SILK: Preventing Latency Spikes in Log-Structured Merge Key-Value Stores

O. Balmau*, F. Dinu*, W. Zwaenepoel*,

K. Gupta†, R. Chandhiramoorthi†, D. Didona§

* THE UNIVERSITY OF SYDNEY

† NUTANIX™

§ IBM Research | Zurich

Talk MSR Seattle, July 2019

About me

PhD Student at the University of Sydney.

Advised by Willy Zwaenepoel.

Bachelor and Master in CS from EPFL.

Research interests:

storage systems, distributed systems, concurrent algorithms, parallelism.

Internships in Nutanix CA/Bangalore, HP Vertica, ABB Research.

Log-Structured Merge (LSM) KVs

- Designed for write-heavy workloads
- Handle large-scale data
- Working set does not fit in RAM

cassandra

LEVELDB

RocksDB

Log-Structured Merge (LSM) KVs

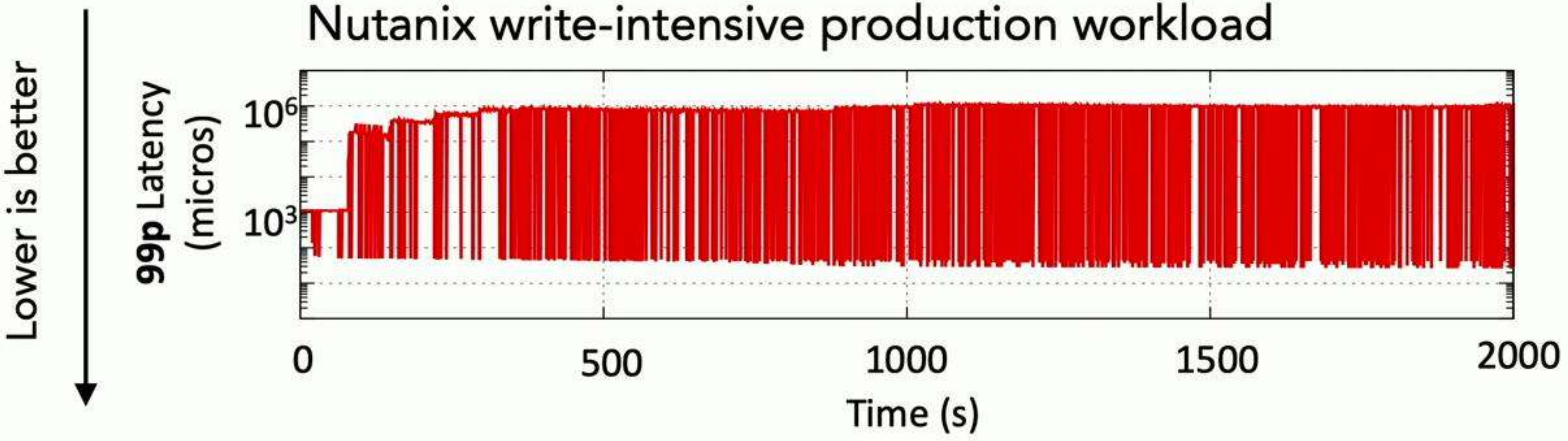
- Designed for write-heavy workloads?
- Handle large-scale data
- Working set does not fit in RAM

cassandra

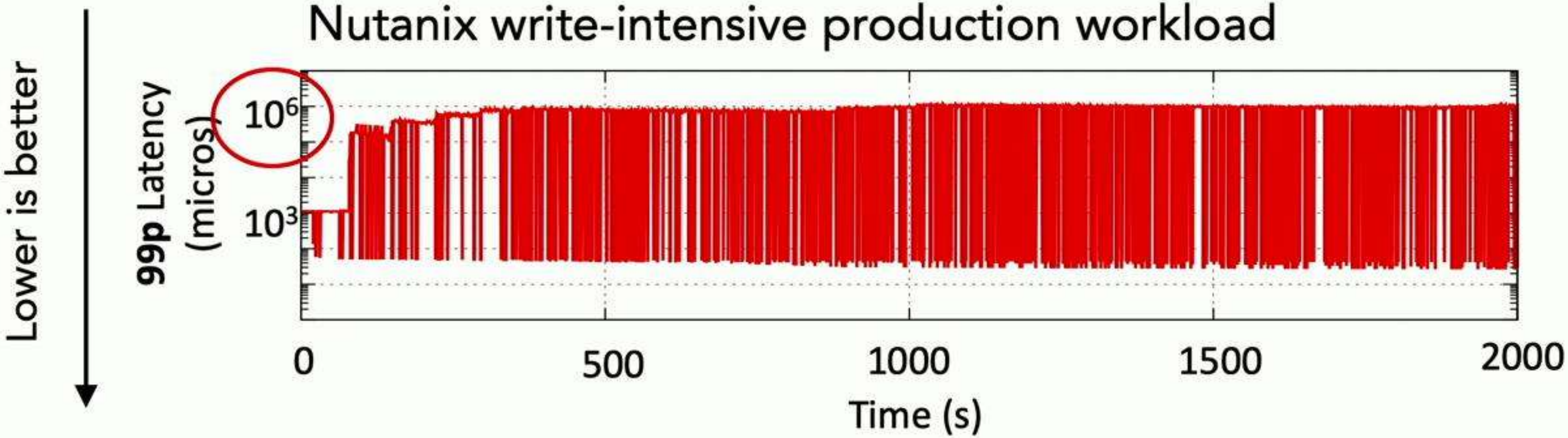
LEVELDB

RocksDB

LSM KV Latency Spikes in RocksDB



LSM KV Latency Spikes in RocksDB



Latency spikes of up to 1s in write dominated workloads!

Spikes are up to 3 orders of magnitude > median tail latency

Latency Spikes in LSM KVs

Why is this important?

- Cannot provide SLA guarantees to clients.
- Unpredictable performance when connecting LSM in larger pipelines.

Our Contribution: The SILK LSM KV

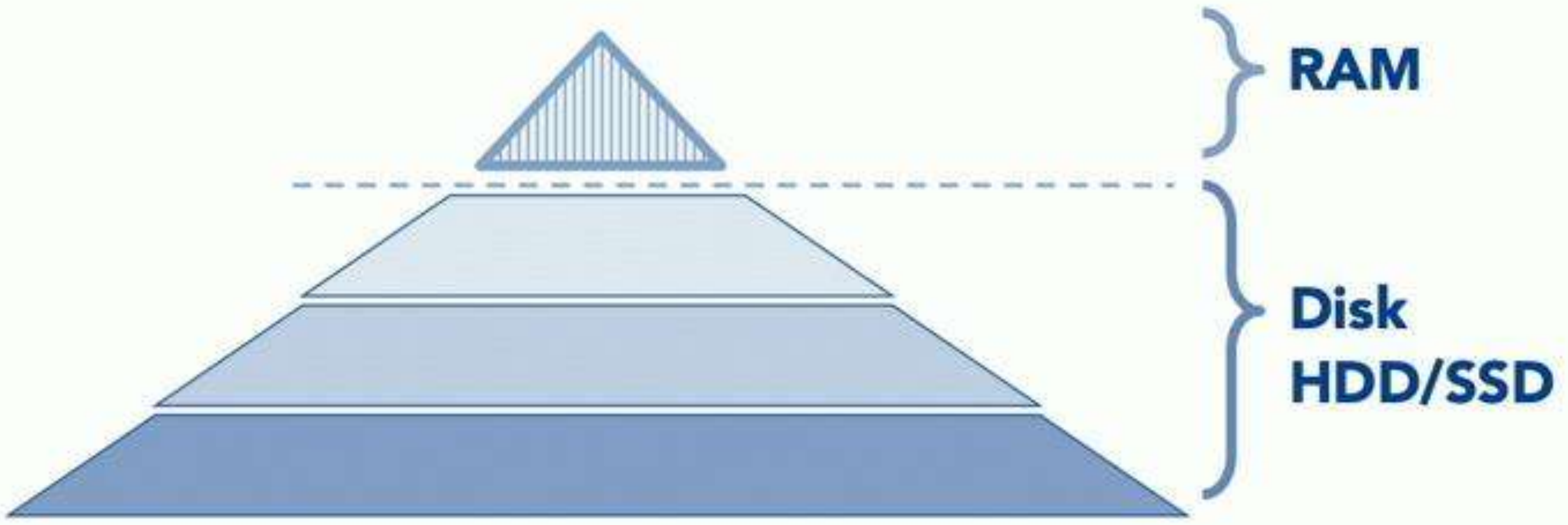
- Solves latency spike problem for write-heavy workloads.
- No negative side-effects for other workloads.
- SILK introduces the notion of an I/O scheduler for LSM KVs.

Experimental Study: Reason Behind Latency Spikes

What Causes LSM Latency Spikes?

Severe competition for I/O bandwidth between client operations and LSM internal operations (~GC).

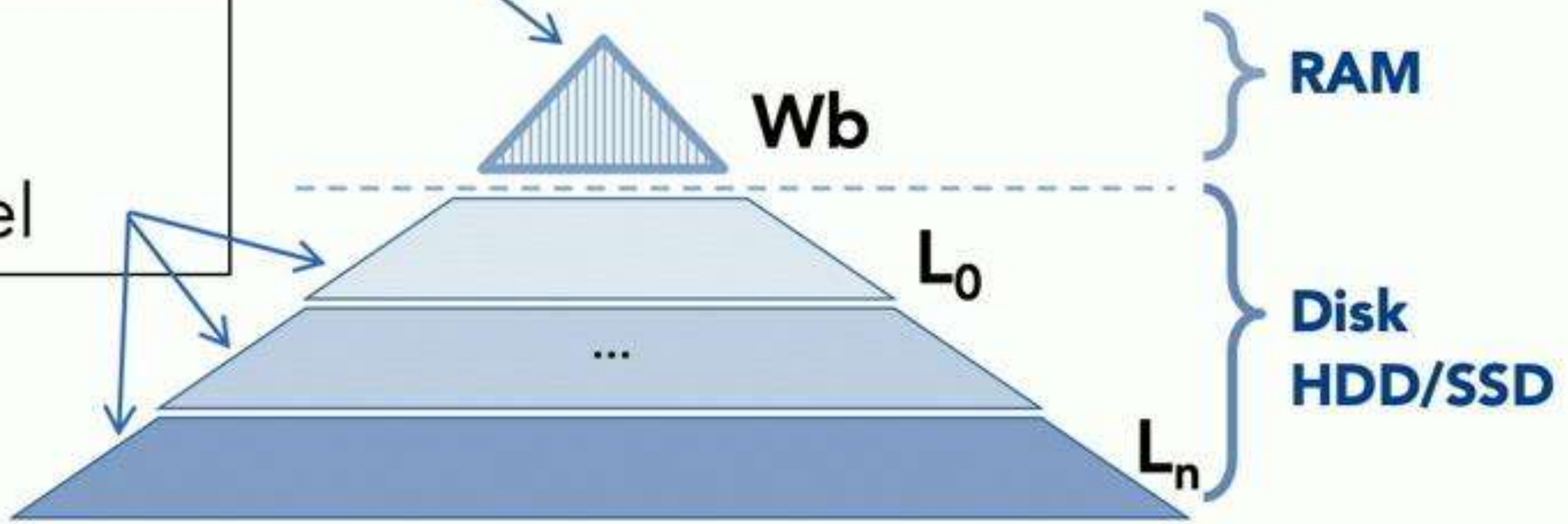
LSM KV Overview



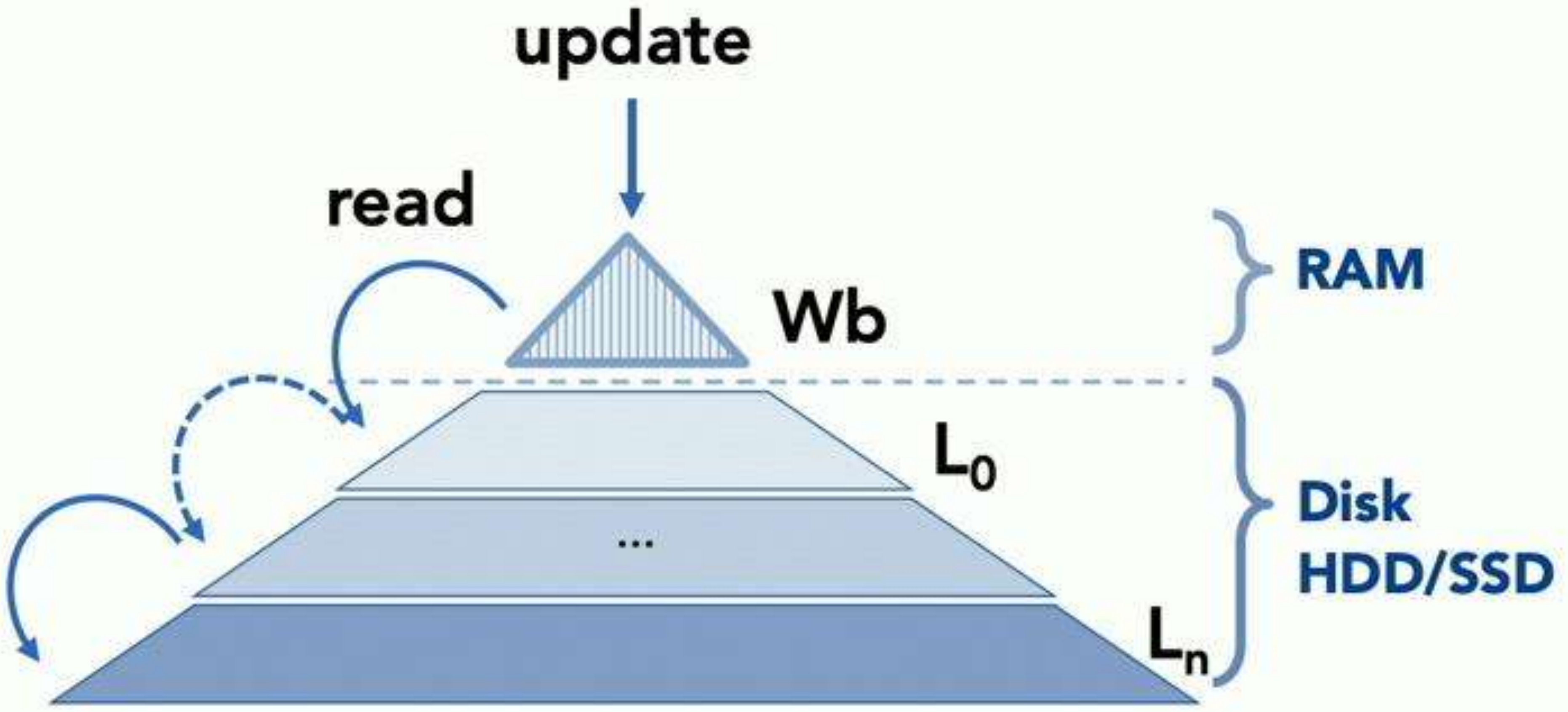
LSM KV Overview

Write buffer

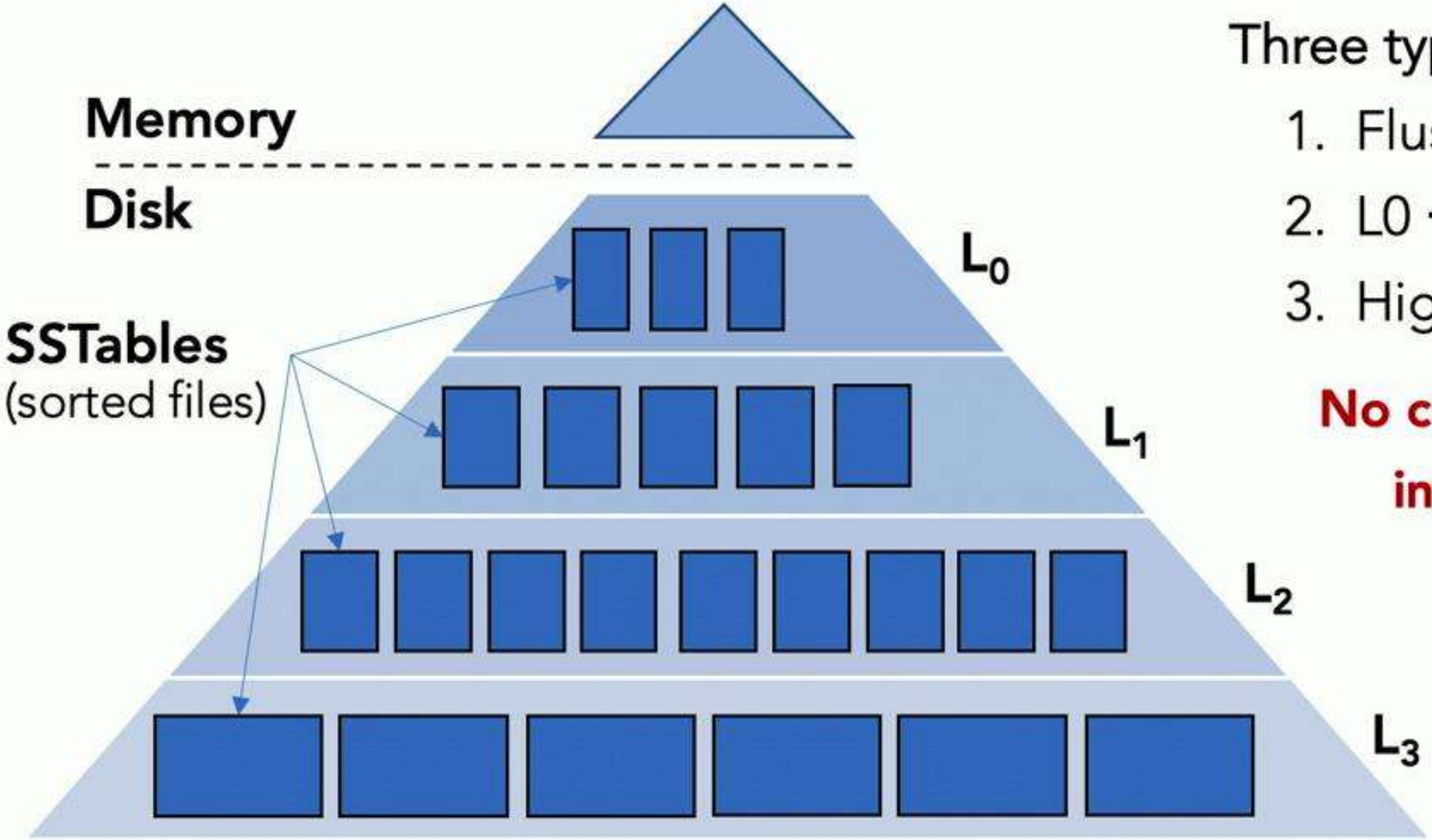
SSTables
- sorted files
- many SSTables/Level



LSM KV Client Operations



LSM Internal Ops

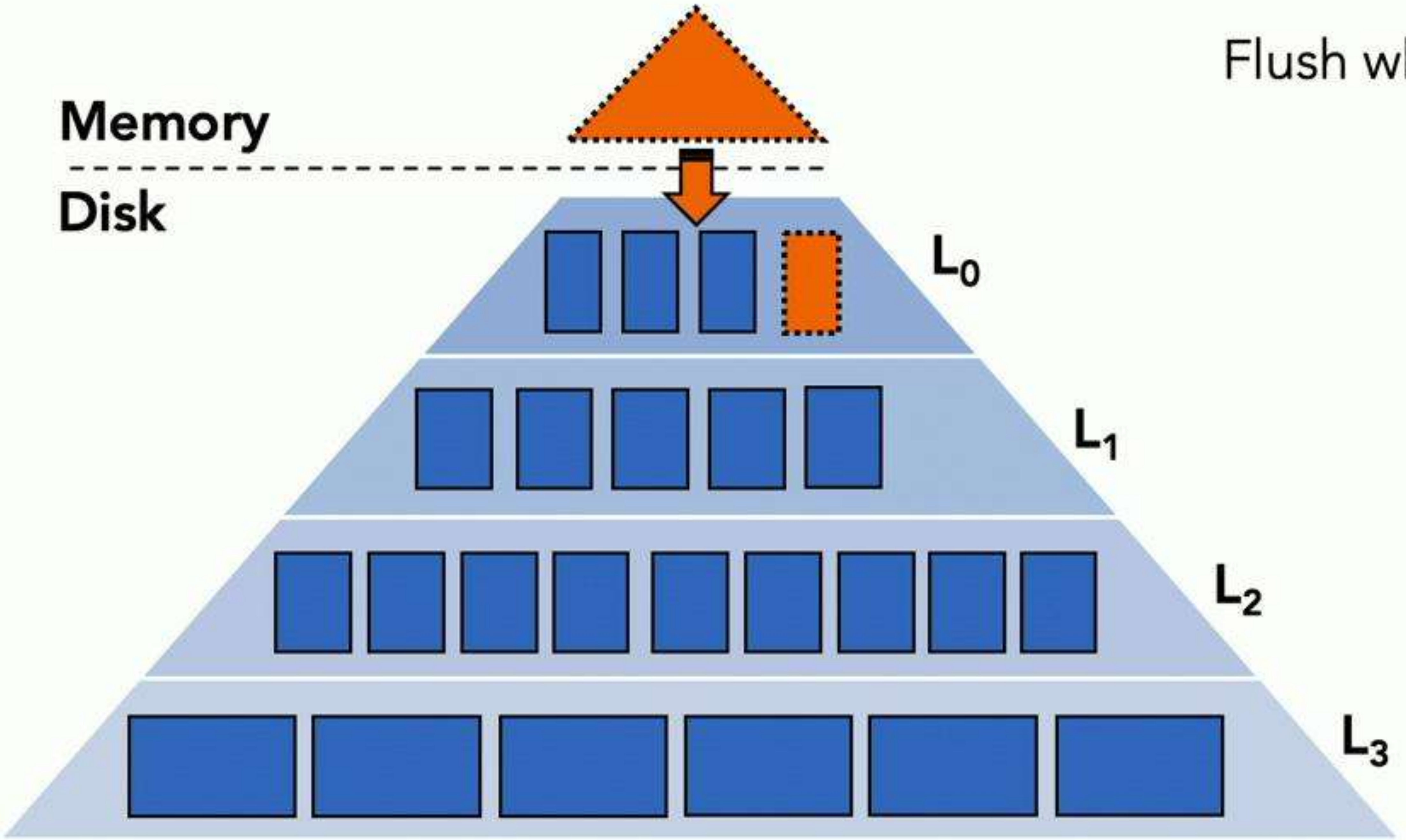


Three types of internal ops:

1. Flushing
2. L0 → L1 compaction
3. Higher level compactions

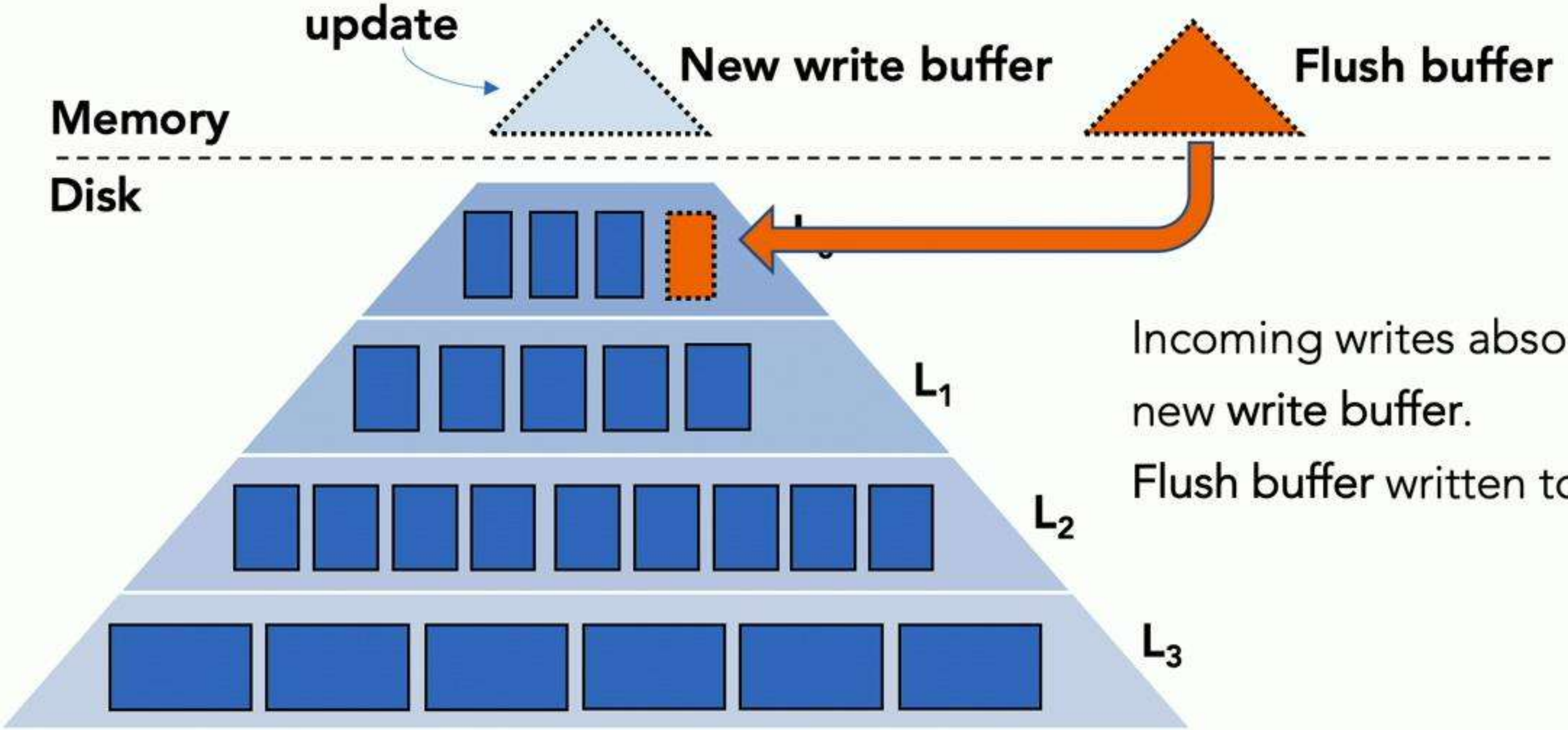
No coordination between internal operations.

