TRIAD: Creating Synergies Between Memory, Disk and Log in Log Structured Key-Value Stores

USENIX ATC '17, Santa Clara CA

O. Balmau
EPFL

D. Didona
EPFL

R. Guerraoui
EPFL

W. Zwaenepoel
EPFL

H. Yuan
Nutanix

A. Arora
Nutanix

K. Gupta
Nutanix

P. Konka
Nutanix

KV Stores

Very **simple** data stores.

KV pairs.

Simple operations: **update, read.**

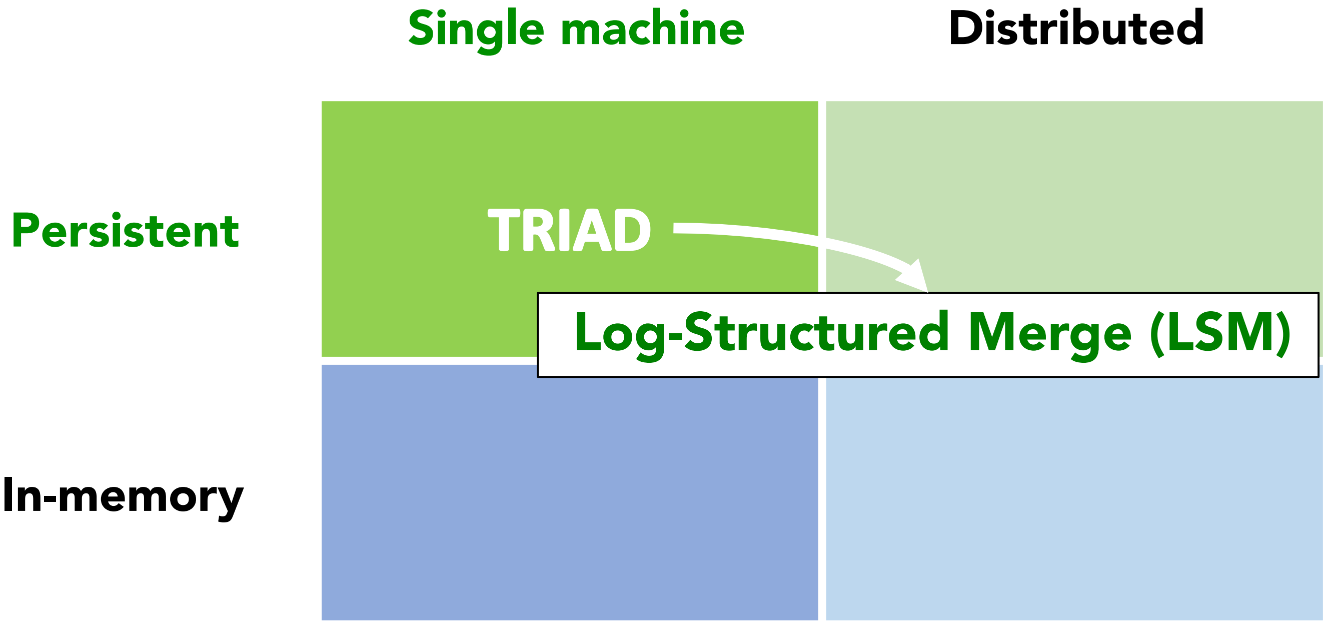
KV Stores

	Single machine	Distributed
Persistent	🟩	🟩
In-memory	🟦	🟦

KV Stores

	Single machine	Distributed
Persistent	TRIAD	
In-memory		

KV Stores



Single machine

Distributed

Persistent

In-memory

TRIAD

Log-Structured Merge (LSM)

TRIAD in a Nutshell

TRIAD LSM KV: achieves 2x throughput on production wklds.

Methods: Reducing background I/O in LSMs.

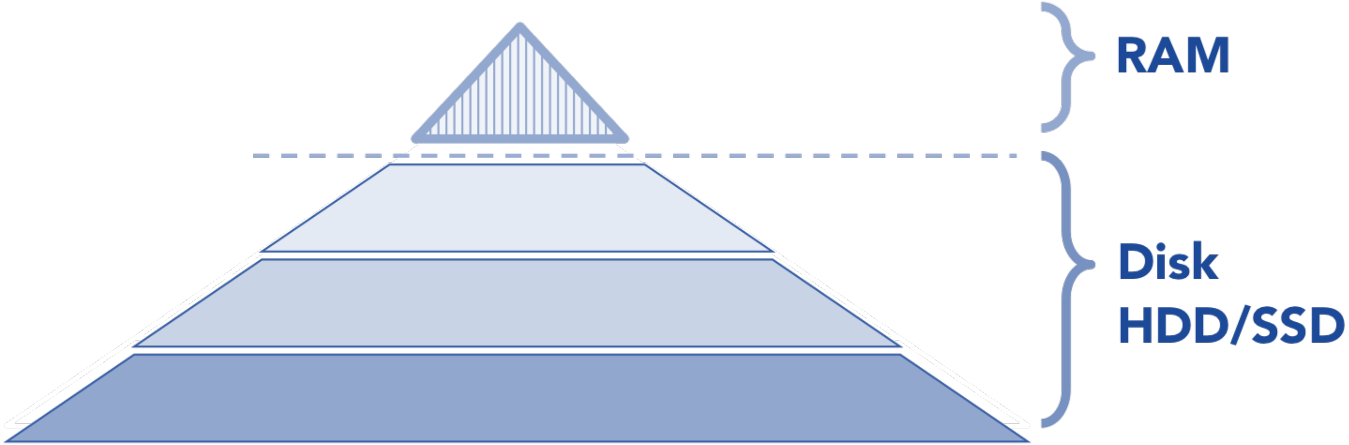
TRIAD in a Nutshell

TRIAD LSM KV: achieves 2x throughput on production wklds.

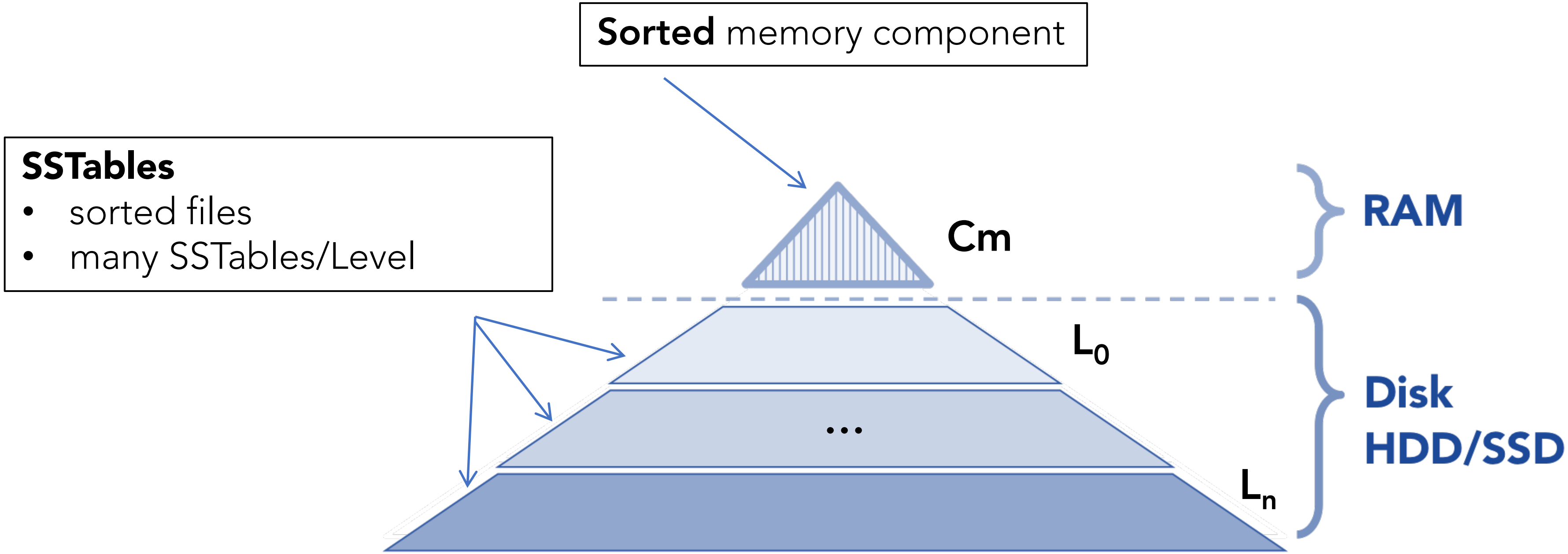
Methods: Reducing background I/O in LSMs.

- ☺ No need to know workload a priori.
- ☺ LSM KV semantics preserved.

LSM Overview



LSM Components

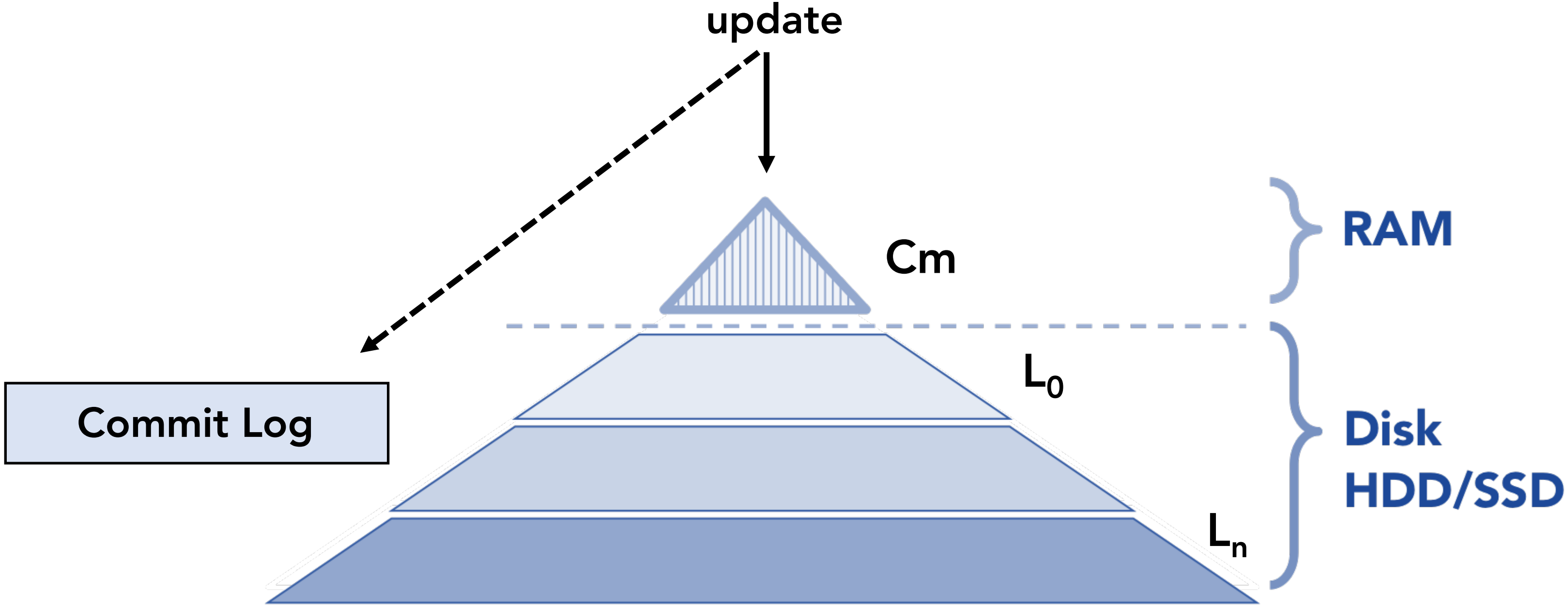


Sorted memory component

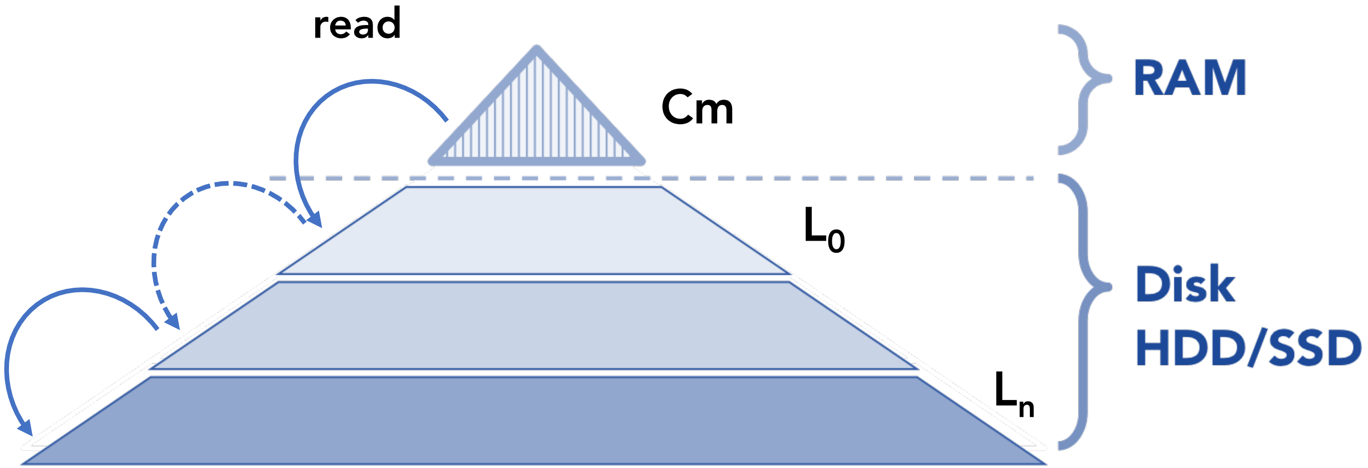
SSTables

- sorted files
- many SSTables/Level

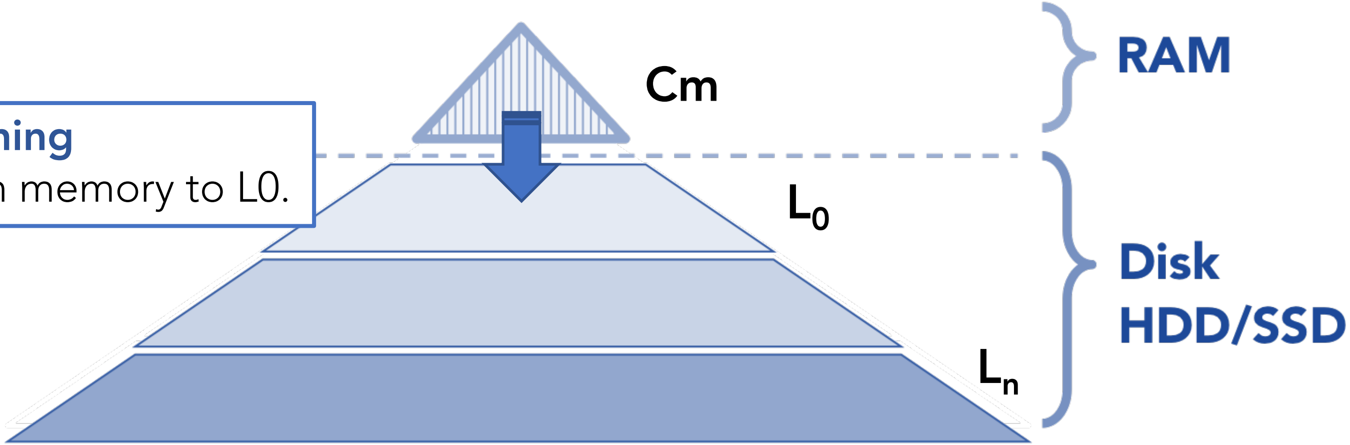
LSM Updates



LSM Reads

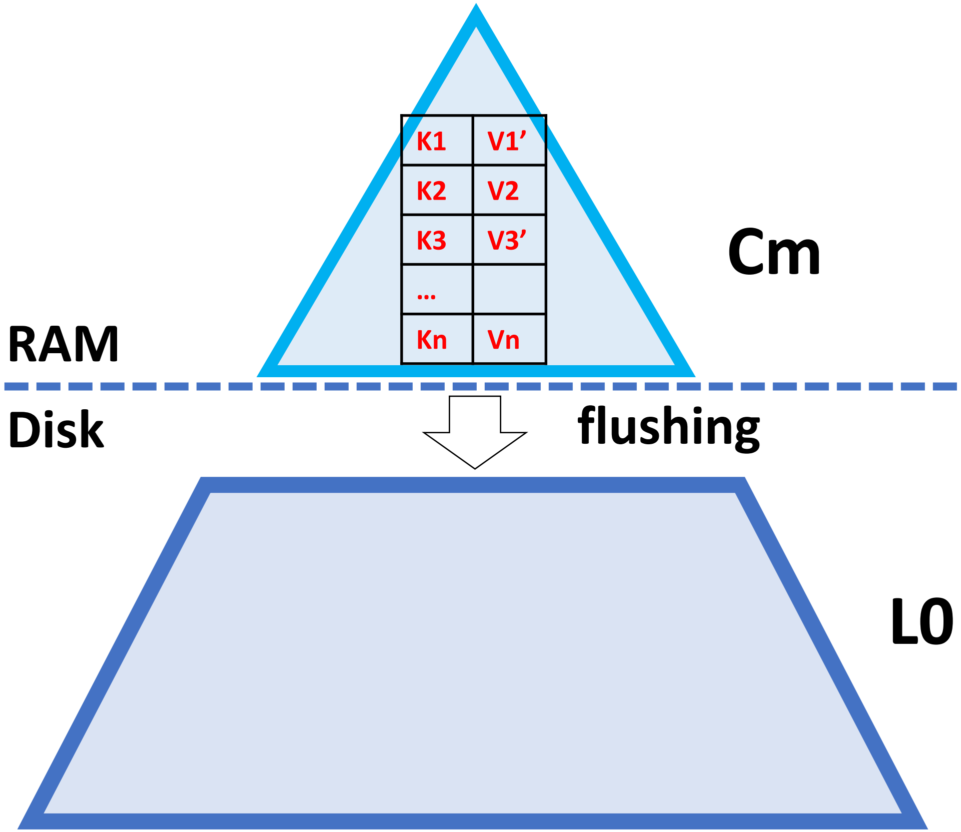


LSM Background Ops: Flushing



Flushing
From memory to L0.

