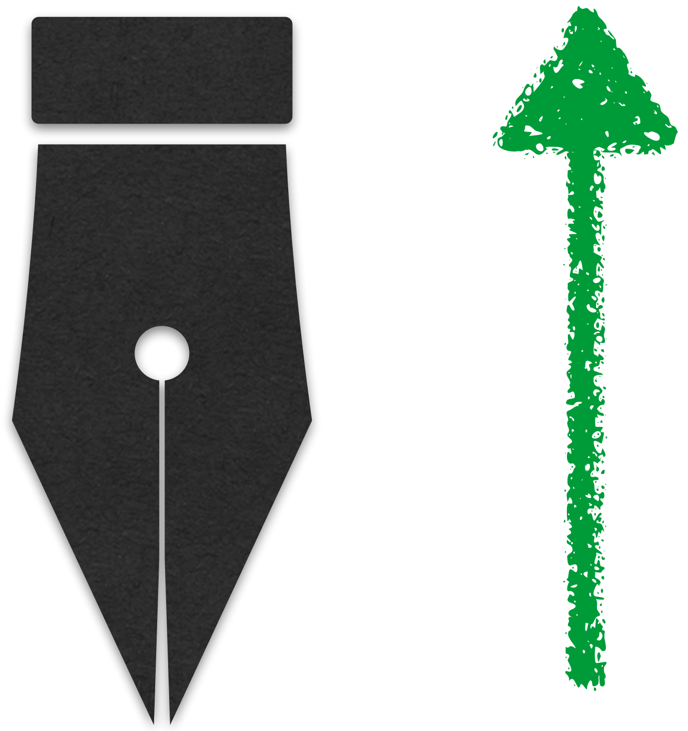
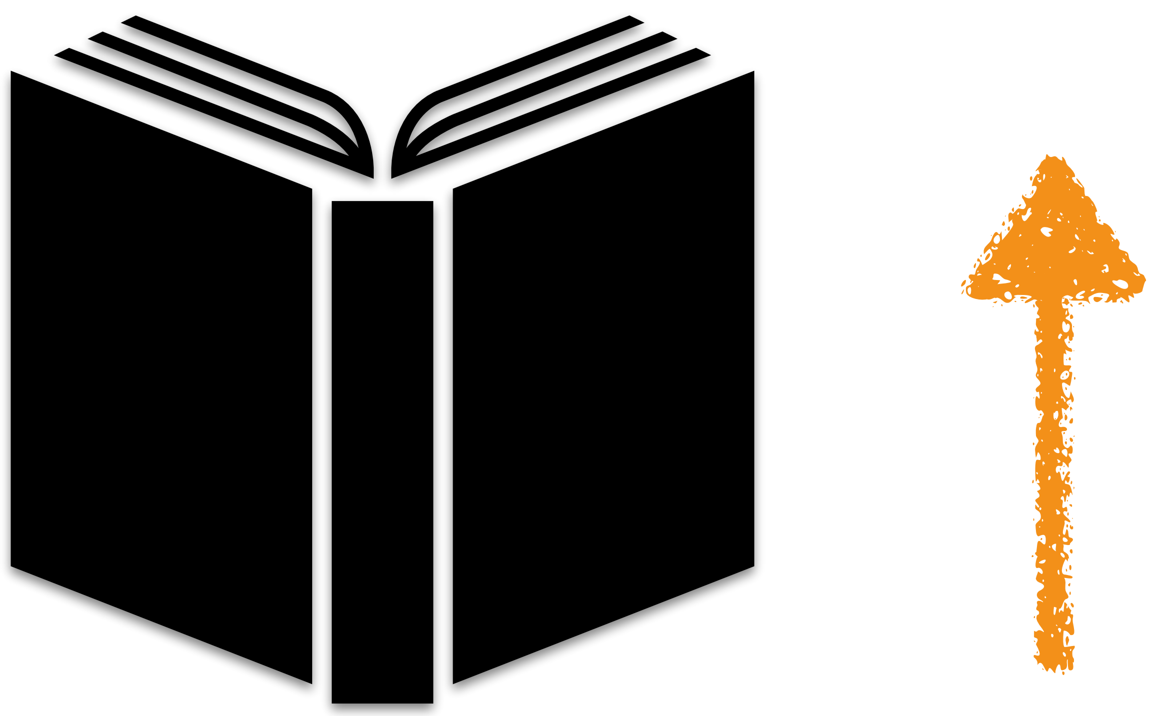
Lethe: A Tunable Delete-Aware LSM-Based Storage Engine

