

# *Dissecting, Designing, and Optimizing* **LSM-Based Data Stores**

*Subhadeep Sarkar*  *Manos Athanassoulis*

BOSTON UNIVERSITY

DiSC lab

# **L**og-**S**tructured **M**erge-tree

BOSTON
UNIVERSITY

# LSM-tree

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

1996

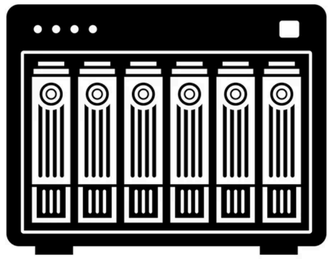
Patrick O'Neil[1], Edward Cheng[2]
Dieter Gawlick[3], Elizabeth O'Neil[1]
To be published: Acta Informatica

**LSM-tree**

**O'Neil *et al.***

1996

- good random writes
- good reads



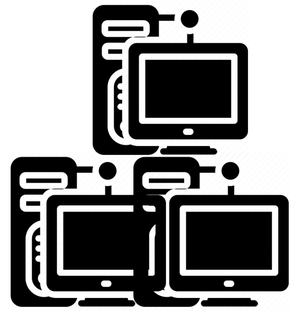
array
of discs

SSD wear-friendly

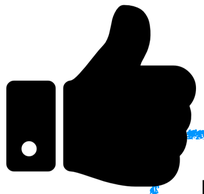
competitive rand. reads

fast ingestion

- poor ingestion perf.
- poor query perf.



commodity
hardware



**LSM-tree**
**O'Neil *et al.***

Bigtable

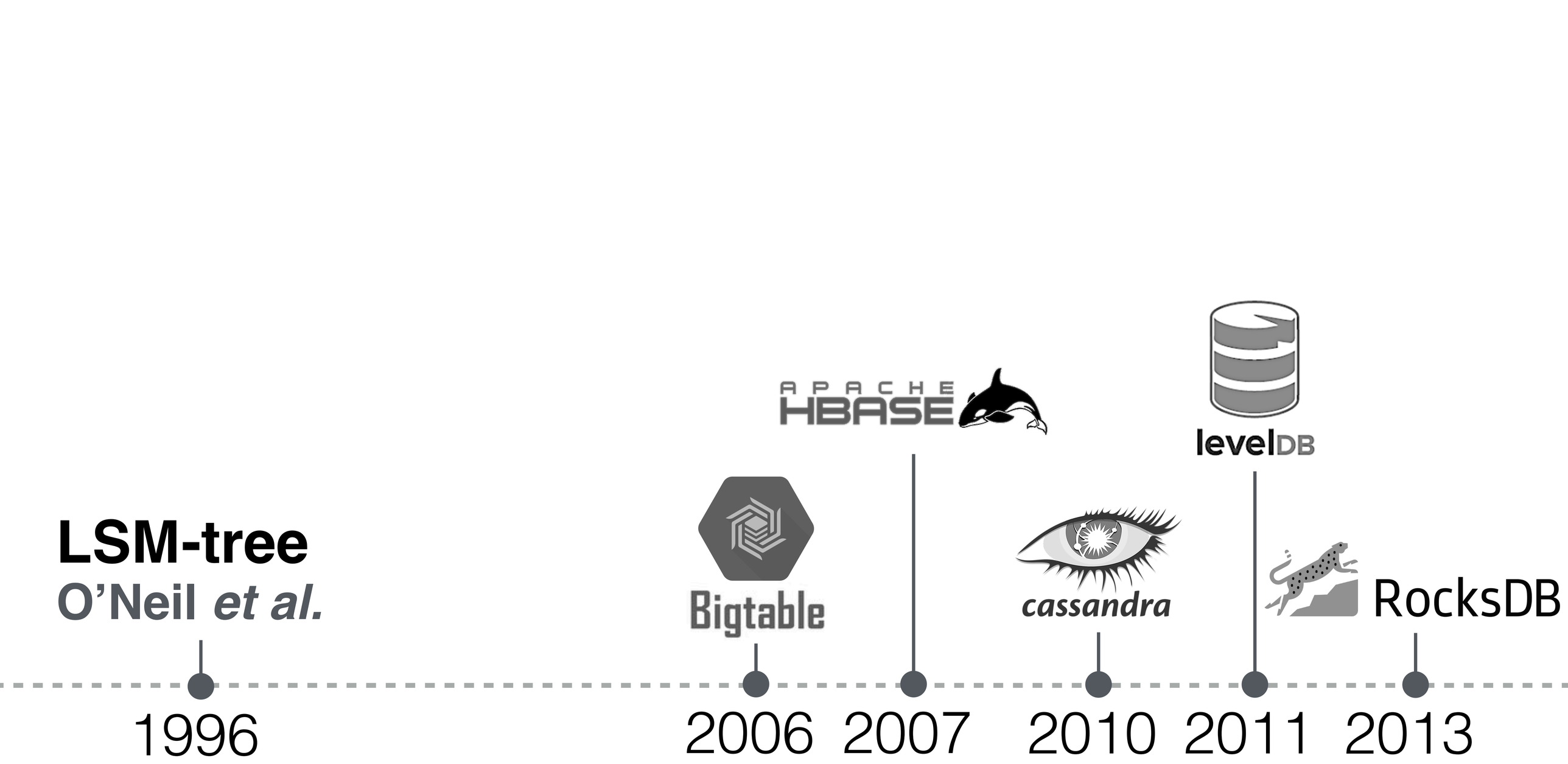
1980s

1996

2006

a decade

BOSTON UNIVERSITY


LSM-tree
O'Neil et al.
1996
Bigtable
2006
APACHE HBASE
2007
cassandra
2010
levelDB
2011
RocksDB
2013

# LSM-tree

NoSQL



relational


time-series

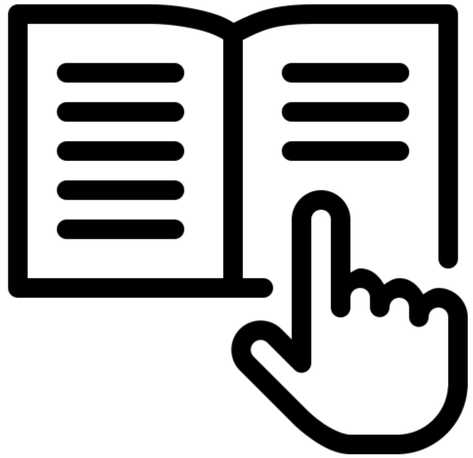
2022

# Why **LSM** ?

fast ingestion

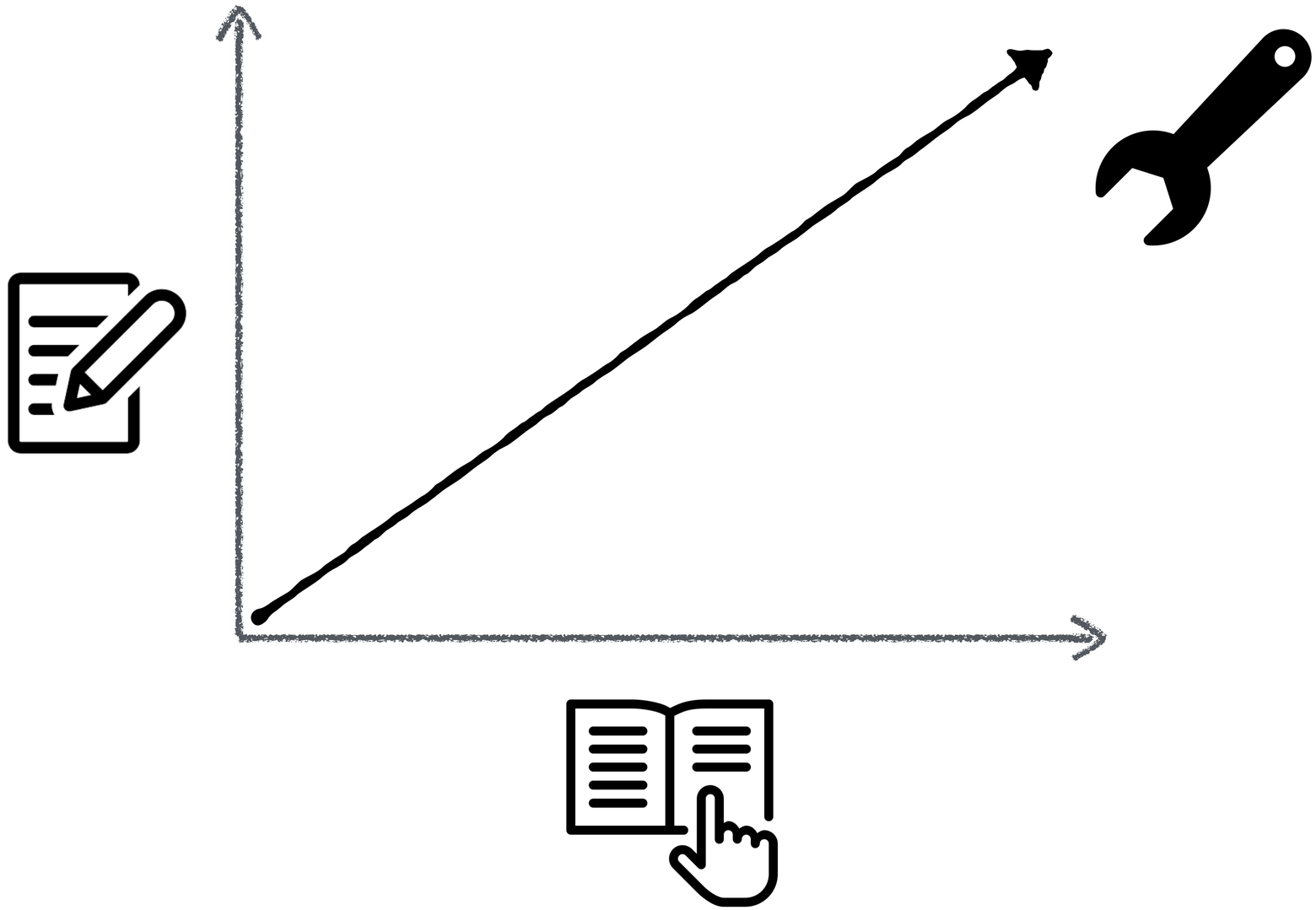
# Why **LSM** ?

fast ingestion



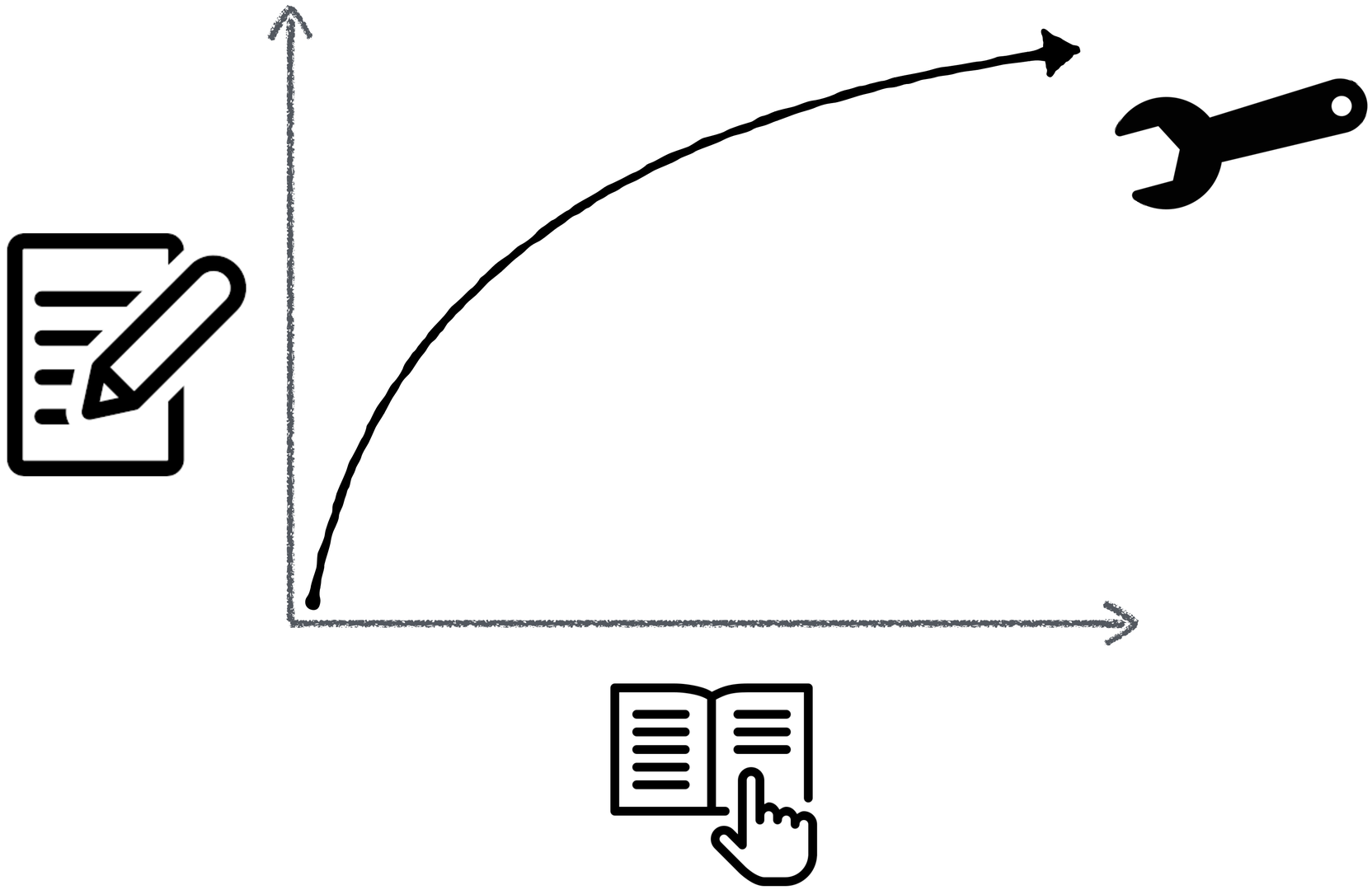
competitive reads

# Why **LSM** ?



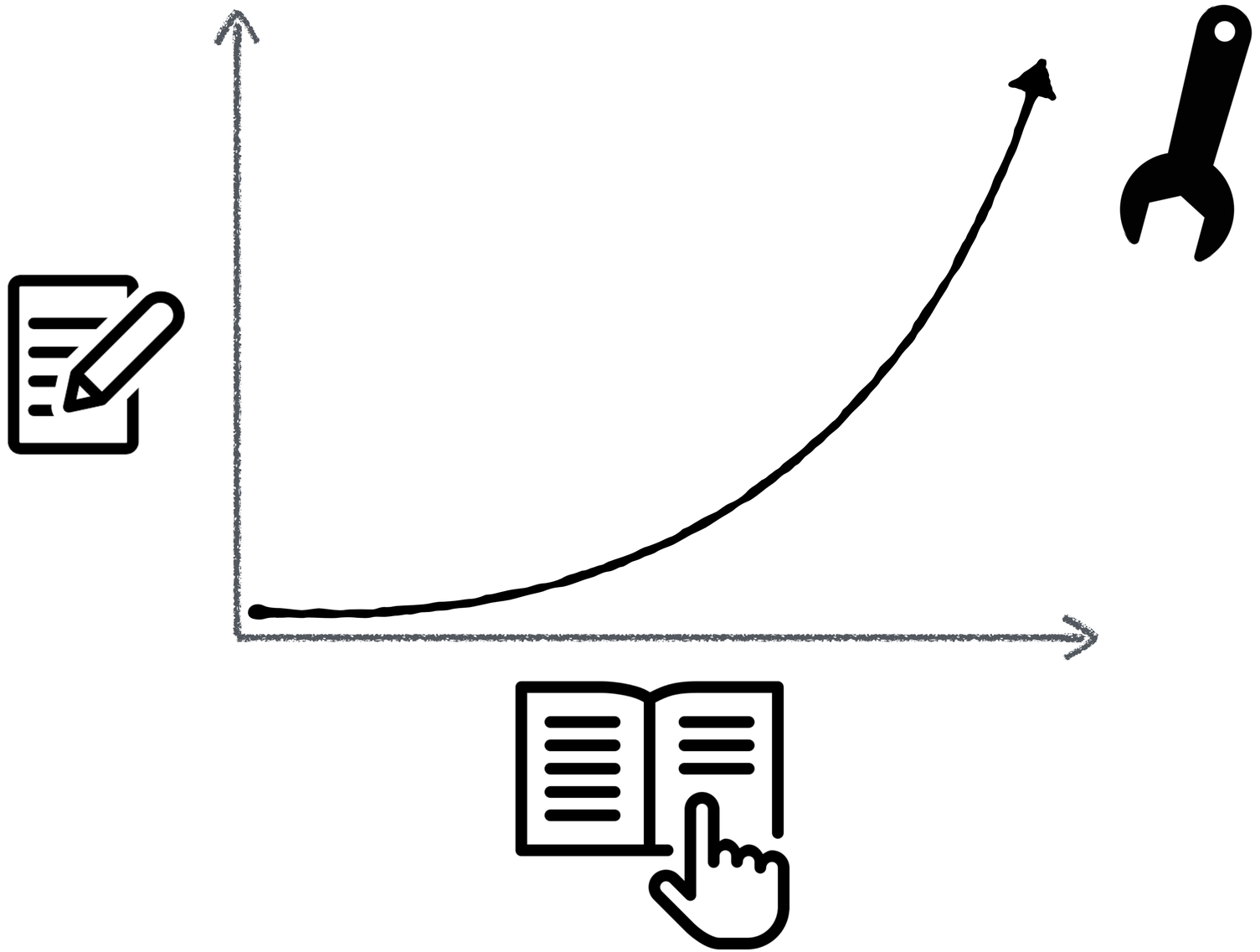
BOSTON UNIVERSITY

# Why **LSM** ?



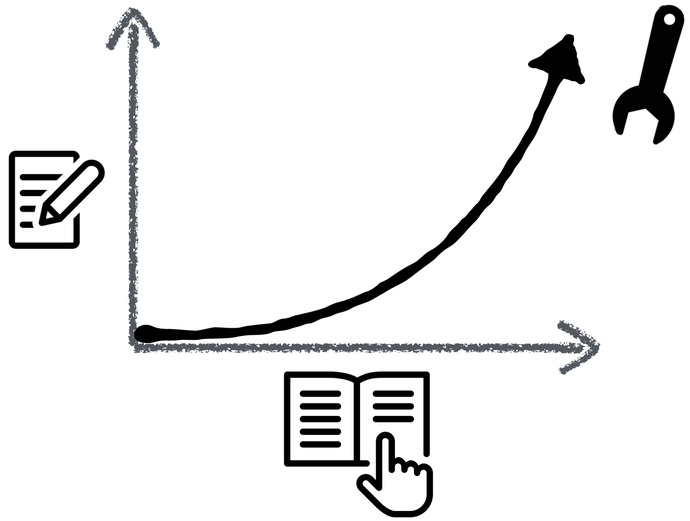
BOSTON UNIVERSITY

# Why **LSM** ?

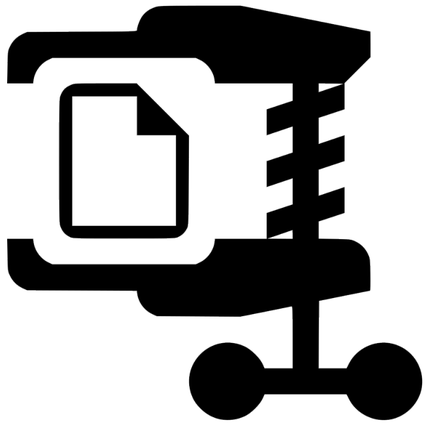


# Why **LSM** ?

fast writes

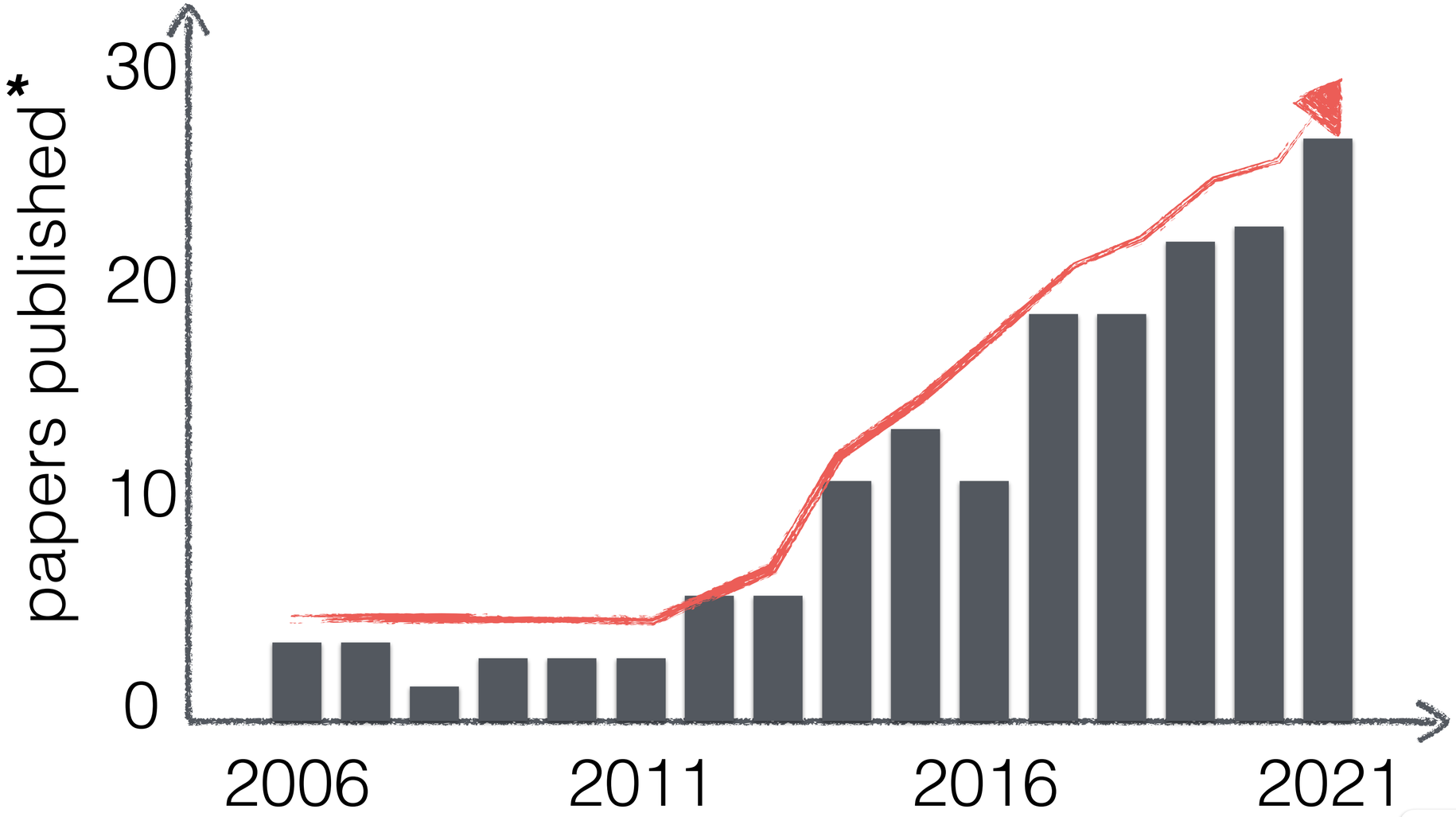


tunable read-write performance



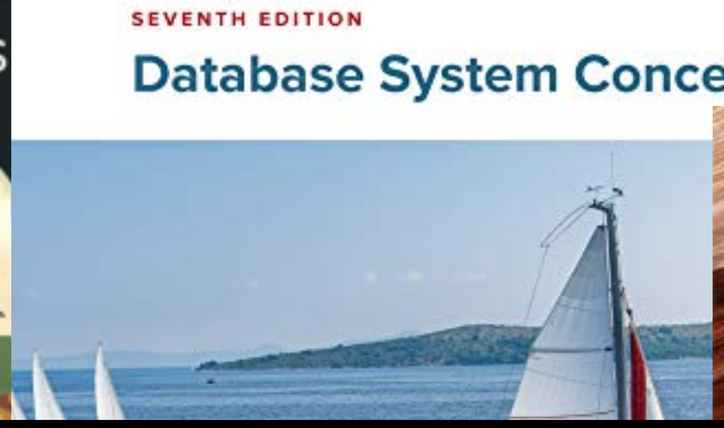
good space utilization

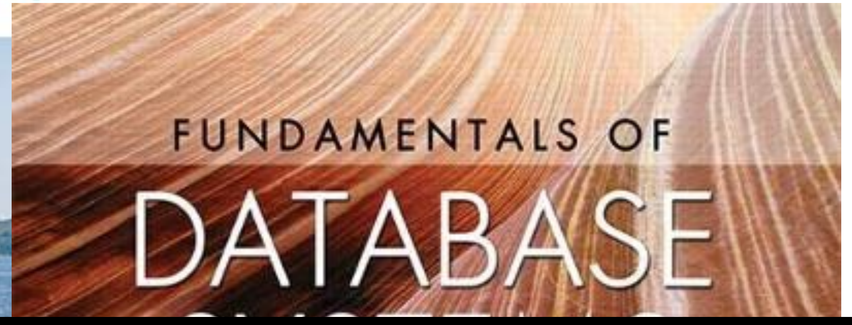
# Research Trend



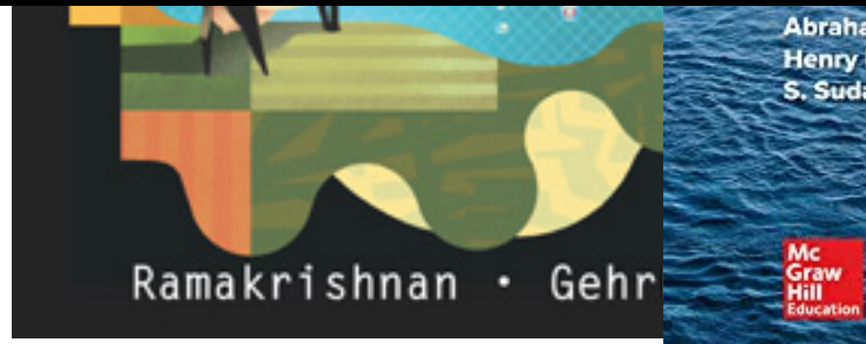

BOSTON UNIVERSITY

* data from DBLP

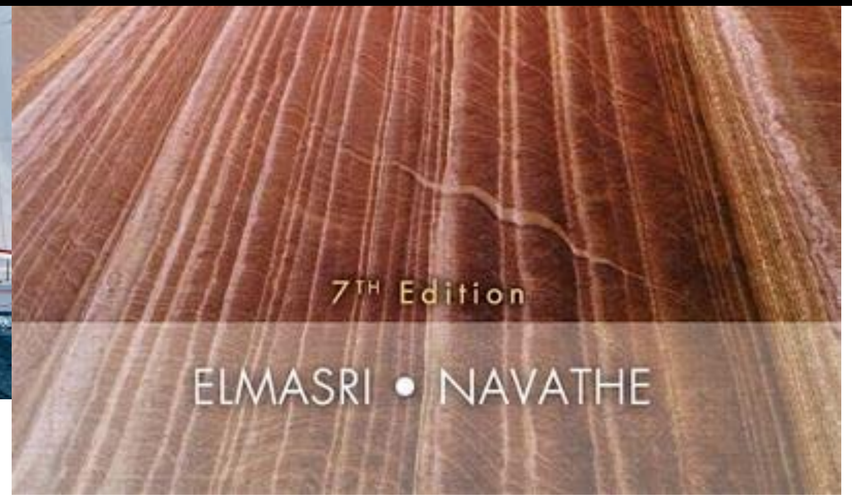




# No Textbook on LSMs !!








BOSTON
UNIVERSITY

# No Textbook on LSMs !!

**explore** the LSM paradigm

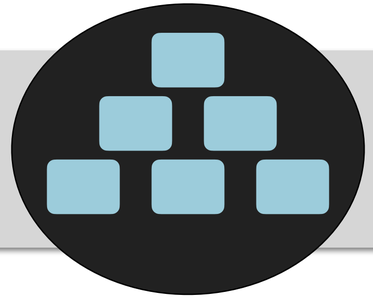
**construct** LSM design space

provide useful **insights**

BOSTON UNIVERSITY

# Outline

Part 1: **LSM Basics**

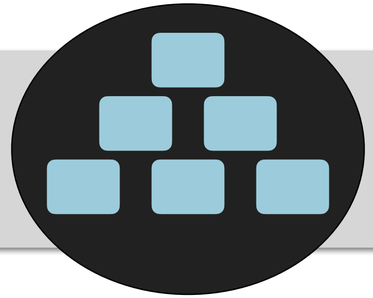


Part 2: **Optimizing Ingestion in LSMs**

Part 3: **Navigating the LSM Design Space**

# Outline

## Part 1: **LSM Basics**



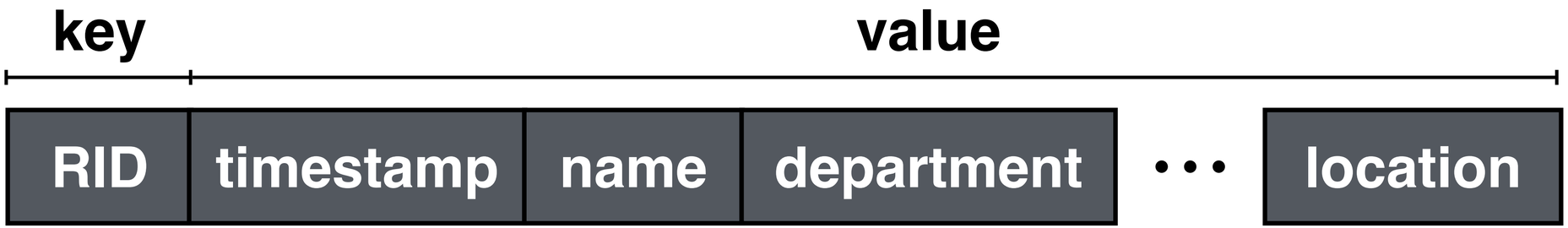
## Part 2: **Optimizing Ingestion in LSMs**



## Part 3: **Navigating the LSM Design Space**

# LSM **Basics**

key-value pairs

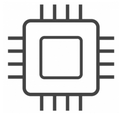


# LSM **Basics**

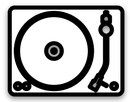
key-value pairs

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

# LSM **Basics**



buffer



level 1

level 2

level 3

level 4

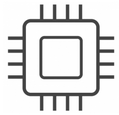
size ratio: T

BOSTON UNIVERSITY

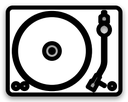
# Buffering ingestion

put(6)

put(2)



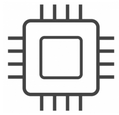
buffer



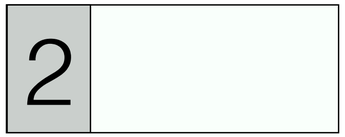
# Buffering ingestion

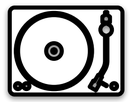
put(1)

put(6)



buffer



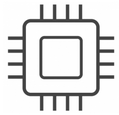



BOSTON UNIVERSITY

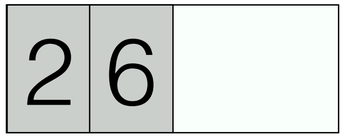
# Buffering ingestion

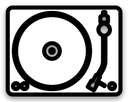
put(4)

put(1)



buffer

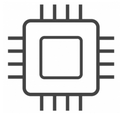


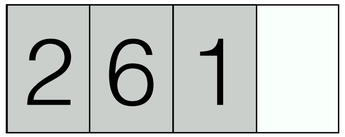
BOSTON UNIVERSITY

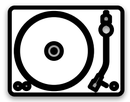
# Buffering ingestion

put(4)

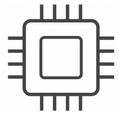


buffer

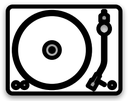


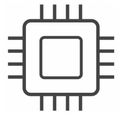
# Buffering ingestion



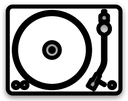
buffer | 2 | 6 | 1 | 4 |



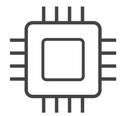
# Buffering ingestion



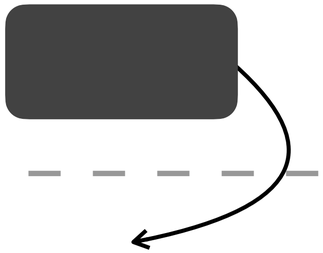
buffer | 1 | 2 | 4 | 6 |

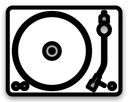


# Buffering ingestion



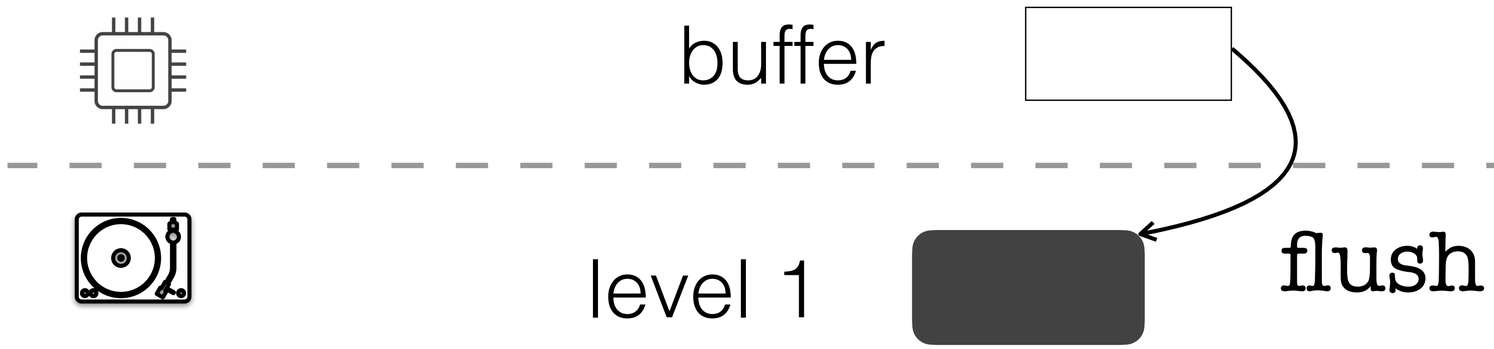
buffer





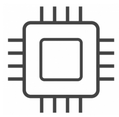
# Buffering ingestion

- low ingestion cost
- fast temporal reads



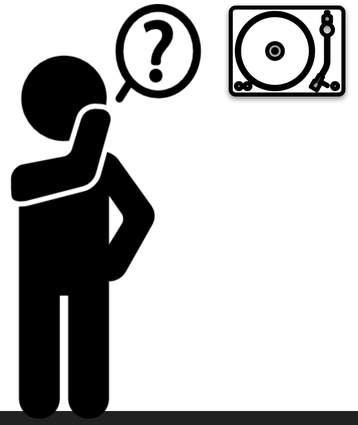
# Immutable files on storage

- compact storage
- good ingestion throughput



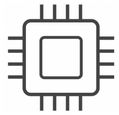
buffer

level 1



How do we update data?

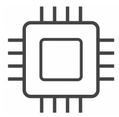
put(6)



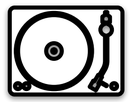
buffer

level 1 6

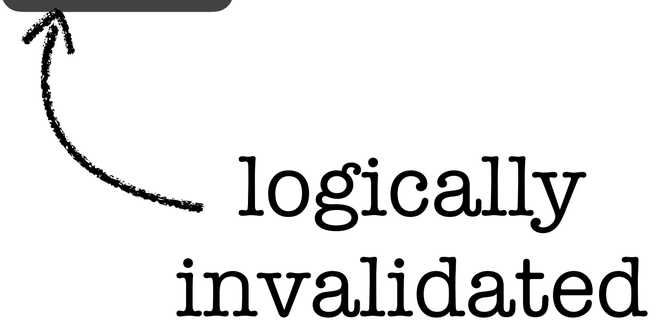
BOSTON UNIVERSITY

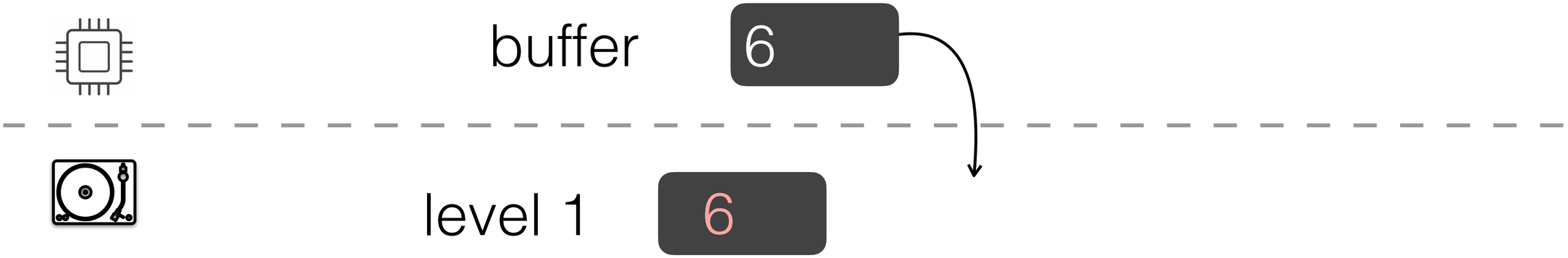


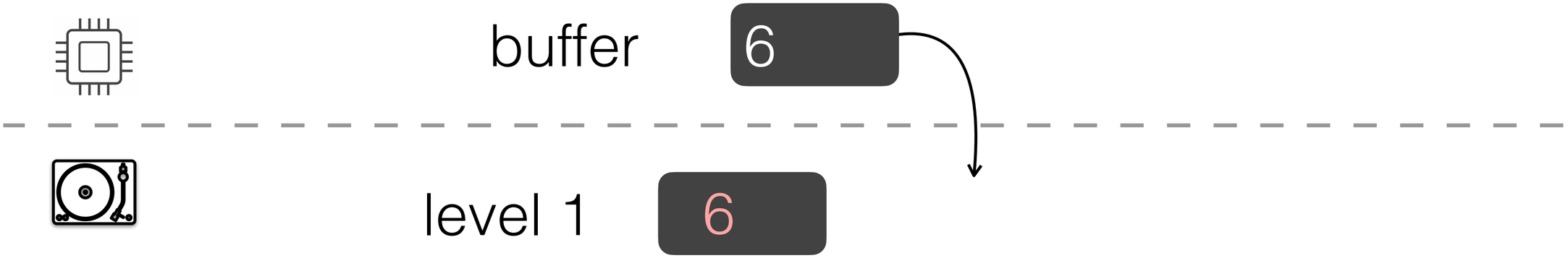
buffer

level 1

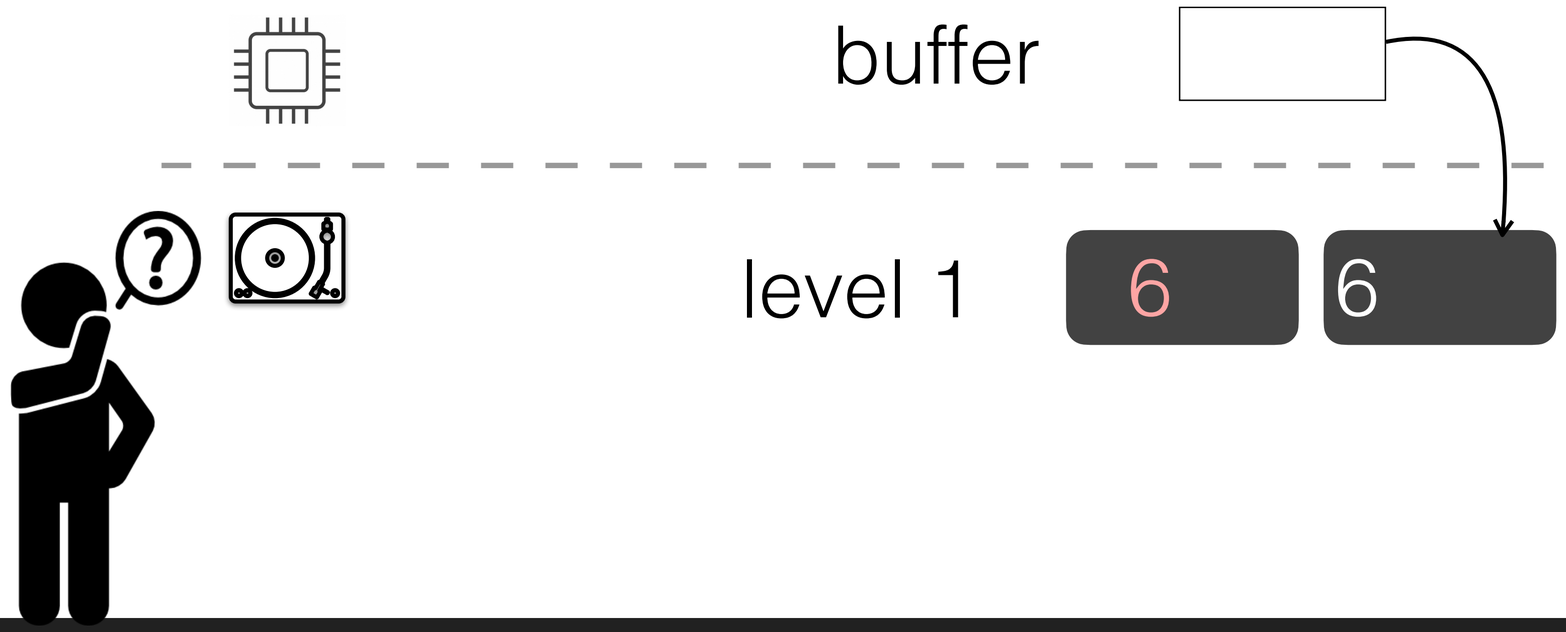





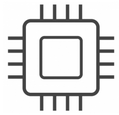


# Out-of-place updates

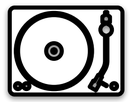
- fast ingestion
- space amplification
- slow reads




How do we reduce this space amplification?



