COSI 167A

Advanced Data Systems

Class 6

Performance Analysis of LSM-Trees

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**).

Deadline is at **12:45 PM** for all **technical questions** and **reviews**!

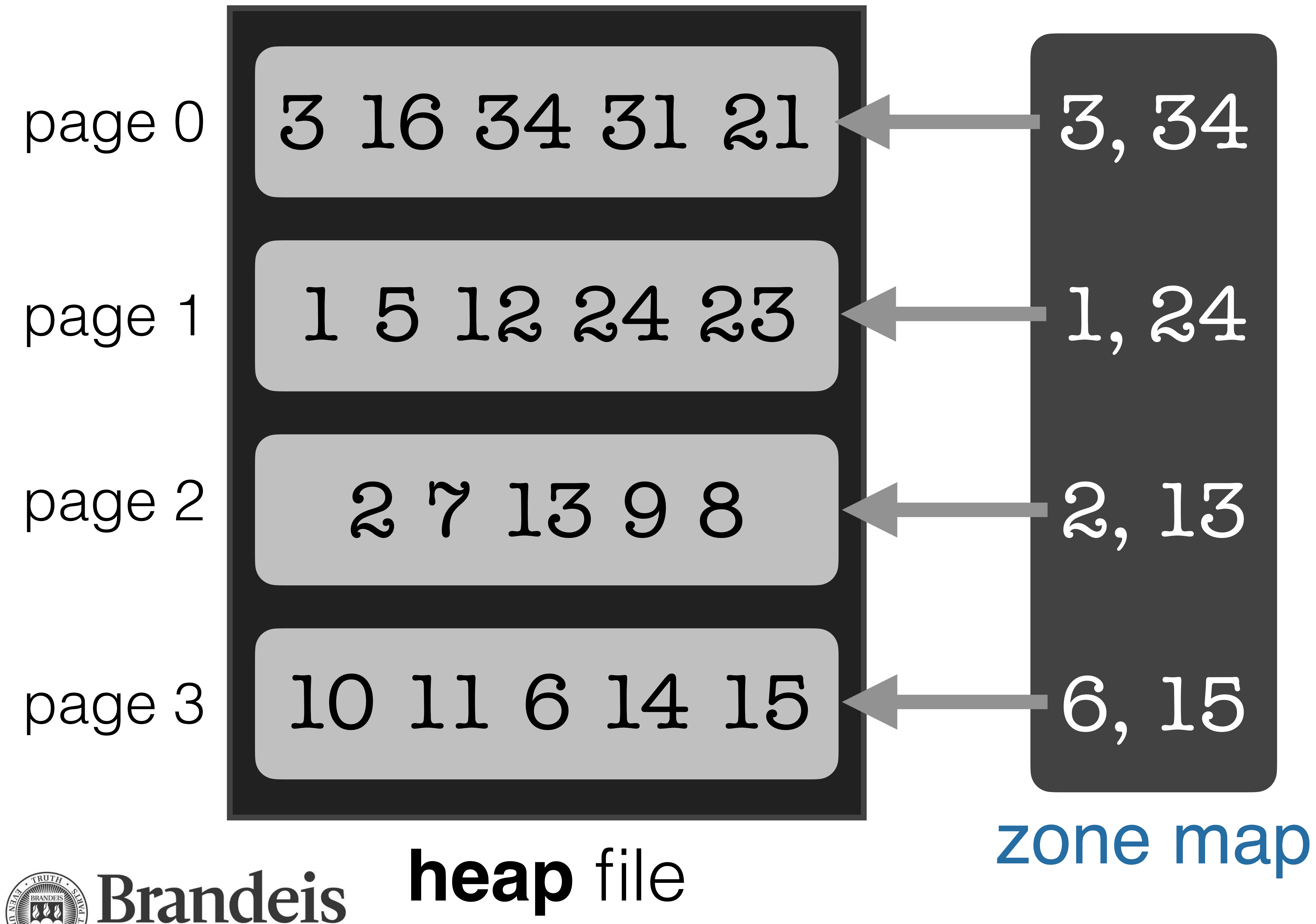
Project 1 is due in **3 days** (**Fri, Sep 20**).

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

Project 1

Testing the waters!

Implementing a Simple Zone Map



do not sort the data for **unsorted** workloads

Requirement

sort(): may want to sort the data **within a zone**

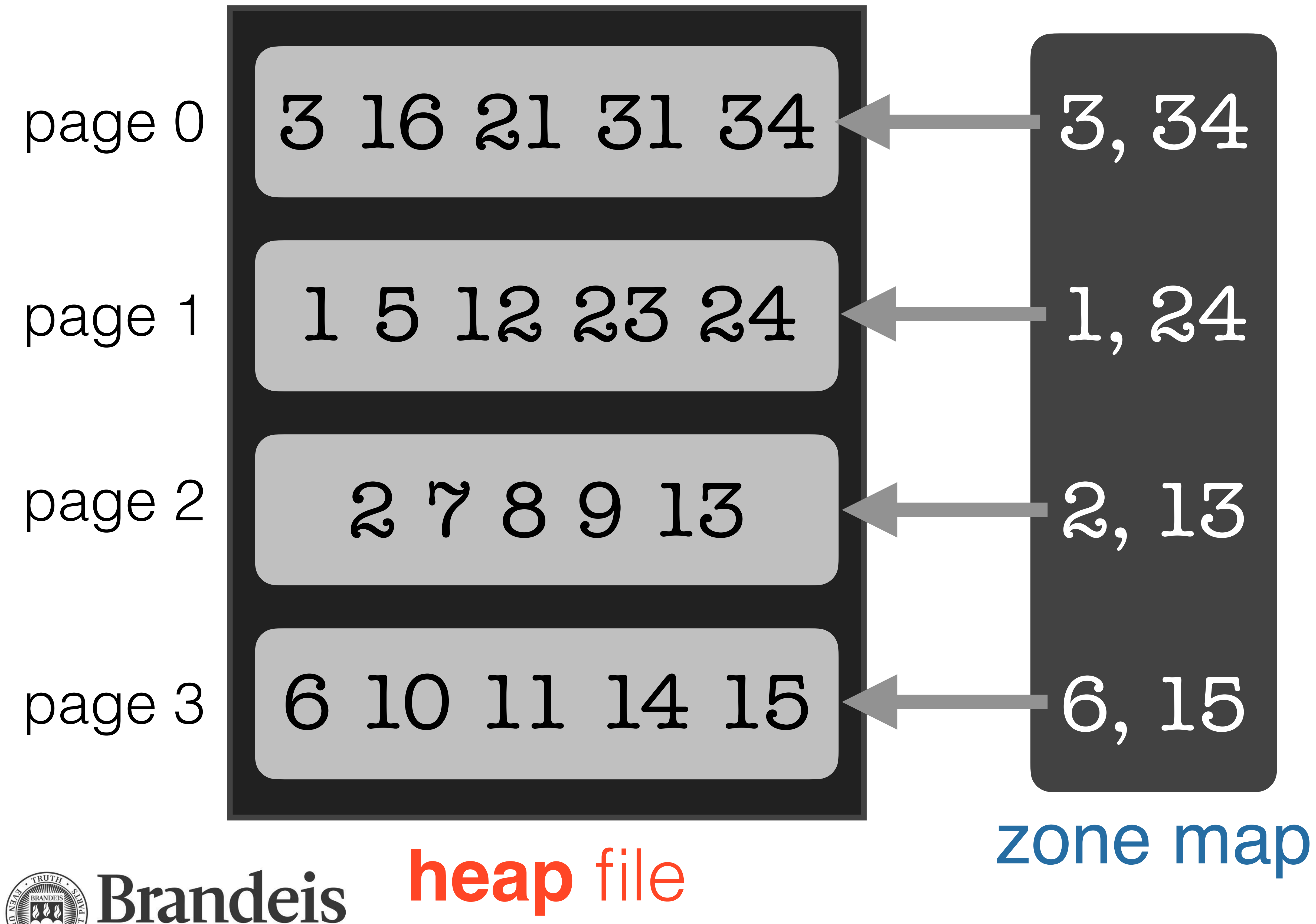
to facilitate binary search

Brandeis UNIVERSITY

Project 1

Testing the waters!

Implementing a Simple Zone Map



do not sort the data for **unsorted** workloads

Requirement

sort(): may want to sort the data **within a zone**

to facilitate binary search

Design optimization (optional)

Brandeis University

Today in COSI 167A

What's on the cards?

Queries in **LSM-trees**

Cost analysis

[P] "Monkey: Optimal Navigable Key-Value Store", *SIGMOD*, 2017

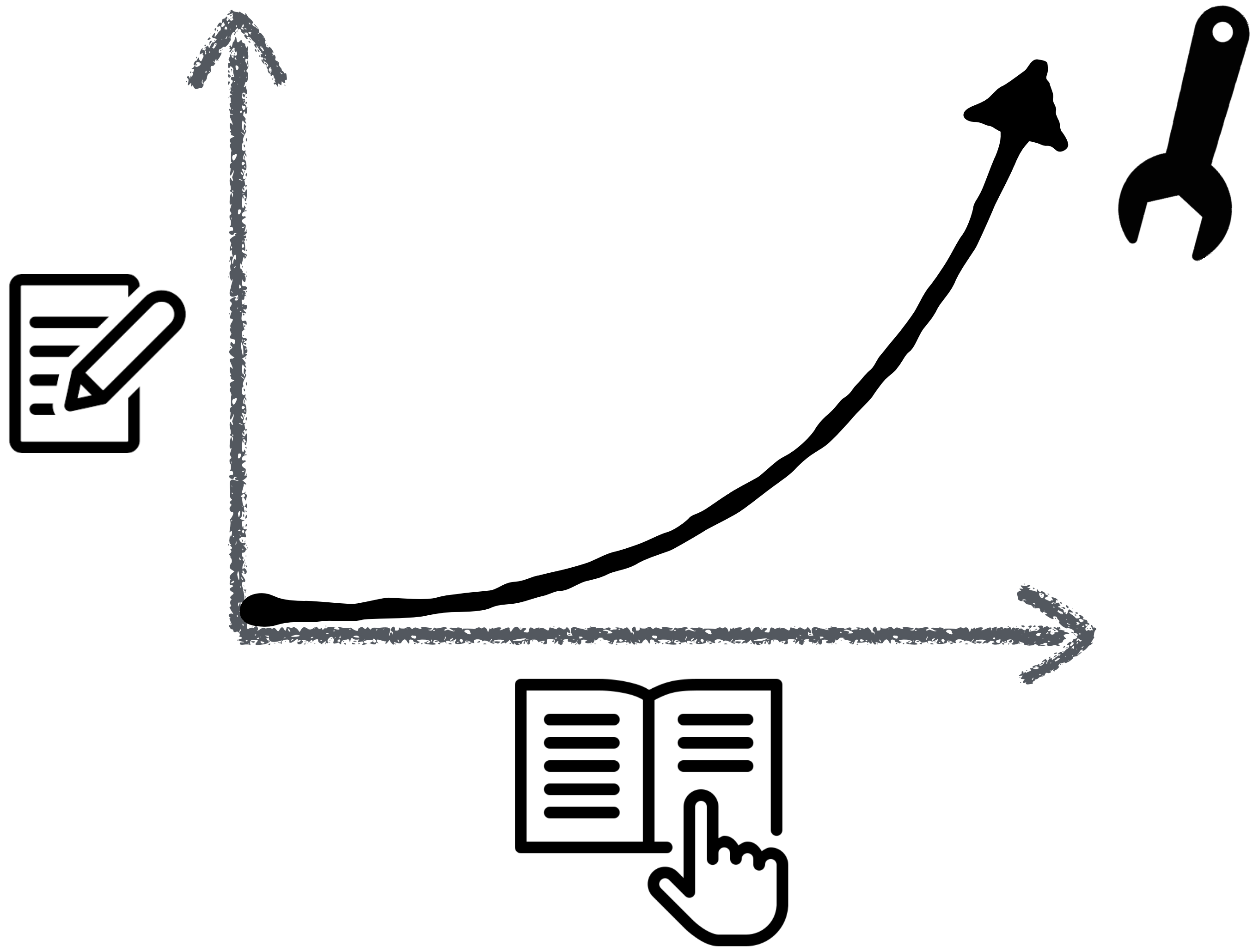
TECHNICAL QUESTION 2 **When does a tiered LSM-tree behave similarly to a leveled LSM-tree? What is the key contribution of the paper?**

[B] "The LSM Design Space and its Read Optimizations", *ICDE*, 2023

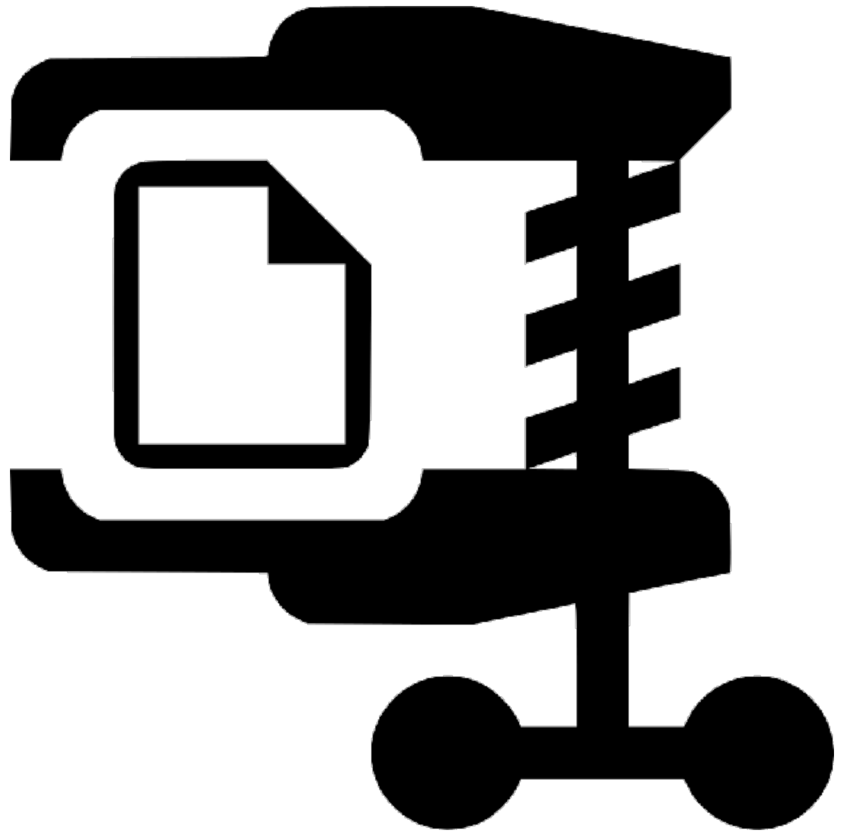
Why **LSM**?

What's the hype all about?

fast writes



tunable read-write performance



good space utilization

LSM Operating **Principles**

Buffering ingestion

Immutable files on storage

Out-of-place updates & deletes

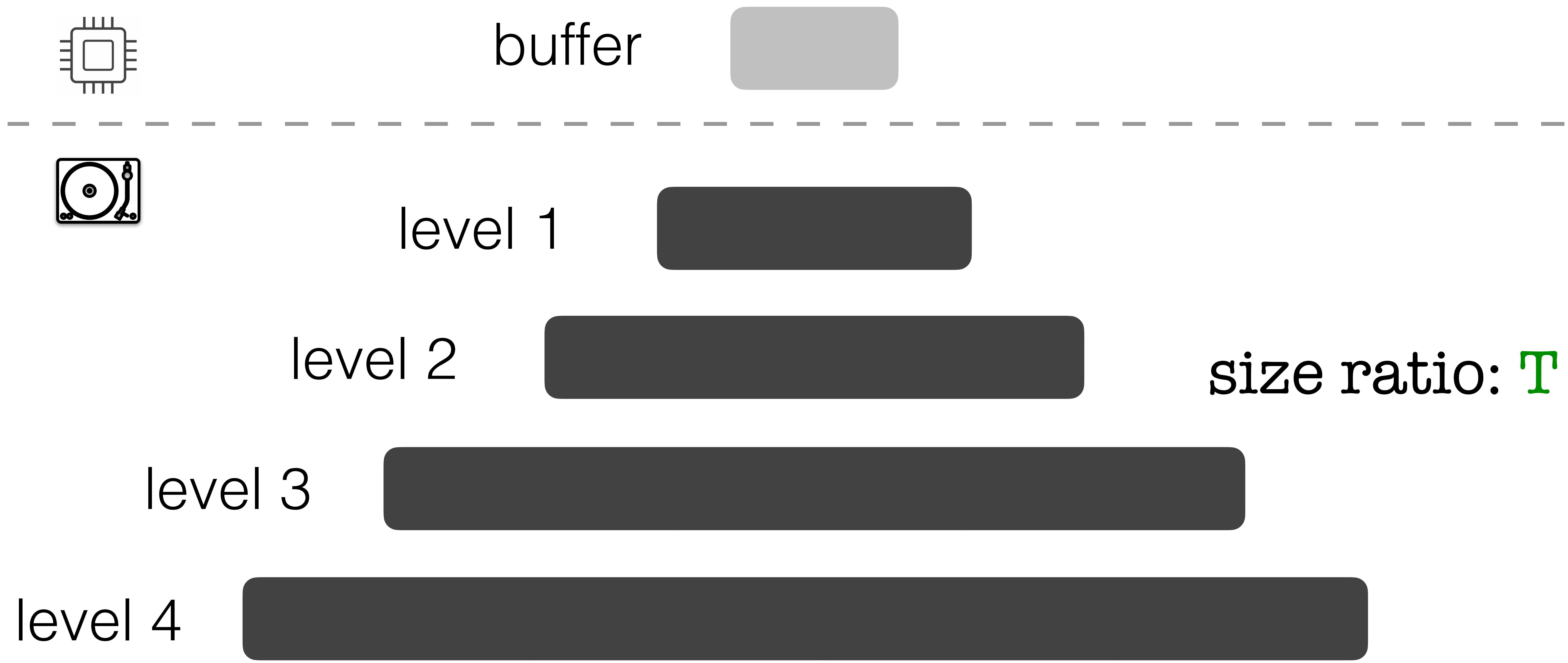
Periodic data layout reorganization

LSM **basics**

How do they look?

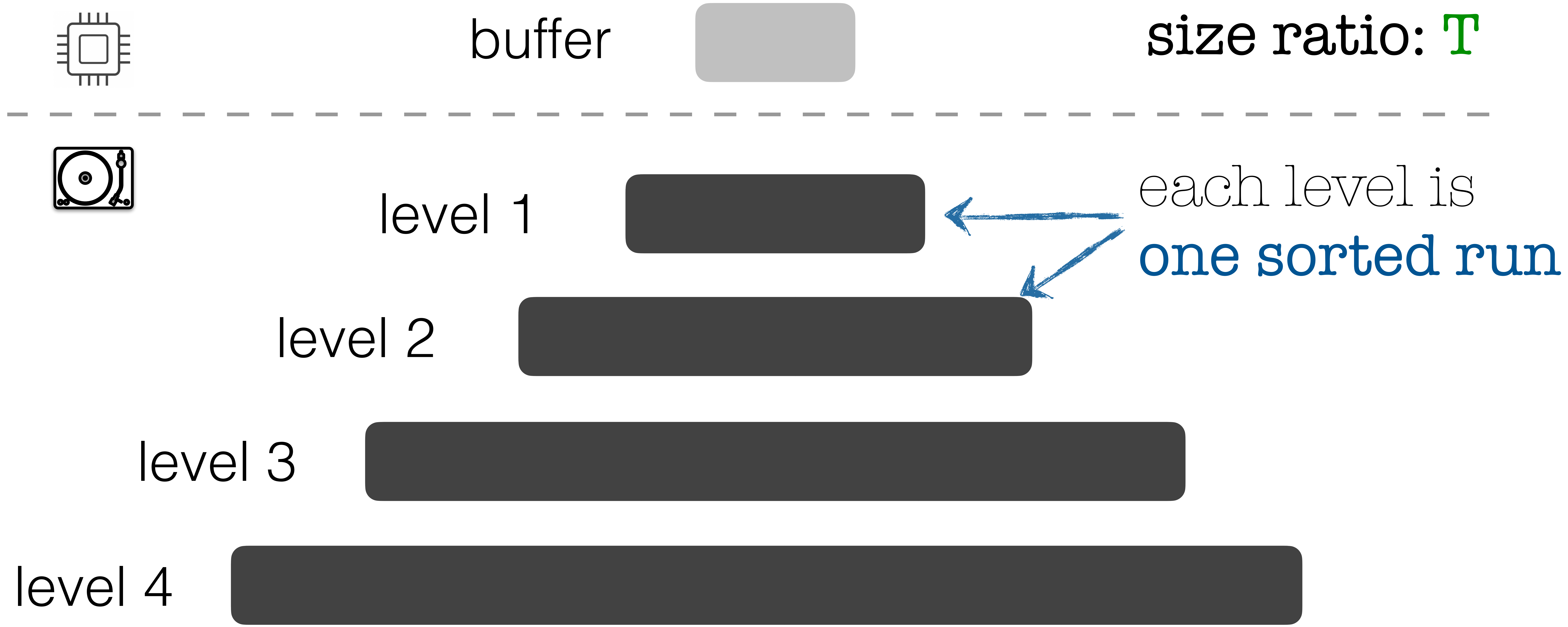
LSM **basics**

How do they look?



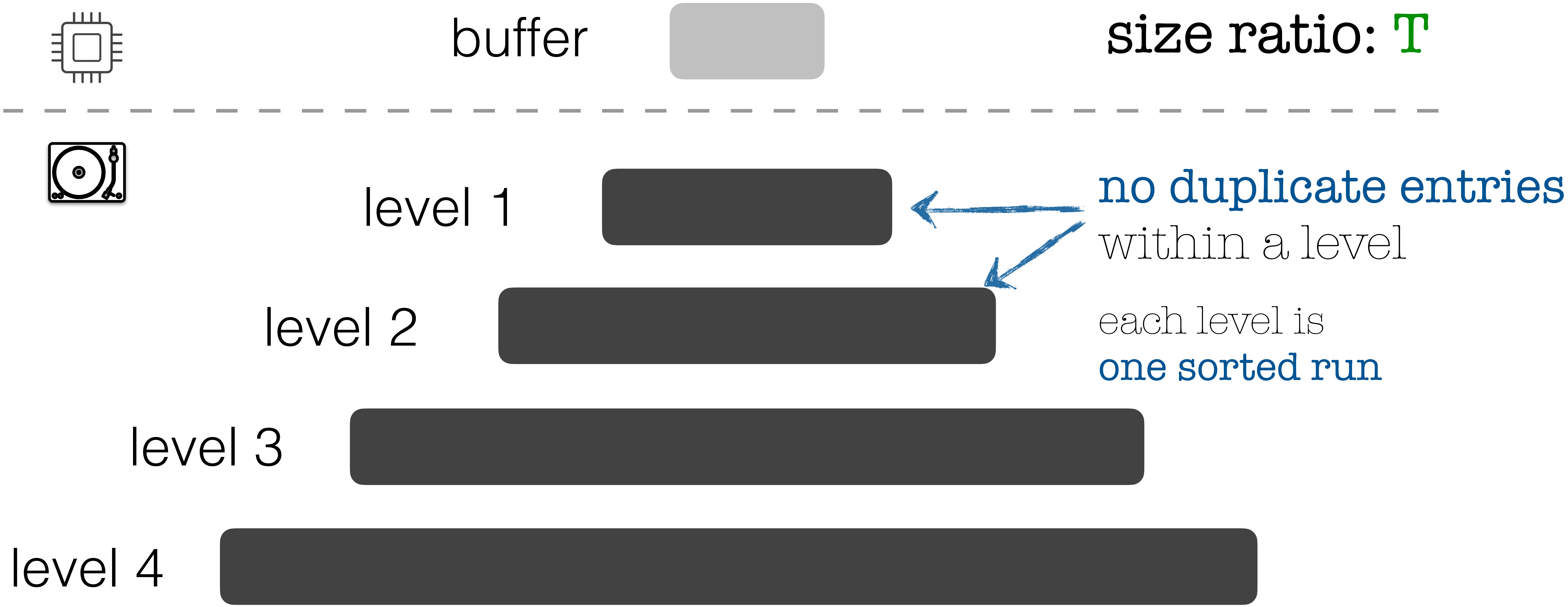
LSM **basics**

How do they look?

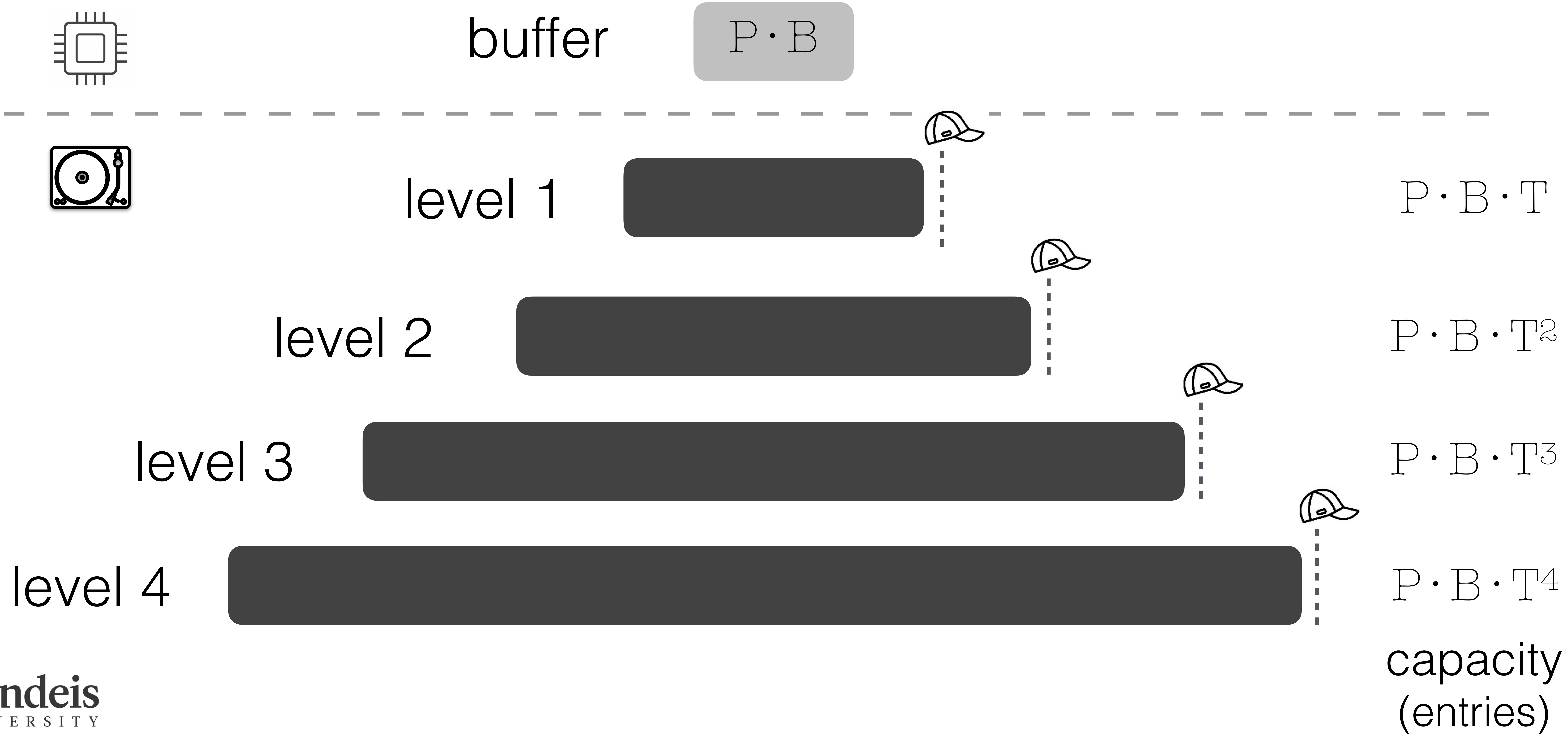


LSM **basics**

How do they look?



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

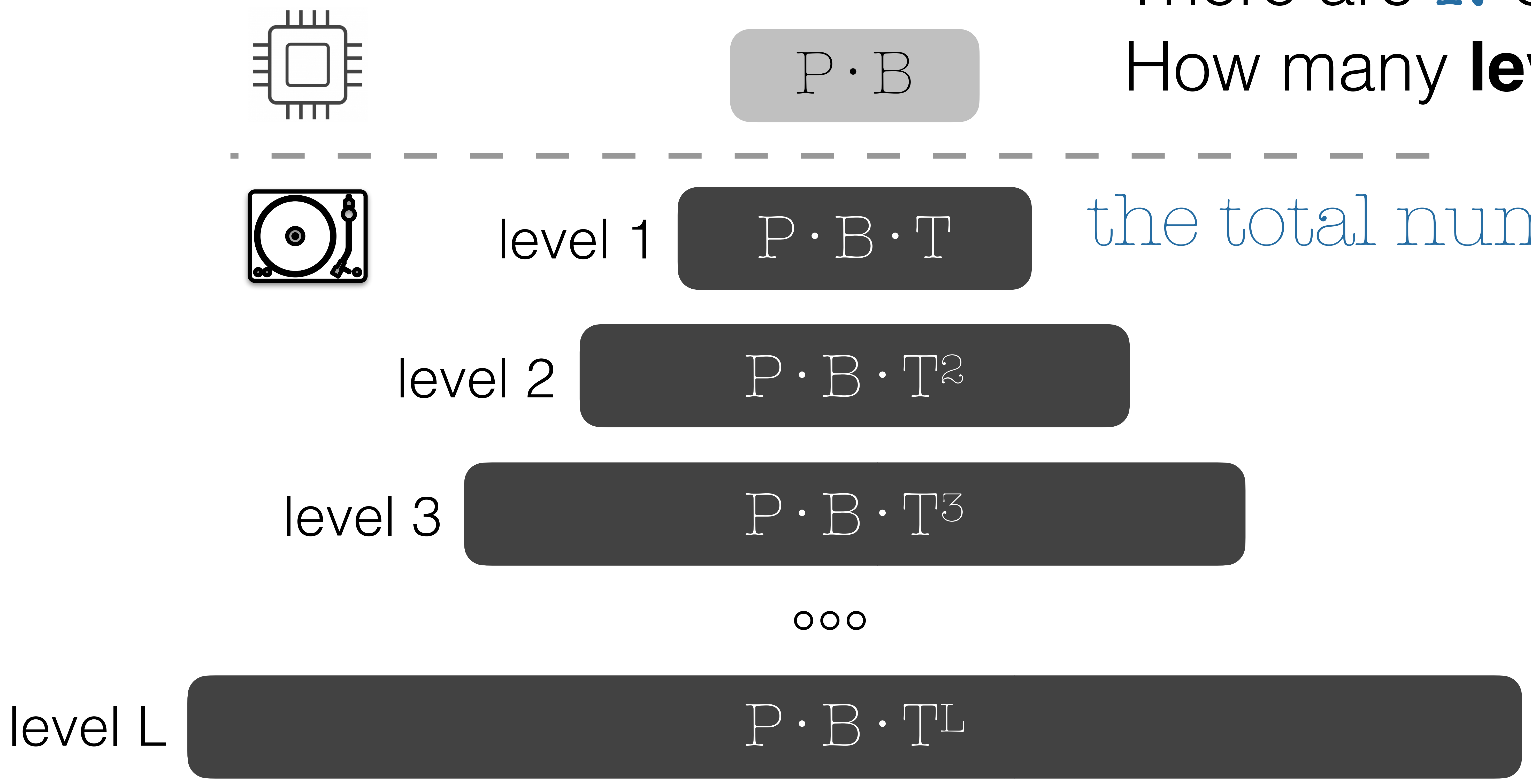


Brandeis UNIVERSITY

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

Thought Experiment 1

There are **N** entries in an LSM with size ratio **T**.
How many **levels** does the tree have?



the total number of entries in the tree is **N**

Brandeis UNIVERSITY

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

Thought Experiment 1

There are **N** entries in an LSM with size ratio **T**.
How many **levels** does the tree have?

$P \cdot B$

level 1 $P \cdot B \cdot T$

level 2 $P \cdot B \cdot T^2$

level 3 $P \cdot B \cdot T^3$

ooo

level L $P \cdot B \cdot T^L$

the total number of entries in the tree is **N**

$$\mathbf{N} = P \cdot B + P \cdot B \cdot T + P \cdot B \cdot T^2 + \cdots + P \cdot B \cdot T^L$$

$$\mathbf{L} = \log_T (N/(P \cdot B) \cdot ((T-1) / T))$$

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

Thought Experiment 1

There are **N** entries in an LSM with size ratio **T**.
How many **levels** does the tree have?