LSM Background Ops: Flushing



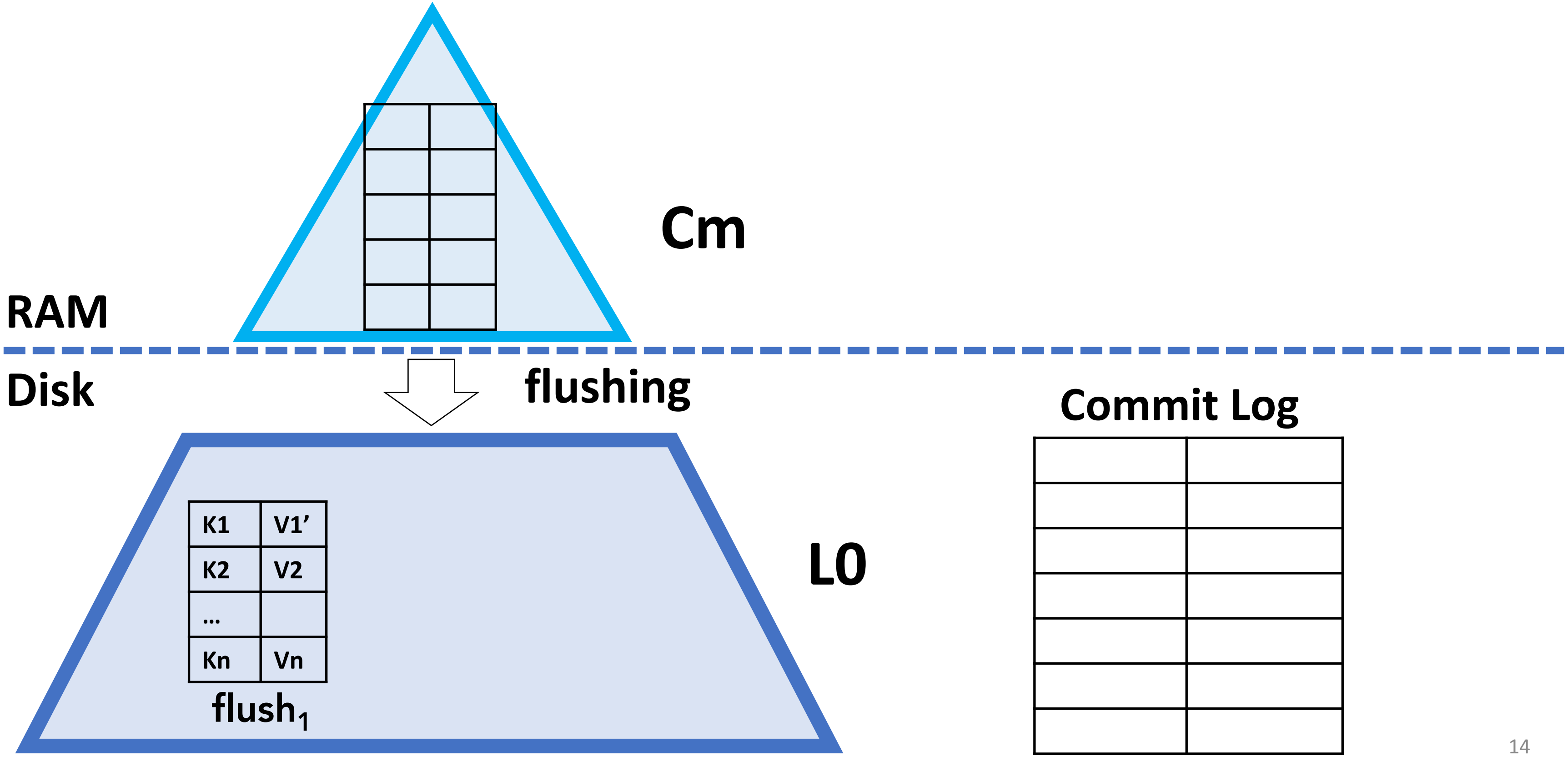
K1	V1’
K2	V2
K3	V3’
…	
Kn	Vn

Mem component full

Commit Log

K1	V1
K2	V2
K1	V1’
K3	V3

LSM Background Ops: Flushing



Cm

RAM

Disk

flushing

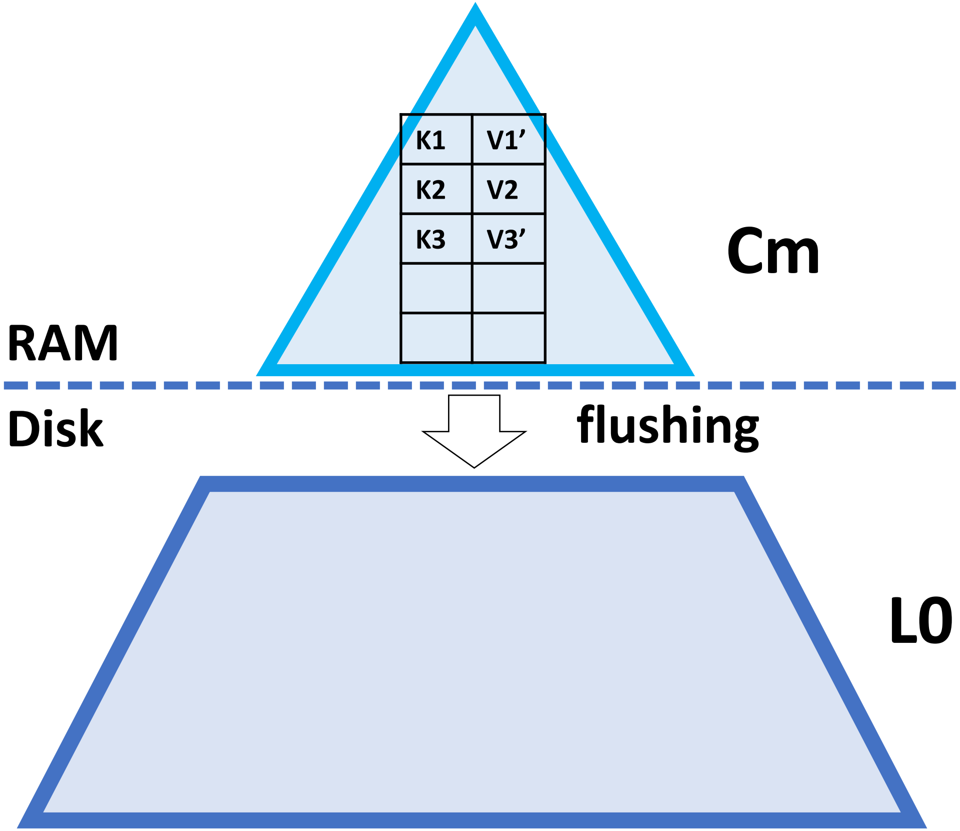
Commit Log

L0

K1	V1’
K2	V2
…	
Kn	Vn

flush_1

LSM Background Ops: Flushing



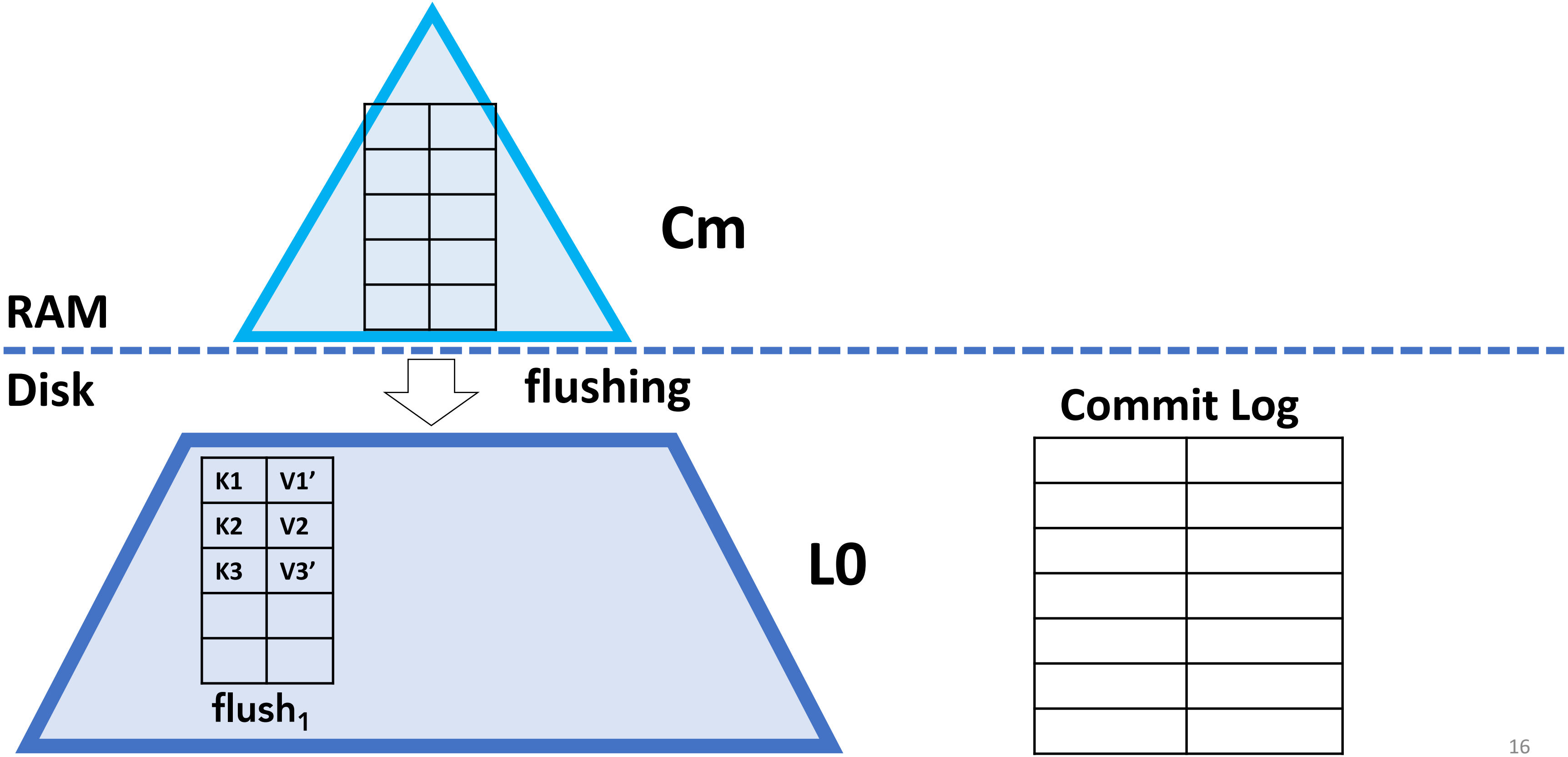
K1	V1’
K2	V2
K3	V3’

Commit log full

Commit Log

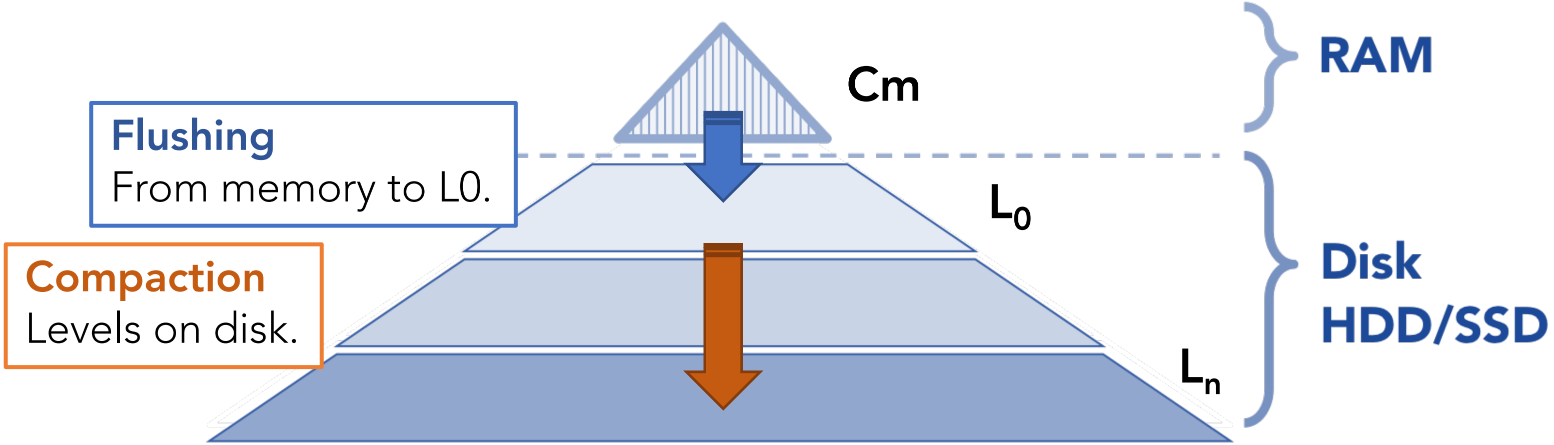
K1	V1
K2	V2
K1	V1”
K3	V3
…	
K1	V1’
K3	V3’

LSM Background Ops: Flushing



K1	V1’
K2	V2
K3	V3’

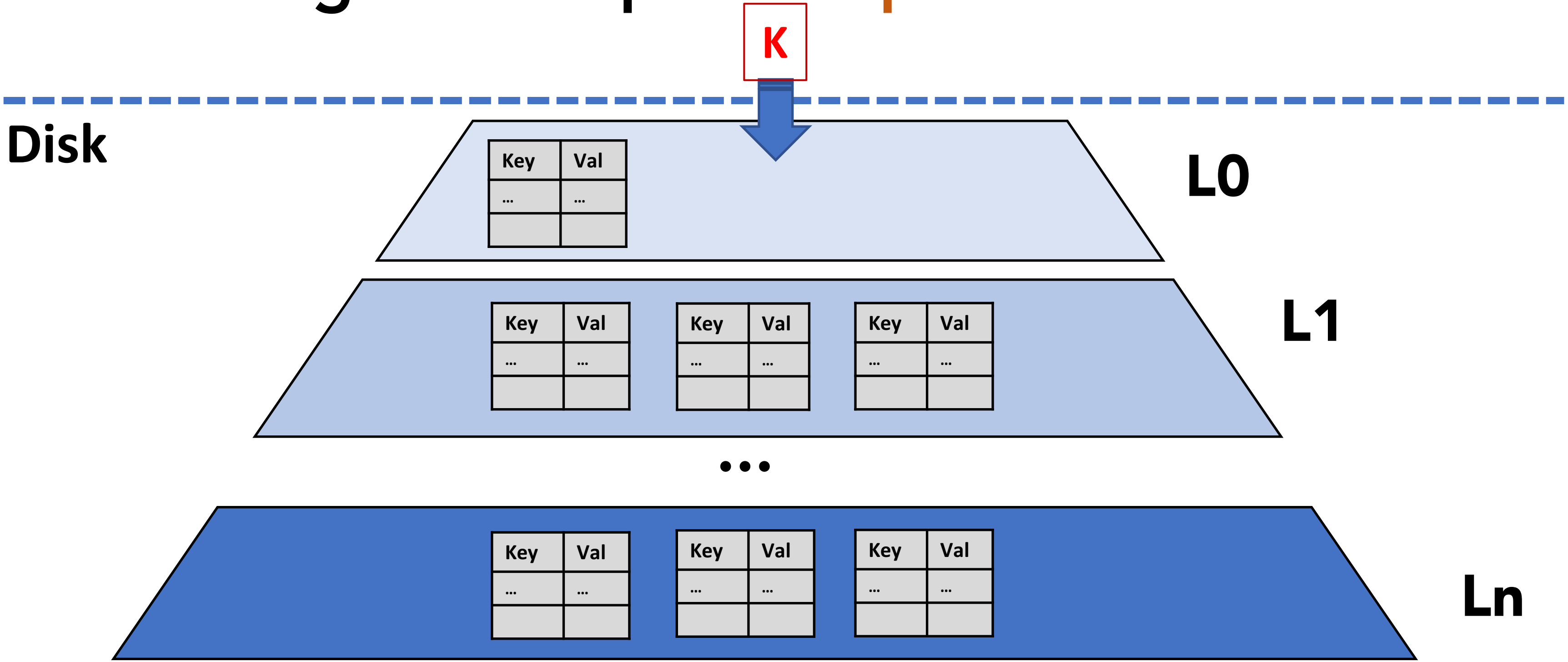
LSM Background Ops: Compaction



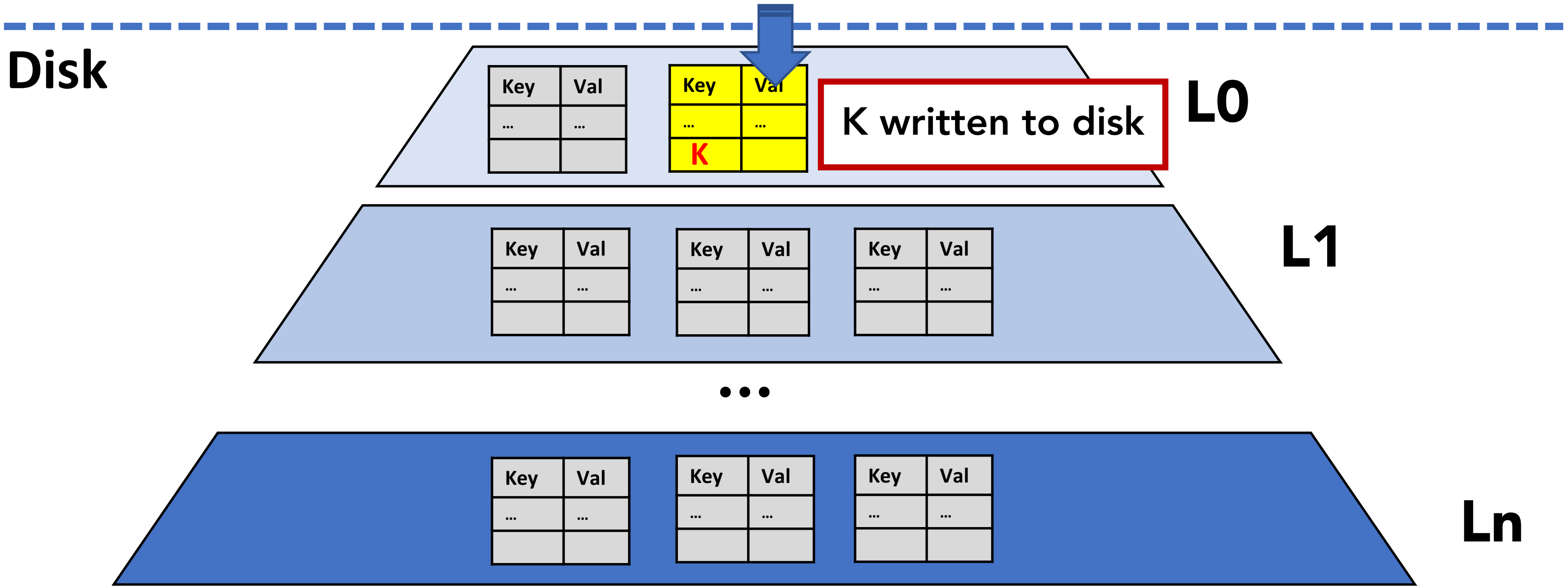
Flushing
From memory to L0.

Compaction
Levels on disk.

