From WiscKey to Bourbon: A Learned Index for Log-Structured Merge Trees

University of Wisconsin – Madison
Microsoft Gray Systems Lab

Dai Y, Xu Y, Ganesan A, et al. From wisckey to bourbon: A learned index for log-structured merge trees[C]//Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. 2020: 155-171.

Data Lookup

Data Lookup is important in systems

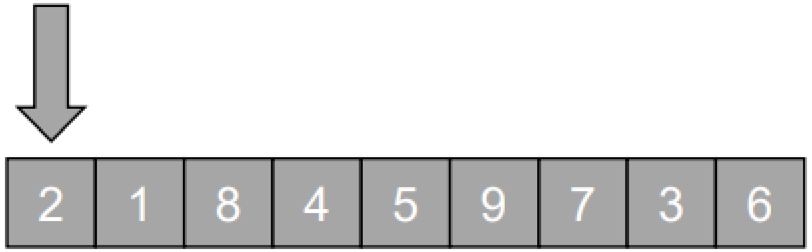
How do we perform a lookup given an array of data?

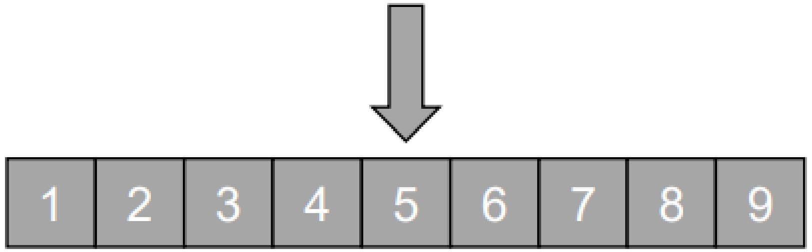
Linear search

What if the array is sorted?

Binary search

What if the data is huge?





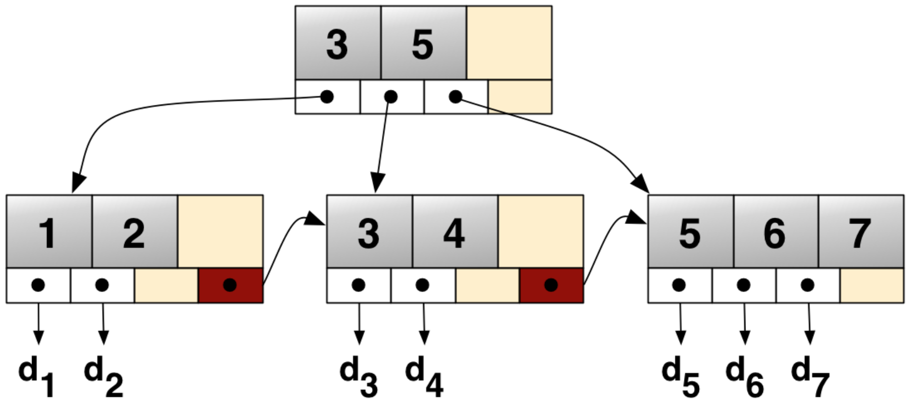
Data Structures to Facilitate Lookups

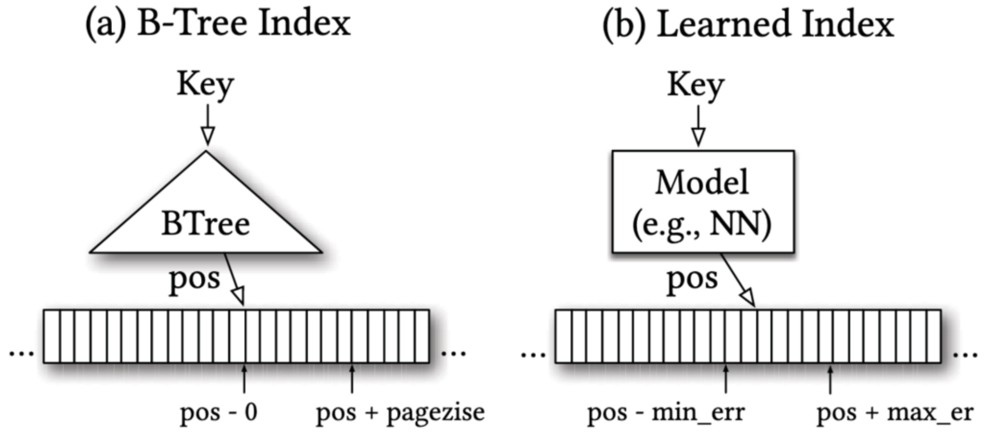
Assume sorted data

Traditional solution: build special data structures for lookups

- B-Tree, for example
- Record the position of the data

What if we know the data beforehand?





Bring Learning to Indexing

Lookups can be faster if we know the distribution

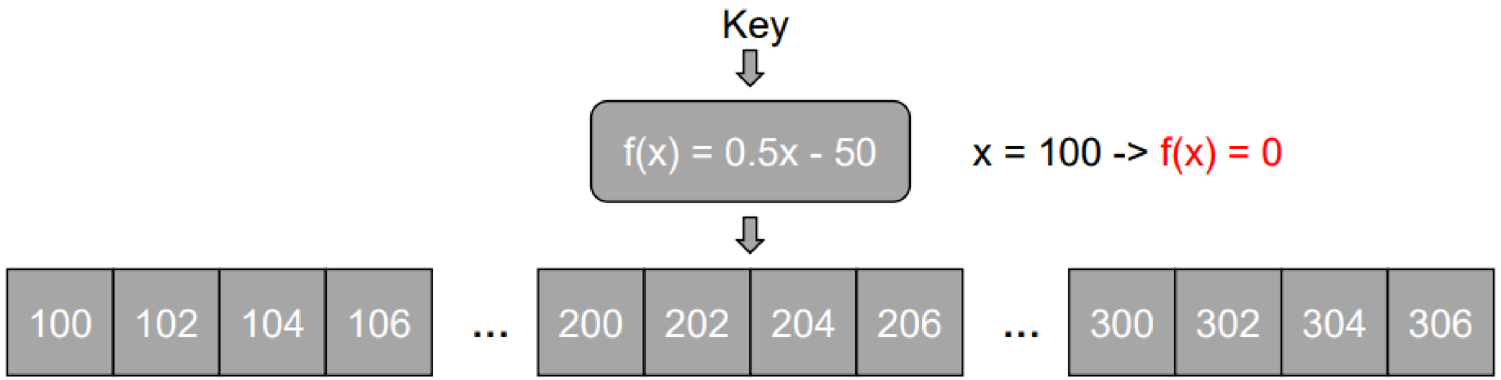
- The model $f(\bullet)$ learns the distribution

Learned Index

Time Complexity – $O(1)$ for lookups

Space Complexity – $O(1)$

- Only 2 floating points – slope + intercept



Kraska T, Beutel A, Chi E H, et al. The case for learned index structures[C]//Proceedings of the 2018 international conference on management of data. 2018: 489-504.

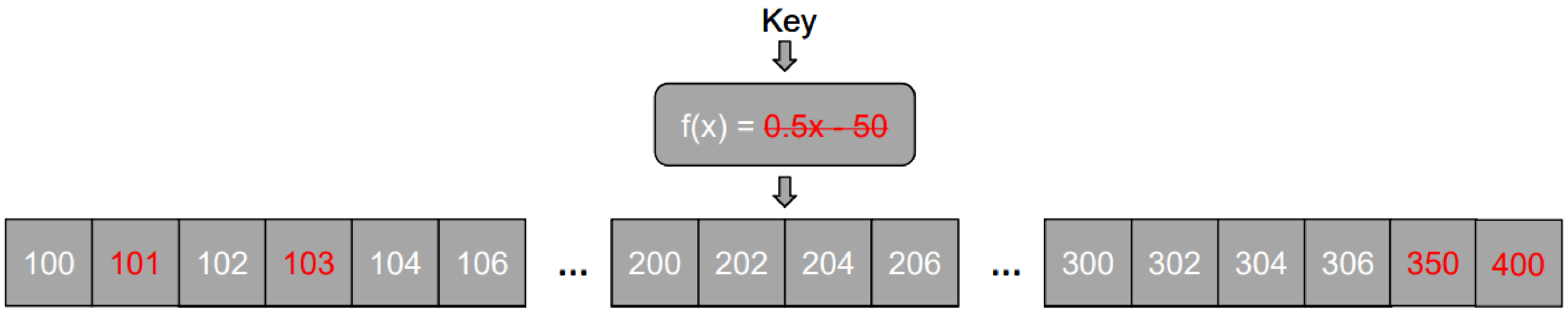
Challenges to Learned Indexes

How to efficiently support insertions/updates?

