Anatomy of LSM Memory Buffer

Shubham Kaushik

kaushiks@brandeis.edu

Santiago, Chile, 2024

Log-Structured Merge-tree

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

1996

Patrick O'Neil[1], Edward Cheng[2]
Dieter Gawlick[3], Elizabeth O'Neil[1]

To be published: Acta Informatica

Brandeis
UNIVERSITY

LSM-based Datastore

2004 — 2024

APACHE HBASE

RocksDB

speedb

InnoDB

BigTable

cassandra

SCYLLA

CockroachDB

levelDB

WT

influxdb

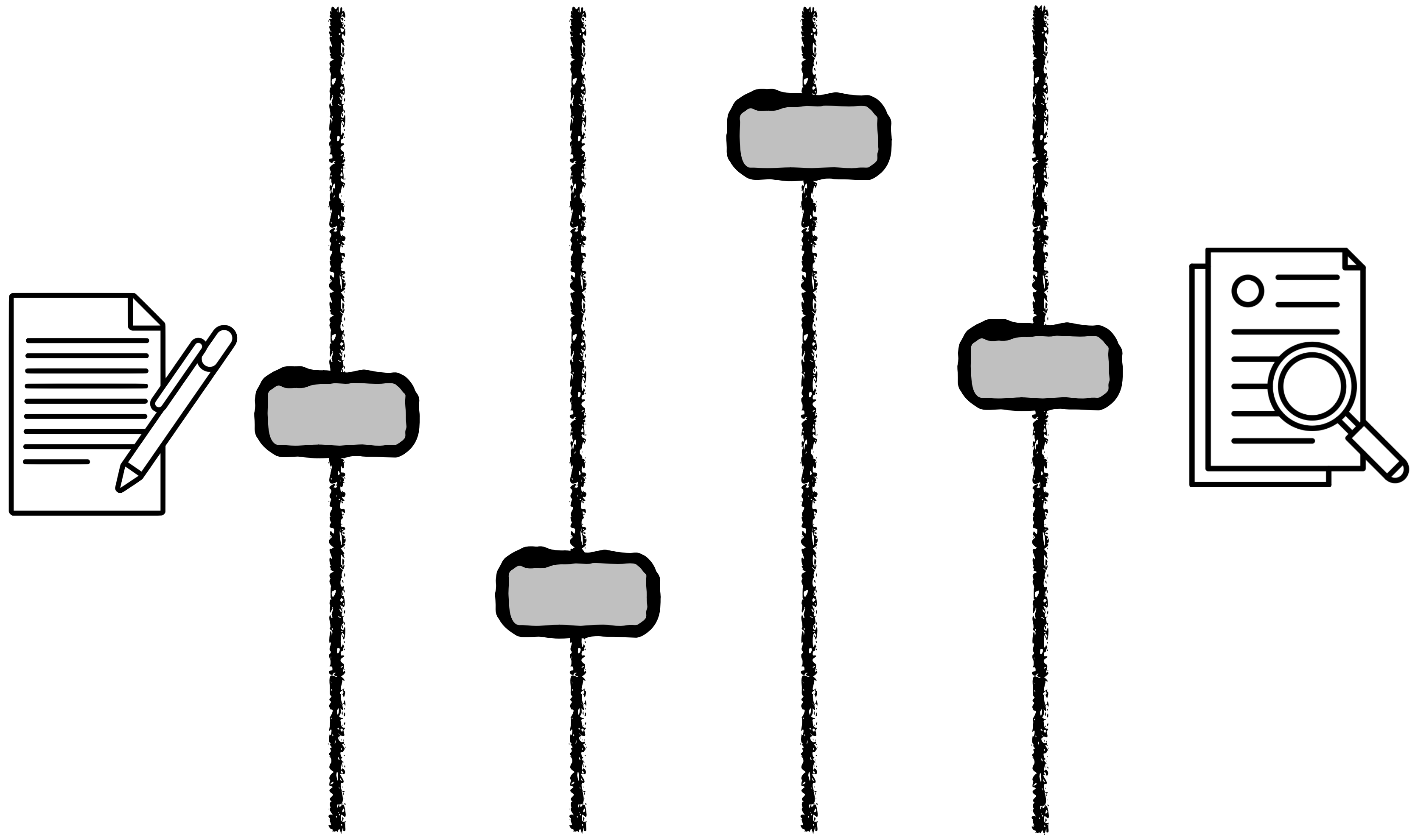
Couchbase

ClickHouse

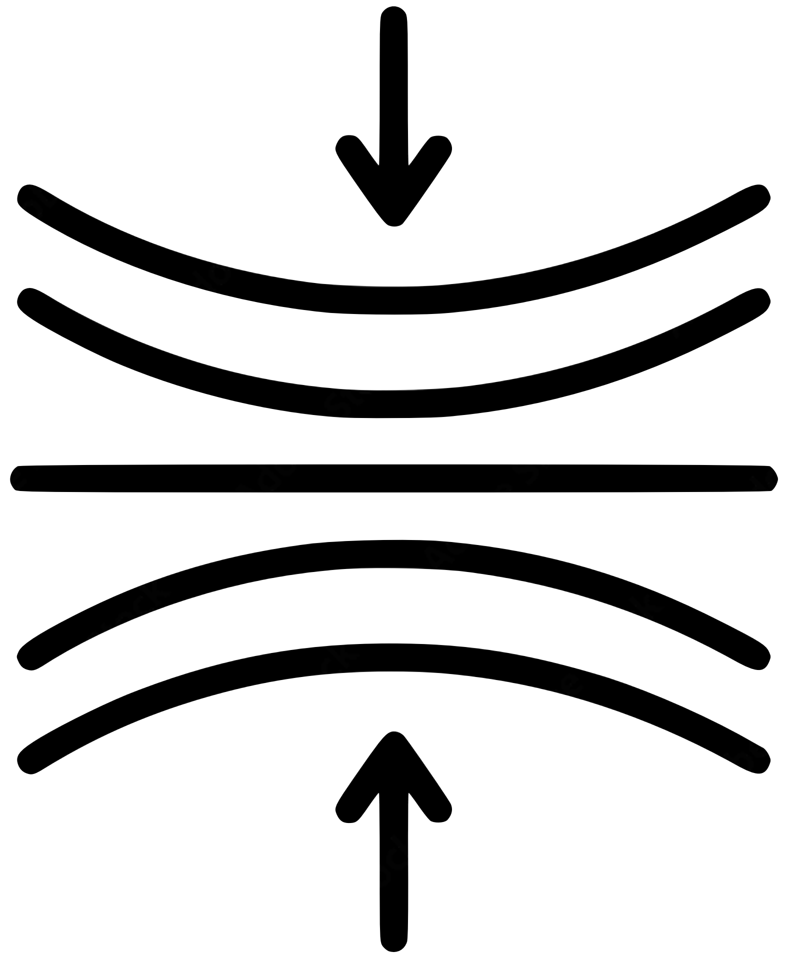
Brandeis UNIVERSITY

LSM-tree

fast **writes**

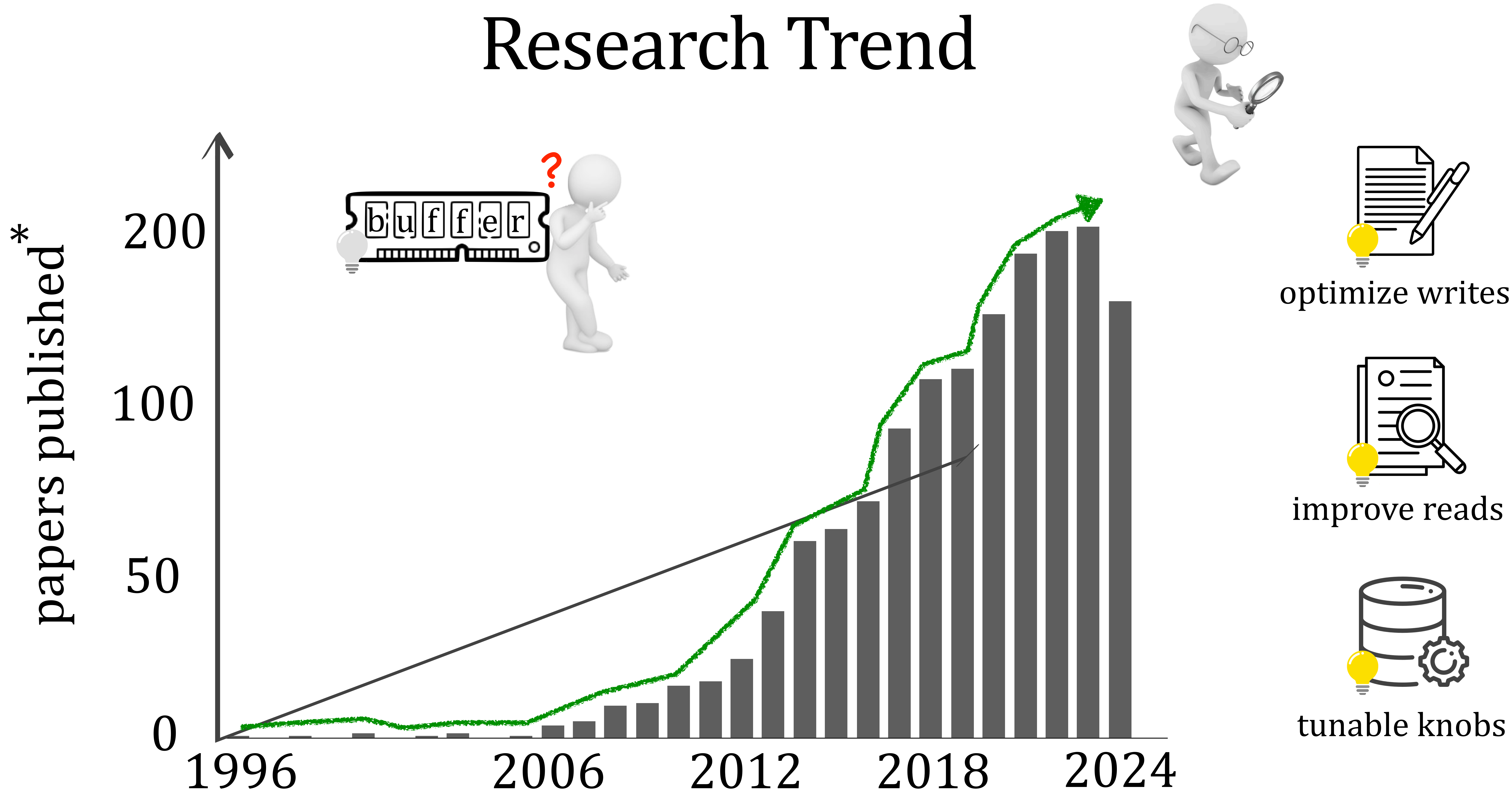


tunable read-write performance



good **space** utilization

Research Trend



* data from Google scholar

Memory Buffer

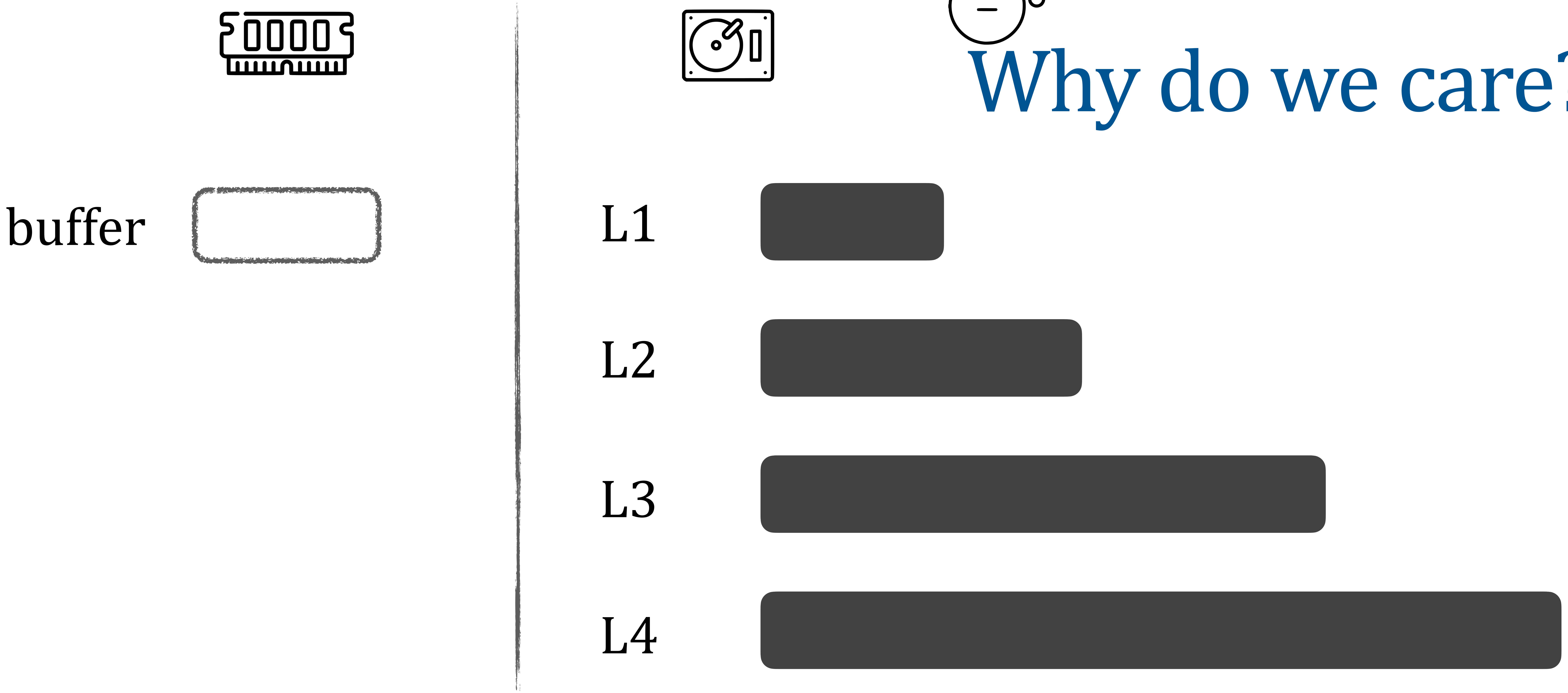
a *temporary space* that holds data *in memory*

Implementations of memory buffer

1. vector
2. skip-list
3. hash-hybrids
4. trie

LSM Basics

Why do we care?



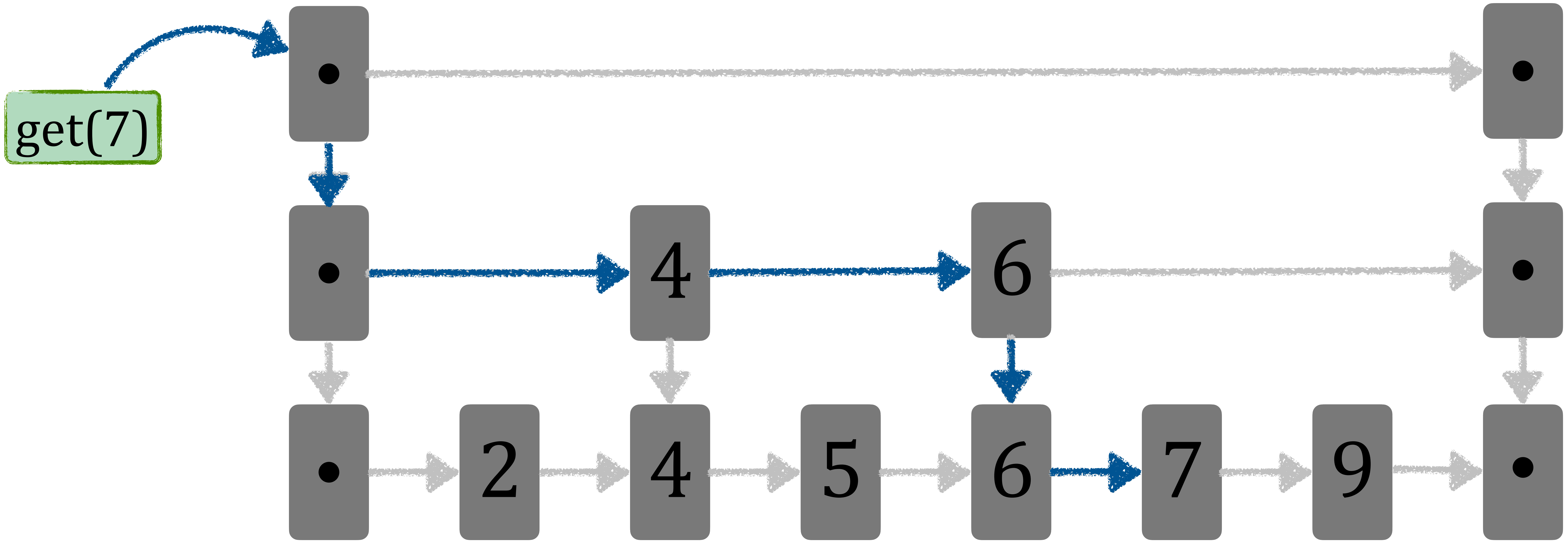
Ideal buffer implementation?

