COSI 167A
Advanced Data Systems

Class 5
Introduction to LSM-Trees

Prof. Subhadeep Sarkar

Brandeis UNIVERSITY

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

lab

Class **logistics**

and administrivia

The **second technical question** is now available on the class website (due **before the class** on **Sep 17**).

Deadline is at **12:45 PM** for all **technical questions** and **reviews**!

Start working on **Project 1** (deadline **Sep 20**).

Today in COSI 167A

What's on the cards?

NoSQL & **key-value stores**

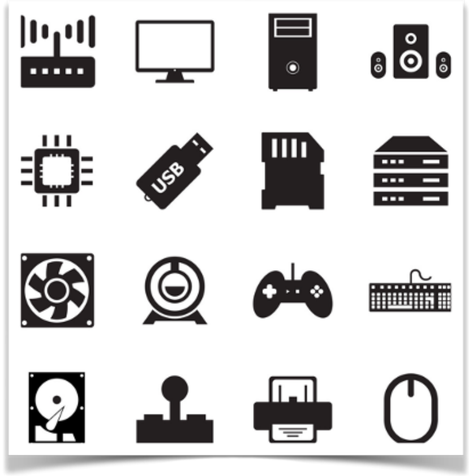
introduction to **LSM-trees**

The **evolving** landscape of **data systems**

Evolution in the world of data systems is FAST!

growing
data size

new
hardware



heterogeneous
applications

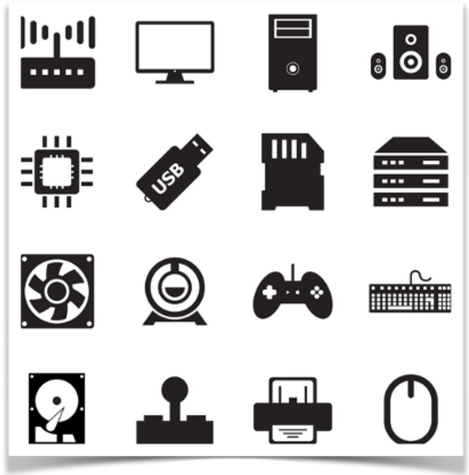
new **performance goals**

The **evolving** landscape of **data systems**

Evolution in the world of data systems is FAST!

growing
data size

new
hardware



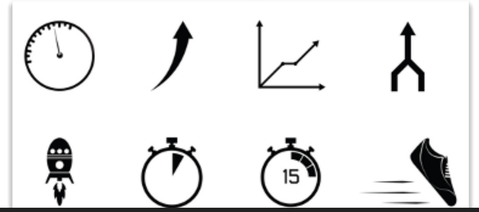
heterogeneous
applications

new **performance goals**

One size does not fit all!

The **evolving** landscape of **data systems**

Evolution in the world of data systems is FAST!



One size does not fit all!

growing
data size

new
hardware

heterogeneous
applications

new **performance goals**

The need for tailored data systems!

The birth of **NoSQL**

A 2000's child

Not only SQL

this is where we will spend our time!

steep competition to the **relational market**

since **early 2000's**

Relational vs. NoSQL

The new uprising!

relational systems

tables with **rows** & **columns**

well-defined **schema**

data **model fits data** rather than functionality

deduplication

$120B

NoSQL systems

unstructured documents or files

schema-less

data is stored in an **application-friendly way**

possible **duplication**

$60B

based on a table from http://readwrite.com

Classification of NoSQL systems

The different types

1. Key-value stores
   stores data as **key-value pairs**
2. Document stores
   stores data in **document format** (JSON, XML, YAML, etc.)
3. Column stores
   stores each attribute in a **separate column**
4. Graph stores
   stores data in form of **edges and vertices**

Classification of NoSQL systems

The different types

Key-value stores

Document stores

Column stores

Graph stores

Key-value stores

The most popular type of NoSQL store

key-value pairs

How **general** are **key-value stores**?

The most popular type of NoSQL store

key-value pairs

Thought Experiment 1

Can we store **relational data** in KVstores?

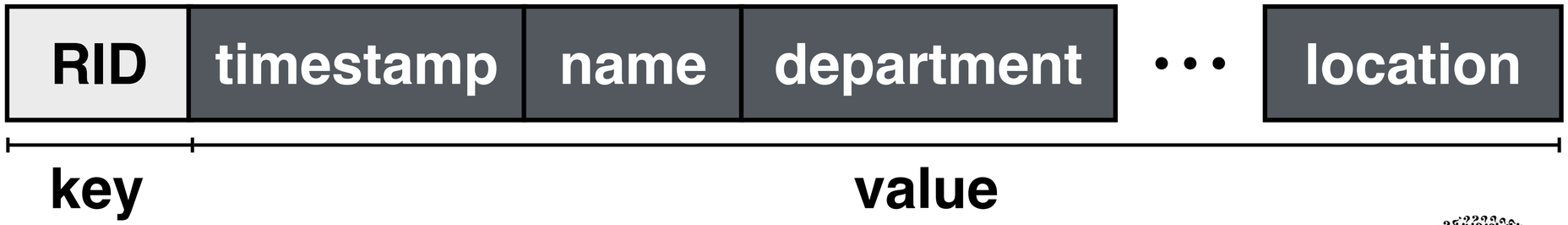
Yes!

Brandeis UNIVERSITY

How **general** are **key-value stores**?

The most popular type of NoSQL store

key-value pairs



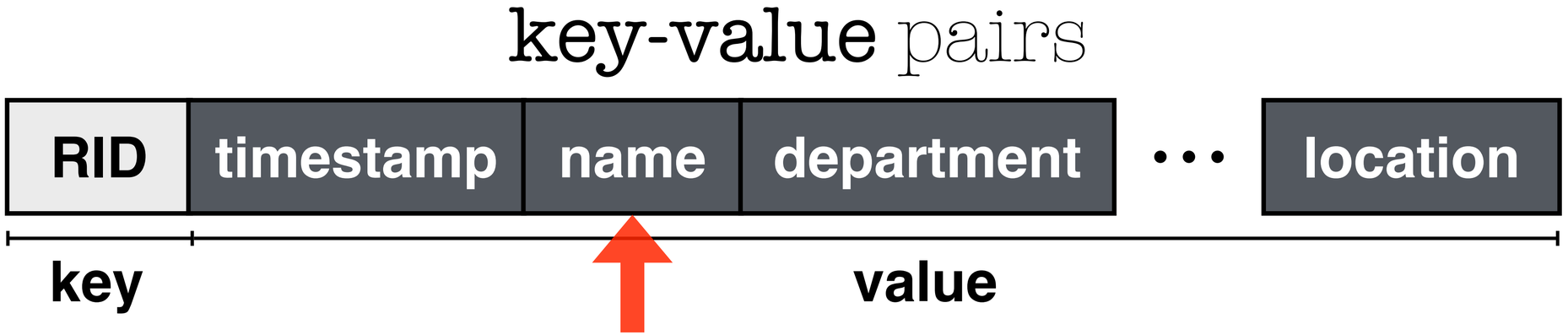
Thought Experiment 1

Can we store **relational data** in KVstores?

Yes!

How **general** are **key-value stores**?

The most popular type of NoSQL store



Thought Experiment 2

How to have an index on **name**?

index: { **name**, { **RID** } }

index: { **name**, { **RID**$_1$, **RID**$_2$, … } }

Key-value **API**

read, writes, and deletes

any other **operations**?

insert: put(k,v)

PQ: get(k) = {v} get(k) = {v_1,v_2,...}

get_set(k_1,k_2,...) = {v_1,v_2,...}

RQ: get_range(k_{min},k_{max}) = {v_1,v_2,...} full_scan() = {v_1,v_2,...}

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

delete: delete(k) delete(k_{min},k_{max})

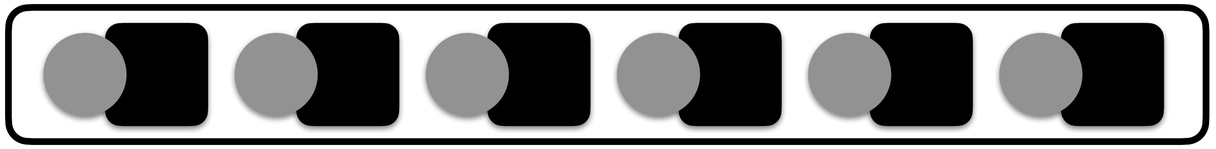
update: update(k,v_{new})

} **not very different from put**

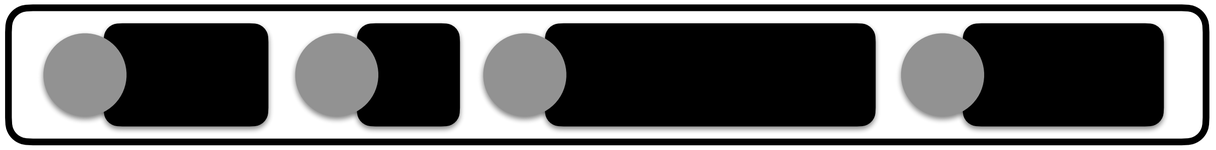
Storing **variable-size** entries

When entries are heterogeneous!

fixed entry size



variable entry size



how to access **variable-length entries**?

use offsets in page header

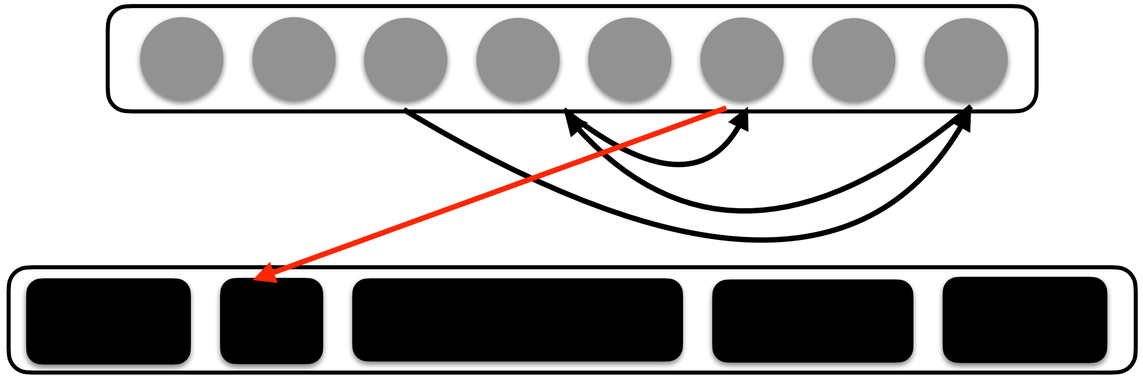
use pointers to values

Storing **variable-size** entries

When entries are heterogeneous!

use pointers to values

- **packed storage** for keys
- **compiler-friendly**
- values **need not be sorted**



any **limitations**?

- poor **range queries**
- loss of **locality**
- increased **code complexity**

Data structures for key-value stores

The underlying infrastructure

Thought Experiment 3

Which **data structure** to use for store data in key-value stores?

data structure:

- B^+-tree
- LSM-tree
- B^ϵ-tree
- Hash-based structures

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

LSM-tree

Patrick O'Neil, UMass Boston

LSM-tree

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

1996

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

LSM-tree

O'Neil *et al.*

1996

LSM-tree

O'Neil *et al.*

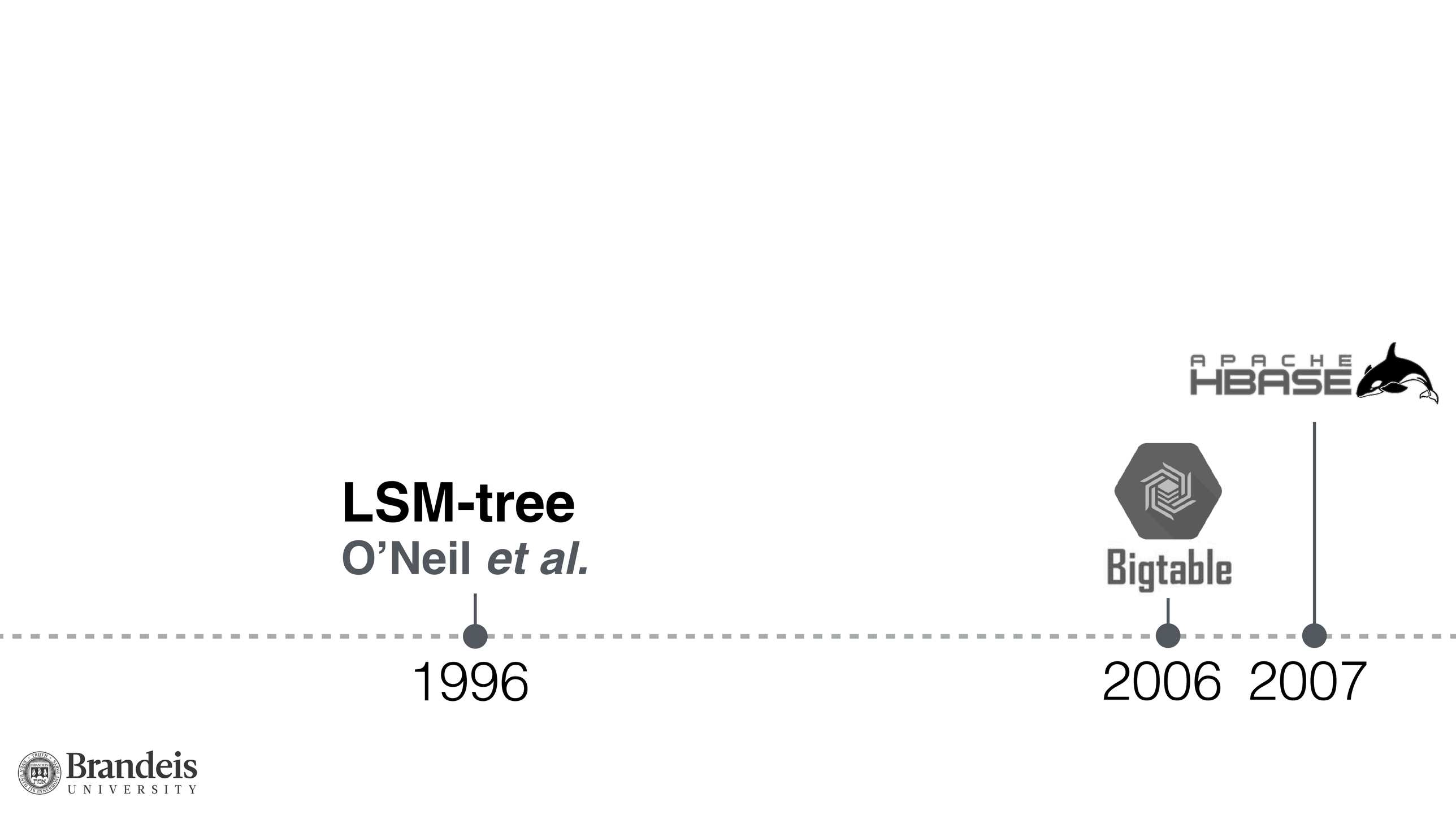
1996

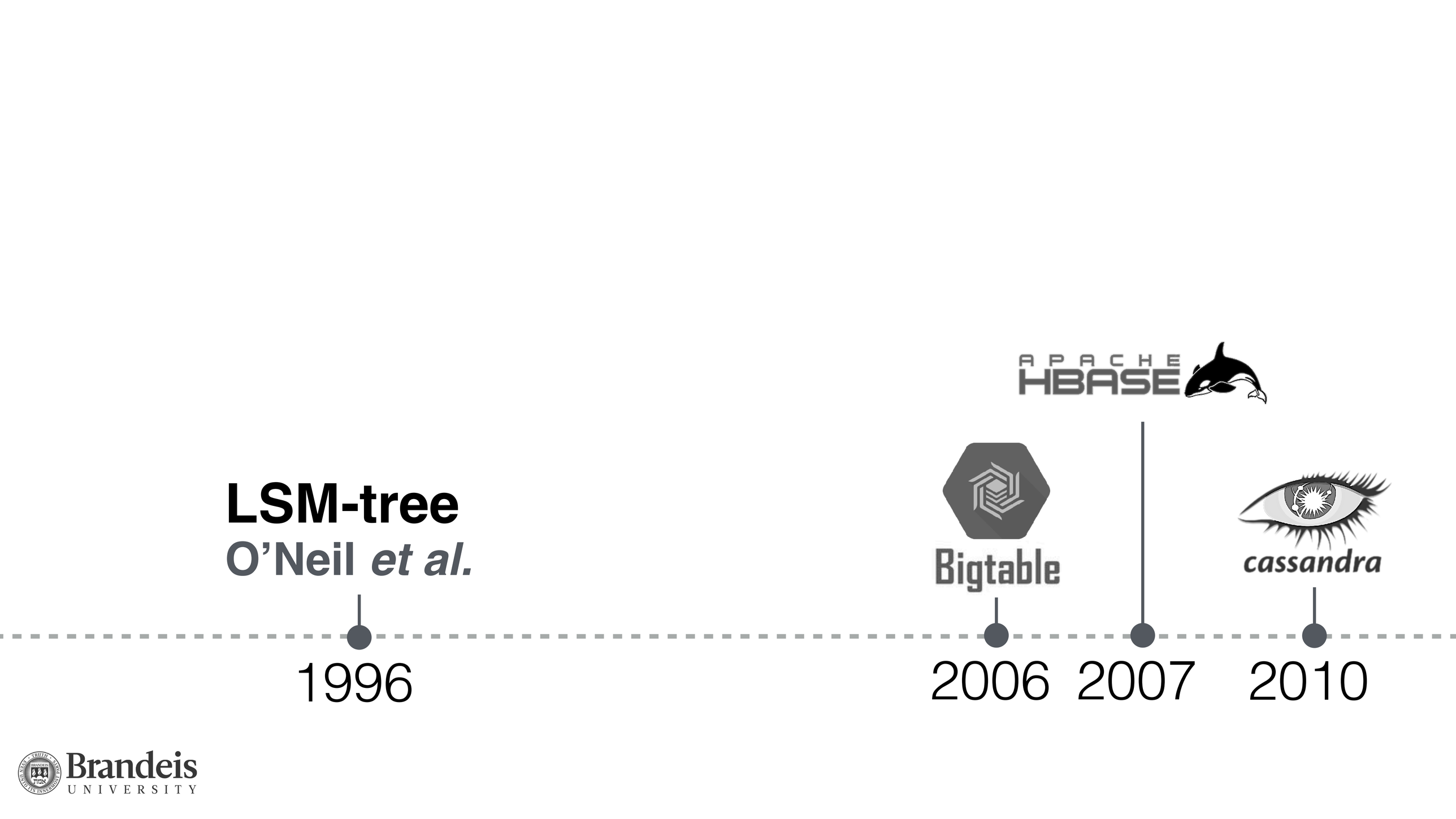
LSM-tree

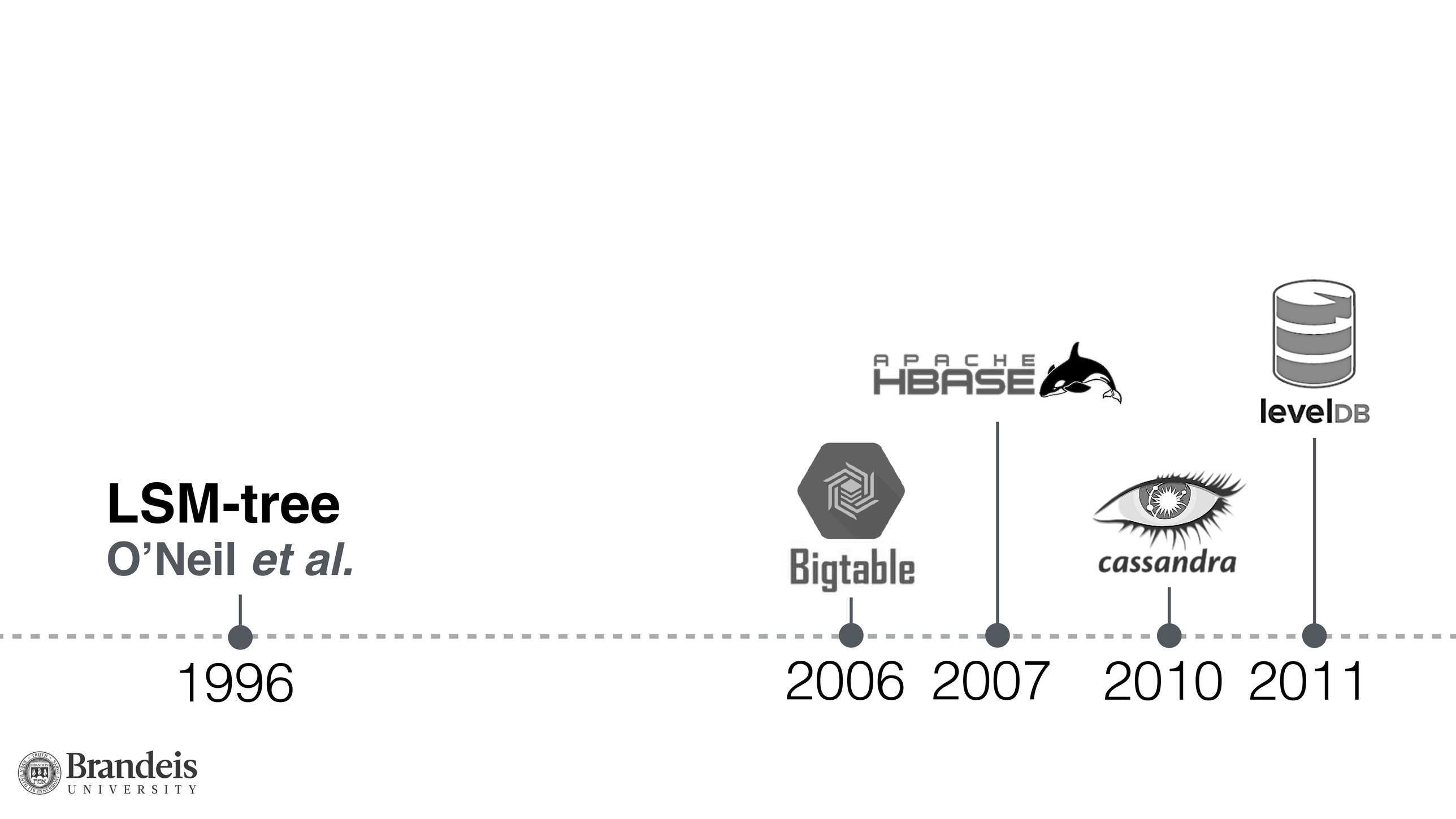
O'Neil *et al.*

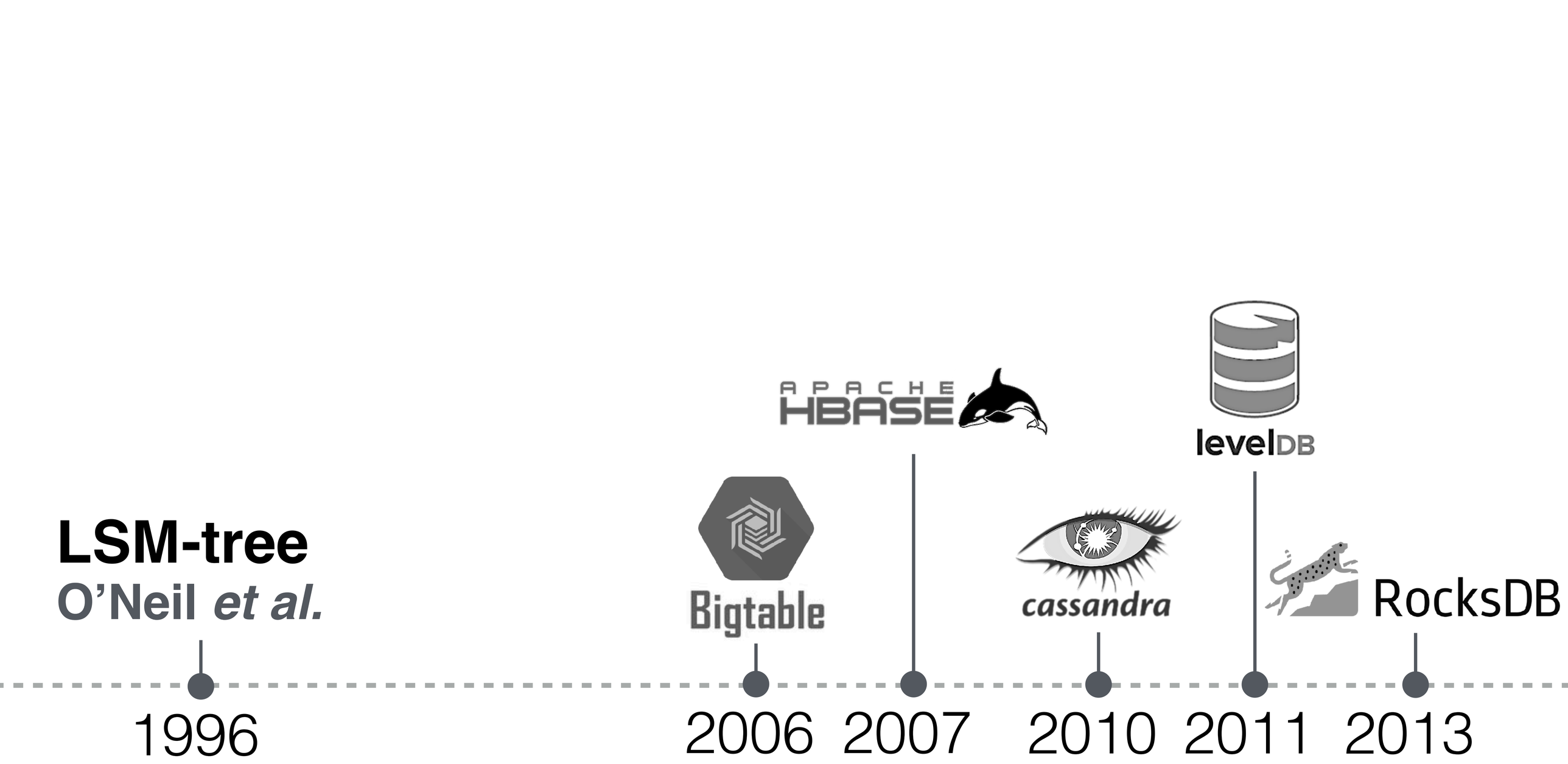
1996

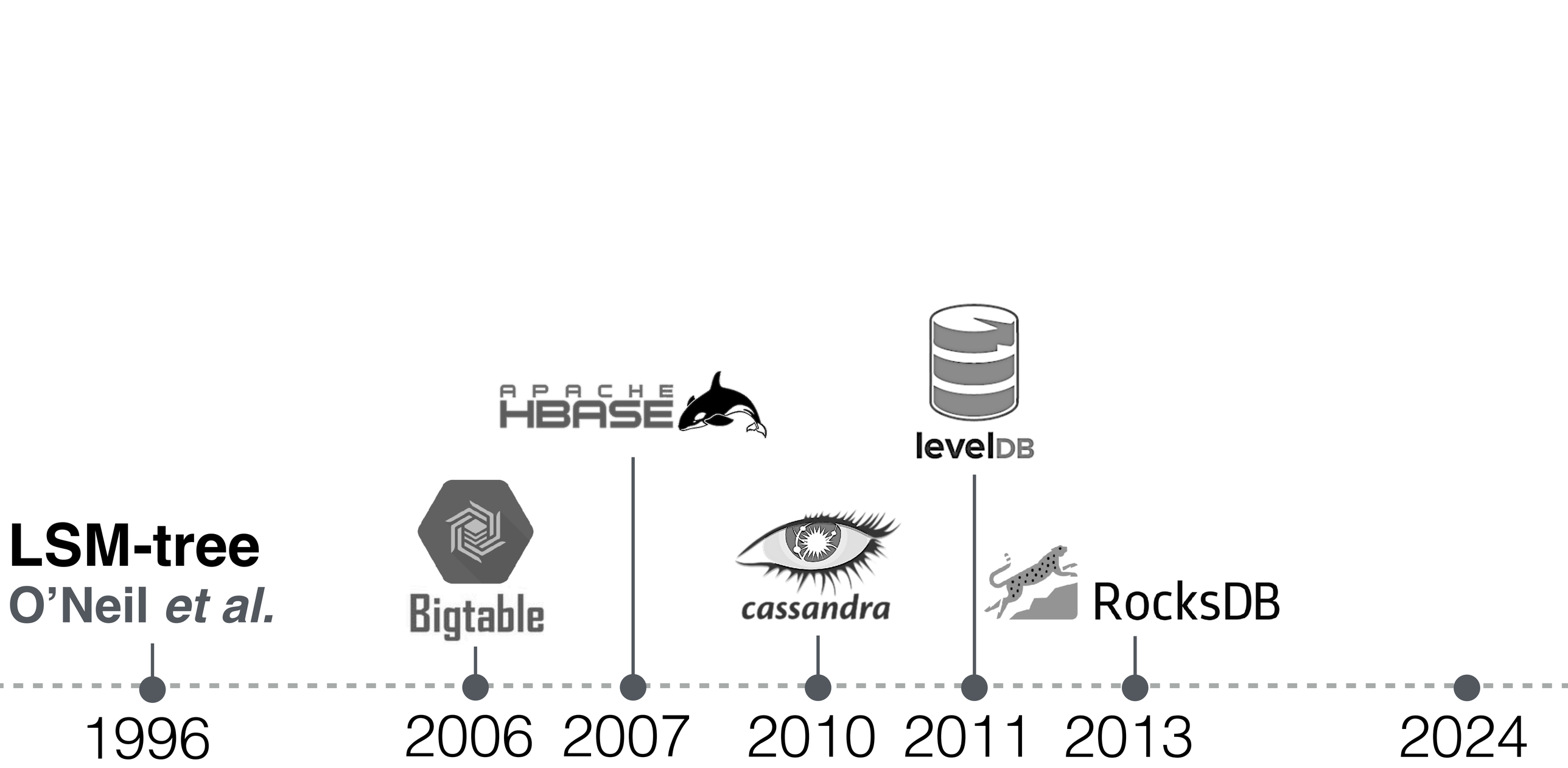
2006

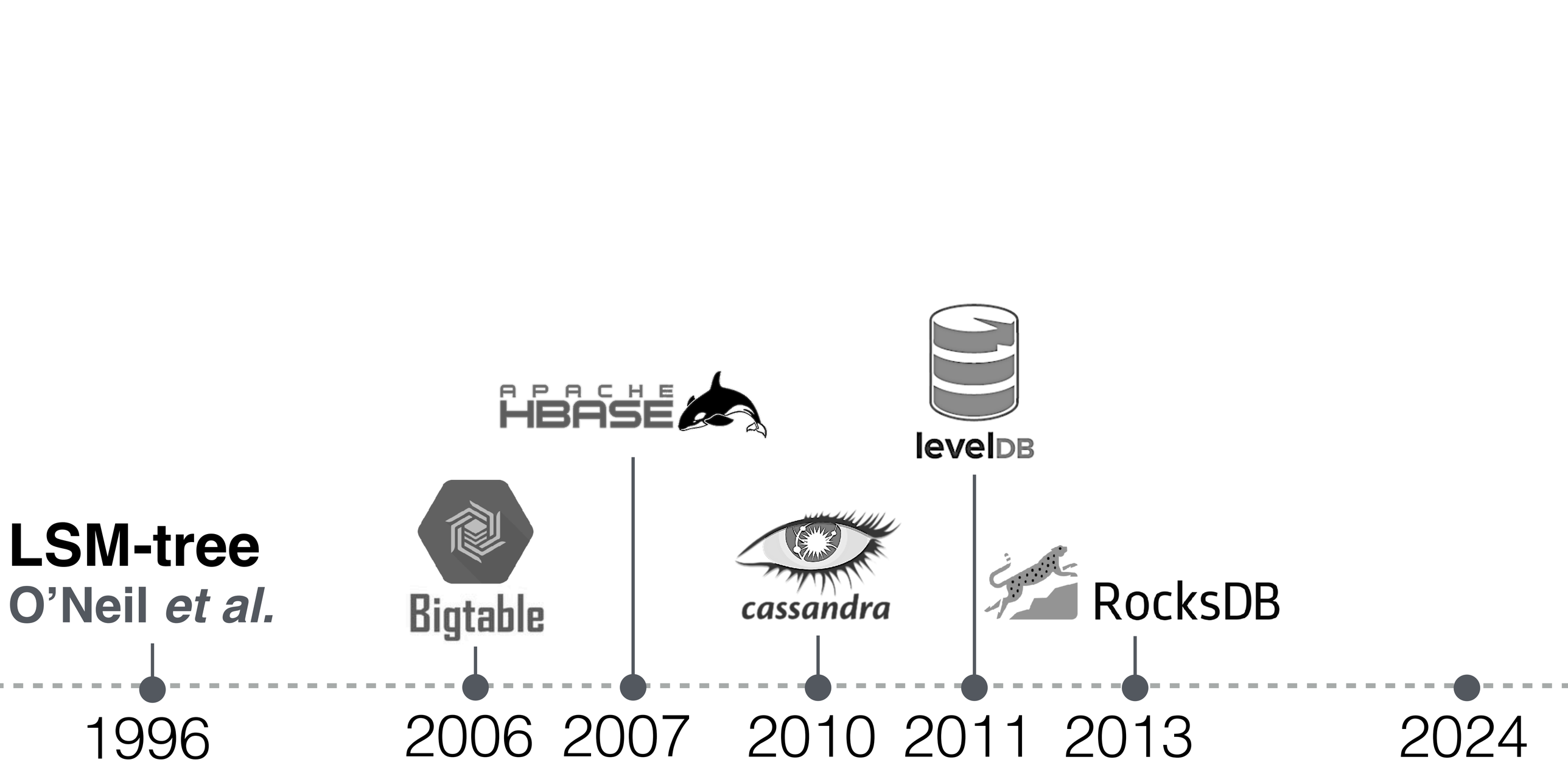

LSM-tree
O'Neil et al.
1996
Bigtable
2006
APACHE HBASE
2007


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

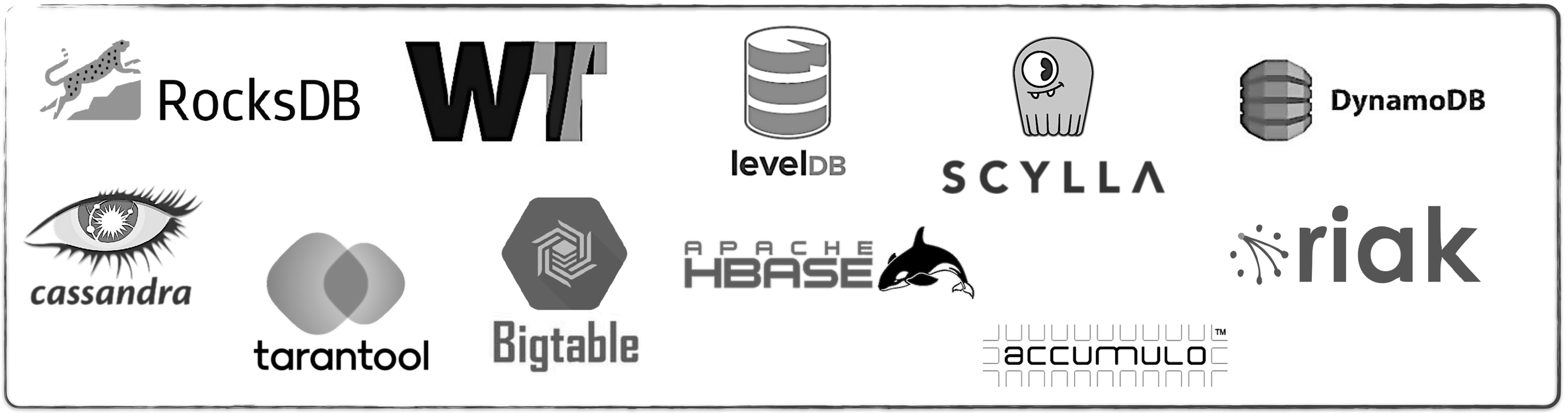

LSM-tree
O'Neil et al.
1996
Bigtable
2006
APACHE HBASE
2007
cassandra
2010
levelDB
2011


