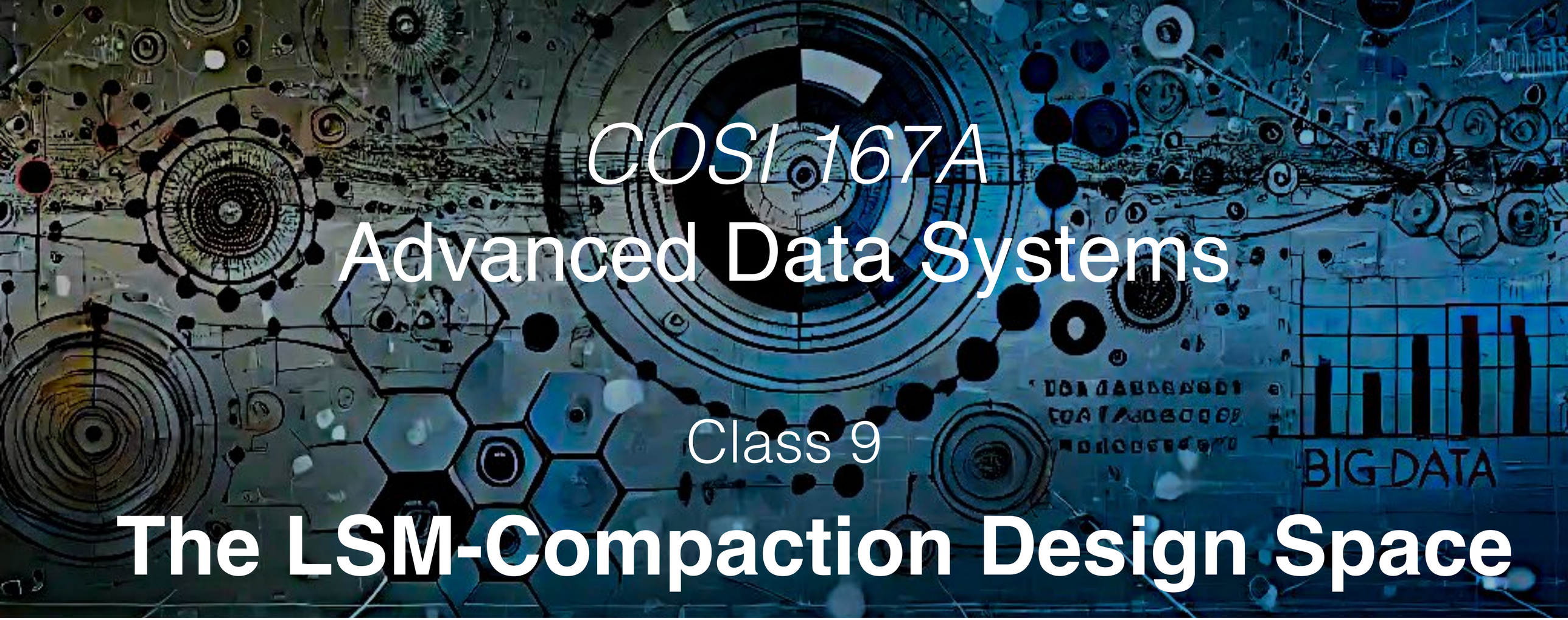


COSI 167A
Advanced Data Systems

Class 9
The LSM-Compaction Design Space

Prof. Subhadeep Sarkar

Brandeis UNIVERSITY

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

lab

Class **logistics**

and administrivia

The **first paper review** is due on **Tue, Oct 1**.

Make sure to go over the **sample review**.

Project proposal is due on **Tue, Oct 8**.

Second guest lecture: next **Tuesday** (**Oct 1**) by **Andy Huynh**.

Paper **review**

How to write a **good** review?

learn

What is the **problem**? Why is it **important**?

What is the **state of the art** and why is it **not enough**?

What is the **key idea** and why/how does it **work**?

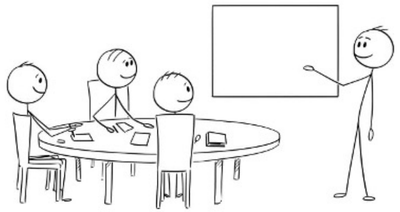
critique

What is **missing**? How can we **improve** this idea?

Does the paper **support all its claims**?

What are some possible **next steps** of the work?

Brandeis UNIVERSITY



Paper **presentation**

and discussion

Register for **paper presentation**! (https://shorturl.at/4POIT)

#	Date	Paper Name	Presenter 1	Presenter 2
1	Oct 18	Learning to Optimize LSM-trees: Towards A Reinforcement Learning based Key-Value Store for Dynamic Workloads	Alex Ott	James Chen
2	Oct 29	The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases, ICDE, 2013	Tal Kronrod	Archer Heffern
3	Nov 1	Adaptive Adaptive Indexing, ICDE, 2018	Parthiv Ganguly	Arun Shrestha
4	Nov 19	FASTER: A Concurrent Key-Value Store with In-Place Updates, SIGMOD, 2018	Alex Stevenson	Abbie Murphy
5	Nov 26	The Data Calculator: Data Structure Design and Cost Synthesis from First Principles and Learned Cost Models		

Today in COSI 167A

What's on the cards?

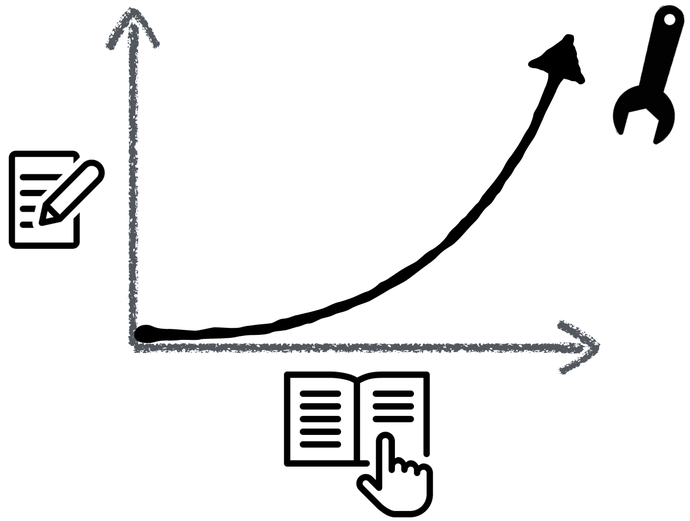
compactions

analyzing the **LSM compaction design space**

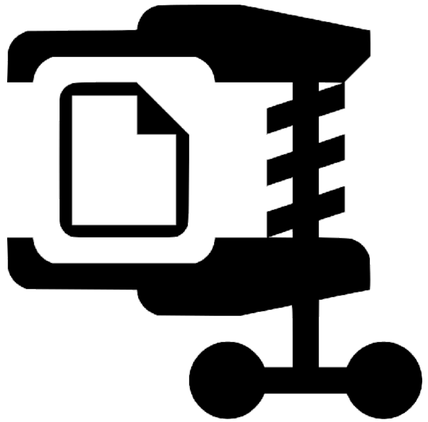
Why **LSM**?

What's the hype all about?

fast writes



tunable read-write performance



good space utilization

Operating **principles**

The foundational pillars!

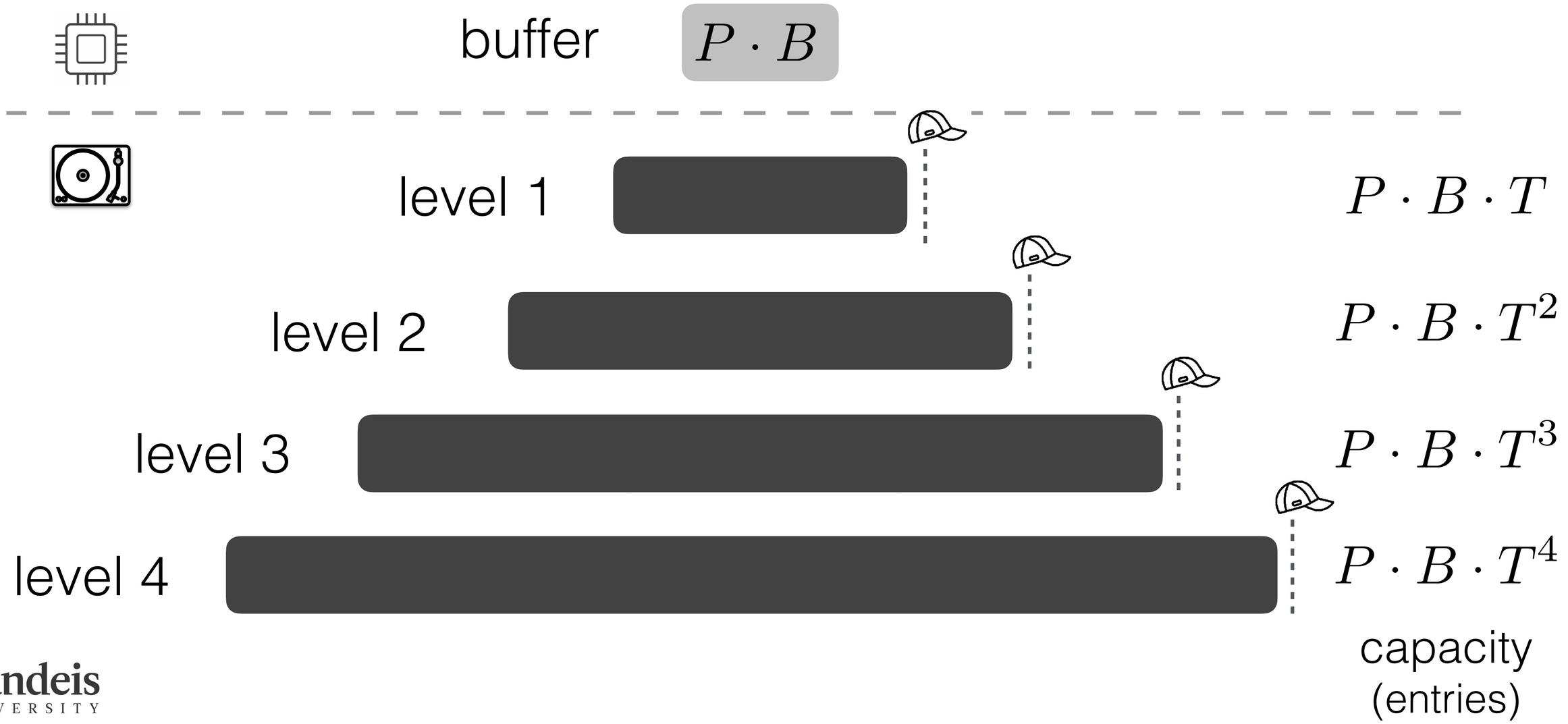
Buffering ingestion

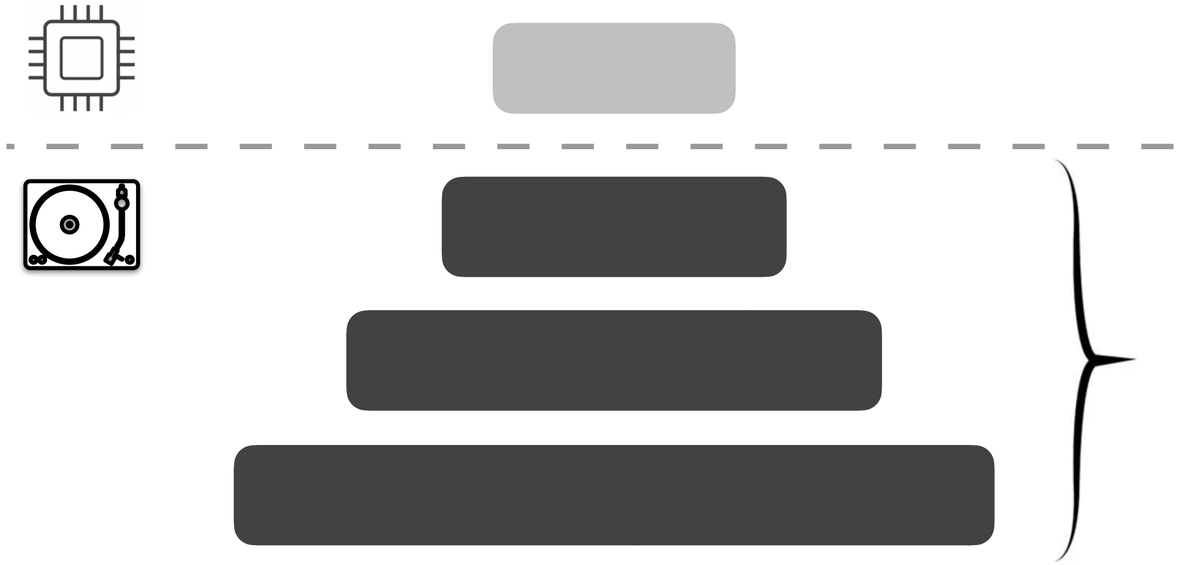
Immutable files on storage

Out-of-place updates & **deletes**

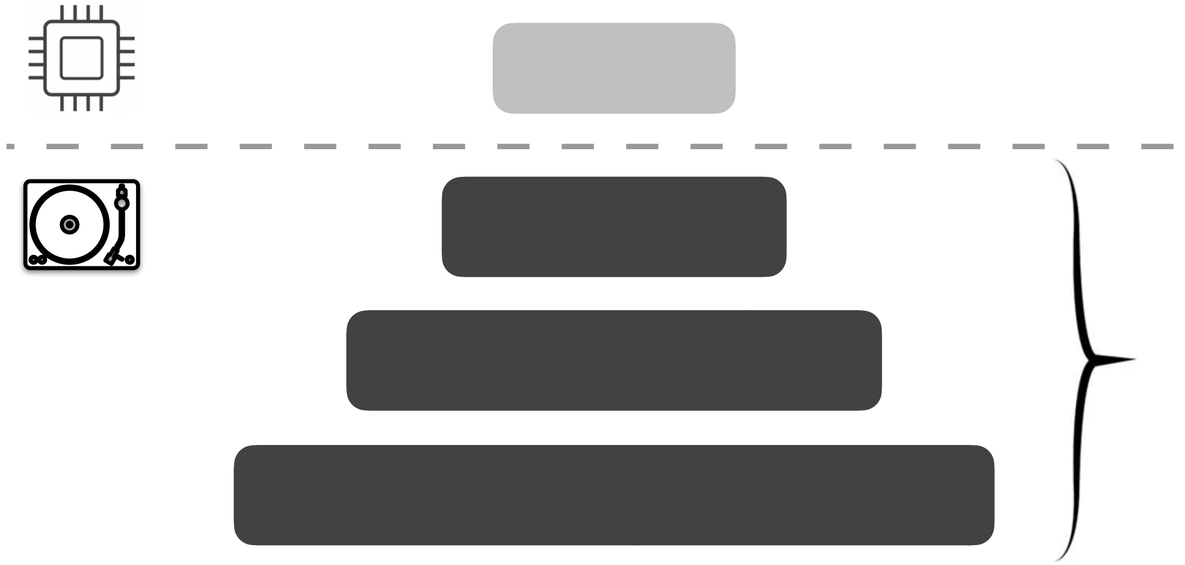
Periodic **data layout reorganization**

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



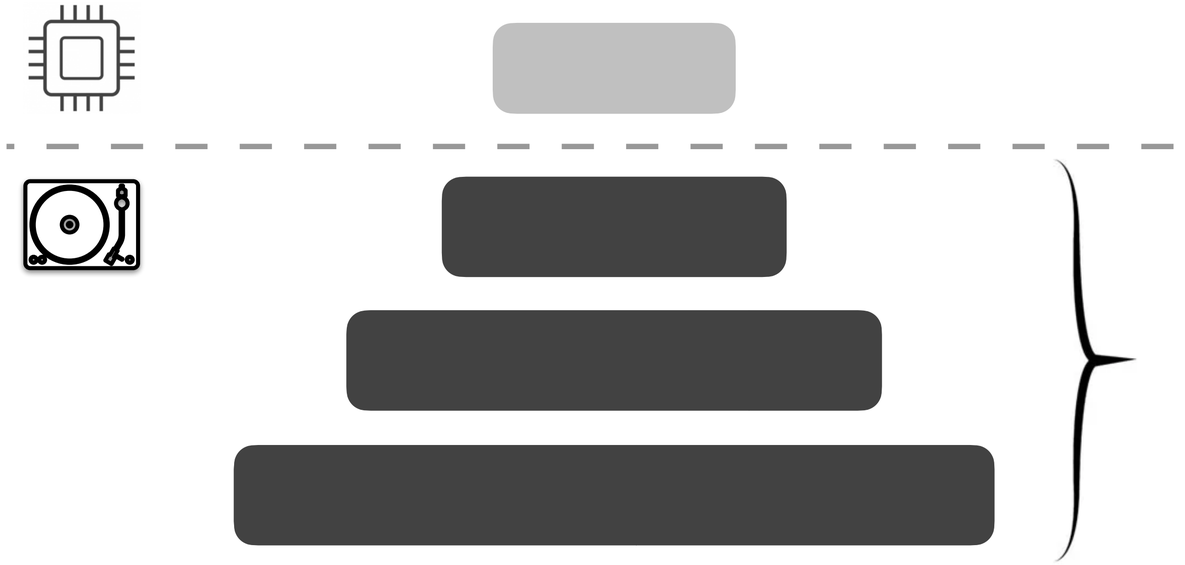


most data
on storage



most data
on storage

L: #levels
T: size ratio

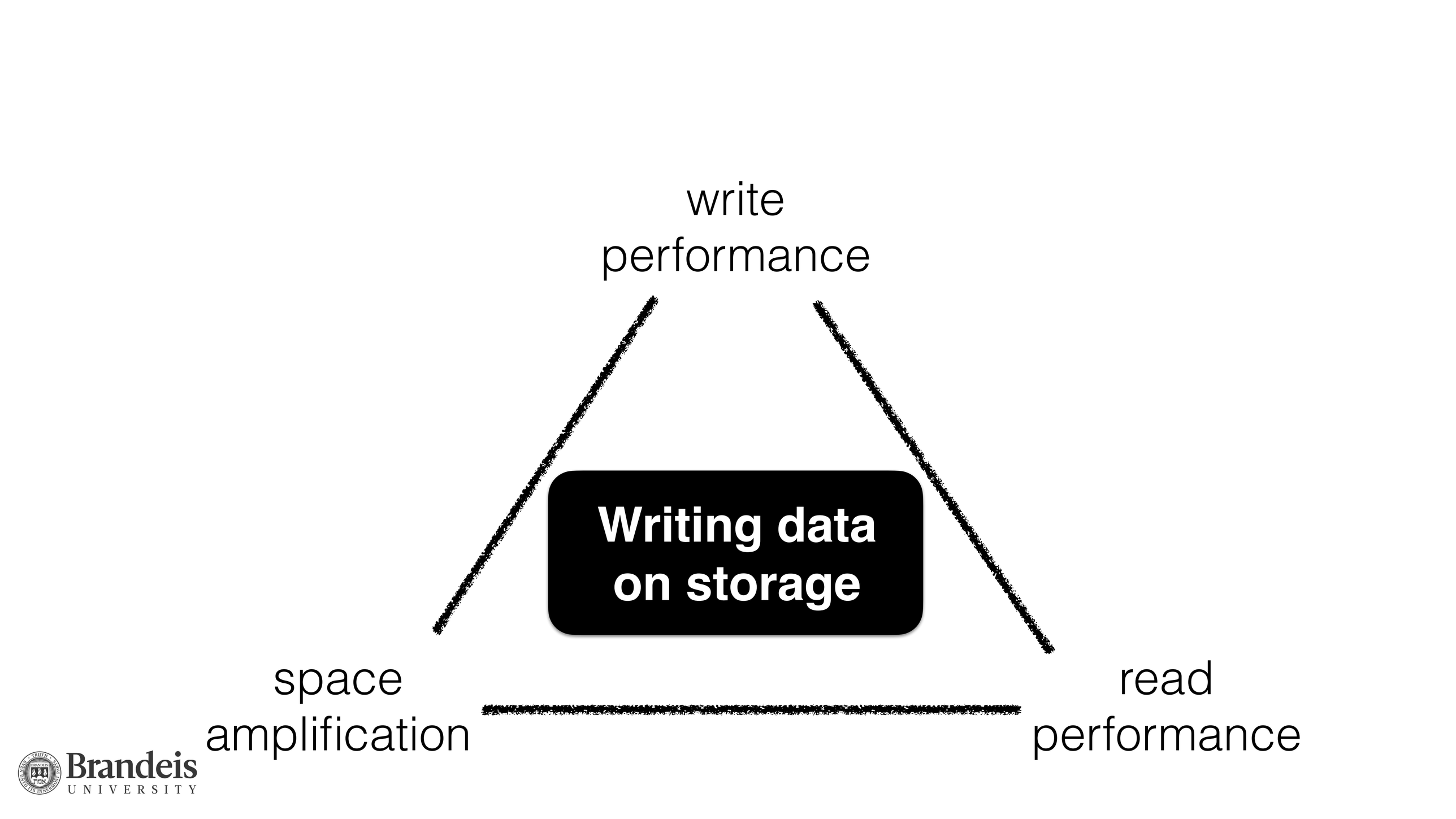


most data
on storage

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

99.9% on storage

How does the storage layer affect performance?



