FloDB: Unlocking Memory in Persistent Key-Value Stores

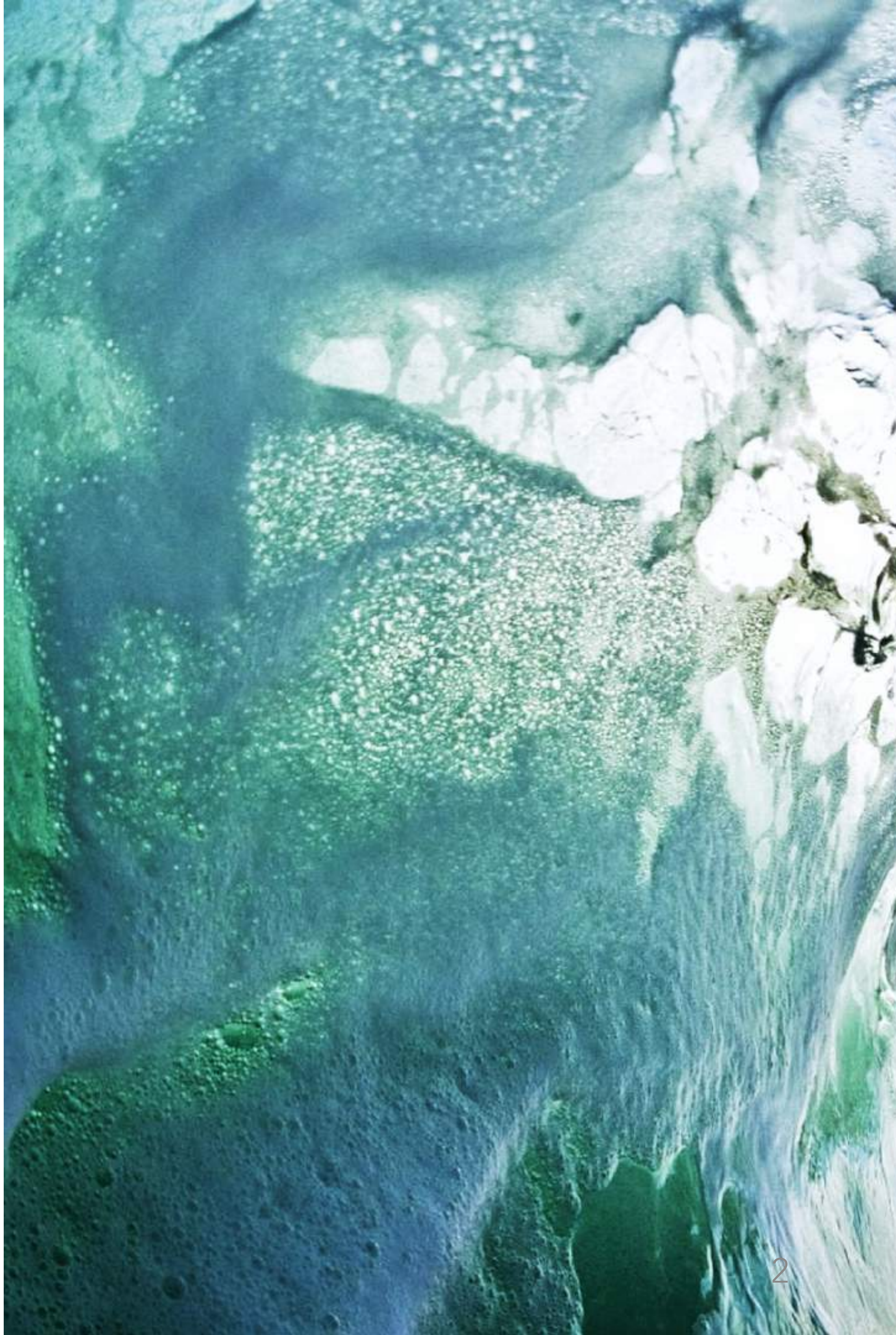
O. Balmau[1], R. Guerraoui[1], V. Trigonakis[2], I. Zablotchi[1]

[1]EPFL, [2]Oracle

EuroSys ’17, Belgrade

Outline

- **Motivation**
- FloDB
- Evaluation
- Conclusion



KV Stores

Very **simple** data stores.

KV pairs.

In-memory vs. <u>persistent</u>

KV Stores

Very **simple** data stores.

KV pairs.

In-memory vs. <u>persistent</u>

Log-Structured Merge (LSM)

