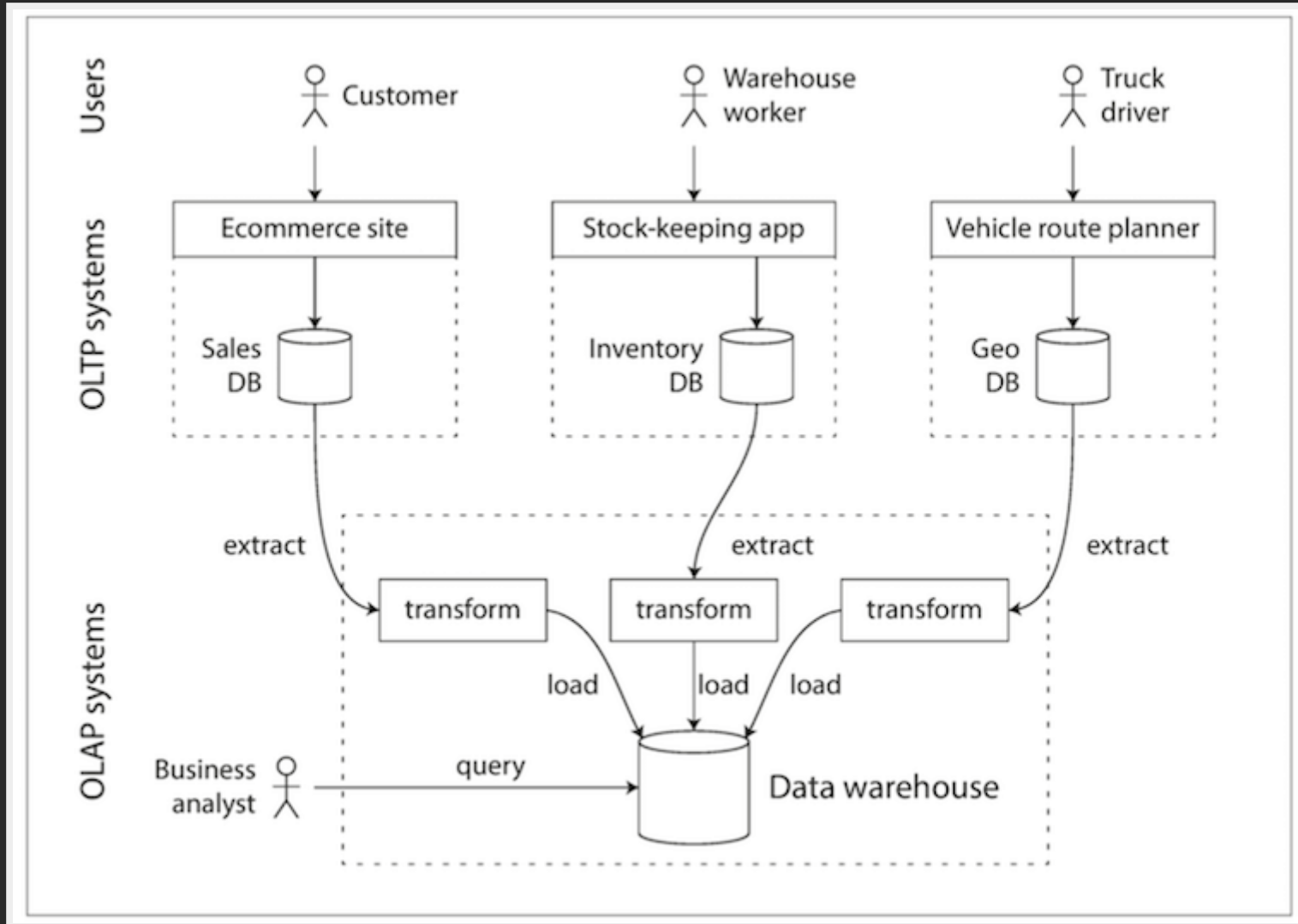


LSM-TREE AND TUNING

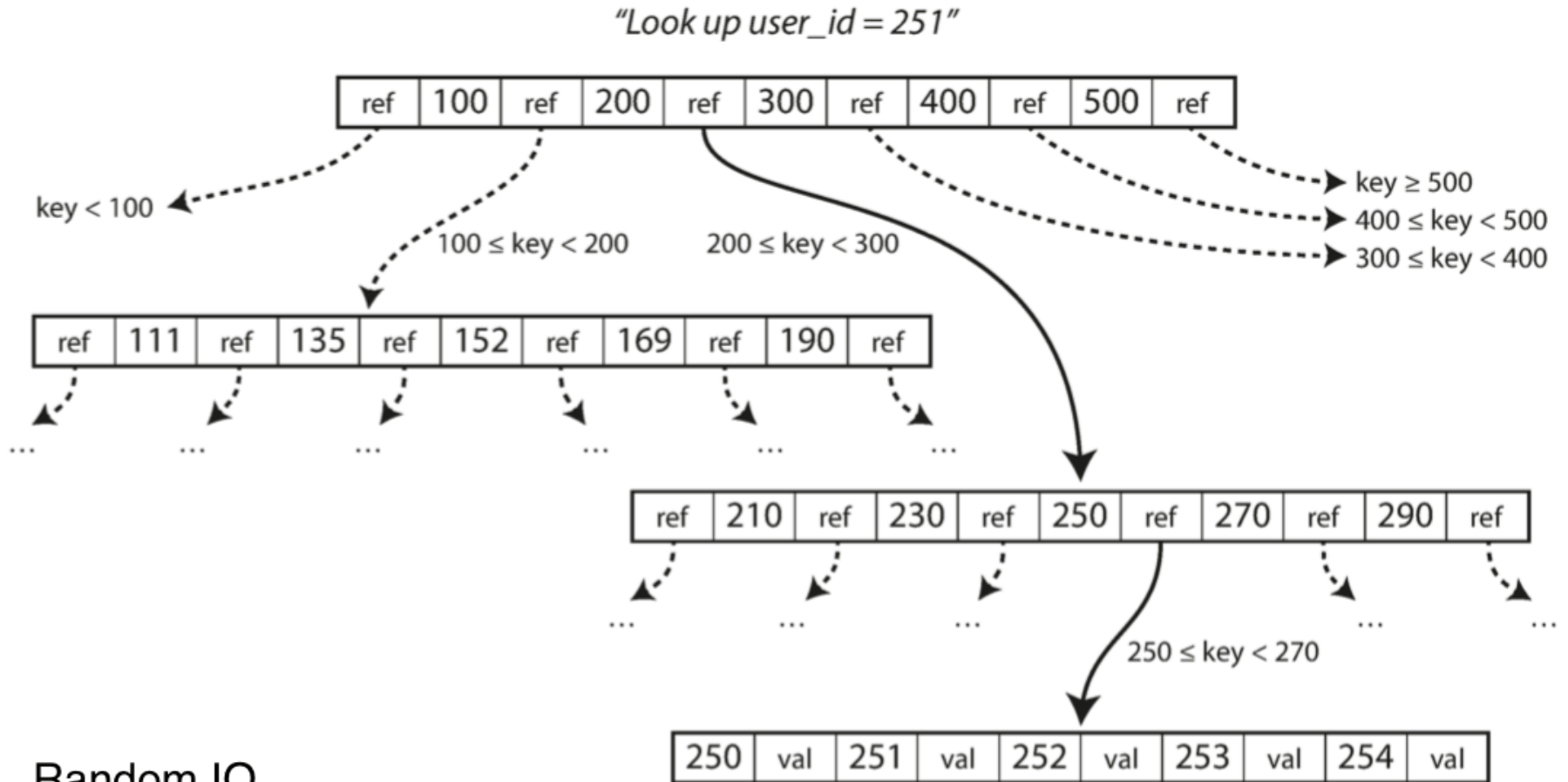
PREFACE

"As many people out there I like to adapt new technologies. There is one rule though, I need to understand what I'm actually using. It isn't necessarily about exact implementation but main concepts behind it."