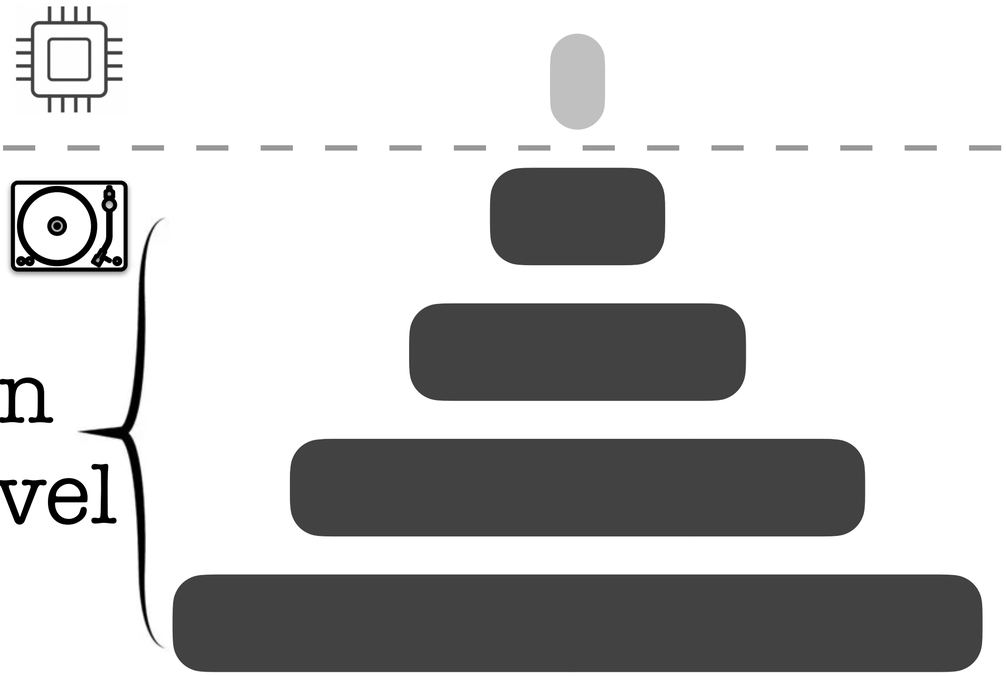
Data **Layout**

Classical LSM design: **leveling**

[eager merging]

Data **Layout**

leveling [eager]

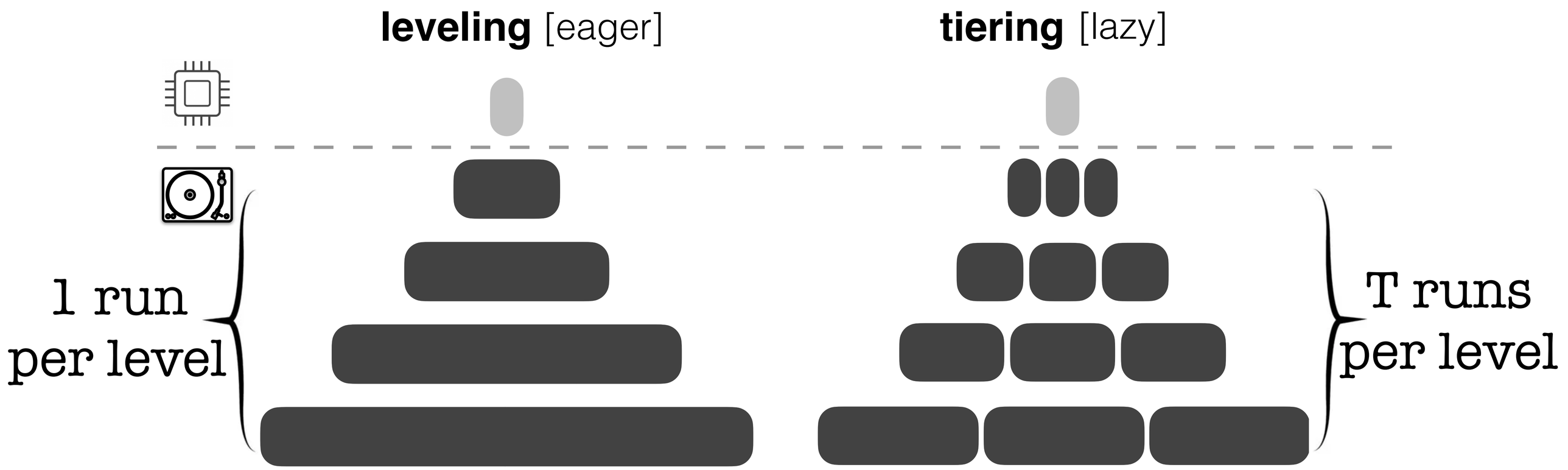


1 run per level

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

any **limitations**?

Data **Layout**



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

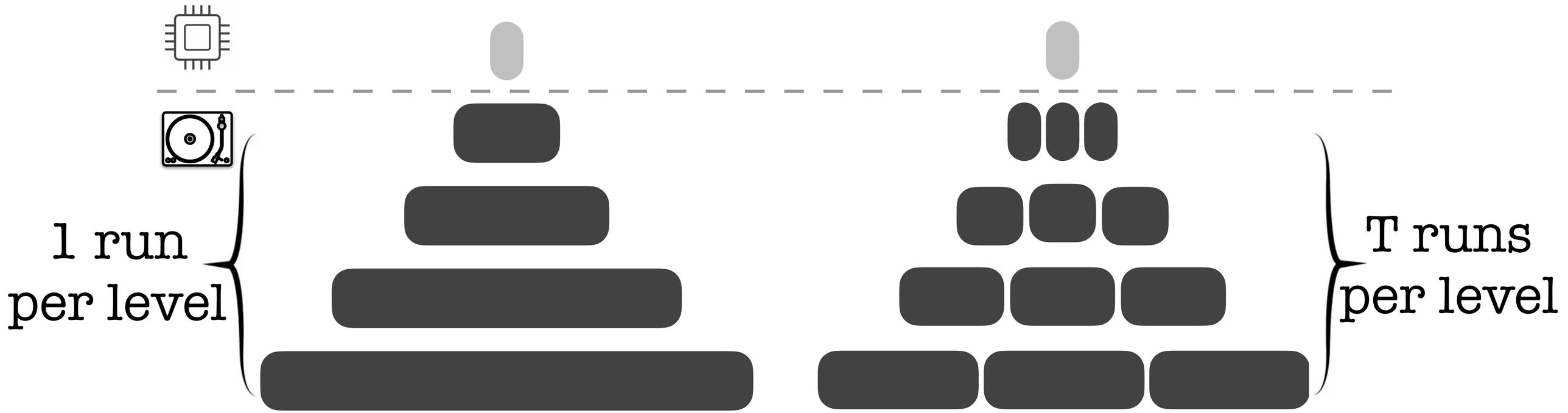
- good ingestion performance
- poor space amplification
- poor read performance

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

Data **Layout**

leveling [eager]

tiering [lazy]



Read cost: $\mathcal{O}(L \cdot \phi)$ $\mathcal{O}(T \cdot L \cdot \phi)$

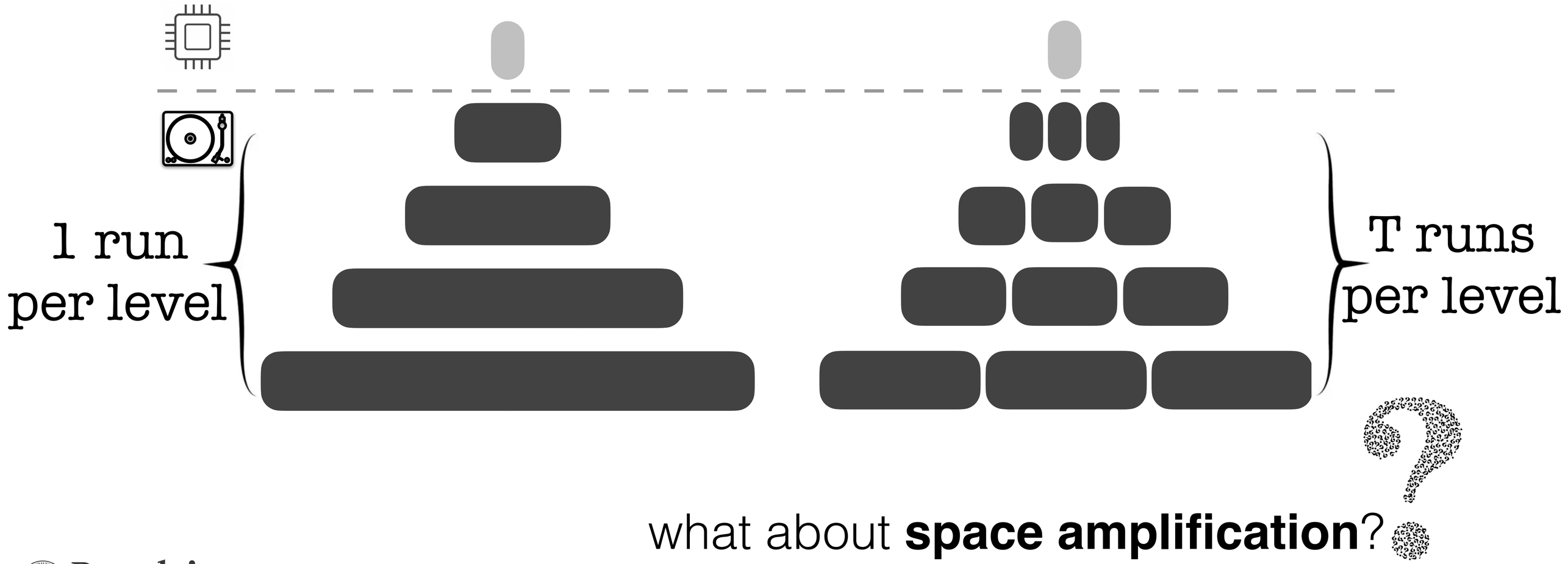
Write cost: $\mathcal{O}(T \cdot L/B)$ $\mathcal{O}(L/B)$

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

Data **Layout**

leveling [eager]

tiering [lazy]



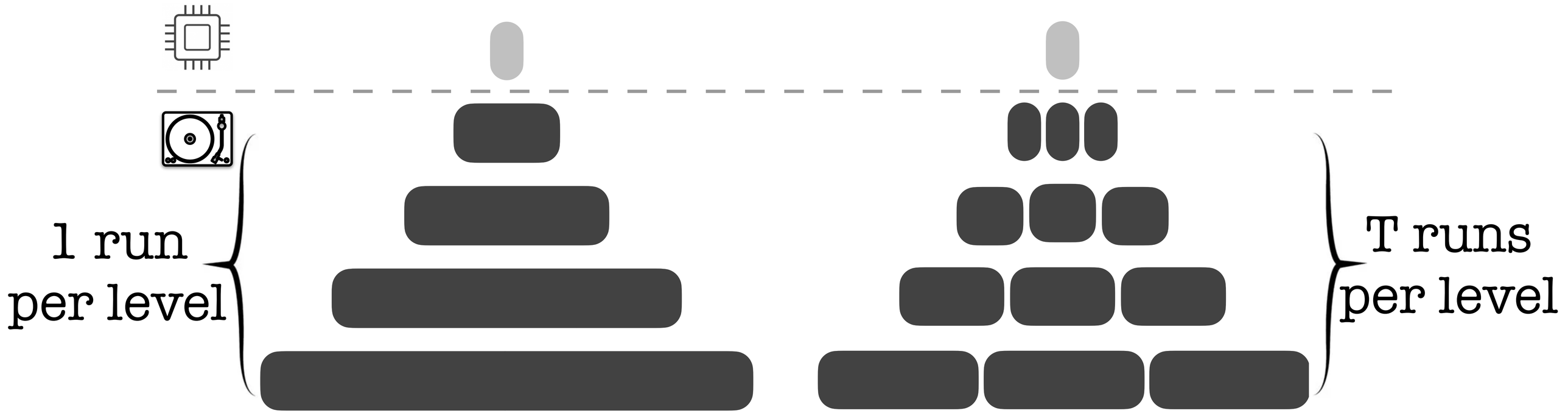
what about **space amplification**?

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

Data **Layout**

leveling [eager]

tiering [lazy]



Space amplification

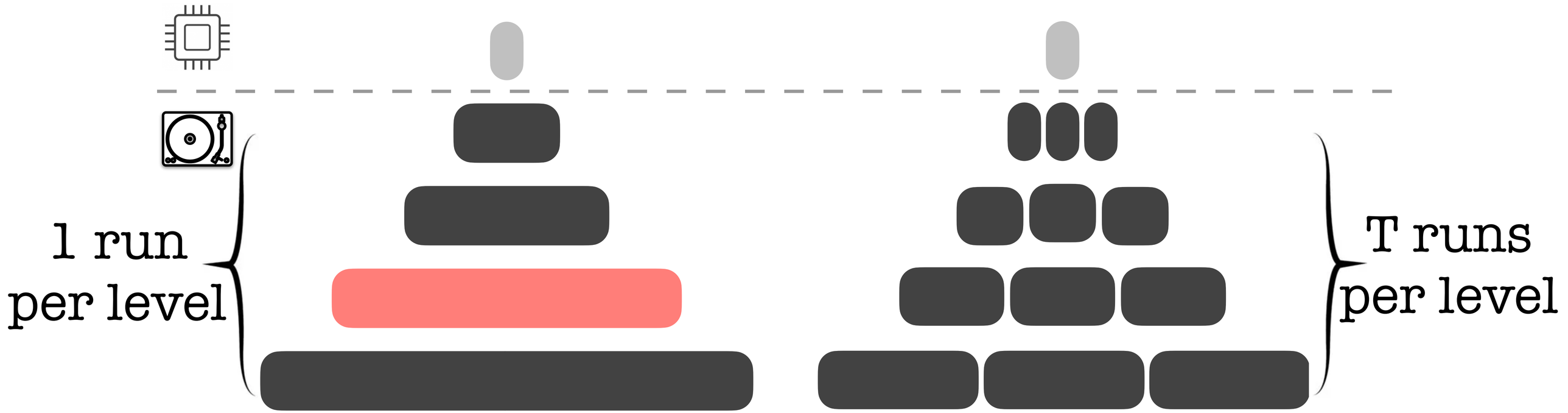
the ratio of **logically invalid data size** to the **total data size**

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

Data **Layout**

leveling [eager]

tiering [lazy]

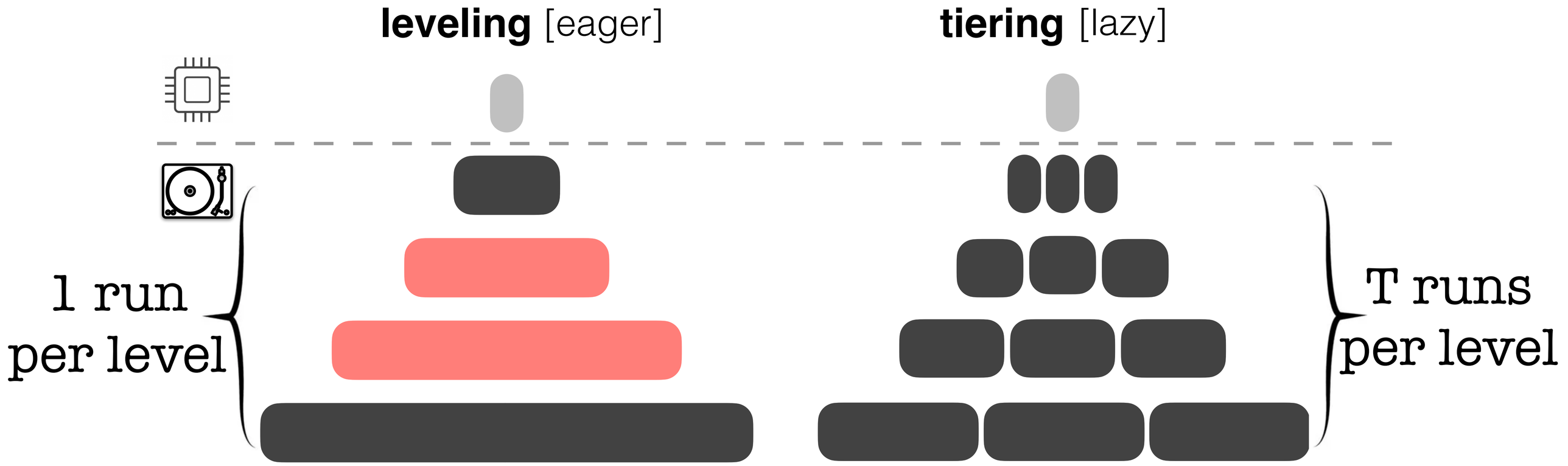


Space amplification

the ratio of **logically invalid data size** to the **total data size**

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

Data **Layout**

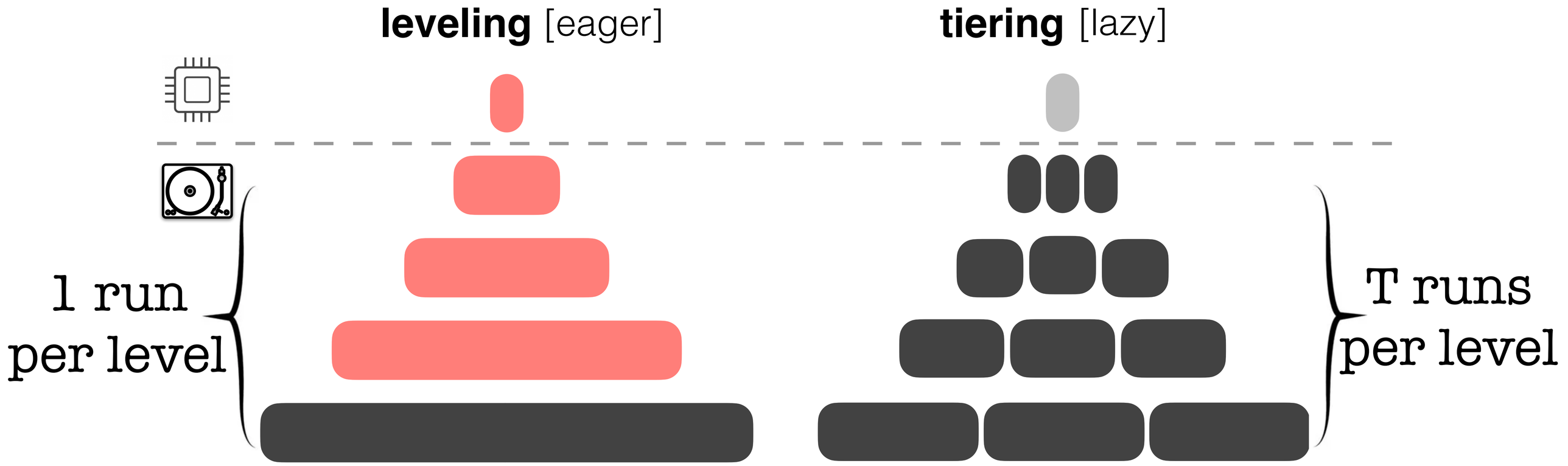


Space amplification

the ratio of **logically invalid data size** to the **total data size**

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

Data **Layout**



Space amplification

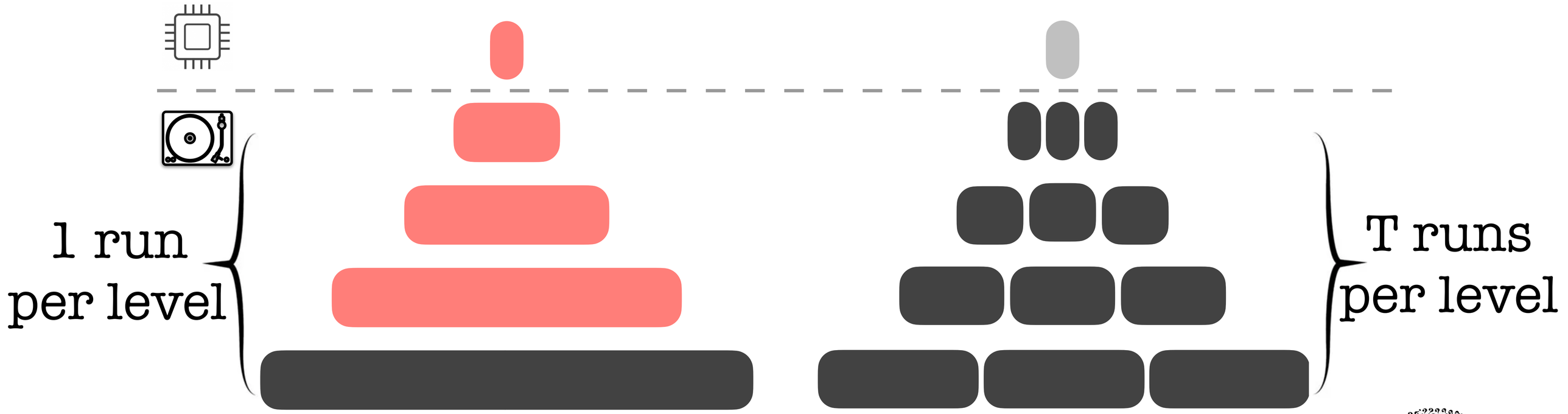
the ratio of **logically invalid data size** to the **total data size**

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

Data **Layout**

leveling [eager]

tiering [lazy]



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

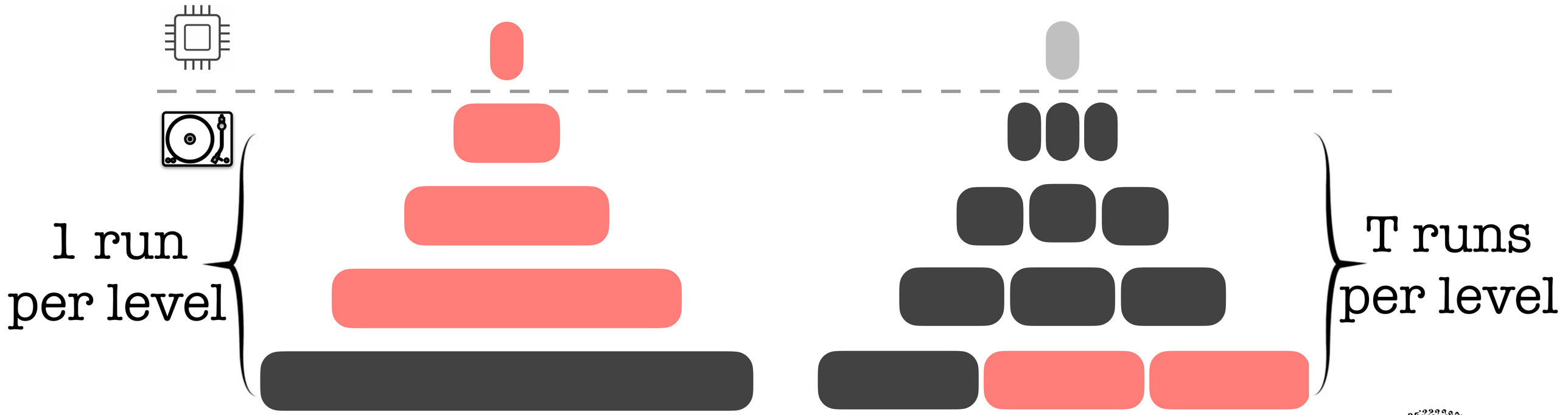
how about **tiering**?

