Benchmarking Learned and LSM Indexes for Data Sortedness

Aneesh Raman

Andy Huynh

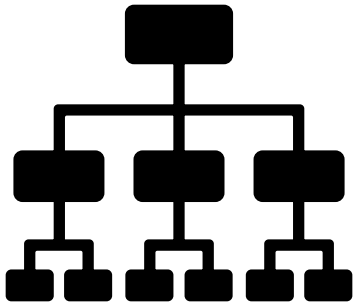
Jinqi Lu

Manos Athanassoulis

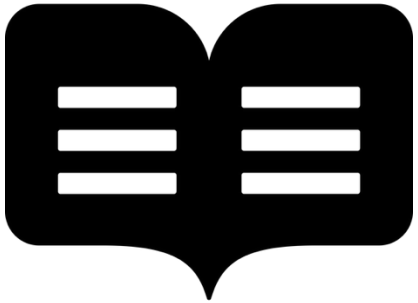
BOSTON UNIVERSITY

DiSC lab

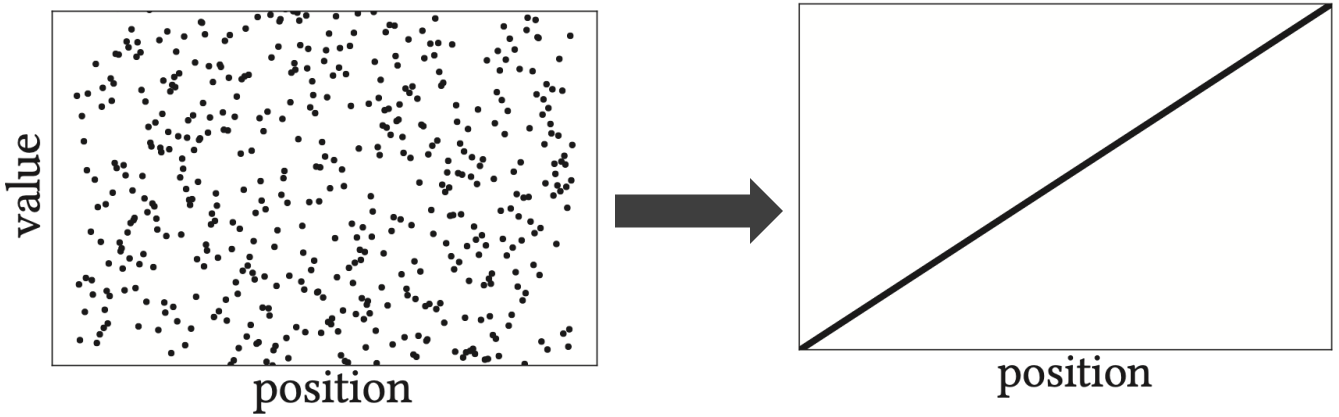
Indexes in Databases



organize data



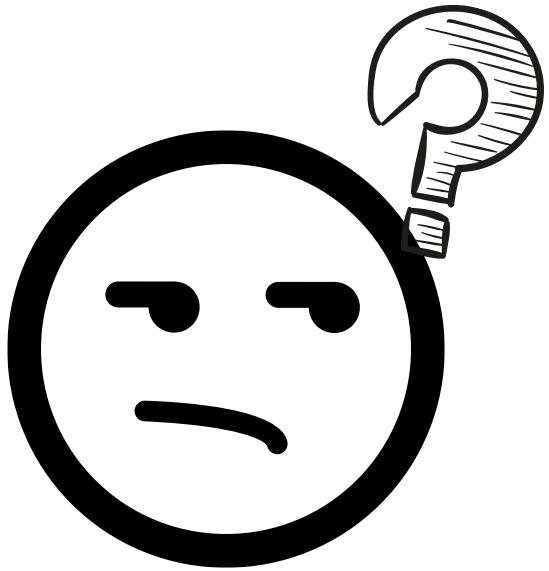
efficient queries



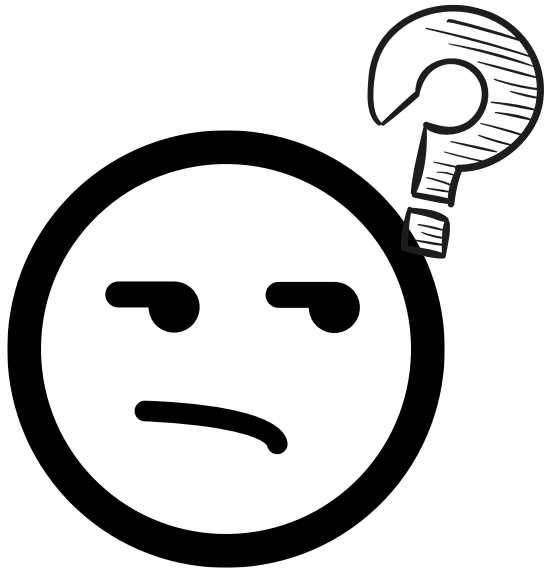
unstructured data

structured data

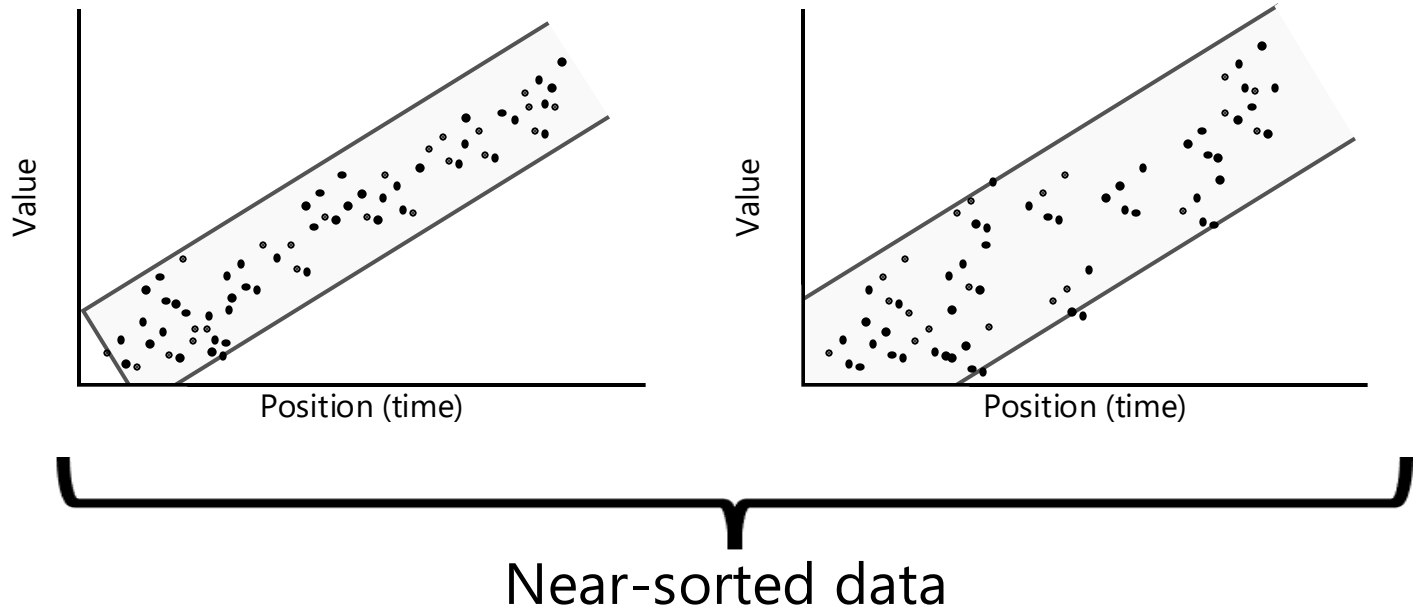
The process of inducing *sortedness* to an otherwise unsorted data collection

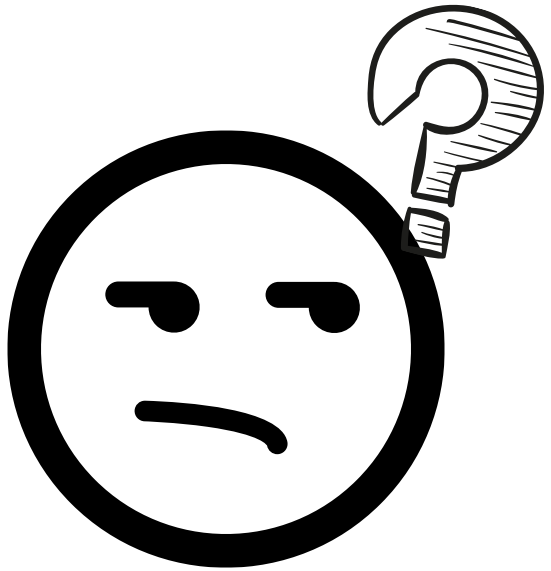


What if data already has some structure?

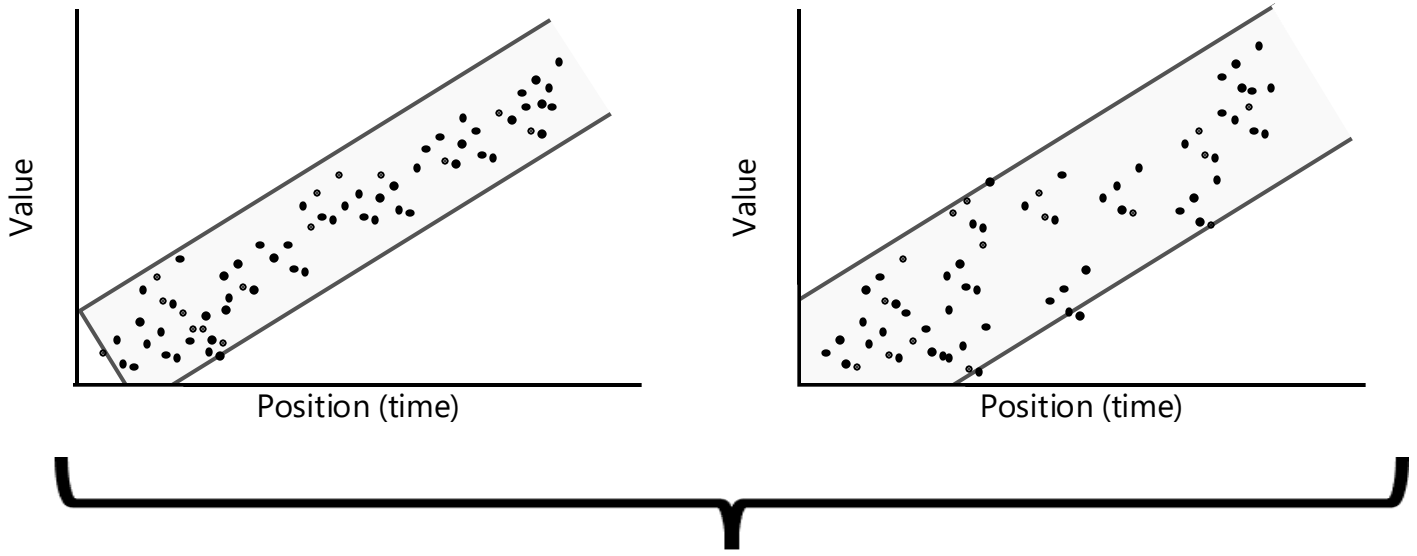


What if data already has some structure?



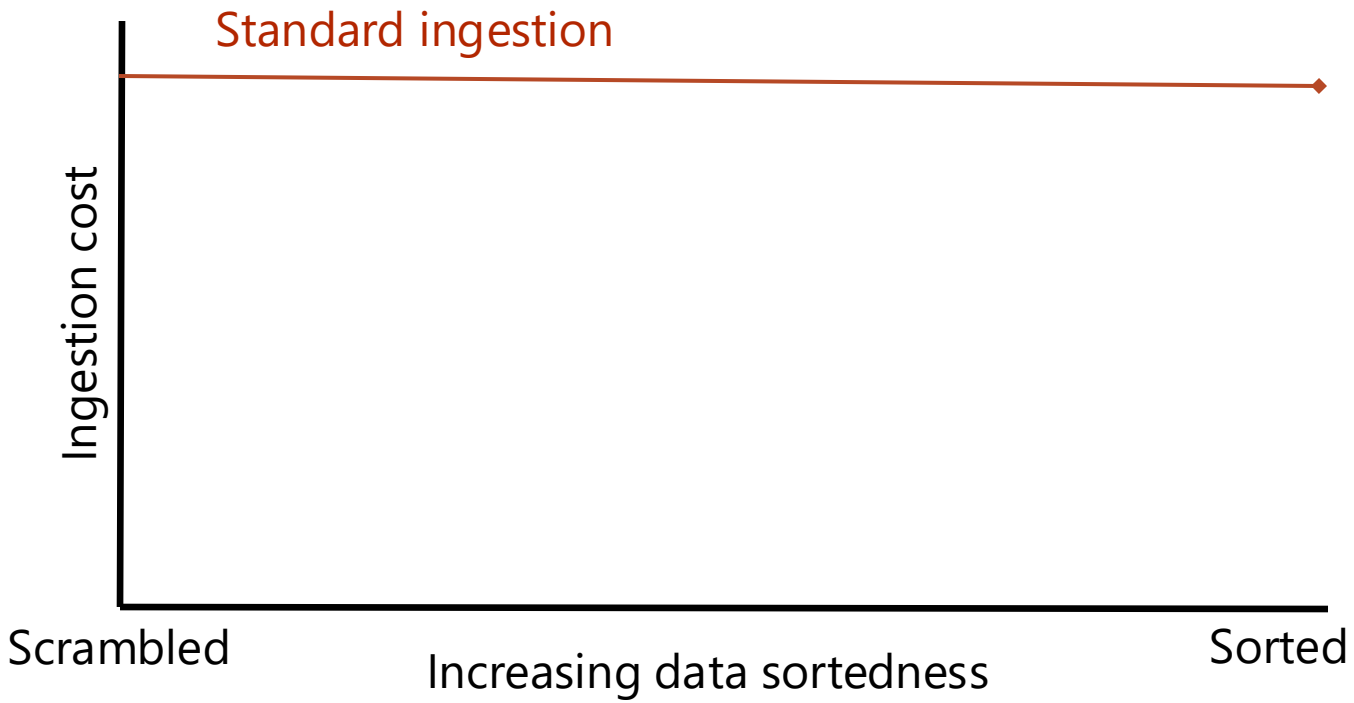


What if data already has some structure?

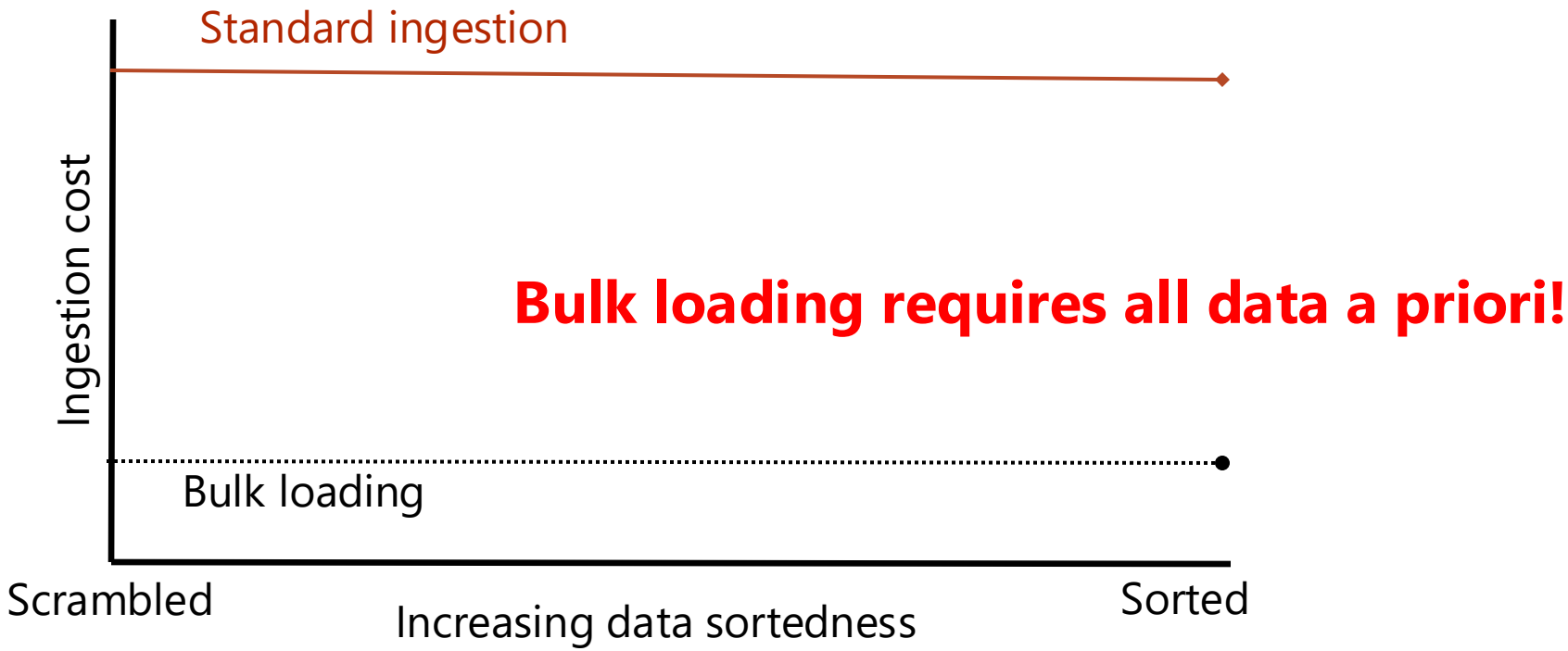


≈ treated same as unstructured data!

