COSI 167A
Advanced Data Systems

Class 12
Efficient Deletes in LSM-Engines

Prof. Subhadeep Sarkar

Brandeis UNIVERSITY

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

lab

Class **logistics**

and administrivia

The **mid-semester project report** is due in **3 weeks** (**Nov 8**).

Start working on the project and use the **office hours**.

First student presentation on **Friday, Oct 18**.

Discuss the slides with me **a week before the presentation.**

A **good presentation**

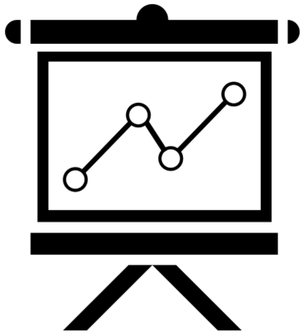
The three key ingredients

tell a **story**

highlight the problem
why it is worth solving
solution intuition & methodology
constructive critisism

prepare a **speech**

prepare a narrative
make sure it flows
rehearse!



presentation

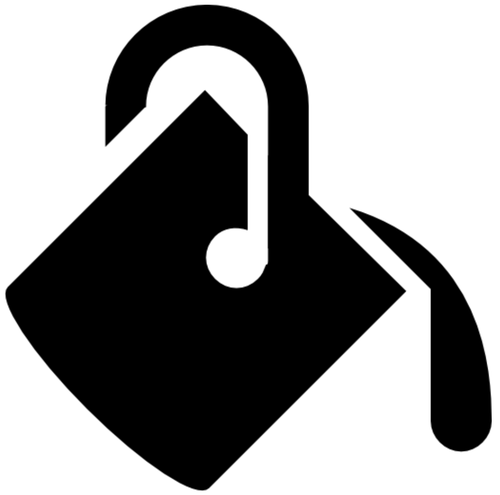
make it engaging!

An **engaging presentation**

Think of what keeps you engaged!

don’t use bullets

1 message per slide



1/2 colors

Today in COSI 167A

What's on the cards?

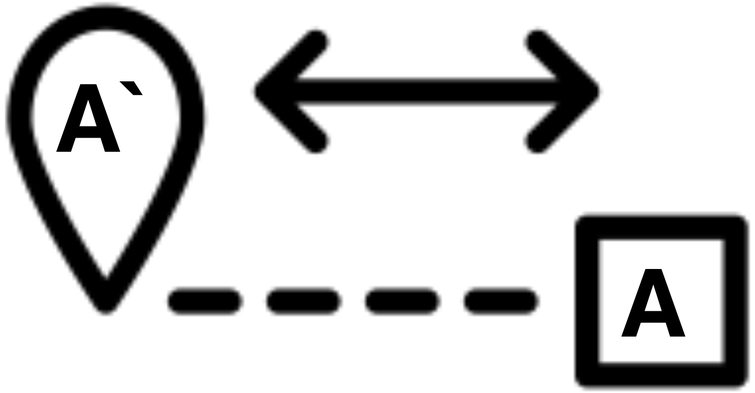
deletes in LSM

the problems with **out-of-place deletes**

building **deletion-compliant data systems**

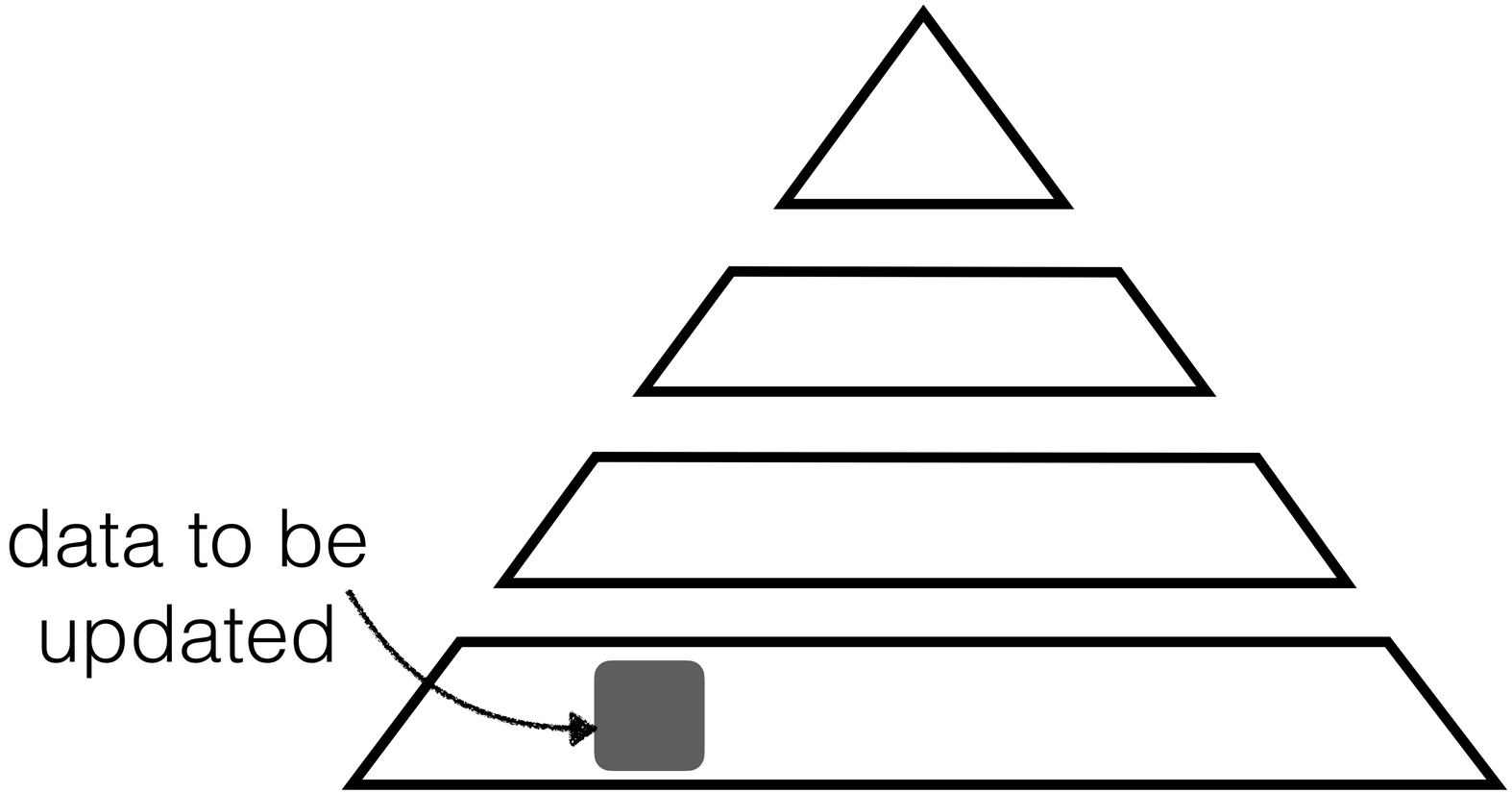
Ingestion-Optimized Systems

batched inserts

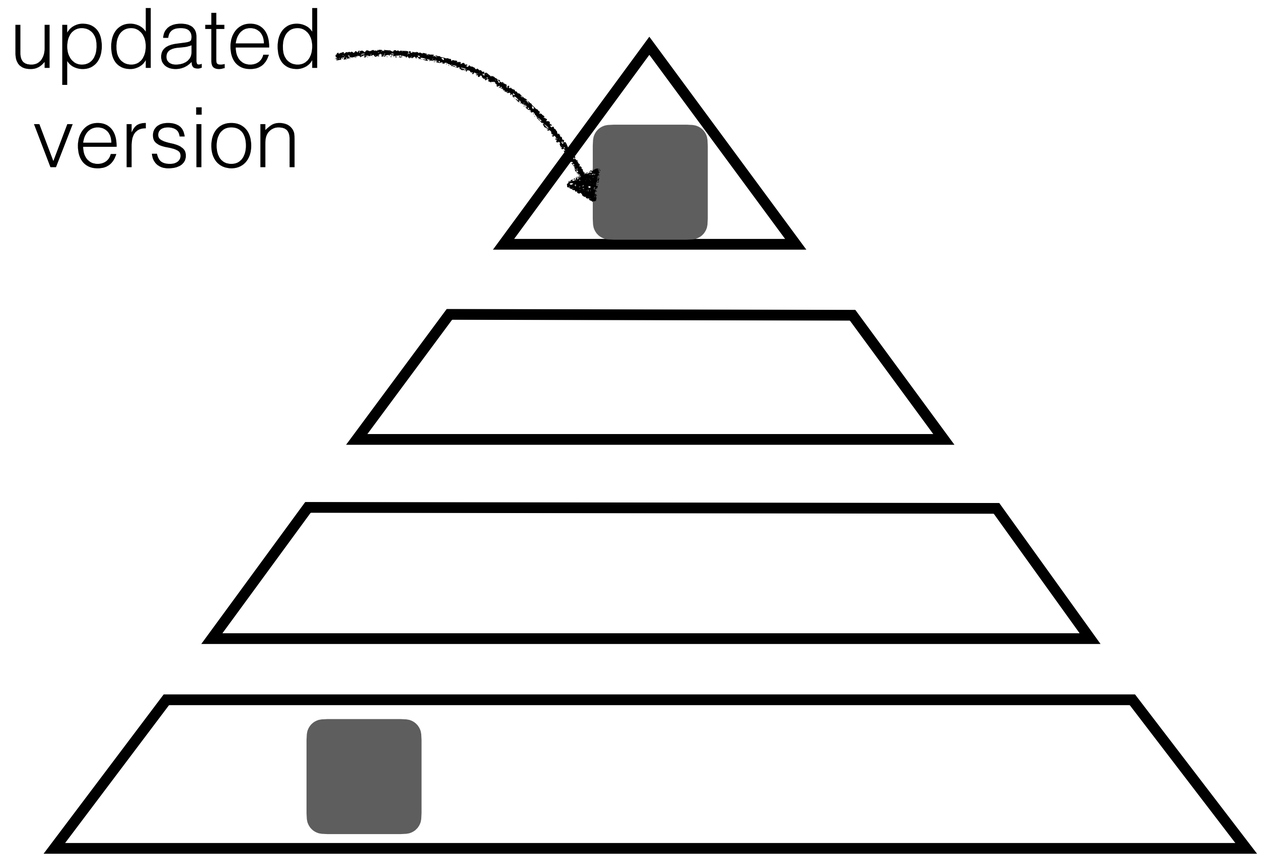


out-of-place
deletes/updates

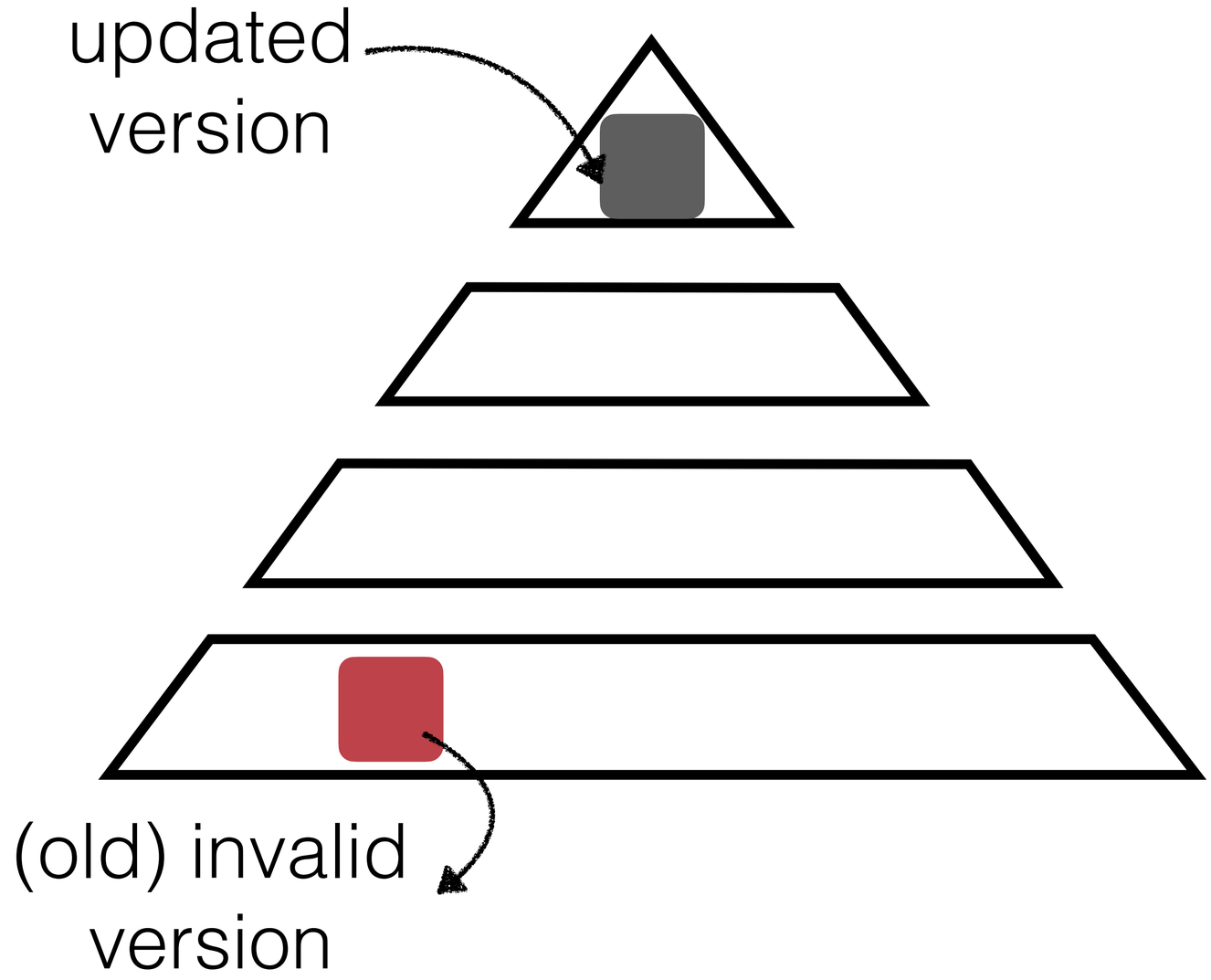
Out-of-place Deletes/Updates



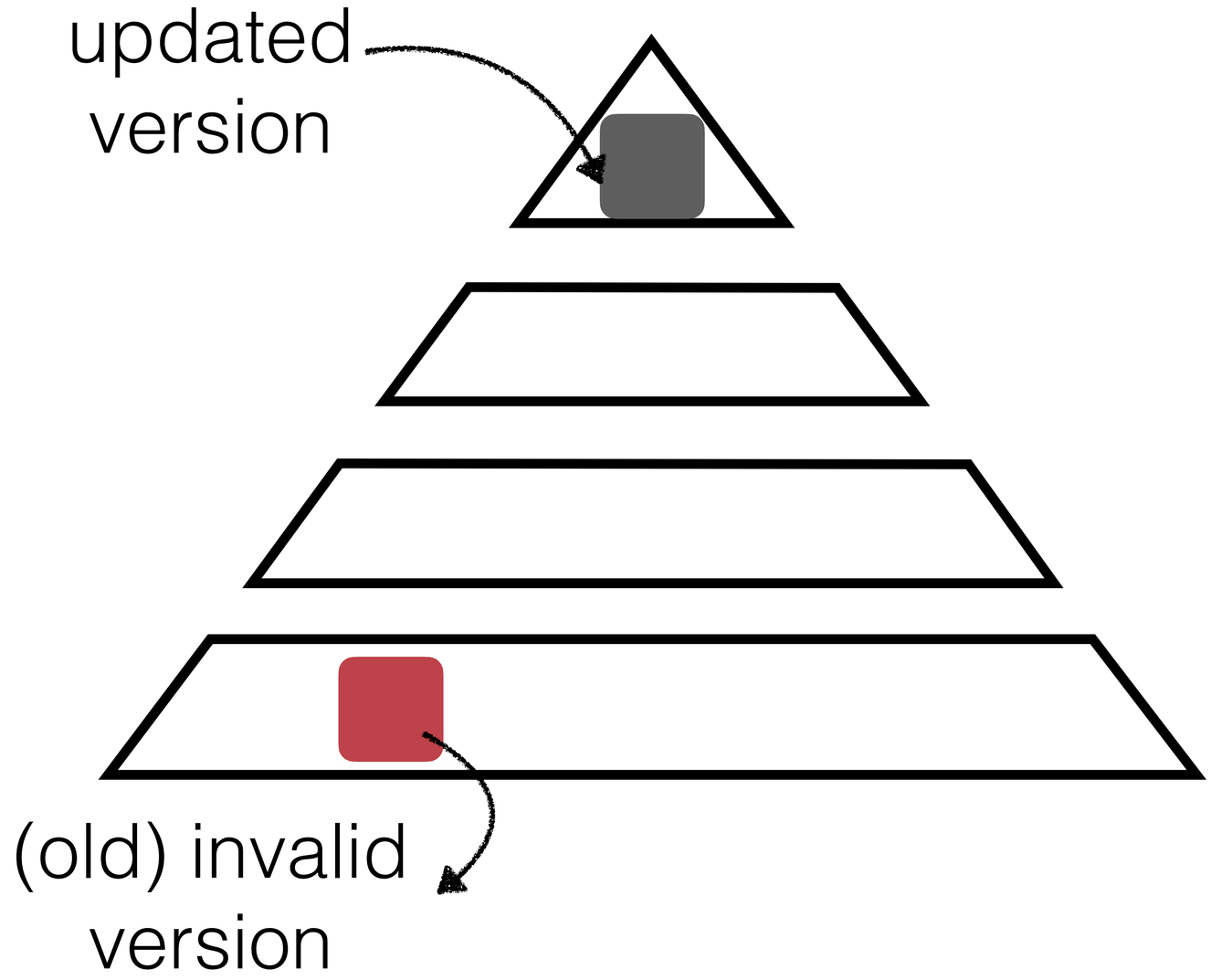
Out-of-place Deletes/Updates



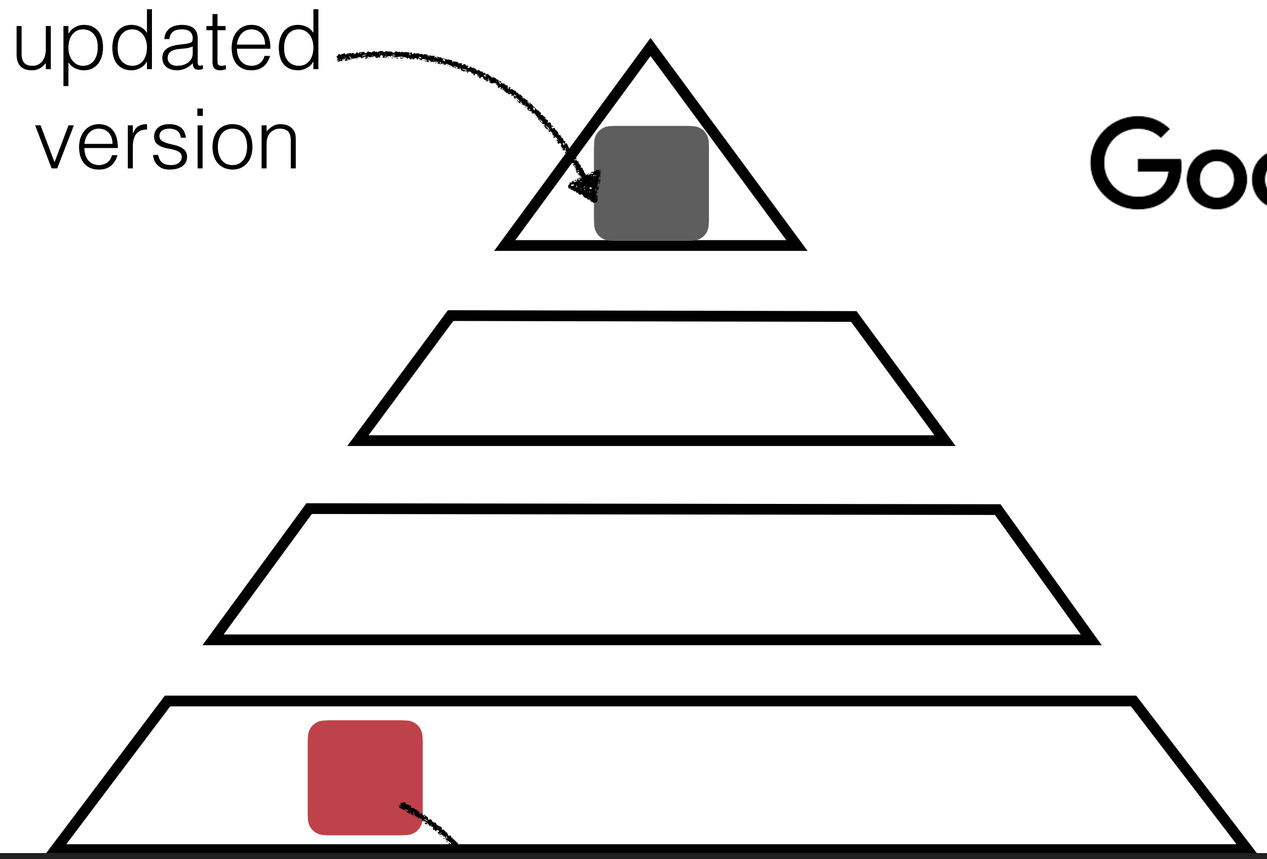
Out-of-place Deletes/Updates



Out-of-place Deletes/Updates



Out-of-place Data Systems



Google Meta amazon

THE APACHE® SOFTWARE FOUNDATION

SAP®

mongoDB

$22B, growing at the

Hidden Cost: does not scale with deletes!

Today's **talk**

Deletes in LSMs

Lethe: A Tunable Delete-Aware LSM Engine

Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, Manos Athanassoulis
Boston University
{ssarkar1, papon, dstara, mathan}@bu.edu

presented at **SIGMOD 2020**

Today's **talk**

Deletes in LSMs

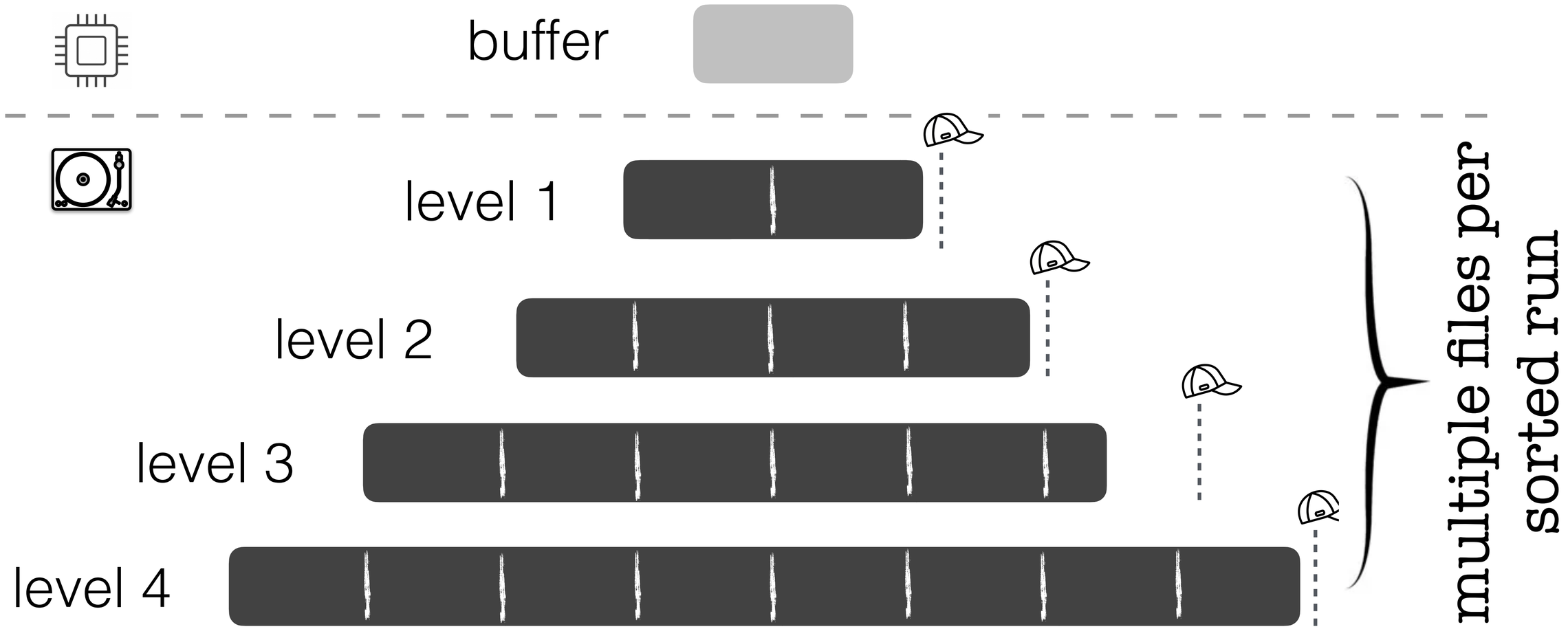
***Lethe*: A Tunable Delete-Aware LSM Engine**

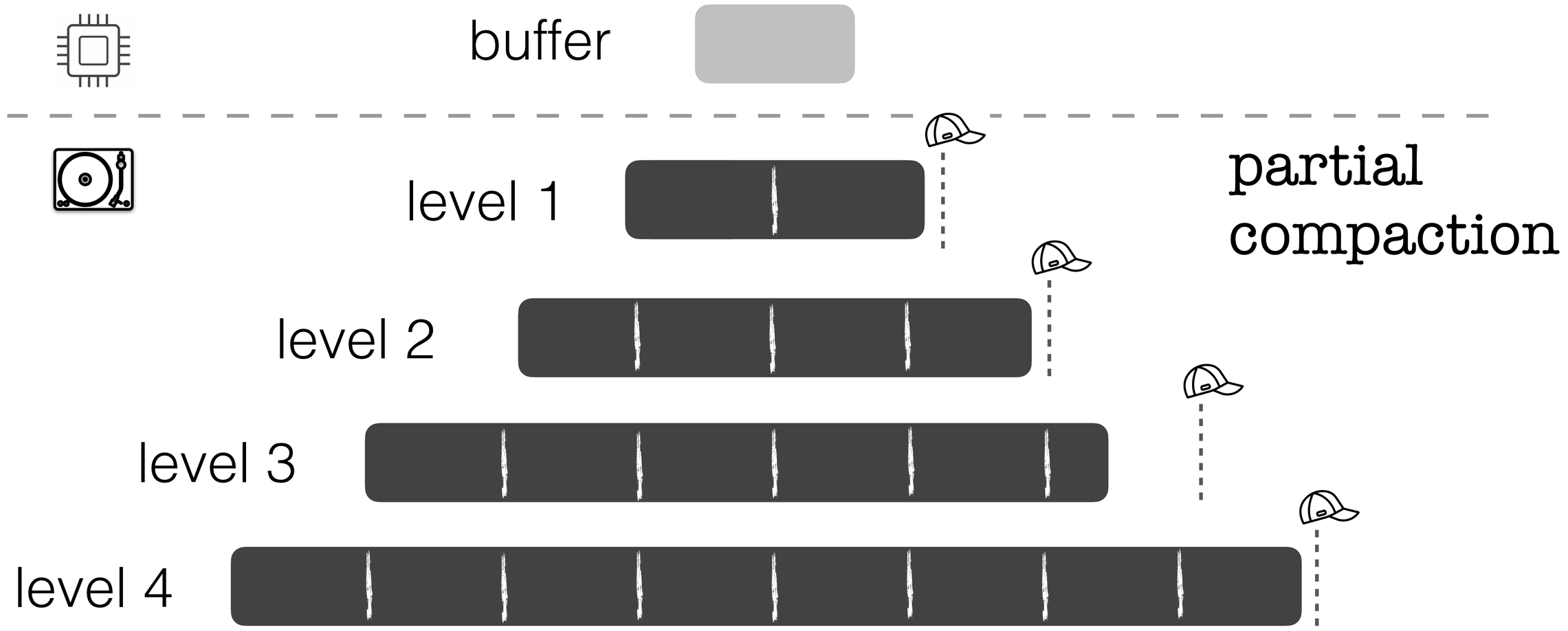
Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, Manos Athanassoulis
Boston University
{ssarkar1, papon, dstara, mathan}@bu.edu

