

Anatomy of LSM Memory Buffer: Insights & Implications

Shubham Kaushik

Subhadeep Sarkar



Brandeis
UNIVERSITY



LSM-tree



Log Structured Merge-tree



Log Structured Merge-tree

The Log-Structured Merge-Tree (LSM-Tree)

1996

Patrick O'Neil¹, Edward Cheng²
Dieter Gawlick³, Elizabeth O'Neil¹
To be published: Acta Informatica



1996

LSM-tree
O'Neil et al.



Brandeis
UNIVERSITY

1996

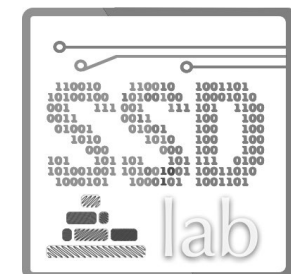
LSM-tree

O'Neil et al.

2006



BigTable



Brandeis
UNIVERSITY

1996

LSM-tree

O'Neil et al.

2006



BigTable

2007



APACHE
HBASE



Brandeis
UNIVERSITY

1996

LSM-tree

O'Neil et al.

2006

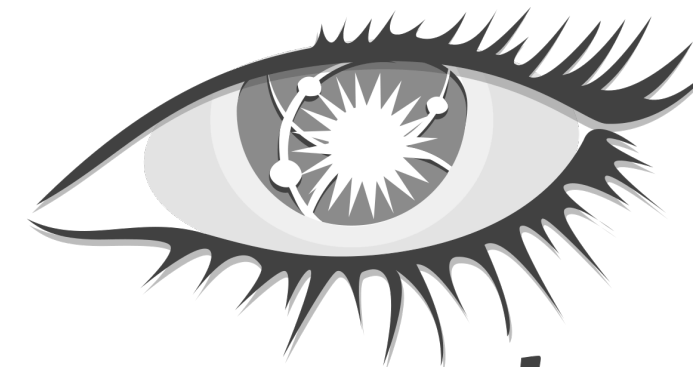


BigTable

2007



2010



cassandra



Brandeis
UNIVERSITY

1996

LSM-tree

O'Neil et al.

2006

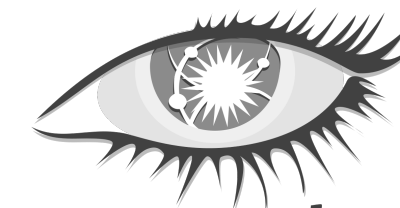


BigTable

2007



2010



cassandra

2011



levelDB



Brandeis
UNIVERSITY

1996

LSM-tree
O'Neil et al.

2006


BigTable

2007


APACHE
HBASE

2010


cassandra

2011


levelDB

2013

 **RocksDB**



1996

LSM-tree

O'Neil et al.

2006

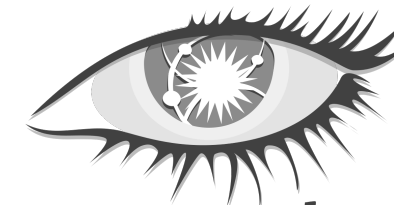


BigTable

2007



2010



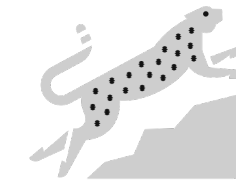
cassandra

2011



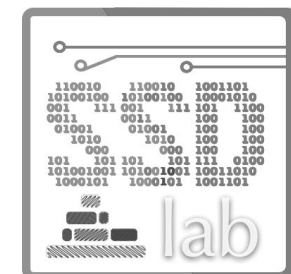
levelDB

2013



RocksDB

2024



Brandeis
UNIVERSITY

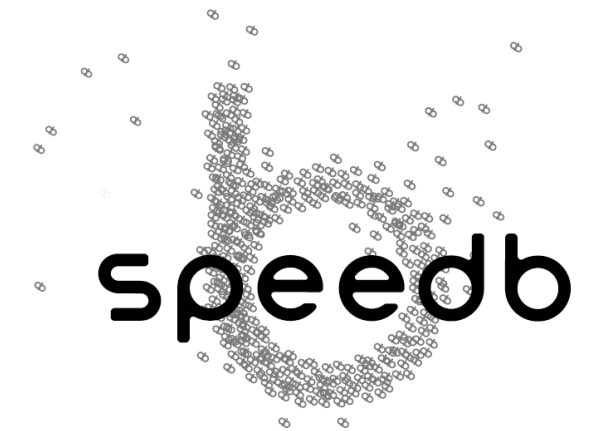


BigTable



mongoDB

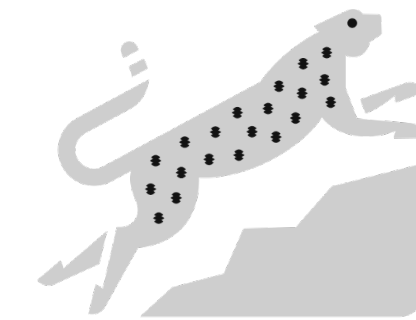
WT



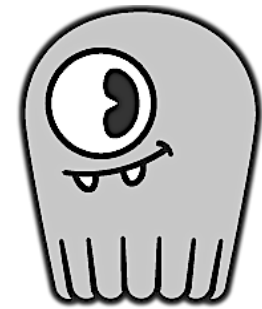
speedb



LSM-tree



RocksDB



SCYLLA



DynamoDB



CockroachDB



influxdb

2024

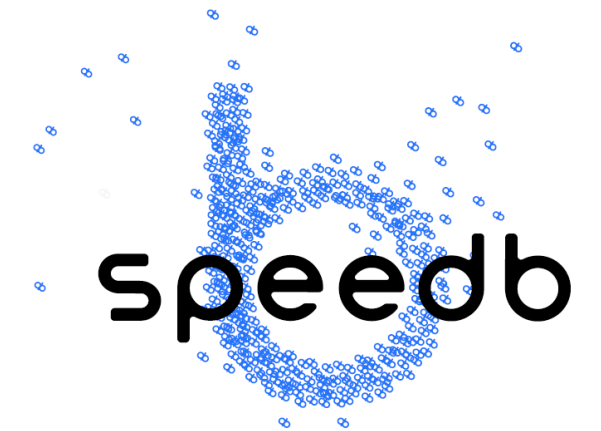




BigTable



mongoDB



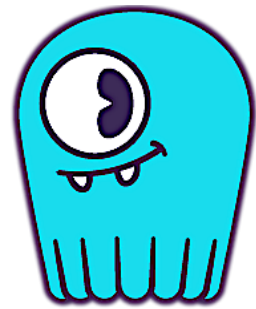
speedb



LSM-tree



RocksDB



SCYLLA



DynamoDB



CockroachDB



influxdb

2024



Brandeis
UNIVERSITY

Why LSM ?



fast writes

Why LSM ?



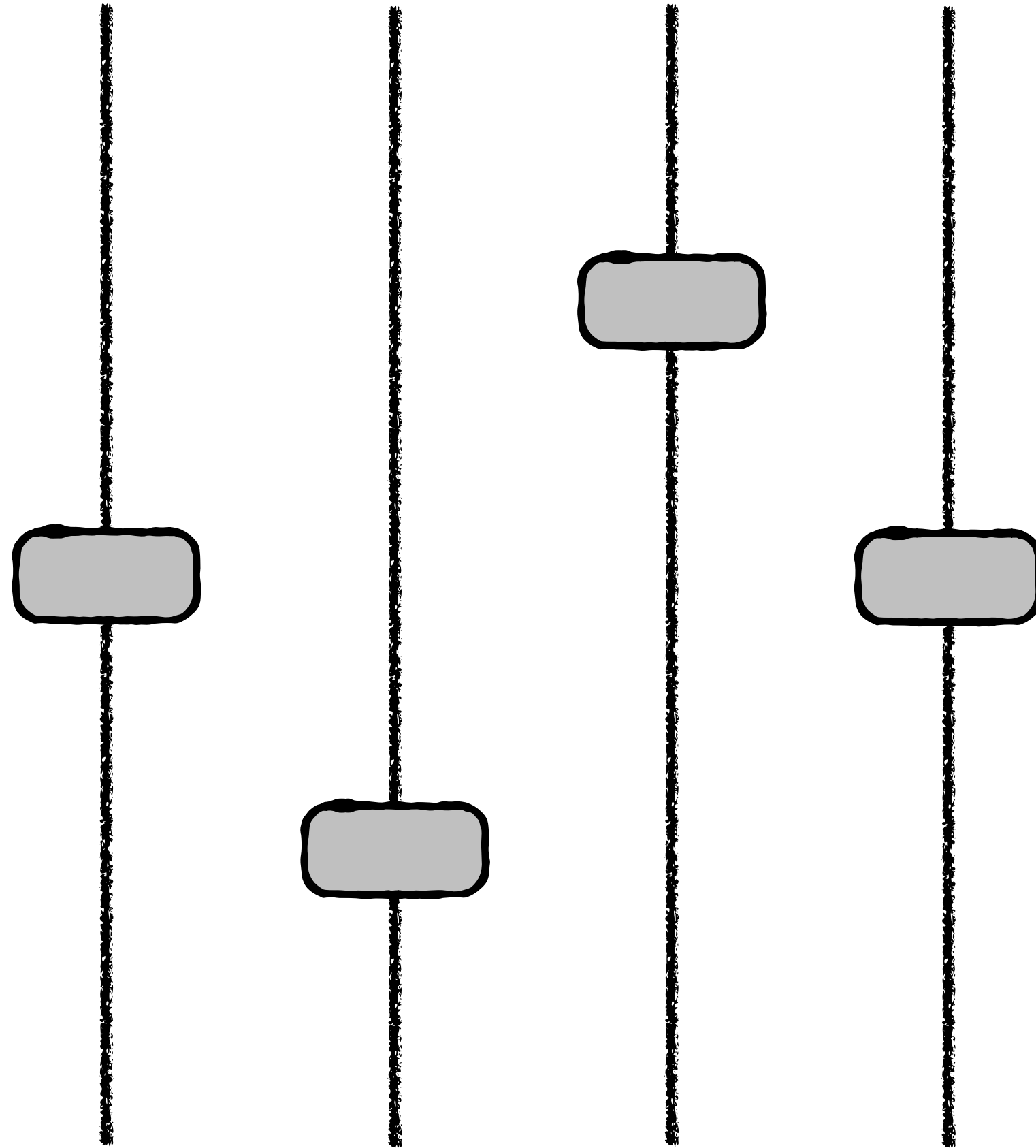
fast writes



competitive reads

Why LSM ?

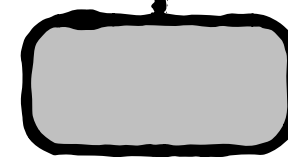
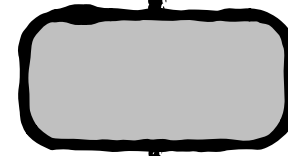
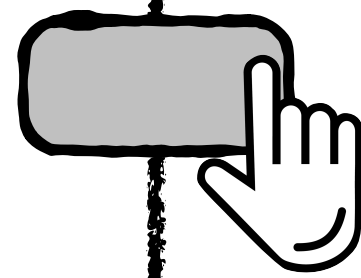
fast writes



competitive reads

Why LSM ?

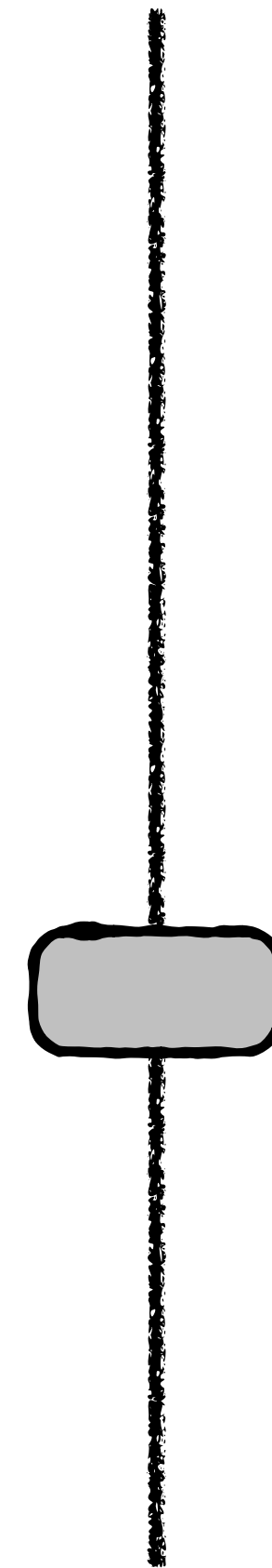
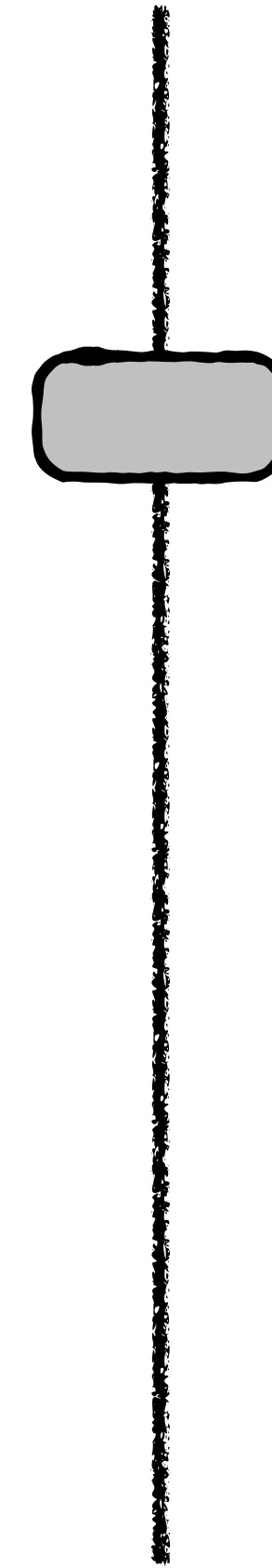
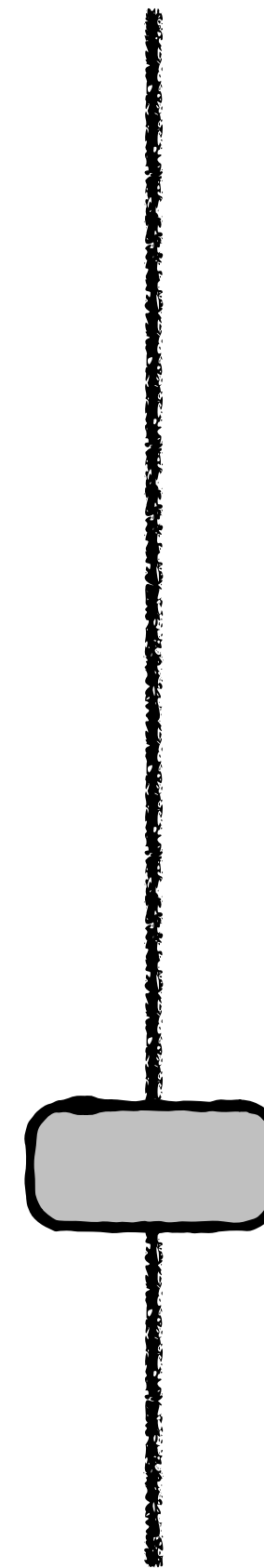
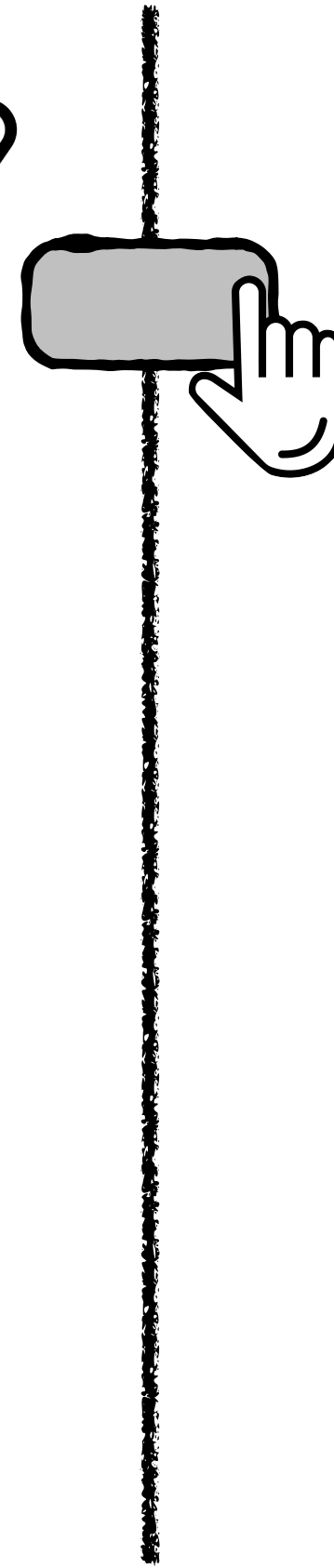
fast **writes**



competitive
reads

Why LSM ?

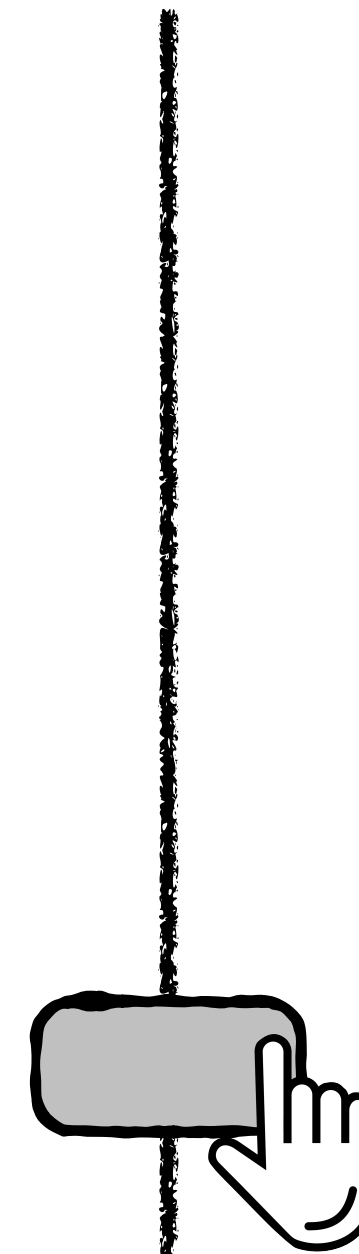
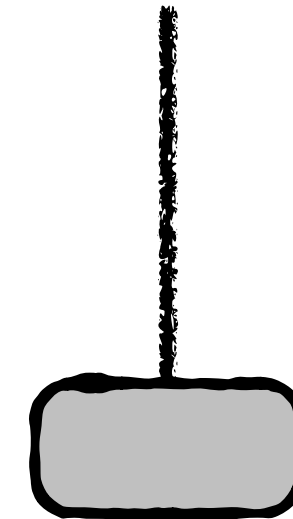
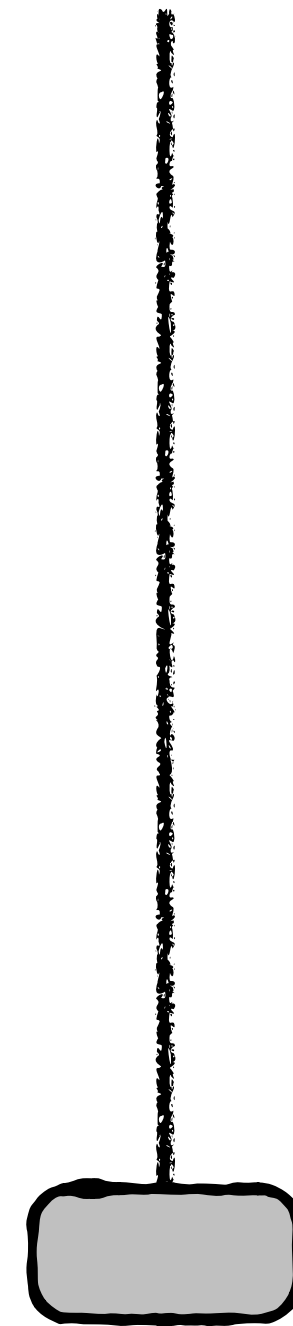
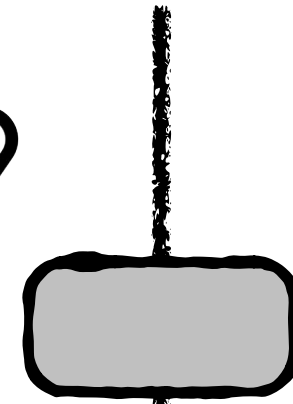
fast writes



competitive reads

Why LSM ?