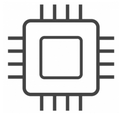
buffer



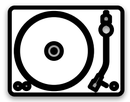
level 1

6

6



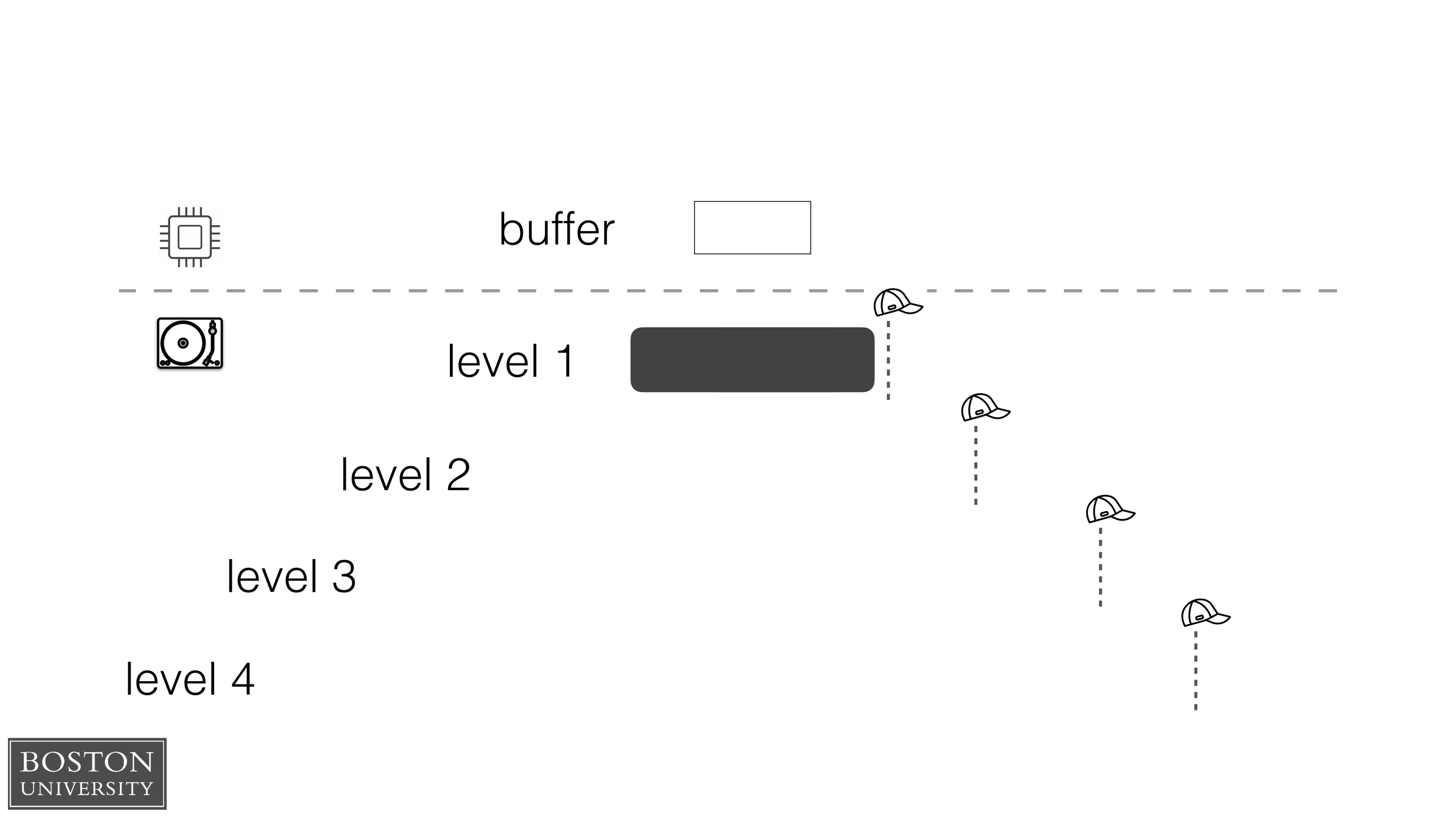
buffer

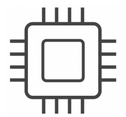


level 1

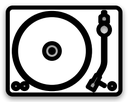
6

compaction

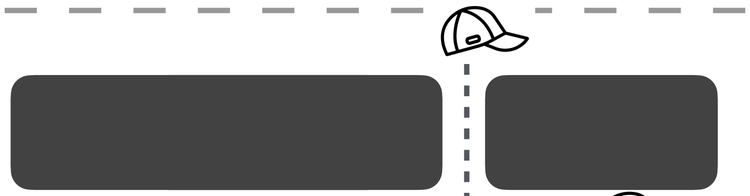




buffer



level 1

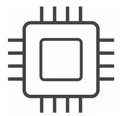


level 2

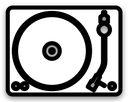
level 3

level 4

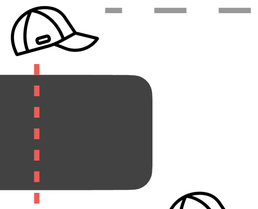
BOSTON UNIVERSITY



buffer



level 1

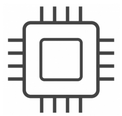


level 2

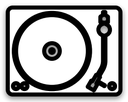
level 3

level 4

BOSTON UNIVERSITY



buffer



level 1

level 2

level 3

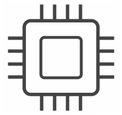
level 4

BOSTON UNIVERSITY

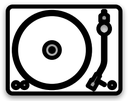
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio

# Periodic compactions

- low space amplification
- makes reads better
- adds write amplification



buffer $P \cdot B$



level 1 $P \cdot B \cdot T$

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

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

level 4 $P \cdot B \cdot T^4$

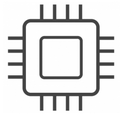
capacity (entries)

BOSTON UNIVERSITY

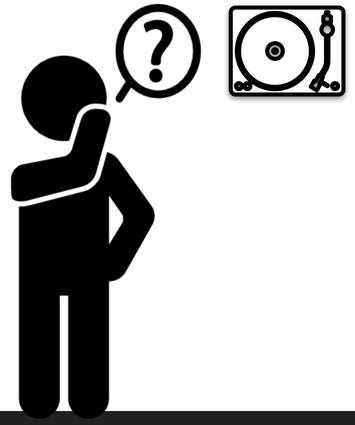
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio

# Periodic compactions

- low space amplification
- makes reads better
- adds write amplification



buffer $P \cdot B$



level 1

$P \cdot B \cdot T$

level 2

$P \cdot B \cdot T^2$

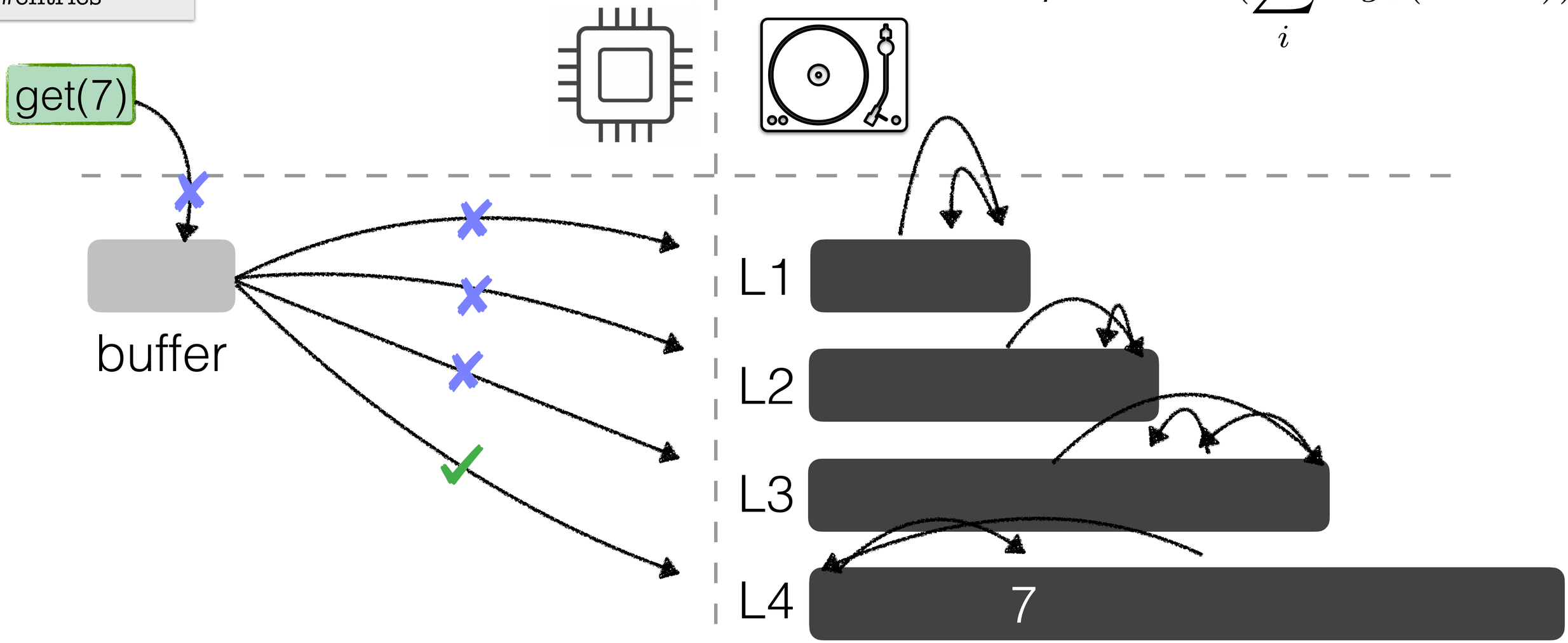
capacity
(entries)

## How about queries?

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

# Cost analysis

w/o F&I: $\mathcal{O}(\sum_i log_2(P \cdot T^i))$

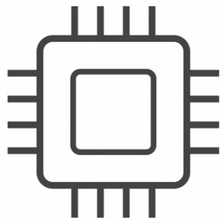


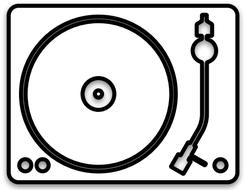
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries

# Cost analysis

w/o F&I: $\mathcal{O}(\sum_i log_2(P \cdot T^i))$

w/ index: $\mathcal{O}(log_T N/(P \cdot B))$





get(7)

buffer

L1

L2

L3

L4

7

fence pointers

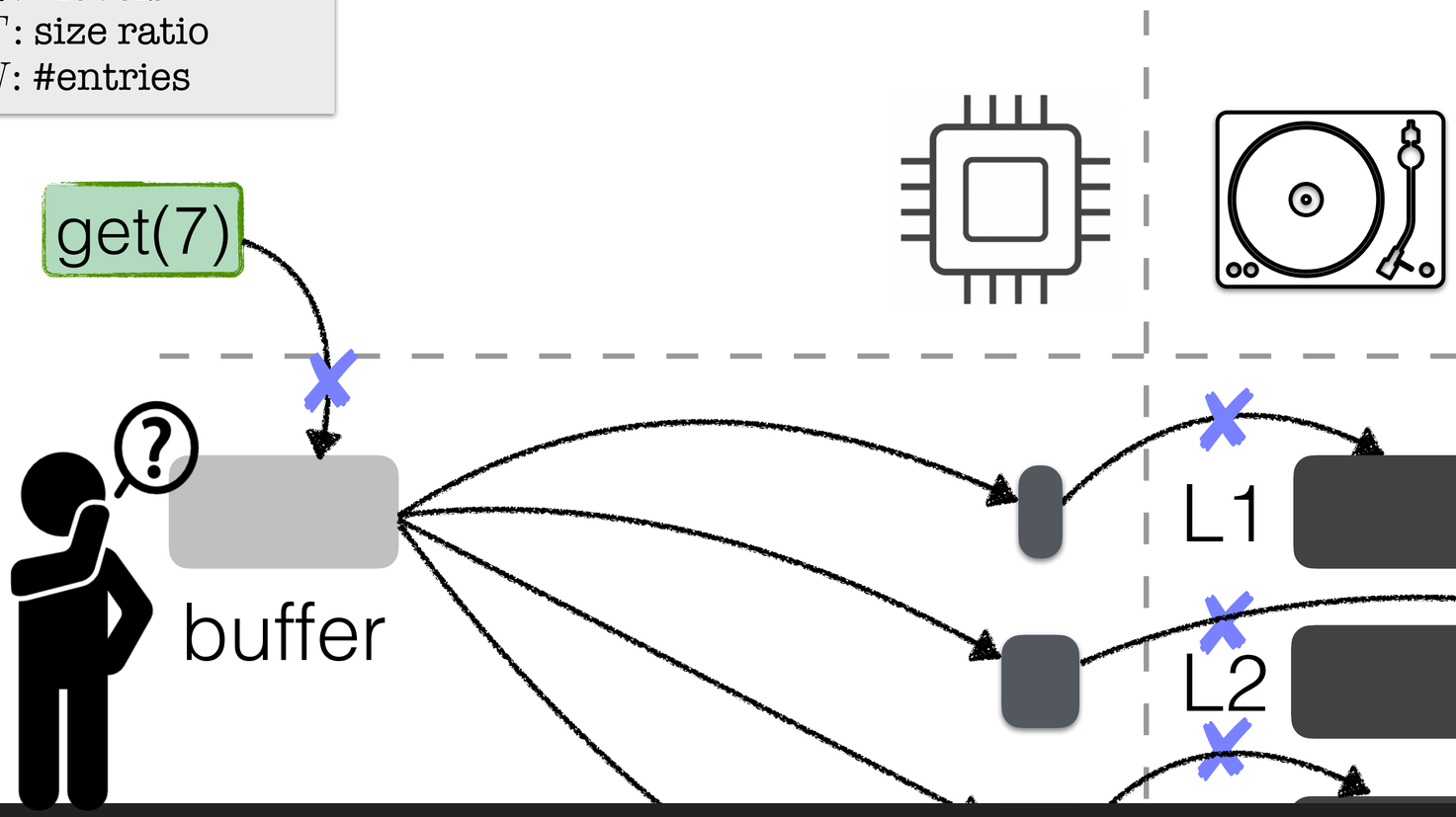
BOSTON UNIVERSITY

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

# Cost analysis

w/o F&I: $\mathcal{O}(\sum_{i} log_2(P \cdot T^i))$

w/ index: $\mathcal{O}(L)$




How to avoid unnecessary I/Os?

fence pointers

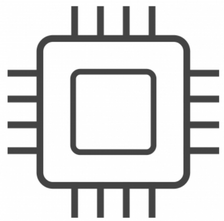
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

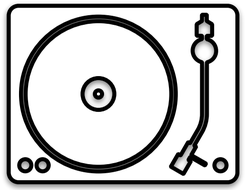
# Cost analysis

w/o F&I: $\mathcal{O}(\sum_{i} log_2(P \cdot T^i))$

w/ index: $\mathcal{O}(L)$

w F&I: $\mathcal{O}(\phi \cdot L)$





get(7)

buffer

L1

L2

L3

L4

7

$\phi = e^{-\frac{m_{BF}}{N} \cdot ln2^2}$

Bloom filters

fence pointers

BOSTON UNIVERSITY

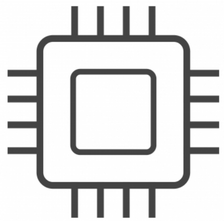
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

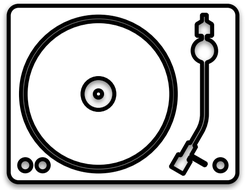
# Cost analysis

w/o F&I: $\mathcal{O}(\sum_{i} log_2(P \cdot T^i))$

w/ index: $\mathcal{O}(L)$

w F&I: $\mathcal{O}(\phi \cdot L)$





get(7)

buffer

L1

L2

L3

L4

7

$\phi = e^{-\frac{m_{BF}}{N} \cdot ln2^2} << 1$

Bloom filters

fence pointers

BOSTON UNIVERSITY

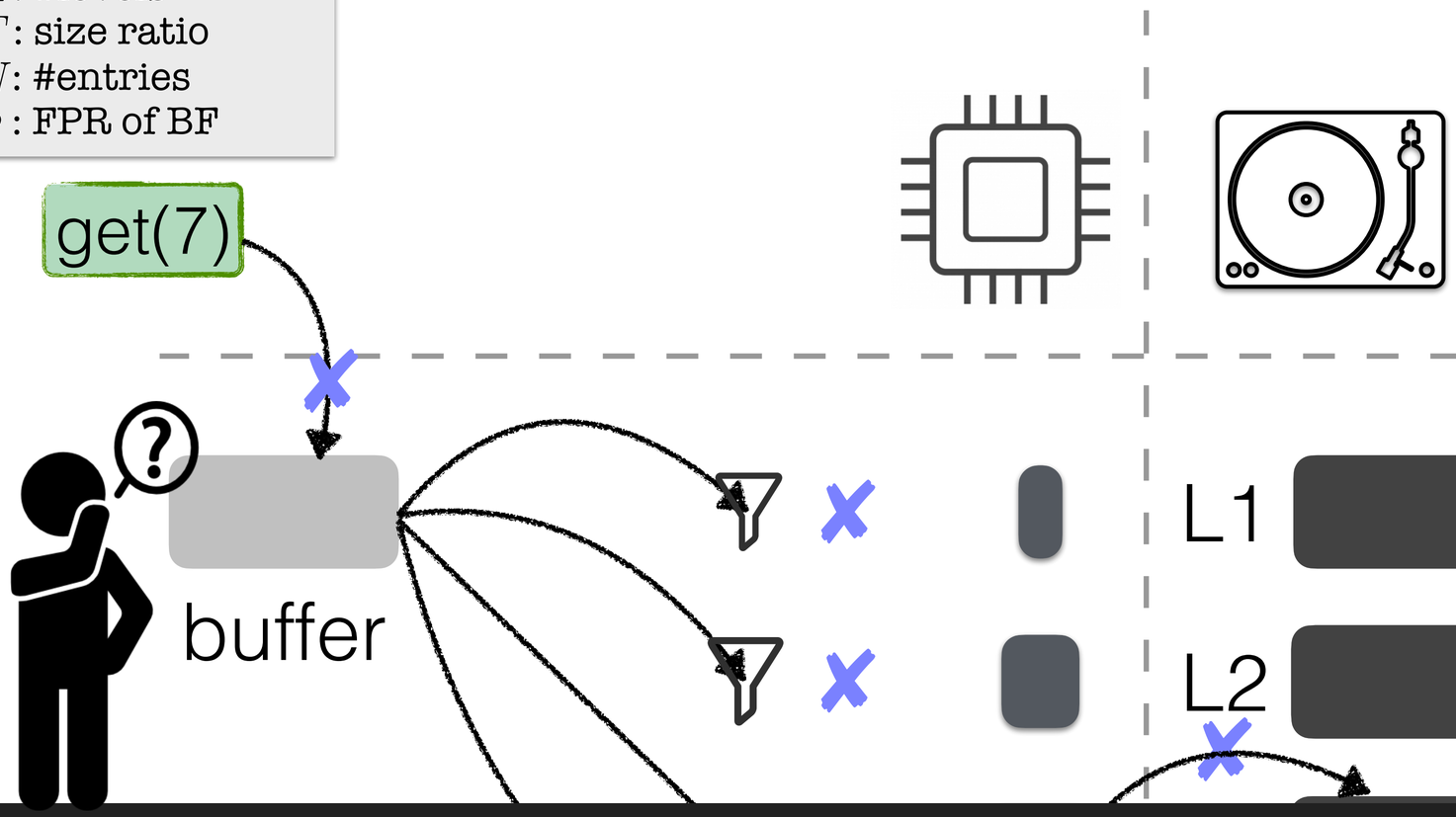
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

# Cost analysis

w/o F&I: $\mathcal{O}(\sum_{i} log_2(P \cdot T^i))$

w/ index: $\mathcal{O}(L)$

w F&I: $\mathcal{O}(\phi \cdot L)$




$$\phi = e^{-\frac{m_{BF}}{N} \cdot ln2^2} << 1$$

How to manage memory?

Bloom filters

fence pointers

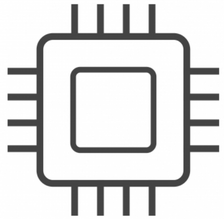
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

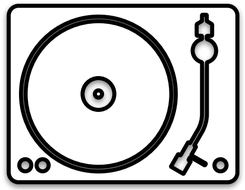
# Cost analysis

w/o F&I: $\mathcal{O}(\sum_{i} log_2(P \cdot T^i))$

w/ index: $\mathcal{O}(L)$

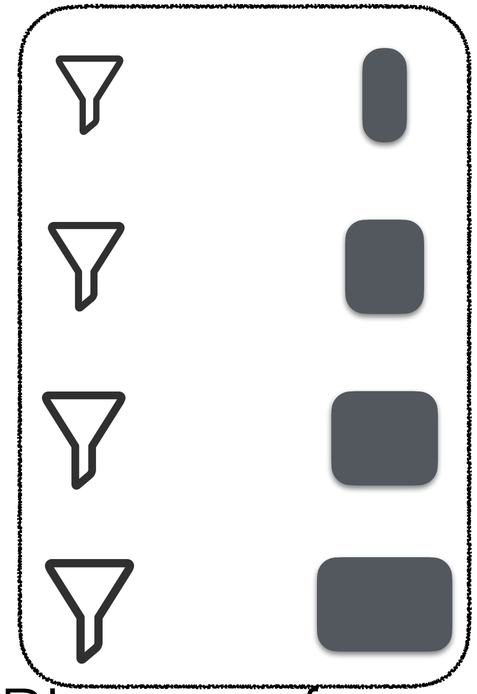
w F&I: $\mathcal{O}(\phi \cdot L)$





buffer

block cache



Bloom filters

fence pointers

L1

L2

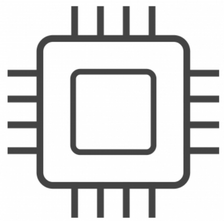
L3

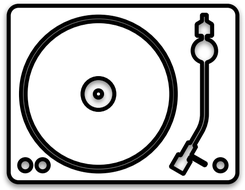
L4

$$\phi = e^{-\frac{m_{BF}}{N} \cdot ln2^2} << 1$$

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

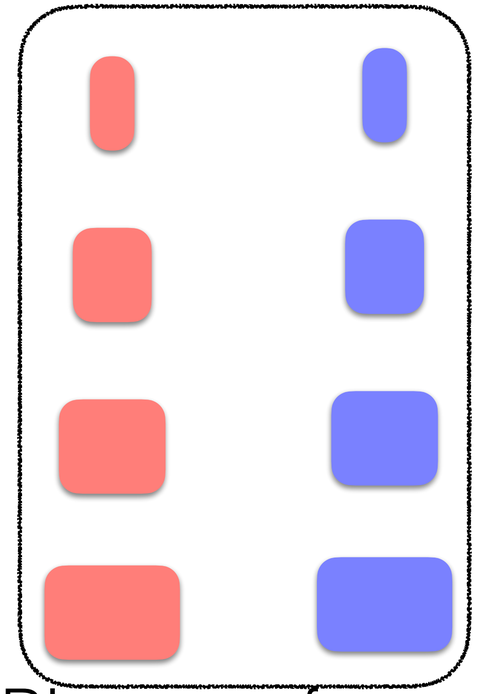
# Block Cache





buffer

block cache



Bloom filters

fence pointers

L1

L2

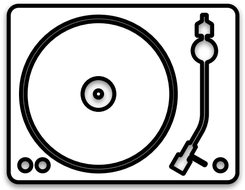
L3

L4

BOSTON UNIVERSITY

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$: FPR of BF

# Block Cache

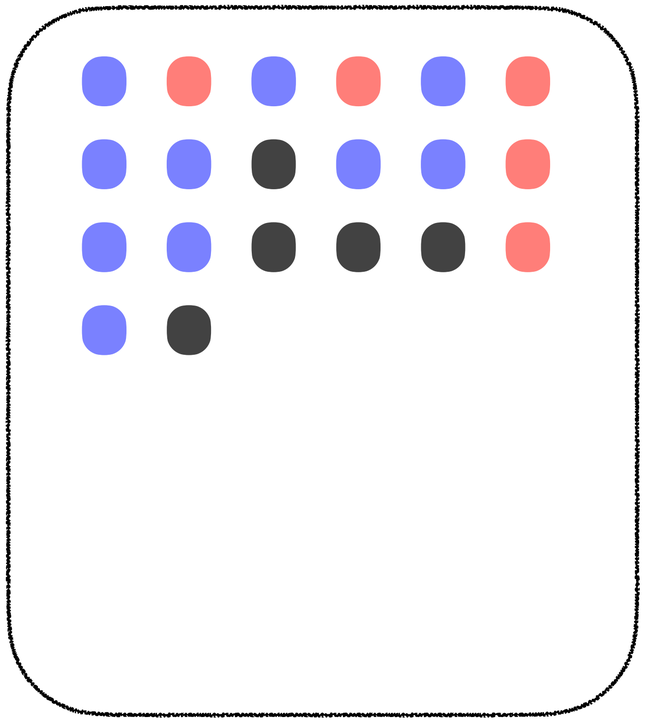


Bloom filters

fence pointers

get(7)

buffer



L1

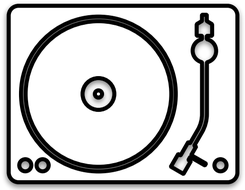
L2

L3

L4

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$: FPR of BF

# Block Cache

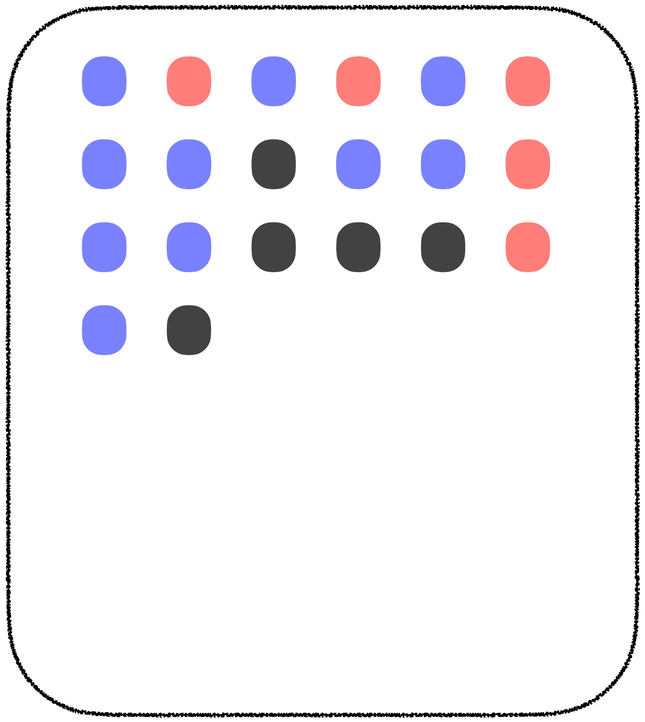


Bloom filters

fence pointers

get(7)

buffer



L1

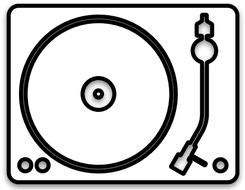
L2

L3

L4

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$: FPR of BF

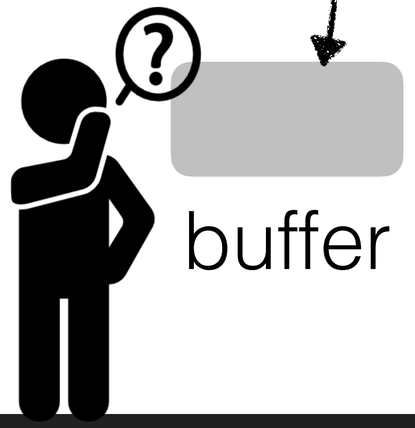
# Block Cache



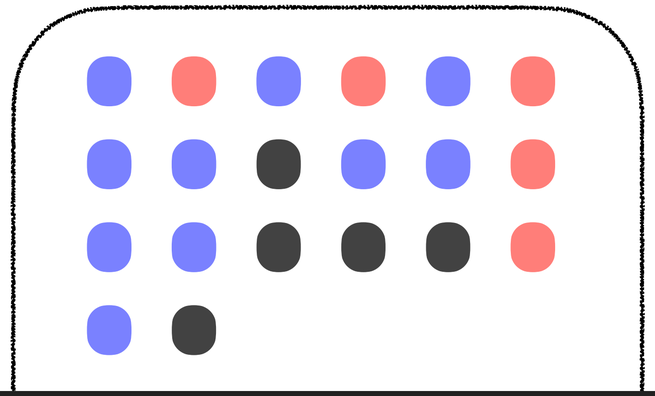
Bloom filters

fence pointers

get(7)



buffer



L1

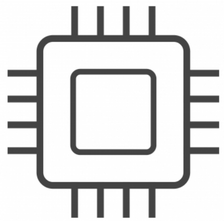
L2

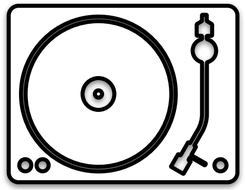
What about range queries?

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

$s$ : selectivity LRQ

# Range Queries





buffer

Bloom filters

fence pointers

L1

L2

L3

L4

BOSTON UNIVERSITY

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$: FPR of BF

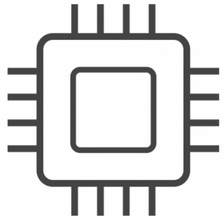
$s$ : selectivity LRQ

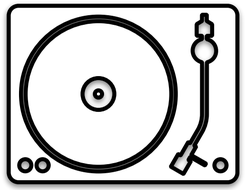
# Range Queries

## Cost analysis

long range: $\mathcal{O}(s \cdot N/B)$

get(9,90)





buffer

Bloom filters

fence pointers

L1

L2

L3

L4

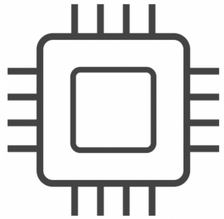
BOSTON UNIVERSITY

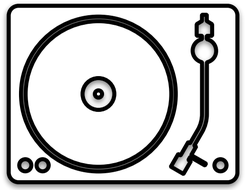
$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$: FPR of BF

$s$ : selectivity SRQ

get(9,15)

# Range Queries





# Cost analysis

long range: $\mathcal{O}(s \cdot N/B)$

short range: $\mathcal{O}(L)$

buffer

L1

L2

L3

L4

Bloom filters

fence pointers

BOSTON UNIVERSITY

# More **Read Optimizations**

## filter structures

Better performance

Chucky — DayanSIGMOD21

Less memory usage

Ribbon filter — DillingerArxiv21

Less CPU usage

Shared Hashing — ZhuDAMON21

Support range queries

SuRF — ZhangSIGMOD18

Rosetta — LuoSIGMOD20

## indexes

Better performance

Learned — DaiOSDI20

Less memory usage

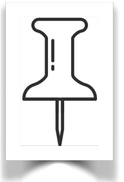
FerraginaVLDB20

## block cache

Better performance

Less memory usage

Prefetching — HuangVLDB20



Pinning — CallaghanSD18

Priority — CallaghanSD16

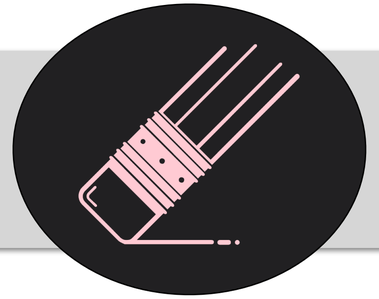
BOSTON UNIVERSITY

# Outline

Part 1: **LSM Basics**

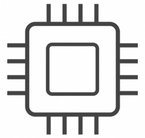


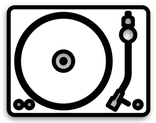
Part 2: **Optimizing Ingestion in LSMs**



Part 3: **Navigating the LSM Design Space**

# **Buffer** Optimizations





buffer

L1

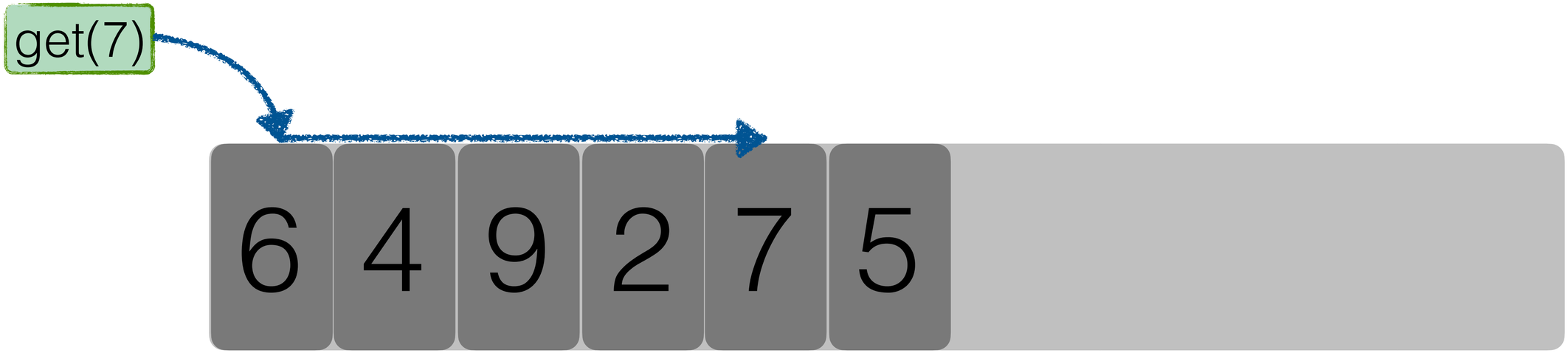
L2

L3

L4

$P$: pages in buffer
$B$: entries/page

# Buffer Implementation: **vector**



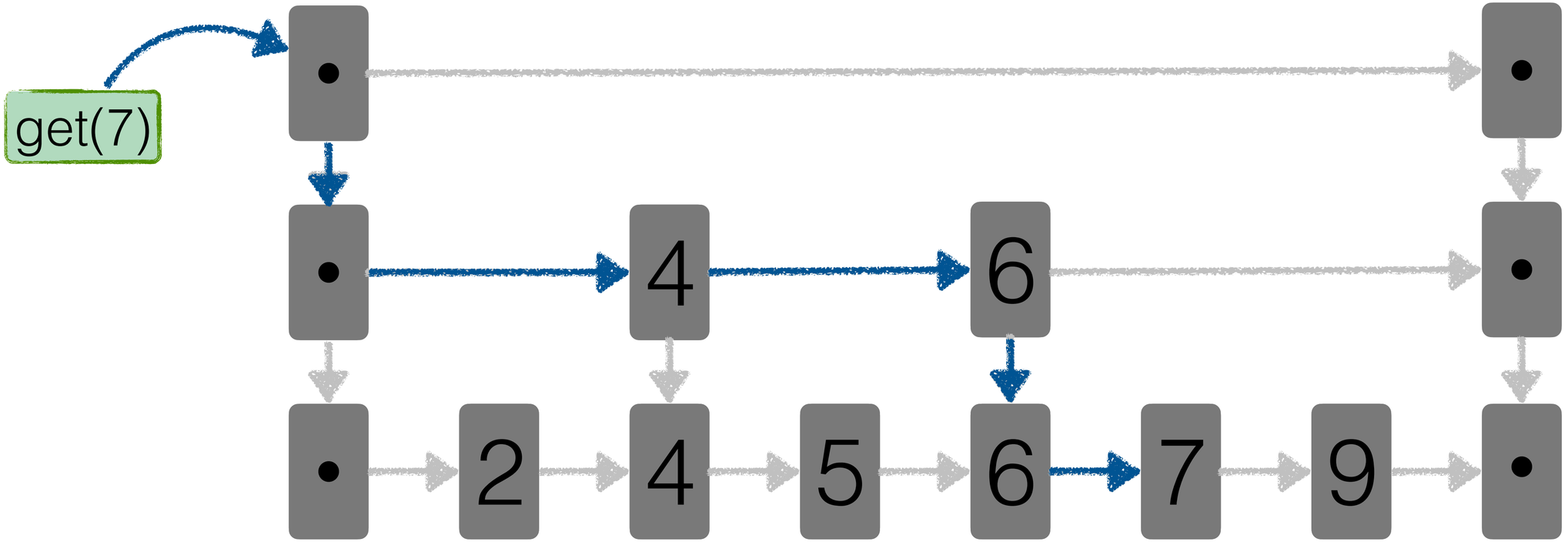

- great for ingestion-heavy w/l
- no extra space needed
- expensive points queries

ingestion cost: $\mathcal{O}(1)$

space complexity: $\mathcal{O}(P \cdot B)$

point query cost: $\mathcal{O}(P \cdot B)$

# Buffer Implementation: **skiplist**




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

$P$: pages in buffer
$B$: entries/page

# Buffer Implementation

| | **vector** | **skiplist** | **hashmap** |
|---|---|---|---|
| **ingestion** cost | $\mathcal{O}(1)$ | $\mathcal{O}(log(P \cdot B))$ | $\mathcal{O}(1)$ |
| **space** complexity | $\mathcal{O}(P \cdot B)$ | $\mathcal{O}(P \cdot B)$ | $\mathcal{O}(P \cdot B)$ |
| **point query** cost | $\mathcal{O}(P \cdot B)$ | $\mathcal{O}(log(P \cdot B))$ | $\mathcal{O}(1)$ |

