RocksDB

Key-Value Store Optimized For Flash

Siying Dong

Software Engineer, Database Engineering Team @ Facebook

April 20, 2016

Agenda

What is RocksDB?

- Key-Value persistent store
- Point / range lookup
- Optimized for flash
- Also work for pure-memory and spinning disks
- C++ library
- Other language bindings
- Fork of LevelDB

RocksDB Interface

- Keys and Values are byte arrays
- Keys have total order.
- Update Operation: Put/Delete/Merge
- Queries: Get/Iterator
- Consistency: atomic multi-put, multi-get, iterator, snapshot read, transactions

How Is RocksDB Used?

- As Storage Engine of Databases
- As Embedded Storage of Applications

RocksDB As Storage Engine of Data Management Systems

mongoDB

RocksDB | Mmap | WiredTiger

MySQL®

MyISAM ...... InnoDB RocksDB

Yahoo Sherpa

RocksDB

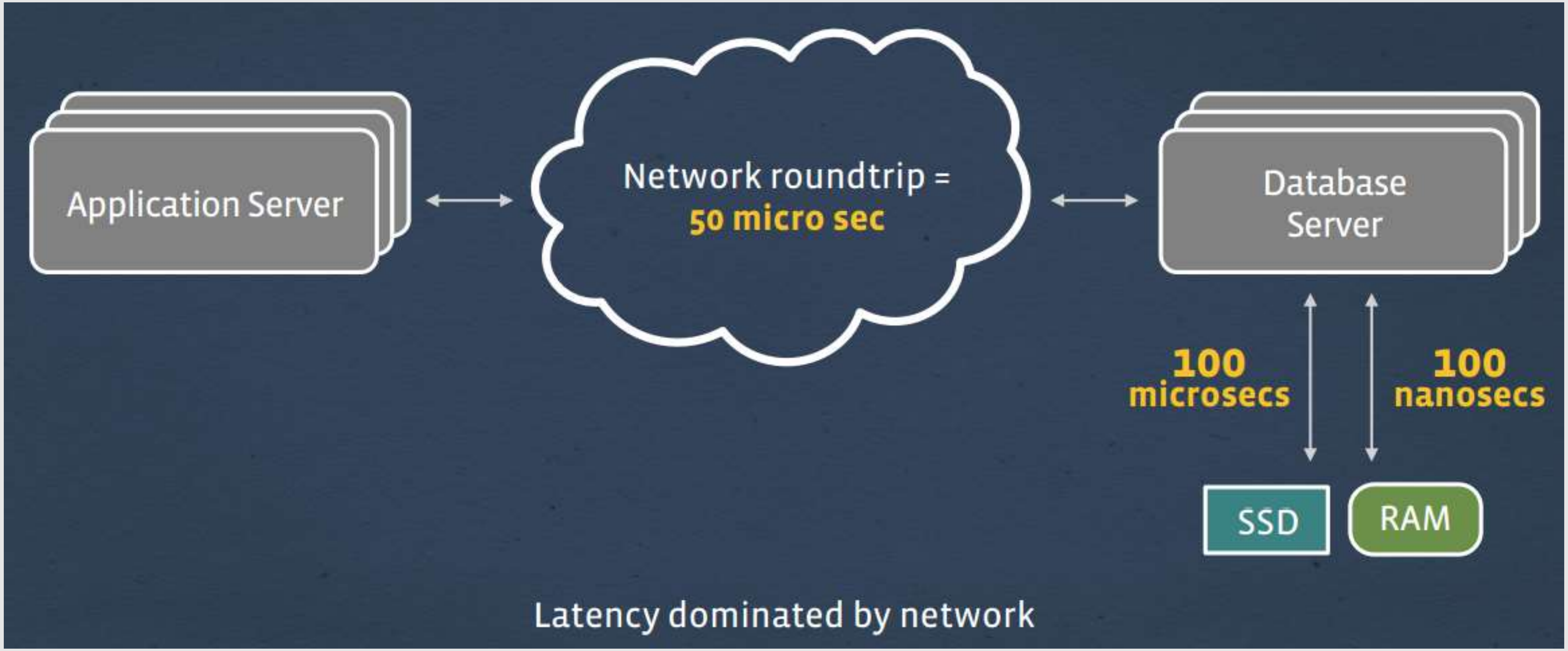
ZippyDB

And many more ...

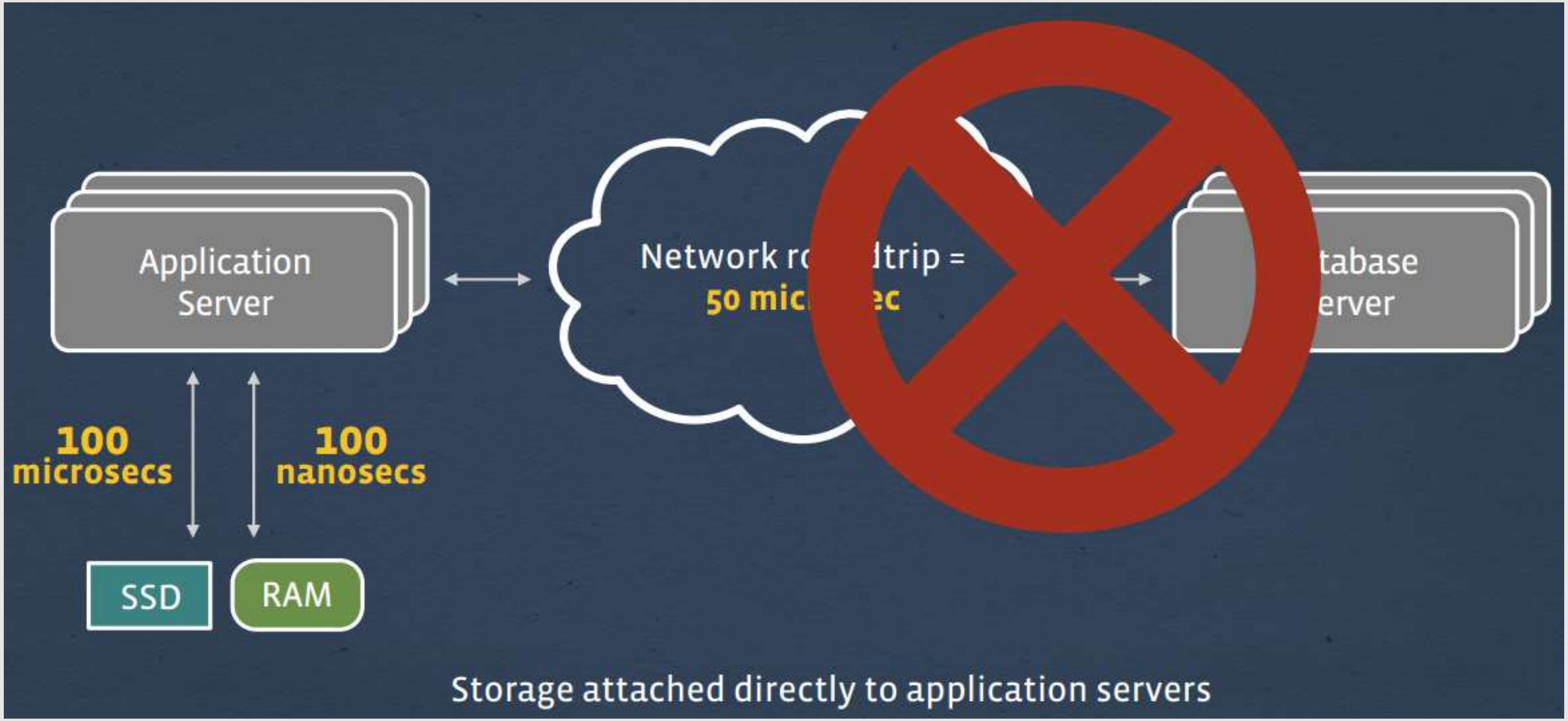
RocksDB As Embedded Storage

- Facebook: many backend services
- LinkedIn’s FollowFeed
- Apache Samza
- Iron.io
- Tango Me
- Ceph
- And more…

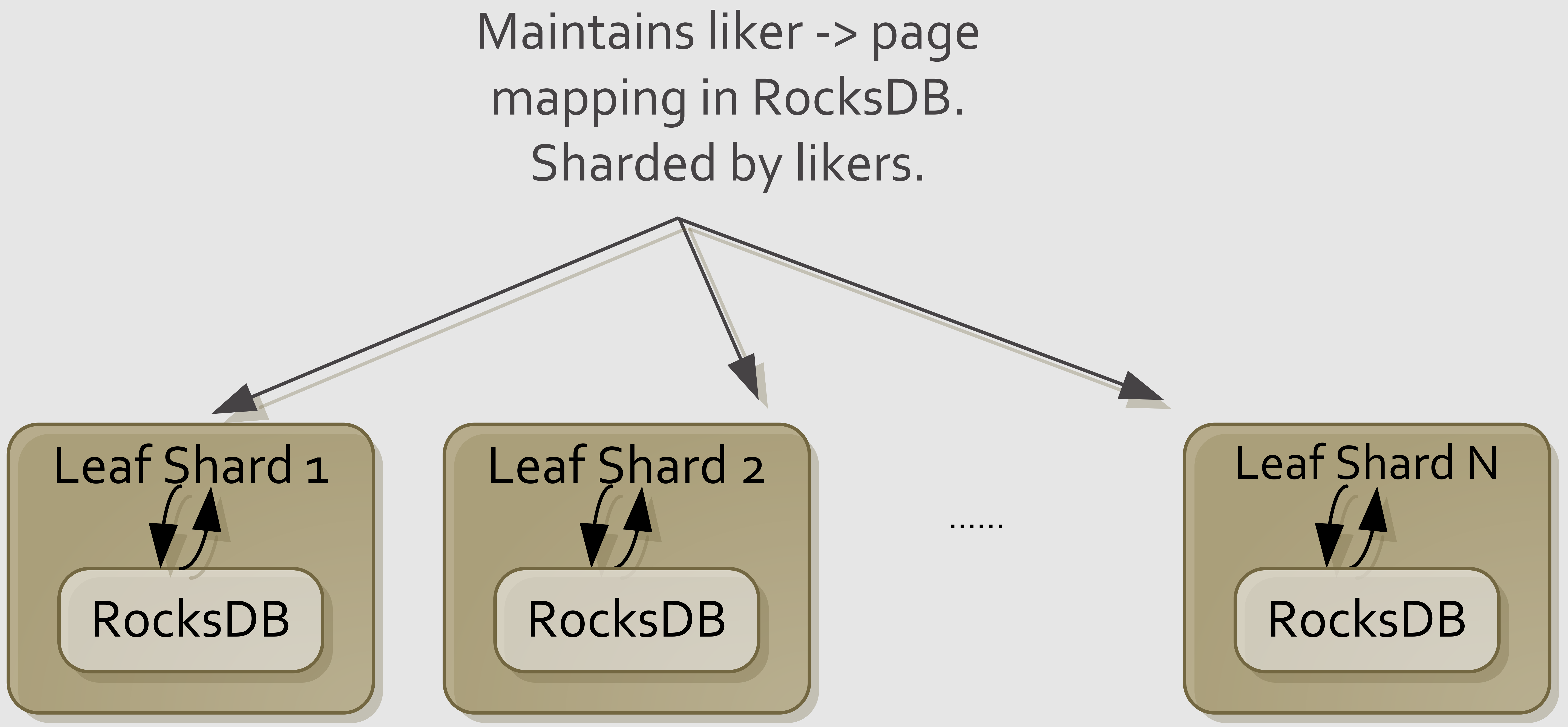
Why Embedded Storage in Application?



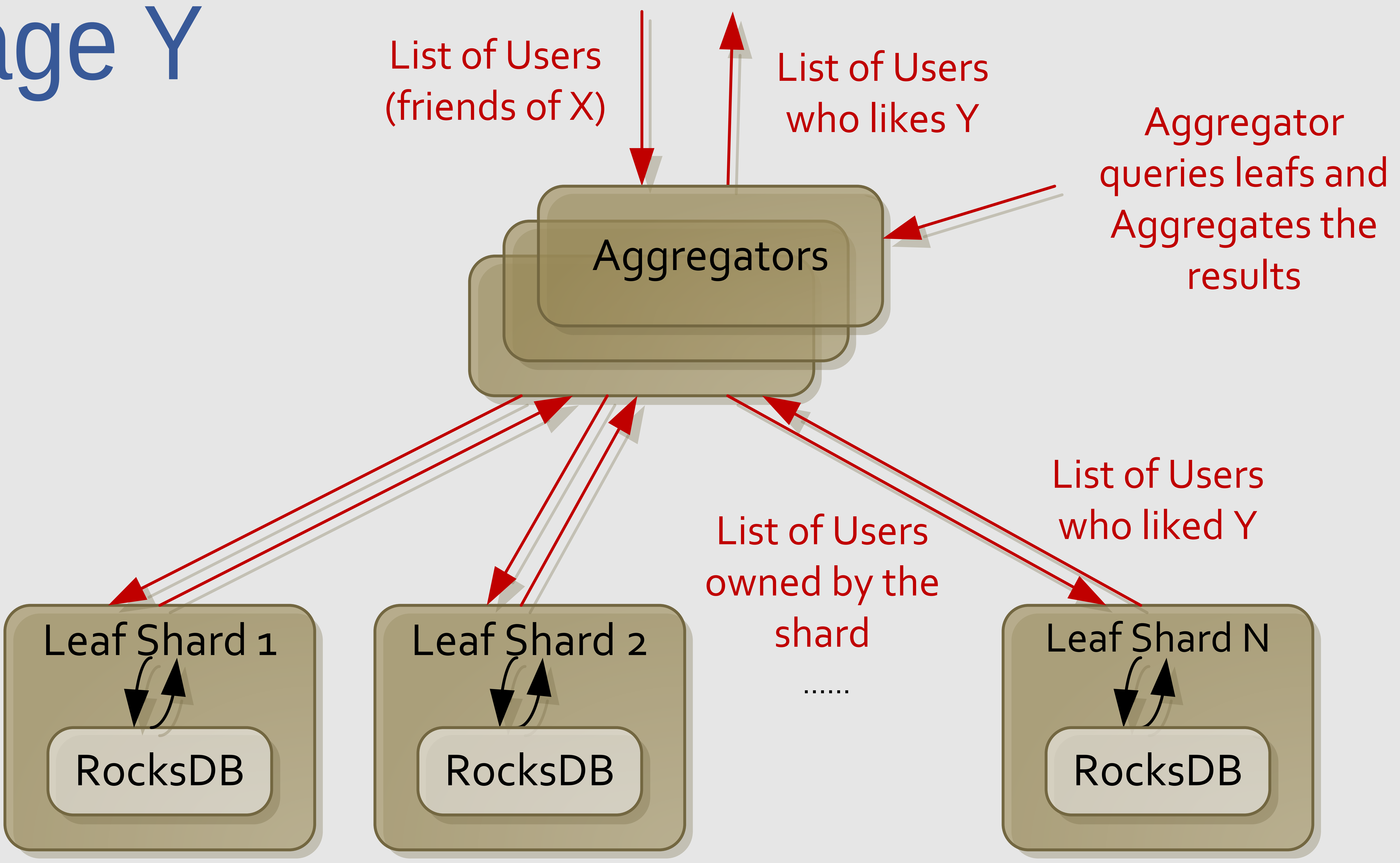
Why Embedded Storage in Application?



