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

Yifan Dai, Yien Xu, Aishwarya Ganesan, Ramnatthan Alagappan,
Brian Kroth, Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau

Data Lookup

Data lookup is important in systems

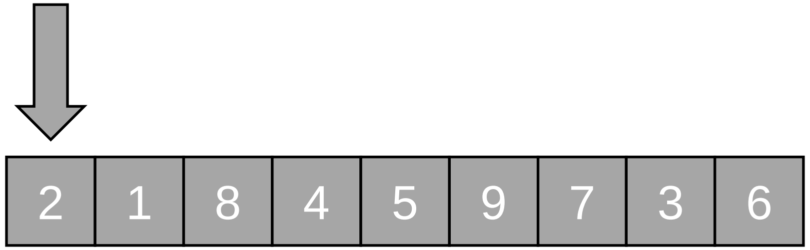
How do we perform a lookup given an array of data?

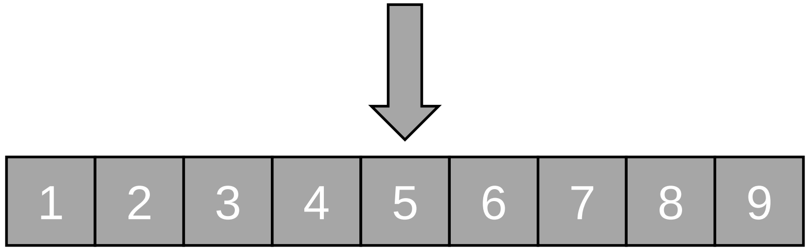
- Linear search

What if the array is sorted?

- Binary search

What if the data is huge?





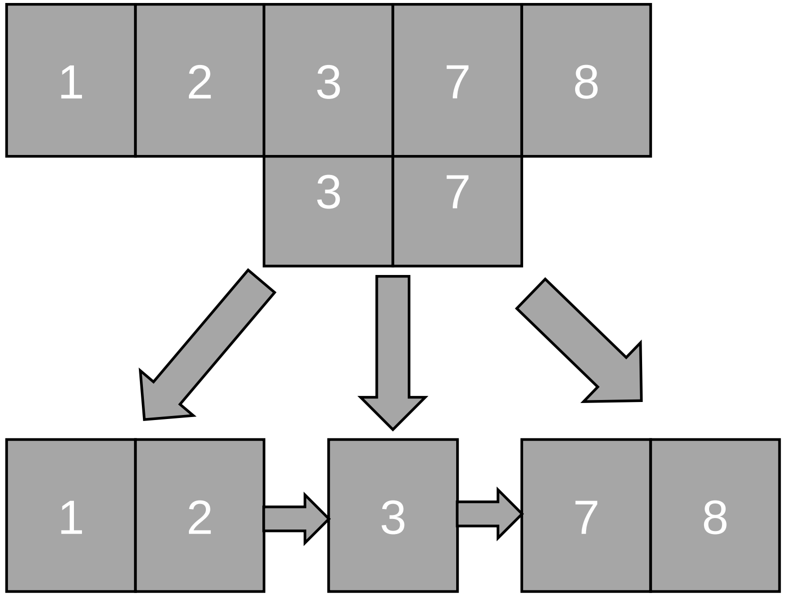
Data Structures to Facilitate Lookups

Assume sorted data

Traditional solution: build specific 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

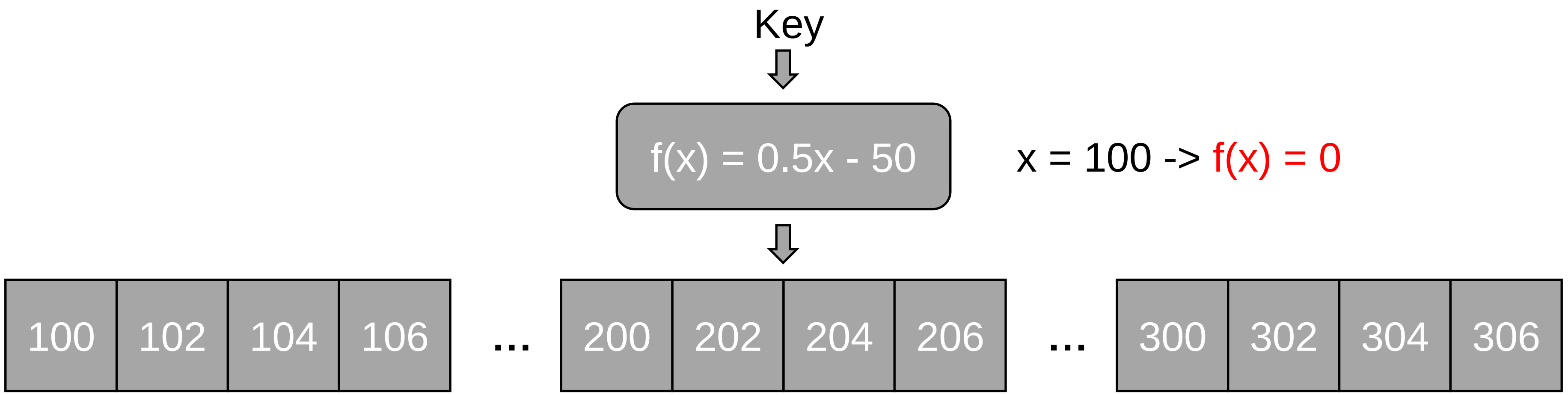
- The model f(•) learns the distribution