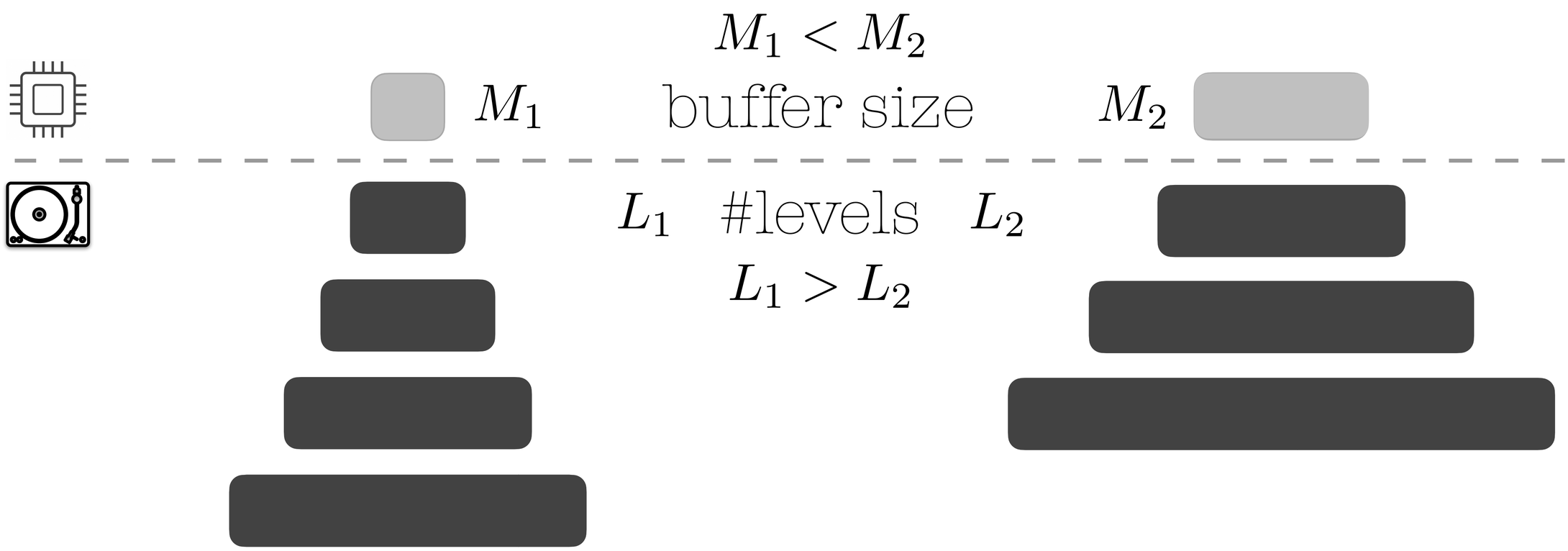
Ingestion-only workloads

Mixed workloads

I/O-bound workloads

BOSTON UNIVERSITY

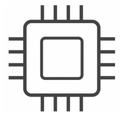
# Size of the Buffer

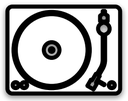



- frequent flushes
- smaller but more levels
- poor read performance

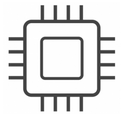
- fewer larger levels
- good for reads
- high tail latency

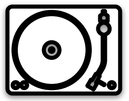
# #Buffer Components



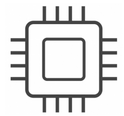


# #Buffer Components

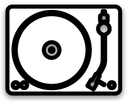




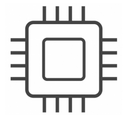
# #Buffer Components



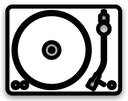
immutable
buffers



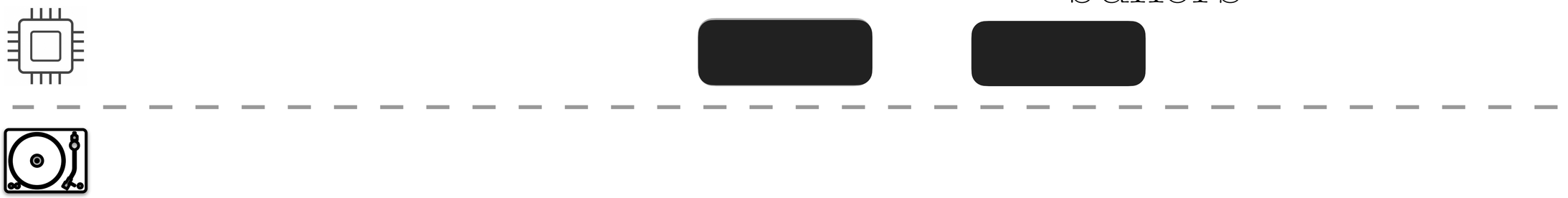
# #Buffer Components



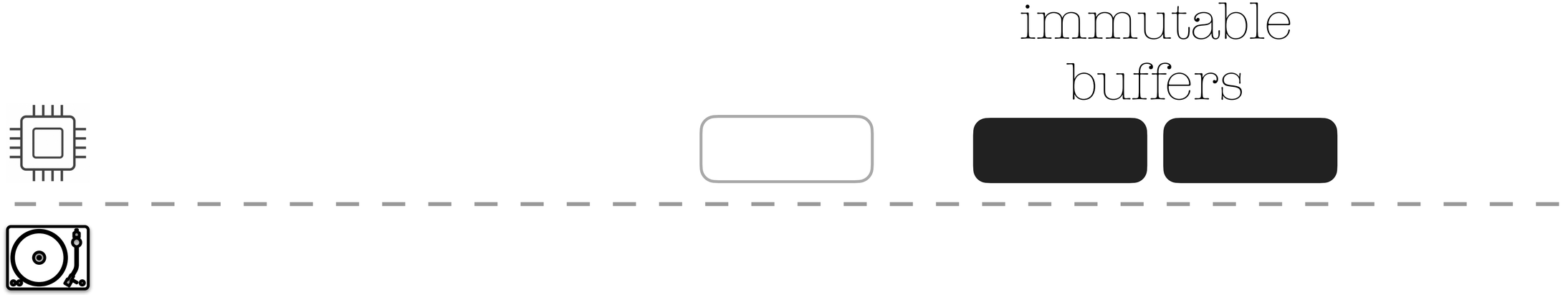
immutable
buffers



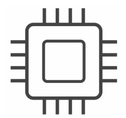
# #Buffer Components



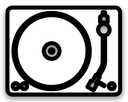
# #Buffer Components



# #Buffer Components

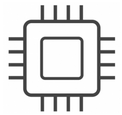


immutable
buffers

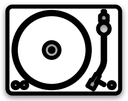


BOSTON
UNIVERSITY

# #Buffer Components

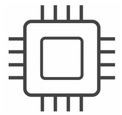


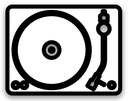
immutable
buffers



# #Buffer Components

immutable
buffers

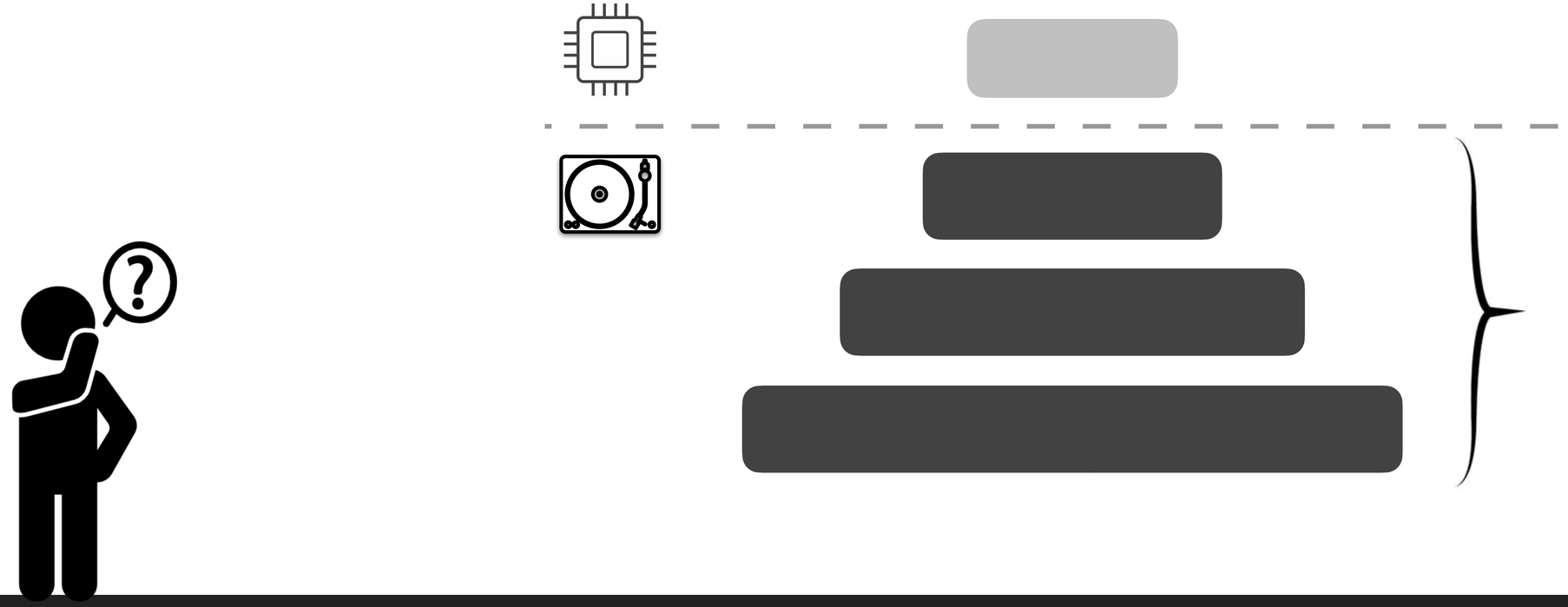




lazy flushing

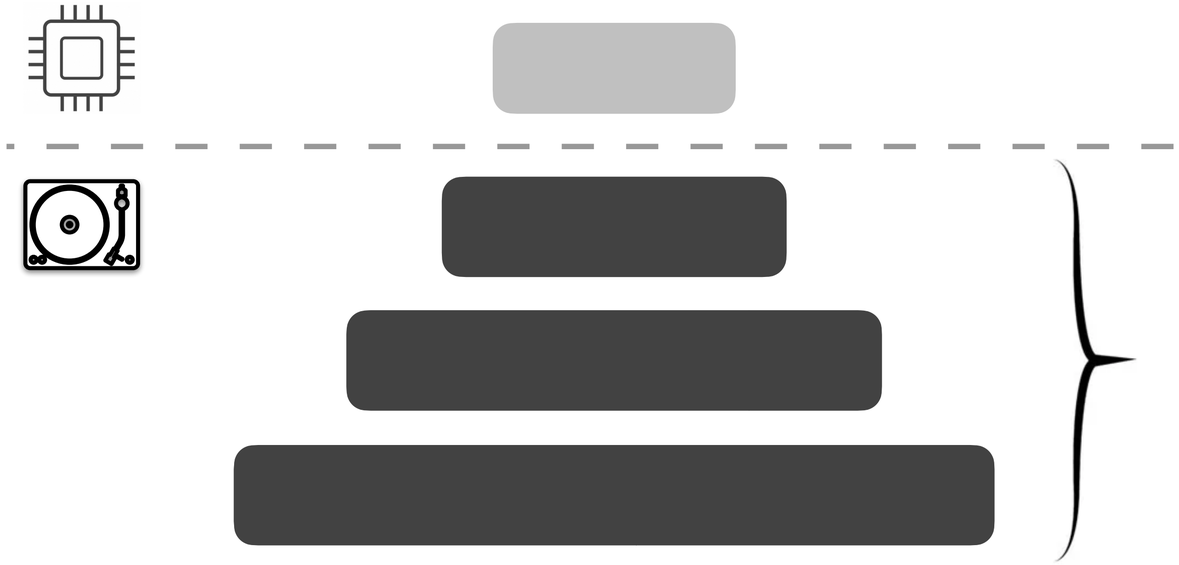
- avoid write stalls
- improved ingestion throughput
- better bandwidth utilization
- requires more memory

BOSTON
UNIVERSITY




# How does the storage layer affect ingestion?

$L$: #levels
$T$: size ratio



most data
on storage

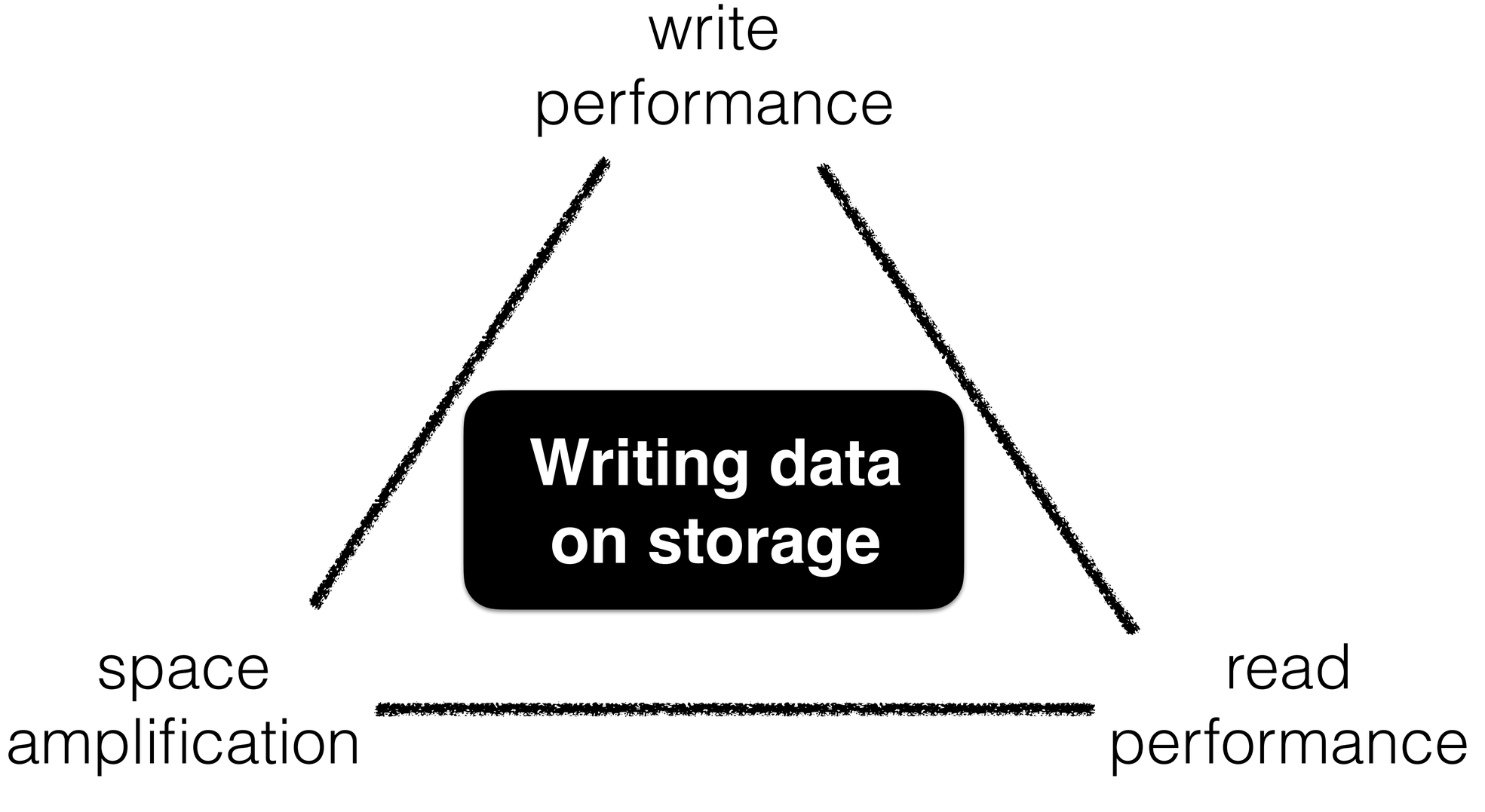
if $T = 10$ & $L = 4$

99.9% on storage

How does the storage layer affect ingestion?

BOSTON
UNIVERSITY

# **Storage** Optimizations



# Data **Layout**

Classical LSM design: **leveling**

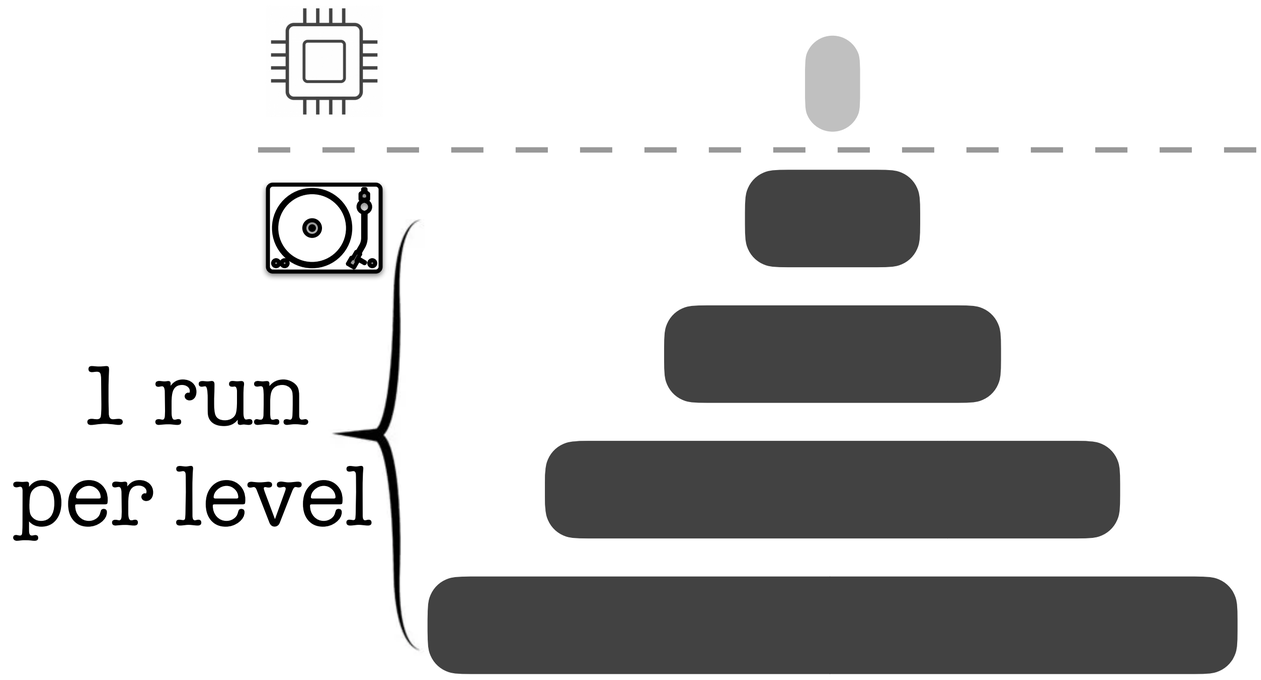
[eager merging]



BOSTON UNIVERSITY

# Data **Layout**

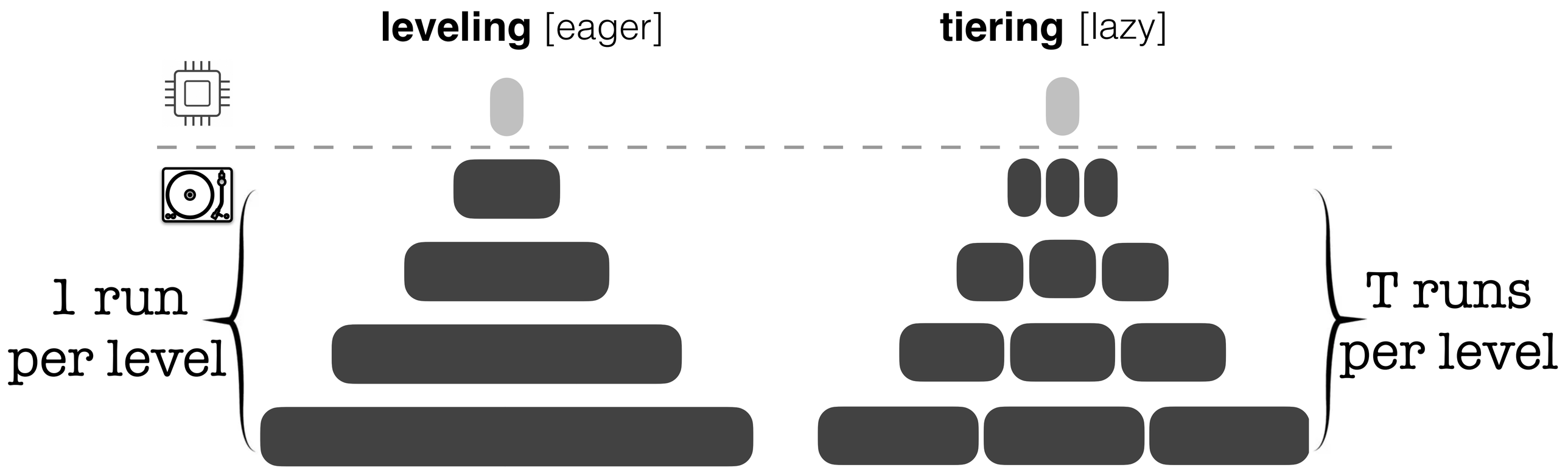
**leveling** [eager]




- good read performance
- good space amplification
- high write amplification

BOSTON UNIVERSITY

# Data **Layout**




- good read performance
- good space amplification
- high write amplification

- poor read performance
- poor space amplification
- good ingestion performance

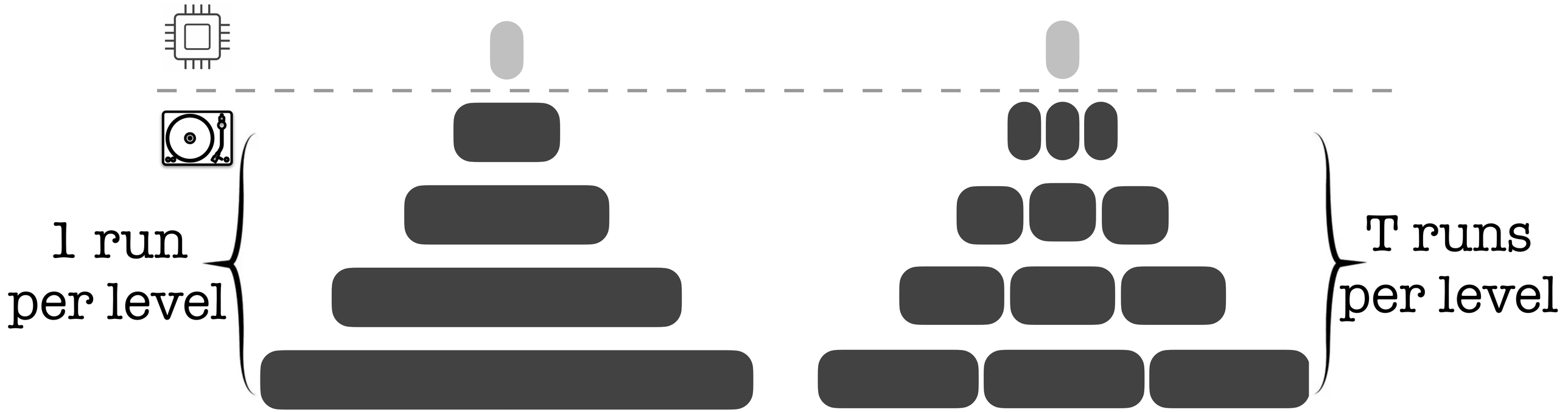
BOSTON UNIVERSITY

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

# Data **Layout**

**leveling** [eager]

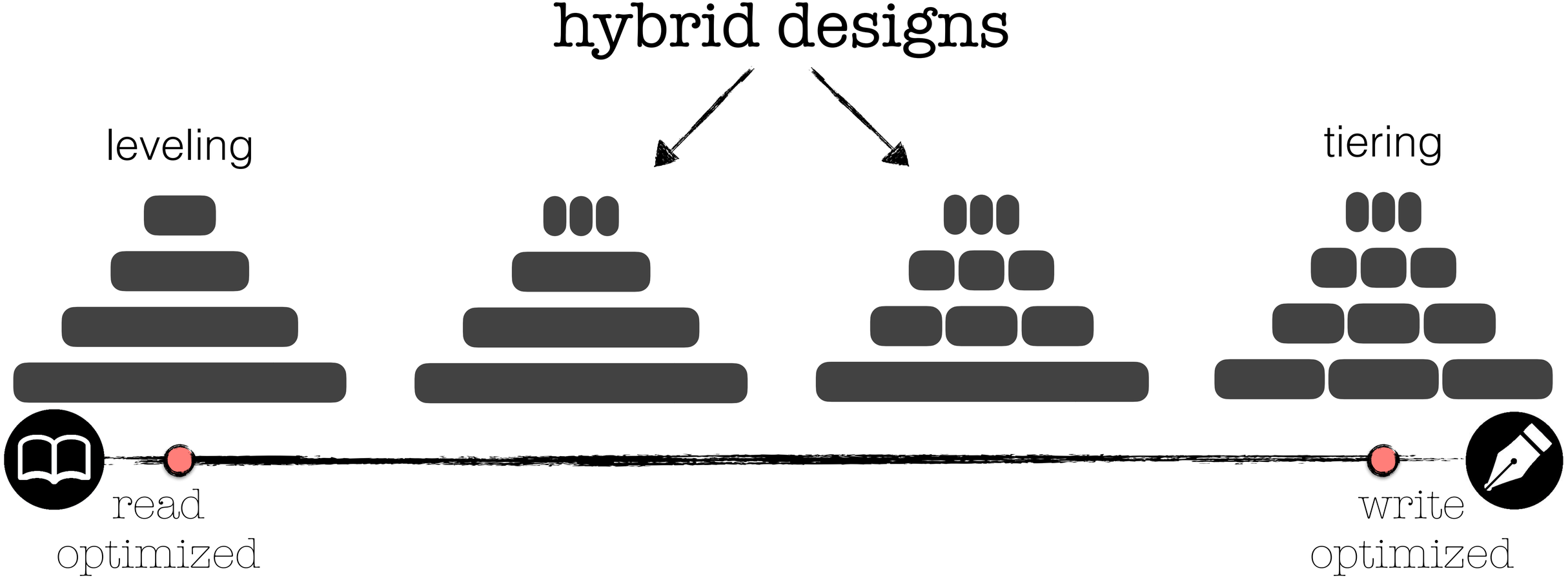
**tiering** [lazy]




| | **leveling** [eager] | **tiering** [lazy] |
|---|---|---|
| Read cost: | $\mathcal{O}(L \cdot \phi)$ | $\mathcal{O}(T \cdot L \cdot \phi)$ |
| Write cost: | $\mathcal{O}(T \cdot L/B)$ | $\mathcal{O}(L/B)$ |
| SA: | $\mathcal{O}(1/T)$ | $\mathcal{O}(T)$ |

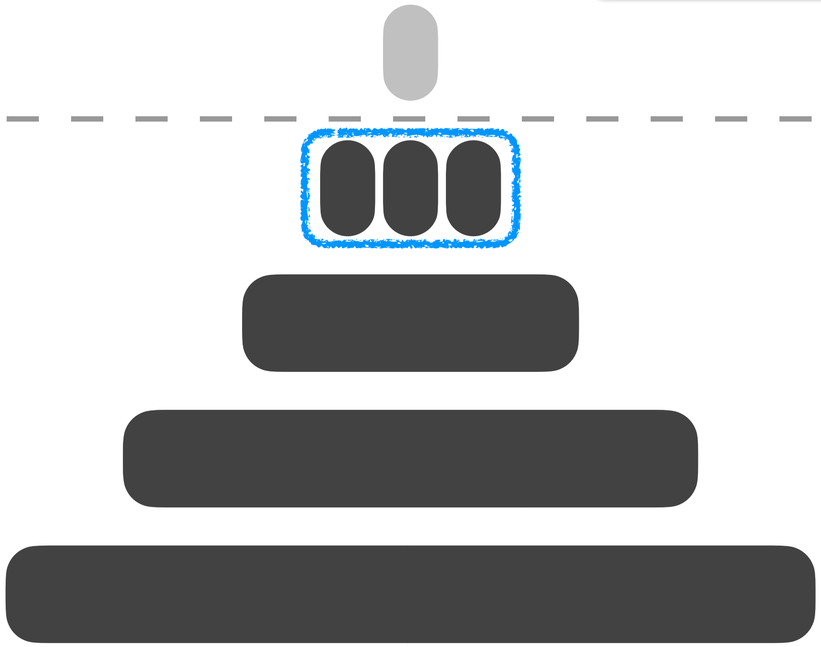
BOSTON UNIVERSITY

# Data **Layout**


hybrid designs
leveling
tiering
read
optimized
write
optimized

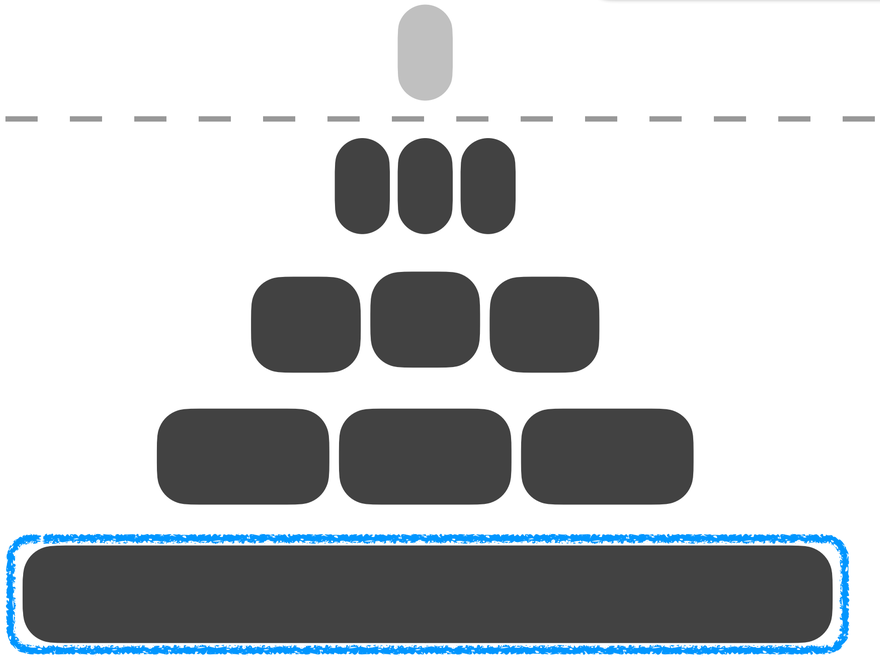
# Data **Layout**

**1-leveling** RocksDB20



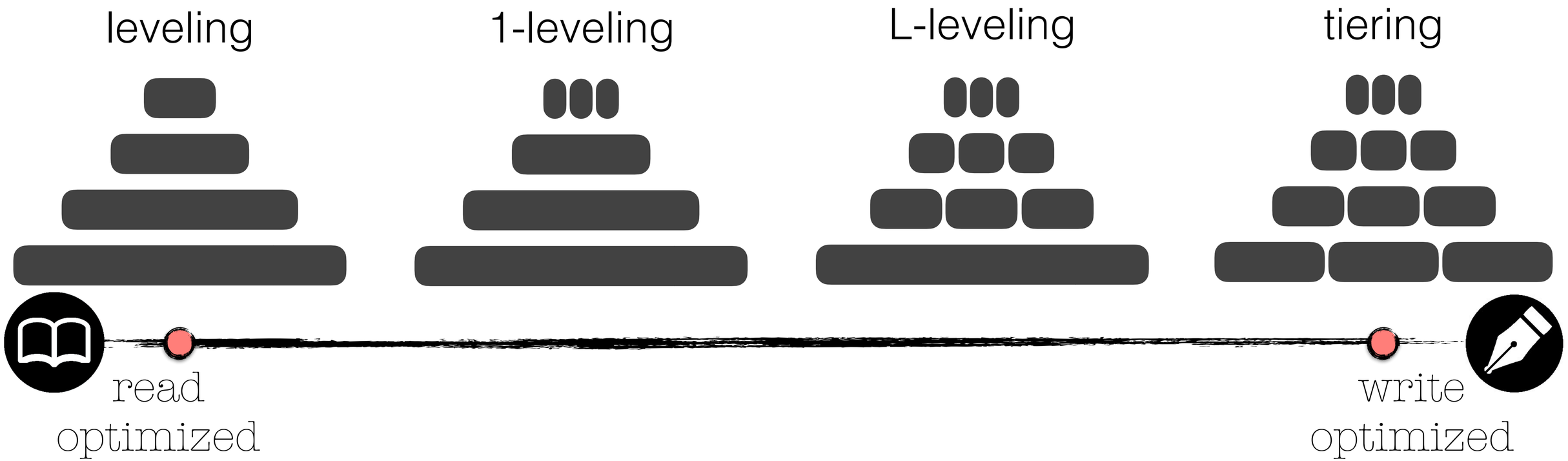
- fewer write stalls
- increased block cache hits

**L-leveling** DayanSIGMOD18

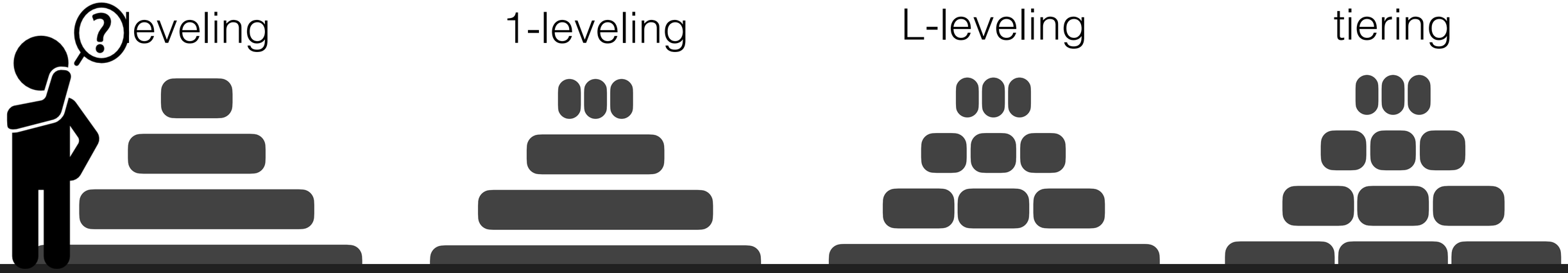


- low write amplification
- better read performance

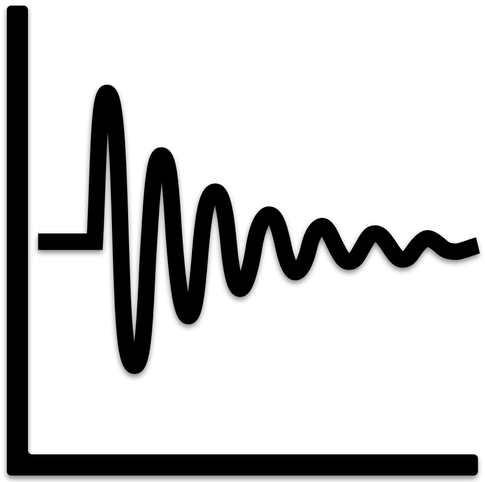
# Data **Layout**


leveling
1-leveling
L-leveling
tiering
read
optimized
write
optimized

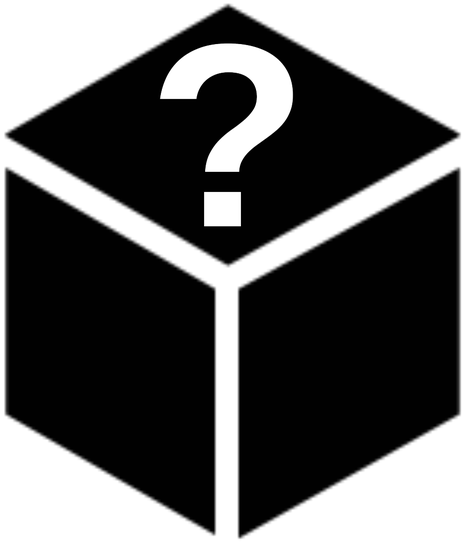
# Data **Layout**



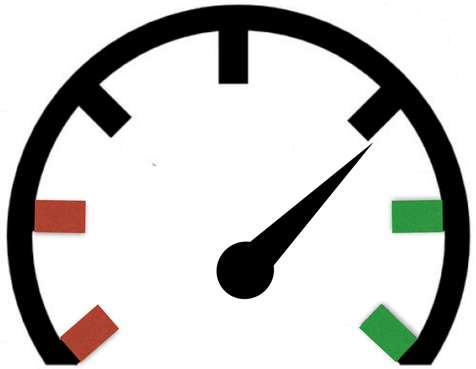

So, how do we reason about the data layout?