$P \cdot B$

level 1 $P \cdot B \cdot T$

level 2 $P \cdot B \cdot T^2$

level 3 $P \cdot B \cdot T^3$

∘∘∘

level L $P \cdot B \cdot T^L$

the total number of entries in the tree is **N**

$$\mathbf{N} = P \cdot B + P \cdot B \cdot T + P \cdot B \cdot T^2 + \cdots + P \cdot B \cdot T^L$$

$$\mathbf{L} = \log_T \left(N/(P \cdot B) \cdot ((T-1)/T)\right)$$

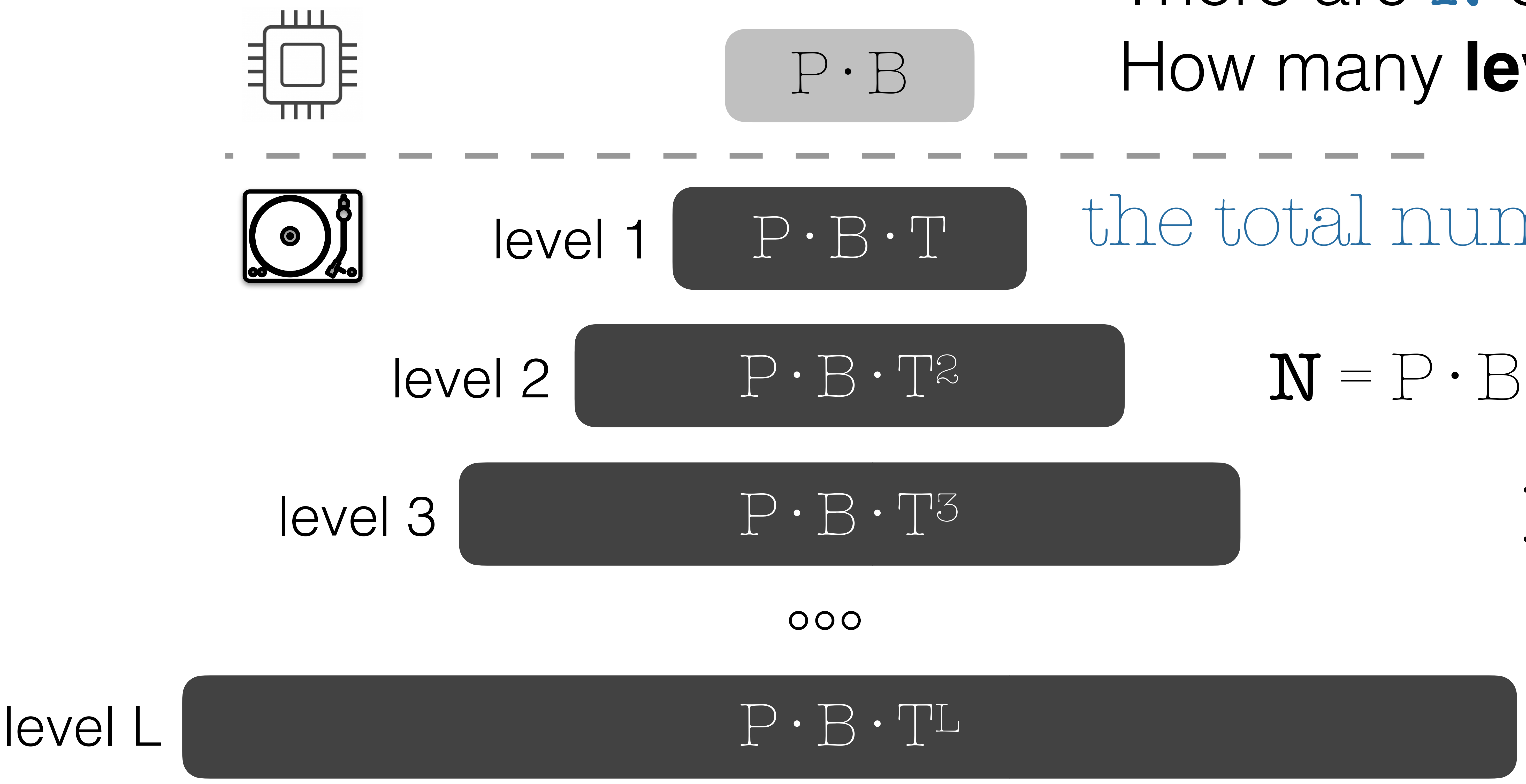
behaves like a constant
for 2 < T < 20

Brandeis UNIVERSITY

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

Thought Experiment 1

There are **N** entries in an LSM with size ratio **T**.
How many **levels** does the tree have?



the total number of entries in the tree is **N**

$$N = P \cdot B + P \cdot B \cdot T + P \cdot B \cdot T^2 + \cdots + P \cdot B \cdot T^L$$

$$L = \log_T (N/(P \cdot B) \cdot ((T-1) / T))$$

behaves like a constant
for $2 < T < 20$

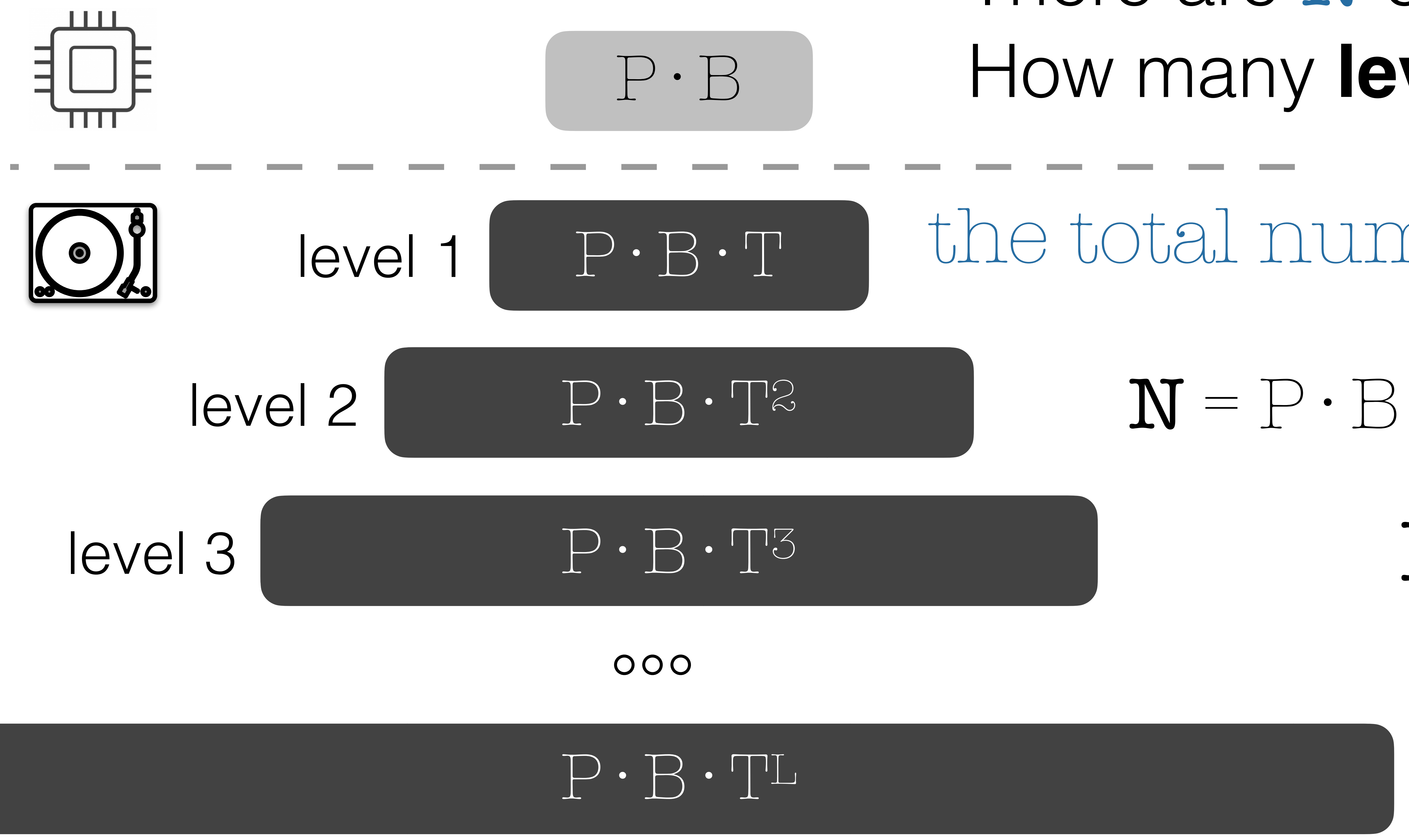
$$L = \log_T (N/(P \cdot B))$$

Brandeis UNIVERSITY

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

Thought Experiment 1

There are **N** entries in an LSM with size ratio **T**.
How many **levels** does the tree have?



the total number of entries in the tree is **N**

$$N = P \cdot B + P \cdot B \cdot T + P \cdot B \cdot T^2 + \cdots + P \cdot B \cdot T^L$$

$$L = \log_T (N/(P \cdot B) \cdot ((T-1)/T))$$

behaves like a constant
for $2 < T < 20$

$$L = \log_T (N/(P \cdot B))$$

$$L = O(\log_T (N))$$

Cost analysis

Counting all I/Os

Cost analysis

Counting all I/Os

data structure	ingestion cost	point lookup cost	range lookup cost
Leveled LSM-tree			
Tiered LSM-tree			
B^+-tree			
Sorted array			
Log			

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**

Ingestion cost

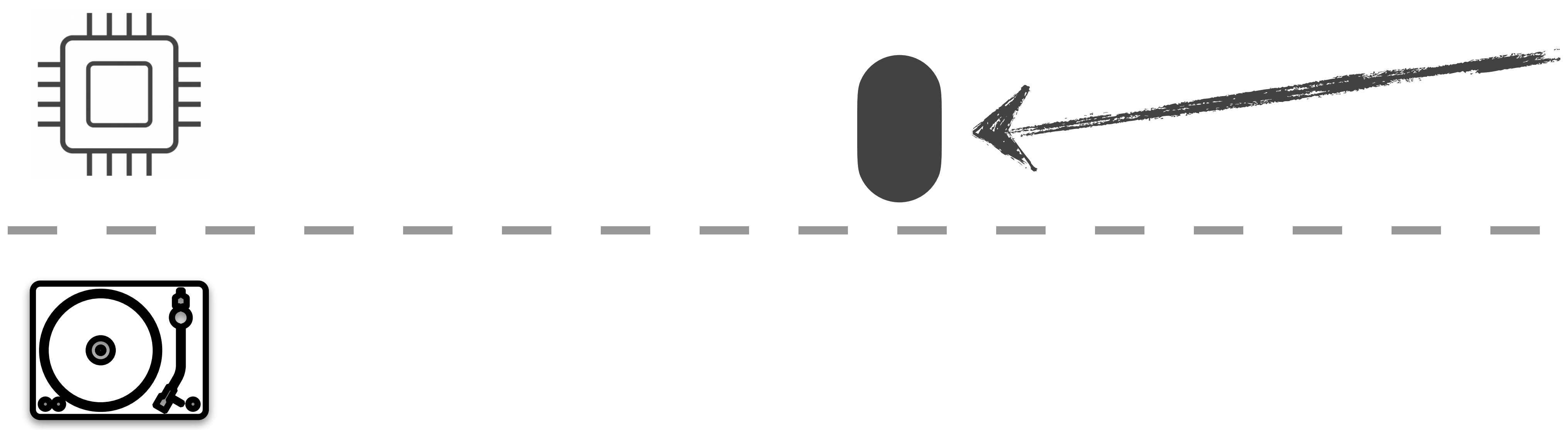
Inserts and updates

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

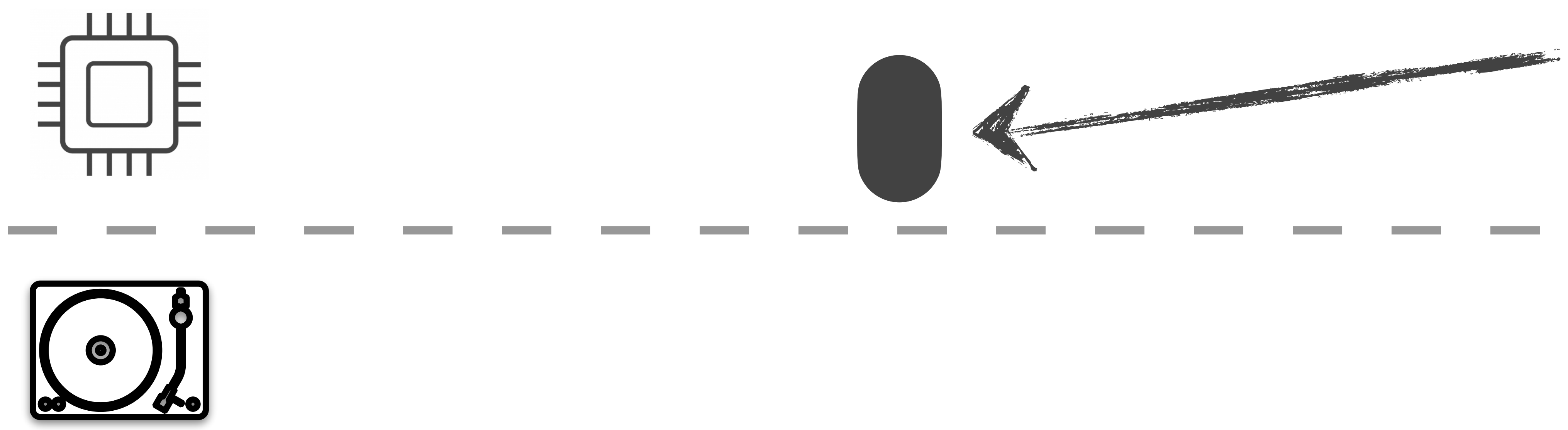
total entries in buffer = **P· B**

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

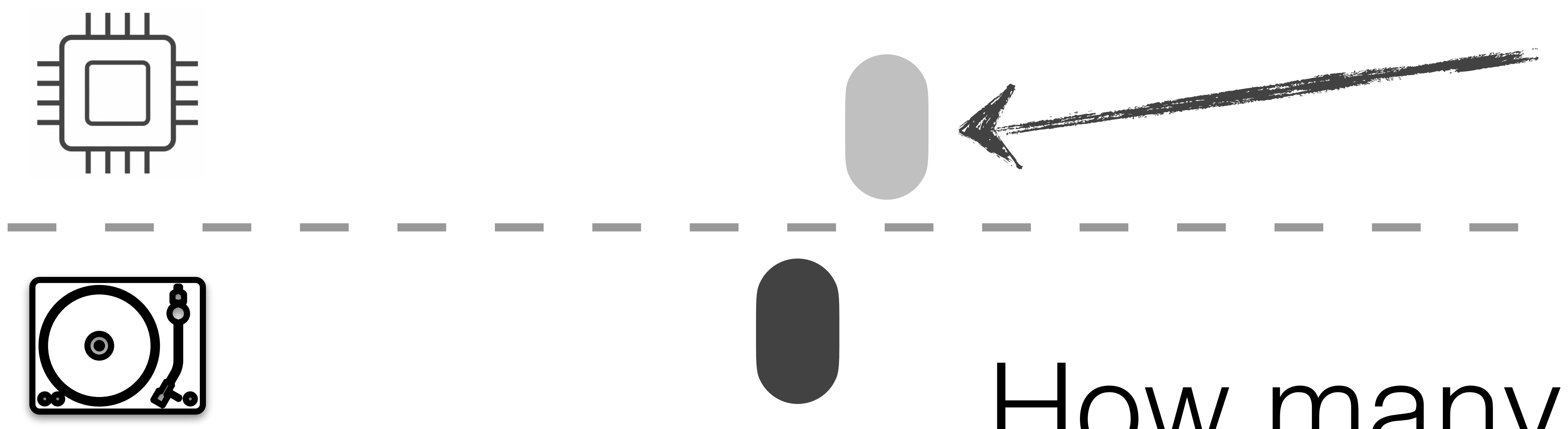
total entries in buffer = **P· B**

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P · B**

How many I/Os to flush the buffer?

P I/Os to flush the **P pages** in buffer

How many entries flushed per I/O?

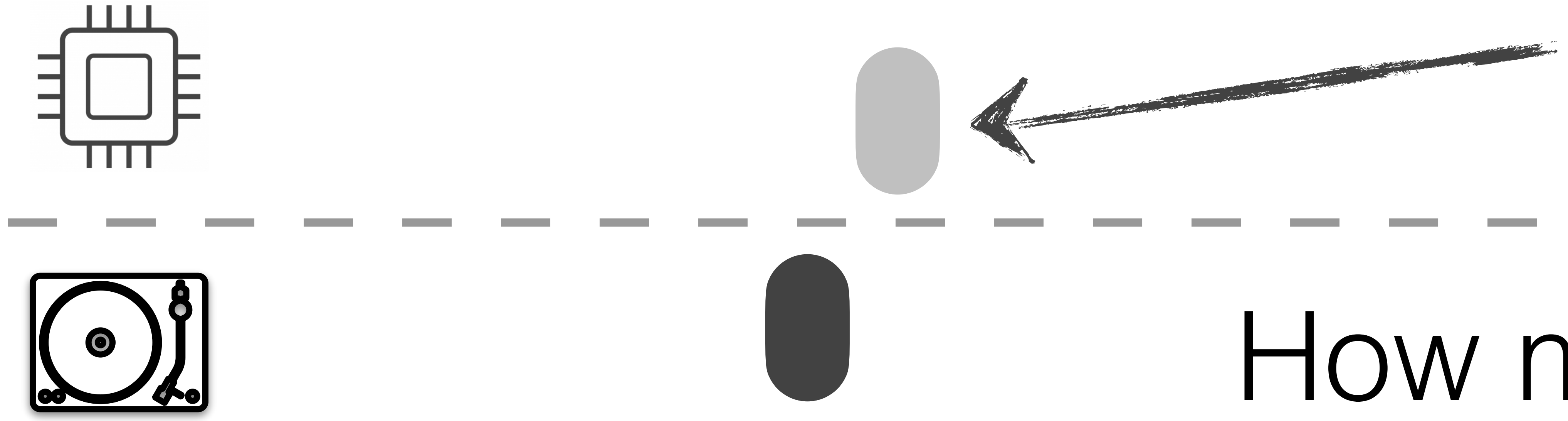
B entries per I/O

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P · B**

How many entries flushed per I/O?

B entries per I/O

How many I/Os to flush one entry?

1/B I/Os to write each entry

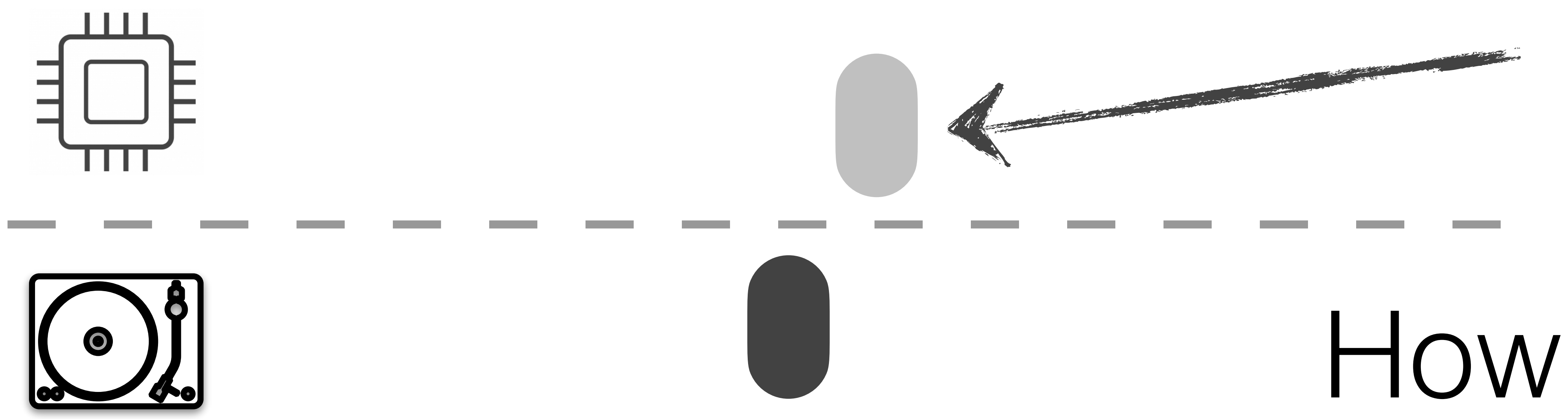
ingestion cost in leveled LSM-trees?

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P** · **B**

How many I/Os to flush one entry?

1/B I/Os to write each entry

Ingestion cost

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

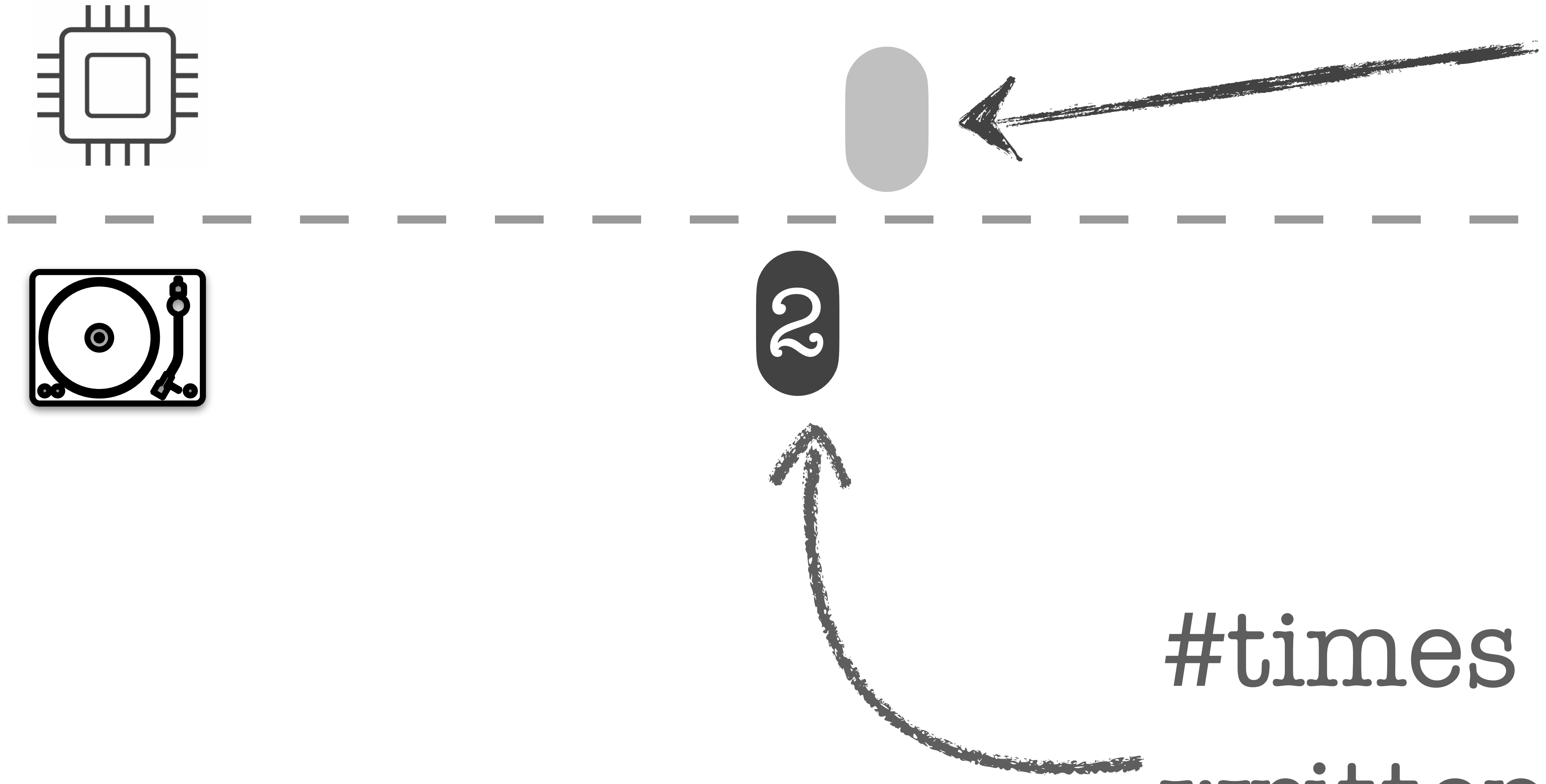
note: if an entry is written **multiple times**, **count all I/Os**

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P** · **B**

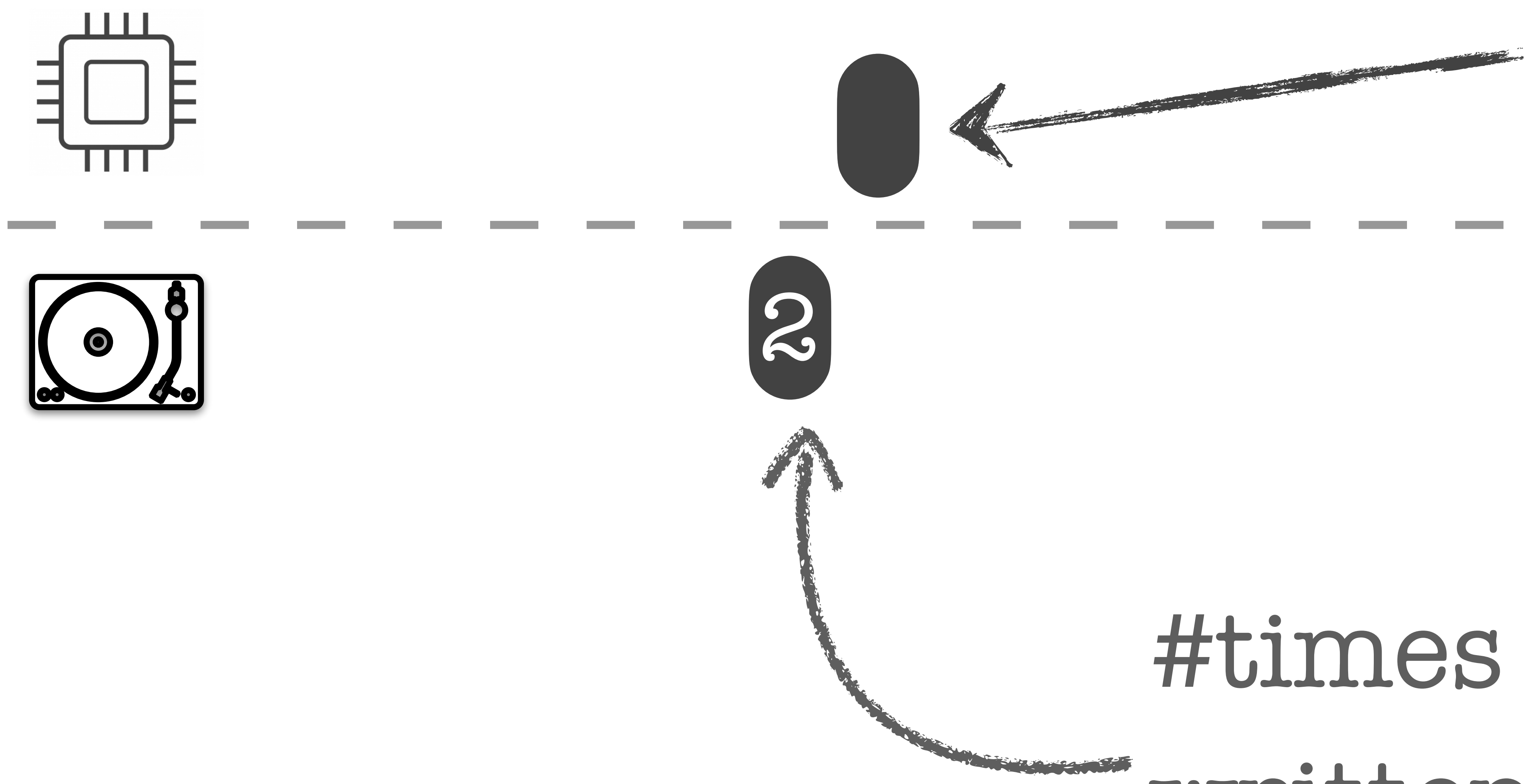
#times 2 has been written to this level = 1

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = $\mathbf{P} \cdot \mathbf{B}$

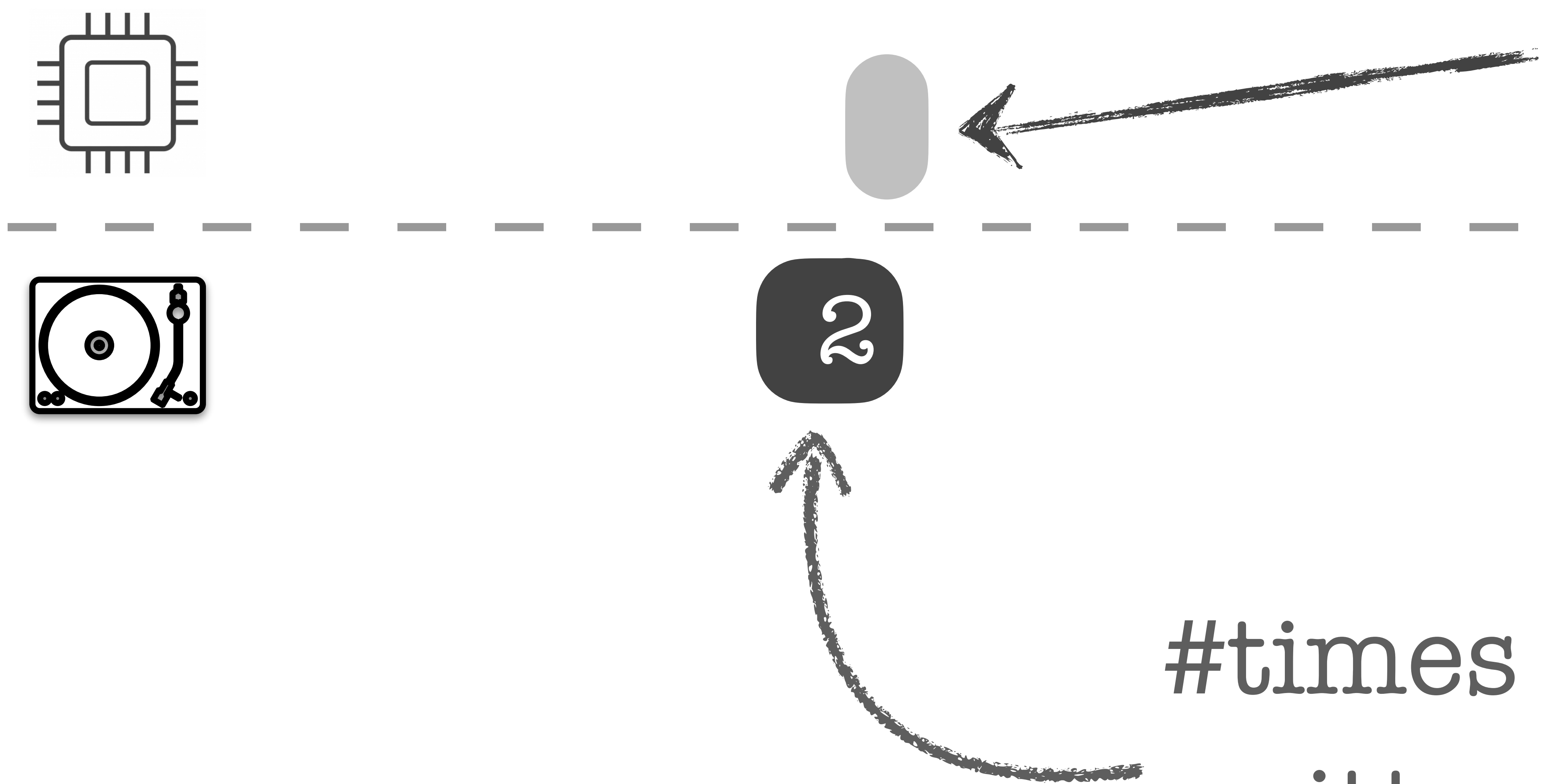
#times 2 has been written to this level = 1

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P** · **B**

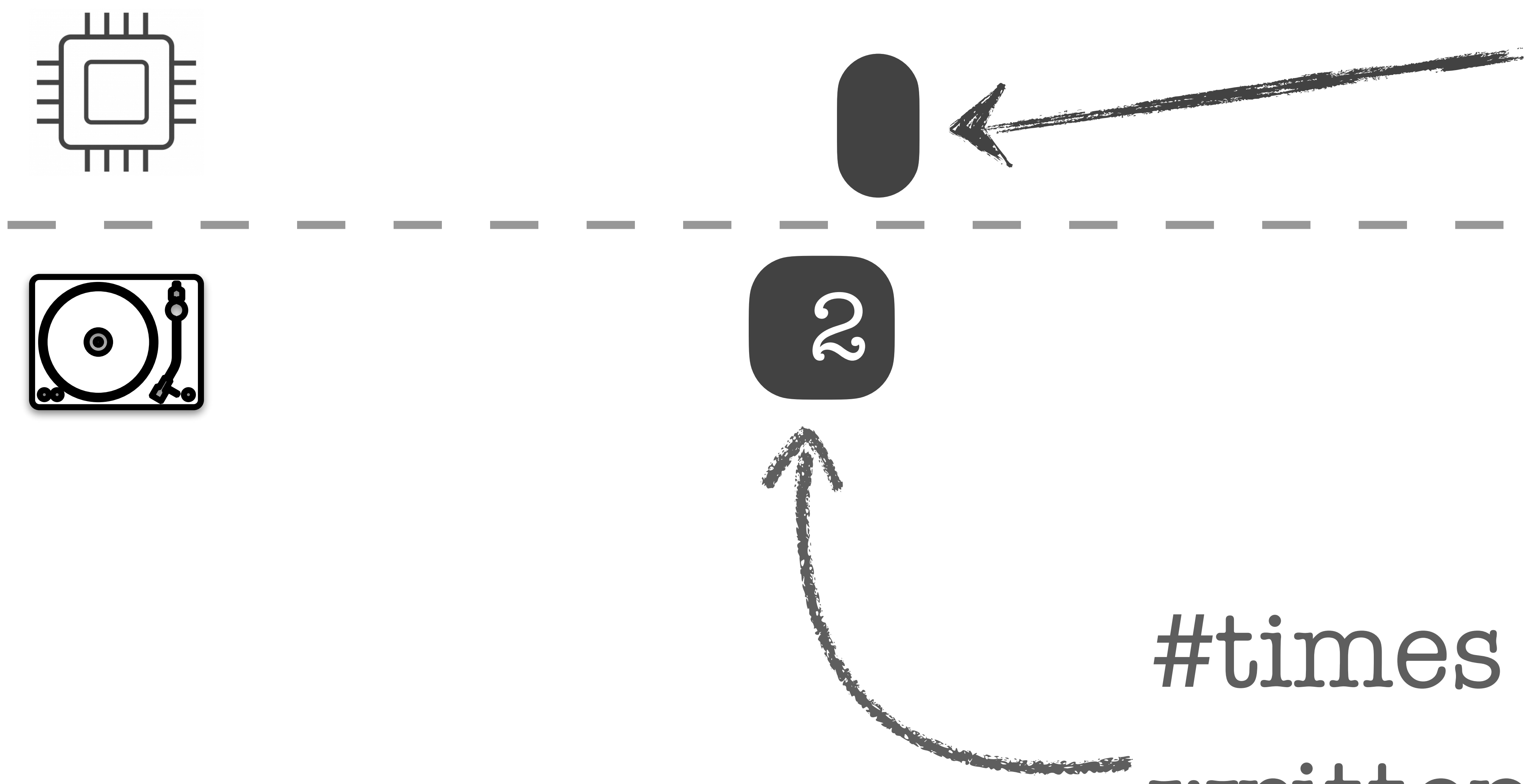
#times 2 has been
written to this level = 1