- Data distribution
- Need re-training, or lowered model accuracy

How to integrate into production systems?

- Key-Value Storage Systems

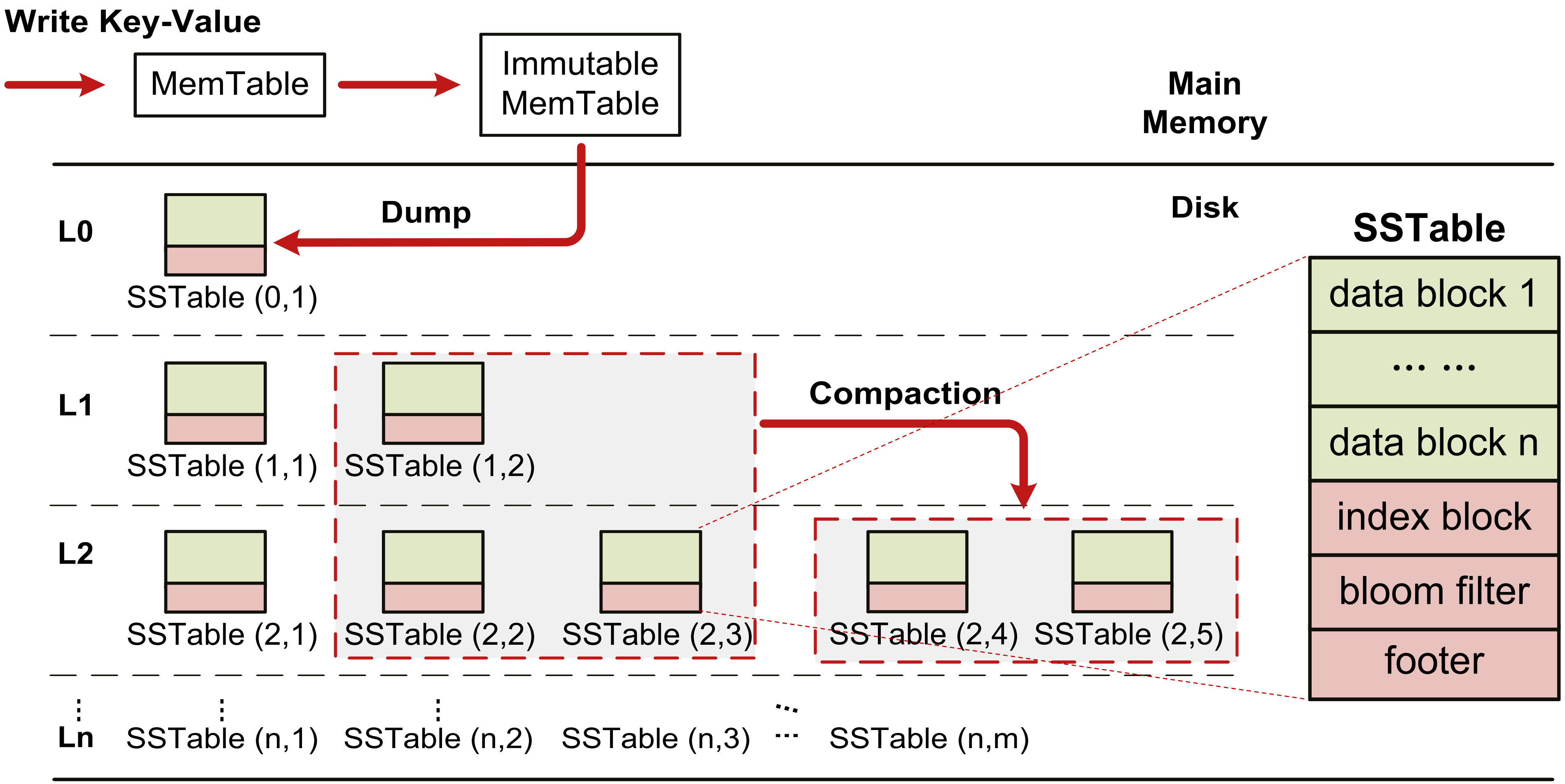


Key-Value Storage Systems

Key-Value stores are widely used in various applications

- Flexibility
- Scalability
- Fast write throughput

LSM-tree (Log-Structured Merge-Tree)



LevelDB

Key-value store based on LSM

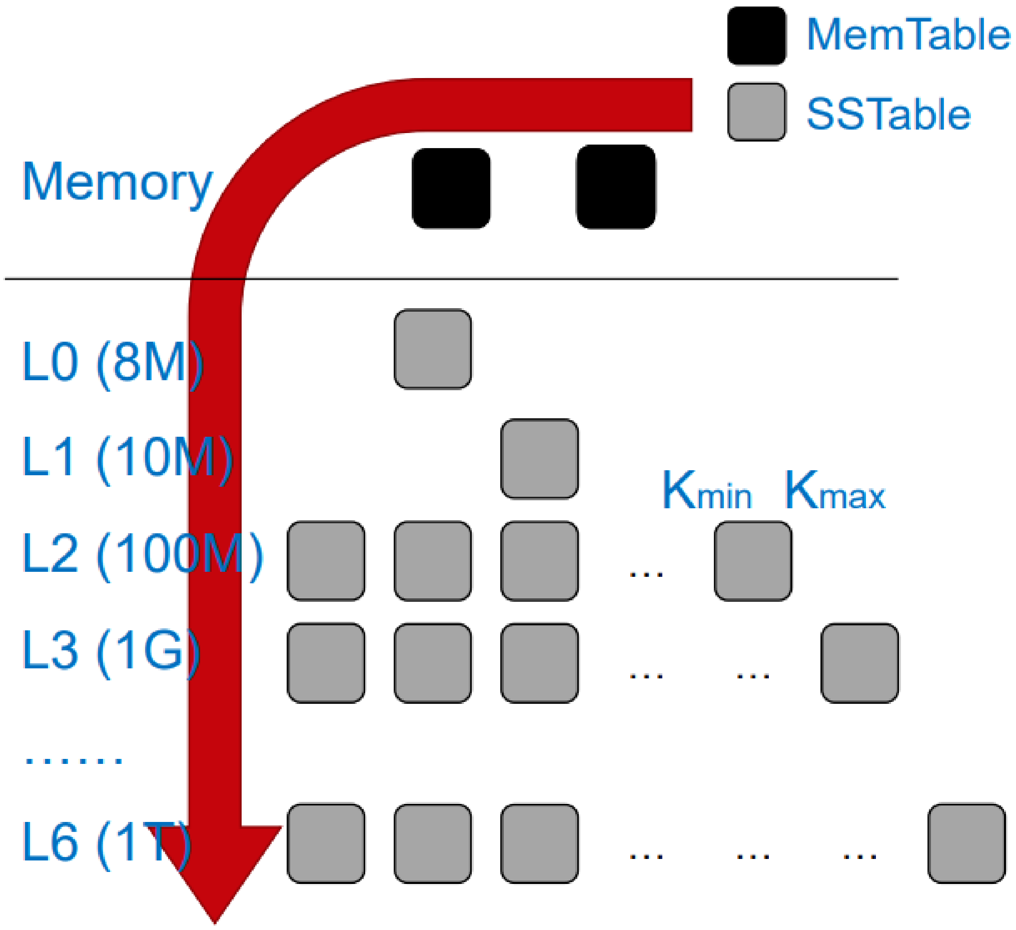
- 2 in-memory tables
- 7 levels of on-disk SSTables (files)