Leaned Indexes

Time Complexity – O(1) for lookups

Space Complexity – O(1)

- Only 2 floating points – slope + intercept



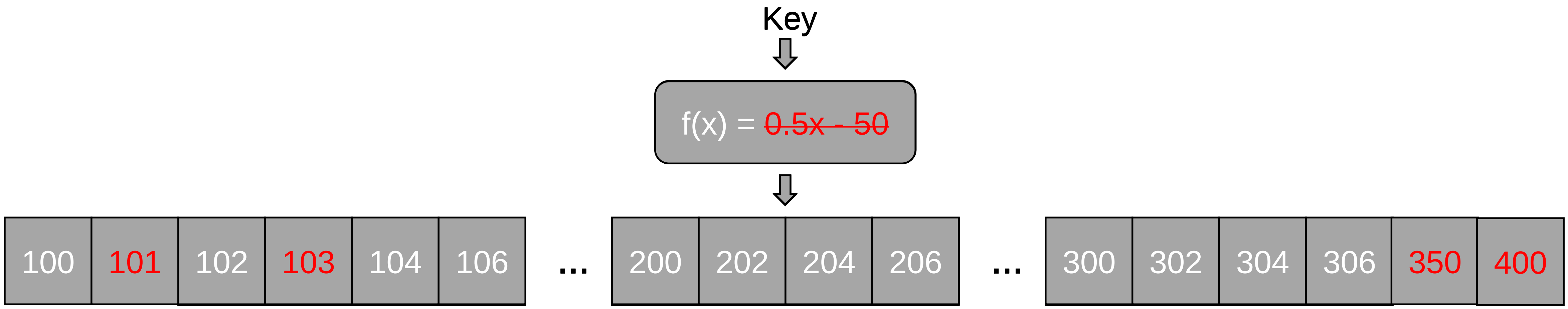
Kraska et al. The Case for Learned Index Structures. 2018

Challenges to Learned Indexes

How to efficiently support insertions/updates?

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

How to integrate into production systems?



Bourbon

- Bourbon
  - A Learned index for LSM-trees
  - Built into production system (WiscKey)
  - Handle writes easily
- LSM-tree fits learned indexes well
  - Immutable SSTables with no in-place updates
- Learning guidelines
  - How and when to learn the SSTables
- Cost-Benefit Analyzer
  - Predict if a learning is beneficial during runtime
- Performance improvement
  - 1.23x – 1.78x for read-only and read-heavy workloads
  - ~1.1x for write-heavy workloads