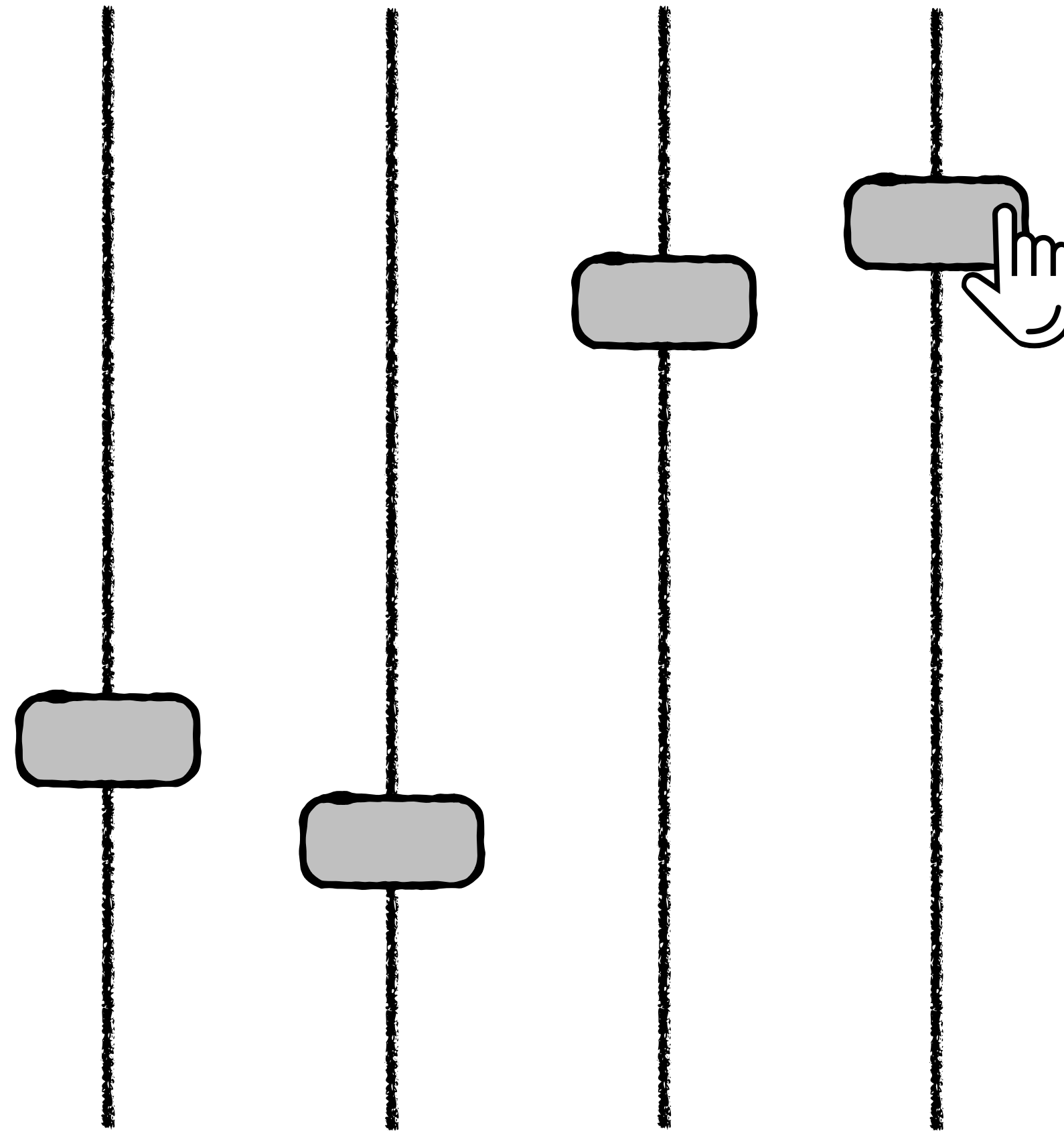
fast writes



competitive reads

Why LSM ?

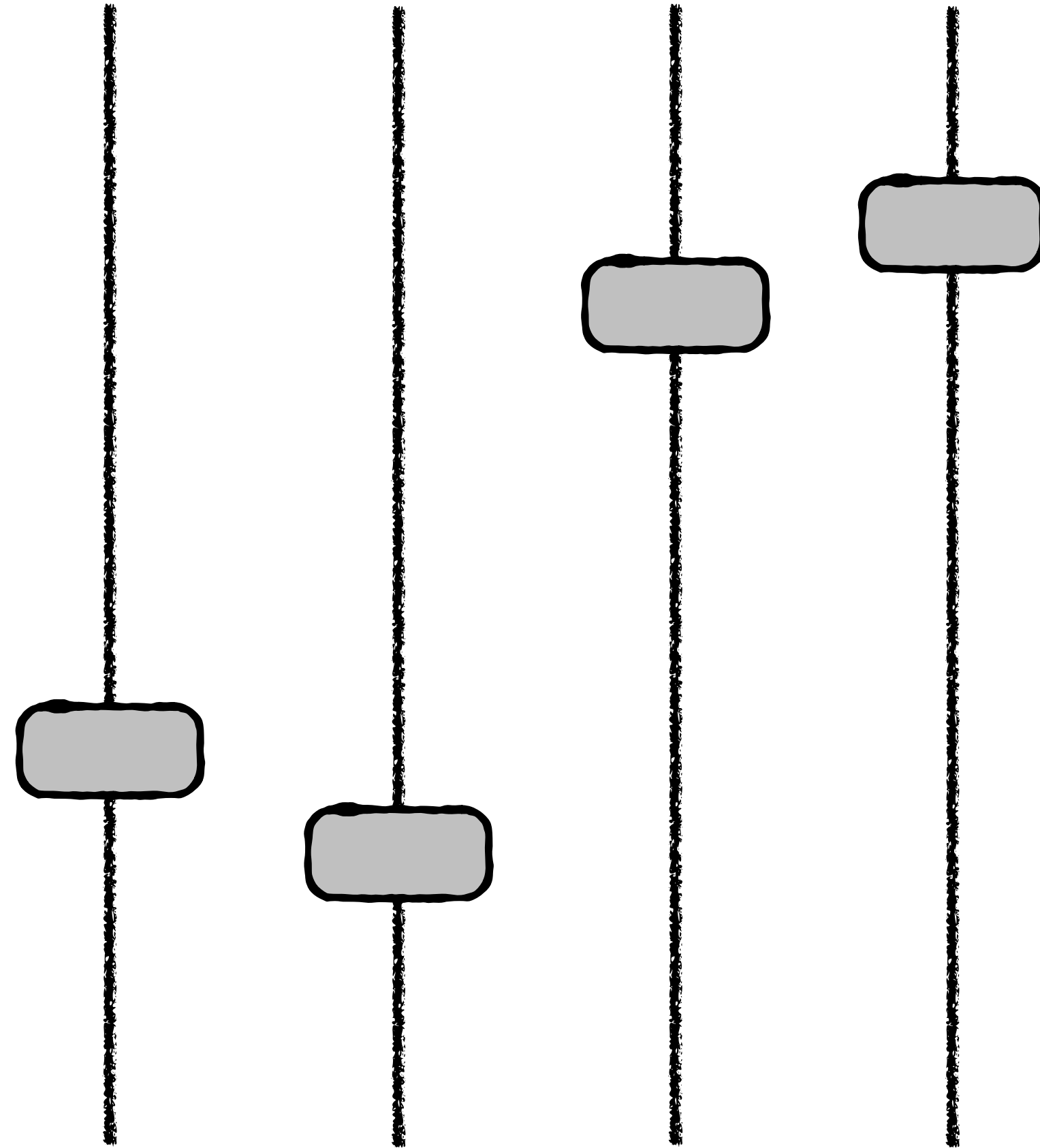
fast writes



competitive reads

Why LSM ?

fast writes

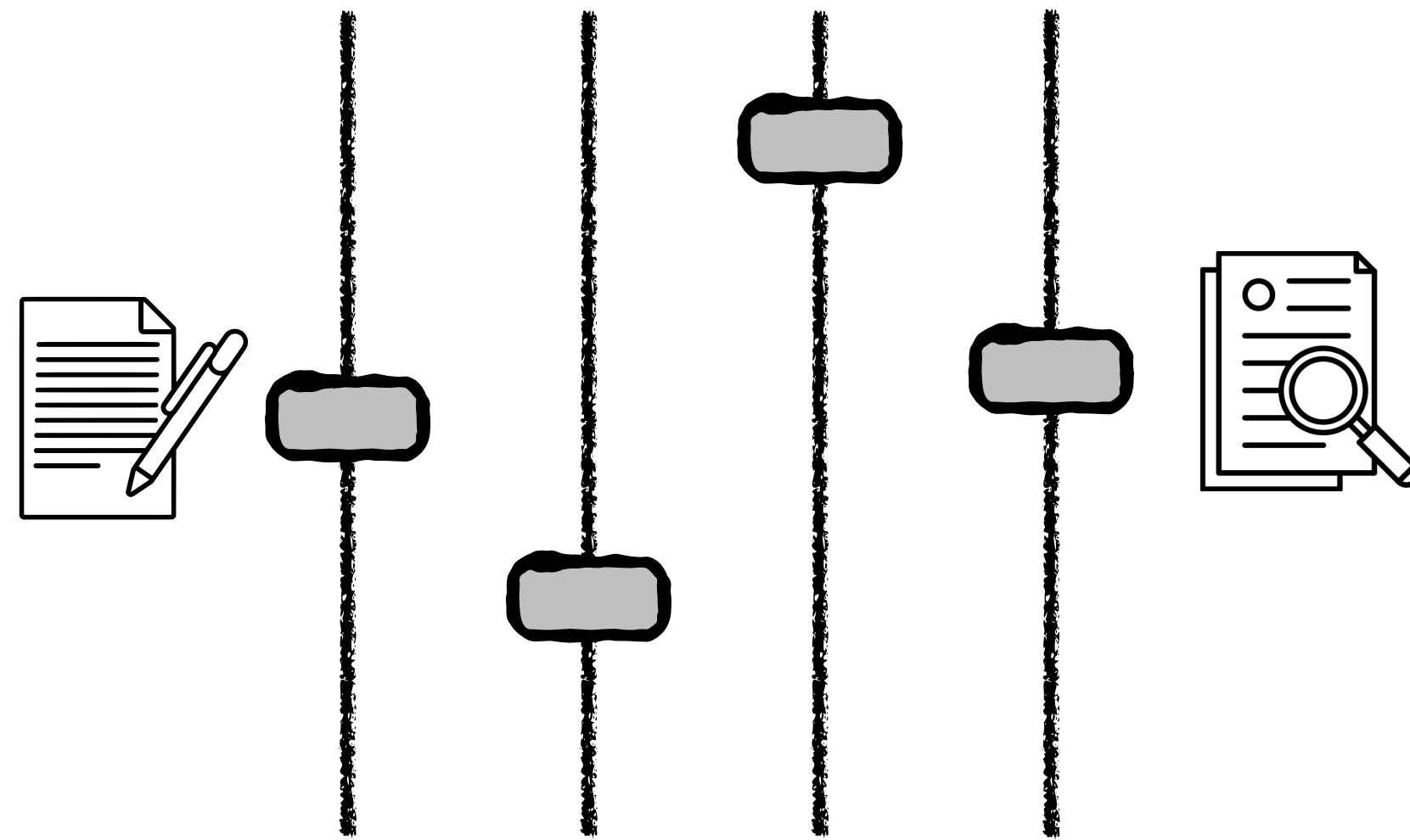


competitive reads

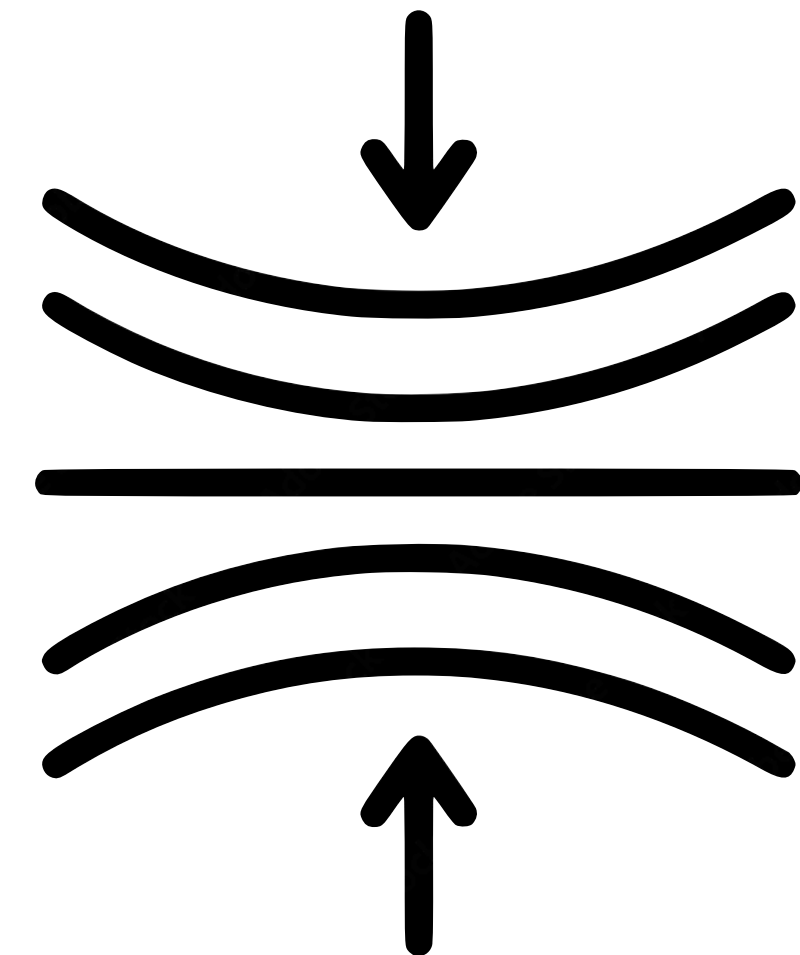
Why LSM ?



fast **writes**



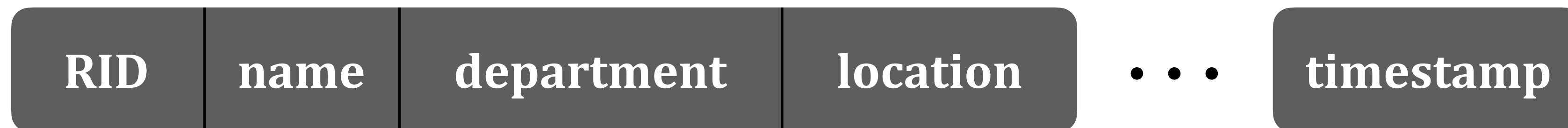
tunable read-write
performance



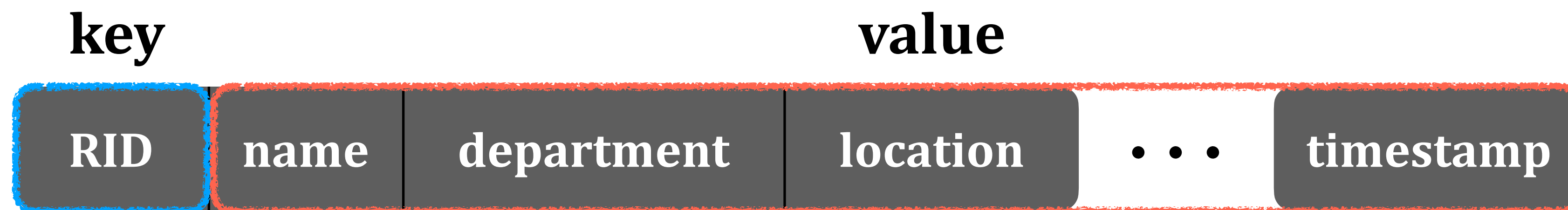
good **space**
utilization

LSM Basics

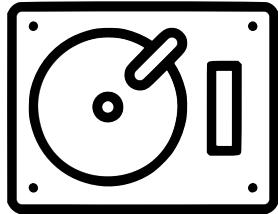
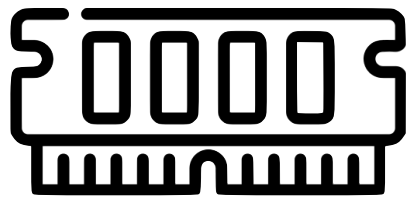
key-value pairs



LSM Basics



LSM Basics



buffer



L1



L2



size ratio: T

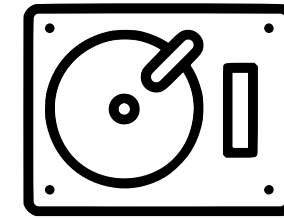
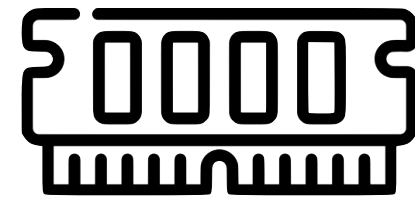
L3



