CS 561: ***Data Systems Architecture***

Class 6

Efficient Deletes in LSM-Engines

Dr. Subhadeep Sarkar

BOSTON UNIVERSITY

DiSC lab

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

Updates: Logistics

First **technical question** is due on **02/07**.

First **review** is due on **02/14**.

Project 1 is now online! Deadline: **02/20**.

Project 1 is a **group project** (2-3 students per group).

The first **student presentation** is next week (on **02/14**)!

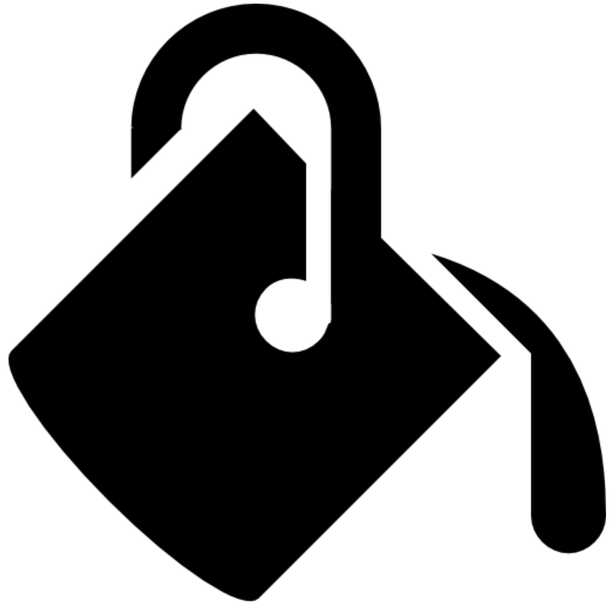
A week before the presentation, discuss the slides with me in OH.

BOSTON UNIVERSITY

How to prepare good slides

don’t use bullets

1 message per slide



1/2 colors

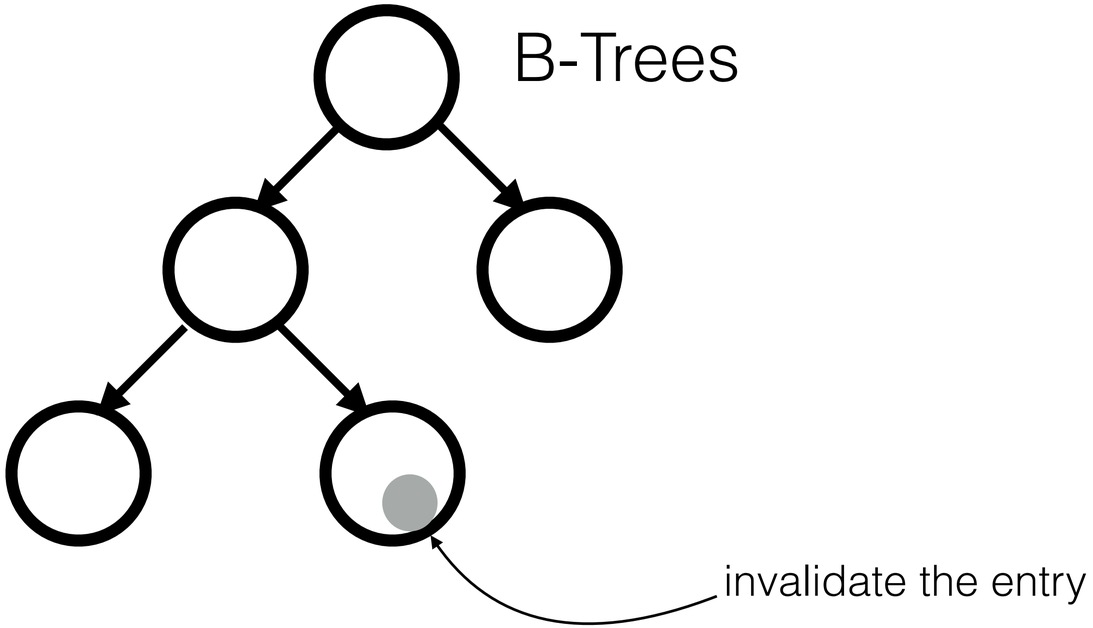
BOSTON UNIVERSITY

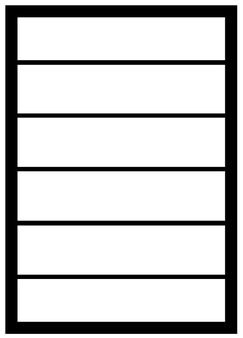
```
<your_favorite_data_structure>::delete (key)
{
  //todo
}
```

How do you delete data?

IN-PLACE

OUT-OF-PLACE

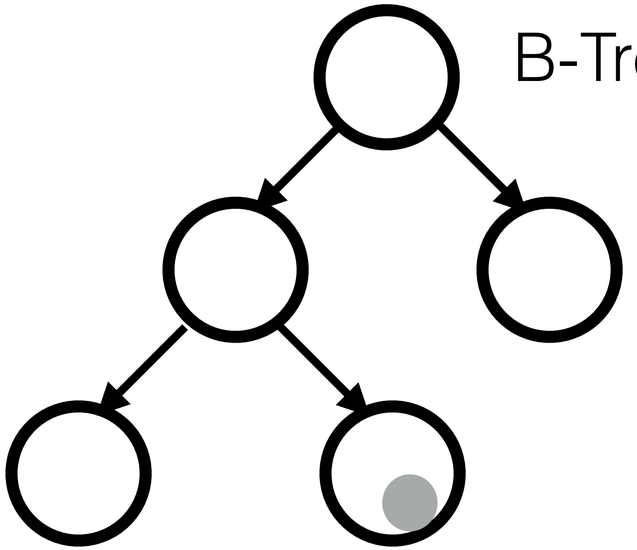


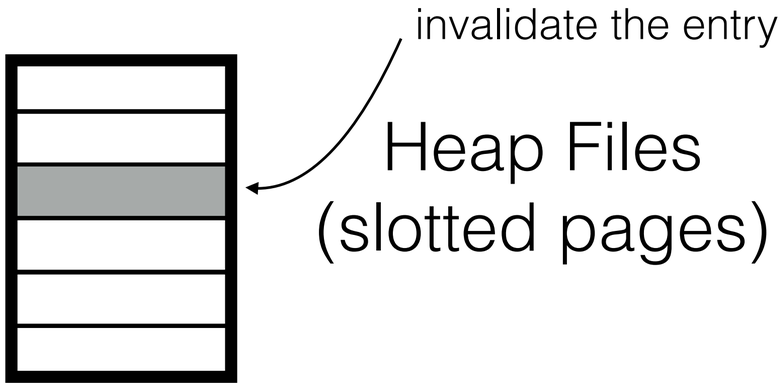


Heap Files
(slotted pages)

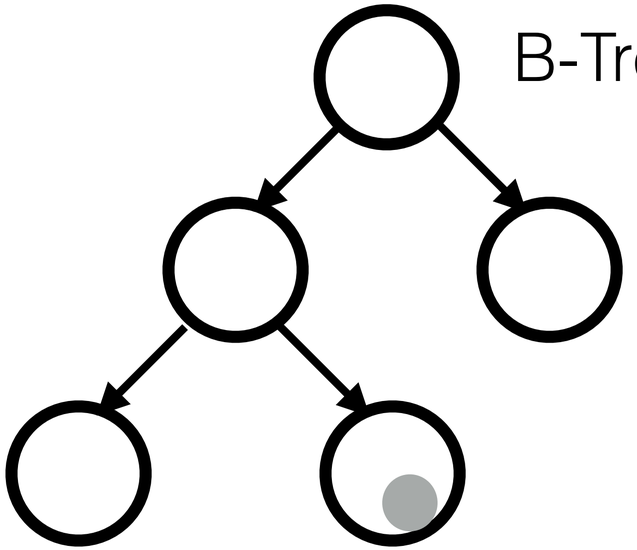
IN-PLACE

OUT-OF-PLACE





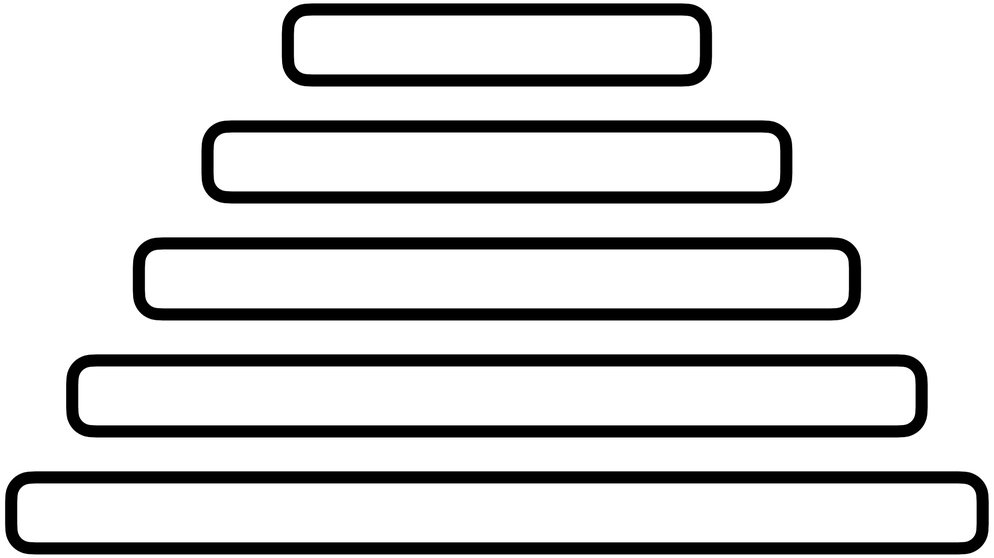
IN-PLACE



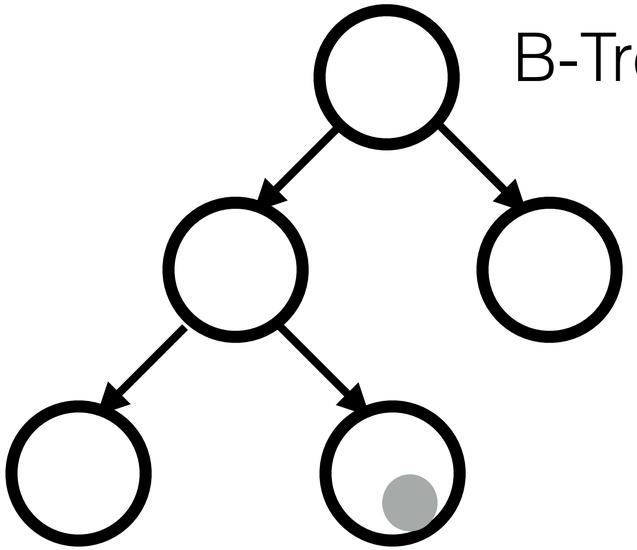
B-Trees

Heap Files
(slotted pages)

OUT-OF-PLACE



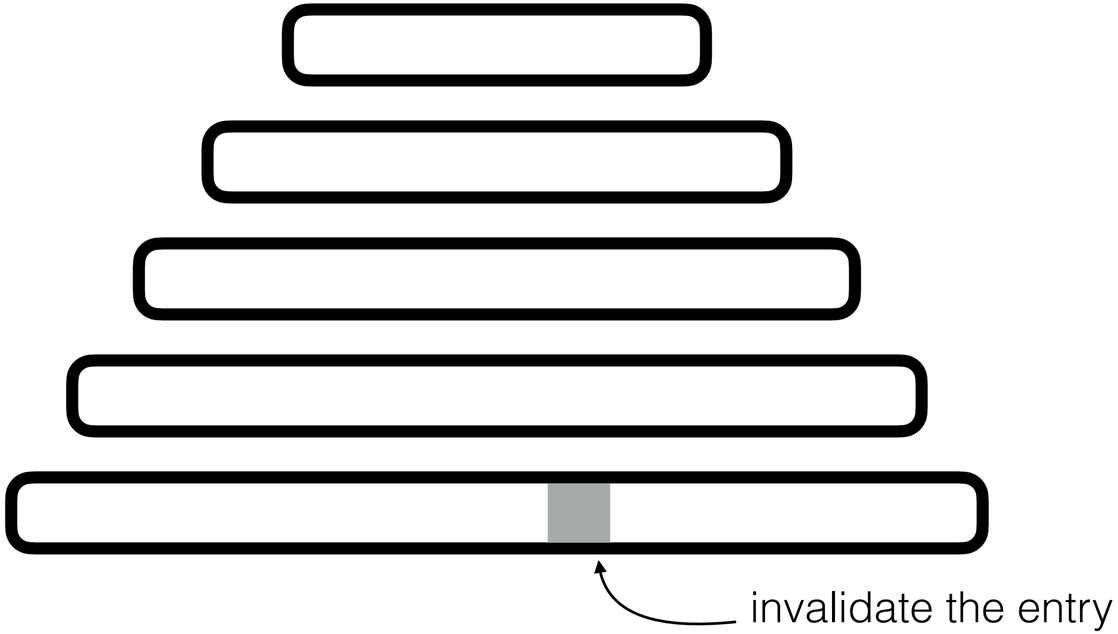
IN-PLACE



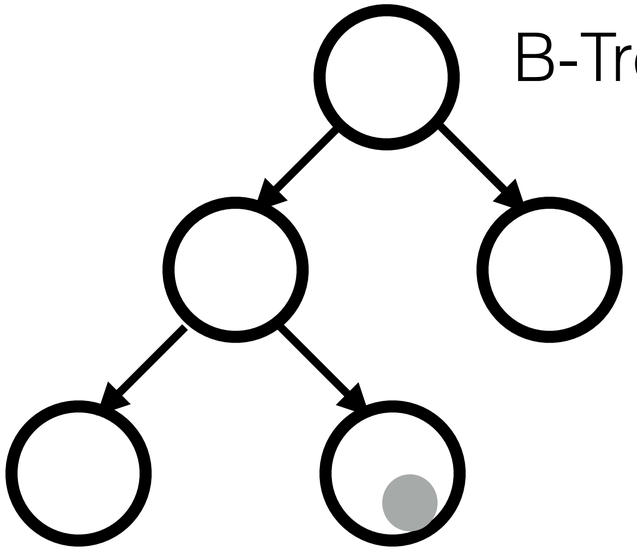
B-Trees

Heap Files
(slotted pages)

OUT-OF-PLACE



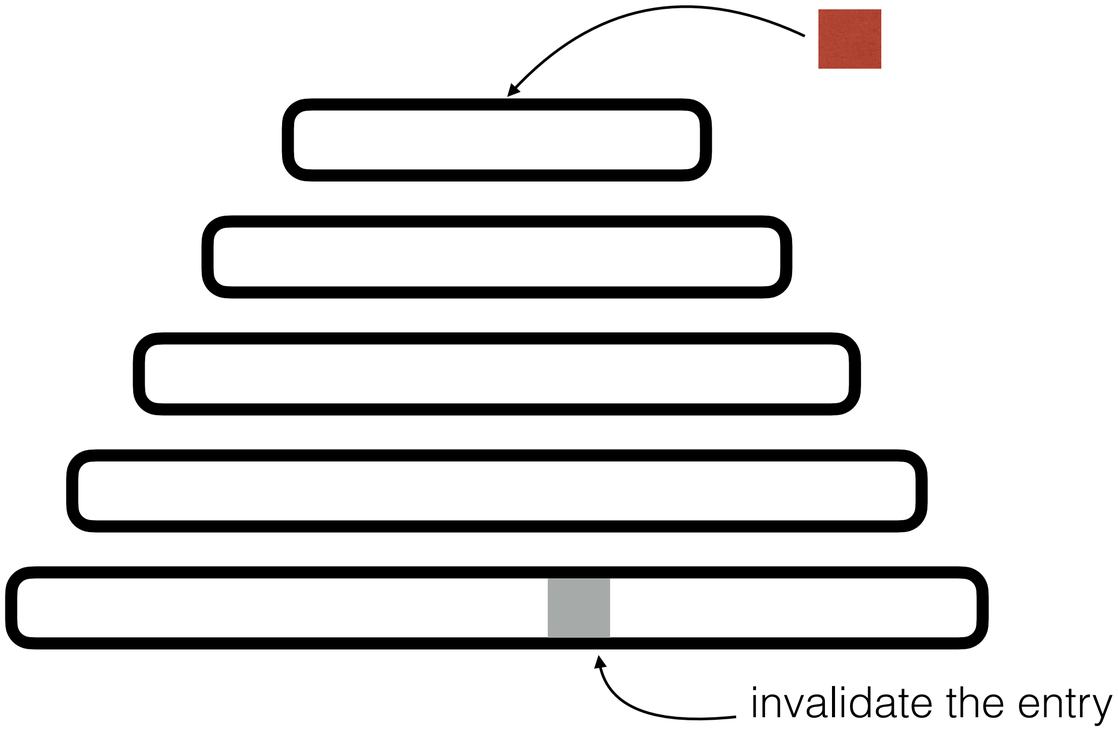
IN-PLACE



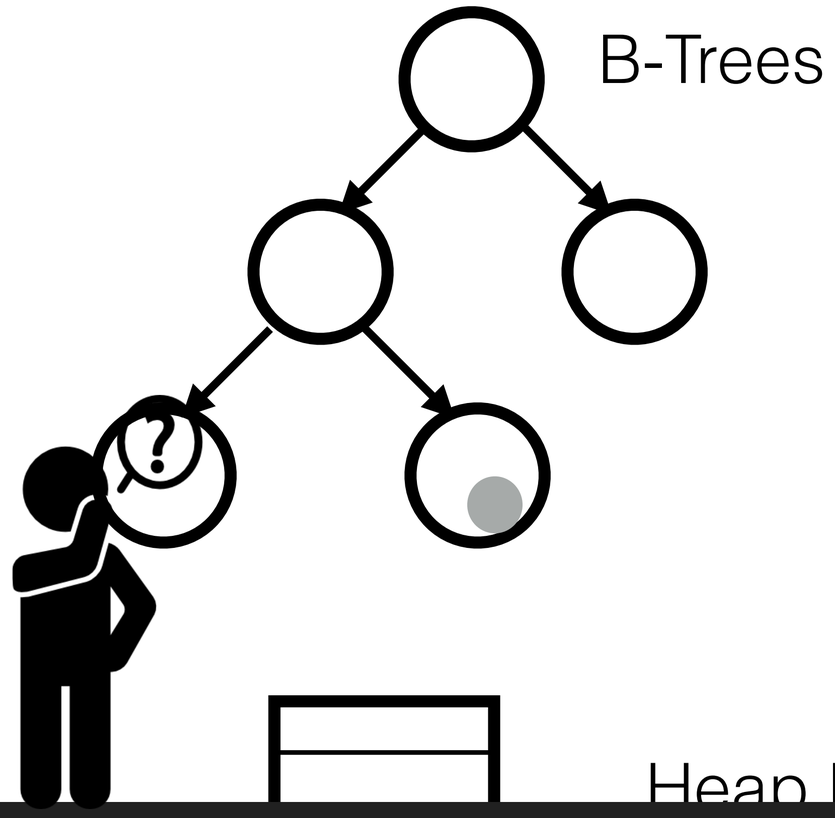
B-Trees

Heap Files
(slotted pages)

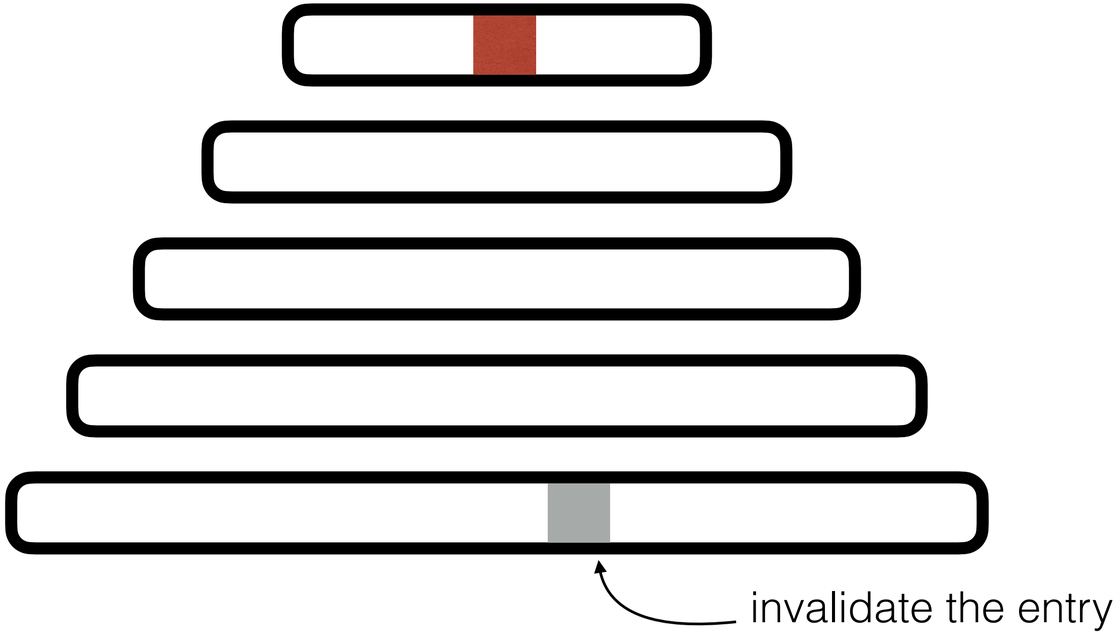
OUT-OF-PLACE



IN-PLACE



OUT-OF-PLACE



What is the tradeoff for deletes?

What is the delete tradeoff?

read

vs.

write

What is the delete tradeoff?

read

vs.

write

Deletes are almost **exclusively** *logical*

Today's talk:

Lethe: A Tunable Delete-Aware LSM-Based Storage Engine

Presented at SIGMOD 2020

DiSC lab

LSM-tree

NoSQL

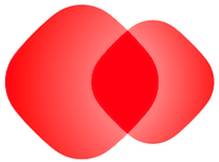

RocksDB


levelDB


SCYLLA


DynamoDB


cassandra


tarantool


Bigtable


APACHE HBASE


riak


accumulo


SQLite

relational


influxdb


QuasarDB

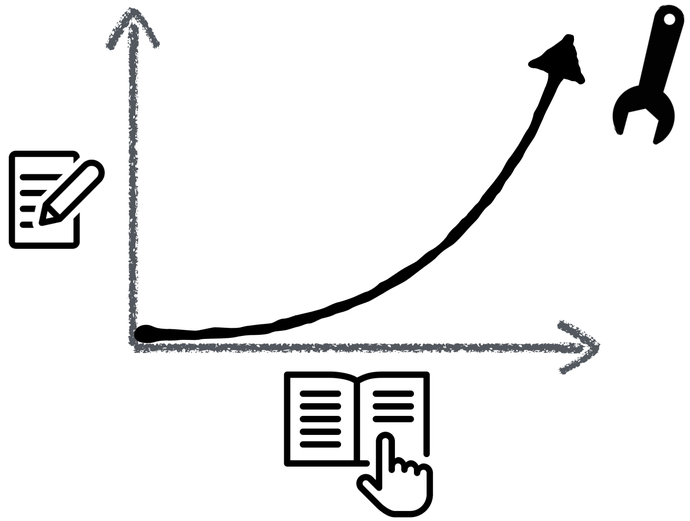
time-series

2023

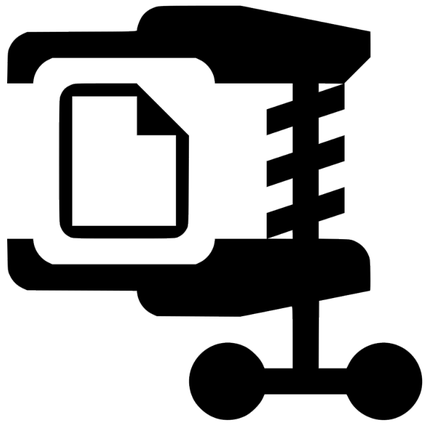
BOSTON UNIVERSITY

Why **LSM** ?

fast writes



tunable read-write performance



good space utilization

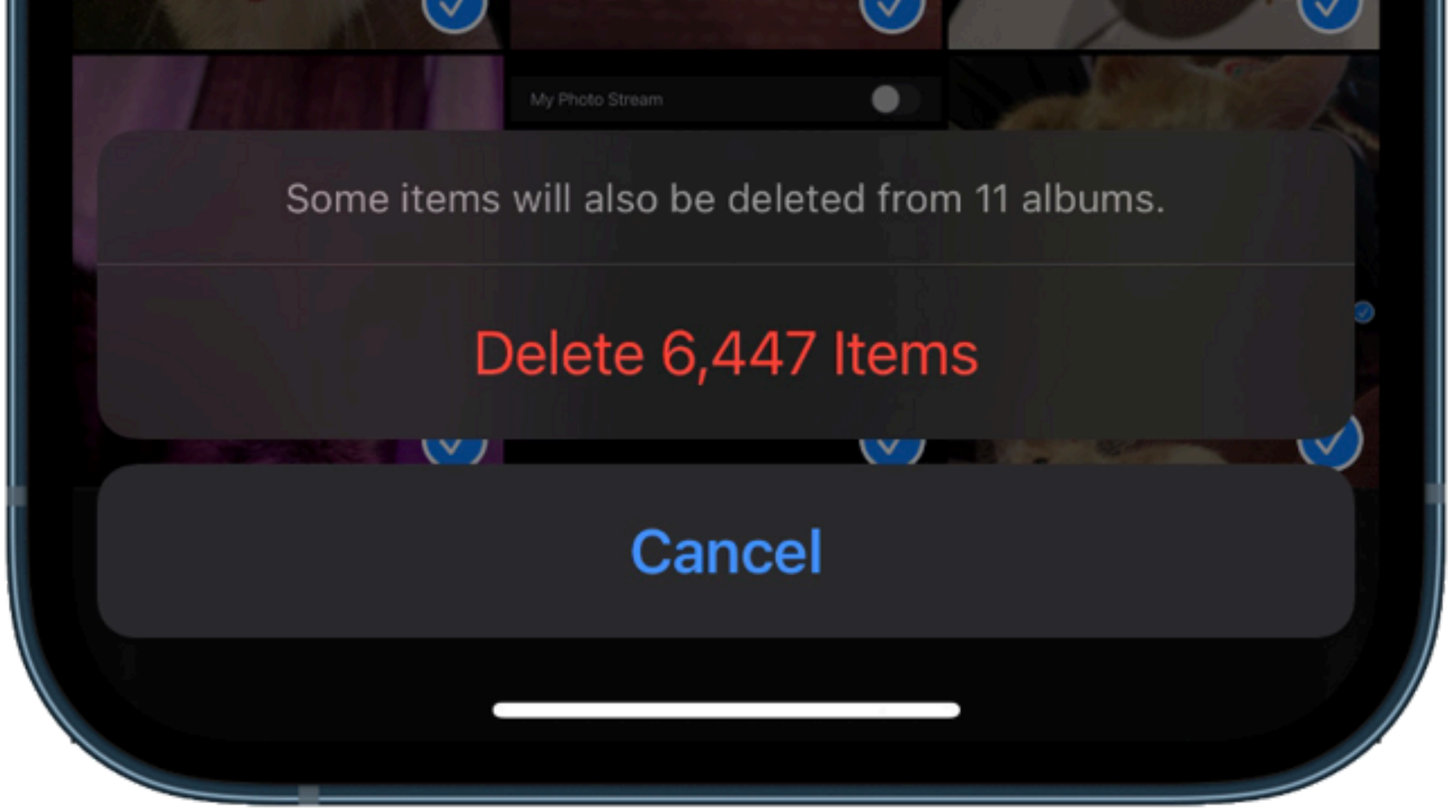
Even years later, Twitter doesn't delete your direct messages

TechCrunch
Feb '19

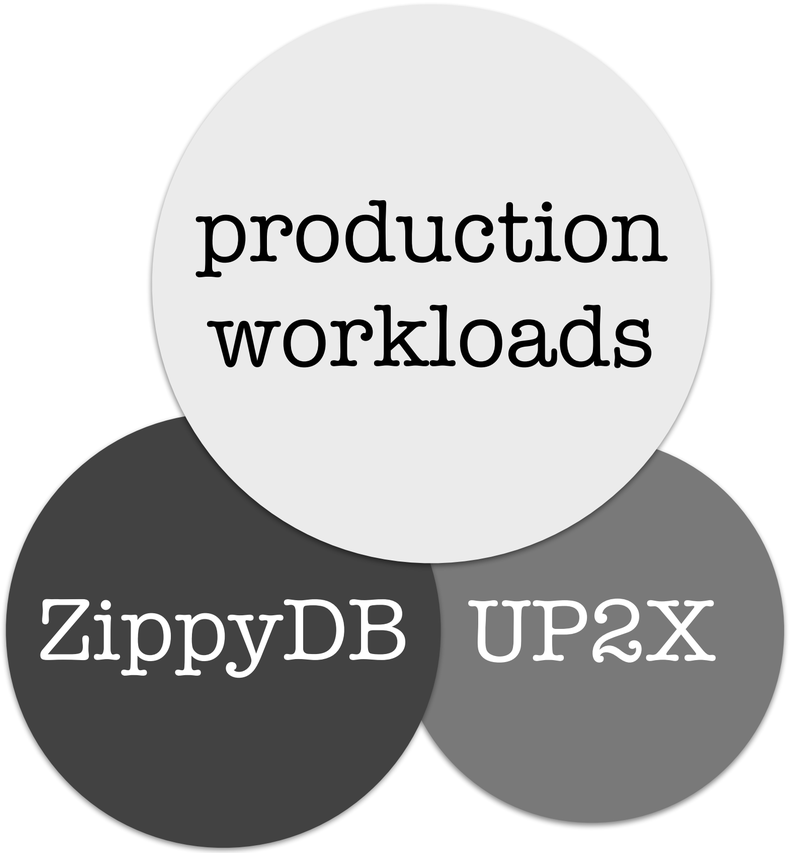
Small Datum
Jan '20

Deletes are fast and slow in an LSM

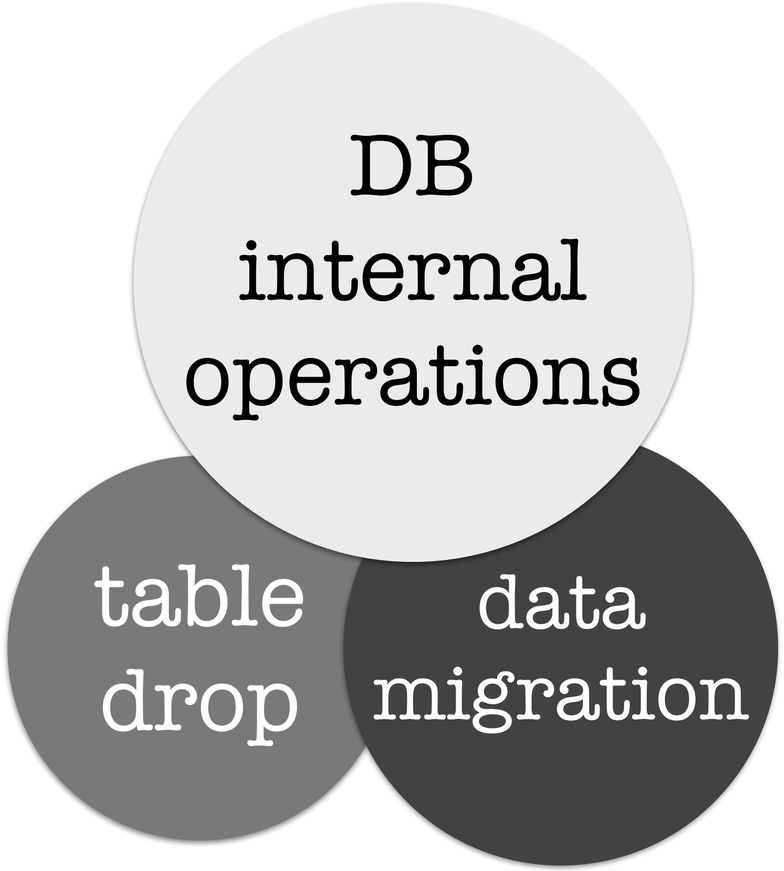
"LSM-based data stores perform suboptimally for workloads with deletes."