workload

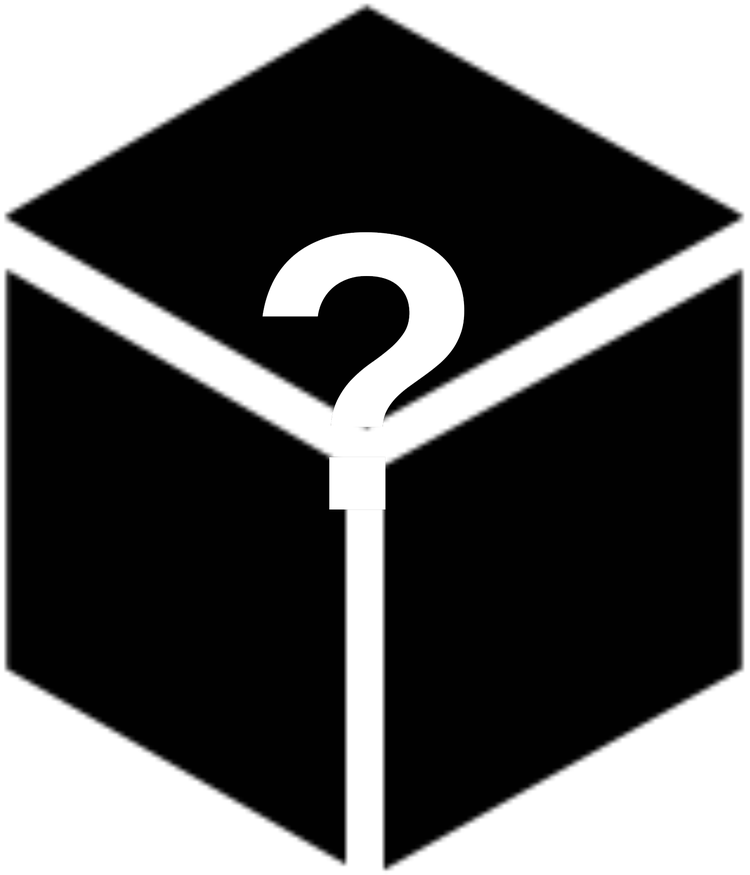



**data layout**

performance

BOSTON UNIVERSITY



# Compaction

black box


**How** to organize the data on device?


**How much** data to move at-a-time?


**Which** block of data to be moved?


**When** to re-organize the data layout?

SarkarVLDB21

BOSTON UNIVERSITY

Data Layout


**How** to organize the data on device? ✓

Compaction Granularity


**How much** data to move at-a-time?

Data Movement Policy


**Which** block of data to be moved?

Compaction Trigger


**When** to re-organize the data layout?

SarkarVLDB21


# Compaction **Granularity**

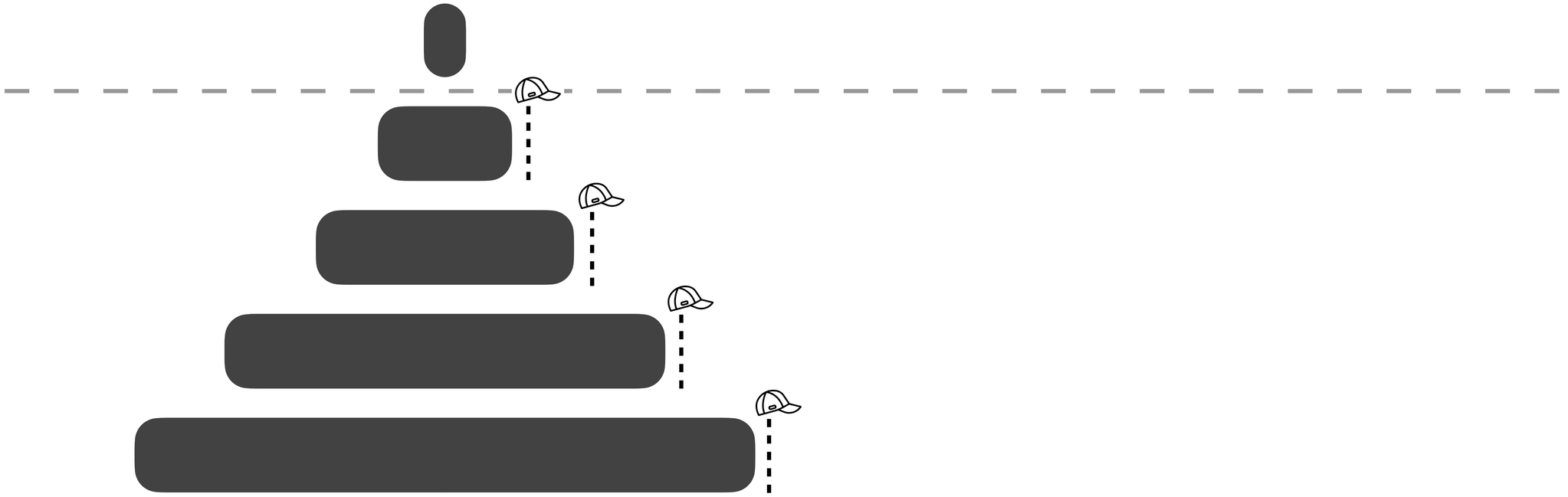
*data moved per compaction*

consecutive levels

AsterixDB

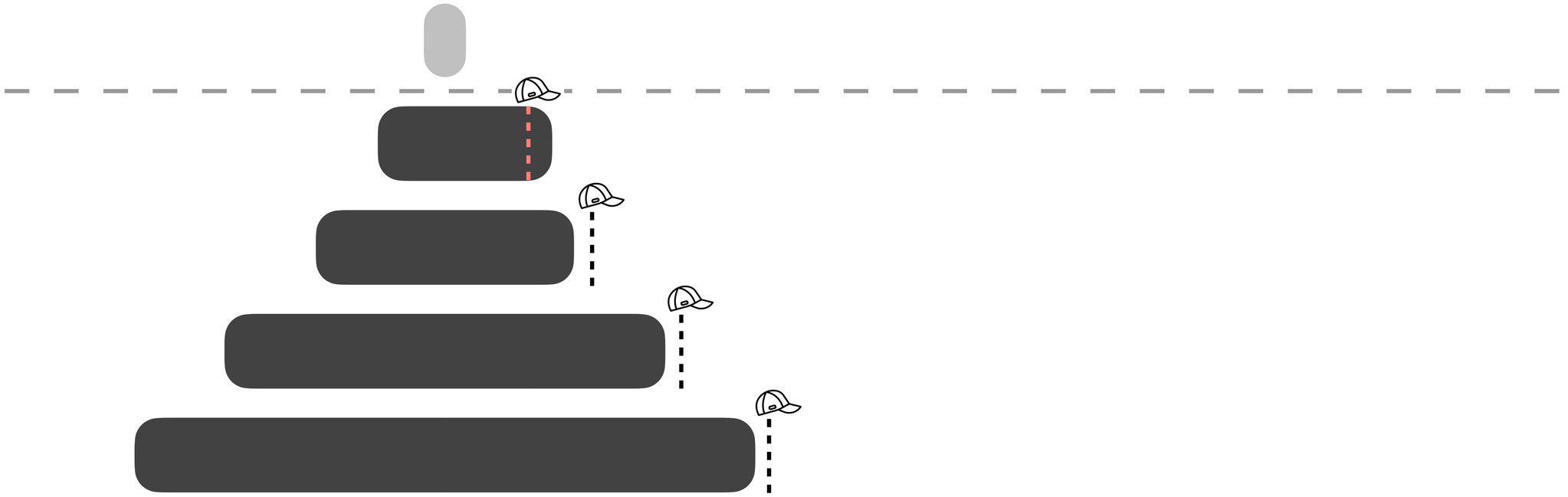

# Compaction **Granularity**

*data moved per compaction*



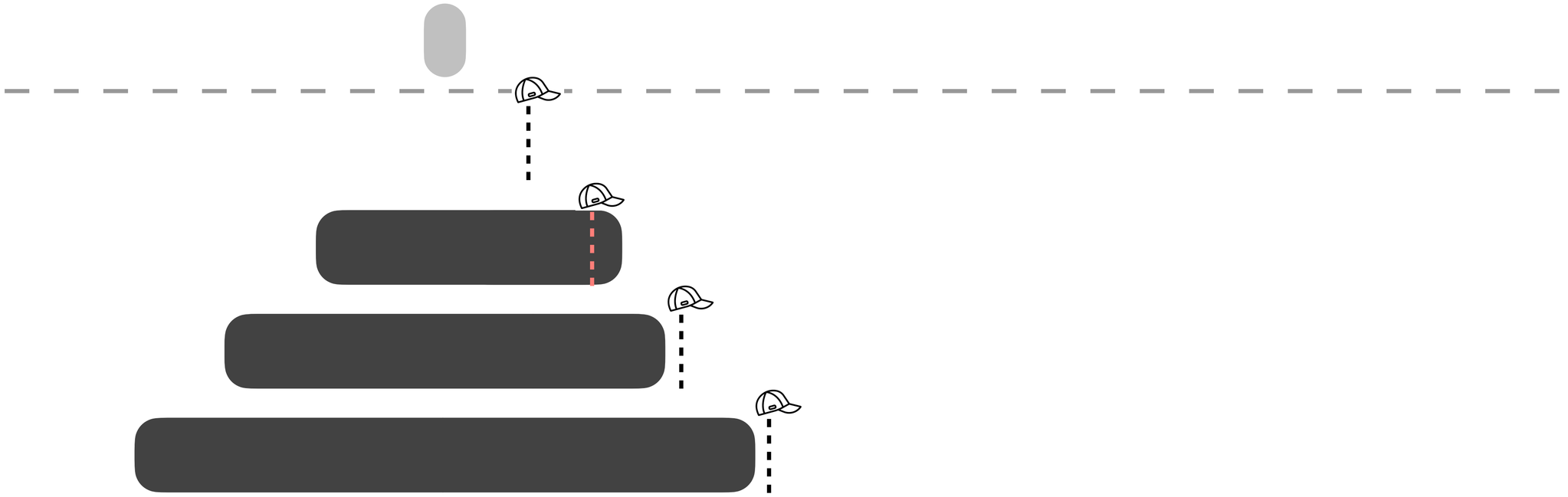

# Compaction **Granularity**

*data moved per compaction*



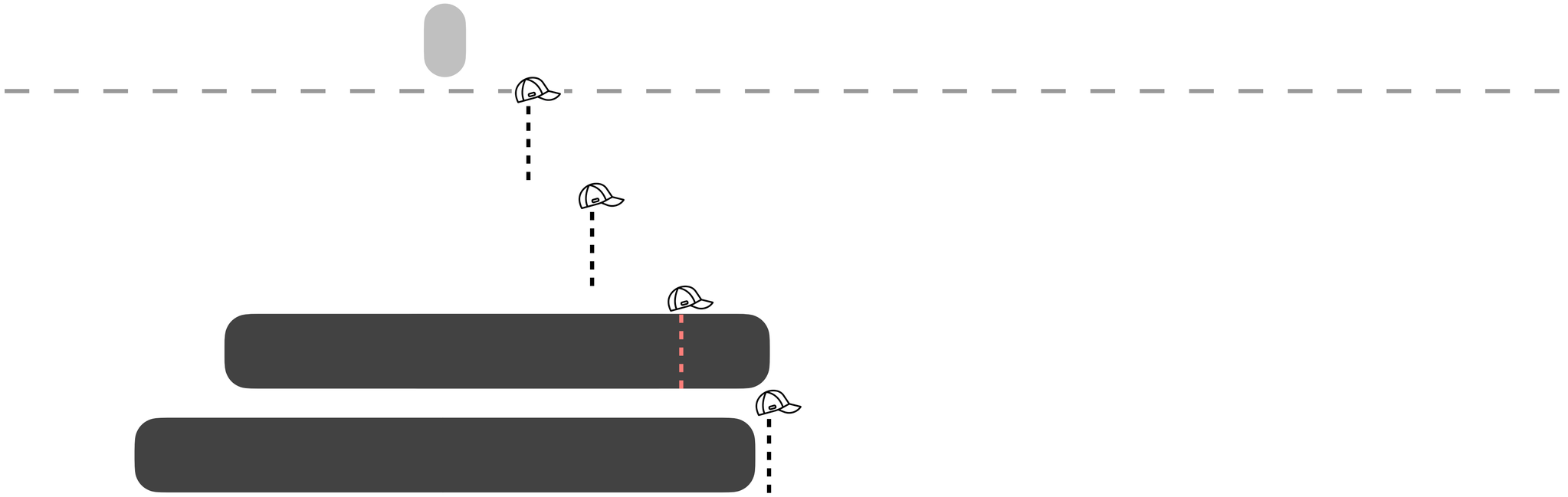

# Compaction **Granularity**

*data moved per compaction*



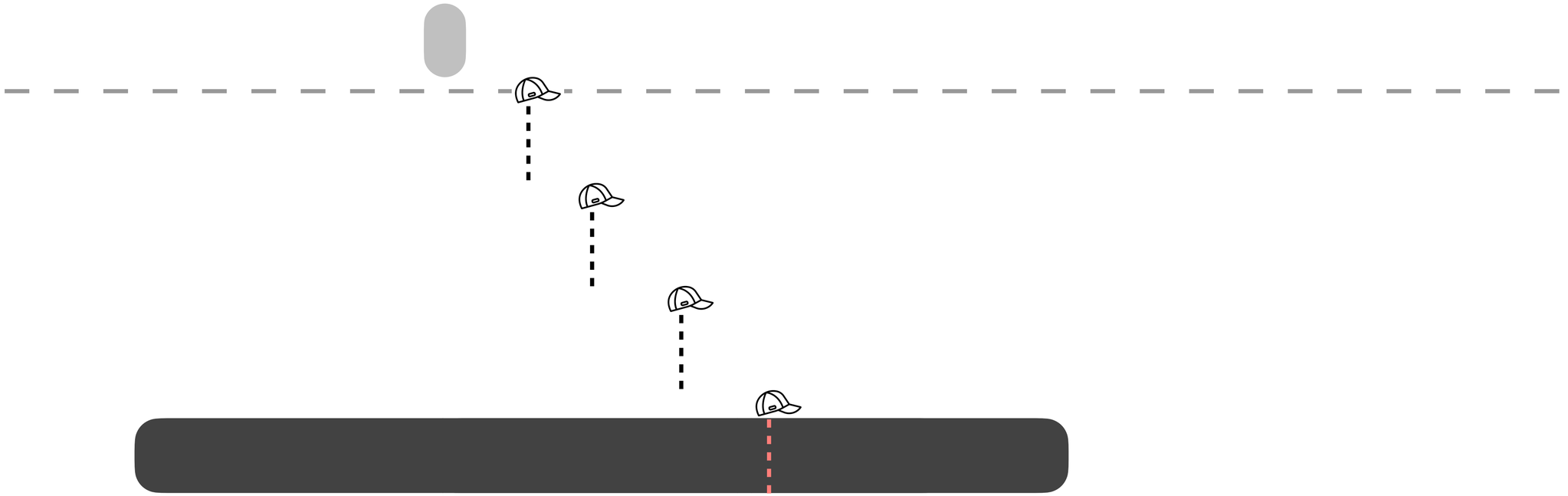

# Compaction **Granularity**

*data moved per compaction*



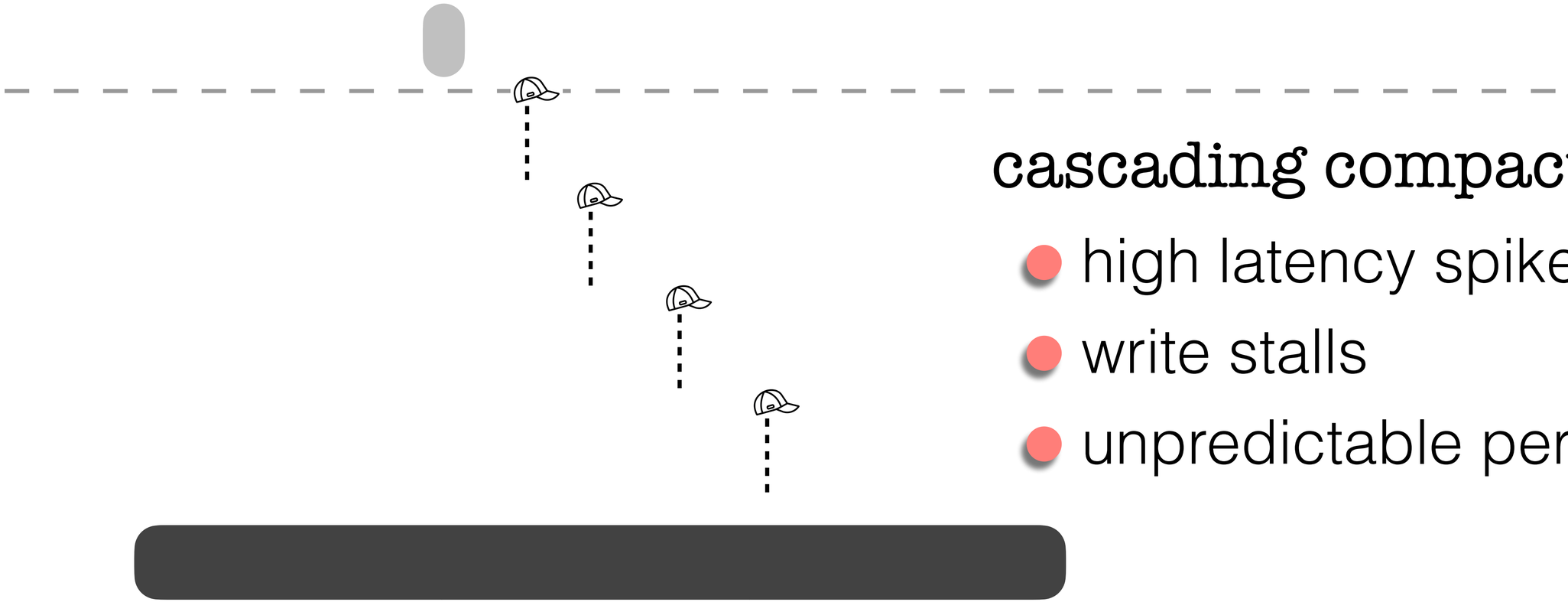

# Compaction **Granularity**

*data moved per compaction*




# Compaction **Granularity**

*data moved per compaction*



cascading compactions

- high latency spikes
- write stalls
- unpredictable perf.


# Compaction **Granularity**

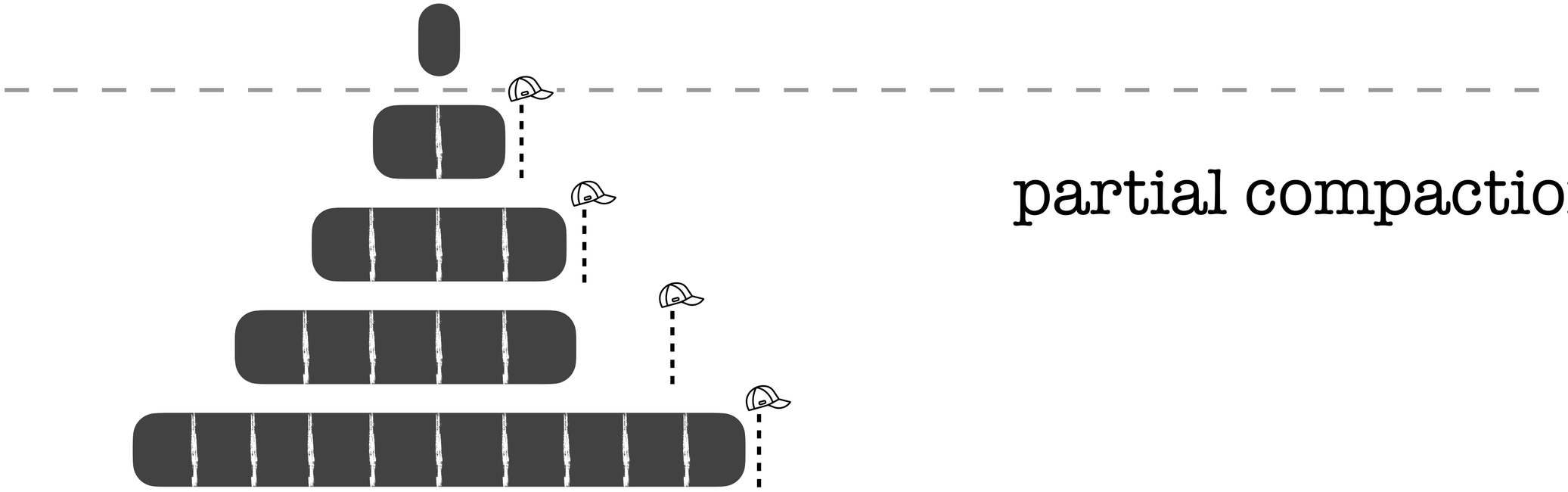
*data moved per compaction*

## partial compaction

granularity: **files**


# Compaction **Granularity**

*data moved per compaction*

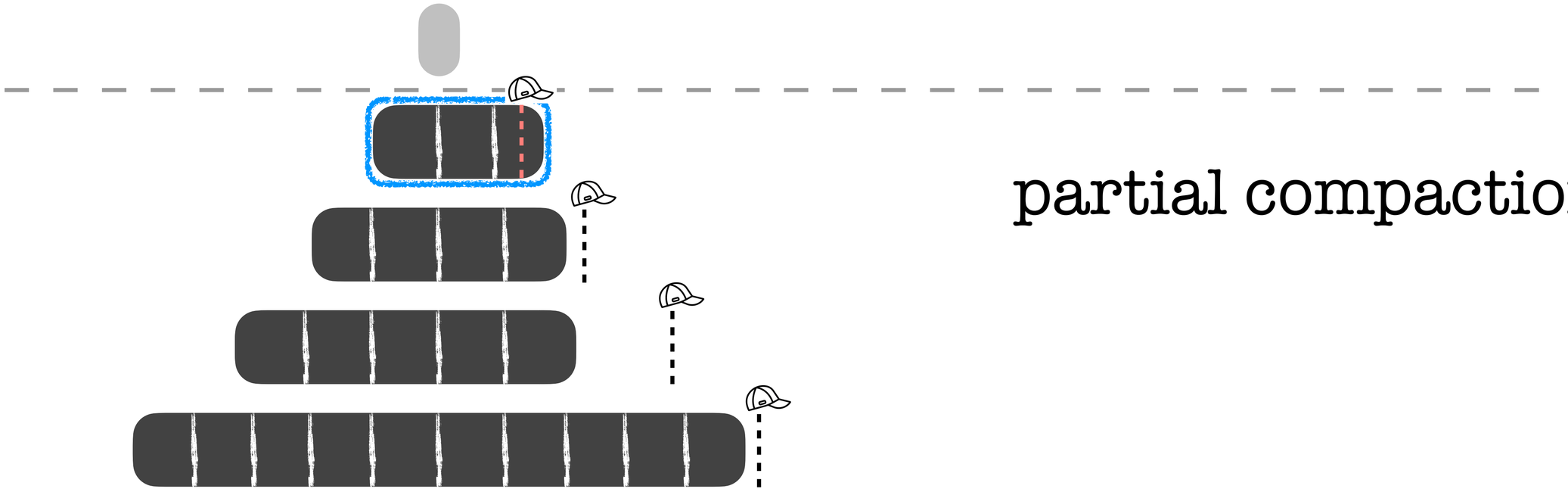


partial compaction

files


# Compaction **Granularity**

*data moved per compaction*

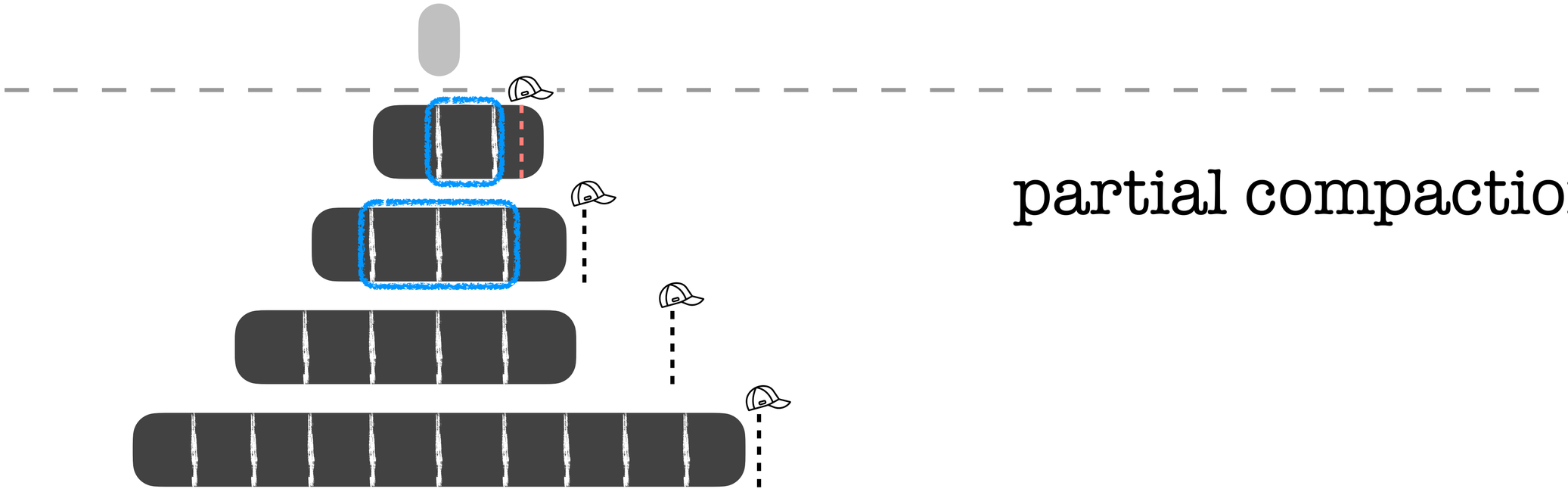


partial compaction

files


# Compaction **Granularity**

*data moved per compaction*

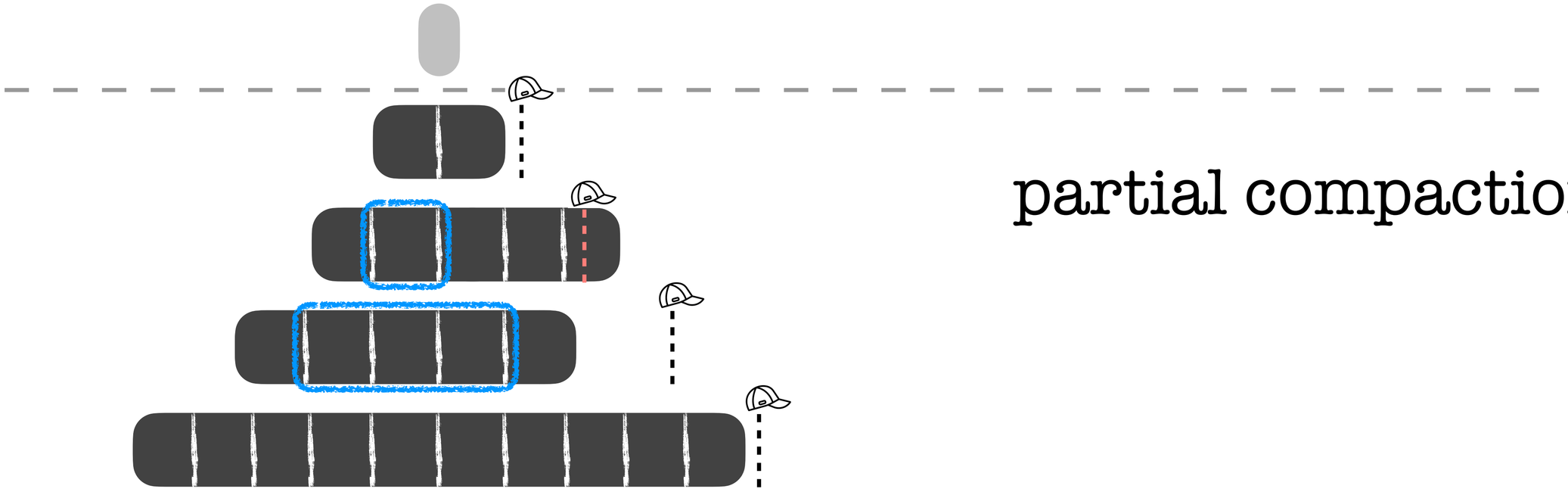


partial compaction

files


# Compaction **Granularity**

*data moved per compaction*

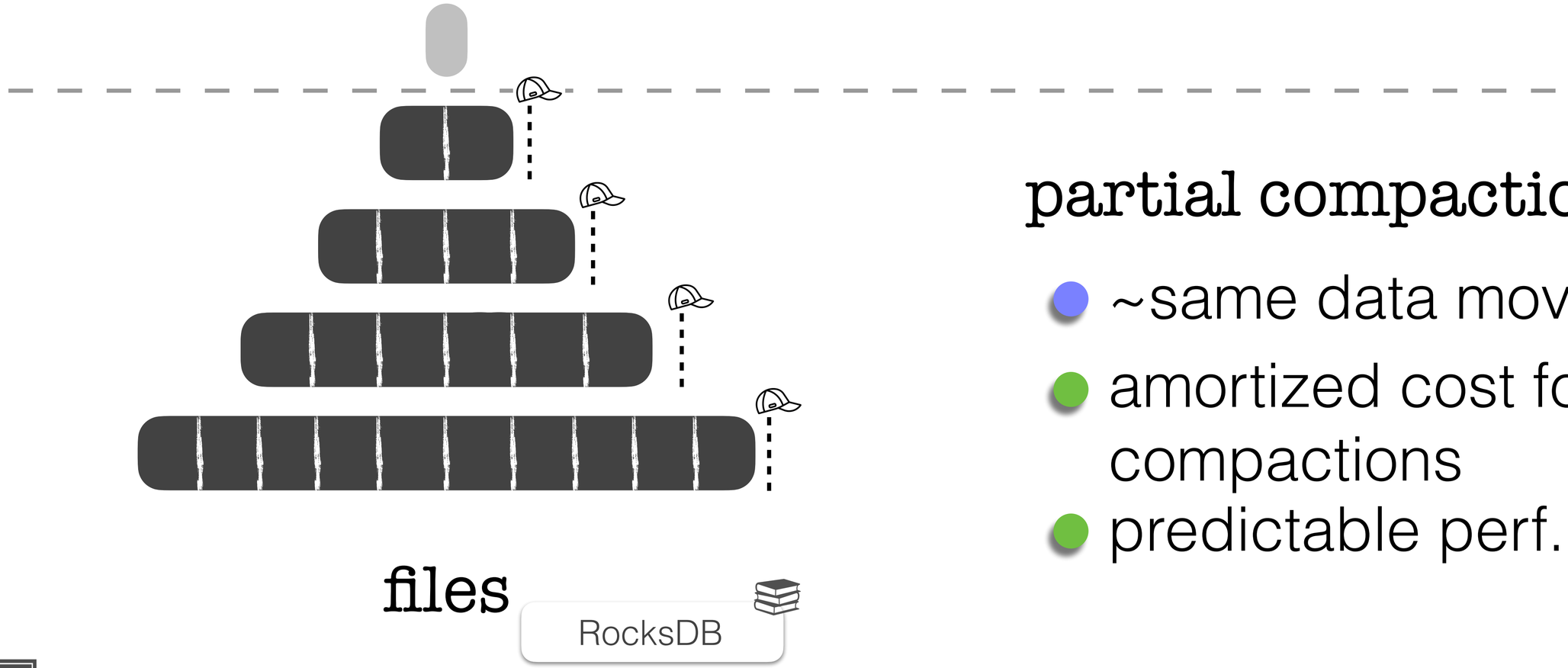


partial compaction

files


# Compaction **Granularity**

*data moved per compaction*

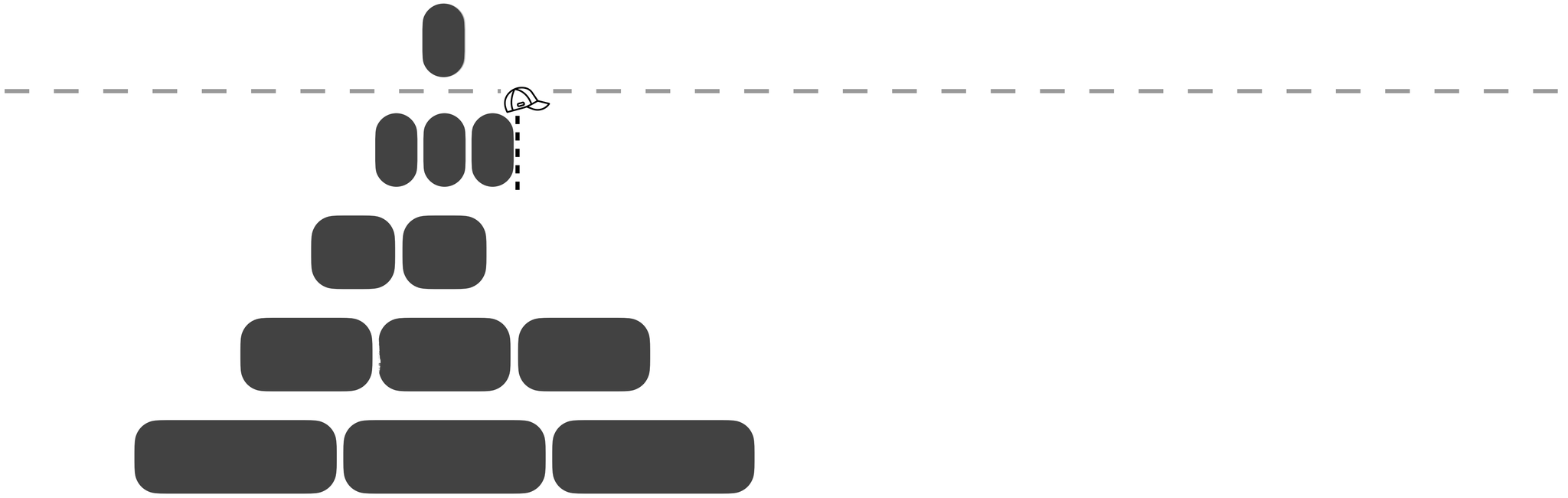



partial compaction

- ~same data movement
- amortized cost for compactions
- predictable perf.


# Compaction **Granularity**

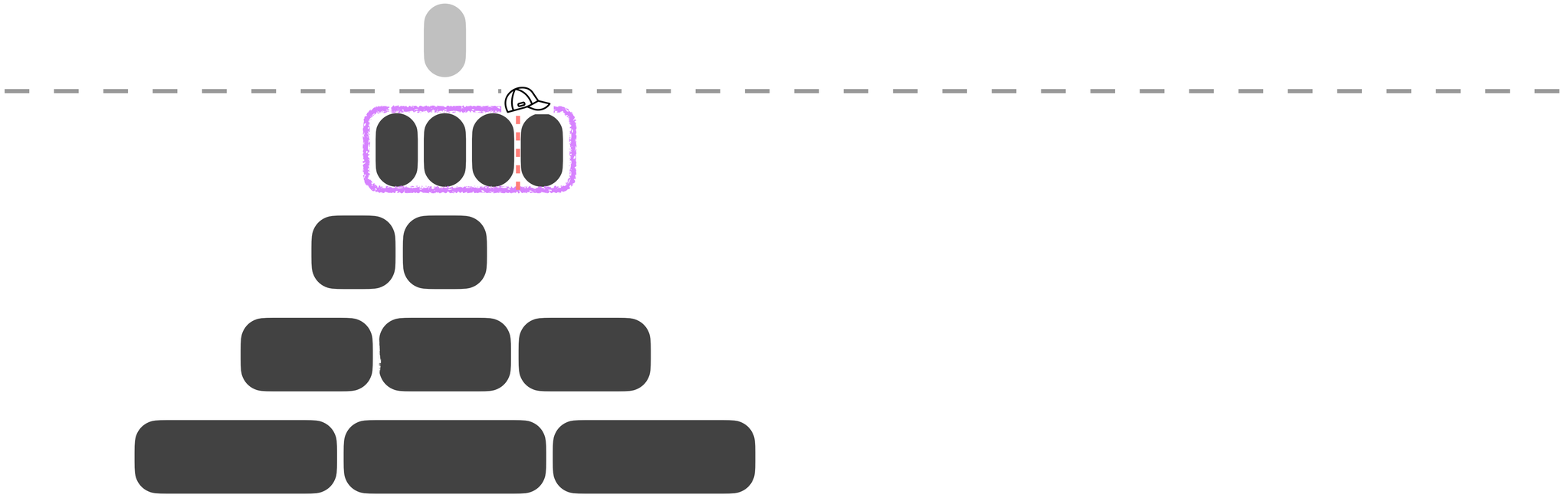
*data moved per compaction*



sorted runs in a level


# Compaction **Granularity**

*data moved per compaction*

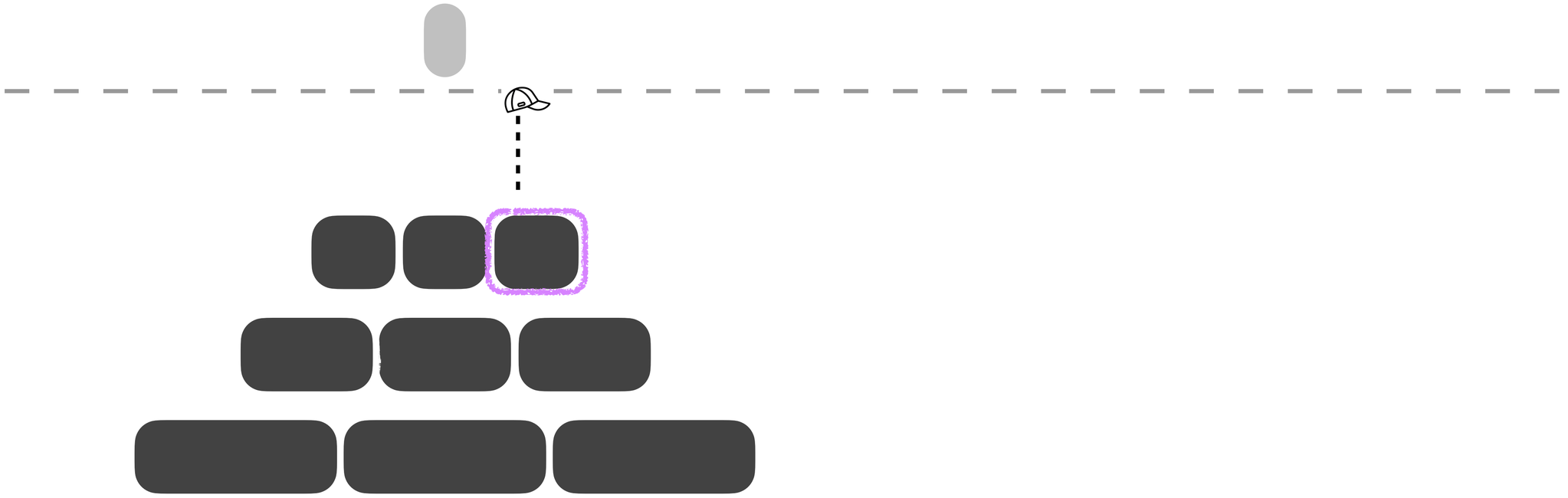


sorted runs in a level

BOSTON UNIVERSITY


# Compaction **Granularity**

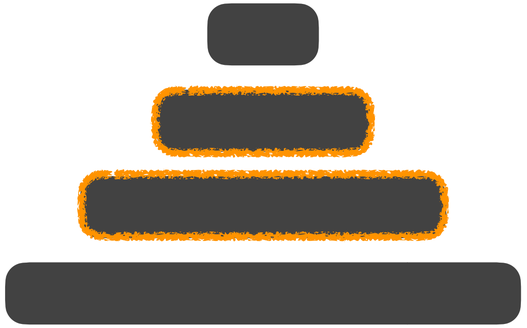
*data moved per compaction*



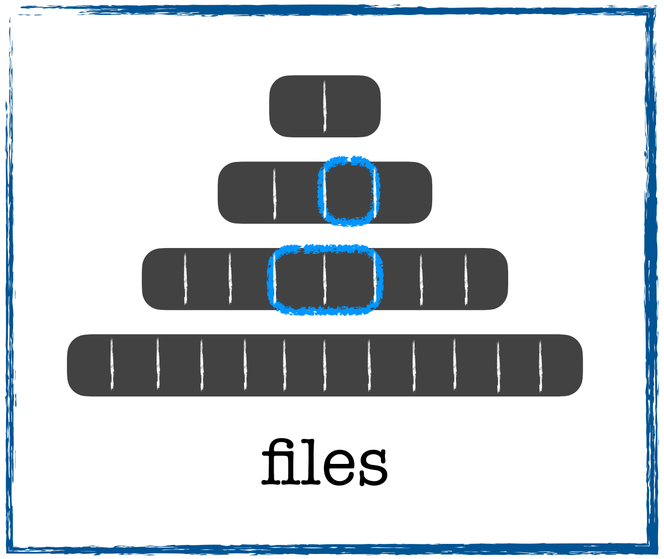
sorted runs in a level


# Compaction **Granularity**

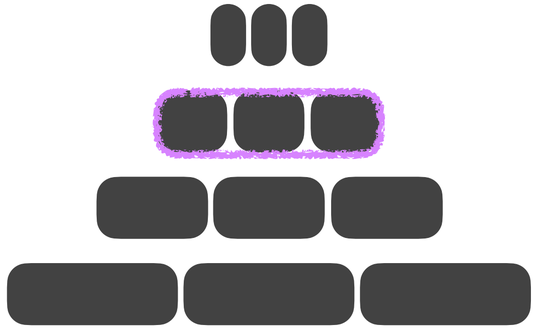
*data moved per compaction*



levels



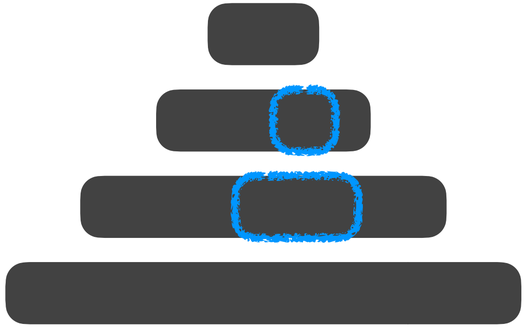
files



sorted runs in a level


# **Data Movement** Policy

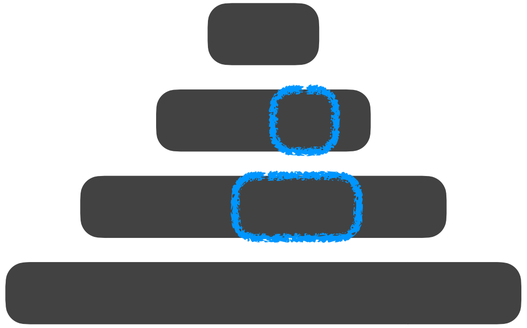
*which data to compact*



files


# **Data Movement** Policy

*which data to compact*



files

**round-robin** !

minimum **overlap with parent** level WA

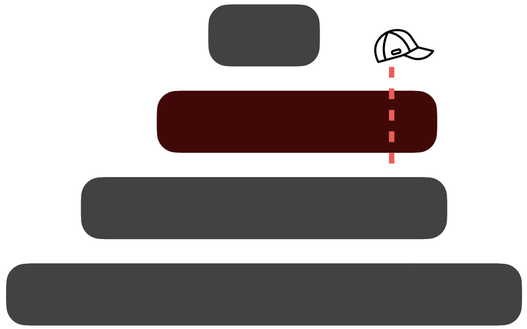
file with most **tombstones** SA

**coldest** file PQ


# Compaction **Trigger**

*invoking the compaction routine*

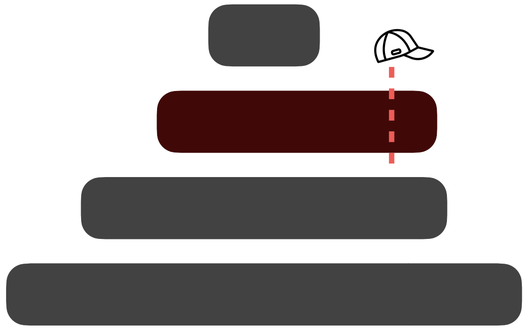
level **saturation**



BOSTON UNIVERSITY


# Compaction **Trigger**

*invoking the compaction routine*



level **saturation**

number of **sorted runs**

**space amplification**

SA

**age** of a file

De


Data Layout


Compaction Granularity


Data Movement Policy

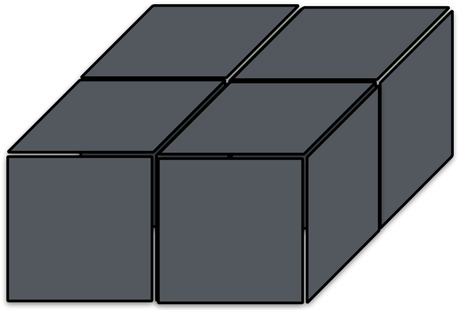

Compaction Trigger

Data Layout

Compaction Granularity

Data Movement Policy

Compaction Trigger



***Any* Compaction Algorithm**

| Database | Data layout | Compaction Trigger | | | | | Compaction Granularity | | | | Data Movement Policy | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Level saturation | #Sorted runs | File staleness | Space amp. | Tombstone-TTL | Level | Sorted run | File (single) | File (multiple) | Round-robin | Least overlap (+1) | Least overlap (+2) | Coldest file | Oldest file | Tombstone density | Expired TS-TTL | N/A (entire level) |
| RocksDB [30], Monkey [22] | Leveling / 1-Leveling | ✓ | | ✓ | | | | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | | |
| | Tiering | | ✓ | | ✓ | ✓ | | ✓ | | | | | | | | | | ✓ |
| LevelDB [32], Monkey (J.) [21] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| SlimDB [47] | Tiering | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Dostoevsky [23] | *L*-leveling | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | | | | | | $✓^{T}$ |
| LSM-Bush [24] | Hybrid leveling | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | | | | | | $✓^{T}$ |
| Lethe [51] | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | | ✓ | |
| Silk [11], Silk+ [12] | Leveling | ✓ | | | | | | | ✓ | ✓ | ✓ | | | | | | | |
| HyperLevelDB [35] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| PebblesDB [46] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Cassandra [8] | Tiering | | ✓ | ✓ | | ✓ | | ✓ | | | | | | | | | | ✓ |
| | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | |
| WiredTiger [62] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| X-Engine [34], Leaper [63] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | ✓ | | | | ✓ | | |
| HBase [7] | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |
| AsterixDB [3] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |

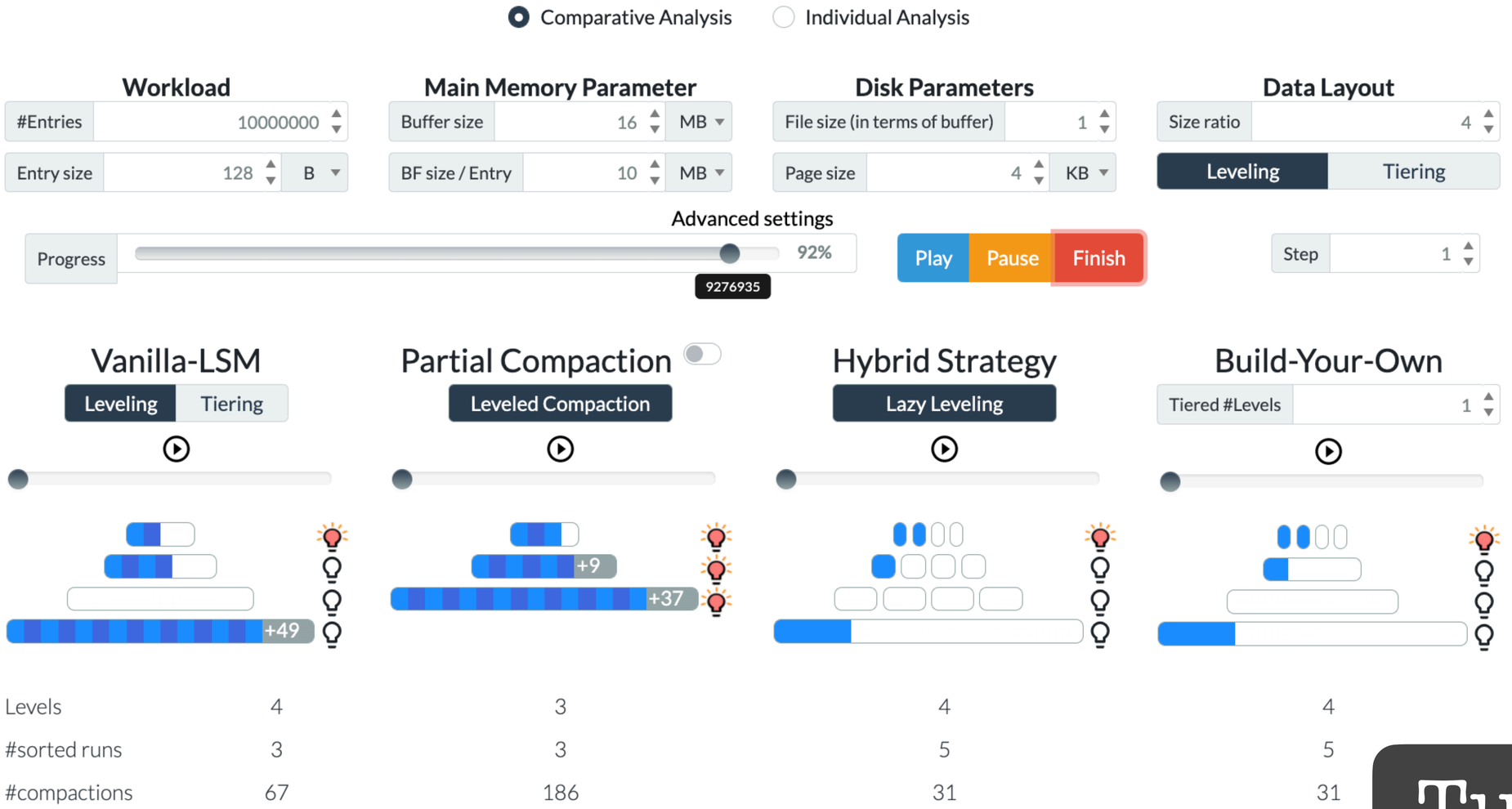
BOSTON UNIVERSITY

| Database | Data layout | Compaction Trigger | | | | | Compaction Granularity | | | | Data Movement Policy | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Level saturation | #Sorted runs | File staleness | Space amp. | Tombstone-TTL | Level | Sorted run | File (single) | File (multiple) | Round-robin | Least overlap (+1) | Least overlap (+2) | Coldest file | Oldest file | Tombstone density | Expired TS-TTL | N/A (entire level) |
| RocksDB [30], Monkey [22] | Leveling / 1-Leveling | ✓ | | ✓ | | | | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | | |
| | Tiering | | ✓ | | ✓ | ✓ | | ✓ | | | | | | | | | | ✓ |
| LevelDB [32], Monkey (J.) [21] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| SlimDB [47] | Tiering | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Dostoevsky [23] | *L*-leveling | $✓^L$ | $✓^T$ | | | | $✓^L$ | $✓^T$ | | | | $✓^L$ | | | | | | $✓^T$ |
| LSM-Bush [24] | Hybrid leveling | $✓^L$ | $✓^T$ | | | | $✓^L$ | $✓^T$ | | | | $✓^L$ | | | | | | $✓^T$ |
| Lethe [51] | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | | ✓ | |
| Silk [11], Silk+ [12] | Leveling | ✓ | | | | | | | ✓ | ✓ | ✓ | | | | | | | |
| HyperLevelDB [35] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| PebblesDB [46] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Cassandra [8] | Tiering | | ✓ | ✓ | | ✓ | | ✓ | | | | | | | | | | ✓ |
| | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | |
| WiredTiger [62] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| X-Engine [34], Leaper [63] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | ✓ | | | | ✓ | | |
| HBase [7] | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |
| AsterixDB [3] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |

| Database | Data layout | Compaction Trigger | | | | | Compaction Granularity | | | | Data Movement Policy | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Level saturation | #Sorted runs | File staleness | Space amp. | Tombstone-TTL | Level | Sorted run | File (single) | File (multiple) | Round-robin | Least overlap (+1) | Least overlap (+2) | Coldest file | Oldest file | Tombstone density | Expired TS-TTL | N/A (entire level) |
| RocksDB [30], Monkey [22] | Leveling / 1-Leveling | ✓ | | ✓ | | | | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | | |
| | Tiering | | ✓ | | ✓ | ✓ | | ✓ | | | | | | | | | | ✓ |
| LevelDB [32], Monkey (J.) [21] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| SlimDB [47] | Tiering | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Dostoevsky [23] | *L*-leveling | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | | | | | | $✓^{T}$ |
| LSM-Bush [24] | Hybrid leveling | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | | | | | | $✓^{T}$ |
| Lethe [51] | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | | ✓ | |
| Silk [11], Silk+ [12] | Leveling | ✓ | | | | | | | ✓ | ✓ | ✓ | | | | | | | |
| HyperLevelDB [35] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| PebblesDB [46] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Cassandra [8] | Tiering | | ✓ | ✓ | | ✓ | | ✓ | | | | | | | | | | ✓ |
| | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | |
| WiredTiger [62] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| X-Engine [34], Leaper [63] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | ✓ | | | | ✓ | | |
| HBase [7] | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |
| AsterixDB [3] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |

| Database | Data layout | Compaction Trigger | | | | | Compaction Granularity | | | | Data Movement Policy | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Level saturation | #Sorted runs | File staleness | Space amp. | Tombstone-TTL | Level | Sorted run | File (single) | File (multiple) | Round-robin | Least overlap (+1) | Least overlap (+2) | Coldest file | Oldest file | Tombstone density | Expired TS-TTL | N/A (entire level) |
| RocksDB [30], Monkey [22] | Leveling / 1-Leveling | ✓ | | ✓ | | | | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | | |
| | Tiering | | ✓ | | ✓ | ✓ | | ✓ | | | | | | | | | | ✓ |
| LevelDB [32], Monkey (J.) [21] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| SlimDB [47] | Tiering | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Dostoevsky [23] | *L*-leveling | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | | | | | | $✓^{T}$ |
| LSM-Bush [24] | Hybrid leveling | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | $✓^{T}$ | | | | $✓^{L}$ | | | | | | $✓^{T}$ |
| Lethe [51] | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | | ✓ | |
| Silk [11], Silk+ [12] | Leveling | ✓ | | | | | | | ✓ | ✓ | ✓ | | | | | | | |
| HyperLevelDB [35] | Leveling | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| PebblesDB [46] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | | | | | | | ✓ |
| Cassandra [8] | Tiering | | ✓ | ✓ | | ✓ | | ✓ | | | | | | | | | | ✓ |
| | Leveling | ✓ | | | | ✓ | | | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | |
| WiredTiger [62] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| X-Engine [34], Leaper [63] | Hybrid leveling | ✓ | | | | | | | ✓ | ✓ | | ✓ | | | | ✓ | | |
| HBase [7] | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |
| AsterixDB [3] | Leveling | ✓ | | | | | ✓ | | | | | | | | | | | ✓ |
| | Tiering | | ✓ | | | | | ✓ | | | | | | | | | | ✓ |

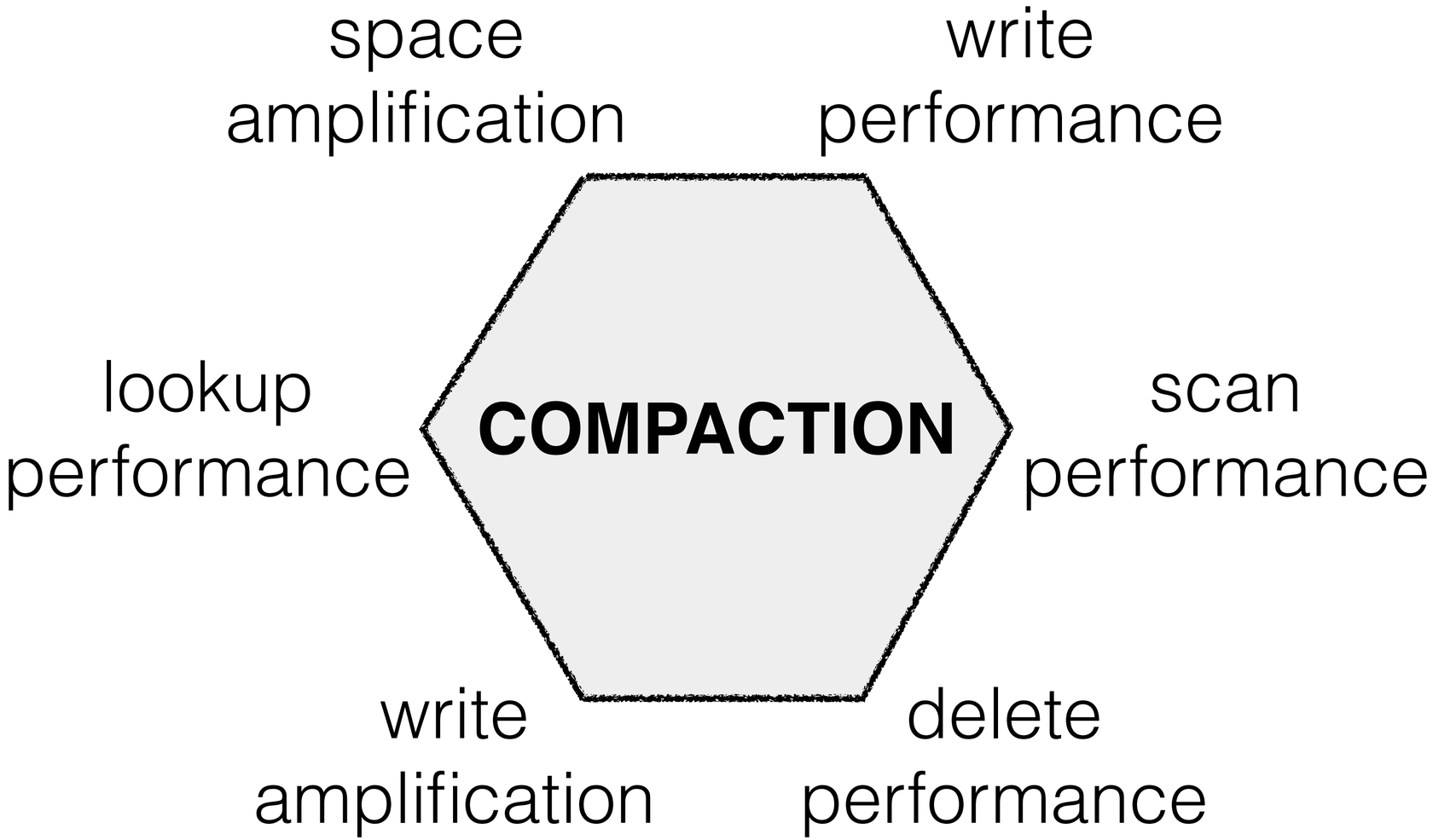
# SIGMOD Demo: Compactionary




BOSTON UNIVERSITY

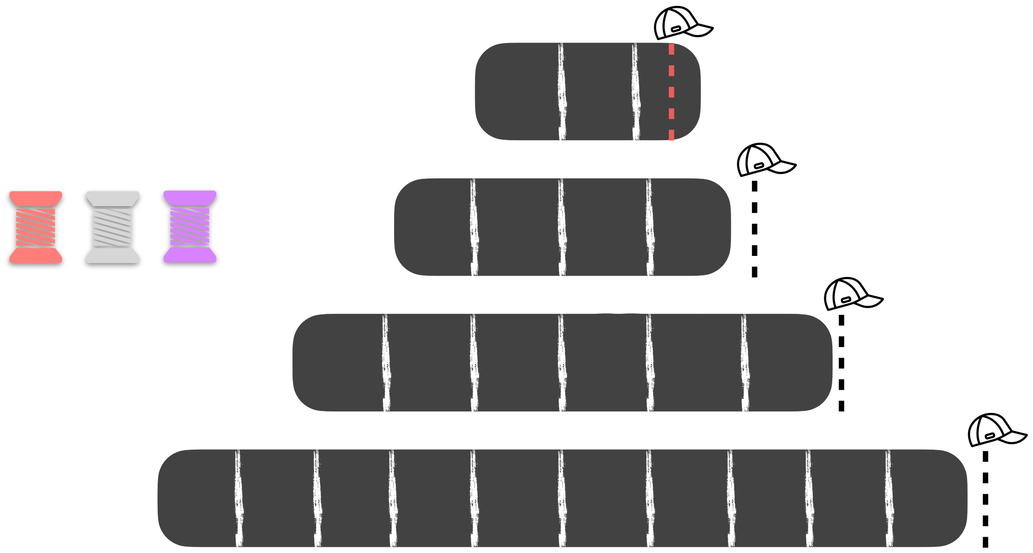
SarkarSIGMOD22

Tuesday 4-6PM


space
amplification
write
performance
lookup
performance
COMPACTION
scan
performance
write
amplification
delete
performance

# Optimizing **Compactions**

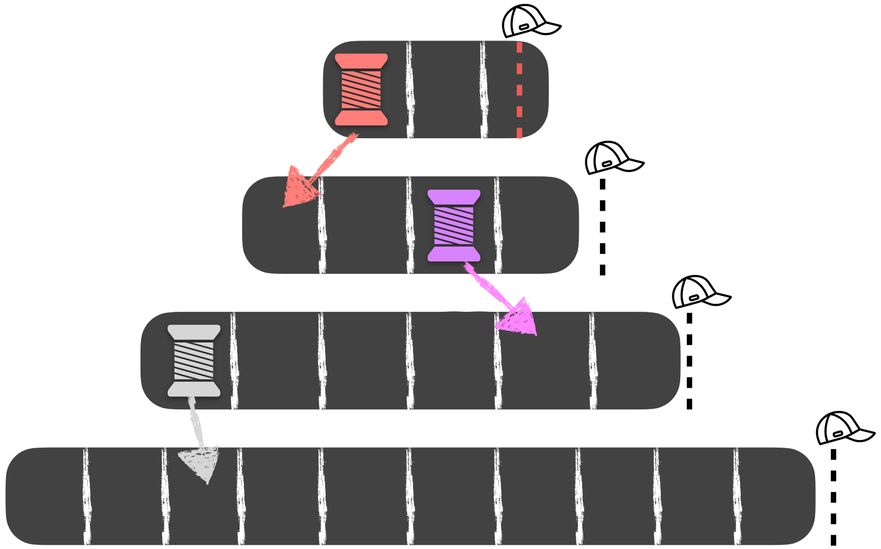
Background Compactions



BOSTON UNIVERSITY

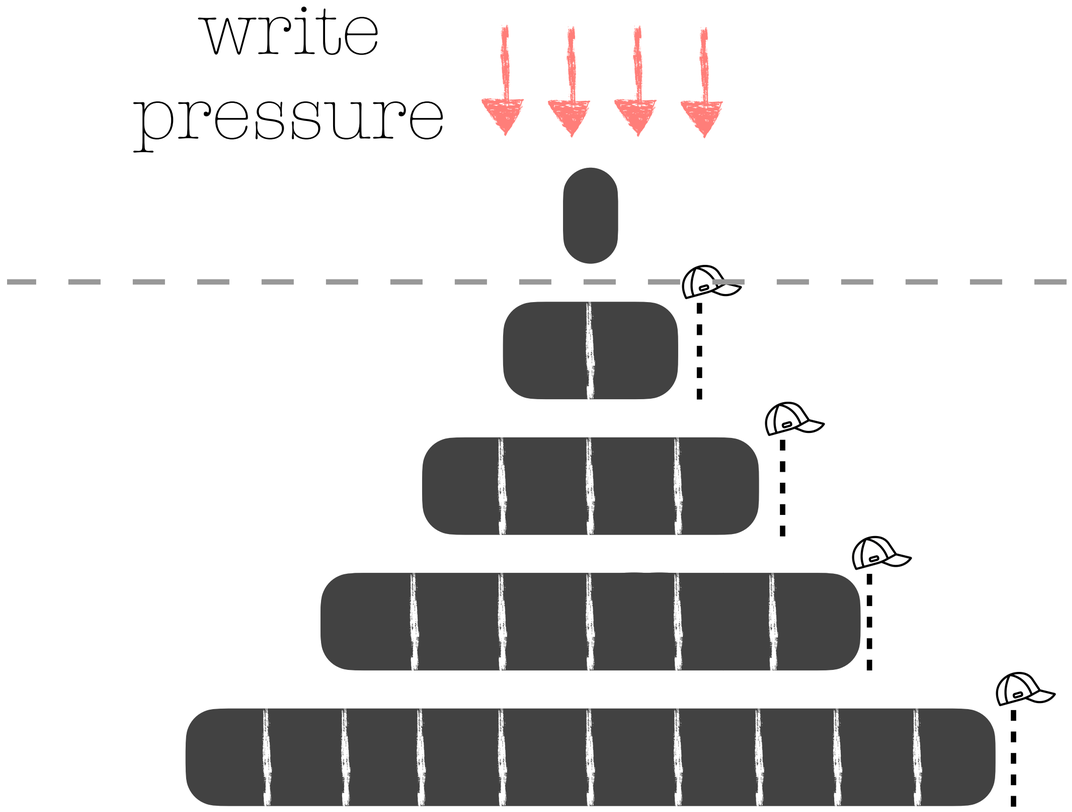
# Optimizing **Compactions**

Background Compactions



- non-blocking reads/writes
- improves write throughput

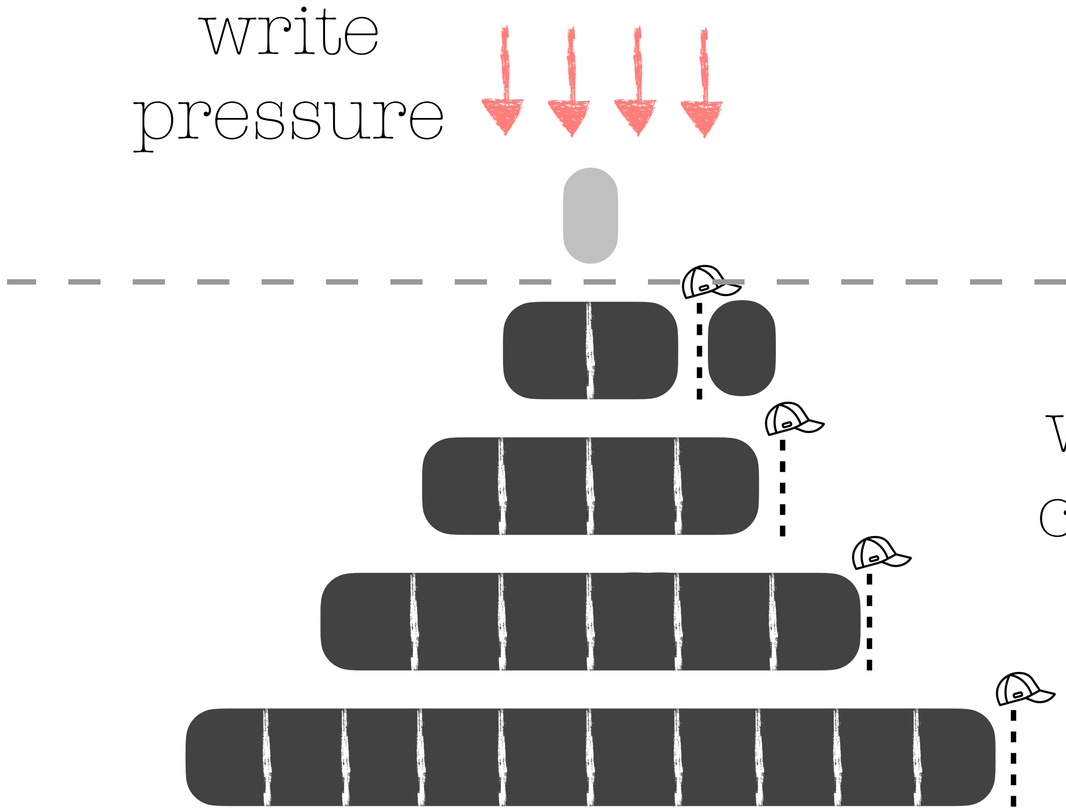
# Optimizing **Compactions**




Background Compactions

Compaction Priority

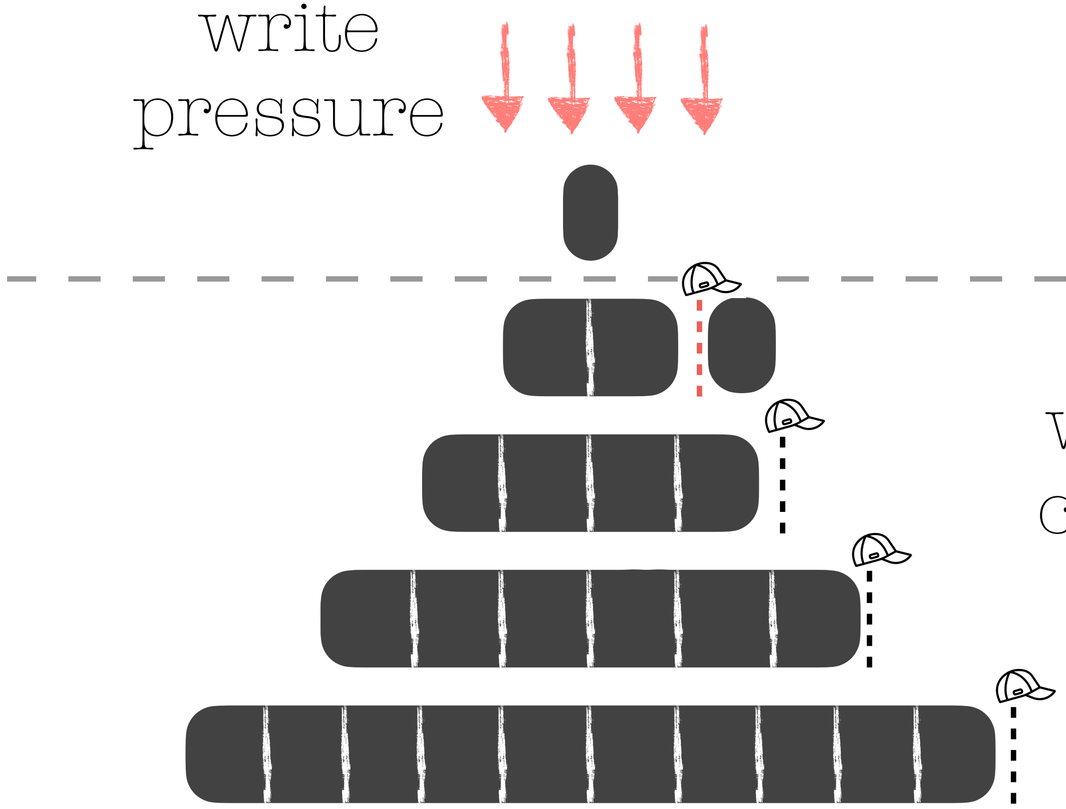
# Optimizing **Compactions**




Background Compactions

Compaction Priority

# Optimizing **Compactions**

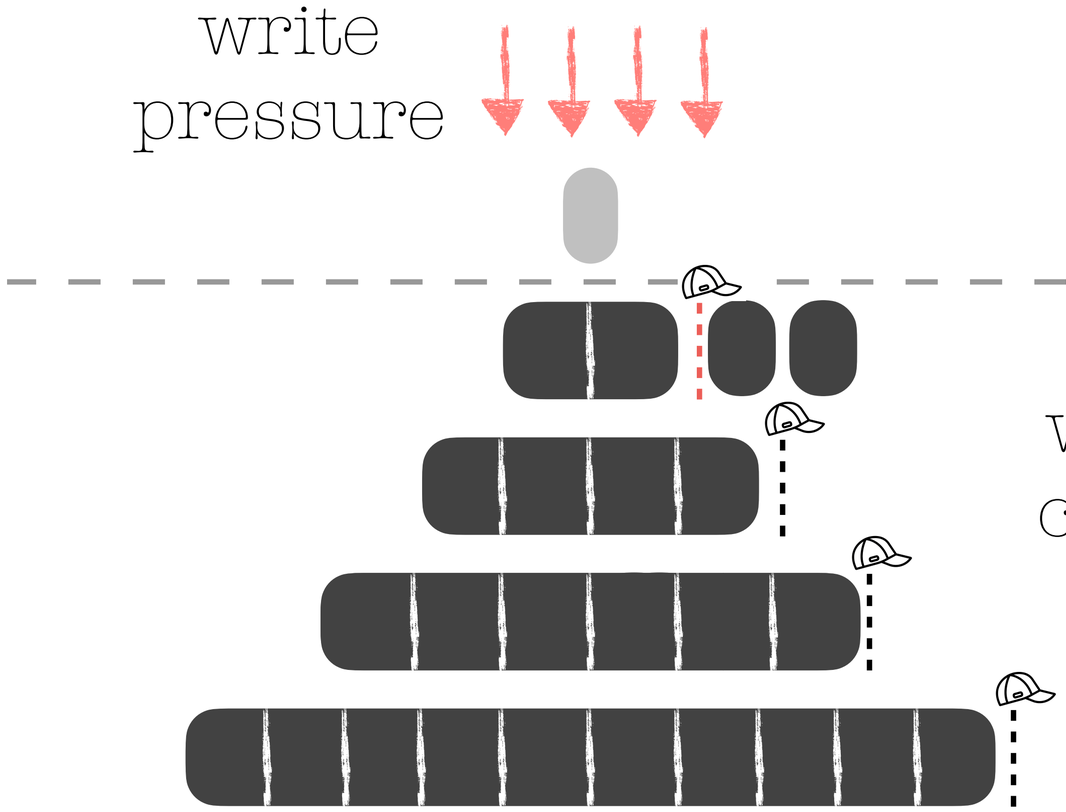



prioritize
writes over
compaction

Background
Compactions

Compaction
Priority

# Optimizing **Compactions**

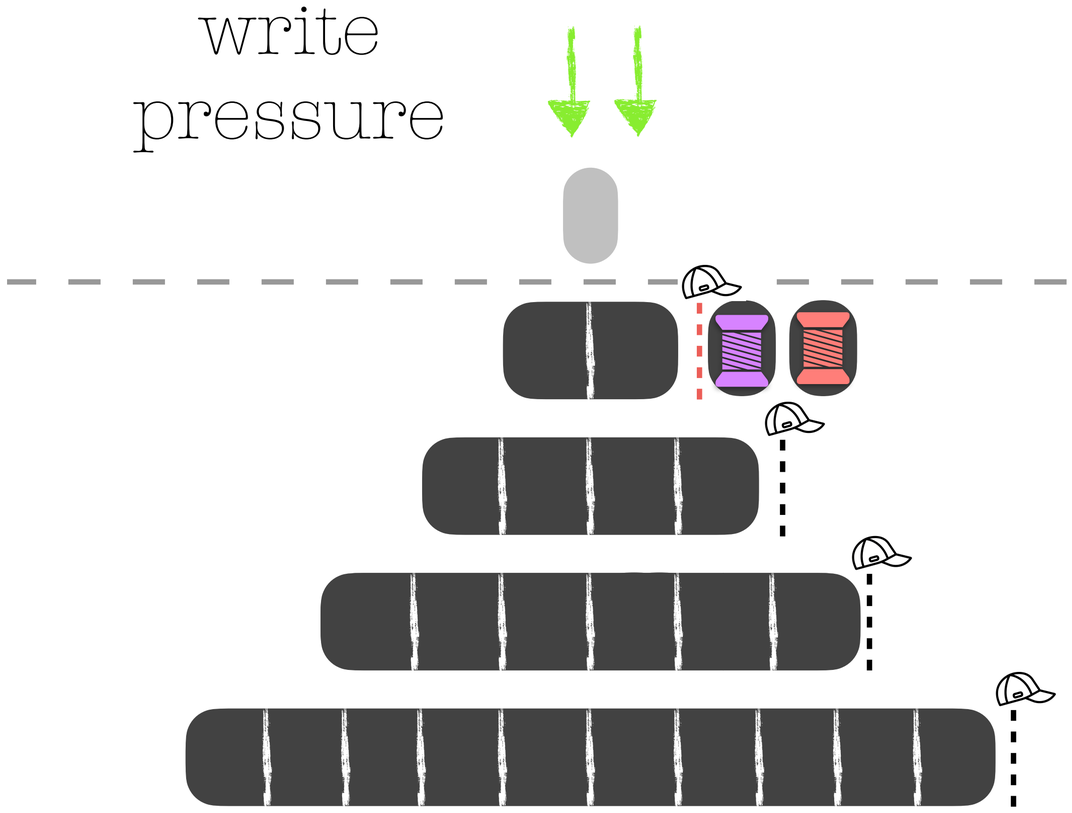



prioritize
writes over
compaction

Background
Compactions

Compaction
Priority

# Optimizing **Compactions**

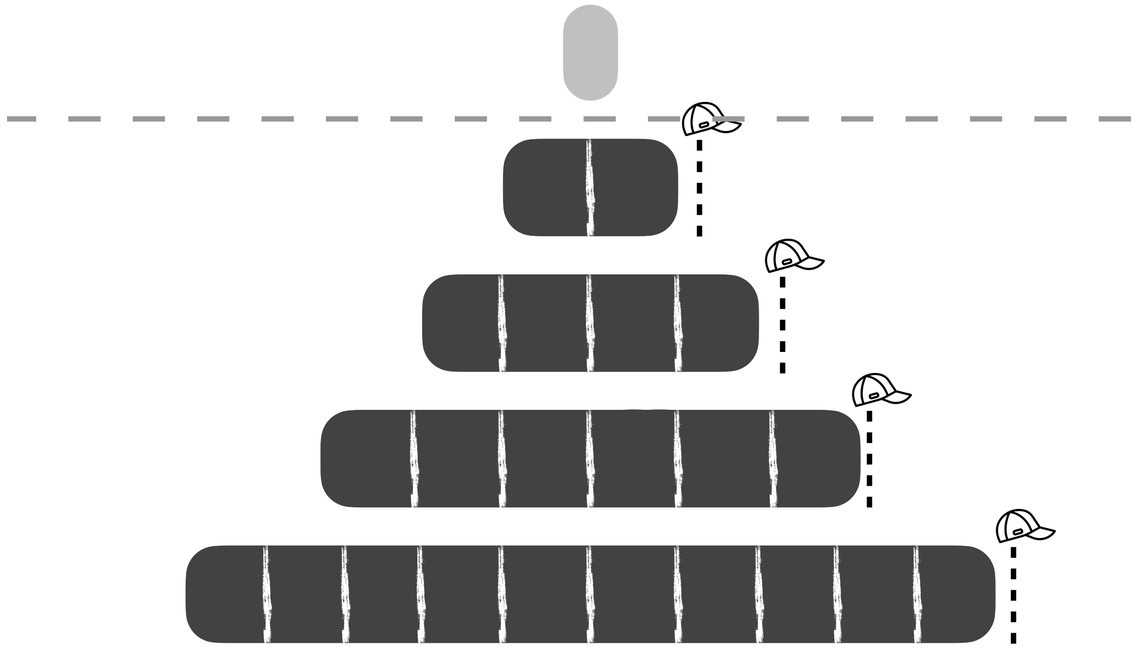



Background Compactions

Compaction Priority

- sustain heavy write bursts
- tree becomes out of shape

# Optimizing **Compactions**

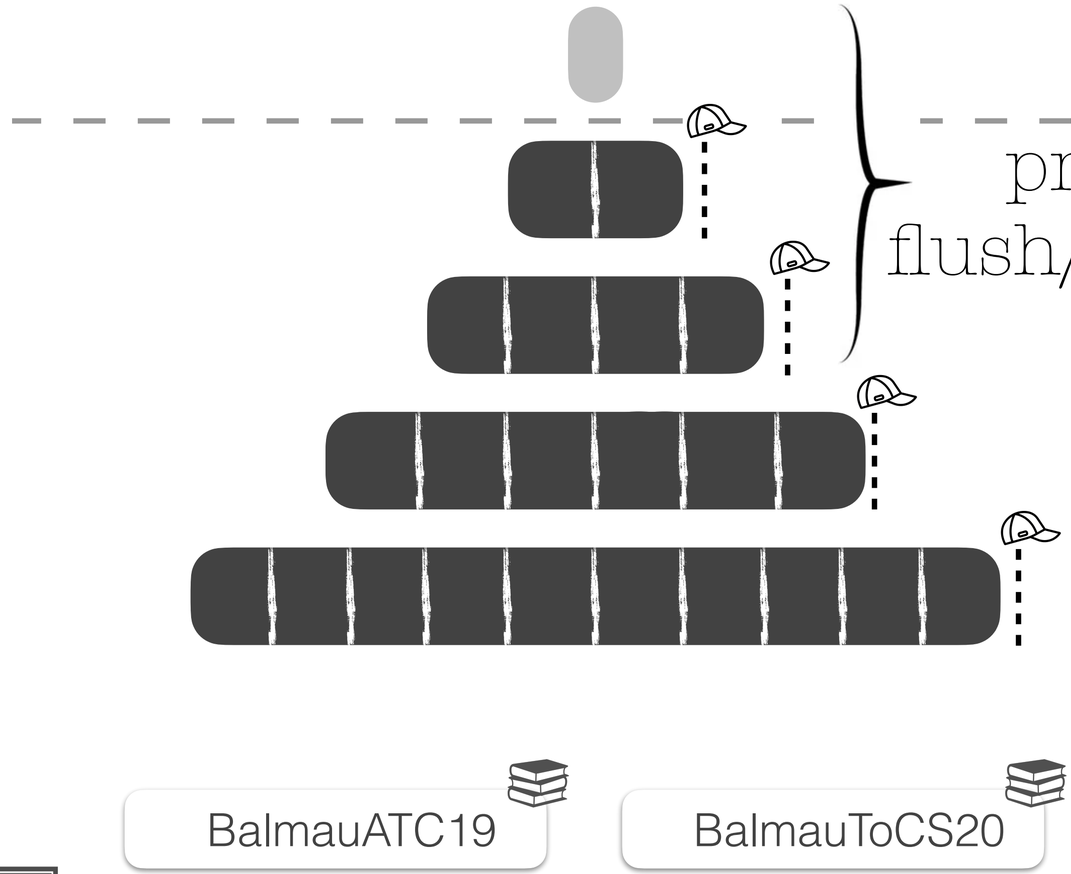


Background Compactions

Compaction Priority

I/O Scheduler

# Optimizing **Compactions**




Background Compactions

Compaction Priority

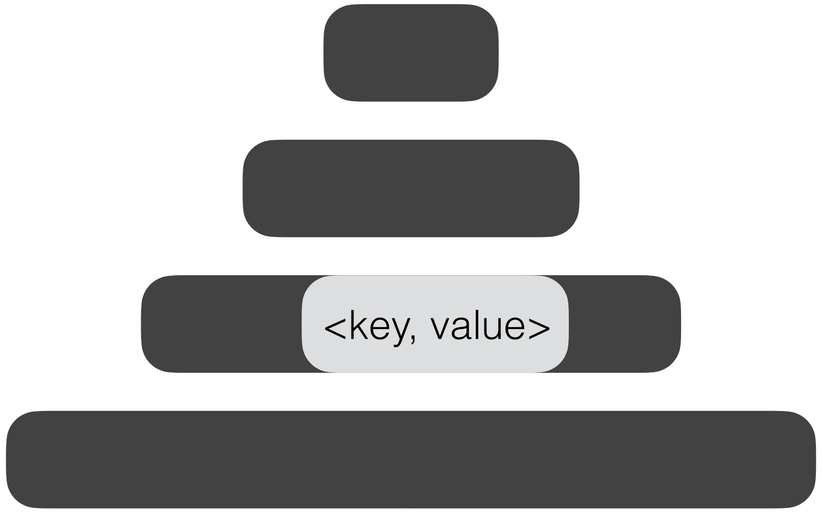
I/O Scheduler

- eliminates write stalls
- no unnecessary high-priority compactions in lower levels