DEFAULT

skip-list

N : entries in buffer
M : meta data

Implementation: Skip-list



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

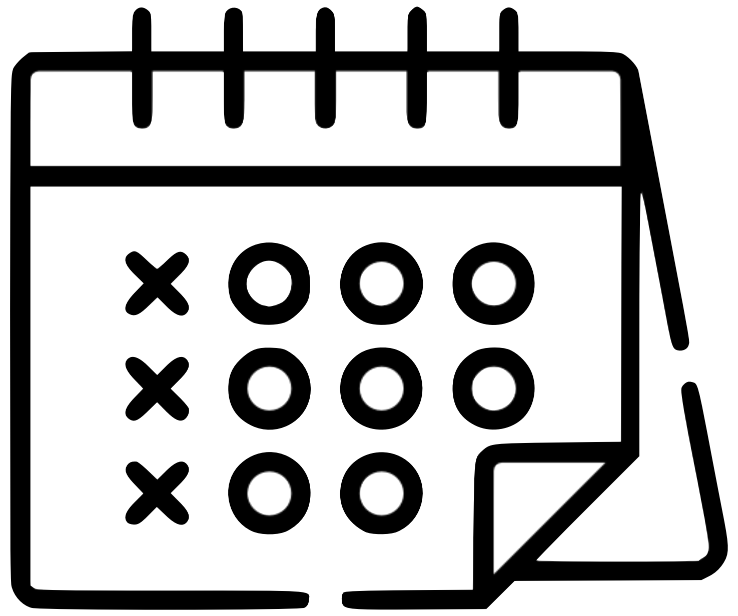
insert cost: $\mathcal{O}(\log N)$

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

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

DEFAULT

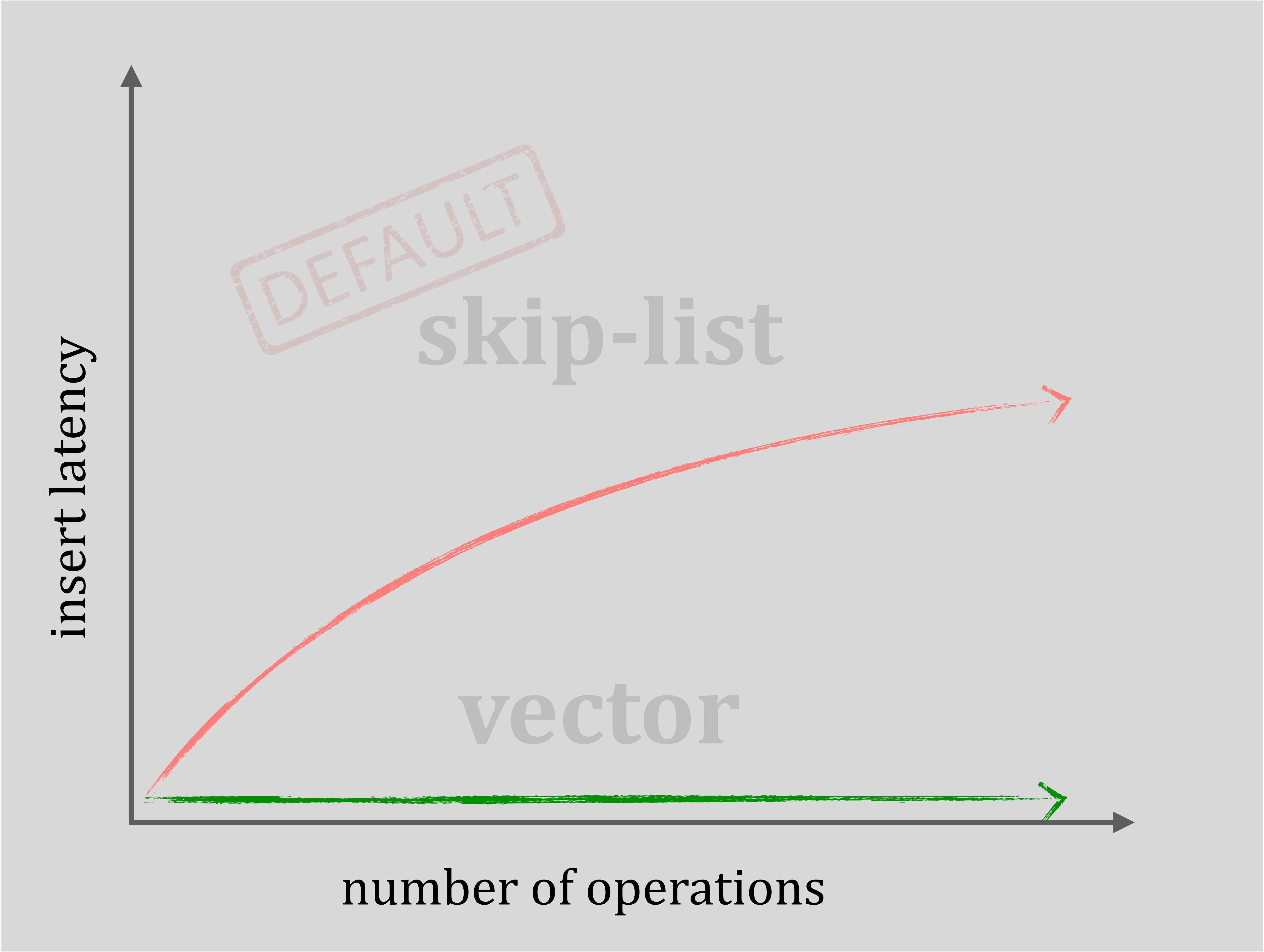
skip-list

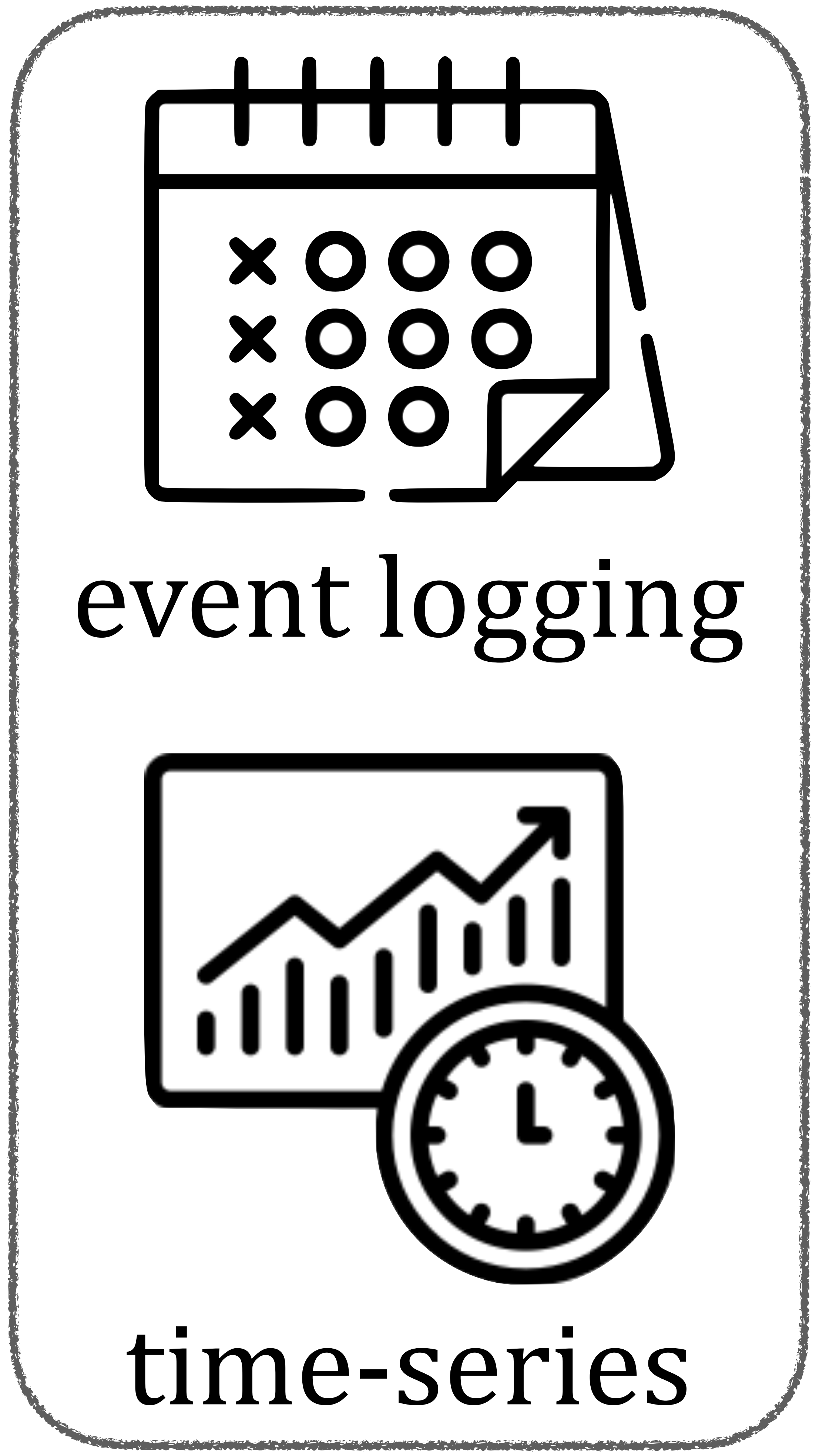


event logging

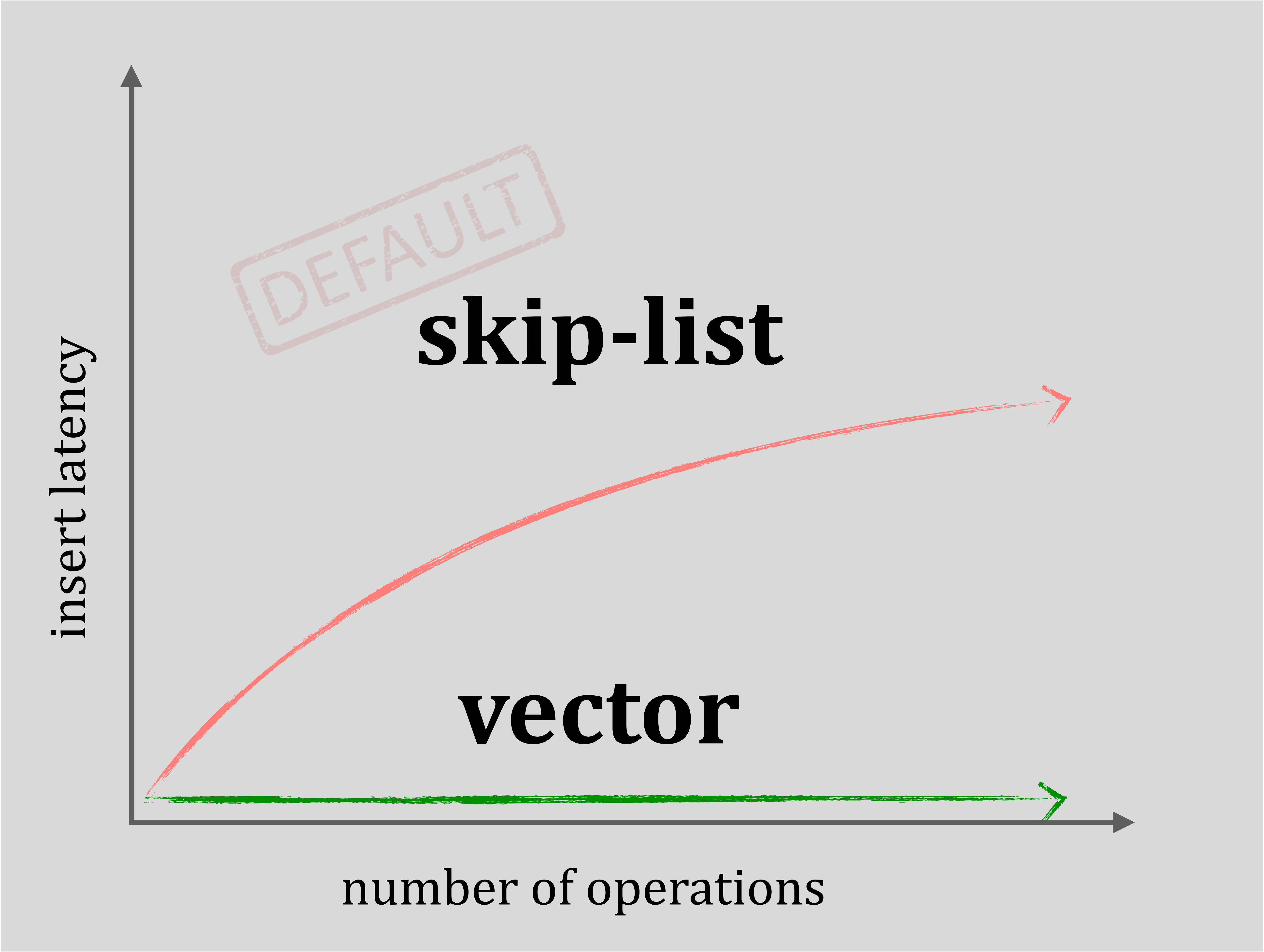
time-series

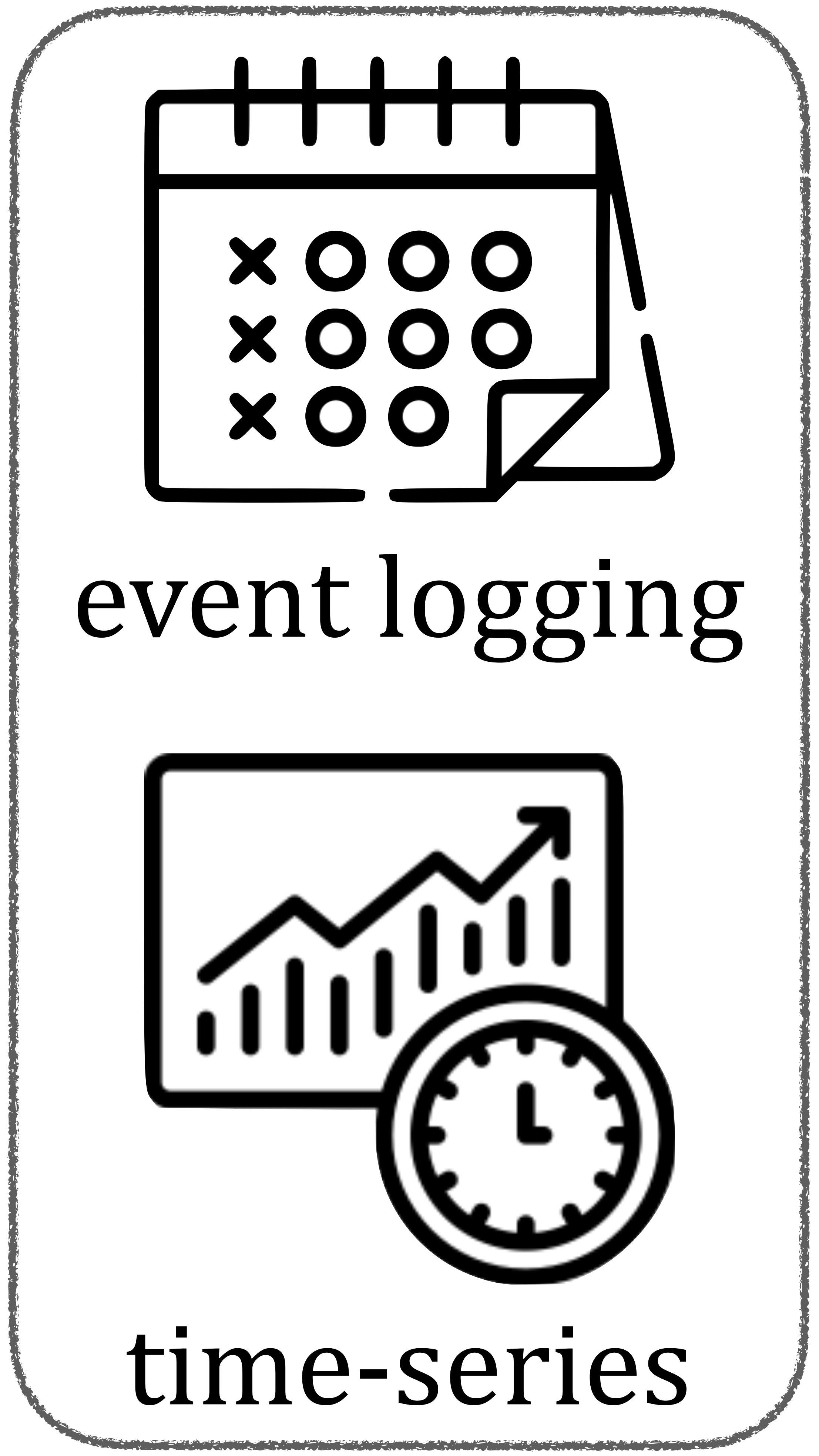
insert heavy





insert heavy

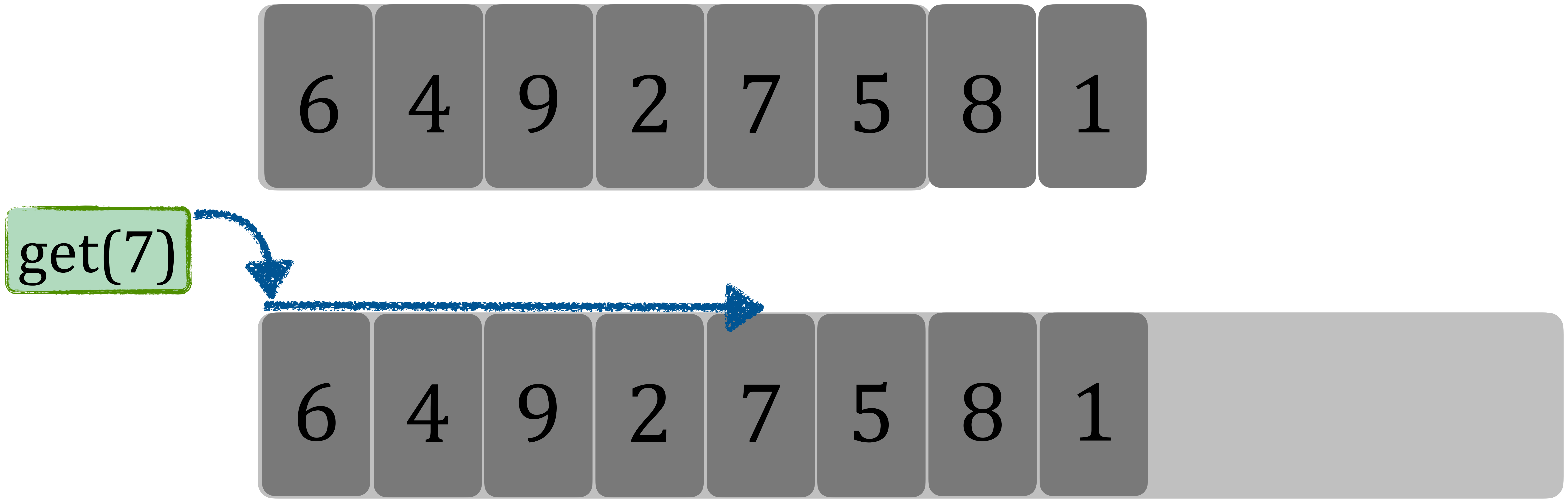




insert heavy

N : entries in buffer

Implementation: Vector



dynamic

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: $\mathcal{O}(N)$