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


LSM-tree
O'Neil et al.
APACHE HBASE
Bigtable
levelDB
cassandra
RocksDB
1996
2006
2007
2010
2011
2013
2024


LSM-tree
O'Neil et al.
1996
Bigtable
2006
APACHE HBASE
2007
cassandra
2010
levelDB
2011
RocksDB
2013
2024

LSM-tree


NoSQL
RocksDB
WT
levelDB
SCYLLA
DynamoDB
cassandra
tarantool
Bigtable
APACHE HBASE
riak
accumulo
SQLite
influxdb
QuasarDB
relational
time-series
2024

LSM-tree

NoSQL

relational

time-series

2024

Why **LSM**?

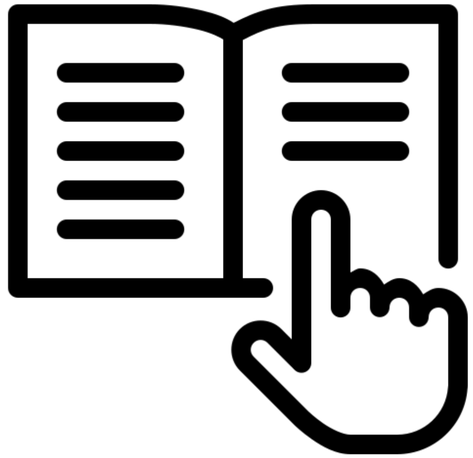
What's the hype all about?

fast ingestion

Why **LSM**?

What's the hype all about?

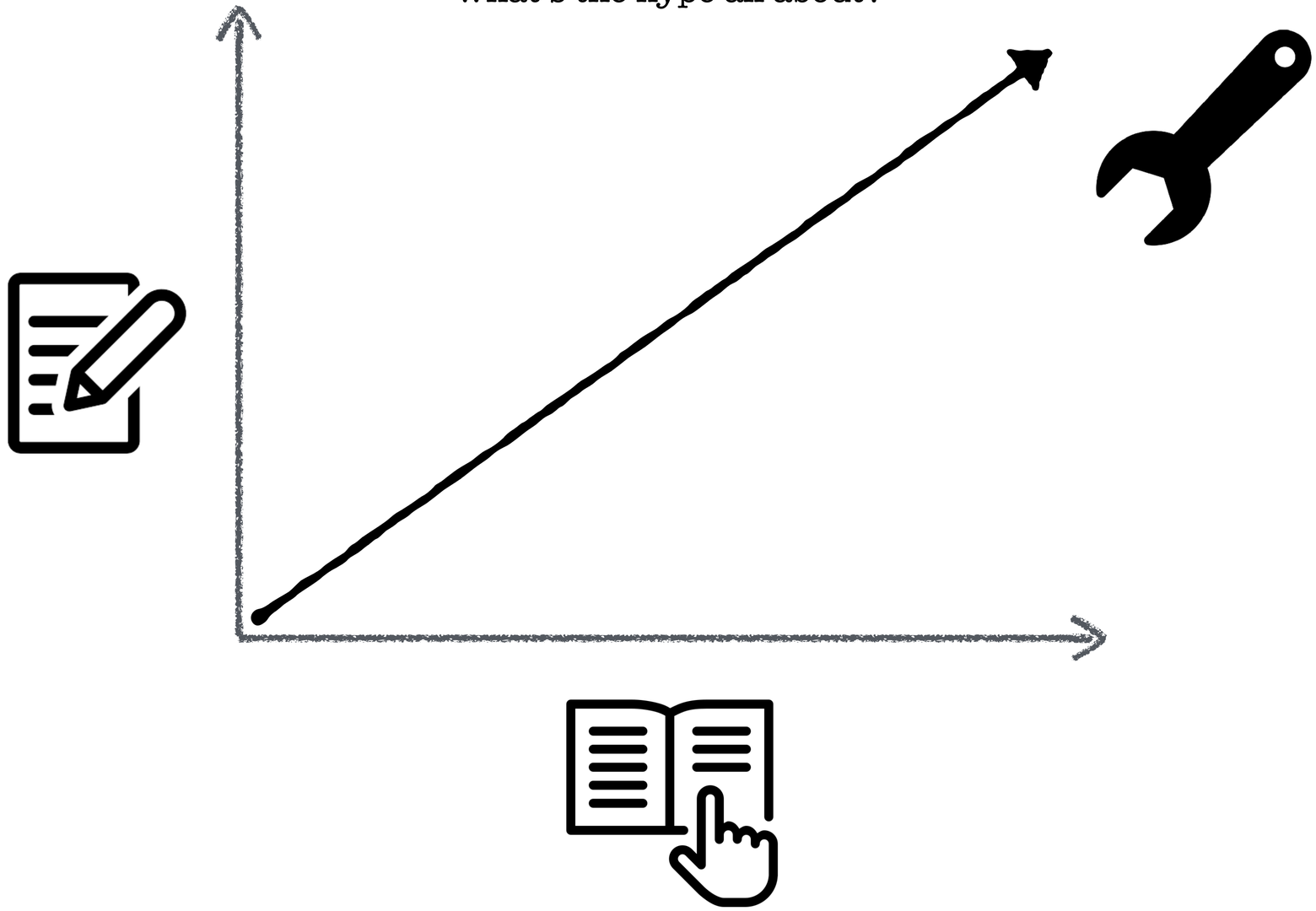
fast ingestion



competitive reads

Why **LSM**?

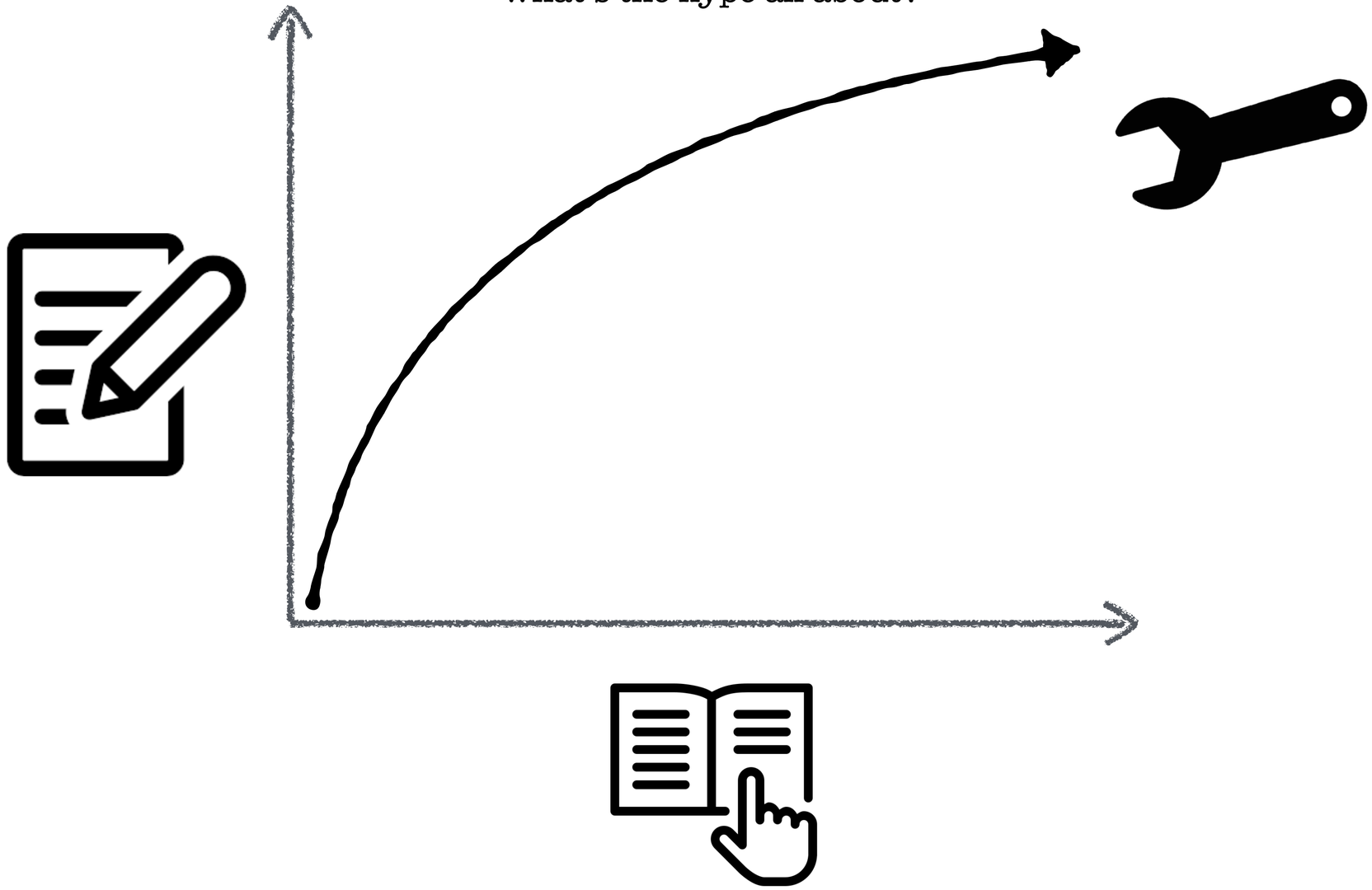
What's the hype all about?



Brandeis UNIVERSITY

Why **LSM**?

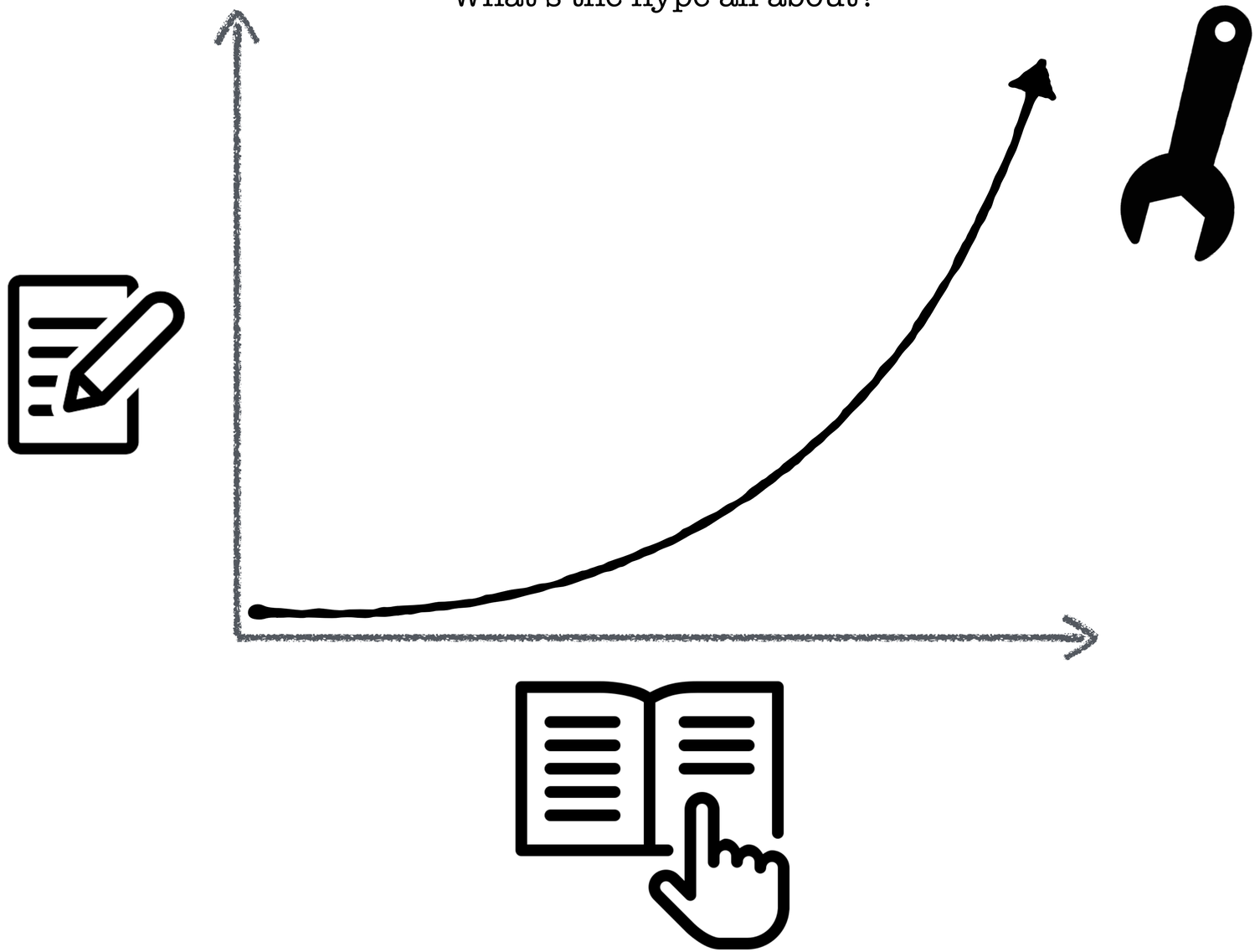
What's the hype all about?



