BOSTON UNIVERSITY

CS 561: Data Systems Architectures

Systems & Research Project

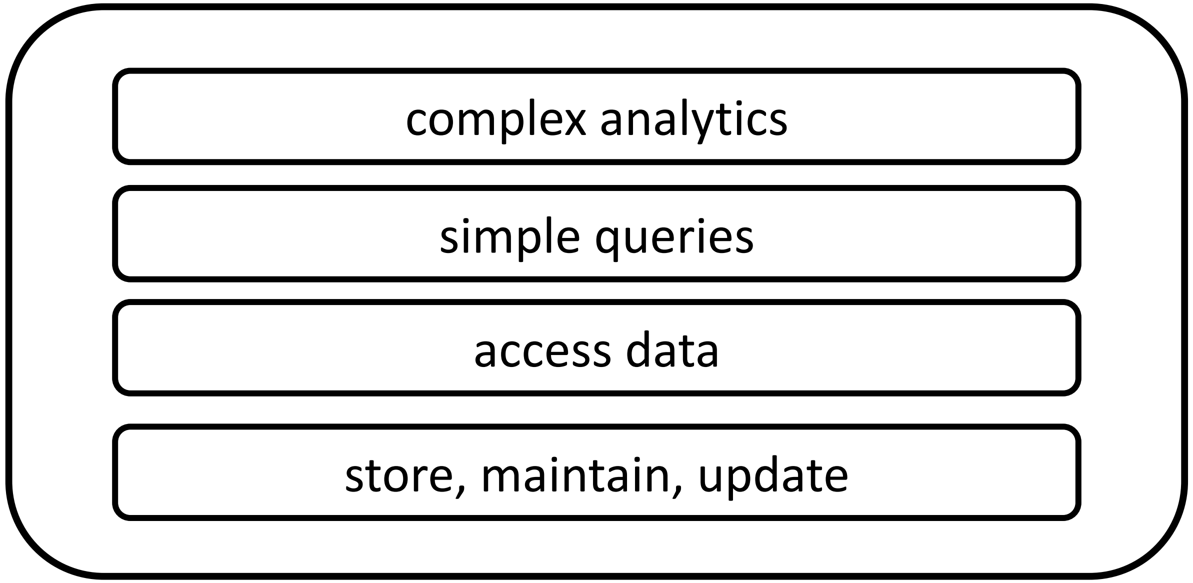
Prof. Manos Athanassoulis

https://midas.bu.edu/classes/CS591A1

data systems

>$200B by 2020, growing at 11.7% every year

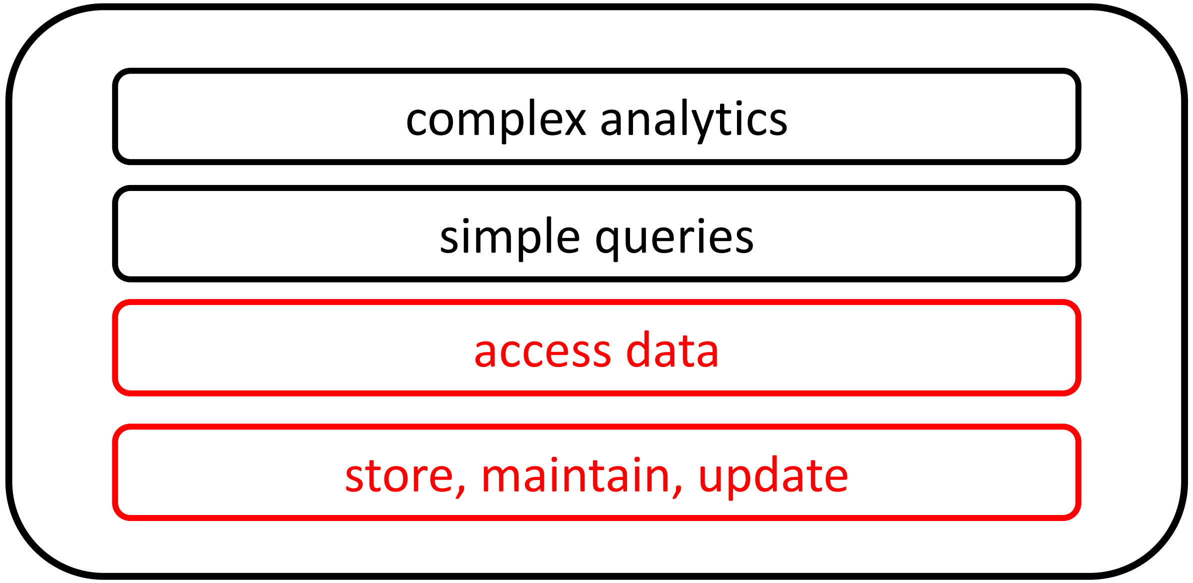
[The Forbes, 2016]



data systems

>$200B by 2020, growing at 11.7% every year

[The Forbes, 2016]



complex analytics

simple queries

access data

store, maintain, update

access methods*

*__algorithms__ and __data structures__ for organizing and accessing data

data systems core: storage engines

main decisions

how to ***store*** data?

how to ***access*** data?

how to ***update*** data?

let’s simplify: **key-value** storage engines

collection of keys-value pairs

query on the key, return both key and value

remember

amazon DynamoDB

APACHE HBASE

Google BigTable

Facebook RocksDB

AsterixDB™

Couchbase

Cassandra

mongoDB

state-of-the-art design

how general is a key value store?

can we store relational data?

yes!   {<primary_key>,<rest_of_the_row>}

example: { **student_id**, { **name**, **login**, **yob**, **gpa** } }

what is the caveat?

how to index these attributes?

other problems?

index: { **name**, { **student_id** } }

index: { **yob**, { **student_id$_1$, student_id$_2$, …** } }

how general is a key value store?

can we store relational data?

yes! {<primary_key>,<rest_of_the_row>}

how to efficiently code if we do not know the structure of the "*value*"

index: { **yob**, { **student_id$_1$, student_id$_2$, …** } }

how to use a key-value store?

basic interface

put(k,v)

{v} = get(k)   {v_1, v_2, …} = get(k)

{v_1, v_2, …} = get_range(k_{min}, k_{max})   {v_1, v_2, …} = full_scan()

c = count(k_{min}, k_{max})

more

deletes: delete(k)

updates: update(k,v) is it different than put?

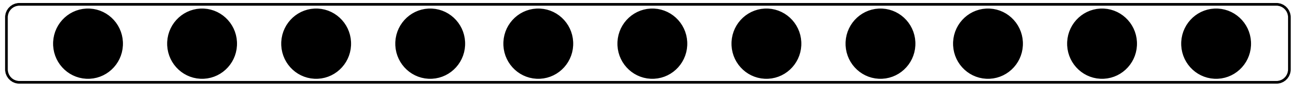
get set: {v_1, v_2, …} = get_set(k_1, k_2, …)

BOSTON UNIVERSITY

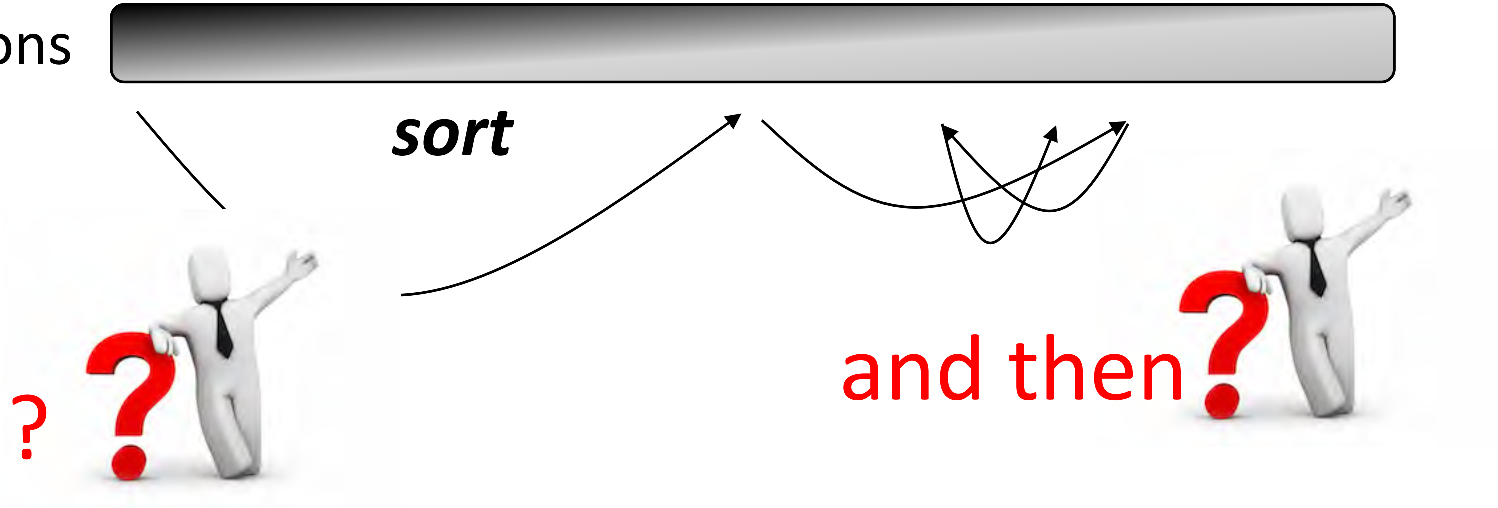
how to build a key-value store?

if we have only ***put*** operations

append



if we mostly have ***get*** operations



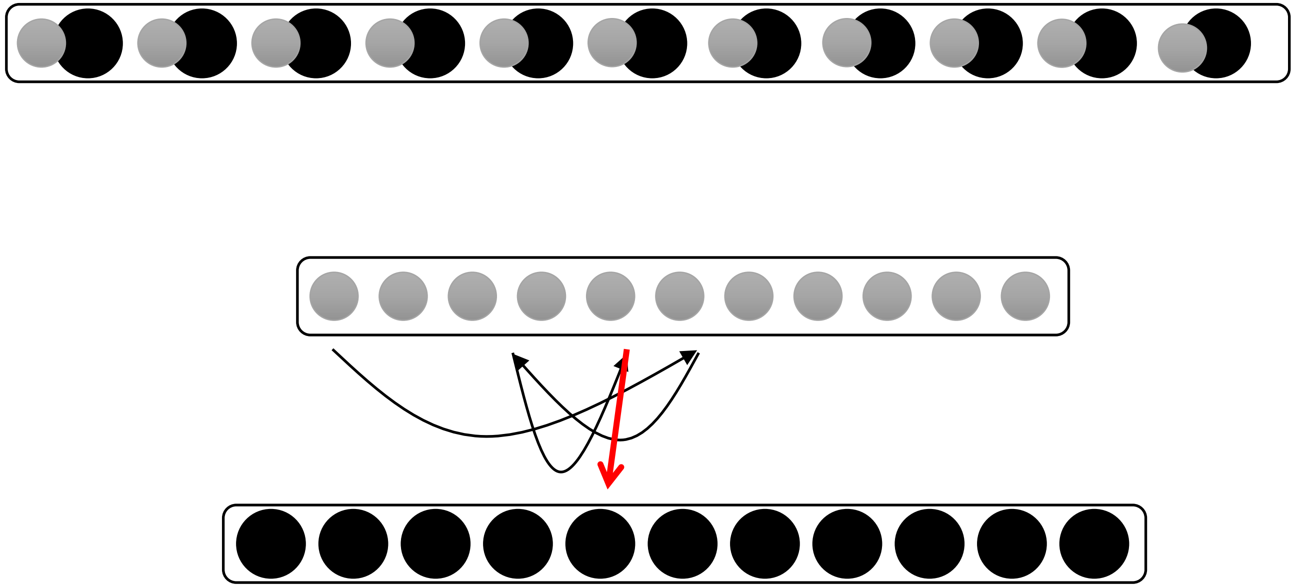
sort

what about full scan?

