COSI 167A
Advanced Data Systems

Class 7
Class Projects

Prof. Subhadeep Sarkar

https://ssd-brandeis.github.io/COSI-167A/

Brandeis UNIVERSITY

Class **logistics**

and administrivia

The **third technical question** is now available on the class website (due **before the class** on **Tue, Sep 24**).

Project 1 is due in **~12 hours**.

First guest lecture: next **Tuesday** (**Sep 24**).

Register for **paper presentation**! (https://shorturl.at/4P0IT)

Paper **presentation**

and discussion

Register for **paper presentation**! (https://shorturl.at/4POIT)

Paper **presentation**

and discussion

Register for **paper presentation**! (https://shorturl.at/4POIT)

Paper **presentation**

and discussion

Register for **paper presentation**! (https://shorturl.at/4POIT)

2 students will be responsible for presenting the paper

learn the art of **technical presentation**, think as: **visualizing a review**

prepare slides **at least a week before** your presentation

get your **slides reviewed** by me **twice** before your final presentation

Pro tip: Start EARLY!

Today in COSI 167A

What's on the cards?

Cost analysis

Class projects!

Cost analysis

Counting all I/Os

Ingestion cost

expected #I/Os performed to write a single entry to **disk**

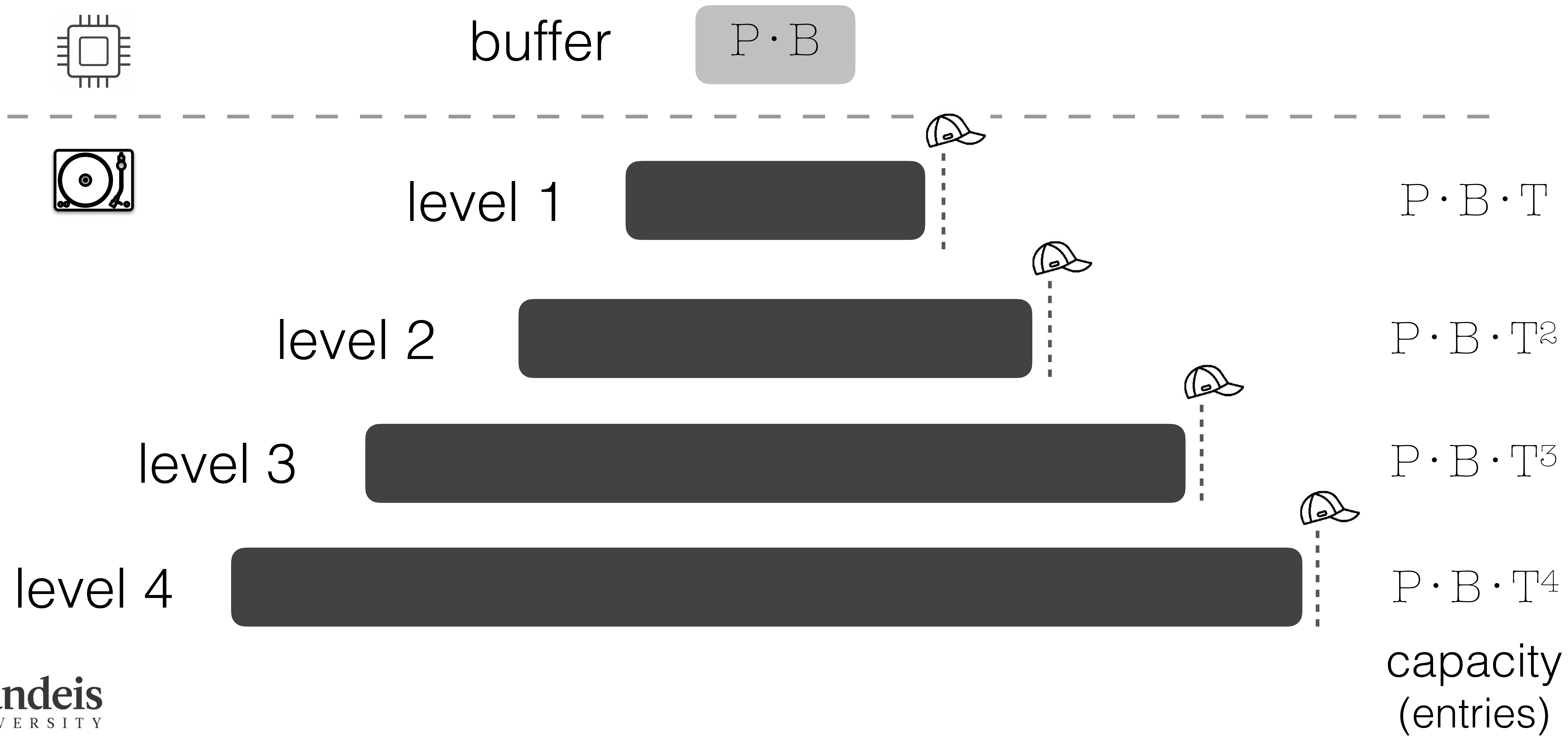
note: if an entry is written **multiple times**, **count all I/Os**

Query cost

expected #I/Os performed to perform a single query

compare costs with and without **auxiliary** (helper) **data structures**

P: pages in buffer
B: entries/page
L: #levels
T: size ratio

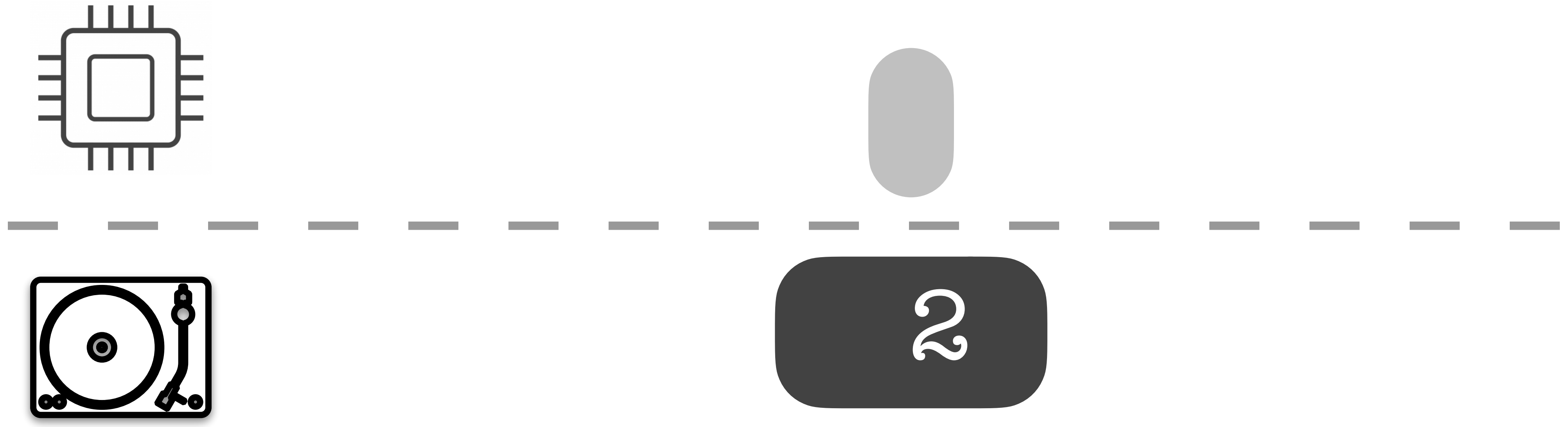


P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Ingestion cost

Inserts and updates

leveled LSM-tree



in general, **#times an entry is written to a level = T**

happens for all **L** levels on disk

Total #times an entry is written in the tree = $\mathbf{L \cdot T}$

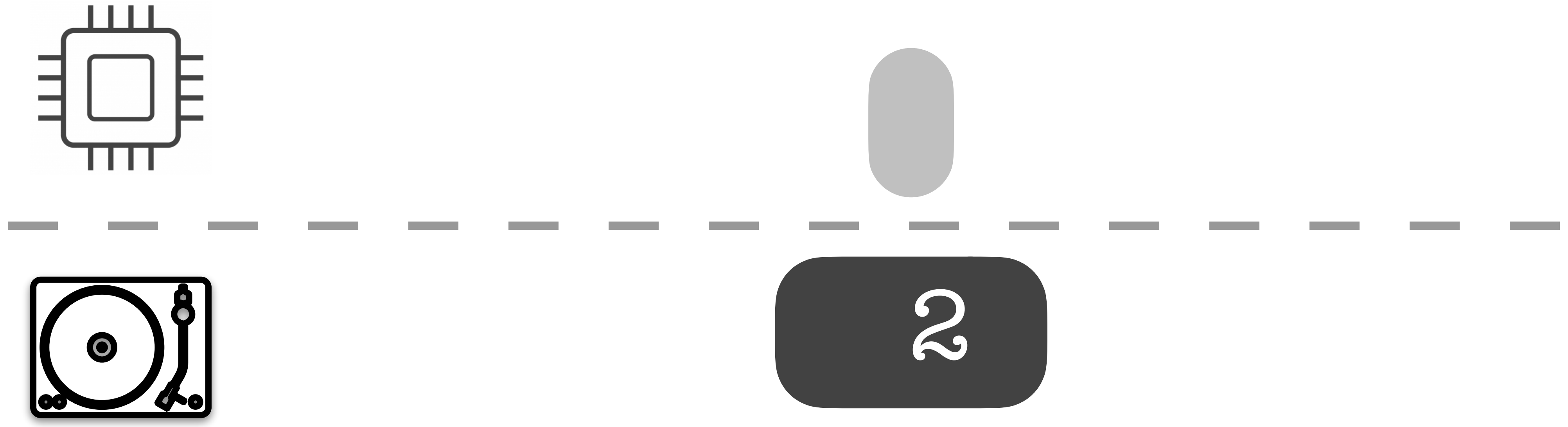
B entries are written to disk per I/O

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Ingestion cost

Inserts and updates

leveled LSM-tree



in general, **#times an entry is written to a level = T**

happens for all **L** levels on disk

Total #times an entry is written in the tree = **L · T**

B entries are written to disk per I/O

Average number of I/Os to write a single entry = **L · T / B**

Cost analysis

Counting all I/Os

data structure	ingestion cost	point lookup cost	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$		
Tiered LSM-tree			
B+-tree			
Sorted array			
Log			

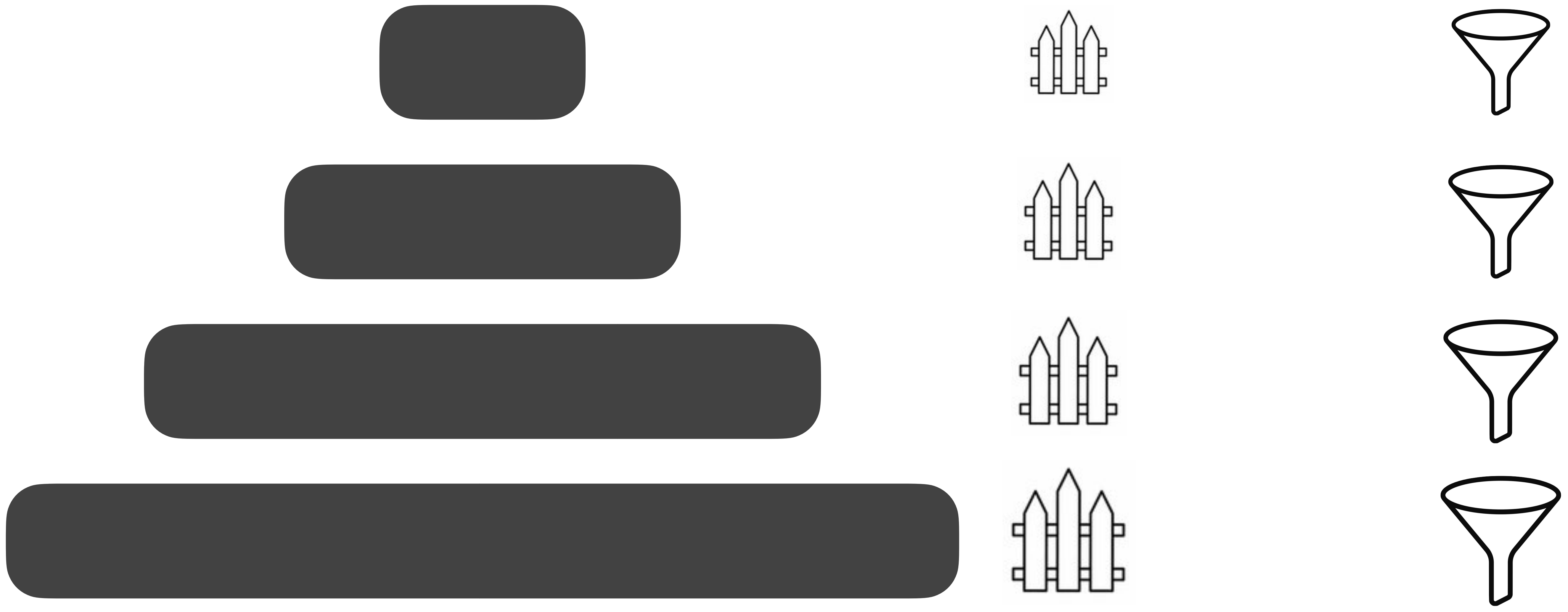
Cost analysis

Counting all I/Os

data structure	ingestion cost	point lookup cost	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$		
Tiered LSM-tree	$O(L / B)$		
B+-tree			
Sorted array			
Log			

Point lookup cost

Looking for a specific key



leveled LSM-tree

fence pointers

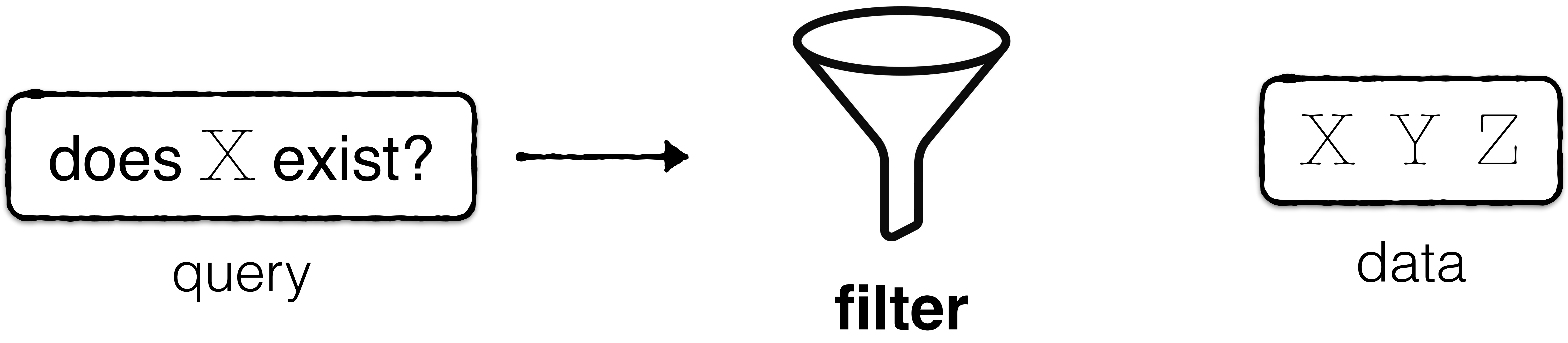
(page-wise zone map)

filter

What is a **filter?**

Answers membership queries

answers
membership queries



no false negatives

false positives with tunable probability

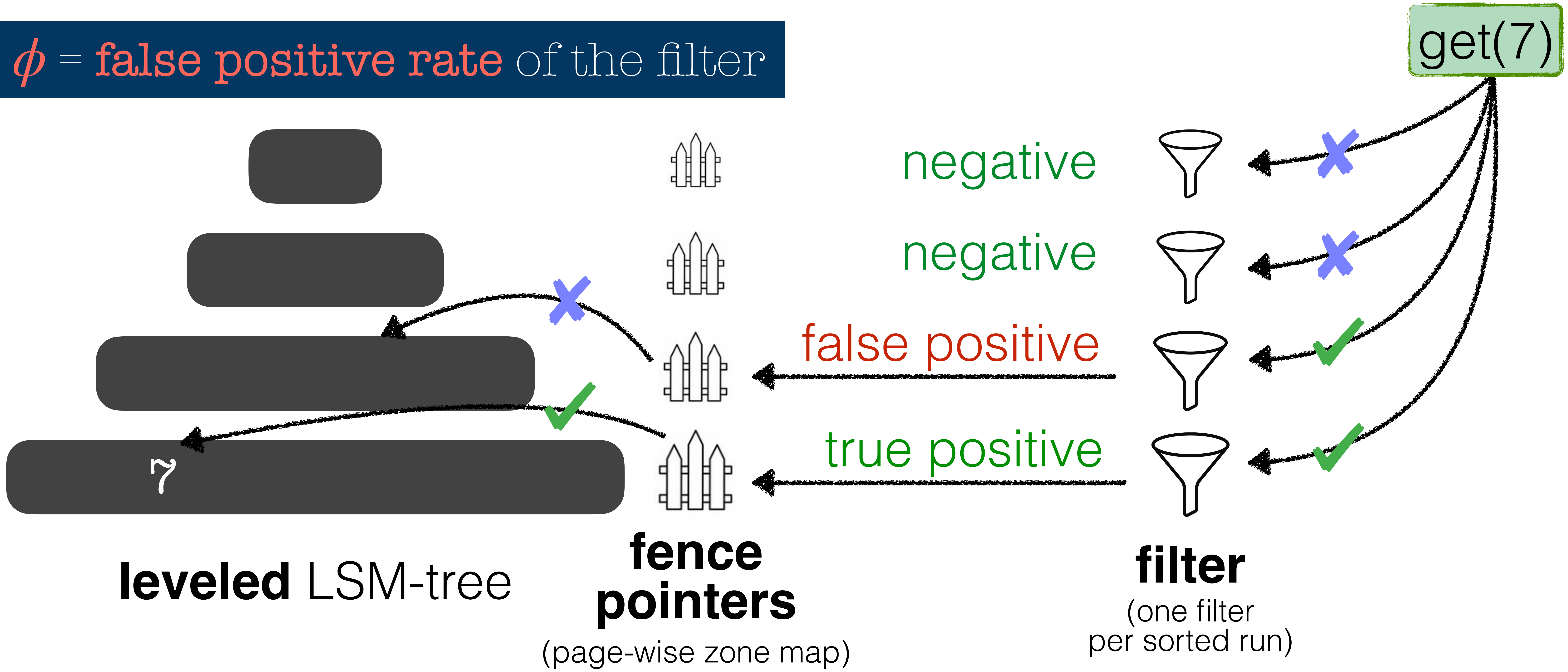
Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Point lookup cost

Looking for a specific key

ϕ = false positive rate of the filter

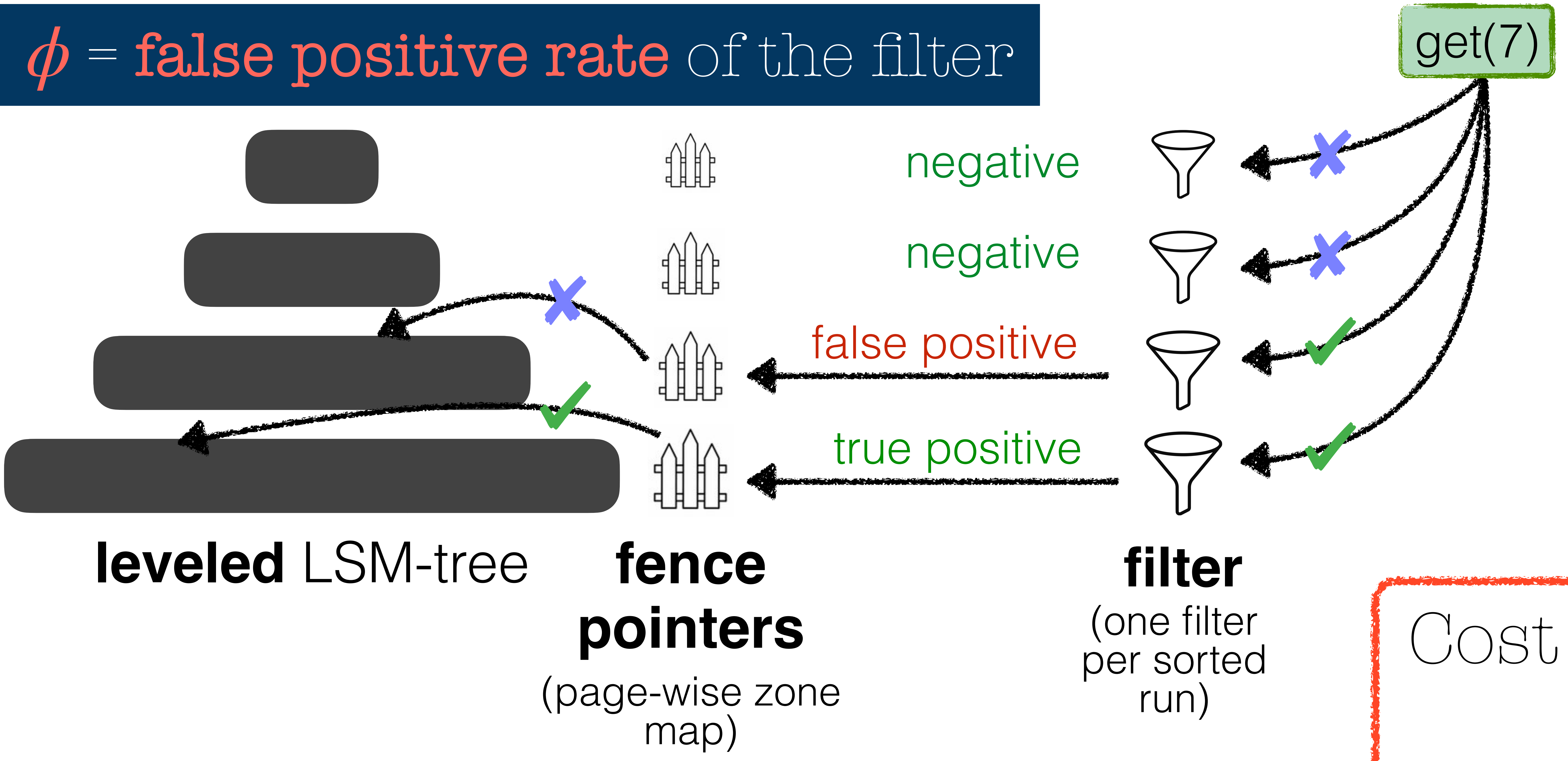


P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Point lookup cost

Looking for a specific key

ϕ = false positive rate of the filter



1 I/O for the sorted run (level) containing the data

+

1 I/O with probability ϕ for all other sorted runs

1 sorted runs per level

Cost of non-empty point lookup
$= 1 + \phi \cdot (L-1)$

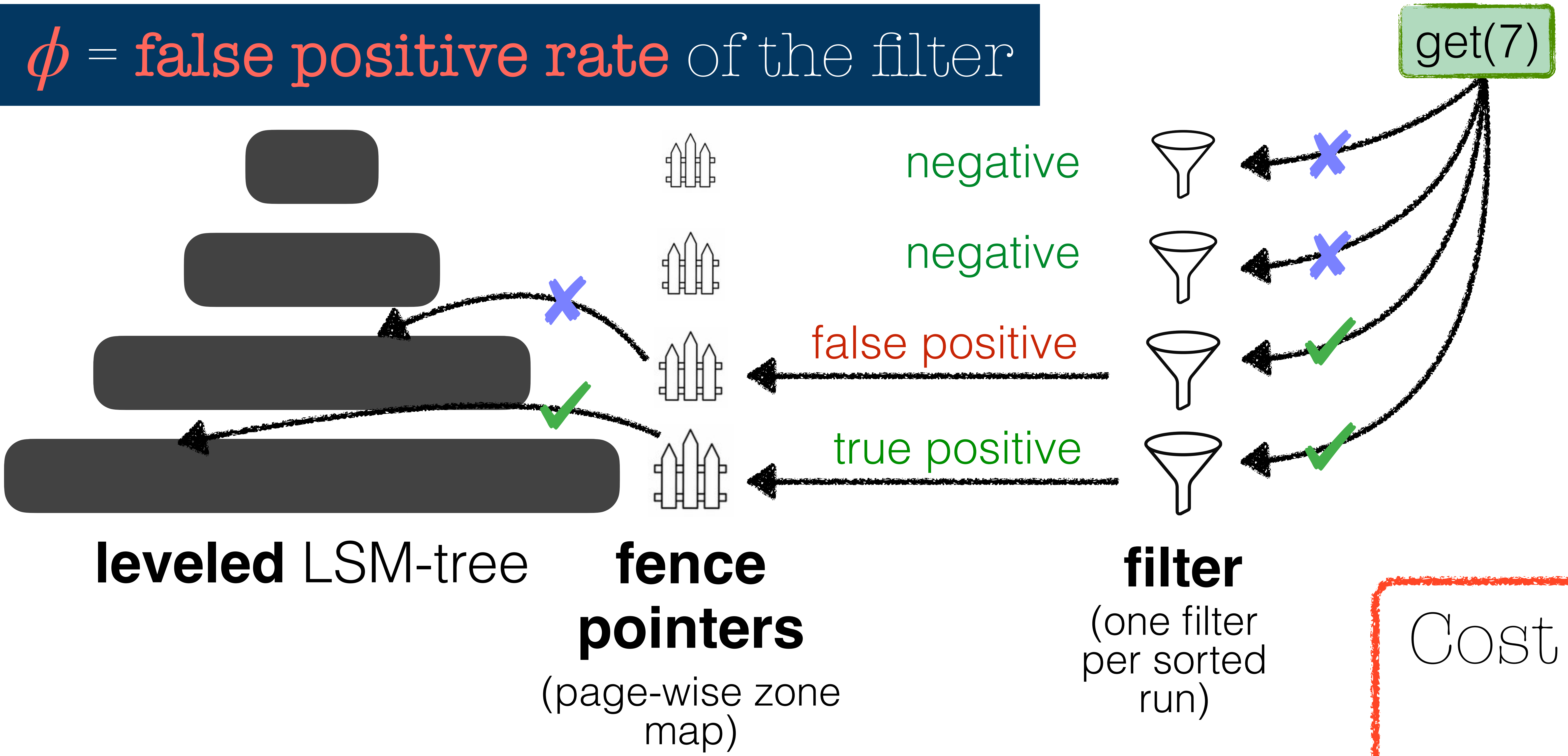
Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Point lookup cost

Looking for a specific key

ϕ = false positive rate of the filter



1 I/O for the sorted run (level) containing the data

+

1 I/O with probability ϕ for all other sorted runs

1 sorted runs per level

Cost of non-empty point lookup $= 1 + \phi \cdot (L-1)$

Cost of **empty** point lookup = $\phi \cdot L$

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	ingestion cost	point lookup cost*	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$	
Tiered LSM-tree	$O(L / B)$		
B+-tree			
Sorted array			
Log			

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	ingestion cost	point lookup cost*	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$	
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$	
B+-tree			
Sorted array			
Log			

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

Cost analysis

Counting all I/Os

ϕ = 0.008 with 10 BPK

data structure	ingestion cost	point lookup cost*	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$	
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$	
B+-tree			
Sorted array			
Log			

Monkey takes this **another step further**!