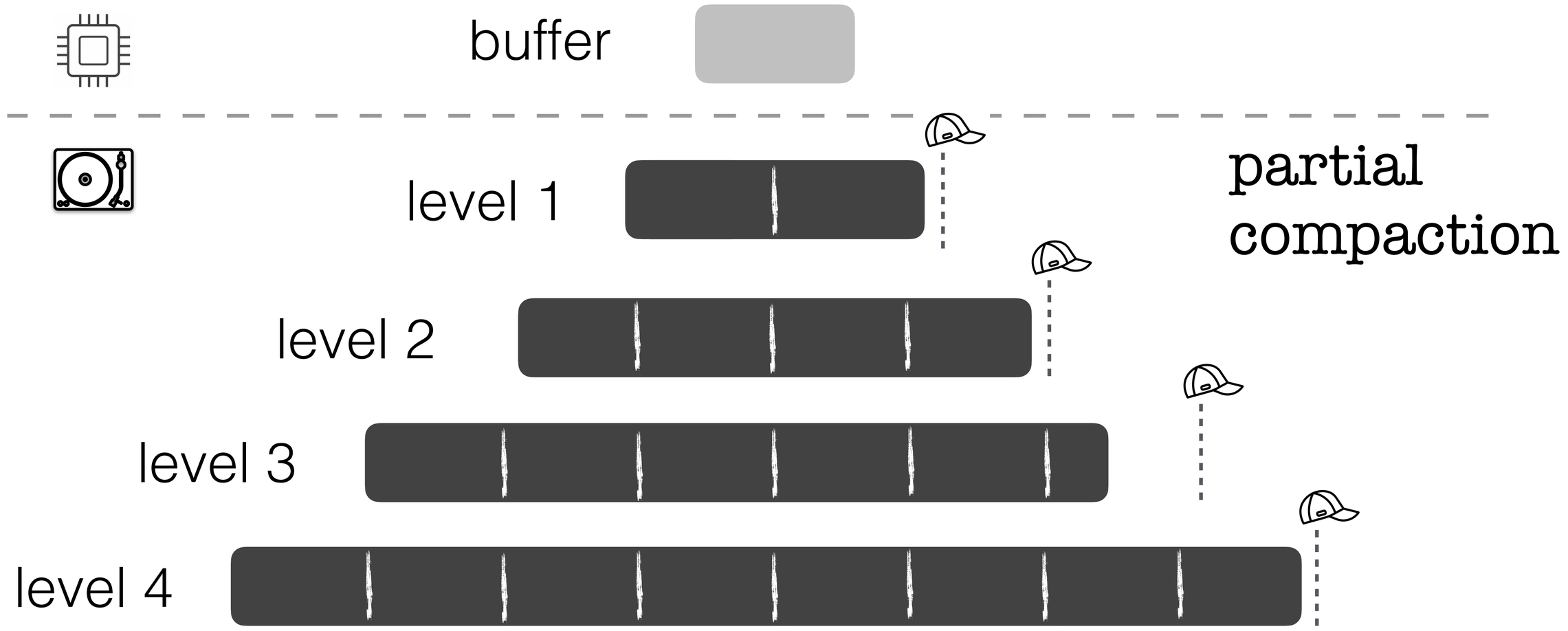
presented at **SIGMOD 2020**

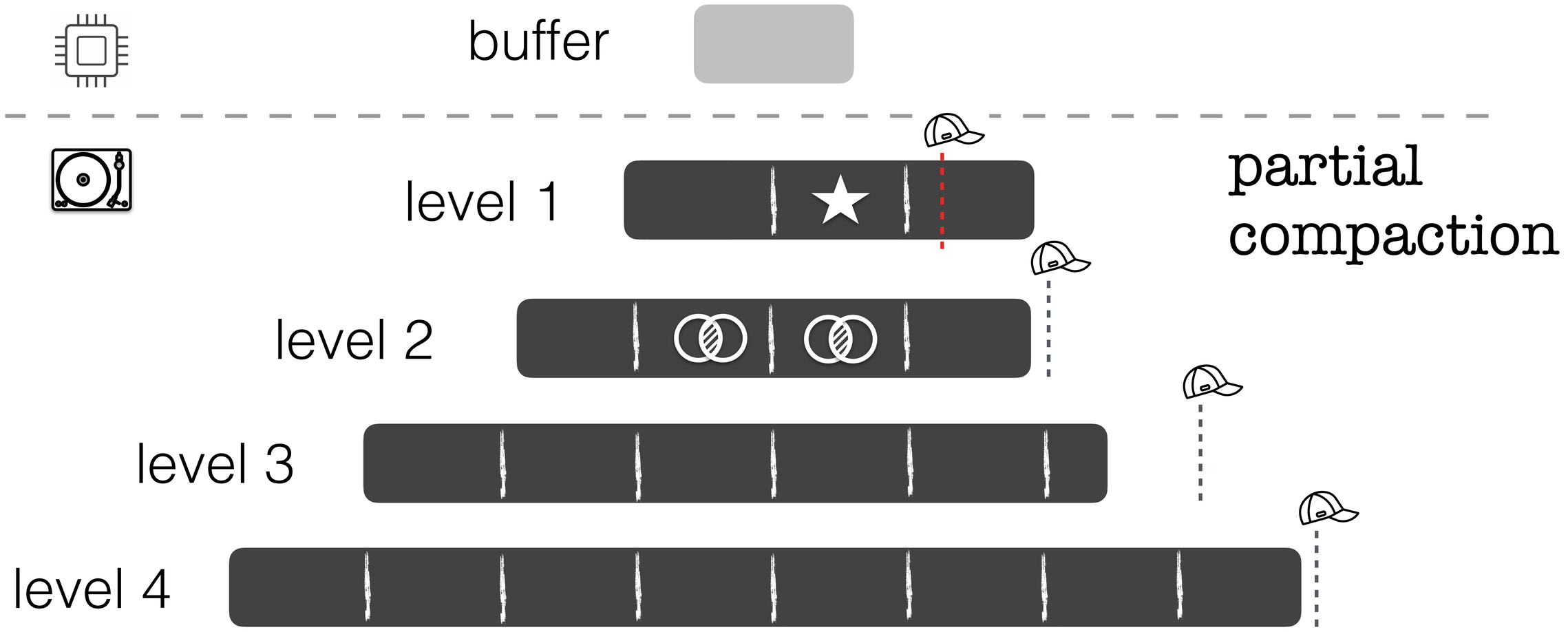
Lethe [\'lē-thē\ ]

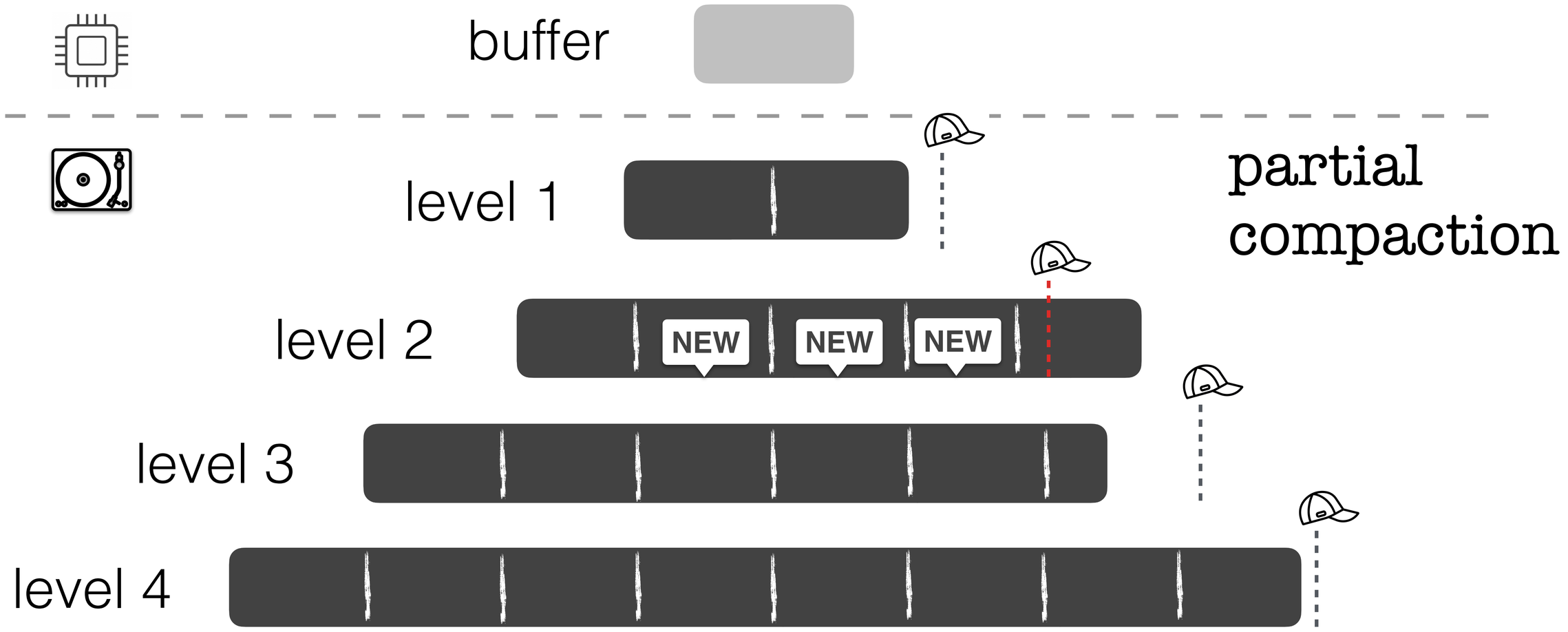
n: the goddess of forgetfulness

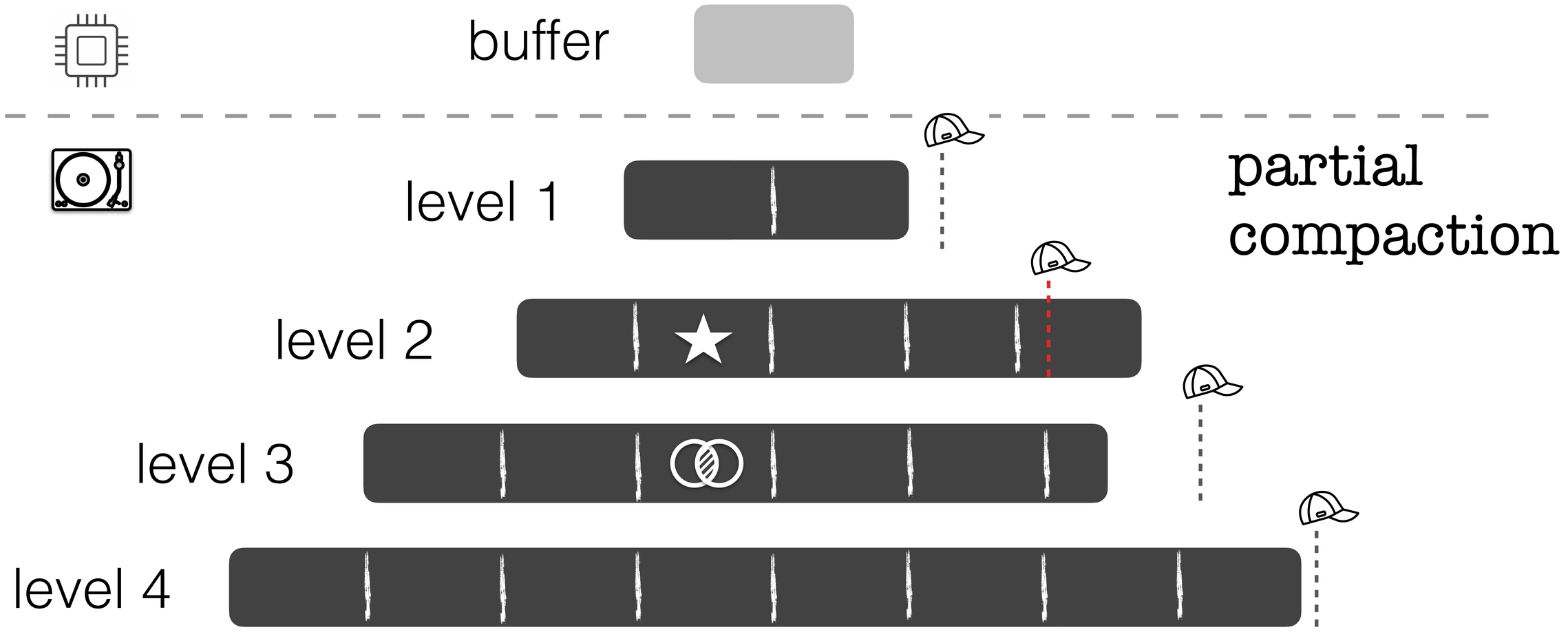

buffer
level 1
level 2
level 3
level 4
multiple files per sorted run

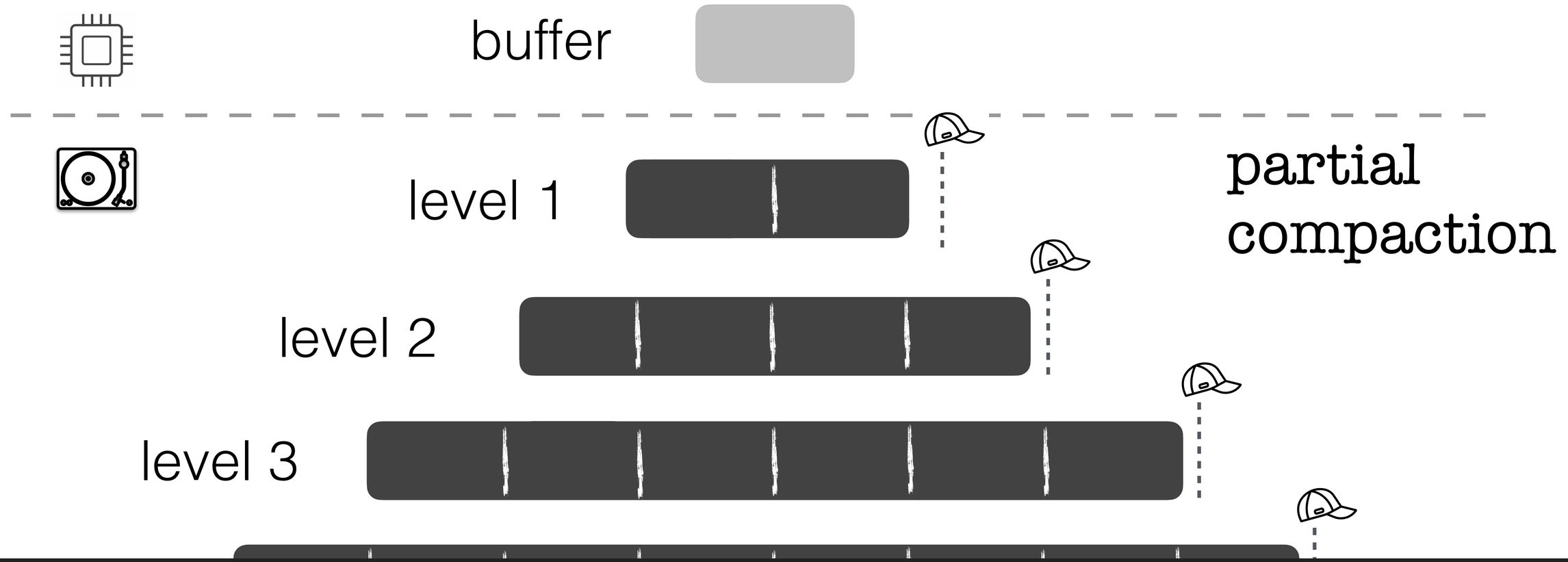

buffer
level 1
level 2
level 3
level 4
partial
compaction


buffer
level 1
level 2
level 3
level 4
partial
compaction


buffer
level 1
level 2
level 3
level 4
partial
compaction


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

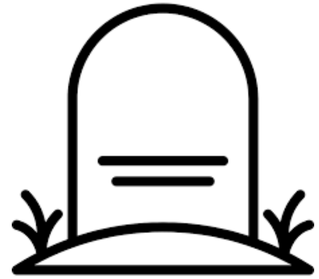

buffer
level 1
level 2
level 3
level 4
partial
compaction



So, what about deletes?

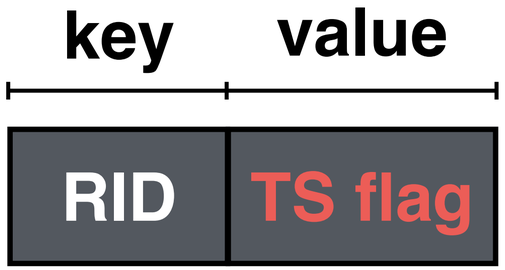
Deletes in LSMs

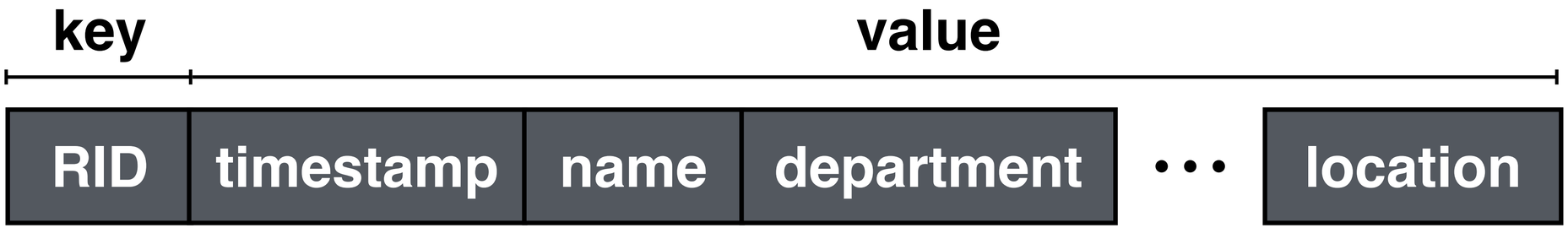
delete := insert tombstone



Deletes in LSMs

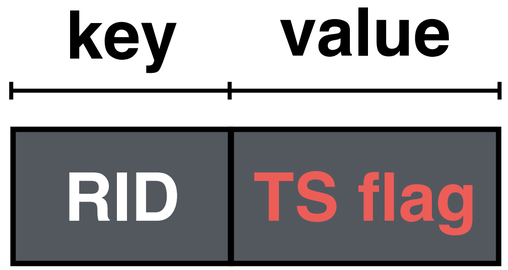
delete := insert tombstone

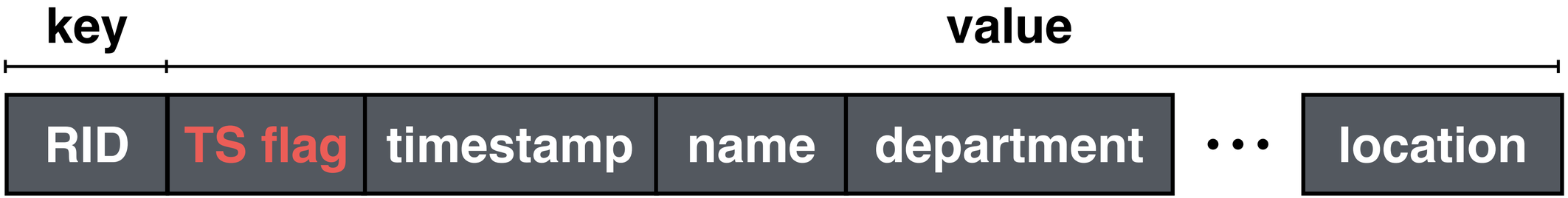




Deletes in LSMs

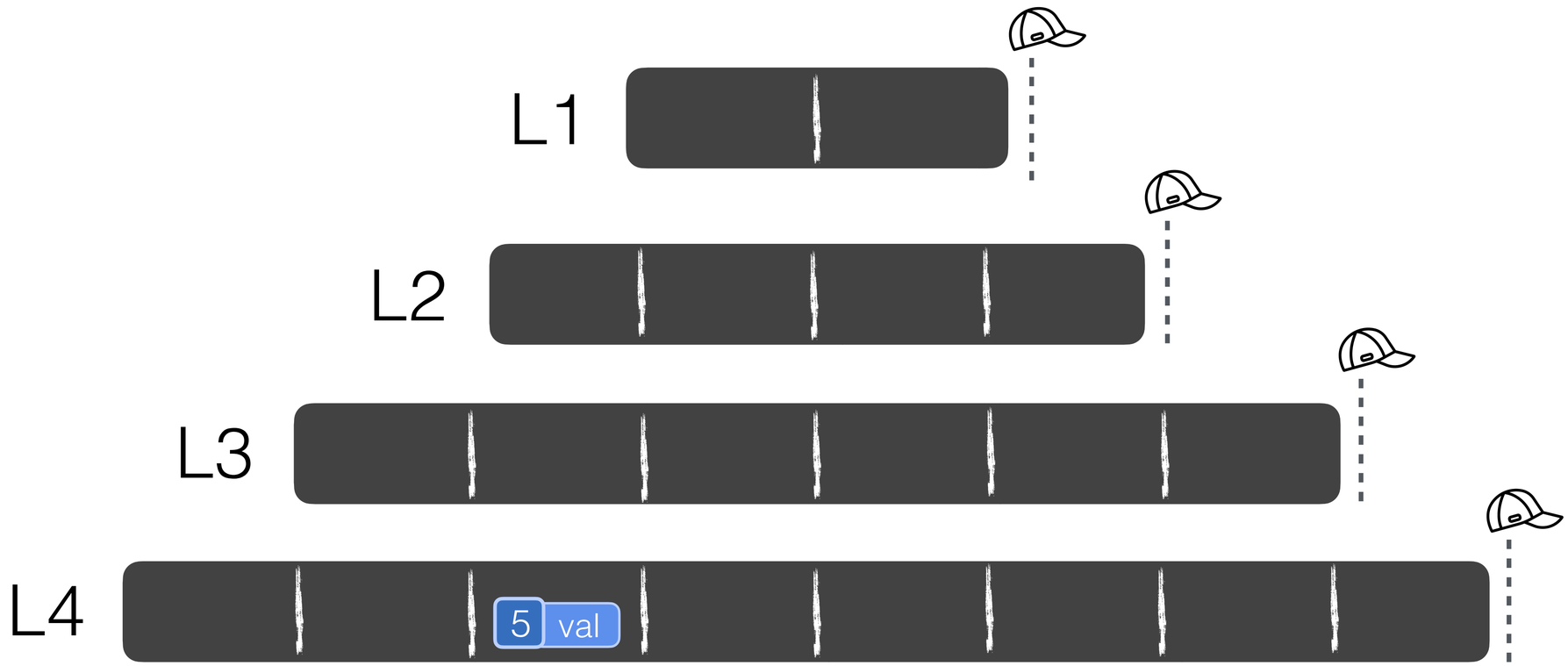
delete := insert tombstone





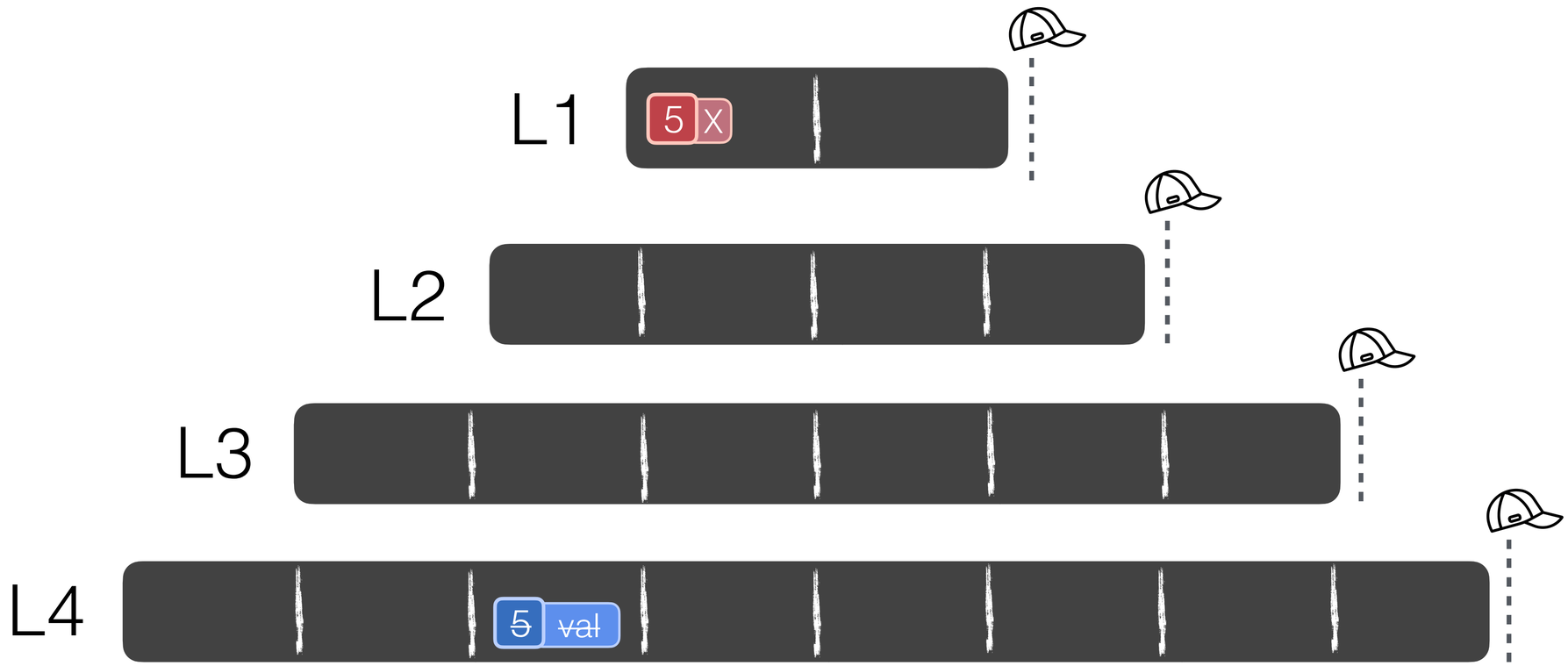
Deletes in LSMs

delete(5)



Deletes in LSMs

delete(5)



Deletes in LSMs

delete(5)

great for inserts

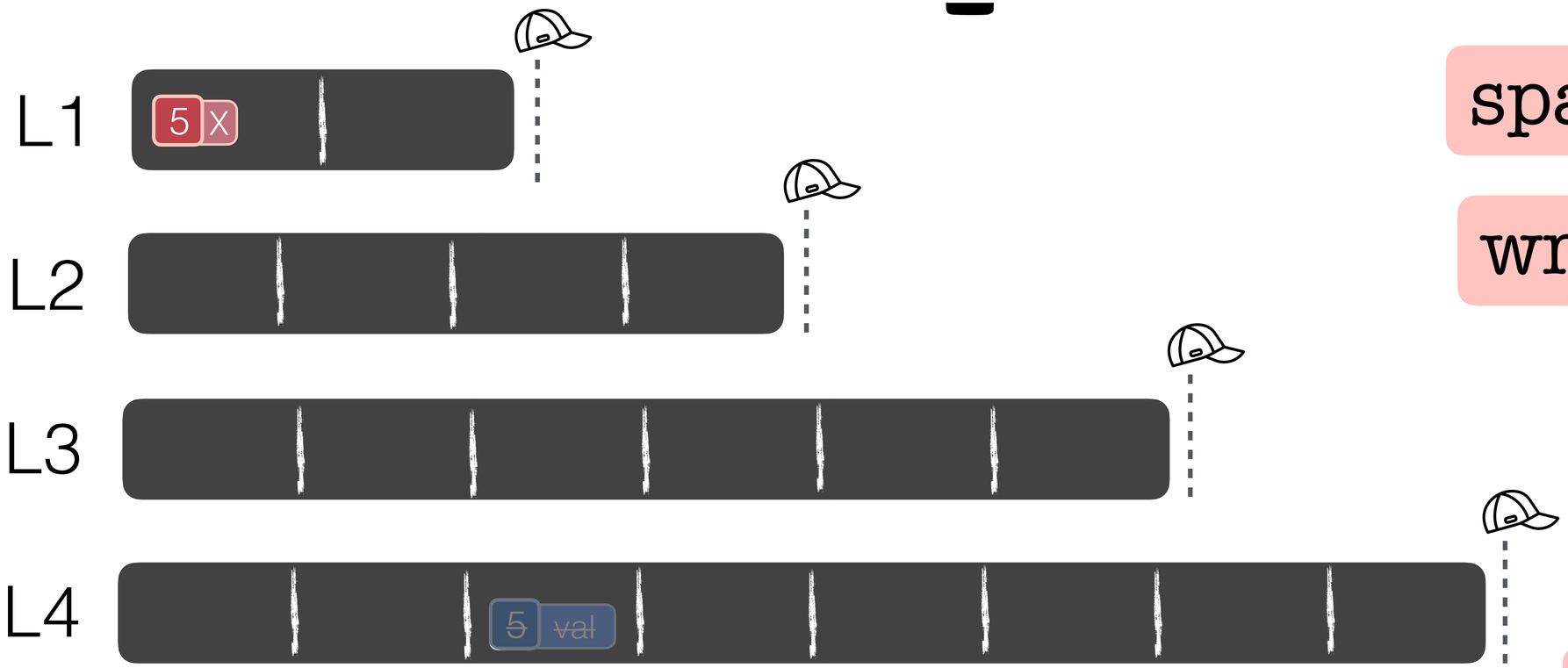
great for queries

What about **queries**?