LevelDB

Key-value store based on LSM

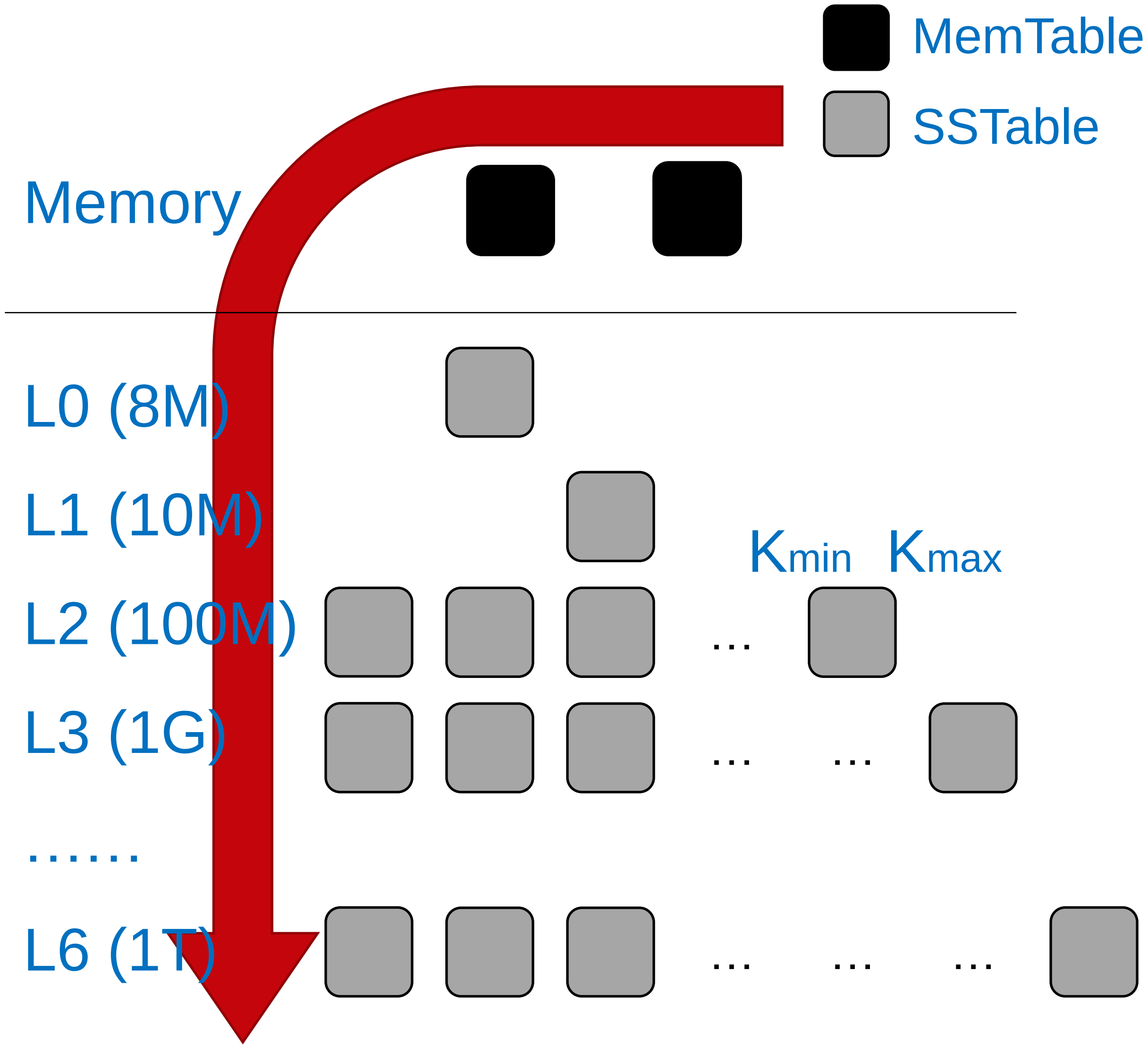
- 2 in-memory tables
- 7 levels of on-disk SSTables (files)

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



Learning Guidelines

Learning at SSTable granularity

No need to update models

Models keep a fixed accuracy

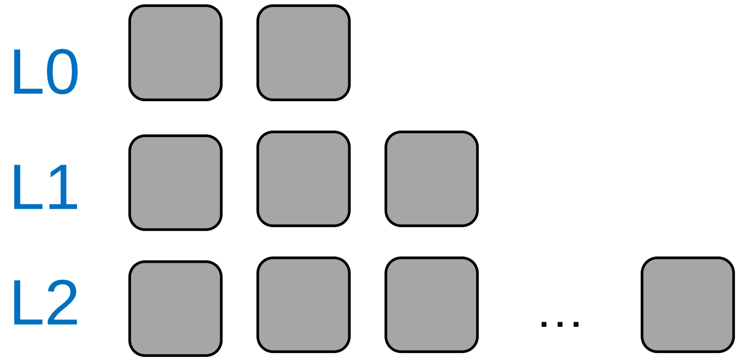
Factors to consider before learning:

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

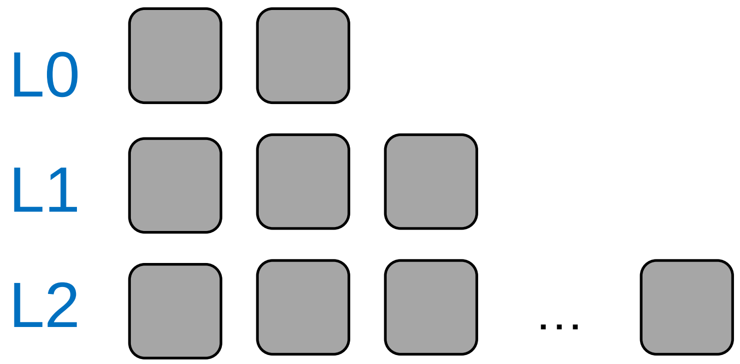
Experimental results

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



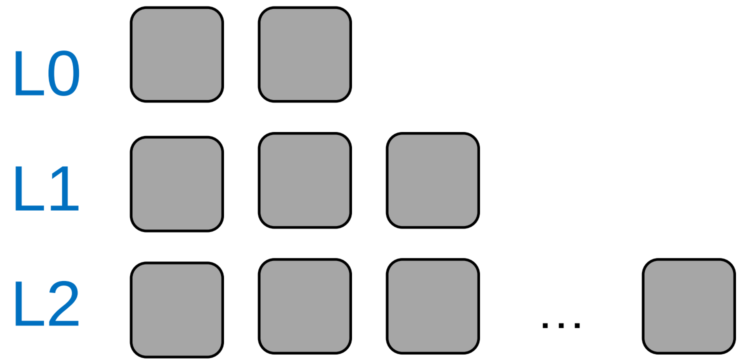
Learning guideline 1: Favor lower level tables