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

Data **Layout**

leveling [eager]

tiering [lazy]



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

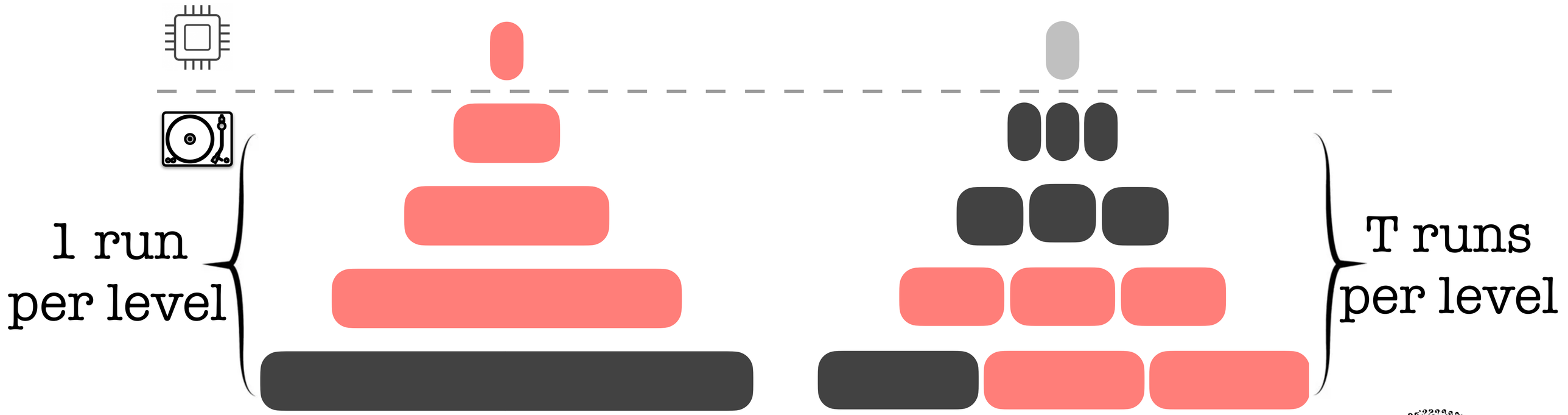
how about **tiering**?

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

Data **Layout**

leveling [eager]

tiering [lazy]



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

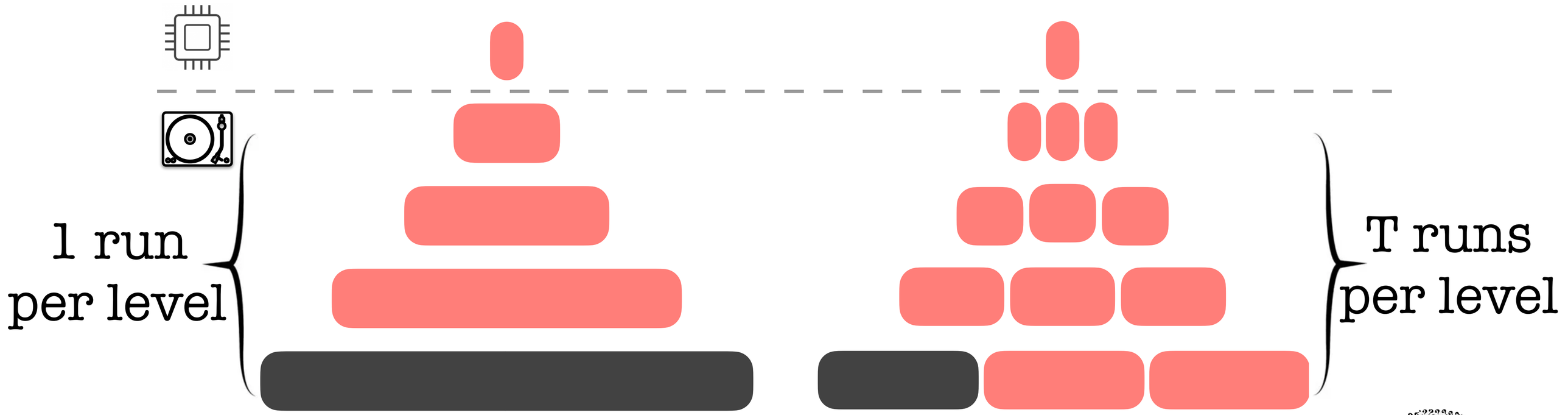
how about **tiering**?

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

Data **Layout**

leveling [eager]

tiering [lazy]



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

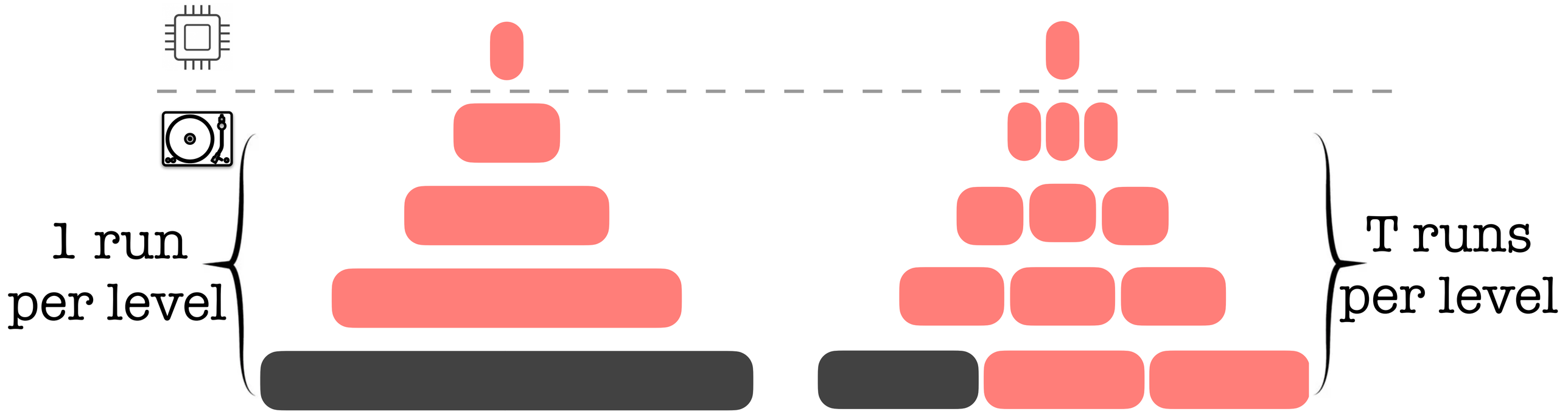
how about **tiering**?

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

Data **Layout**

leveling [eager]

tiering [lazy]

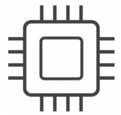


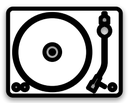
Space amplification: $\mathcal{O}(1/T)$ $\mathcal{O}(T)$

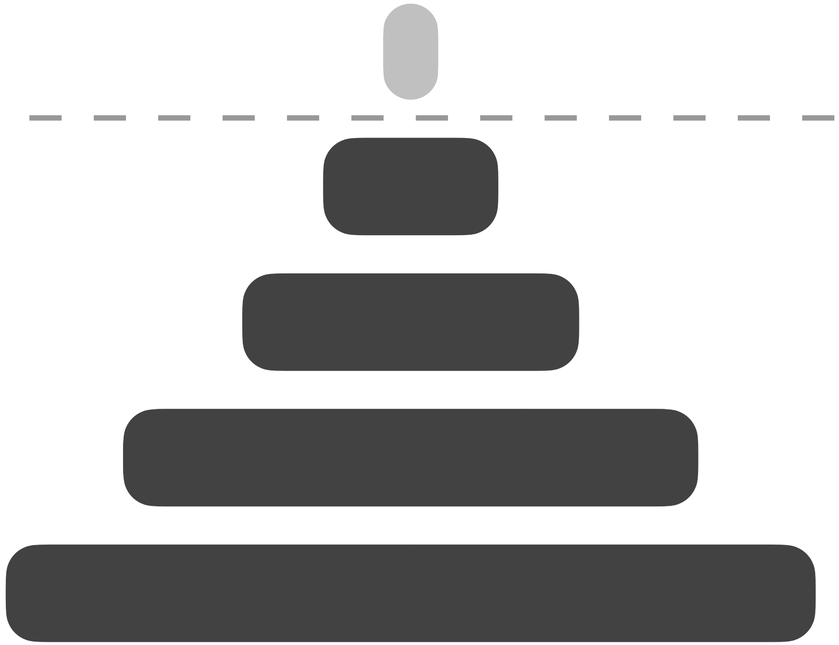
Data **Layout**

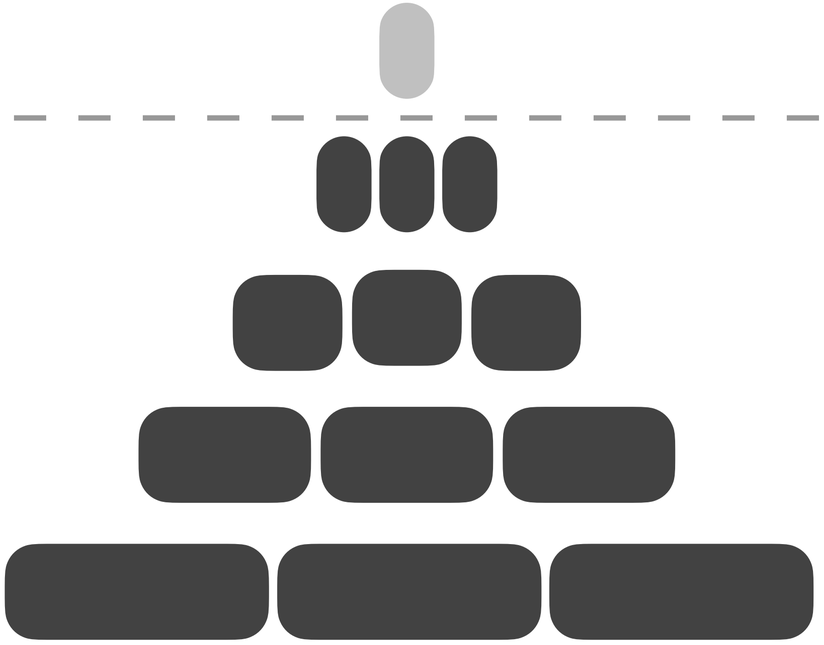
leveling [eager]

tiering [lazy]







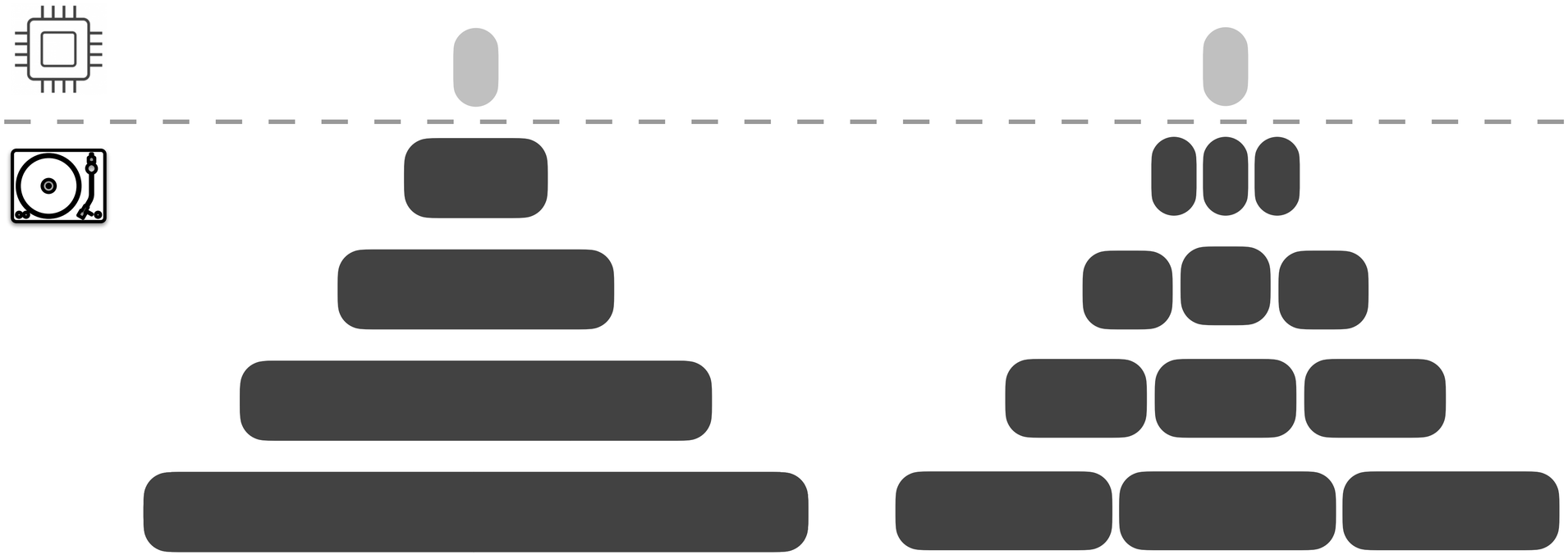


Brandeis UNIVERSITY

Data **Layout**

leveling [eager]

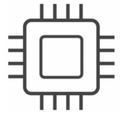
tiering [lazy]

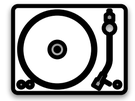


What happens if T becomes too large?

Data **Layout**

leveling [eager]



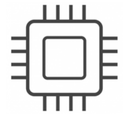


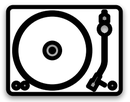
sorted array

Data **Layout**

leveling [eager]

tiering [lazy]