Deletes in LSMs

Thought Experiment 1

What's the problem with **logical deletes**?



space amplification

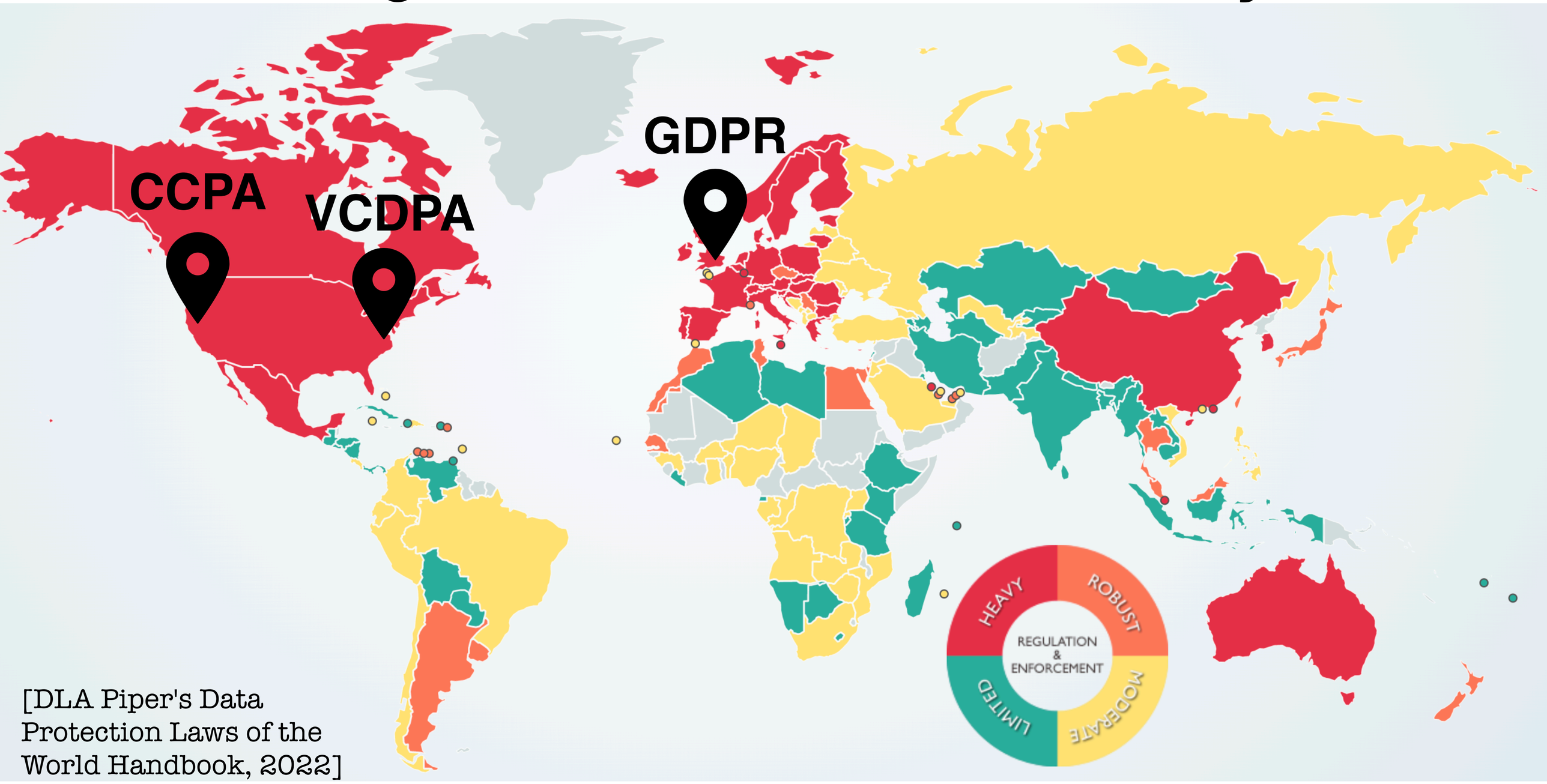
write amplification

poor reads

increased cost

loss of privacy!

Logical Deletes & **Data Privacy**



[DLA Piper's Data Protection Laws of the World Handbook, 2022]

GDPR
(EU, UK)

Right to be forgotten

CCPA
(California)

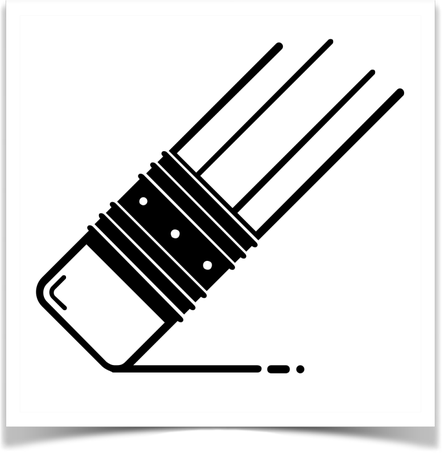
Right to delete

VCDPA
(Virginia)

