BOSTON UNIVERSITY

CS 561: **Data Systems Architectures**

class 6

Log-structured Merge Trees

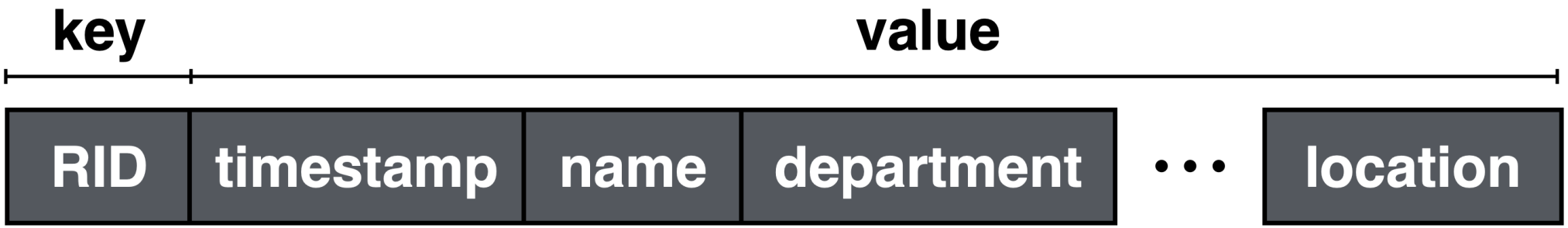
Prof. Manos Athanassoulis

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

Do we have a quiz today … ?

Yes!

Key-Value Stores

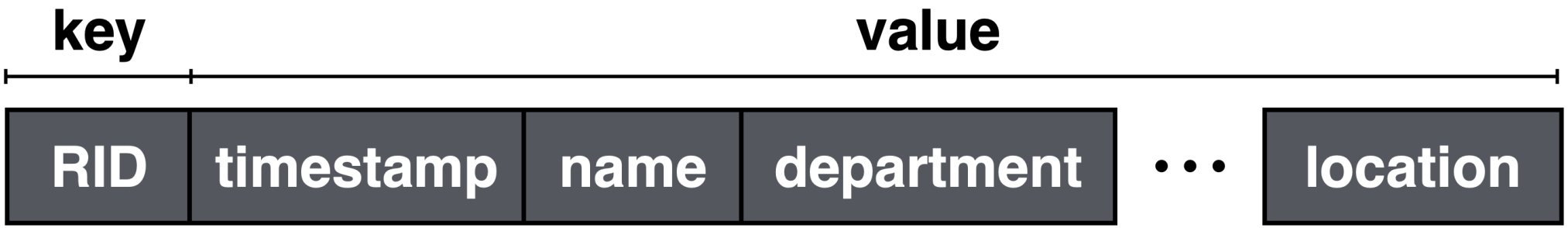


semi-structured data

relational data

document data

LSM-tree based Key-Value Stores



semi-structured data

document data

relational data

Log-**S**tructured **M**erge-tree

LSM-tree

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

BOSTON UNIVERSITY

LSM-tree

O'Neil *et al.*

1996

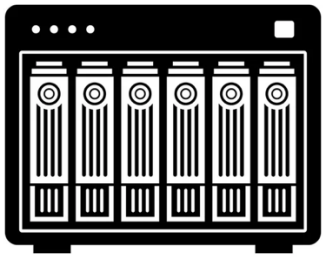
Patrick O'Neil
UMass Boston

LSM-tree
O'Neil *et al.*

1996

BOSTON UNIVERSITY

- good random writes
- good reads



array
of discs

how many IOPS?

10KRPM
max seek time 1.5ms
100 disks

10KRPM: 10K rev in 60s
60/10000=6ms per rev
avg. rot. delay: 3ms (6ms/2)
avg. seek time: 0.75ms (1.5ms/2)
1 I/O / 3.75ms: 267 IOPS
100 disks: 26,700 IOPS

why?

RAID, striping ←

✗ LSM not explicitly needed

LSM-tree
O'Neil *et al.*

so, **arrays of disks** were **enough**!

Bigtable

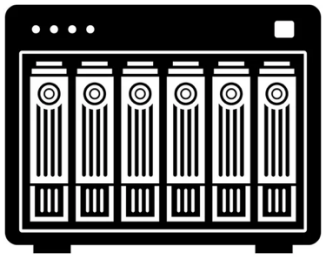
1980s — 1996 ⟷ 2006

a decade

BOSTON UNIVERSITY

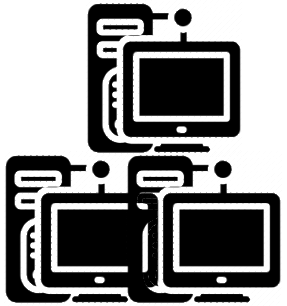
- good random writes
- good reads

- poor ingestion perf.
- poor query perf.



array
of discs

what happened in 2006?



commodity
hardware

LSM-tree
O'Neil *et al.*

We set up a Bigtable cluster with N tablet servers to measure the performance and scalability of Bigtable as N is varied. The tablet servers were configured to use 1 GB of memory and to write to a GFS cell consisting of 1786 machines with two 400 GB IDE hard drives each.

Bigtable

1980s

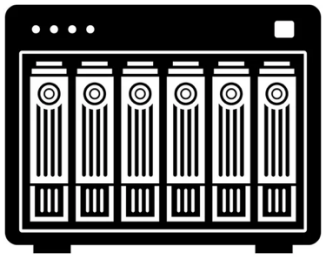
1996

2006

a decade

BOSTON UNIVERSITY

- good random writes
- good reads



array
of discs

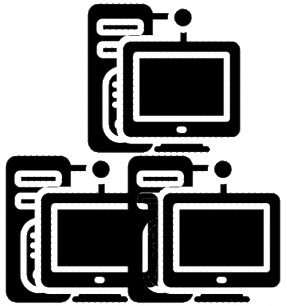
SSD wear-friendly

competitive rand. reads

fast ingestion (sequential)

why this
is important?

- poor ingestion perf.
- poor query perf.



commodity
hardware

LSM-tree
O'Neil *et al.*

SSDs have dominated
storage since!

