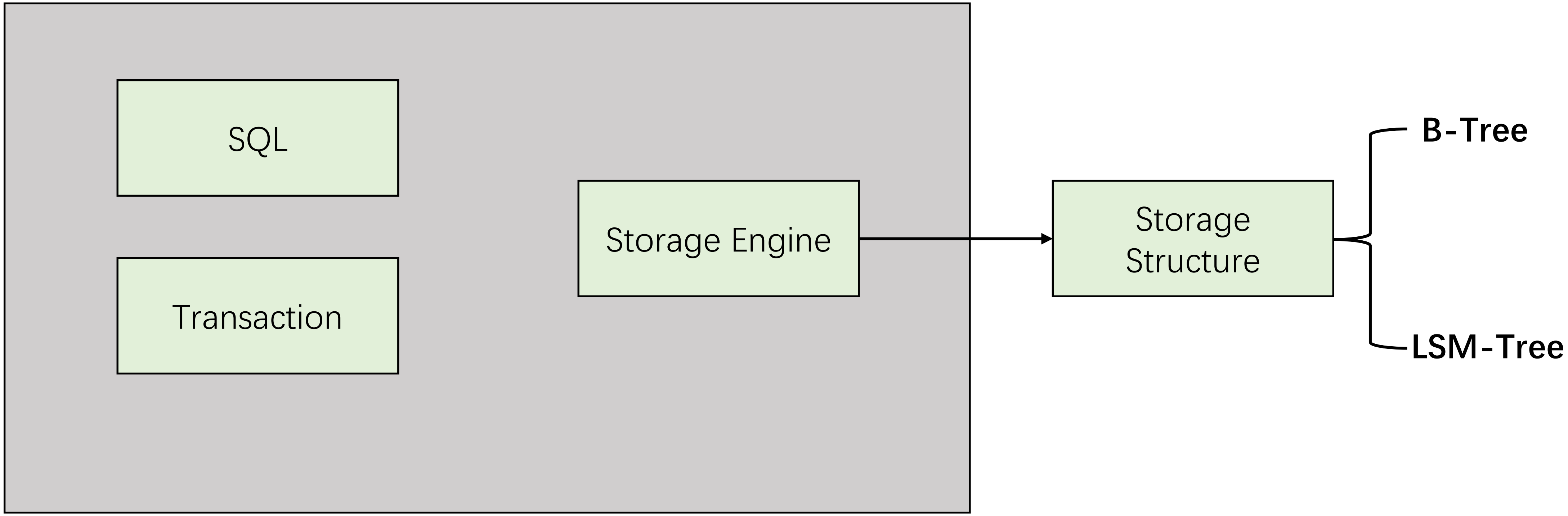
Removing Double-Logging with Passive Data Persistence in LSM-tree based Relational Databases