Example: Find all friends of user X who like Page Y



Example: Find all friends of user X who like Page Y

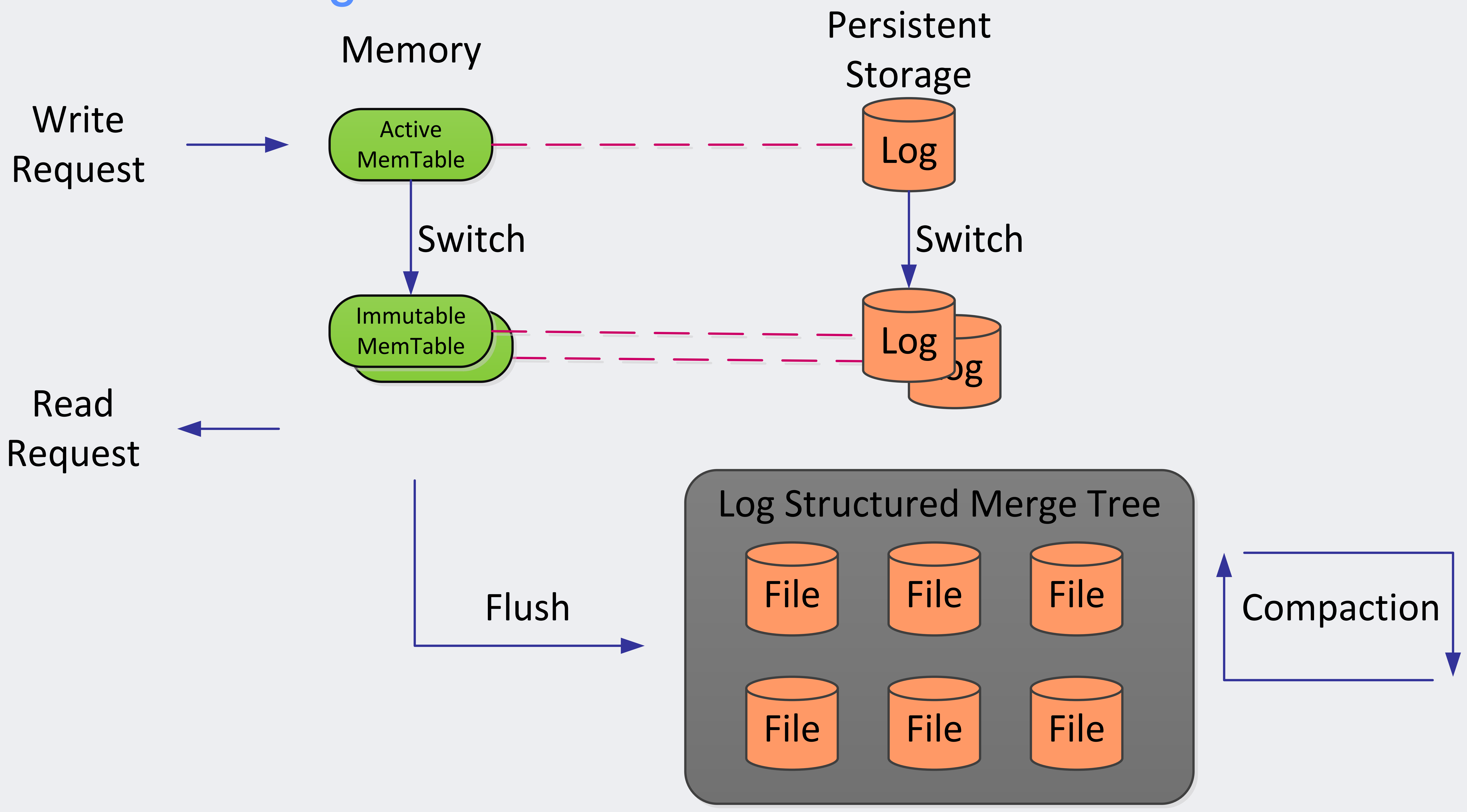


RocksDB Design

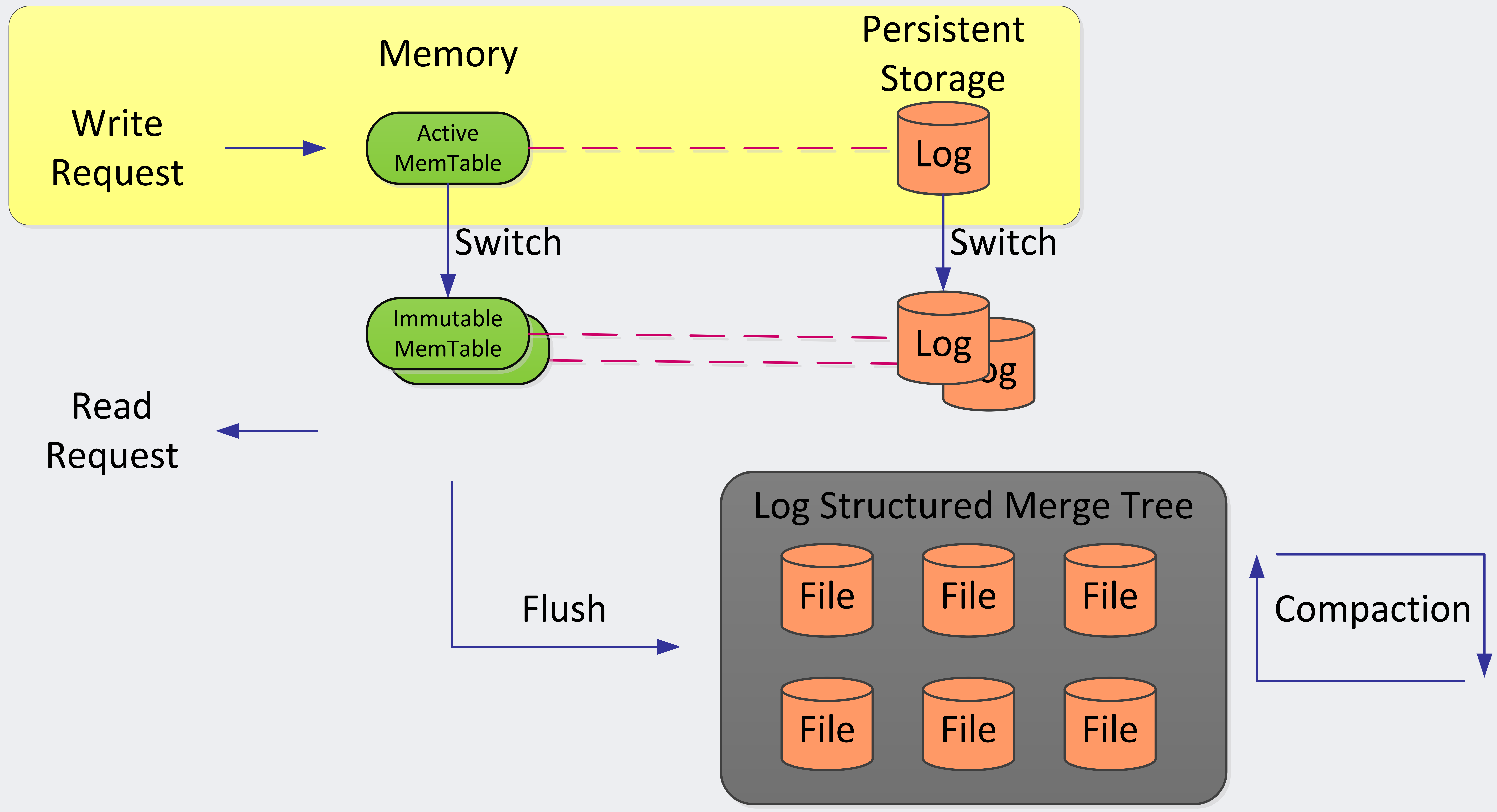
RocksDB Design

RocksDB Architecture

Log-Structured Merge-Tree

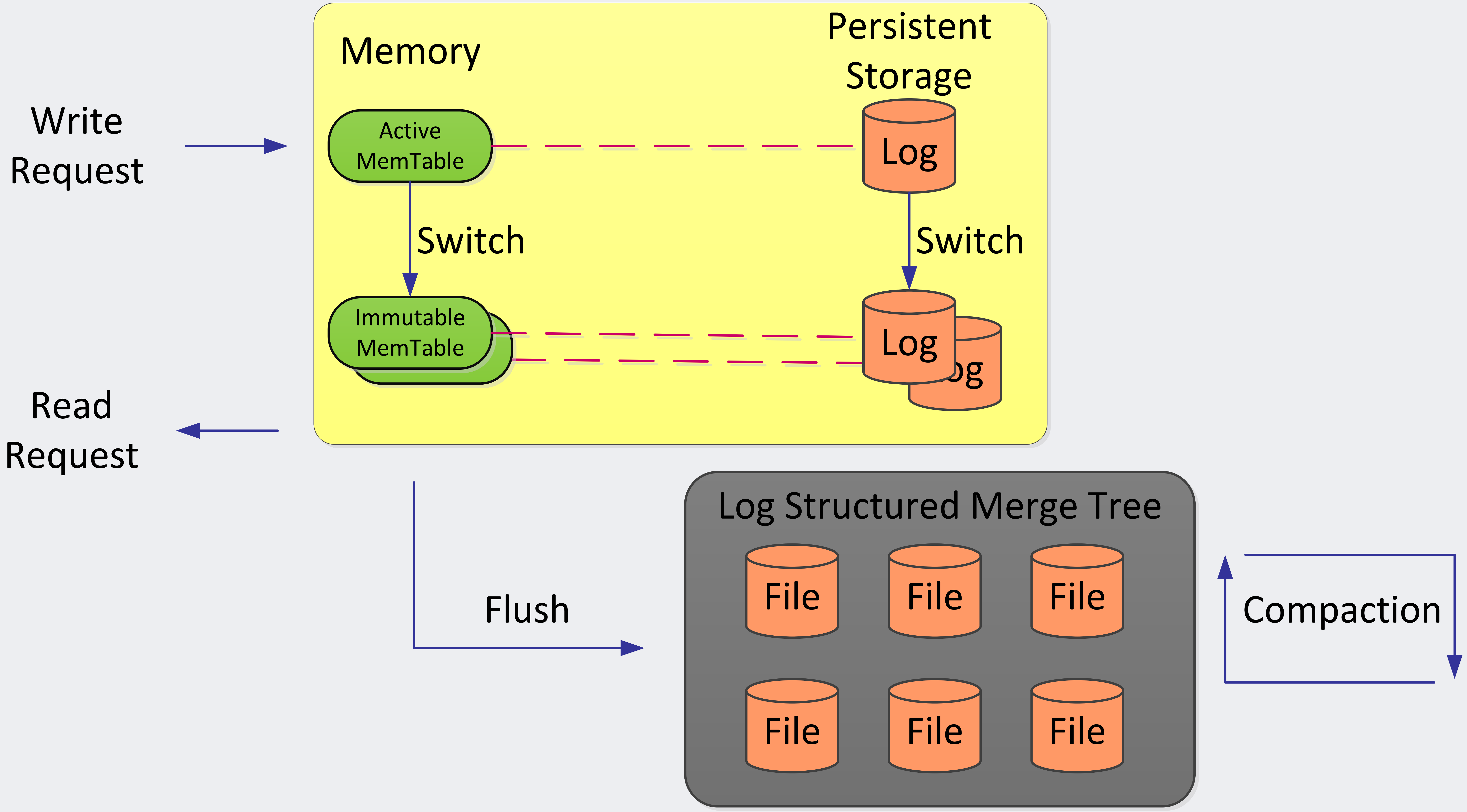


Write Path (1)

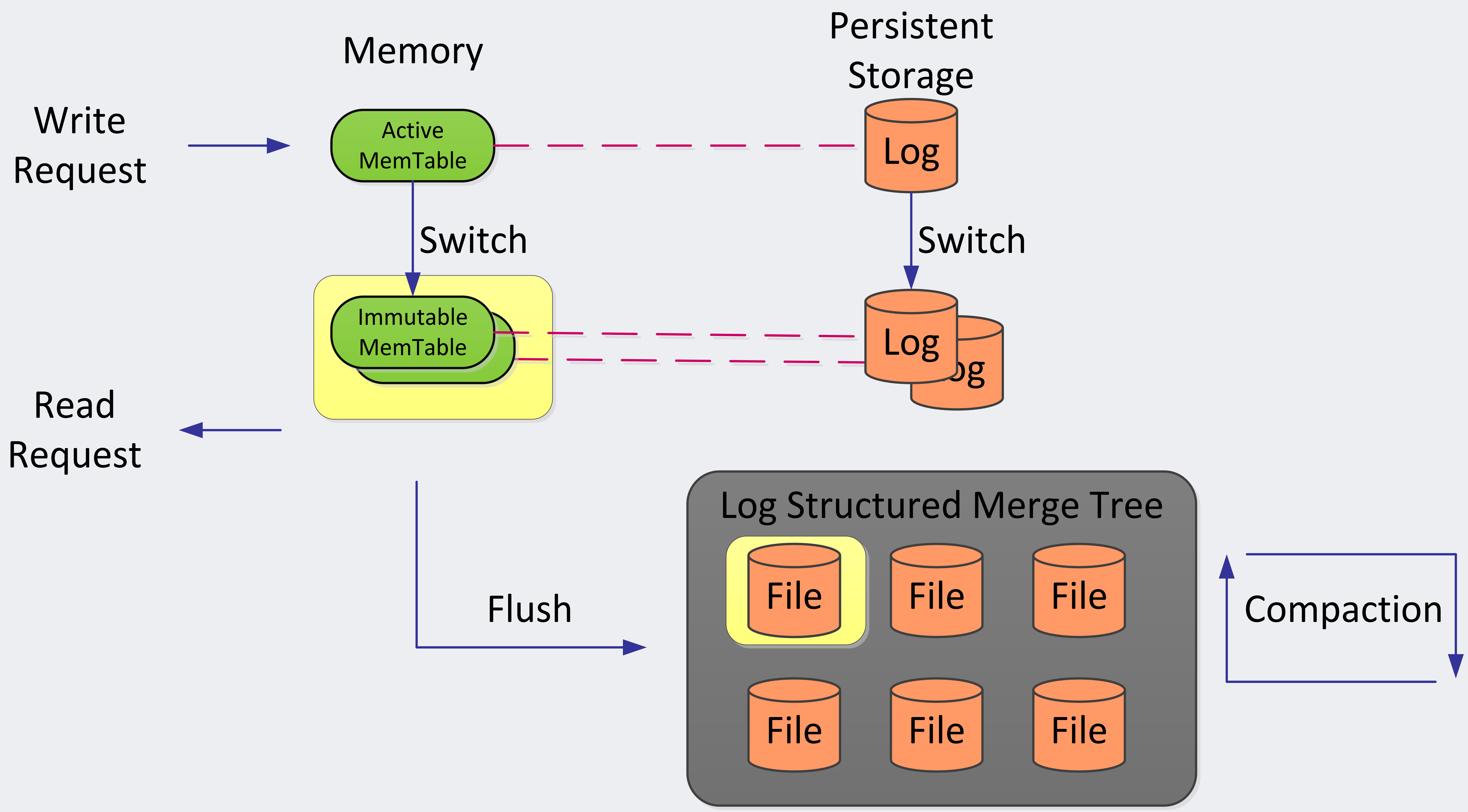


Memory
Persistent Storage
Write Request
Active MemTable
Log
Switch
Switch
Immutable MemTable
Log
Log
Read Request
Flush
Log Structured Merge Tree
File
File
File
File
File
File
Compaction

