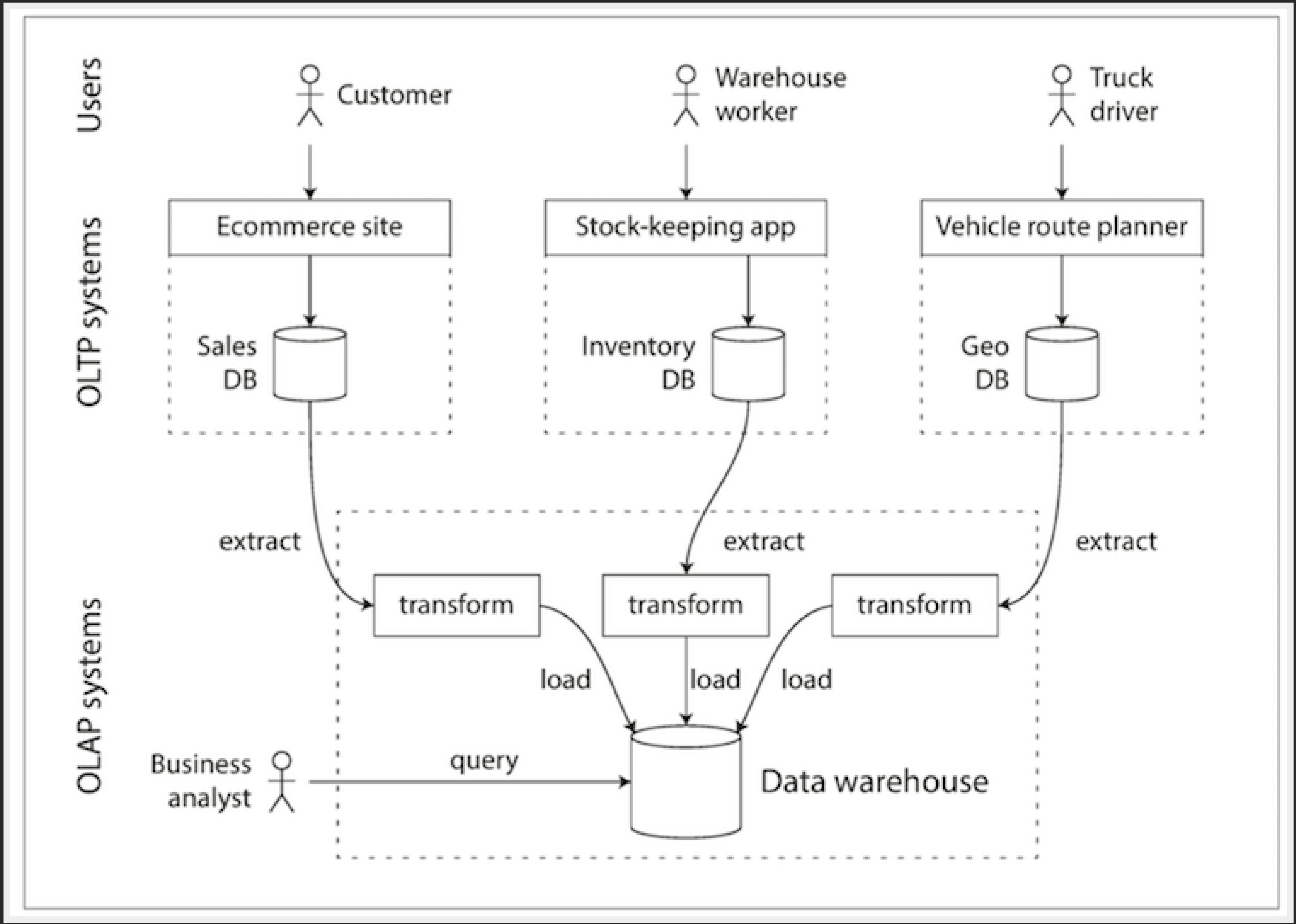
LSM-TREE AND TUNING

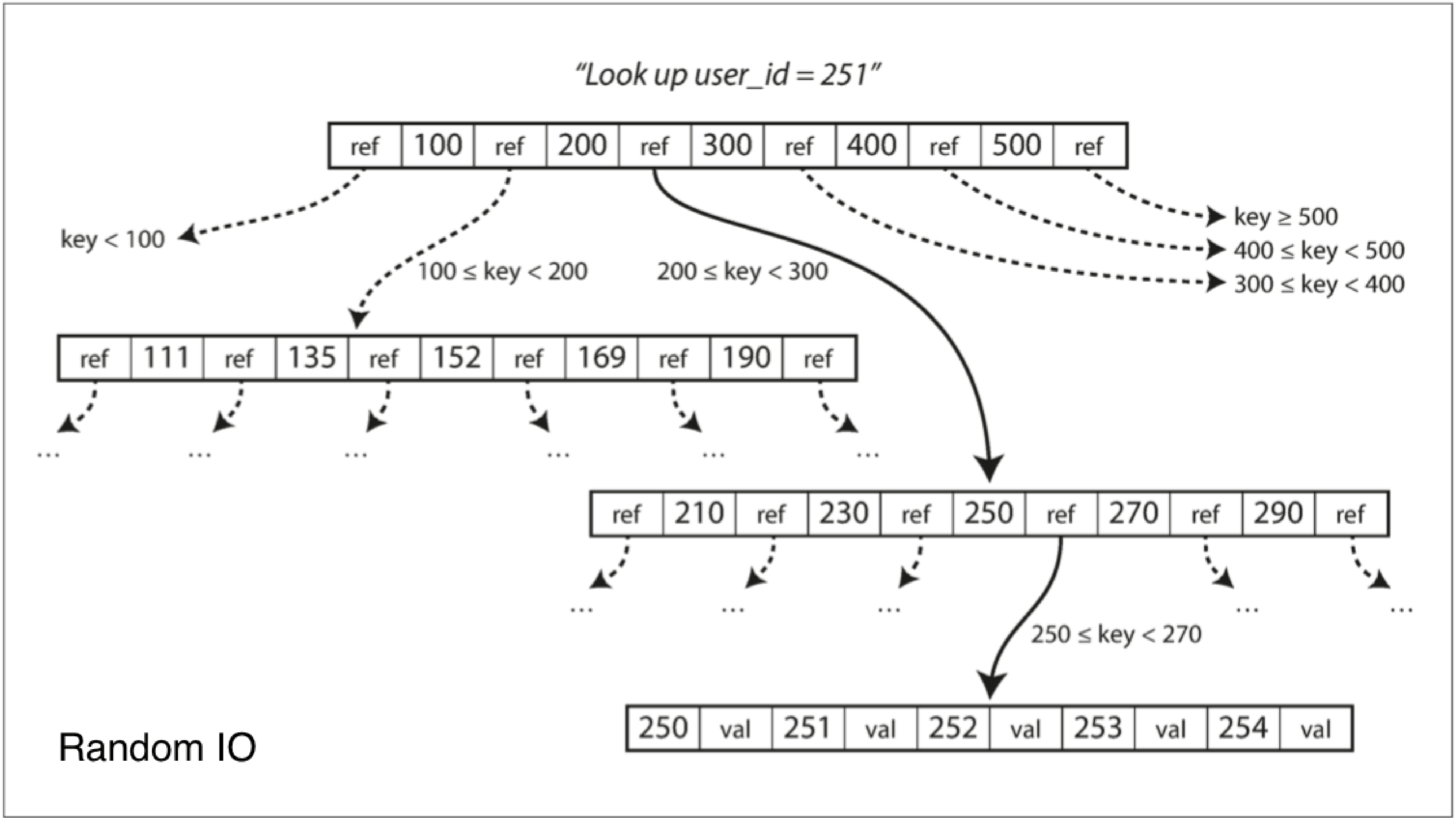
PREFACE

> *"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."*

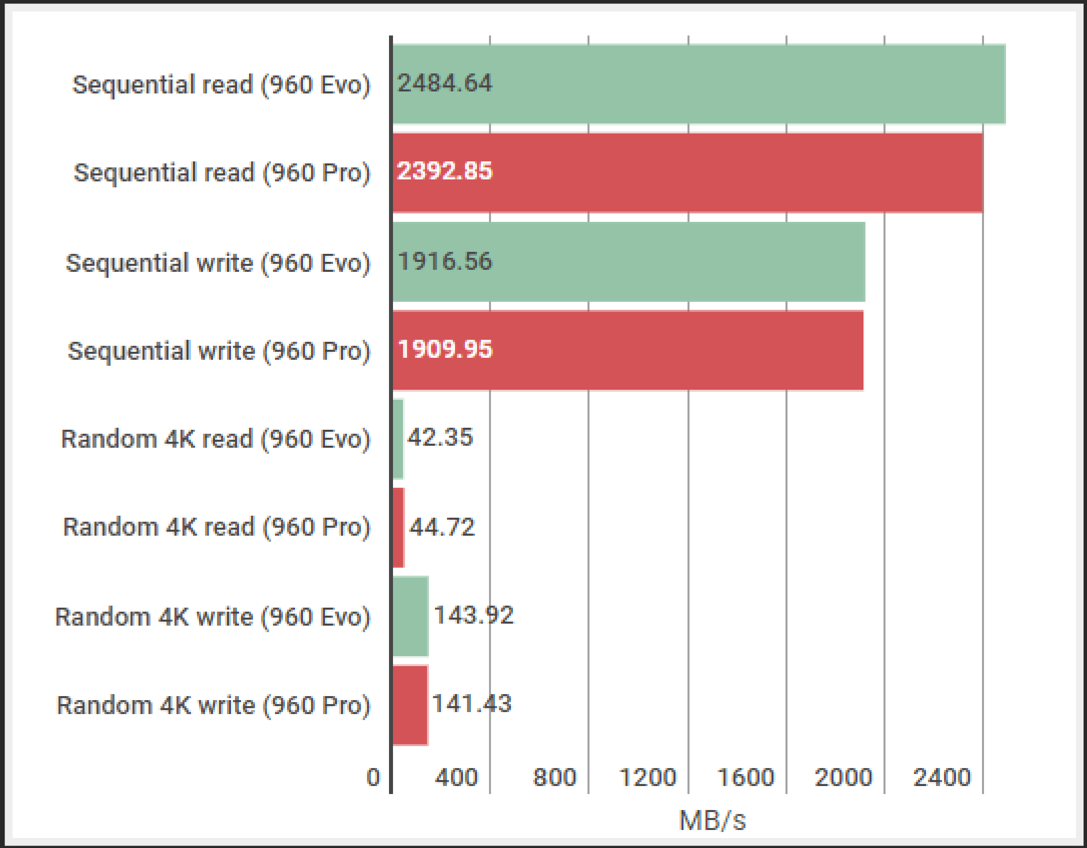
DATABASES IN DIFFERENT SCENARIOS