My Photo Stream
Some items will also be deleted from 11 albums.
Delete 6,447 Items
Cancel

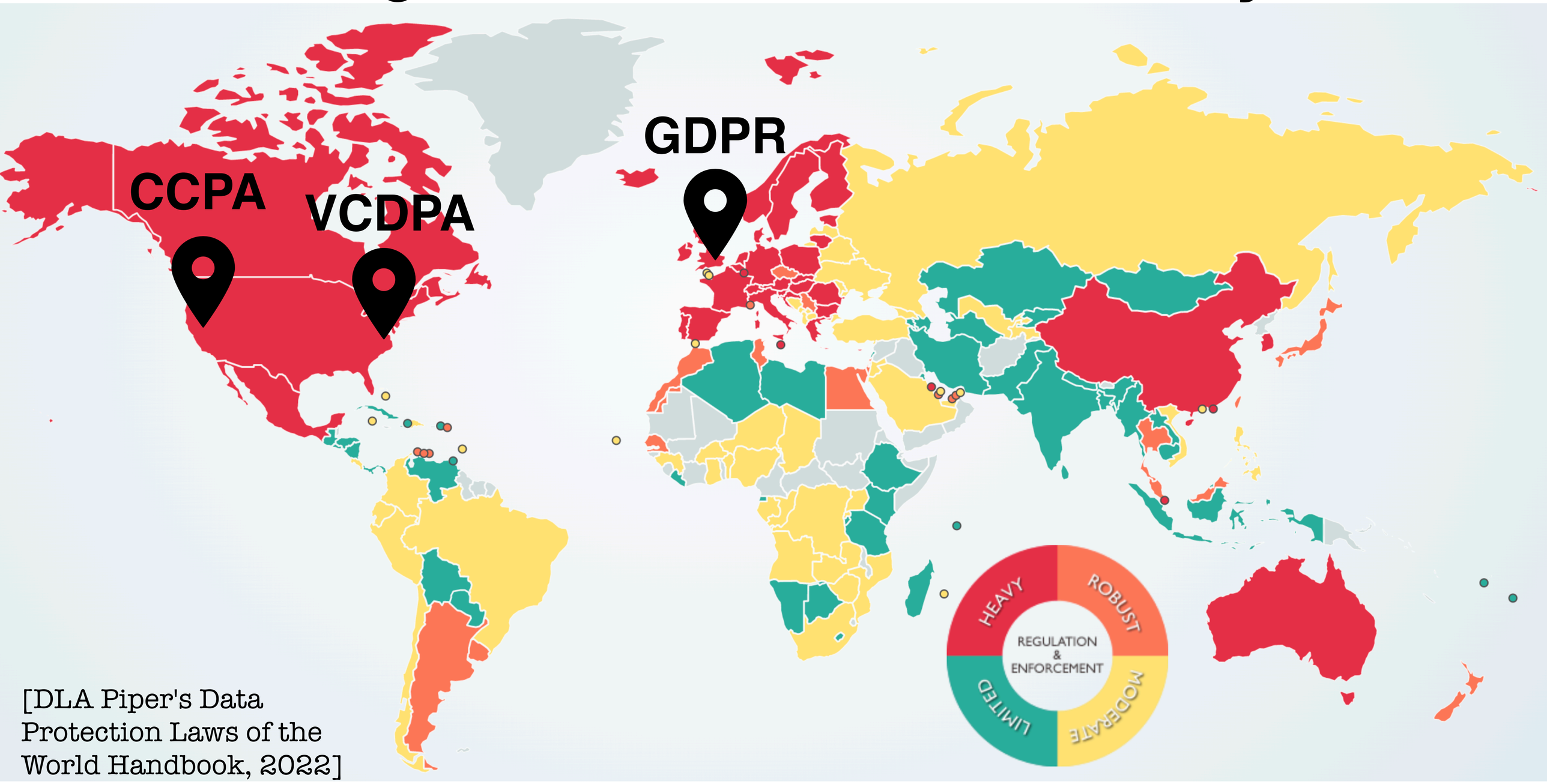
100M+ deletes/day



deletes in **batches**



Logical Deletes & **Data Privacy**



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

GDPR
(EU, UK)

CCPA
(California)

VCPDA
(Virginia)

on-demand

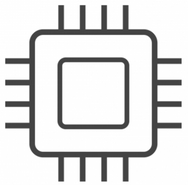
delete all data for user X within D days

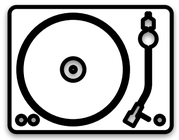
rolling

keep deleting all data older than D days

A reminder on how LSM-trees work!

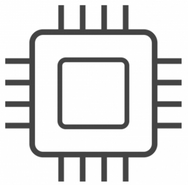
log-structured merge-tree

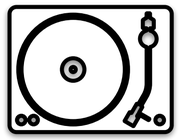




buffer

log-structured merge-tree

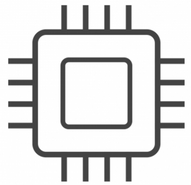


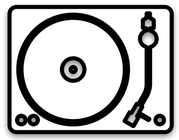


buffer

2	6	1	4

log-structured merge-tree

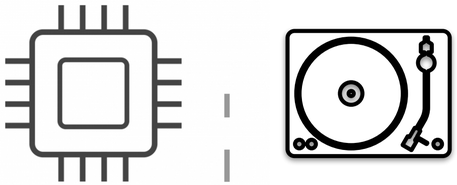




buffer

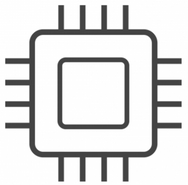
1	2	4	6

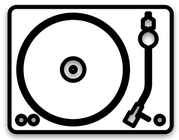
log-structured merge-tree



buffer 

log-structured merge-tree

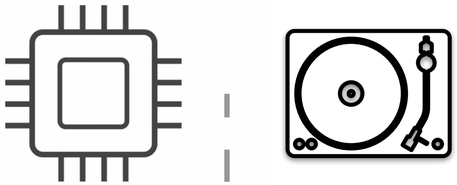




buffer

L1

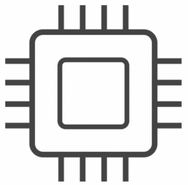
log-structured merge-tree

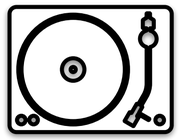


buffer

L1

log-structured merge-tree

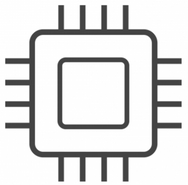


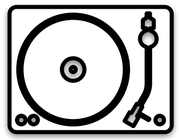


buffer

L1

log-structured merge-tree





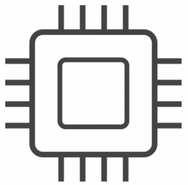
buffer

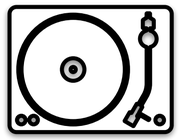
L1

L2

compaction

log-structured merge-tree





buffer

L1

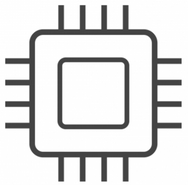
L2

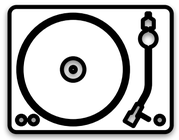
L3

size ratio = T

exponentially larger capacity

log-structured merge-tree





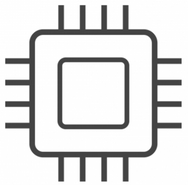
buffer 

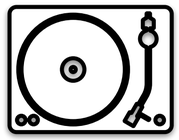
L1 

L2 

L3 

log-structured merge-tree





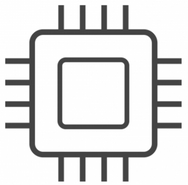
buffer 

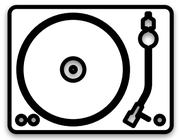
L1 

L2 

L3 

log-structured merge-tree





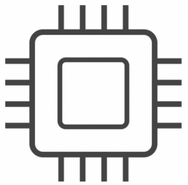
buffer

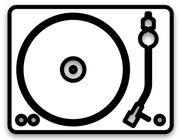
L1

L2

L3

log-structured merge-tree





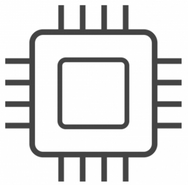
buffer

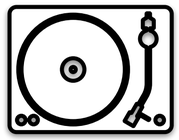
L1

L2

L3

log-structured merge-tree





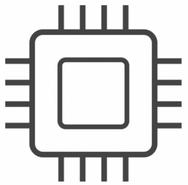
buffer

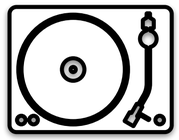
L1

L2

L3

log-structured merge-tree





buffer

L1

L2

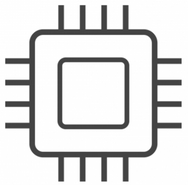
L3

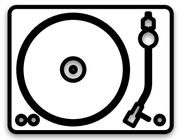
burst of I/Os

prolonged write stalls

How do we avoid this?

log-structured merge-tree





buffer

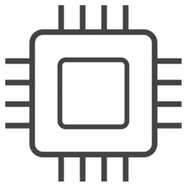
partial compaction

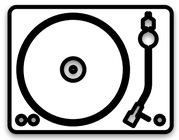
L1

L2

L3

log-structured merge-tree





buffer

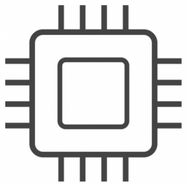
partial compaction

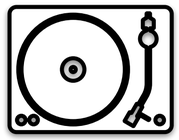
L1

L2

L3

log-structured merge-tree





buffer

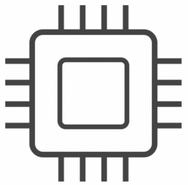
partial compaction

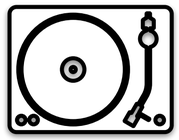
L1

L2

L3

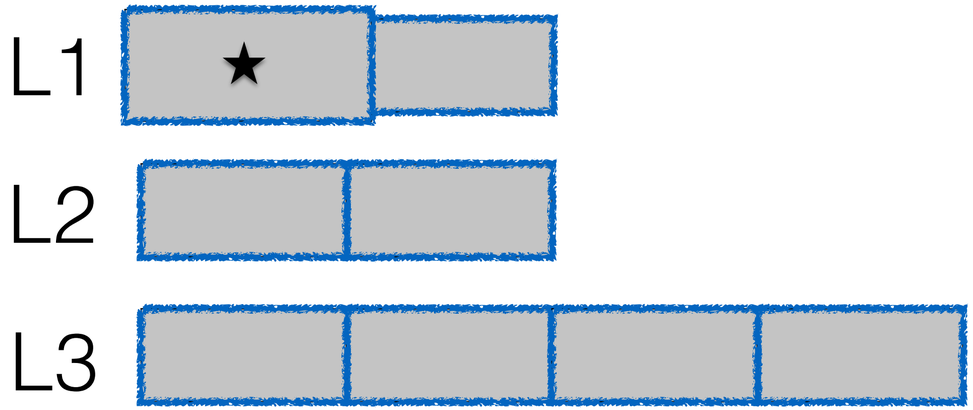
log-structured merge-tree



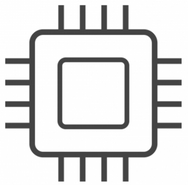


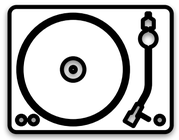
buffer

partial compaction



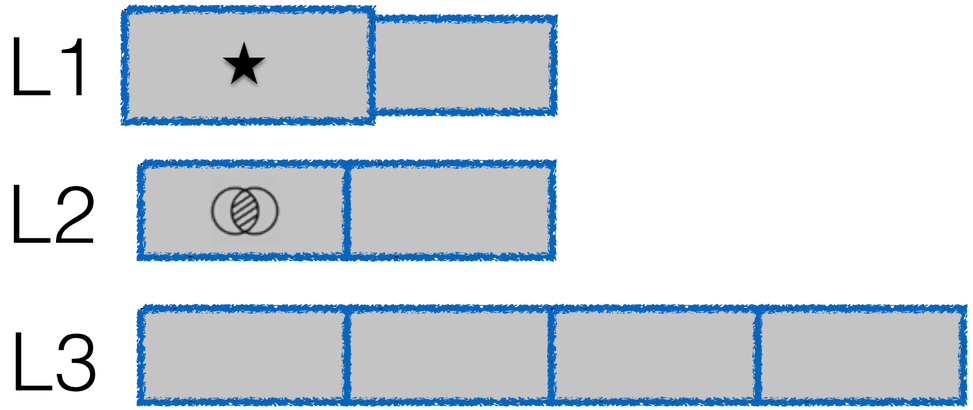
log-structured merge-tree



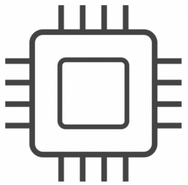


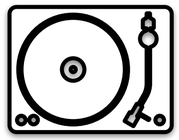
buffer

partial compaction



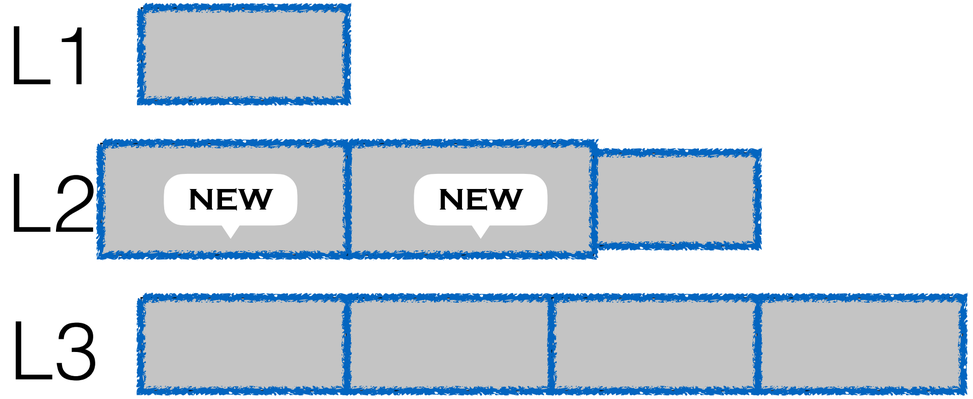
log-structured merge-tree



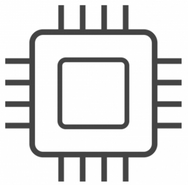


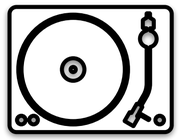
buffer

partial compaction



log-structured merge-tree





buffer

partial compaction

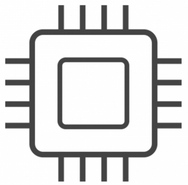
L1

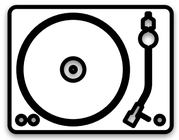
L2

NEW NEW

L3

log-structured merge-tree





buffer

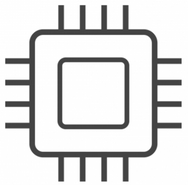
partial compaction

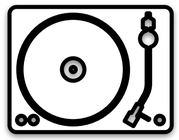
L1

L2

L3

log-structured merge-tree





buffer

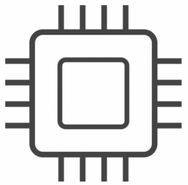
partial compaction

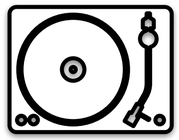
L1

L2

L3

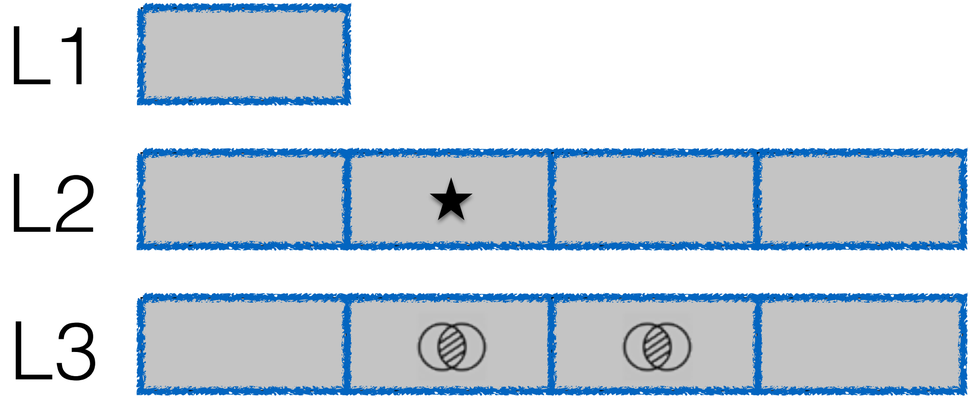
log-structured merge-tree



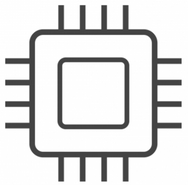


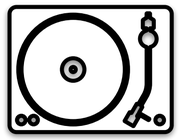
buffer

partial compaction



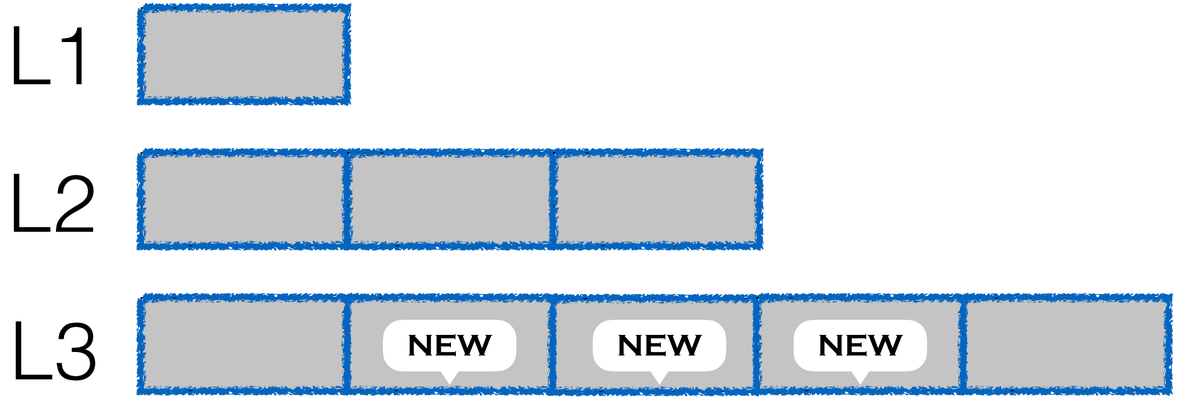
log-structured merge-tree





buffer

partial compaction

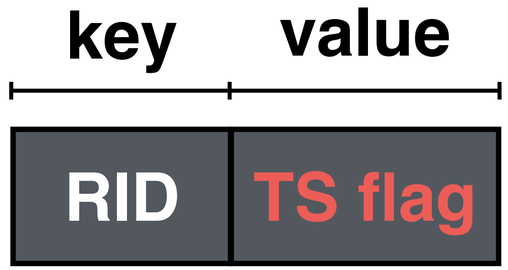


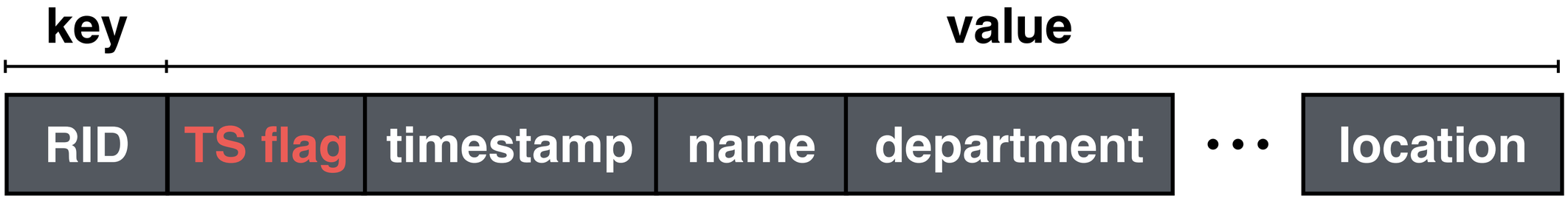
amortized compaction cost

OK! But how to delete in LSM?

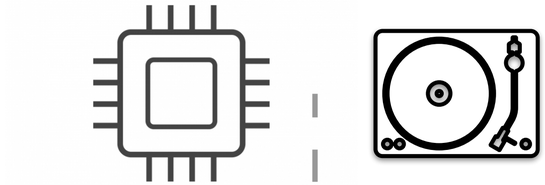
deletes in LSM-tree

delete := insert tombstone





deletes in LSM-tree



delete(5)

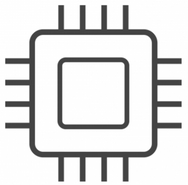
buffer 5

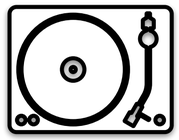
L1

L2 5

L3 5

deletes in LSM-tree





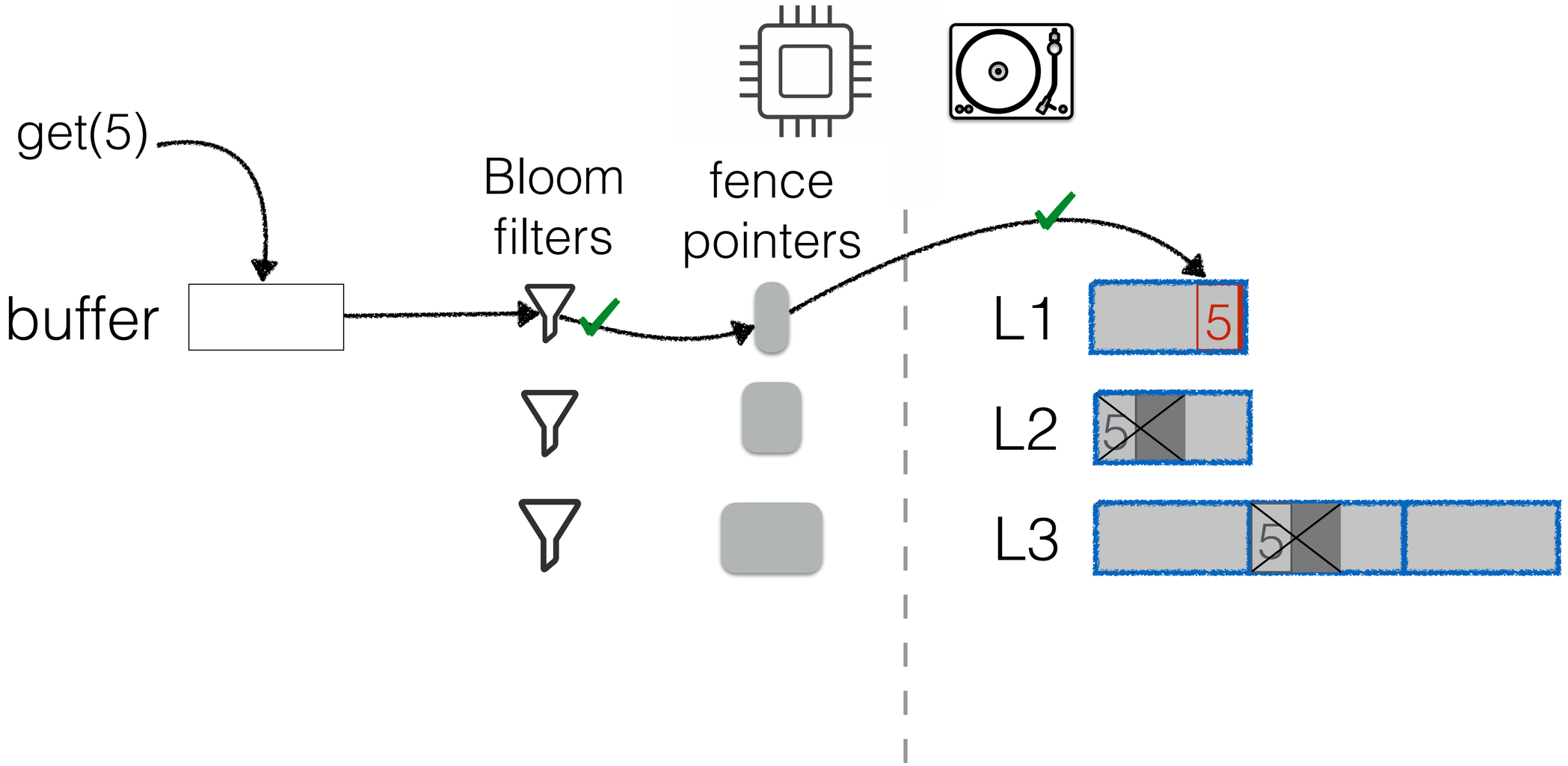
buffer

L1 5

L2 5

L3 5

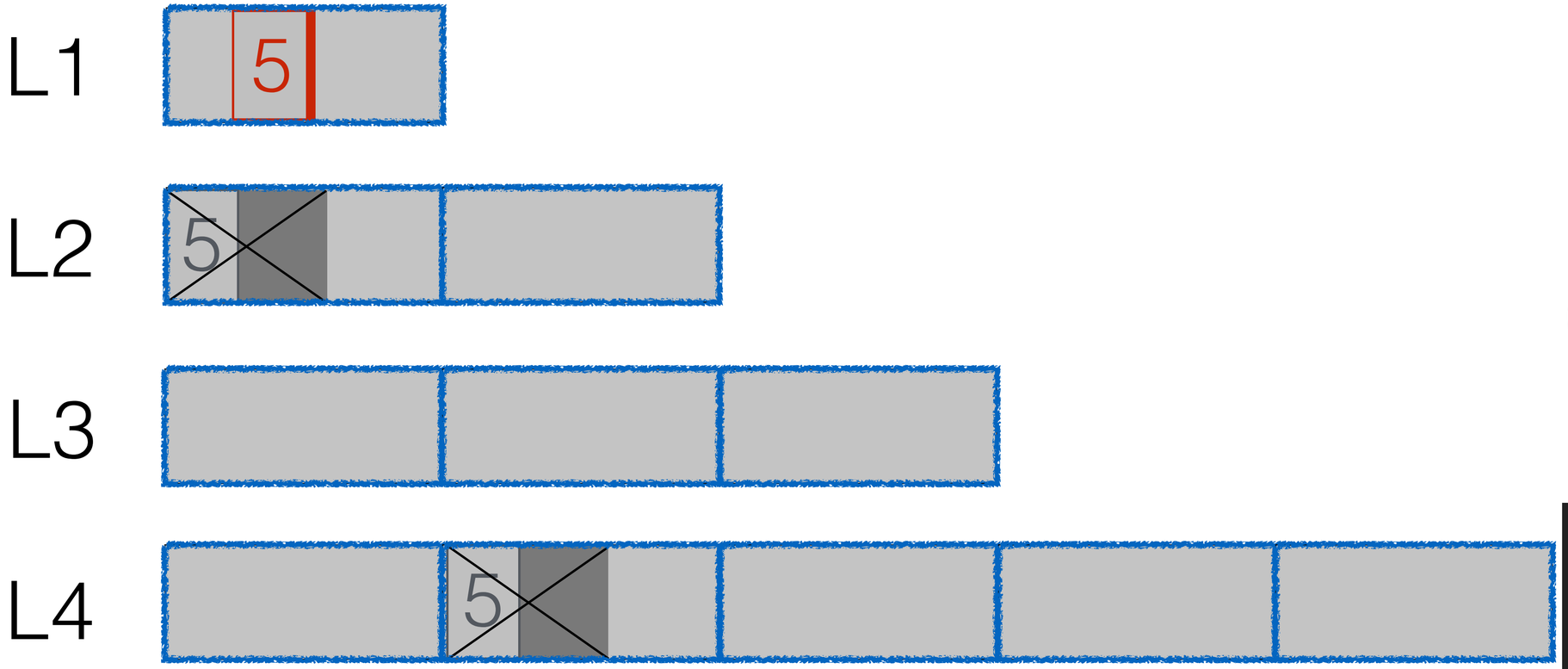
deletes in LSM-tree



the problems

out-of-place deletes

out-of-place deletes



Problem?