Bigtable

1980s

1996

2006

a decade

BOSTON
UNIVERSITY

LSM-tree

O'Neil *et al.*

1996

Bigtable

2006

APACHE HBASE

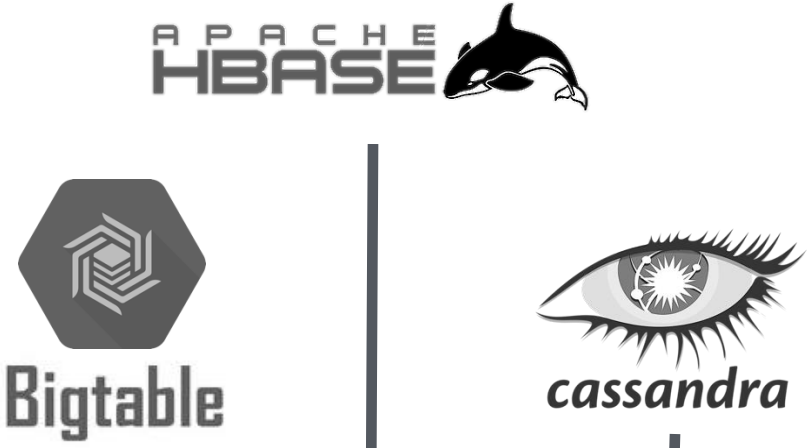
2007

BOSTON UNIVERSITY

LSM-tree

O'Neil *et al.*

1996



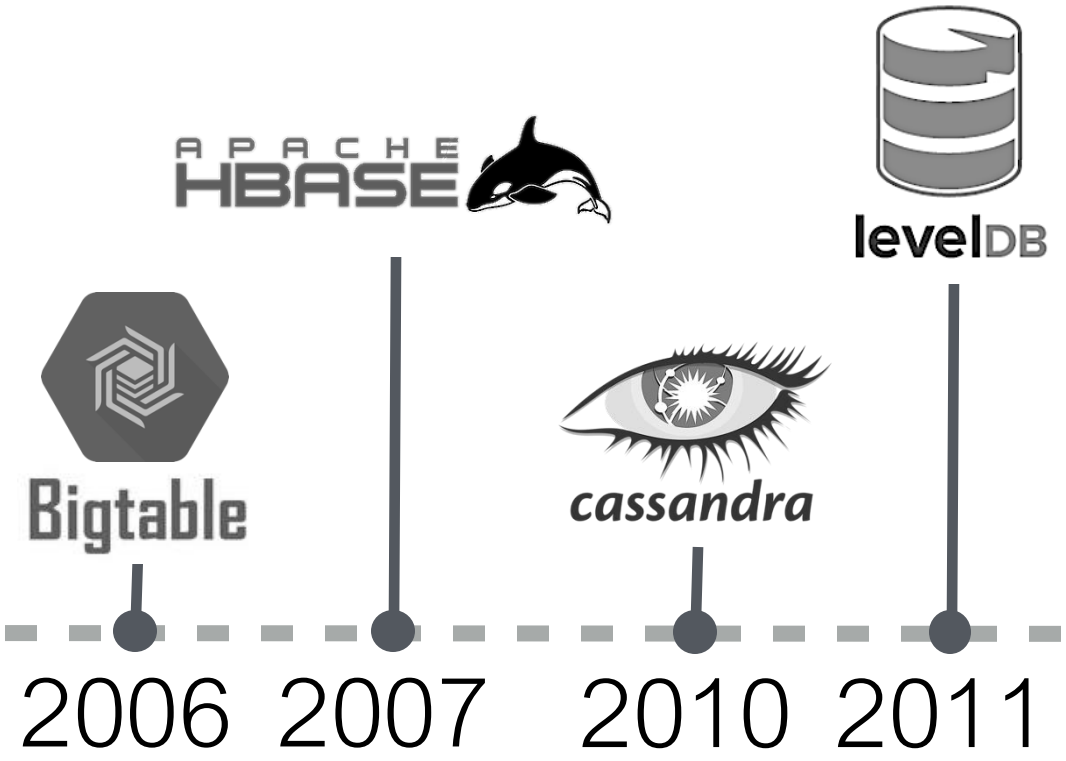
2006 2007 2010

BOSTON UNIVERSITY

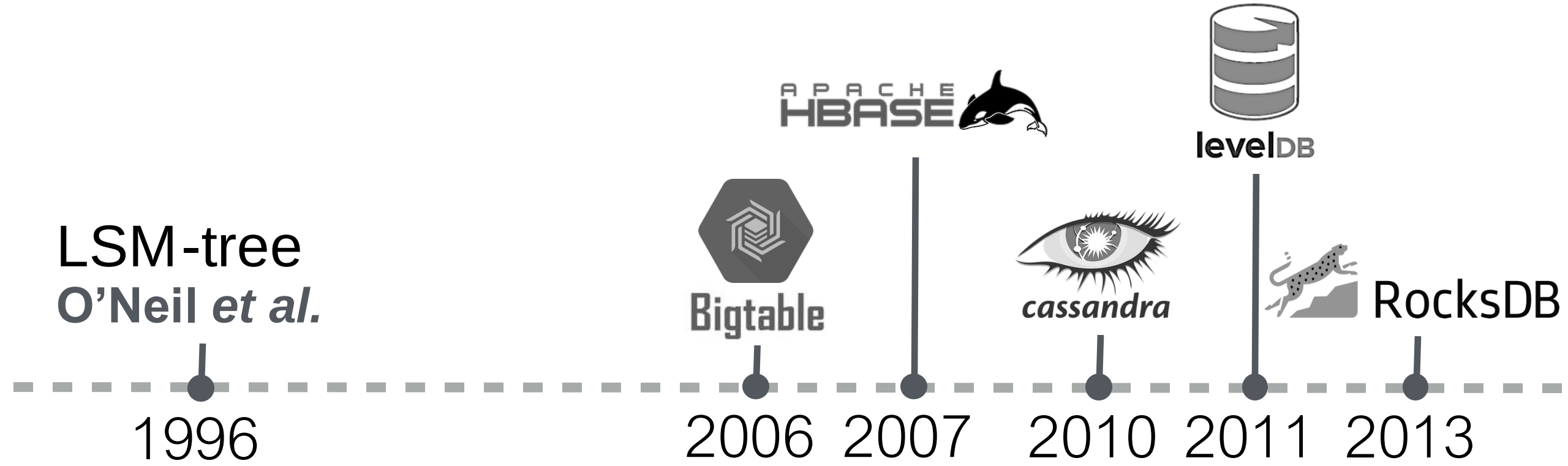
LSM-tree

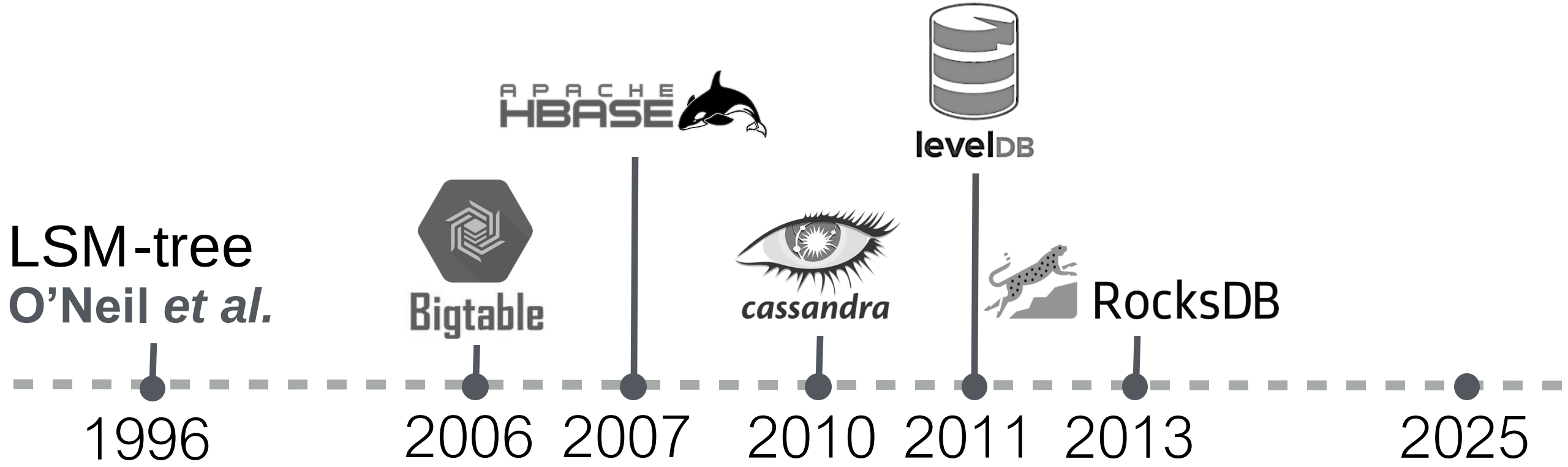
O'Neil *et al.*

1996

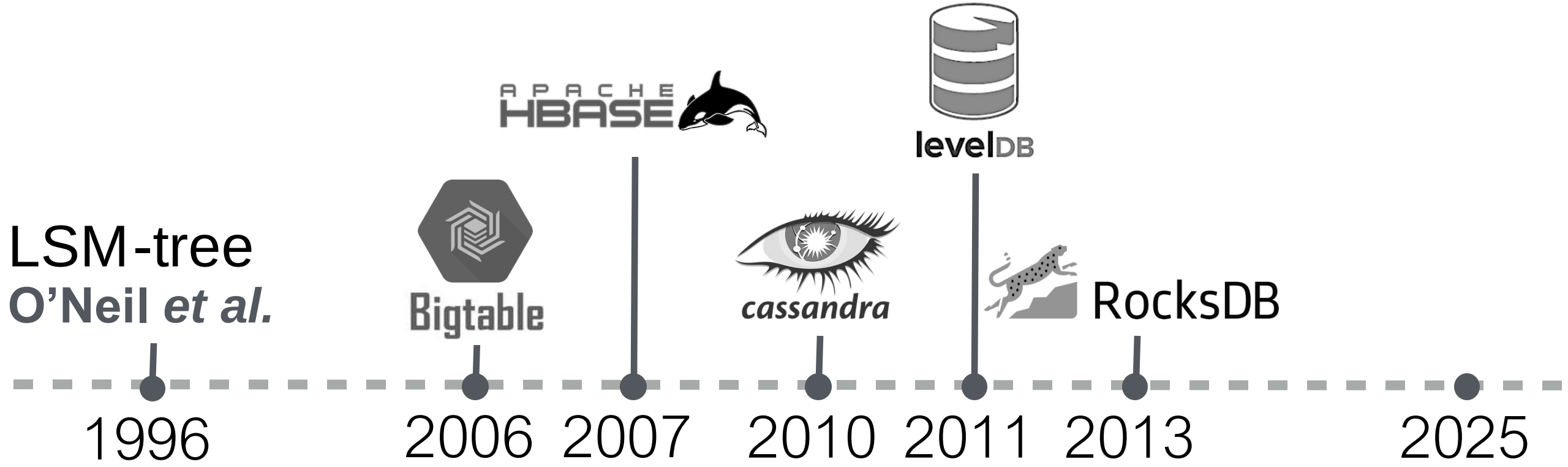


BOSTON UNIVERSITY


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



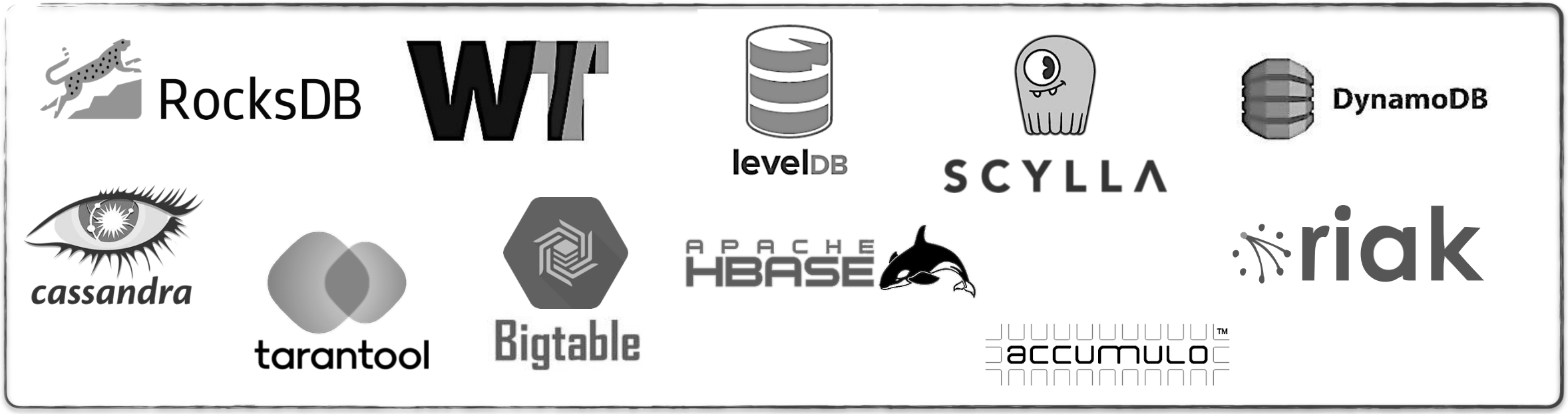
BOSTON UNIVERSITY



BOSTON UNIVERSITY

LSM-tree

NoSQL



relational

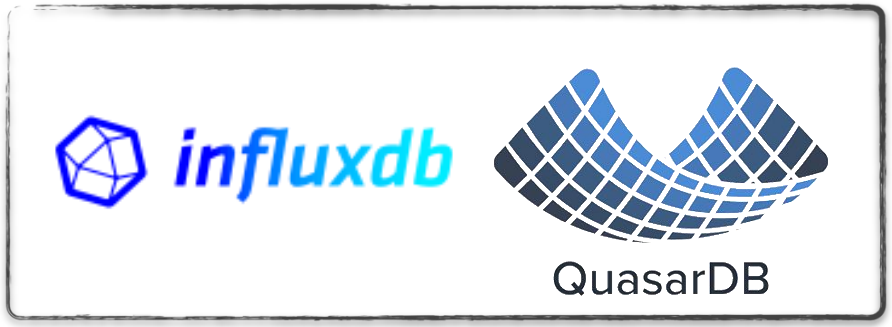
time-series

2025

LSM-tree

NoSQL

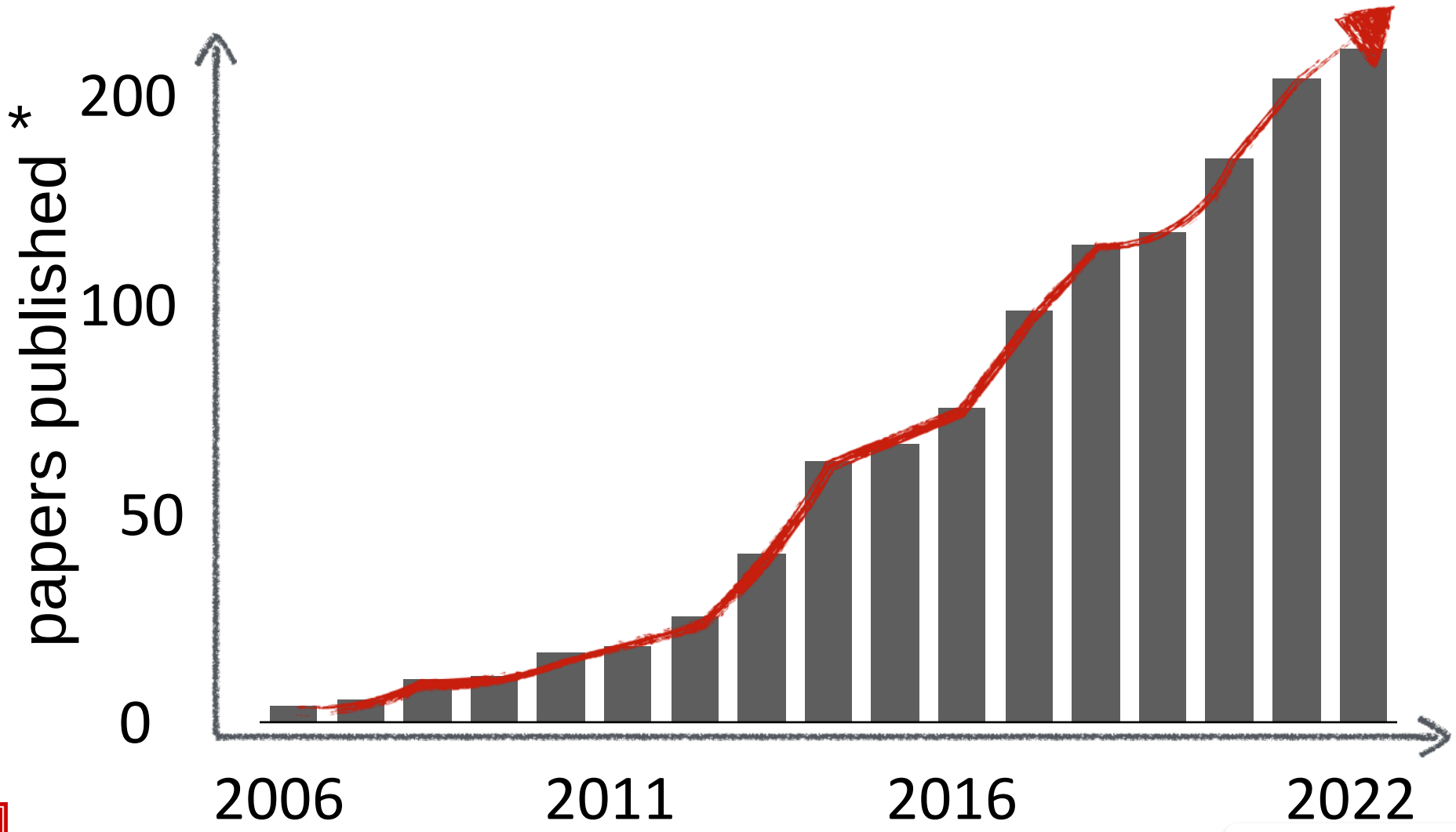
relational



time-series

2025

Research Trend

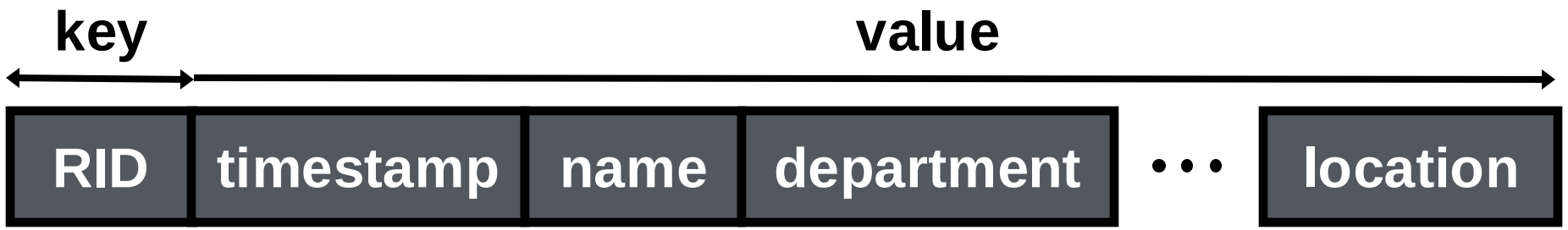


* data from Google scholar

BOSTON UNIVERSITY

LSM **Basics**

key-value
pairs

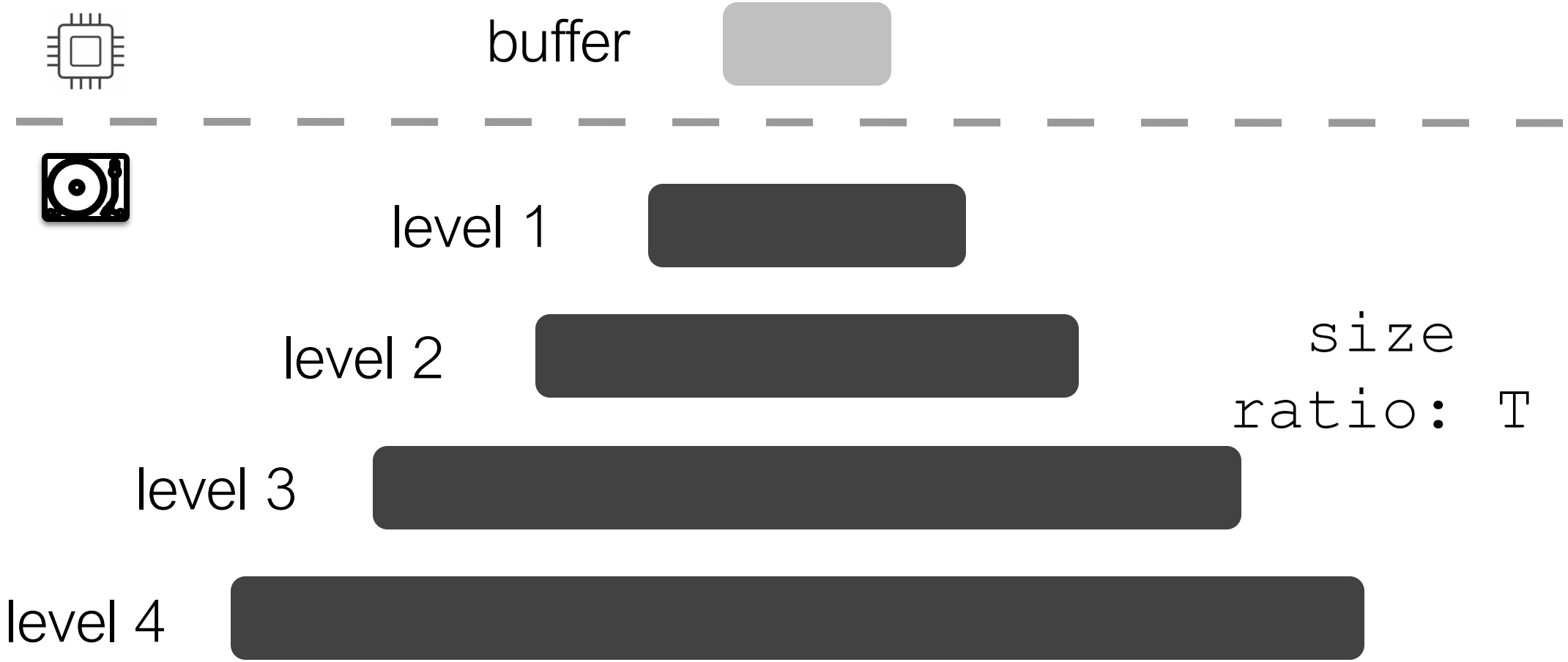


LSM **Basics**

`key-value pairs`

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

LSM **Basics**



LSM Operating **Principles**

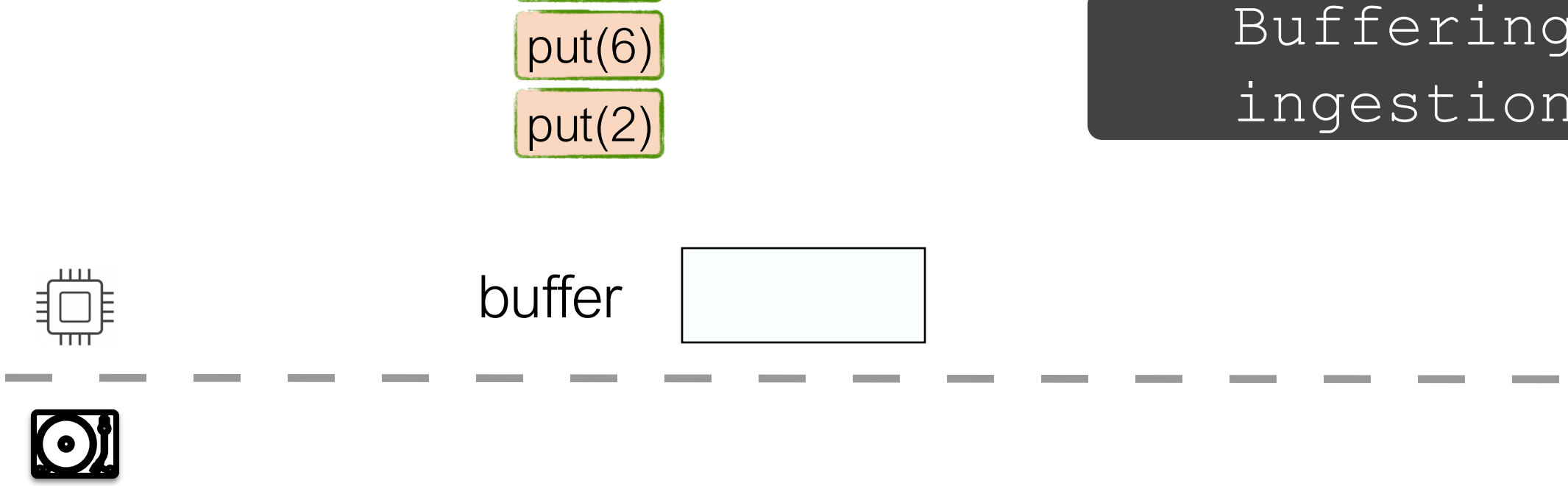
Buffering ingestion

Immutable files on storage

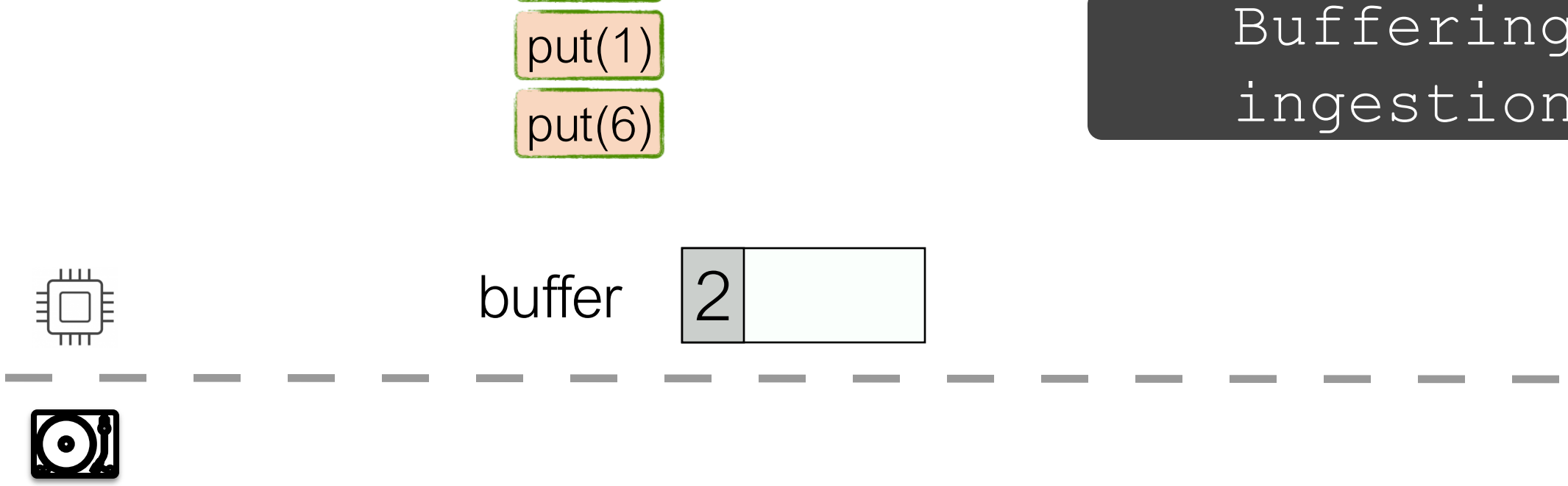
Out-of-place updates & deletes

Periodic data layout reorganization

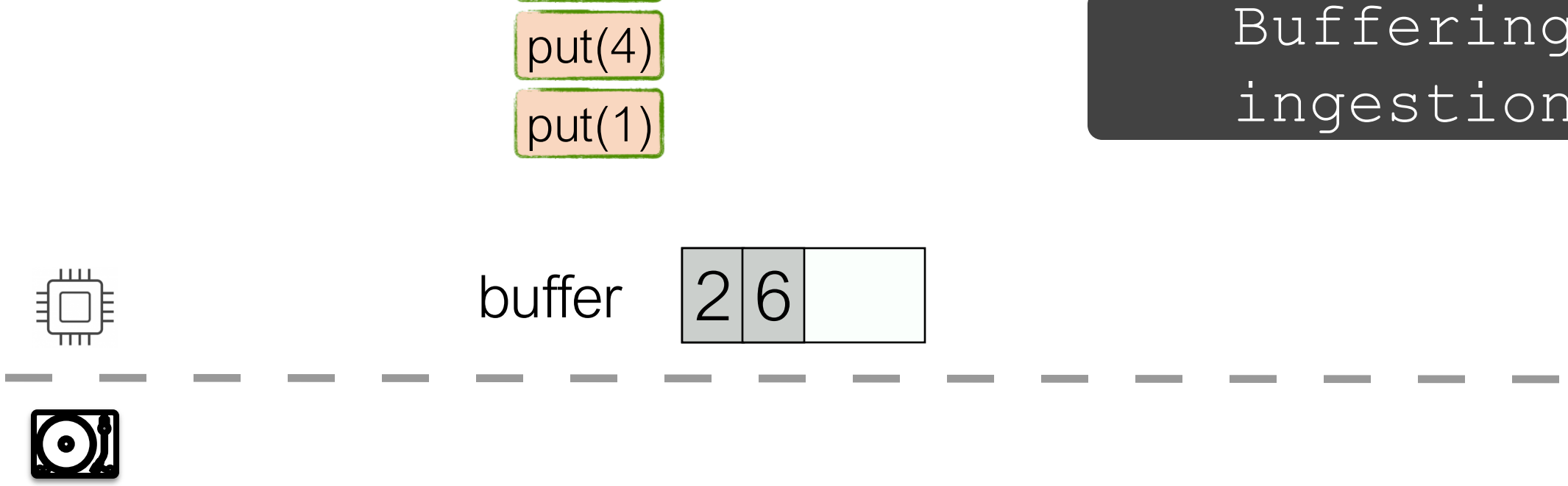
Buffering ingestion



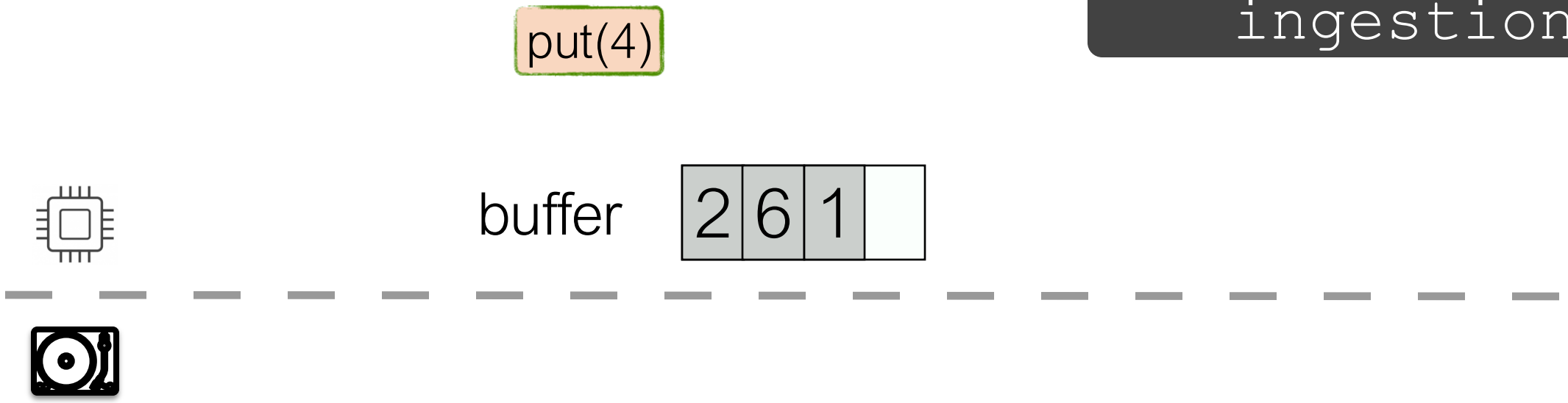
Buffering ingestion



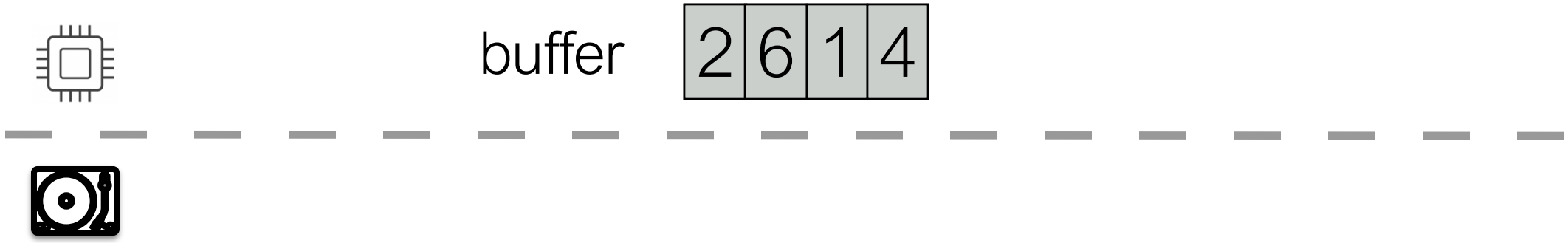
Buffering ingestion



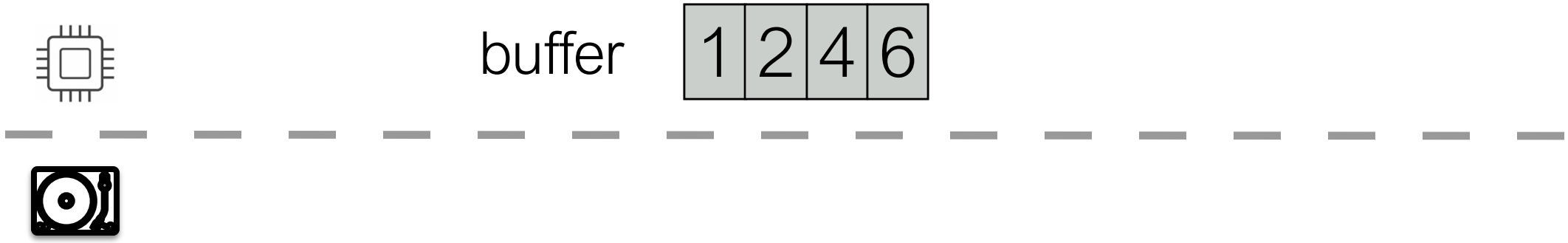
Buffering ingestion



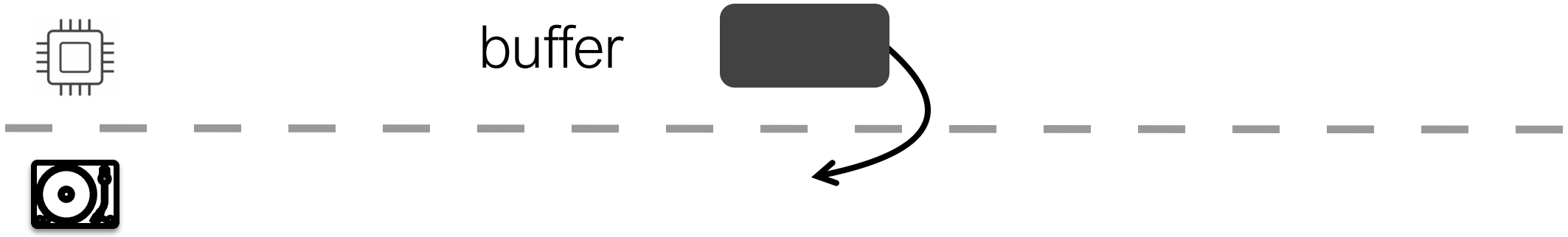
Buffering ingestion



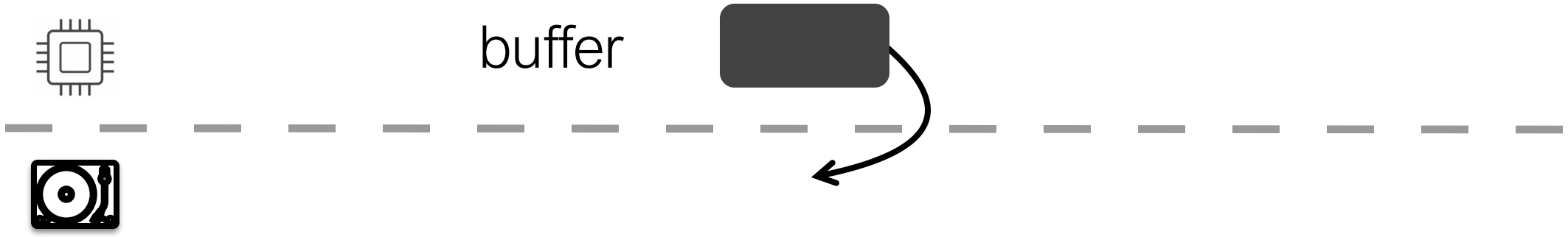
Buffering ingestion



Buffering ingestion

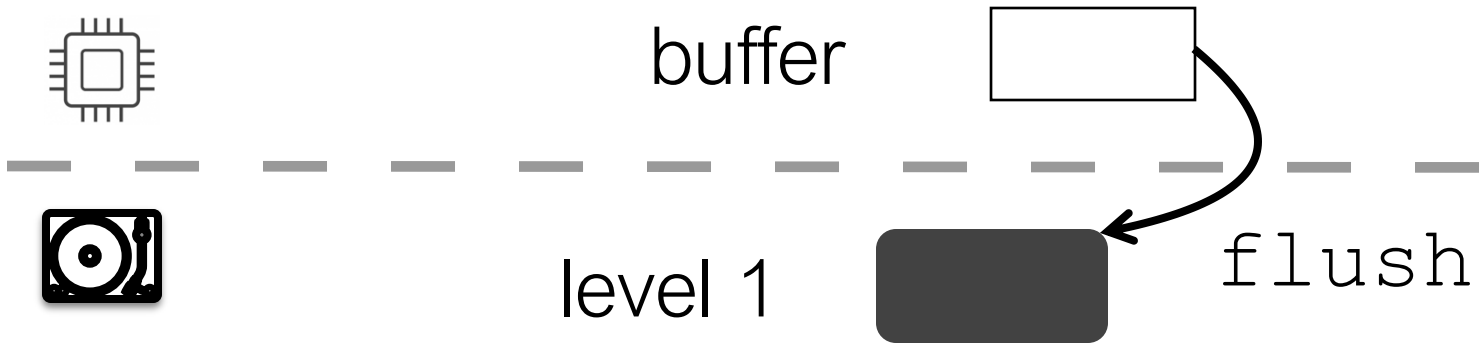


Buffering ingestion



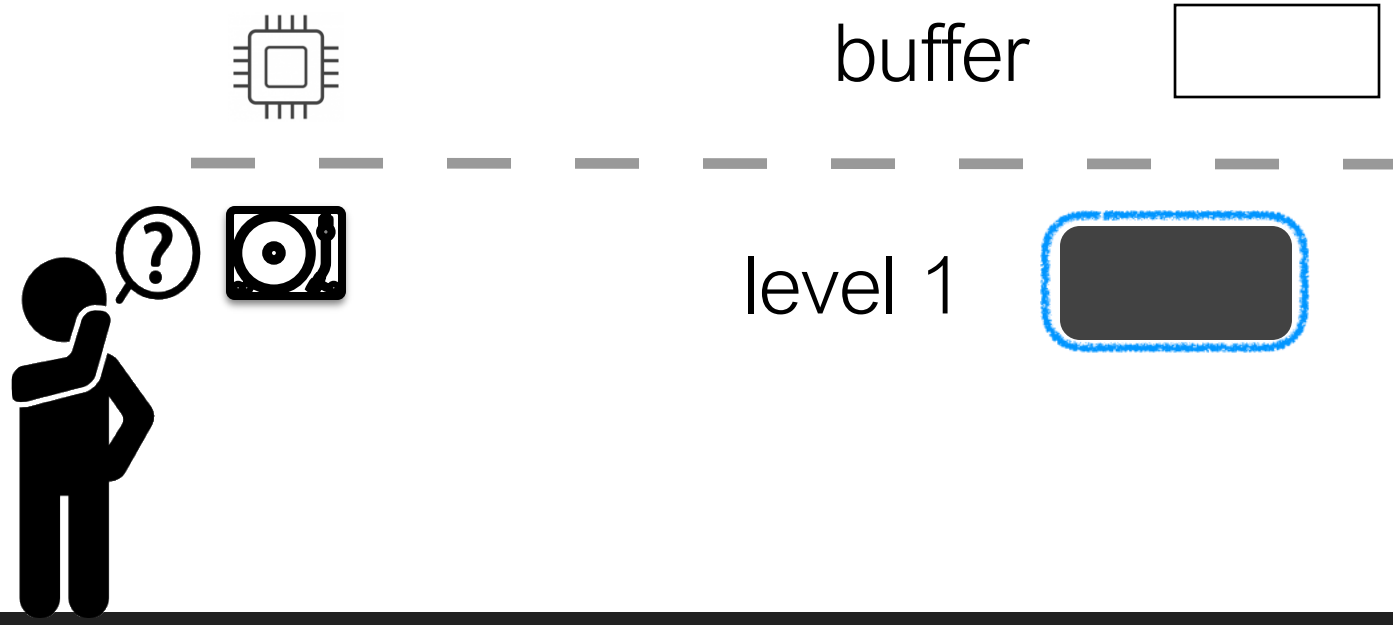
Buffering ingestion

- low ingestion cost
- fast temporal reads

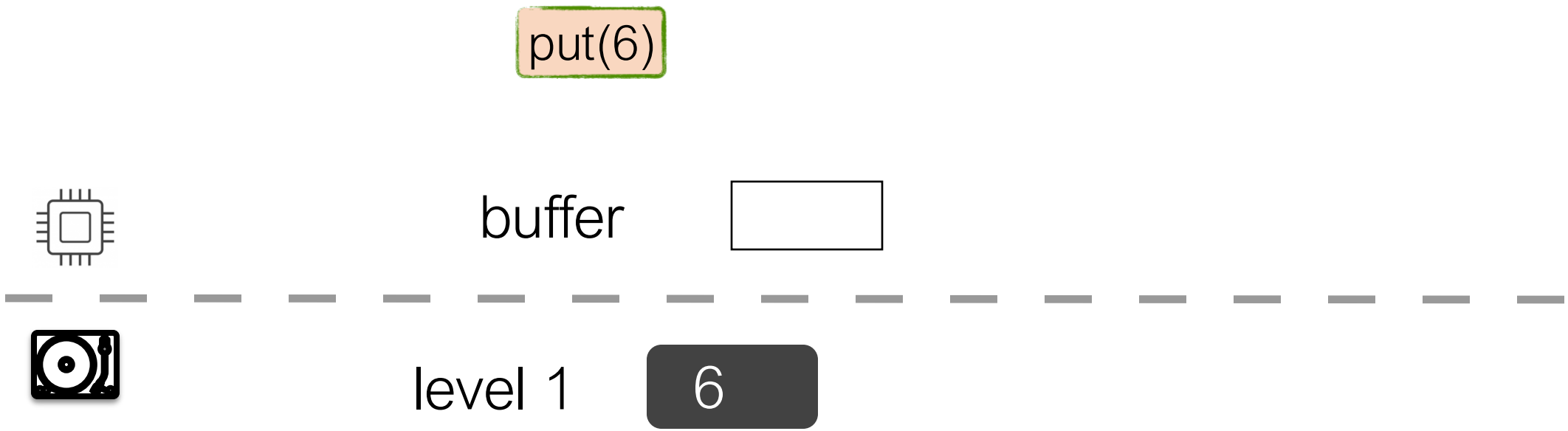


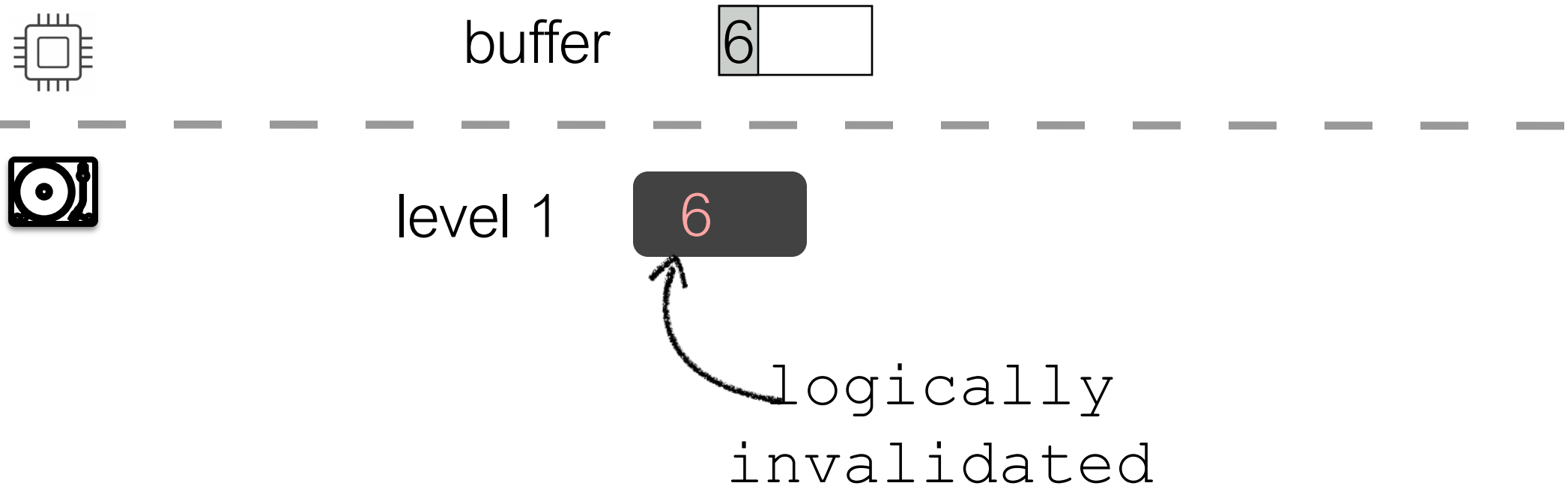
Immutable files on storage

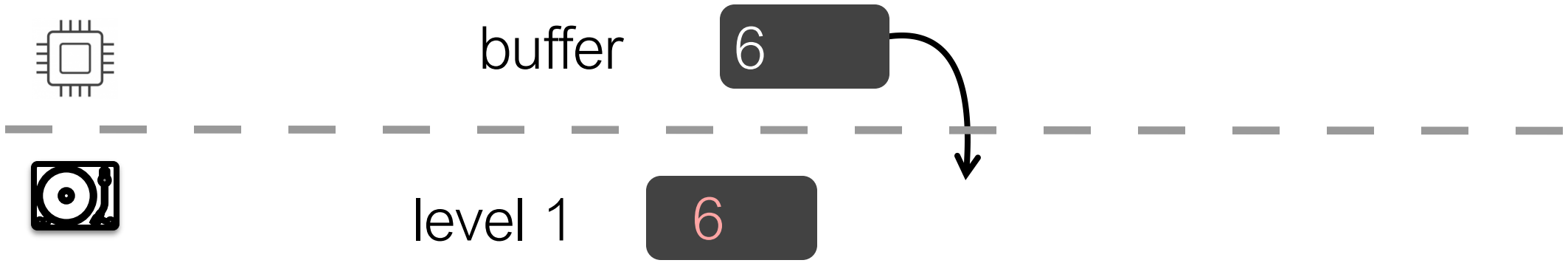
- compact storage
- good ingestion throughput

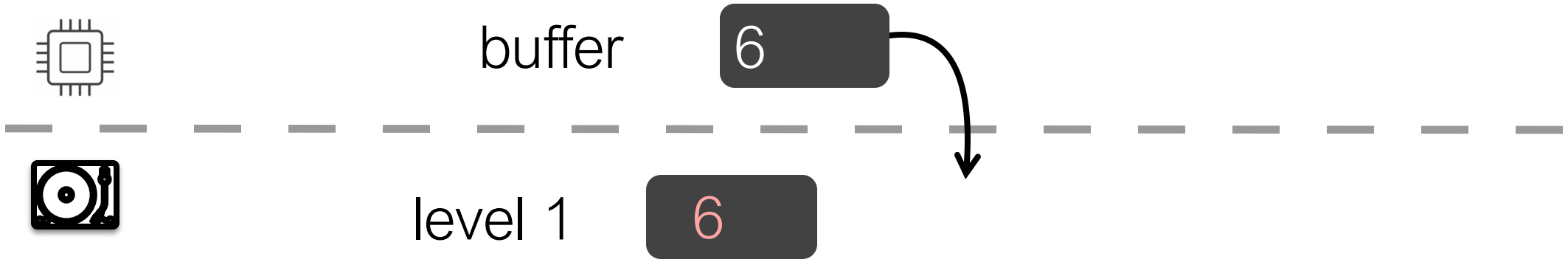


How do we update data?



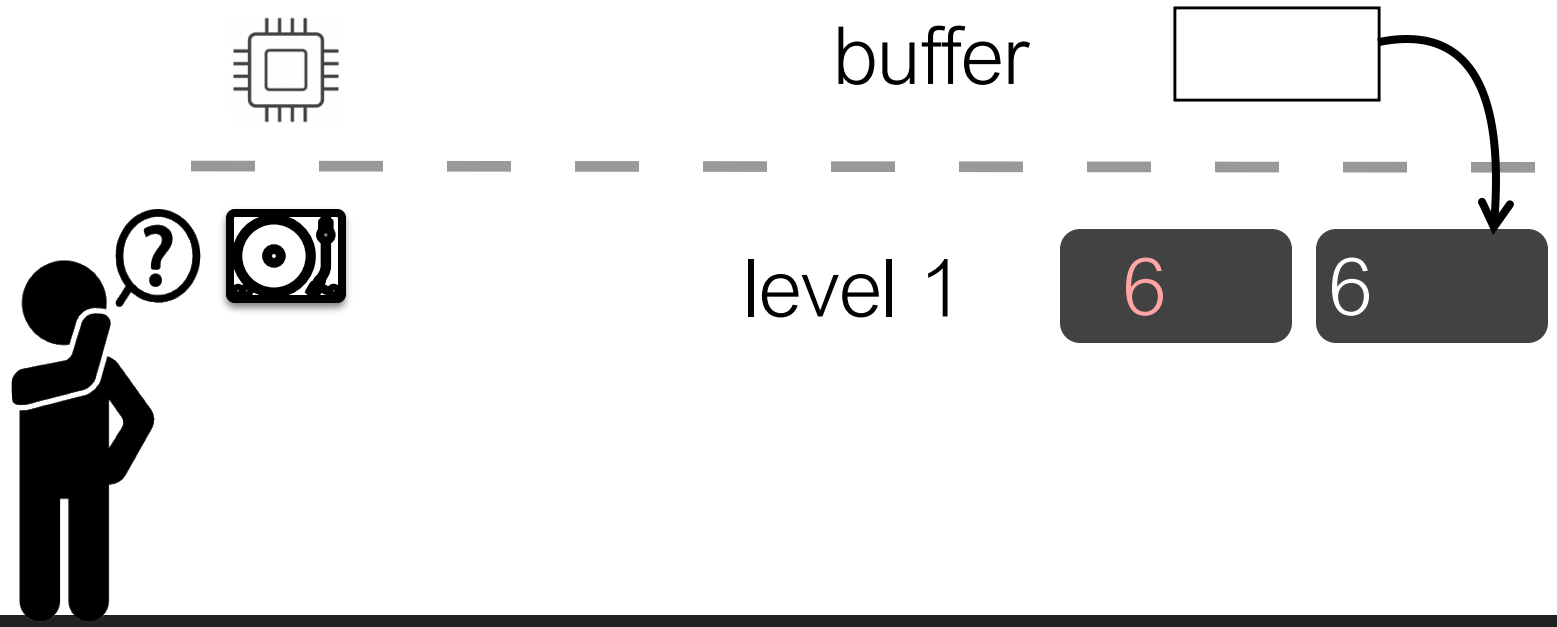



buffer
6
level 1
6

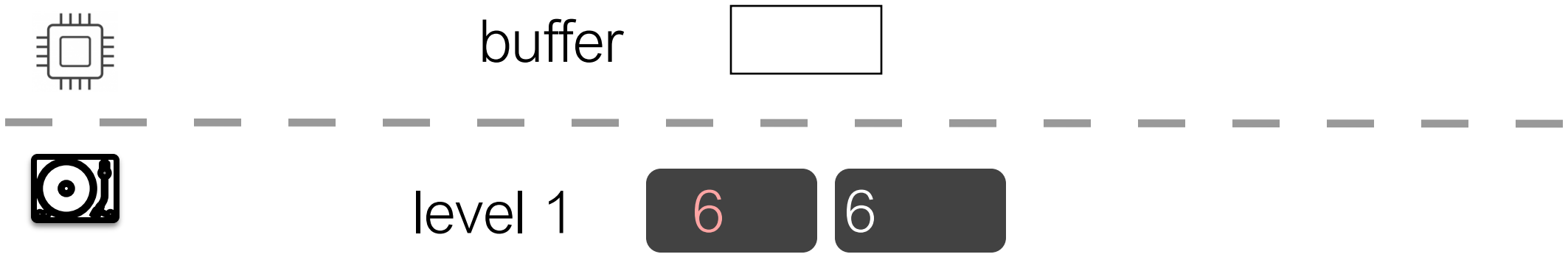


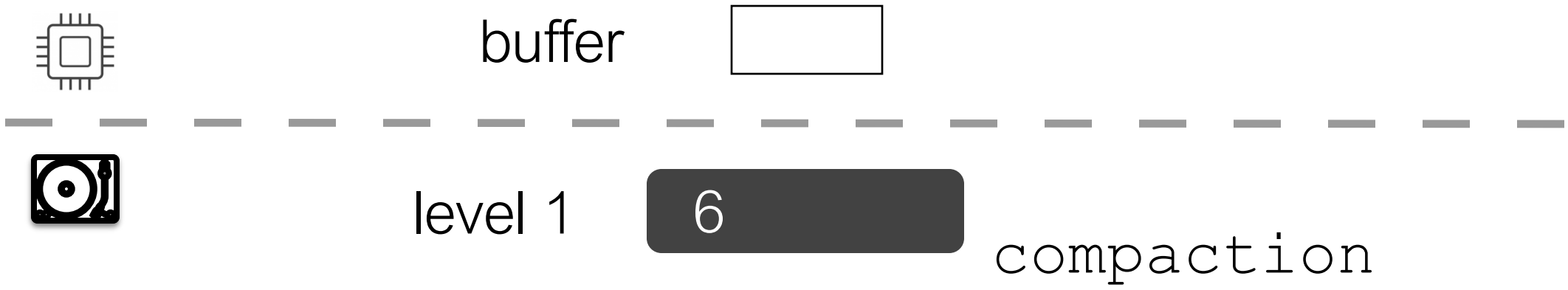
Out-of-place updates

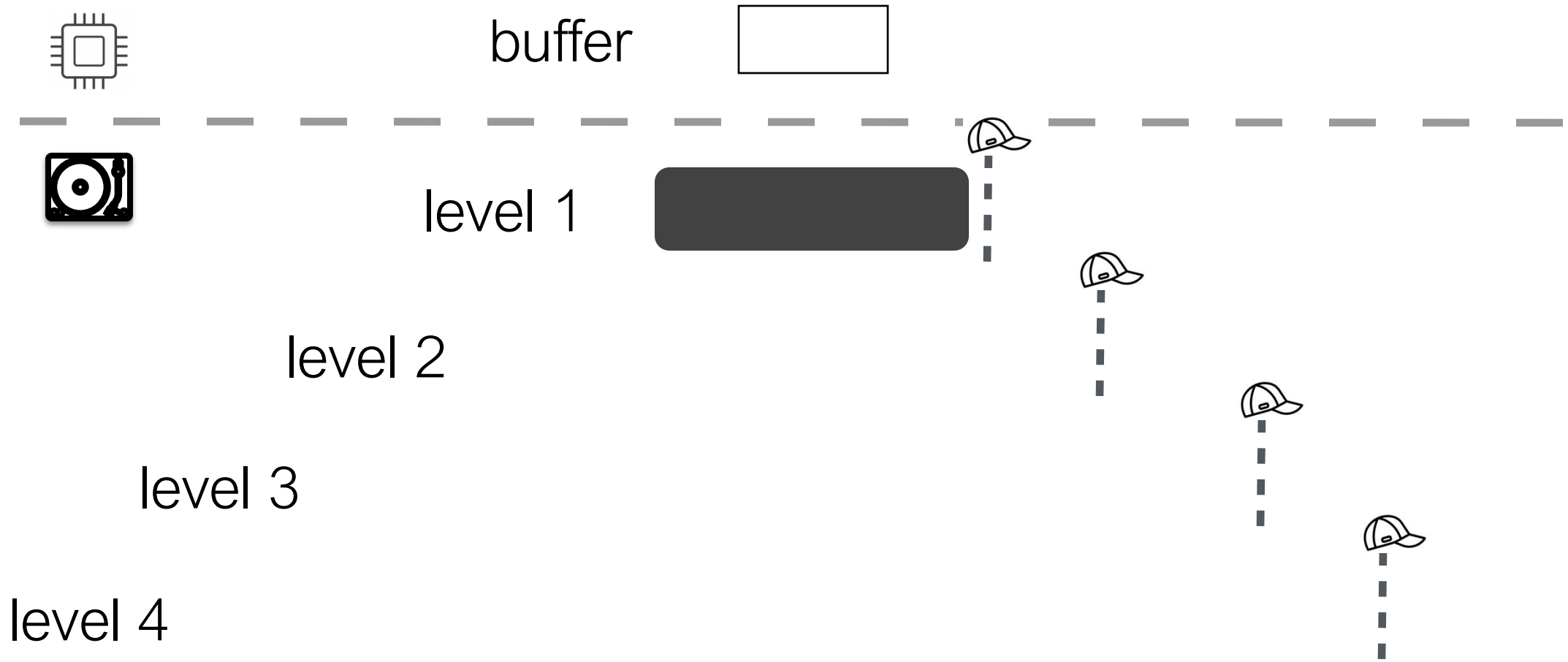
- fast ingestion
- space amplification
- slow reads

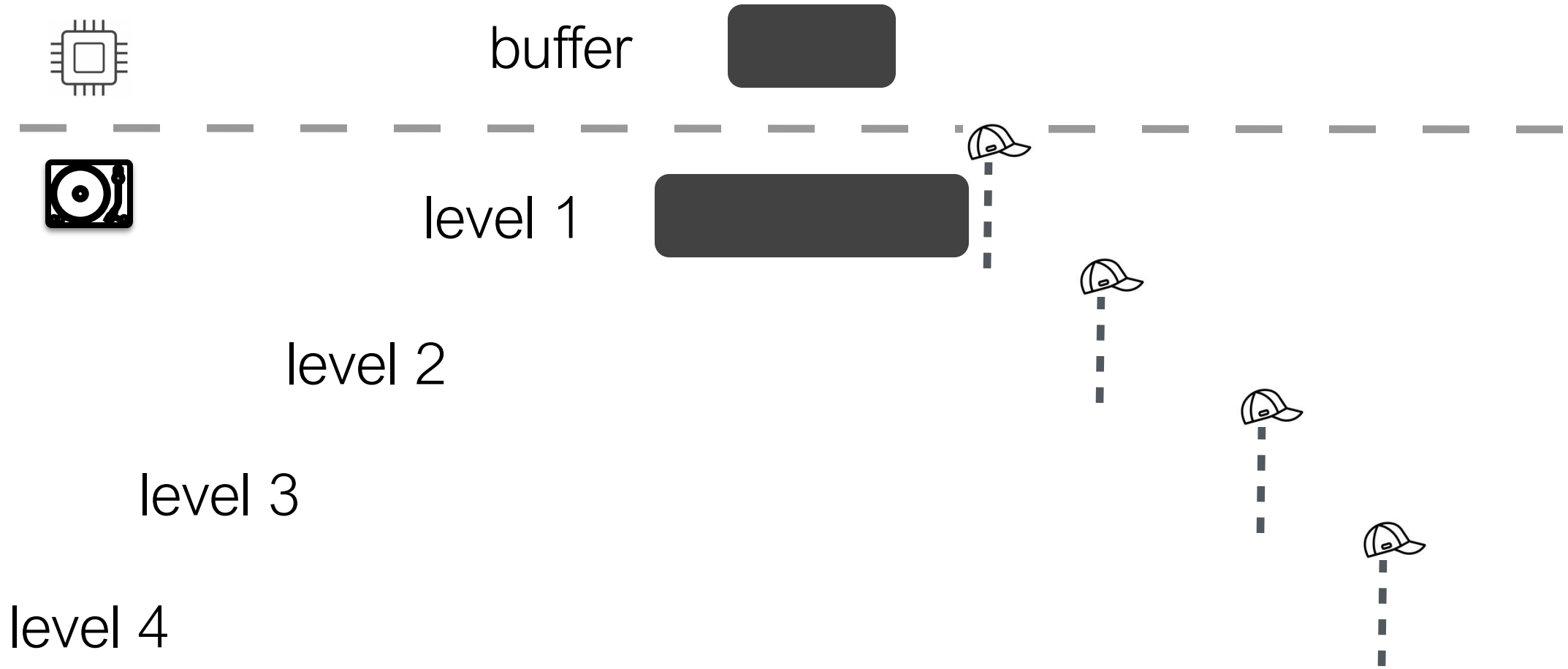


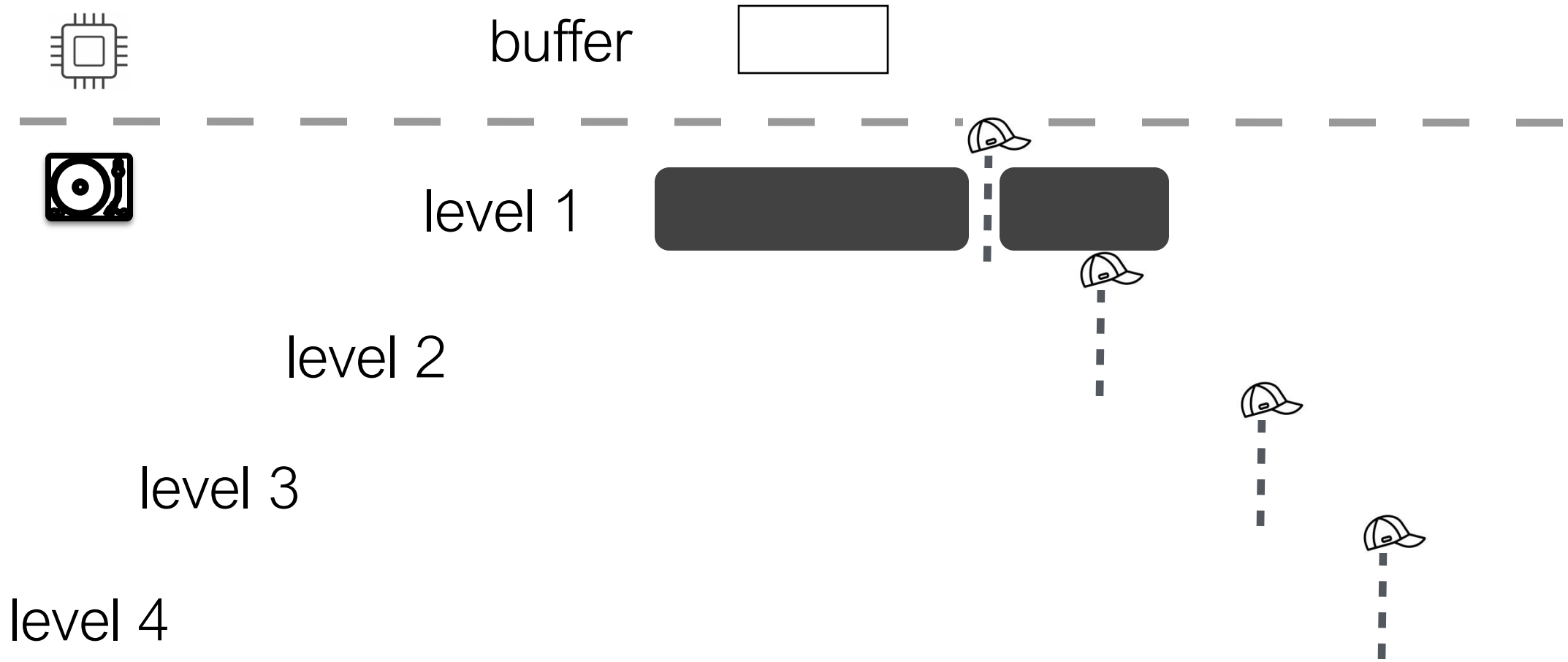
How do we reduce this space amplification?