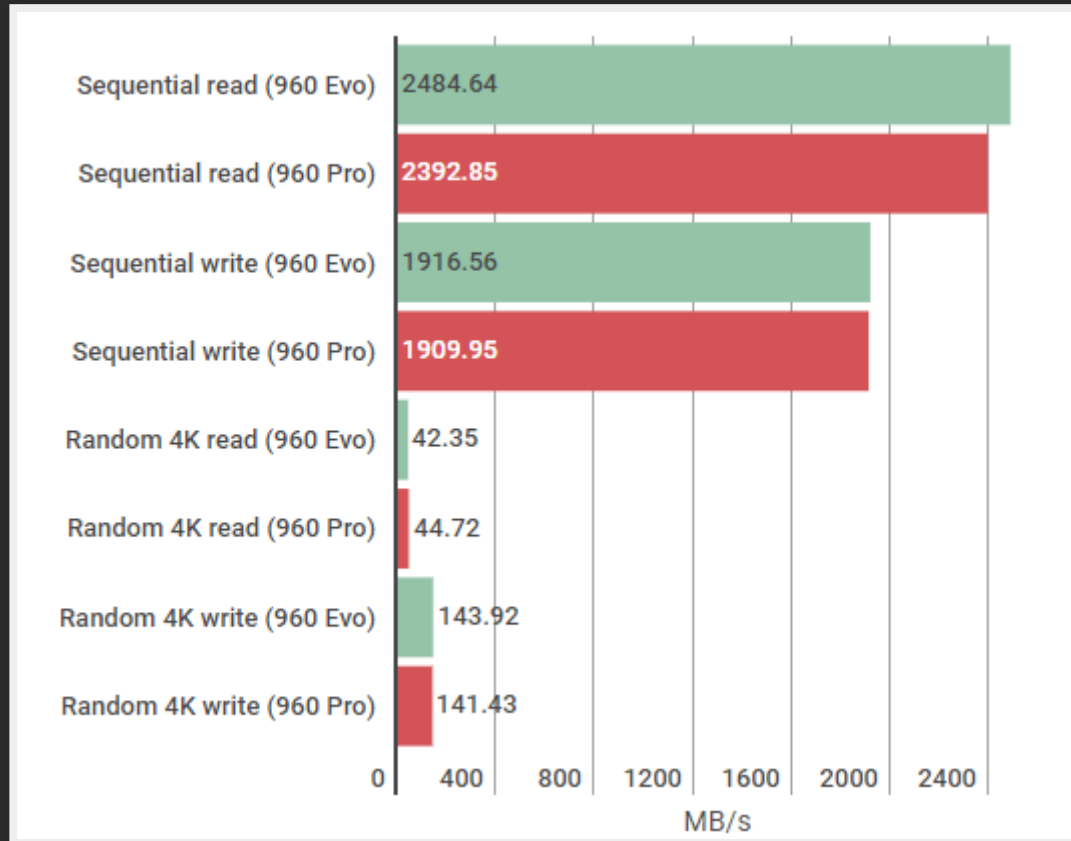
DATABASES IN DIFFERENT SCENARIOS



B-TREE: A STANDARD INDEX IMPLEMENTATION FOR RELATIONAL DATABASE



LIES, DAMNED LIES AND BENCHMARKS



Benchmark Results of Samsung 960 Evo NVMe SSD

How can we transform random writes to sequential writes?

LOG-STRUCTURED MERGE-TREE

Used in a range of modern databases

- **Key-Value Stores:** BigTable, Cassandra, HBase, LevelDB, Riak, RocksDB, WiredTiger
- **Time-Series Databases:** InfluxDB
- **Relational Databases:** SQLite4, MyRocks