mongoDB

BetrFS

MEMCACHED

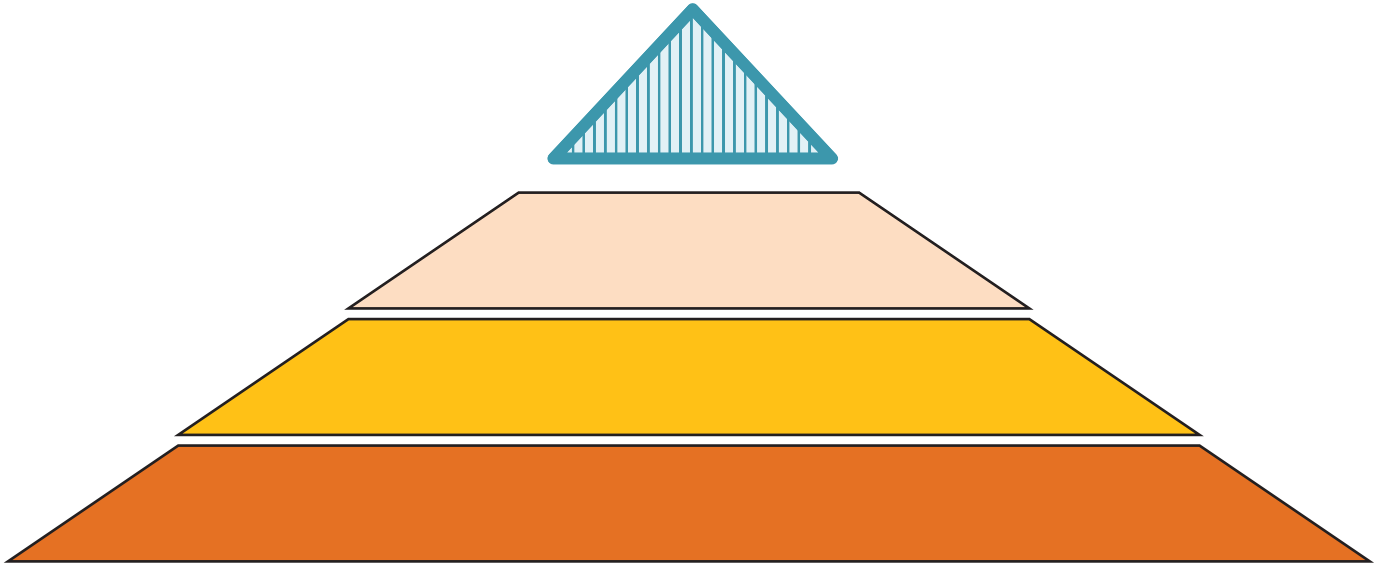
cassandra

redis

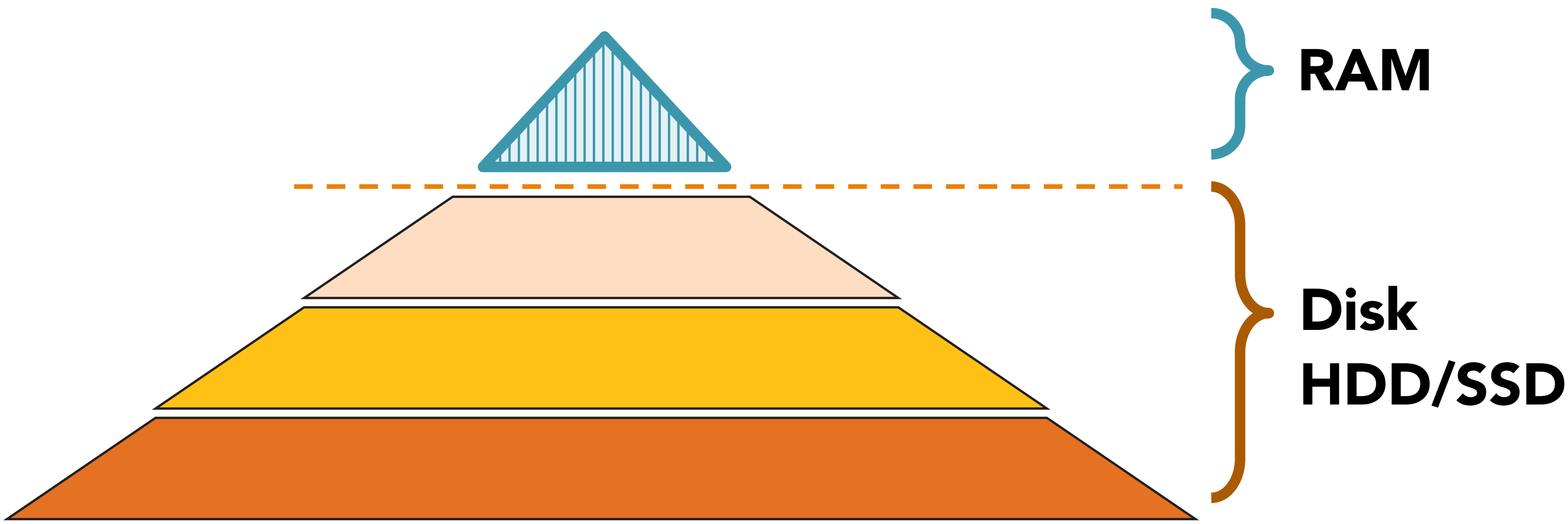
LEVELDB

RocksDB

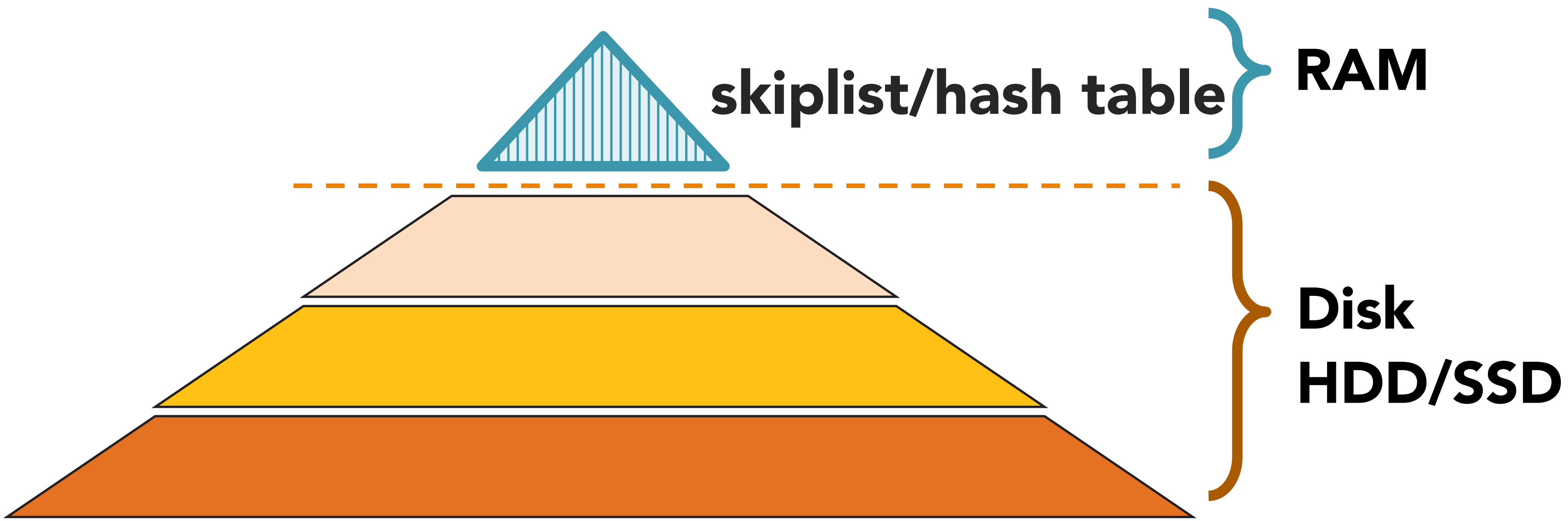
LSM Overview



LSM Overview

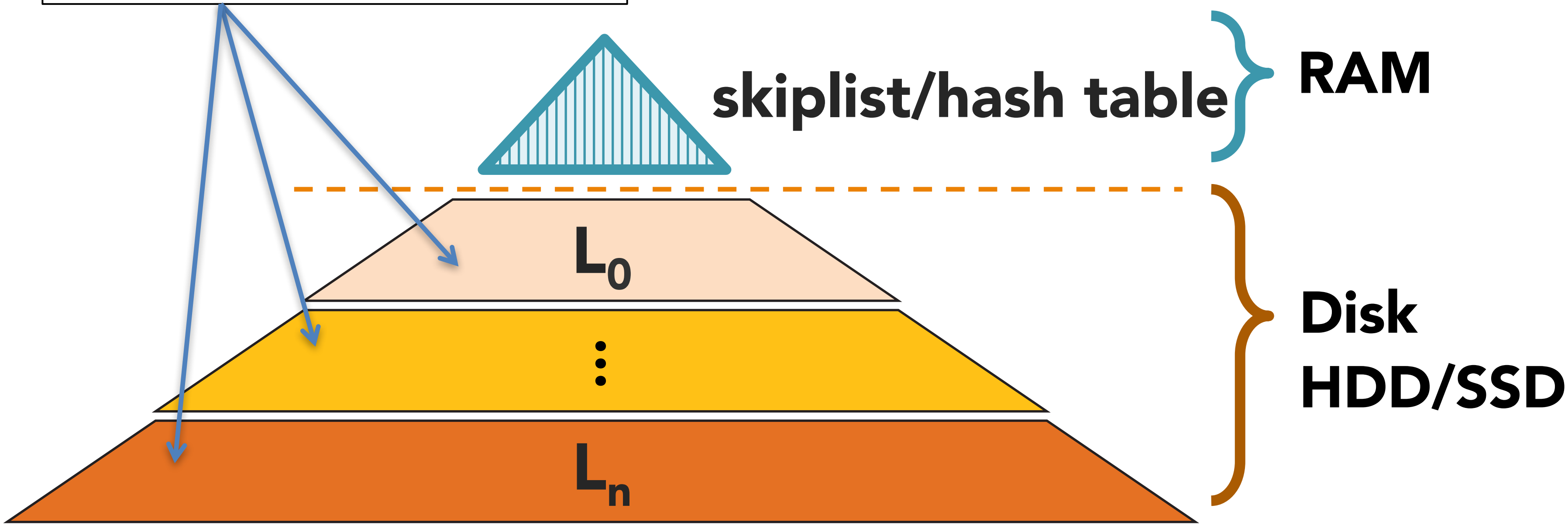


LSM Overview

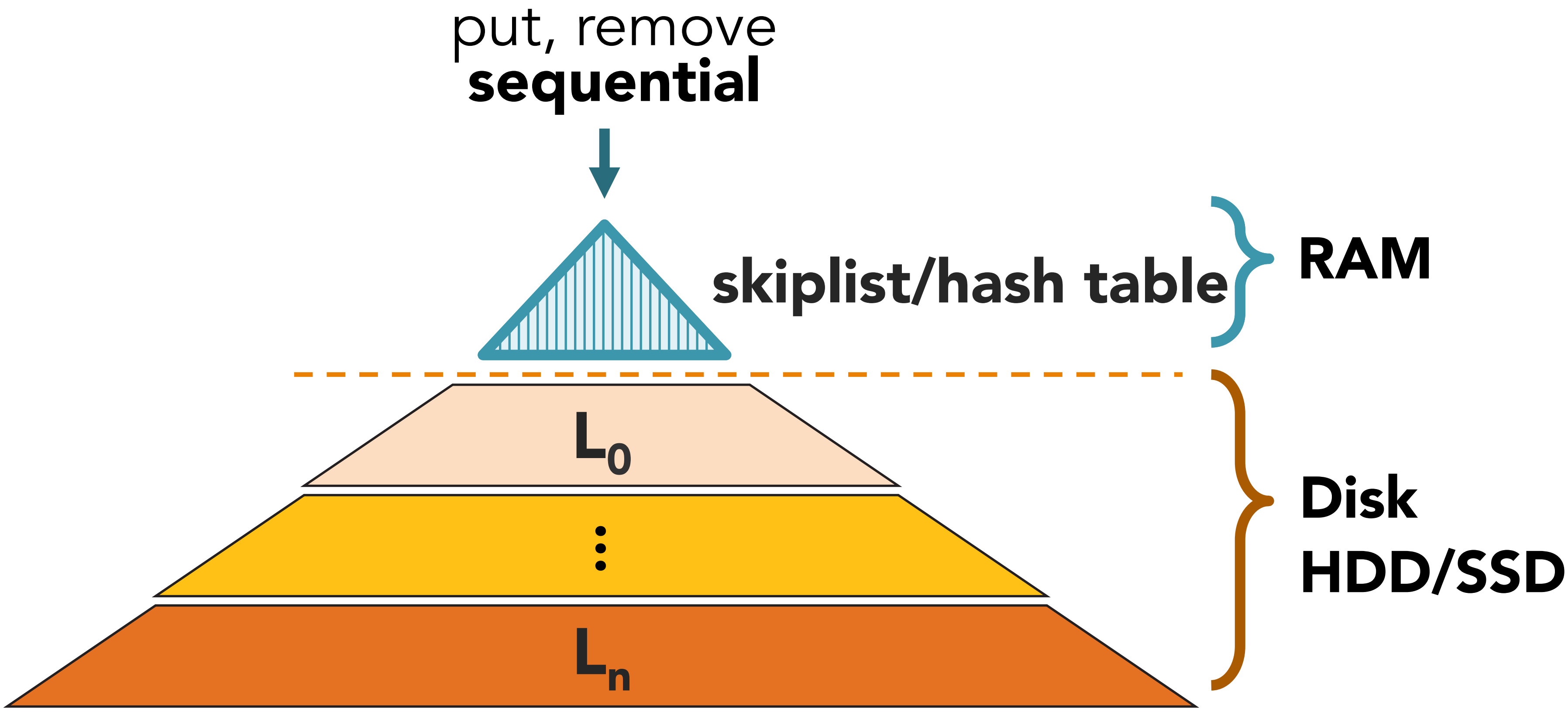


LSM Overview

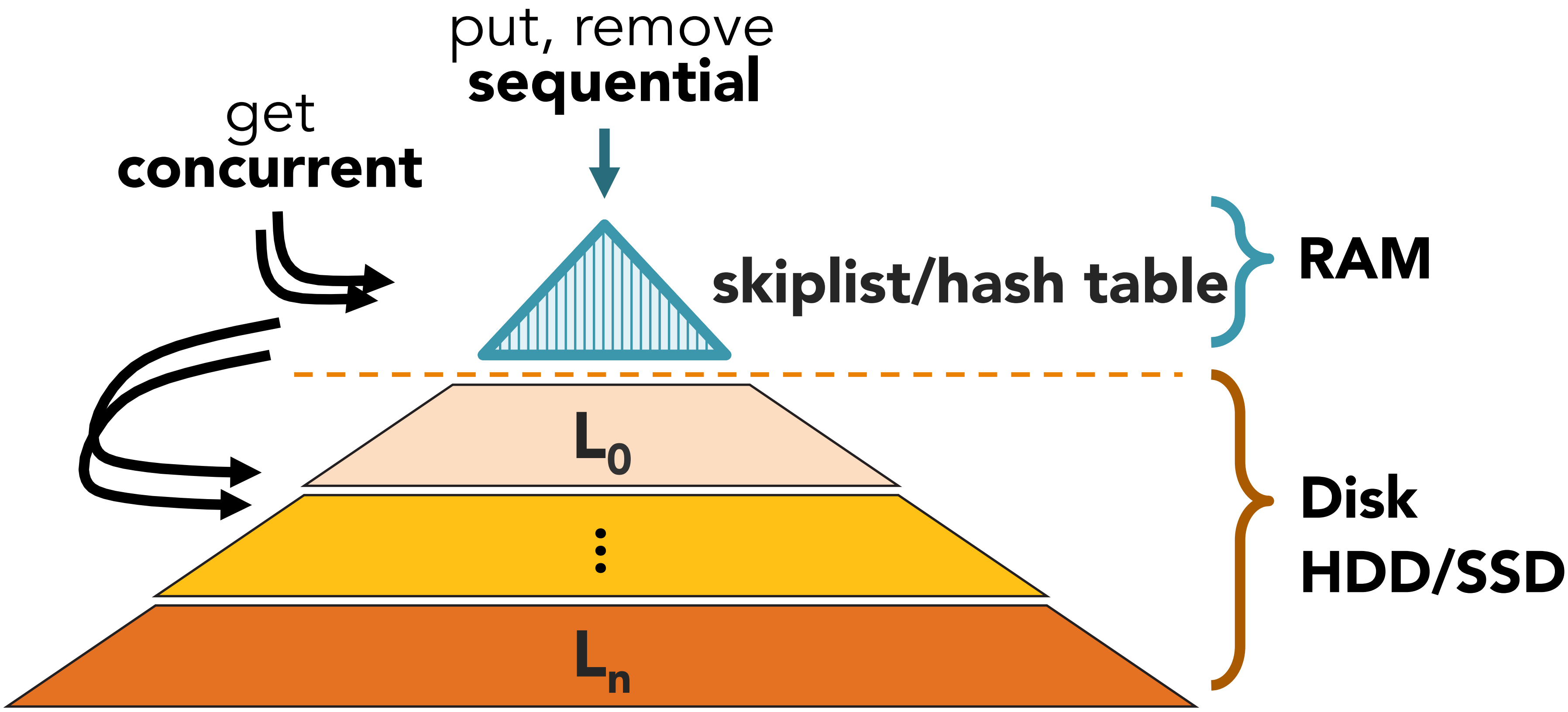
- On-disk levels
- Sorted files (SST)
- Many SSTs/Level



LSM Overview



LSM Overview



Current LSM Limitations

1. Scalability with **threads.**

   Due to global locking synchronization.

2. Scalability with **memory size.**

   Need to keep elements sorted (expensive) .

Current LSM Limitations

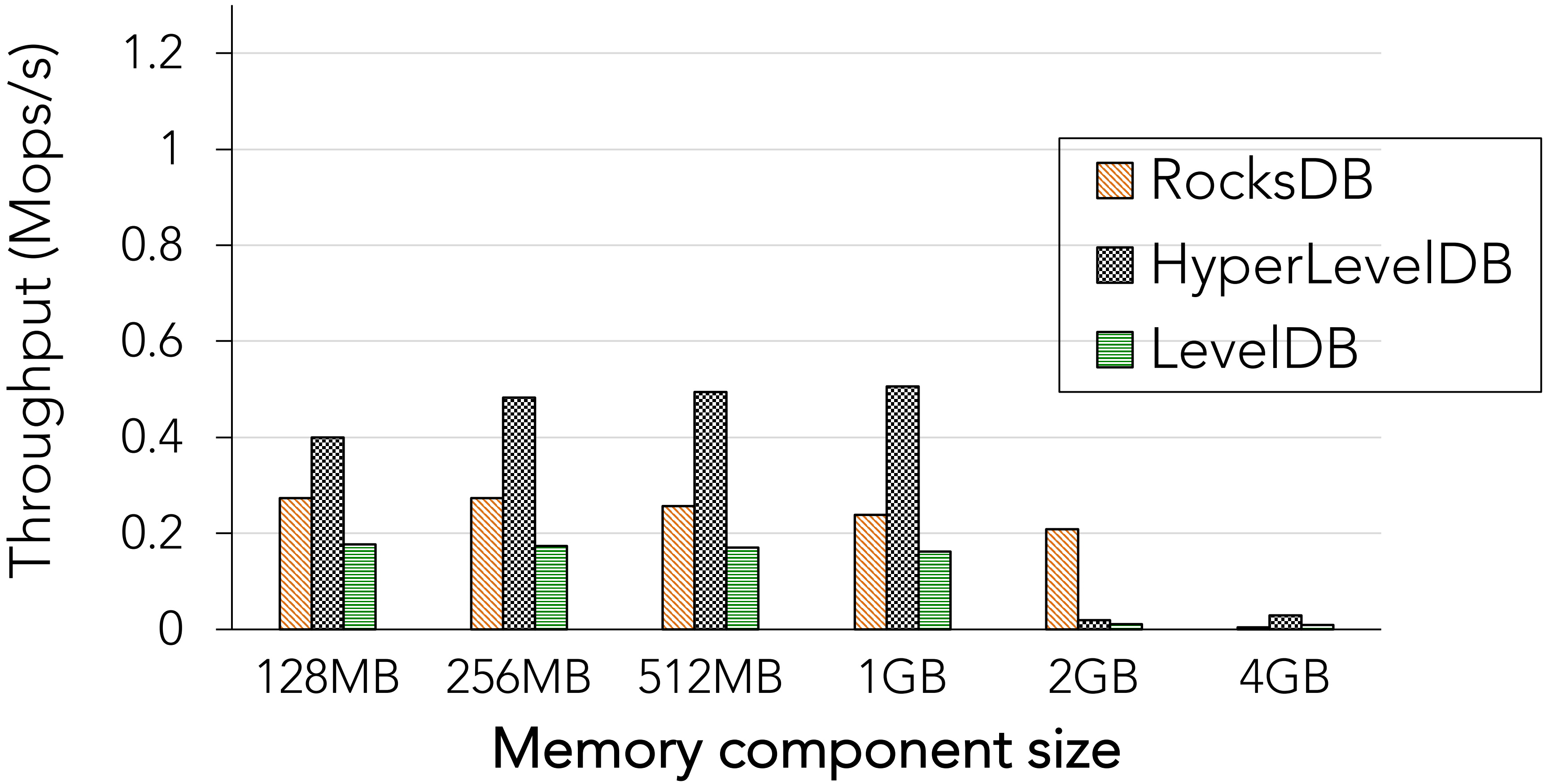
1. Scalability with **threads.**

This talk: focus on memory component.