and then

range queries?

can we separate keys and values?



at what price?

locality? code?

read queries

(point or range)

inserts

(or updates)

sort data

simply append

amortize sorting cost

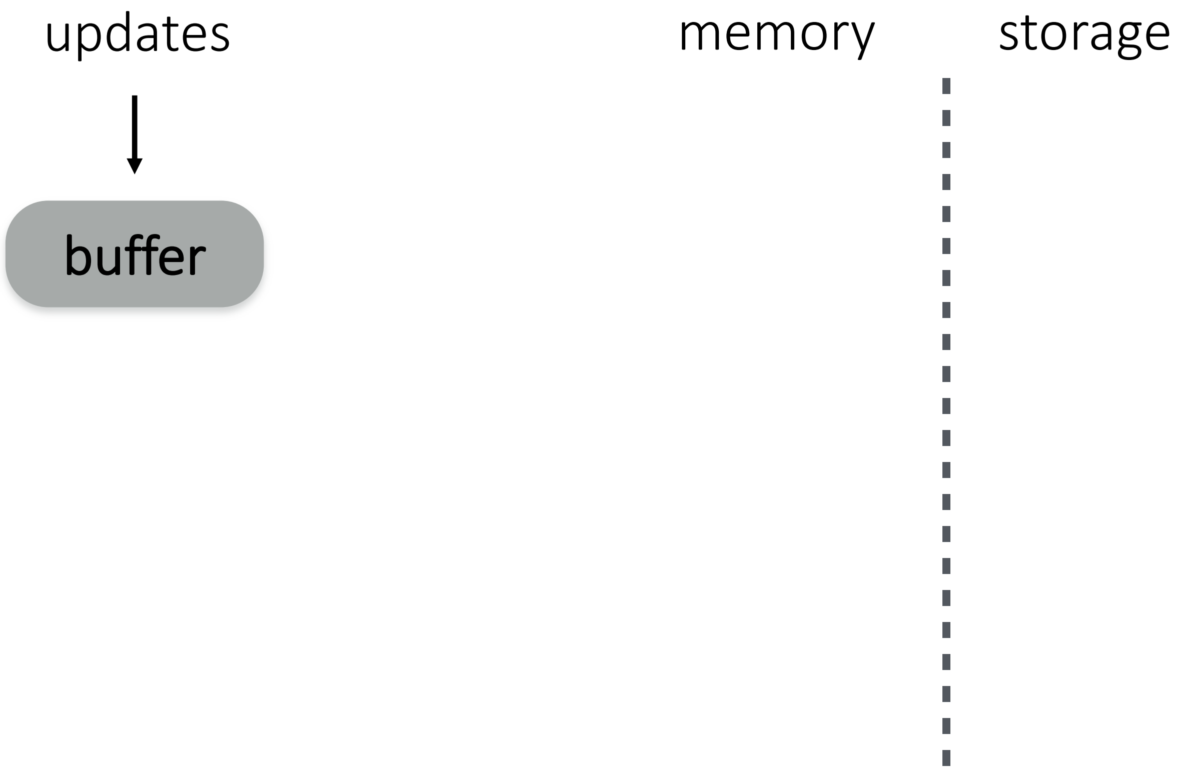
avoid resorting after every update

how to bridge?

LSM-tree
Key-Value Stores

What are they really?