Update/Insertion procedure

- Buffered in MemTables
- Merging compaction
- From upper to lower levels
- No in-place updates to SSTables

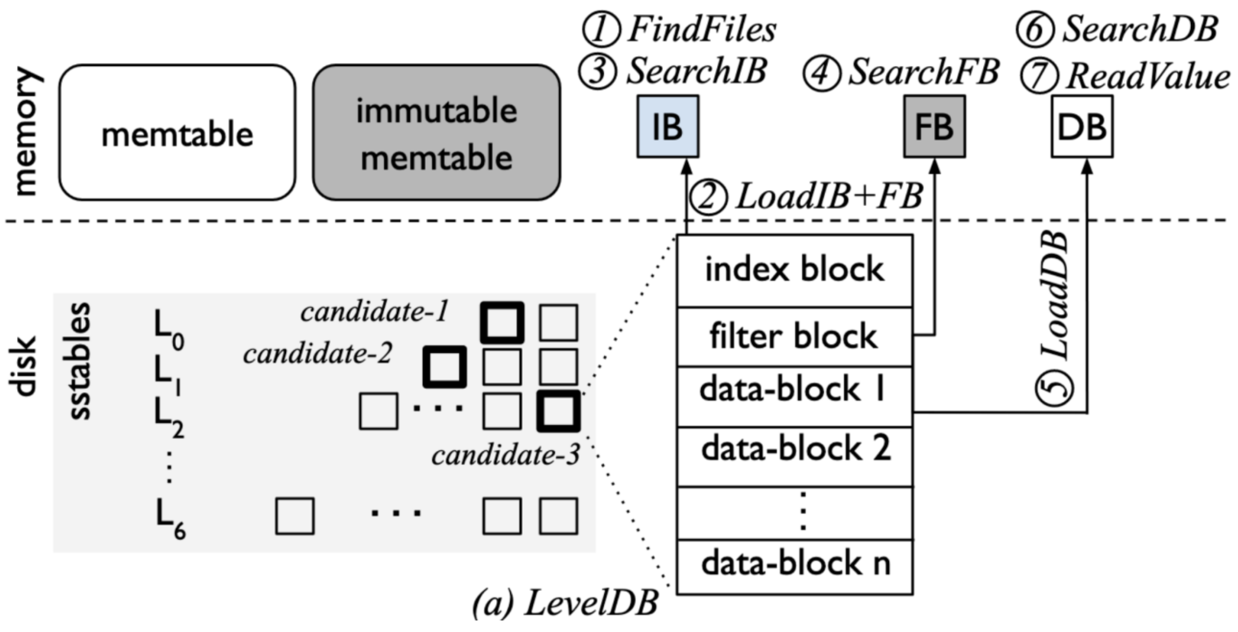
Lookup procedure

- From upper to lower levels
- Positive/Negative internal lookups

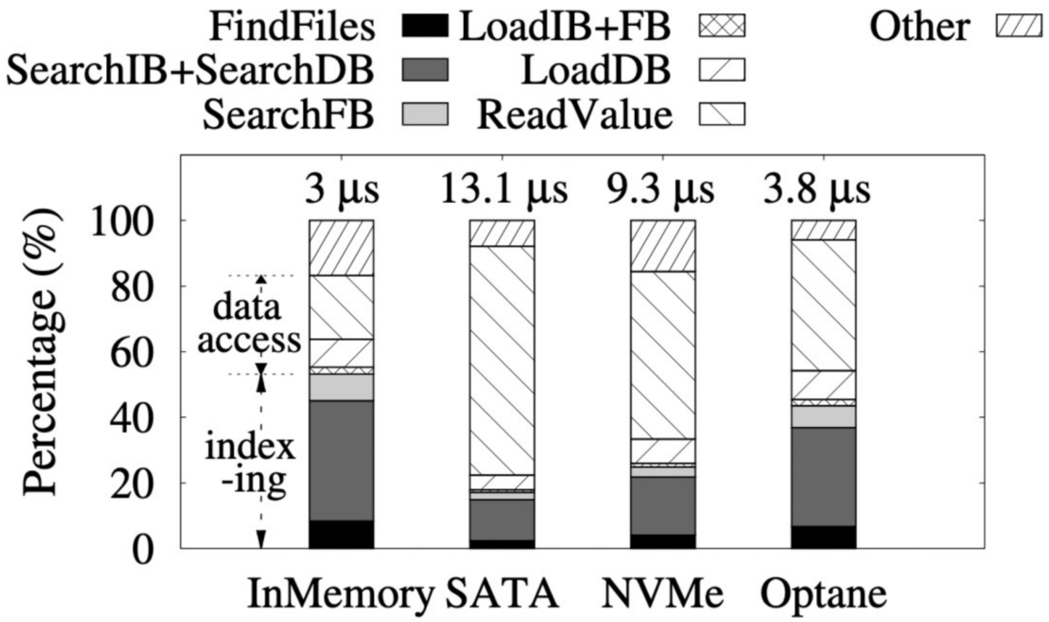


Motivation

Take an insight into the latency of each operation of LSM-tree



LevelDB



Lookup Latency Breakdown

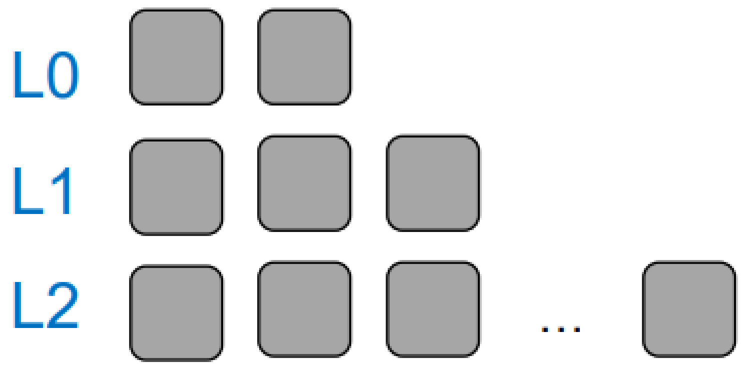
Learning Guidelines

Learning at SSTable granularity

- No need to update models
- Models keep a fixed accuracy

Factors to consider before learning:

1. Lifetime of SSTables
   - How long a model can be useful
2. Number of lookups into SSTables
   - How often a model can be useful



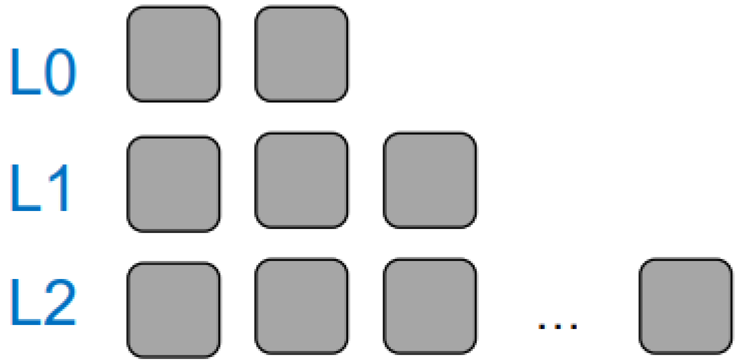
Learning Guidelines

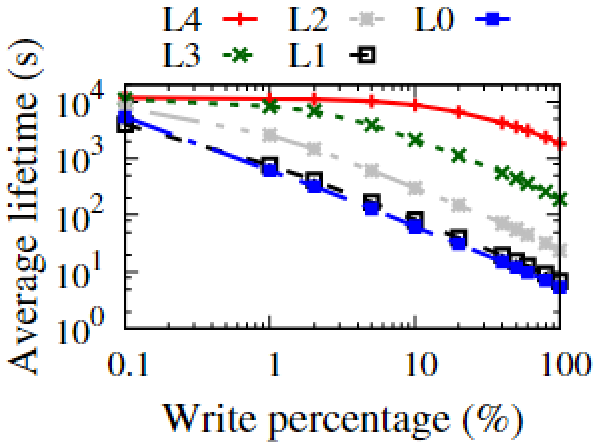
1. Lifetime of SSTables

    How long a model can be useful

Experimental results

- Under 15Kops/s and 50% writes
- Average lifetime of L0 tables: 10 seconds
- Average lifetime of L4 tables: 1 hour
- A few very short-lived tables: < 1 second





