

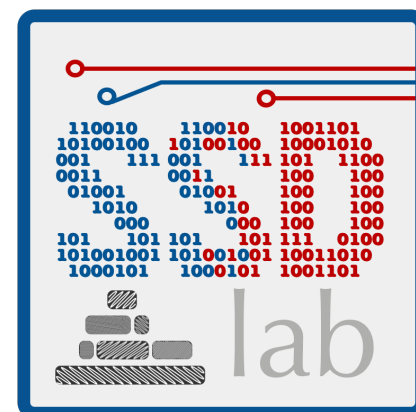
Tectonic: Bridging Synthetic and Real-World Workloads for Key-Value Benchmarking

Alexander H. Ott

James Chen

Shubham Kaushik

Subhadeep Sarkar



Brandeis
UNIVERSITY

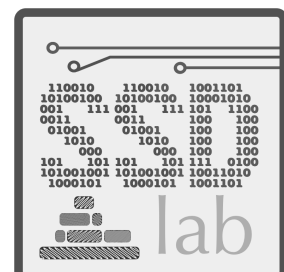
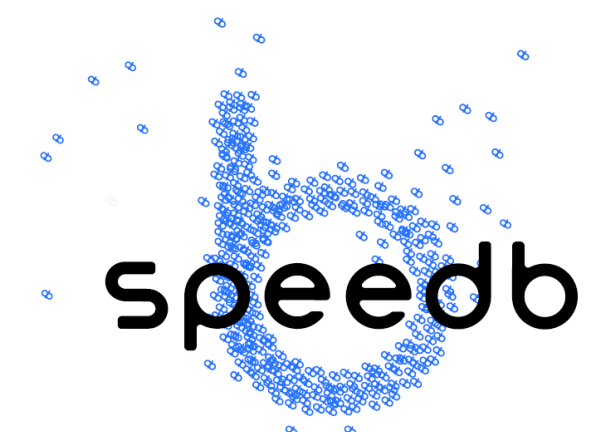


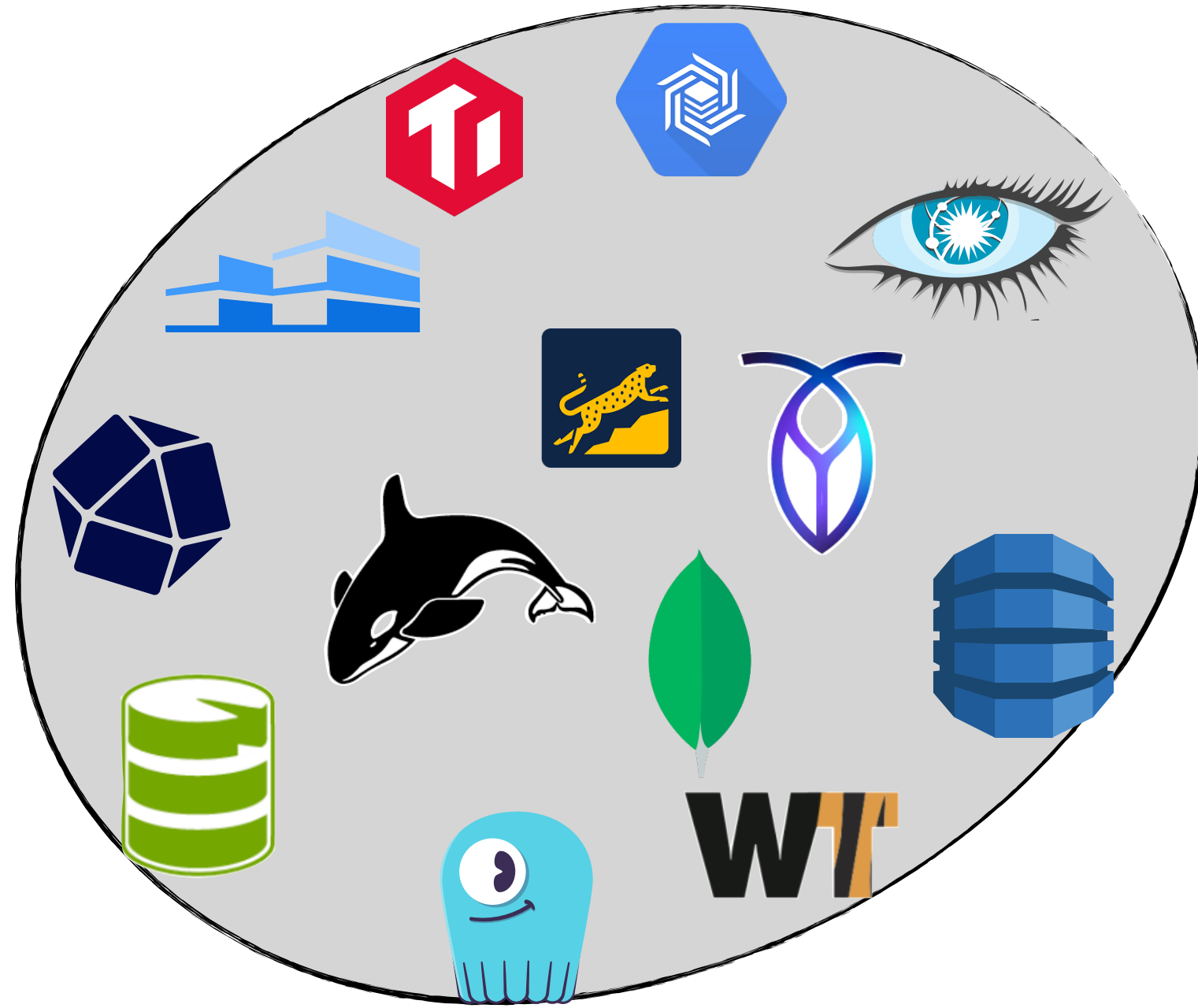


CockroachDB

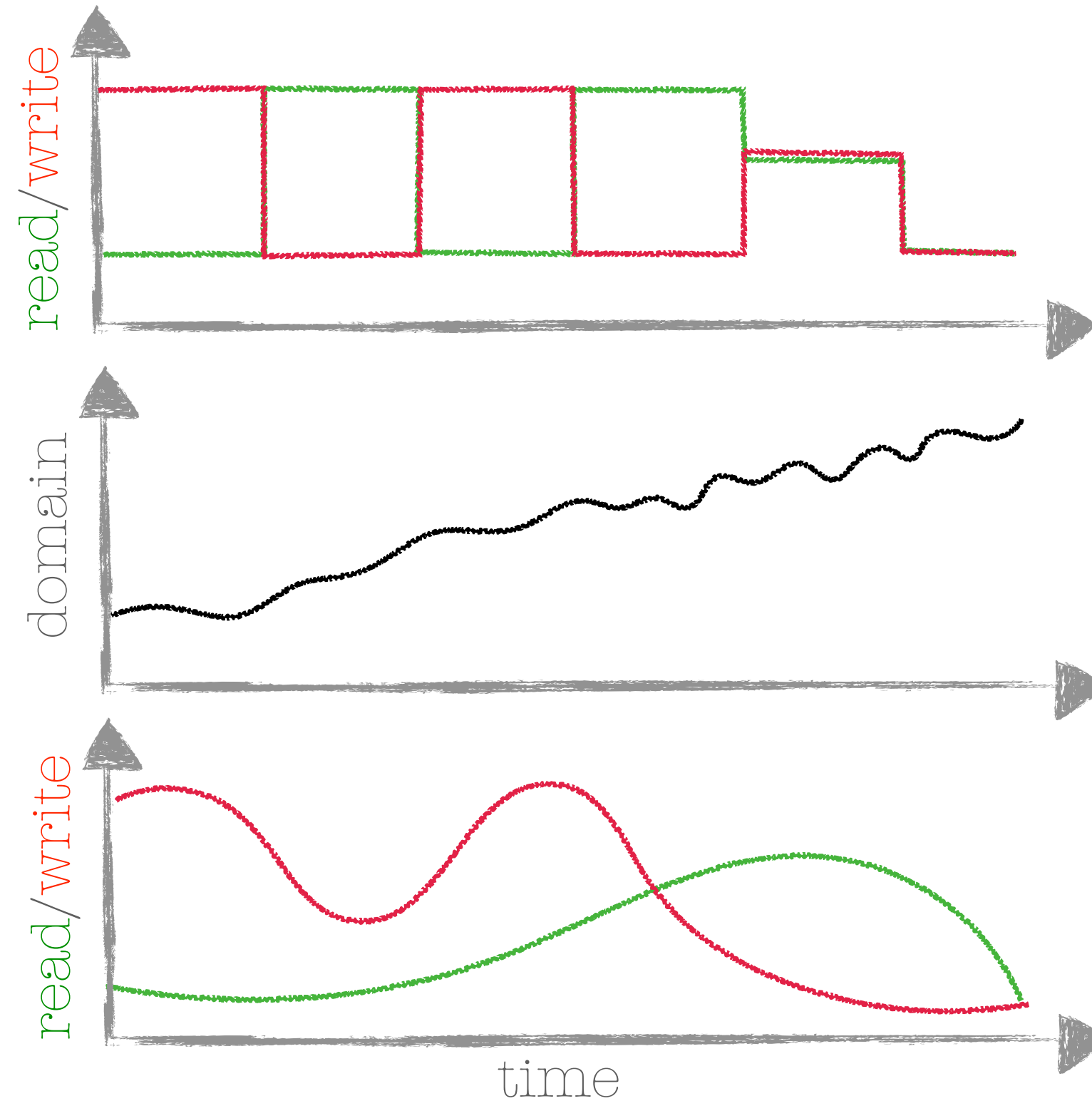


Key-Value Stores





many database vendors in the playground



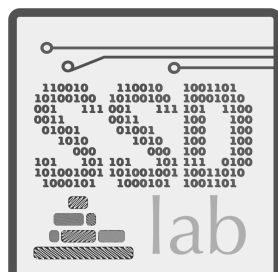
several workload scenarios for every application

How to **choose** the right database?

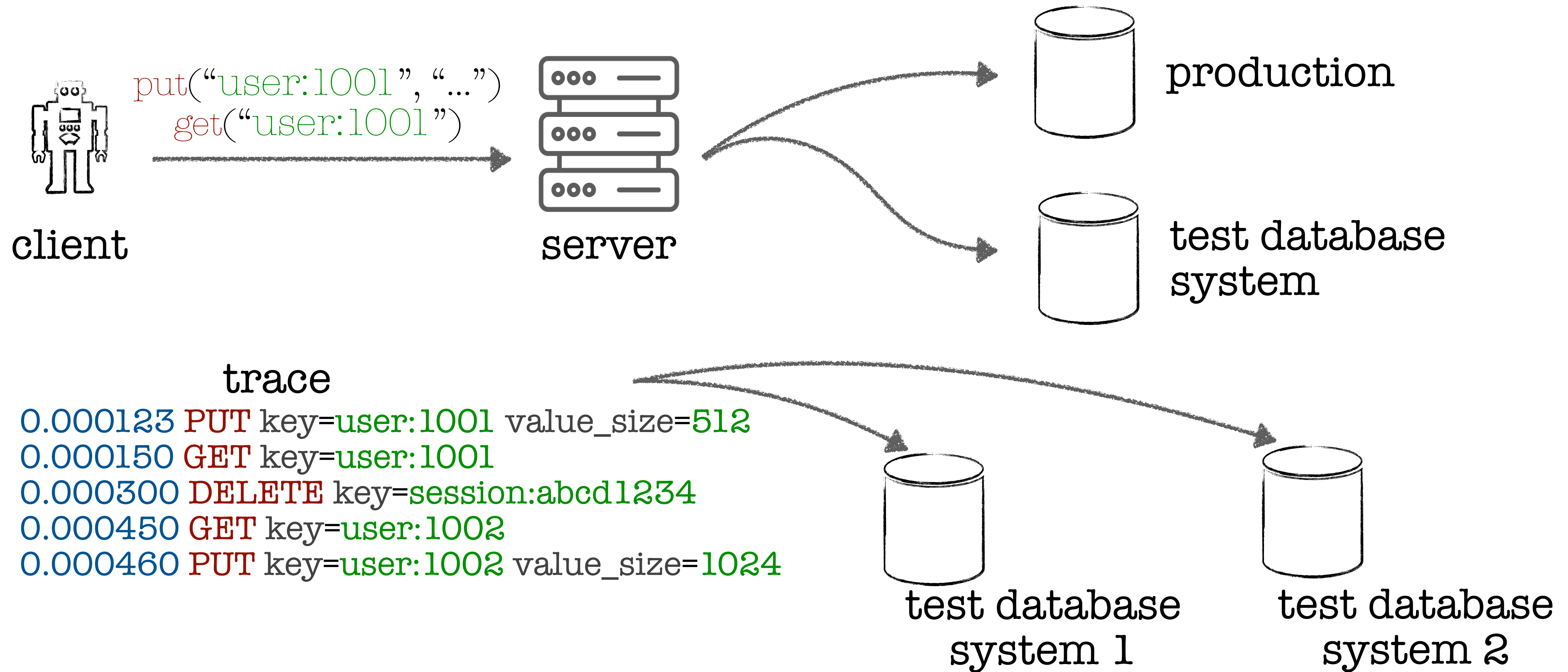
How to **tune** a database for specific goals?

How to **compare** database performance?