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

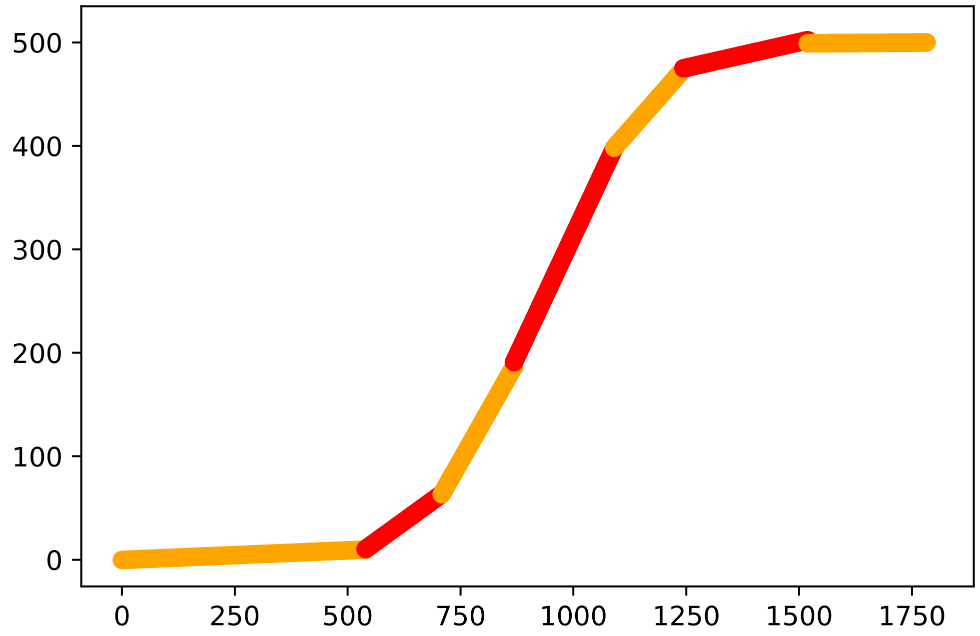
Learning guideline 3: Do not neglect higher level tables
   - Models for them may be more often used

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 $f(\cdot)$
- $\forall(x,y) \in D, |f(x) - y| < error$
- $error$ is specified beforehand
- In bourbon, we set $error = 8$



Train complexity: O(n)

- Typically ~40ms

Inference complexity: O(log #seg)

- Typically <1μs

Xie et al. Maximum error-bounded piecewise linear representation for online stream approximation. 2014

