

CS 561: Data Systems Architectures

class 4

Systems & Research Project

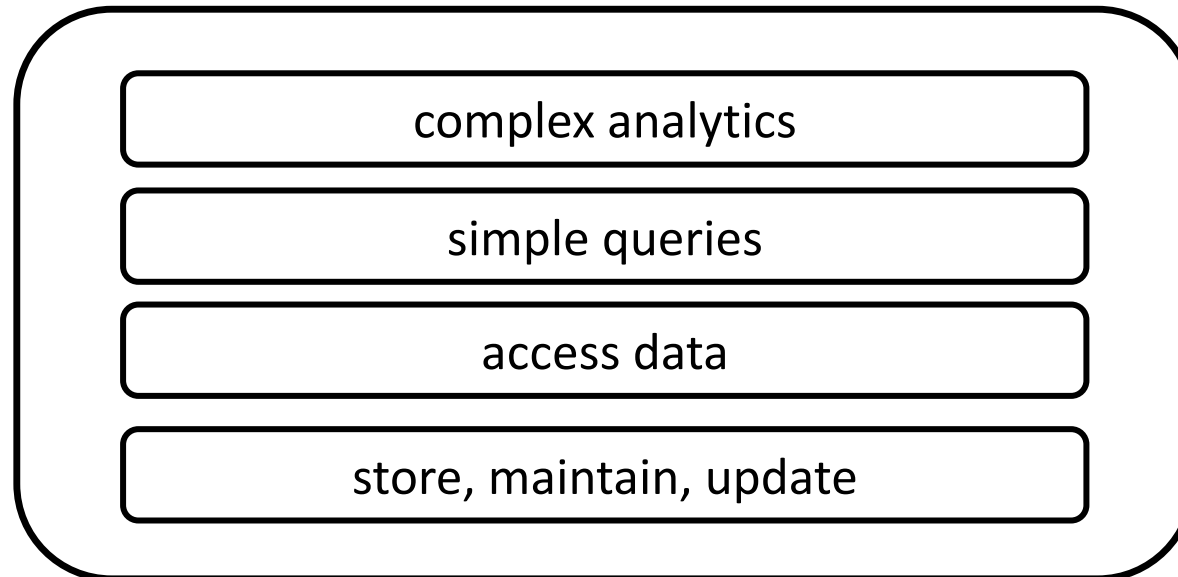
Prof. Manos Athanassoulis

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

data systems



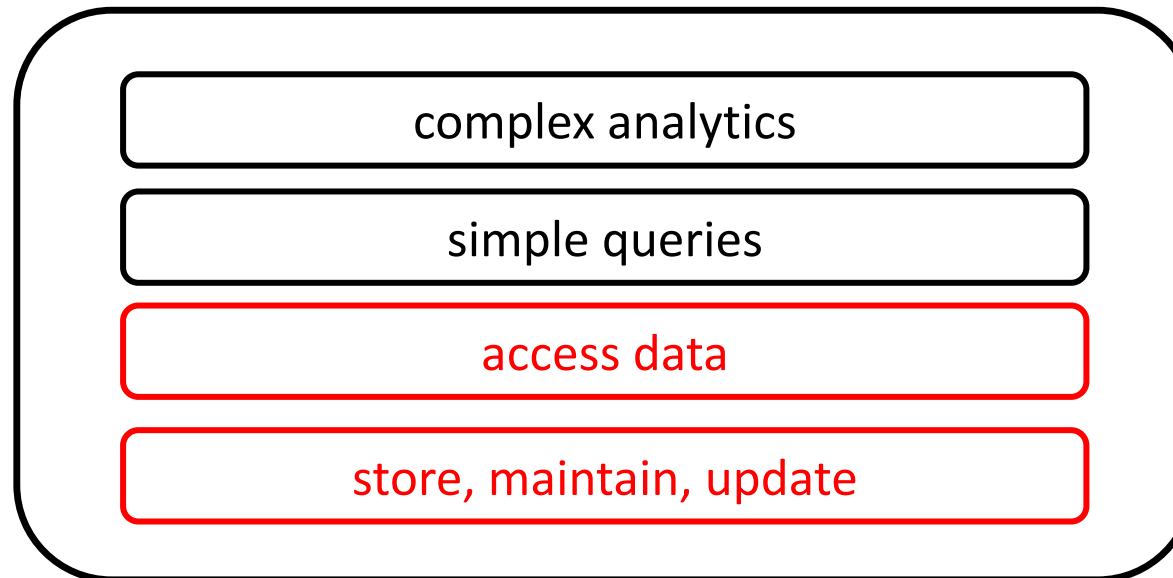
>\$200B by 2020, growing at 11.7% every year
[The Forbes, 2016]



data systems



>\$200B by 2020, growing at 11.7% every year
[The Forbes, 2016]



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



state-of-the-art design

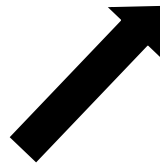
how general is a key value store?

can we store relational data?



yes! {<primary_key>, <rest_of_the_row>}

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



how to index these attributes?

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

what is the caveat?

other problems?

index: { **yob**, { **student_id₁**, **student_id₂**, ... } }

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_id₁**, **student_id₂**, ... } }

how to use a key-value store?

basic interface

`put(k,v)`

`{v} = get(k)` `{v1, v2, ...} = get(k)`

`{v1, v2, ...} = get_range(kmin, kmax)` `{v1, v2, ...} = full_scan()`

`c = count(kmin, kmax)`

deletes: delete(k)

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

get set: `{v1, v2, ...} = get_set(k1, k2, ...)`



how to build a key-value store?

if we have only **put** operations

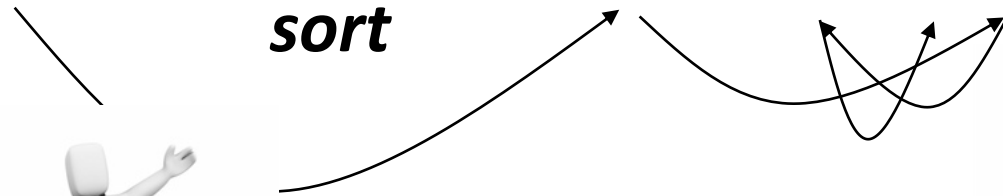
append



if we mostly have **get** operations



sort



what about full scan?

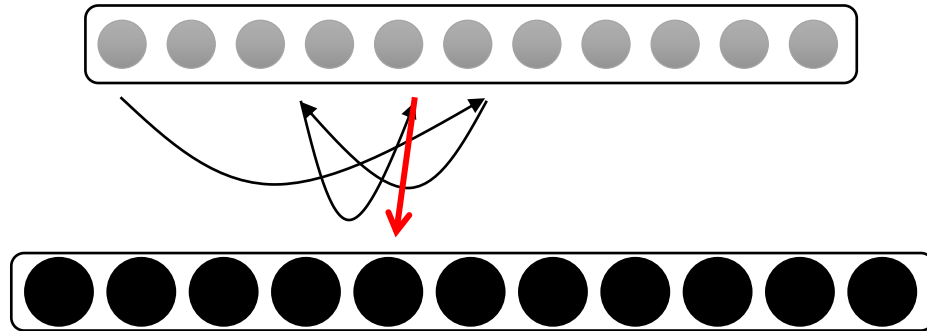
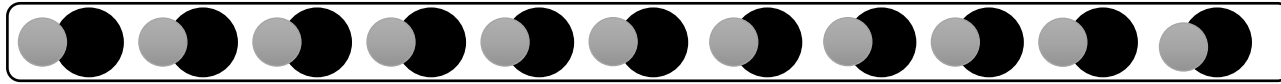


and then?



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?