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P**· **B**

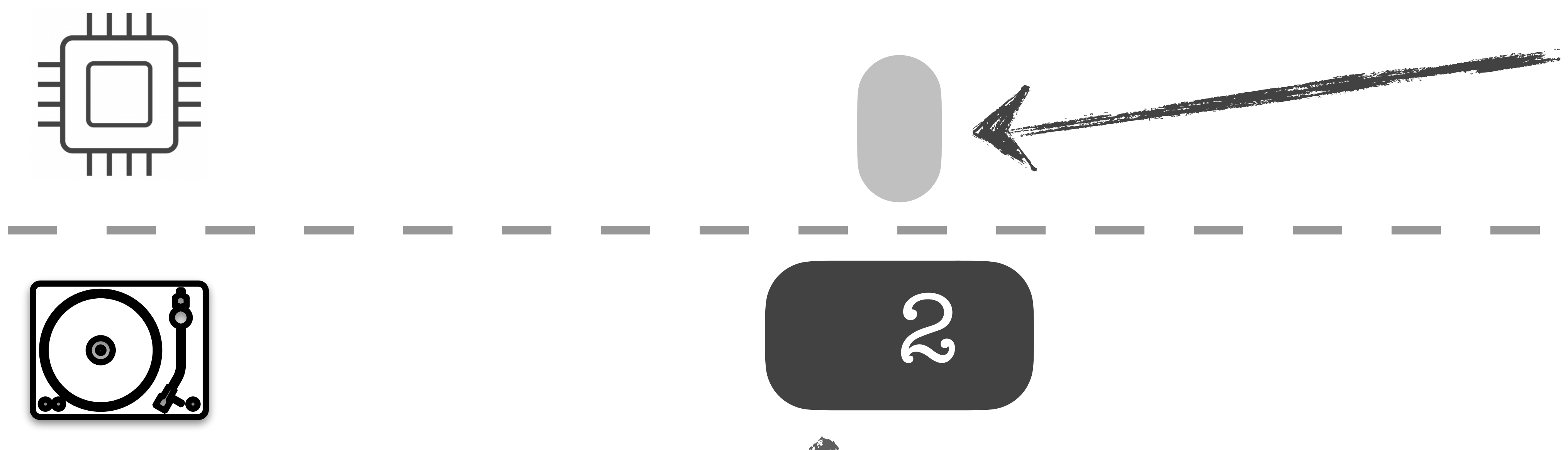
#times 2 has been written to this level = 2

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P**· **B**

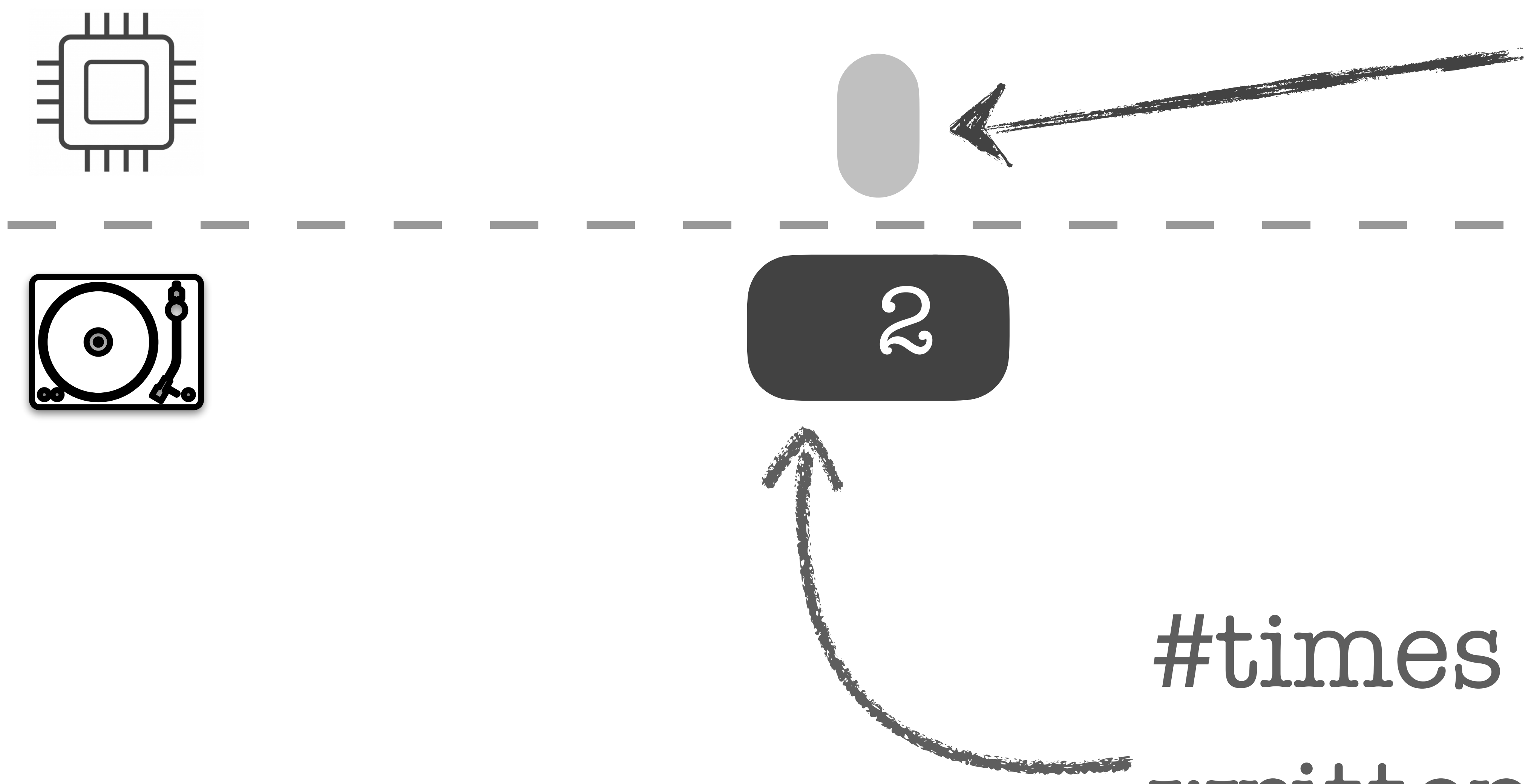
#times 2 has been written to this level = 2

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

Ingestion cost

Inserts and updates

leveled LSM-tree



P pages each with **B** entries

total entries in buffer = **P· B**

#times 2 has been written to this level = 3

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

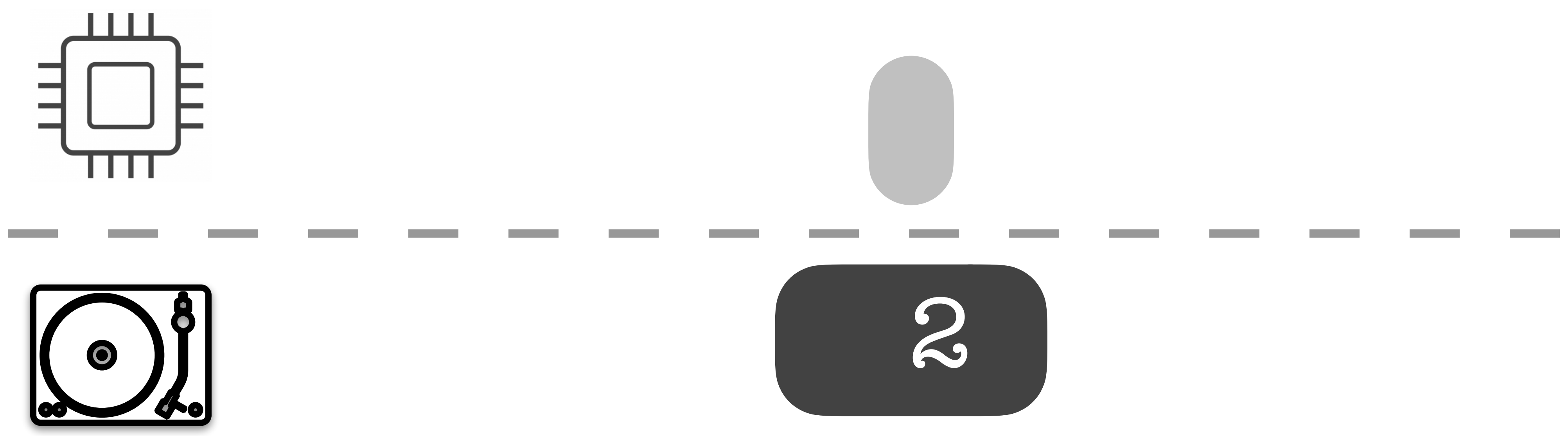
Brandeis UNIVERSITY

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

Ingestion cost

Inserts and updates

leveled LSM-tree



Thought Experiment 2

What is the ingestion cost in leveled LSM-trees?

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 = $\mathbf{L \cdot T}$

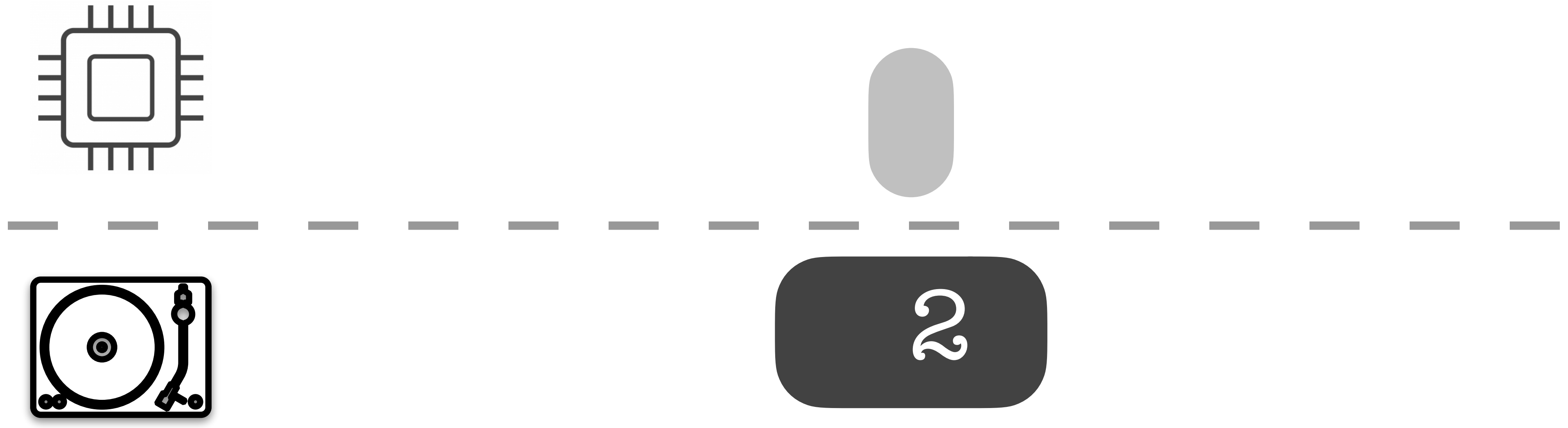
B entries are written to disk per I/O

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

Ingestion cost

Inserts and updates

leveled LSM-tree



Thought Experiment 2

What is the ingestion cost in leveled LSM-trees?

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

Average number of I/Os to write a single entry = $\mathbf{L \cdot T / B}$

Brandeis UNIVERSITY

Cost analysis

Counting all I/Os

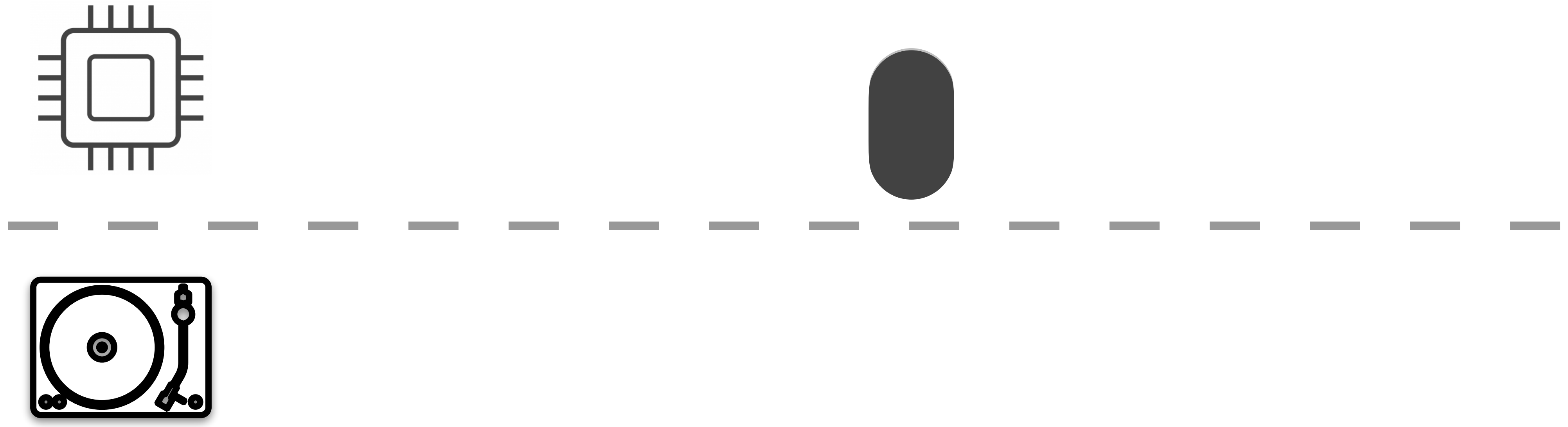
data structure	ingestion cost	point lookup cost	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$		
Tiered LSM-tree			
B^+-tree			
Sorted array			
Log			

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

Ingestion cost

Inserts and updates

tiered LSM-tree

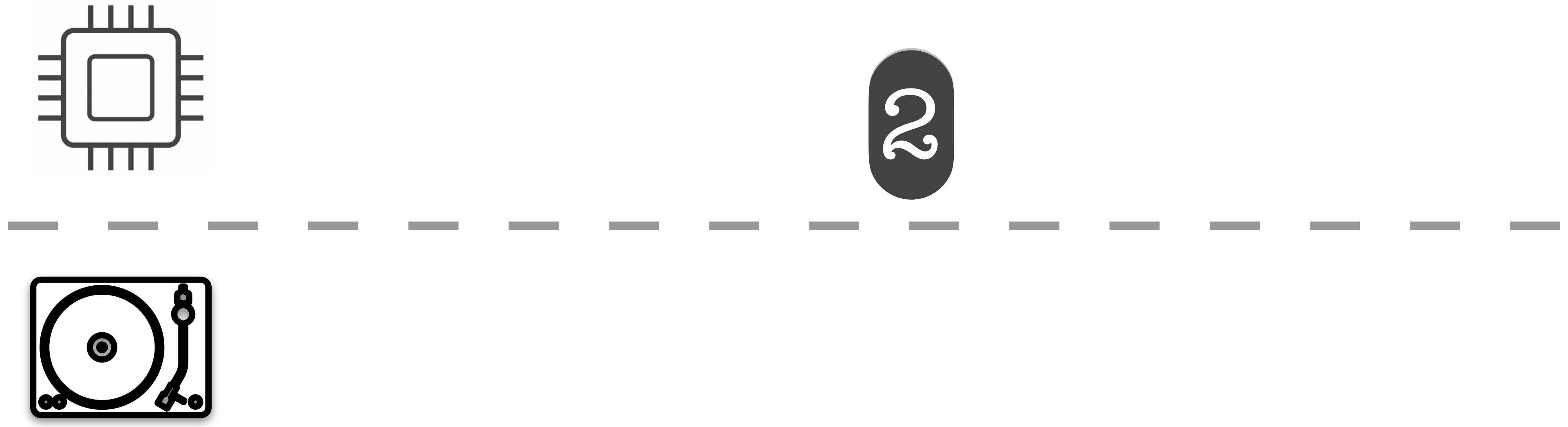


P: pages in buffer
B: entries/page
L: #levels
T: size ratio

Ingestion cost

Inserts and updates

tiered LSM-tree



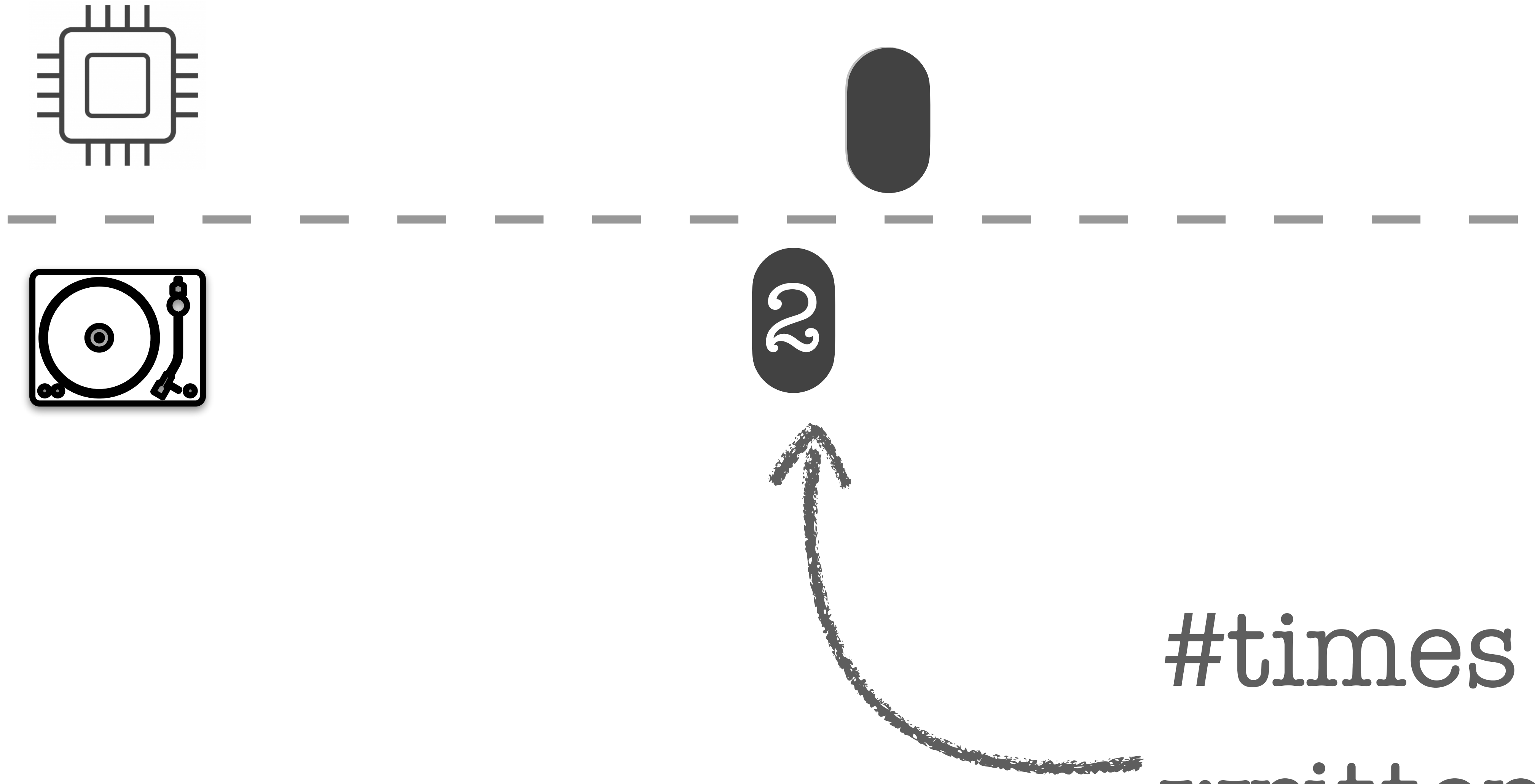
Brandeis UNIVERSITY

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

Ingestion cost

Inserts and updates

tiered LSM-tree



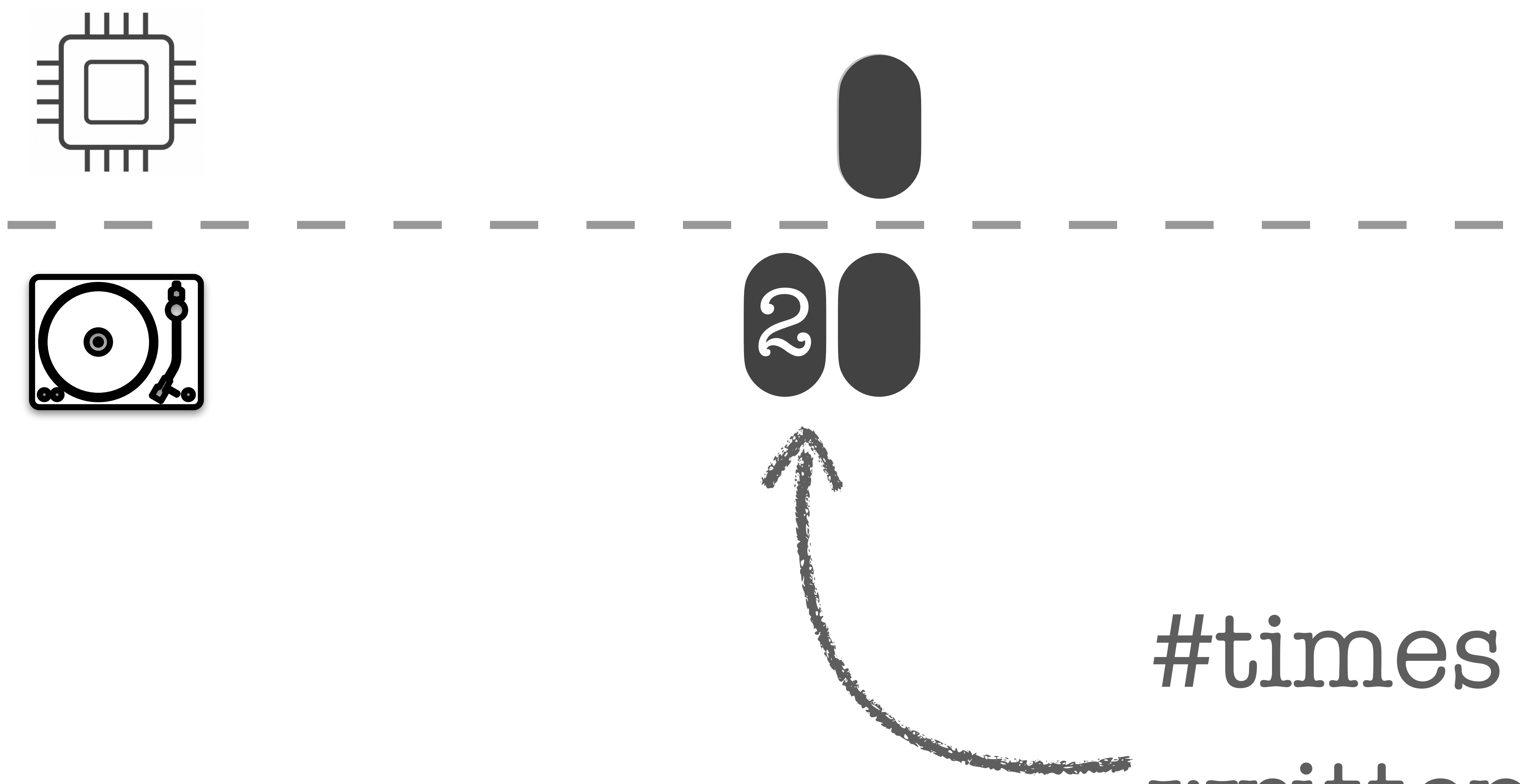
#times 2 has been written to this level = 1

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

Ingestion cost

Inserts and updates

tiered LSM-tree



#times 2 has been written to this level = 1

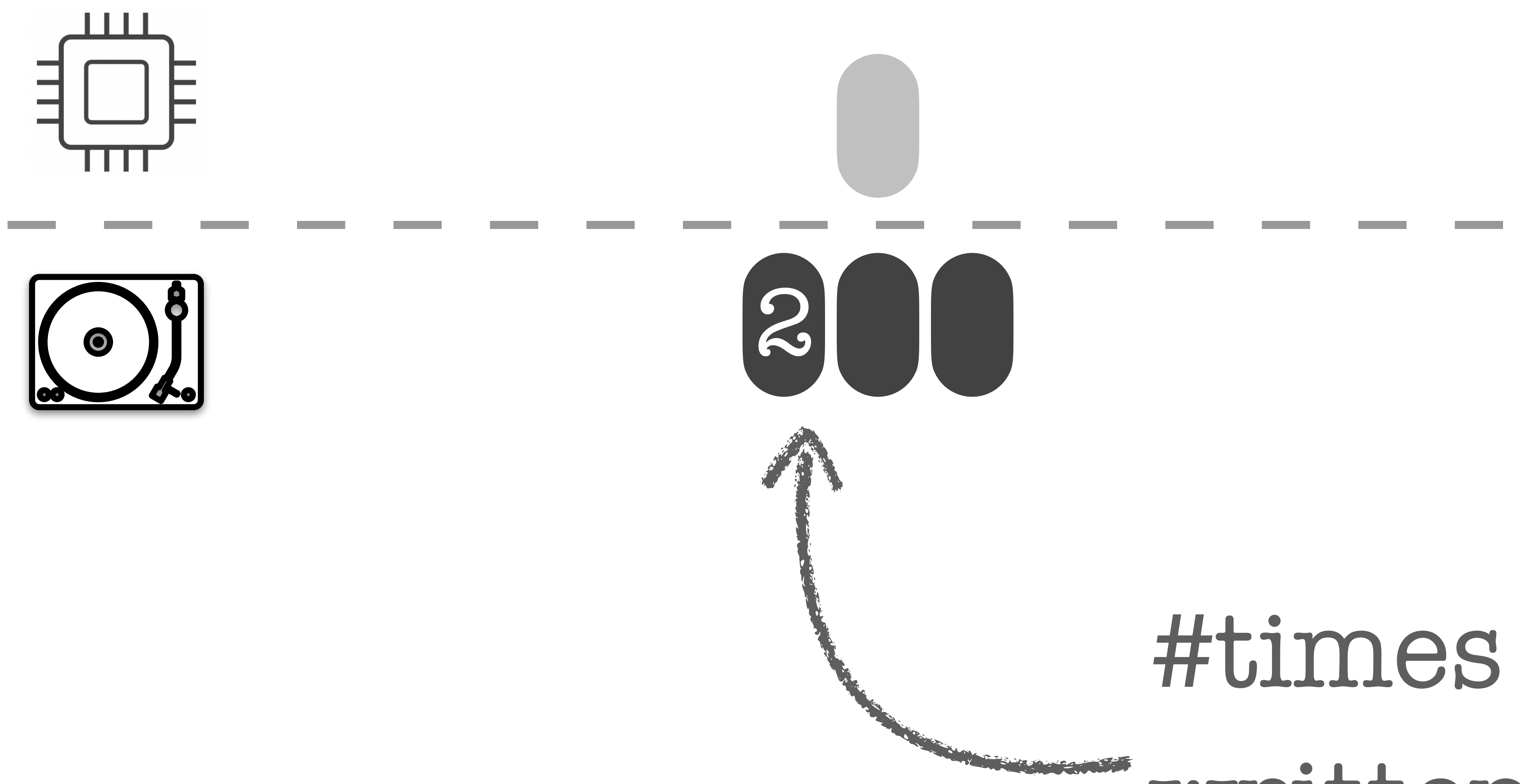
Brandeis UNIVERSITY

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

Ingestion cost

Inserts and updates

tiered LSM-tree



#times 2 has been written to this level = 1

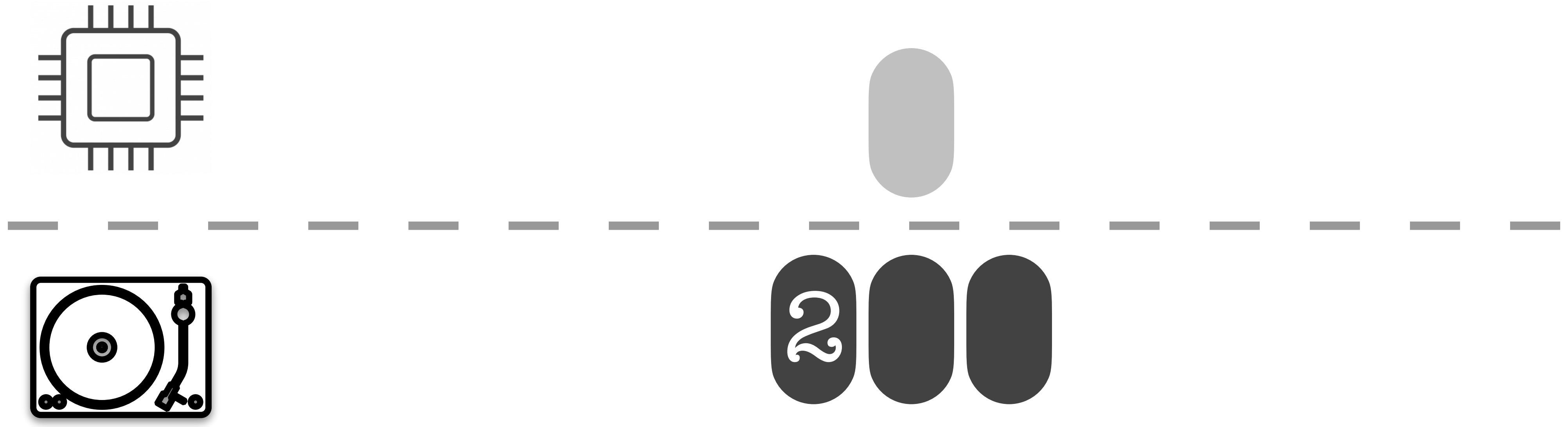
in general, **#times an entry is written to a level = 1**