Cost analysis

Counting all I/Os

ϕ = 0.008 with 10 BPK

data structure	ingestion cost	point lookup cost *	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+\phi \cdot L)^*$	
Tiered LSM-tree	$O(L / B)$	$O(1+\phi \cdot L \cdot T)^*$	
B+-tree			
Sorted array			
Log			

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

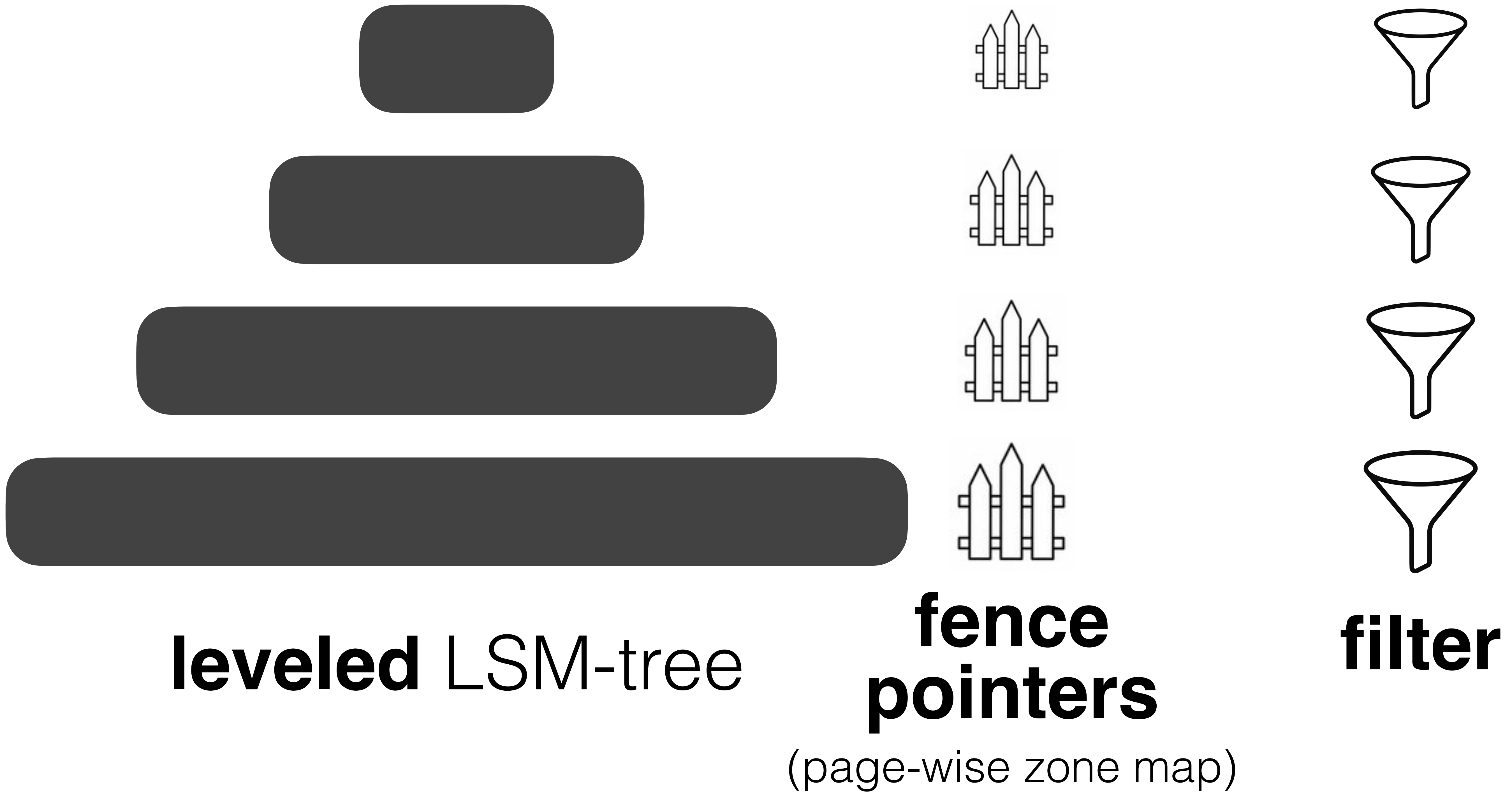
Range lookup cost

Looking for keys in a range

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range

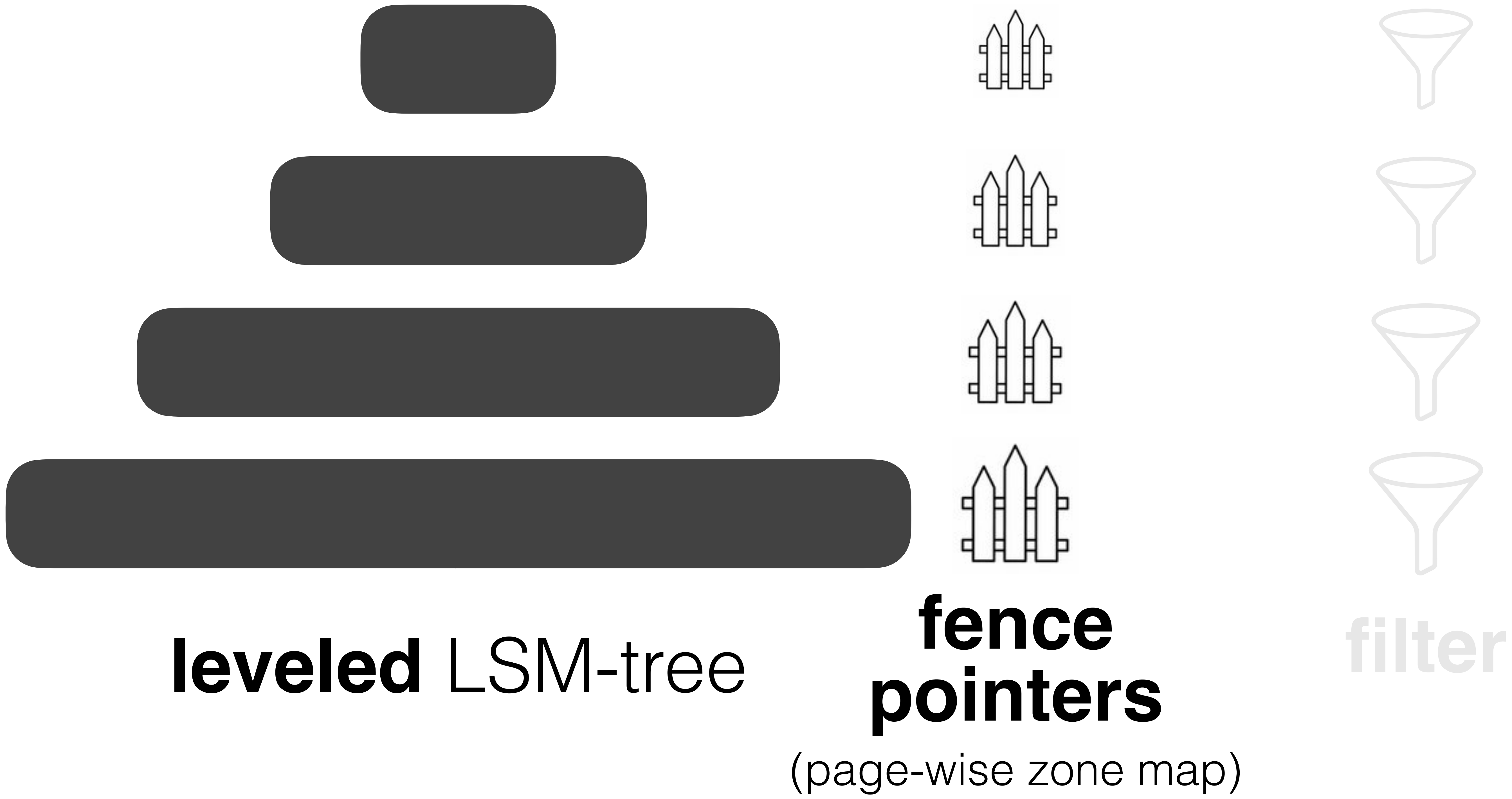


Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range

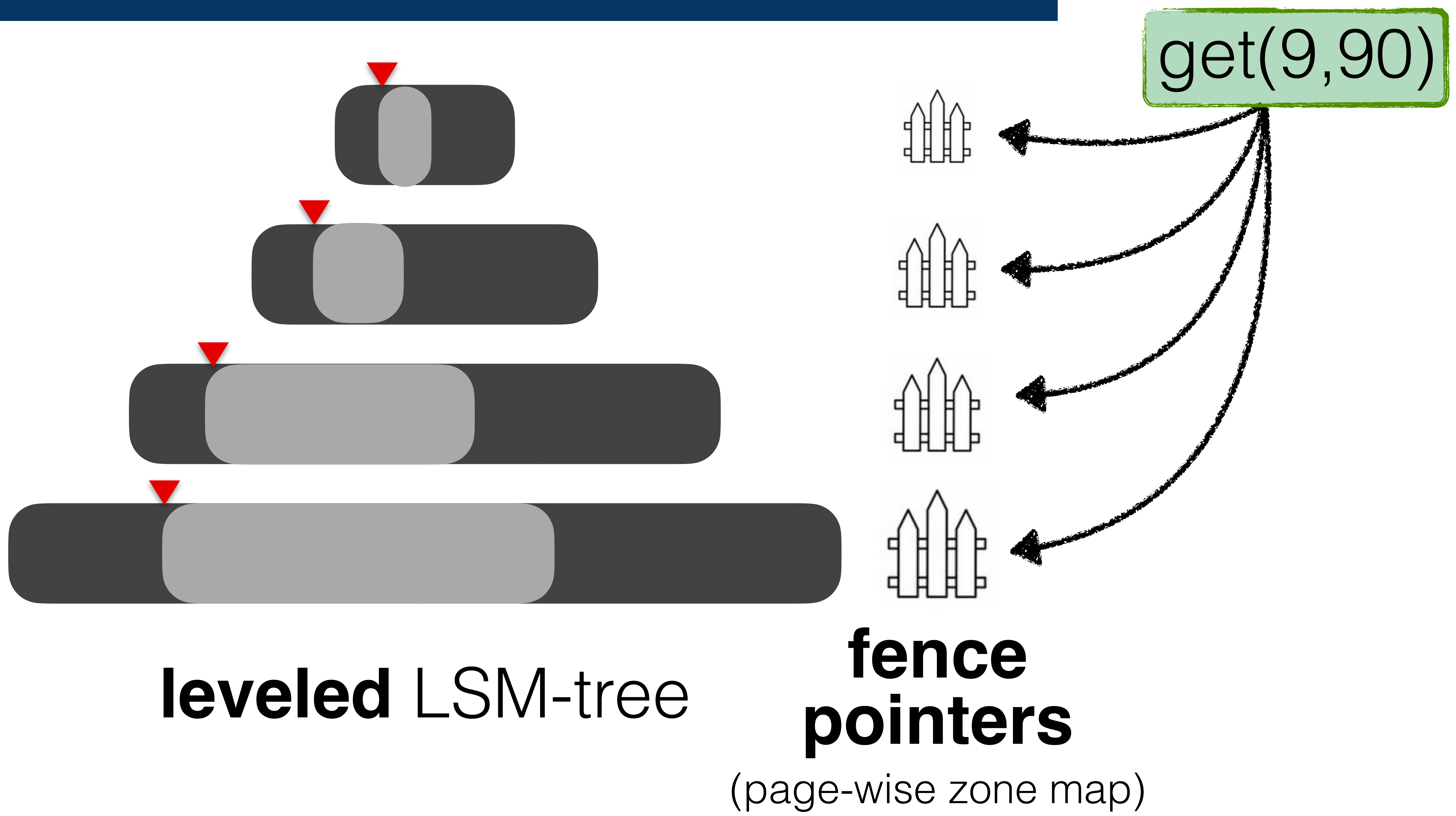


P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range

s = selectivity of the range query

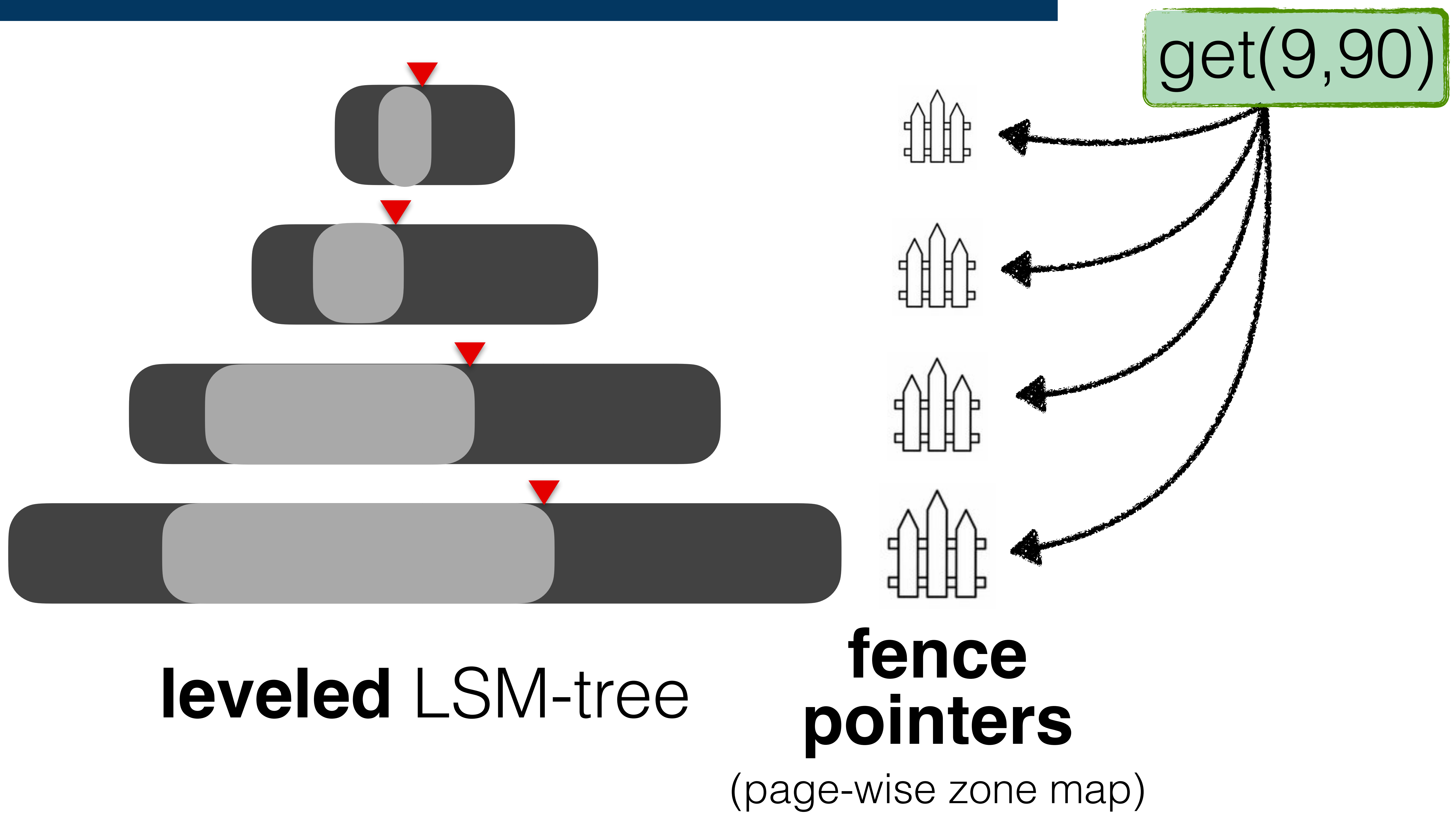


P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range

s = selectivity of the range query



total entries in tree = **N**

#entries per page = **B**

#pages in tree = N/B

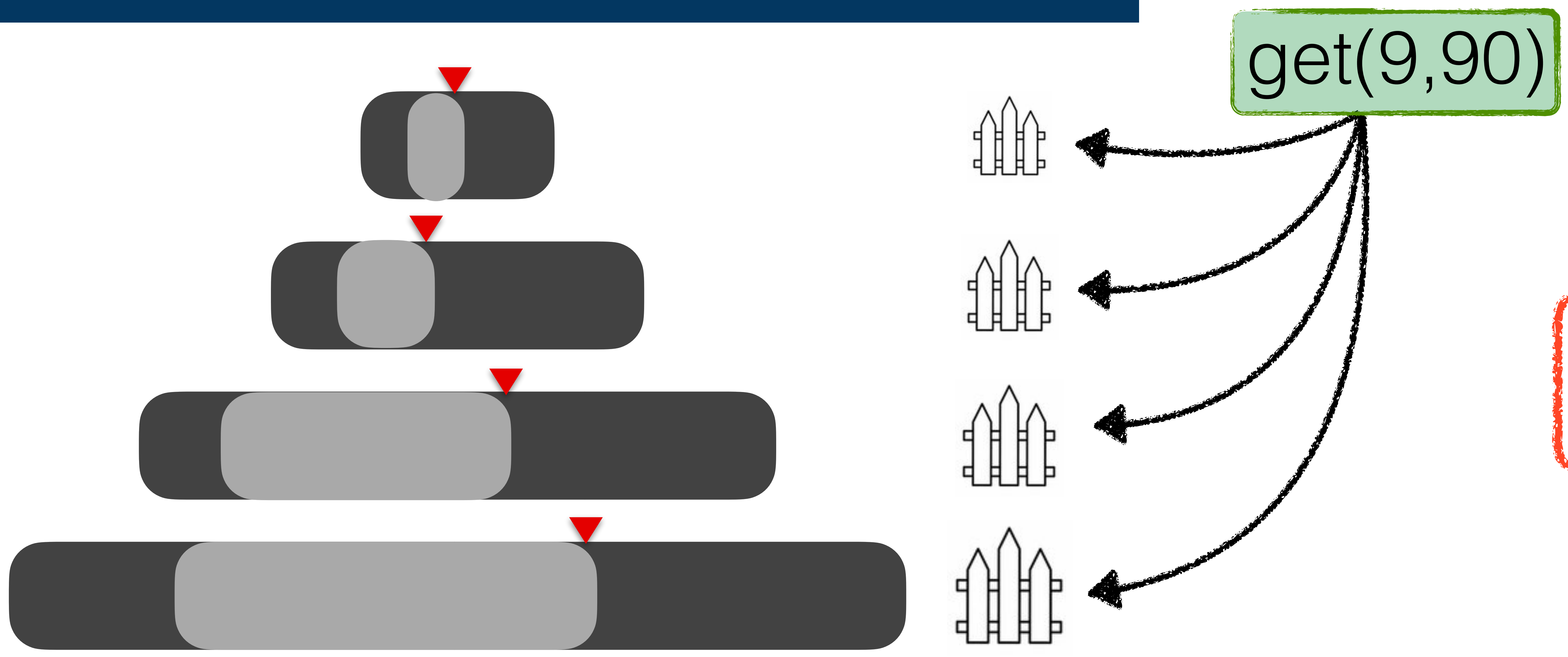
Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range

s = selectivity of the range query



leveled LSM-tree

fence pointers
(page-wise zone map)

total entries in tree = **N**

#entries per page = **B**

#pages in tree = N/B

Cost of range lookup = $s \cdot N/B$

What about the **range query cost** in a **tiered LSM-tree**?

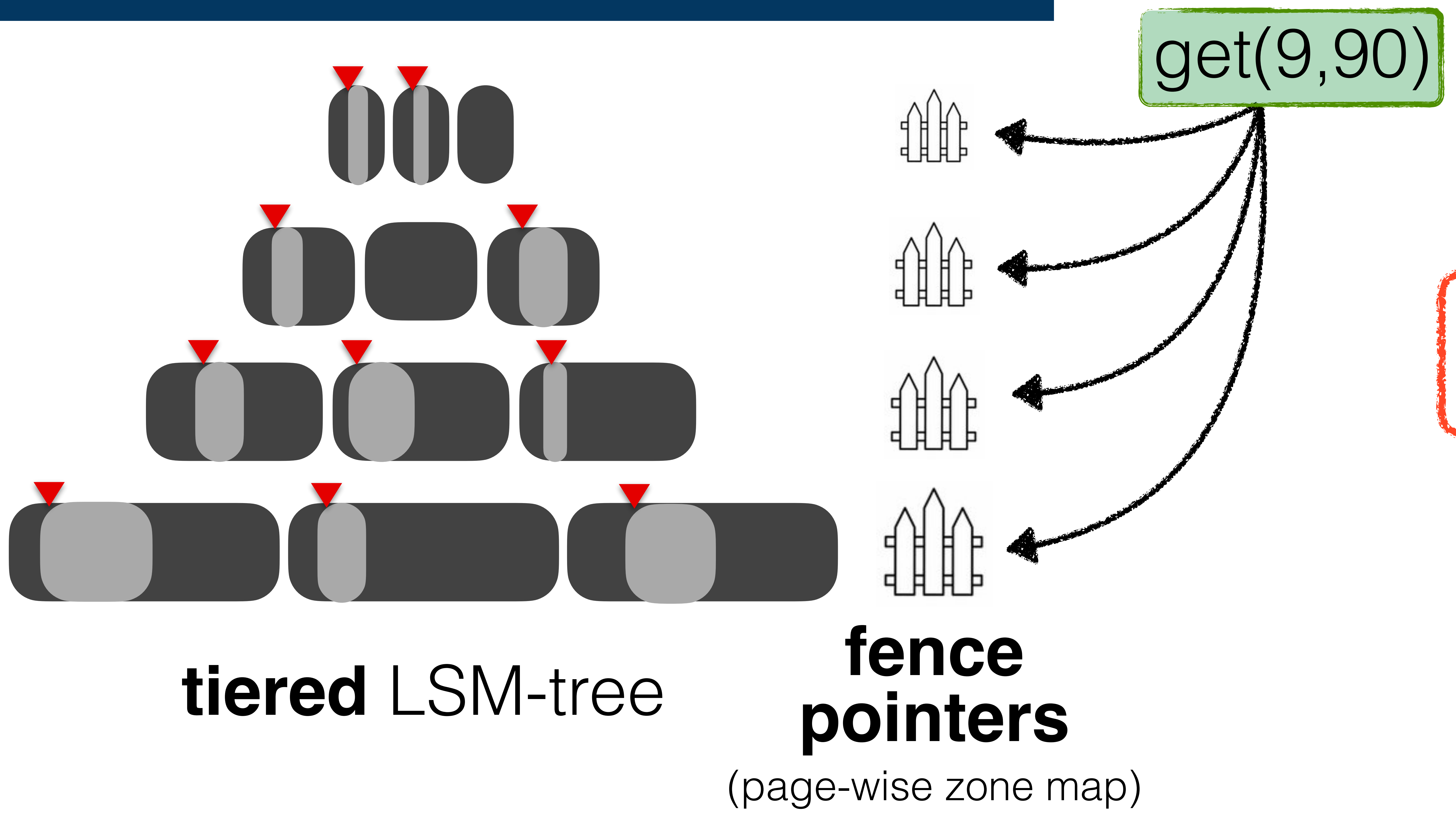
same cost for
leveled & tiered LSM

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range

s = selectivity of the range query



total entries in tree = **N**
#entries per page = **B**

#pages in tree = N/B

Cost of range lookup = $s \cdot N/B$

What about the **range query cost** in a **tiered LSM-tree**?