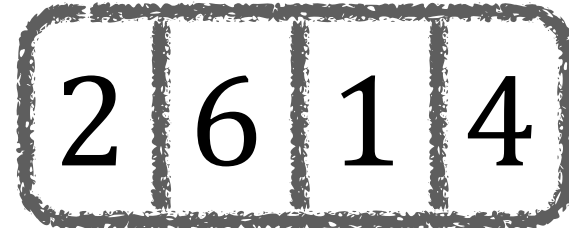
L4



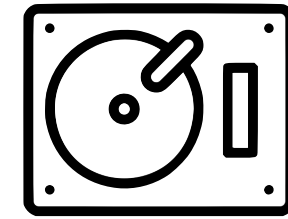
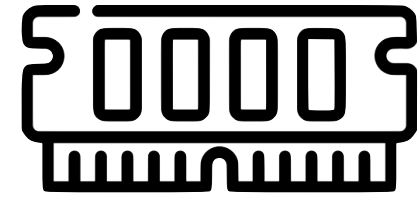
LSM Ingestion



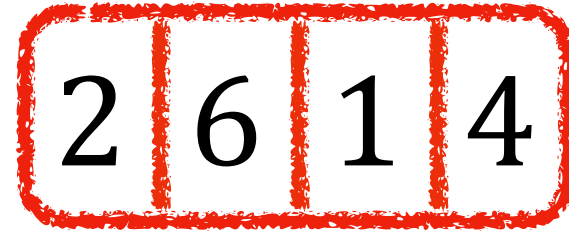
buffer



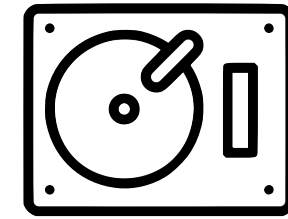
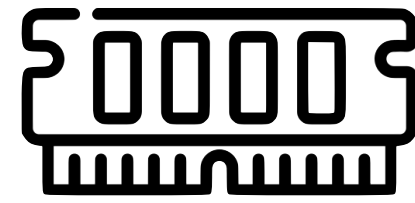
LSM Ingestion



buffer



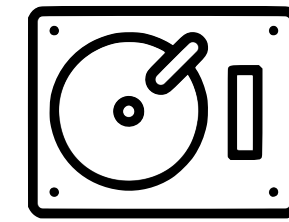
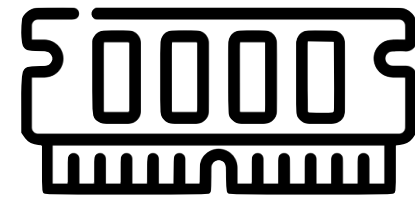
LSM Ingestion



buffer



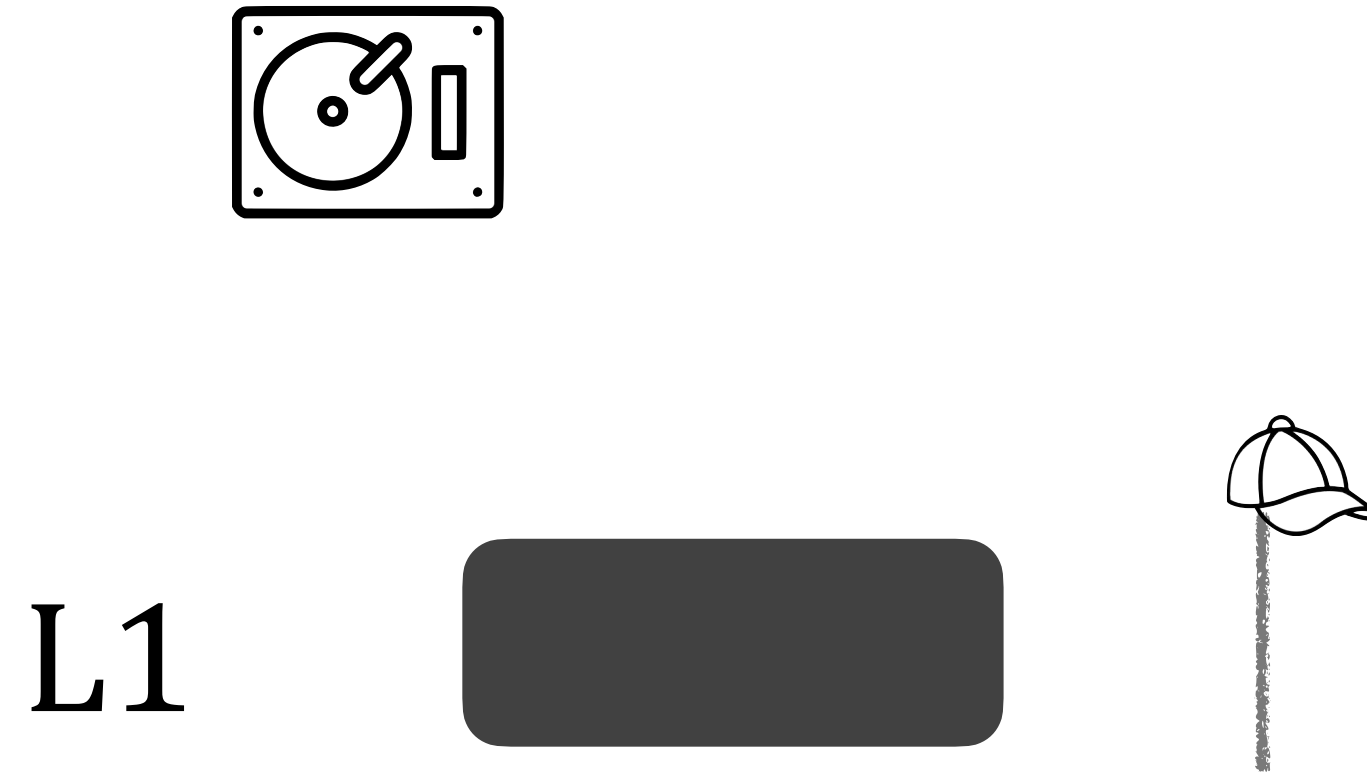
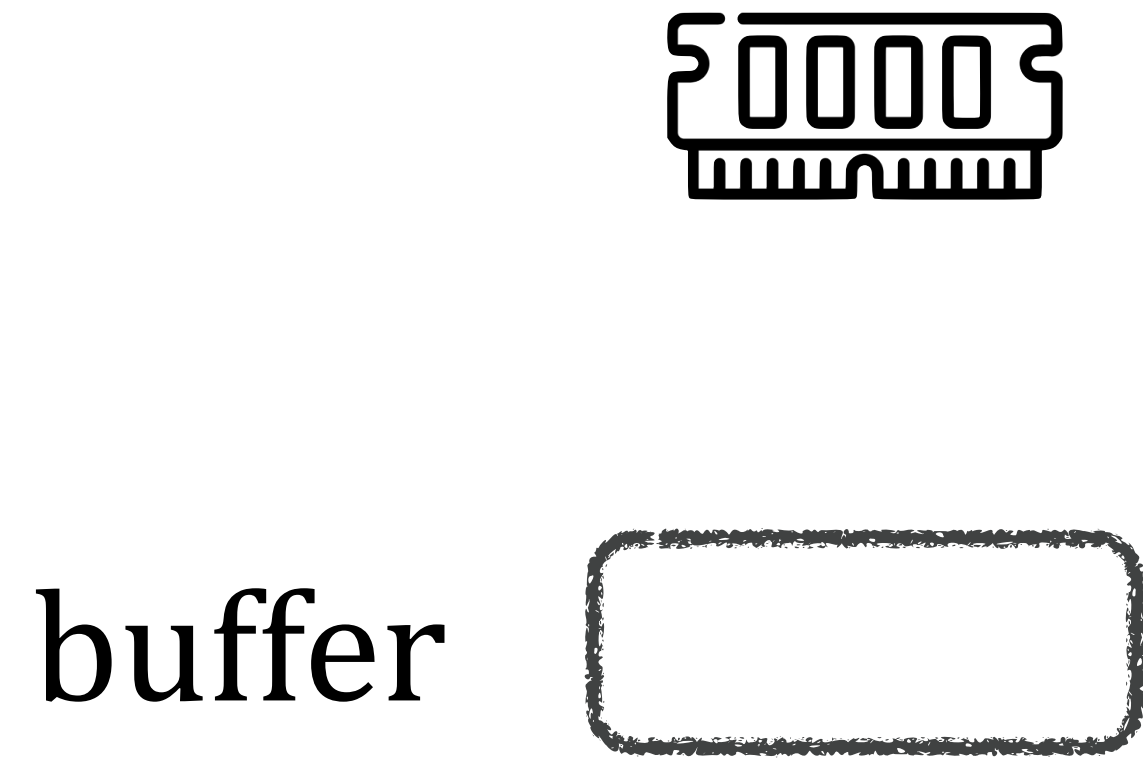
LSM Ingestion



buffer

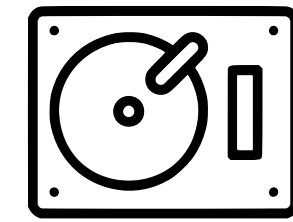


LSM Ingestion

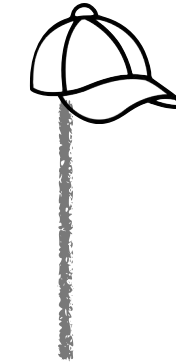


flush

LSM Ingestion

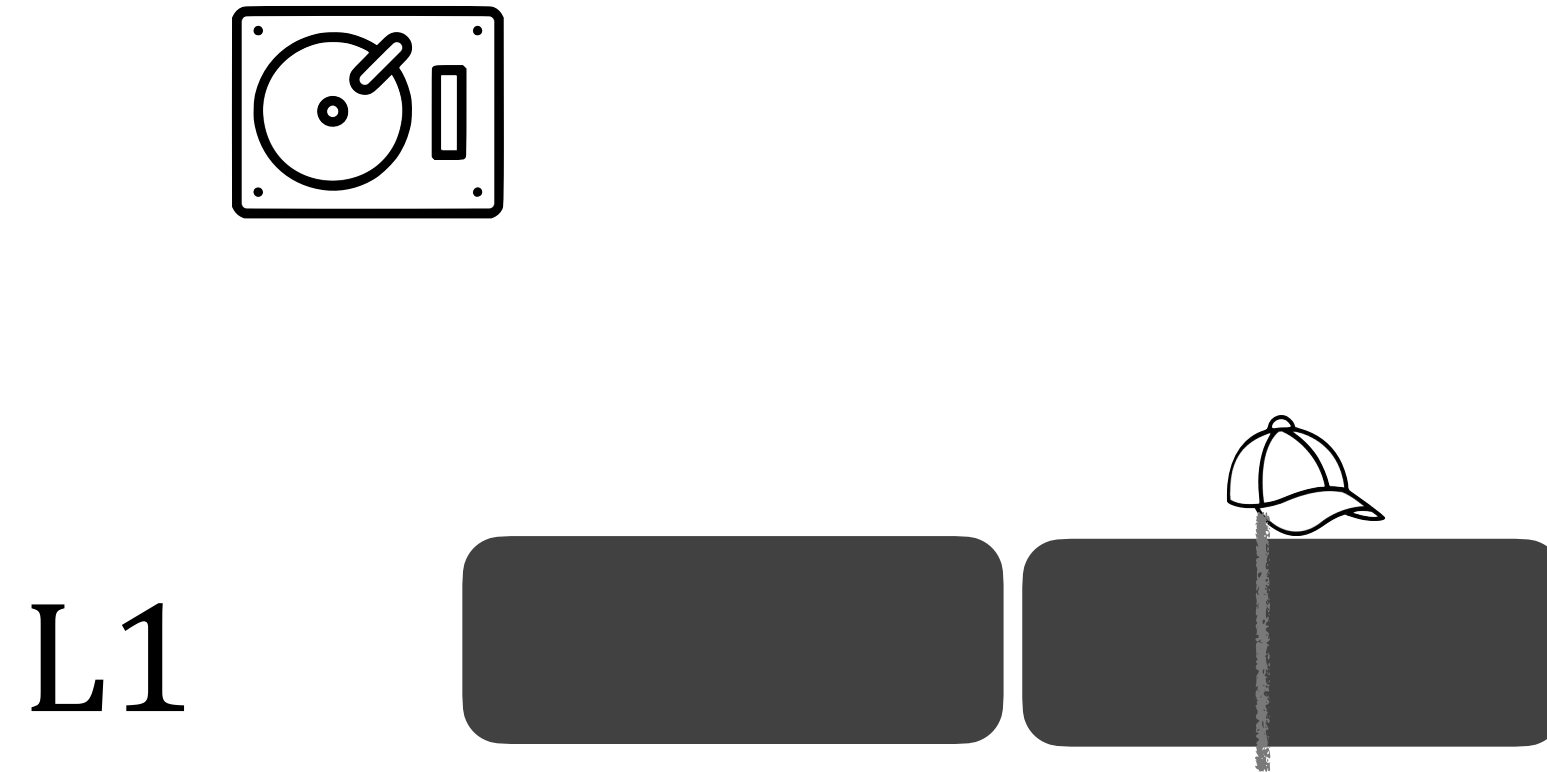
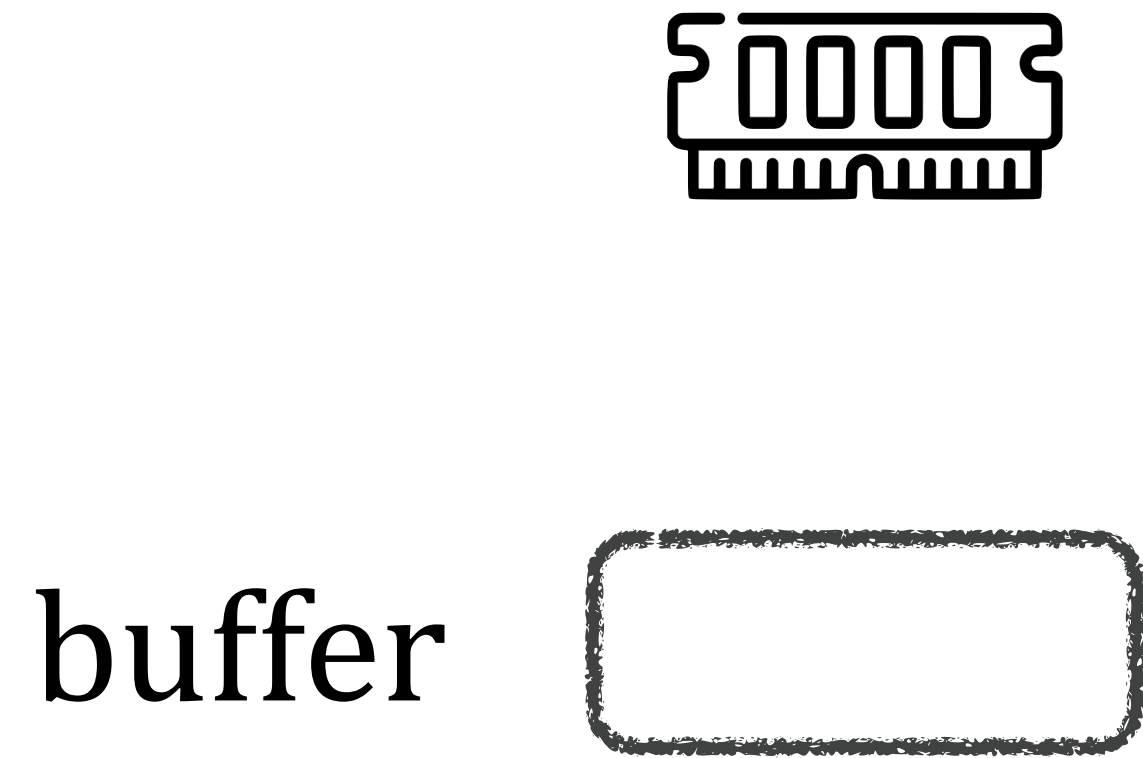