LSM Internal Ops: Flushing



Flush when **Write buffer full.**

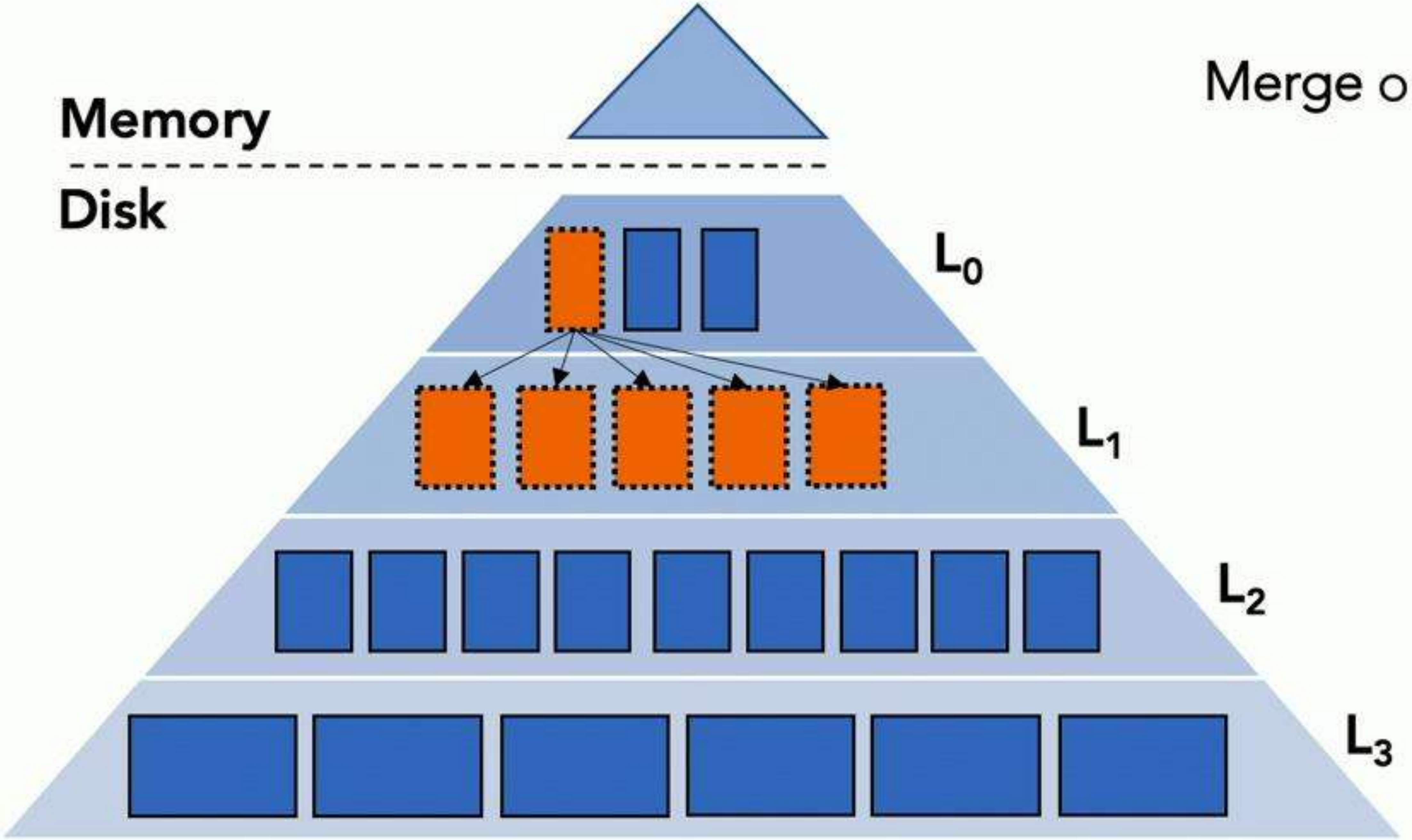
LSM Internal Ops: Flushing



Incoming writes absorbed in new **write buffer**.

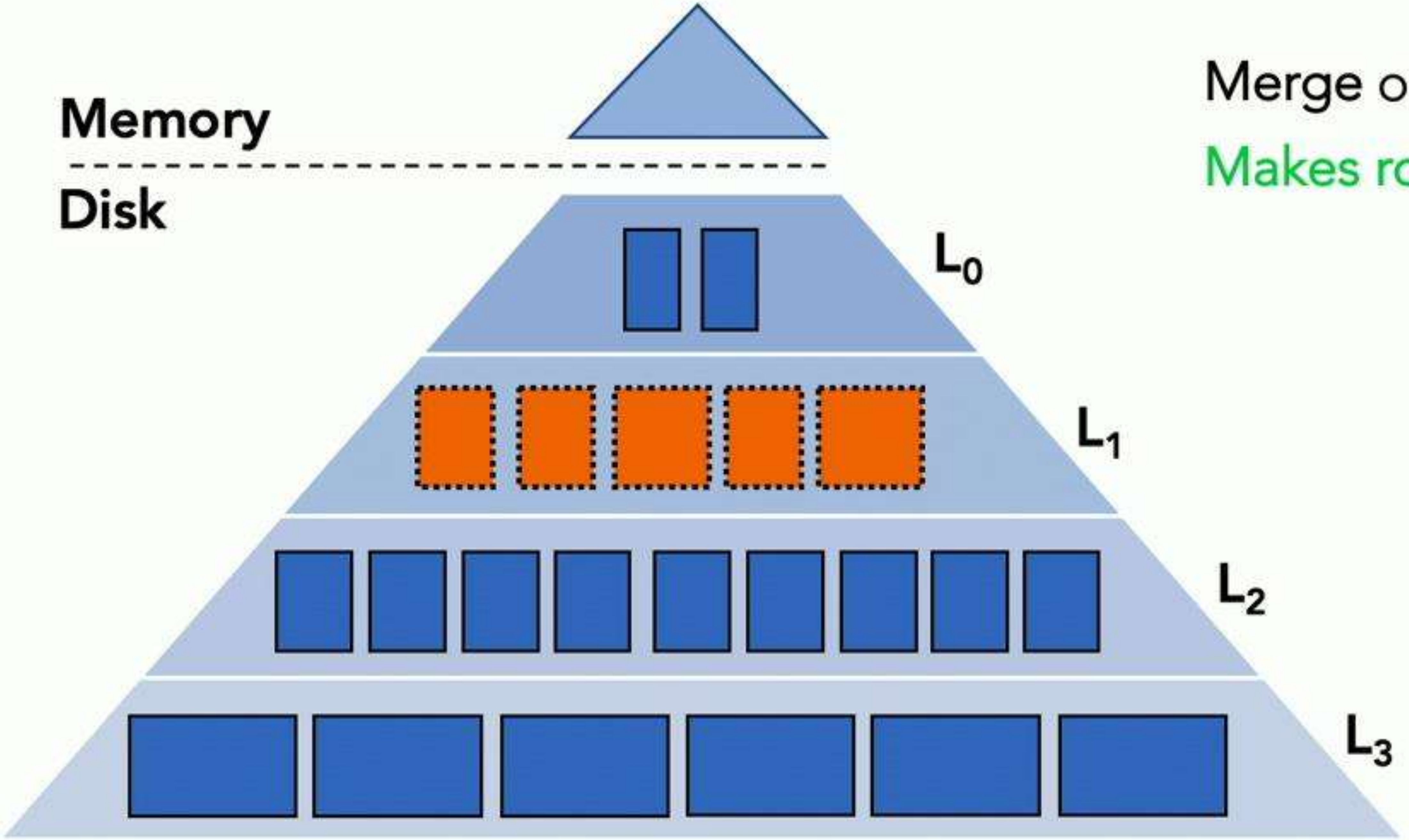
Flush buffer written to L0.

LSM Internal Ops: L0 → L1 compactions



Merge one L0 SSTable with L1.

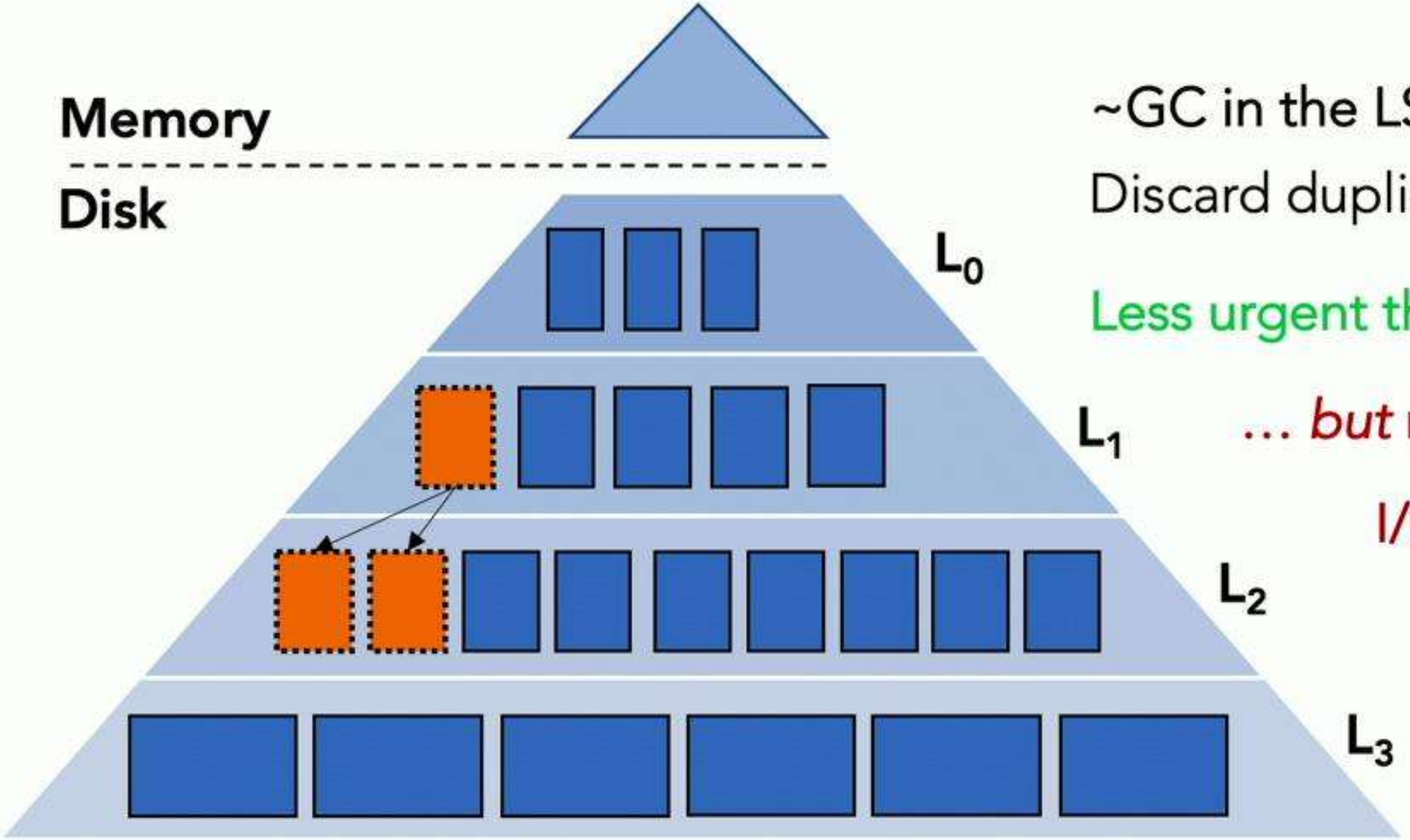
LSM Internal Ops: L0 → L1 compactions



Merge one L0 SSTable with L1.

Makes room on L0 for flushing.

LSM Internal Ops: Higher Level Compactions



~GC in the LSM tree.

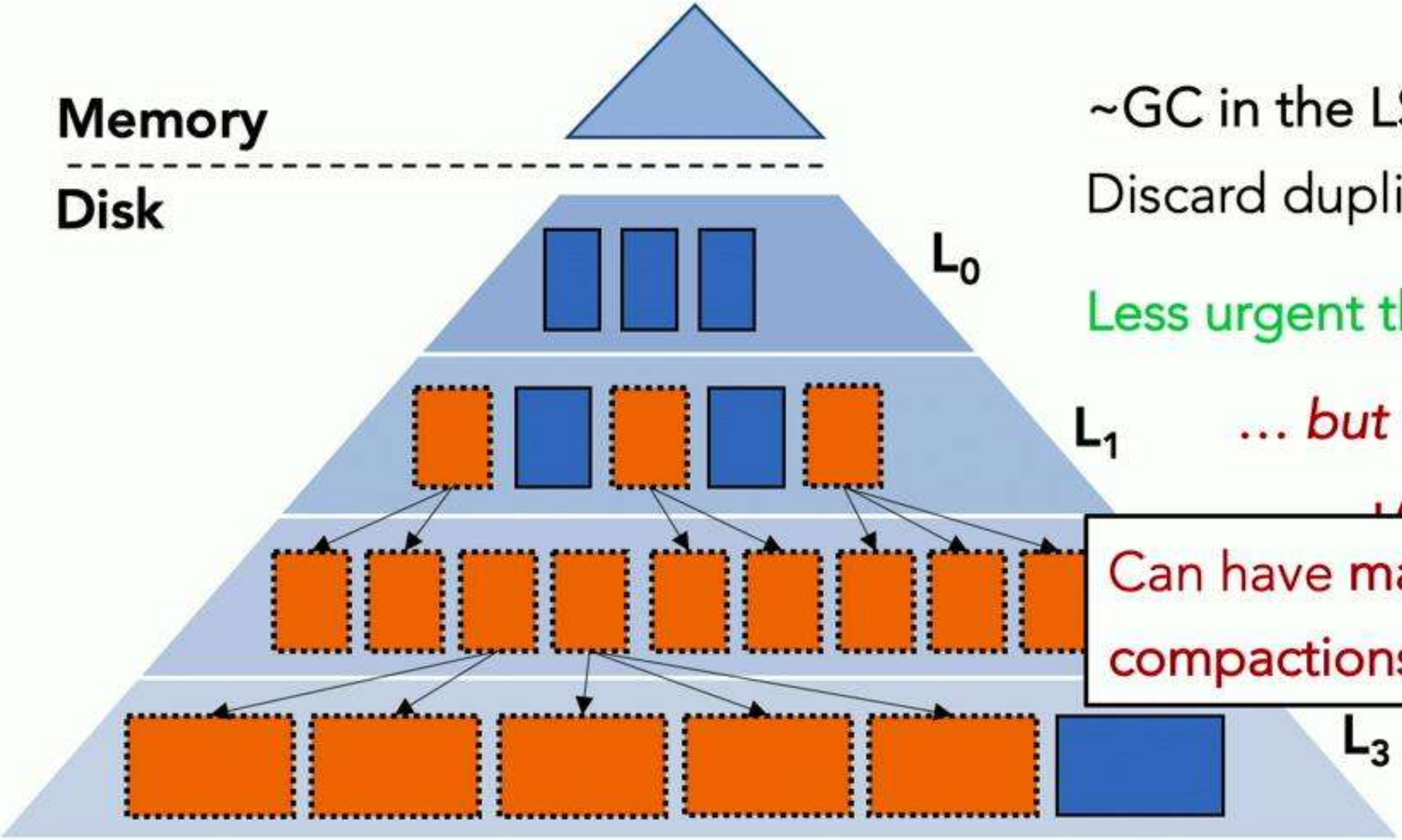
Discard duplicates & delete values.

Less urgent than L0→L1 compactions.

… *but* need to complete.

I/O bandwidth intensive.

LSM Internal Ops: Higher Level Compactions



~GC in the LSM tree.

Discard duplicates & delete values.

Less urgent than L0->L1 compactions.

… *but* need to complete.

Can have **many** higher level **compactions** running **in parallel**.

LSM Review

Internal operations:

1. Flushing. From memory to disk.
2. L0 → L1 compaction. Make room to flush new files.
3. Higher level compactions. ~GC, I/O intensive.

No coordination between internal ops and client ops.

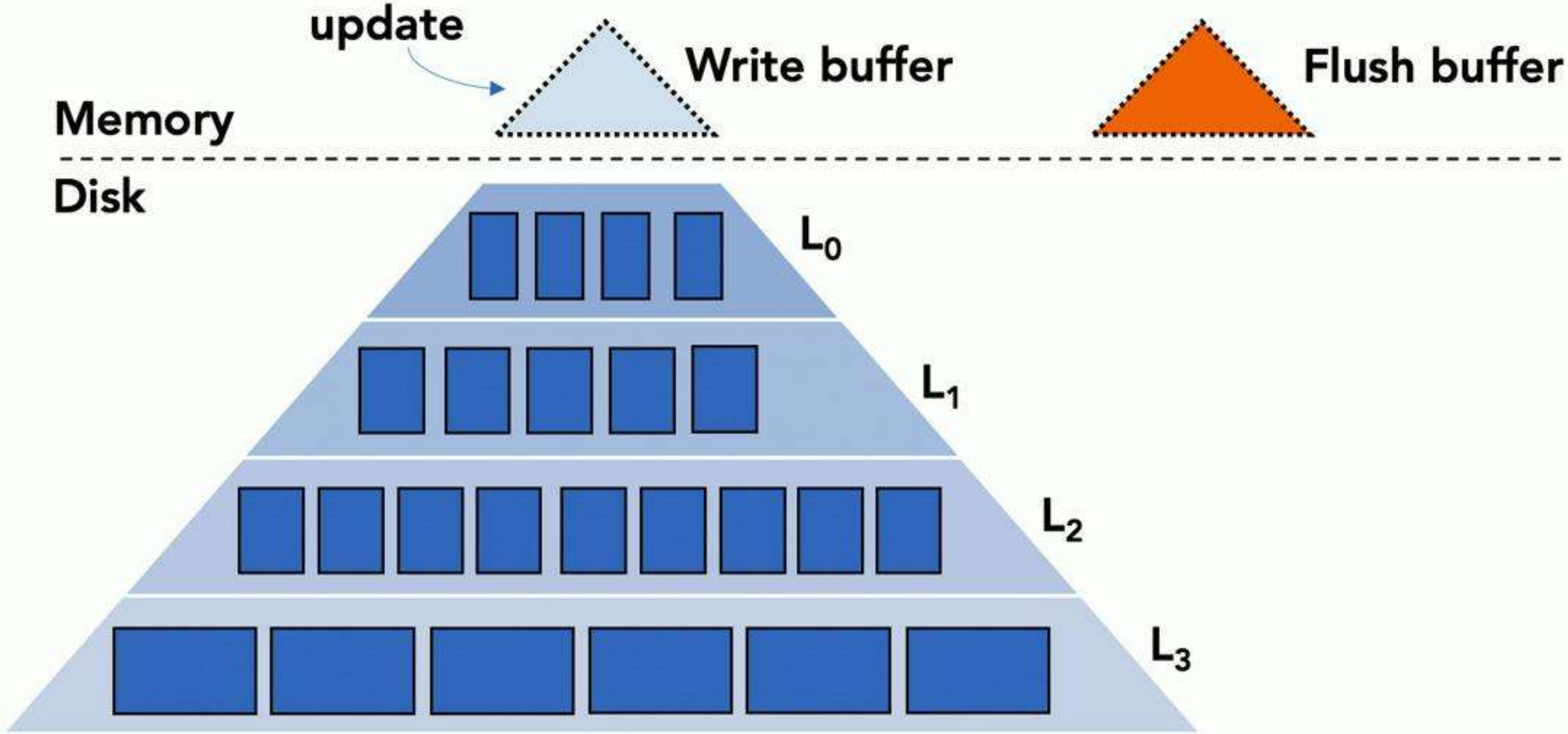
What Causes LSM Latency Spikes?

Both reads and writes experience latency spikes.

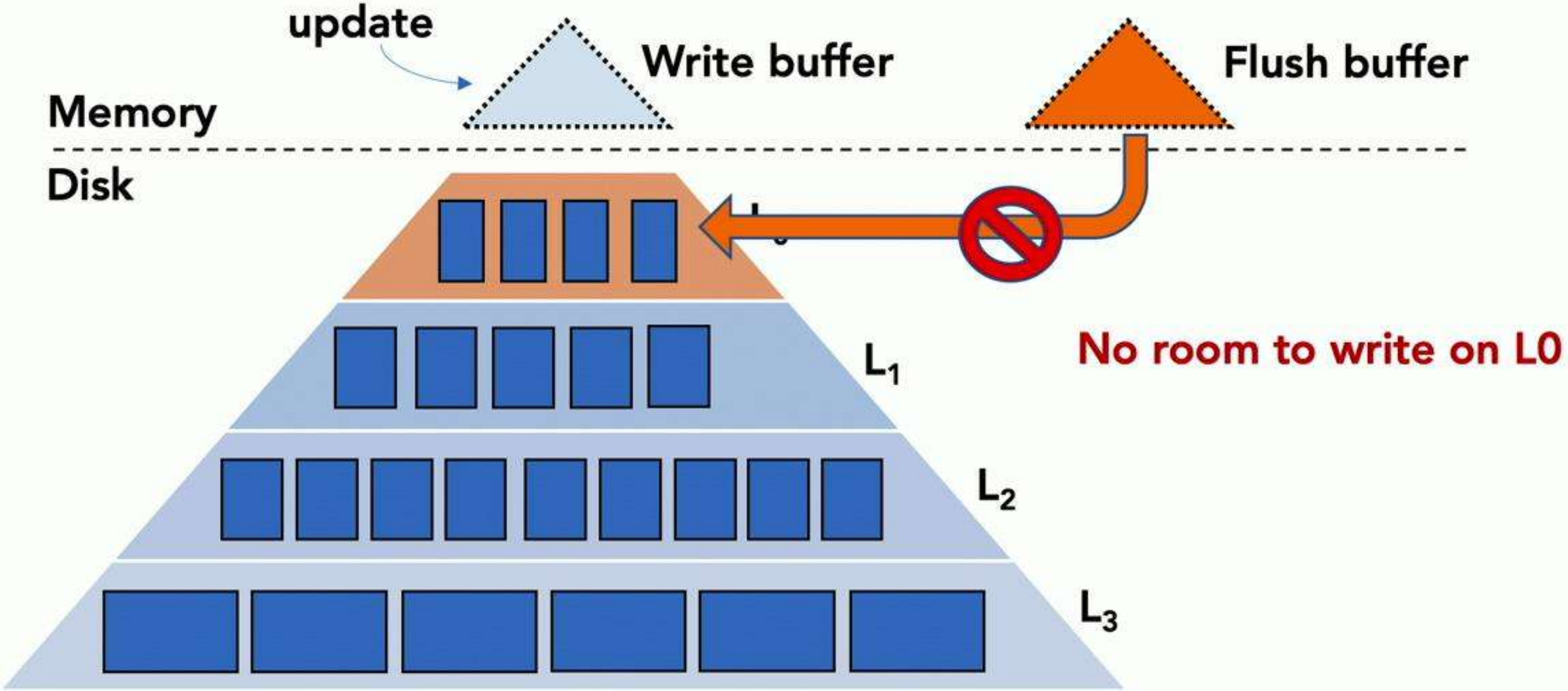
Focus on writes. Less intuitive.