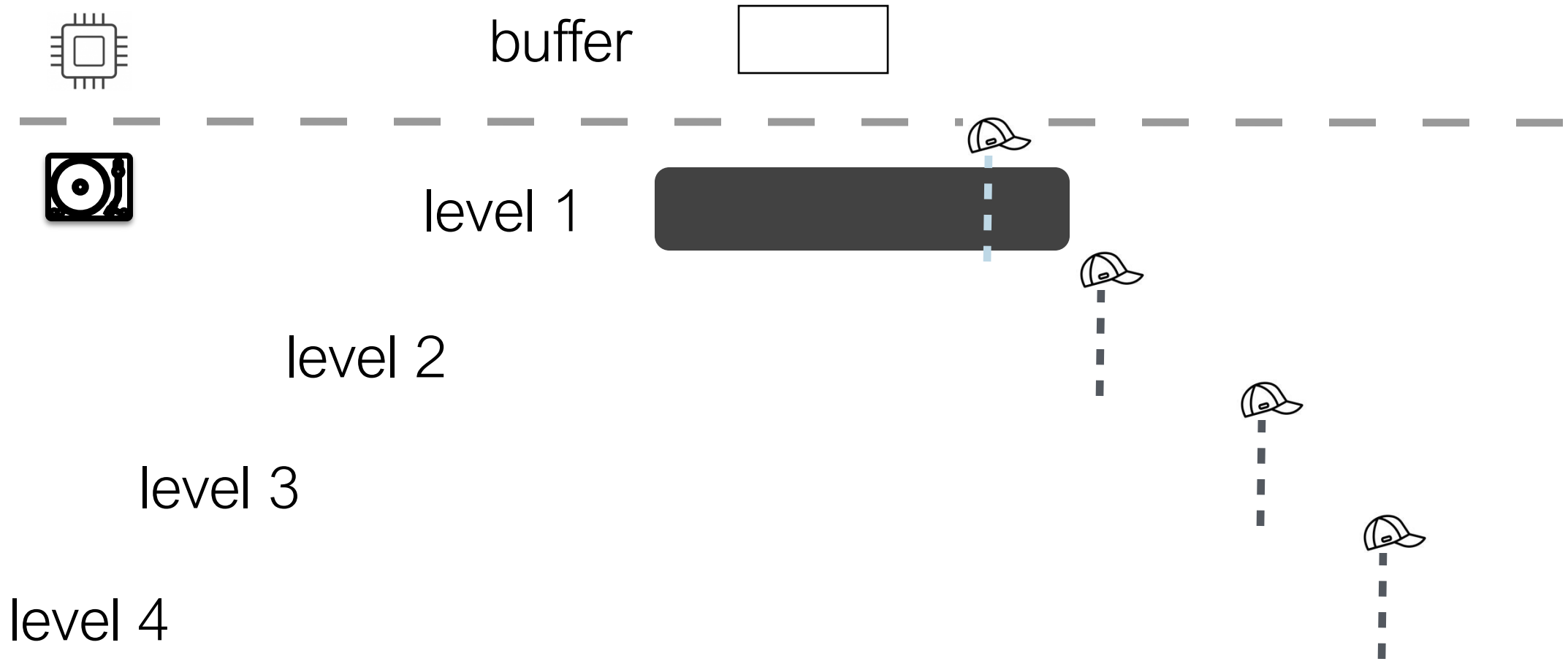


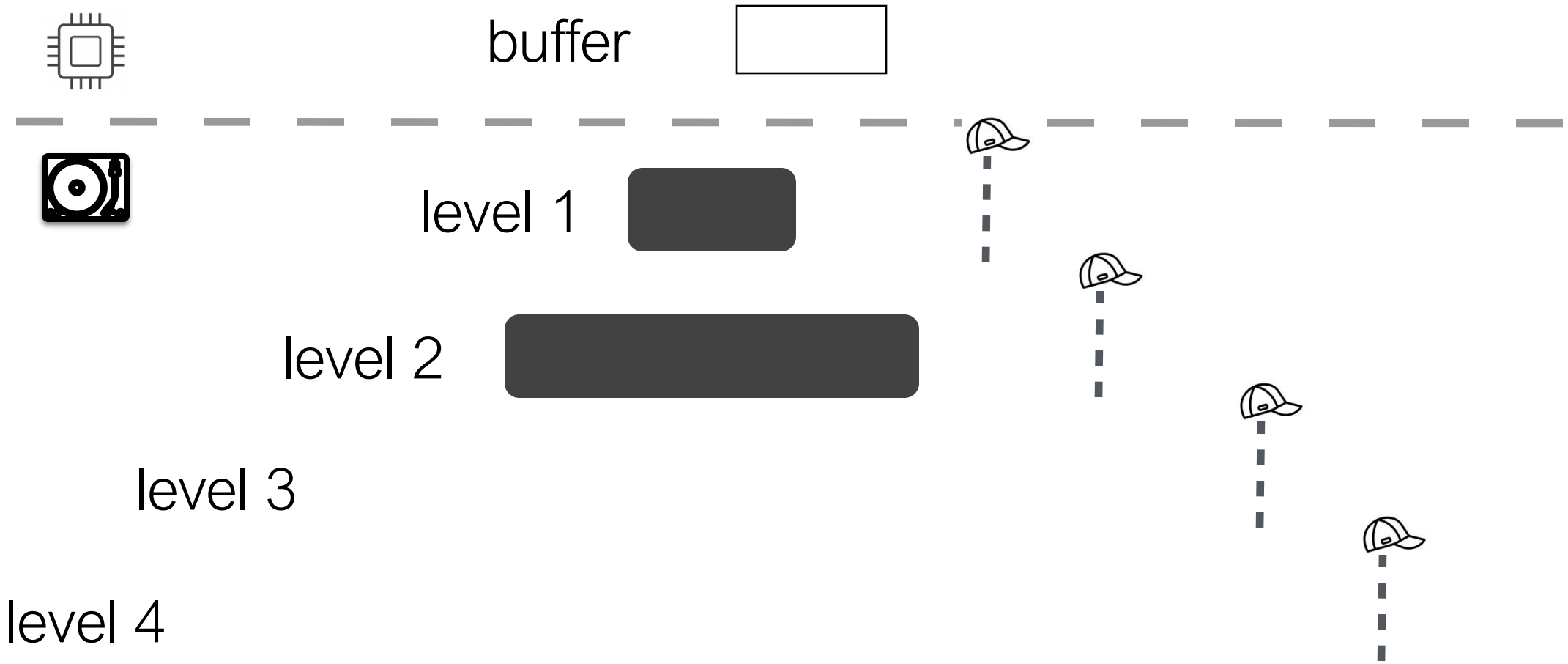


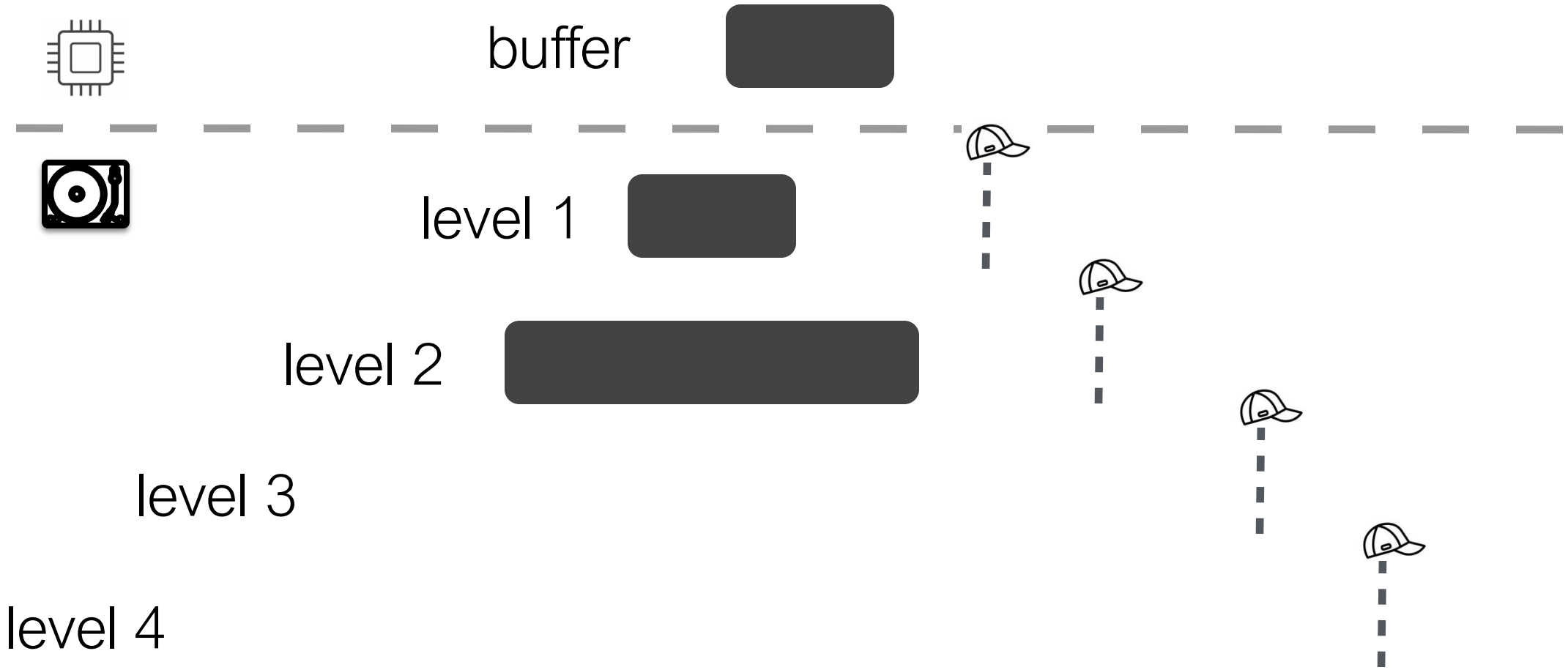


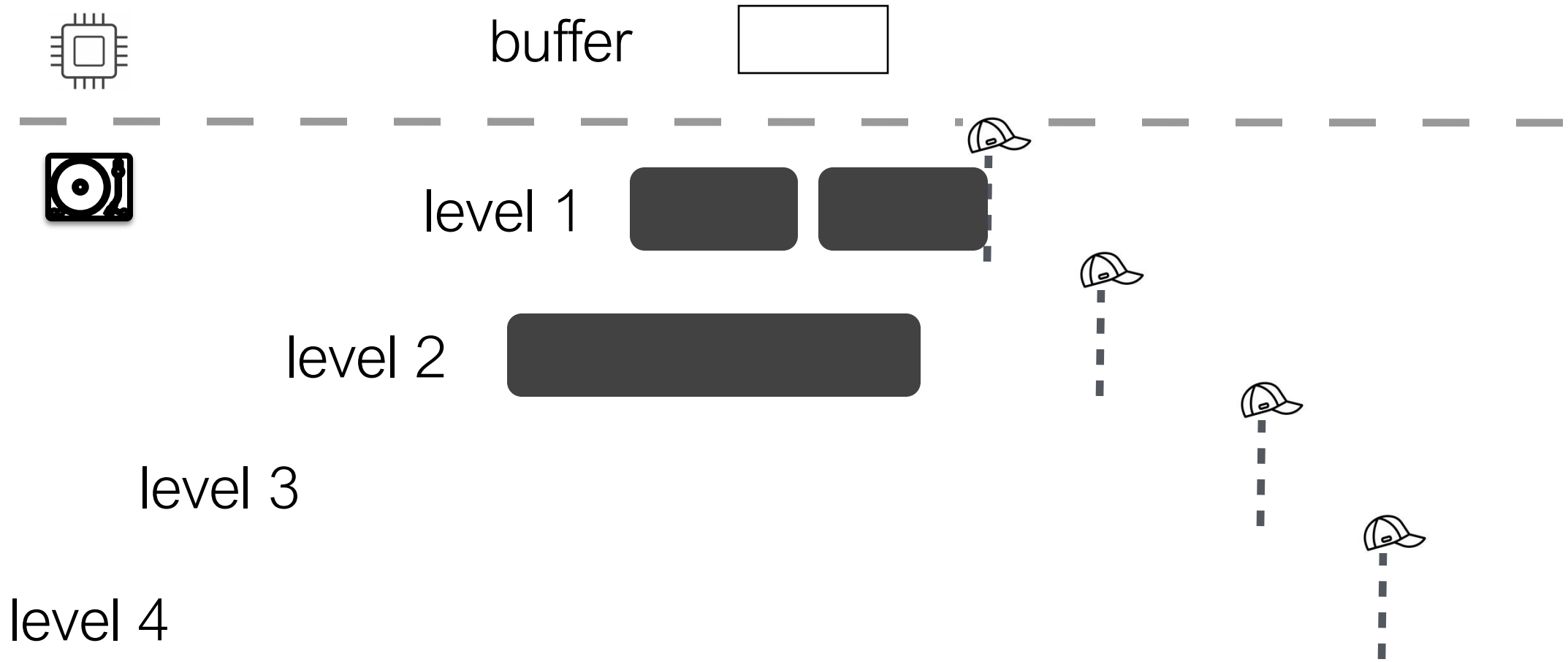


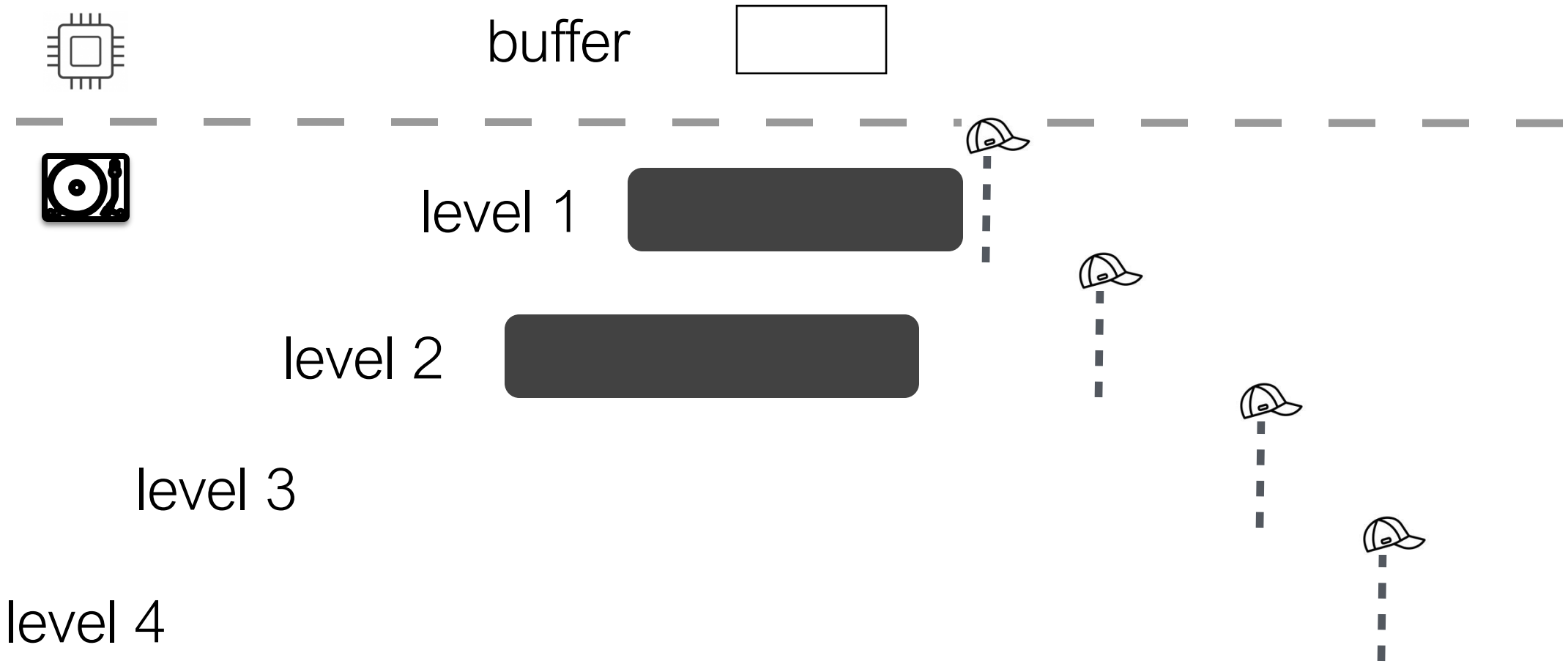


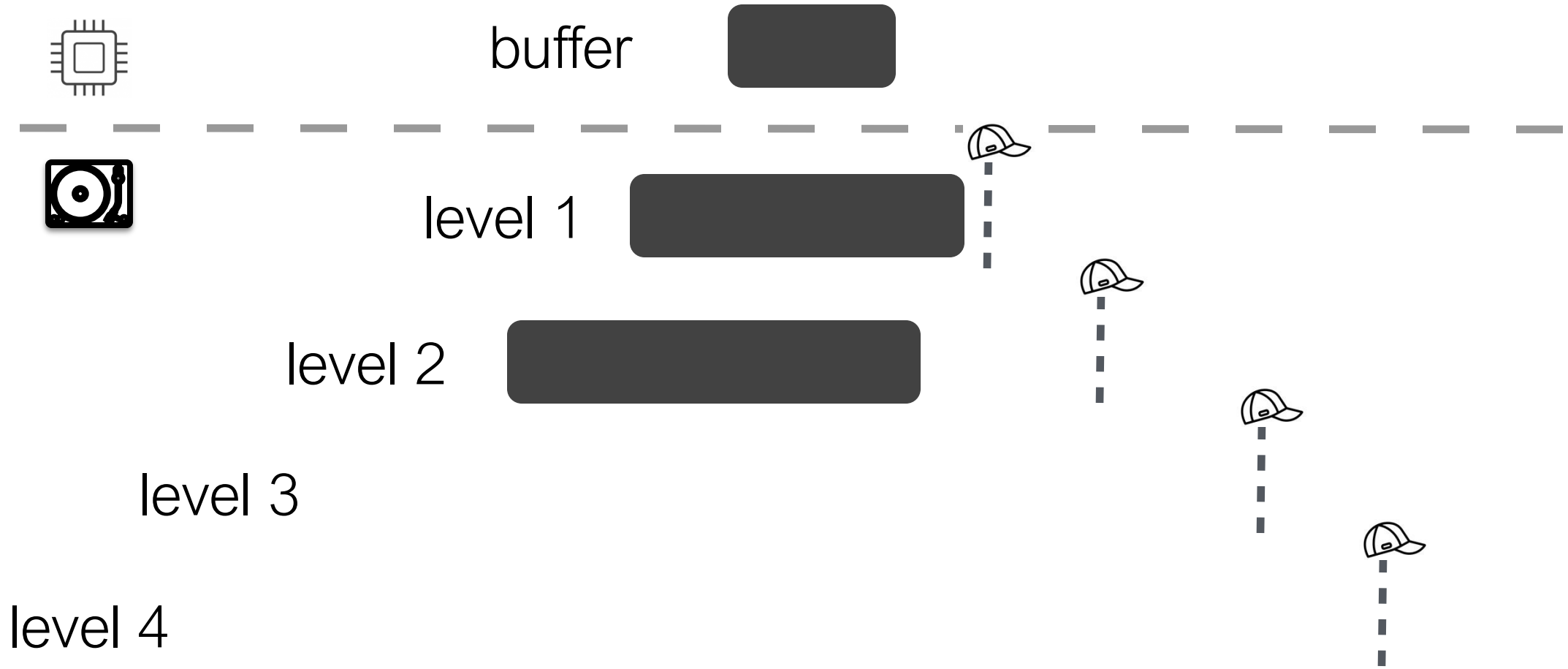


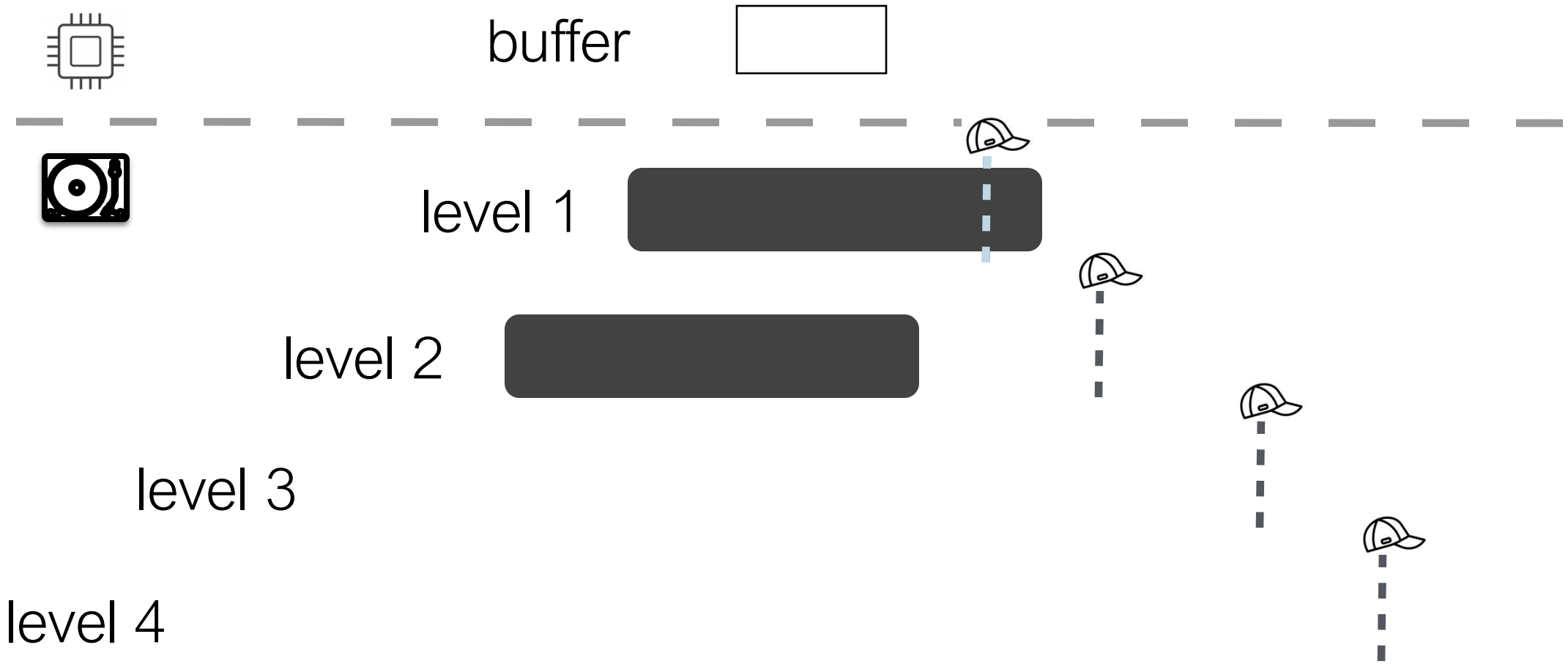


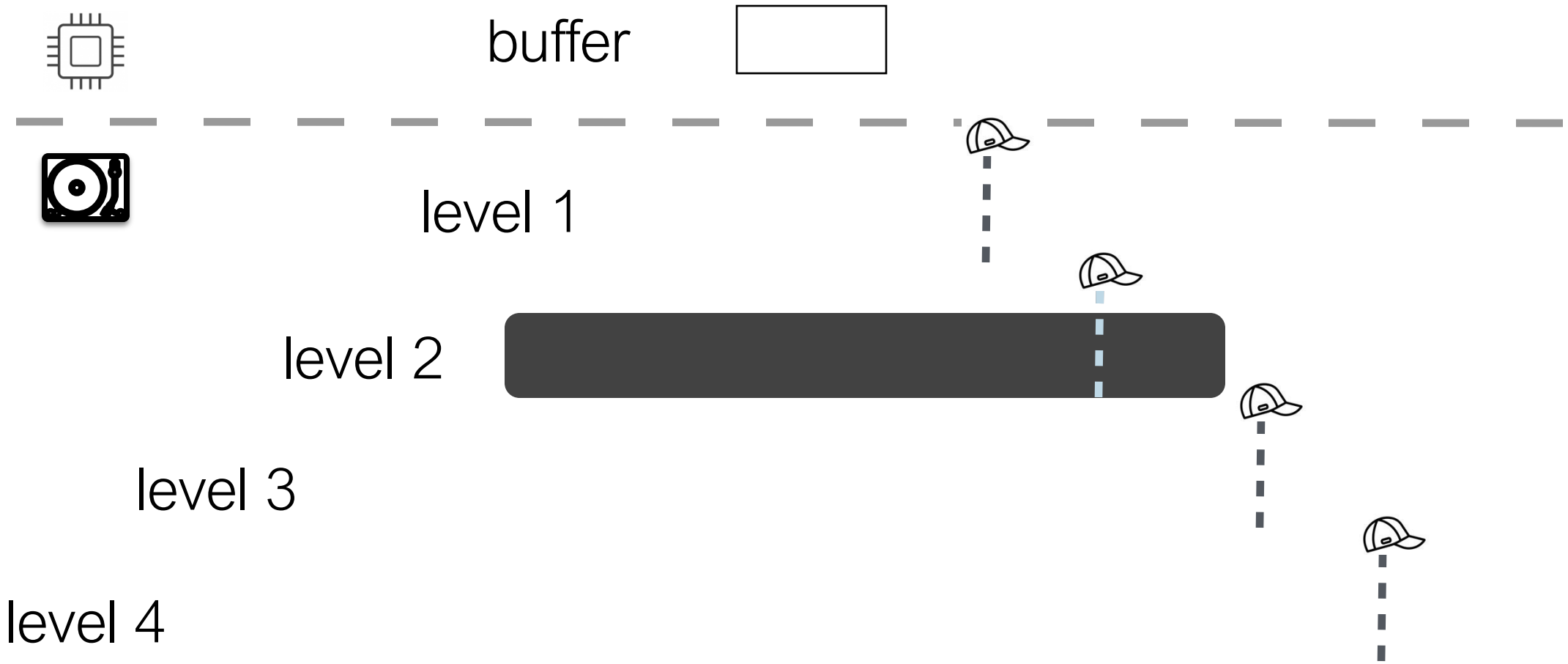


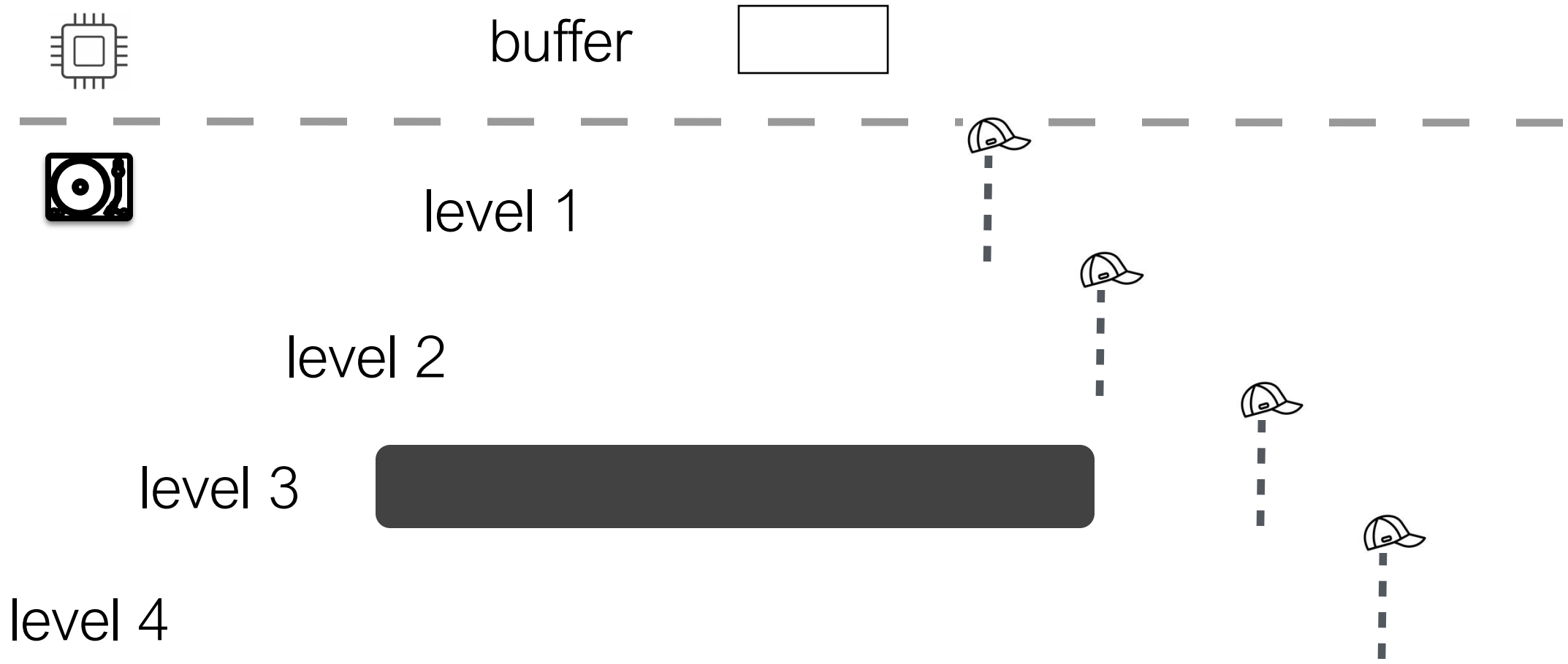








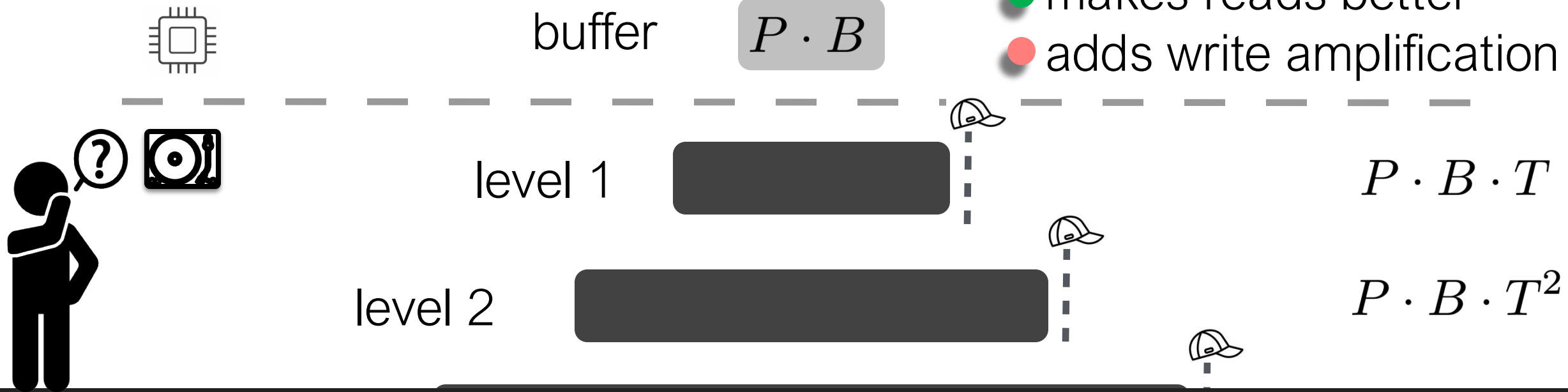

buffer
level 1
level 2
level 3
level 4



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

Periodic compactions

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



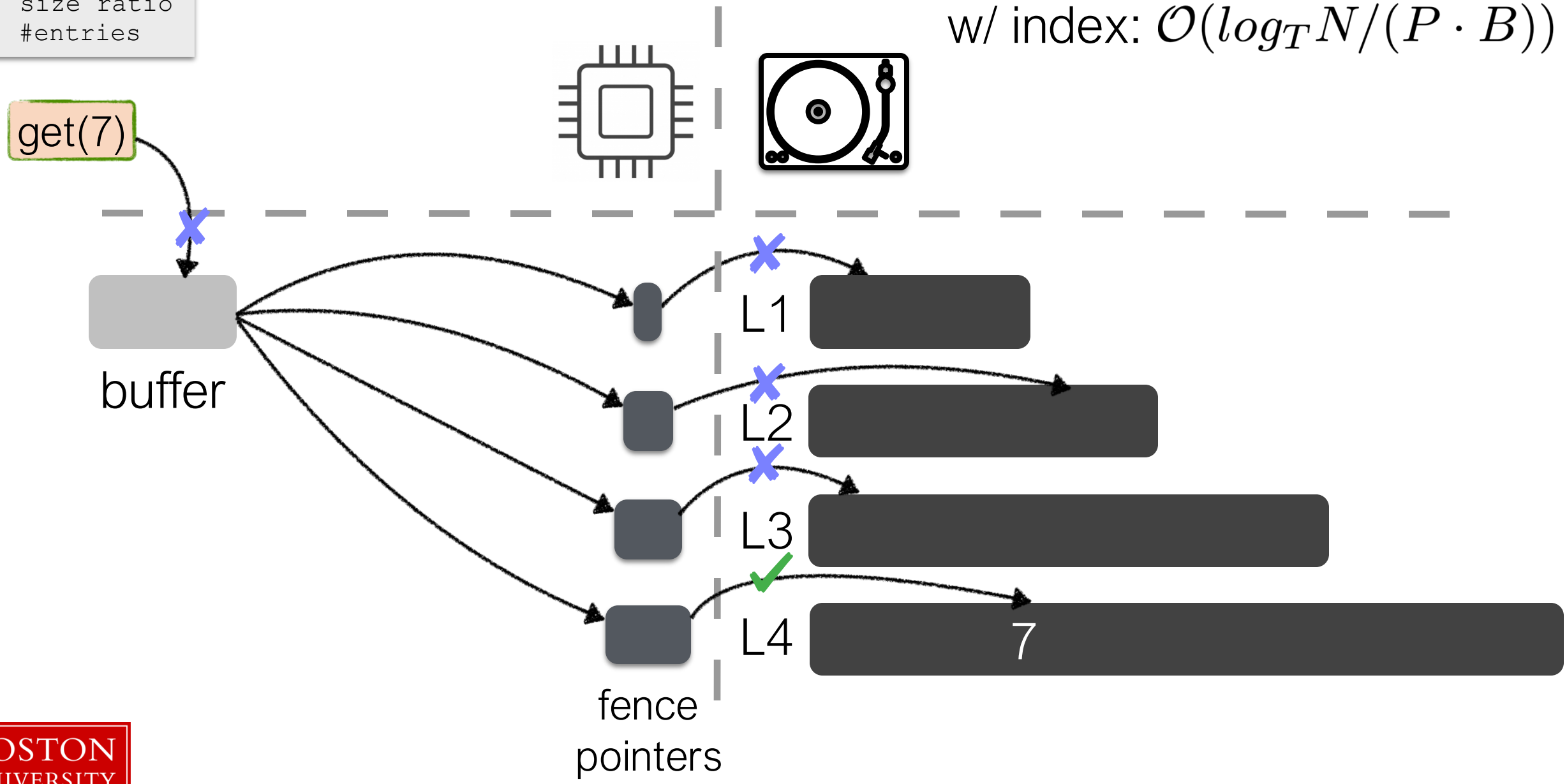
capacity (entries)

How about queries?

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

Cost analysis

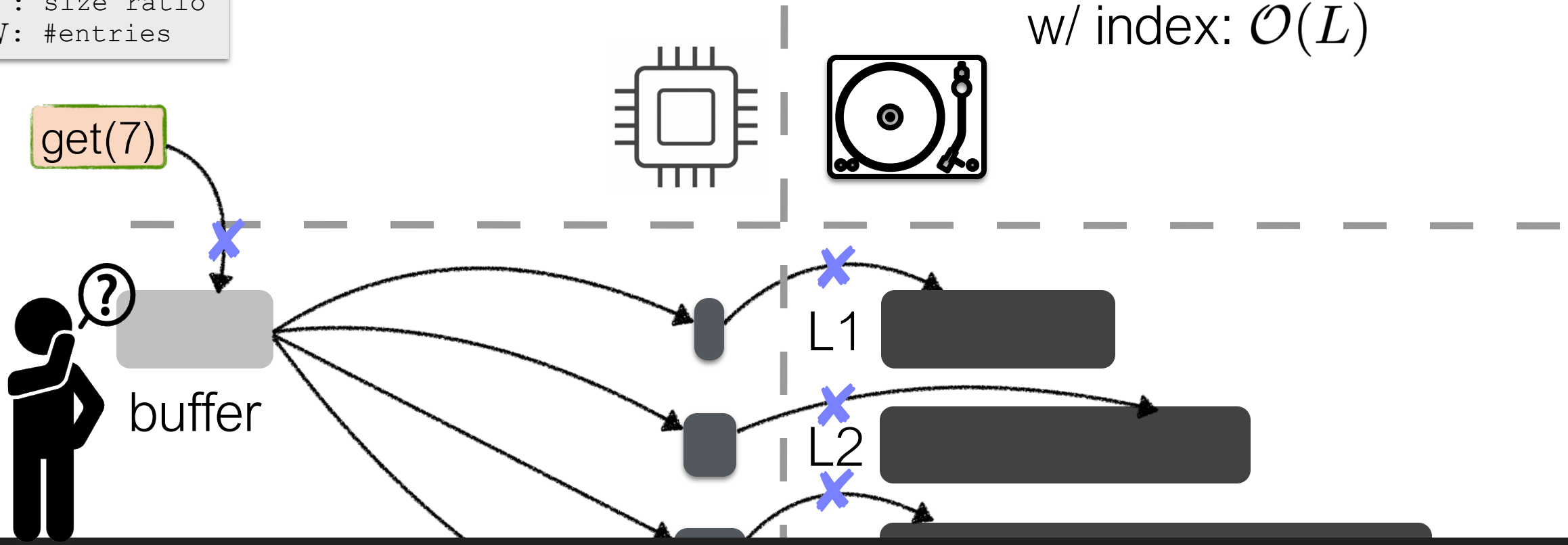
w/ index: $\mathcal{O}(log_T N/(P \cdot B))$



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

Cost analysis

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



How to avoid unnecessary I/Os?