Subhadeep Sarkar
Tarikul Islam Papon
Dimitris Staratzis
Manos Athanassoulis





LSM-TREE

cassandra

levelDB

RocksDB

WT

accumulo

DynamoDB

SQLite

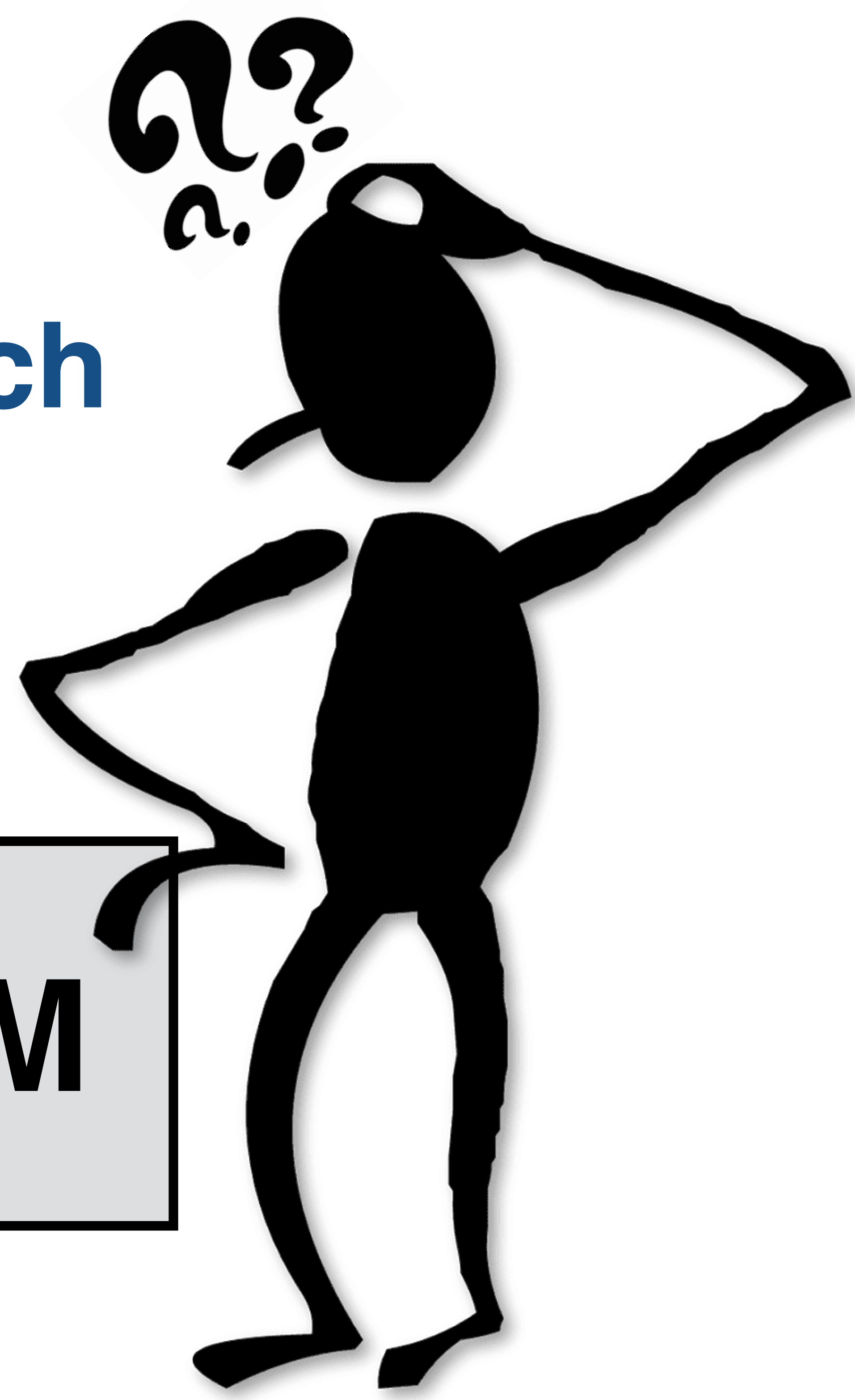
APACHE HBASE

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

TechCrunch
Feb '19

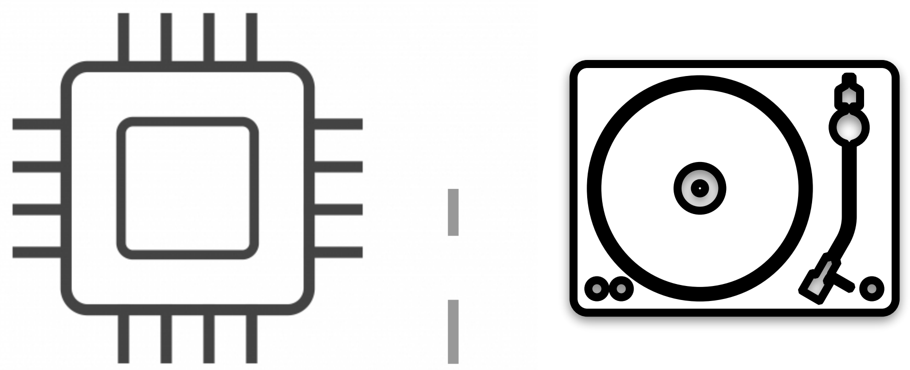
Small Datum
Jan '20

Deletes are fast and slow in an LSM



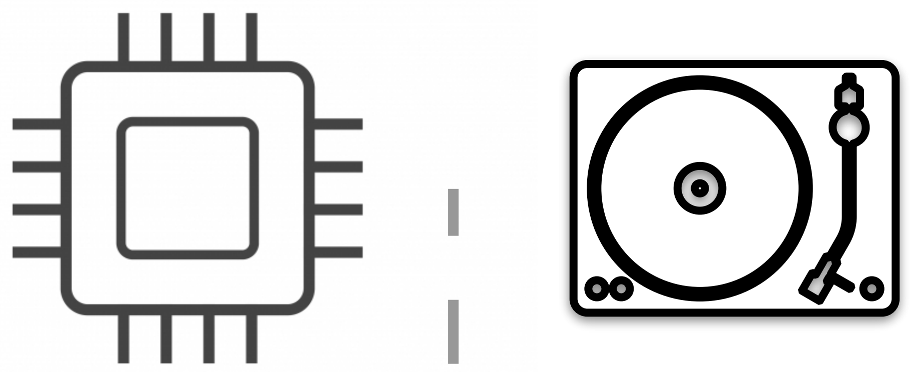
"LSM-based data stores perform suboptimally for workloads with deletes."