2. Scalability with **memory size.**

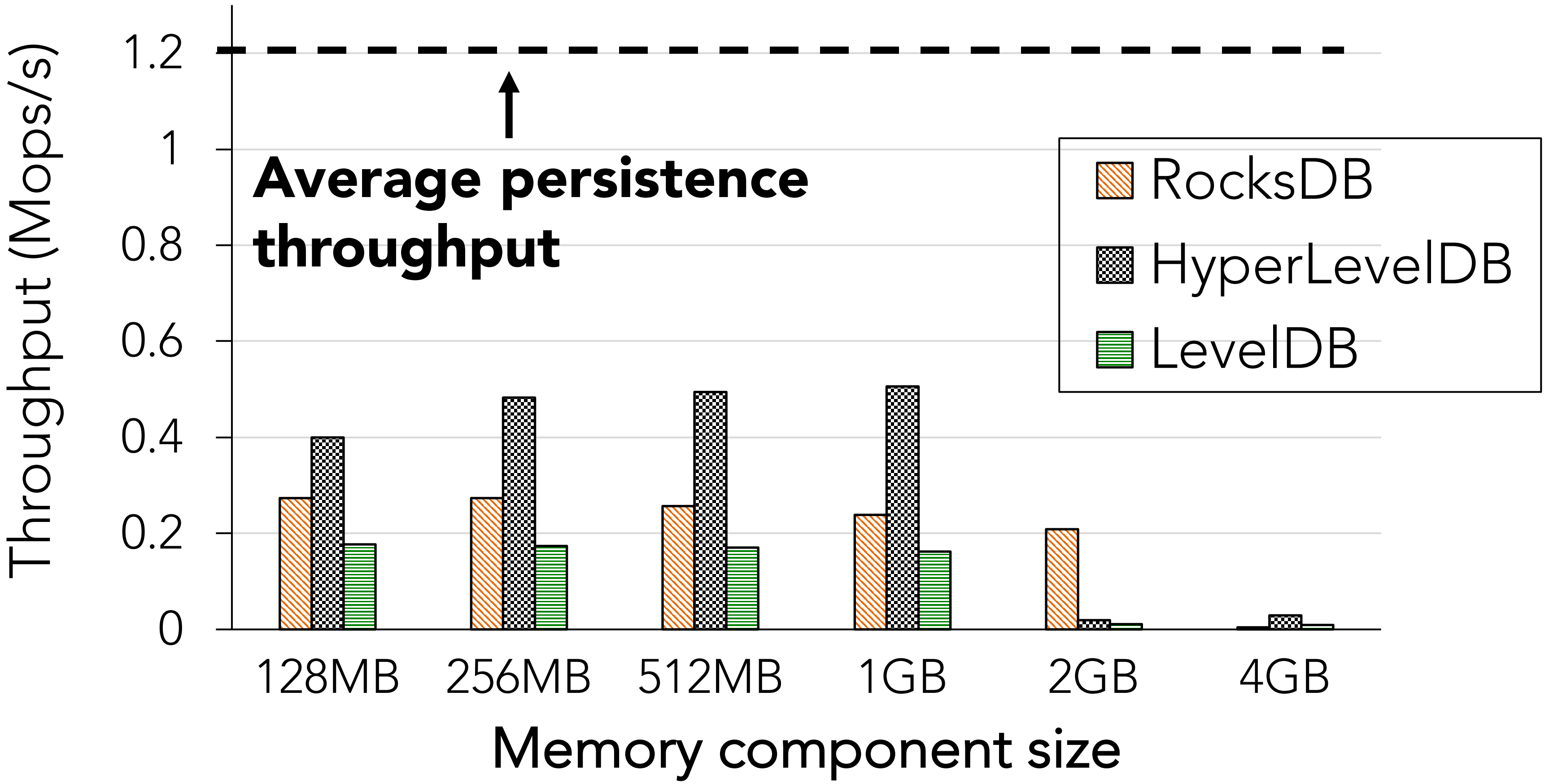
   Need to keep elements sorted (expensive) .

Memory Scalability



16 threads; write-only workload.

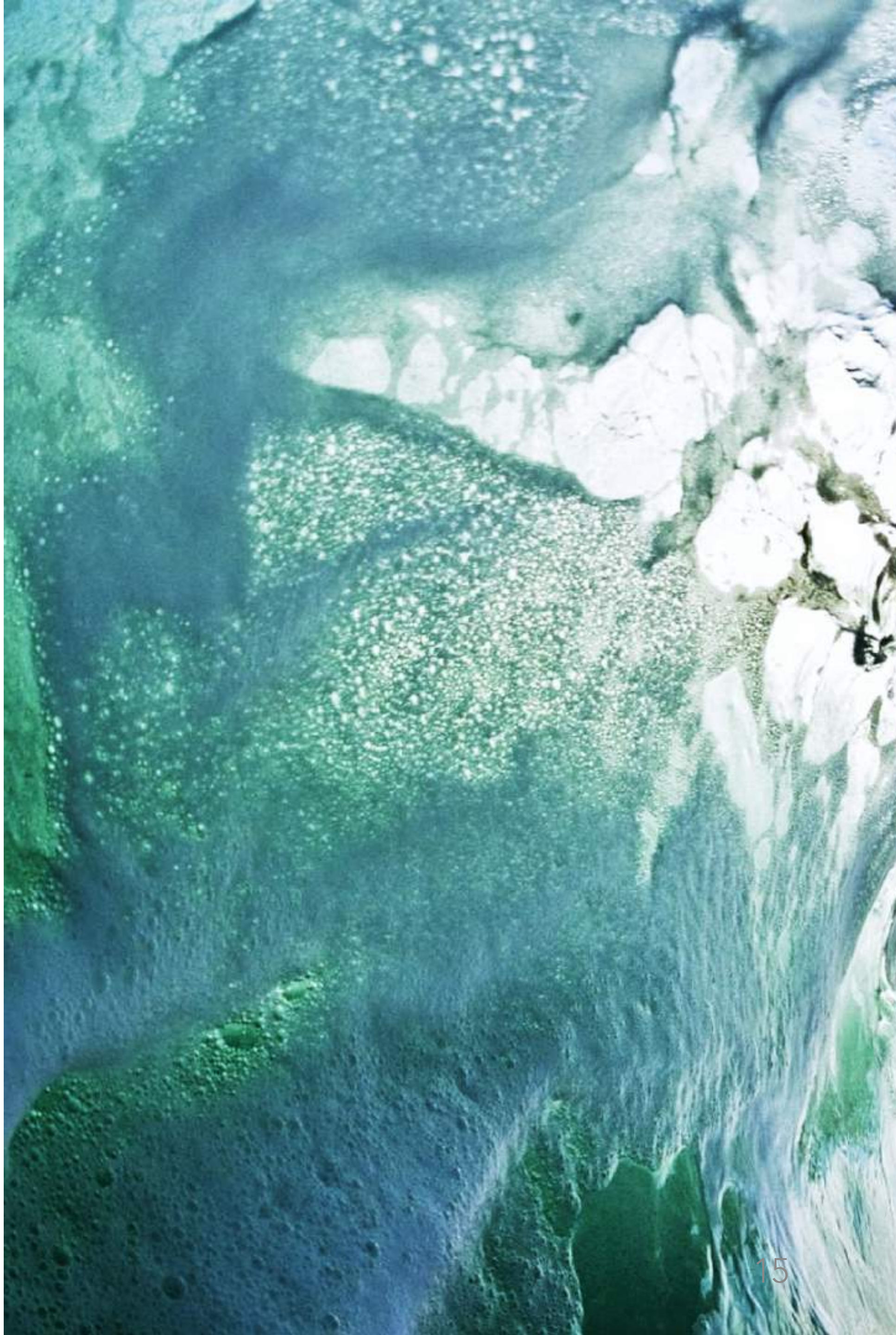
Memory Scalability



16 threads; write-only workload.

Outline

- Motivation
- **FloDB**
- Evaluation
- Conclusion

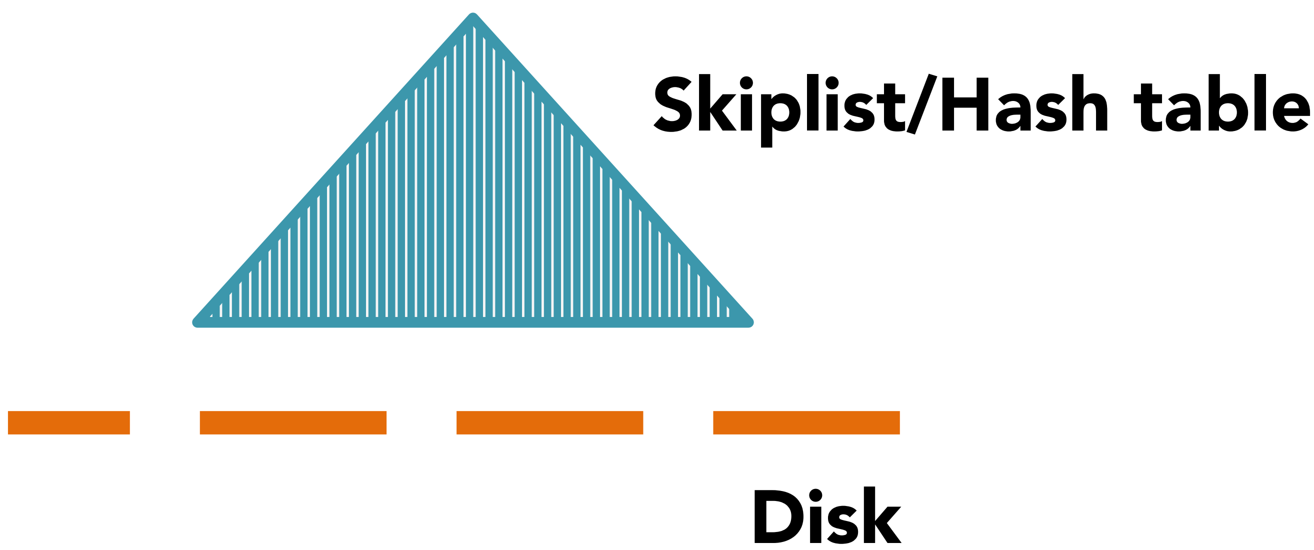


FloDB

New design for LSM memory component

1-Level Mem. Component

Classic LSM



1-Level Mem. Component

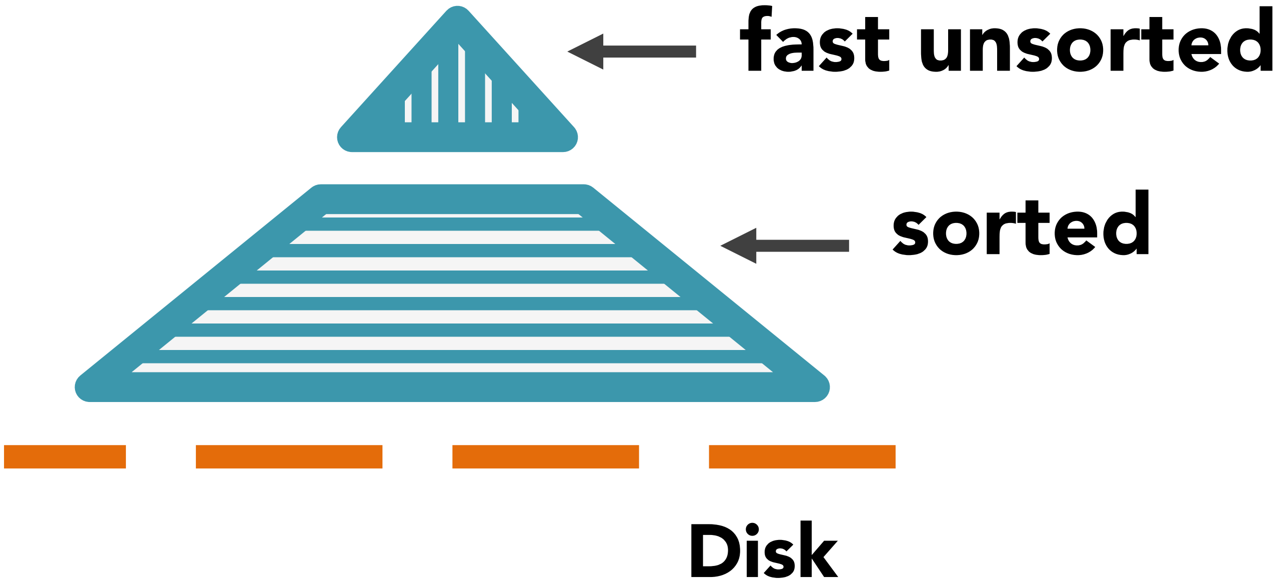
Skiplist

- Already sorted, flush to disk as is.
- O(log n) time to insert elements.