BalmauATC19

BalmauToCS20

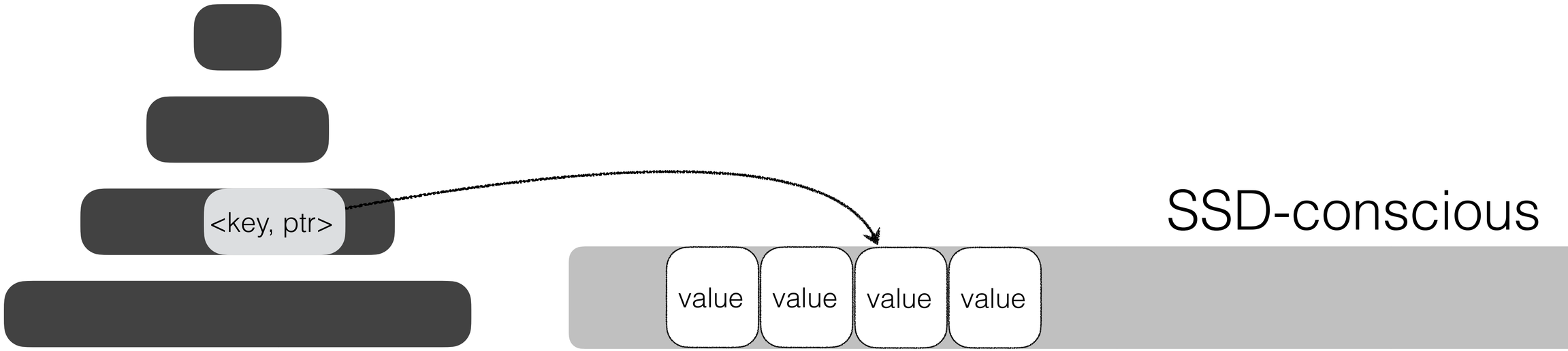
# **Data Placement** Variations




key-value separation

LuFAST16

# **Data Placement** Variations




key-value separation

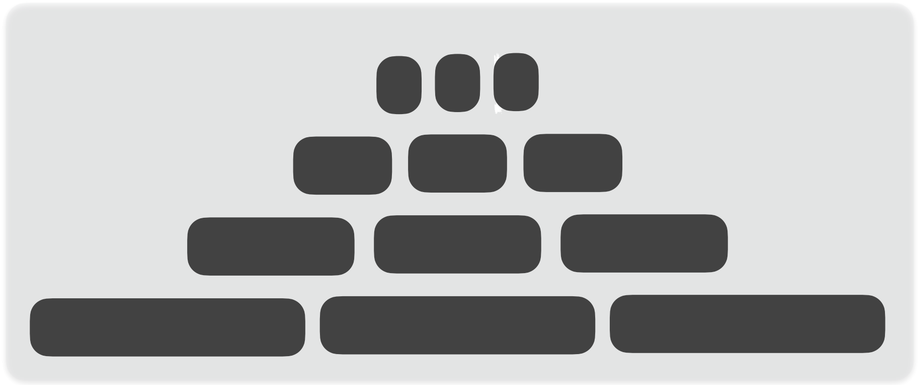
LuFAST16

- reduced write amplification
- better read performance

# **Data Placement** Variations

partitioning / sharding

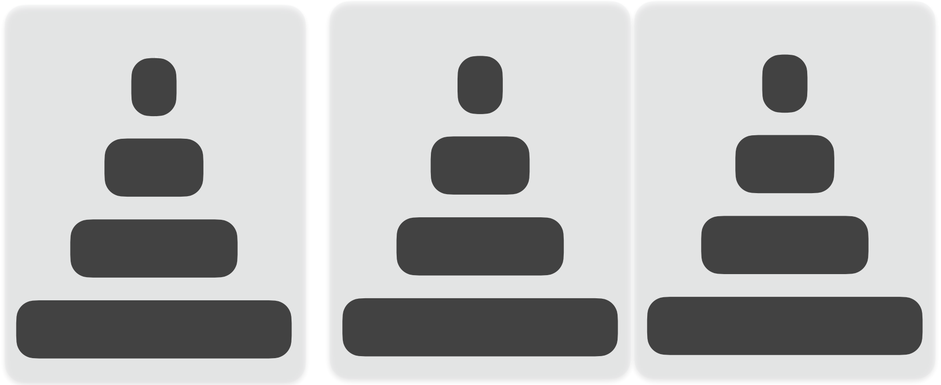
# **Data Placement** Variations



storage

partitioning

RajuSOSP17



storage-1 storage-2 storage-3

sharding

HuangSIGMOD21

- improved ingestion throughput
- reduced write amplification

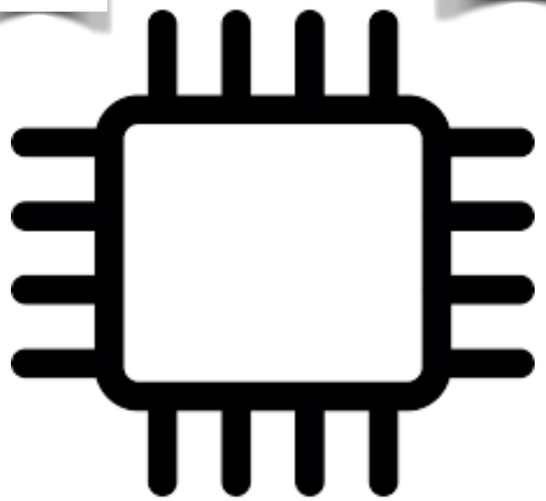
BOSTON UNIVERSITY

# Summary: **Ingestion Optimization**

data structure

size

flush strategy



main memory design elements

compaction design space

tuning compactions

data layout on storage

key-value separation

data partitioning

data sharding

hardware-conscious designs

BOSTON UNIVERSITY

# Outline

Part 1: **LSM Basics**



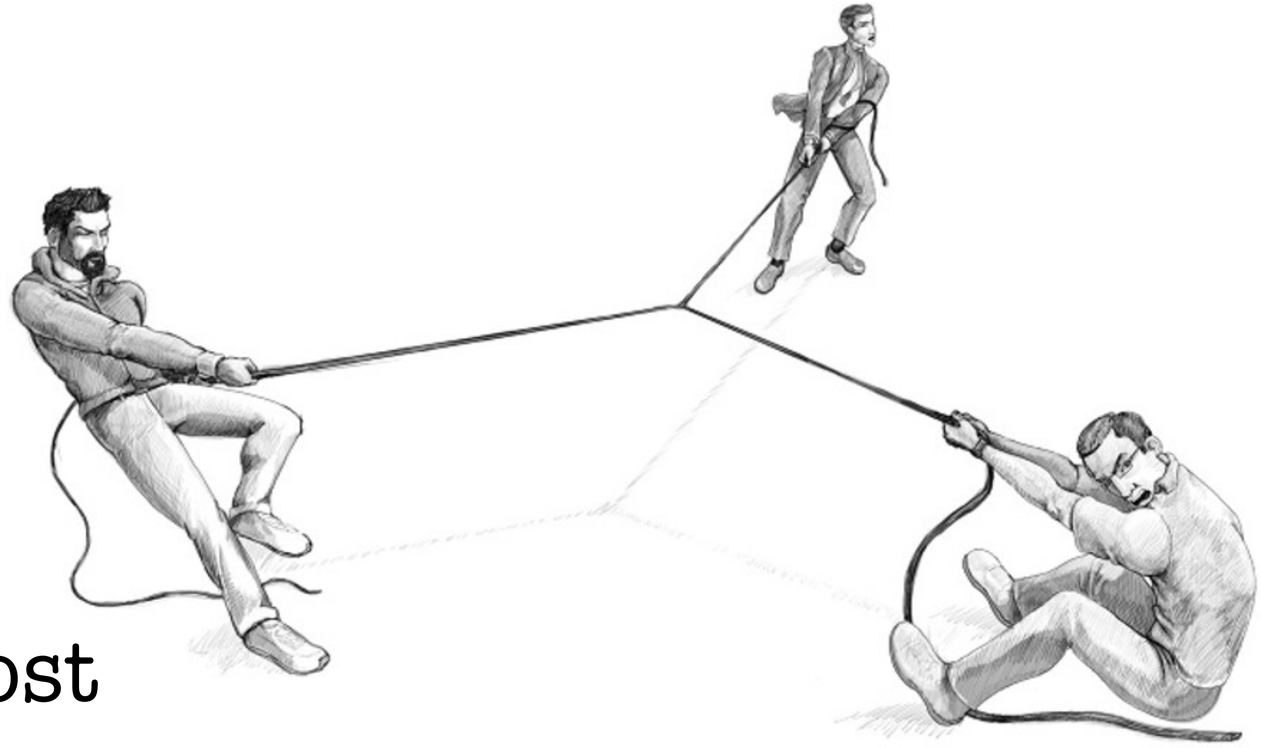
Part 2: **Optimizing Ingestion in LSMs**



Part 3: **Navigating the LSM Design Space**

# LSM **Design Space**

Update cost



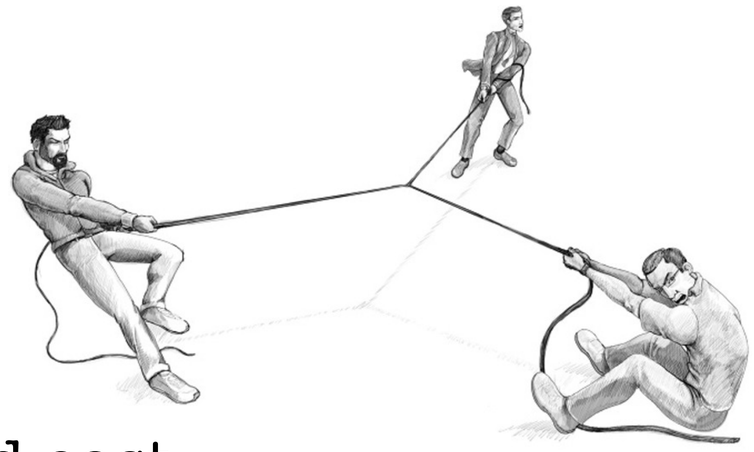
Read cost

Memory/space footprint

BOSTON UNIVERSITY

AthanassoulisEDBT16

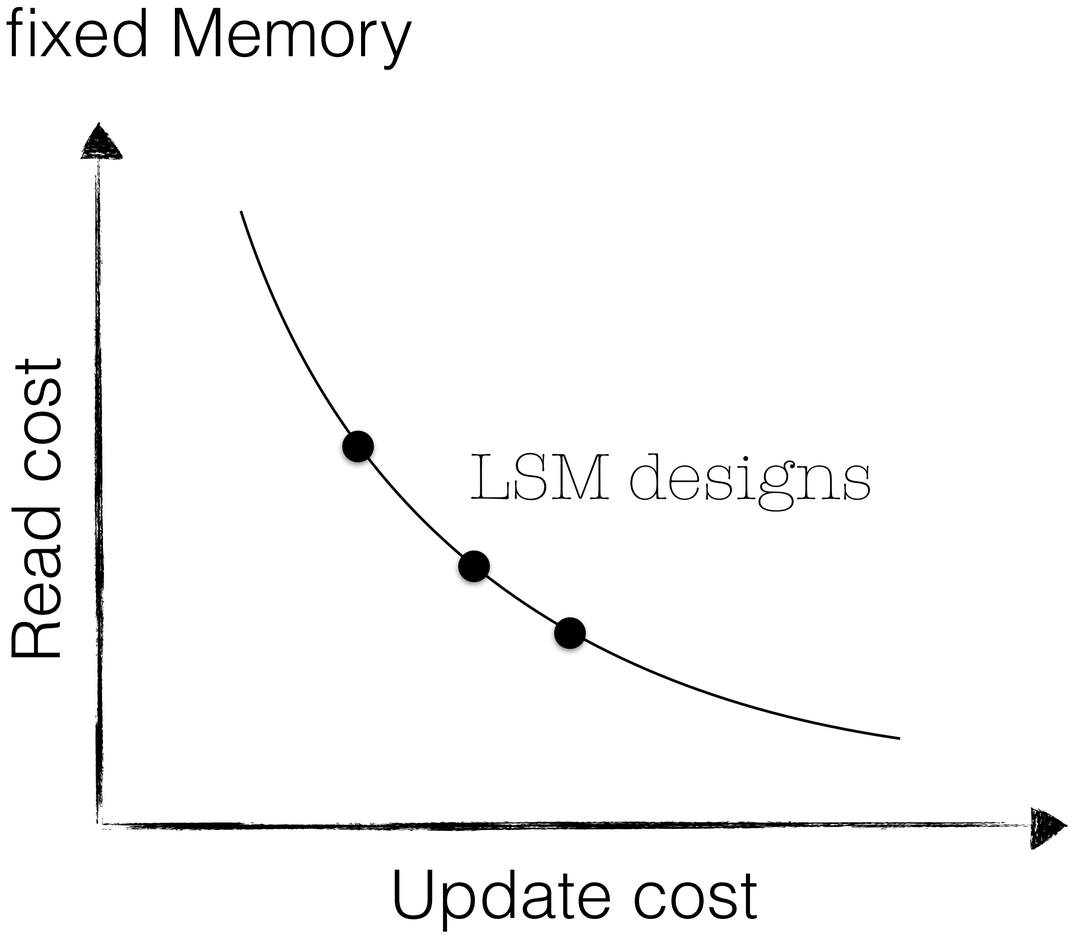
# LSM **Design Space**



Update cost

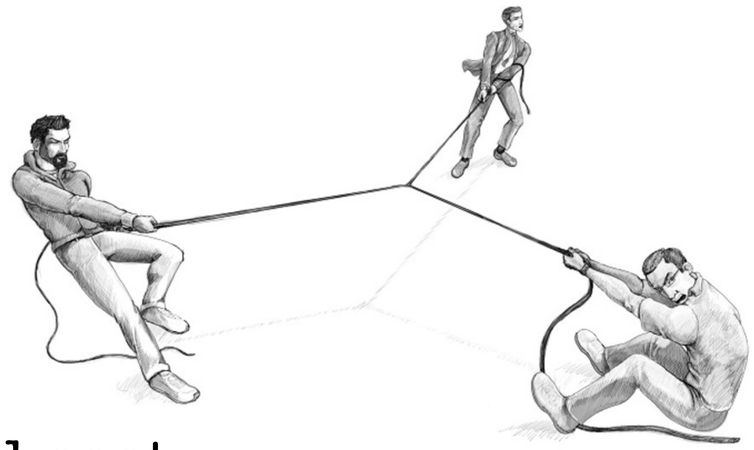
Read cost

Memory/space footprint



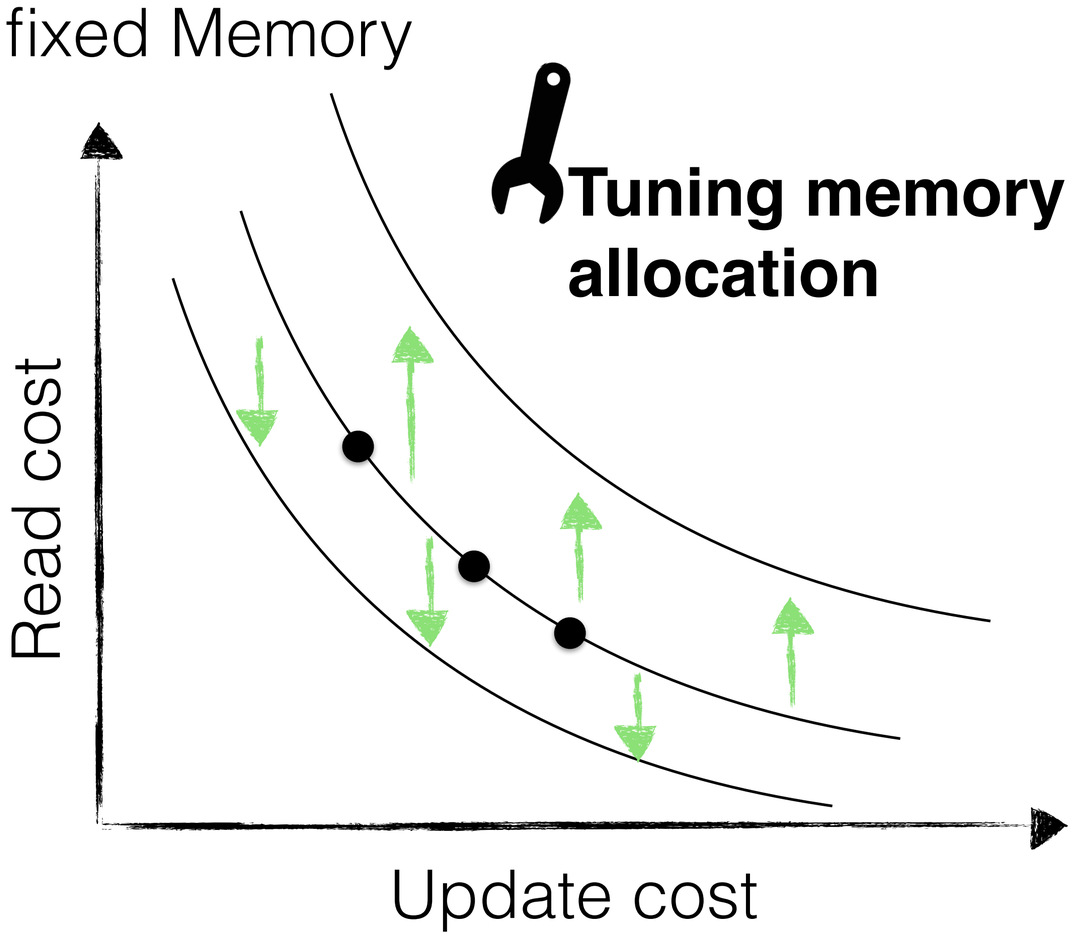
# LSM **Design Space**

Update cost

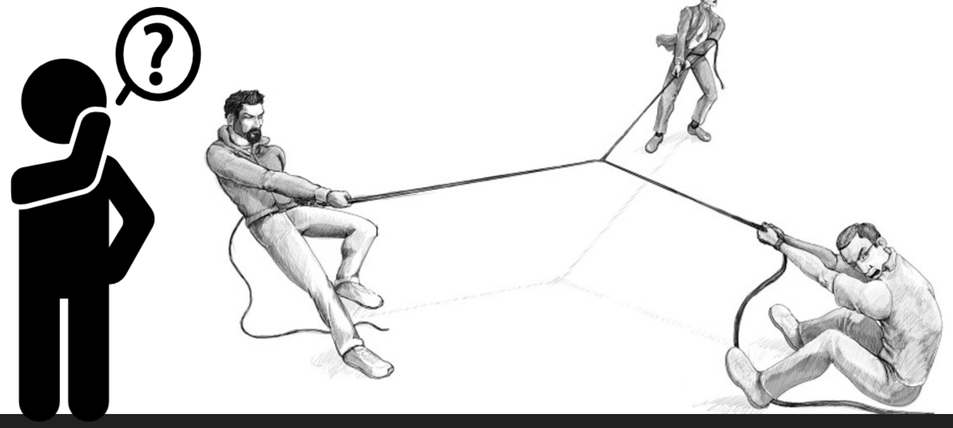


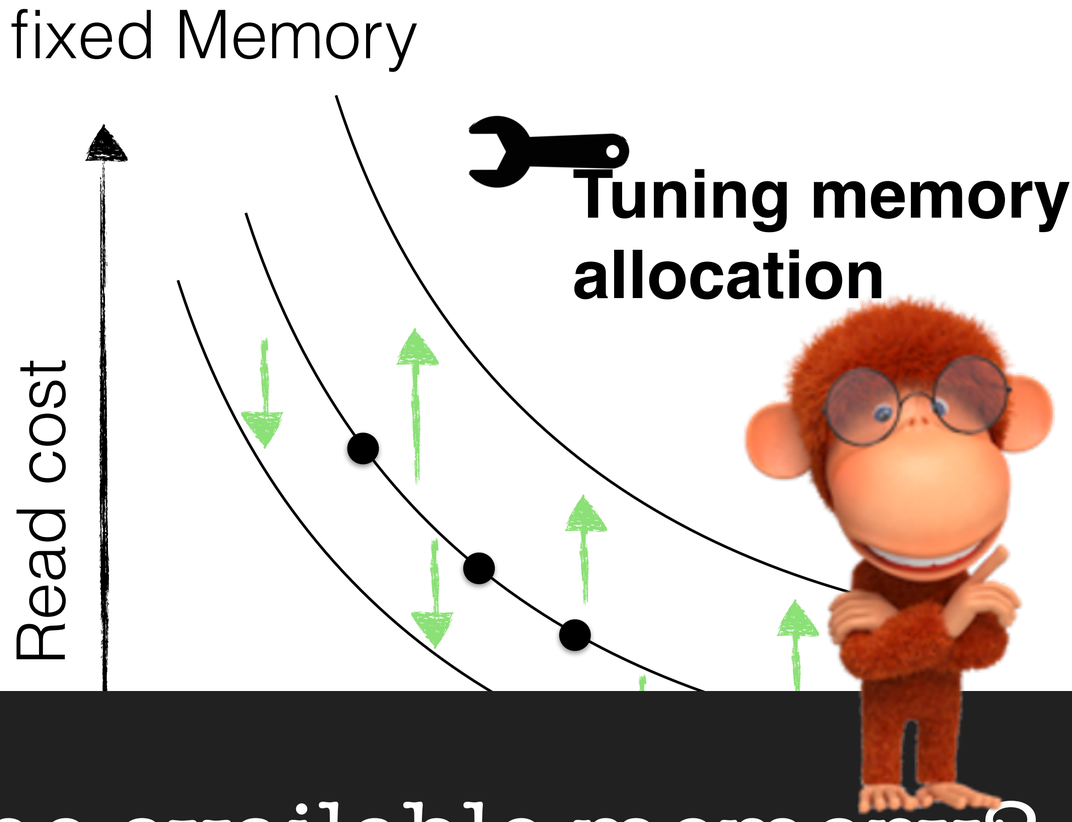
Read cost

Memory/space footprint

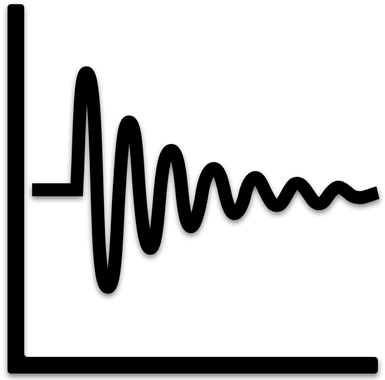


# LSM **Design Space**

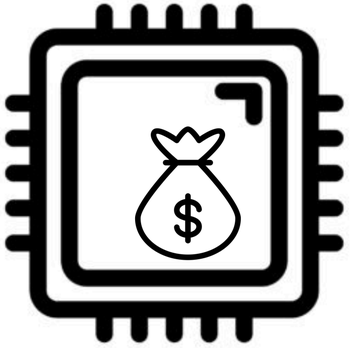


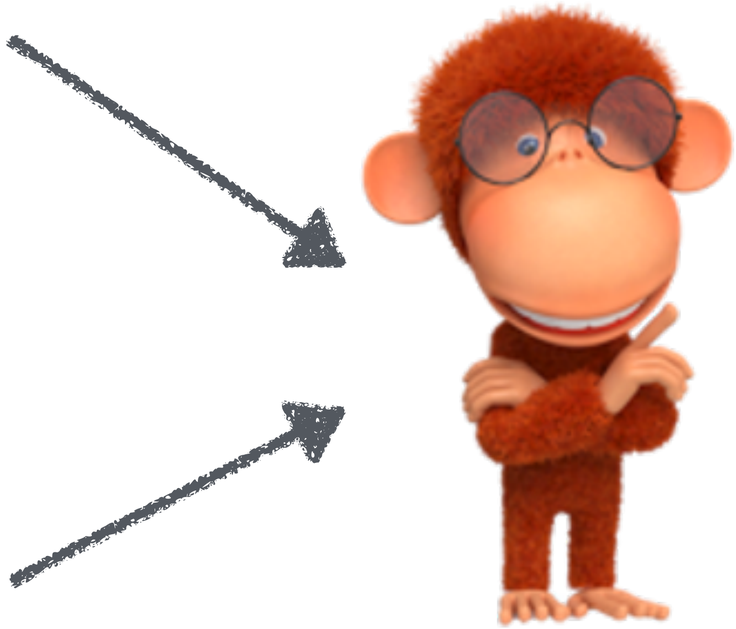

How to optimally allocate the available memory?



workload

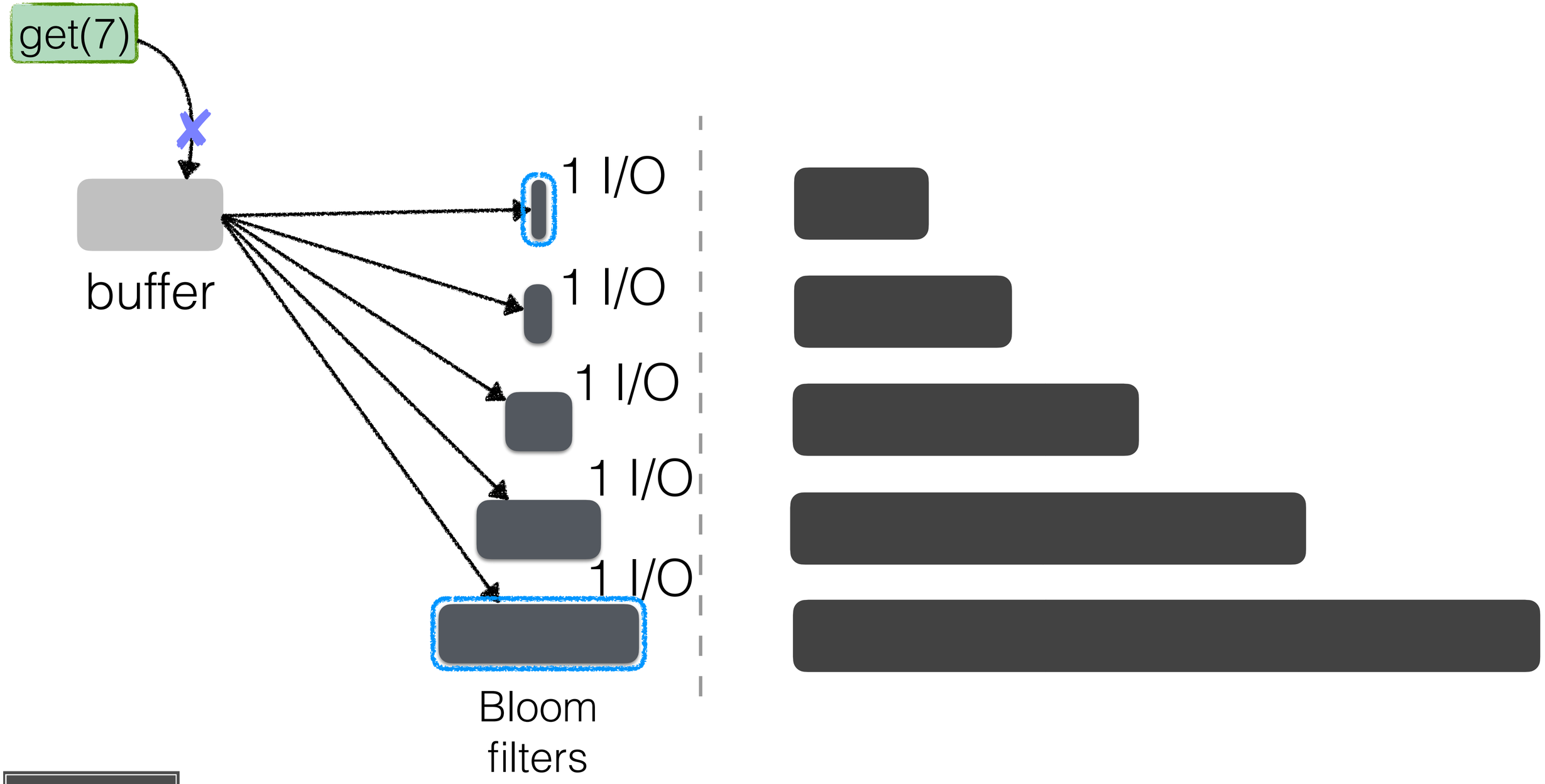


memory budget

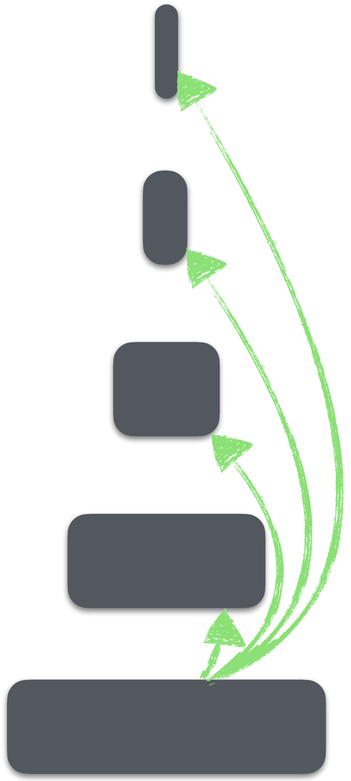


How to allocate memory between buffer and BF

How to allocate memory among BFs in LSM


get(7)
buffer
1 I/O
1 I/O
1 I/O
1 I/O
1 I/O
Bloom
filters

Bloom filters



FPR

Monkey FPR

$$\phi > \phi_1 = \phi_0/T^4$$

$$\phi > \phi_2 = \phi_0/T^3$$

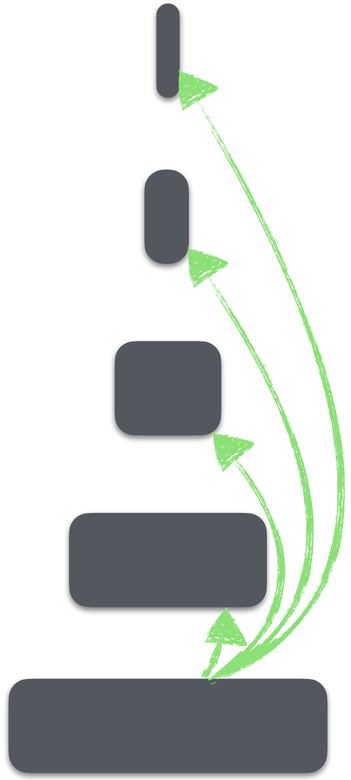
$$\phi > \phi_3 = \phi_0/T^2$$

$$\phi > \phi_4 = \phi_0/T$$

$$\phi < \phi_5 = \phi_0$$

exponentially decreasing

Bloom filters



| FPR | | Monkey FPR |
|---|---|---|
| $\phi$ | $>$ | $\phi_1 = \phi_0/T^4$ |
| $\phi$ | $>$ | $\phi_2 = \phi_0/T^3$ |
| $\phi$ | $>$ | $\phi_3 = \phi_0/T^2$ |
| $\phi$ | $>$ | $\phi_4 = \phi_0/T$ |
| $\phi$ | $<$ | $\phi_5 = \phi_0$ |

exponentially decreasing

$L \cdot \phi$ $\quad$ $\sum_i \phi_i = c \cdot \phi_0$

point lookup cost

$$\mathcal{O}(L \cdot \phi)$$

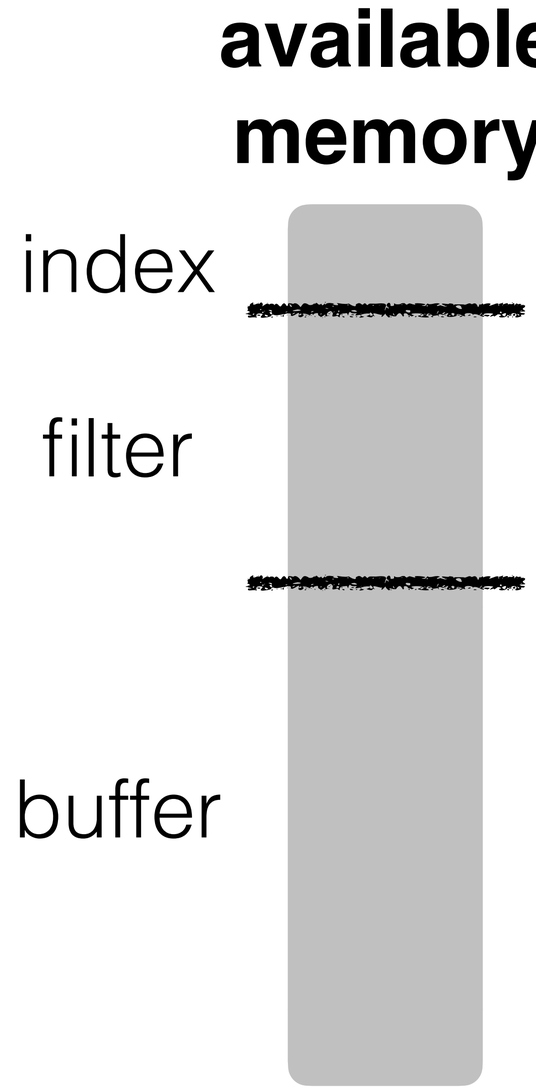
$$\vee$$

$$\mathcal{O}(c \cdot \phi_0)$$

BOSTON UNIVERSITY

DayanSIGMOD17

# The **Optimal** Memory Allocation

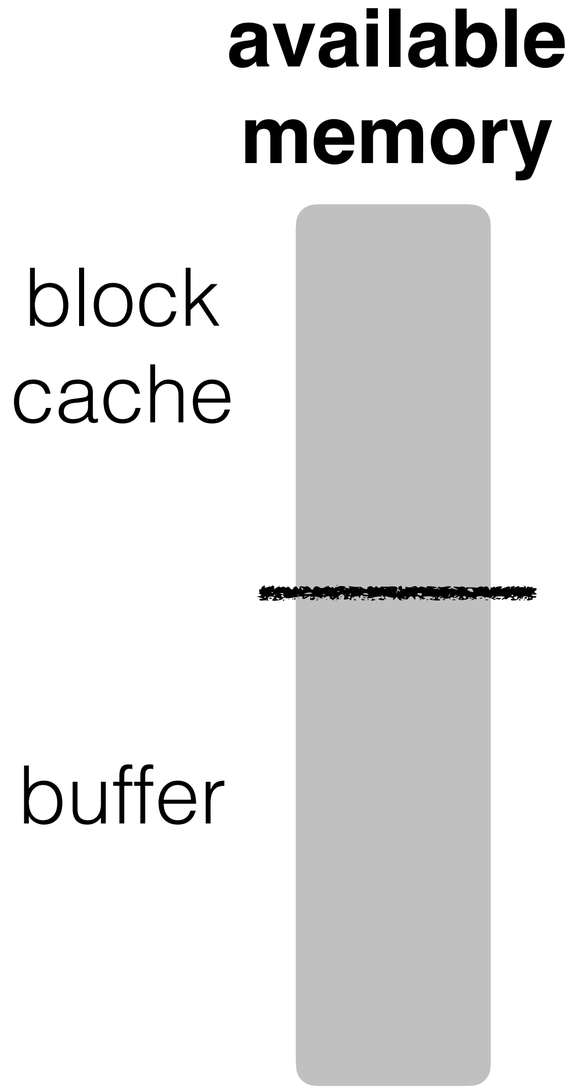



read_cost(N, mem_idx, mem_filt, mem_buf)

write_cost(N, mem_buf)

minimize overall cost given workload
(e.g., read ratio vs. write ratio)

# The **Optimal** Memory Allocation




read_cost(N, mem_block_cache)

write_cost(N, mem_buf)

minimize overall cost given workload
(e.g., read ratio vs. write ratio)



LuoSIGMOD20 LuoVLDB20

[Open Problem]