BOSTON UNIVERSITY

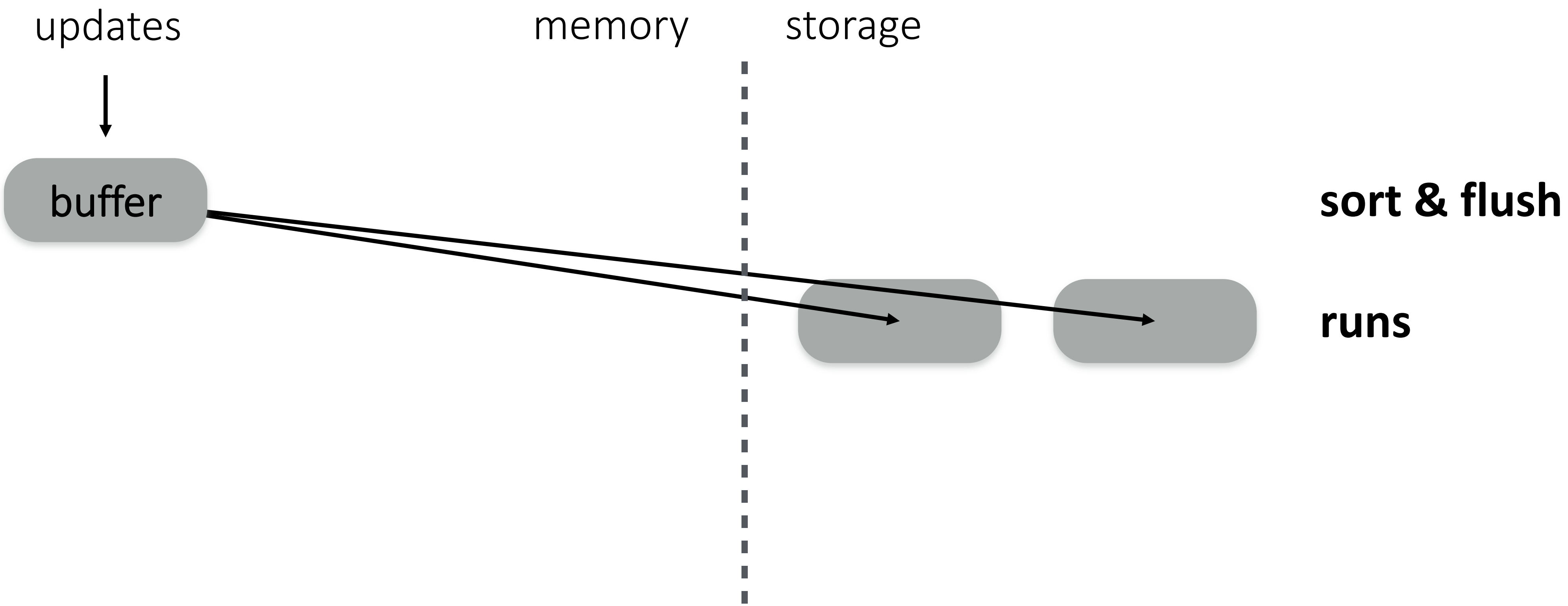


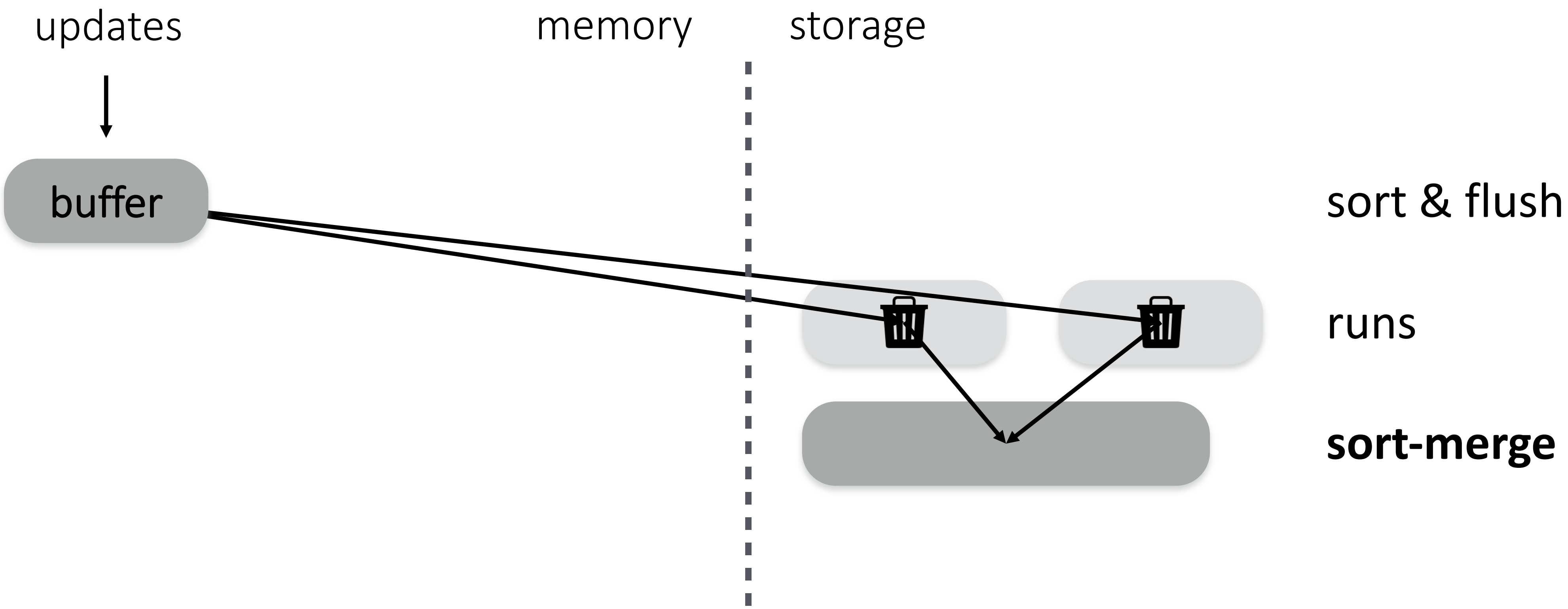
updates

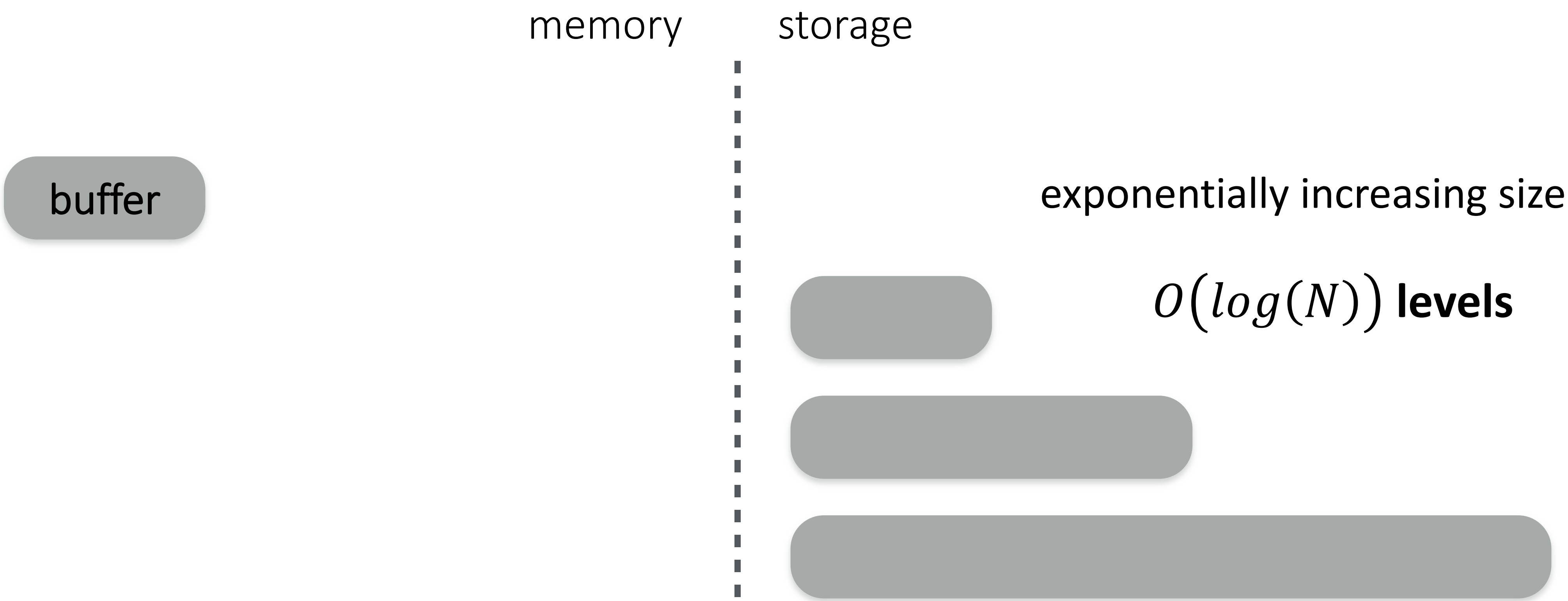
buffer

memory

storage


updates
memory
storage
buffer
sort & flush
runs


updates
memory
storage
buffer
sort & flush
runs
sort-merge



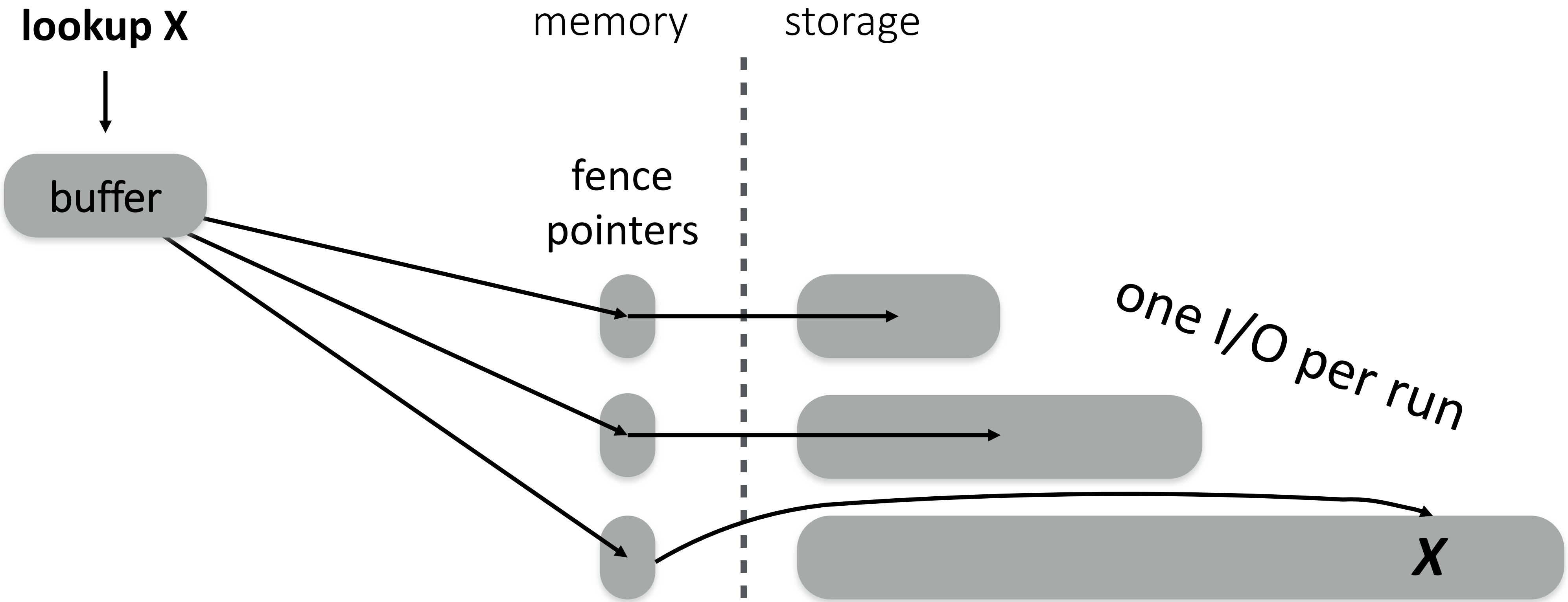
memory

storage

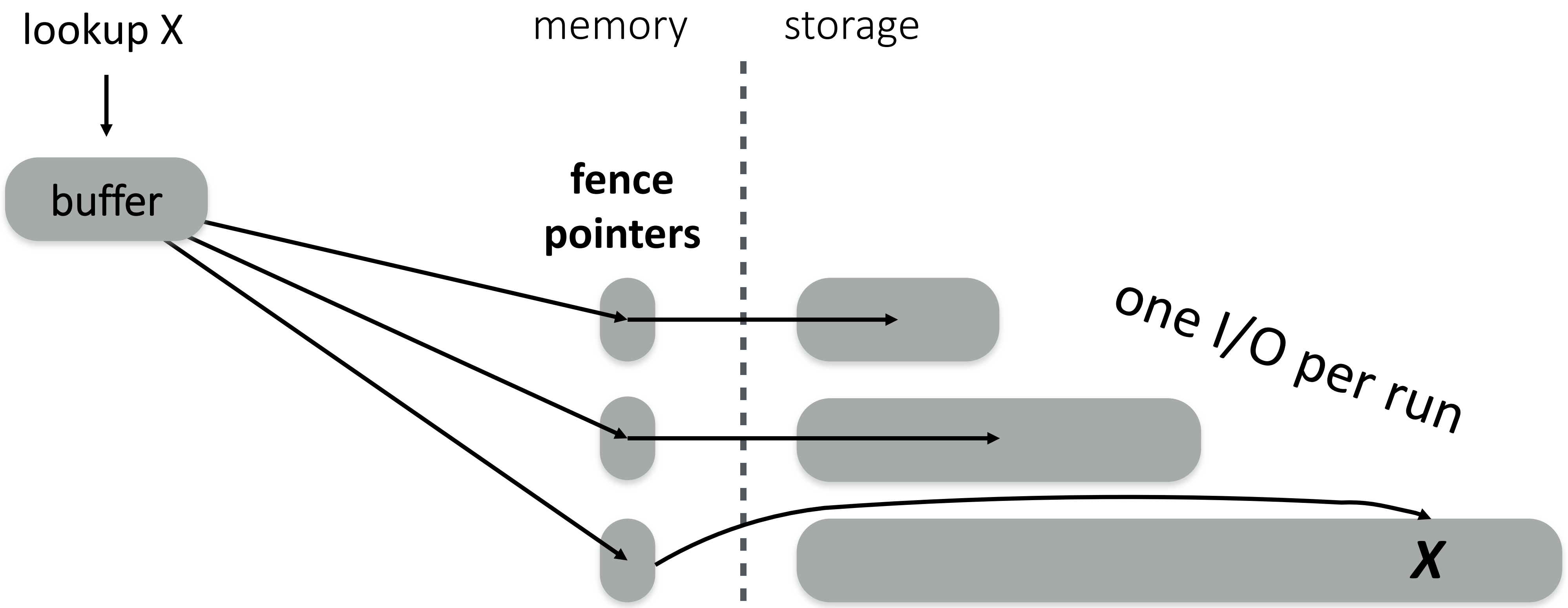
buffer

exponentially increasing sizes

$O\left(log(N)\right)$ **levels**



lookup X
memory
storage
buffer
fence
pointers
one I/O per run
X



lookup X

memory

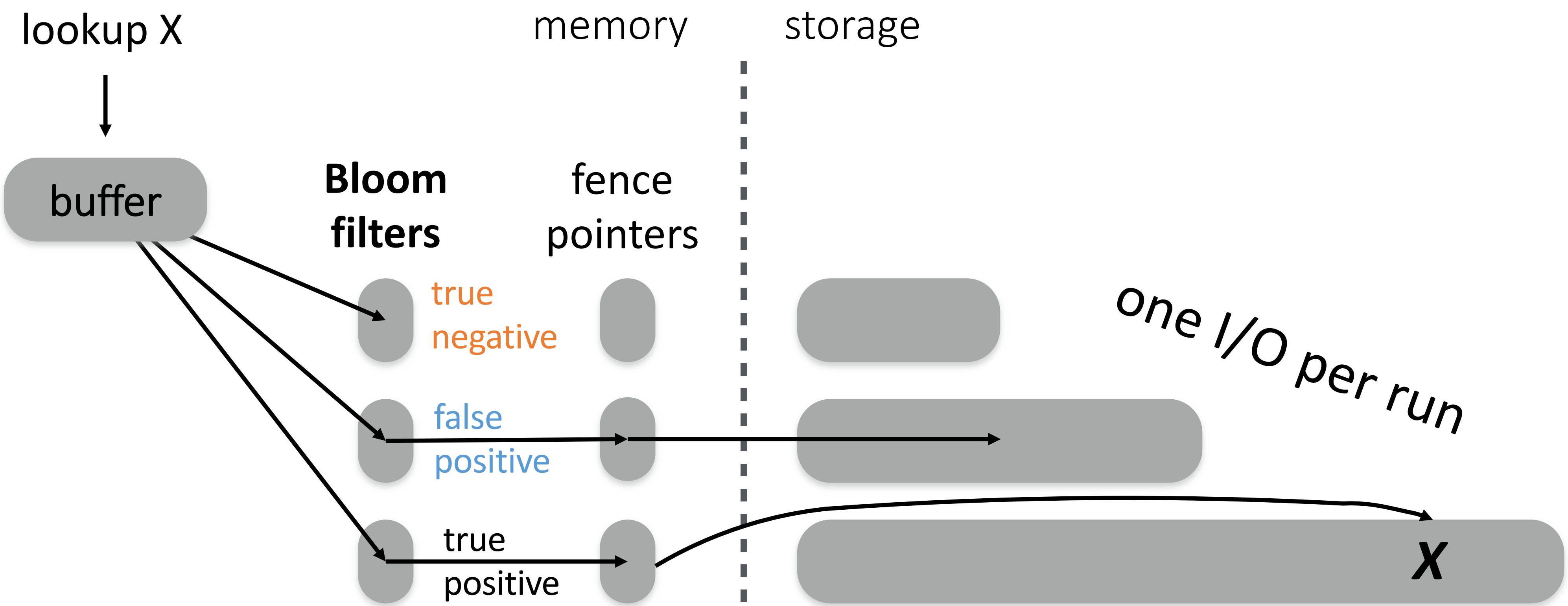
storage

buffer

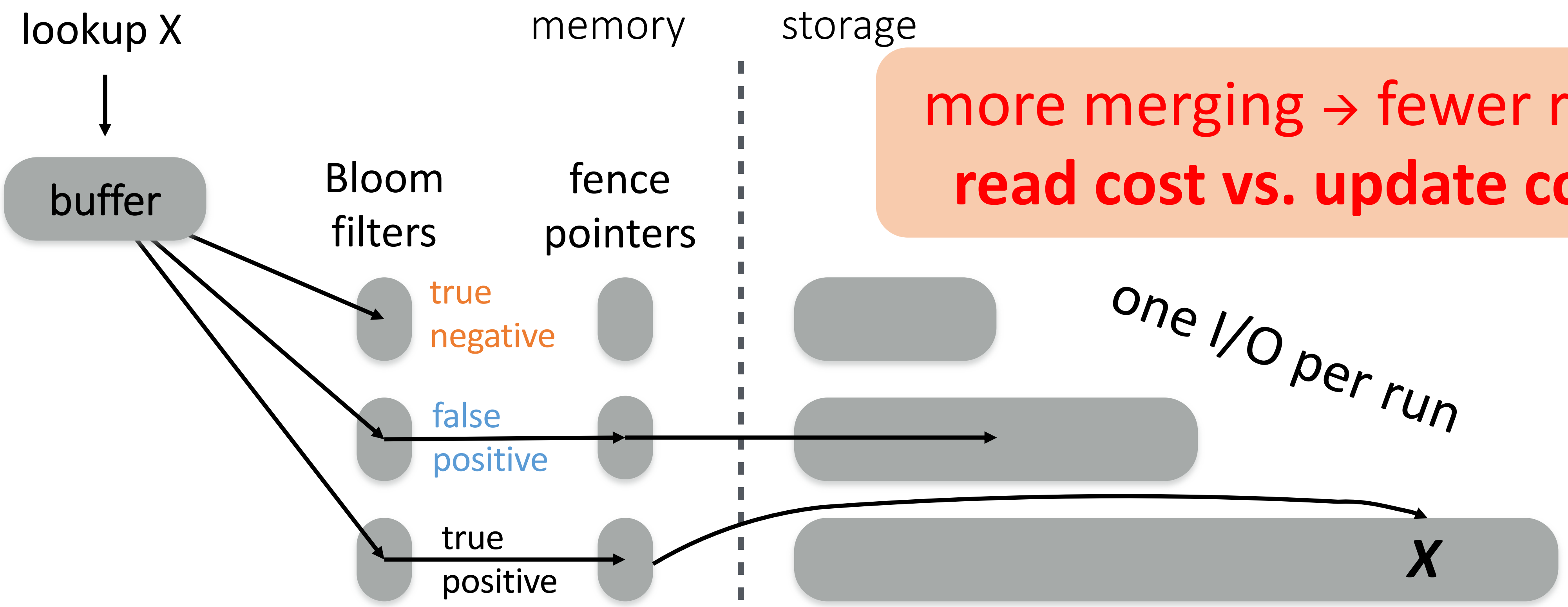