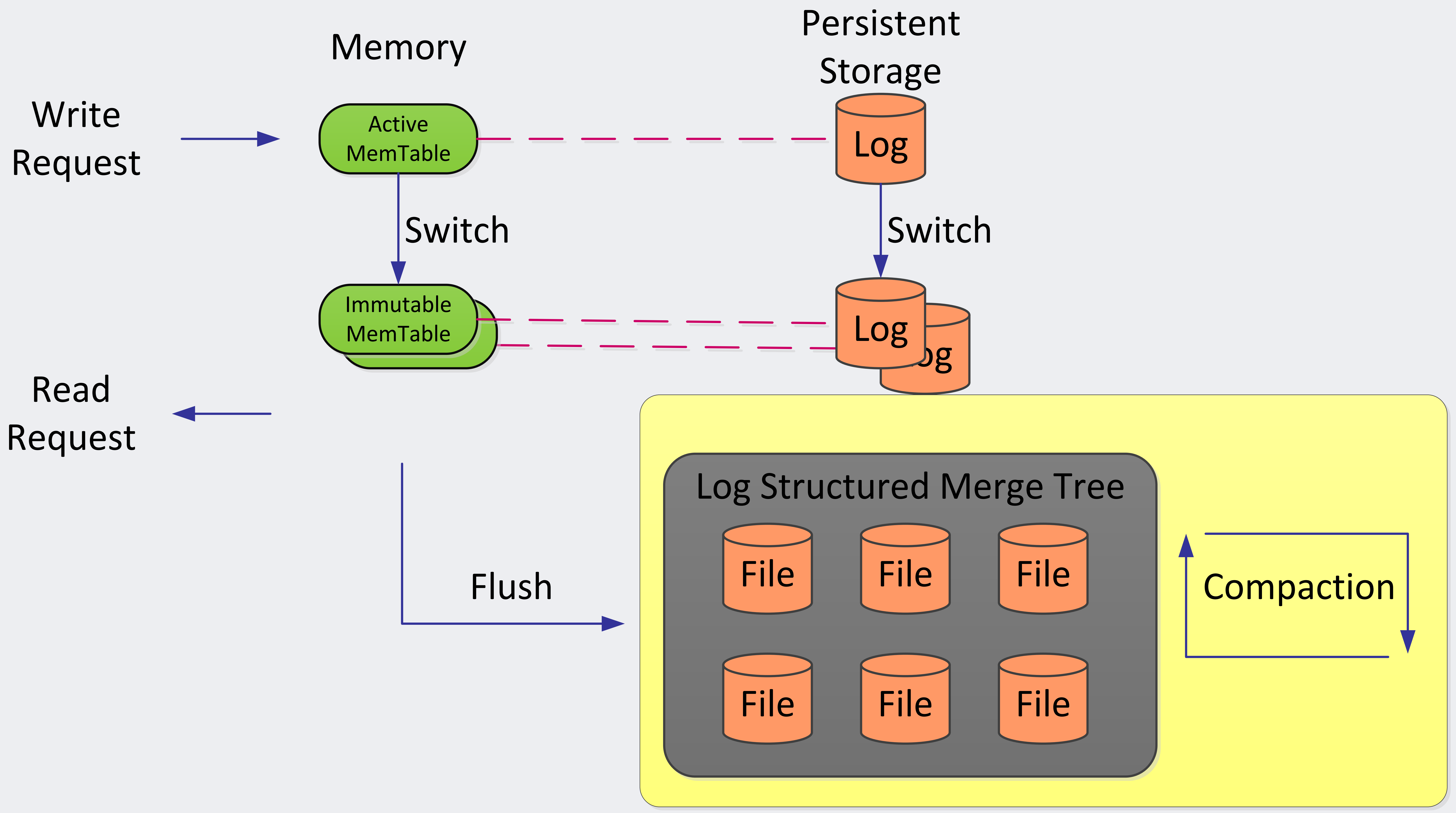
Write Path (2)


Write Request
Read Request
Memory
Persistent Storage
Active MemTable
Log
Switch
Switch
Immutable MemTable
Log
Log
Flush
Log Structured Merge Tree
File
File
File
File
File
File
Compaction

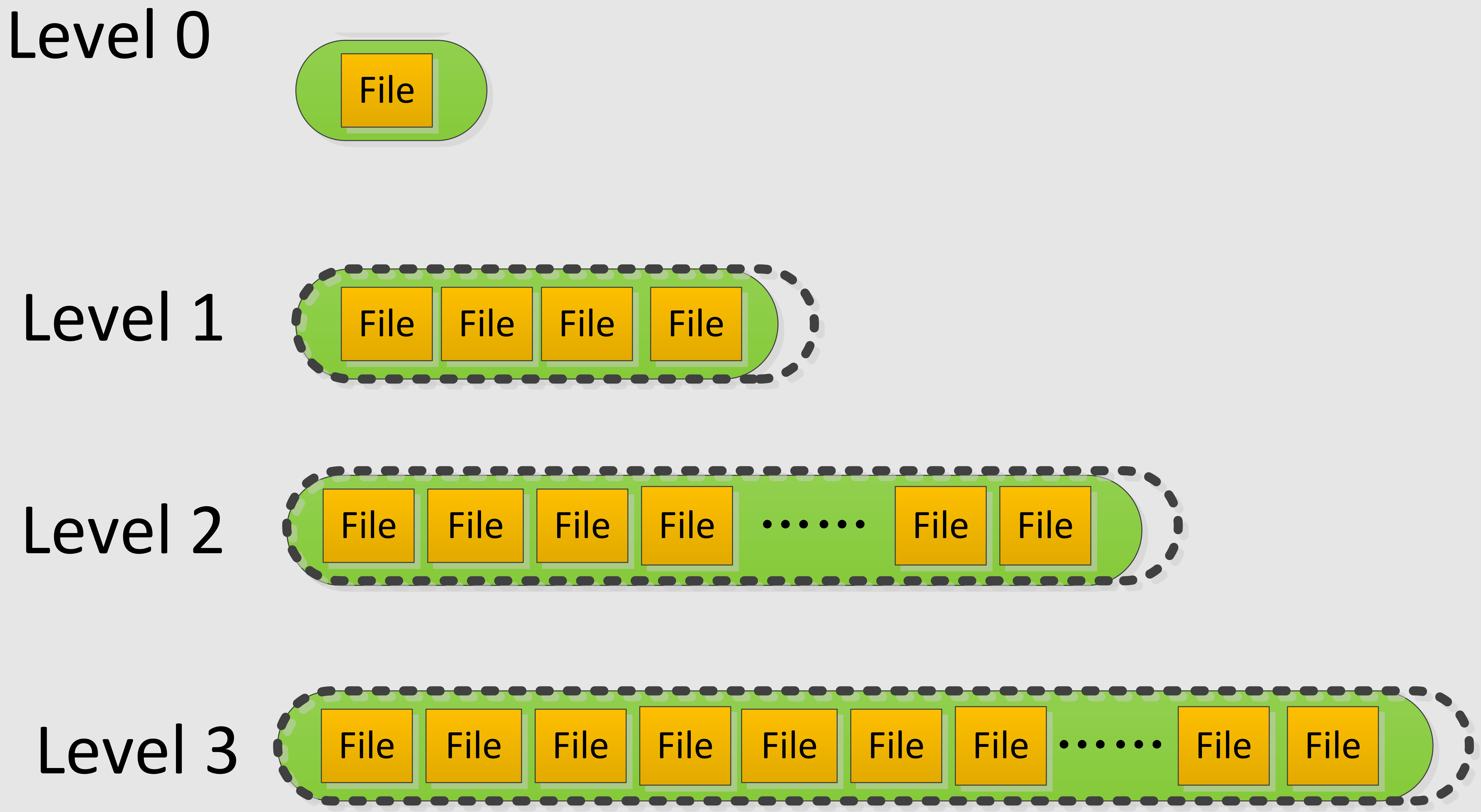
Write Path (3)


Memory
Persistent Storage
Write Request
Active MemTable
Log
Switch
Switch
Immutable MemTable
Log
Log
Read Request
Flush
Log Structured Merge Tree
File
File
File
File
File
File
Compaction

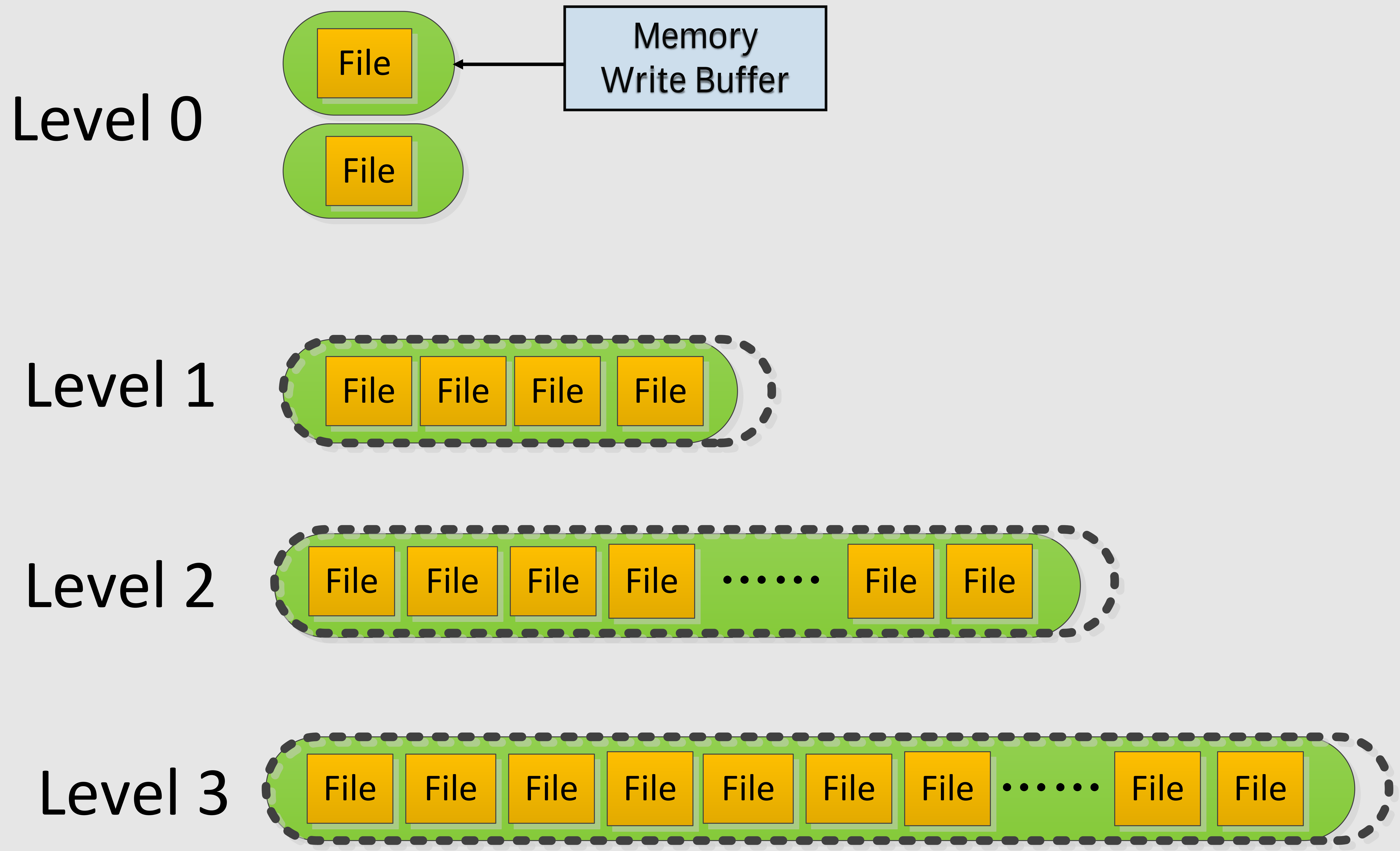
Write Path (4)


Memory
Persistent Storage
Write Request
Active MemTable
Log
Switch
Switch
Immutable MemTable
Log
Log
Read Request
Log Structured Merge Tree
File
File
File
File
File
File
Flush
Compaction