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

Ingestion cost

Inserts and updates

tiered LSM-tree



in general, **#times an entry is written to a level = 1**

happens for all **L** levels on disk

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

B entries are written to disk per I/O

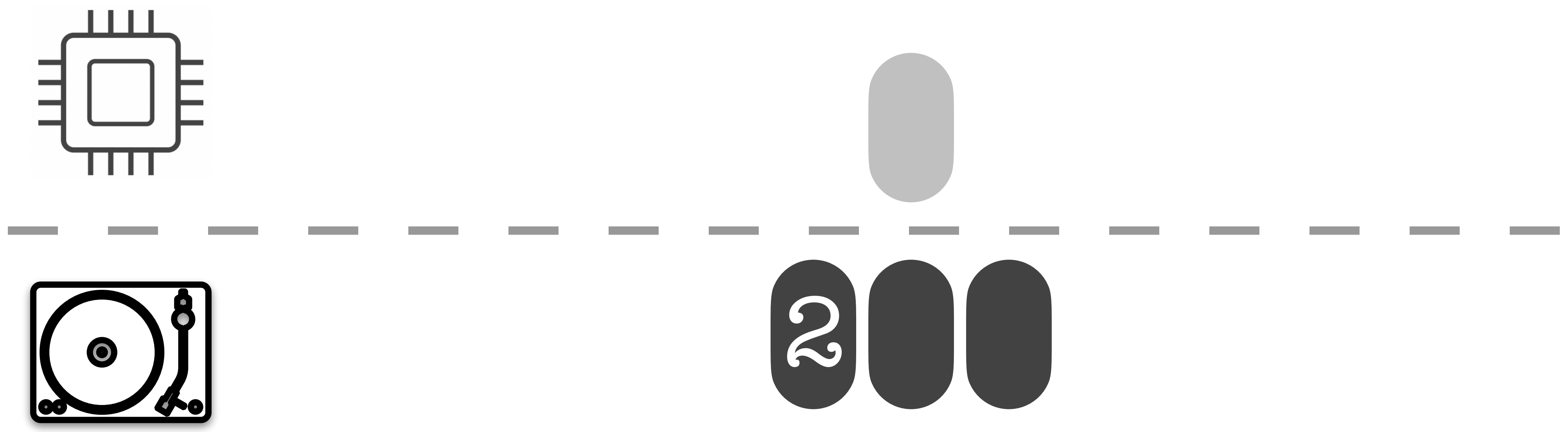
Brandeis UNIVERSITY

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

Ingestion cost

Inserts and updates

tiered LSM-tree



in general, **#times an entry is written to a level = 1**

happens for all **L** levels on disk

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

B entries are written to disk per I/O

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

Brandeis
UNIVERSITY

Cost analysis

Counting all I/Os

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

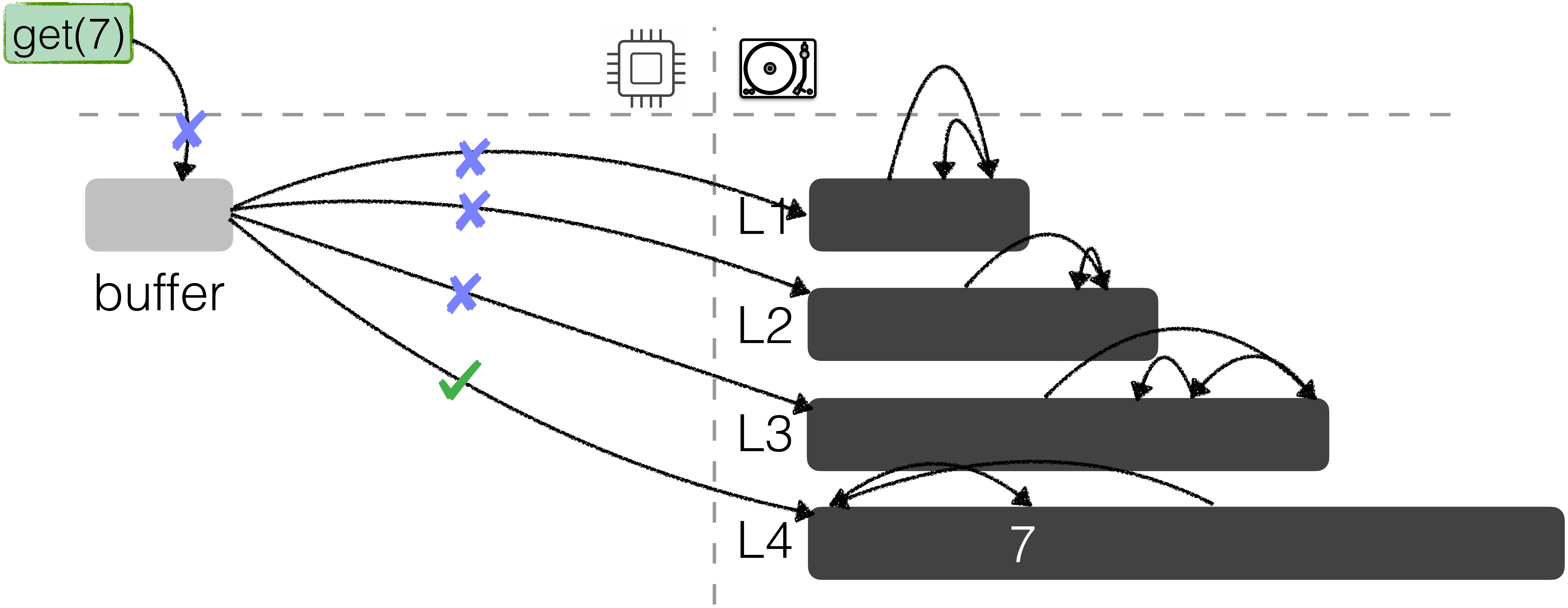
Point lookup cost

Looking for a specific key

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

Point lookup cost

Looking for a specific key

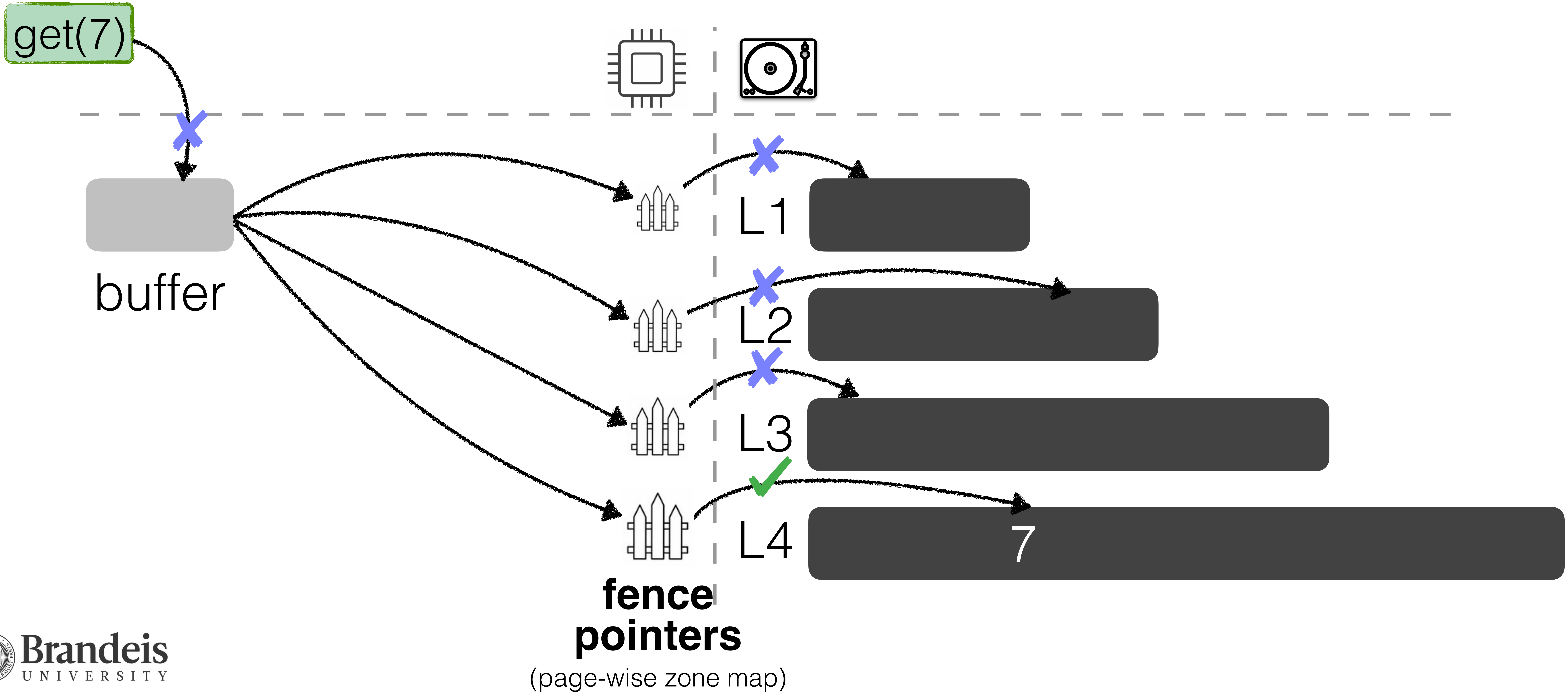


Brandeis UNIVERSITY

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

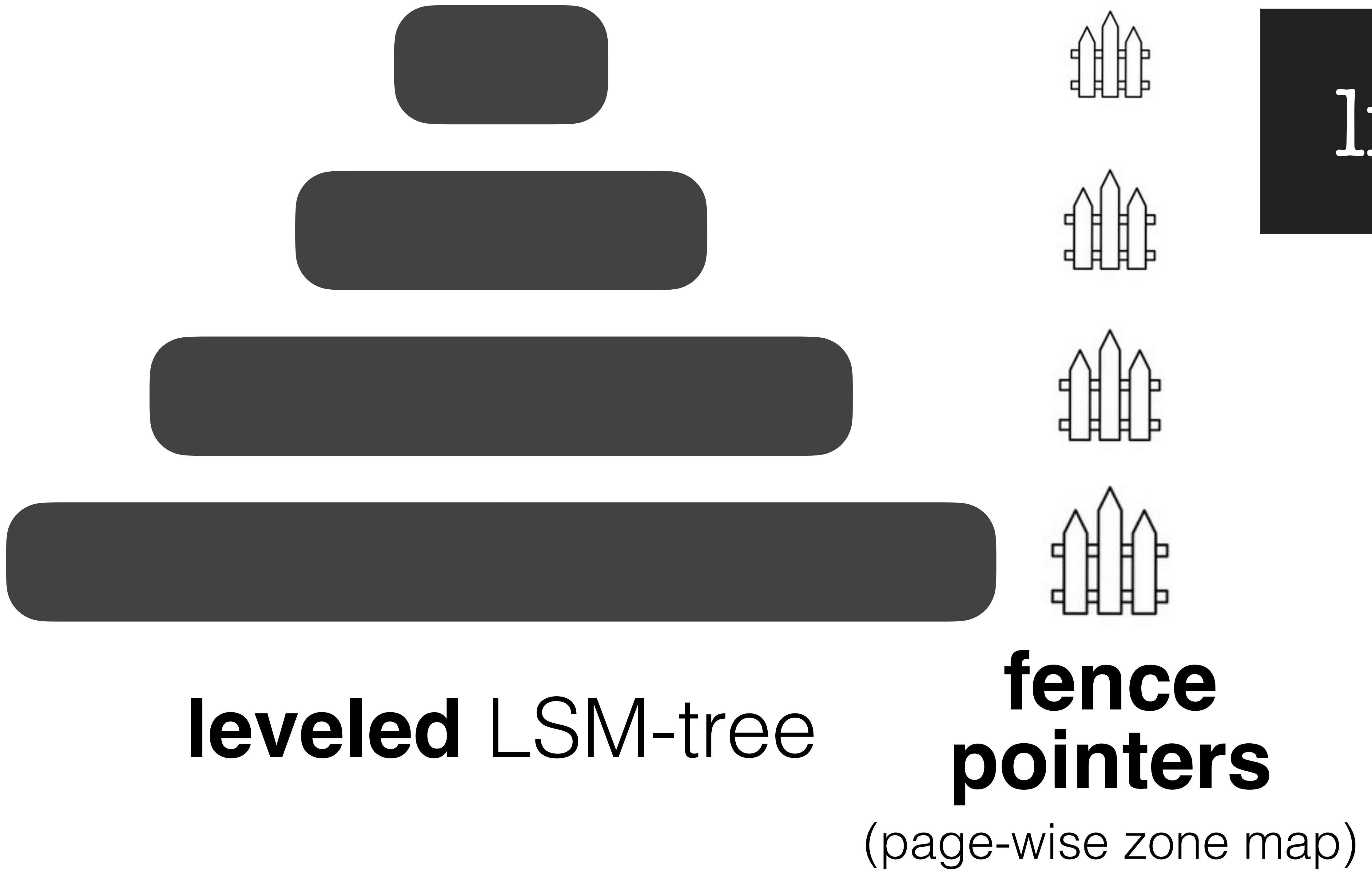
Point lookup cost

Looking for a specific key



Point lookup cost

Looking for a specific key



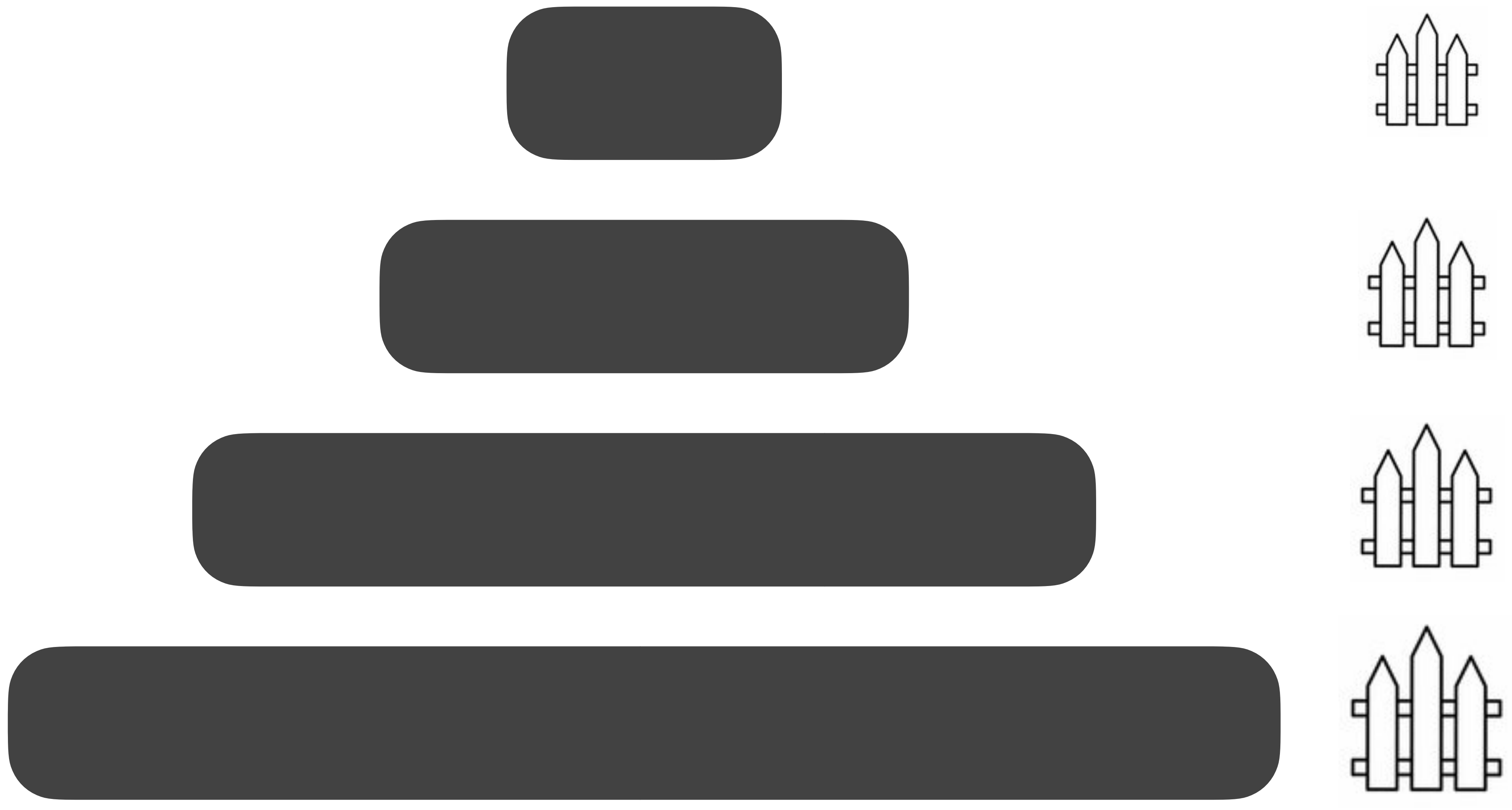
Fence pointers

limit #I/Os per sorted run (level) = 1

Cost of a point lookup = L

Point lookup cost

Looking for a specific key



leveled LSM-tree

fence pointers
(page-wise zone map)

Fence pointers

limit #I/Os per sorted run (level) = 1

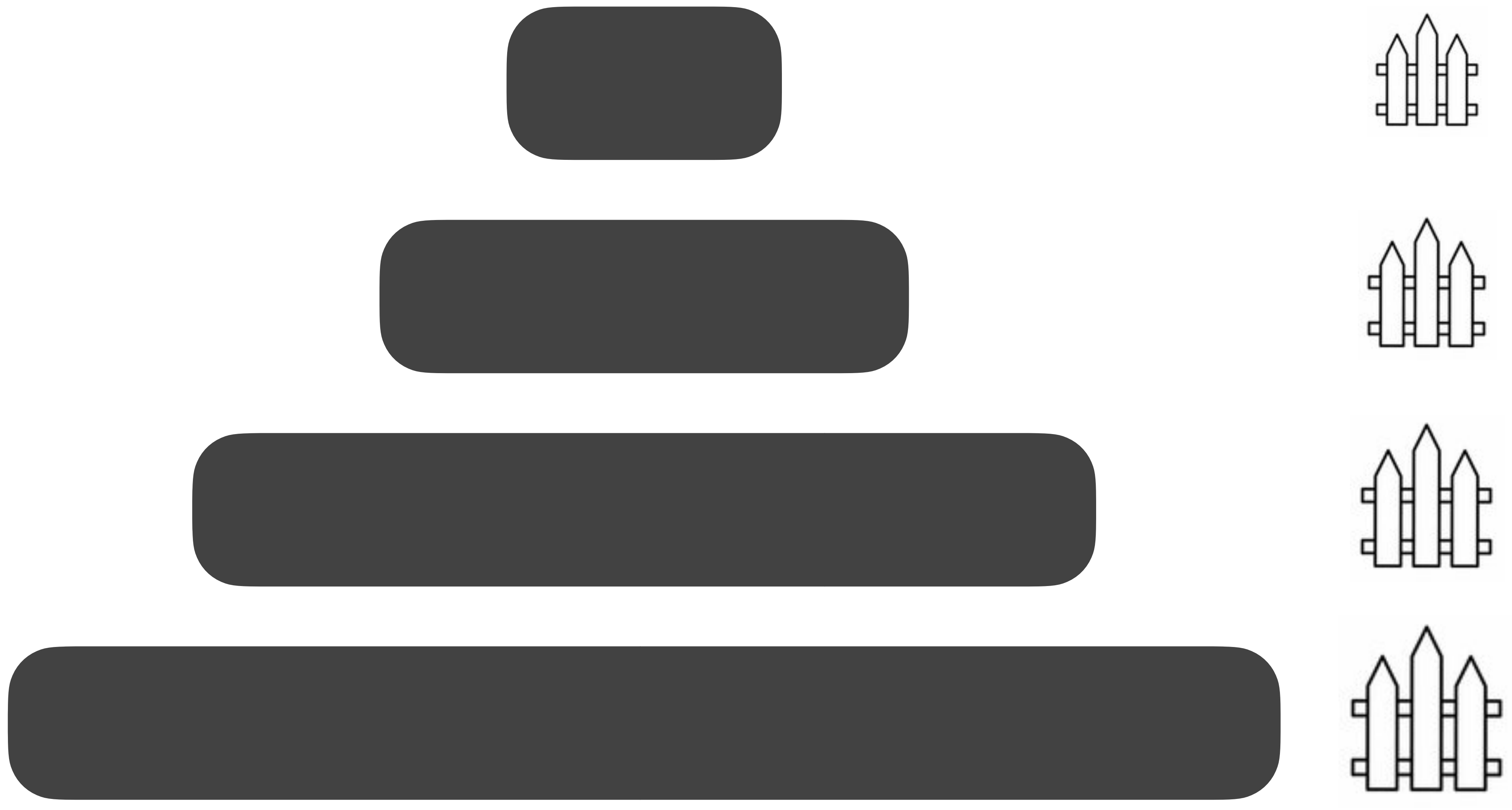
Cost of a point lookup = L

Thought Experiment 3

What is the point lookup cost for entries **not in the tree**?

Point lookup cost

Looking for a specific key



leveled LSM-tree

fence pointers

(page-wise zone map)

Fence pointers

limit #I/Os per sorted run (level) = 1

Cost of a point lookup = L

Thought Experiment 3

What is the point lookup cost for entries **not in the tree**?

Same!

Cost analysis

Counting all I/Os

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

Cost analysis

Counting all I/Os

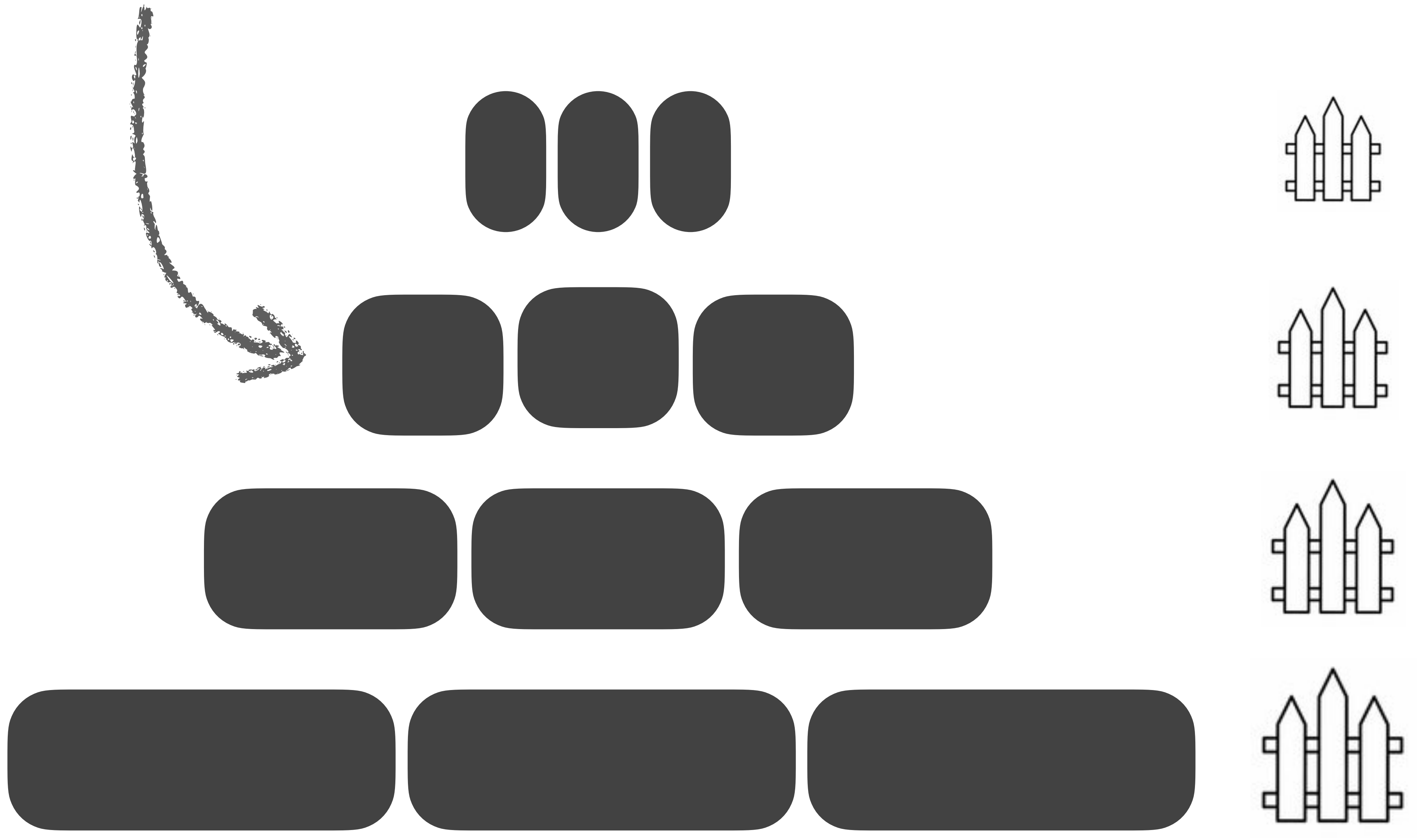
data structure	ingestion cost	point lookup cost*	range lookup cost
Leveled LSM-tree	$O(L \cdot T / B)$	$O(L)$	
Tiered LSM-tree	$O(L / B)$		
B^+-tree			
Sorted array			
Log			

* with **fence pointers**
same cost for empty and non-empty lookups

Point lookup cost

Looking for a specific key

T sorted runs or tiers
per level



leveled LSM-tree

fence pointers
(page-wise zone map)

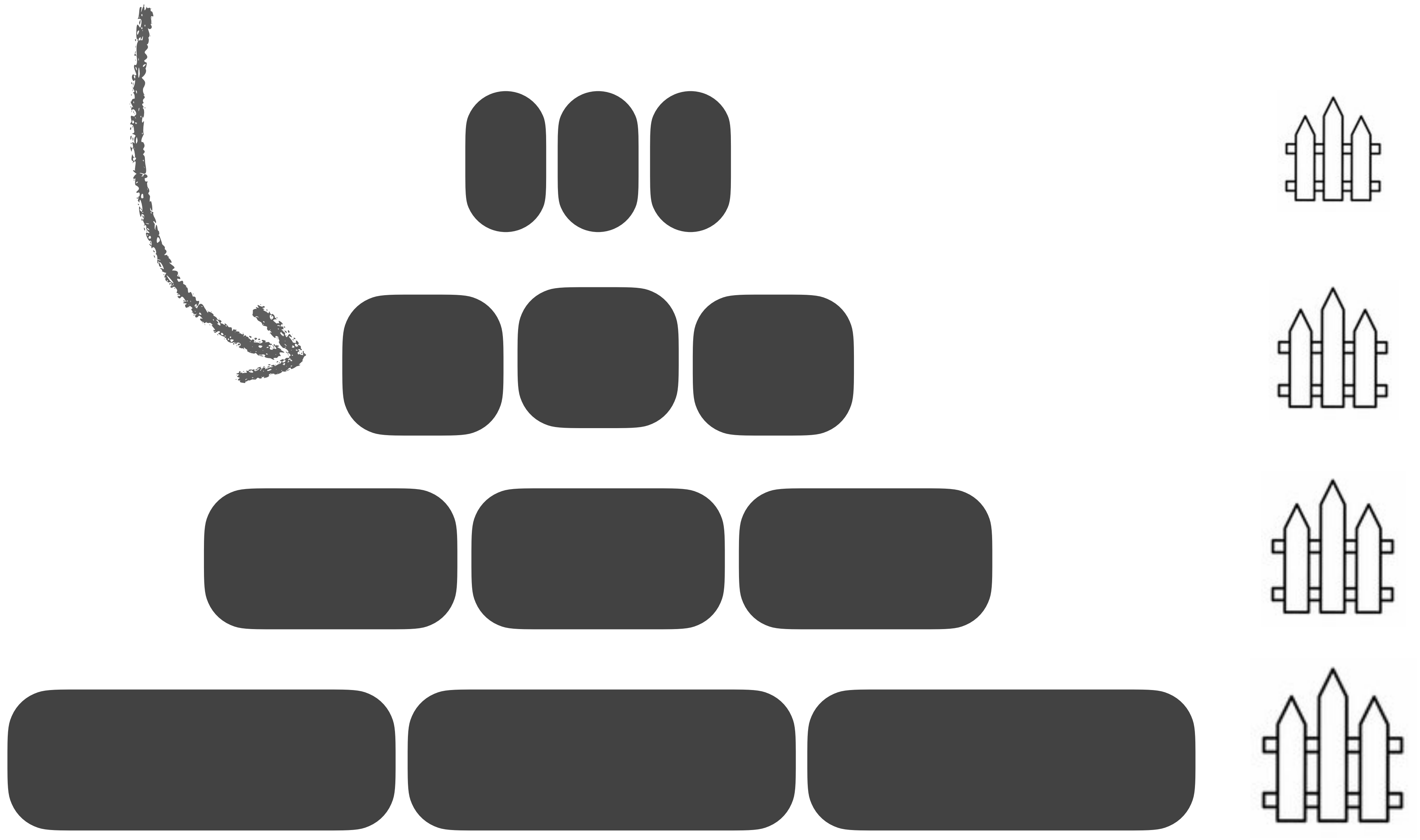
Thought Experiment 4

Point lookup cost in **tiered LSM**?

Point lookup cost

Looking for a specific key

T sorted runs or tiers
per level



leveled LSM-tree

fence pointers
(page-wise zone map)

Thought Experiment 4

Point lookup cost in **tiered LSM**?

Fence pointers

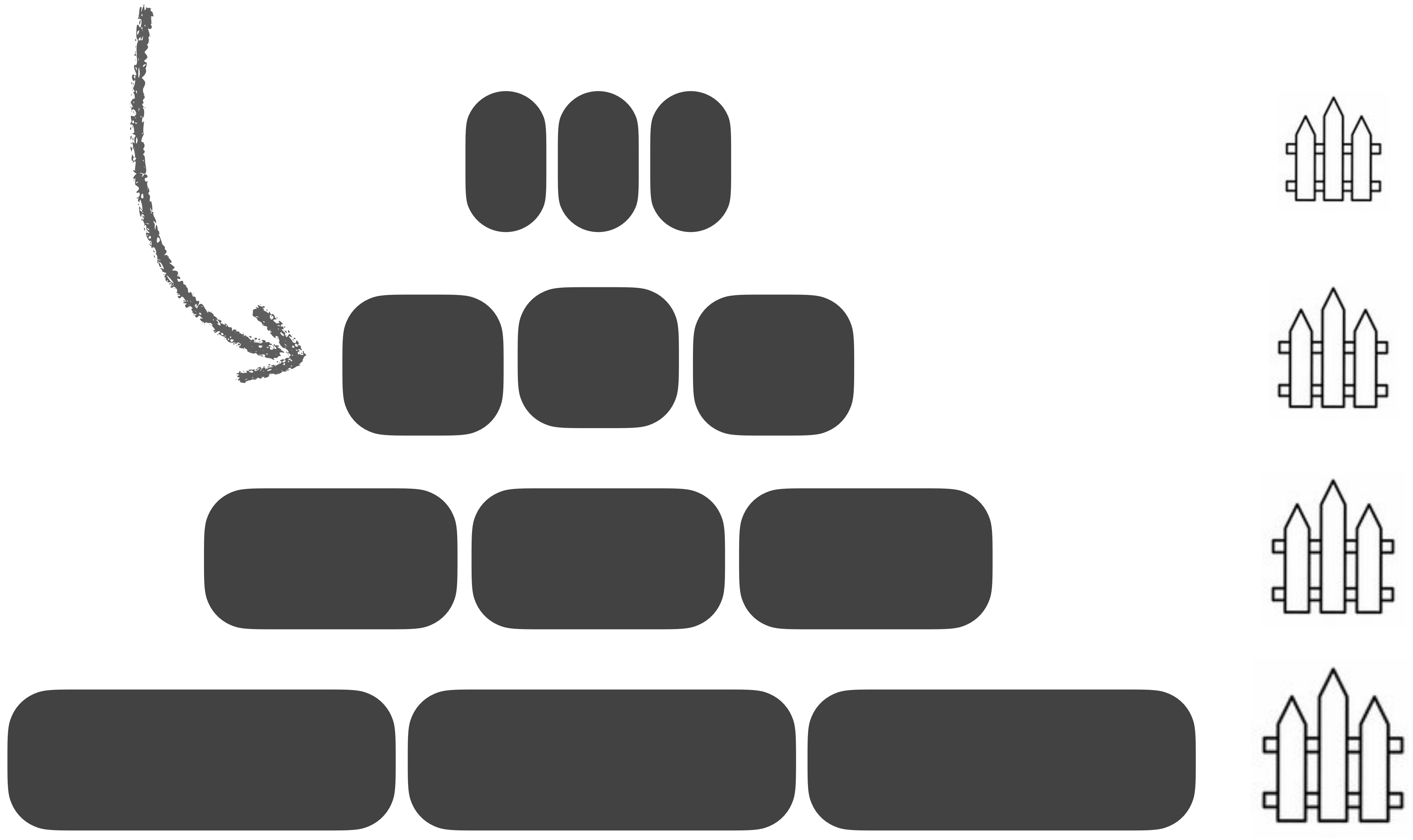
limit #I/Os per sorted run = 1

T sorted runs per level

Point lookup cost

Looking for a specific key

T sorted runs or tiers
per level



leveled LSM-tree

fence pointers
(page-wise zone map)

Thought Experiment 4

Point lookup cost in **tiered LSM**?

Fence pointers

limit #I/Os per sorted run = 1

T sorted runs per level

Cost of a point lookup = $L \cdot T$

Brandeis
UNIVERSITY

Cost analysis

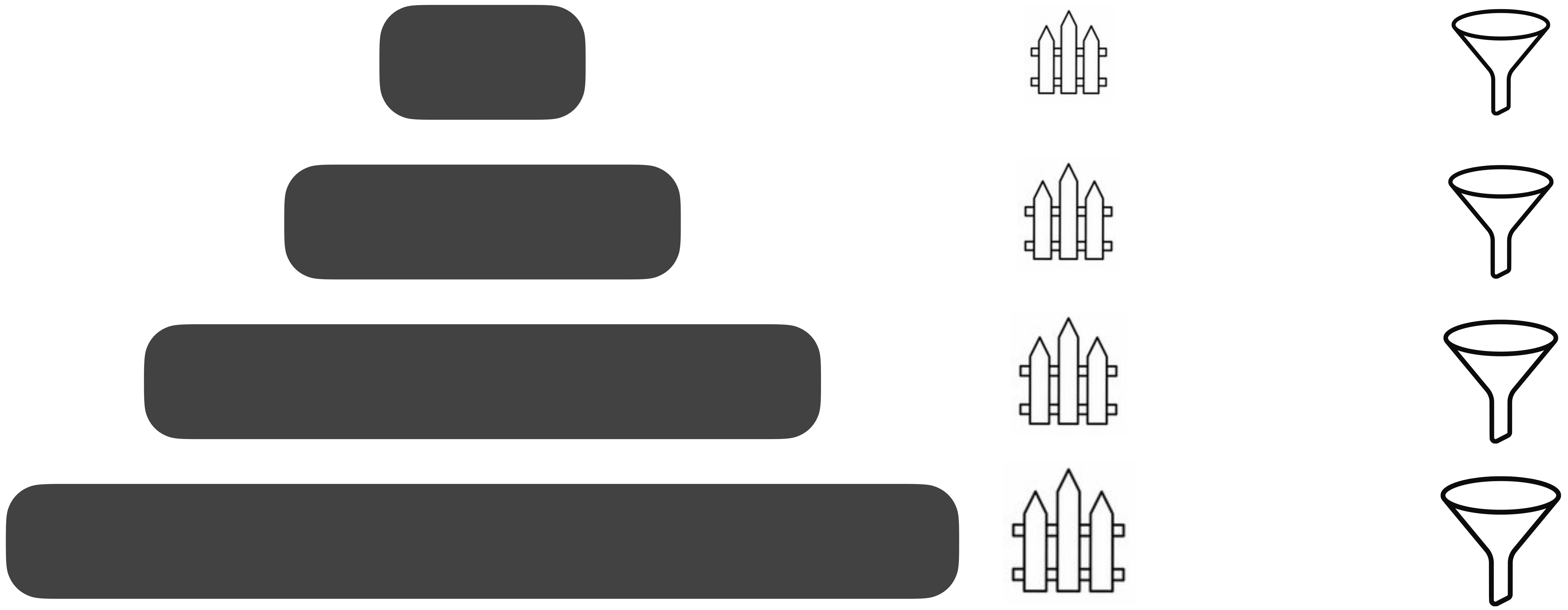
Counting all I/Os

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

Lookups are still very costly! Can we do better?