fence pointers

one I/O per run

X


lookup X
memory
storage
buffer
Bloom filters
fence pointers
true negative
false positive
true positive
one I/O per run
X

performance & cost trade-offs



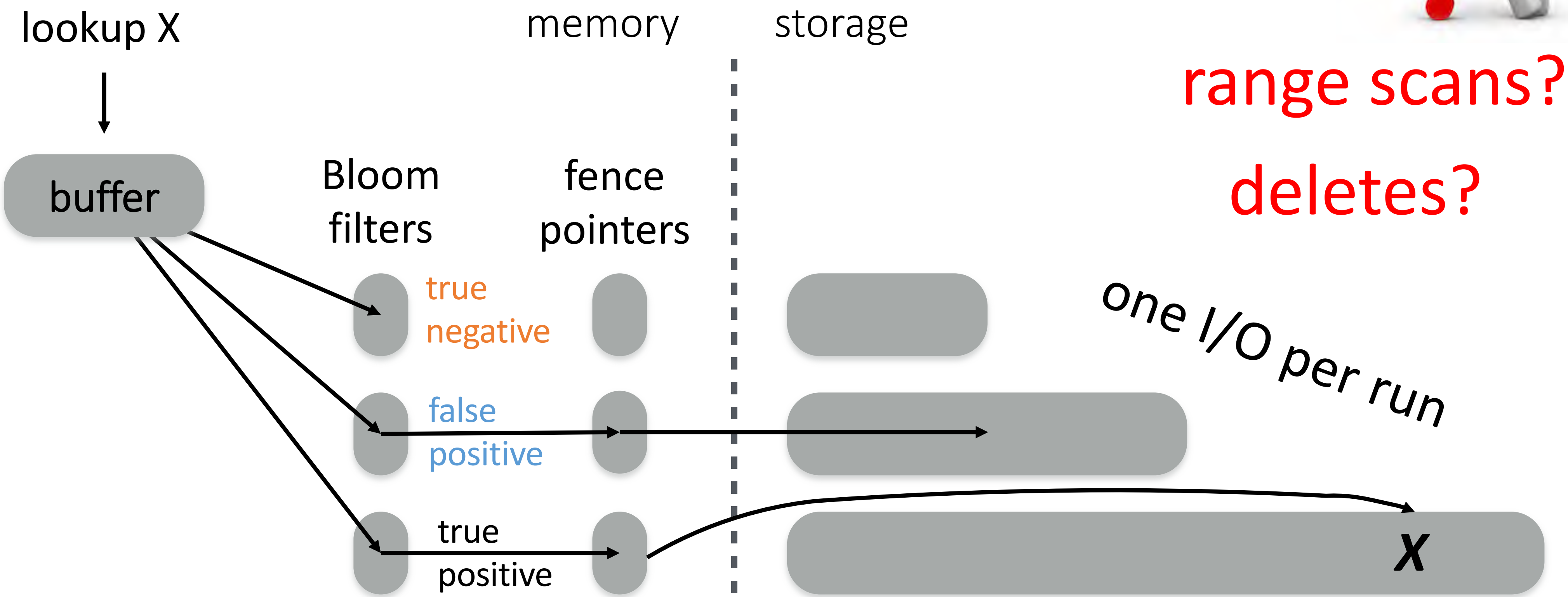
more merging → fewer runs

read cost vs. update cost

bigger filters → fewer false positives

memory space vs. read cost

other operations



lookup X

memory

storage

buffer

Bloom filters

fence pointers

true negative

false positive

true positive

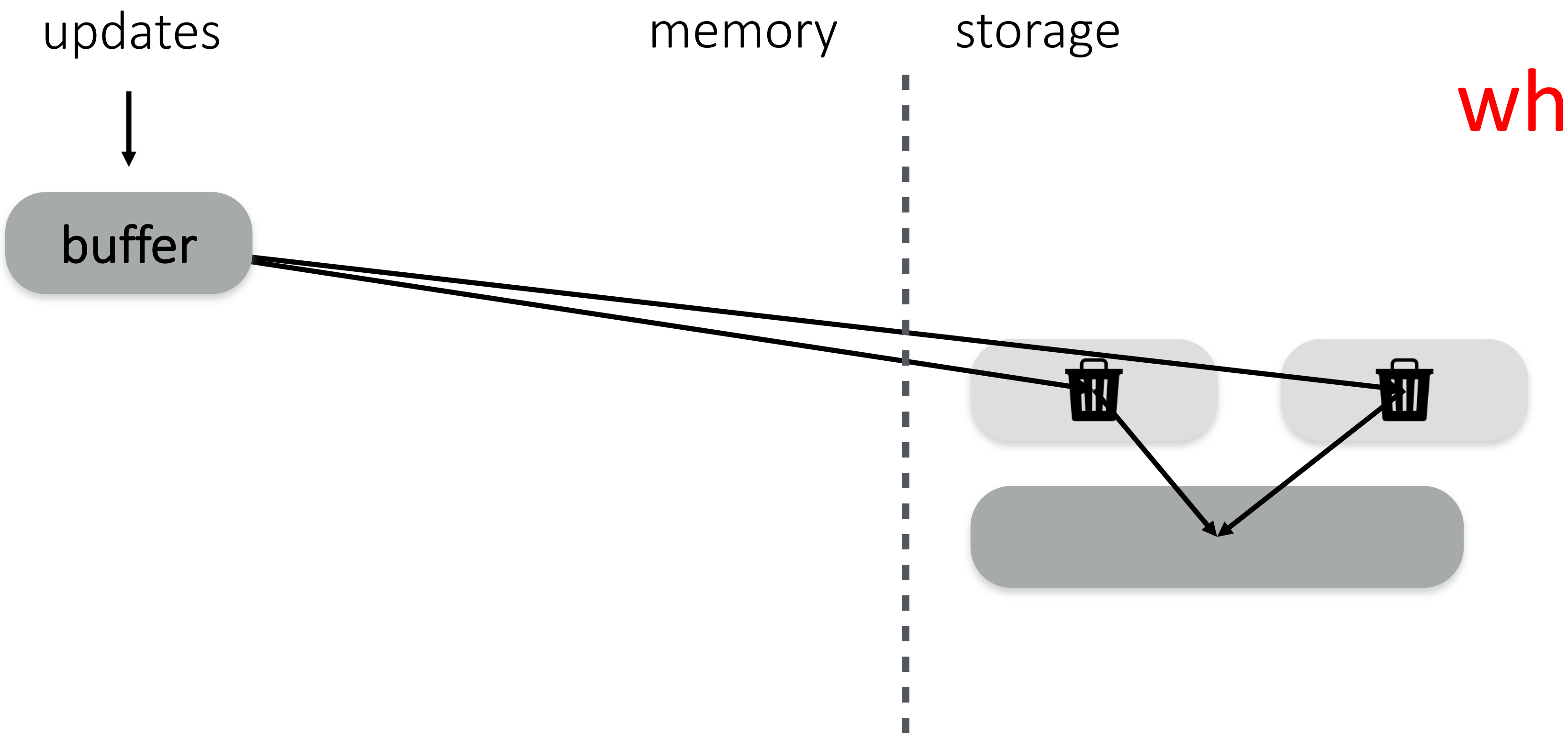
one I/O per run

X

range scans?

deletes?

remember merging?



what strategies?

