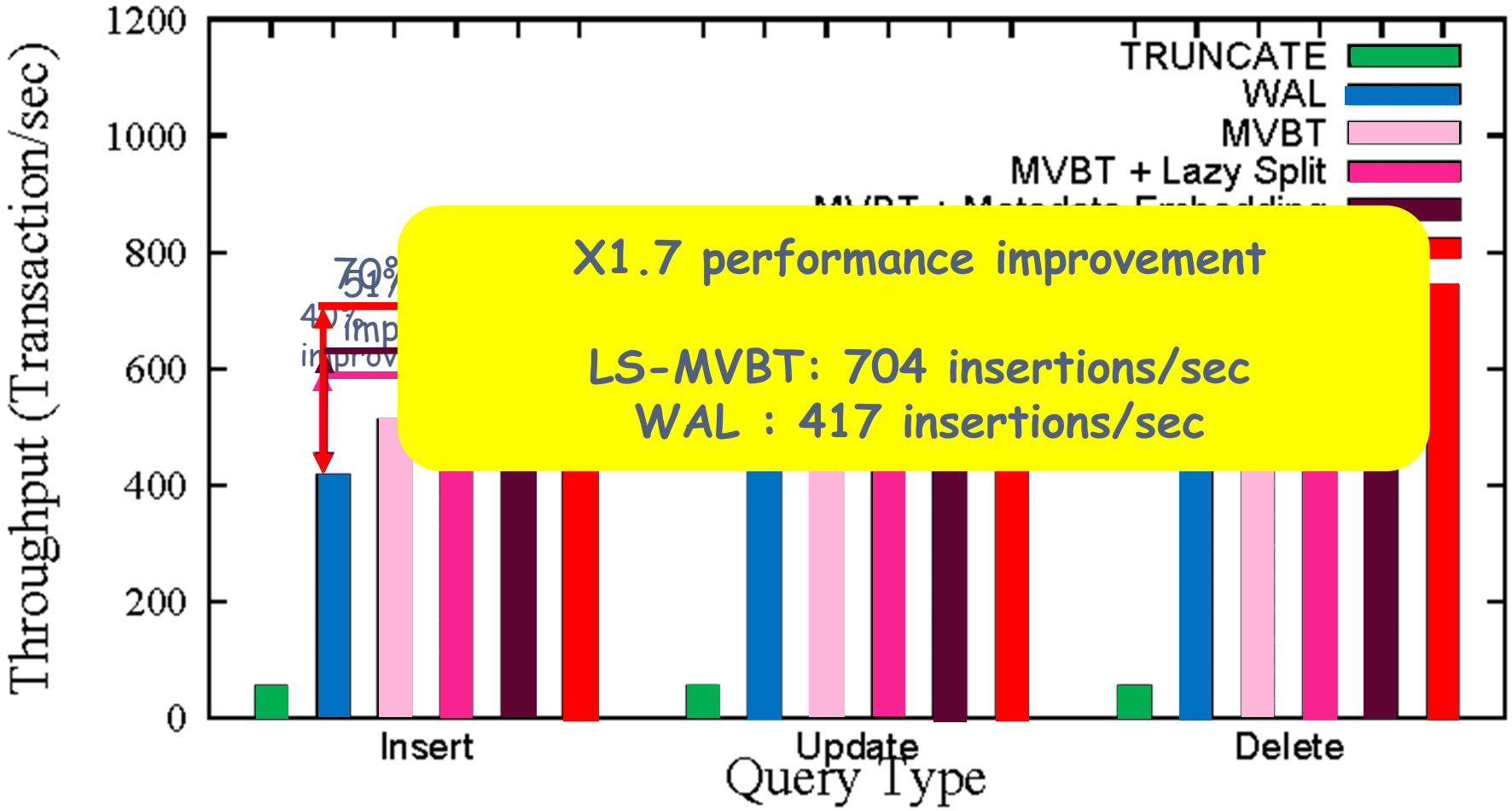
Disabling sibling redistribution

↓

20% faster insertion

Performance Quantification

Throughput of SQLite: Combining All Optimizations



Conclusion

- LS-MVBT resolves the Journaling of Journal anomaly via
  - Reducing the number of fsync()'s with **Version based B-tree**.
  - Reducing the overhead of single fsync() via lazy split, disabling sibling redistribution, metadata embedding, reserved buffer space, lazy garbage collection.

- LS-MVBT achieves 70% performance gain against WAL mode (417 ins/sec →704 ins/sec) solely via software optimization.

Future/Ongoing Works

1. Improve the performance of WAL mode
   - 3 dirty pages per insertion → not really necessary
   - Easier work than replacing the entire B-tree
   - Compatible with current SQLite database files
   - We expect the performance benefit would be similar to LS-MVBT

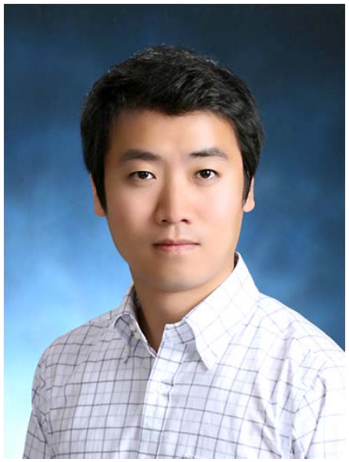
2. SQLite for Non-Volatile Memory
   - Logging/Journaling in NVRAM

3. SQLite for multi-cores

Credit

Woo-Hee Kim

Beomseok Nam



Dongil Park

Youjip Won

Thank you…