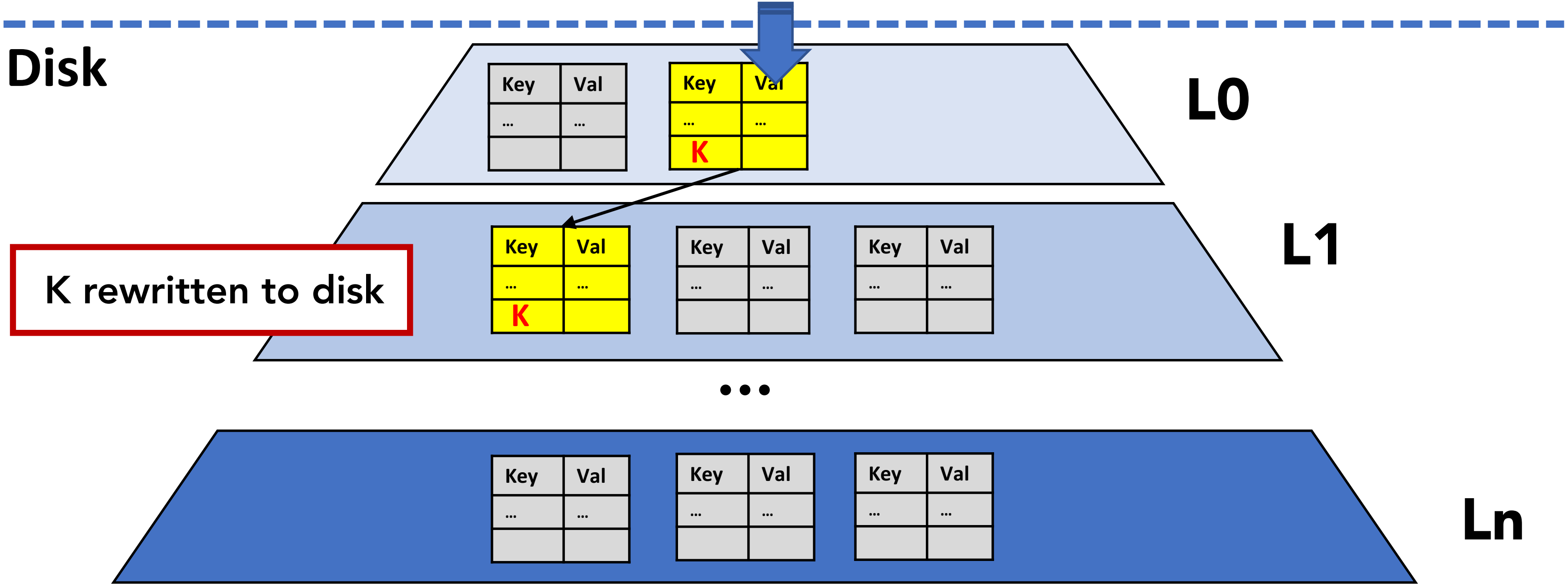
LSM Background Ops: Compaction



LSM Background Ops: Compaction

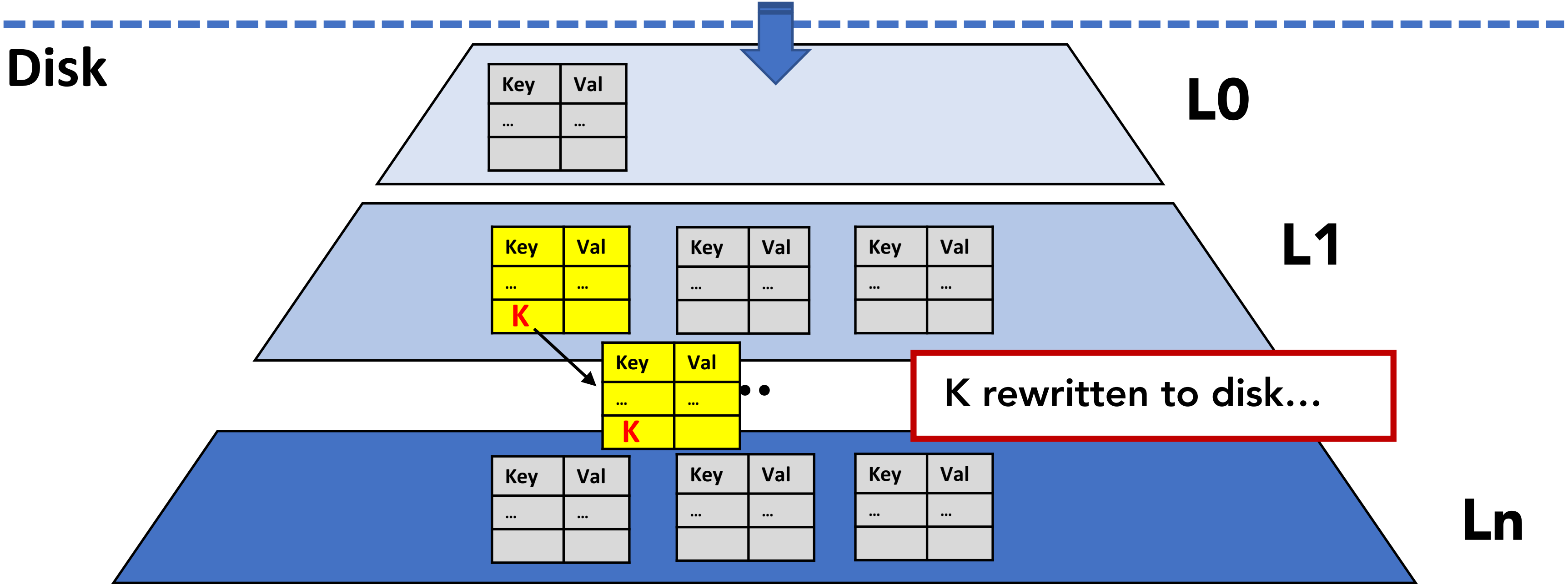


LSM Background Ops: Compaction



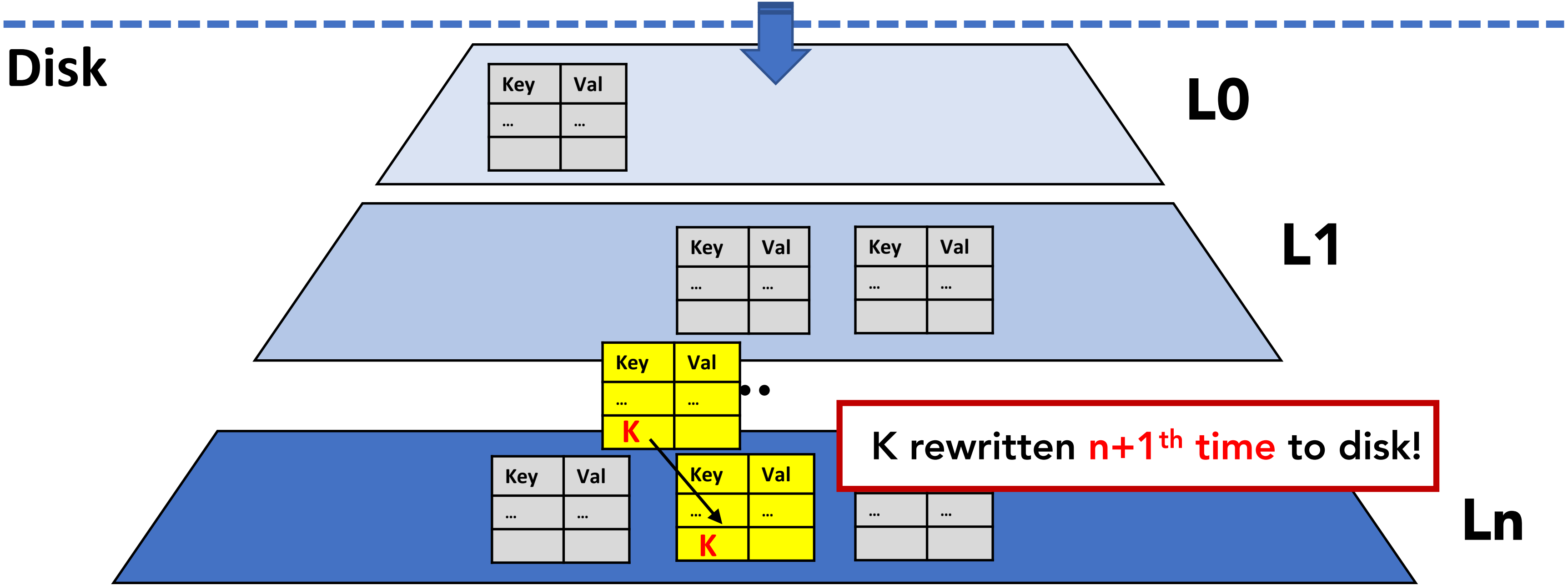
K rewritten to disk

LSM Background Ops: Compaction



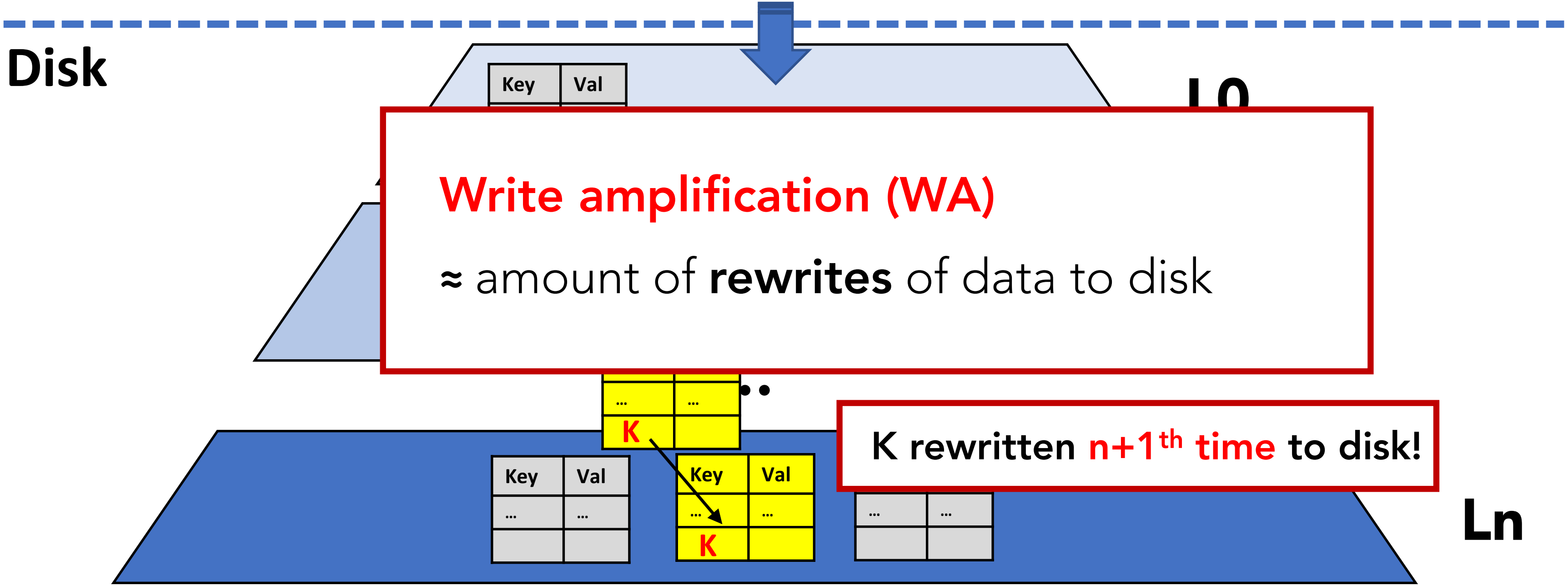
K rewritten to disk…

LSM Background Ops: Compaction



K rewritten n+1th time to disk!

LSM Background Ops: Compaction



Write amplification (WA)

≈ amount of **rewrites** of data to disk

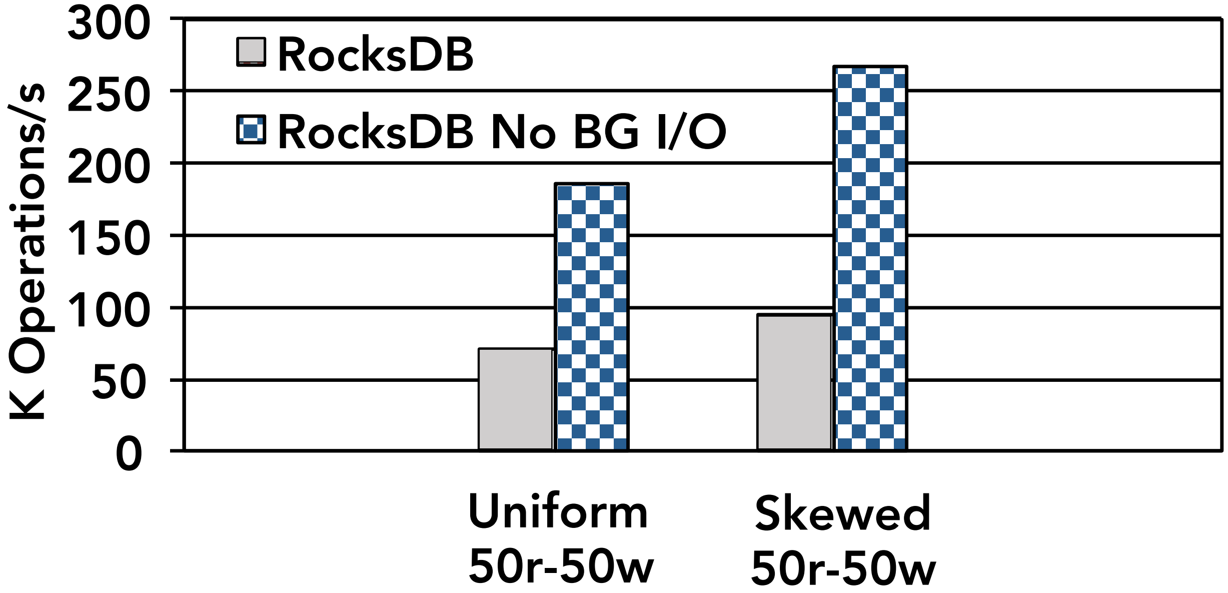
K rewritten n+1th time to disk!

Insight

Severe competition for compute/storage resources between LSM background ops and user ops.

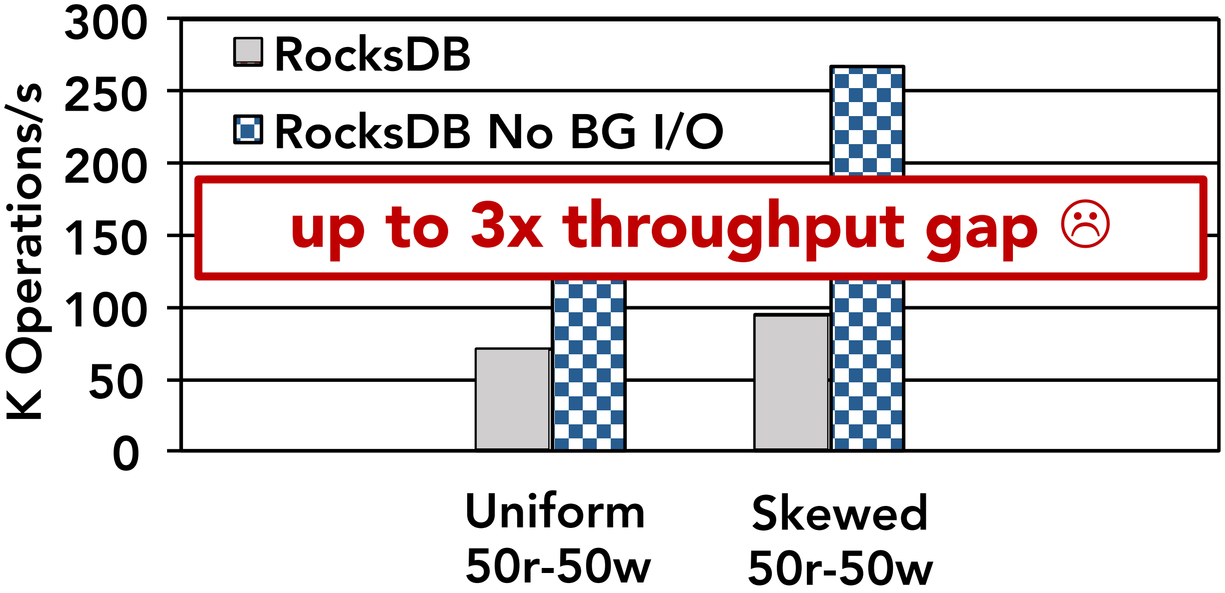
Background I/O Overhead

- Long & slow bg. ops ⟶ slowdown of user ops.



Background I/O Overhead

- **Long & slow bg. ops ⟶ slowdown of user ops.**



up to 3x throughput gap ☹

Goal

Decrease background ops overhead to increase user throughput.

TRIAD

TRIAD

	Workload	Improve WA in
TRIAD-MEM	Skewed workloads	Flushing and Compaction
TRIAD-DISK	In-between	Compaction
TRIAD-LOG	Uniform workloads	Flushing

Three techniques work together and are complementary.

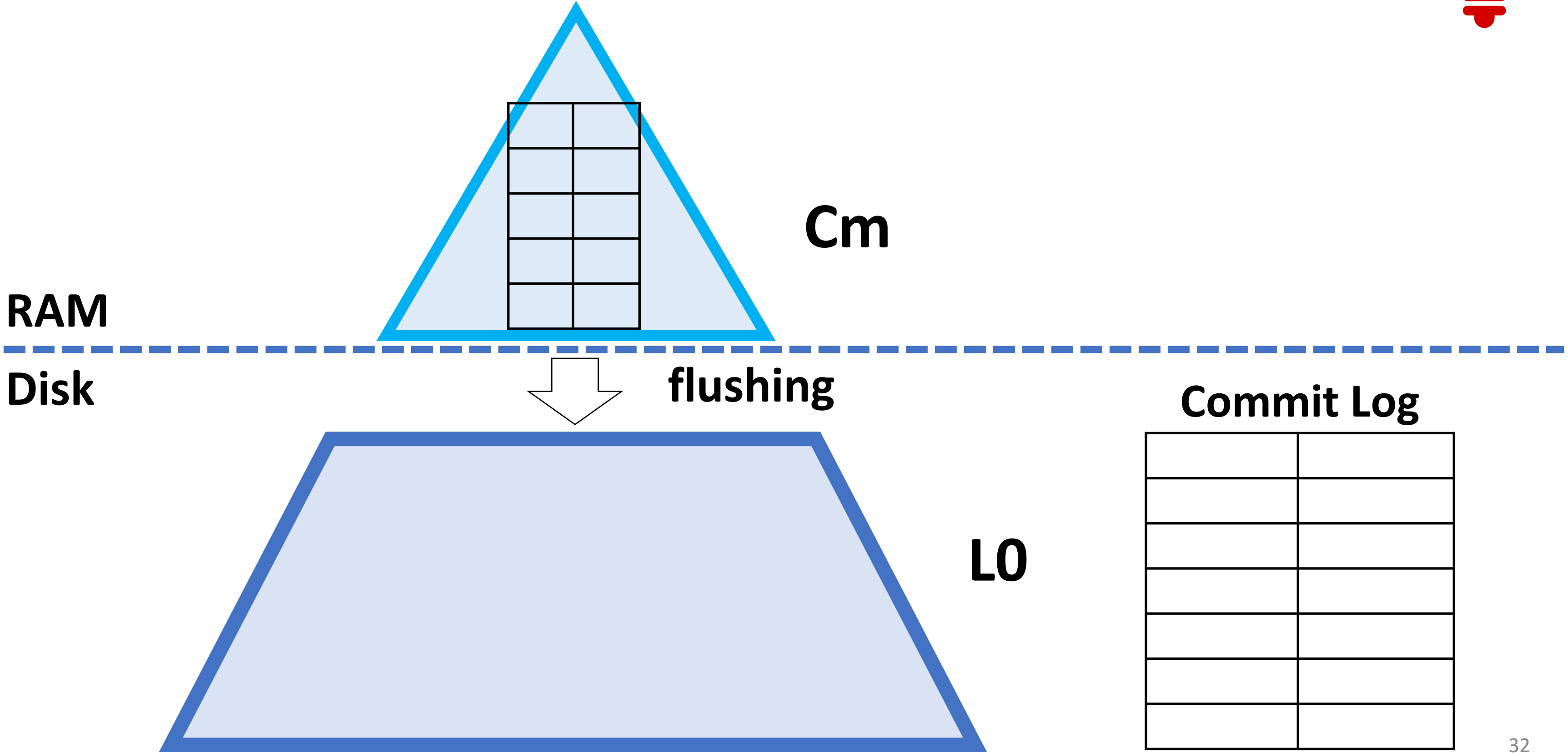
TRIAD

	Workload	Improve WA in
TRIAD-MEM	Skewed workloads	Flushing and Compaction
TRIAD-LOG	Uniform workloads	Flushing