sort & flush

runs

sort-merge

BOSTON UNIVERSITY

Merge Policies

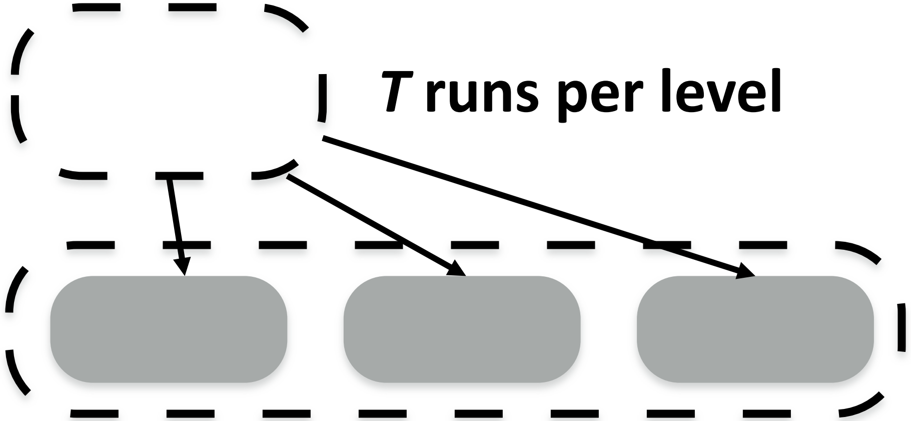
Tiering

write-optimized

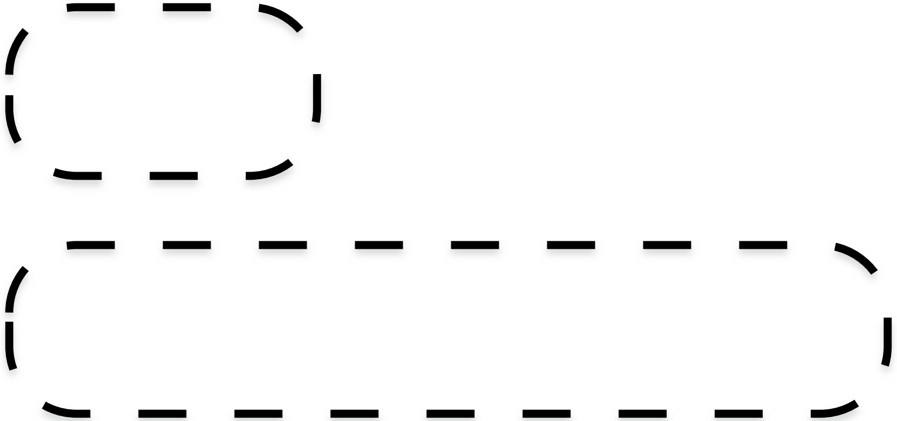
Leveling

read-optimized

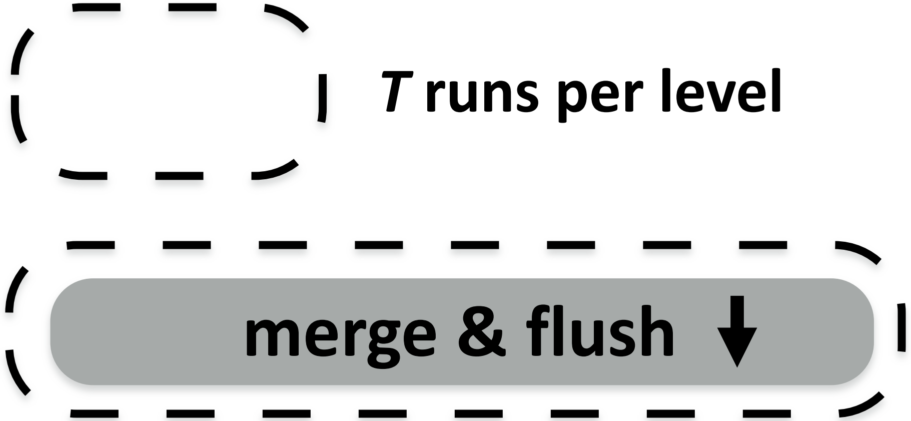
Tiering
write-optimized



Leveling
read-optimized



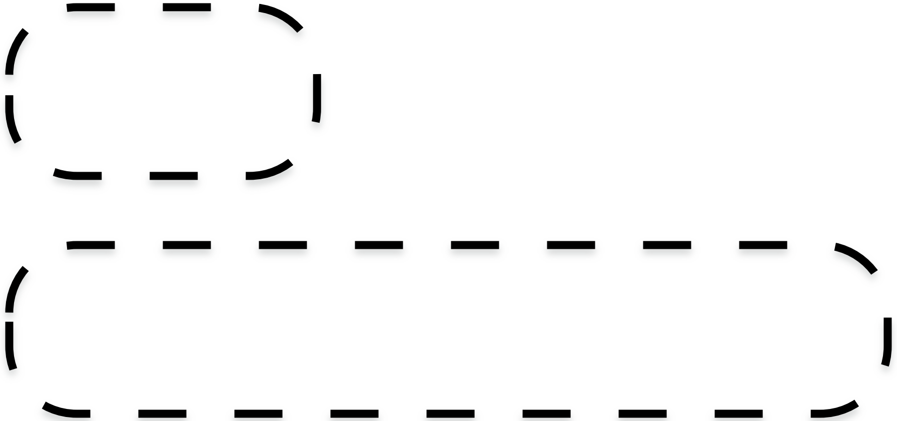
Tiering
write-optimized



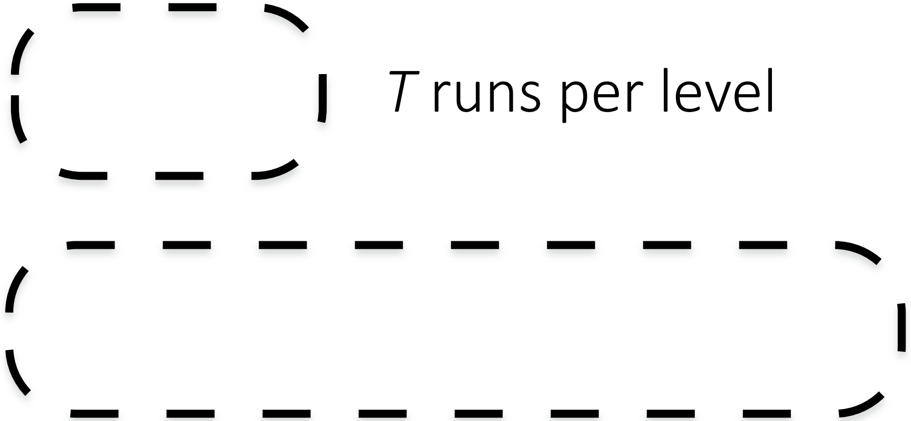
***T* runs per level**

merge & flush ↓

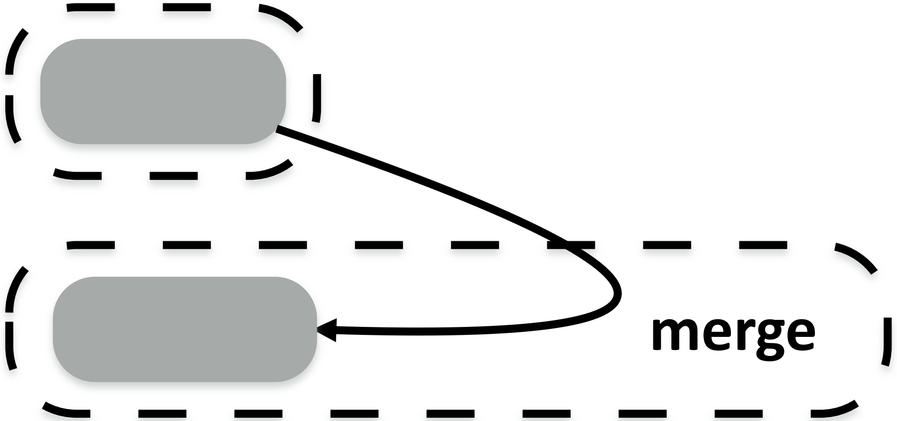
Leveling
read-optimized



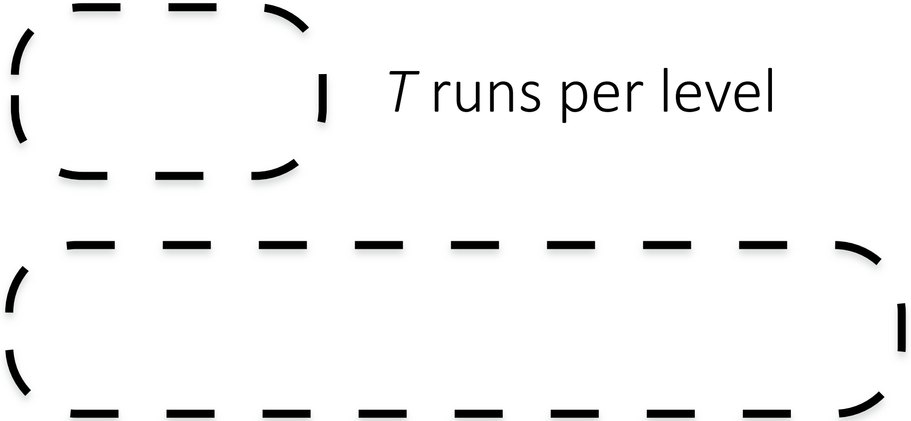
Tiering
write-optimized



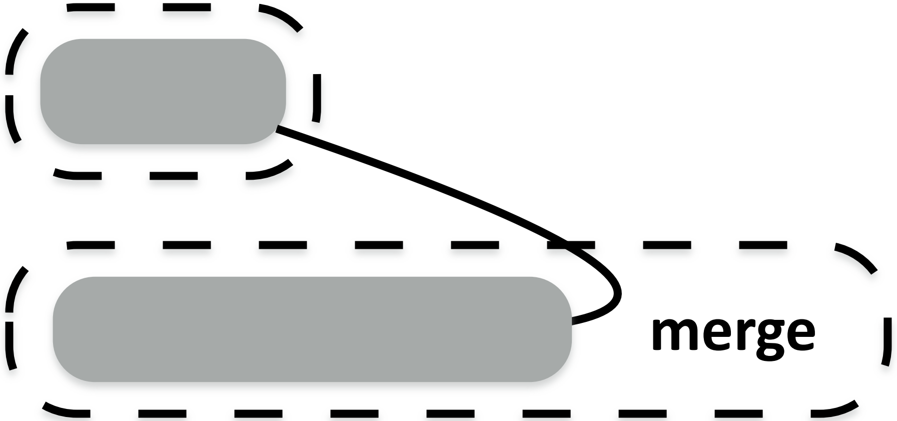
Leveling
read-optimized



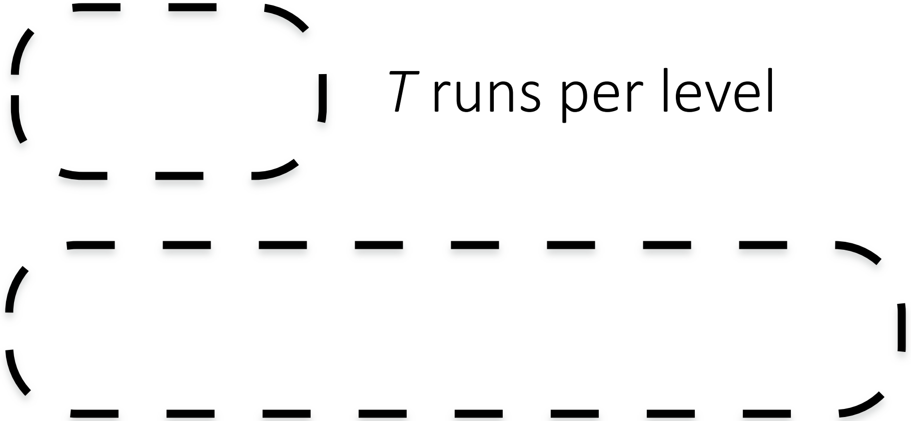
Tiering
write-optimized



Leveling
read-optimized

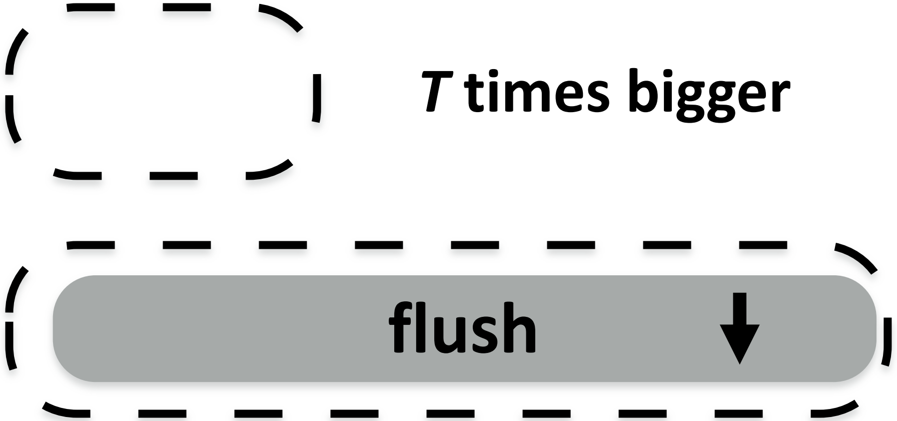


Tiering
write-optimized



T runs per level

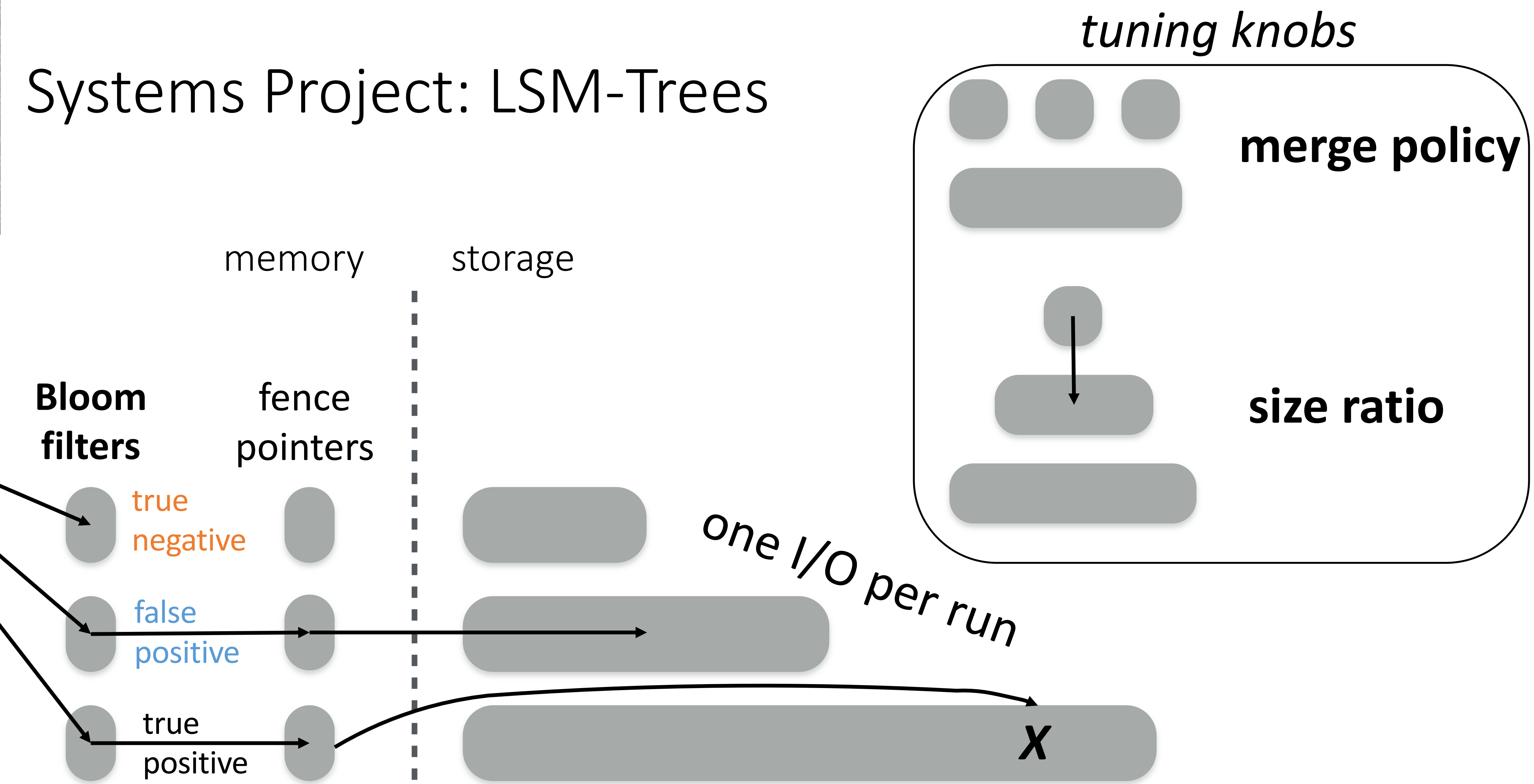
Leveling
read-optimized



***T* times bigger**

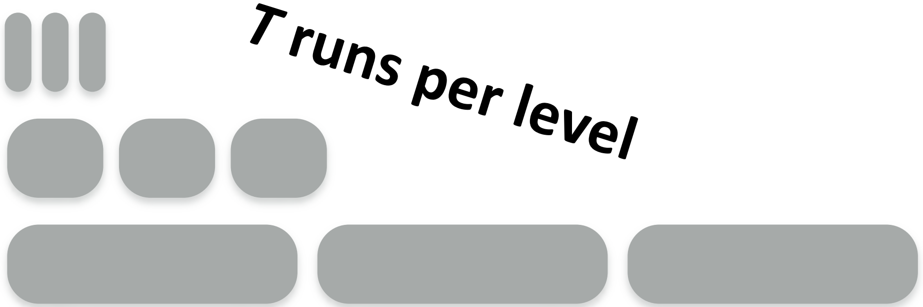