out-of-place deletes

Bloom filters

L1

L2

L3

L4

poor read perf. ✗

write amplification ✗

space amplification ✗

the problems

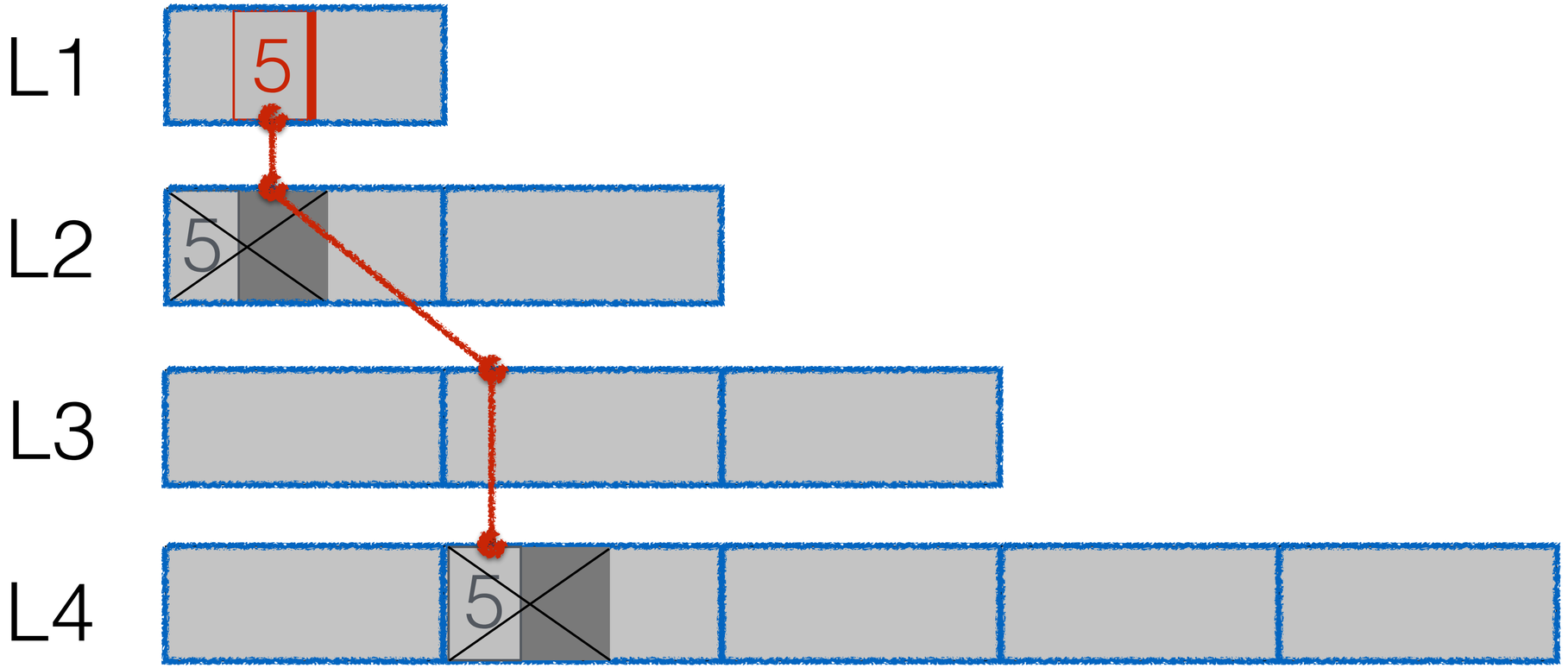
poor read perf.

write amplification

space amplification

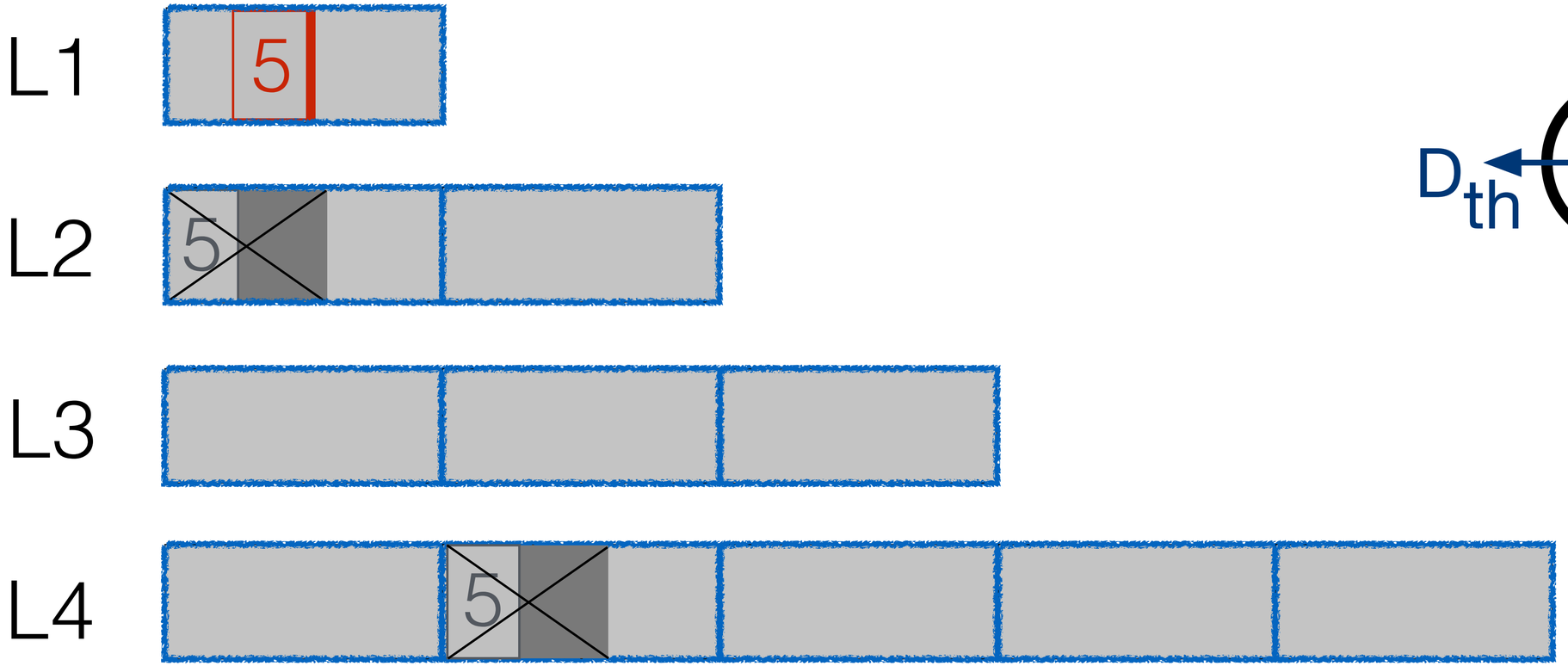
delete persistence latency

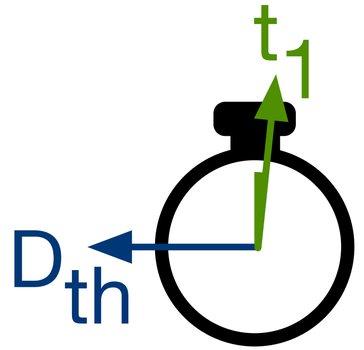
delete persistence latency



delete persistence latency

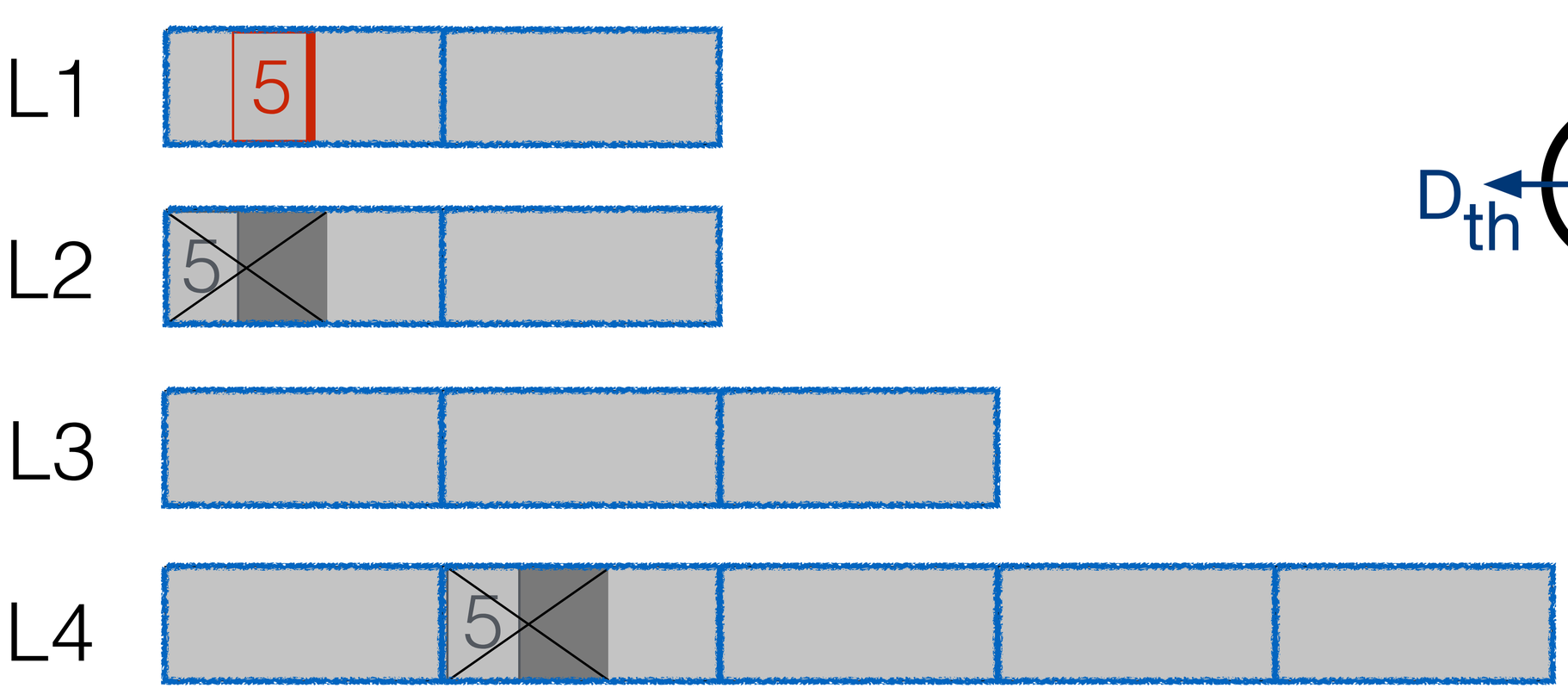
delete(5) within a threshold time: **D_{th}**

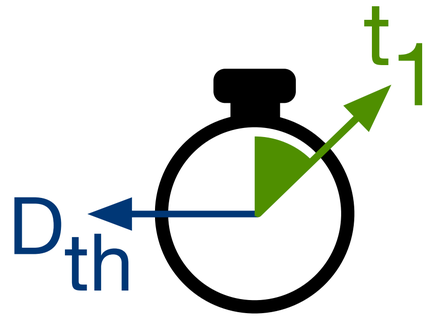




delete persistence latency

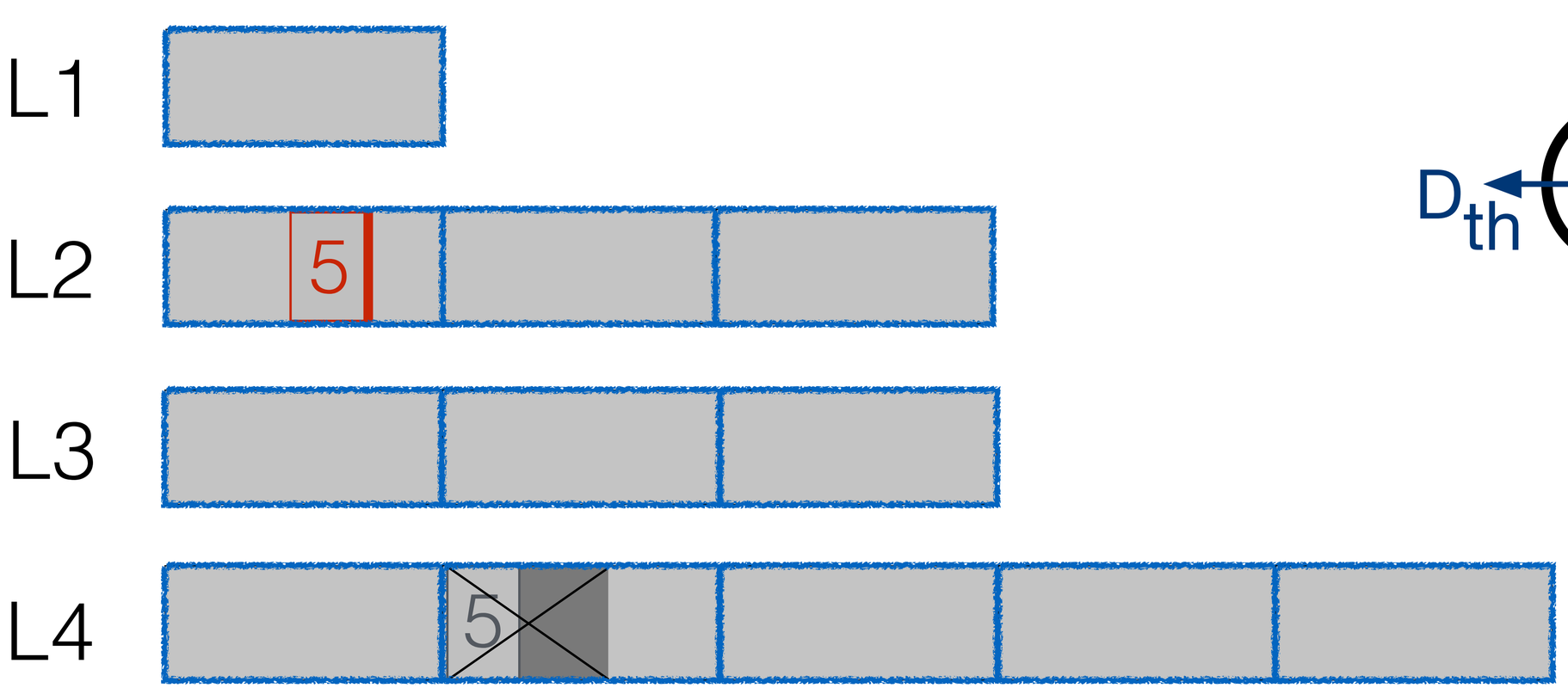
delete(5) within a threshold time: **D_{th}**

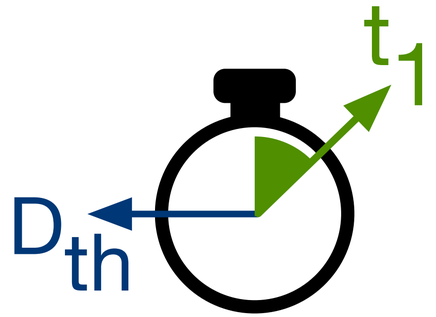




delete persistence latency

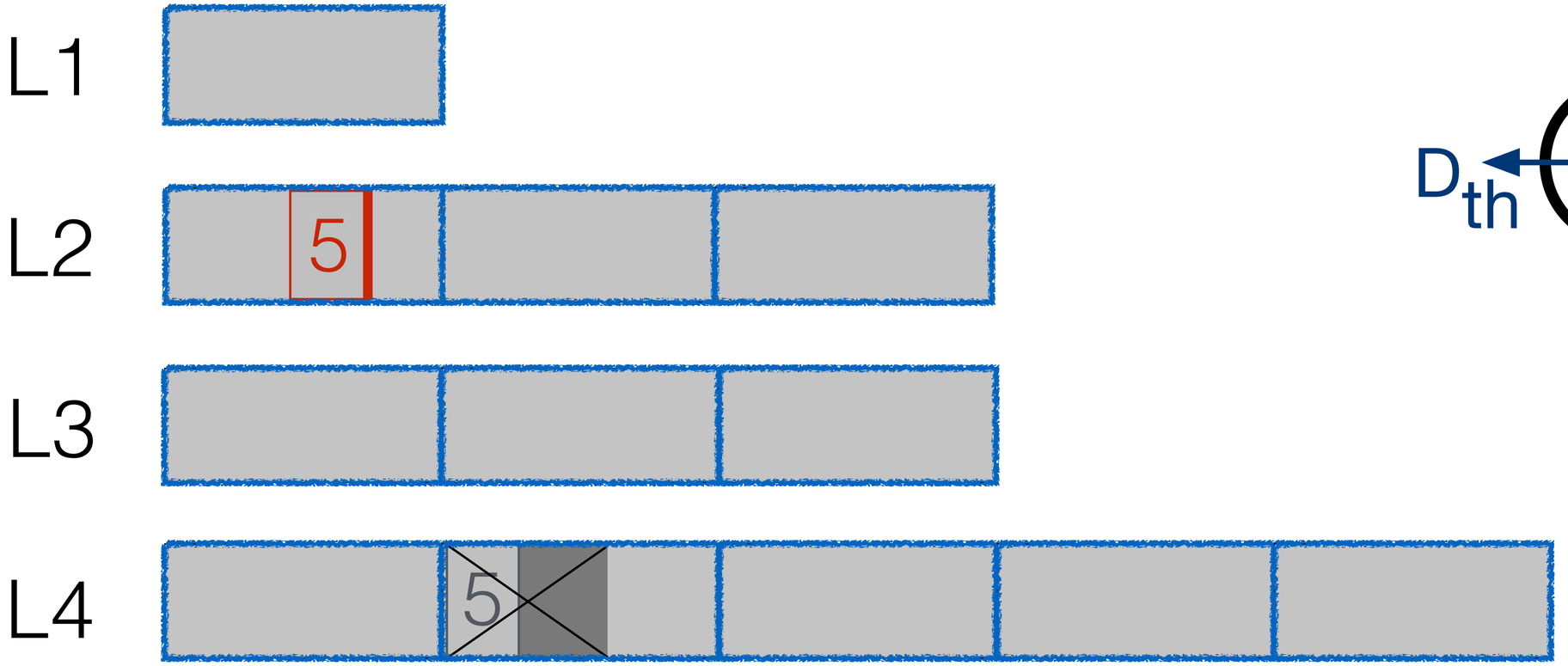
delete(5) within a threshold time: **D_{th}**

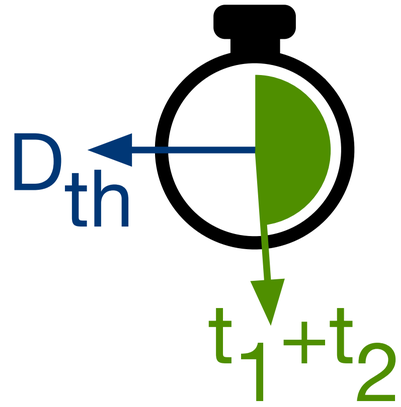




delete persistence latency

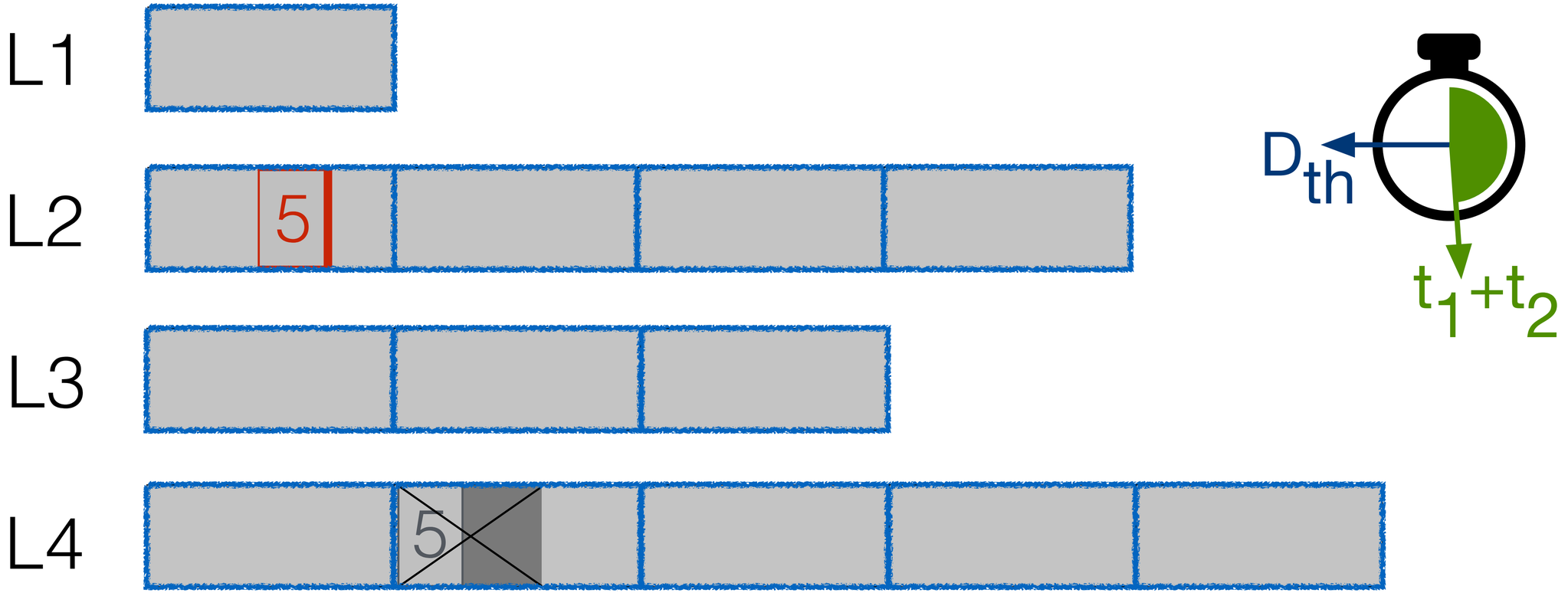
delete(5) within a threshold time: **D_{th}**





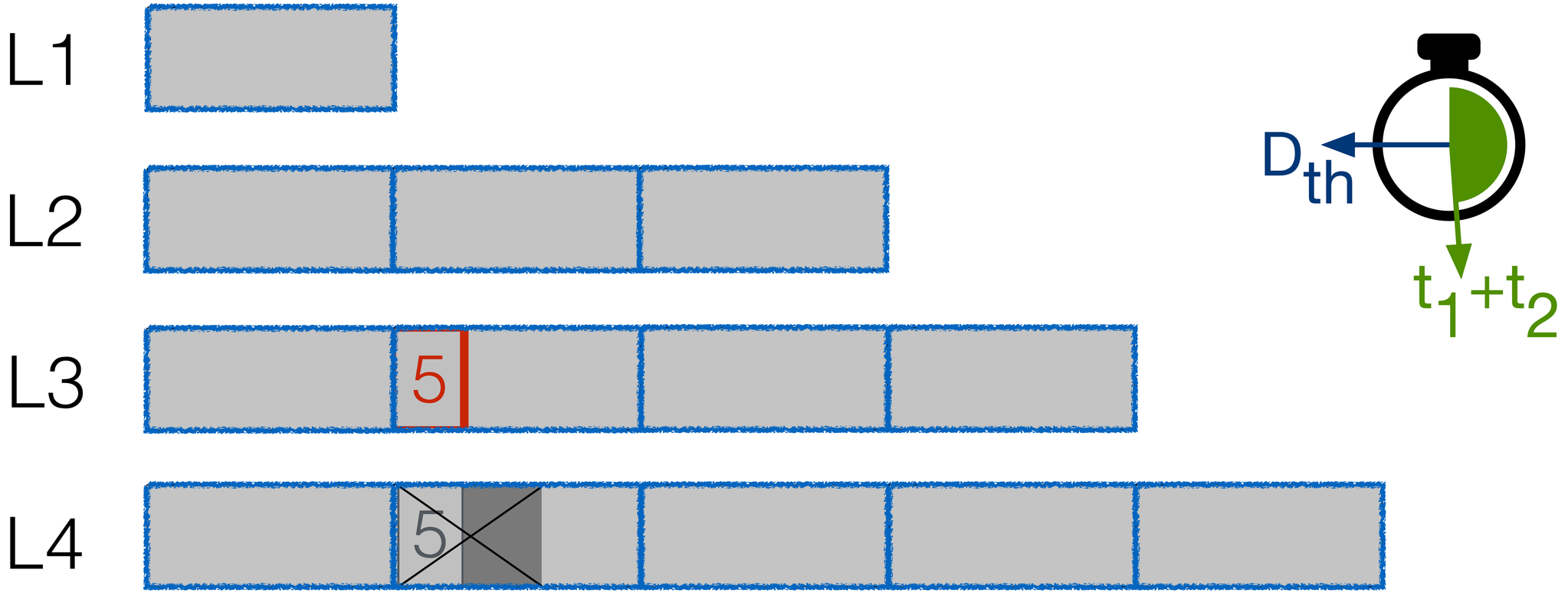
delete persistence latency

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



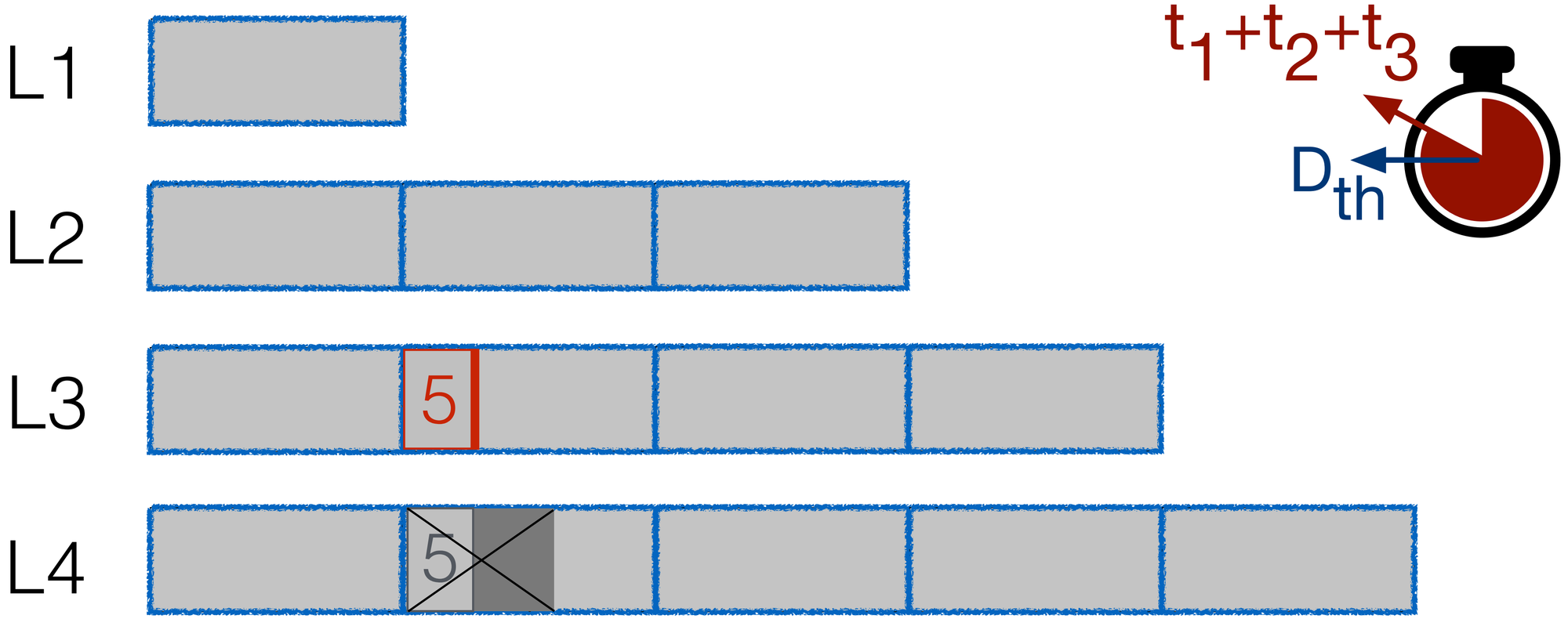
delete persistence latency

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



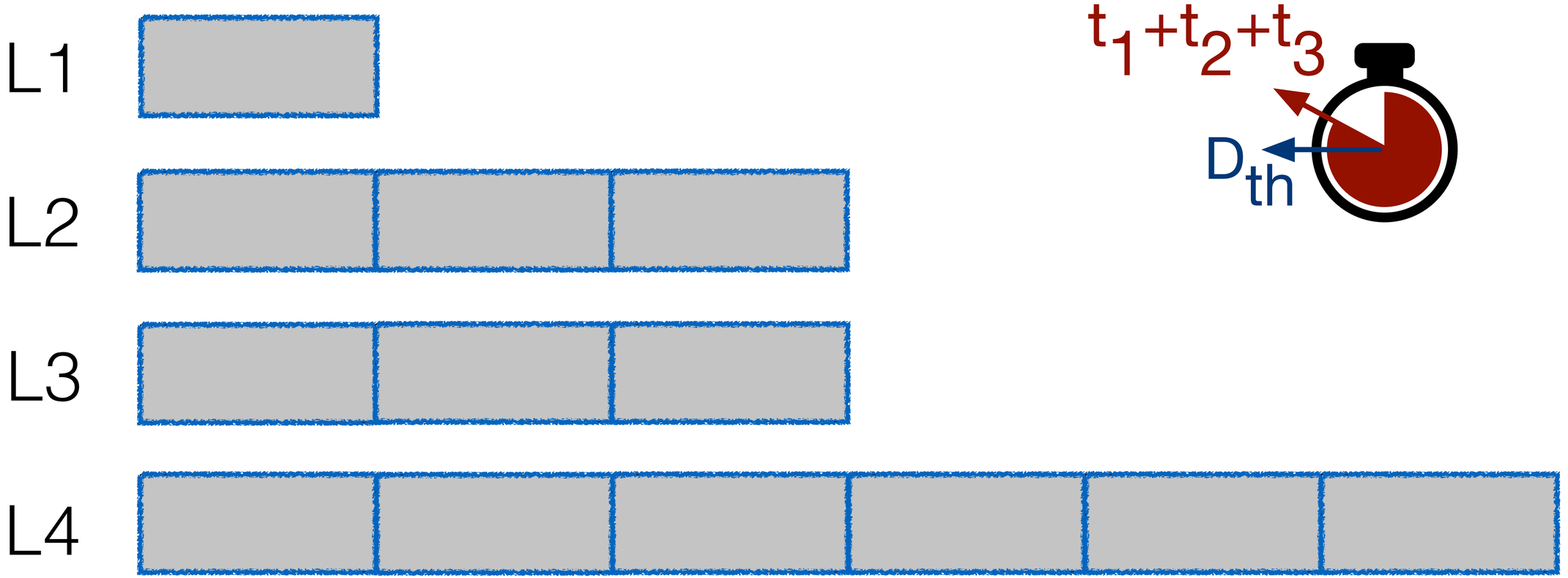
delete persistence latency

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



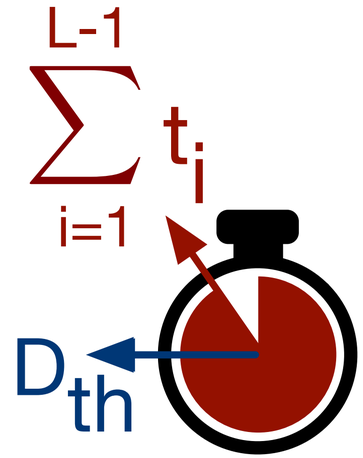
delete persistence latency

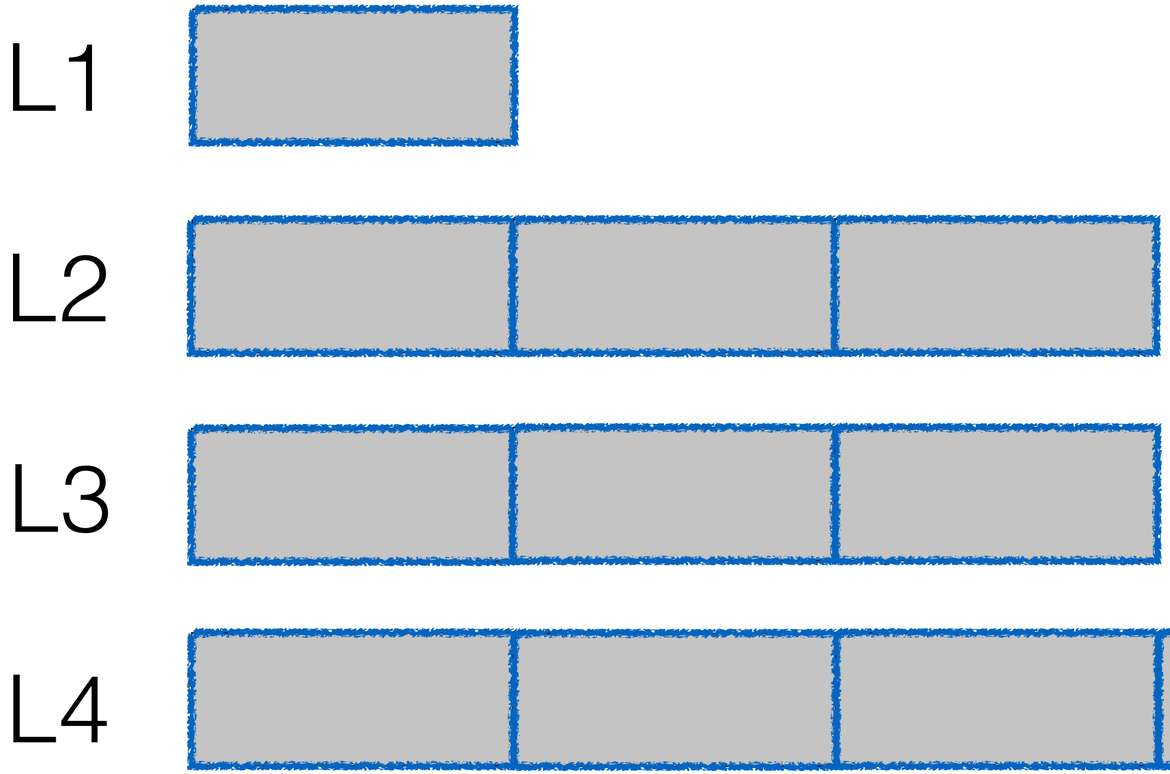
delete(5) within a threshold time: **D_{th}**



delete persistence latency

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





unbounded delete persistence latency

the solution

poor read perf.