Point lookup cost

Looking for a specific key

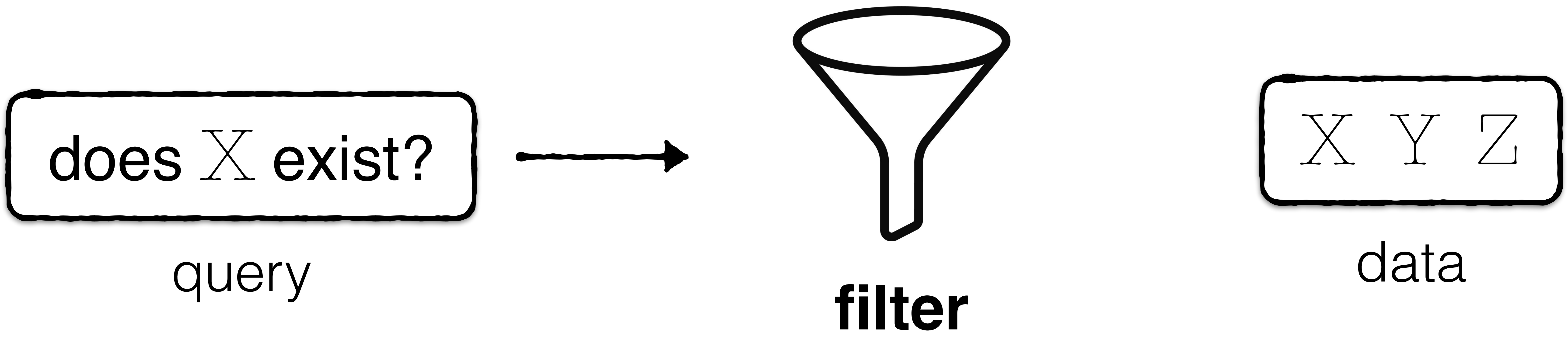


What is a **filter?**

Answers membership queries

answers

membership queries

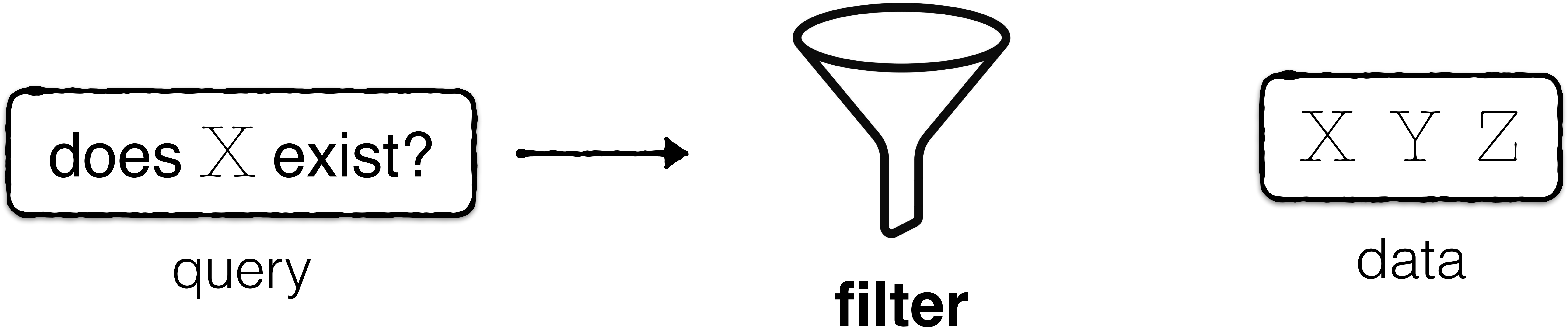


What is a **filter?**

Answers membership queries

answers

membership queries



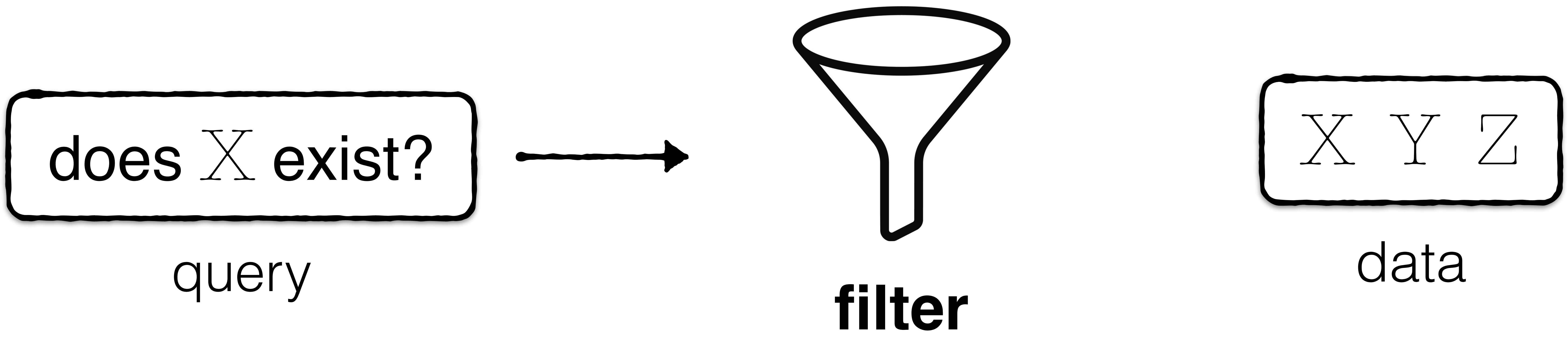
no false negatives

Brandeis UNIVERSITY

What is a **filter?**

Answers membership queries

answers
membership queries

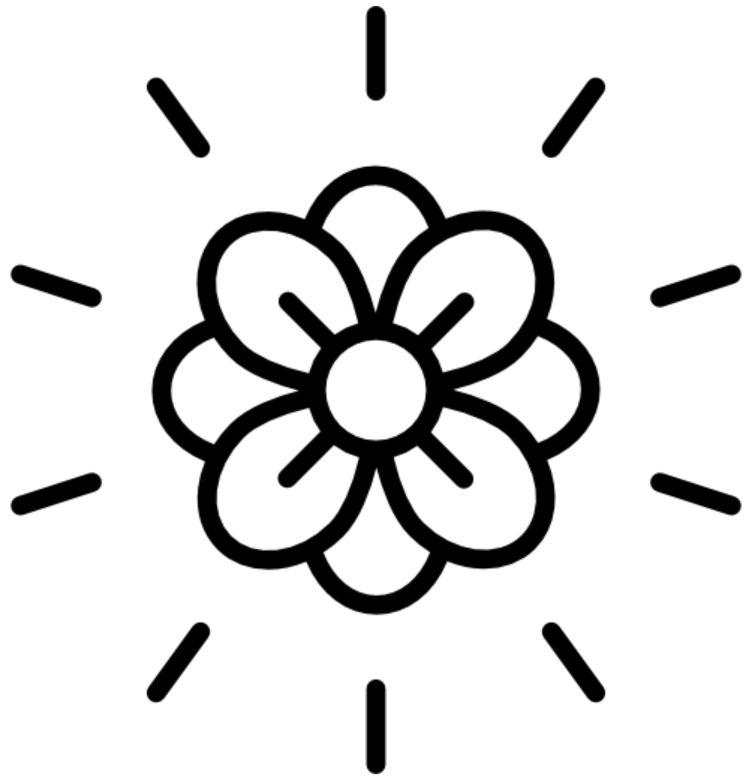


no false negatives

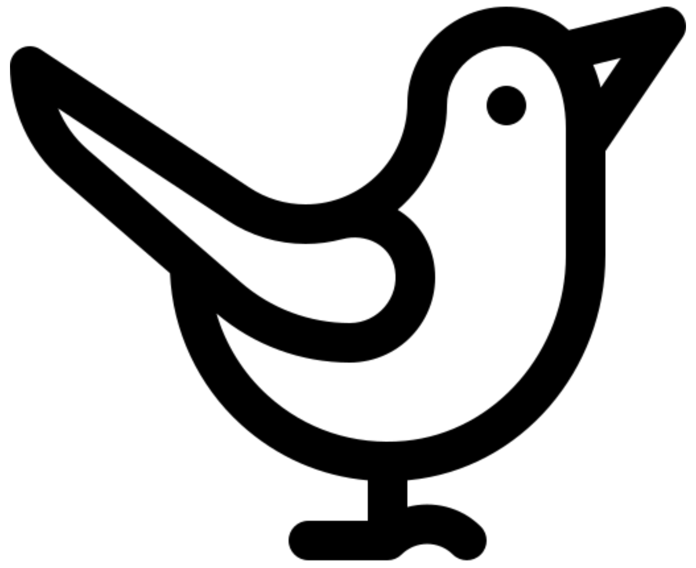
false positives with tunable probability

Types of **filters?**

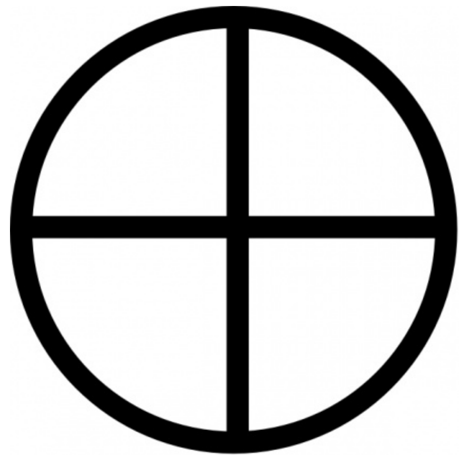
Point and range filters



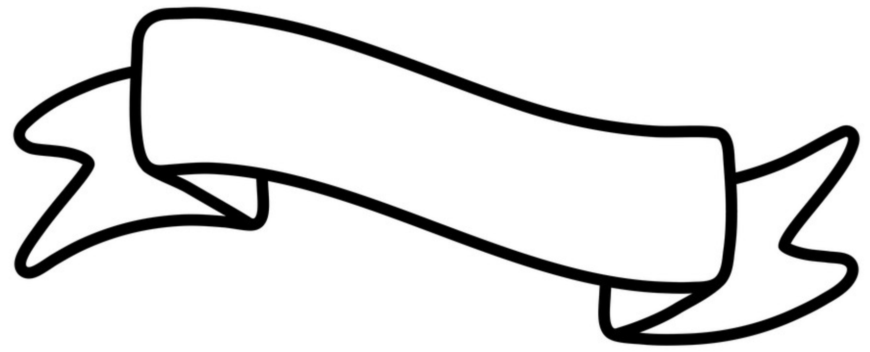
Bloom



Cuckoo



XOR



Ribbon

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

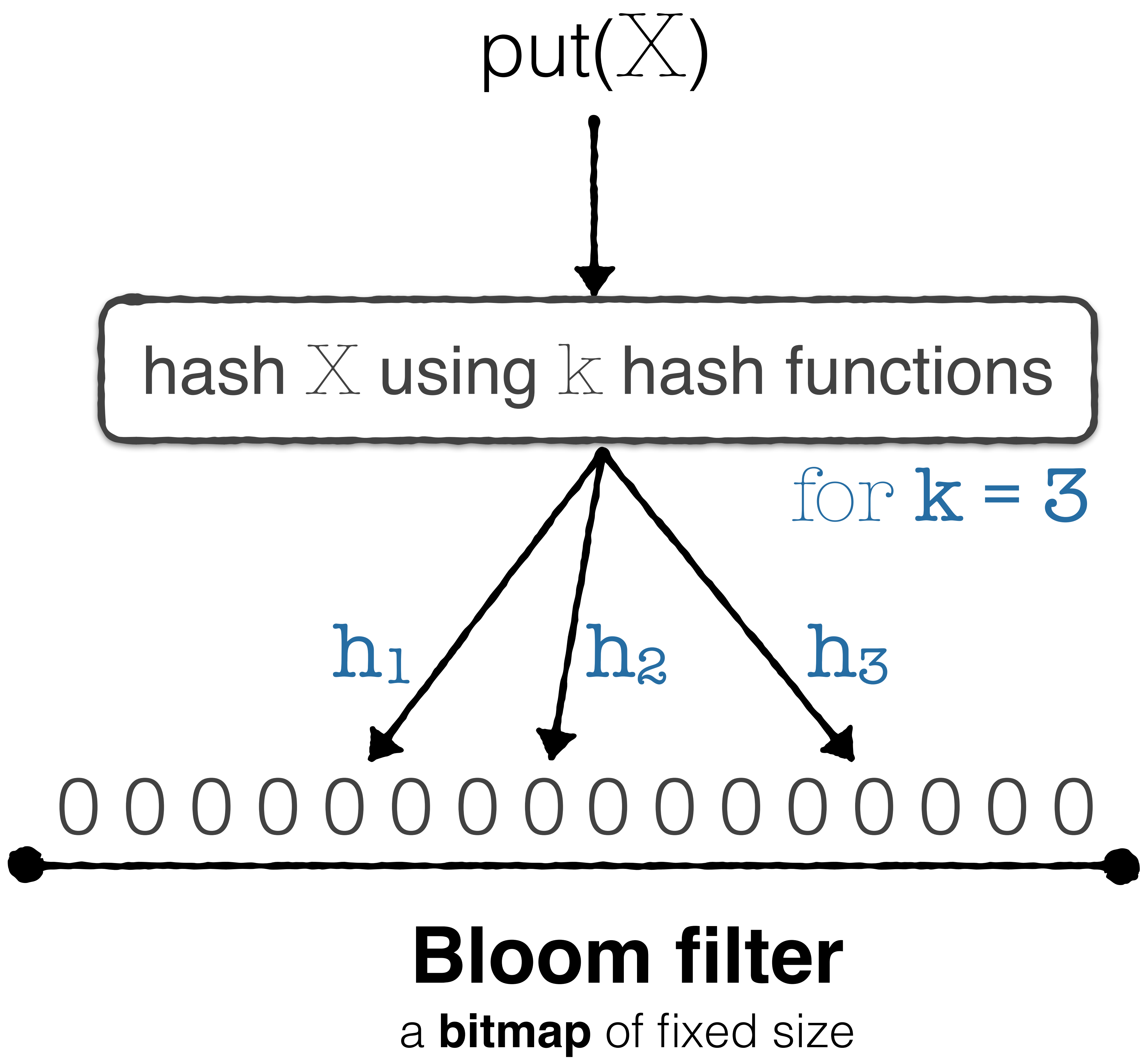
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bloom filter

a **bitmap** of fixed size

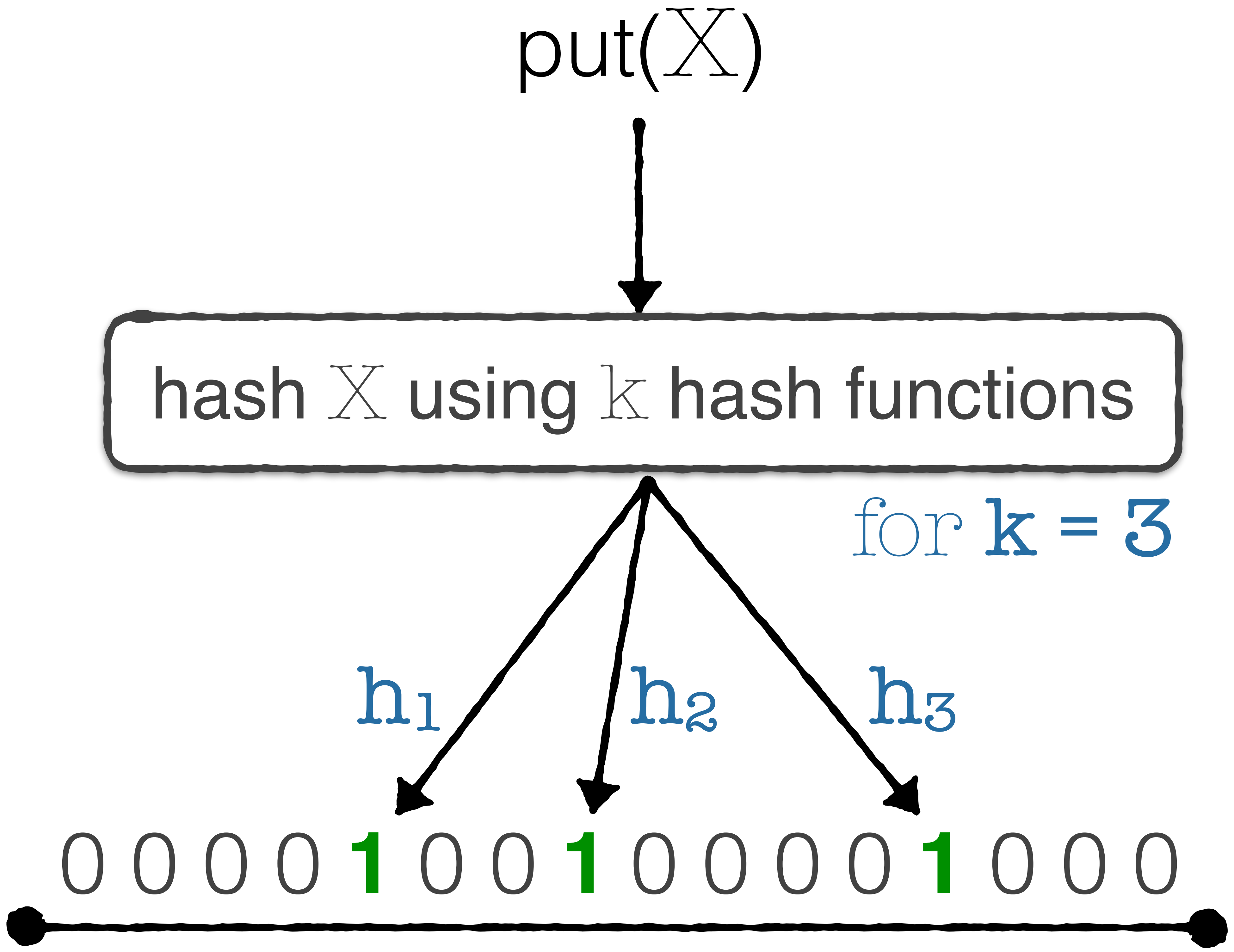
Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Bloom filter

a **bitmap** of fixed size

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

get(X)

hash X using k hash functions

for **k = 3**

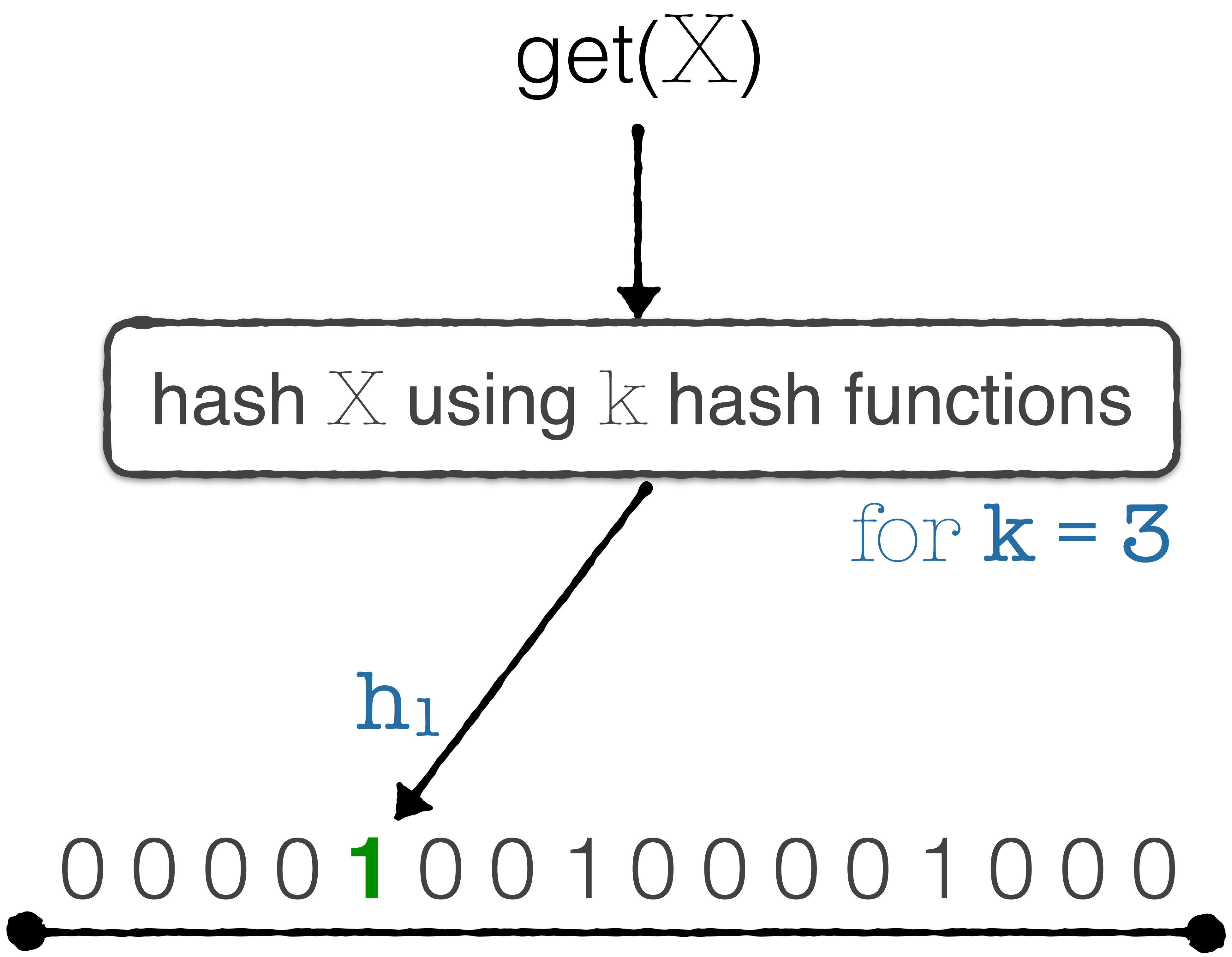
0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0

Bloom filter

a **bitmap** of fixed size

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

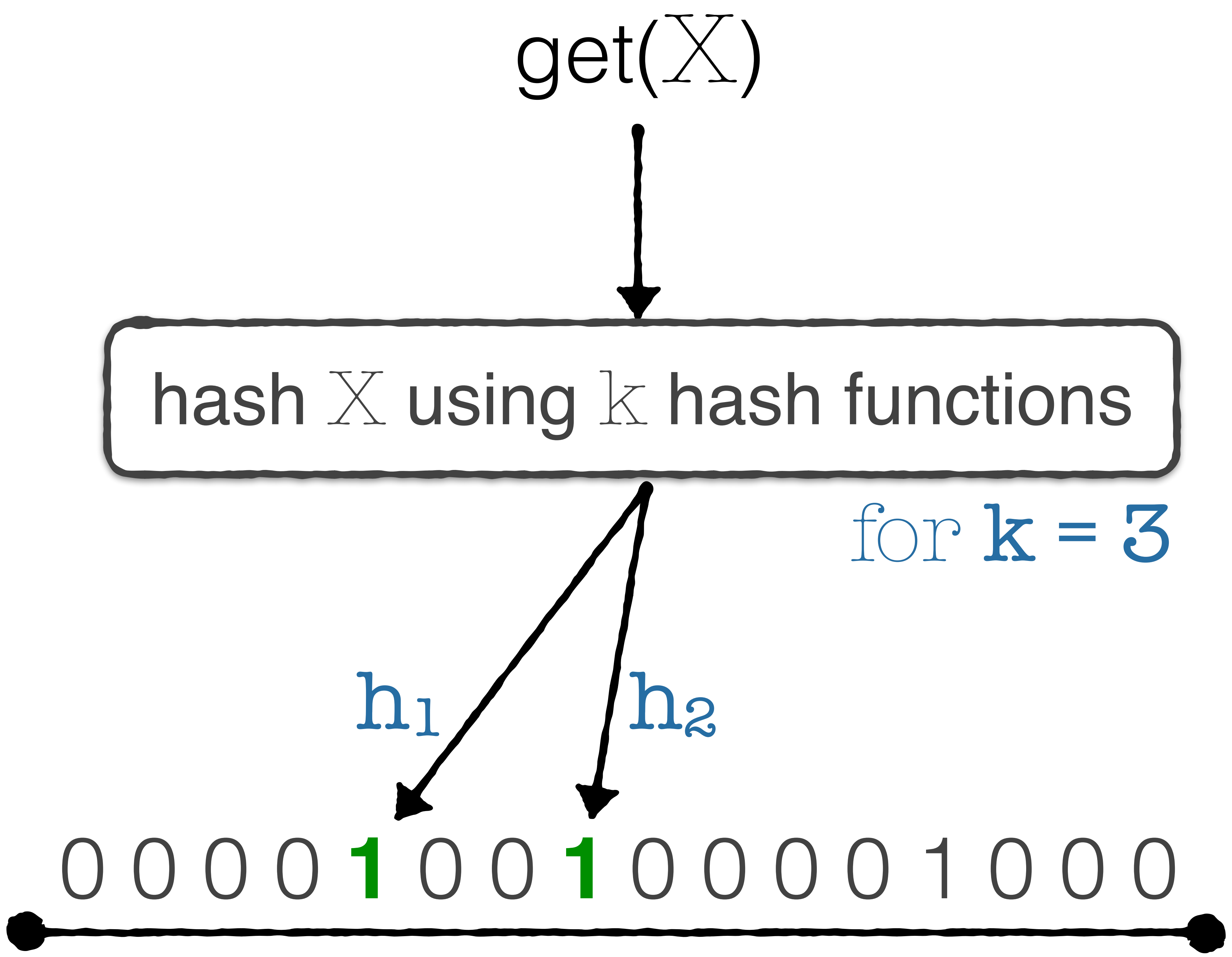


Bloom filter

a **bitmap** of fixed size

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



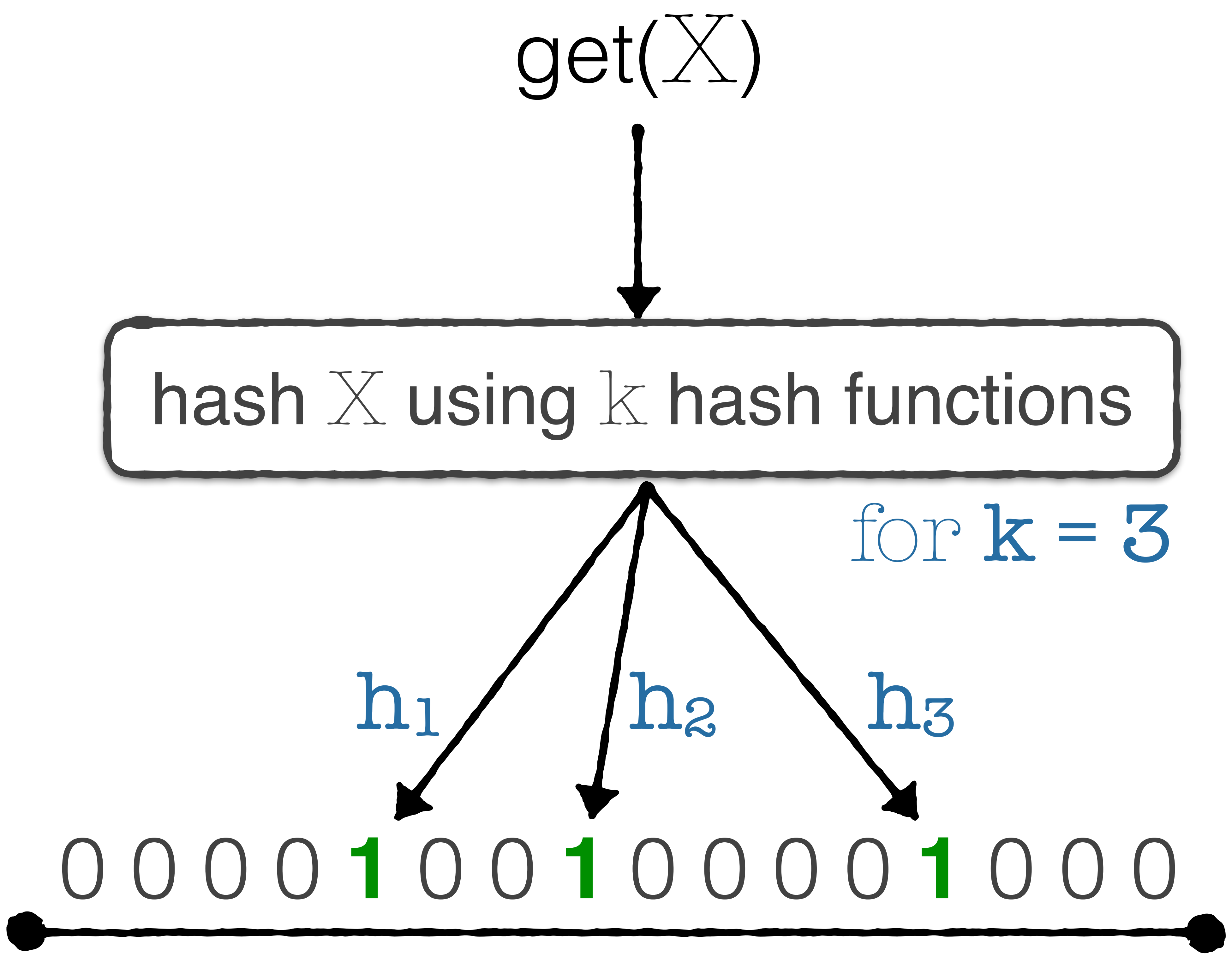
Bloom filter

a **bitmap** of fixed size

Brandeis UNIVERSITY

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Bloom filter

a **bitmap** of fixed size

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

Thought Experiment 5

When does a Bloom filter return a **false positive** result?

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bloom filter

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

Thought Experiment 5

When does a Bloom filter return a **false positive** result?

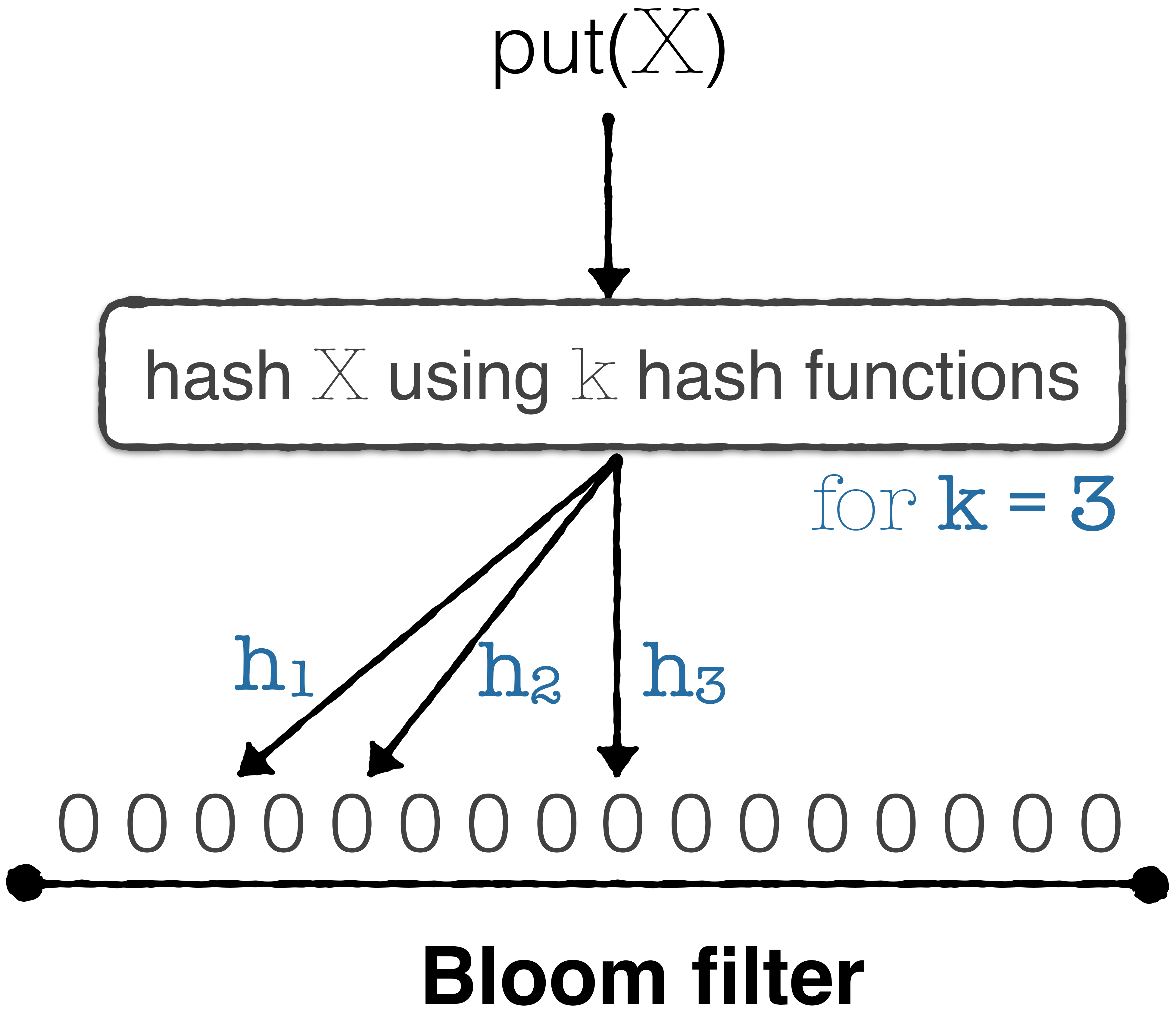
put(X) ⟶ hashes bits {3,5,9}

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bloom filter

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



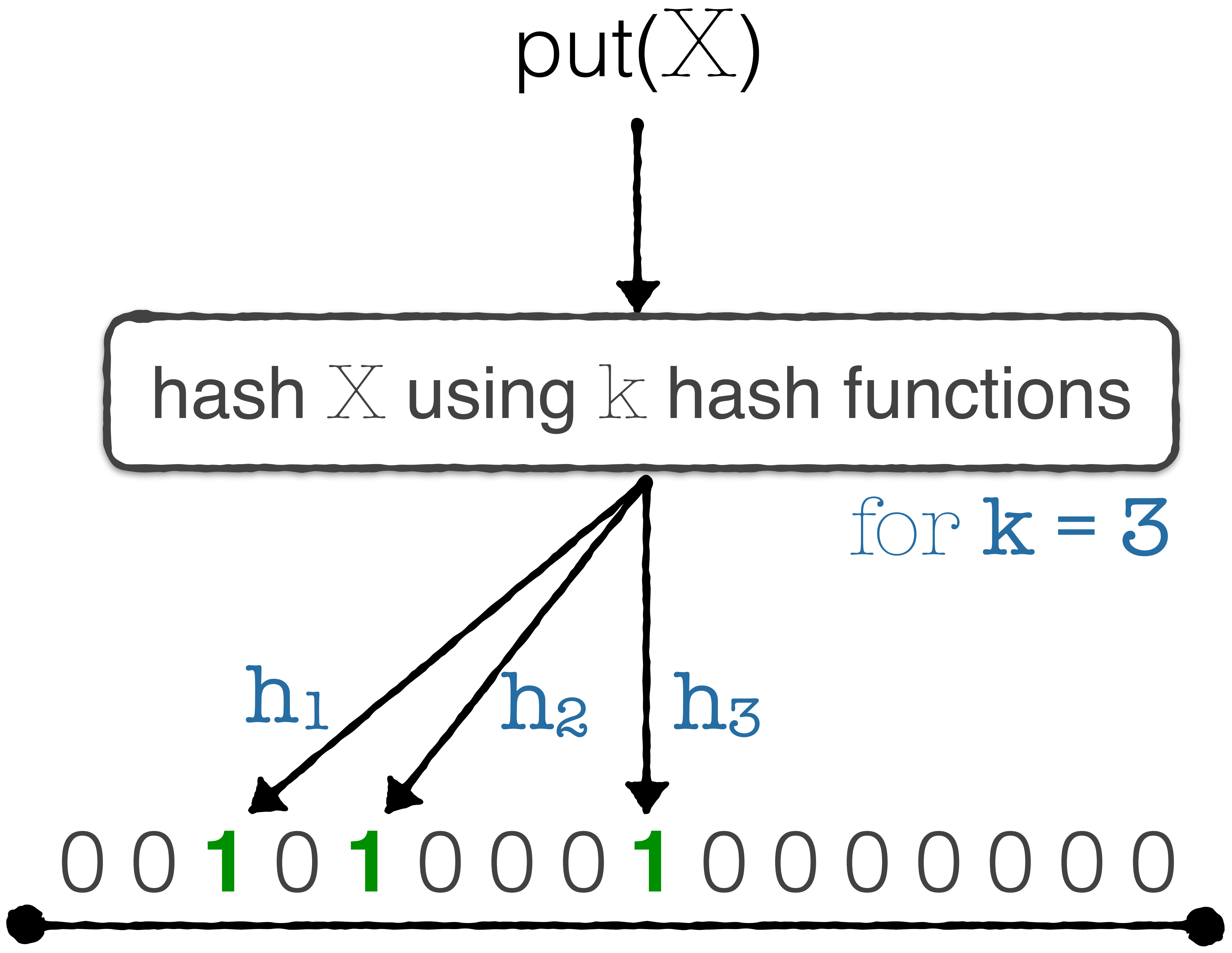
Thought Experiment 5

When does a Bloom filter return a **false positive** result?