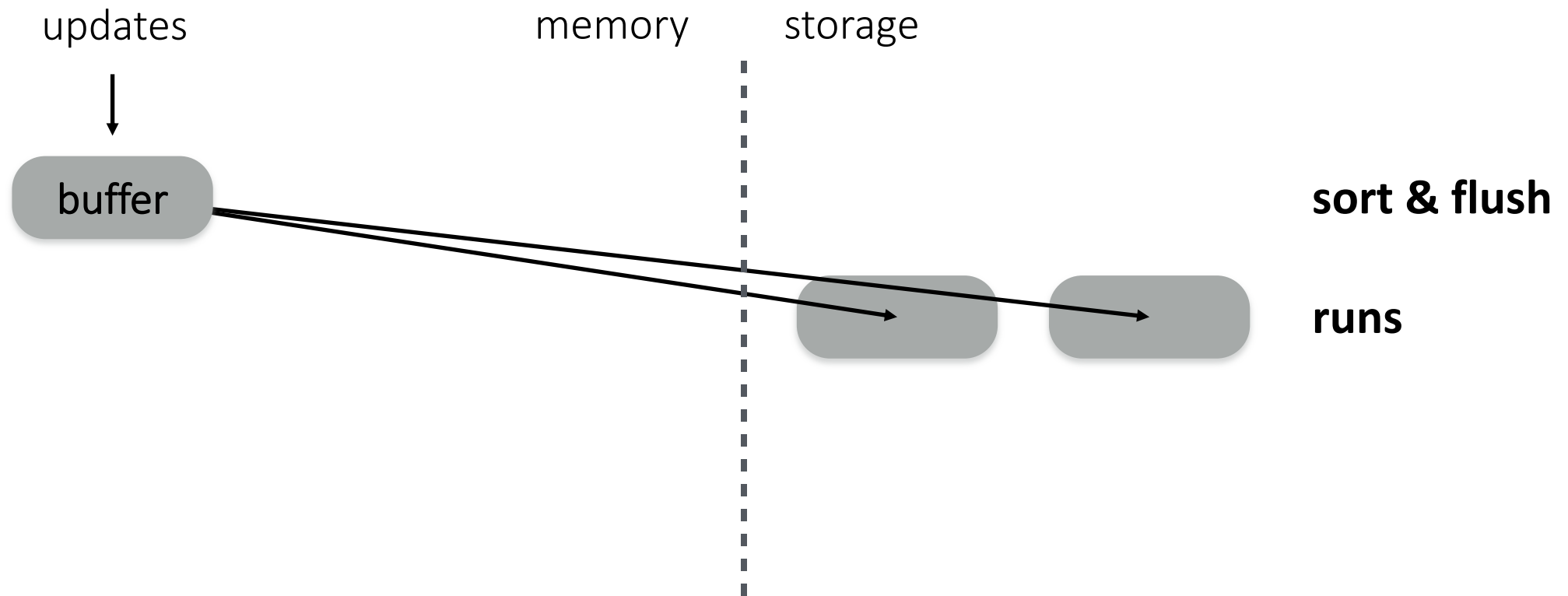
updates

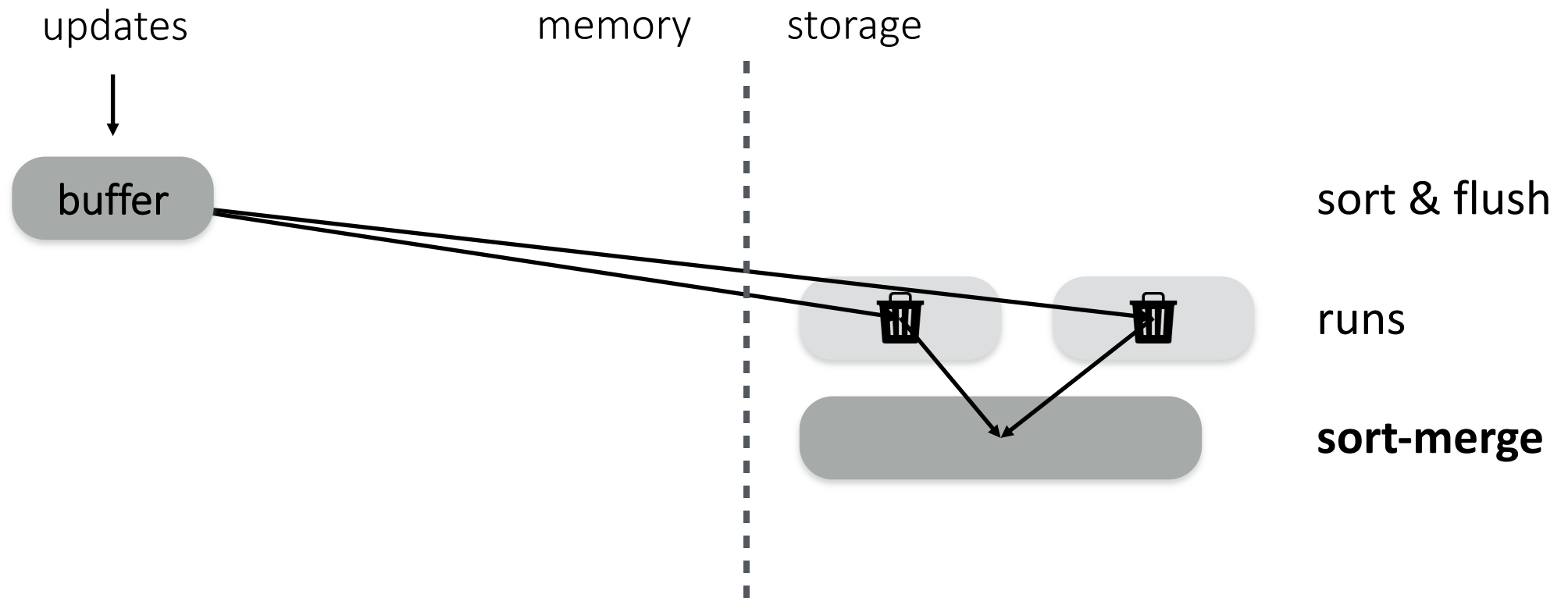


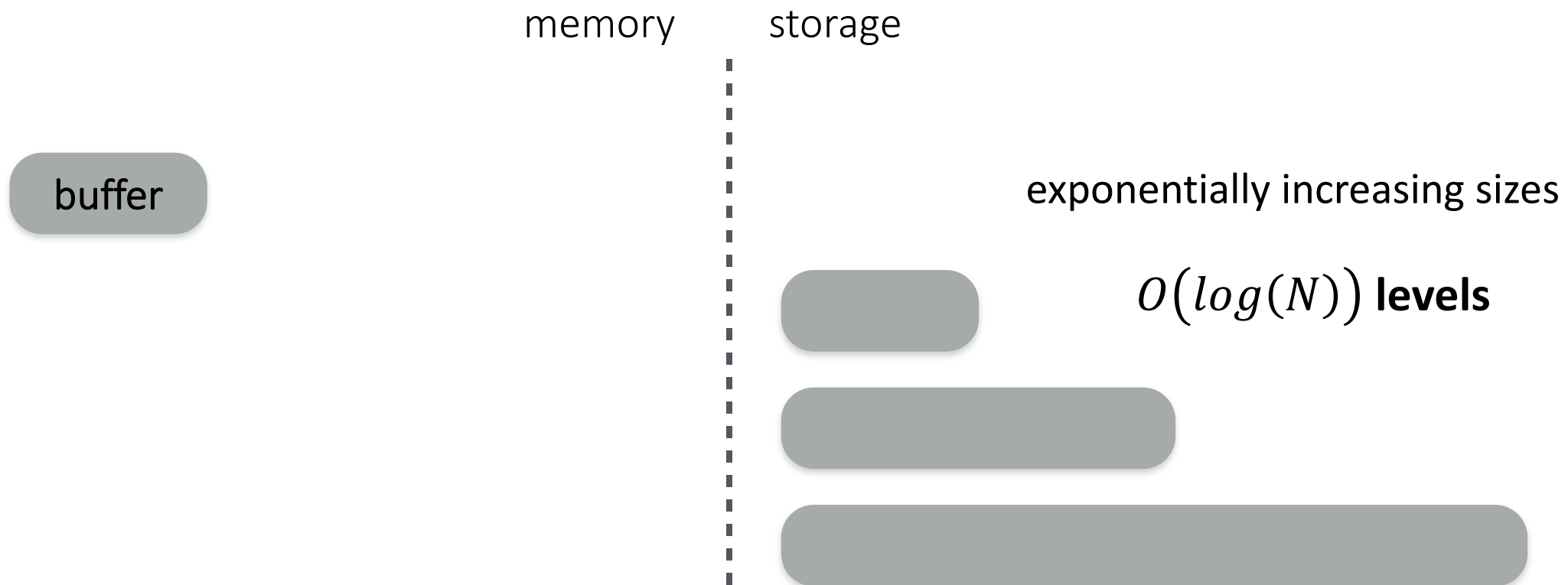
memory

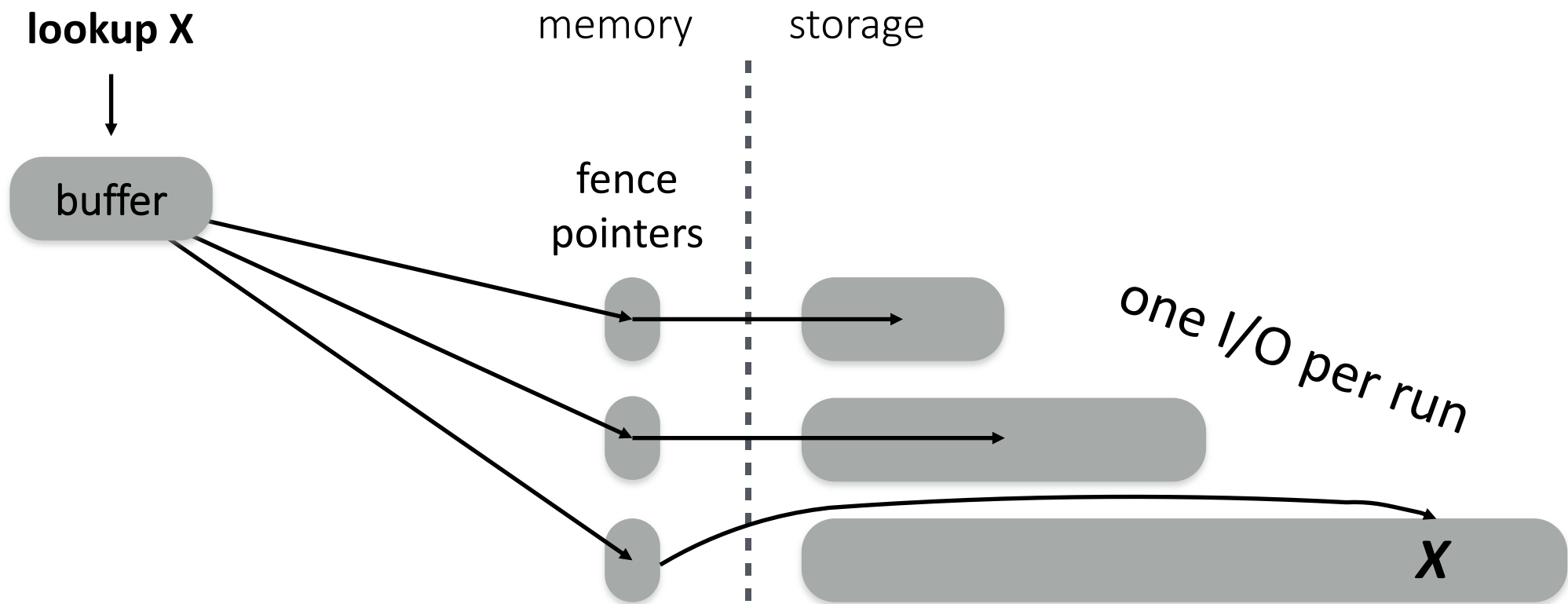
