本文是作者在ACMUG 2016 MySQL年会上的演讲内容，版权归作者所有。

中国MySQL用户组（China MySQL User Group）简称ACMUG。ACMUG是覆盖中国MySQL技术爱好者的一个技术社区，是Oracle User Group Community和MairaDB Foundation共同认可的MySQL技术社区。

我们关注MySQL，MariaDB，以及其他一切周边的开源数据库和开源工具，我们交流使用经验，推广开源技术，为开源贡献力量。

我们是开放社区，欢迎任何关注MySQL及其相关技术的人加入，我愿意跟其他任何技术组织和团体保持沟通和展开合作。

我们期望在我们的活动中大家都能以开心的、轻松的姿态交流技术，分享技术，形成一个良性循环，从而每个人都可以有一份收获。

ACMUG的口号：开源，开放，开心

关注ACMUG公众号，参与社区活动，交流开源技术，分享学习心得，一起共同进步。

RocksDB

Key-Value Store Optimized For Flash

Siying Dong

Software Engineer, Database Engineering Team @ Facebook

Dec 10, 2016

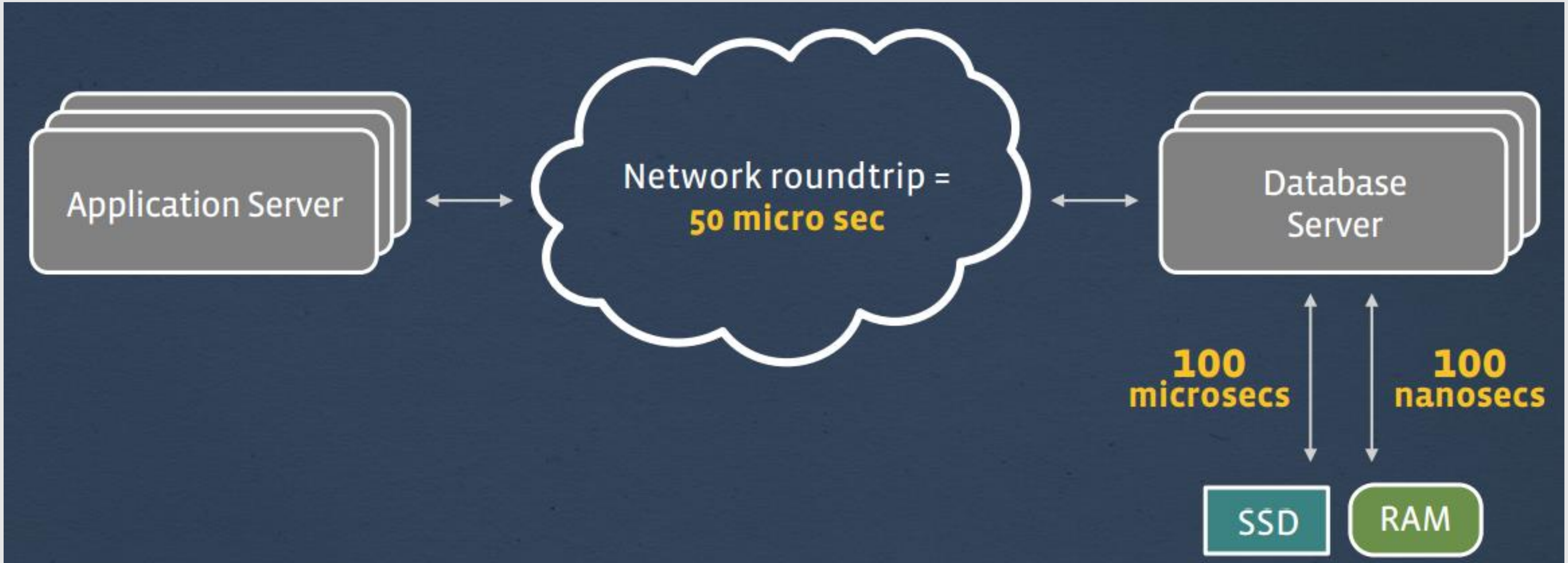
Agenda

1 What is RocksDB?

2 RocksDB Design

3 RocksDB Is Flash-Friendly

What is RocksDB?



Application Server

Network roundtrip =
50 micro sec

Database
Server

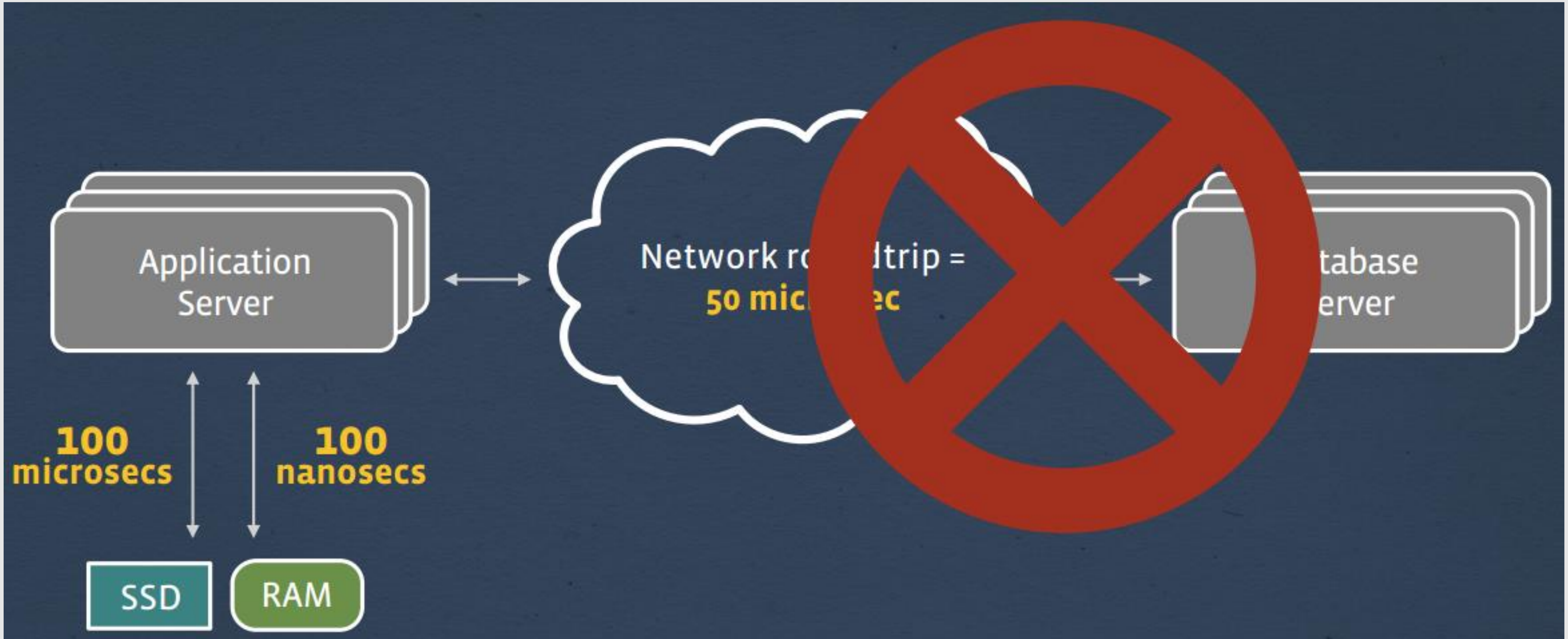
100
microsecs

100
nanosecs

SSD

RAM

Latency dominated by network



Application Server

100 microsecs

100 nanosecs

SSD

RAM

Network ro dtrip =

50 mic ec

tabase erver

Storage attached directly to application servers

What is RocksDB?

- Fork of LevelDB
- Key-Value persistent store
- Point / range lookup
- C++ library