SSTable: Sorted String Table

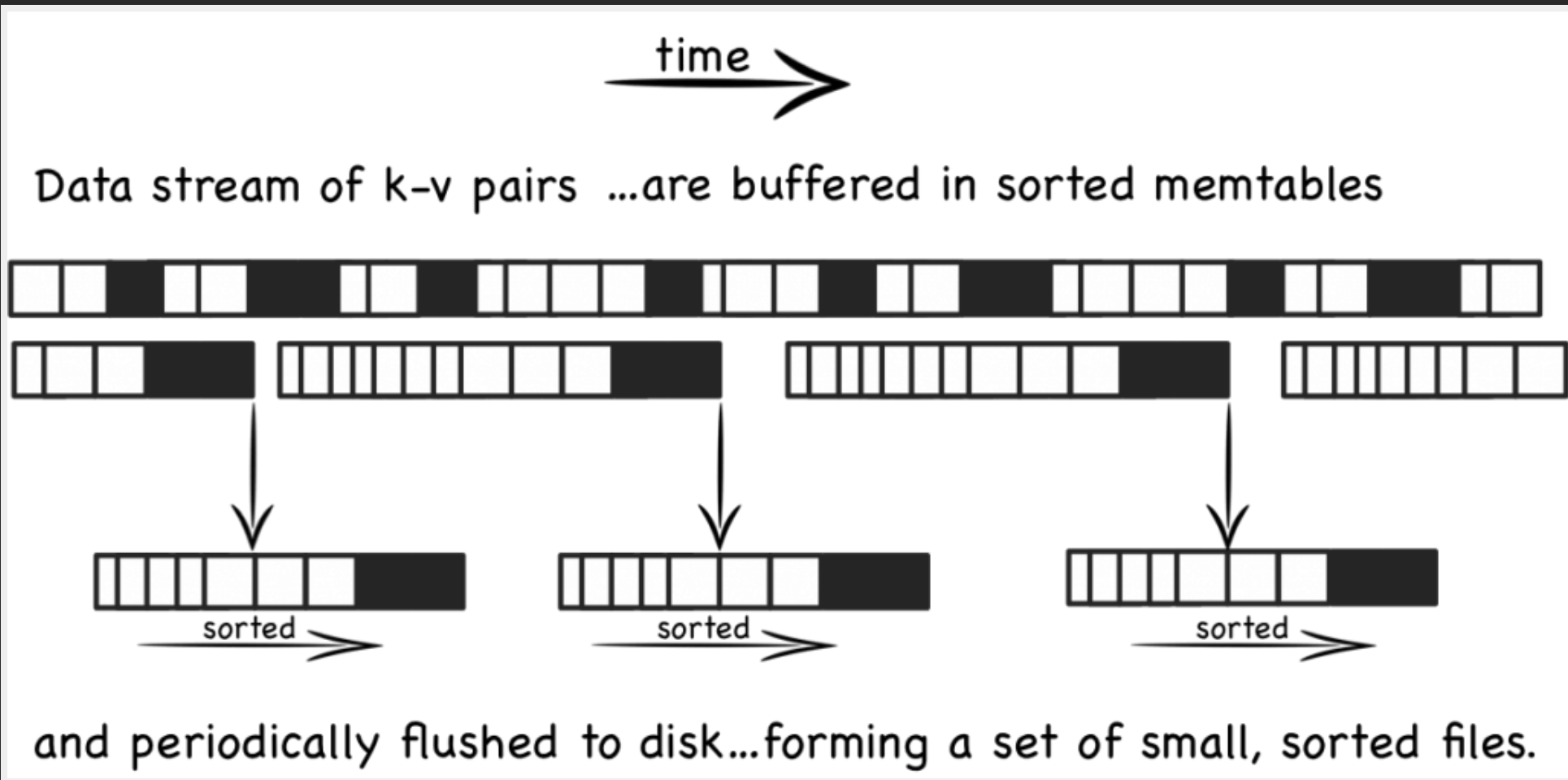
Index

key	offset
key	offset
...	...

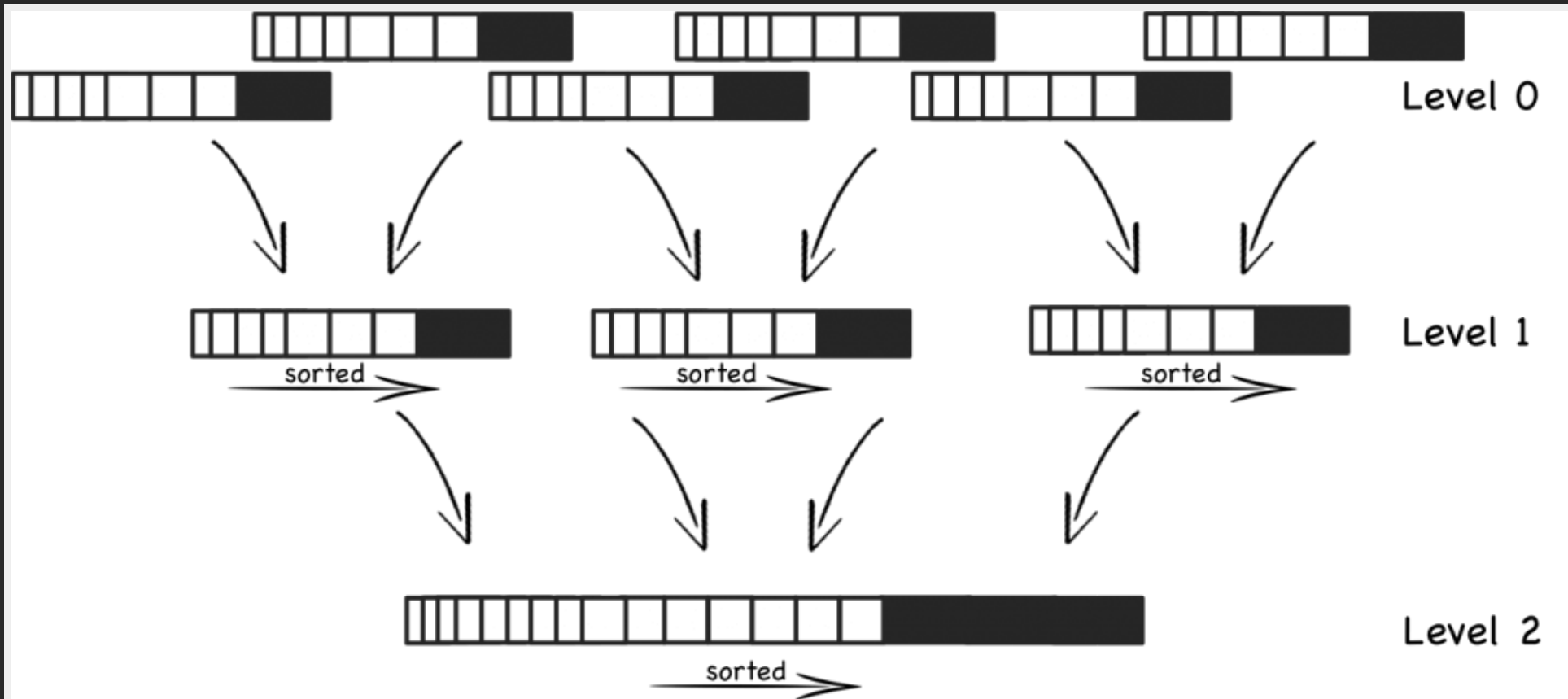
SSTable file

key	value	key	value	key	value
-----	-------	-----	-------	-----	-------	-----	-----