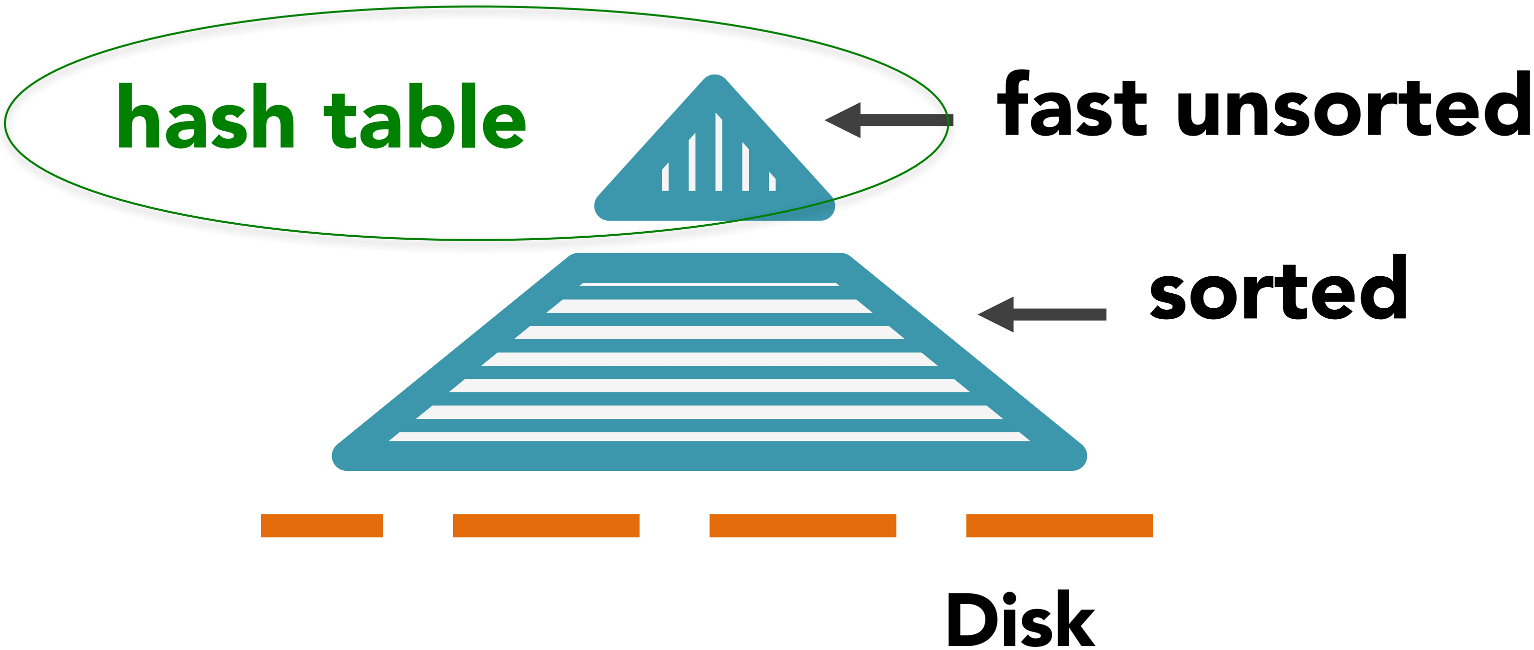
Hash Table

- Sort before writing to disk.
- O(1) updates

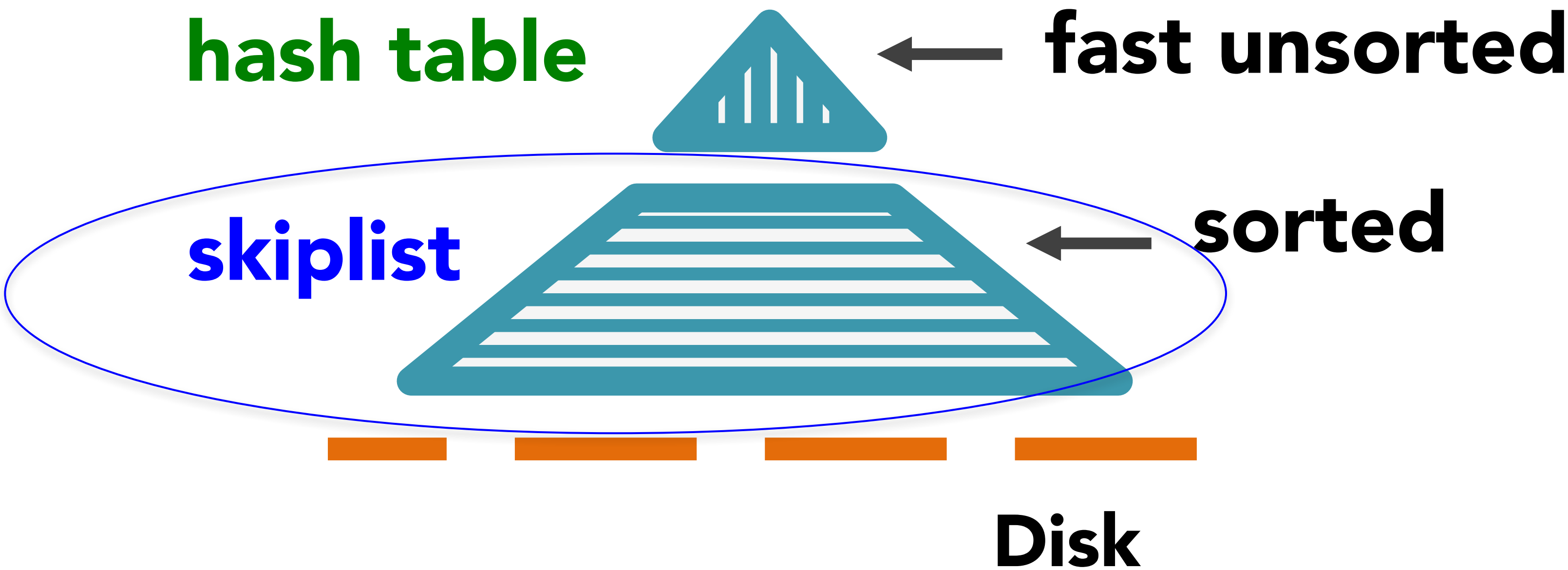
FloDB Structure



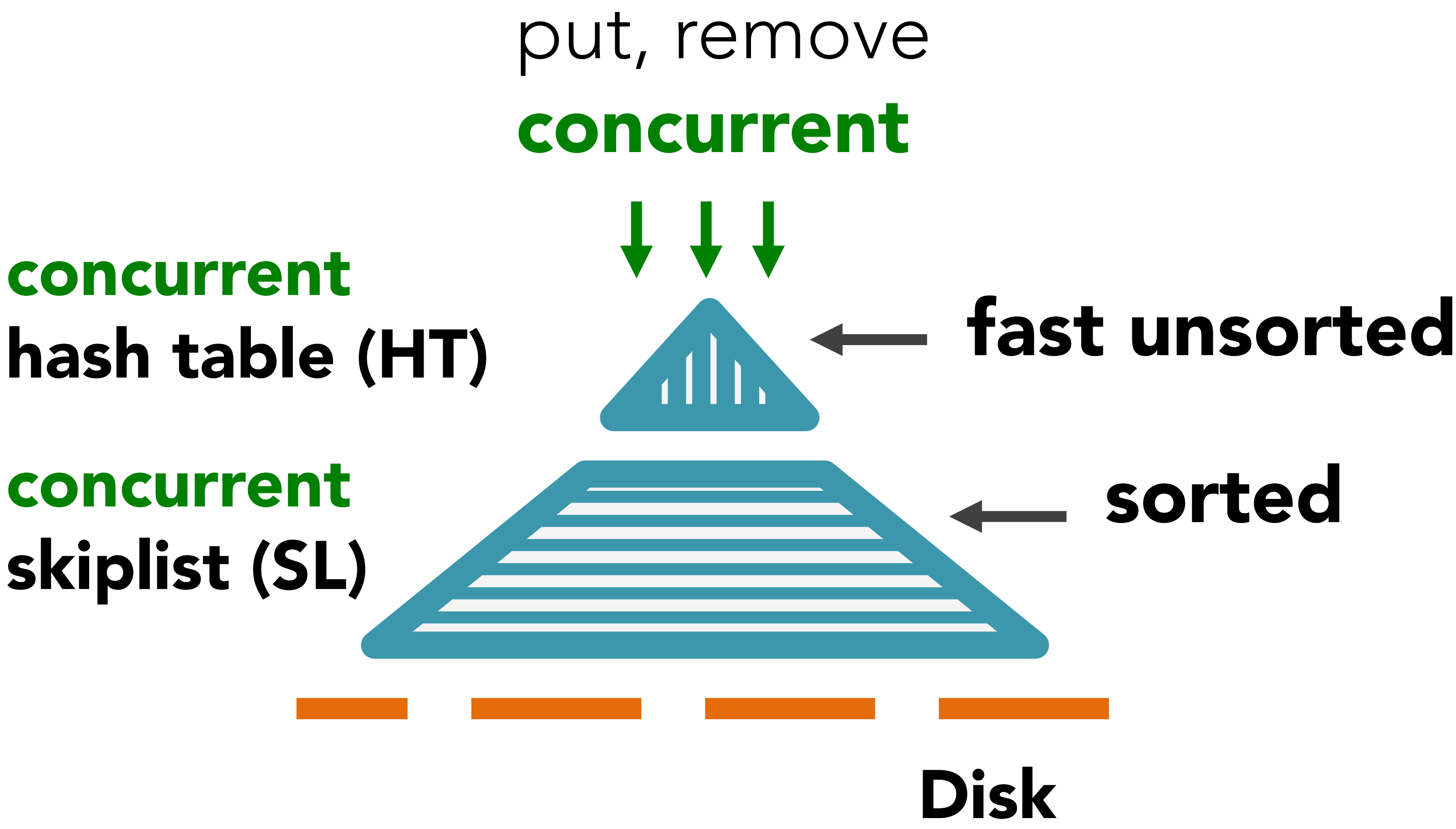
FloDB Structure



FloDB Structure



FloDB Concurrency



FloDB Main Challenge

Ensure efficient **data flow**:

hashtable → skiplist → disk

~100M ops/s ~10M ops/s ~1M ops/s

FloDB Data Flow

Draining

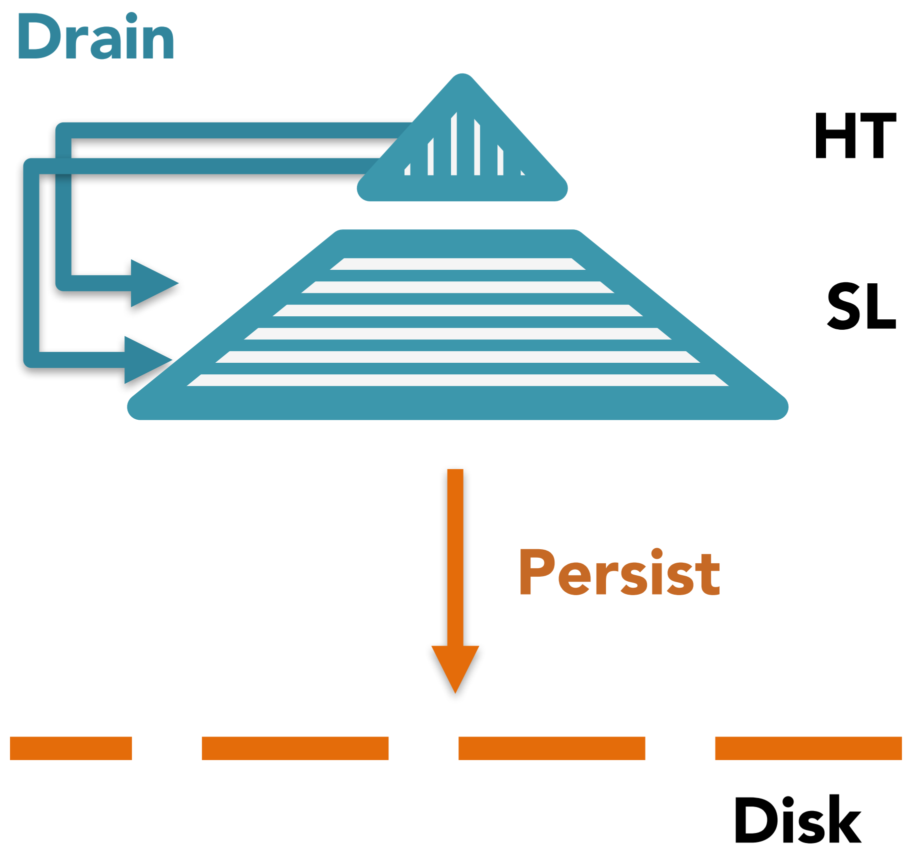
Goal: Keep HT empty

Continuous bg. op

SL Multi-insert

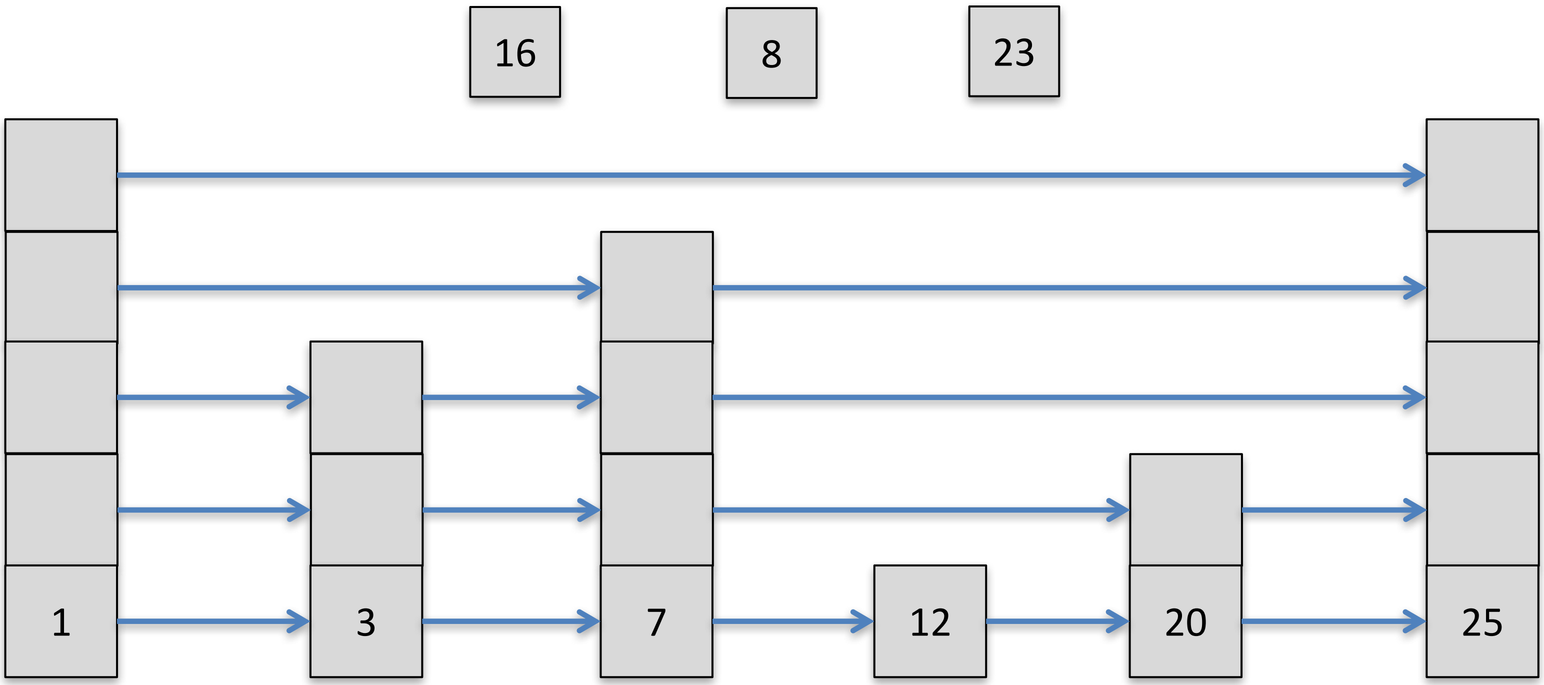
Novel operation

Insert multiple elements in SL at a time



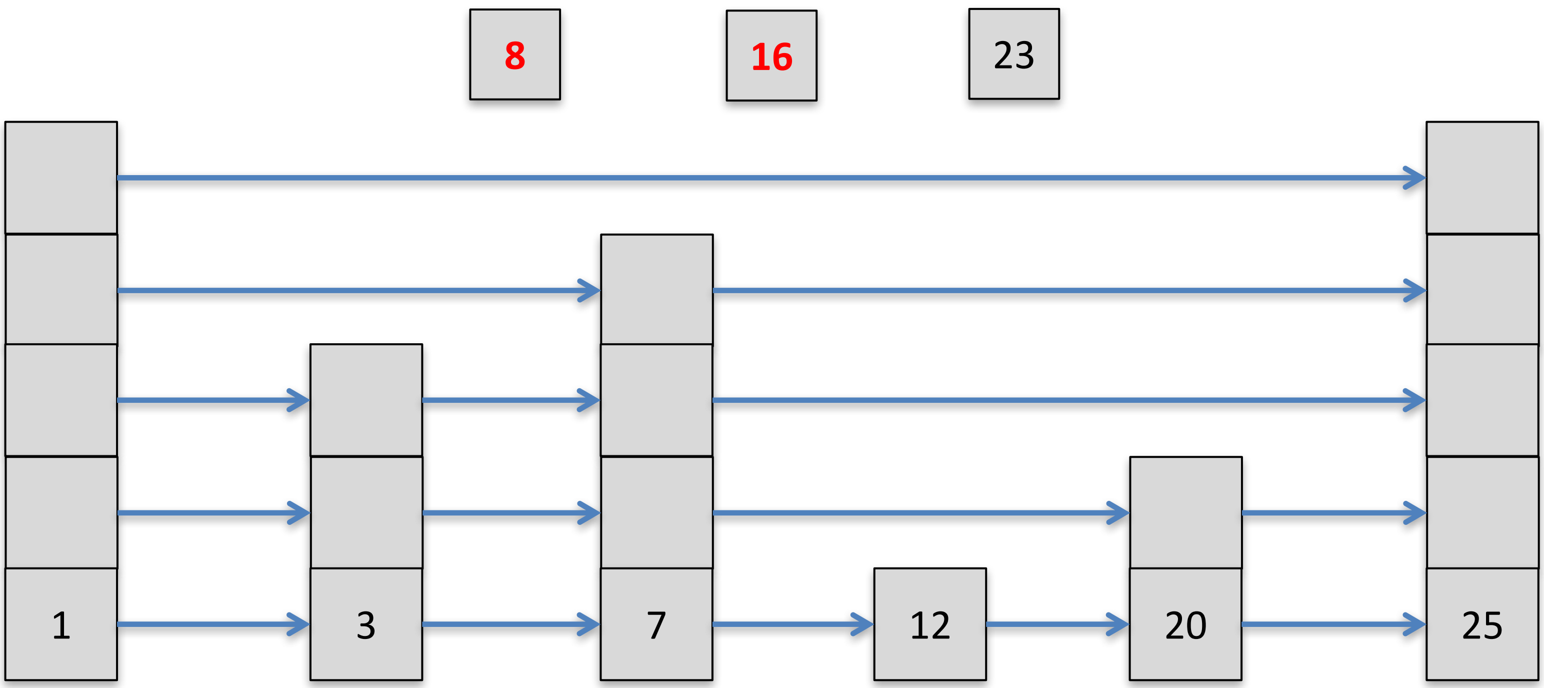
Skiplist Multi-insert

Intuition: **Path sharing.**



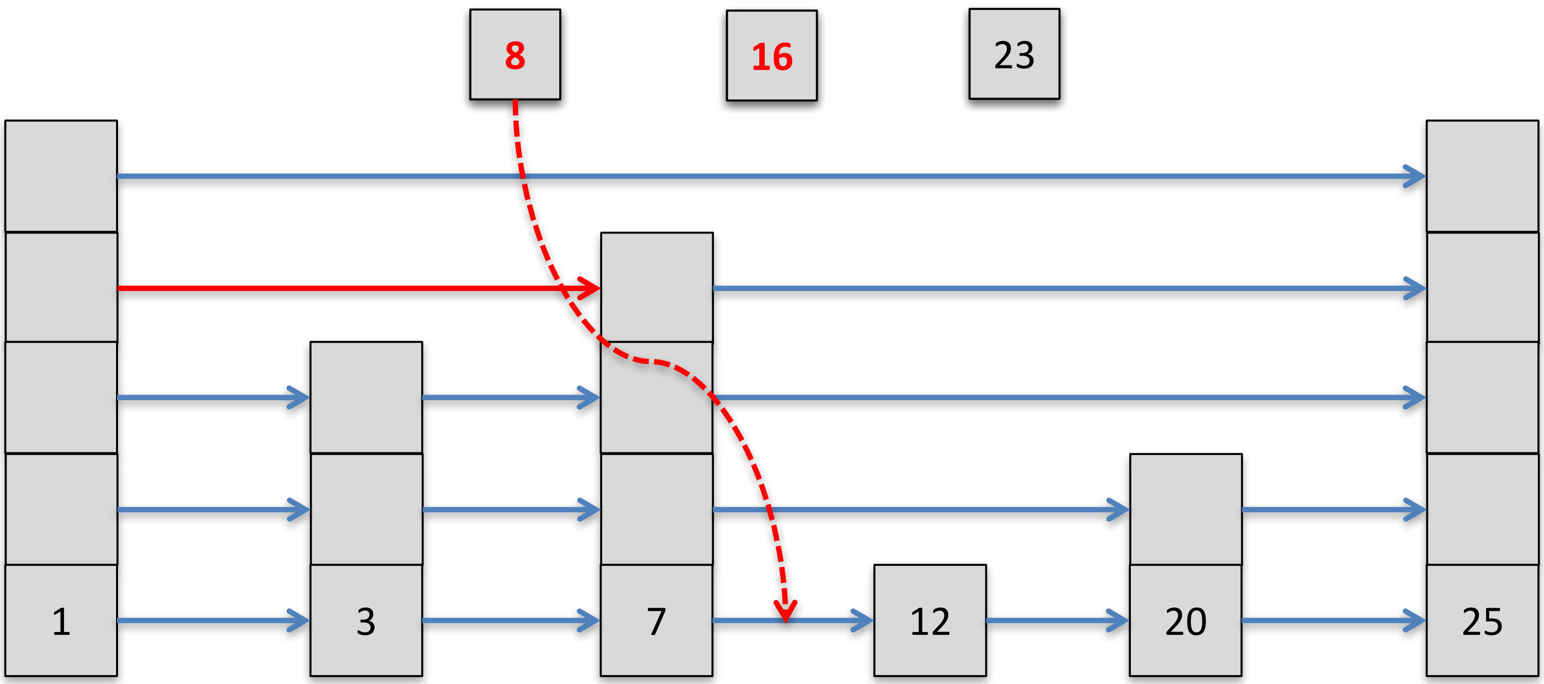
Skiplist Multi-insert

Intuition: **Path sharing.**



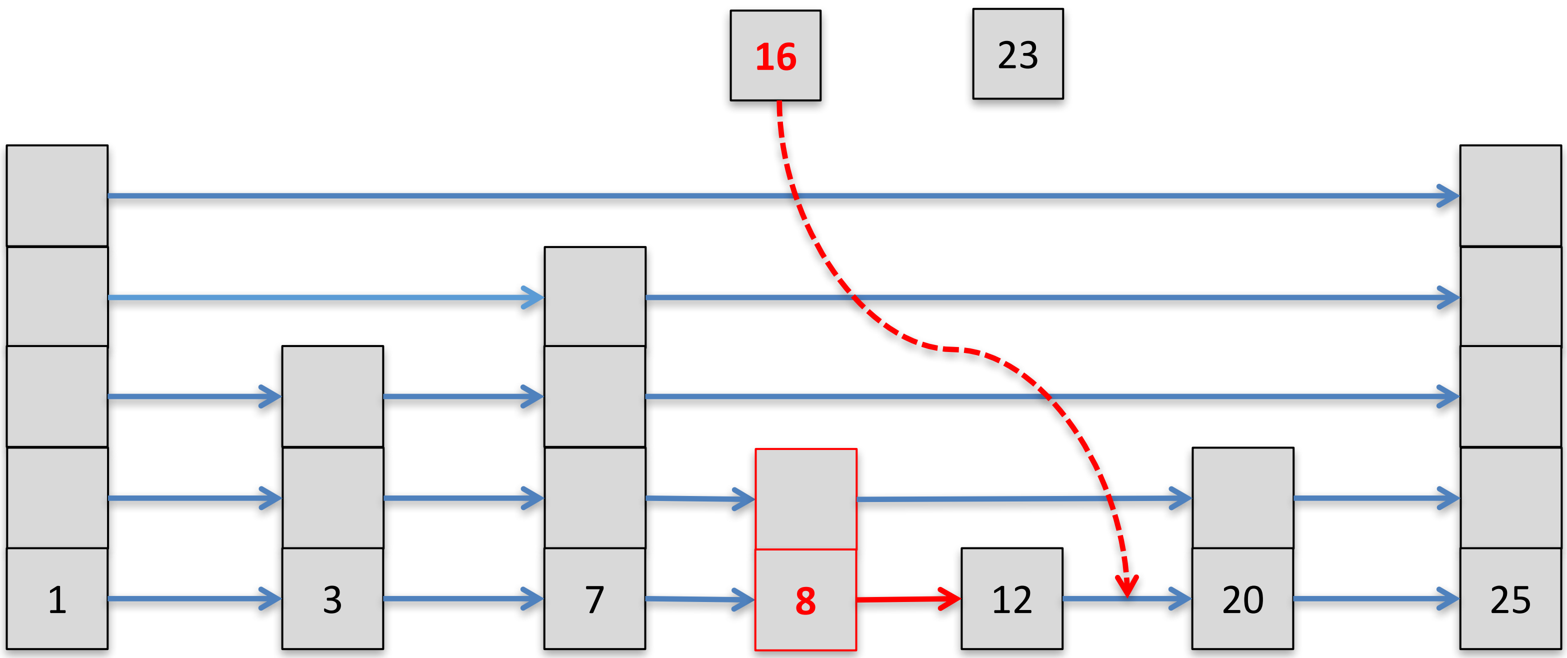
Skiplist Multi-insert

Intuition: **Path sharing.**



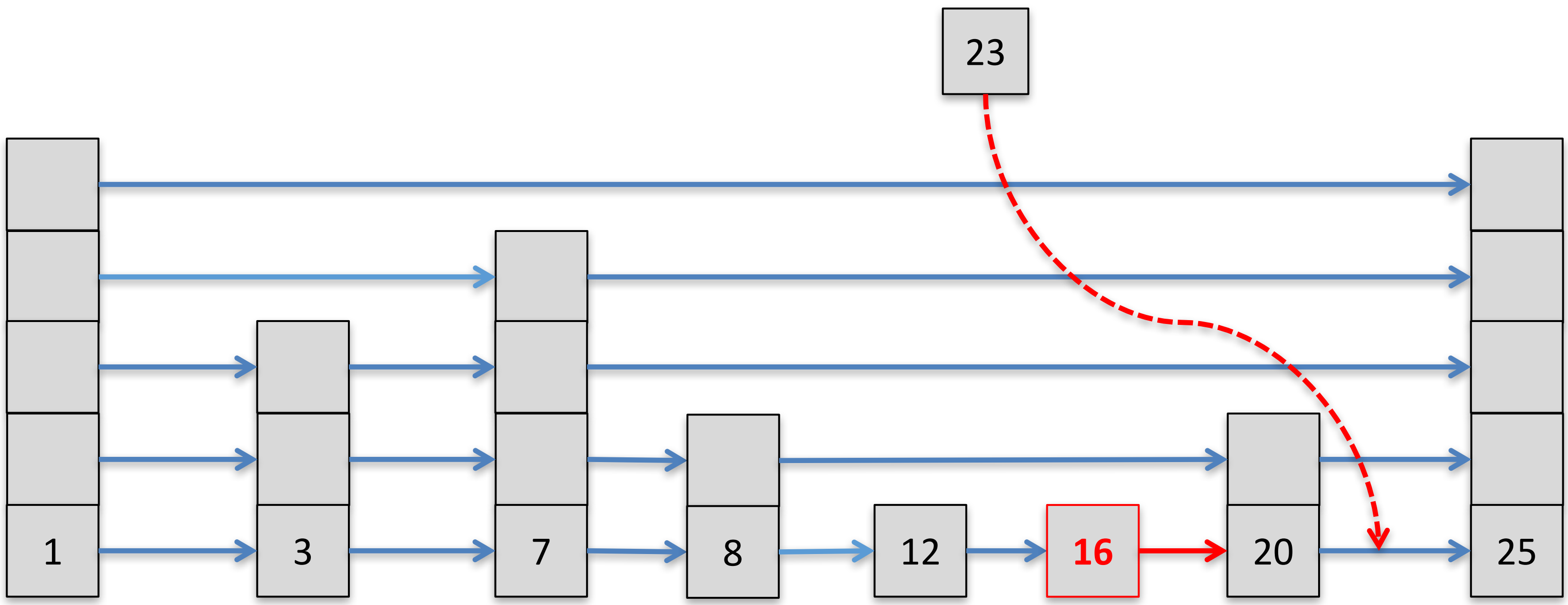