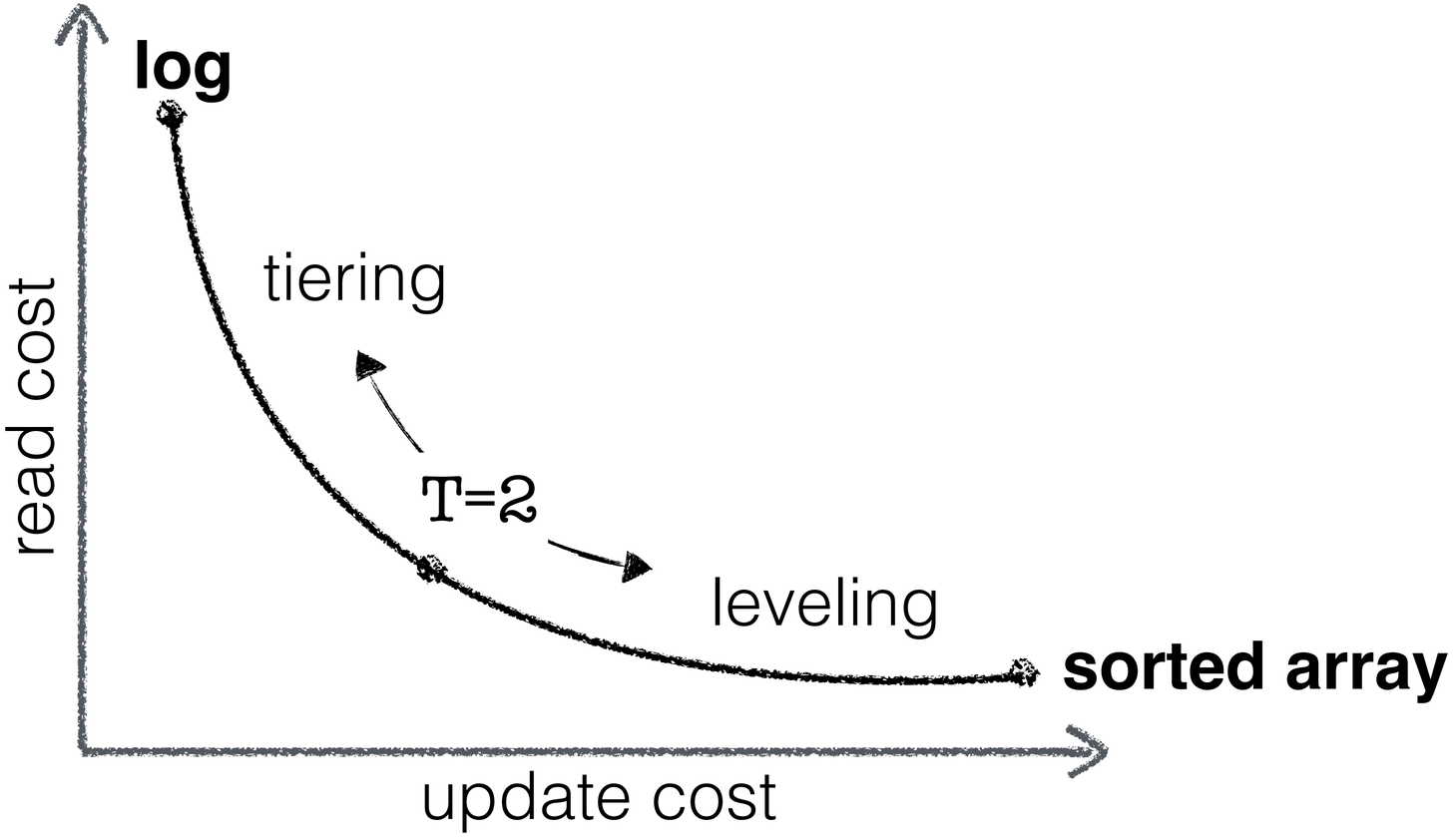


sorted array

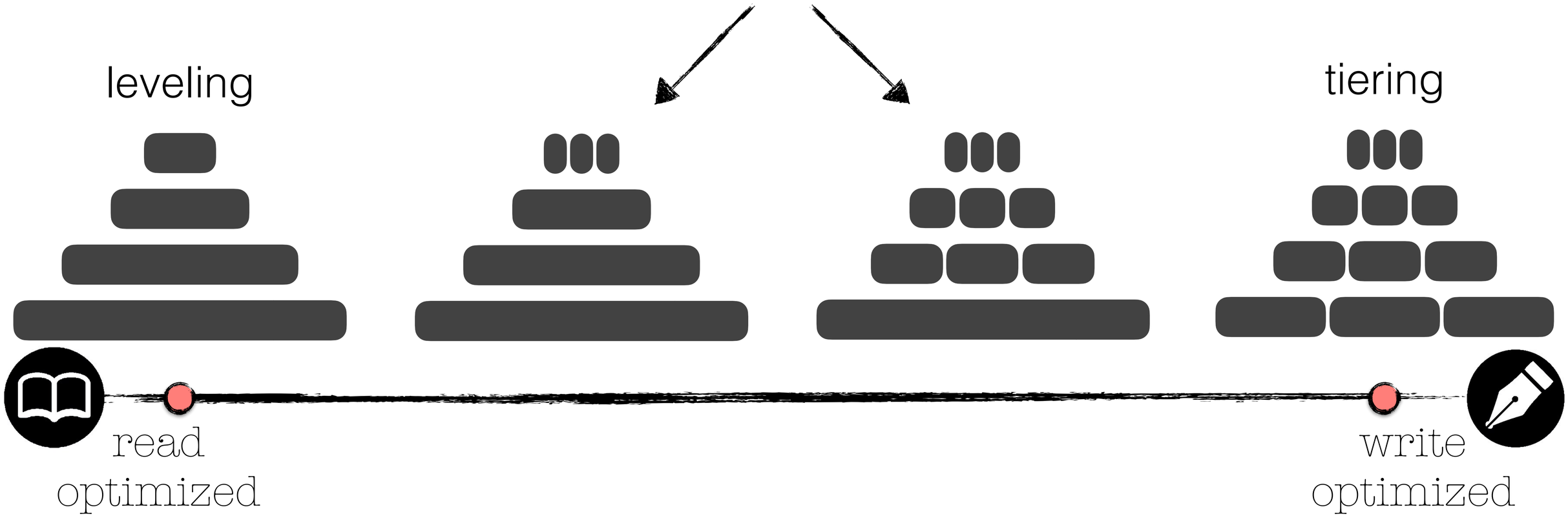
log

T: size ratio

Data **Layout**

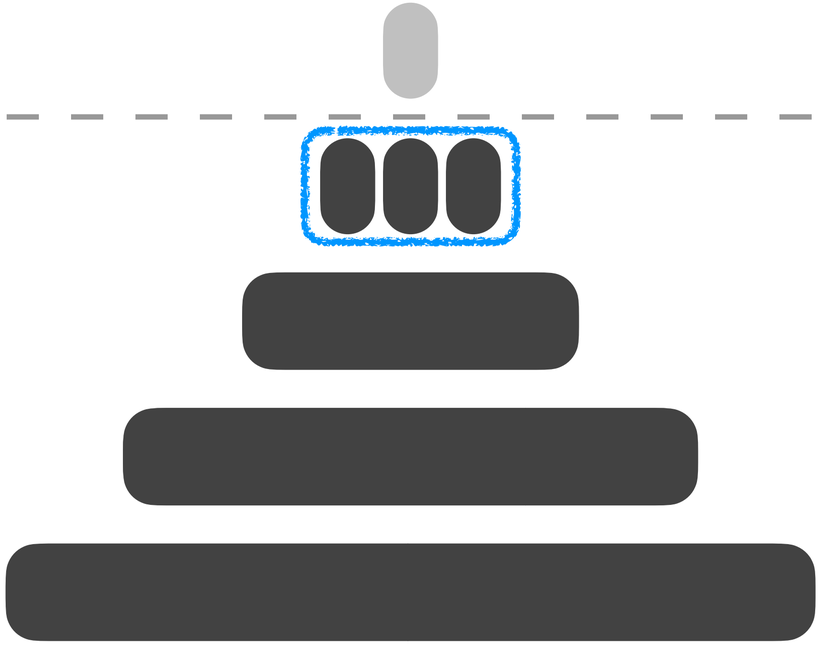


Data **Layout**


hybrid designs
leveling
tiering
read
optimized
write
optimized

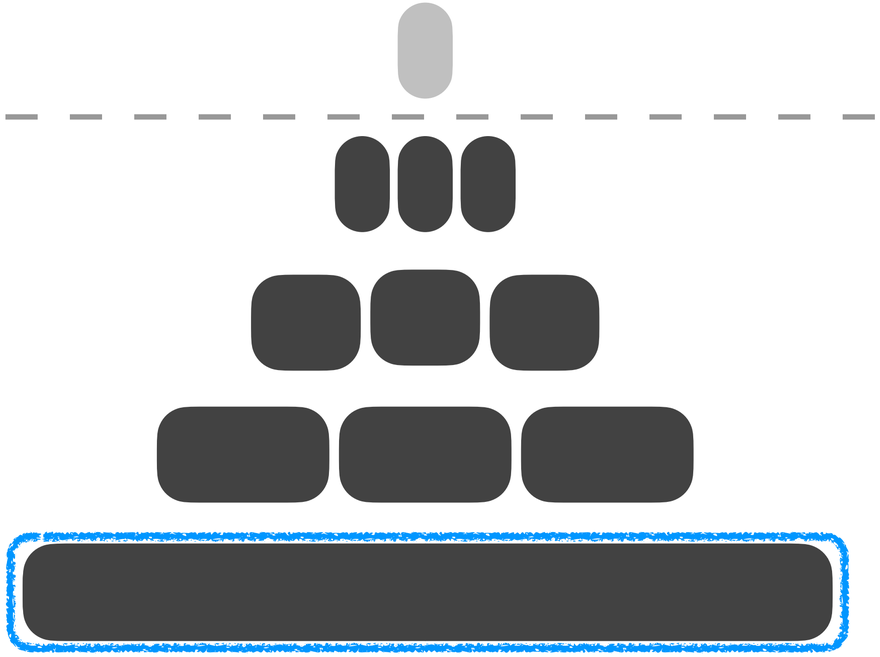
Data **Layout**

1-leveling



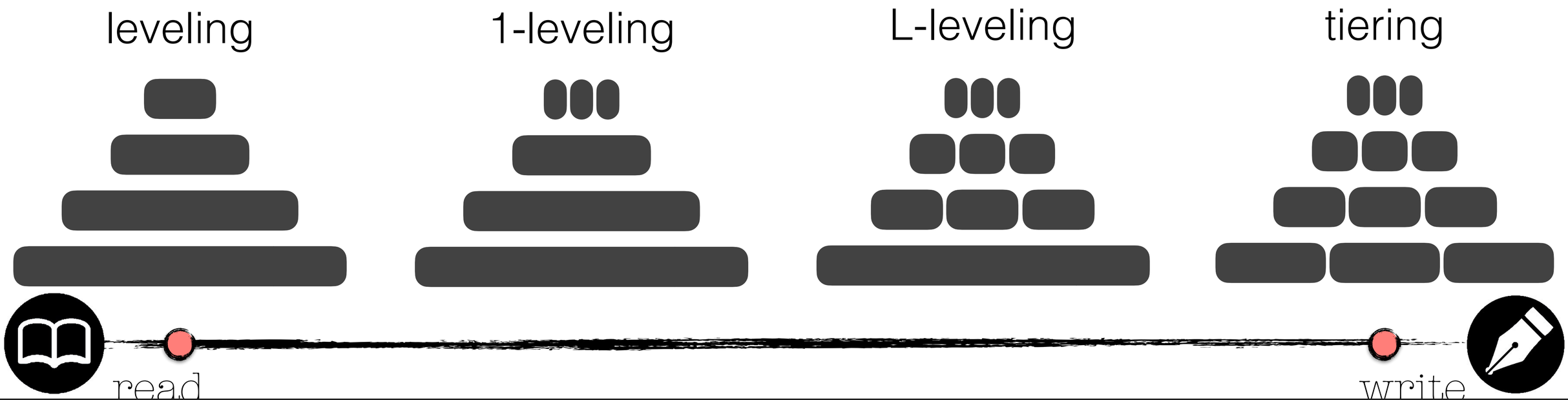
- fewer write stalls
- increased block cache hits

L-leveling

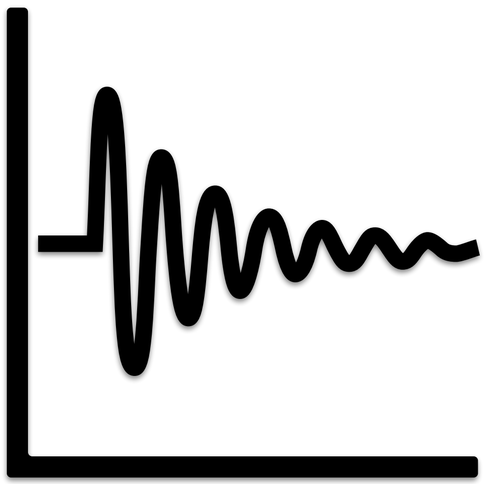


- low write amplification
- better read performance

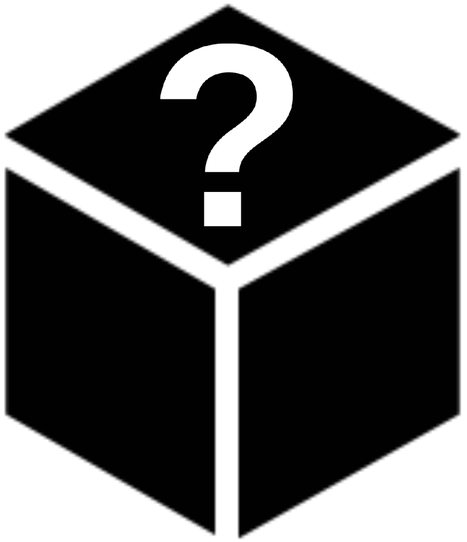
Data **Layout**



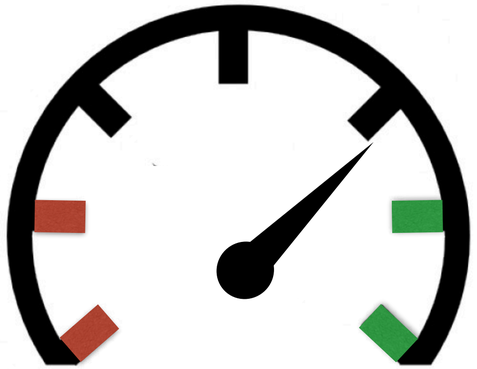
So, how do we reason about the data layout?



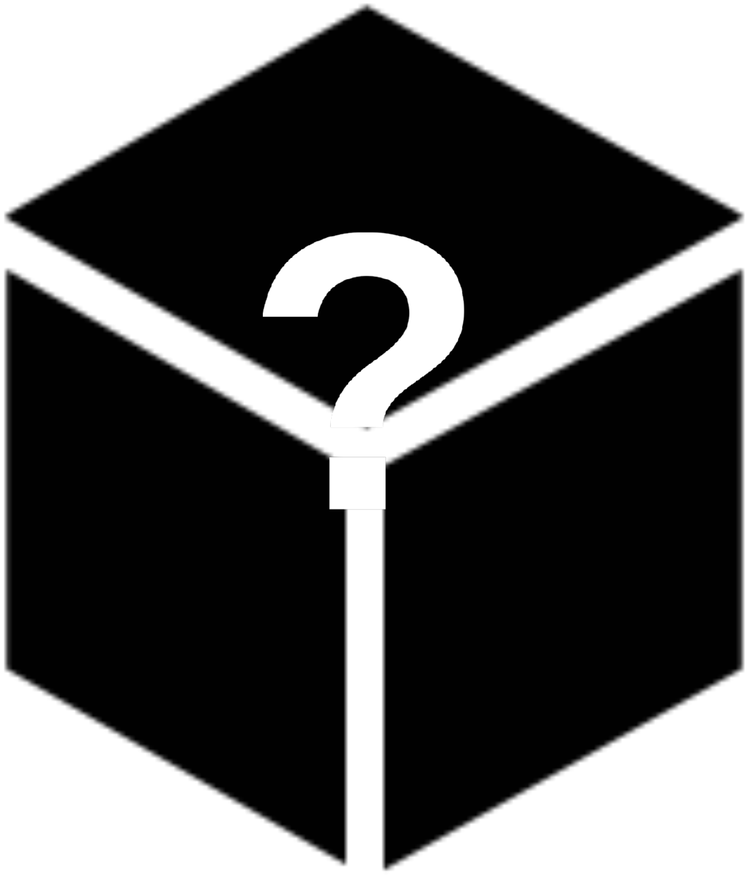
workload



data layout



performance



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?

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

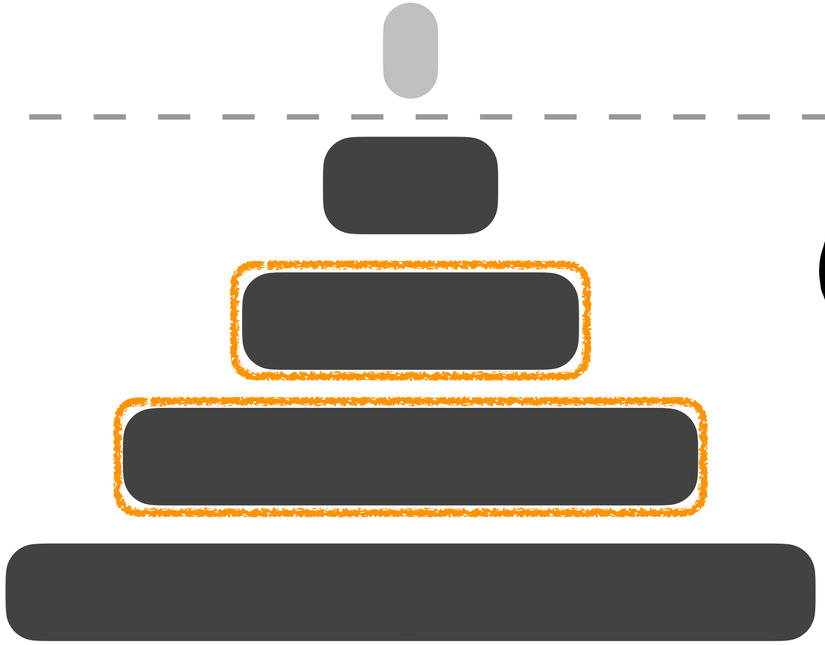
When to re-organize the data layout?

Compaction **Granularity**

data moved per compaction

Compaction **Granularity**

data moved per compaction



consecutive levels

Thought Experiment 1

Problems with **full-level compaction**?

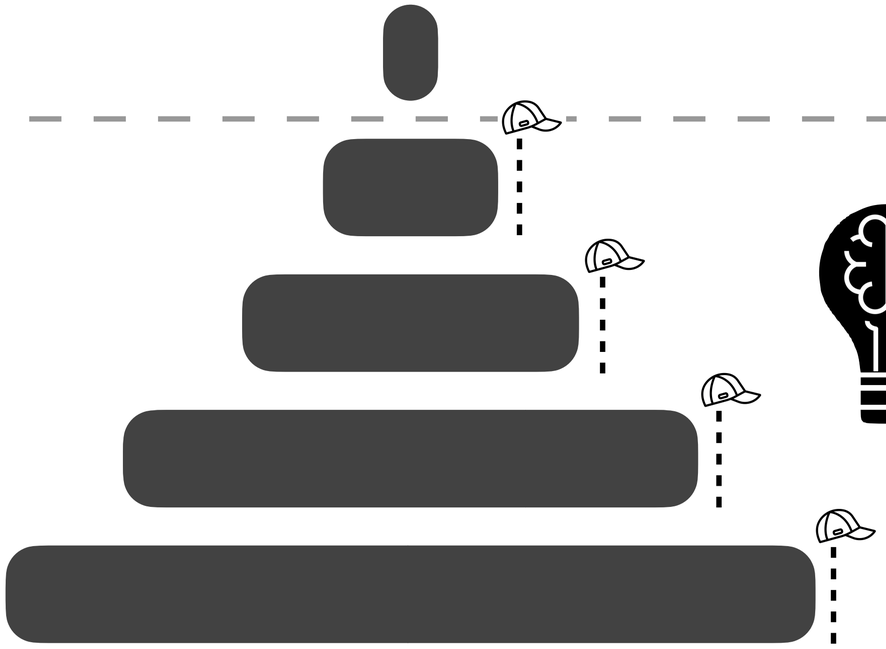
Cascading compactions!

AsterixDB™

Brandeis UNIVERSITY

Compaction **Granularity**

data moved per compaction



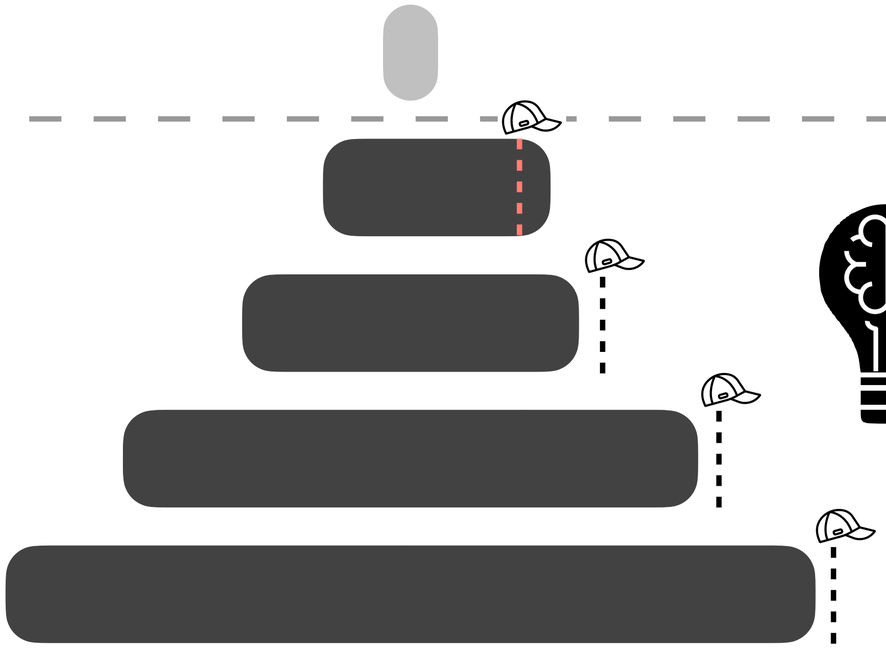
Thought Experiment 1

Problems with **full-level compaction**?

Cascading compactions!

Compaction **Granularity**

data moved per compaction



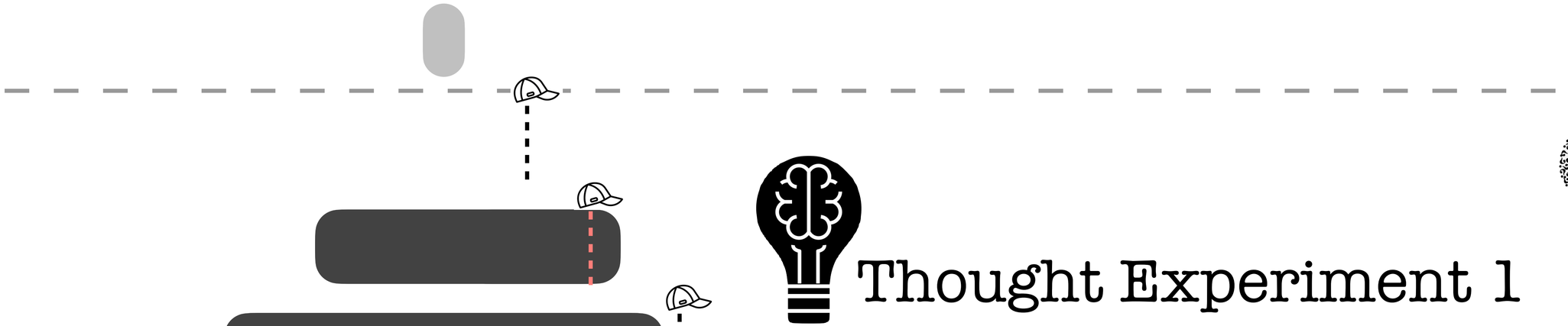
Thought Experiment 1

Problems with **full-level compaction**?

Cascading compactions!

Compaction **Granularity**

data moved per compaction



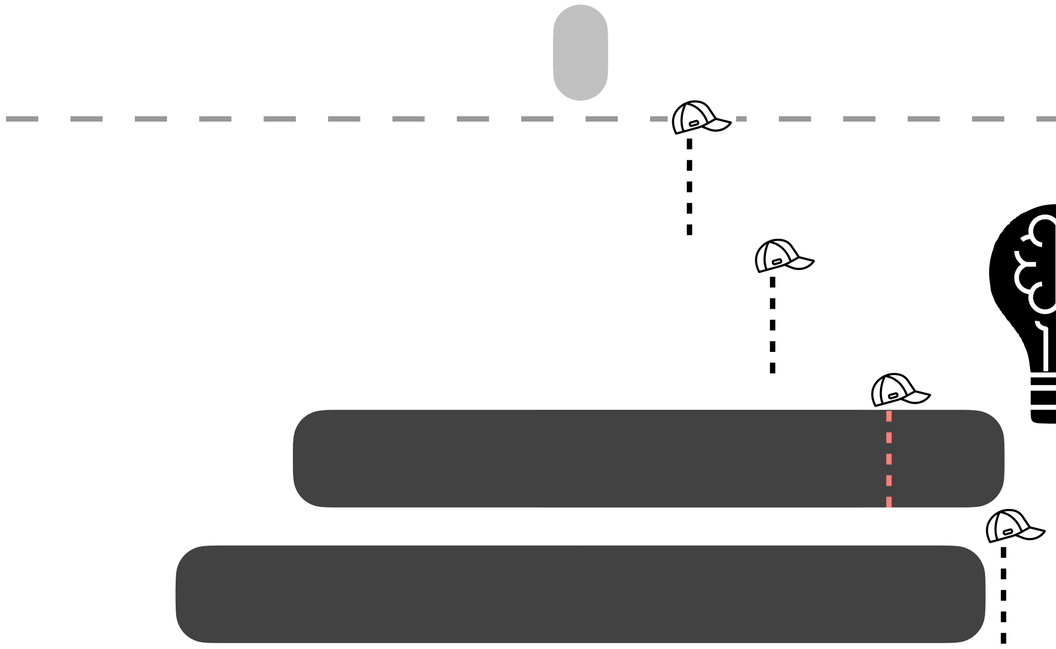
Thought Experiment 1

Problems with **full-level compaction**?

Cascading compactions!

Compaction **Granularity**

data moved per compaction



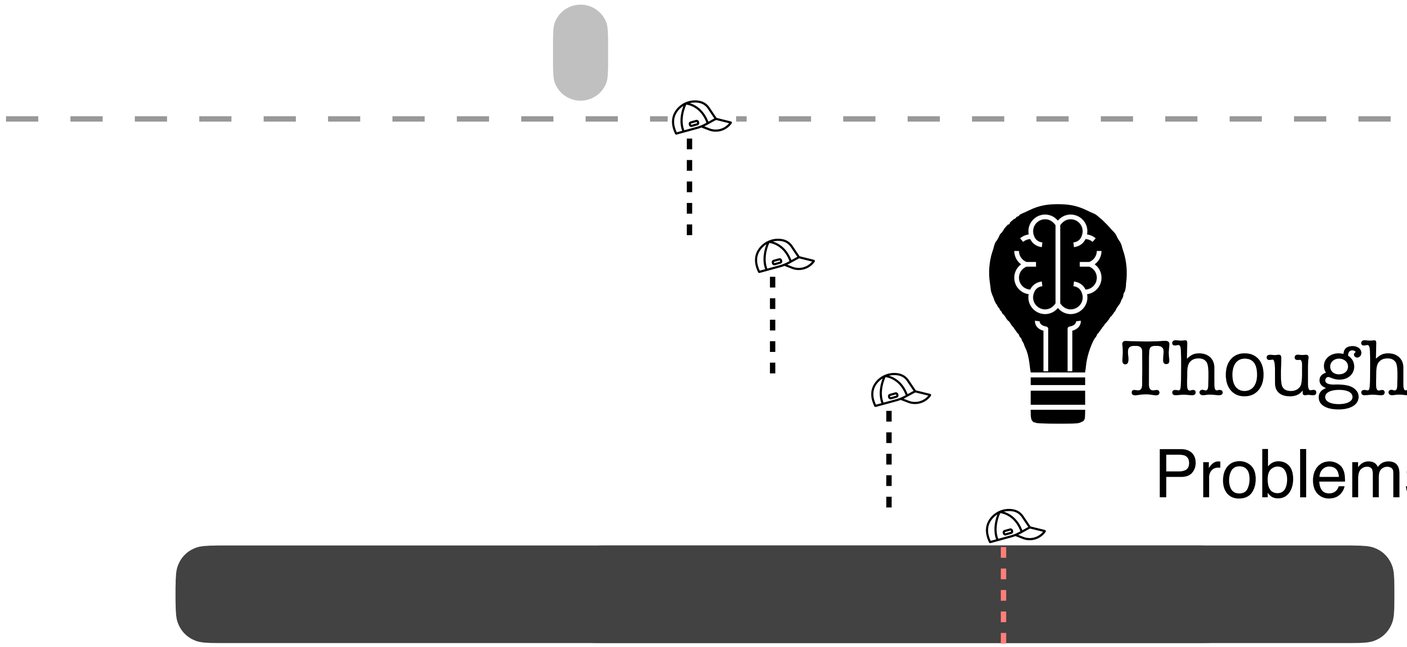
Thought Experiment 1

Problems with **full-level compaction**?

Cascading compactions!