storage

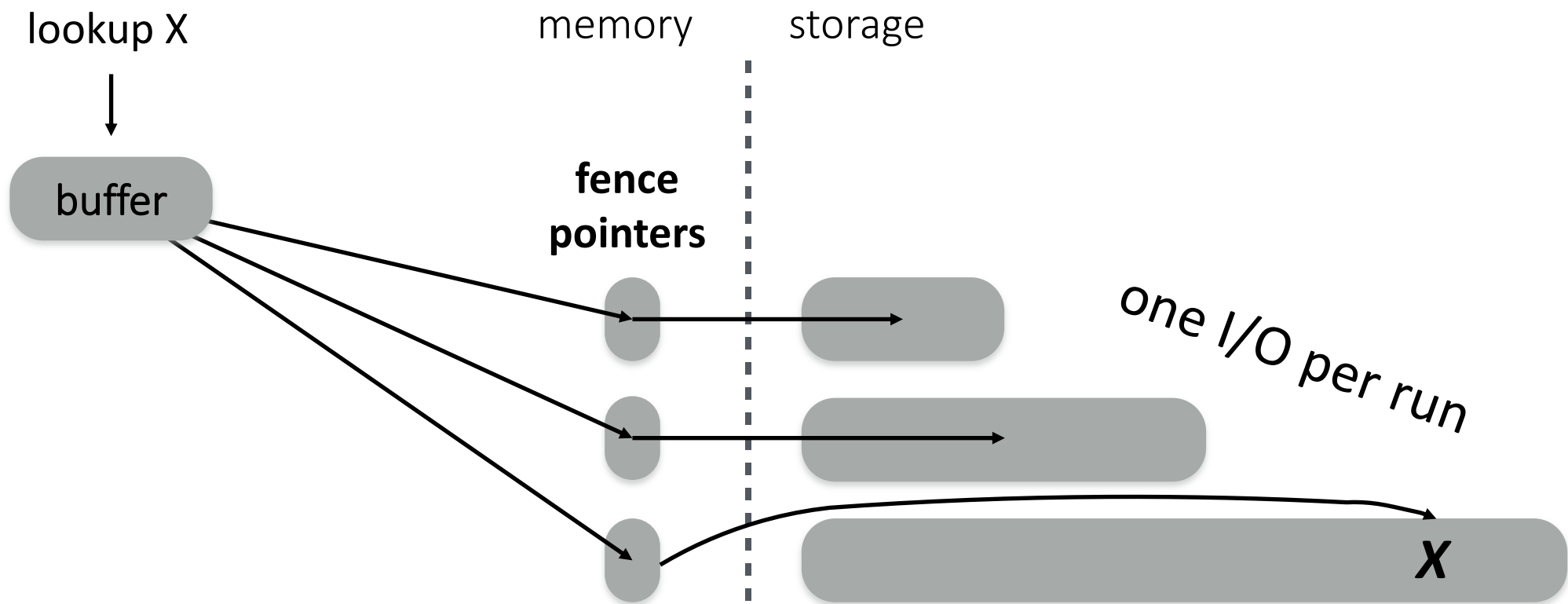


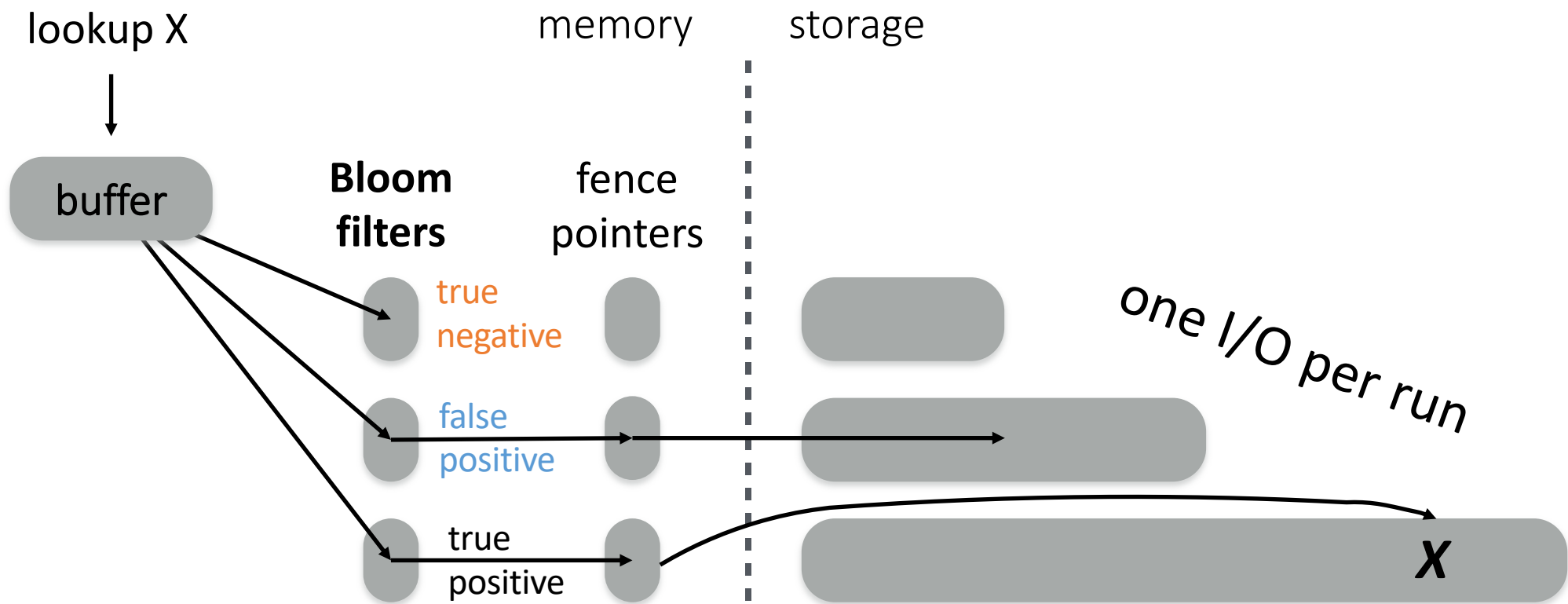




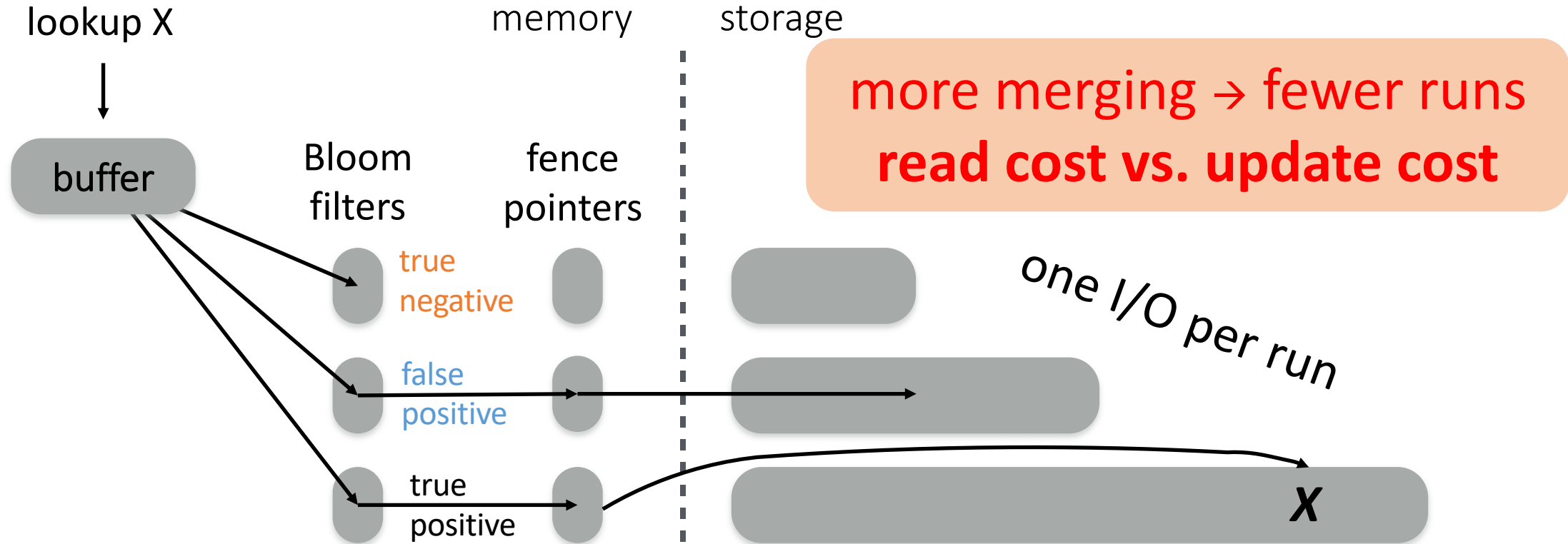