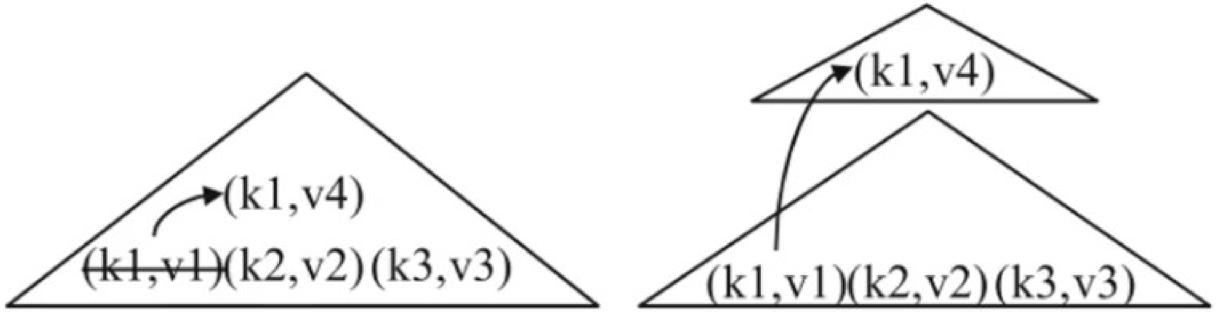
FAST-22

Background



Relational Database

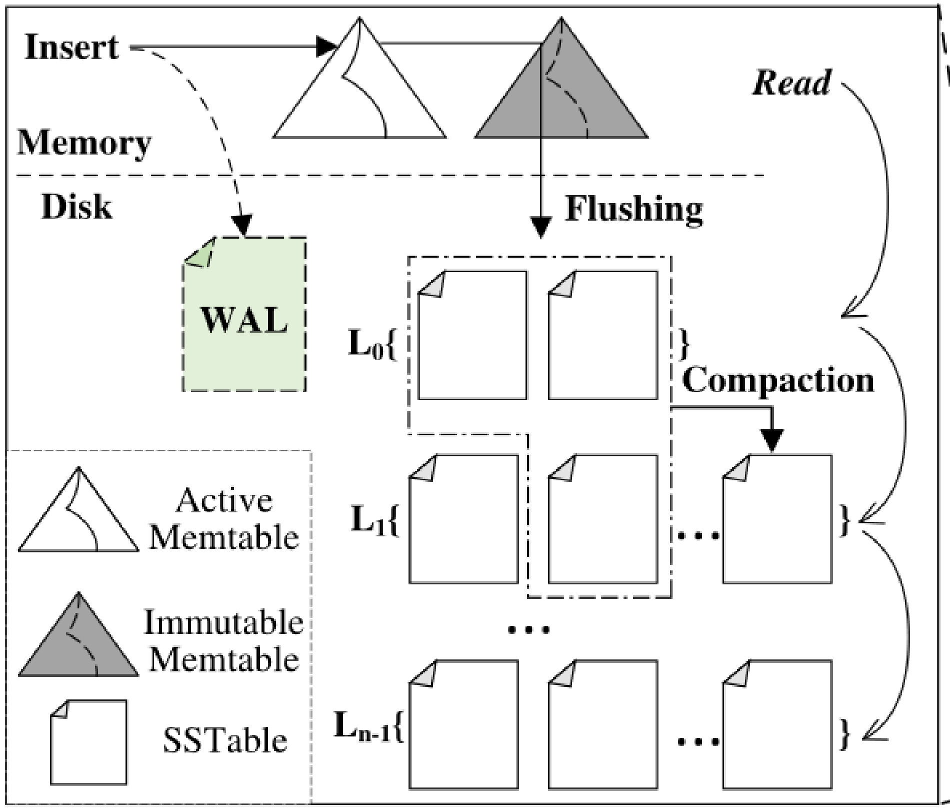
Background



(a) In-place update structure **(b)** Out-of-place update structure

- **(a) The vast majority of B-tree based storage engine**
- **(b) LSM-based storage engine**