log-structured merge-tree



buffer

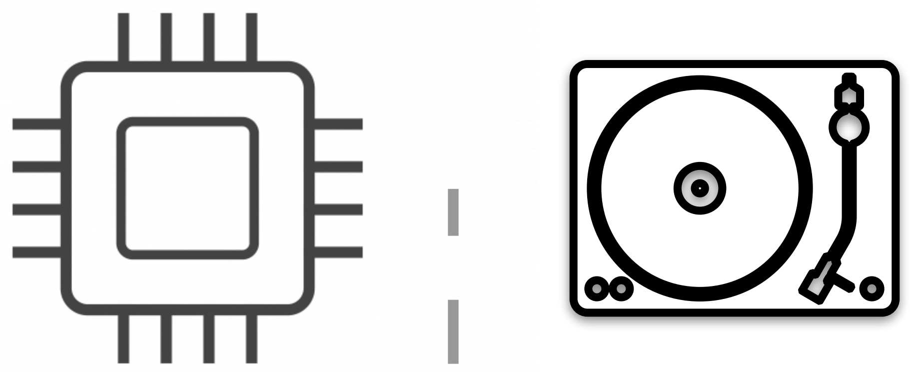
log-structured merge-tree



buffer

2	6	1	4

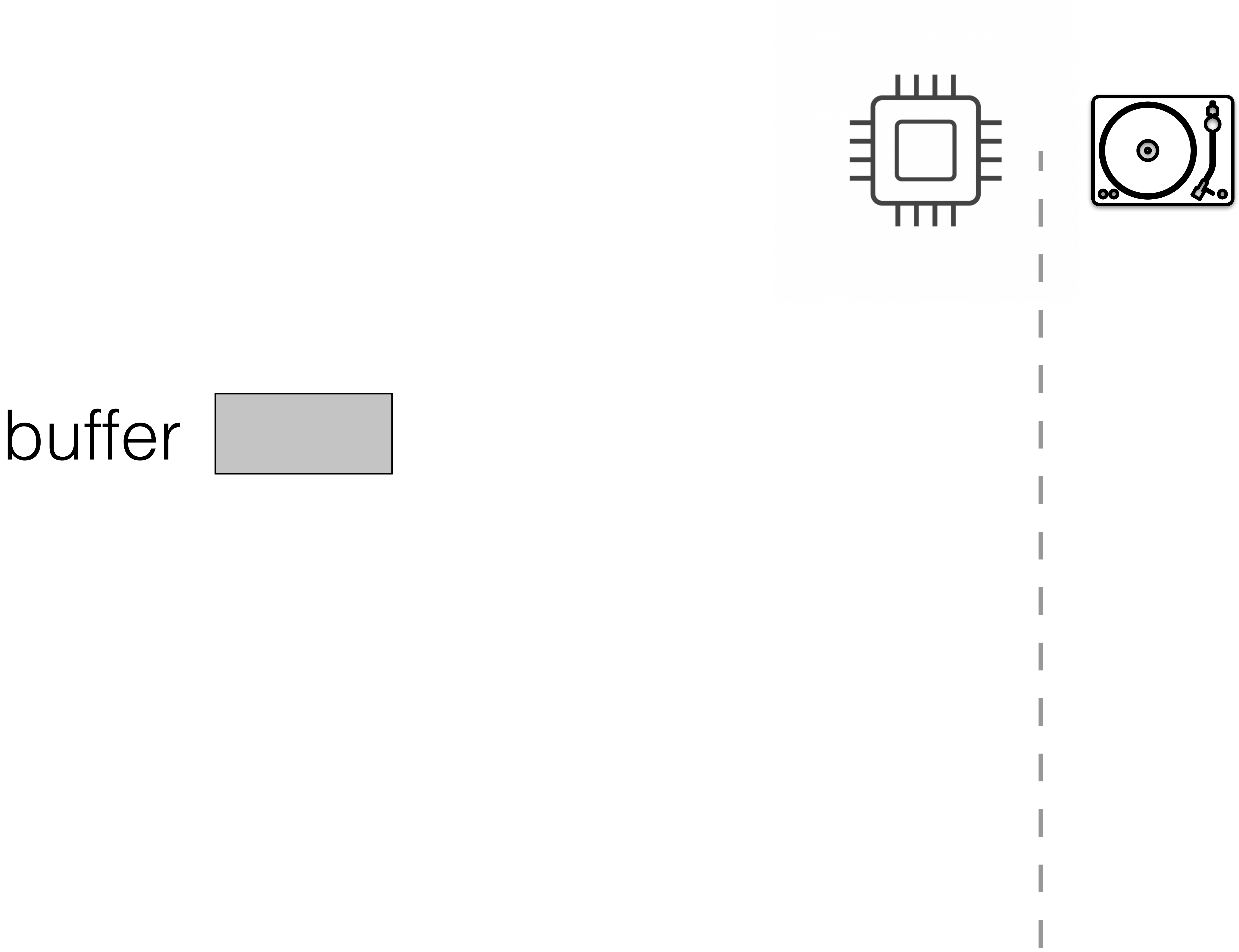
log-structured merge-tree



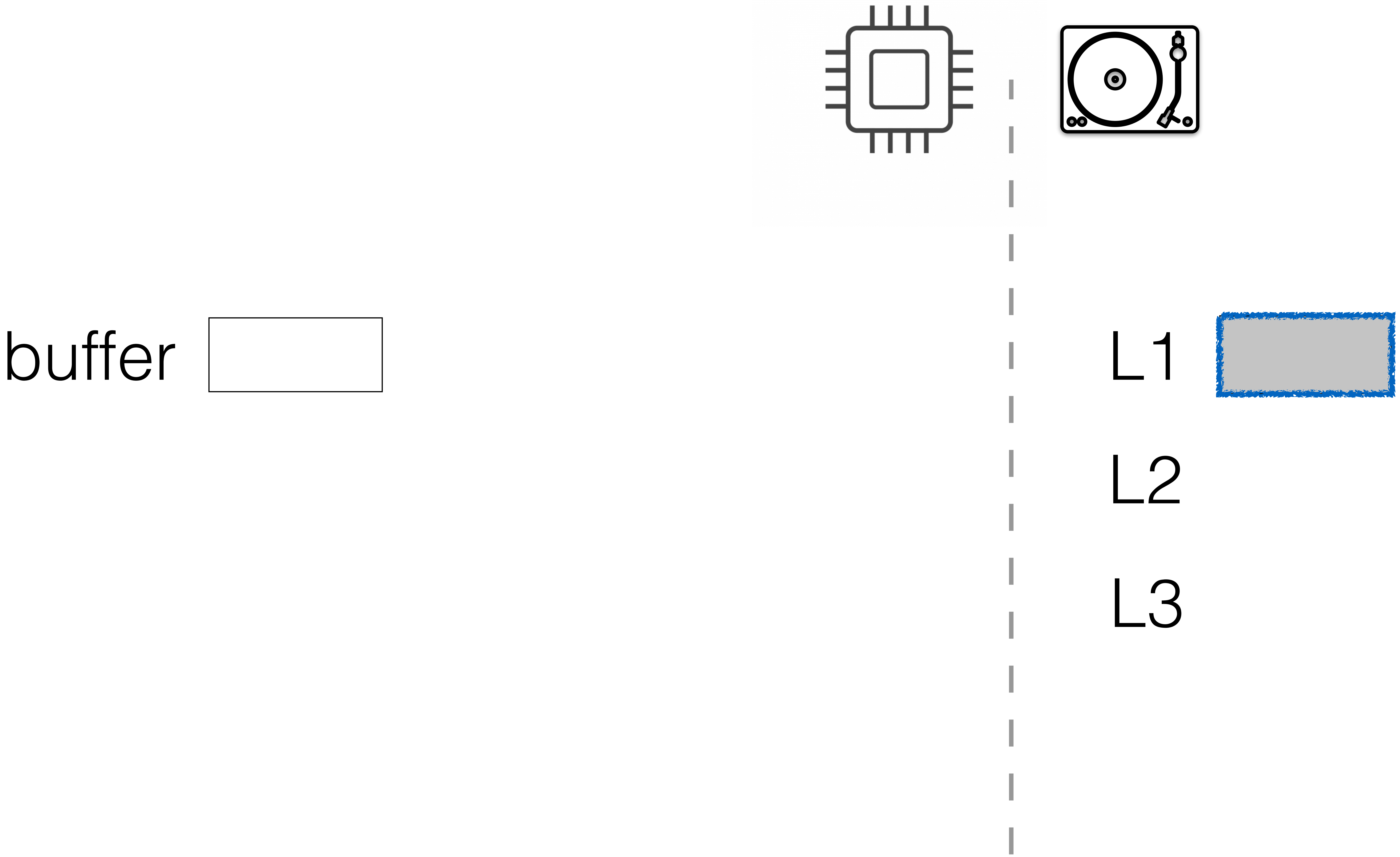
buffer