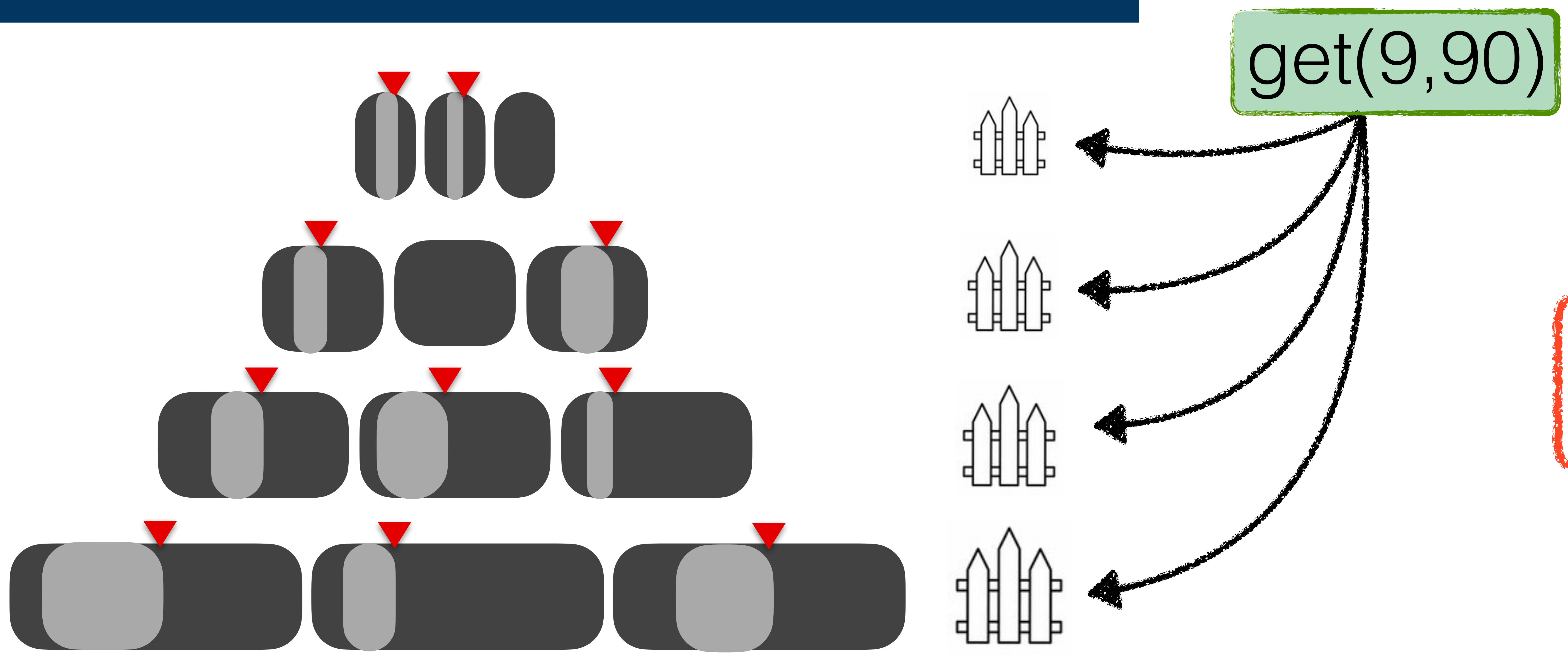
same cost for
leveled & tiered LSM

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range

s = selectivity of the range query



tiered LSM-tree

fence pointers
(page-wise zone map)

total entries in tree = **N**
#entries per page = **B**

#pages in tree = N/B

Cost of range lookup = $s \cdot N/B$

What about the **range query cost** in a **tiered LSM-tree**?

same cost for

Very different story when workload has update/deletes.

Cost analysis

Counting all I/Os

data structure	ingestion cost	point lookup cost*	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)^*$	$O(s \cdot N / B)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)^*$	$O(s \cdot N / B)$
B^+-tree			
Sorted array			
Log			

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

Cost analysis

Counting all I/Os

* **long** range lookups

data structure	ingestion cost	point lookup cost*	range lookup cost*
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)^*$	$O(s \cdot N / B)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)^*$	$O(s \cdot N / B)$
B^+-tree			
Sorted array			
Log			

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

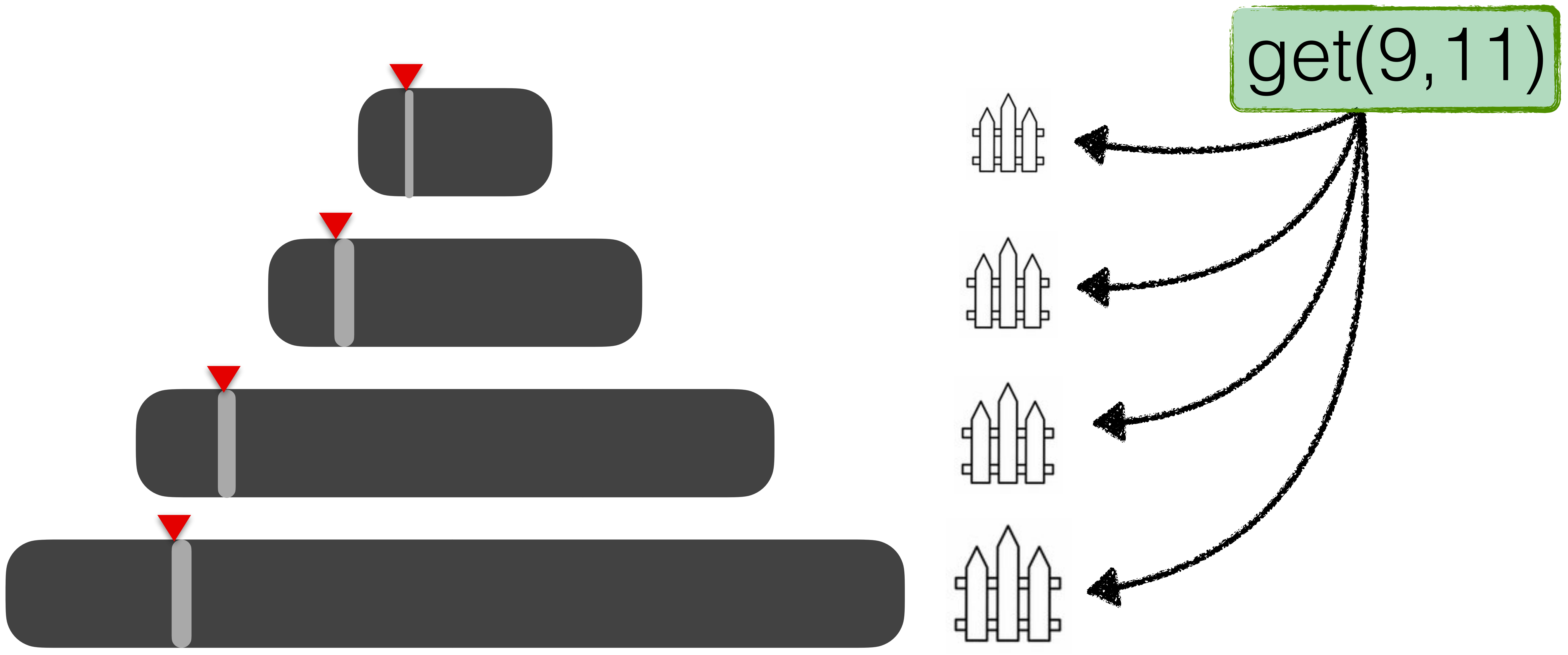
* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range



leveled LSM-tree

fence pointers

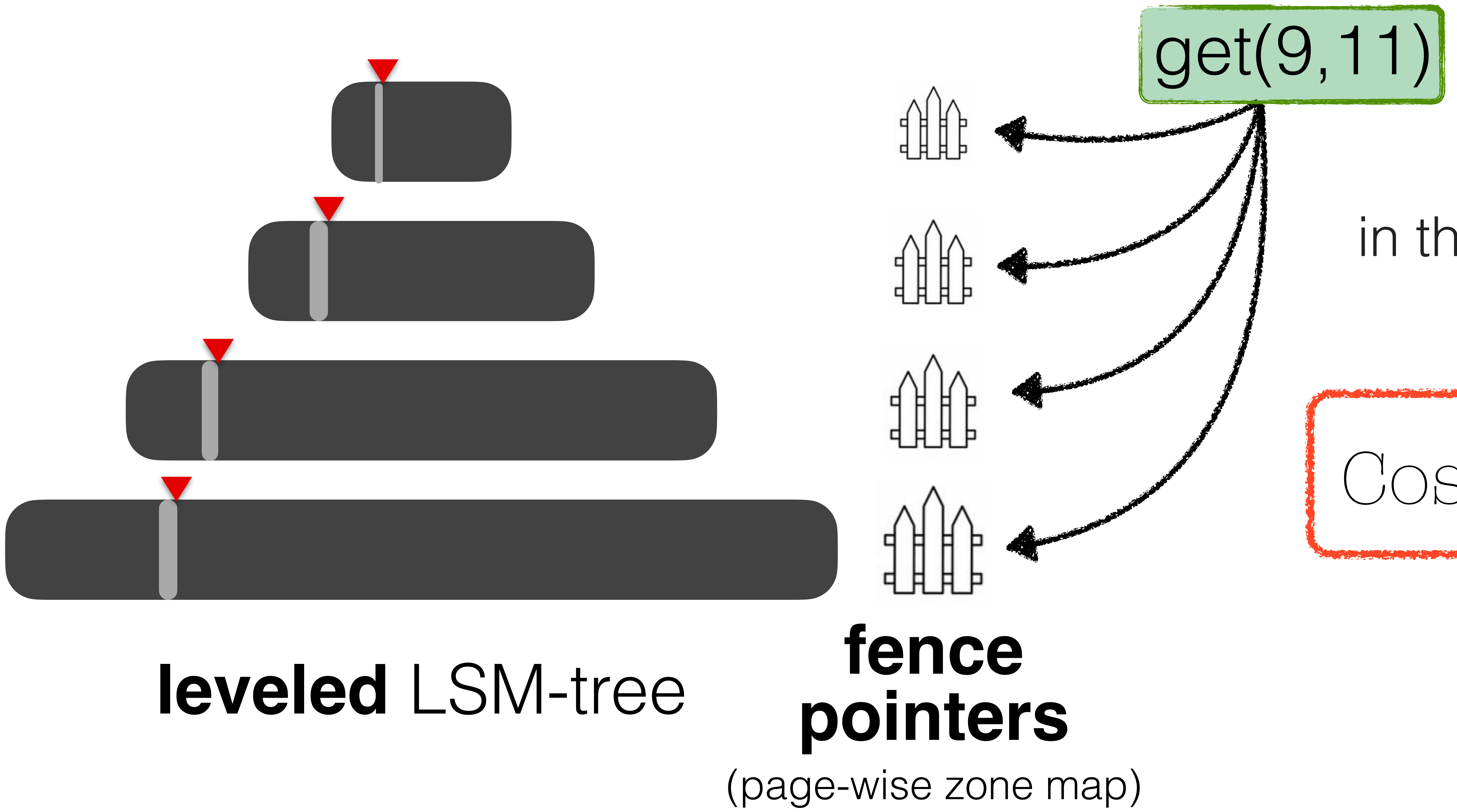
(page-wise zone map)

What if the **range query** has a **very low selectivity**?

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range



What if the **range query** has a **very low selectivity**?

in the **worst case**,
need to read **2 pages per sorted run**

Cost of short range lookup = $2 \cdot L$

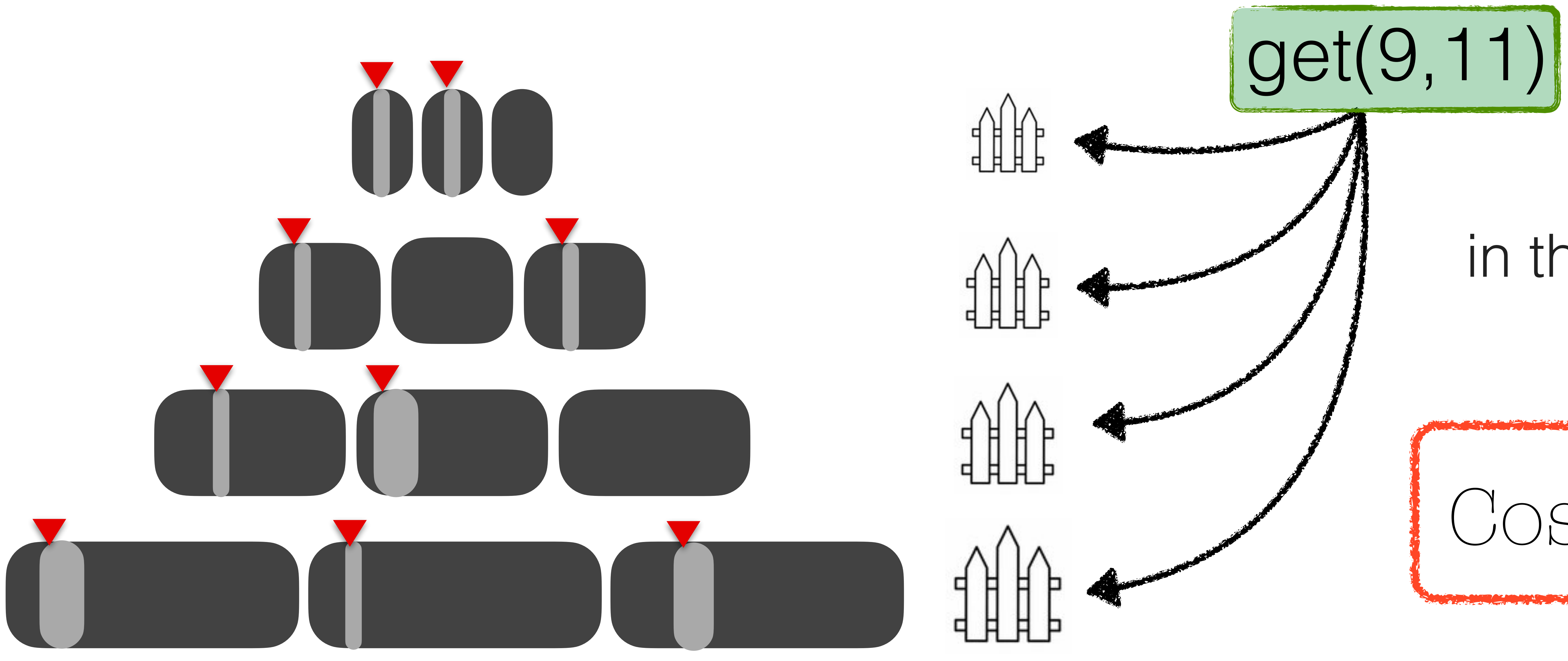
What about the **tiering**?

Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range



tiered LSM-tree

fence pointers

(page-wise zone map)

What if the **range query** has a **very low selectivity**?

in the **worst case**,
need to read **2 pages per sorted run**

Cost of short range lookup = $2 \cdot L$

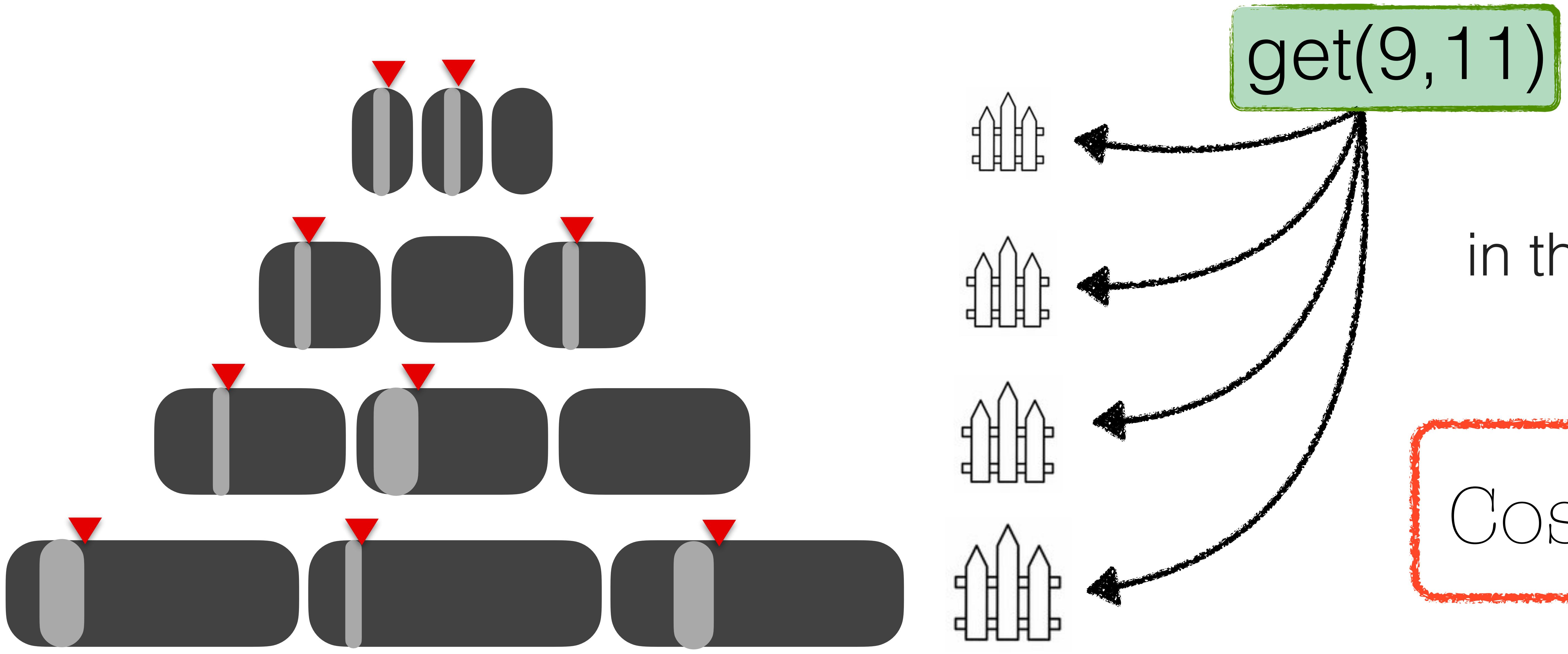
What about the **tiering**?

Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range



tiered LSM-tree

fence pointers
(page-wise zone map)

What if the **range query** has a **very low selectivity**?

in the **worst case**,
need to read **2 pages per sorted run**

Cost of short range lookup = $2 \cdot L$

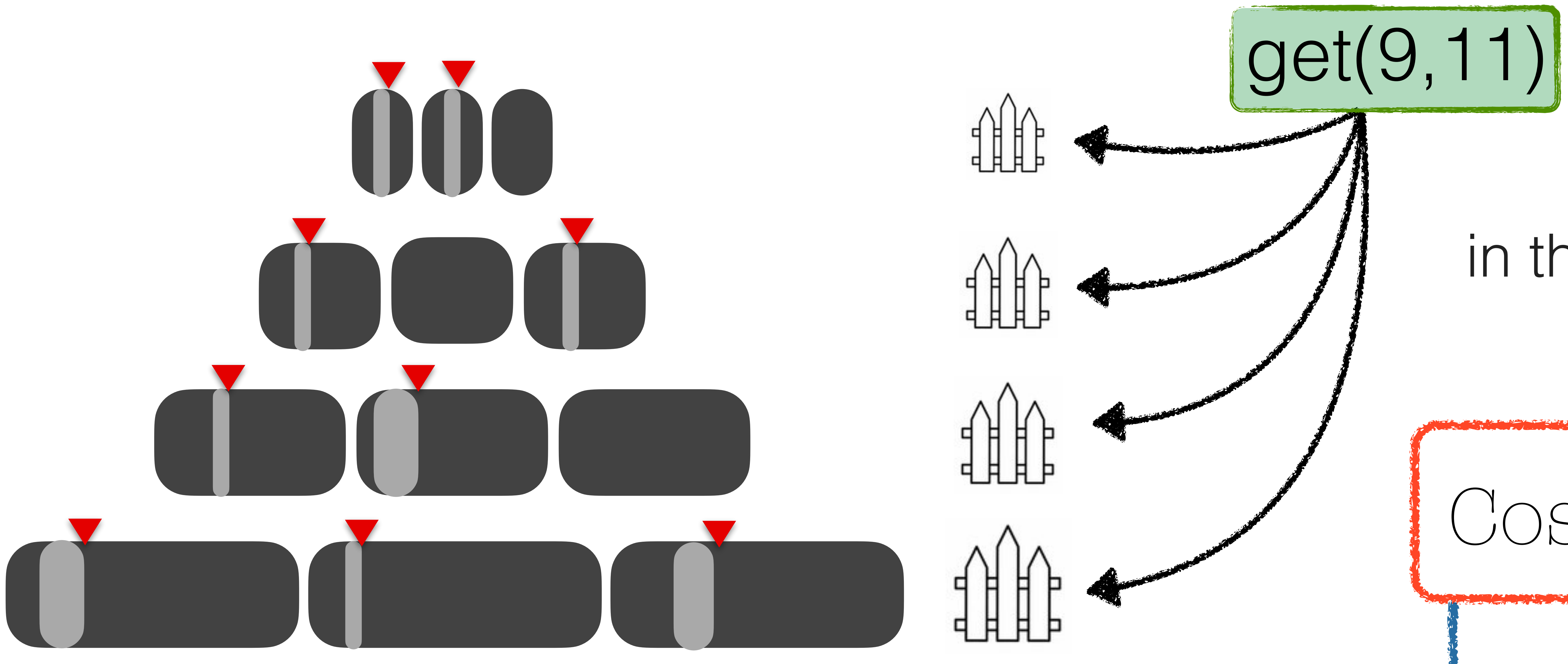
What about the **tiering**?
in the **worst case**,
need to read **2 pages per sorted run**

Brandeis UNIVERSITY

P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries in tree

Range lookup cost

Looking for keys in a range



tiered LSM-tree

fence pointers
(page-wise zone map)

What if the **range query** has a **very low selectivity**?

in the **worst case**,
need to read **2 pages per sorted run**

Cost of short range lookup = $2 \cdot L$

leveling

tiering

Cost of short range lookup = $2 \cdot L \cdot T$

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)^*$	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)^*$	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree				
Sorted array				
Log				

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)^{*}$	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)^{*}$	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	?	?	?	?
Sorted array				
Log				

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

B+-trees

The go to index data structures for relational databases

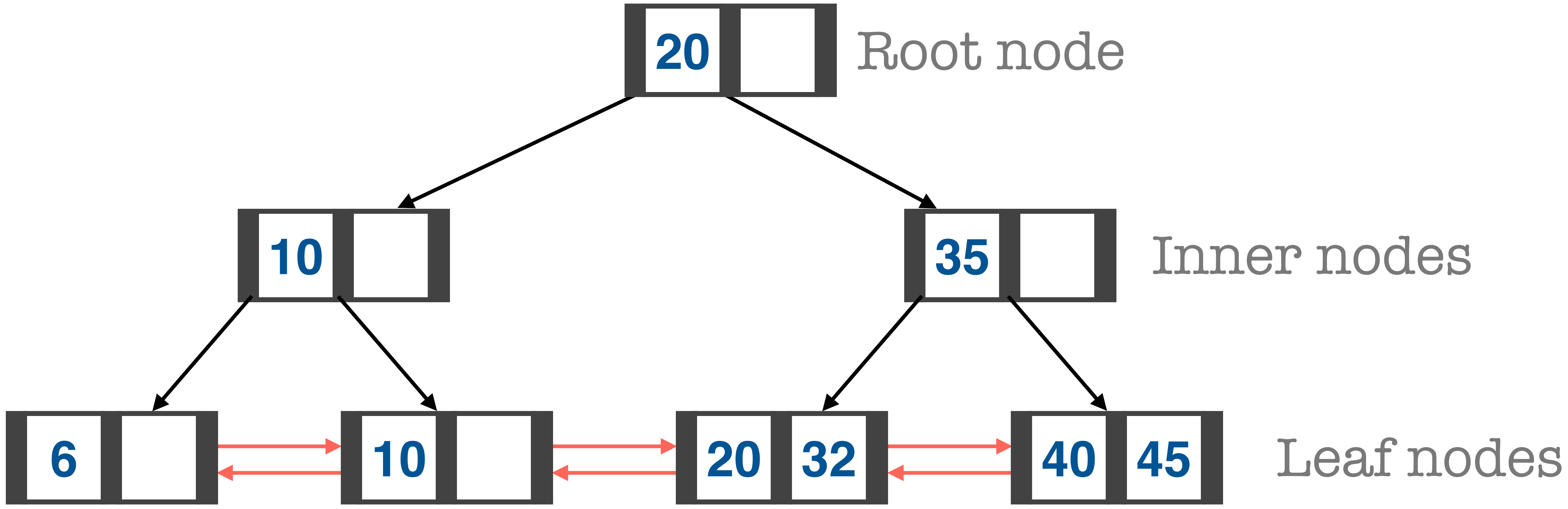
It could be said that the world's information is at our fingertips because of B-trees.

Goetz Graefe, Google, Ex-Microsoft, HP Fellow

Google ACM Software System Award

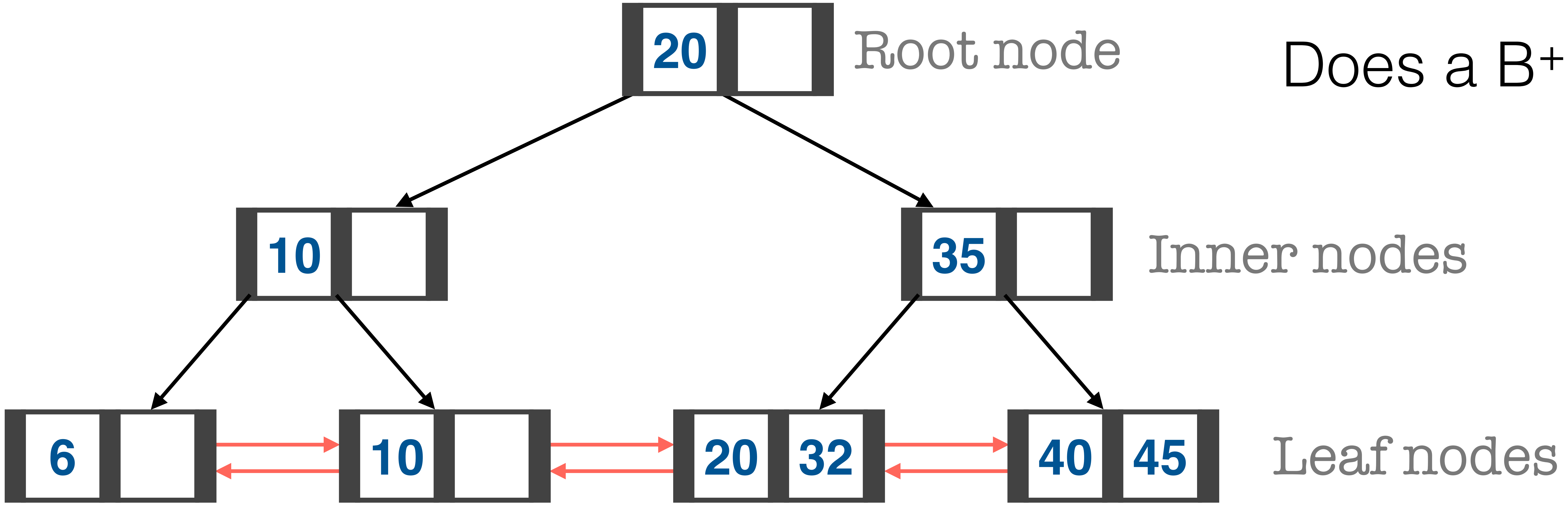
B+-trees

The go to index data structures for relational databases



B+-trees

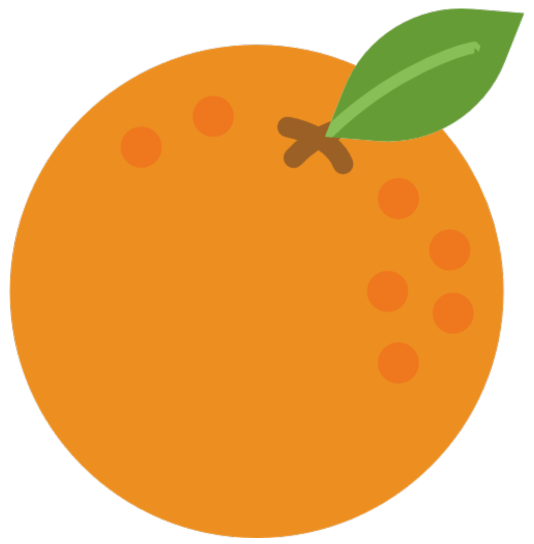
The go to index data structures for relational databases



Does a B+-tree have a **buffer**?