Background



(a) Log-Structured Merge Tree

Generally speaking：

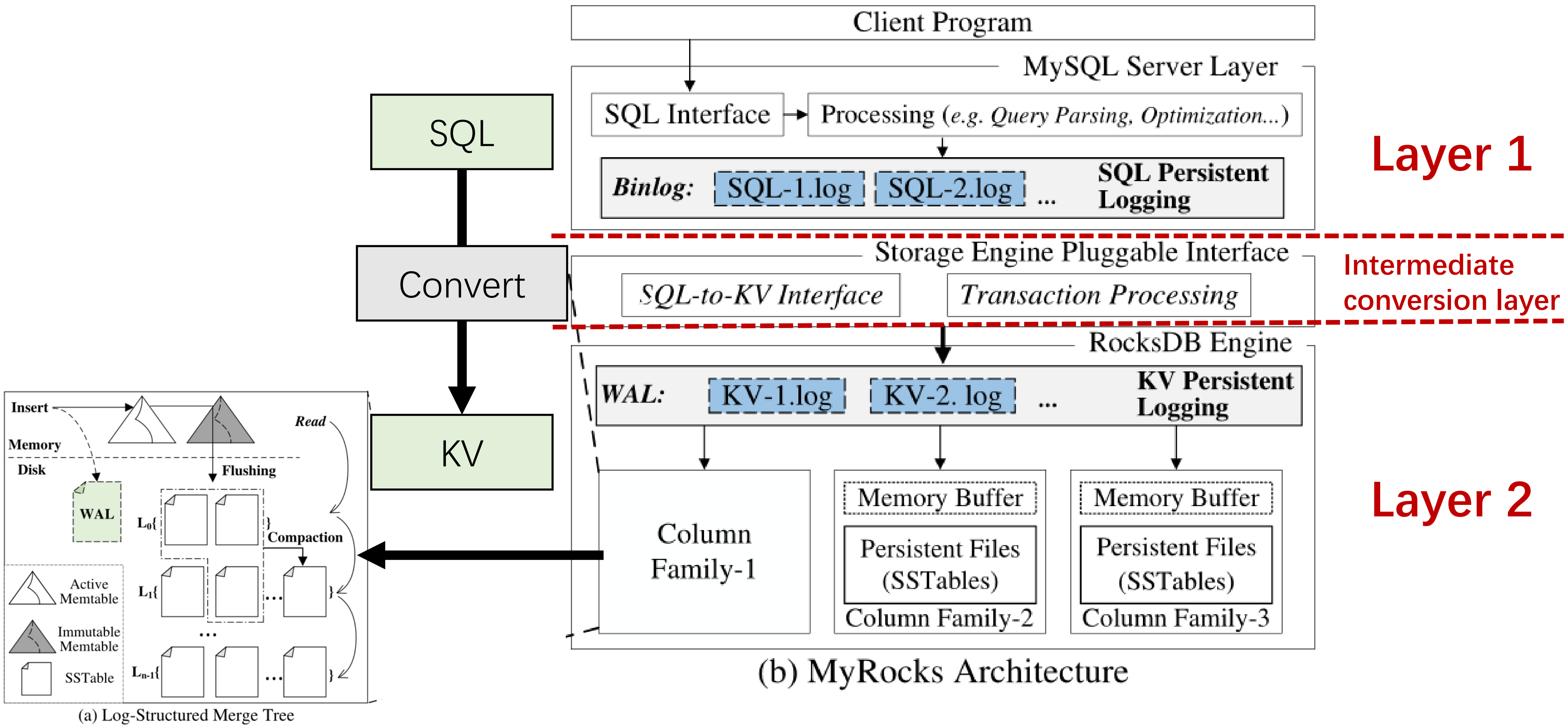
M0<M1<M2···<Mn-1

L0 Max Size：M0

L1 Max Size：M1

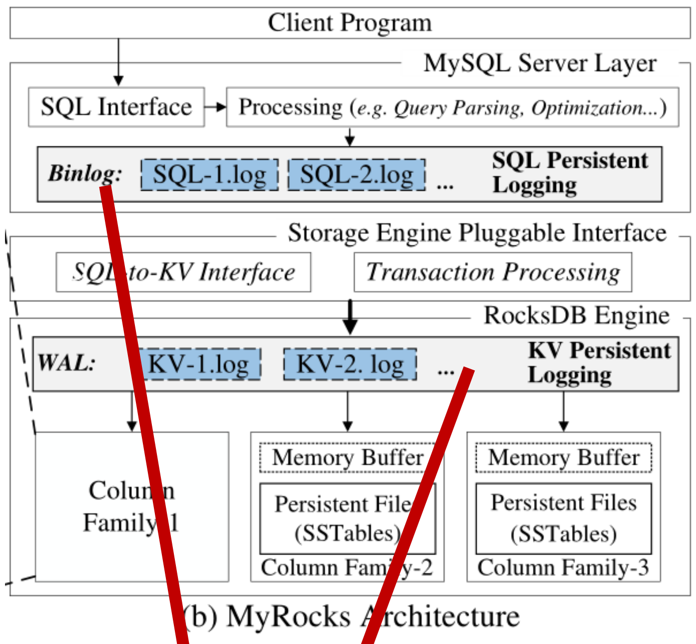
Ln-1 Max Size：Mn-1

Background

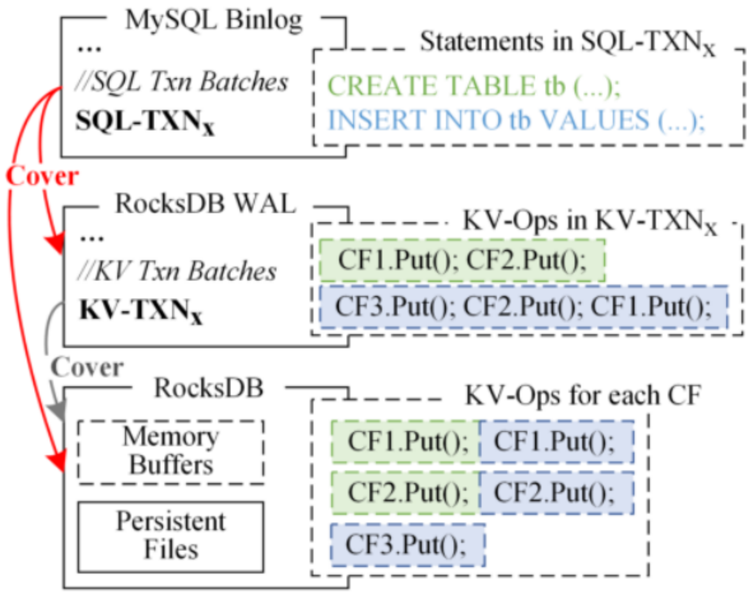


SQL
Convert
KV
Insert
Memory
Disk
Read
Flushing
WAL
Compaction
Active Memtable
Immutable Memtable
SSTable
(a) Log-Structured Merge Tree
Client Program
MySQL Server Layer
SQL Interface
Processing (e.g. Query Parsing, Optimization...)
Binlog: SQL-1.log SQL-2.log ...
SQL Persistent Logging
Storage Engine Pluggable Interface
SQL-to-KV Interface
Transaction Processing
RocksDB Engine
WAL: KV-1.log KV-2. log ...
KV Persistent Logging
Column Family-1
Memory Buffer
Persistent Files (SSTables)
Column Family-2
Memory Buffer
Persistent Files (SSTables)
Column Family-3
(b) MyRocks Architecture
Layer 1
Intermediate conversion layer
Layer 2