Writes finish in memory. **Why do we have 1s latencies?**

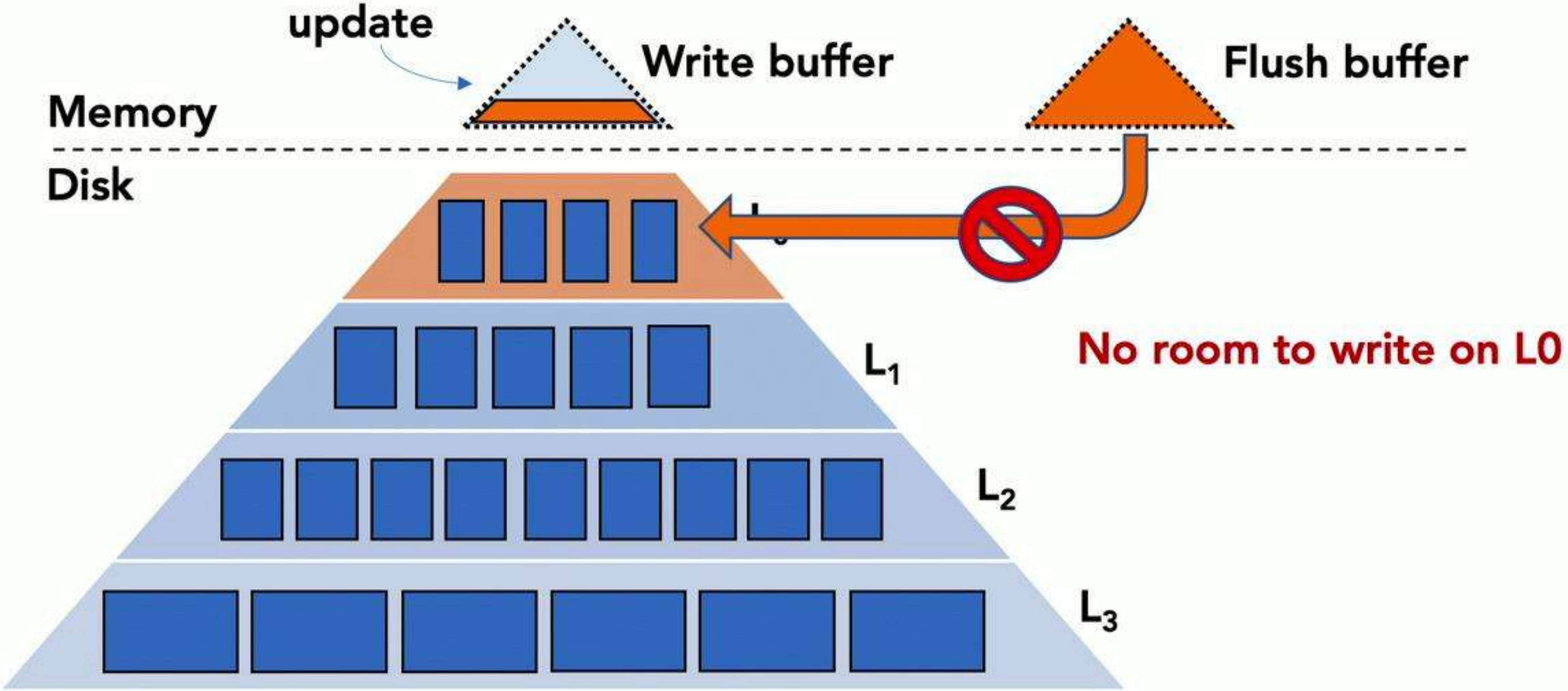
Cannot Flush



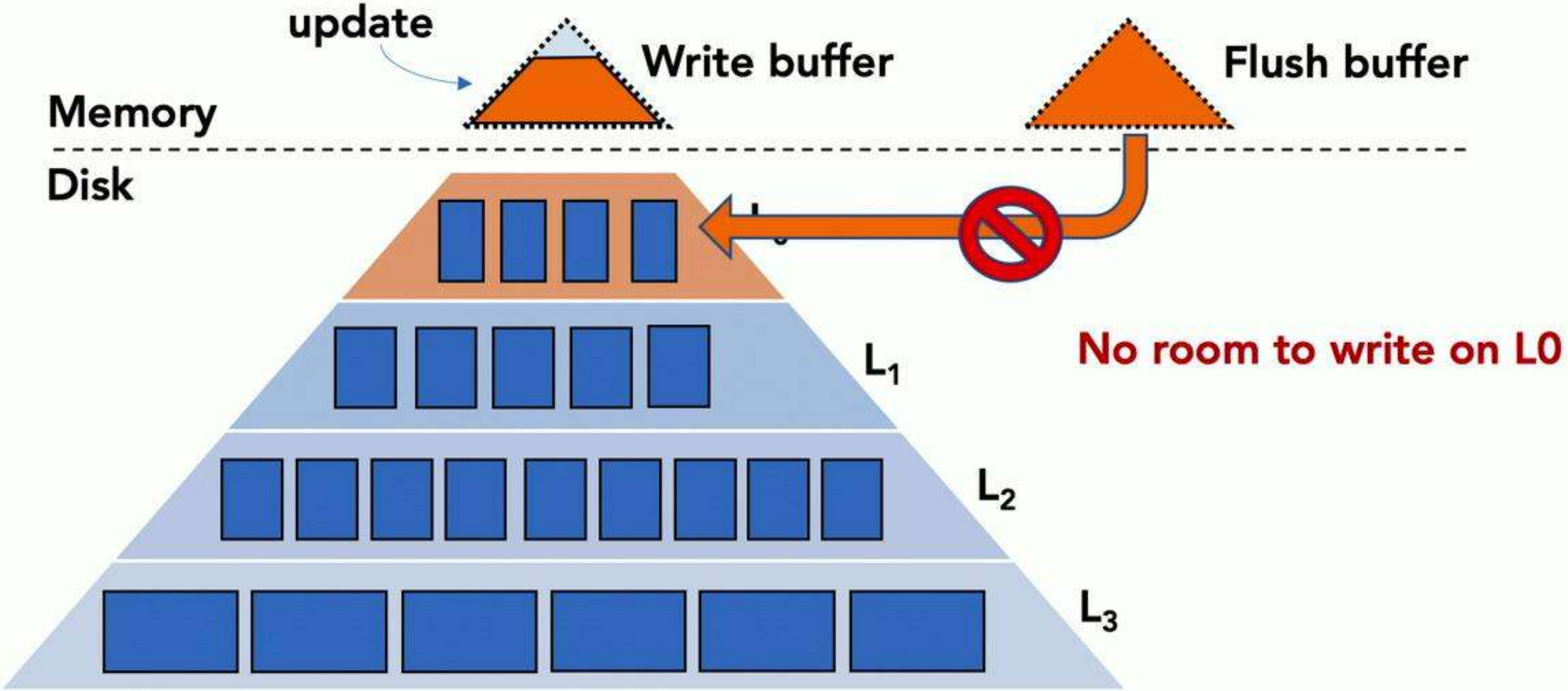
Cannot Flush



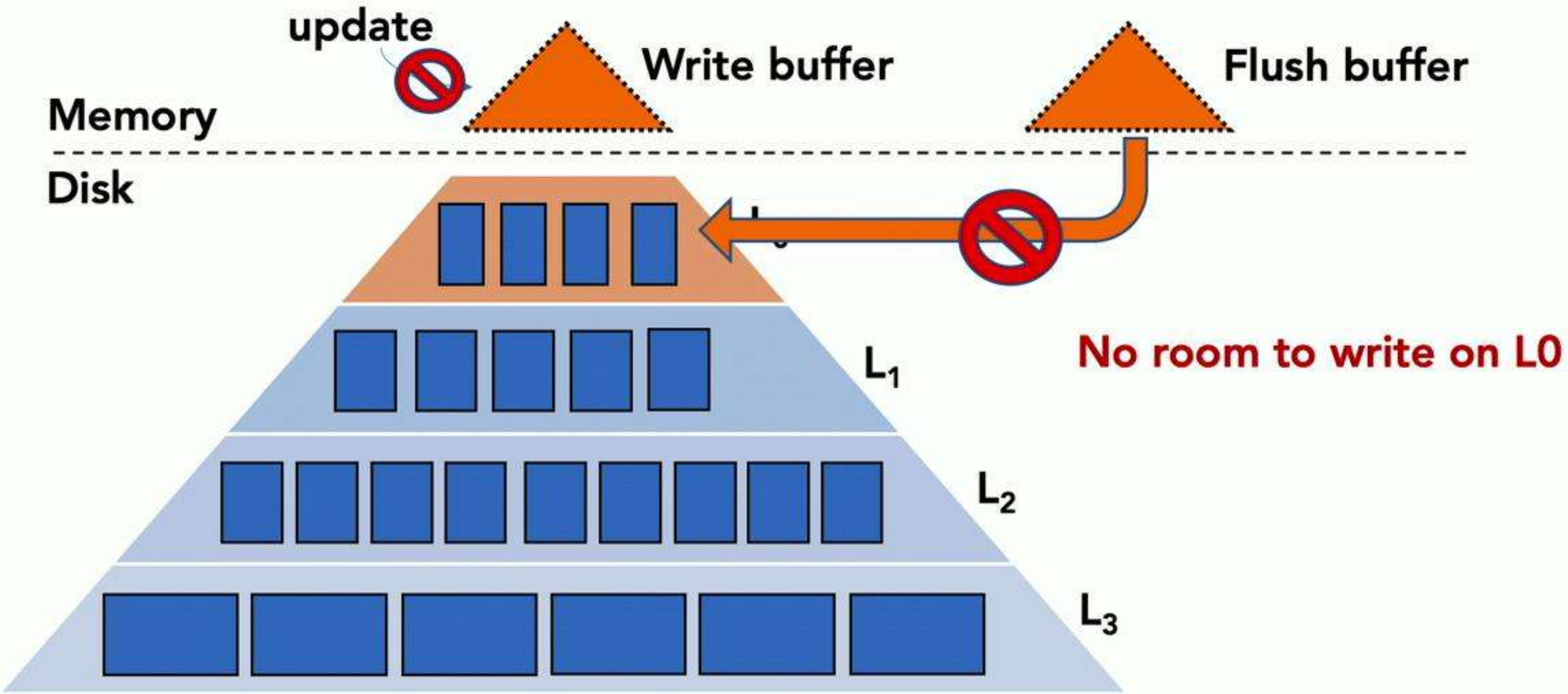
Cannot Flush



Cannot Flush



Cannot Flush



1. Writes Blocked Because L0 is Full.

No coordination between internal ops.

↓

Higher level compactions take over I/O.

↓

L0 → L1 compaction is too slow.

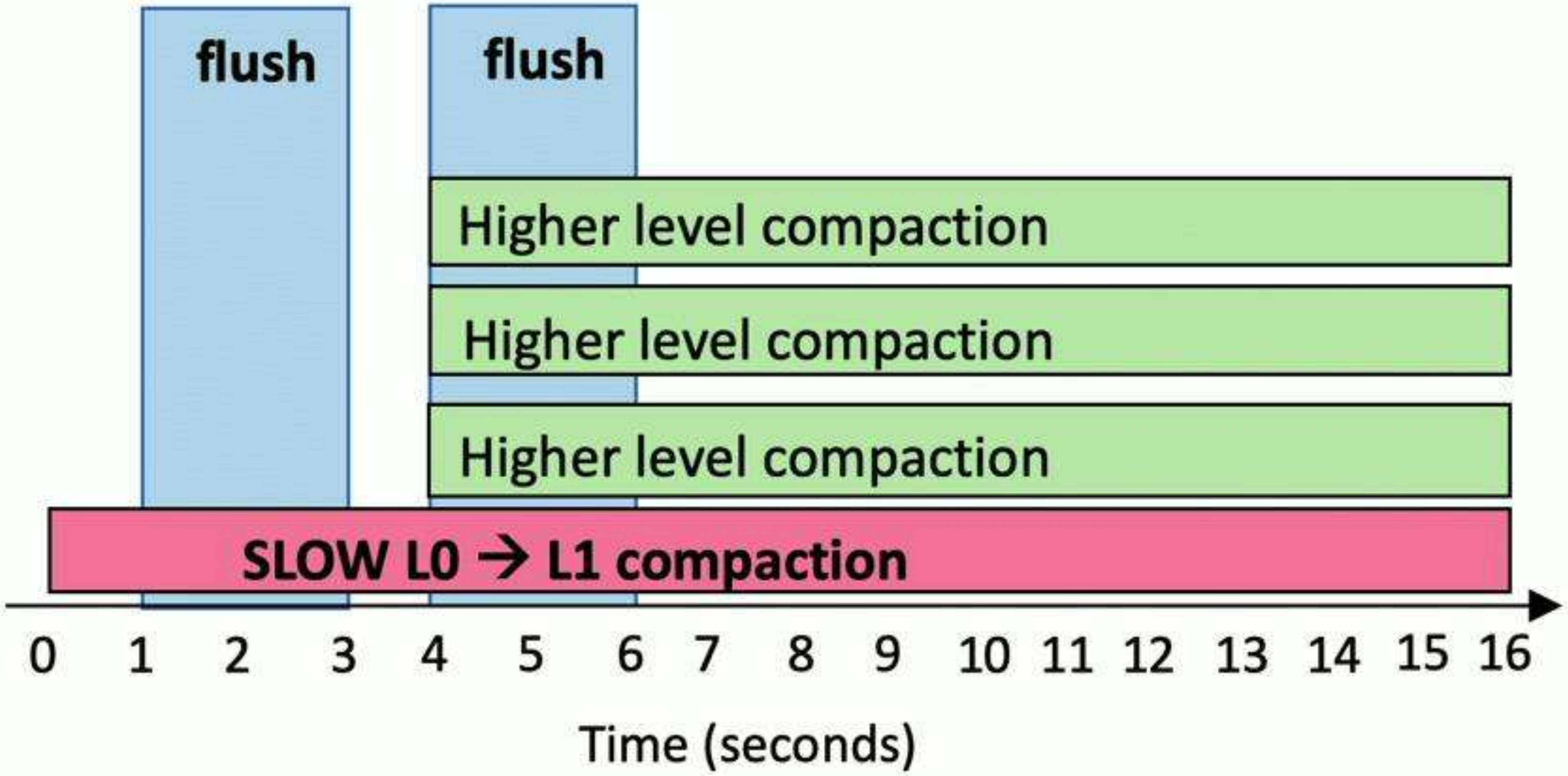
↓

Not enough space on L0.

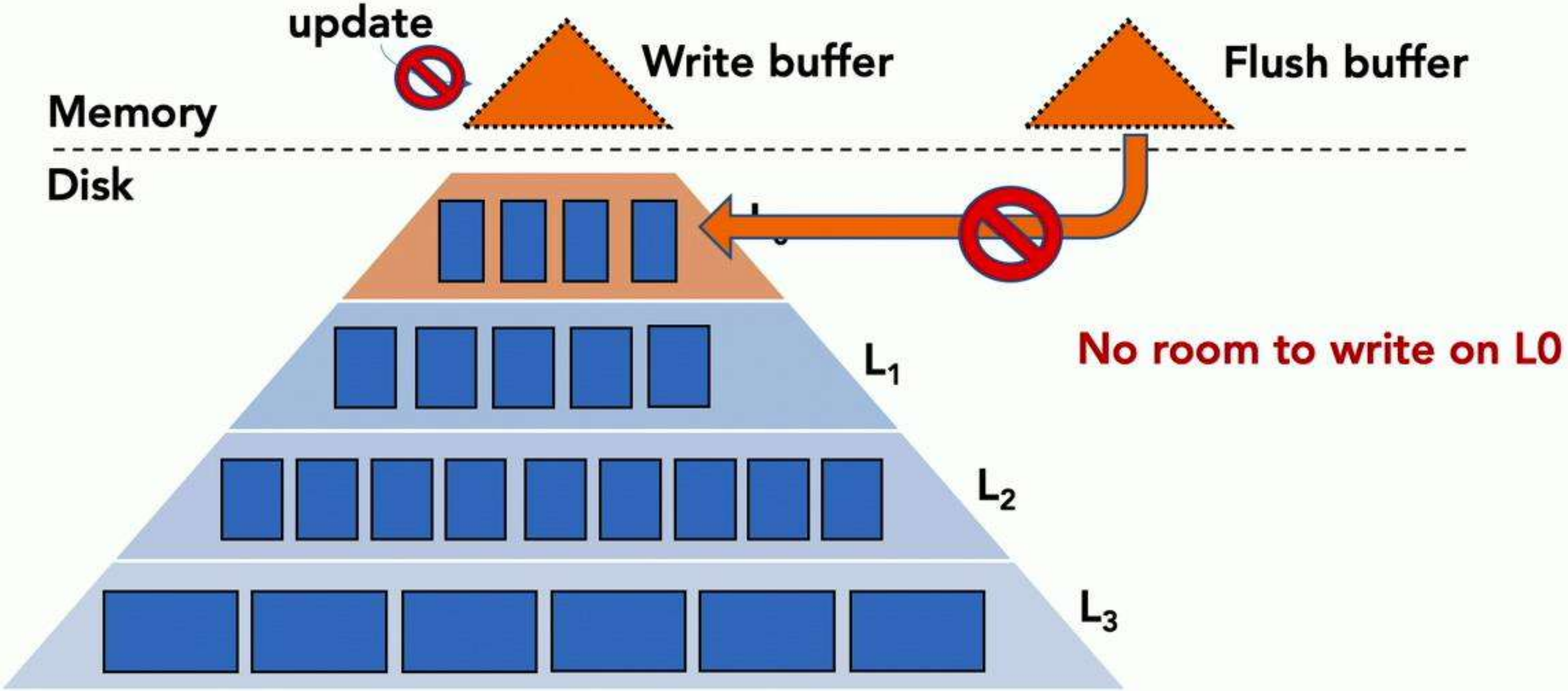
↓

Cannot flush memory component.

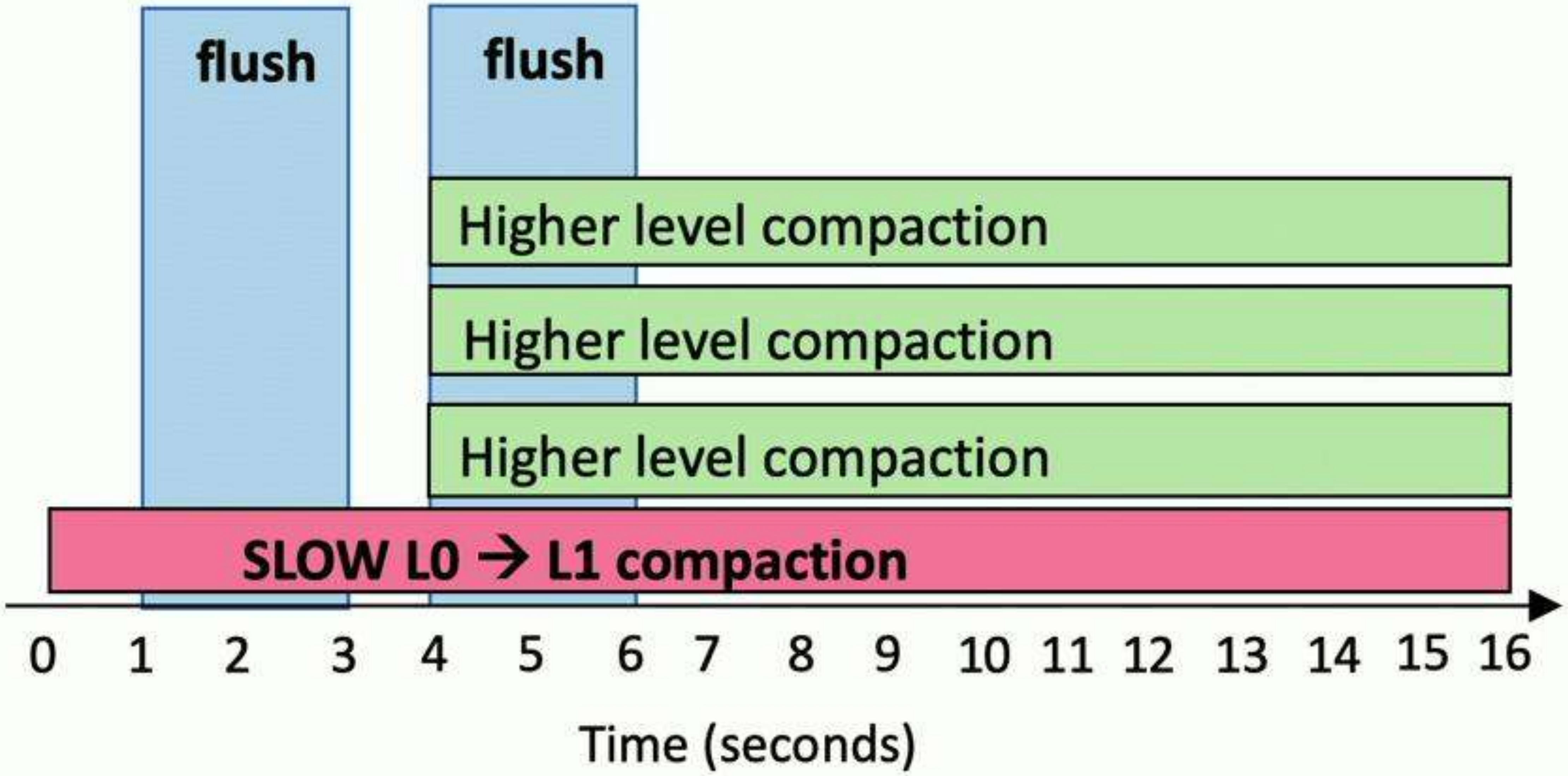
1. Writes Blocked Because L0 is Full.



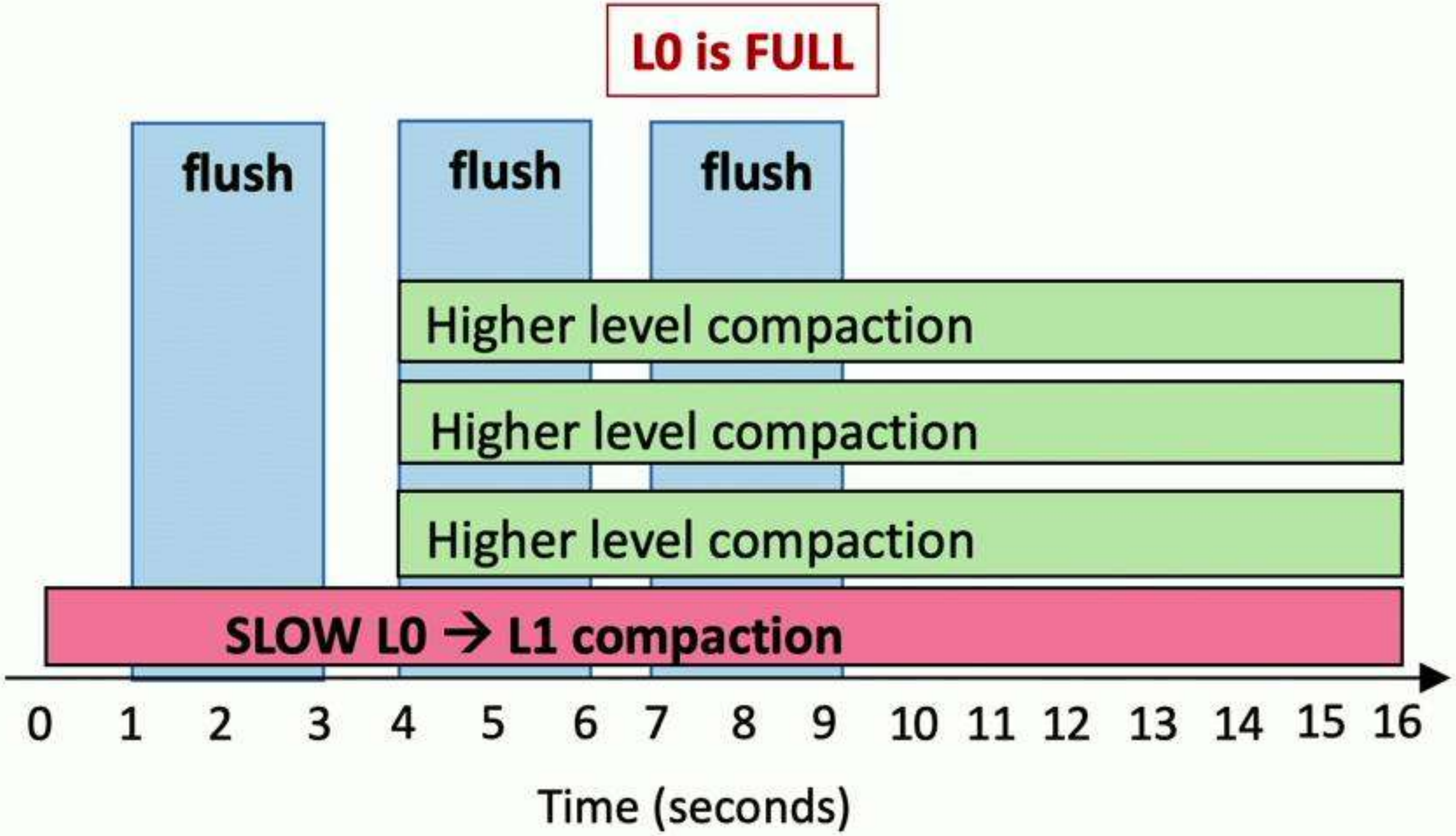
Cannot Flush



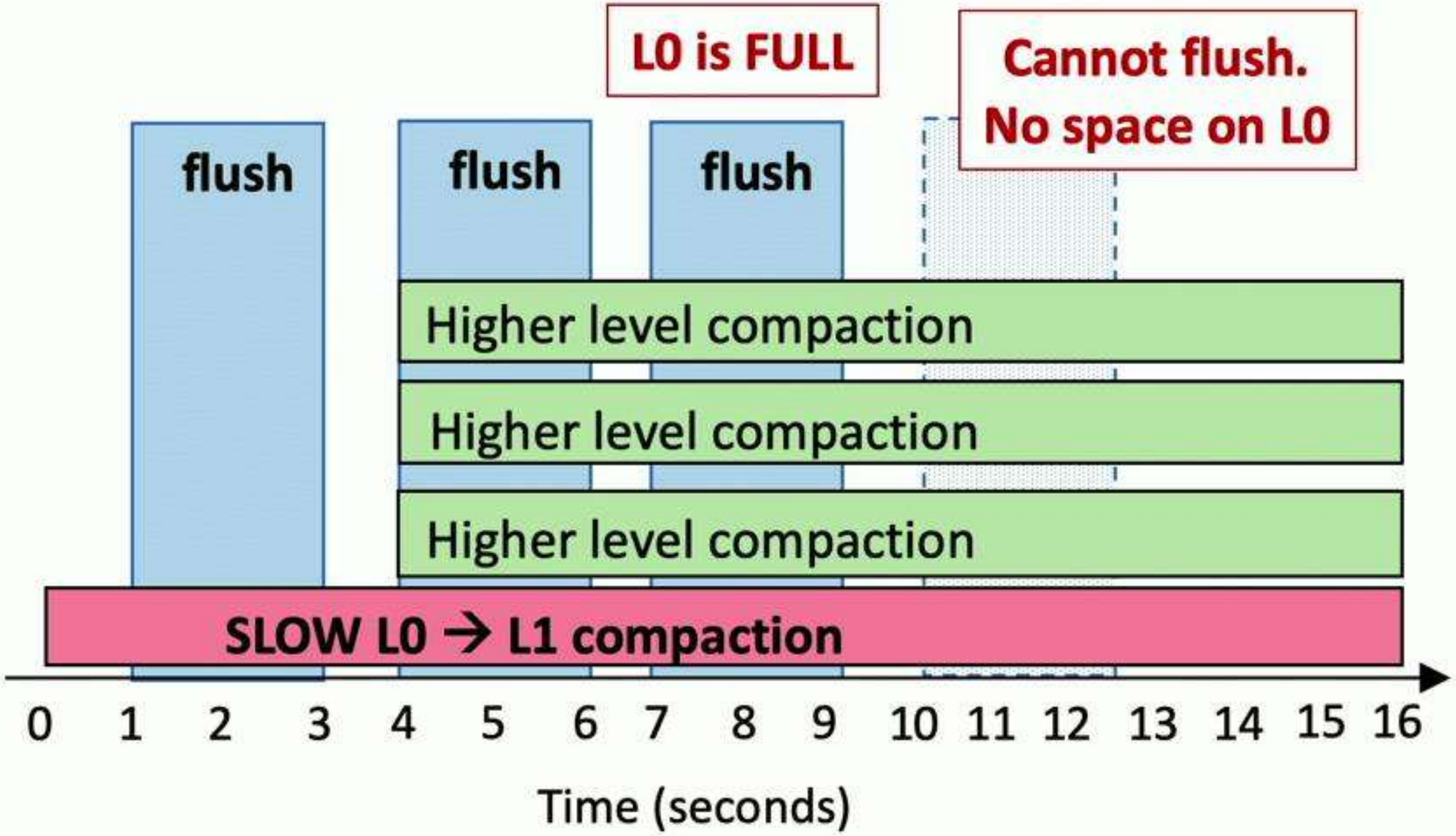
1. Writes Blocked Because L0 is Full.