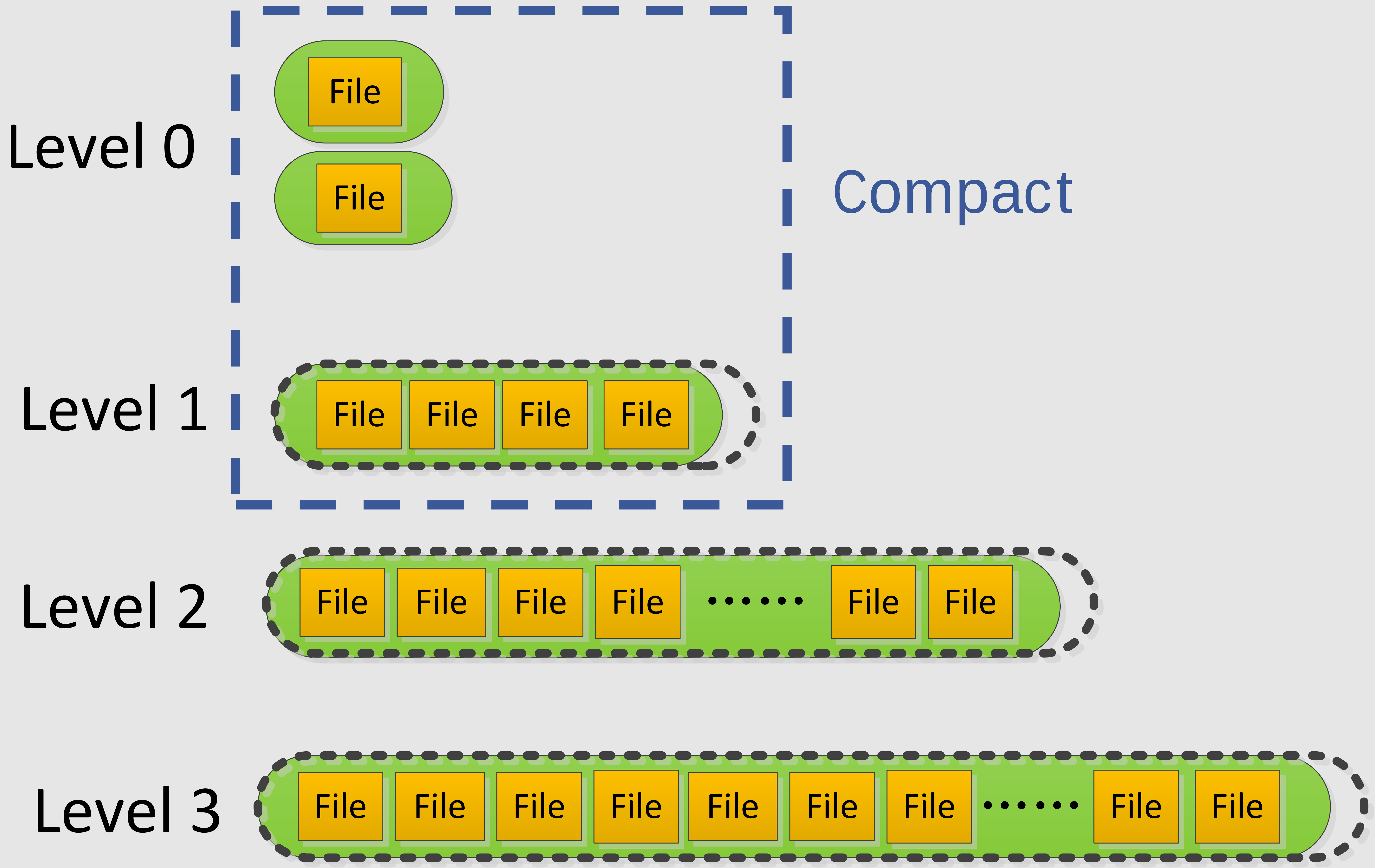
Level-Based Compaction



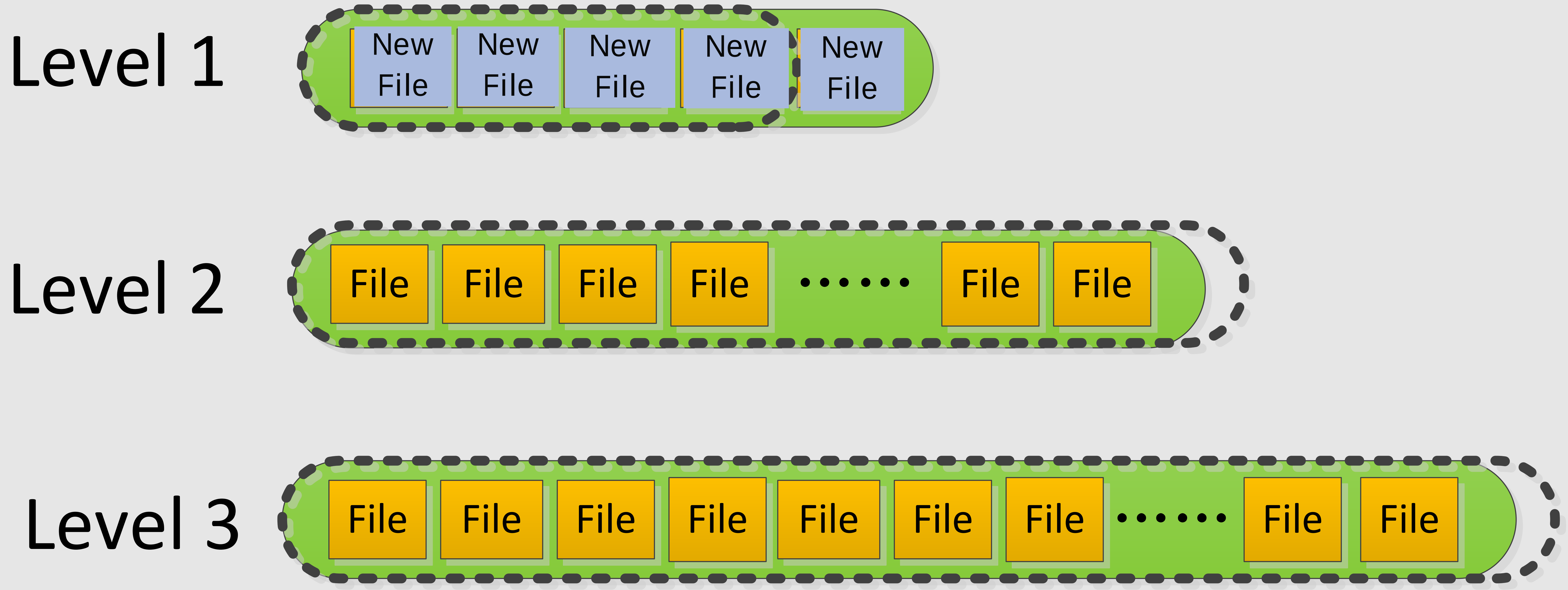
Level-Based Compaction



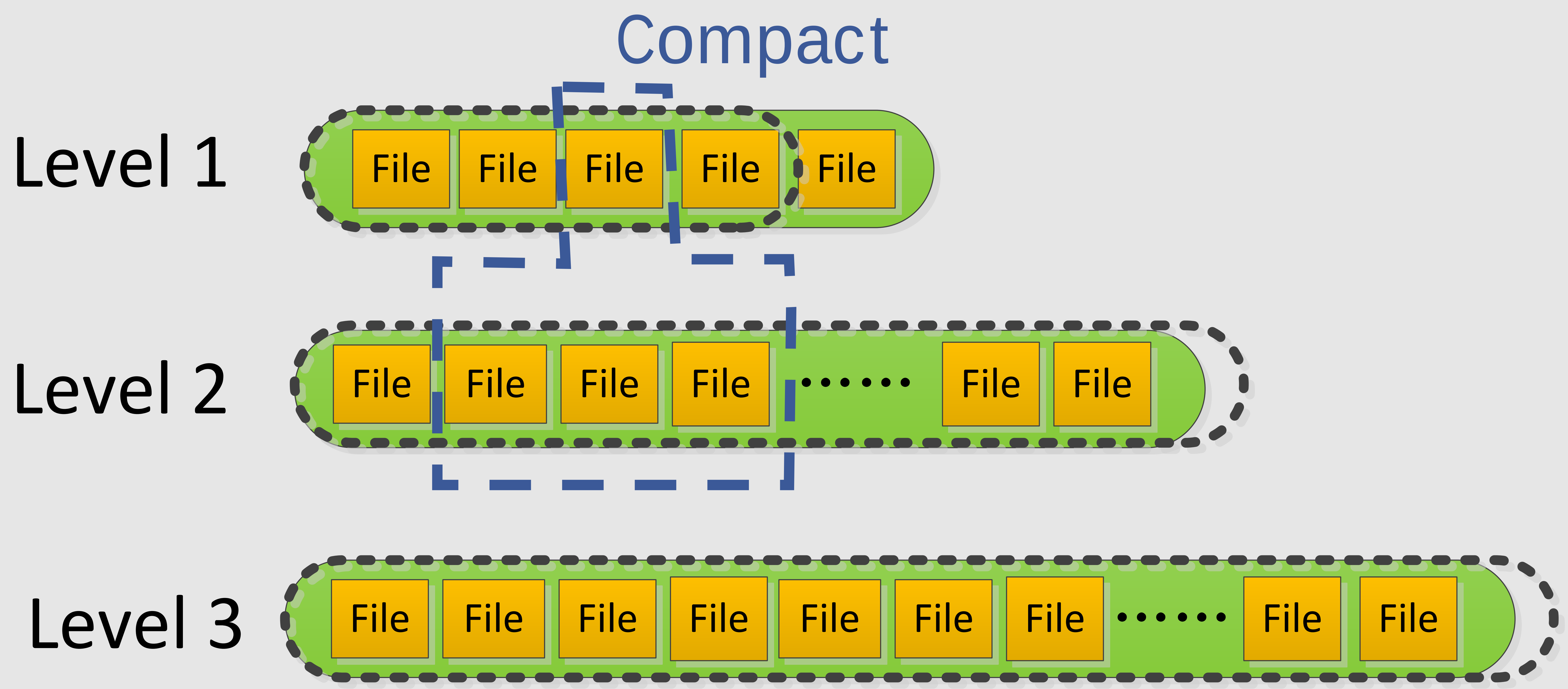
Level-Based Compaction



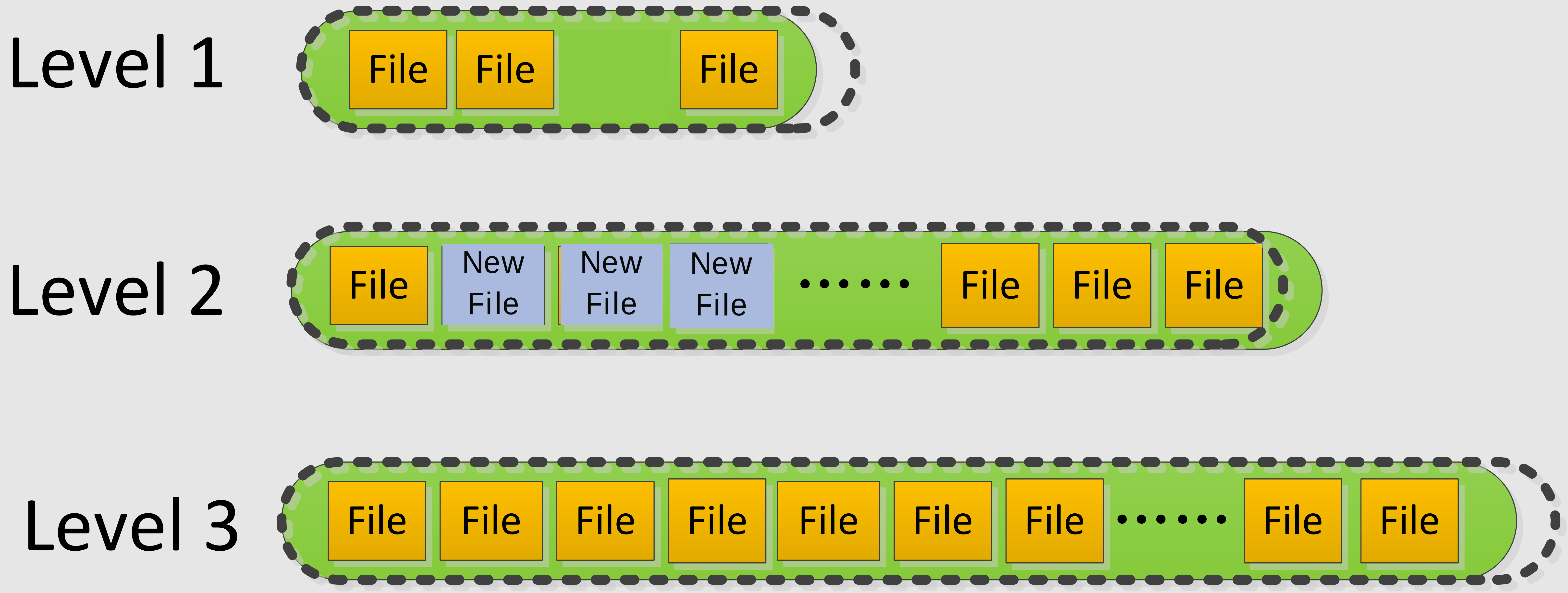
Level-Based Compaction



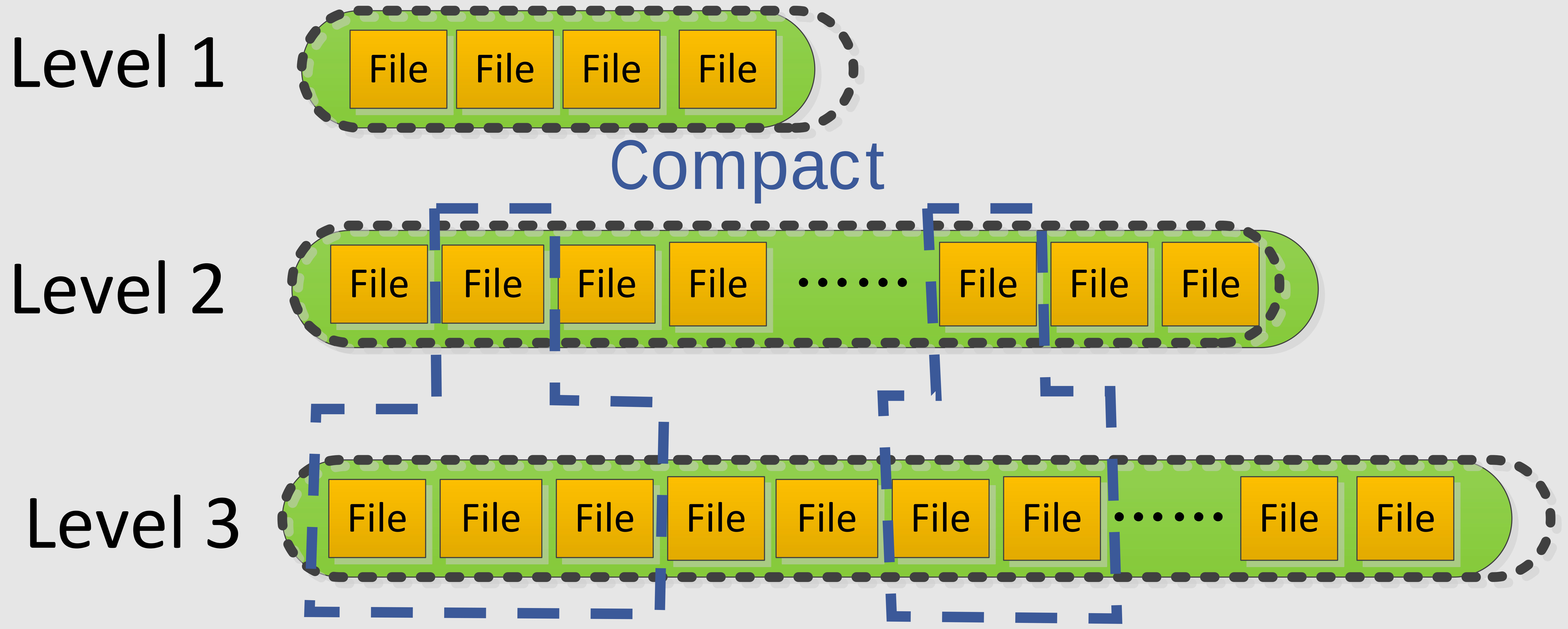
Level-Based Compaction



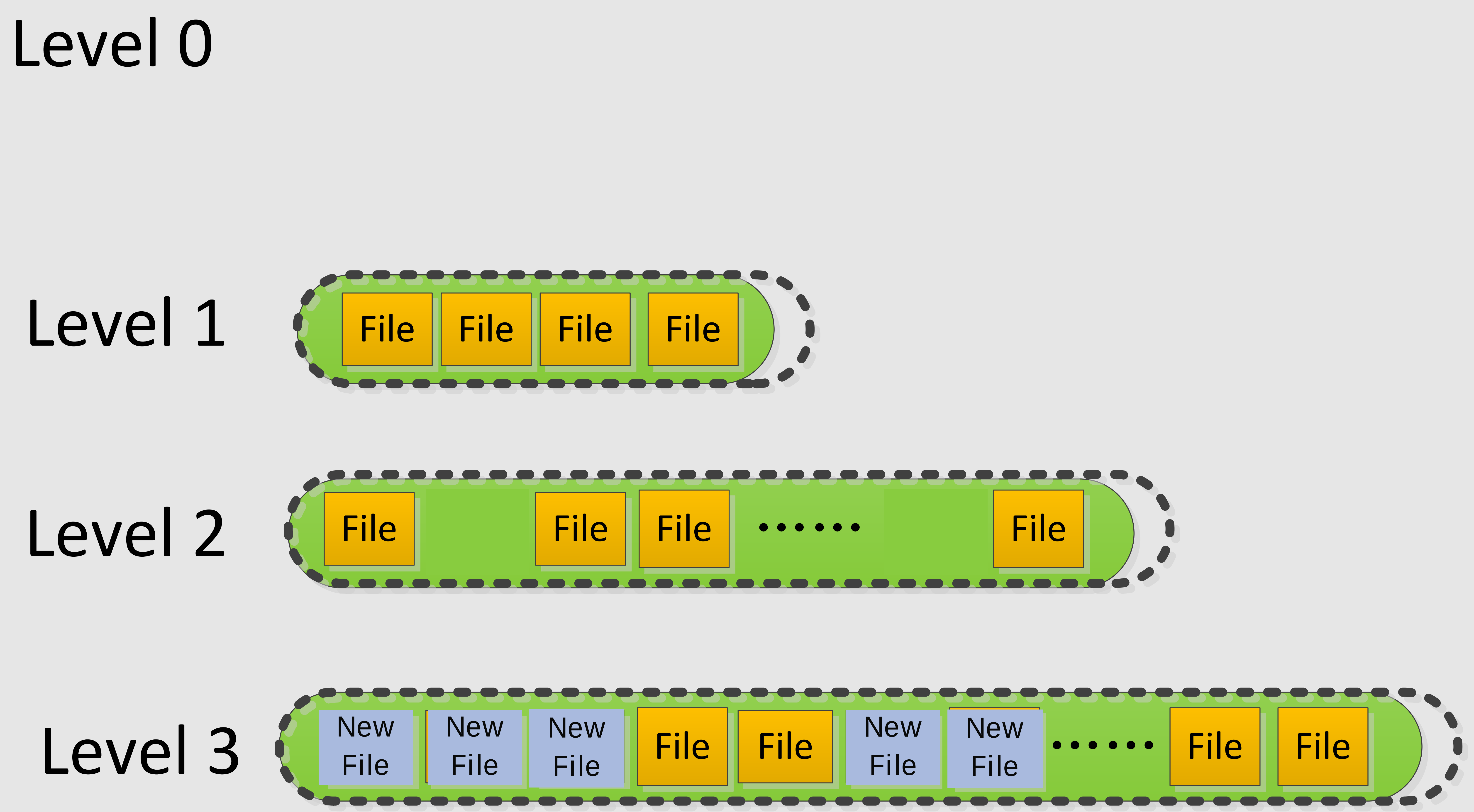
Level-Based Compaction



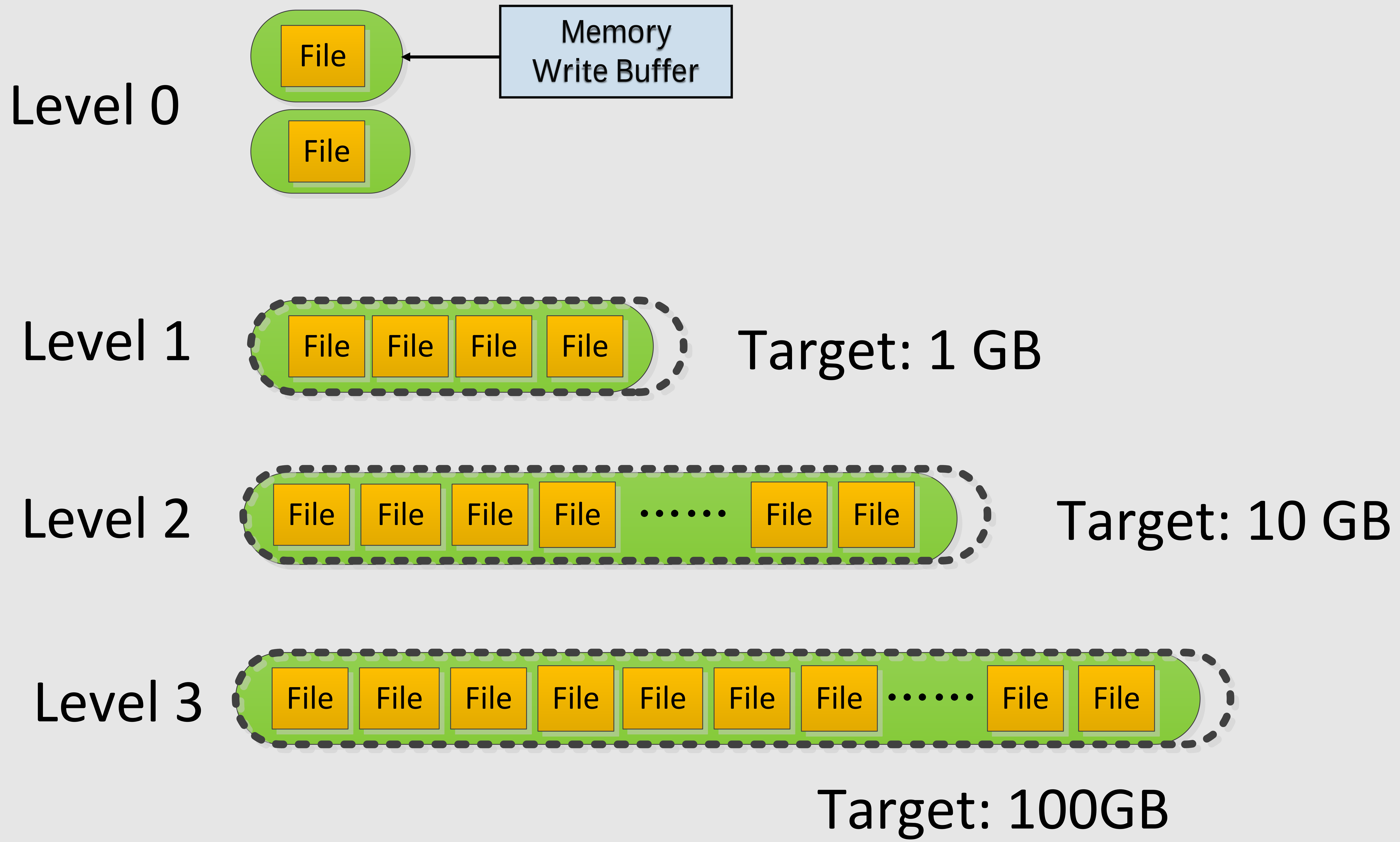
Level-Based Compaction



Level-Based Compaction



Example of Level Base Targets



Why is it flash-friendly?