put(X) ⟶ hashes bits $\{3,5,9\}$

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Bloom filter

Thought Experiment 5

When does a Bloom filter return a **false positive** result?

put(X) ⟶ hashes bits $\{3,5,9\}$

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

Thought Experiment 5

When does a Bloom filter return a **false positive** result?

put(X) ⟶ hashes bits {3,5,9}

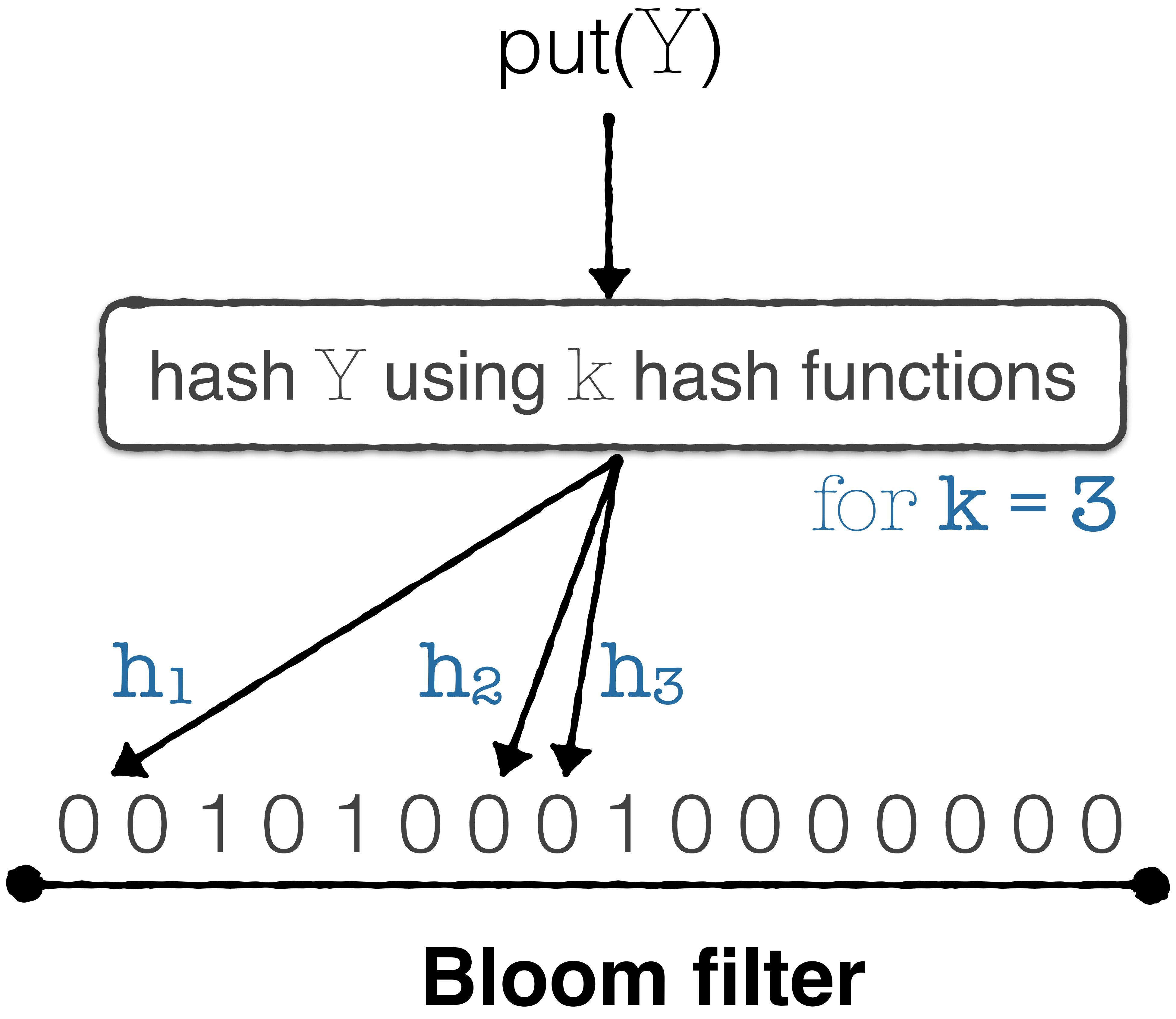
put(Y) ⟶ hashes bits {1,7,8}

0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0

Bloom filter

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Thought Experiment 5

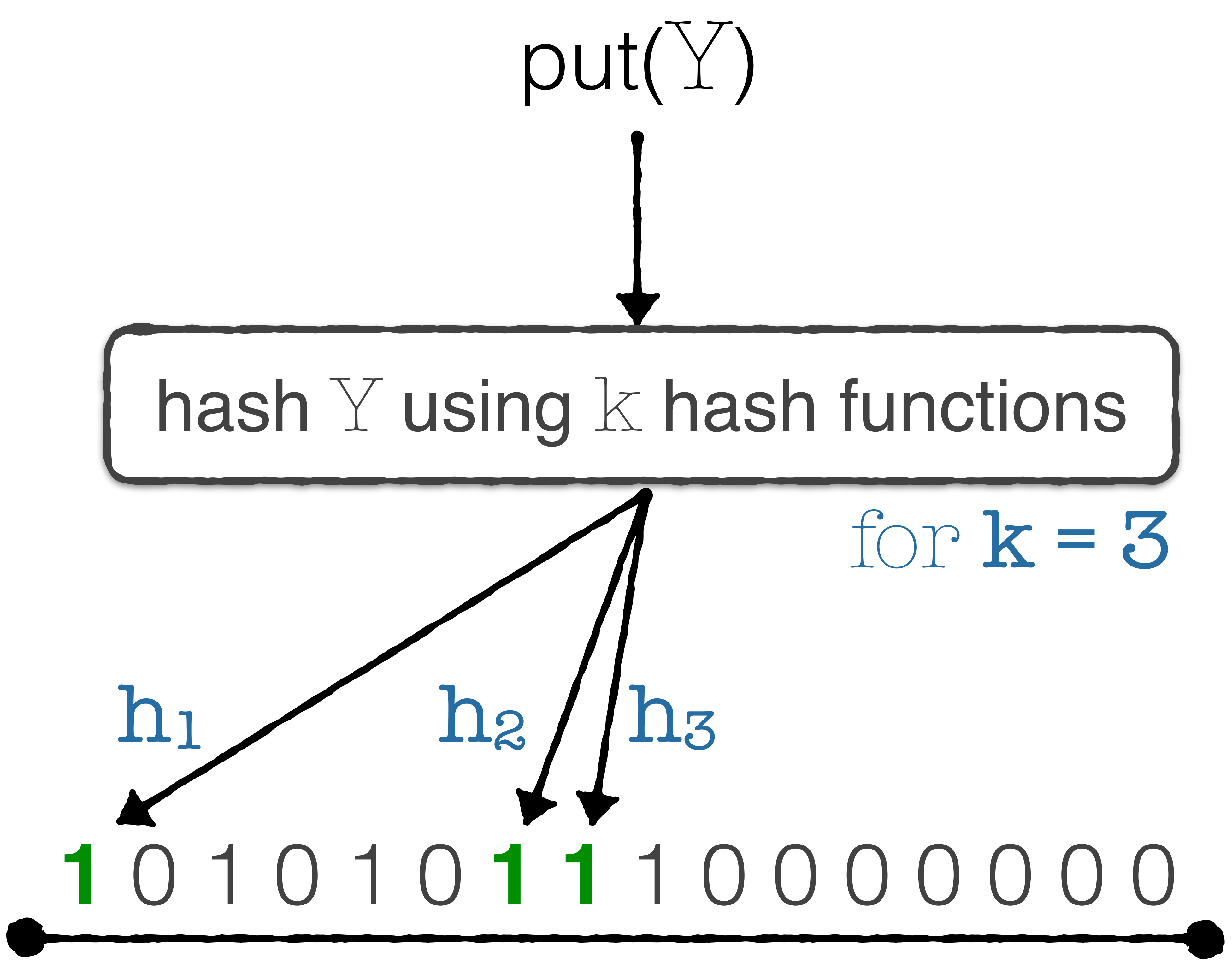
When does a Bloom filter return a **false positive** result?

put(X) ⟶ hashes bits {3,5,9}

put(Y) ⟶ hashes bits {1,7,8}

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Bloom filter

Thought Experiment 5

When does a Bloom filter return a **false positive** result?

put(X) ⟶ hashes bits $\{3,5,9\}$

put(Y) ⟶ hashes bits $\{1,7,8\}$

Bloom filter

Invented in 1970 by Burton Howard **Bloom**

Thought Experiment 5

When does a Bloom filter return a **false positive** result?

put(X) ⟶ hashes bits {3,5,9}

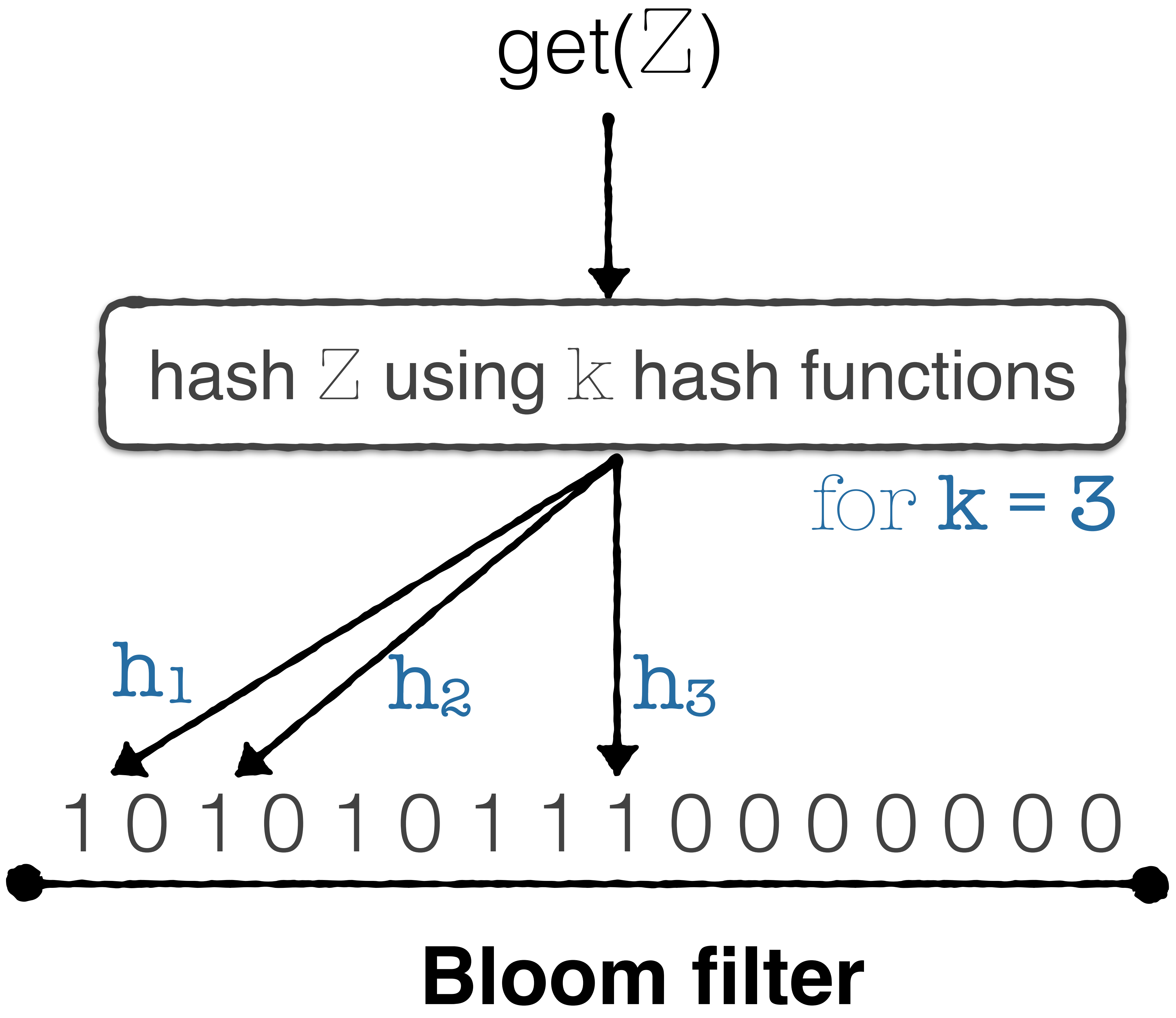
put(Y) ⟶ hashes bits {1,7,8}

get(Z) ⟶ hashes bits {1,3,9}

Bloom filter

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Thought Experiment 5

When does a Bloom filter return a **false positive** result?

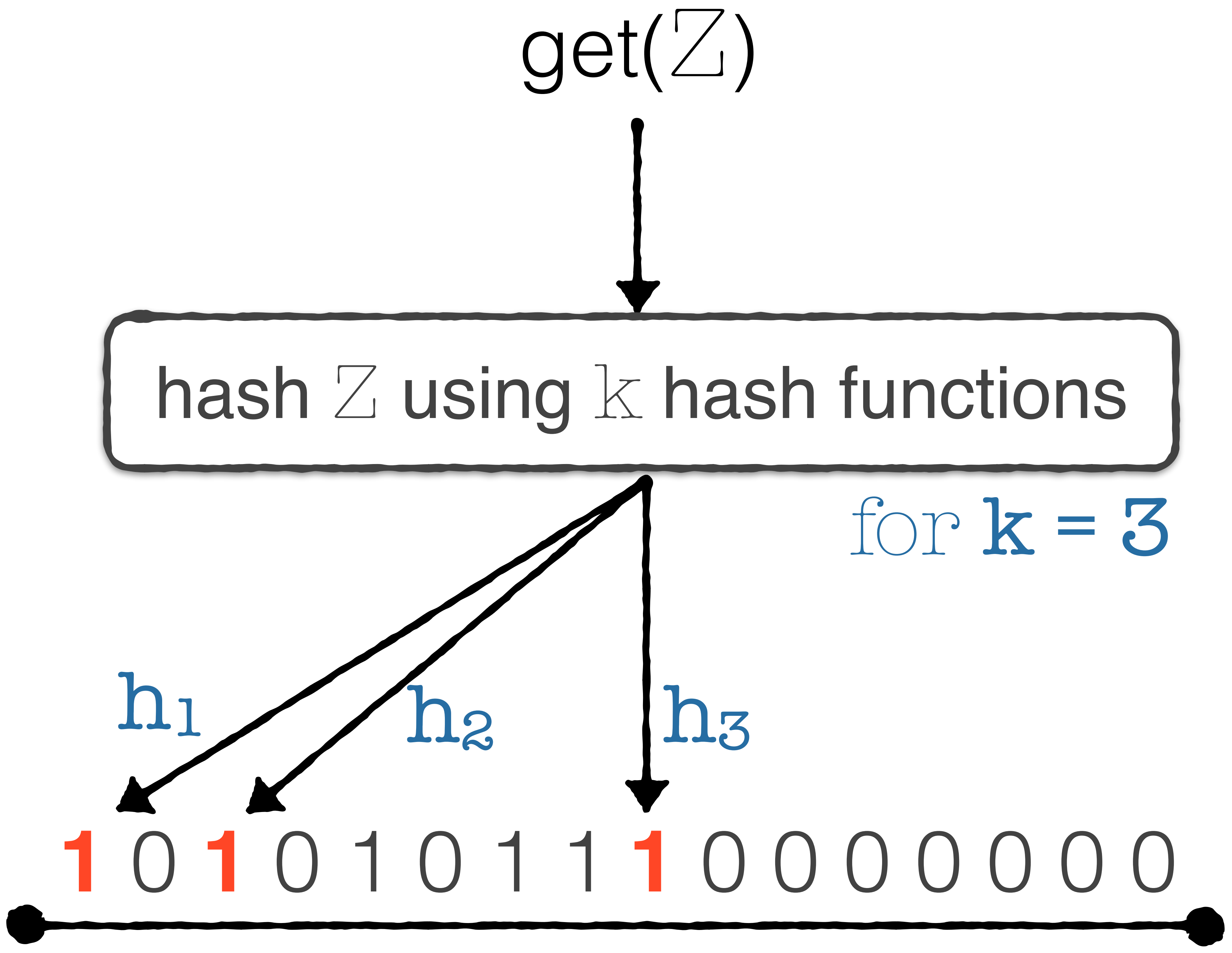
put(X) ⟶ hashes bits $\{3,5,9\}$

put(Y) ⟶ hashes bits $\{1,7,8\}$

get(Z) ⟶ hashes bits $\{1,3,9\}$

Bloom filter

Invented in 1970 by Burton Howard **Bloom**



Bloom filter

Thought Experiment 5

When does a Bloom filter return a **false positive** result?

put(X) ⟶ hashes bits {3,5,9}

put(Y) ⟶ hashes bits {1,7,8}

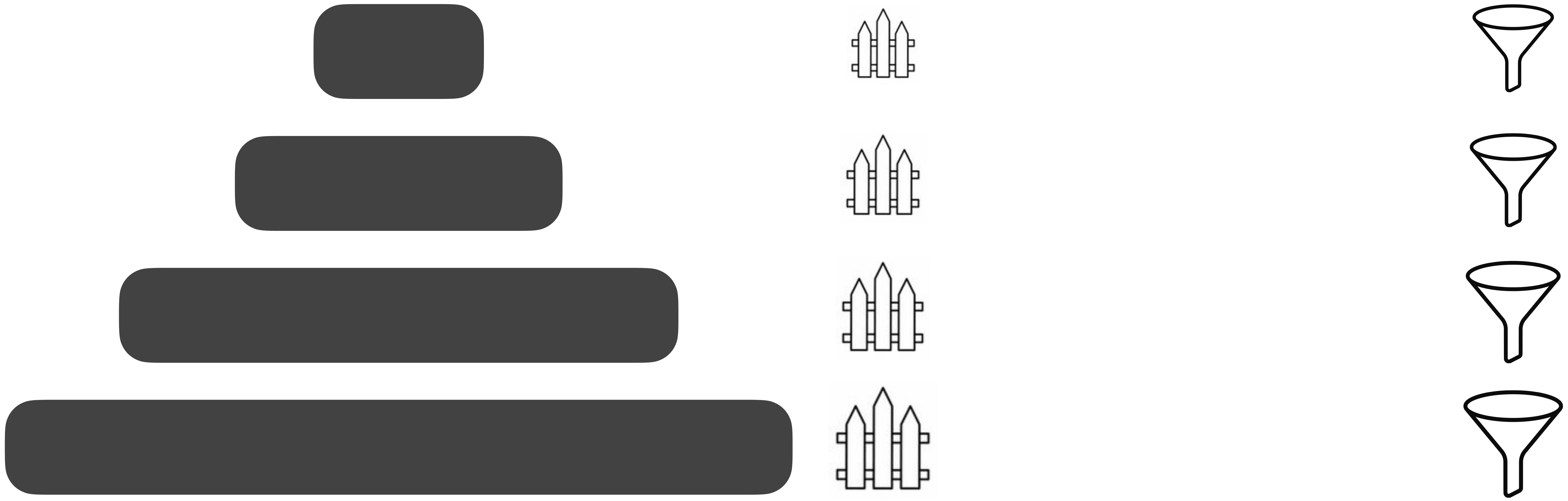
get(Z) ⟶ hashes bits {1,3,9}

false positive

Brandeis UNIVERSITY

Point lookup cost

Looking for a specific key



leveled LSM-tree

fence pointers

(page-wise zone map)

filter

(one filter per sorted run)

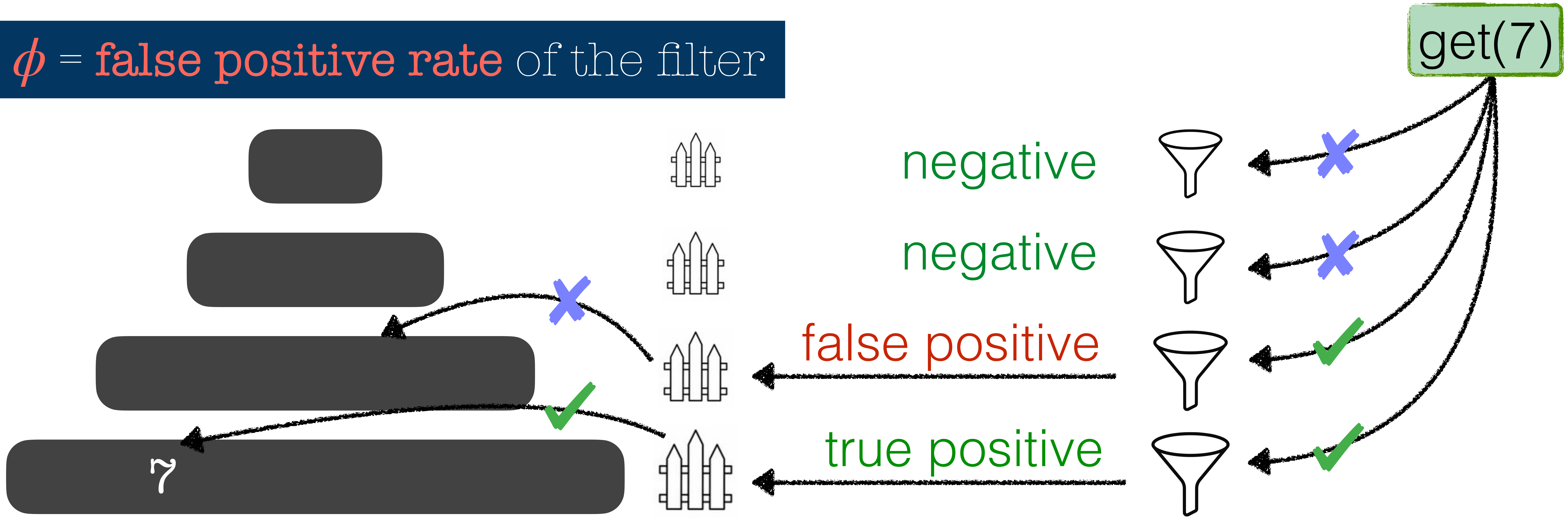
Thought Experiment 6

Cost of a **non-empty point lookup**?

Point lookup cost

Looking for a specific key

ϕ = false positive rate of the filter



leveled LSM-tree

fence pointers

(page-wise zone map)

filter

(one filter per sorted run)

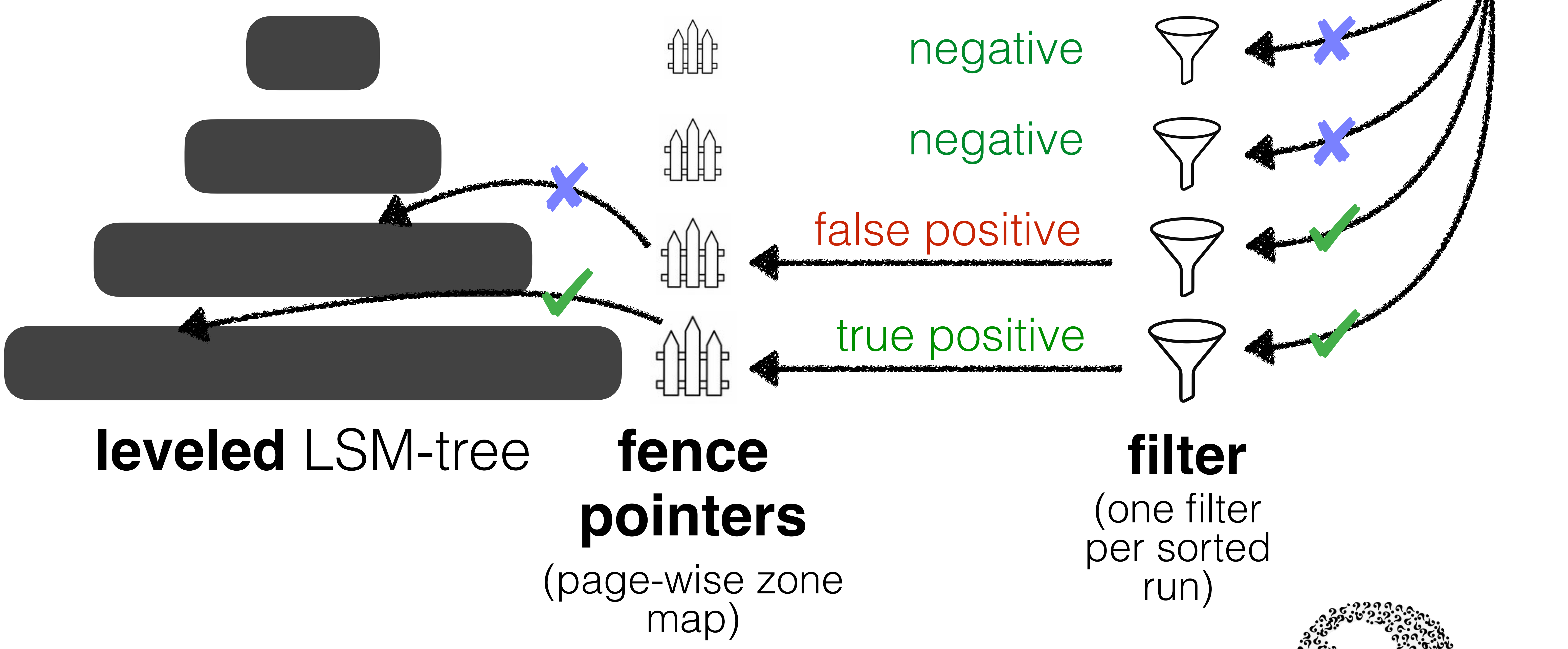
Thought Experiment 6

Cost of a **non-empty point lookup**?

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

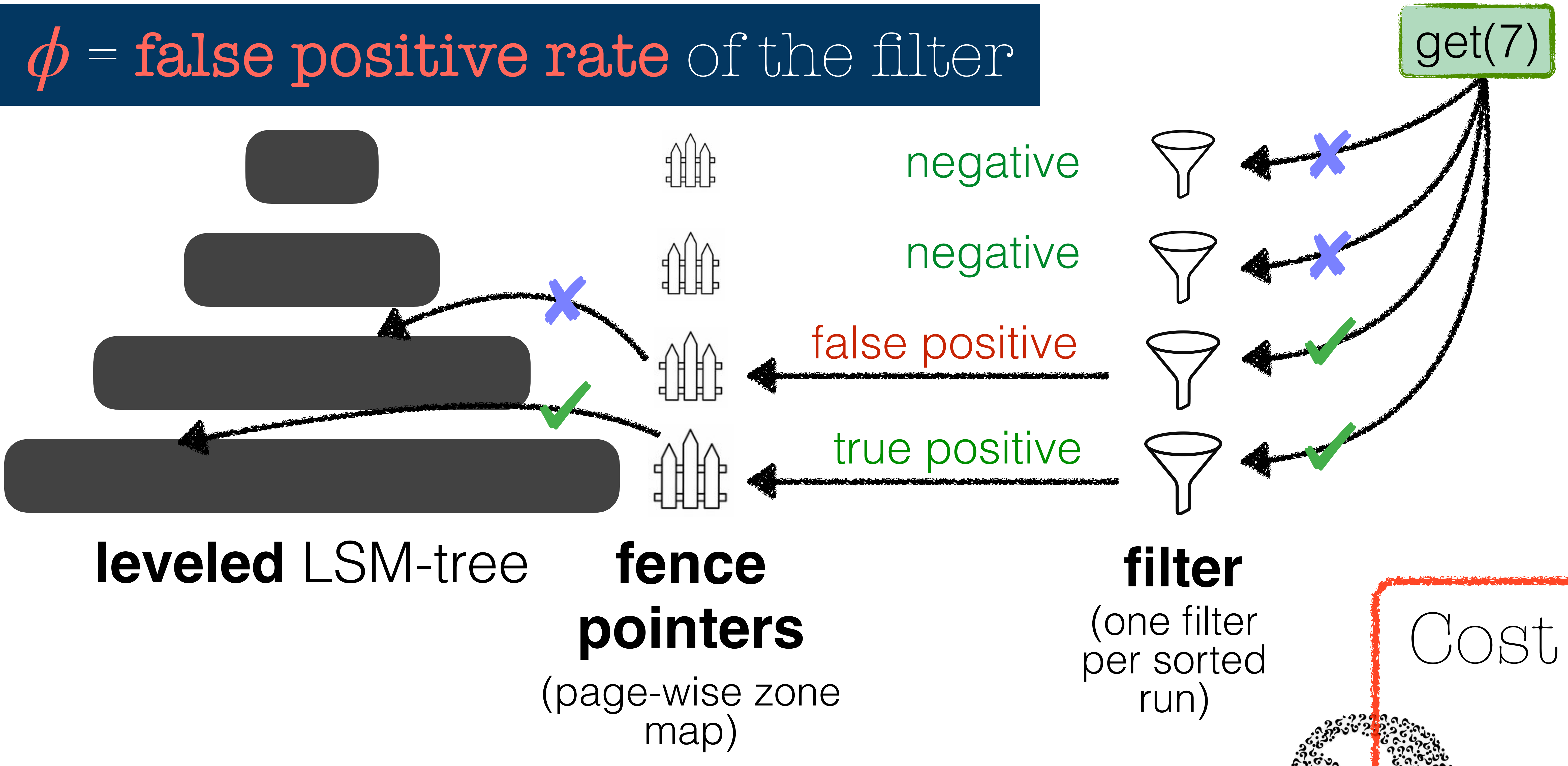
Thought Experiment 6

Cost of a **non-empty point lookup**?

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

$$= 1 + \phi \cdot (L-1)$$

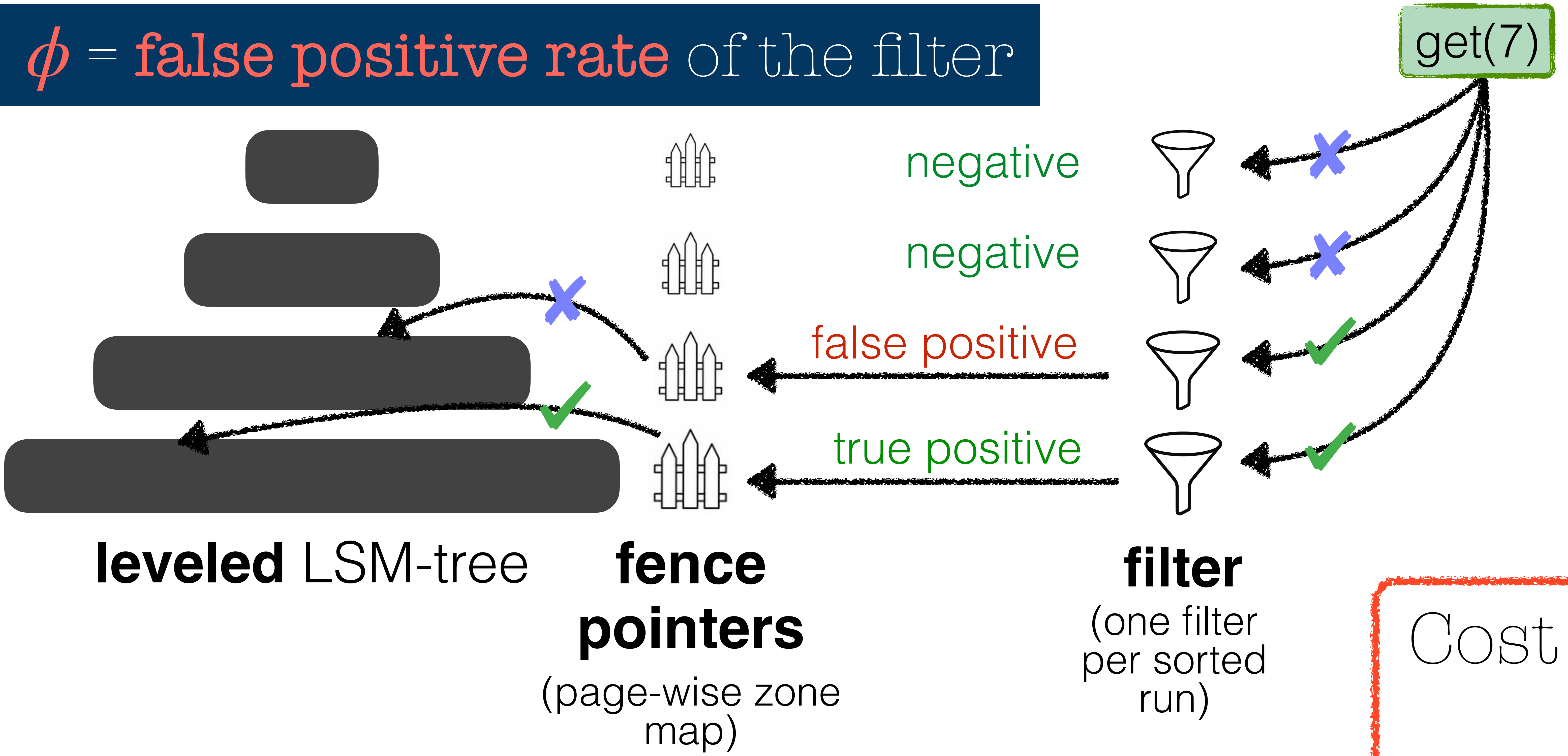
Thought Experiment 6

Cost of a **non-empty point lookup**?