B-TREE: A STANDARD INDEX IMPLEMENTATION FOR RELATIONAL DATABASE



LIES, DAMNED LIES AND BENCHMARKS



Benchmark Results of Samsung 960 Evo NVMe SSD

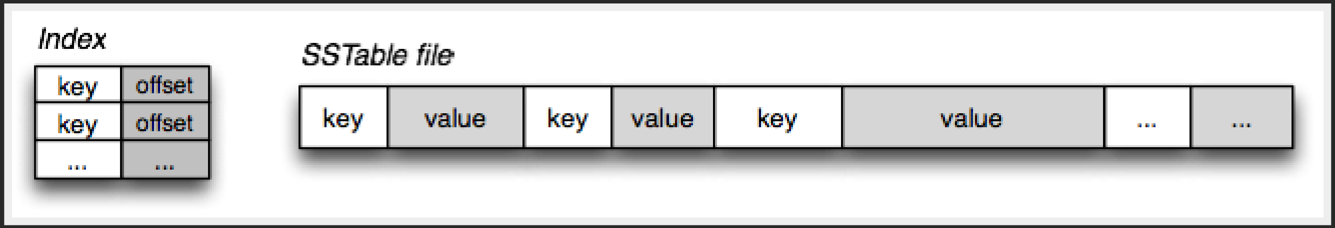
How can we transform random writes to sequential writes?