Tuning Flexibility for Flash

Performance Metrics for applications on flash devices

- Write Amplification –wear out devices slower
- Space Amplification – store more data
- Read Amplification – better read IOPs

Compactions' Impact on Amplifications

	Space Amplification	Write Amplification	Memory Cache Required for ReadAmp = 1
More Aggressive Compactions	🙂	🙁	🙂
Less Aggressive Compactions	🙁	🙂	🙁

Space Amplification is the bottleneck

- Example: our MySQL host on InnoDB:
  - *Read IOPS: < 10%*
  - *Write IOPS: < 35%*
  - *Peak Write Bandwidth: < 25%*
  - *CPU: < 40%*
  - *Write Endurance: last more than 3 years.*

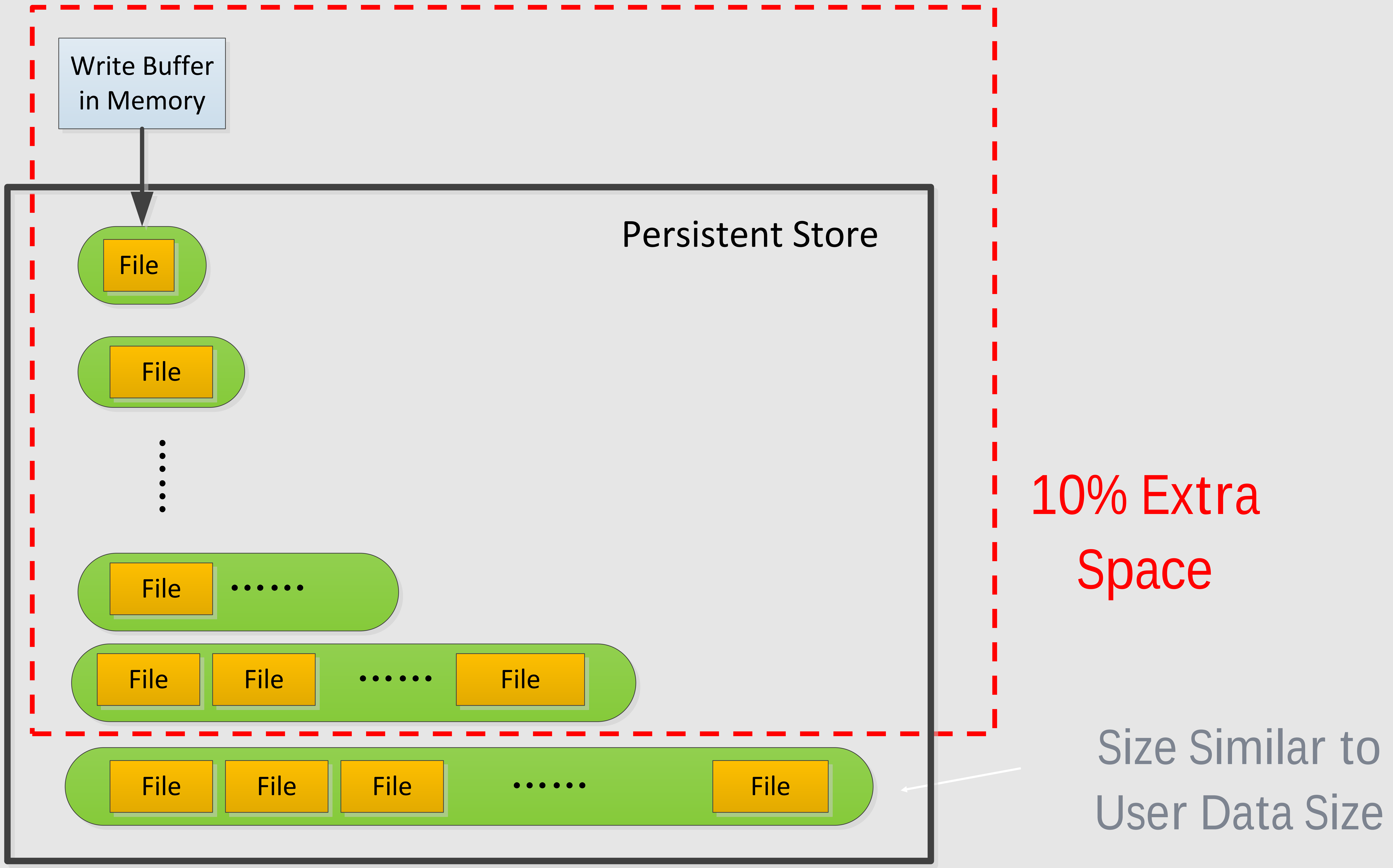
Everything except space has room to go!

Space Amplification of RocksDB

Only 10% Extra Space

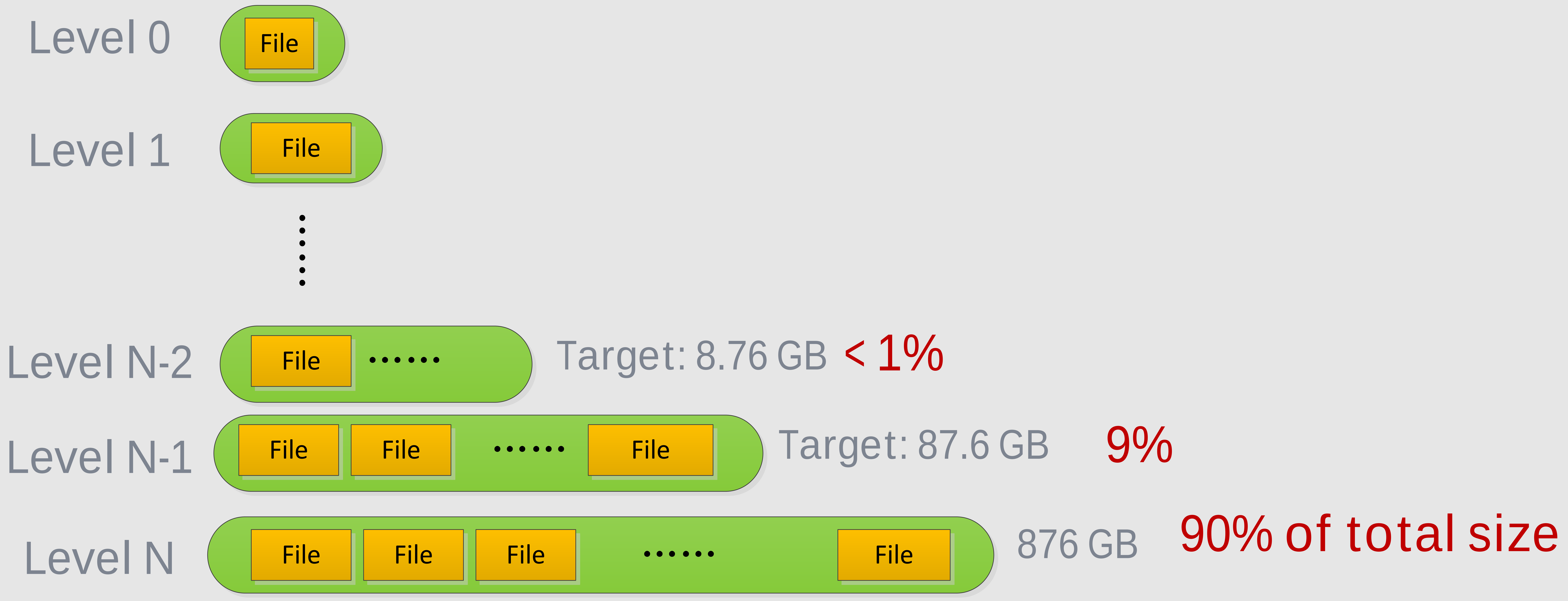
How?

Space efficiency in LSM?

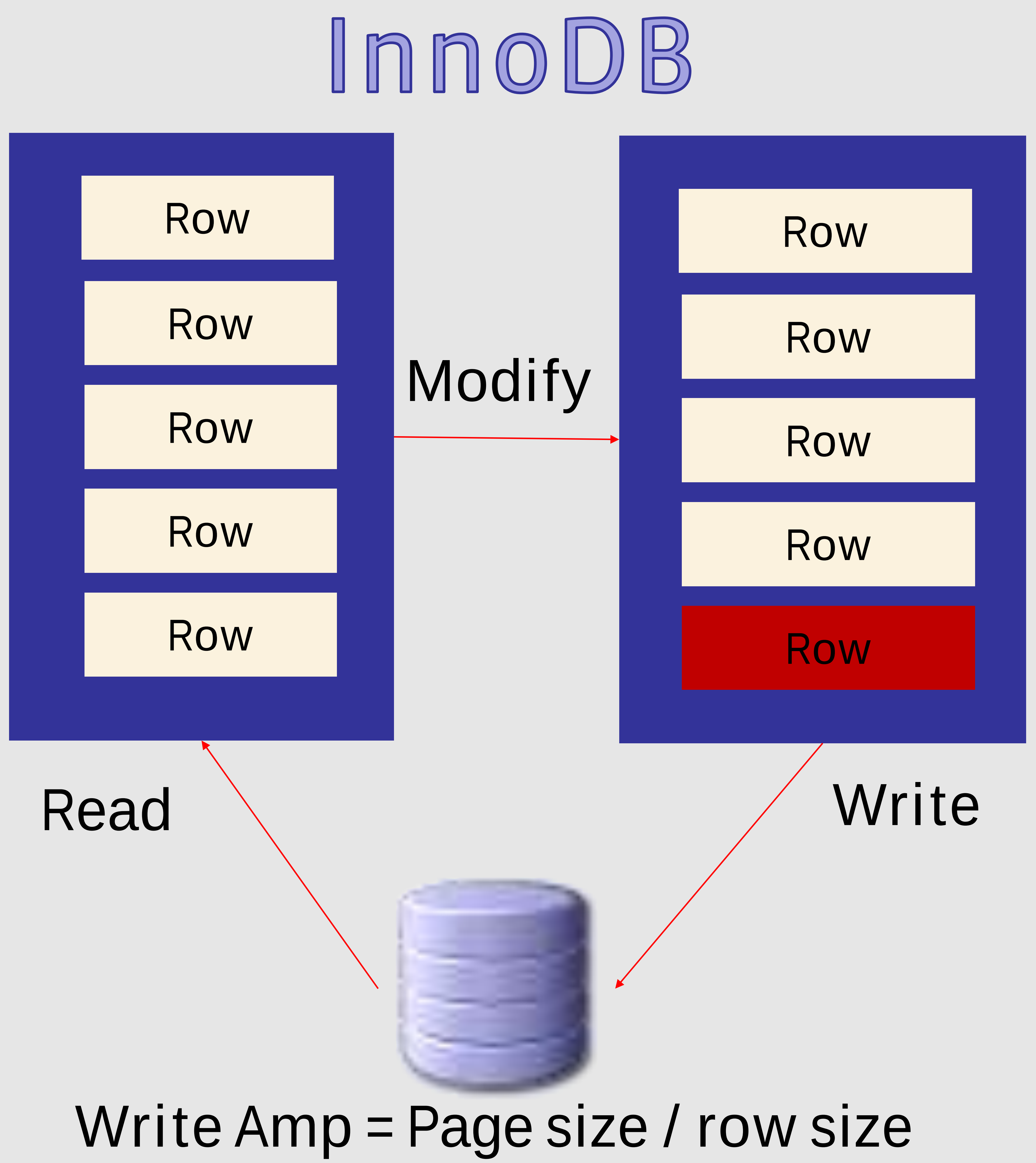


How Did We Guarantee 10%?

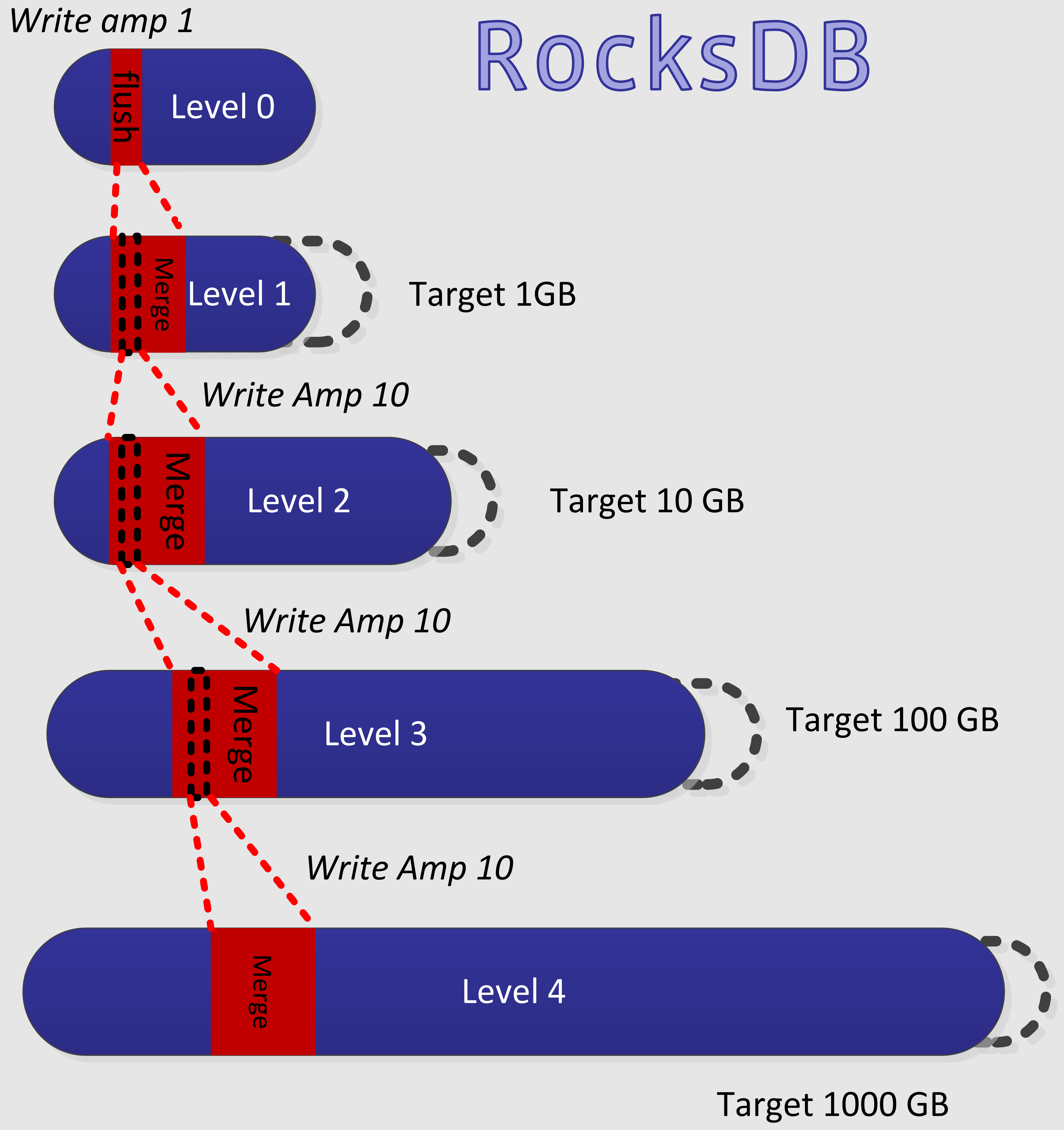
A Space-Efficient Approach



Lower Write Amplification



Write Amp = Page size / row size



How About Other Metrics?