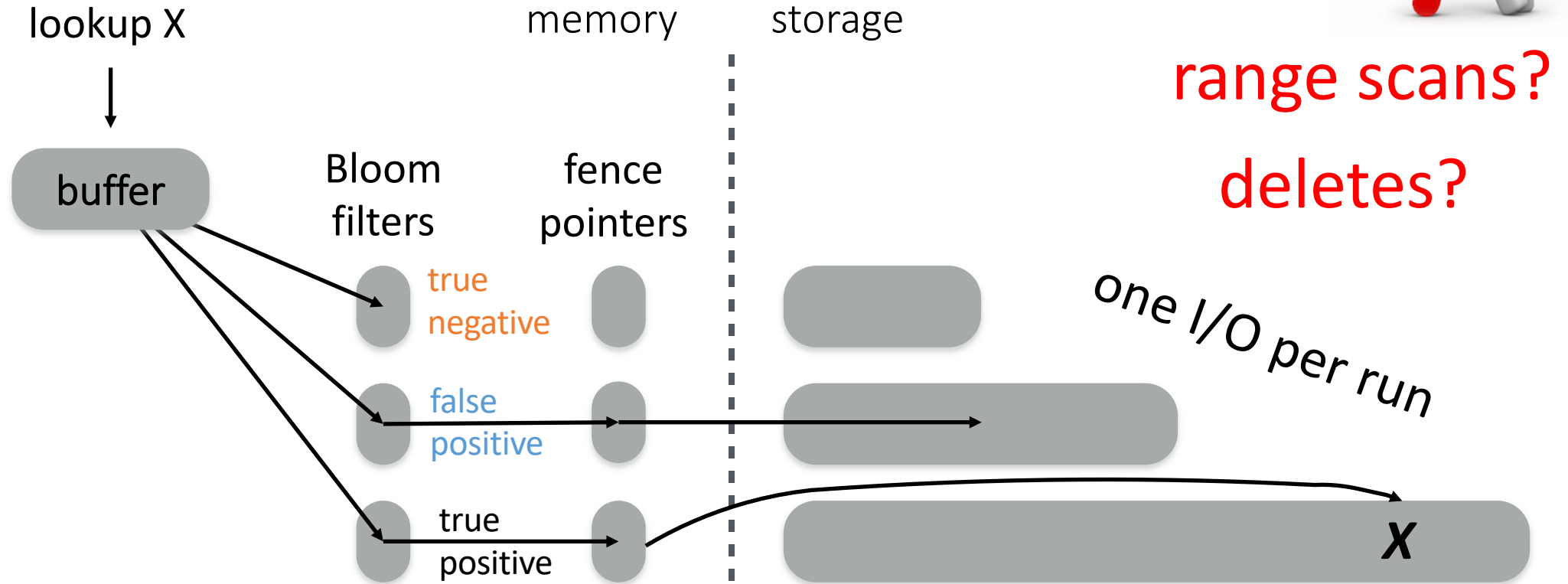




performance & cost trade-offs



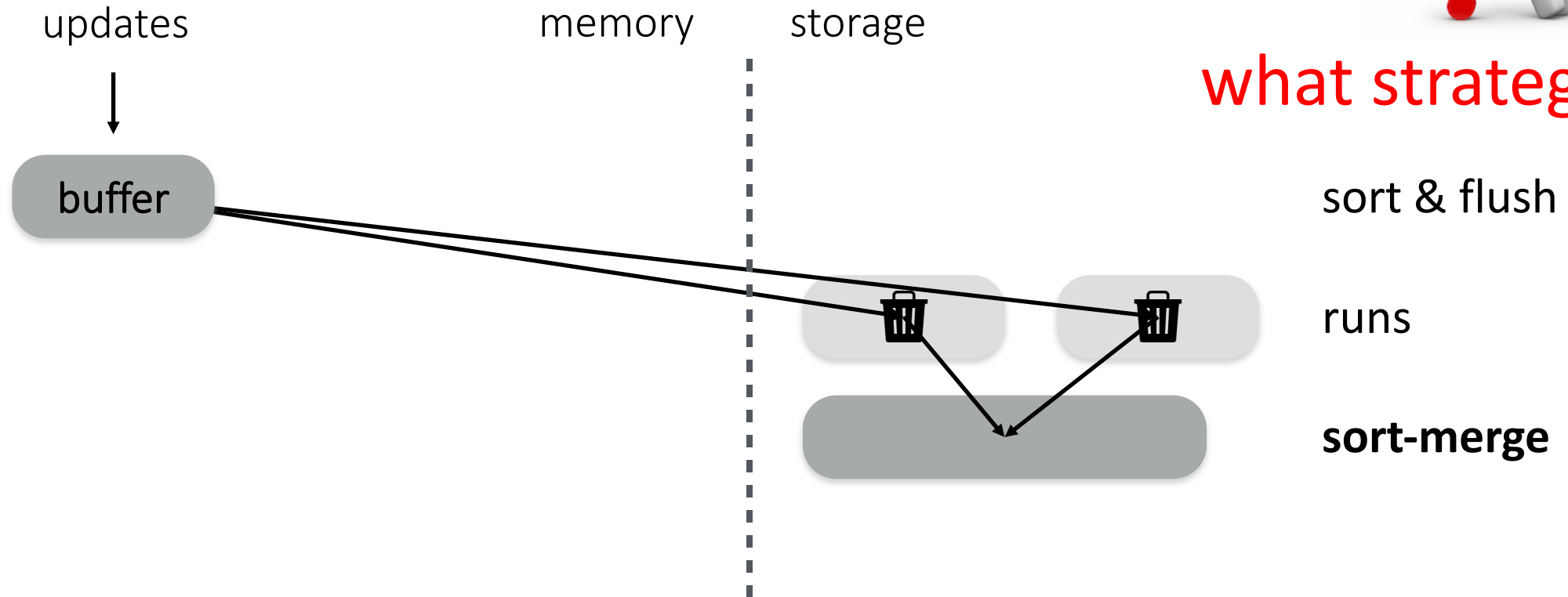
other operations



remember merging?



what strategies?