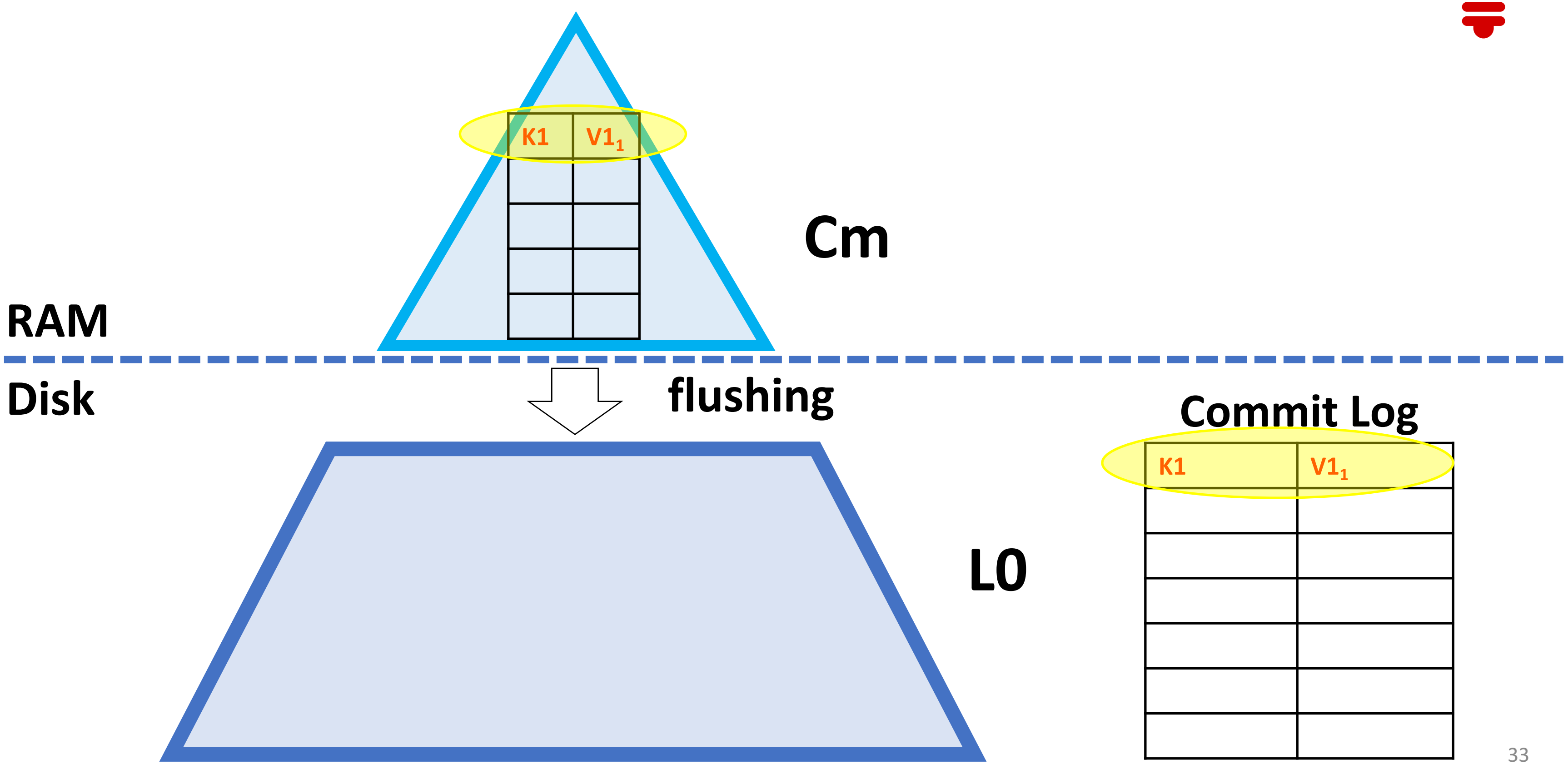
TRIAD-MEM

	Workload	Improve WA in
TRIAD-MEM	Skewed workloads	Flushing and Compaction

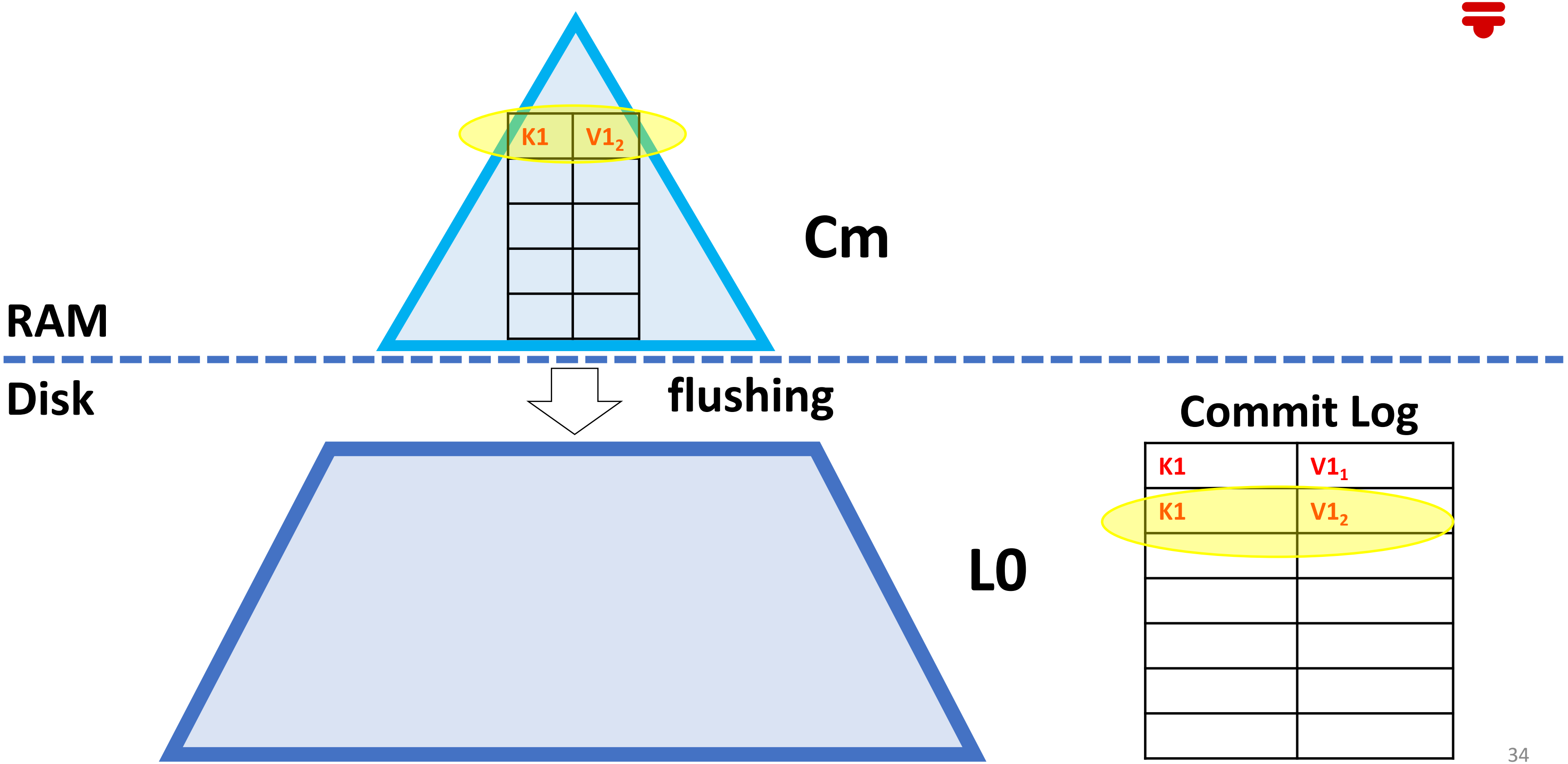
Problem: Flushing with Skewed Workloads



Problem: Flushing with Skewed Workloads



Problem: Flushing with Skewed Workloads

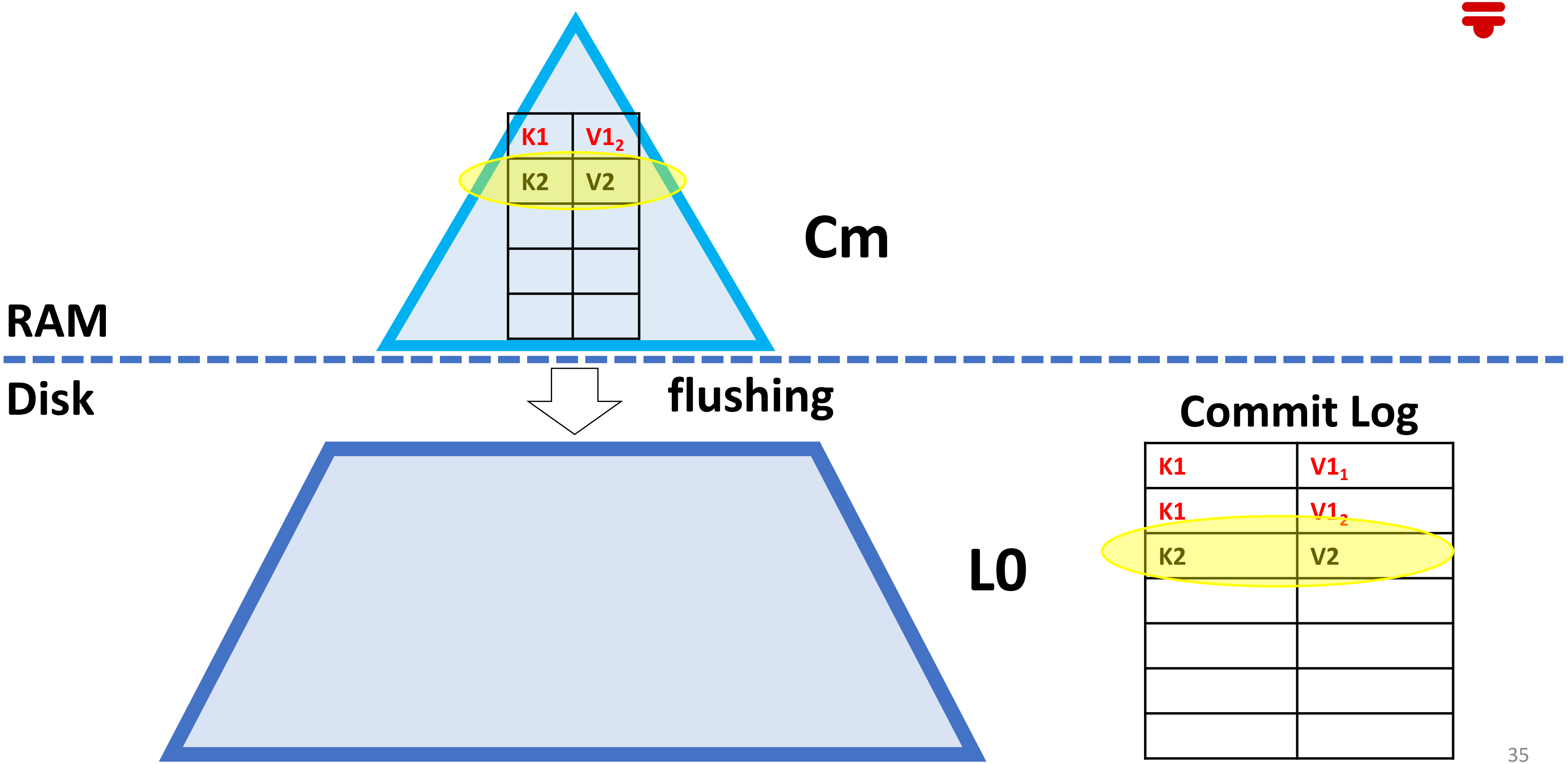


K1	$V1_2$

Commit Log

K1	$V1_1$
K1	$V1_2$

Problem: Flushing with Skewed Workloads

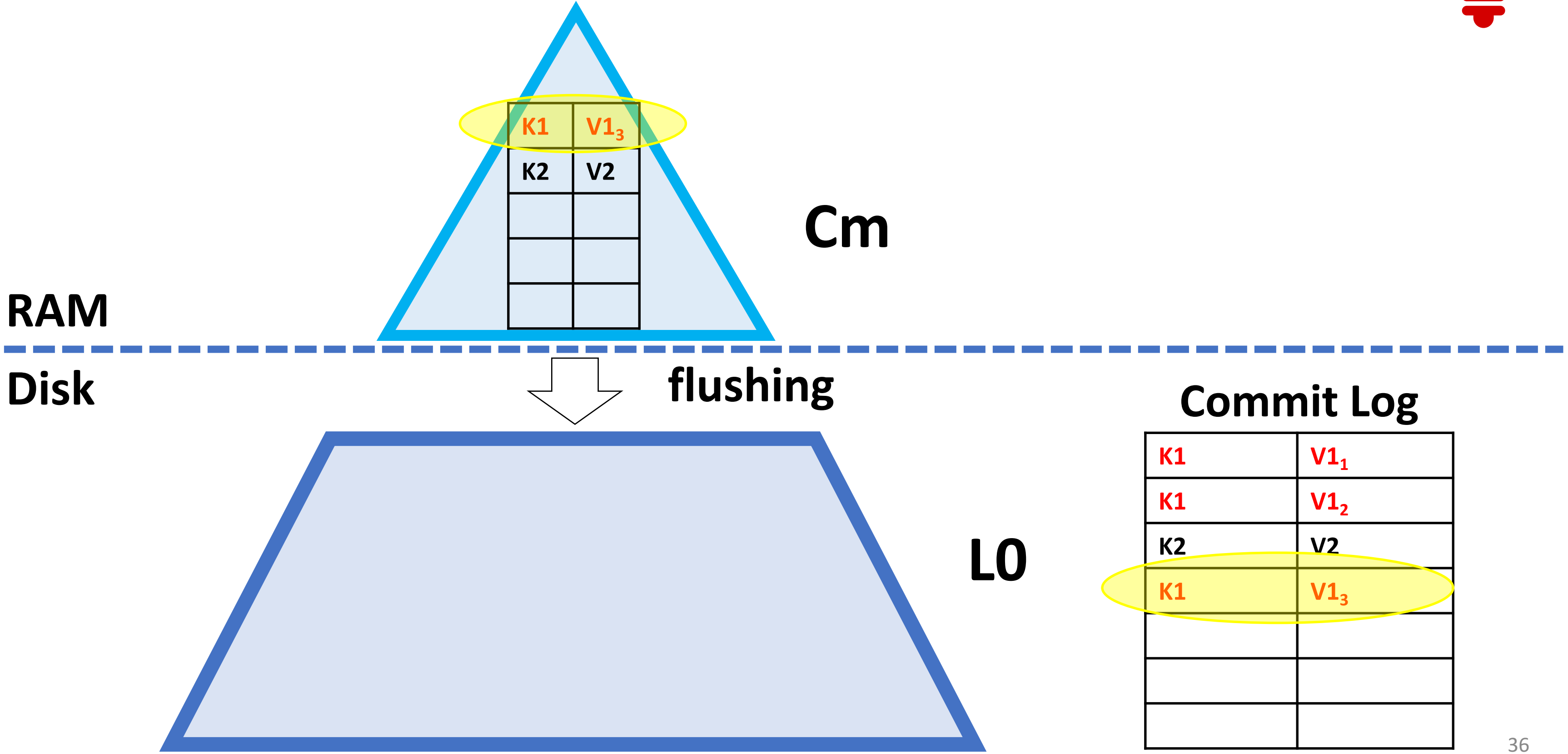


K1	$V1_2$
K2	V2

Commit Log

K1	$V1_1$
K1	$V1_2$
K2	V2

Problem: Flushing with Skewed Workloads

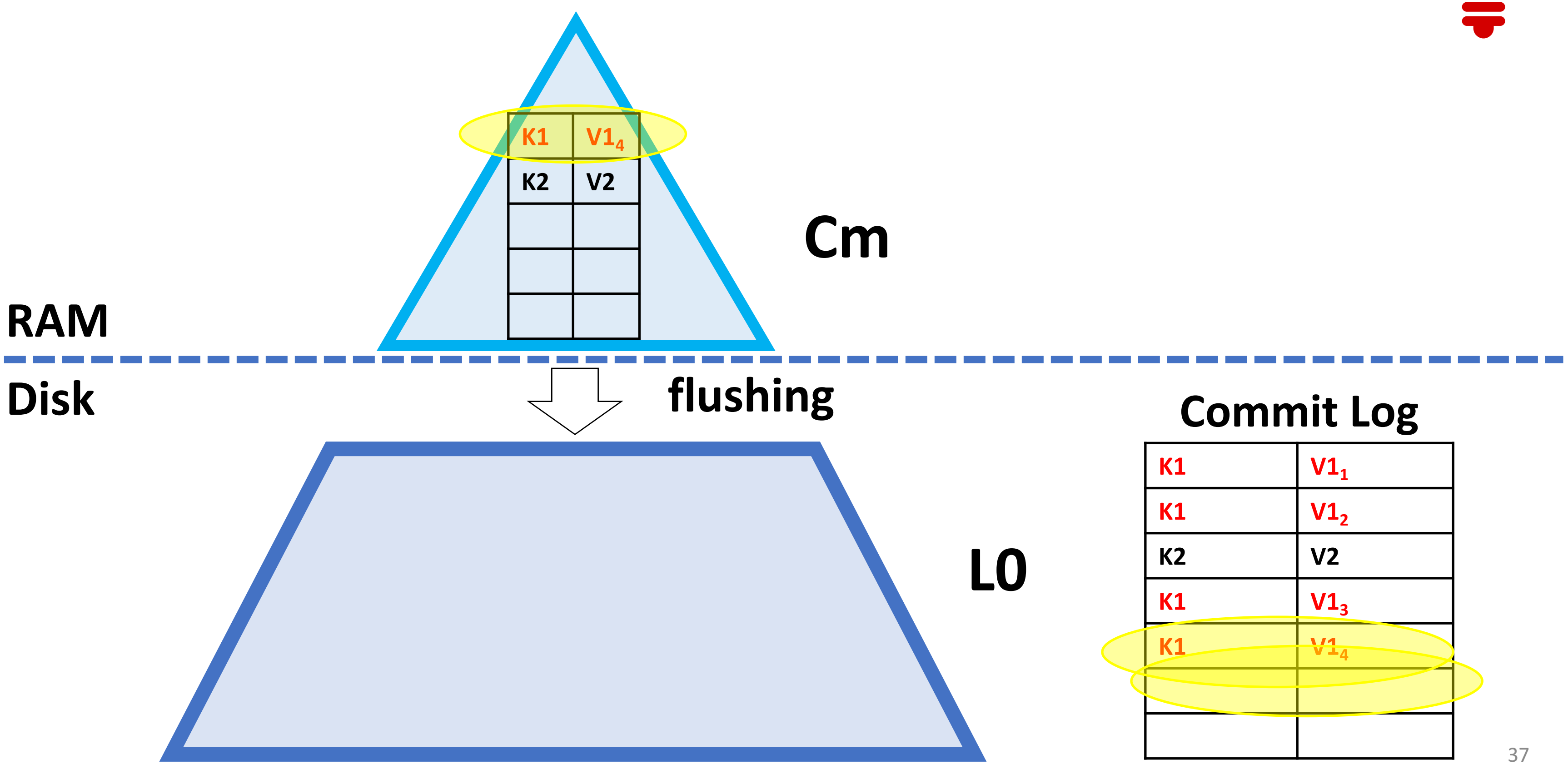


K1	$V1_3$
K2	V2

Commit Log

K1	$V1_1$
K1	$V1_2$
K2	V2
K1	$V1_3$

Problem: Flushing with Skewed Workloads

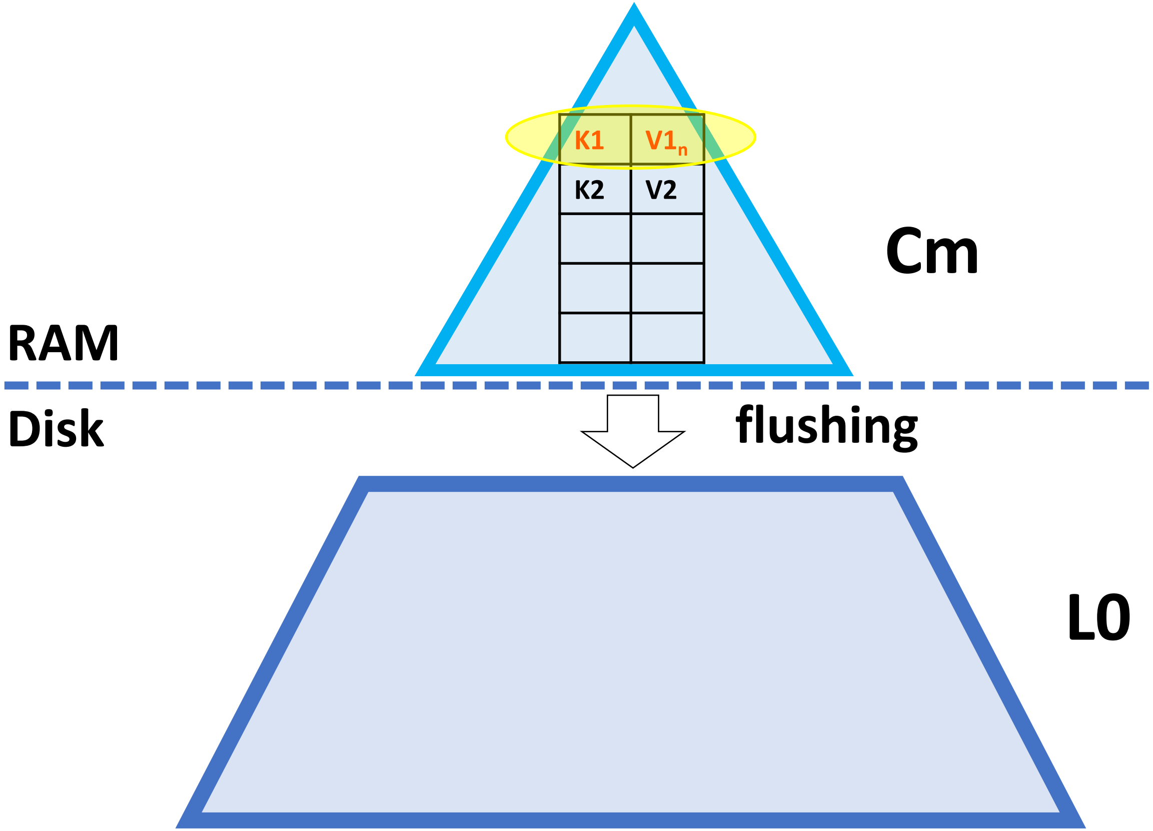


K1	$V1_4$
K2	V2

Commit Log

K1	$V1_1$
K1	$V1_2$
K2	V2
K1	$V1_3$
K1	$V1_4$

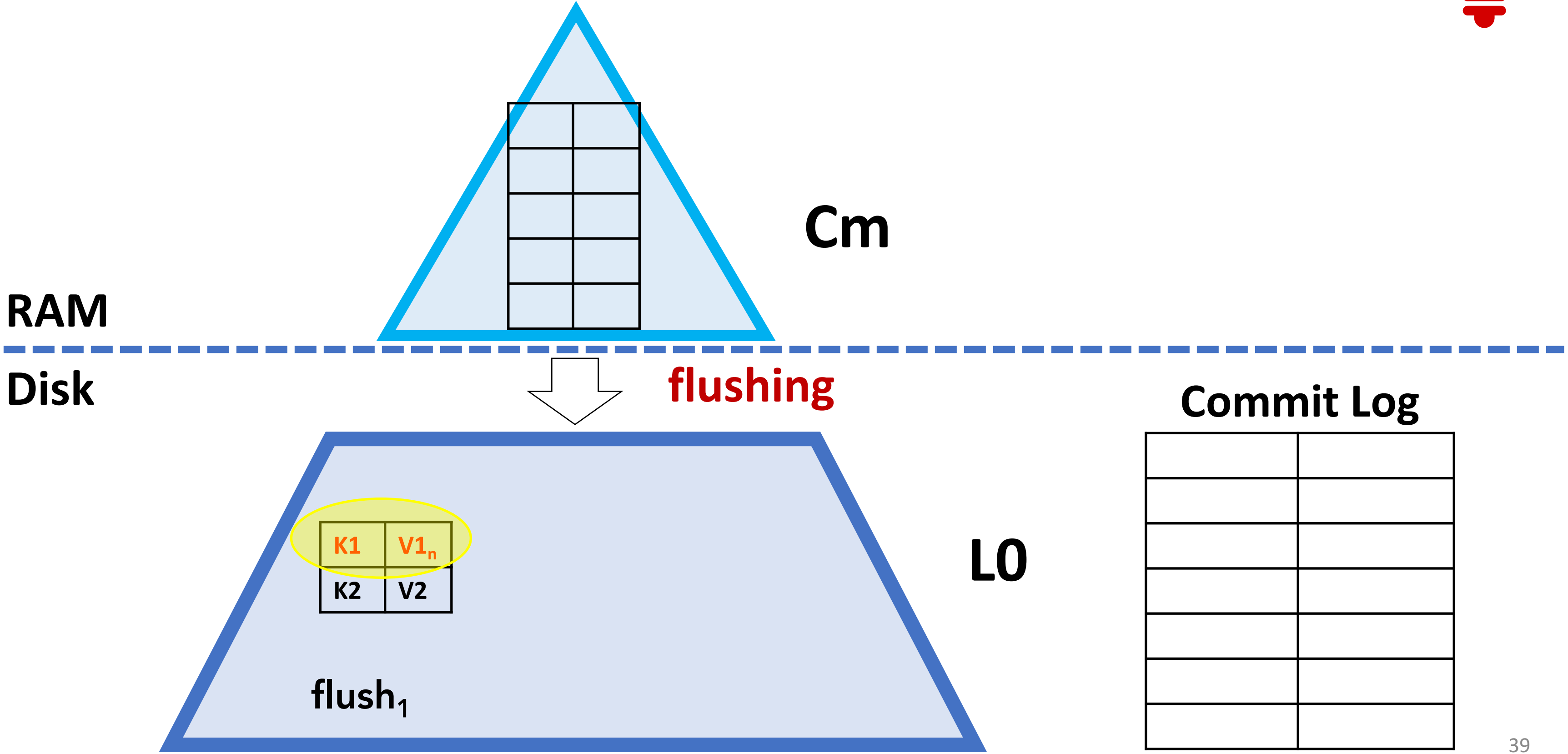