write amplification

space amplification

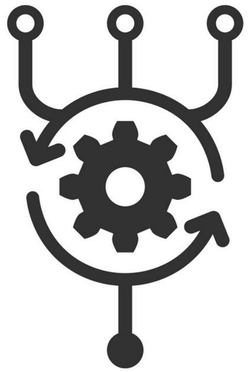
$\sum_{i=1}^{L-1} t_i$

D_{th}

unbounded delete persistence latency

FADE

FAst DElete



family of
compaction strategies

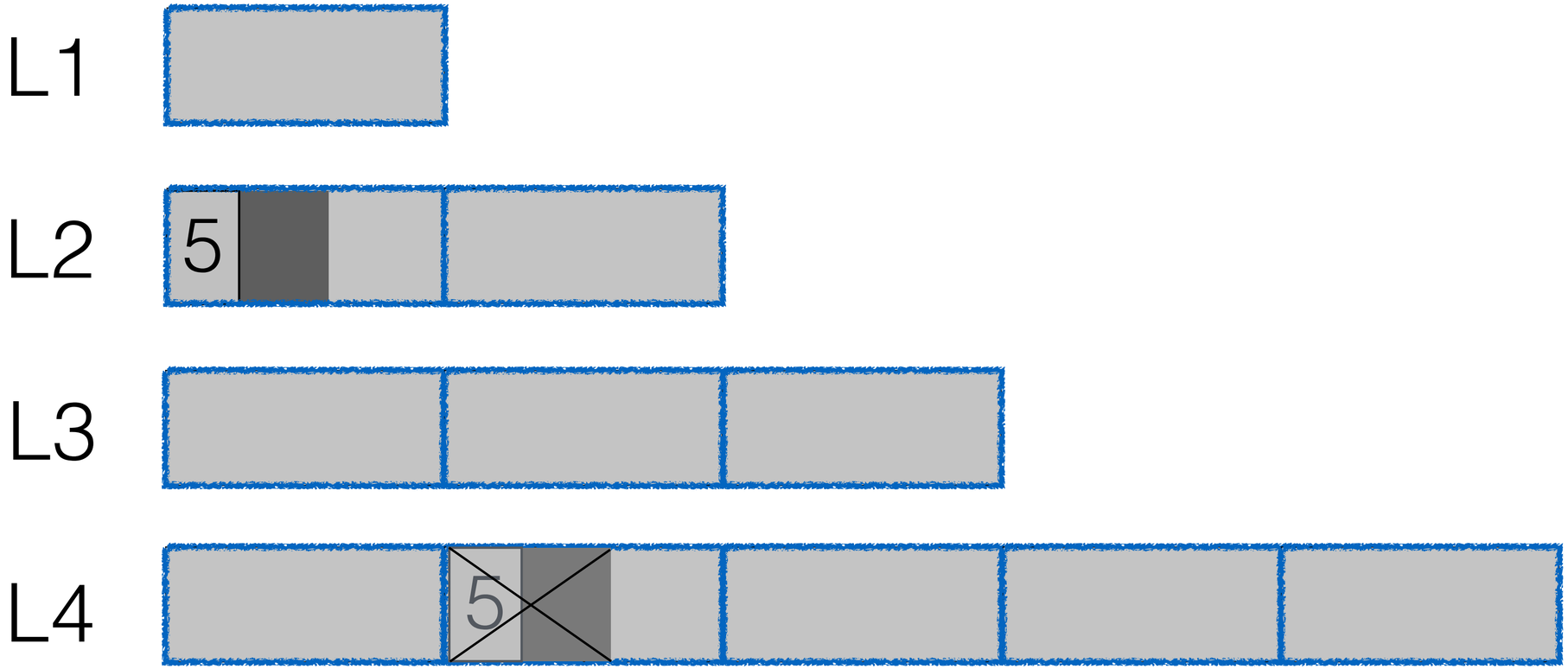
FAst DElete

compaction trigger

compaction file picking policy

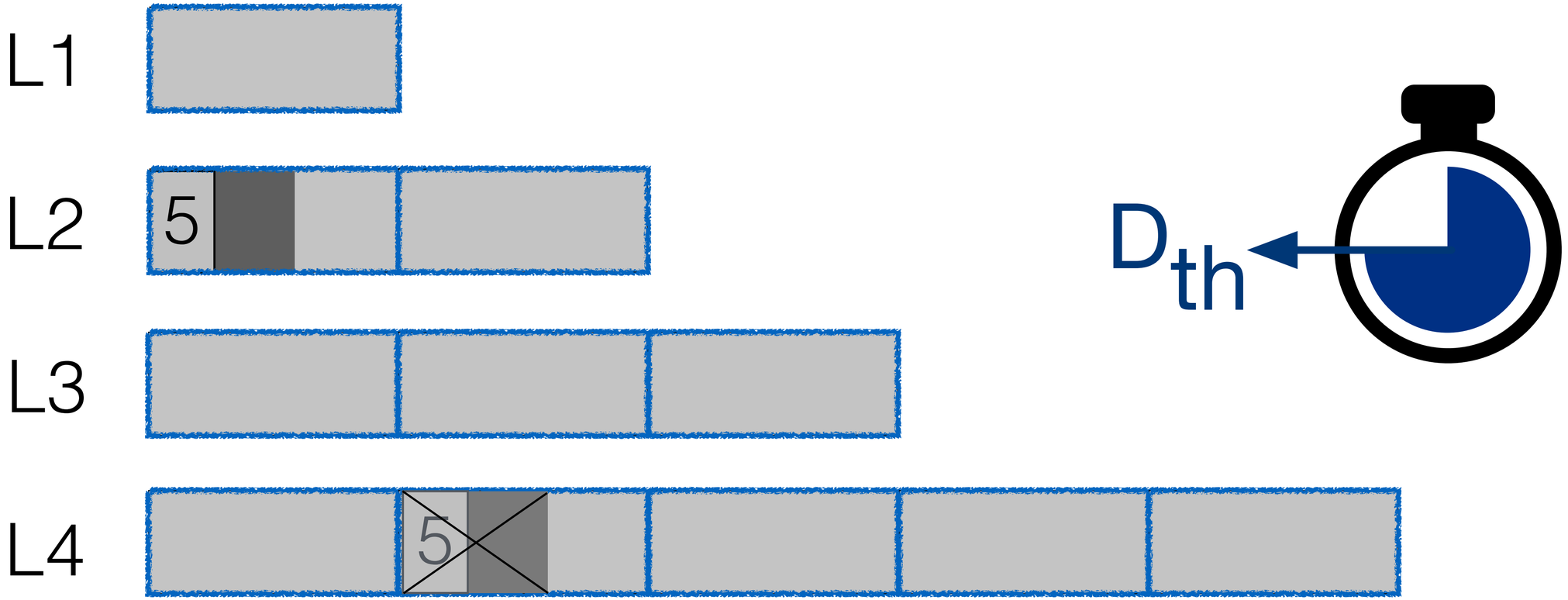
FAst DElete

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



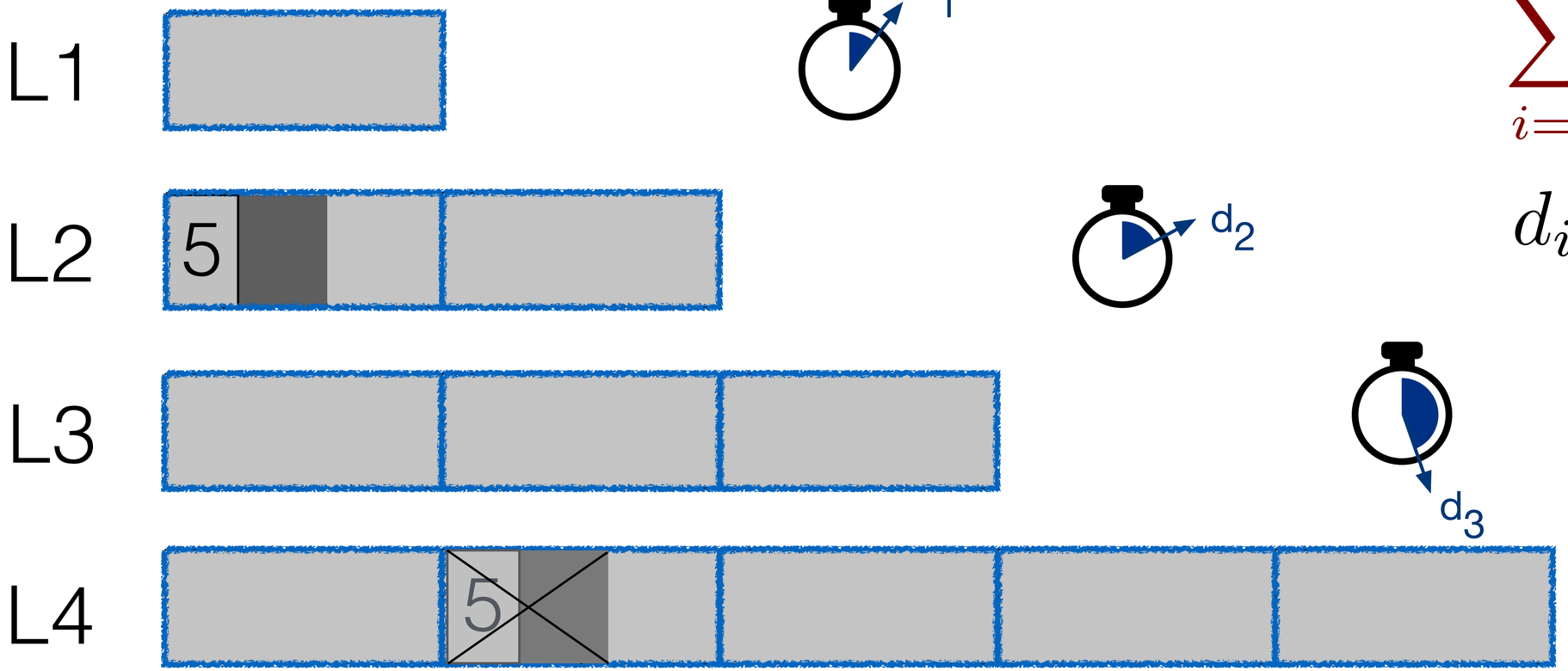
FAst DElete

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



FAst DElete

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

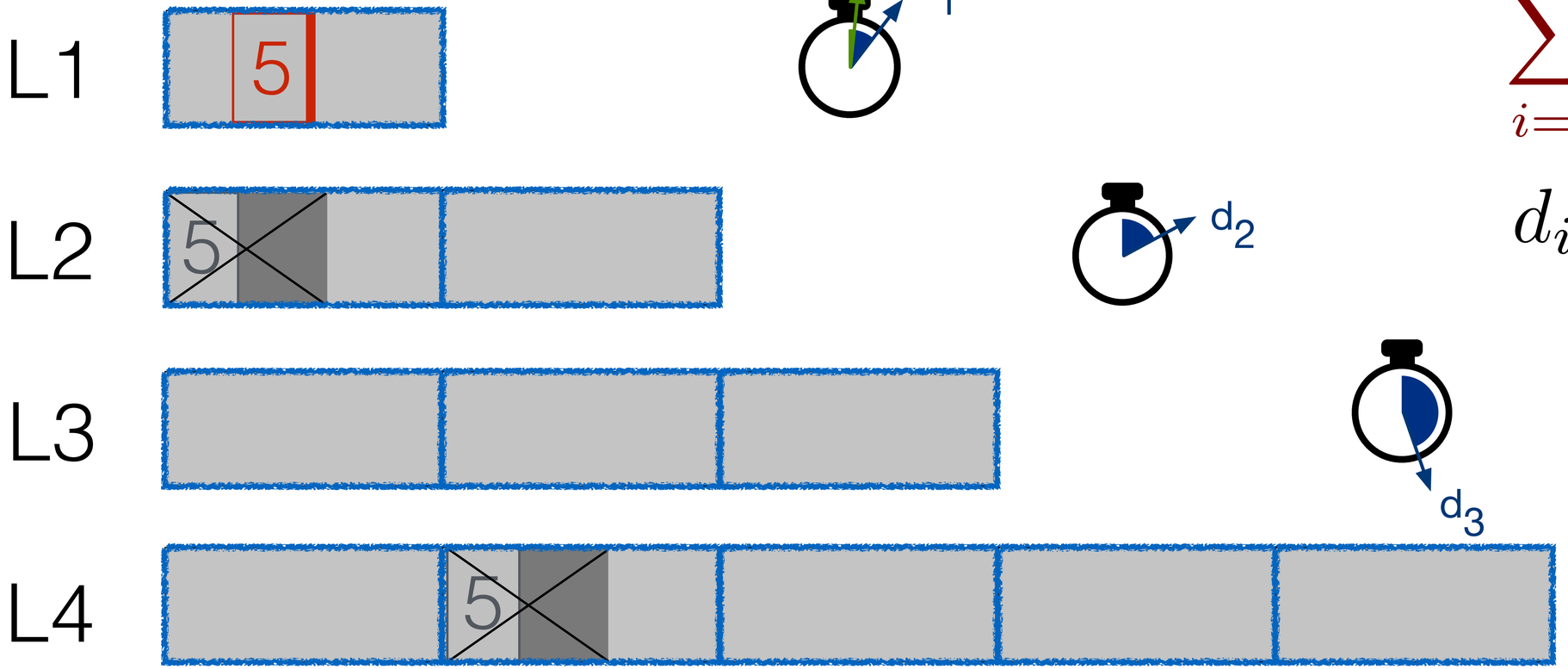


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

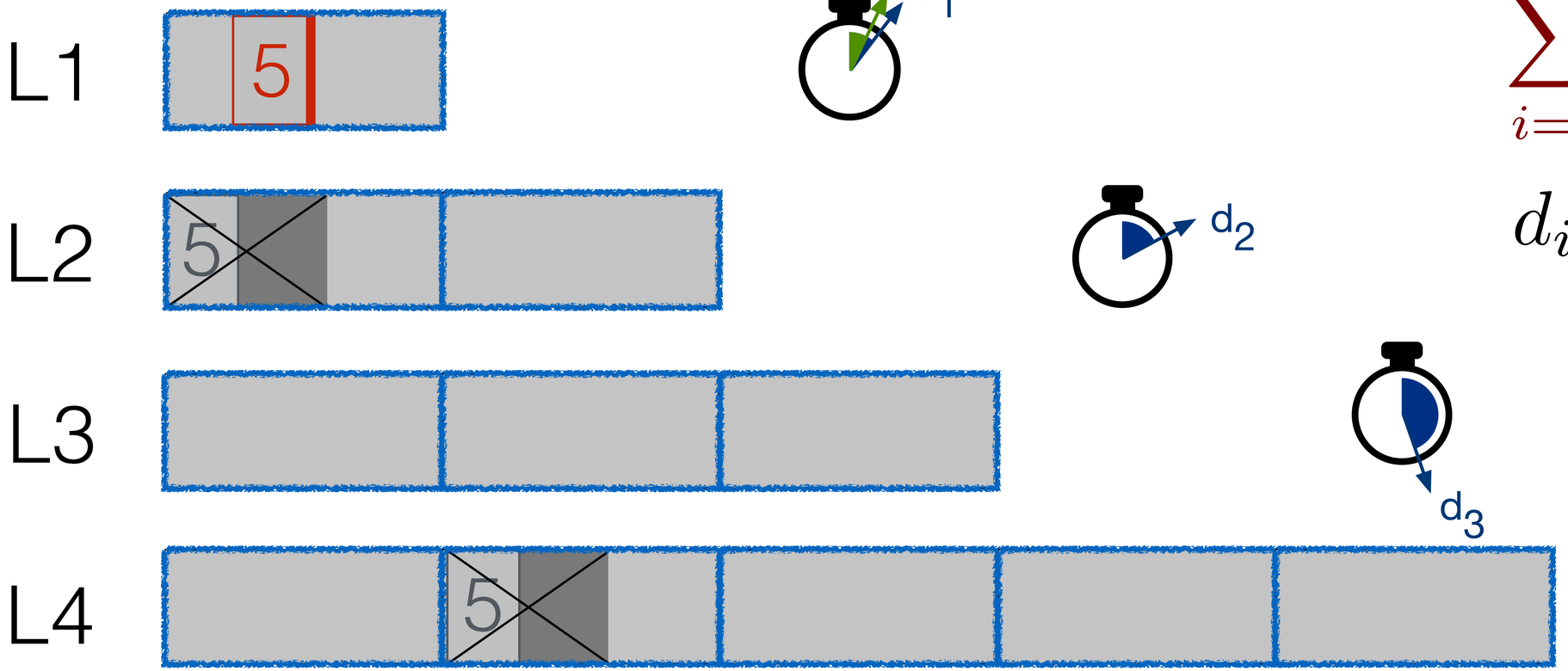


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

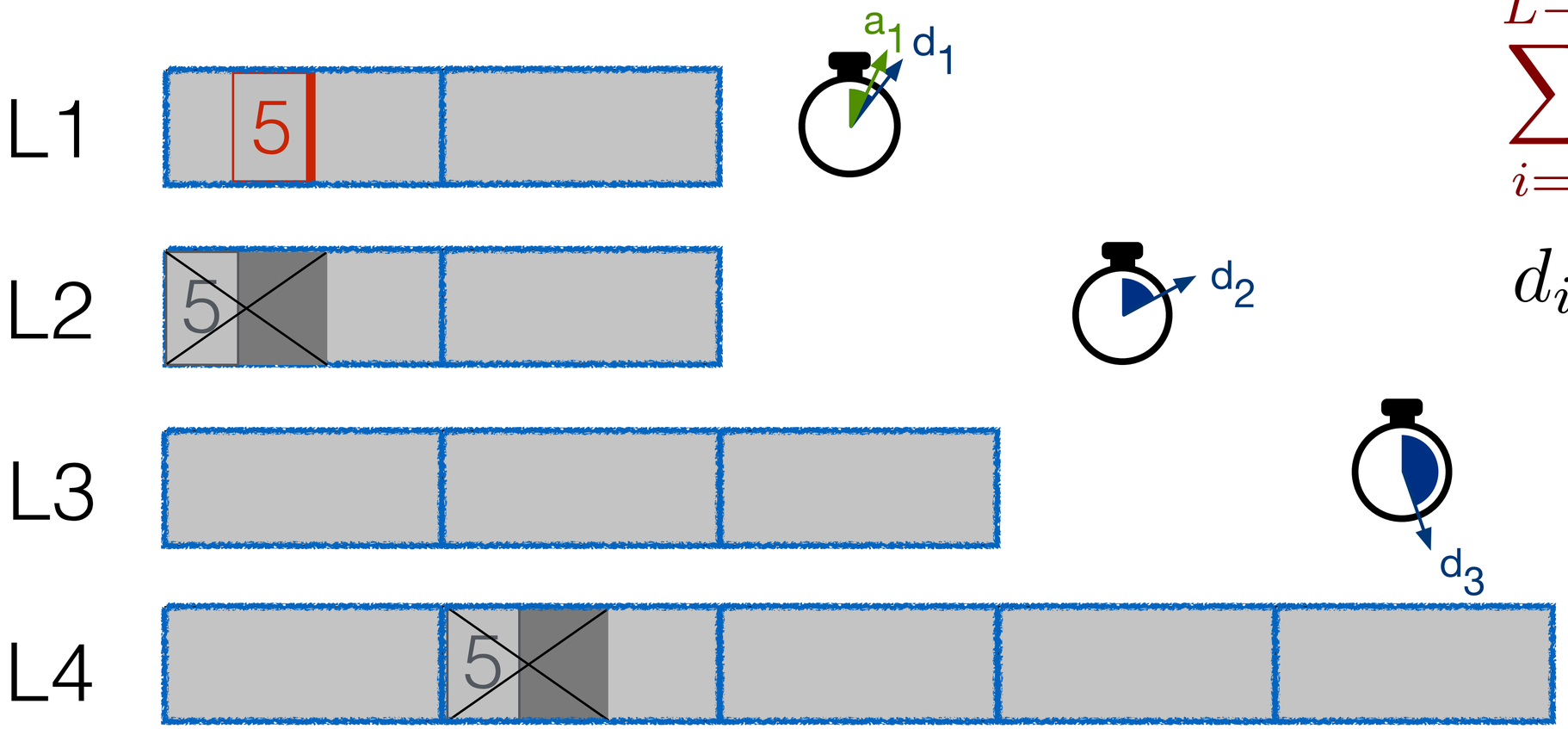


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

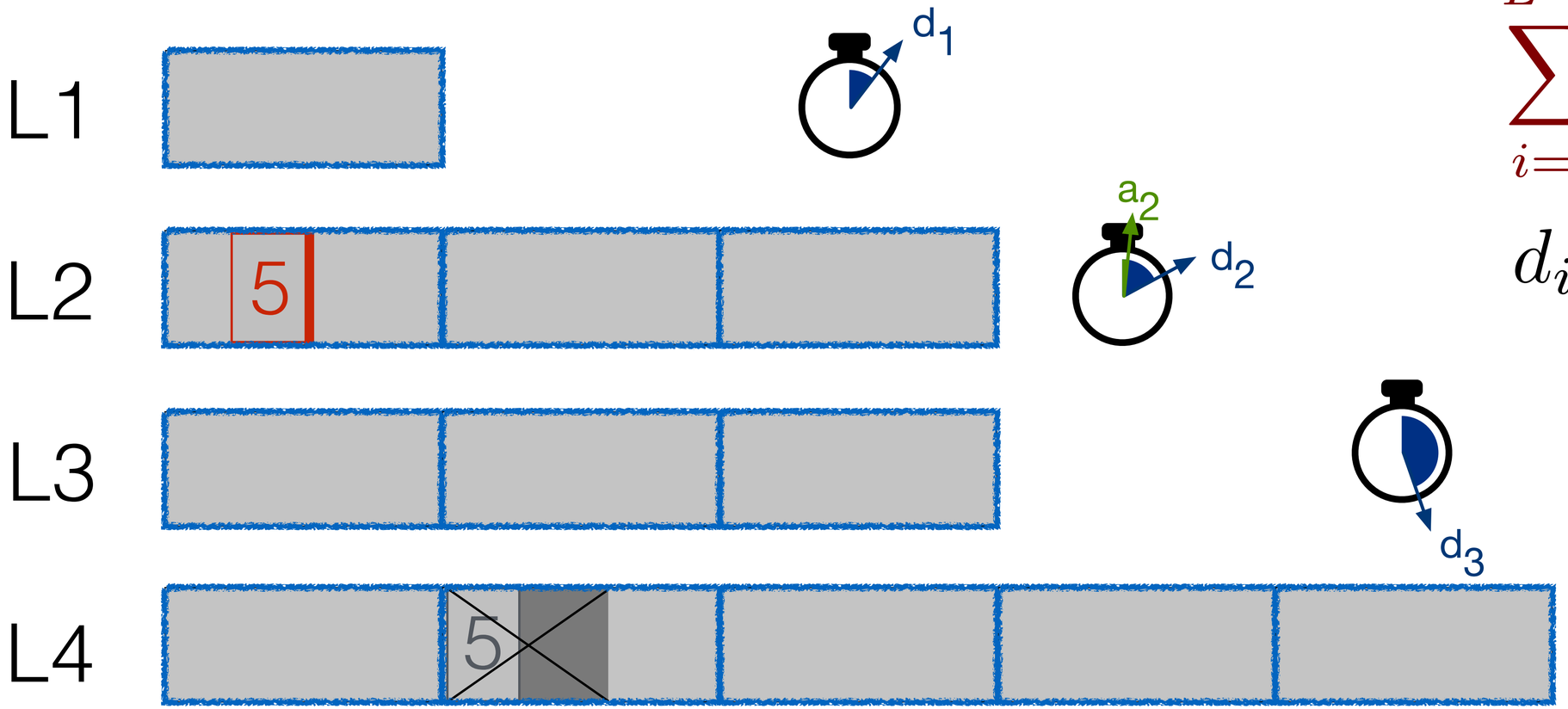


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