# The **Optimal** Memory Allocation

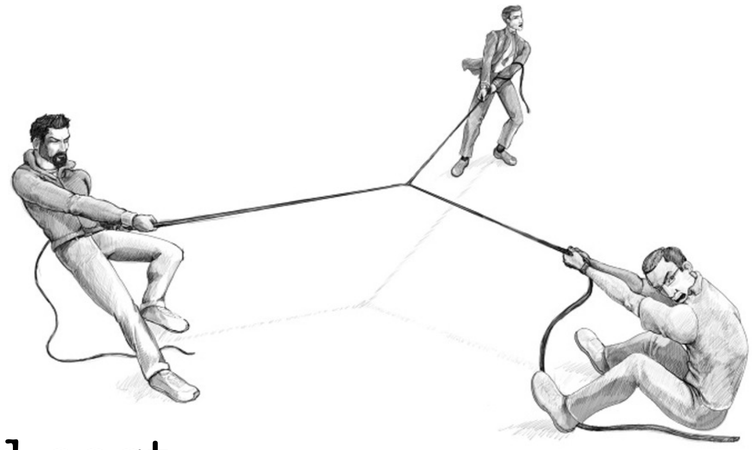
**available memory**

index

filter

block cache

buffer



Update cost

Read cost

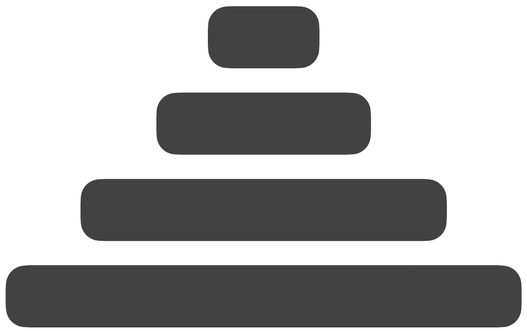
Memory/space footprint

Holistic memory allocation & tuning

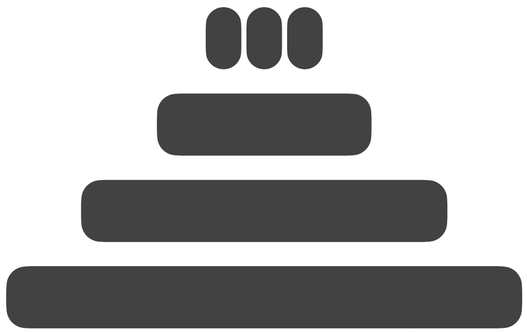
BOSTON UNIVERSITY

# Storage Layer **Design Continuum**

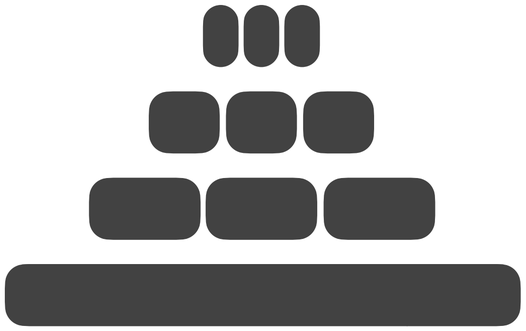
leveling



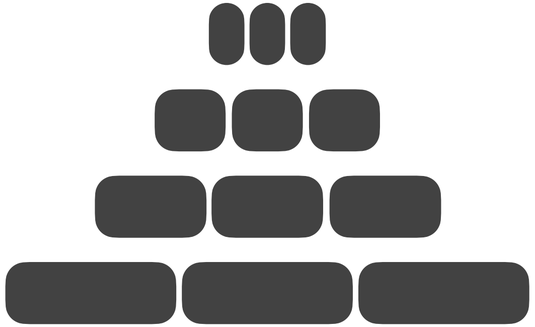
1-leveling



L-leveling



tiering

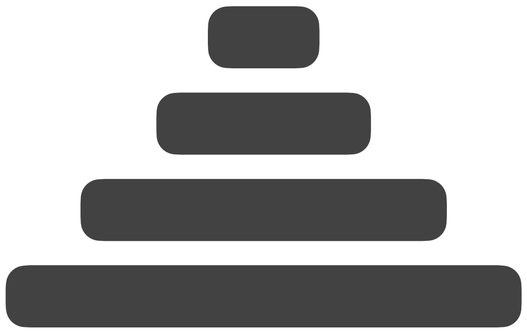


Any design can be defined by the tuple-set: $(\mathtt{T}, \mathtt{i})$

# Storage Layer **Design Continuum**

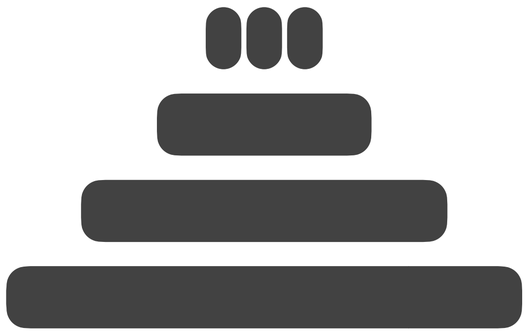
leveling

(T, 0)



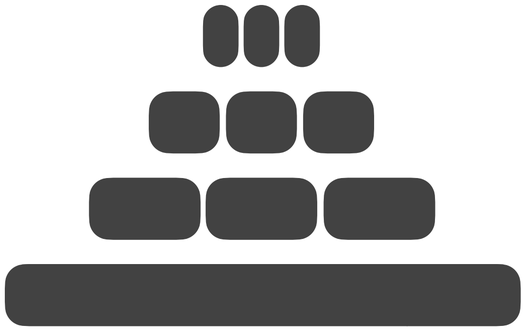
1-leveling

(T, 1)



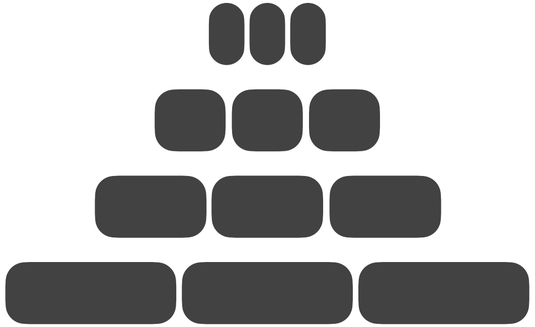
L-leveling

(T, L)



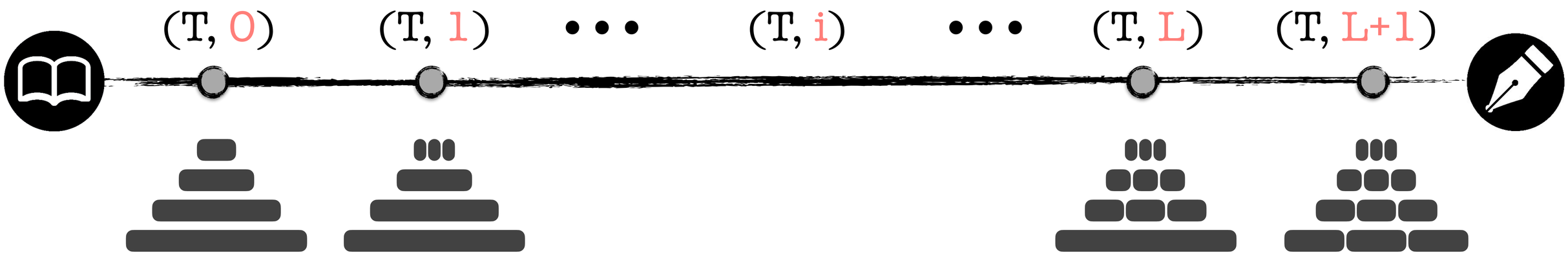
tiering

(T, L+1)

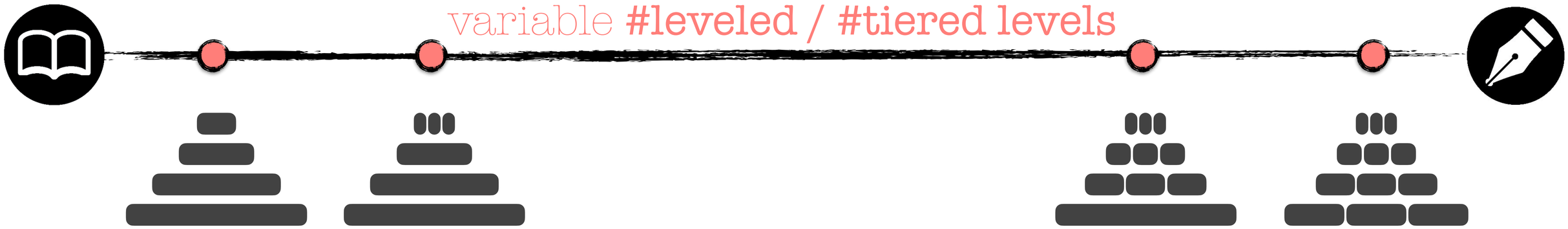


Any design can be defined by the tuple-set: (T, i)

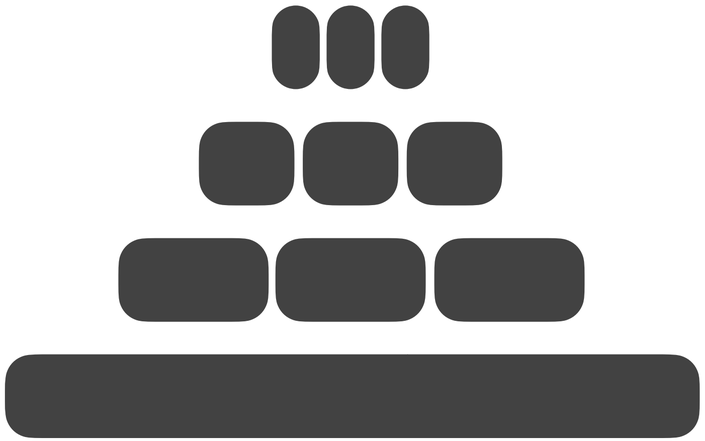
# Storage Layer **Design Continuum**



# Storage Layer **Design Continuum**



# Storage Layer **Design Continuum**

size ratio

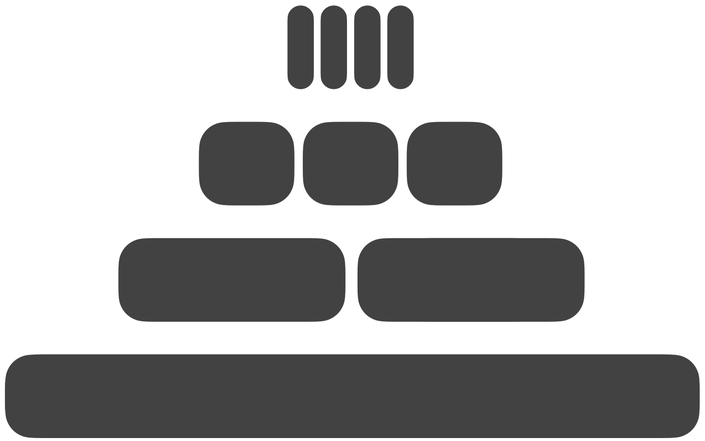
T

T

T

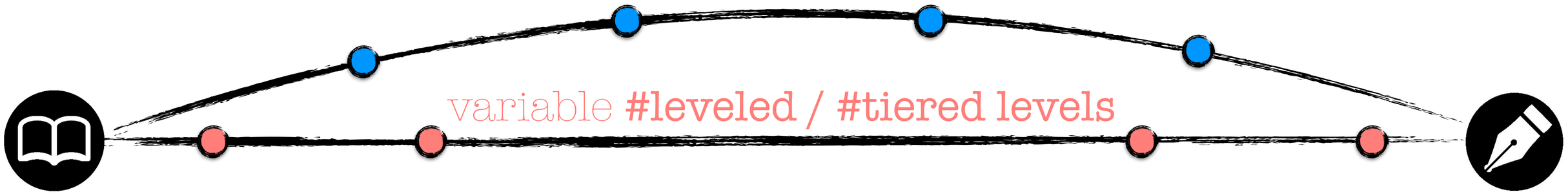
T

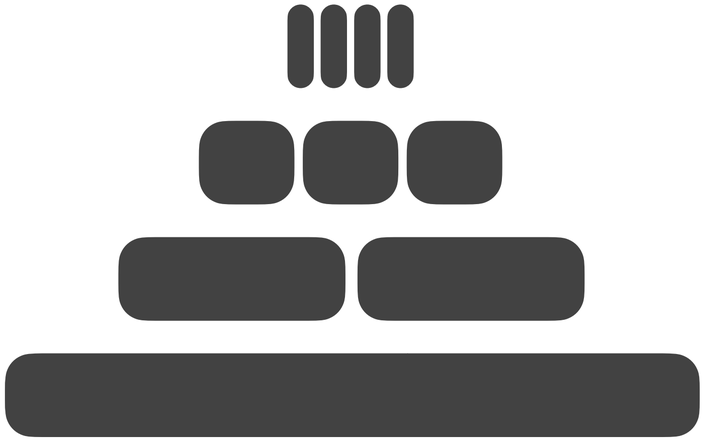
# Storage Layer **Design Continuum**

| size ratio | #runs |
|---|---|
| T | 4 |
| T | 3 |
| T | 2 |
| T | 1 |

# Storage Layer **Design Continuum**

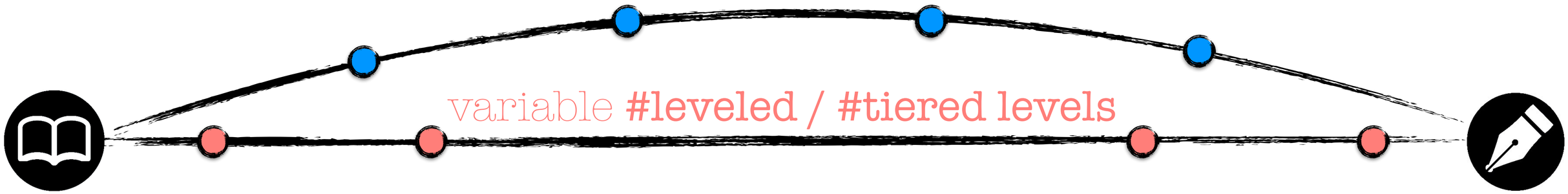


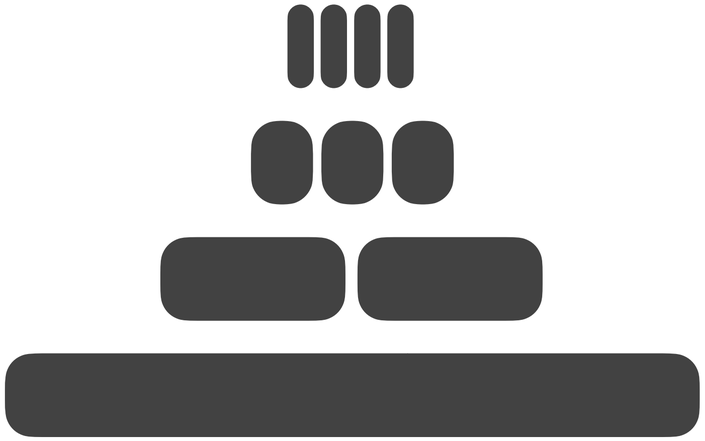
| size ratio | #runs |
|---|---|
| T | 4 |
| T | 3 |
| T | 2 |
| T | 1 |

# Storage Layer **Design Continuum**

variable **#runs/level**

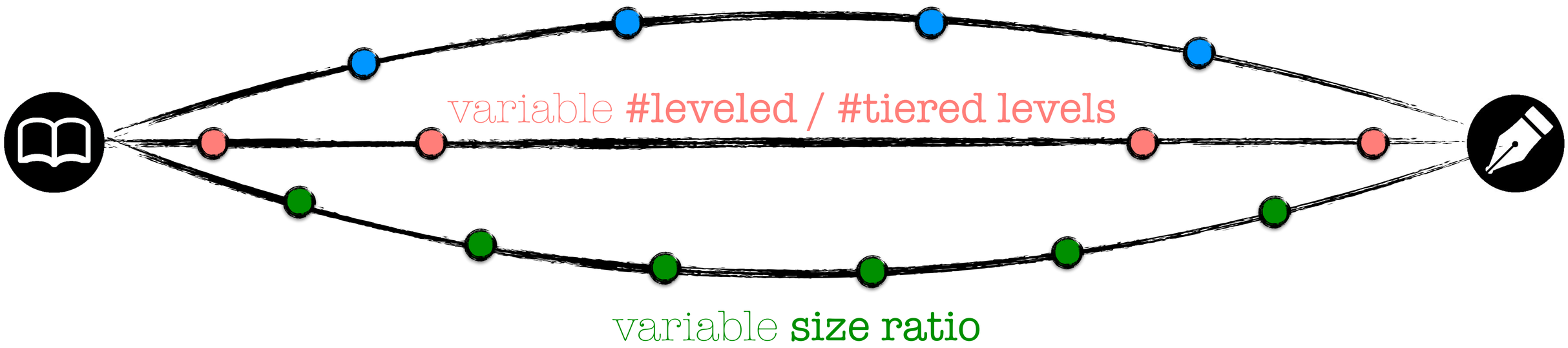


variable **#leveled / #tiered levels**

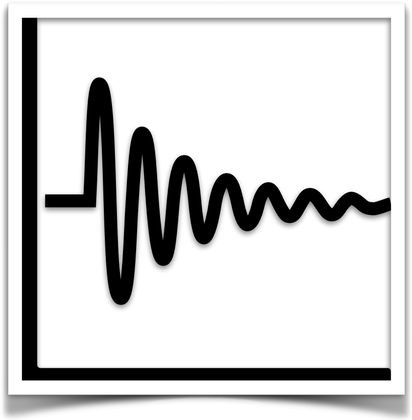


| size ratio | #runs |
|---|---|
| 2 | 4 |
| 2.5 | 3 |
| 3 | 2 |
| 4 | 1 |

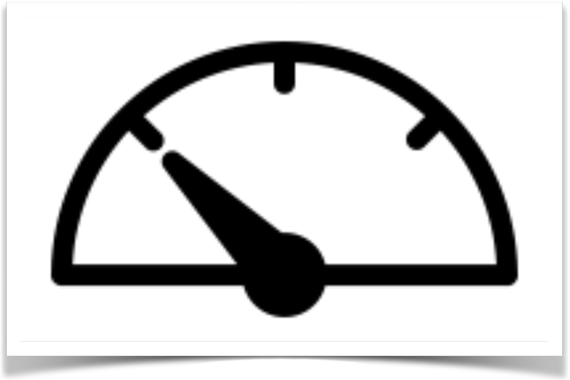
# Storage Layer **Design Continuum**



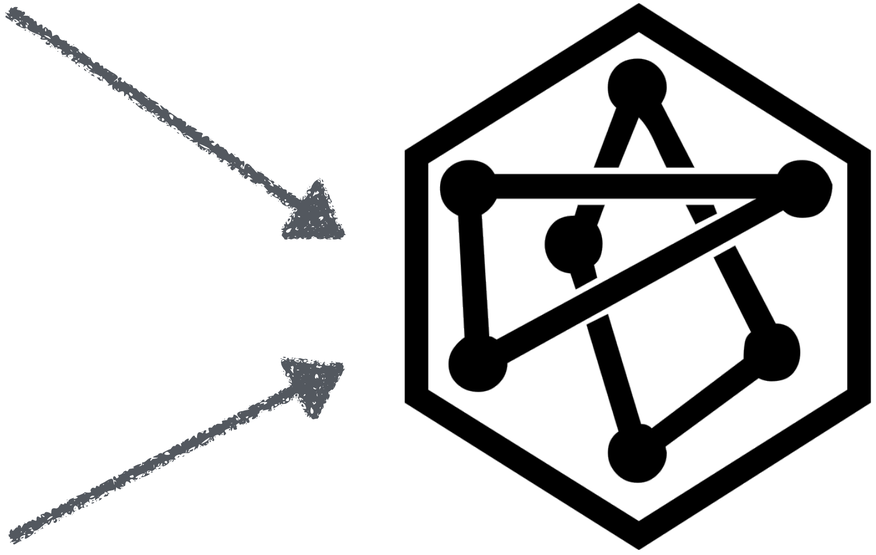

The LSM storage layer design continuum



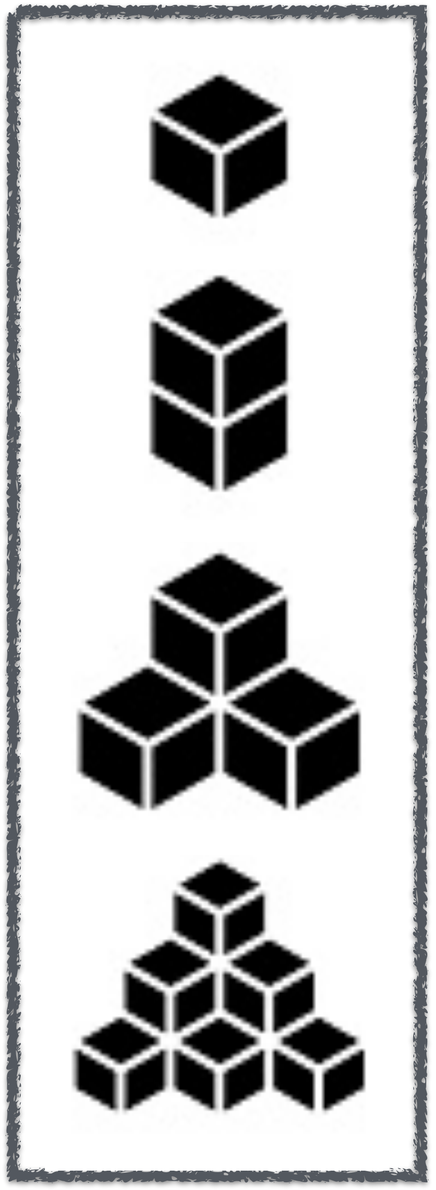
workload



performance target

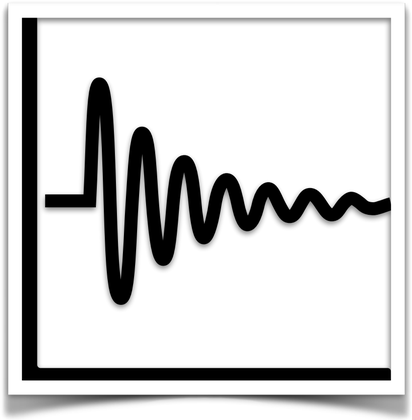


performance modeling

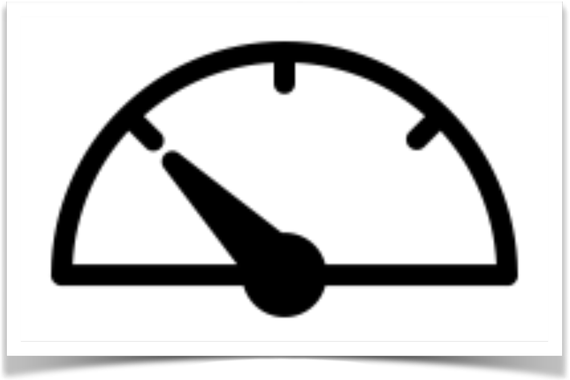


LSM designs

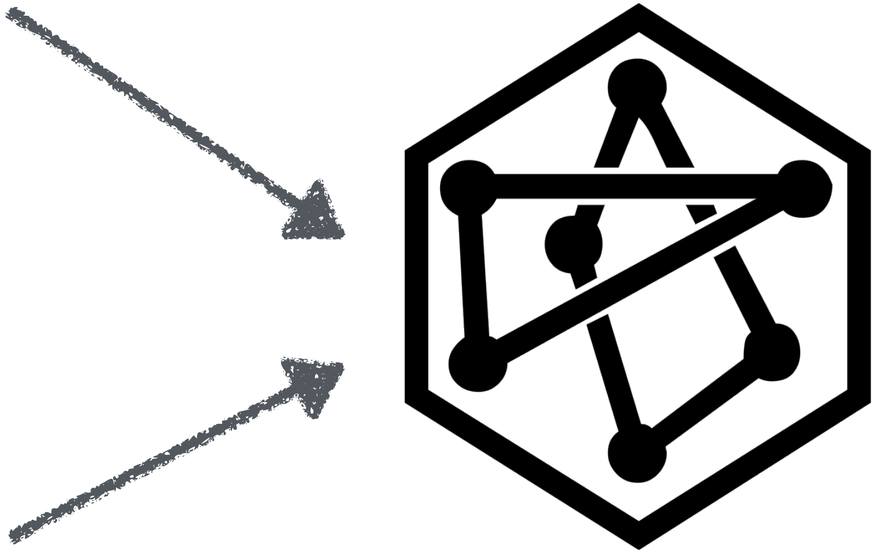
BOSTON UNIVERSITY



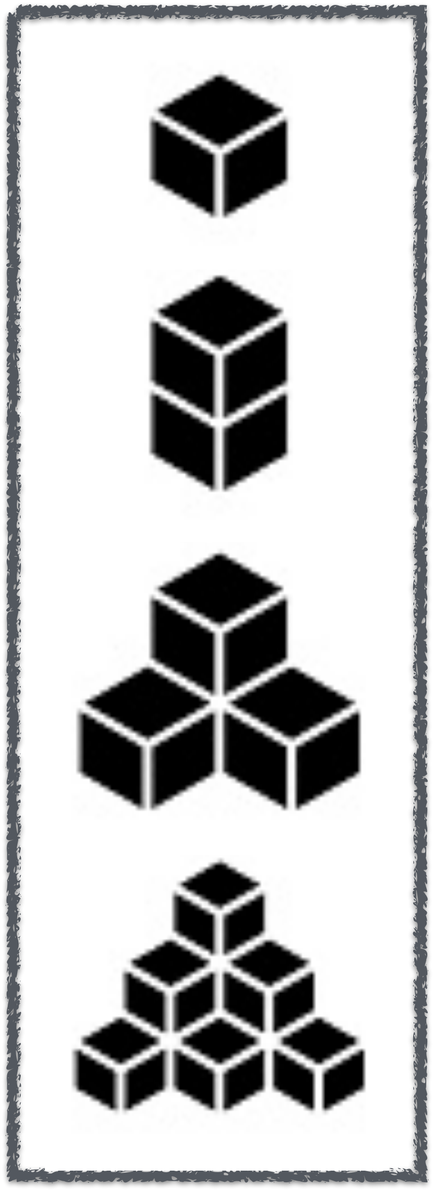
workload



performance target



**worst-case** performance modeling

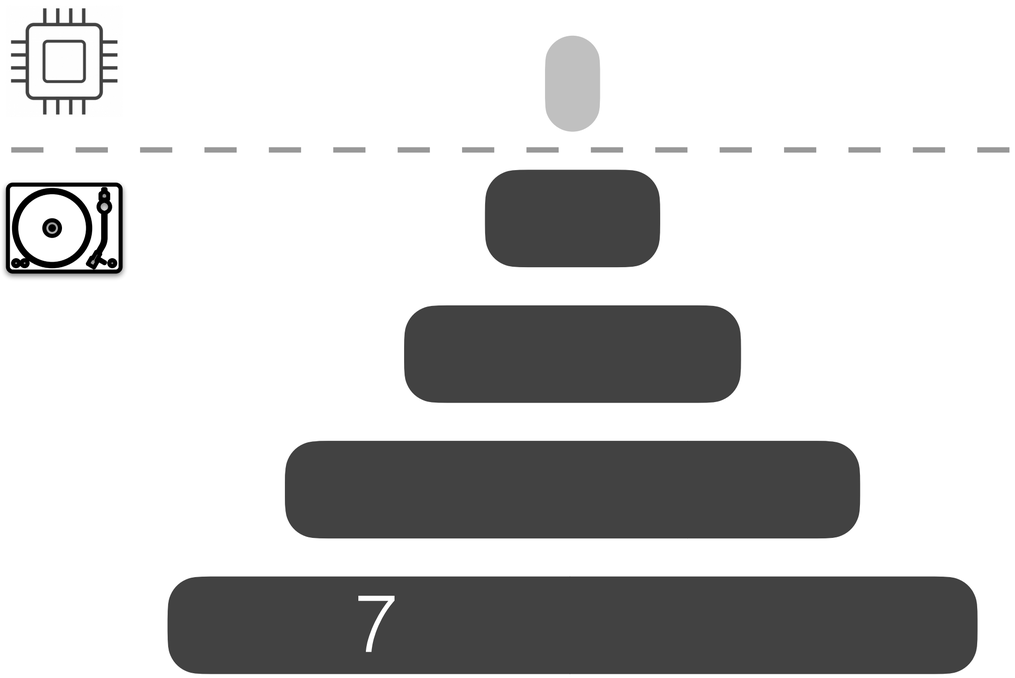


LSM designs

BOSTON UNIVERSITY

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

# **worst-case** performance modeling

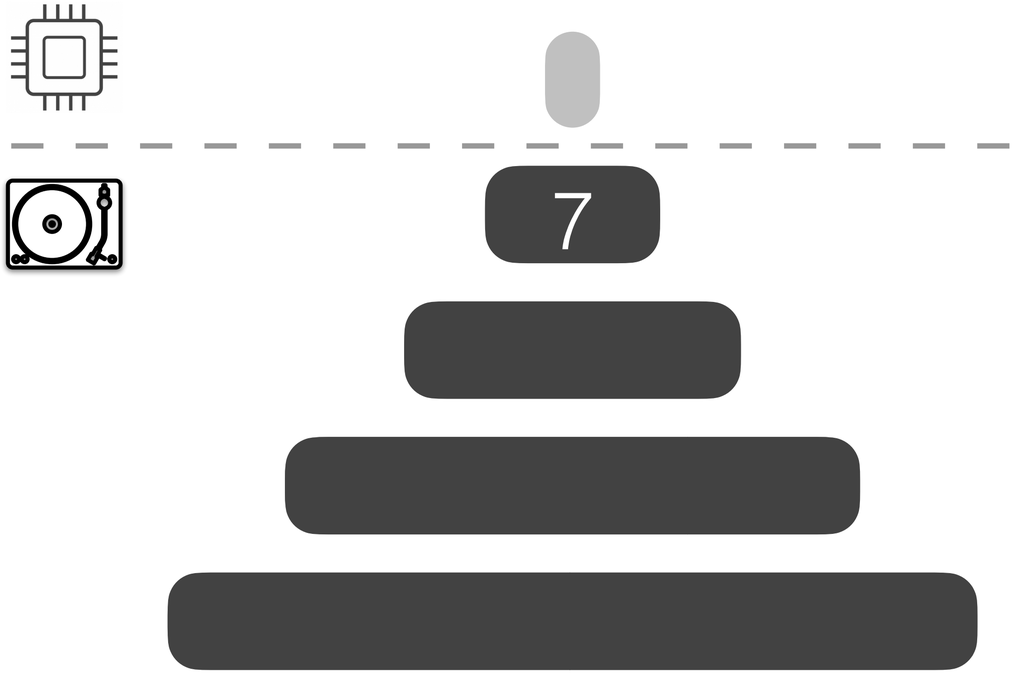



**worst-case** read cost: $1 + \sum_{i=1}^{L-1} \phi_i$

BOSTON UNIVERSITY

$P$: pages in buffer
$B$: entries/page
$L$: #levels
$T$: size ratio
$N$: #entries
$\phi$ : FPR of BF

**worst-case**
performance
modeling



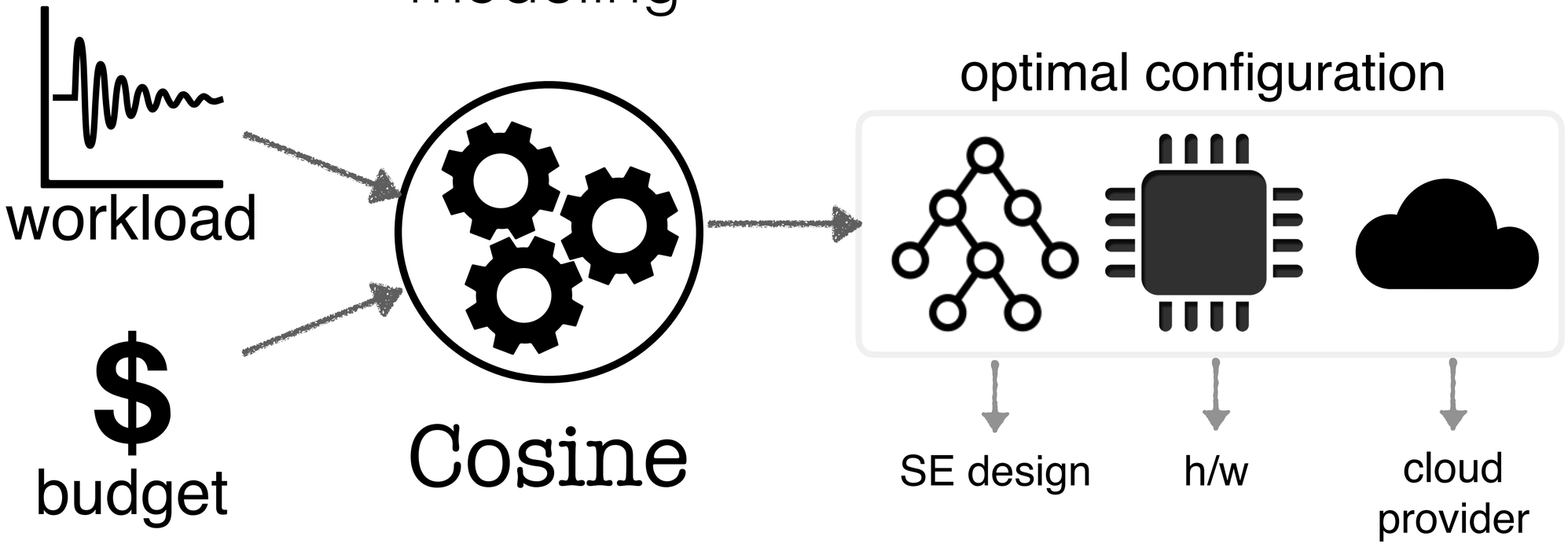

**worst-case** read cost: $1 + \sum_{i=1}^{L-1} \phi_i$

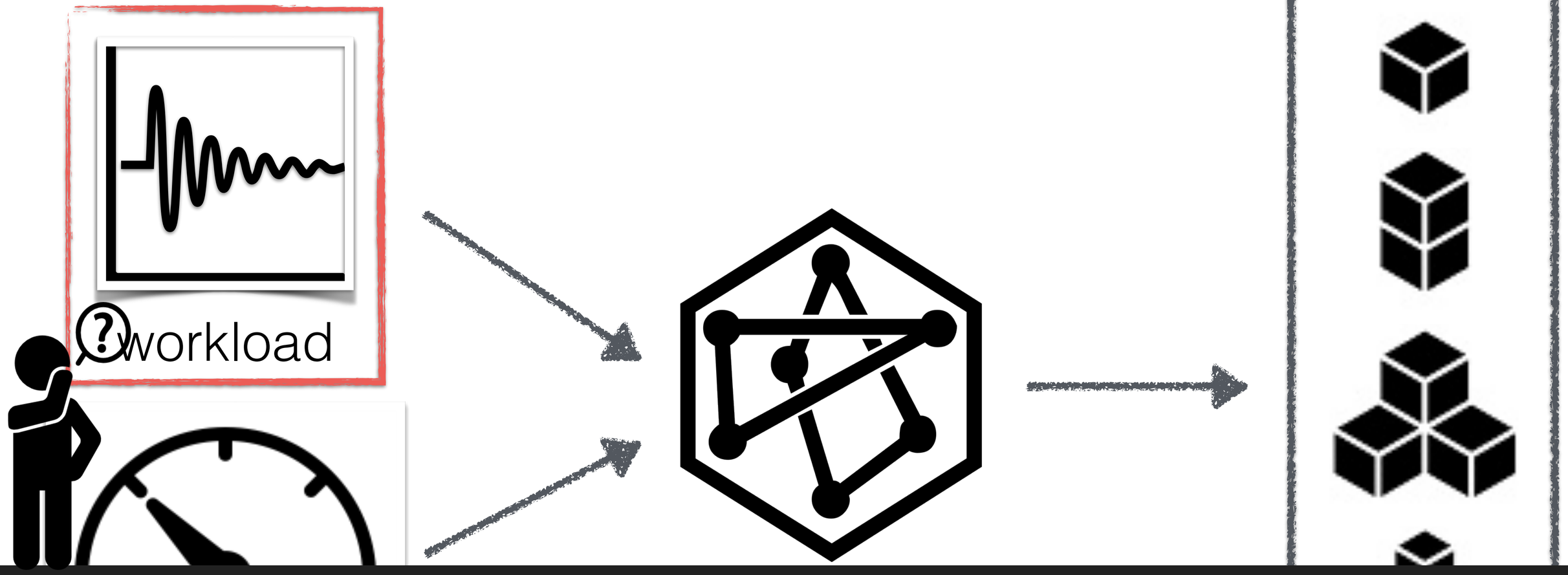
**average-case**
performance
modeling

$$\sum_{i=1}^{L} (\mathbb{P}[\text{query in } L_i] \cdot (1 + \sum_{j=1}^{i-1} \phi_i))$$

BOSTON UNIVERSITY

**average-case**
performance
modeling

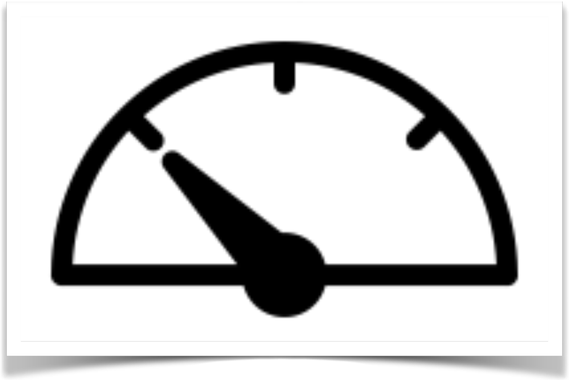




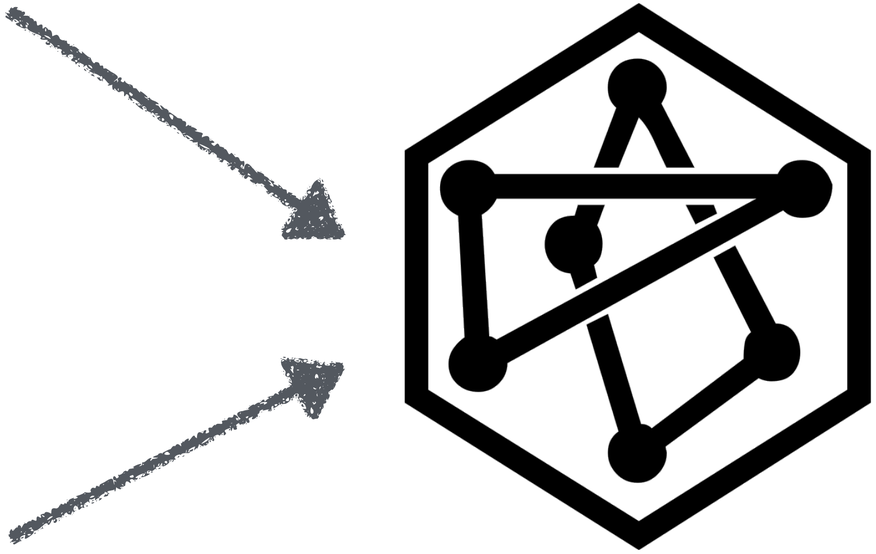

What if the workload changes?