Deletion right

timely
deletes

+

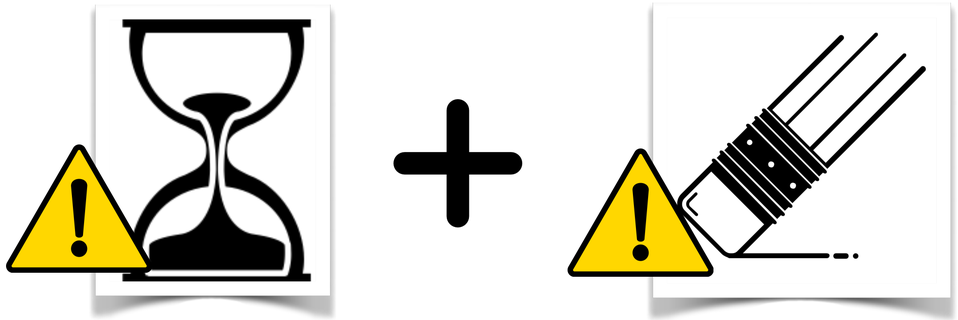


persistent
deletes

Right to be forgotten

Right to delete

Deletion right



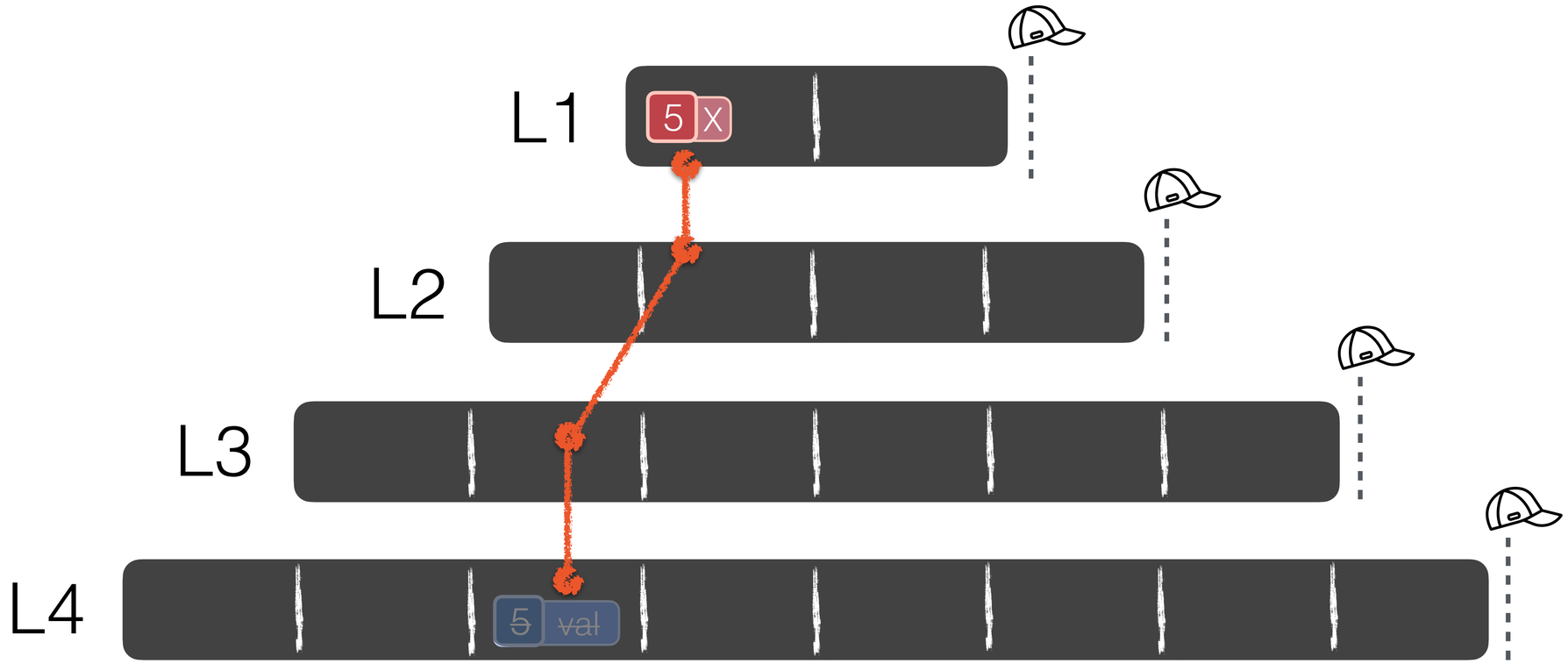
timely deletes + persistent deletes

Even years later, Twitter doesn't delete your direct messages

Zack Whittaker, Natasha Lomas / 1:57 PM EST • February 15, 2019

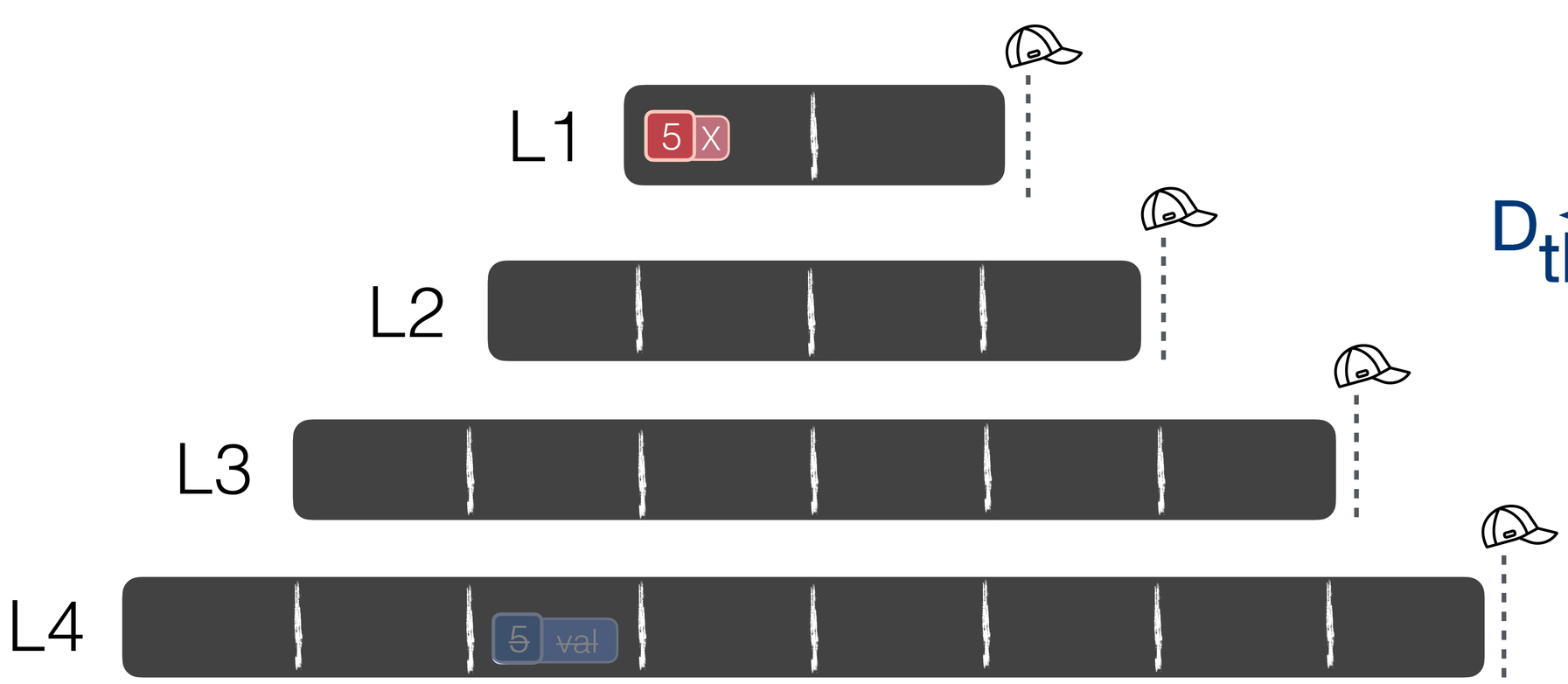
Problem: Persisting Deletes Timely

delete(5) within threshold: D_{th}



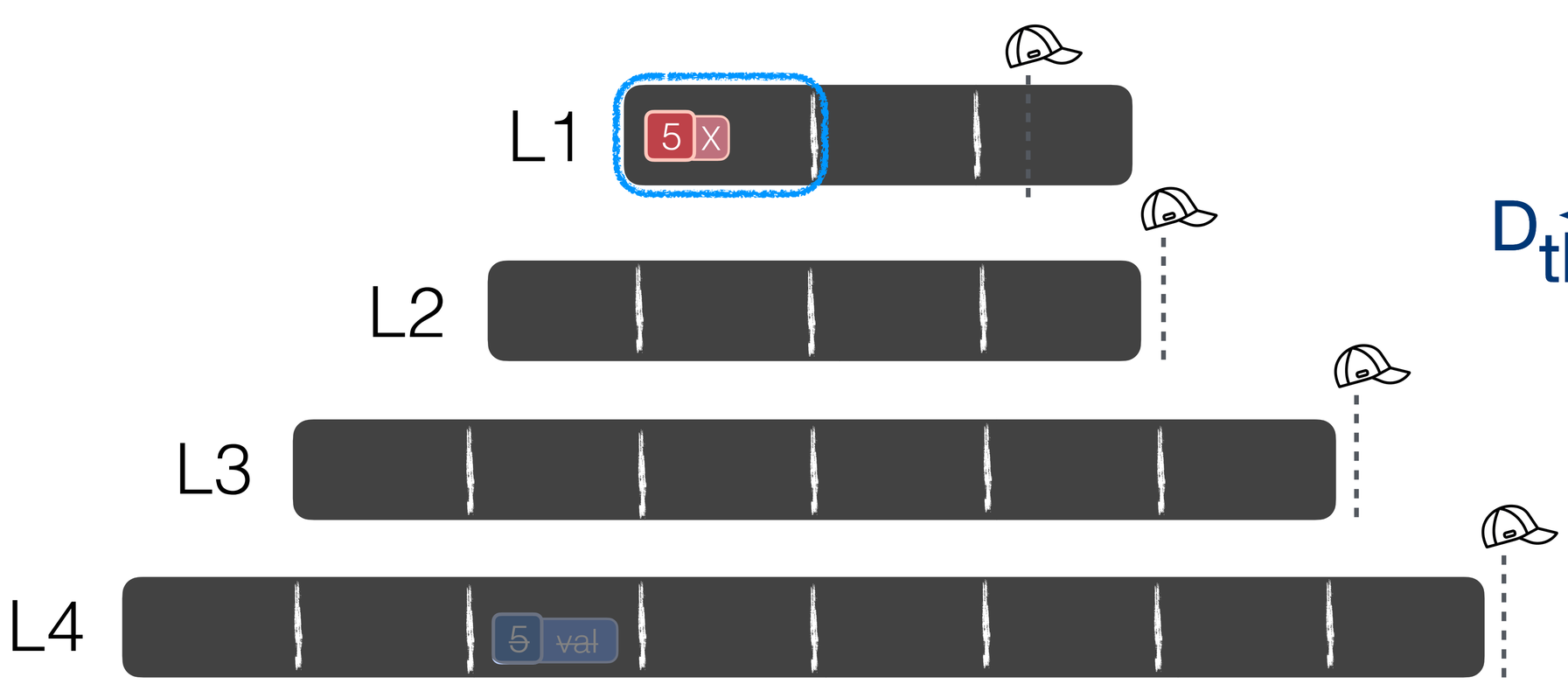
Problem: Persisting Deletes Timely

delete(5) within threshold: D_{th}



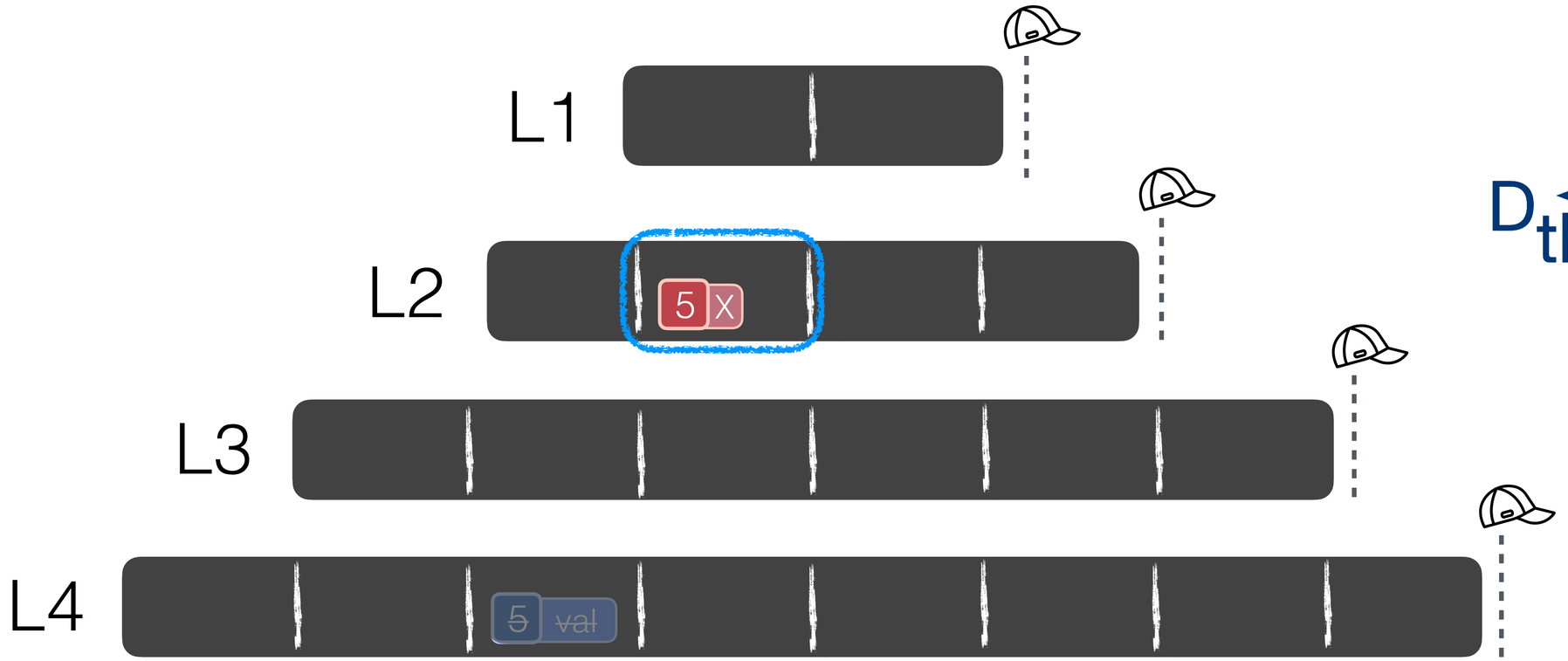
Problem: Persisting Deletes Timely

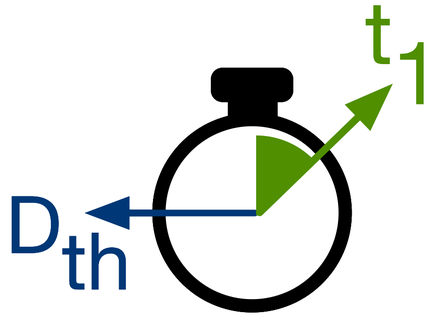
delete(5) within threshold: D_{th}



Problem: Persisting Deletes Timely

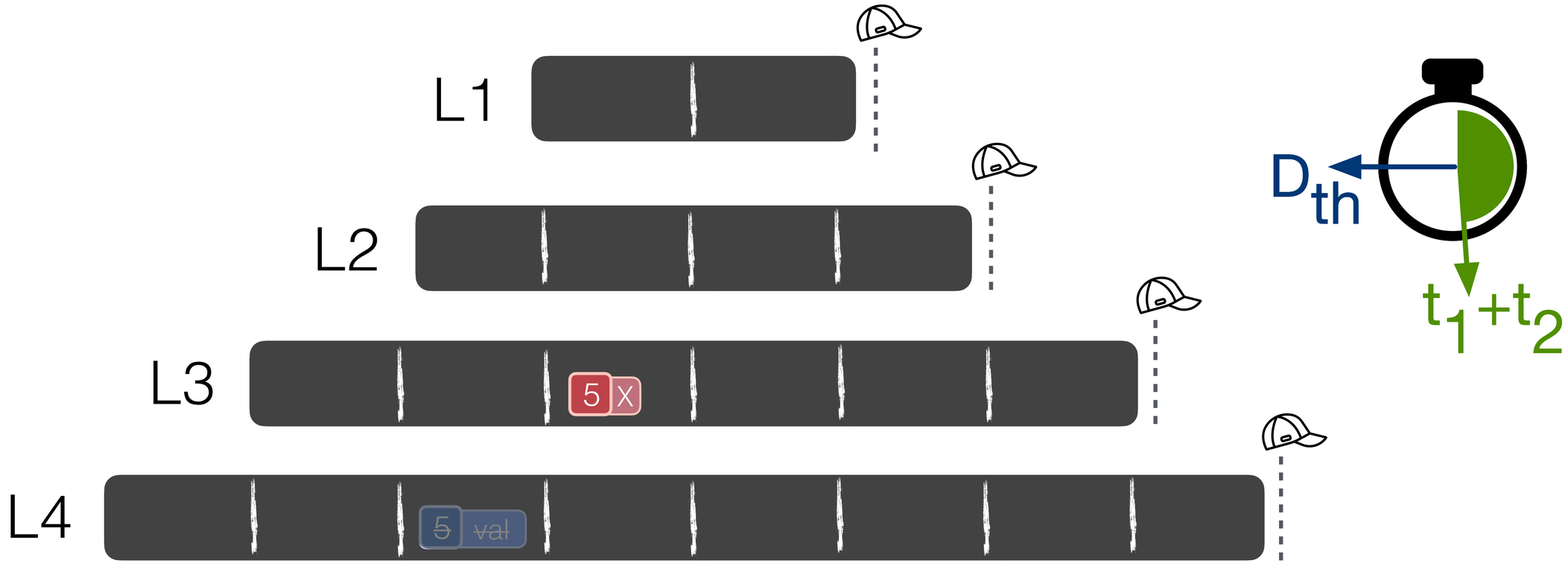
delete(5) within threshold: D_{th}





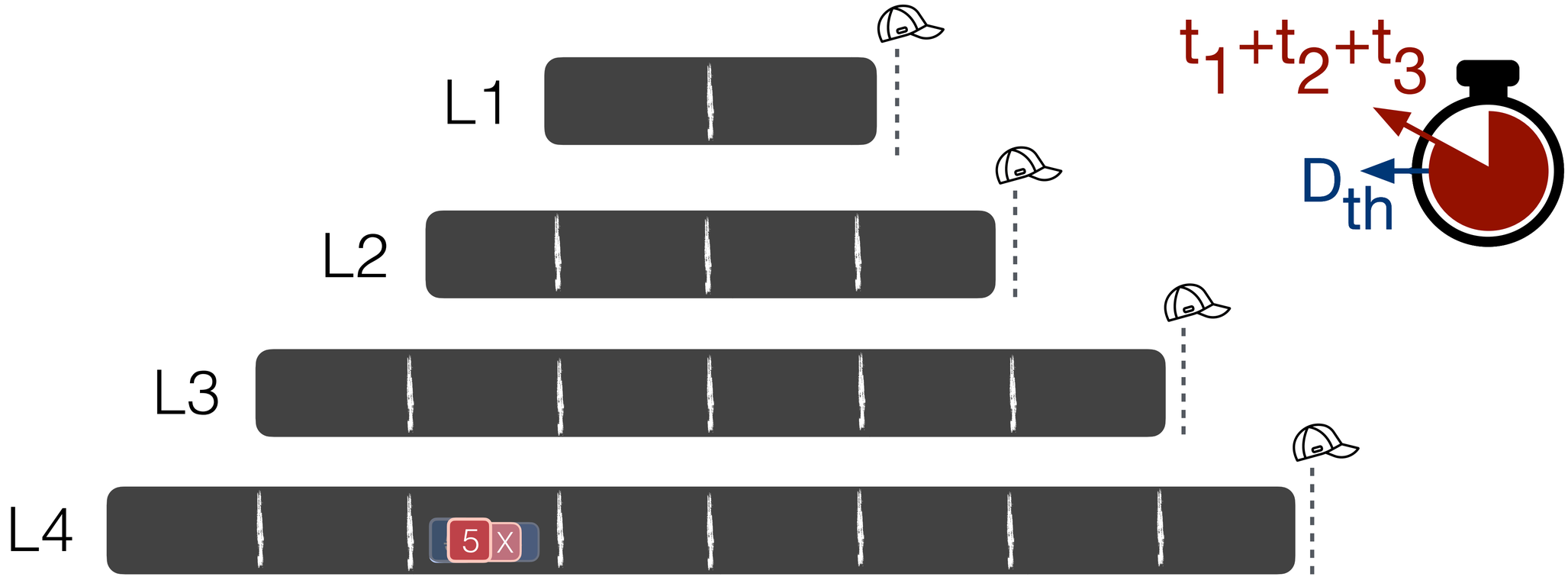
Problem: Persisting Deletes Timely

delete(5) within threshold: D_{th}



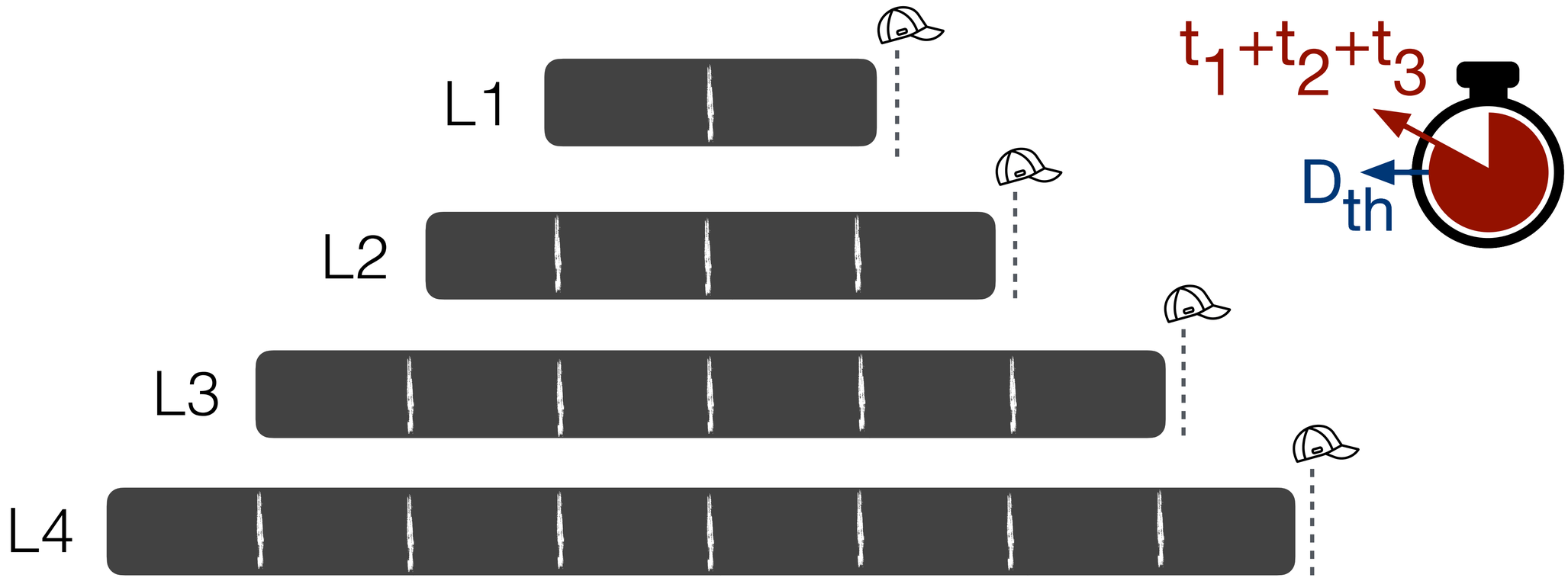
Problem: Persisting Deletes Timely

delete(5) within threshold: D_{th}



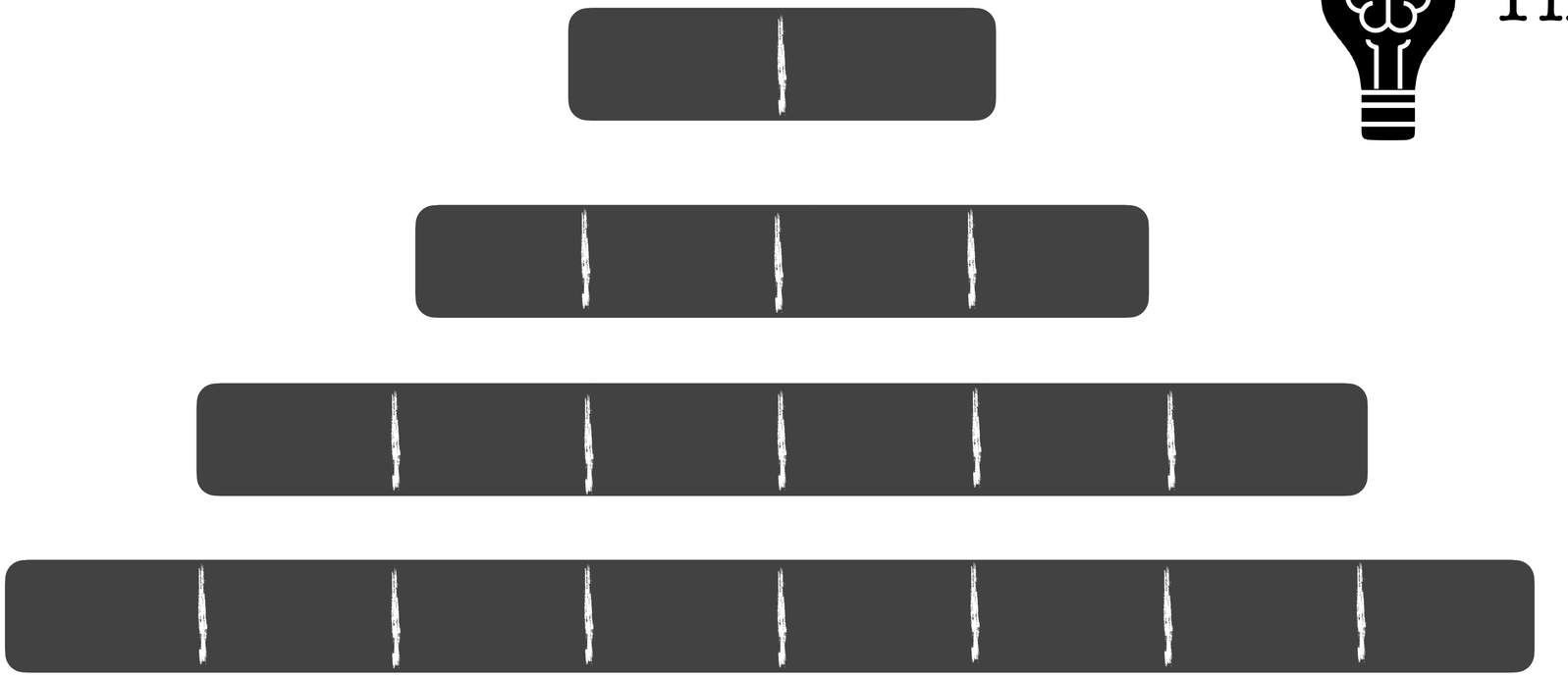
Problem: Persisting Deletes Timely

delete(5) within threshold: D_{th}



Problem: Persisting Deletes Timely

delete(5) within threshold: D_{th}



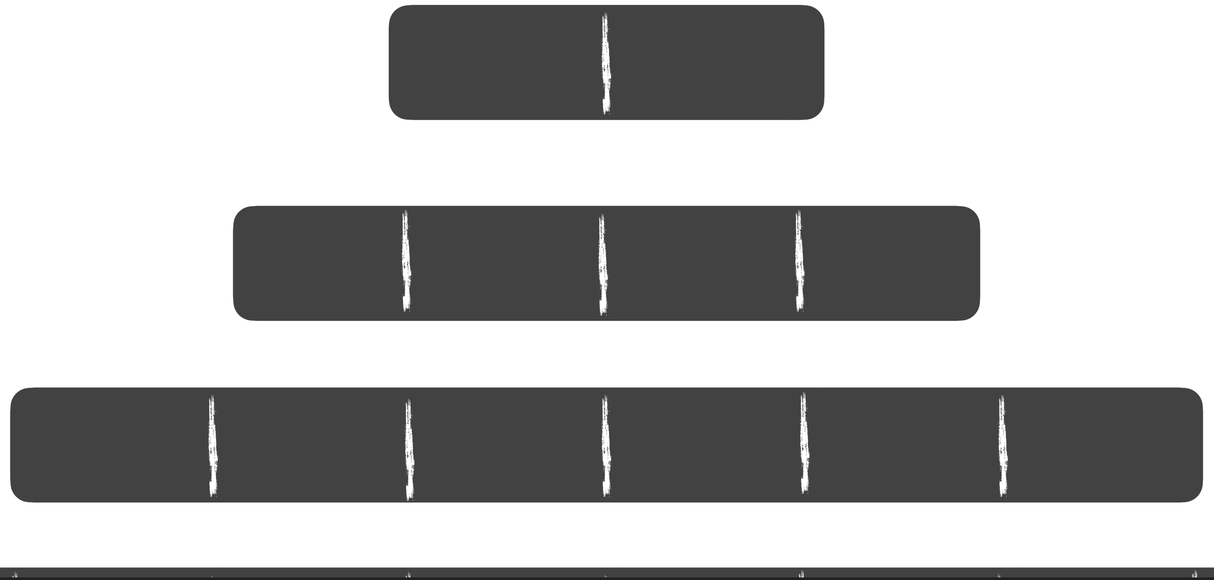
Thought Experiment 2

What drives the **delete persistence latency**?

unbounded delete persistence latency

Problem: Persisting Deletes Timely

delete(5) within threshold: D_{th}



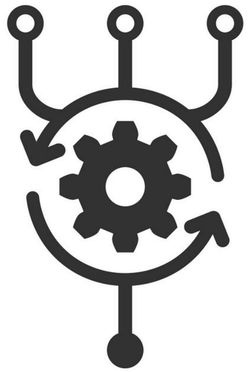
unbounded delete persistence latency

- File picking policy
- Tree shape
- Ingestion rate

Intuition: Compaction holds the key!

FADE

FAst **DE**lete



family of
compaction strategies

FAst **DE**lete

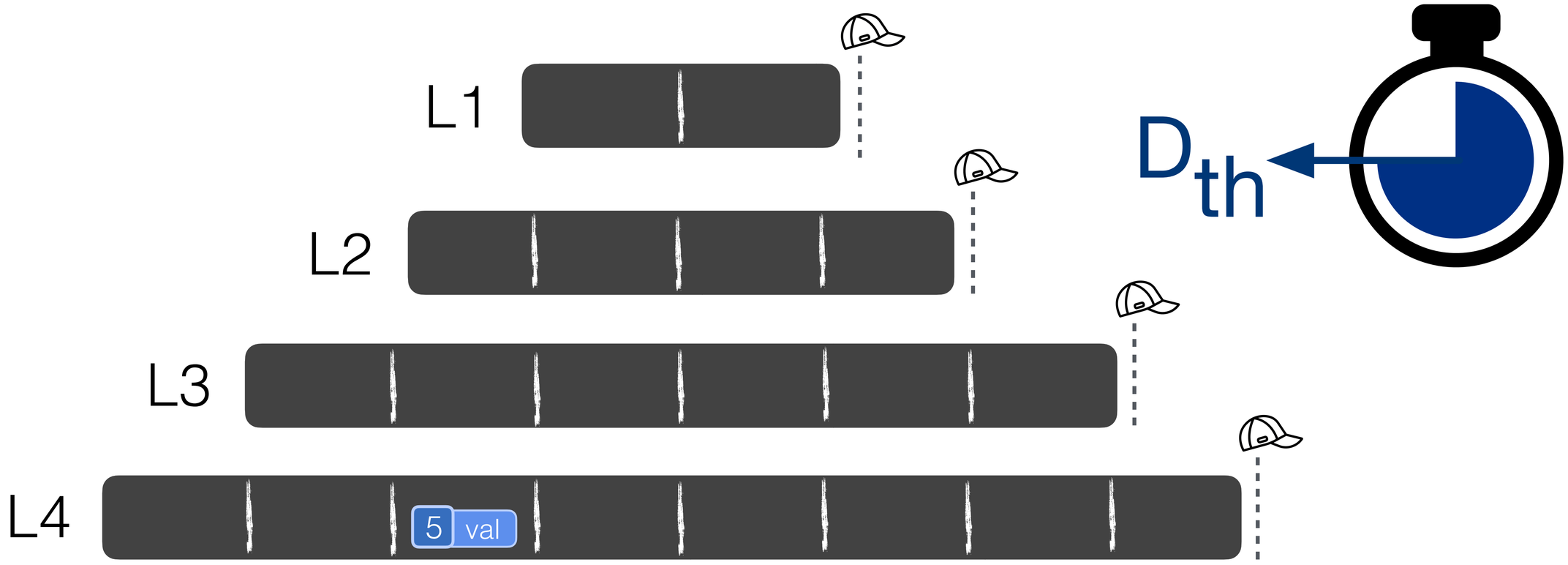
compaction trigger	compaction file picking policy
	