Learning guideline 1: Favor lower level tables

    Lower level files live longer

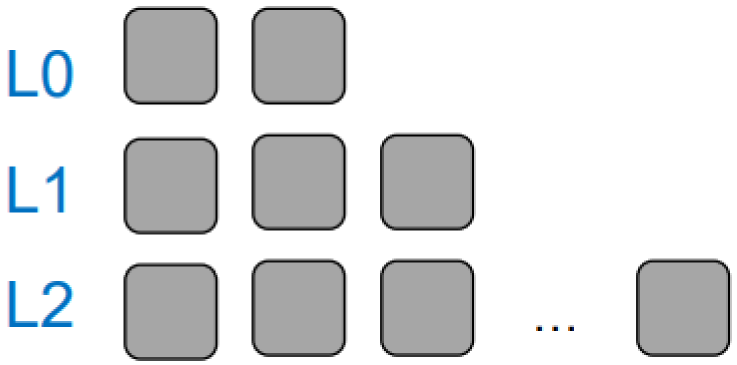
Learning guideline 2: Wait shortly before learning

    Avoid learning extremely short-lived tables

Learning Guidelines

2. Number of lookups into SSTables

  How often a model can be useful



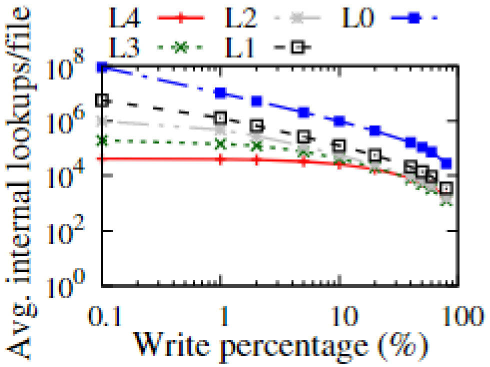
Affected by various factors

  Depending on workload distribution, load order, etc.

  Higher level files may serve more internal lookups

Learning guideline 3: Do not neglect higher level tables

  Models for them may be more often used



Learning guideline 4: Be workload- and data-aware

  Number of internal lookups affected by various factors

Learning Algorithm: Greedy-PLR

Greedy Piecewise Linear Regression

From Dataset D

Multiple linear segments $f(\bullet)$

$\forall (x, y) \in D, |f(x) - y| < error$

$error$ is specified beforehand

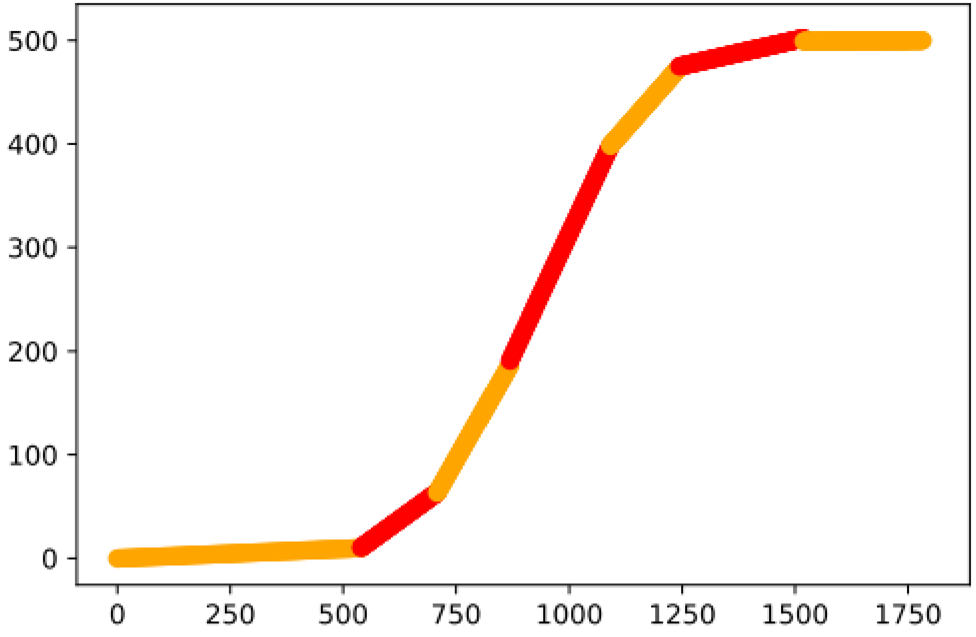
Bourbon set $error = 8$

Train complexity: $O(n)$

Typically ~40ms

Inference complexity: $O(\log \#seg)$

Typically <1μs



Xie Q, Pang C, Zhou X, et al. Maximum error-bounded piecewise linear representation for online stream approximation[J]. The VLDB journal, 2014, 23: 915-937.

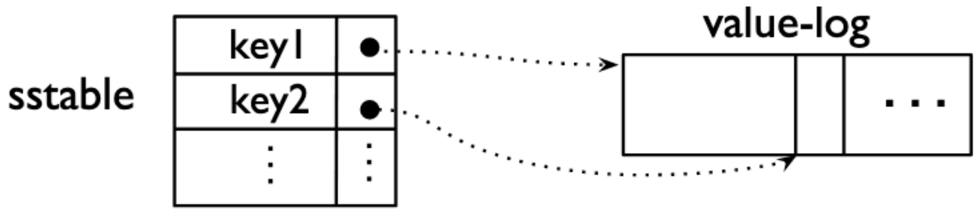
Bourbon Implementation

Bourbon: build upon WiscKey

- WiscKey: key-value separation built upon LevelDB
- (key, value_addr) pair instead of (key, value) in LSM-tree
- A separate value log

Why WiscKey?

- Help handle large and variable sized values
- Constant-sized KV pairs in the LSM-tree
- Prediction much easier