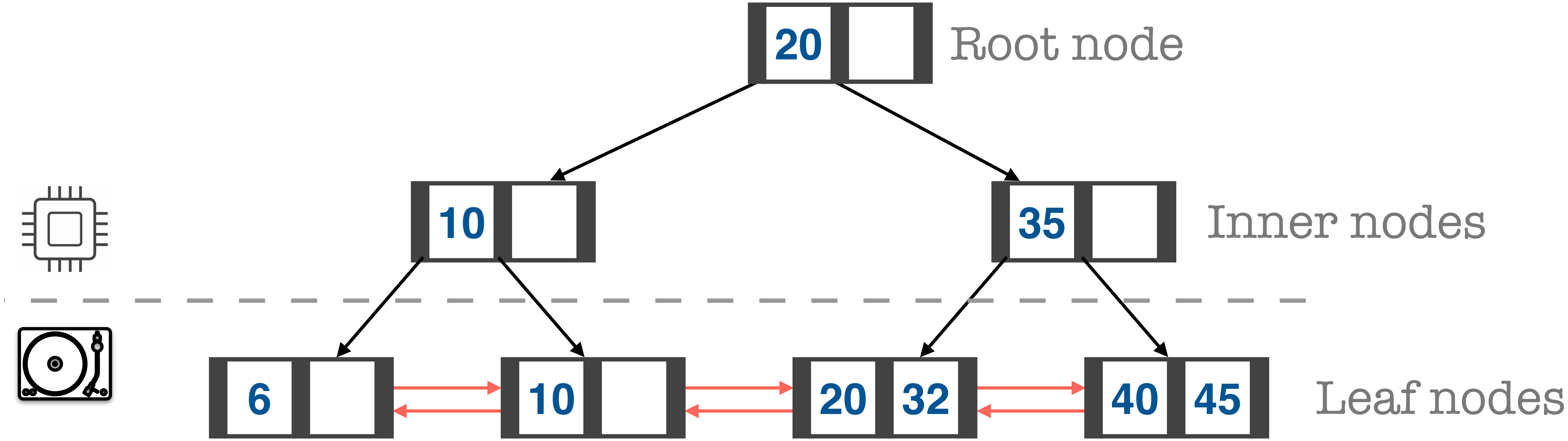
No!

VS.



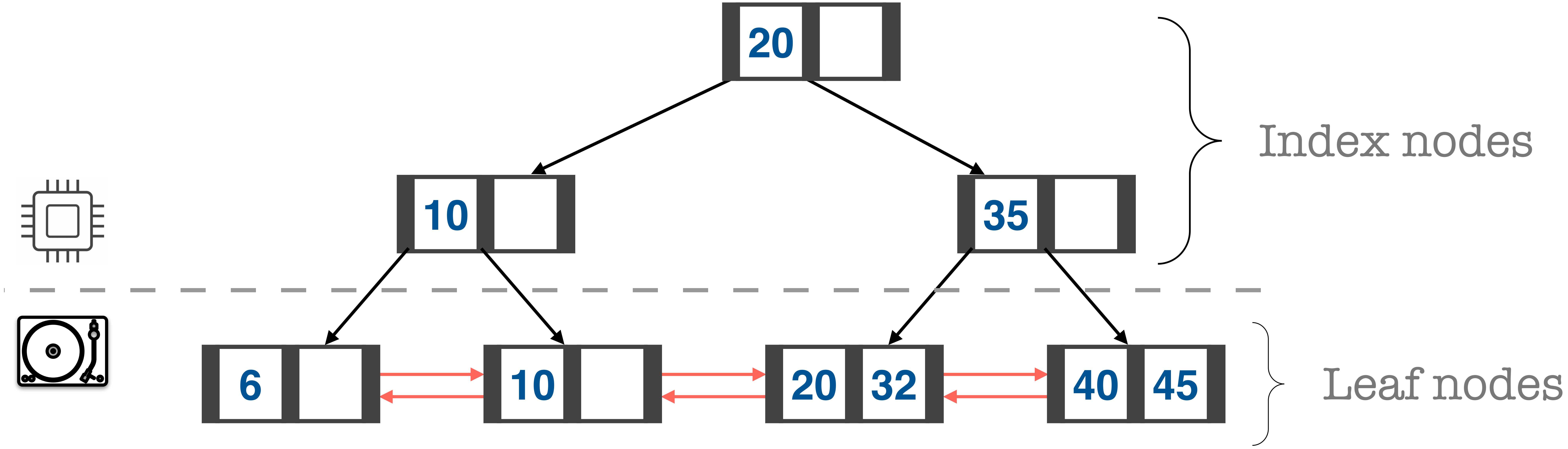
B+-trees

The go to index data structures for relational databases



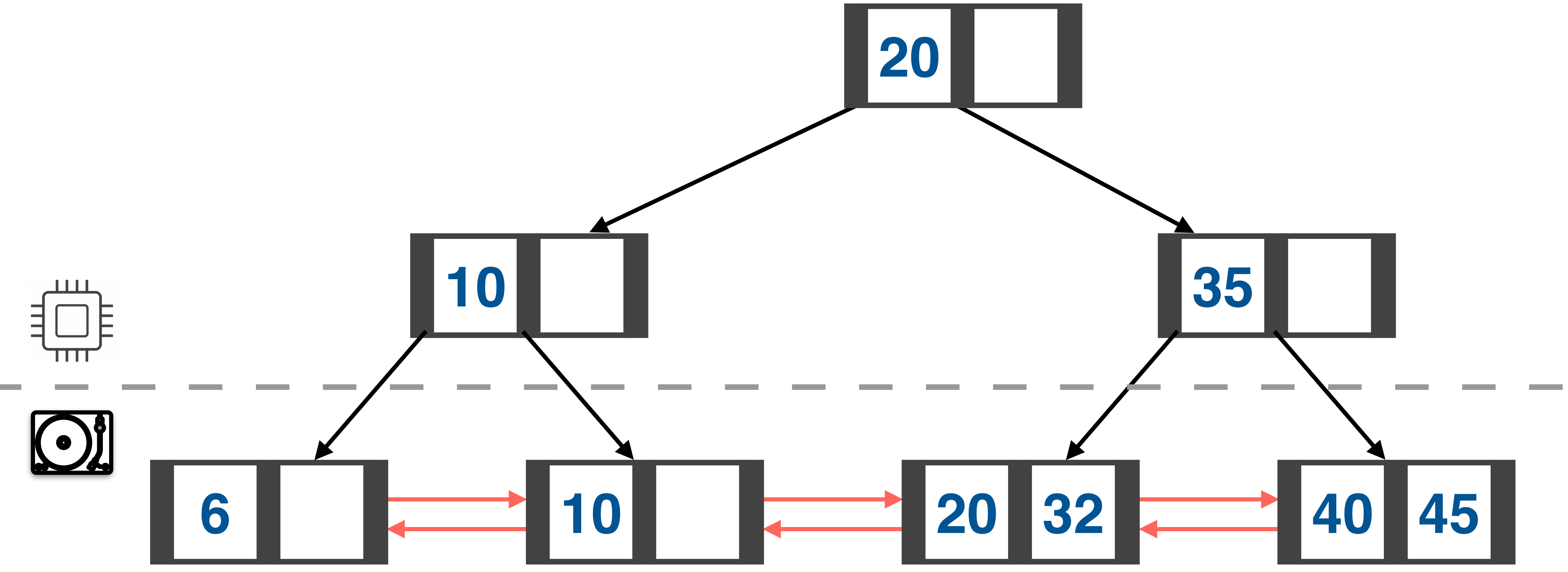
B+-trees

The go to index data structures for relational databases



B+-trees

The go to index data structures for relational databases



Thought Experiment 1

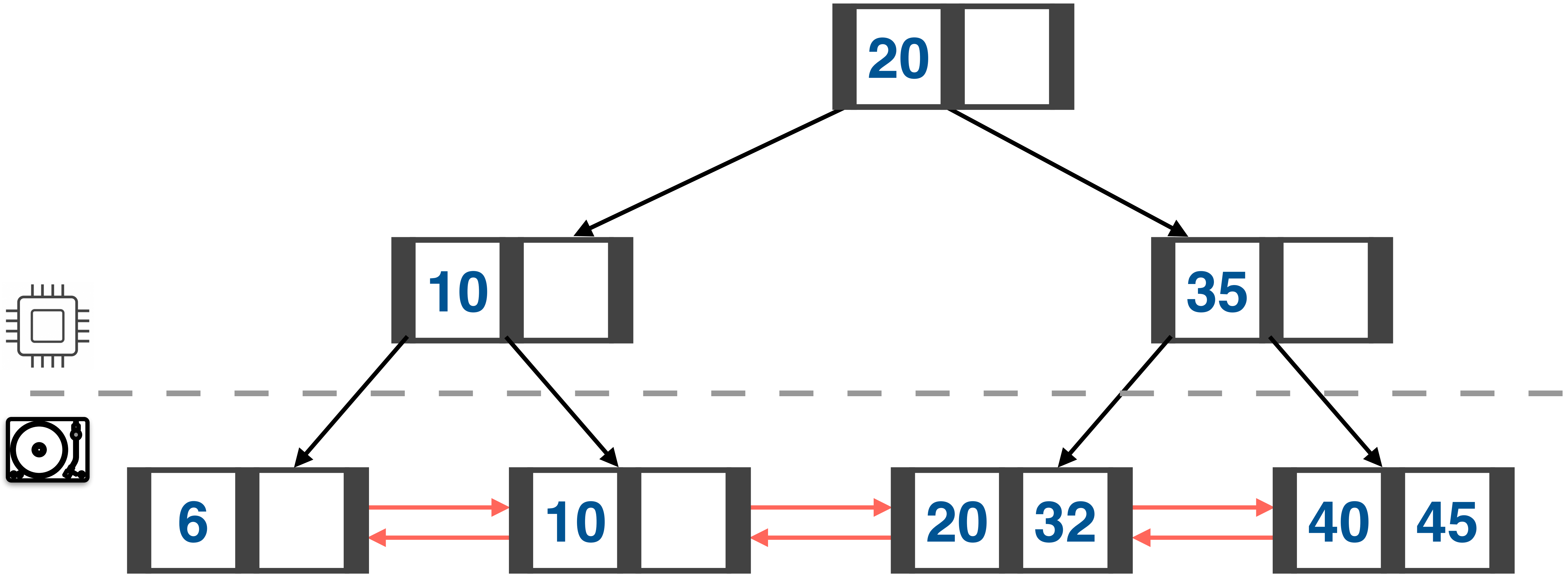
What is the **insert cost** in a **buffered B+-tree**?

2 I/Os per insert

1 to read, 1 to write

B+-trees

The go to index data structures for relational databases



Thought Experiment 1

What is the **insert cost** in a **buffered B+-tree**?

2 I/Os per insert

1 to read, 1 to write

Cost of ingestion = 2

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+\phi \cdot L)^{*}$	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+\phi \cdot L \cdot T)^{*}$	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	?	?	?
Sorted array				
Log				

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+ \phi \cdot L)^{*}$	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+ \phi \cdot L \cdot T)^{*}$	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	?	?	?
Sorted array				
Log				

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

Cost analysis

Counting all I/Os

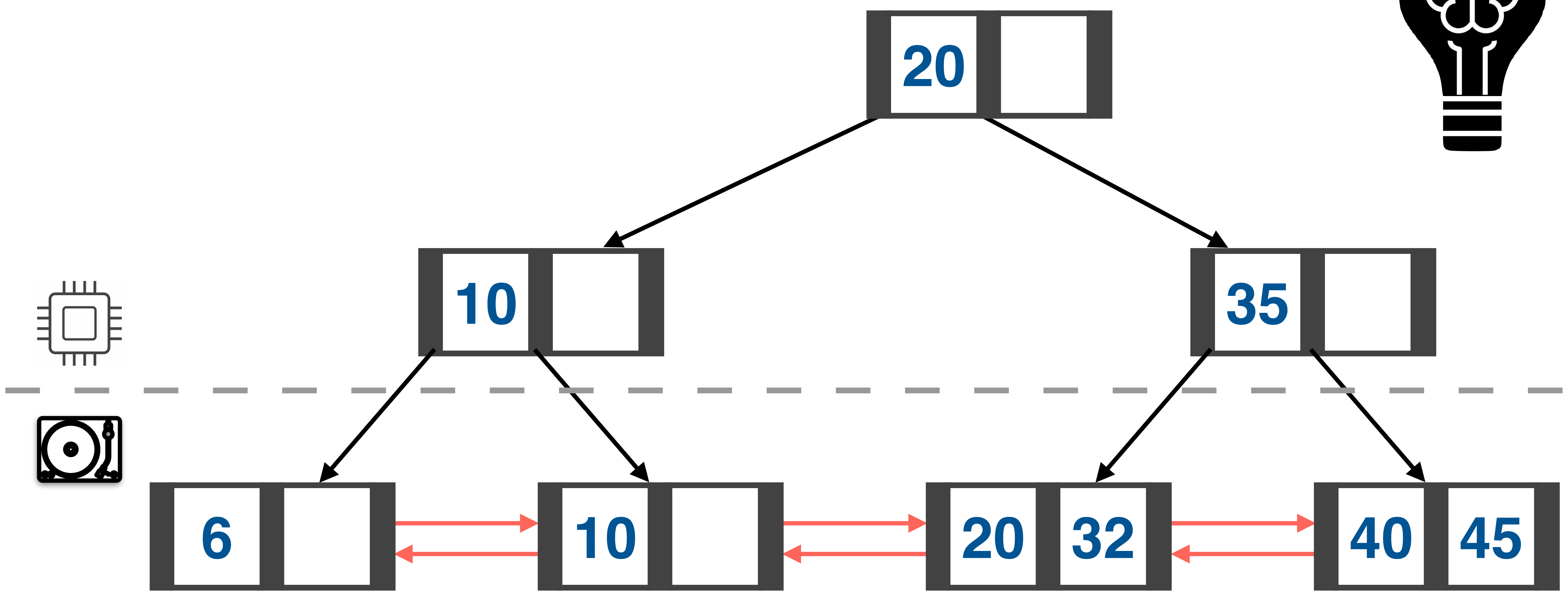
data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)^{*}$	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)^{*}$	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	?	?
Sorted array				
Log				

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

B+-trees

The go to index data structures for relational databases



Thought Experiment 2

What is the **range lookup cost** in a **buffered B+-tree**?

total entries in tree = **N**

#entries per page = **B**

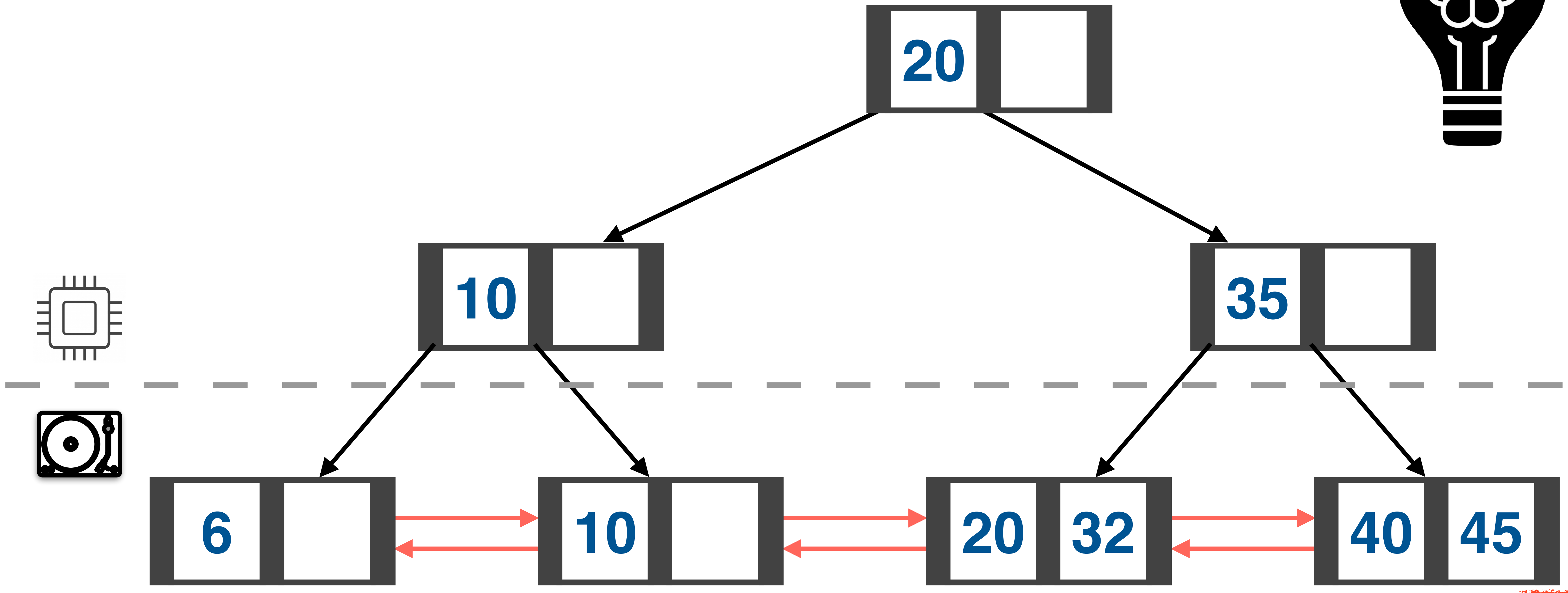
#pages in tree = N/B

s = selectivity of the range query

Cost of range lookup = $s \cdot N/B$

B^+-trees

The go to index data structures for relational databases



s = selectivity of the range query

Thought Experiment 2

What is the **range lookup cost** in a **buffered B^+-tree**?

total entries in tree = **N**

#entries per page = **B**

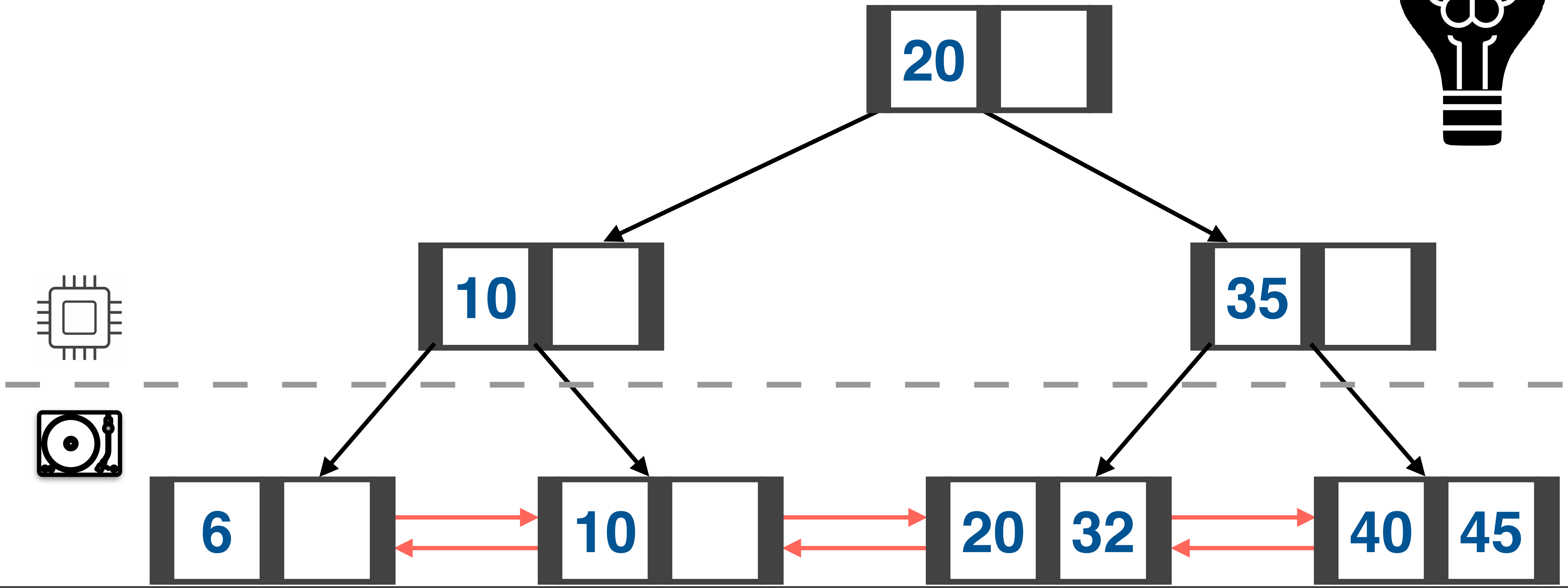
#pages in tree = N/B

Cost of range lookup = $s \cdot N/B$

On average, nodes in a B^+-tree are 67% full!

B+-trees

The go to index data structures for relational databases



Thought Experiment 2

What is the **range lookup cost** in a **buffered B+-tree**?

total entries in tree = **N**

#entries per page = **B**

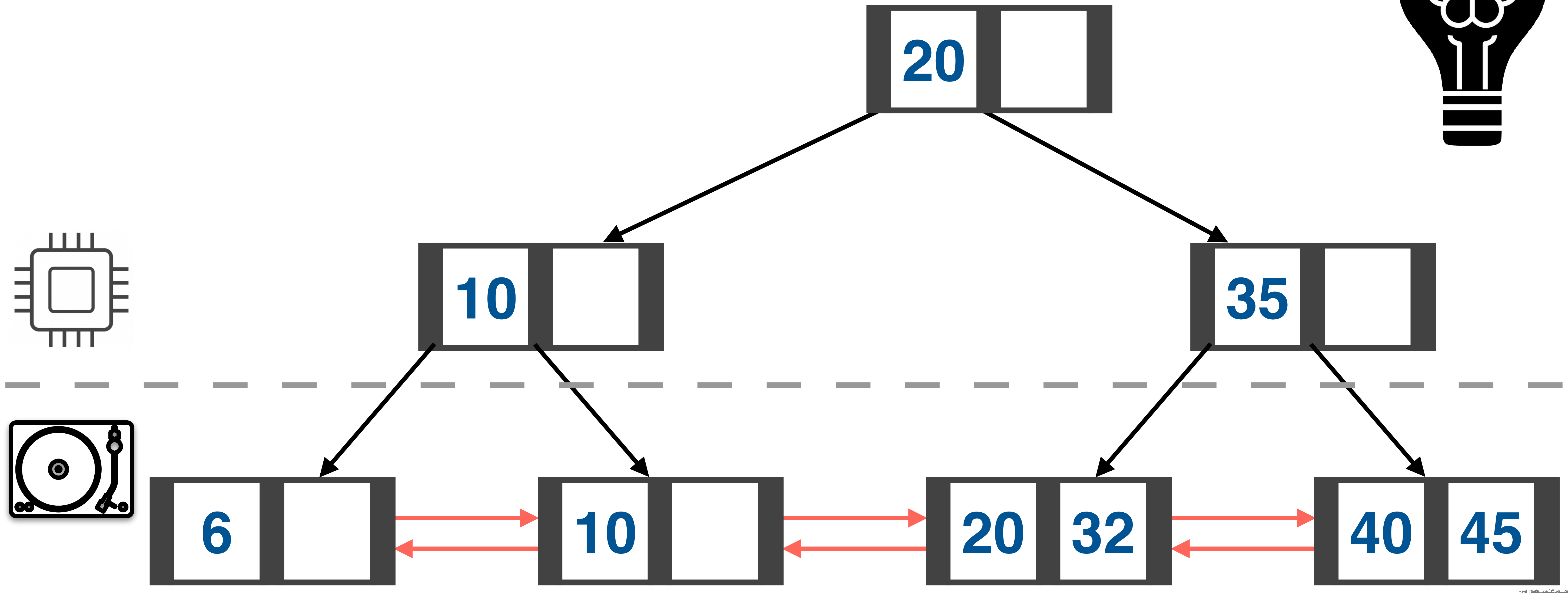
#pages in tree = N/B

On average, nodes in a B+-tree are 67% full!

On average, B+-tree have a 50% read amplification!