LOG-STRUCTURED MERGE-TREE

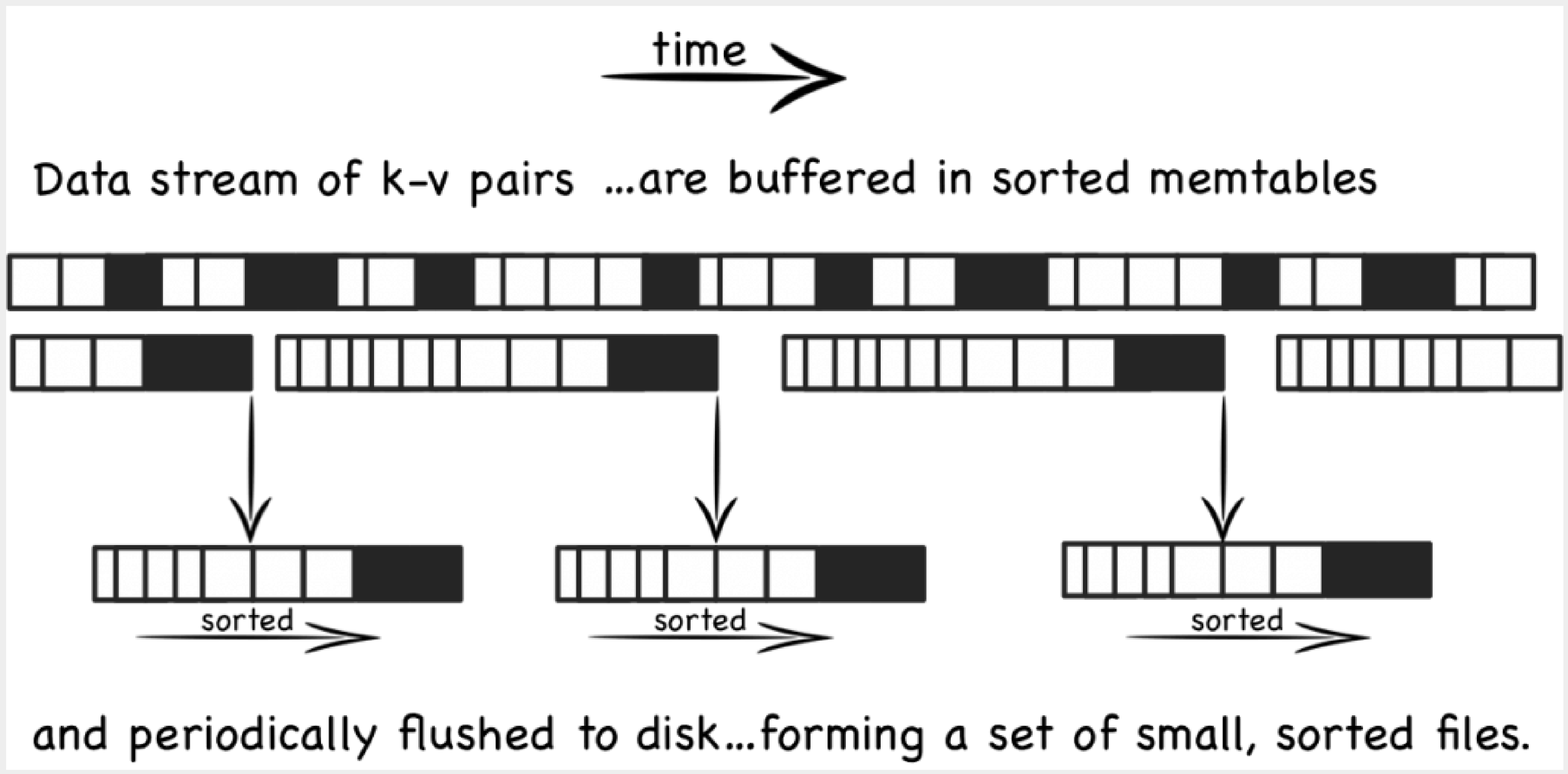
Used in a range of modern databases

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

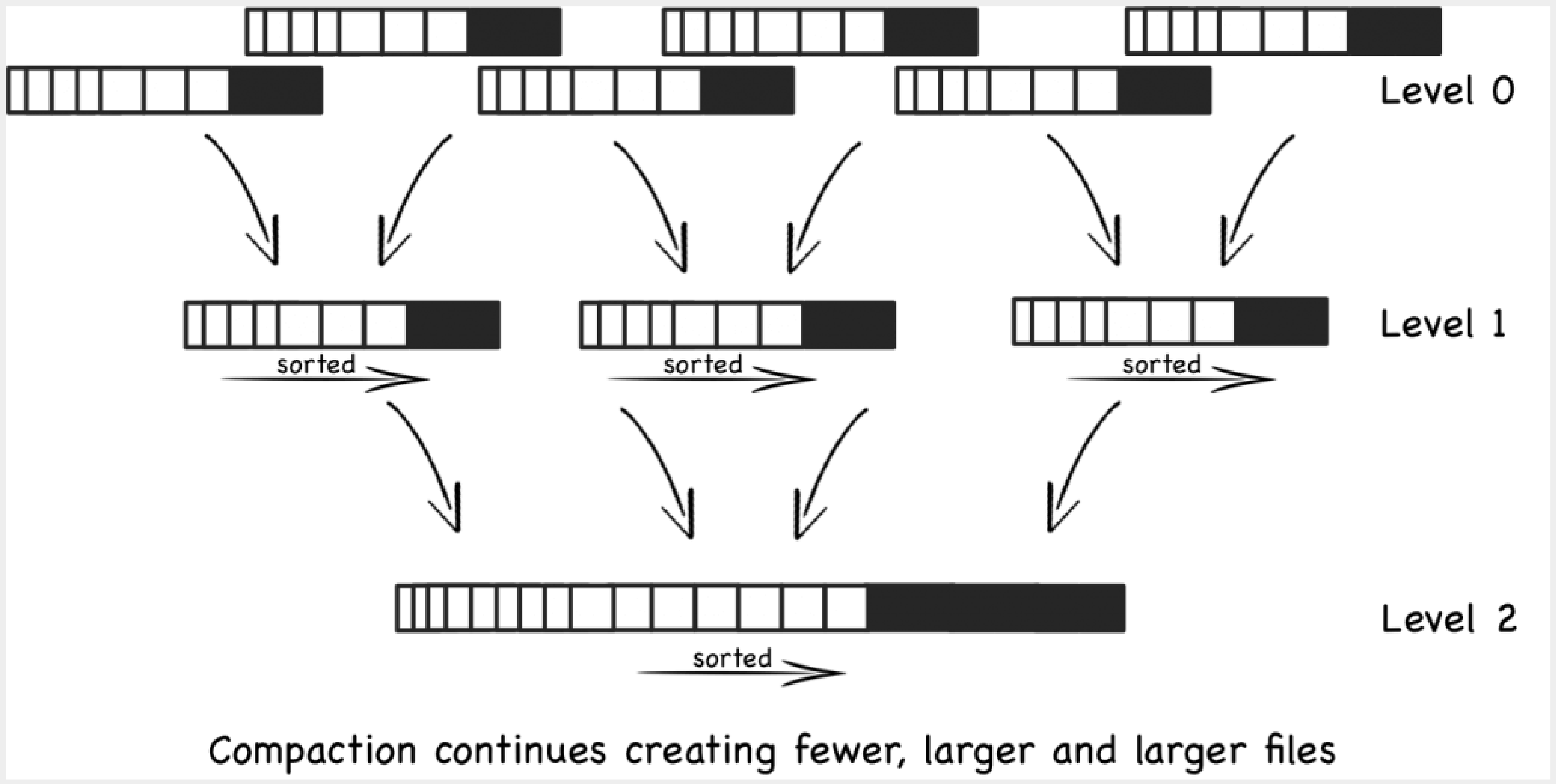
SSTable: Sorted String Table