L1



flush

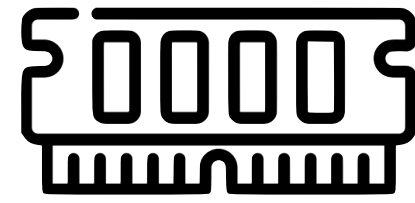
LSM Ingestion



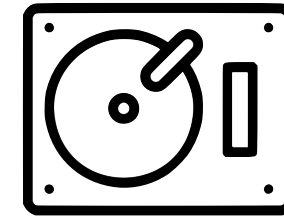
flush

LSM Ingestion

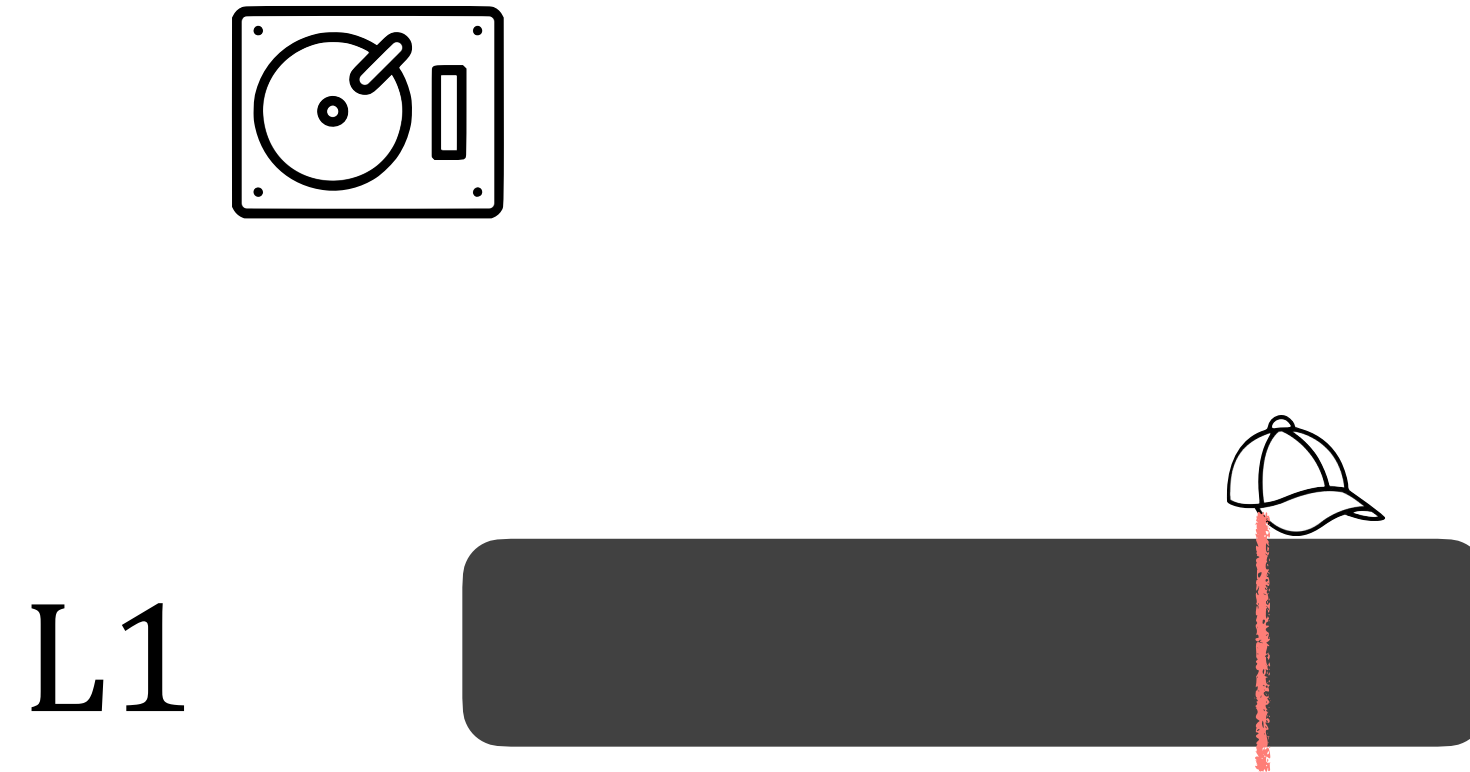
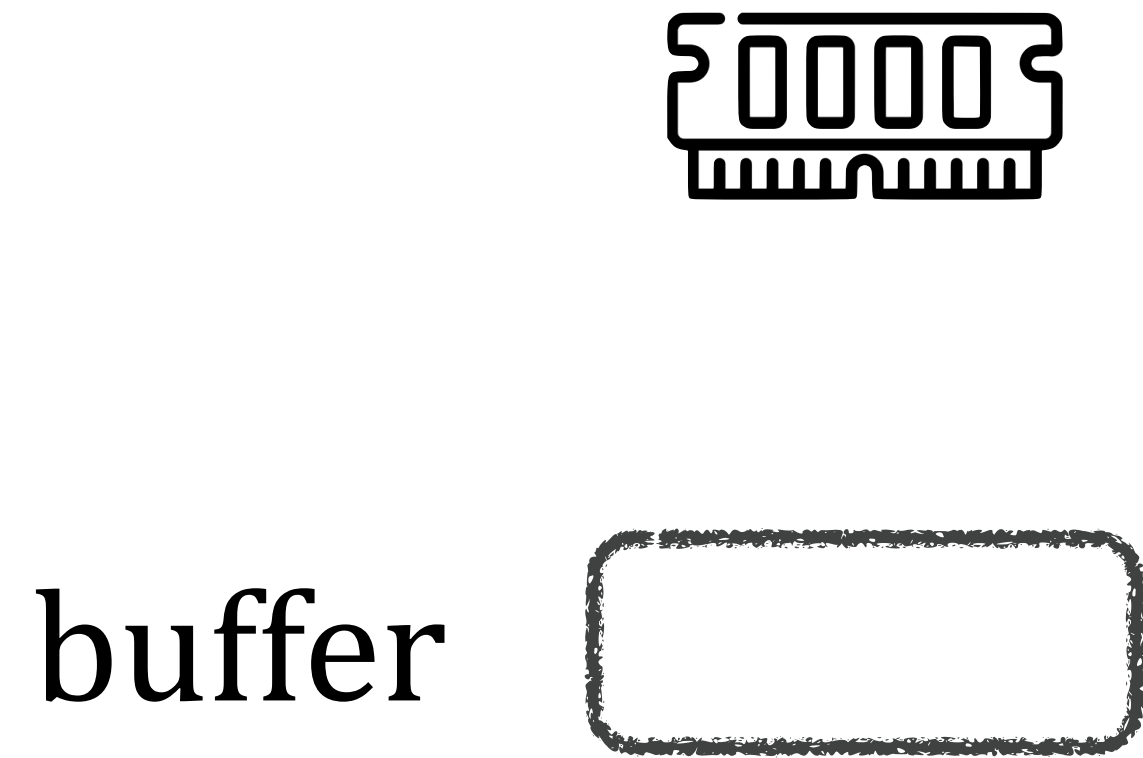
buffer



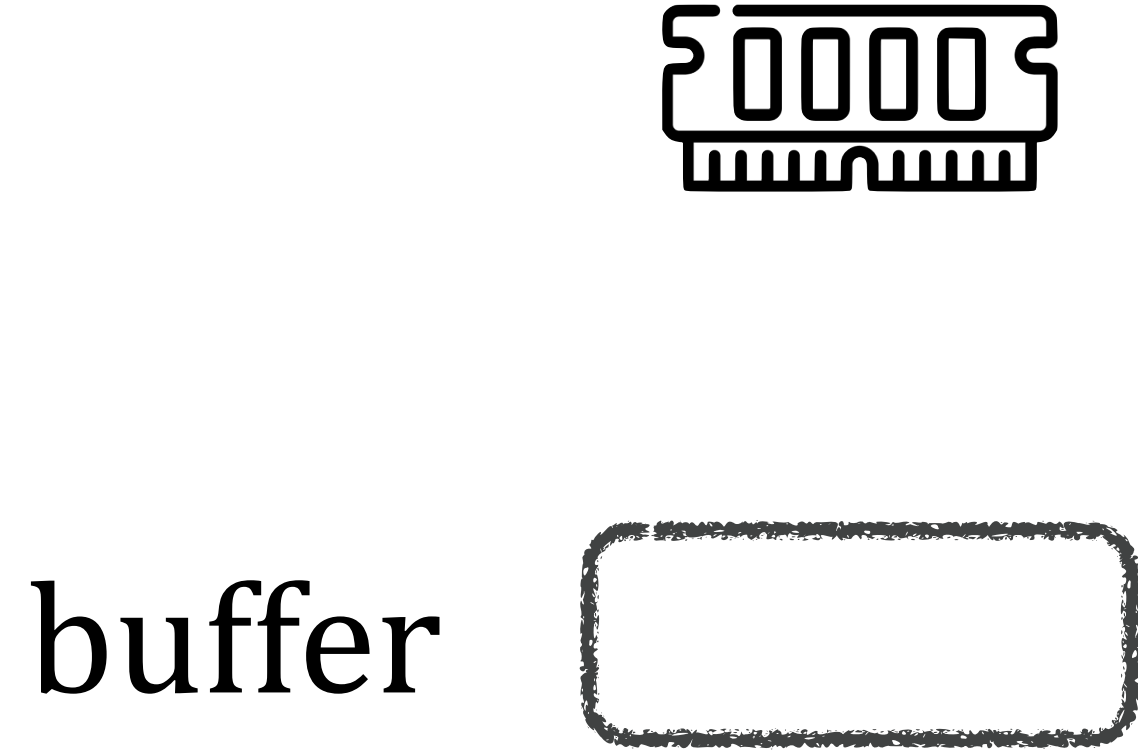
L1



LSM Ingestion

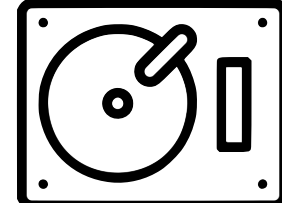
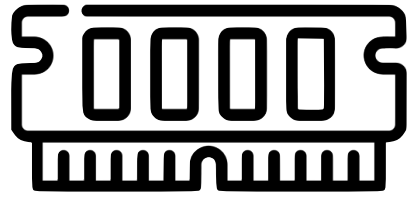