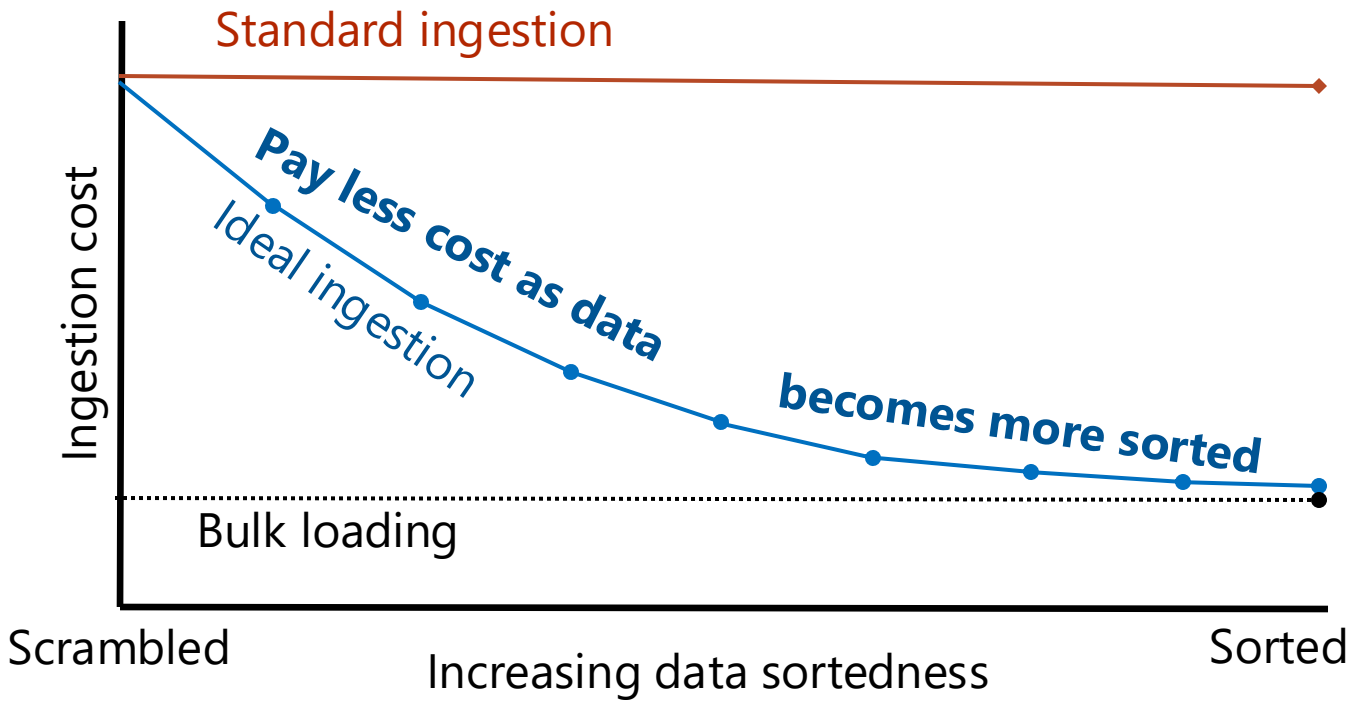
Irrespective of Sortedness, Same Ingestion Performance



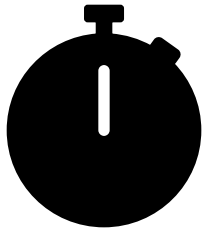
Are There Faster Alternatives?



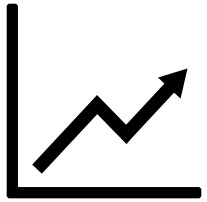
Ideally, Higher Sortedness Should Lead to Faster Ingestion



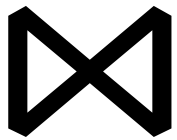
Near-Sorted Data is Frequently Found



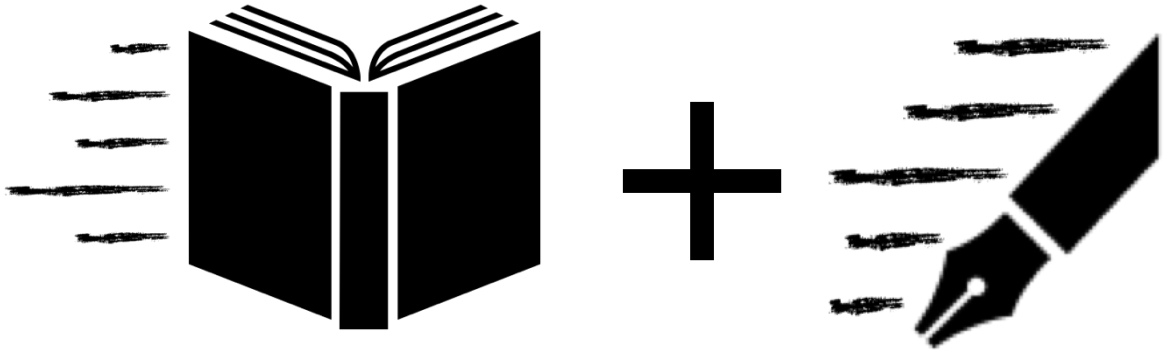
Time Series



Stock market



Join/query

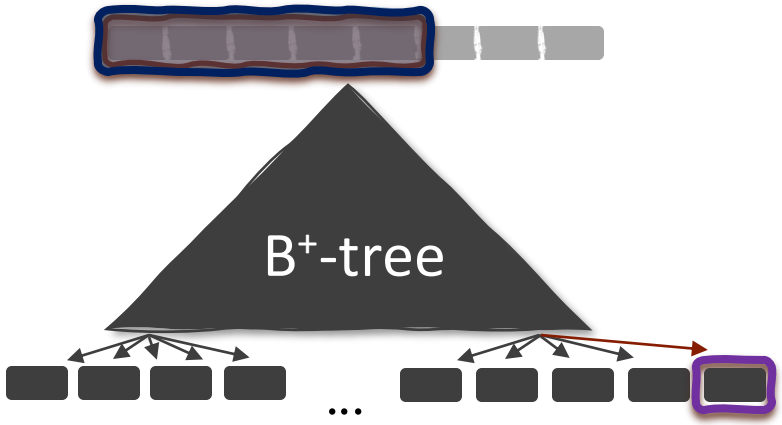


efficient reads + fast writes

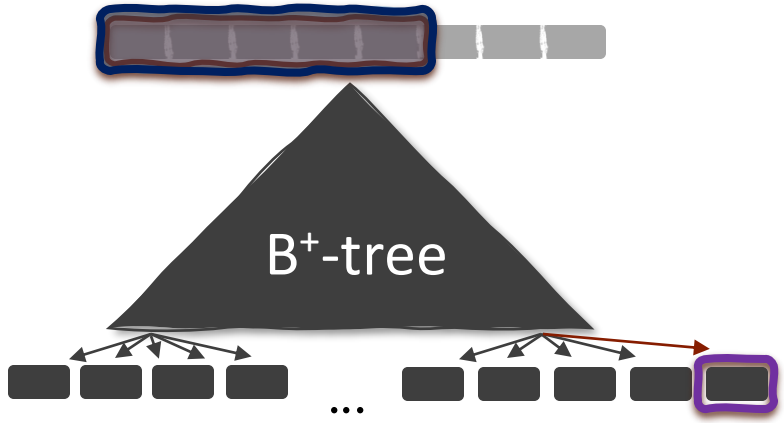
classical indexes carry ***redundant effort!***

Prior Work Focuses on Classical Indexes

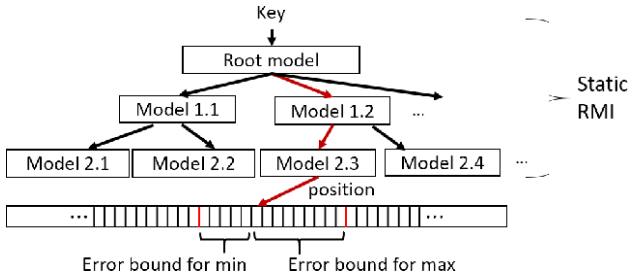
- ✓ In-memory buffering
- ✓ Opportunistic bulk loading
- ✓ Sortedness-adaptiveness
- ✓ Better memory utilization



Prior Work Focuses on Classical Indexes



Other Index Designs?