Writes are collected in memory
then sort and flush to disk



Compaction: sort and merge



Compaction continues creating fewer, larger and larger files

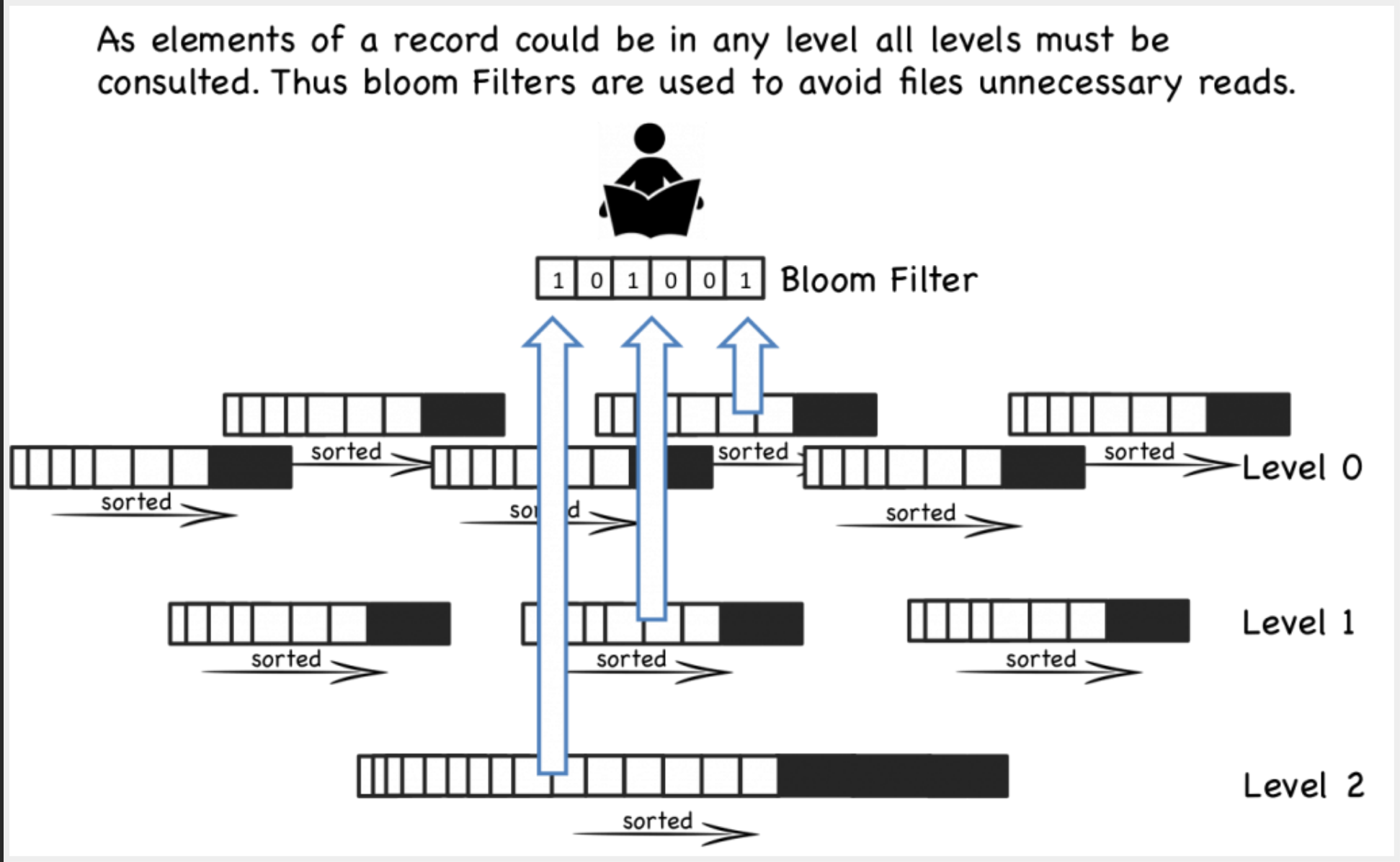
Optimizing reads is easier than optimising writes

As elements of a record could be in any level all levels must be consulted. Thus bloom Filters are used to avoid files unnecessary reads.

The diagram illustrates a multi-level index structure. At the top, a user icon is shown reading a document. Below it, a Bloom Filter is represented as a binary array:

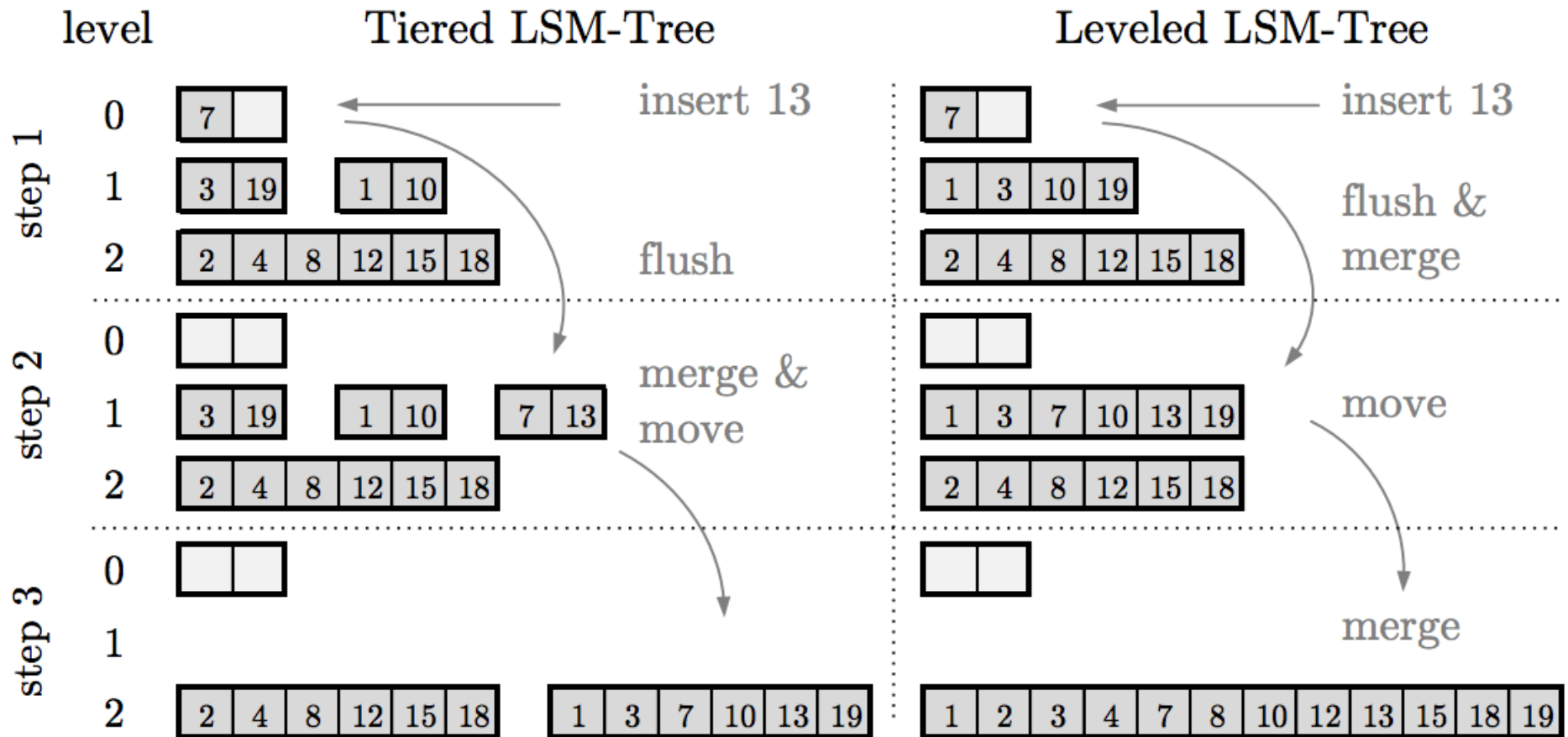
1	0	1	0	0	1
---	---	---	---	---	---

 with the label "Bloom Filter" to its right. Three blue arrows point upwards from the index levels to the Bloom Filter. The index structure consists of three levels: Level 0, Level 1, and Level 2. Level 0 is the top level, containing three blocks of data. Level 1 is the middle level, containing three blocks of data. Level 2 is the bottom level, containing one large block of data. Each block is represented as a horizontal array of cells, some white and some black. Arrows labeled "sorted" indicate the flow of data from Level 2 to Level 1, and from Level 1 to Level 0. The blue arrows indicate that the Bloom Filter is used to filter out unnecessary reads by checking the bits in the filter against the data in the index blocks.