- Read QPS
- Write Throughput

Make Read Throughput High: Reduced Locking in Reads

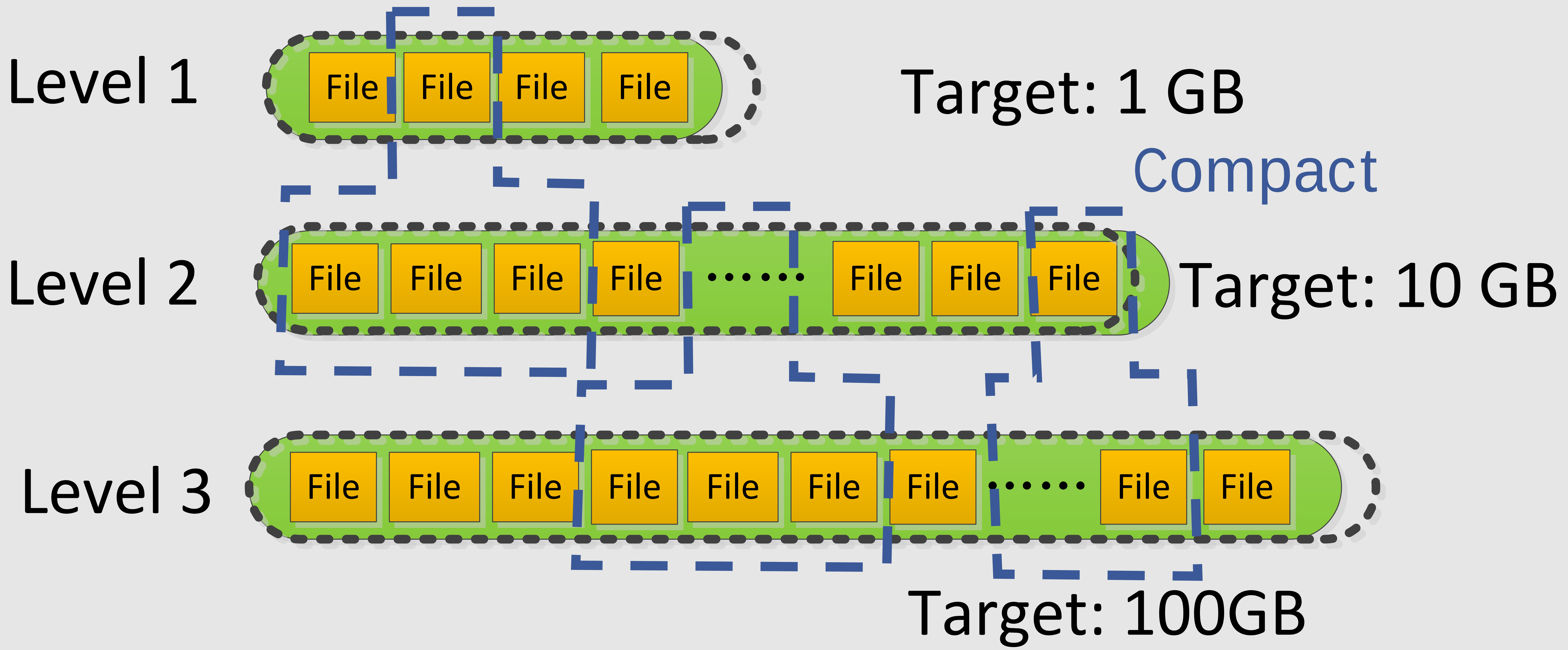
- Memtable: skip list
- Data Files: immutable
- LSM tree change: thread-local cache of the tree
- Synchronize opened files: allow to keep all files open
- Block cache mutex: sharded; more optimization coming.

Write Throughput

- Throughput of Compactions
- Throughput of Memtable Inserts

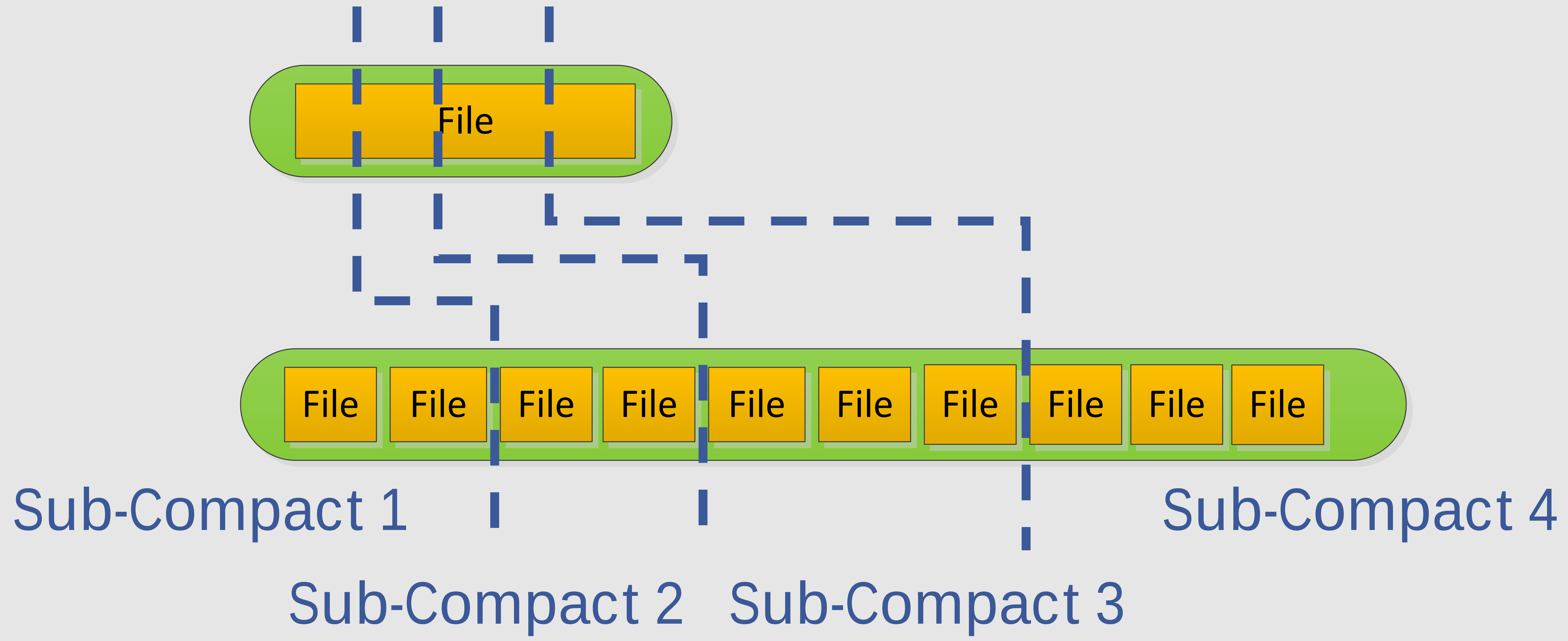
Multi-thread compactions

Compact non-overlapping files



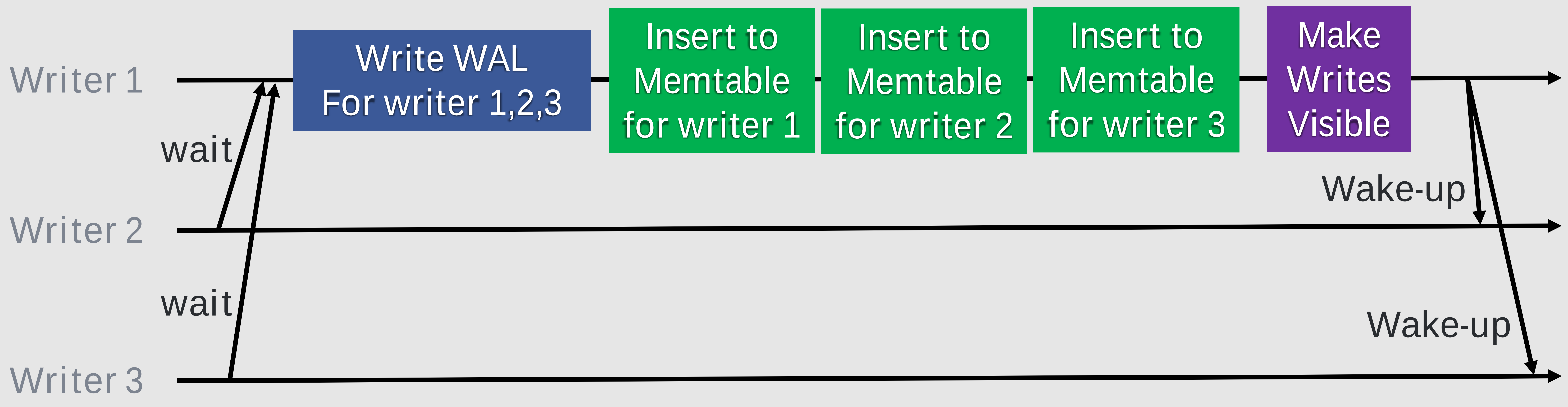
Multi-thread compactions

Divide One Compaction to sub-compactions



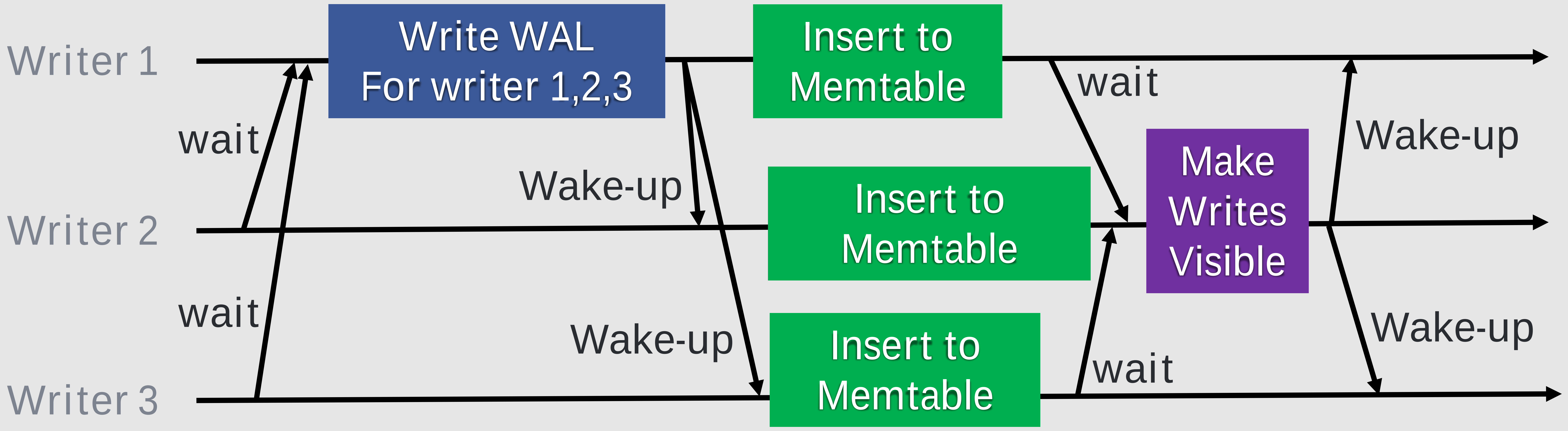
Parallel Memtable Insert

Previous Workflow of Group Commit


Writer 1
Writer 2
Writer 3
wait
wait
Write WAL
For writer 1,2,3
Insert to Memtable for writer 1
Insert to Memtable for writer 2
Insert to Memtable for writer 3
Make Writes Visible
Wake-up
Wake-up

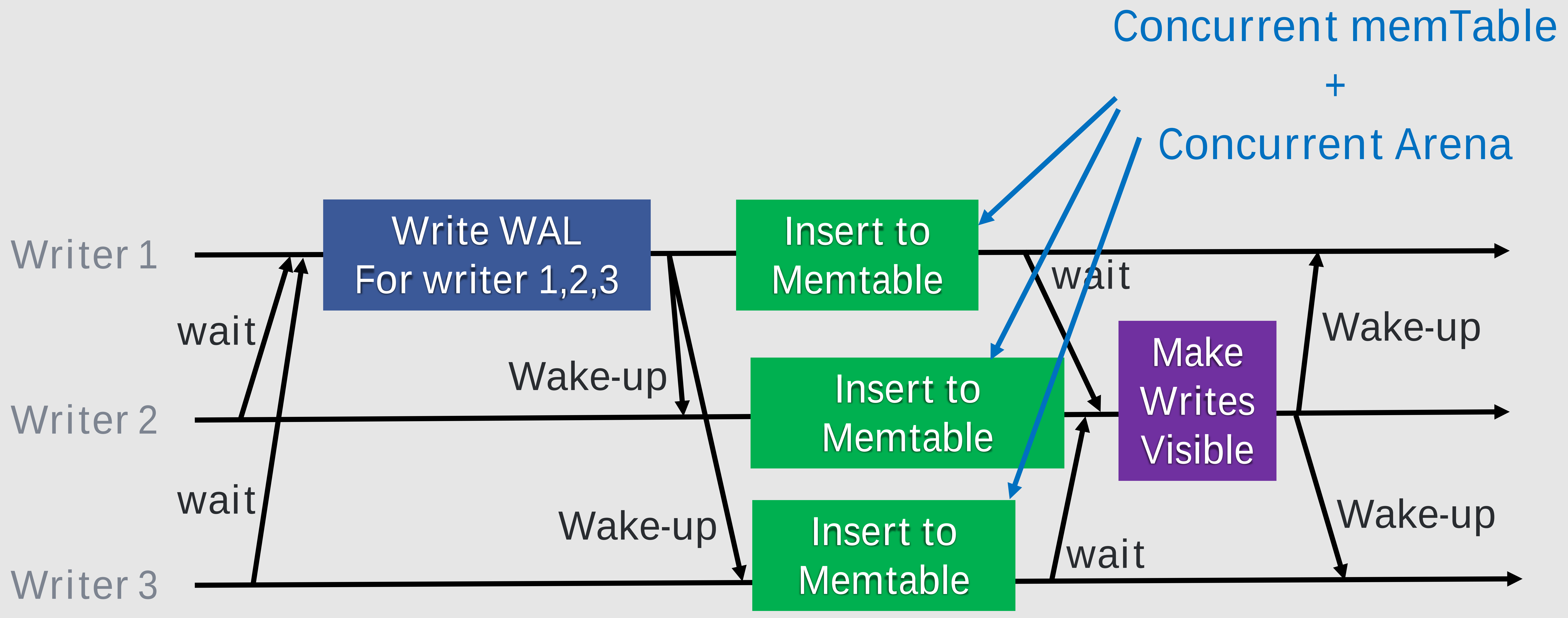
Parallel Memtable Insert

Parallel Memtable Insert for Group Commit



Parallel Memtable Insert

Parallel Memtable Insert for Group Commit



RocksDB Performance On Flash

- Space, Read And Write Amplificaton Trade-offs
- Low Space Amplification
- High Read QPS: Reduced Mutex Locking
- High Write Throughput: Parallel Compaction; Concurrent Memtable Insert

Other Storage Media?