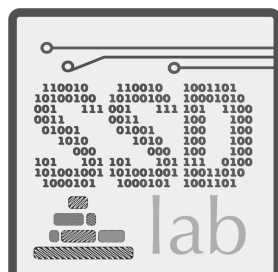
Benchmark databases



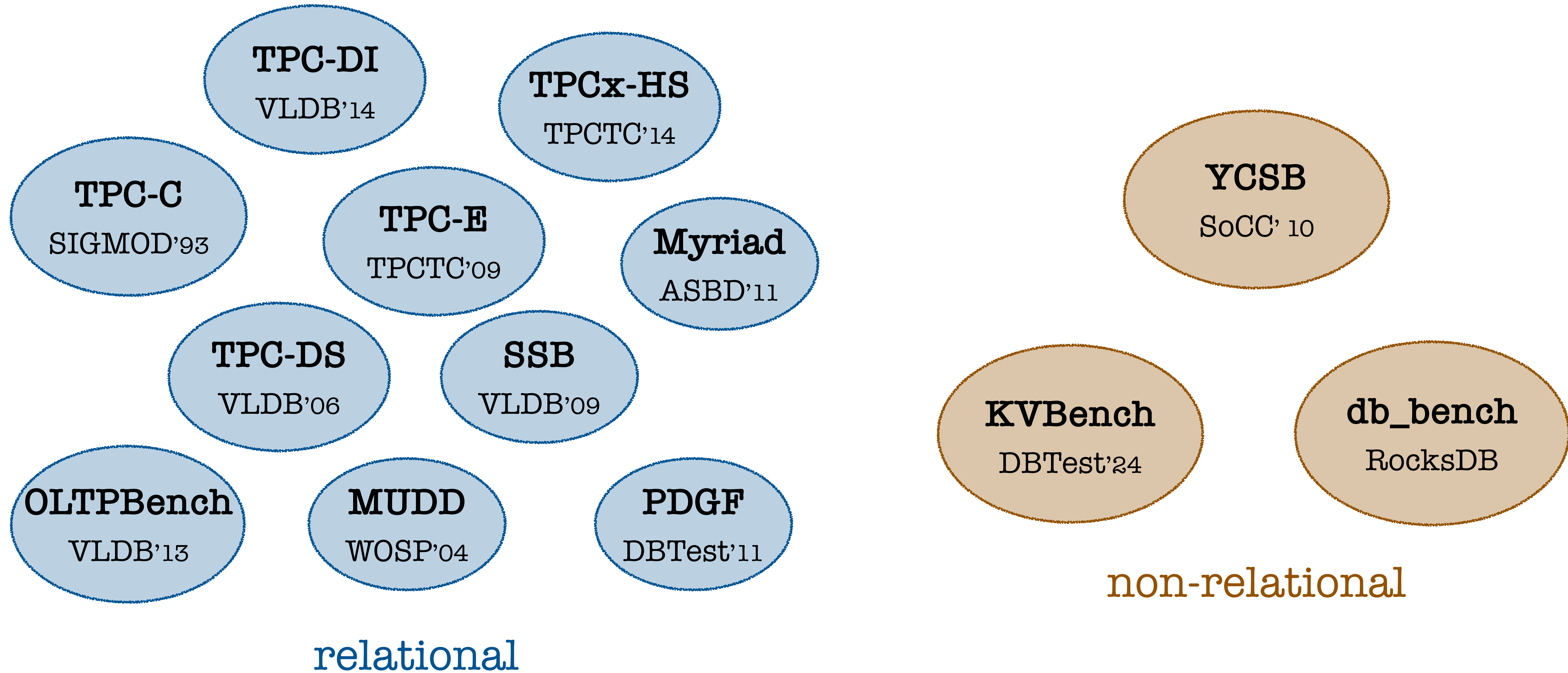
Shadow Testing & Trace Replays



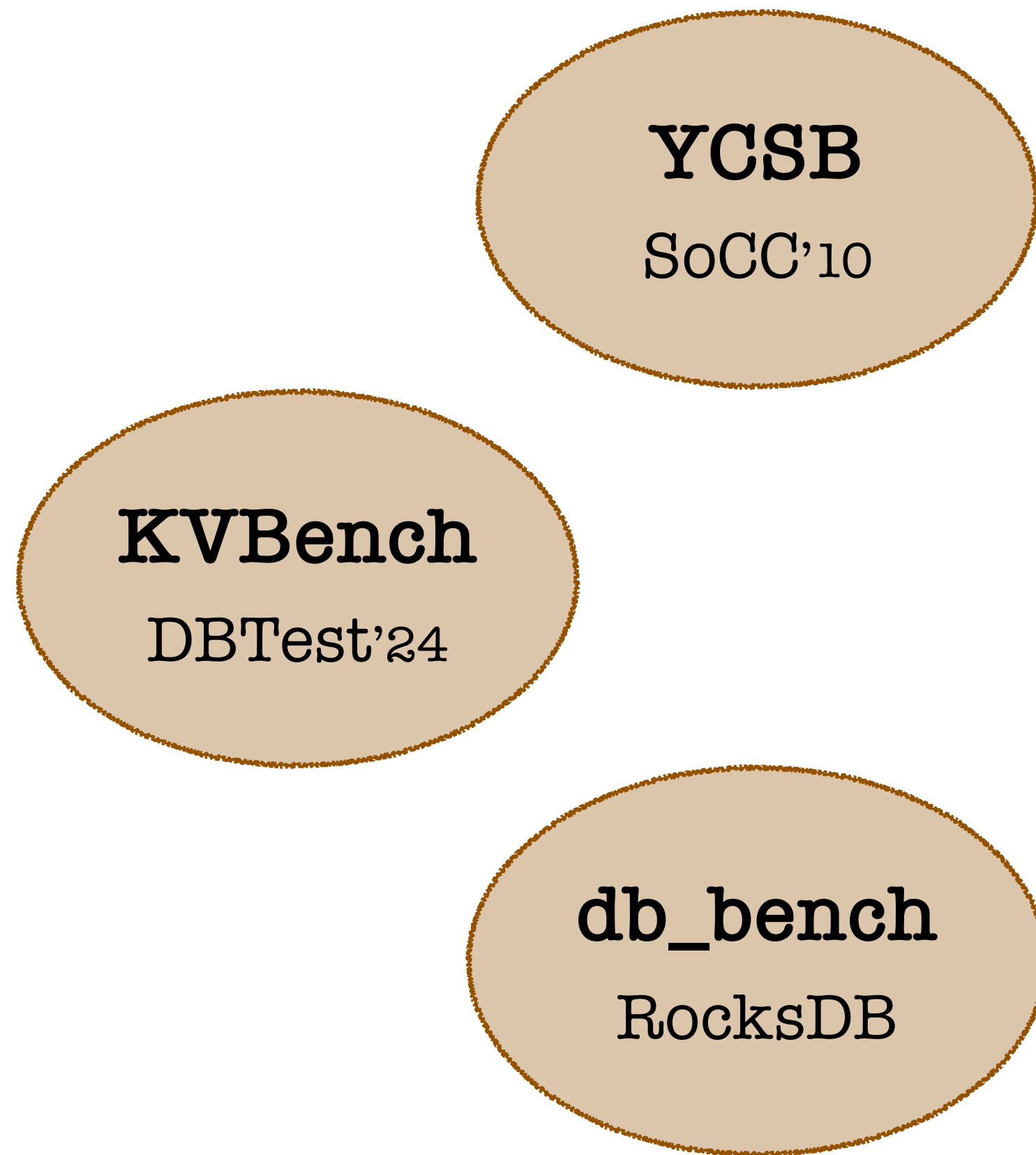
What if we don't have access to production data or traces?



Workload Generators / Benchmarks



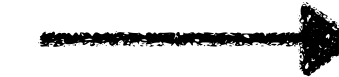
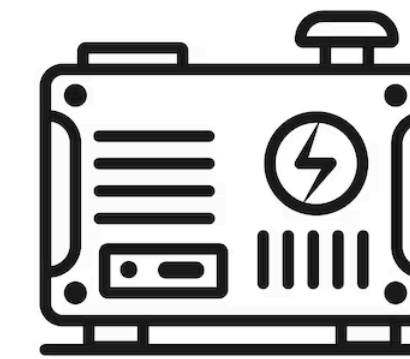
Workload Generators / Benchmarks



specifications

- put ●
- get ●
- scan ●
- delete ●
- ...

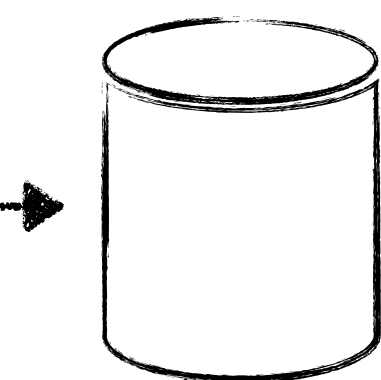
generator



on-the-fly

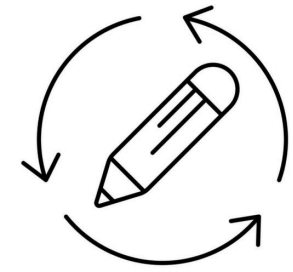
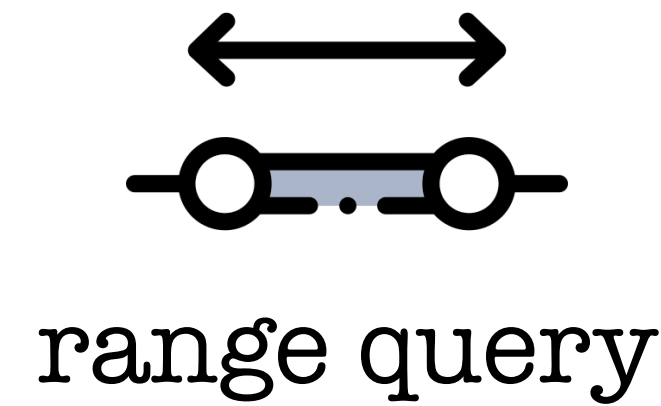
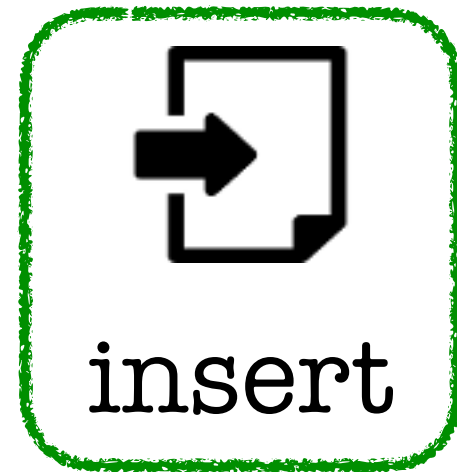
```
...  
get("k")  
put("k", "v")  
put("kk", "vv")  
put("k", "v2")  
scan("k", "kk")  
delete("k")  
put("kk", "vw")  
...
```

wrapper
program

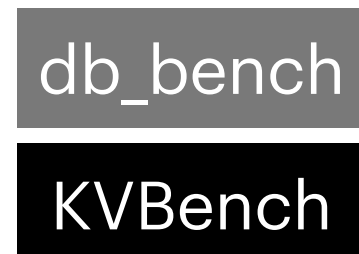


test
database system

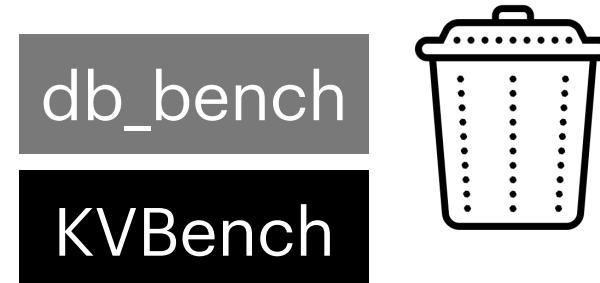
Workload Generators: Features



read-modify-write



delete



range delete

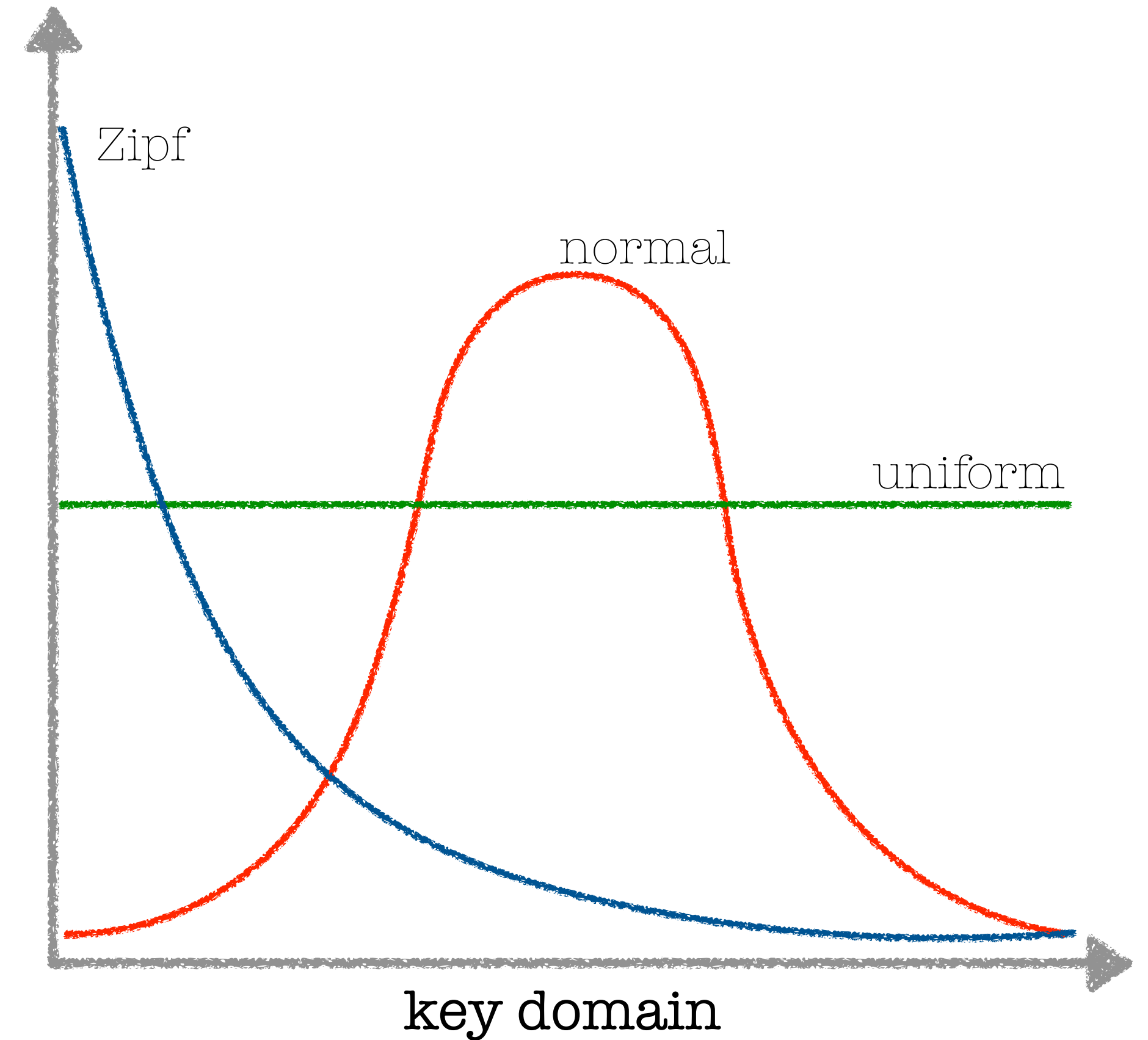


empty
point query



empty
point delete

supports different operations



offers flexible distribution

Limitations of the SoA

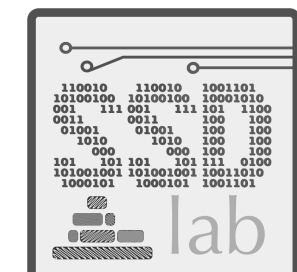
shifting workloads

nearly sorted workloads

read/write requests



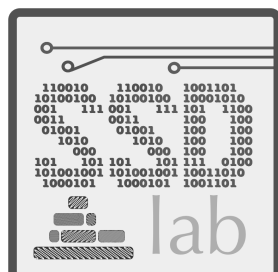
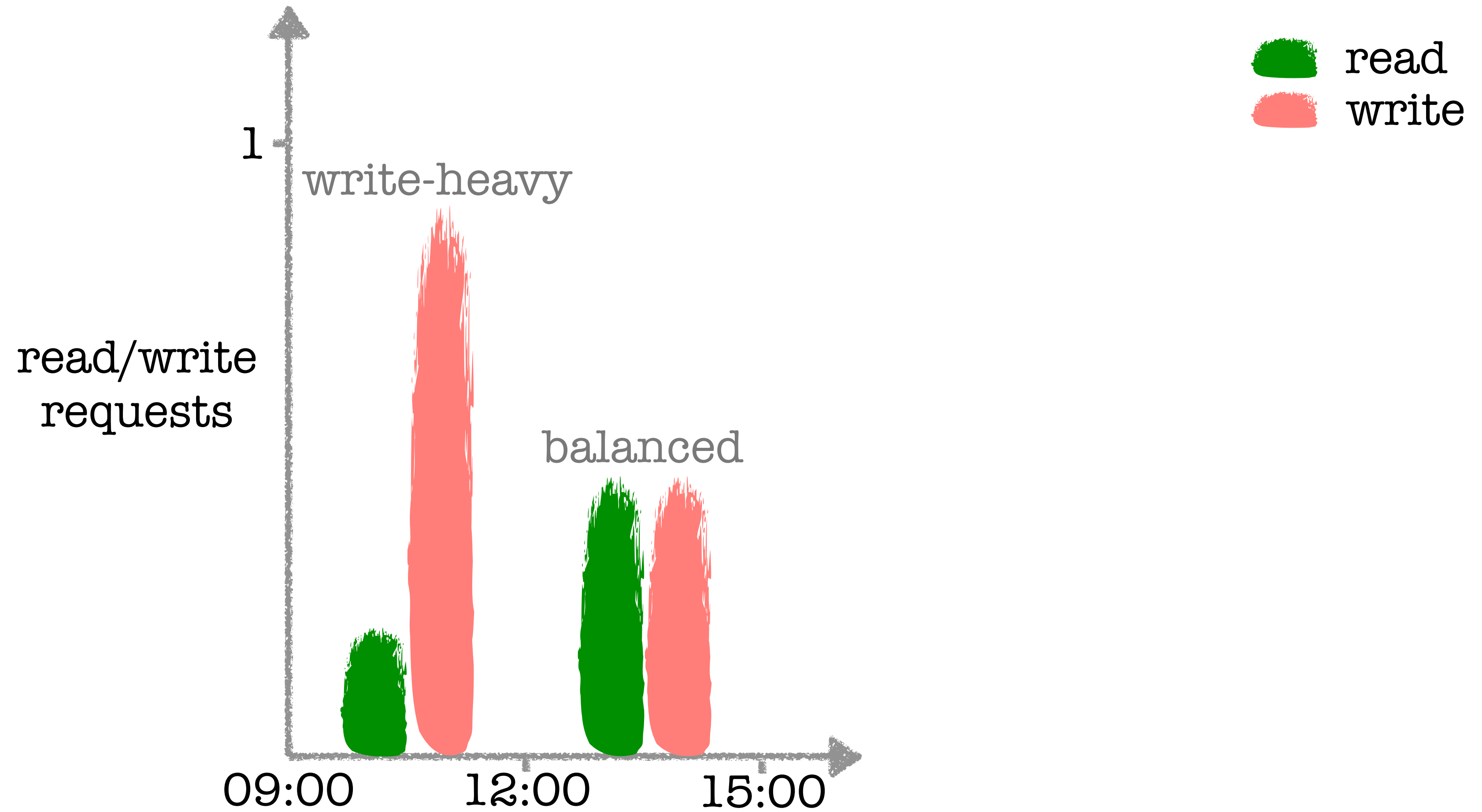
read
write