Merge Policies

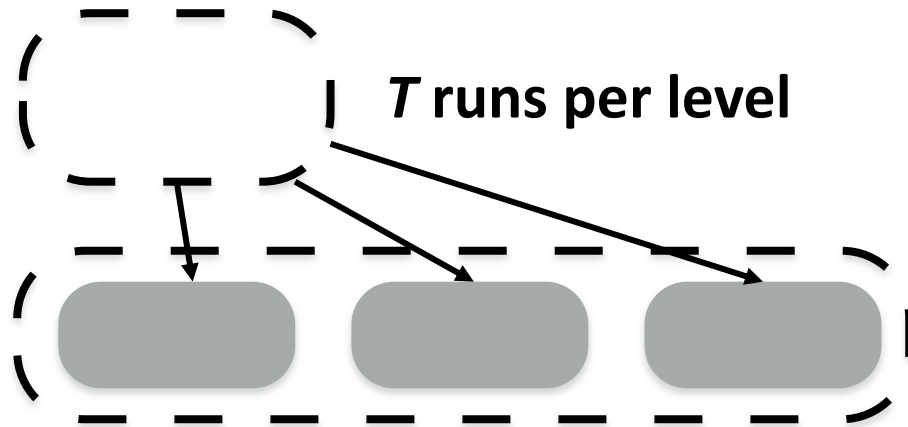
Tiering

write-optimized

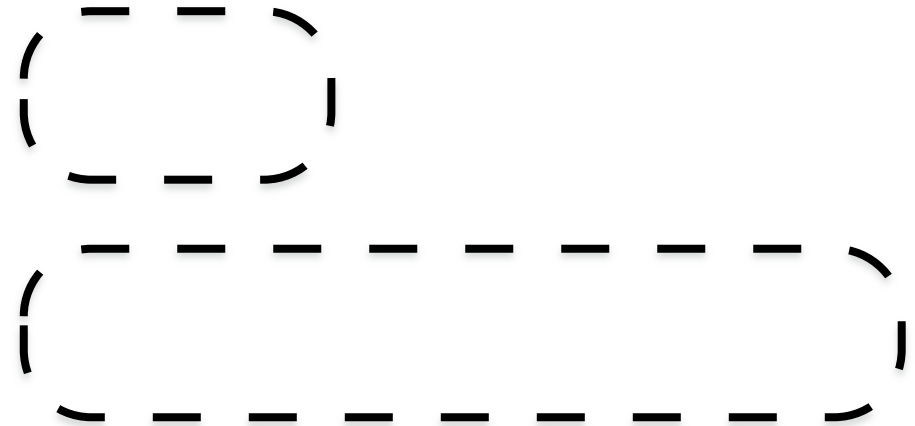
Leveling

read-optimized

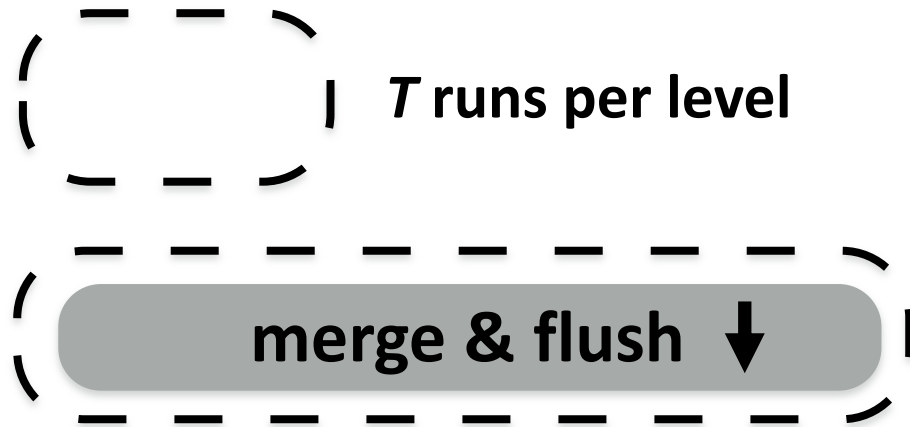
Tiering
write-optimized



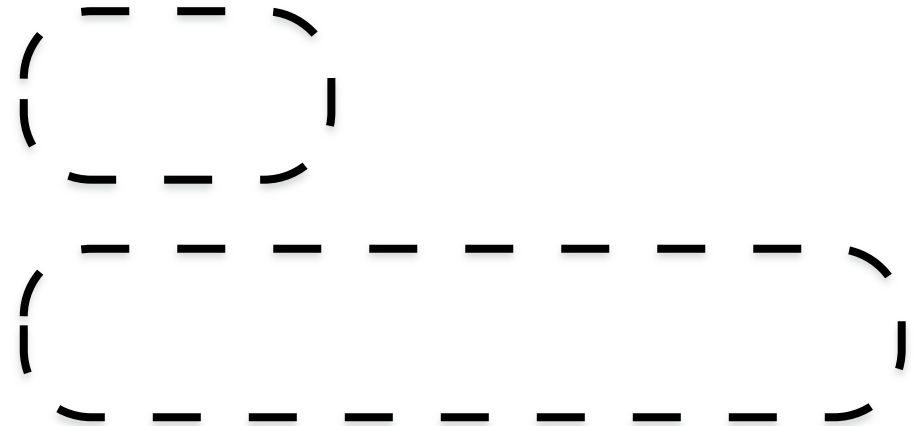
Leveling
read-optimized



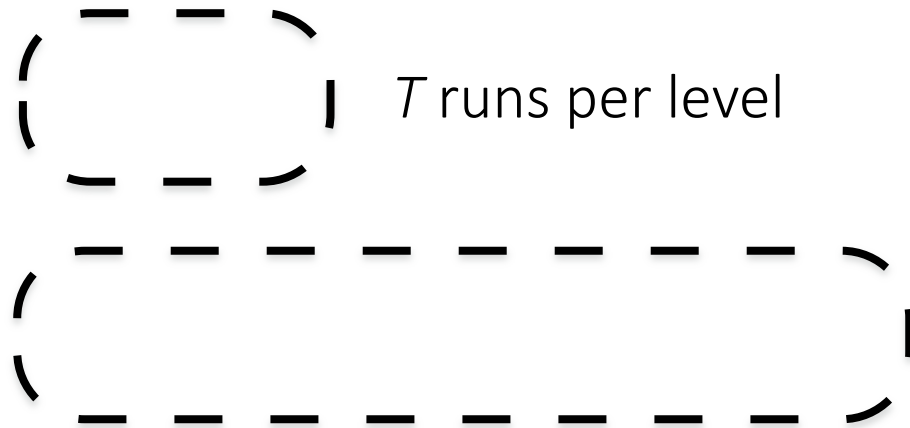
Tiering
write-optimized



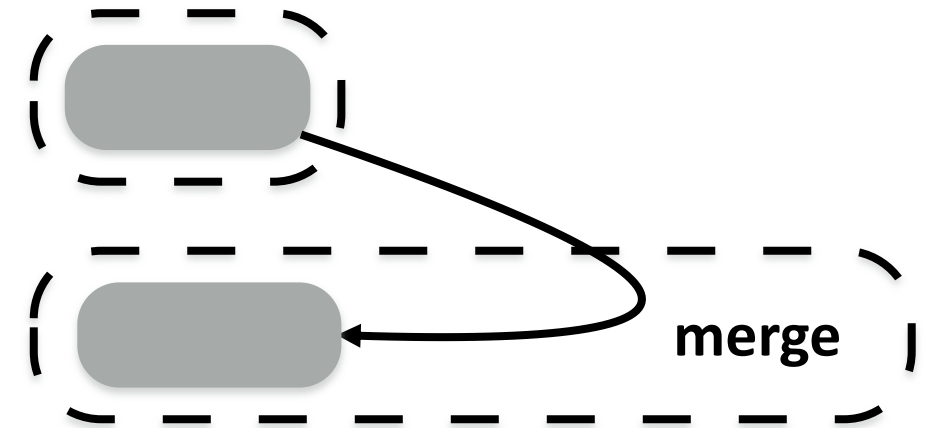
Leveling
read-optimized



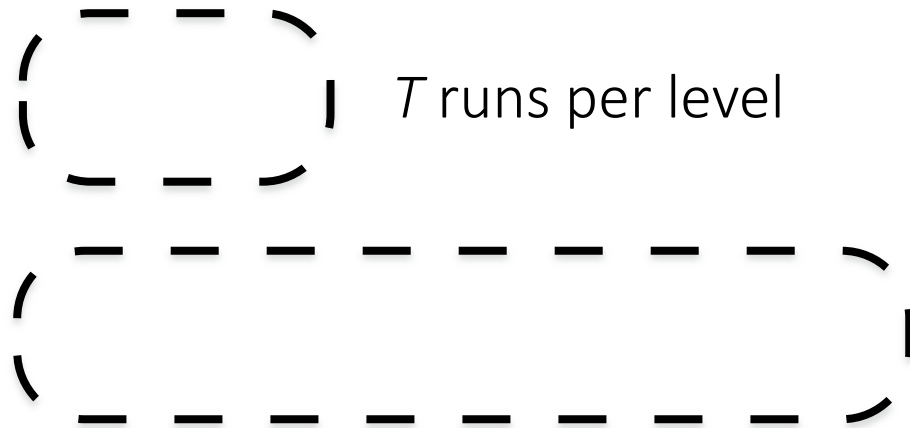
Tiering write-optimized



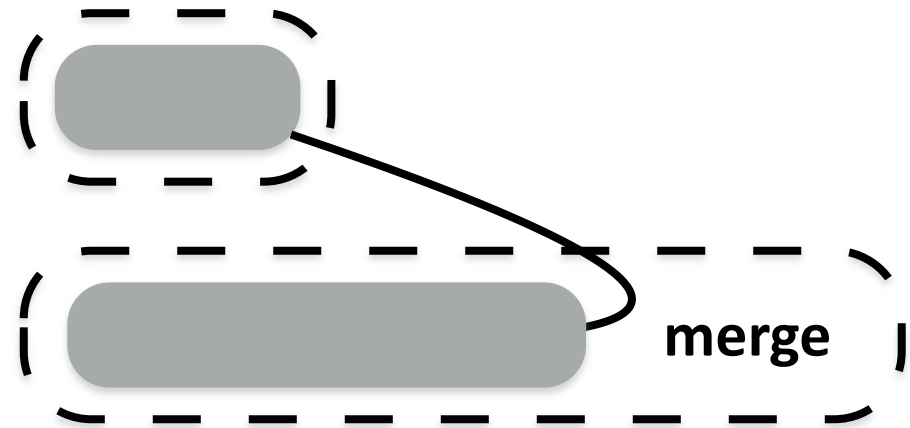
Leveling read-optimized



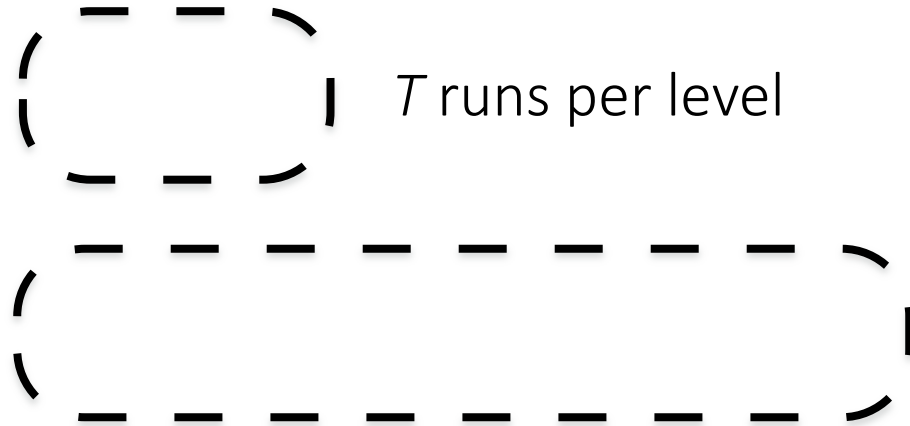
Tiering write-optimized