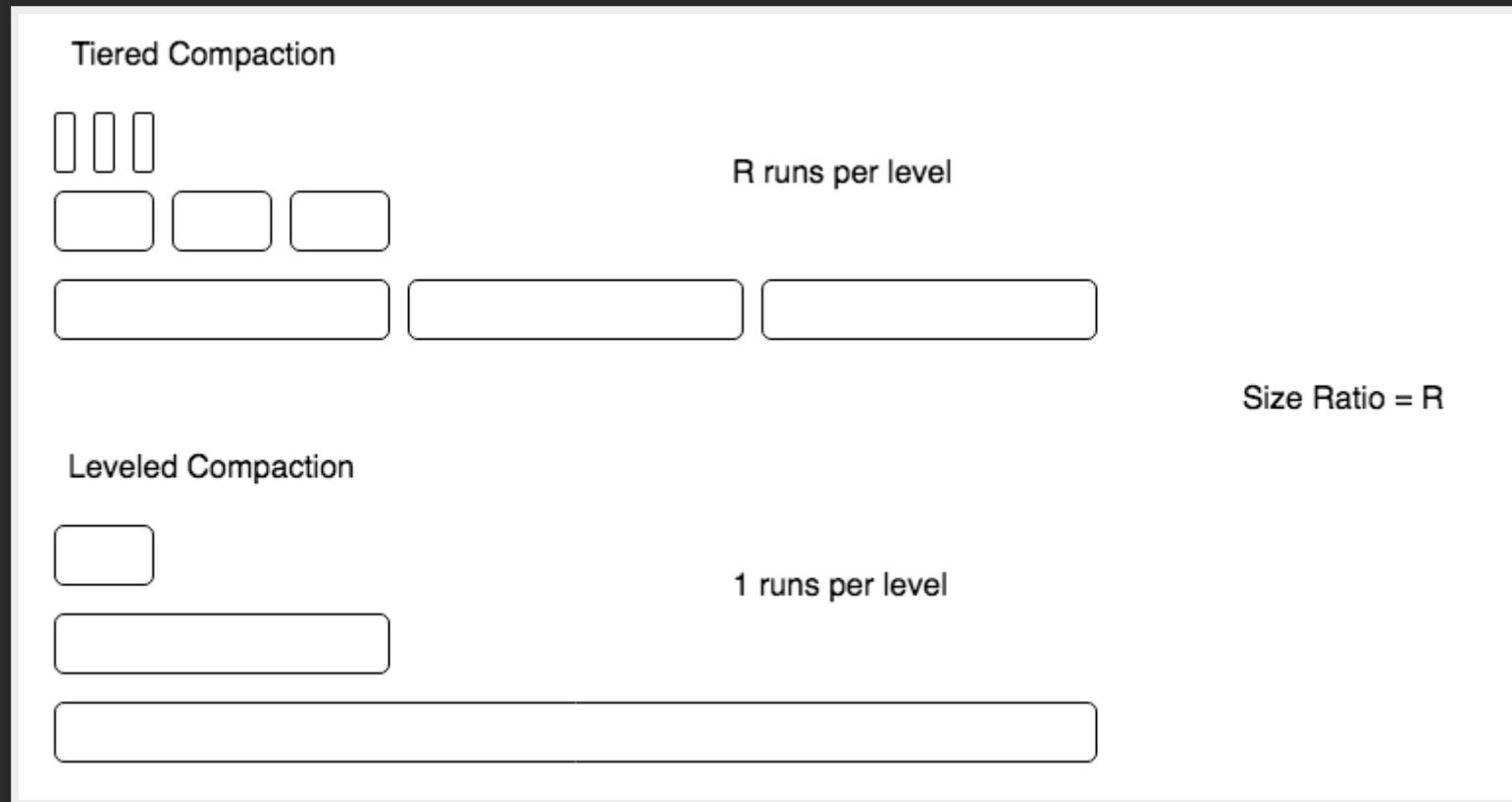
COMPACTION STRATEGIES

Name	Feature	Representative
Size-Tiered Compaction	Update Optimized	Cassandra
Leveled Compaction	Lookup Optimized	RocksDB