LSM Compaction



exponentially larger capacity

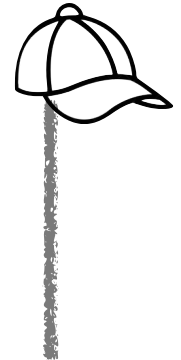
LSM Compaction



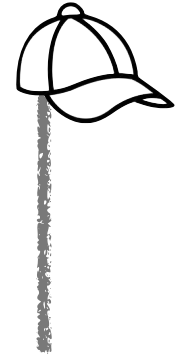
buffer



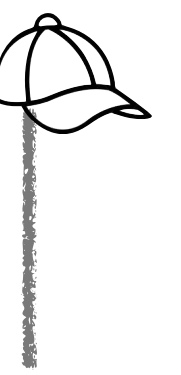
L1



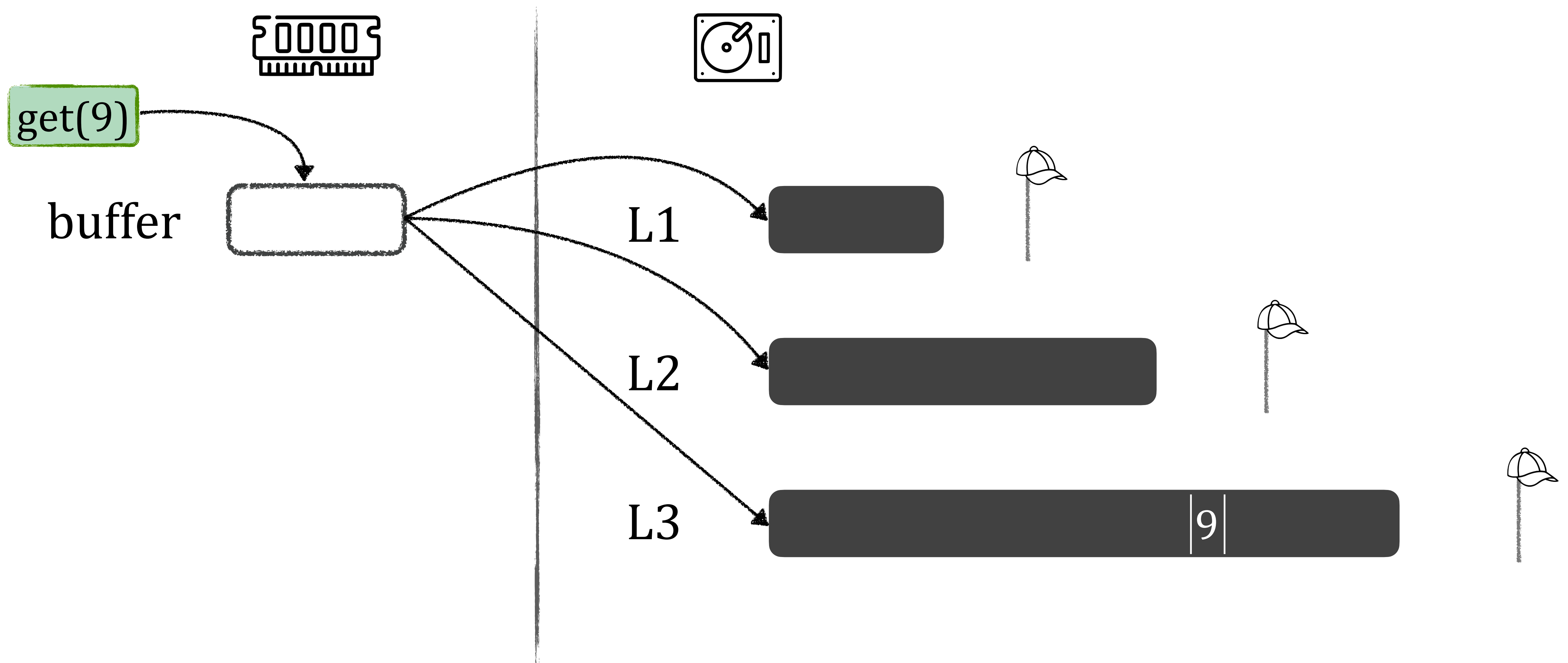
L2



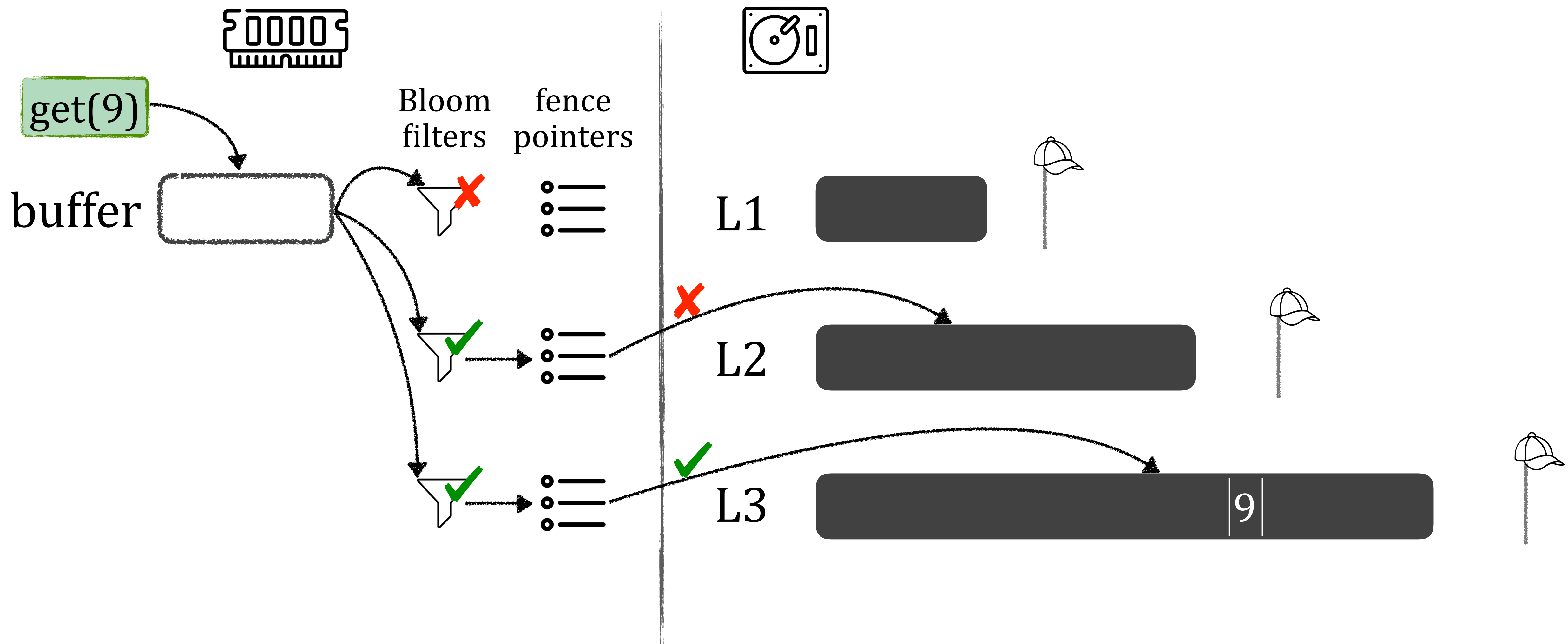
L3



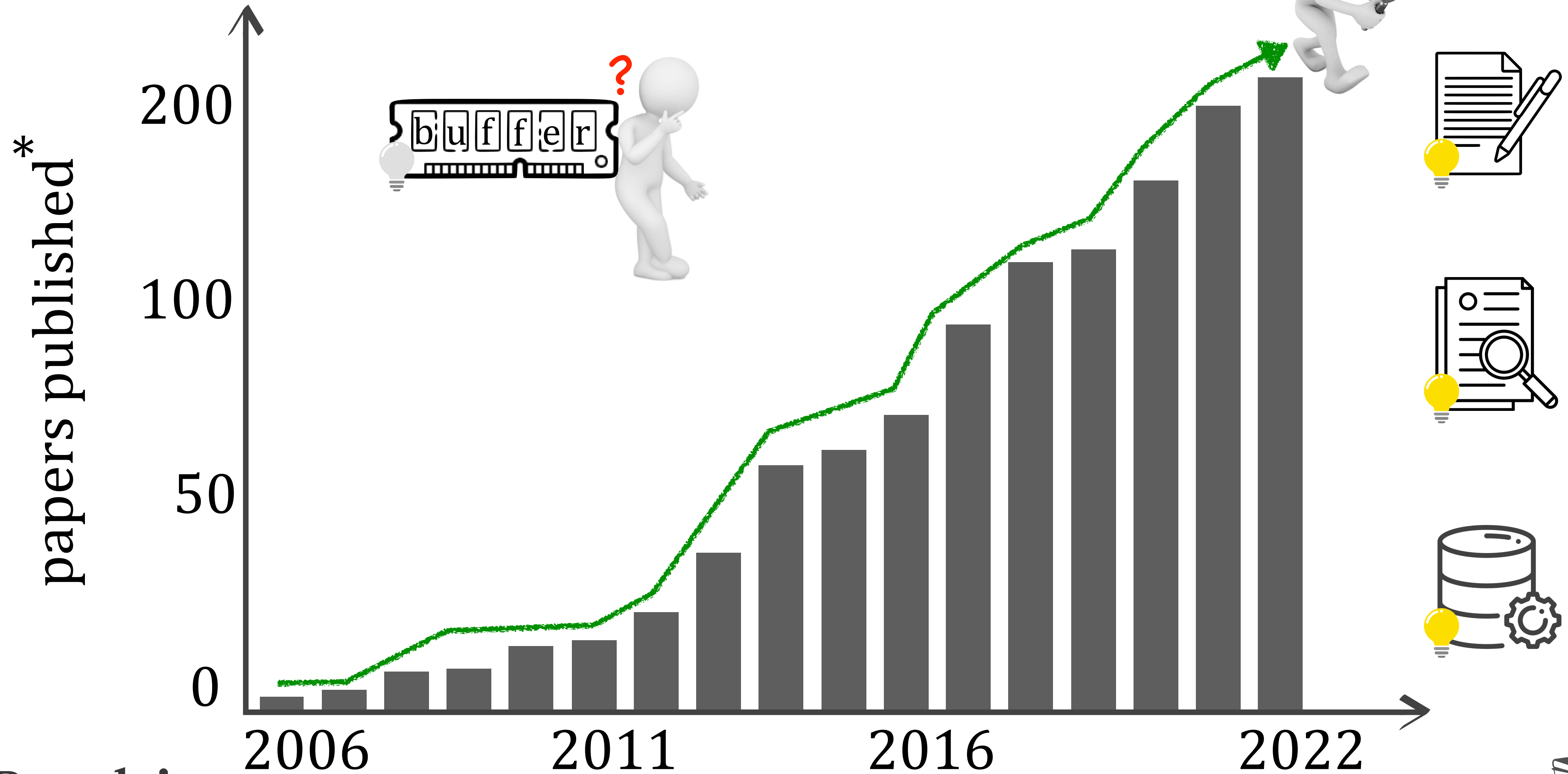
LSM Lookup



LSM Lookup



Research Trend



* data from Google scholar



Motivation

DEFAULT
skip-list



Motivation

DEFAULT
skip-list



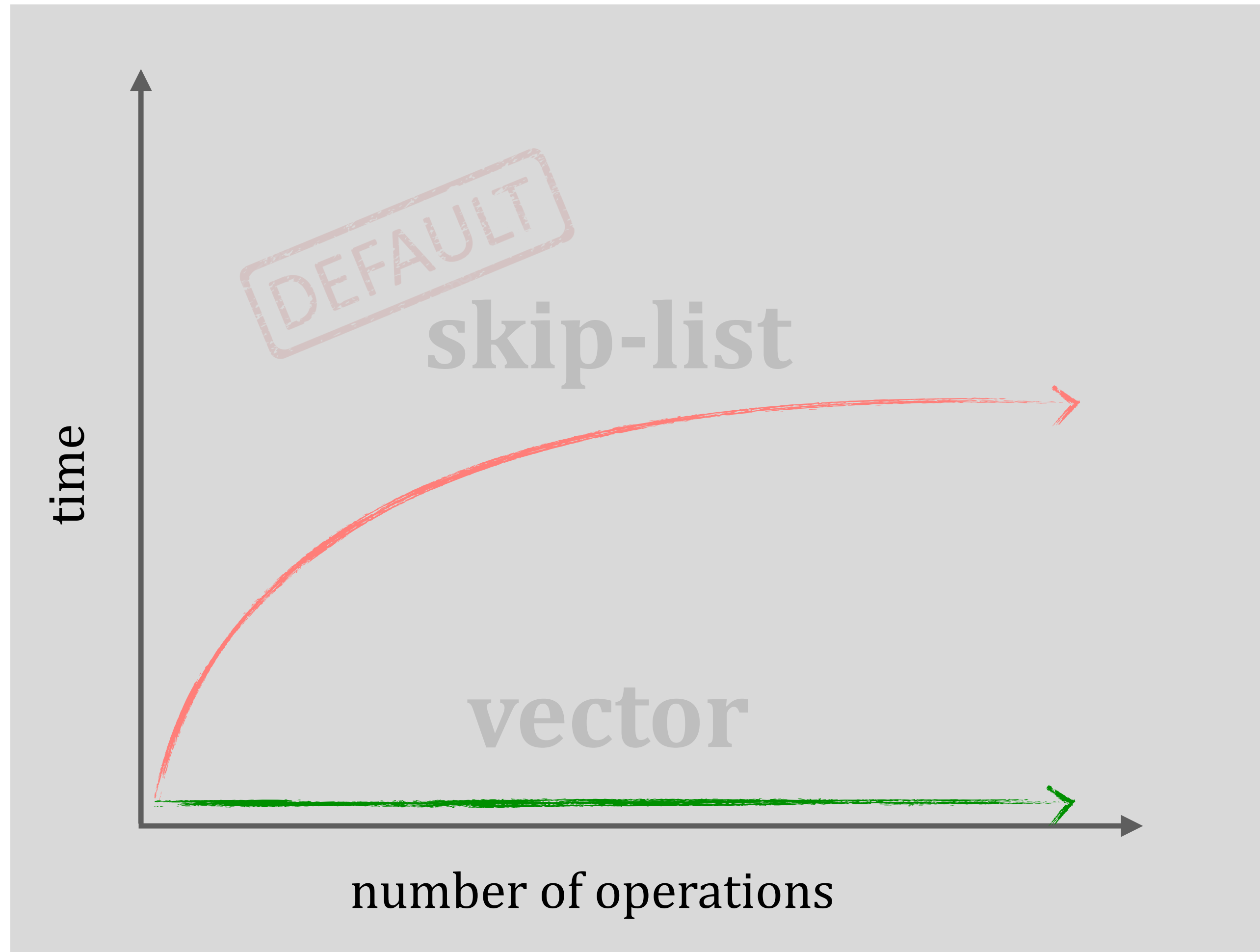
event logging



time-series

insert only

Motivation

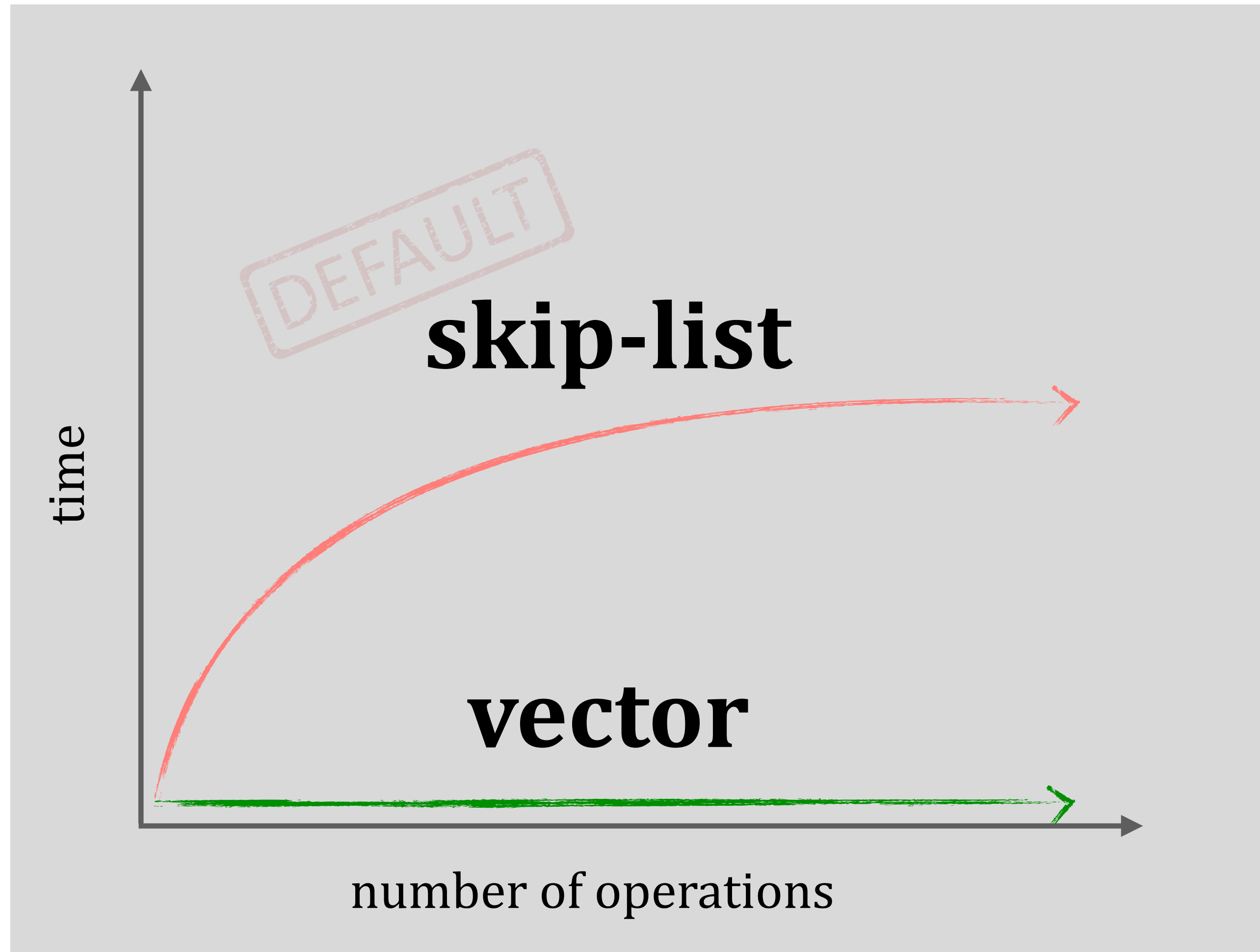


event logging

time-series

insert only

Motivation

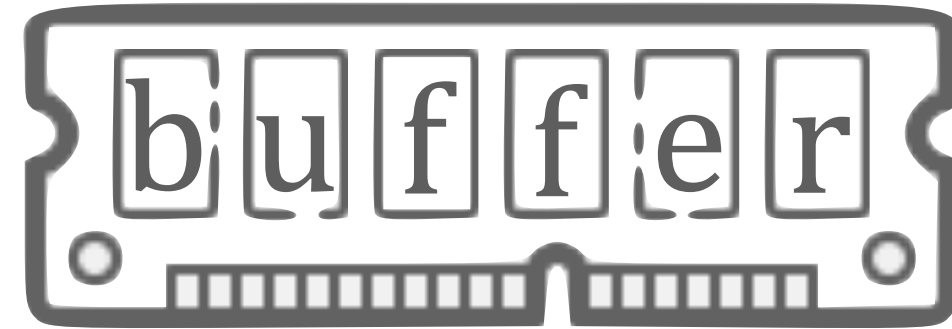


event logging

time-series

insert only

Contribution



1 **vector** dynamic

4 **hash** skip-list

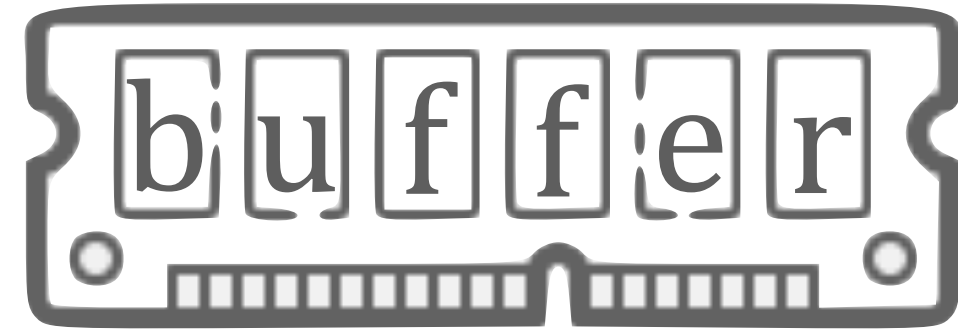
2 **vector** static

3 **skip-list**

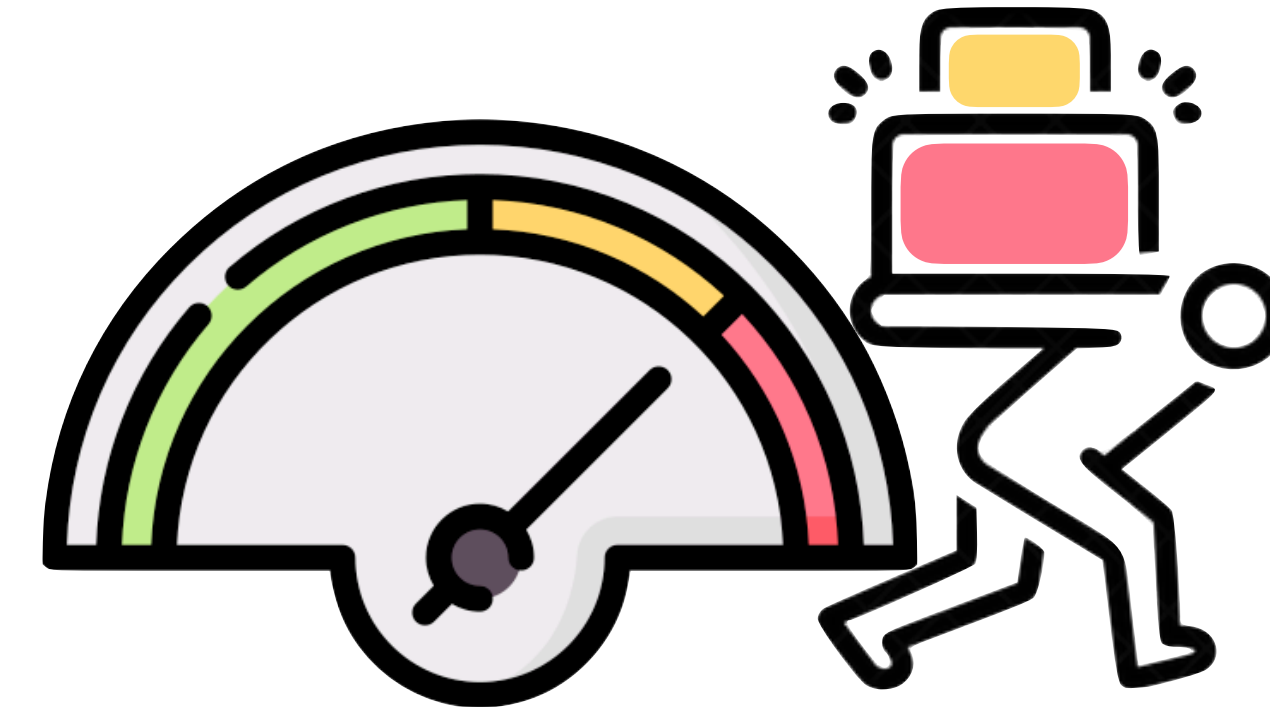
5 **hash** linked-list



Contribution

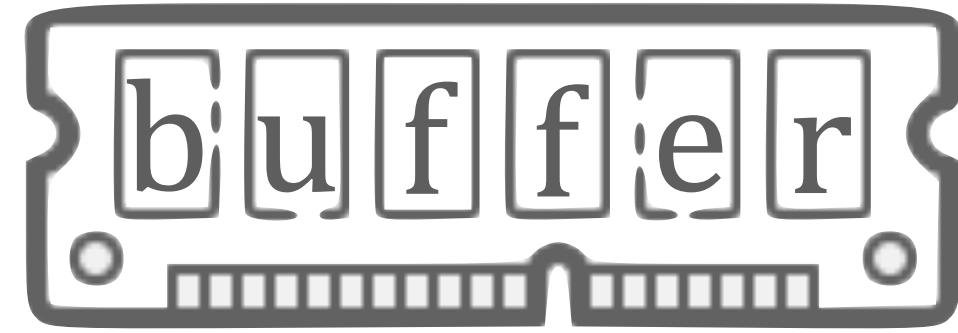


- 1 **vector** dynamic
- 2 **vector** static
- 3 **skip-list**
- 4 **hash** skip-list
- 5 **hash** linked-list

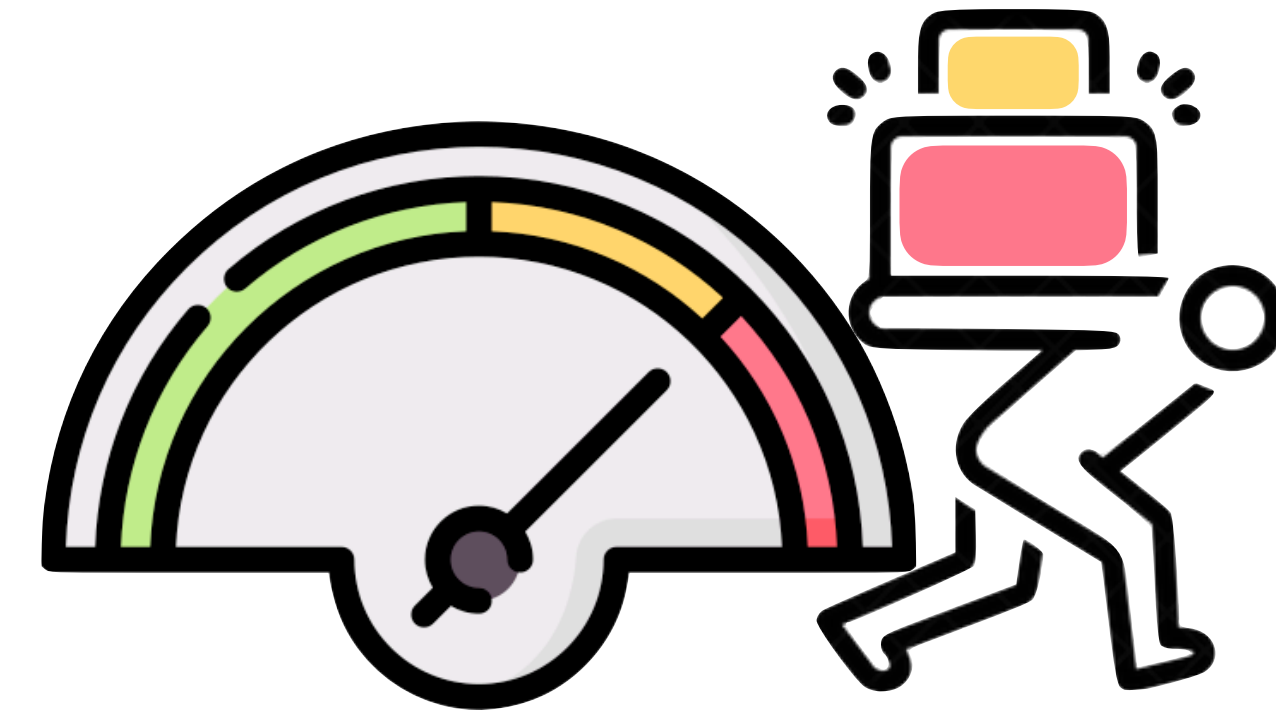


benchmark

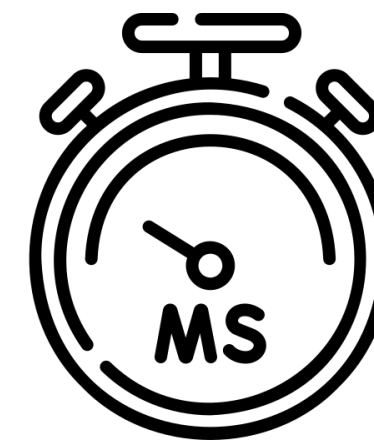
Contribution



- 1 **vector** dynamic
- 2 **vector** static
- 3 **skip-list**
- 4 **hash** skip-list
- 5 **hash** linked-list