Compaction **Granularity**

data moved per compaction



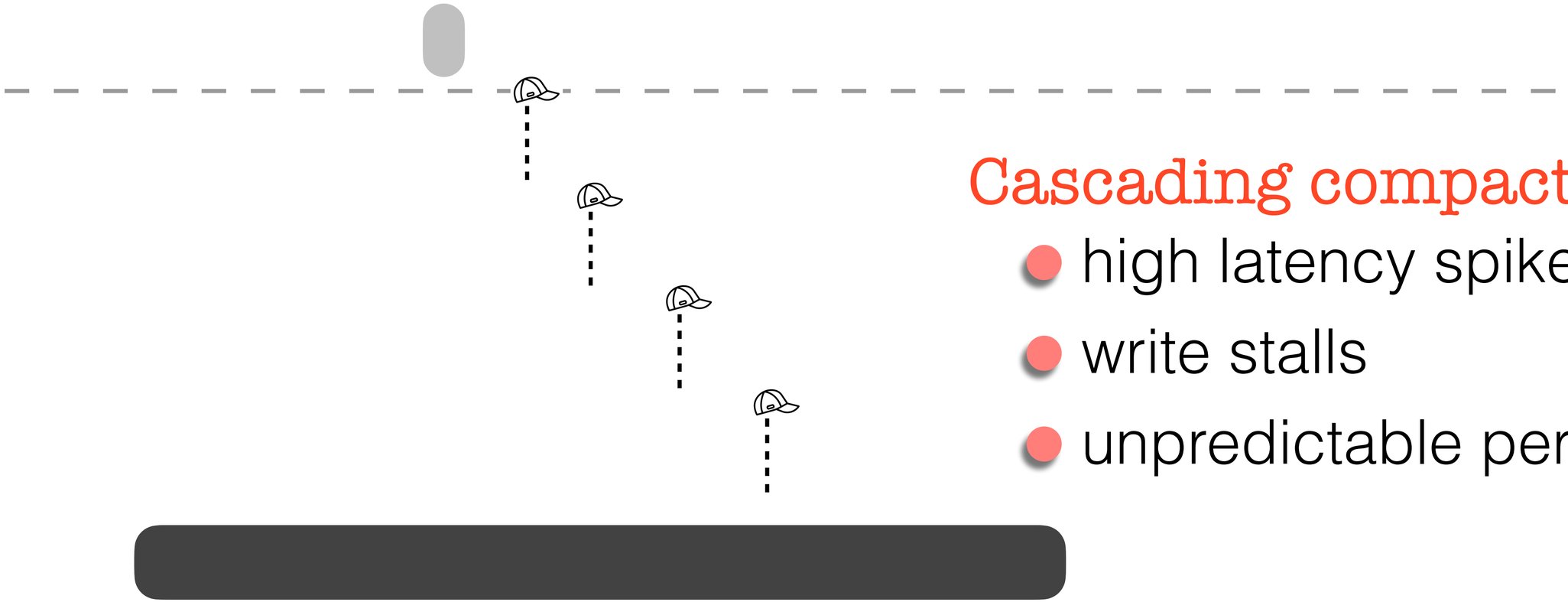
Thought Experiment 1

Problems with **full-level compaction**?

Cascading compactions!

Compaction **Granularity**

data moved per compaction



Cascading compactions!

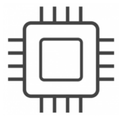
- high latency spikes
- write stalls
- unpredictable perf.