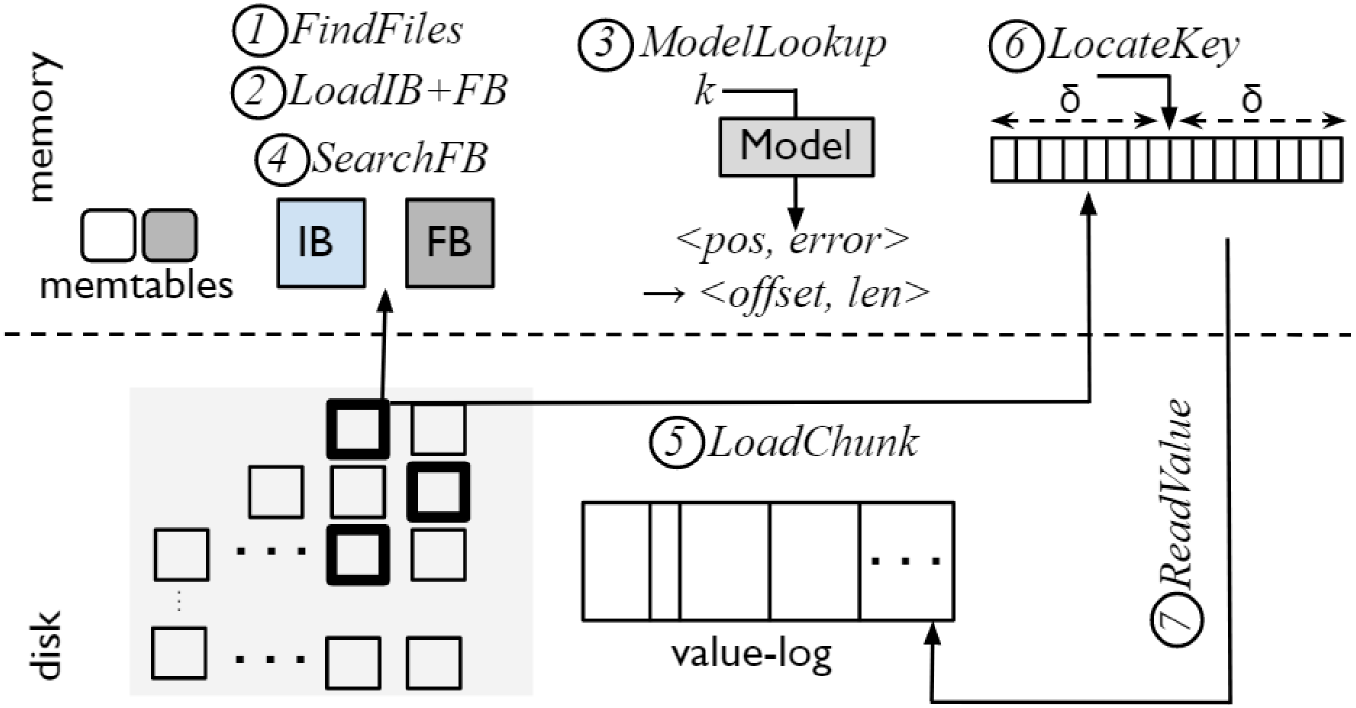
(b) WiscKey

Lu L, Pillai T S, Gopalakrishnan H, et al. Wisckey: Separating keys from values in ssd-conscious storage[J]. ACM Transactions on Storage, 2017, 13(1): 1-28.

Bourbon Lookup Path

Modify the lookup procedure of WiscKey

- Model exists
- No model (baseline)

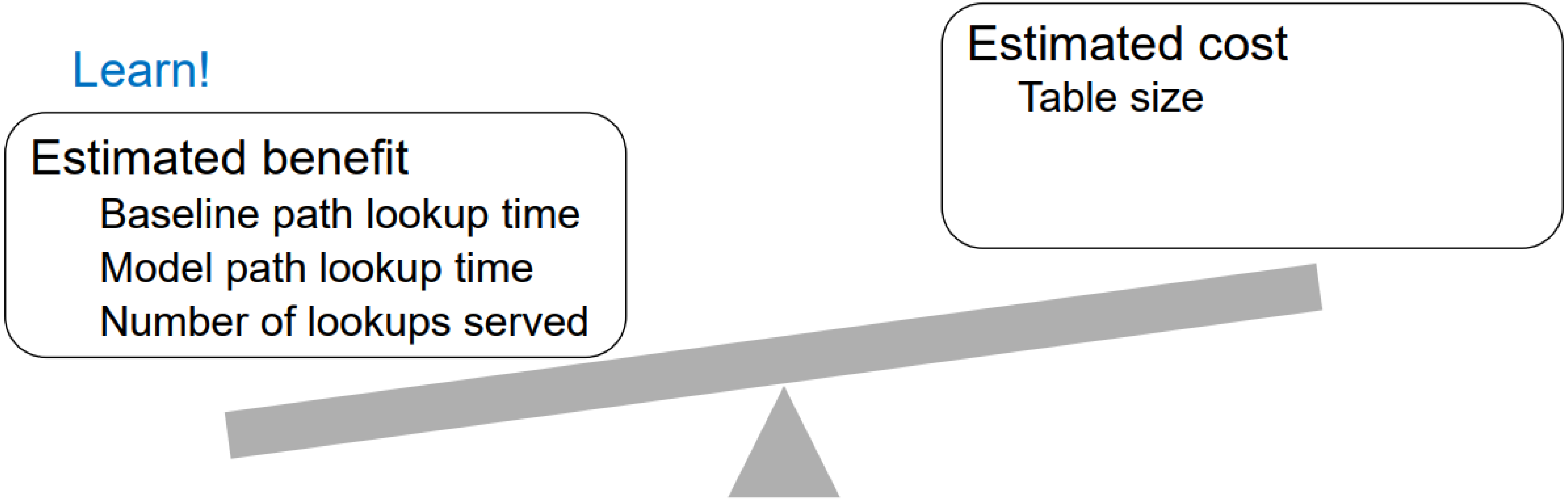


(b) Lookup via model - detailed steps

Cost-Benefit Analyzer

Goal: Minimize total CPU time

A balance between always-learn and no-learn



Evaluation

1. Environment

   20-core Intel Xeon CPU E5-2660, 160-GB memory, 480-GB SATA SSD

2. Trace

   4 synthetic traces (64M) and 2 real-world traces(33M/22M)

3. Workload

   Read-only/heavy, range-heavy, write-heavy

   10M operations

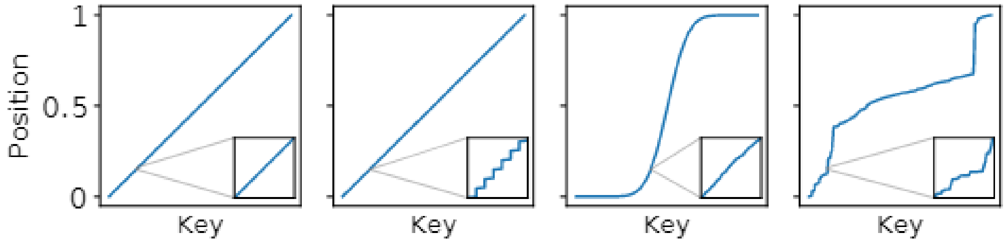
4. Parameter

   16B-sized integer keys, 64B-sized values

   Error bound = 8

5. Baseline

   WiscKey



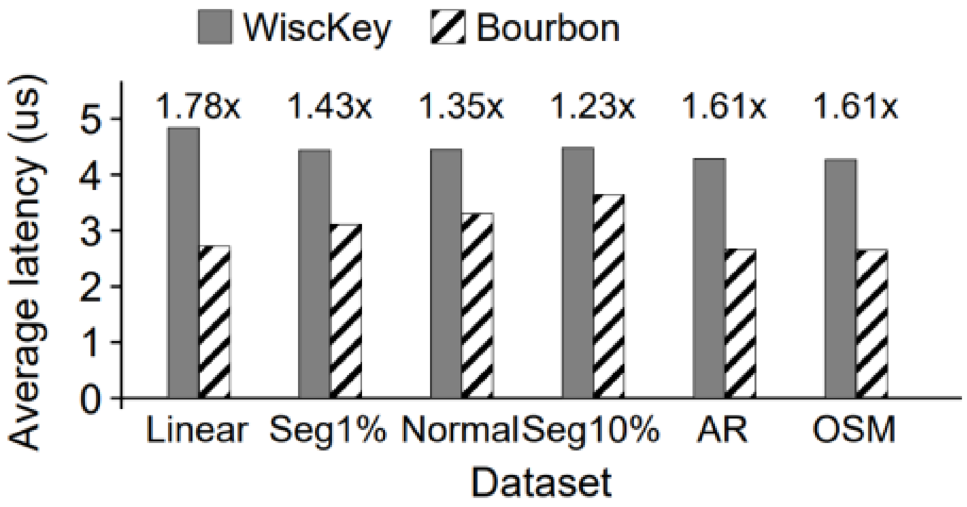
(a) Linear (b) Seg10% (c) Normal (d) OSM

Can Bourbon adapt to different datasets?