Problem: Flushing with Skewed Workloads



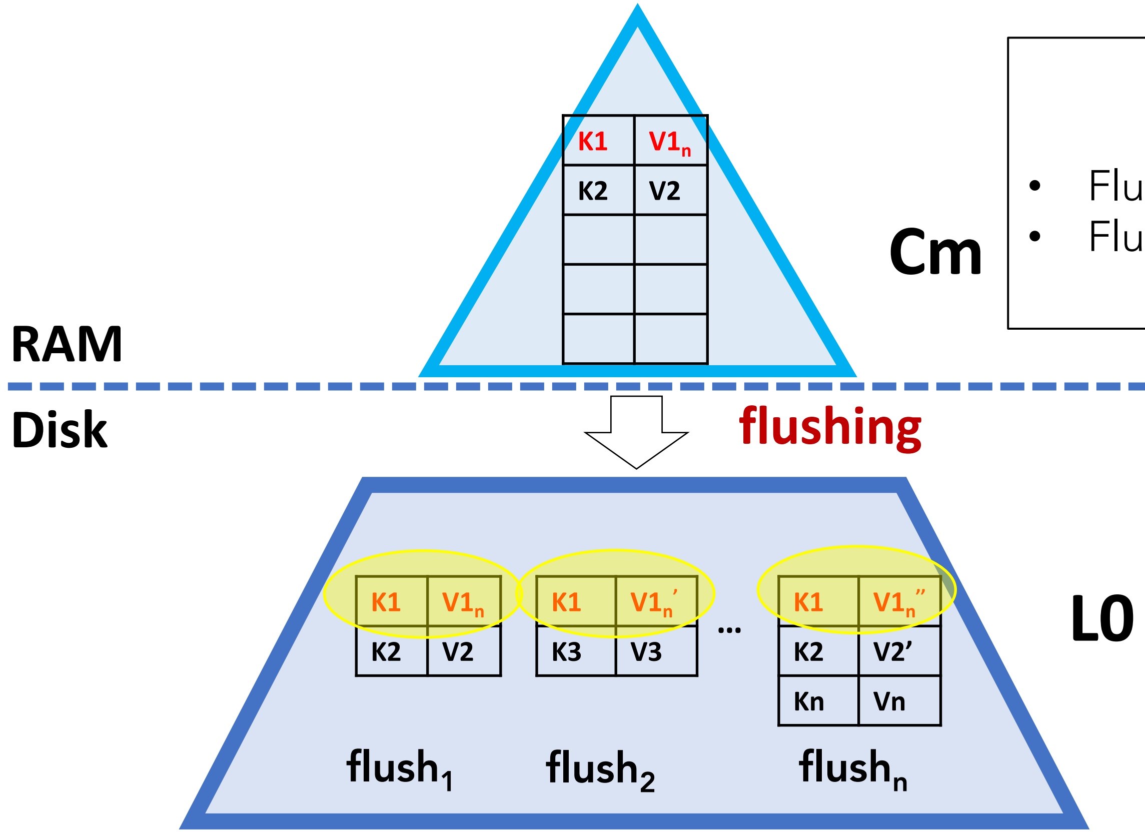
Commit Log

K1	$V1_1$
K1	$V1_2$
K2	V2
K1	$V1_3$
K1	$V1_4$
…	
K1	$V1_n$

Problem: Flushing with Skewed Workloads



Problem: Flushing with Skewed Workloads



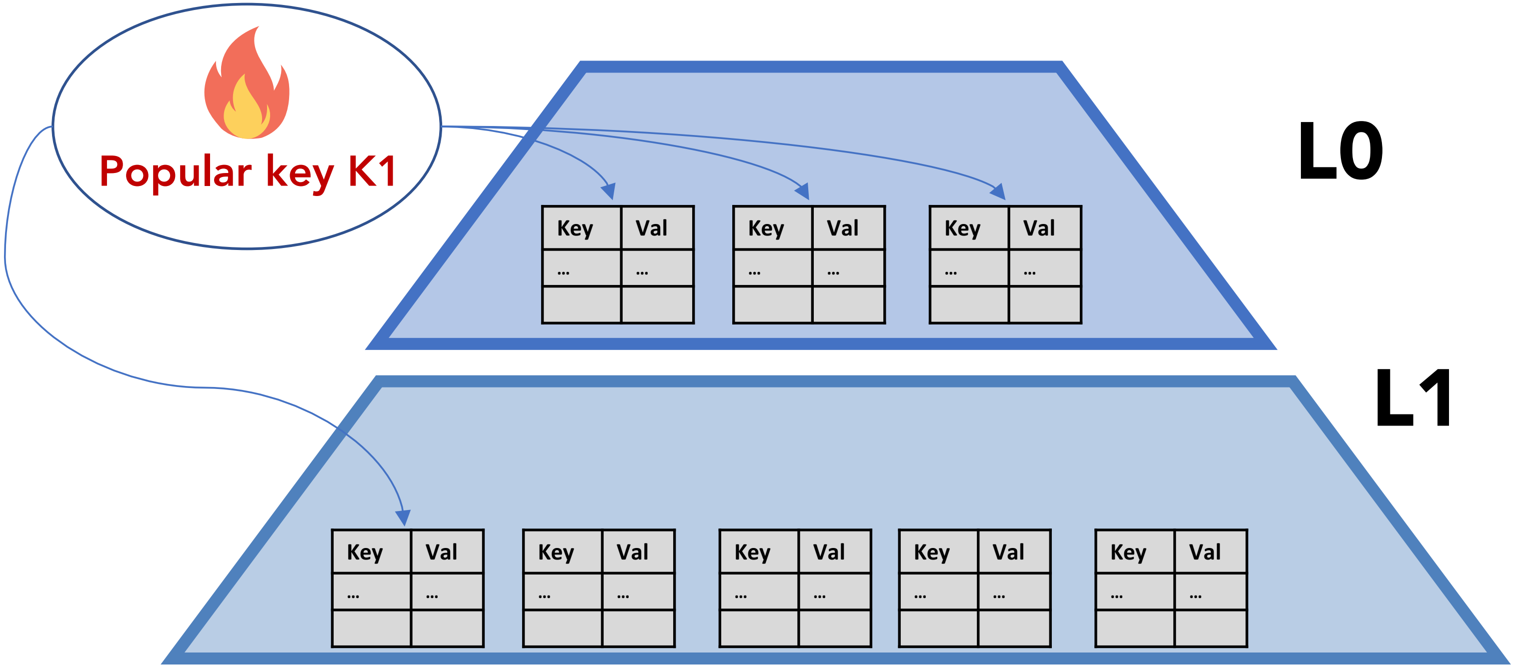
High data skew

- Flush because commit log is full
- Flush mostly empty mem comp

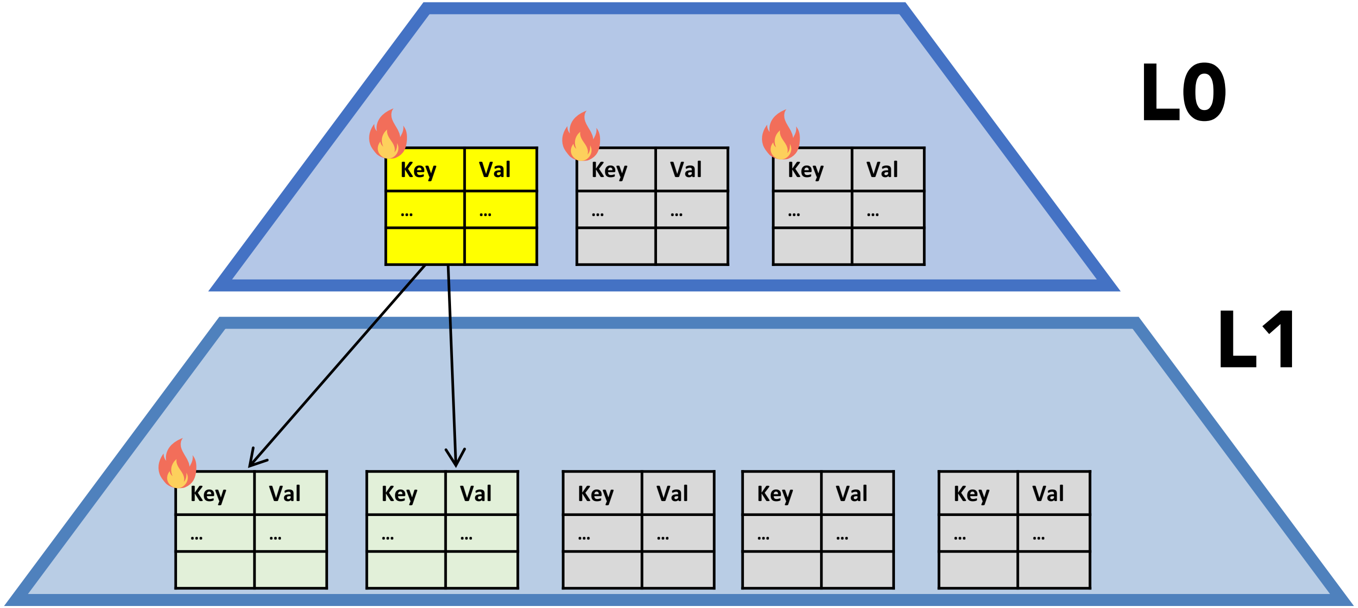
Commit Log

K1	V1$_1$
K2	V2
K1	V1$_2$
K1	V1$_3$
K1	V1$_4$
…	
K1	V1$_n$

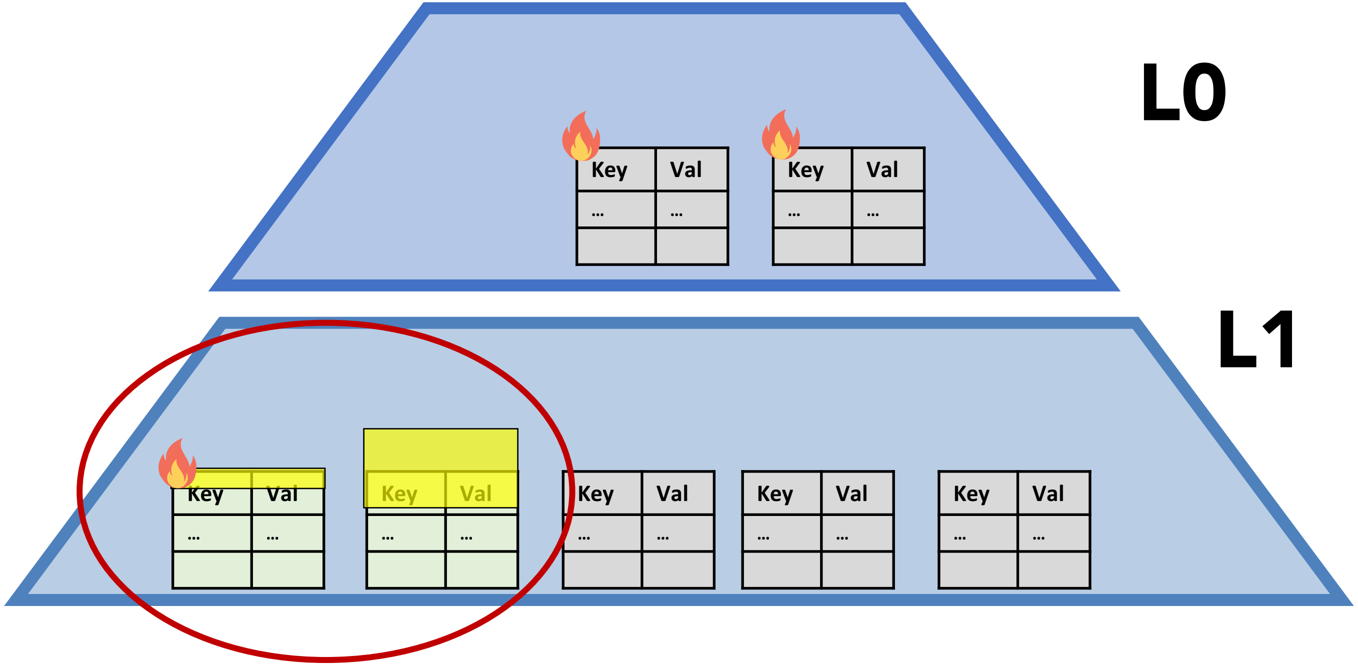
Problem: Compaction with Skewed Workloads



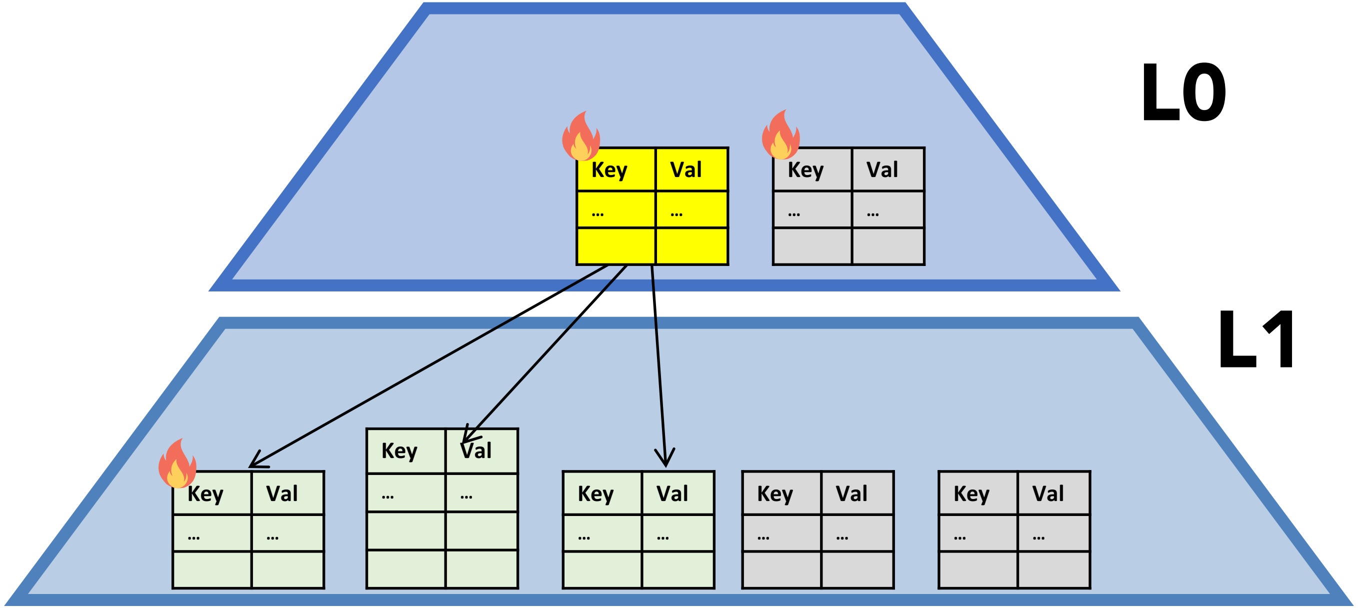
Problem: Compaction with Skewed Workloads



Problem: Compaction with Skewed Workloads

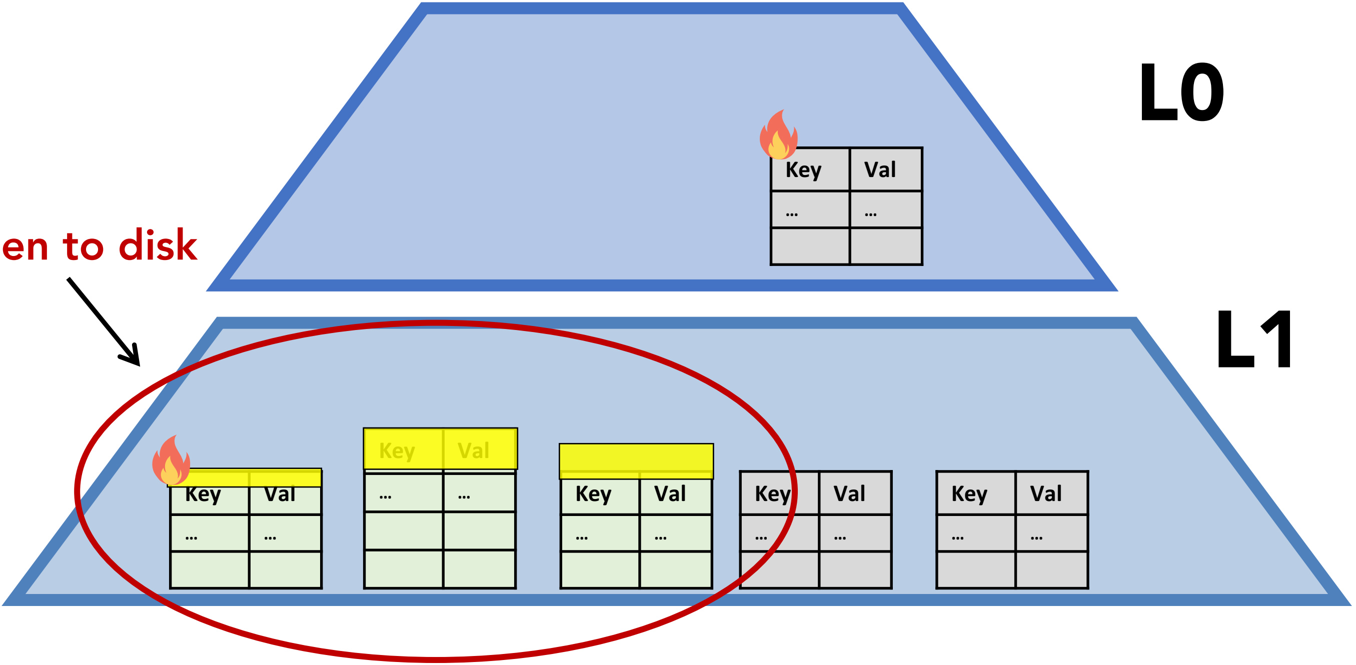


Problem: Compaction with Skewed Workloads



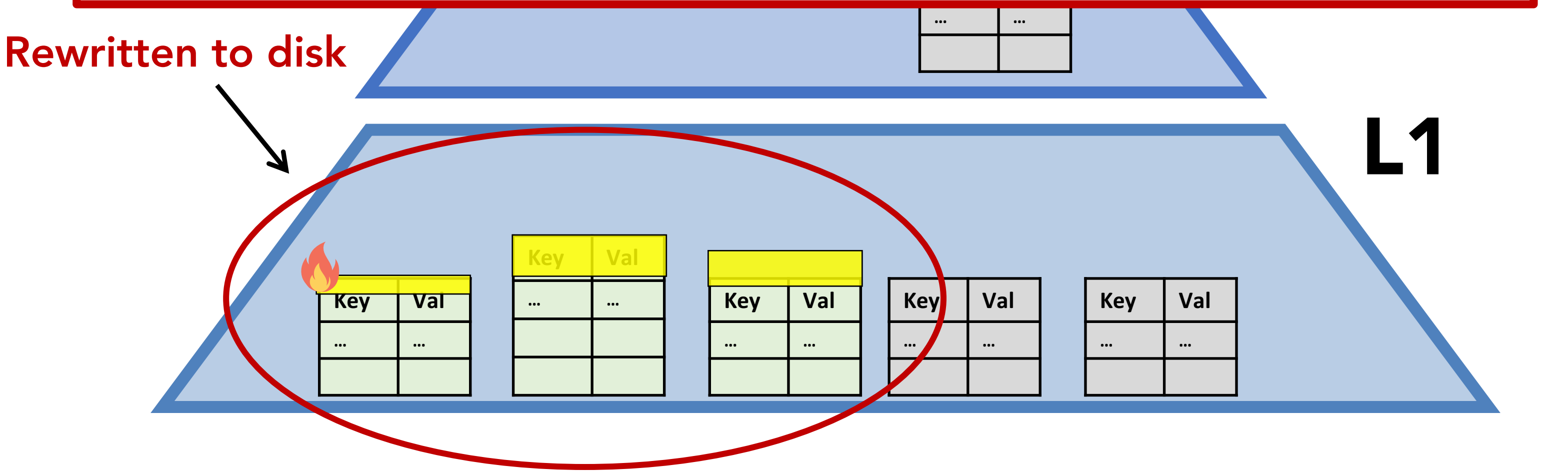
Problem: Compaction with Skewed Workloads