Problem

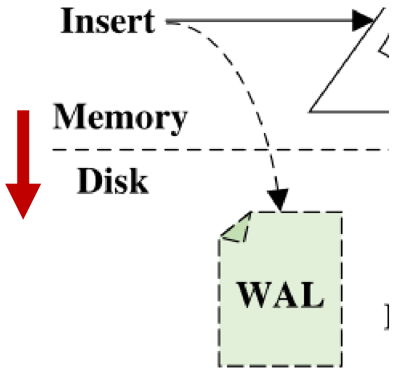


※Double-logging !!!



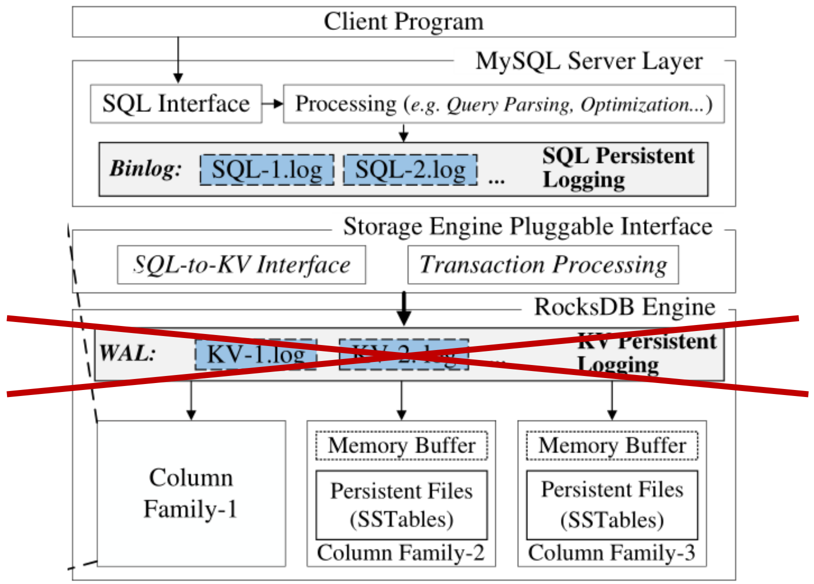
Coverage relationship

between Binlog and WAL



Frequent IO

Problem



(b) MyRocks Architecture

Straightforward Solution

Challenges：

- **Data persistence cannot be guaranteed.** We can't really know if the key-value pairs corresponding to the Transaction are actually persisted in memory and stored on the hard disk.
- **Partial persistence.** Some of the key-value pairs of a transaction may have been flushed to the hard disk at the time of the error, but other parts have not.
- **Lost track of LSN.** The transaction stored in the binlog cannot know in which LSM-Tree the translated key-value pairs are stored, and even if the binlog is replayed, it cannot regenerate the missing LSN numbers, which may make the LSN numbers in the LSM tree no longer contiguous.

Design

Design Goals：

- **Effectiveness and efficiency.** Solution should effectively and efficiently address the double-logging problem.

- **Data persistence and correctness.** Data reliability should remain identical to the existing system.

- **Minimal and non-intrusive changes.** Solution should avoid introducing complicated, intrusive changes and retain the current system’s modular structure.