B+-trees

The go to index data structures for relational databases



Thought Experiment 2

What is the **range lookup cost** in a **buffered B+-tree**?

total entries in tree = **N**

#entries per page = **B**

#pages in tree = N/B

s = selectivity of the range query

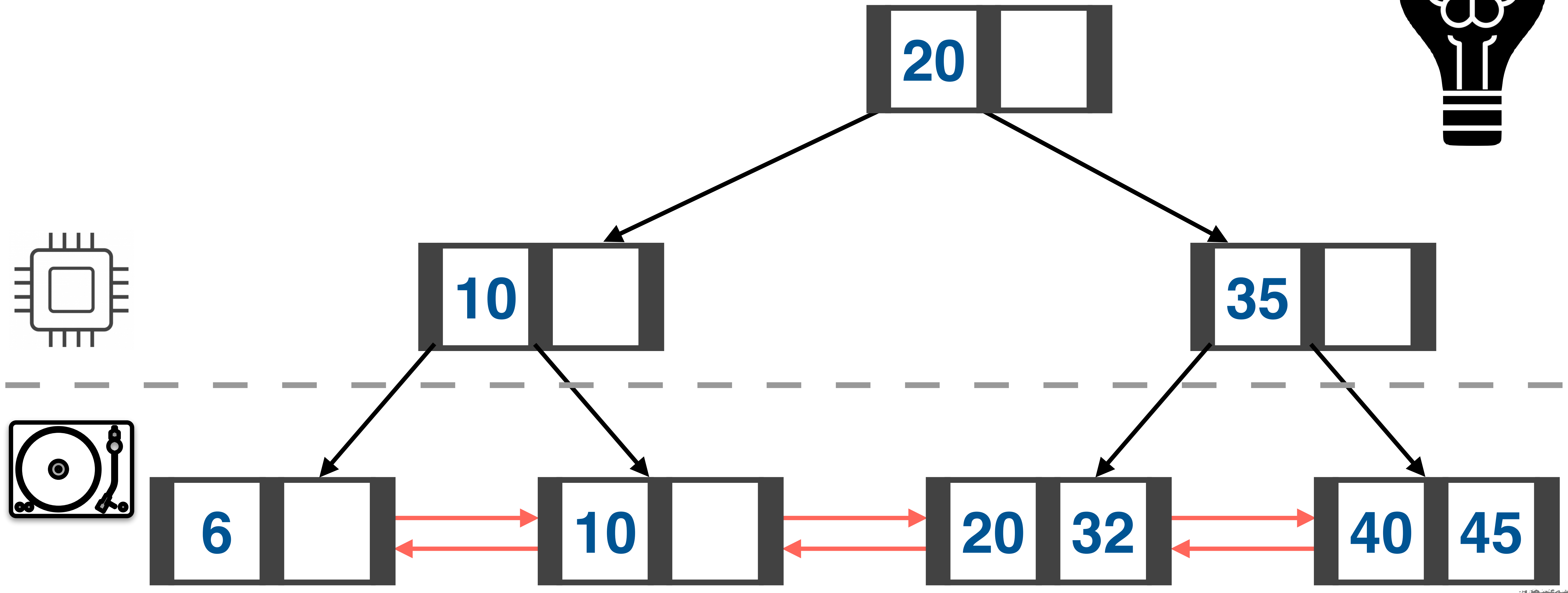
Cost of range lookup = s · N/B

x1.5

Brandeis UNIVERSITY

B+-trees

The go to index data structures for relational databases



s = selectivity of the range query

Thought Experiment 2

What is the **range lookup cost** in a **buffered B+-tree**?

total entries in tree = **N**

#entries per page = **B**

#pages in tree = N/B

Cost of range lookup = $s \cdot N/B$

x1.5

Cost of range lookup = $1.5 \cdot s \cdot N/B$

Brandeis University

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$*	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$*	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$*	?
Sorted array				
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	?
Sorted array				
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)^{*}$	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)^{*}$	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)^{*}$	$O(1)$
Sorted array	?	?	?	?
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

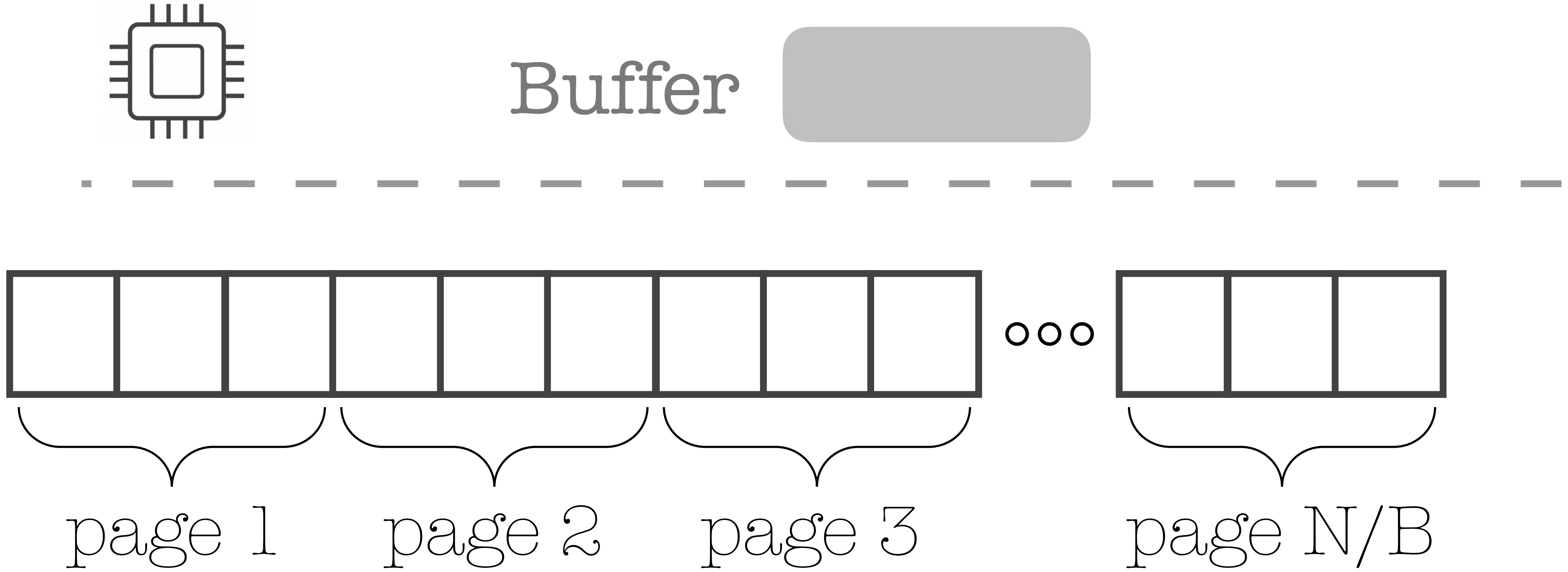
* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Sorted array

Optimizing for queries

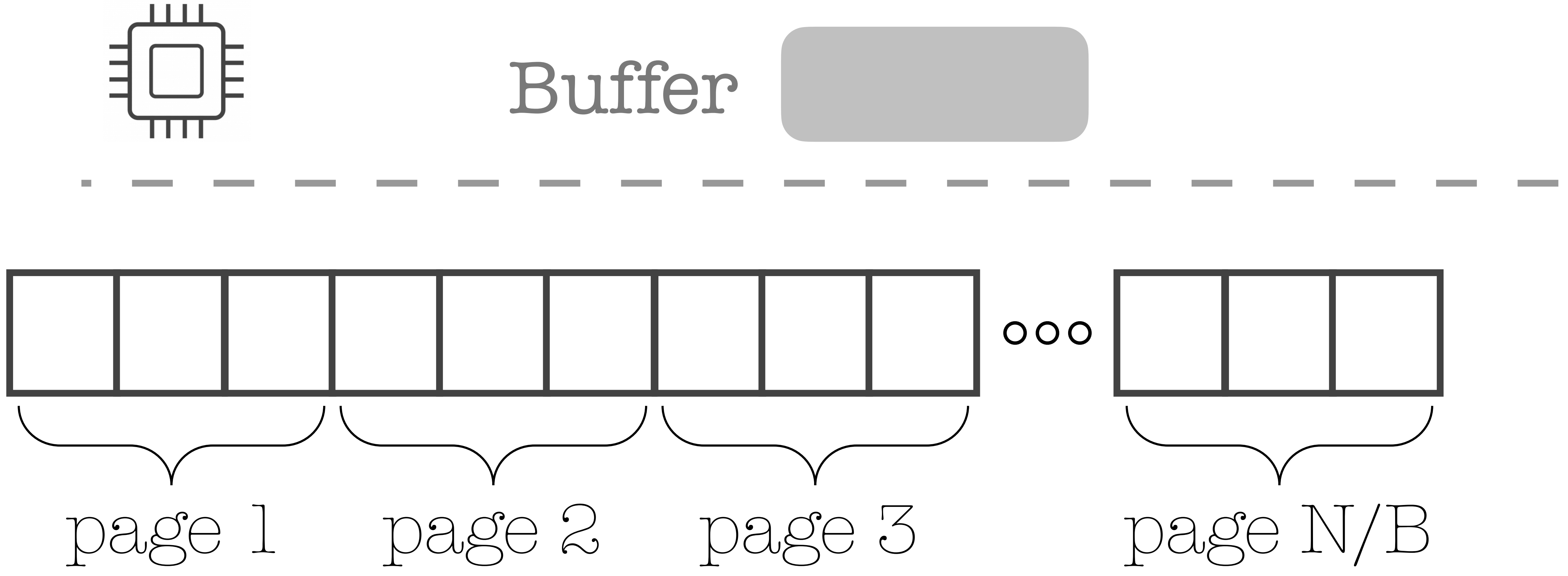


Thought Experiment 3

What is the **insert cost** in a **sorted array** with **a buffer**?

Sorted array

Optimizing for queries



Thought Experiment 3

What is the **insert cost** in a **sorted array** with **a buffer**?

read + write all N/B pages

for uniform data distribution

B entries are written to disk per I/O

Cost of ingestion = N/B^2

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+\phi \cdot L)$*	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+\phi \cdot L \cdot T)$*	$O(s \cdot N / B)$	$O(L \cdot T)$
B^+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$*	$O(1)$
Sorted array	$O(N/B^2)$	?	?	?
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B^+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	?	?	?
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$*	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$*	$O(s \cdot N / B)$	$O(L \cdot T)$
B^+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$*	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	?	?
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "$1 +$" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	?	?
Log				

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "$1 +$" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	?
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	?
Log				

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "$1 +$" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+ \phi \cdot L)$*	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+ \phi \cdot L \cdot T)$*	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$*	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log				

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "$1 +$" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	?	?	?	?

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+\phi \cdot L)$*	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+\phi \cdot L \cdot T)$*	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$*	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	?	?	?

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	?	?	?

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+ \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+ \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	?	?

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+ \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+ \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	?	?

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "$1 +$" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	?

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	?

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B^+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+\phi \cdot L)$*	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+\phi \cdot L \cdot T)$*	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$*	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+ \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+ \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+ \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1+ \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B^+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR =** ϕ
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* **Monkey** shaves off the "L" factor from the cost

* 1.5x costlier than LSM-trees

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

data structure	**ingestion** cost	**point** lookup cost *	**long range** lookup cost	**short range** lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1 + \phi \cdot L)$ *	$O(s \cdot N / B)$	$O(L)$
Tiered LSM-tree	$O(L / B)$	$O(1 + \phi \cdot L \cdot T)$ *	$O(s \cdot N / B)$	$O(L \cdot T)$
B+-tree	$O(1)$	$O(1)$	$O(s \cdot N / B)$ *	$O(1)$
Sorted array	$O(N/B^2)$	$O(1)$	$O(s \cdot N / B)$	$O(1)$
Log	$O(1/B)$	$O(N/B)$	$O(N/B)$	$O(N/B)$

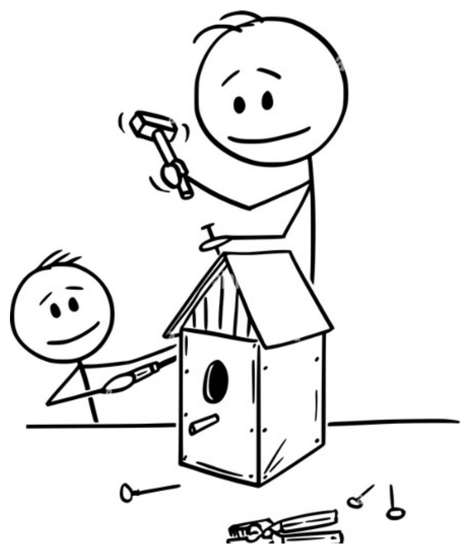
* with **fence pointers & Bloom filter** with **FPR = ϕ**
cost for **non-empty lookups** (remove "1 +" to get cost of empty lookups)

* 1.5x costlier than LSM-trees

* **Monkey** shaves off the "L" factor from the cost

Brandeis UNIVERSITY

Projects



Projects

Let's get our hands dirty

Class Project

Groups of 2

Out **tonight**; due in **December**

Multiple milestones in between

Systems project

Research project

Brandeis UNIVERSITY

Projects

Let's get our hands dirty

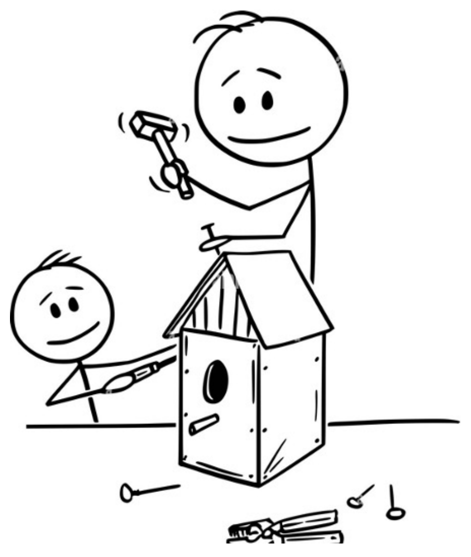
Class Project

Systems project

Research project

implementation-heavy project

develop, **experiment**, and **analyze**



Projects

Let's get our hands dirty

Class Project

Systems project

implementation-heavy project

develop, **experiment**, and **analyze**

Research project

pick a topic (list available *tonight*)

design, **develop**, and **analyze**

large-scale experimentation

Class project: **Theme**

Working with state-of-the-art systems

NoSQL key-value stores

RocksDB

AsterixDB

cassandra

APACHE HBASE

MEMCACHED

Google BigTable

amazon DynamoDB

redis

Class project: **Theme**

Working with state-of-the-art systems

NoSQL key-value stores

RocksDB

cassandra

APACHE HBASE

MEMCACHED

AsterixDB

Google BigTable

amazon DynamoDB

redis

Systems Project **1**

Developing your way through!

Implementing an LSM-based KV store

Data layout: **leveling** & **tiering**

File structure: **sorted run** as **one or more files**; **files** have **multiples pages**

Auxiliary data structures: **Bloom filters** & **fence pointers**

Granularity: filters **per sorted run / file**; fence pointers **per page**

APIs: **insert**, **update**, **delete**, **point** and **range queries**

Parameterize everything!

Report **all relevant performance numbers!**

Systems Project **2**

Developing your way through!

Implementing Compactions on RocksDB

Working with **RocksDB** — **C++**

Data layout: **leveling**, **tiering**, **hybrids**

Data movement policy: **partial-level**, **full-level**, **tiered**

File picking policy: **coldest**, **oldest**, **most tombstones**

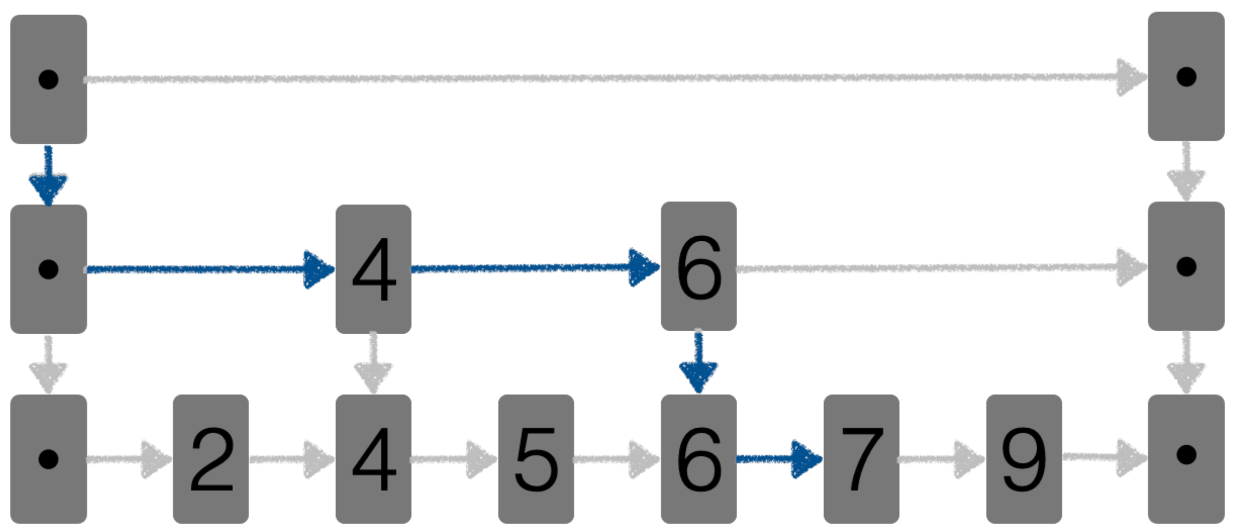
Benchmark performance for different workloads

Reserach projects

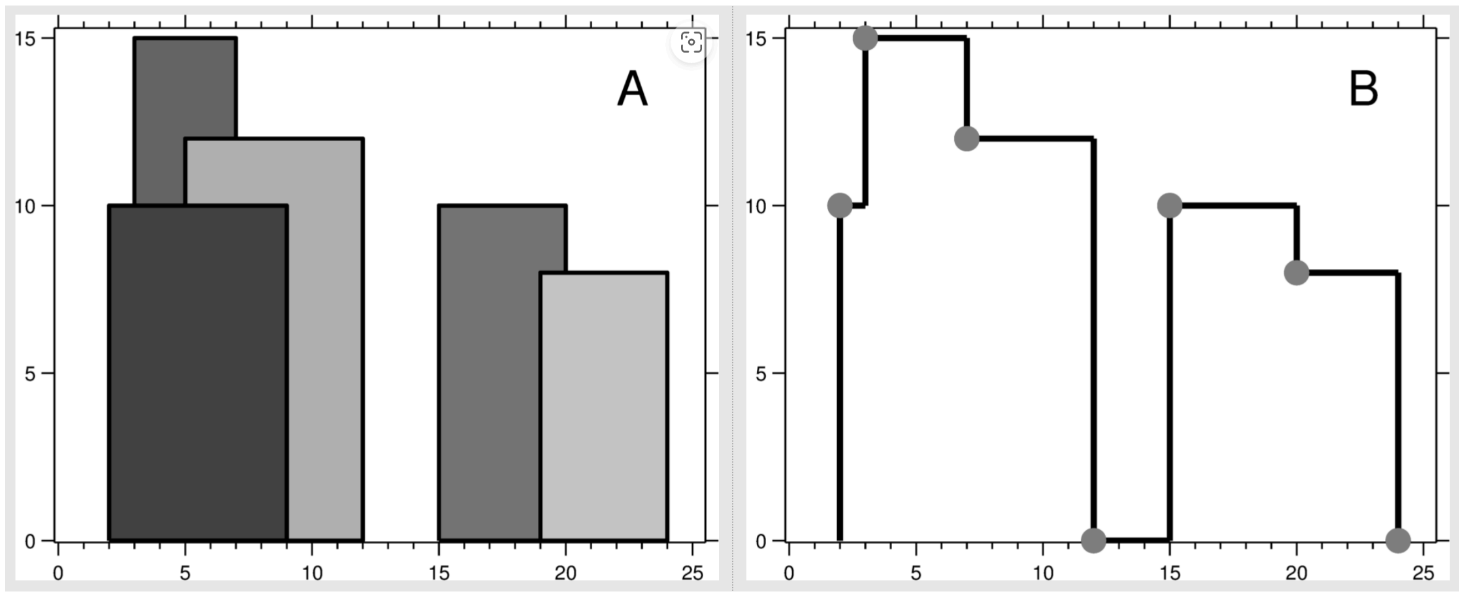
Solving problems on your way through!

Reserach projects

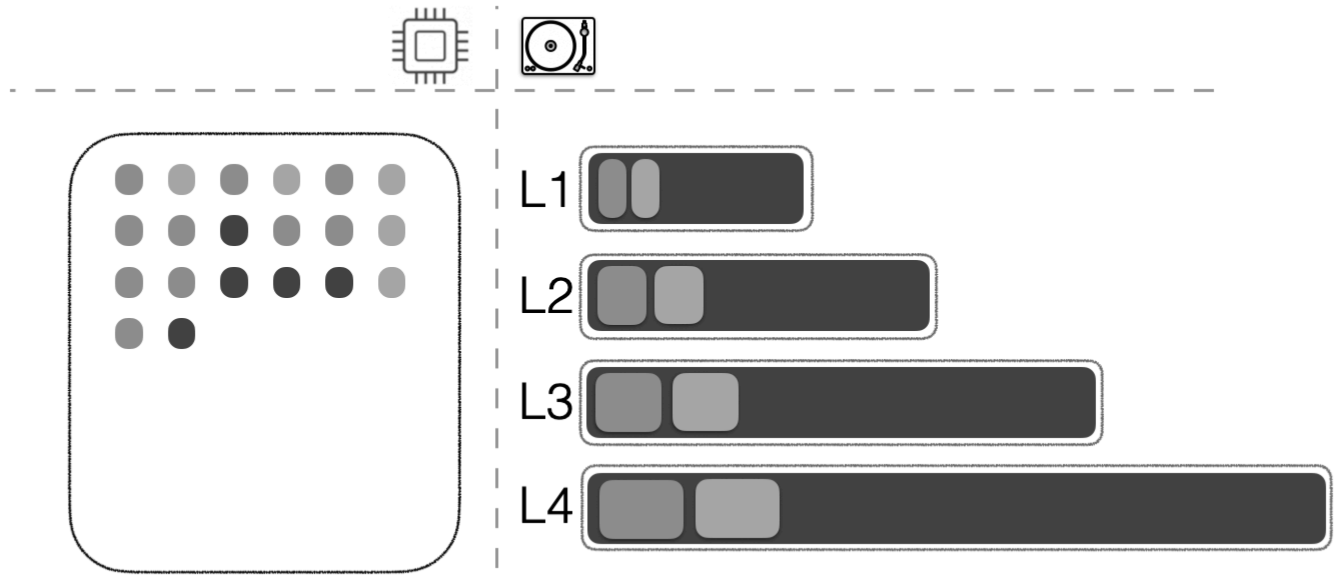
Solving problems on your way through!