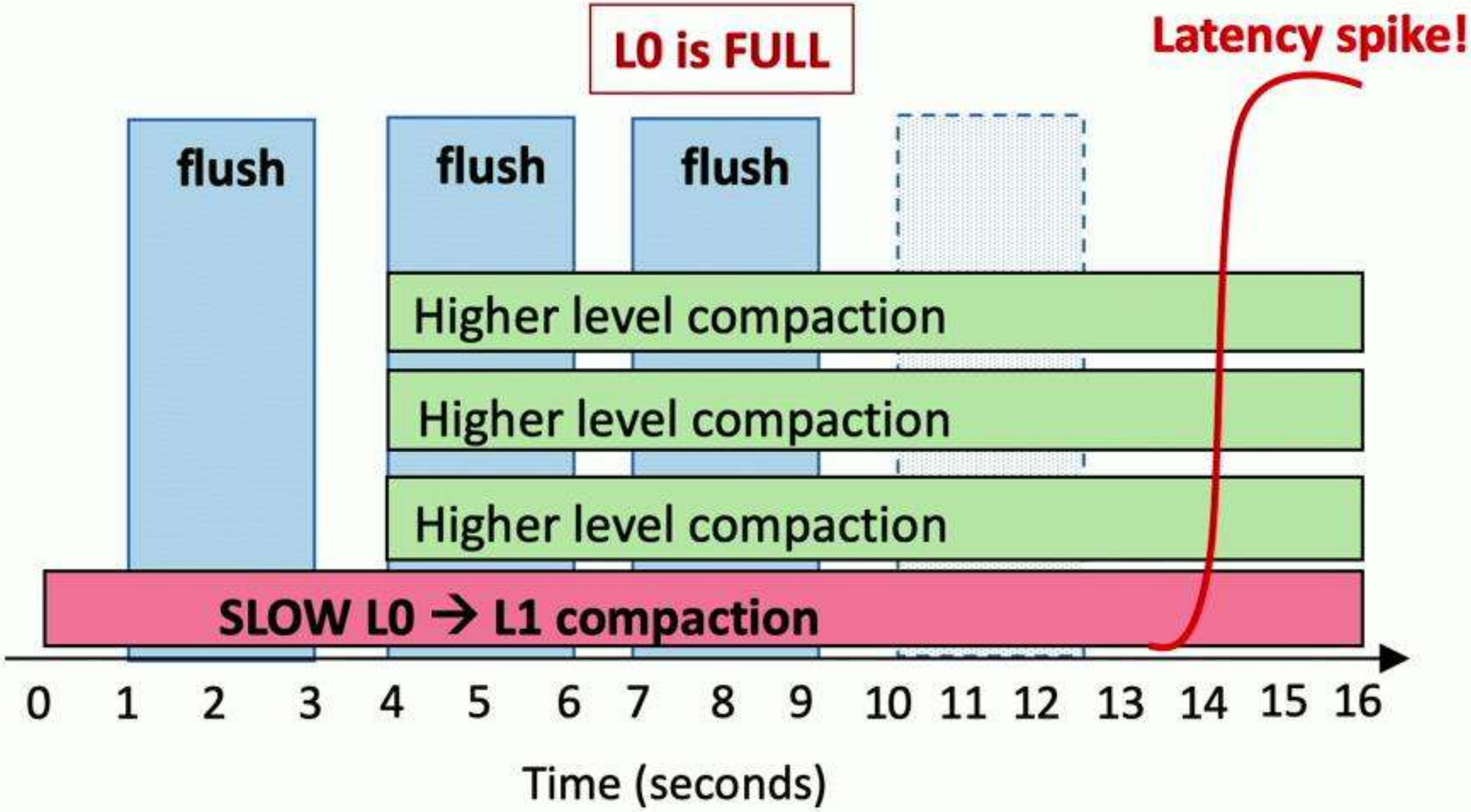
1. Writes Blocked Because L0 is Full.



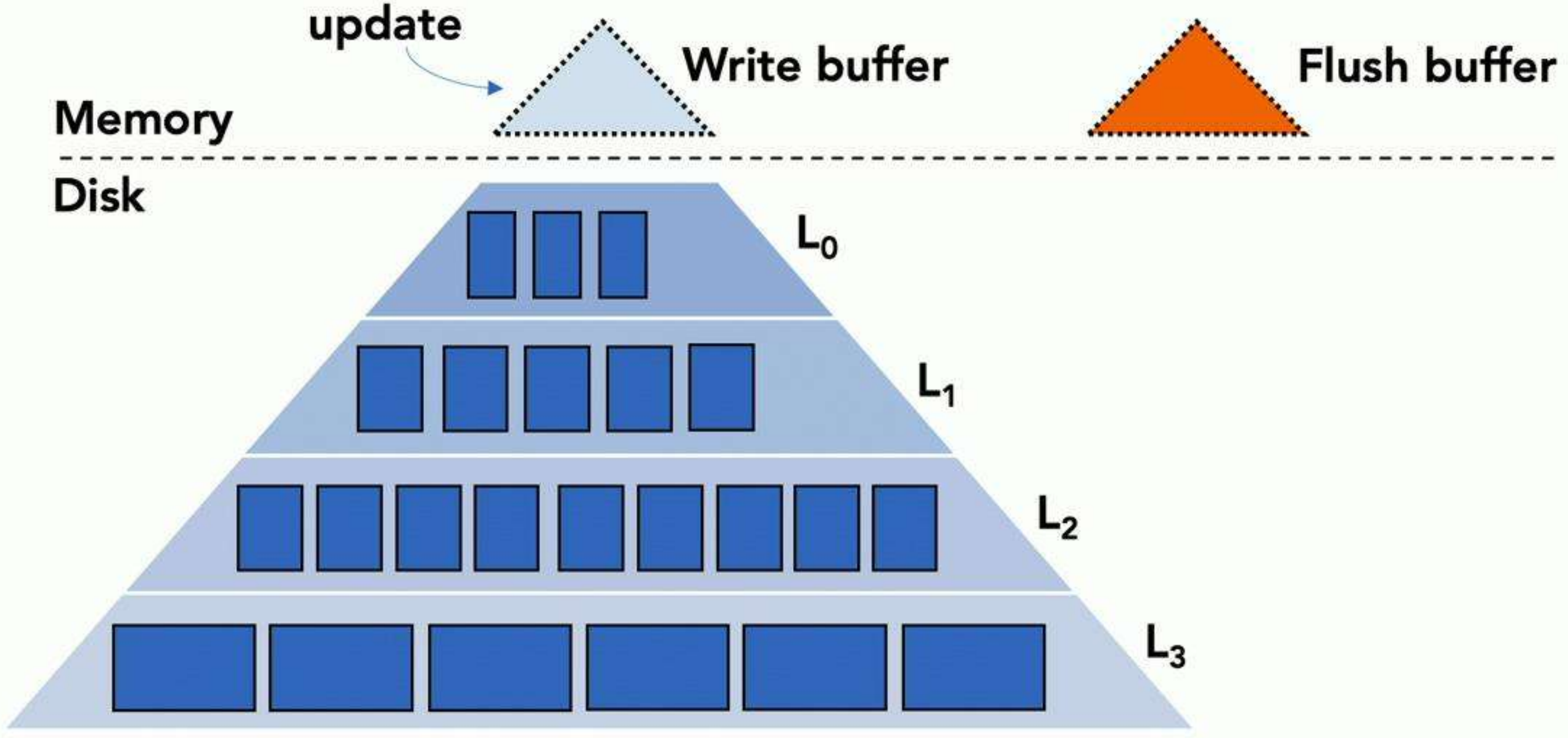
1. Writes Blocked Because L0 is Full.



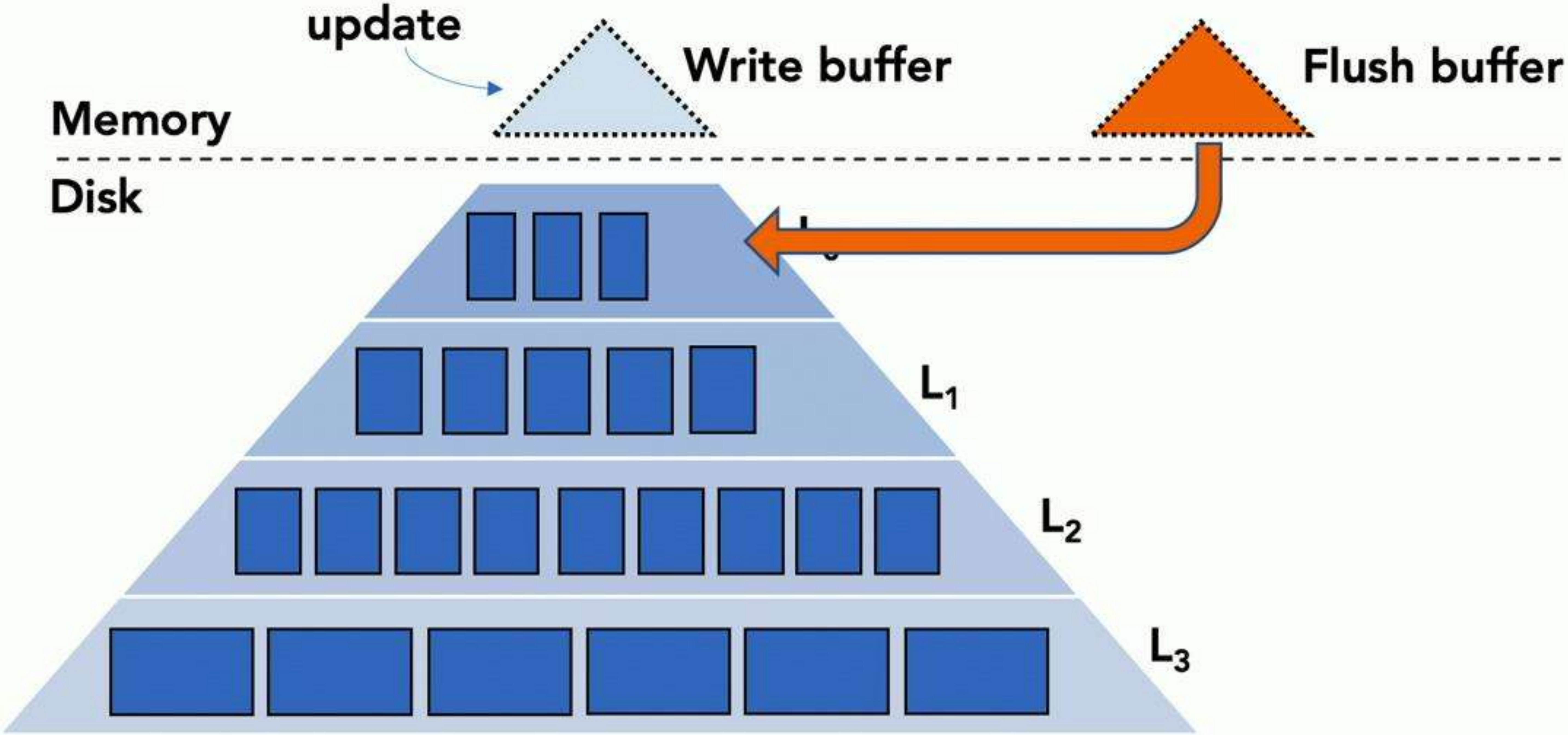
1. Writes Blocked Because L0 is Full.



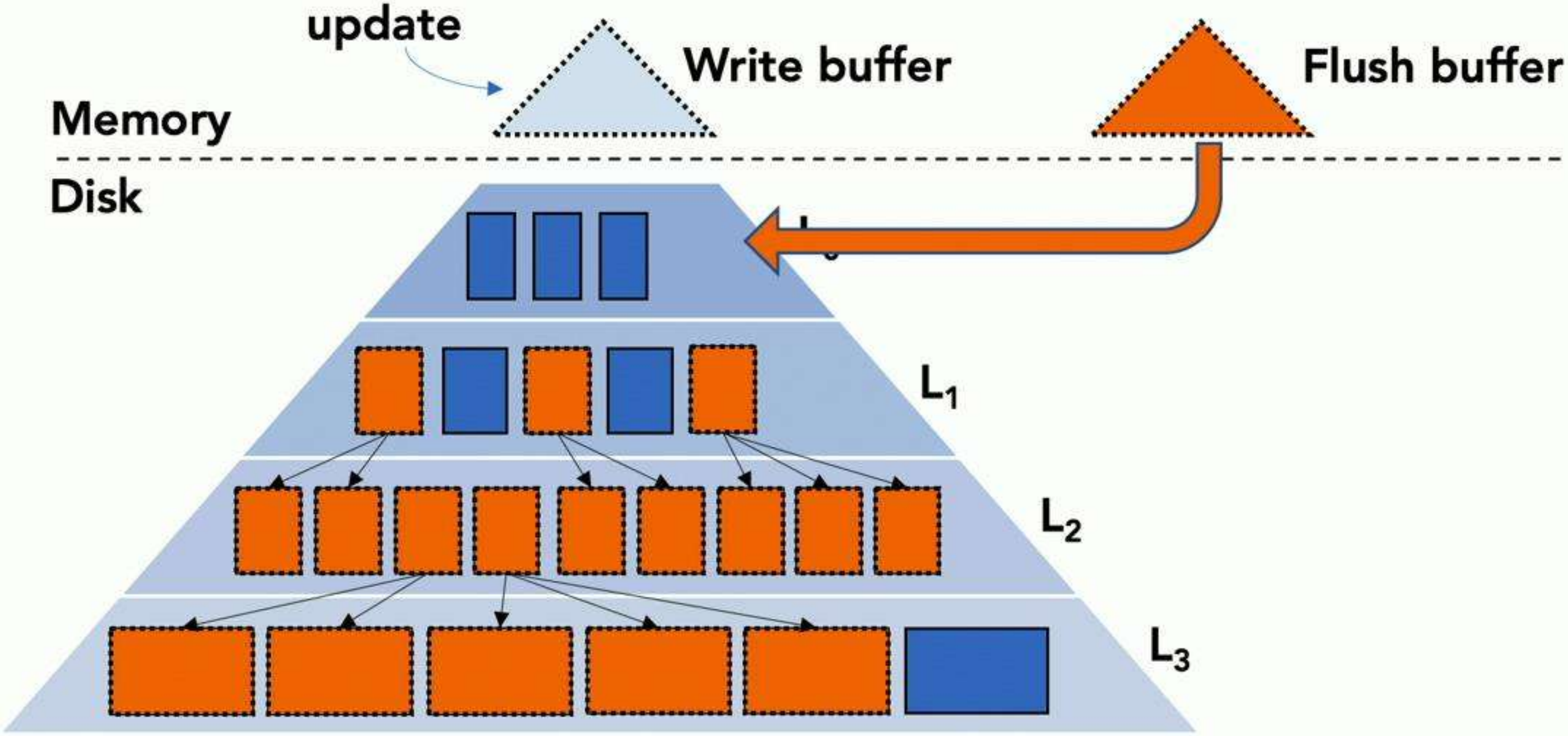
Flushing is Slow



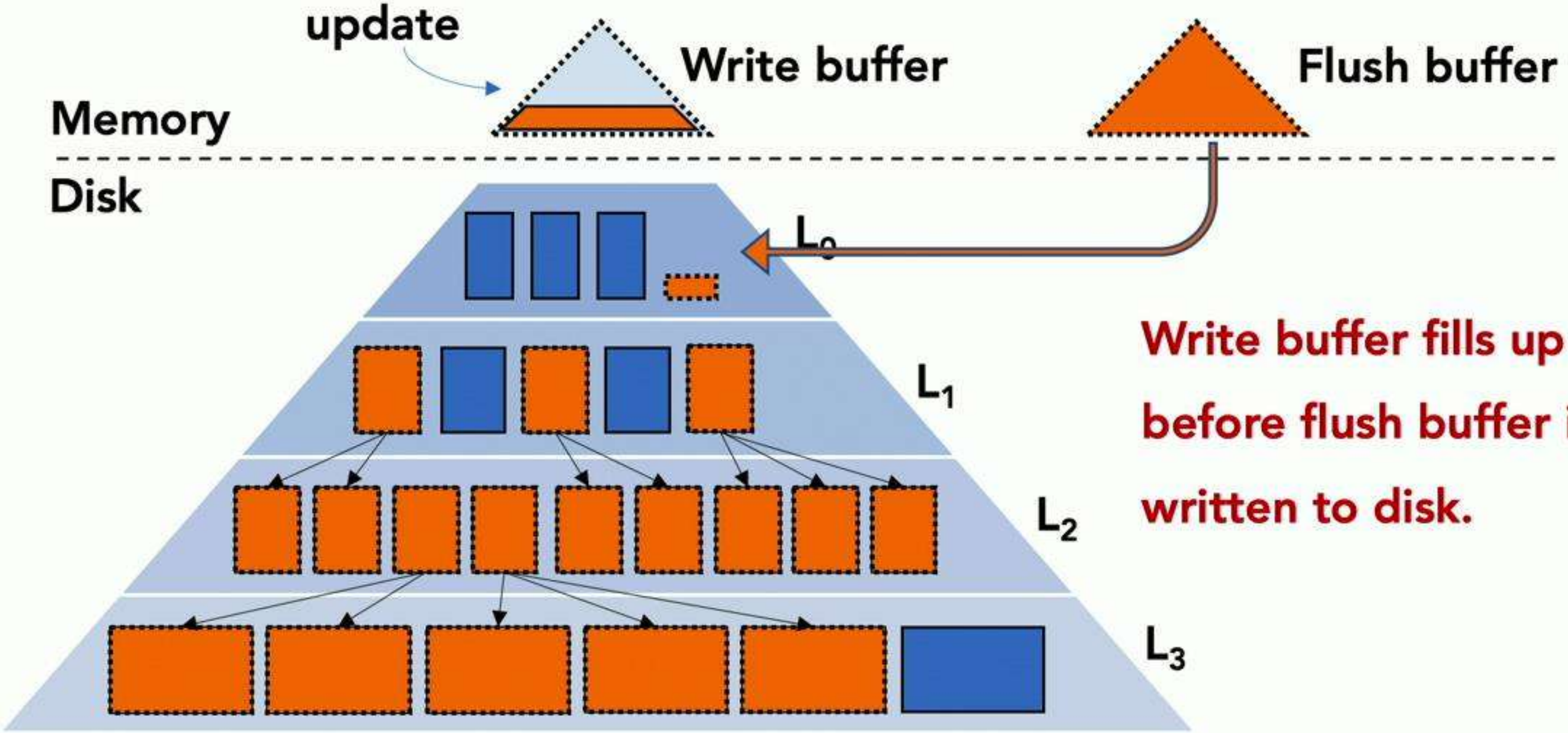
Flushing is Slow



Flushing is Slow

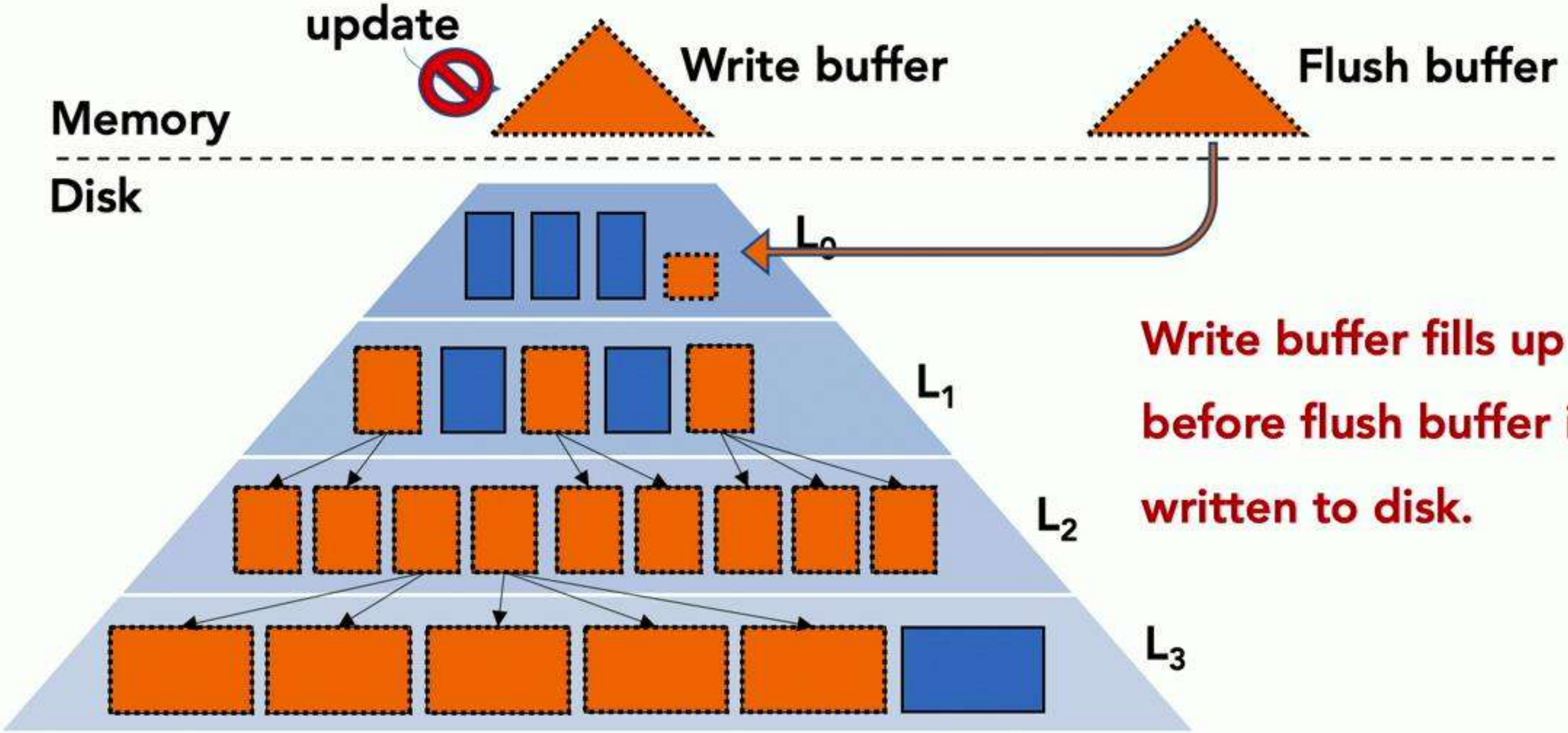


Flushing is Slow



Write buffer fills up before flush buffer is written to disk.

Flushing is Slow



Write buffer fills up before flush buffer is written to disk.

2. Writes Blocked Because Flushing is Slow.

No coordination between internal ops.

↓

Higher level compactions take over I/O.

↓

Flushing does not have enough I/O.