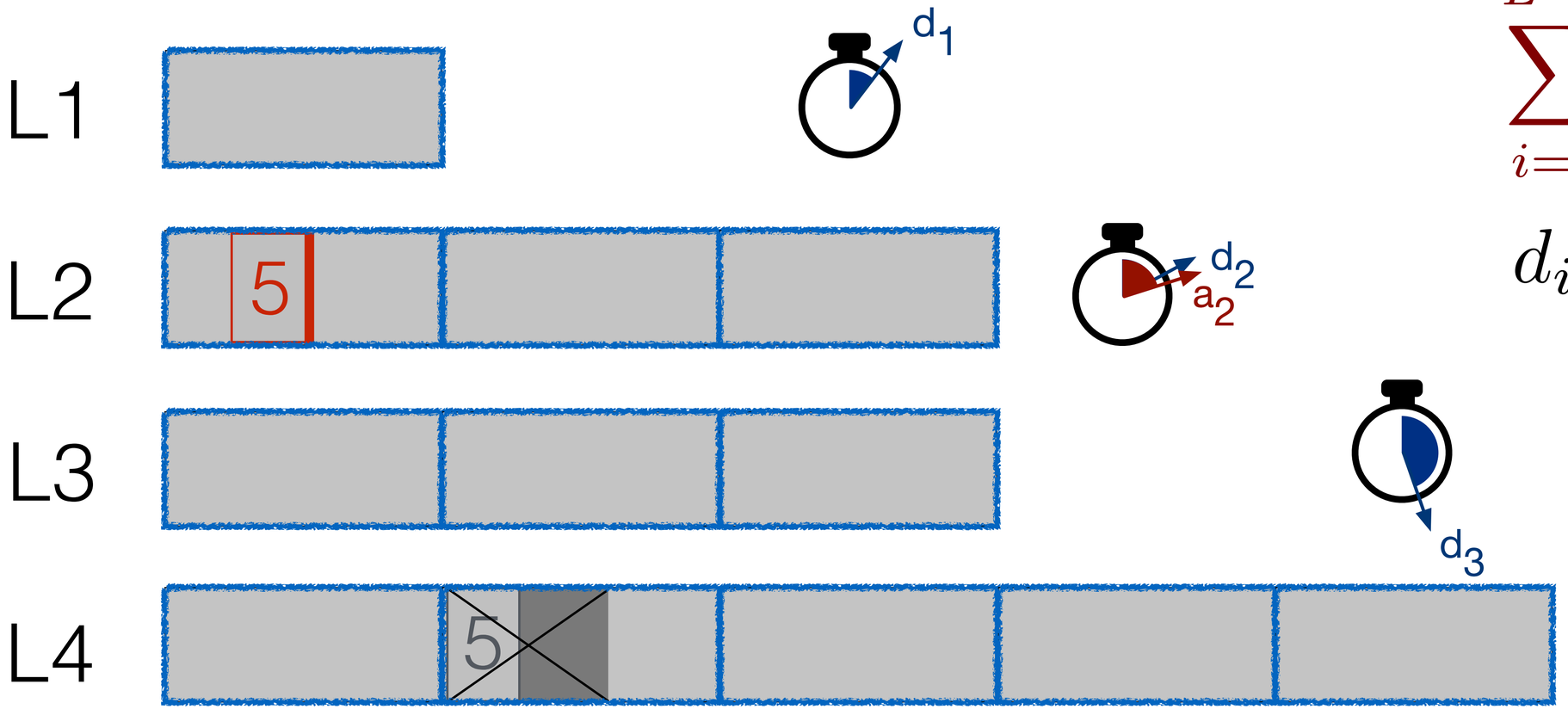


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

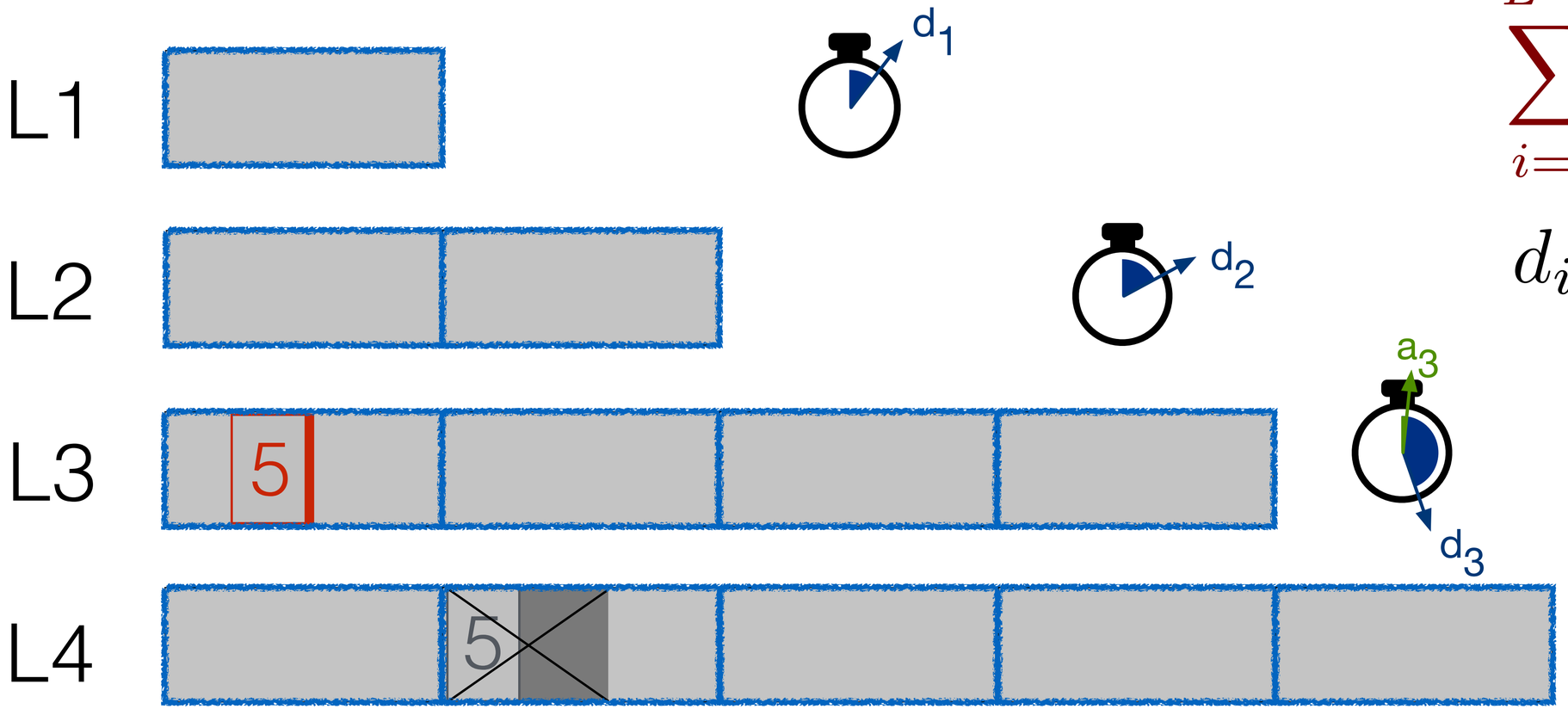


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

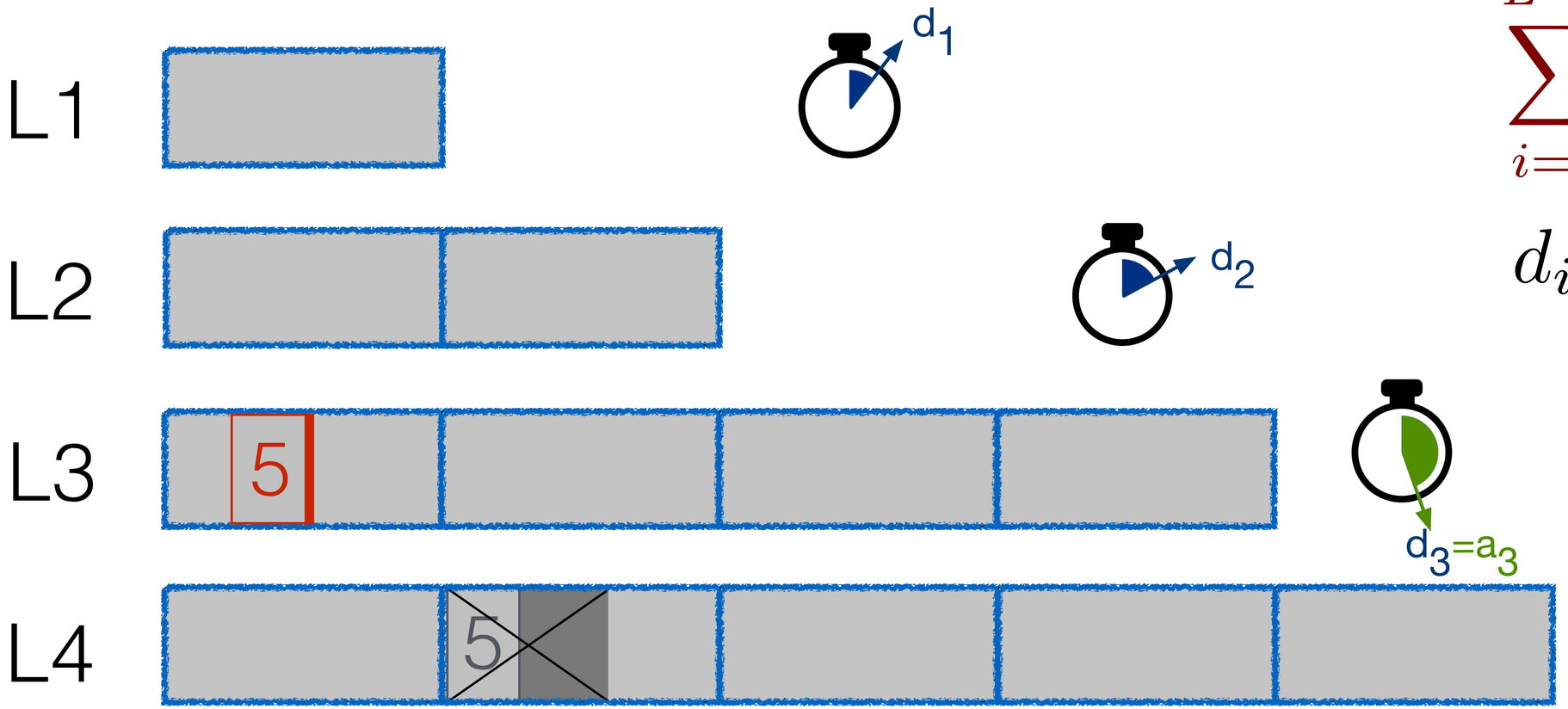


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

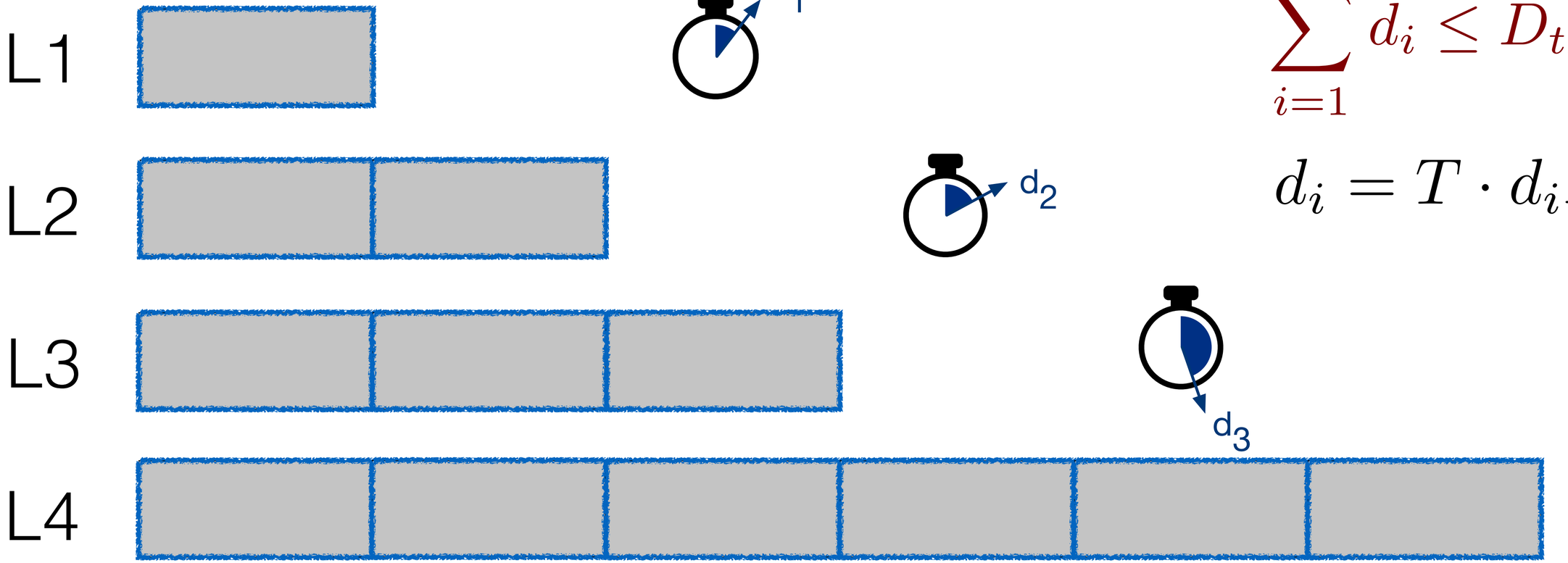


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

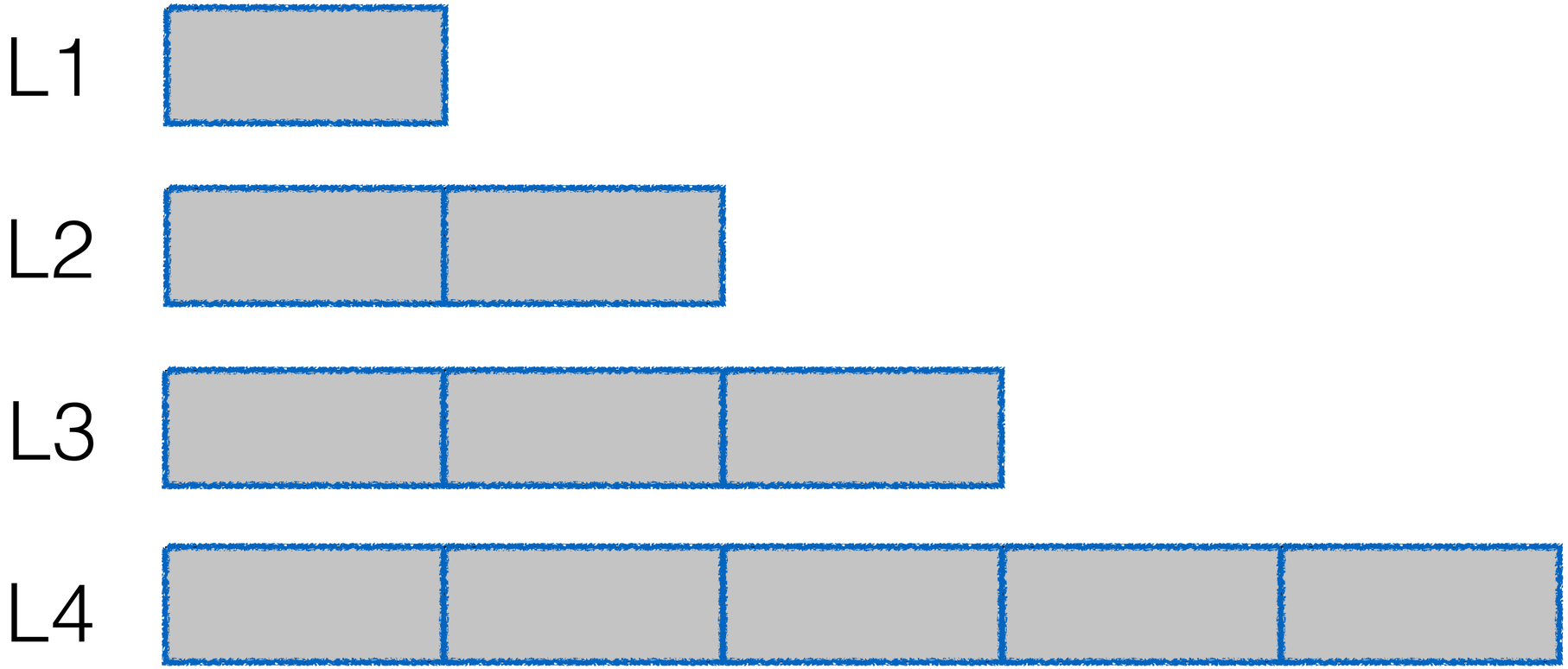


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

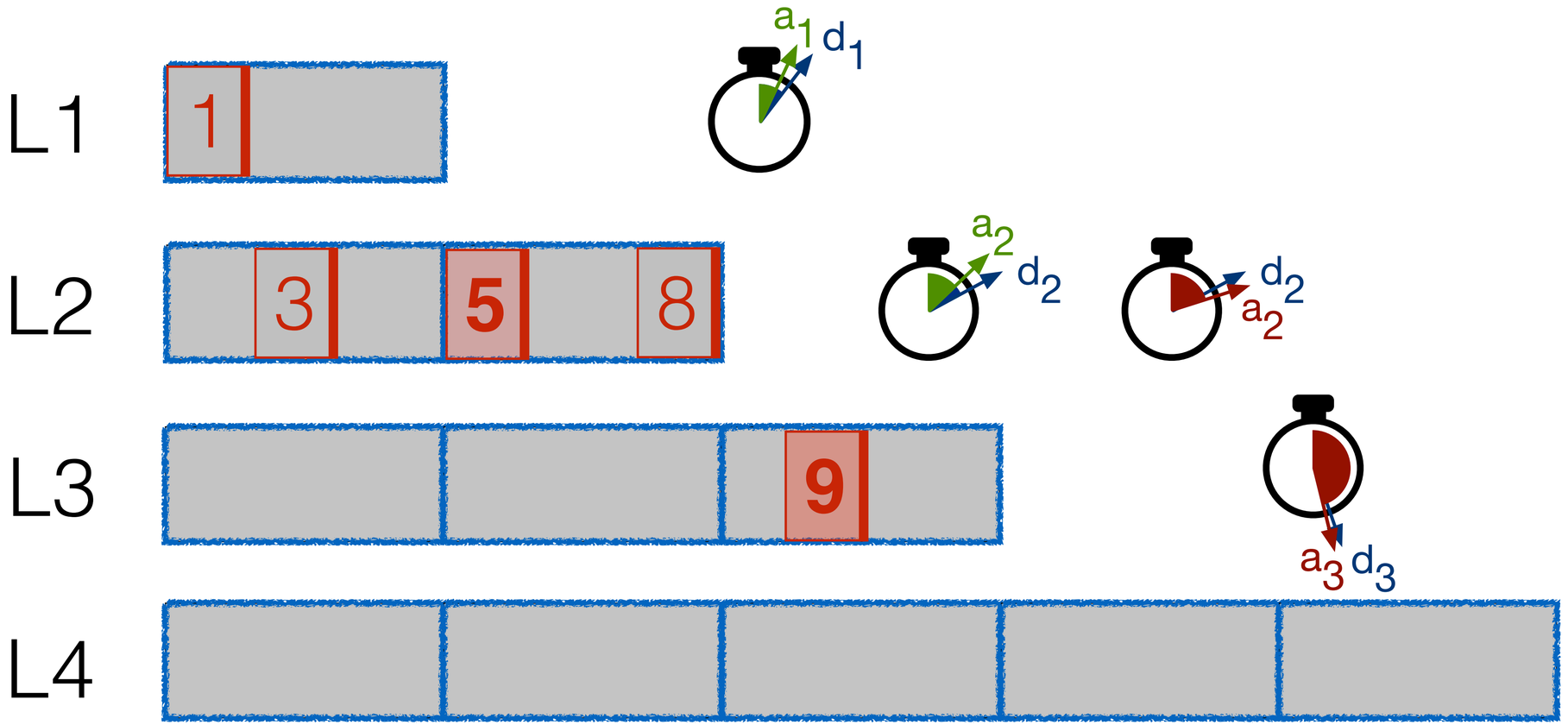
FAst DElete

breaking ties in practical workloads



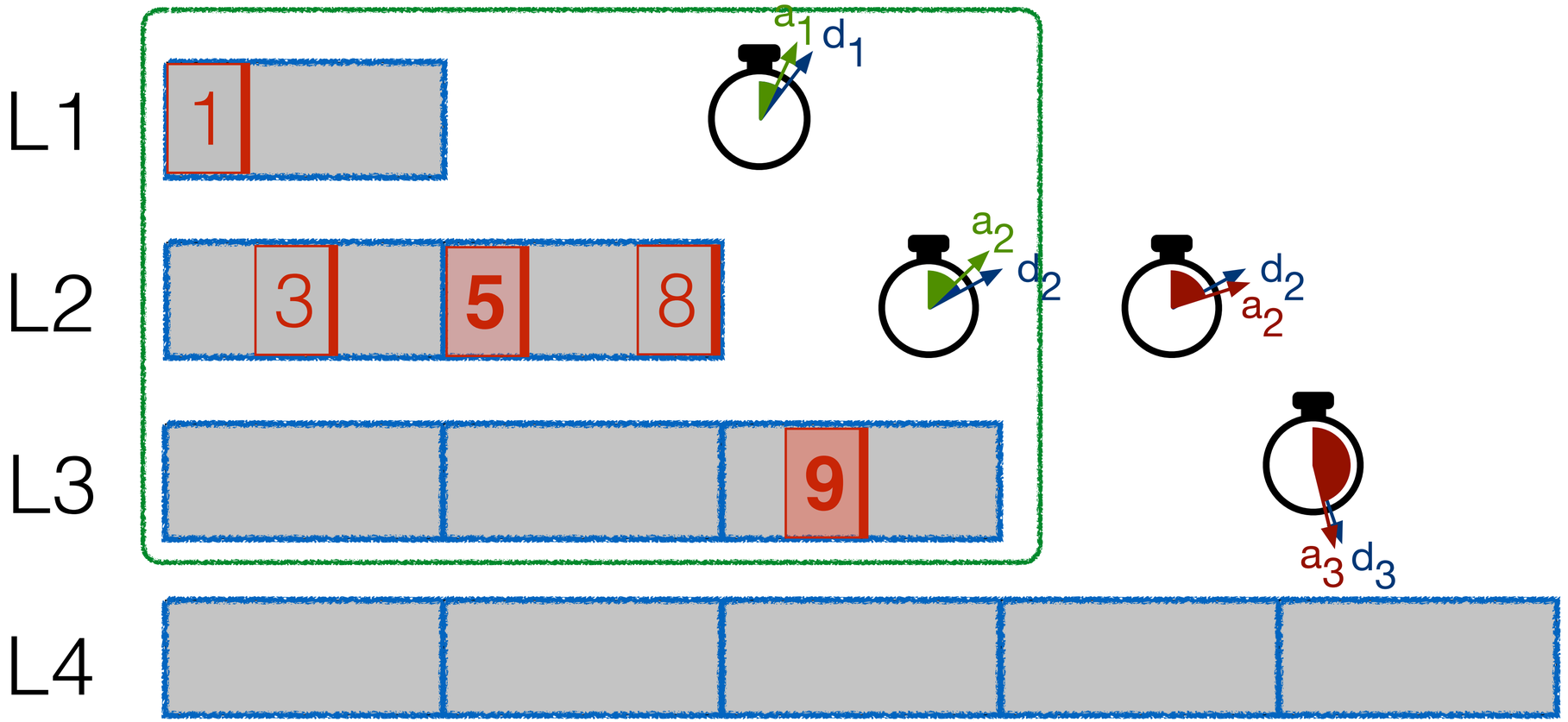
FAst DElete

breaking ties in practical workloads



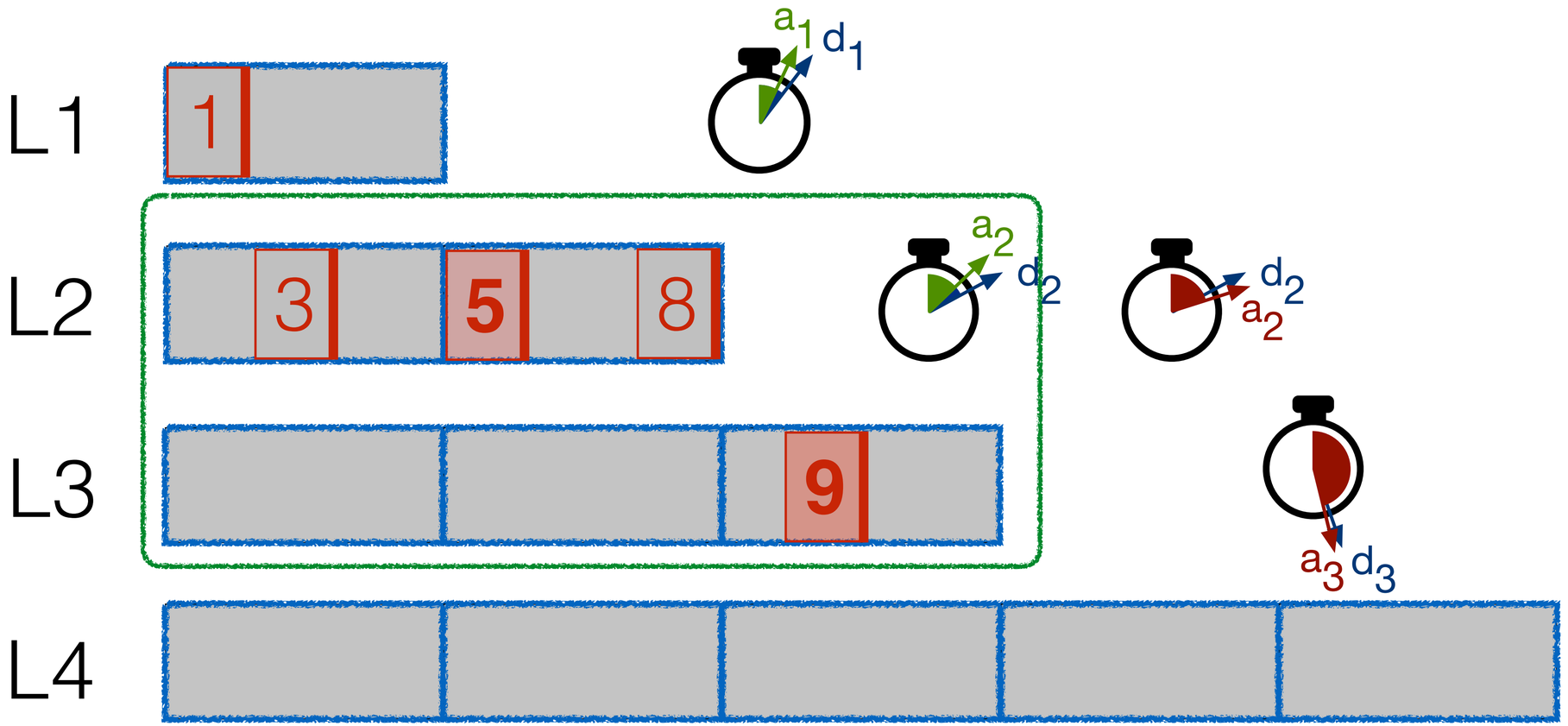
FAst DElete

breaking ties in practical workloads



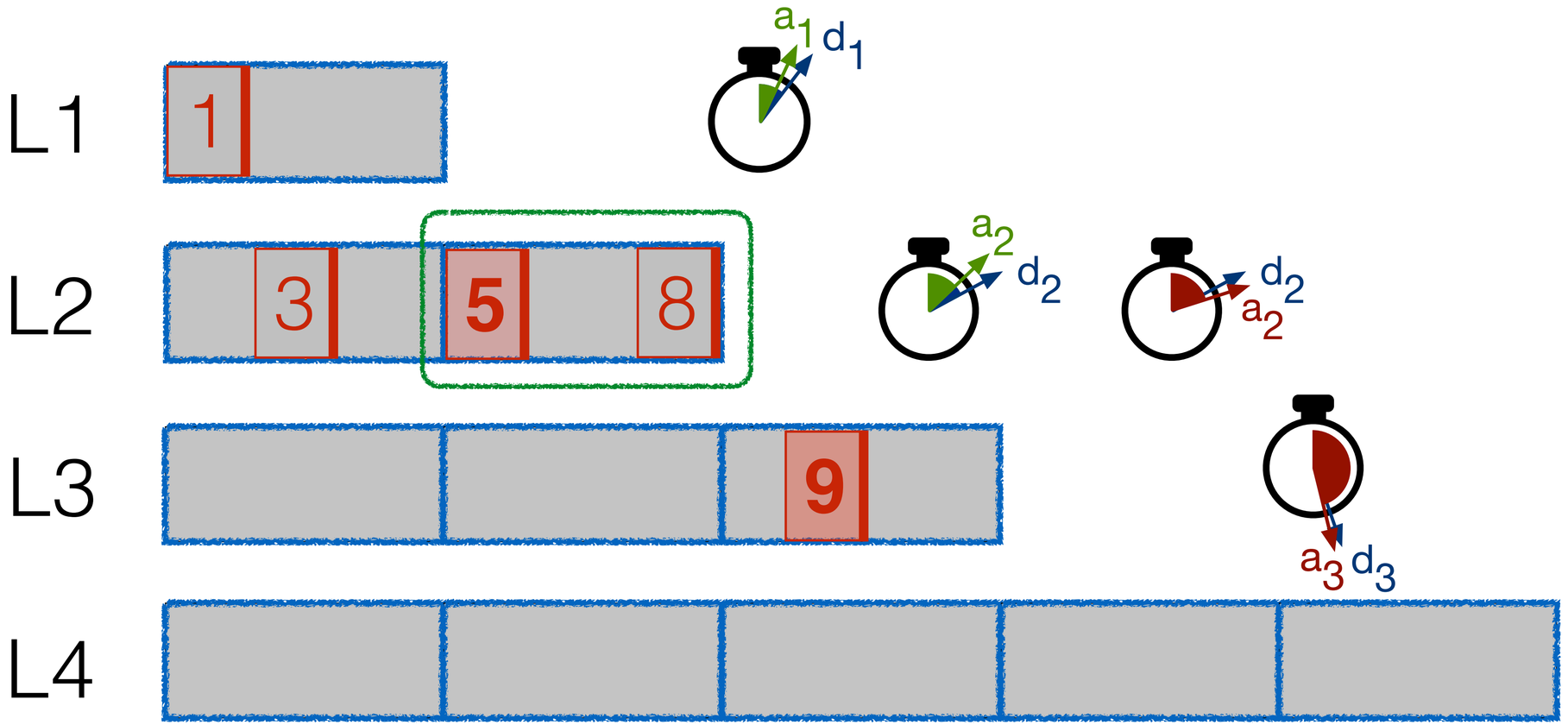
FAst DElete

breaking ties in practical workloads

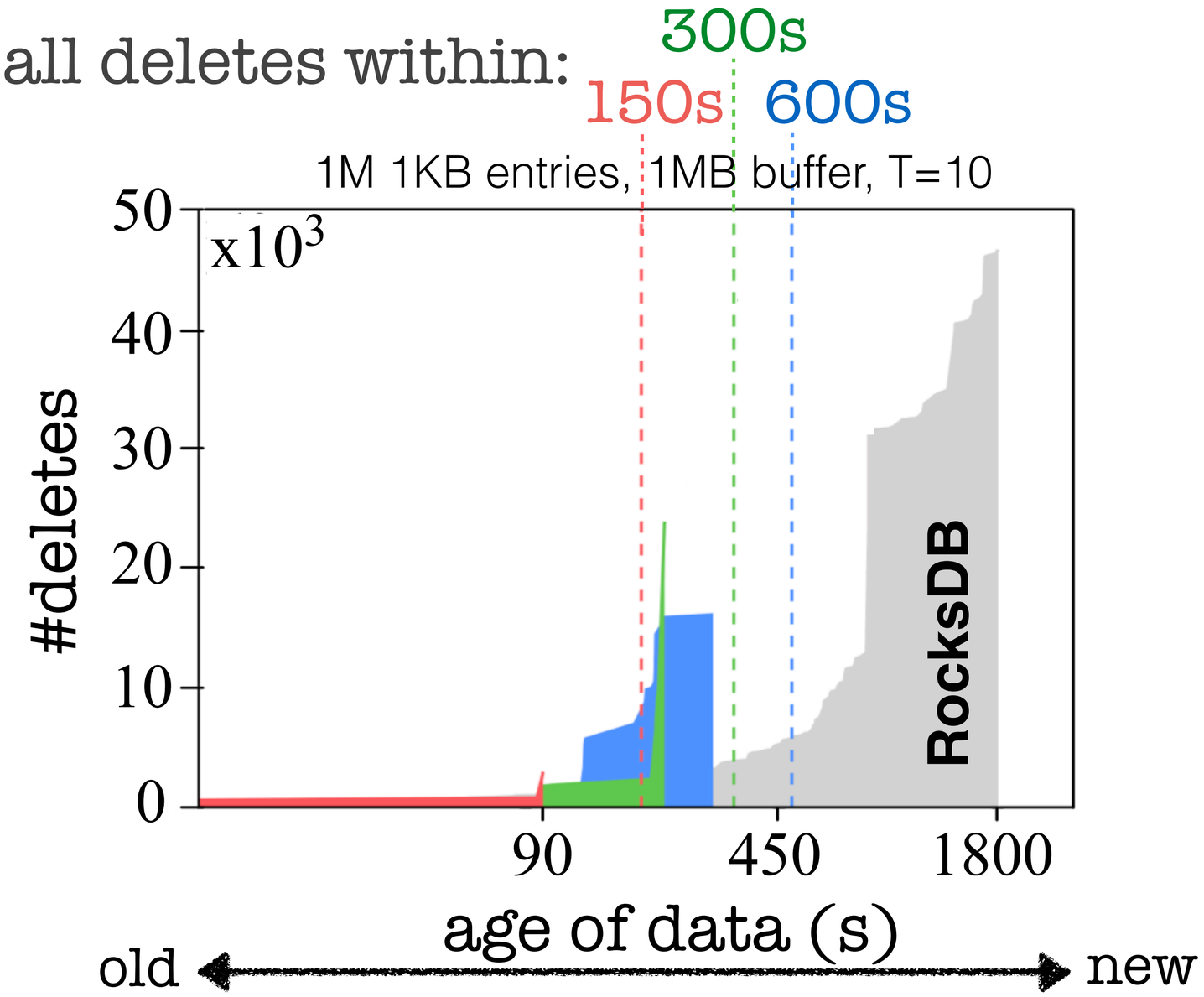


FAst DElete

breaking ties in practical workloads



persist all deletes within:



persists deletes timely

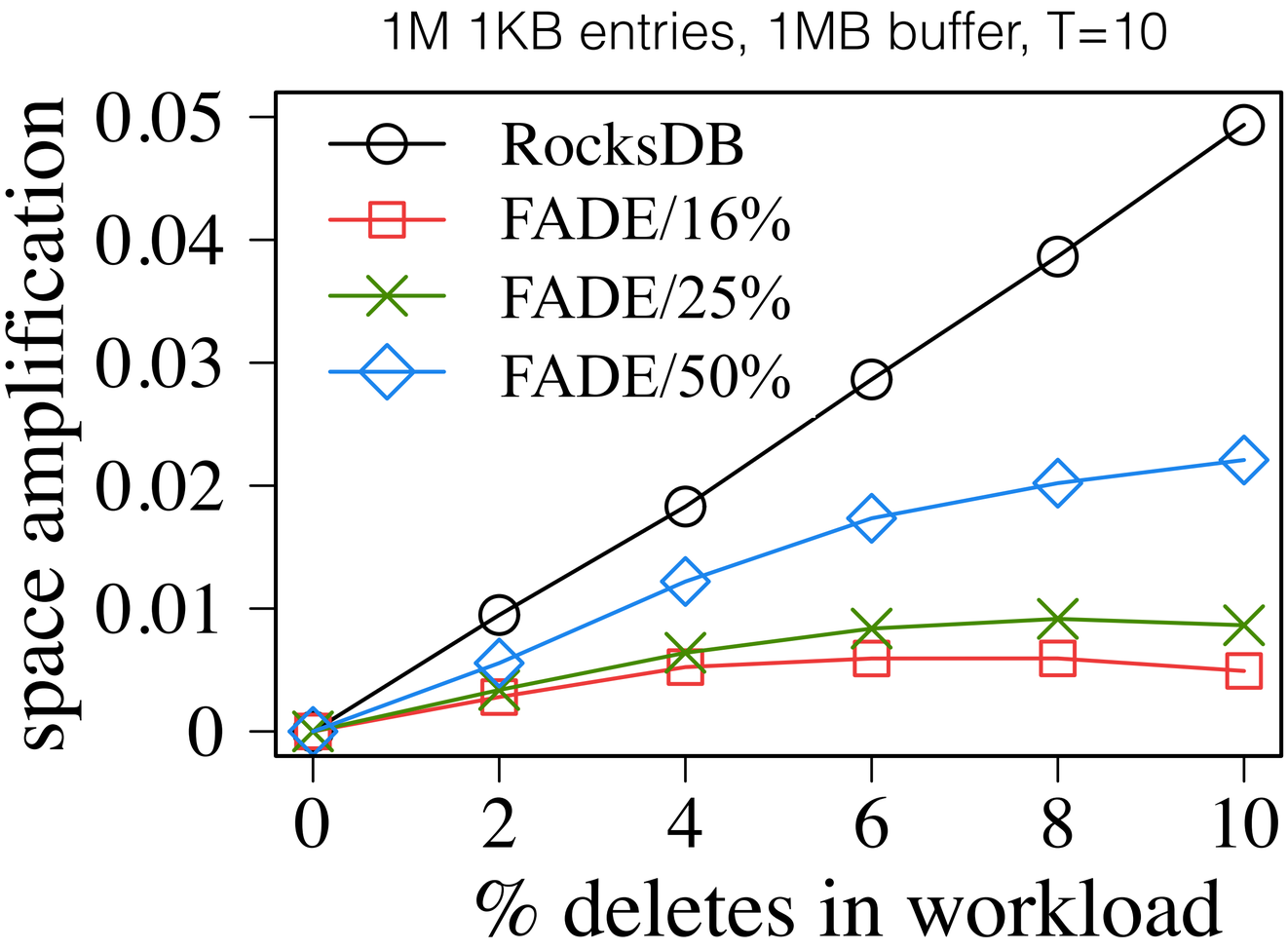
within threshold

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



improved read performance

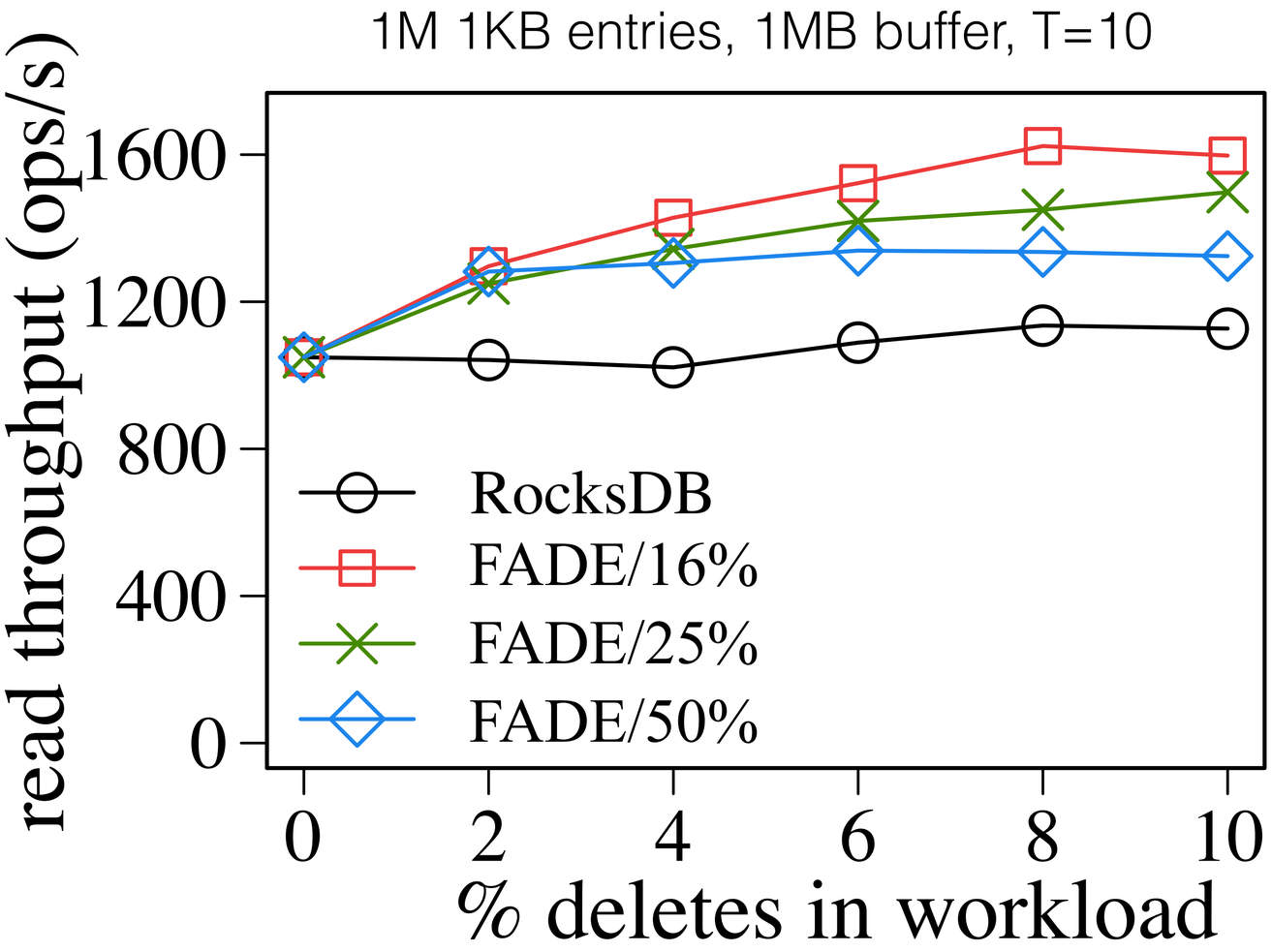
1.2x - 1.4x

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



higher write amplification

4% - 25%

improved read performance

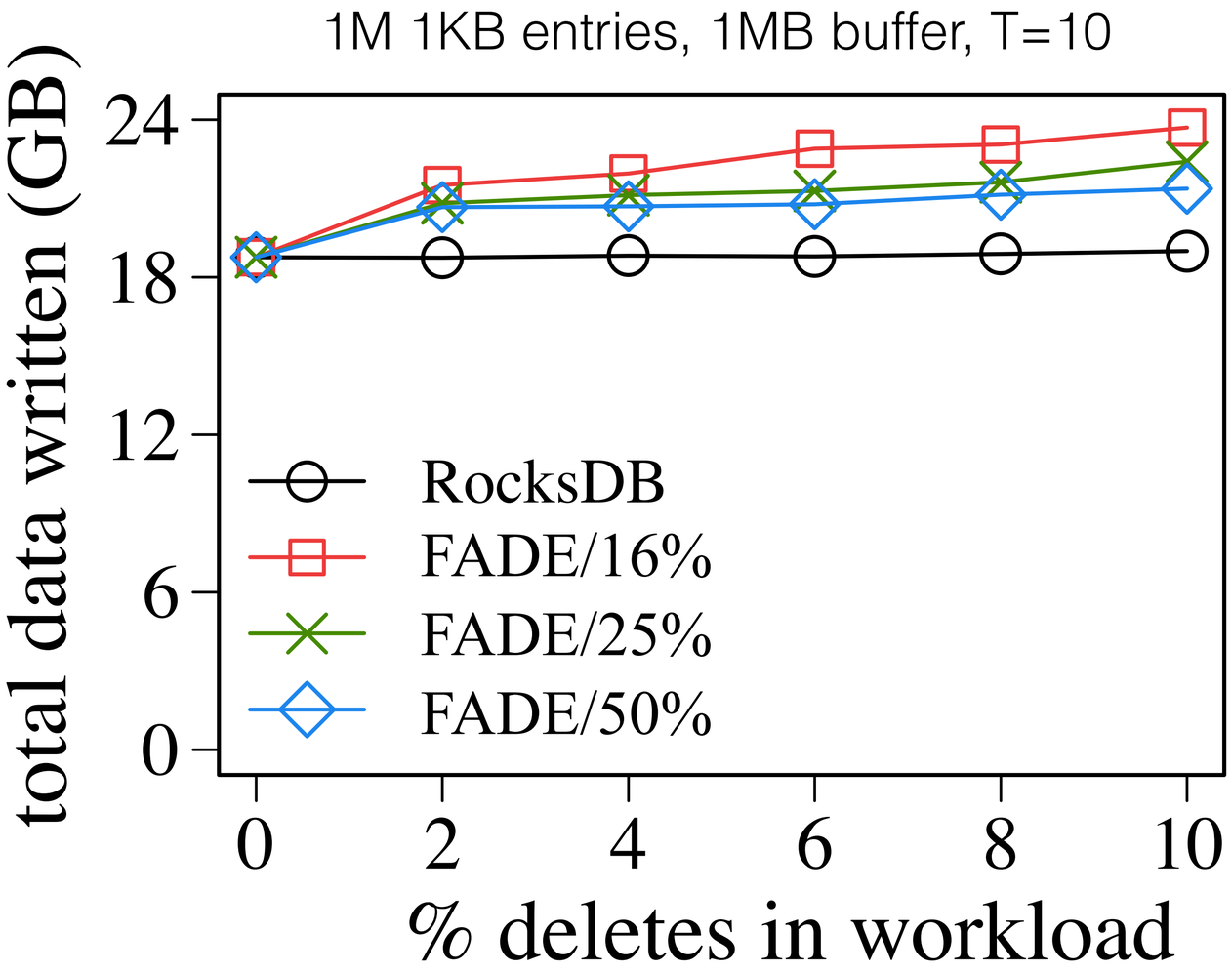
1.2x - 1.4x

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



higher write amplification

4% - 25%

improved read performance

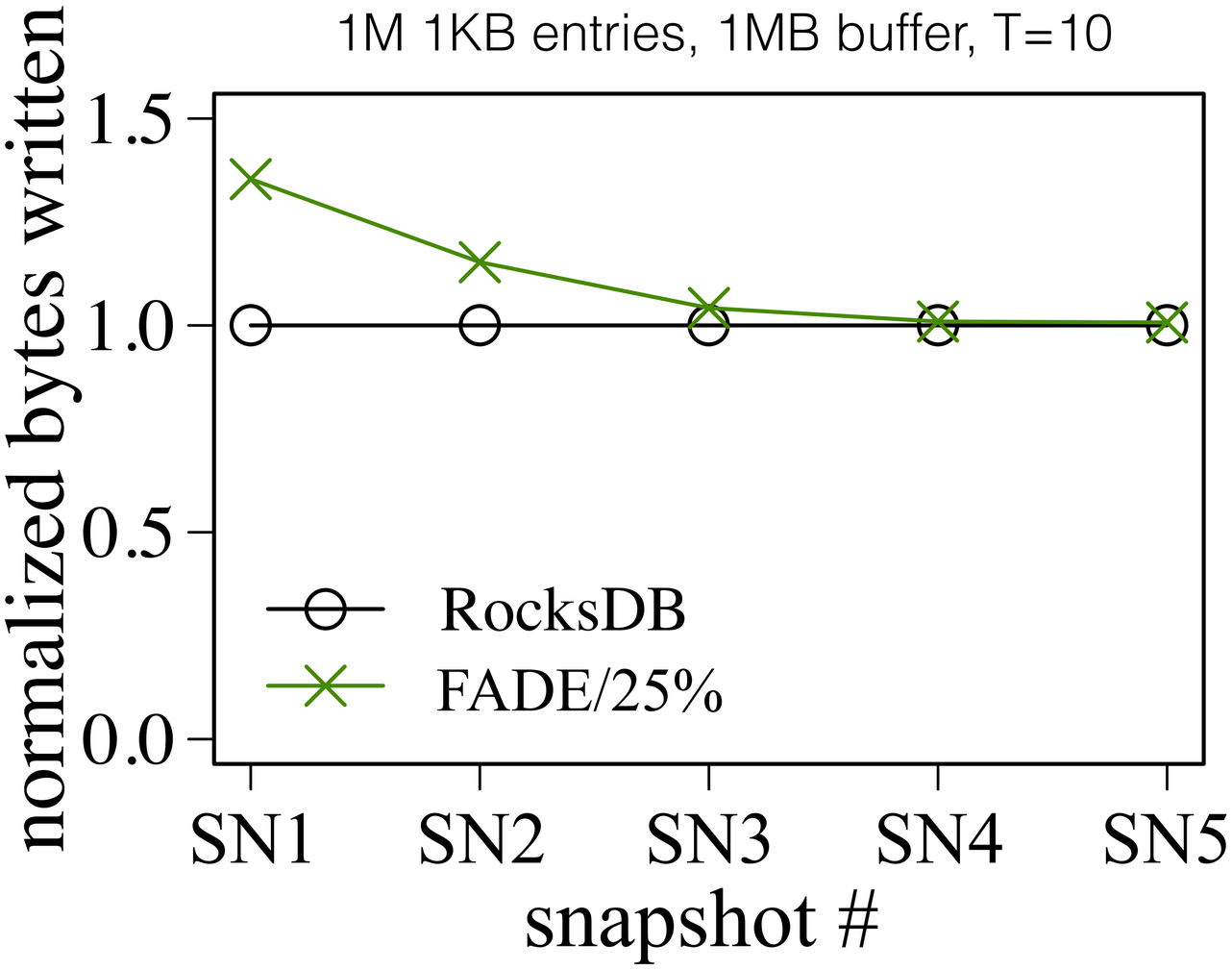
1.2x - 1.4x

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



higher write amplification

0.7%

improved read performance

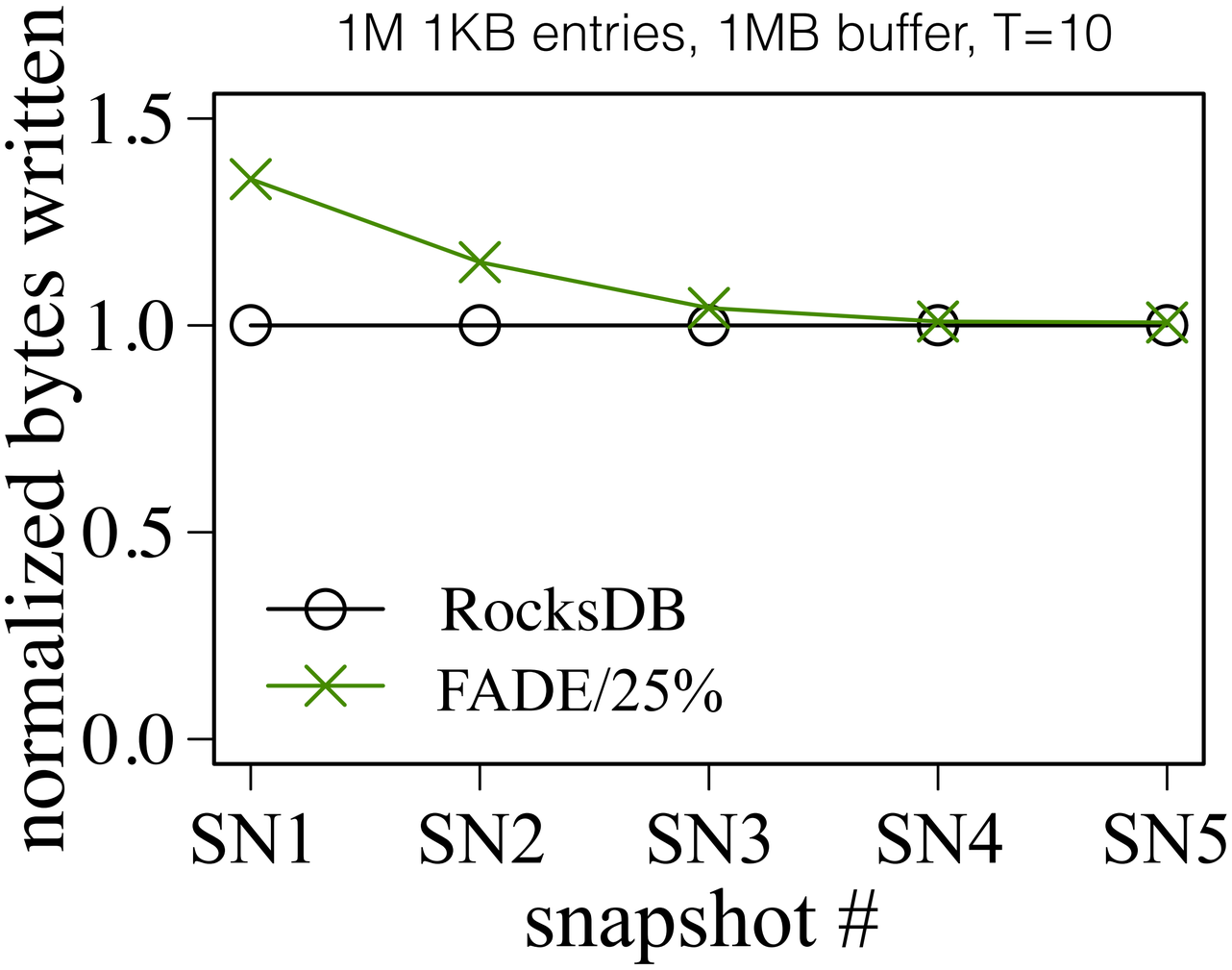
1.2x - 1.4x

reduced space amplification

2.1x - 9.8x

persists deletes timely

within threshold



the problems

poor read perf.

write amplification

space amplification

$$\sum_{i=1}^{L-1} t_i$$

D_{th}

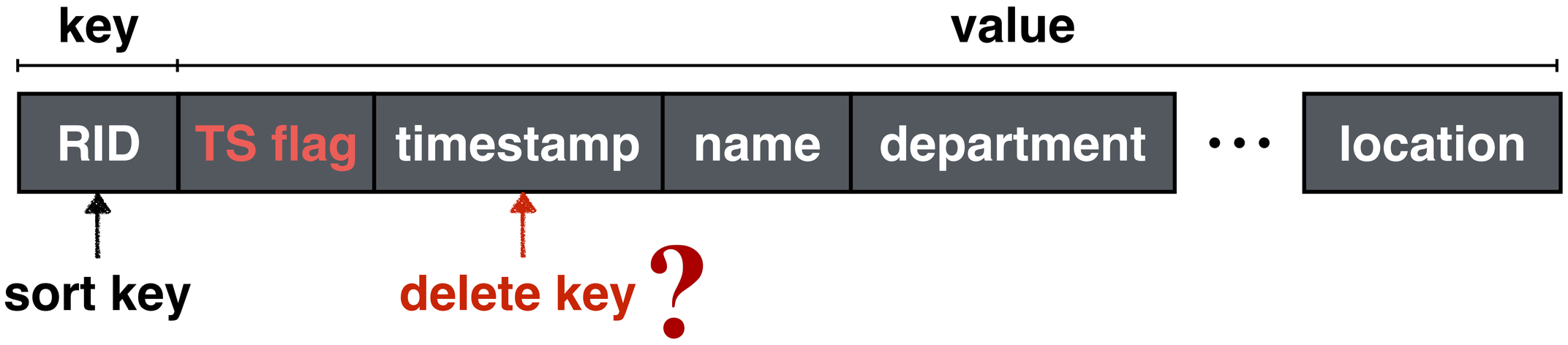
unbounded delete persistence latency

FADE

deletes on a secondary attribute

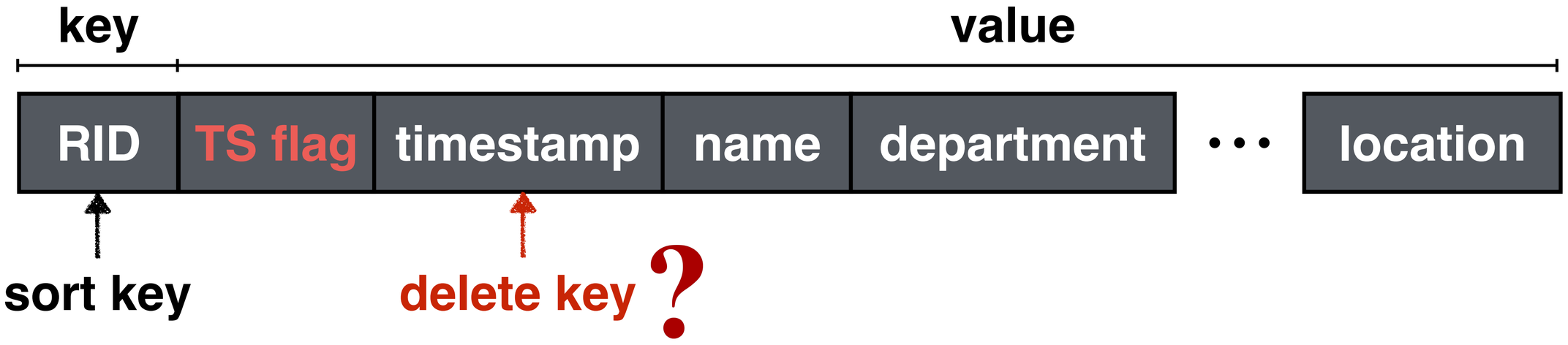
deletes on a secondary attribute

delete all entries older than: **D days**



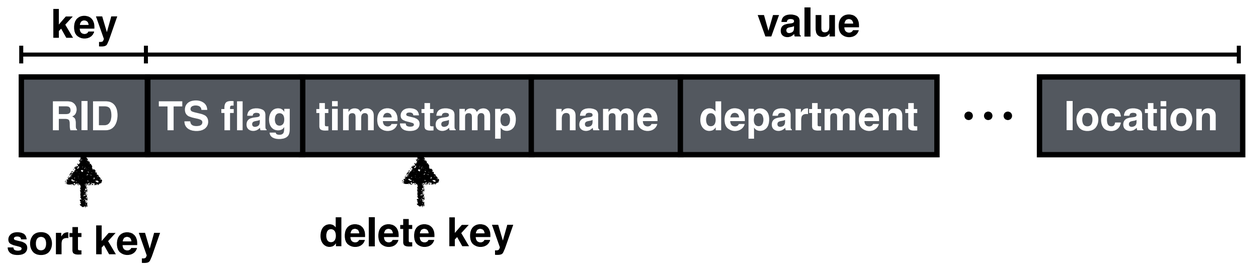
deletes on a secondary attribute

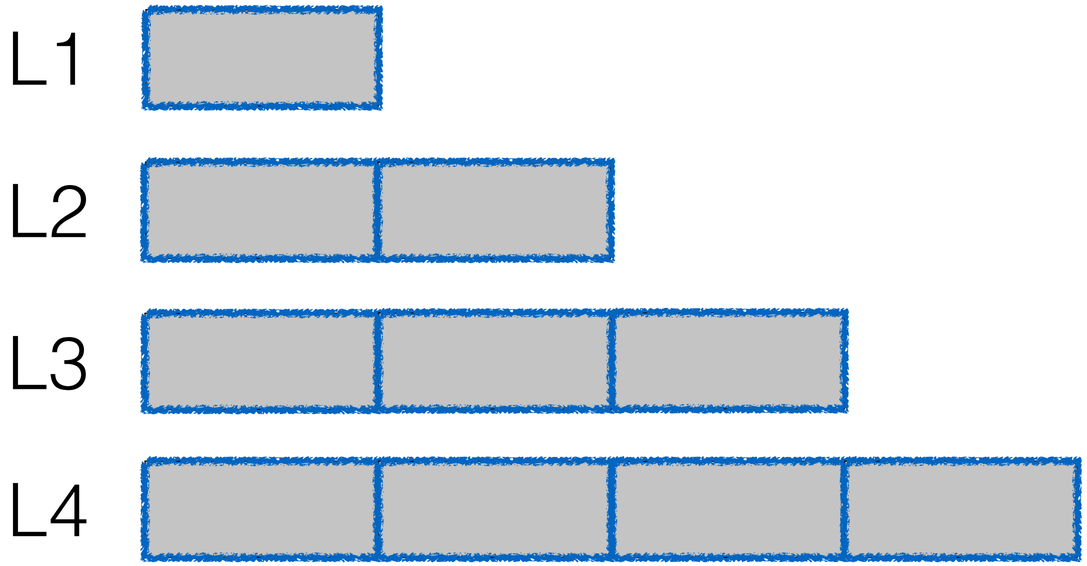
delete all entries older than: **D days**



deletes on a secondary attribute

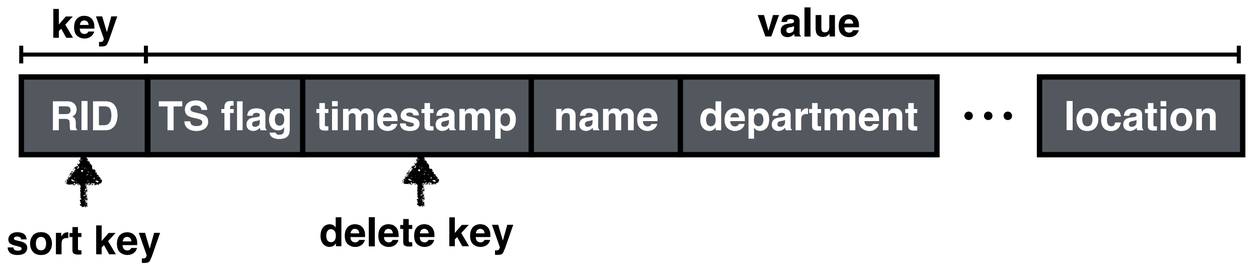
delete all entries older than: **D days**

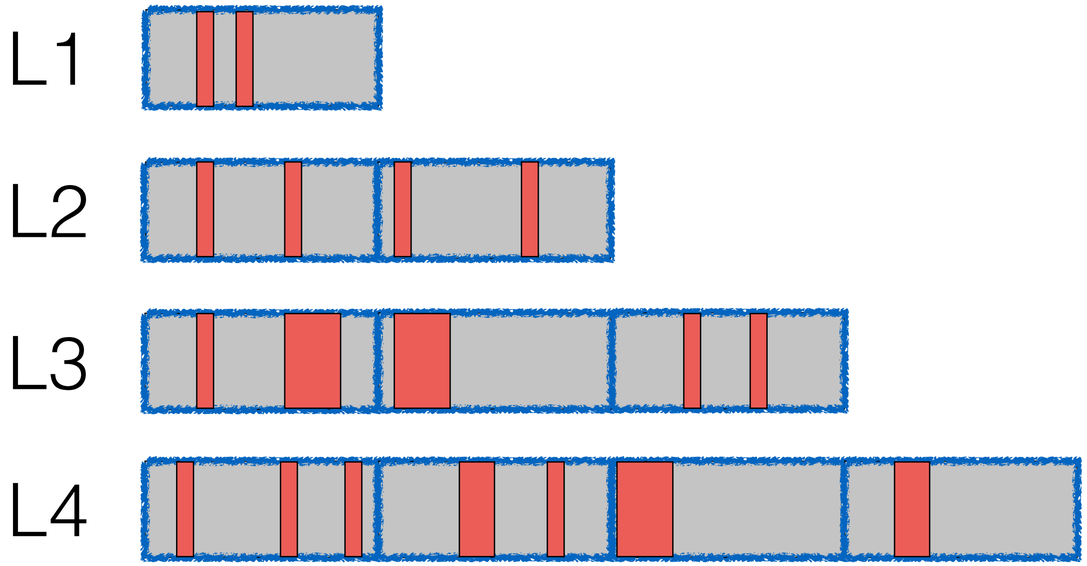




deletes on a secondary attribute

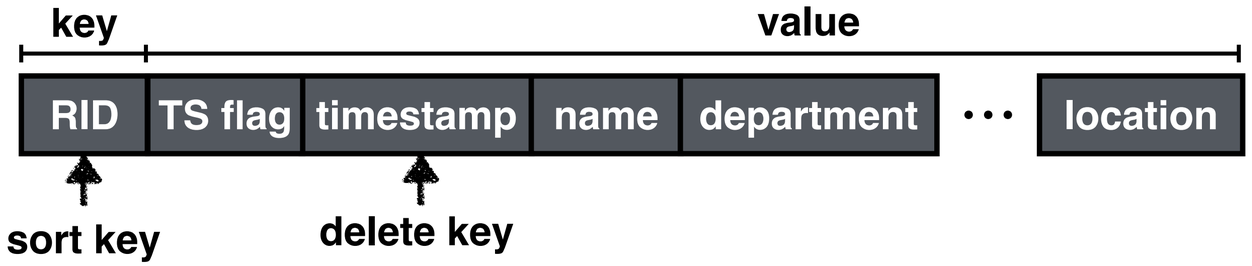
delete all entries older than: **D days**