Brandeis UNIVERSITY

Why **LSM**?

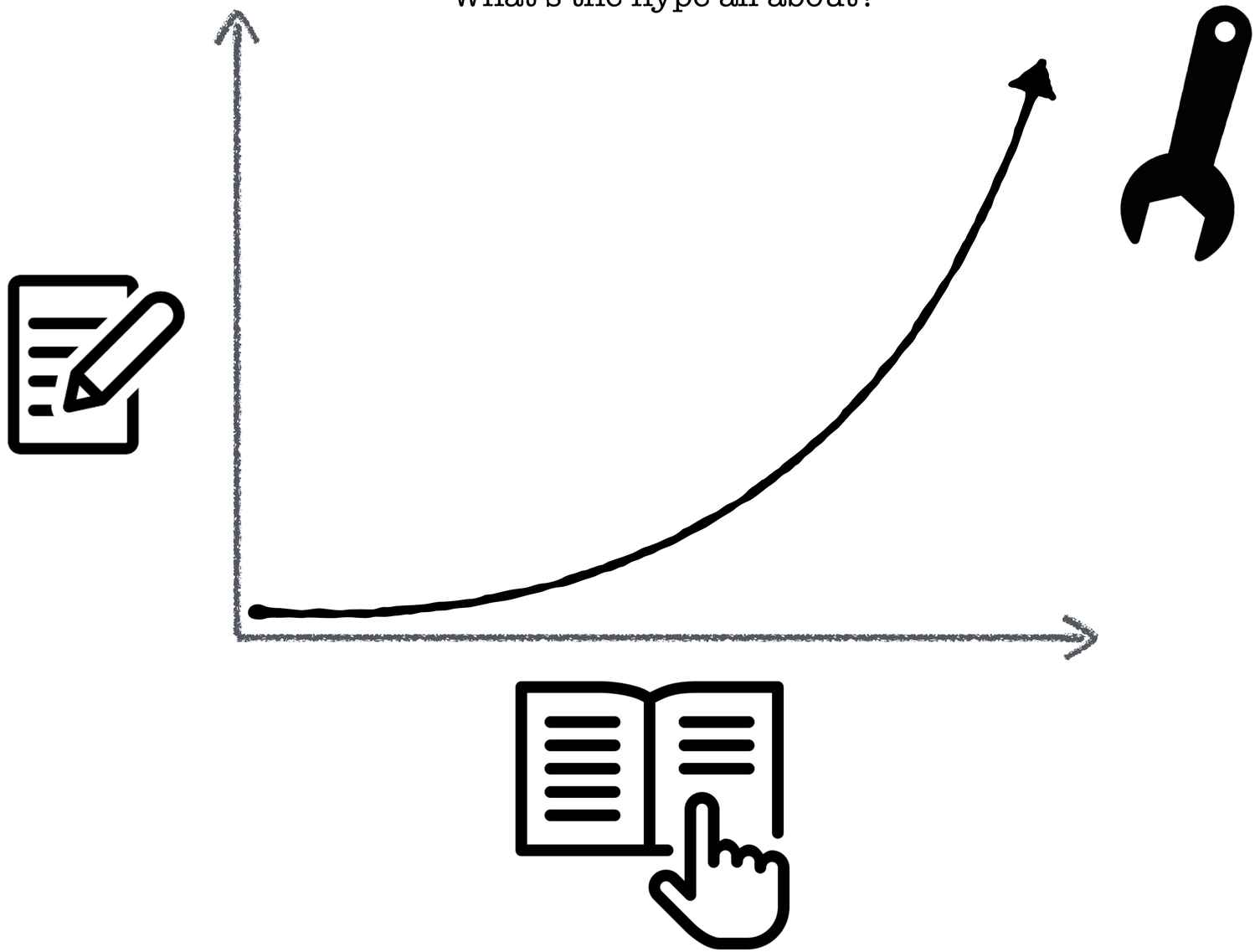
What's the hype all about?



Brandeis UNIVERSITY

Why **LSM**?

What's the hype all about?

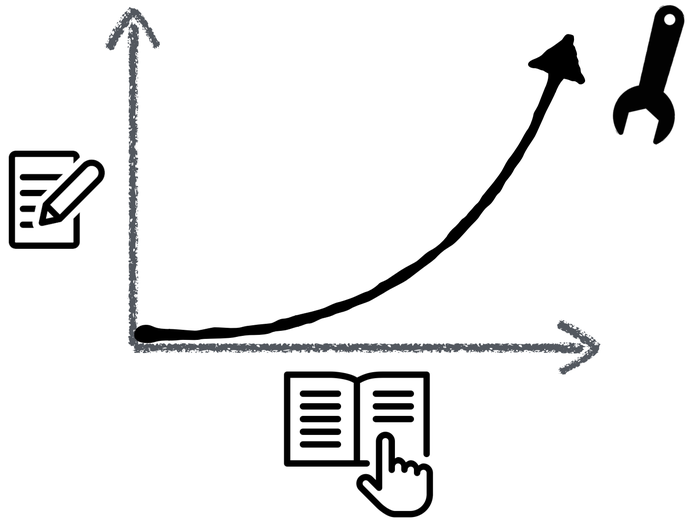


Brandeis UNIVERSITY

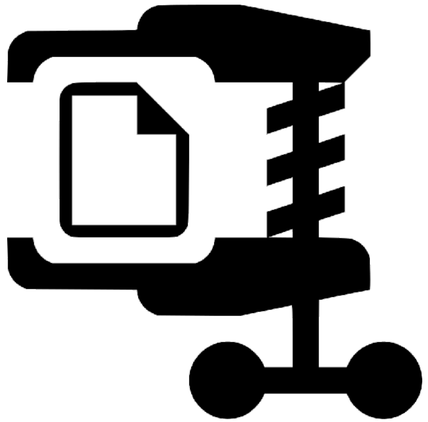
Why **LSM**?

What's the hype all about?

fast writes



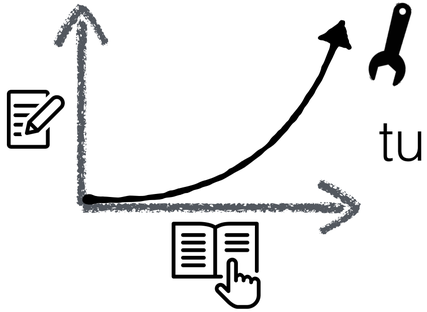
tunable read-write performance



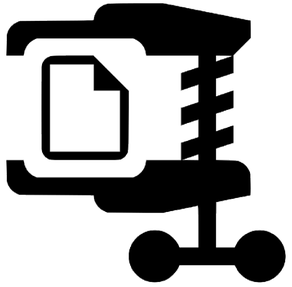
good space utilization

Research **trend**

fast writes

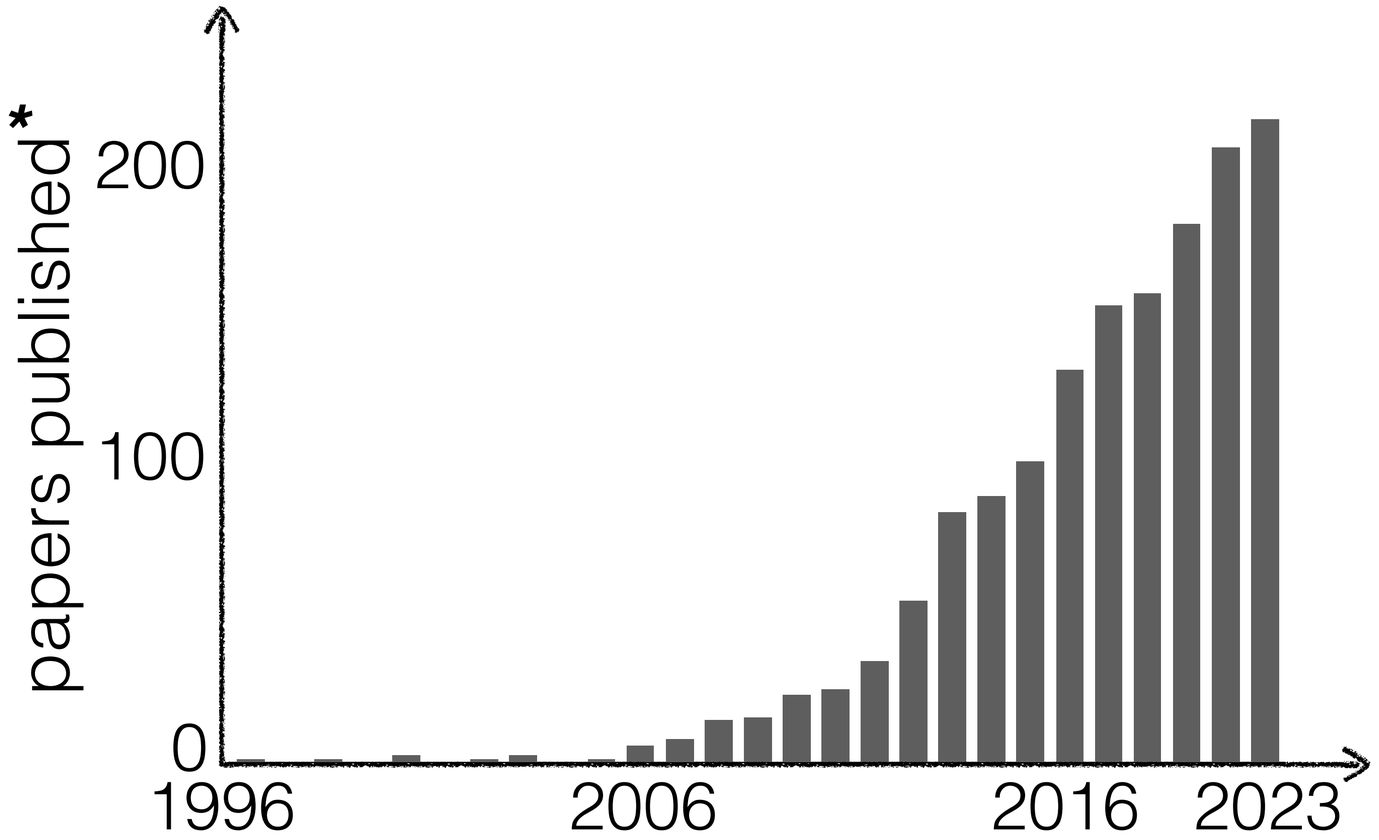


tunable read-write performance



good space utilization

LSMs everywhere!



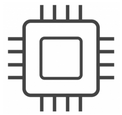
* data from Google scholar

LSM **basics**

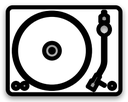
How do they look?

LSM **basics**

How do they look?



buffer



level 1

level 2

level 3

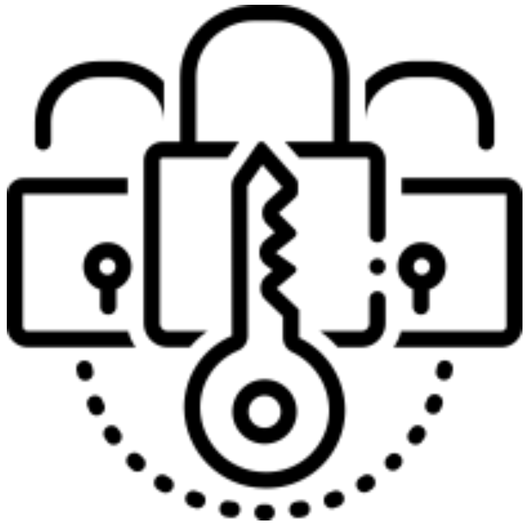
size ratio: T

Great! But, how does it work?

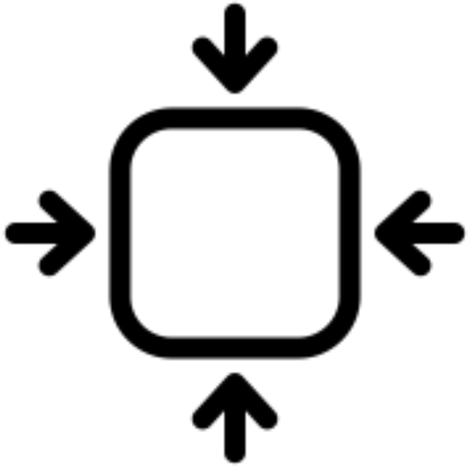
Design principles

How do they work?

buffering ingestion



immutable files on storage

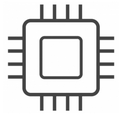


periodic **compaction**

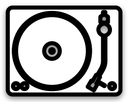
Buffering ingestion

put(6)

put(2)



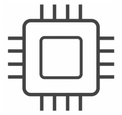
buffer



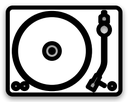
Buffering ingestion

put(1)

put(6)



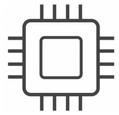
buffer 2



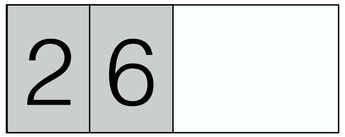
Buffering ingestion

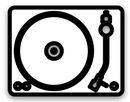
put(4)

put(1)



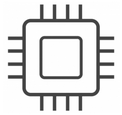
buffer



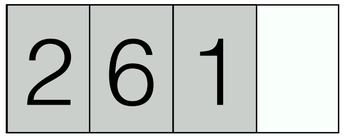


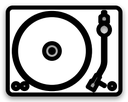
Buffering ingestion

put(4)

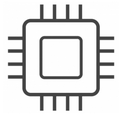


buffer



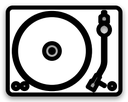


Buffering ingestion

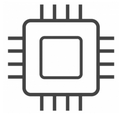


buffer

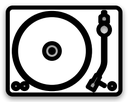
2	6	1	4



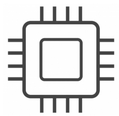
Buffering ingestion



buffer | 1 | 2 | 4 | 6 |

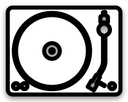


Buffering ingestion

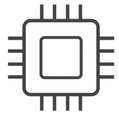


buffer

1 2 4 6

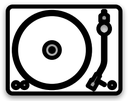


Buffering ingestion



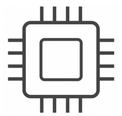
buffer

1 2 4 6

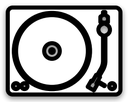


Buffering ingestion

- low ingestion cost
- fast temporal reads



buffer



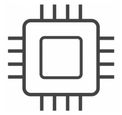
level 1

1 2 4 6

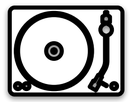
flush

Immutable files on storage

- compact storage
- good ingestion throughput



buffer



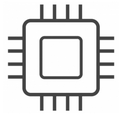
level 1

no changes can be made to data once it's on disk

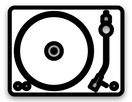
Thought Experiment 4

But how do we **update data**?

Out of place updates!