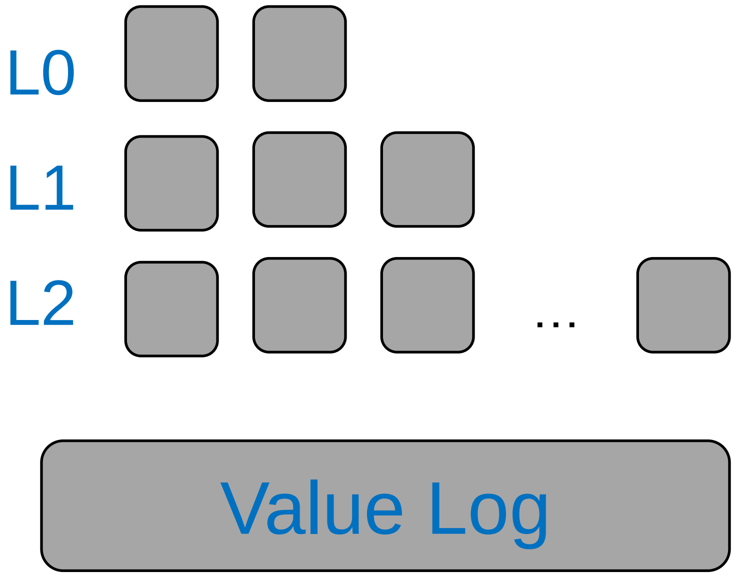
Bourbon Design

Bourbon: Build upon WiscKey

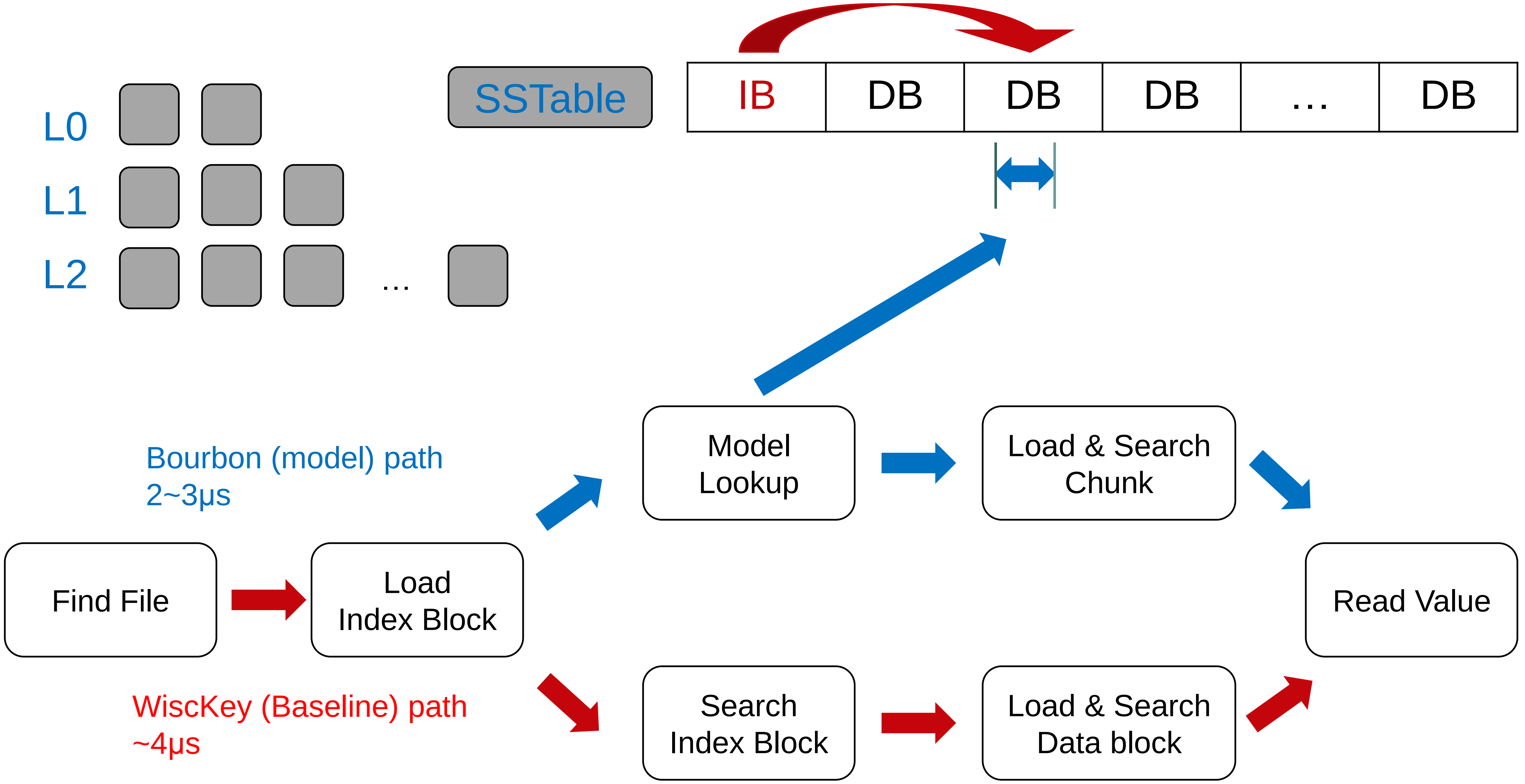
- WiscKey: key-value separation built upon LevelDB
- (Key, value_addr) pair in the LSM-tree
- A separate value log

Why WiscKey?