decompose tasks

piggyback

FAst **DE**lete

delete(5) within threshold: D_{th}



FAst **DE**lete

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

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

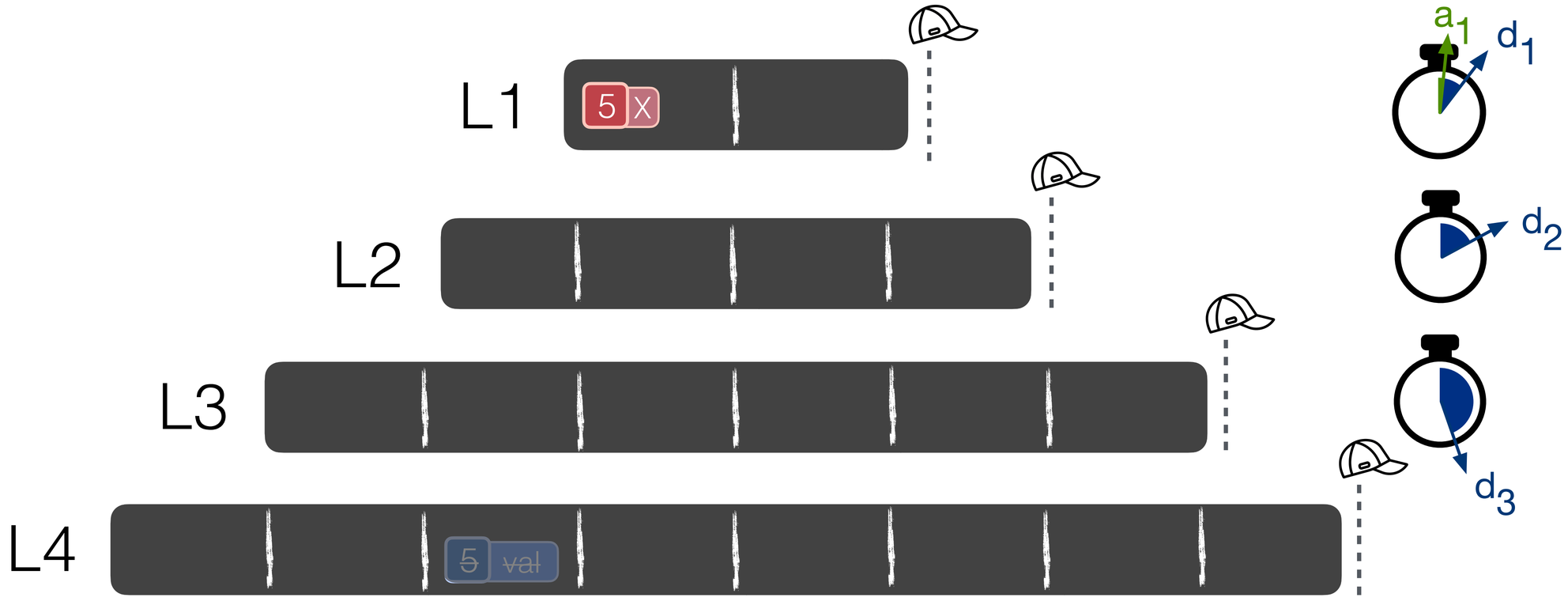
delete(5) within threshold: D_{th}

FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

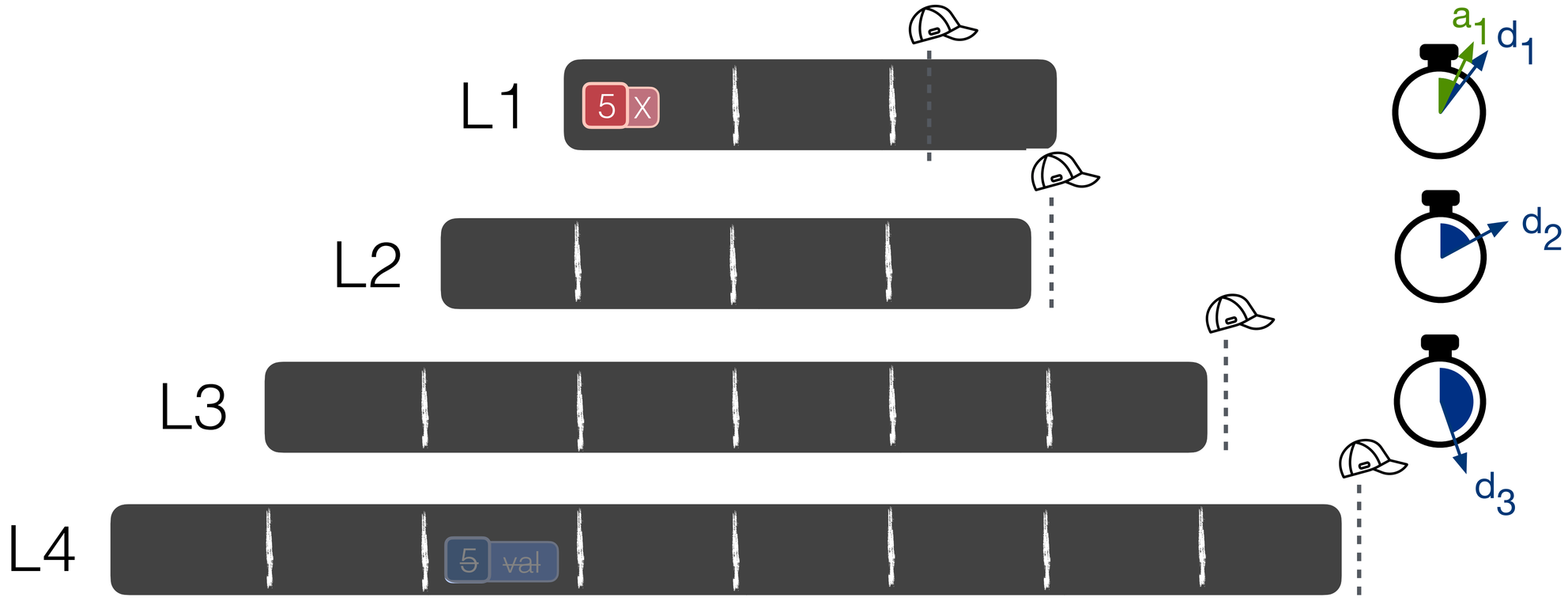


FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

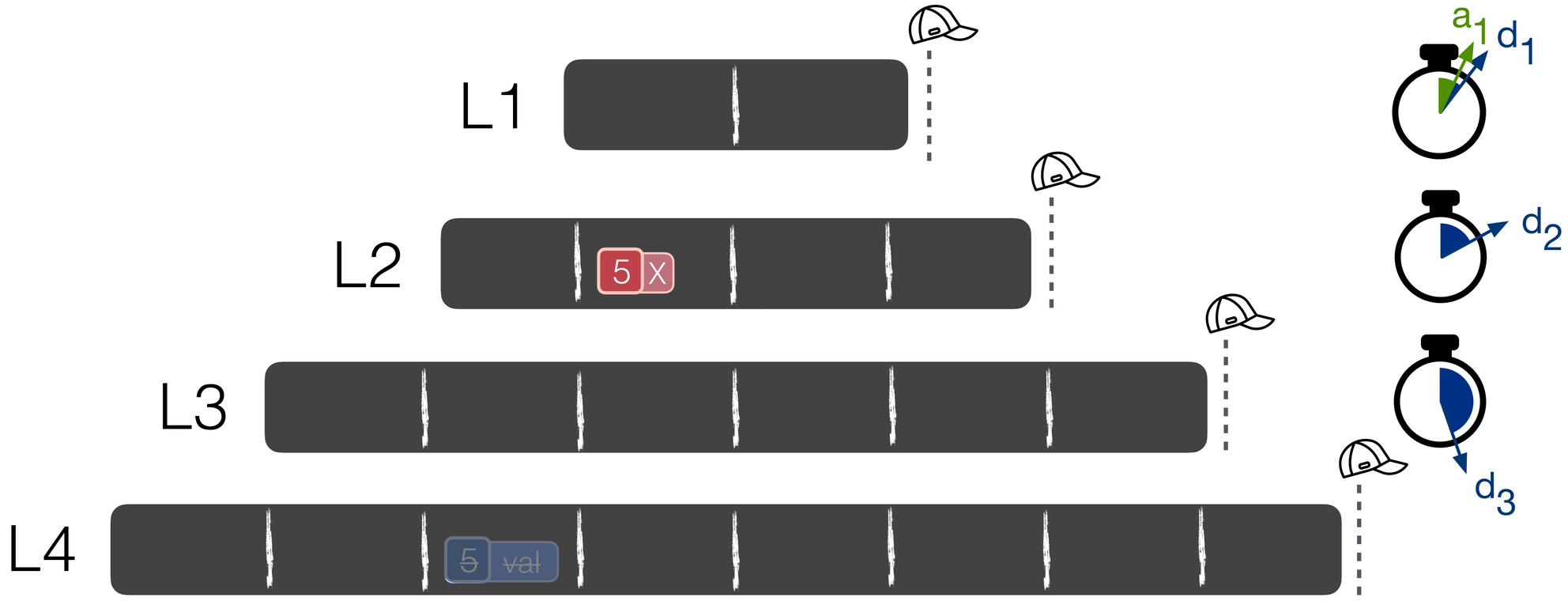


FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

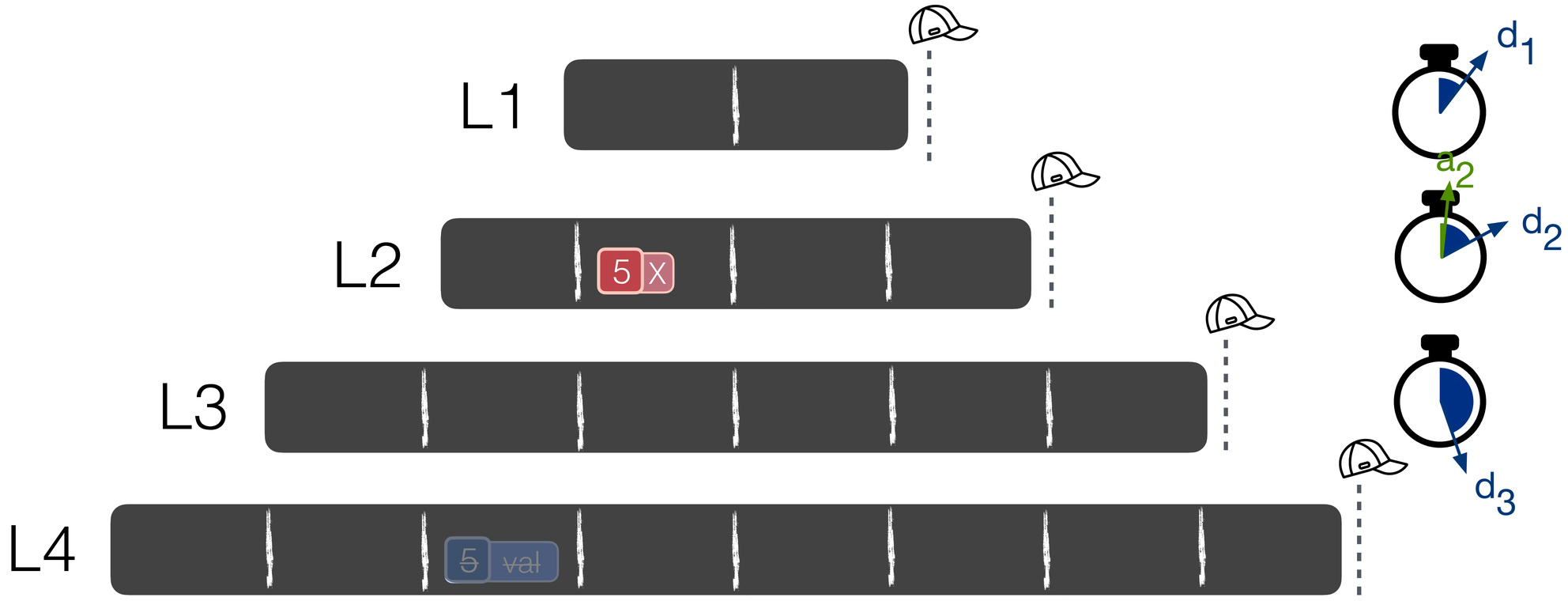


FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

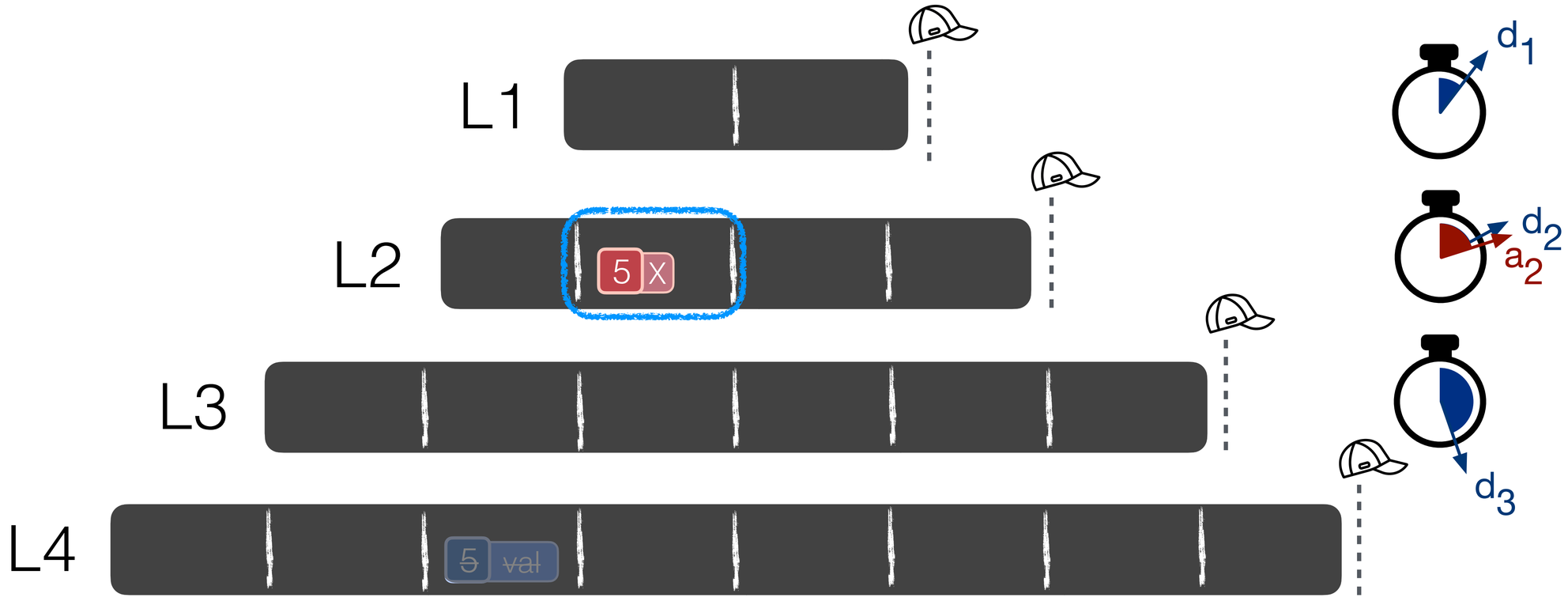


FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

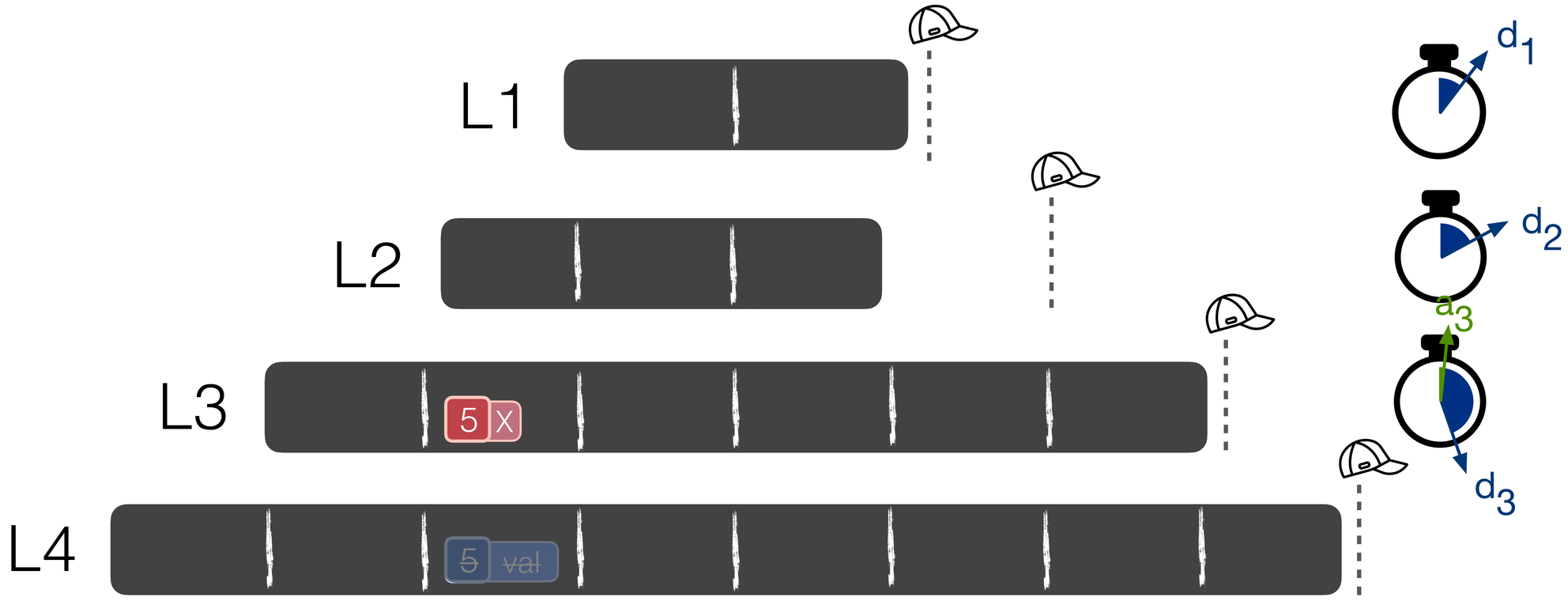


FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

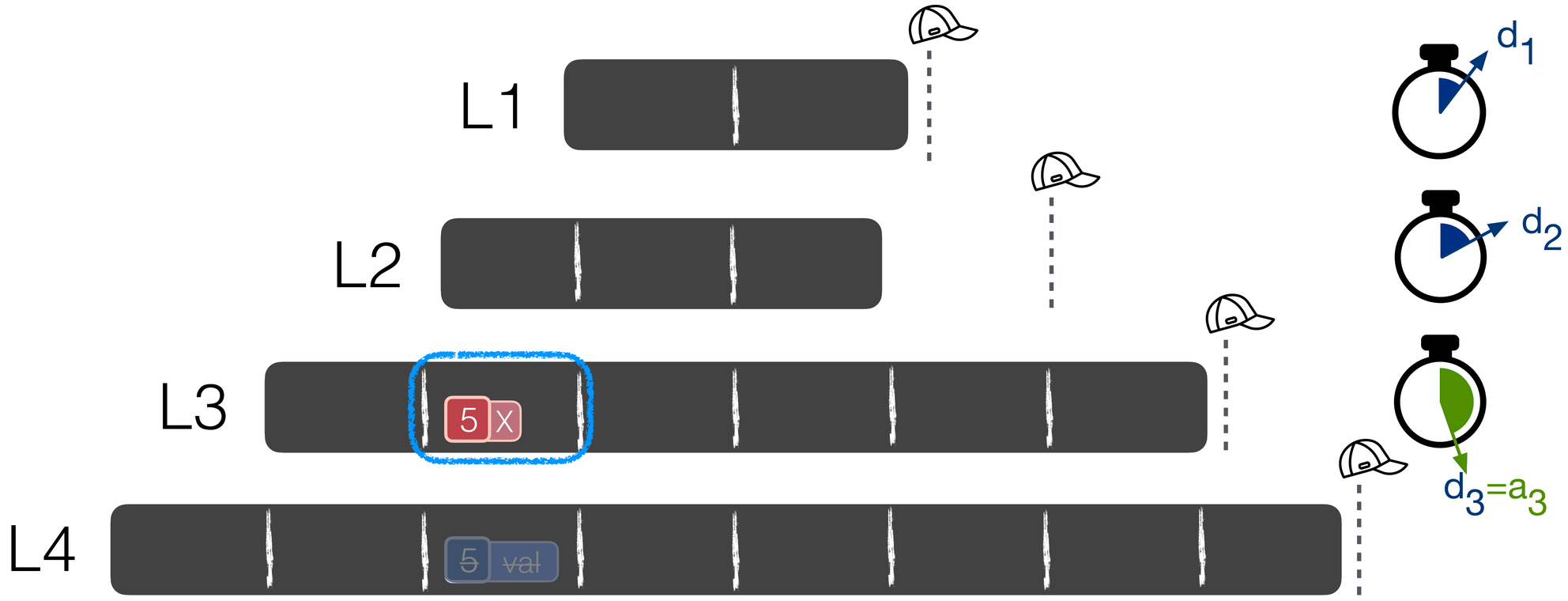


FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

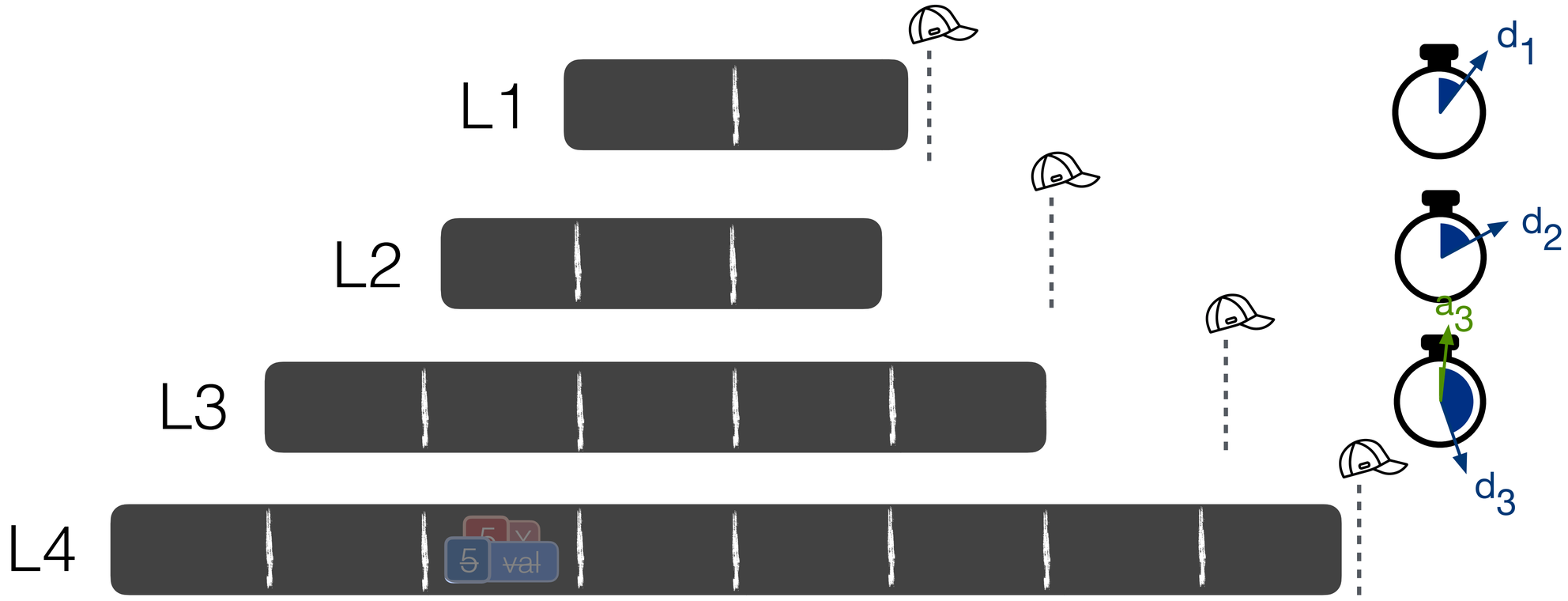


FAst **DE**lete

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

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

delete(5) within threshold: D_{th}

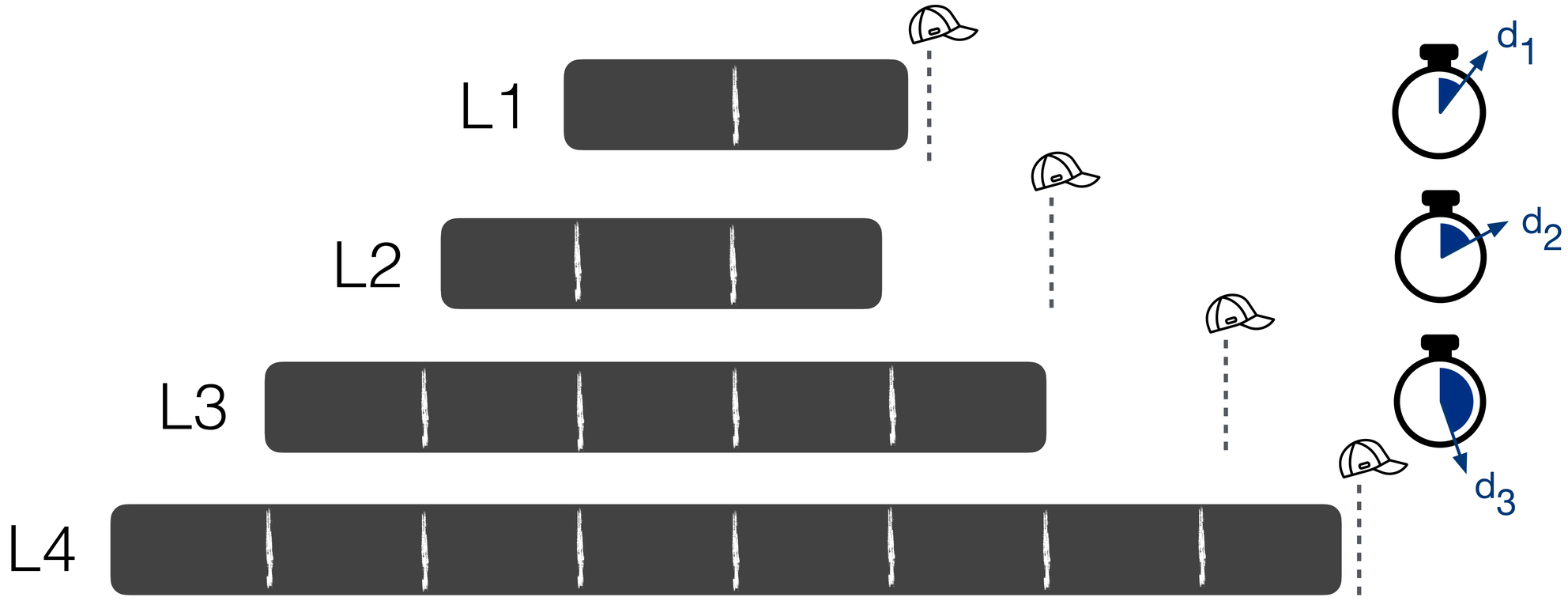


FAst **DE**lete

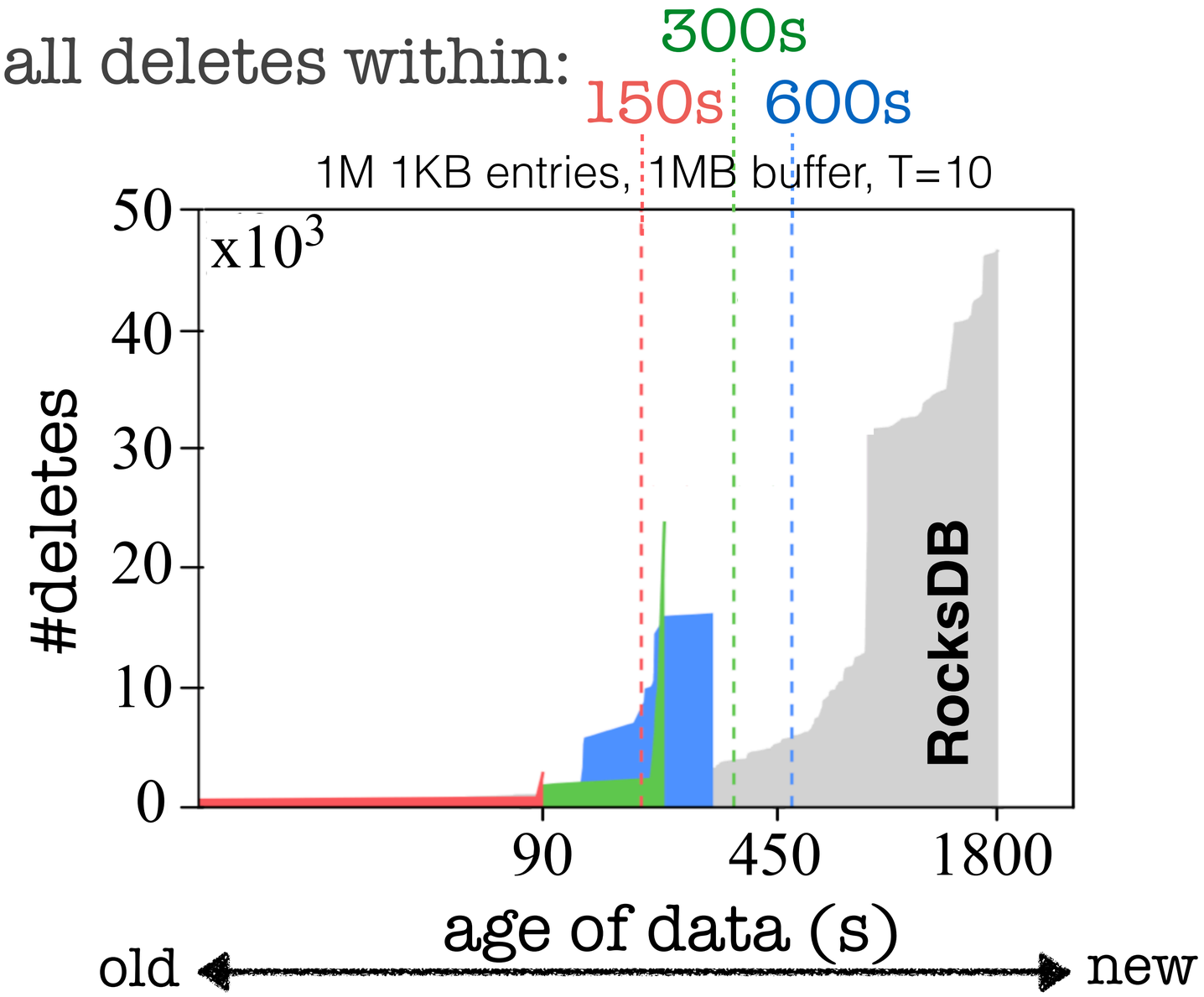
$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

delete(5) within threshold: D_{th}



persist all deletes within:



persists deletes timely

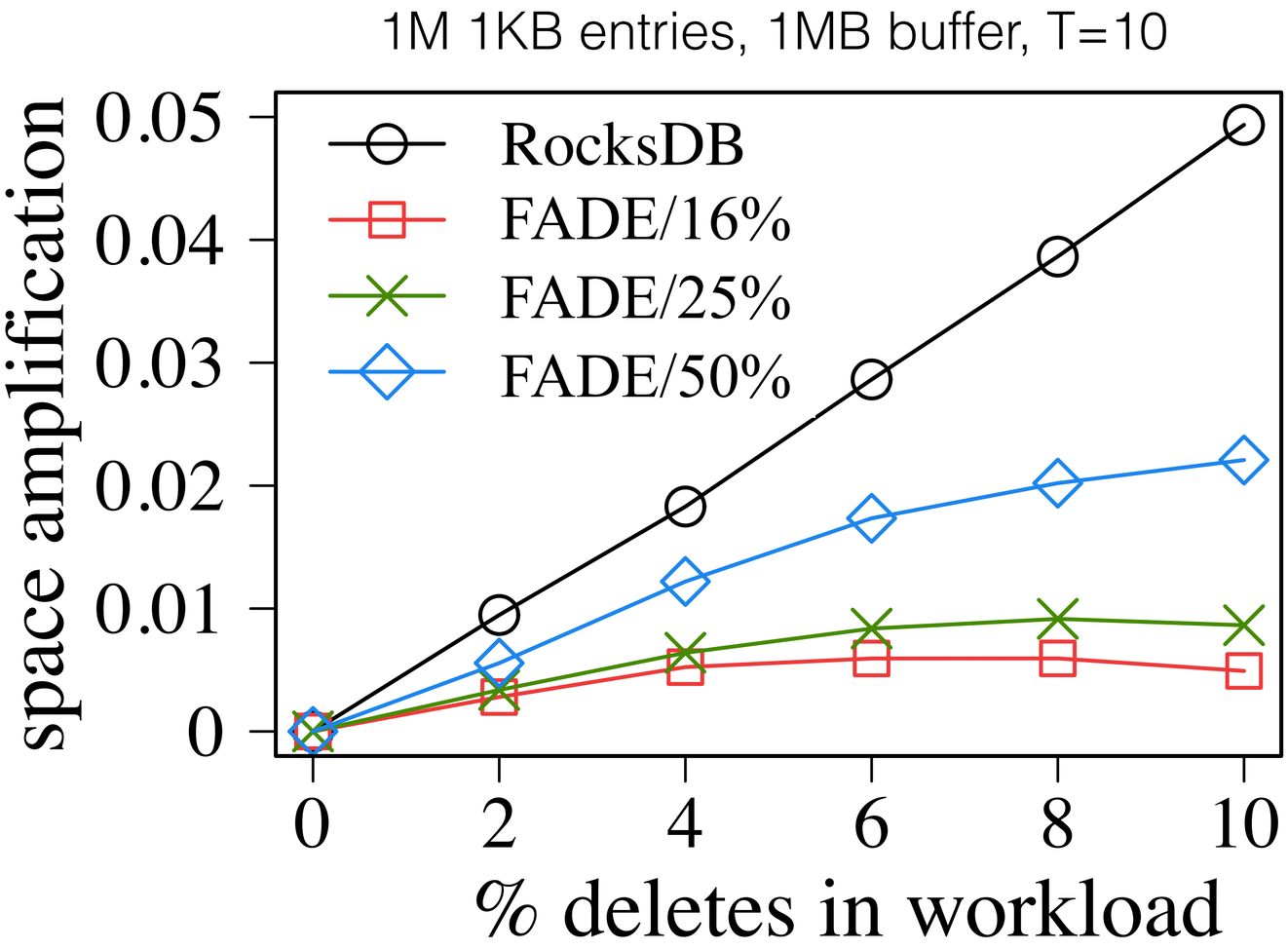
within threshold

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



improved read performance

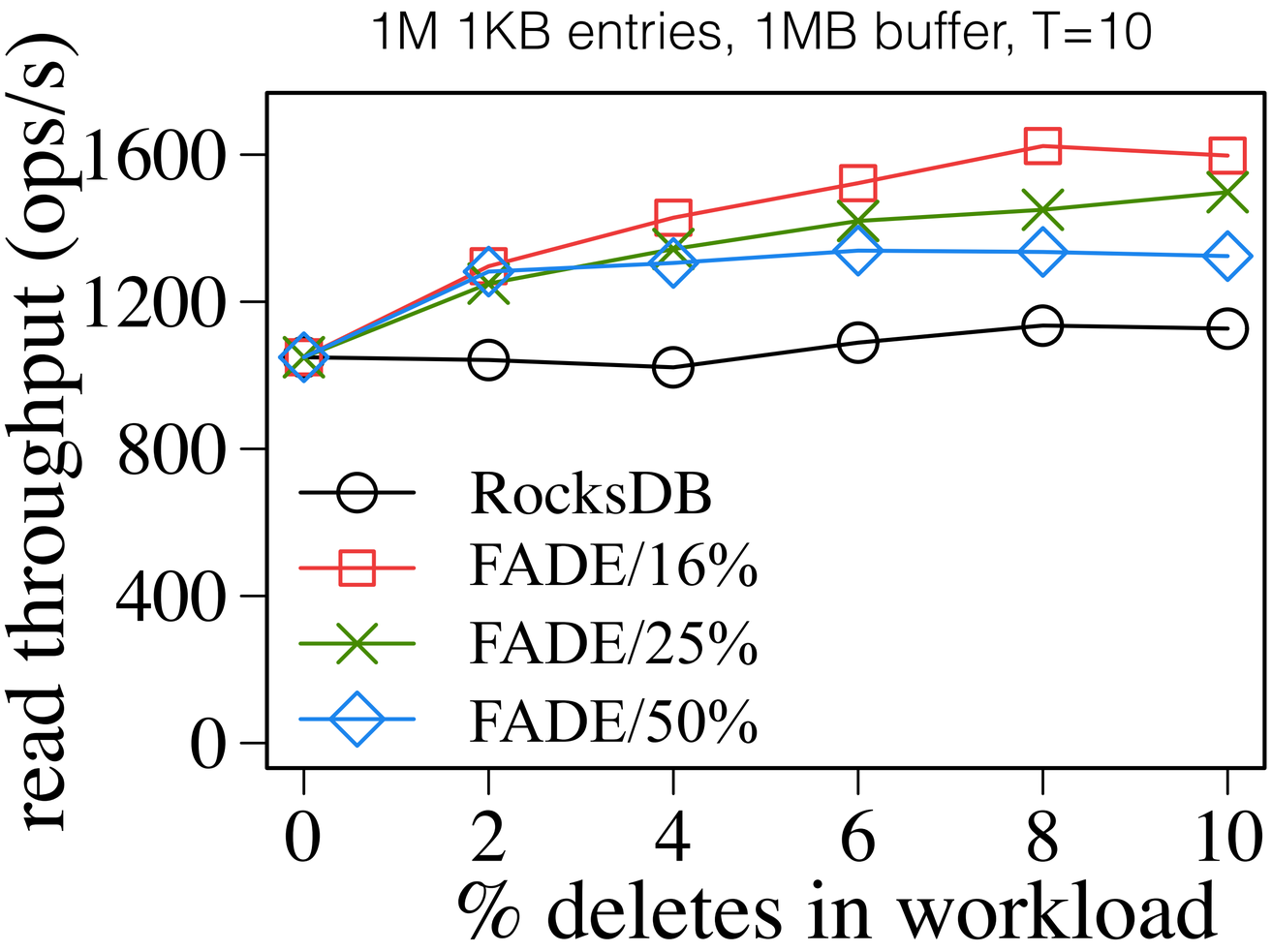
1.2x - 1.4x

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



higher write amplification

4% - 25%

improved read performance

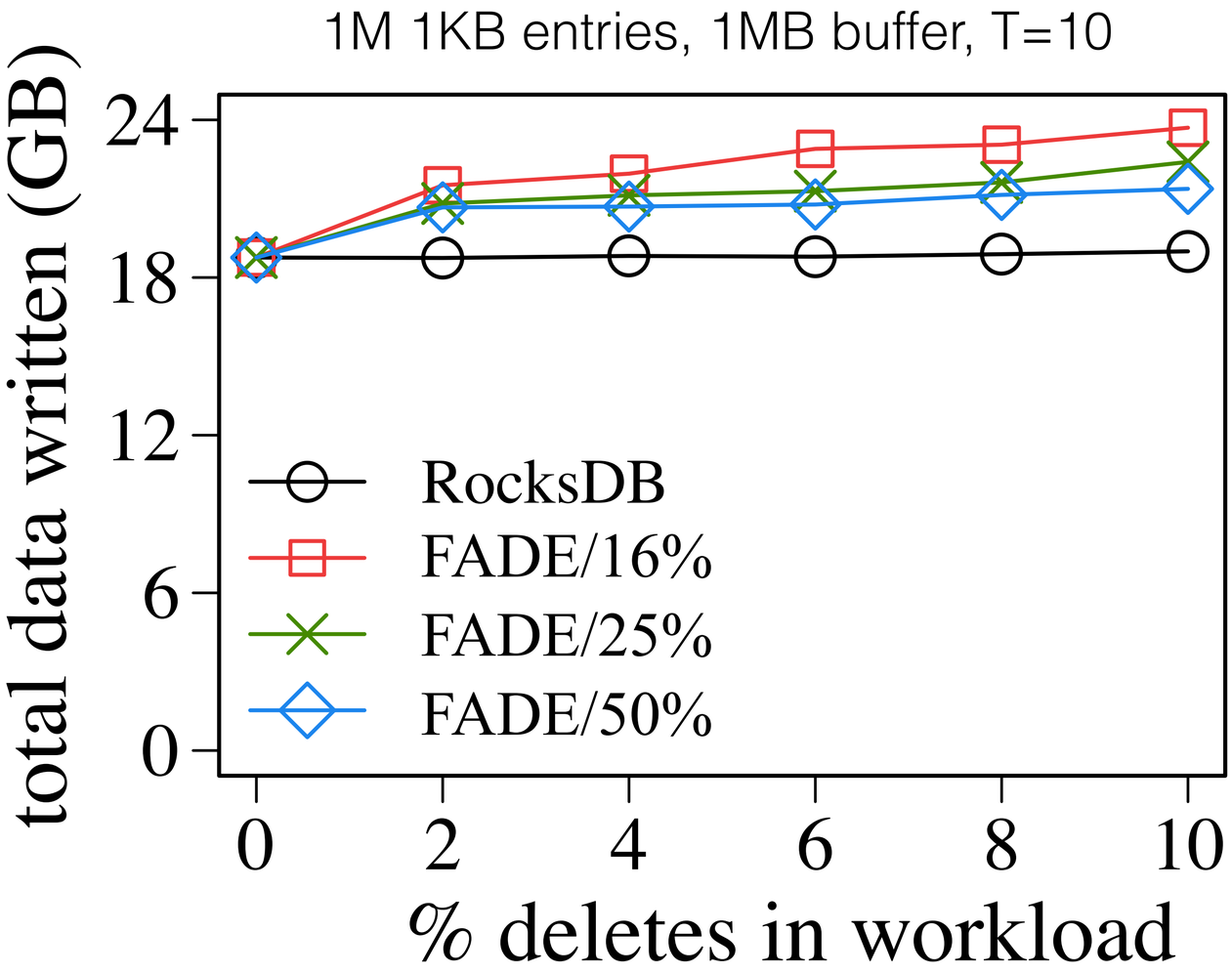
1.2x - 1.4x

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



higher write amplification

4% - 25%

improved read performance

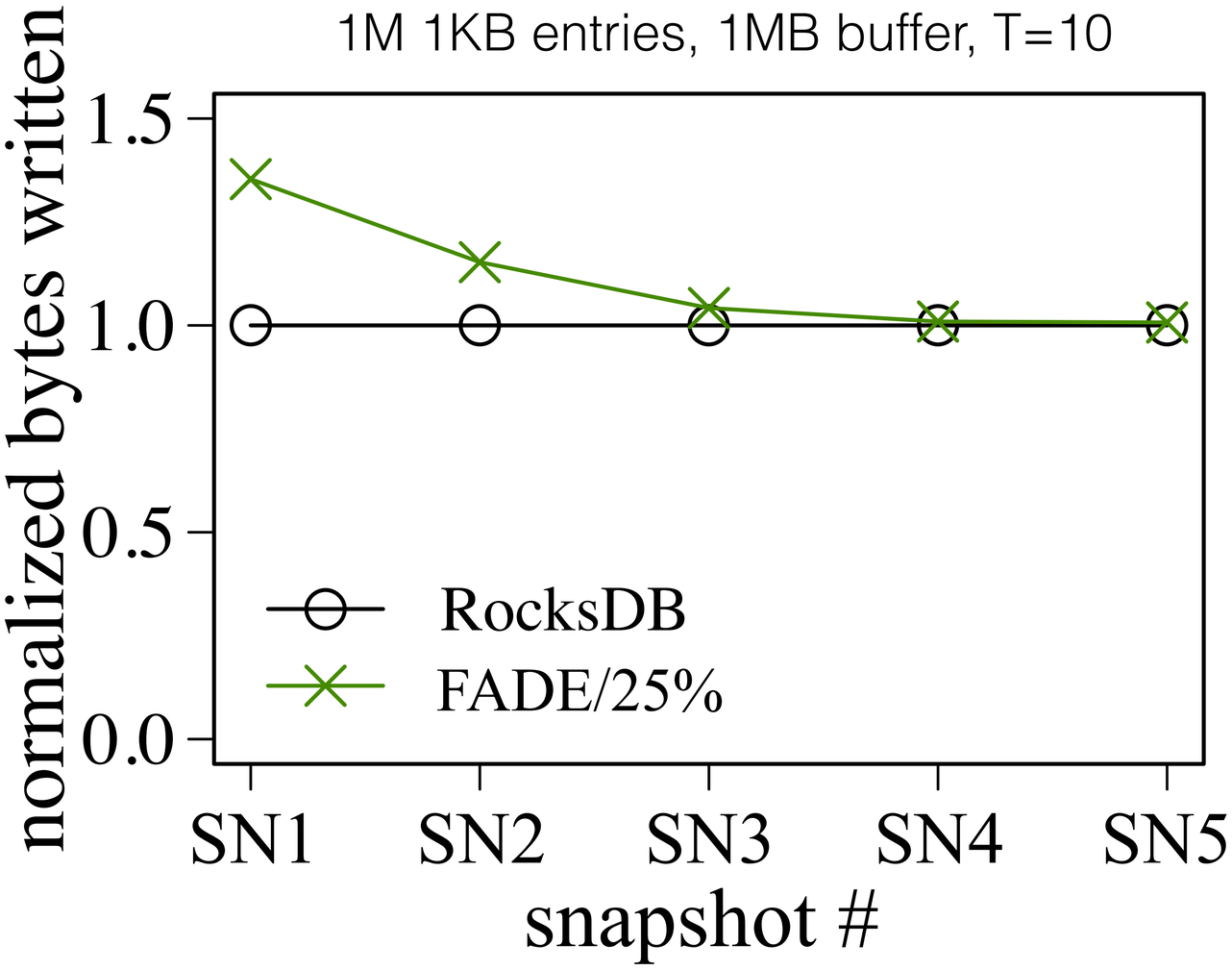
1.2x - 1.4x

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



higher write amplification

0.7%

improved read performance

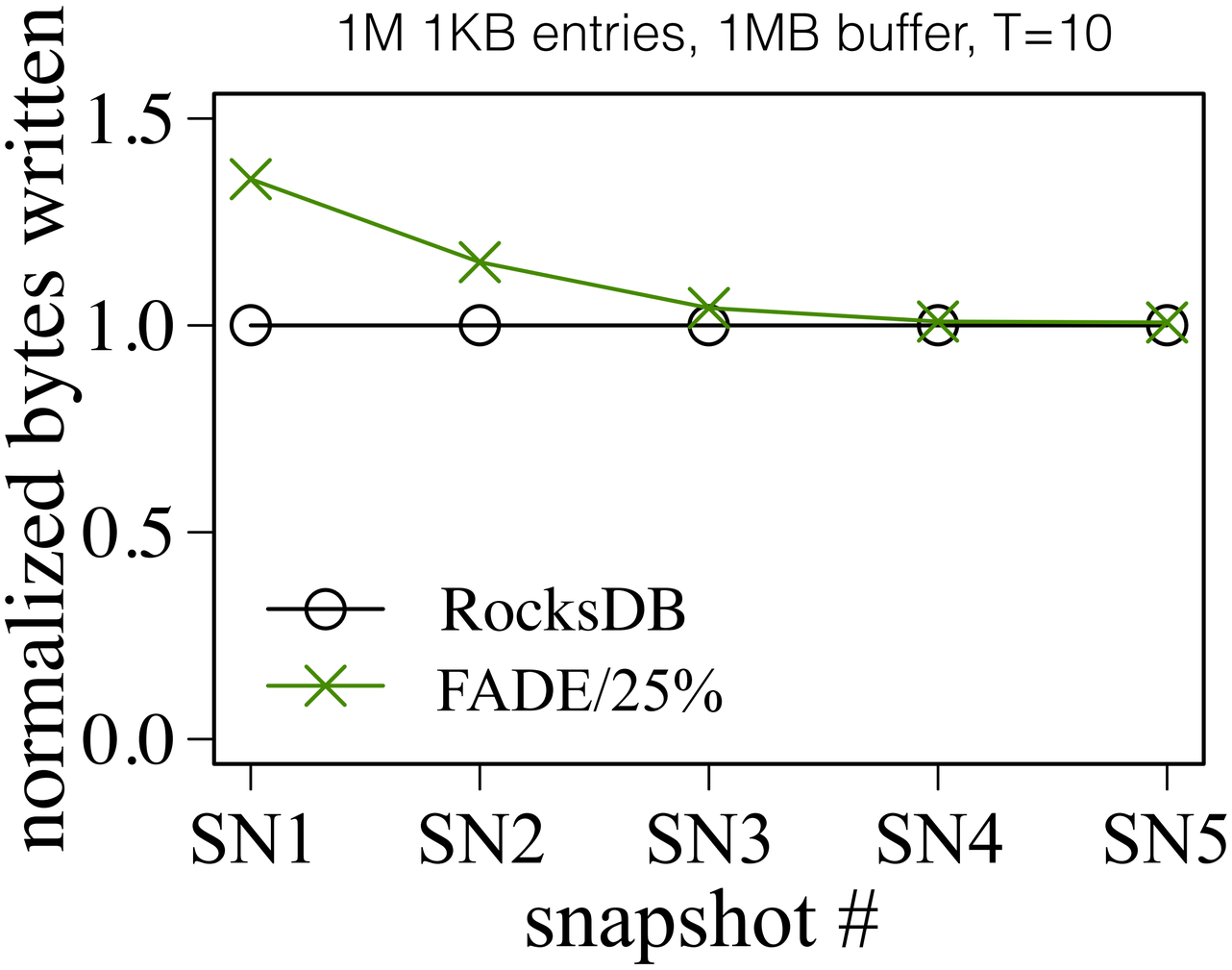
1.2x - 1.4x

reduced space amplification

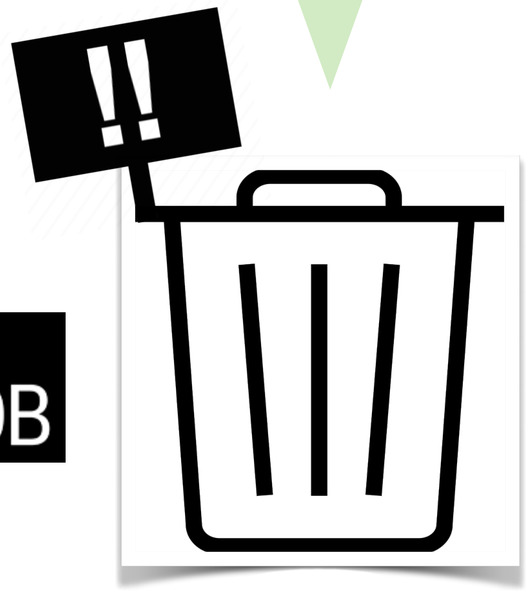
2.1x - 9.8x

persists deletes timely

within threshold



FADE

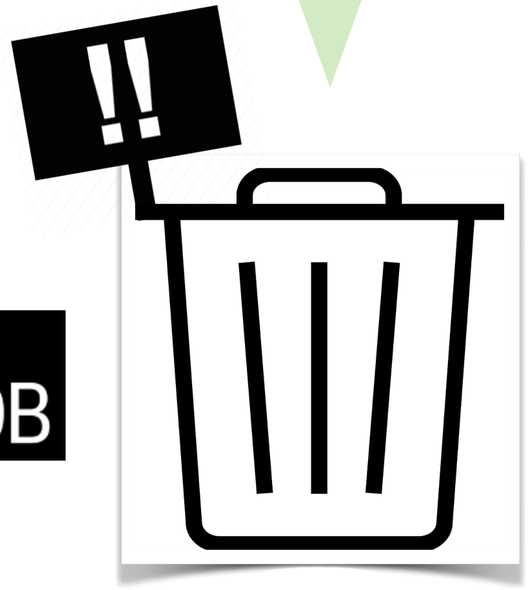


RocksDB

on-demand

persist all logical
deletes within D days

FADE



RocksDB

on-demand

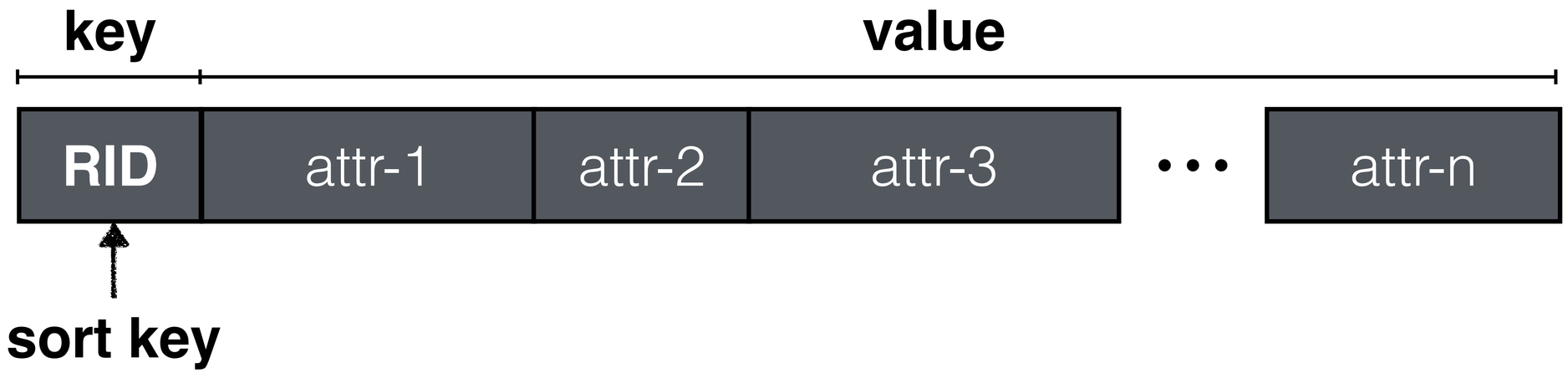
persist all logical deletes within D days

retention-based

delete all data older than T days

Realizing **Retention-Based Deletes**

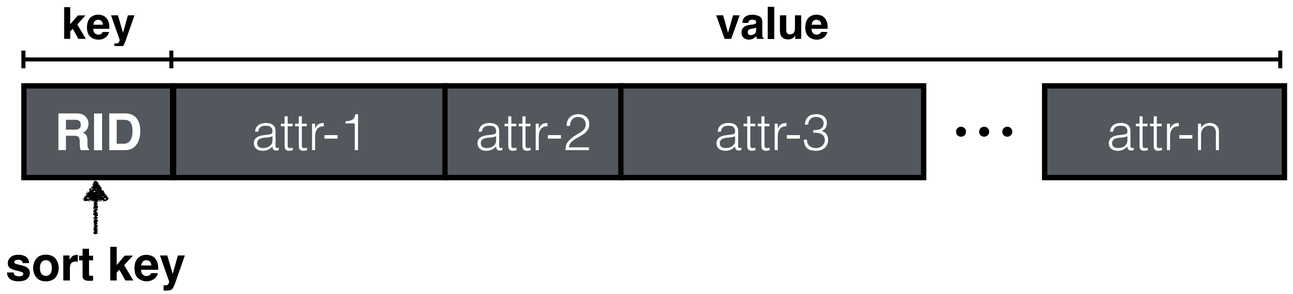
delete all entries older than: TS_x



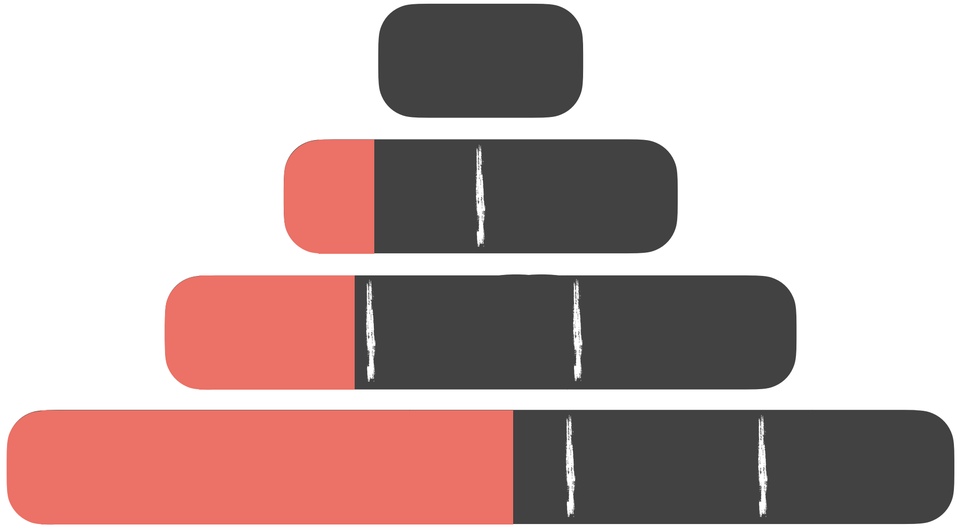
sort key = delete key

Realizing **Retention-Based Deletes**

delete all entries older than: TS_x

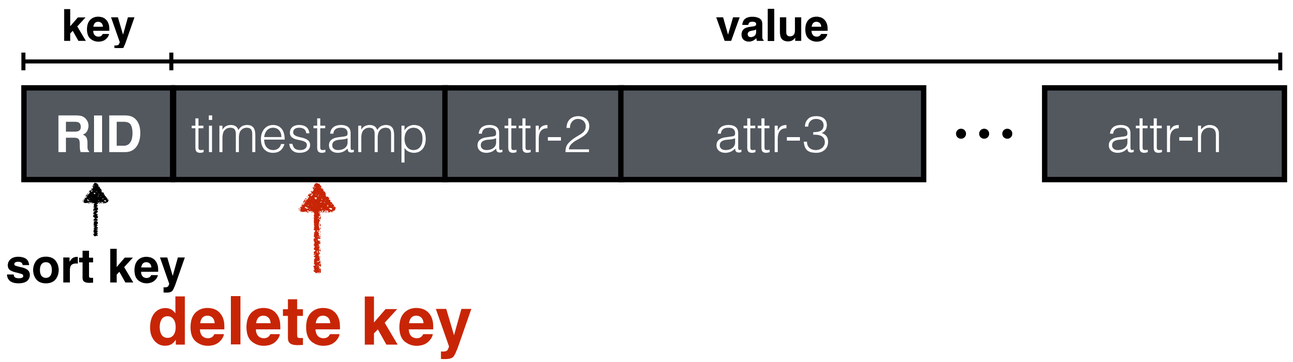


sort key = delete key

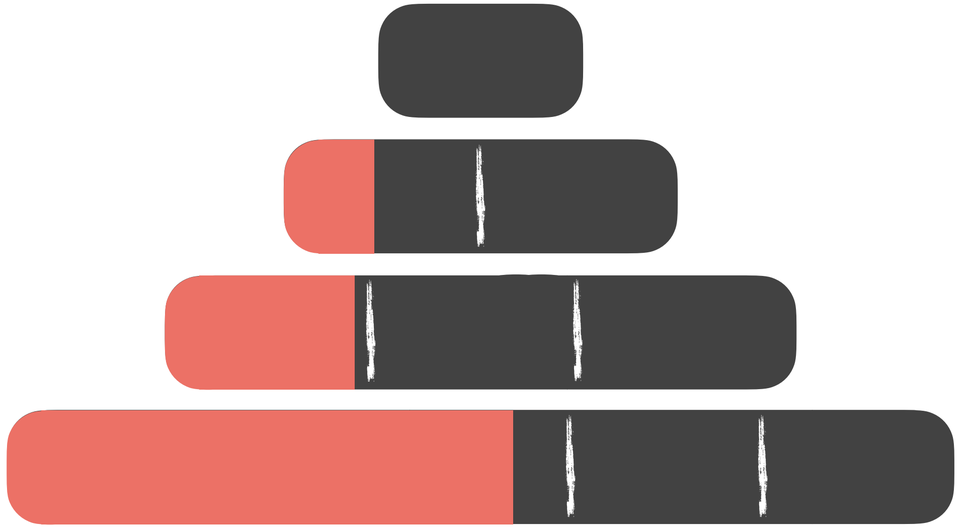


Realizing **Retention-Based Deletes**

delete all entries older than: TS_x

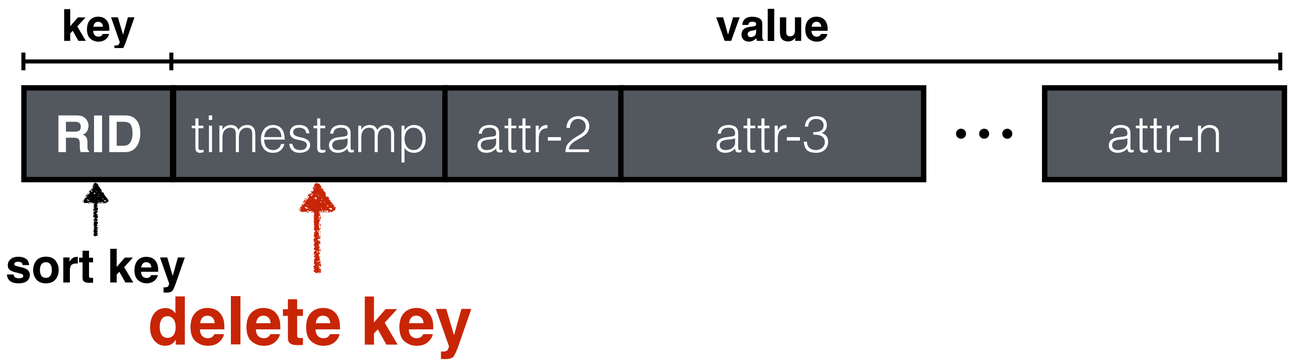


sort key ≠ delete key

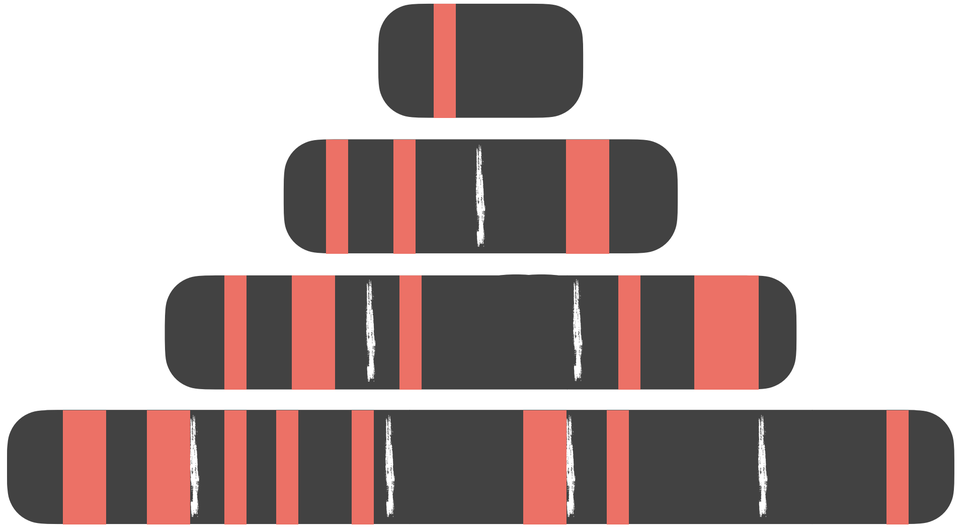


Realizing **Retention-Based Deletes**

delete all entries older than: TS_x



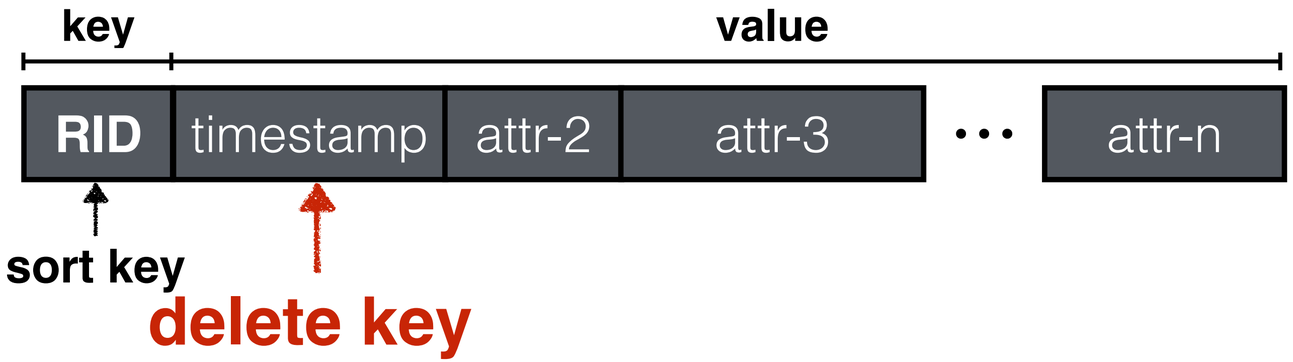
sort key ≠ delete key



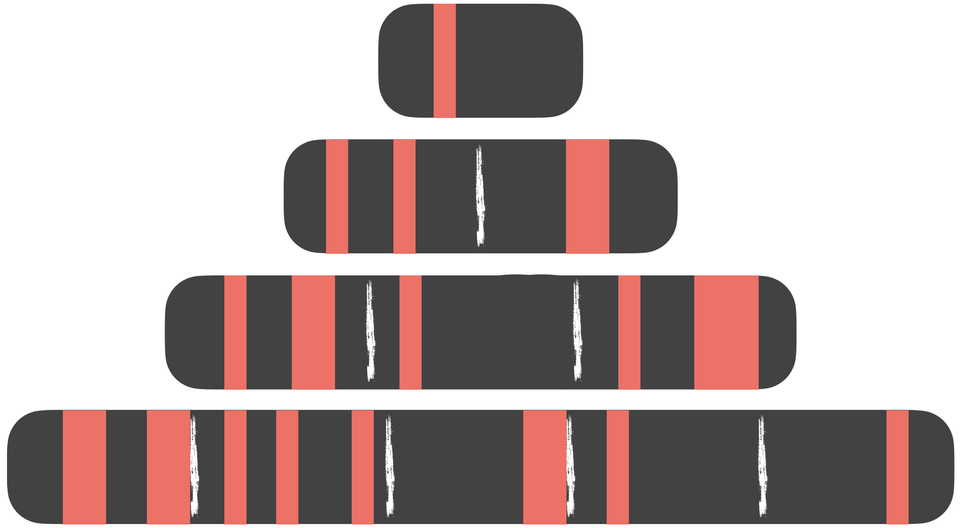
scattered occurrences

Realizing **Retention-Based Deletes**

delete all entries older than: TS_x



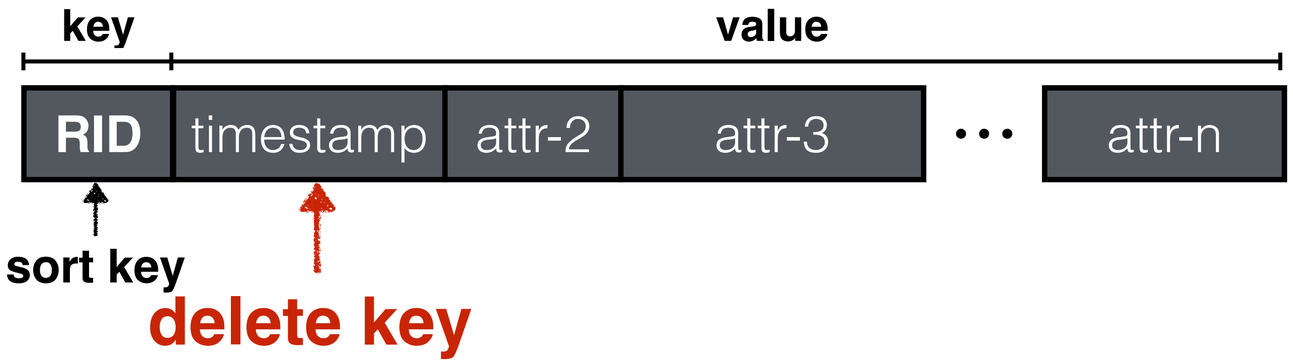
sort key ≠ delete key



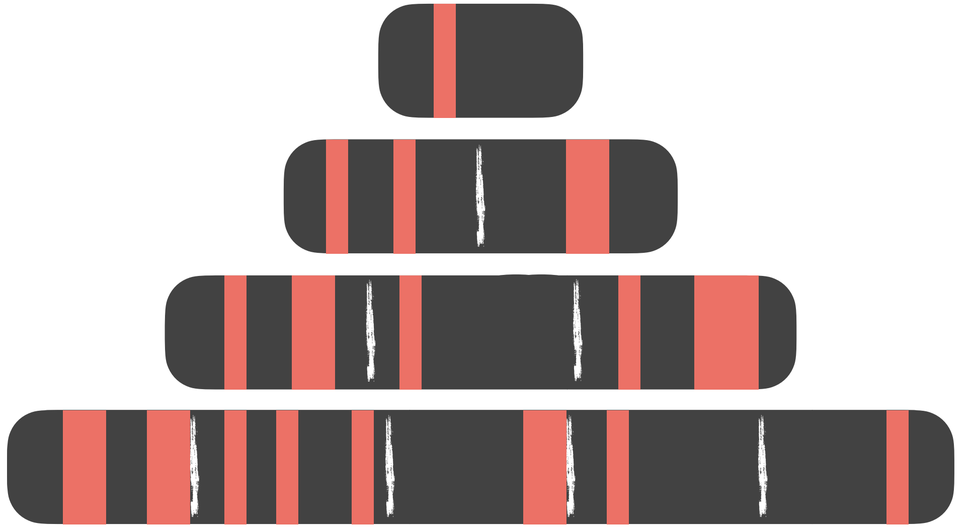
scattered occurrences

Realizing **Retention-Based Deletes**

delete all entries older than: TS_x



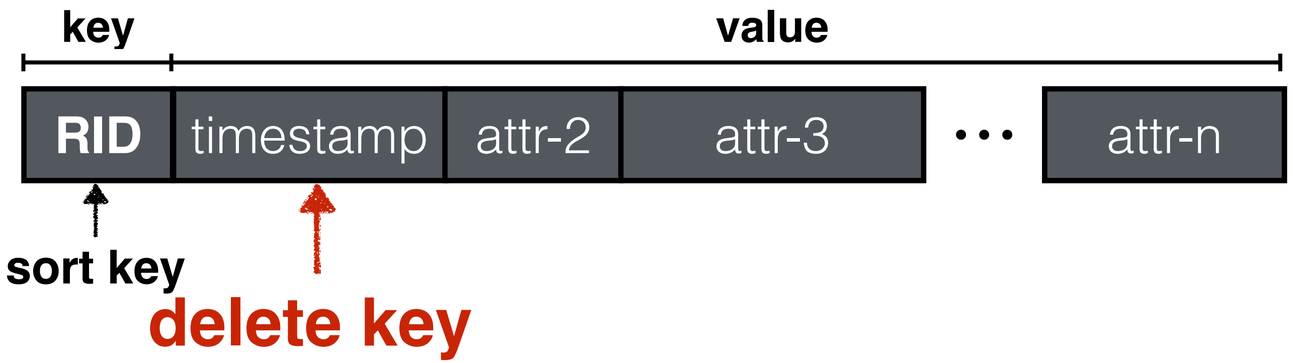
sort key ≠ delete key



scattered occurrences

Realizing **Retention-Based Deletes**

delete all entries older than: TS_x



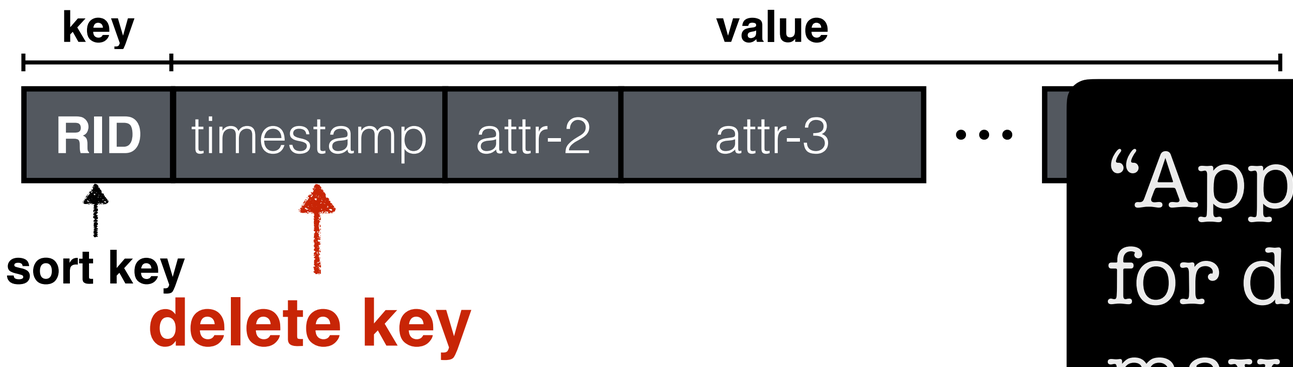
sort key ≠ delete key

latency spikes

superfluous I/Os

Realizing **Retention-Based Deletes**

delete all entries older than: TS_x



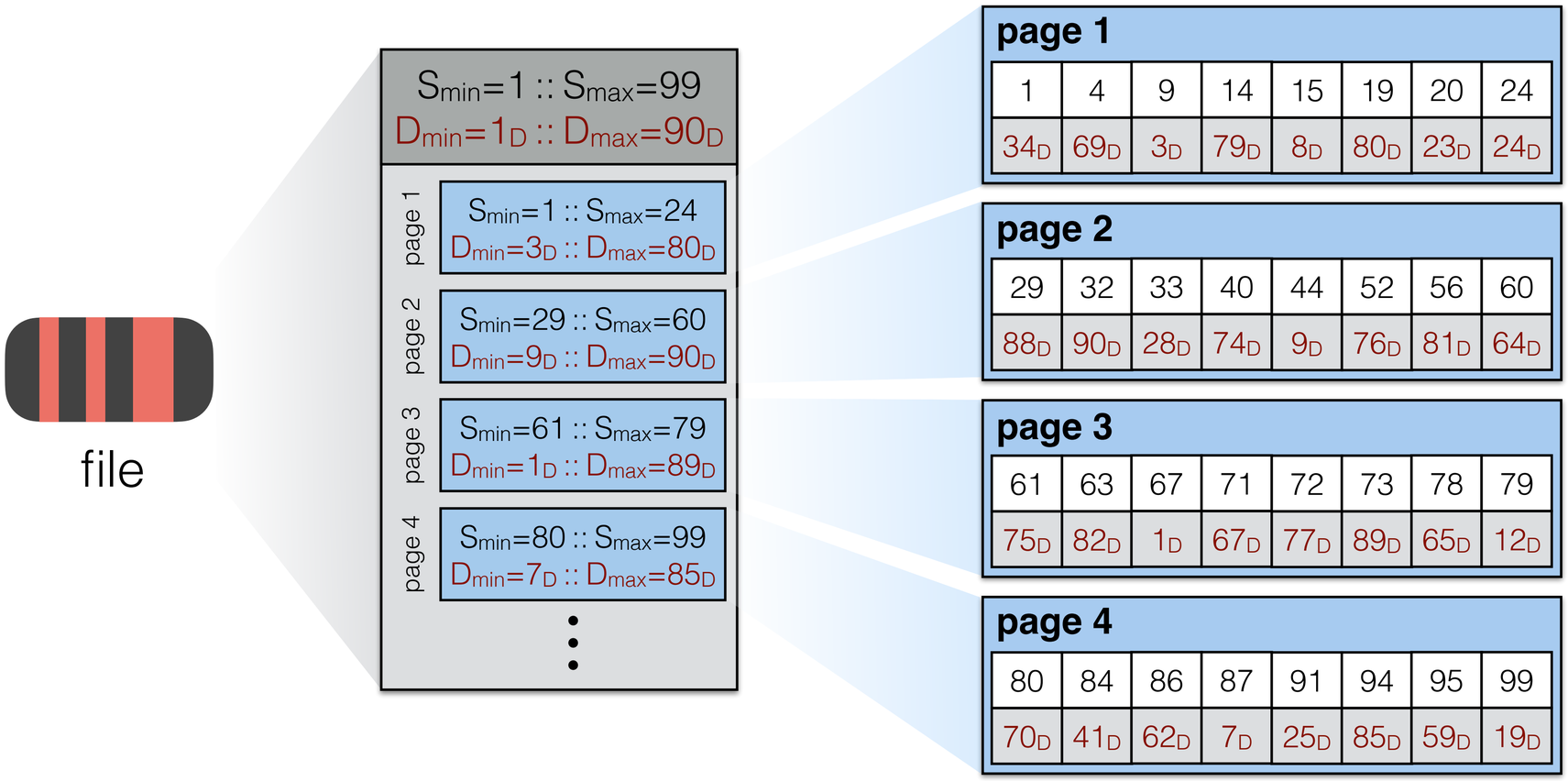
sort key ≠ delete ke

"Applications have requirements for deletes every day. E.g., they may keep data for 30 days, … effectively purging 1/30 of the database every day.