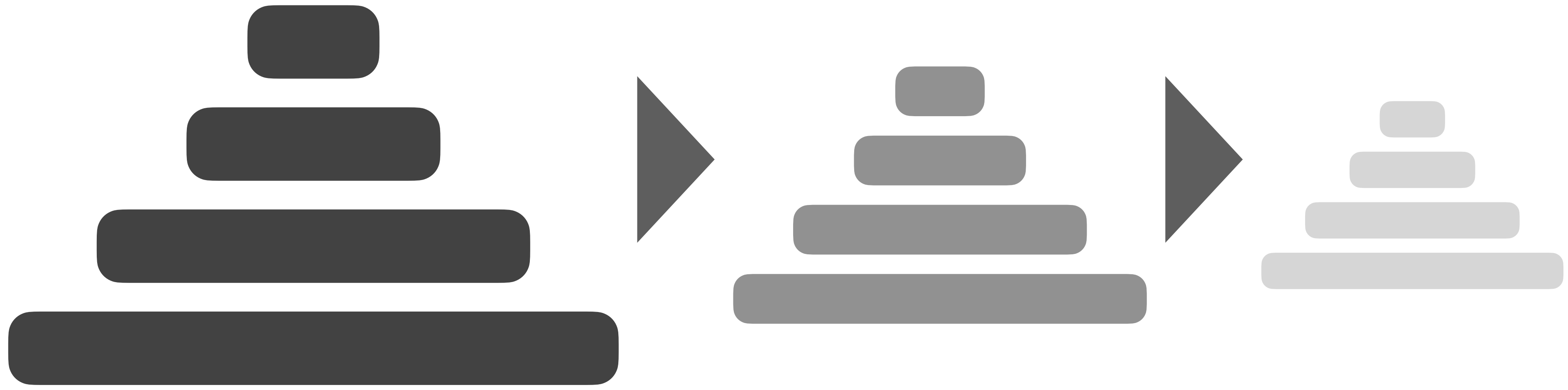
Self-designing LSM Buffer



Range Deletes in LSM



Optimal Caching in LSM

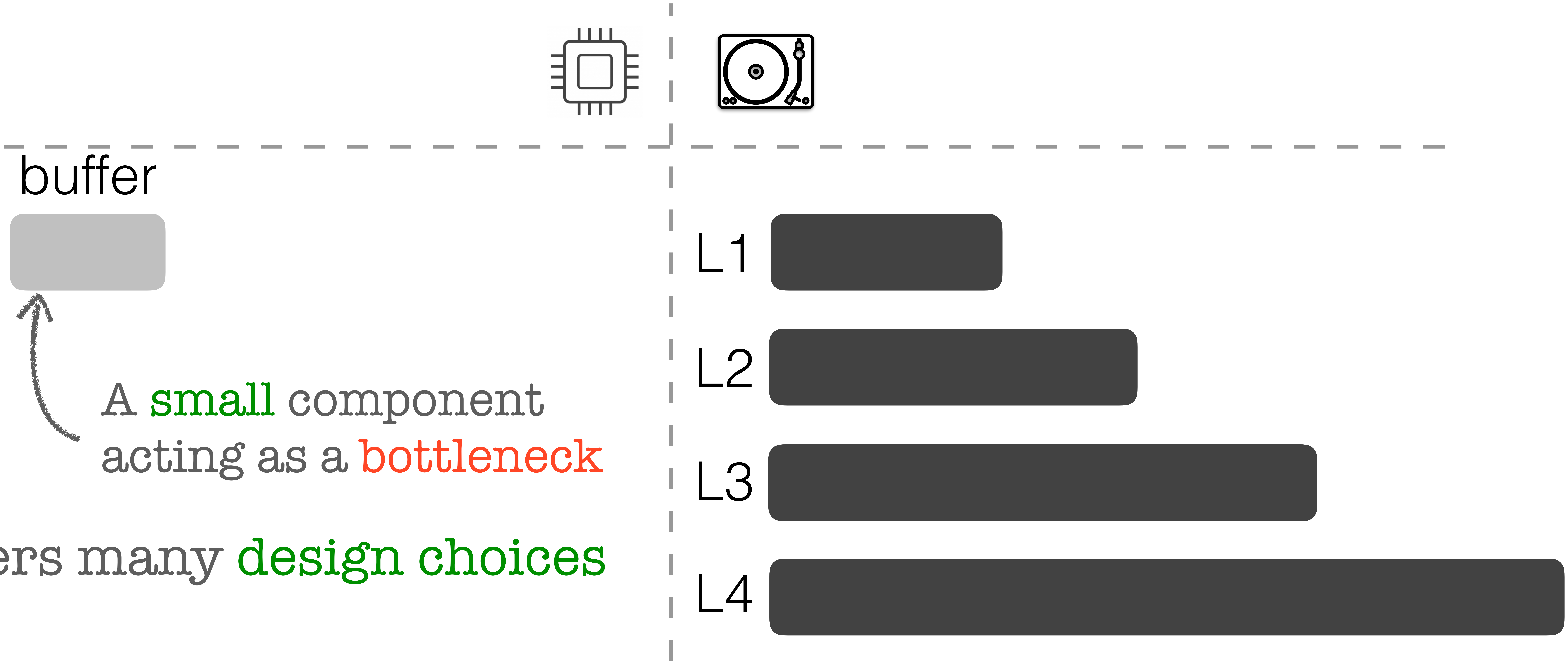


Time-Traveling LSM

Brandeis UNIVERSITY

Buffer Design

Buffer Design



A small component acting as a bottleneck

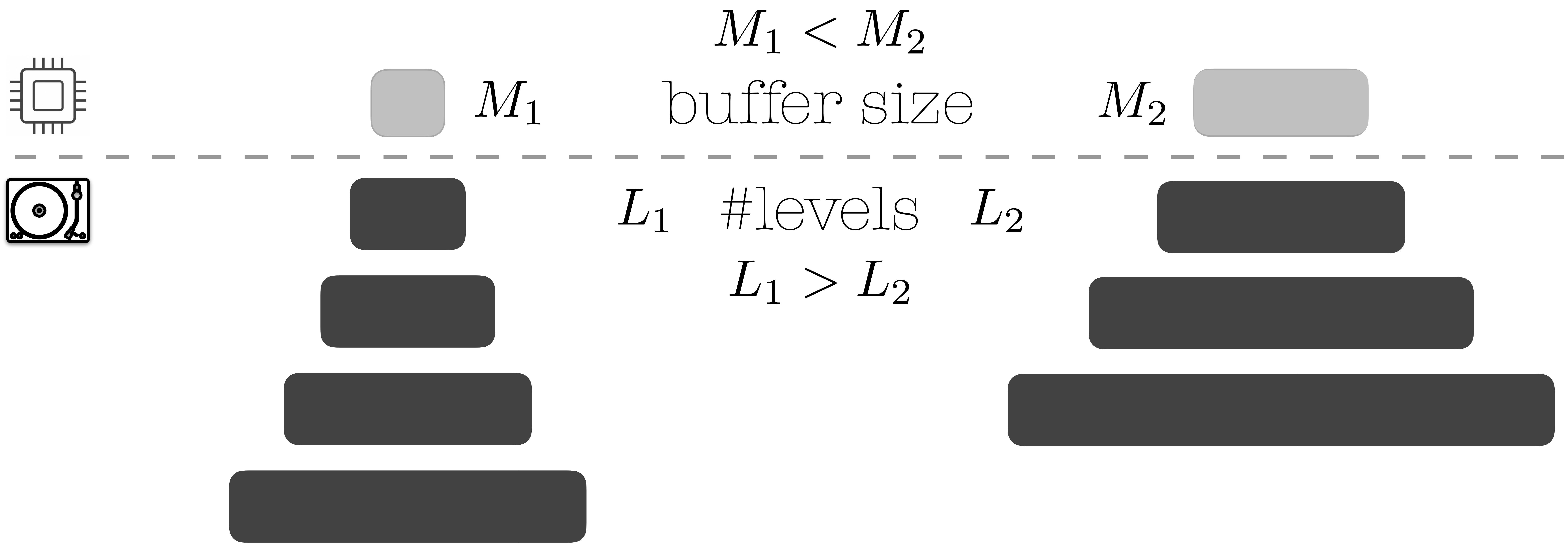
Offers many design choices

P: pages in buffer
B: entries/page

Buffer Implementation

	vector	**skiplist**	**hashmap**
ingestion cost	$\mathcal{O}(1)$	$\mathcal{O}(log(P \cdot B))$	$\mathcal{O}(1)$
point query cost	$\mathcal{O}(P \cdot B)$	$\mathcal{O}(log(P \cdot B))$	$\mathcal{O}(1)$
	Ingestion-only workloads	Mixed workloads	I/O-bound workloads

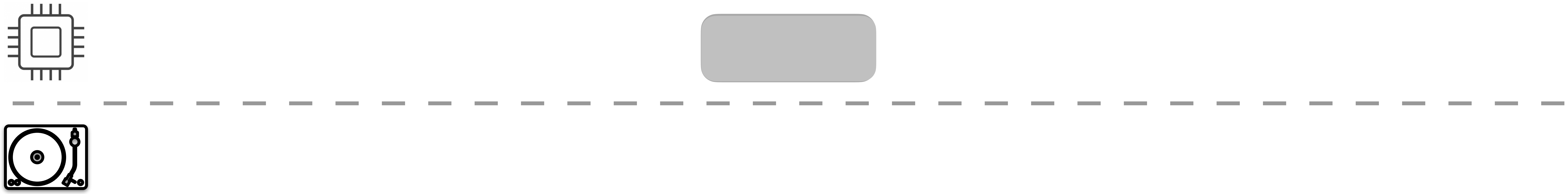
Size of the Buffer



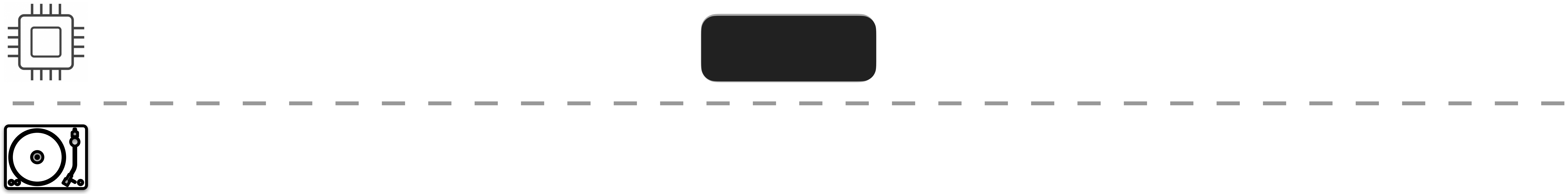
- frequent flushes
- smaller but more levels
- poor read performance

- fewer larger levels
- good for reads
- high tail latency

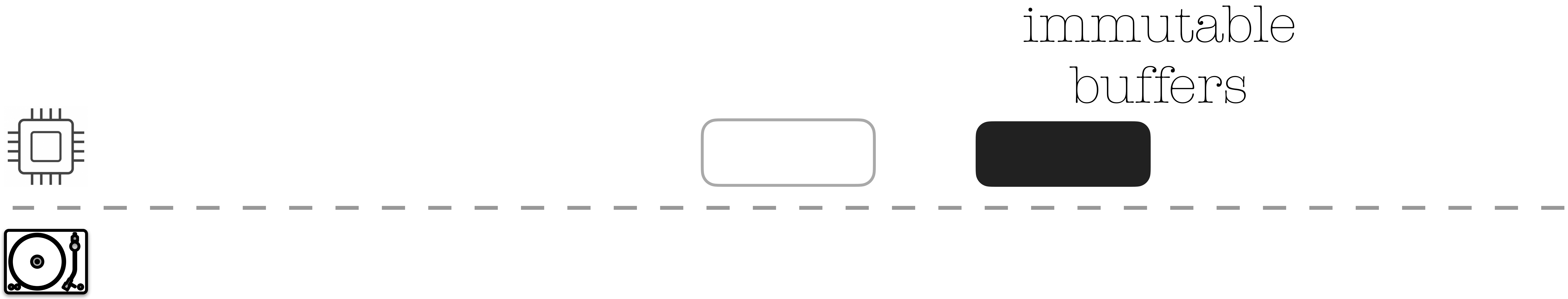
#Buffer Components



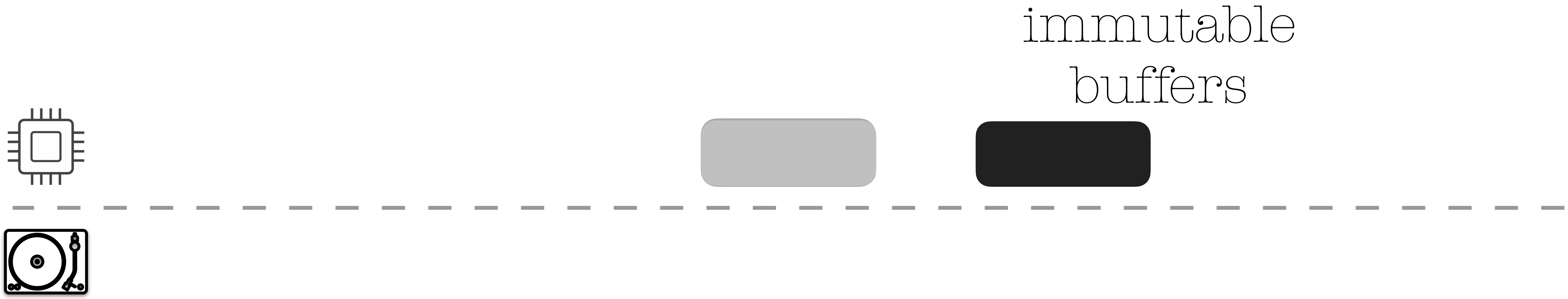
#Buffer Components



#Buffer Components

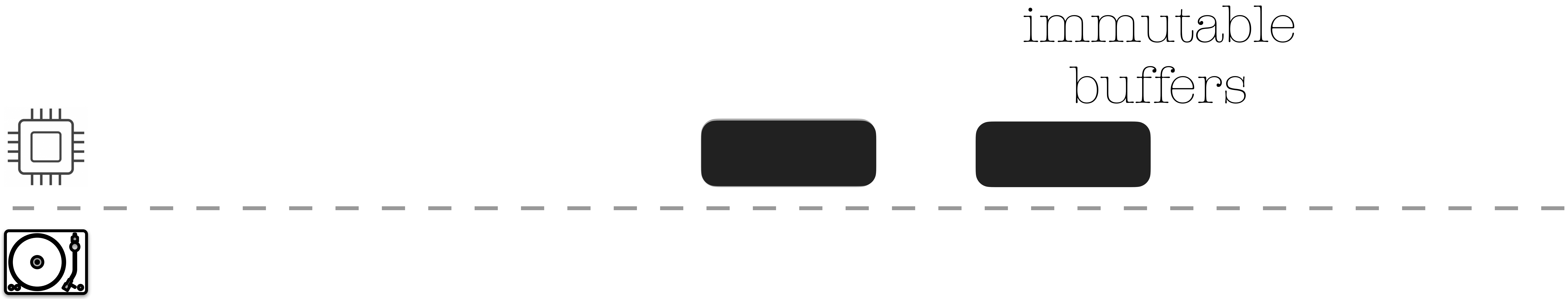


#Buffer Components

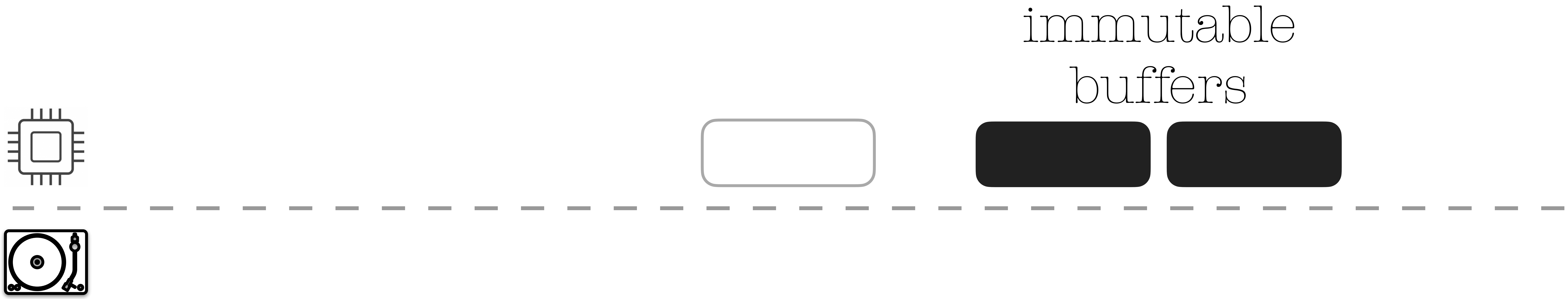


Brandeis
UNIVERSITY

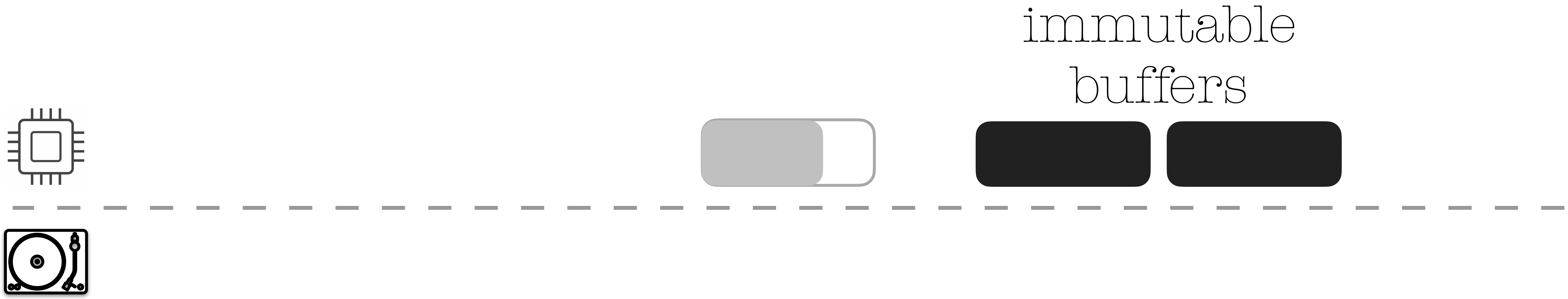
#Buffer Components



#Buffer Components

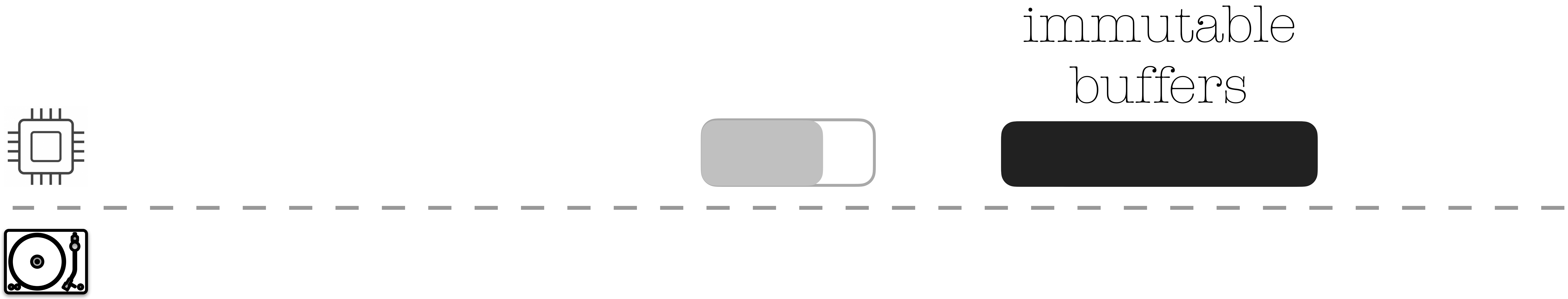


#Buffer Components



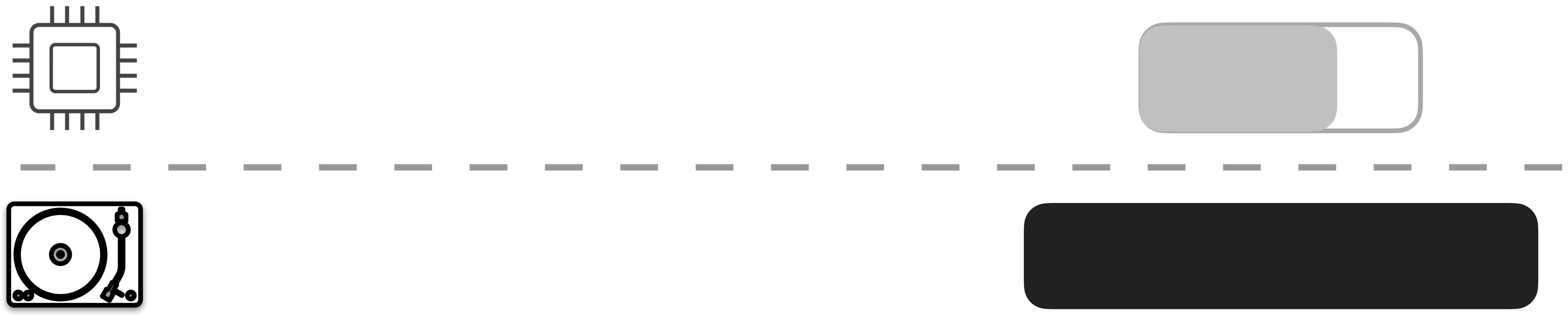
Brandeis
UNIVERSITY

#Buffer Components



#Buffer Components

immutable buffers



lazy flushing

- avoid write stalls
- improved ingestion throughput
- better bandwidth utilization
- requires more memory

Research Project **1**

Solving problems on your way through!

Self-designing LSM Buffer

Which buffer implementation to choose?

How to tune the particular buffer implementation?

What if the **workload changes**?

What if the **application requirements** change?

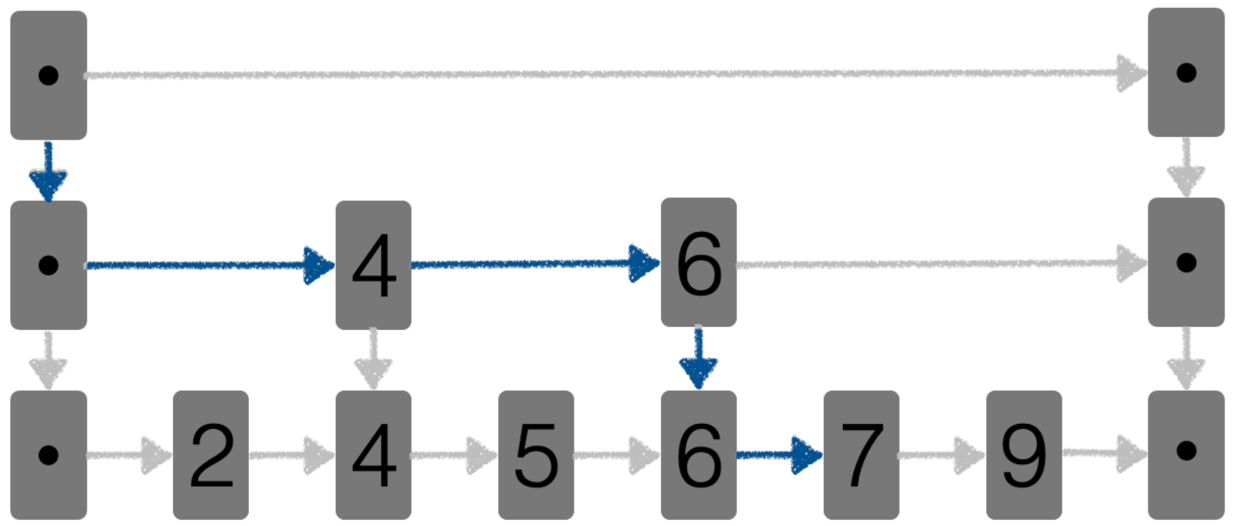
Benchmark performance for different workloads & performance goals

Design an **ML-based optimizer**

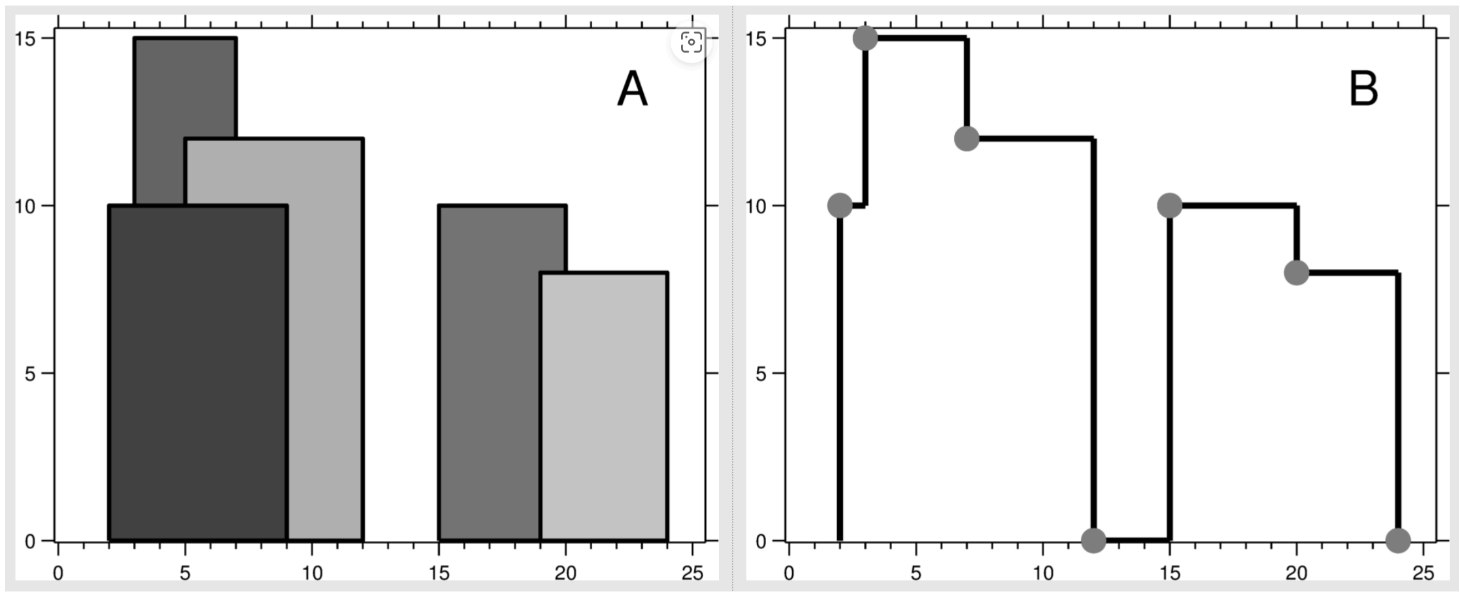
Integrate the **self-designing buffer** with RocksDB

Reserach projects

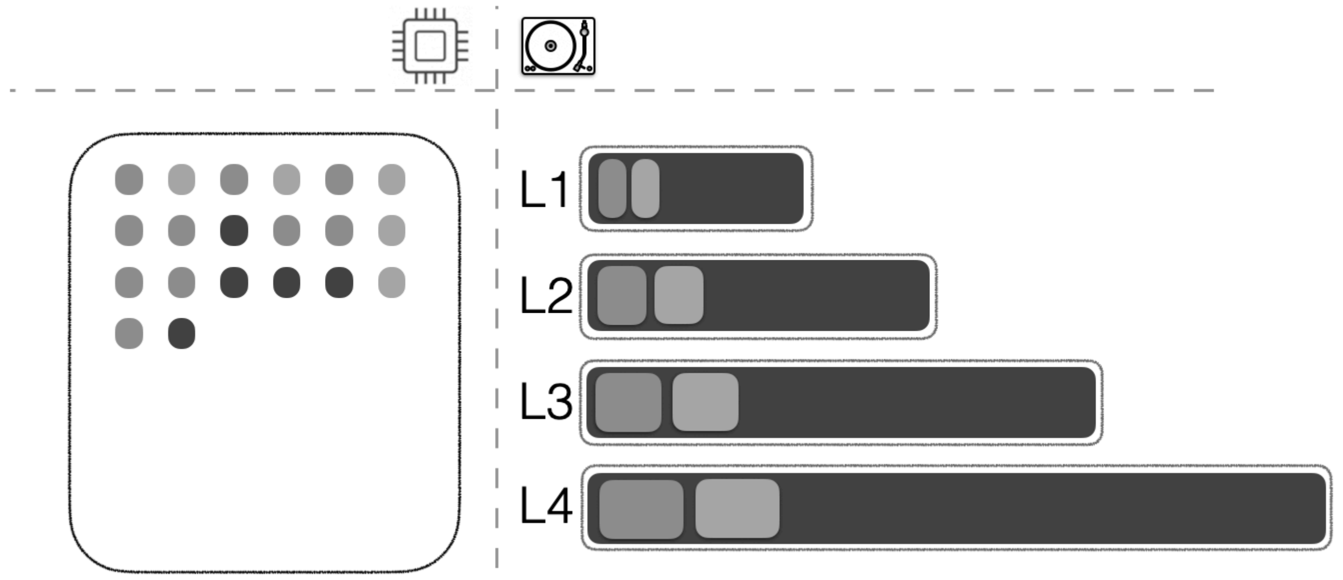
Solving problems on your way through!