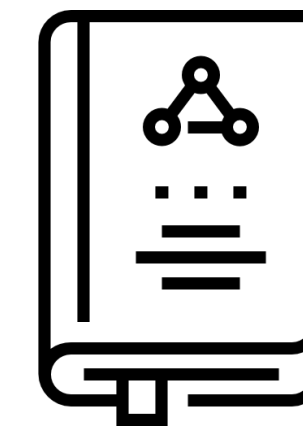
benchmark



latency

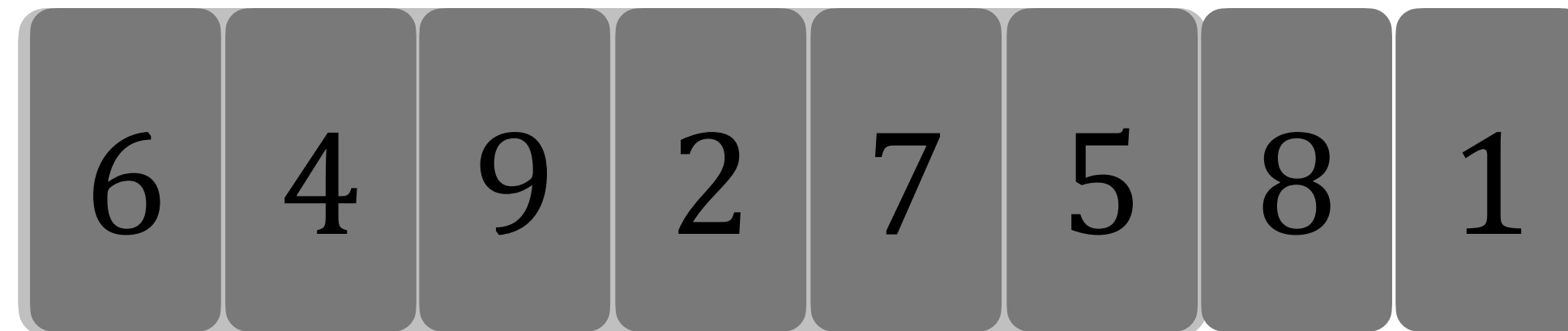


footprint



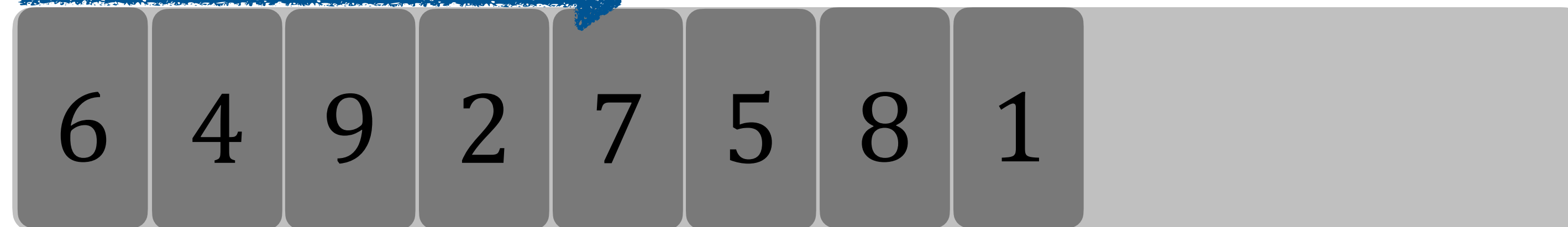
handbook

Implementation: Vector



dynamic

get(7)



static

- great for insert-heavy w/l
- no extra space needed
- expensive point queries

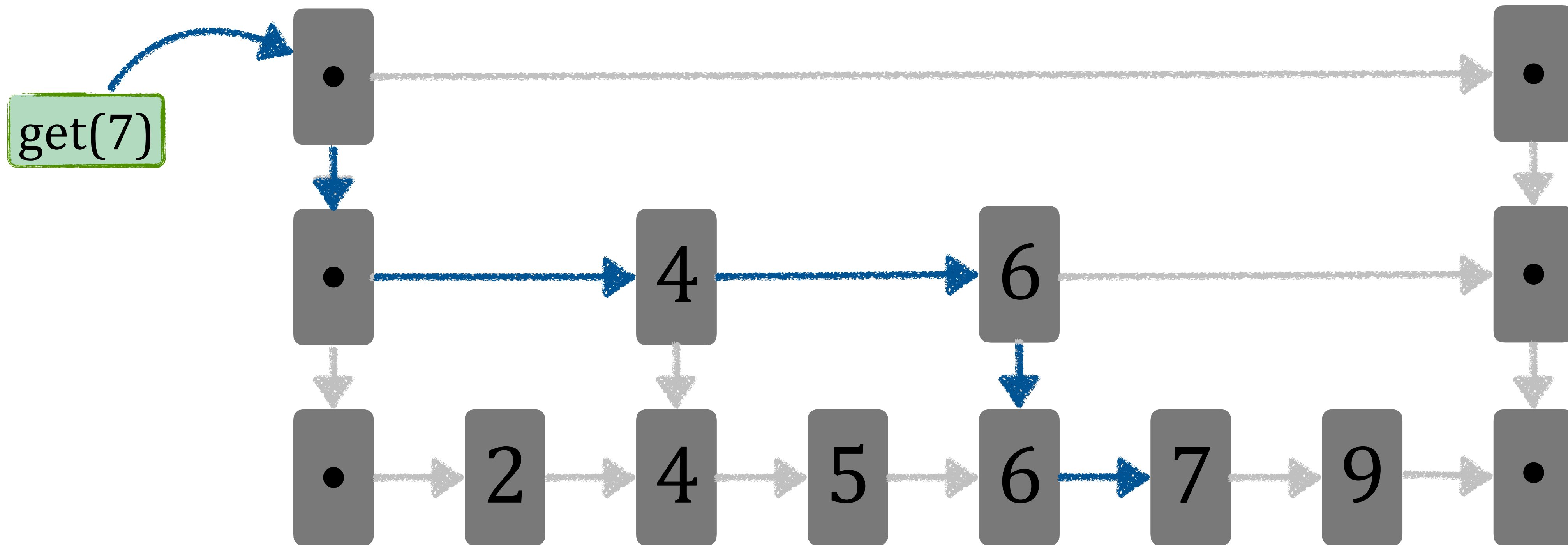
insert cost: $\mathcal{O}(1)$

space complexity: $\mathcal{O}(N)$

point query cost: $\mathcal{O}(N)$

N : number of entries
M : meta data

Implementation: Skip-list



- great for mixed w/l
- some extra space needed
- good for point queries

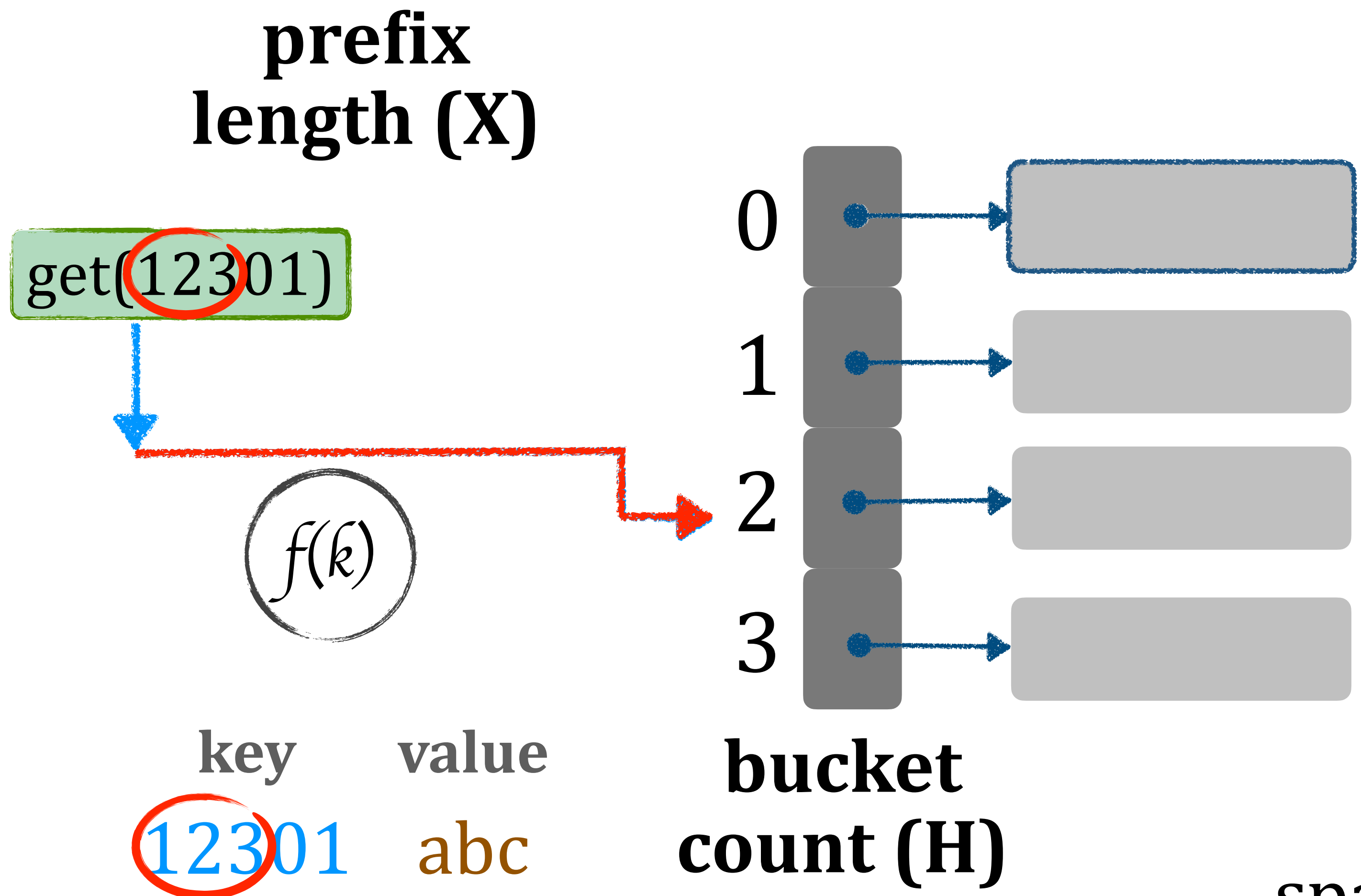
insert cost: $\mathcal{O}(\log N)$

space complexity: $\mathcal{O}(N + M)$

point query cost: $\mathcal{O}(\log N)$

N : number of entries
M : meta data

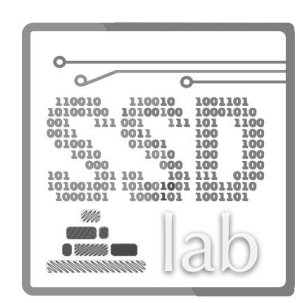
Implementation: Hash Hybrids



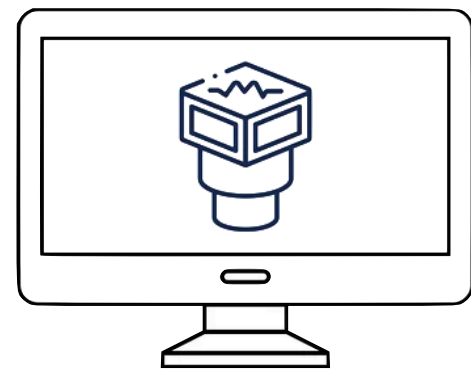
**skip-list
linked-list**

- great for mixed w/l
- more meta space needed
- good for point queries

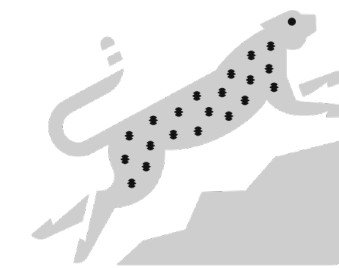
insert cost: $\mathcal{O}(\log N/H)$
space complexity: $\mathcal{O}(N + M)$
point query cost: $\mathcal{O}(\log N/H)$



Experiments



2 Intel Gold 6126 vCPUs
192 GB RAM
240 GB SSD
Ubuntu 20.04 LTS

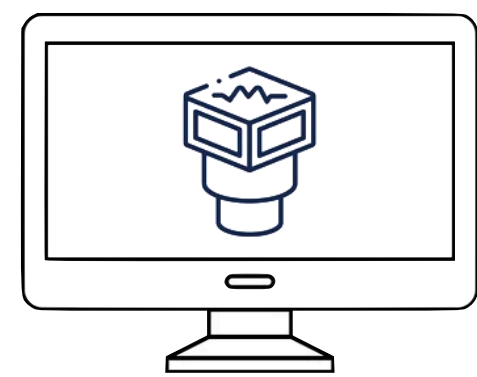


RocksDB v9.0.0

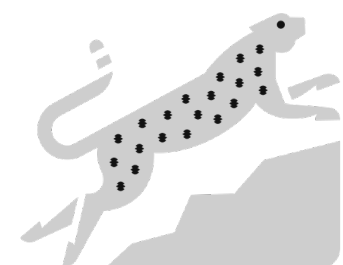


size ratio	4
buffer size	16MB
page size	4KB
entry size	64B
key size	10

Experiments



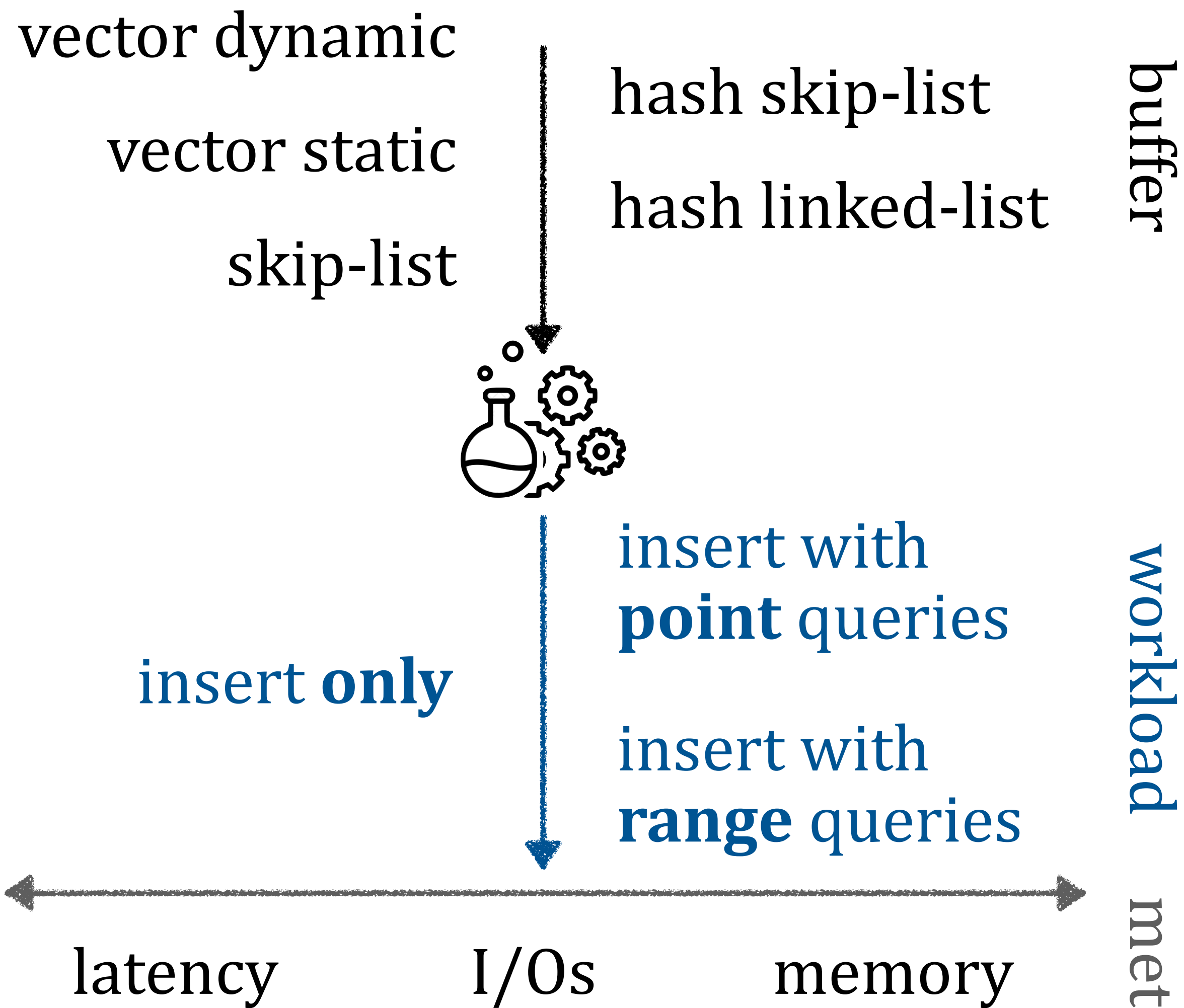
2 Intel Gold 6126 vCPUs
192 GB RAM
240 GB SSD
Ubuntu 20.04 LTS