1	2	4	6

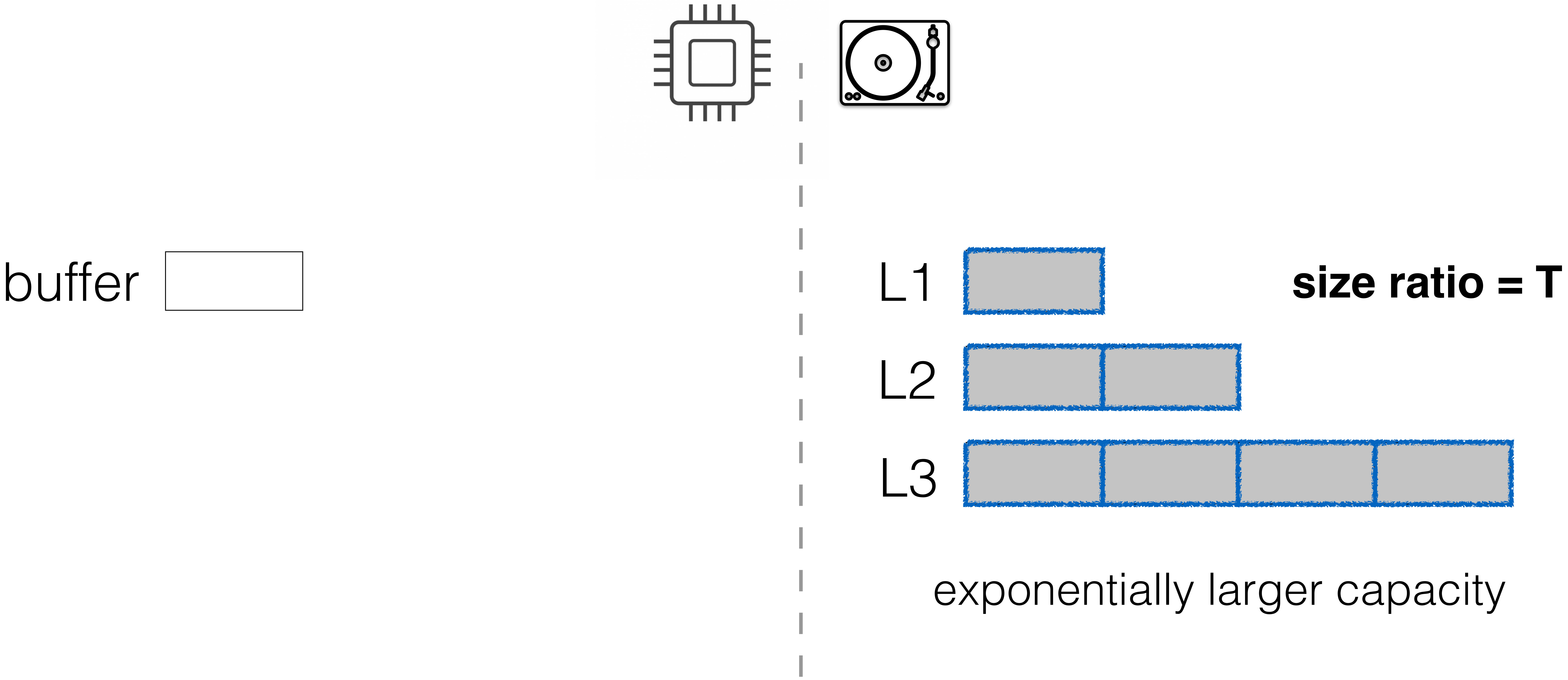
log-structured merge-tree



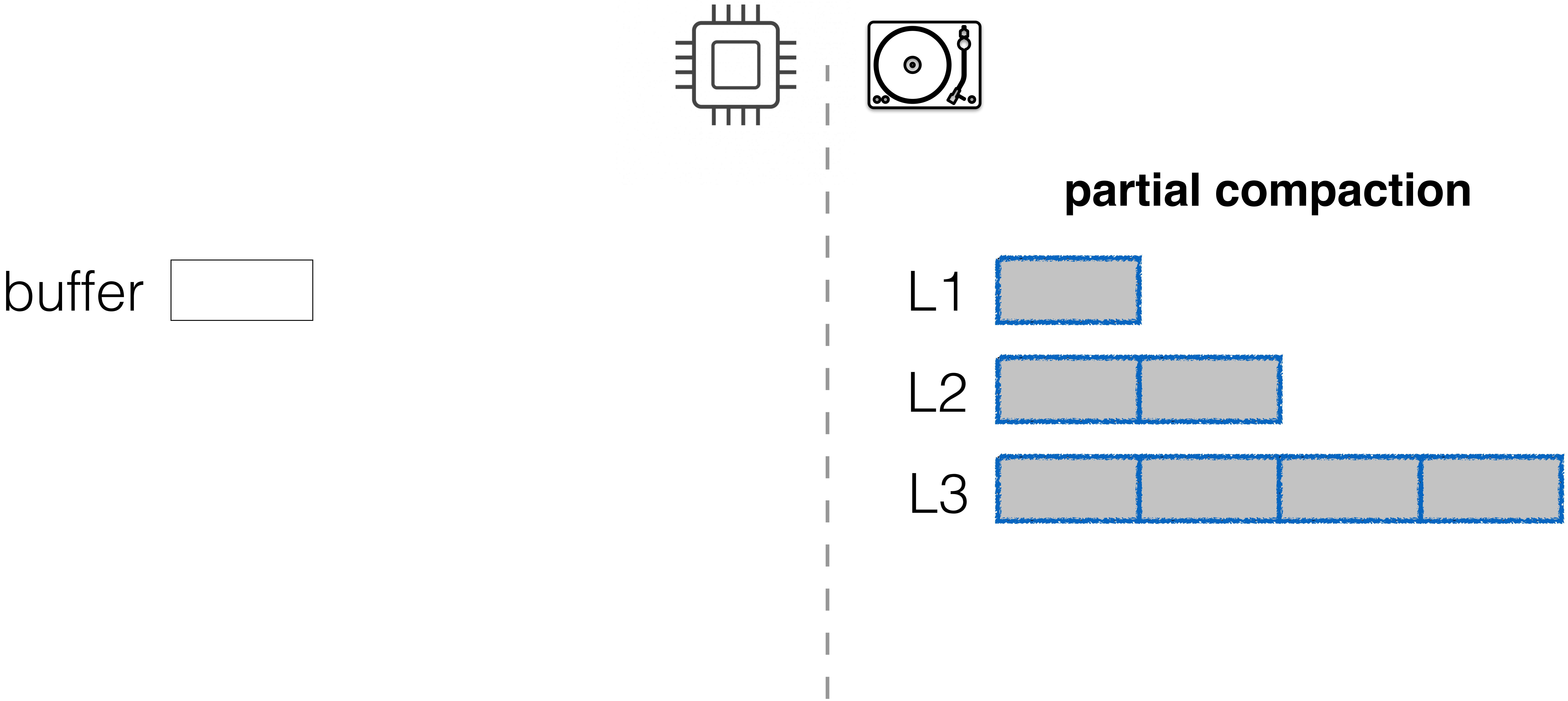