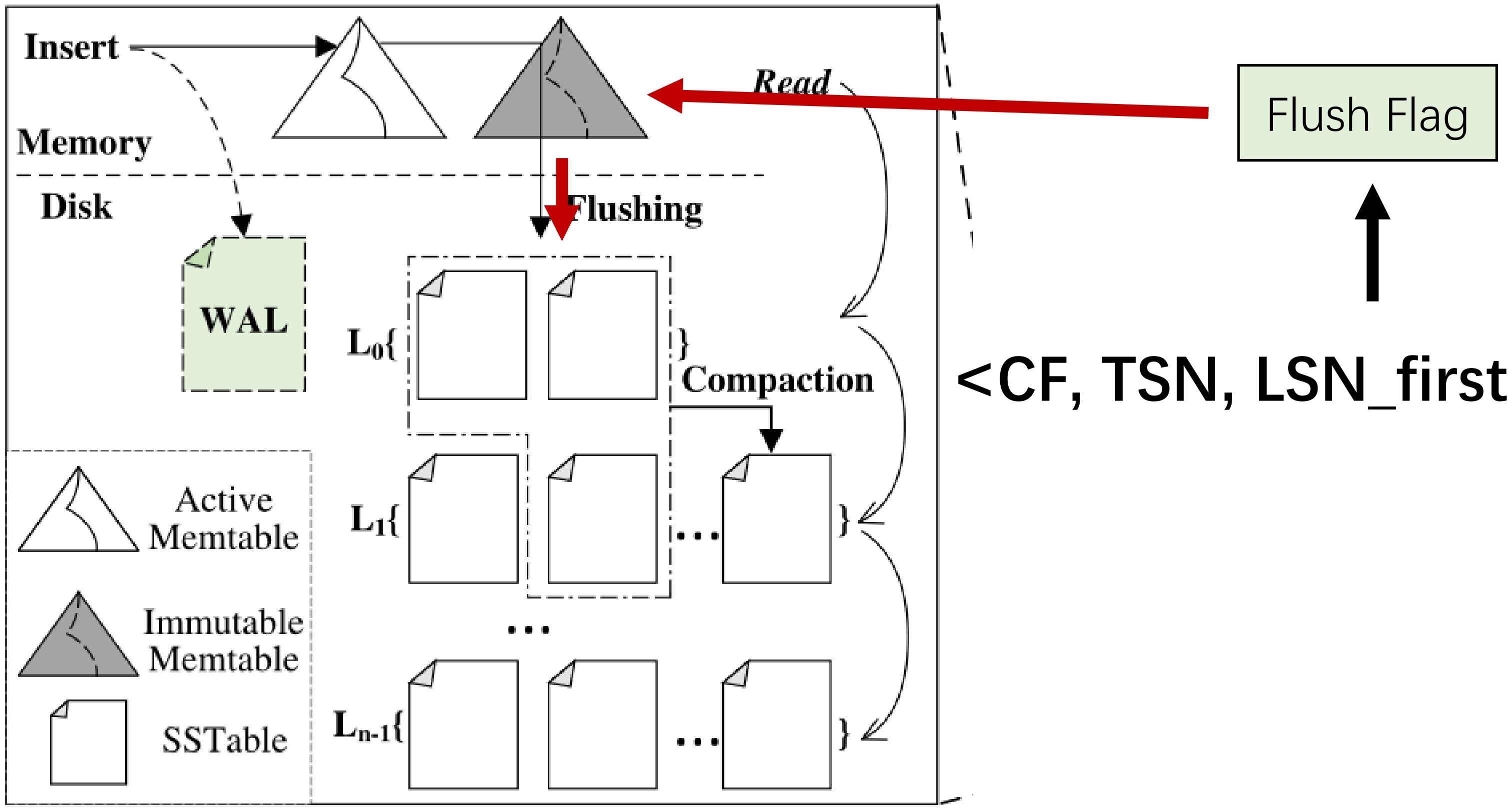
Design

New Design: Flush Flag

<CF, TSN, LSN_first, LSN_last>

- **CF.** the column family (LSM-tree) whose memory buffer is being flushed.
- **TSN.** the Transaction Sequence Number of the last transaction whose KV items are inserted in the memory buffer of the column family.
- **LSN_first, LSN_last.** LSNs of the first KV and the last persisted KV of the transaction, respectively.