Micro benchmark: datasets

4 synthetic datasets: linear, normal, seg1%, and seg10%
2 real-world datasets: AmazonReviews (AR) and OpenStreetMap (OSM)
Uniform random read-only workloads

Dataset	#Data	#Seg	%Seg
Linear	64M	900	0%
Seg1%	64M	640K	1%
Normal	64M	705K	1.1%
Seg10%	64M	6.4M	10%
AR	33M	129K	0.39%
OSM	22M	295K	1.3%



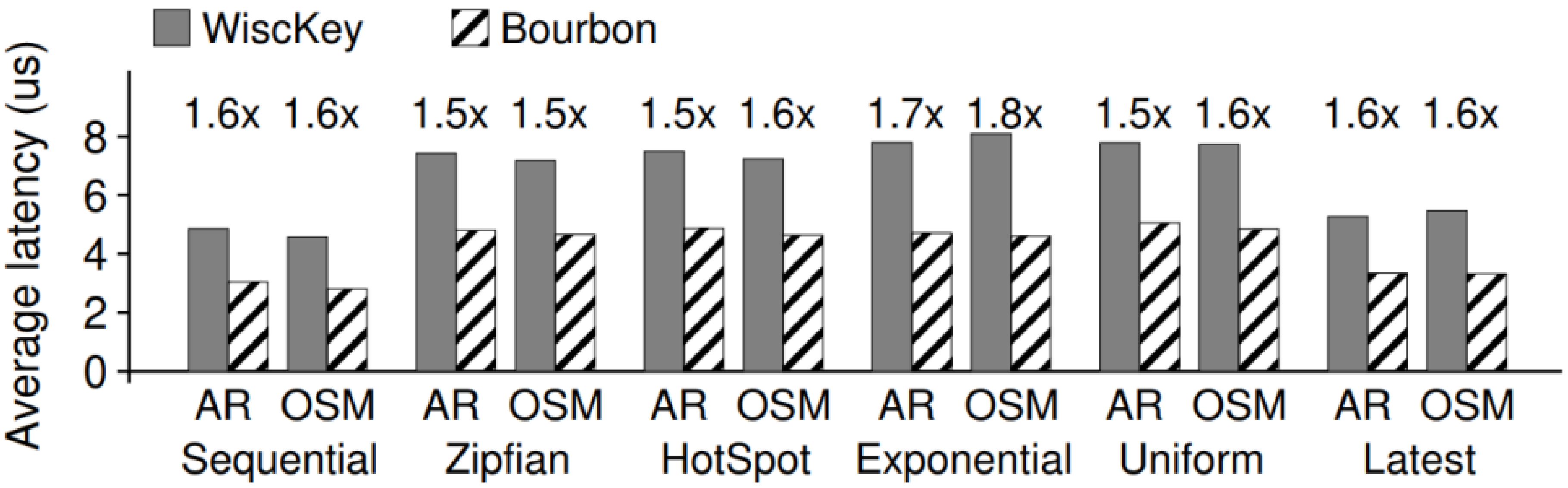
Bourbon performs better with lower number of segments

Reach 1.6× gain for two real-world datasets with 1% segments

Performance with different request distributions?

Micro benchmark: request distribution

- Read-only workloads
- Sequential, zipfian, hotspot, exponential, uniform, and latest



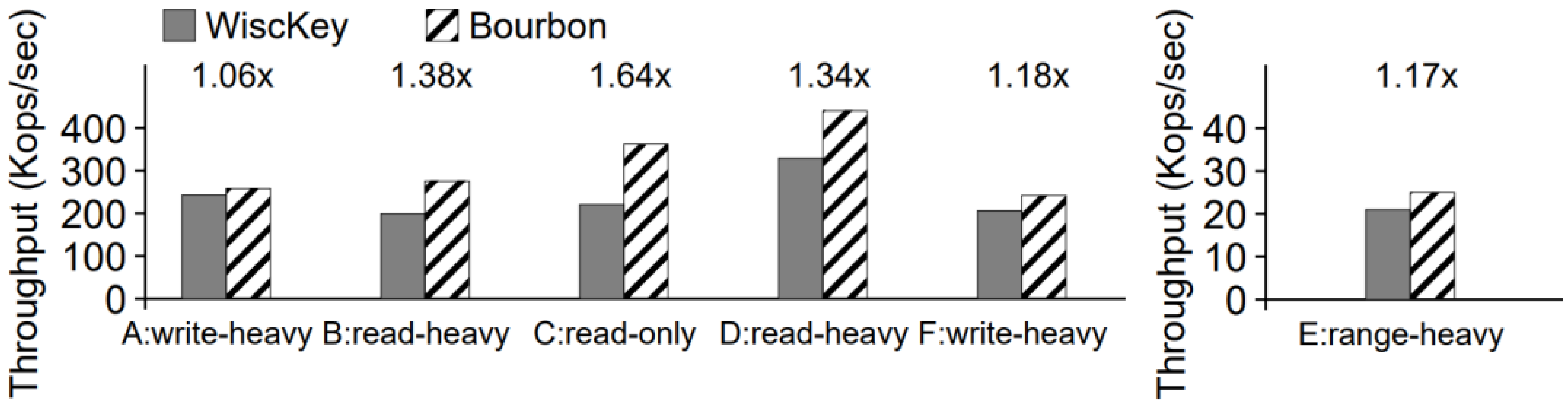
Bourbon improves performance by ~1.6×

- Regardless of request distributions

Can Bourbon perform well on real benchmarks?

Micro benchmark: YCSB

- 6 core workloads on YCSB default dataset
- Bourbon improves reads without affecting writes



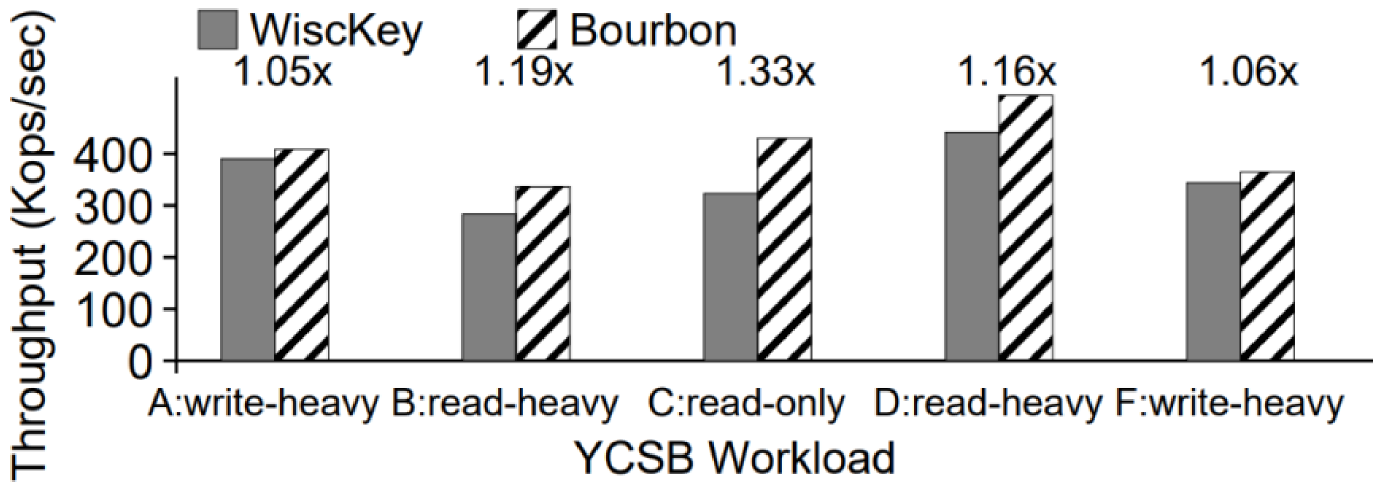
Bourbon's gain holds on real benchmarks

Bourbon improves reads without affecting writes

Is Bourbon beneficial when data is on storage?