RocksDB As Embedded Storage

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

RocksDB As Storage Engine of Data Management Systems

mongoDB

RocksDB Mmap WiredTiger

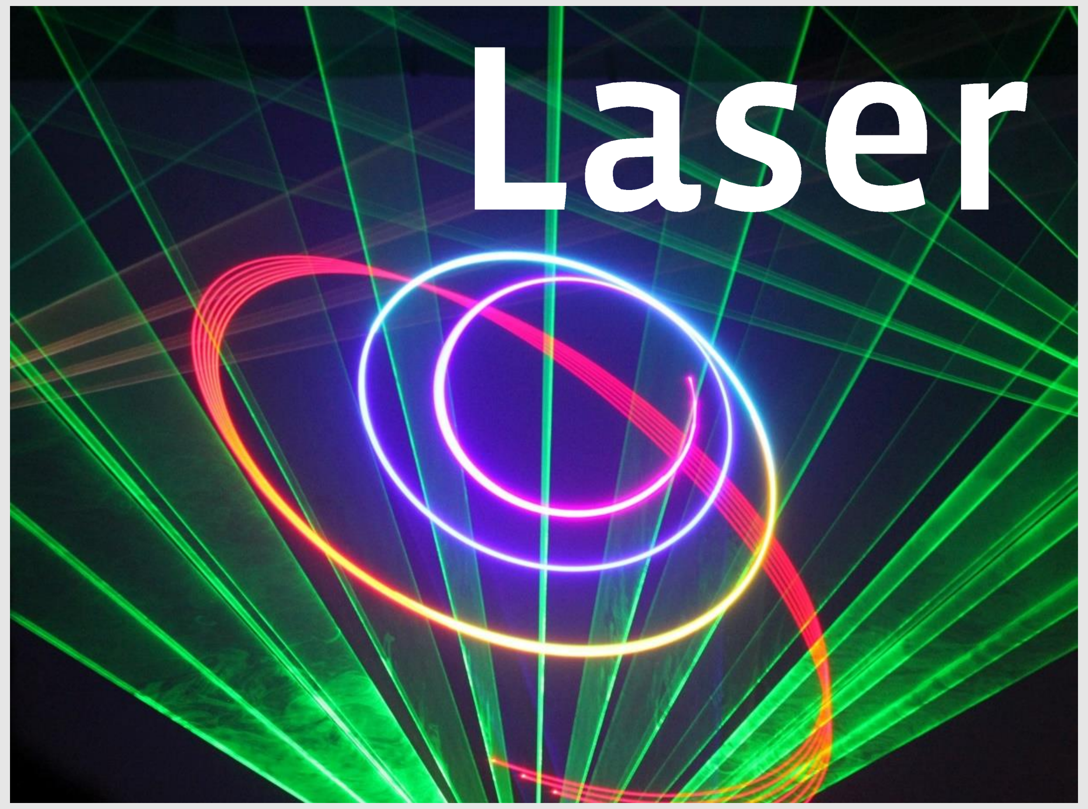
MySQL®

MyISAM ...... InnoDB RocksDB

Yahoo Sherpa

RocksDB

ZippyDB

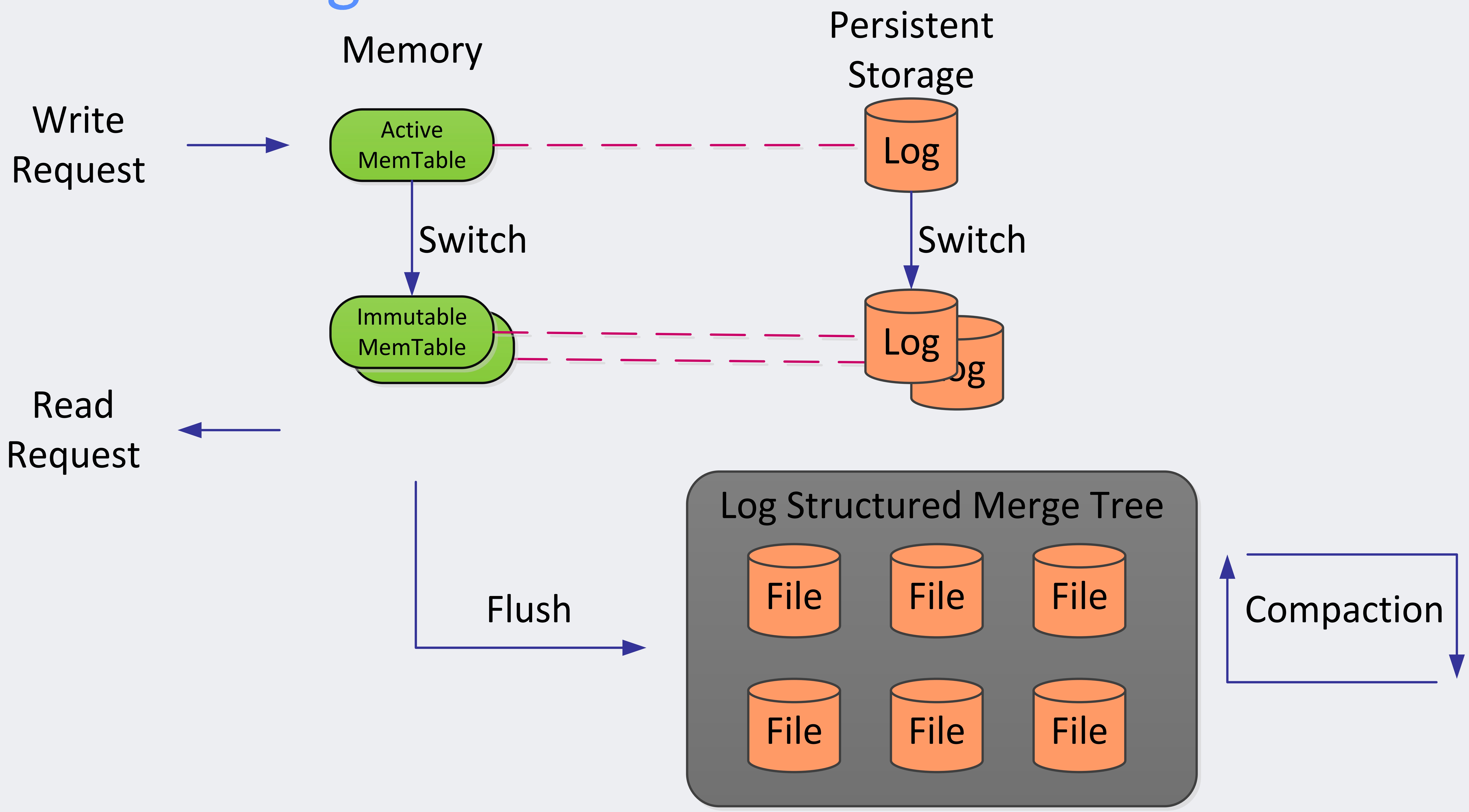

Laser

And many more ...