Agenda

Introduction

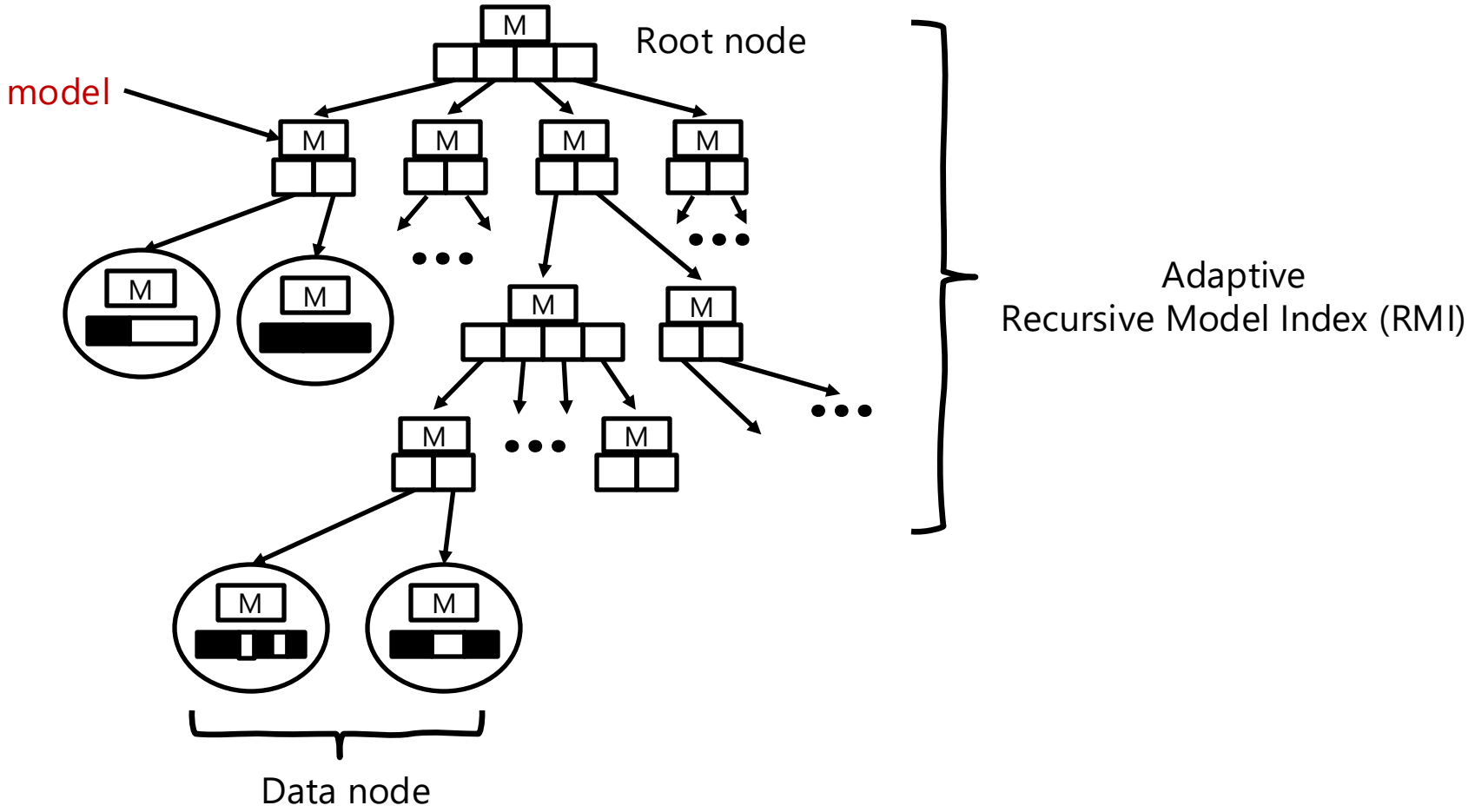
Vision

Background on Index Designs

Sortedness Metrics & Evaluation Framework

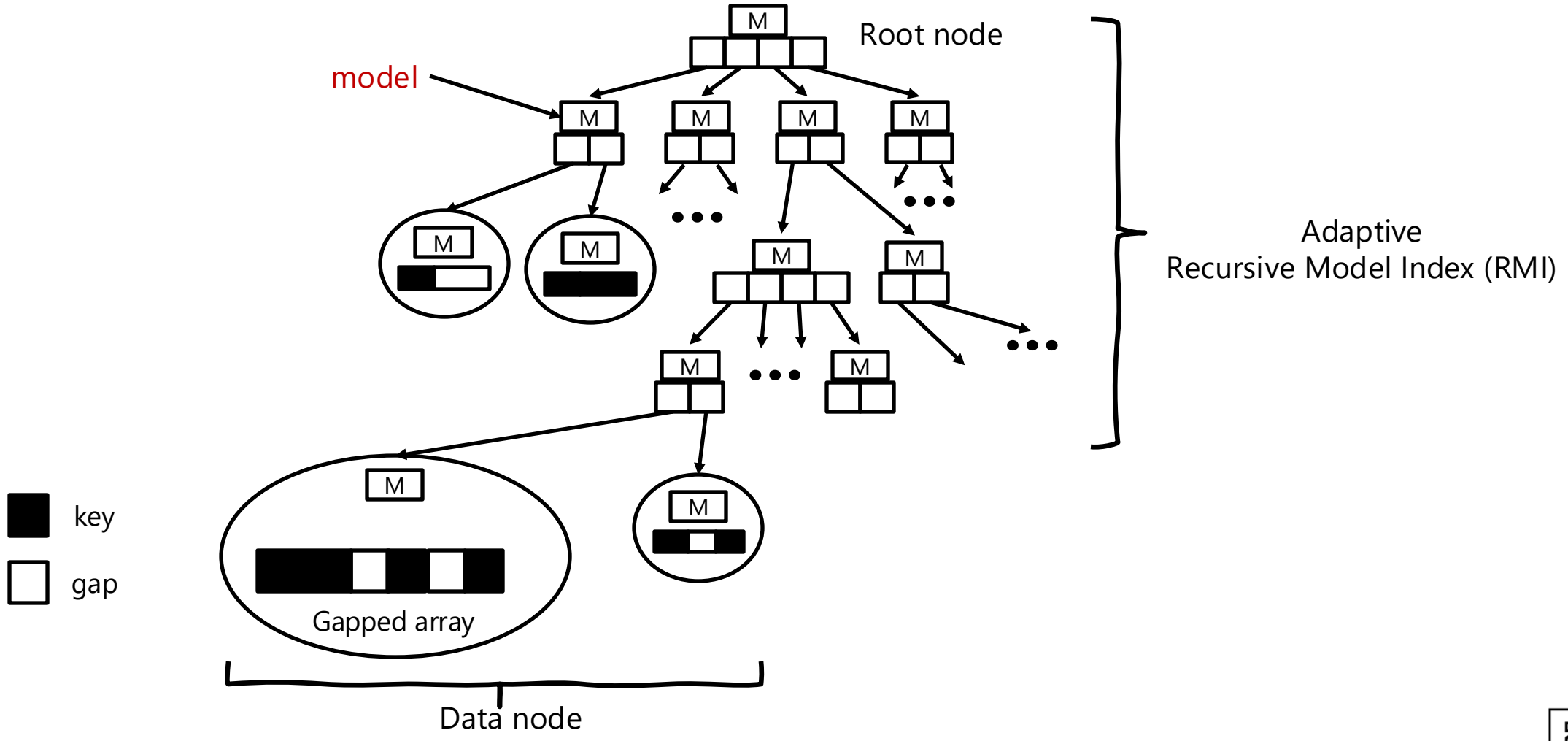
Benchmarking Results

ALEX: An Updatable Learned Index

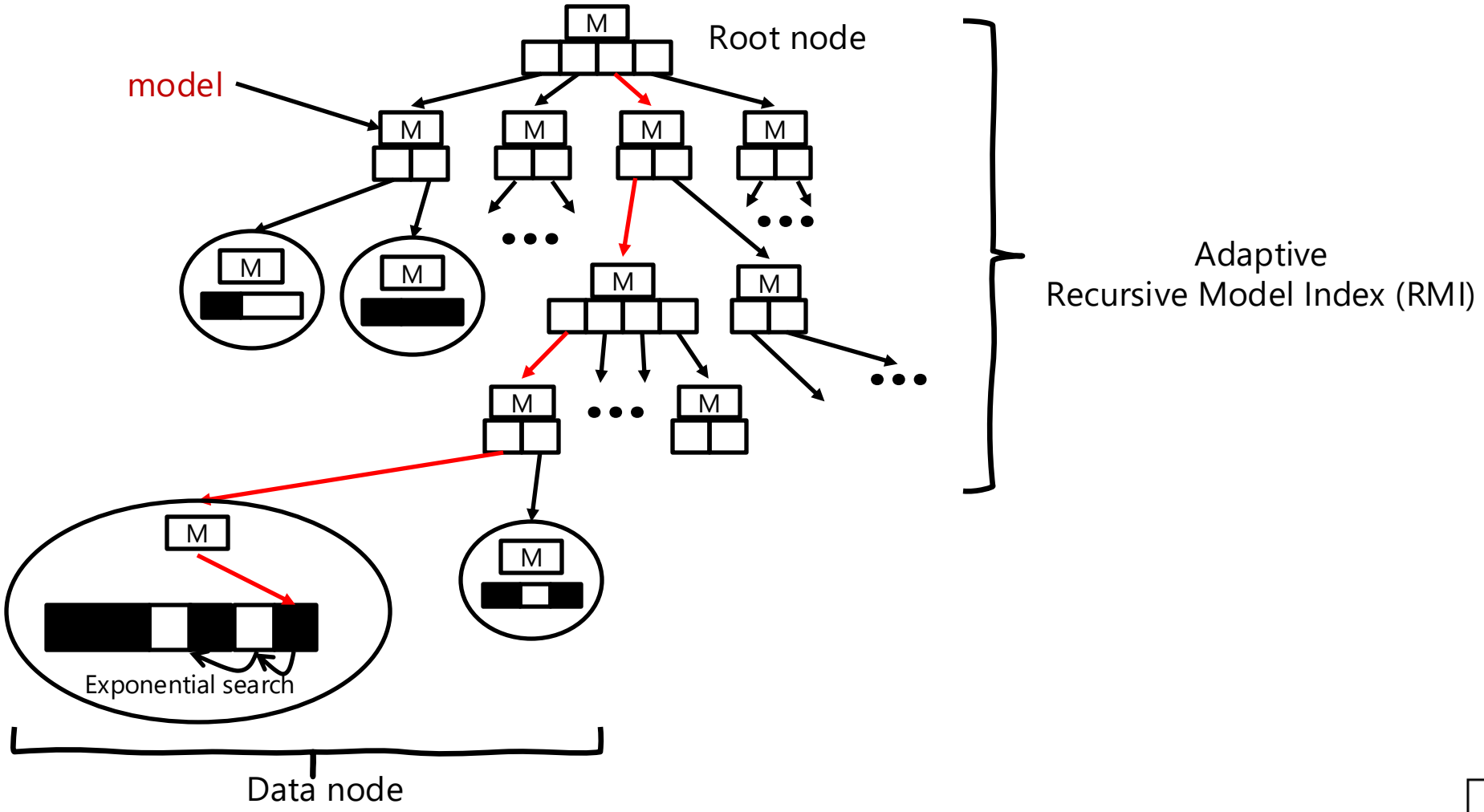


Ding, et.al [SIGMOD 2020]

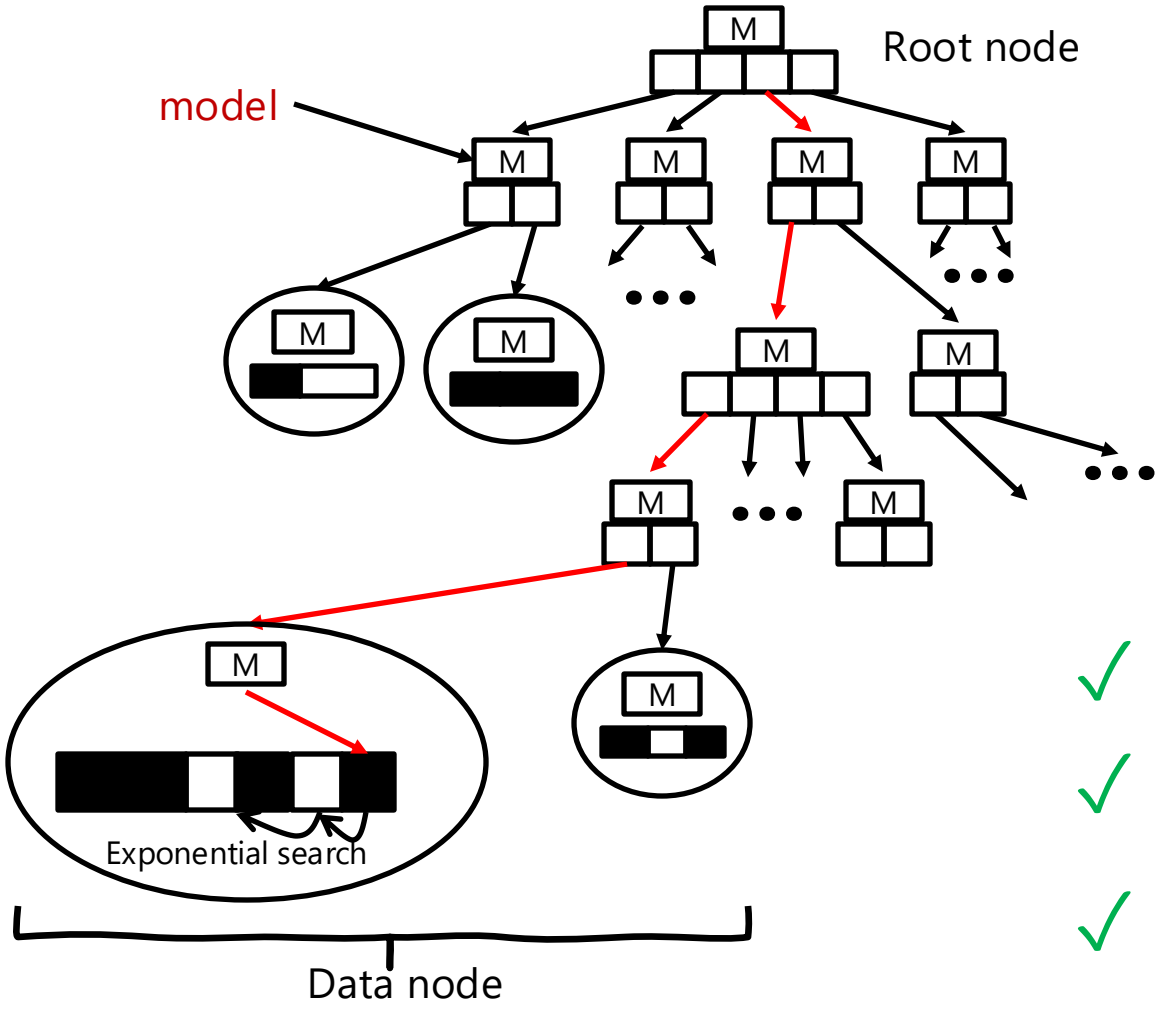
ALEX: An Updatable Learned Index



Lookups in ALEX



Insertions in ALEX



Adaptive
Recursive Model Index (RMI)

- ✓ model based insertions
- ✓ gapped array uses fewer shifts
- ✓ exponential search scales with error size

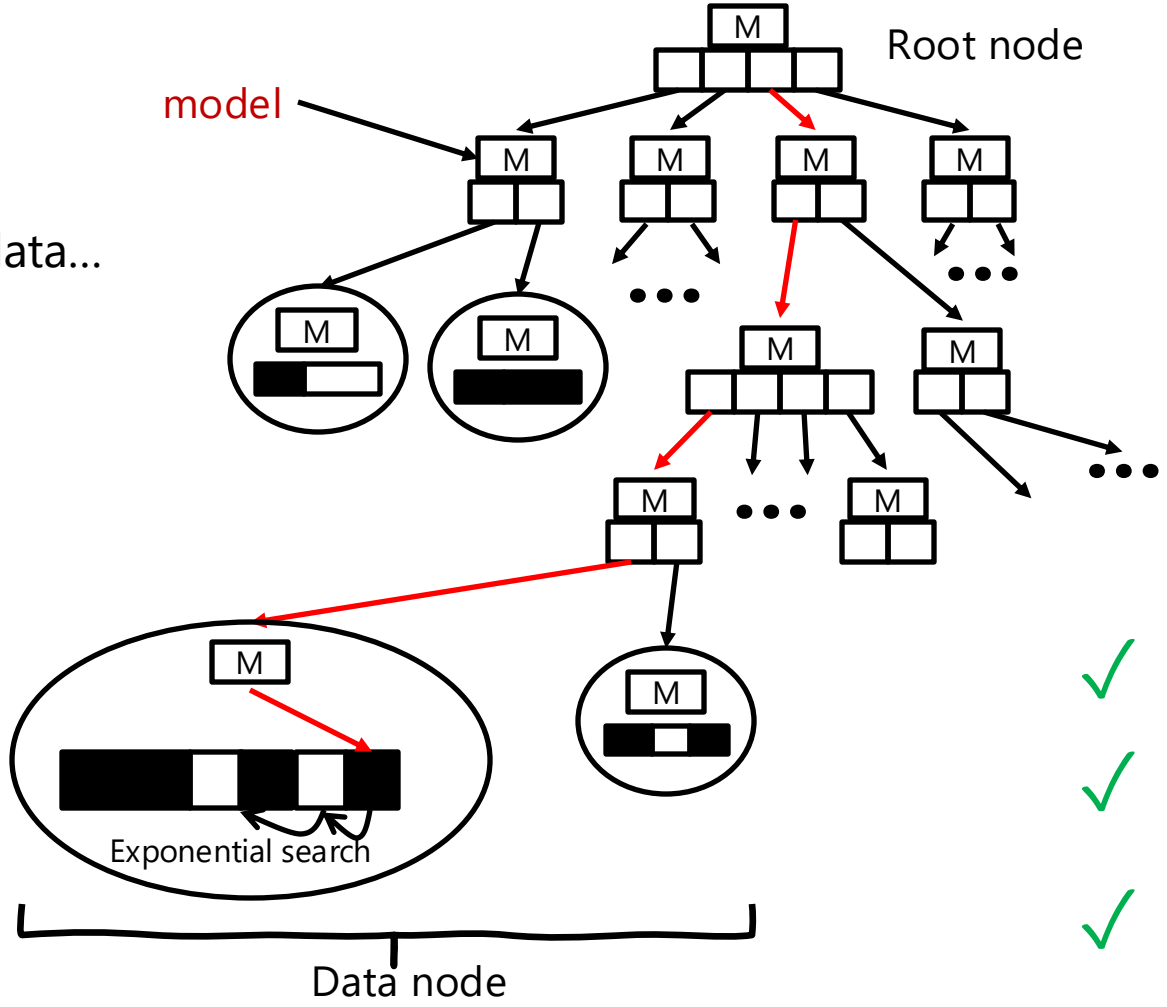
Insertions in ALEX

Inserting pre-structured data…

like a "normal" array

👉 gaps are infrequent

👉 faster exp. Search!



Adaptive
Recursive Model Index (RMI)

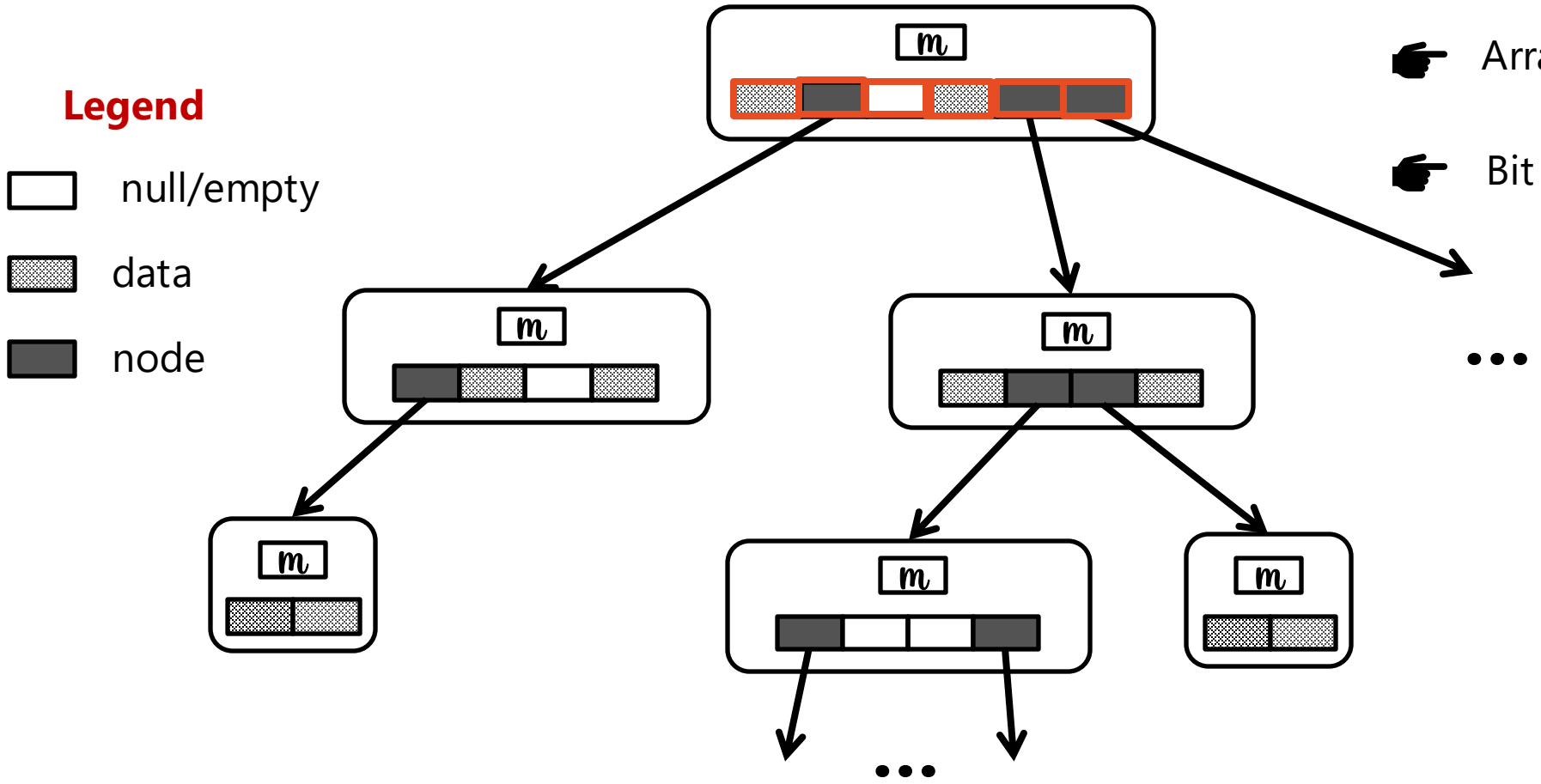
✓ model based insertions

✓ gapped array uses fewer shifts

✓ exponential search scales with error size

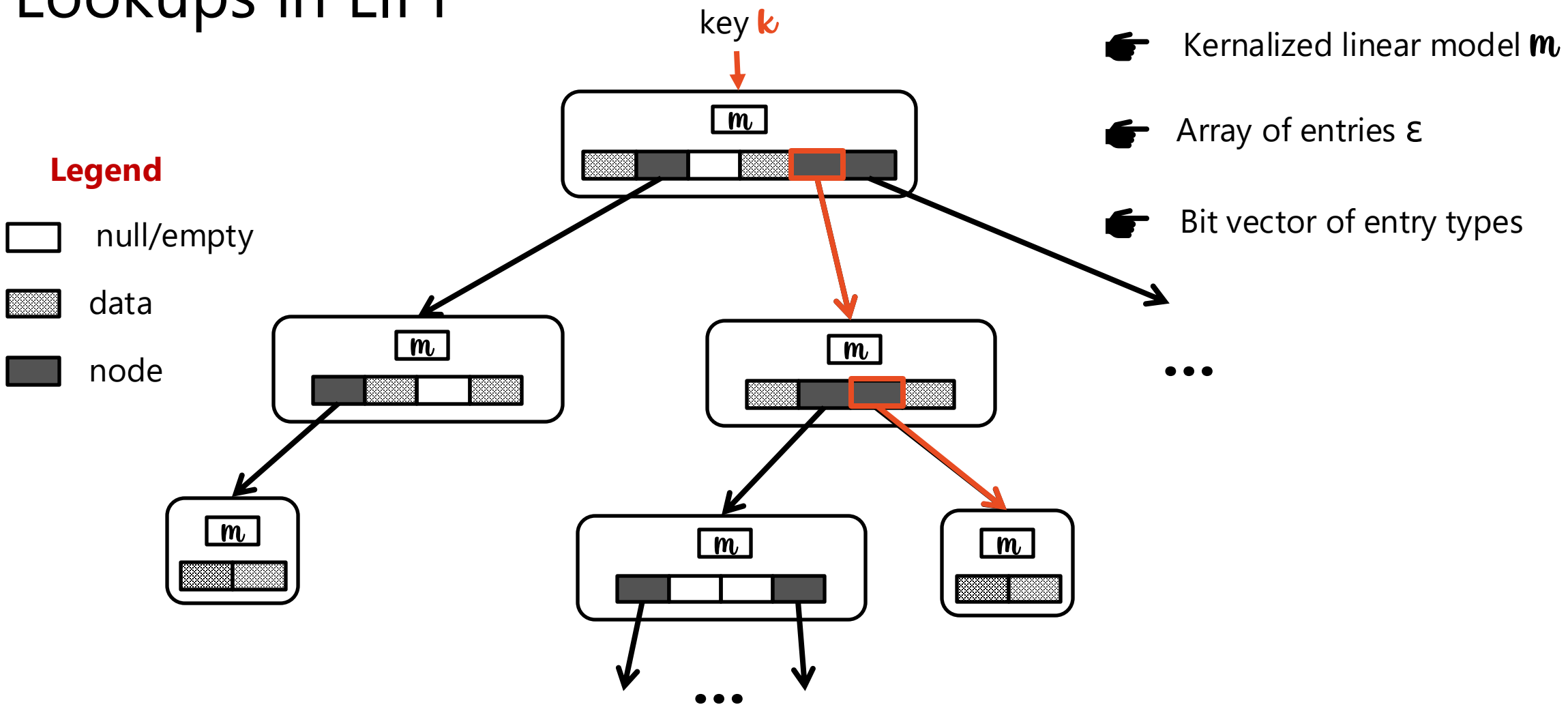
LIPP: An Updatable Learned Index with Precise Positions