flush

Systems Project: LSM-Trees

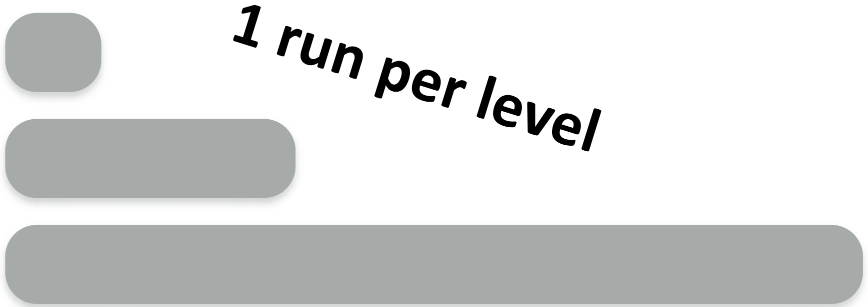


more on LSM-Tree performance

Tiering
write-optimized



Leveling
read-optimized



lookup cost:

$$O\left(T \cdot log_T(N) \cdot e^{-M/N}\right)$$

runs
per level

levels

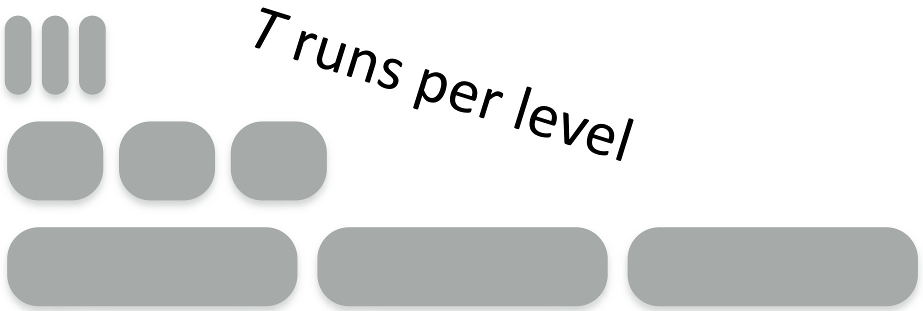
false
positive rate

$$O\left(log_T(N) \cdot e^{-M/N}\right)$$

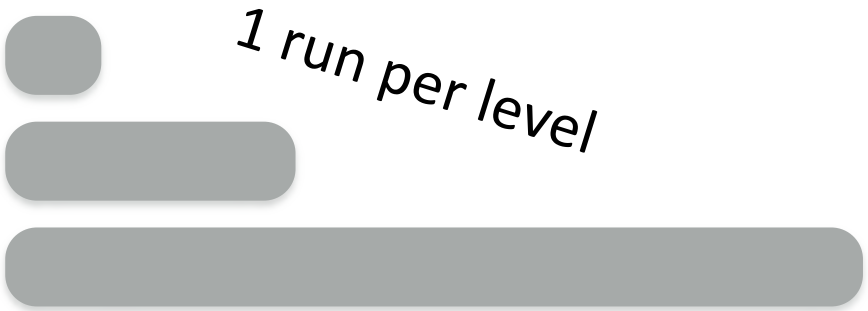
levels

false
positive rate

Tiering
write-optimized



Leveling
read-optimized



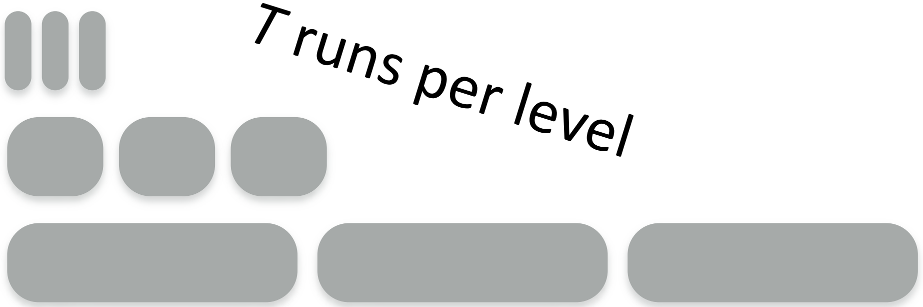
lookup cost: $O\left(T \cdot log_T(N) \cdot e^{-M/N}\right)$ $O\left(log_T(N) \cdot e^{-M/N}\right)$

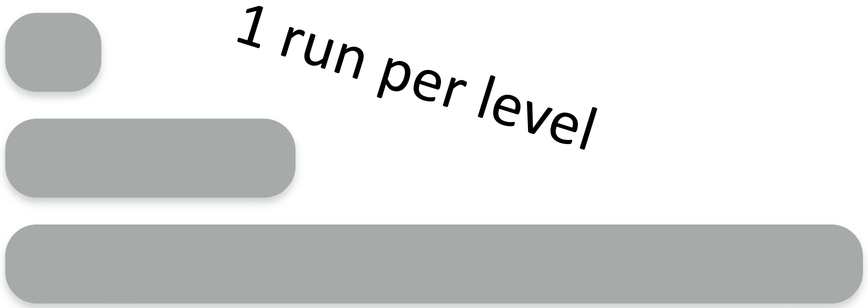
update cost: $\boldsymbol{O\left(\underset{\underset{\text{levels}}{\uparrow}}{log_T(N)}\right)}$ $\boldsymbol{O\left(\underset{\underset{\text{merges per level}}{\nearrow}}{T} \cdot \underset{\underset{\text{levels}}{\nwarrow}}{log_T(N)}\right)}$