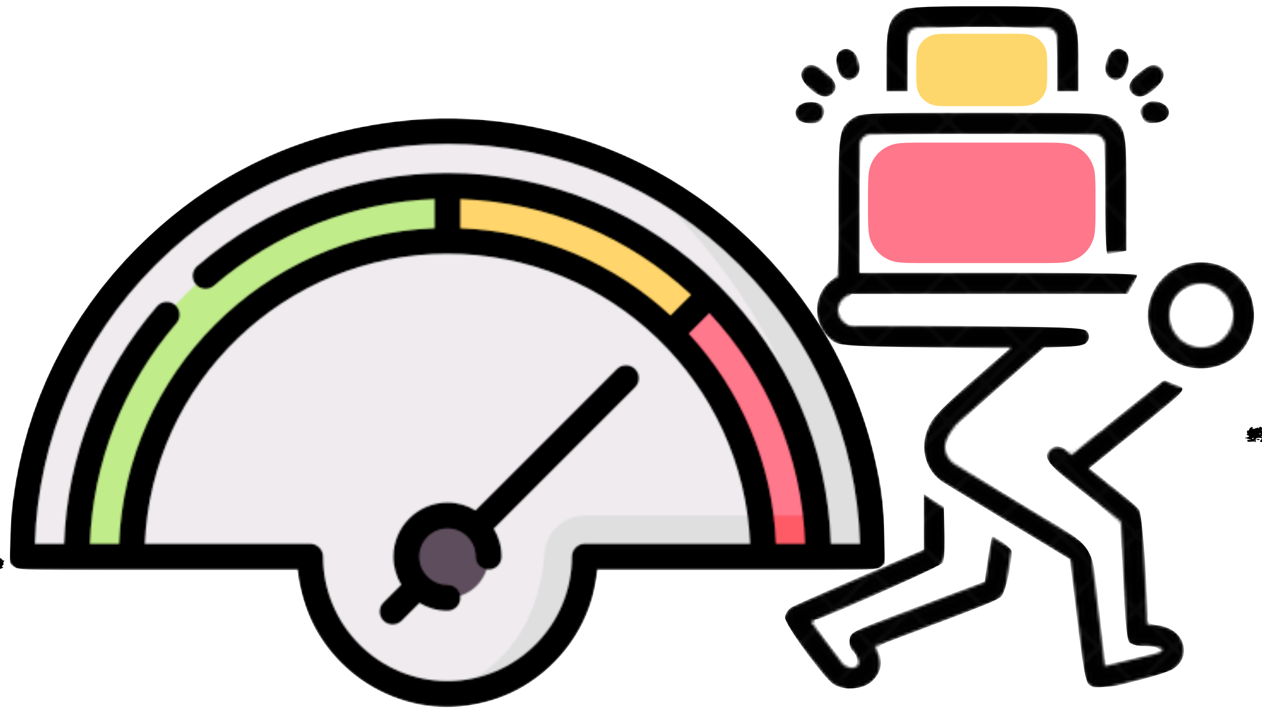
Ideal buffer implementation?

depends on a *workload composition*

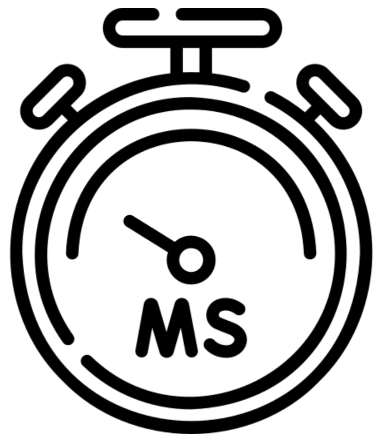
1. insert **heavy**
2. insert with **point** queries
3. insert with **range** queries
4. insert with **updates**

Research Focus

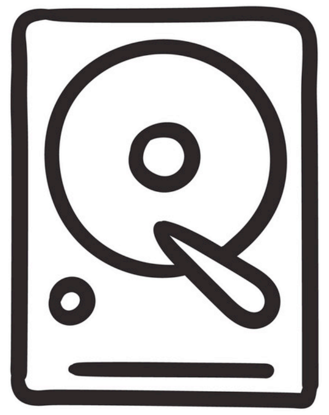
1. insert **heavy**
2. insert with **point** queries
3. insert with **range** queries
4. insert with **updates**



benchmark



latency



memory footprint

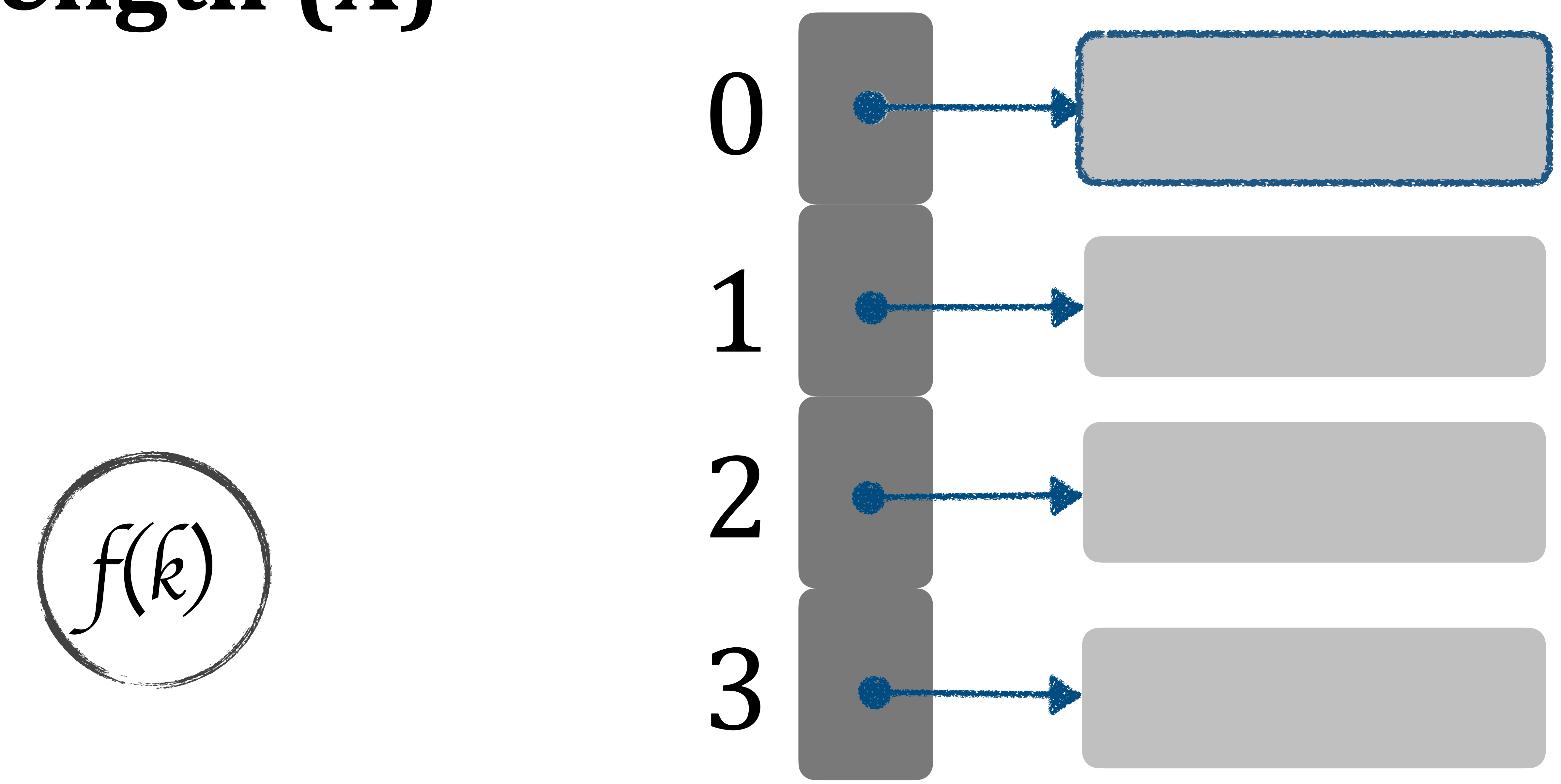
throughput

	vector	skip-list	hash-hybrid	trie
1	??			
2		??		
3			??	
4				??

N : entries in buffer
M : meta data

Implementation: Hash Hybrids

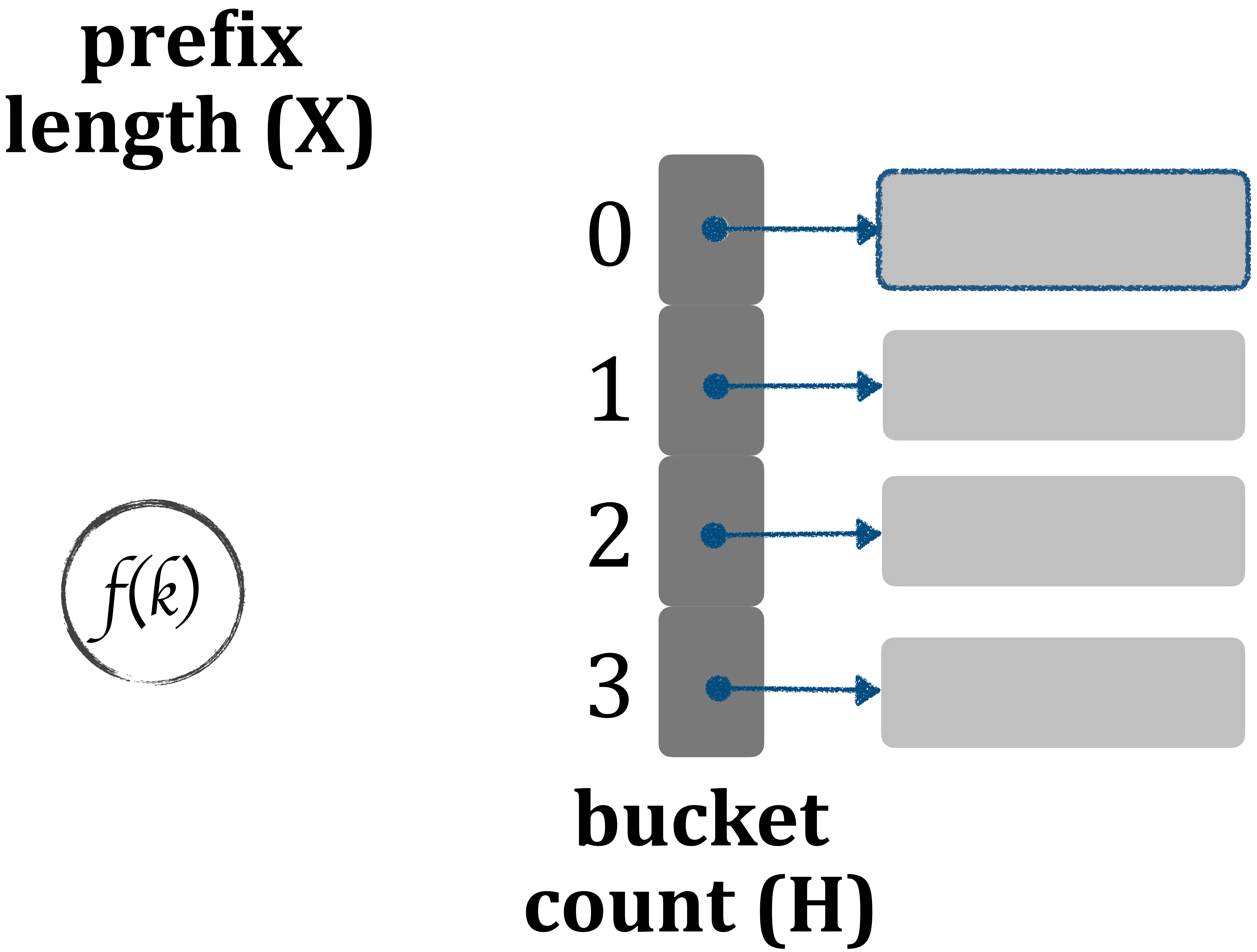
prefix length (X)



bucket count (H)