- Kernalized linear model m
- Array of entries ε
- Bit vector of entry types

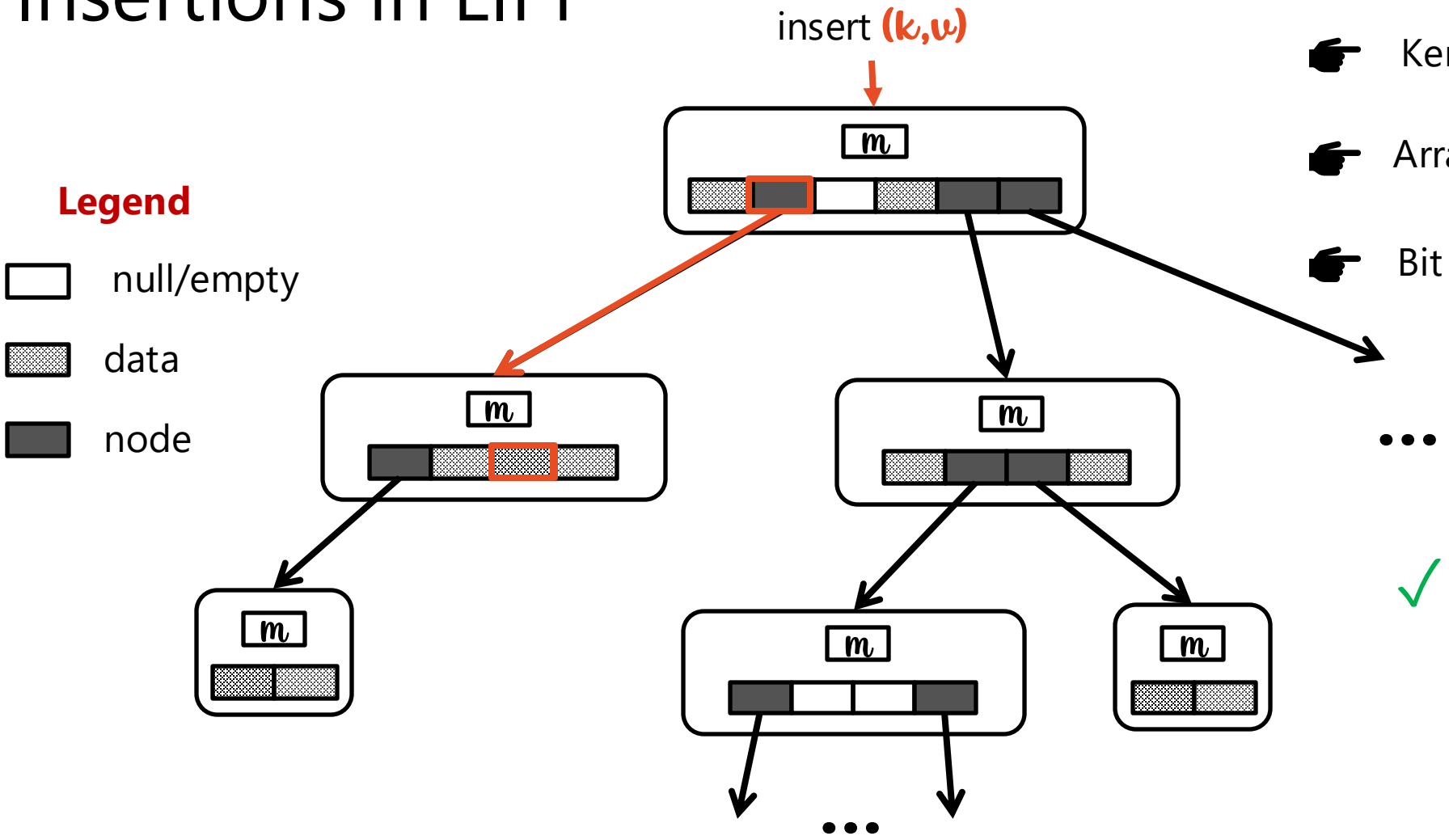


Wu, et.al [VLDB 2021]

Lookups in LIPP



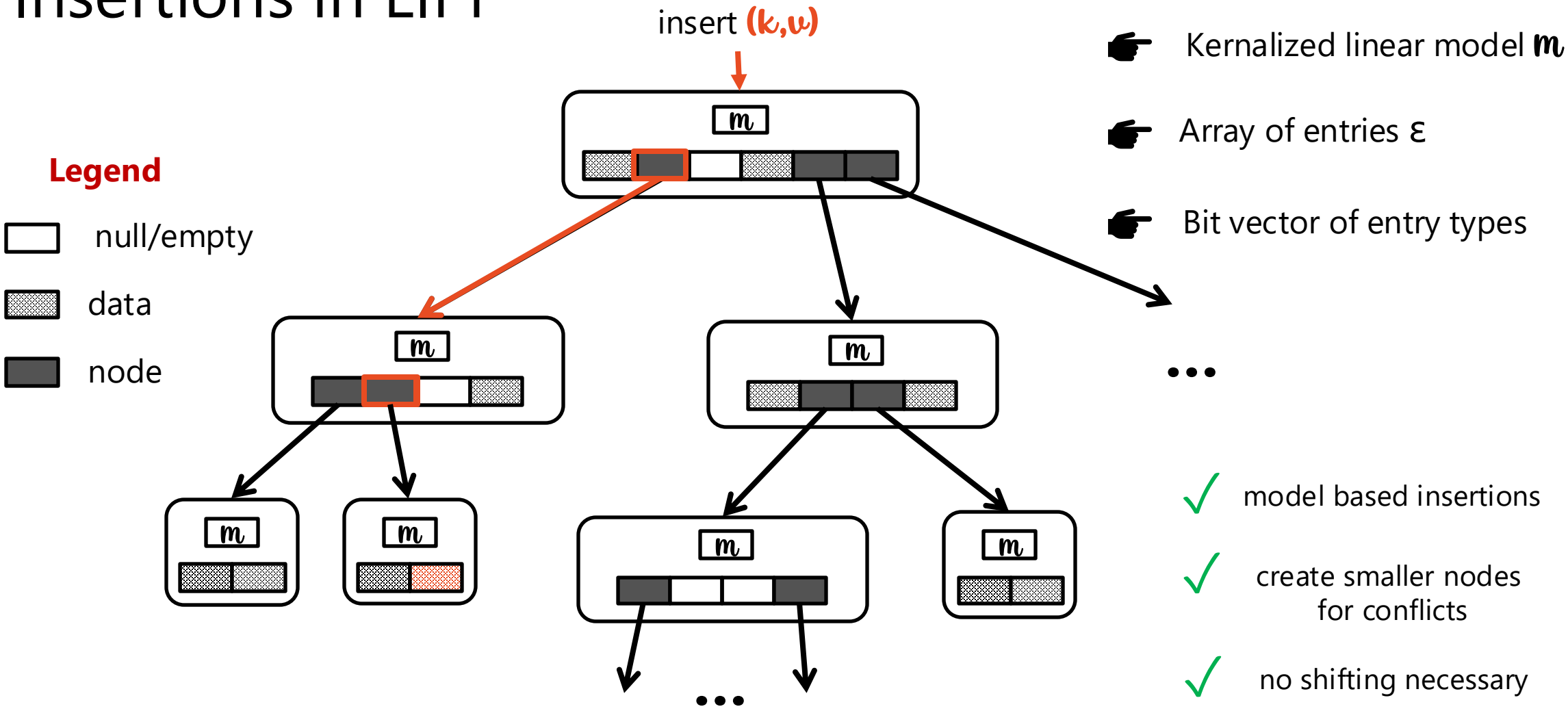
Insertions in LIPP



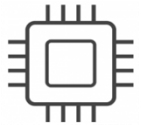
- Kernalized linear model m
- Array of entries ε
- Bit vector of entry types

✓ model based insertions

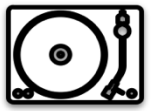
Insertions in LIPP



Log Structured Merge Trees (LSM-trees)

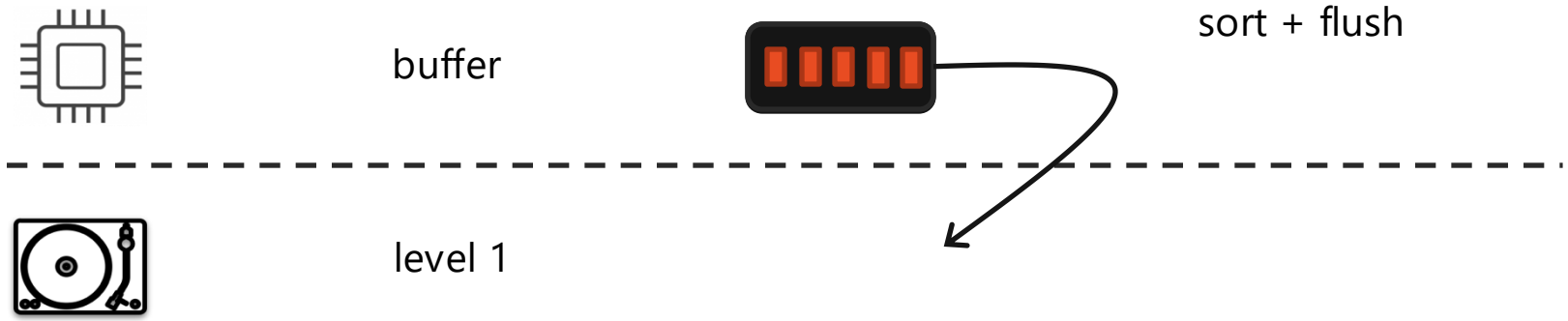


buffer

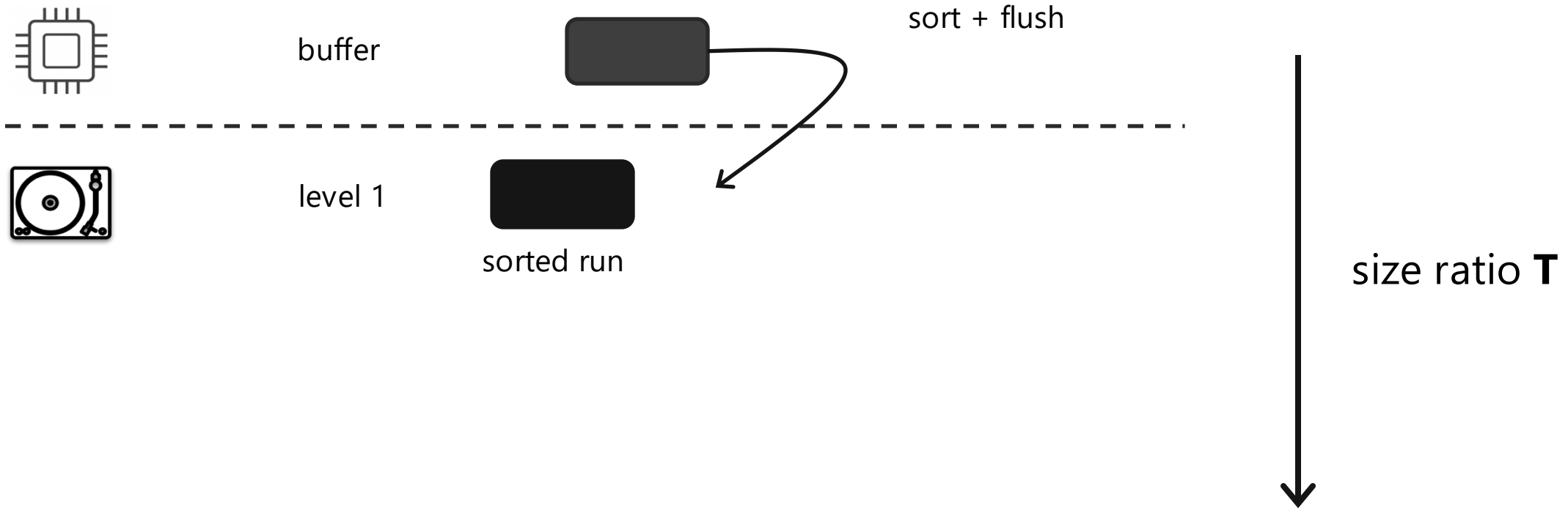


level 1

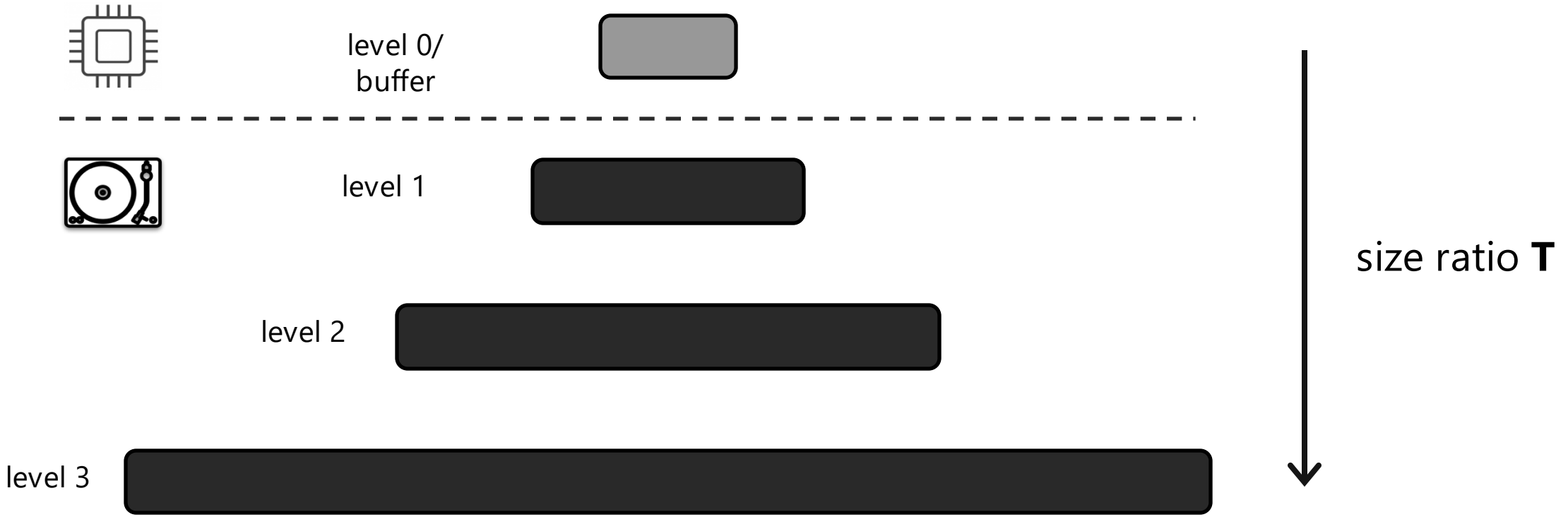
Log Structured Merge Trees (LSM-trees)



Log Structured Merge Trees (LSM-trees)