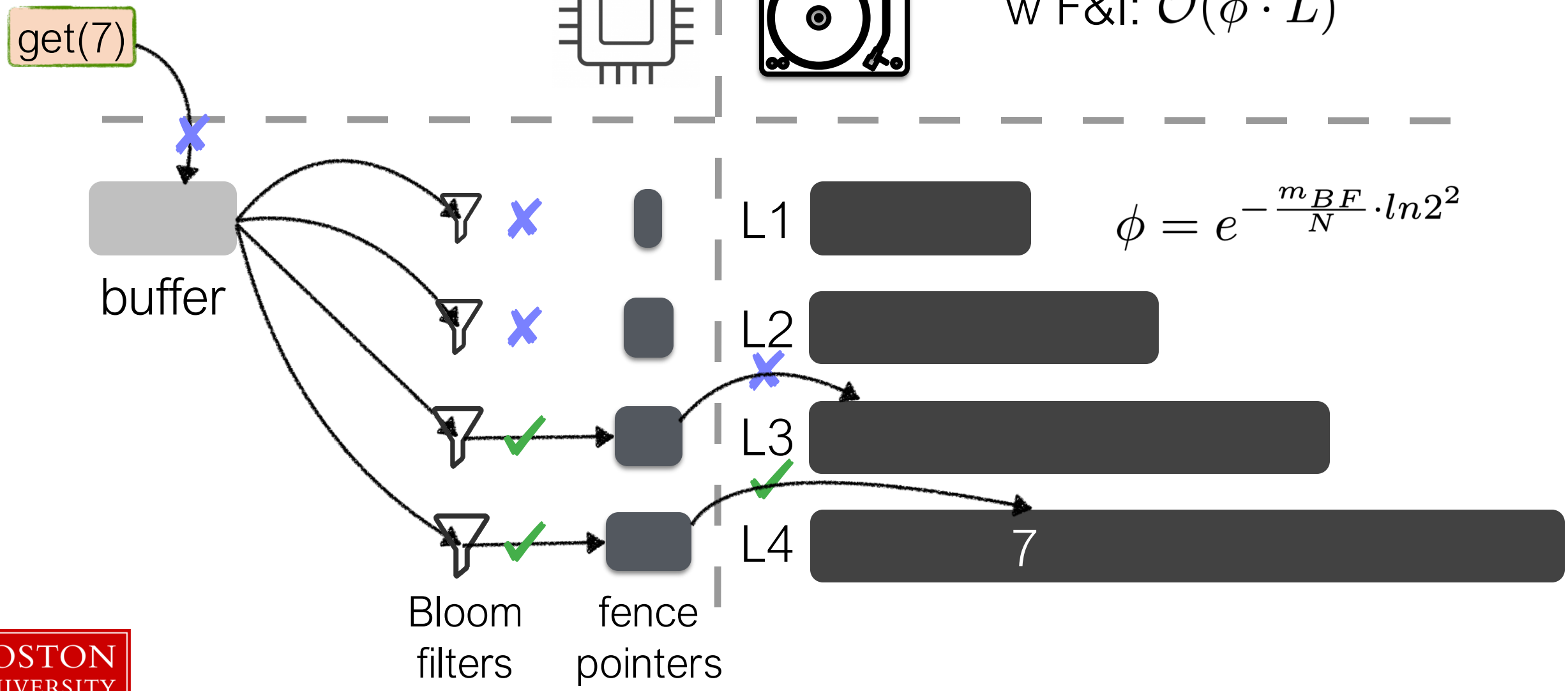
pointers

Cost analysis

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

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

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



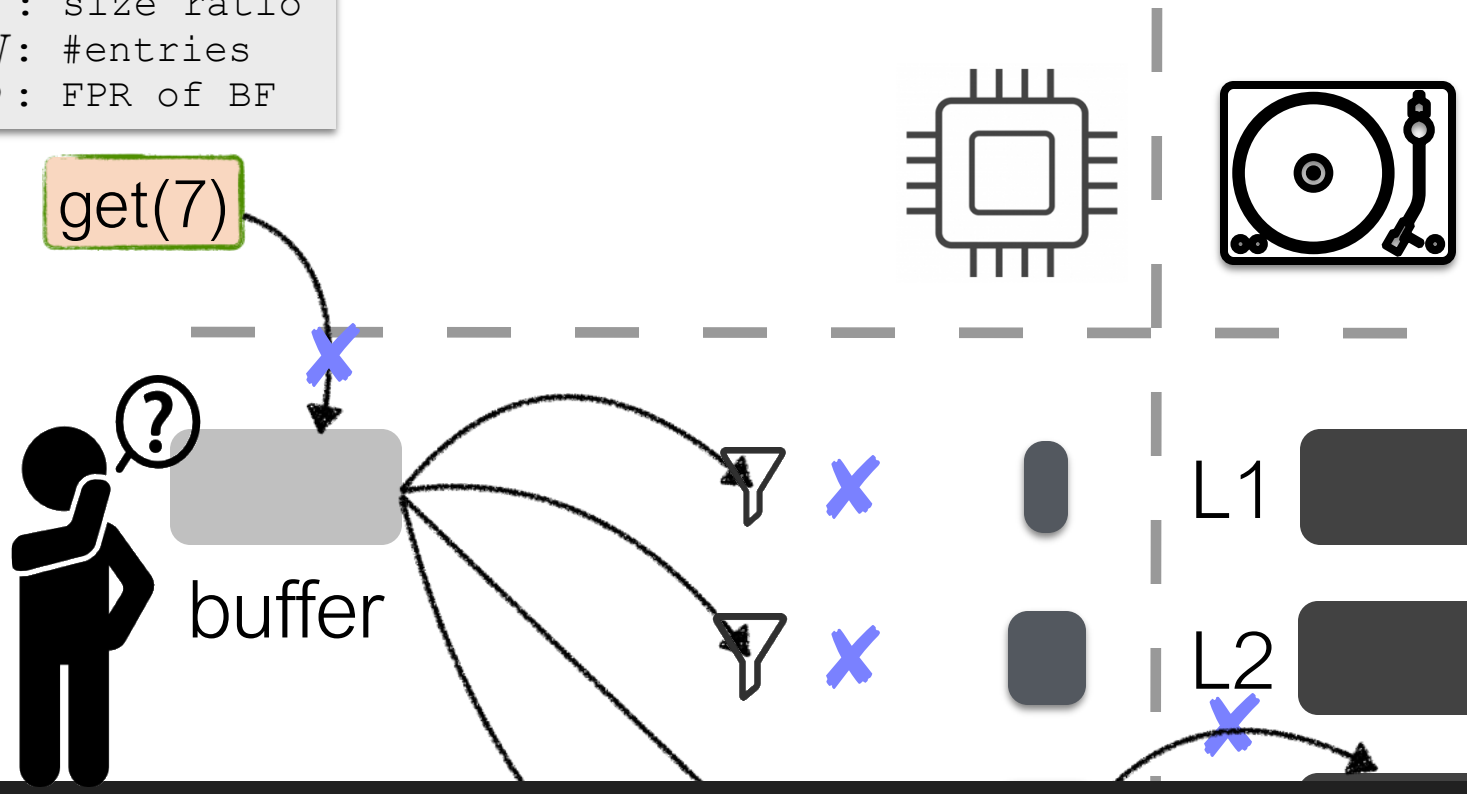
$$\phi = e^{-\frac{m_{BF}}{N} \cdot ln2^2}$$

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

Cost analysis

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?

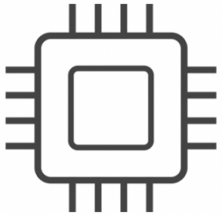
filters pointers

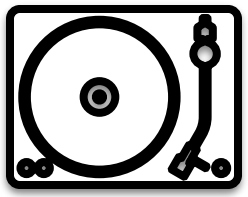
P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries
ϕ: FPR of BF

Cost analysis

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

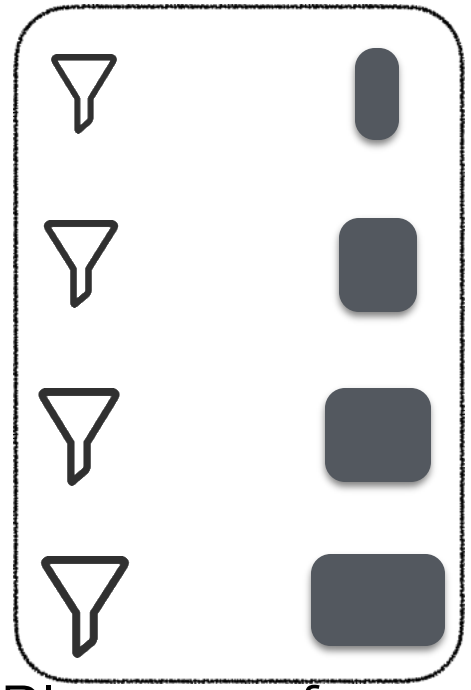
w F&I: $\mathcal{O}(\phi \cdot L)$





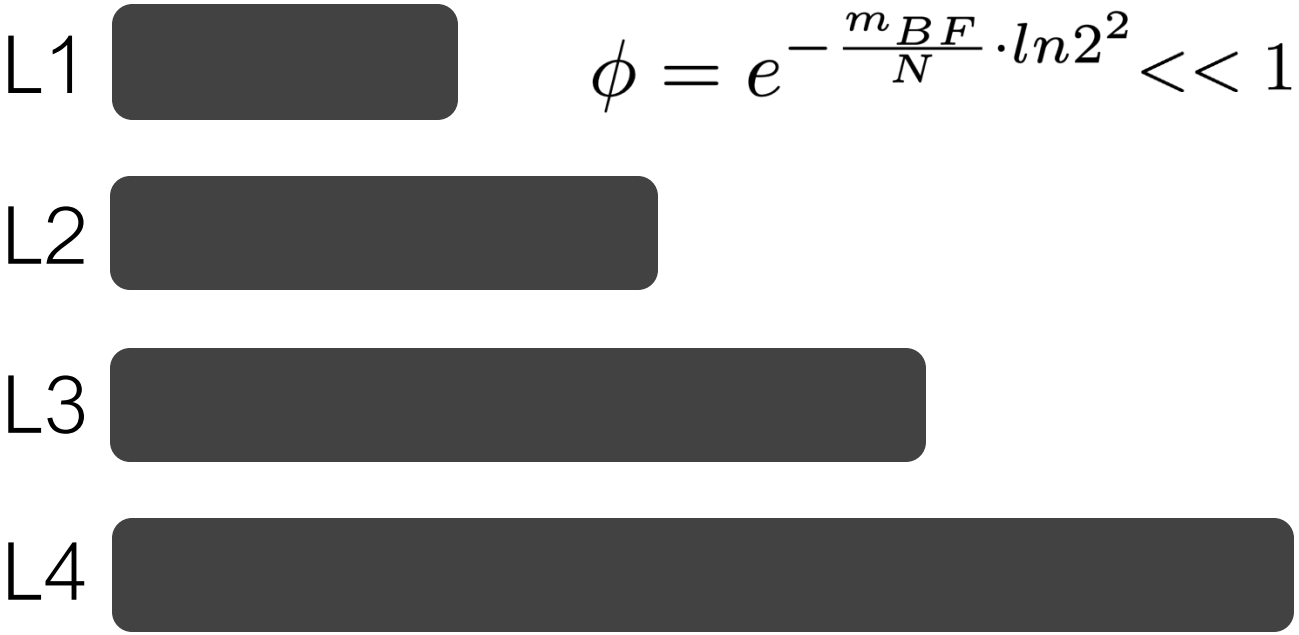
buffer

block cache



Bloom filters

fence pointers

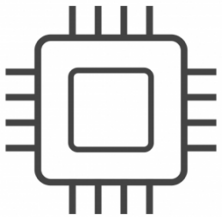


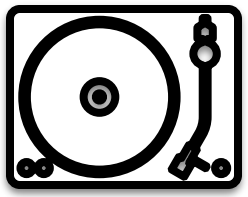
$$\phi = e^{-\frac{m_{BF}}{N} \cdot ln2^2} << 1$$

BOSTON UNIVERSITY

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

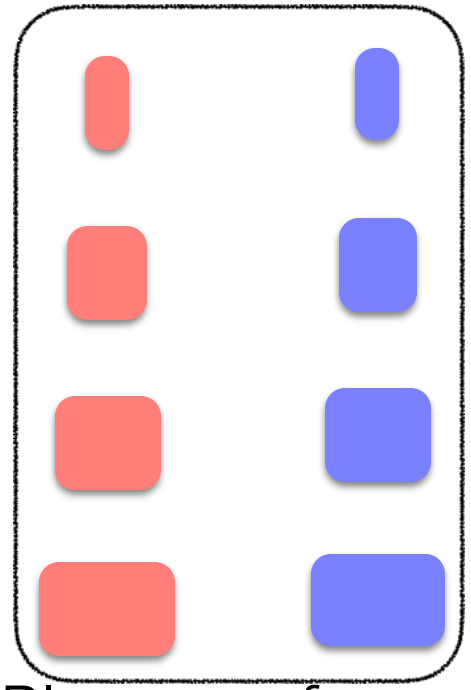
Block Cache





buffer

block cache

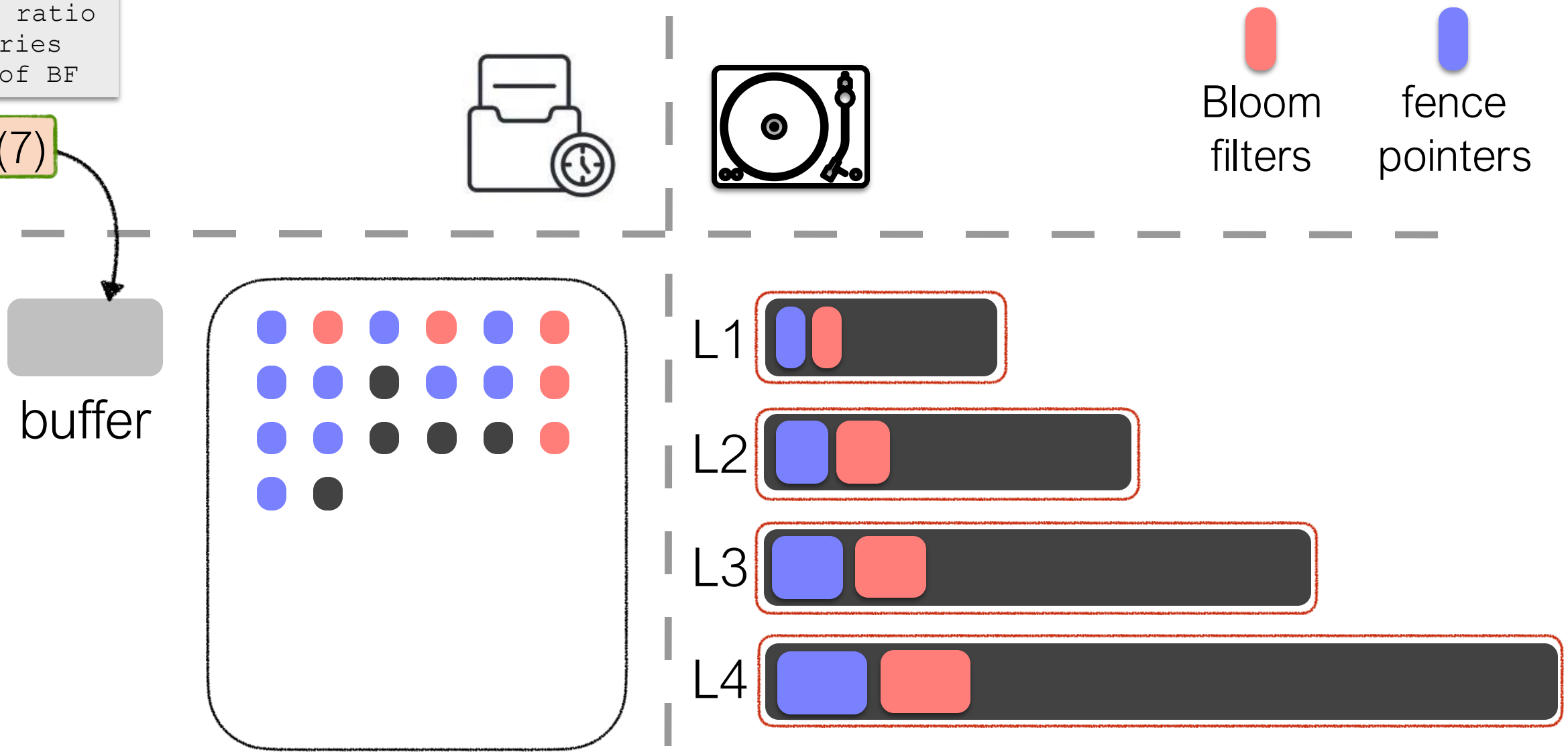


Bloom filters

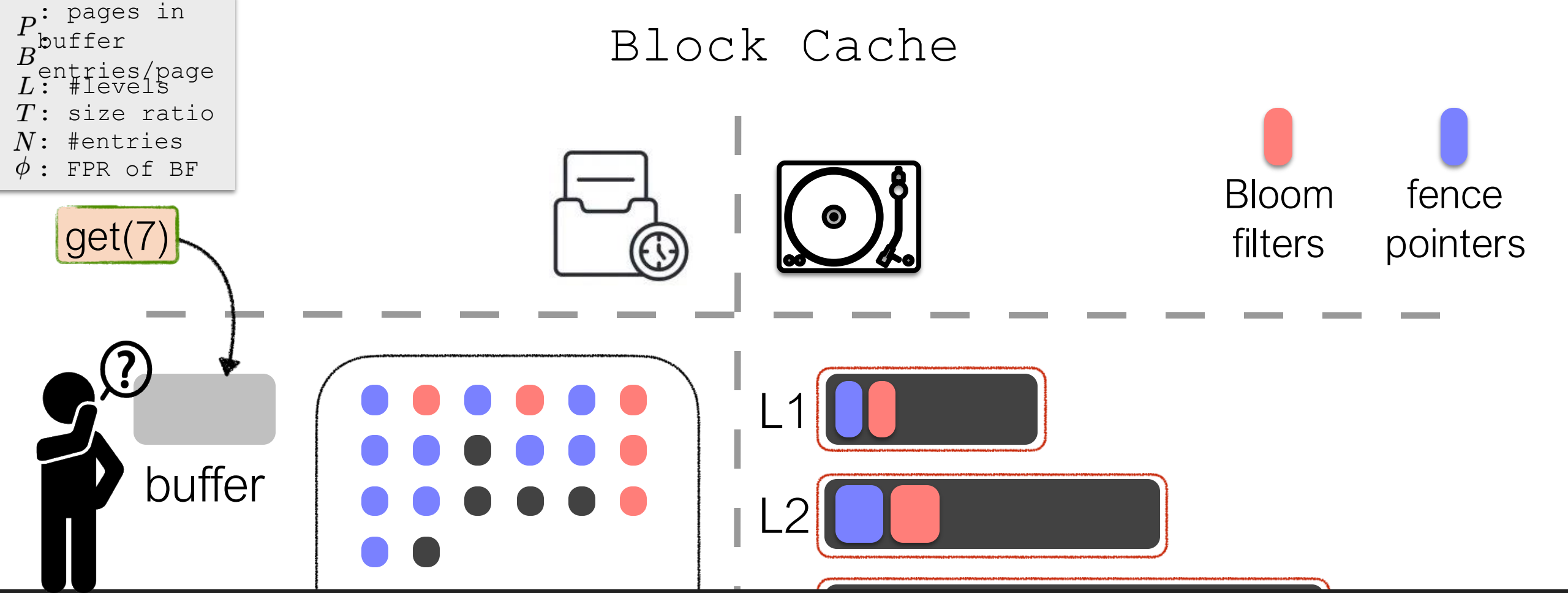
fence pointers

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

Block Cache



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

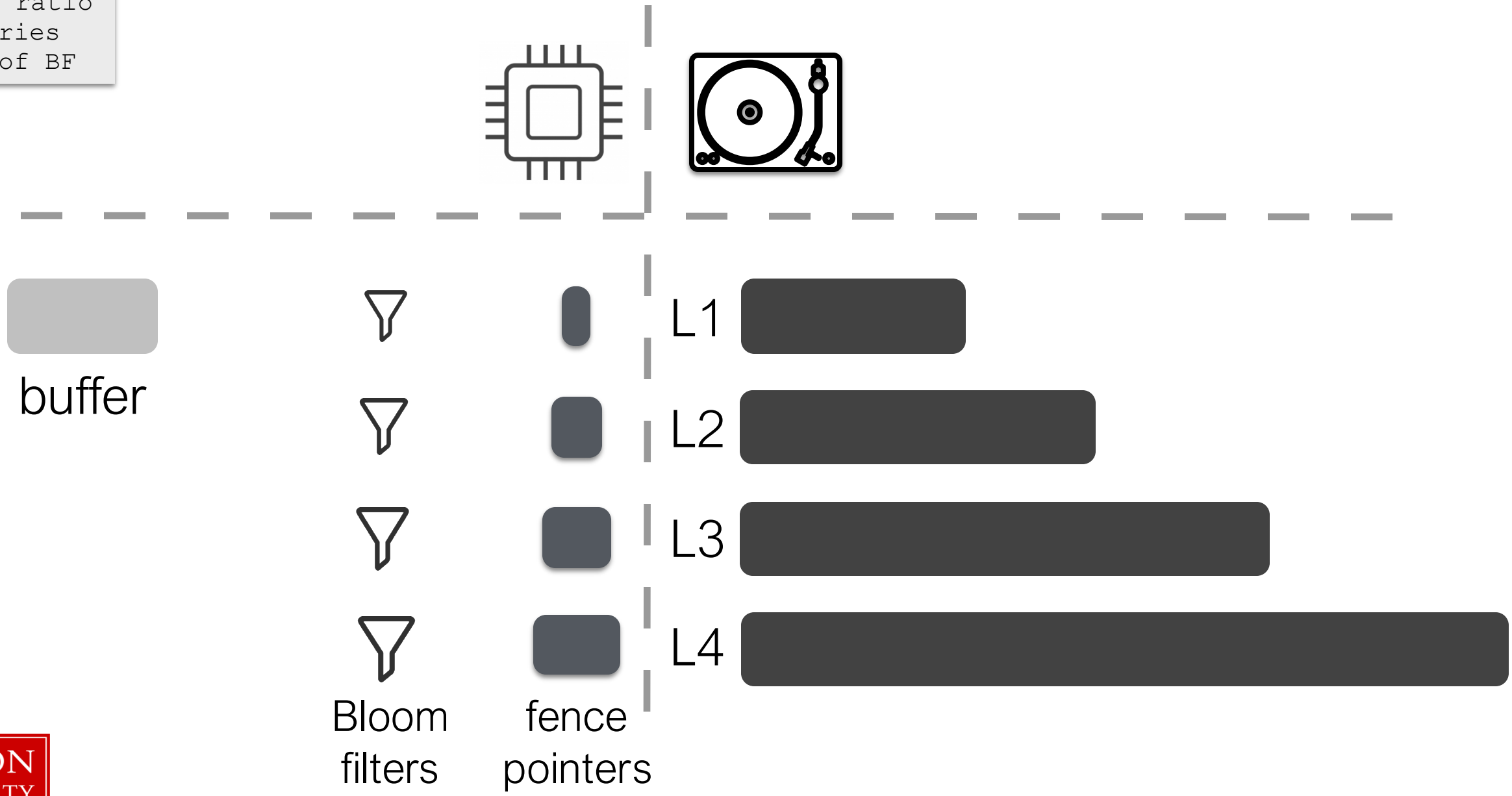


What about range queries?

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

s selectivity
LRQ

Range Queries



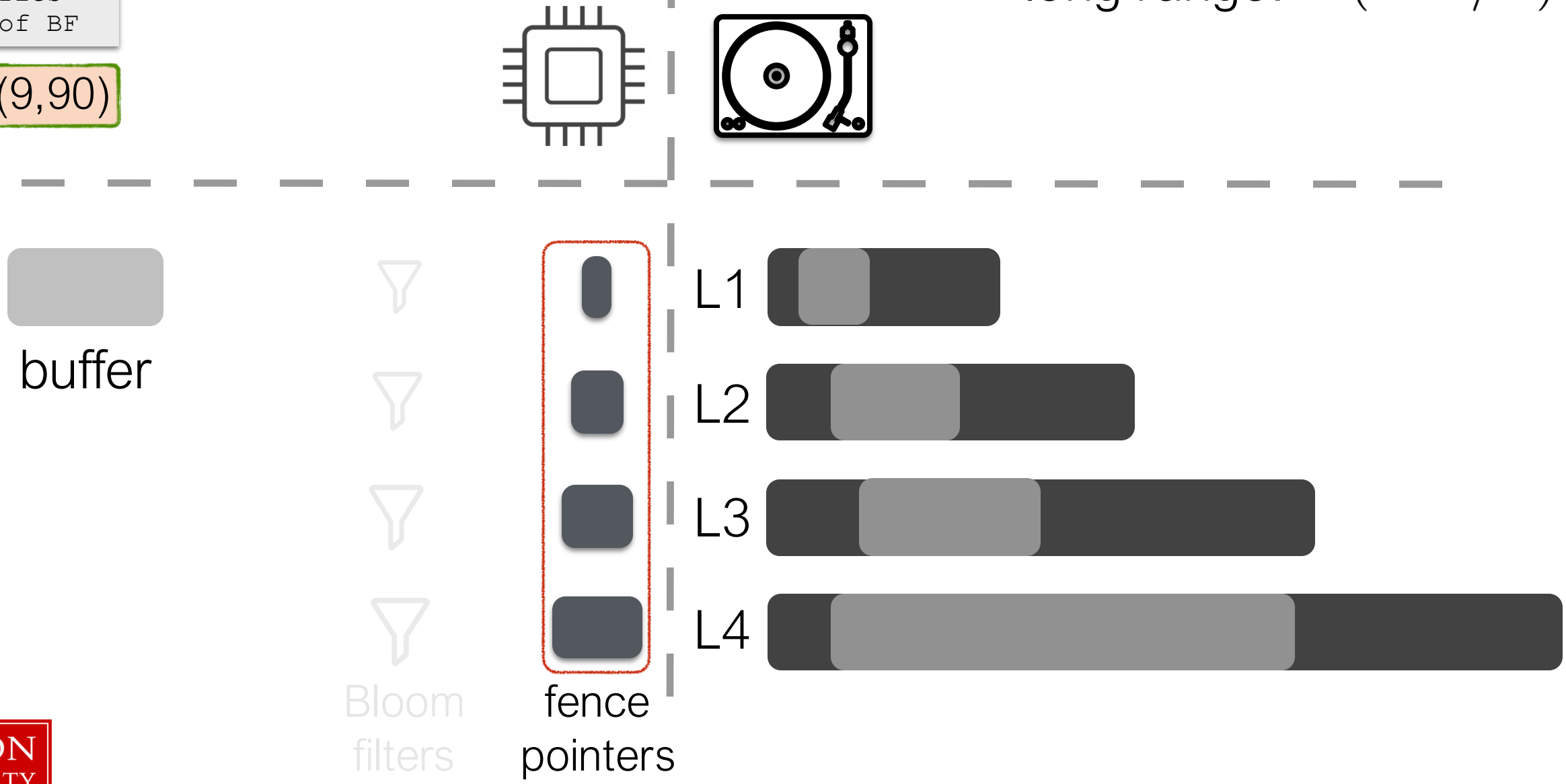
P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries
ϕ: FPR of BF

s selectivity
LRQ

Range Queries Cost analysis

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

get(9,90)



BOSTON UNIVERSITY

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

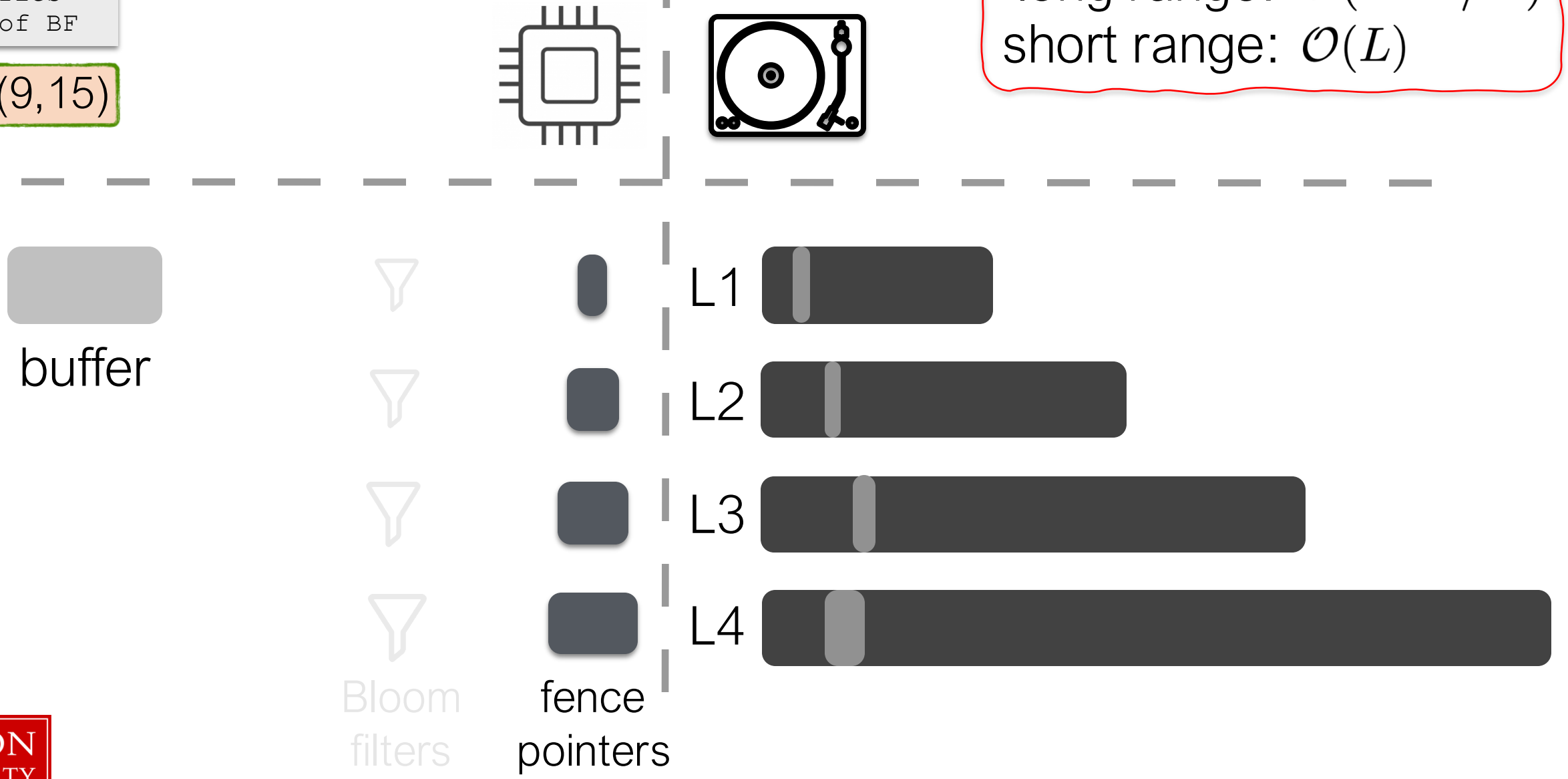
s selectivity
SRQ

Range Queries

Cost analysis

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

get(9,15)



BOSTON UNIVERSITY

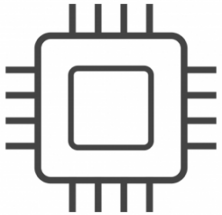
P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries
ϕ: FPR of BF

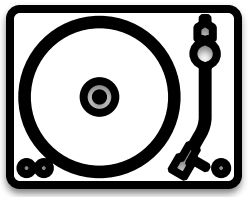
Going back to point queries

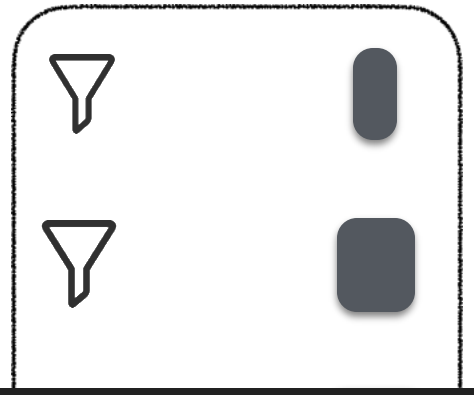
Cost analysis

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

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







L1

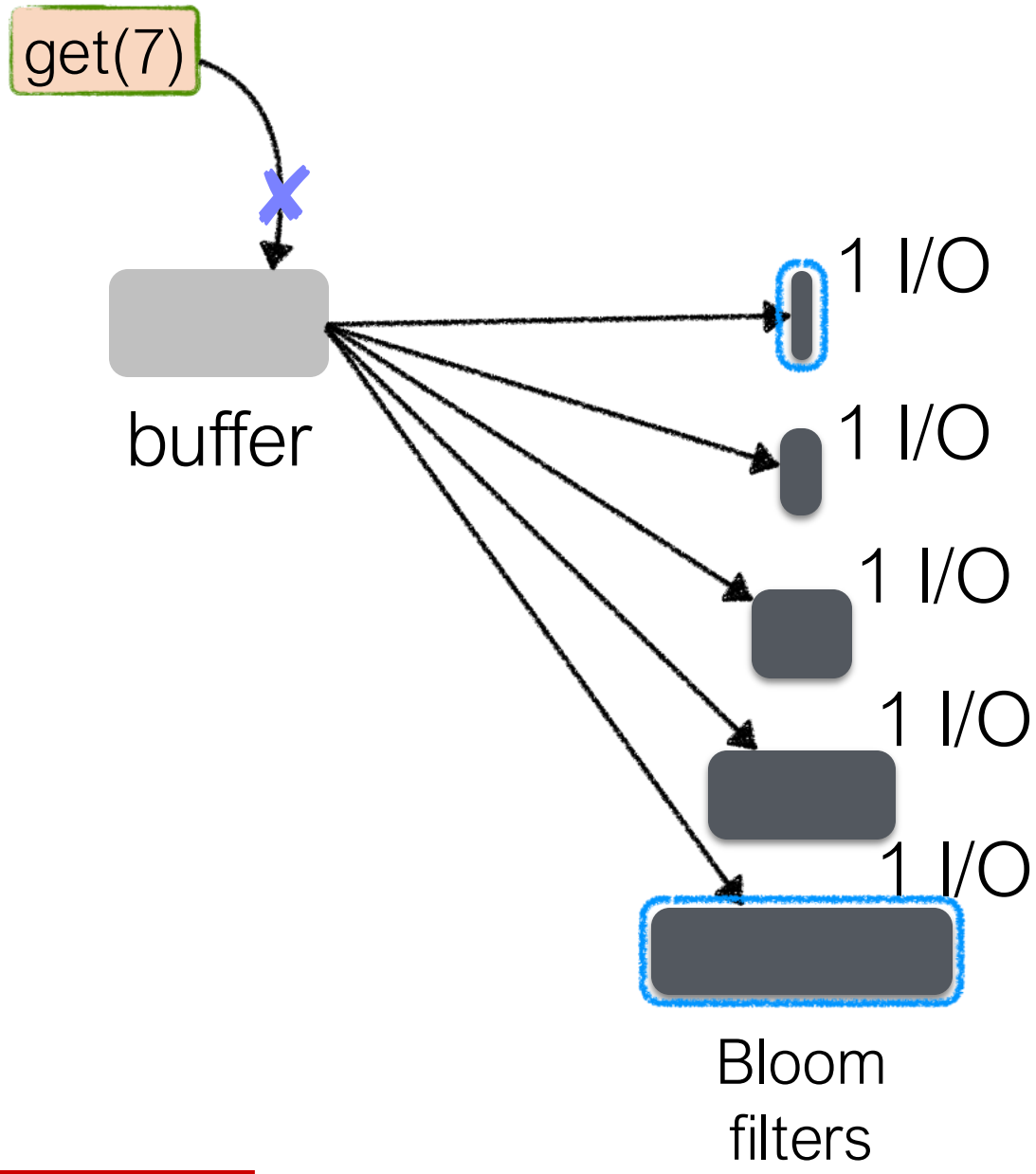
L2

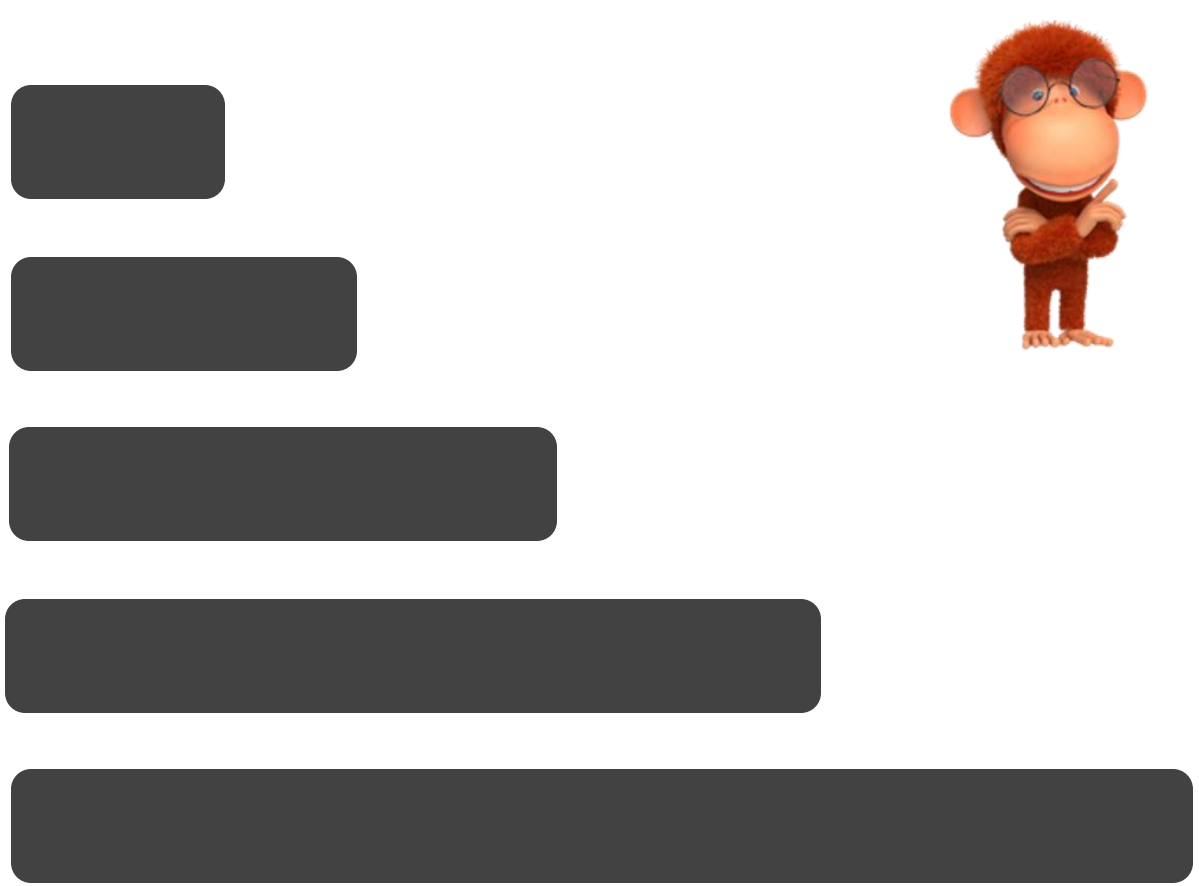
$$\phi = e^{-\frac{m_{BF}}{N} \cdot ln2^2} << 1$$

Should all BFs be equally accurate?