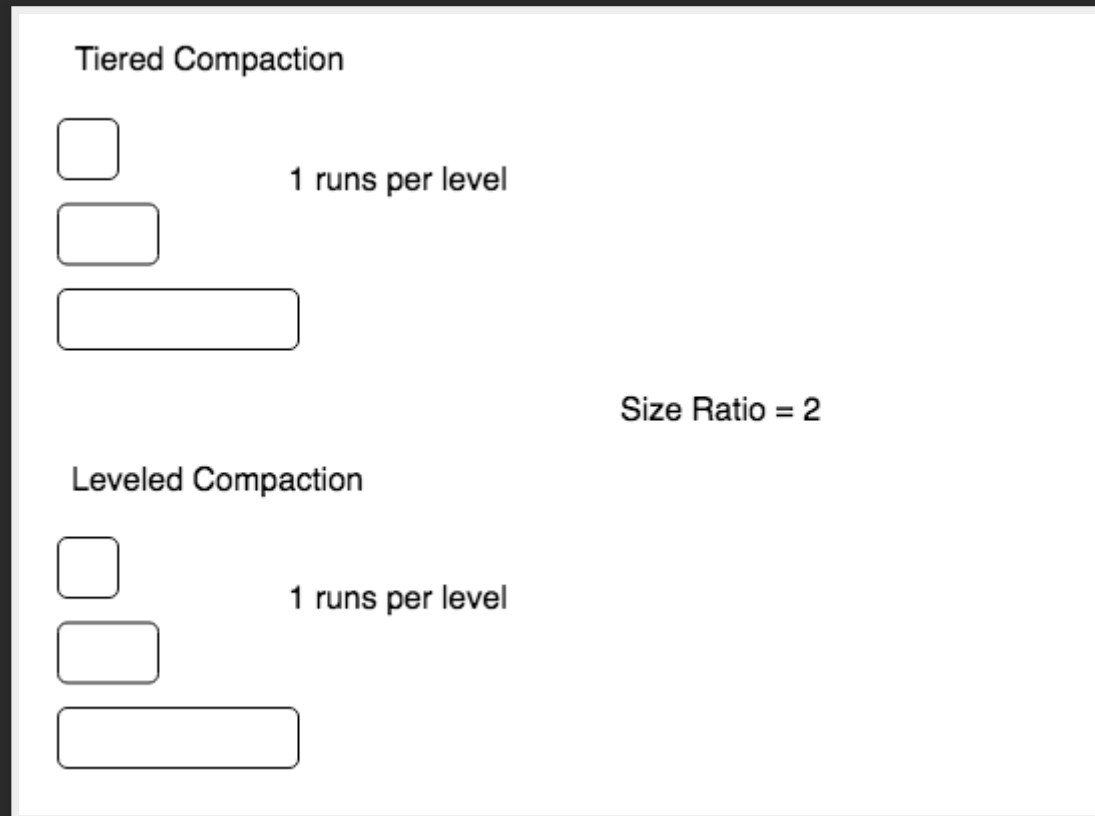


- Level 0 is the buffer which in memory
- Level 1 and 2 are on disk

Size-ratio: Ratio between the capacities of different levels



LSM-tree with a really small size-ratio



LSM-tree with a really high size-ratio

Tiered Compaction



N runs per level

log

Size Ratio = ∞

Leveled Compaction



1 runs per level

sorted array

Trade-off Curve

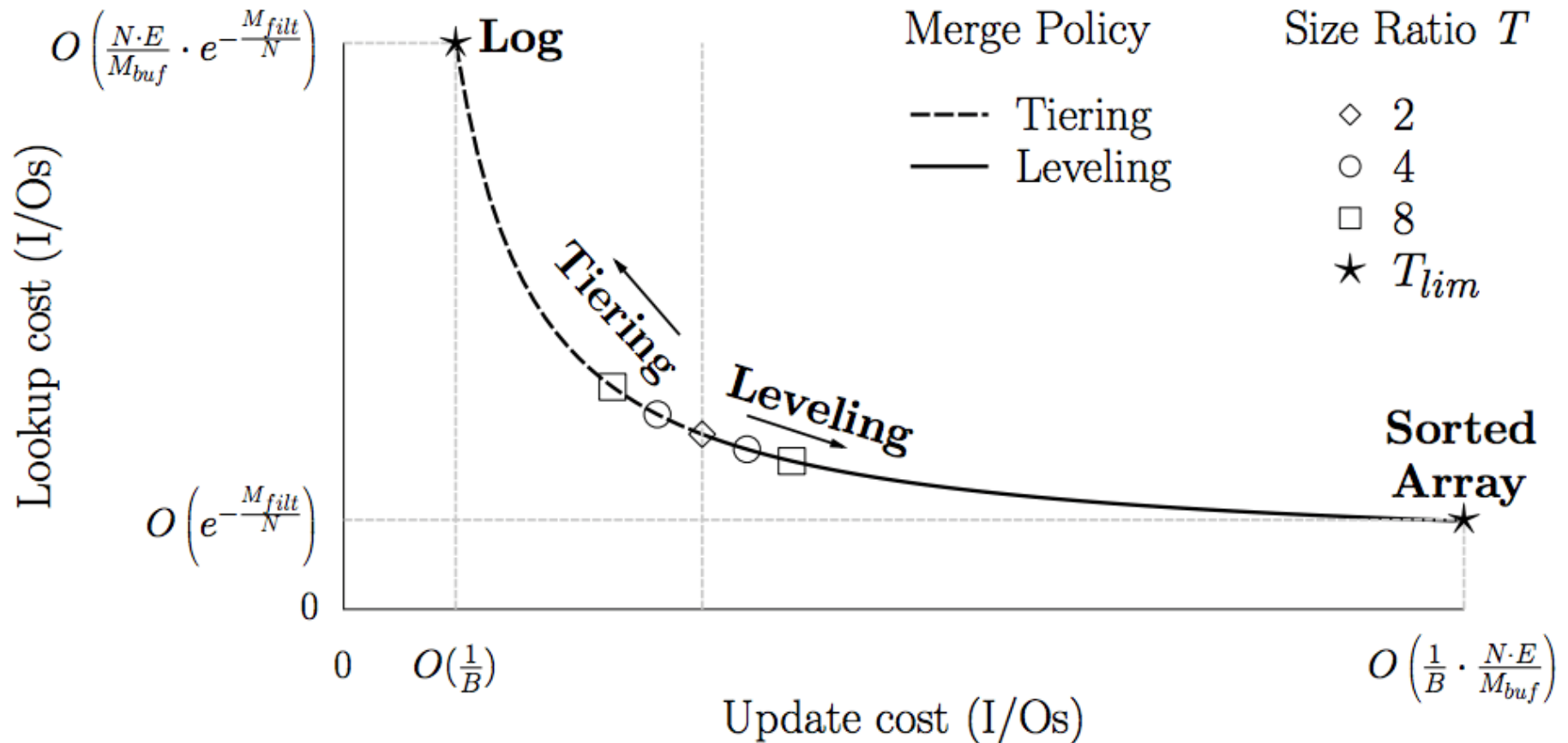
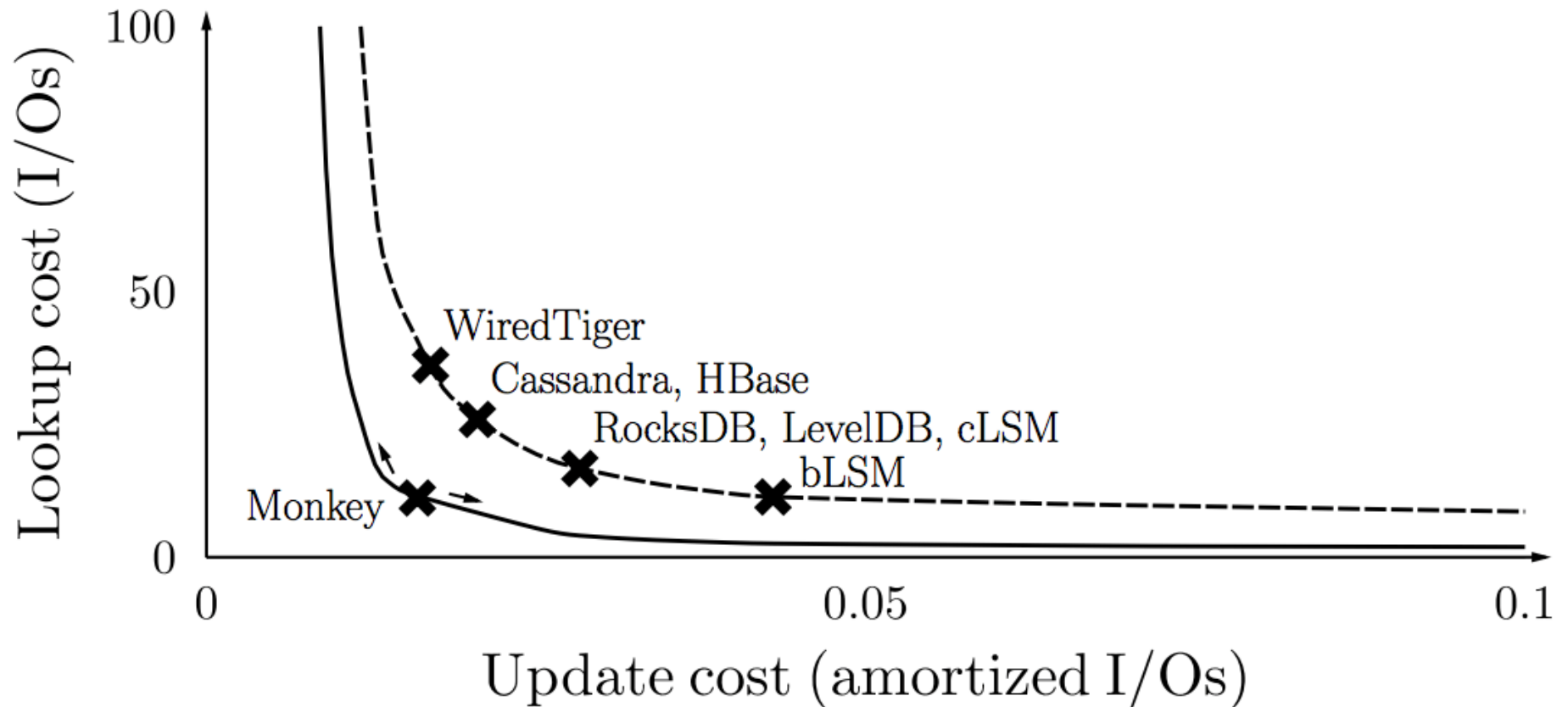