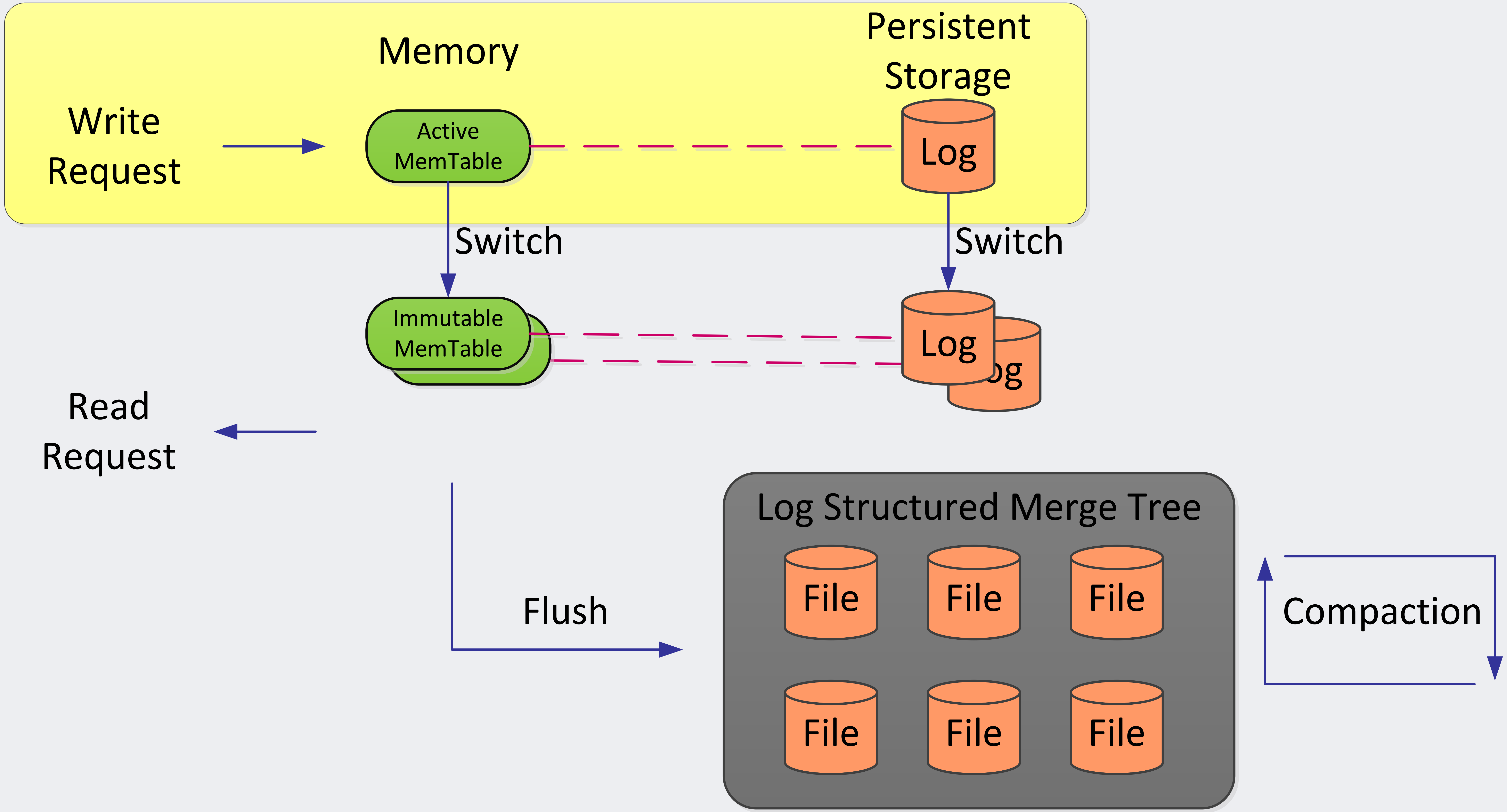
RocksDB Design

RocksDB Architecture

Log-Structured Merge-Tree

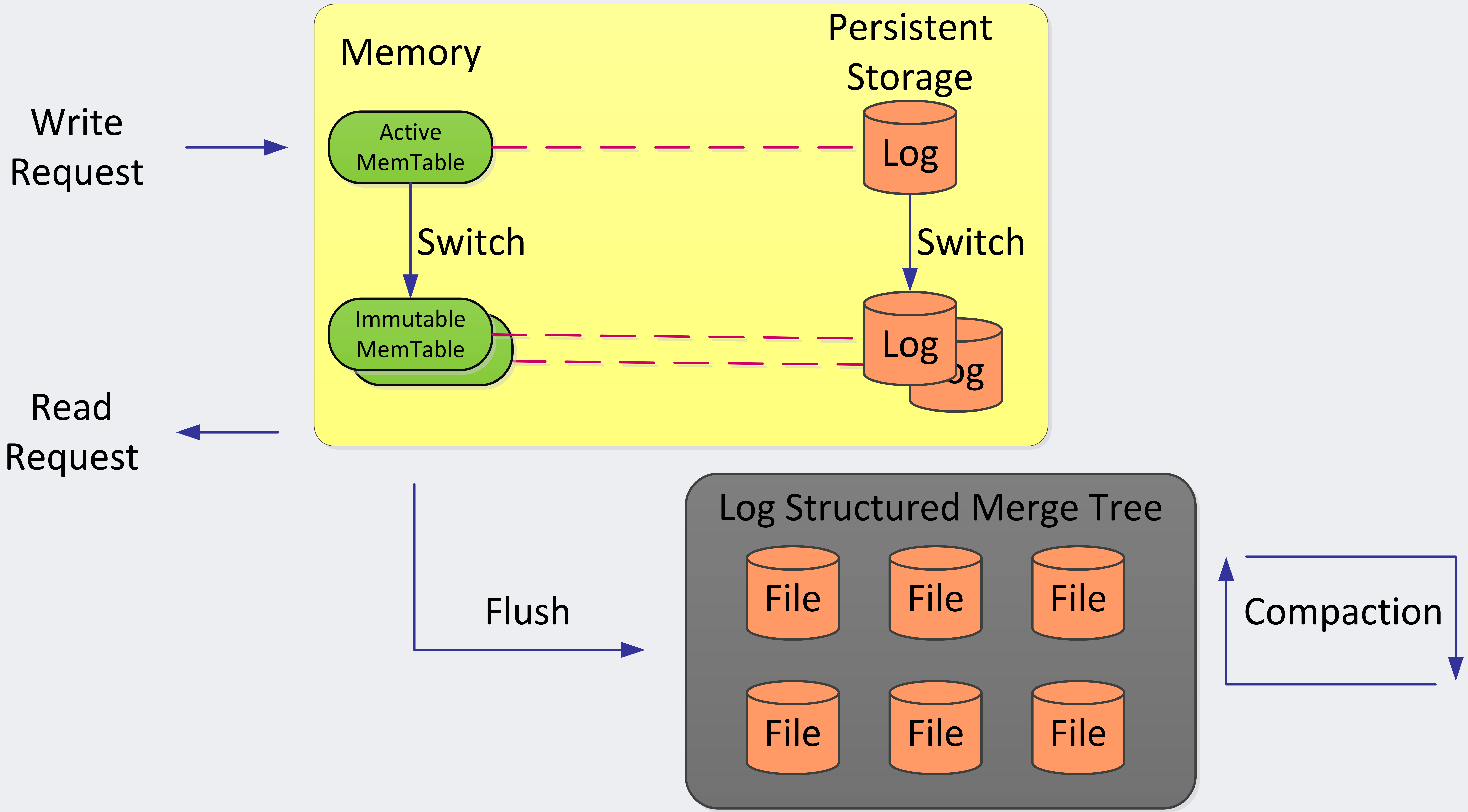


Write Path (1)

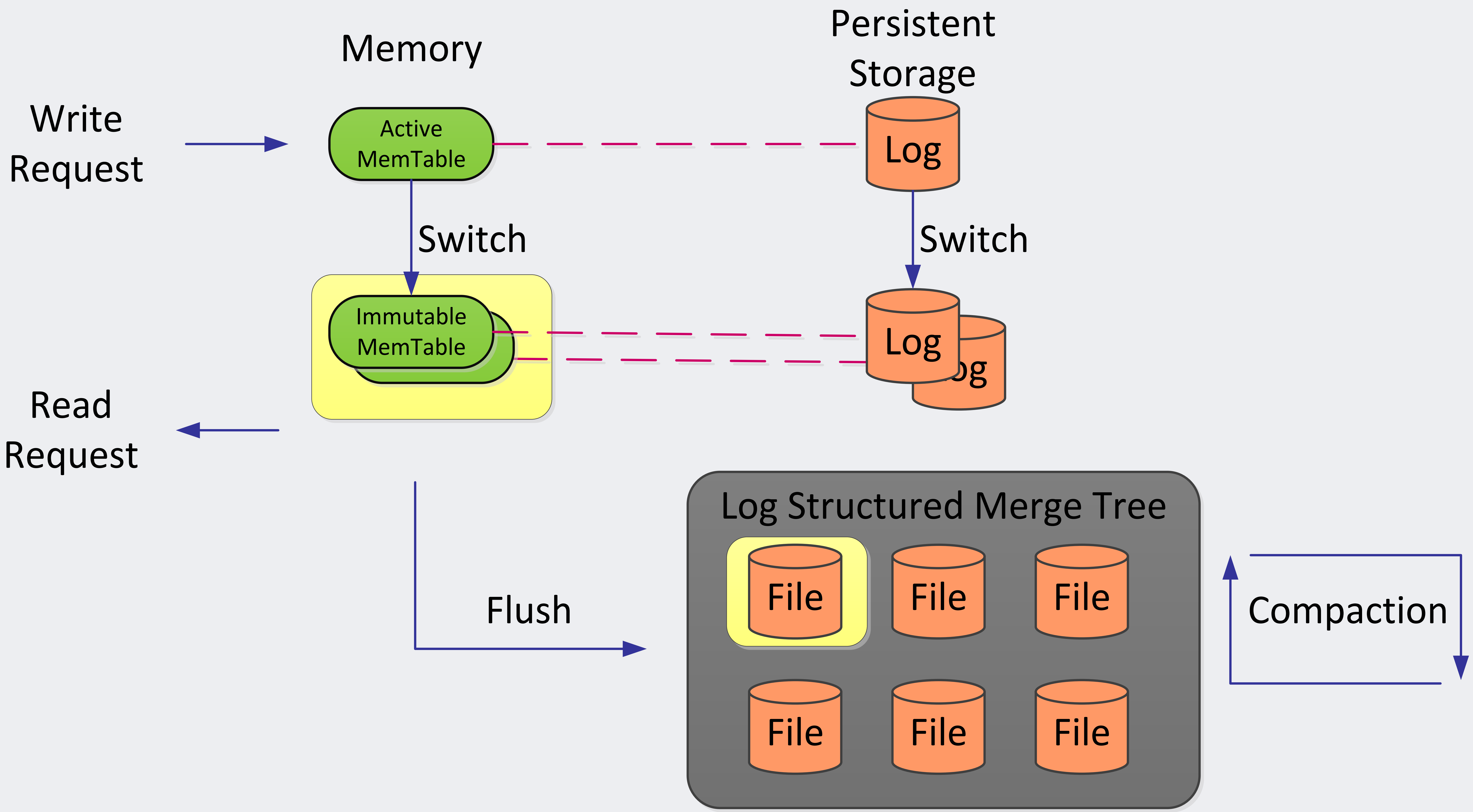


Memory
Persistent Storage
Write Request
Active MemTable
Log
Switch
Switch
Immutable MemTable
Log
Log
Read Request
Flush
Log Structured Merge Tree
File
File
File
File
File
File
Compaction

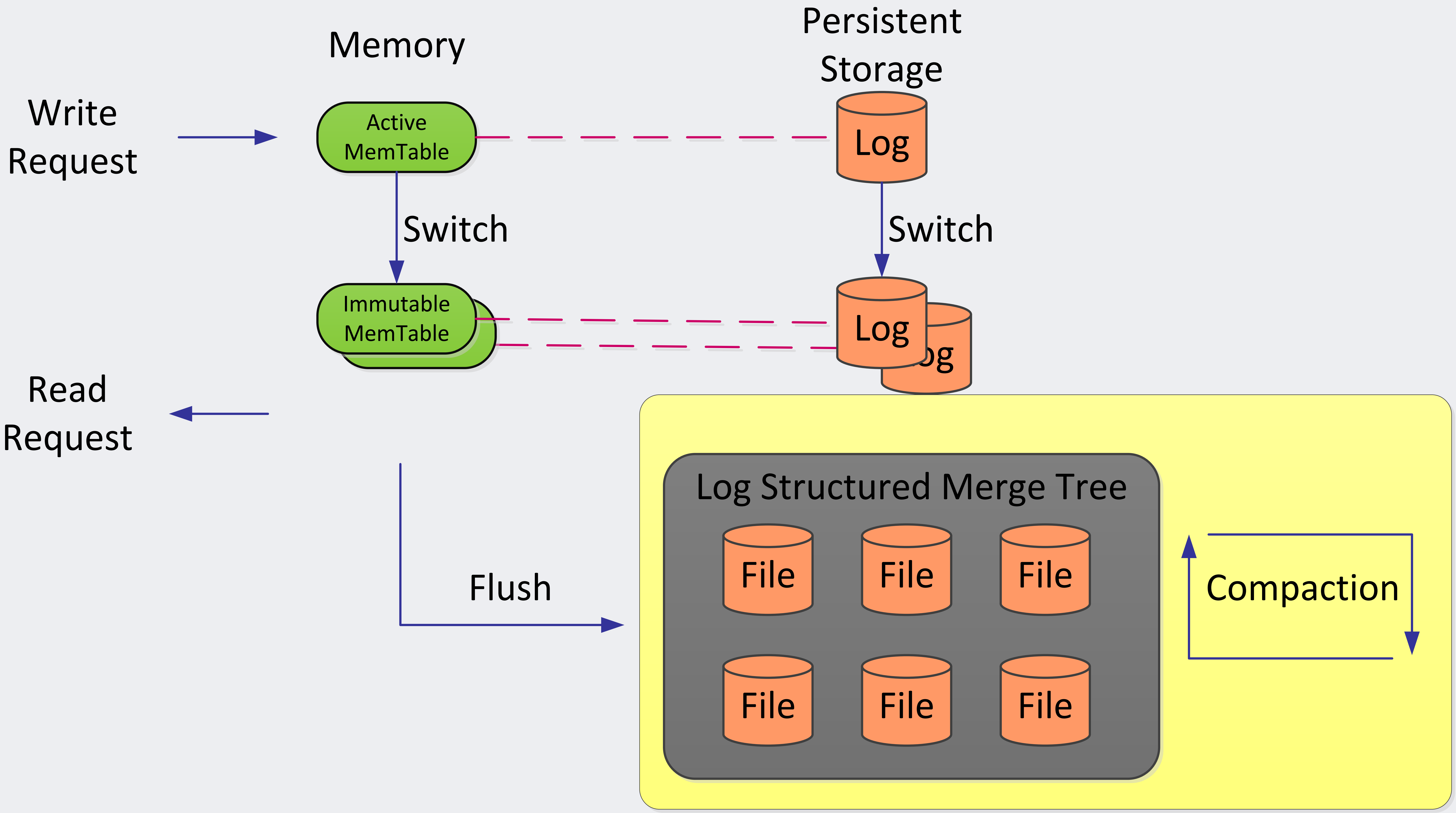
Write Path (2)