Rewritten to disk

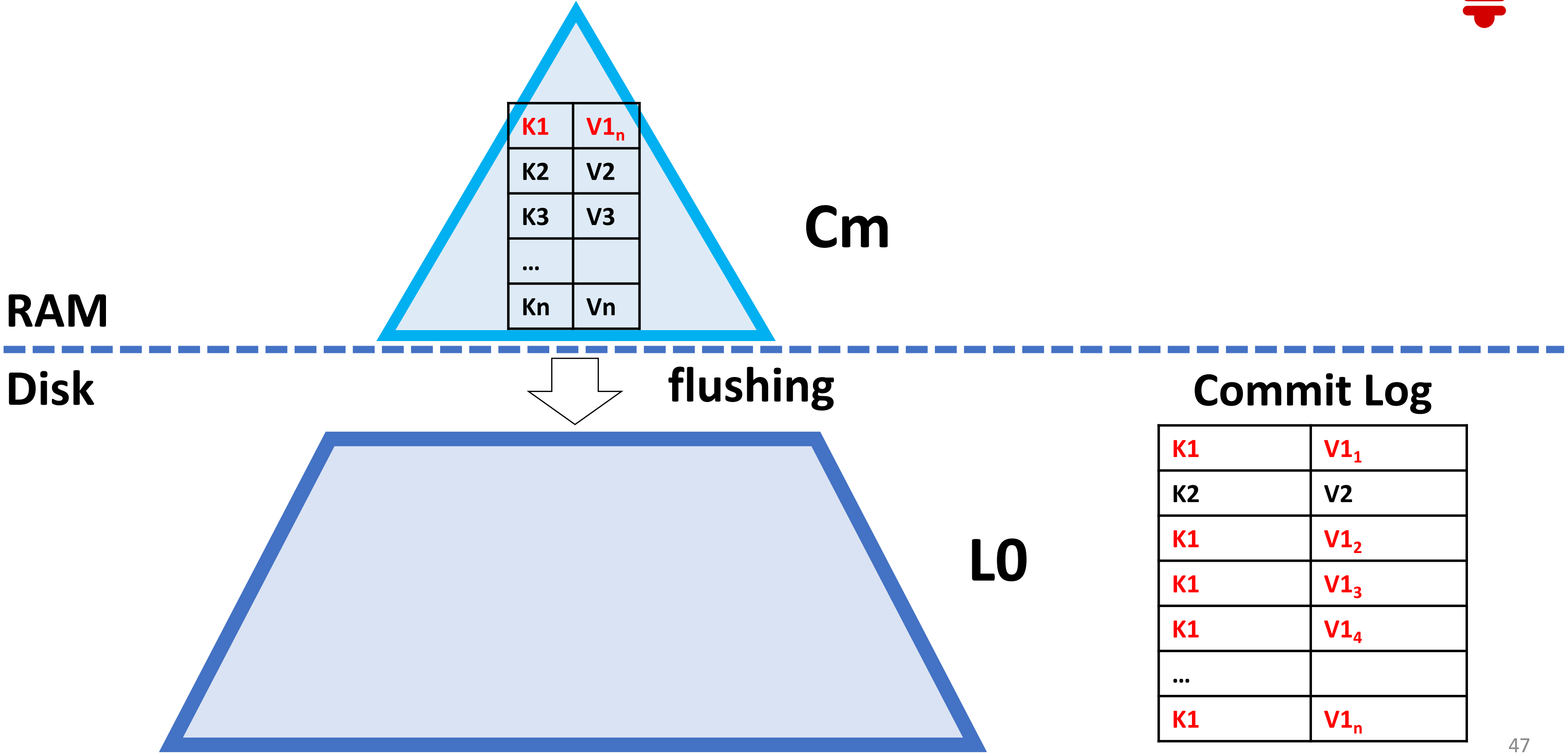


Problem: Compaction with Skewed Workloads

File on L1 **rewritten to disk twice** because of one key ☹



TRIAD-MEM: Hot-cold key separation



K1	$V1_n$
K2	V2
K3	V3
…	
Kn	Vn

Commit Log

K1	$V1_1$
K2	V2
K1	$V1_2$
K1	$V1_3$
K1	$V1_4$
…	
K1	$V1_n$

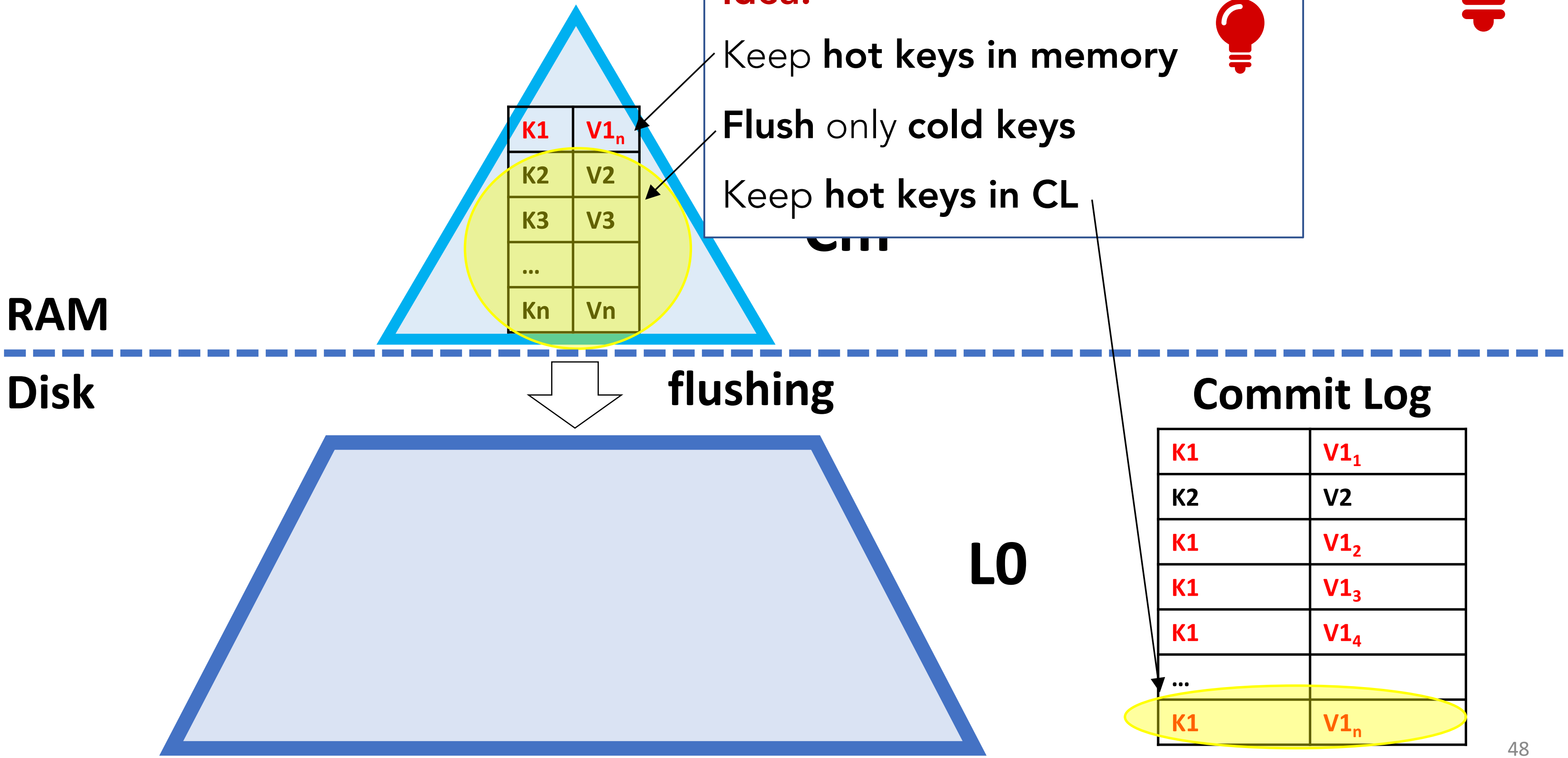
TRIAD-MEM: Hot-cold key separation

Idea:

Keep **hot keys in memory**

Flush only **cold keys**

Keep **hot keys in CL**



K1	$V1_n$
K2	V2
K3	V3
…	
Kn	Vn

Commit Log

K1	$V1_1$
K2	V2
K1	$V1_2$
K1	$V1_3$
K1	$V1_4$
…	
K1	$V1_n$

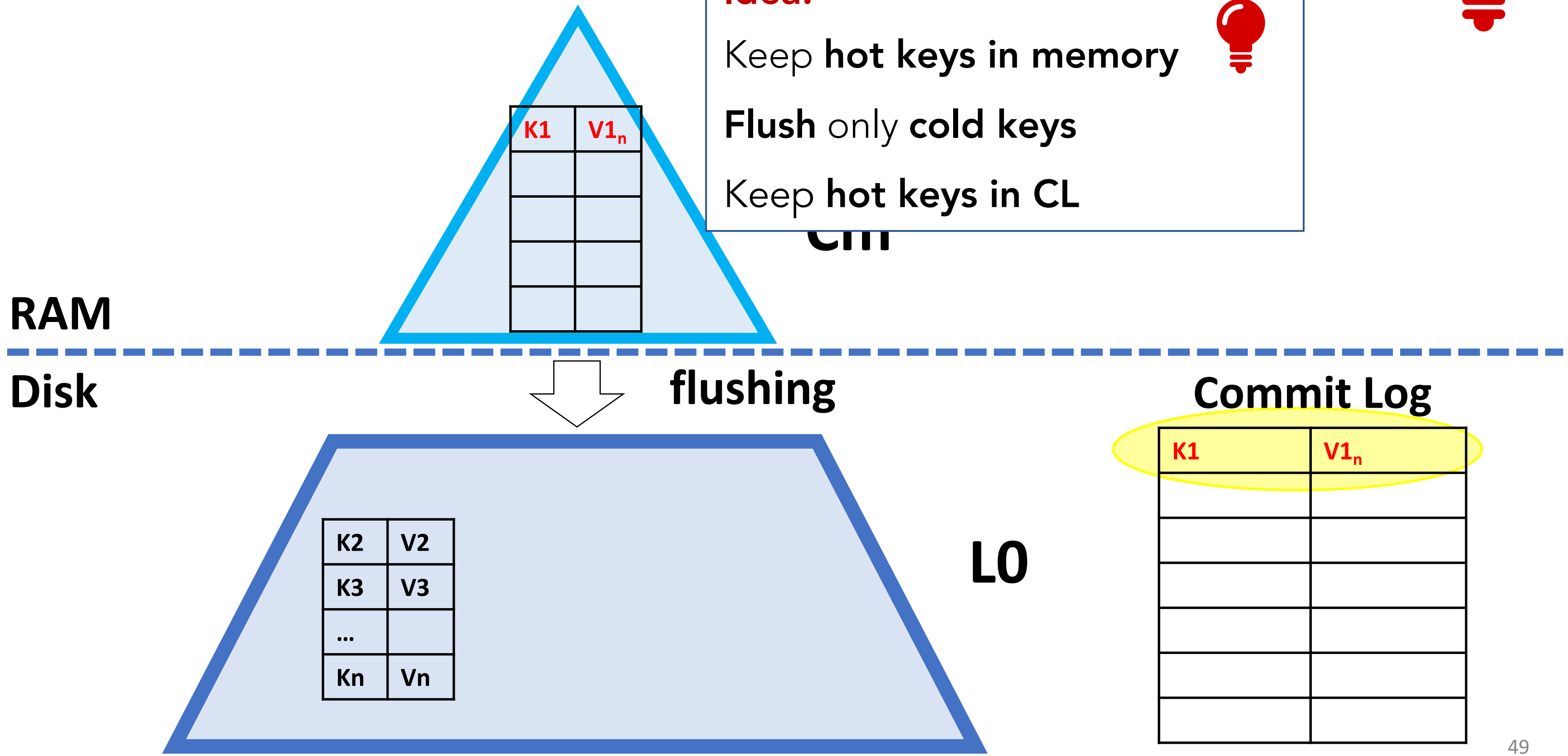
TRIAD-MEM: Hot-cold key separation

Idea:

Keep **hot keys in memory**

Flush only **cold keys**

Keep **hot keys in CL**



K1	$V1_n$

K2	V2
K3	V3
…	
Kn	Vn

K1	$V1_n$

TRIAD-MEM Summary

✓ **Good for skewed workloads.**

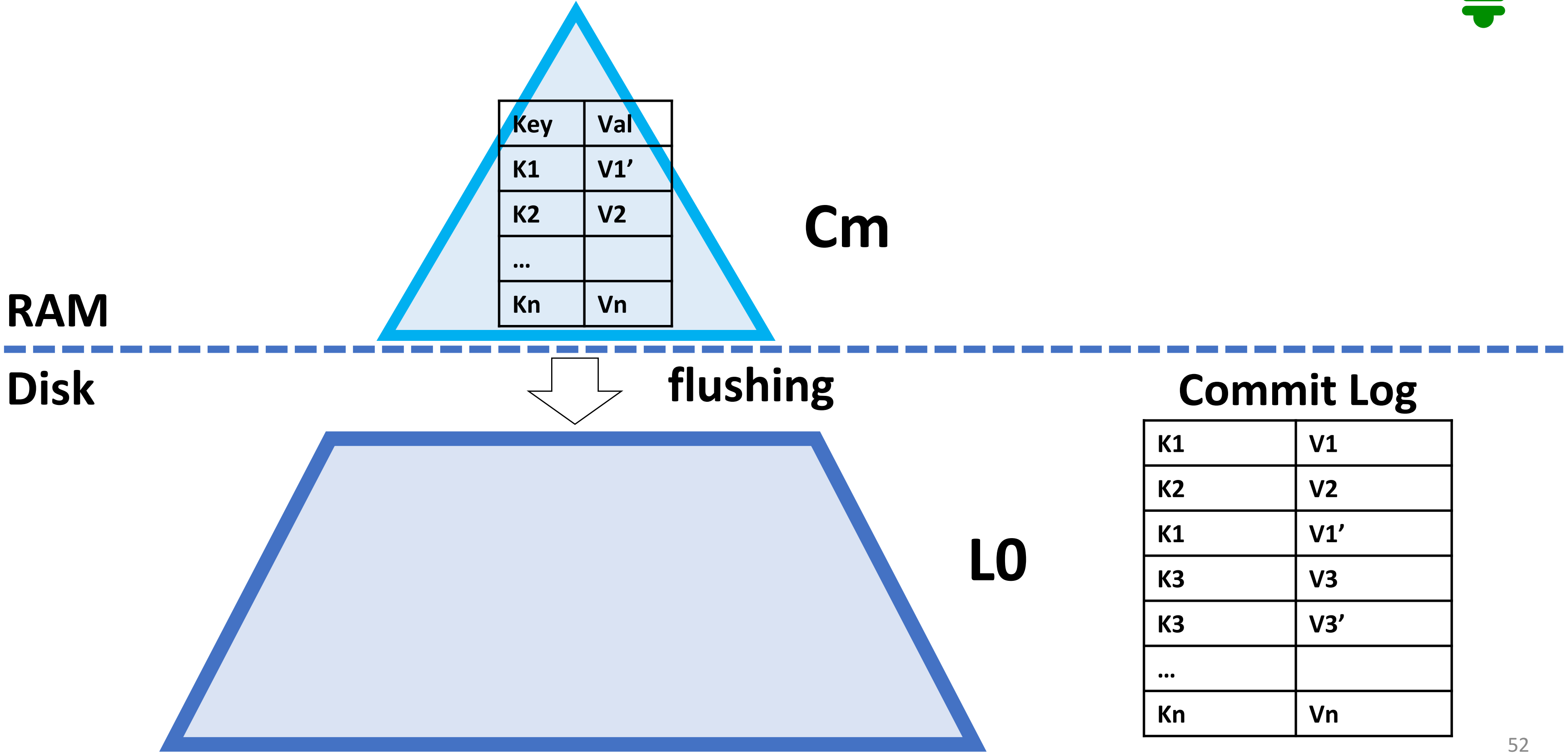
✓ **Reduce flushing WA: less data written from memory to disk.**