Limitations of the SoA

shifting workloads

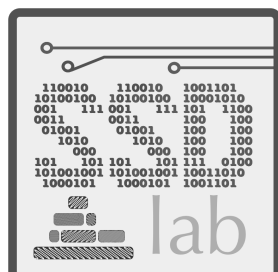
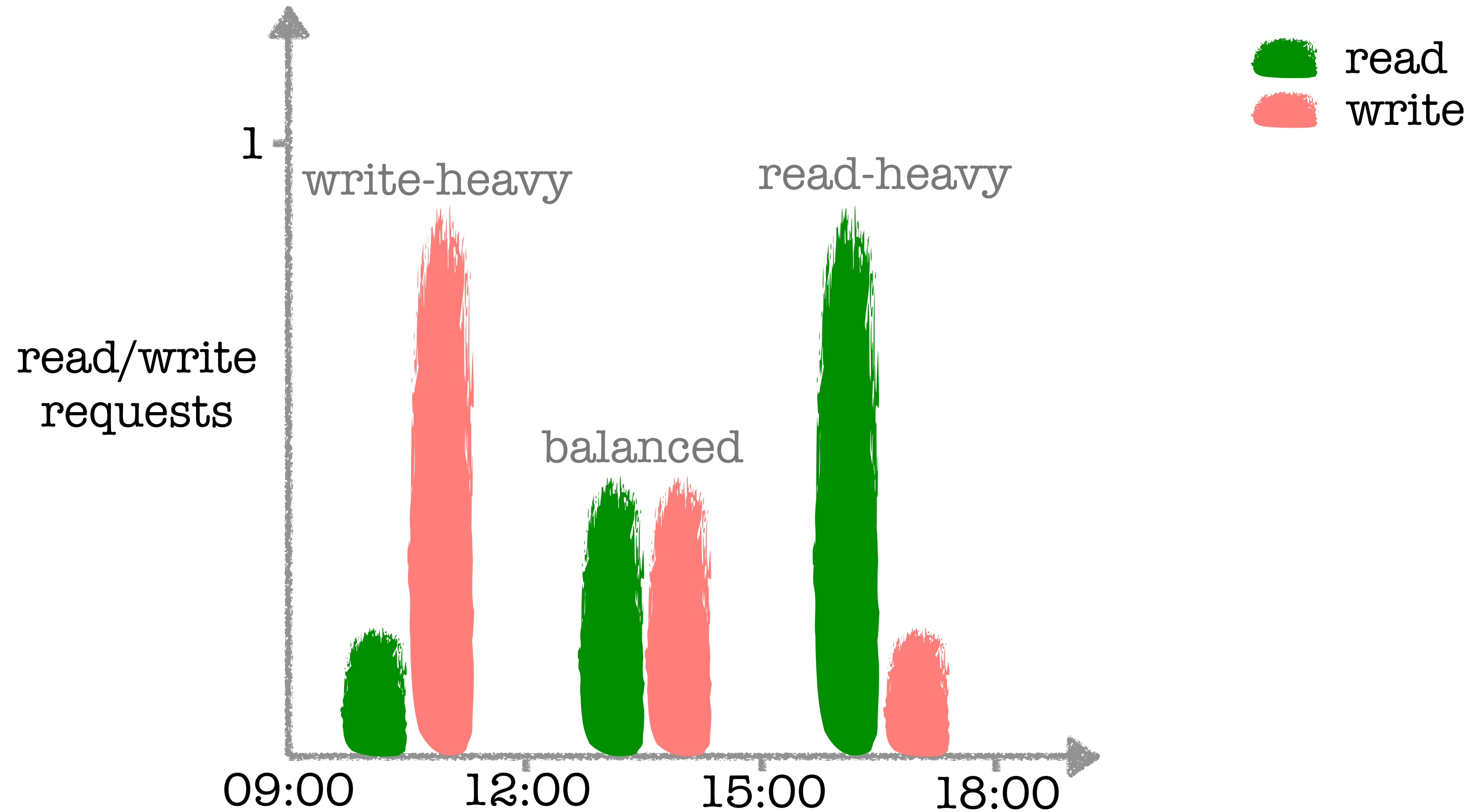
nearly sorted workloads



Limitations of the SoA

shifting workloads

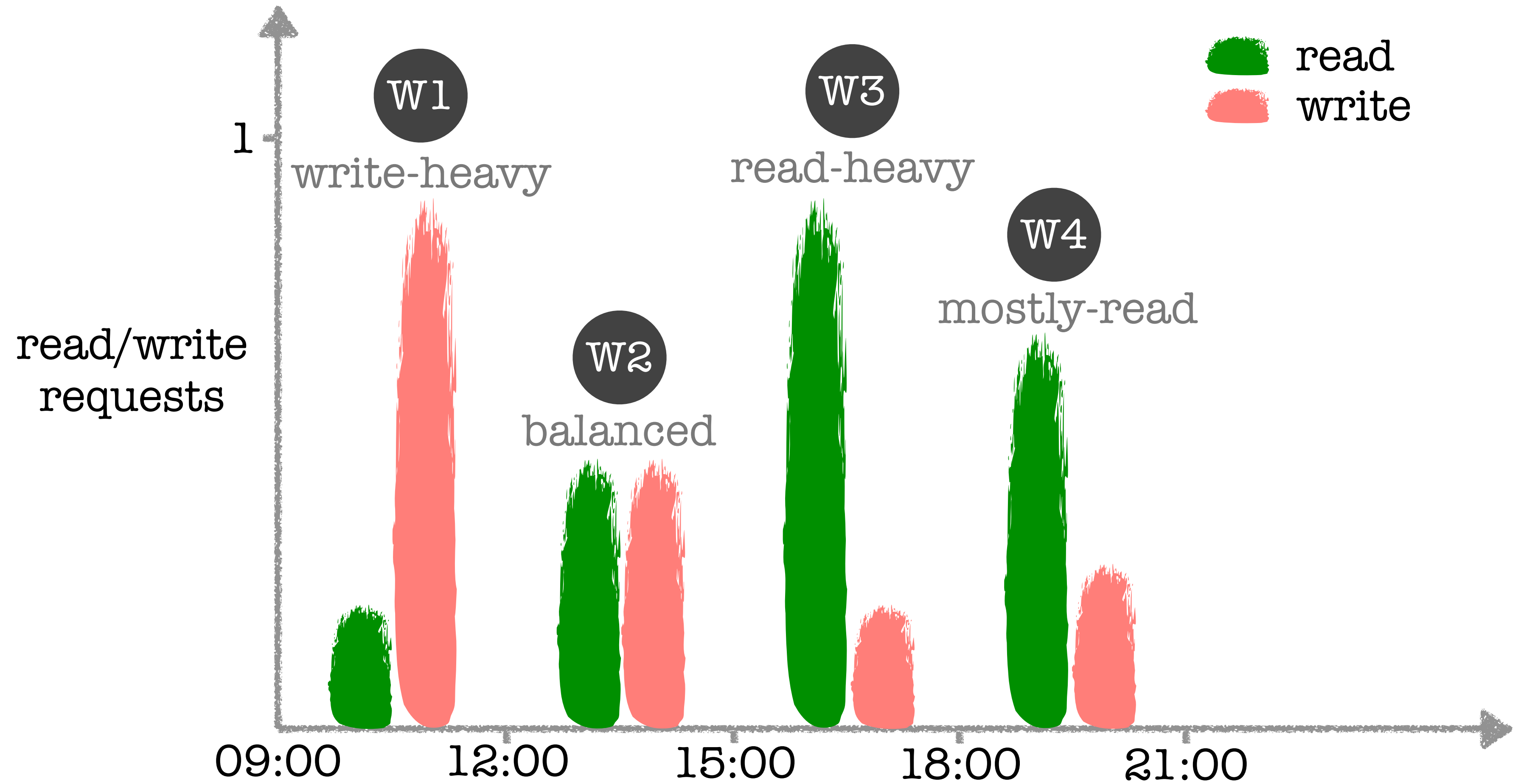
nearly sorted workloads



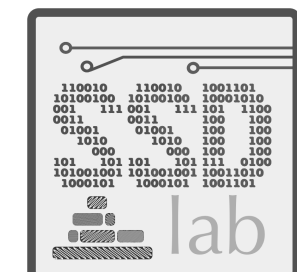
Limitations of the SoA

shifting workloads

nearly sorted workloads



How can we generate such shifting workloads?



Limitations of the SoA

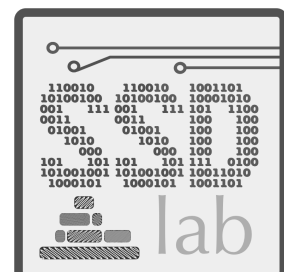
write-heavy W1

balanced W2

read-heavy W3

mostly-read W4

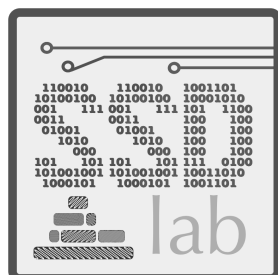
YCSB/KVBench



Limitations of the SoA

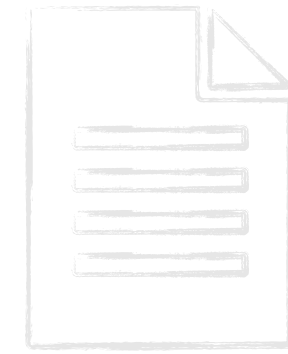


YCSB/KVBench



Limitations of the SoA

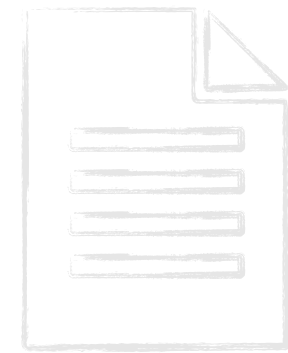
write-heavy W1



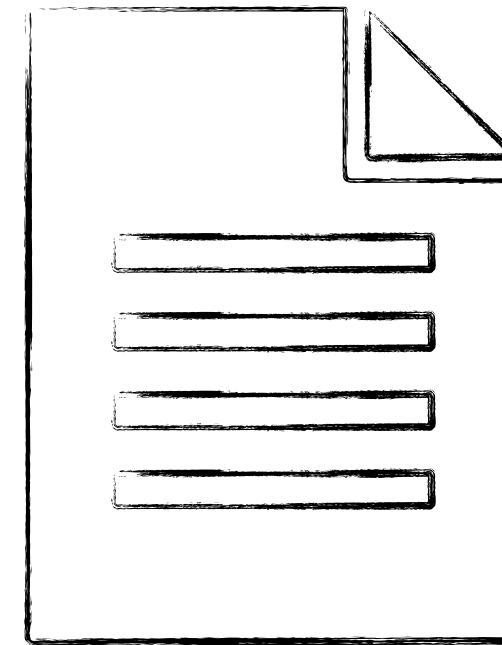
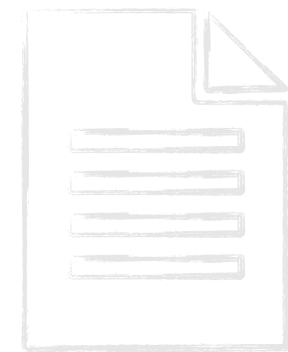
balanced W2



read-heavy W3



mostly-read W4



always human/script in the loop

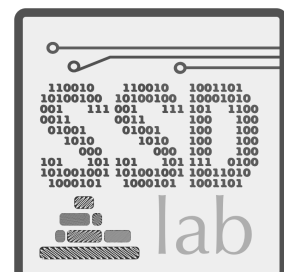


each workload is agnostic to the others



hard to mimic the real-world workloads

YCSB/KVBench



Limitations of the SoA

shifting workloads

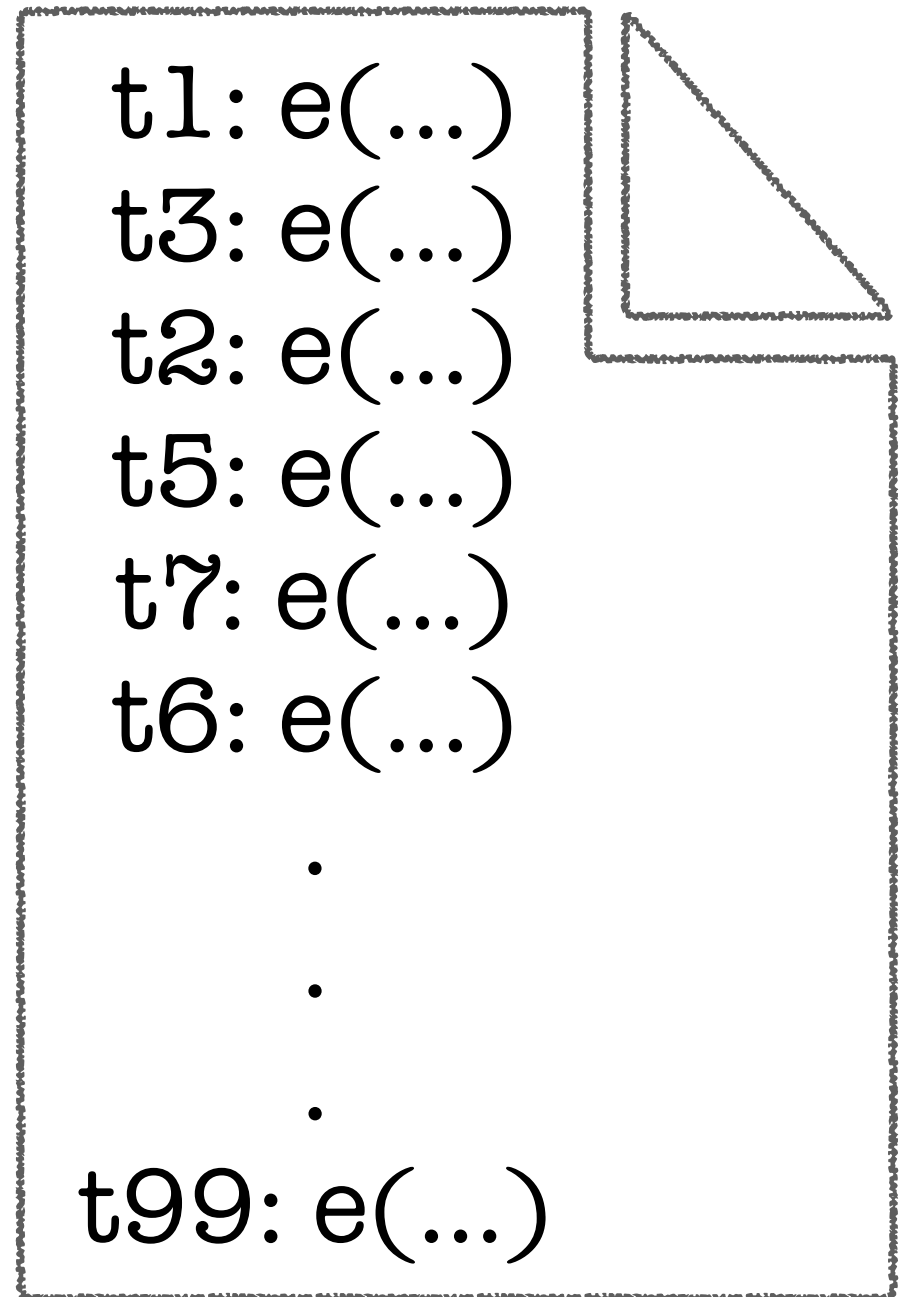


data warehouse (TPC-H)