Skiplist Multi-insert

Intuition: **Path sharing.**



Skiplist Multi-insert

Intuition: **Path sharing.**



FloDB Tradeoffs

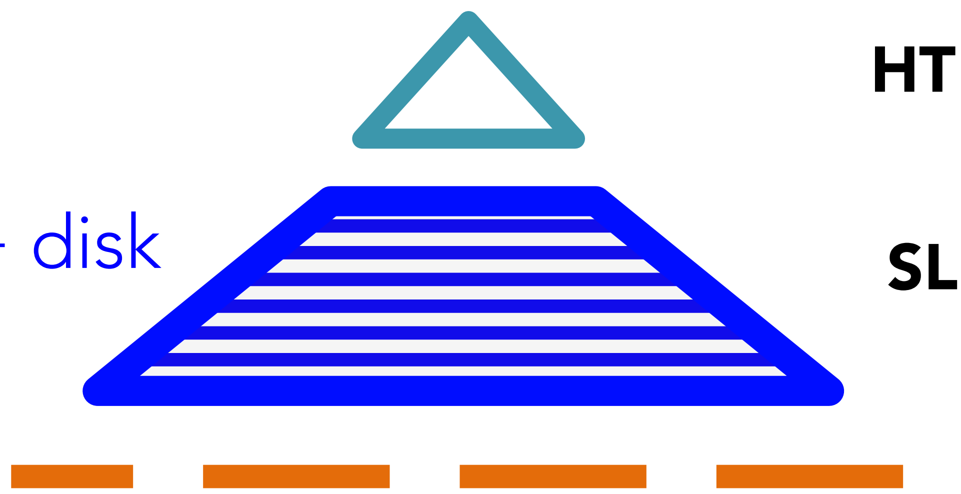
Scans + In-place updates

Drain HT completely

FloDB Tradeoffs

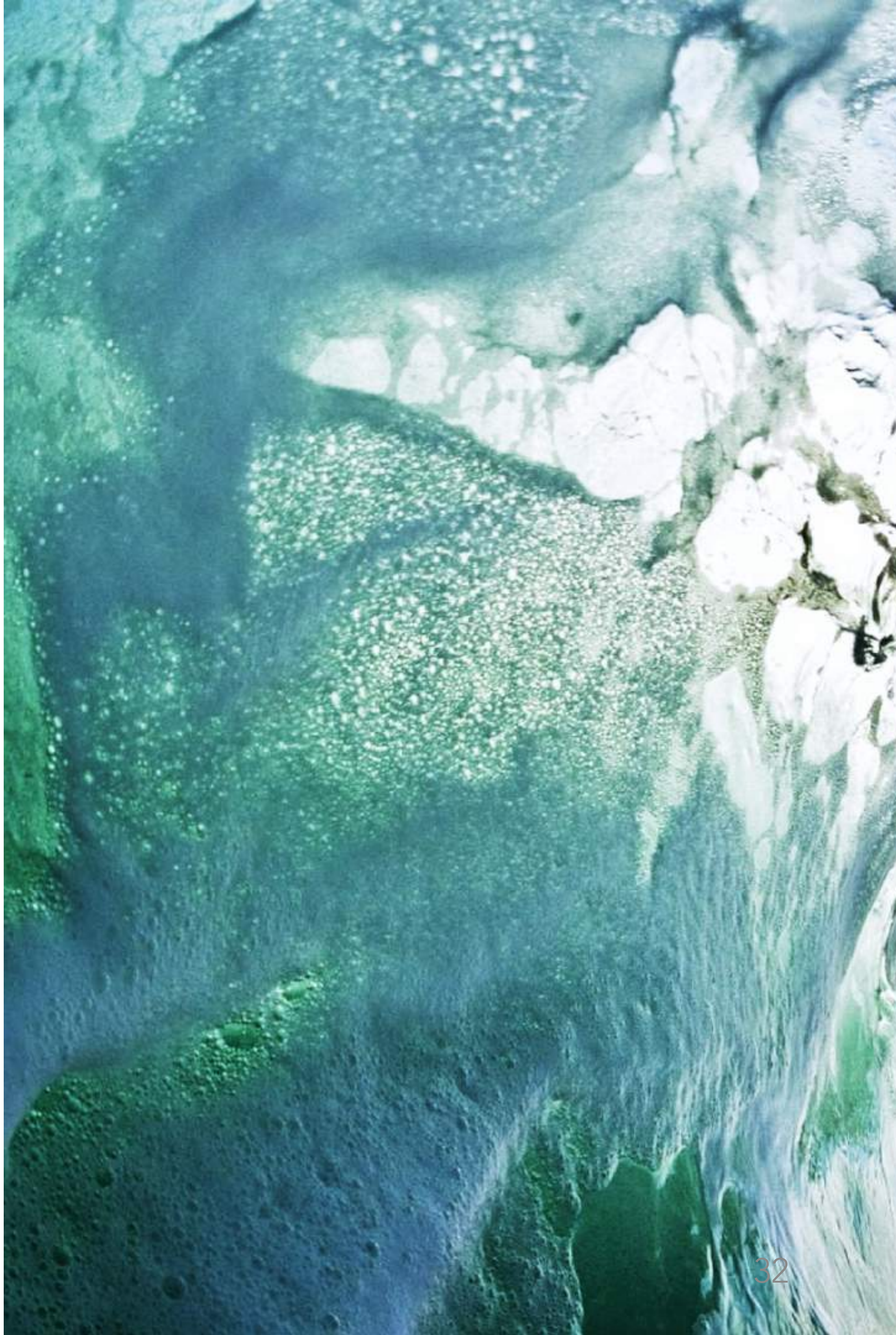
Scans + In-place updates

Scan only SL + disk



Outline

- Motivation
- FloDB
- **Evaluation**
- Conclusion



Evaluation Setup

Code:

FloDB – open source, implemented on top of LevelDB.

http://lpd.epfl.ch/site/flodb

Workloads:

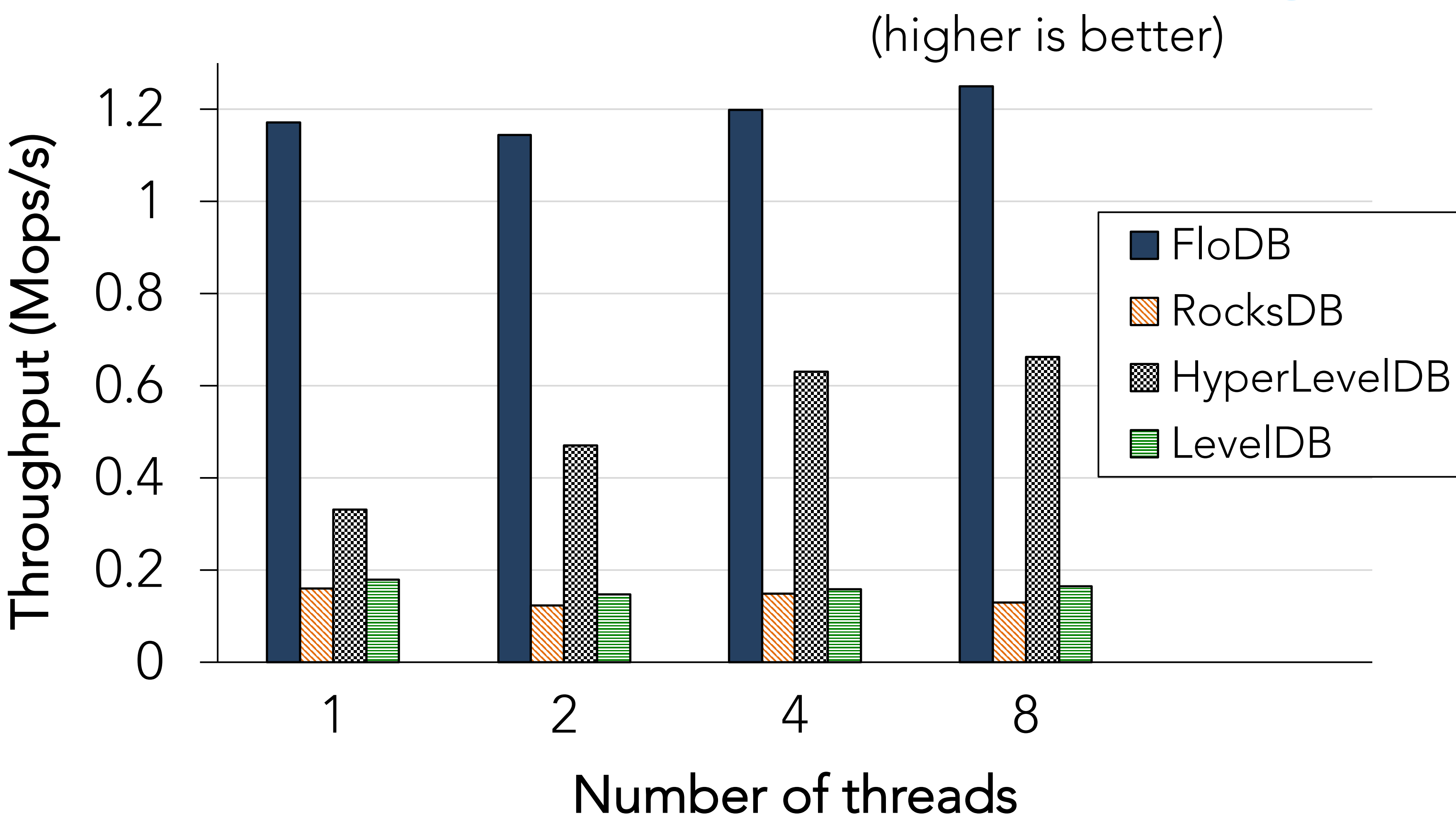
Focus on write-intensive workloads.