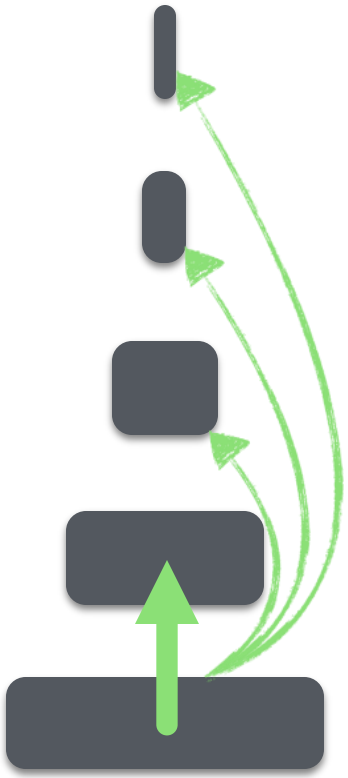
filters pointers

Insight: most of the memory allocated to BFs is helping us save only 1 I/O (90% for T=10)





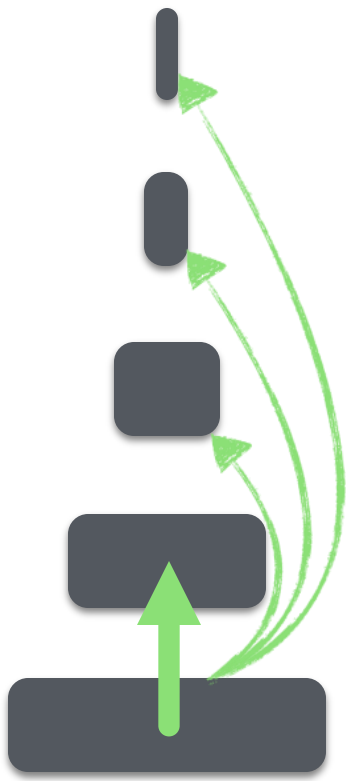
Bloom filters



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

exponentially decreasing

Bloom filters



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

exponentially decreasing

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

point lookup cost

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

$$\vee$$

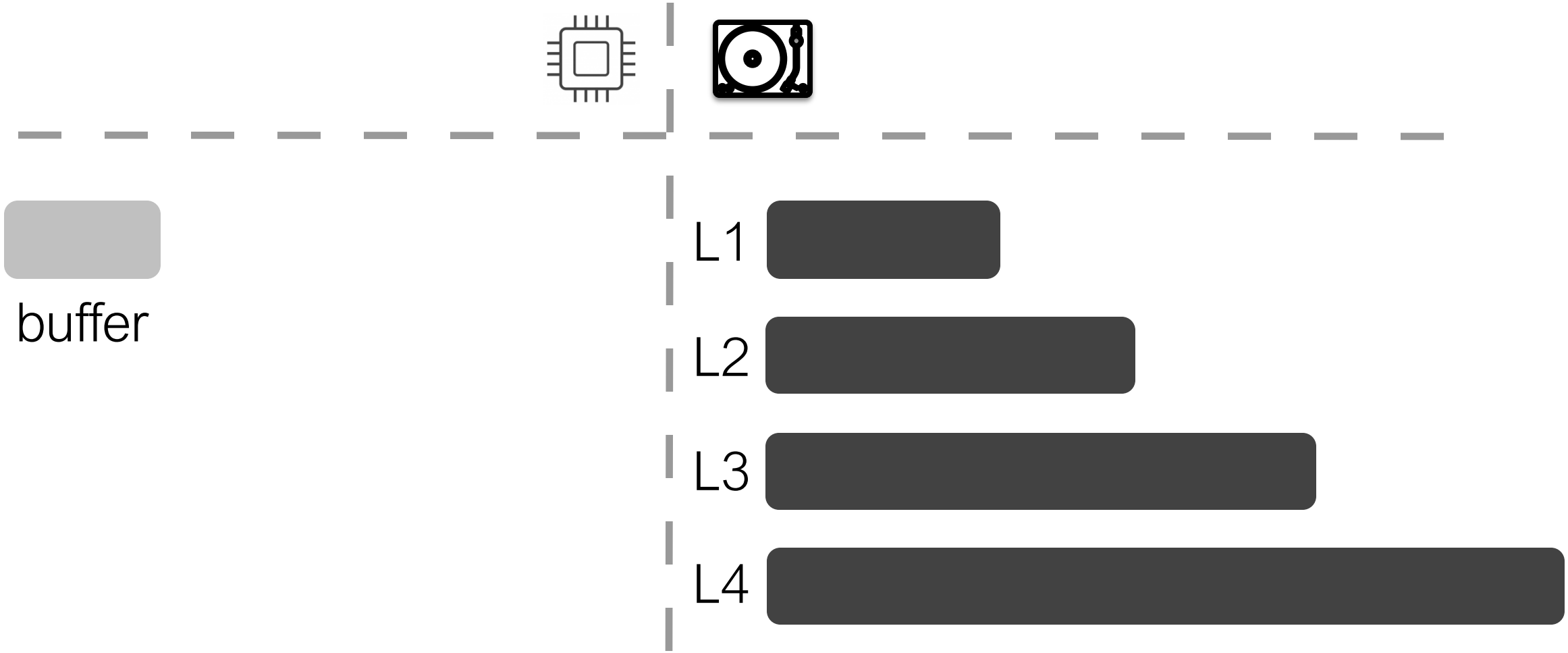
$$\mathcal{O}(c \cdot \phi_0)$$

BOSTON UNIVERSITY

DayanSIGMOD17

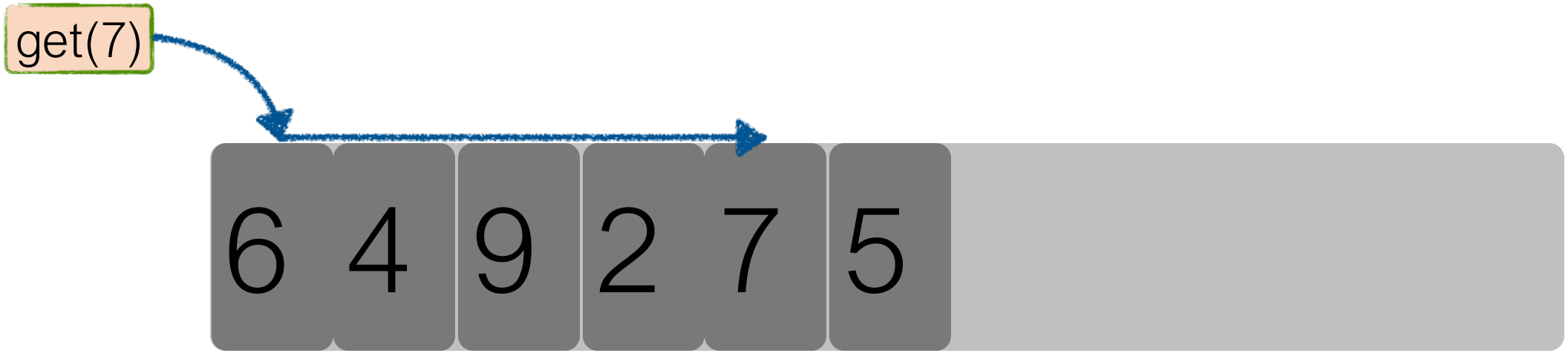
Buffer Optimizations

Buffer Optimizations



P: pages in buffer
B: entries/page

Buffer Implementation: **vector**



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

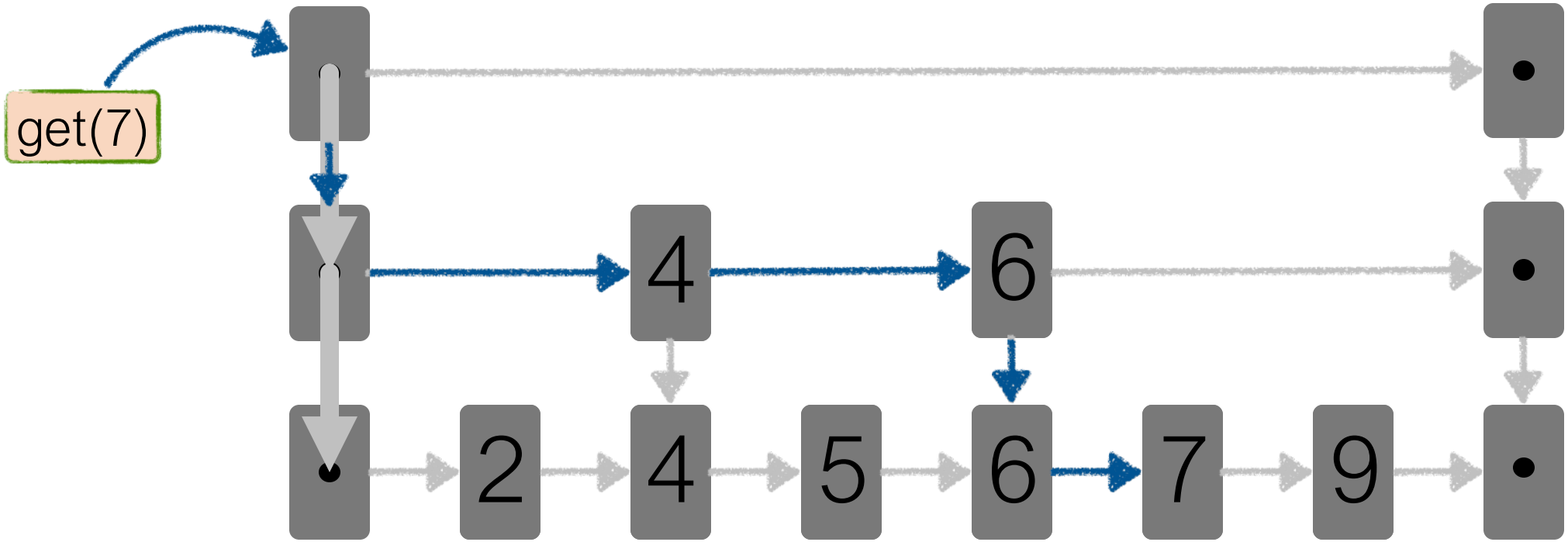
ingestion cost: $\mathcal{O}(1)$

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

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

BOSTON UNIVERSITY

Buffer Implementation: **skiplist**



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

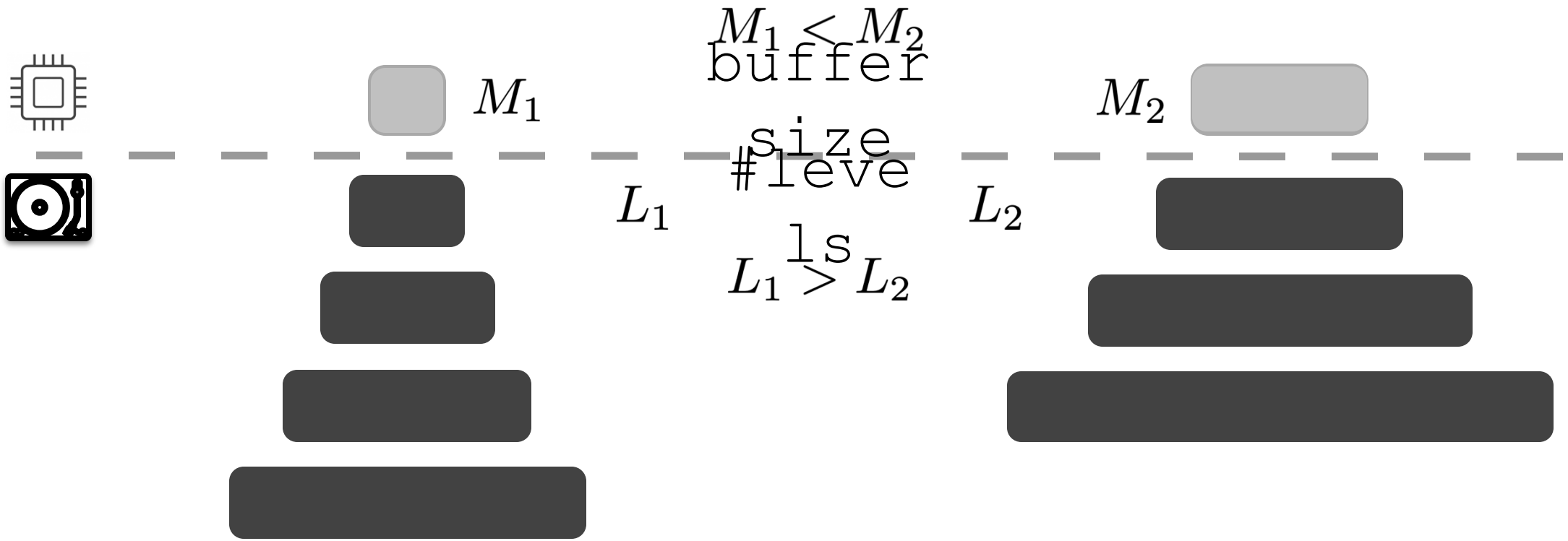
P: pages in buffer
B: entries/page

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

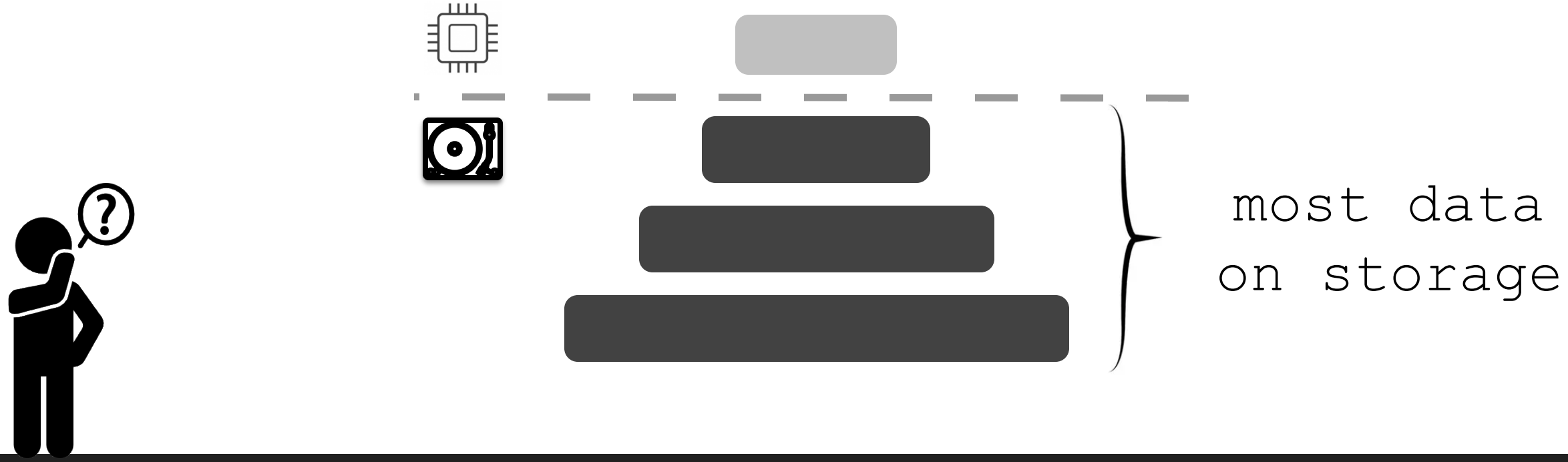
BOSTON UNIVERSITY

Size of the Buffer



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

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



How does the storage layer affect ingestion?

L: #levels
T: size ratio

most data on storage

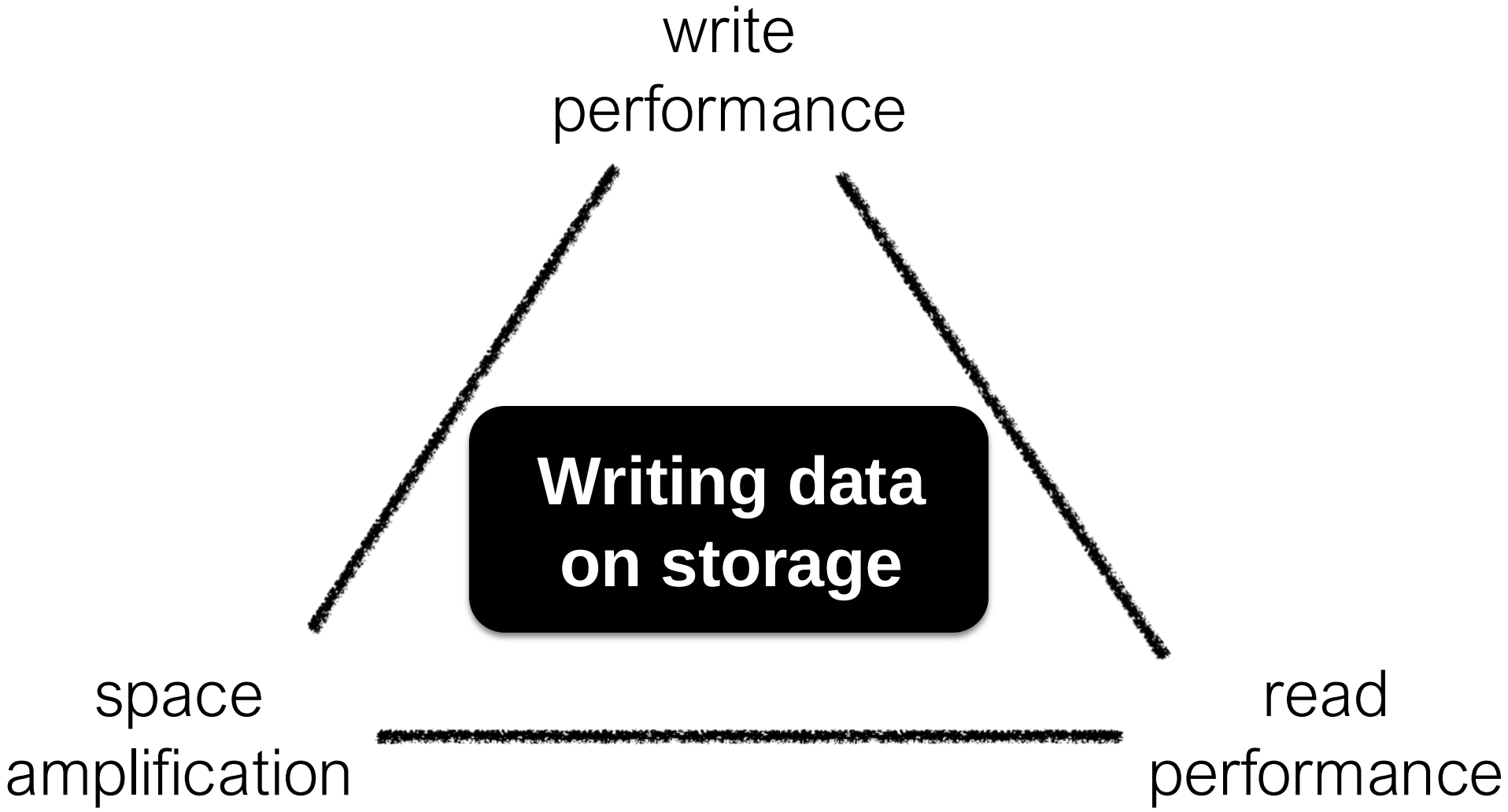
if $T = 10$ & $L = 4$

99.9% on storage

How does the storage layer affect ingestion?

BOSTON UNIVERSITY

Storage Optimizations



Data **Layout**

Classical LSM design: `leveling`
[eager merging]

Data **Layout**

leveling [eager]

Data **Layout**

leveling [eager]

Data **Layout**

leveling [eager]

Data **Layout**

leveling [eager]

Data **Layout**

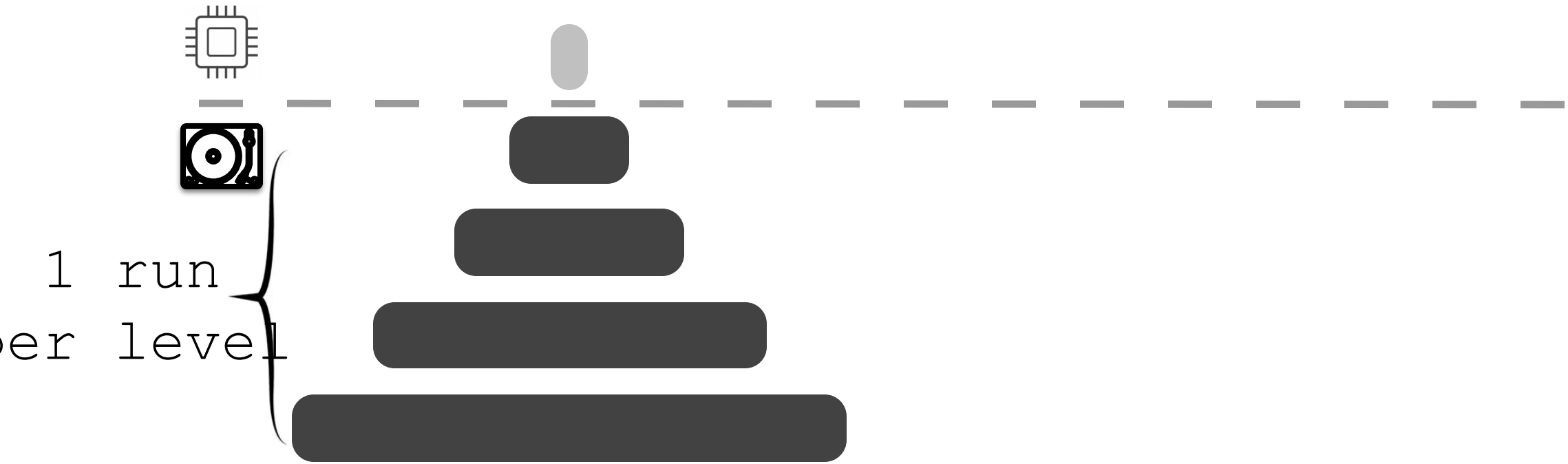
leveling [eager]

Data **Layout**

leveling [eager]

Data **Layout**

leveling [eager]

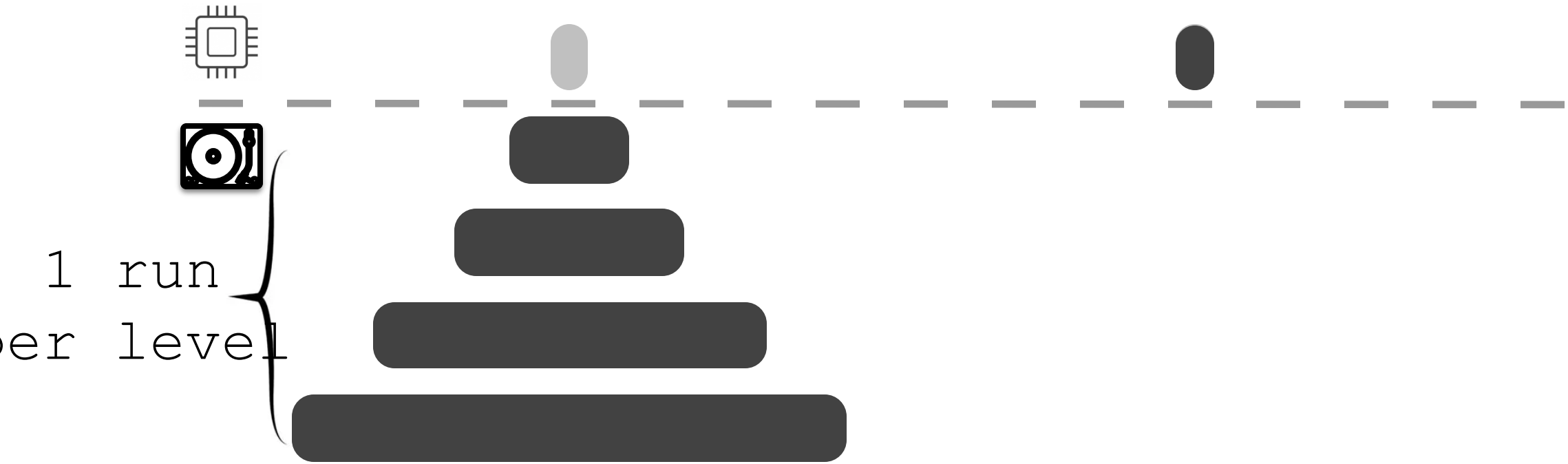


- good read performance
- good space amplification
- high write amplification

Data **Layout**

leveling [eager]

tiering [lazy]



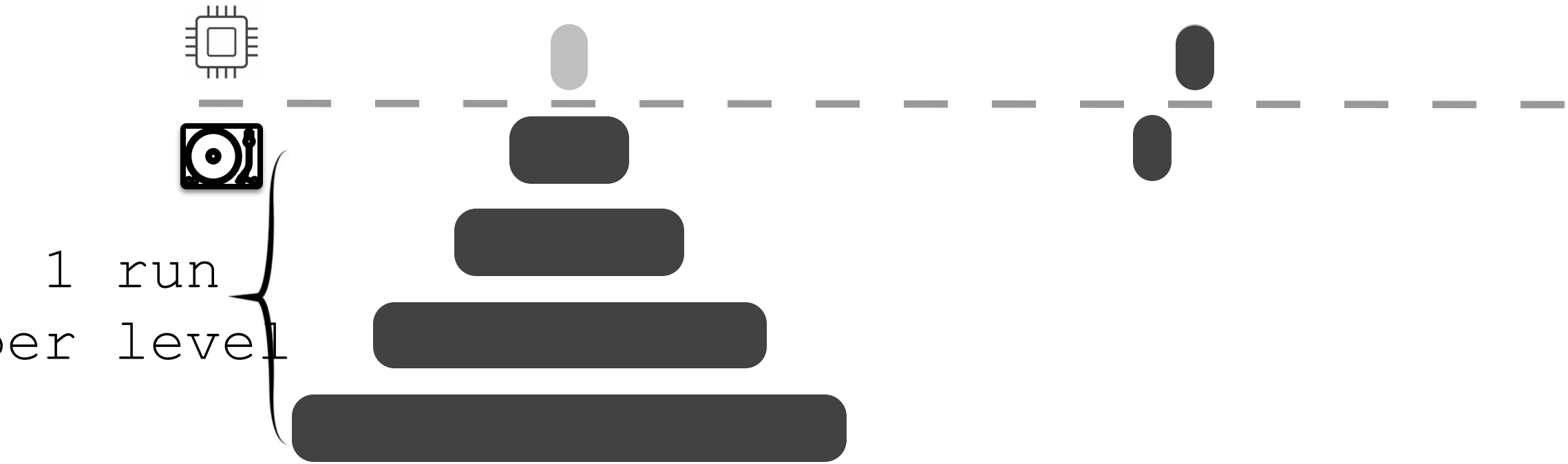
- good read performance
- good space amplification
- high write amplification

BOSTON UNIVERSITY

Data **Layout**

leveling [eager]

tiering [lazy]



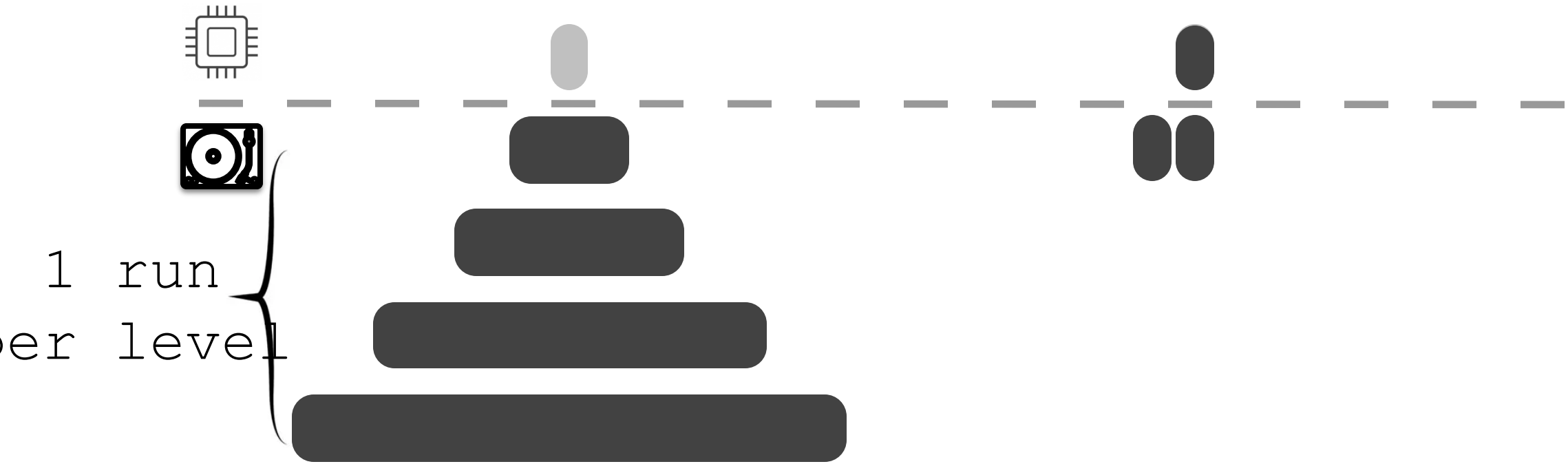
- good read performance
- good space amplification
- high write amplification

BOSTON UNIVERSITY

Data **Layout**

leveling [eager]

tiering [lazy]



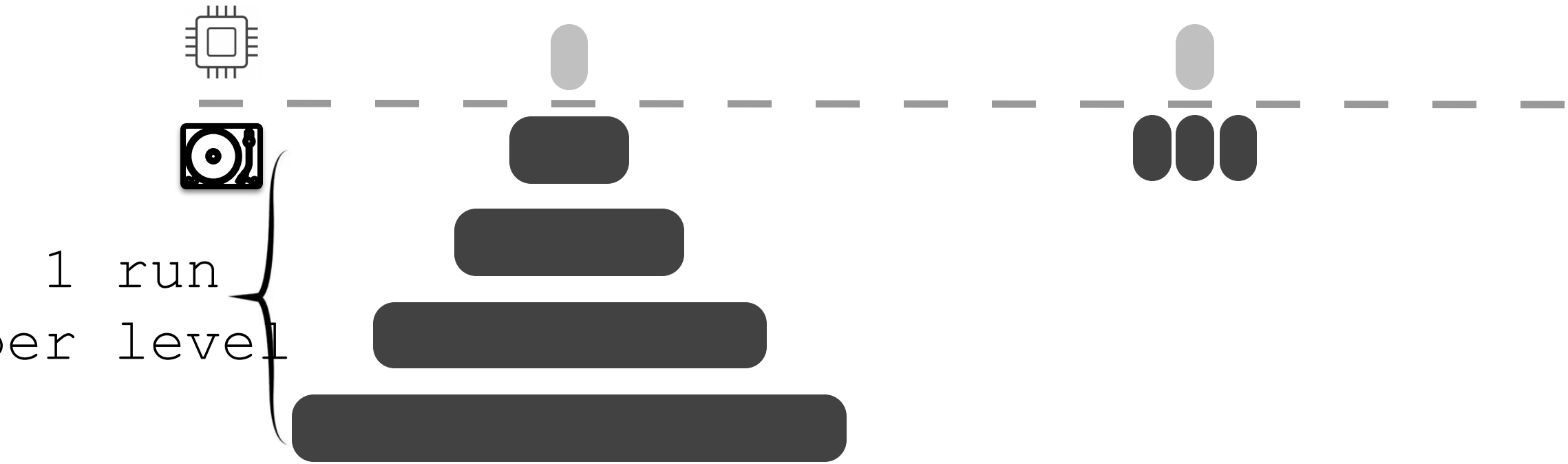
- good read performance
- good space amplification
- high write amplification

BOSTON UNIVERSITY

Data **Layout**

leveling [eager]

tiering [lazy]

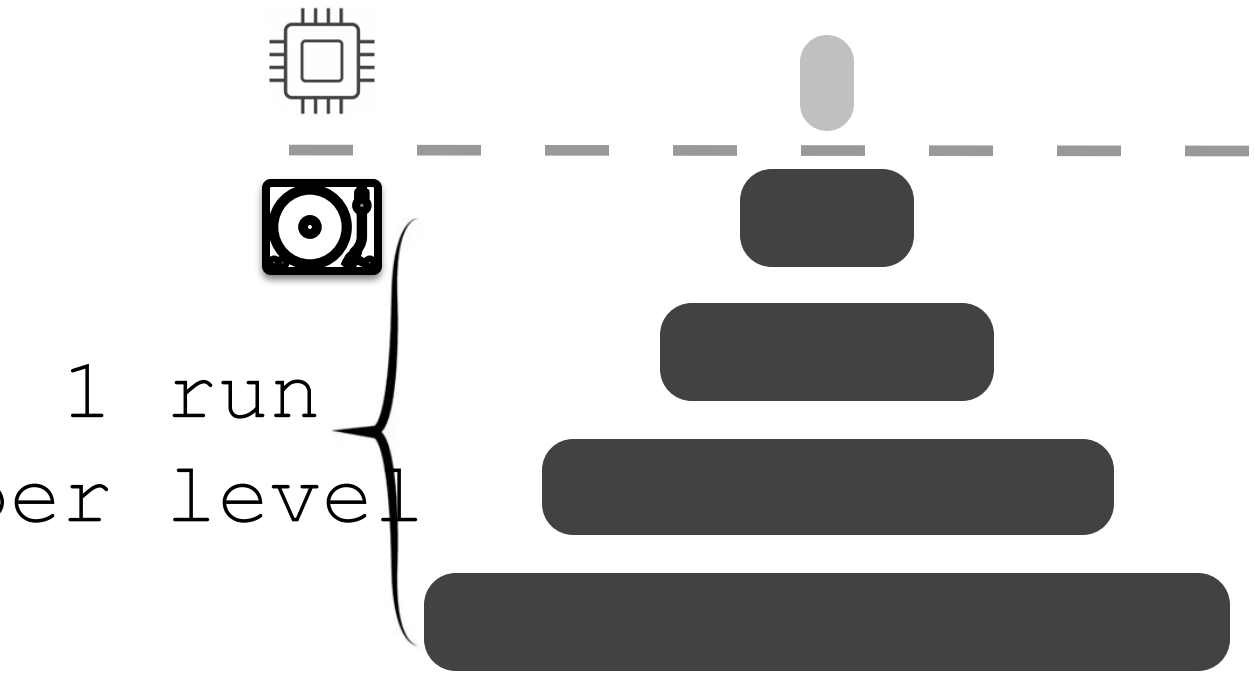


- good read performance
- good space amplification
- high write amplification

BOSTON UNIVERSITY

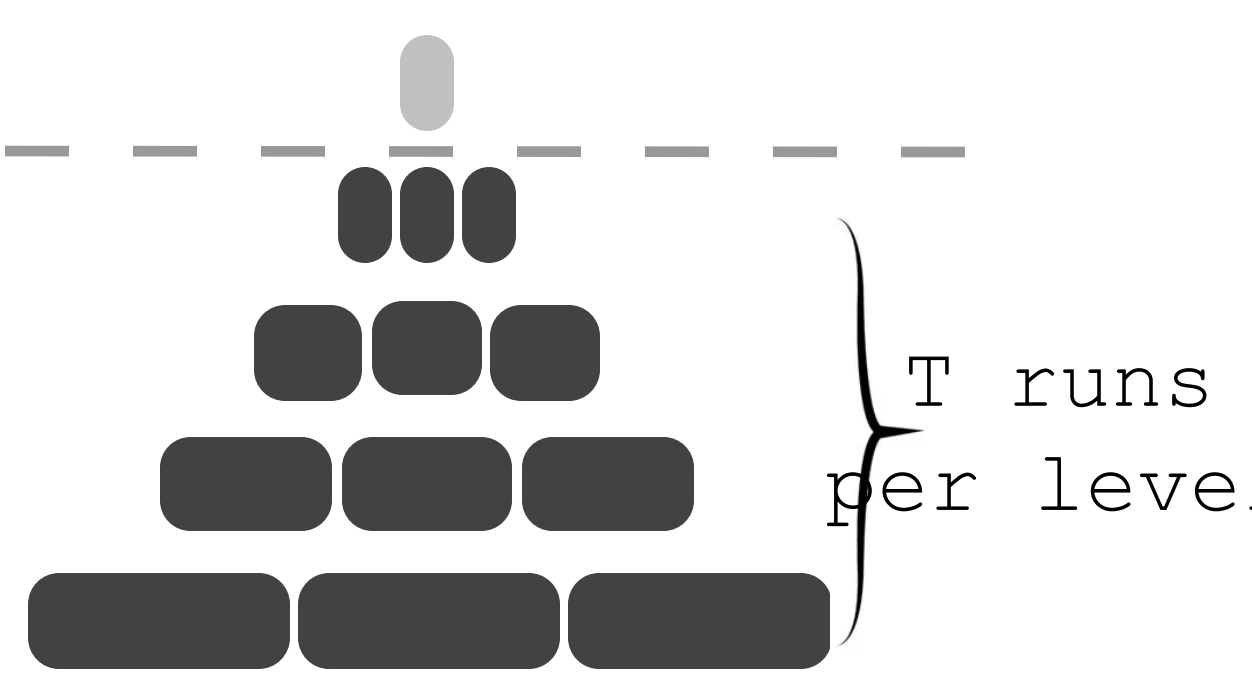
Data **Layout**

leveling [eager]