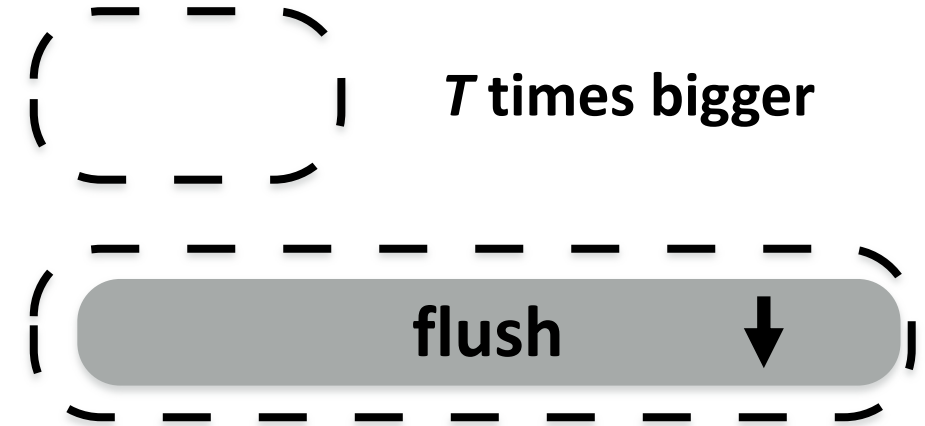
Leveling read-optimized



Tiering write-optimized

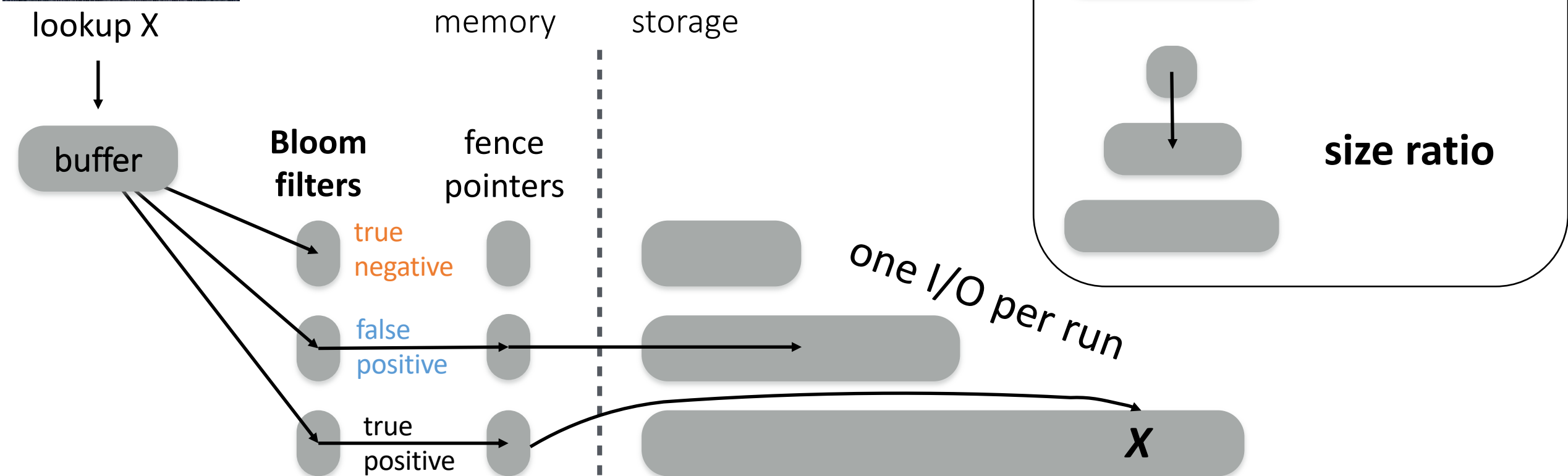


Leveling read-optimized



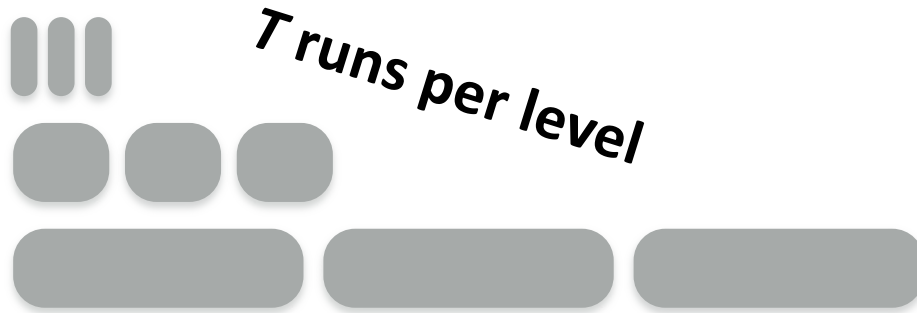


Systems Project: LSM-Trees



more on LSM-Tree performance

Tiering write-optimized



Leveling read-optimized



lookup cost:

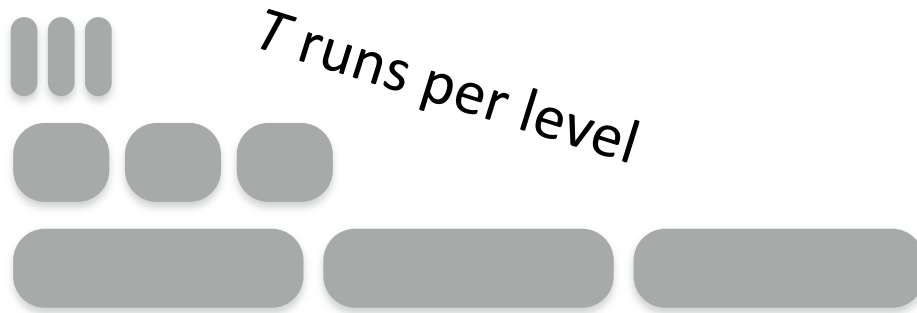
$$O(T \cdot \log_T(N) \cdot e^{-M/N})$$

runs
per level levels false
positive rate

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

levels false
positive rate

Tiering write-optimized



Leveling read-optimized



lookup cost:

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

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

update cost:

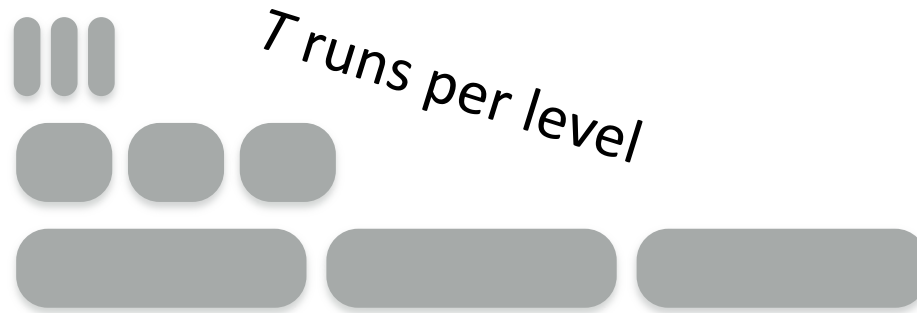
$$O(\log_T(N))$$

↑
levels

$$O(T \cdot \log_T(N))$$

↑ ↑
merges per level levels

Tiering write-optimized



Leveling read-optimized



lookup cost:

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

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

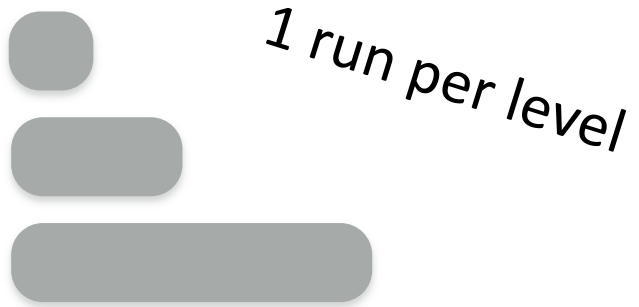
update cost:

$$O(\log_T(N))$$

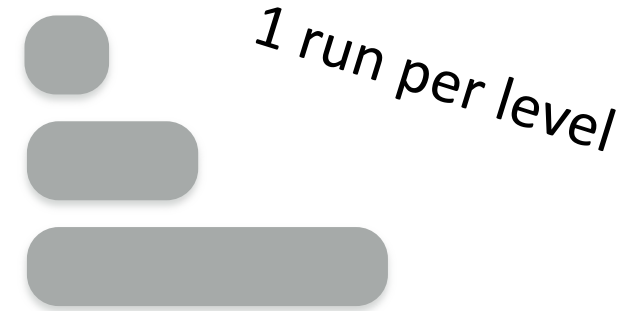
$$O(T \cdot \log_T(N))$$

for size ratio $T \gg 1$

Tiering
write-optimized



Leveling
read-optimized



lookup cost:

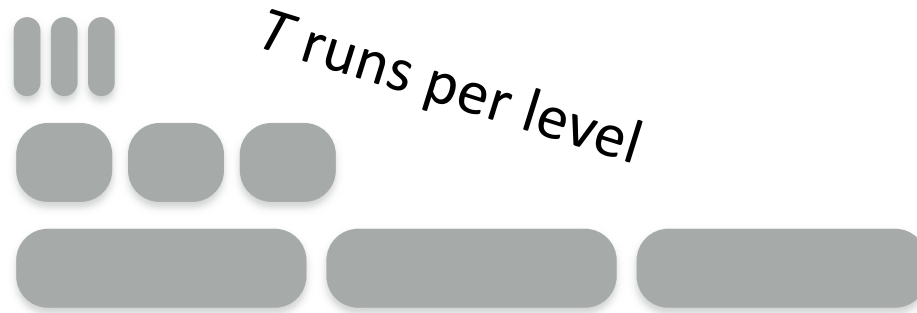
$$O(\log_T(N) \cdot e^{-M/N}) = O(\log_T(N) \cdot e^{-M/N})$$

update cost:

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

for size ratio $T \gg 1$

Tiering write-optimized



Leveling read-optimized



lookup cost:

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

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

update cost:

$$O(\log_T(N))$$

$$O(T \cdot \log_T(N))$$

for size ratio T \gg

Tiering
write-optimized

$O(N)$ runs per level



log

Leveling
read-optimized

1 run per level



sorted array

lookup cost:

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

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

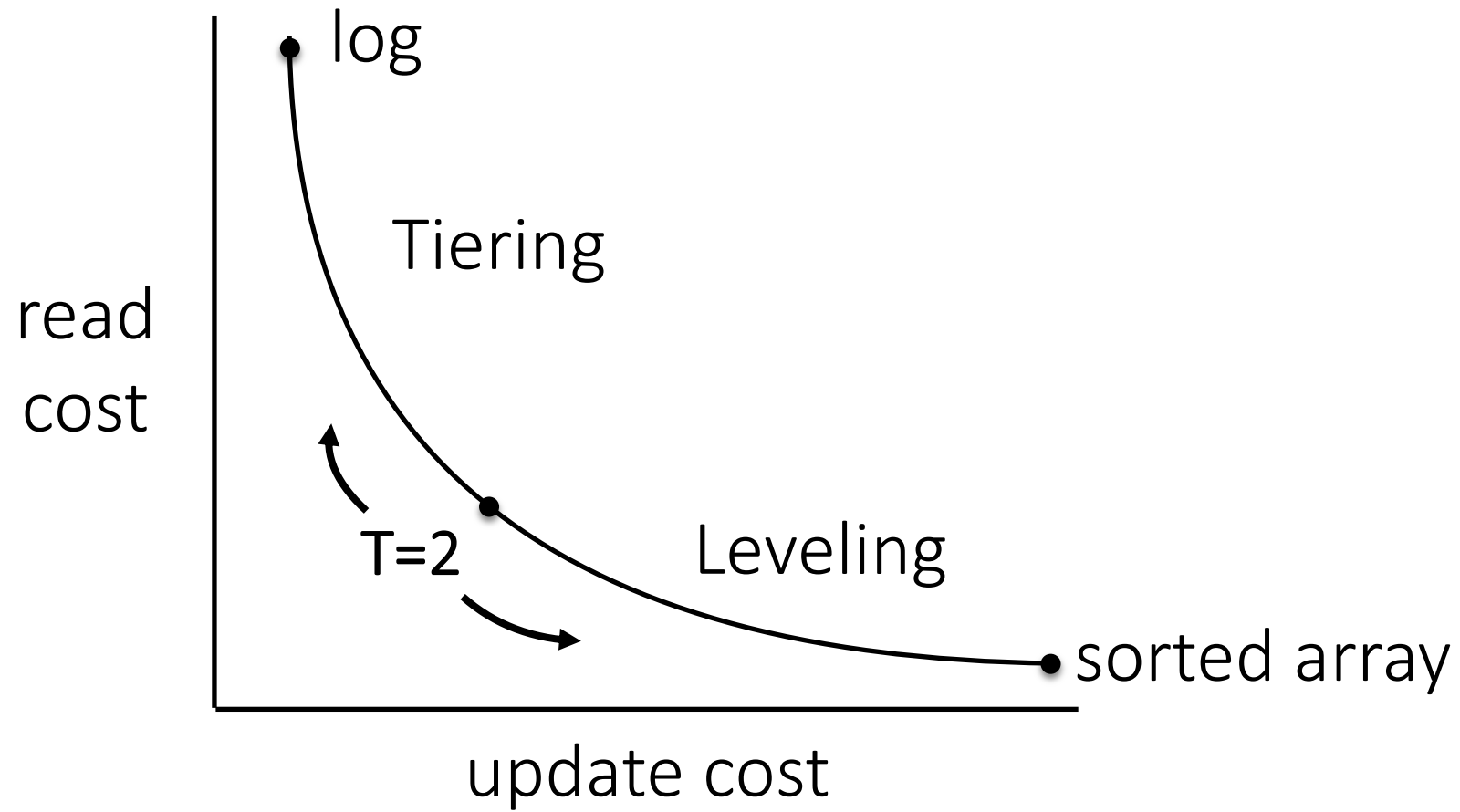
update cost:

$$O(\log_N(N)) = \mathbf{O(1)}$$

$$O(N \cdot \log_N(N)) = \mathbf{O(N)}$$

for size ratio T \gg

N
 \gg



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*?