Evaluation Setup

Compare FloDB with state-of-the-art LSM KV stores:

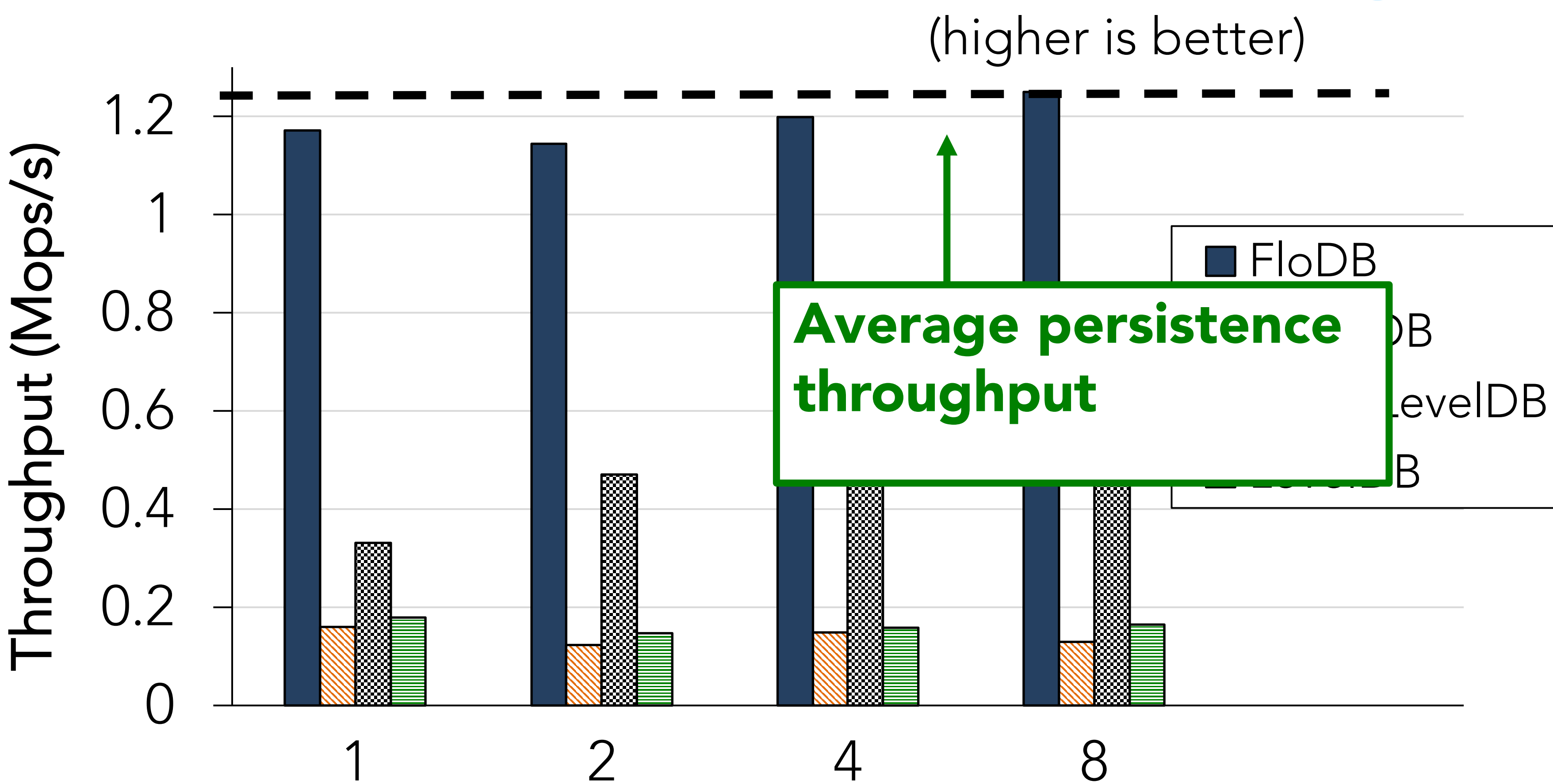
- FloDB
- RocksDB
- HyperLevelDB
- LevelDB

Steady-state Throughput
Write-only workload



Mem. component size 128MB

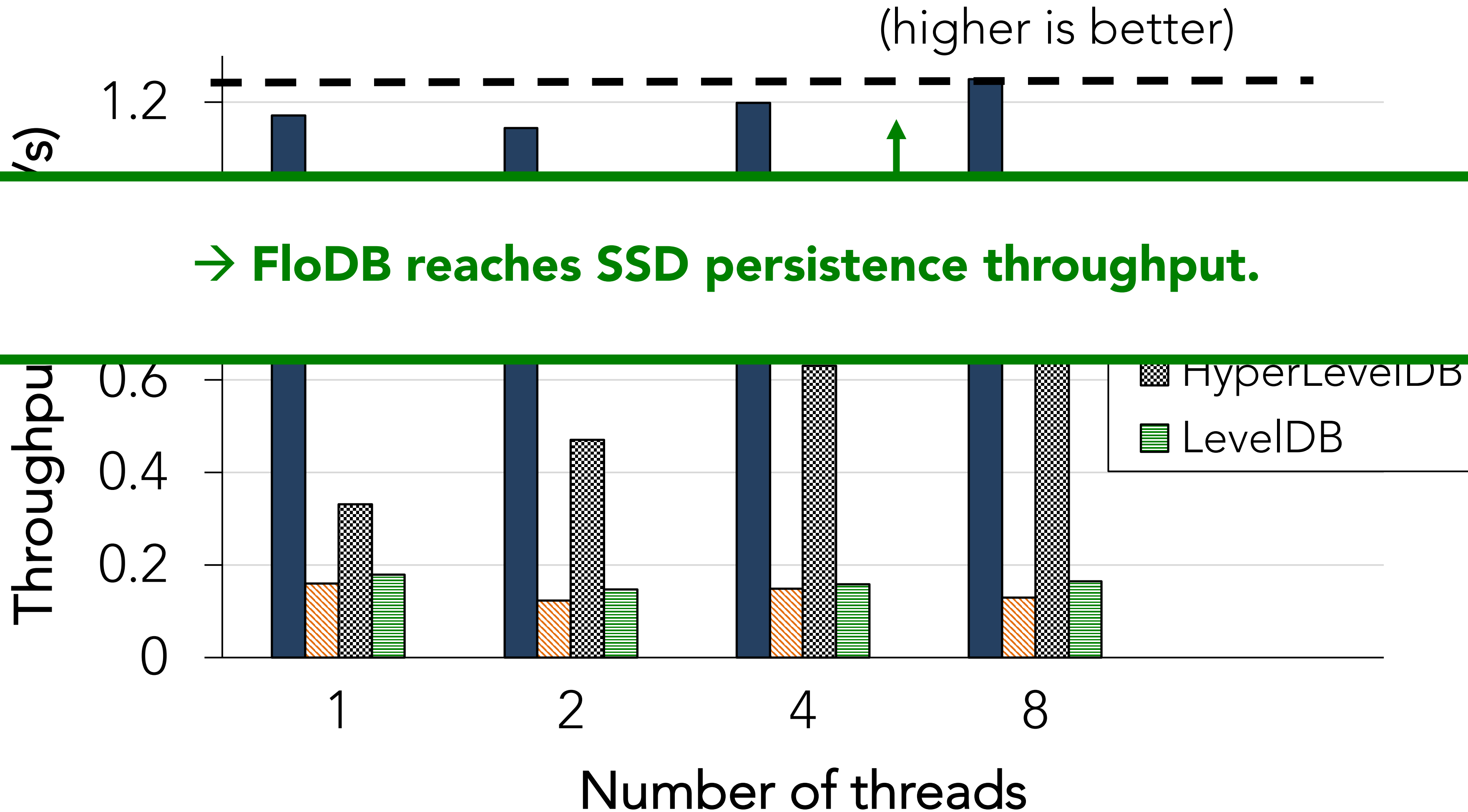
Steady-state Throughput
Write-only workload



Number of threads

Mem. component size 128MB

Steady-state Throughput
Write-only workload

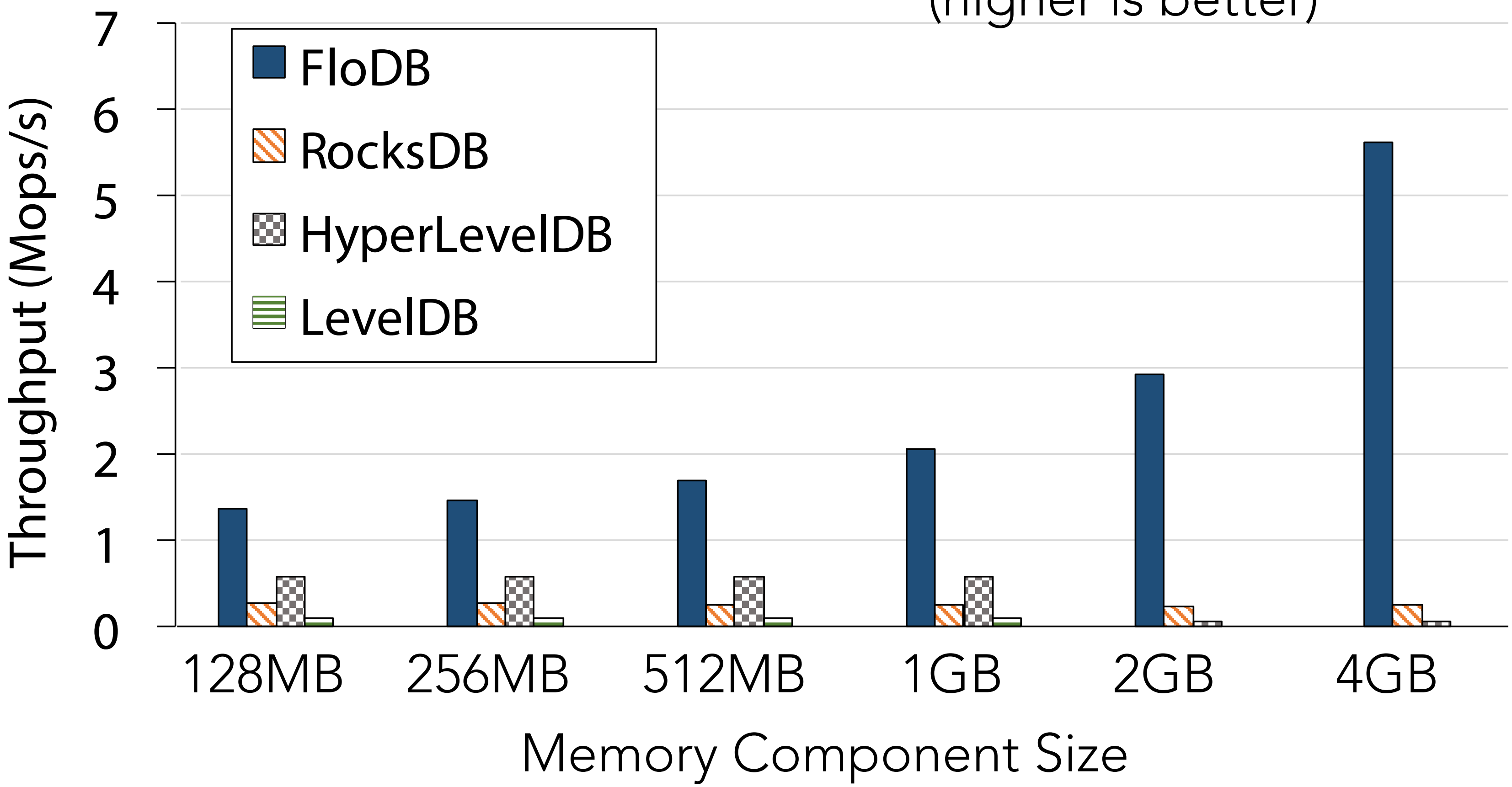


→ FloDB reaches SSD persistence throughput.