This induces performance pains!"

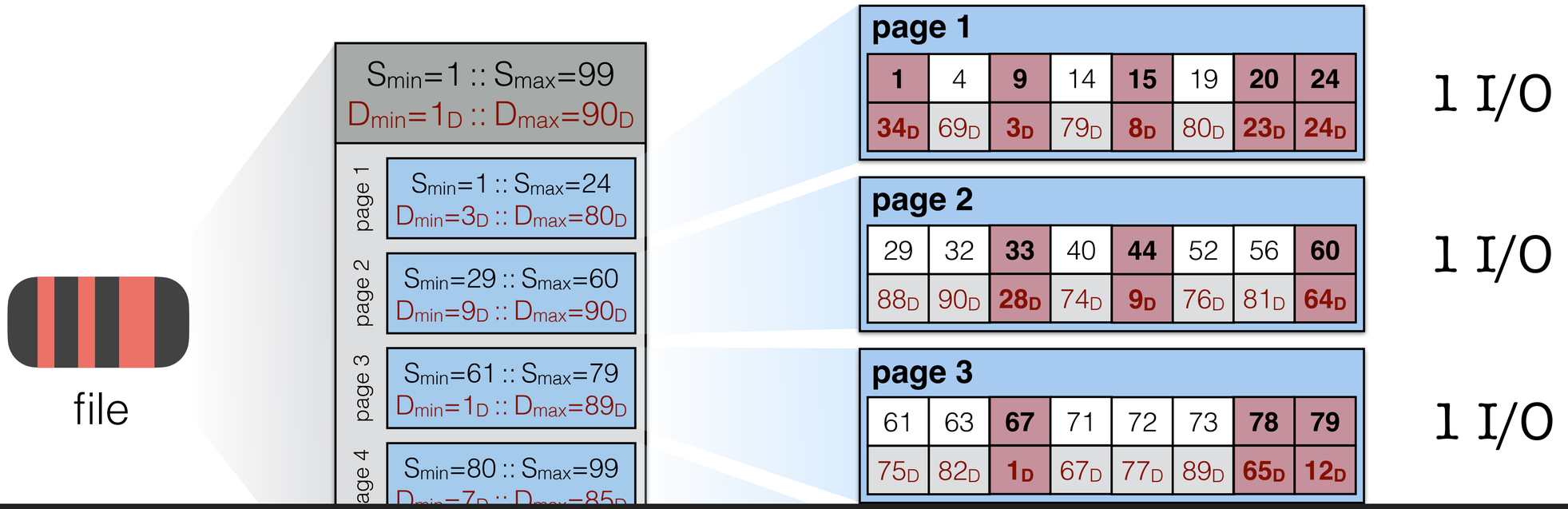
Realizing **Retention-Based Deletes**

delete all entries older than <= 65_D



Realizing **Retention-Based Deletes**

delete all entries older than <= 65_D



Intuition: Data Layout holds the key!