log-structured merge-tree



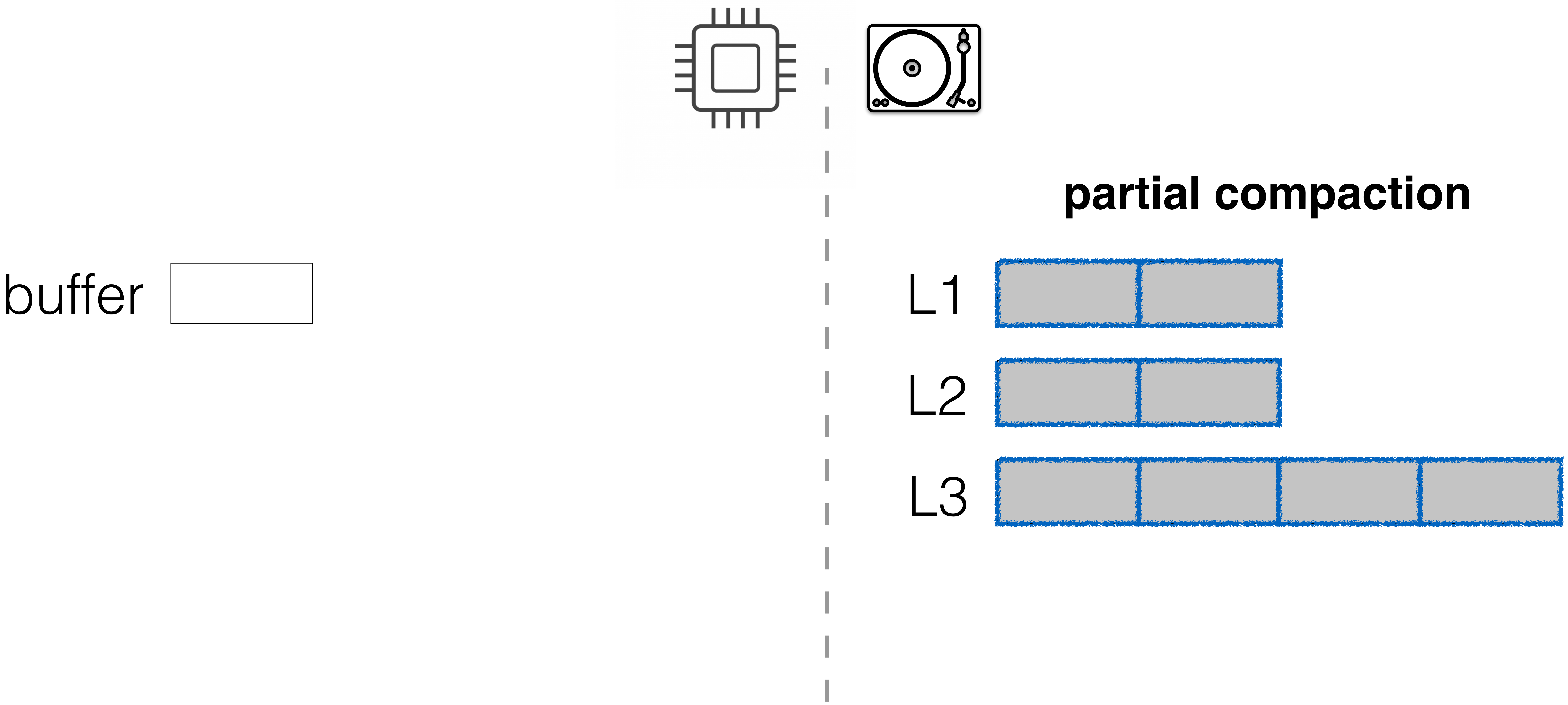
log-structured merge-tree



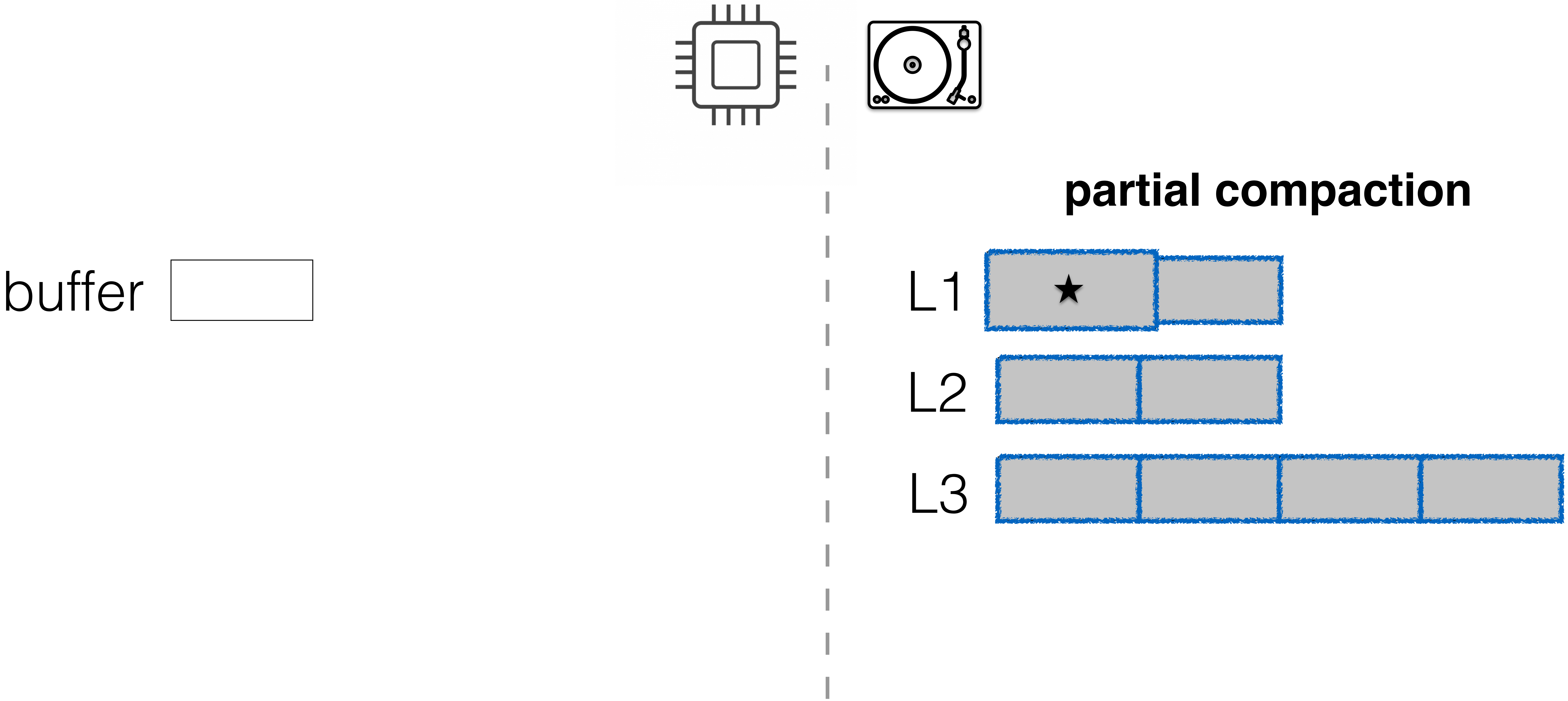