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

$$= 1 + \phi \cdot (L-1)$$

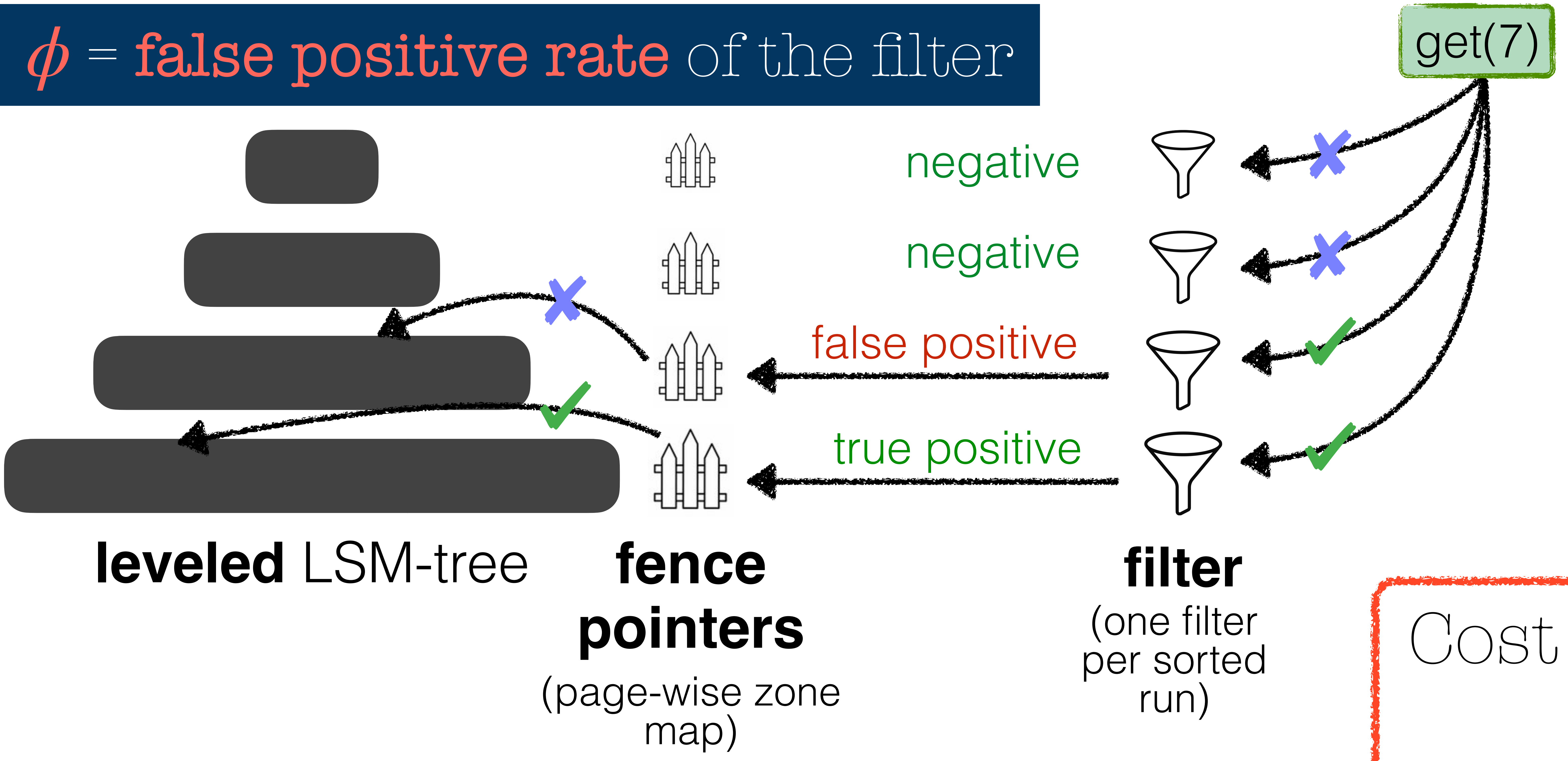
Is the lookup cost for entries **not in the tree** the same?

No

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 $= \phi \cdot L$

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	$O(L / B)$	$O(L \cdot T)$	
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)

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			

How do we **compute** ϕ?

Computing **FPR**

Looking for a specific key

ϕ = false positive rate of the filter

depends on

memory available for filter (bits per entry) ← M

#hash functions used ← optimal = $\ln(2) \cdot M$

optimal FPR for Bloom filters = $2^{-M \cdot \ln(2)}$

Computing **FPR**

Looking for a specific key

ϕ = false positive rate of the filter

depends on

memory available for filter (bits per entry) ← M

#hash functions used ← optimal = $\ln(2) \cdot M$

optimal FPR for Bloom filters = $2^{-M \cdot \ln(2)}$

with 10 bits/entry: $\phi = 0.008$

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**!

Monkey: Optimal Navigable Key-Value Store

Niv Dayan   Manos Athanassoulis   Stratos Idreos

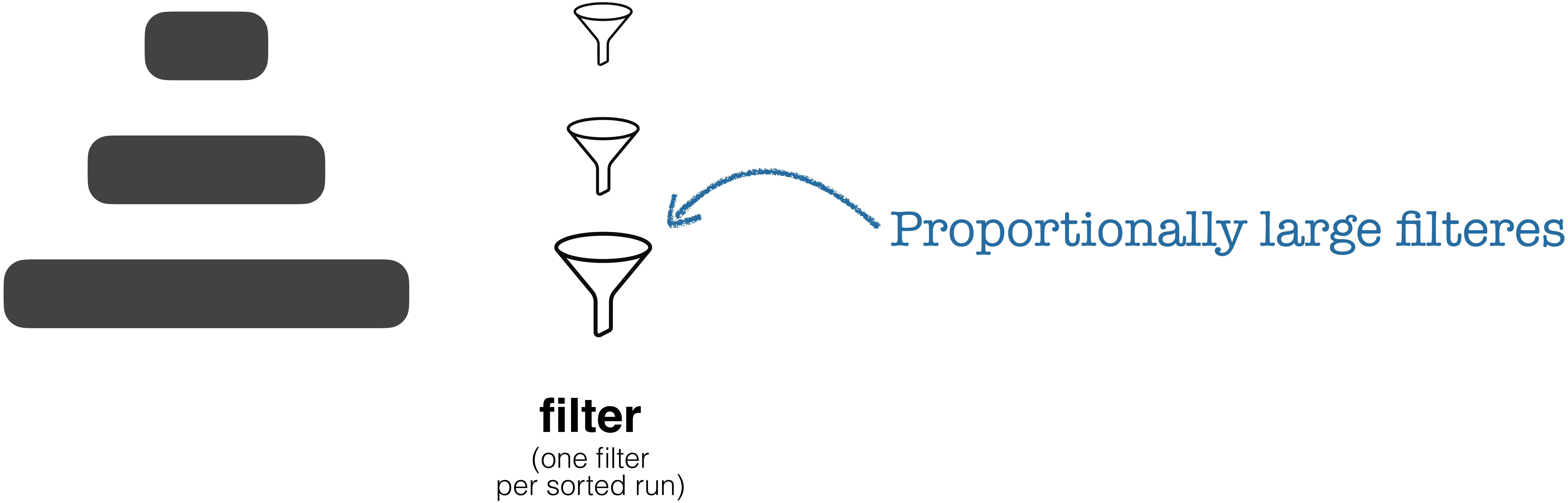
Harvard University

{dayan, manos, stratos}@seas.harvard.edu

Brandeis
UNIVERSITY

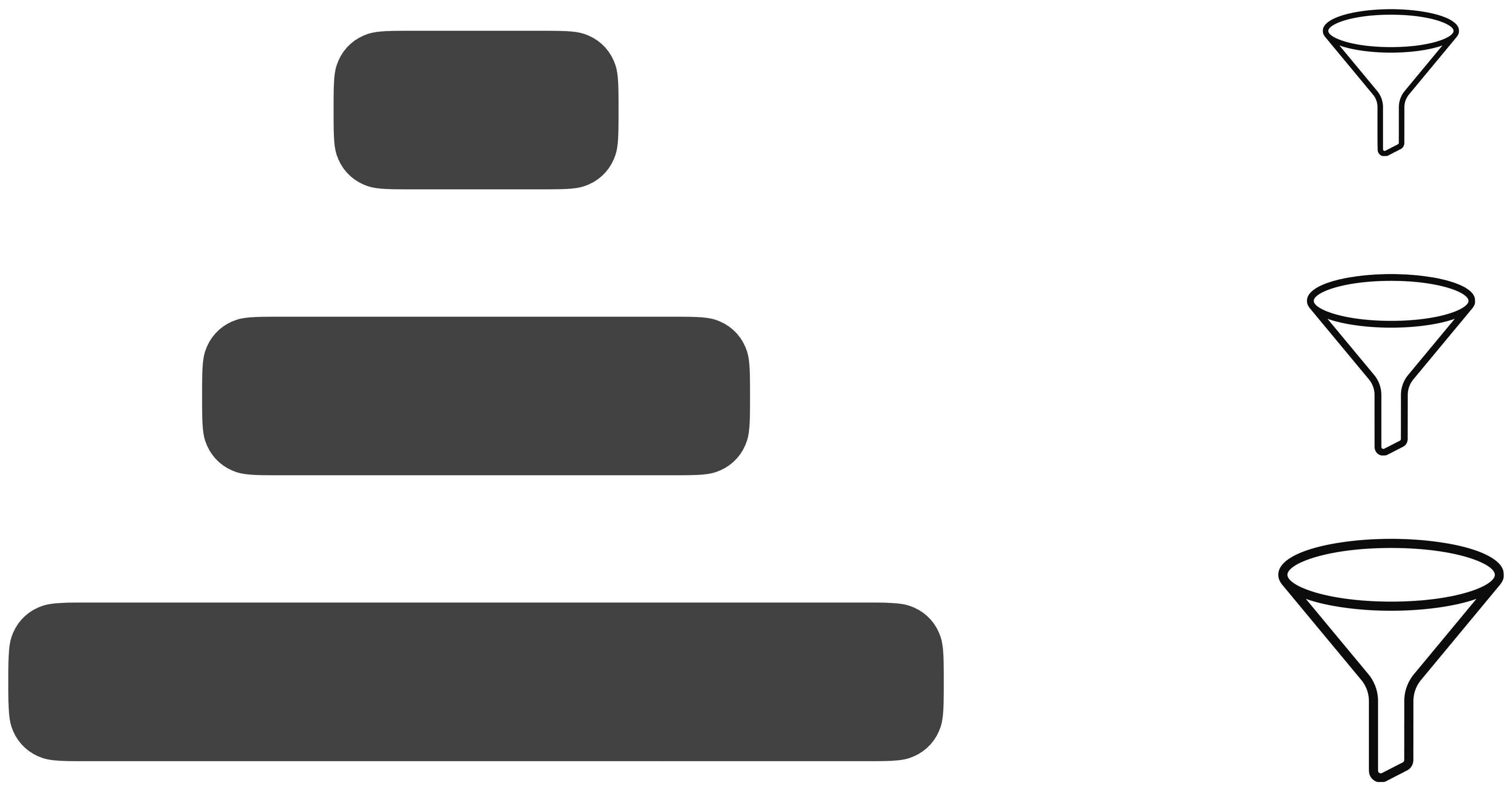
Monkey

Optimal **N**avigable **Key**-Value Store



Monkey

Optimal **N**avigable **Key**-Value Store



M

M

M

filter

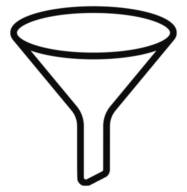
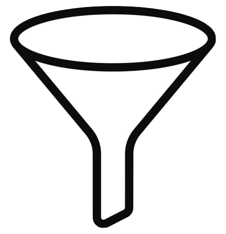
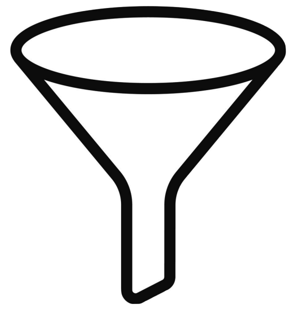
(one filter
per sorted run)

bits/entry

Brandeis
UNIVERSITY

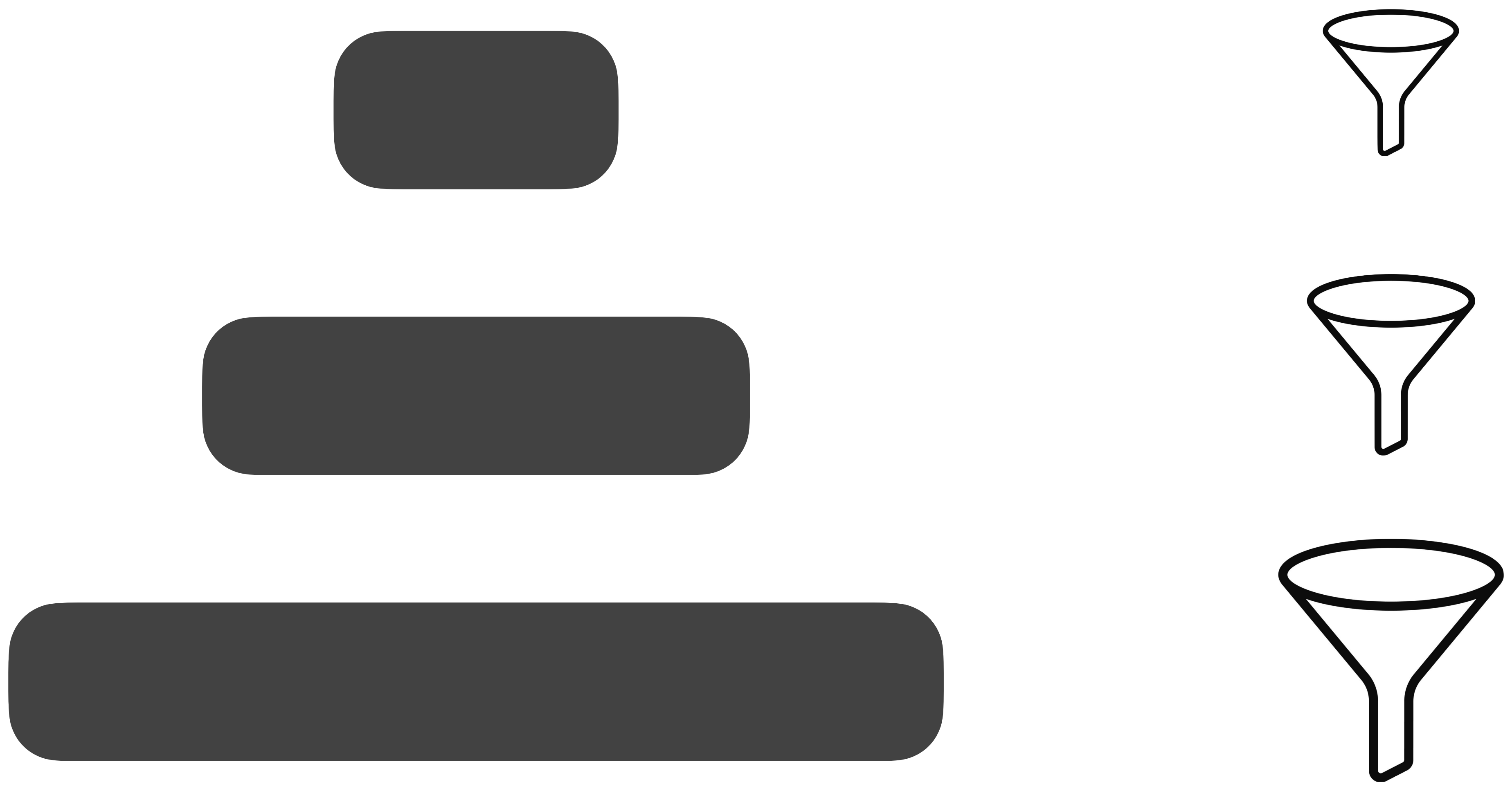
Monkey

Optimal **N**avigable **Key**-Value Store

filter (one filter per sorted run)	bits/entry	FPR
	M	$2^{-M \cdot \ln(2)}$
	M	$2^{-M \cdot \ln(2)}$
	M	$2^{-M \cdot \ln(2)}$

Monkey

Optimal **N**avigable **Key**-Value Store



filter (one filter per sorted run)	bits/entry	FPR
	M	2^{-M}
	M	2^{-M}
	M	2^{-M}

Monkey

Optimal **N**avigable **Key**-Value Store

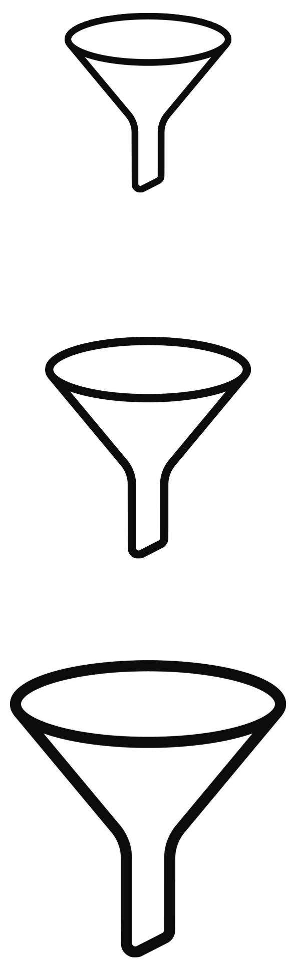
filter (one filter per sorted run)	bits/entry	FPR
filter	M	2^{-M}
filter	M	2^{-M}
filter	M	2^{-M}

$$= O(2^{-M} \cdot L)$$

Brandeis UNIVERSITY

Monkey

Optimal **N**avigable **Key**-Value Store



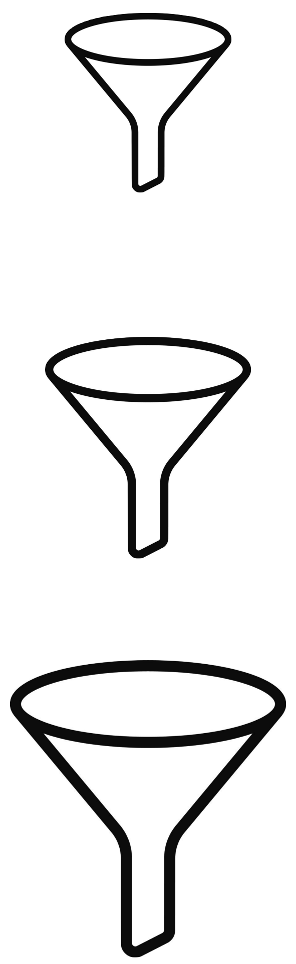
most memory

filter (one filter per sorted run)	bits/entry	FPR
	M	2^{-M}
	M	2^{-M}
most memory	M	2^{-M}

Brandeis UNIVERSITY

Monkey

Optimal **N**avigable **Key**-Value Store



filter		bits/entry	FPR
		M	2^{-M}
		M	2^{-M}
most memory	saves at most 1 access! →	M	2^{-M}

filter
(one filter per sorted run)

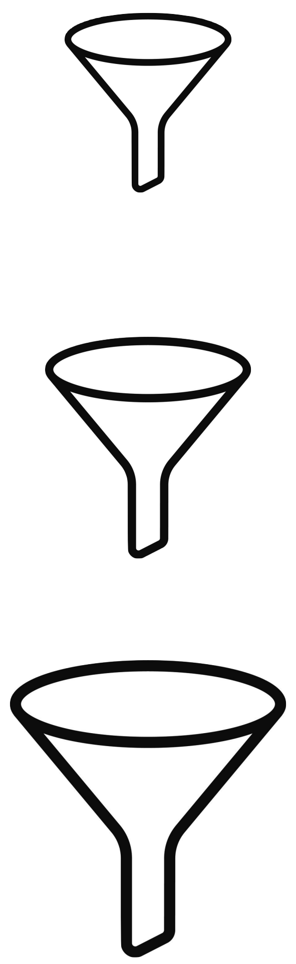
bits/entry

FPR

Brandeis UNIVERSITY

Monkey

Optimal **N**avigable **Key**-Value Store



filter
(one filter
per sorted run)

M+2

M+1

M-1

bits/entry

reallocate

Brandeis
UNIVERSITY

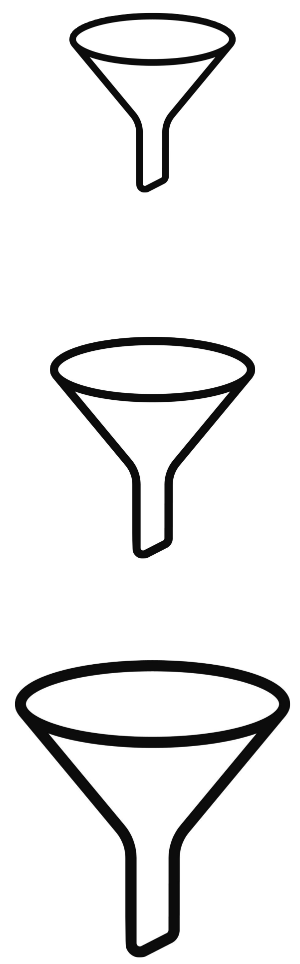
Monkey

Optimal **N**avigable **Key**-Value Store

filter (one filter per sorted run)	bits/entry	FPR
	$M+2$	$2^{-(M+2)}$
	$M+1$	$2^{-(M+1)}$
	$M-1$	$2^{-(M-1)}$

Monkey

Optimal **N**avigable **Key**-Value Store



$\propto 2^{-M} / T^2$

$\propto 2^{-M} / T^1$

$\propto 2^{-M} / T^0$

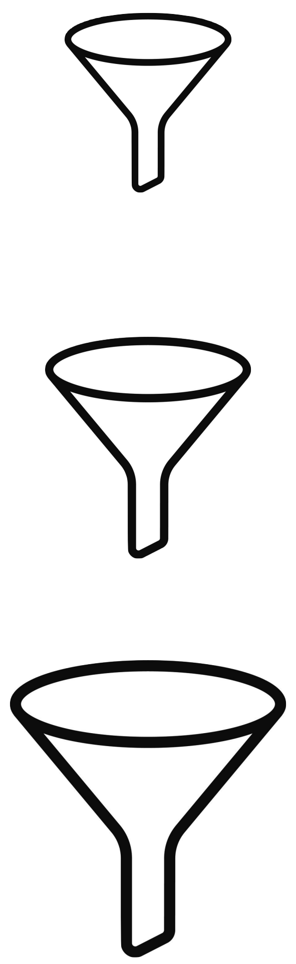
$= O(2^{-M})$

filter
(one filter
per sorted run)

FPR

Monkey

Optimal **N**avigable **Key**-Value Store



$\propto 2^{-M} / T^2$

$\propto 2^{-M} / T^1$

$\propto 2^{-M} / T^0$

$= O(2^{-M}) < O(2^{-M} \cdot L)$

filter

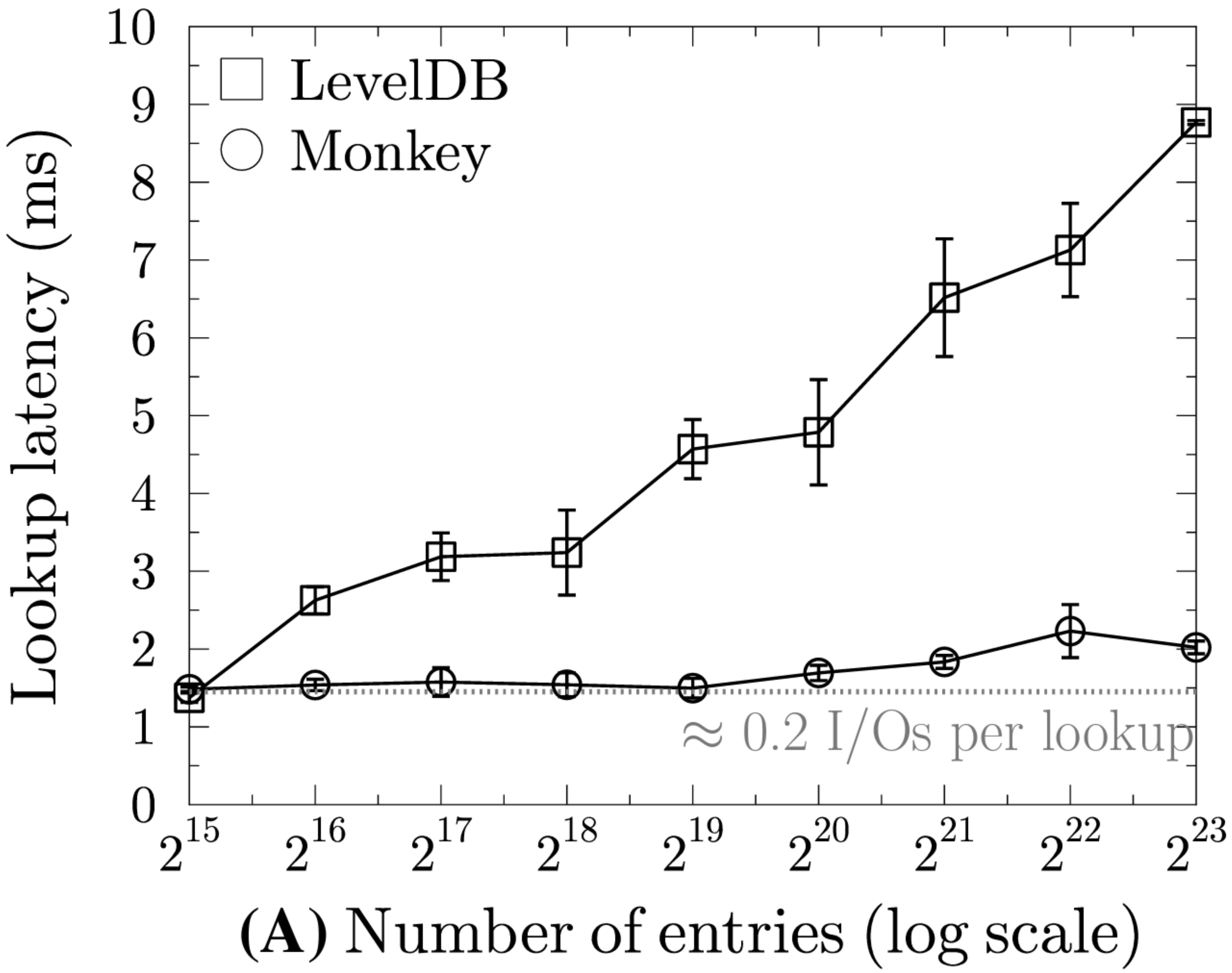
(one filter
per sorted run)

FPR

Brandeis UNIVERSITY

Monkey

Optimal **N**avigable **Key**-Value Store



Brandeis UNIVERSITY

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)

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)

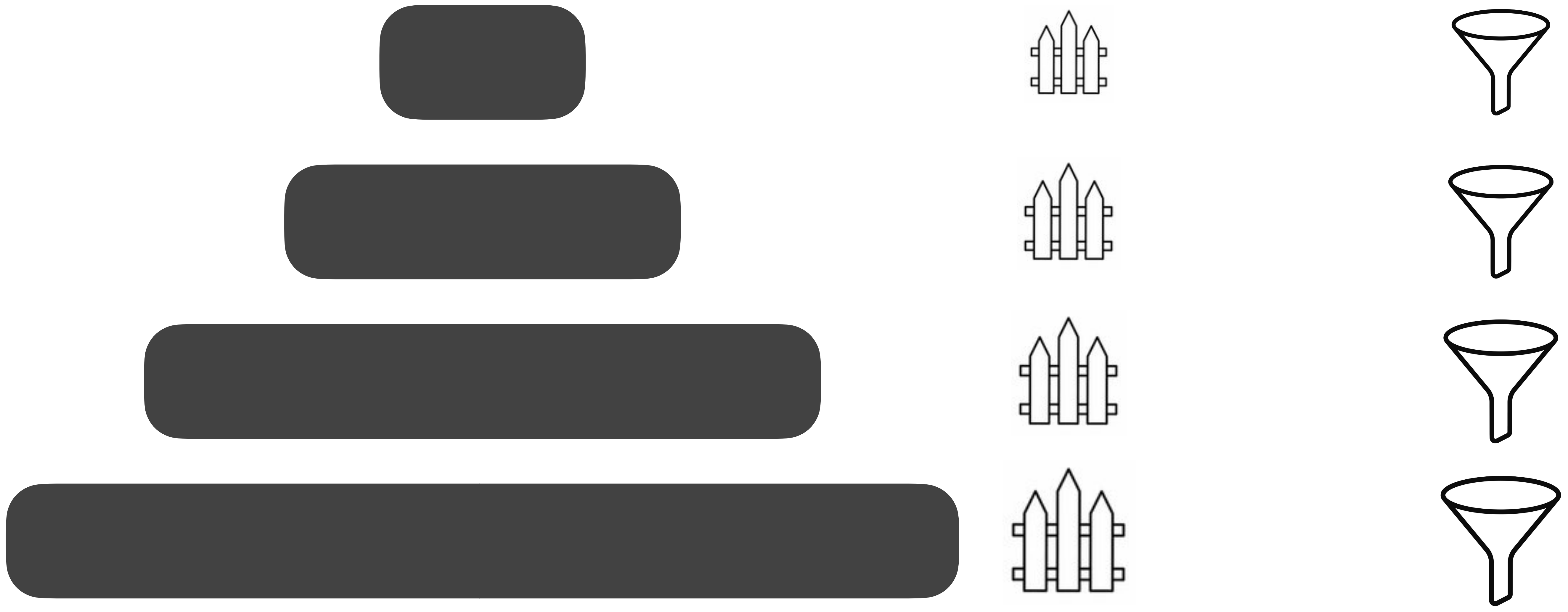
* **Monkey** shaves off the "L" factor from the cost

Range lookup cost

Looking for keys in a range

Range lookup cost

Looking for keys in a range



leveled LSM-tree

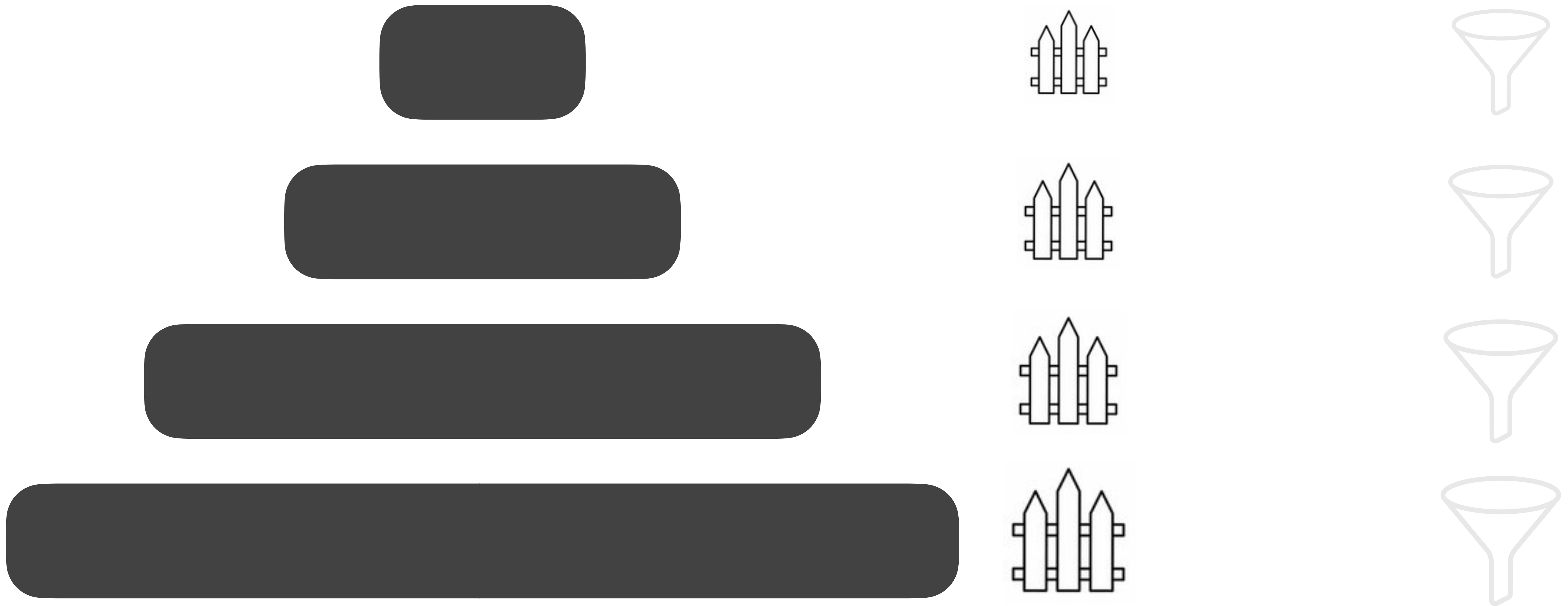
fence pointers

(page-wise zone map)

filter

Range lookup cost

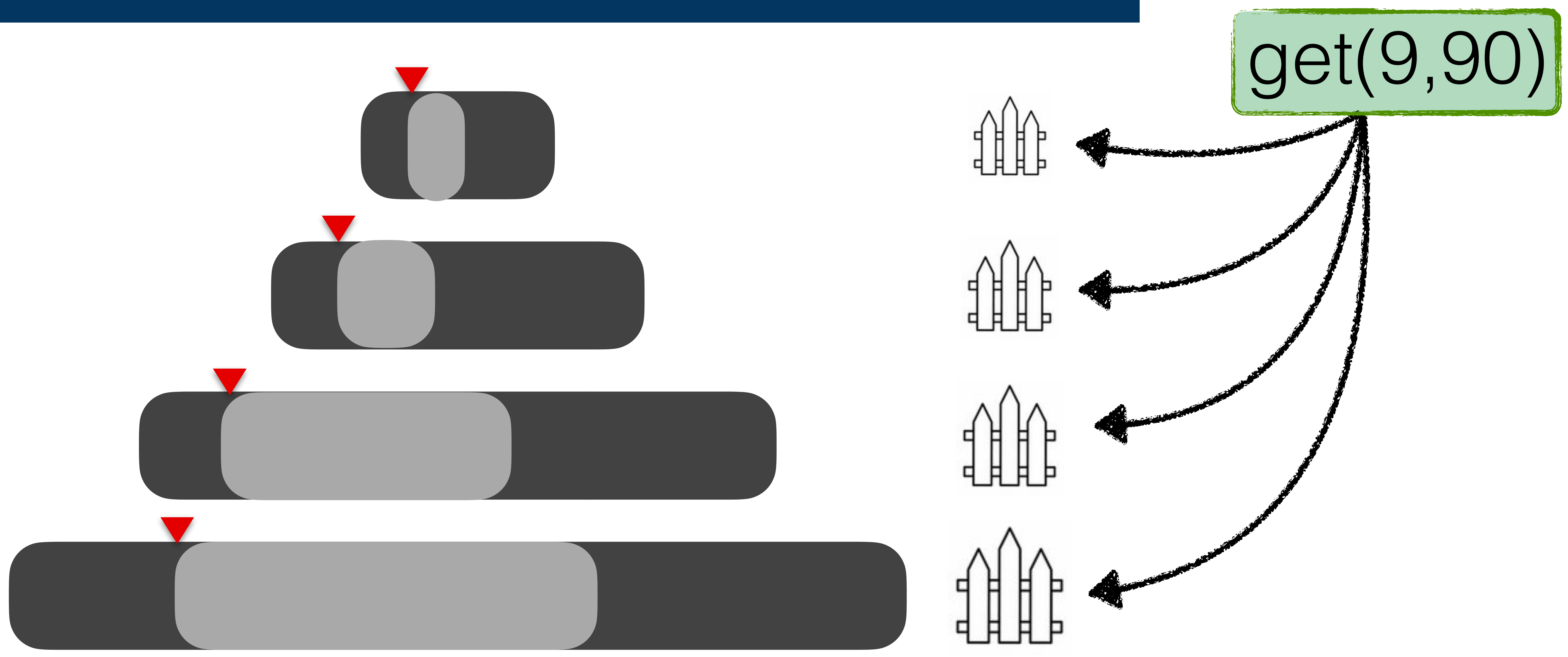
Looking for keys in a range



Range lookup cost

Looking for keys in a range

s = selectivity of the range query



leveled LSM-tree

fence pointers
(page-wise zone map)