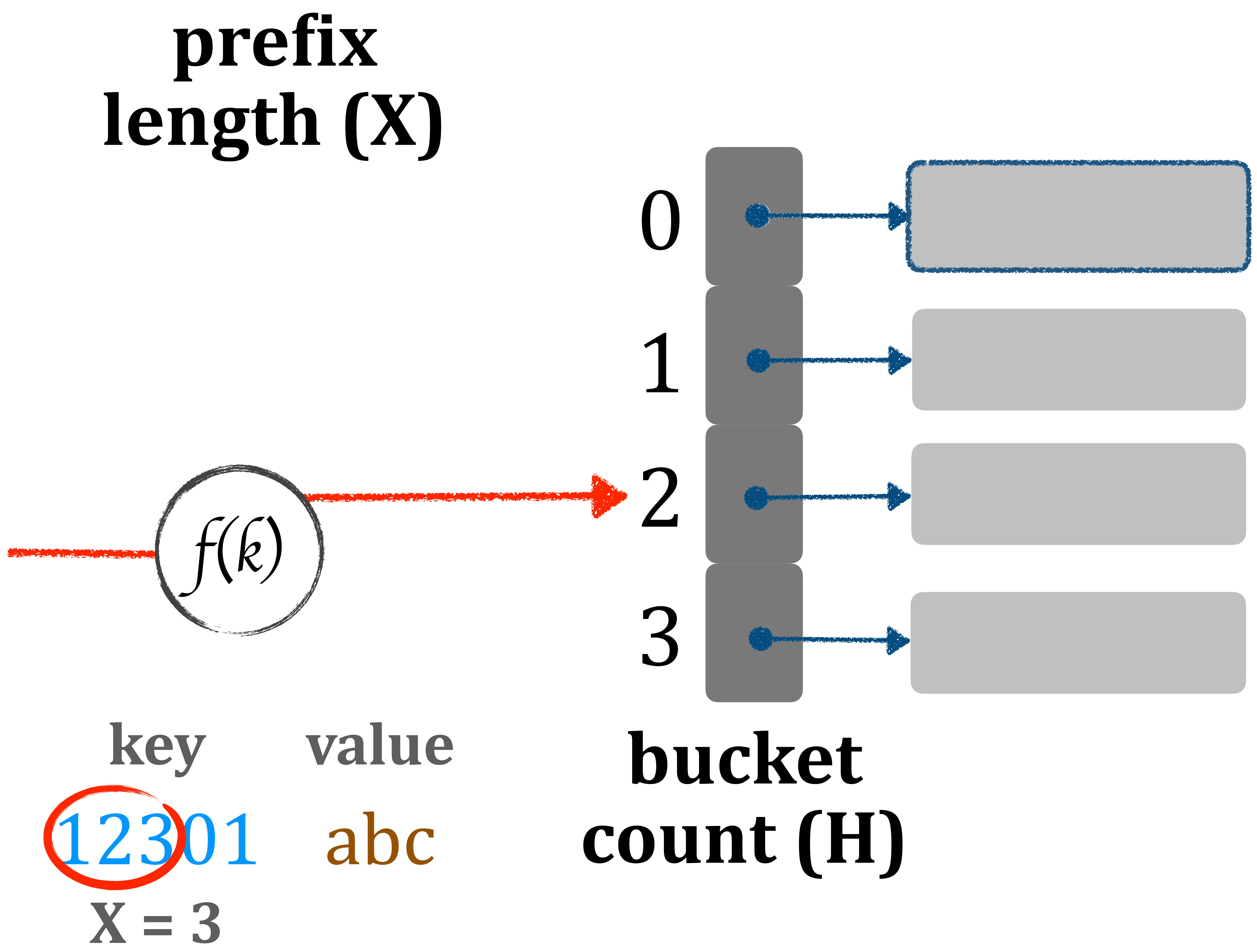
N : entries in buffer
M : meta data

Implementation: Hash Hybrids



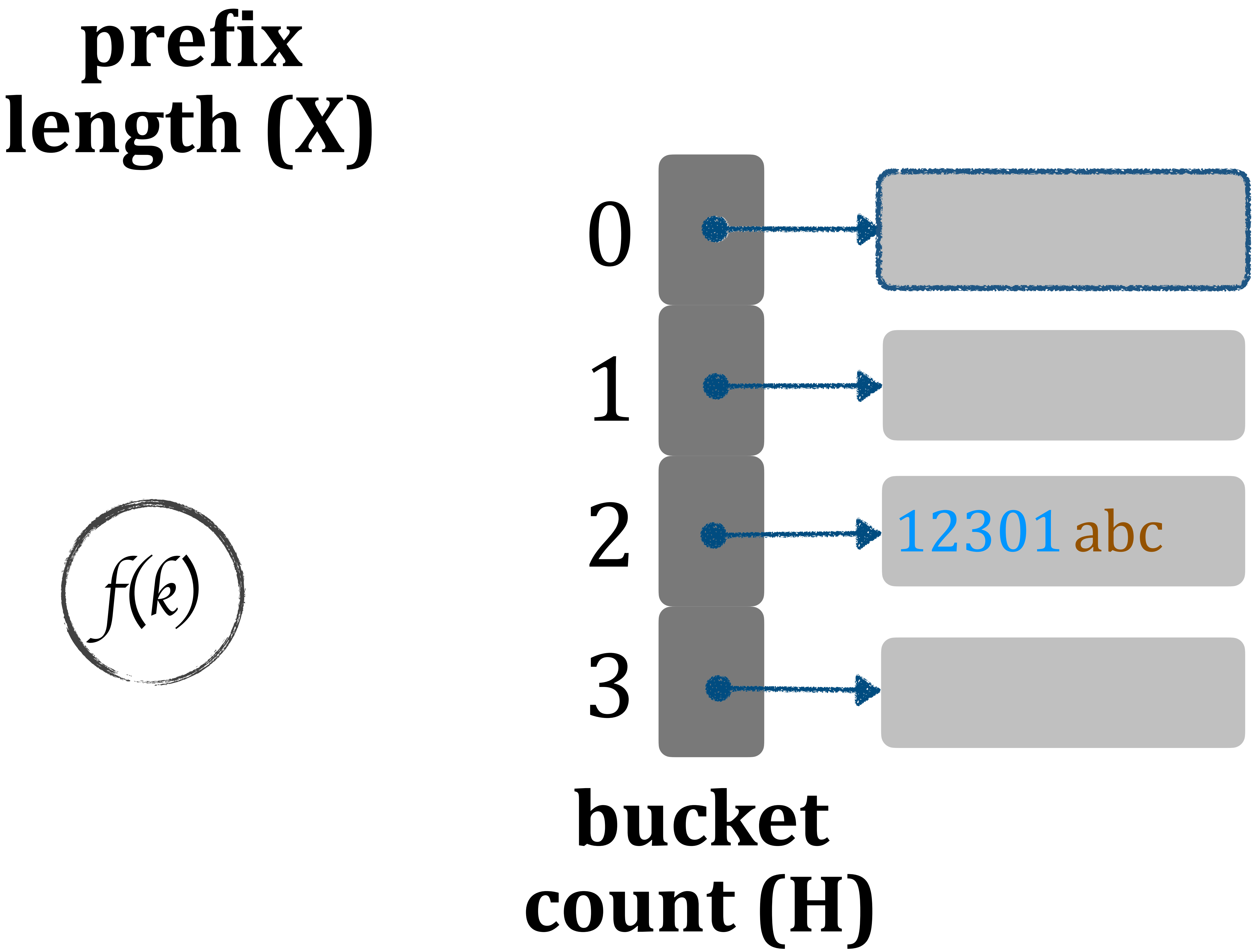
N : entries in buffer
M : meta data

Implementation: Hash Hybrids



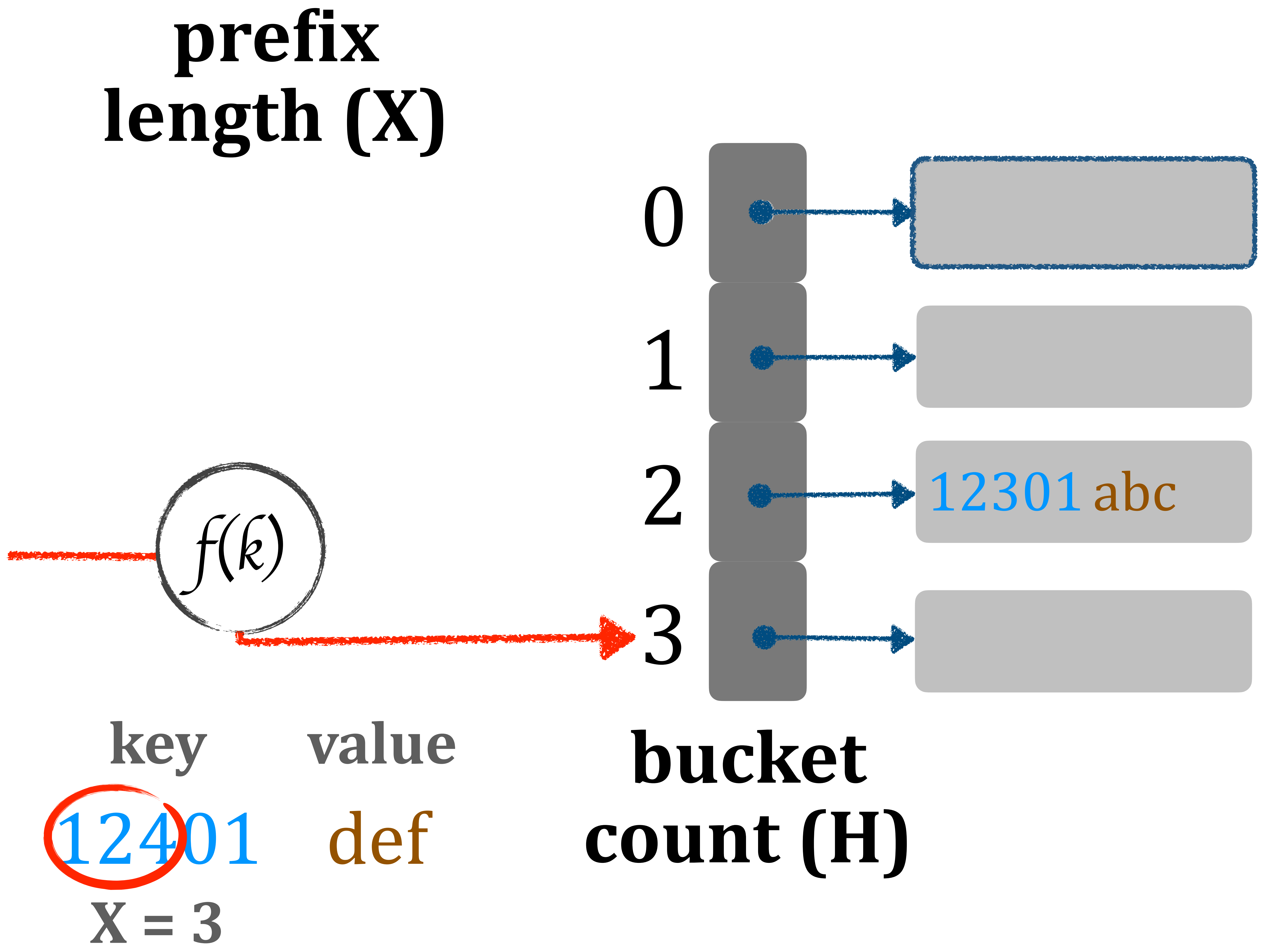
N : entries in buffer
M : meta data

Implementation: Hash Hybrids



N : entries in buffer
M : meta data

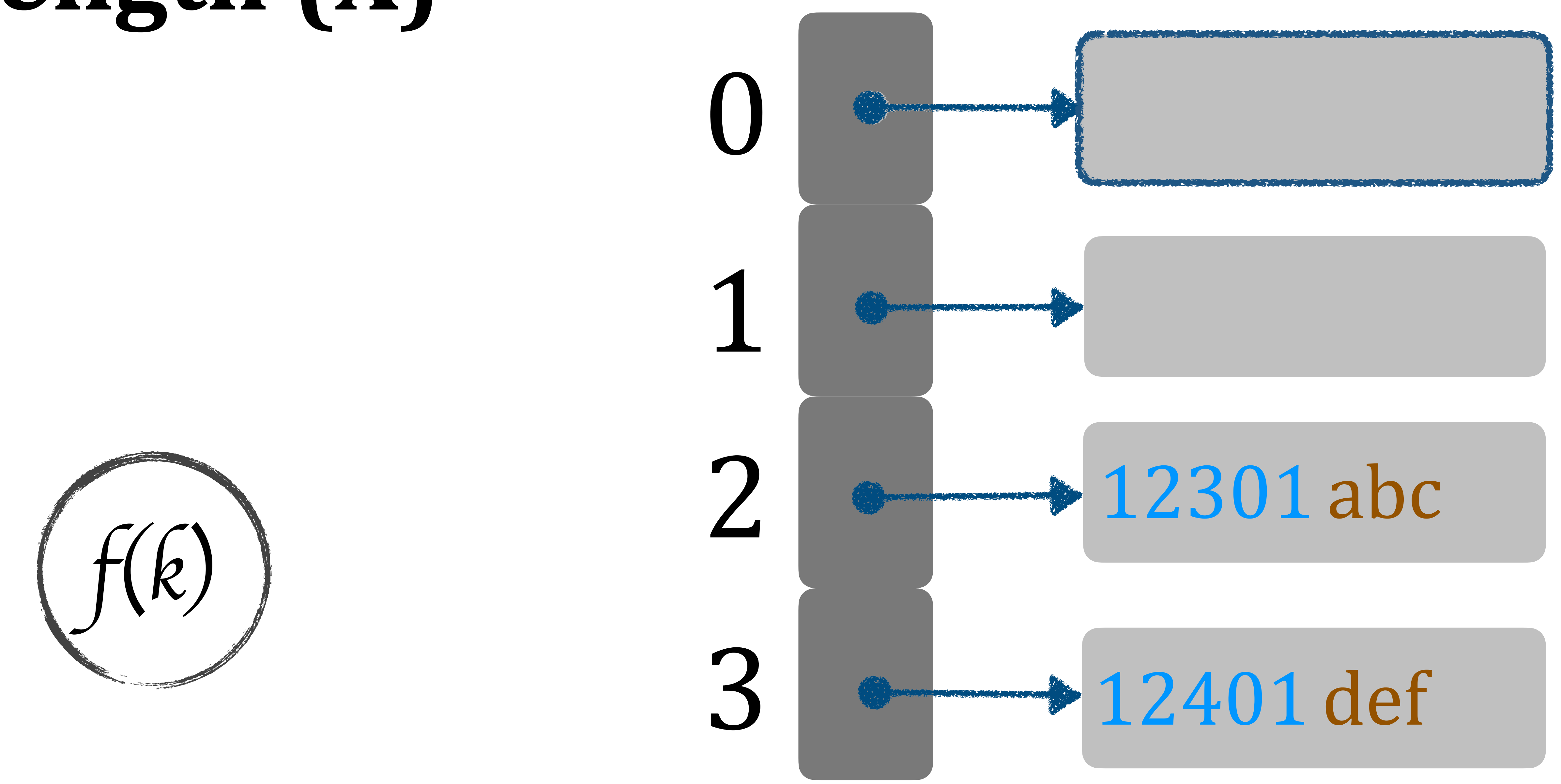
Implementation: Hash Hybrids



N : entries in buffer
M : meta data

Implementation: Hash Hybrids

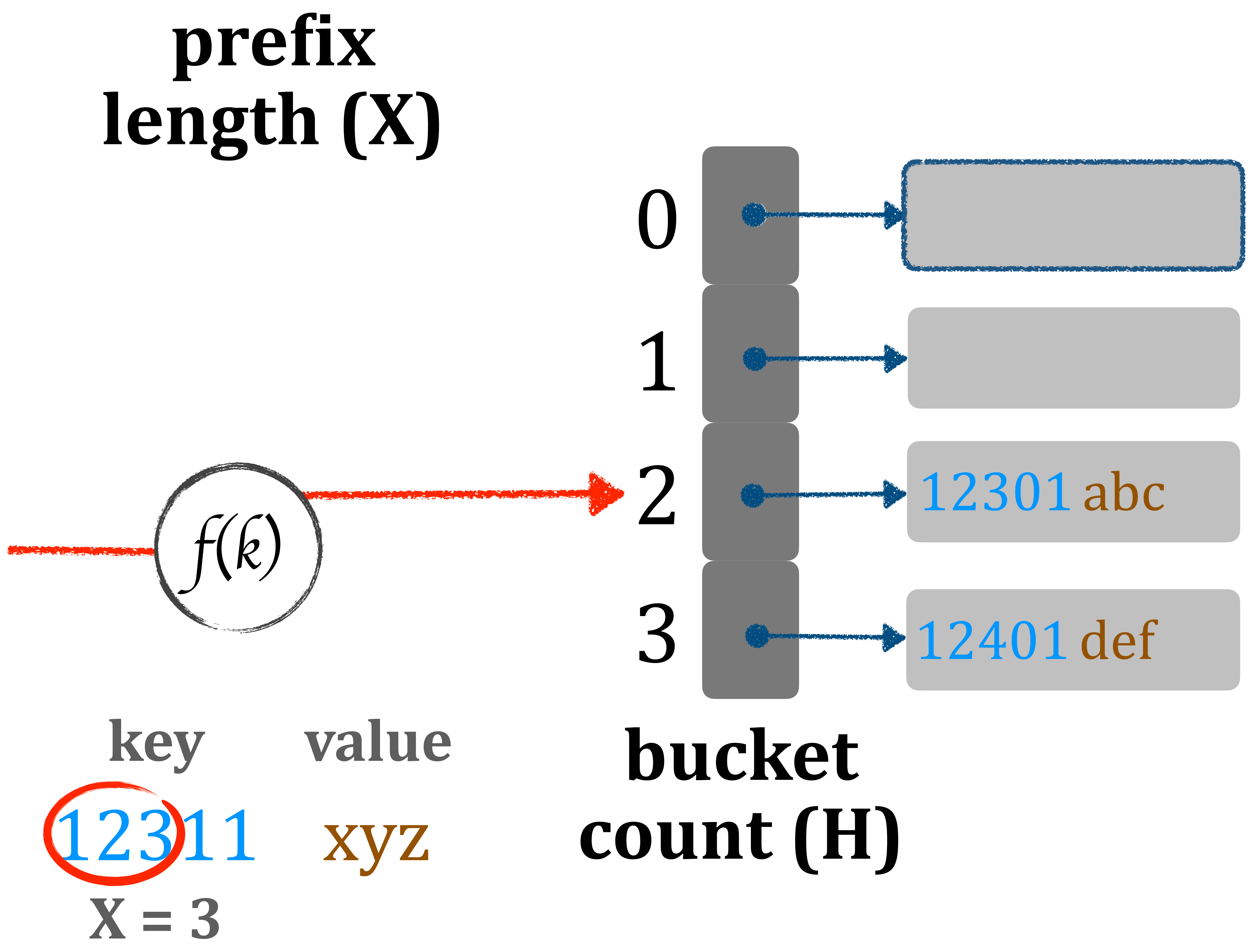
prefix length (X)



bucket count (H)

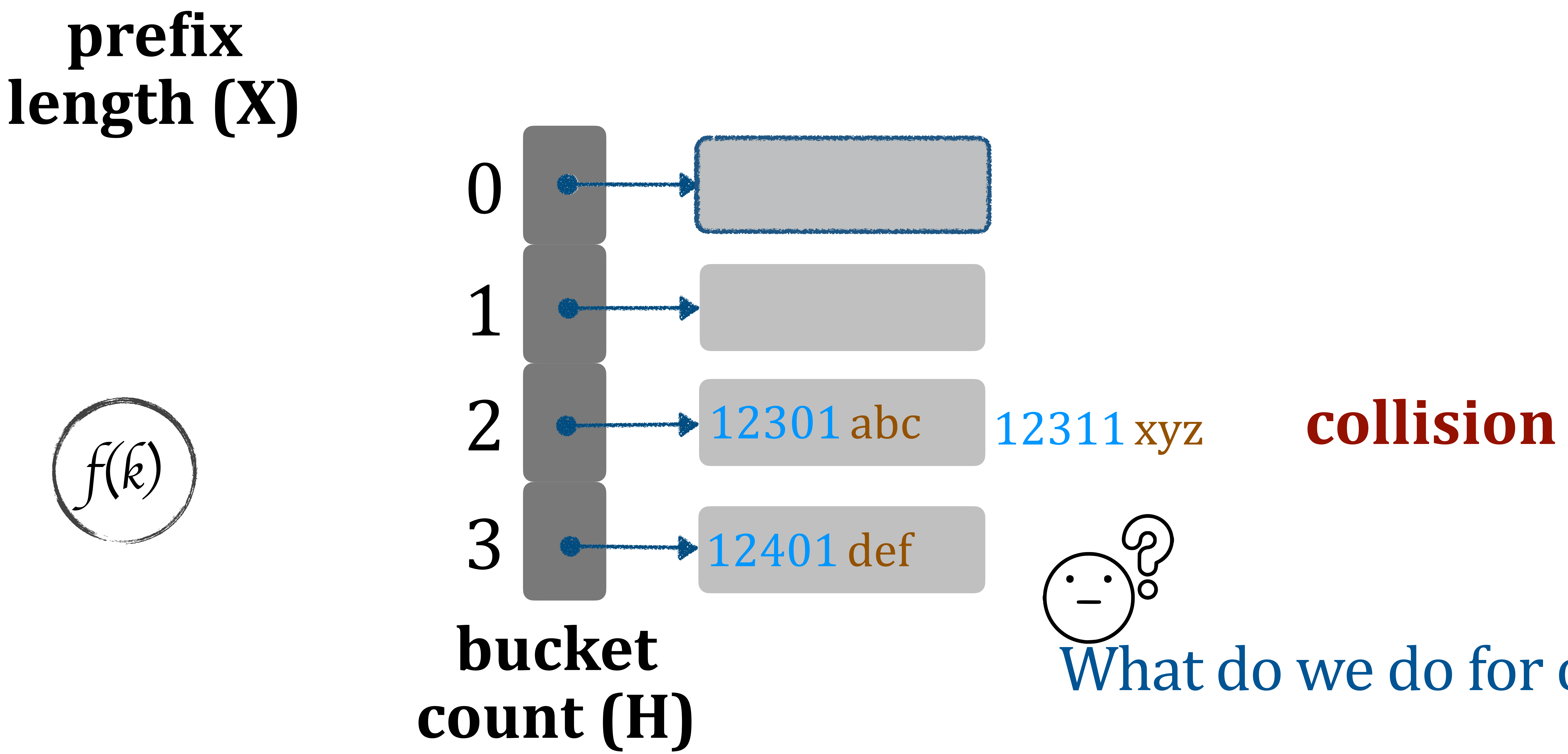
N : entries in buffer
M : meta data

Implementation: Hash Hybrids



N : entries in buffer
M : meta data

Implementation: Hash Hybrids



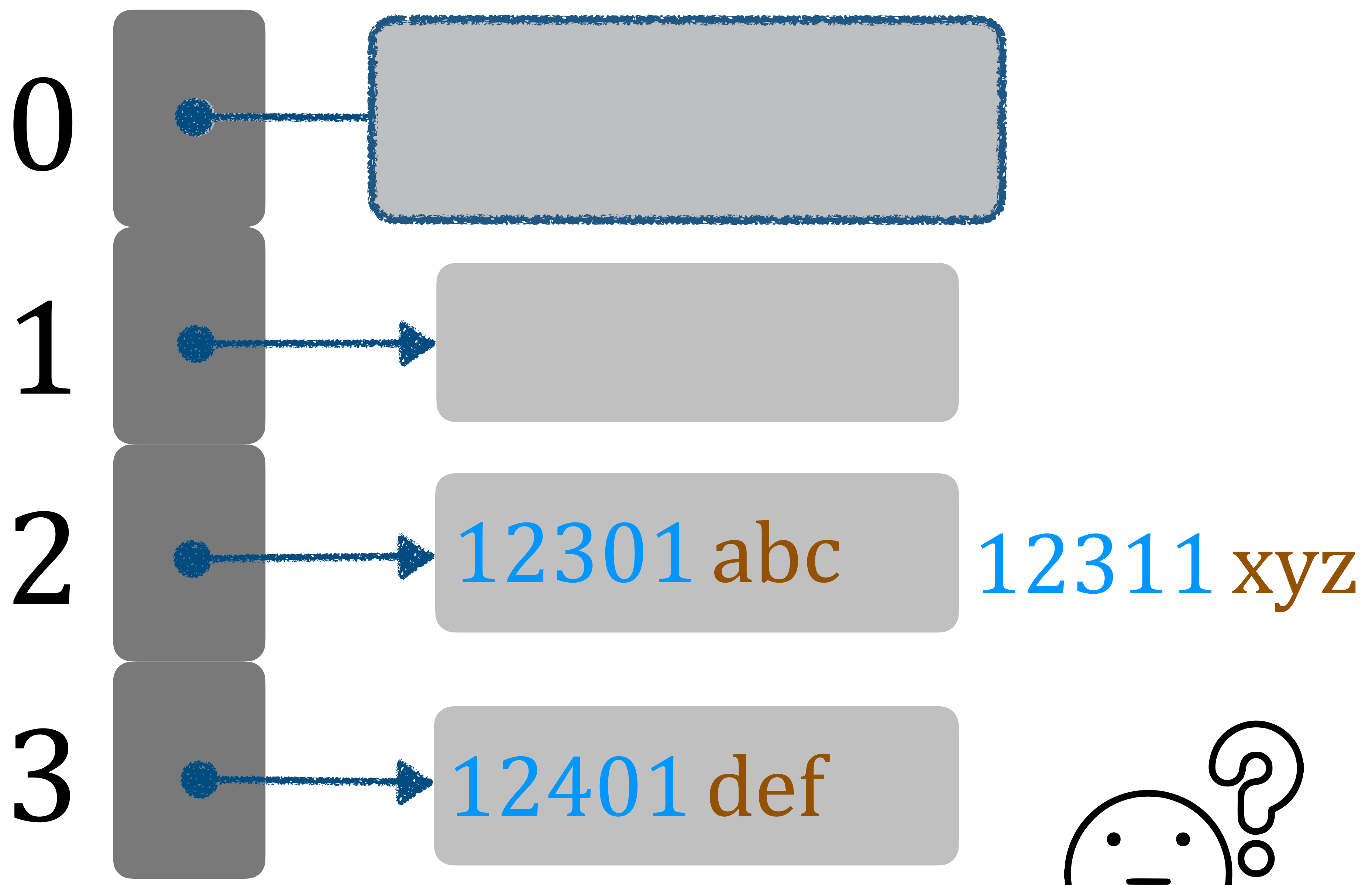
What do we do for overflow?