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

tiering [lazy]

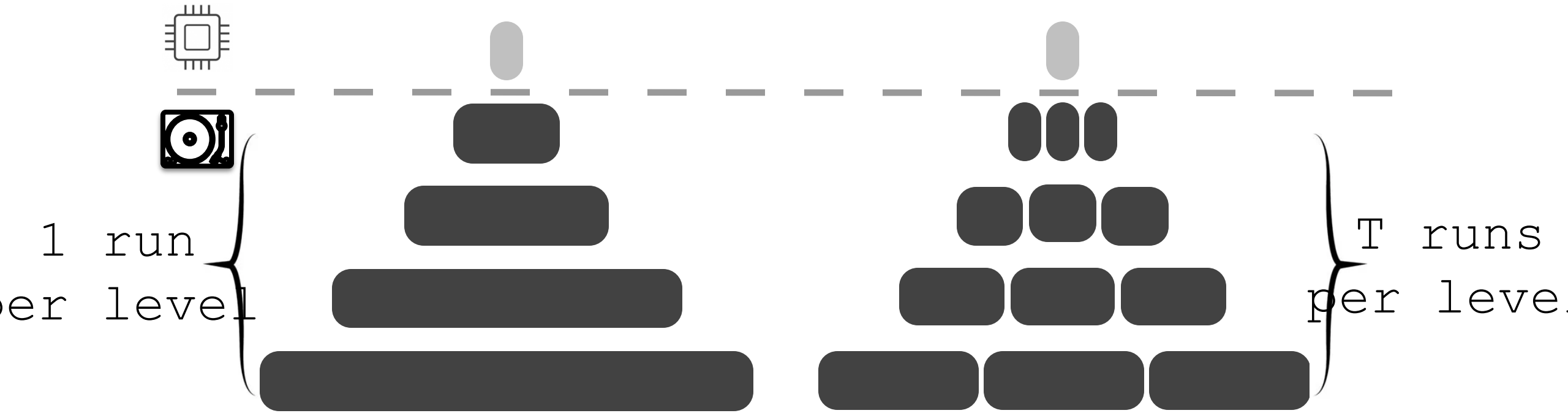


- poor read performance
- poor space amplification
- good ingestion performance

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

Data **Layout**

leveling [eager] | **tiering** [lazy]



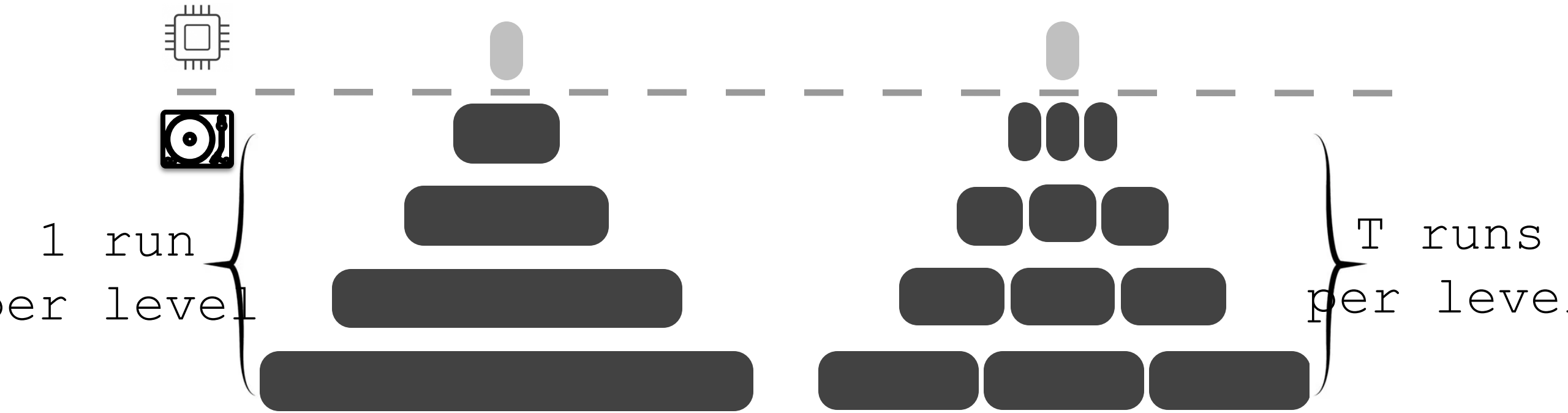
	leveling [eager]	**tiering** [lazy]
Read cost:	$\mathcal{O}(L \cdot \phi)$	$\mathcal{O}(T \cdot L \cdot \phi)$
Write cost:	$\mathcal{O}(T \cdot L/B)$	$\mathcal{O}(L/B)$
SA:	$\mathcal{O}(1/T)$	$\mathcal{O}(T)$

BOSTON UNIVERSITY

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

Data **Layout**

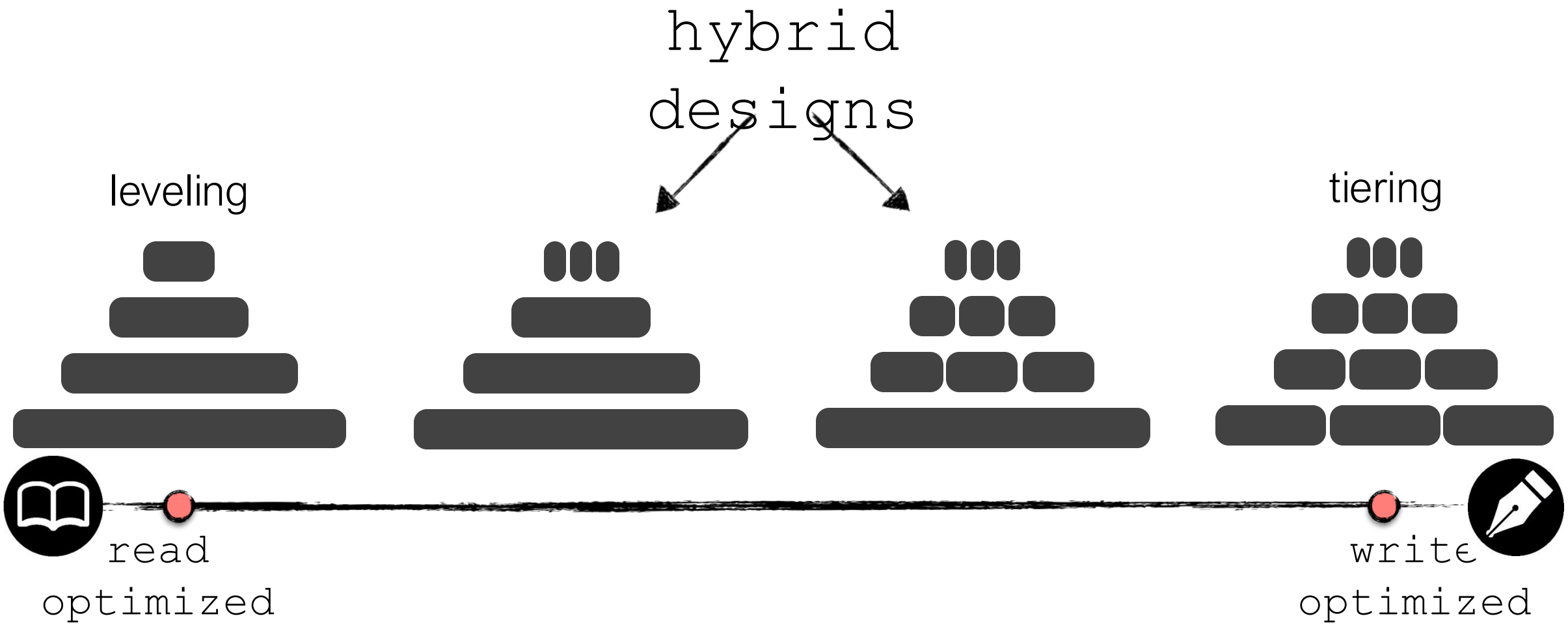
leveling [eager] | **tiering** [lazy]



	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:	$\mathcal{O}(1/T)$	$\mathcal{O}(T)$

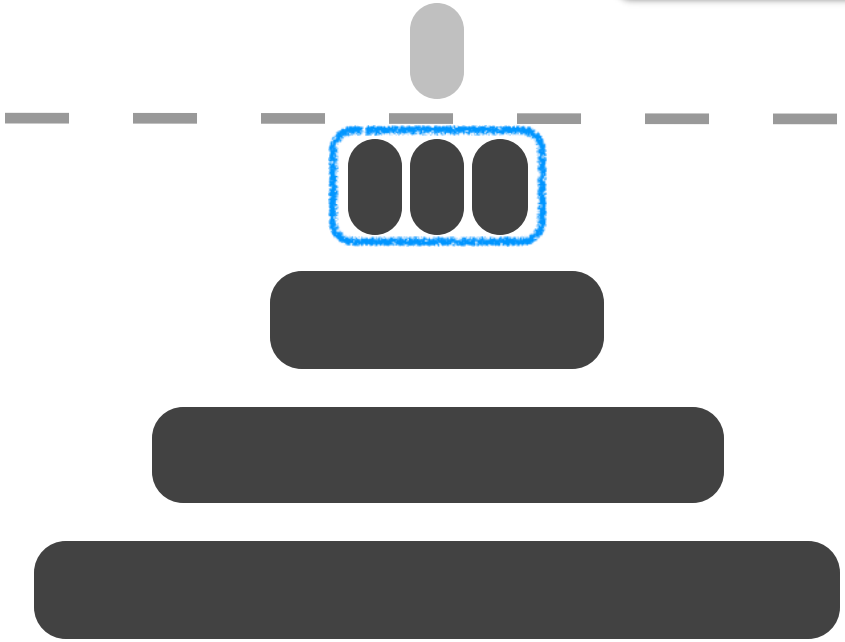
BOSTON UNIVERSITY

Data **Layout**



Data **Layout**

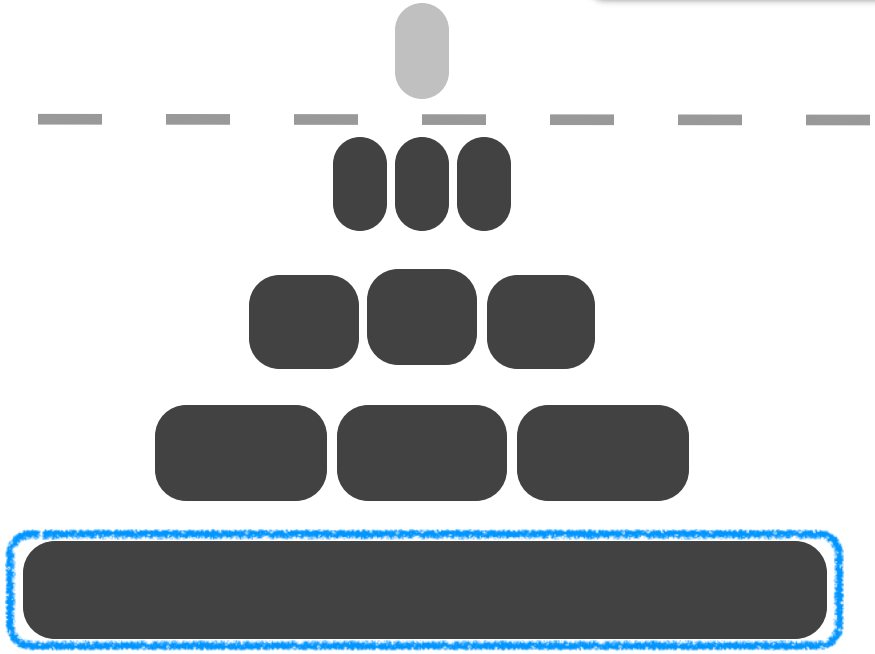
1-leveling RocksDB20



- fewer write stalls
- increased block cache hits

compared to leveling!

L-leveling DayanSIGMOD18

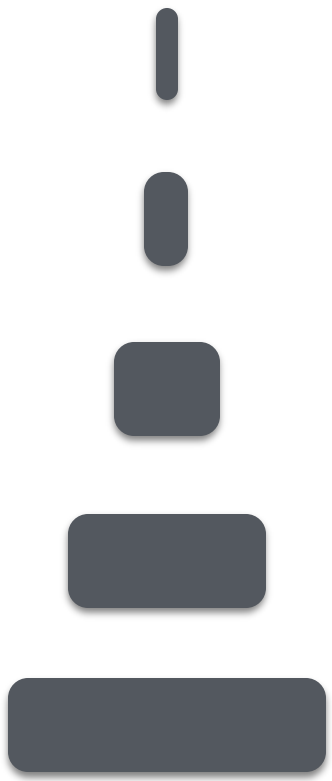


- low write amplification
- better read performance

compared to tiering!

BOSTON UNIVERSITY

Bloom filters



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

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

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

$$\phi_4 = \phi_0/T$$

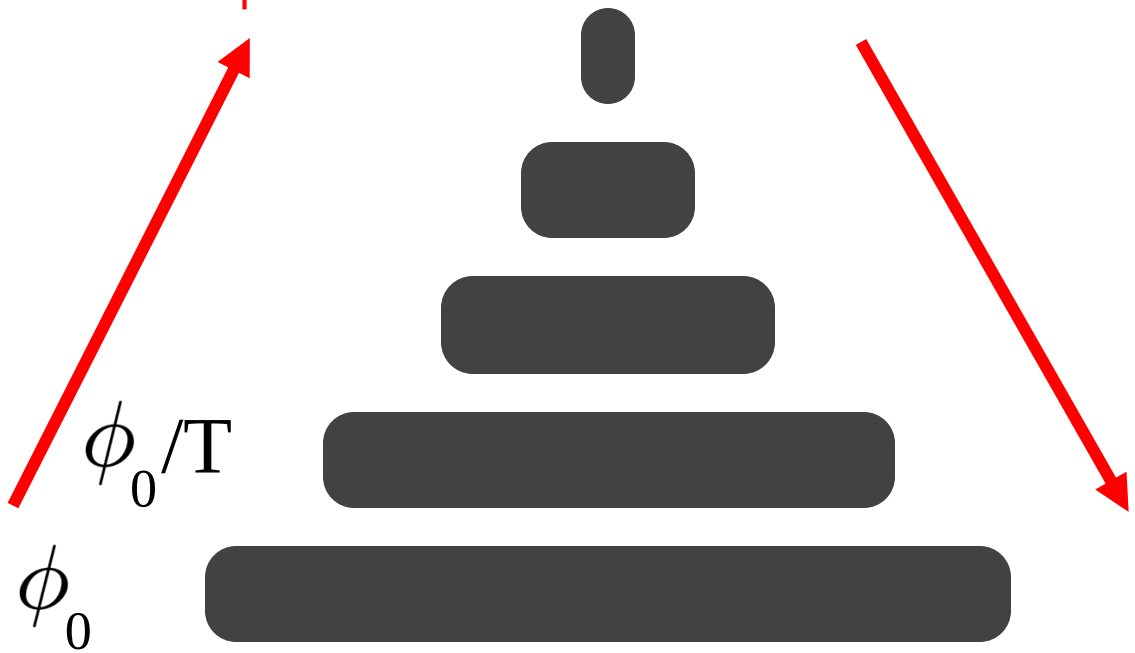
$$\phi_5 = \phi_0$$

exponentially decreasing

LSM-tree

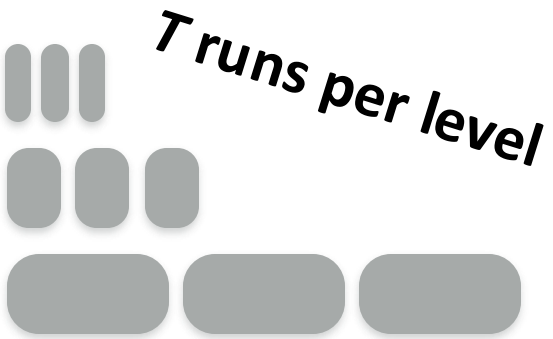
exponentially fewer I/Os in queries

however, every run participates in T merges

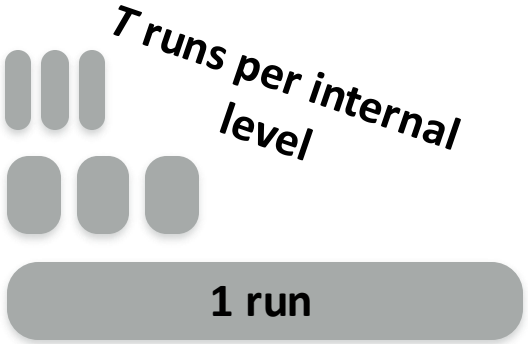


Insight: merging in top levels can become lazy (tiering)!

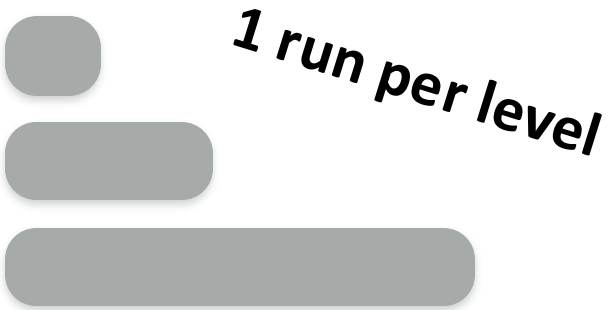
Tiering
write-optimized



Lazy Leveling
hybrid



Leveling
read-optimized



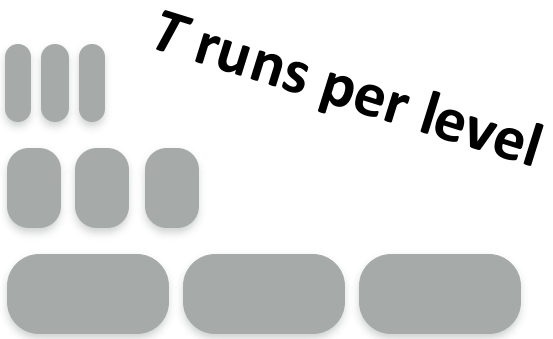
lookup: $O\left(T \cdot log_T(N) \cdot e^{-M/N}\right)$

runs per level — levels — false positive rate

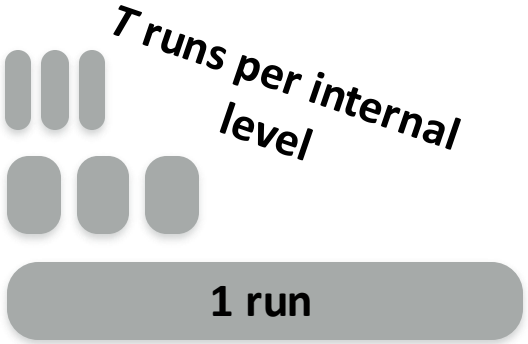
$O\left(log_T(N) \cdot e^{-M/N}\right)$

levels — false positive rate

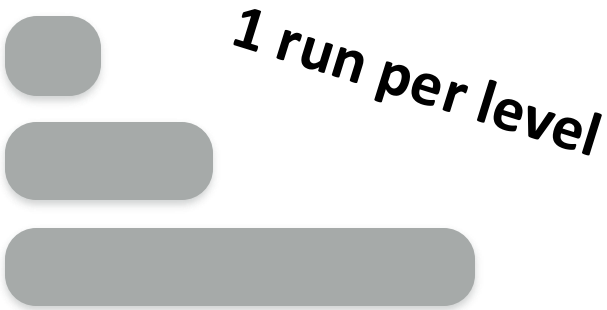
Tiering
write-optimized



Lazy Leveling
hybrid



Leveling
read-optimized



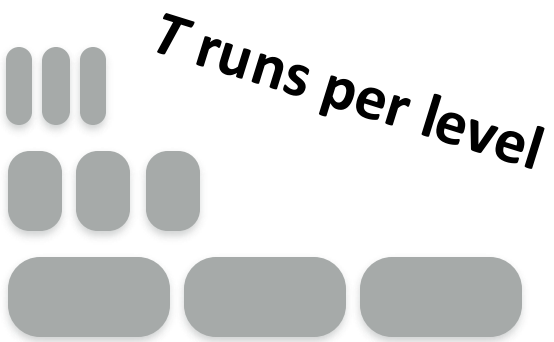
lookup: $O(T \cdot log_T(N) \cdot e^{-M/N})$

runs per level — levels — false positive rate

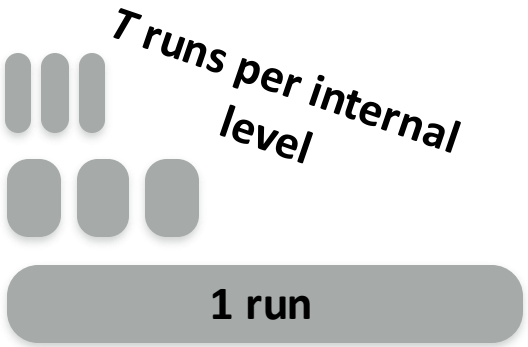
$O(log_T(N) \cdot e^{-M/N})$

levels — false positive rate

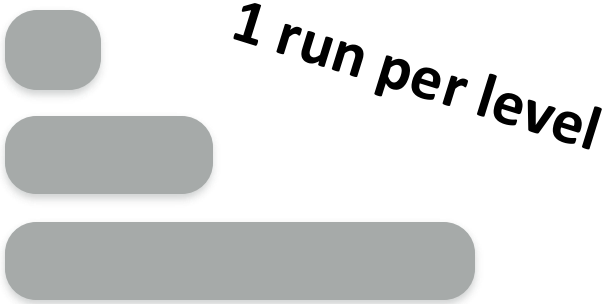
Tiering
write-optimized



Lazy Leveling
hybrid



Leveling
read-optimized



DayanSIGMOD17

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

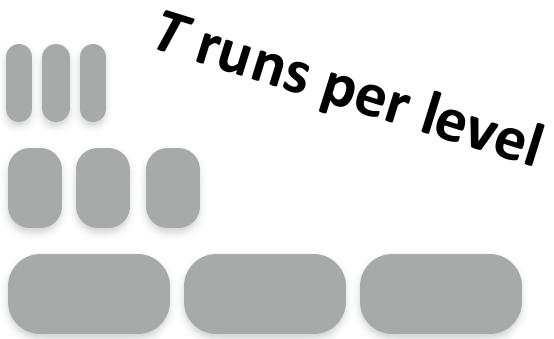
runs per level

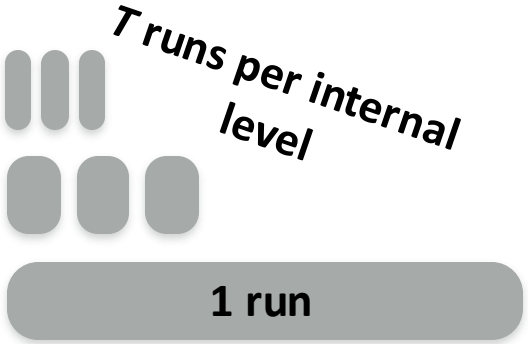
false positive rate

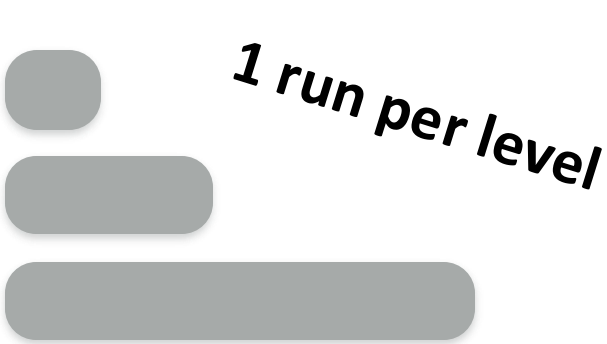
$O(e^{-M/N})$

false positive rate

Tiering write-optimized	Lazy Leveling hybrid	Leveling read-optimized







lookup:

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

runs per level

false positive rate

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

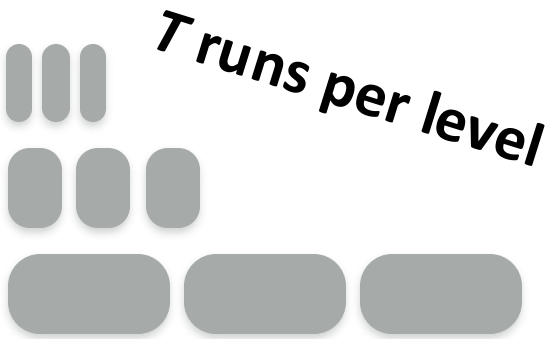
there is a slightly higher constant factor than leveling

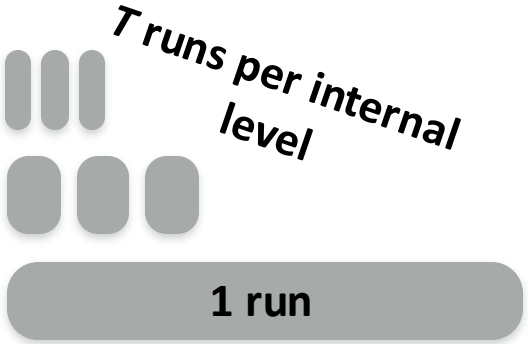
$$O\left(e^{-M/N}\right)$$

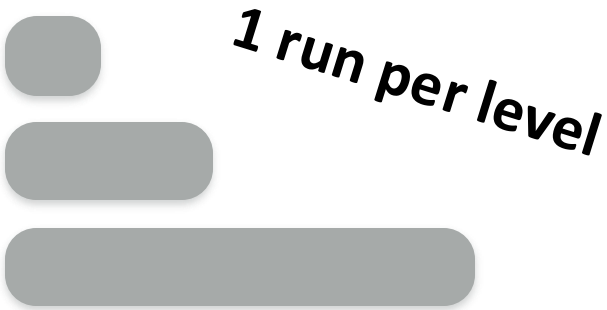
false positive rate

	Tiering write-optimized	Lazy Leveling hybrid	Leveling read-optimized

DayanSIGMOD18

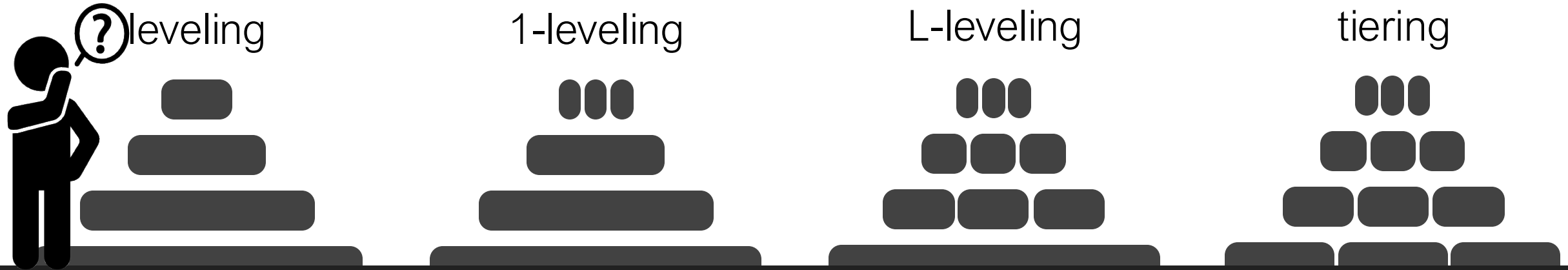




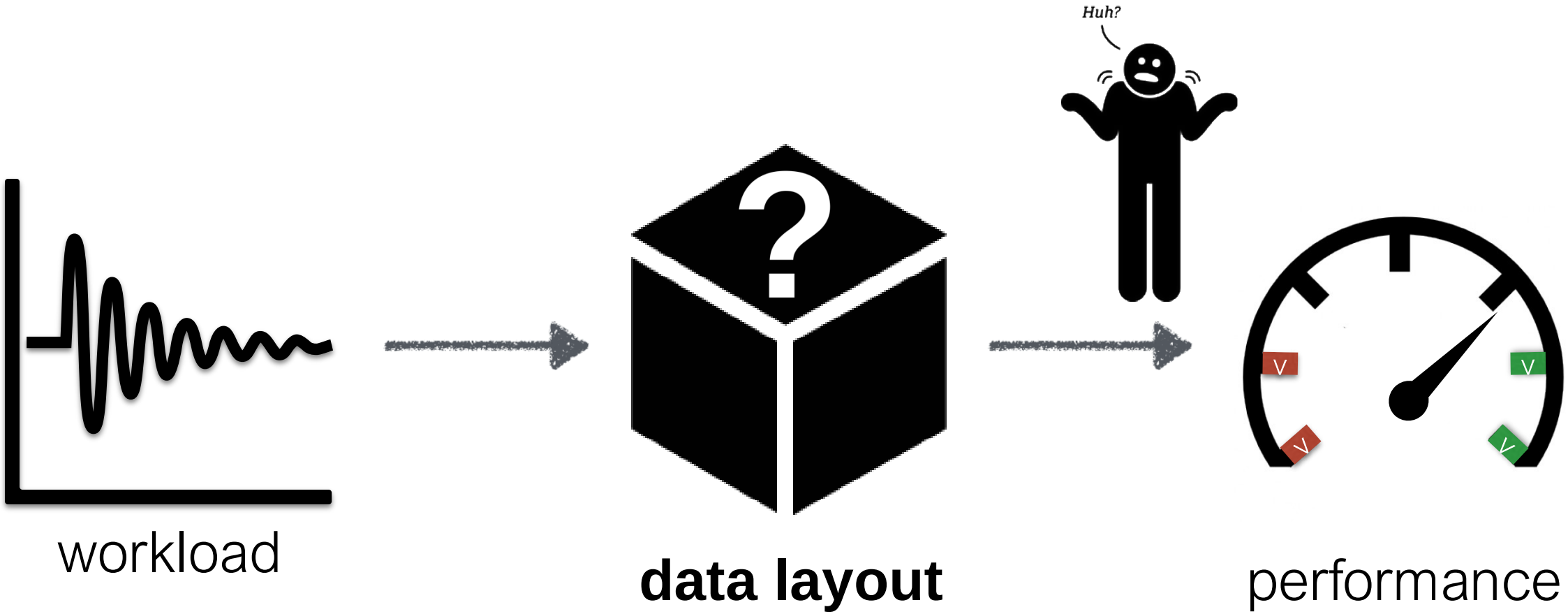


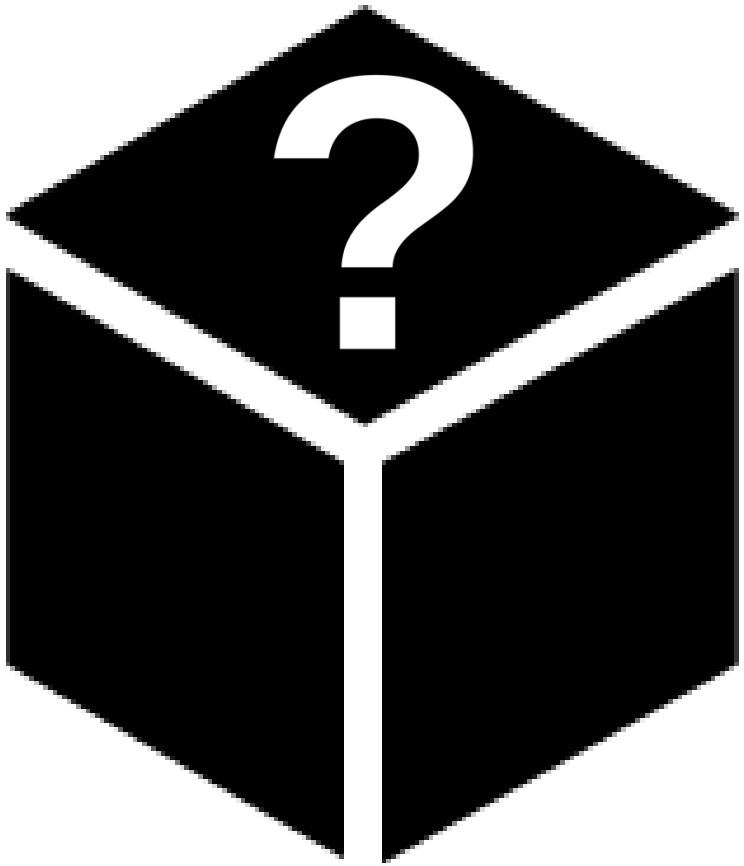
lookup:	$O(T \cdot e^{-M/N})$	$O(e^{-M/N})$	$O(e^{-M/N})$
update cost:	$\boldsymbol{O(log_T(N))}$	$\boldsymbol{O(T + log_T(N))}$	$\boldsymbol{O(T \cdot log_T(N))}$
	levels	T merges for the last level 1 merge for all internal levels	merges per level levels

Data **Layout**



So, how do we reason about the data layout?

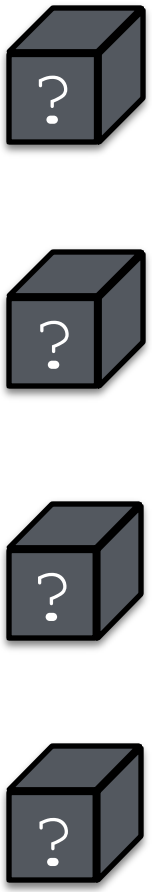

Huh?
workload
data layout
performance



Compaction

black box

BOSTON UNIVERSITY



SarkarVLDB21

1. **How** to organize the data on device?
2. **How much** data to move at-a-time?
3. **Which** block of data to be moved?
4. **When** to re-organize the data layout?