Writes are collected in memory then sort and flush to disk

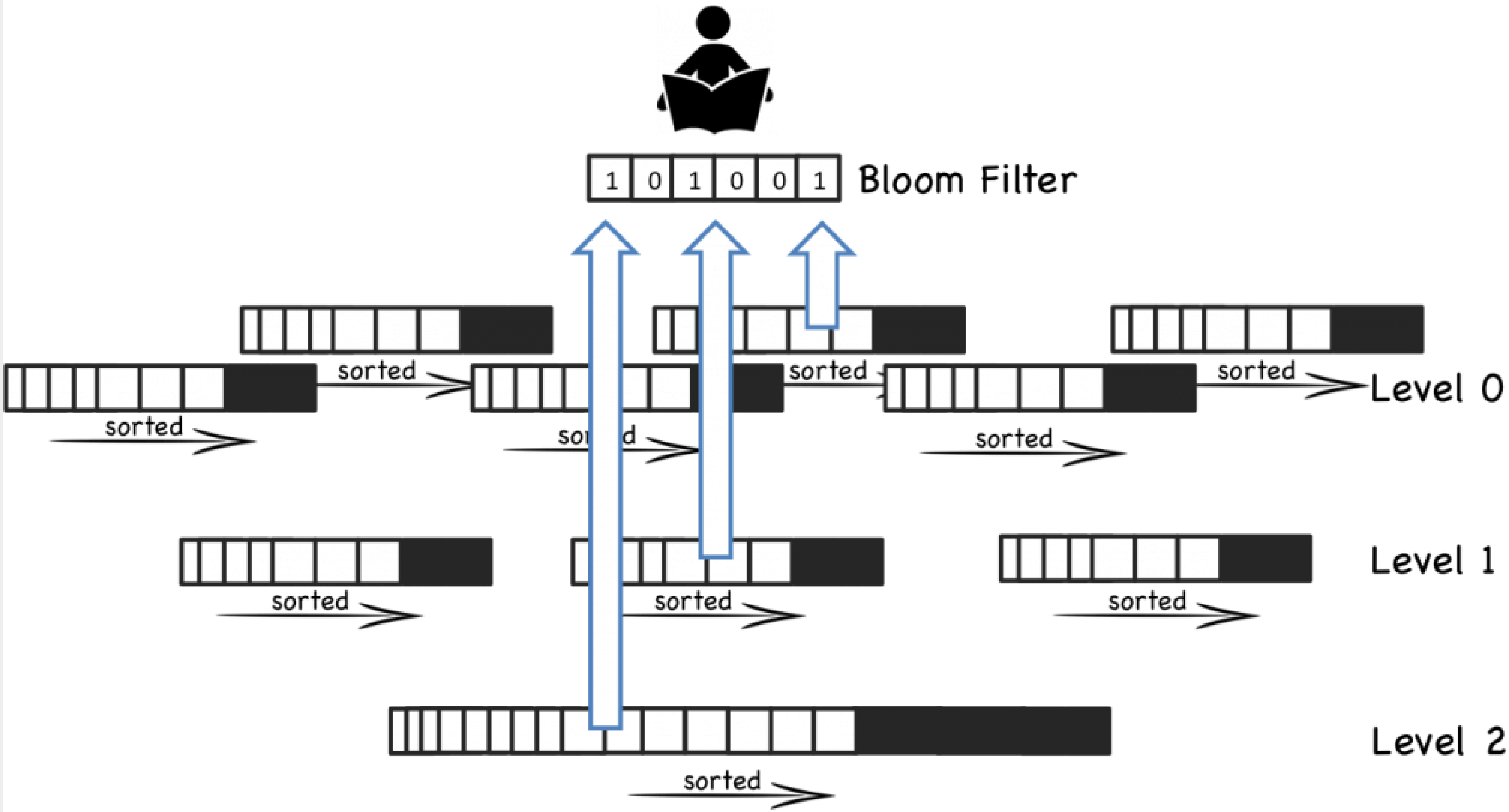


Compaction: sort and merge



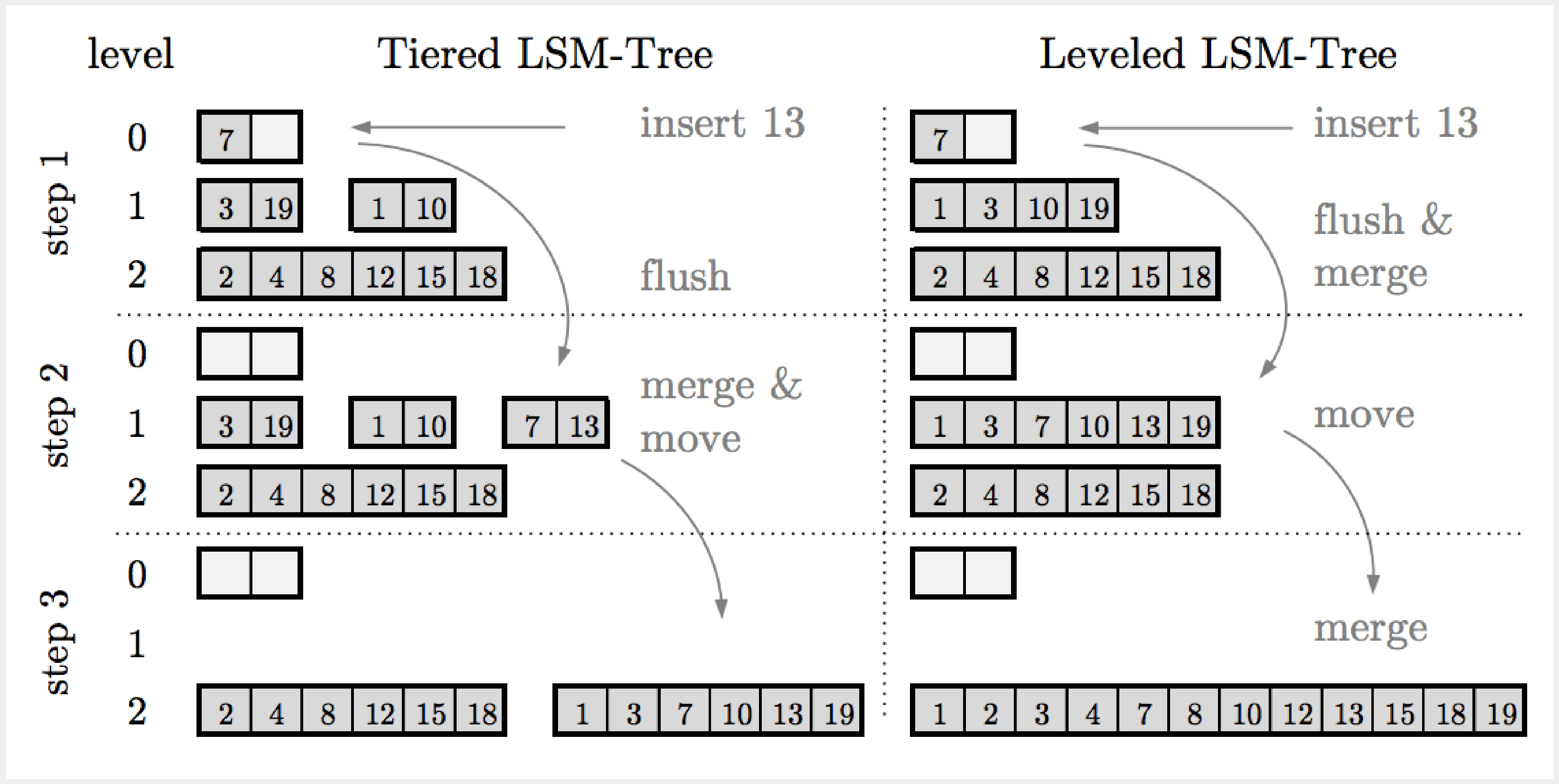
Optimizing reads is easier than optimising writes

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.