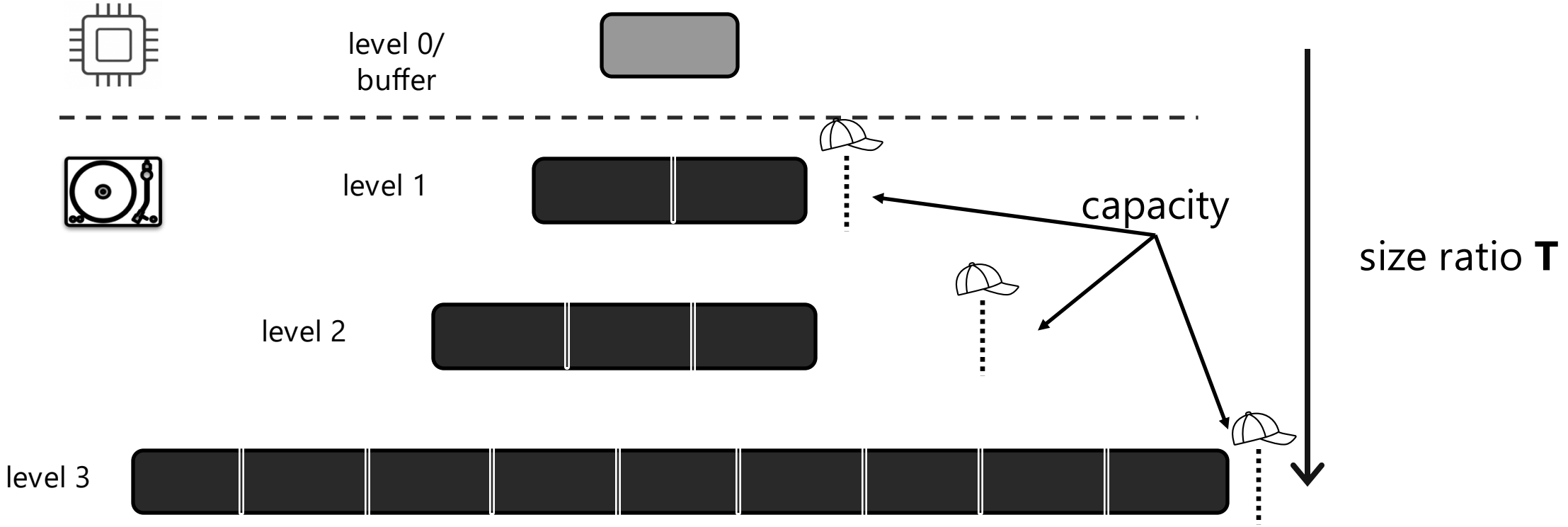


Log Structured Merge Trees (LSM-trees)



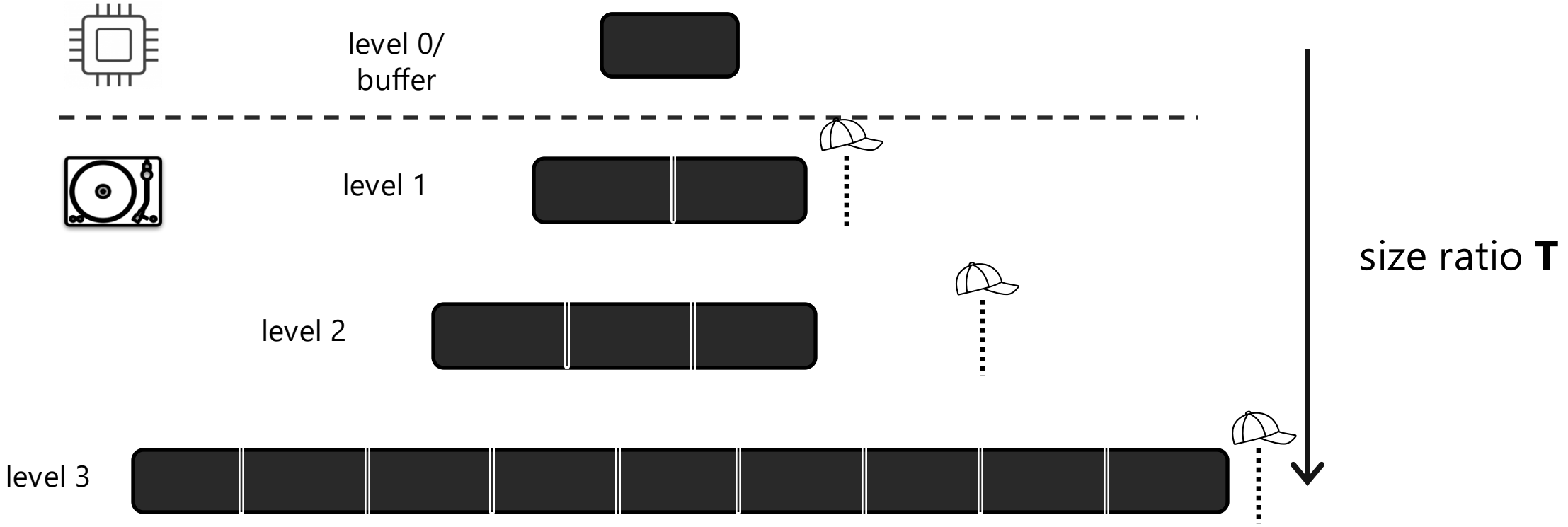
Sort-merging is called ***compaction***

Partial Compactions in RocksDB



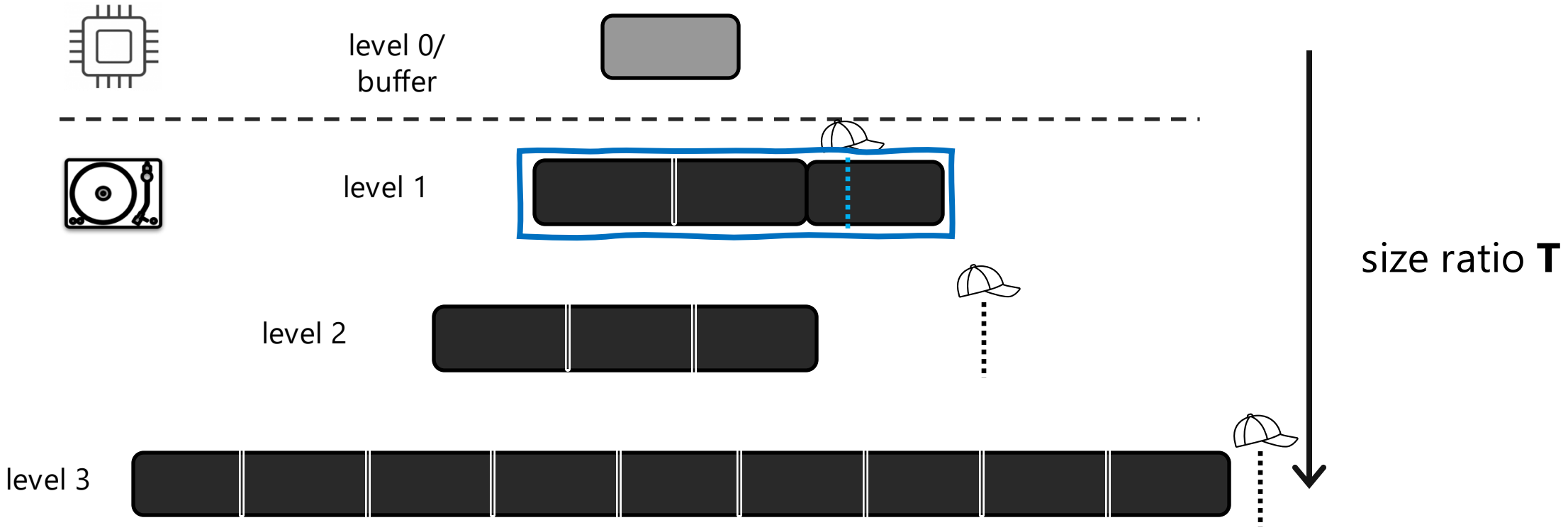
Compaction performed at the granularity of files

Partial Compactions in RocksDB



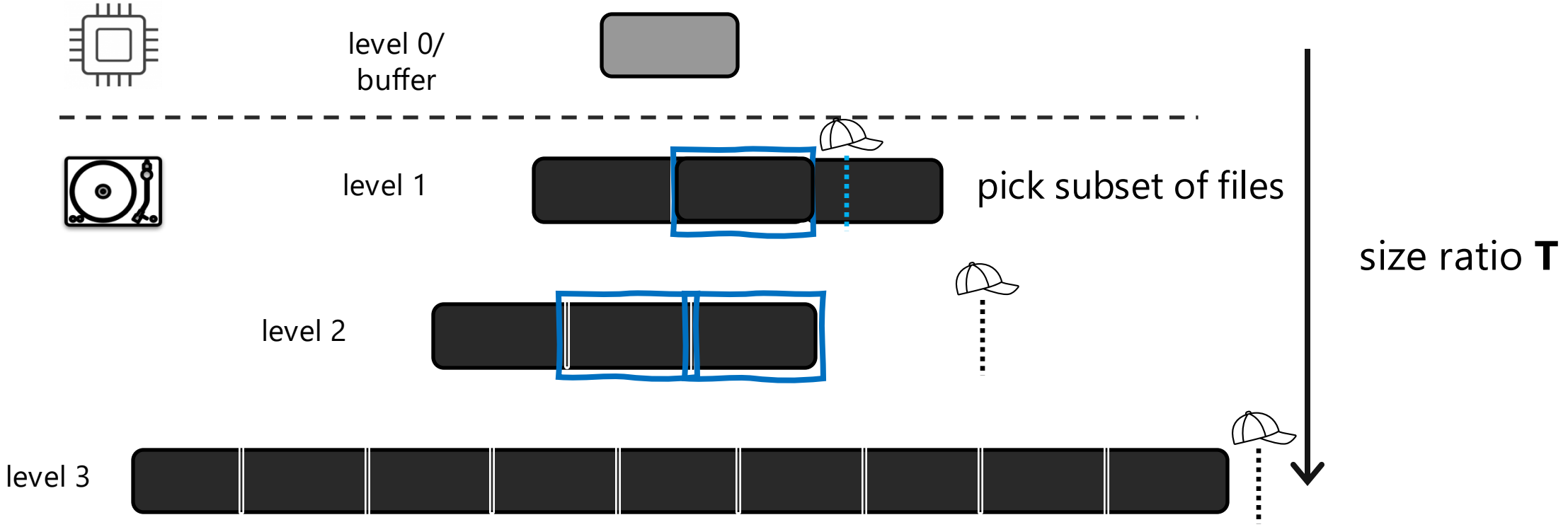
Compaction performed at the granularity of files

Partial Compactions in RocksDB



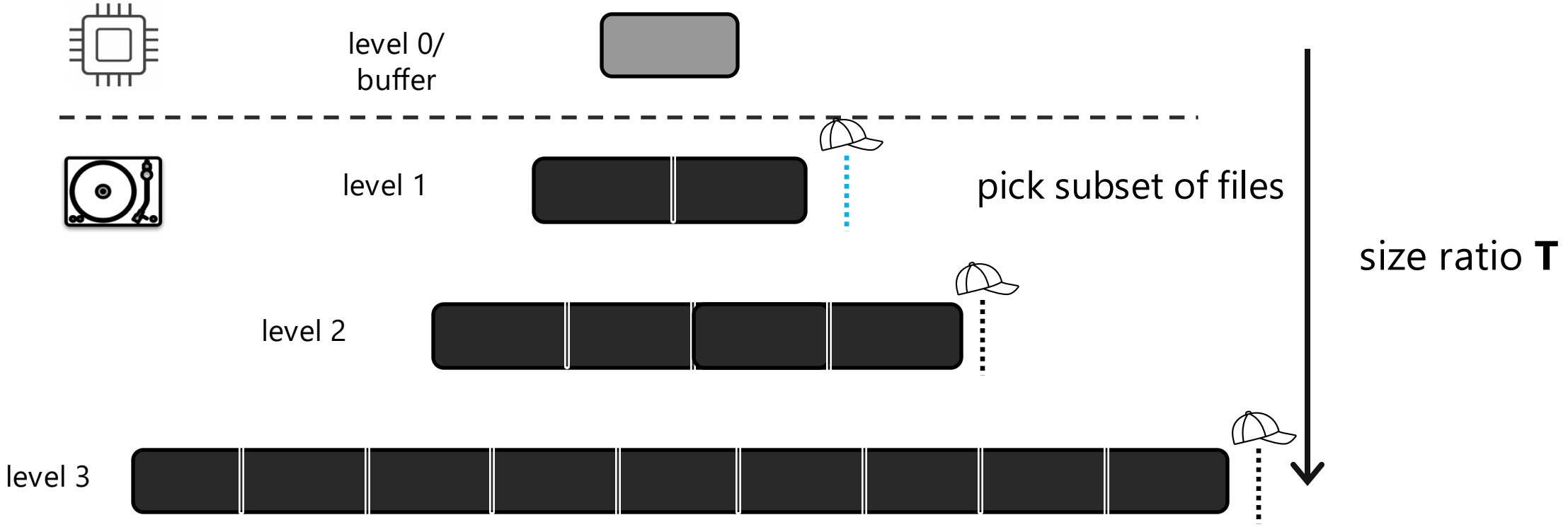
Compaction performed at the granularity of files

Partial Compactions in RocksDB



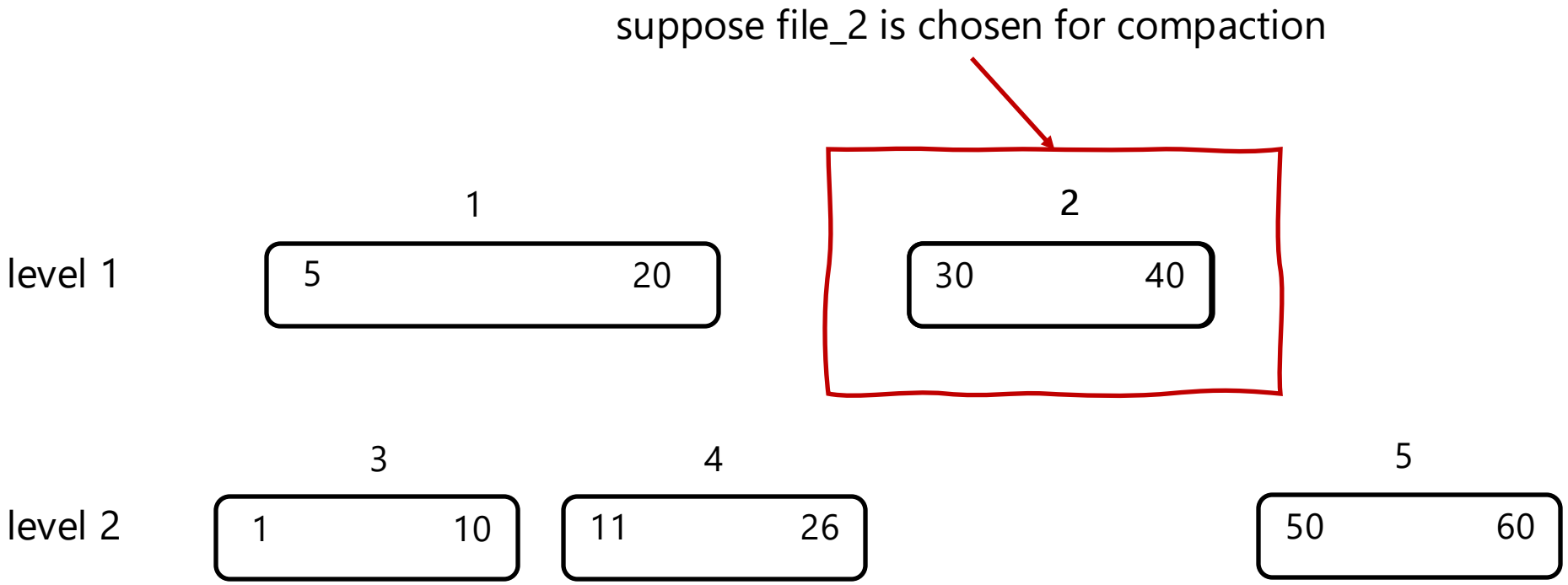
Compaction performed at the granularity of files

Partial Compactions in RocksDB



Compaction performed at the granularity of files

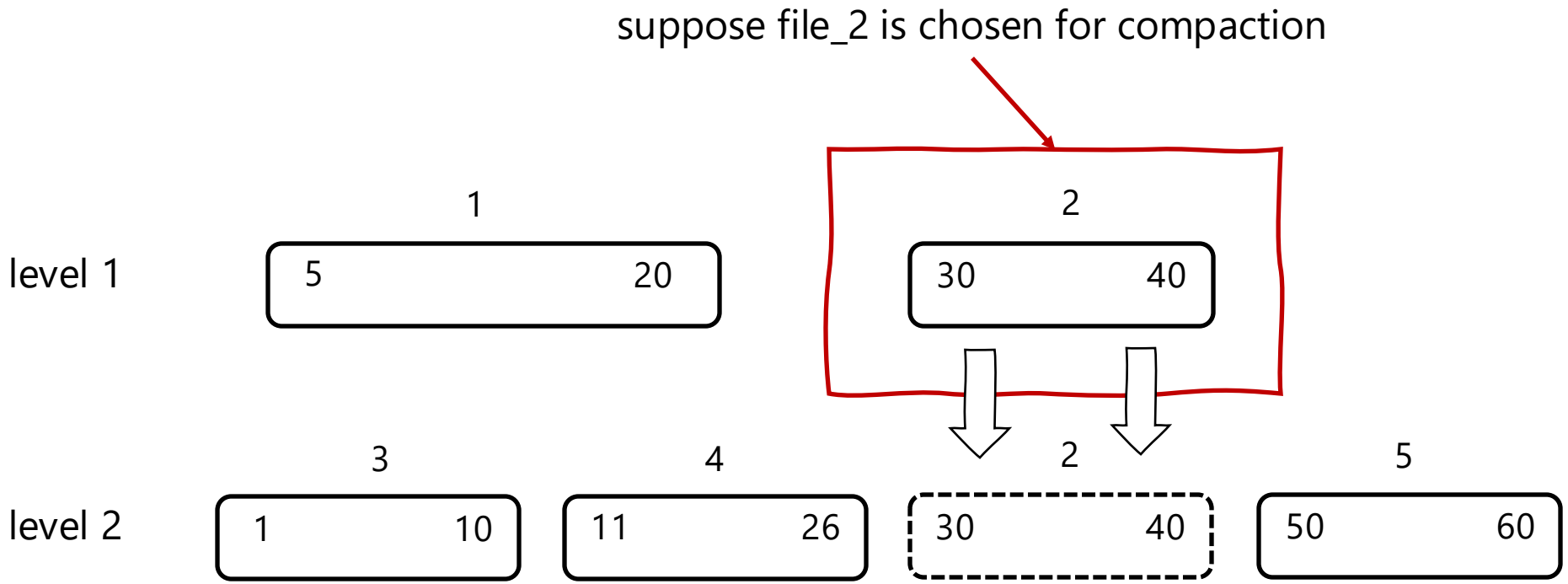
Trivial Moves



compaction would re-write file_2 in level_2

BUT, file_2 has no overlapping keys in level_2

Trivial Moves



compaction would re-write file_2 in level_2

file_2 is actually moved without compaction!