N : entries in buffer
M : meta data

Implementation: Hash Hybrids

prefix length (X)

$f(k)$



bucket count (H)

linked-list
skip-list
vector

collision

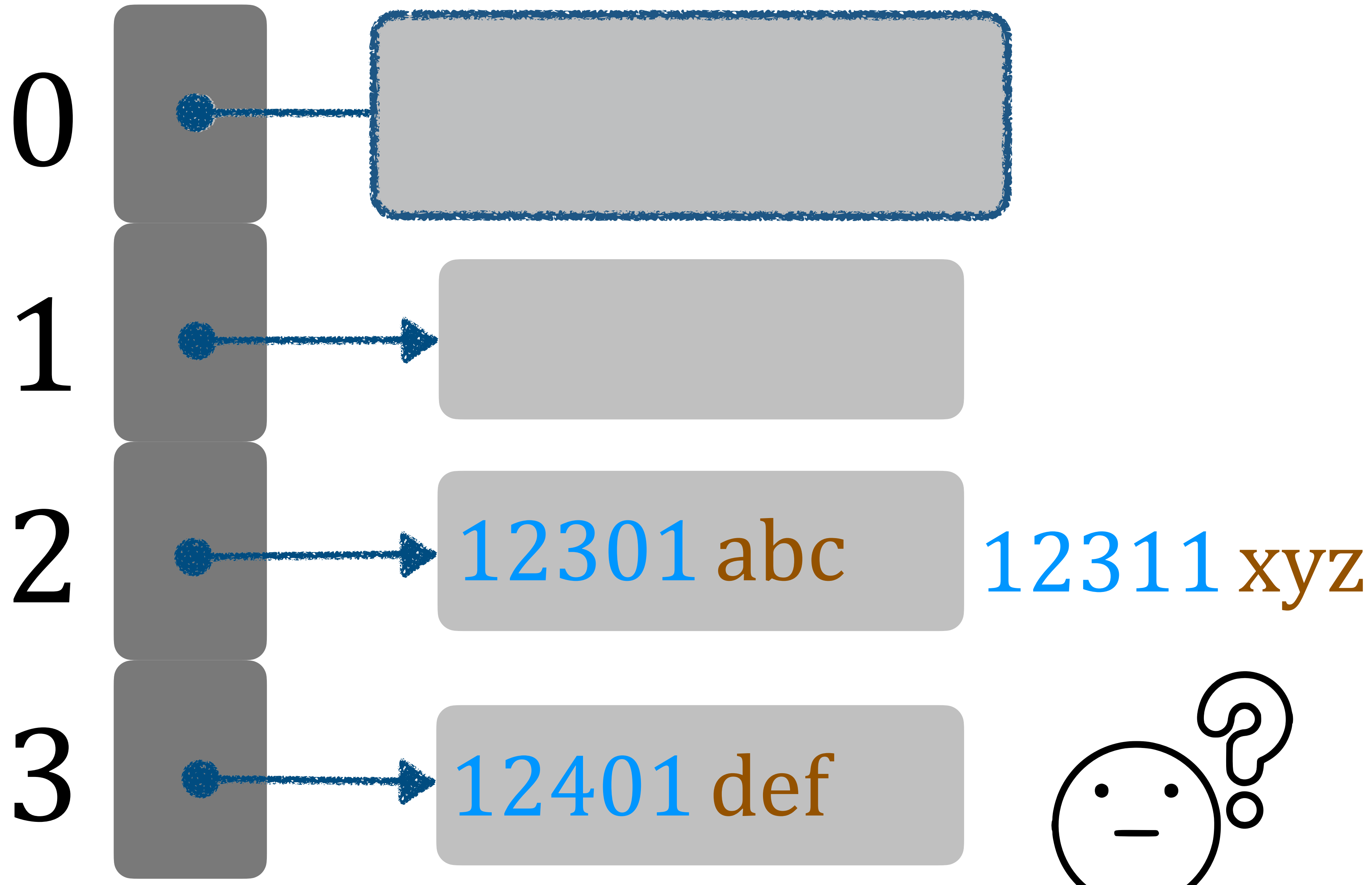
What do we do for overflow?

N : entries in buffer
M : meta data

Implementation: Hash Hybrids

prefix length (X)

$f(k)$



bucket count (H)

linked-list
skip-list
vector

collision

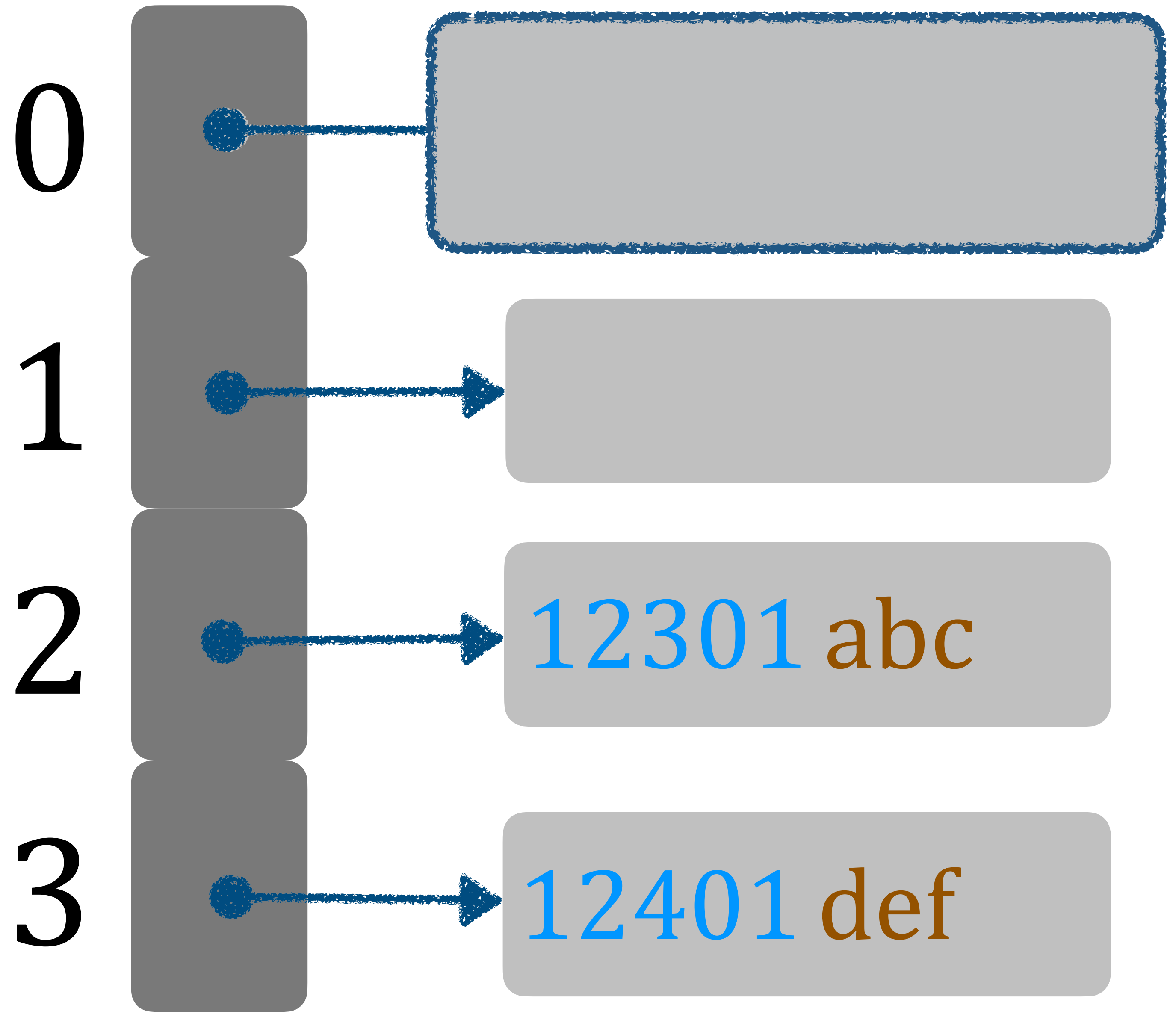
What if we don’t store prefix?

N : entries in buffer
M : meta data

Implementation: Hash Hybrids

prefix length (X)

$f(k)$



bucket count (H)

linked-list
skip-list
vector

- 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: $\mathcal{O}(\log N/H)$

N : entries in buffer
M : meta data

Analyze: Hash Hybrids

prefix length (X)

bucket count (H)

any data structure

$f(k_{i\text{-}\mathcal{N}}) \rightarrow k_i v, \ldots, k_N v$

X = 0, H > 0
or
X = key size,
H = 1

hash-table

$f(k_i) \rightarrow kv$

$f(k_{\mathcal{N}}) \rightarrow kv$

X = key size,
H = N

N : entries in buffer
M : meta data

Analyze: Hash Hybrids

prefix length (X)

bucket count (H)

What could be the possible values for X and H?

any data structure

$f(k_{i\text{-}N})$ → $k_i v, \ldots, k_N v$

X = 0, H > 0
or
X = key size,
H = 1

hash-table

$f(k_i)$ → kv

$f(k_N)$ → kv

X = key size,
H = N

X ??
H ??

b is a finite set of prefixes $\{b_1, \ldots, b_m\}$ for range queries

same prefix for start and end key

N : entries in buffer
M : meta data

Analyze: Hash Hybrids

prefix length (X)

bucket count (H)

What could be the possible values for X and H?

any data structure

$f(k_{i\text{-}N}) \rightarrow k_i v, \ldots, k_N v$

X = 0, H > 0
or
X = key size,
H = 1

hash-table

$f(k_i) \rightarrow kv$

$f(k_N) \rightarrow kv$

X = key size,
H = N

X = len(b_1)
H > len(b)

b is a finite set of prefixes $\{b_1, \ldots, b_m\}$ for range queries

same prefix for start and end key

N : entries in buffer
M : meta data

Analyze: Hash Hybrids

prefix length (X)

bucket count (H)

What could be the possible H value?

any data structure

$f(k_{i\text{-}N}) \rightarrow k_i v, \ldots, k_N v$

$X = 0, H > 0$
or
$X =$ key size,
$H = 1$

hash hybrid$_{(a)}$

$X = \text{len}(b_1)$
$H > \text{len}(b)$

b is a finite set of prefixes $\{b_1, \ldots, b_m\}$ for range queries

$X = \text{len}(b_1)$,
$H <=$??

hash-table

$f(k_i) \rightarrow kv$

$f(k_N) \rightarrow kv$

$X =$ key size,
$H = N$

s is selectivity of range query

same prefix for start and end key

N : entries in buffer
M : meta data

Analyze: Hash Hybrids

prefix length (X)

bucket count (H)

What could be the possible H value?

any data structure

$f(k_{i\text{-}N}) \rightarrow k_i v, \ldots, k_N v$

X = 0, H > 0
or
X = key size,
H = 1

hash hybrid(a)

$X = len(b_1)$
$H > len(b)$

b is a finite set of prefixes $\{b_1, \ldots, b_m\}$ for range queries

hash-table

$f(k_i) \rightarrow kv$

$f(k_N) \rightarrow kv$

X = key size,
H = N

$X = len(b_1)$,
$H \le size(buffer)/(N \times s)$

s is selectivity of range query

same prefix for start and end key

Brandeis UNIVERSITY

N : entries in buffer
M : meta data

Analyze: Hash Hybrids

prefix length (X)

bucket count (H)

any data structure

$f(k_{i\text{-}N}) \rightarrow k_i v, \ldots, k_N v$

X = 0, H > 0
or
X = key size,
H = 1

hash hybrid$_{(a)}$

X = len(b_1)
H > len(b)

b is a finite set of prefixes $\{b_1, \ldots, b_m\}$ for range queries

hash hybrid$_{(b)}$

X = len(b_1),
H <= size(buffer)/(N x s)