Performance on fast storage

- Data resides on an Intel Optane SSD
- 5 YCSB core workloads on YCSB default dataset



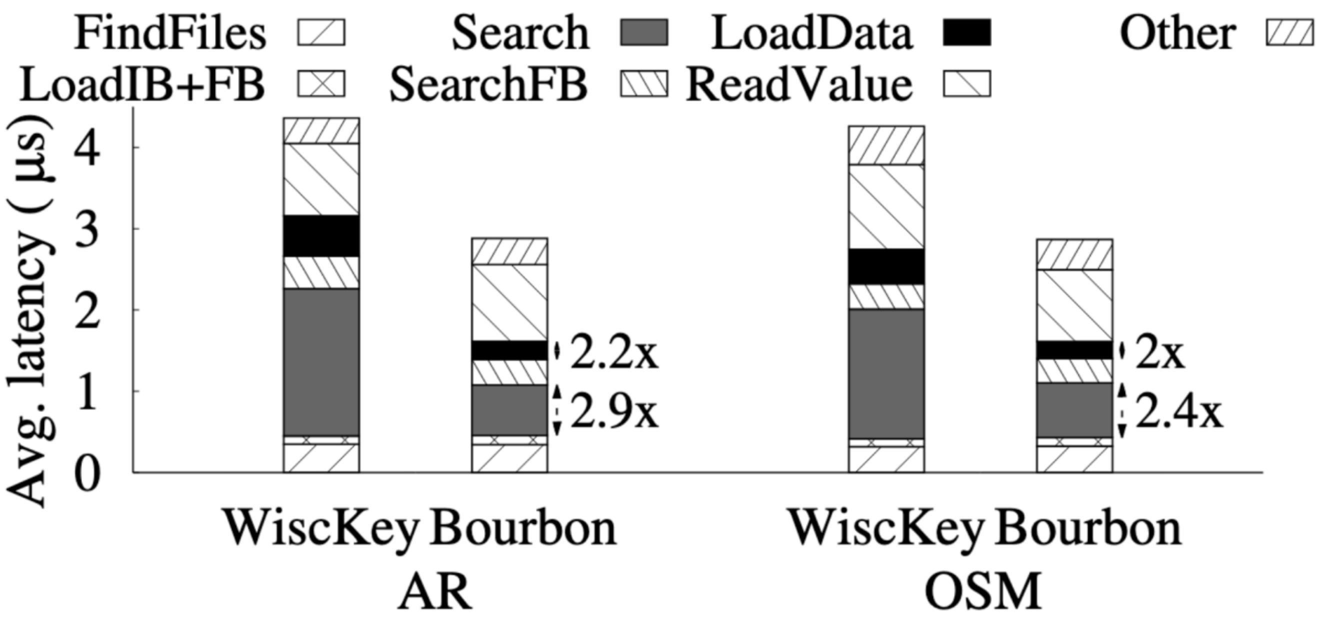
Bourbon can still offer benefits when data is on storage

- Will be better with emerging storage technologies

Which Portions does Bourbon optimize?

Lookup latency of WiscKey and Bourbon on AR and OSM

- Bourbon reduce the indexing portions by up to 2×
- Interestingly, Bourbon reduce the data-access costs too, by up to 2×



File vs. Level Learning

Lookup latency of WiscKey and Bourbon using different granularity

- Write-heavy/read-heavy: file model beats level model
- Read-only: level model beats file model but gains a little benefit

Workload	Baseline time (s)	File model		Level model	
		Time(s)	% model	Time(s)	% model
Mixed: Write-heavy	82.6	71.5 (1.16 ×)	74.2	95.1 (0.87 ×)	1.5
Mixed: Read-heavy	89.2	62.05 (1.44 ×)	99.8	74.3 (1.2 ×)	21.4
Read-only	48.4	27.2 (1.78 ×)	100	25.2 (1.92 ×)	100

Effectiveness of Cost-Benefit Analyzer

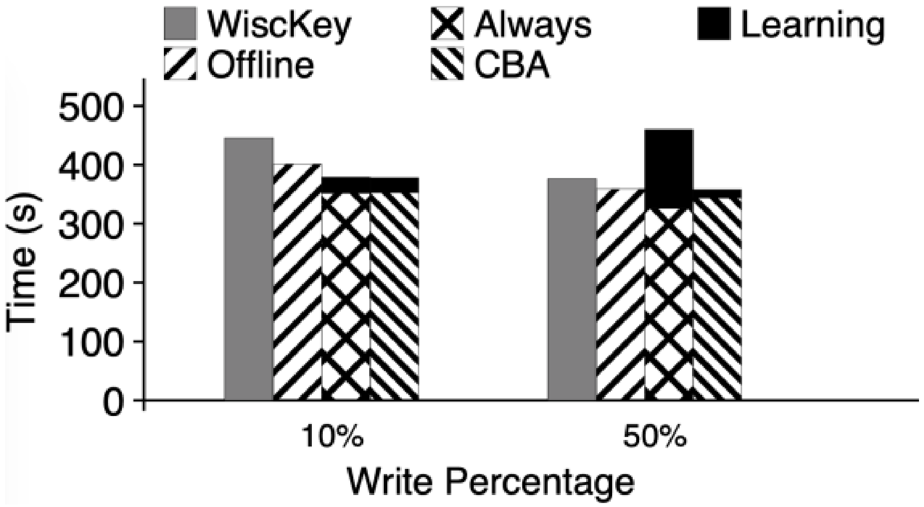
Learn most/all new tables at low write percentages

- Reach a better foreground latency than offline learning

Limit learning at a high write percentages

- Reduce learning time and have a good foreground latency

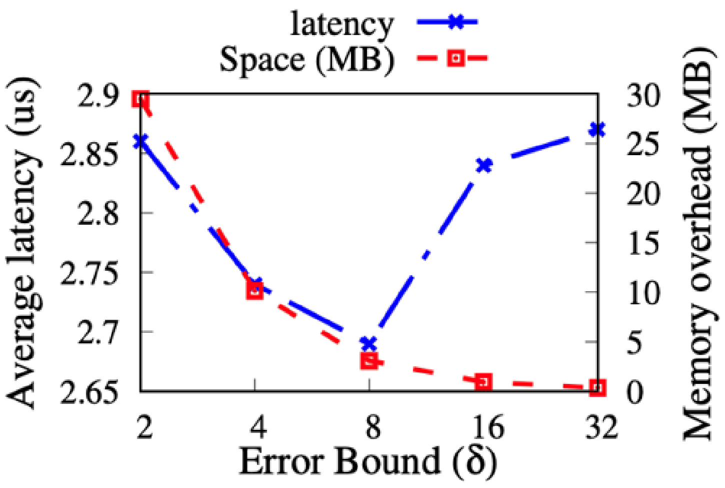
Minimal total CPU cost in all scenarios



Error-bound Trade-off and Overhead

Error bound affects both the lookup performance and space overhead

- Important hyperparameter in Bourbon
- Future research: auto tuning?