Compaction **Granularity**

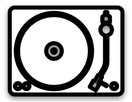
data moved per compaction

partial compaction

granularity: **files**



buffer



level 1

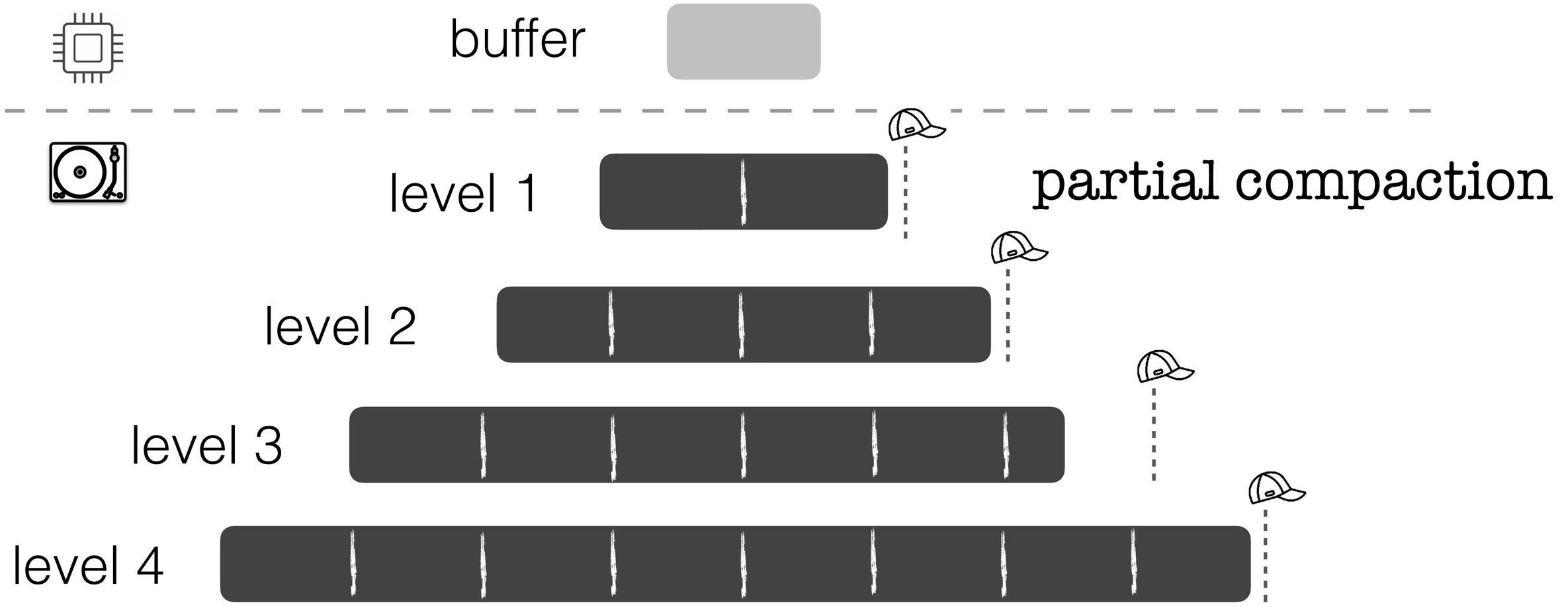
partial compaction

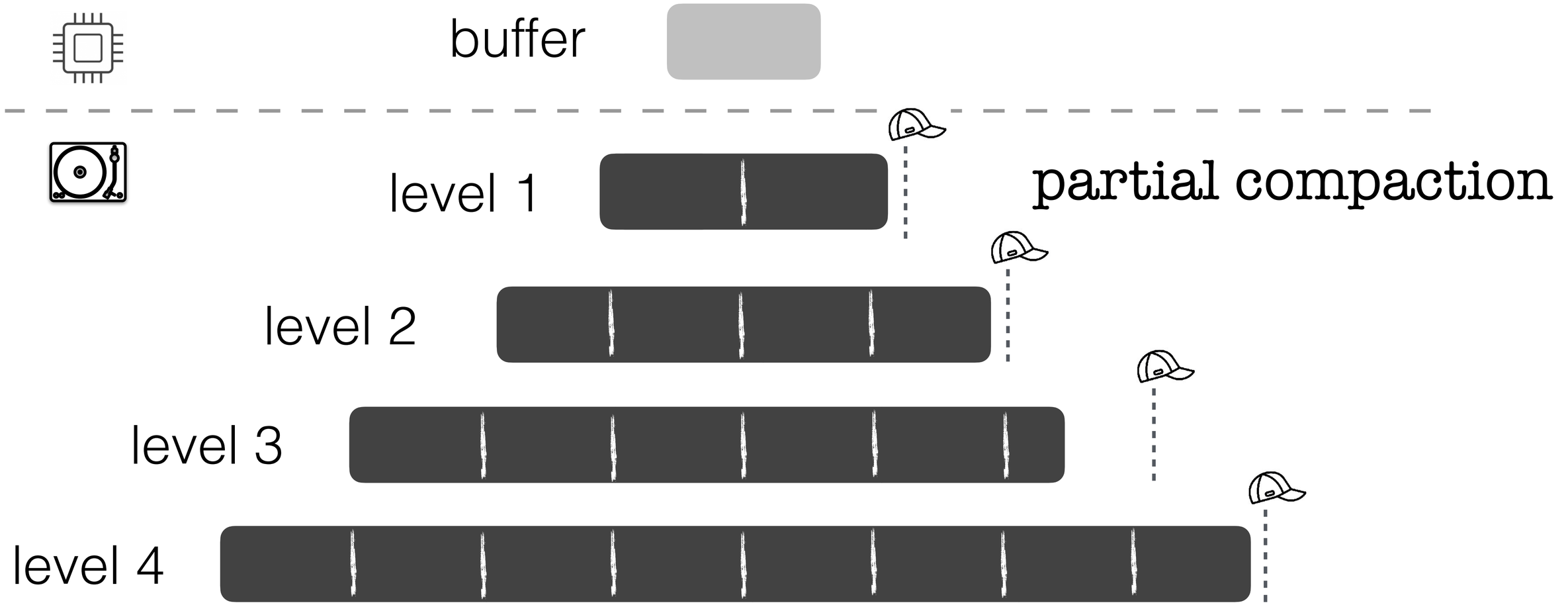
level 2

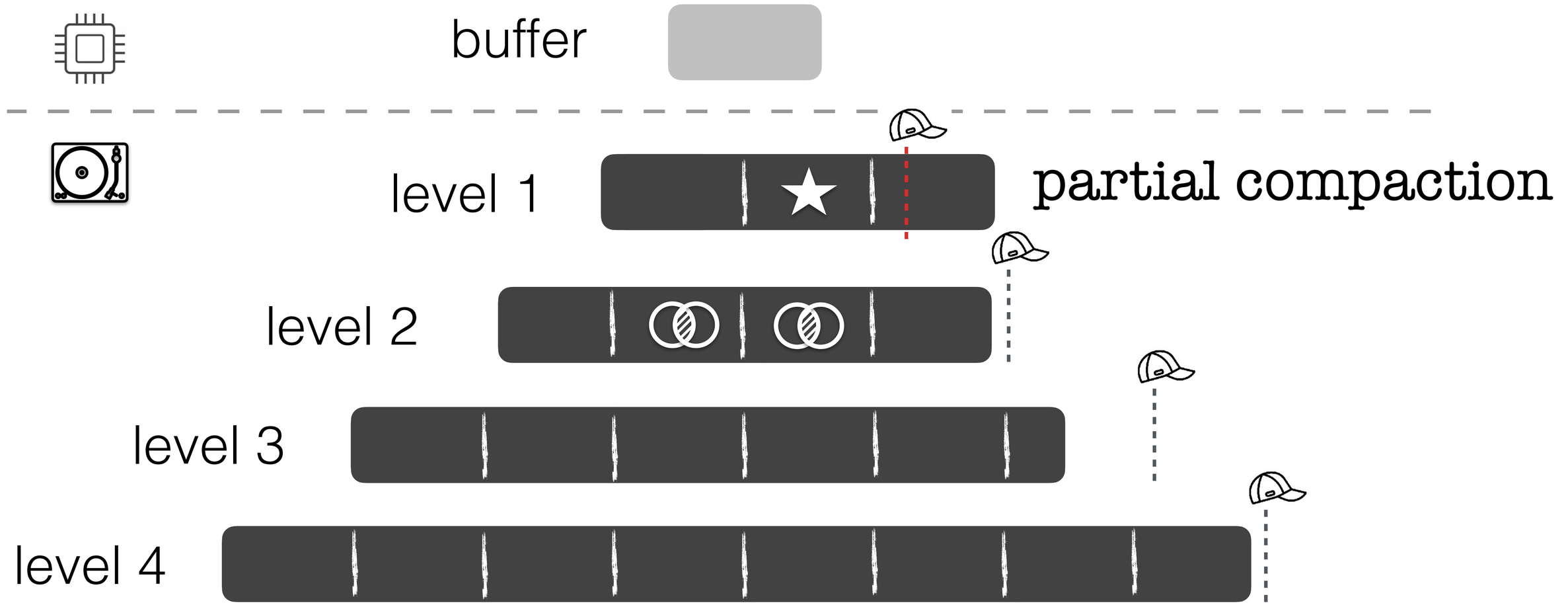
level 3

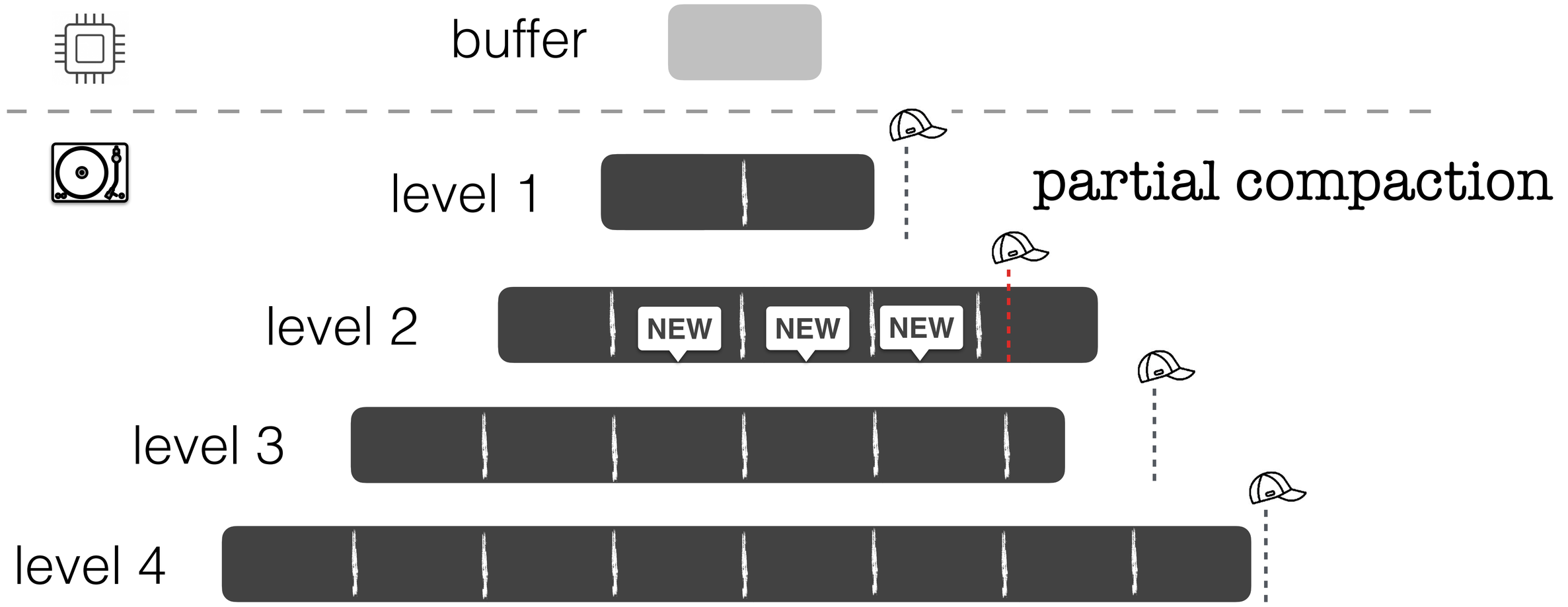
level 4

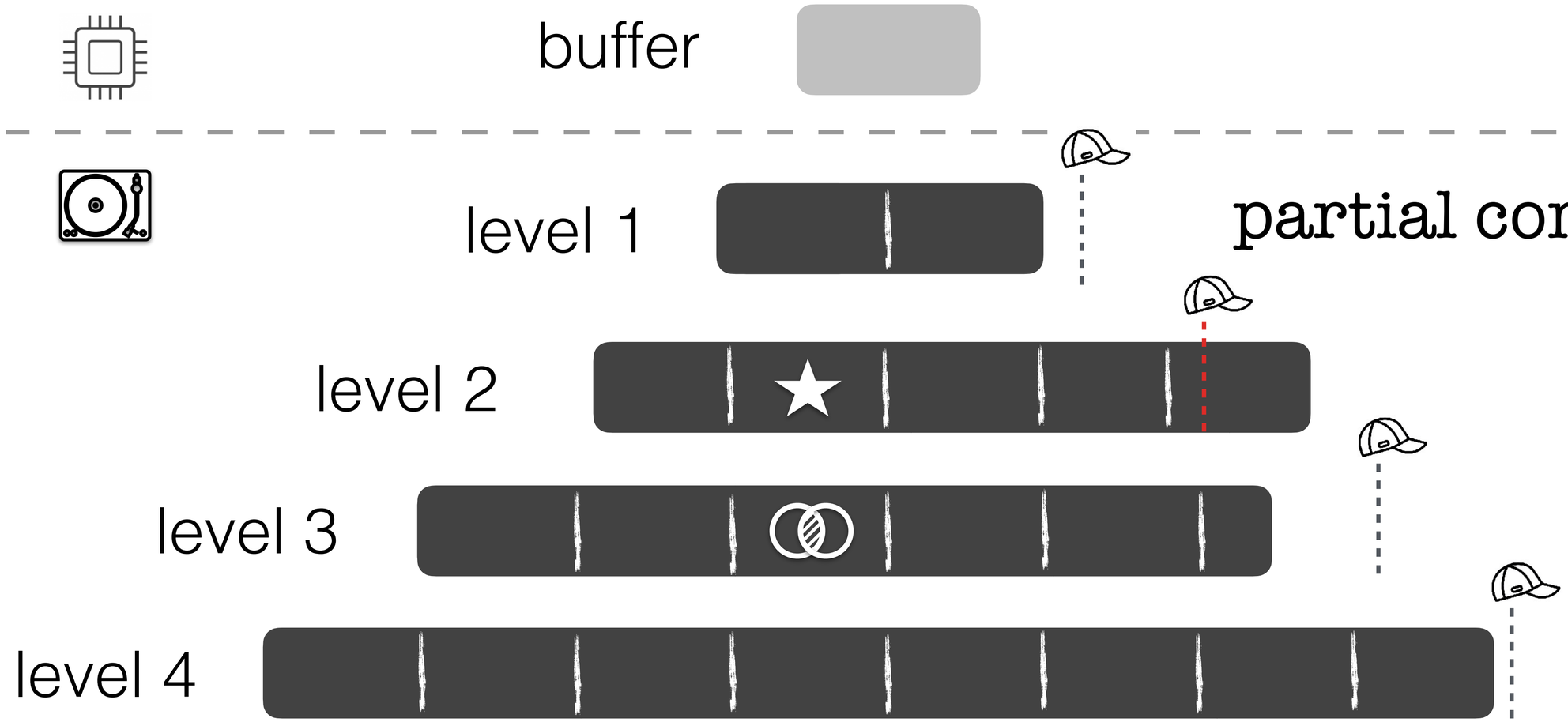
Brandeis UNIVERSITY

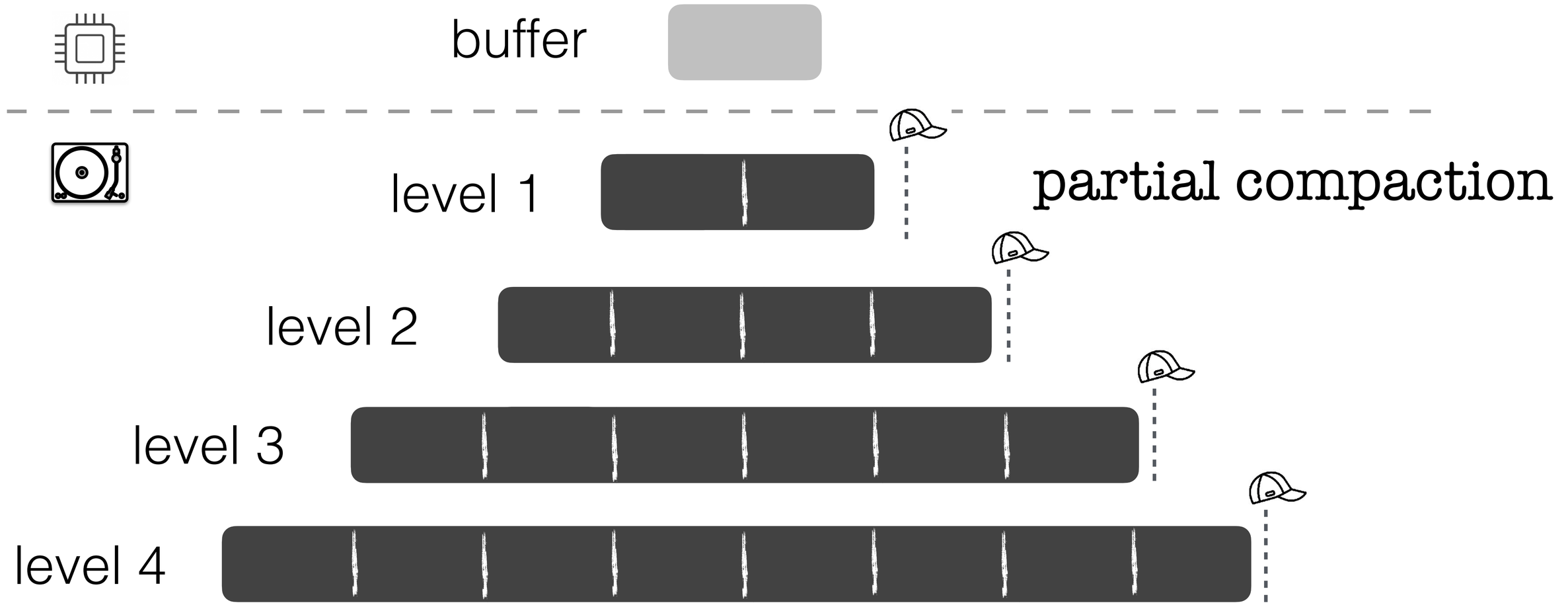

buffer
level 1
partial compaction
level 2
level 3
level 4


buffer
level 1
partial compaction
level 2
level 3
level 4


buffer
level 1
partial compaction
level 2
level 3
level 4

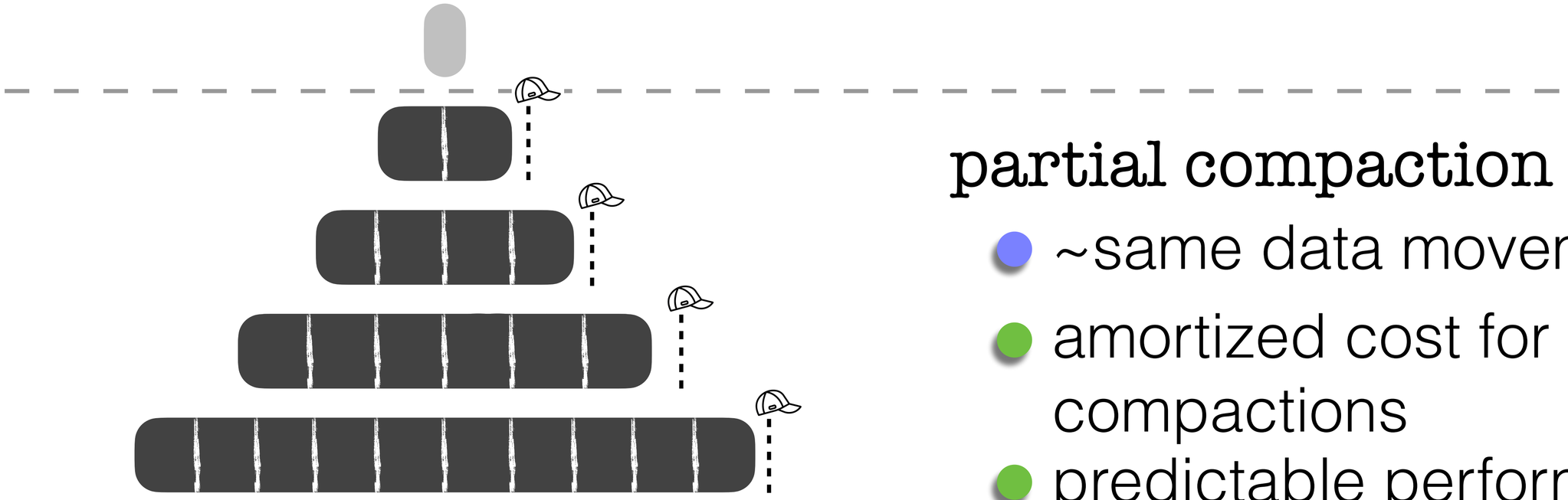

buffer
level 1
partial compaction
level 2
NEW
NEW
NEW
level 3
level 4


buffer
level 1
partial compaction
level 2
level 3
level 4


buffer
level 1
partial compaction
level 2
level 3
level 4

Compaction **Granularity**

data moved per compaction



files

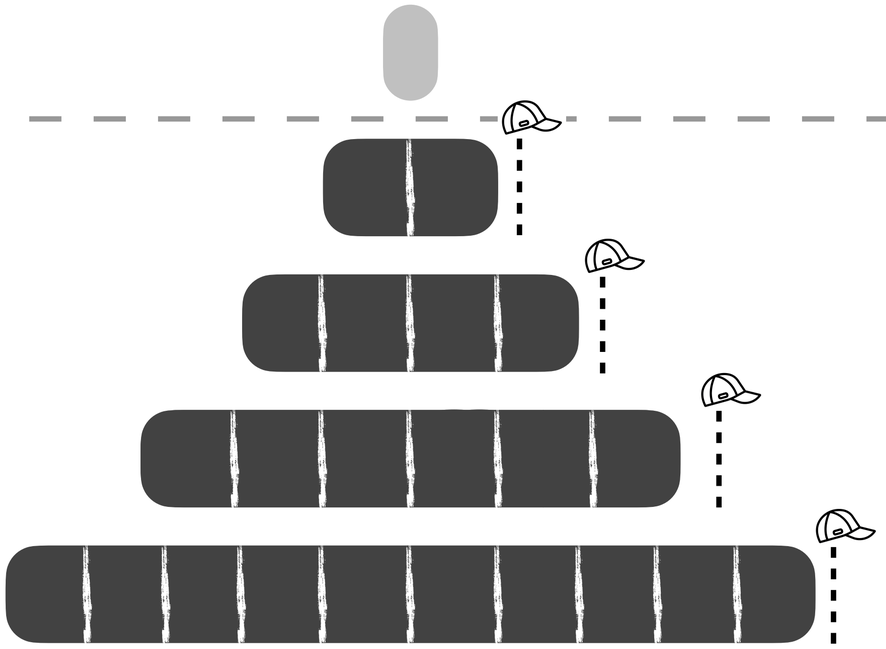
RocksDB

partial compaction

- ~same data movement
- amortized cost for compactions
- predictable performance

Compaction **Granularity**

data moved per compaction



files

RocksDB

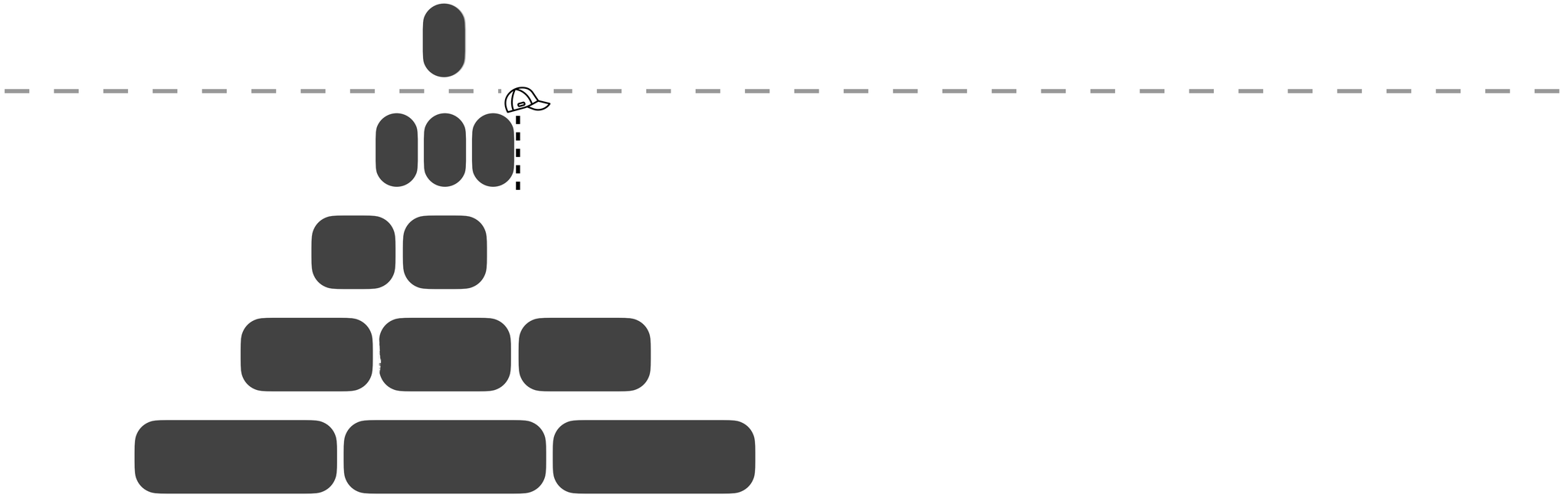
Thought Experiment 2

Limitations of **partial compactions**?

More #compactions triggered

Compaction **Granularity**

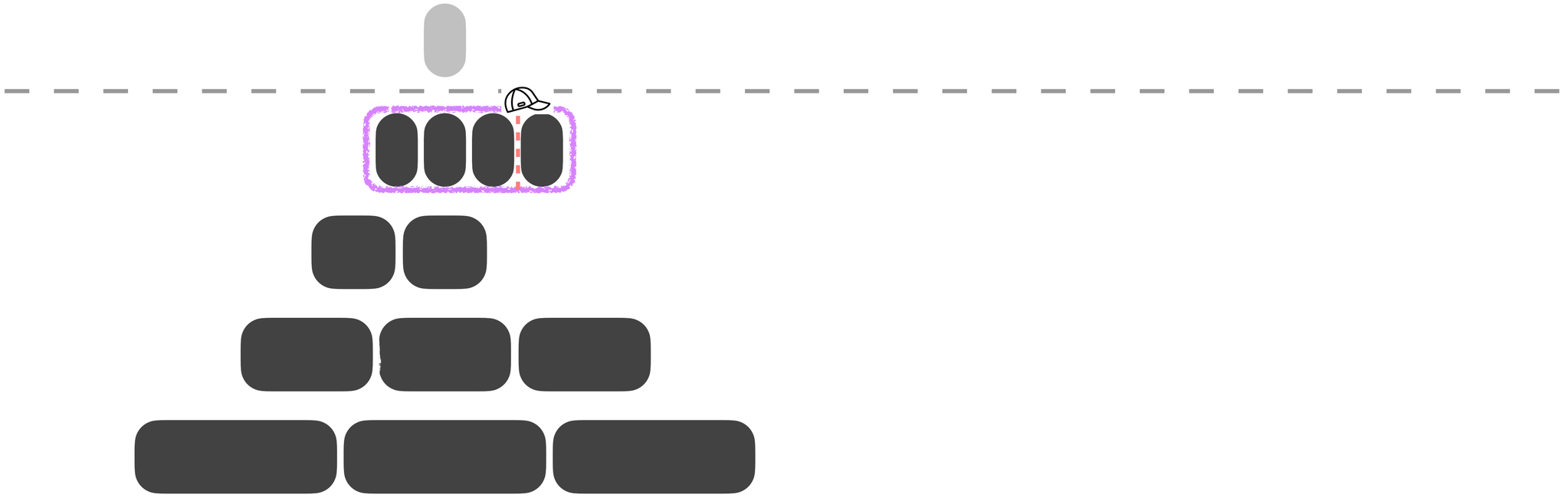
data moved per compaction



sorted runs in a level

Compaction **Granularity**

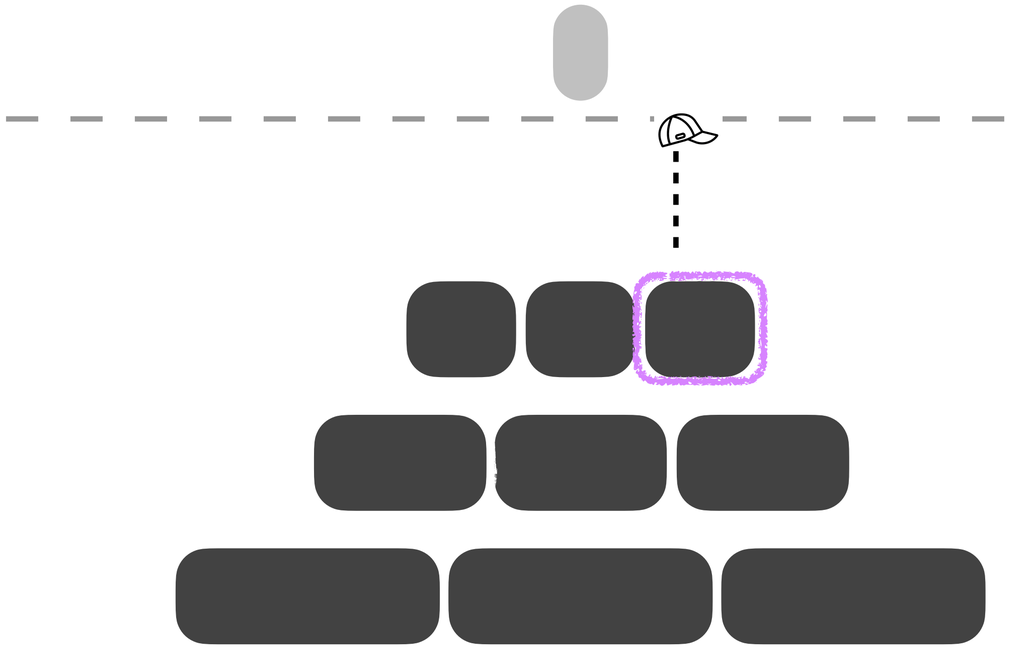
data moved per compaction



sorted runs in a level

2 Compaction **Granularity**

data moved per compaction



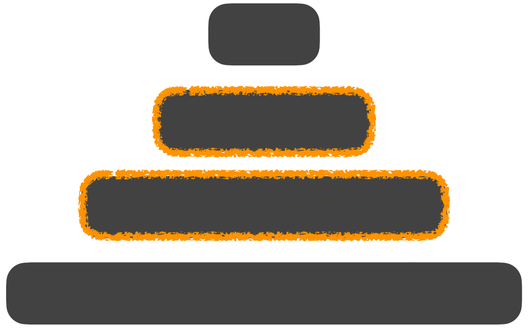
sorted runs in a level

tiered compaction

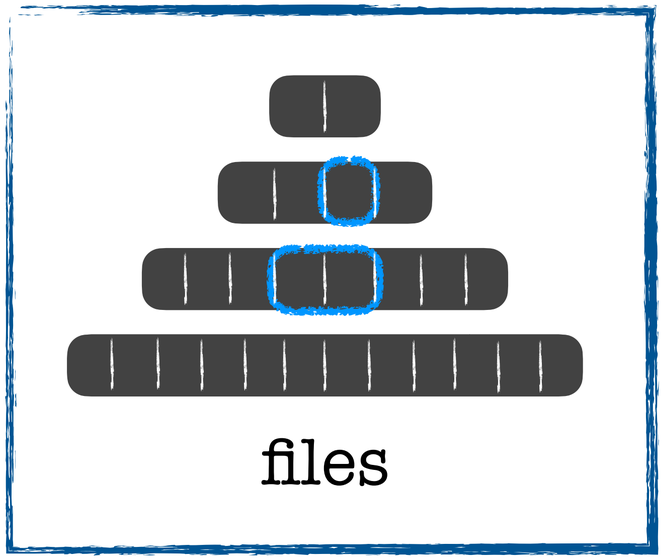
- triggered infrequently
- reduced space amplification

Compaction **Granularity**

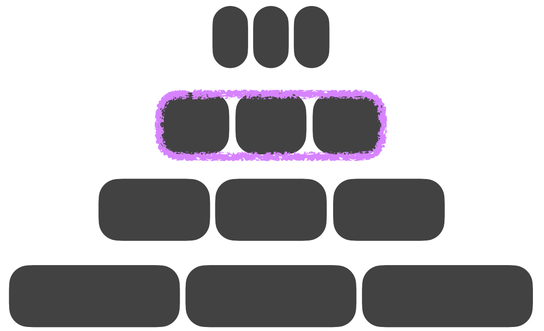
data moved per compaction



levels



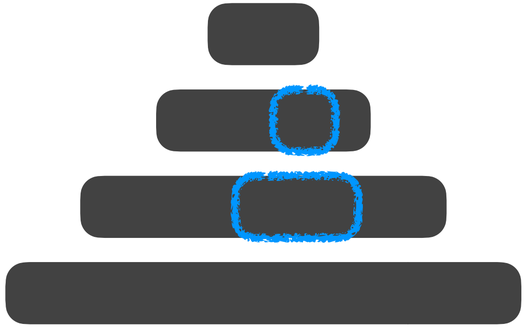
files



sorted runs in a level

Data Movement Policy

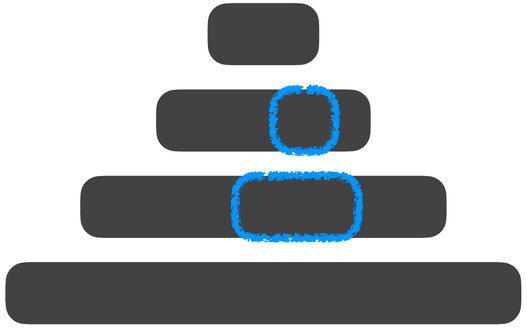
which data to compact



files

Data Movement Policy

which data to compact



files

round-robin !

minimum **overlap with parent** level WA

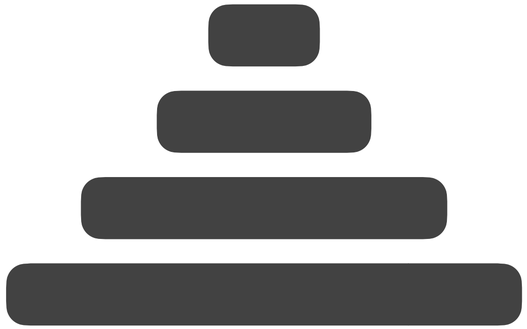
file with most **tombstones** SA

coldest file PQ

Compaction **Trigger**

invoking the compaction routine

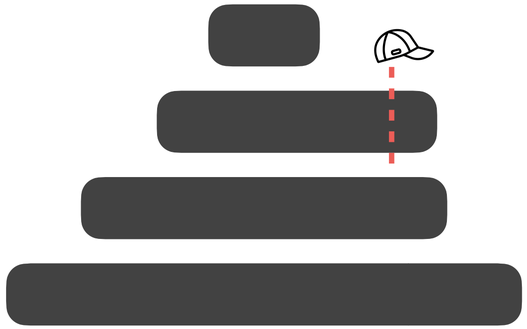
level **saturation**



Compaction **Trigger**

invoking the compaction routine

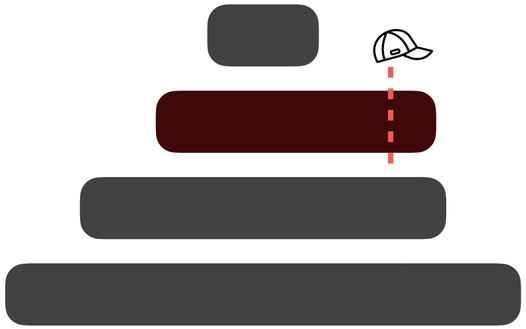
level **saturation**



Compaction **Trigger**

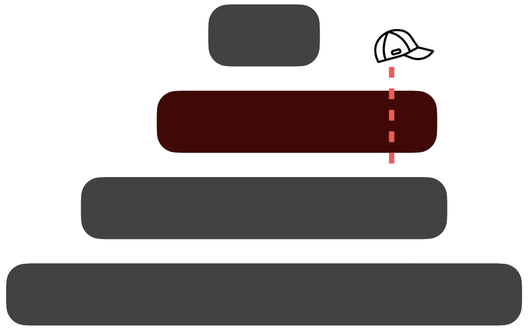
invoking the compaction routine

level **saturation**



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

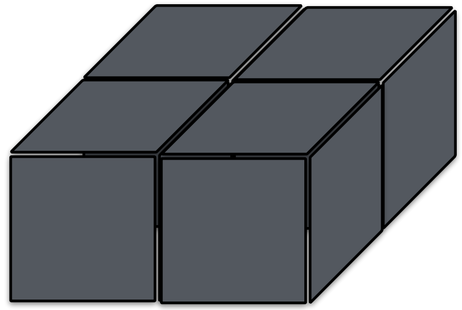
Compaction Trigger

Data Layout

Compaction Granularity

Data Movement Policy

Compaction Trigger

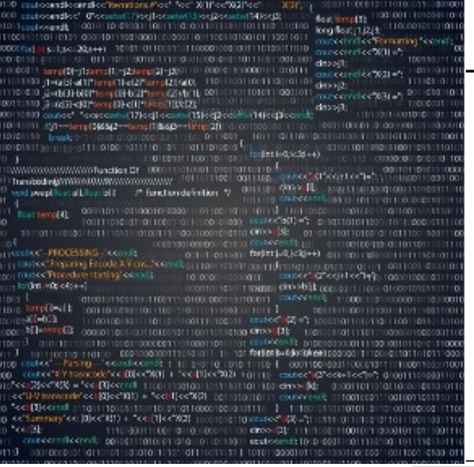


***Any* Compaction Algorithm**

SarkarVLDB21

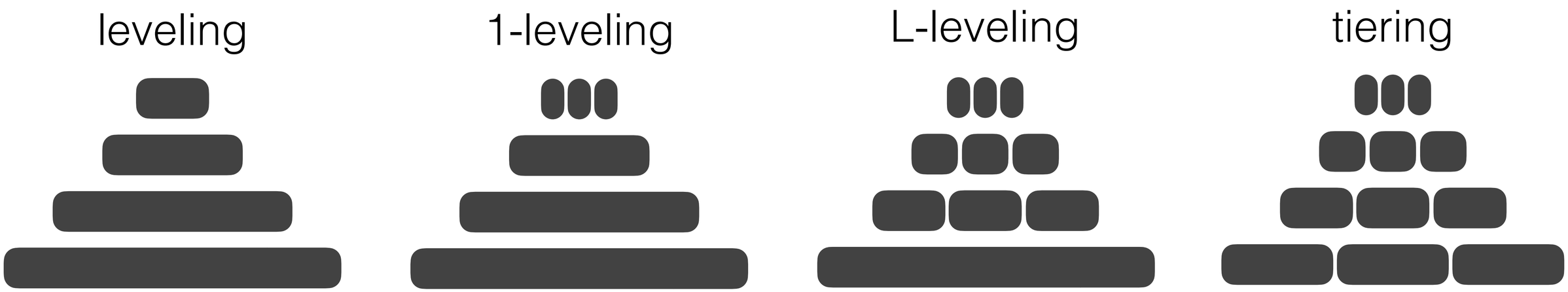
Database	Data layout	Compaction Trigger					Compaction Granularity				Data Movement Policy							
		Level saturation	#Sorted runs	File staleness	Space amp.	Tombstone-TTL	Level	Sorted run	File (single)	File (multiple)	Round-robin	Least overlap (+1)	Least overlap (+2)	Coldest file	Oldest file	Tombstone 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		✓					✓										✓

Storage Layer **Design Continuum**

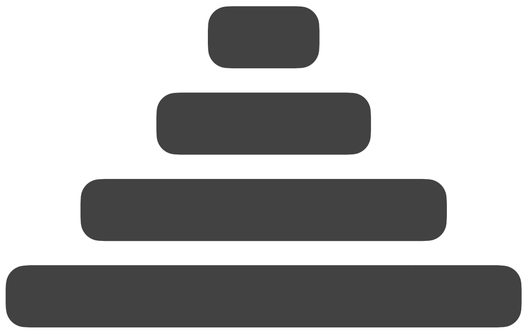


Any design can be defined by the tuple-set: $(\mathtt{T}, \mathtt{i})$

Storage Layer **Design Continuum**

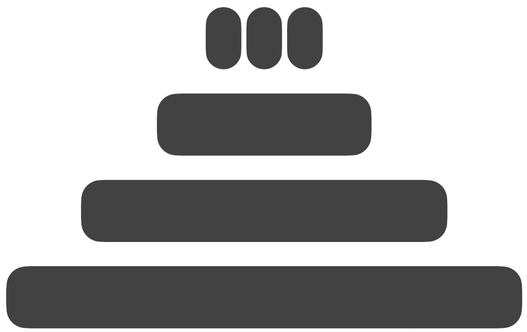
leveling

(T, 0)