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



Bourbon Design



Cost-Benefit Analyzer

Goal: Minimize total CPU time

- A balance between always-learn and no-learn

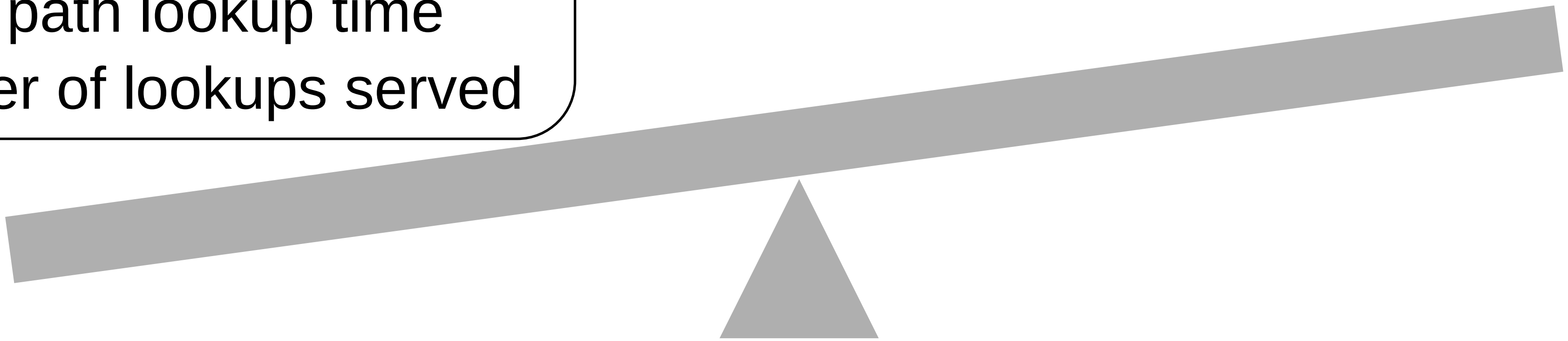
Learn!

Estimated benefit

- Baseline path lookup time
- Model path lookup time
- Number of lookups served

Estimated cost

- Table size



Effectiveness of Cost-Benefit Analyzer

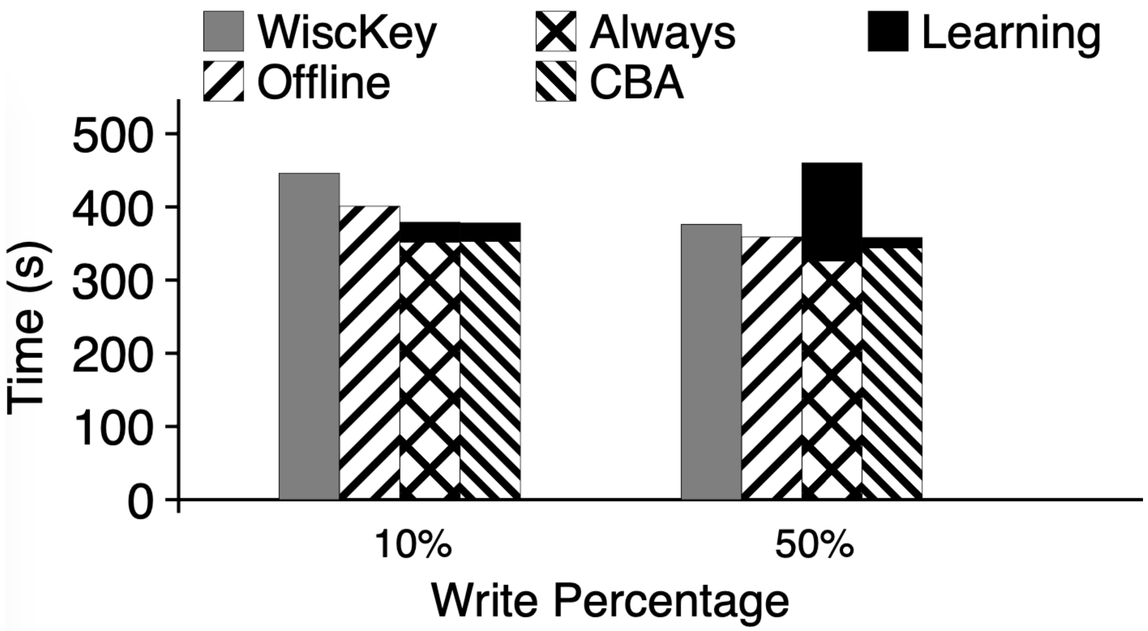
Learn most/all new tables at low write percentages

- Reach a better foreground latency than offline learning

Limit learning at high write percentages

- Reduce learning time and have a good foreground latency

Minimal total CPU cost in all scenarios



Evaluation

Various micro and macro benchmarks

- Dataset
- Load order
- Request distribution
- Range queries
- YCSB
- SOSD
- On-disk database