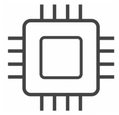
buffer



level 1

1 2 4 6

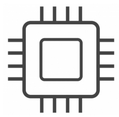
put(6)



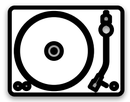
buffer

level 1

1 2 4 6

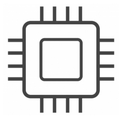


buffer



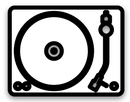
level 1

logically
invalidated



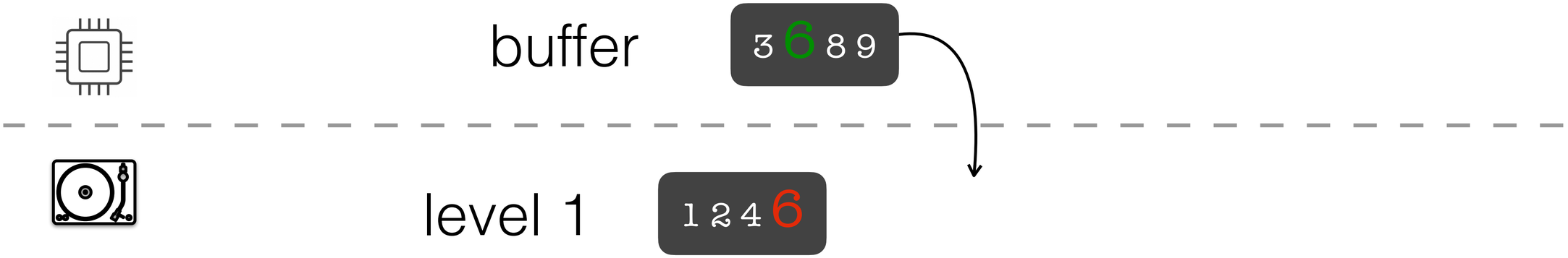
buffer

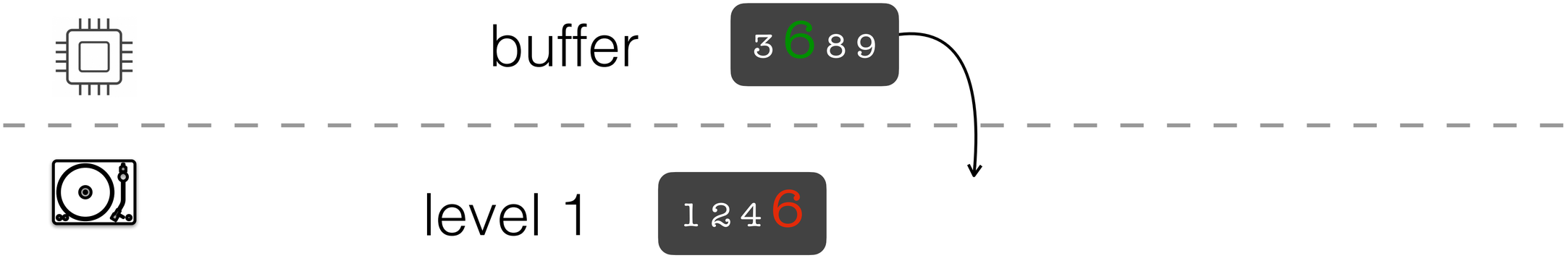
3 6 8 9



level 1

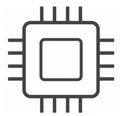
1 2 4 6


buffer
3 6 8 9
level 1
1 2 4 6

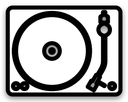

buffer
3 6 8 9
level 1
1 2 4 6

Out-of-place updates

- fast updates
- space amplification



buffer

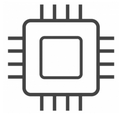


level 1

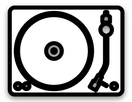
Thought Experiment 5

But how do we **reduce space amplification**?

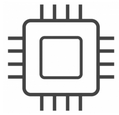
Merge sorted runs!