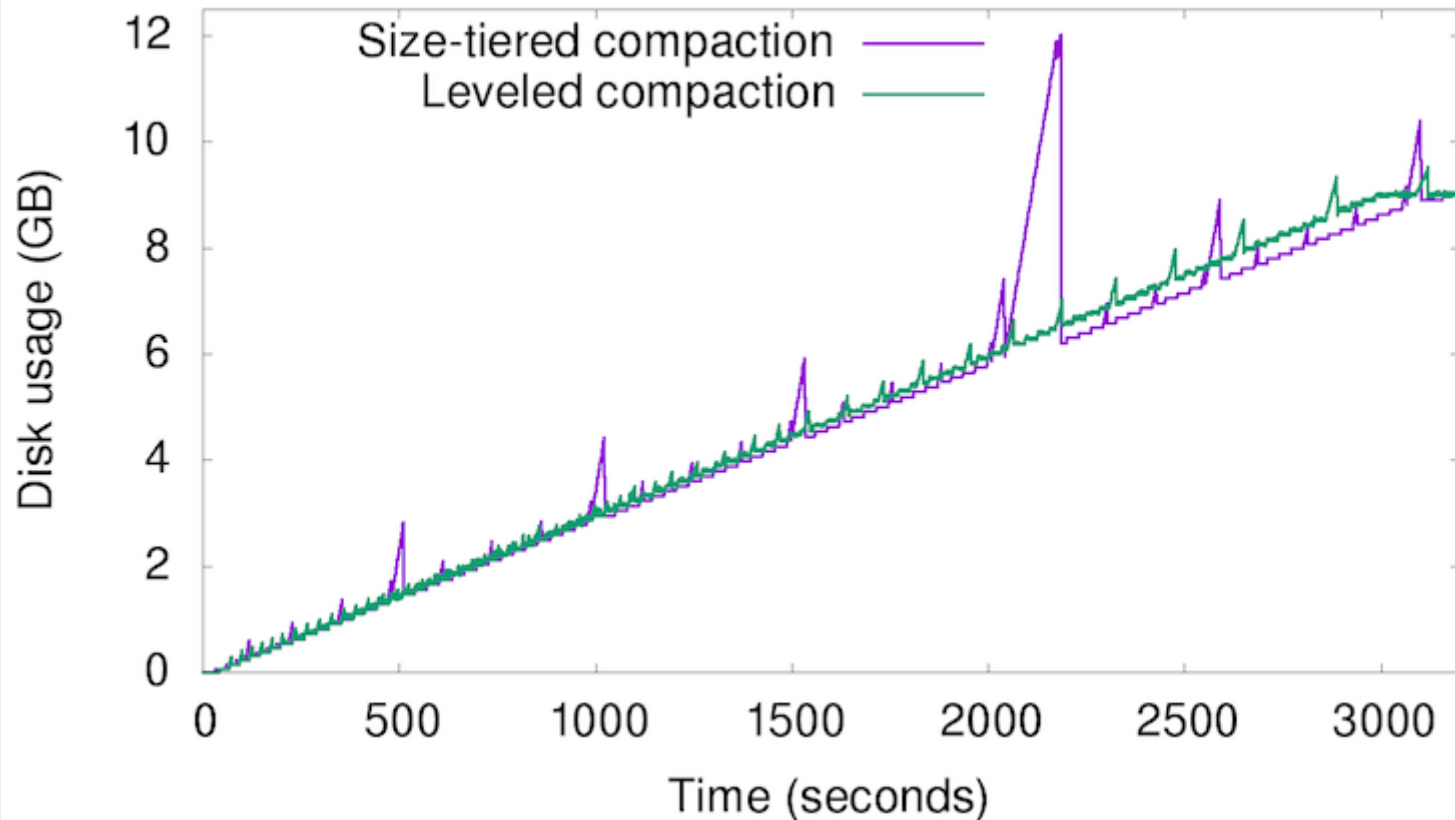
Fig. 4. LSM-tree design space: from a log to a sorted array.

The state of the art on Trade-off Curve



WEAKNESSES OF LSM-TREE

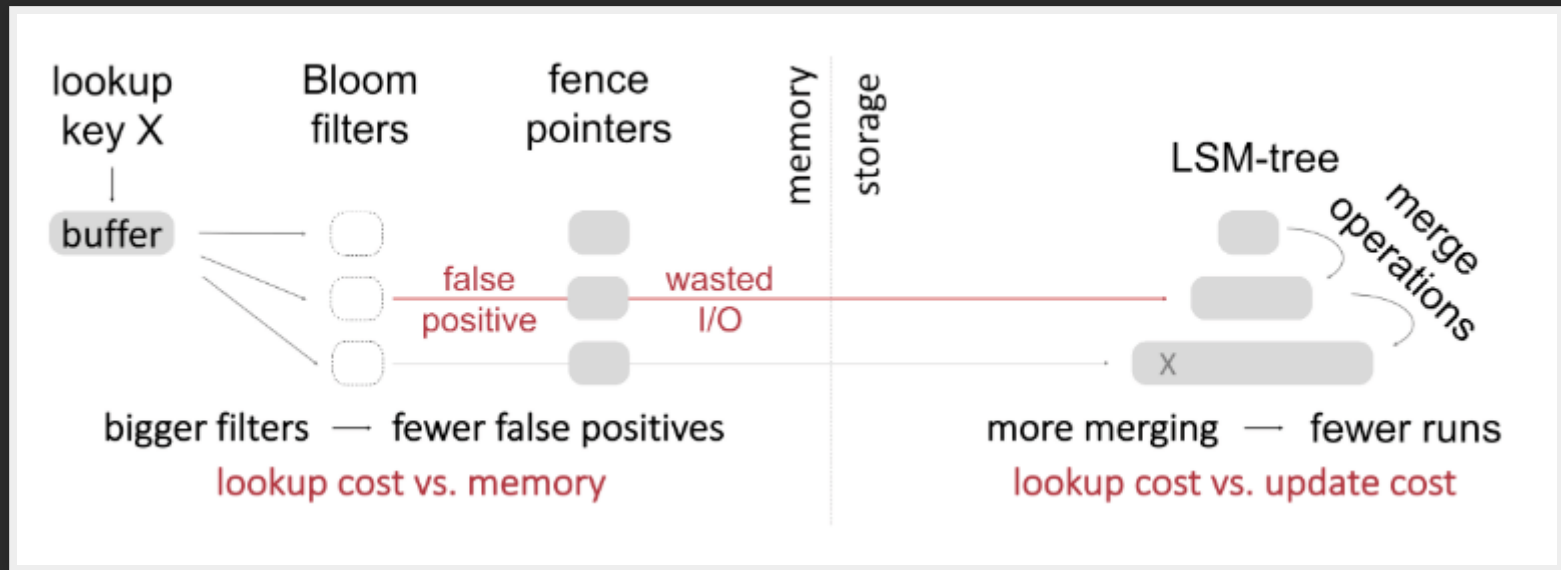
Space Amplification and Write Amplification



LSM TUNING










- **Monkey** by Harvard DASlab, 2017
- **Dostoevsky** by Harvard DASlab, 2018
- **Universal Compaction** by Facebook RocksDB, 2018
- ...