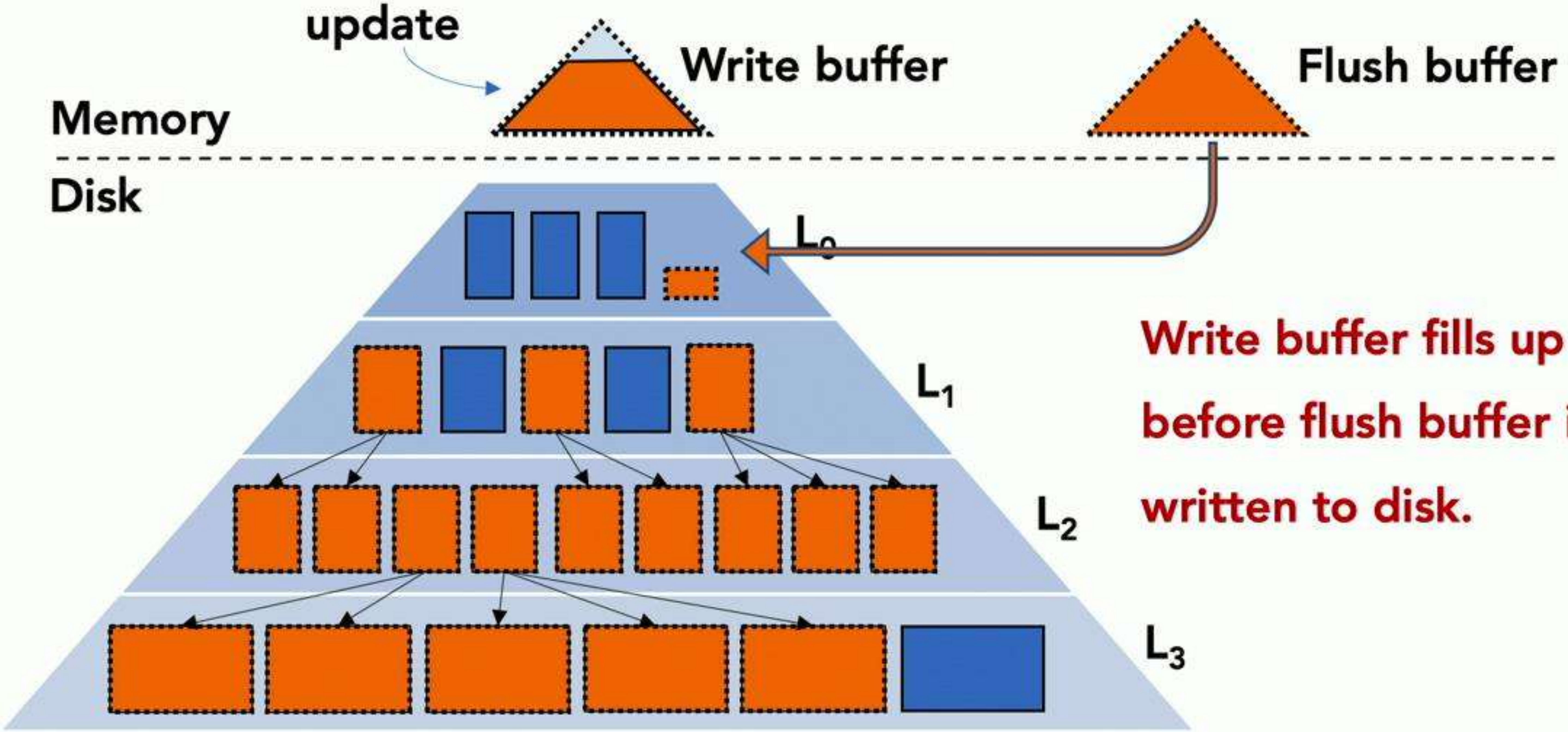
↓

Flushing is very slow.

↓

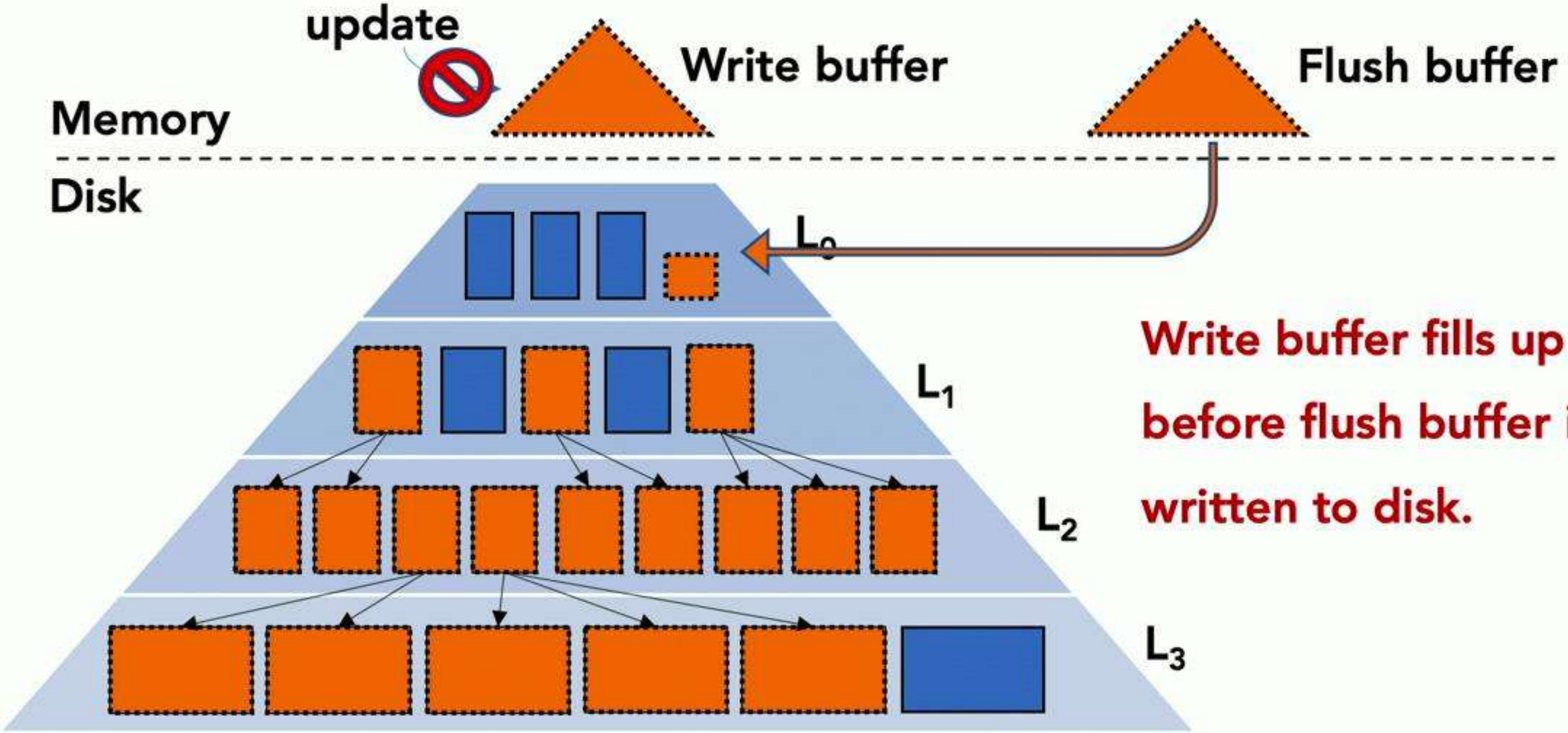
Memory component becomes full.

Flushing is Slow



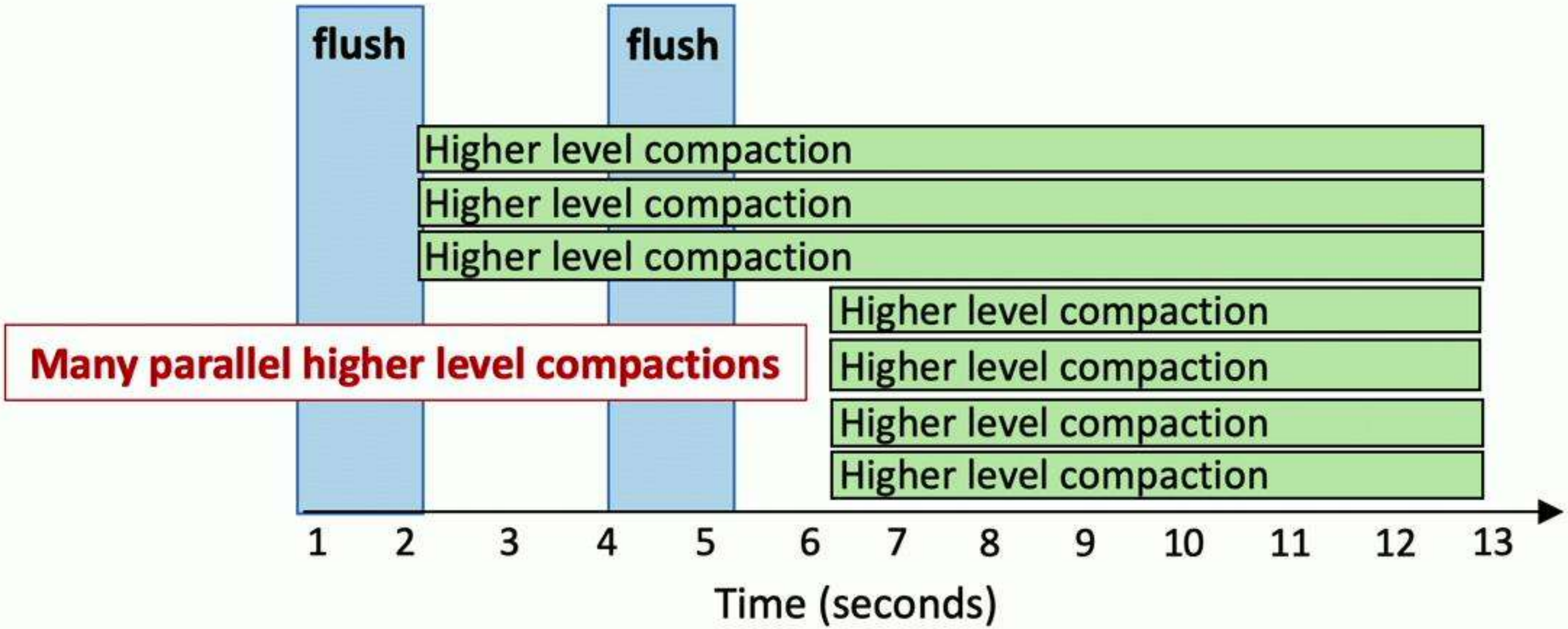
Write buffer fills up before flush buffer is written to disk.

Flushing is Slow

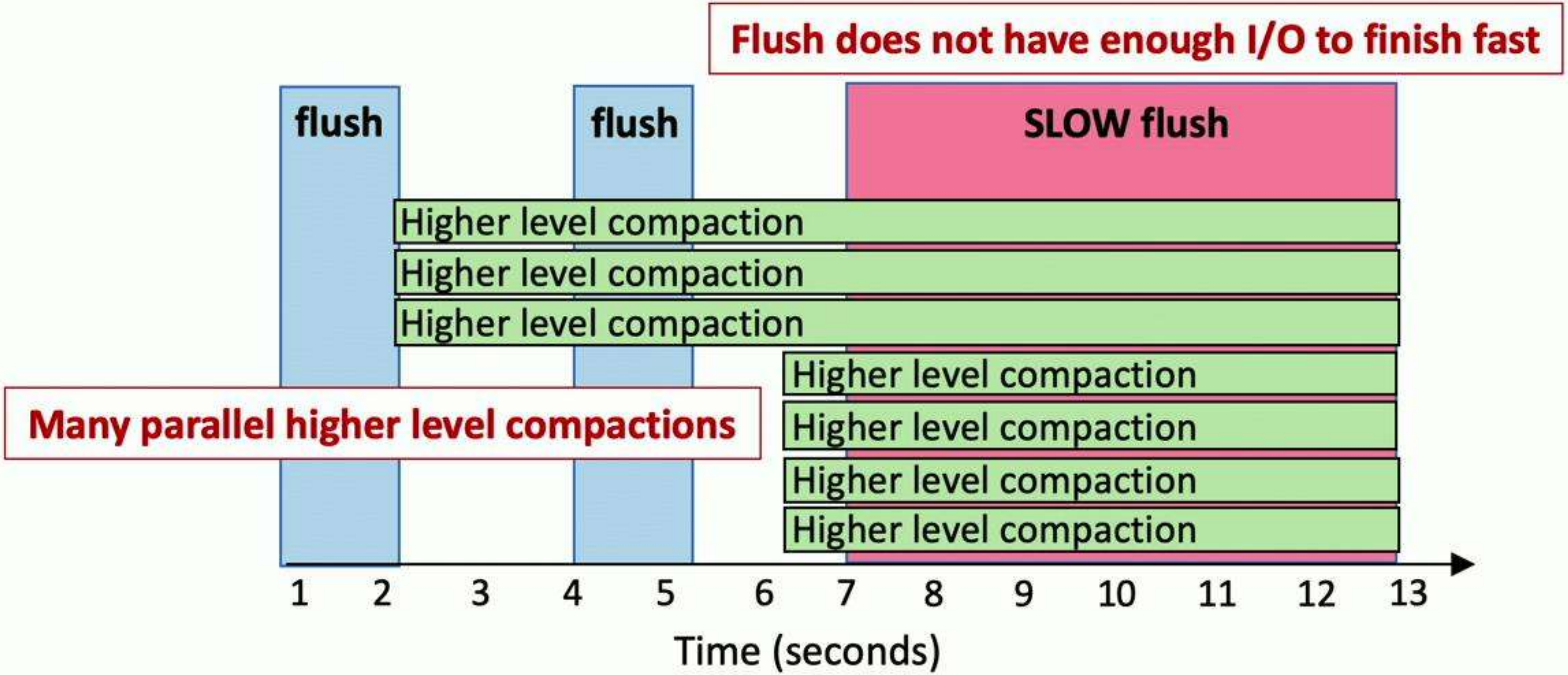


Write buffer fills up before flush buffer is written to disk.

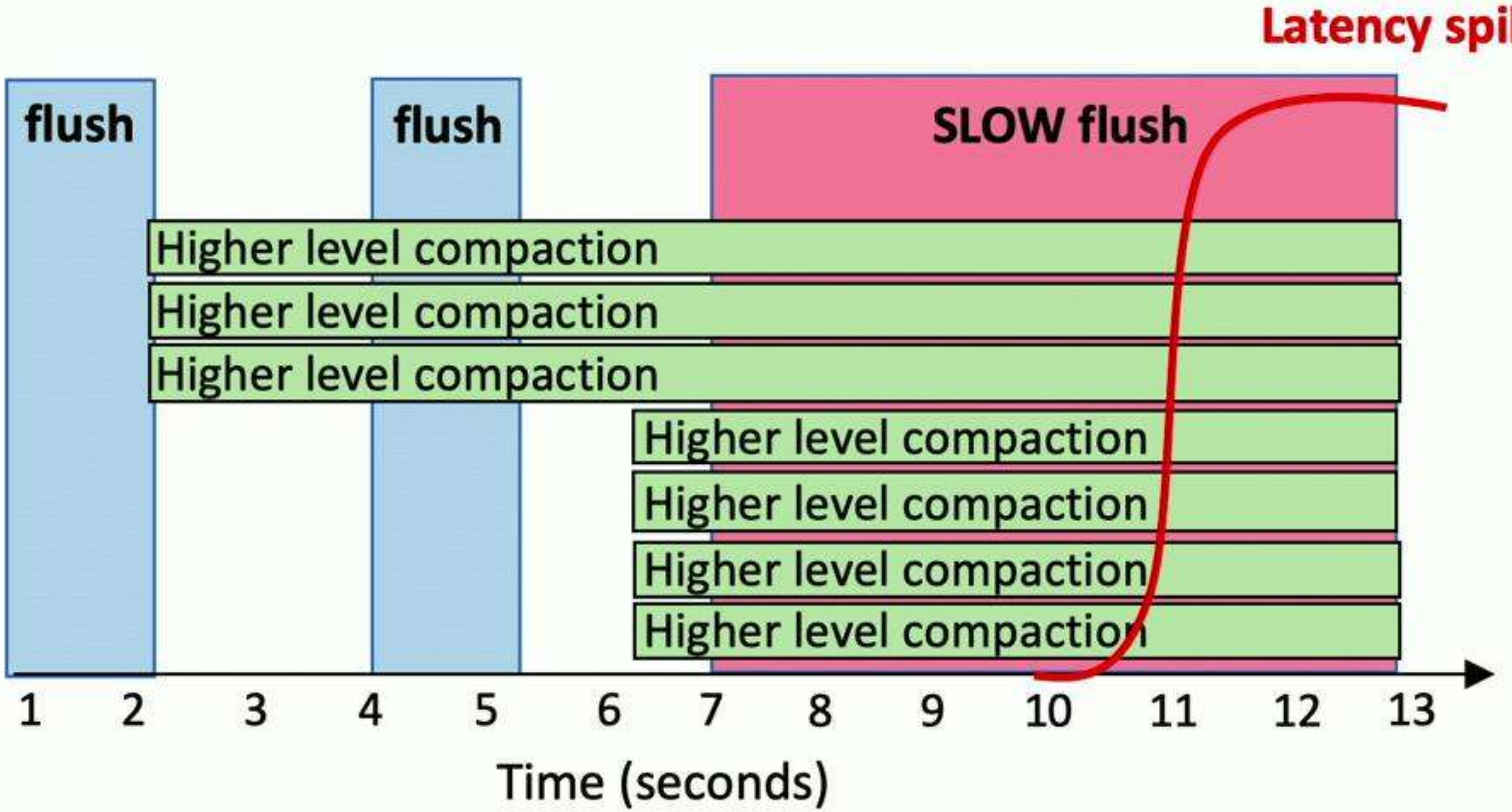
2. Writes Blocked Because Flushing is Slow.



2. Writes Blocked Because Flushing is Slow.

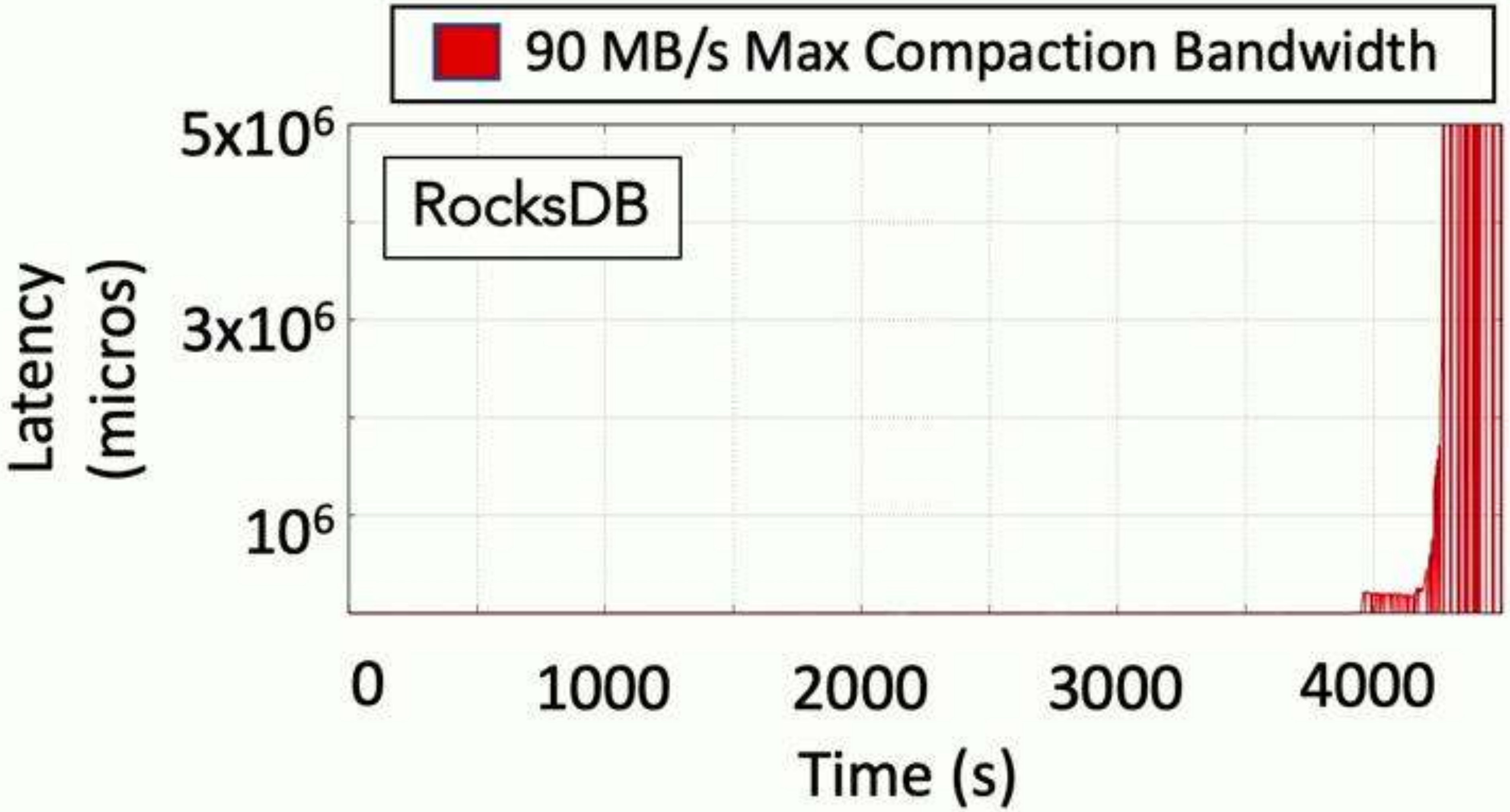


2. Writes Blocked Because Flushing is Slow.



Naïve Solution 1: Compaction Rate Limiting

Rate Limiting: simple attempt to coordinate between internal and external ops.

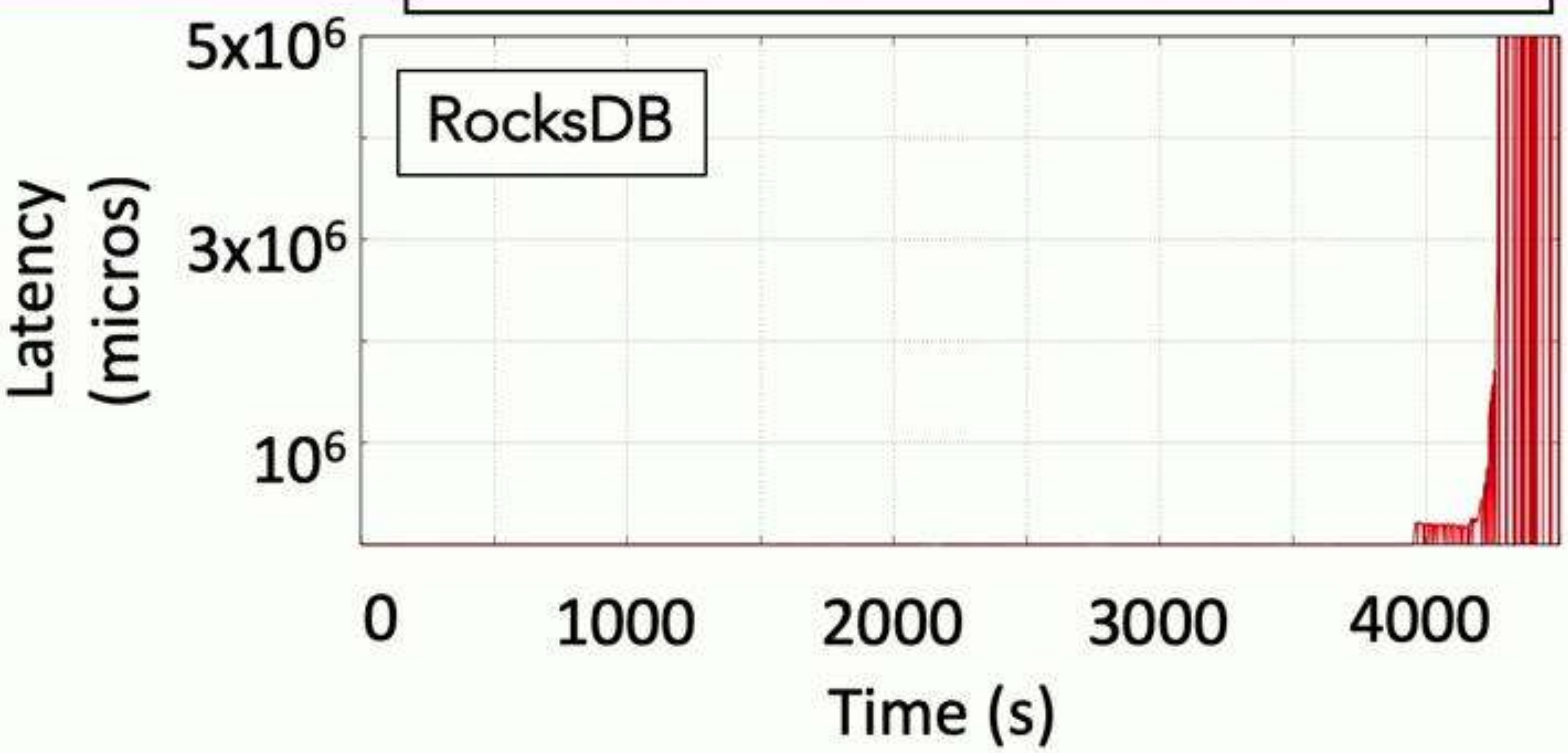


Naïve Solution 1: Compaction Rate Limiting

Rate l… s.

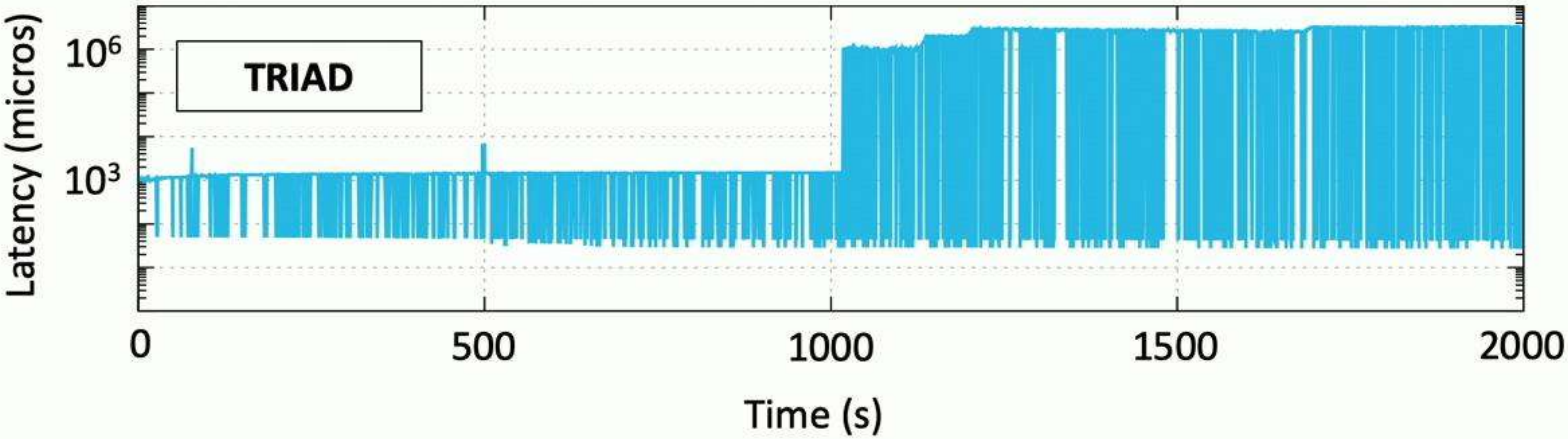
Static compaction rate limiting does not work in the long term.