(a) Error-bound tradeoff

Dataset	Space Overheads	
	MB	%
Linear	0.02	0.0
Seg1%	15.38	0.21
Seg10%	153.6	2.05
Normal	16.94	0.23
AR	3.09	0.08
OSM	7.08	0.26

(b) Space overheads

Conclusion

Bourbon

- Integrates learned indexes into a production LSM system
- Beneficial on various workloads
- Learning guidelines on how and when to learn
- Cost-Benefit Analyzer on whether a learning a worthwhile

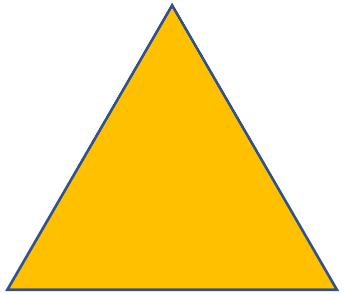
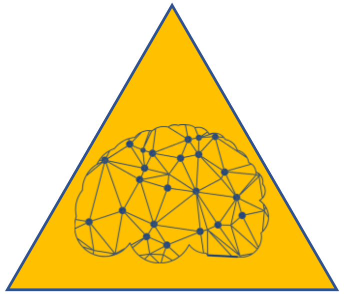
How will ML change computer system mechanisms?

- Not just policies
- Bourbon improves the lookup process with learned indexes
- What other mechanisms can ML replace or improve?
- Careful study and deep understanding are required


Q & A

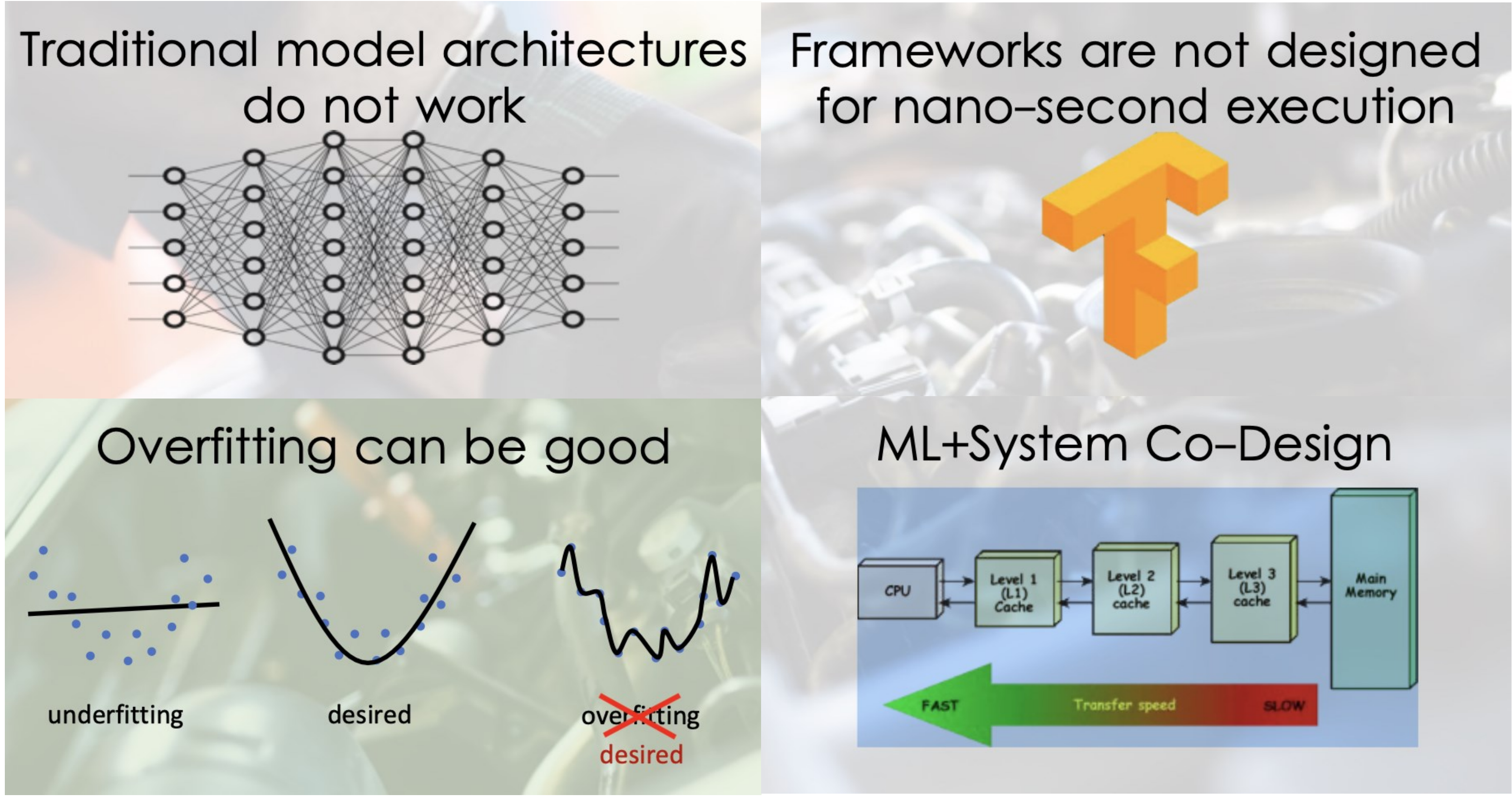
Appendix A

How About Using Neural Network to Learn?

Tensorflow	State-Of-The-Art B-Tree	Learned Index
		
>80,000ns	260ns	85ns
	13MB	0.7MB

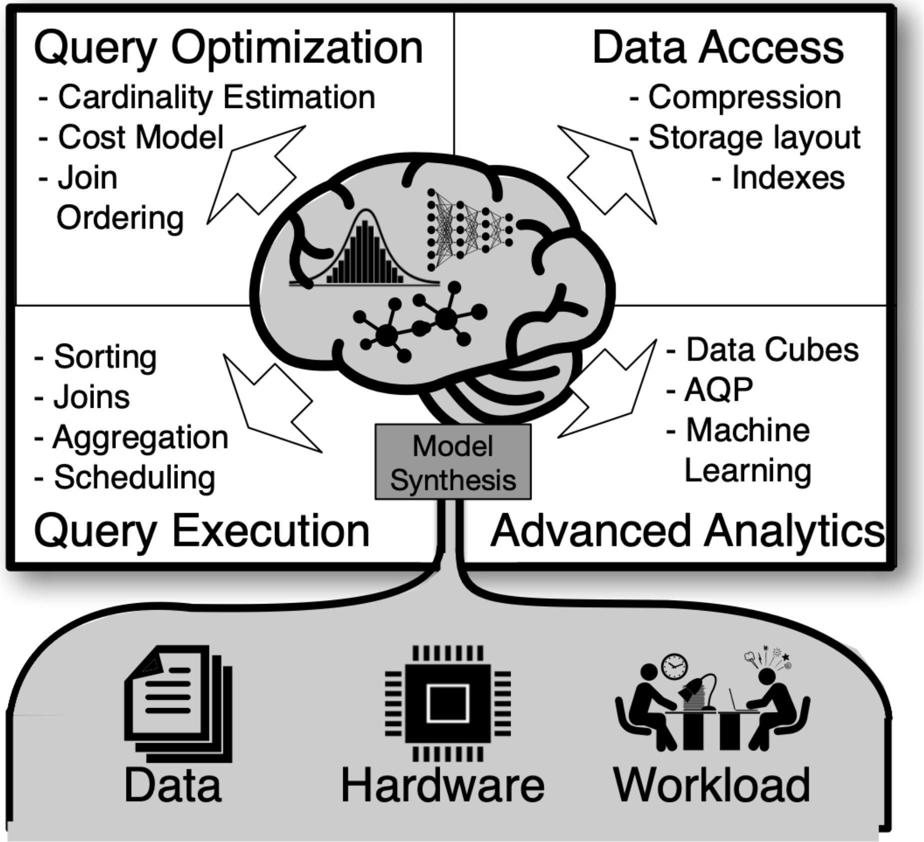
Appendix B

Current Challenges of Learned Index



Appendix C

A Work-in-Progress Learned Database Proposed by MIT



MIT CSAIL DSAIL Data Systems and AI Lab Microsoft Google intel

Appendix D

Other Cases Benefit From Learned Index