RocksDB v9.0.0

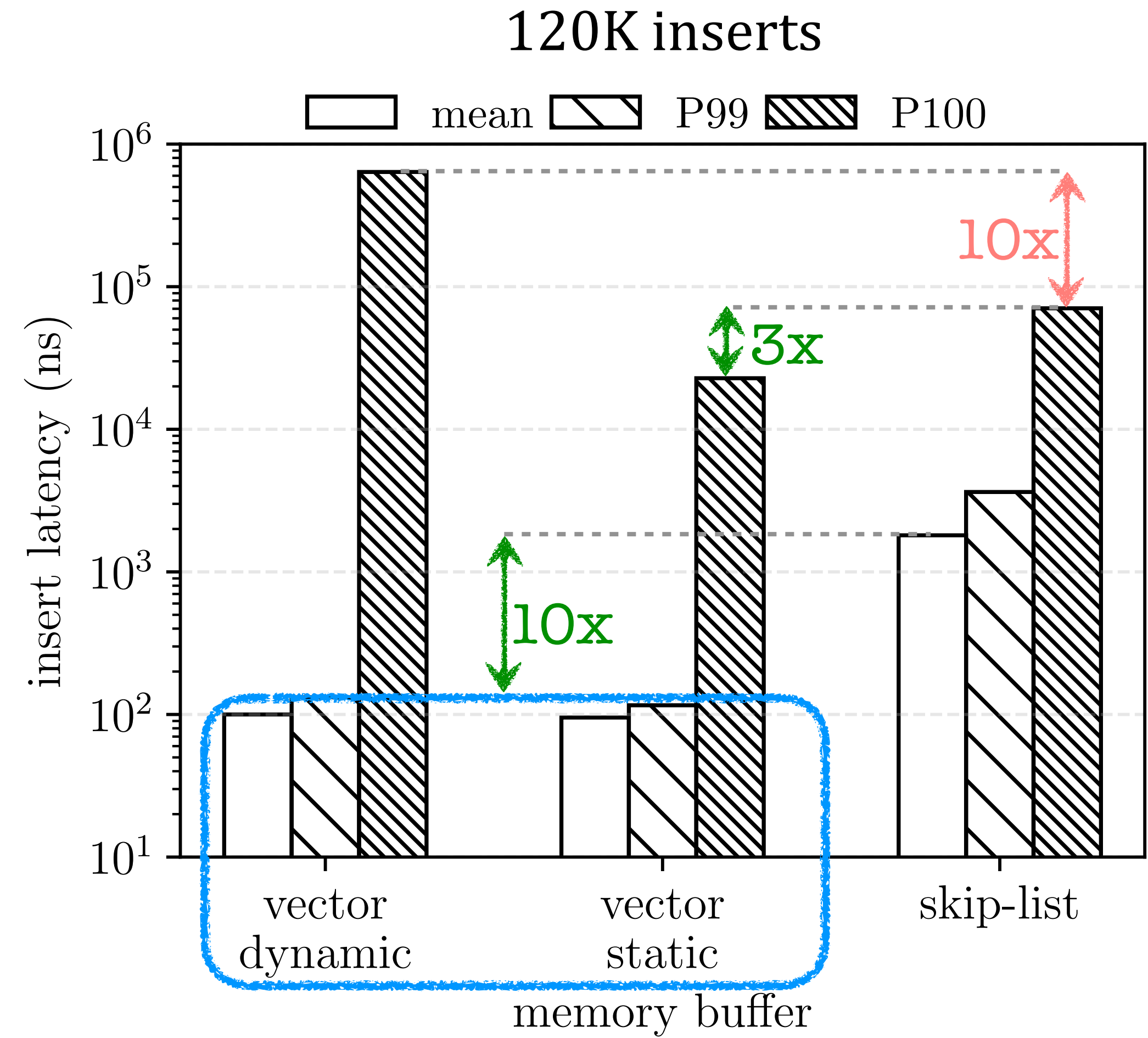


size ratio	4
buffer size	16MB
page size	4KB
entry size	64B
key size	10



Evaluation

vector is 10x faster than skip-list for an *insert only* w/l



insert only



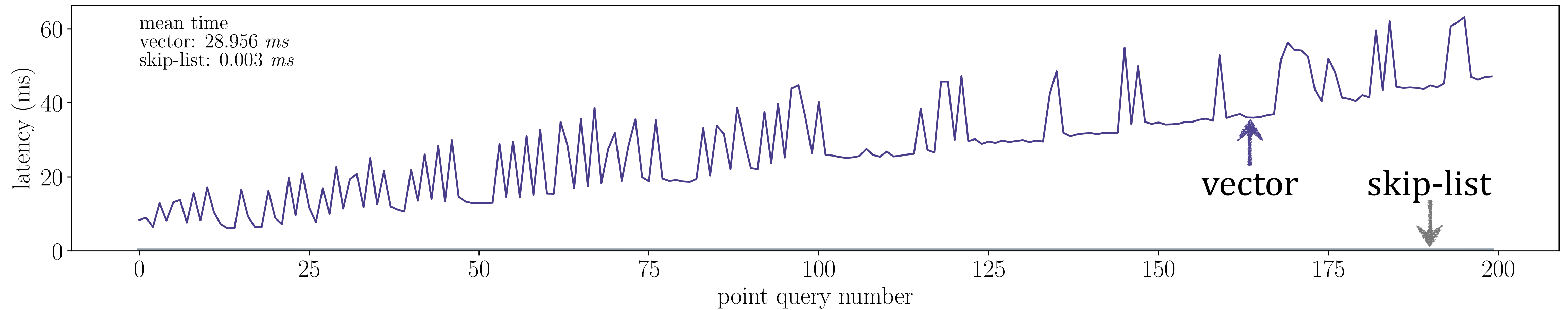
Brandeis UNIVERSITY

Evaluation

vector is 10x faster than skip-list for an *insert only* w/l



140K inserts, 200 interleaved point queries



insert with point queries



Evaluation

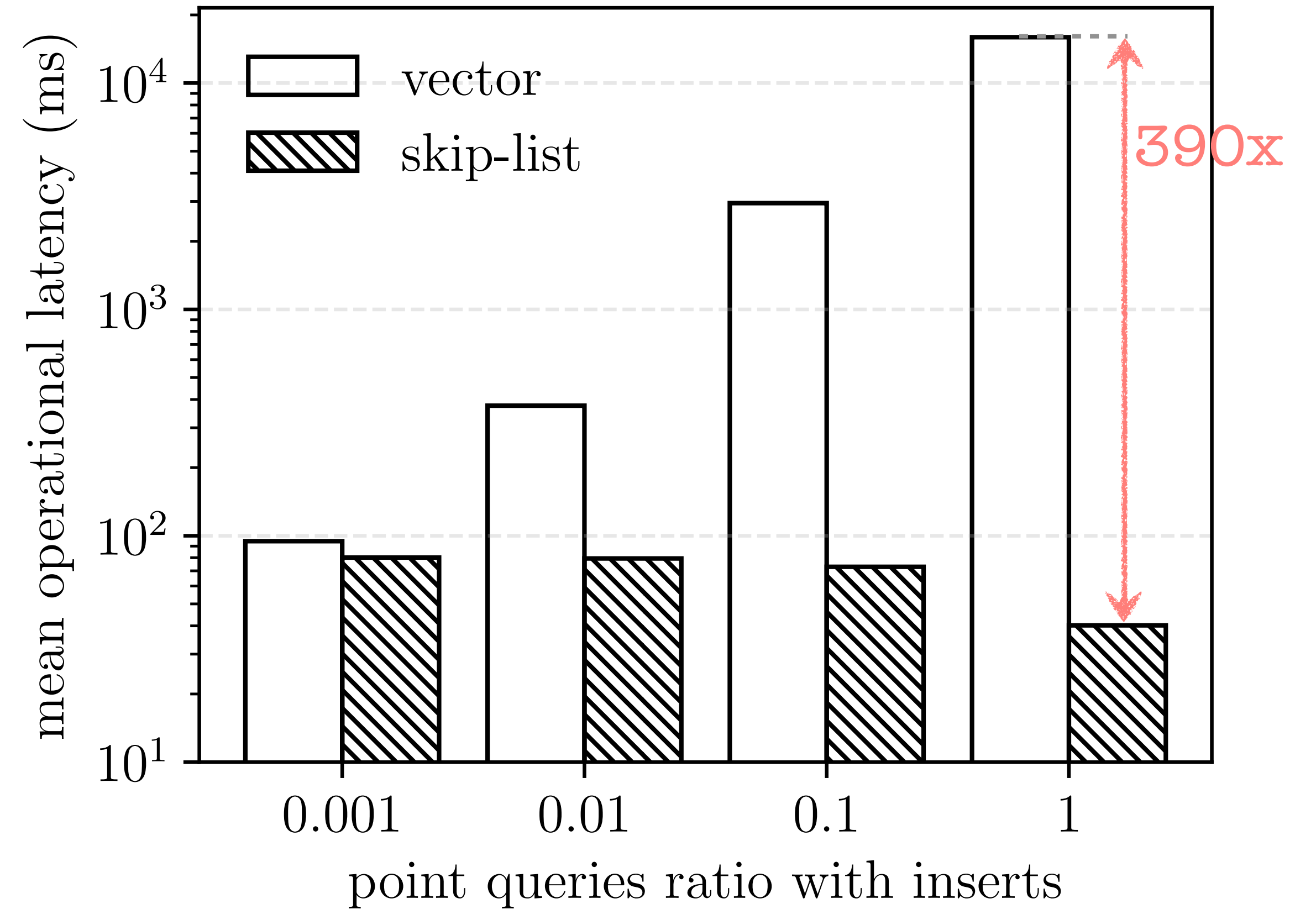
vector is 10x faster than skip-list for an *insert only* w/l



vector is worst choice for w/l's with point queries



120K inserts, interleaved point queries



insert with point queries



Evaluation

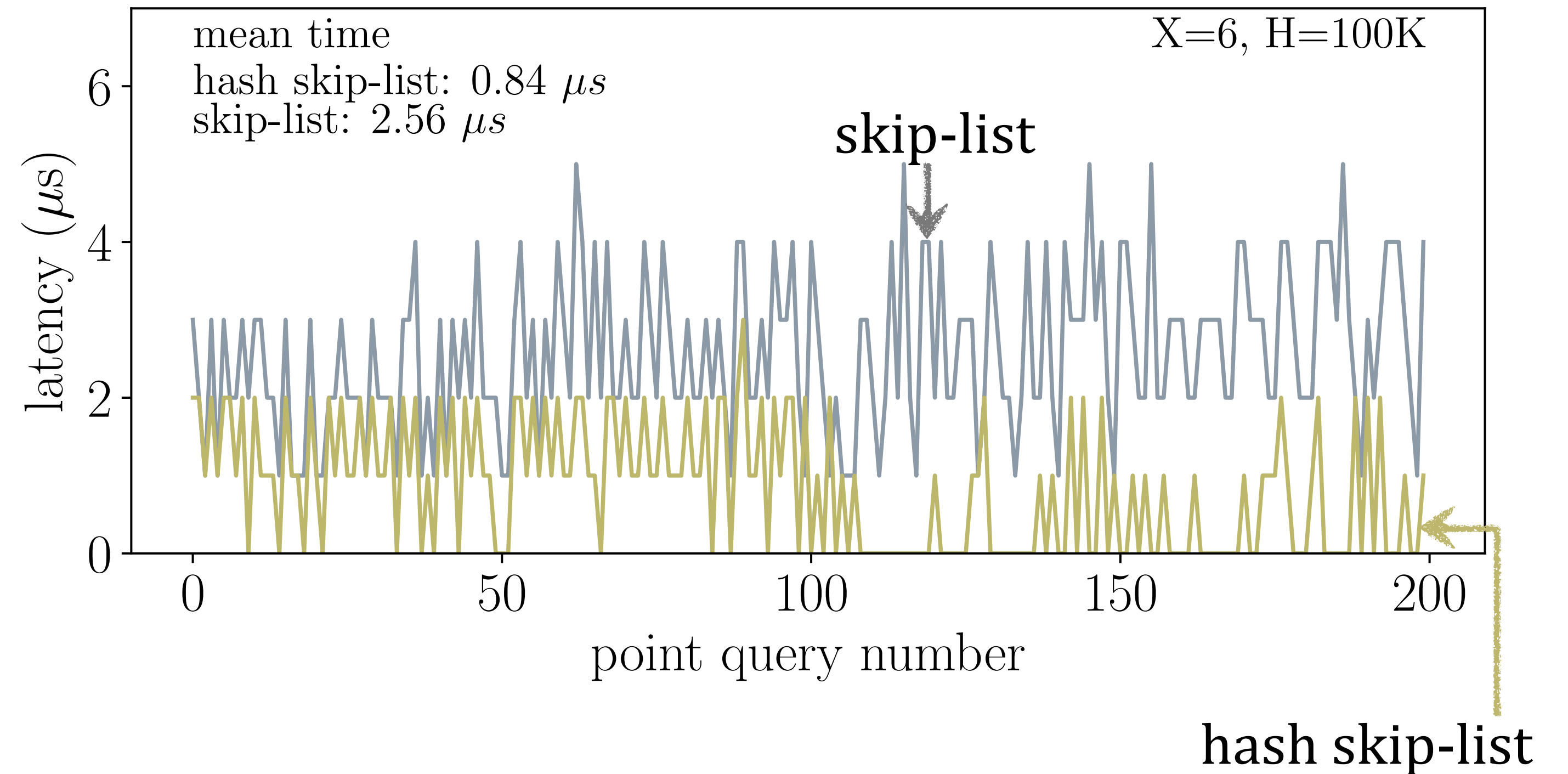
vector is 10x faster than skip-list for an *insert only* w/l



vector is worst choice for w/l's with point queries



140K inserts, 200 interleaved point queries



insert with point queries



Evaluation

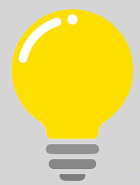
vector is 10x faster than skip-list for an *insert only* w/l



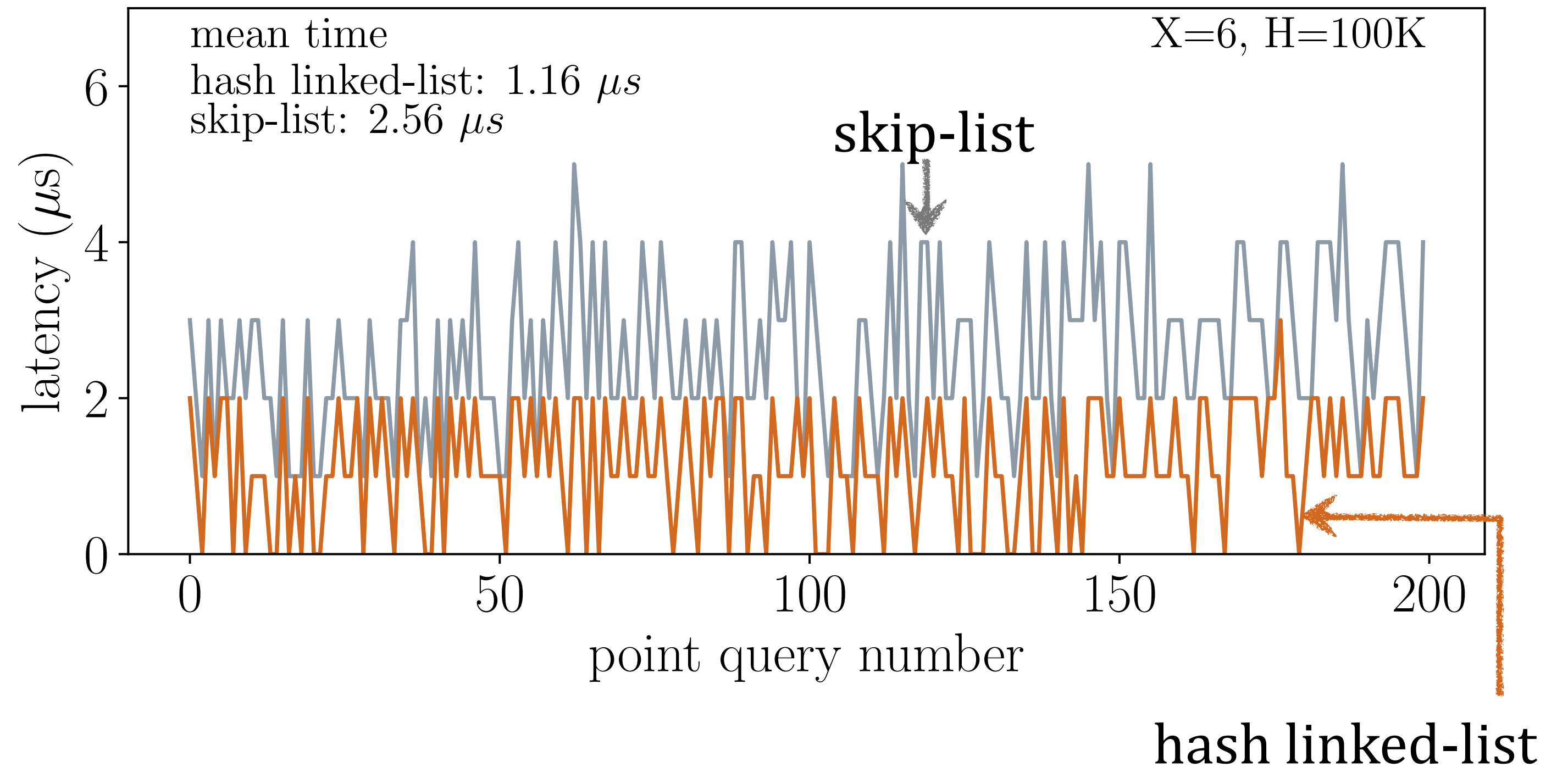
vector is worst choice for w/l's with point queries



hash buffers are 2-3x faster than skip-list for point queries w/l's



140K inserts, 200 interleaved point queries



insert with point queries



Evaluation

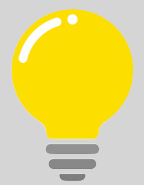
vector is 10x faster than skip-list for an *insert only* w/l



vector is worst choice for w/l's with point queries



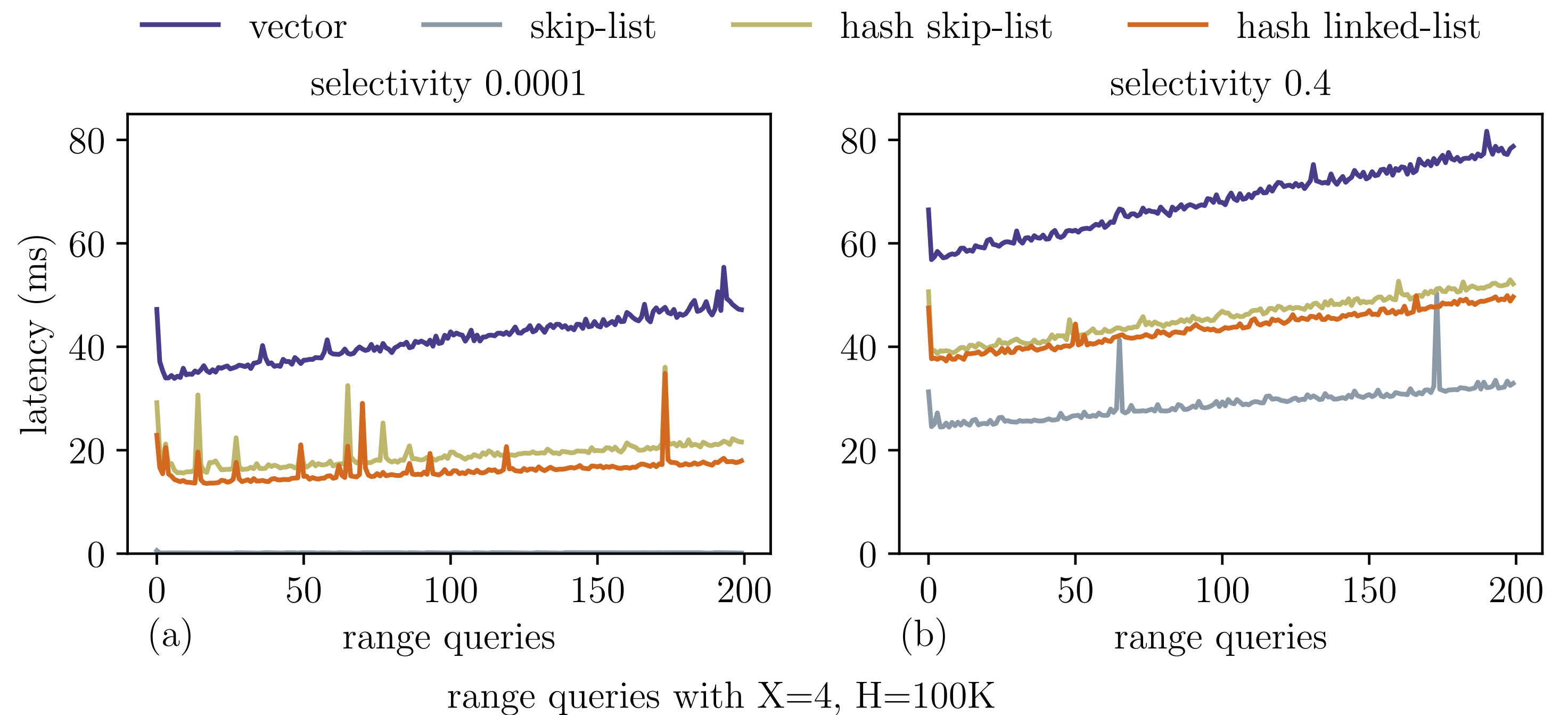
hash buffers are 2-3x faster than skip-list for point queries w/l's



skip-list outperforms for w/l's with range queries



140K inserts, 200 interleaved range queries



insert with range queries



Evaluation

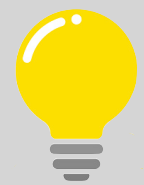
vector is 10x faster than skip-list for an *insert only* w/l