Chance to run many parallel high level compactions increases.



Naïve Solution 2: Delay Compaction Work

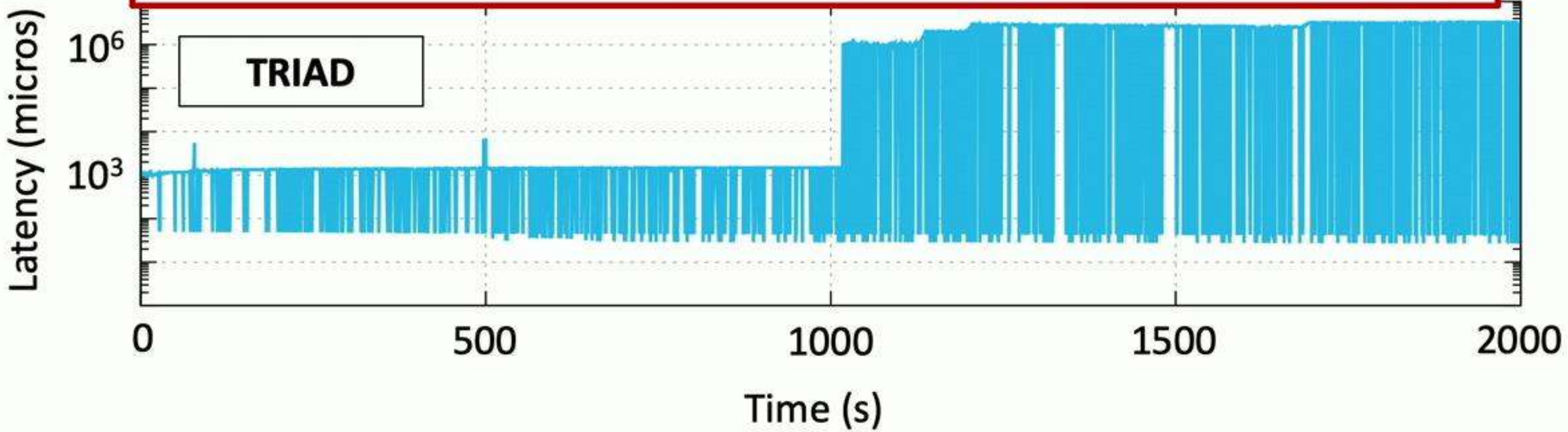
Selective/Delayed Compaction (TRIAD [USENIX ATC '17], PebblesDB [SOSP '17]).



Naïve Solution 2: Delay Compaction Work

Select

Being selective about compactions does not avoid interference.
Eventually need to do the delayed compaction work.



Lessons Learned

1. Make sure L0 is never full.

Lessons Learned

1. Make sure L0 is never full.

2. Ensure sufficient I/O for flush/compactions on low levels.

Lessons Learned

1. Make sure L0 is never full.

2. Ensure sufficient I/O for flush/compactions on low levels.

3. Higher level compactions should not fall behind too much.

The SILK I/O Scheduler

SILK Key Idea

I/O scheduler for LSM KVs: **coordinate I/O bandwidth sharing** to **minimize interference** between internal ops and client ops.

Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/ compactions on low levels.

Make sure other compactions do not fall behind too much.

SILK Design

Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/ compactions on low levels.

Make sure other compactions do not fall behind too much.

SILK Design

Prioritize internal operations at lower levels of the tree.

Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/ compactions on low levels.

Make sure other compactions do not fall behind too much.

SILK Design

Prioritize internal operations at lower levels of the tree.

Preempt higher level compactions if necessary.

Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/ compactions on low levels.

Make sure other compactions do not fall behind too much.

SILK Design

Prioritize internal operations at lower levels of the tree.

Preempt higher level compactions if necessary.

Opportunistically allocate I/O for higher level compactions.

Prioritize & Preempt

Prioritize internal ops at **lower tree levels:**

1. First priority: **Flushing**
2. Second priority: **L0 → L1 compactions**
3. Third priority: **Higher level compactions**

Prioritize & Preempt

Prioritize internal ops at **lower tree levels:**

1. **Flushing** – *dedicated flush operation queue.*
2. **L0 → L1 compactions**
3. **Higher level compactions**

Prioritize & Preempt

Prioritize internal ops at **lower tree levels:**

1 **Flushing** – *dedicated flush operation queue.*

2 **L0 → L1 compactions**

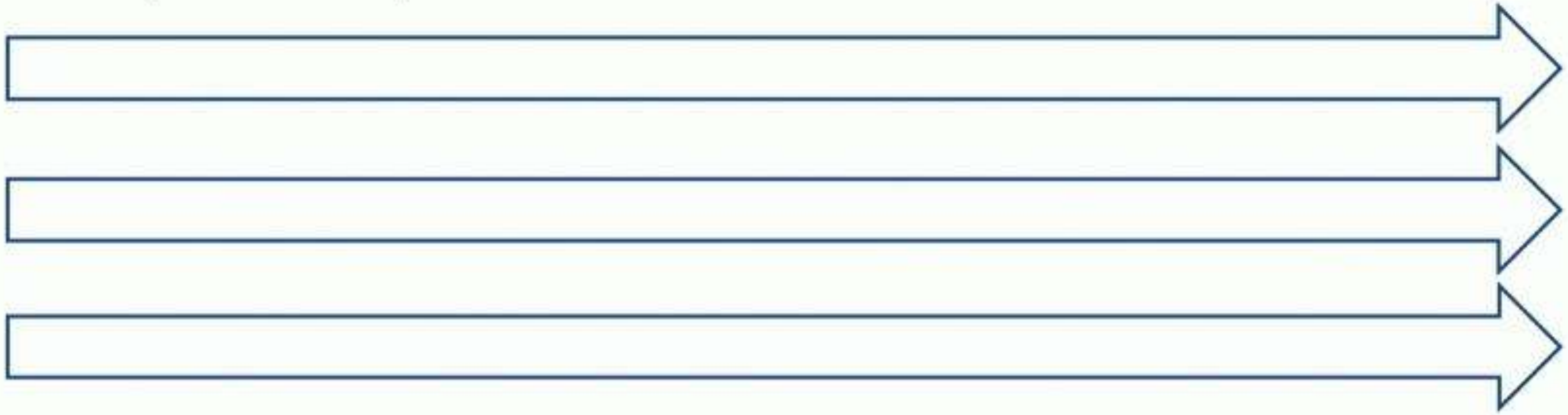
3 **Higher level compactions**

L0 → L1 compaction preempts higher level compactions.

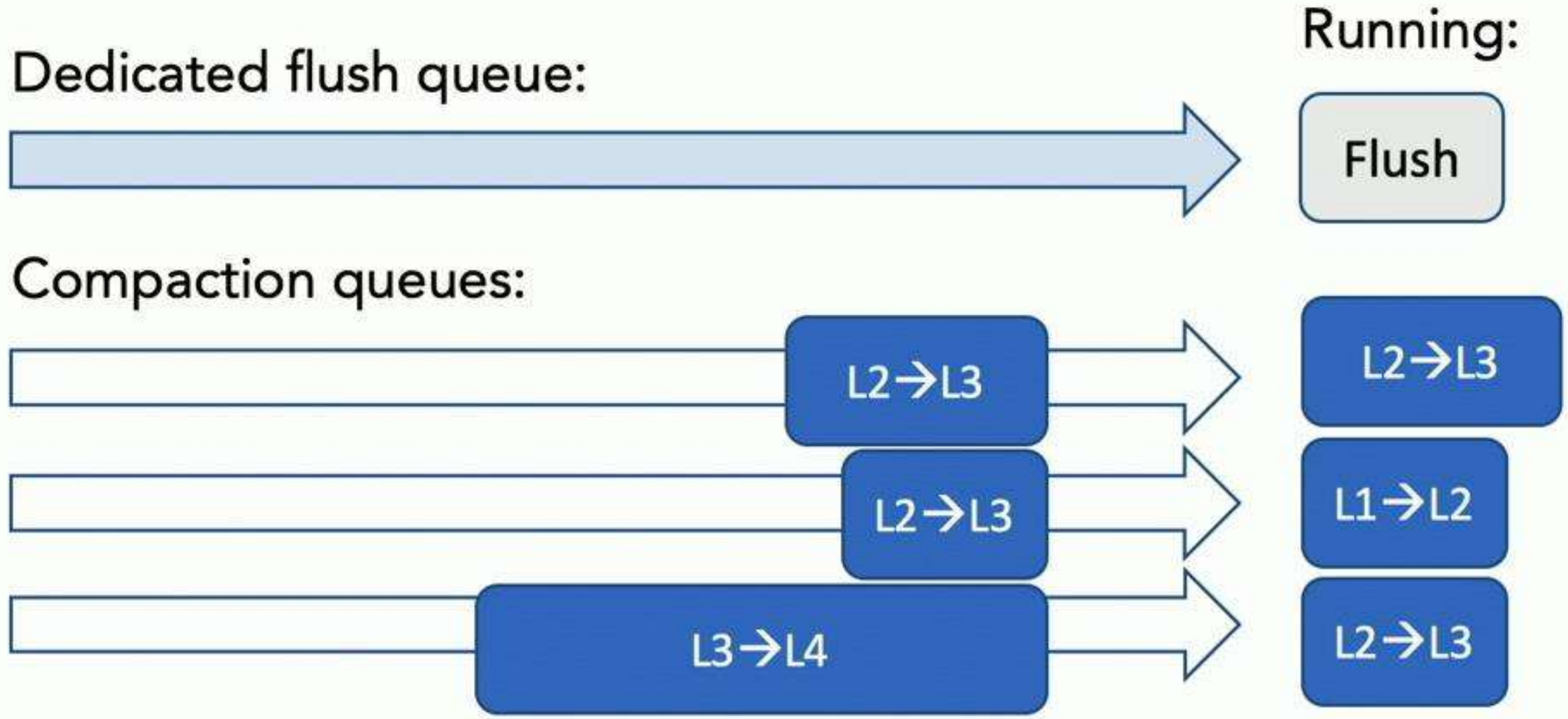
2. Preempt

Dedicated flush queue:

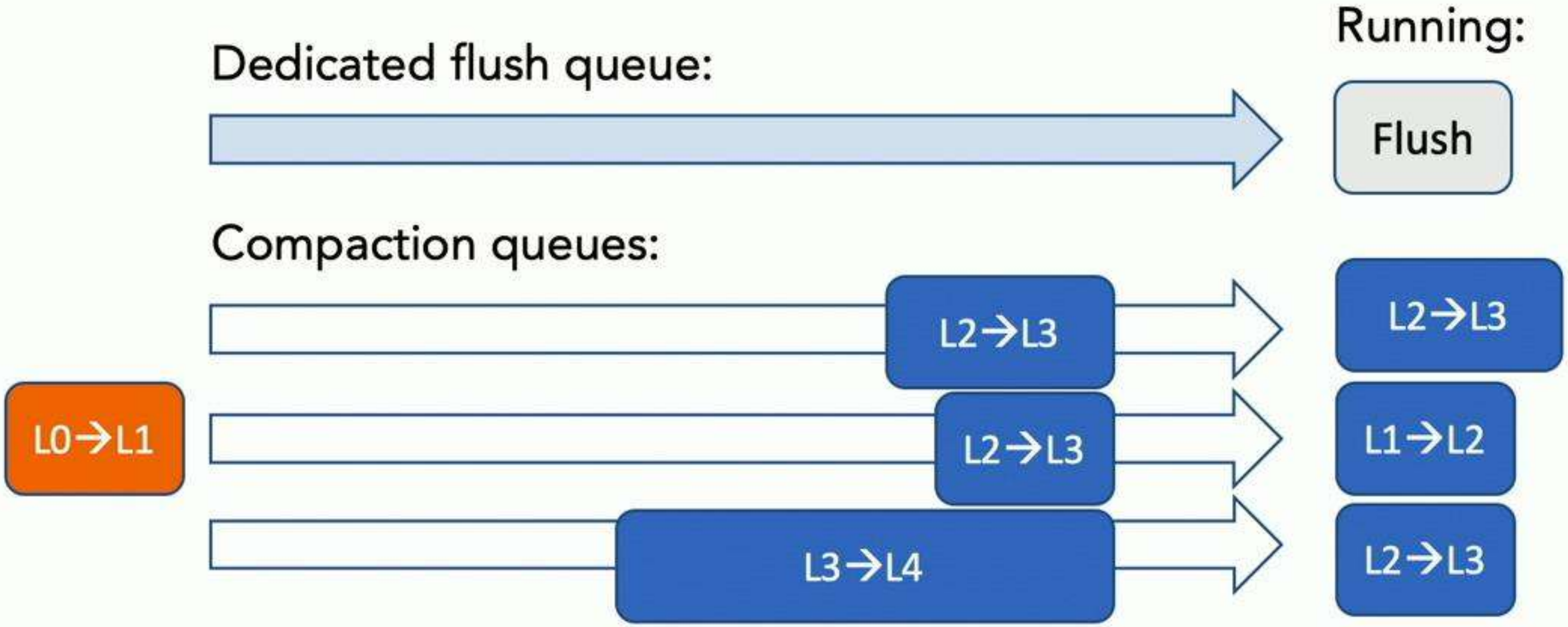
Compaction queues:



2. Preempt

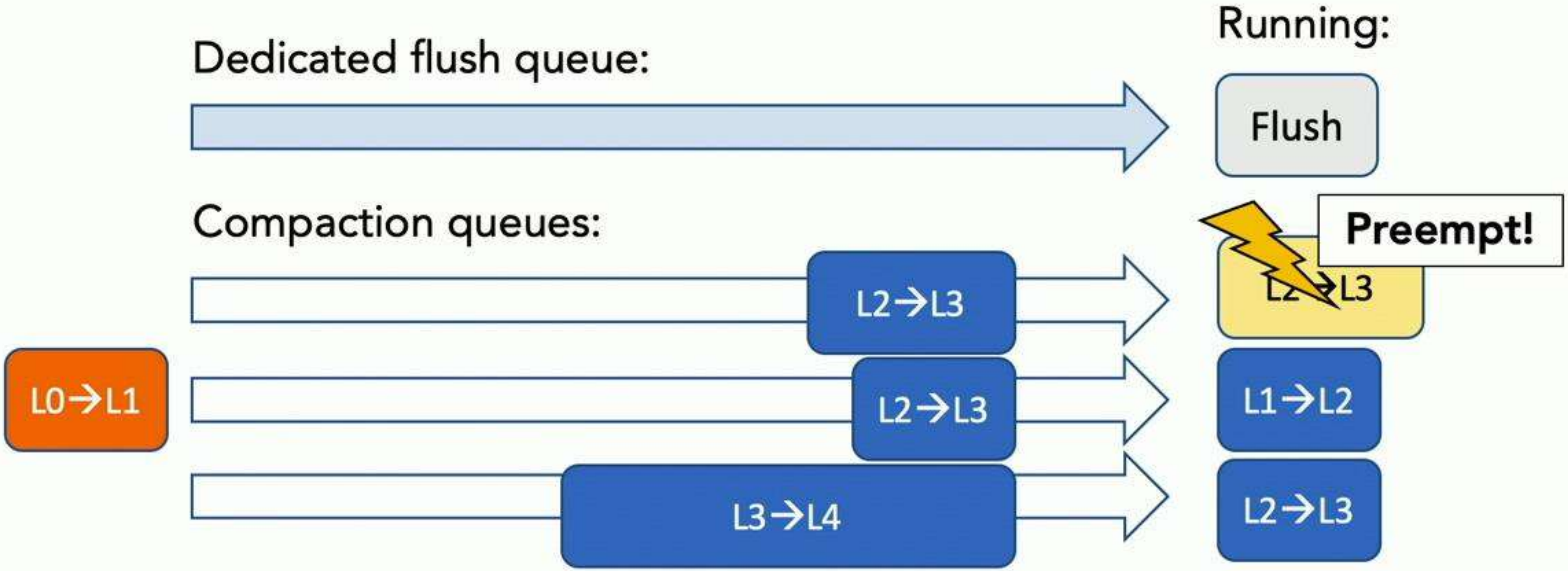

Dedicated flush queue:
Running:
Flush
Compaction queues:
L2→L3
L2→L3
L3→L4
L2→L3
L1→L2
L2→L3

2. Preempt

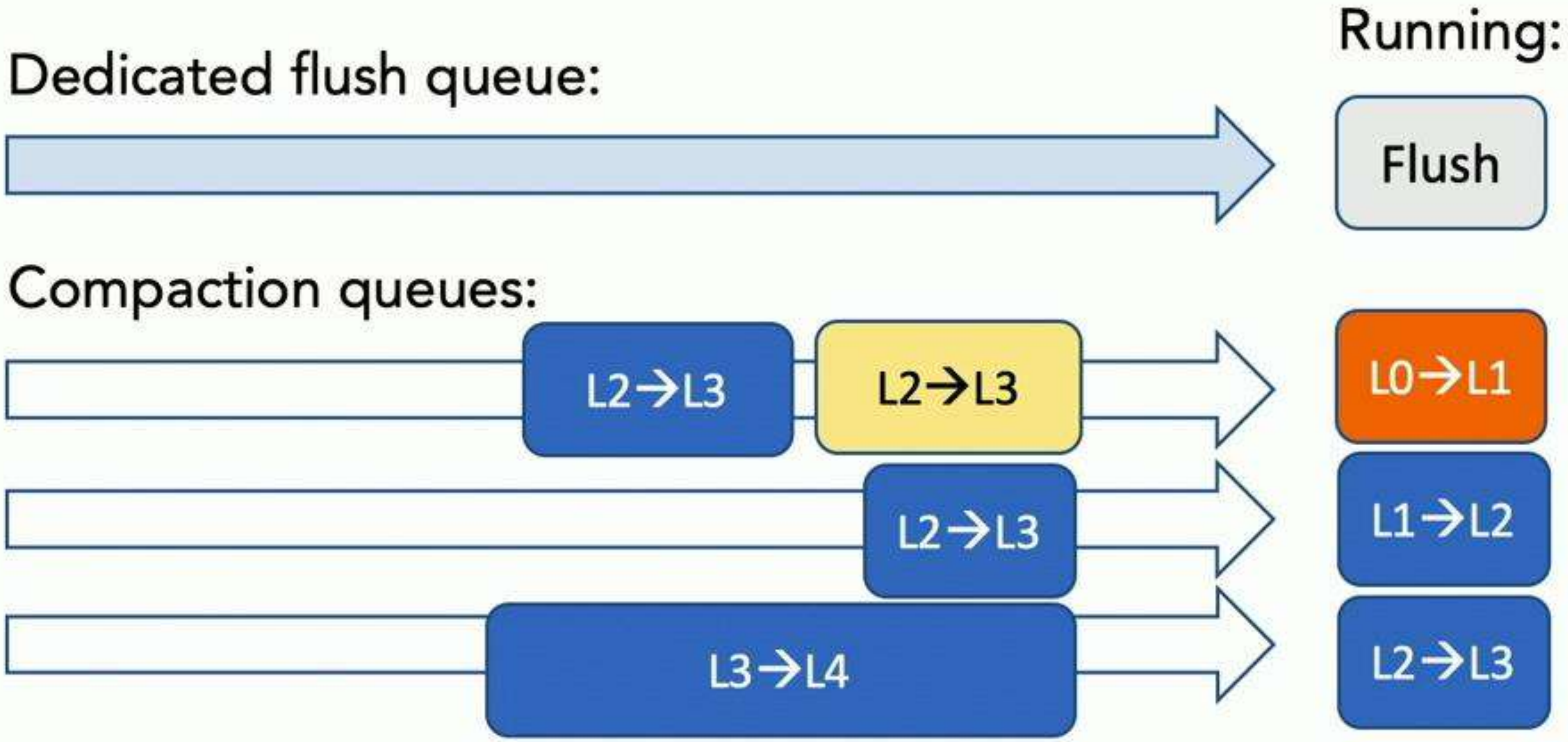


Dedicated flush queue:
Running:
Flush
Compaction queues:
L2→L3
L0→L1
L2→L3
L3→L4
L2→L3
L1→L2
L2→L3

2. Preempt


Running:
Dedicated flush queue:
Flush
Compaction queues:
Preempt!
L2→L3
L2→L3
L2→L3
L0→L1
L1→L2
L3→L4
L2→L3

2. Preempt

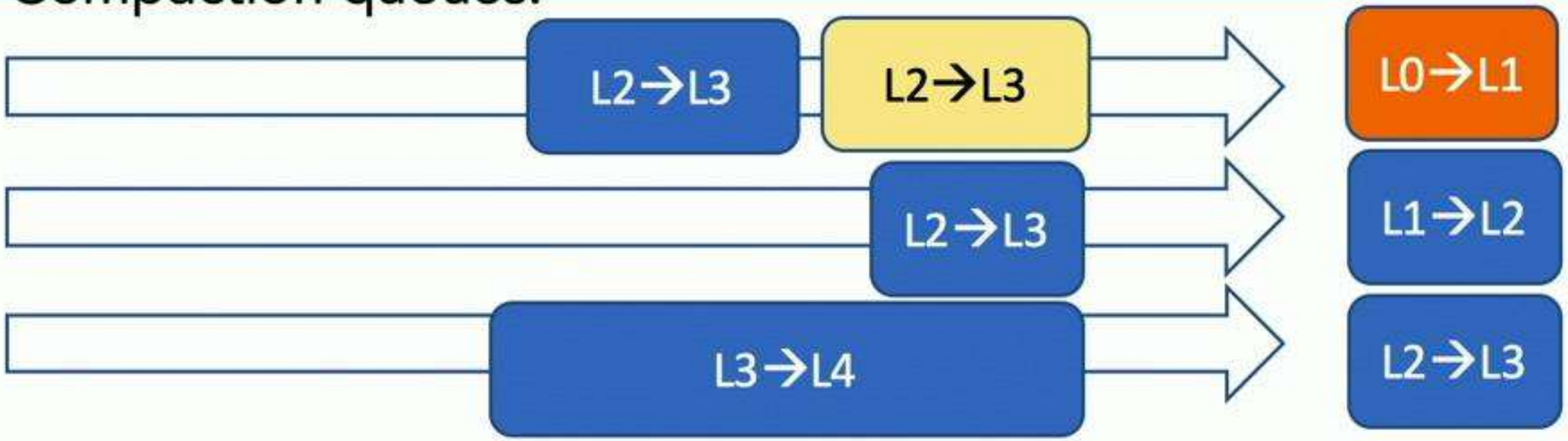

Running:
Dedicated flush queue:
Flush
Compaction queues:
L2→L3
L2→L3
L0→L1
L2→L3
L1→L2
L3→L4
L2→L3

2. Preempt

Running:

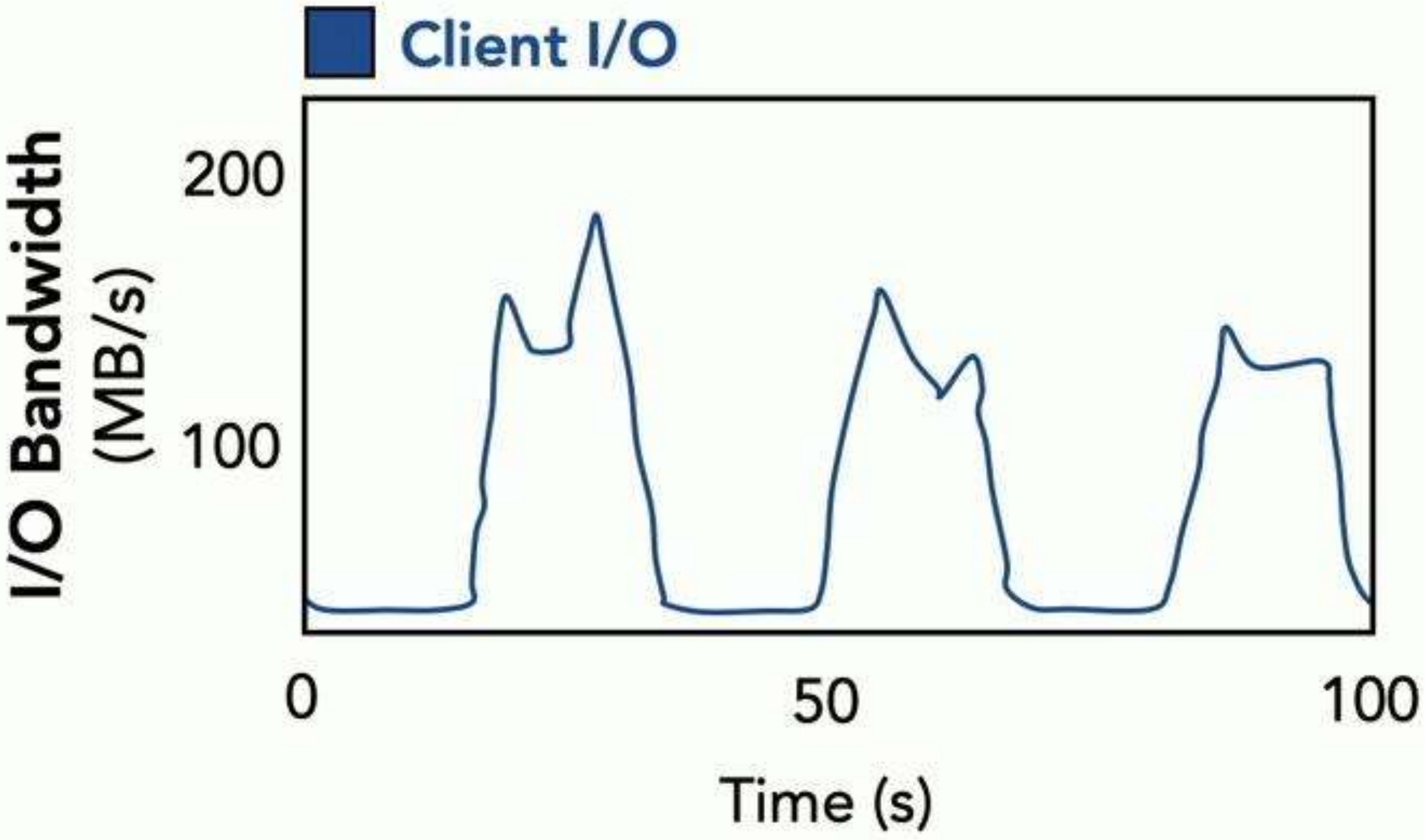
☺ L0→L1 compactions never wait behind higher level compactions