nearly sorted workloads

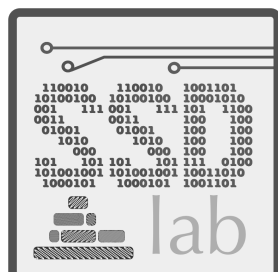


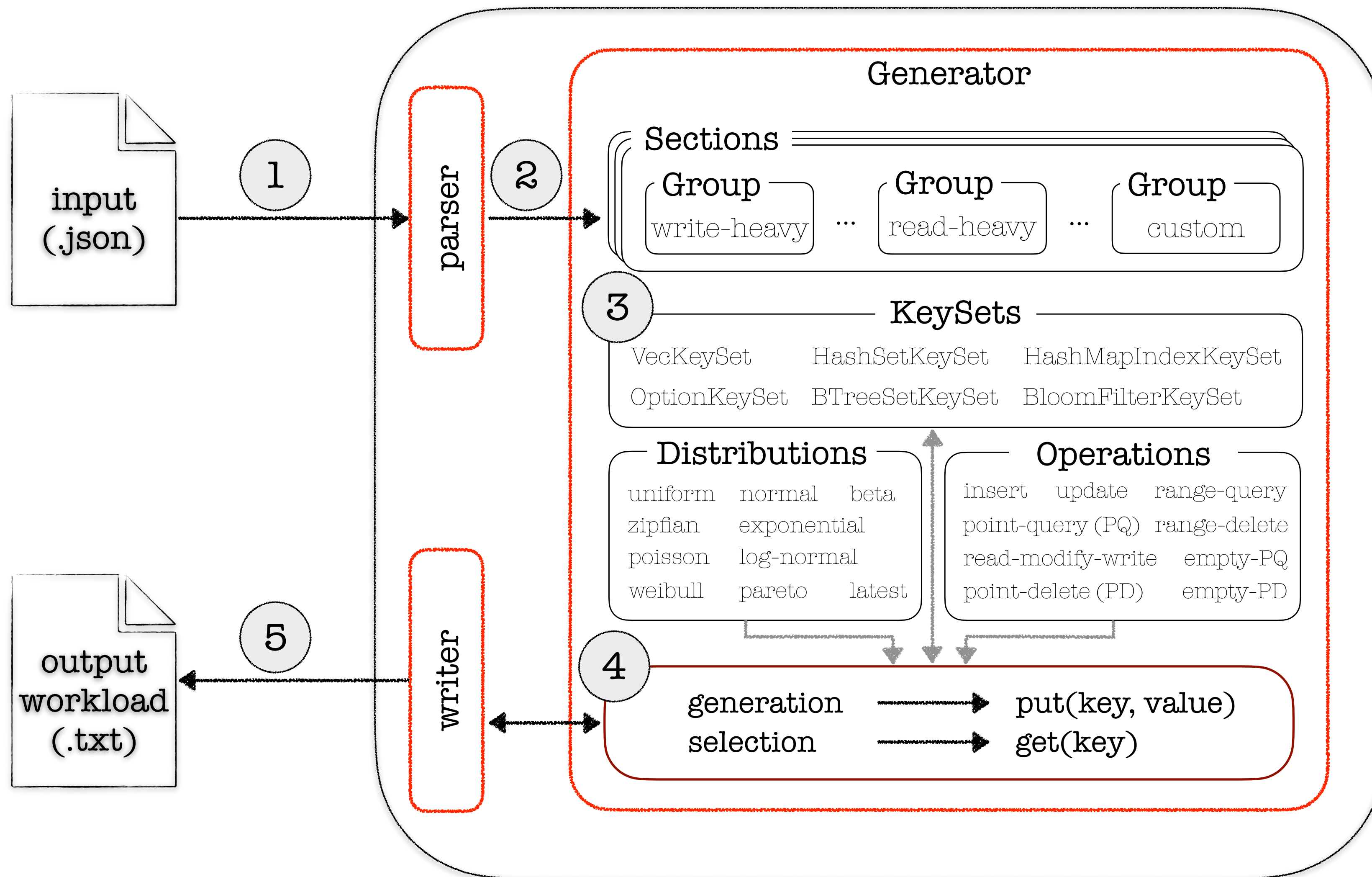
stock market trading



nearly sorted

YCSB/KVBench does not support nearly-sorted data generation





Section

represents one phase of a workload

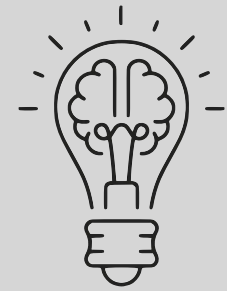
Group

a set of interleaved operations that executes in a section

KeySet

a data structure that helps in generating operations more optimally

Tectonic: Features



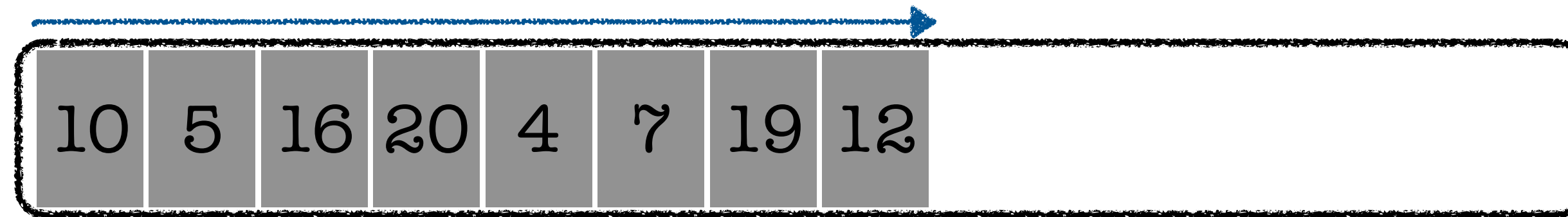
advanced operations

● inserts vs updates

● empty vs non-empty

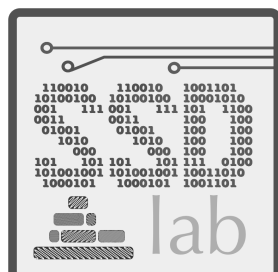
● range delete

KeySet
(vector)



vector (unsorted)

✗ Inserts



Tectonic: Features



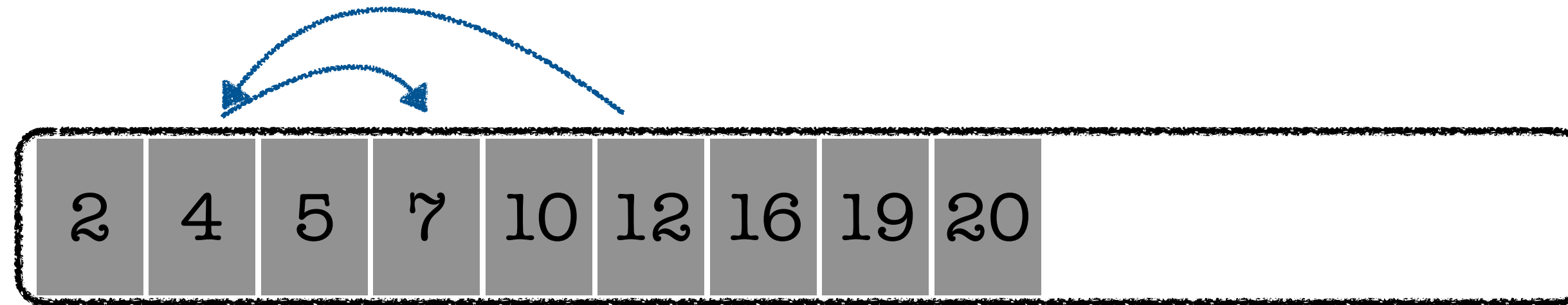
advanced operations

● inserts vs updates

● range delete

● empty vs non-empty

KeySet
(vector)



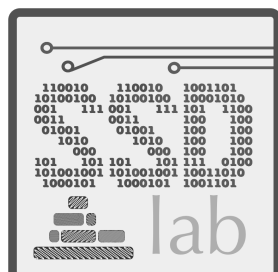
vector (unsorted)

✗ Inserts

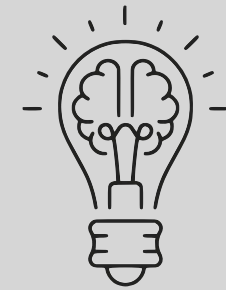
vector (sorted)

✗ Inserts

— Updates + point queries



Tectonic: Features

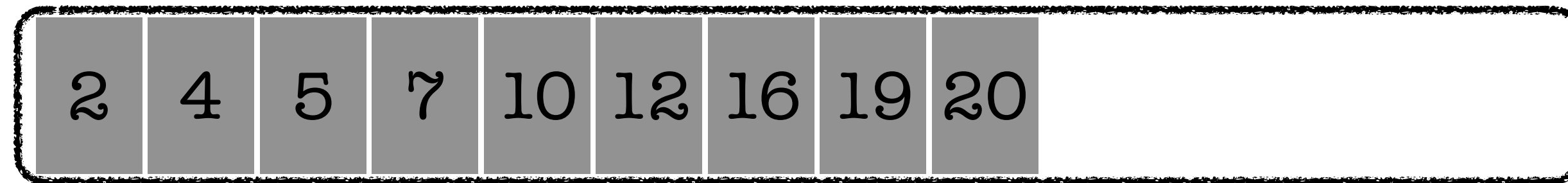


advanced operations

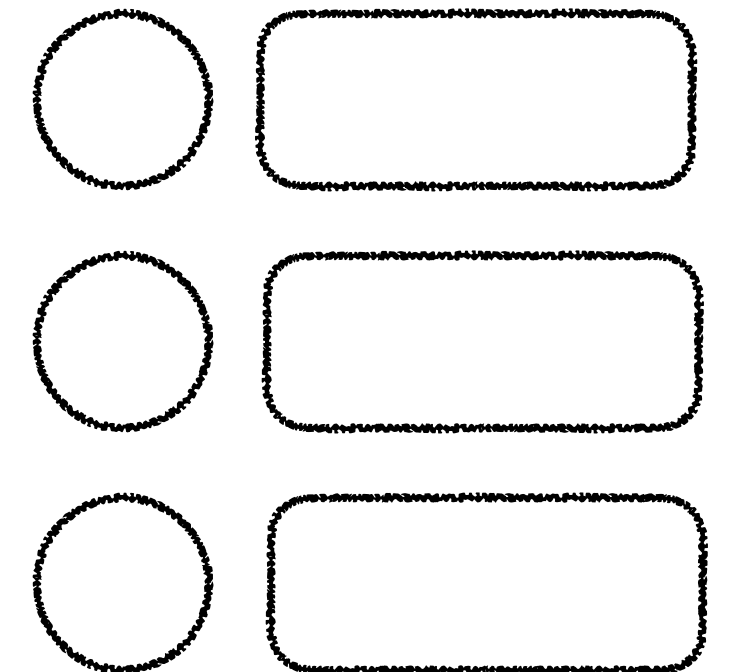
- inserts vs updates
- empty vs non-empty

- range delete

KeySet
(vector)



KeySet
(hash table)



vector (unsorted)

✗ Inserts

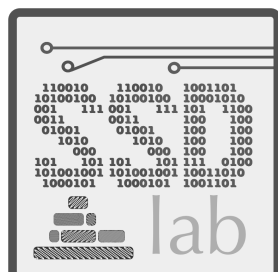
vector (sorted)

✗ Inserts
— Updates + point queries

hash set

✓ Inserts
✗ Updates + point queries

What is the appropriate data structure?



Tectonic: Features - KeySet

VecKeySet



- ✓ blind inserts
- ✓ updates
- ✓ point queries

- ✓ range queries
- ✗ empty queries
- ✗ deletes

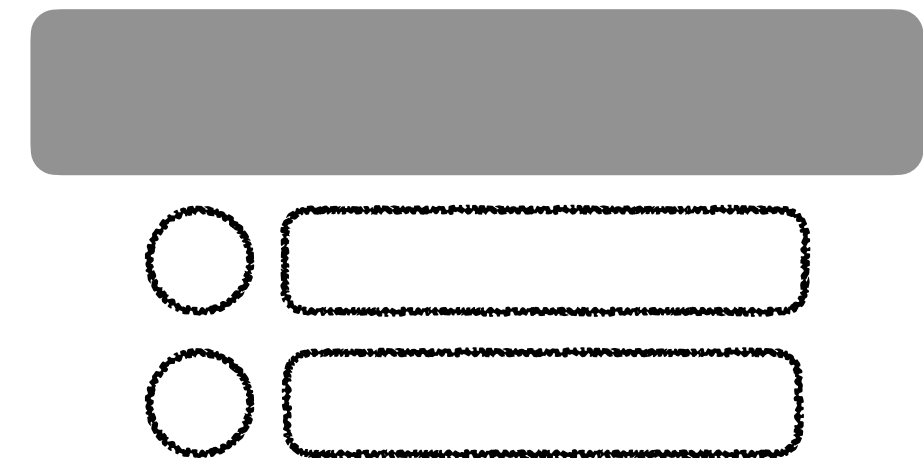
BloomFilterKeySet



- ✓ empty deletes
- ✓ empty queries

- ✓ unique inserts
- ✗ non-empty deletes

HashMapIndexKeySet



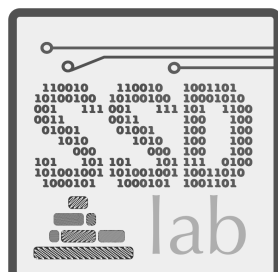
- ✓ non-empty deletes
- ✗ range queries
- ✗ updates