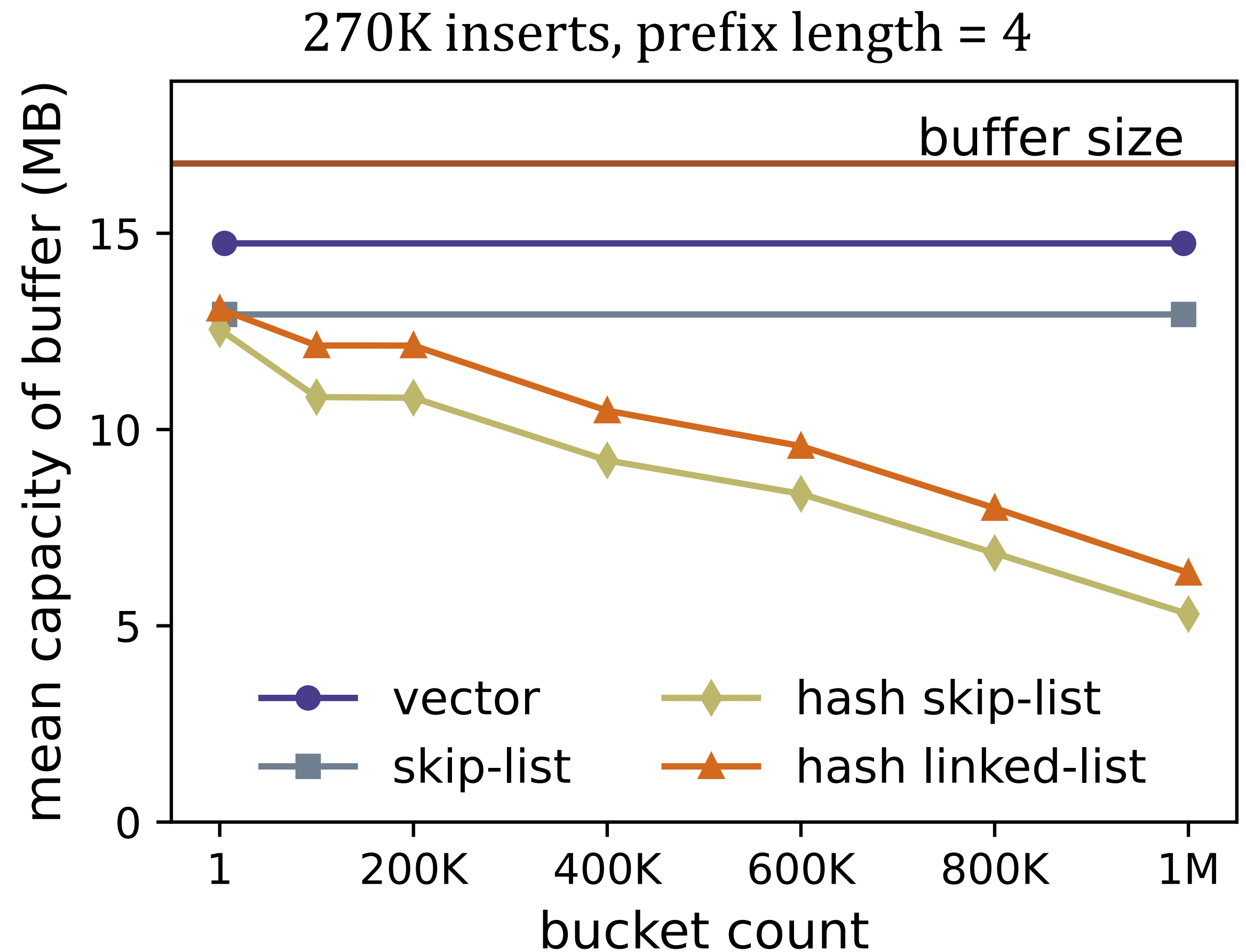
vector is worst choice for w/l's with point queries



hash buffers are 2-3x faster than skip-list for point queries w/l's



skip-list outperforms for w/l's with range queries



memory footprint



Evaluation

vector is 10x faster than skip-list for an *insert only* w/l



vector is worst choice for w/l's with point queries



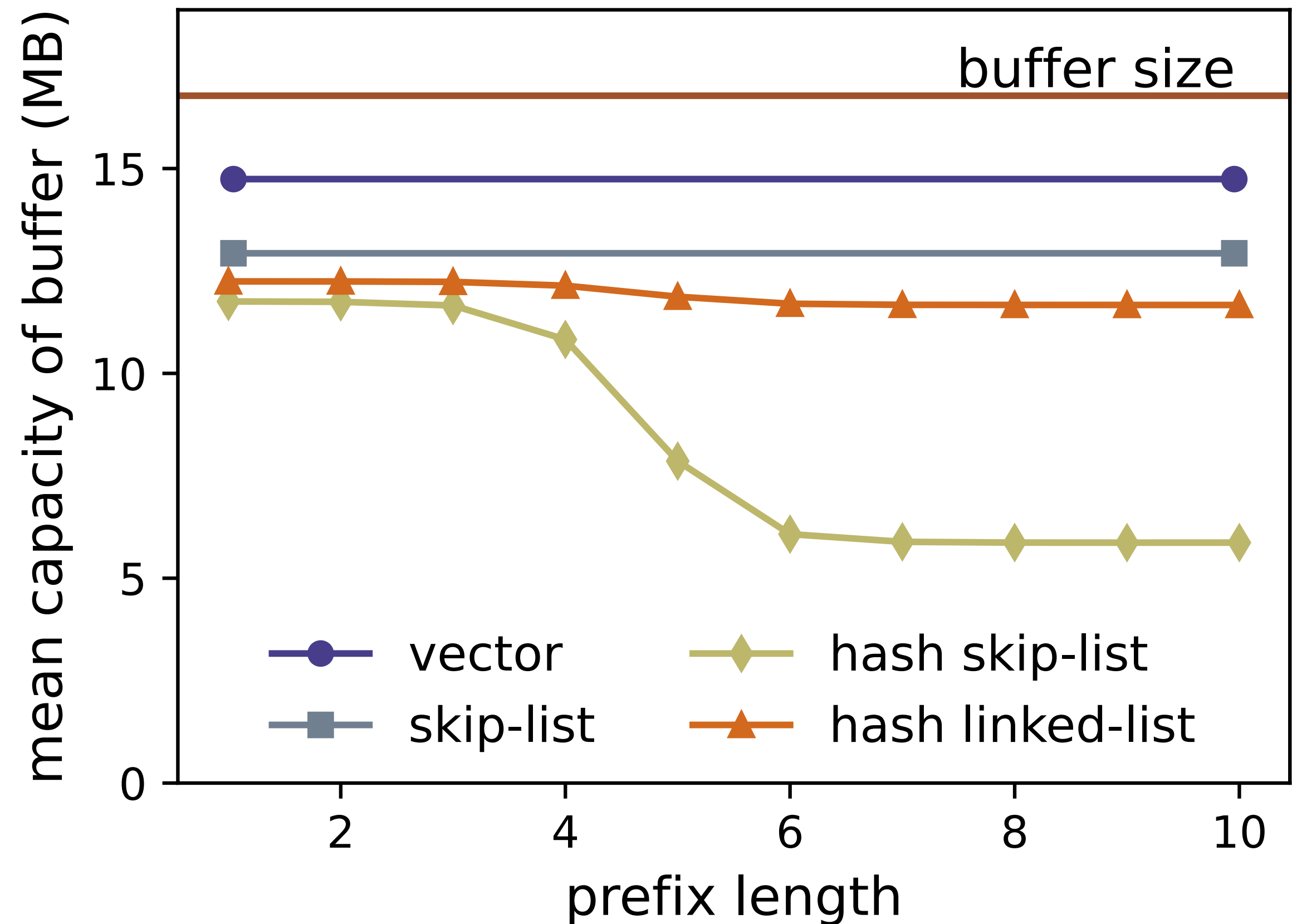
hash buffers are 2-3x faster than skip-list for point queries w/l's



skip-list outperforms for w/l's with range queries



270K inserts, bucket count = 100K




memory footprint




Evaluation

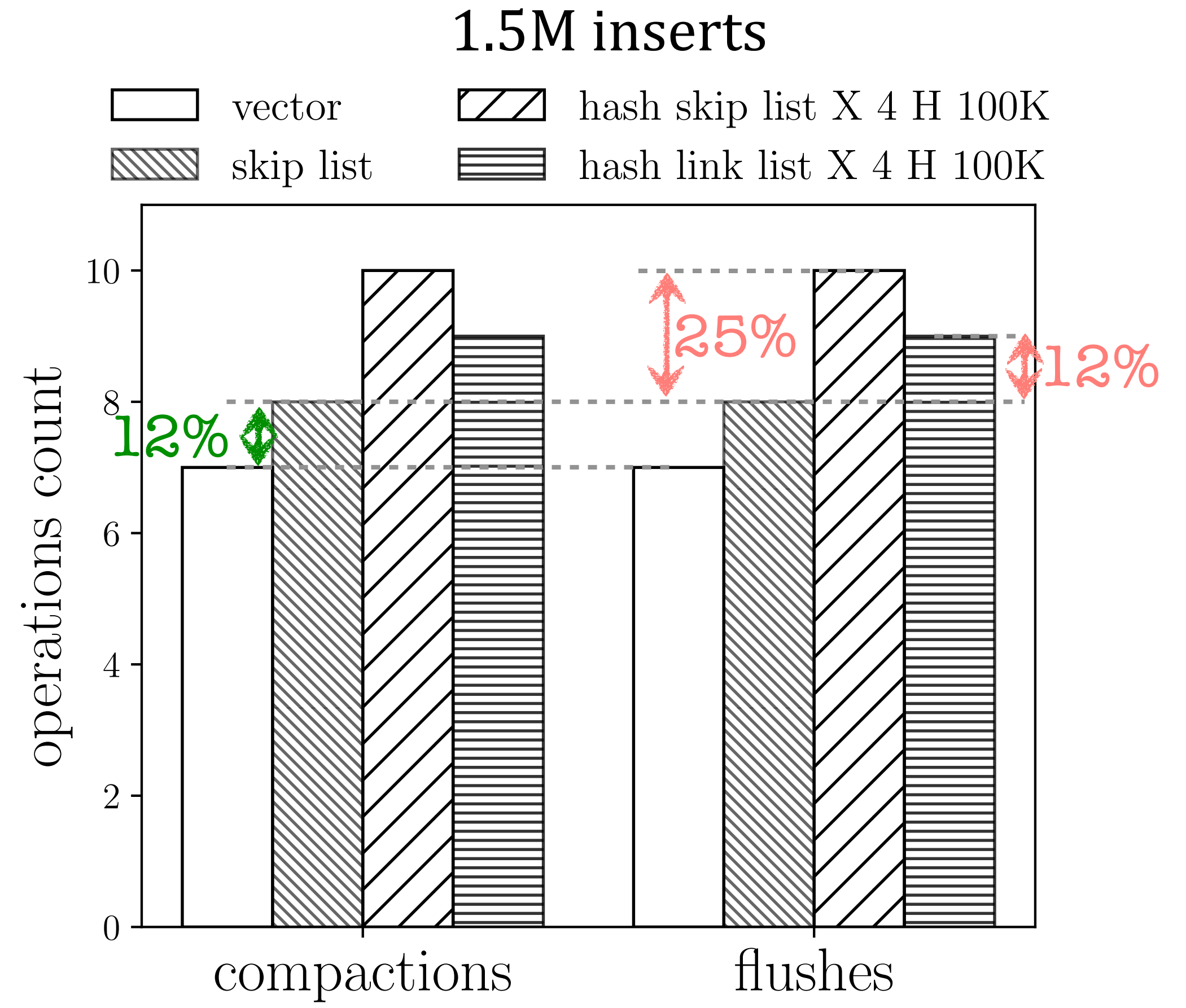
vector is 10x faster than skip-list for an *insert only* w/l 

vector is worst choice for w/l's with point queries 

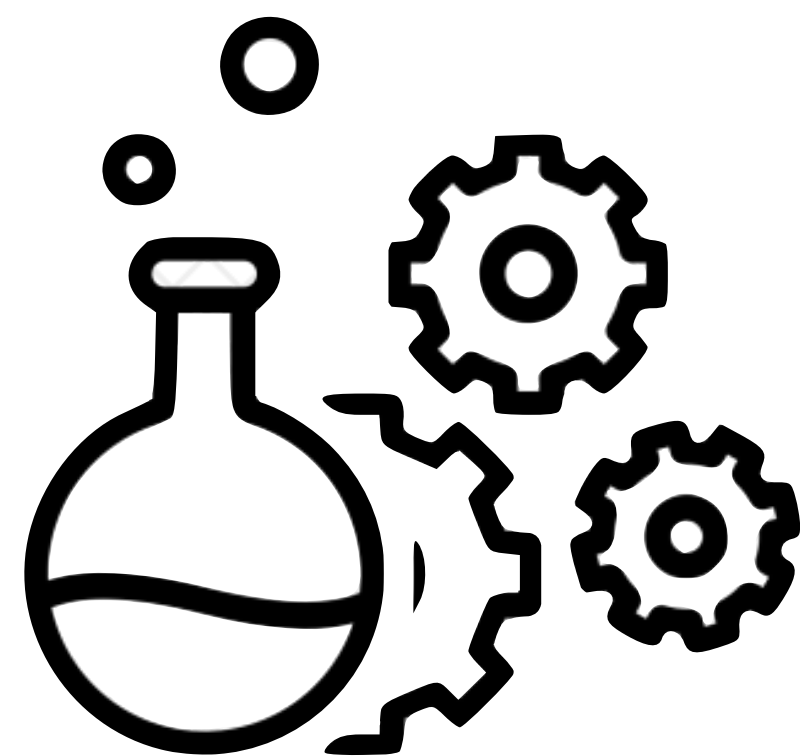
hash buffers are 2-3x faster than skip-list for point queries w/l's 

skip-list outperforms for w/l's with range queries 

vector perform 12% less flushes; more I/Os in hash based buffers 

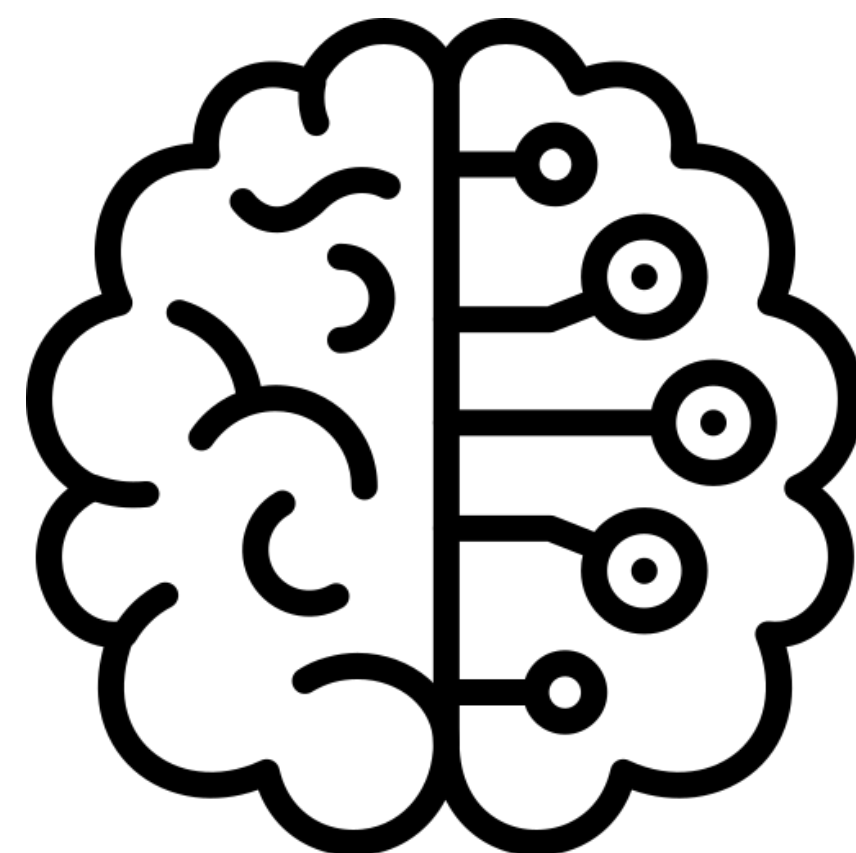


Future Work



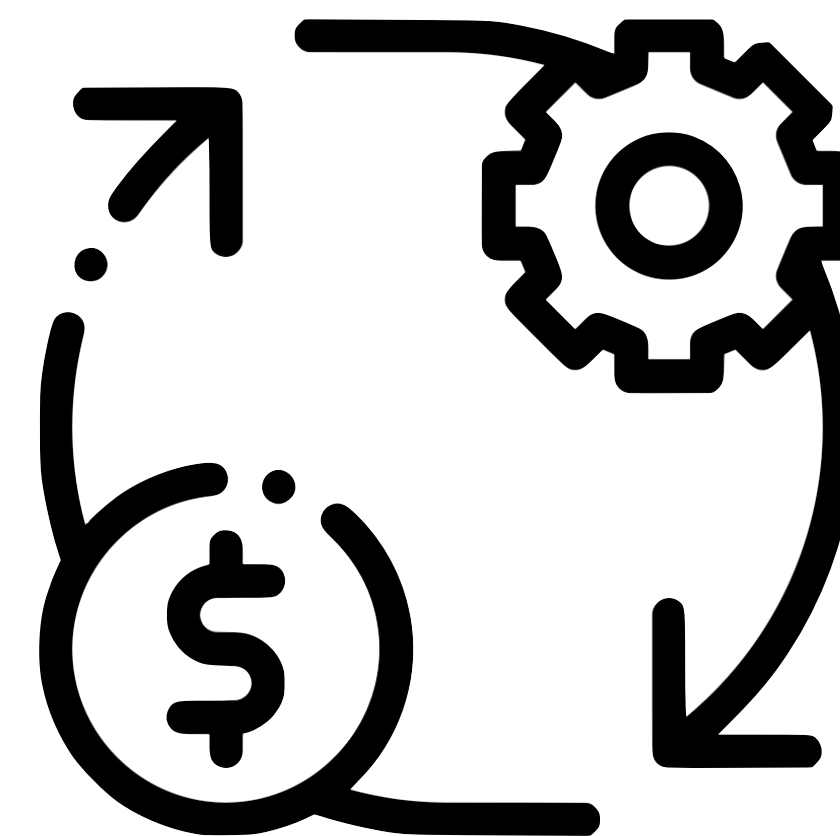
explore other
implementations

tree, trie, heap



finding best
buffer

using machine learning



switching
buffer

on the fly when w/l changes

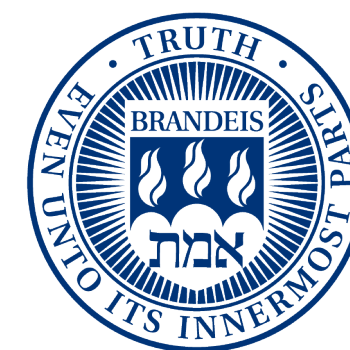
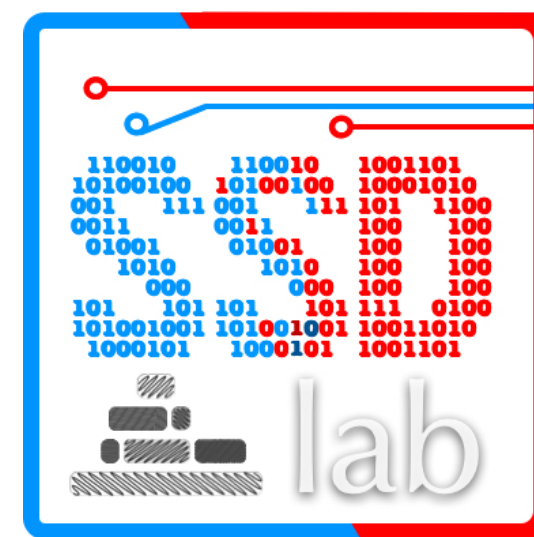
Anatomy of LSM Memory Buffer: Insights & Implications

Thank You!

Questions?

Shubham Kaushik

Subhadeep Sarkar



Brandeis
UNIVERSITY