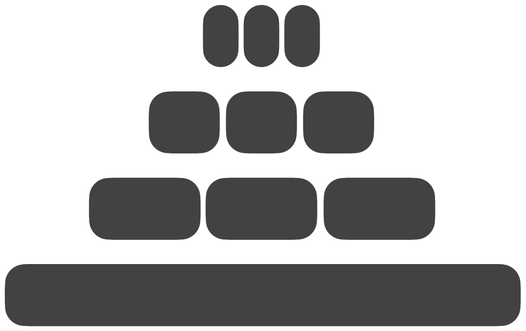
1-leveling

(T, 1)



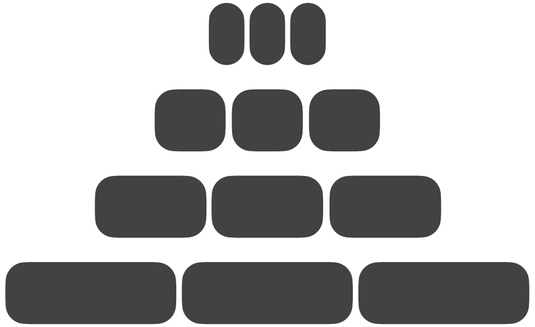
L-leveling

(T, L)



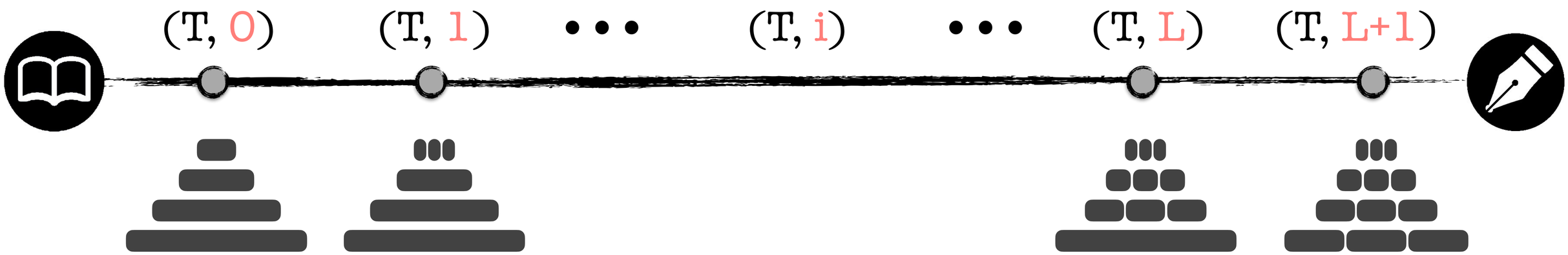
tiering

(T, L+1)

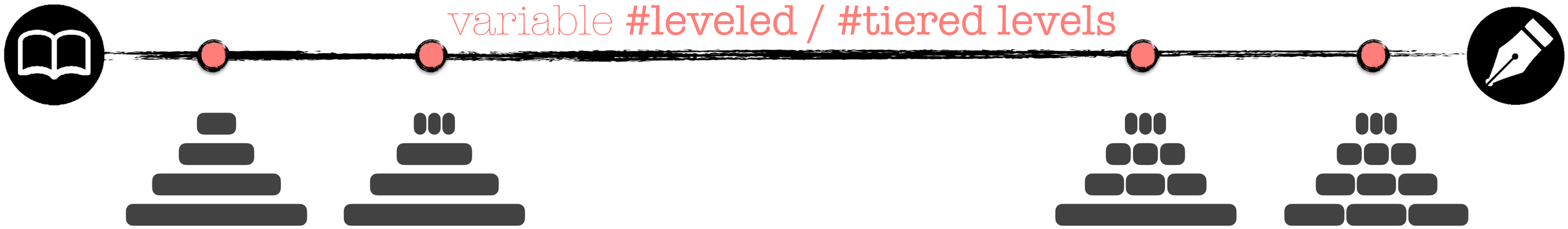


Any design can be defined by the tuple-set: (T, i)

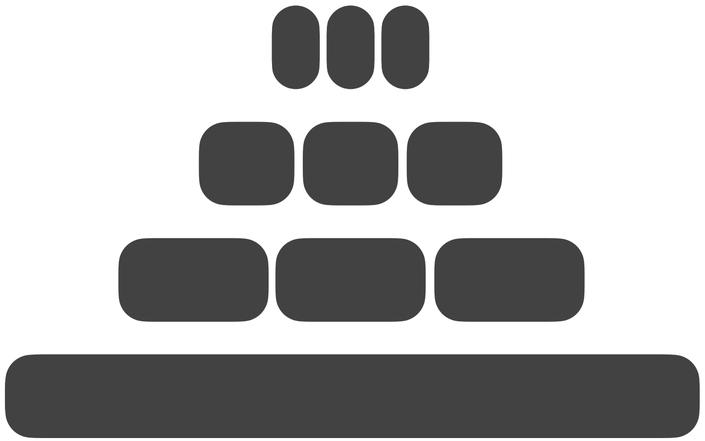
Storage Layer **Design Continuum**



Storage Layer **Design Continuum**



Storage Layer **Design Continuum**



size ratio

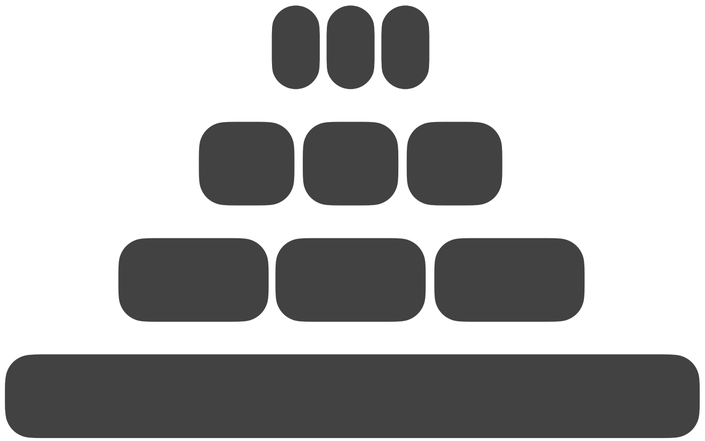
T

T

T

T

Storage Layer **Design Continuum**



size ratio

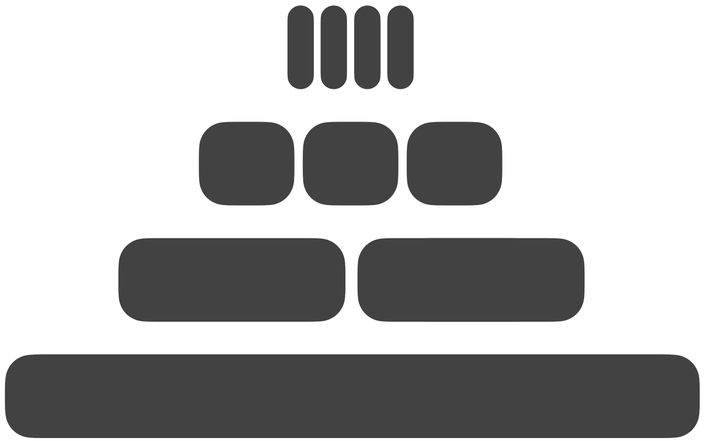
T

T

T

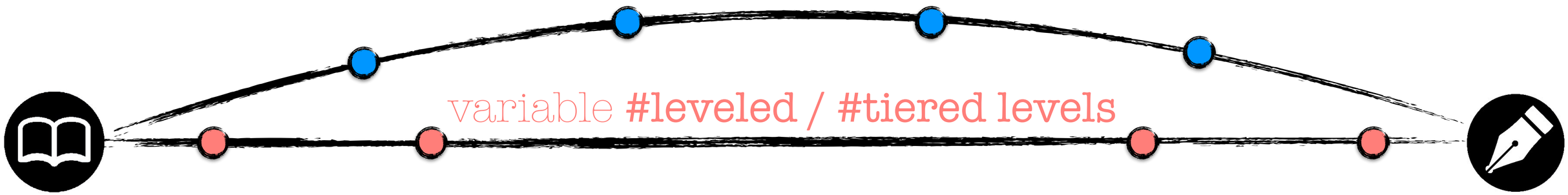
T

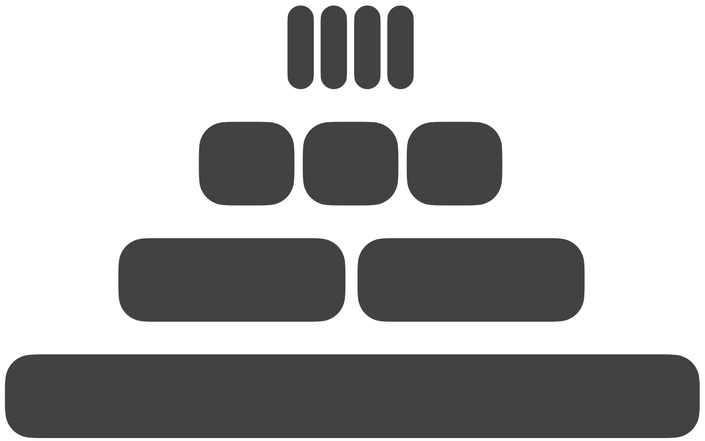
Storage Layer **Design Continuum**



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

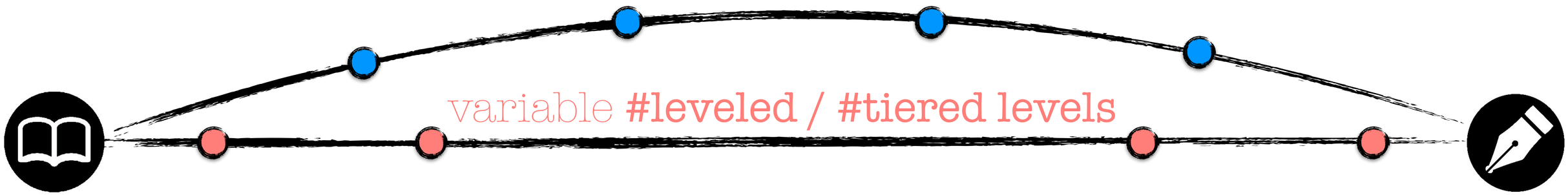
Storage Layer **Design Continuum**

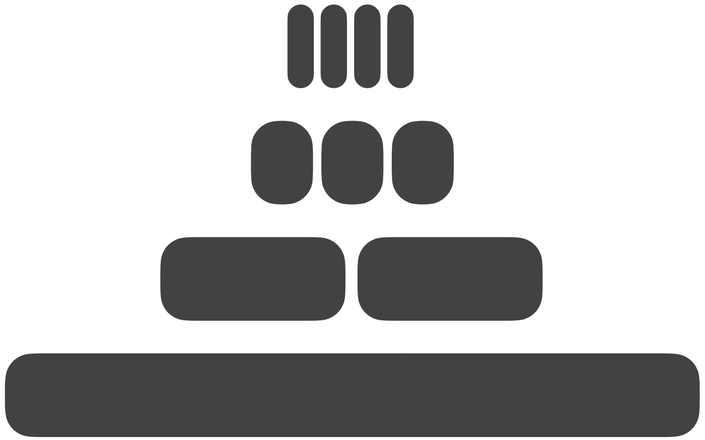




size ratio	#runs
T	4
T	3
T	2
T	1

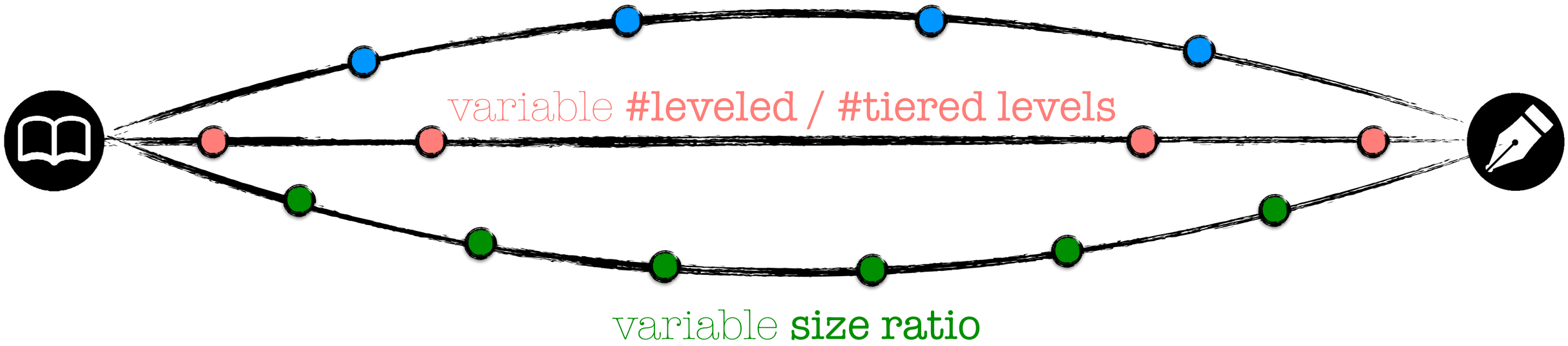
Storage Layer **Design Continuum**





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

Storage Layer **Design Continuum**



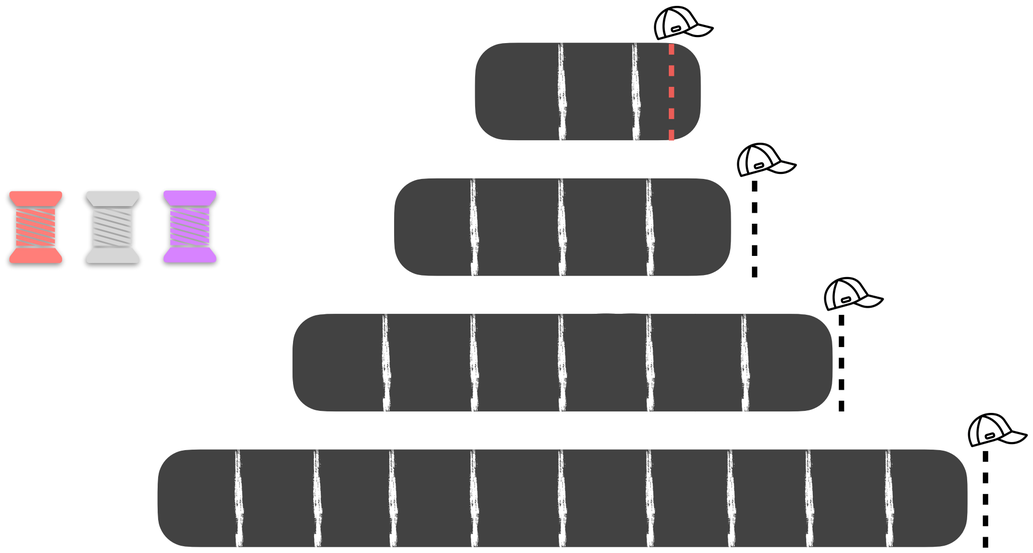
The LSM storage layer design continuum

Optimizing **Compactions**

Background Compactions

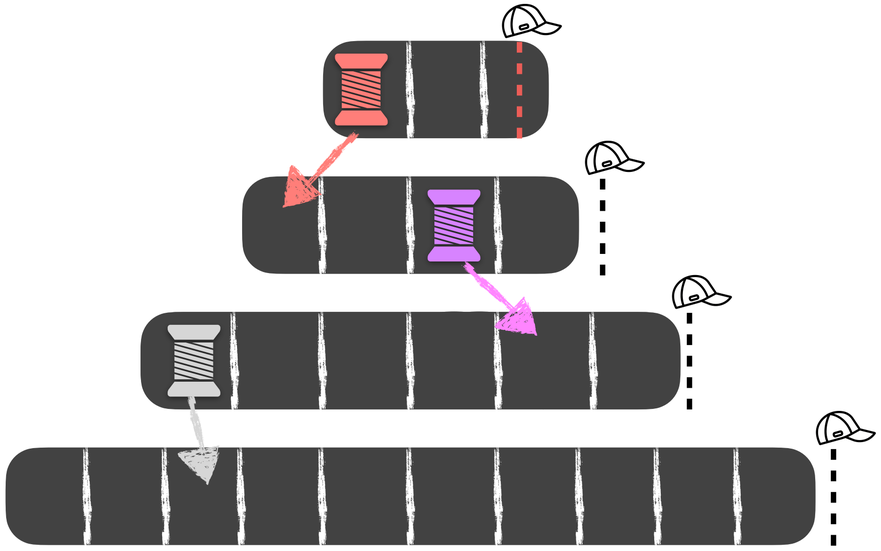
Optimizing **Compactions**

Background Compactions



Optimizing **Compactions**

Background Compactions



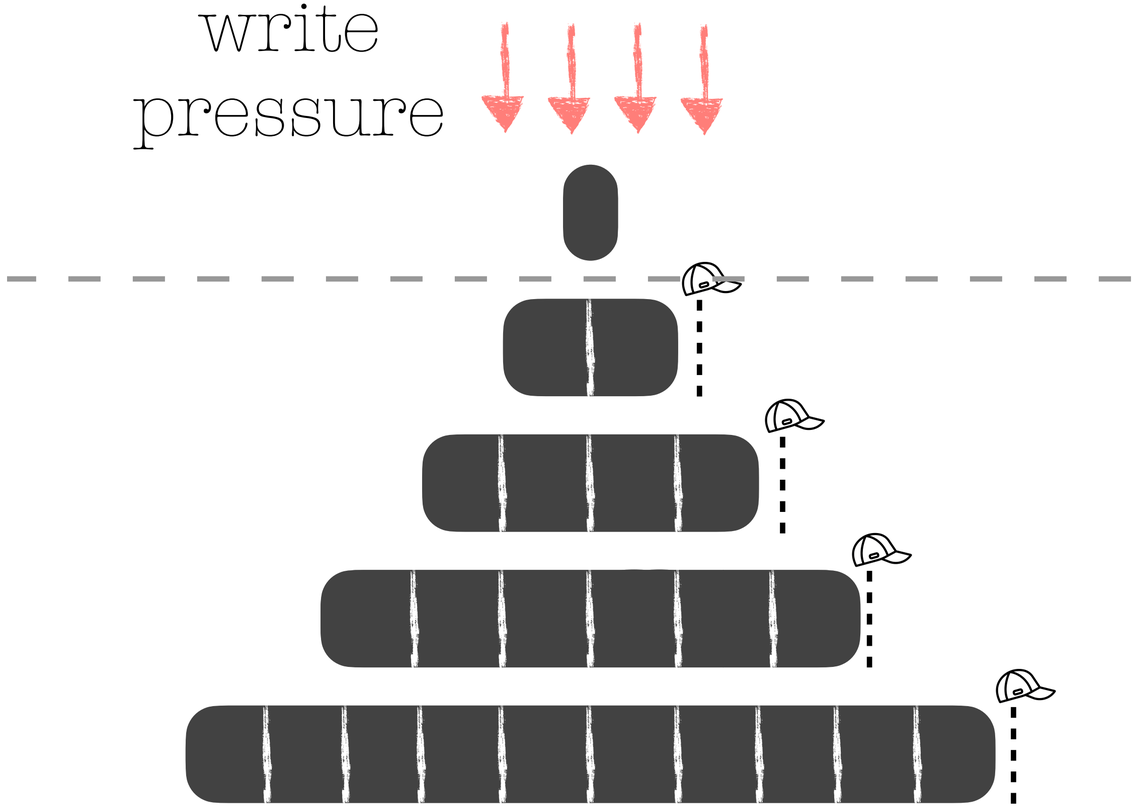
- non-blocking reads/writes
- improves write throughput