Database resides in memory

Reduce data access time

Better show benefits in indexing time

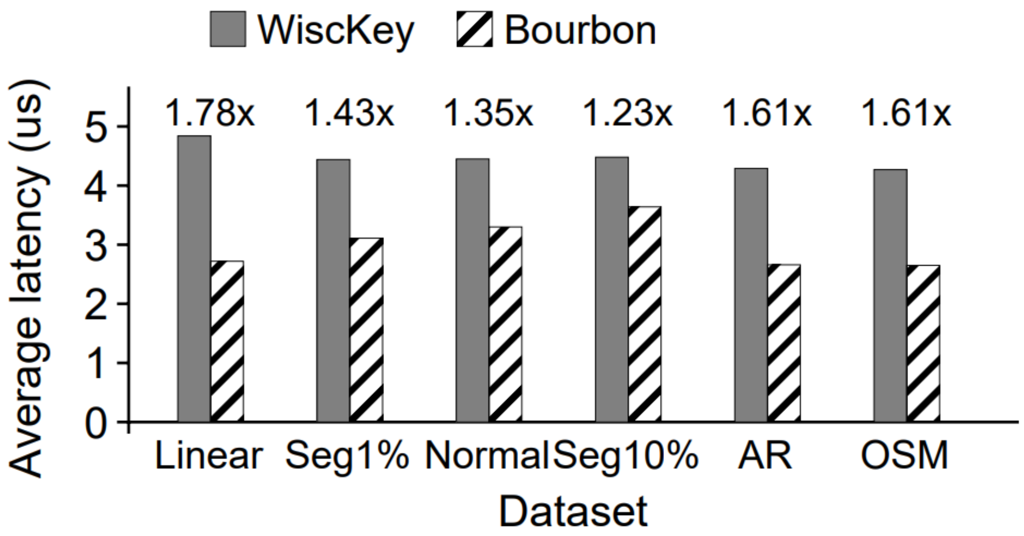
Come back to this condition later

Can Bourbon adapt to different datasets?

Micro benchmark: datasets

- 4 synthetic datasets: linear, normal, seg1%, and seg10%
- 2 real-world datasets: AmazonReviews and OpenStreetMapNY
- 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%



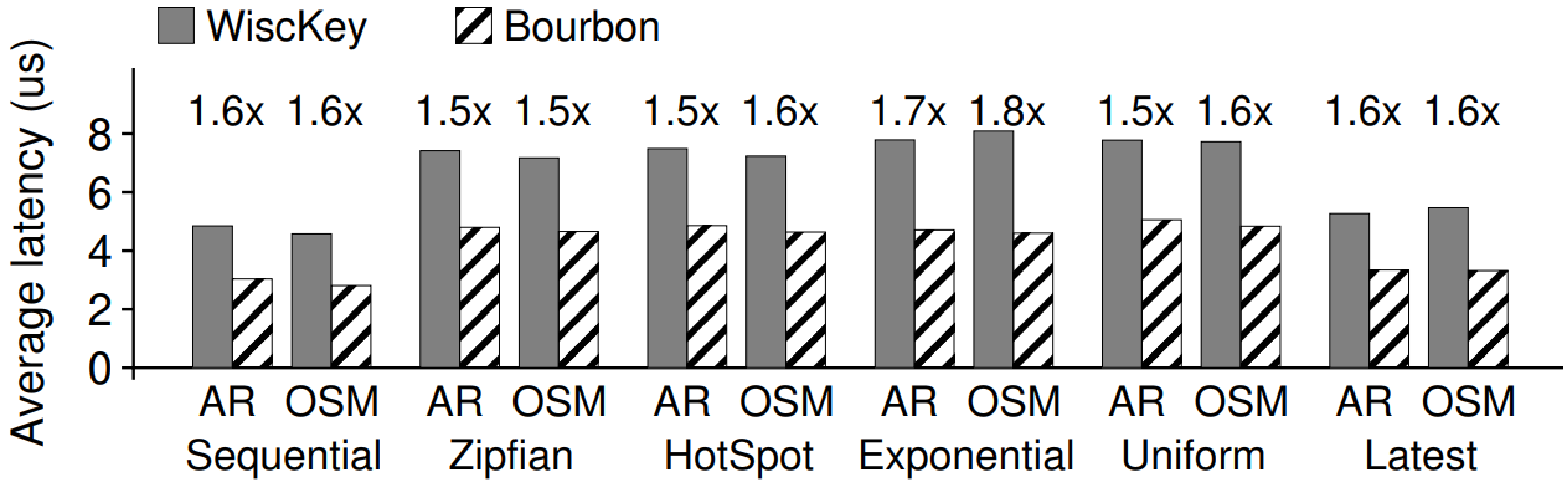
Bourbon performs better with lower number of segments