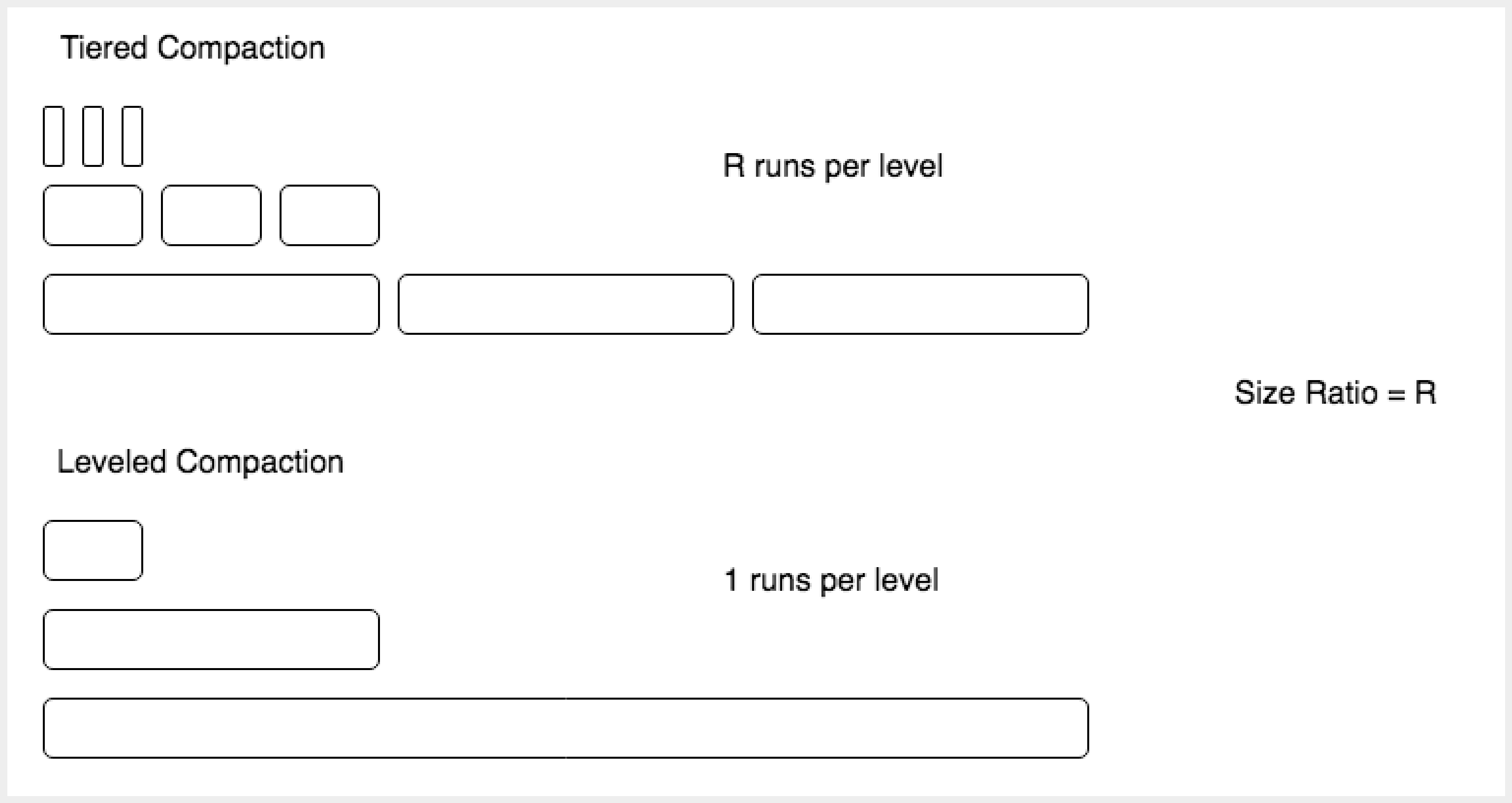
COMPACTION STRATEGIES

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

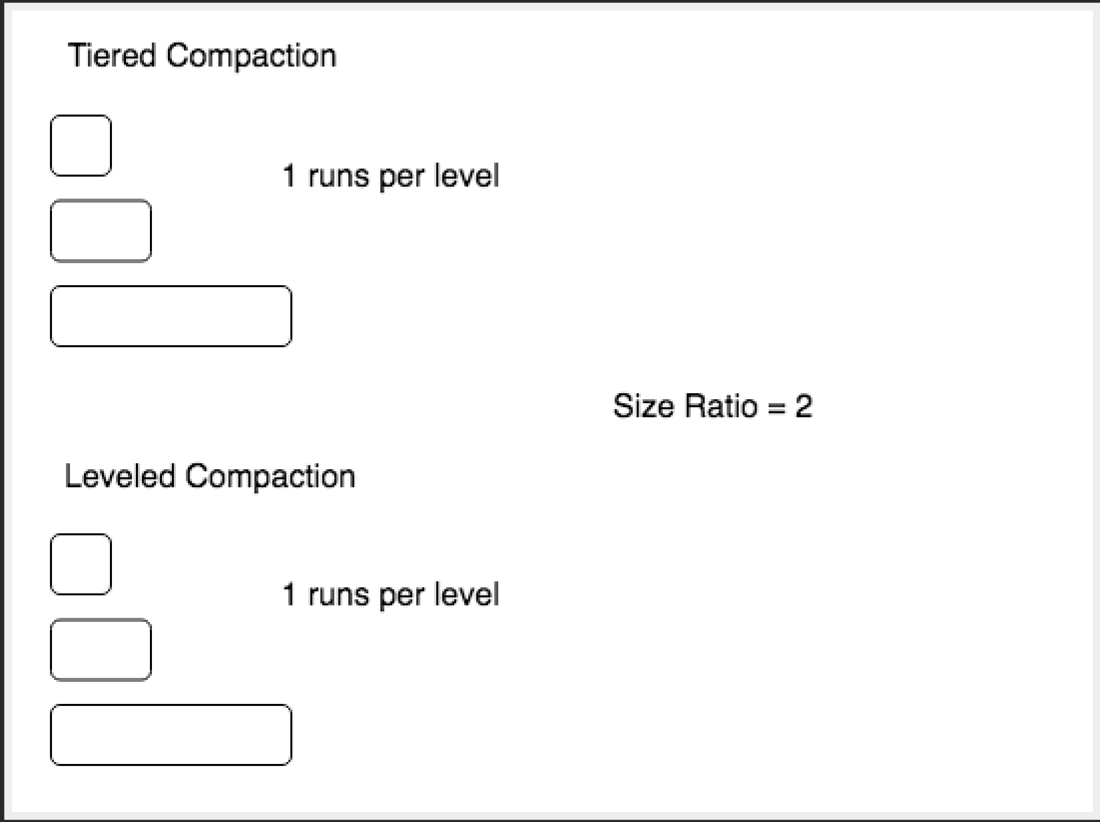


- Level 0 is the buffer which in memory
- Level 1 and 2 are on disk

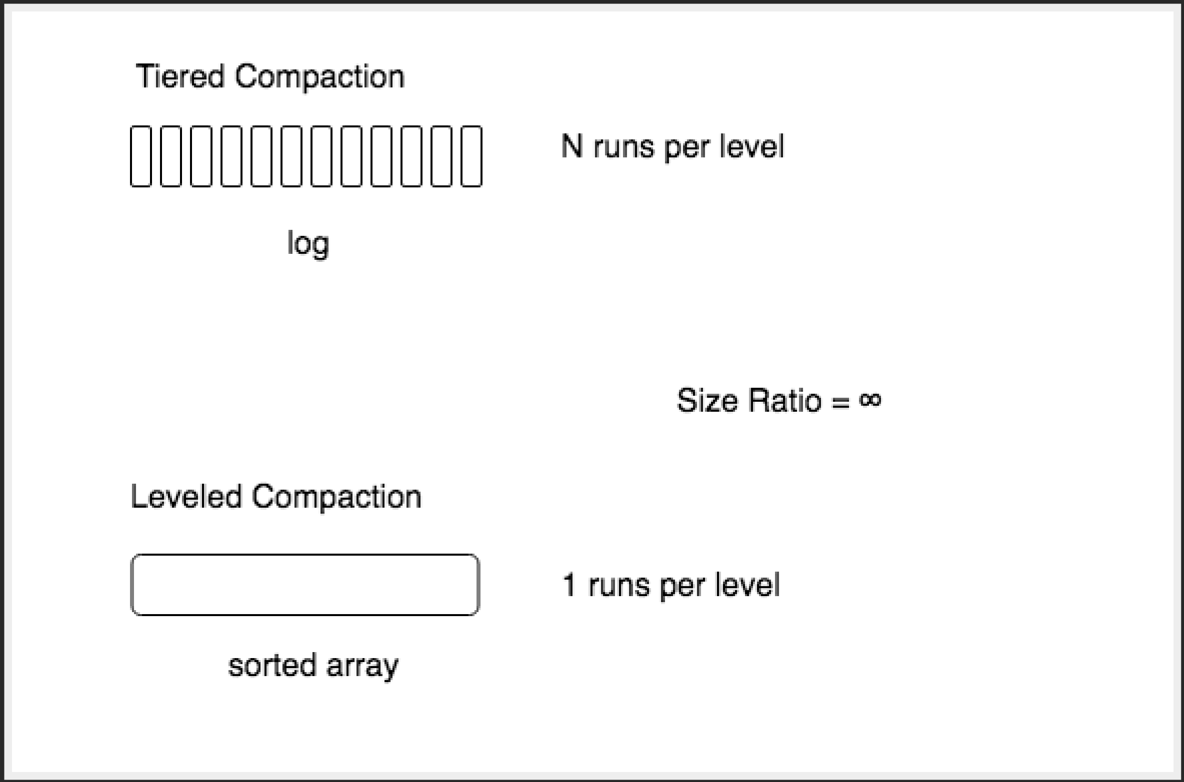