log-structured merge-tree



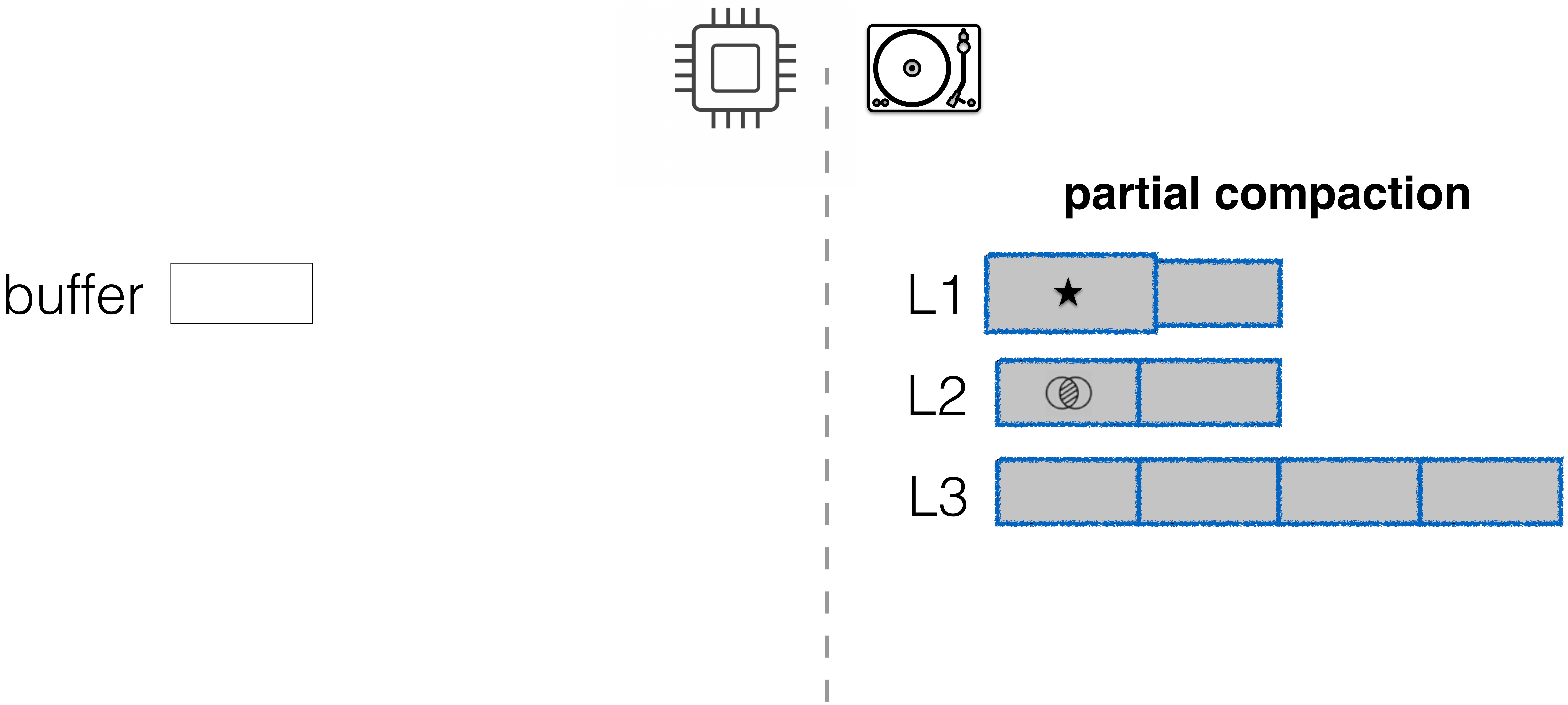
log-structured merge-tree