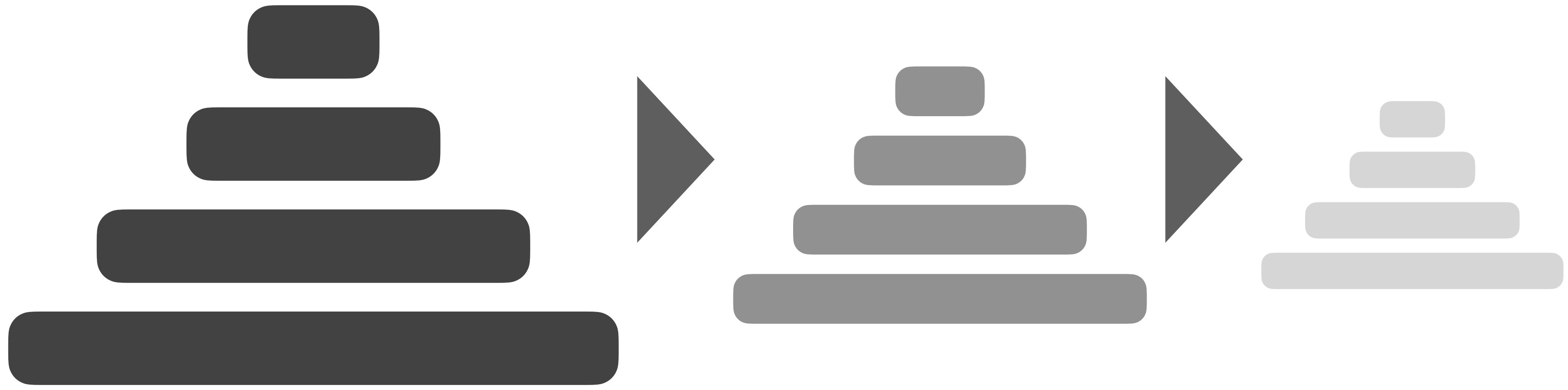
Self-designing LSM Buffer



Range Deletes in LSM



Optimal Caching in LSM

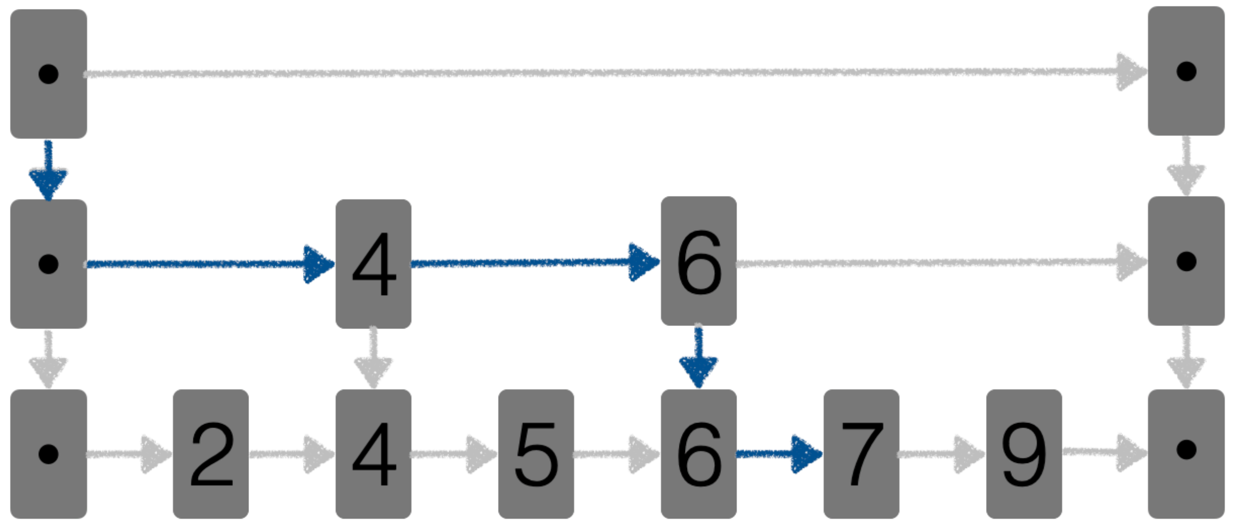


Time-Traveling LSM

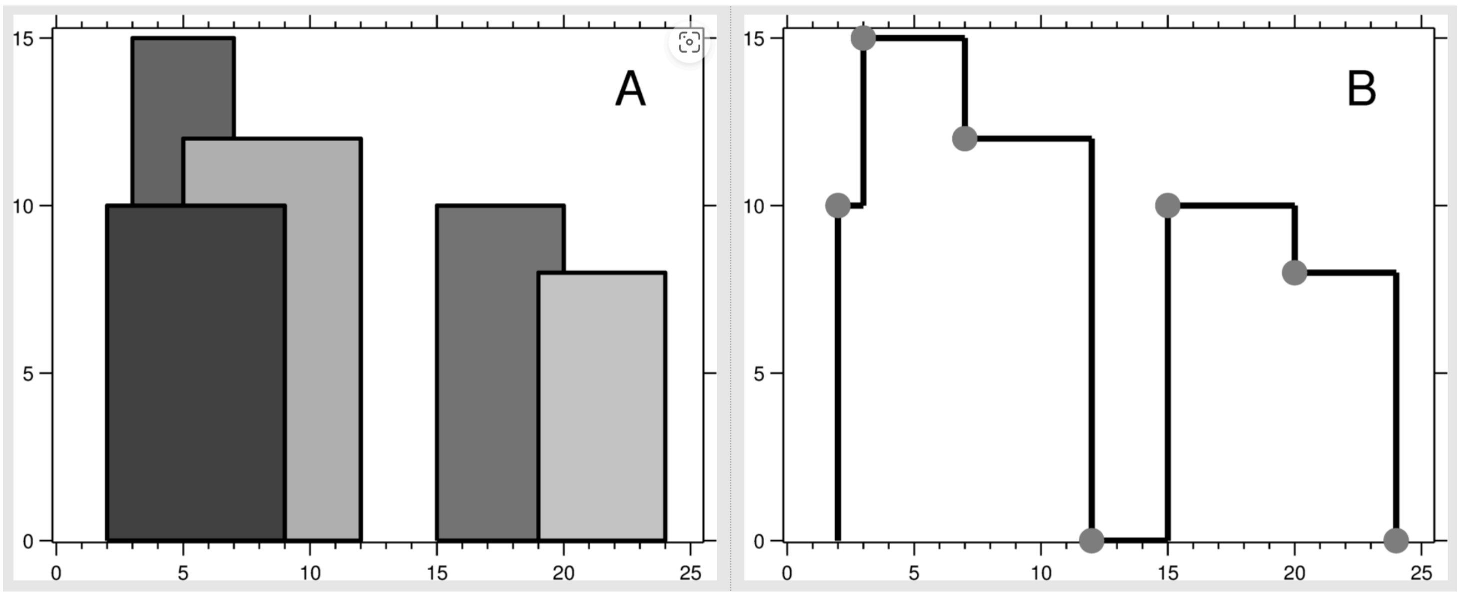
Brandeis
UNIVERSITY

Reserach projects

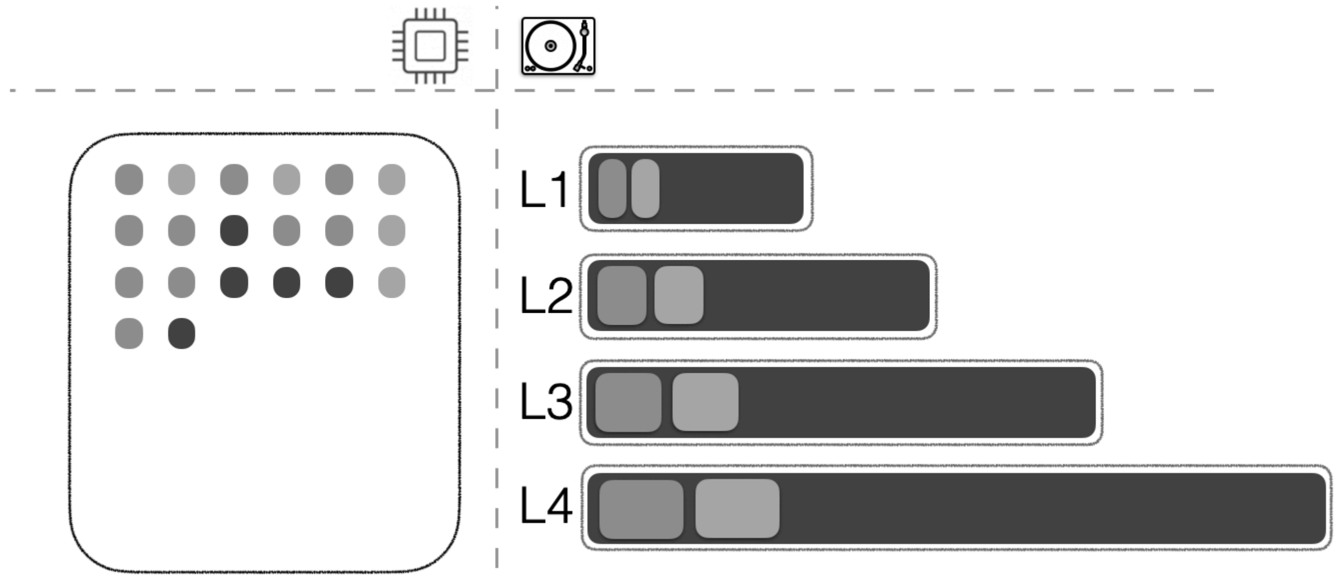
Solving problems on your way through!



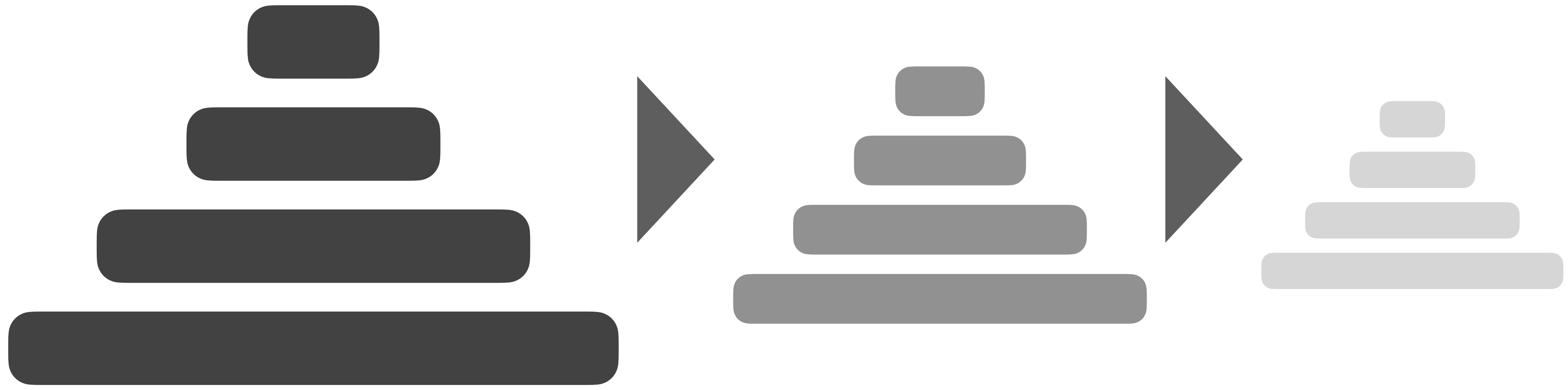
Self-designing LSM Buffer



Range Deletes in LSM



Optimal Caching in LSM

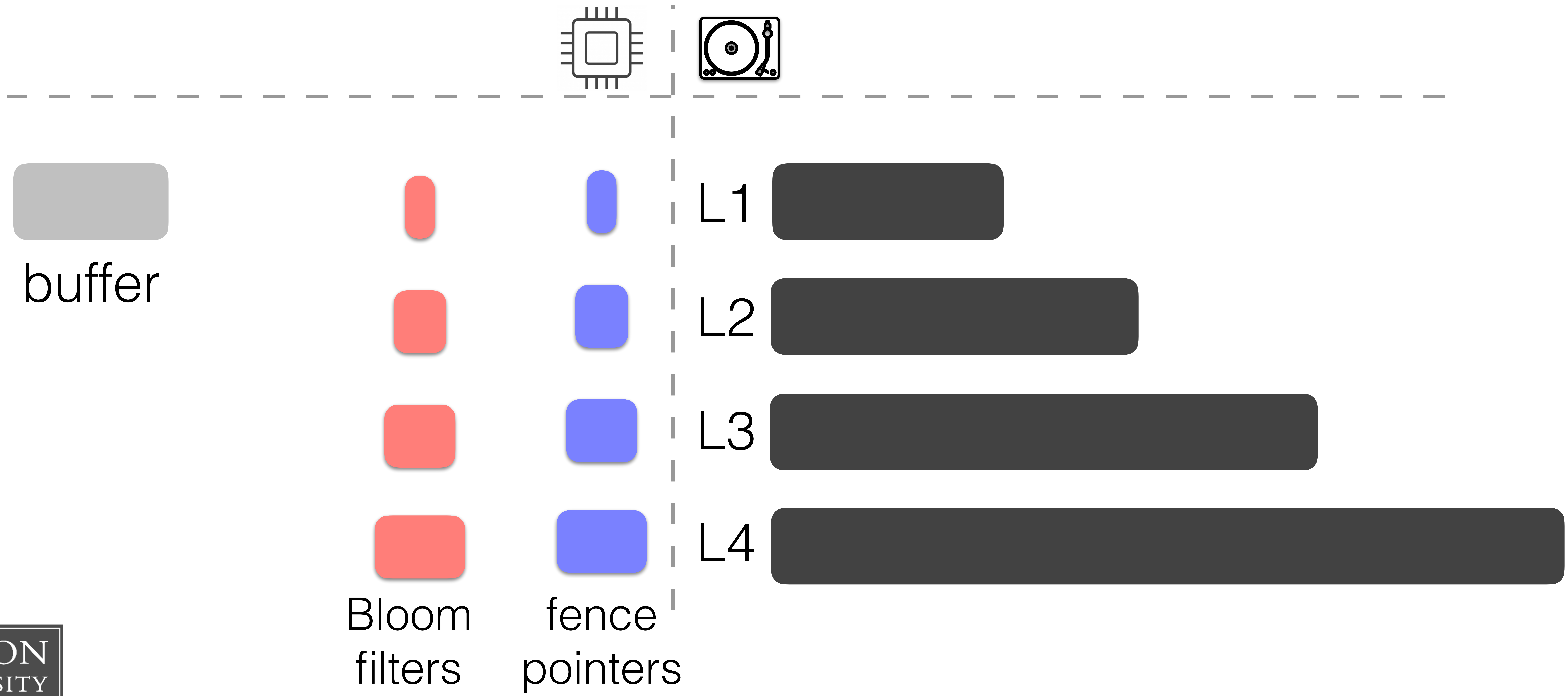


Time-Traveling LSM

Brandeis UNIVERSITY

Block cache

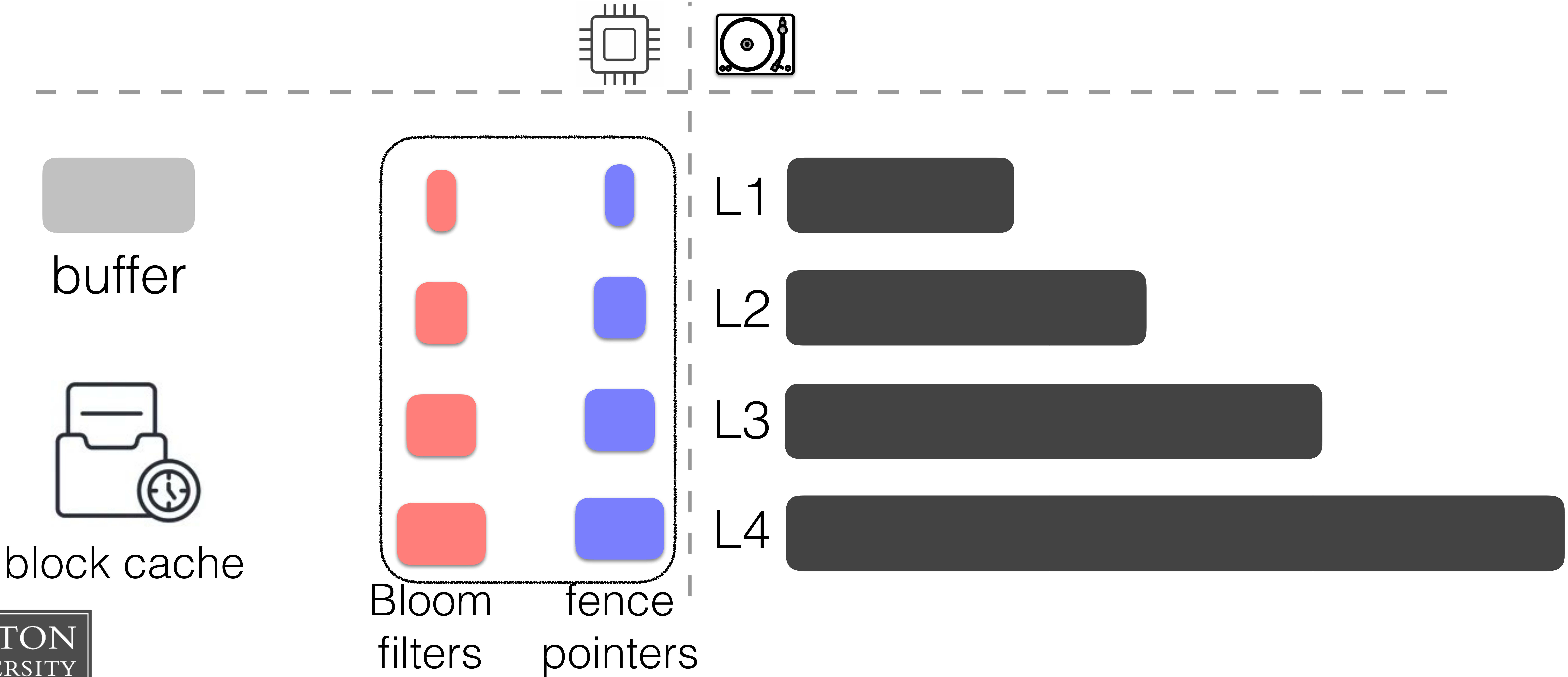
Saving trips to disk



BOSTON UNIVERSITY

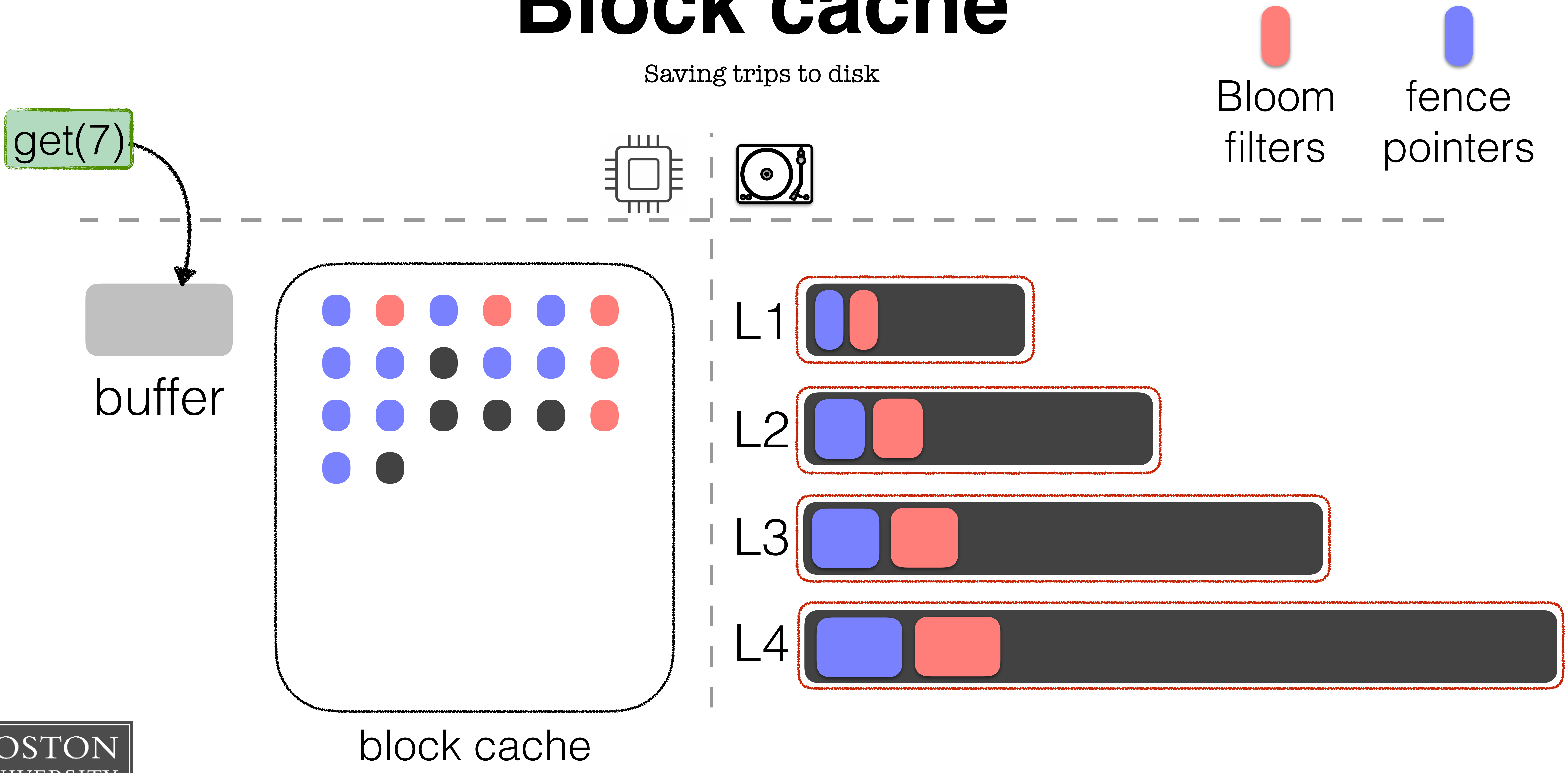
Block cache

Saving trips to disk



Block cache

Saving trips to disk



Research Project **2**

Solving problems on your way through!

Optimal Caching in LSM

Memory is **finite**!

How useful is **block-based caching**?

Which blocks to cache: **index**, **filter**, **data**?

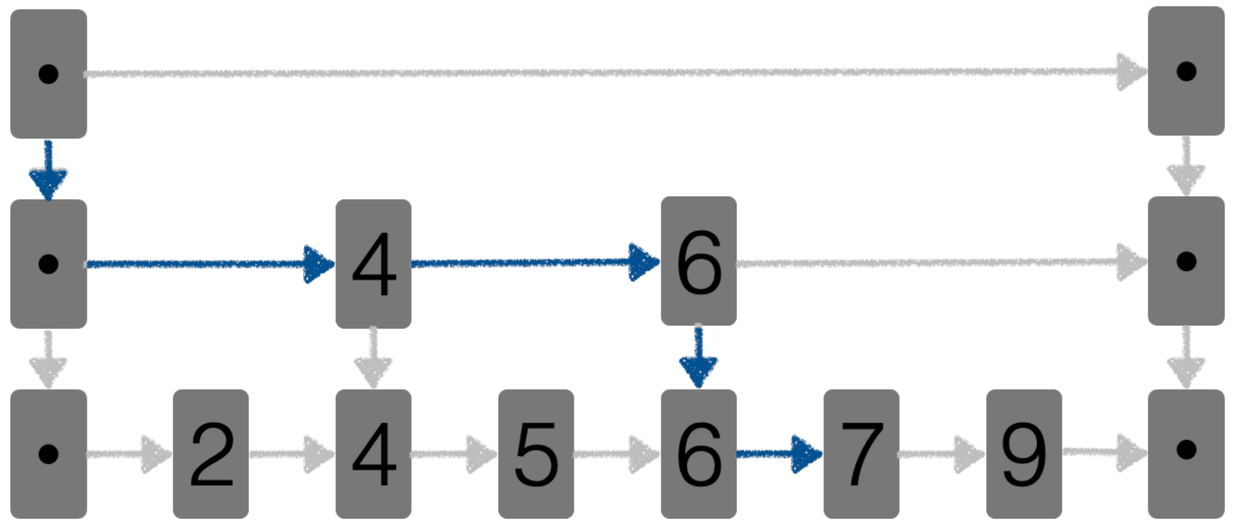
What is the impact of **pinning blocks**?

Which blocks to **evict**?