RocksDB On Other Storage Media

- Memory-Only:
  - *Memory Efficiency*
  - *7 million reads/s in single host benchmark*
- Spinning Disk:
  - *Write-Optimized*
  - *Reasonable Read Performance*

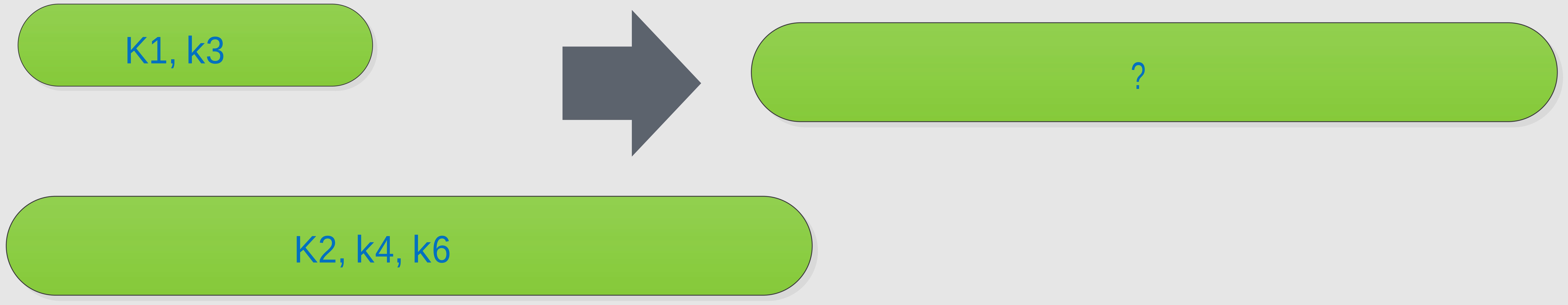
Other Benefits?