OptionKeySet



- ✓ non-empty deletes
- ✓ range deletes

- ✓ updates
- ✓ range queries

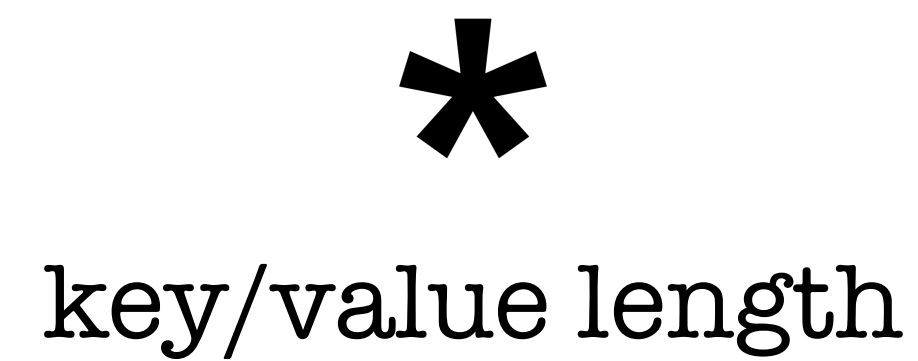
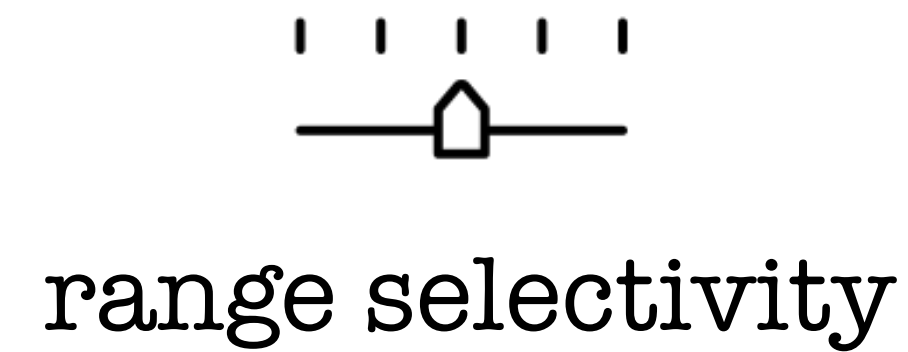
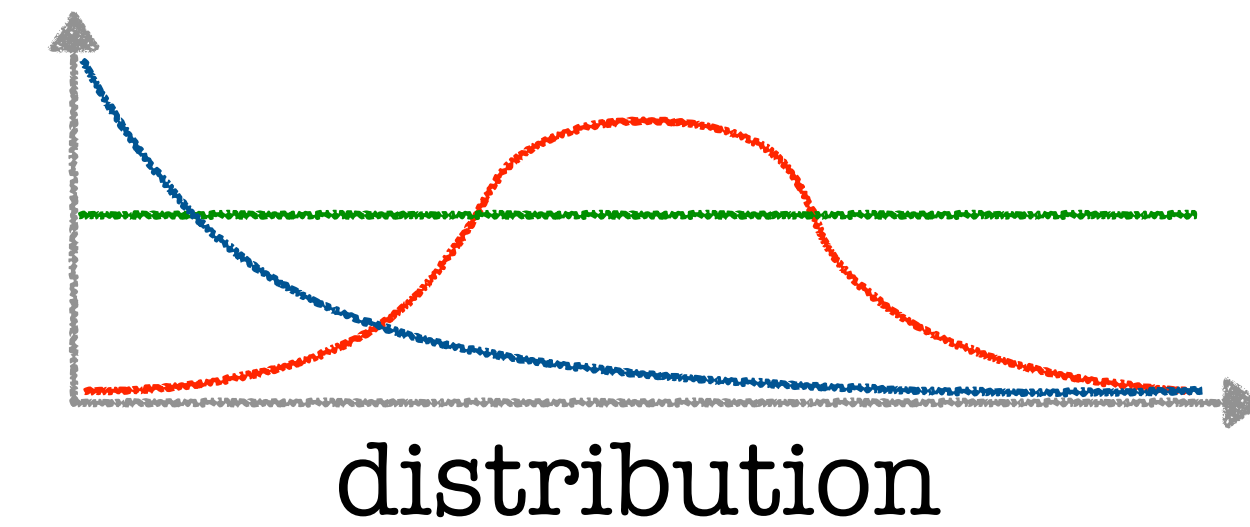
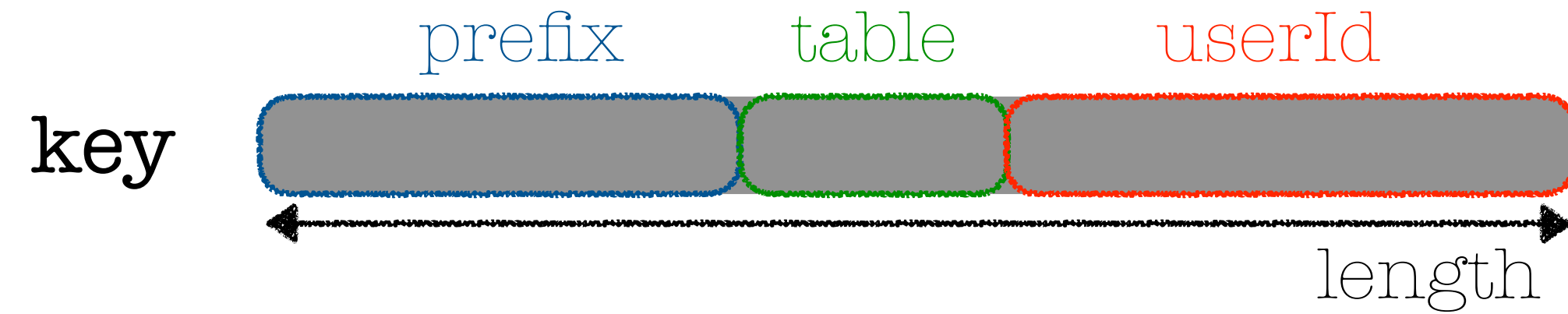


Tectonic: Features



- prefix-based keys
- different distribution

- varied key-value lengths
- varied range of selectivity

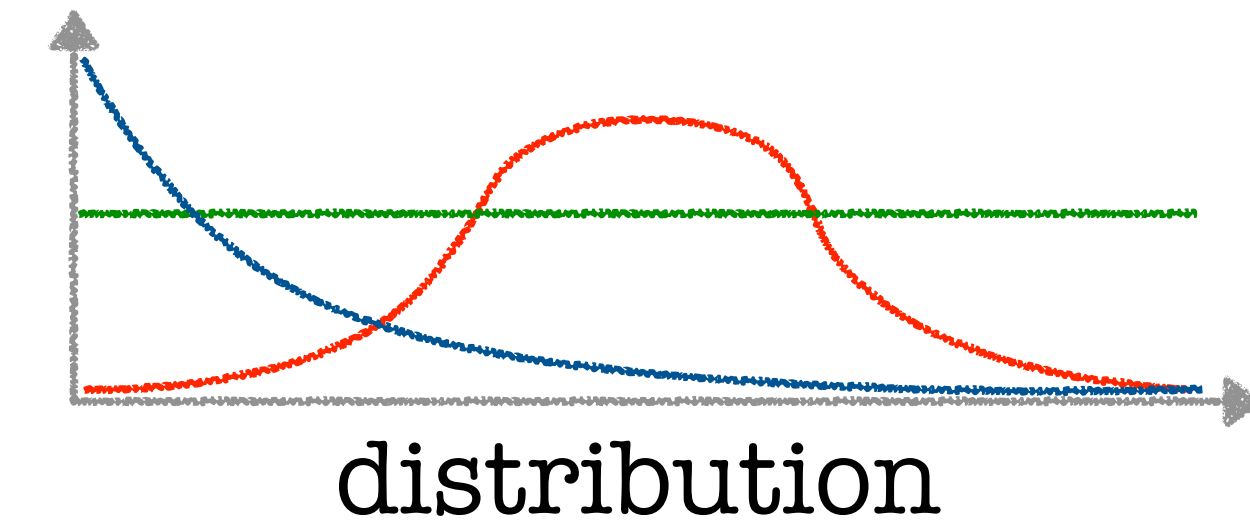
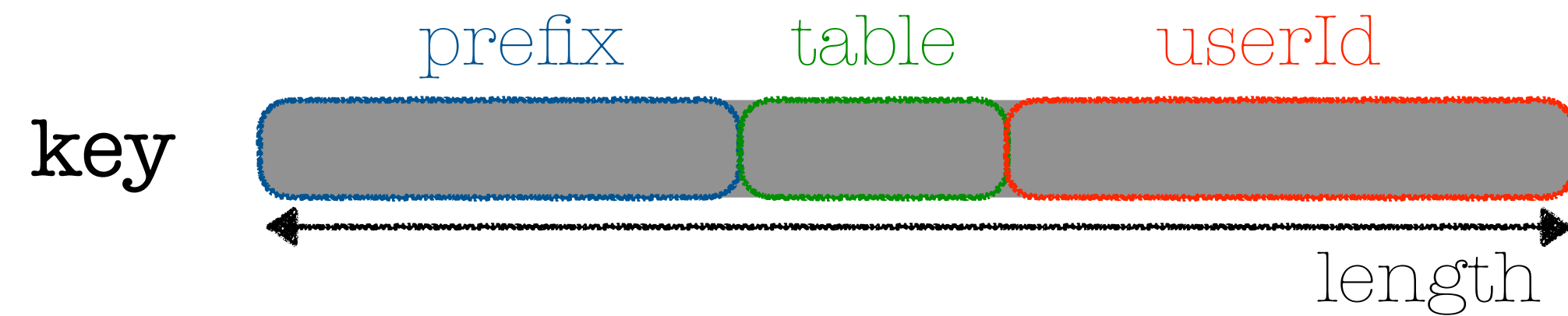


Tectonic: Features



- prefix-based keys
- different distribution

- varied key-value lengths
- varied range of selectivity



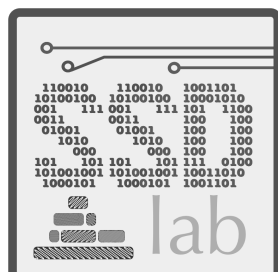
usertable:user:12345678901234567

prefix: usertable:user

separator: ":"

length: 17
distribution: uniform

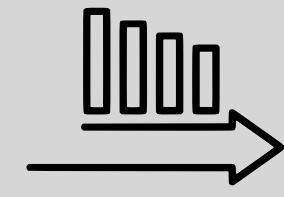
```
“segmented”: {  
  “separator”: “:”,  
  “segments”: [  
    “usertable:user”,  
    {  
      “uniform”: {  
        “len”: 17,  
        “character_set”: “numeric”  
      }  
    }  
  ]  
}
```



Tectonic: Features

Ben-Moshe et al., "Detecting and exploiting..." (EDBT/ICDT '11)

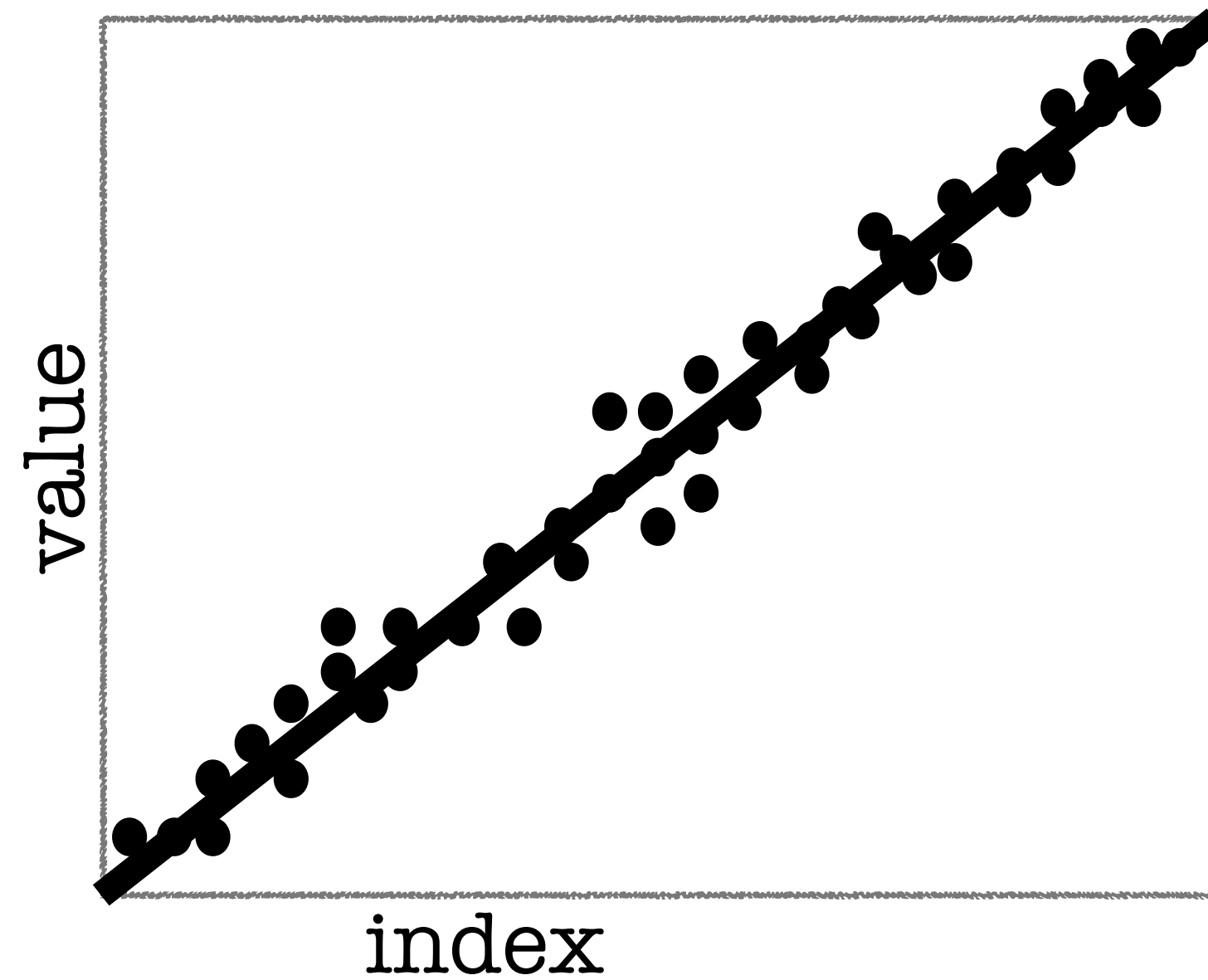
Raman et al., "BoDS" (TPCTC'22)