Size-ratio: Ratio between the capacities of different levels



LSM-tree with a really small size-ratio



LSM-tree with a really high size-ratio



Trade-off Curve

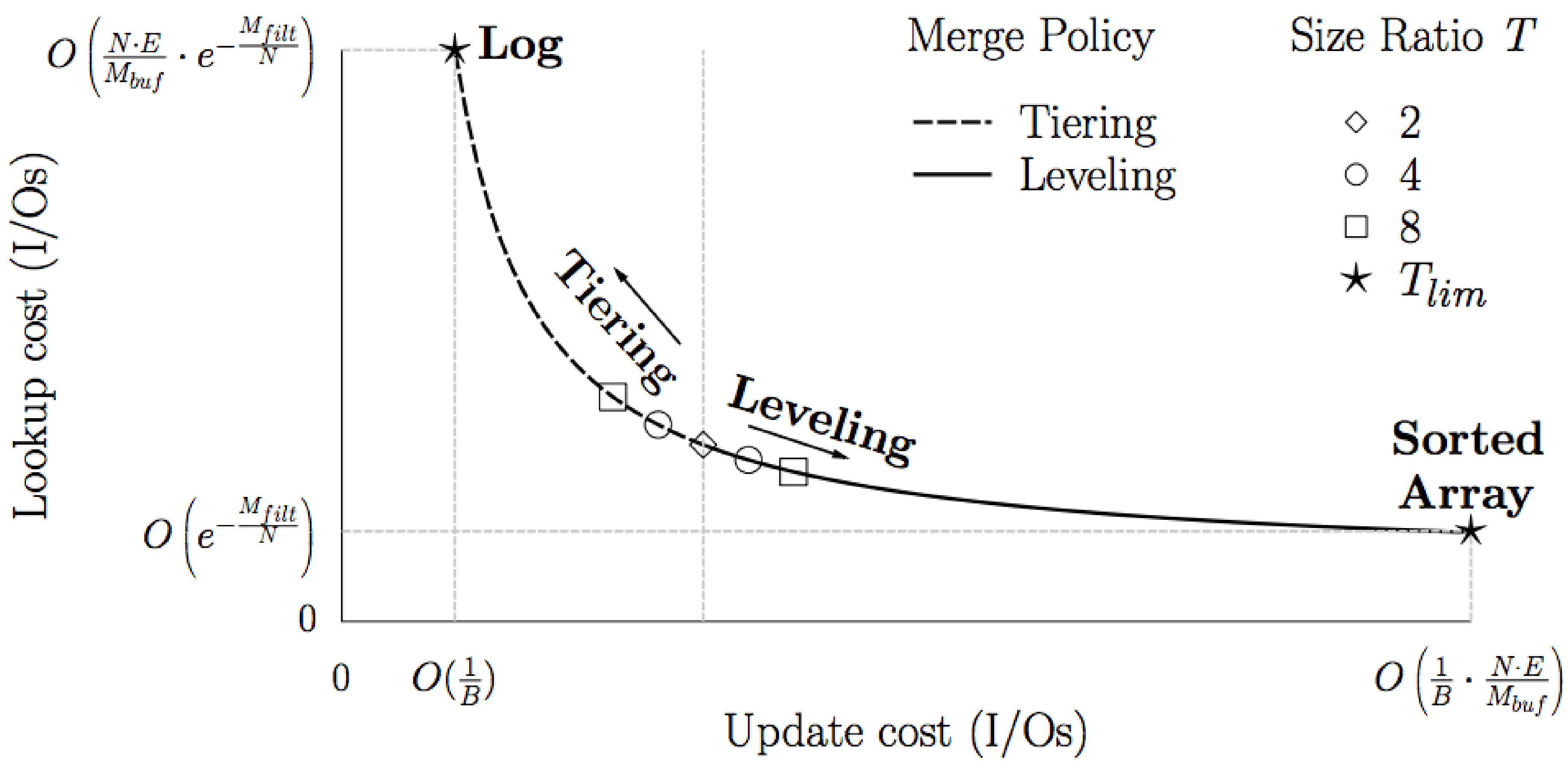
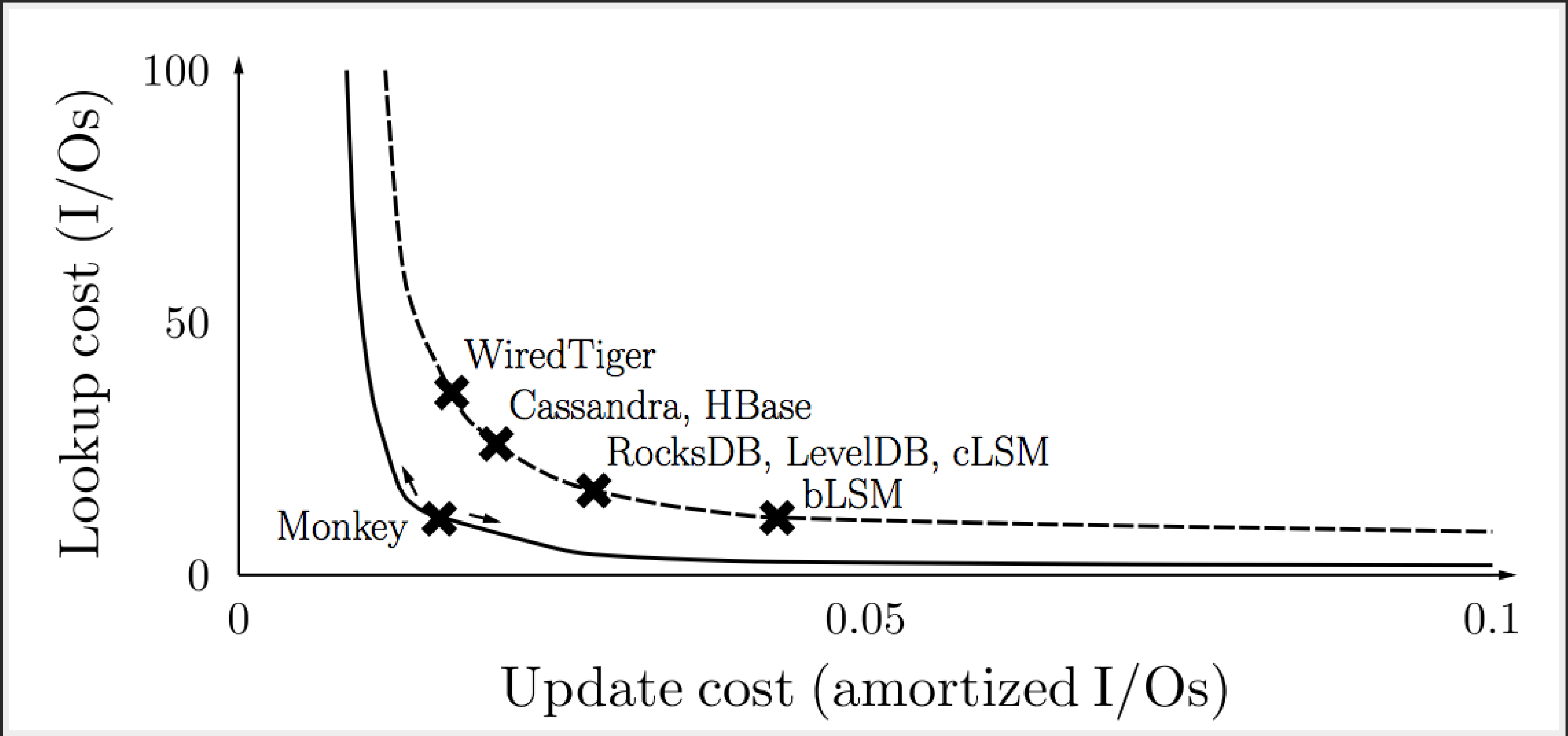