Features Enabled by LSM

- Compaction Filter
- Merge Operator

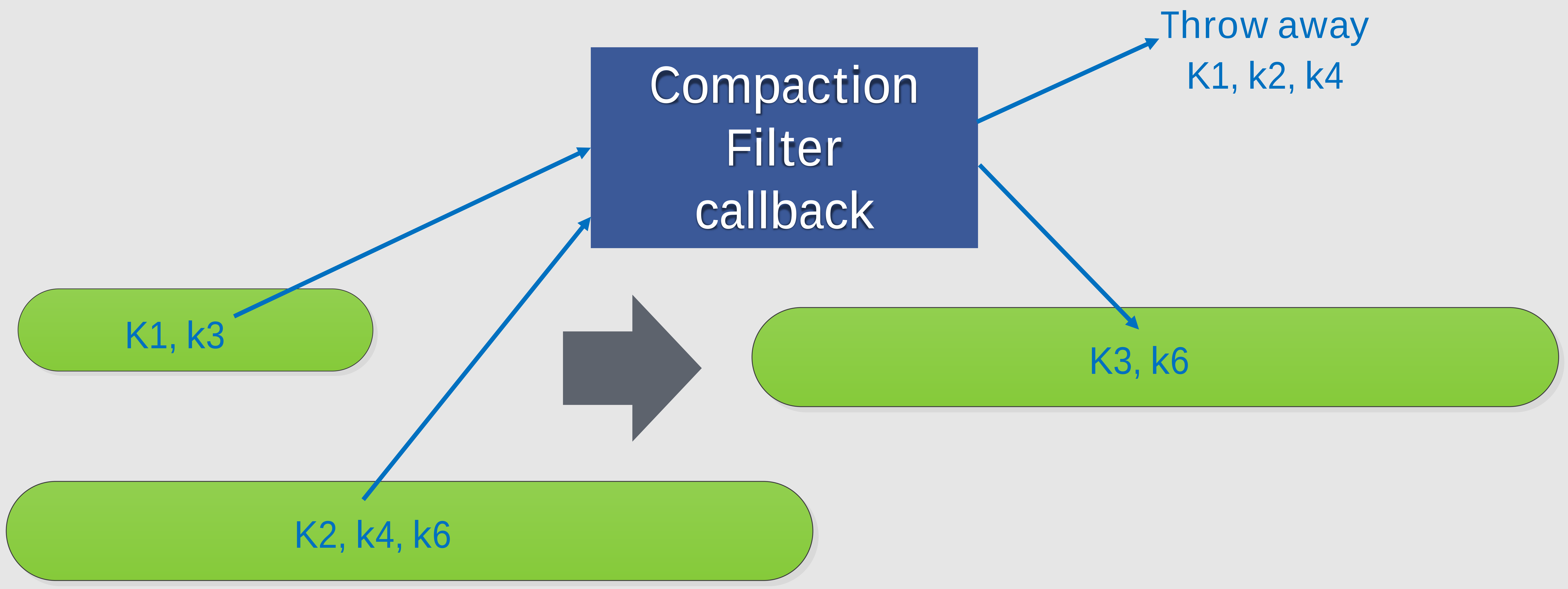
Compaction Filter

Expire Stale Keys In Compactions



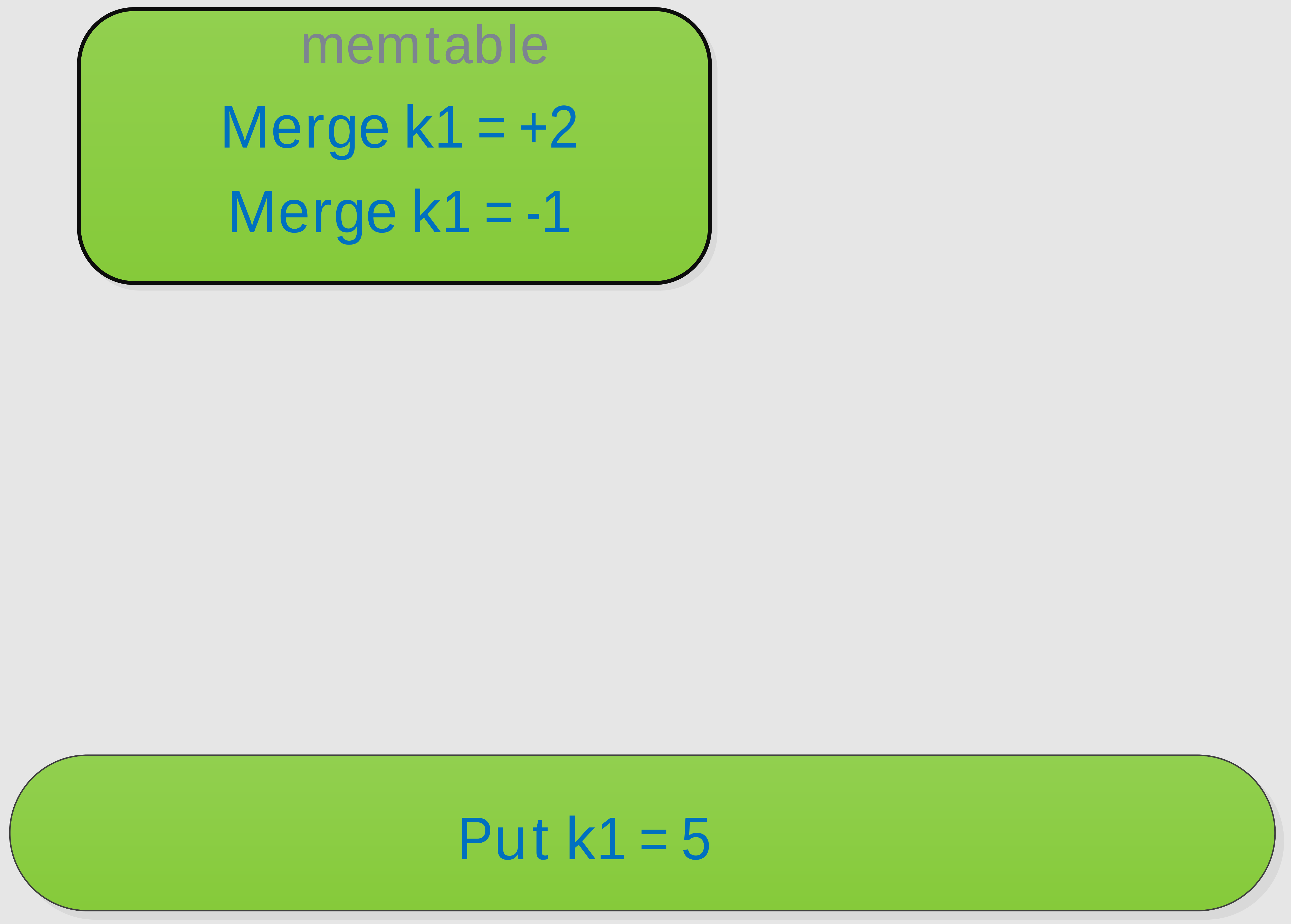
Compaction Filter

Expire Stale Keys In Compactions

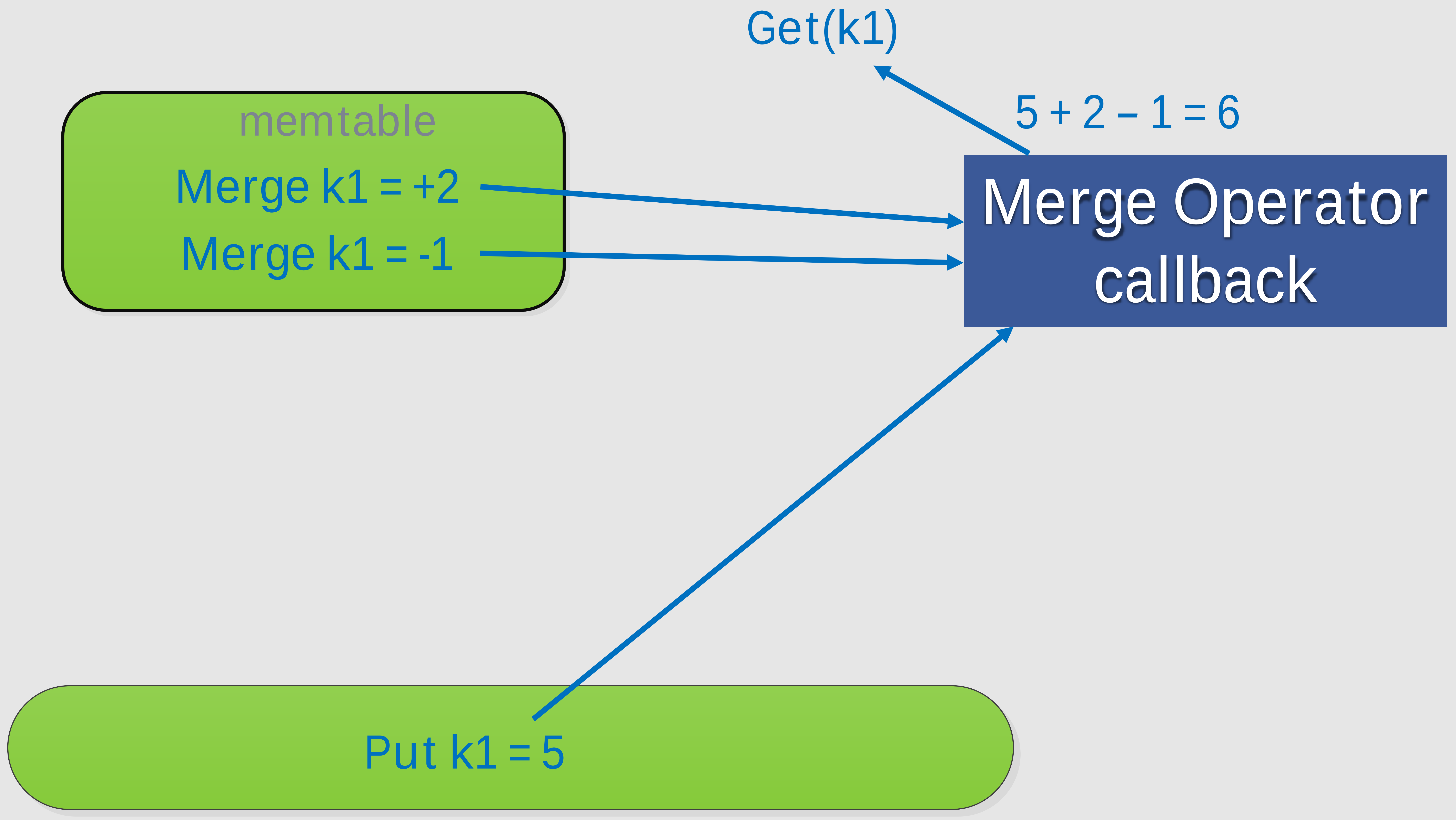


Merge Operator

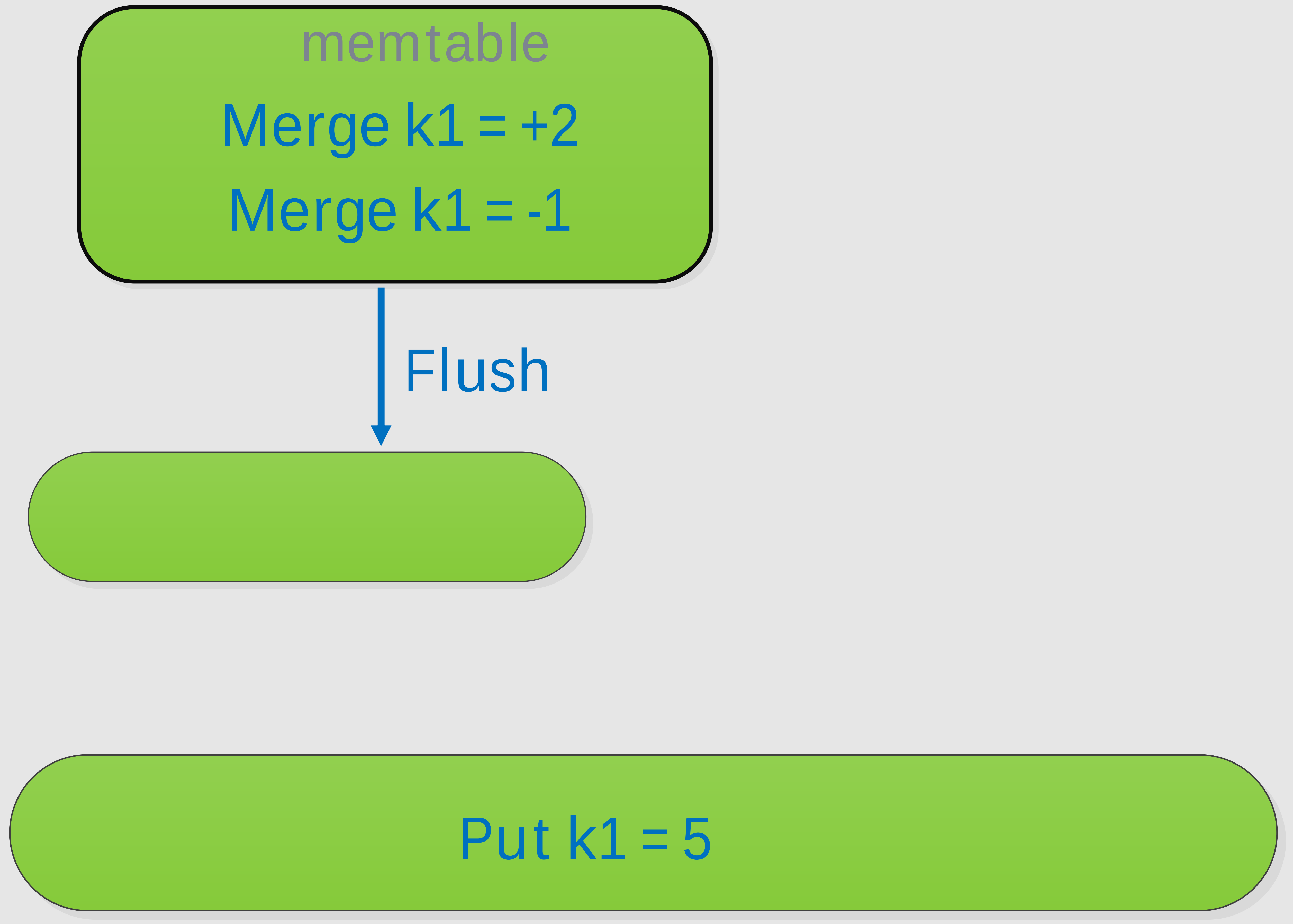
Avoid Read-Modify-Write



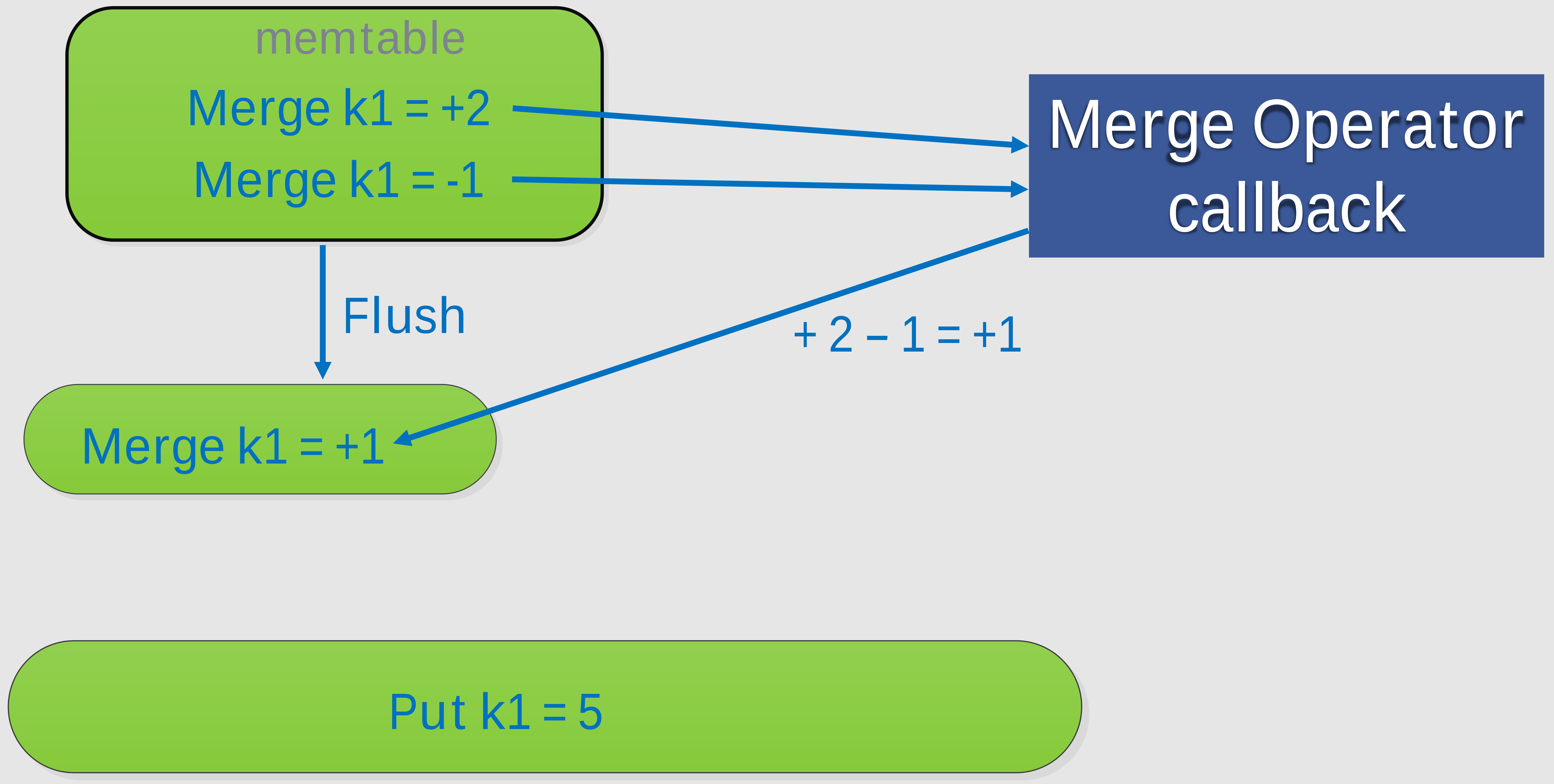
Merge Operator



Merge Operator



Merge Operator

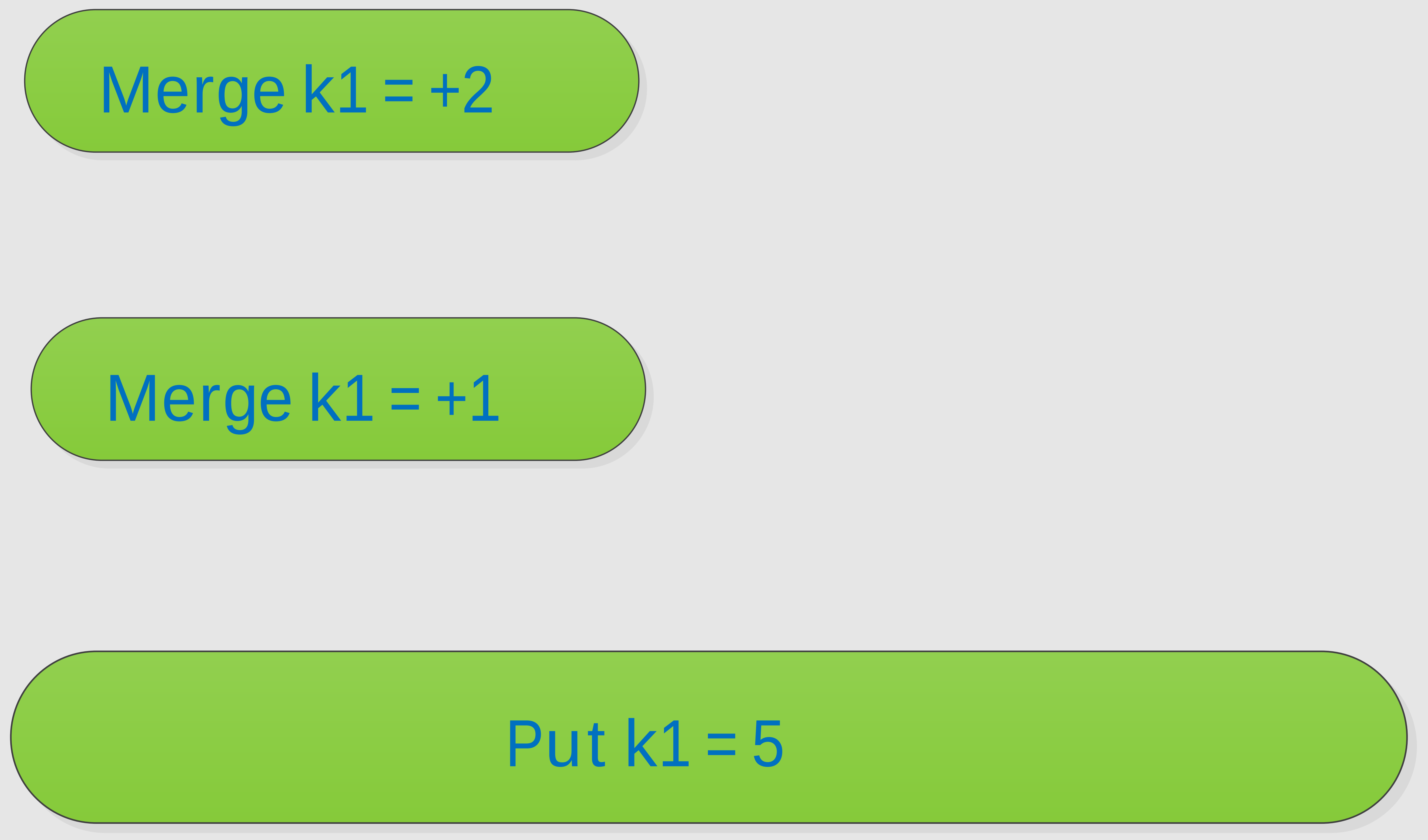


Merge Operator

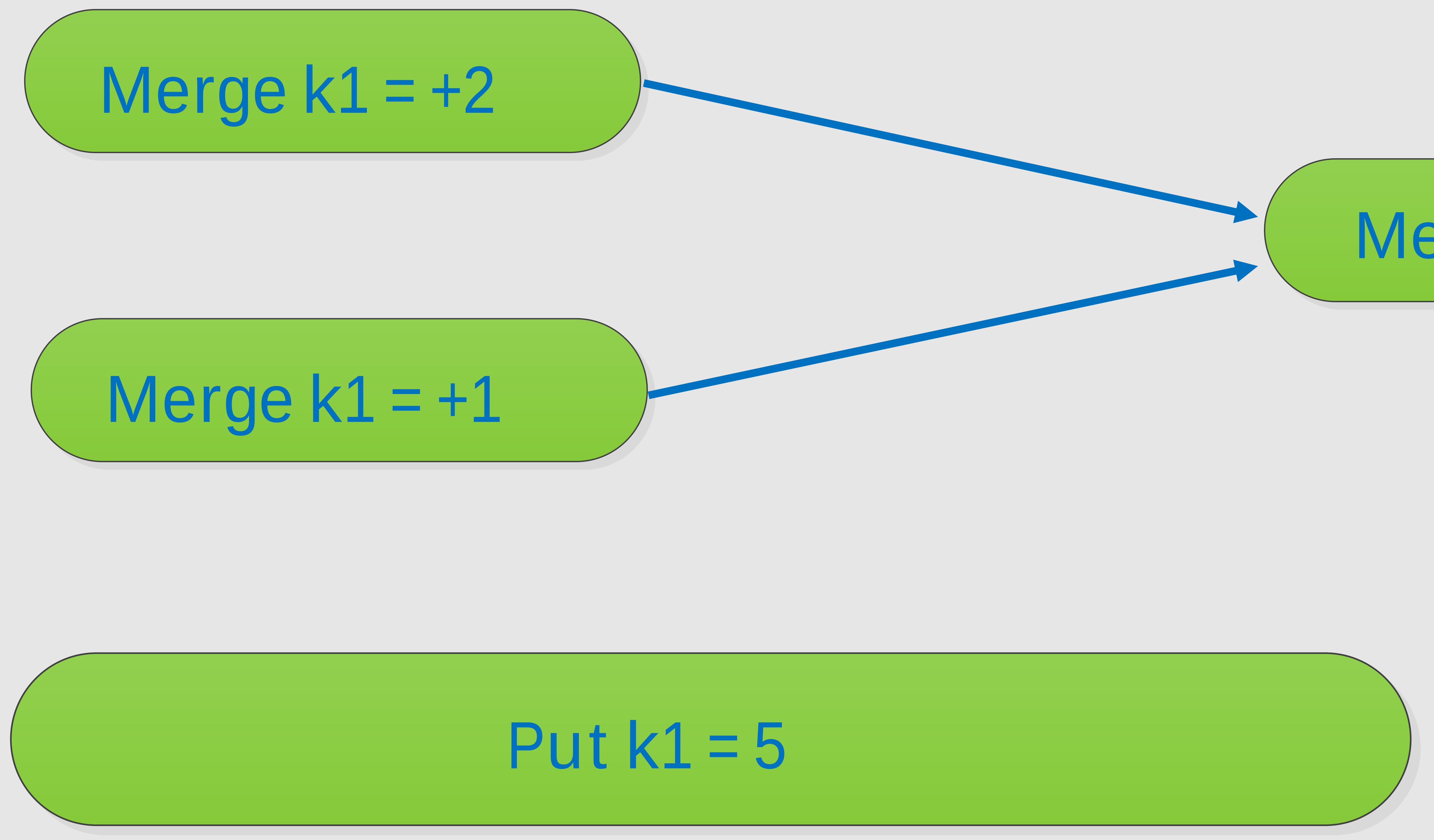
Merge k1 = +1

Put k1 = 5

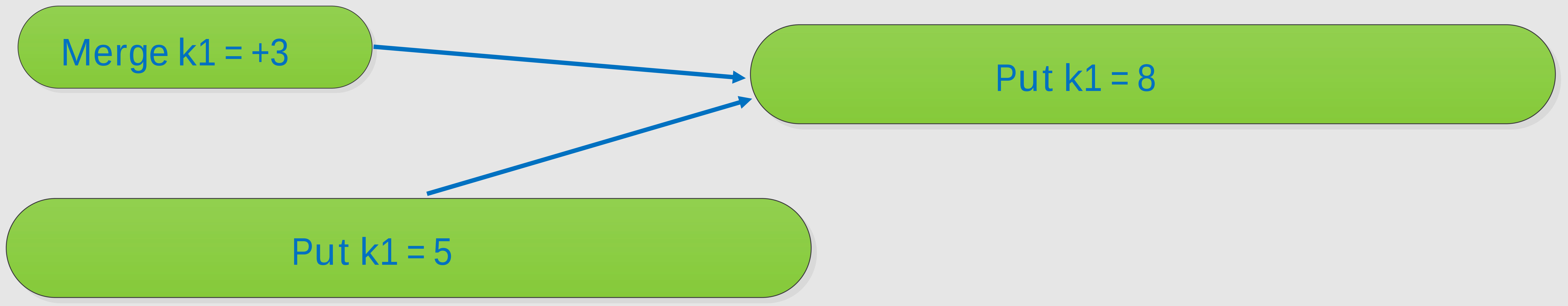
Merge Operator



Merge Operator



Merge Operator



Other Features

Other Features

- Transactions
- Column Families
- Monitoring
- And many more...

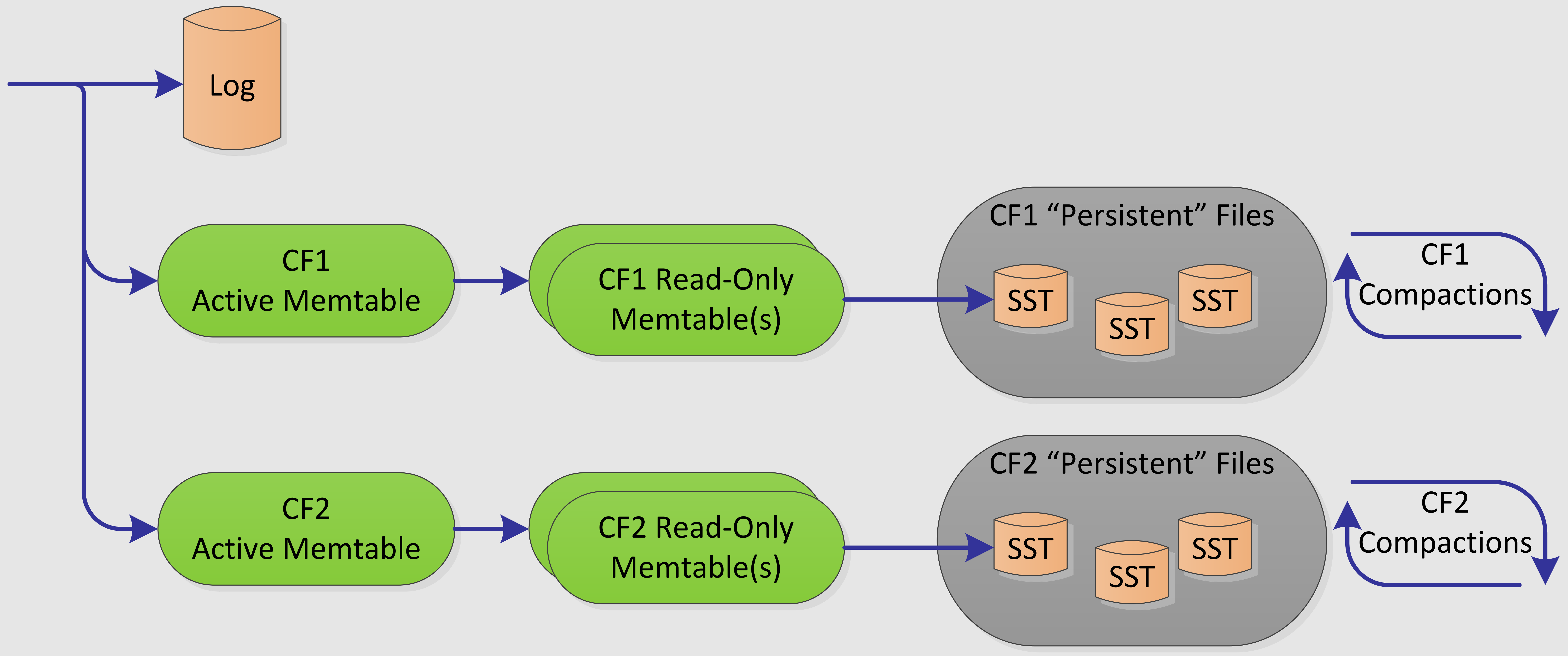
Transactions

- Optimistic and Pessimistic Transactions
- Optimistic: verify conflicts with memtable values
- Pessimistic: lock keys being modified.

Column Families

- Separate key spaces
- Allow atomic multiput, multiget, transactions
- Can apply different compaction setting, comparators, compaction filters, merge operators, etc.

Column Families



Monitoring

- Statistics
- DB Properties
- Per request profiling (perf context)
- User defined per SST file properties

Conclusion

- RocksDB is widely used
- RocksDB uses LSM-tree
- RocksDB is highly tunable for flash
- RocksDB can be tuned to be space efficient
- RocksDB has good performance
- RocksDB has nice features

Thank You!

- Portal: http://rocksdb.org/
- Github: https://github.com/facebook/rocksdb
- Discussion Group: https://www.facebook.com/groups/rocksdb.dev/