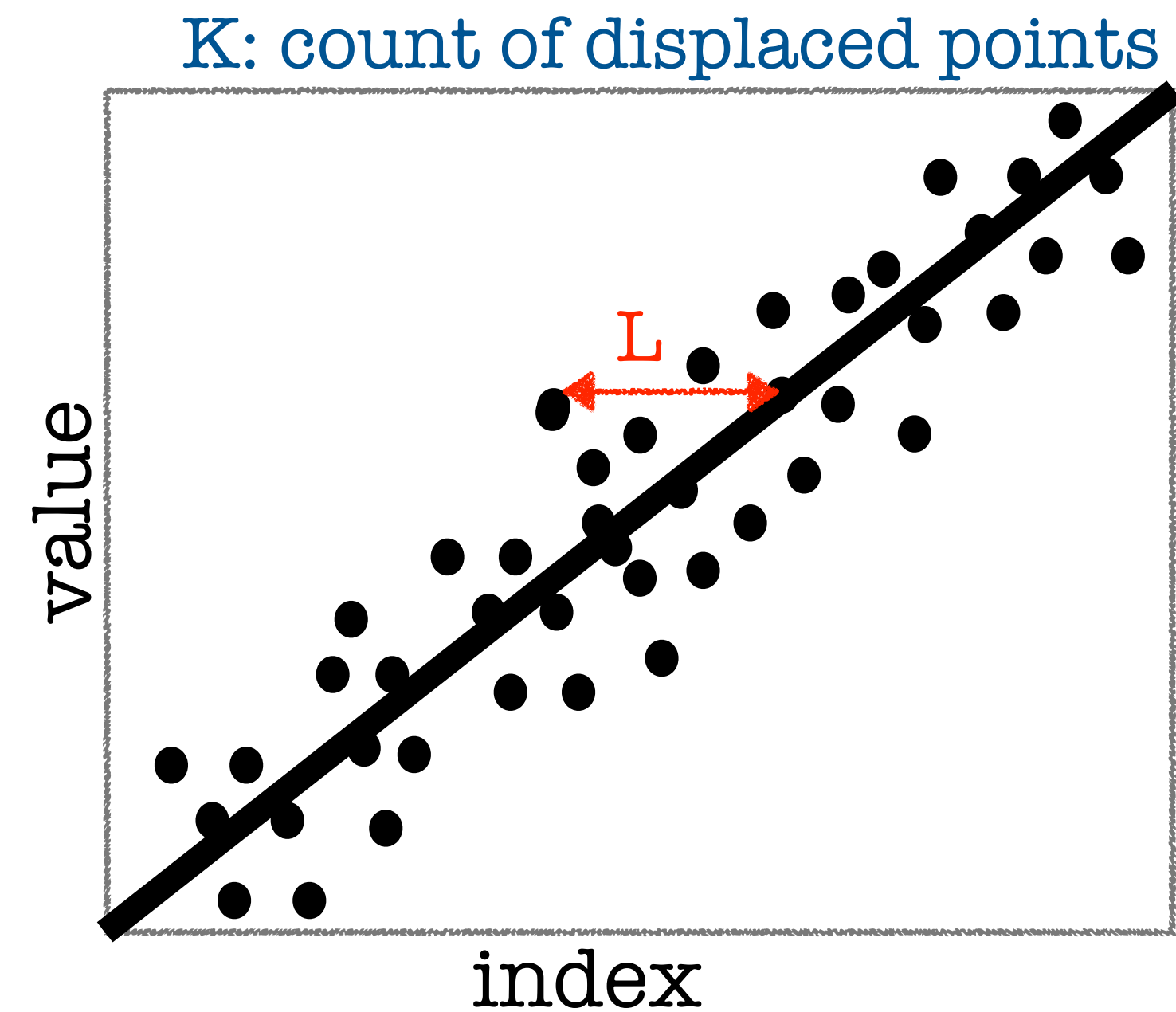
varied sortedness

● fully sorted workloads

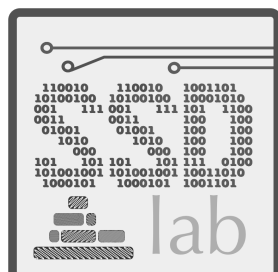
● nearly sorted workloads



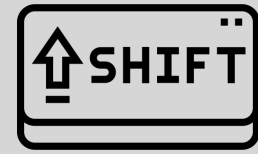
```
"sorted": {  
  "k": 10000,  
  "l": {  
    "uniform": { "min": 0, "max": 10 }  
  }  
}
```



```
"sorted": {  
  "k": 10000,  
  "l": {  
    "uniform": { "min": 0, "max": 1000 }  
  }  
}
```



Tectonic: Features



shifting workloads

● generates a shifting workload

Sections

Group
write-heavy

...

Group
read-heavy

...

Group
custom

mimic the stock exchange workload

Monday

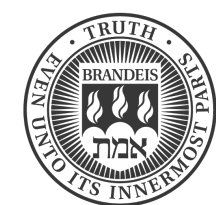
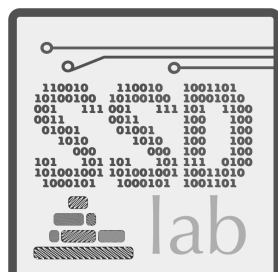
Phase 1: open market

Phase 2: close market

Tuesday

Phase 1: open market

Phase 2: close market



Brandeis
UNIVERSITY

Tectonic Demo

Monday

Phase 1: open market

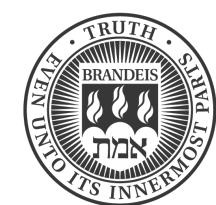
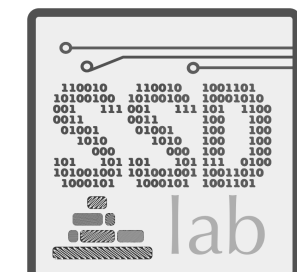
```
“sorted”: {  
  “k”: 1000,  
  “l”: {...}  
},  
“inserts”: {  
  “op_count”: 1M,  
  “key”: {...},  
  “val”: {  
    “uniform”: { “len”: 64 }  
  }  
},  
“point_queries”: {  
  “op_count”: 100K  
}
```

Group 1

Phase 2: close market

```
“point_queries”: {  
  “op_count”: 1M,  
  “selection”: {  
    “beta”: {  
      “alpha”: 0.5,  
      “beta”: 1.2  
    }  
  }  
}
```

Group 2



Tectonic Demo

Section A

Monday

```
“sorted”: {
  “k”: 1000,
  “l”: {...}
}

“inserts”: {
  “op_count”: 1M,
  “key”: {...},
  “val”: {
    “uniform”: { “len”: 64 }
  }
},

“point_queries”: {
  “op_count”: 100K
}

“point_queries”: {
  “op_count”: 1M,
  “selection”: {
    “beta”: {
      “alpha”: 0.5,
      “beta”: 1.2
    }
  }
}
}
```

Group 1

Group 2

Section B

Tuesday

```
“sorted”: {
  “k”: 1000,
  “l”: {...}
}

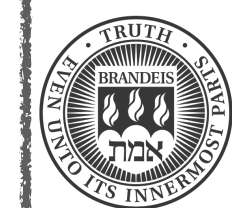
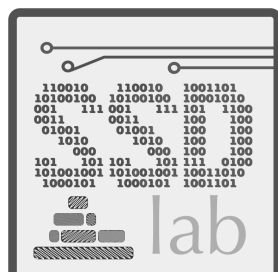
“inserts”: {
  “op_count”: 1M,
  “key”: {...},
  “val”: {
    “uniform”: { “len”: 64 }
  }
},

“point_queries”: {
  “op_count”: 90K
}

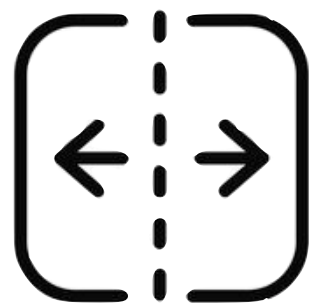
“point_queries”: {
  “op_count”: 1.1M,
  “selection”: {
    “beta”: {
      “alpha”: 0.7,
      “beta”: 2.1
    }
  }
}
}
```

Group 1

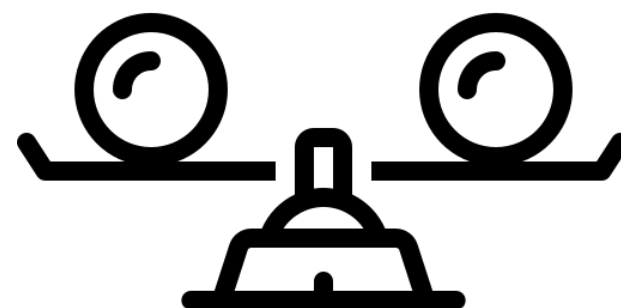
Group 2



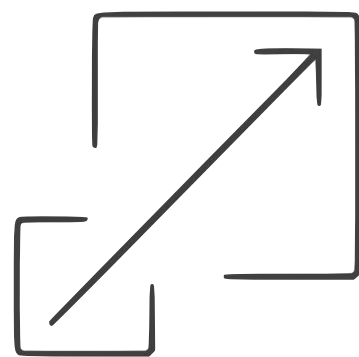
Experimental Setup



How **efficiently** Tectonic
mimic SoA
benchmarks?



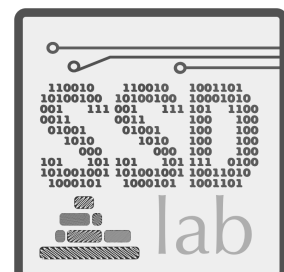
Can Tectonic generate
similar patterns
in the database?



How **scalable** is Tectonic
while offering better
performance?

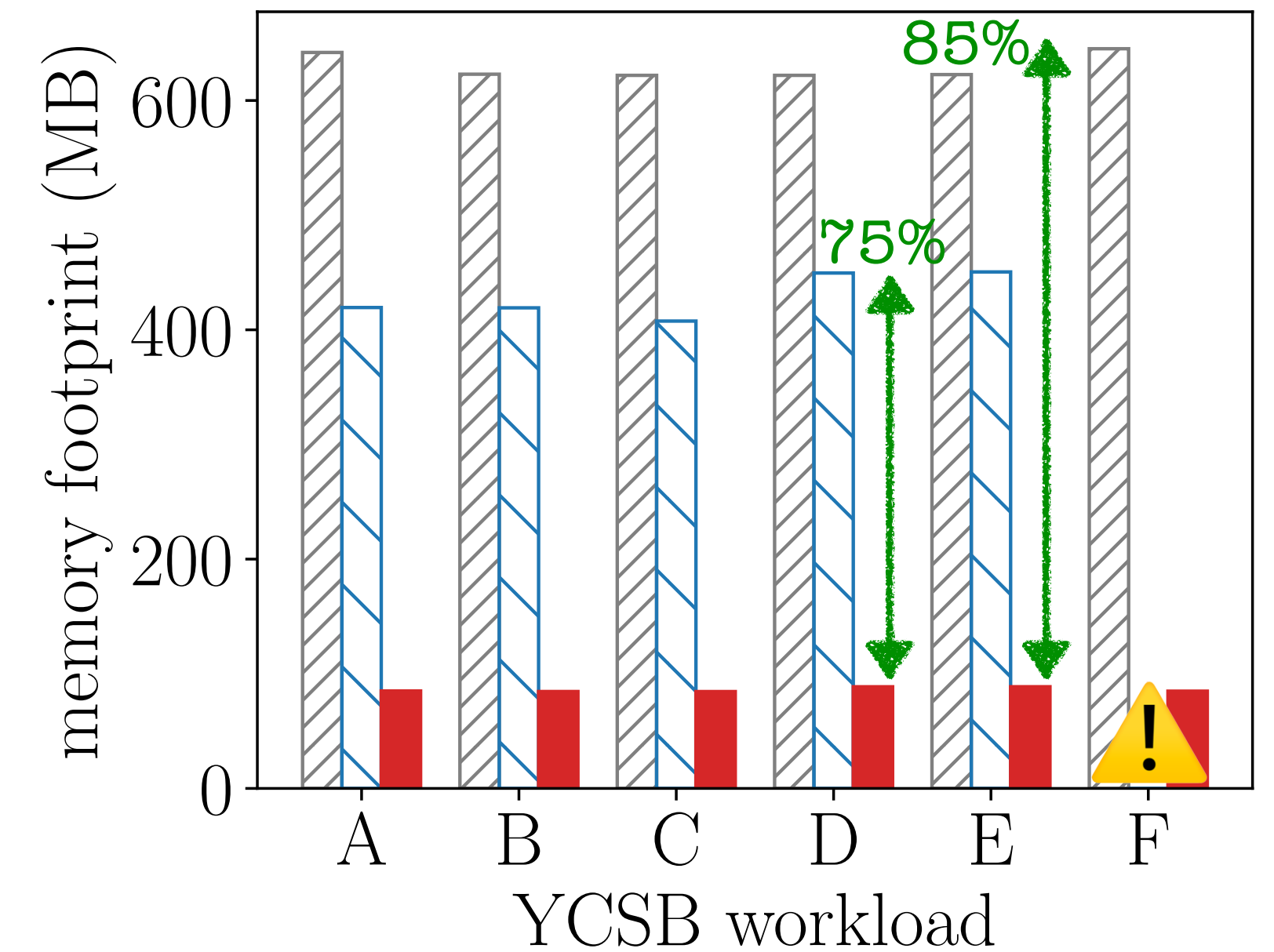
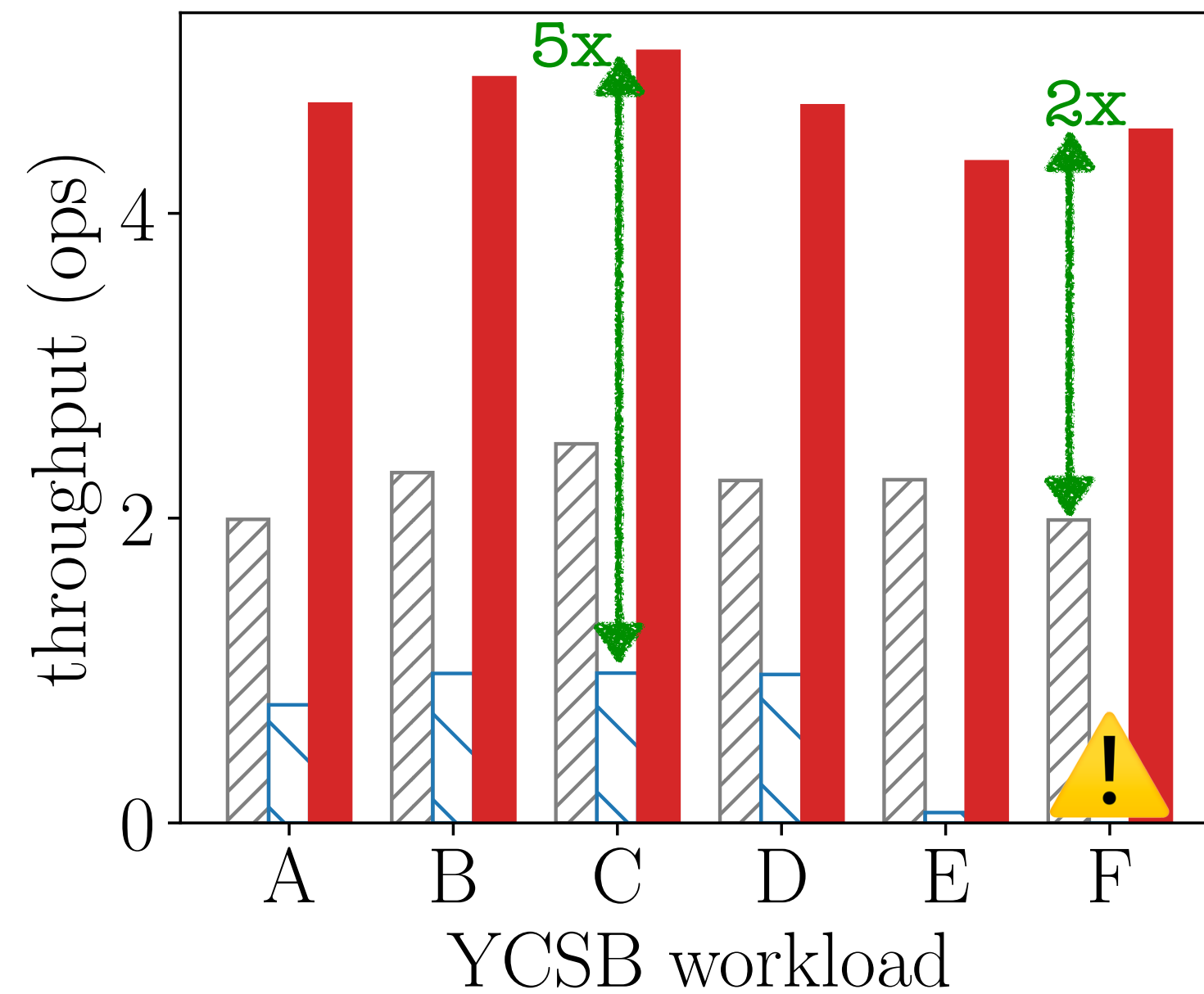
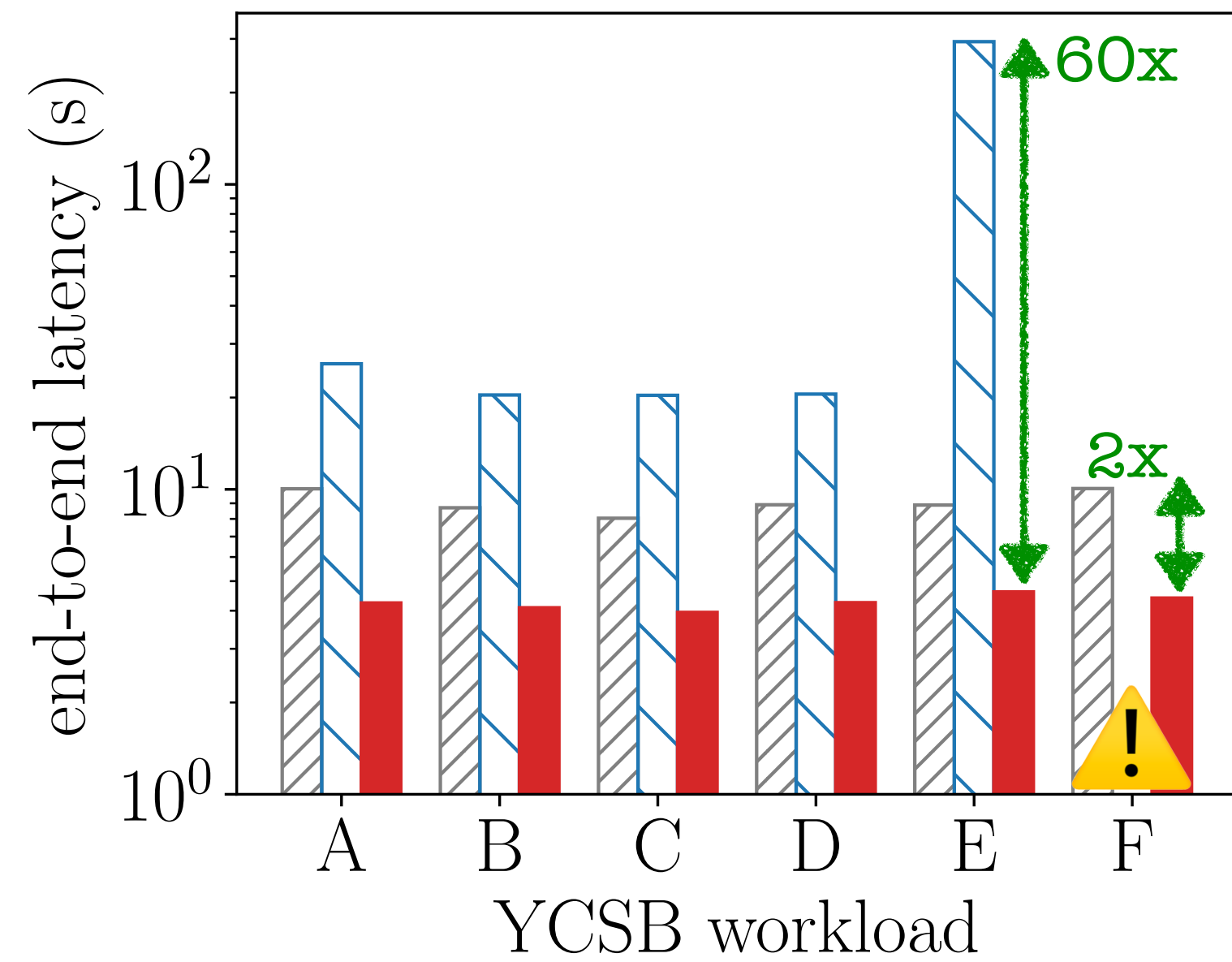