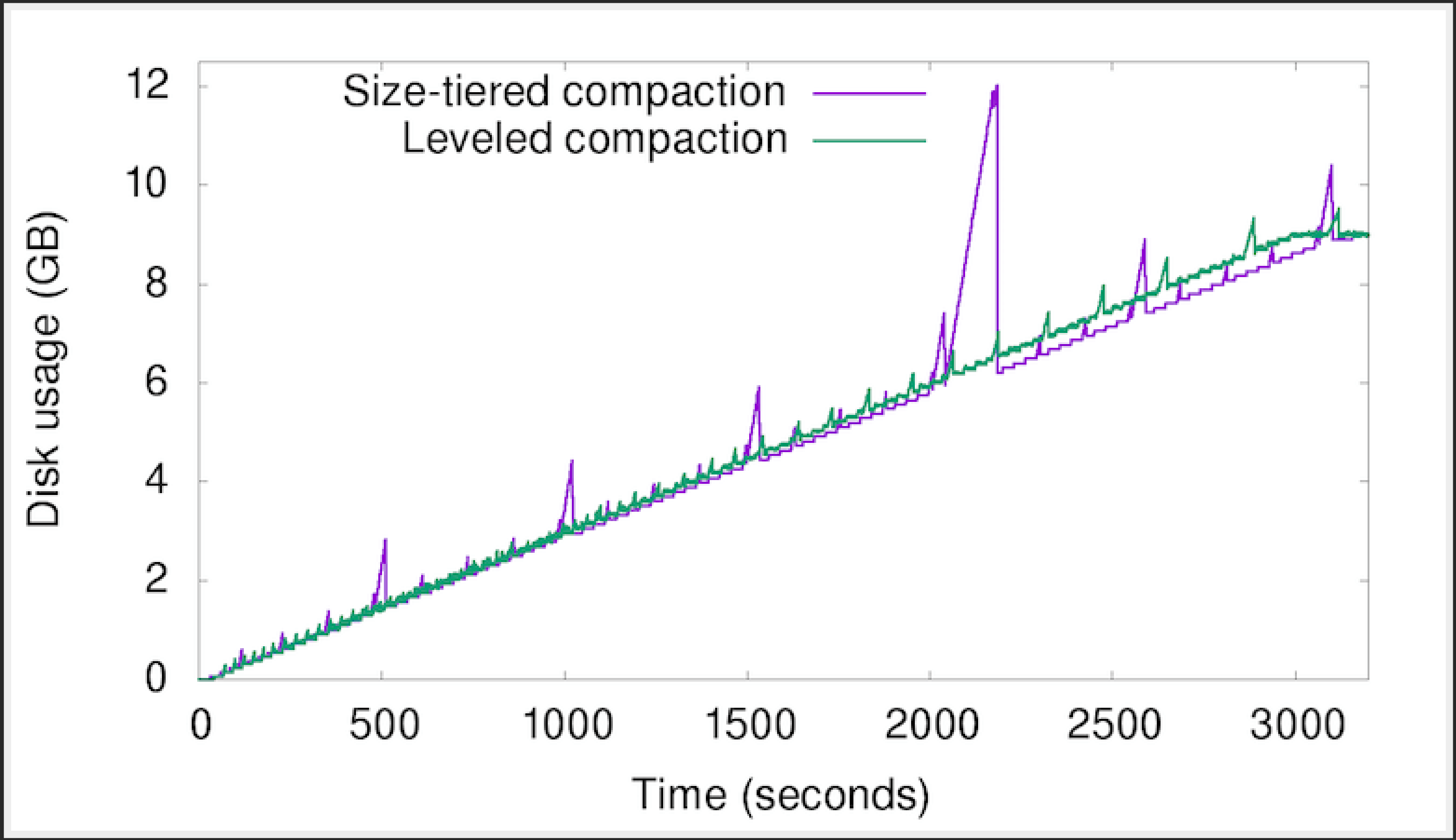
Fig. 4. LSM-tree design space: from a log to a sorted array.