Design



(a) Log-Structured Merge Tree

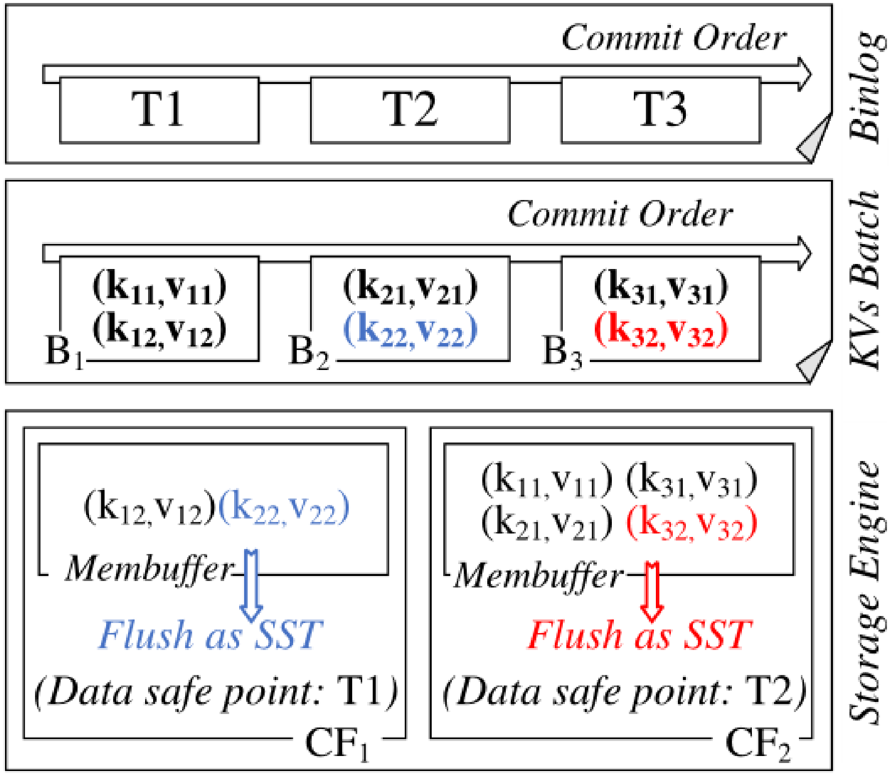
<CF, TSN, LSN_first, LSN_last>

Design

Approach is safe due to the serial property of transaction processing in LSM-tree based RDBs:

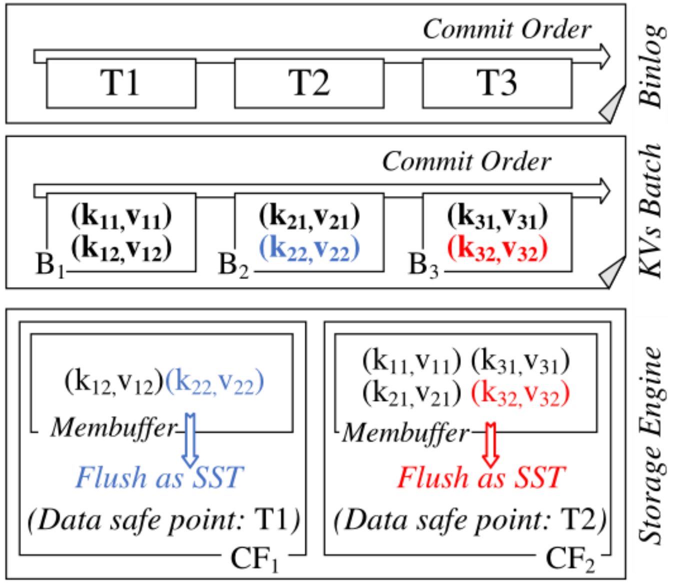
- During the transaction commit process, all transaction records are persisted to the binlog in serial;
- The SQL transactions are parsed in the RDB layer and translated into KV batches in the storage engine layer in serial;
- The KV items that are translated from a transaction are inserted into the LSM-trees' memory buffers in serial. Hence we can ensure that all KV items logically prior to the last KV item of the last transaction in a column family would never be persisted to storage later than it.

Design



Commit Order
T1
T2
T3
Binlog
Commit Order
(k11,v11)
(k12,v12)
B1
(k21,v21)
(k22,v22)
B2
(k31,v31)
(k32,v32)
B3
KVs Batch
(k12,v12)(k22,v22)
Membuffer
Flush as SST
(Data safe point: T1)
CF1
(k11,v11) (k31,v31)
(k21,v21) (k32,v32)
Membuffer
Flush as SST
(Data safe point: T2)
CF2
Storage Engine

Details

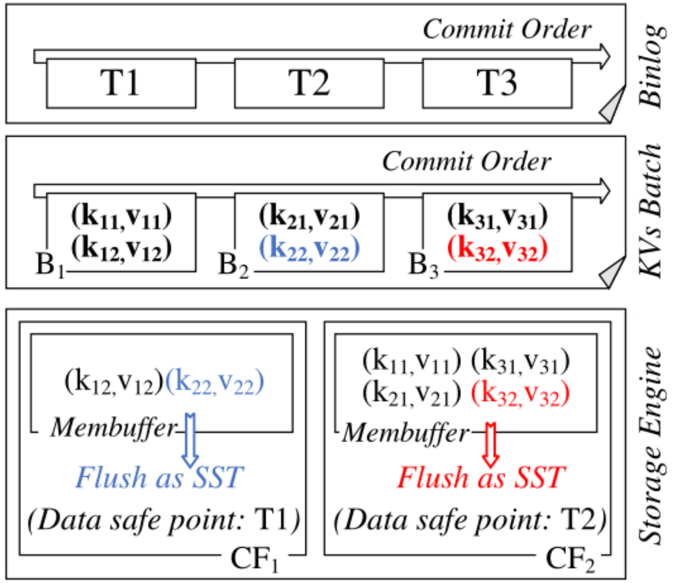


Multi-LSM-tree recovery:

- When there are multiple column families in the system, we can't select Flush Flag at will.