s is selectivity of range query

b is a finite set of prefixes $\{b_1, \ldots, b_m\}$ for range queries

hash-table

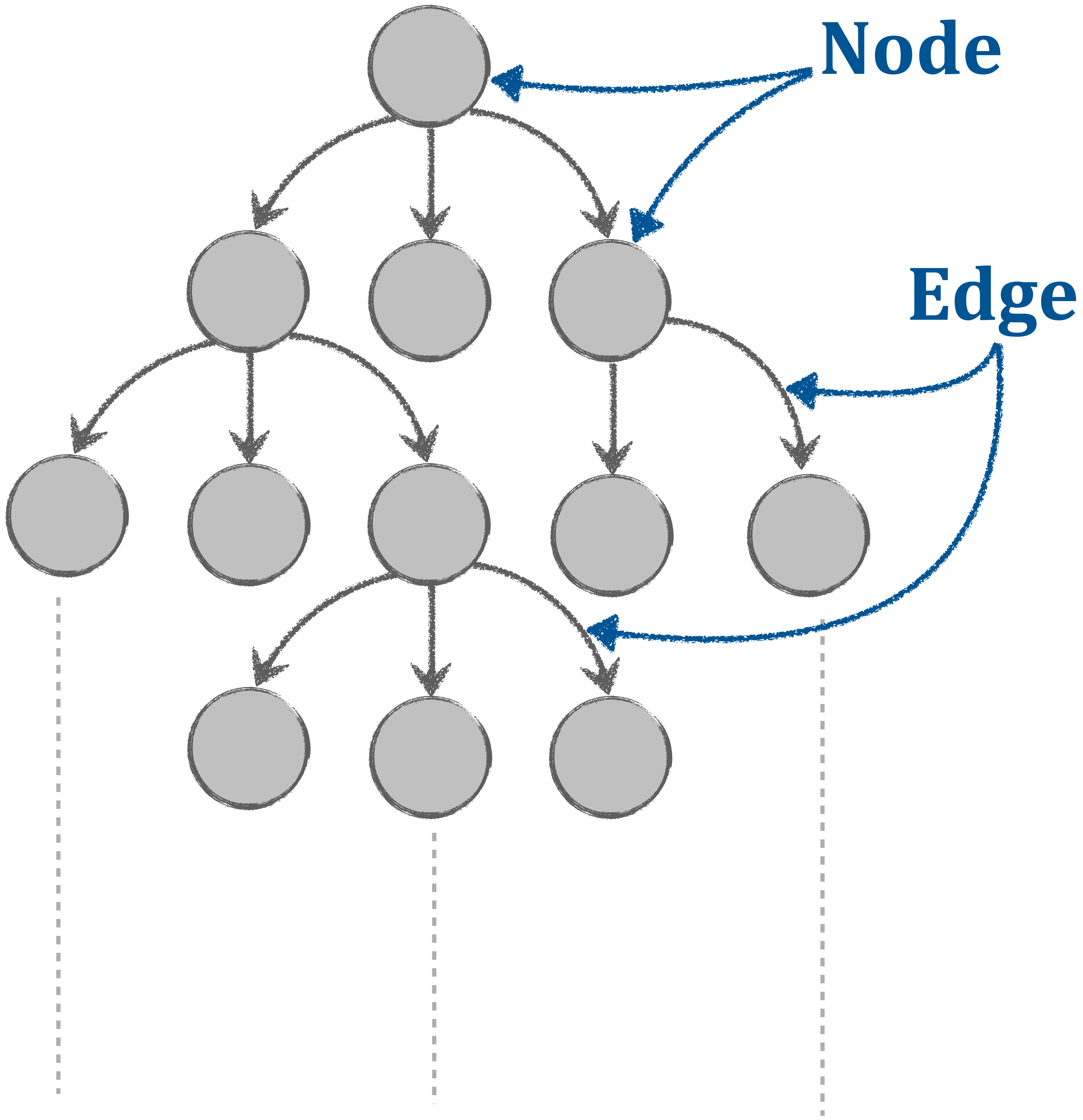
$f(k_i) \rightarrow$ kv

$f(k_N) \rightarrow$ kv

X = key size,
H = N

Brandeis
UNIVERSITY

Implementation: Trie



K : max size of key
M : meta data
N : entries in buffer

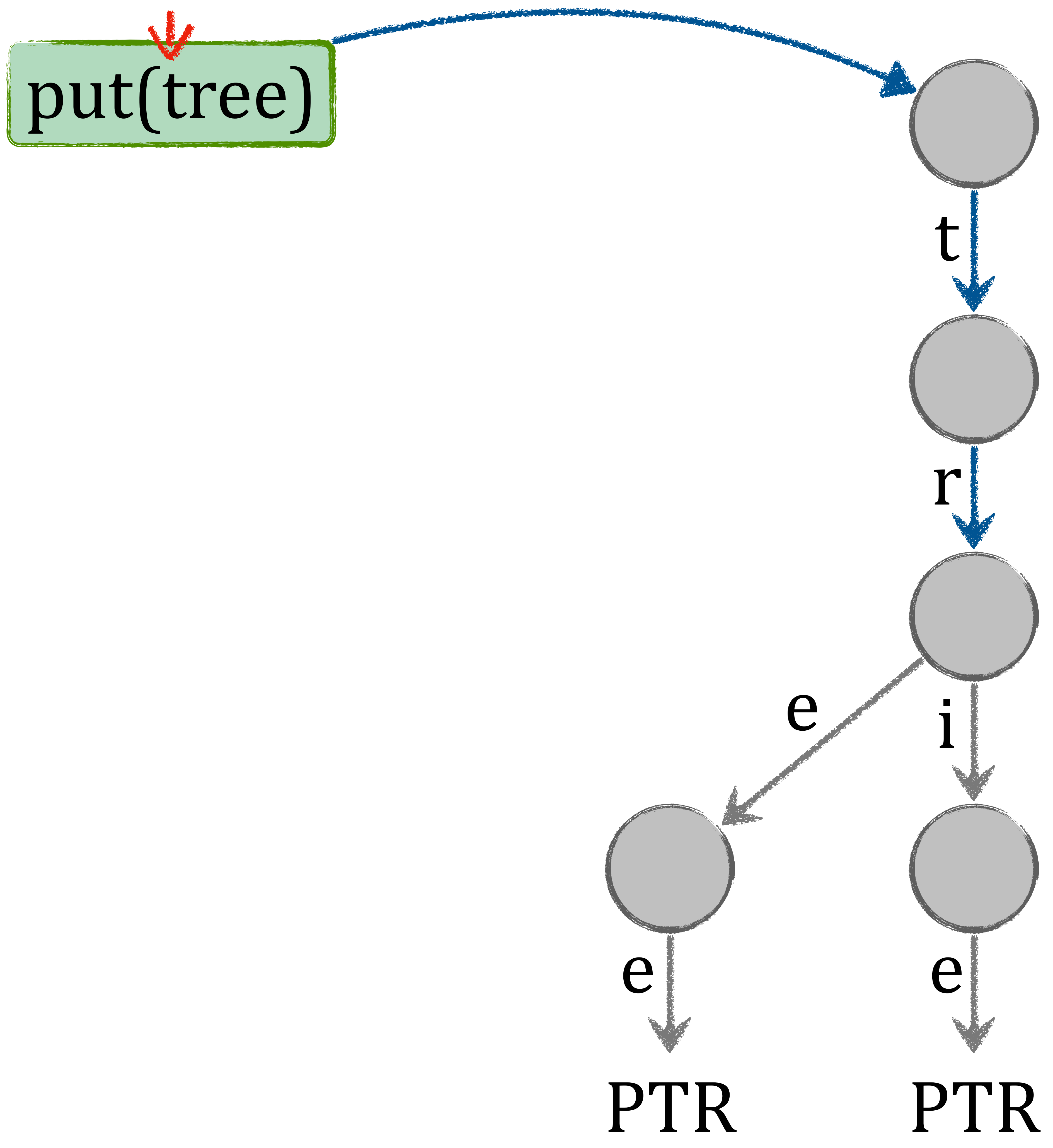
Put Implementation: Trie

put(trie)

pointer to value

K : max size of key
M : meta data
N : entries in buffer

Put Implementation: Trie

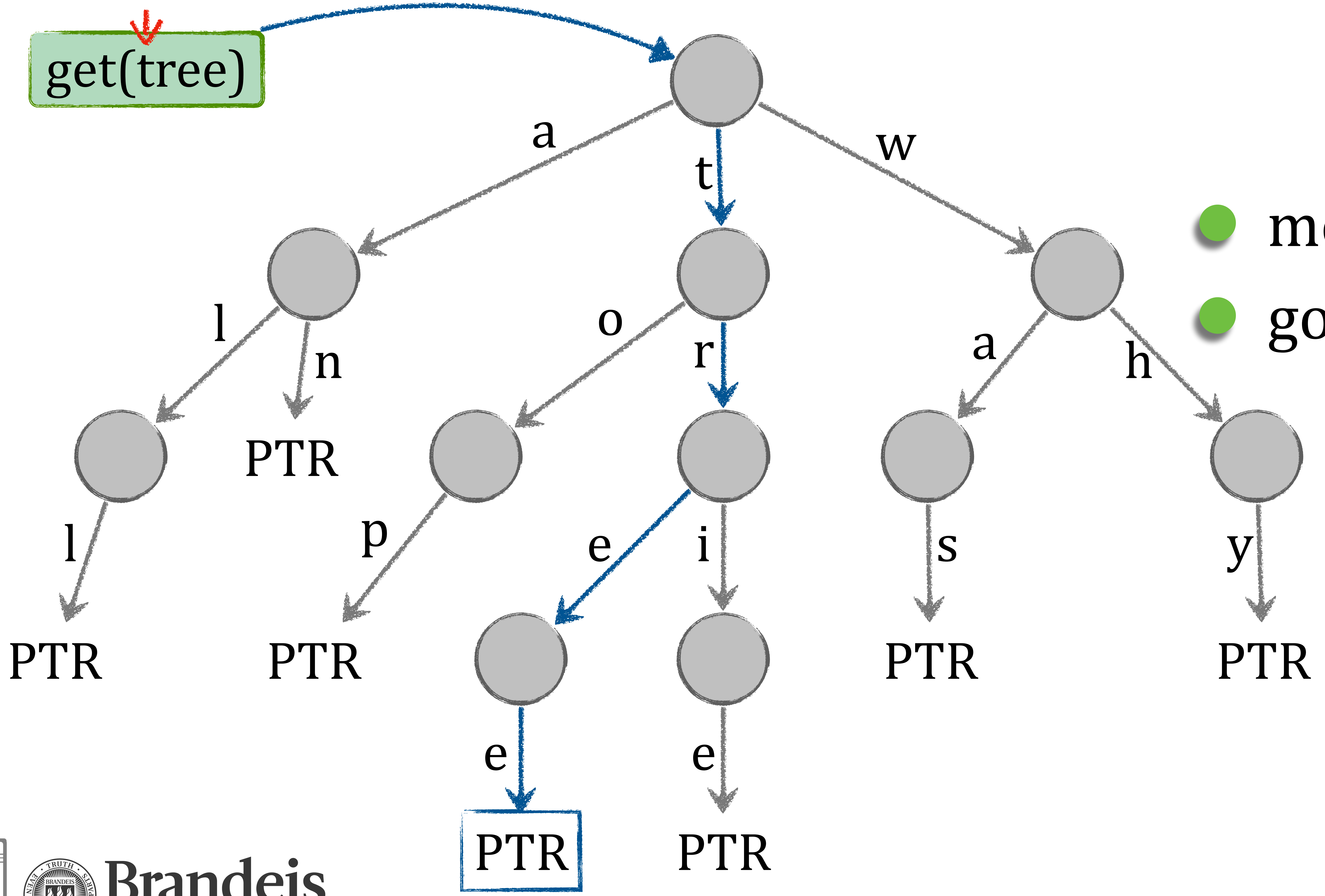


What's the difference?