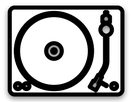
buffer



level 1



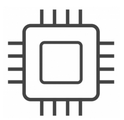
buffer



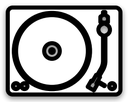
level 1

1 2 3 4 6 8 9

compaction



buffer



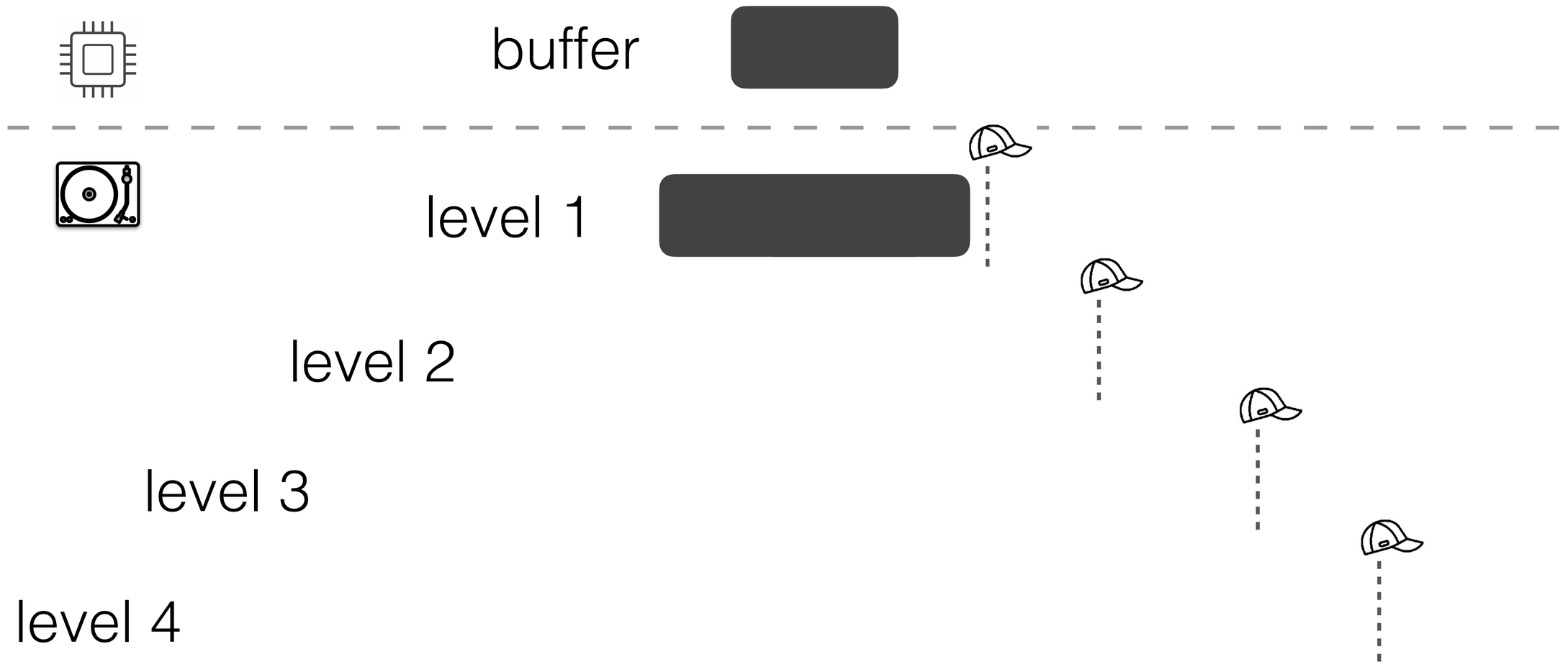
level 1

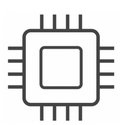
level 2

level 3

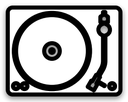
level 4

Brandeis UNIVERSITY


buffer
level 1
level 2
level 3
level 4



buffer

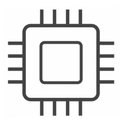


level 1

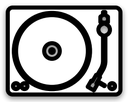
level 2

level 3

level 4



buffer

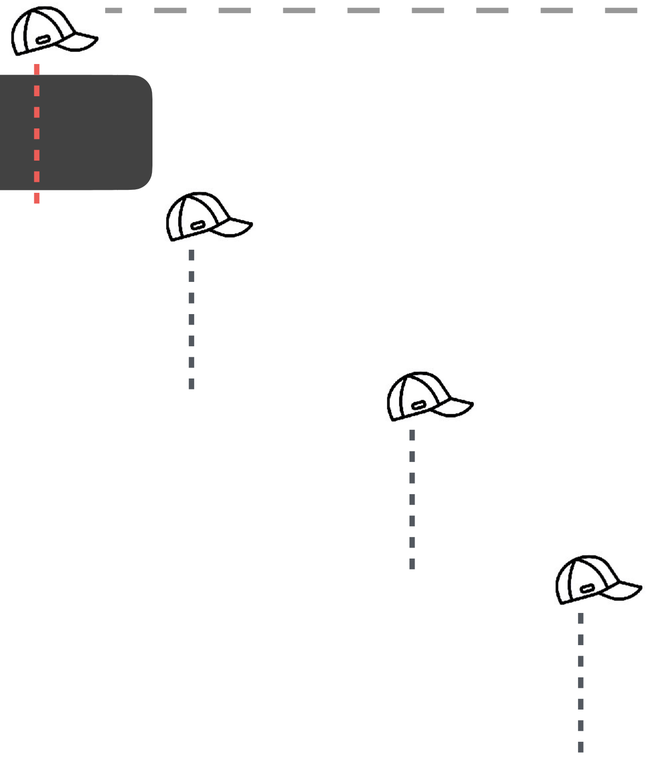


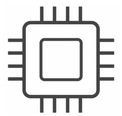
level 1

level 2

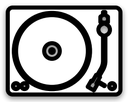
level 3

level 4





buffer

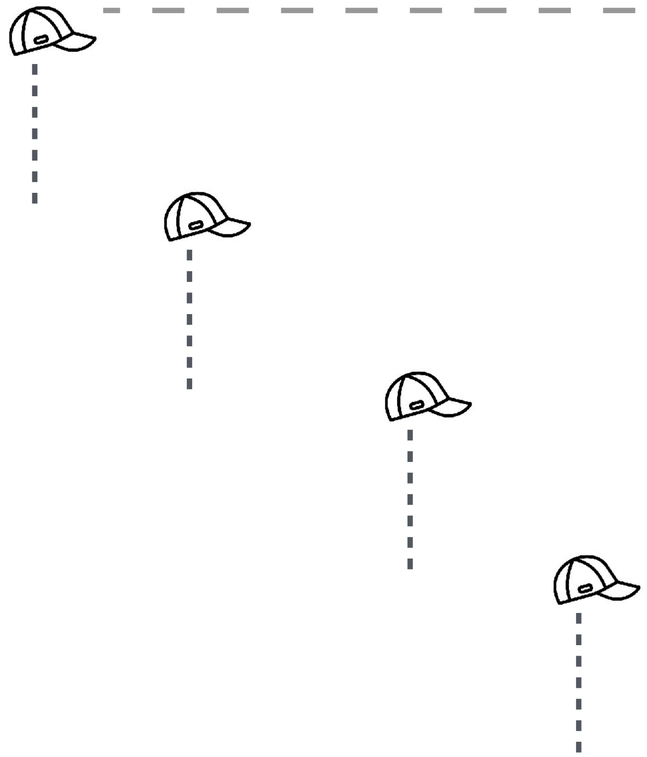


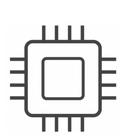
level 1

level 2

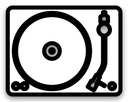
level 3

level 4





buffer

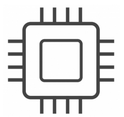


level 1

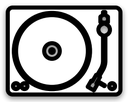
level 2

level 3

level 4



buffer

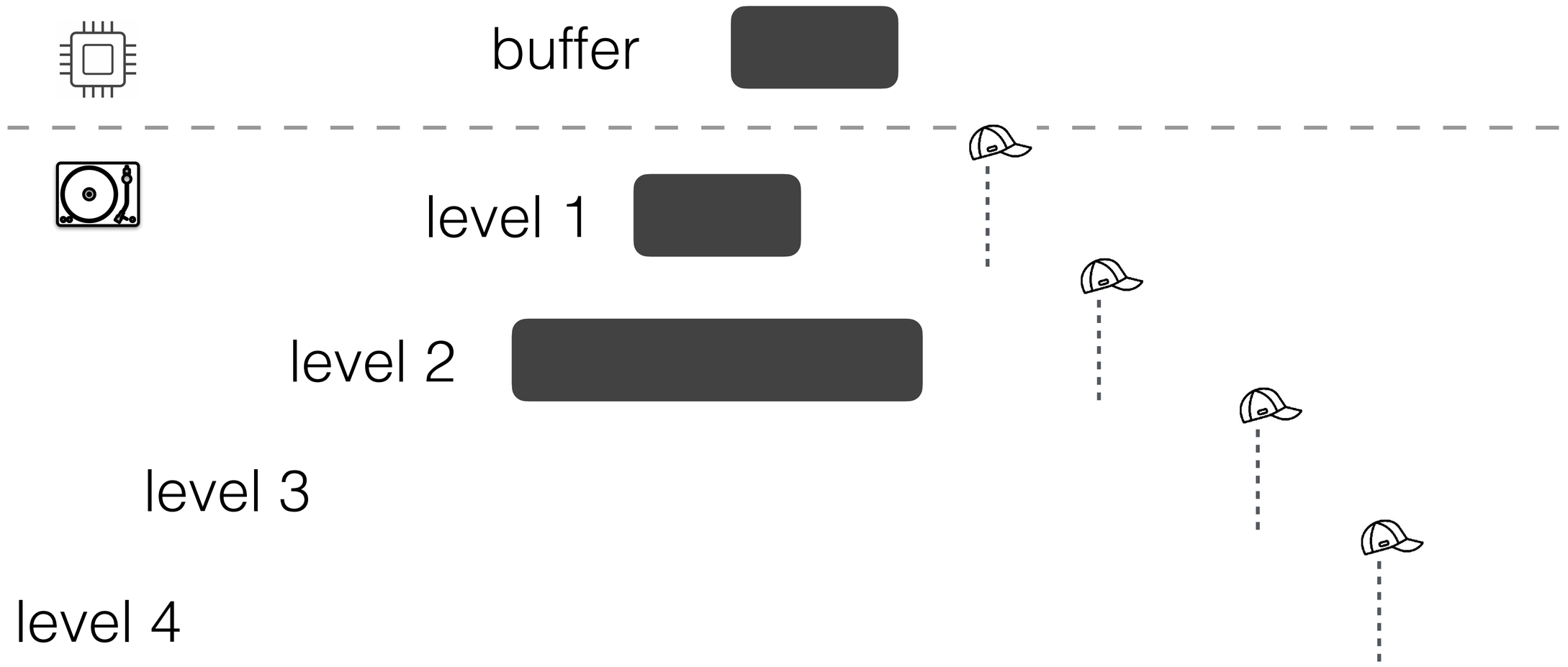


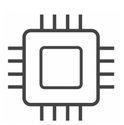
level 1

level 2

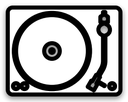
level 3

level 4


buffer
level 1
level 2
level 3
level 4



buffer

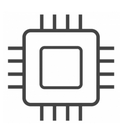


level 1

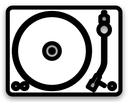
level 2

level 3

level 4



buffer

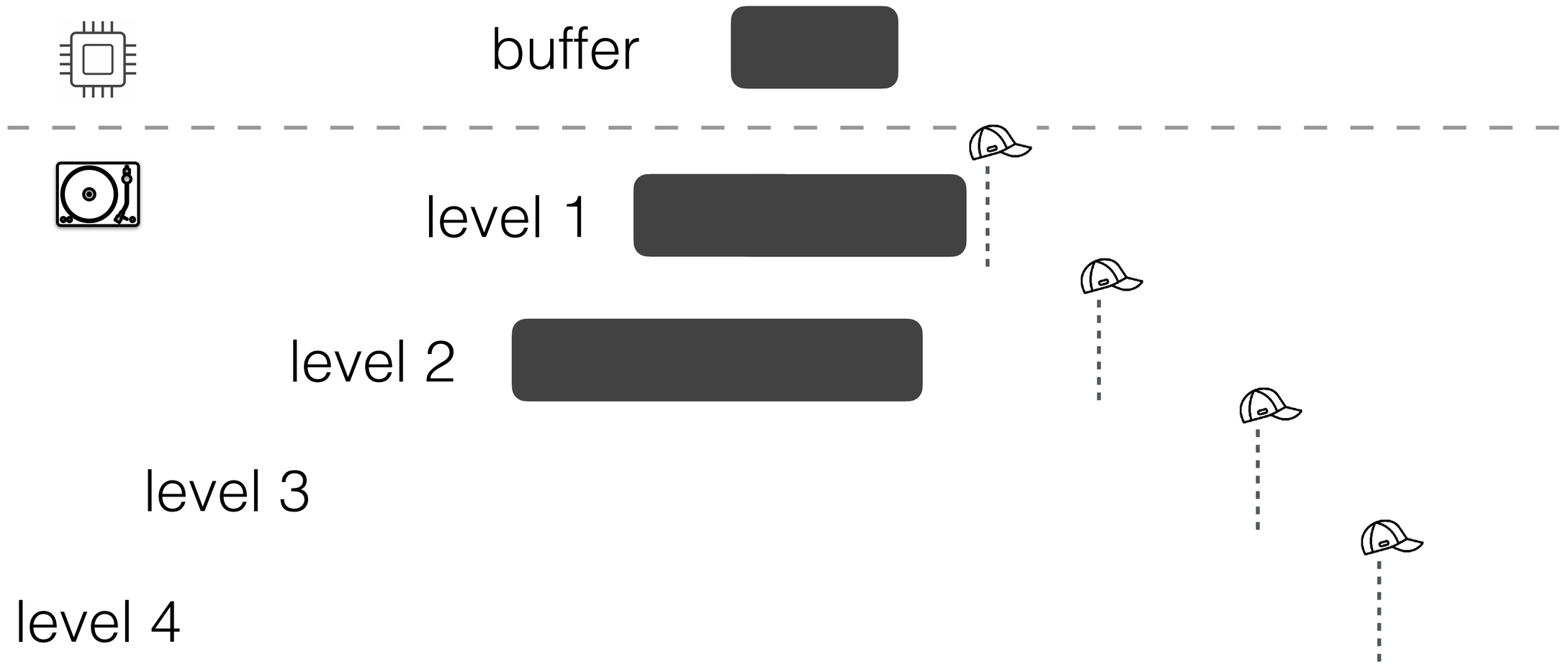


level 1

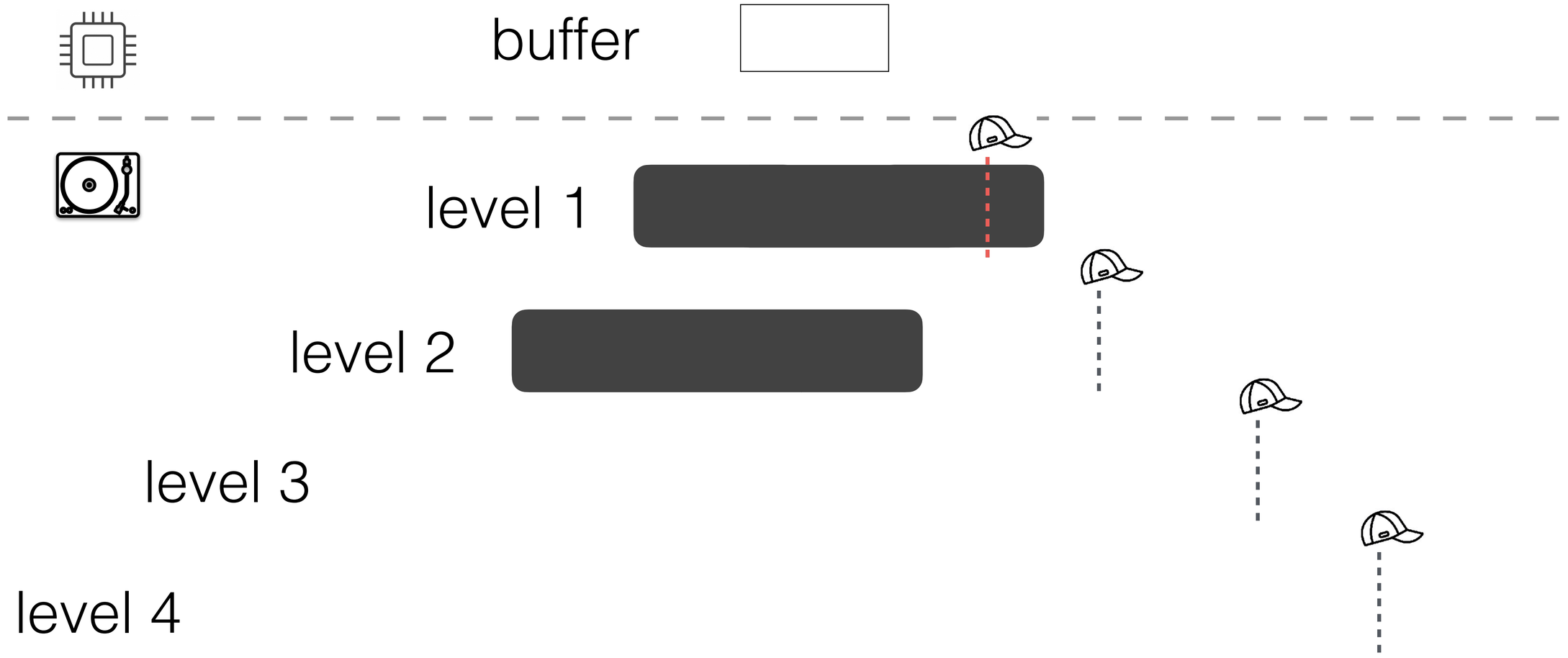
level 2

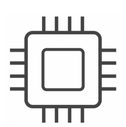
level 3

level 4

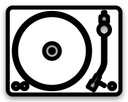

buffer
level 1
level 2
level 3
level 4