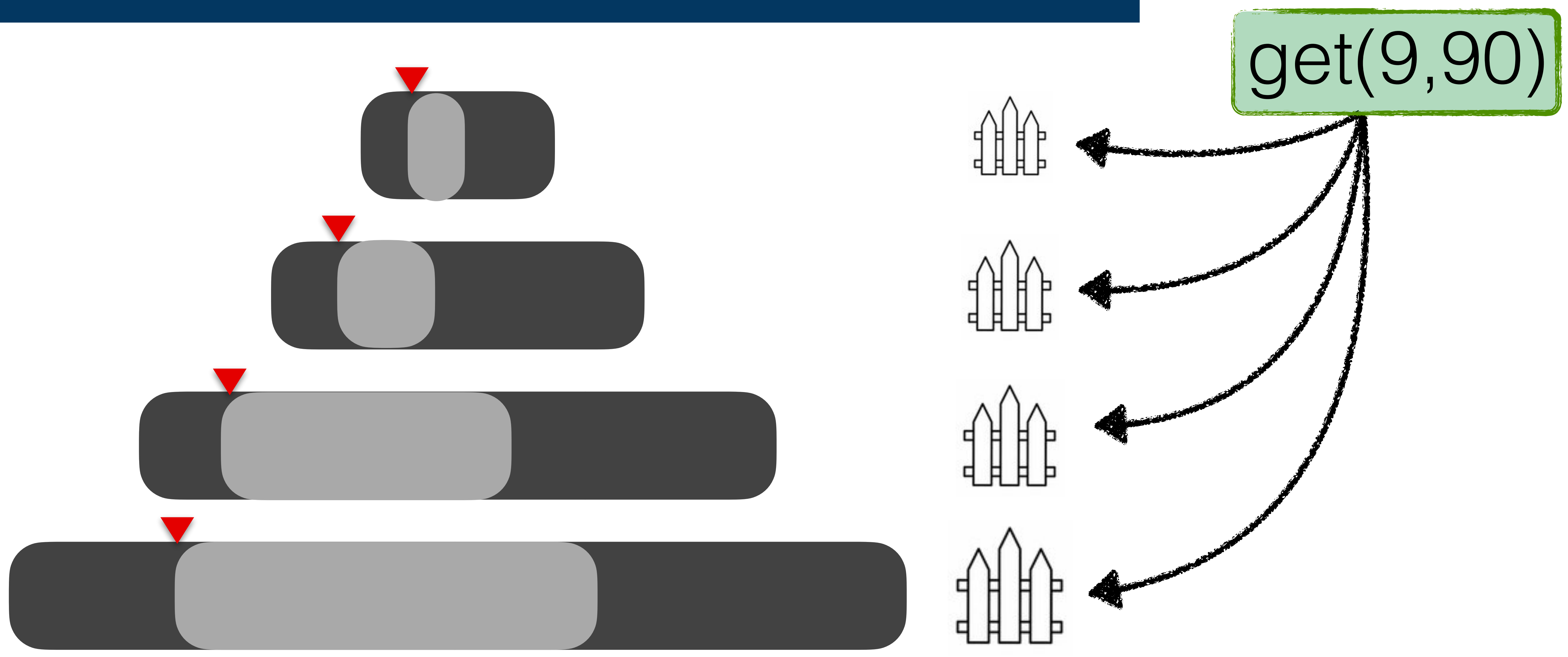
Thought Experiment 7

Cost of a **range query**?

Range lookup cost

Looking for keys in a range

s = selectivity of the range query



leveled LSM-tree

fence pointers
(page-wise zone map)

Thought Experiment 7

Cost of a **range query**?

total entries in tree = **N**

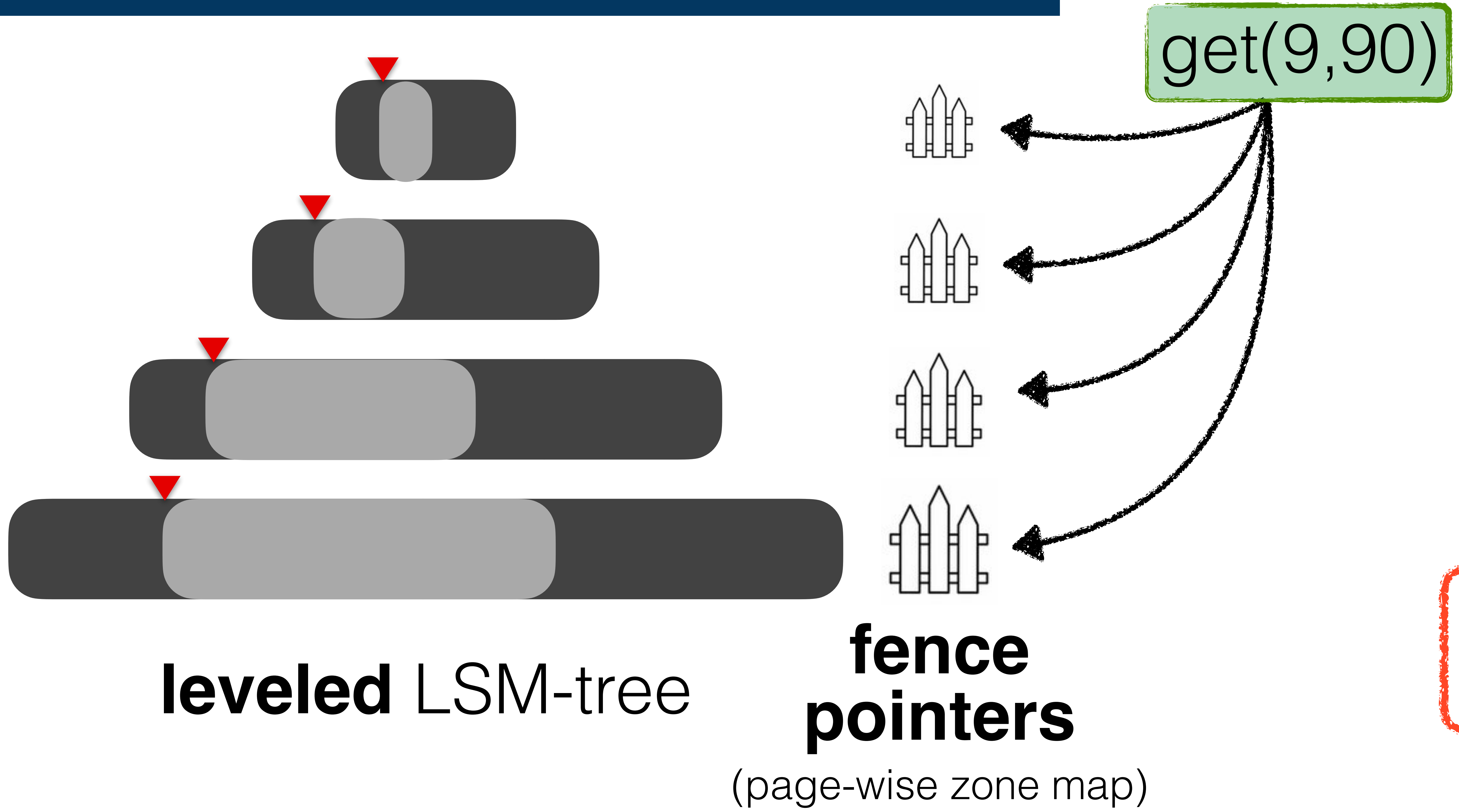
#entries per page = **B**

#pages in tree = N/B

Range lookup cost

Looking for keys in a range

s = selectivity of the range query



Thought Experiment 7

Cost of a **range query**?

total entries in tree = **N**

#entries per page = **B**

#pages in tree = N/B

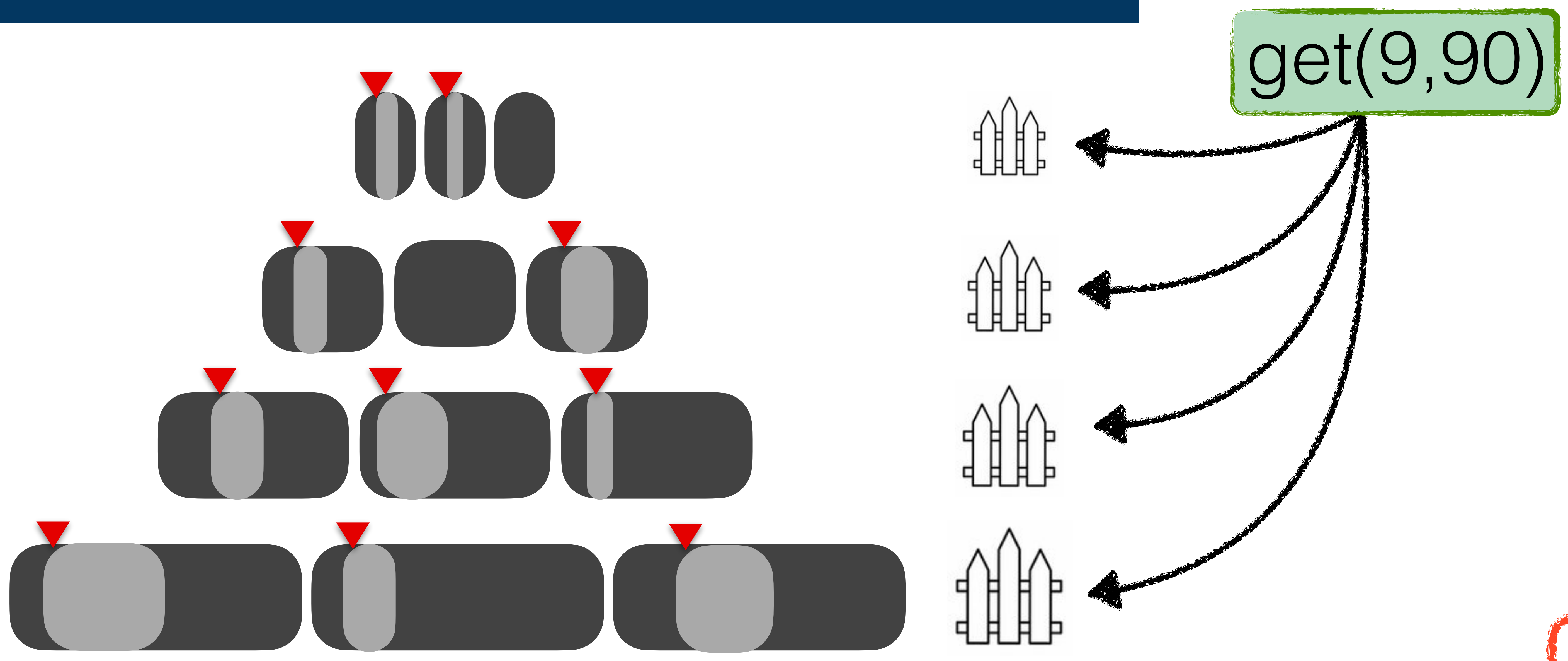
Cost of range lookup = $s \cdot N/B$

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

Range lookup cost

Looking for keys in a range

s = selectivity of the range query



tiered LSM-tree

fence pointers

(page-wise zone map)

same cost for
leveled & tiered LSM

Thought Experiment 7

Cost of a **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**?

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

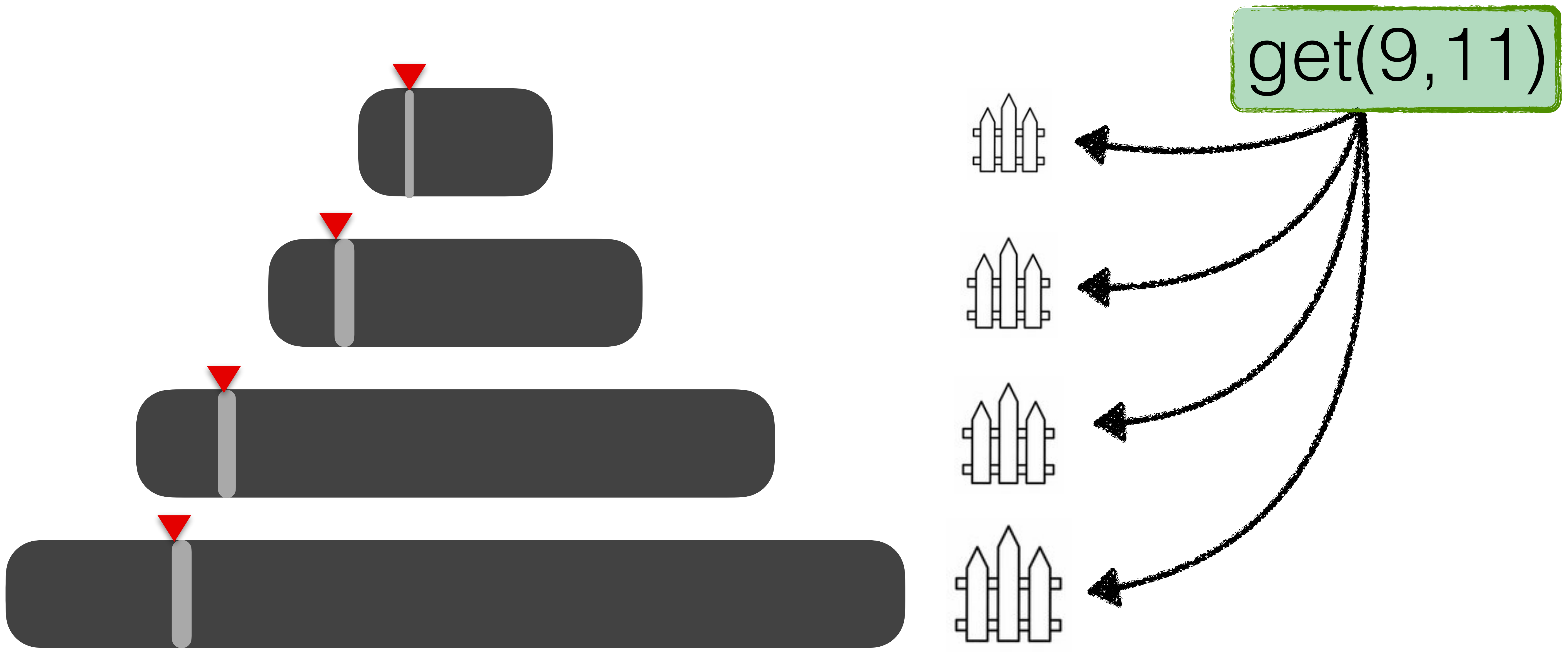
data structure	ingestion cost	point lookup cost*	range lookup cost*
Leveled LSM-tree	$O(L \cdot T / B)$	$O(1+\phi \cdot L)$*	$O(s \cdot N / B)$
Tiered LSM-tree	$O(L / B)$	$O(1+\phi \cdot L \cdot T)$*	$O(s \cdot N / B)$
B+-tree			
Sorted array			
Log			

* 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

Range lookup cost

Looking for keys in a range



leveled LSM-tree

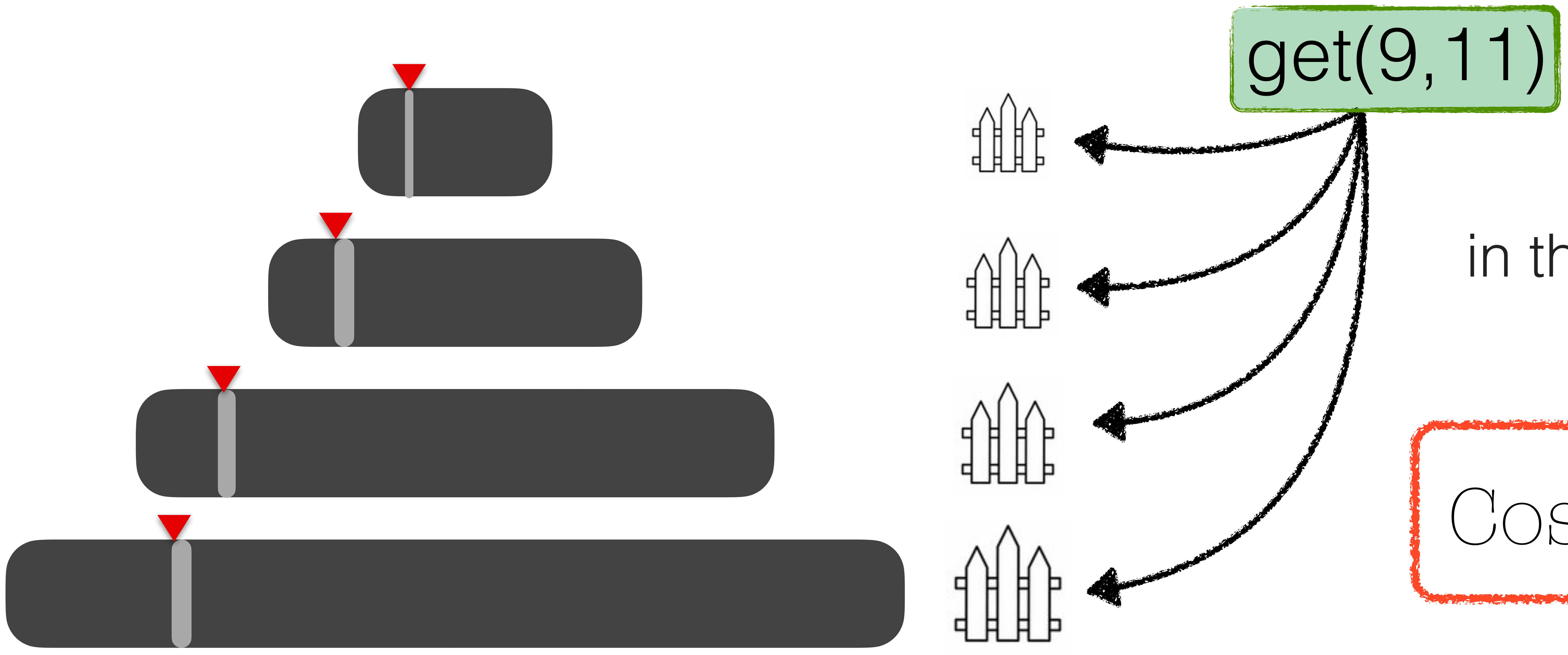
fence pointers

(page-wise zone map)

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

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**?

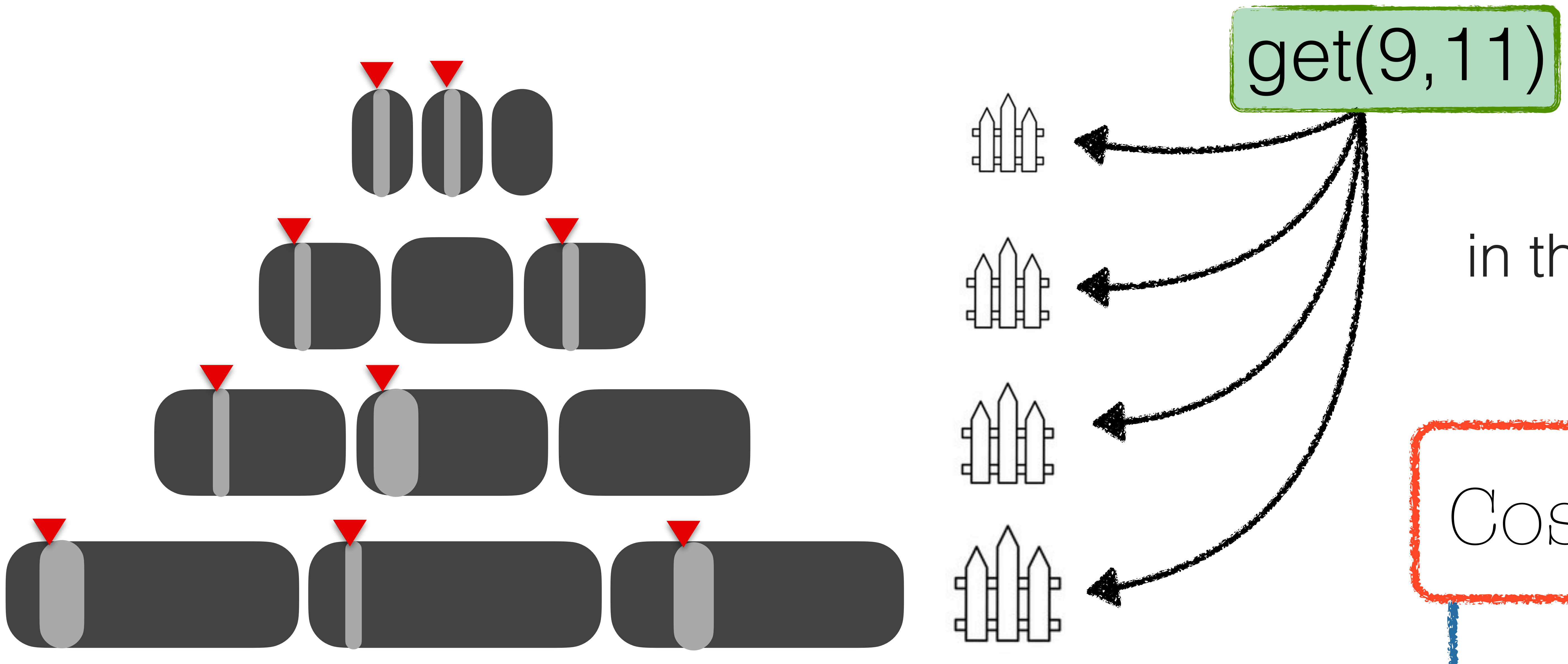
in the **worst case**,

need to read **2 pages per sorted run**

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

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

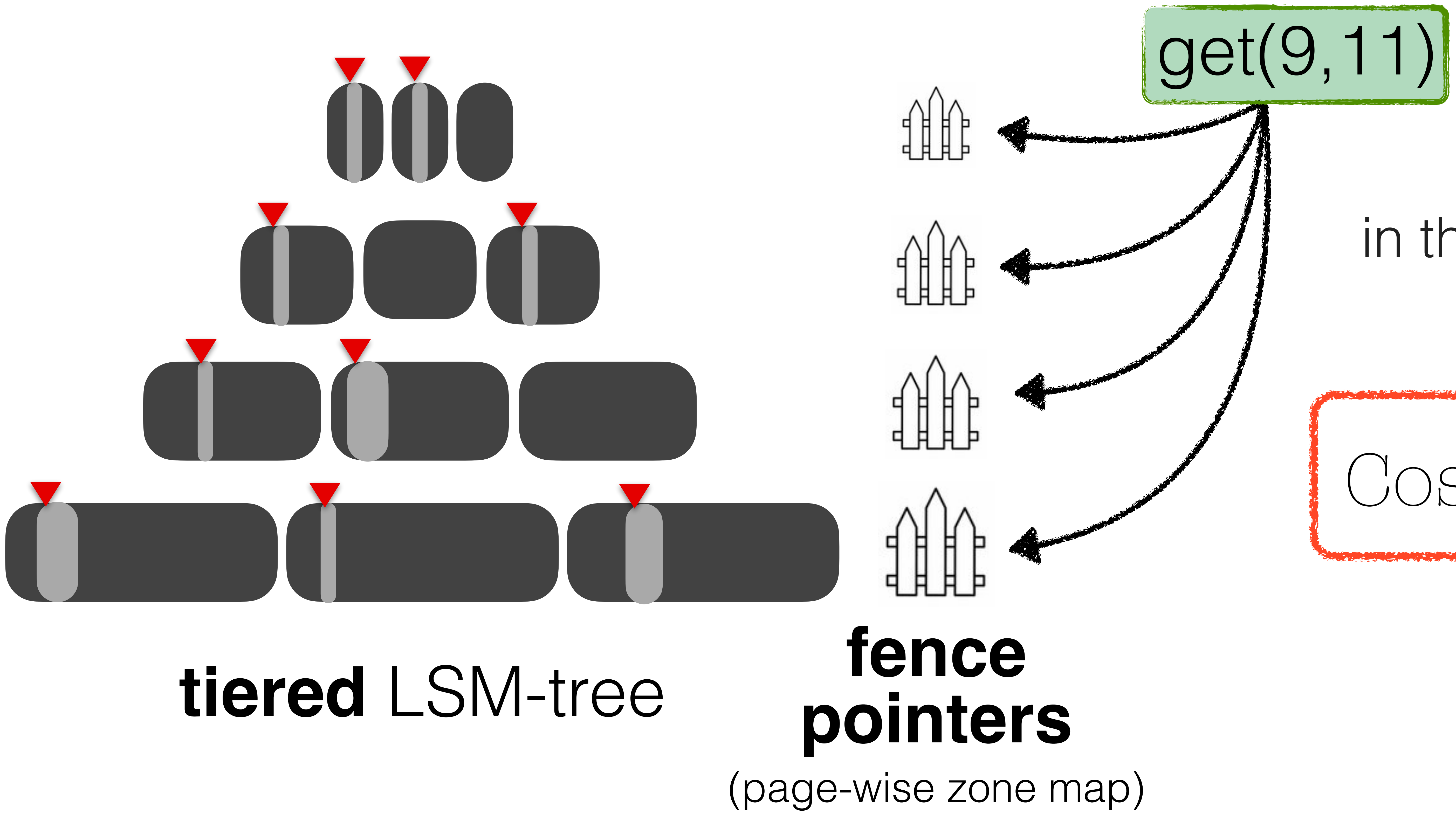
tiering

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

Brandeis UNIVERSITY

Range lookup cost

Looking for keys in a range



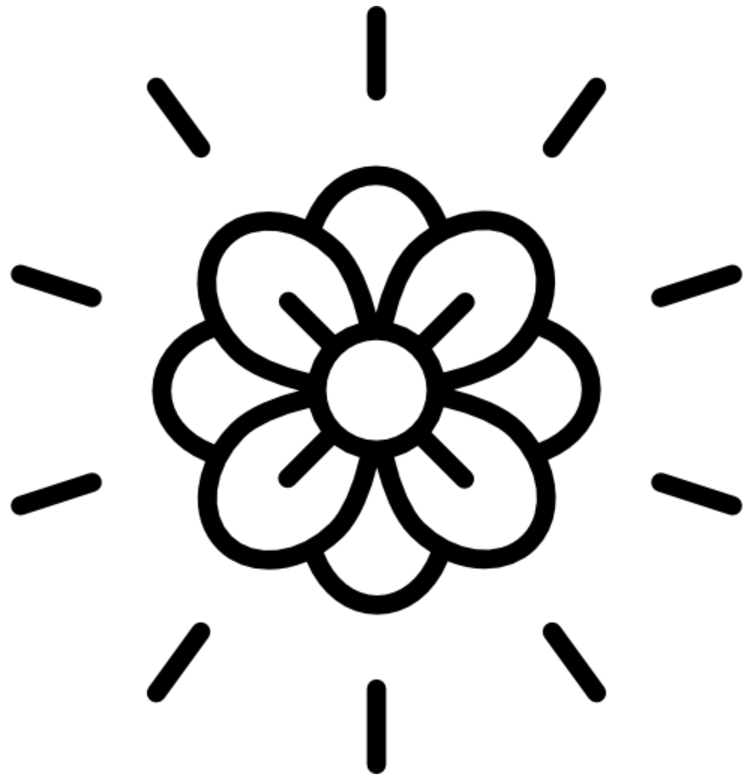
What if the **range query** has a **very low selectivity**?

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

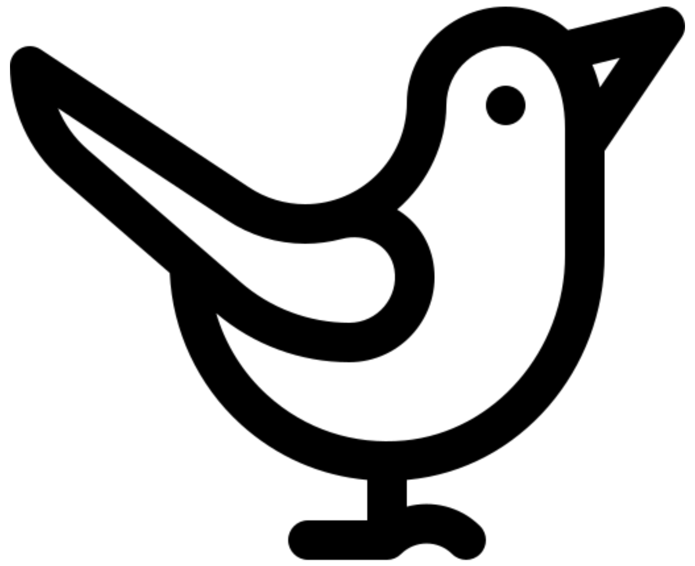
Cost of short range lookup = $2 \cdot L$

Filters?

Point and range filters



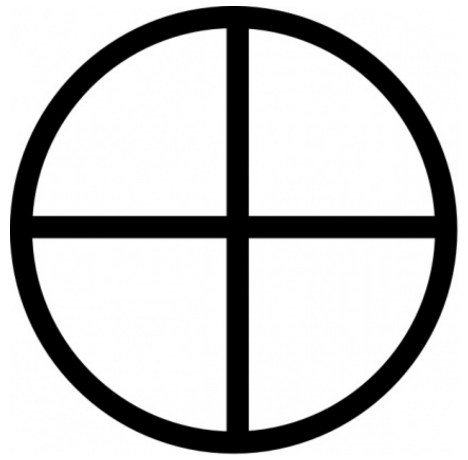
Bloom



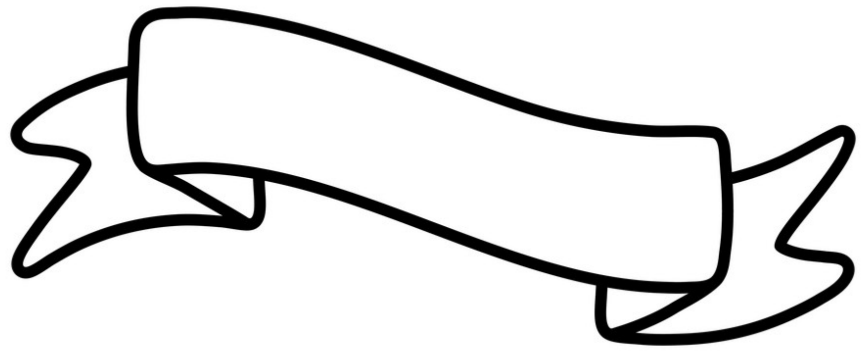
Cuckoo

SuRF

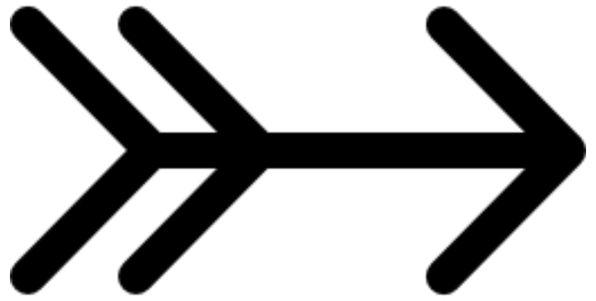
Rosetta



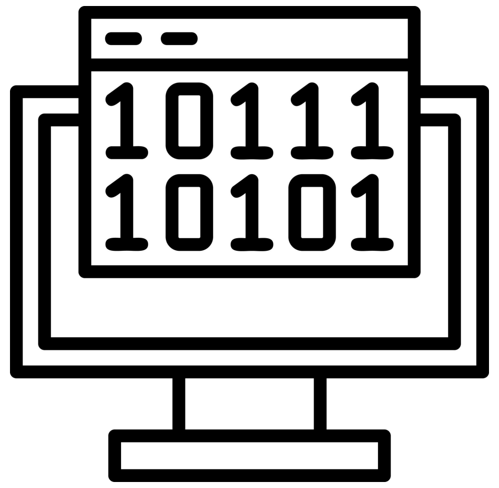
XOR



Ribbon



prefix



REncoder

Next time in COSI 167A

More on LSMs

Cost analysis

Class Project discussion

COSI 167A

Advanced Data Systems

Class 6

Performance Analysis of LSM-Trees

Prof. Subhadeep Sarkar

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

Brandeis University