Mem. component size 128MB

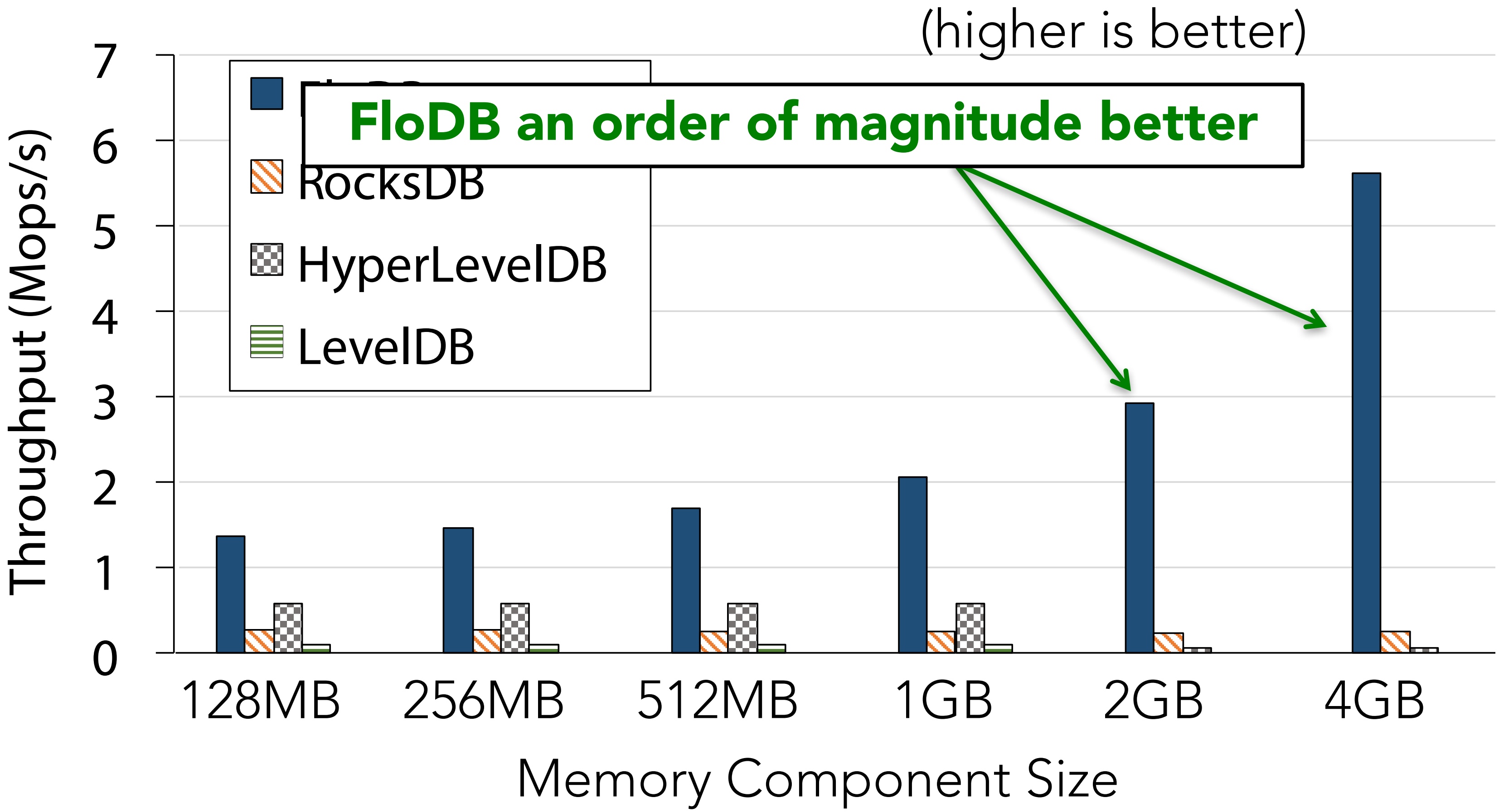
Scalability with Memory Size
Bursts of Writes

(higher is better)



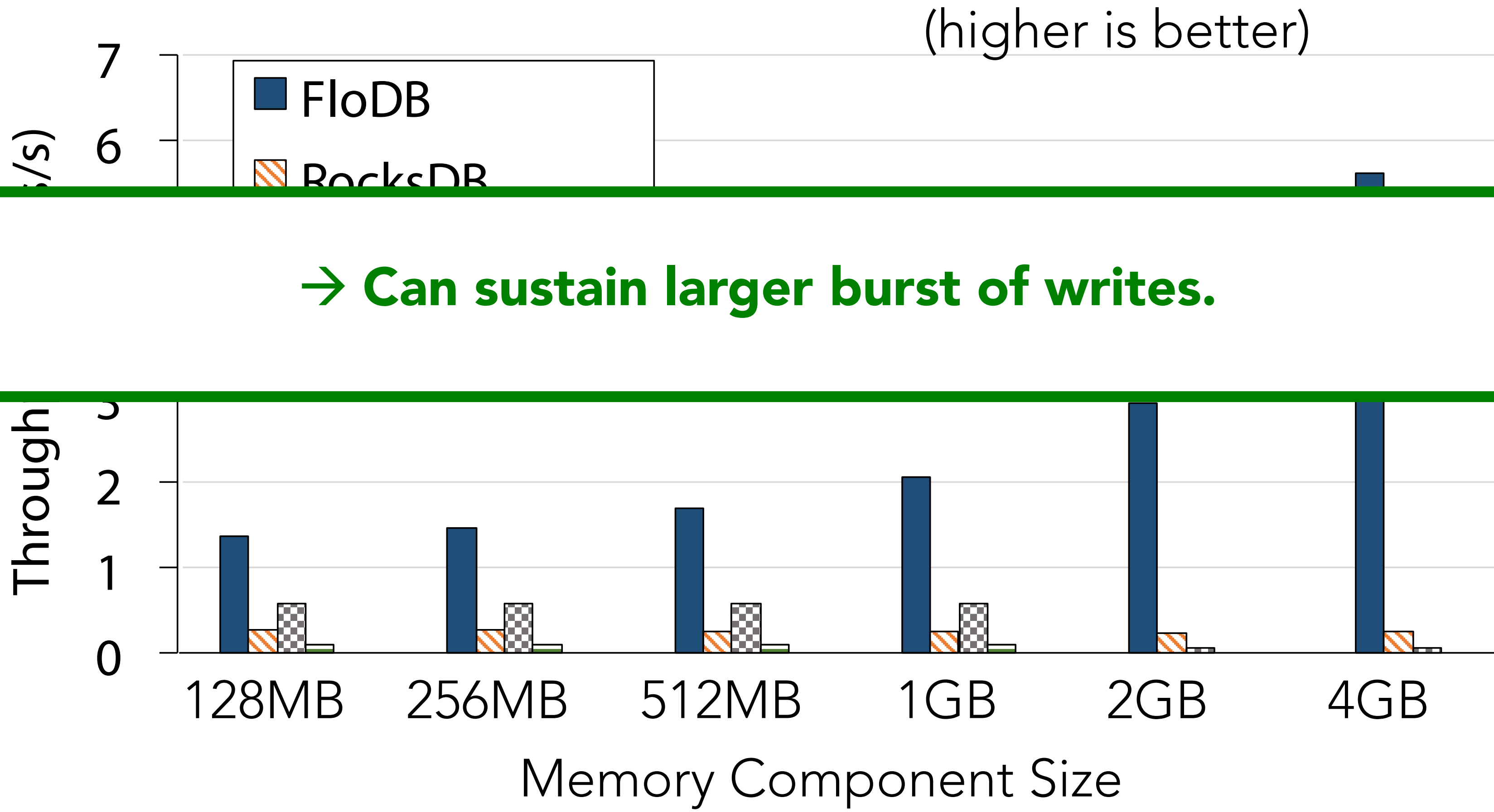
16 threads; FloDB mem. split: ¼ HT, ¾ SL

Scalability with Memory Size
Bursts of Writes



16 threads; FloDB mem. split: ¼ HT, ¾ SL

Scalability with Memory Size
Bursts of Writes



→ **Can sustain larger burst of writes.**

16 threads; FloDB mem. split: ¼ HT, ¾ SL

Related work

- RocksDB, HyperLevelDB.
  - *Better concurrency by reducing size and number of critical sections.*
- cLSM (EuroSys ‘15)
  - *based on LevelDB. Design goal to increase thread scalability.*
- LSM disk-component:
  - bLSM (SIGMOD/PODS ‘12), HyperLevelDB, LSM-trie (USENIX ATC ‘15), VT-tree (FAST ‘13), WiscKey (FAST ’16)
- In-memory KV stores:
  - KiWi (PODC ‘16), Masstree (EuroSys ‘12), MemC3 (NSDI ‘13), Memcache (NSDI ‘13), MICA (NSDI ‘14)

More in the paper

- o **Operations implementation**
  - Range scan consistency.
  - In-place updates.
- o **Experiments**
  - Thread scalability.
  - Skewed workloads.
  - Scans.
  - Multi-insert.
  - And more…

FloDB: Unlocking Memory in Persistent Key-Value Stores

Oana Balmau[1], Rachid Guerraoui[1], Vasileios Trigonakis[2] *, and Igor Zablotchi[1] †
[1]EPFL, {first.last}@epfl.ch
[2]Oracle Labs, {first.last}@oracle.com

Abstract

Log-structured merge (LSM) data stores enable to store and process large volumes of data while maintaining good performance. They mitigate the I/O bottleneck by absorbing updates in a memory layer and transferring them to the disk layer in sequential batches. Yet, the LSM architecture fundamentally requires elements to be in sorted order. As the amount of data in memory grows, maintaining this sorted order becomes increasingly costly. Contrary to intuition, existing LSM systems could actually lose throughput with larger memory components.

In this paper, we introduce FloDB, an LSM memory component architecture which allows throughput to scale on modern multicore machines with ample memory sizes. The main idea underlying FloDB is essentially to bootstrap the traditional LSM architecture by adding a small in-memory buffer layer on top of the memory component. This buffer offers low-latency operations, masking the write latency of the sorted memory component. Integrating this buffer in the classic LSM memory component to obtain FloDB is not trivial and requires revisiting the algorithms of the user-facing LSM operations (search, update, scan). FloDB's two layers can be implemented with state-of-the-art, highly-concurrent data structures. This way, as we show in the paper, FloDB eliminates significant synchronization bottlenecks in classic LSM designs, while offering a rich LSM API.

We implement FloDB as an extension of LevelDB, Google's popular LSM key-value store. We compare FloDB's performance to that of state-of-the-art LSMs. In short, FloDB's performance is up to one order of magnitude higher than that of the next best-performing competitor in a wide range of multi-threaded workloads.

* The project was completed while the author was at EPFL.
† Authors appear in alphabetical order.

Figure 1: LSM data store using FloDB.

1. Introduction

Key-value stores are a crucial component of many systems that require low-latency access to large volumes of data [1, 6, 12, 15, 16]. These stores are characterized by a flat data organization and simplified interface, which allow for efficient implementations. However, the amount of data targeted by key-value stores is usually larger than main memory; thus, persistent storage is generally required. Since accessing persistent storage is slow compared to CPU speed [41, 42], updating data directly on disk yields a severe bottleneck.

To address this challenge, many key-value stores adopt the log-structured merge (LSM) architecture [35, 36]. Examples include LevelDB [5], RocksDB [12], cLSM [26], bLSM [39], HyperLevelDB [6] and HBase [11]. LSM data stores are suitable for applications that require low latency accesses, such as message queues that undergo a high number of updates, and for maintaining session states in user-facing applications [12]. Basically, the LSM architecture masks the disk access bottleneck, on the one hand, by caching reads and, on the other hand, by absorbing writes in memory and writing to disk in batches at a later time. Although LSM key-value stores go a long way addressing the challenge posed by the I/O bottleneck, their performance does not however scale with the size of the in-memory component, nor does it scale up with the number of threads. In other words, and maybe surprisingly, increasing the in-memory parts of existing LSMs only benefits performance up to a relatively small size. Similarly, adding threads does not improve the throughput of many existing LSMs, due to their use of global blocking synchronization.

As we discuss in this paper, the two aforementioned scalability limitations are inherent to the design of traditional

Key Takeaways

- **FloDB – novel two-level memory component for LSM.**

Key Takeaways

- FloDB – novel two-level memory component for LSM.
- Novel multi-insert operation for concurrent skiplists.

Key Takeaways

- **FloDB – novel two-level memory component for LSM.**
- **Novel multi-insert operation for concurrent skiplists.**
- **Scales with memory size and with threads.**

Thank you! Questions?