buffer
level 1
level 2
level 3
level 4


buffer
level 1
level 2
level 3
level 4



buffer

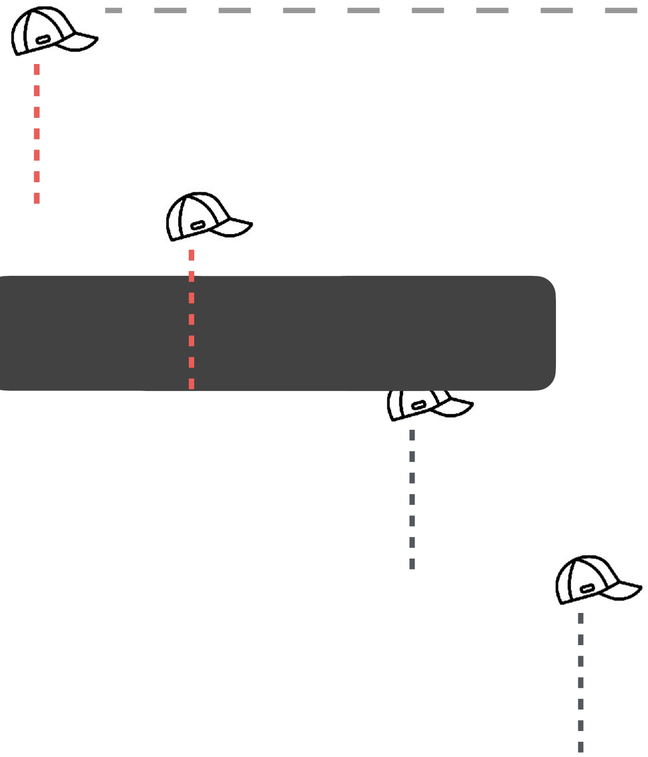


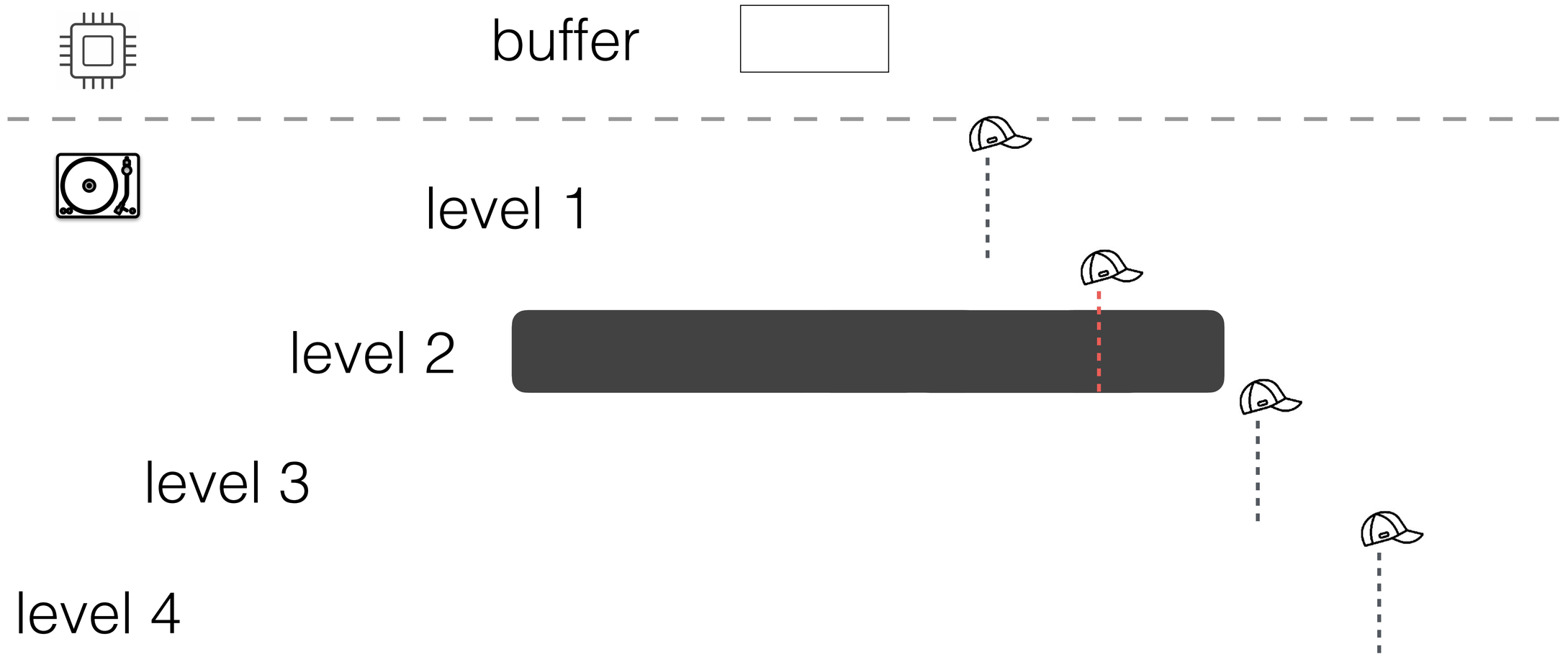
level 1

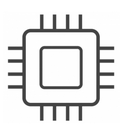
level 2

level 3

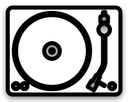
level 4




buffer
level 1
level 2
level 3
level 4



buffer



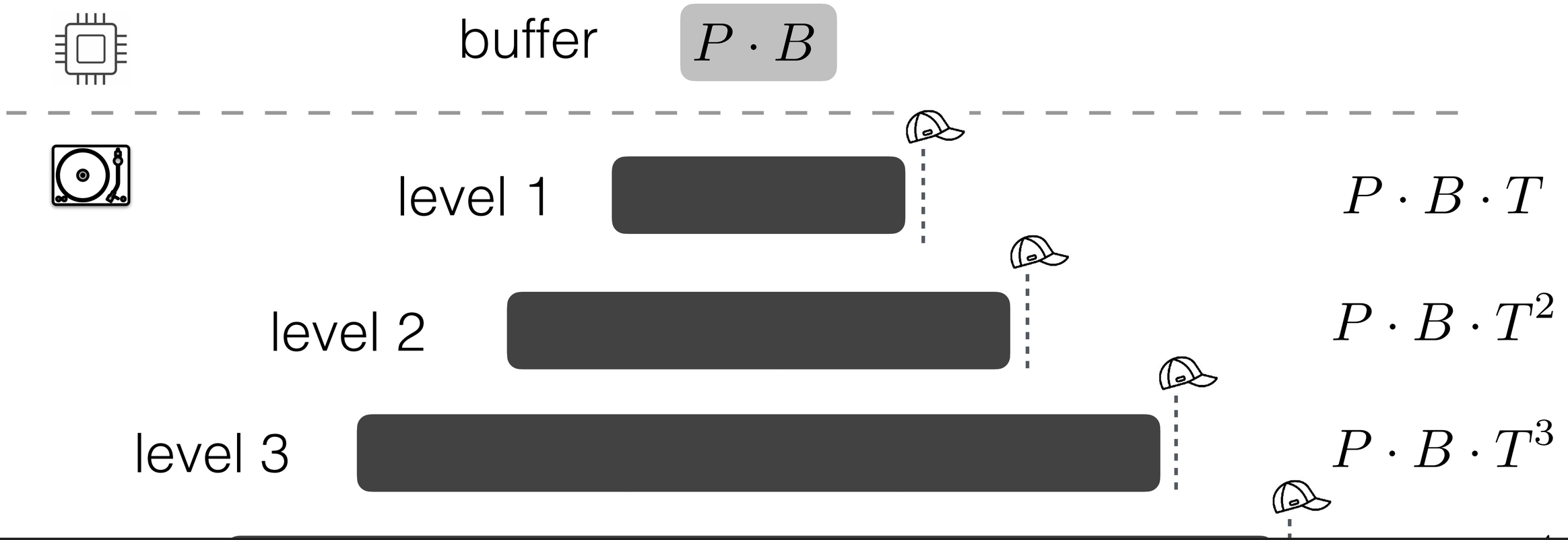
level 1

level 2

level 3

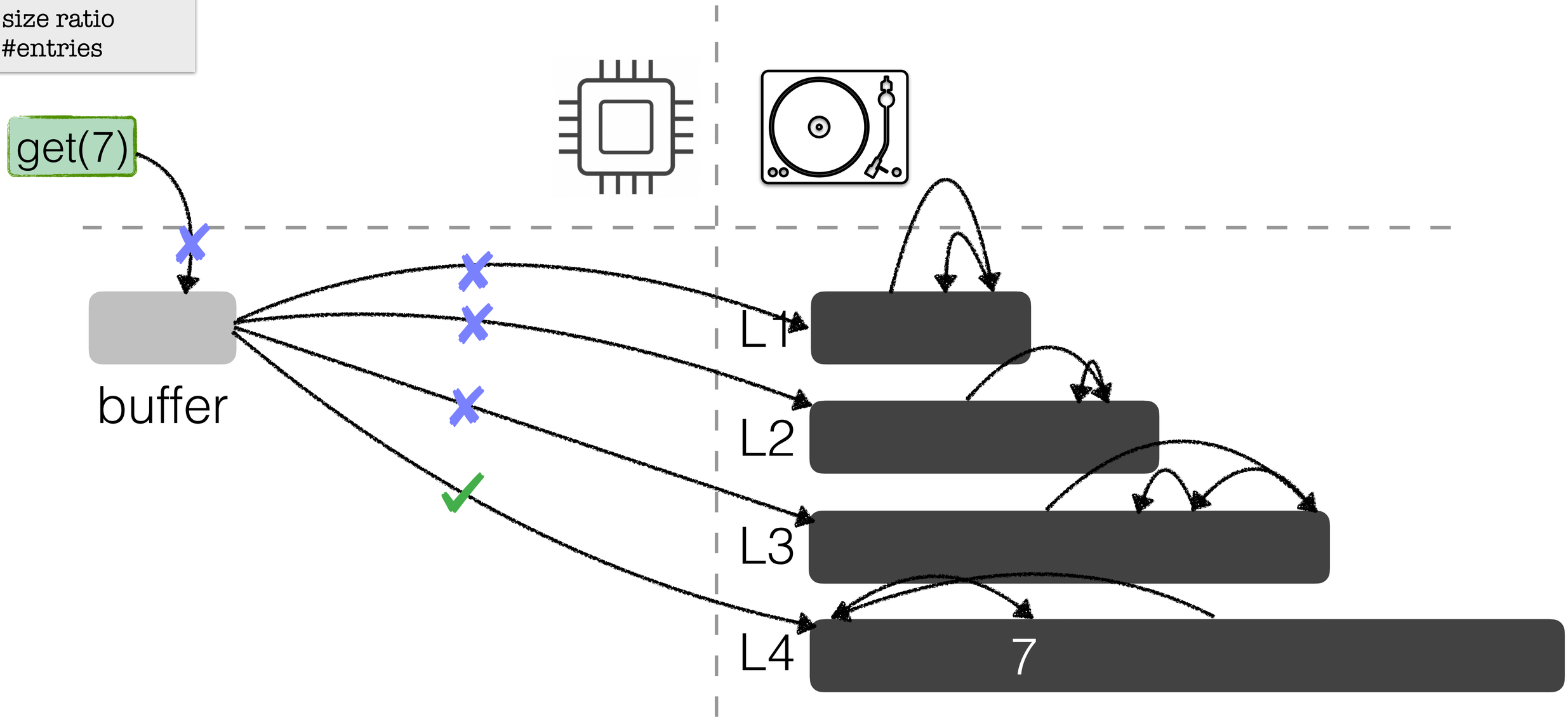
level 4

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

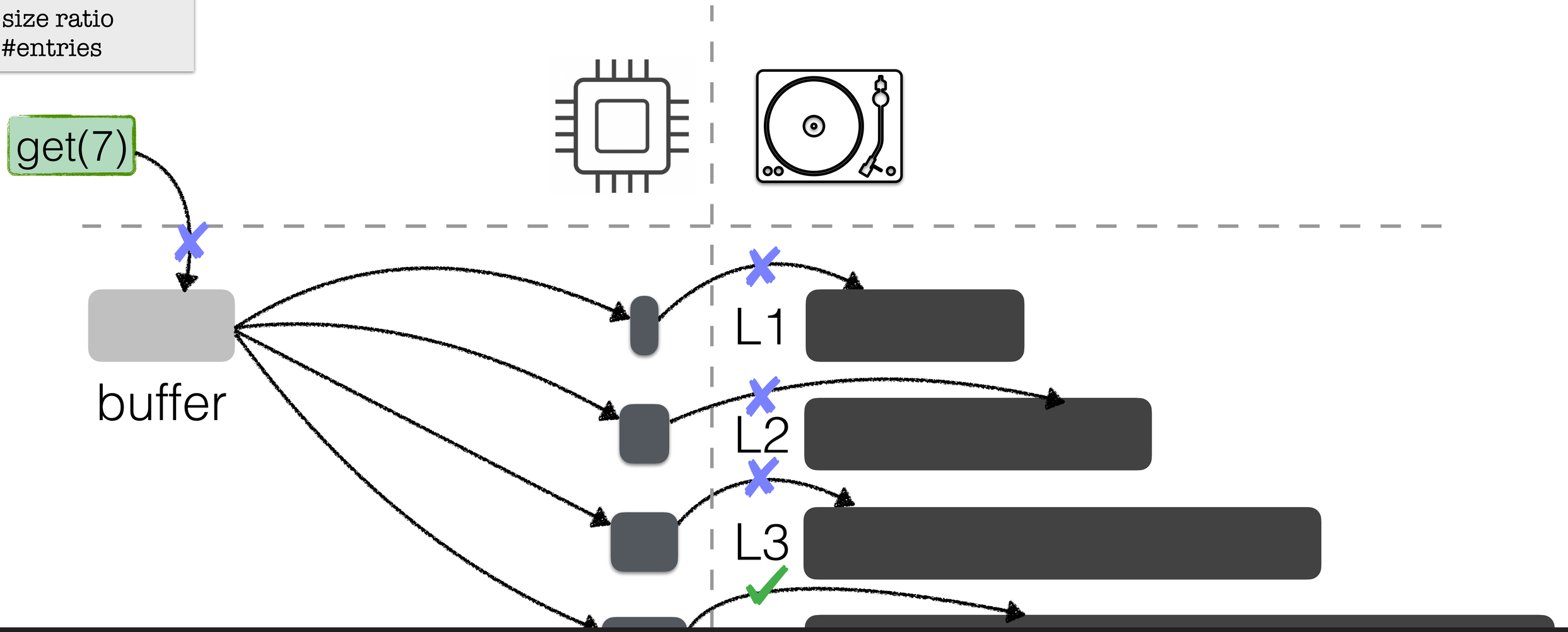


How about queries?

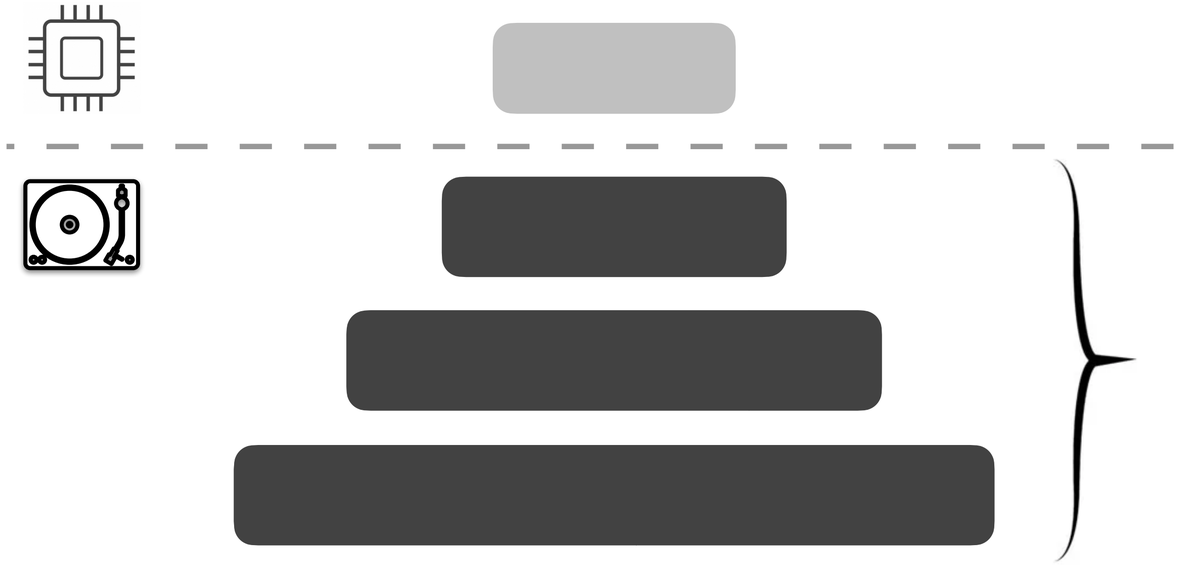
P: pages in buffer
B: entries/page
L: #levels
T: size ratio
N: #entries



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

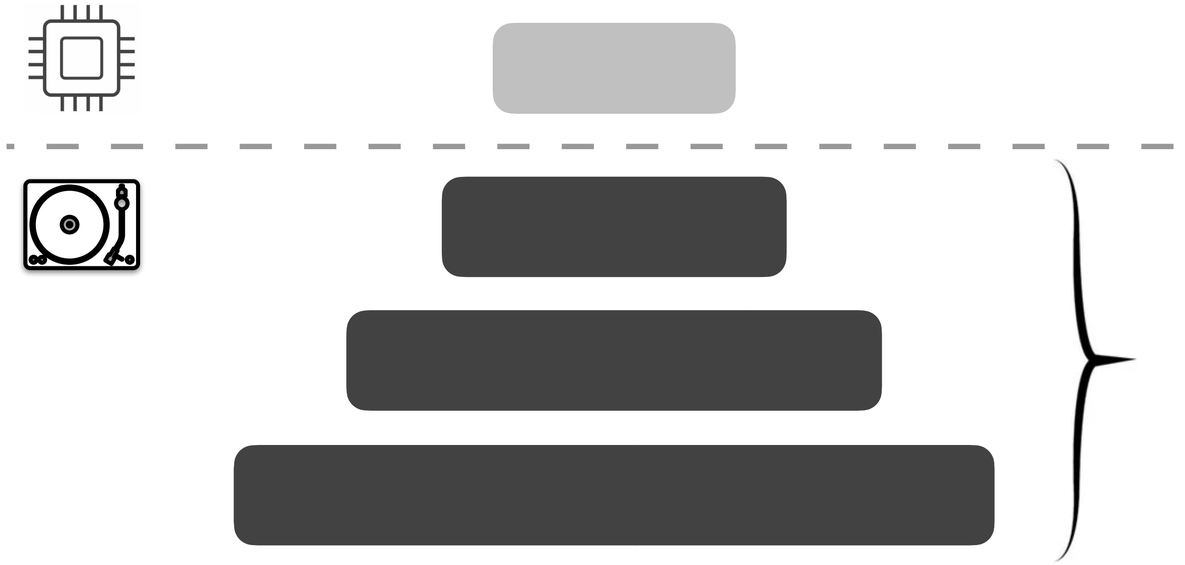


Can we do better?



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 ingestion?

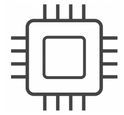
Data **Layout**

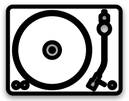
Classical LSM design: **leveling**

[eager merging]

Data **Layout**

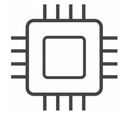
leveling [eager]

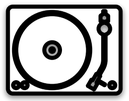




Data **Layout**

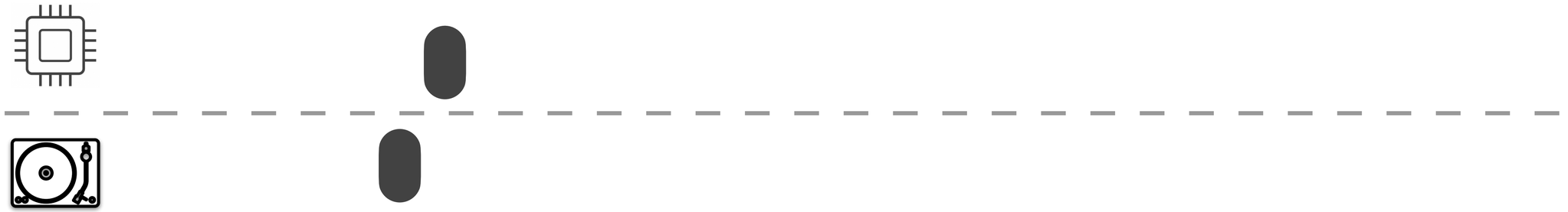
leveling [eager]





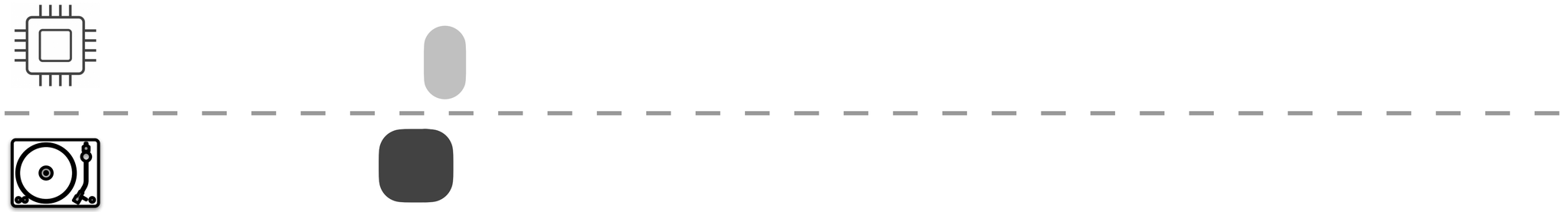
Data **Layout**

leveling [eager]



Data **Layout**

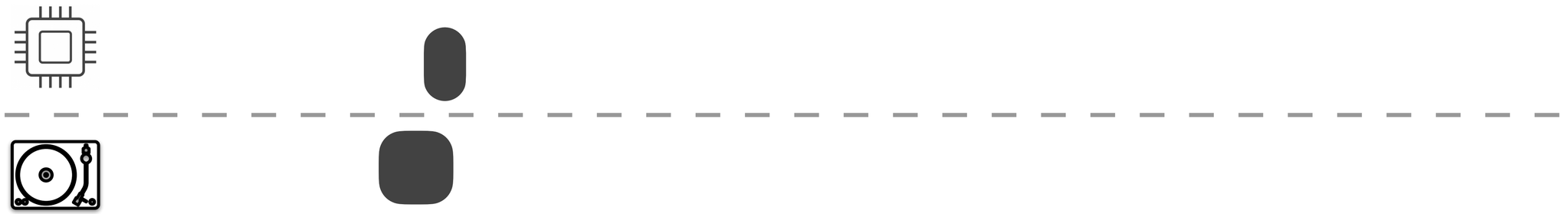
leveling [eager]



Brandeis UNIVERSITY

Data **Layout**

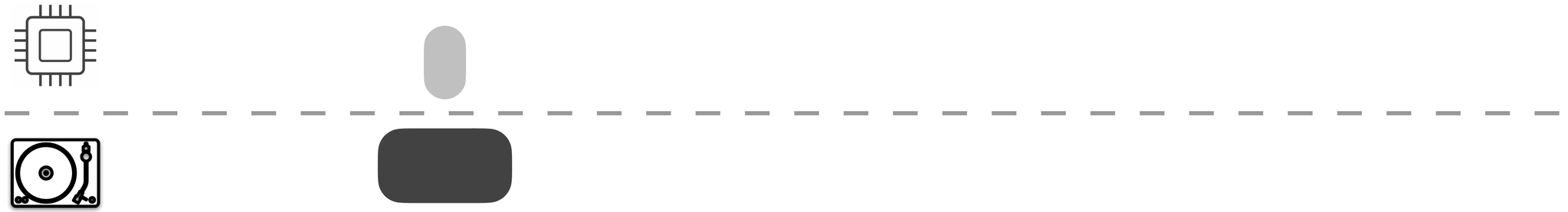
leveling [eager]



Brandeis UNIVERSITY

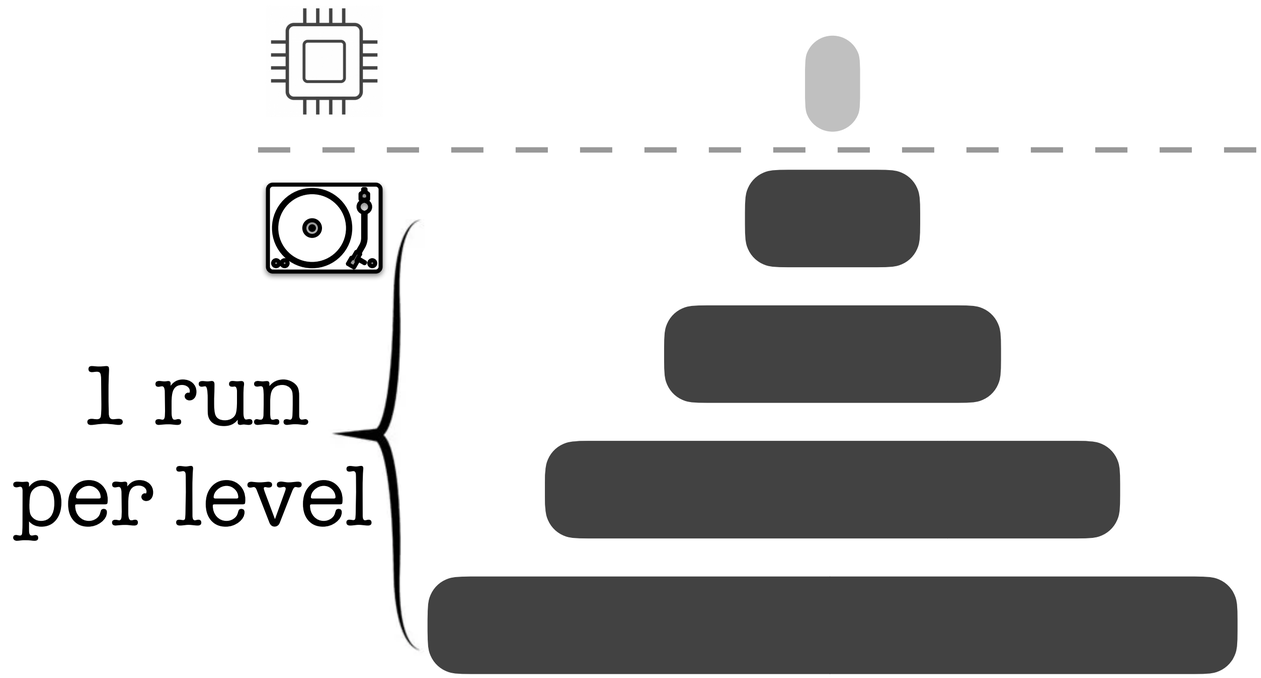
Data **Layout**

leveling [eager]



Data **Layout**

leveling [eager]