SarkarVLDB21

Data Layout

Compaction Granularity

Data Movement Policy

Compaction Trigger

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

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

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

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

SarkarVLDB21

BOSTON UNIVERSITY

2 Compaction **Granularity**

data moved per compaction

2 Compaction **Granularity**

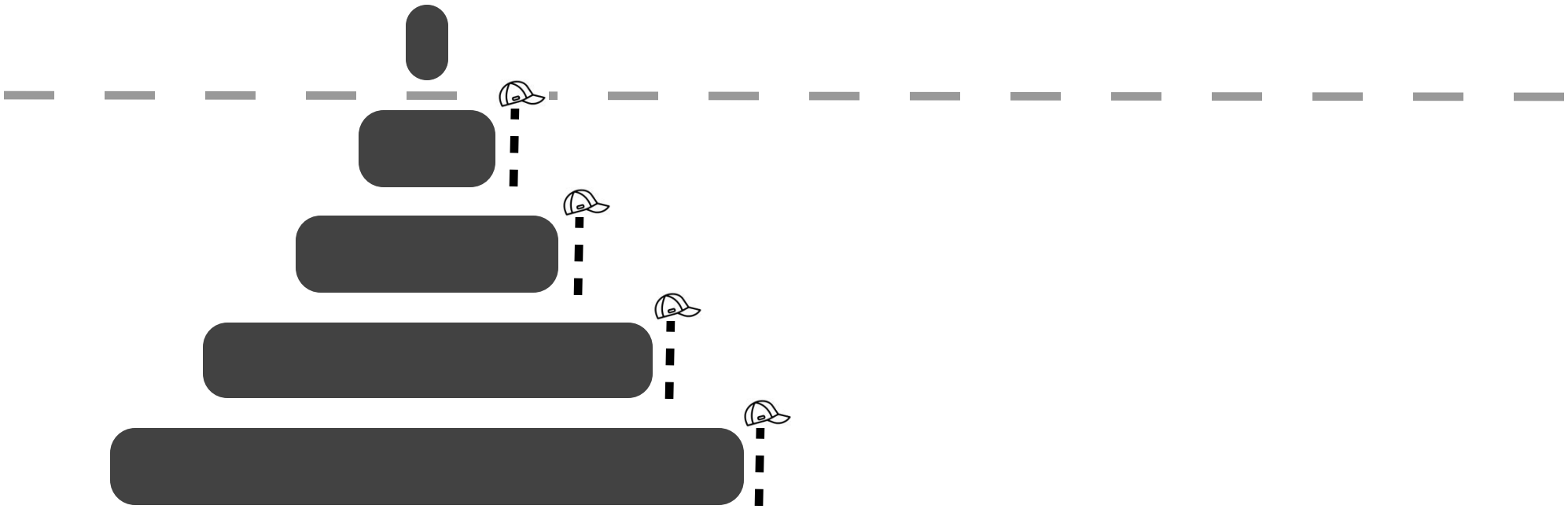
data moved per compaction

consecutiv
e levels

AsterixDB

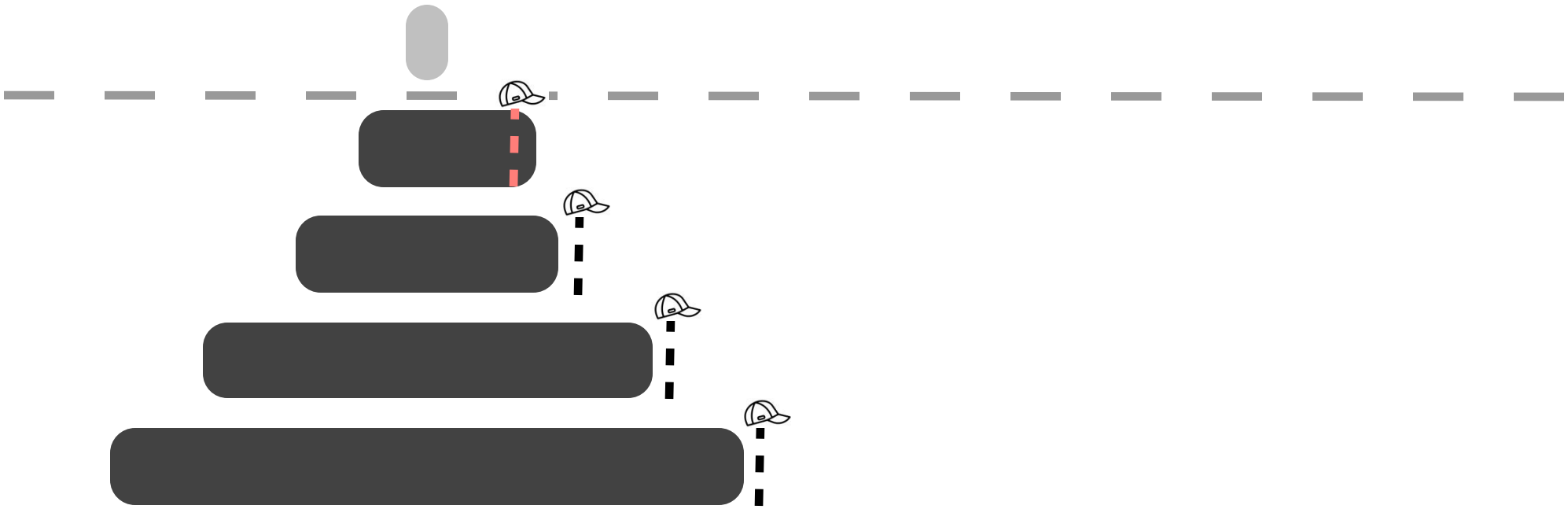
2 Compaction **Granularity**

data moved per compaction



2 Compaction **Granularity**

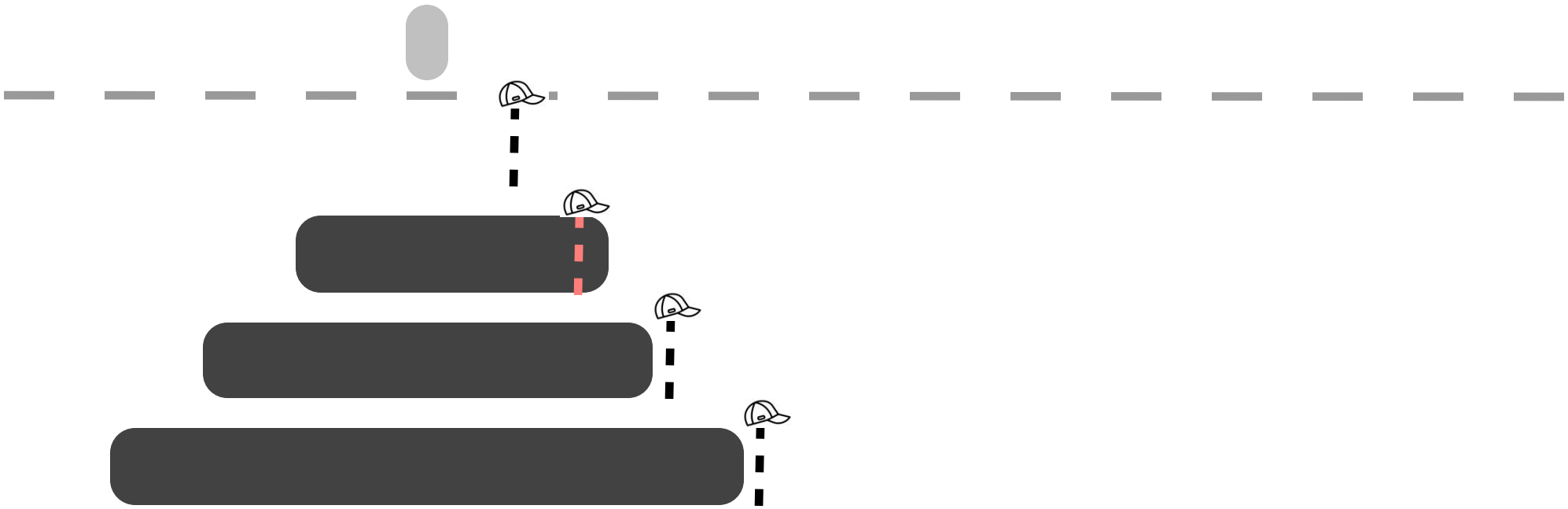
data moved per compaction



BOSTON UNIVERSITY

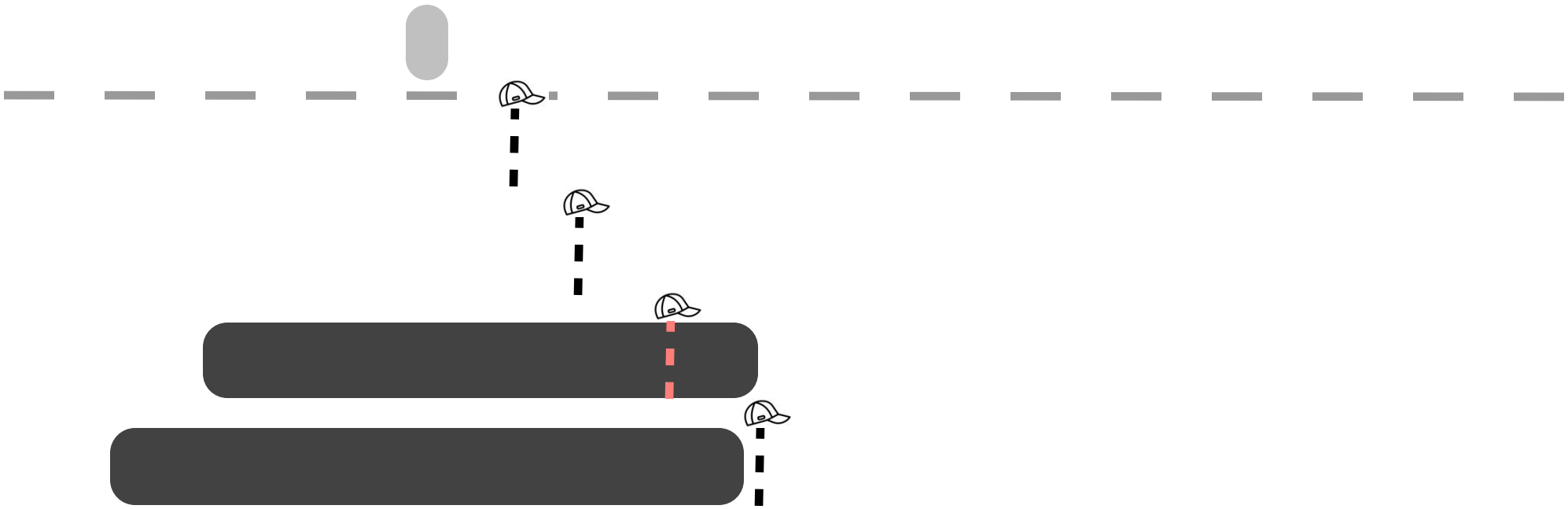
2 Compaction **Granularity**

data moved per compaction



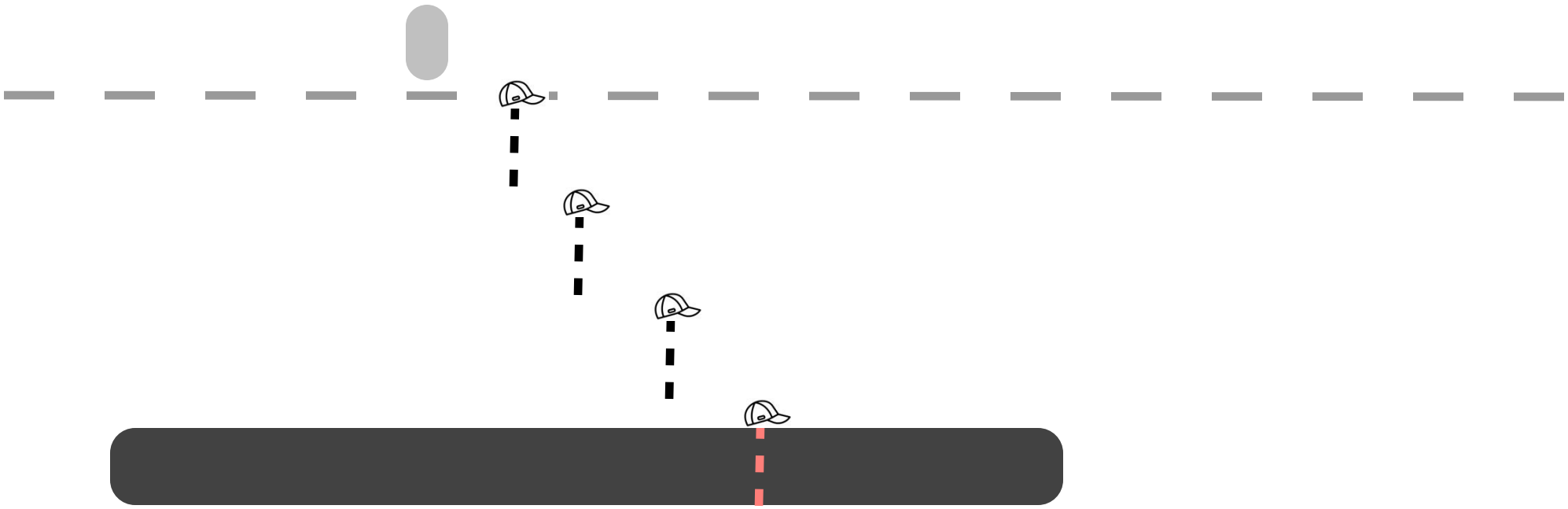
2 Compaction **Granularity**

data moved per compaction



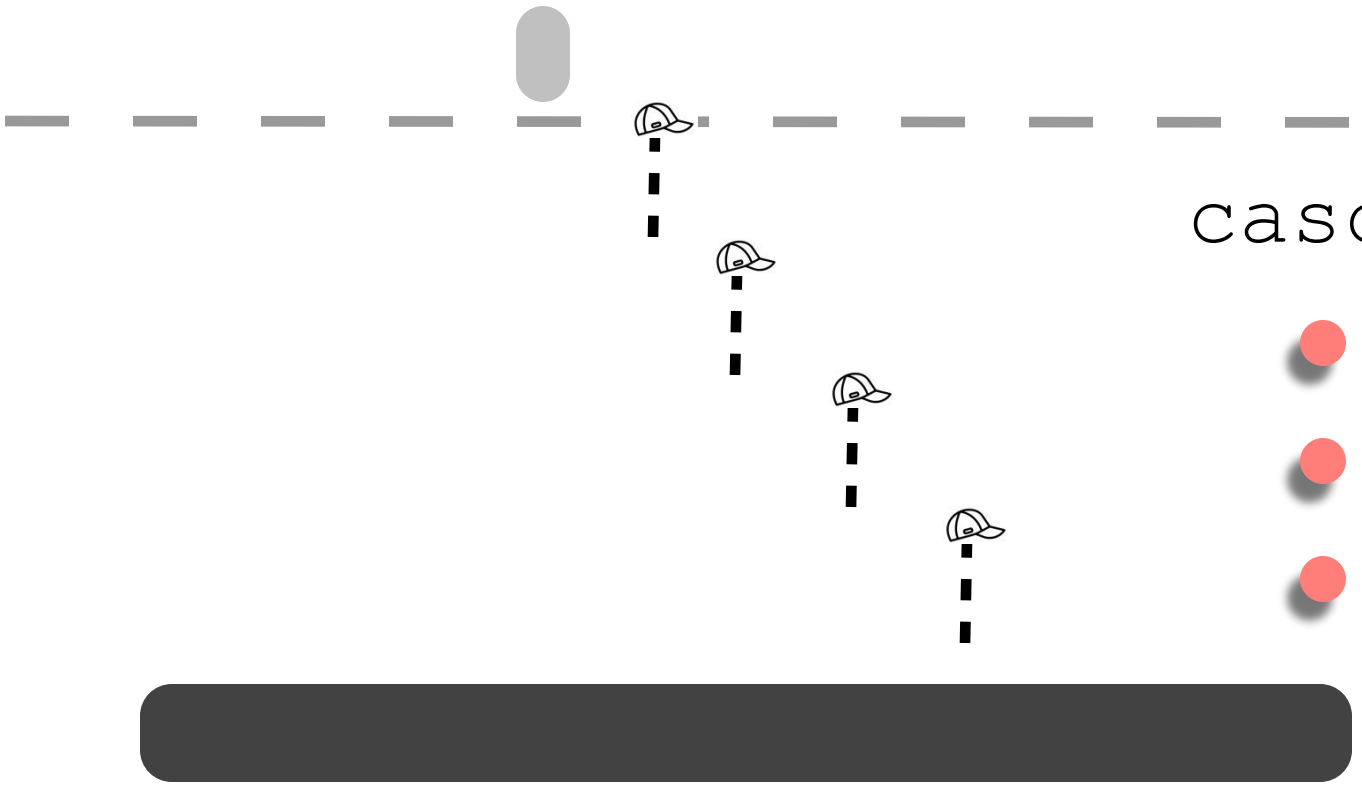
2 Compaction **Granularity**

data moved per compaction



2 Compaction **Granularity**

data moved per compaction



`cascading compactions`

- high latency spikes
- write stalls
- unpredictable perf.

2 Compaction **Granularity**

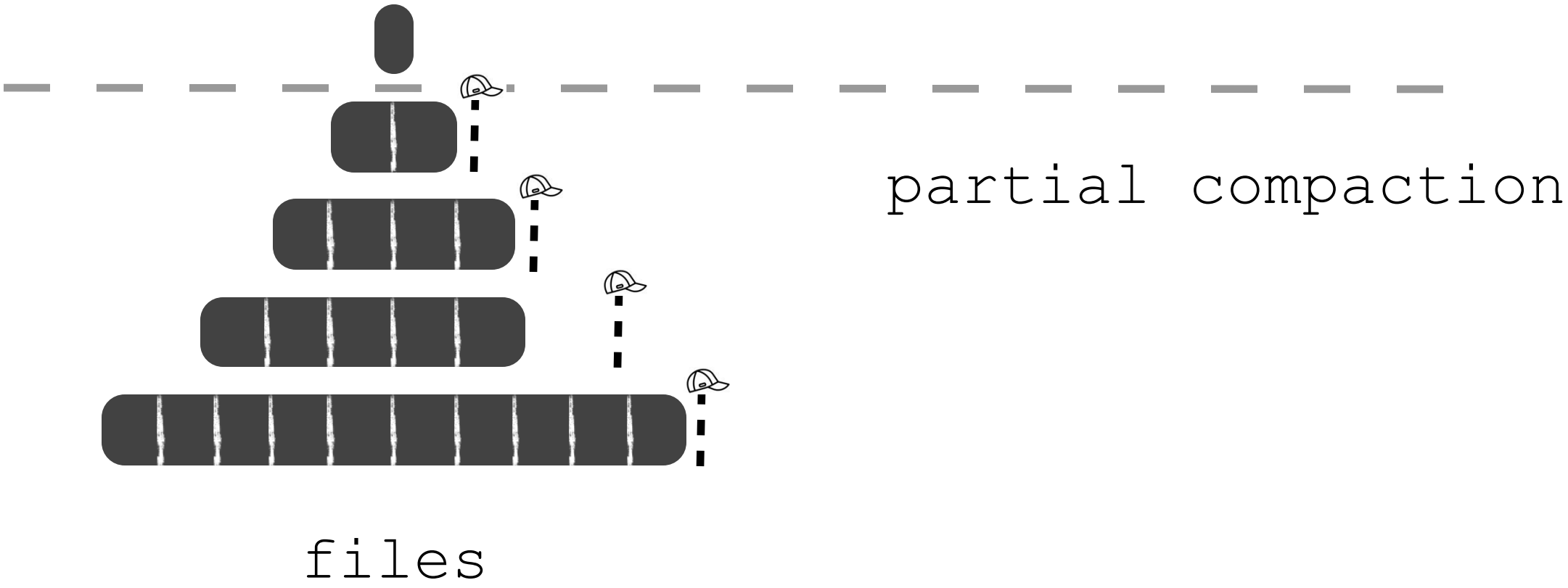
data moved per compaction

`partial compaction`

granularity:`files`

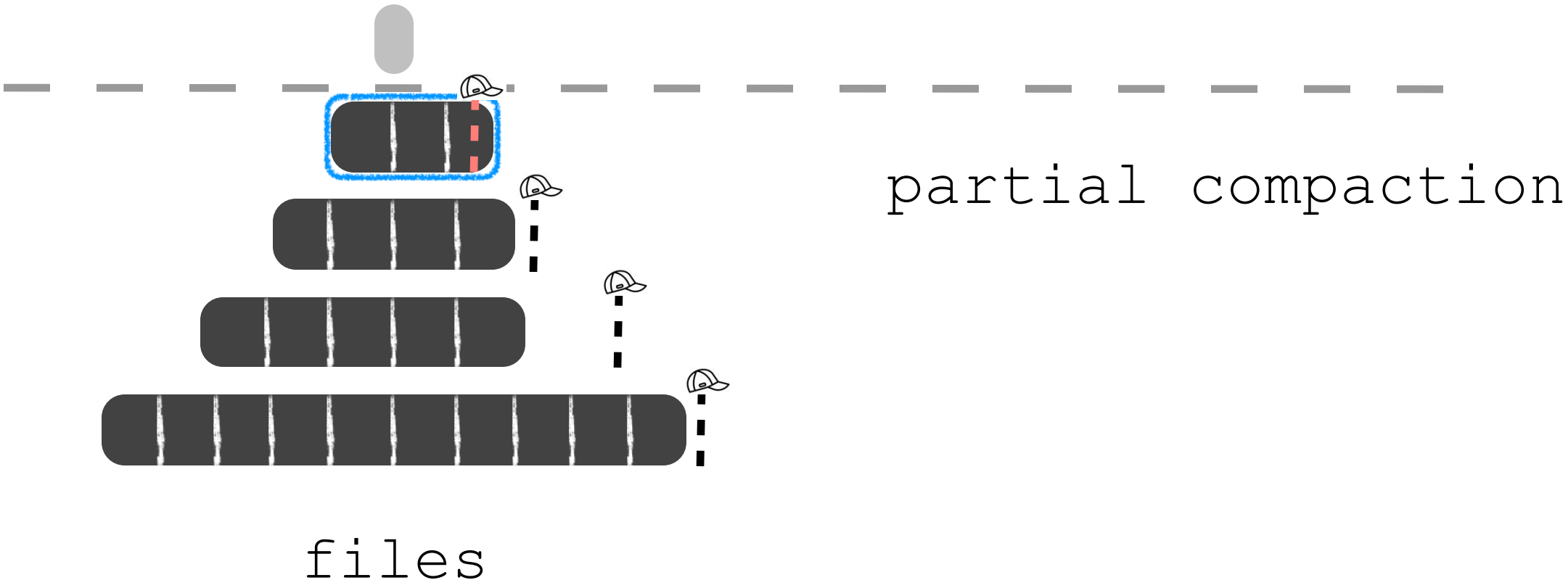
2 Compaction **Granularity**

data moved per compaction



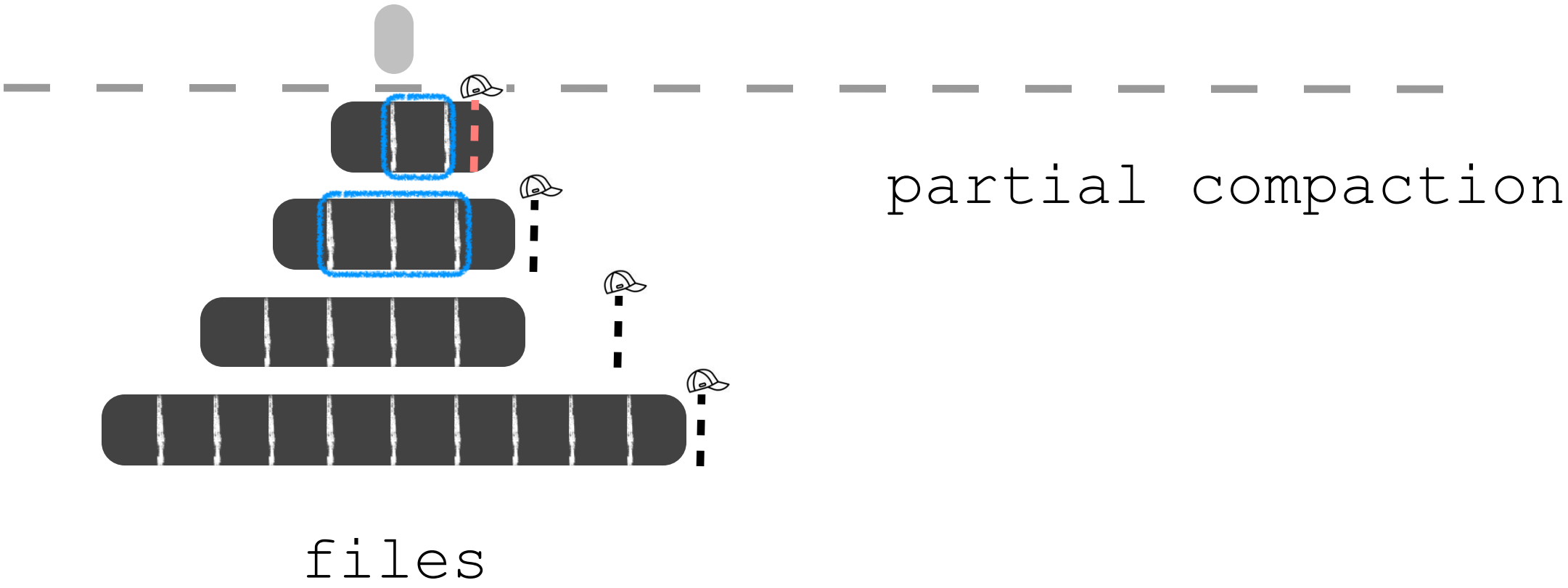
2 Compaction **Granularity**

data moved per compaction



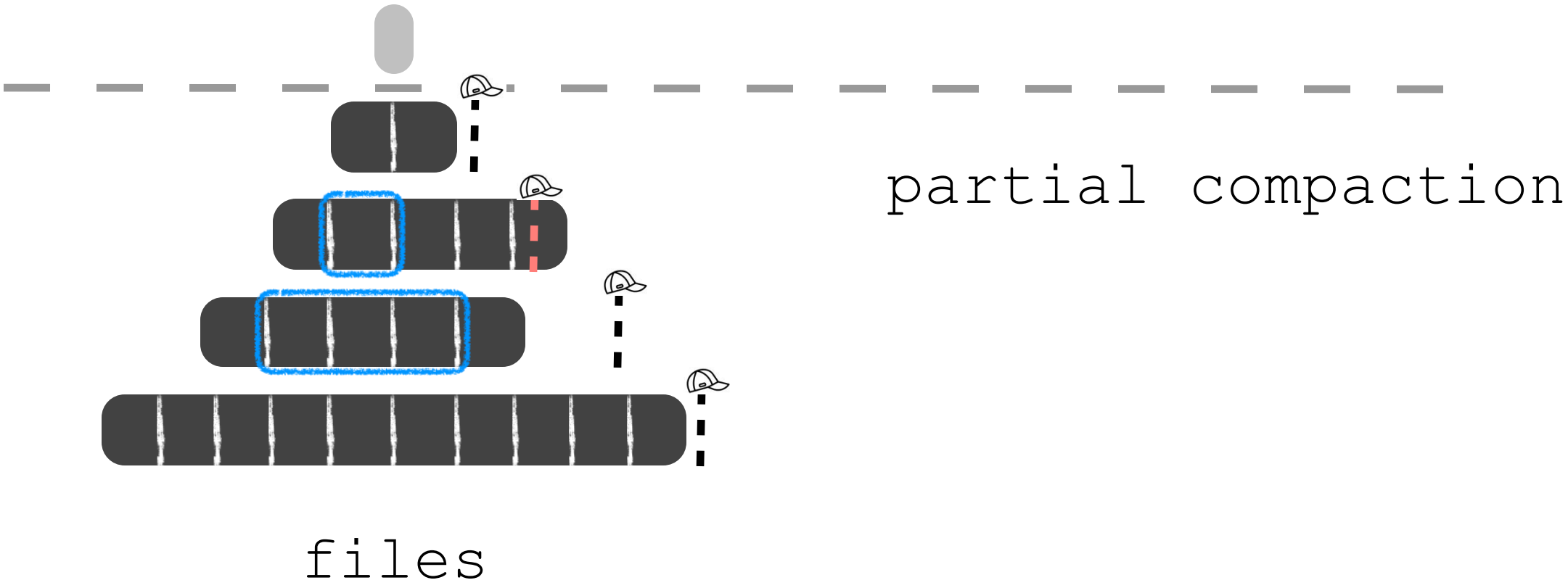
2 Compaction **Granularity**

data moved per compaction



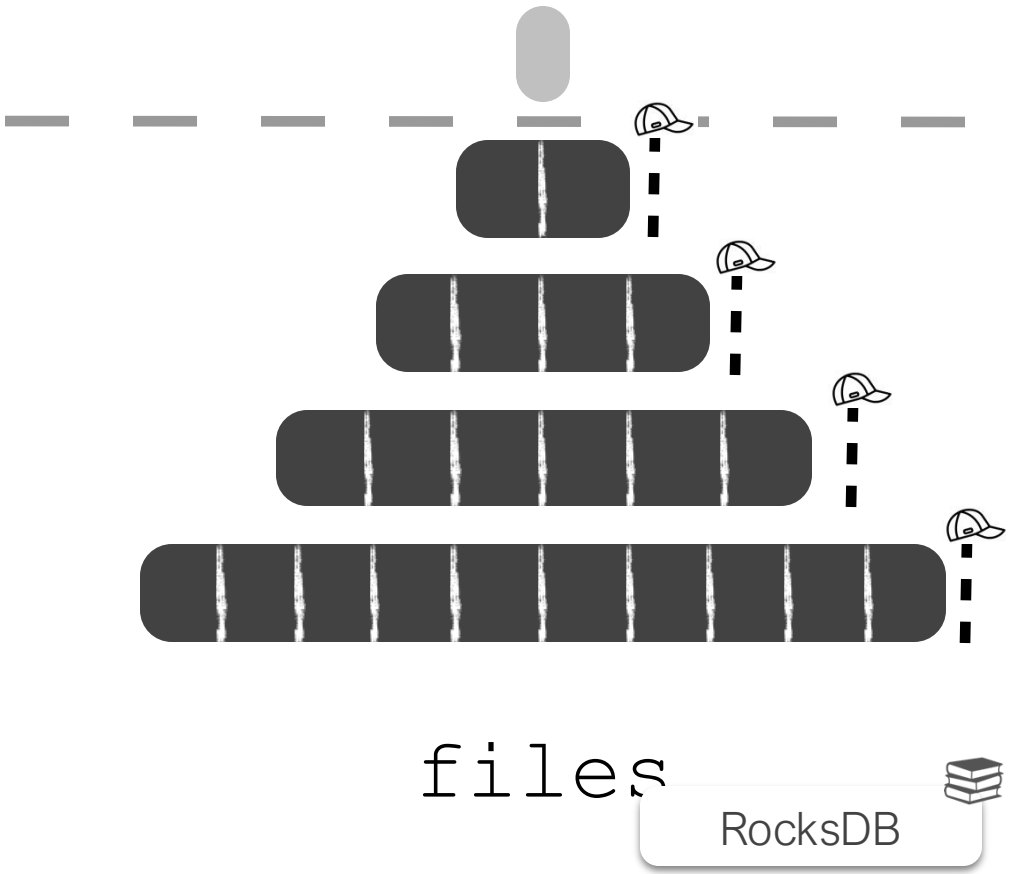
2 Compaction **Granularity**

data moved per compaction



2 Compaction **Granularity**

data moved per compaction



`partial compaction`

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

2 Compaction **Granularity**

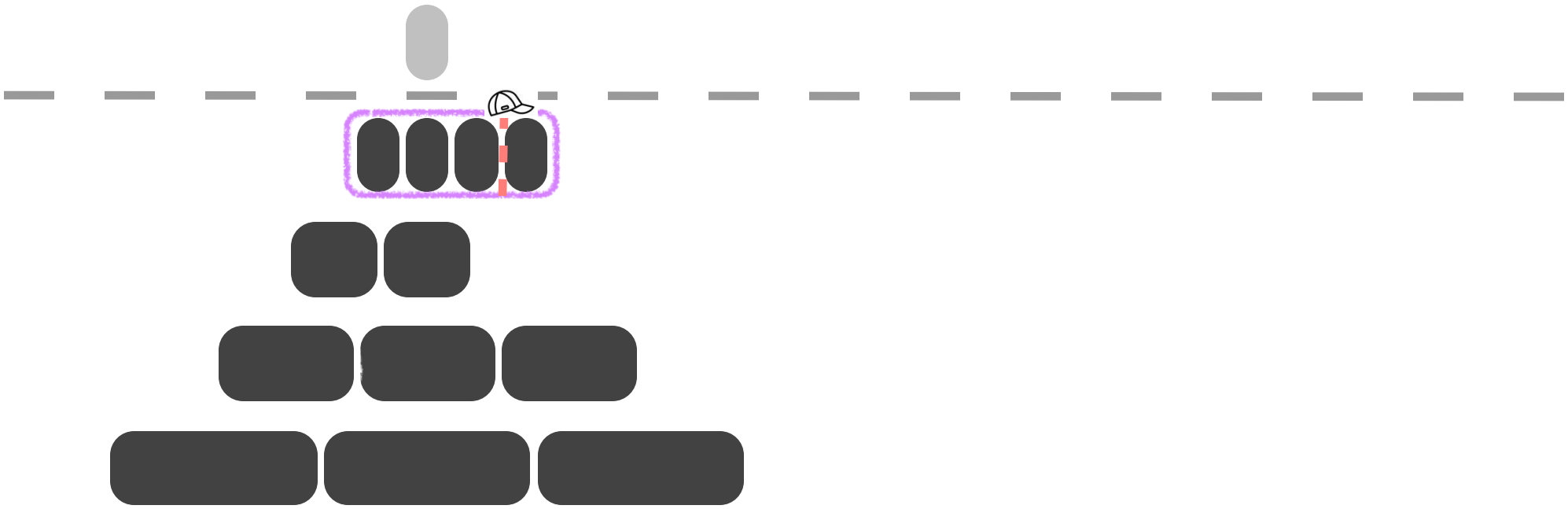
data moved per compaction

sorted runs in a level

BOSTON UNIVERSITY

2 Compaction **Granularity**

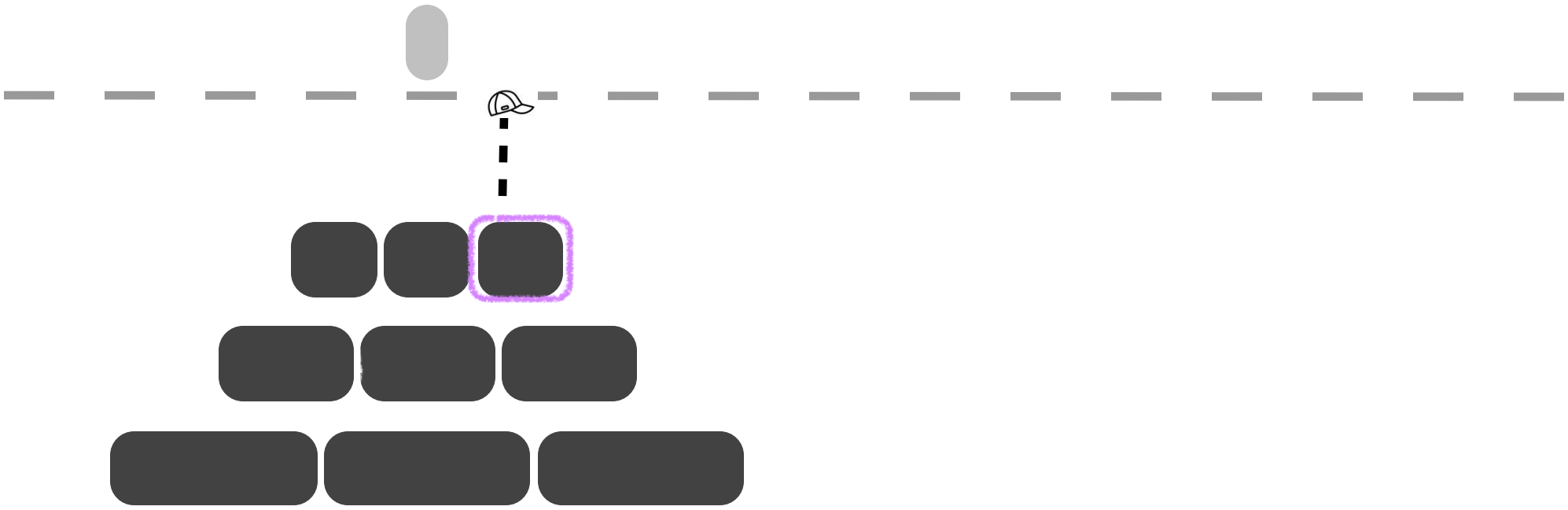
data moved per compaction



sorted runs in a level

2 Compaction **Granularity**

data moved per compaction

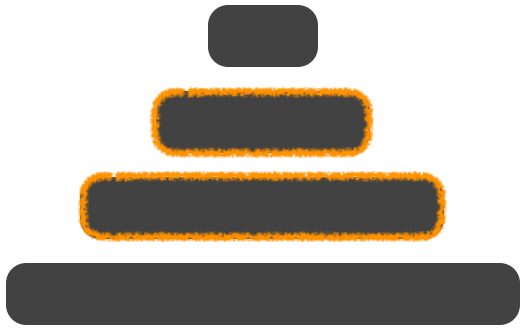


sorted runs in a level

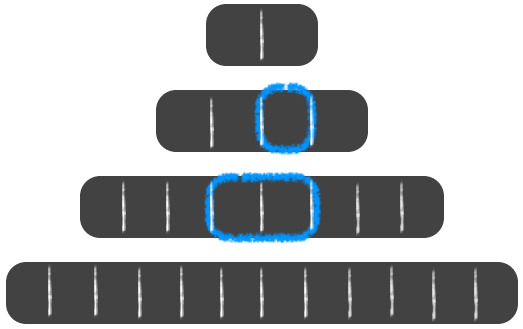
BOSTON UNIVERSITY

2 Compaction **Granularity**

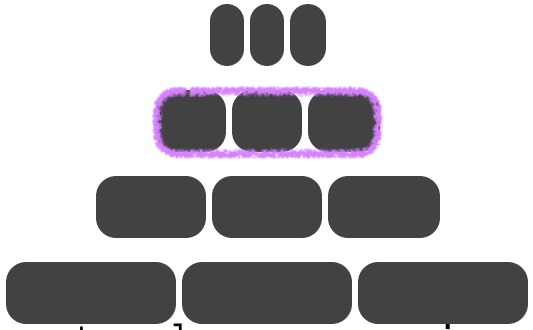
data moved per compaction



levels



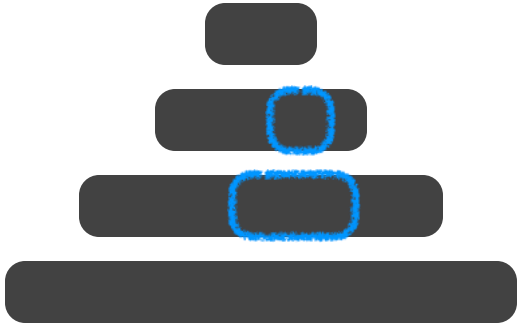
files



sorted runs in a level

3 **Data Movement** Policy

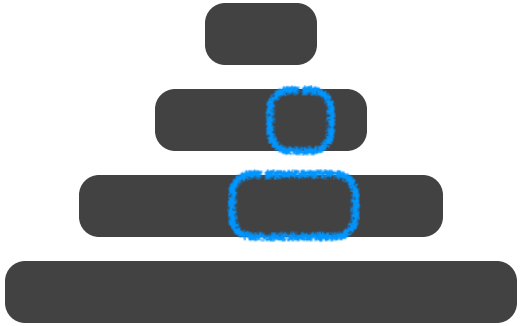
which data to compact



files

Data Movement Policy

which data to compact



files

round-robin !