Compaction queues:



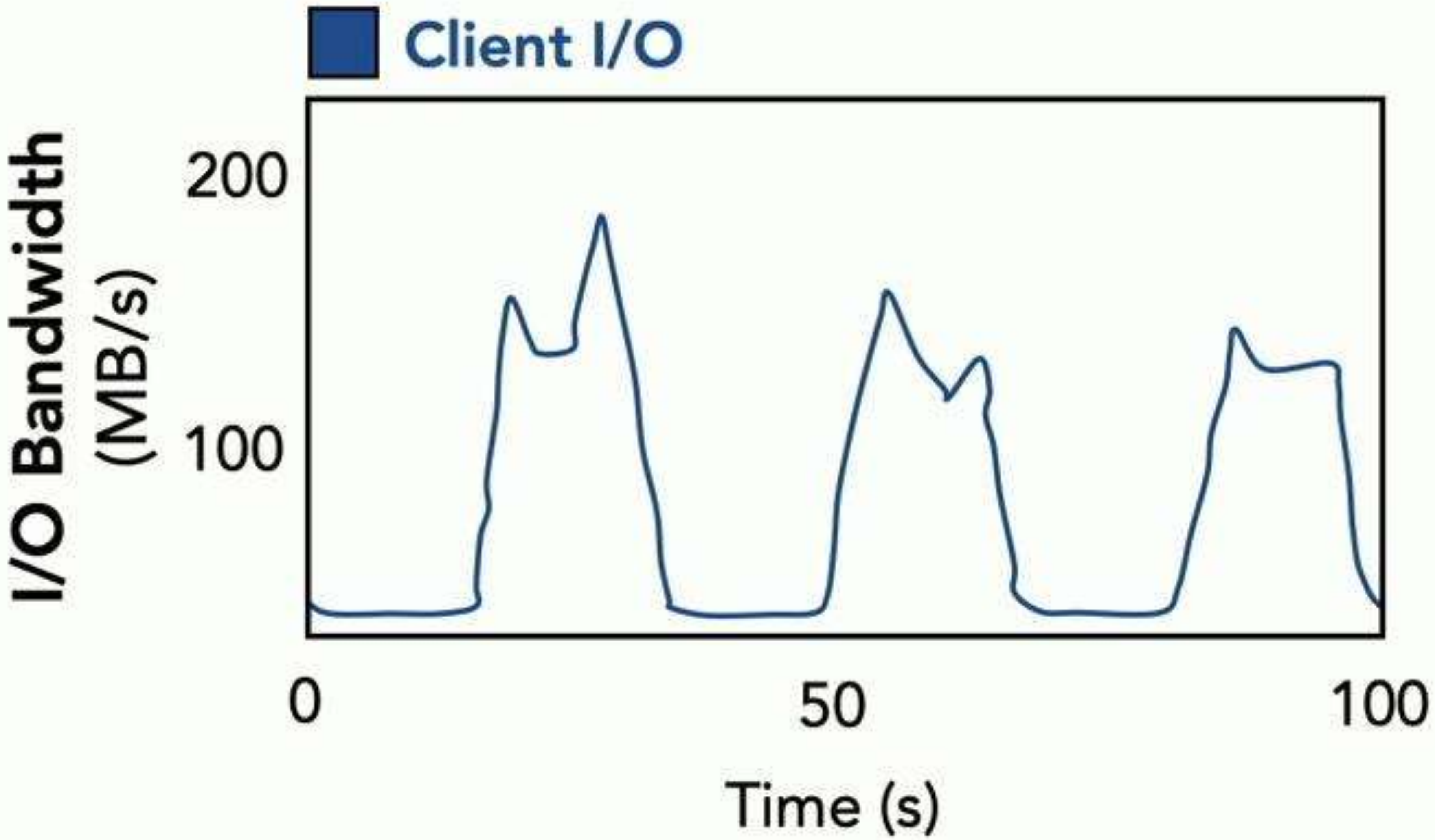
Opportunistically allocate I/O for compactions

Real Nutanix client load example



Opportunistically allocate I/O for compactions

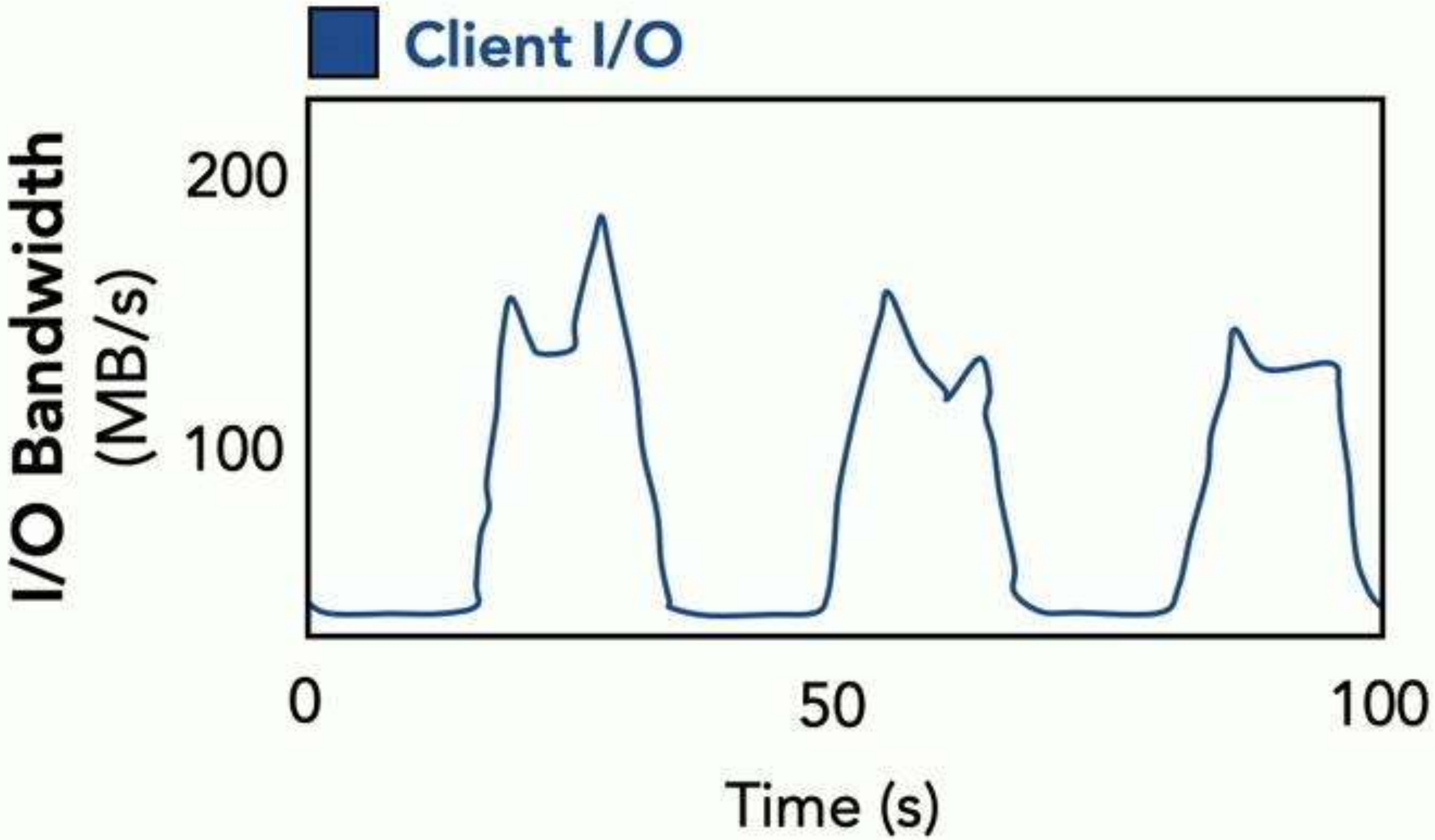
Real Nutanix client load example



Client workload is **not constant**.

Opportunistically allocate I/O for compactions

Real Nutanix client load example

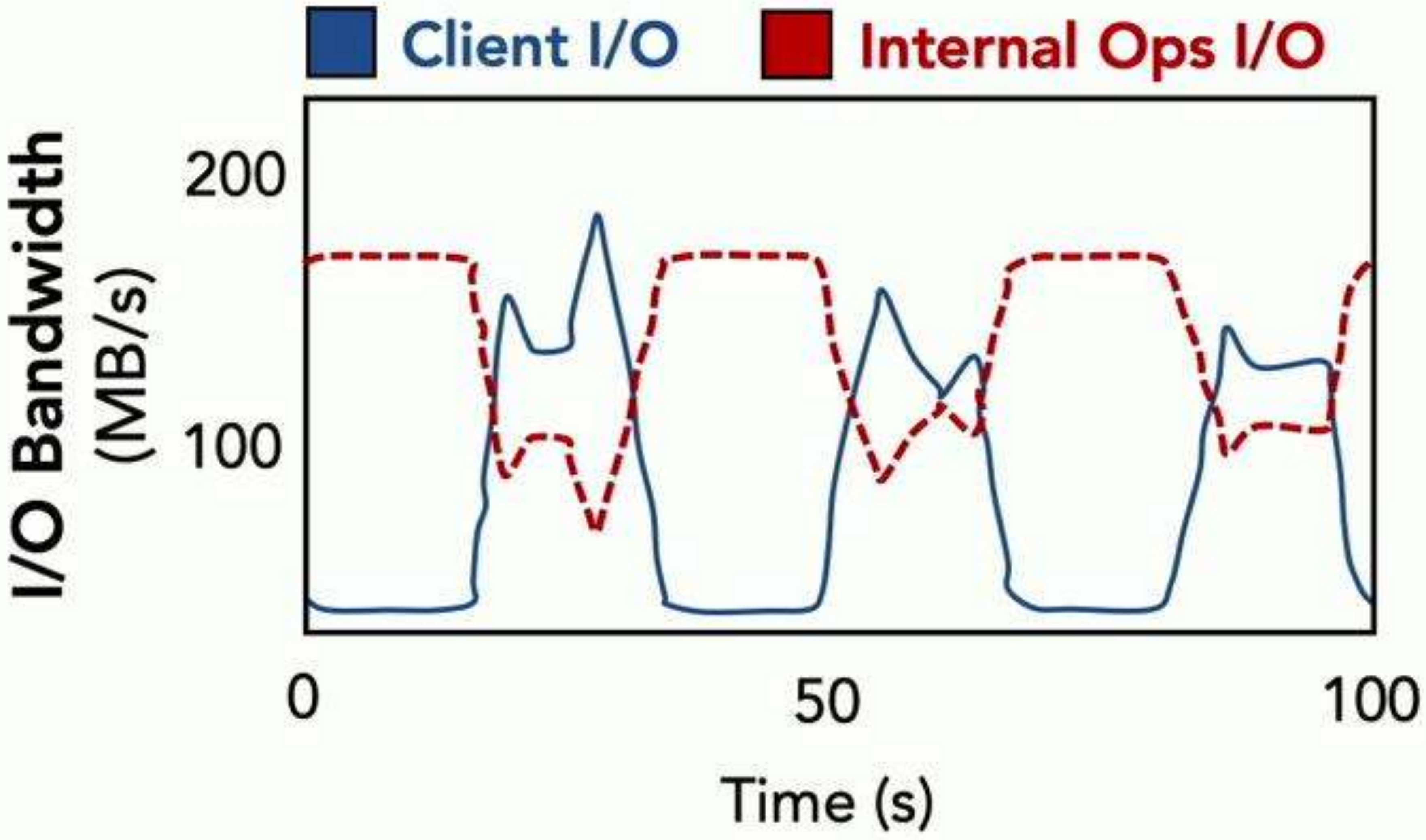


Client workload is **not constant.**

SILK **continuously monitors** client **I/O bandwidth** use.

Opportunistically allocate I/O for compactions

Real Nutanix client load example

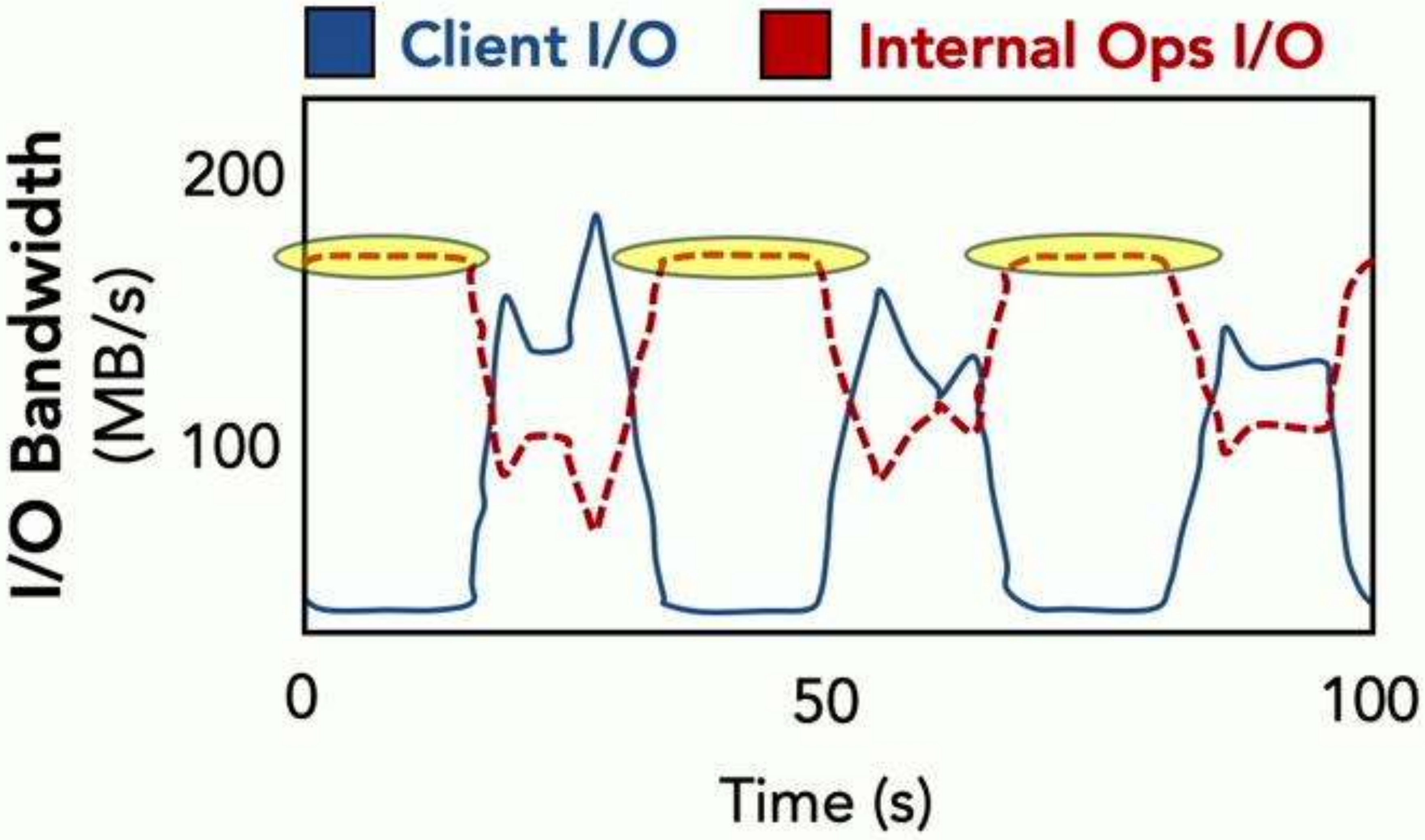


Allocate **less I/O to compactions** during **client peaks.**

Allocate **more I/O to compactions** during **client low load.**

Opportunistically allocate I/O for compactions

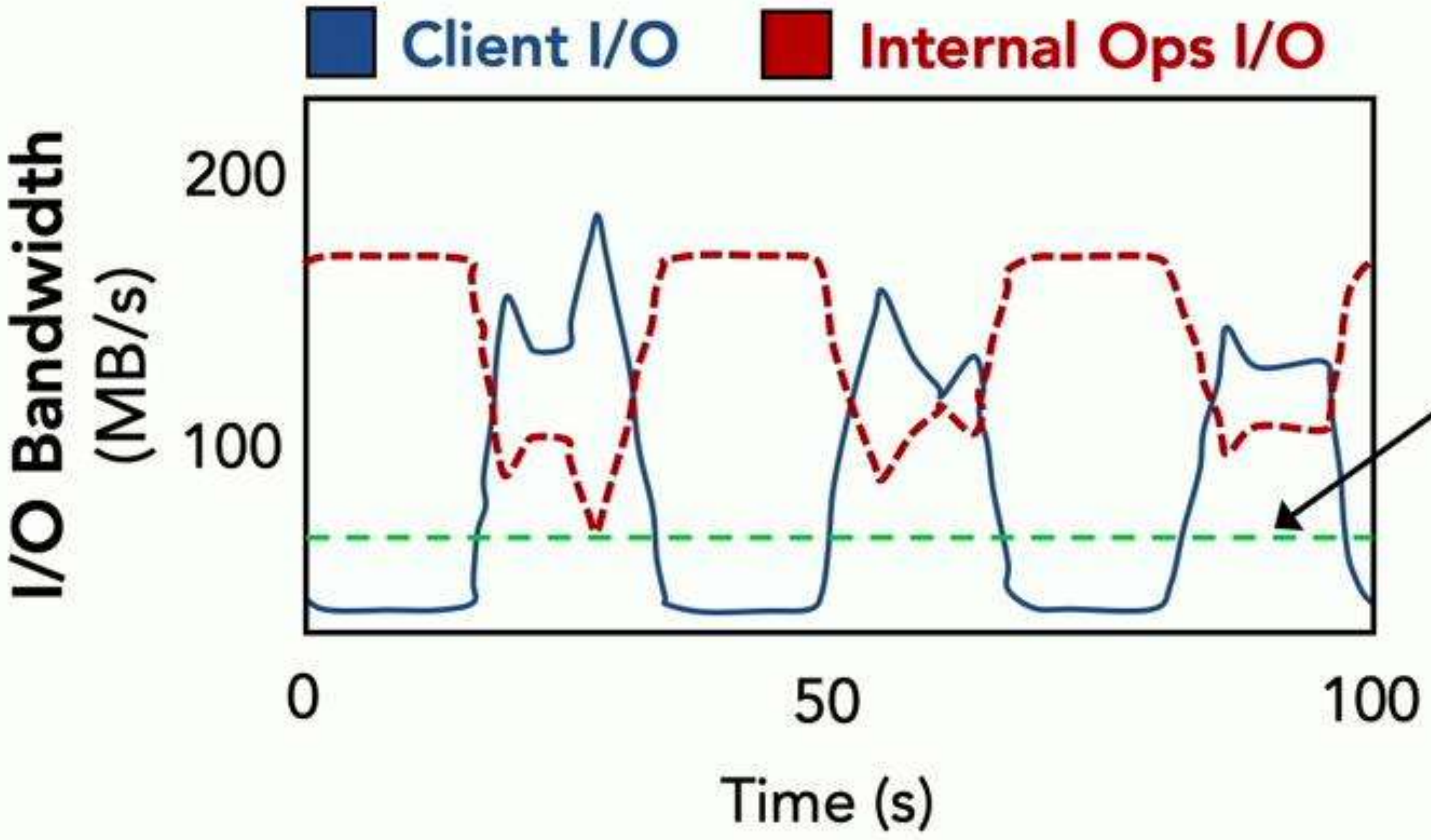
Real Nutanix client load example



More I/O to high level compactions during low load → **don't fall behind**.

Opportunistically allocate I/O for compactions

Real Nutanix client load example



More I/O to high level compactions during low load → **don't fall behind**.

Even in peak load, guarantee min I/O for flushing and L0 → L1 compaction.

SILK Evaluation

SILK Implementation

Extends RocksDB.

Open Source `https://github.com/theoanab/SILK-USENIXATC2019`

YCSB

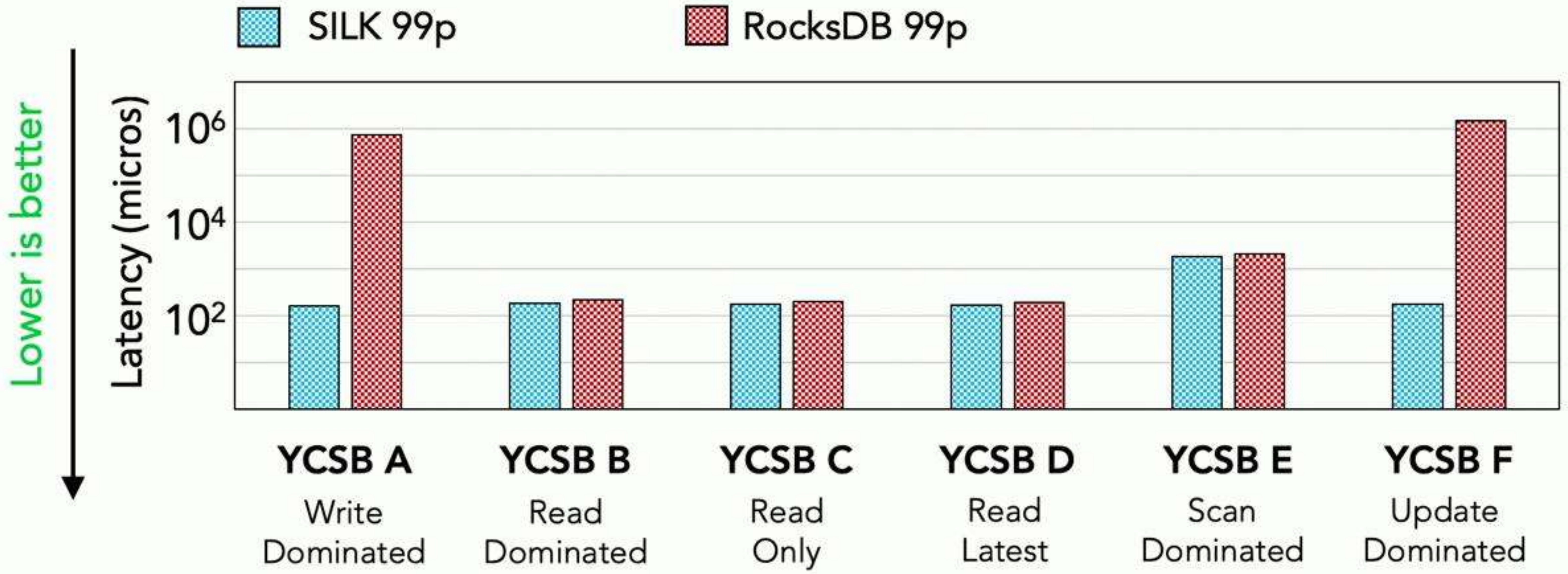
Benchmark with different workloads:

write-intensive, read-intensive, scan-intensive.