The state of the art on Trade-off Curve



WEAKNESSES OF LSM-TREE

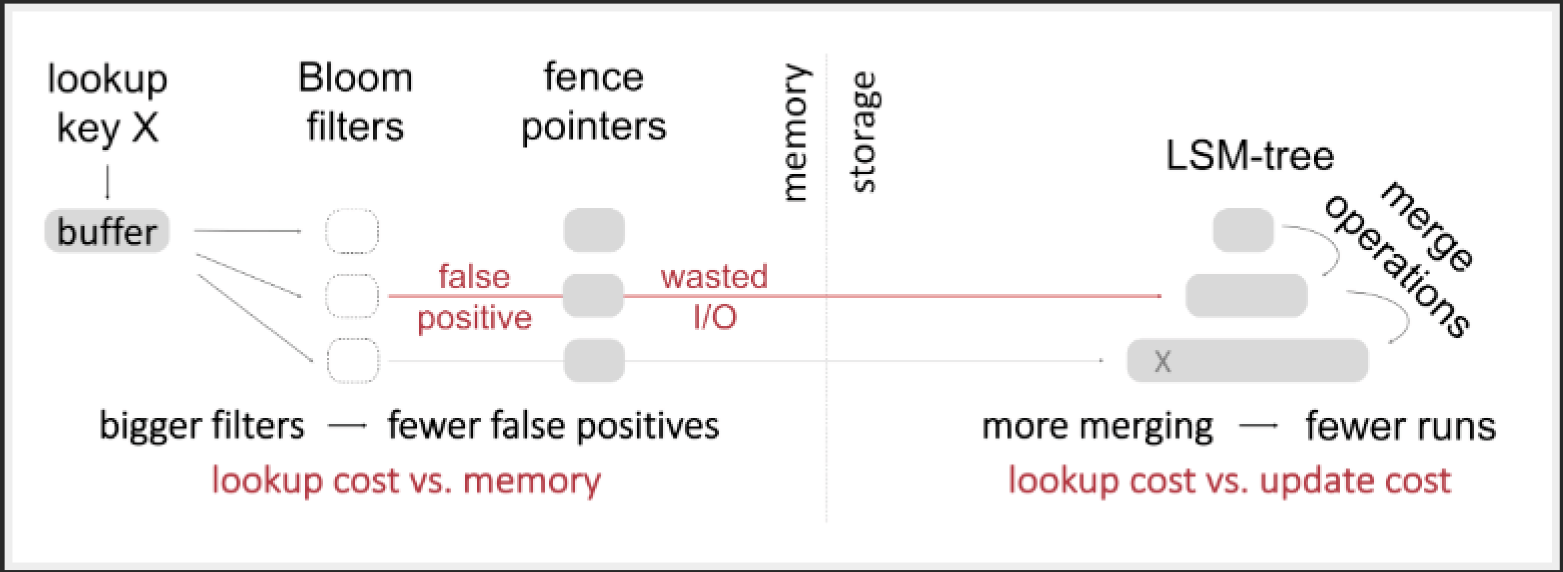
Space Amplification and Write Amplification



LSM TUNING

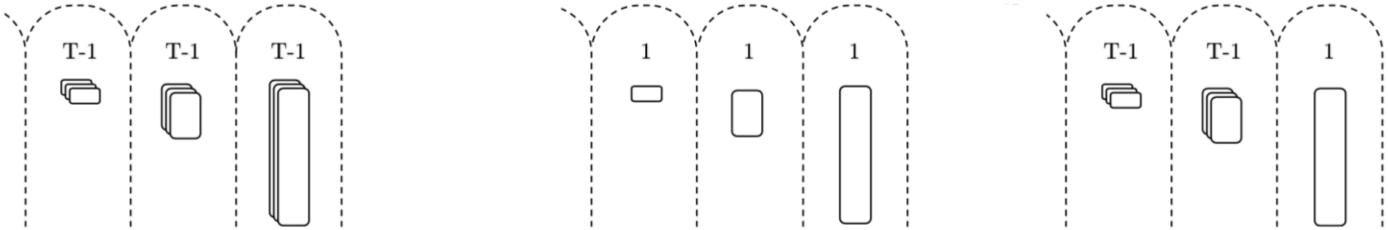
- Monkey by Harvard DASlab, 2017
- Dostoevsky by Harvard DASlab, 2018
- Universal Compaction by Facebook RocksDB, 2018
- ...

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

Lazy Leveling and Cost breakdown

Let us talk it in simple way

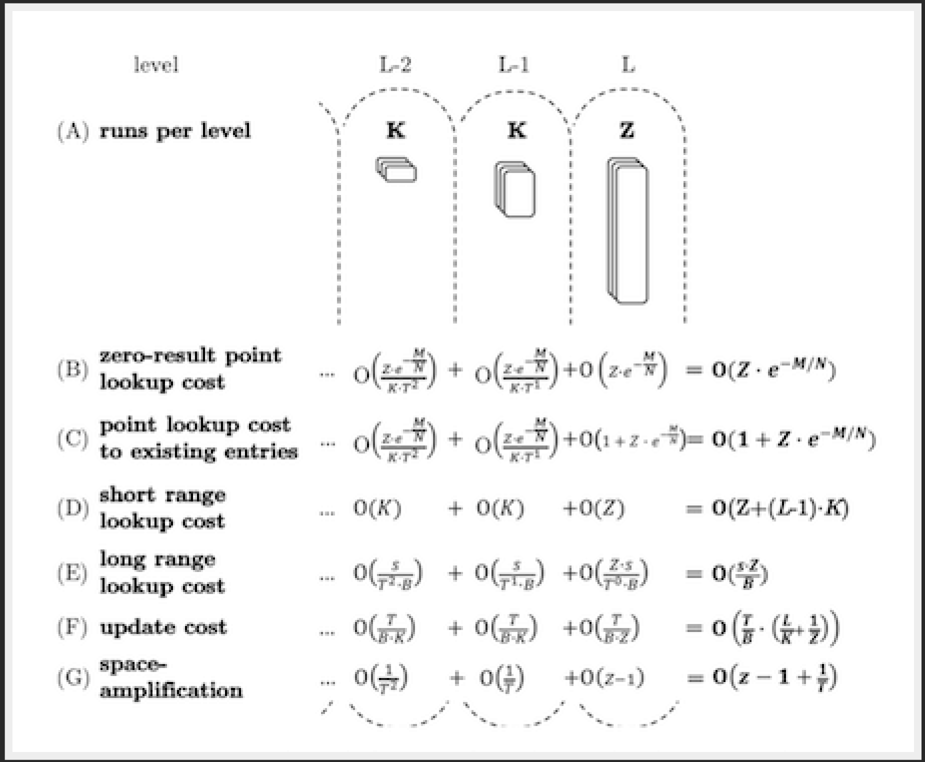
- Tiering is great for: updates
- Leveling is great for: short range lookup
- Lazy Leveling is great for: updates & point lookup

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



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 2O17
- 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