minimum **overlap with parent** level WA

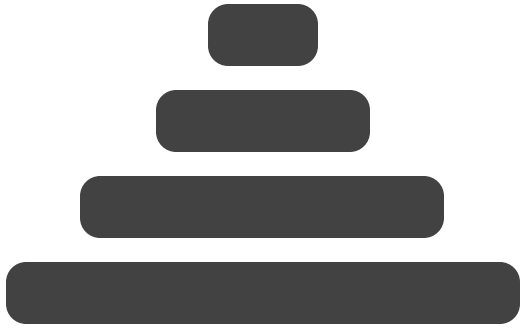
file with most **tombstones** SA

coldest file PQ

4 Compaction **Trigger**

invoking the compaction routine

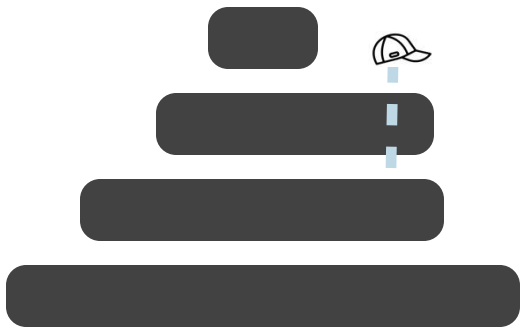
level **saturation**



4 Compaction **Trigger**

invoking the compaction routine

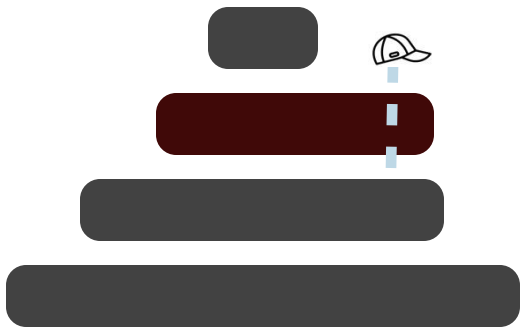
level **saturation**



4 Compaction **Trigger**

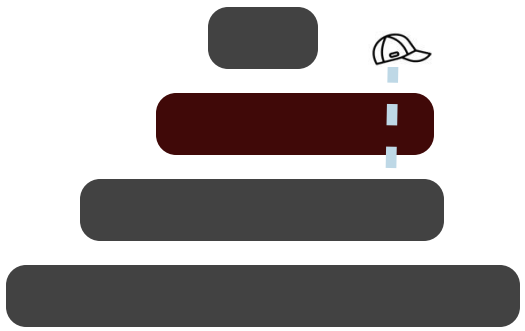
invoking the compaction routine

level **saturation**



4 Compaction **Trigger**

invoking the compaction routine



level **saturation**

number of **sorted runs**

space amplification SA

age of a file De

1 Data Layout

2 Compaction Granularity

3 Data Movement Policy

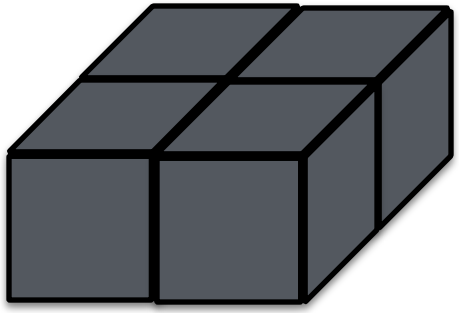
4 Compaction Trigger

Data Layout

Compaction Granularity

Data Movement Policy

Compaction Trigger



***Any* Compaction Algorithm**

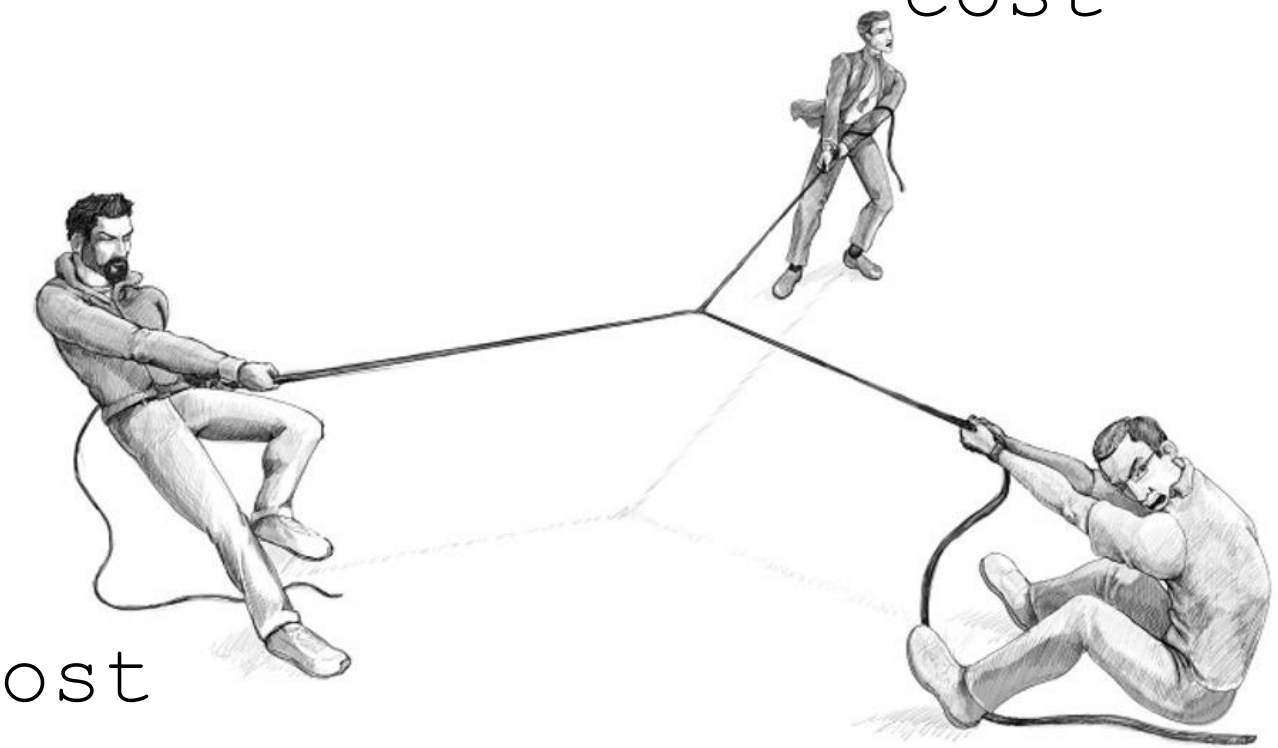
SarkarVLDB21

BOSTON UNIVERSITY

LSM **Design Space**

LSM **Design Space**

Update cost



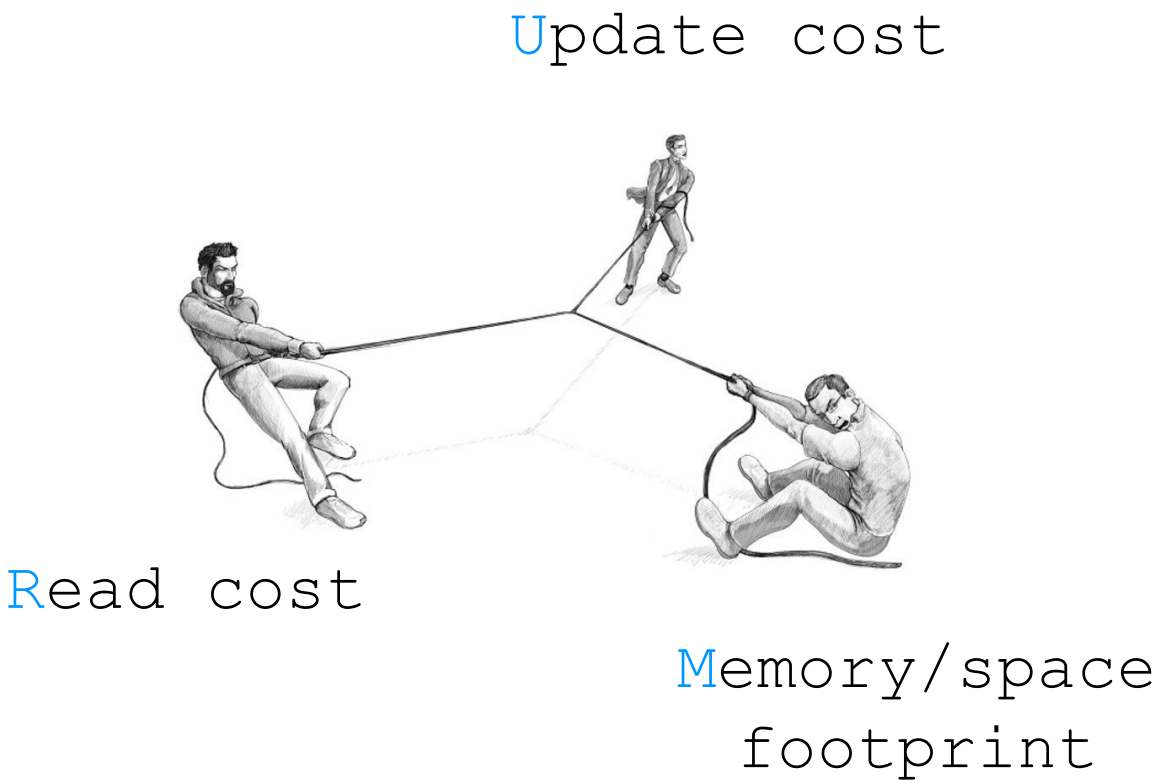
Read cost

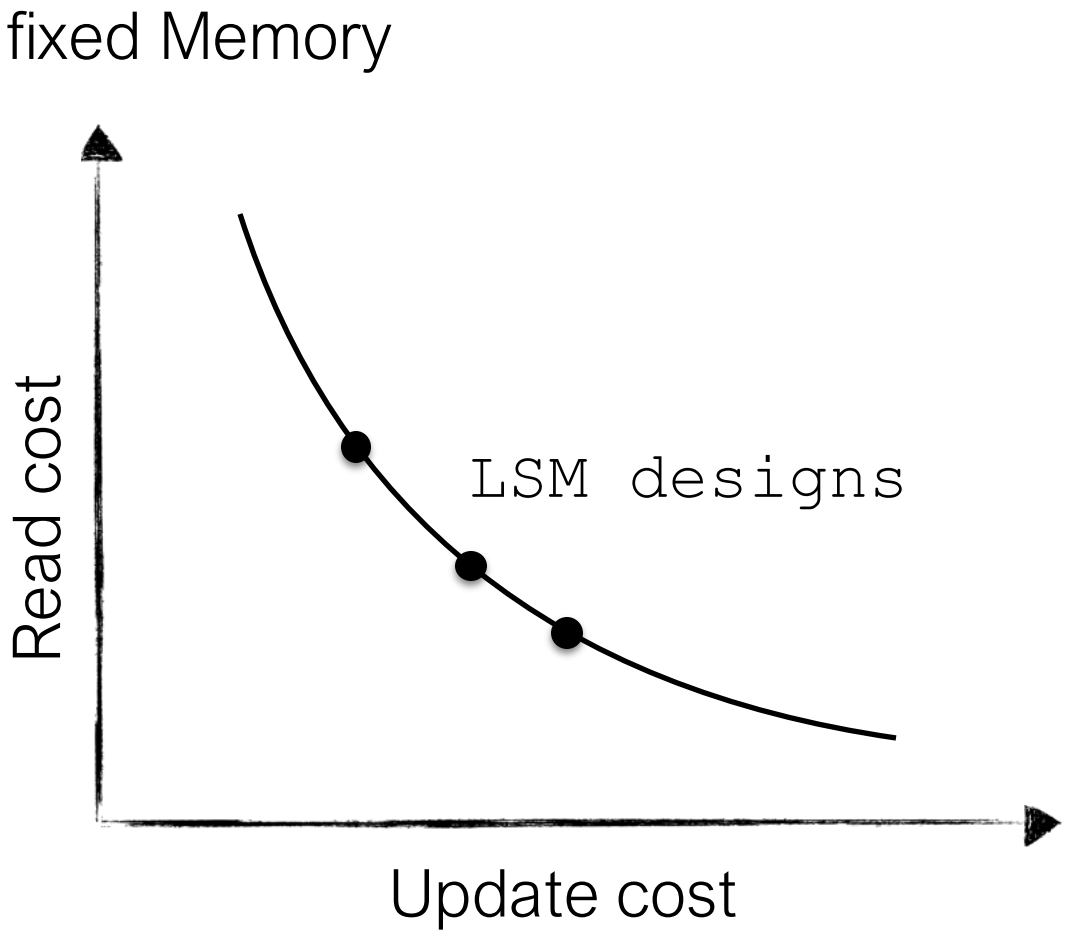
Memory/space footprint

AthanassoulisEDBT16

BOSTON UNIVERSITY

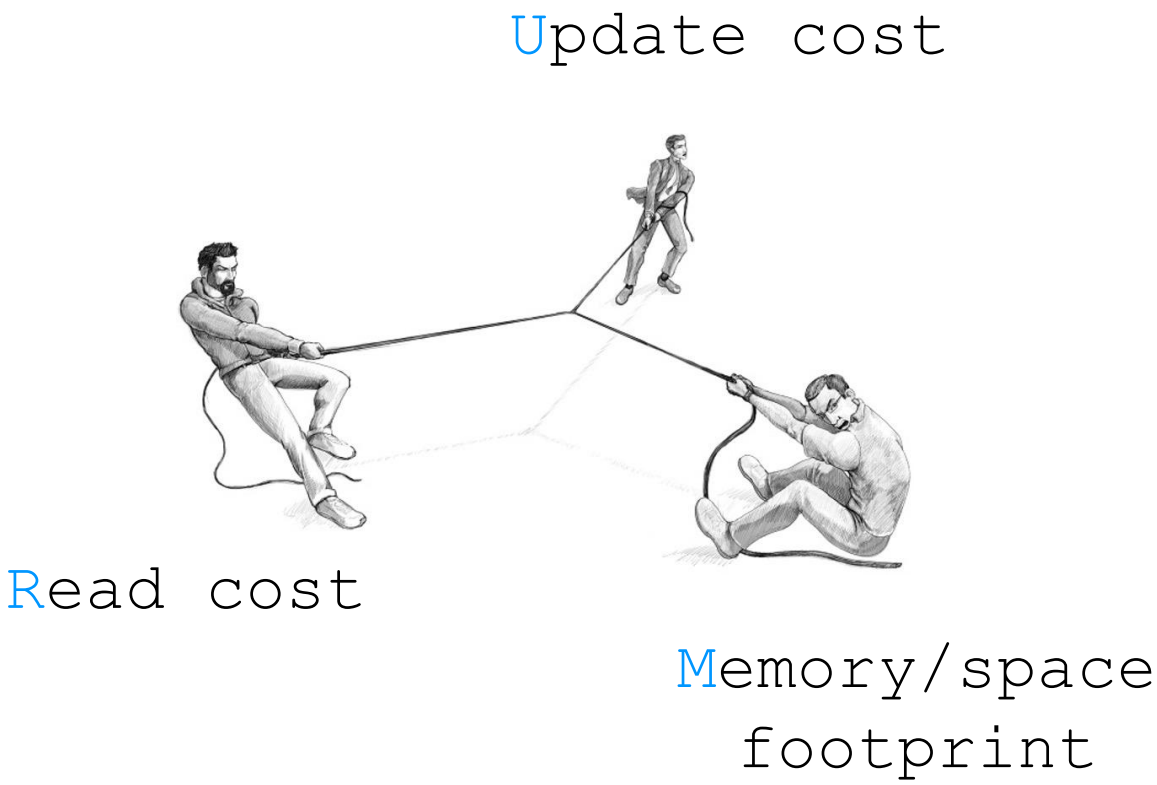
LSM **Design Space**

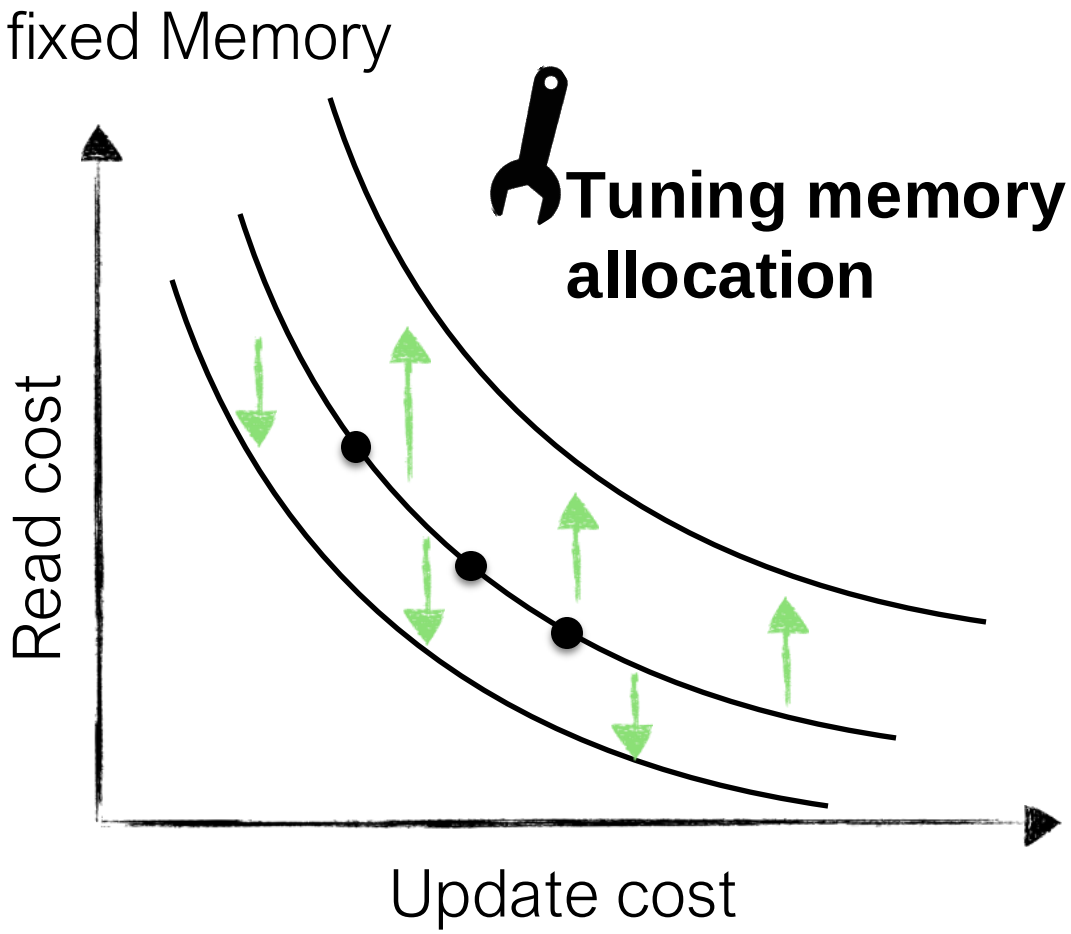




BOSTON UNIVERSITY

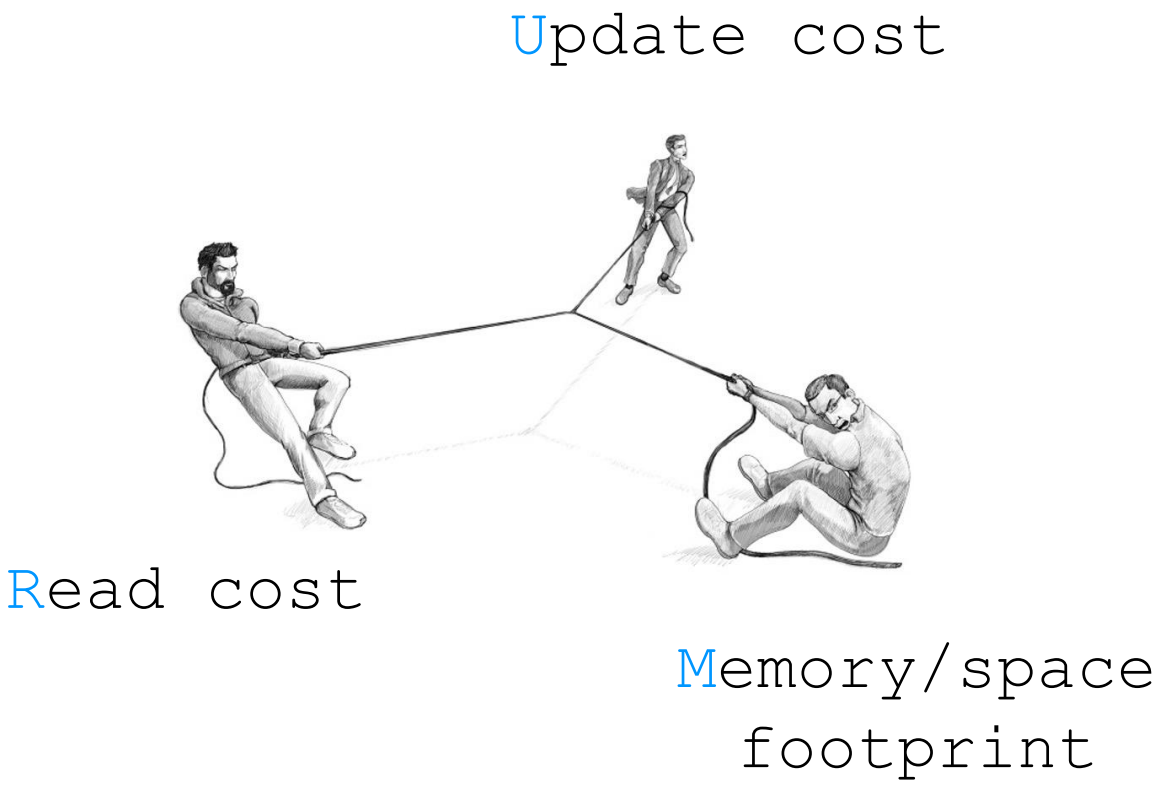
LSM **Design Space**

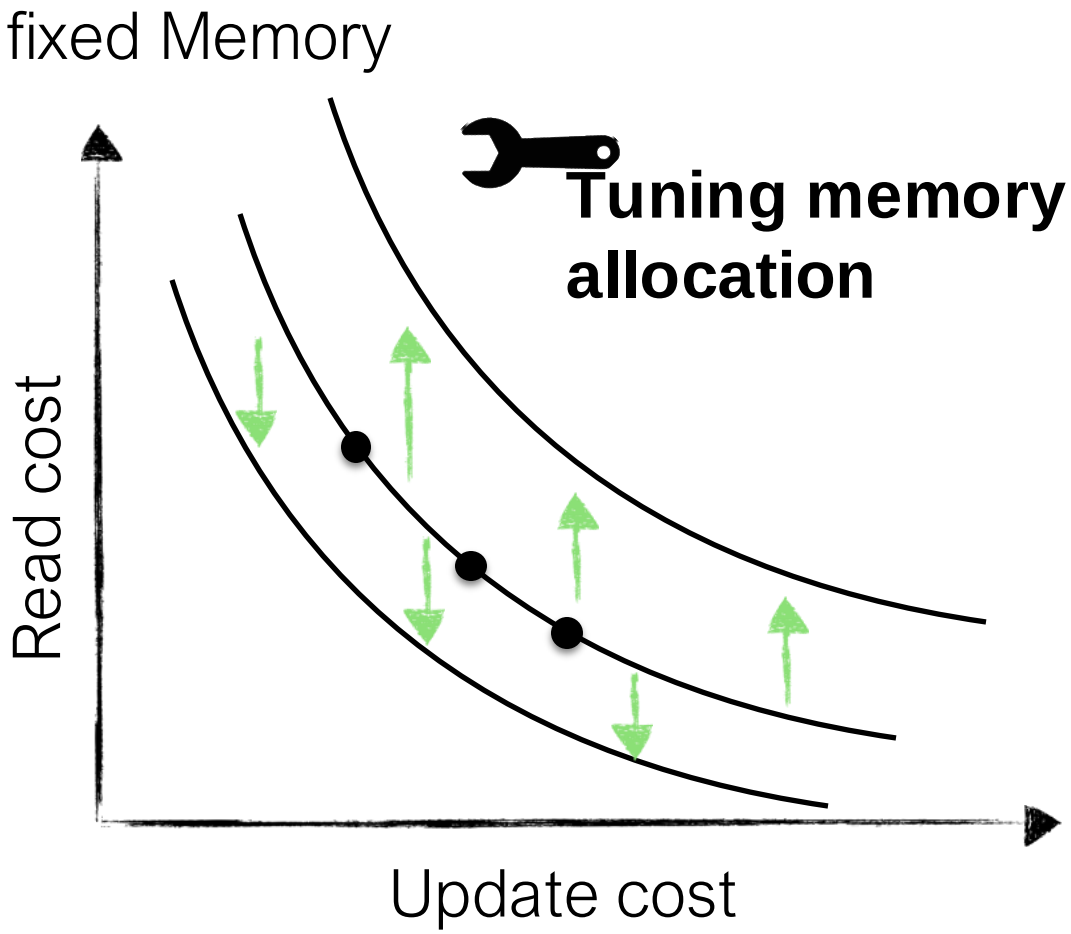




BOSTON UNIVERSITY

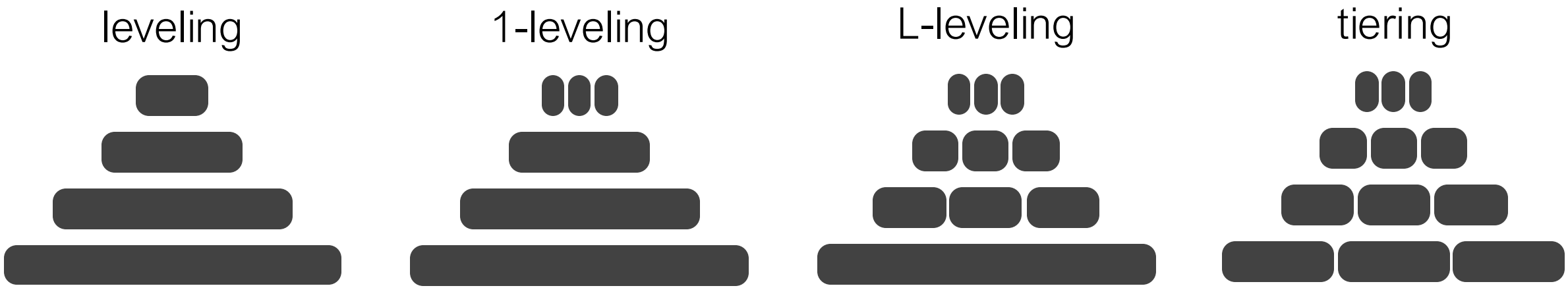
LSM **Design Space**





BOSTON UNIVERSITY

Storage Layer **Design Continuum**



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

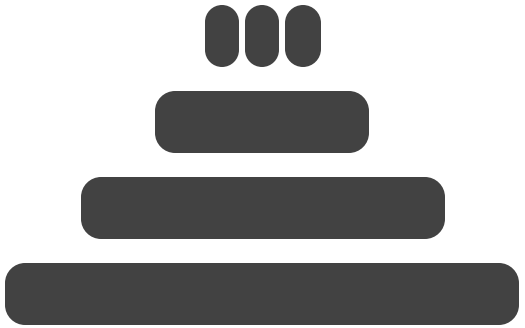
Storage Layer **Design Continuum**

leveling

(T, 0)

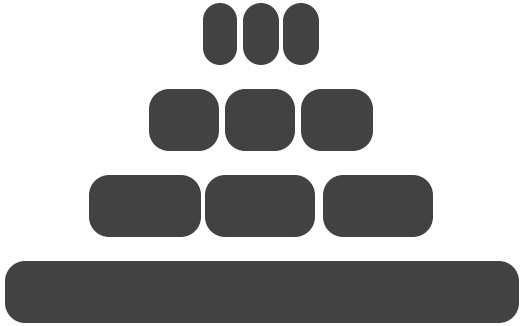
1-leveling

(T, 1)



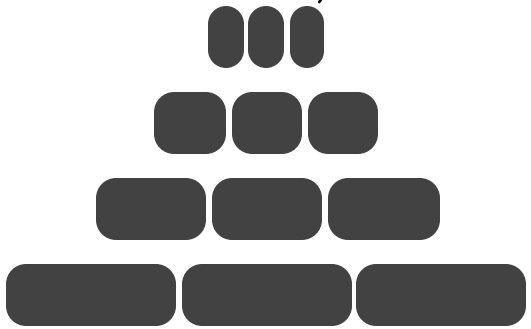
L-leveling

(T, L)



tiering

(T, L+1)

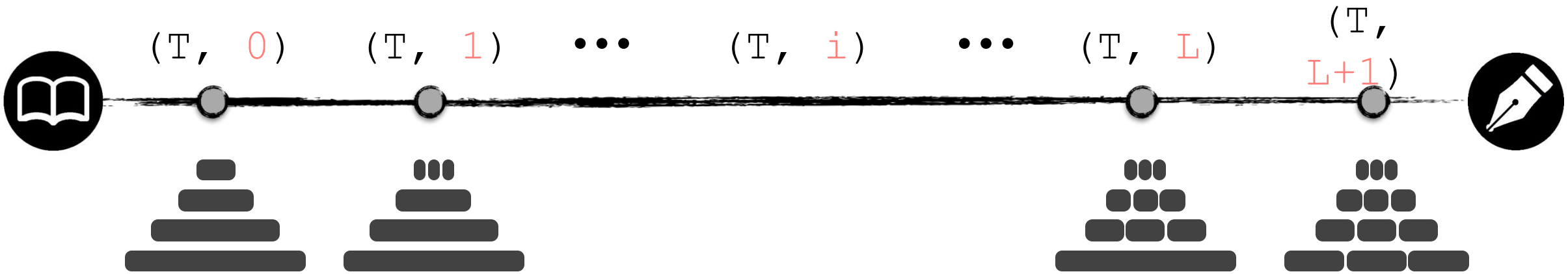


Any design can be defined by the tuple-set: (T, i)

BOSTON UNIVERSITY

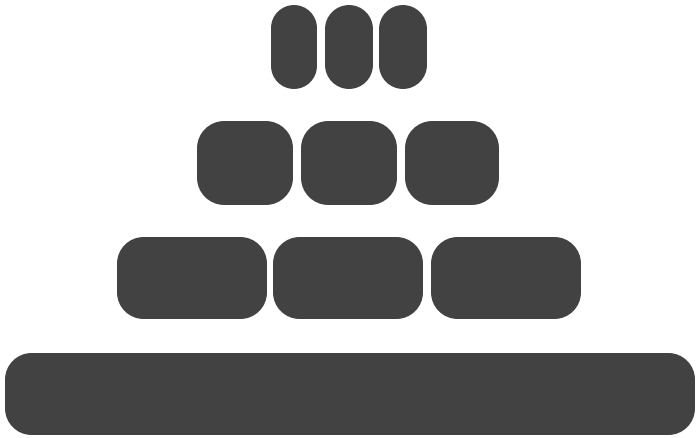
SarkarVLDB21

Storage Layer **Design Continuum**



Storage Layer **Design Continuum**

Storage Layer **Design Continuum**



size ratio

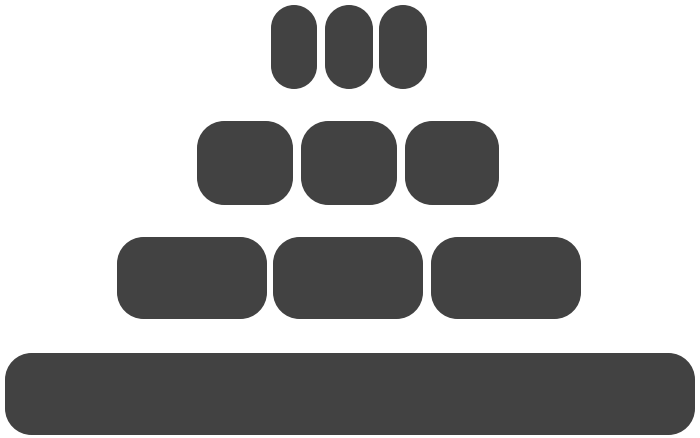
T

T

T

T

Storage Layer **Design Continuum**



size ratio

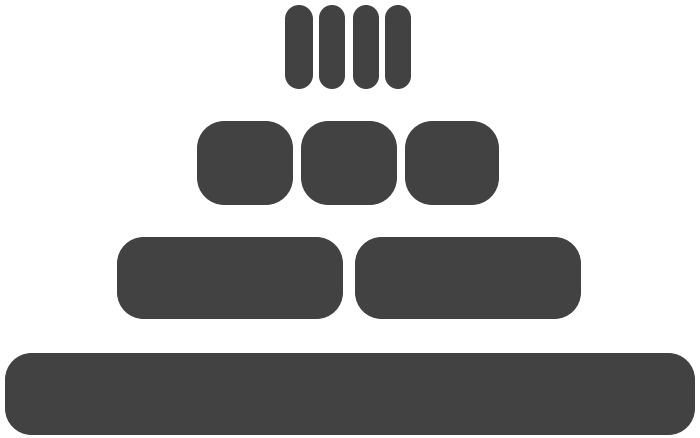
T

T

T

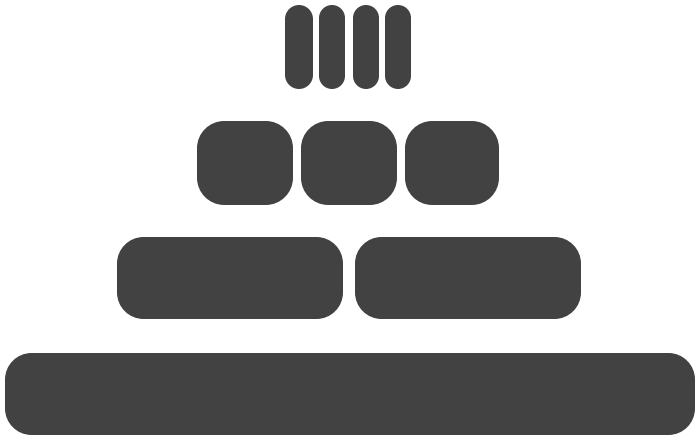
T

Storage Layer **Design Continuum**



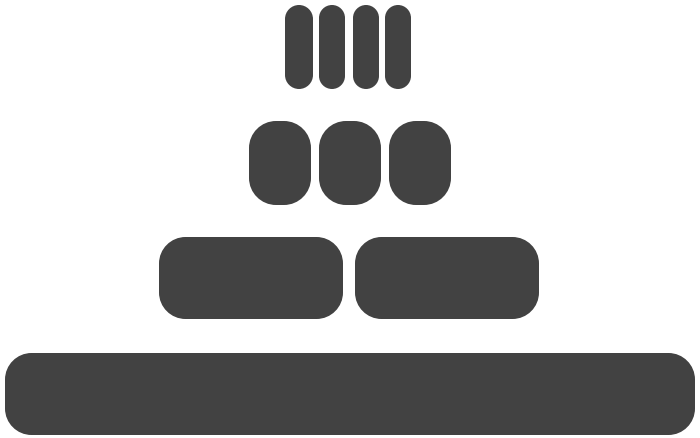
size ratio	#runs
T	4
T	3
T	2
T	1

Storage Layer **Design Continuum**



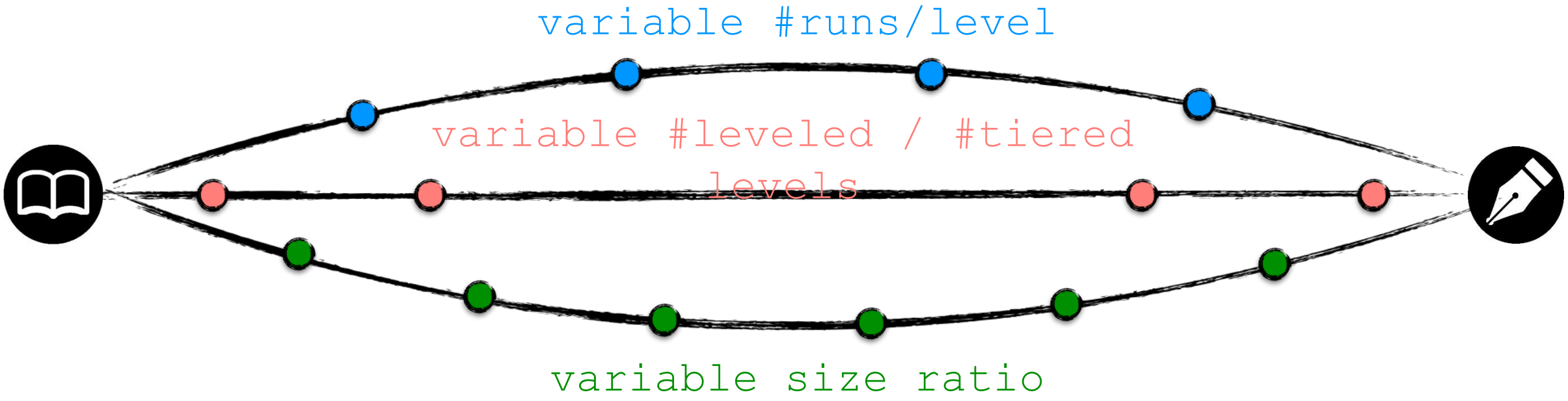
size ratio	#runs
T	4
T	3
T	2
T	1

Storage Layer **Design Continuum**



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

Storage Layer **Design Continuum**



The LSM storage layer design continuum

BOSTON UNIVERSITY

BOSTON UNIVERSITY

CS 561: **Data Systems Architectures**

class 6

Log-structured Merge Trees

Prof. Manos Athanassoulis

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