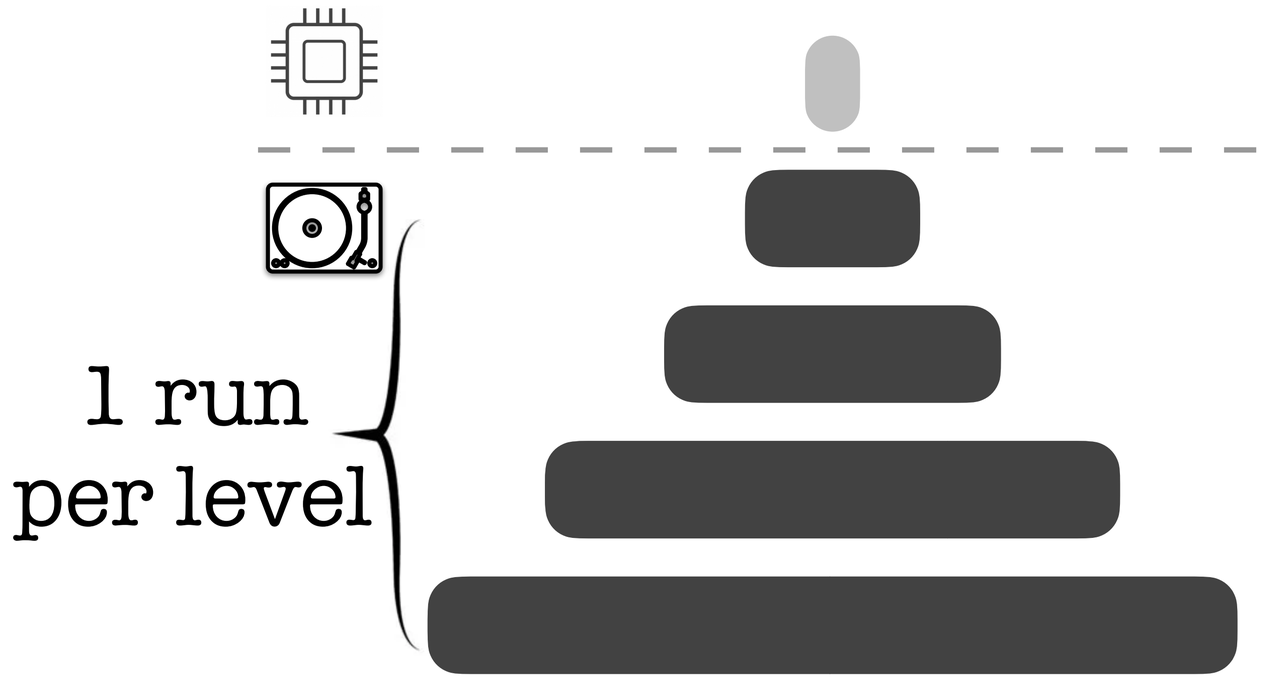


- good read performance
- good space amplification
- high write amplification

Data **Layout**

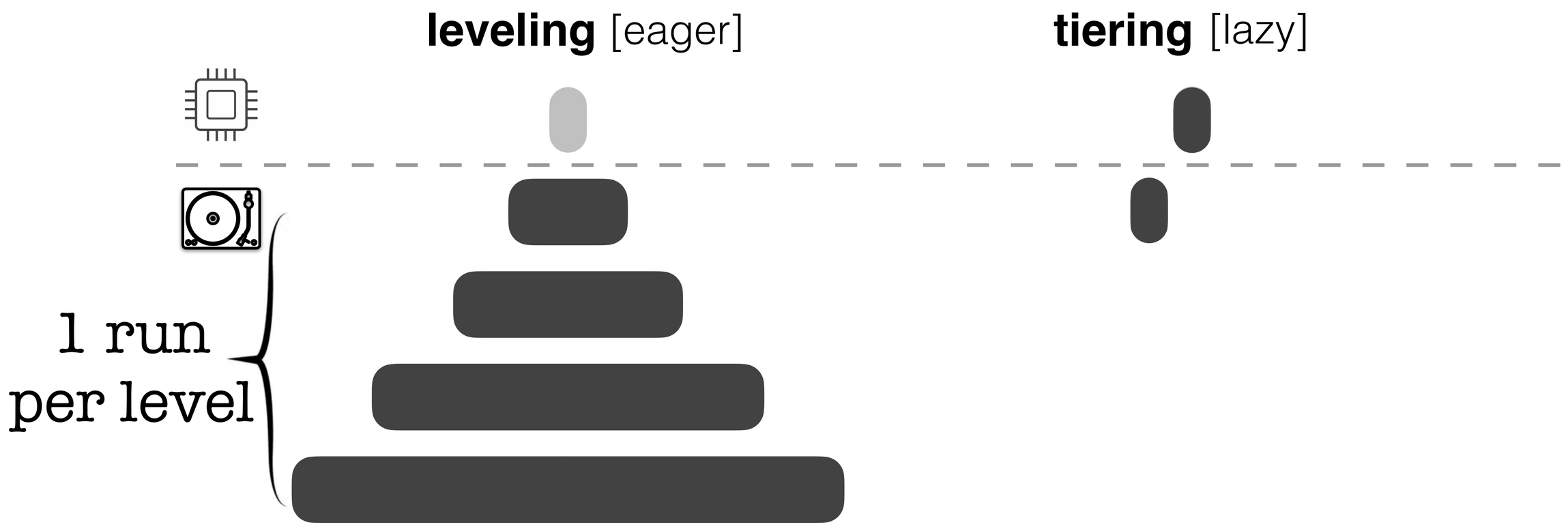
leveling [eager]

tiering [lazy]



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

Data **Layout**

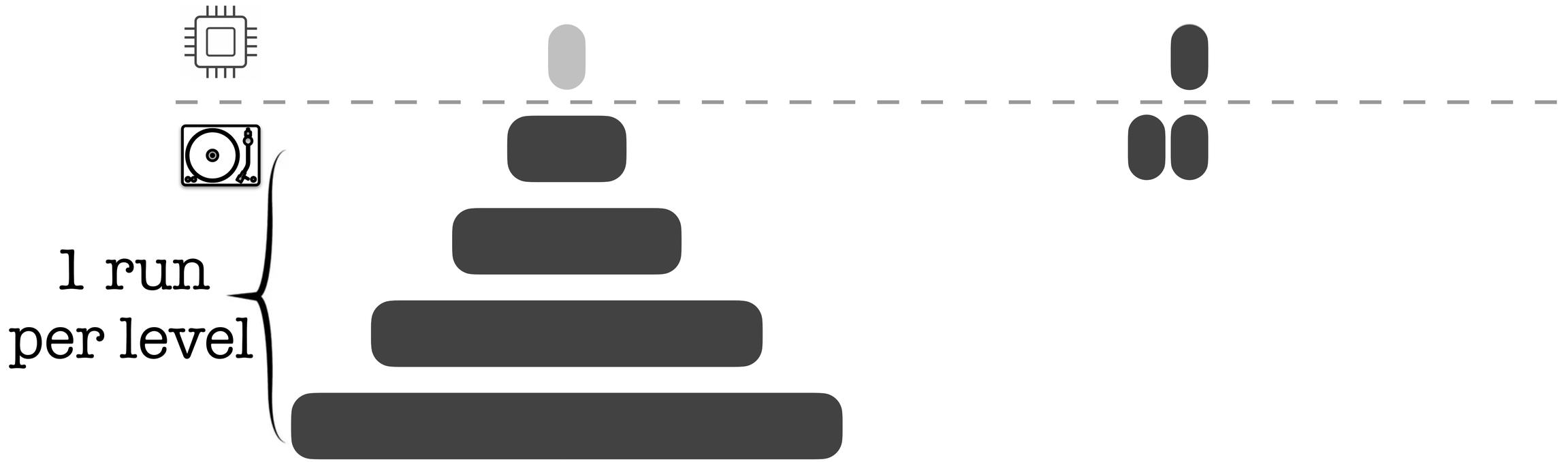


- good read performance
- good space amplification
- high write amplification

Data **Layout**

leveling [eager]

tiering [lazy]

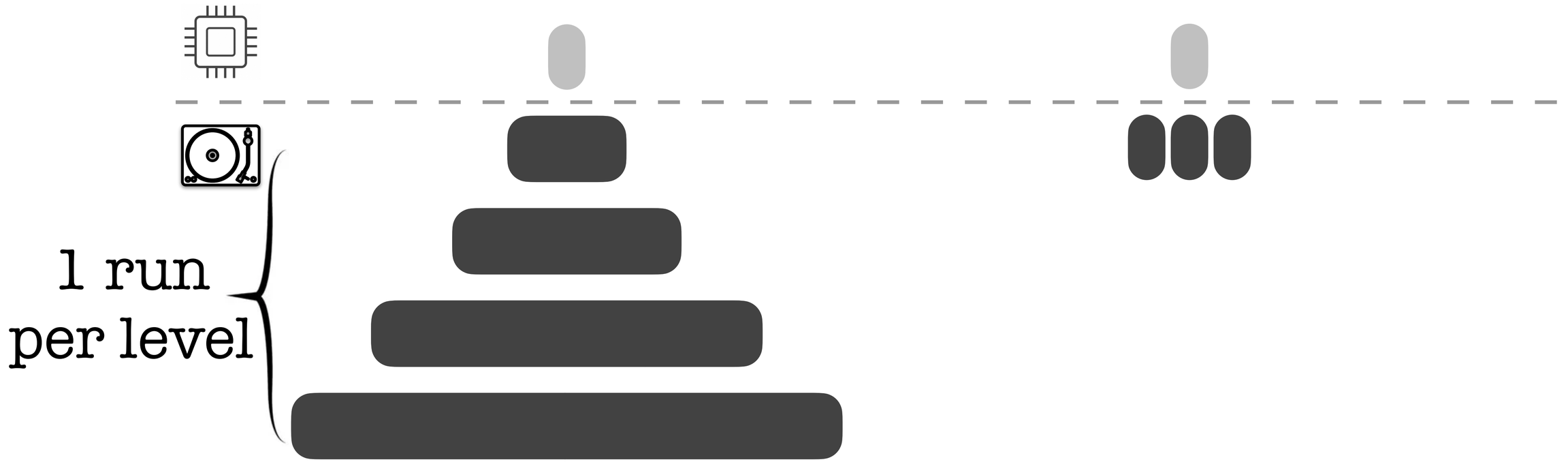


- good read performance
- good space amplification
- high write amplification

Data **Layout**

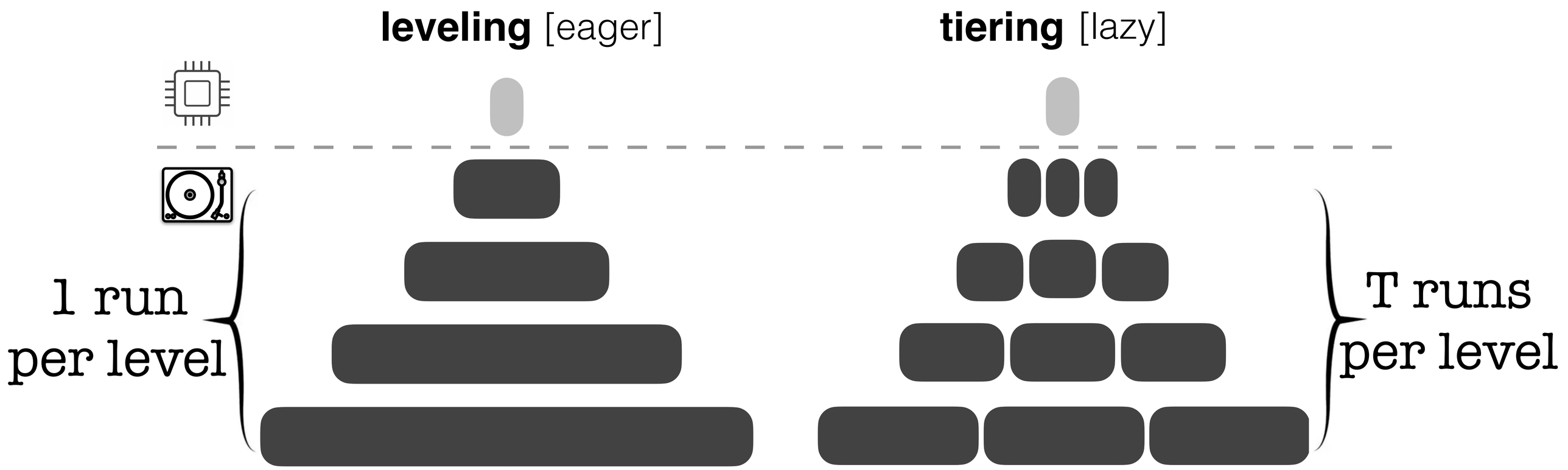
leveling [eager]

tiering [lazy]



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

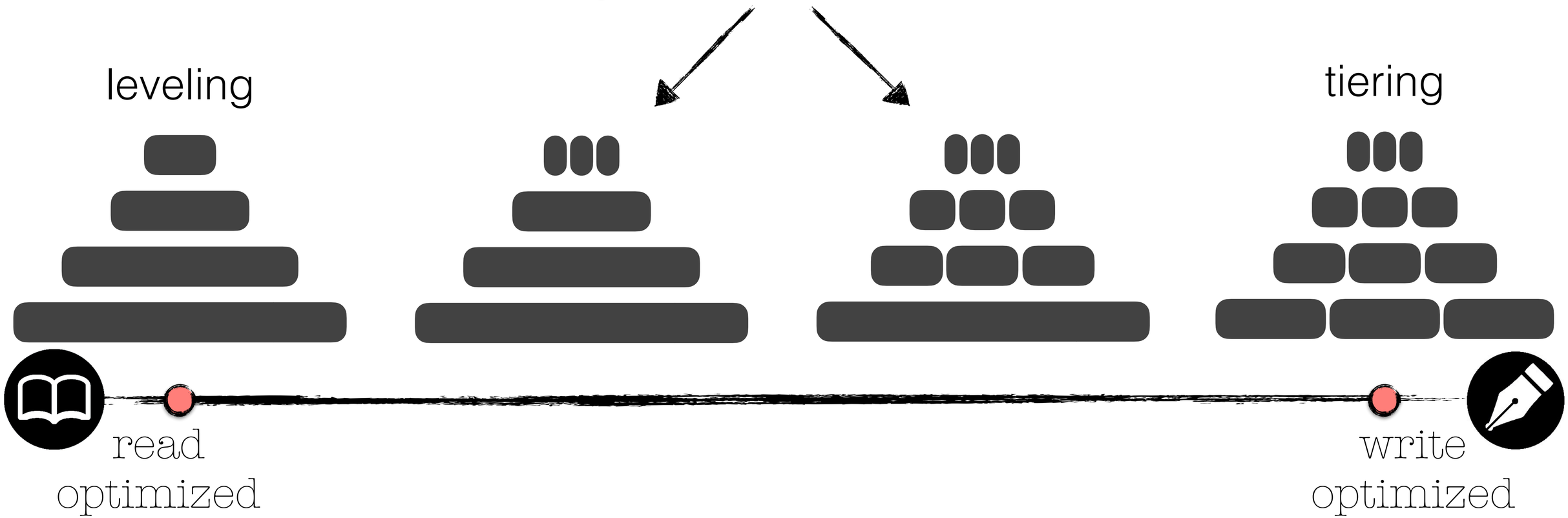
Data **Layout**



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

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

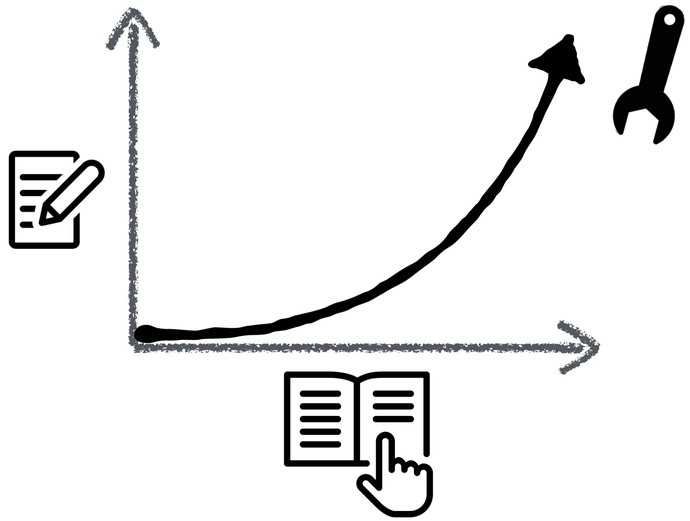
Data **Layout**


hybrid designs
leveling
tiering
read
optimized
write
optimized

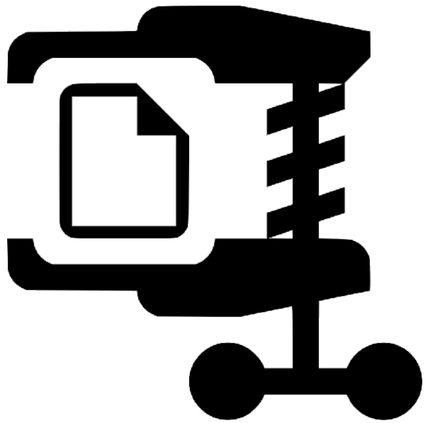
Summary

Understanding the hype!

fast writes



tunable read-write performance



good space utilization

Next time in COSI 167A

More on LSMs

Queries in **LSM-trees**

Cost analysis

[P] "Monkey: Optimal Navigable Key-Value Store", *SIGMOD*, 2017

TECHNICAL QUESTION 2 **When does a tiered LSM-tree behave similarly to a leveled LSM-tree? What is the key contribution of the paper?**

[B] "The LSM Design Space and its Read Optimizations", *ICDE*, 2023

COSI 167A
Advanced Data Systems

Class 5
Introduction to LSM-Trees

Prof. Subhadeep Sarkar

Brandeis UNIVERSITY

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

lab