- more space optimized

K : max size of key
M : meta data
N : entries in buffer

Get Implementation: Trie

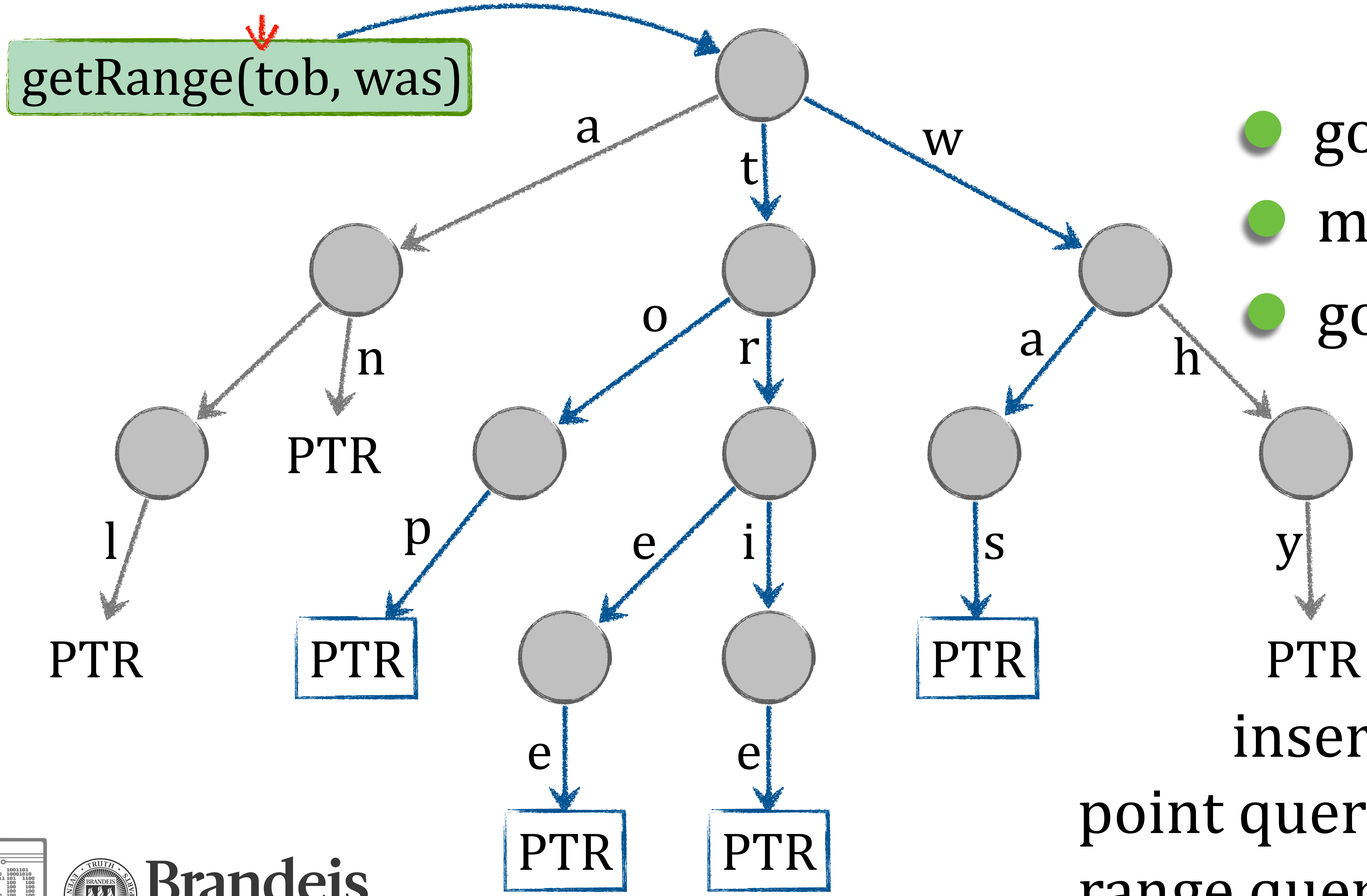


- more space optimized
- good for point queries

Brandeis
UNIVERSITY

K : max size of key
M : meta data
N : entries in buffer

GetRange Implementation: Trie



- good for range queries
- more space optimized
- good for point queries

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

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

range query cost: $\mathcal{O}(K \times N \times s)$

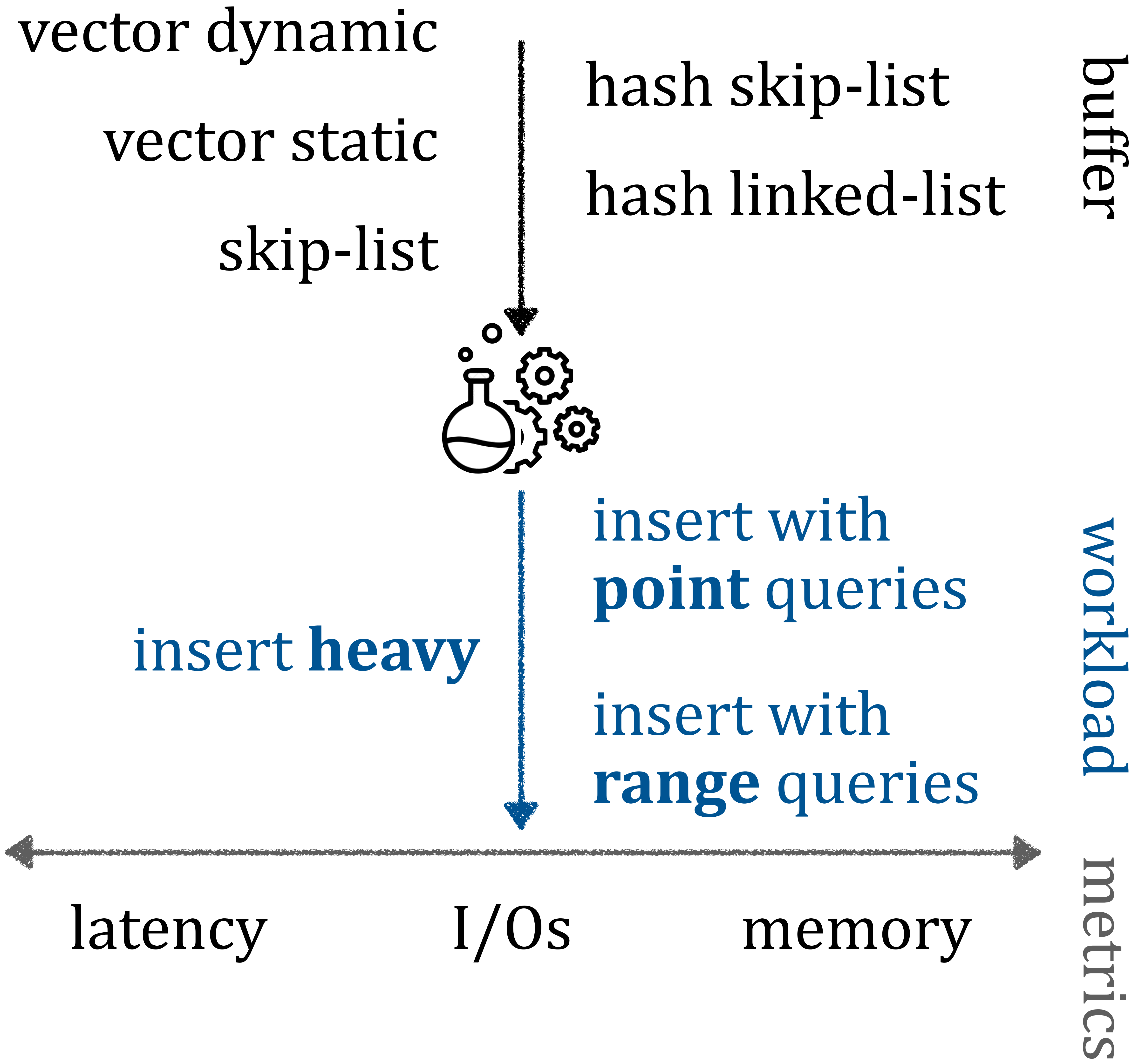
Brandeis UNIVERSITY

Experiments

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

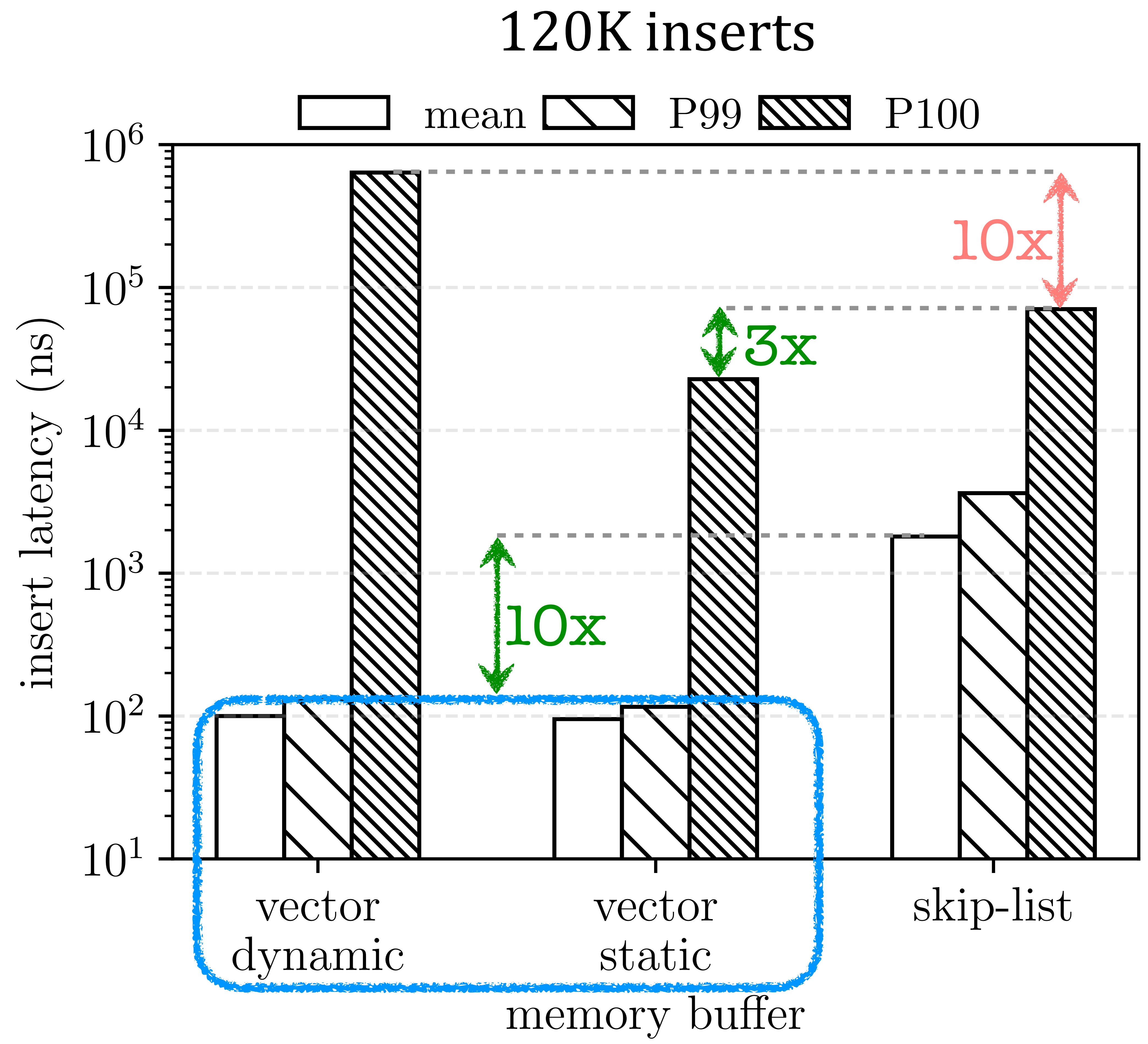
RocksDB v9.0.0

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 heavy* w/l



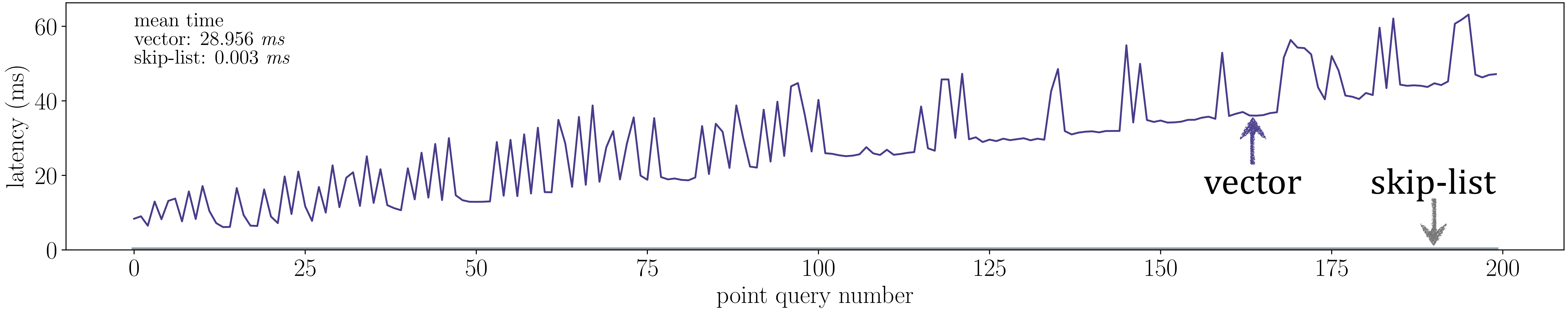
insert heavy

Brandeis UNIVERSITY

Evaluation

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

140K inserts, 200 interleaved point queries



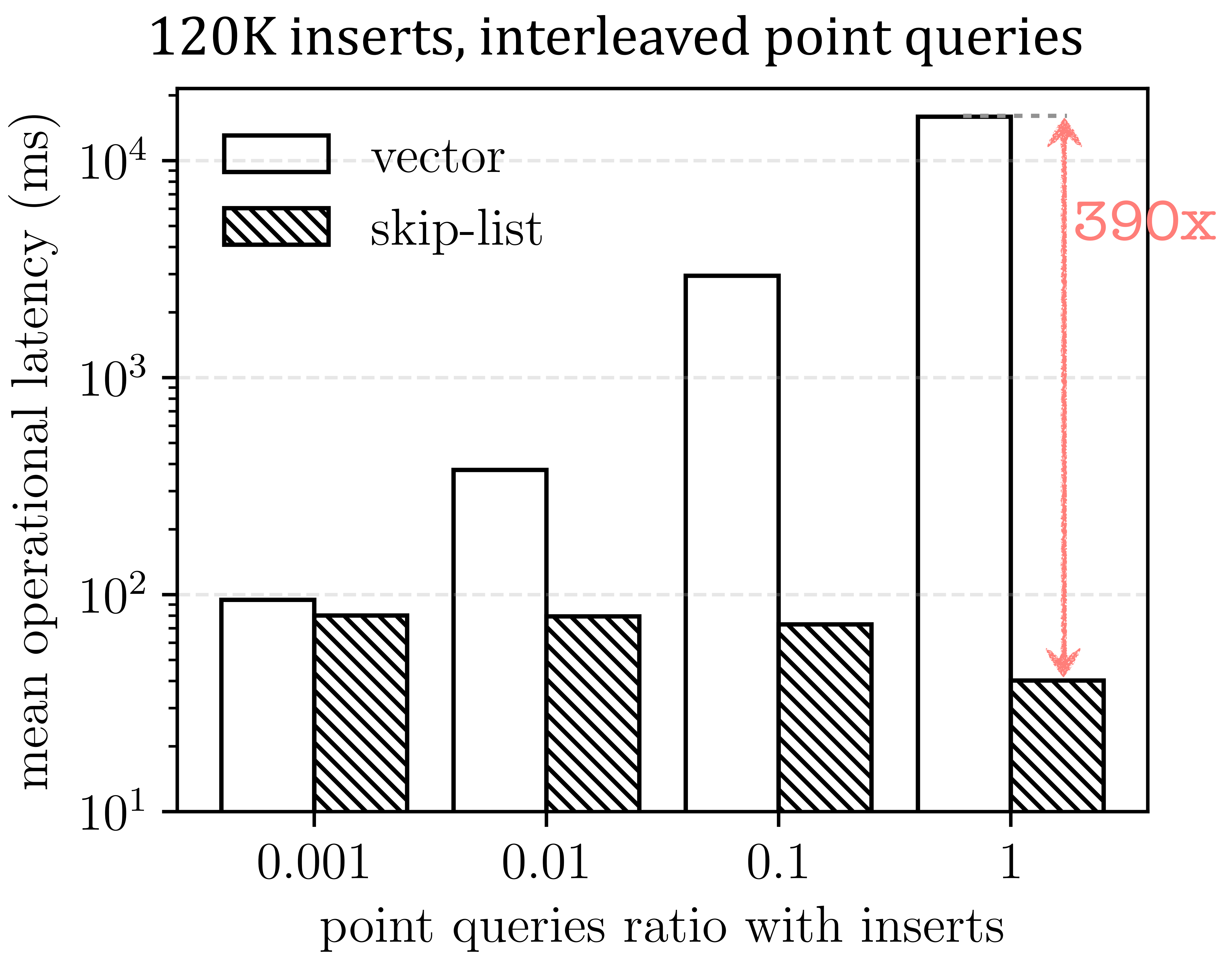
insert with point queries

Brandeis UNIVERSITY

Evaluation

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

vector is worst choice for w/ls with point queries



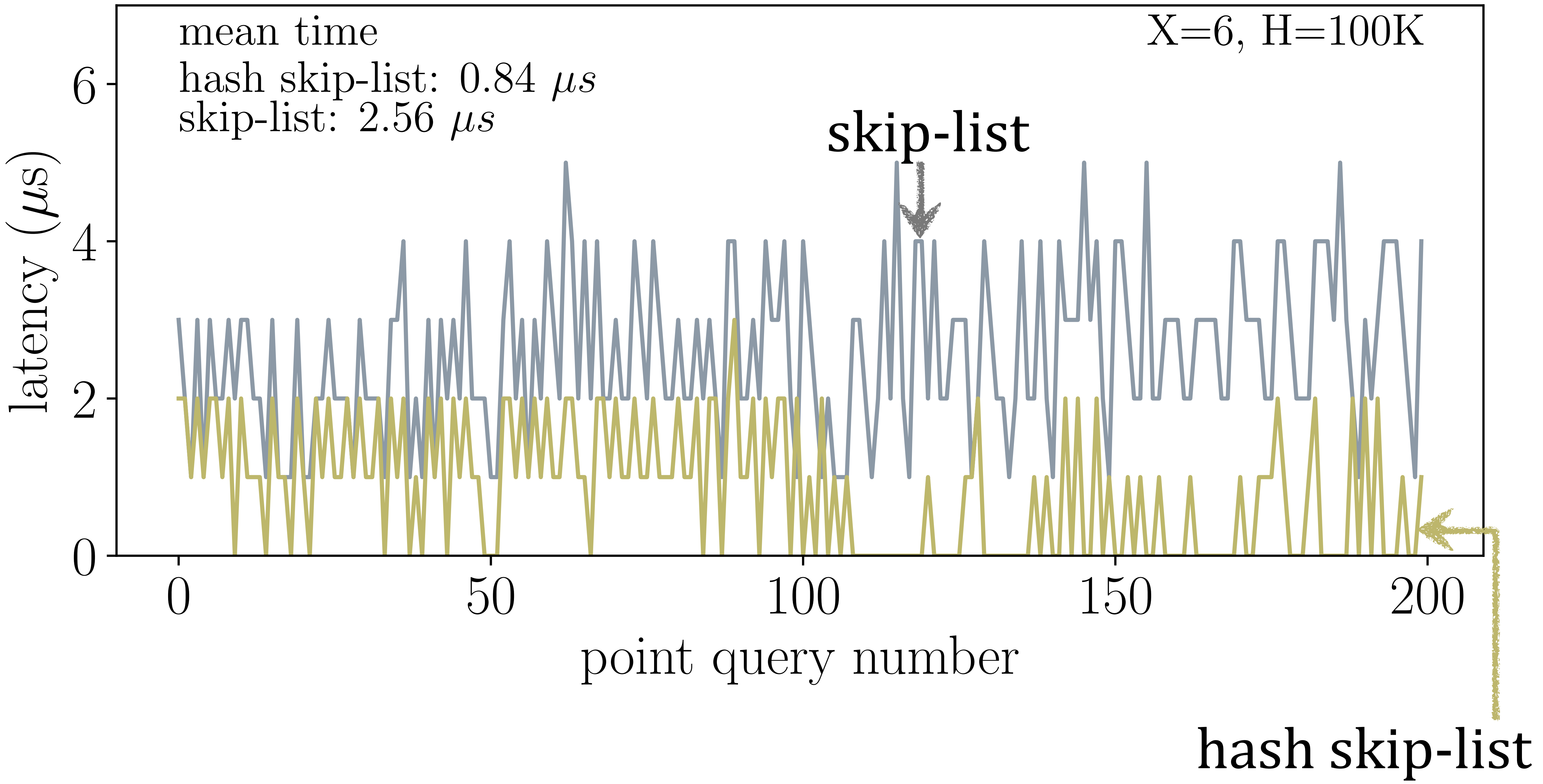
insert with point queries

Evaluation

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

vector is worst choice for w/ls with point queries

140K inserts, 200 interleaved point queries



insert with point queries

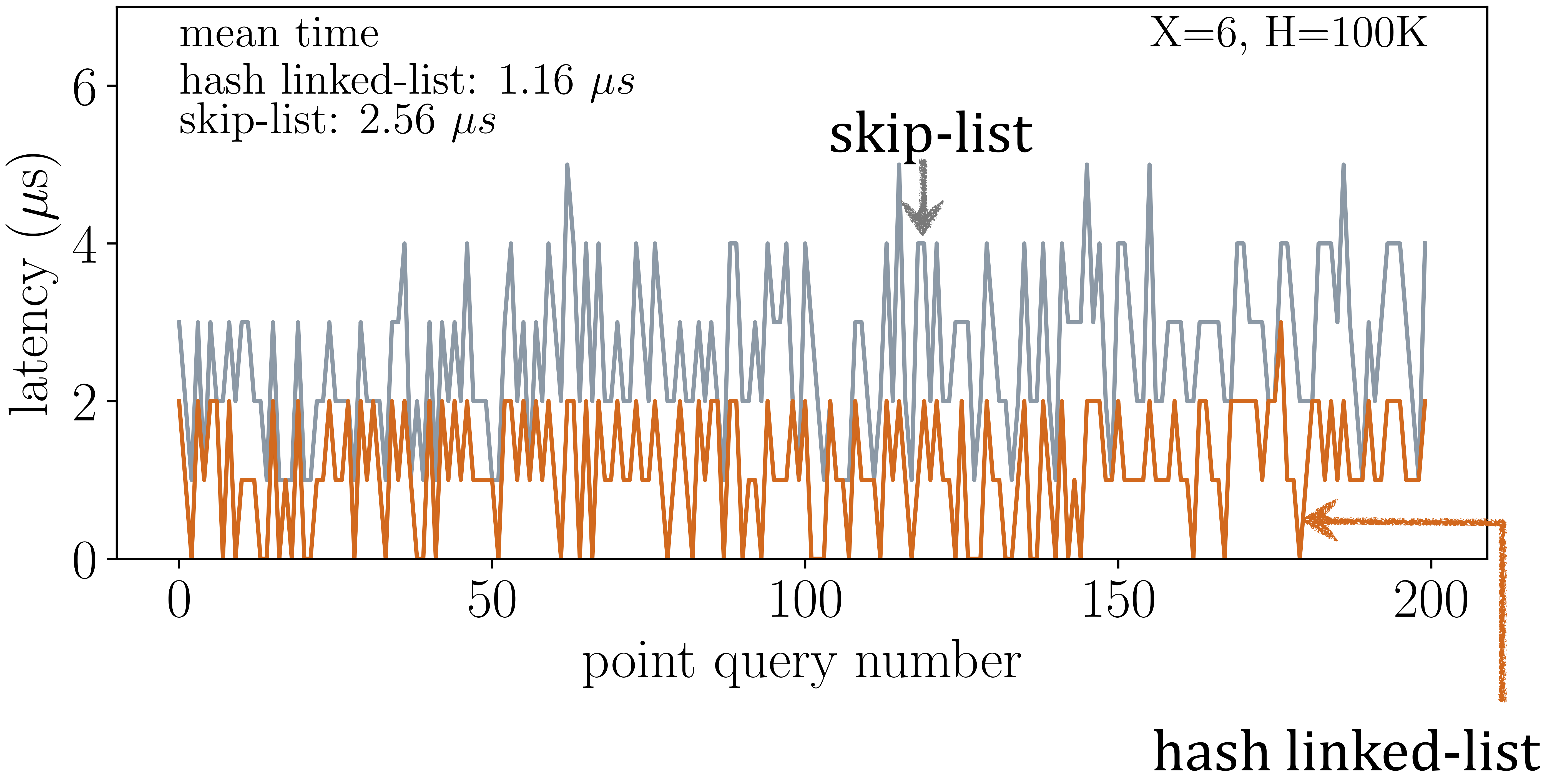
Evaluation

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

vector is worst choice for w/ls with point queries

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

140K inserts, 200 interleaved point queries



insert with point queries

Evaluation

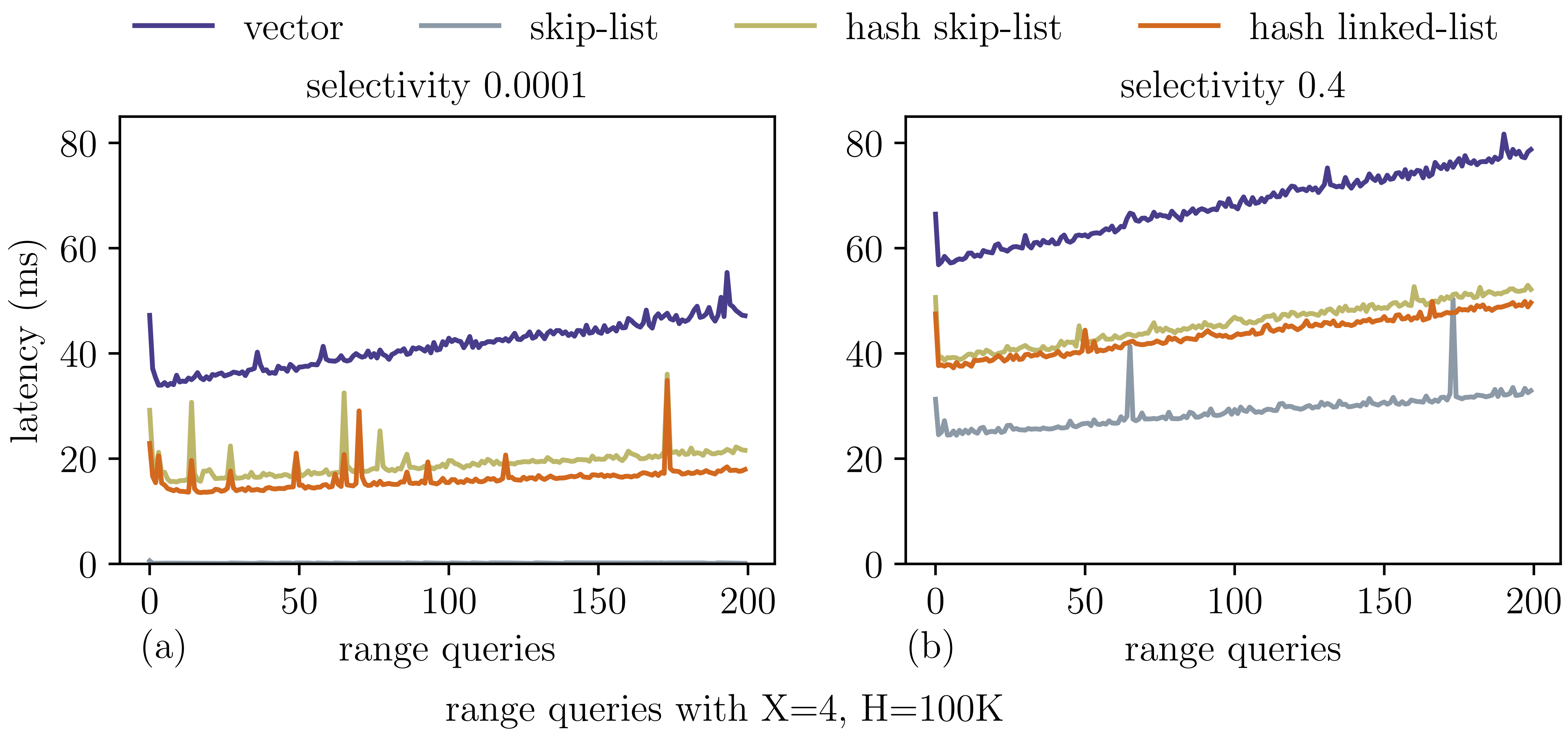
vector is 10x faster than skip-list for an *insert heavy* w/l