Trivial Moves saves wasteful effort!

Agenda

Introduction

Vision

Background on Index Designs

Sortedness Metrics & Evaluation Framework

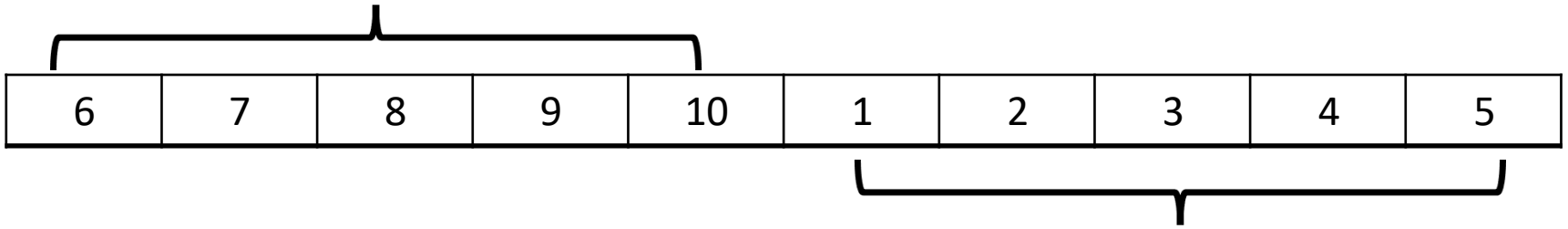
Benchmarking Results

Quantifying Data Sortedness

Metric	Description
Inversions	# pairs in incorrect order
Runs	# increasing contiguous subsequences
Exchanges	least # swaps needed to establish total order

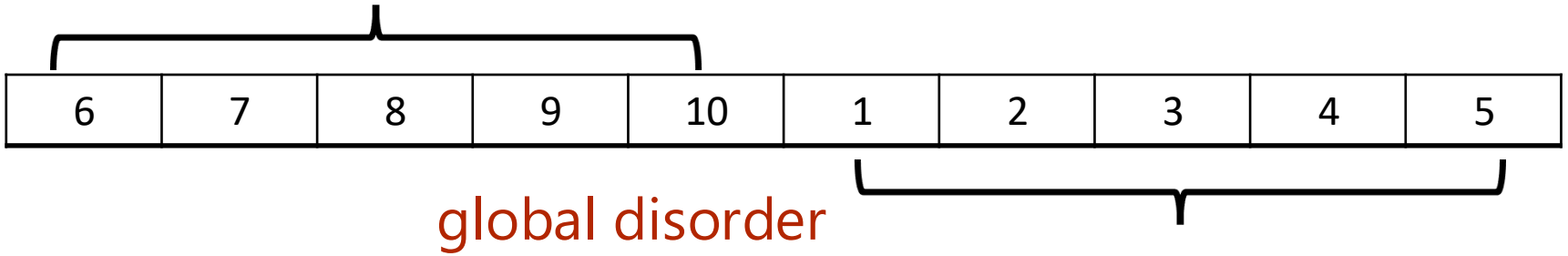
Quantifying Data Sortedness

Metric	Description
Inversions	# pairs in incorrect order
Runs	# increasing contiguous subsequences
Exchanges	least # swaps needed to establish total order



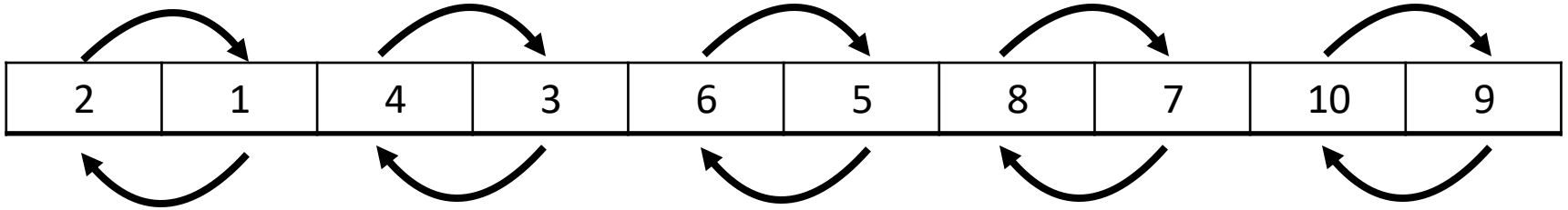
Quantifying Data Sortedness

Metric	Description
Inversions ⛔	# pairs in incorrect order
Runs ✅	# increasing contiguous subsequences
Exchanges ⛔	least # swaps needed to establish total order



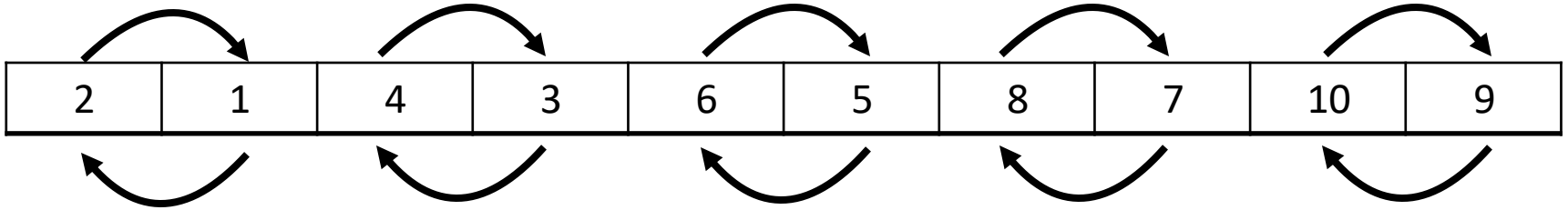
Quantifying Data Sortedness

Metric	Description
Inversions	# pairs in incorrect order
Runs	# increasing contiguous subsequences
Exchanges	least # swaps needed to establish total order



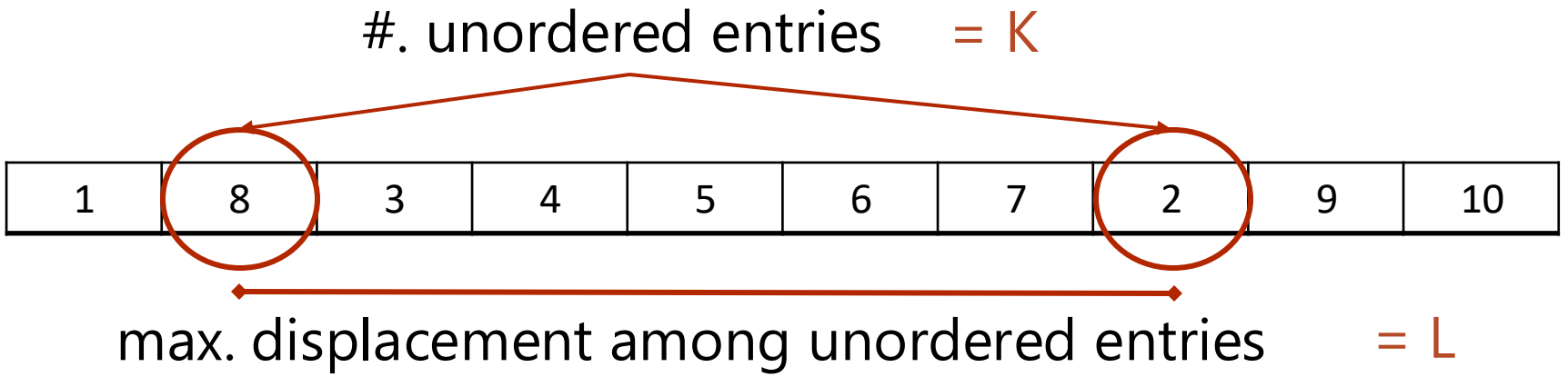
Quantifying Data Sortedness

Metric	Description
Inversions ✅	# pairs in incorrect order
Runs ⛔	# increasing contiguous subsequences
Exchanges ✅	least # swaps needed to establish total order



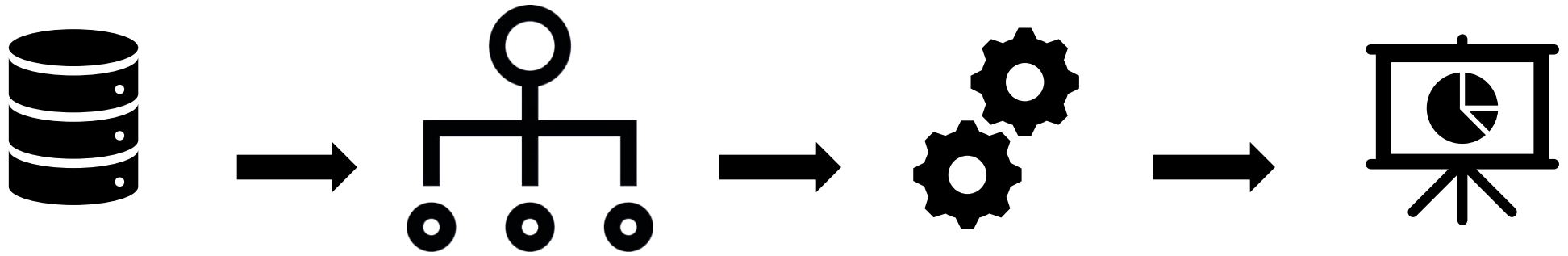
local disorder

(K, L)-Sortedness Metric



inspired by Ben-Moshe, et.al [ICDT 2011]

Framework



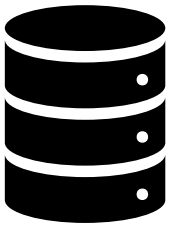
Data generation

Index initialization

Workload execution

Results

Framework



Data generation

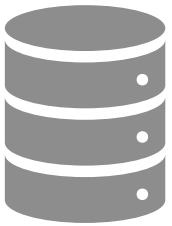
BoDS data generator

Index initialization

Workload execution

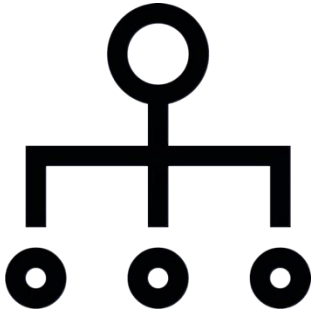
Results

Framework



Data generation

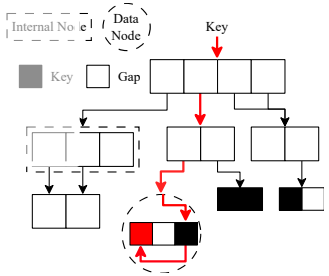
BoDS data generator



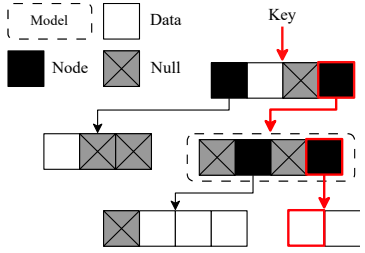
Index initialization

Workload execution

Results



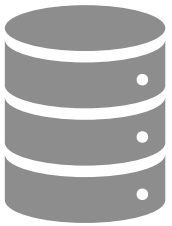
ALEX

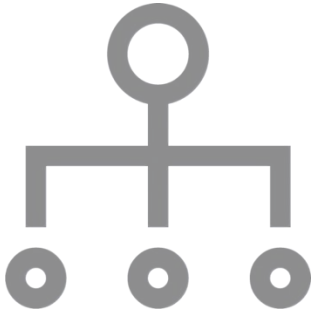


LIPP

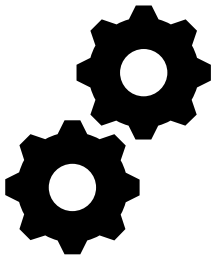
LSM-tree

Framework





read, write or read+write



Data generation

Index initialization

Workload execution

Results

BoDS data generator

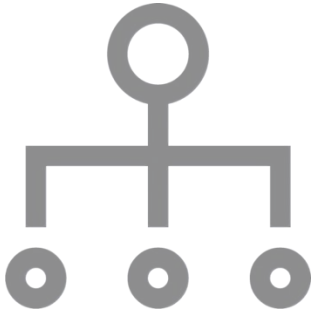
ALEX

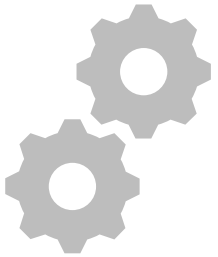
LIPP

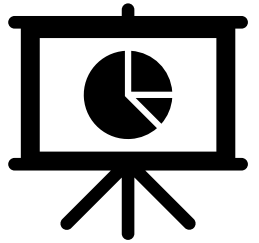
LSM-tree

Framework

read, write or read+write







Data generation

Index initialization

Workload execution

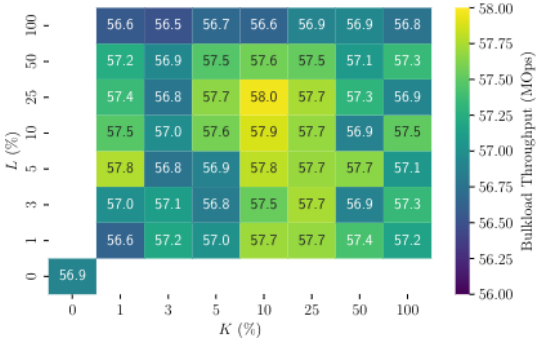
Results

BoDS
data generator

ALEX

LIPP

LSM-tree



Experimental Setup

Server Setup

2 Intel Xeon Gold 6230 processors

384GB main memory

1TB Dell P4510 NVMe drive

CentOS 7.9.2009

Default page size = 4KB

Index Setup

ALEX: Node size = 16MB (default setting)

LIPP: Bitmap width = 1 Byte (default setting)