Optimizing **Compactions**

Background Compactions

Compaction Priority

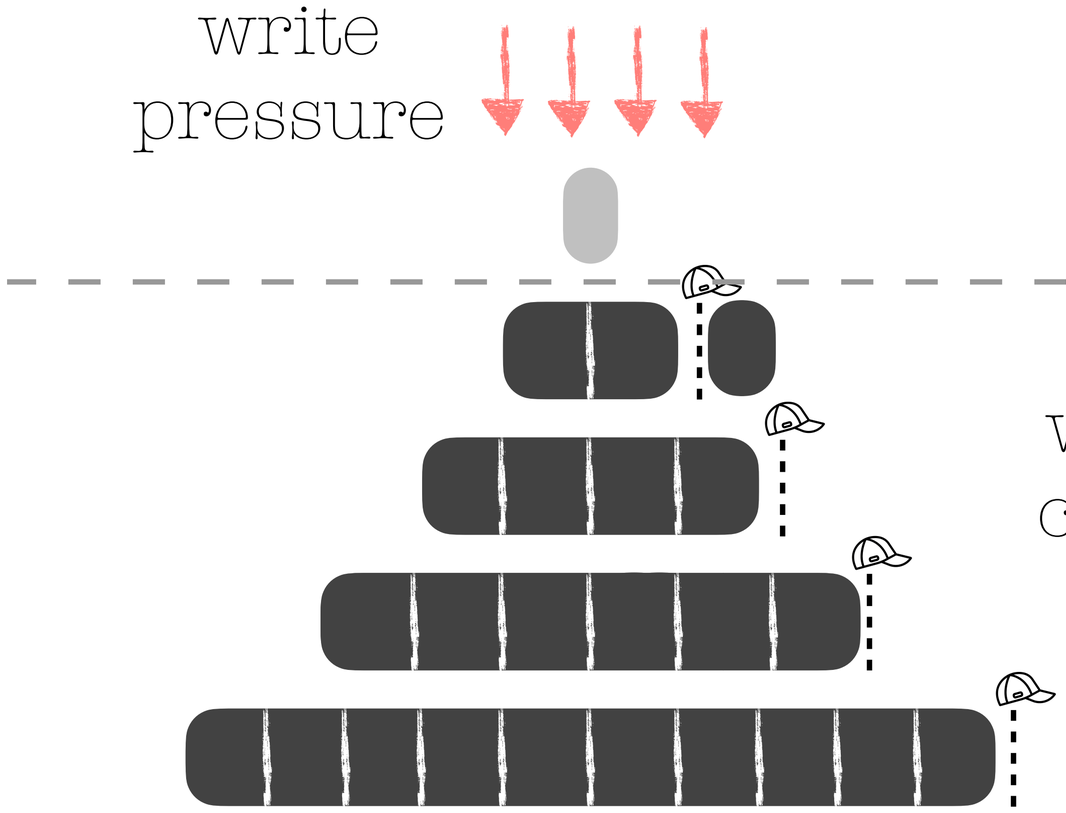
Optimizing **Compactions**



Background Compactions

Compaction Priority

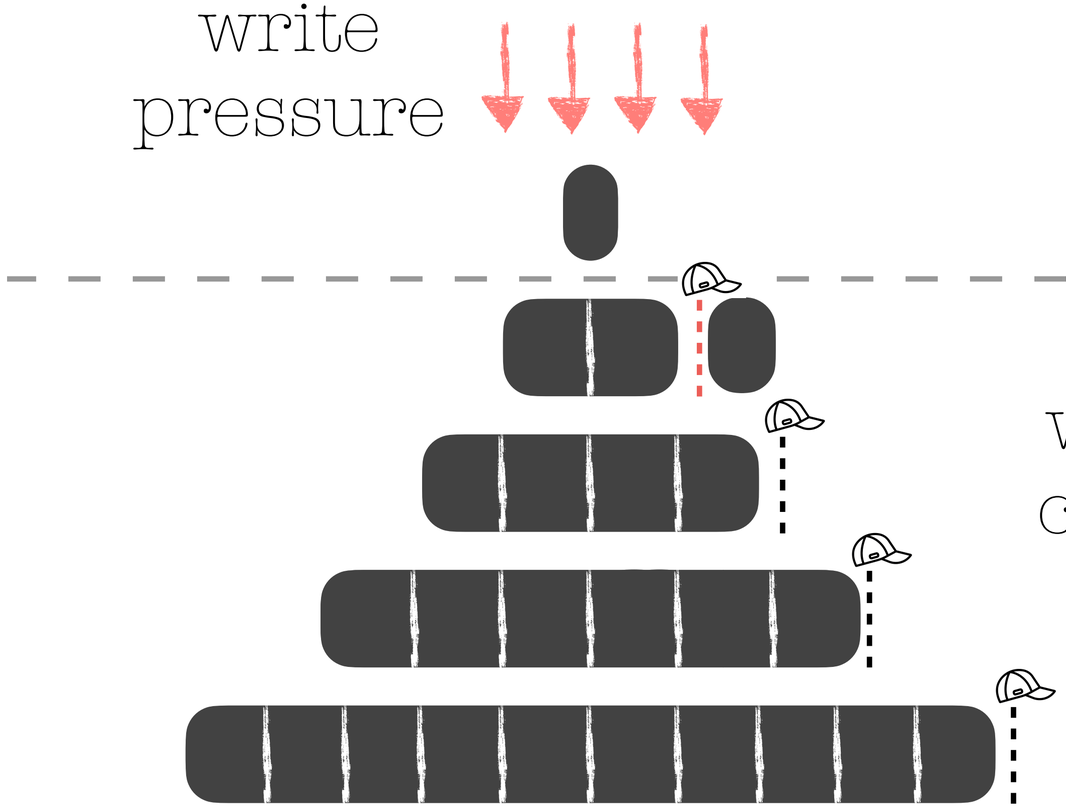
Optimizing **Compactions**



Background Compactions

Compaction Priority

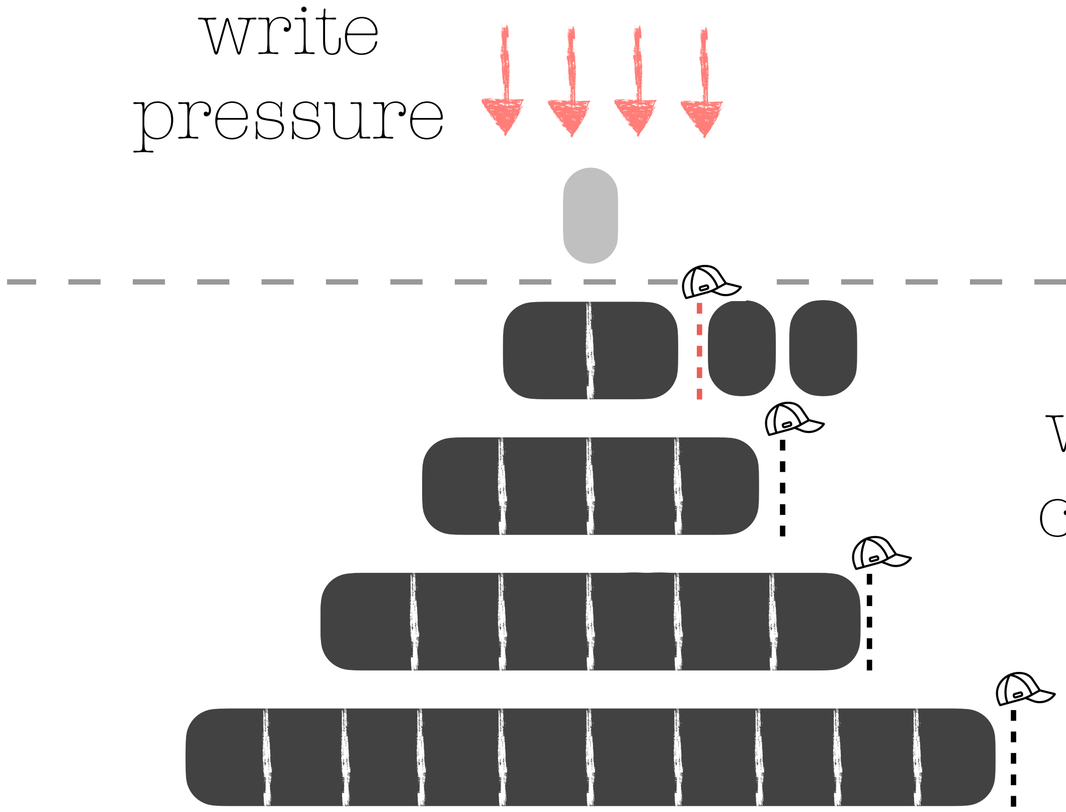
Optimizing **Compactions**



Background Compactions

Compaction Priority

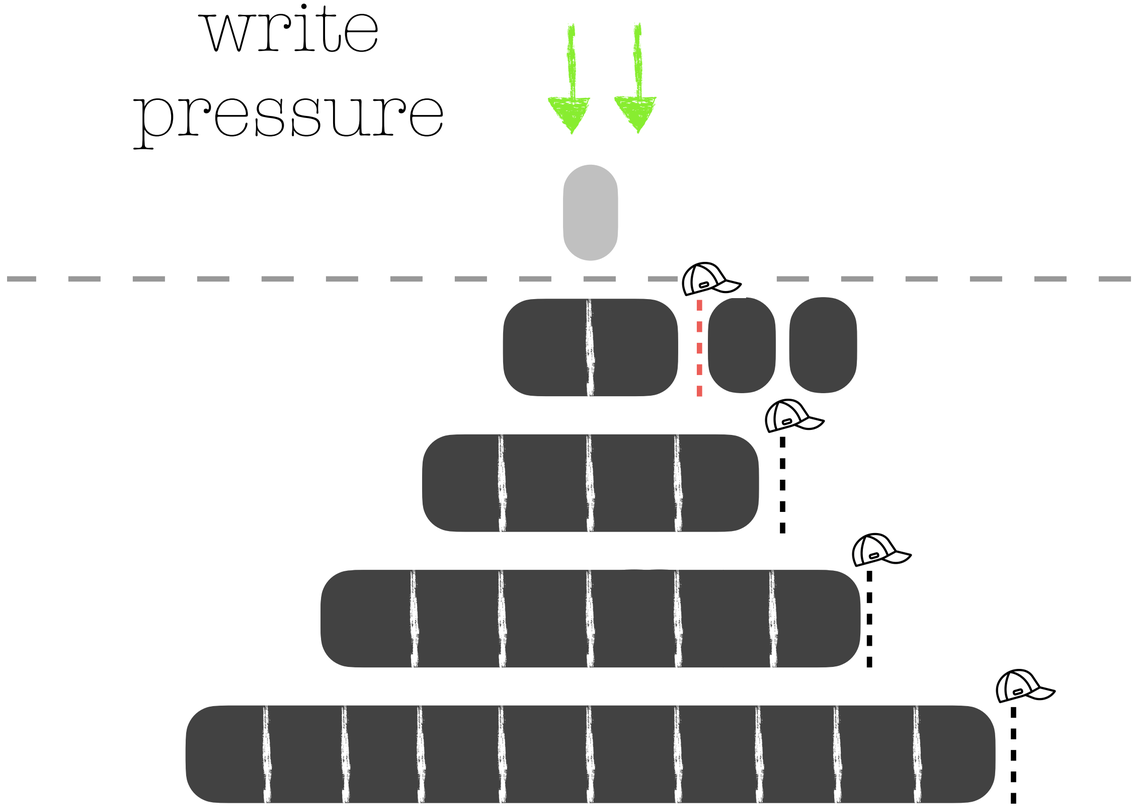
Optimizing **Compactions**



Background Compactions

Compaction Priority

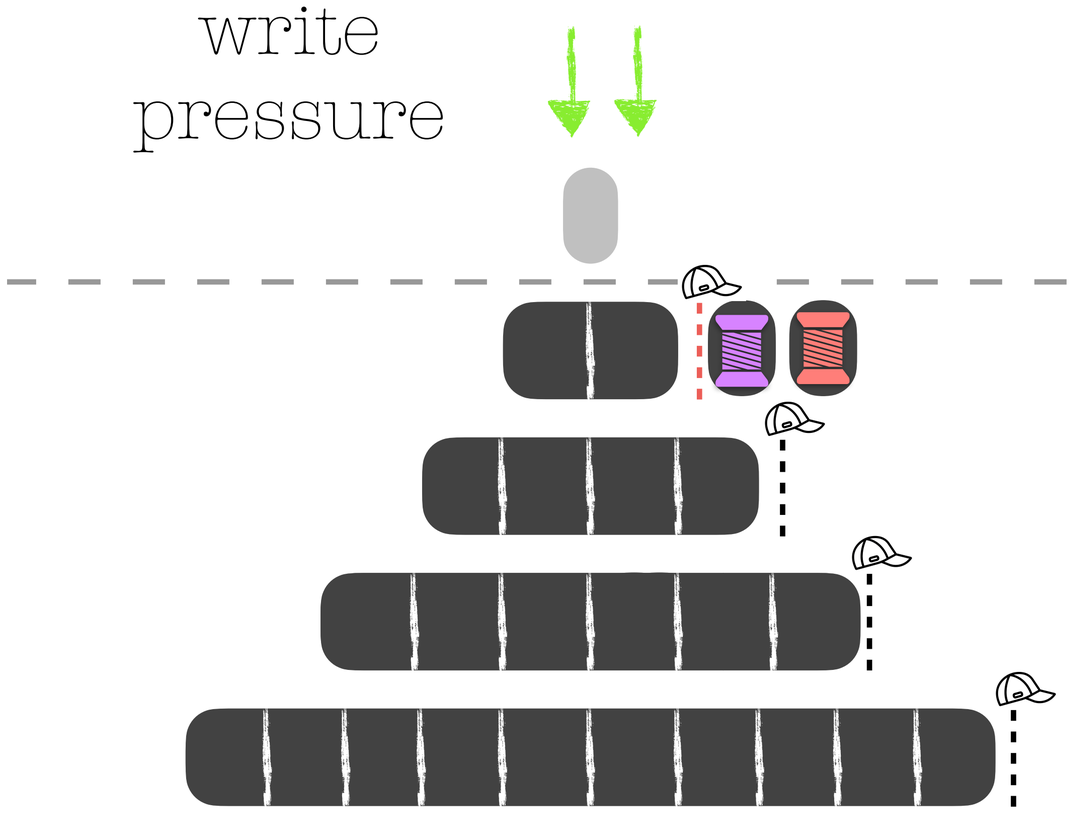
Optimizing **Compactions**



Background Compactions

Compaction Priority

Optimizing **Compactions**



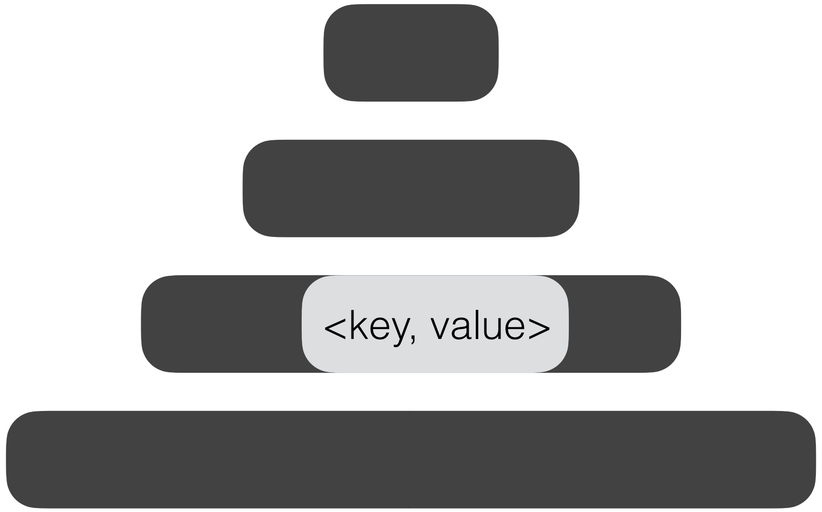
Background Compactions

Compaction Priority

- sustain heavy write bursts
- tree becomes out of shape

Data Placement Variations

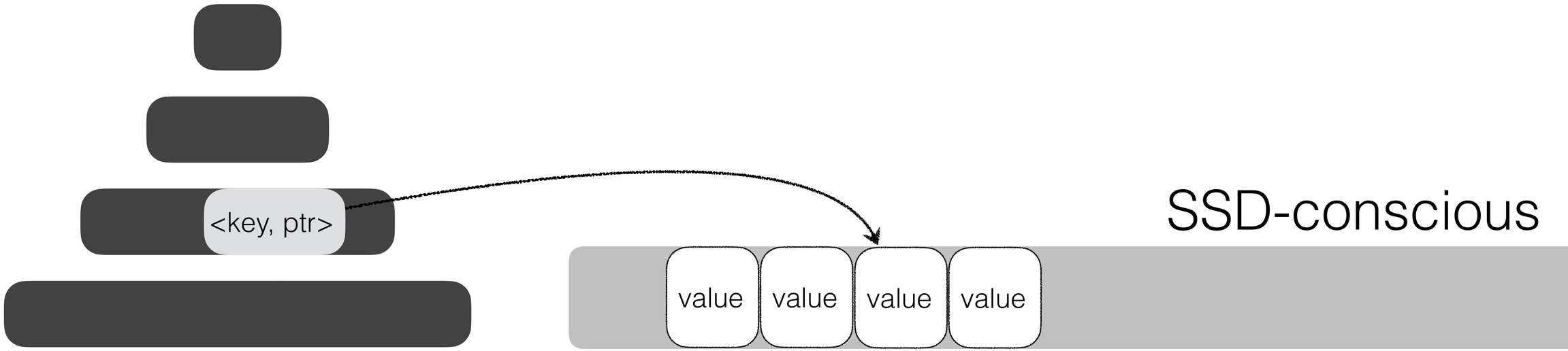
Data Placement Variations



key-value separation

LuFAST16

Data Placement Variations



key-value separation

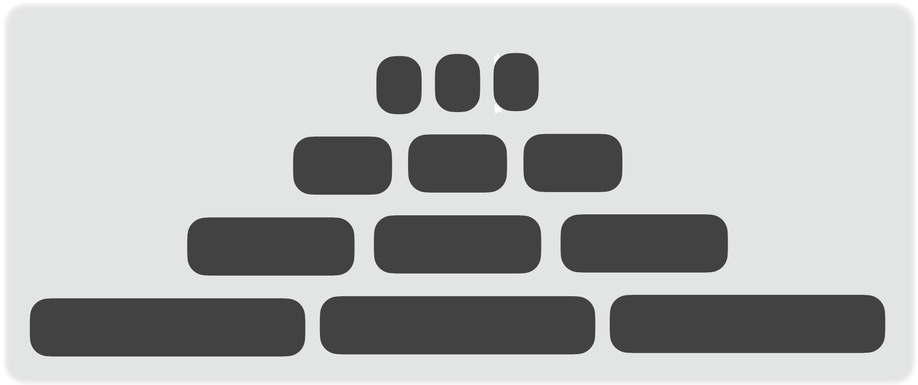
LuFAST16

- reduced write amplification
- better read performance

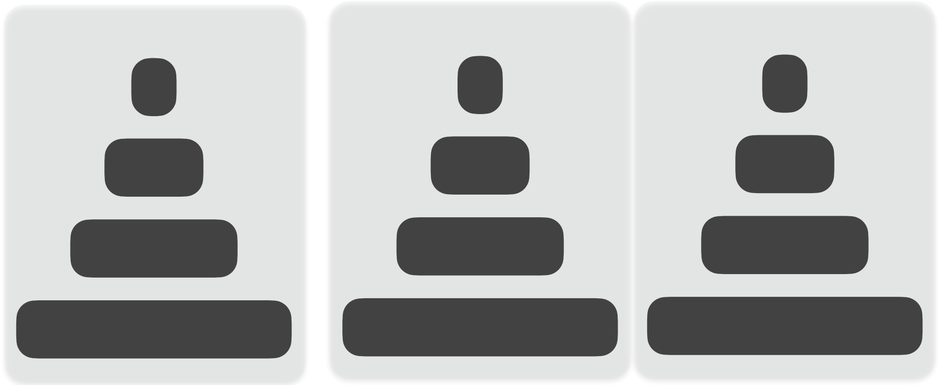
Data Placement Variations

partitioning / sharding

Data Placement Variations



storage



storage-1 storage-2 storage-3

partitioning RajuSOSP17

sharding HuangSIGMOD21

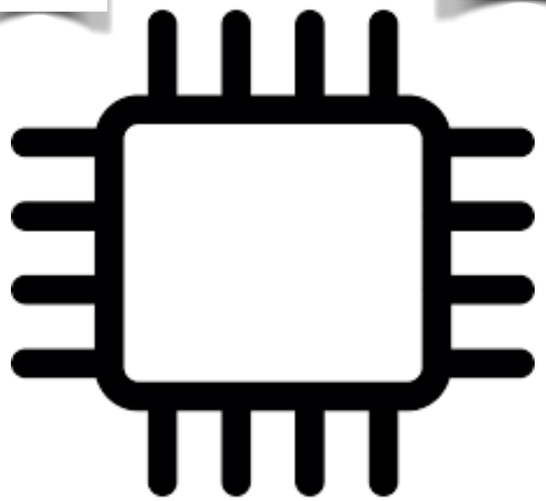
- improved ingestion throughput
- reduced write amplification

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

Next time in COSI 167A

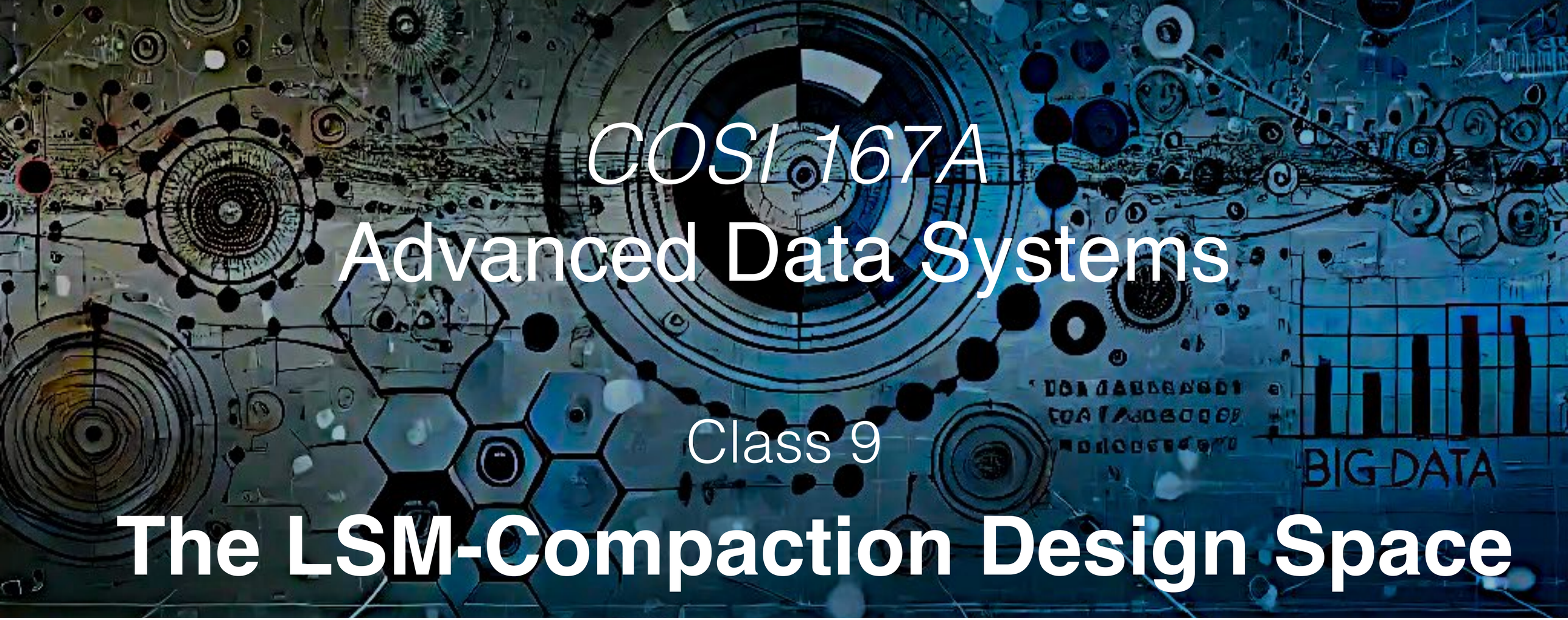
More on LSMs

robust LSM **tuning**

[P] "Endure: A Robust Tuning Paradigm for LSM Trees Under Workload Uncertainty", *VLDB*, 2022

REVIEW PAPER 1

[B] "CliffGuard: A Principled Framework for Finding Robust Database Designs", *SIGMOD*, 2015



COSI 167A
Advanced Data Systems

Class 9
The LSM-Compaction Design Space

Prof. Subhadeep Sarkar

Brandeis UNIVERSITY

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

lab