ESM designs

workload

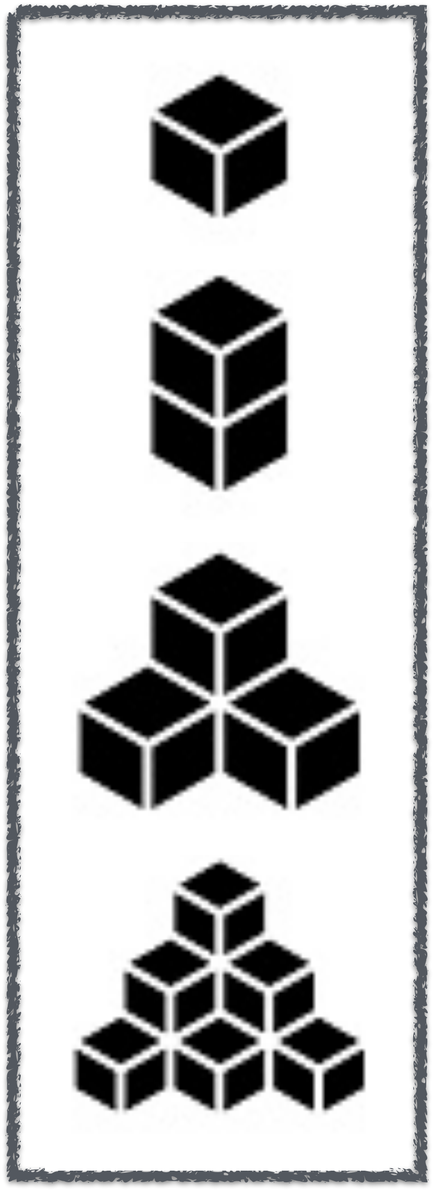


performance target



**average-case**

performance modeling



LSM designs

BOSTON UNIVERSITY

# Tuning

workload
hardware
design

optimal
tuning

given workload $\mathcal{W}$, dataset $\mathcal{D}$, find a tuning $\mathcal{T}^{\text{opt}}$:

$$\mathcal{T}^{\text{opt}} = \text{argmin}_{\mathcal{T}}\big(\text{cost}(\mathcal{W}, \mathcal{D}, \mathcal{T})\big)$$

# **Robust** Tuning

**unpredictability in**

workload
hardware
design

optimal tuning

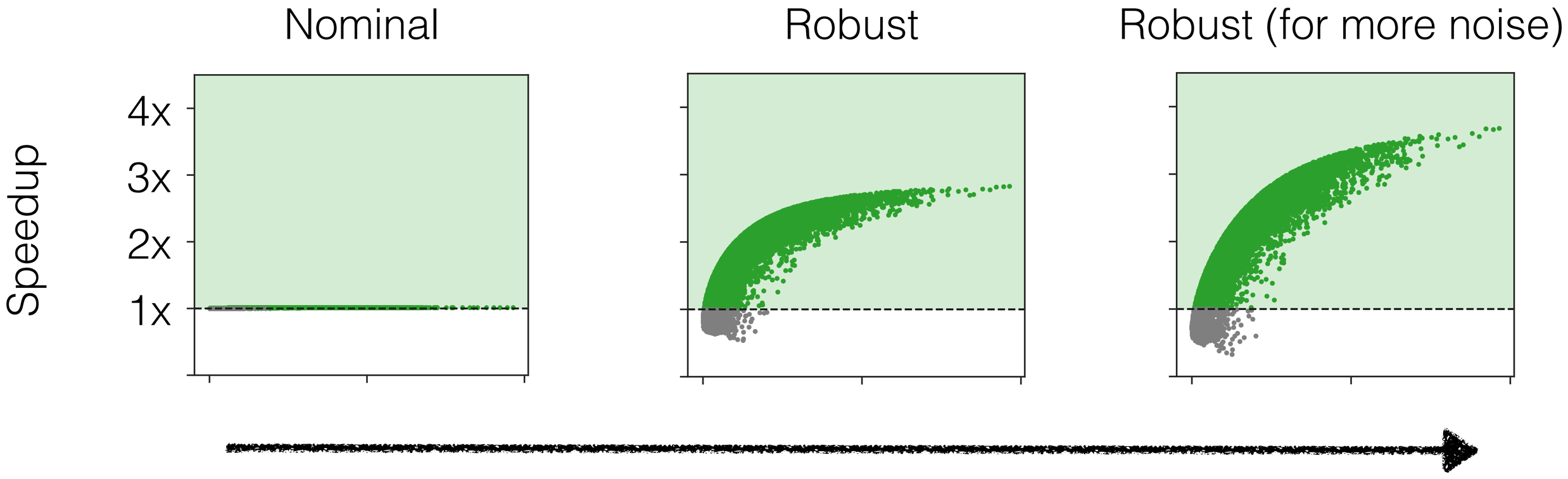
given workload $\mathcal{W}$, dataset $\mathcal{D}$, and a region $\mathcal{R}(\mathcal{W})$, find a tuning $\mathcal{T}^{rob}$:

$$\mathcal{T}^{\text{rob}} = \text{argmin}_{\mathcal{T}} \text{max}_{\mathcal{W}' \in \mathcal{R}(\mathcal{W})} \big(\text{cost}(\mathcal{W}', \mathcal{D}, \mathcal{T})\big)$$

BOSTON UNIVERSITY

HuynhVLDB22

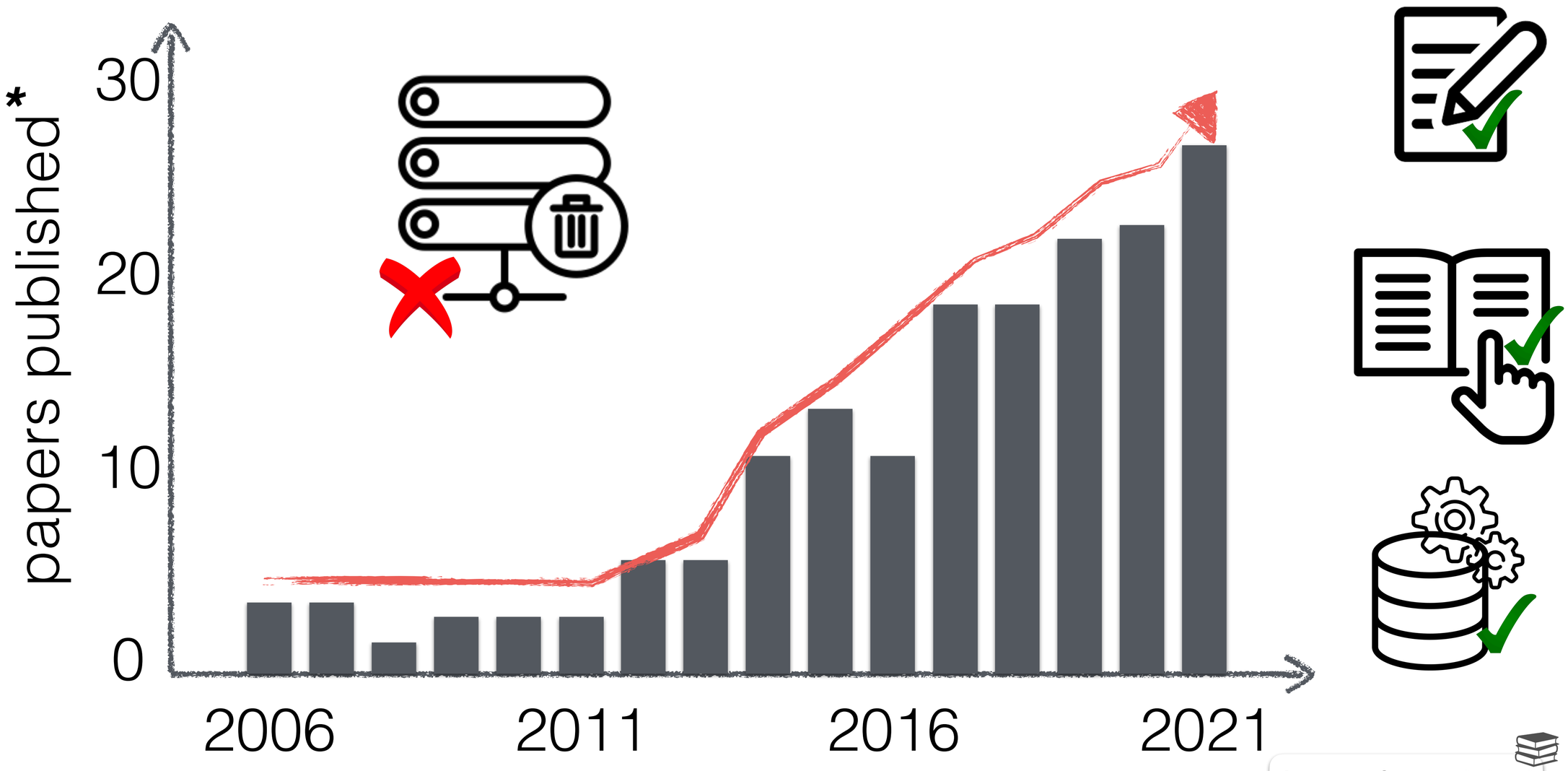
# **Robust** Tuning




BOSTON UNIVERSITY

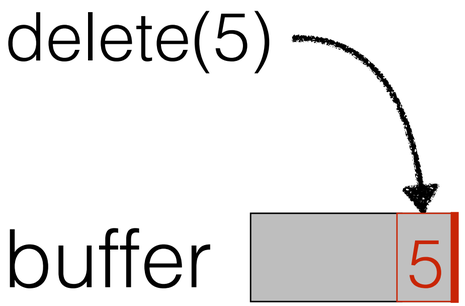
HuynhVLDB22

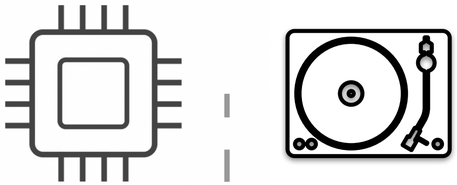
# Research Trend: **What about Deletes?**

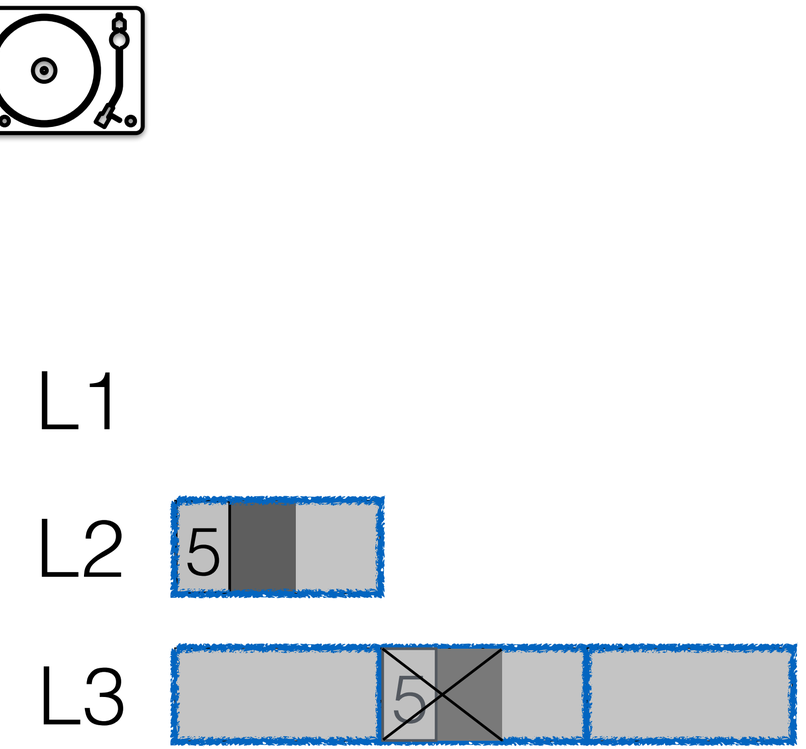



* data from DBLP

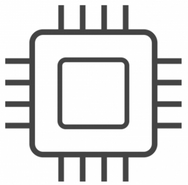
# deletes in LSM-tree

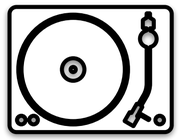




# deletes in LSM-tree





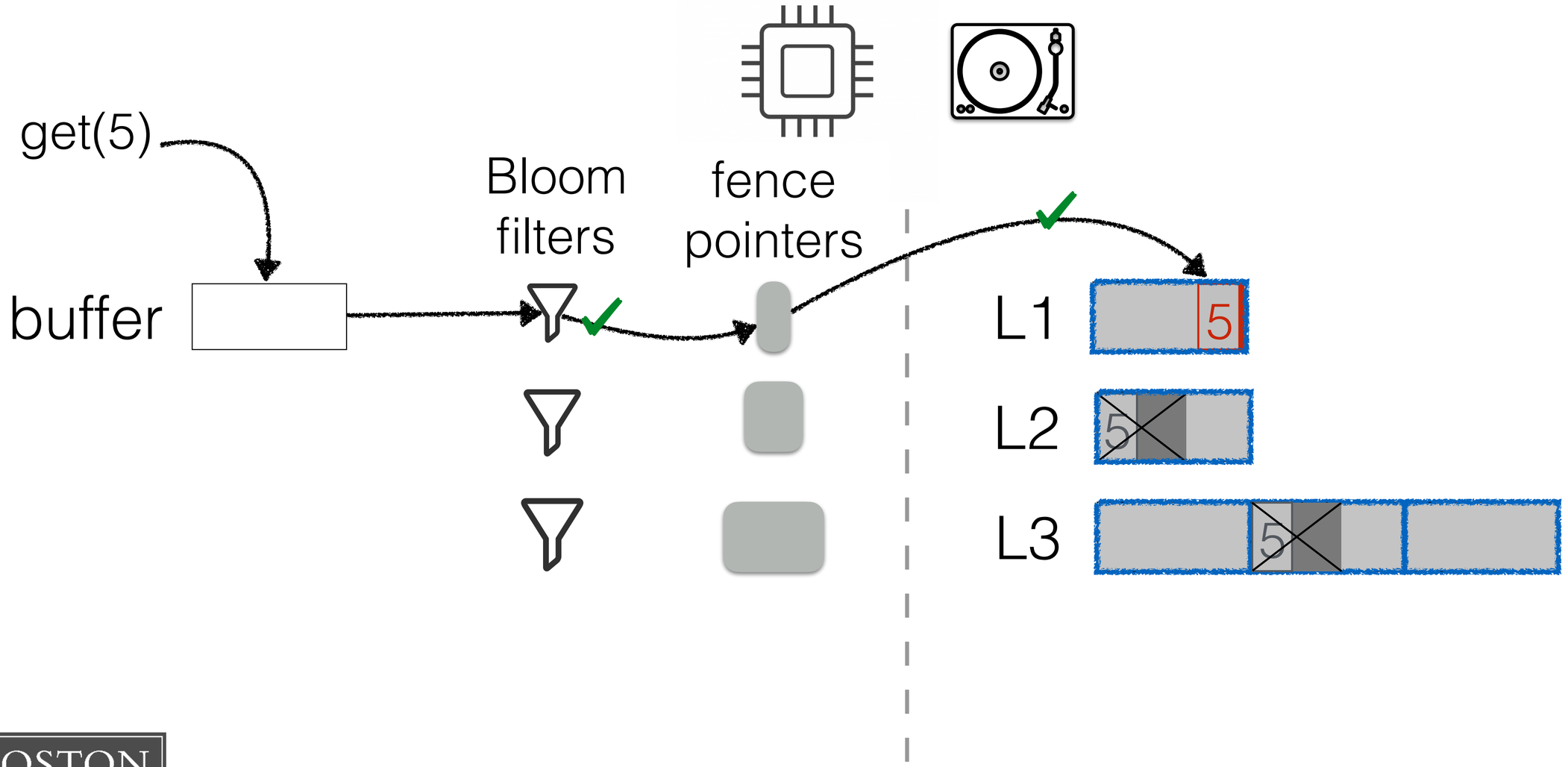
buffer

L1

L2

L3

# deletes in LSM-tree



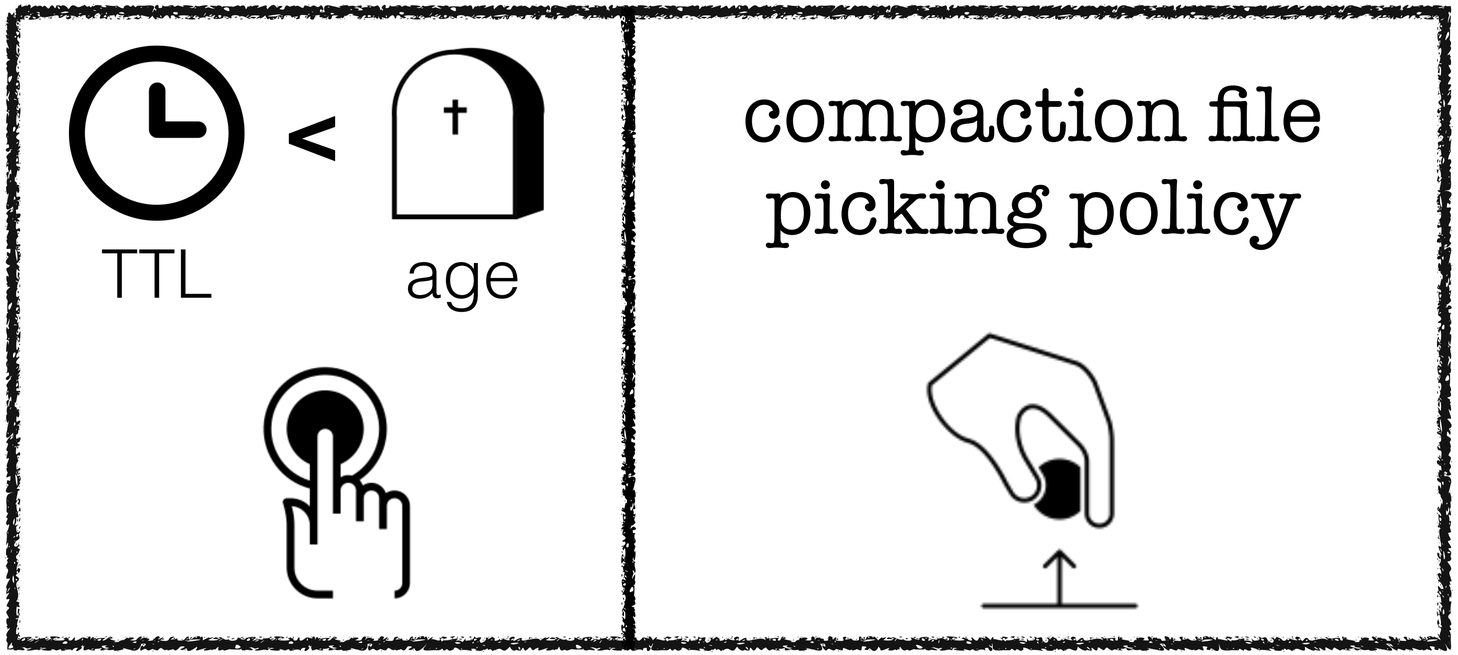
| Large-scale production | Internal DB ops | Privacy |
| --- | --- | --- |
| ZippyDB<br>**25.2M/day** | table drop | GDPR (EU, UK) |
| UP2X<br>**92.5% merge through deletes** | data migration | CCPA (California) |
| | periodic cleanup | VCDPA (Virginia) |

BOSTON UNIVERSITY

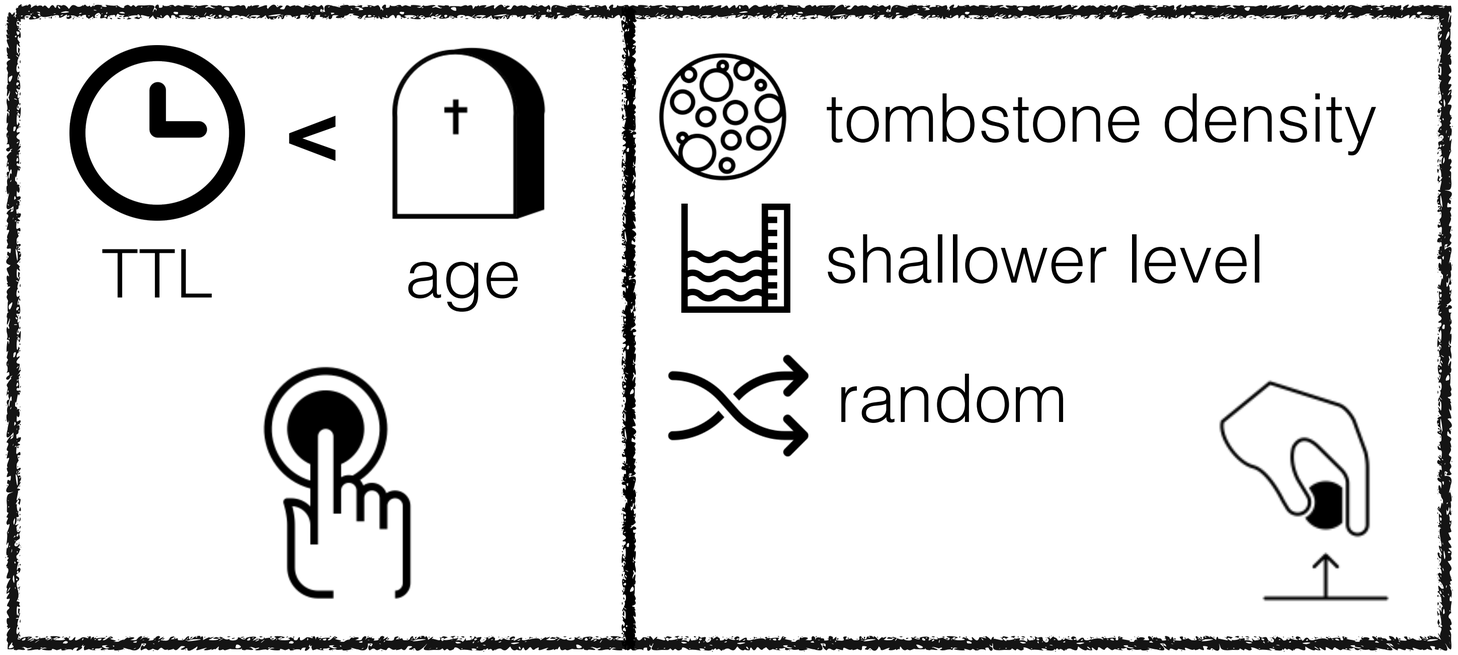
# Facilitating **Timely Deletes**

| compaction trigger | compaction file picking policy |
| --- | --- |
|  |  |

# Facilitating **Timely Deletes**

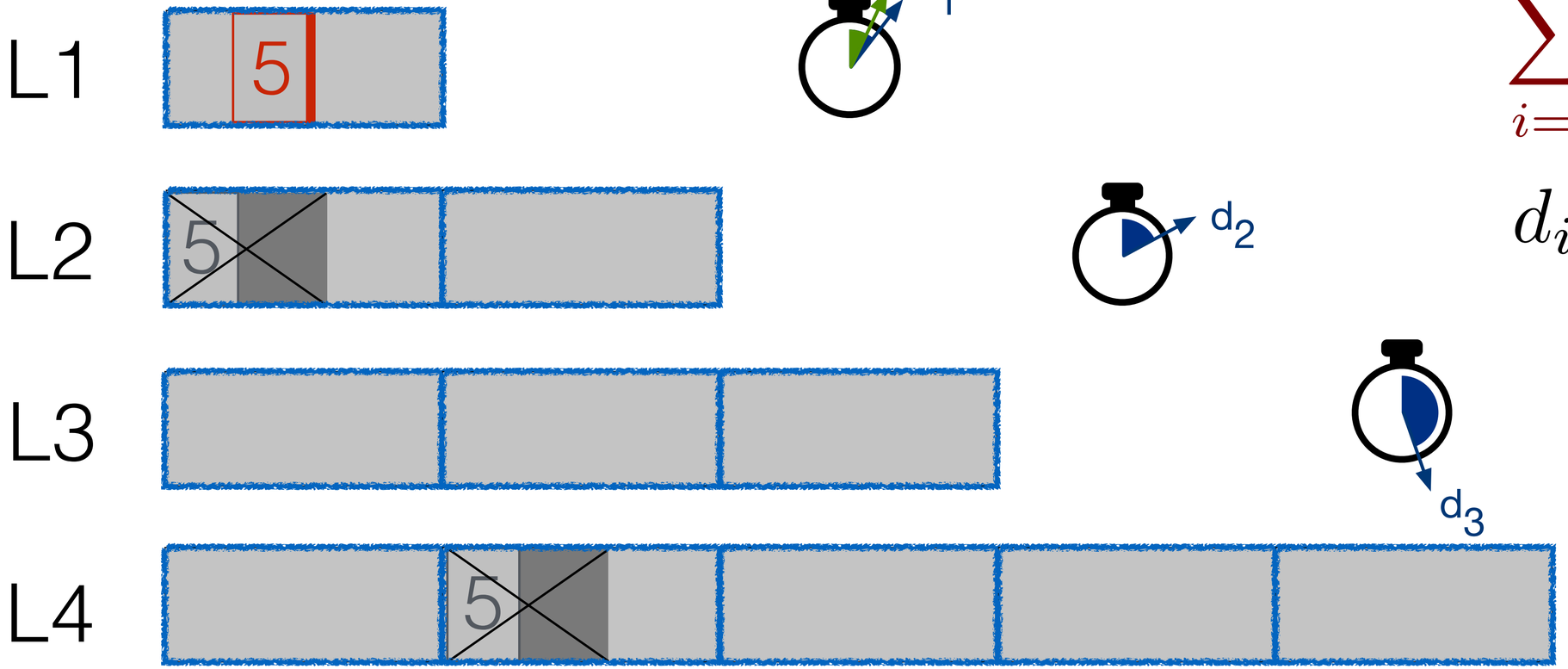


# Facilitating **Timely Deletes**



# Facilitating **Timely Deletes**

delete(5) within a threshold time: **$D_{th}$**




$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

$$d_i = T \cdot d_{i-1}$$

## Policy layer

*Right to be forgotten*

*Right to delete*

*Deletion right*

## Requirements layer

Retention-based Deletes

On-demand Deletes

AthanassoulisDEBull22

## Application layer

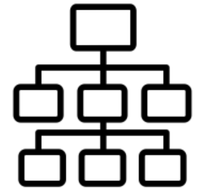
```
CREATE TABLE R (...)
   WITH RET_DUR
   {ARBITRARY | FIXED(...)}
   WITH DPT
   {ARBITRARY | FIXED(...)};
```

```
INSERT INTO R (...)
   WITH RET_DUR {<t>|t<i>};
```

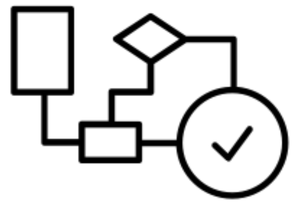
```
DELETE FROM R
   WHERE (...)
      WITH DPT {<d>|d<i>};
```

SarkarEDBT22

## System layer



Data layout re-organization



Data deletion algorithms

SarkarSIGMOD20

BOSTON UNIVERSITY

## Policy layer

*Right to be forgotten*

*Right to delete*

*Deletion right*

## Requirements layer

Retention-based Deletes

On-demand Deletes

AthanassoulisDEBull22

## Application layer

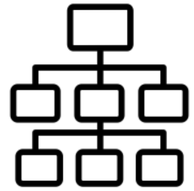
```
CREATE TABLE R (...)
  WITH RET_DUR
  {ARBITRARY | FIXED(...)}
  WITH DPT
  {ARBITRARY | FIXED(...)};
```

```
INSERT INTO R (...)
  WITH RET_DUR {<t>|t<i>};
```

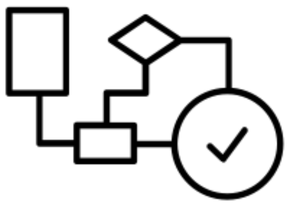
```
DELETE FROM R
  WHERE (...)
    WITH DPT {<d>|d<i>};
```

SarkarEDBT22

## System layer



Data layout re-organization



Data deletion algorithms

SarkarSIGMOD20

BOSTON UNIVERSITY

## Policy layer

*Right to be forgotten*

*Right to delete*

*Deletion right*

## Requirements layer

Retention-based Deletes

On-demand Deletes

AthanassoulisDEBull22

## Application layer

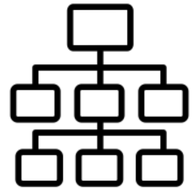
```
CREATE TABLE R (...)
   WITH RET_DUR
   {ARBITRARY | FIXED(...)}
   WITH DPT
   {ARBITRARY | FIXED(...)};
```

```
INSERT INTO R (...)
   WITH RET_DUR {<t>|t<i>};
```

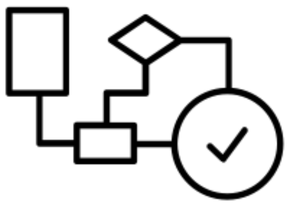
```
DELETE FROM R
   WHERE (...)
      WITH DPT {<d>|d<i>};
```

SarkarEDBT22

## System layer



Data layout re-organization



Data deletion algorithms

SarkarSIGMOD20

BOSTON UNIVERSITY

# The **Key Takeaways**

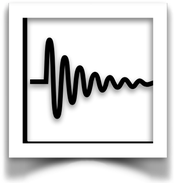
The LSM design space is **vast and complex**.

**Compactions are key** to building ingestion-friendly systems.

A **tuned LSM** engine can offer superior performance.

# **Open** Research **Challenges**

Reduce write amplification



Workload-aware compactions & layout transformation

Performance Stability & Holistic Tuning

Automatic Tuning & Adaptive Behavior

Privacy-aware LSM designs

BOSTON UNIVERSITY

# Please see our manuscript for all references!

## REFERENCES


[1] Regulation (EU) 2016/679 of the European Parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. C
– L119/88, 2
[2] California C
[3] The Califor
[4] Virginia Co
*Publication*
[5] H. Abu-Libd
Li, A. Ly, an
In *Proceedi*
[6] W. Y. Alkov
paction Fra
13(9):1388–
[7] S. Alsubaiee
I. Cetindil,
Kim, C. Li,
and T. Wes
*the VLDB E*
[8] Amazon. C
[9] Apache. Ac
[10] Apache. HI
[11] Apache. Ca
[12] M. Athanas
Efficient On
*Internationa*
[13] M. Athanas
*Proceedings*
*Data, Tutor*
[14] M. Athanas
M. Callagha
*of the Interr*
461–466, 20
[15] M. Athanas
Deletion-C
[16] O. Balmau,
dona, SILK

[56] S. Idreos and M. Callaghan. Key-Value Storage Engines. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 2667–2672, 2020.
[57] S. Idreos, N. Dayan, W. Qin, M. Akmanalp, S. Hilgard, A. Ross, J. Lennon, V. Jain, H. Gupta, D. Li, and Z. Zhu. Design Continuums and the Path Toward Self-Designing K
*Conference*
[58] S. Idreos, K.
Kester, D. G
Data Structu
[59] S. Idreos, K.
Calculator:
and Learne
*Conference*
[60] Influxdata.
*https://docs.*
[61] S. Kannan,
Dusseau. Re
nawi and B.
*{ATC} 2018,*
ciation, 201
[62] K. Keahey.
*https://www*
[63] M. L. Kerste
*on Innovati*
[64] J. Kim, S. Le
value store f
*Conference*
*SC 2017, De*
[65] Y.-S. Kim, T.
Merge-Tree
*Internationa*
[66] A. Kipf, R. M

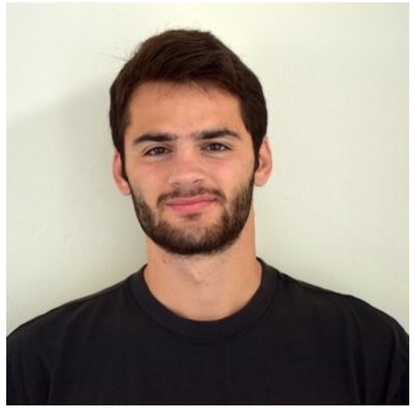
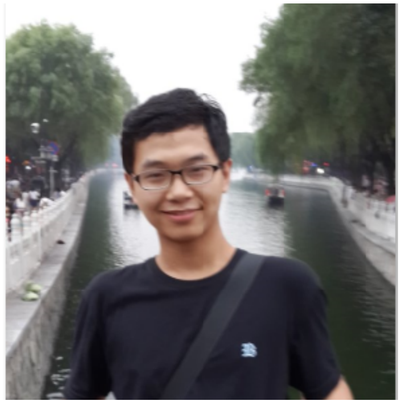
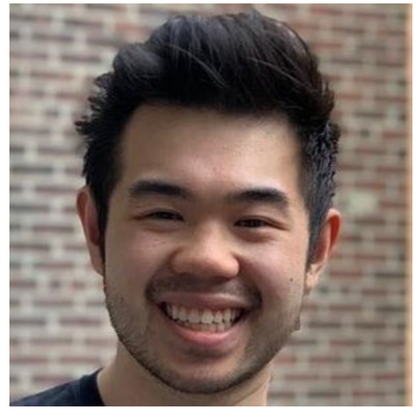
[113] S. Sarkar, D. Staratzis, Z. Zhu, and M. Athanassoulis. Constructing and Analyzing the LSM Compaction Design Space. *Proceedings of the VLDB Endowment*, 14(11):2216–2229, 2021.
[114] ScyllaDB. Online reference. *https://www.scylladb.com/*.
[115] R. Sears and R. Ramakrishnan. bLSM: A General Purpose Log Structured Merge Tree. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 217–228, 2012.
[116] M. I. Seltzer. Berkeley DB: A Retrospective. *IEEE Data Engineering Bulletin*, 30(3):21–28, 2007.
[117] W. Tan, S. Tata, Y. R. Tang, and L. L. Fong. Diff-Index: Differentiated Index in Distributed Log-Structured Data Stores. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 700–711, 2014.
[118] Y. R. Tang, A. Iyengar, W. Tan, L. L. Fong, L. Liu, and B. Palanisamy. Deferred Lightweight Indexing for Log-Structured Key-Value Stores. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, pages 11–20, 2015.
[119] D. Teng, L. Guo, R. Lee, F. Chen, S. Ma, Y. Zhang, and X. Zhang. LSbM-tree: Re-Enabling Buffer Caching in Data Management for Mixed Reads and Writes. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, pages 68–79, 2017.
[120] R. Thonangi, S. Babu, and J. Yang. A Practical Concurrent Index for Solid-State Drives. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1332–1341, 2012.
[121] R. Thonangi and J. Yang. On Log-Structured Merge for Solid-State Drives. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 683–694, 2017.
[122] T. Vinçon, S. Hardock, C. Riegger, J. Oppermann, A. Koch, and I. Petrov. NoFTL-

2017.
[28] Y. Chen, Y. Lu, F. Yang, Q. Wang, Y. Wang, and J. Shu. FlatStore: An Efficient Log-Structured Key-Value Storage Engine for Persistent Memory. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems,*

14(3):241–254, 2020.
[83] C. Luo and M. J. Carey. LSM-based Storage Techniques: A Survey. *The VLDB Journal*, 29(1):393–418, 2020.
[84] S. Luo, S. Chatterjee, R. Ketsetsidis, N. Dayan, W. Qin, and S. Idreos. Rosetta: A Robust Space-Time Optimized Range Filter for Key-Value Stores. In *Proceedings*

[127] X. Wu, Y. Xu, Z. Shao, and S. Jiang. LSM-trie: An LSM-tree-based Ultra-Large Key-Value Store for Small Data Items. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 71–82, 2015.
[128] L. Yang, H. Wu, T. Zhang, X. Cheng, F. Li, L. Zou, Y. Wang, R. Chen, J. Wang, and G. Huang. Leaper: A Learned Prefetcher for Cache Invalidation in LSM-tree based Storage Engines. *Proceedings of the VLDB Endowment*, 13(11):1976–1989, 2020.
[129] T. Yao, J. Wan, P. Huang, X. He, Q. Gui, F. Wu, and C. Xie. A Light-weight Compaction Tree to Reduce I/O Amplification toward Efficient Key-Value Stores. In *Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2017.
[130] T. Yao, J. Wan, P. Huang, X. He, F. Wu, and C. Xie. Building Efficient Key-Value Stores via a Lightweight Compaction Tree. *ACM Transactions on Storage (TOS)*, 13(4):29:1–29:28, 2017.
[131] H. Zhang, H. Lim, V. Leis, D. G. Andersen, M. Kaminsky, K. Keeton, and A. Pavlo. SuRF: Practical Range Query Filtering with Fast Succinct Tries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 323–336, 2018.
[132] H. Zhang, H. Lim, V. Leis, D. G. Andersen, M. Kaminsky, K. Keeton, and A. Pavlo. Succinct Range Filters. *ACM Transactions on Database Systems (TODS)*, 45(2):5:1—-5:31, 2020.
[133] W. Zhang, Y. Xu, Y. Li, and D. Li. Improving Write Performance of LSMT-Based Key-Value Store. In *22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, December 13-16, 2016*, pages 553–560, 2016.
[134] W. Zhang, X. Zhao, S. Jiang, and H. Jiang. ChameleonDB: a key-value store for optane persistent memory. In *EuroSys '21: Sixteenth European Conference*

# The **DiSC** Lab

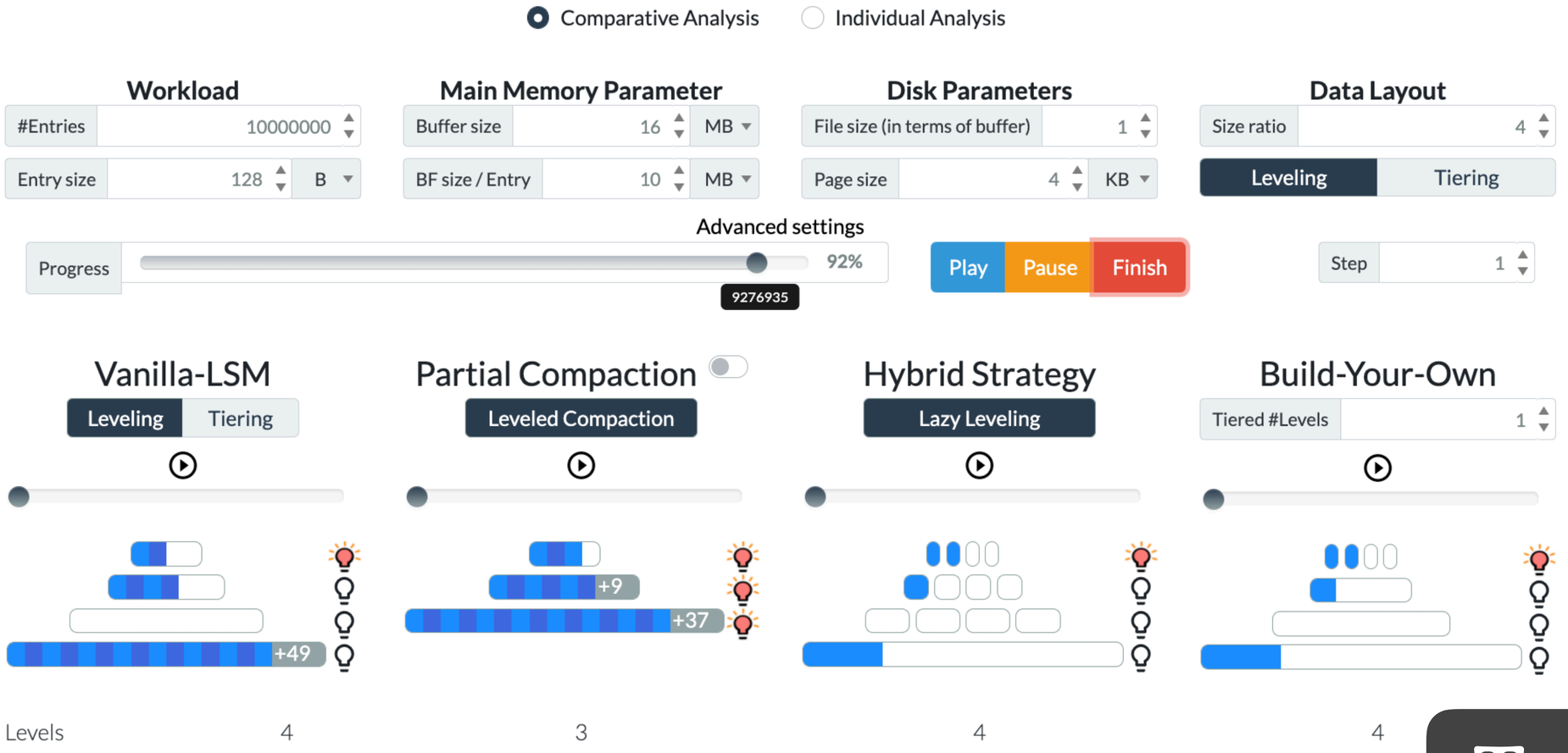







disc.bu.edu

BOSTON UNIVERSITY

# Don’t Miss: Compactionary




disc-projects.bu.edu/compactionary

Tuesday
4-6PM

BOSTON
UNIVERSITY

# *Dissecting, Designing, and Optimizing* **LSM-Based Data Stores**

*Subhadeep Sarkar* *Manos Athanassoulis*

Thank You!

Questions?

BOSTON UNIVERSITY

DiSC lab