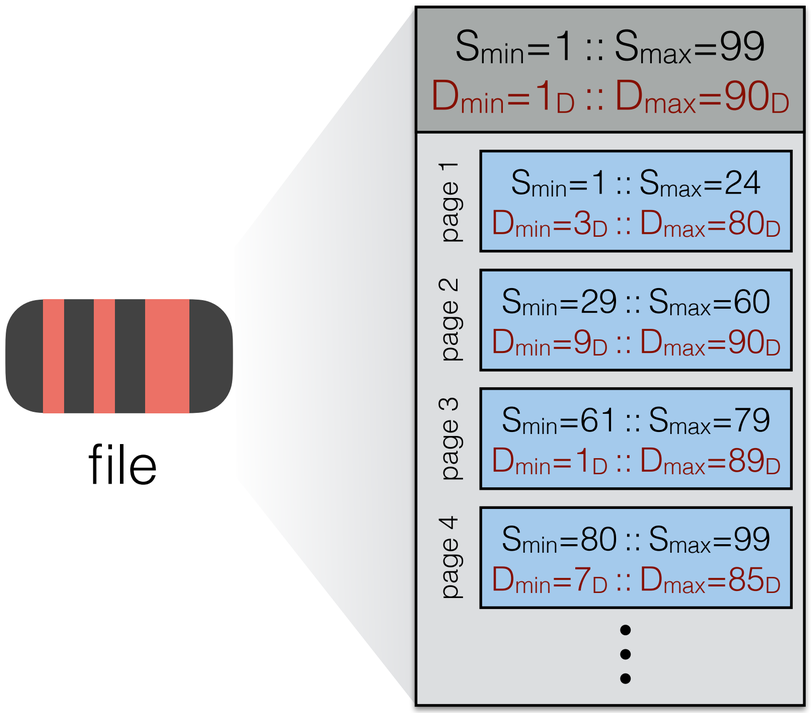
Realizing **Retention-Based Deletes**

KiWi

Key Weaving storage layout

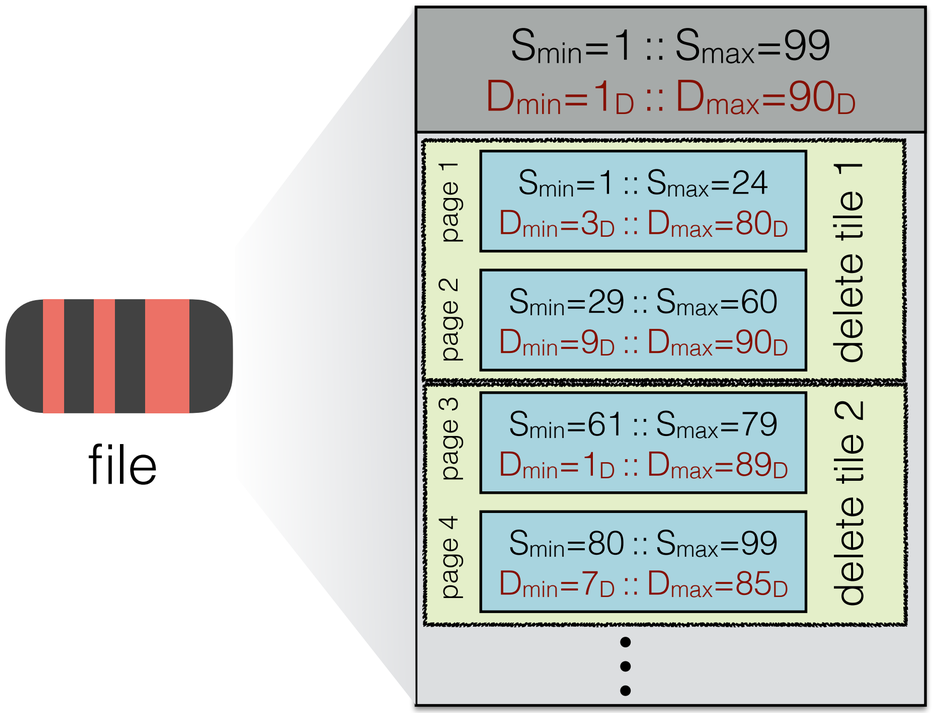
Key Weaving storage layout

delete all entries older than <= 65_D



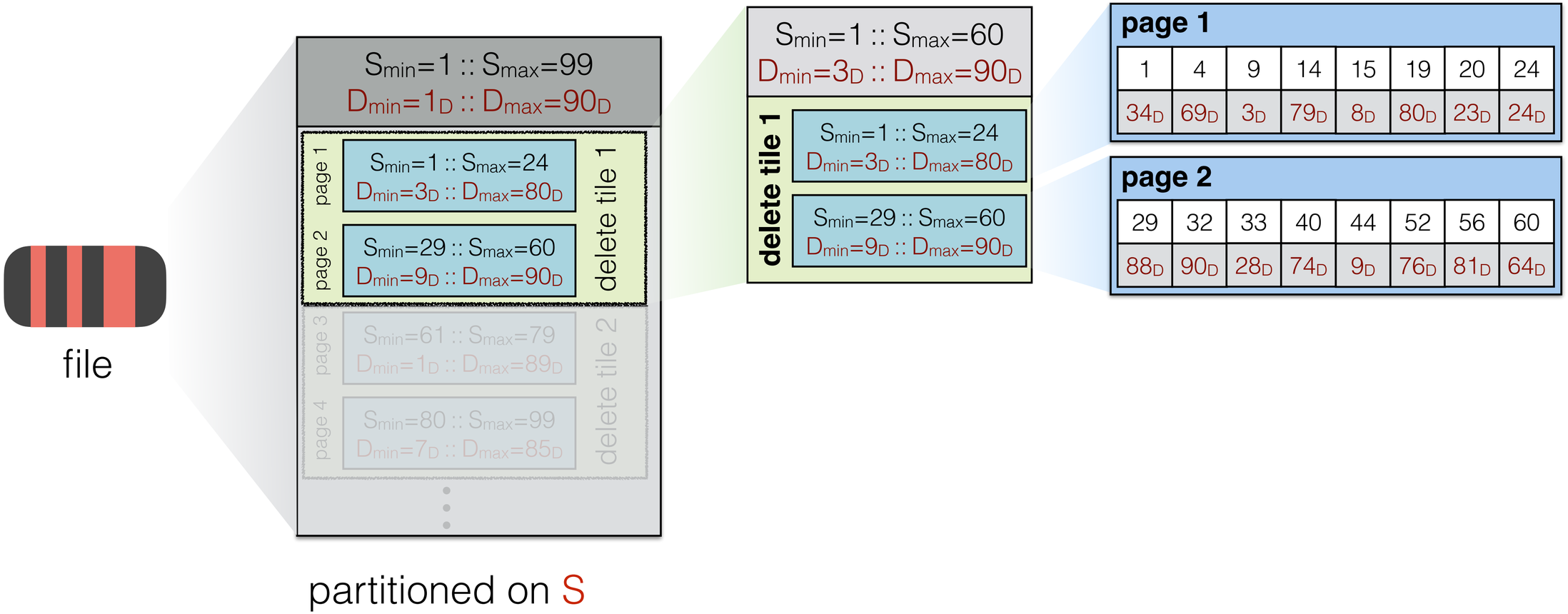
Key Weaving storage layout

delete all entries older than <= 65_D



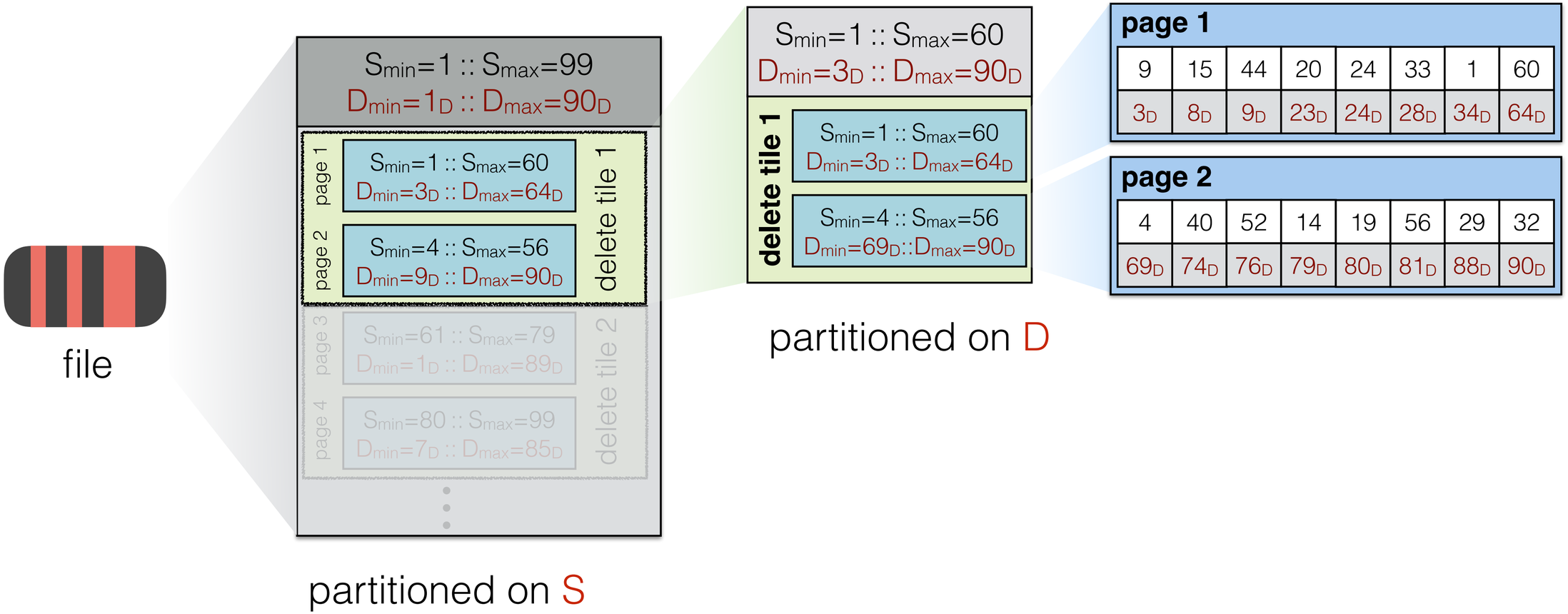
Key Weaving storage layout

delete all entries older than <= 65_D



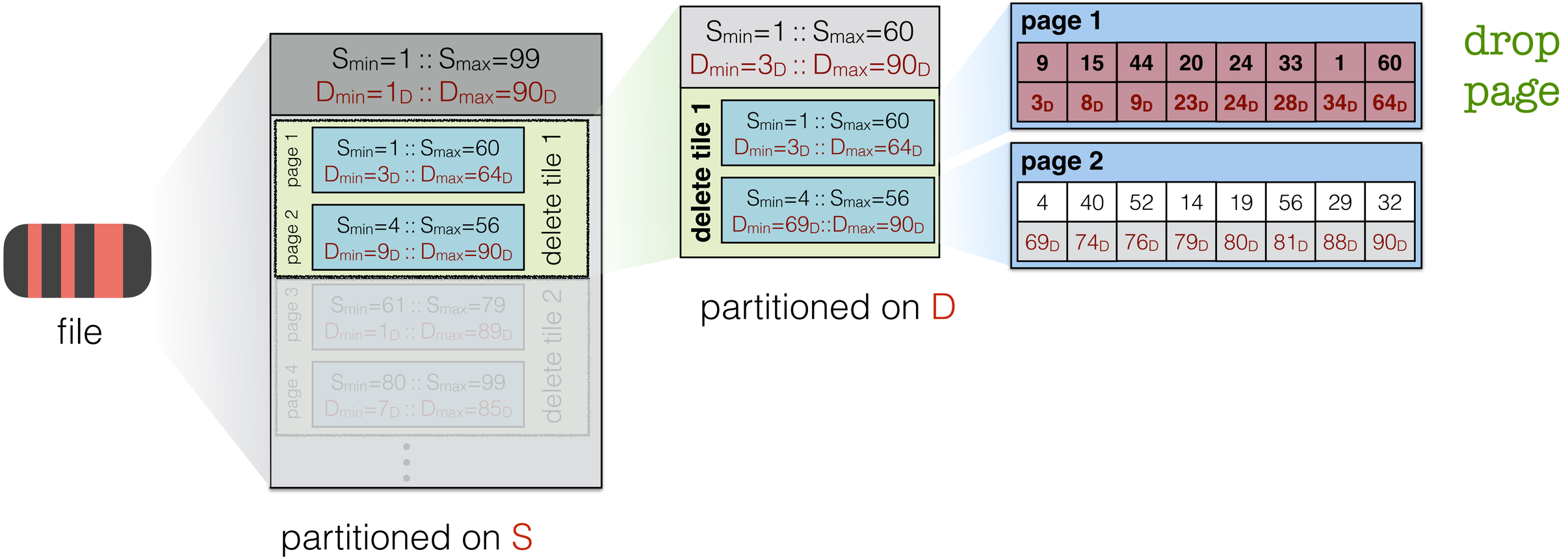
Key Weaving storage layout

delete all entries older than <= 65_D



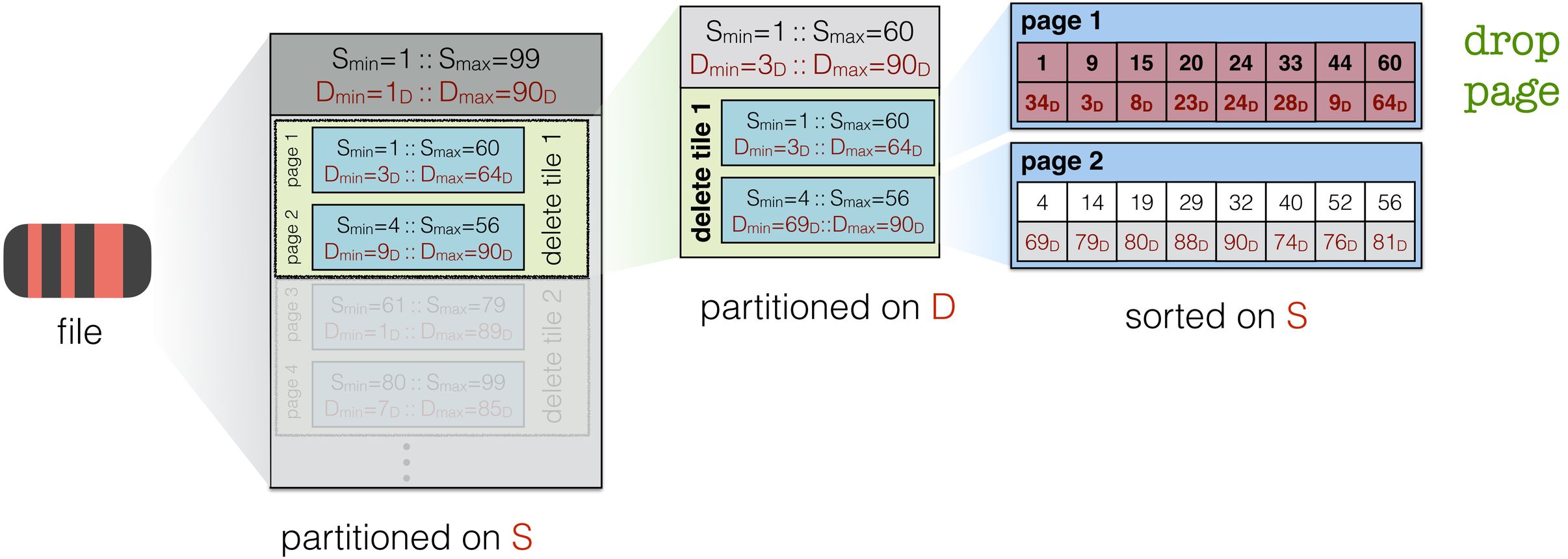
Key Weaving storage layout

delete all entries older than <= 65_D



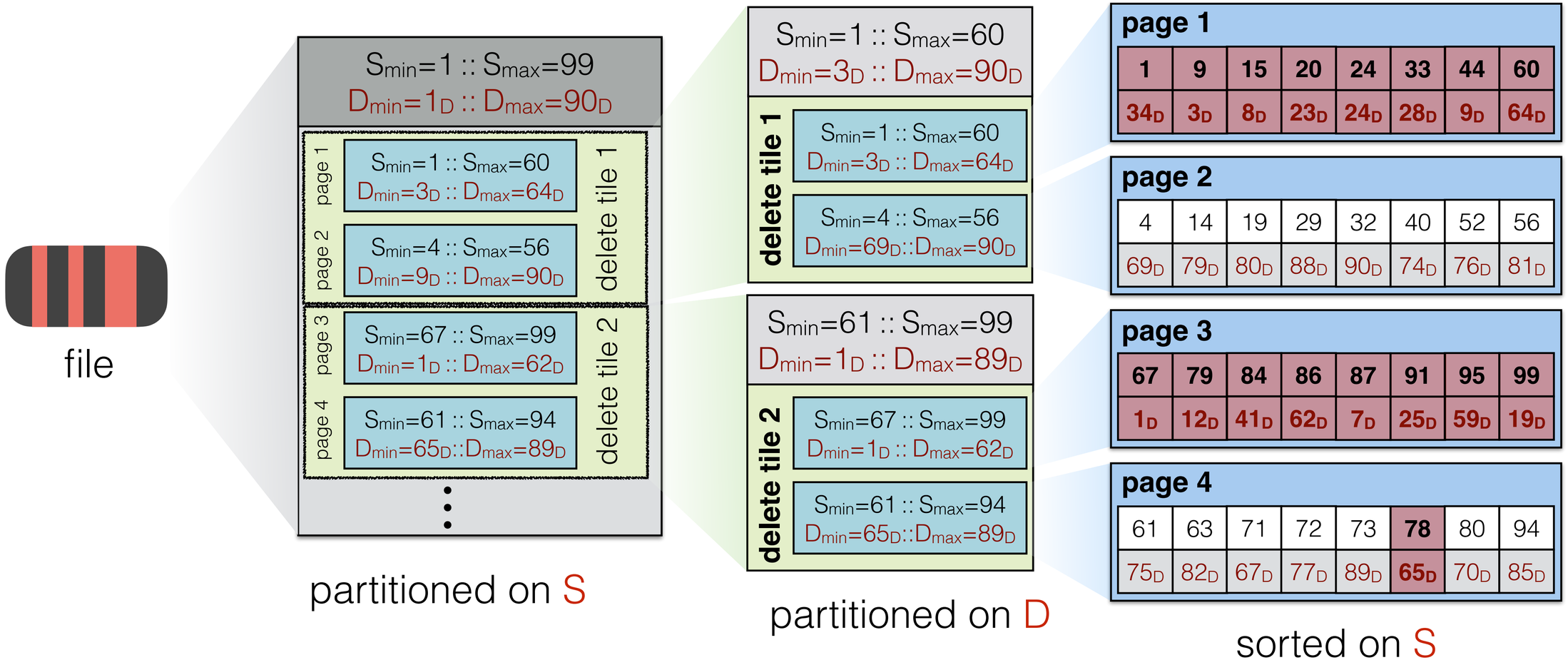
Key Weaving storage layout

delete all entries older than <= 65_D



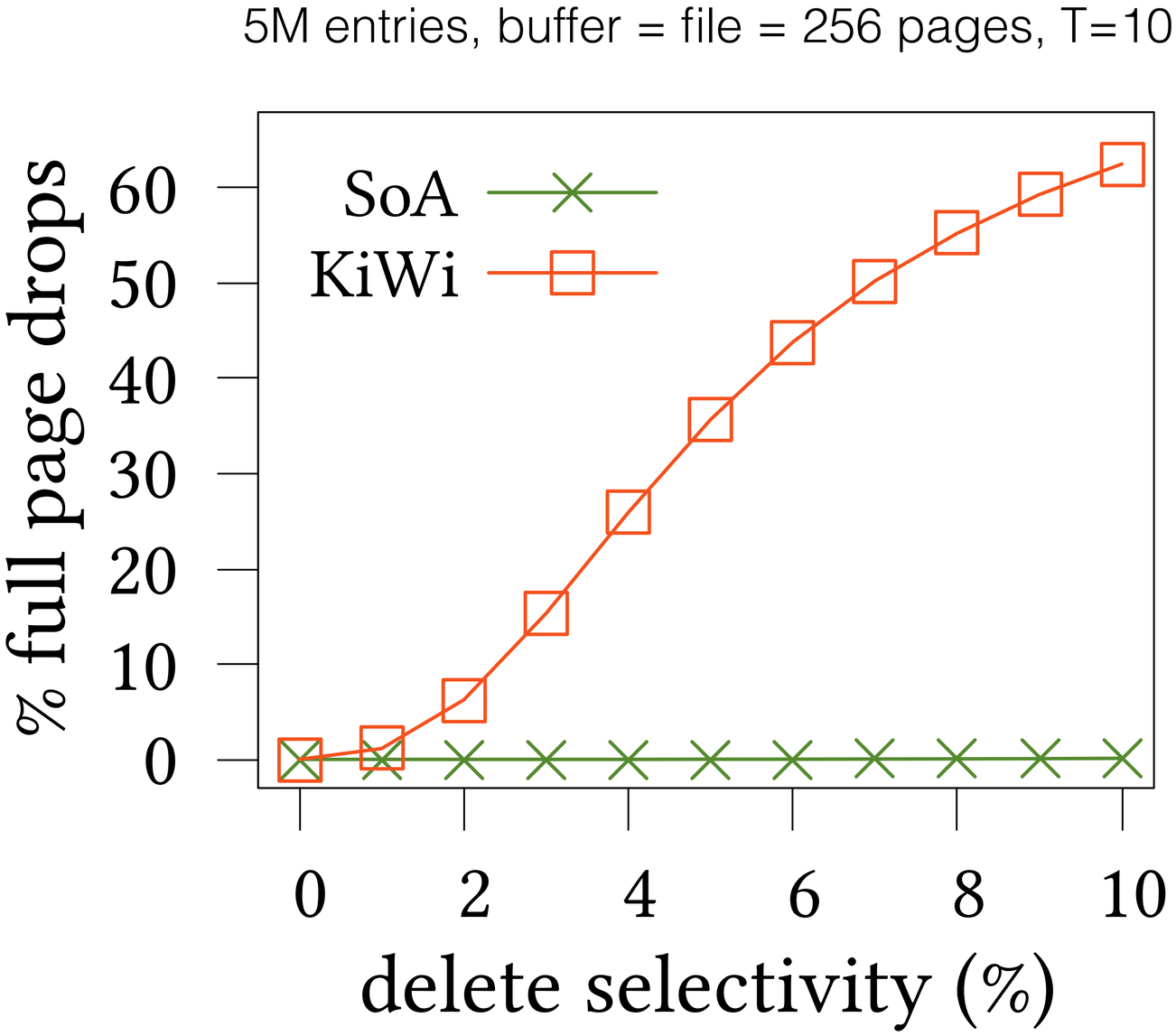
Key Weaving storage layout

delete all entries older than <= 65_D

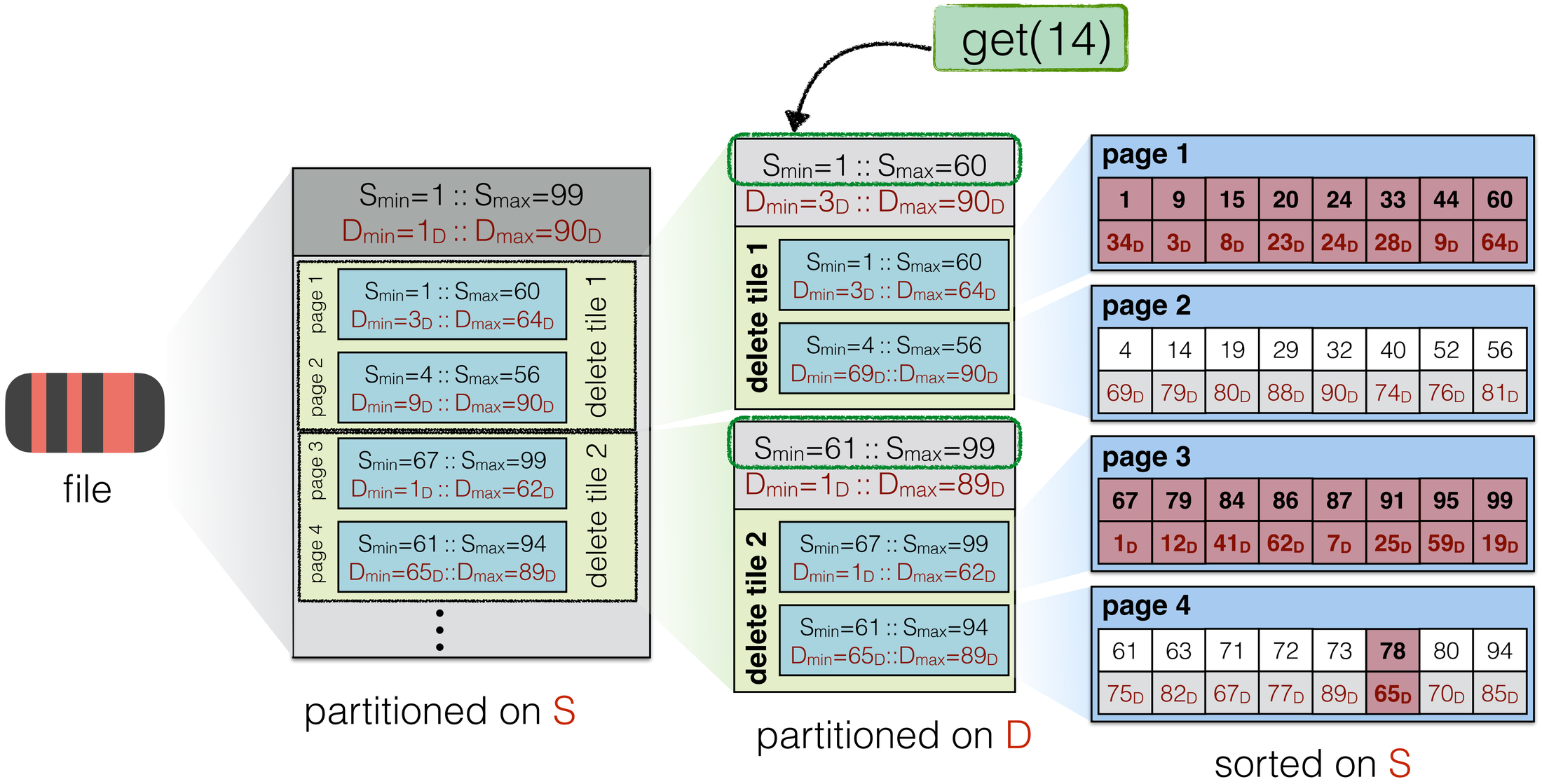


superior delete performance

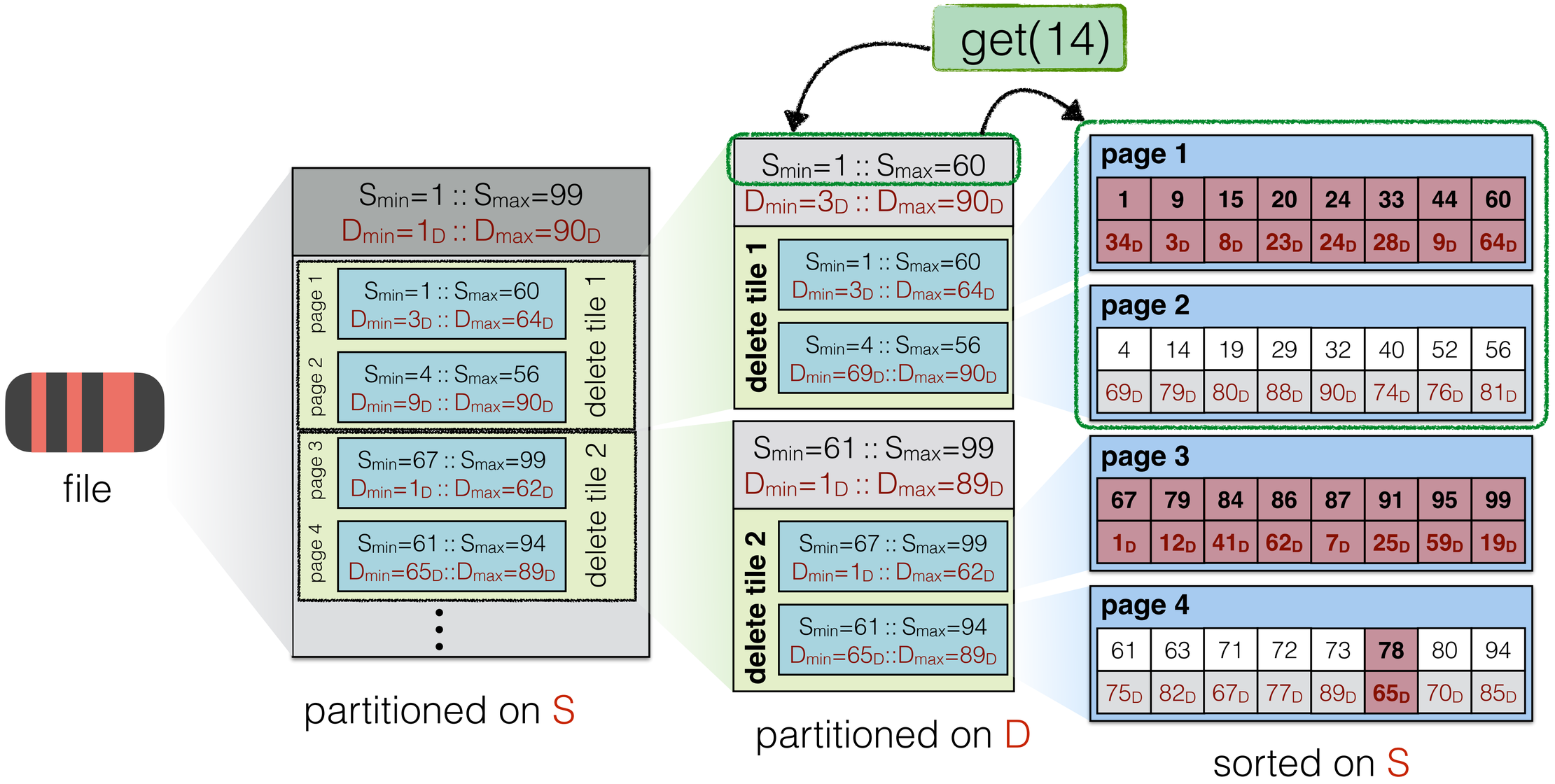
up to 2.5x

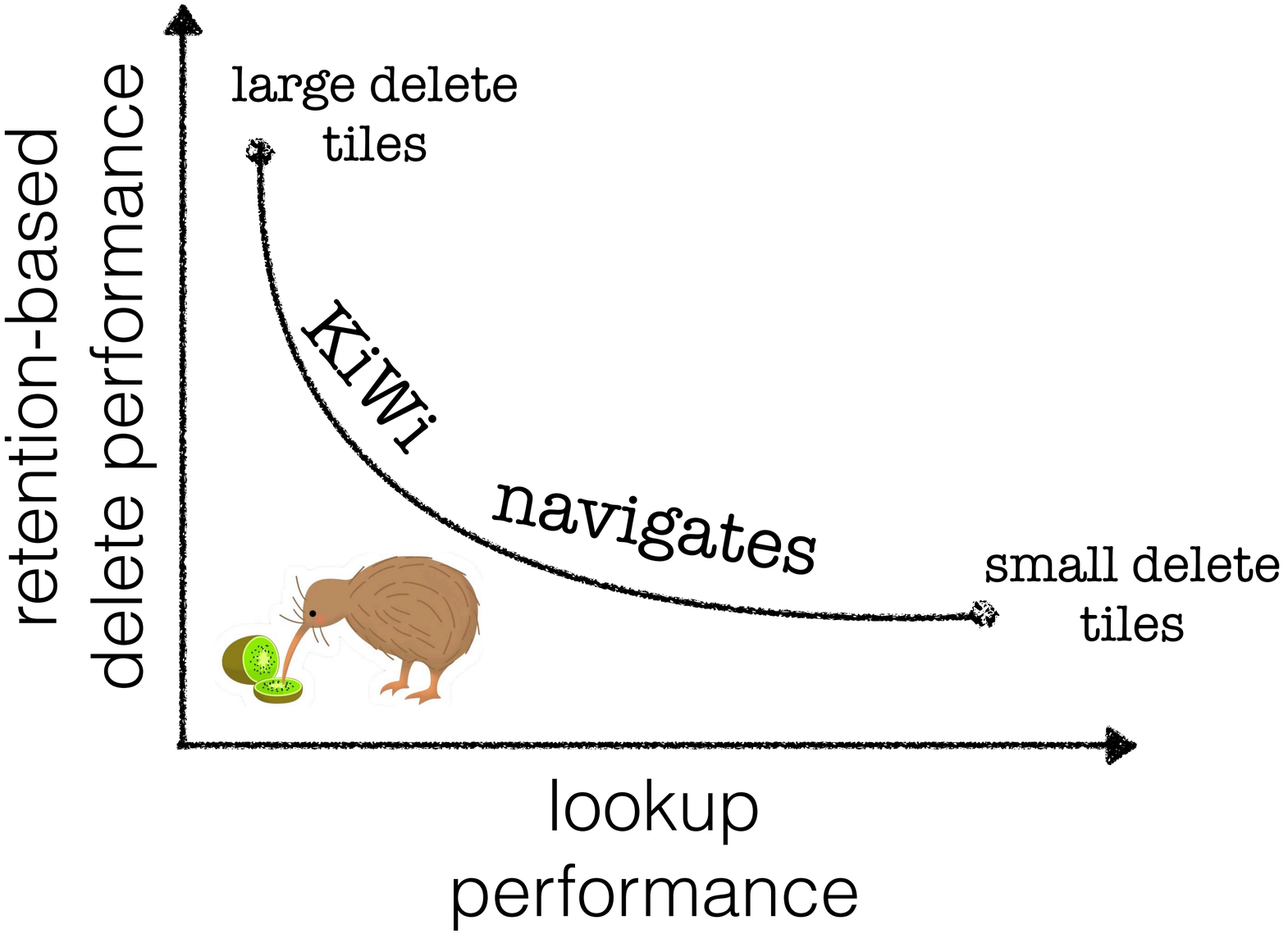


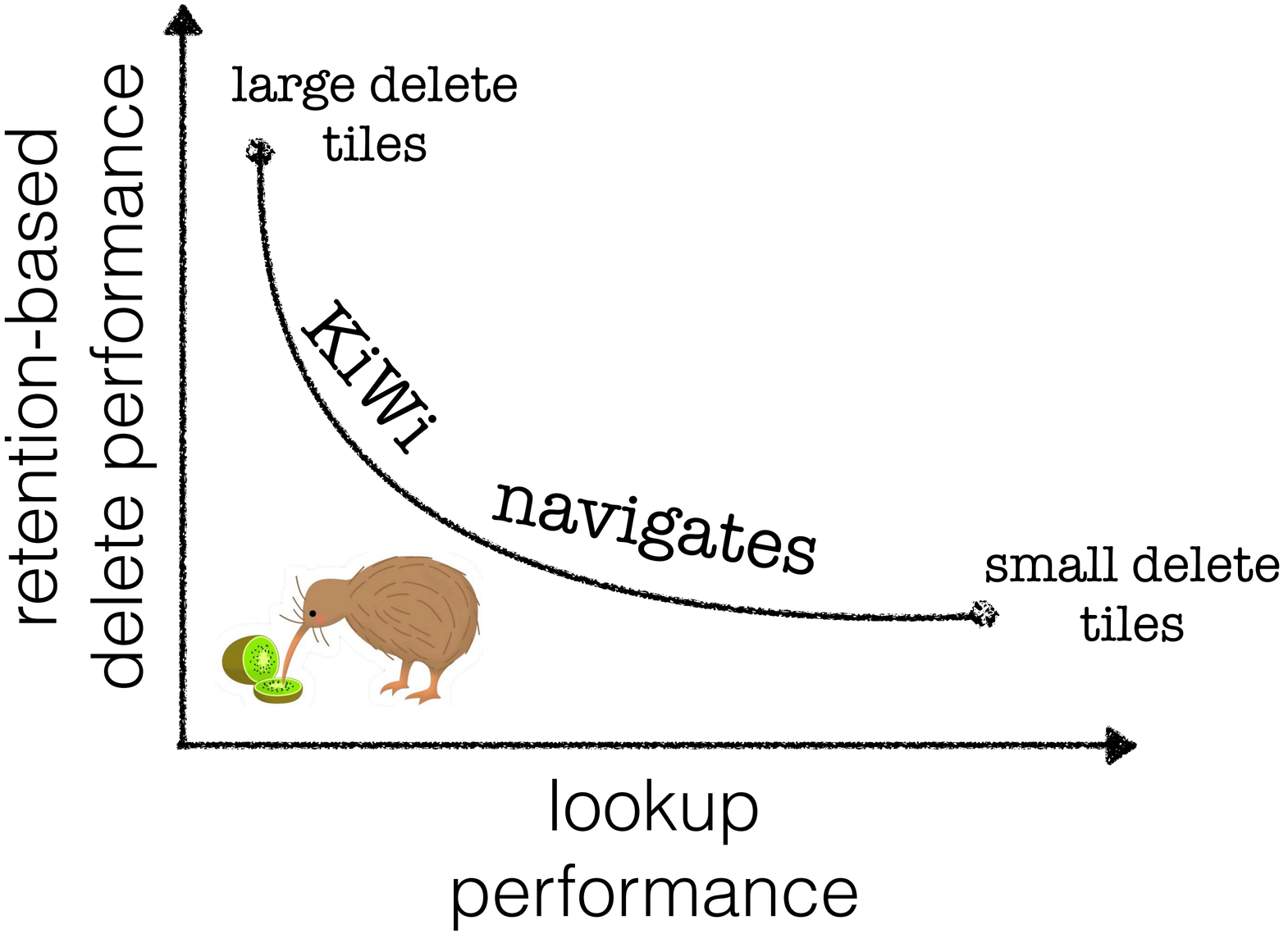
Key Weaving storage layout

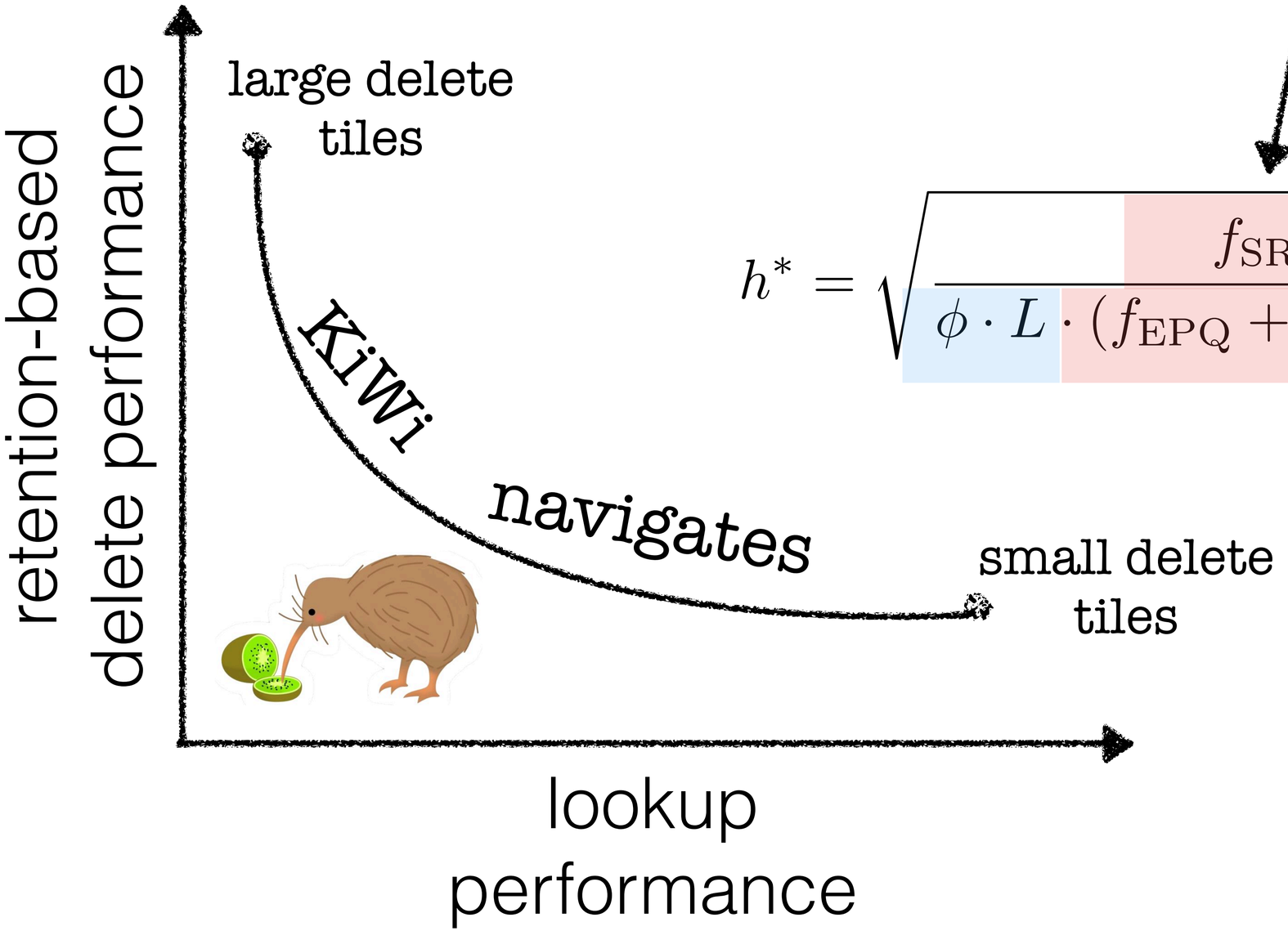


Key Weaving storage layout




retention-based
delete performance
large delete
tiles
KiWi
navigates
small delete
tiles
lookup
performance


retention-based
delete performance
large delete
tiles
KiWi
navigates
small delete
tiles
lookup
performance



workload

data
structure

$$h^* = \sqrt{\frac{f_{\text{SRD}} \cdot \frac{N}{B}}{\phi \cdot L \cdot (f_{\text{EPQ}} + f_{\text{PQ}}) \cdot + L \cdot f_{\text{SRQ}}}}$$

h^* = optimal delete tile size

f_{SRD} = prop. of retention-based deletes

f_{EPQ} = prop. of empty point queries

f_{PQ} = prop. of non-empty point queries

f_{SRQ} = prop. of short range queries

L = levels in tree

N = entries in tree

B = entries in a page

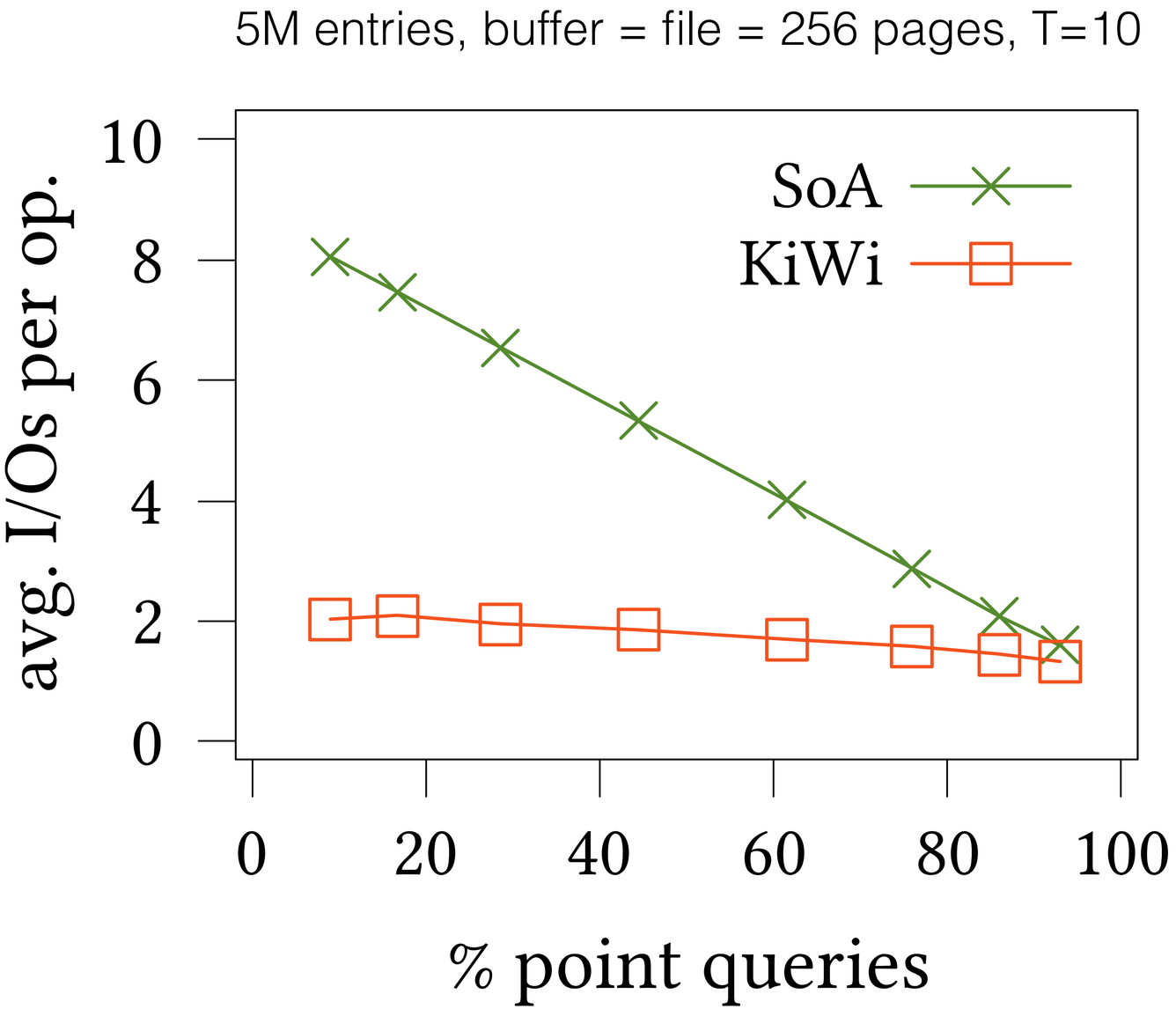
ϕ = false positive rate of query filter

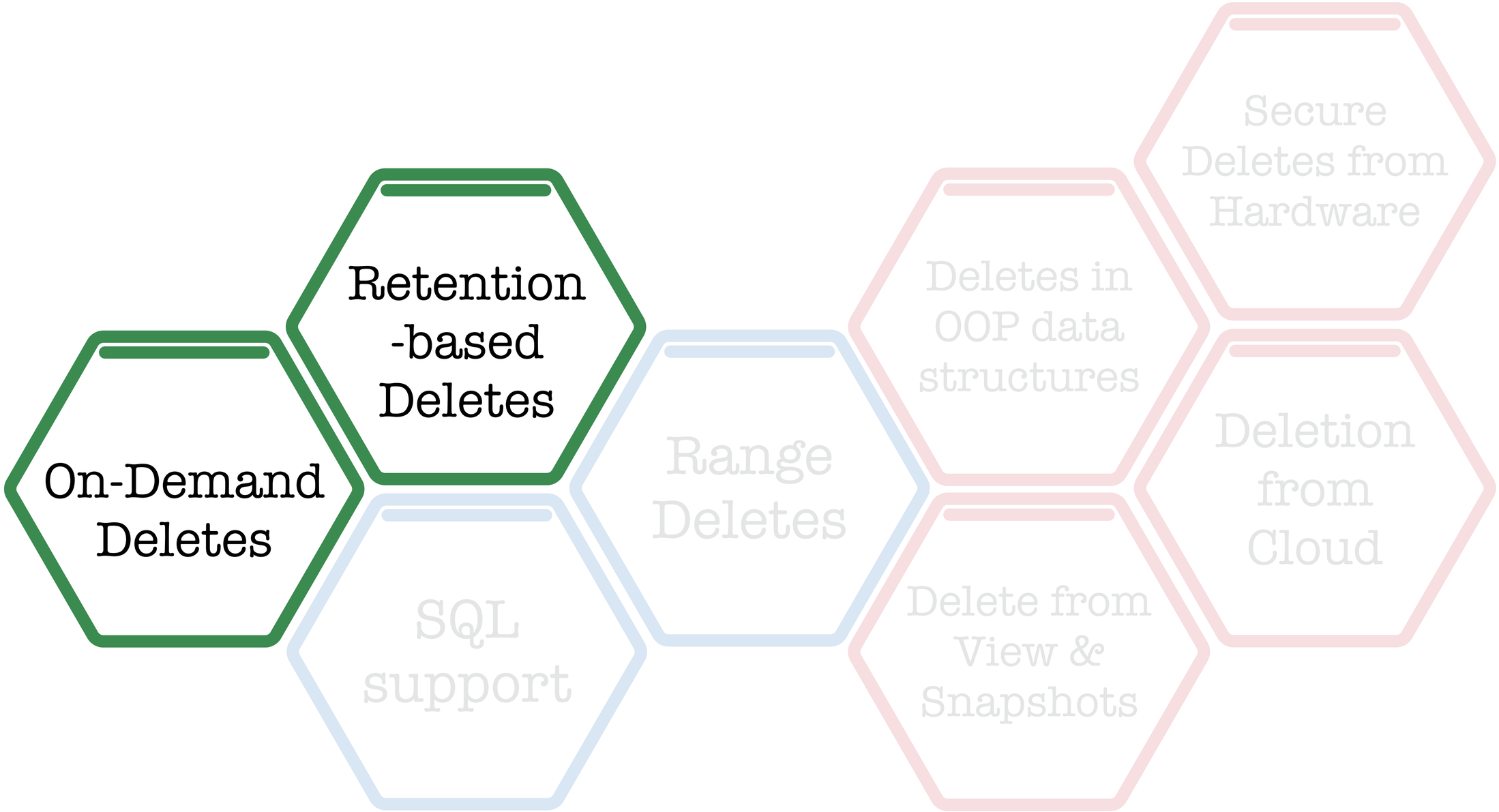
better overall performance

up to 4x

superior delete performance

up to 2.5x




On-Demand Deletes
Retention -based Deletes
SQL support
Range Deletes
Delete from View & Snapshots
Deletes in OOP data structures
Deletion from Cloud
Secure Deletes from Hardware

FADE

Range Deletes

SQL support

Deletes in OOP data structures

Secure Deletes from Hardware

Deletion from Cloud

Delete from View & Snapshots

FADE

SQL support

Range Deletes

Deletes in OOP data structures

Delete from View & Snapshots

Secure Deletes from Hardware

Deletion from Cloud

COSI 167A
Advanced Data Systems

Class 12
Efficient Deletes in LSM-Engines

Prof. Subhadeep Sarkar

Brandeis UNIVERSITY

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

lab