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

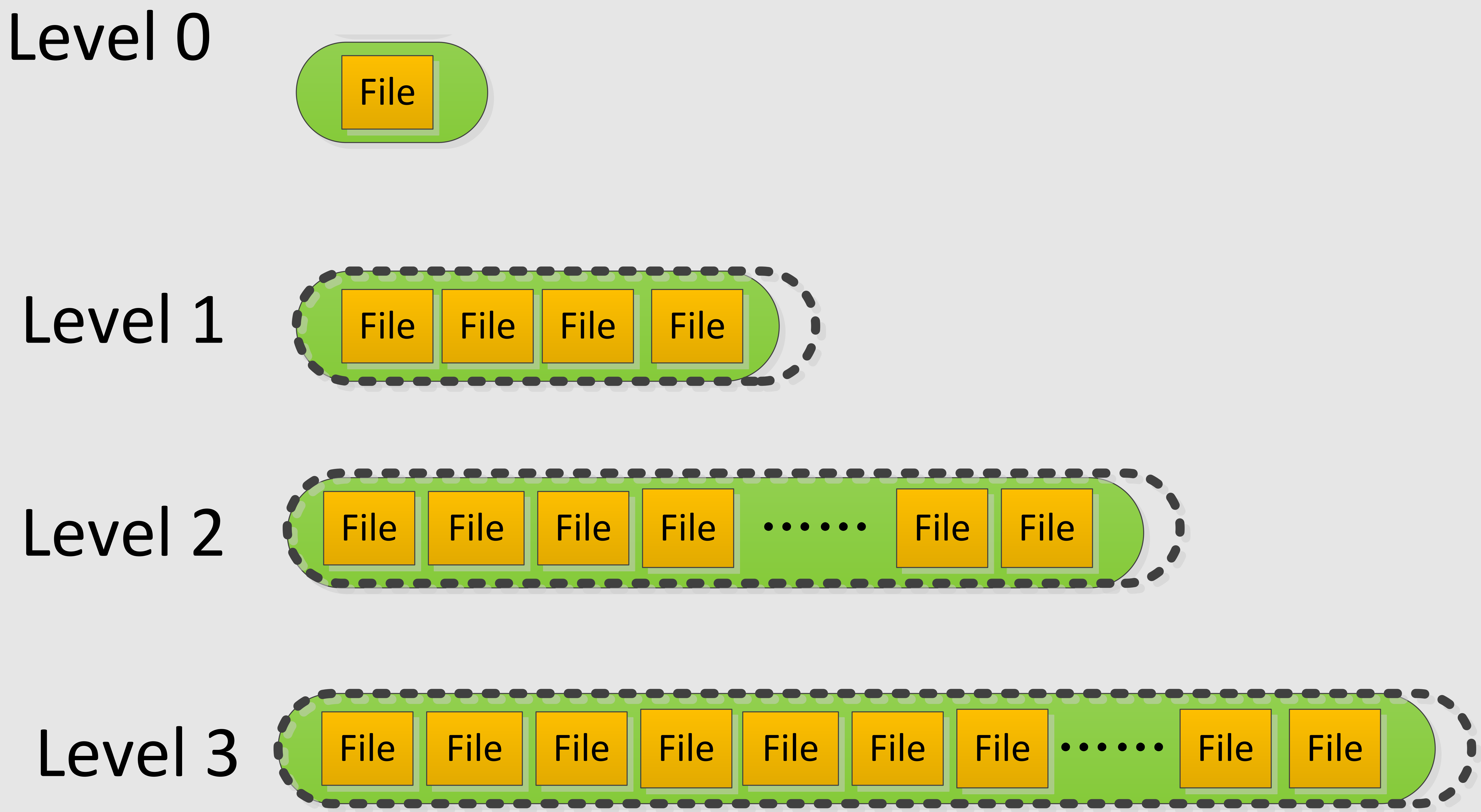
Write Path (3)


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

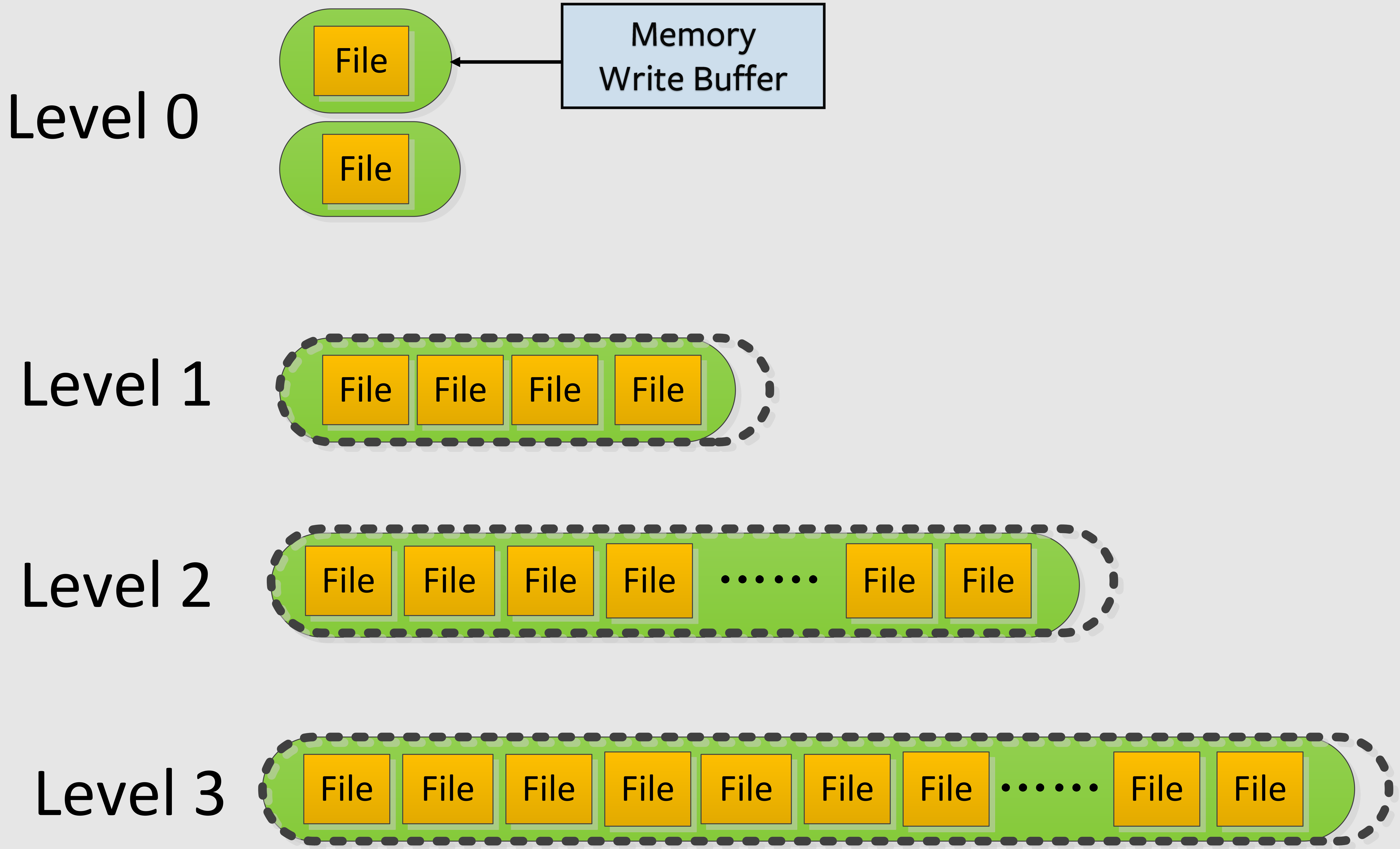
Write Path (4)