Intel(R) Xeon(R)
Gold 6240R CPU
2.40GHz cores
192GB RAM
240GB SSD



How efficiently Tectonic mimic SoA benchmarks?

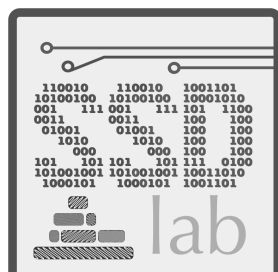
YCSB KV Bench Tectonic workload size: 1GB



Tectonic is up to **60x** faster than KV Bench and **2x** faster than YCSB

Tectonic is **5x** more efficient than KV Bench and **2x** than YCSB

Tectonic uses **75% to 85%** less memory than state-of-the-art



How scalable is Tectonic?

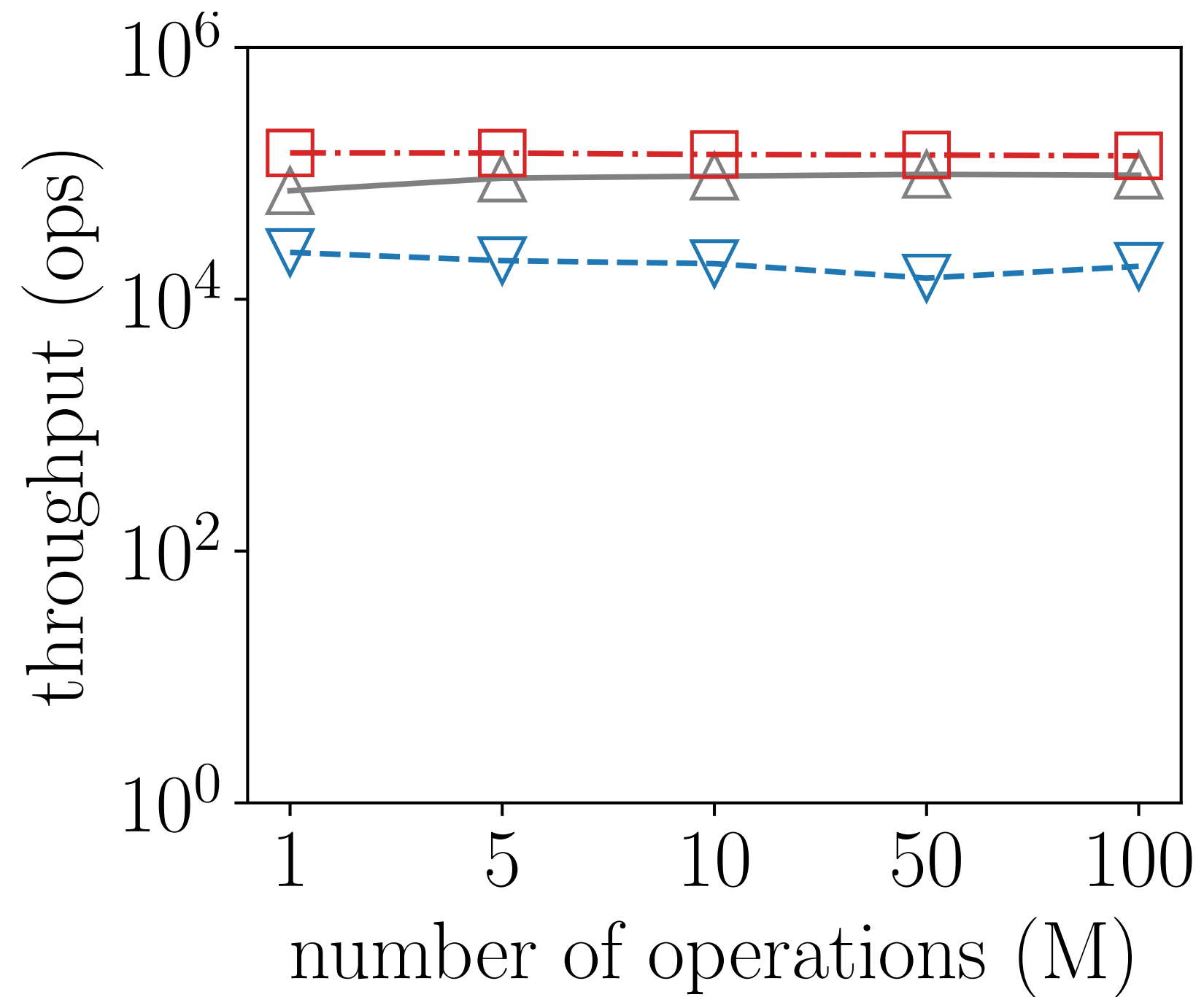
w/l: 50% inserts followed by interleaved 40% inserts, 5% update, and 5% point queries

△ YCSB

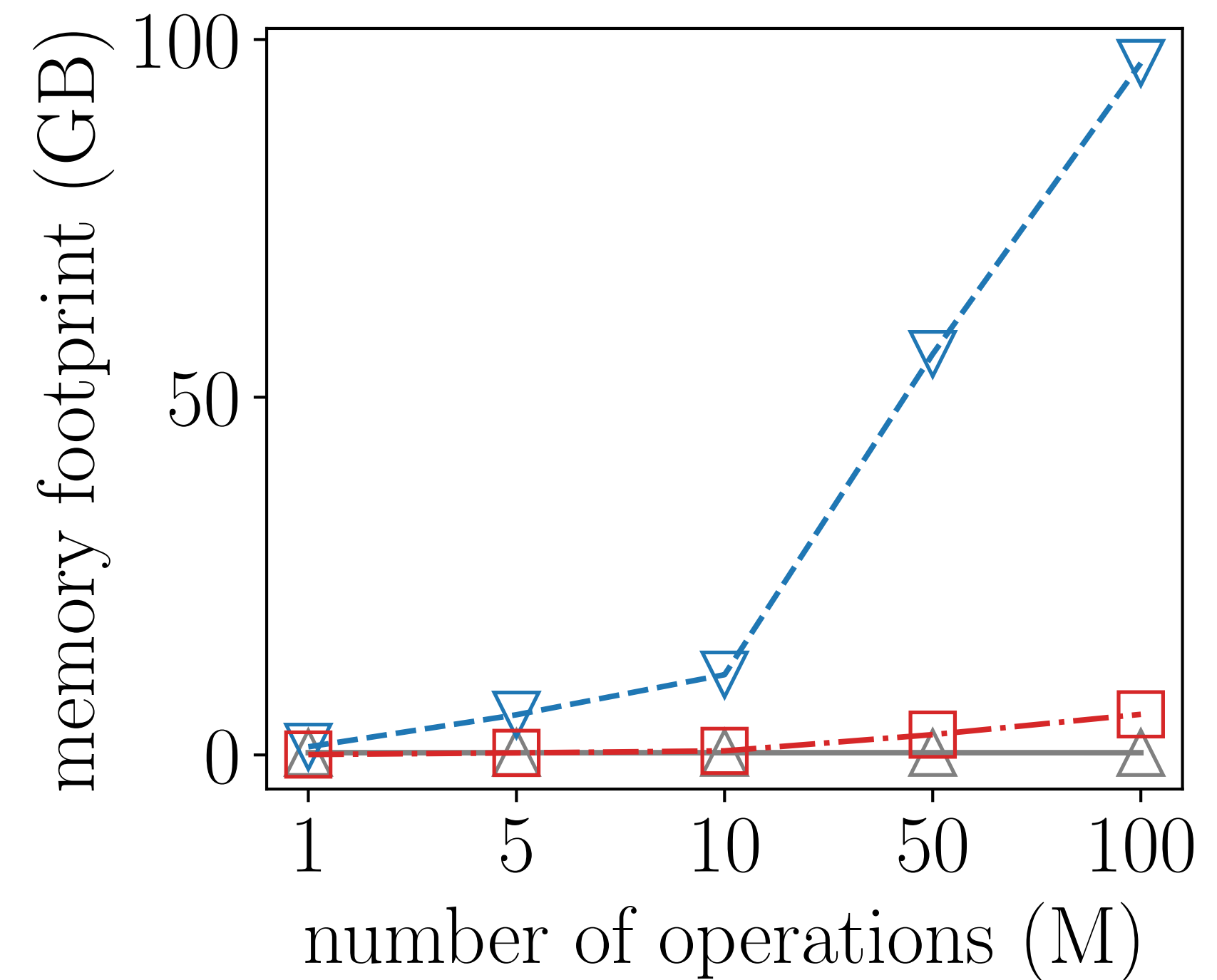
▽ KV Bench

□ Tectonic

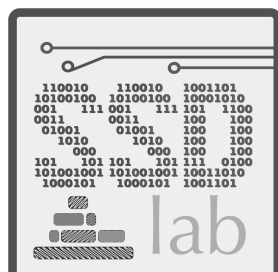
entry size: 1KB



Tectonic is **scalable** and offers **better throughput** with an increasing number of operations



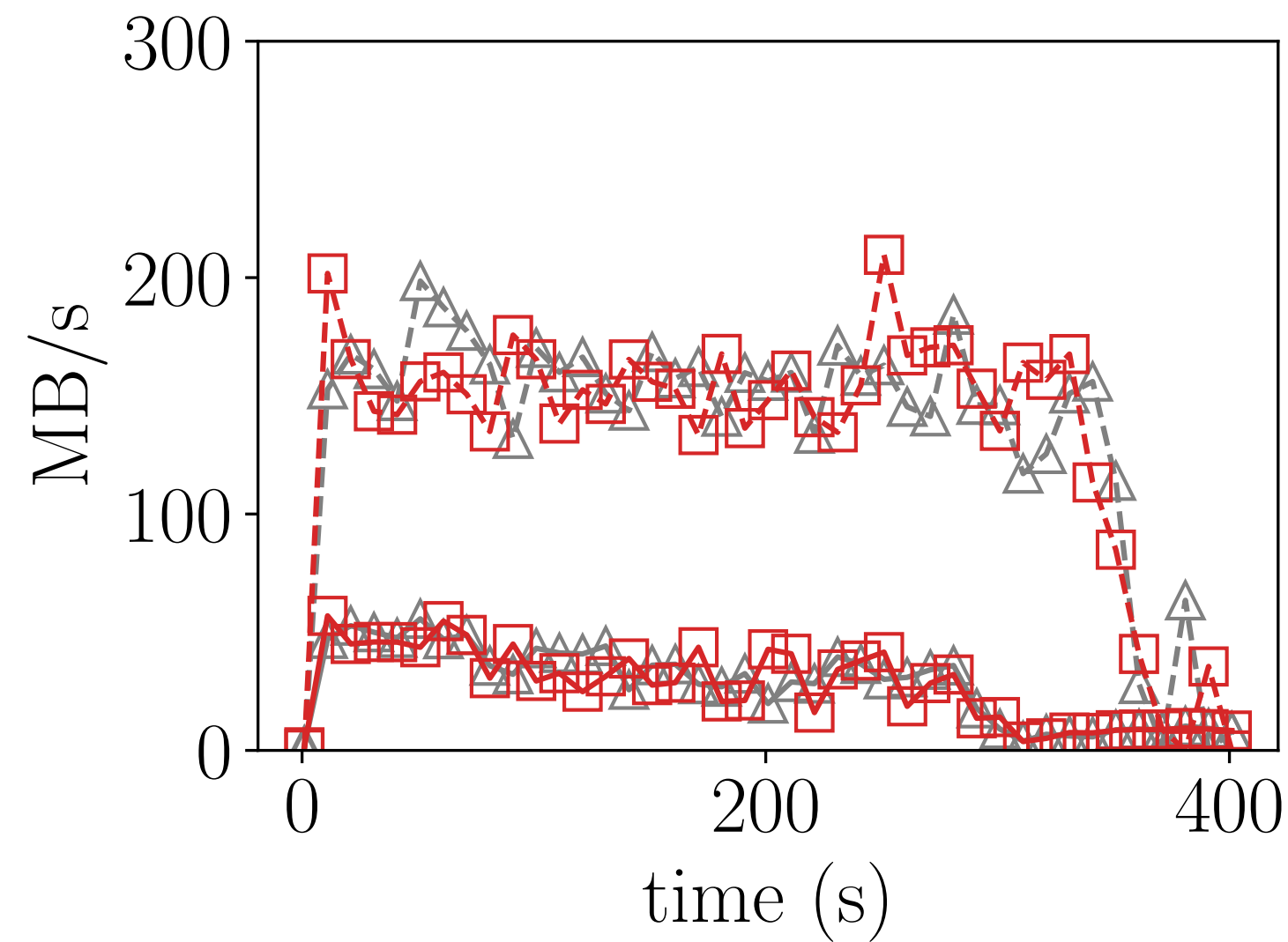
Tectonic offers **better memory usage**, unlike KV Bench, with an increasing number of operations