Monkey: Optimal Navigable Key-Value Store



Monkey improves read performance (50-90% lookup cost in the worst-case) by optimal allocation of memory to the bloom filters

Dostoevsky: Space-Time Optimized Evolvable Scalable Key-Value Store

	tiering			leveling			lazy leveling					
level	L-2	L-1	L	L-2	L-1	L	L-2	L-1	L			
(A) runs per level												
(B) zero-result point lookup cost	$\dots 0 \left(\frac{e^{-M/N}}{T^1} \right) + 0 \left(\frac{e^{-M/N}}{T^0} \right) + 0 \left(\frac{e^{-M/N}}{T^{-1}} \right) = \mathbf{0}(e^{-M/N} \cdot T)$			$\dots 0 \left(\frac{e^{-M/N}}{T^2} \right) + 0 \left(\frac{e^{-M/N}}{T^1} \right) + 0 \left(\frac{e^{-M/N}}{T^0} \right) = \mathbf{0}(e^{-M/N})$			$\dots 0 \left(\frac{e^{-M/N}}{T^2} \right) + 0 \left(\frac{e^{-M/N}}{T^1} \right) + 0 \left(\frac{e^{-M/N}}{T^0} \right) = \mathbf{0}(e^{-M/N})$					
(C) point lookup cost to existing entries	$\dots 0 \left(\frac{e^{-M/N}}{T^1} \right) + 0 \left(\frac{e^{-M/N}}{T^0} \right) + 0 \left(1 + \frac{e^{-M/N}}{T^{-1}} \right) = \mathbf{0}(1 + e^{-M/N} \cdot T)$			$\dots 0 \left(\frac{e^{-M/N}}{T^2} \right) + 0 \left(\frac{e^{-M/N}}{T^1} \right) + 0(1) = \mathbf{0}(1)$			$\dots 0 \left(\frac{e^{-M/N}}{T^2} \right) + 0 \left(\frac{e^{-M/N}}{T^1} \right) + 0(1) = \mathbf{0}(1)$					
(D) short range lookup cost	$\dots 0(T)$	$+0(T)$	$+0(T)$	$= \mathbf{0}(L \cdot T)$			$\dots 0(T)$	$+0(T)$	$+0(1)$	$= \mathbf{0}(1 + (L-1) \cdot T)$		
(E) long range lookup cost	$\dots 0 \left(\frac{s}{T^1 \cdot B} \right)$	$+0 \left(\frac{s}{T^0 \cdot B} \right)$	$+0 \left(\frac{s}{T^{-1} \cdot B} \right)$	$= \mathbf{0} \left(\frac{T \cdot s}{B} \right)$			$\dots 0 \left(\frac{s}{T^2 \cdot B} \right)$	$+0 \left(\frac{s}{T^1 \cdot B} \right)$	$+0 \left(\frac{s}{T^0 \cdot B} \right)$	$= \mathbf{0} \left(\frac{s}{B} \right)$		
(F) update cost	$\dots 0 \left(\frac{1}{B} \right)$	$+0 \left(\frac{1}{B} \right)$	$+0 \left(\frac{1}{B} \right)$	$= \mathbf{0} \left(\frac{L}{B} \right)$			$\dots 0 \left(\frac{T}{B} \right)$	$+0 \left(\frac{T}{B} \right)$	$+0 \left(\frac{T}{B} \right)$	$= \mathbf{0} \left(\frac{L \cdot T}{B} \right)$		
(F) space-amplification	$\dots 0 \left(\frac{1}{T^2} \right)$	$+0 \left(\frac{1}{T^1} \right)$	$+0(T)$	$= \mathbf{0}(T)$			$\dots 0 \left(\frac{1}{T^2} \right)$	$+0 \left(\frac{1}{T^1} \right)$	$+0(0)$	$= \mathbf{0} \left(\frac{1}{T} \right)$		

Lazy Leveling and Cost breakdown

Let us talk it in simple way




- **Tiering** is great for: updates
- **Leveling** is great for: short range lookup
- **Lazy Leveling** is great for: updates & point lookup

Fluid LSM-tree and its three parameters

- T = size ratio
- K = runs at smaller levels
- Z = runs at largest level

If we set ...

- $Z=1, K=1$, it will be **Leveling**
- $Z=1, K=T-1$, it will be **Lazy Leveling**
- $Z=T-1, K=T-1$, it will be **Tiering**

level	L-2	L-1	L
(A) runs per level			
(B) zero-result point lookup cost	$\dots O\left(\frac{Z \cdot e^{-\frac{M}{N}}}{K \cdot T^2}\right)$	$+ O\left(\frac{Z \cdot e^{-\frac{M}{N}}}{K \cdot T^1}\right)$	$+ O\left(Z \cdot e^{-\frac{M}{N}}\right) = O(Z \cdot e^{-M/N})$
(C) point lookup cost to existing entries	$\dots O\left(\frac{Z \cdot e^{-\frac{M}{N}}}{K \cdot T^2}\right)$	$+ O\left(\frac{Z \cdot e^{-\frac{M}{N}}}{K \cdot T^1}\right)$	$+ O(1 + Z \cdot e^{-\frac{M}{N}}) = O(1 + Z \cdot e^{-M/N})$
(D) short range lookup cost	$\dots O(K)$	$+ O(K)$	$+ O(Z) = O(Z + (L-1) \cdot K)$
(E) long range lookup cost	$\dots O\left(\frac{s}{T^2 \cdot B}\right)$	$+ O\left(\frac{s}{T^1 \cdot B}\right)$	$+ O\left(\frac{Z \cdot s}{T^0 \cdot B}\right) = O\left(\frac{s \cdot Z}{B}\right)$
(F) update cost	$\dots O\left(\frac{T}{B \cdot K}\right)$	$+ O\left(\frac{T}{B \cdot K}\right)$	$+ O\left(\frac{T}{B \cdot Z}\right) = O\left(\frac{T}{B} \cdot \left(\frac{L}{K} + \frac{1}{Z}\right)\right)$
(G) space-amplification	$\dots O\left(\frac{1}{T^2}\right)$	$+ O\left(\frac{1}{T^1}\right)$	$+ O(Z-1) = O\left(z - 1 + \frac{1}{T}\right)$

REFERENCES

- [Designing Data-Intensive Applications](#), O'Reilly Book 2017
- [Log Structured Merge Trees](#), ben stopford Blog
- [Power of the Log: LSM & Append Only Data Structures](#), QCon 2017
- [Monkey](#): Optimal Navigable Key-Value Store, ACM SIGMOD 2017
- [Dostoevsky](#): Better Space-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging, ACM SIGMOD 2018
- [Universal Compaction](#), RocksDB Wiki

THANK YOU

Q&A