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
Compaction
Flush

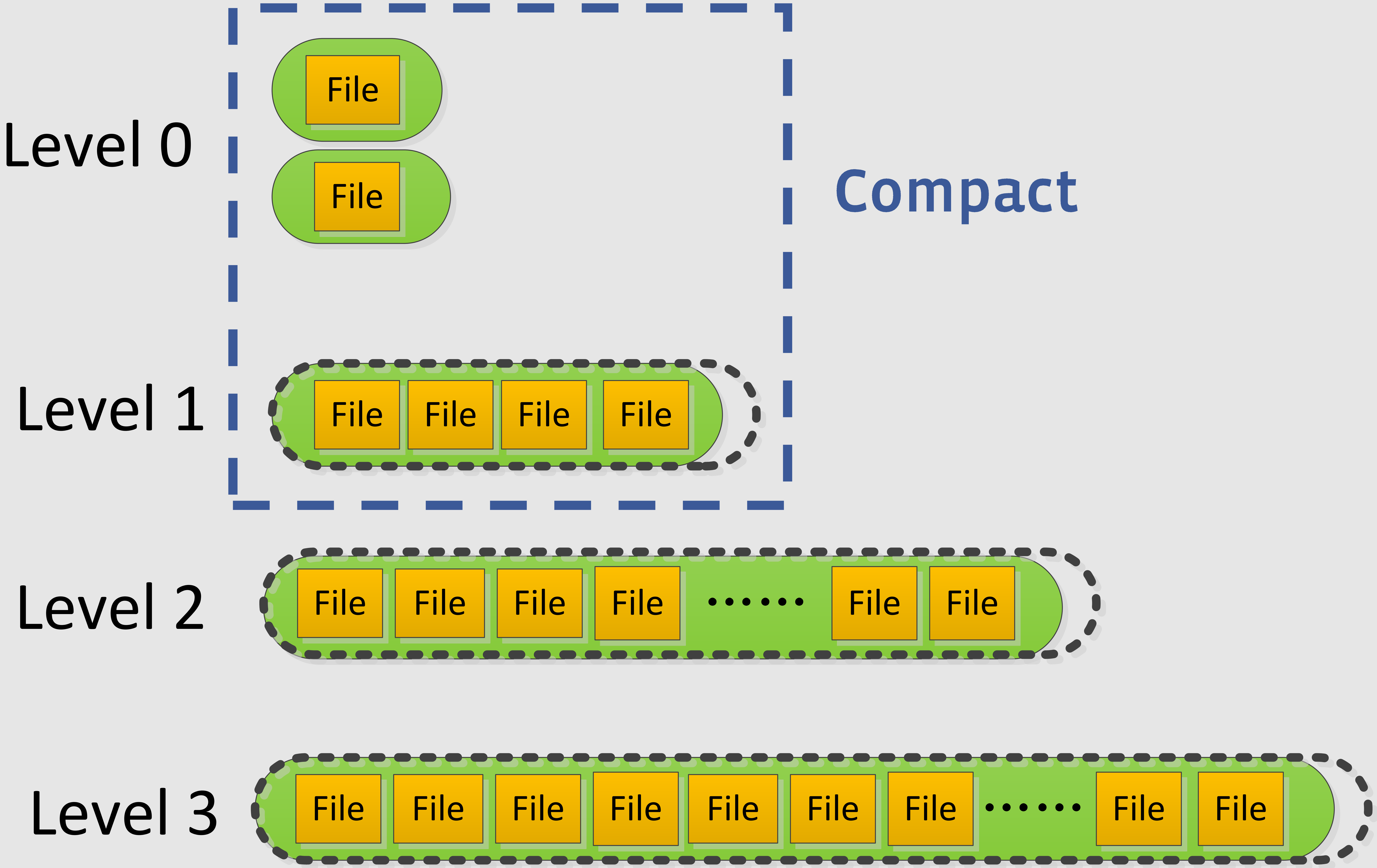
Level-Based Compaction



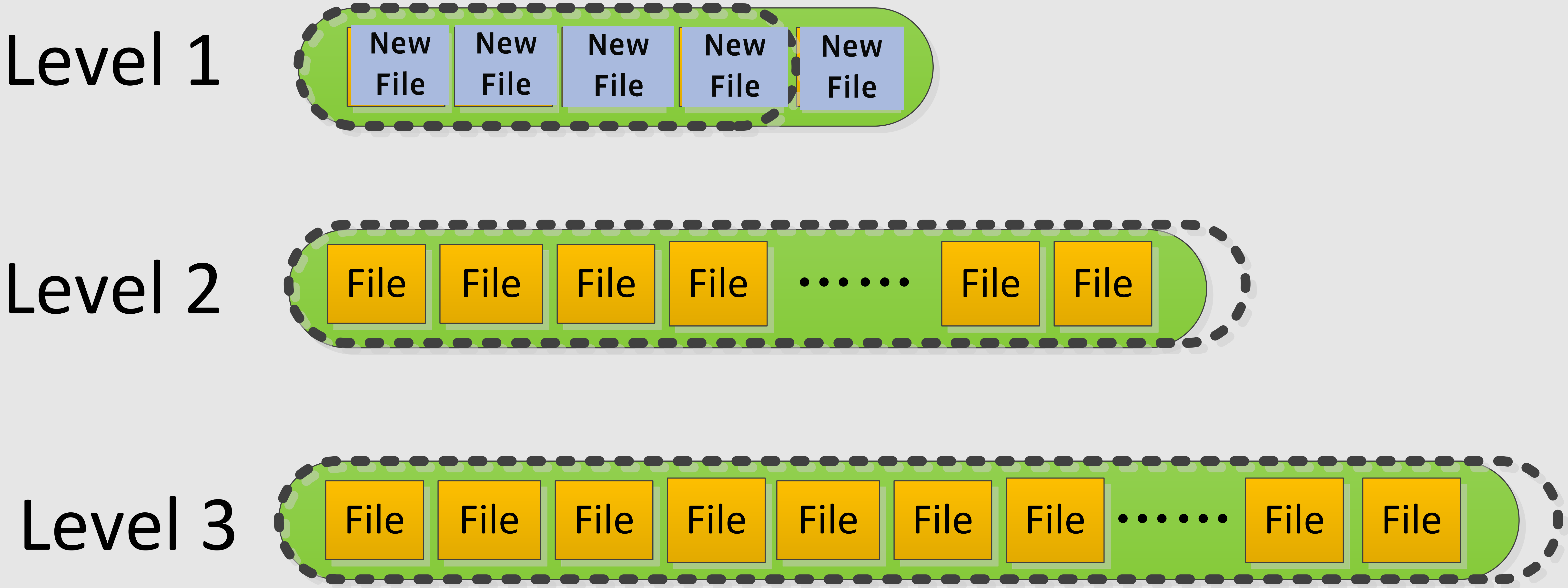
Level-Based Compaction


Level 0
File
File
Memory Write Buffer
Level 1
File File File File
Level 2
File File File File ······ File File
Level 3
File File File File File File File ······ File File