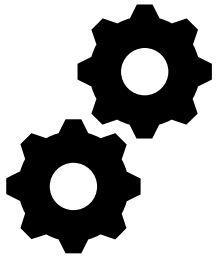
RocksDB:

kCompactionStyleLevel & kMinOverlappingRatio

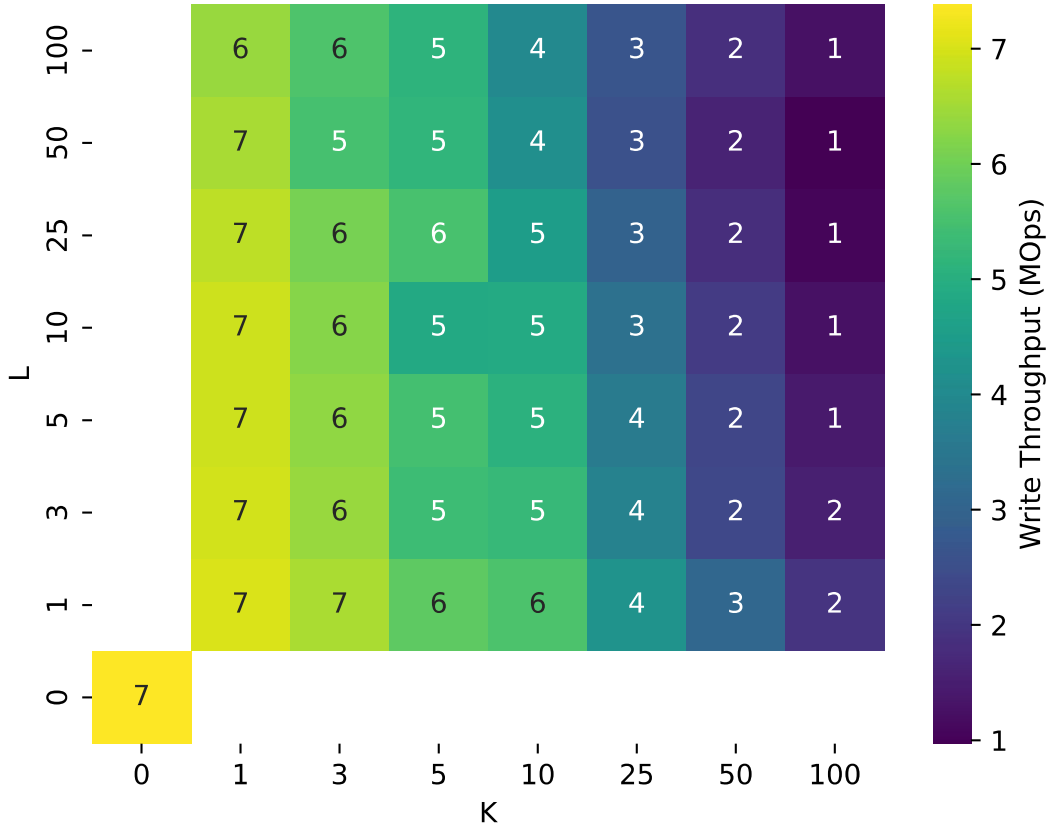
Buffer size = 40MB; Size ratio = 4

Reading the Results

read, write or read+write

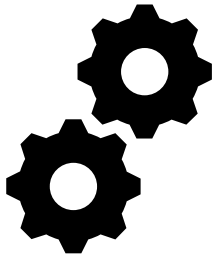


Workload execution

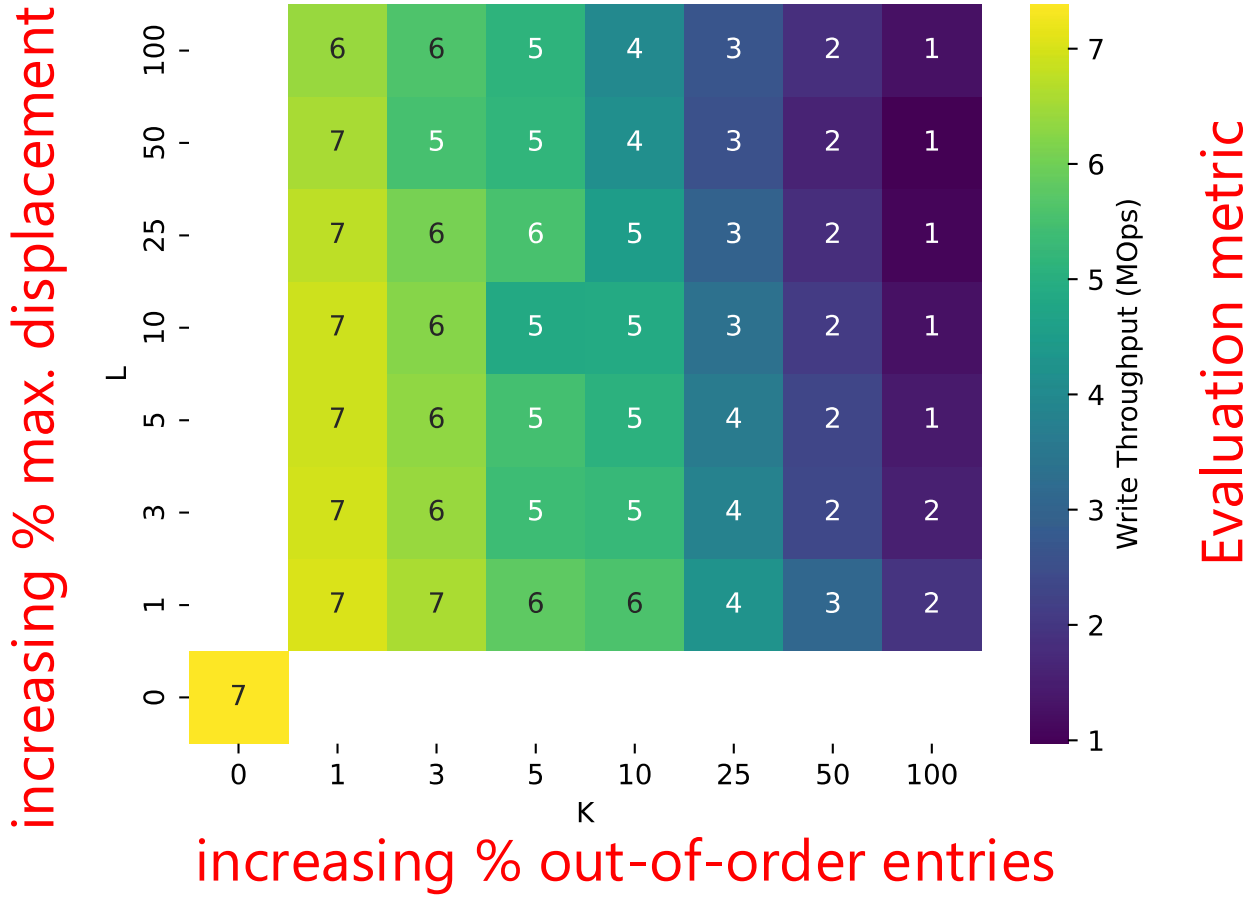


Reading the Results

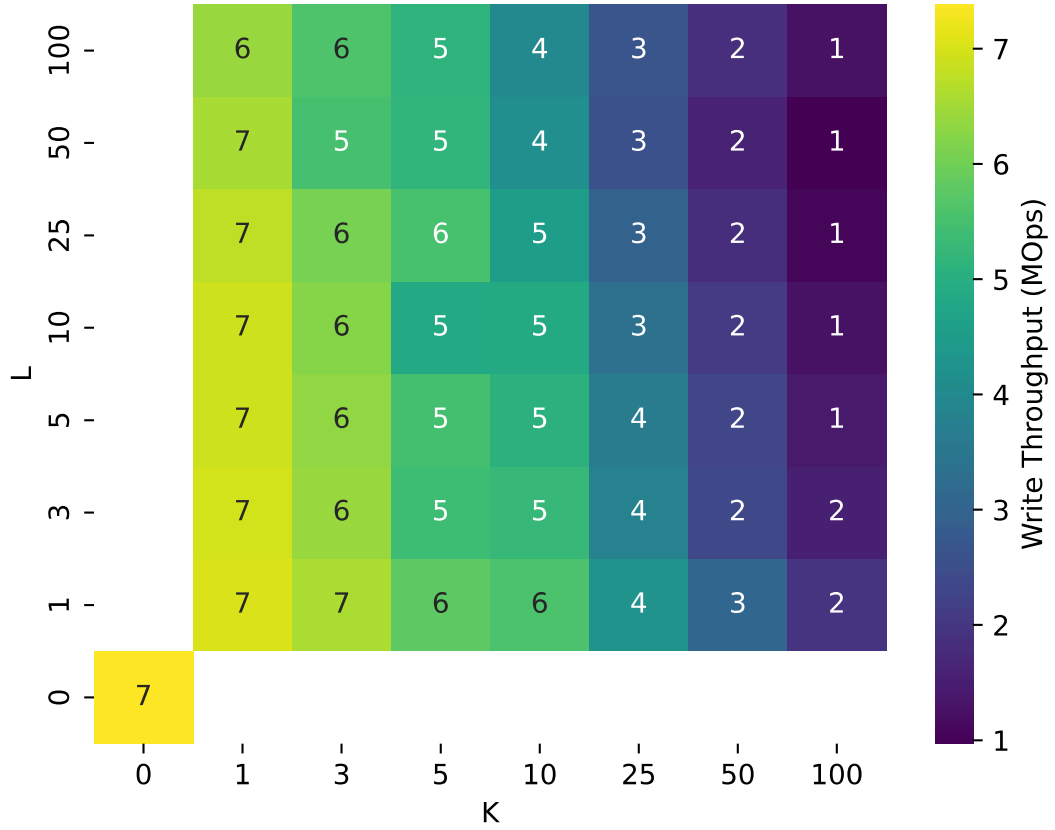
read, write or read+write



Workload execution



Let's talk about the B⁺-tree



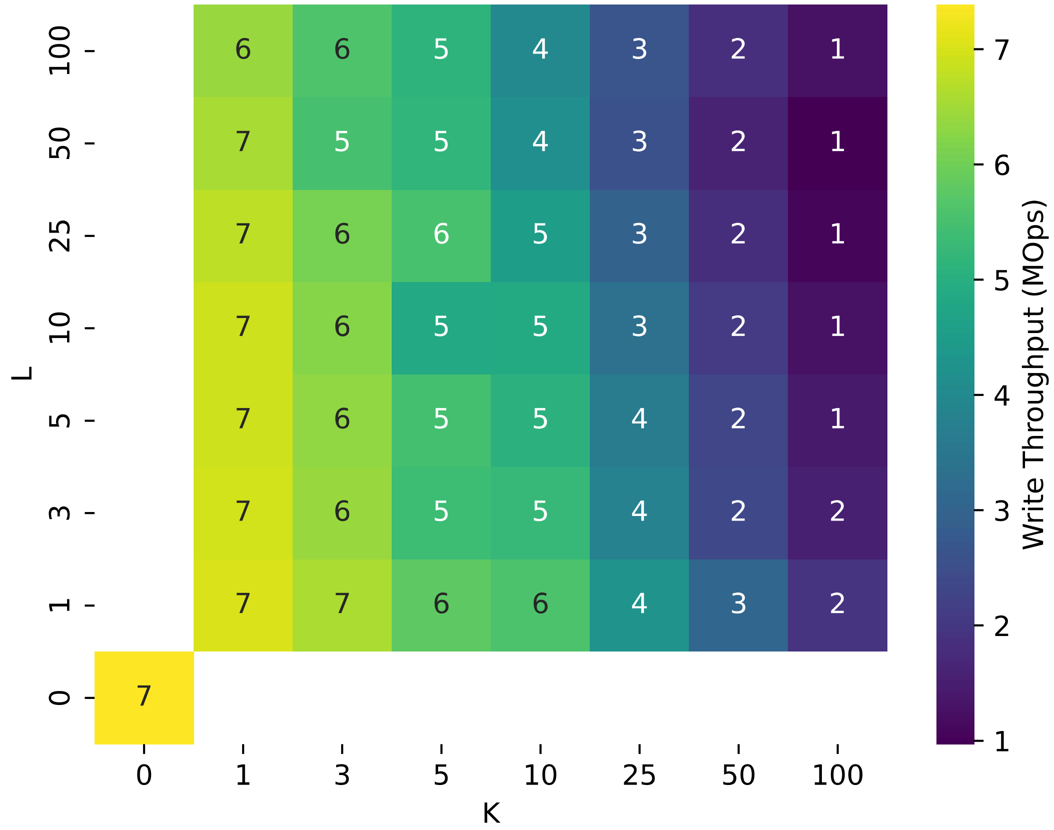
☛ B⁺-tree always performs index traversals

Why higher throughput with high sortedness?

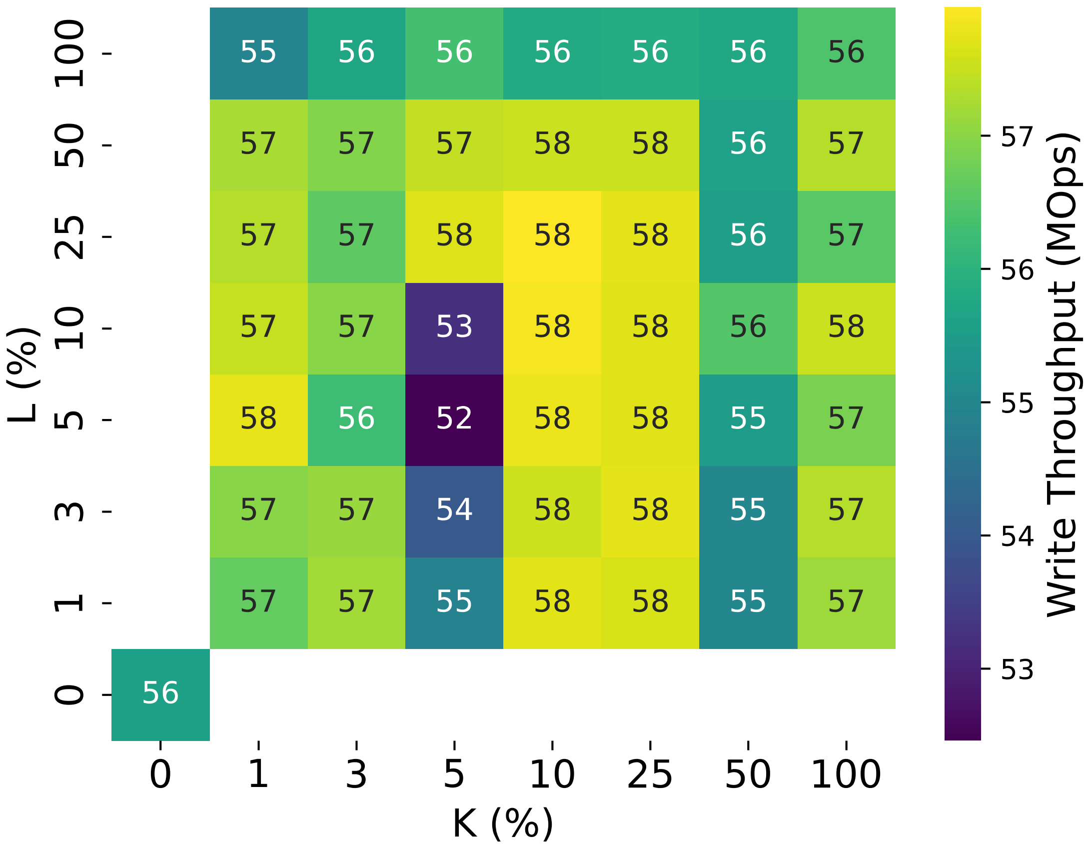
✓ improved performance due to caching effects

However, this is not the ideal performance…

Let’s talk about the B⁺-tree



Preloading performance



Bulk loading is at least 8x faster!

Indexing for Near-Sorted Data [ICDE ‘23]

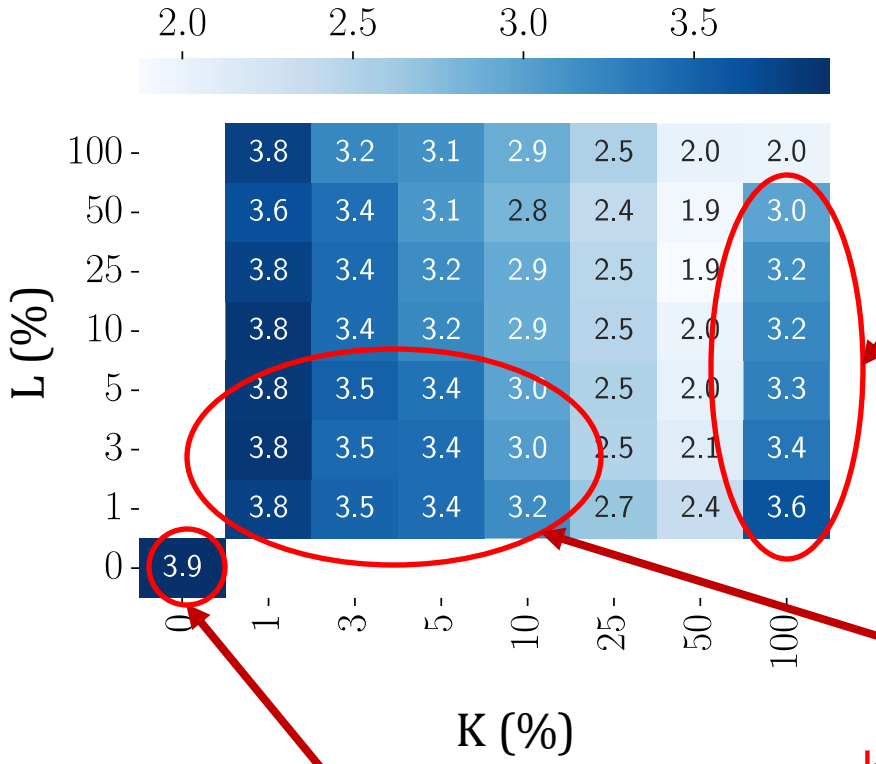
Getting back to Learned Indexes and LSM-trees…

sortedness-responsiveness ≠ sortedness-adaptivity

Performance can change with differently sorted data, but is it optimal?

ALEX Performs Best With High Data Sortedness!