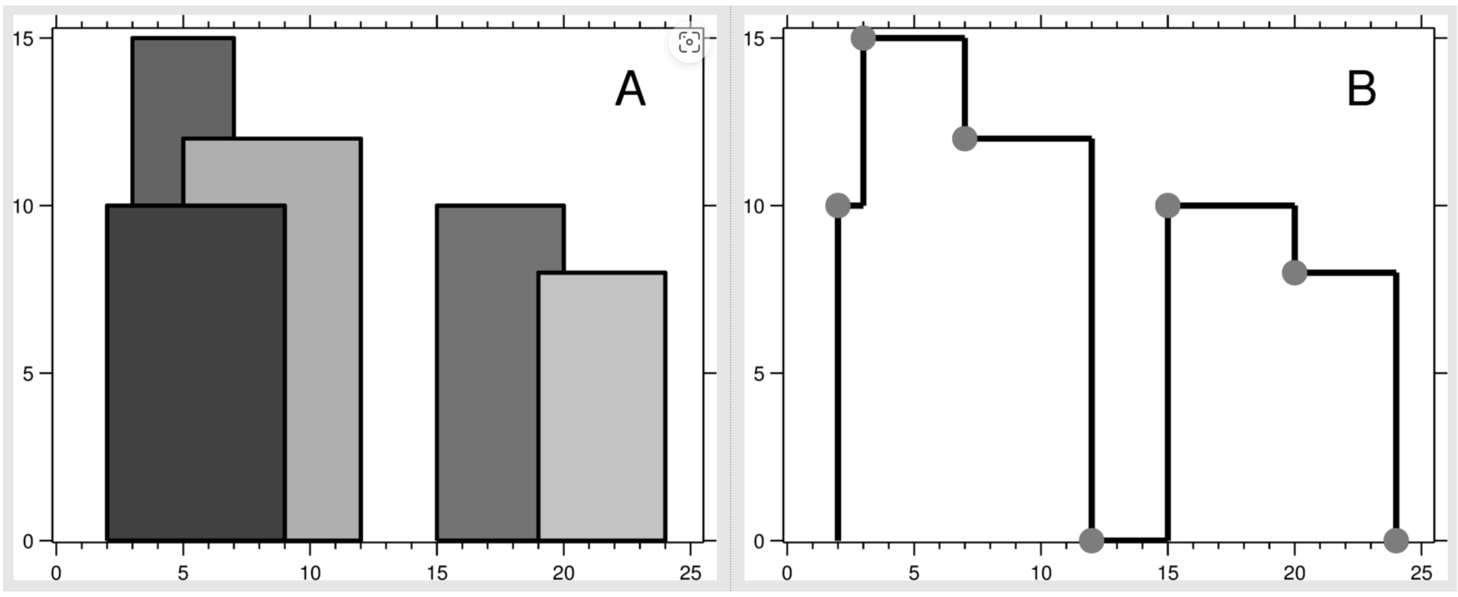
Benchmark performance for different caching schemes

Reserach projects

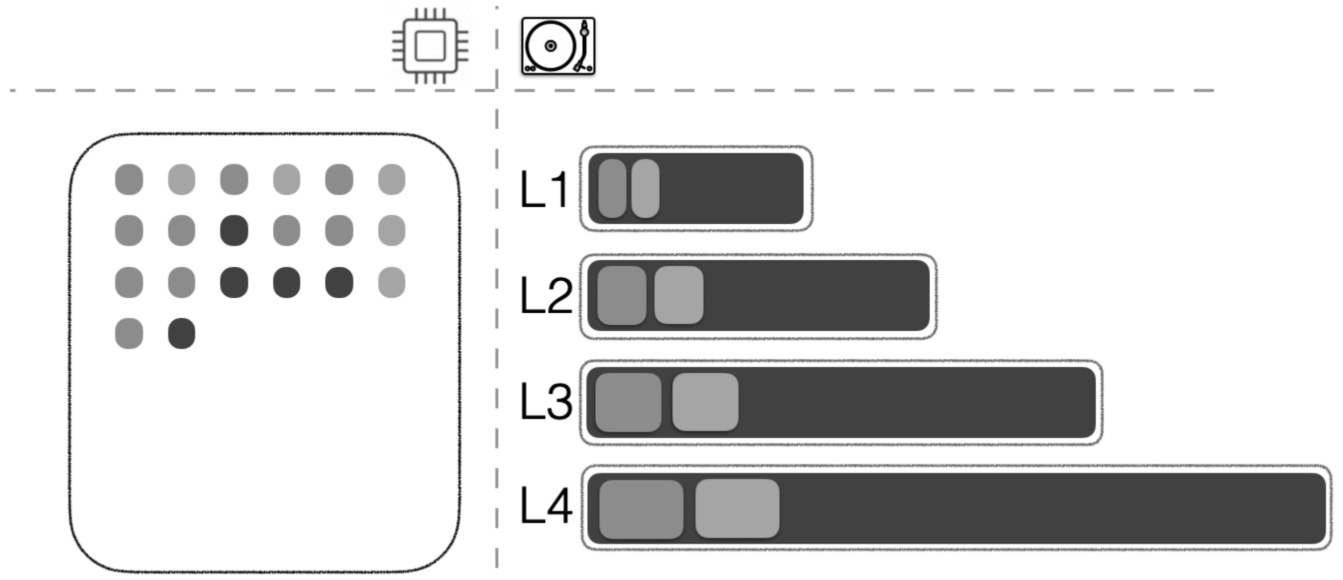
Solving problems on your way through!



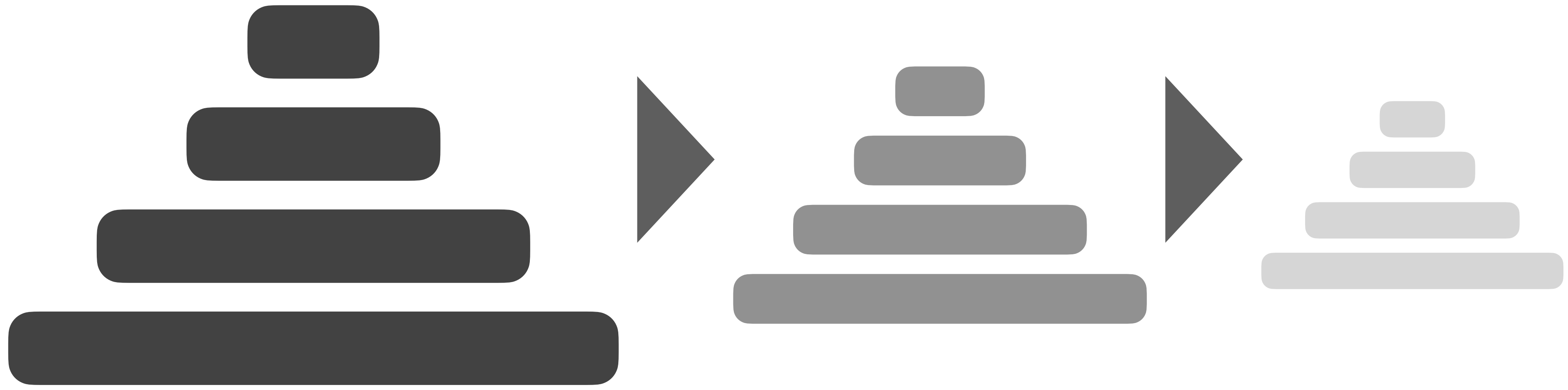
Self-designing LSM Buffer



Range Deletes in LSM



Optimal Caching in LSM

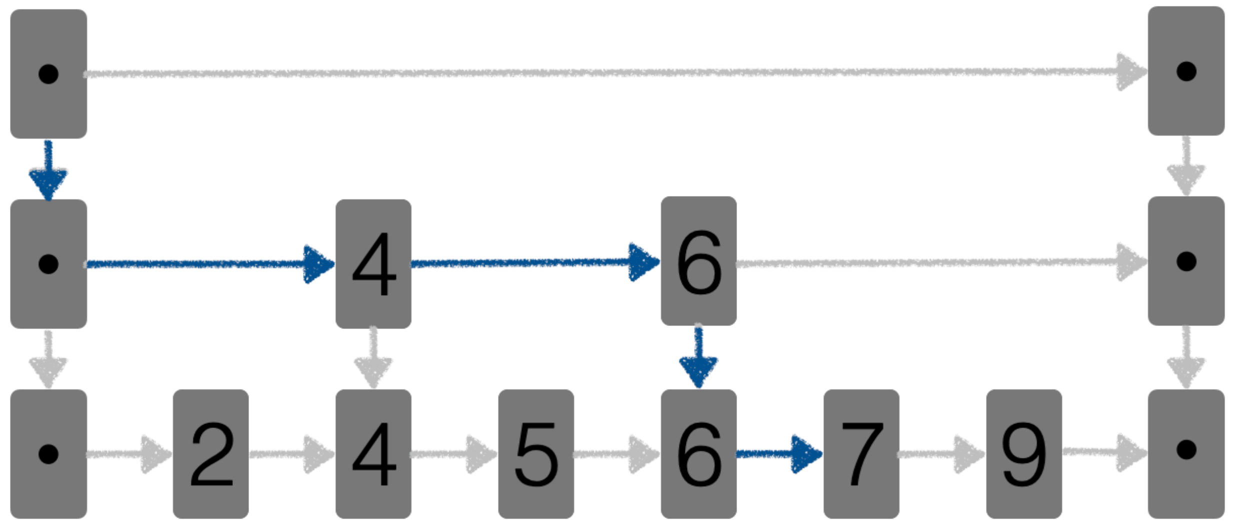


Time-Traveling LSM

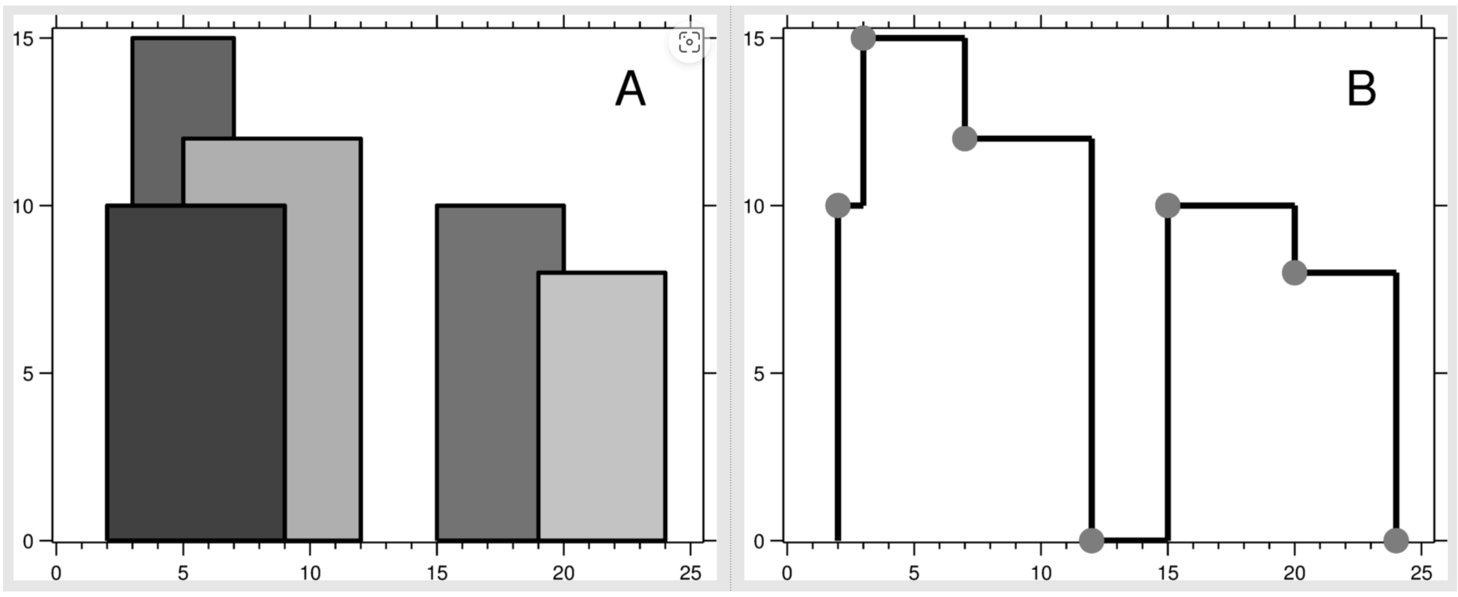
Brandeis UNIVERSITY

Reserach projects

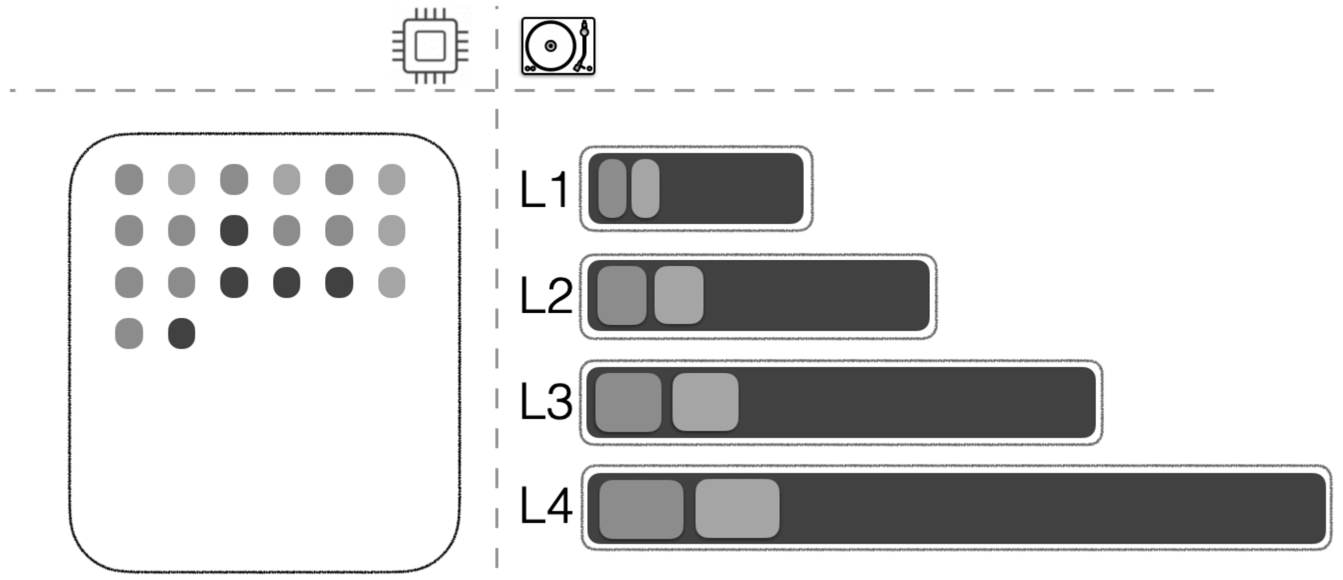
Solving problems on your way through!



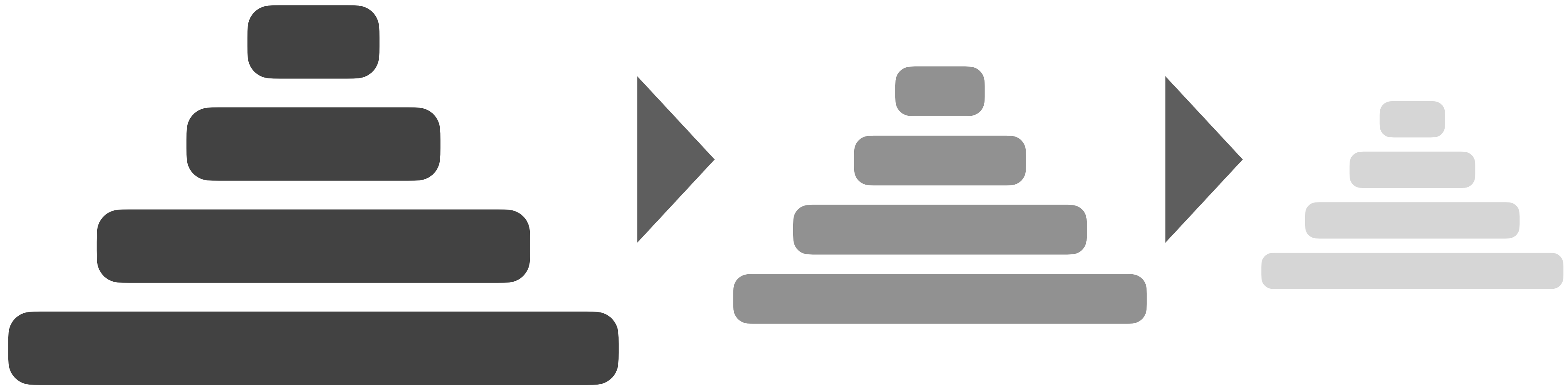
Self-designing LSM Buffer



Range Deletes in LSM



Optimal Caching in LSM

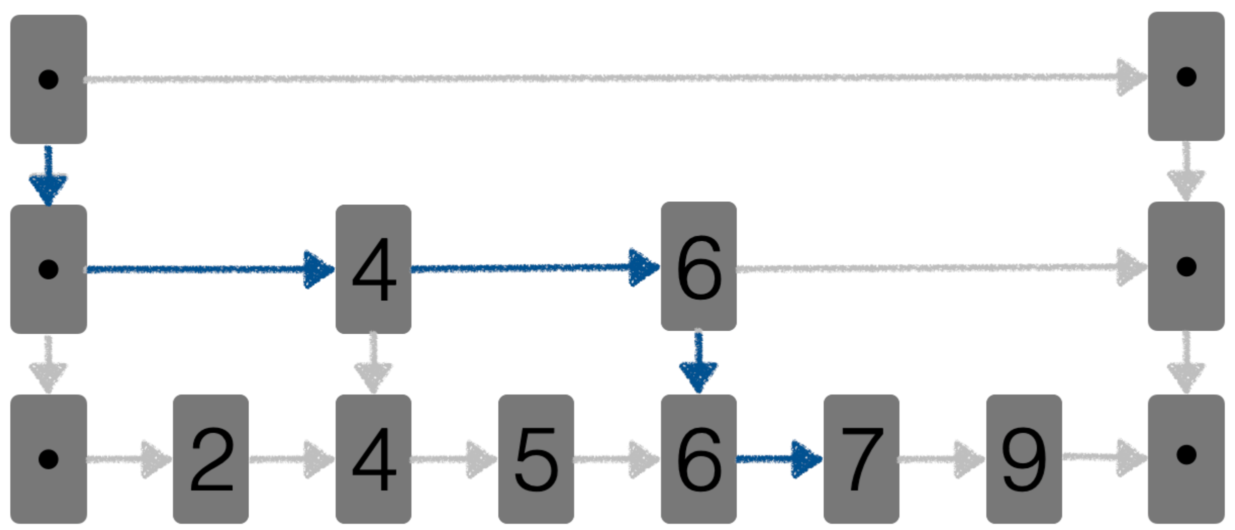


Time-Traveling LSM

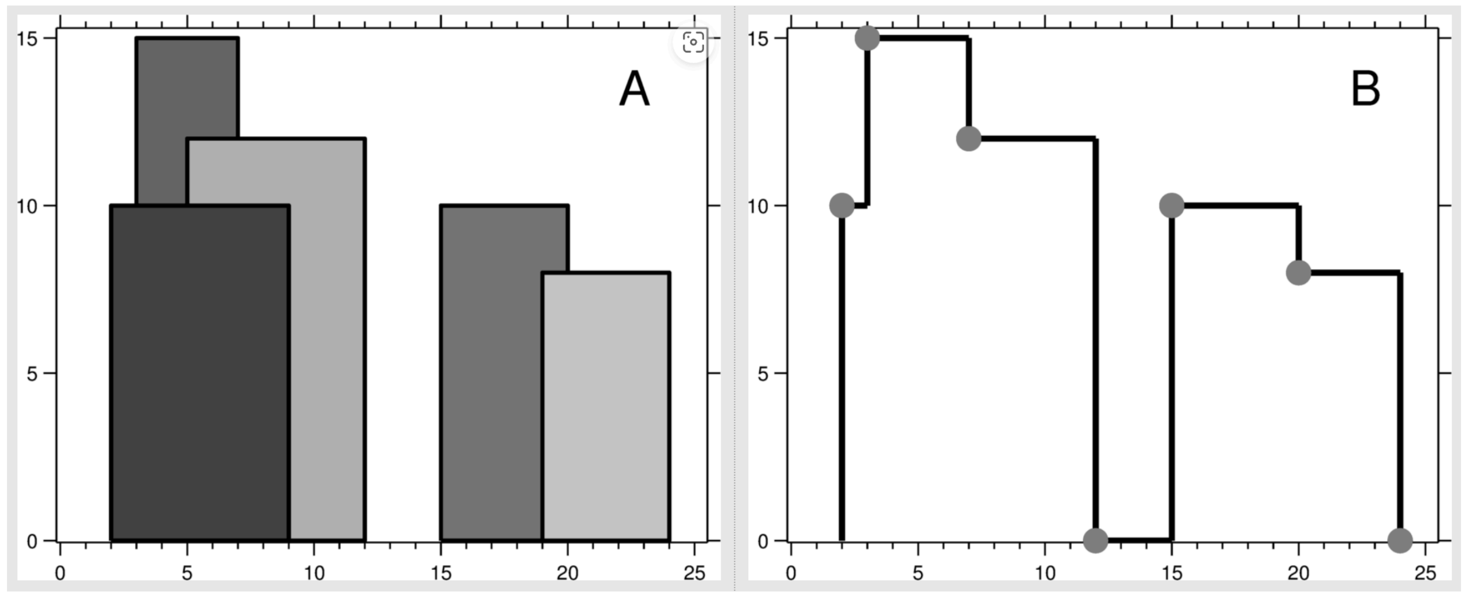
Brandeis UNIVERSITY

Reserach projects

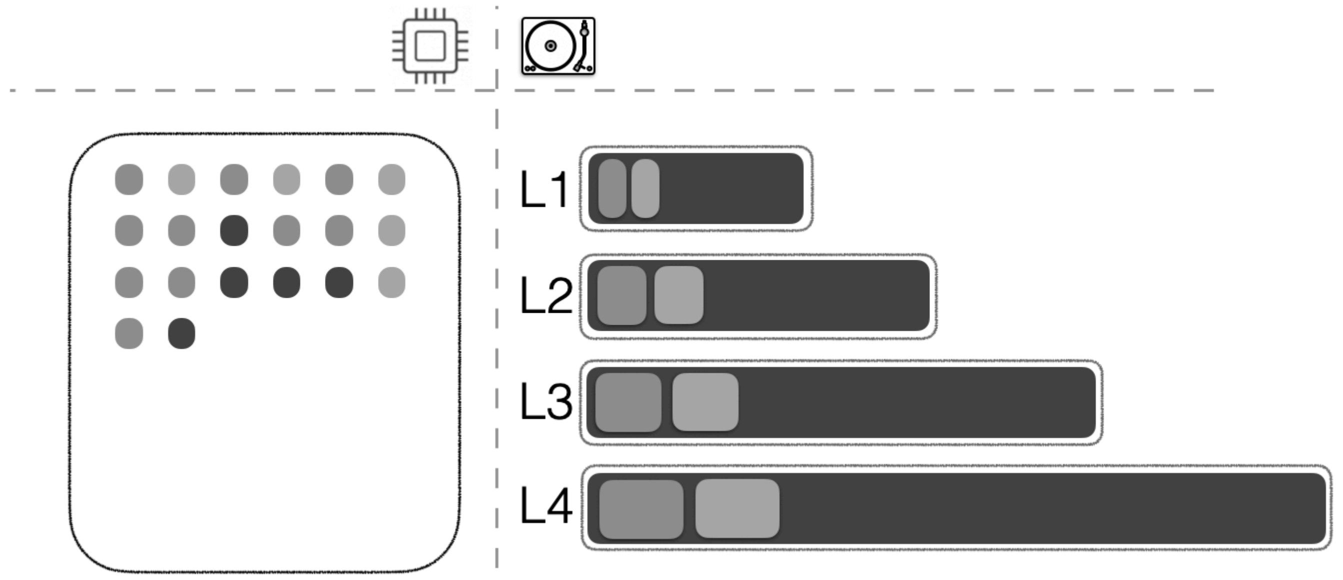
Solving problems on your way through!



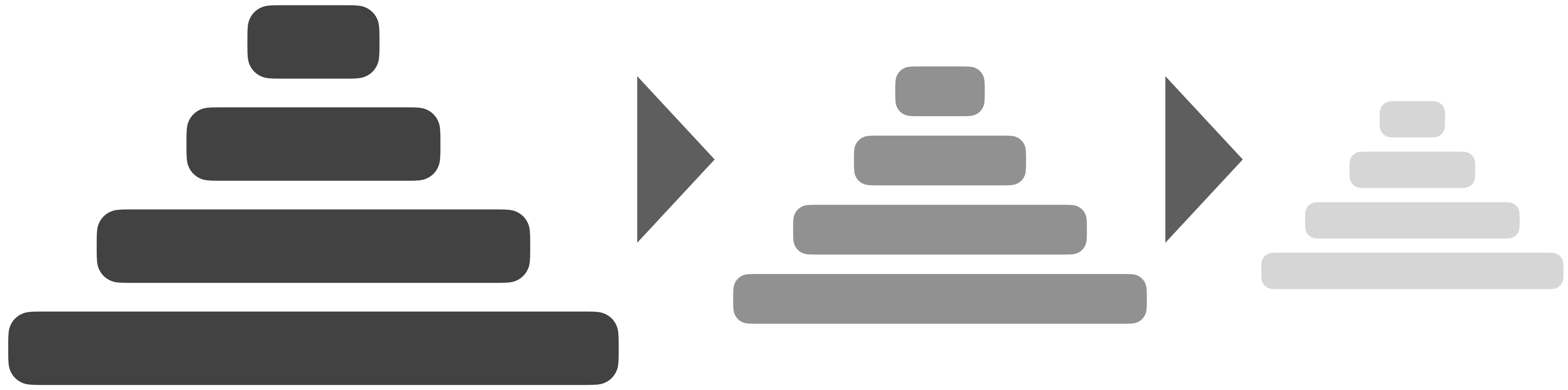
Self-designing LSM Buffer



Range Deletes in LSM



Optimal Caching in LSM



Time-Traveling LSM

Next time in COSI 167A

More on LSMs

LSM buffer design space

[P] "Anatomy of the LSM Memory Buffer: Insights & Implications", *DBTest*, 2024

TECHNICAL QUESTION 3 **How does the memory allocation strategy affect the query latency for a pre-allocated vector vs. a dynamically allocated one? Given a workload, how would do decide the prefix length for a hash-hybrid buffer implementation?**

[B] "Breaking Down Memory Walls: Adaptive Memory Management in LSM-based Storage Systems", *VLDB*, 2020

COSI 167A
Advanced Data Systems

Class 7
Class Projects

Prof. Subhadeep Sarkar

https://ssd-brandeis.github.io/COSI-167A/