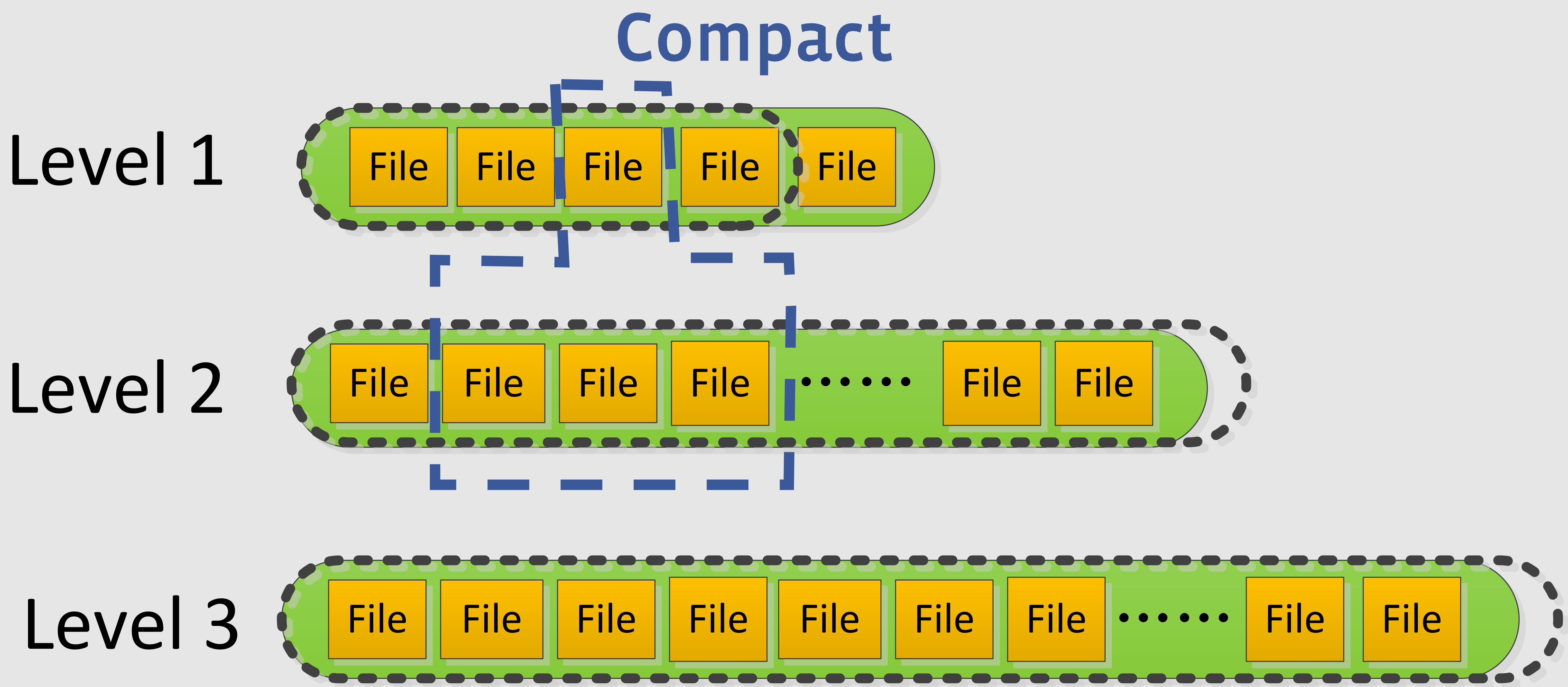
Level-Based Compaction



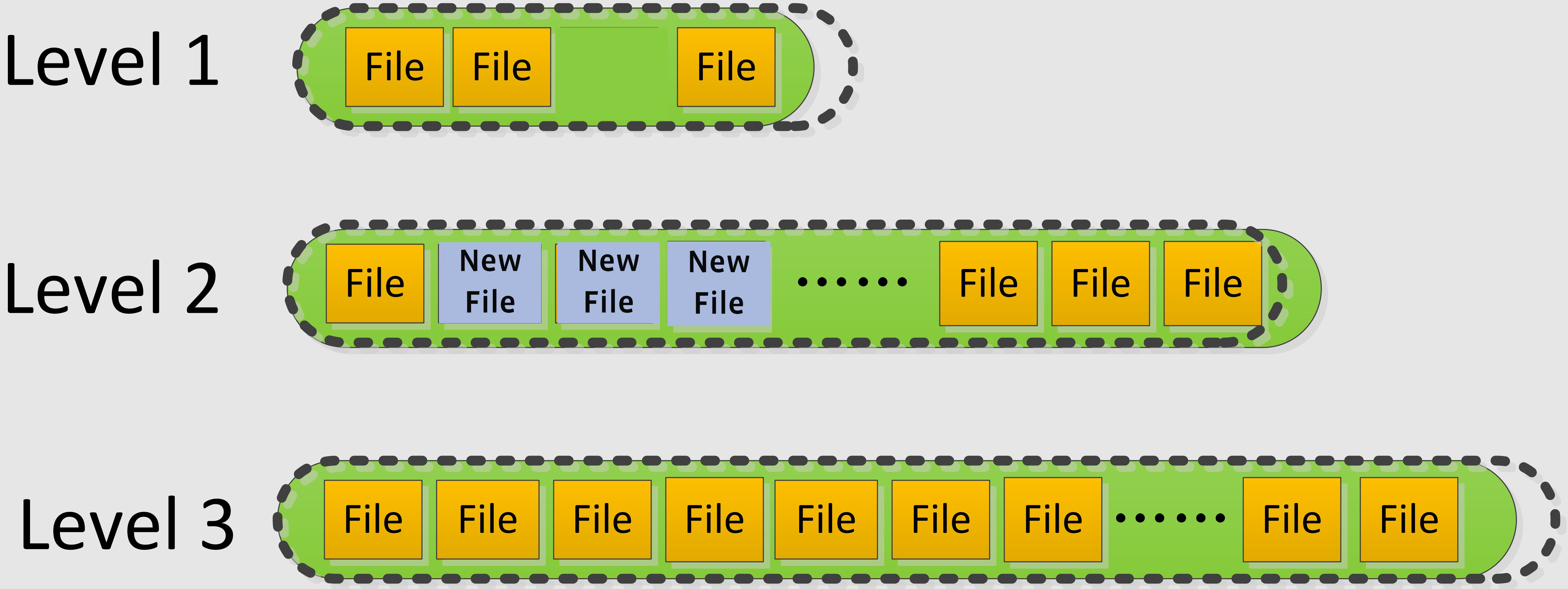
Level-Based Compaction



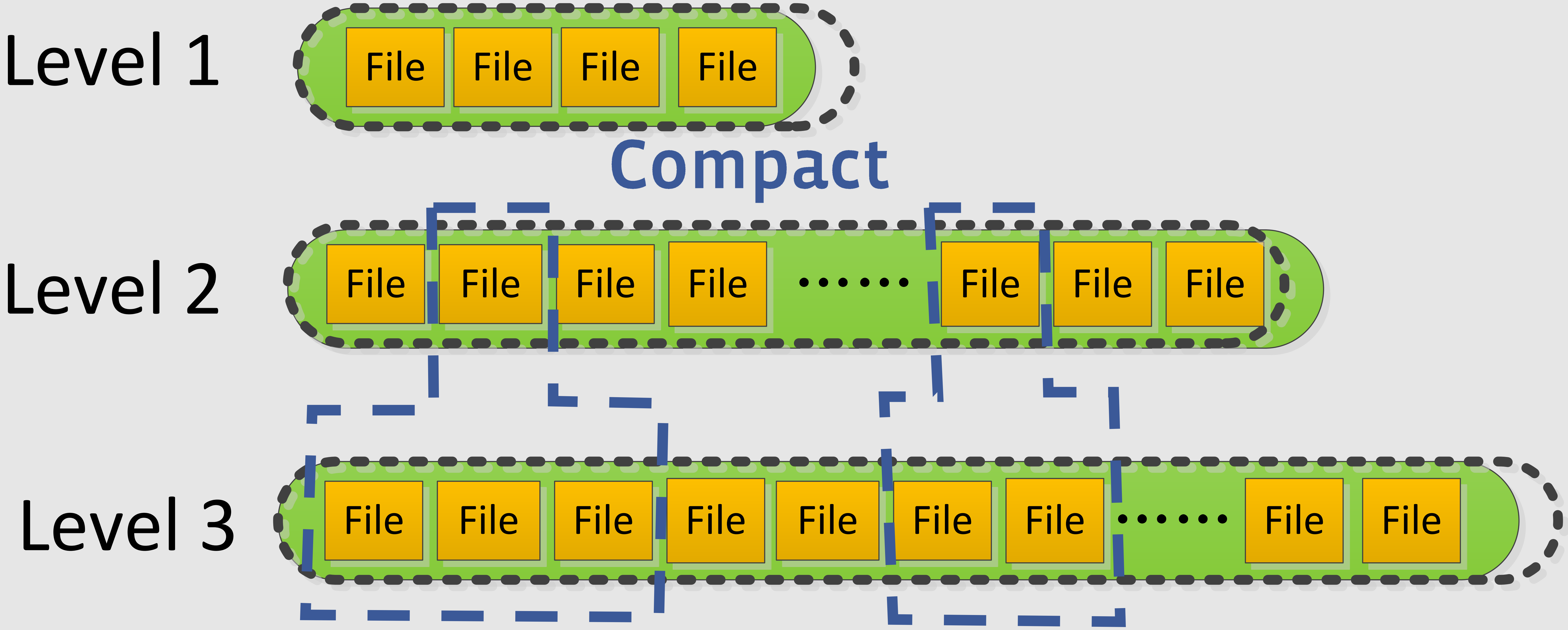
Level-Based Compaction



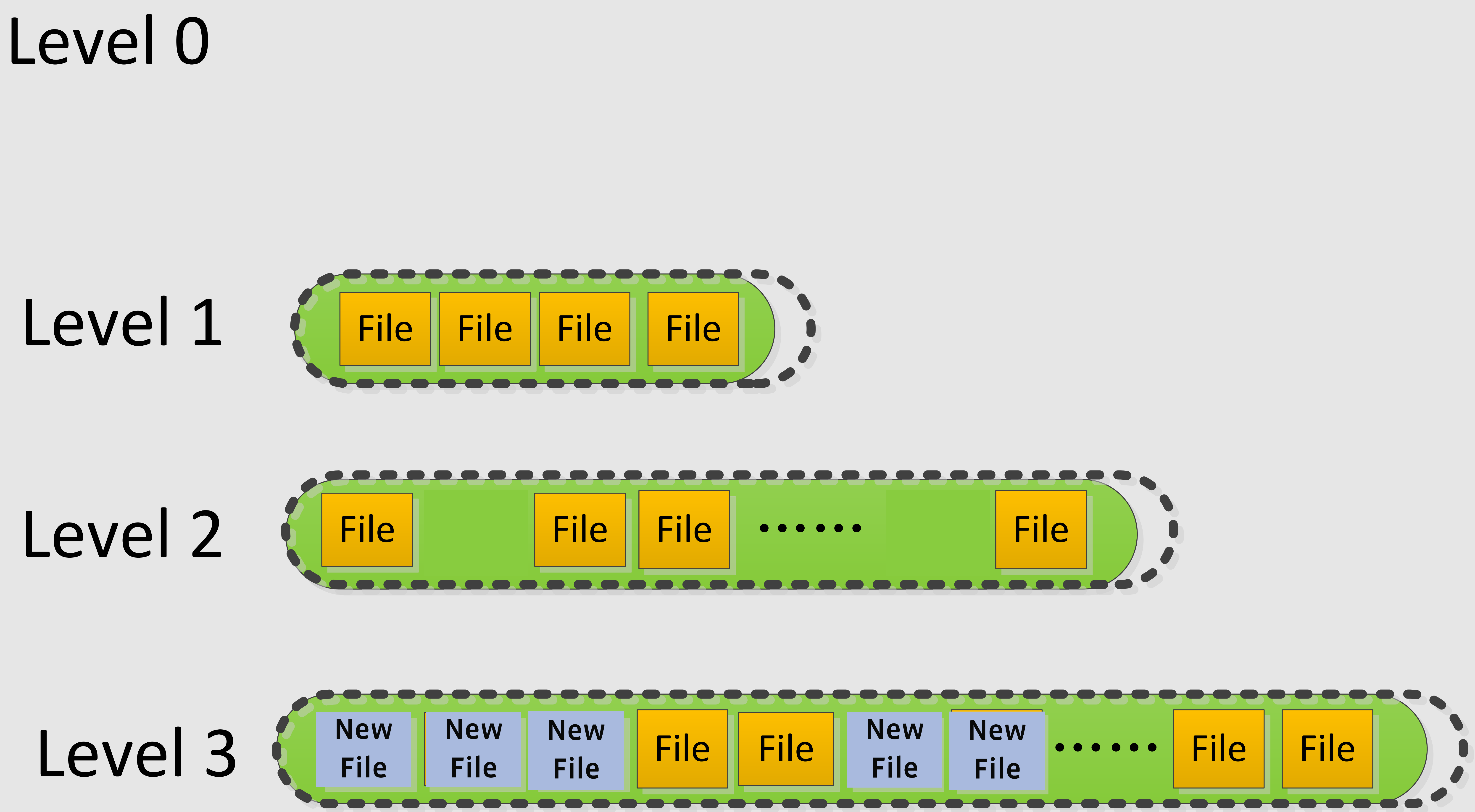
Level-Based Compaction



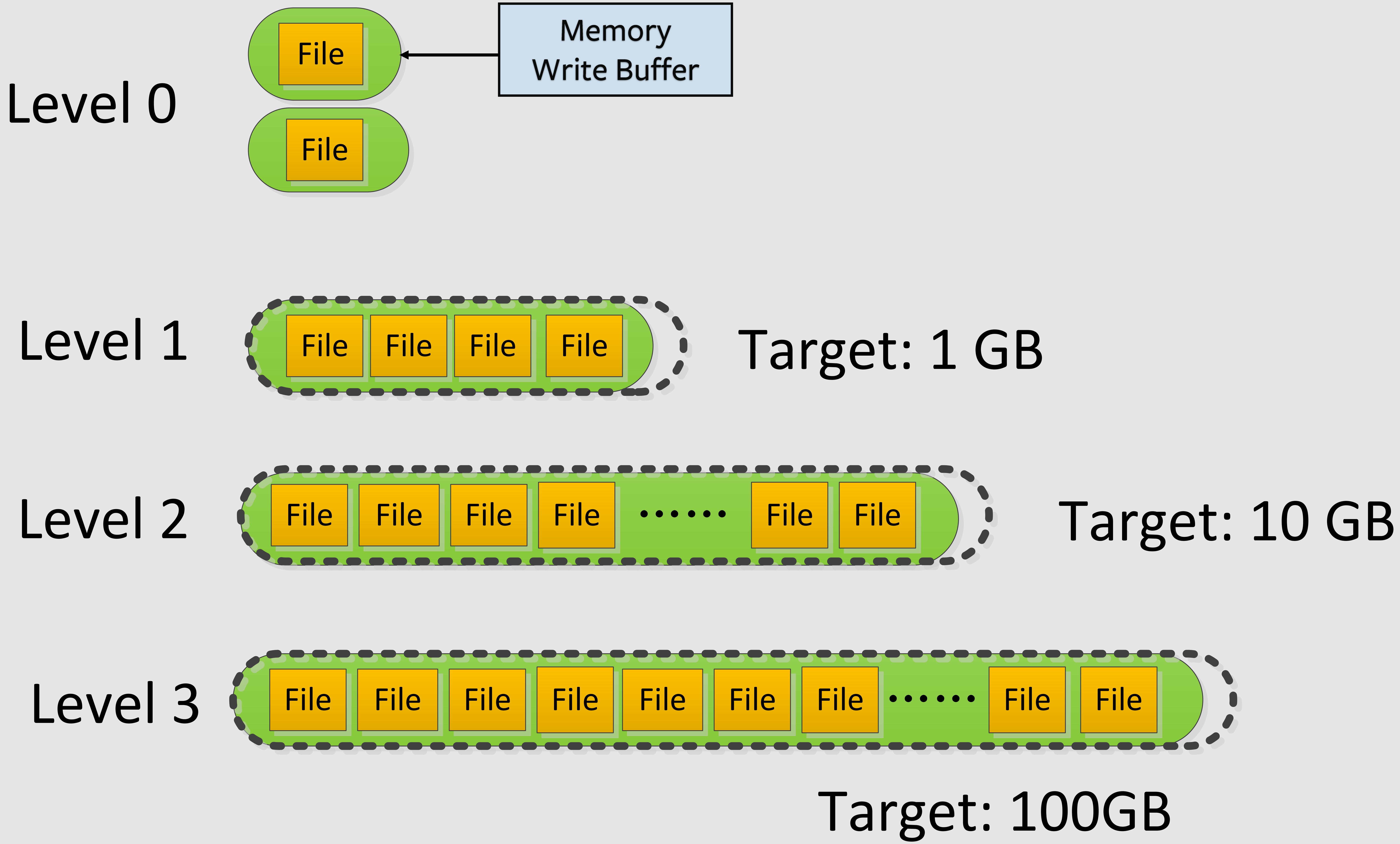
Level-Based Compaction



Level-Based Compaction



Example of Level Base Targets



Level 0

File

File

Memory Write Buffer

Level 1

File File File File

Target: 1 GB