- Reach 1.6x 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



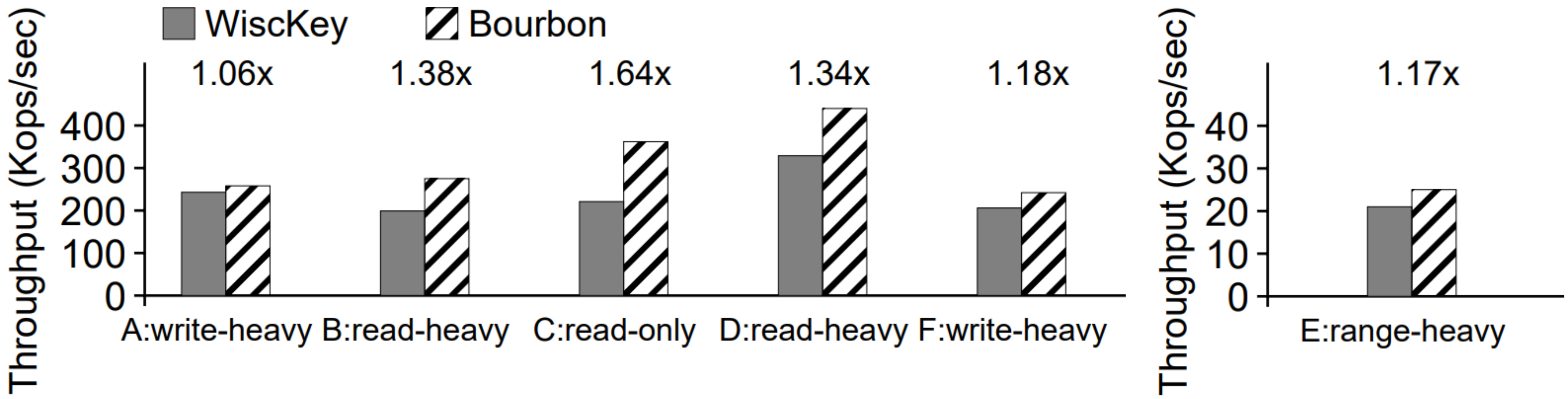
Bourbon improves performance by ~1.6x

- Regardless of request distributions

Can Bourbon perform well on real benchmarks?

Macro benchmark: YCSB

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



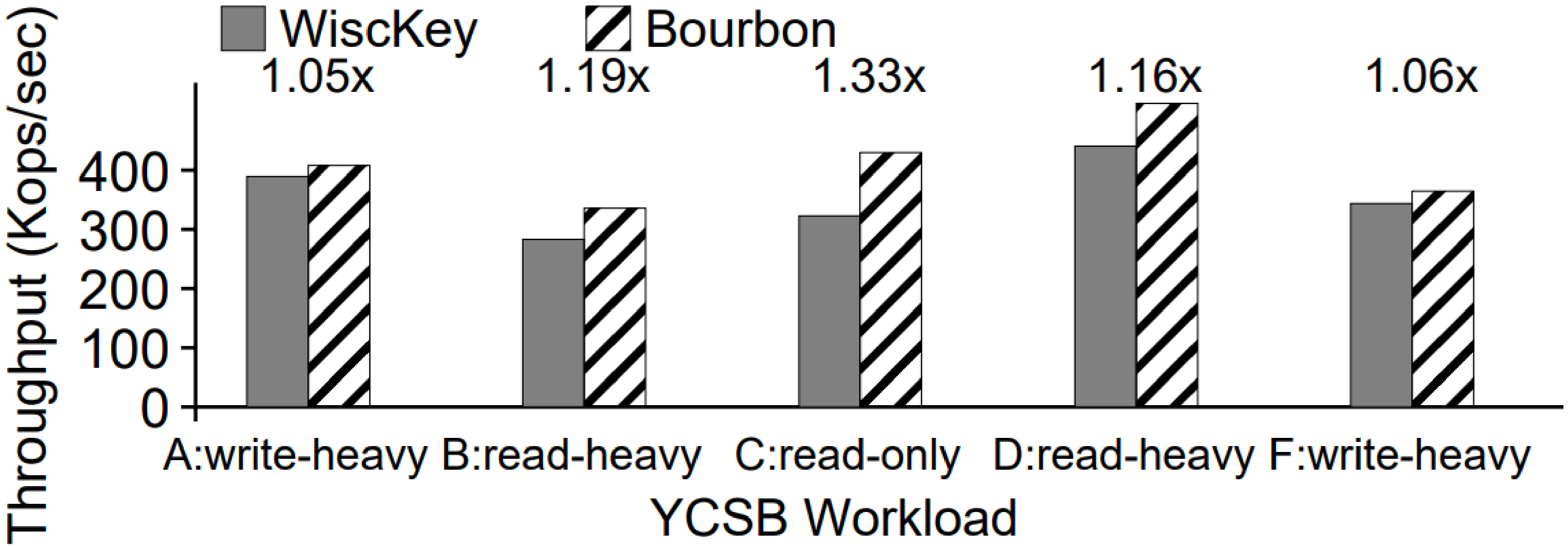
Bourbon's gain holds on real benchmarks

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

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 is 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

Thank You for Watching!

The ADvanced Systems Laboratory (ADSL)

https://research.cs.wisc.edu/wind/

Microsoft Gray Systems Laboratory

https://azuredata.microsoft.com/