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

buffer

Bloom
filters

fence
pointers



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*

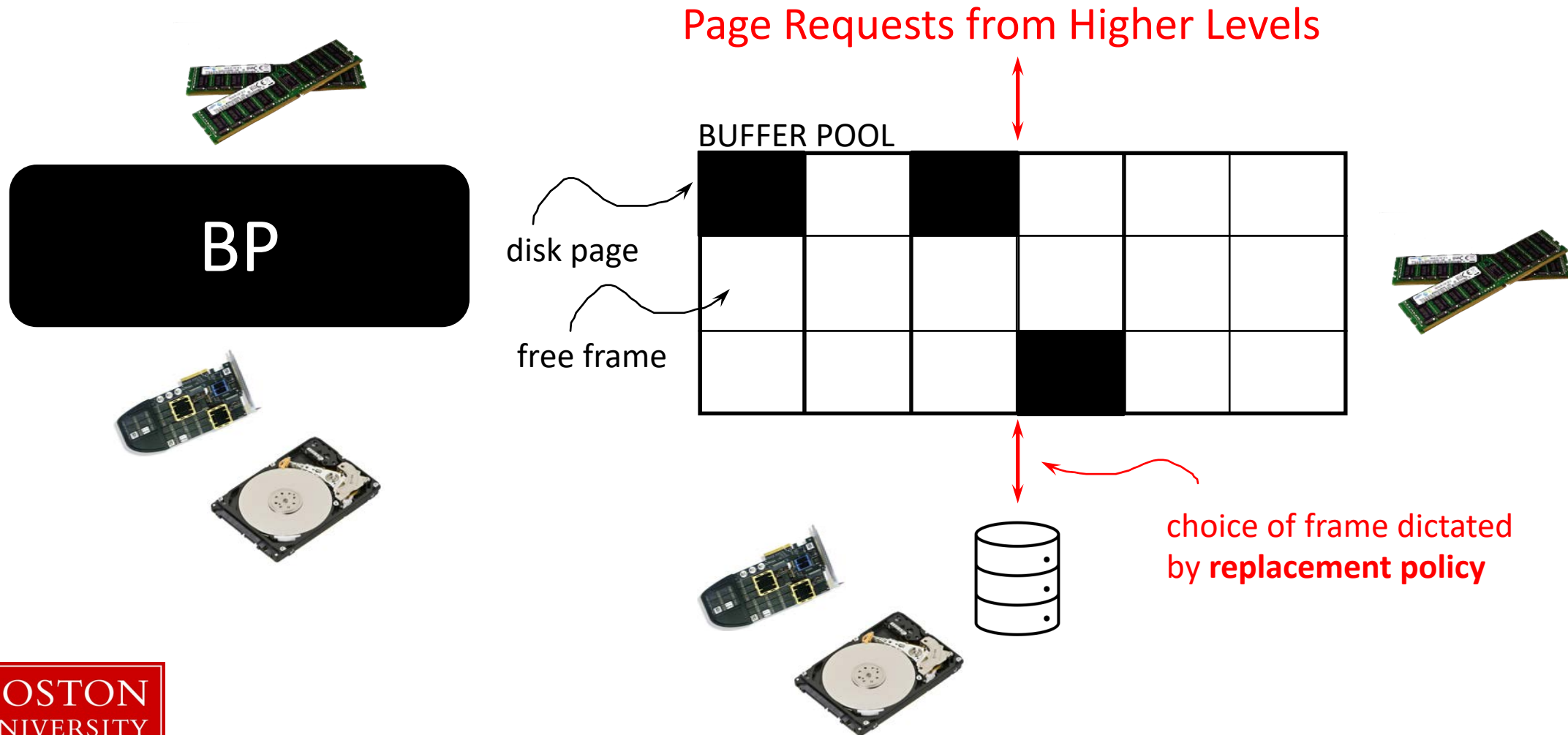
what if workload information is a bit wrong?

robust optimization (come and find me)

Asynchronous Bufferpool



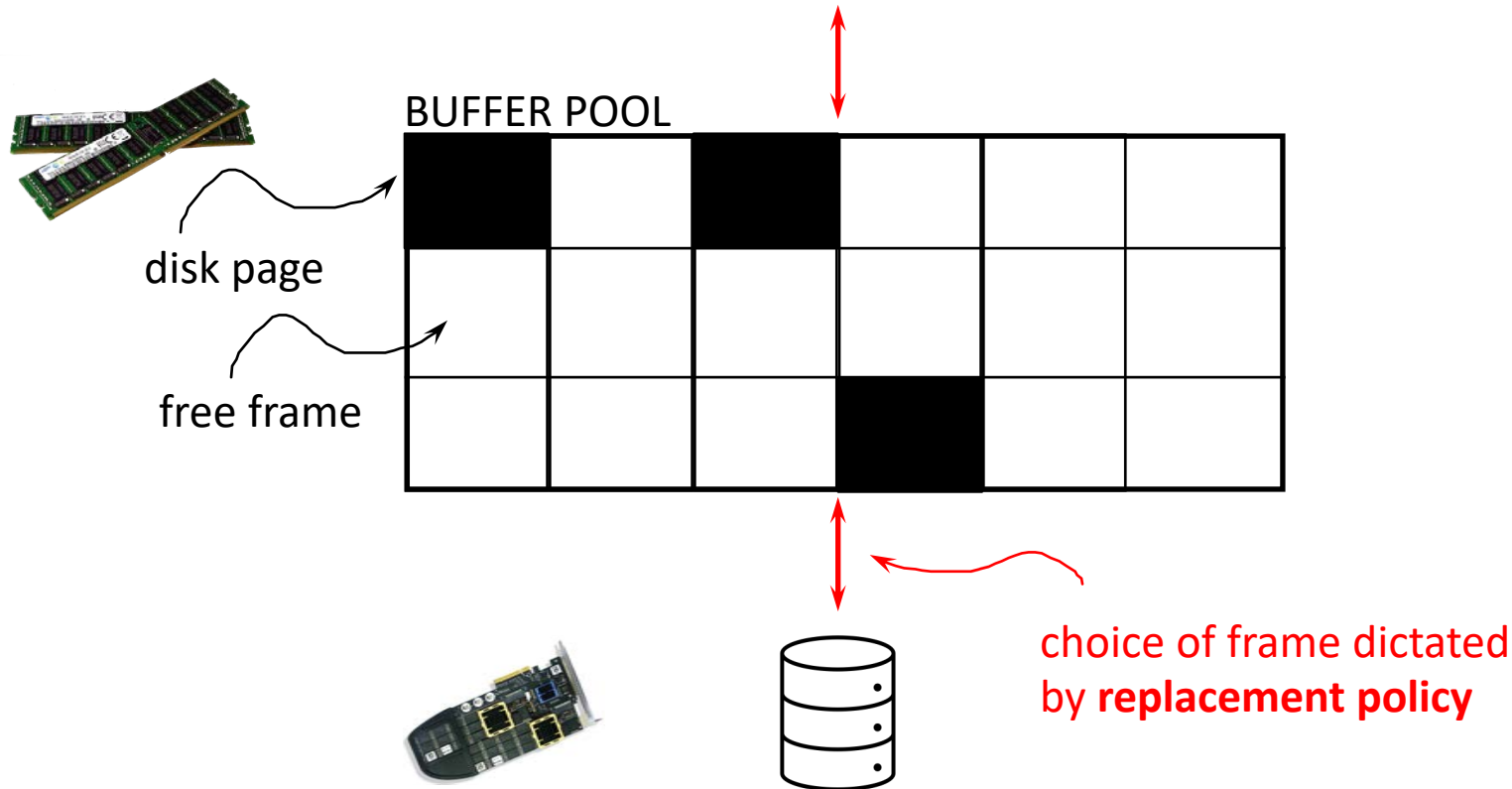
what is the bufferpool?



Systems Project: Bufferpool

Implementation of a bufferpool

Page Requests from Higher Levels



- **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**

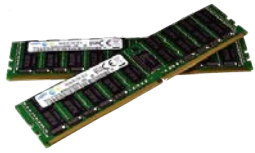
Core Idea: Eviction Policy

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

Asynchronous Bufferpool



what is the bufferpool?



BP



manages available memory
reads/writes from/to disk

what happens when full?

writes one page back and reads on page

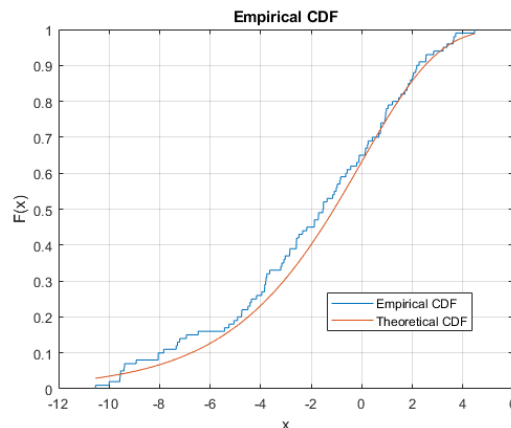
is this optimal?

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



$$position(val) = CDF(val) \cdot array_size$$



can you learn the CDF?
what is the best way to do so?
how to update that?

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

CS 561: Data Systems Architectures

Systems & Research Project

Prof. Manos Athanassoulis

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