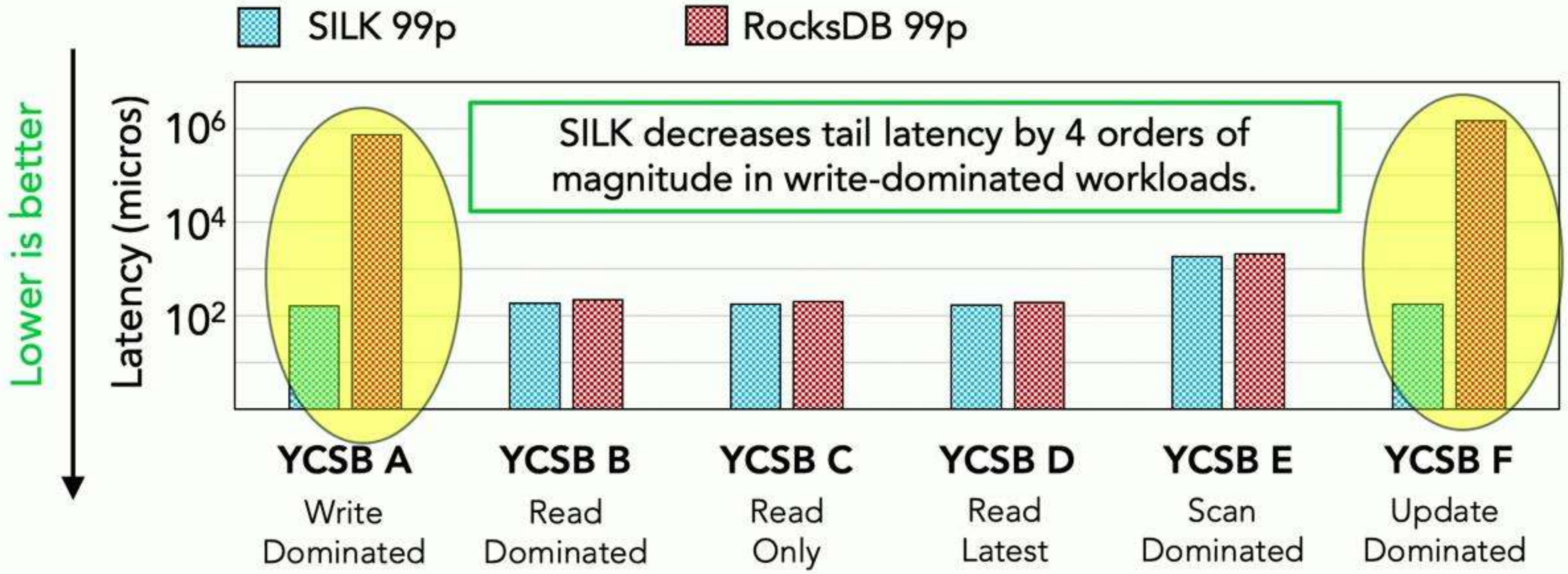
Show:

1. Write-heavy workloads: SILK is much better for tail latency.
2. Other workloads: SILK is not detrimental.

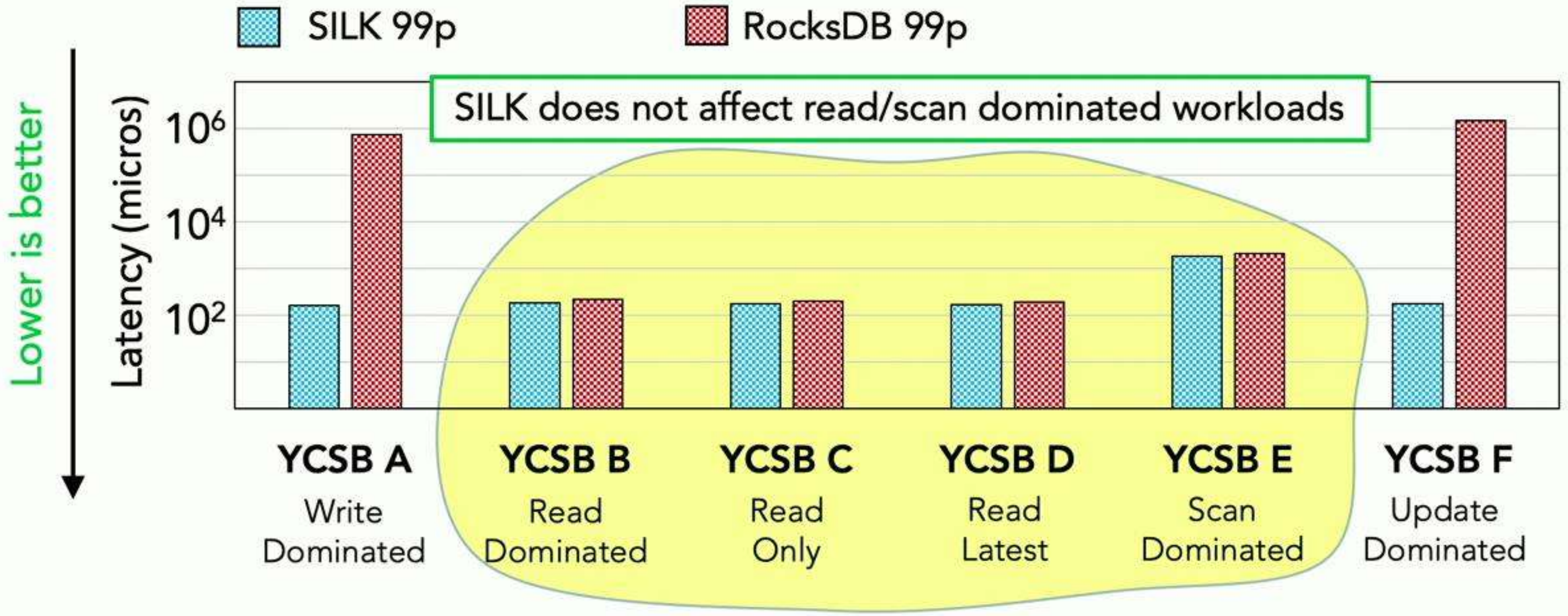
YCSB Benchmark



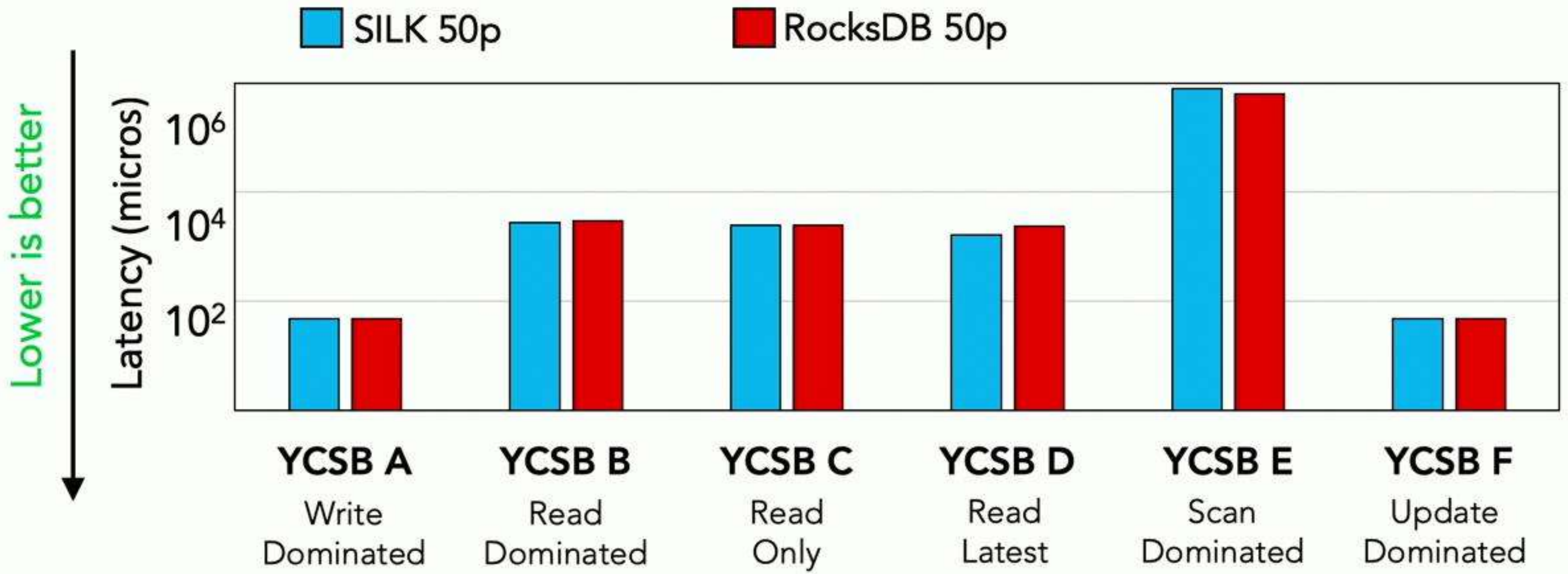
YCSB Benchmark



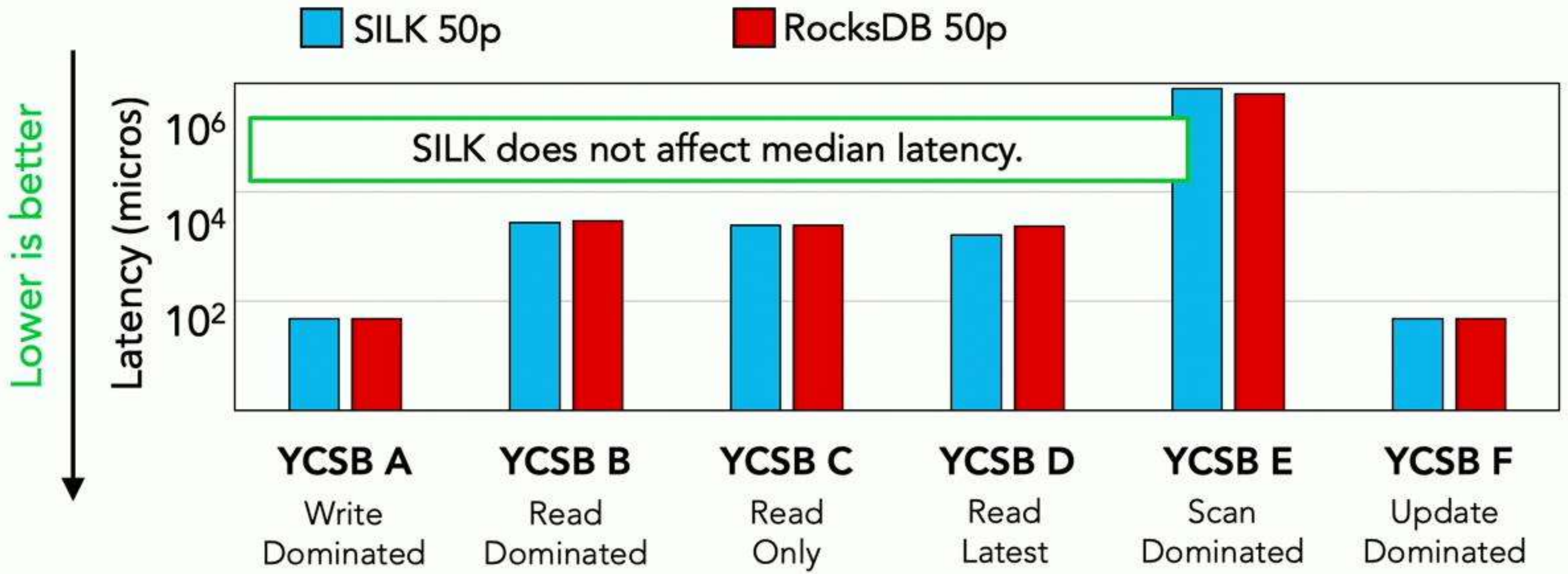
YCSB Benchmark



YCSB Benchmark Median Latency



YCSB Benchmark Median Latency



Nutanix Production Workload

Write dominated:

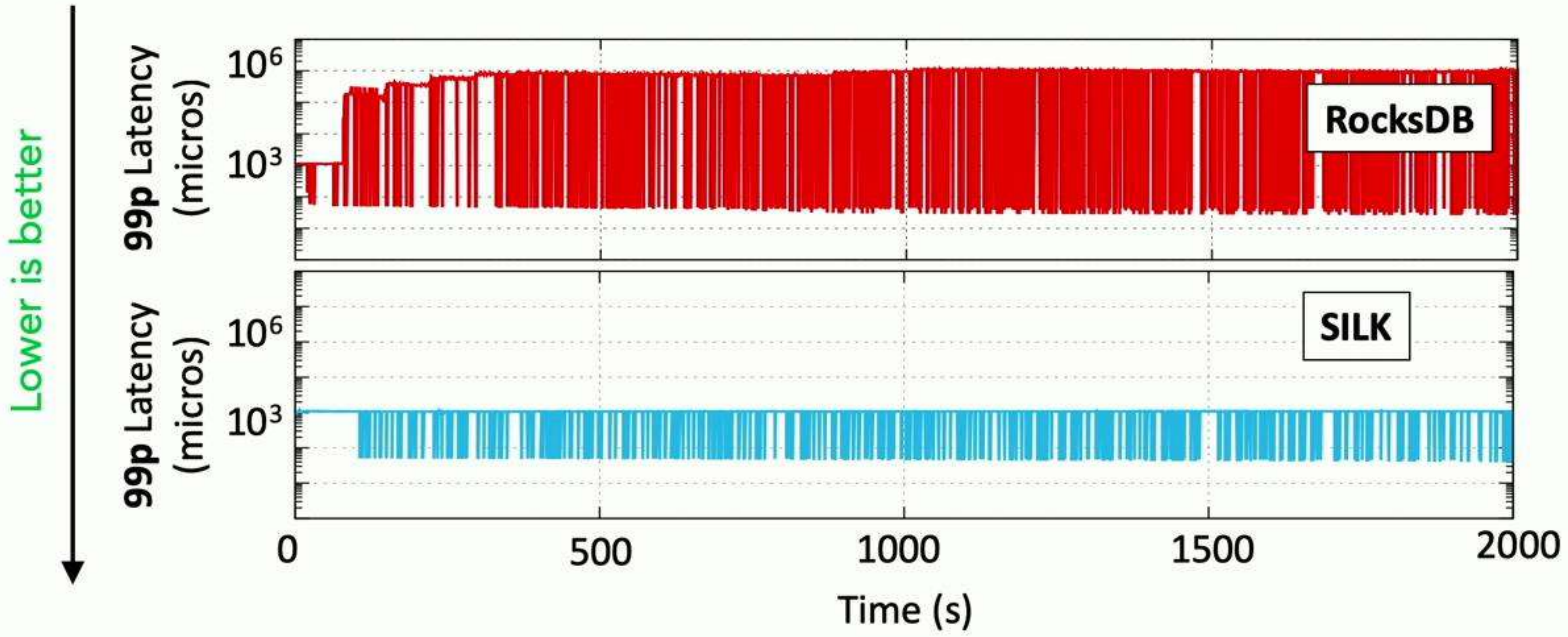
57% writes, 41% reads, 2% scans.

Bursty (open loop):

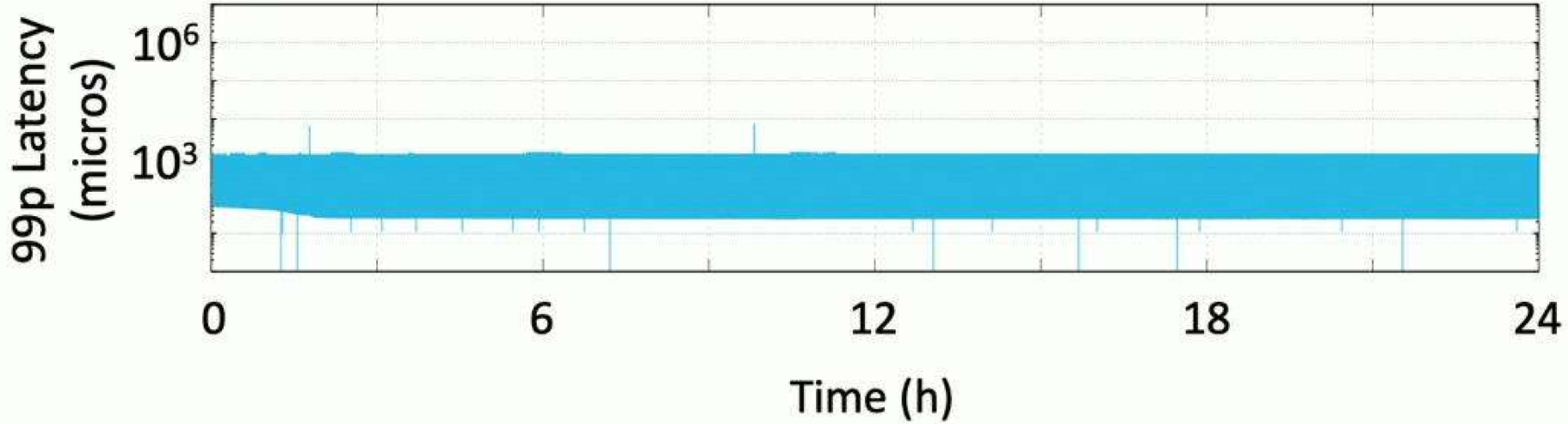
Peaks and valleys in client load.

Dataset size: 500GB, KV tuple size 400B on average.

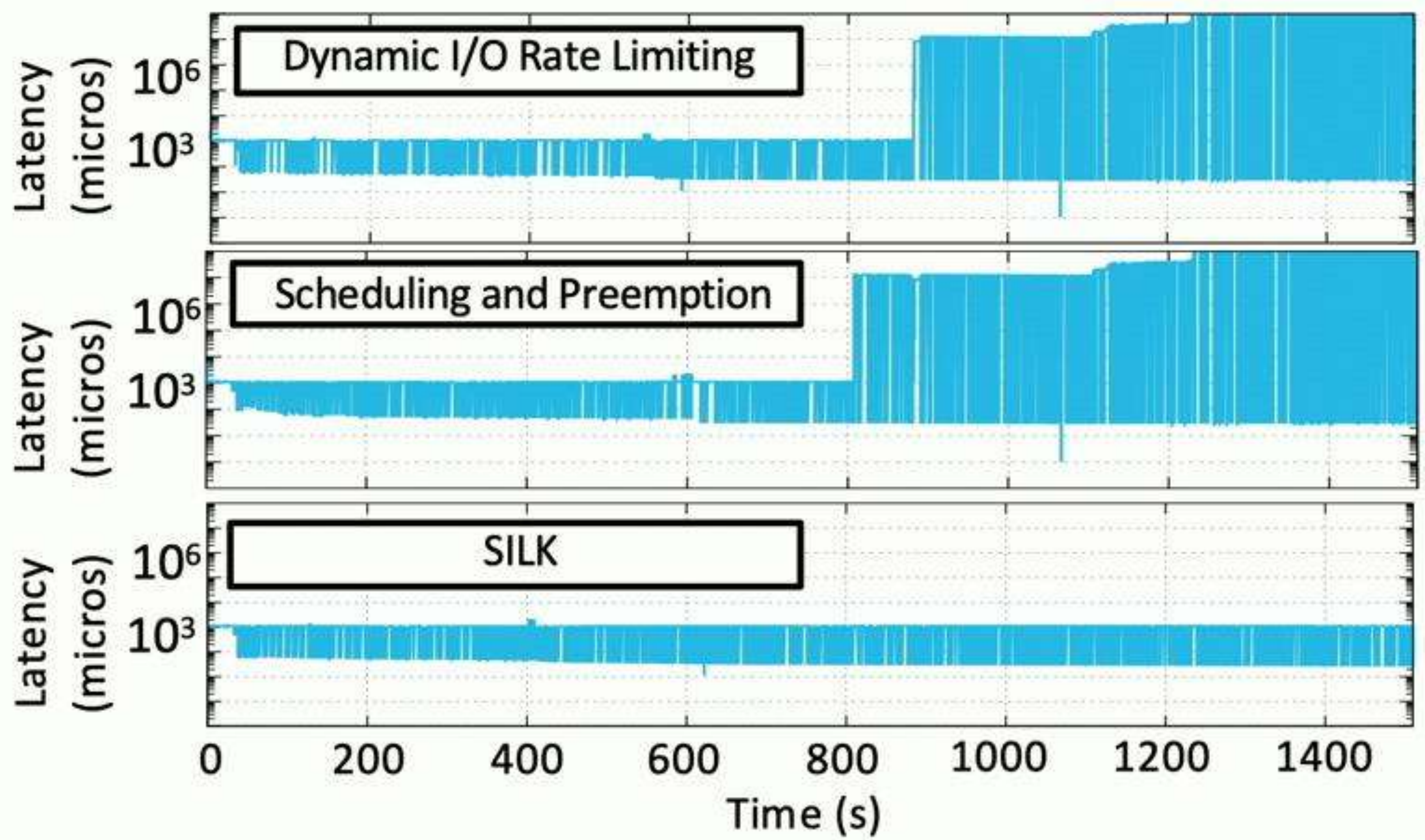
SILK vs RocksDB Tail Latency 99P



SILK for Nutanix Production Workload 24h

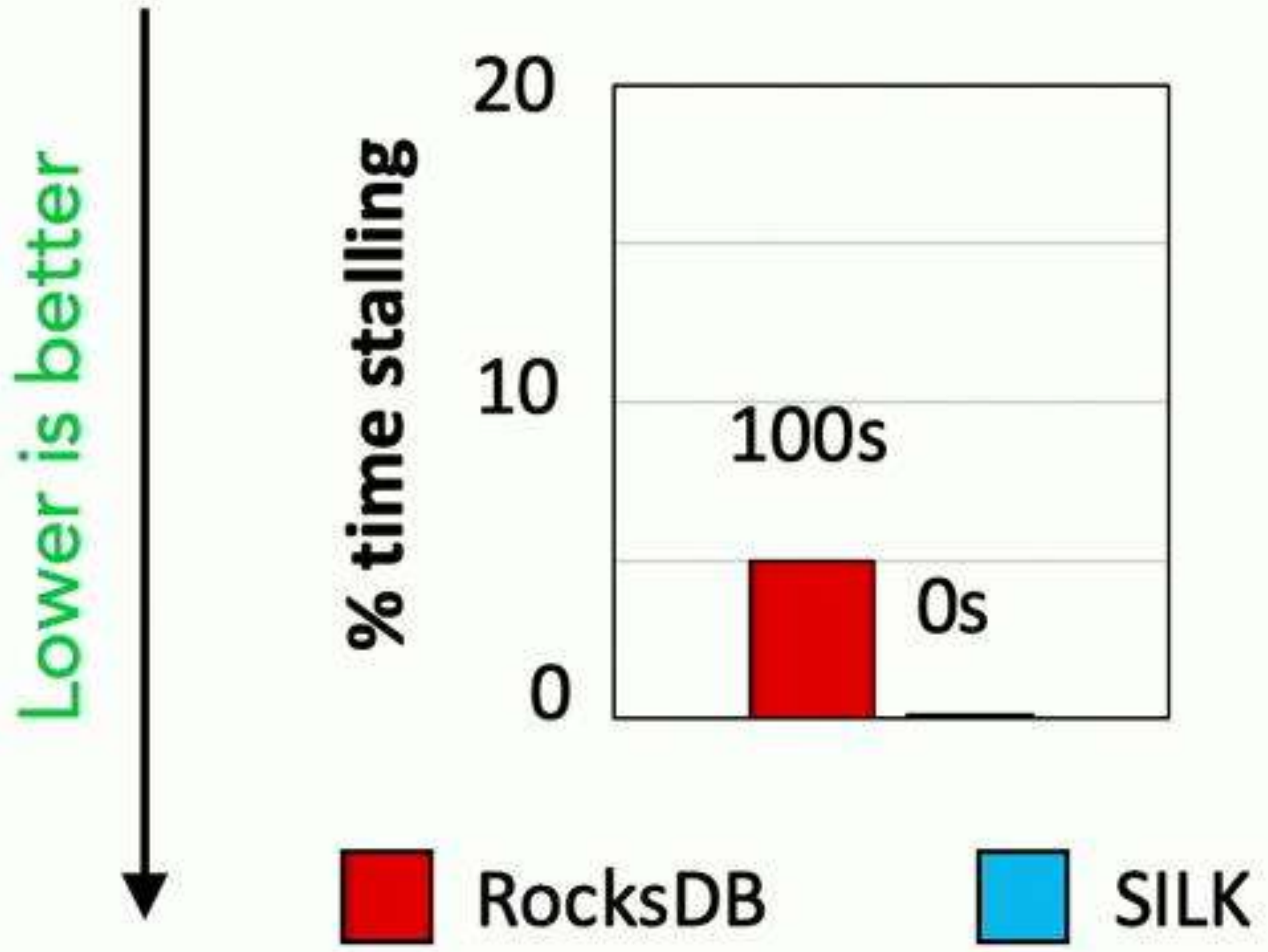


Breakdown of SILK Techniques

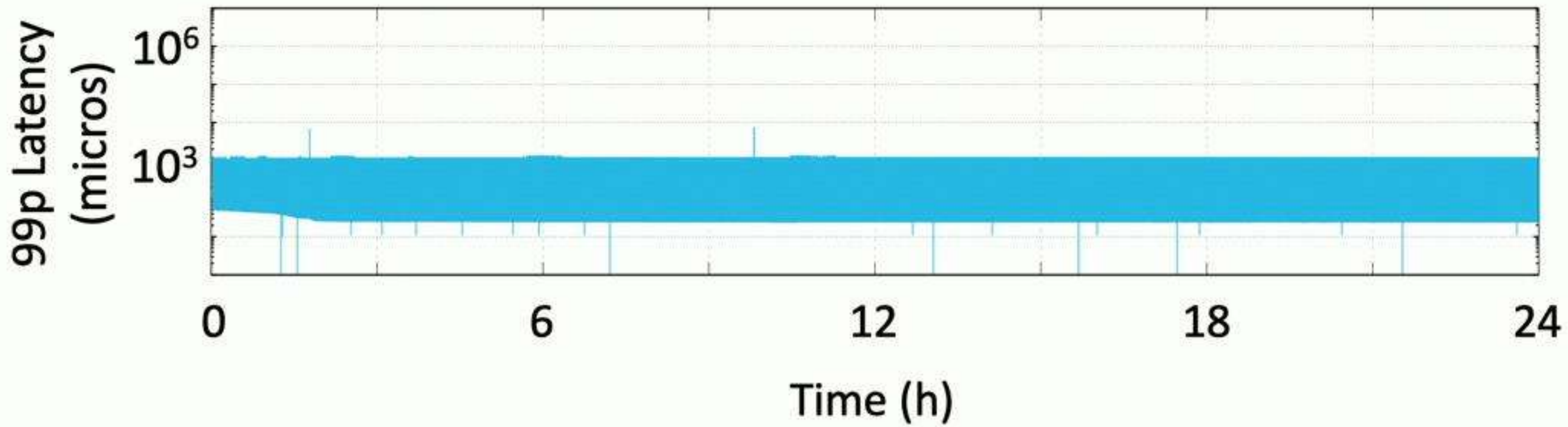


oana.balmau@sydney.edu.au

SILK vs RocksDB Stalling



SILK for Nutanix Production Workload 24h



SILK Take-Home Message

- We introduce the **new concept** of an **I/O scheduler for LSM**.
- **Coordinate I/O sharing** to avoid latency spikes.
- **Three orders-of-magnitude improvements** on tail latency.

SILK Take-Home Message

- We introduce the **new concept** of an **I/O scheduler for LSM**.
- **Coordinate I/O sharing** to avoid latency spikes.
- **Three orders-of-magnitude improvements** on tail latency.

Thank you! Questions?