- ✓ High sortedness ⇒dense runs + terminal gaps
- ✓ Less shifting required
- ✓ Faster exponential search!



higher throughput L low or moderate

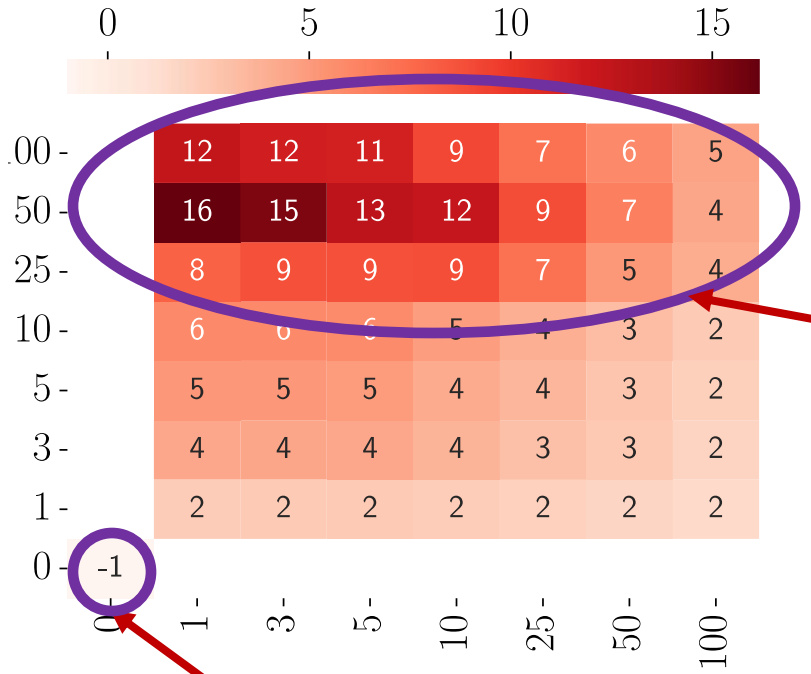
Bulk loading is at least 2x faster!

Best throughput

higher throughput when K and L are low

LIPP Performs Better for High L

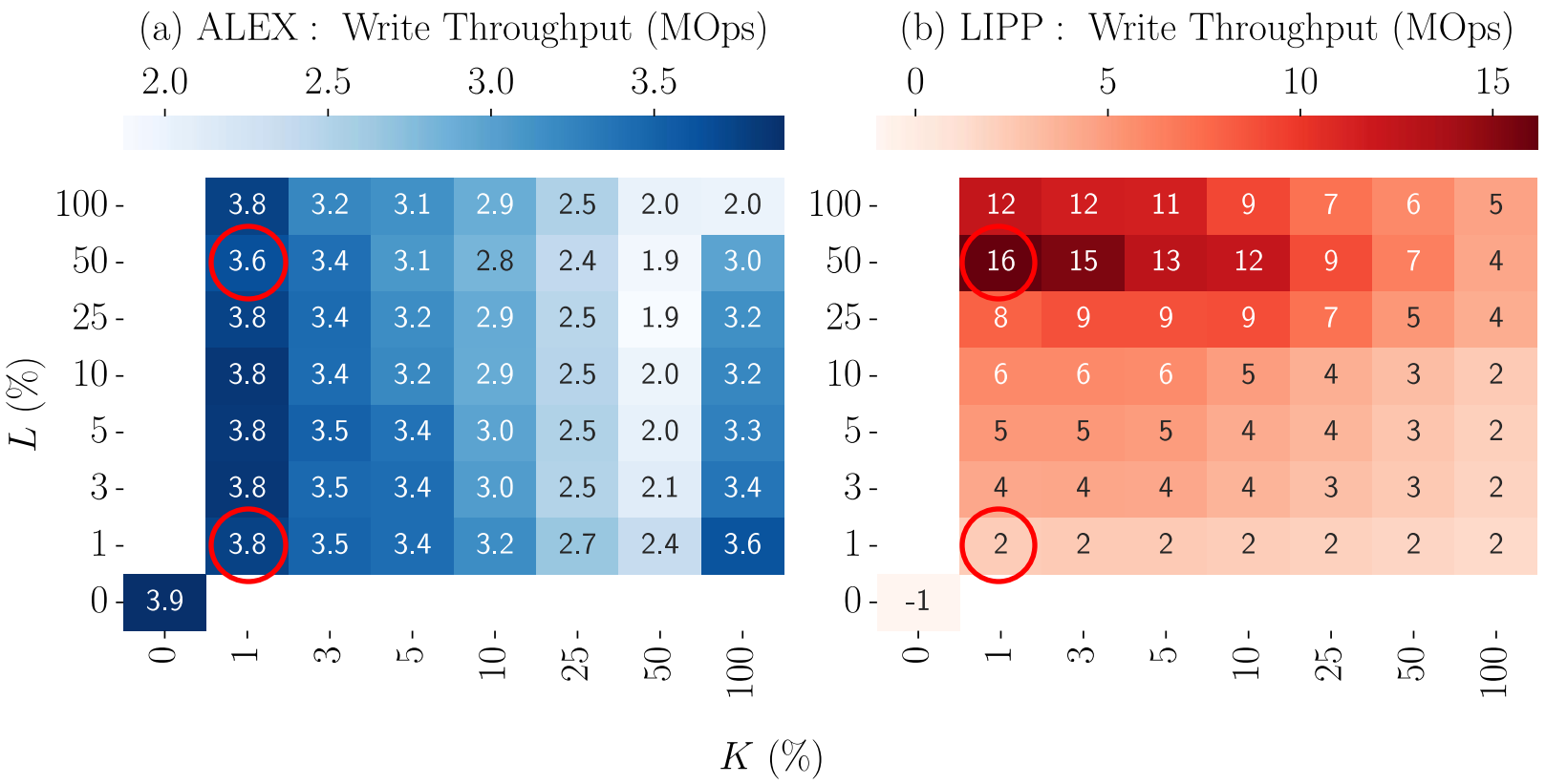
- Too many conflicts creates a deep subtree (like a linked list)
- Fails rebalancing!



Higher throughput for high L

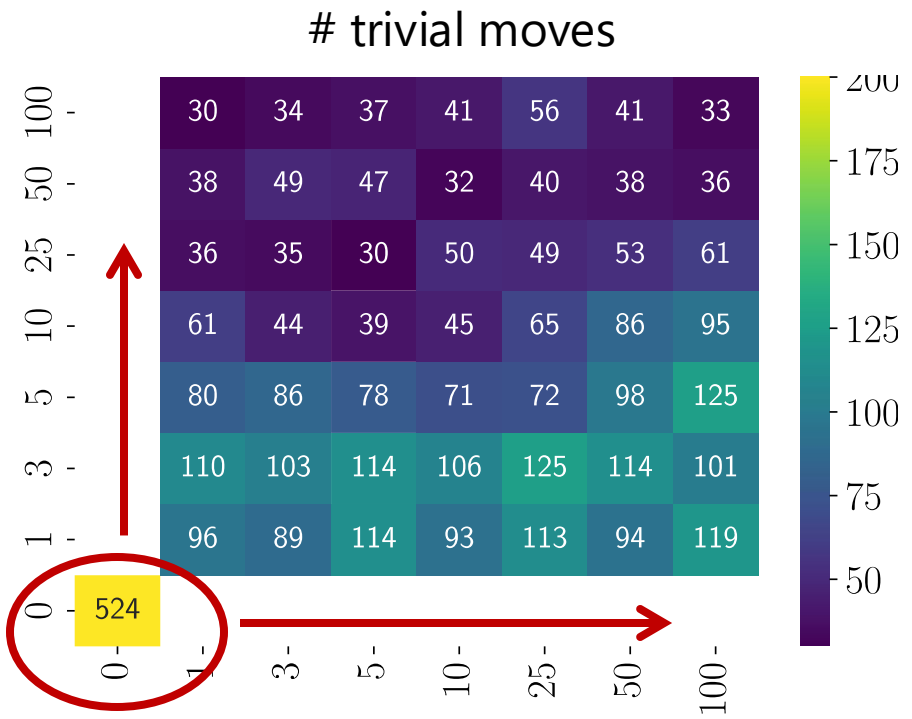
Fails to ingest fully-sorted data

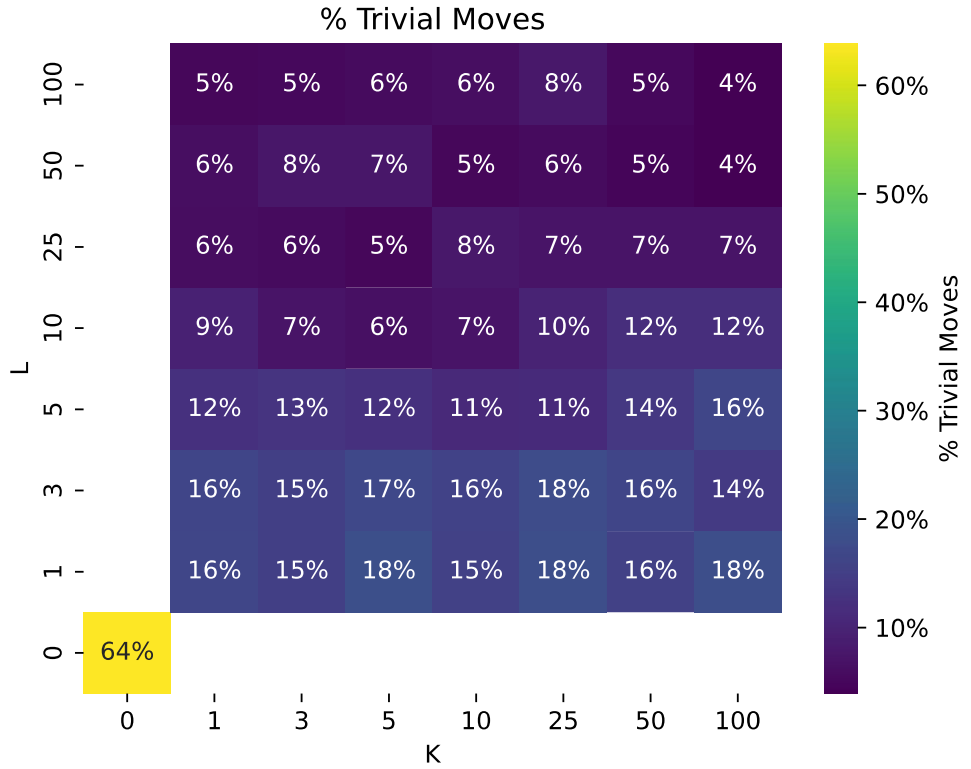
Performance Can Be Unpredictable!



ALEX v/s LIPP:
LIPP can be anywhere between 4.4x faster

LSM Benefits from Trivial Moves



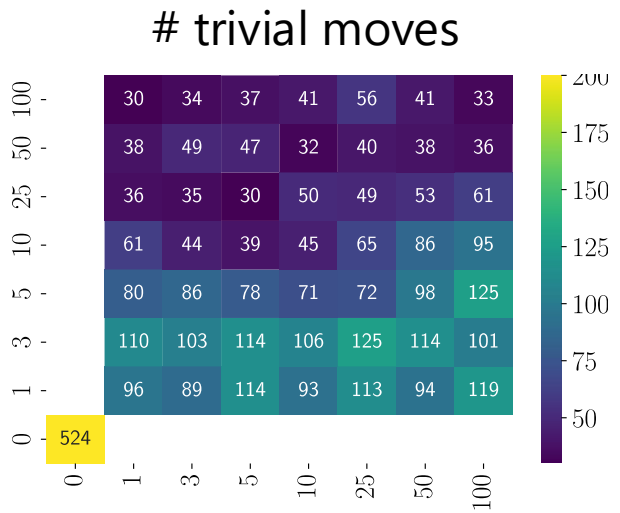


Trivial moves maximized for fully-sorted data ✓

Trivial moves significantly drop with minor unorderness ✗

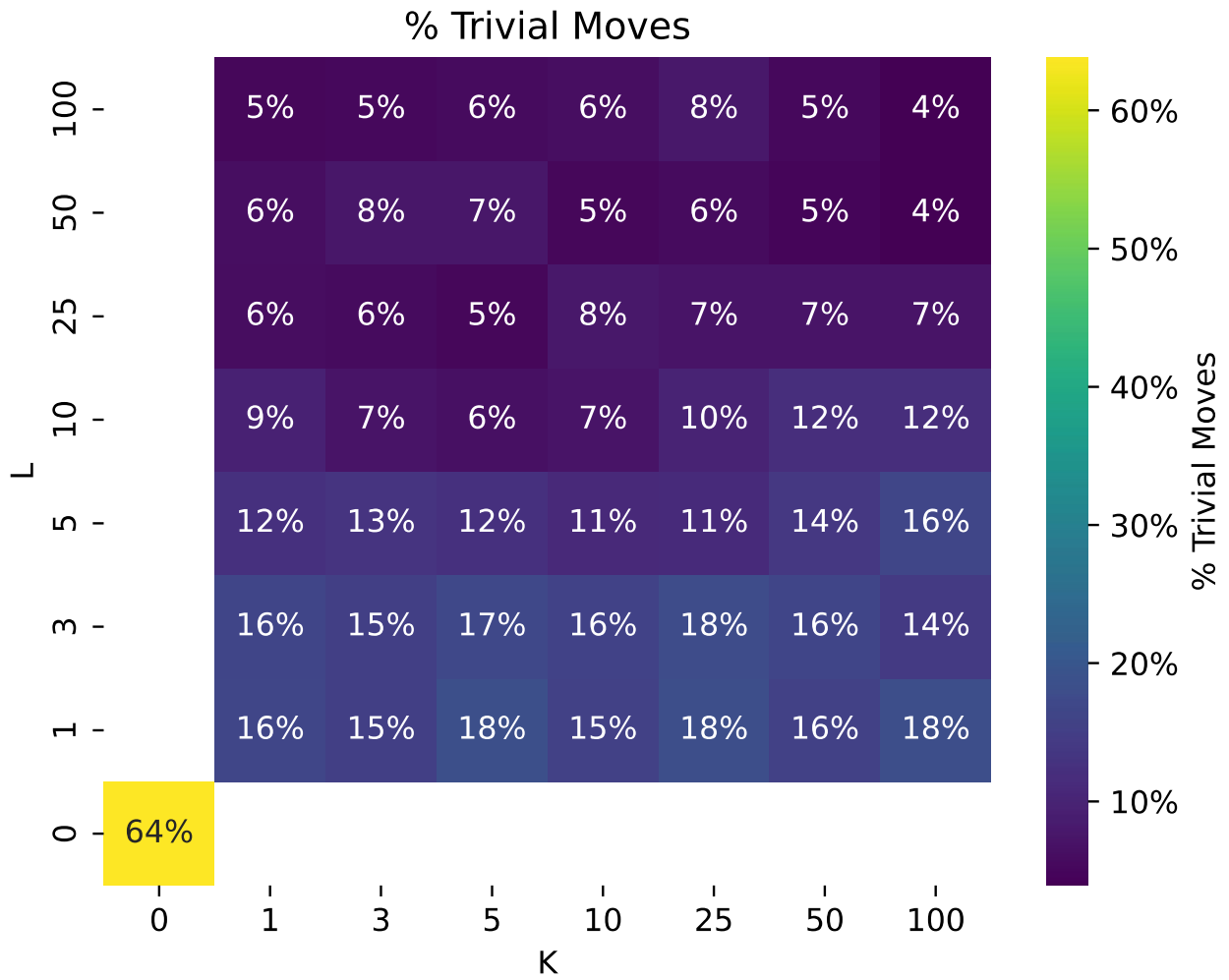
Is this the best we can do?

LSM Benefits from Trivial Moves



Can a more fine-grained compaction granularity work?

Can we further improve trivial moves for fully-sorted data?



Summary

Indexes **should** perform less effort for pre-structured data

Analyzing performance by varying sortedness should be standardized

Learned Indexes are **unpredictable** when varying sortedness

LSM benefit from trivial moves **BUT** → potential room for improvement

Scan for link to paper

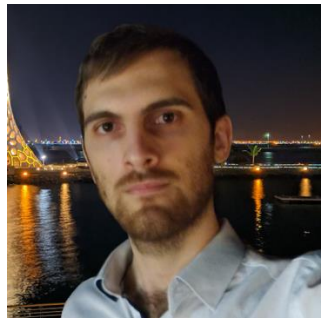
BOSTON UNIVERSITY

lab DiSC

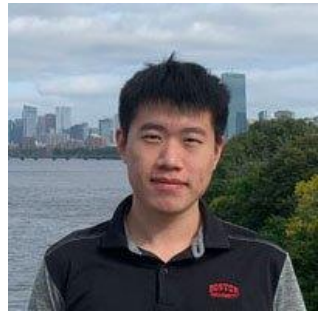
Our Team

Aneesh Raman

Andy Huynh



Konstantinos Karatsenidis



Jinqi Lu

Shaolin Xie

Matthaios Olma

Subhadeep Sarkar

Manos Athanassoulis