✓ **Reduce compaction WA: avoid repeatedly compacting hot keys.**

TRIAD-LOG

	Workload	Improve WA in
TRIAD-LOG	Uniform workloads	Flushing

Problem: Flushing with Uniform Workloads

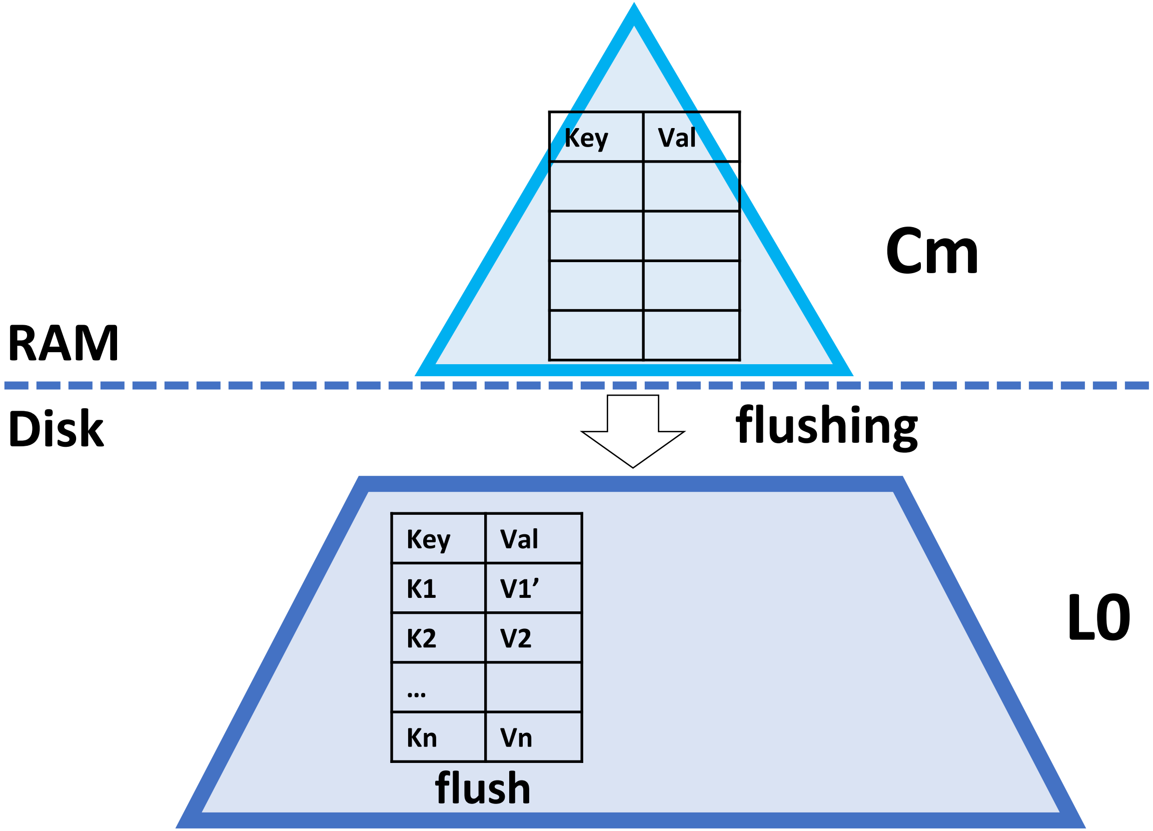


Key	Val
K1	V1’
K2	V2
…	
Kn	Vn

Commit Log

K1	V1
K2	V2
K1	V1’
K3	V3
K3	V3’
…	
Kn	Vn

Problem: Flushing with Uniform Workloads



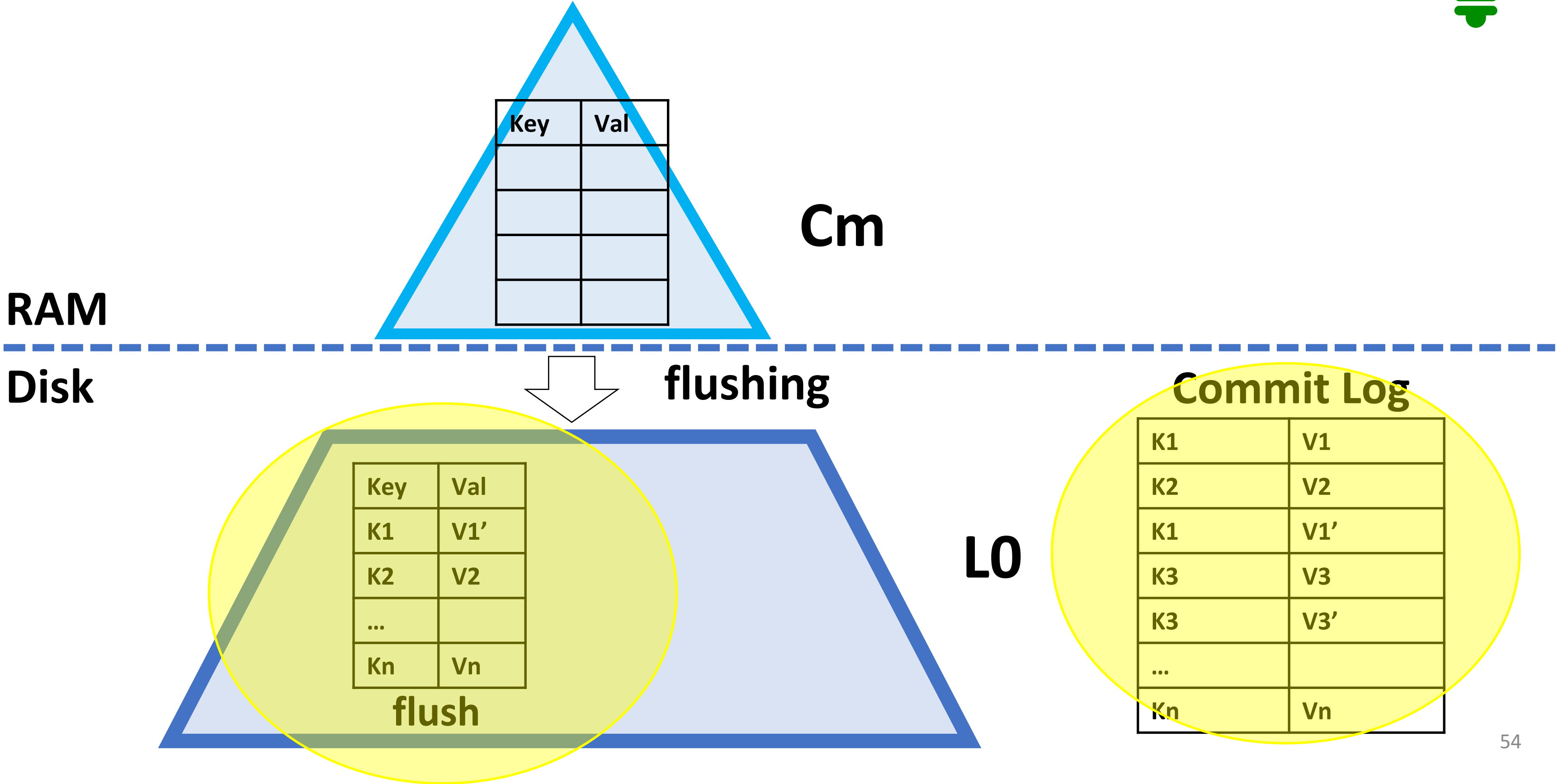
Key	Val

Key	Val
K1	V1’
K2	V2
…	
Kn	Vn

Commit Log

K1	V1
K2	V2
K1	V1’
K3	V3
K3	V3’
…	
Kn	Vn

Problem: Flushing with Uniform Workloads



Key	Val

Key	Val
K1	V1’
K2	V2
…	
Kn	Vn

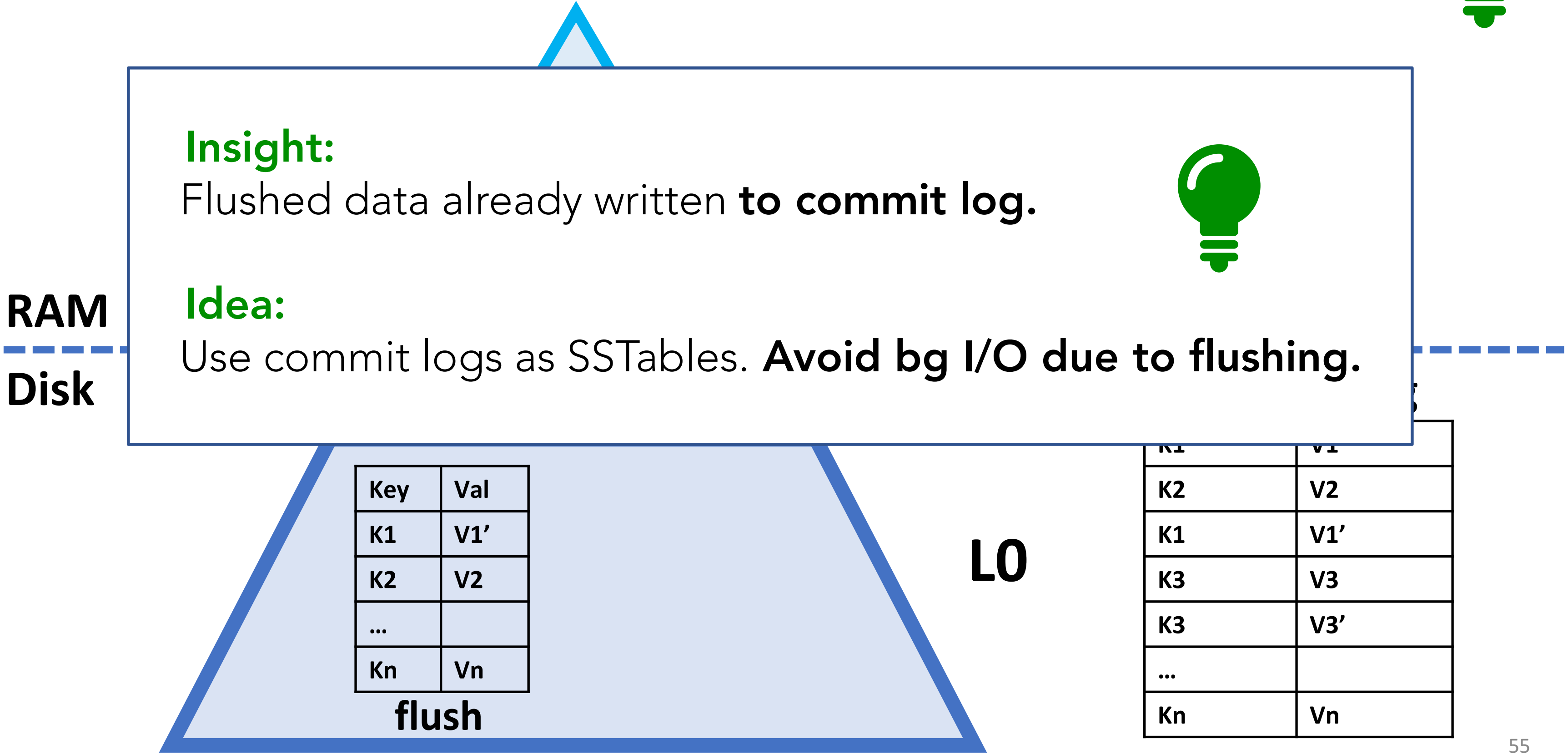
Commit Log

K1	V1
K2	V2
K1	V1’
K3	V3
K3	V3’
…	
Kn	Vn

Problem: Flushing with Uniform Workloads

Insight:
Flushed data already written **to commit log.**

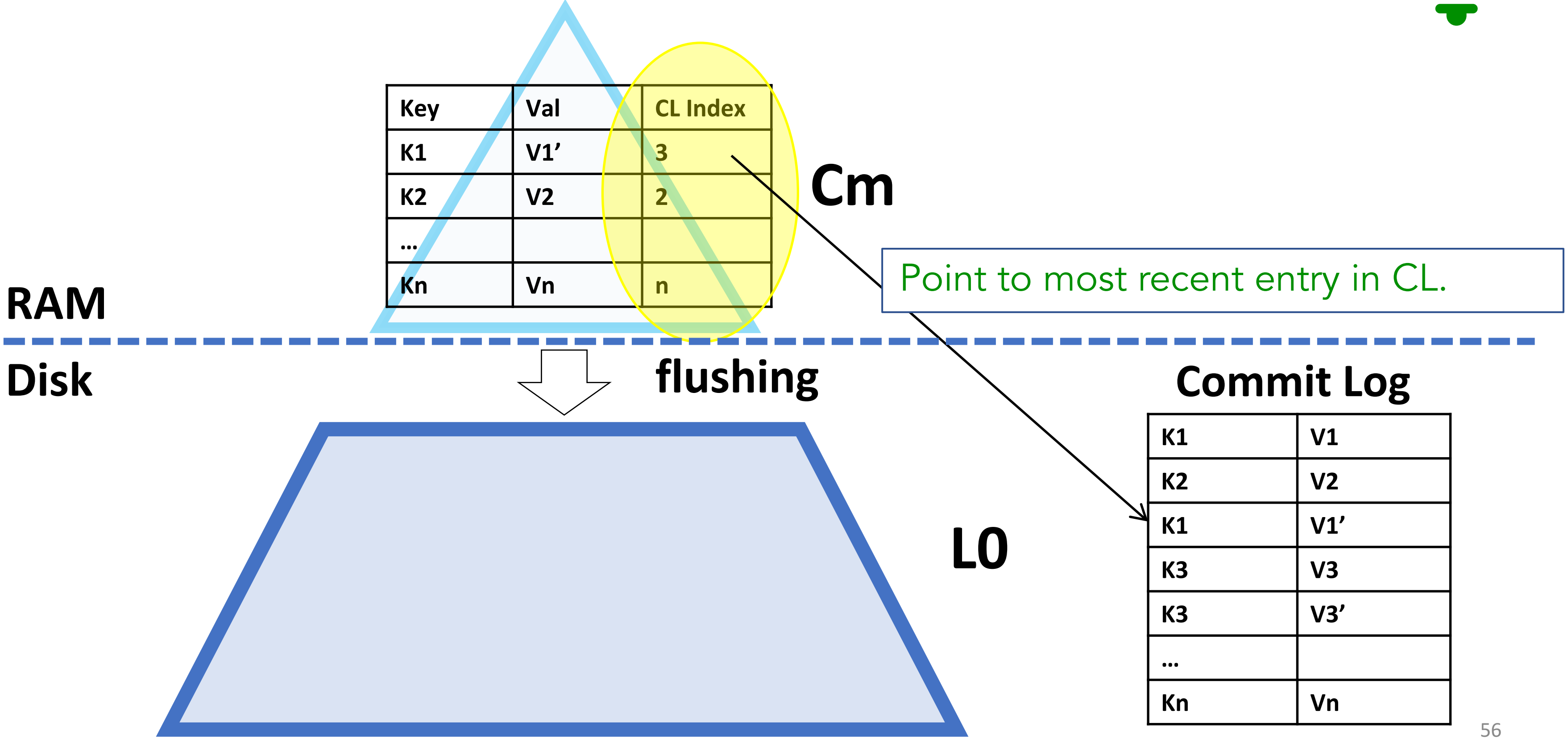
Idea:
Use commit logs as SSTables. **Avoid bg I/O due to flushing.**



Key	Val
K1	V1’
K2	V2
…	
Kn	Vn

K1	V1
K2	V2
K1	V1’
K3	V3
K3	V3’
…	
Kn	Vn

TRIAD-LOG



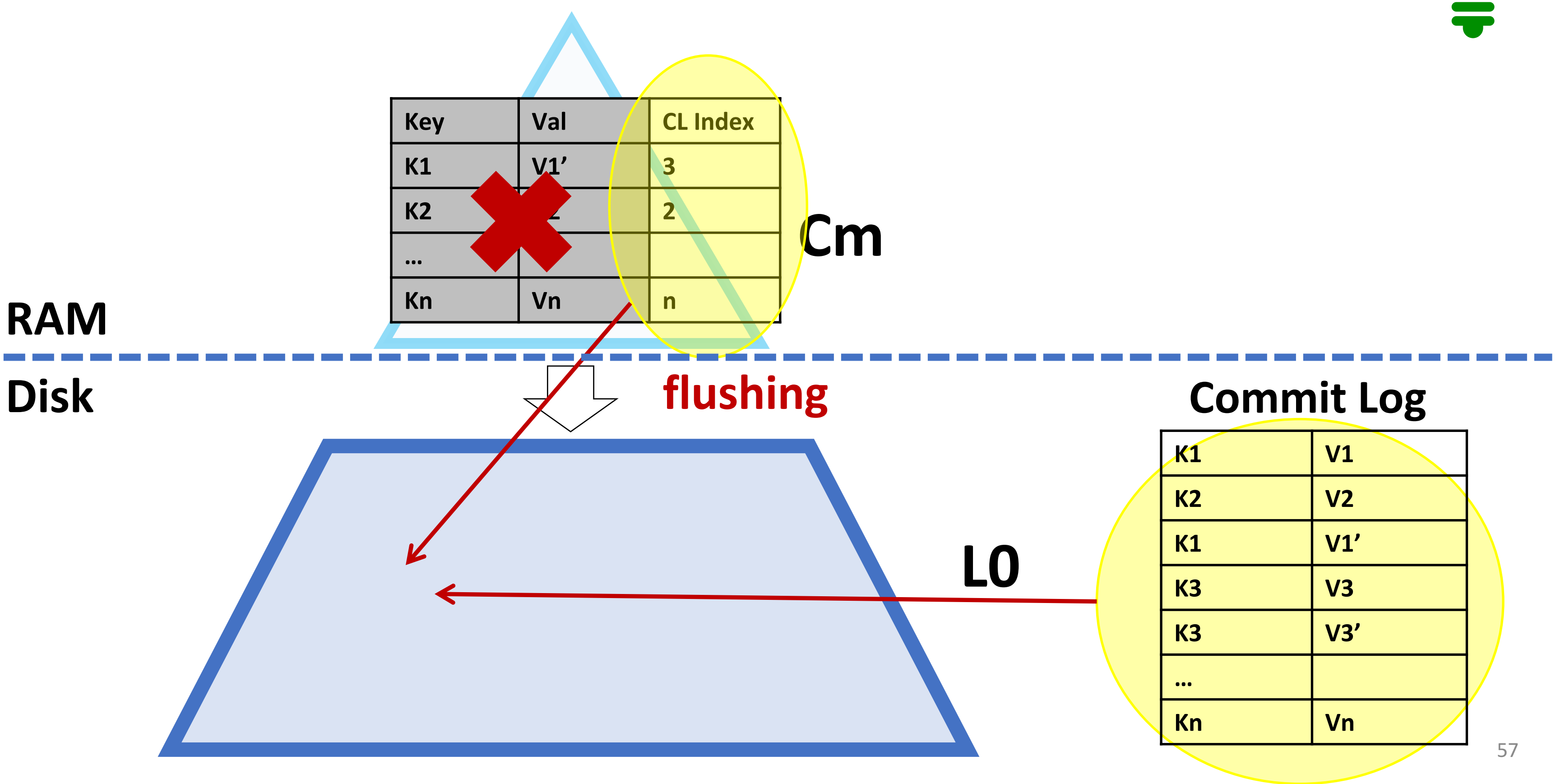
Key	Val	CL Index
K1	V1’	3
K2	V2	2
…		
Kn	Vn	n

Point to most recent entry in CL.

Commit Log

K1	V1
K2	V2
K1	V1’
K3	V3
K3	V3’
…	
Kn	Vn

TRIAD-LOG

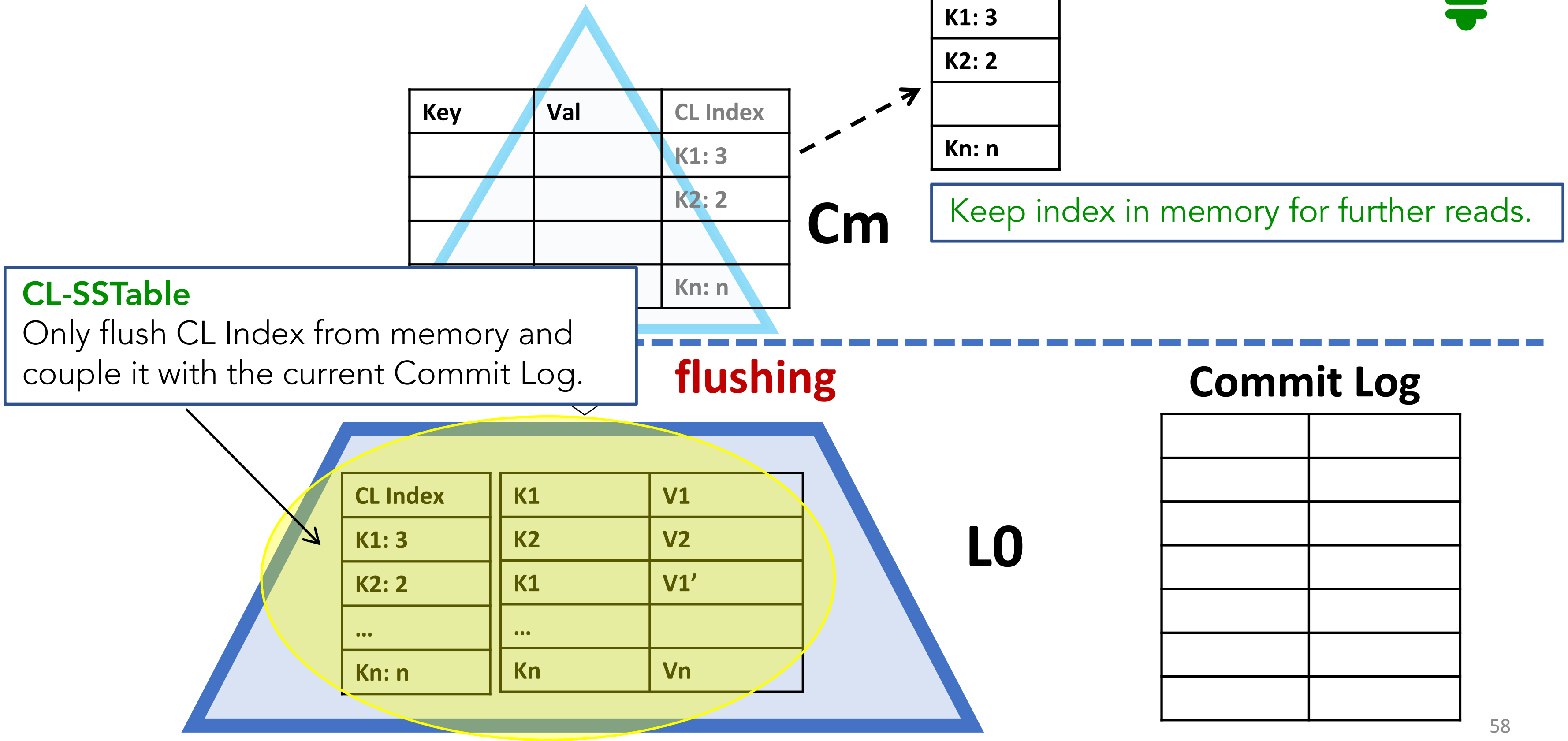


Key	Val	CL Index
K1	V1’	3
K2	2	2
…		
Kn	Vn	n

Commit Log

K1	V1
K2	V2
K1	V1’
K3	V3
K3	V3’
…	
Kn	Vn

TRIAD-LOG



Key	Val	CL Index
		K1: 3
		K2: 2
		Kn: n

CL Index
K1: 3
K2: 2
Kn: n

Keep index in memory for further reads.

CL-SSTable
Only flush CL Index from memory and couple it with the current Commit Log.

CL Index
K1: 3
K2: 2
…
Kn: n

K1	V1
K2	V2
K1	V1’
…	
Kn	Vn

Commit Log

TRIAD-LOG Summary

- ✓ **Good for uniform workloads.**
- ✓ **Reuse Commit Log as L0 SST.**
- ✓ **No more flushing of mem component to disk.**

TRIAD Summary

TRIAD-MEM, TRIAD-DISK, TRIAD-LOG:

- **Complementary**, targeting different wklds.
- Working **simultaneously**.
- **Transparent** to the workloads; no a priori knowledge needed.