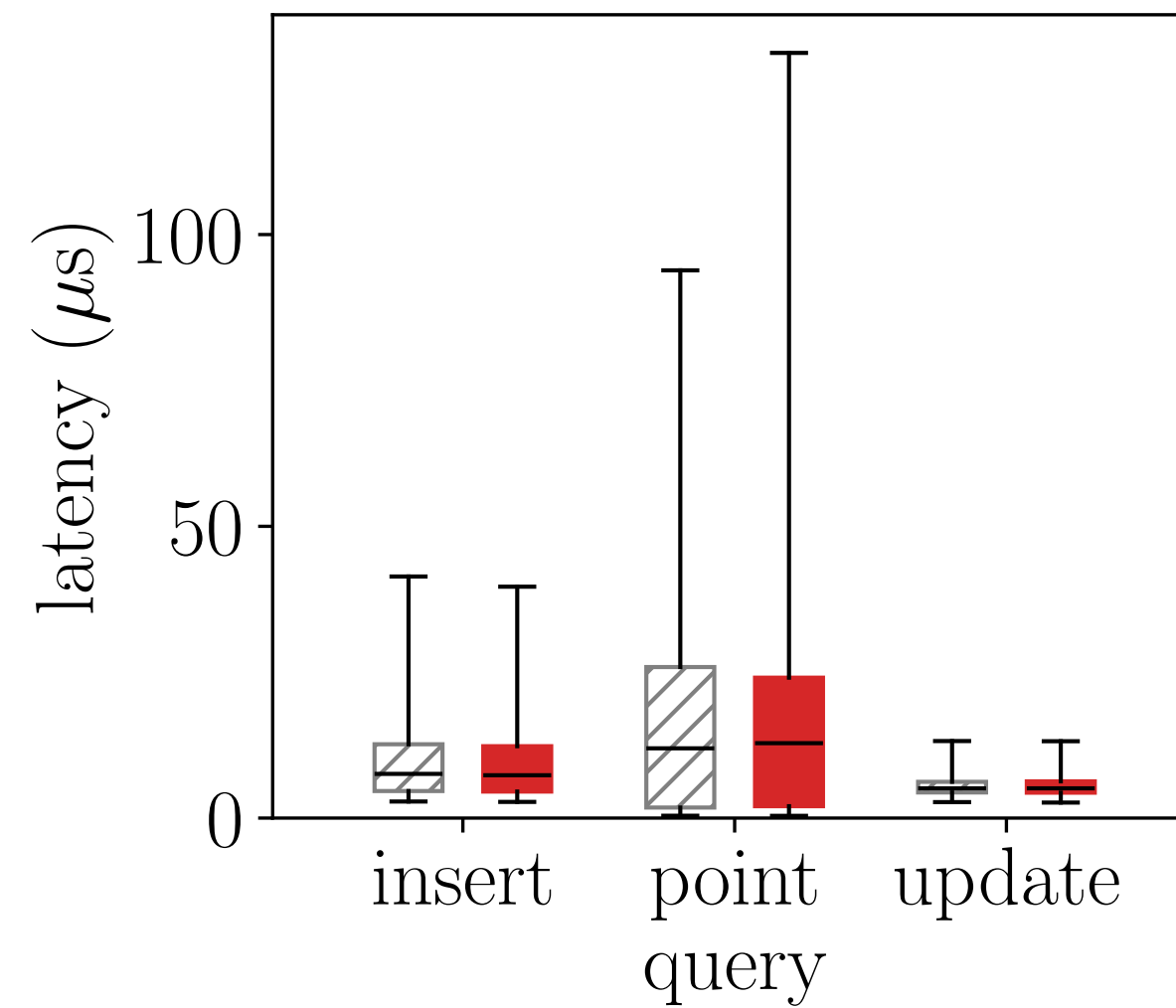
Can Tectonic generate similar patterns?

workload: preload DB with 10M inserts, then runs 5M updates interleaved with 5M point queries

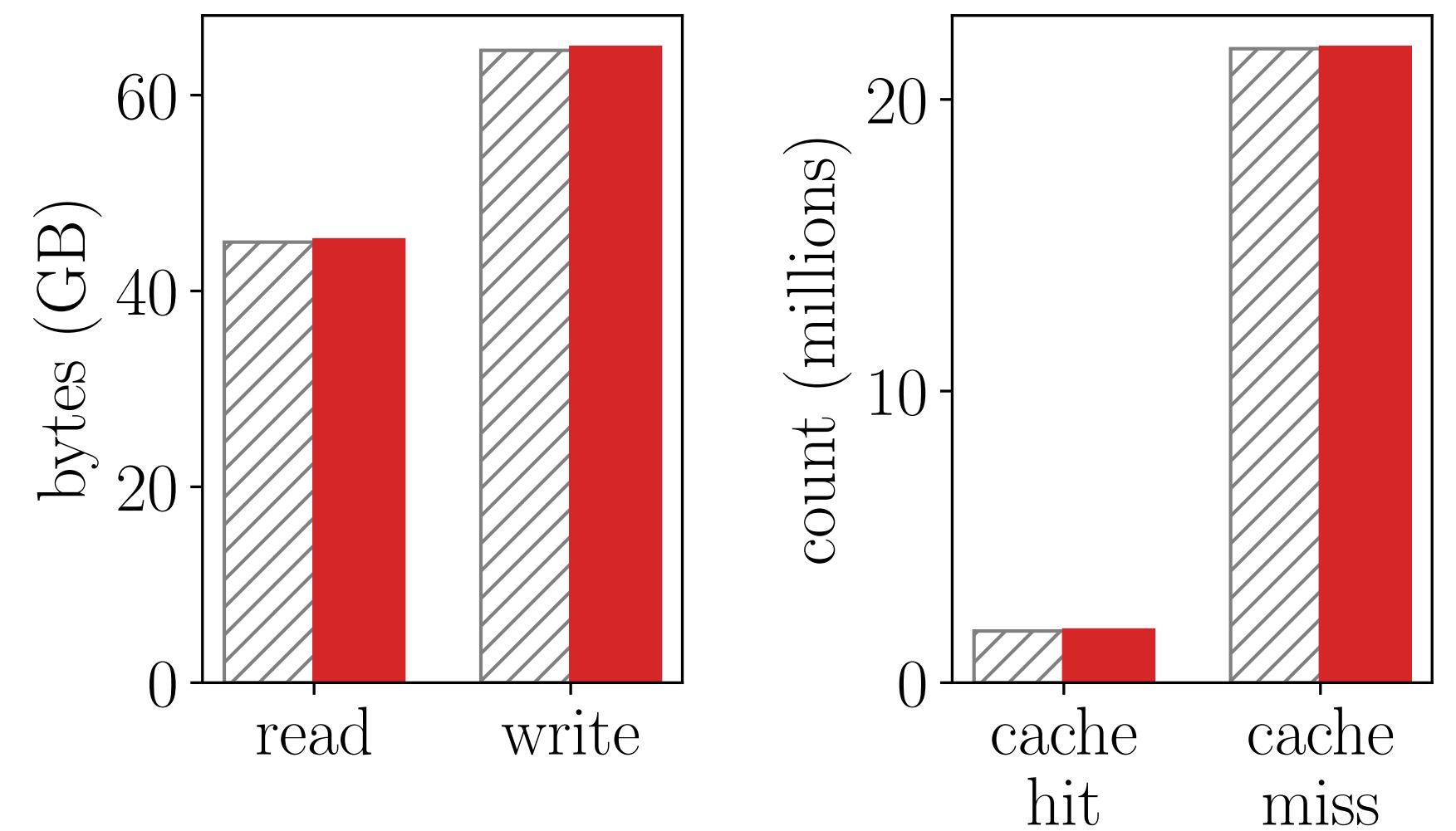
—△— read (YCSB) --△-- write (YCSB) —□— read (Tectonic) --□-- write (Tectonic) ▨ YCSB ■ Tectonic



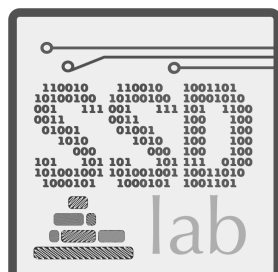
the read and write patterns are similar



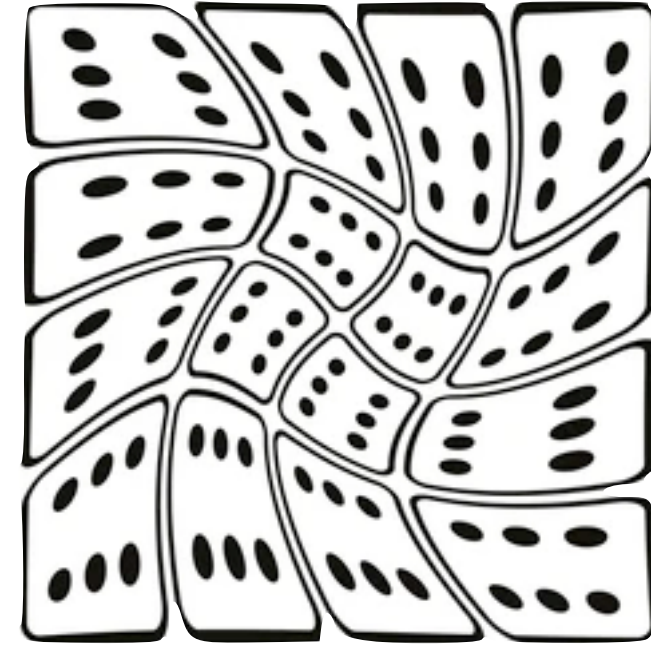
different operations' latencies are the same



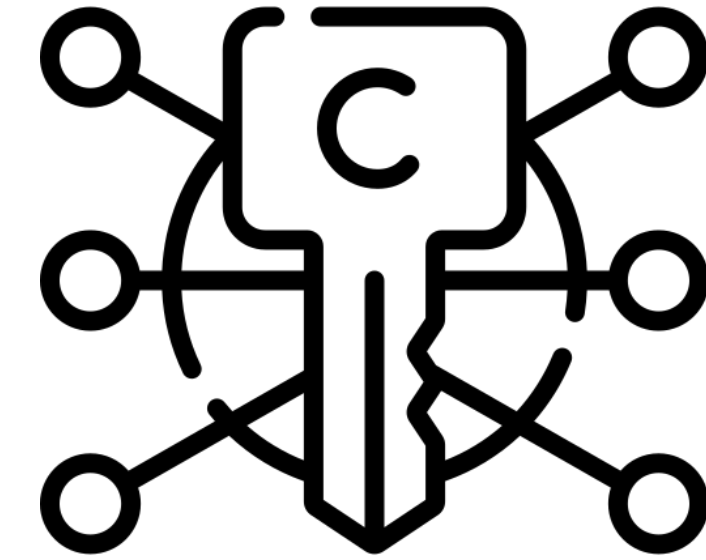
same amount of bytes read and written; cache hits and misses are similar



Conclusion



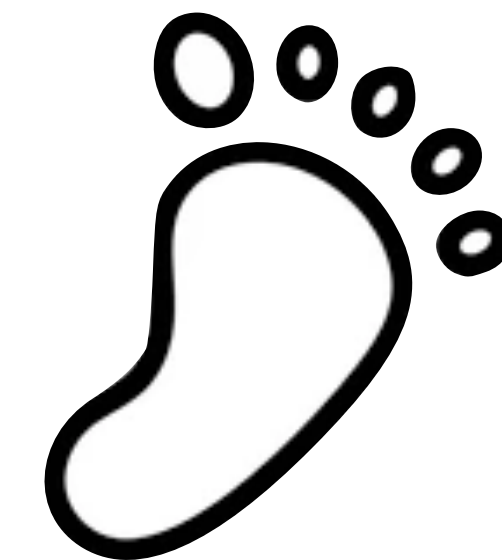
supports **realistic workloads**, including **shifting and dynamic patterns**



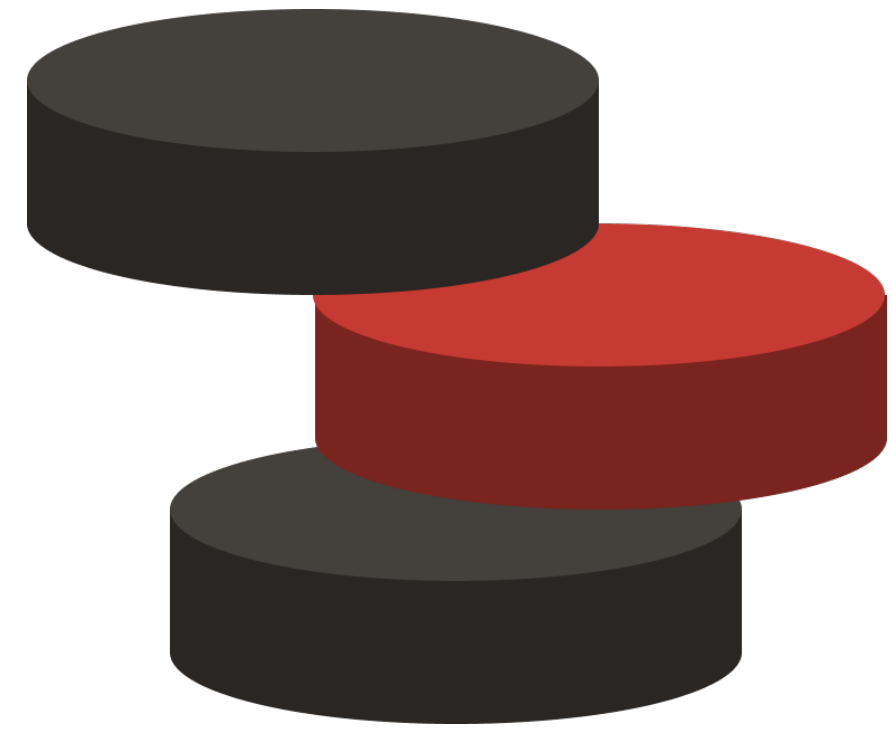
composite key generation; several **operations** with **different distributions**



capable of generating **fully** as well as **nearly-sorted workloads**



offers up to **2x better throughput** than SoA; up to **84 % lower memory footprint**



Tectonic: Bridging Synthetic and Real-World Workloads for Key-Value Benchmarking

Thank You!



Alexander H. Ott
<https://0xott.zip>



Shubham Kaushik
<https://shubhamkaushik.com>

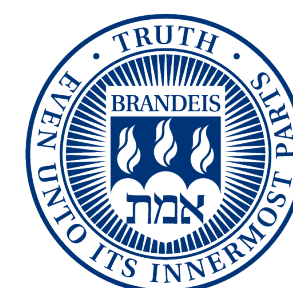
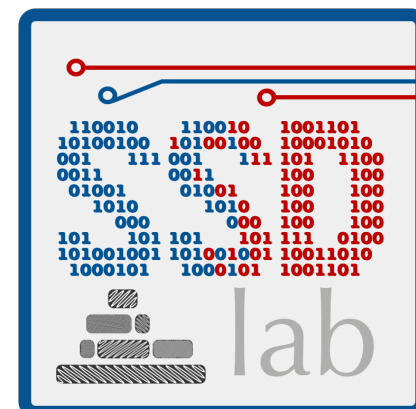
Questions?



James Chen
<https://boao-james-chen.github.io/>



Subhadeep Sarkar
<https://subhadeep.net/>



Brandeis
UNIVERSITY