BOSTON
UNIVERSITY

Tiering
write-optimized

Leveling
read-optimized

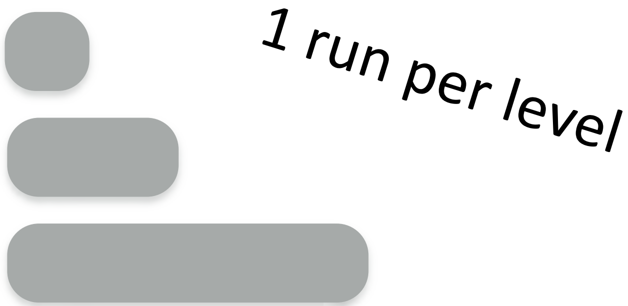




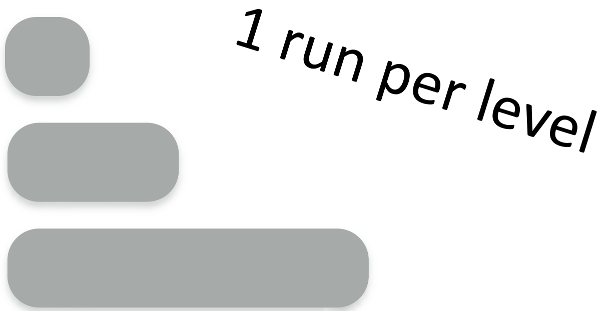
lookup cost:	$O\left(T \cdot log_T(N) \cdot e^{-M/N}\right)$	$O\left(log_T(N) \cdot e^{-M/N}\right)$
update cost:	$O\left(log_T(N)\right)$	$O\left(T \cdot log_T(N)\right)$

for size ratio T

Tiering
write-optimized



Leveling
read-optimized



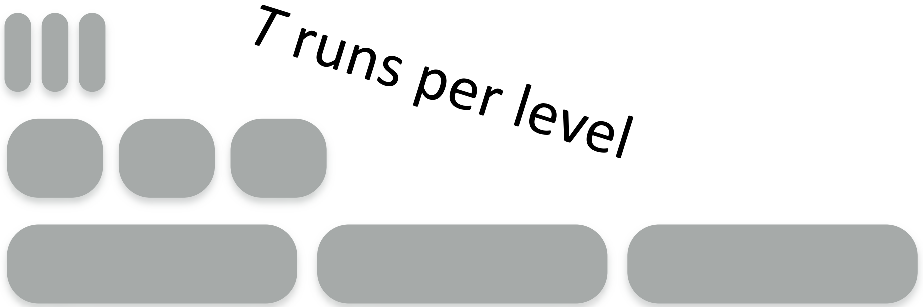
lookup cost: $O\left(log_T(N) \cdot e^{-M/N}\right) = O\left(log_T(N) \cdot e^{-M/N}\right)$

update cost: $O\left(log_T(N)\right) = O\left(log_T(N)\right)$

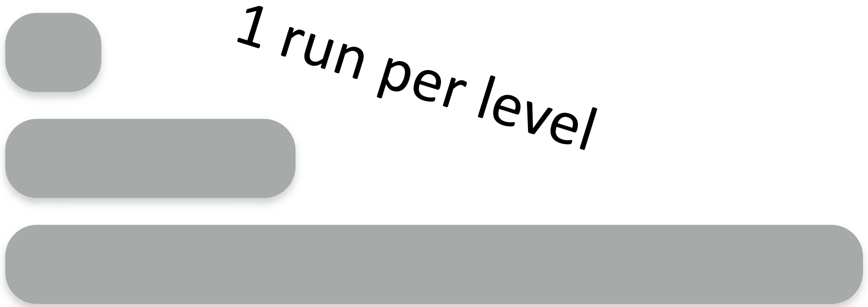
for size ratio T ⌄⌄

BOSTON UNIVERSITY

Tiering
write-optimized



Leveling
read-optimized

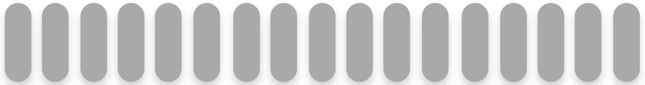


	Tiering	Leveling
lookup cost:	$O\left(T \cdot log_T(N) \cdot e^{-M/N}\right)$	$O\left(log_T(N) \cdot e^{-M/N}\right)$
update cost:	$O\left(log_T(N)\right)$	$O\left(T \cdot log_T(N)\right)$

for size ratio T ⇑

Tiering
write-optimized

$O(N)$ runs per level



log

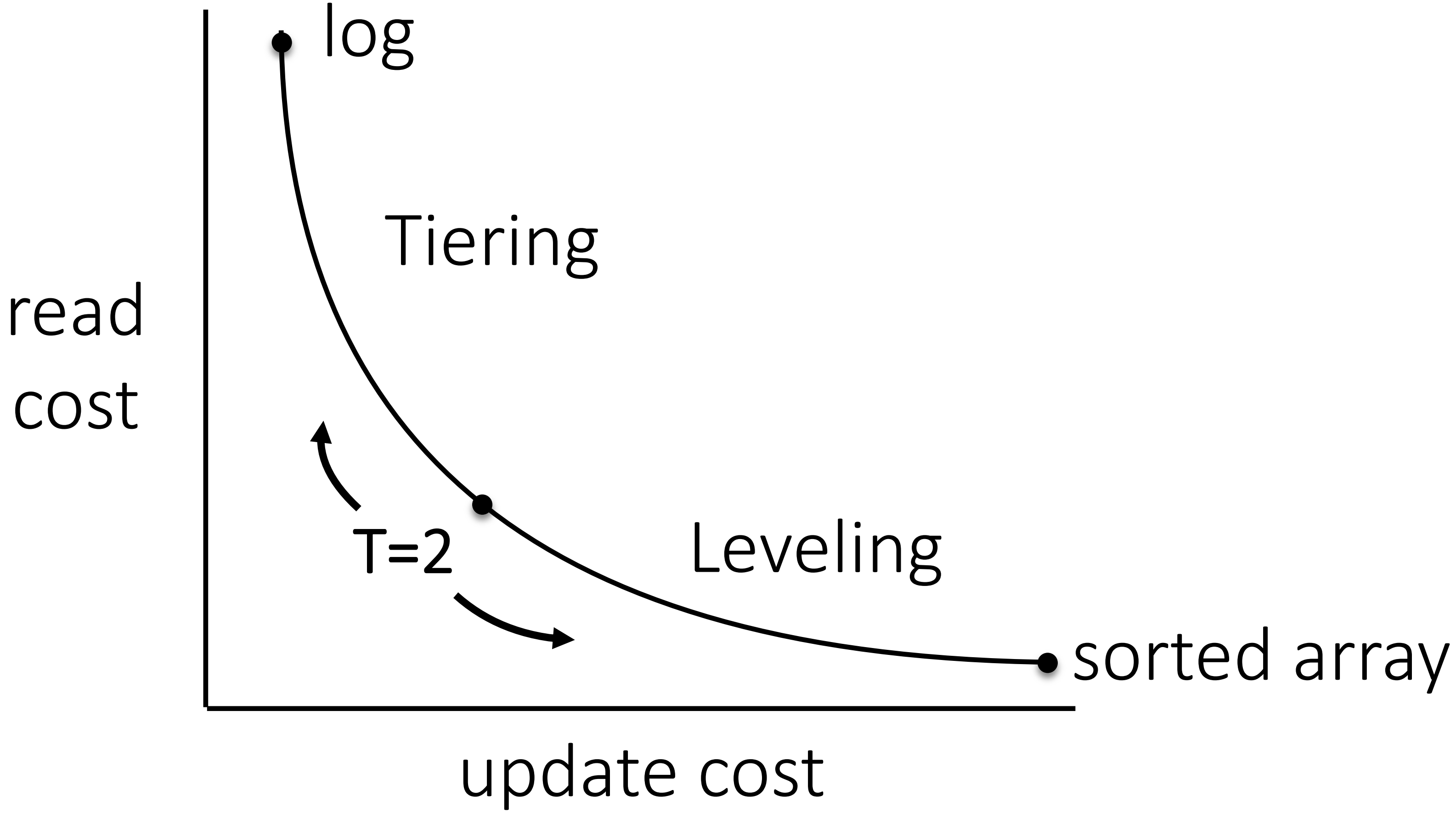
Leveling
read-optimized

1 run per level

sorted array

	Tiering	Leveling
lookup cost:	$O\left(T \cdot log_T(N) \cdot e^{-M/N}\right)$	$O\left(log_T(N) \cdot e^{-M/N}\right)$
update cost:	$O\left(log_N(N)\right) = \boldsymbol{O(1)}$	$O\left(N \cdot log_N(N)\right) = \boldsymbol{O(N)}$

for size ratio T $\overset{\mathbf{N}}{\Uparrow}$



T : size ratio

Research Question on LSM-Trees

how to do range scans?

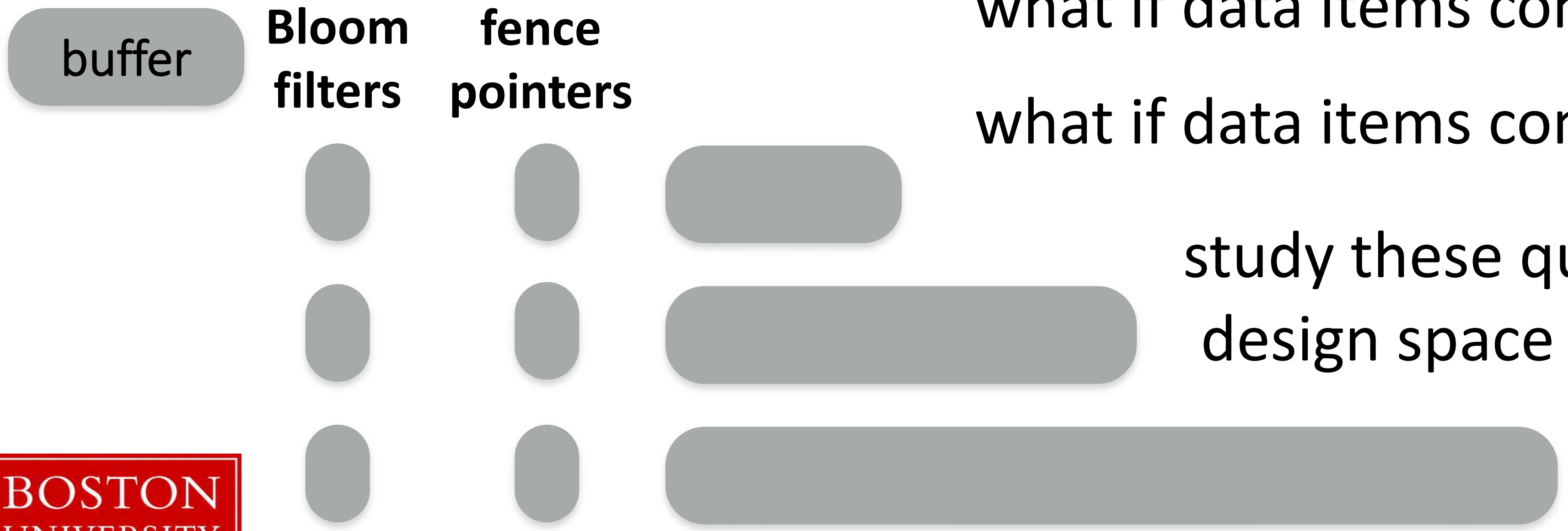
how to delete? how to delete ***quickly***?

how to allocate memory between buffer/Bloom filters/fence pointers?

what is the CPU overhead of Bloom filters?

what if data items come ordered?

what if data items come ***almost ordered***?



buffer

Bloom filters

fence pointers

study these questions and navigate LSM design space using Facebook’s RocksDB

What “almost ordered” even mean?

Research question on ***sortedness***

How to quantify it? Need a metric!