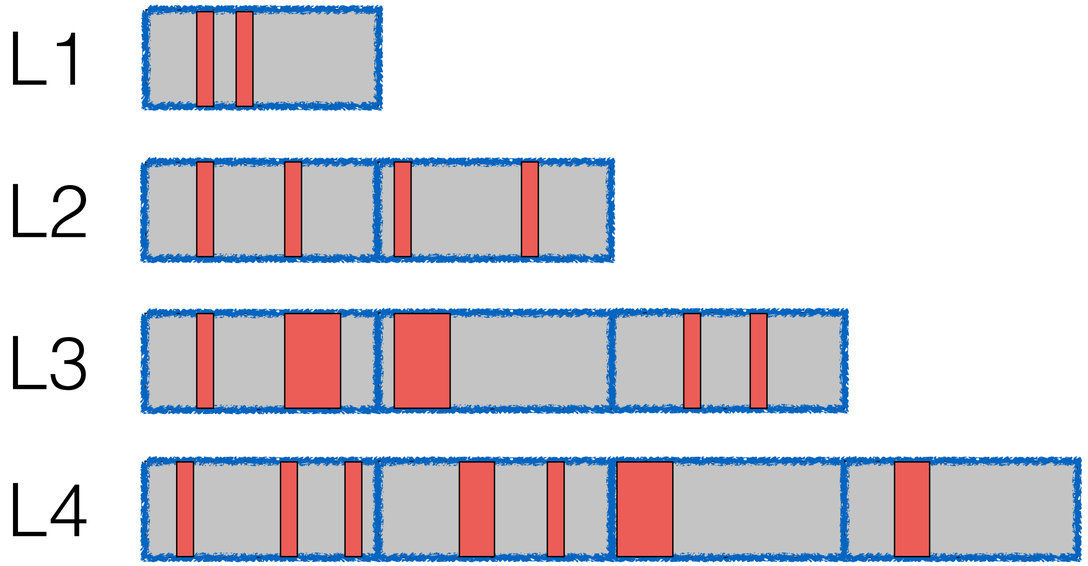




deletes on a secondary attribute

delete all entries older than: **D days**

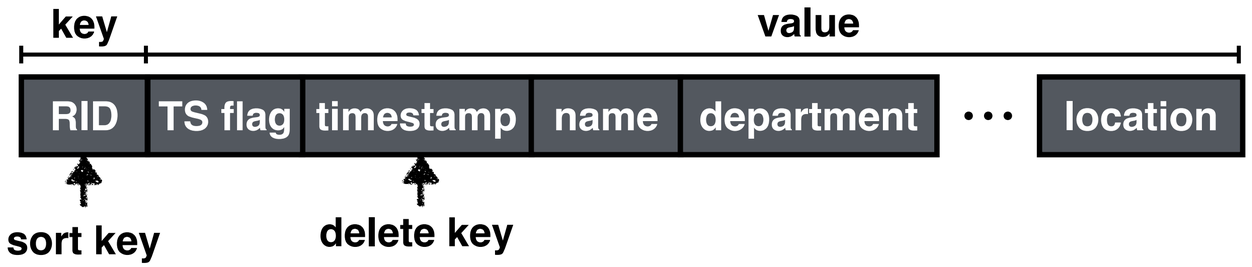




scattered occurrences

deletes on a secondary attribute

delete all entries older than: **D days**



L1

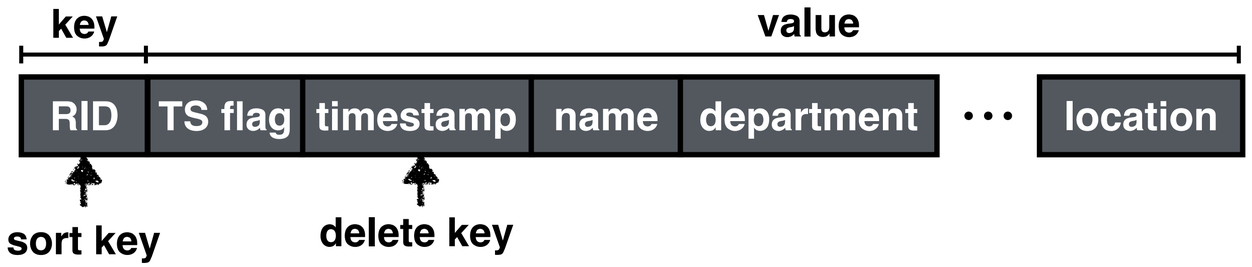
L2

L3

L4

deletes on a secondary attribute

delete all entries older than: **D days**



latency spikes ✗

superfluous I/Os ✗

L1

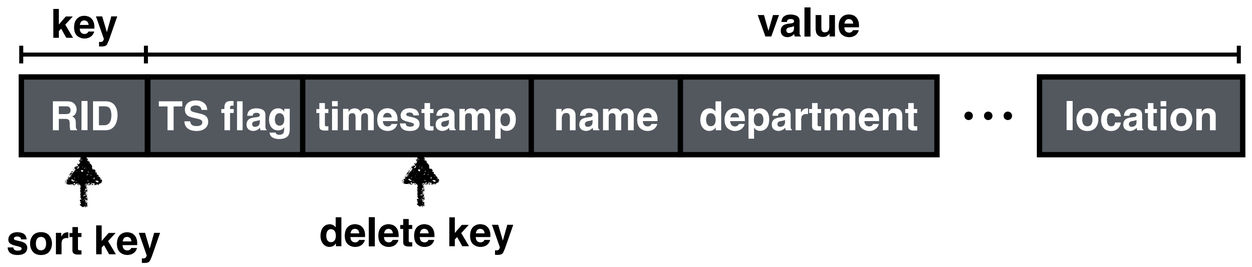
L2

L3

L4

deletes on a secondary attribute

delete all entries older than: **D days**



latency spikes ✗

superfluous I/Os ✗

L1

L2

L3

L4

the problems

poor read perf.

write amplification

space amplification

$\sum_{i=1}^{L-1} t_i$

D_{th}

unbounded delete persistence latency

latency spikes

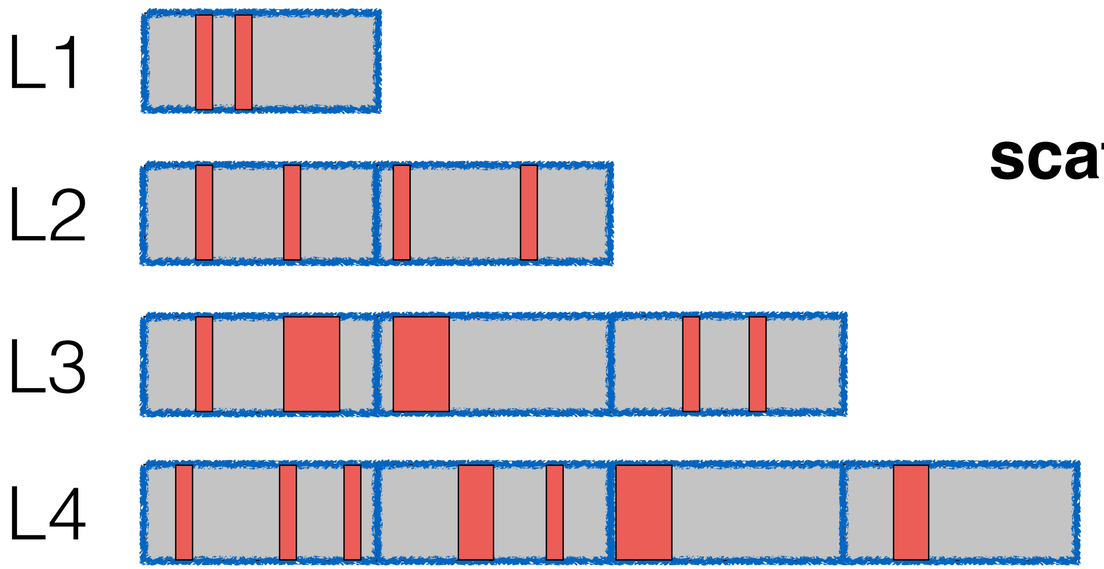
superfluous I/Os

FADE

KiWi

Key Weaving storage layout

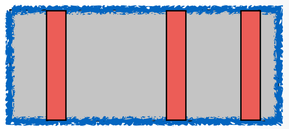
delete all entries older than: **D days**



scattered occurrences

Key Weaving storage layout

delete all entries with timestamp **<= 65_D**



$S_{min}=1 :: S_{max}=99$
$D_{min}=1_D :: D_{max}=90_D$

page 1: $S_{min}=1 :: S_{max}=24$, $D_{min}=3_D :: D_{max}=80_D$

page 2: $S_{min}=29 :: S_{max}=60$, $D_{min}=9_D :: D_{max}=90_D$

page 3: $S_{min}=61 :: S_{max}=79$, $D_{min}=1_D :: D_{max}=89_D$

page 4: $S_{min}=80 :: S_{max}=99$, $D_{min}=7_D :: D_{max}=85_D$

page 1

1	4	9	14	15	19	20	24
34_D	69_D	3_D	79_D	8_D	80_D	23_D	24_D

page 2

29	32	33	40	44	52	56	60
88_D	90_D	28_D	74_D	9_D	76_D	81_D	64_D

page 3

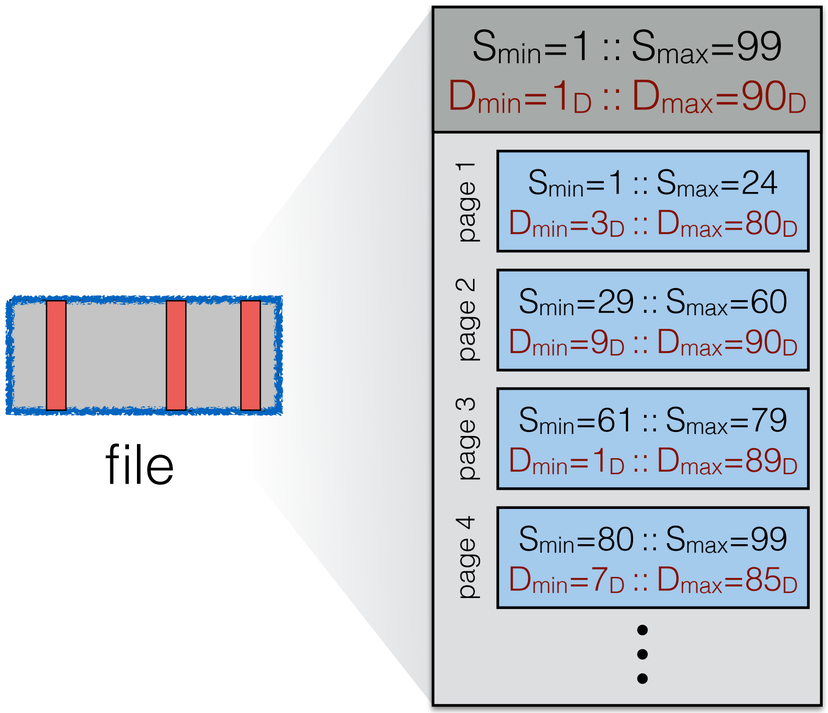
61	63	67	71	72	73	78	79
75_D	82_D	1_D	67_D	77_D	89_D	65_D	12_D