- The system should select the smallest transaction number among all Flush Flags as the starting point for global recovery, so that all column families can be covered.

Details



Epoch-based Persistence：

- Basic idea is to use **Local Epoch** to separately manage each CF's data safe point, and use **Global Epoch** to identify the global data safe point, which determines where we should start in binlog for recovery.

Details

Reconstructing LSNs:

- LSN represents the internal order for KV items in a KV batch (corresponding to a transaction in the RDB layer). The loss of LSNs may lead to erroneous data updates. For example, assuming a KV batch contains two update operations to the same key.

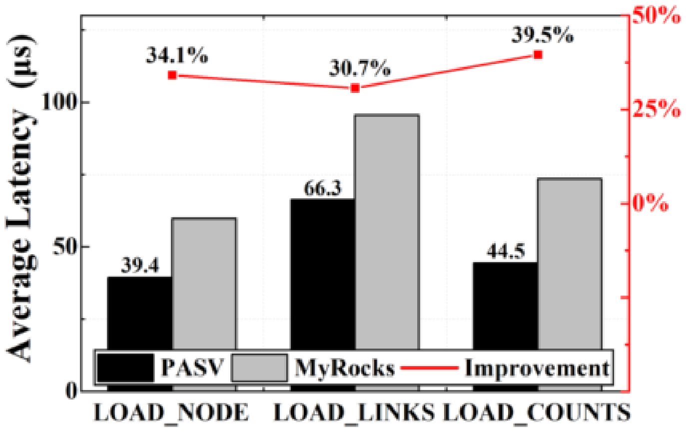
- As long as we know the original LSN of the first KV item in the batch, we can recover the entire sequence of LSNs of all generated KV items due to the serial property. The flush flag contains the last transaction and the LSNs of the first KV and the last persisted KV of the transaction. When replaying a transaction, we simply re-translate the transaction and assign the LSNs one by one. As we know the range of LSNs that are originally assigned, we can derive all the related LSNs for the KV items that need to be recovered.

<CF, TSN, LSN_first, LSN_last>

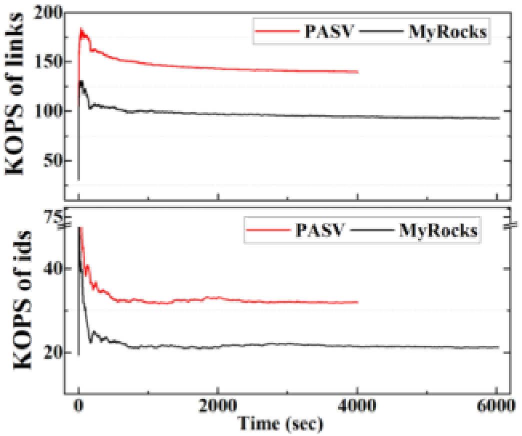
Evaluation

	MyRocks	PASV	Improvement
Total Loading Time (Seconds)	6,027.3	4,019.9	33.3%
Throughput (KOPS)	72.6	108.8	49.9%
Total Binlog Size (GB)	54.4	54.4	
Storage Engine's IO (GB)	206.5	117.9	42.9%

(a.1) General performance (data loading)



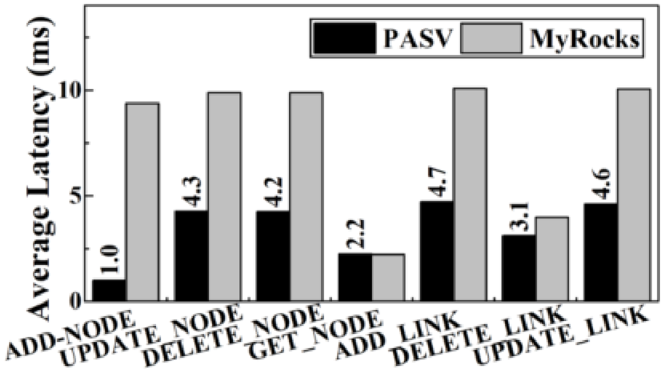
(a.2) Average latency (data loading)