How does the sortedness of the data affect
the behavior of LSM-Trees, B-Trees, Zonemaps?

similar question to:
how does the order of the values in an array affect a sorting algorithm

How to tune our system?

if we know the workload …

LSM-Trees: memory (Buffer/BF/FP) – what about caching?

Back to column-stores: do we need to sort?

partition the data?

add ***empty slots*** in the column for future inserts?

Workload-based tuning

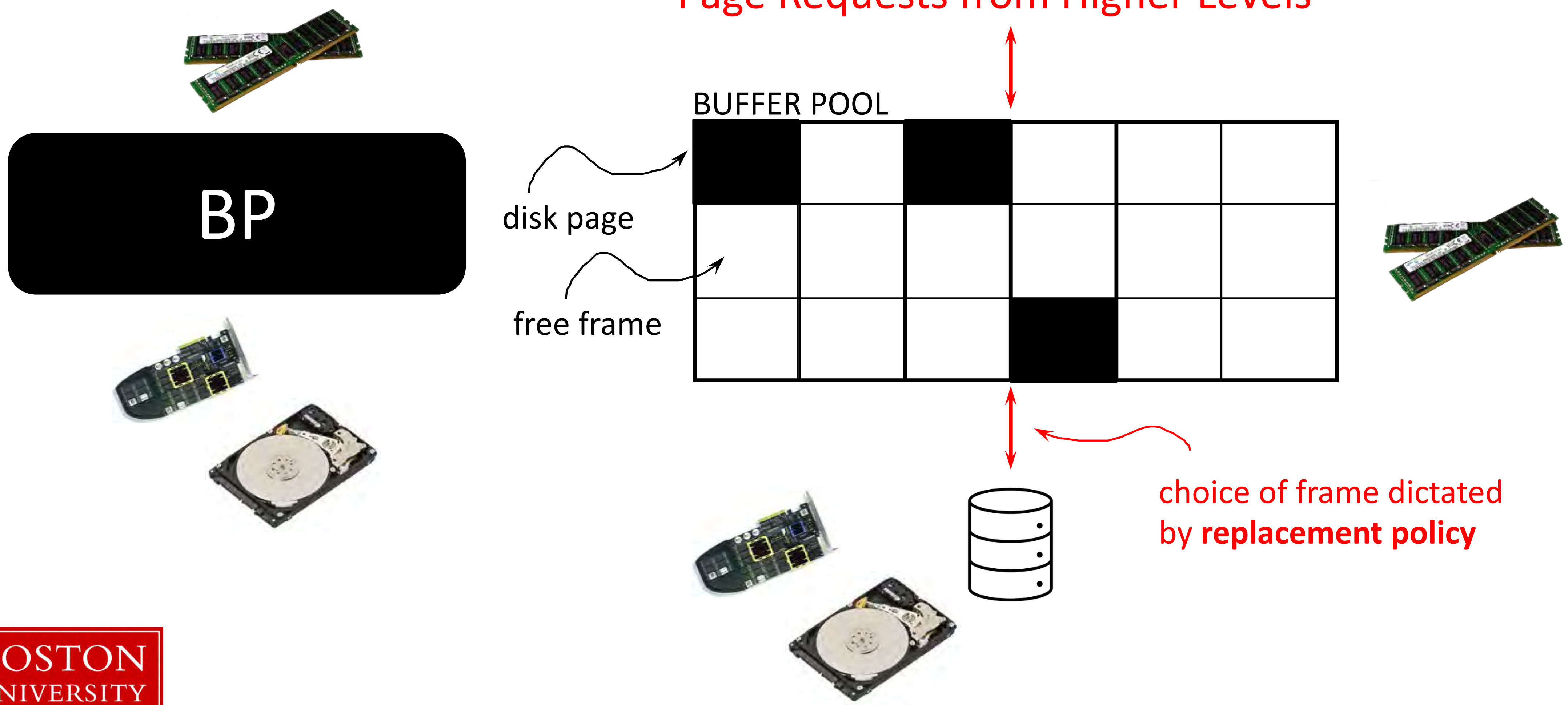
find *Tuning*, s.t.
min *cost*(*Workload, Data, Tuning)*
given *Workload* and *Data*

what if workload information is a bit wrong?

robust optimization (come and find me)

Asynchronous Bufferpool

what is the bufferpool?



BP

Page Requests from Higher Levels

BUFFER POOL

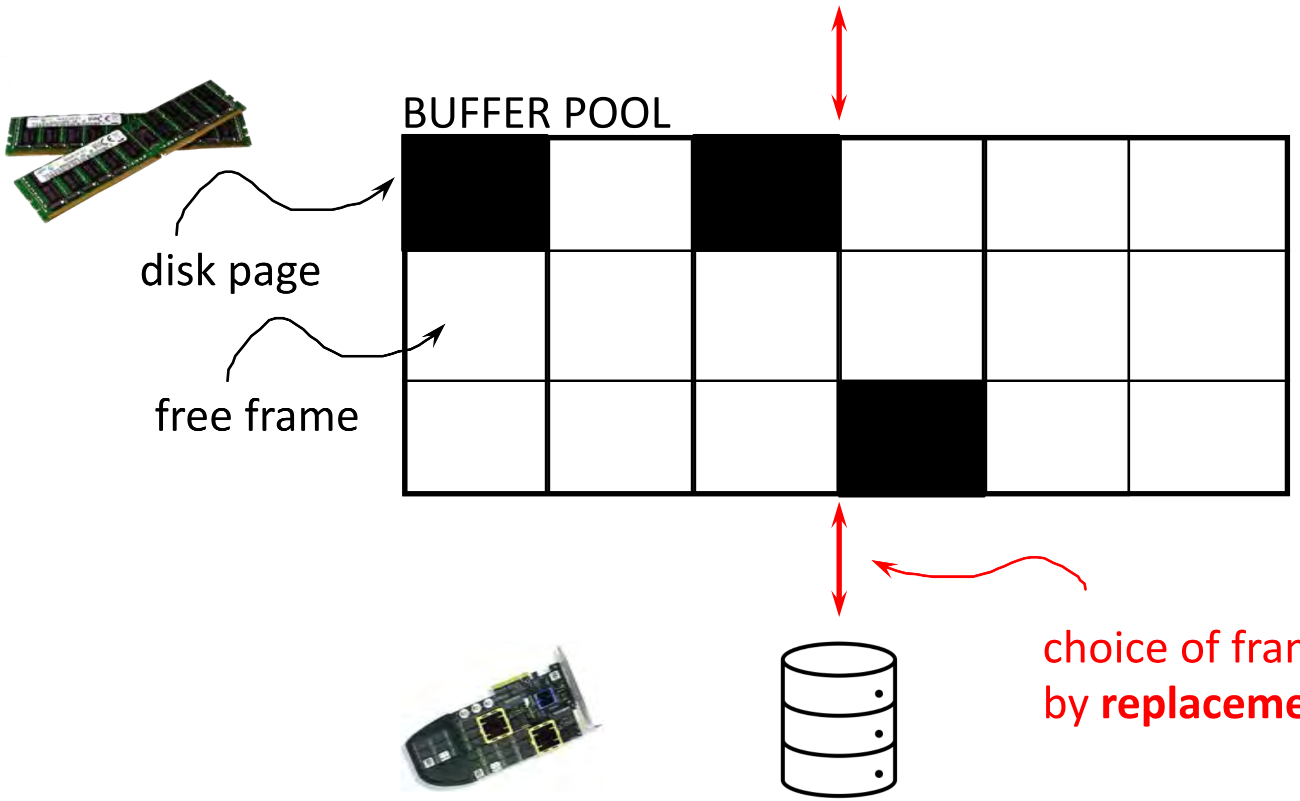
disk page

free frame

choice of frame dictated by **replacement policy**

Systems Project: Bufferpool

Page Requests from Higher Levels



choice of frame dictated by **replacement policy**

Implementation of a bufferpool

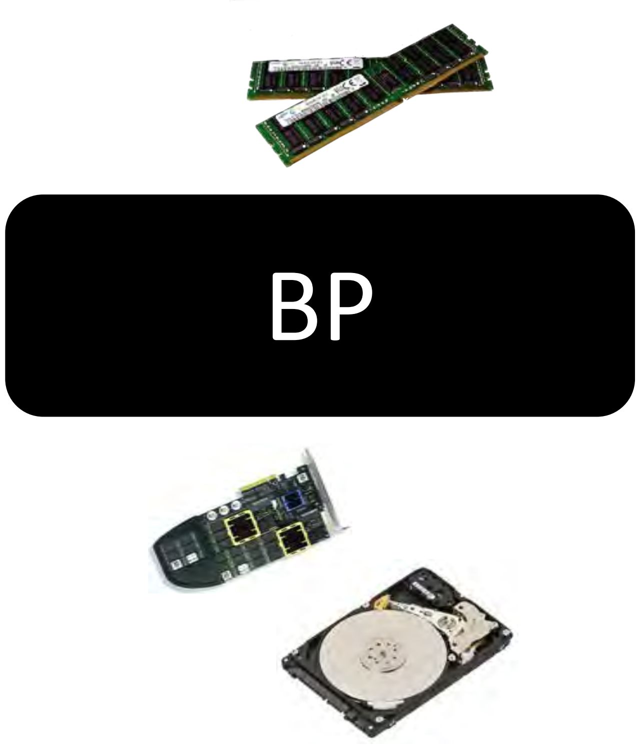
- **Application requests** a page
  - If **in the bufferpool** return it
  - If **not in the bufferpool** fetch it from the disk
    - If bufferpool is full select page to **evict**

Core Idea: Eviction Policy

- *Least Recently Used*
- *First In First Out*
- *more …*

Asynchronous Bufferpool

what is the bufferpool?



manages available memory
reads/writes from/to disk

what happens when full?

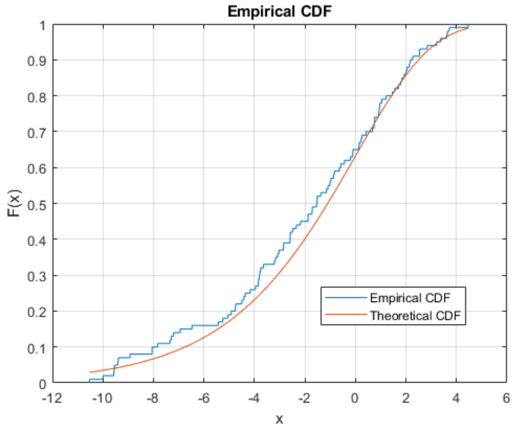
writes one page back and reads on page

is this optimal?

what is an index?

sorted data

1 1 1 2 3 5 10 11 12 13 18 19 20 50 54 58 62 98 101 102



$$postition(val) = CDF(val) \cdot array\_size$$

can you learn the CDF?

what is the best way to do so?

how to update that?

BOSTON UNIVERSITY

what to do now?

systems project
form groups of 1-2
(speak to me in OH if you want to work on your own)

research project
form groups of 2-3
pick one of the subjects & read background material
define the behavior you will study and address
sketch approach and success metric
(if LSM-related get familiar with RocksDB)

what to do now?

systems project

form groups of 1-2

(speak to me in OH if you want to work on your own)

research project

form groups of 2-3

pick one of the subjects & read background material

define the behavior you will study and address

sketch approach and success metric

come to OH

finalize your project in 1-2 weeks (by Feb 14th)

submit proposal on February 21st

BOSTON UNIVERSITY

CS 561: Data Systems Architectures

Systems & Research Project

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/