Level 2

File File File File ······ File File

Target: 10 GB

Level 3

File File File File File File File ······ File File

Target: 100GB

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.*

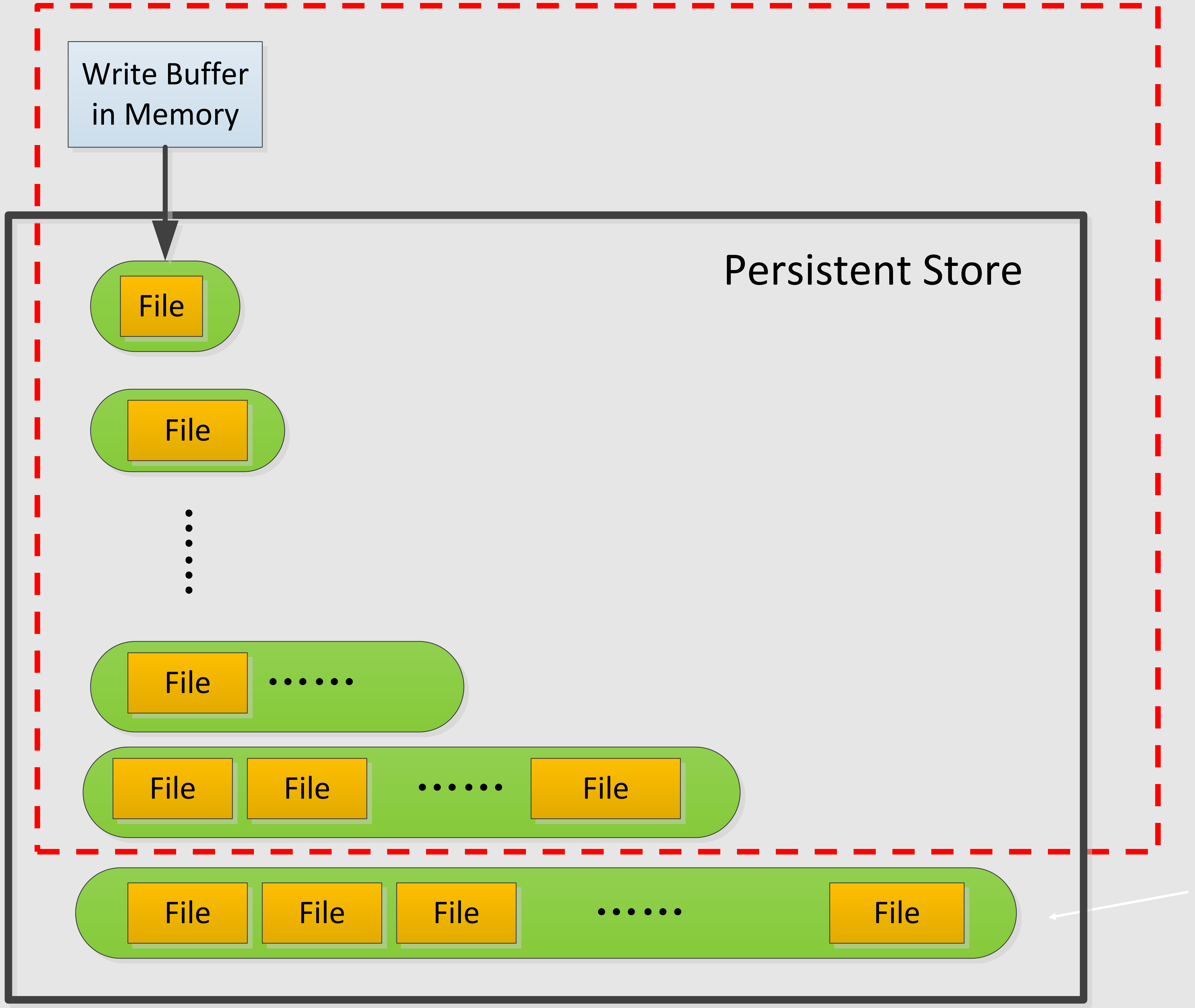
Everything except space has room to go!

Space Amplification of RocksDB

Only 10% Extra Space

How?

Space efficiency in LSM?

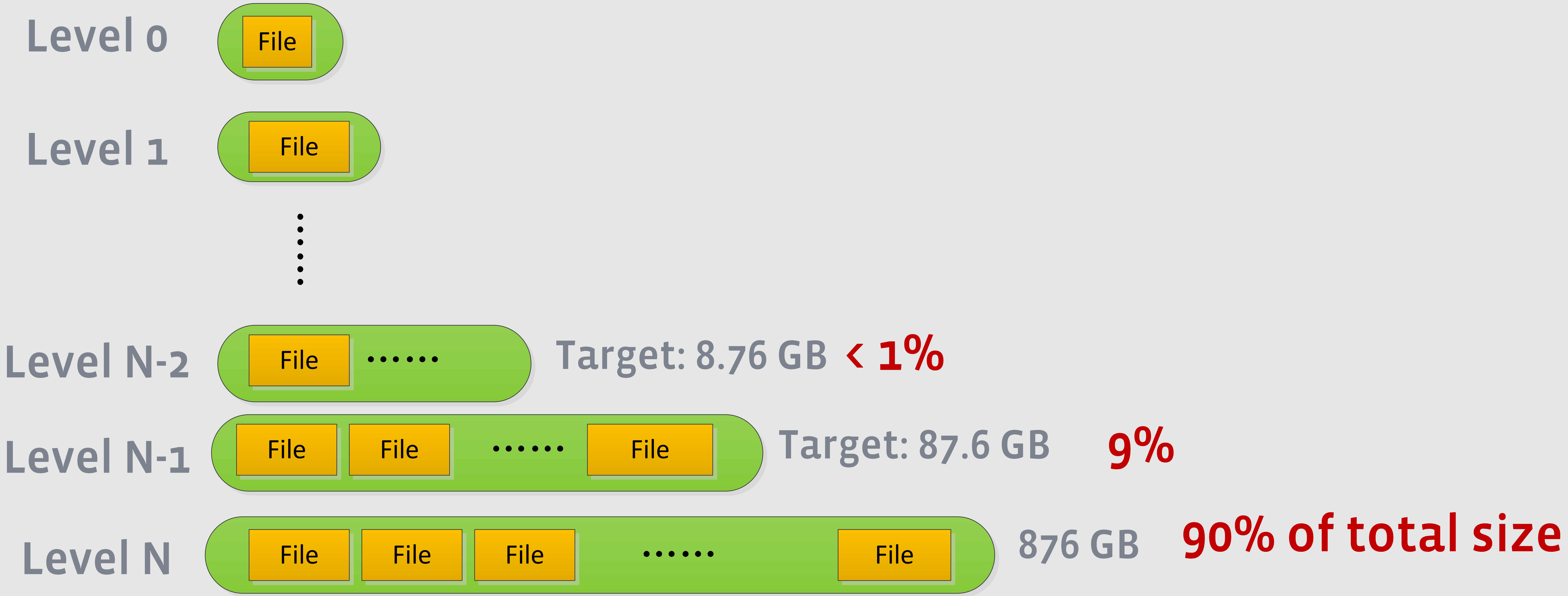


10% Extra Space

Size Similar to User Data Size

How Did We Guarantee 10%?