Evaluation

Evaluation

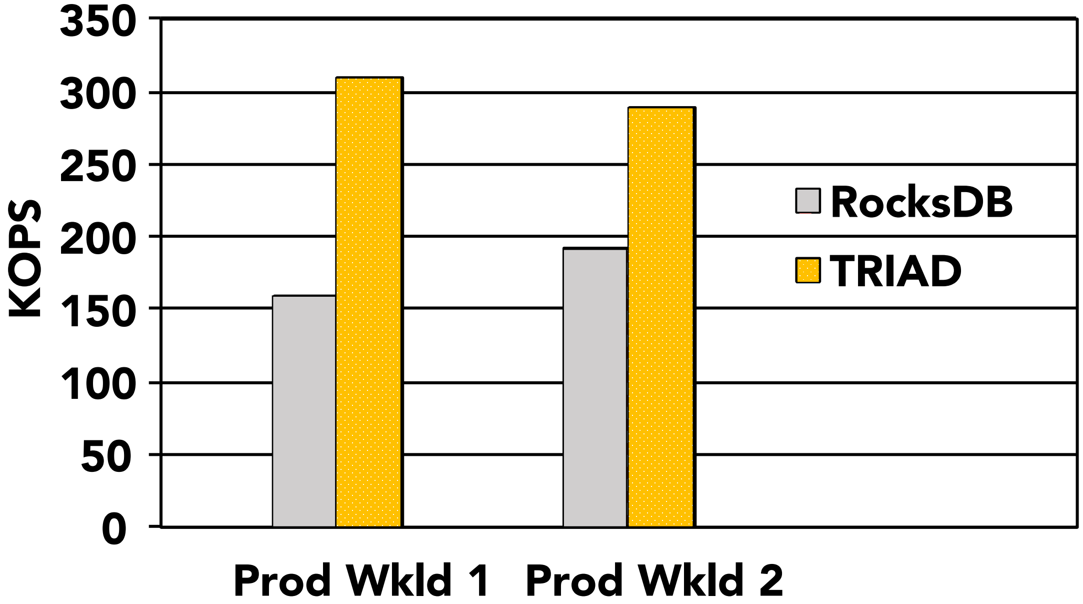
- **Compare TRIAD with RocksDB**
- **Workloads:** Production, Synthetic
- **Metrics:** Throughput, Write Amplification (WA)
- **Code:** `https://github.com/epfl-labos/TRIAD`

Write Amplification (WA)

$$\text{WA} = \frac{\text{total data written to storage}}{\text{data written by app}}$$

Production Workloads: Throughput

higher
is better

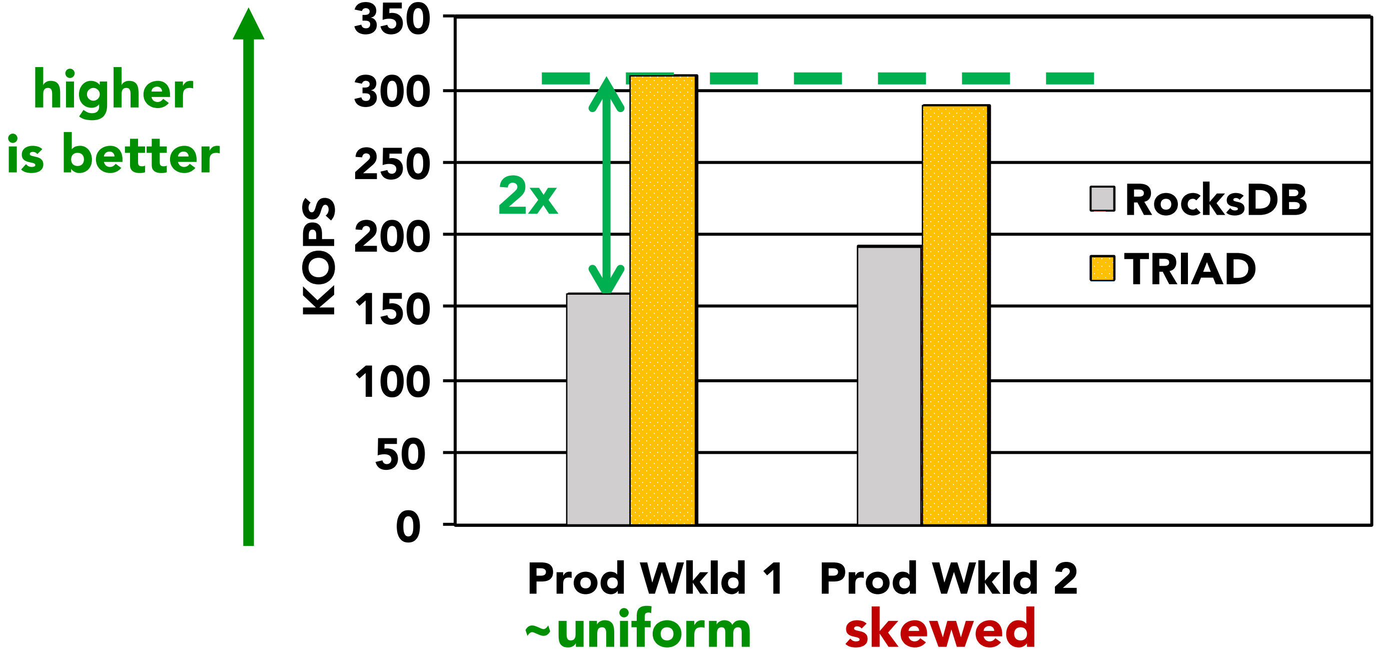


~uniform

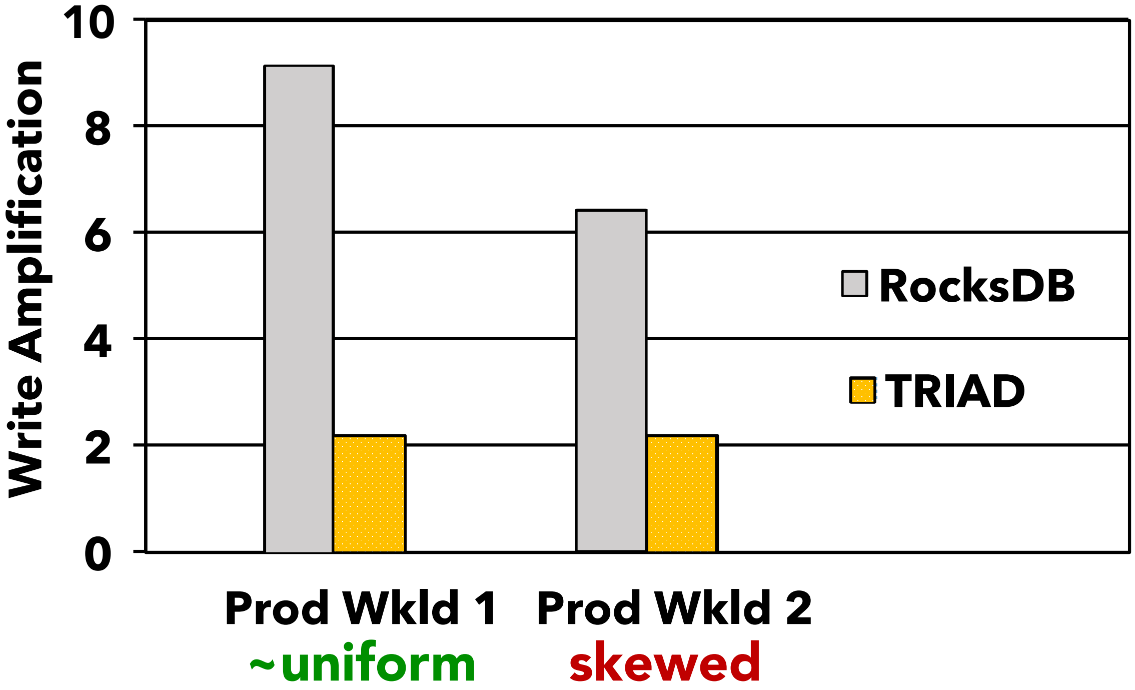
skewed

Production Workloads: Throughput

TRIAD: stable throughput across wklds.

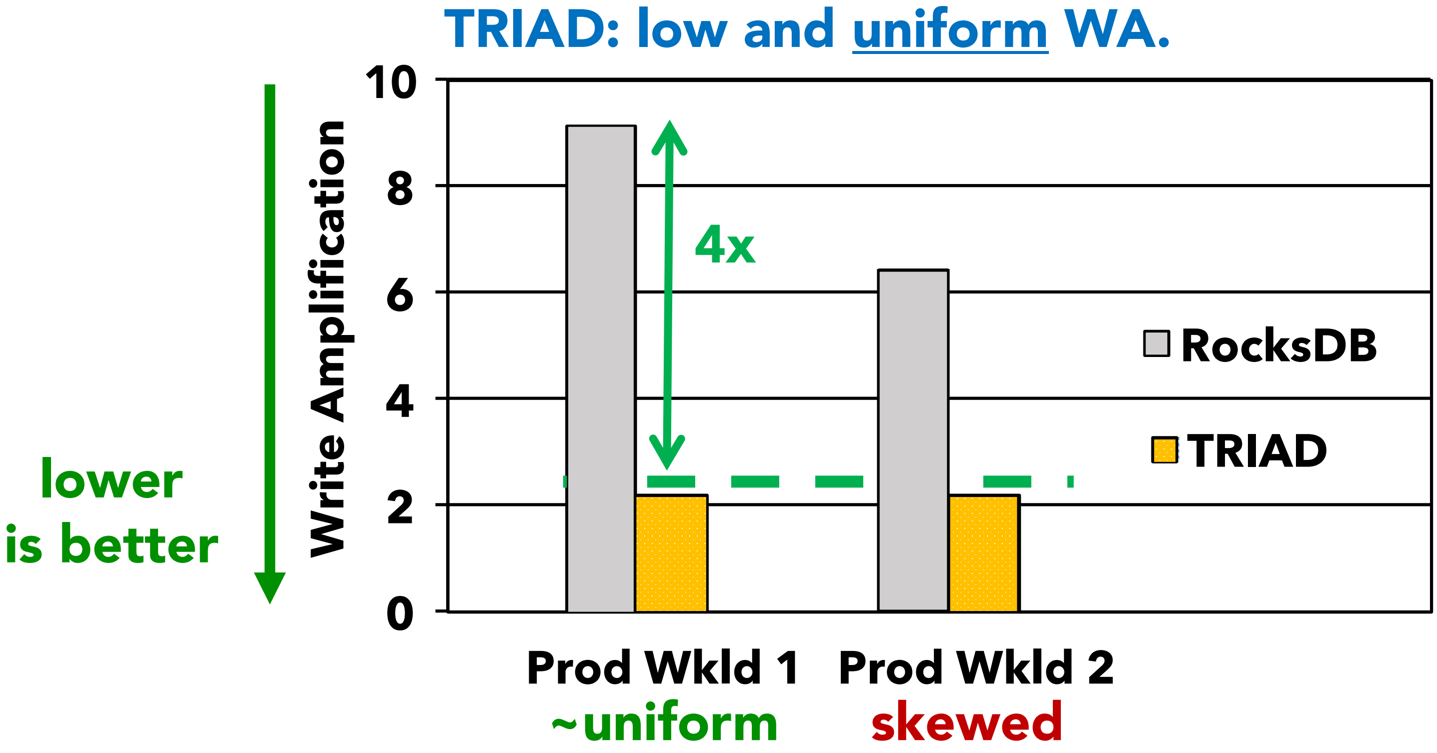


Production Workloads: Write Amplification



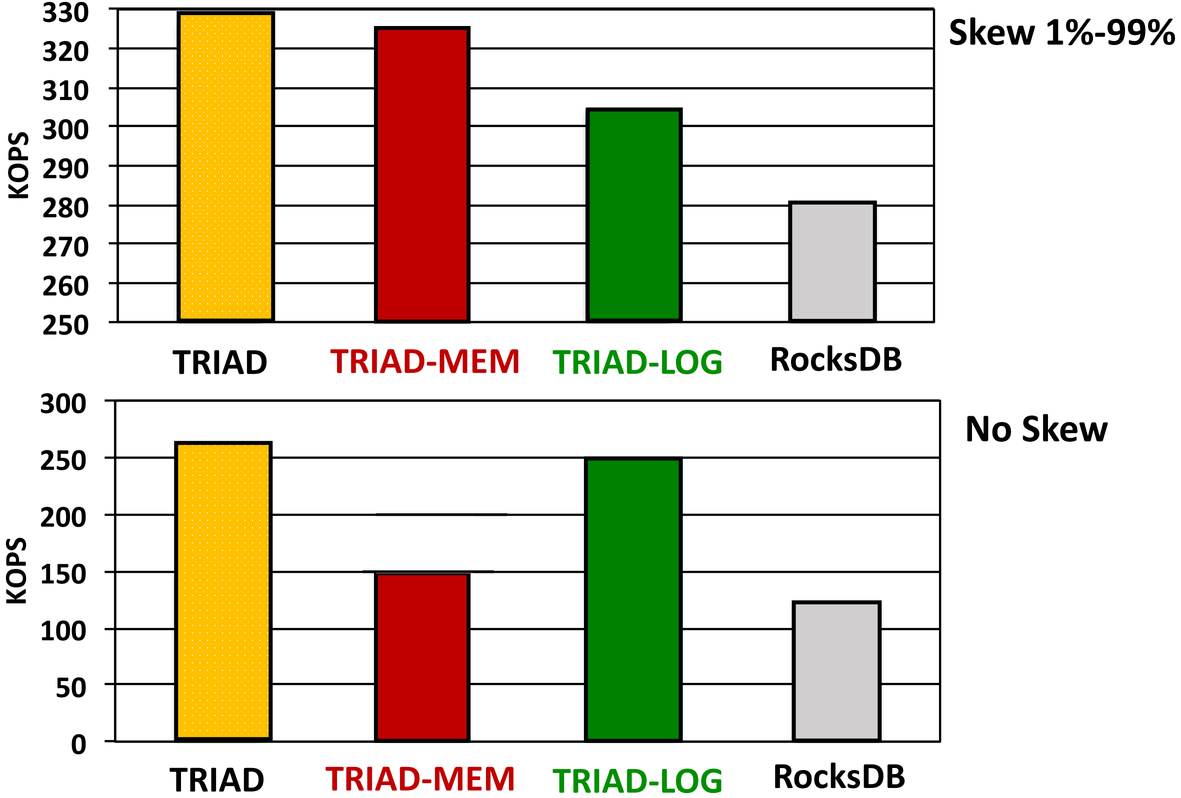
lower is better

Production Workloads: Write Amplification

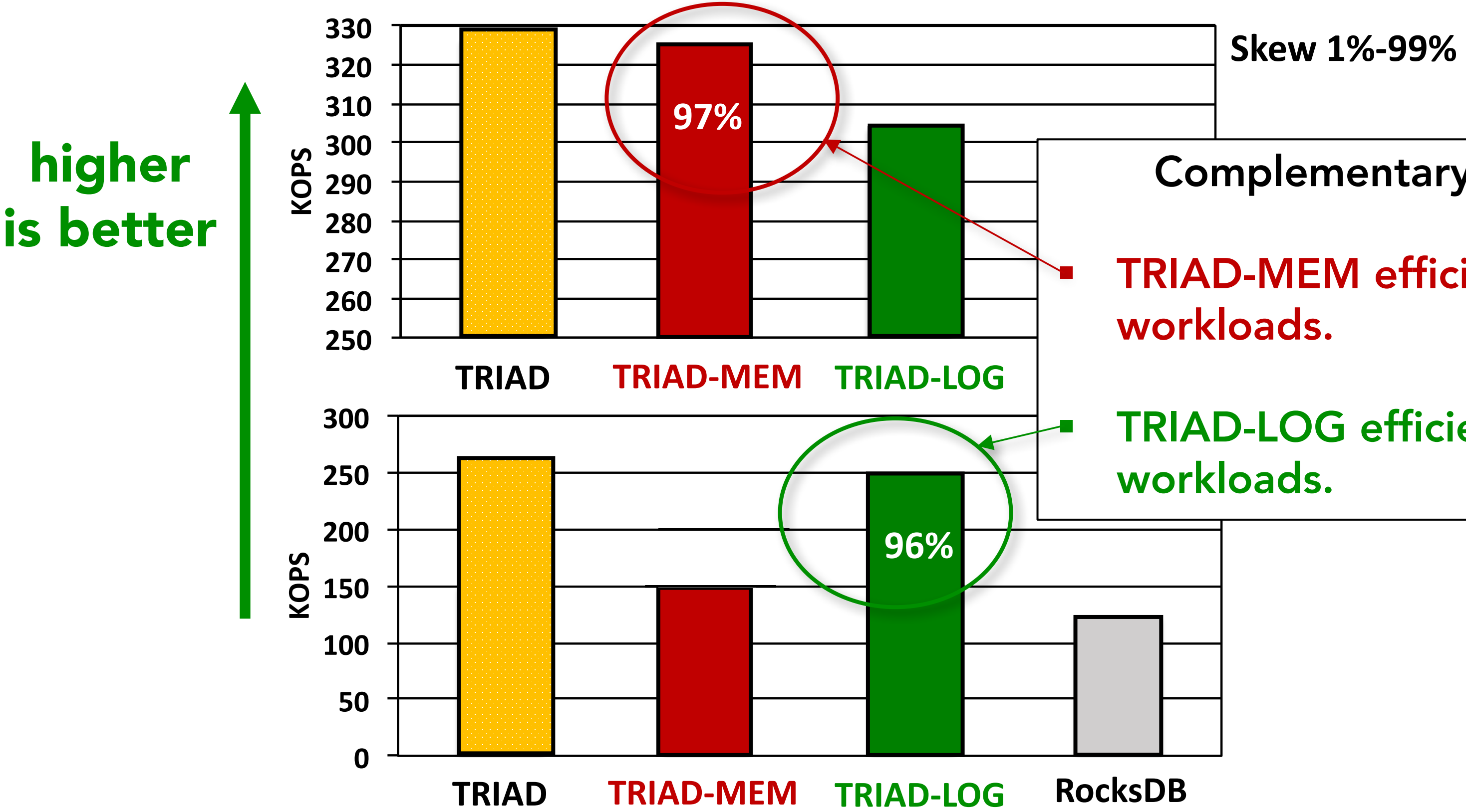


TRIAD: Throughput Breakdown Synthetic Workloads

higher is better



TRIAD: Throughput Breakdown Synthetic Workloads



Complementary techniques

- TRIAD-MEM efficient for skewed workloads.
- TRIAD-LOG efficient for uniform workloads.

More in Our Paper

- More production workloads
- More synthetic workloads
- Detailed breakdown of TRIAD techniques
- TRIAD-DISK

Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to **reduce WA.**
  - **LSMs:** RocksDB, *bLSM* (SIGMOD/PODS '12), *VT-tree* (FAST '13), *HyperLevelDB*, *LSM-trie* (USENIX ATC '15), Cassandra, WiscKey (FAST '16).

Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to **reduce WA.**
  - **LSMs:** RocksDB, *bLSM* (SIGMOD/PODS ’12), *VT-tree* (FAST ‘13), *HyperLevelDB, LSM-trie* (USENIX ATC ‘15), Cassandra, WiscKey (FAST ‘16).

  - **B-epsilon trees:** Tucana (USENIX ATC ’16), BetrFS (FAST ’15, ‘16)

Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to **reduce WA.**
  - **LSMs:** RocksDB, *bLSM* (SIGMOD/PODS '12), *VT-tree* (FAST '13), *HyperLevelDB, LSM-trie* (USENIX ATC '15), Cassandra, WiscKey (FAST '16).

  - **B-epsilon trees:** Tucana (USENIX ATC '16), BetrFS (FAST '15, '16)

  - But no hot/cold key separation, not use commit log as a pseudo-SST.

Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to **reduce WA.**
  - **LSMs:** RocksDB, *bLSM* (SIGMOD/PODS '12), *VT-tree* (FAST '13), *HyperLevelDB, LSM-trie* (USENIX ATC '15), Cassandra, WiscKey (FAST '16).
  - **B-epsilon trees:** Tucana (USENIX ATC '16), BetrFS (FAST '15, '16)
  - But no hot/cold key separation, not use commit log as a pseudo-SST.

- Hot/cold key separation idea used in different context.
  - **FLASH storage systems:** dual-pool algorithm (SAC '07), Application-Managed Flash (FAST '16)

Take-home Messages

✓ **TRIAD: LSM key-value store with high throughput and low WA.**

Take-home Messages

✓ **TRIAD: LSM key-value store with high throughput and low WA.**

✓ **Complementary techniques transparent to workload types.**

Take-home Messages

✓ **TRIAD: LSM key-value store with high throughput and low WA.**

✓ **Complementary techniques transparent to workload types.**

✓ **Impact of LSM I/O on user throughput reduced.**