page 4

80	84	86	87	91	94	95	99
70_D	41_D	62_D	7_D	25_D	85_D	59_D	19_D

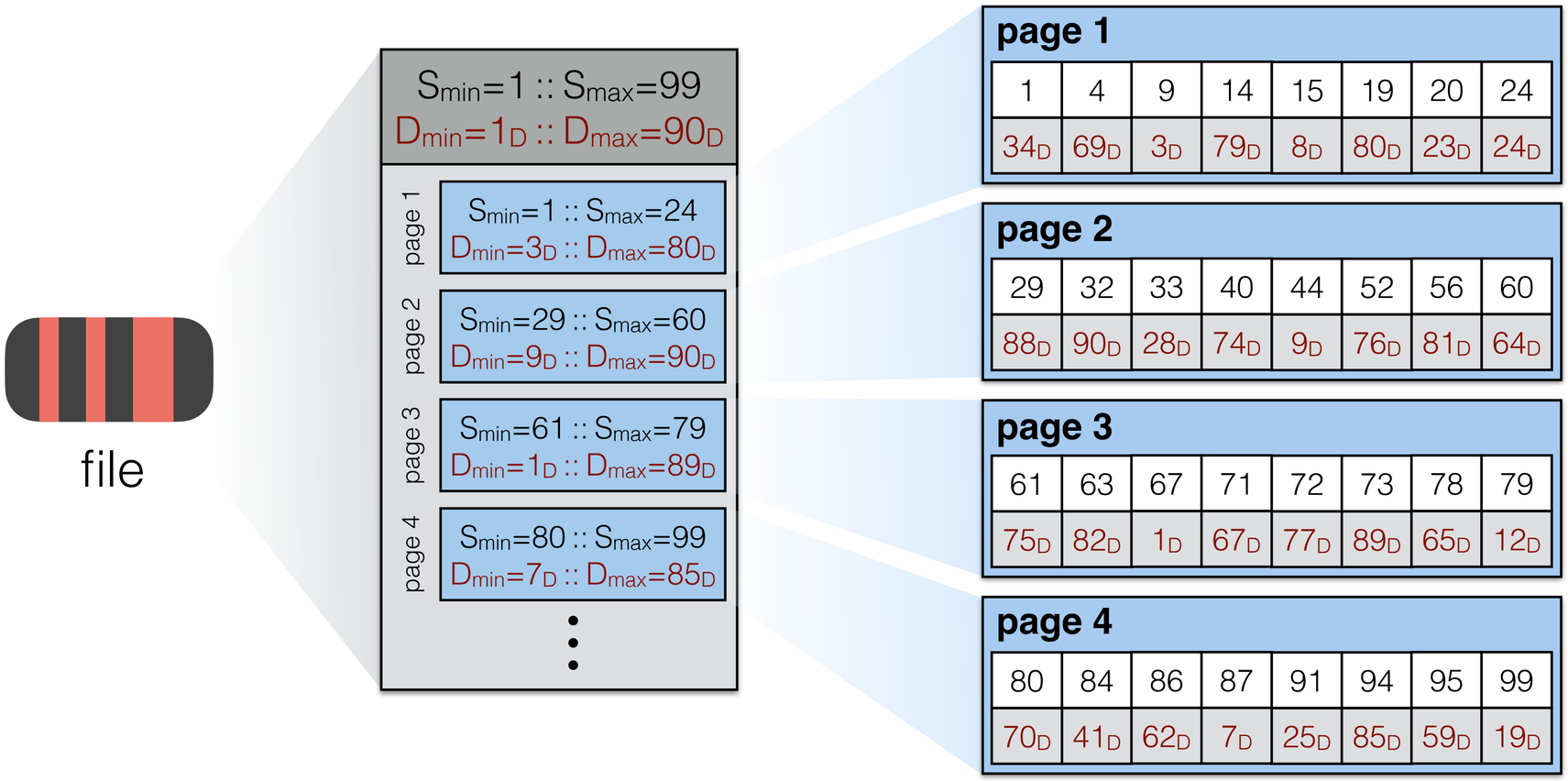
Key Weaving storage layout

delete all entries with timestamp **<= 65_D**



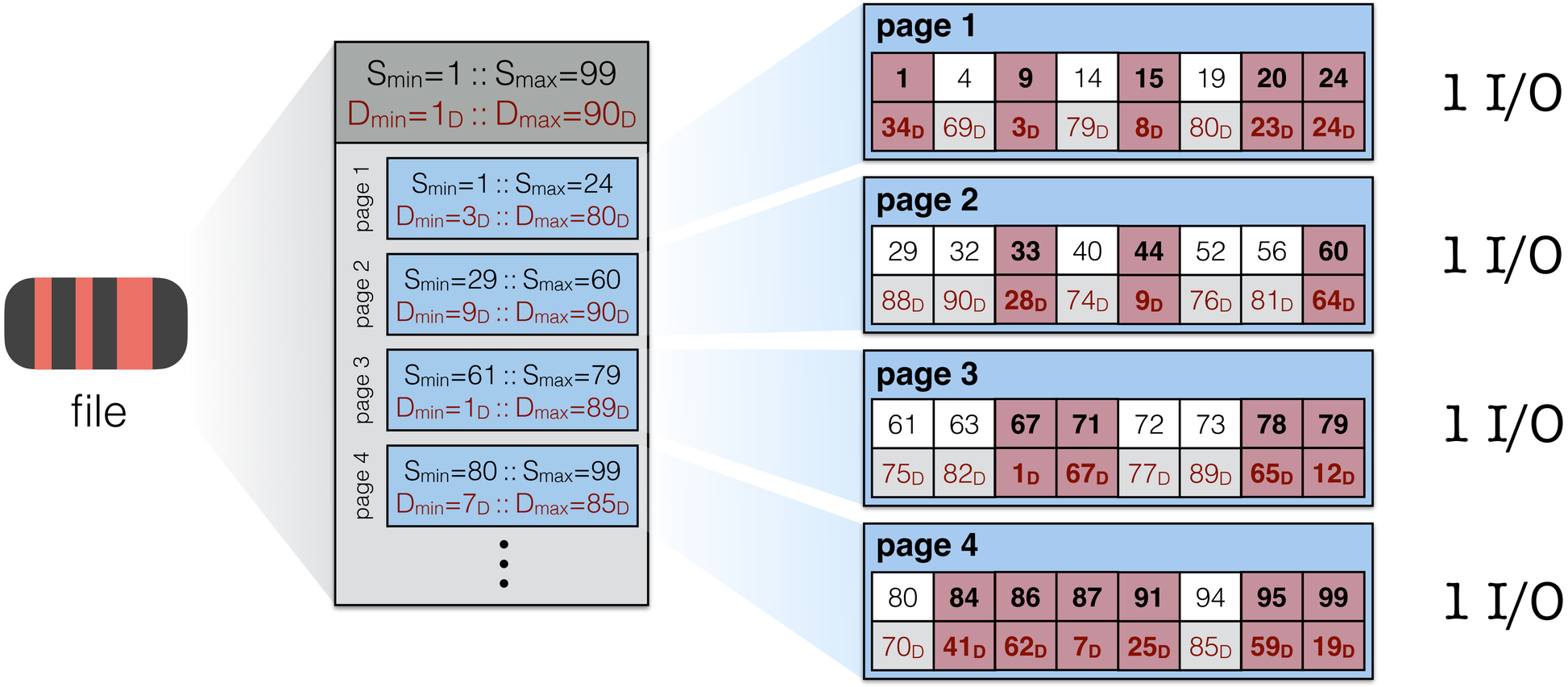
Realizing **Retention-Based Deletes**

delete all entries older than <= 65_D



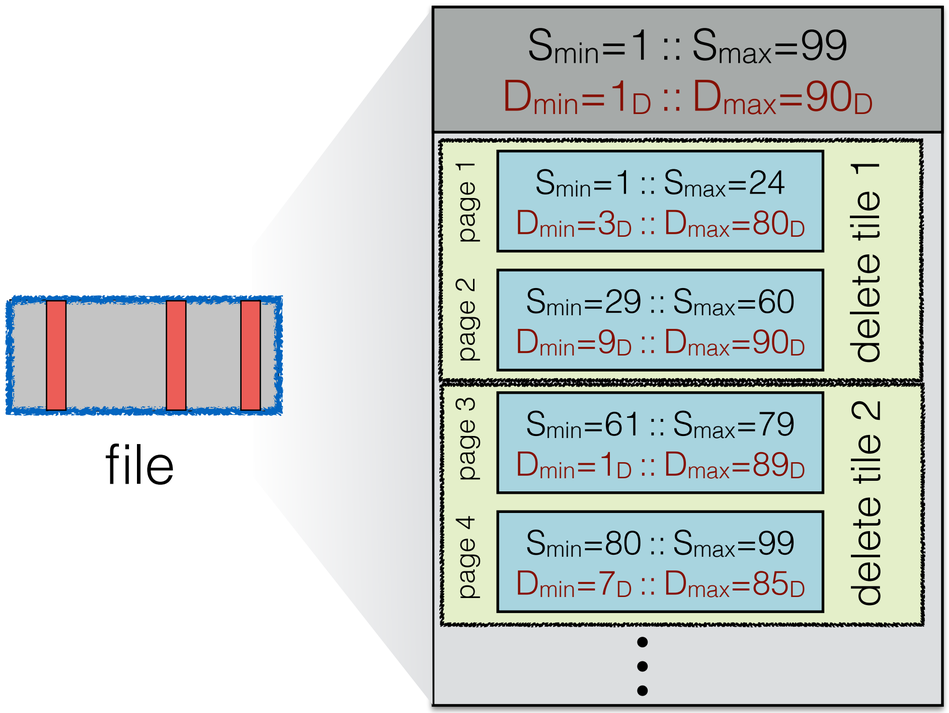
Realizing **Retention-Based Deletes**

delete all entries older than <= 65_D



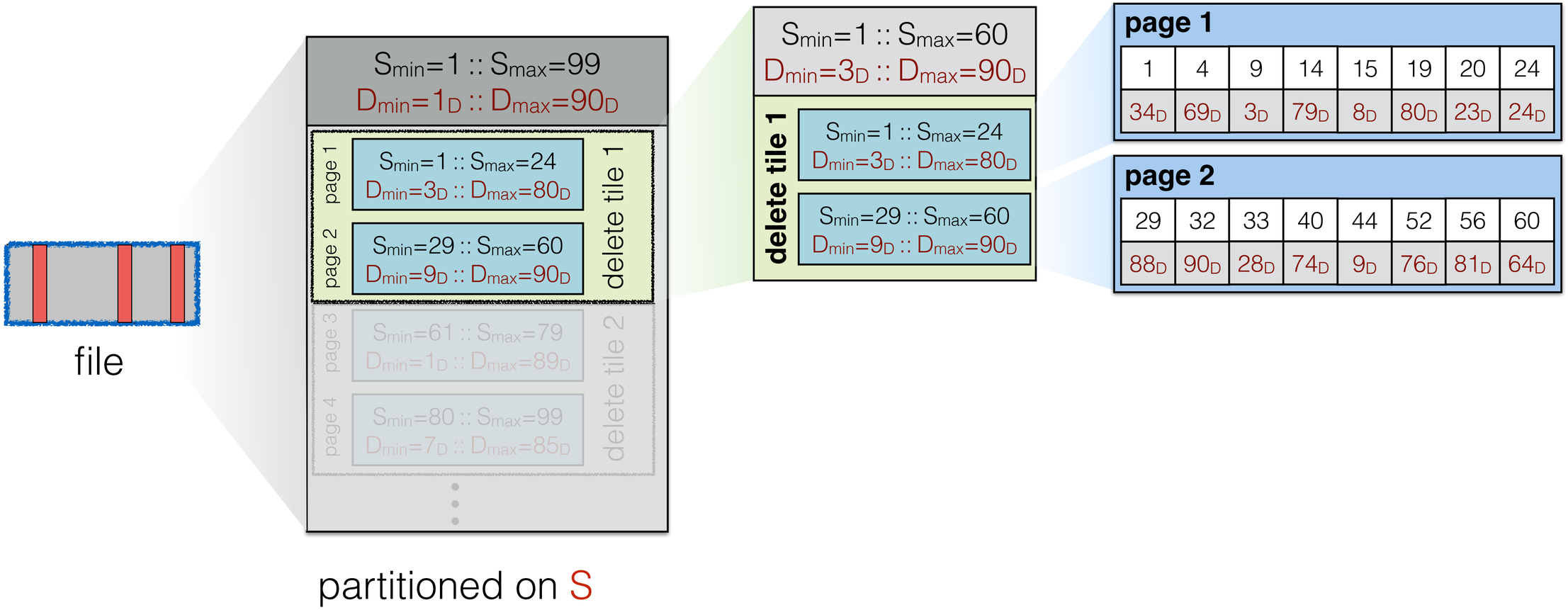
Key Weaving storage layout

delete all entries with timestamp **<= 65_D**



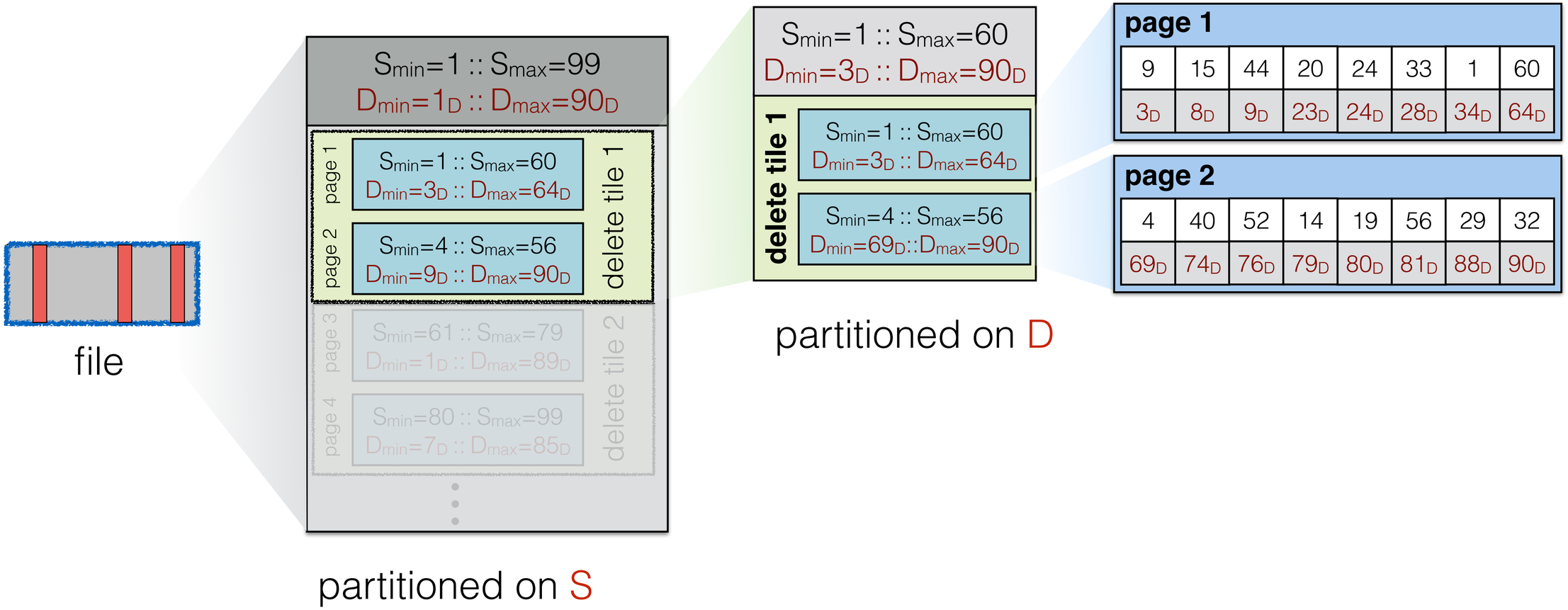
Key Weaving storage layout

delete all entries with timestamp <= 65_D



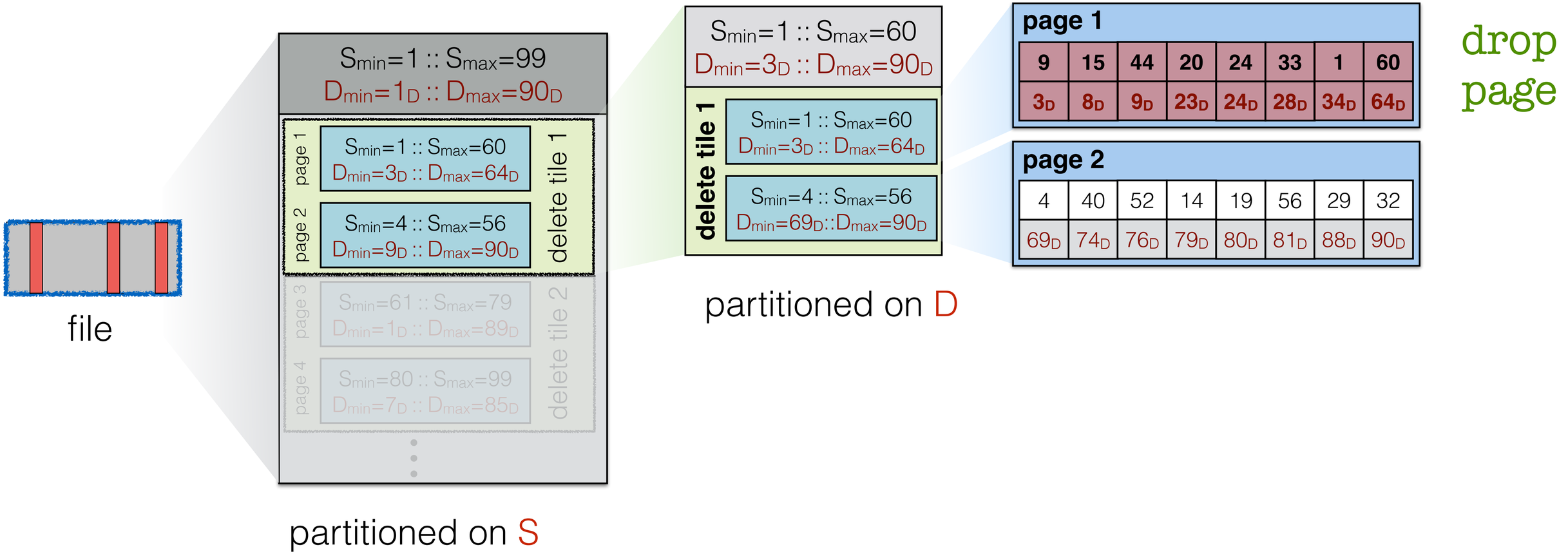
Key Weaving storage layout

delete all entries with timestamp <= 65_D



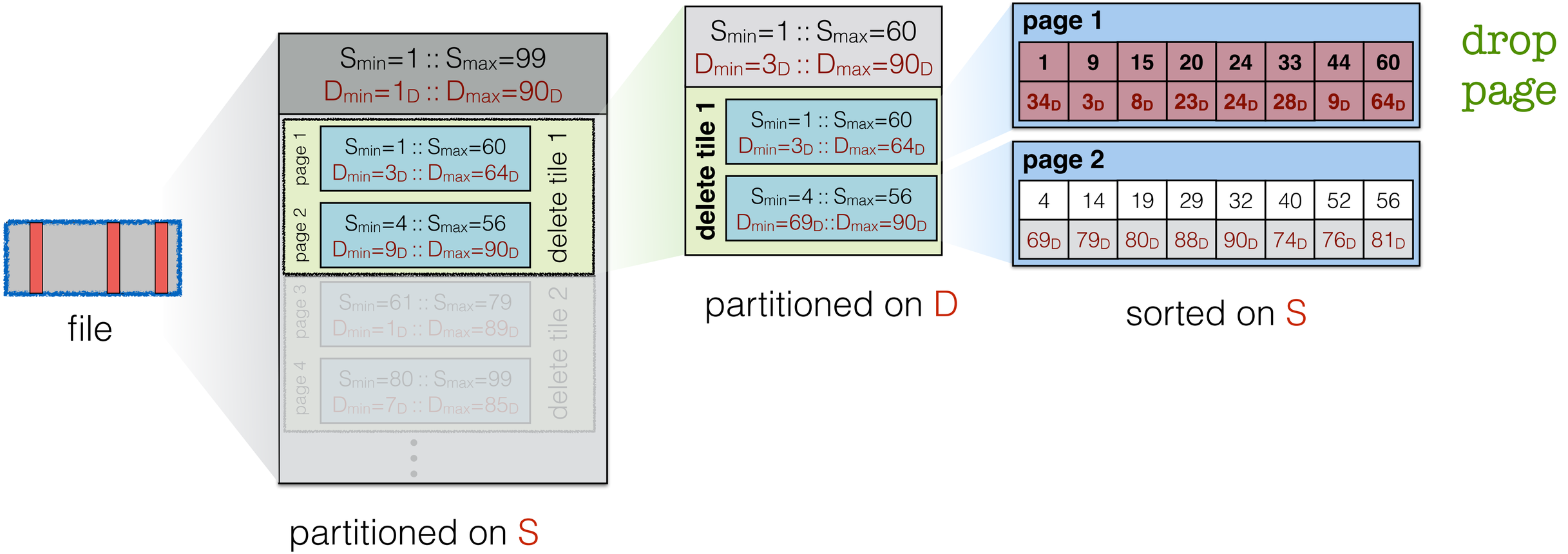
Key Weaving storage layout

delete all entries with timestamp **<= 65_D**



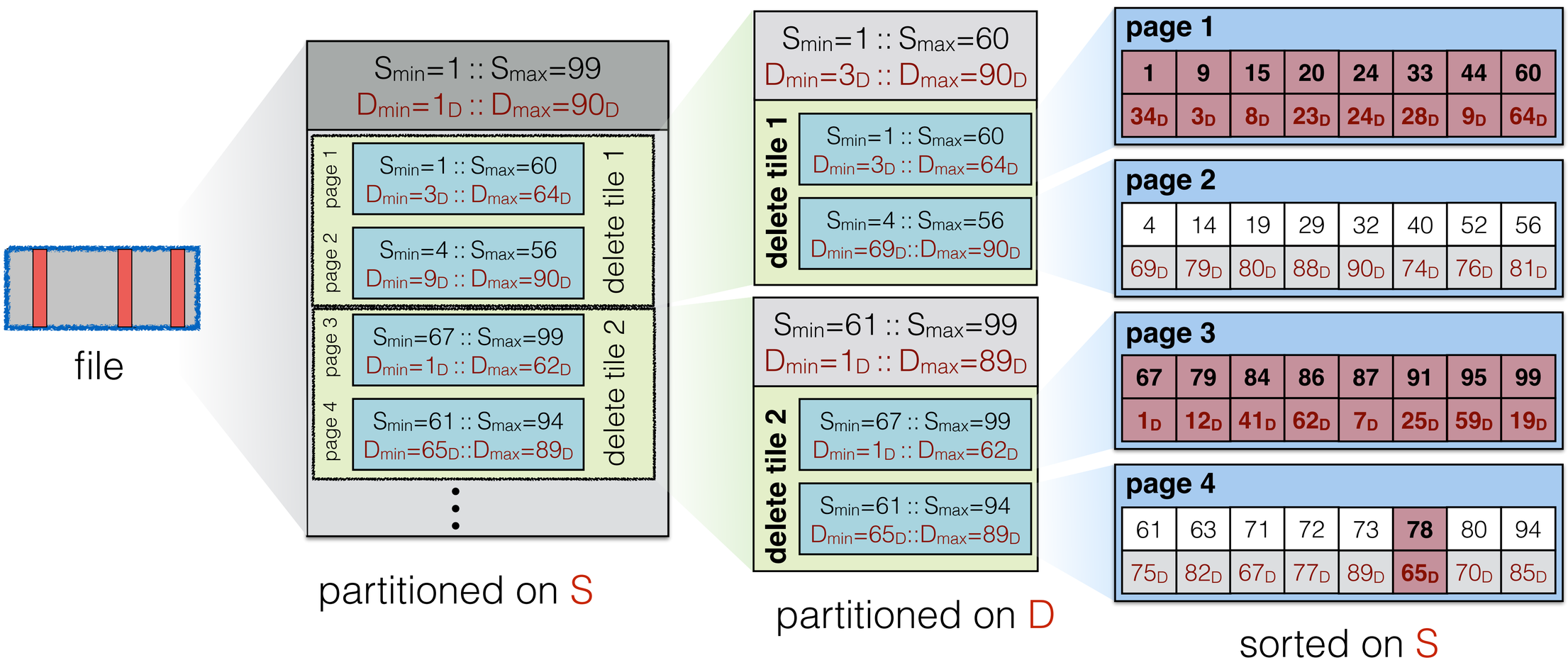
Key Weaving storage layout

delete all entries with timestamp **<= 65_D**



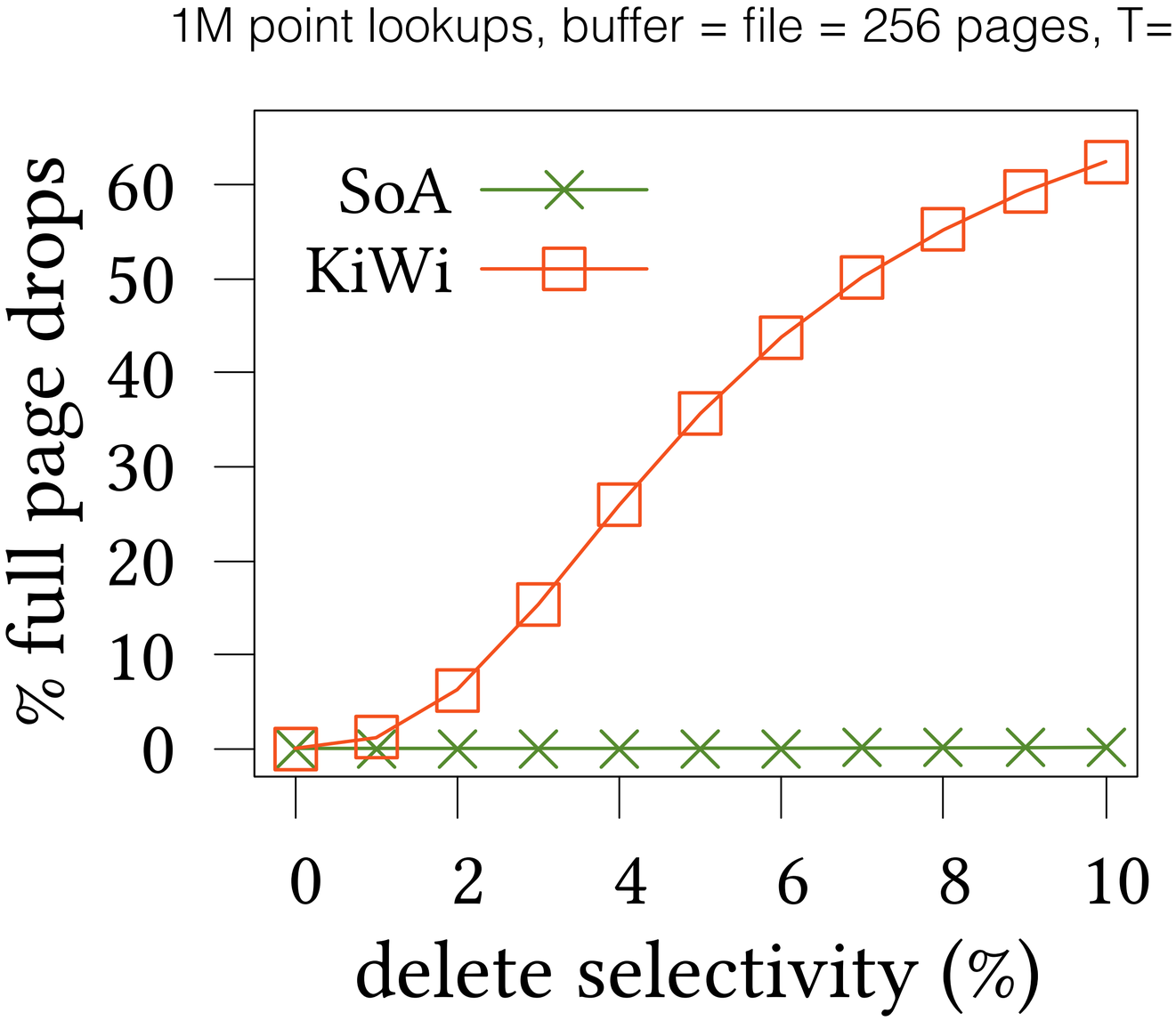
Key Weaving storage layout

delete all entries with timestamp <= 65_D

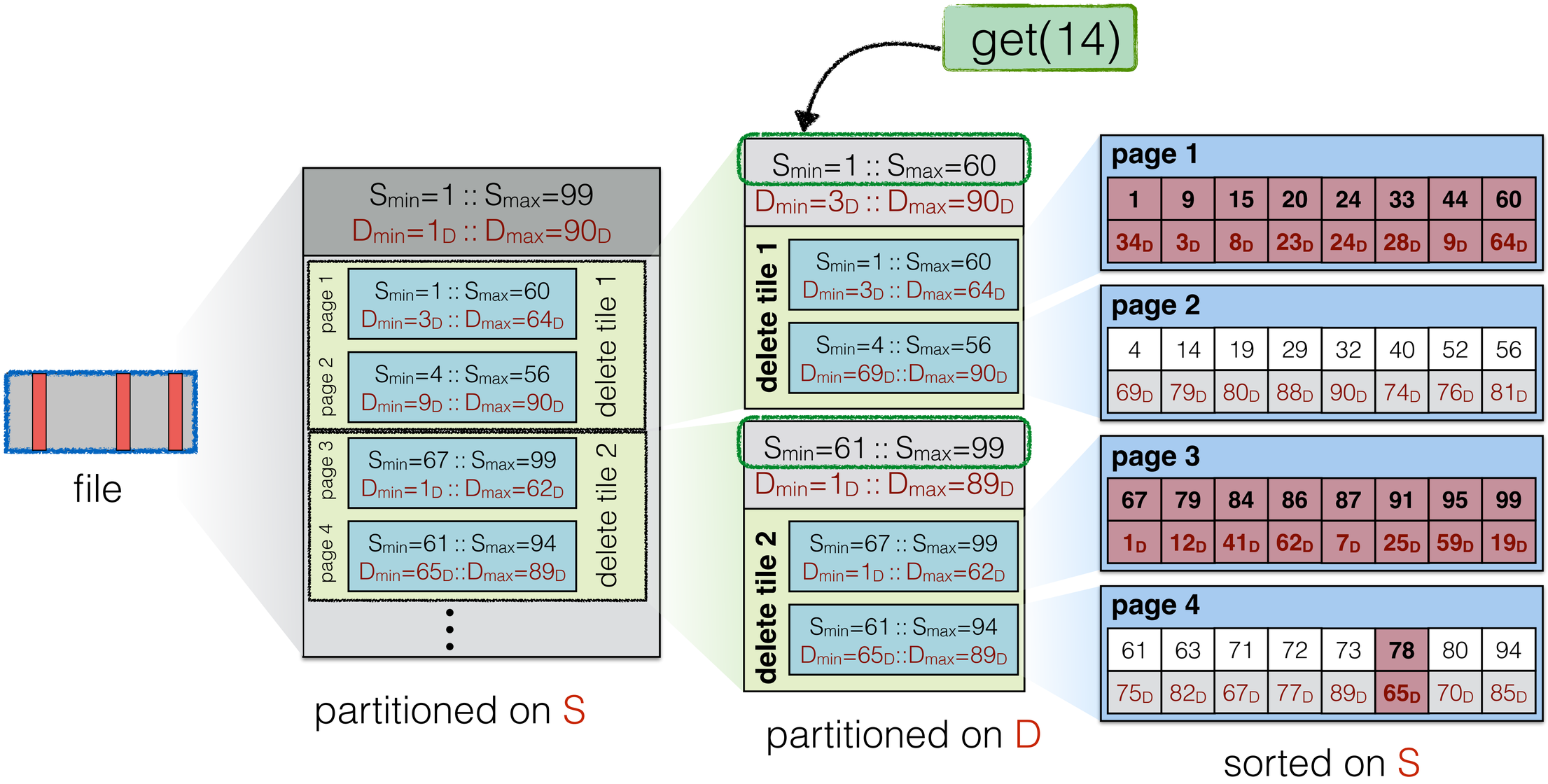


superior delete performance

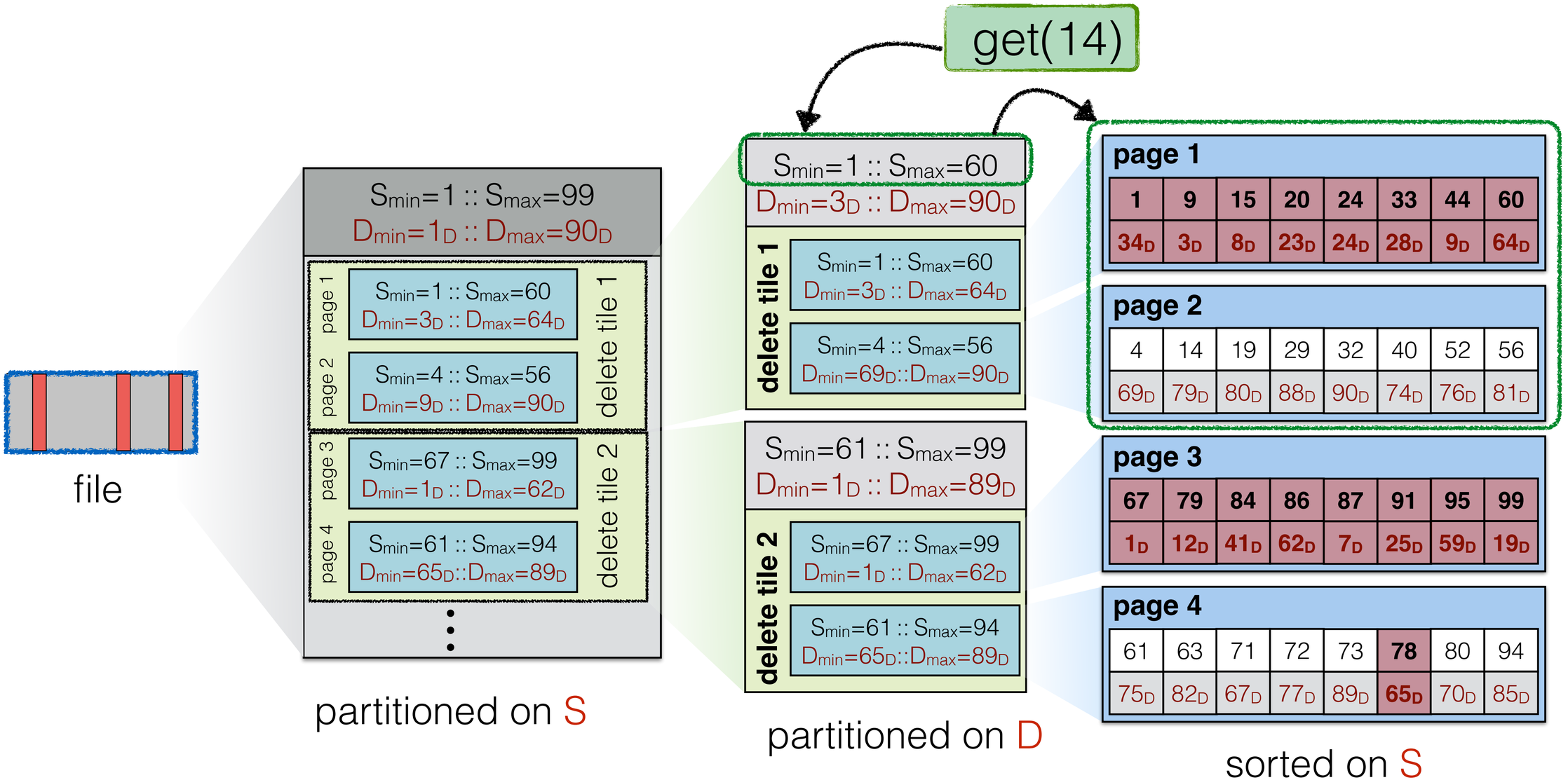
up to 2.5x

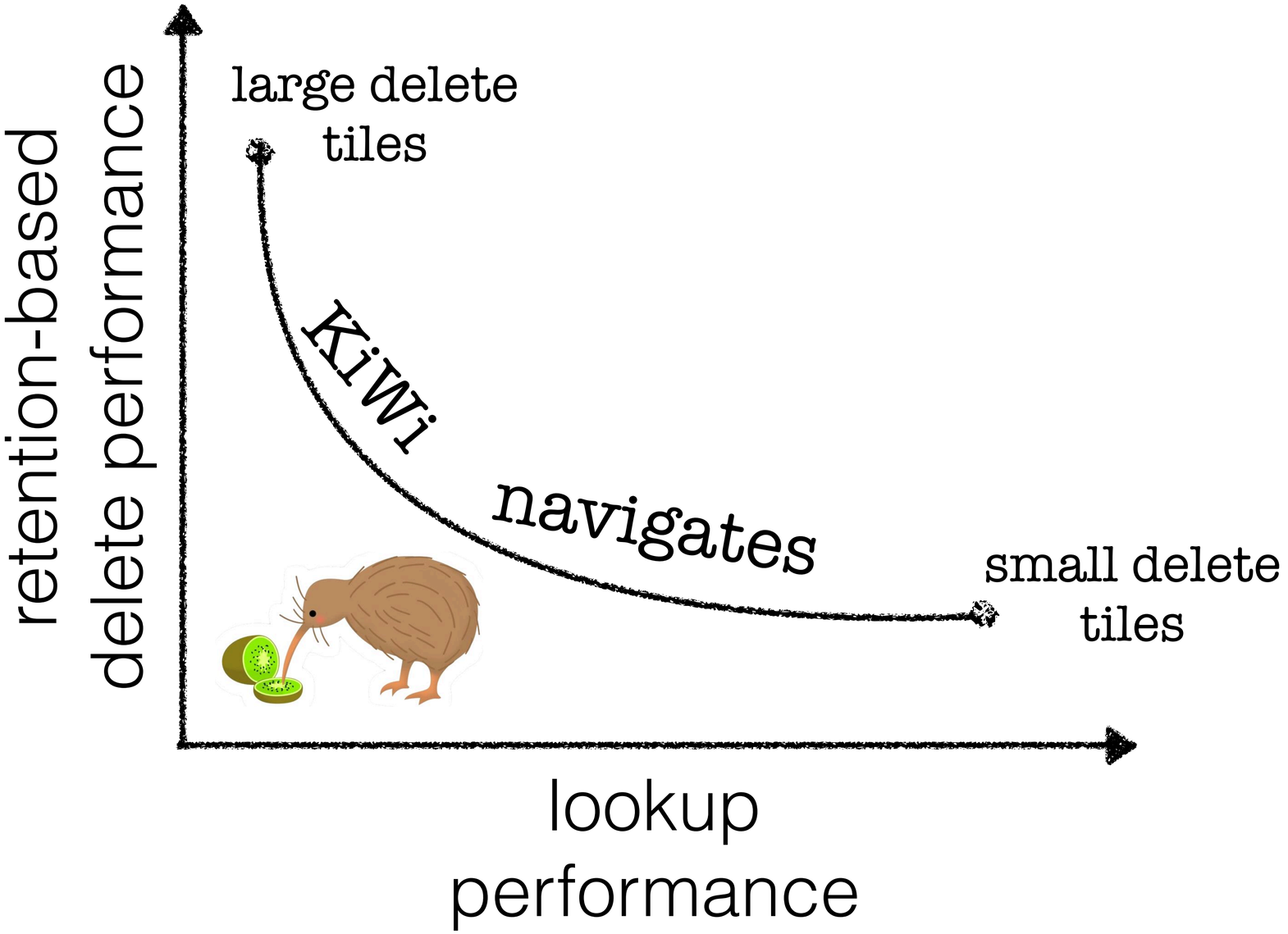


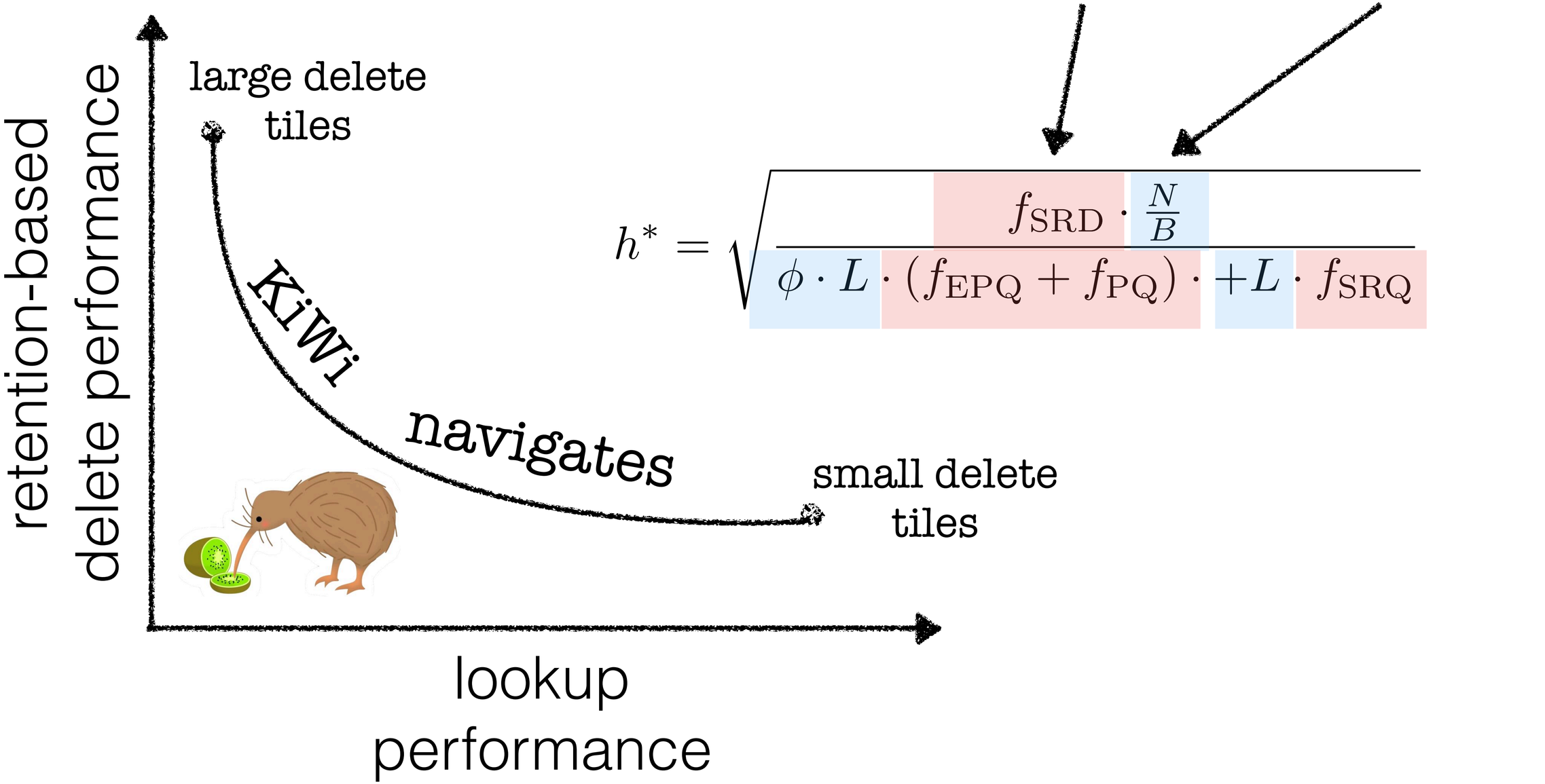
Key Weaving storage layout



Key Weaving storage layout




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

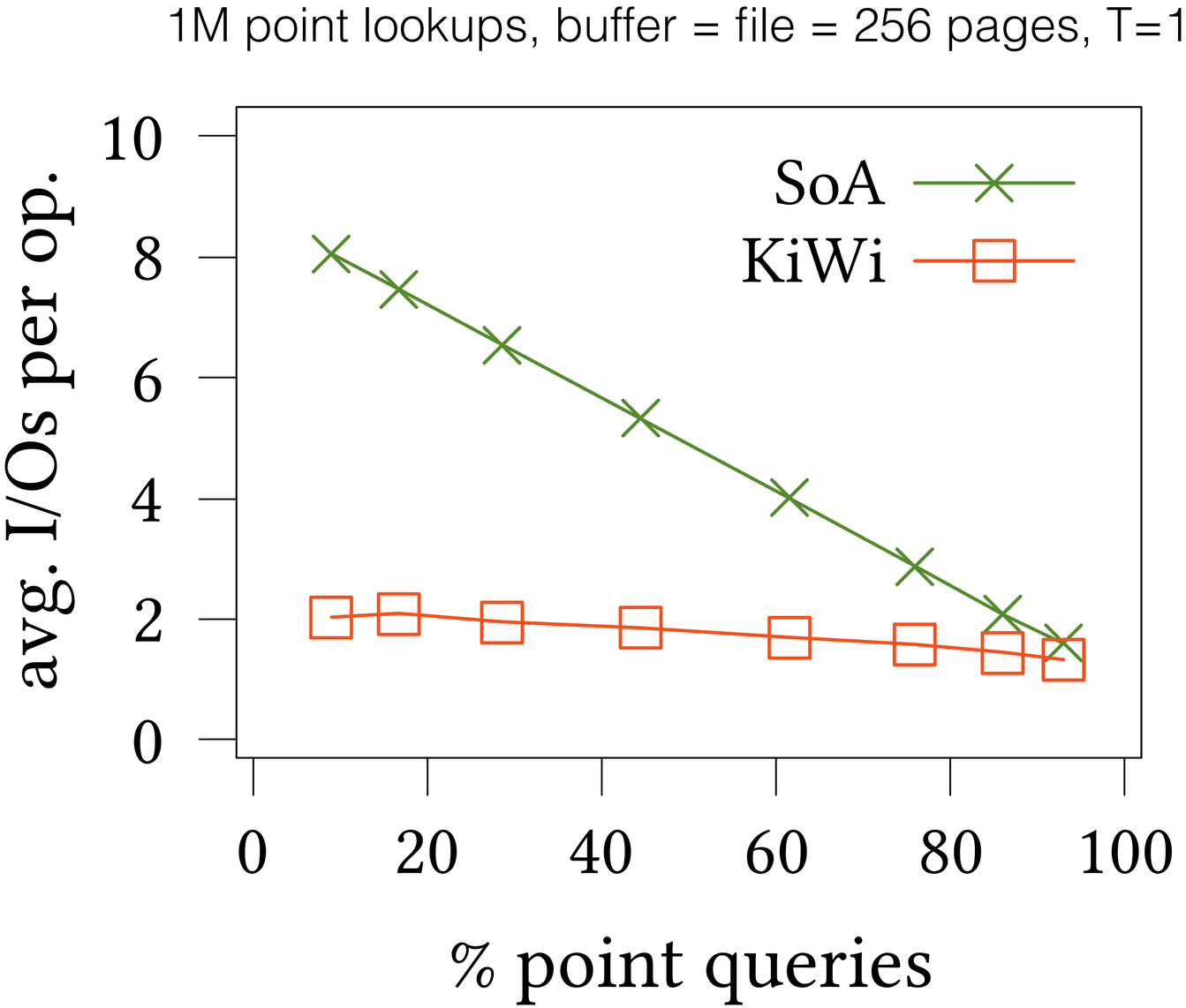

data structure
workload
large delete tiles
small delete tiles
KiWi navigates
retention-based delete performance
lookup performance
$h^* = \sqrt{\frac{f_{\text{SRD}} \cdot \frac{N}{B}}{\phi \cdot L \cdot (f_{\text{EPQ}} + f_{\text{PQ}}) \cdot + L \cdot f_{\text{SRQ}}}}$

better overall performance

up to 4x

superior delete performance

up to 2.5x



the solution

FAst DElete

amortized write

reduced space

improved read

timely delete persistence

FADE

higher lookup cost

reduced latency spikes

full page drops reduces superfluous I/Os

KiWi

suboptimal state-of-the-art design

for workloads with deletes

FADE persists deletes timely

using latency-driven compactions

KiWi supports efficient
secondary range deletes

using key-interweaved data storage

CS 561: ***Data Systems Architecture***

Class 6

Efficient Deletes in LSM-Engines

Dr. Subhadeep Sarkar

BOSTON UNIVERSITY

DiSC lab

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