A Space-Efficient Approach



Level 0 File

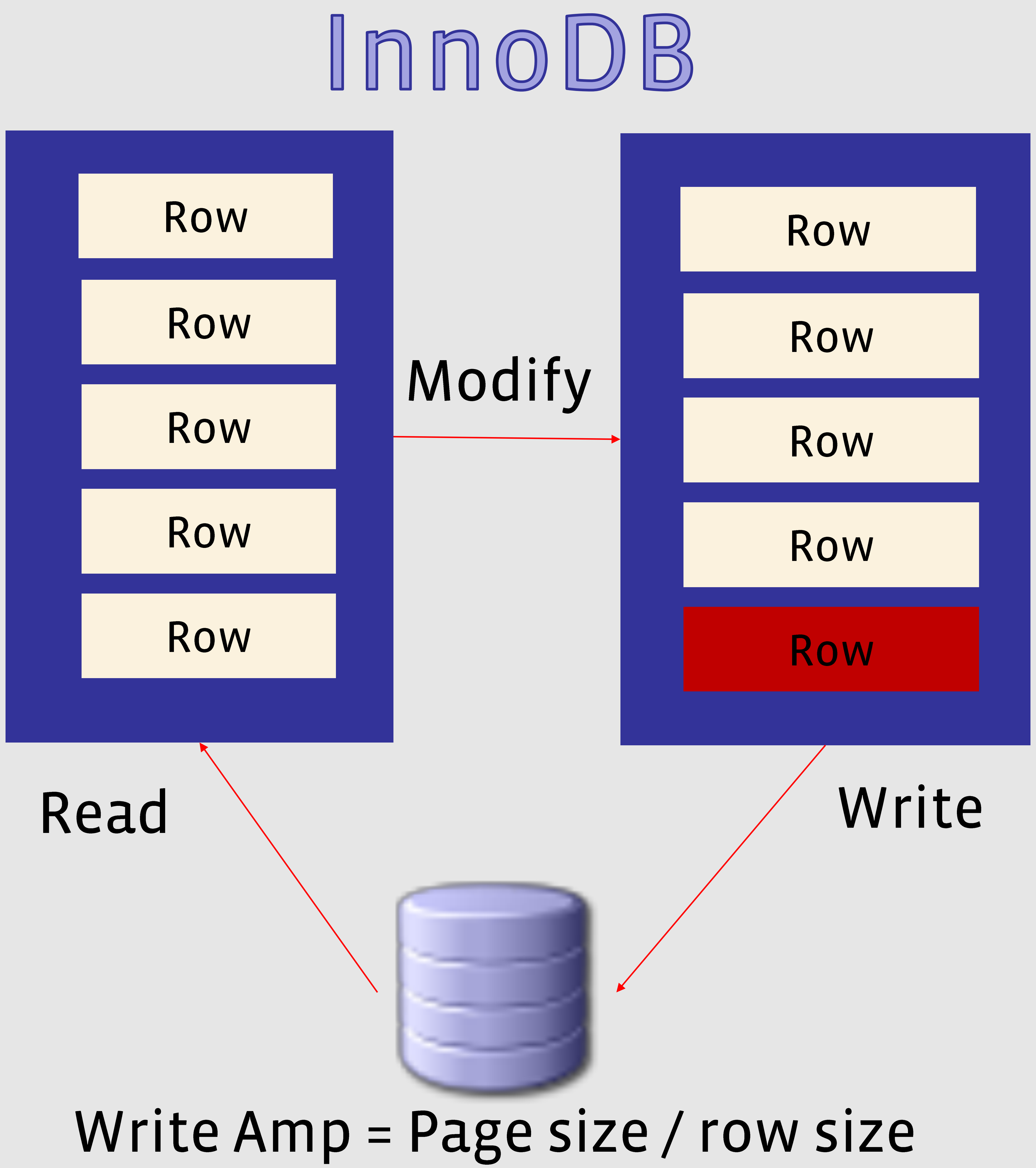
Level 1 File

Level N-2 File ······ Target: 8.76 GB < 1%

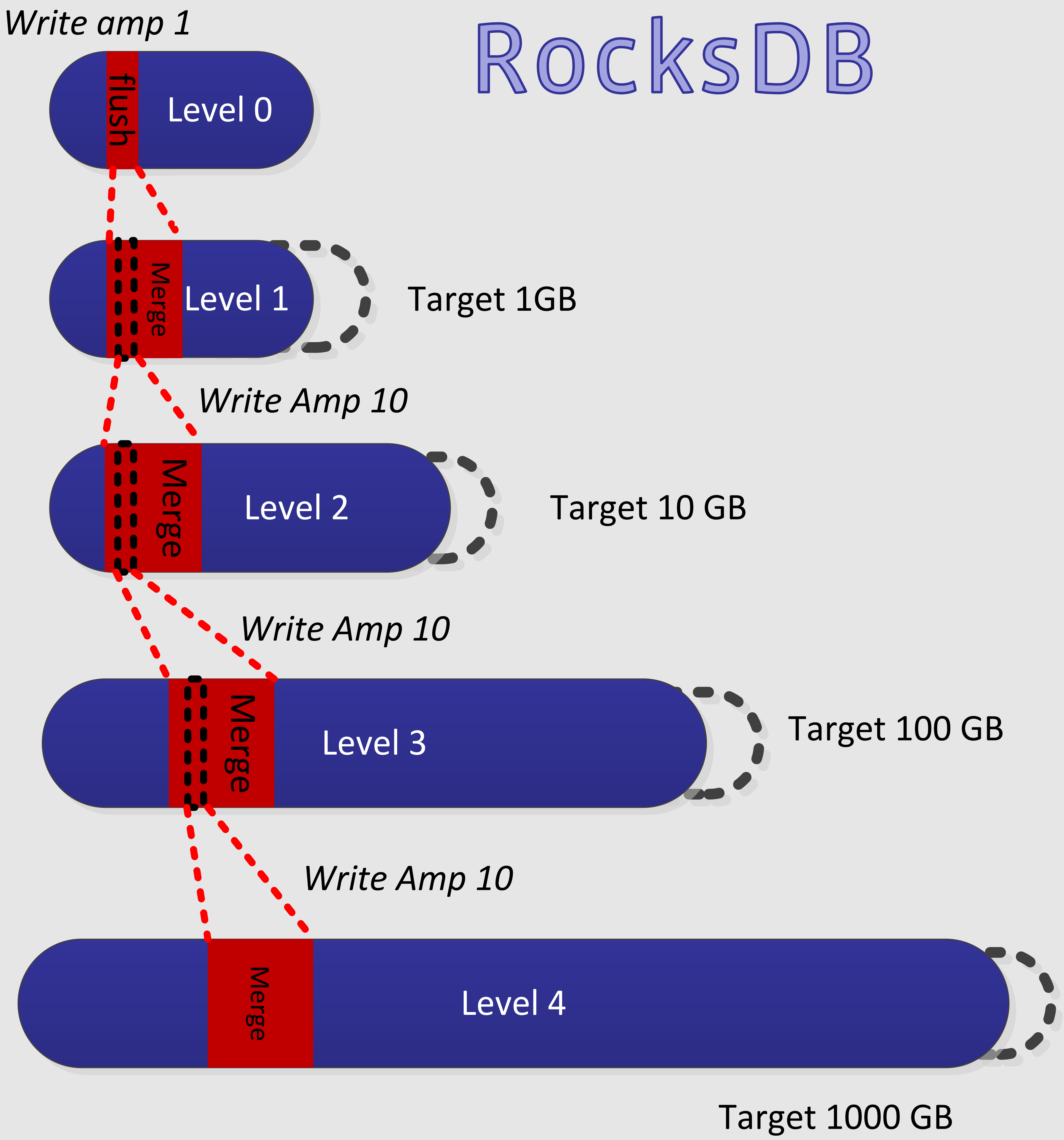
Level N-1 File File ······ File Target: 87.6 GB 9%

Level N File File File ······ File 876 GB 90% of total size

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

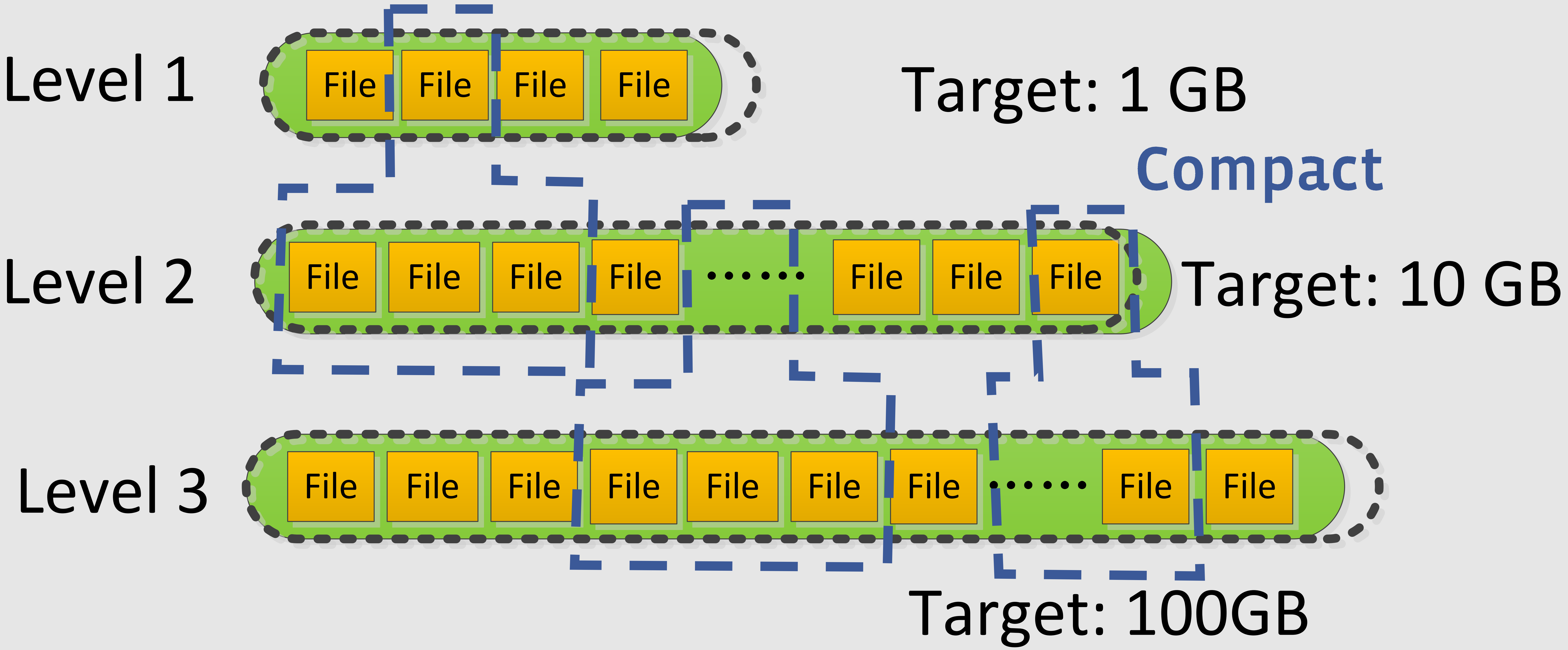
- Memtable: skip list
- Data Files: immutable
- LSM tree change: thread-local cache of the tree
- Synchronize opened files: allow to keep all files open
- Block cache mutex: sharded; more optimization coming.

Write Throughput

- Throughput of Compactions
- Throughput of Memtable Inserts

Multi-thread compactions

Compact non-overlapping files



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

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*

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

Thank You!

- Portal: http://rocksdb.org/
- Github: https://github.com/facebook/rocksdb
- Discussion Group: https://www.facebook.com/groups/rocksdb.dev/
- Mailing List: https://groups.google.com/forum/#!forum/rocksdb