vector is worst choice for w/ls with point queries

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

skip-list outperforms for w/ls with range queries

140K inserts, 200 interleaved range queries



insert with range queries

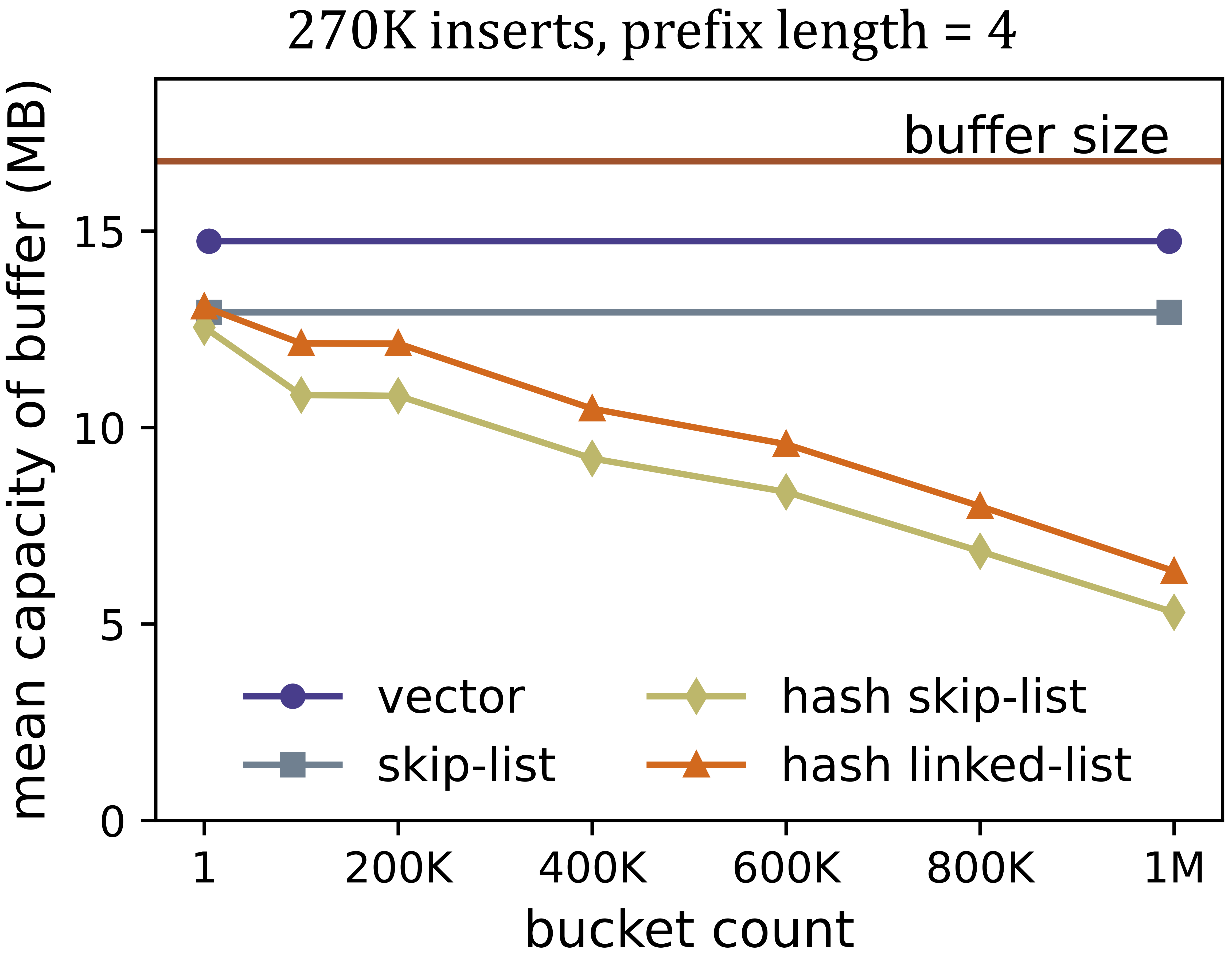
Evaluation

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

vector is worst choice for w/ls with point queries

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

skip-list outperforms for w/ls with range queries



memory footprint

Brandeis
UNIVERSITY

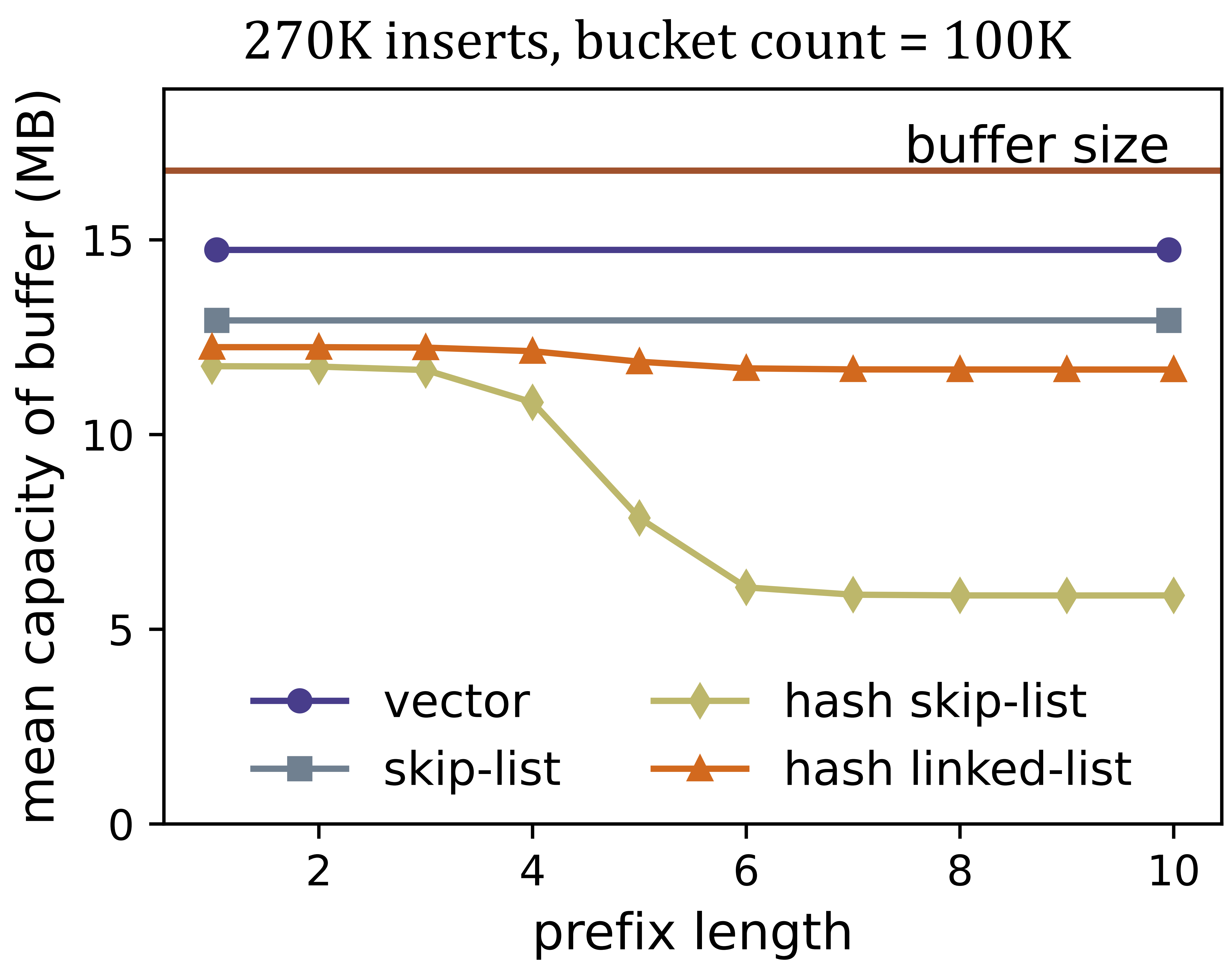
Evaluation

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

vector is worst choice for w/ls with point queries

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

skip-list outperforms for w/ls with range queries



memory footprint

Evaluation

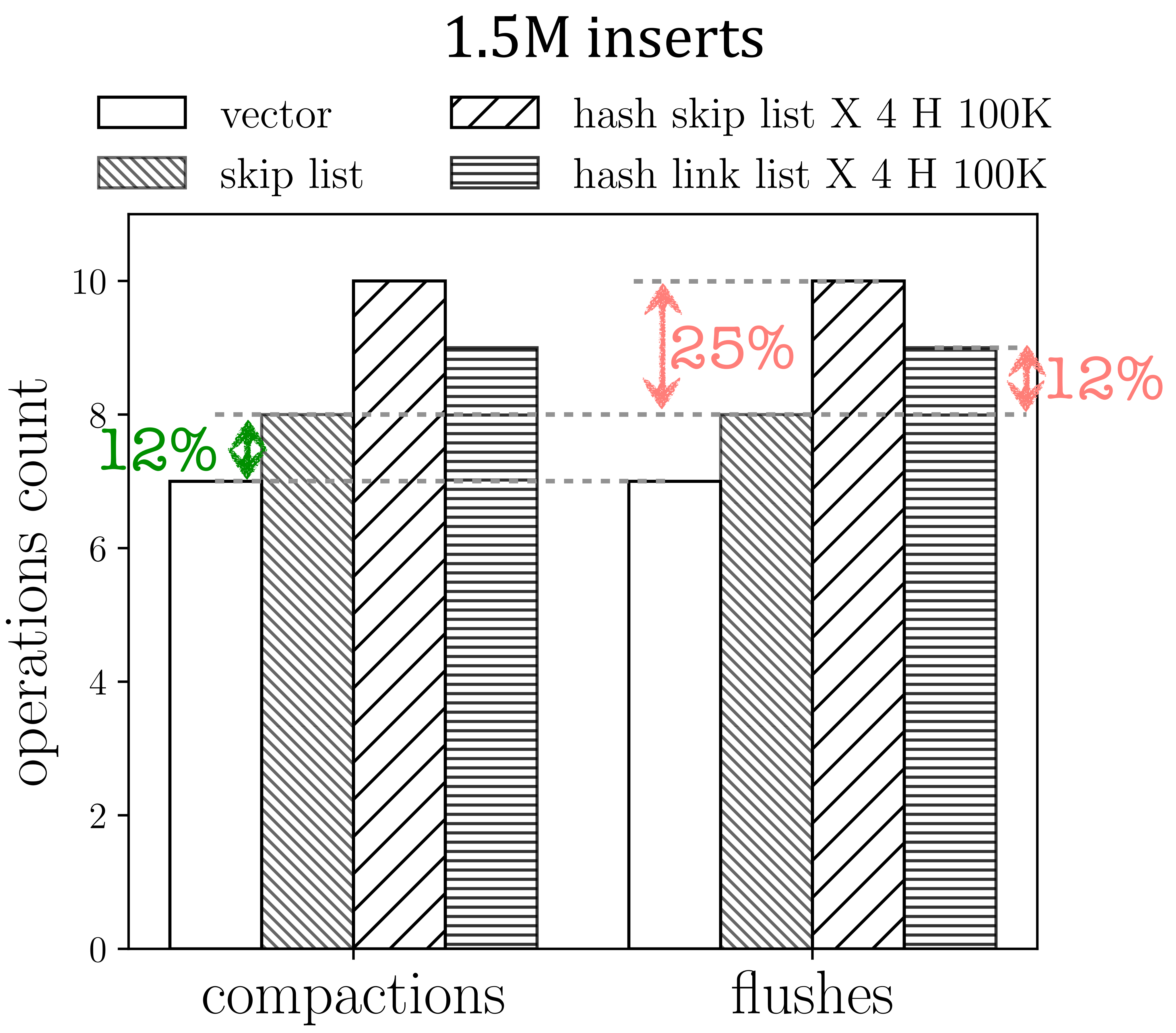
vector is 10x faster than skip-list for an *insert heavy* w/l

vector is worst choice for w/ls with point queries

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

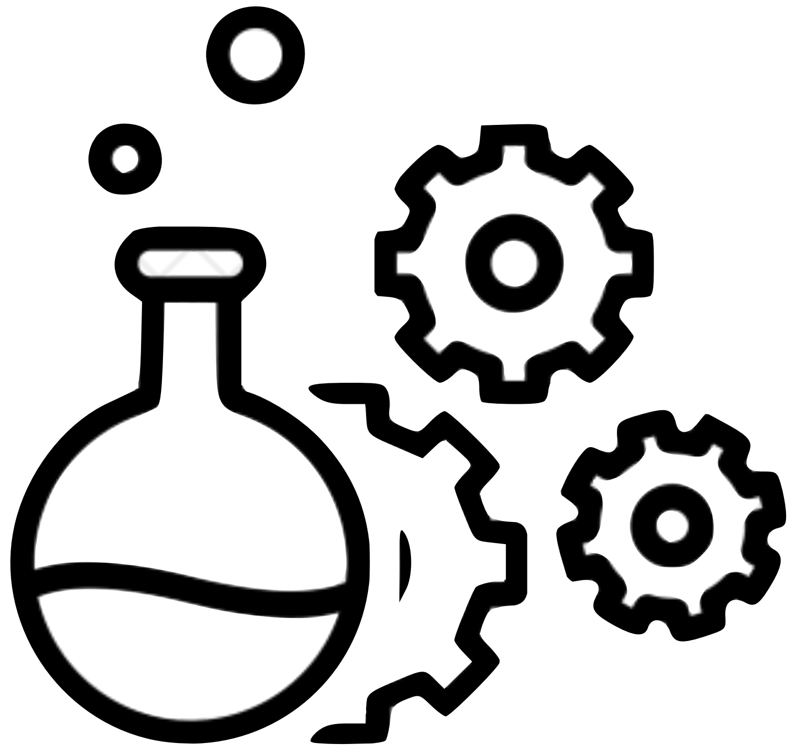
skip-list outperforms for w/ls with range queries

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



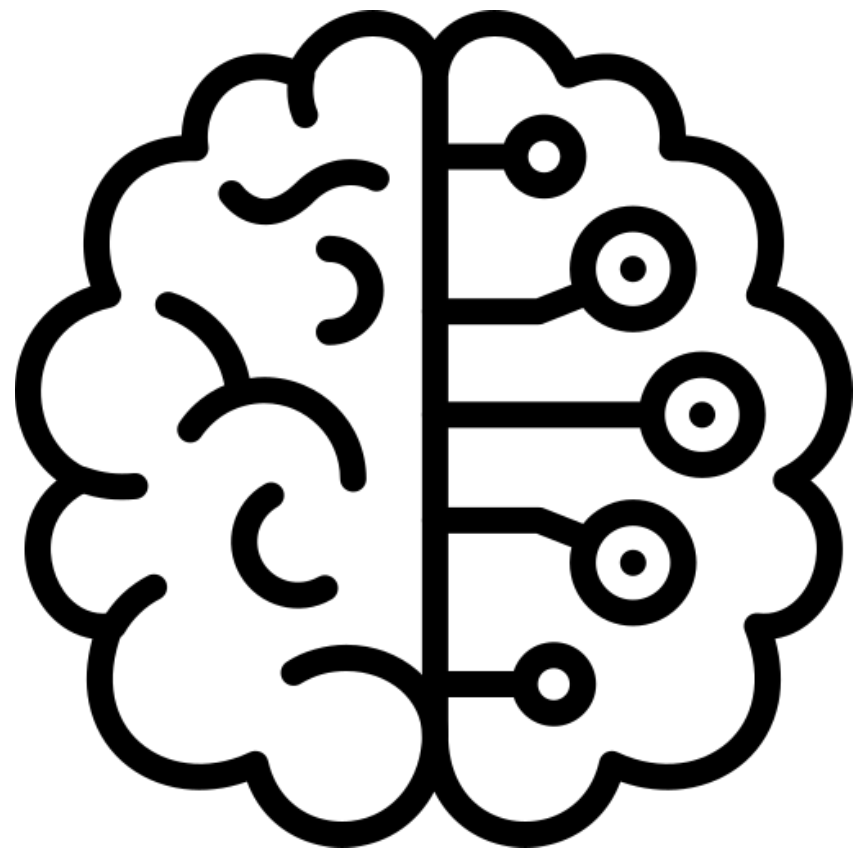
Brandeis UNIVERSITY

Ongoing Work



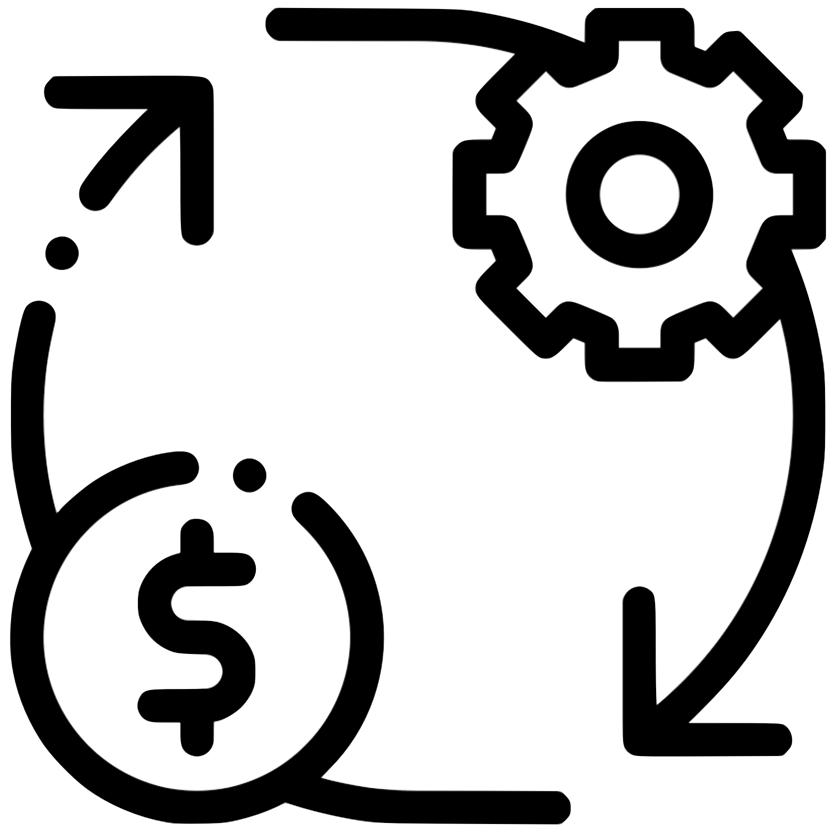
explore other implementations

tree, trie, heap



finding best buffer

using machine learning



switching buffer

on the fly when w/l changes

Anatomy of LSM Memory Buffer

Thank You!

Questions?

Santiago, Chile, 2024