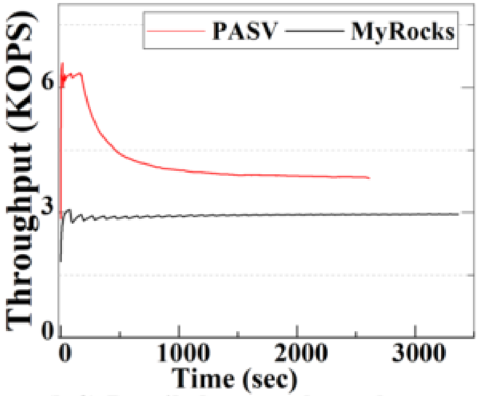
(a.3) Detailed loading throughput

	MyRocks	PASV	Improvement
Total Execution Time (Seconds)	3,371	2,614	22.5%
Throughput (KOPS)	2.97	3.82	28.6%

(b.1) General performance (query running)

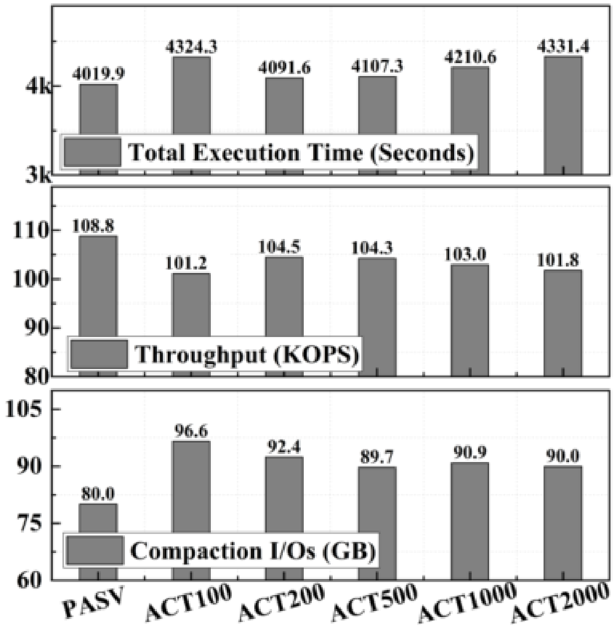


(b.2) Average latency (query running)



(b.3) Detailed query throughput

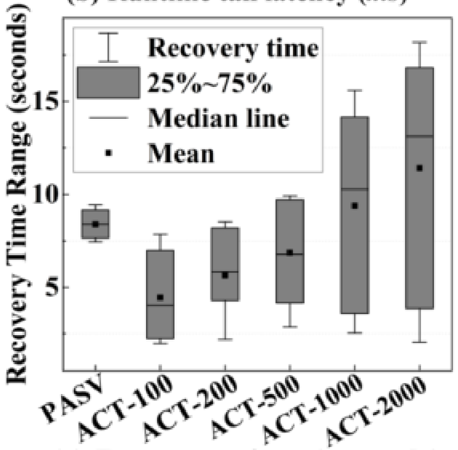
Evaluation



(a) Runtime performance and I/Os

	PASV	ACT 100	ACT 200	ACT 500	ACT 1000	ACT 2000
99%	56.9	72.6	62.3	67.2	65.3	68.8
99.9%	121.8	215.2	163.4	160.7	356.7	366.8
99.99%	526.1	632.8	584.3	635.5	827.9	699.4

(b) Runtime tail latency (*ms*)

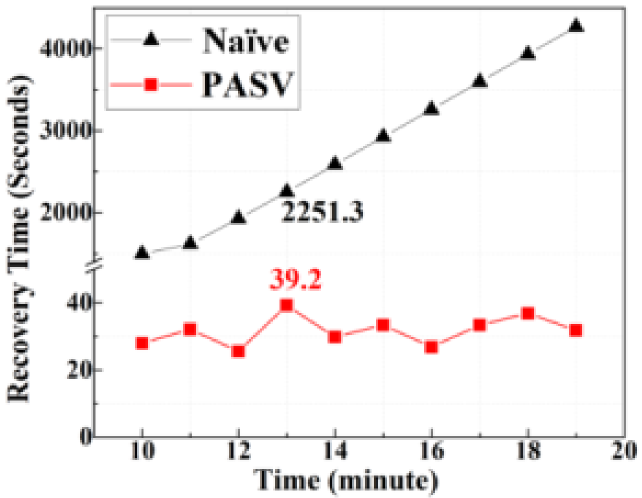


(c) Recovery time (seconds)

Evaluation

	PASV	Naïve	MyRocks
Total Execution Time (Seconds)	2,658.4	2,651	3,612
Throughput (KQPS)	35.6	35.7	26.2
Storage Engine's IO (GB)	204.2	204.2	267.6

(a) General performance under TPCC workload



(b) Recovery time under TPCC workload

Summary

PASV：

- **Identify the problem: Double-logging Problem.**
- **Propose a straightforward solution: Disable WAL.**
- **Presented Solution: PASV, and a clean design: Flush Flag.**
- **Use Flush Flag wisely for challenges**