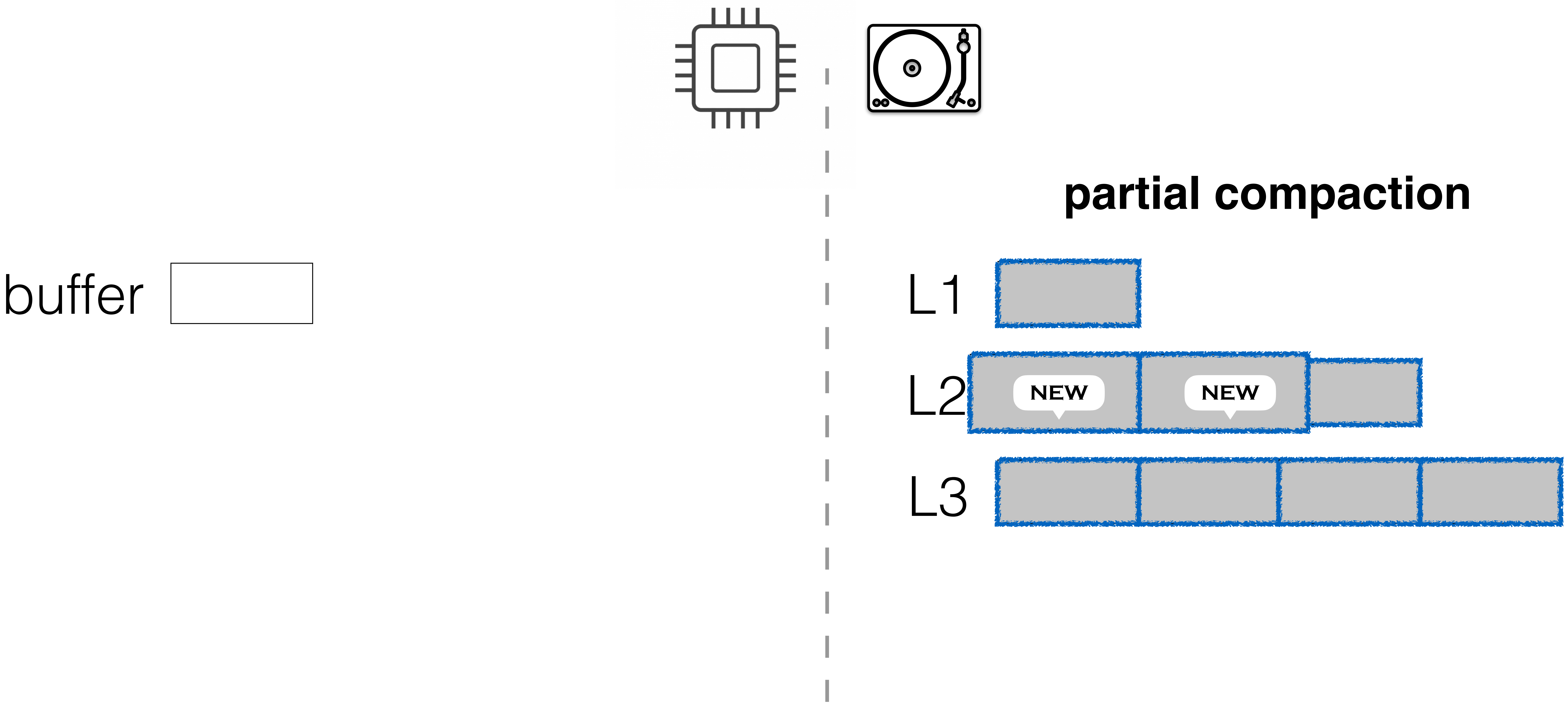
log-structured merge-tree



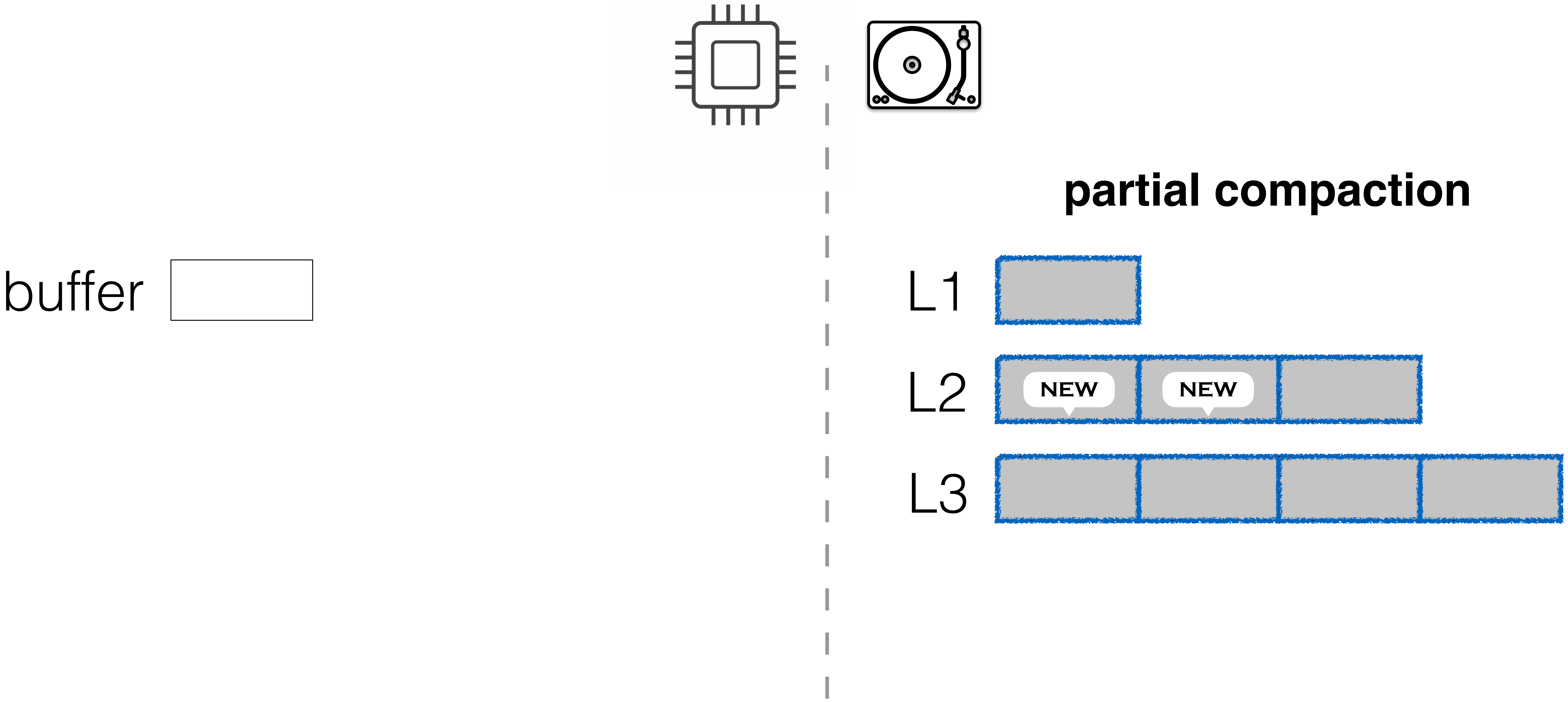
log-structured merge-tree



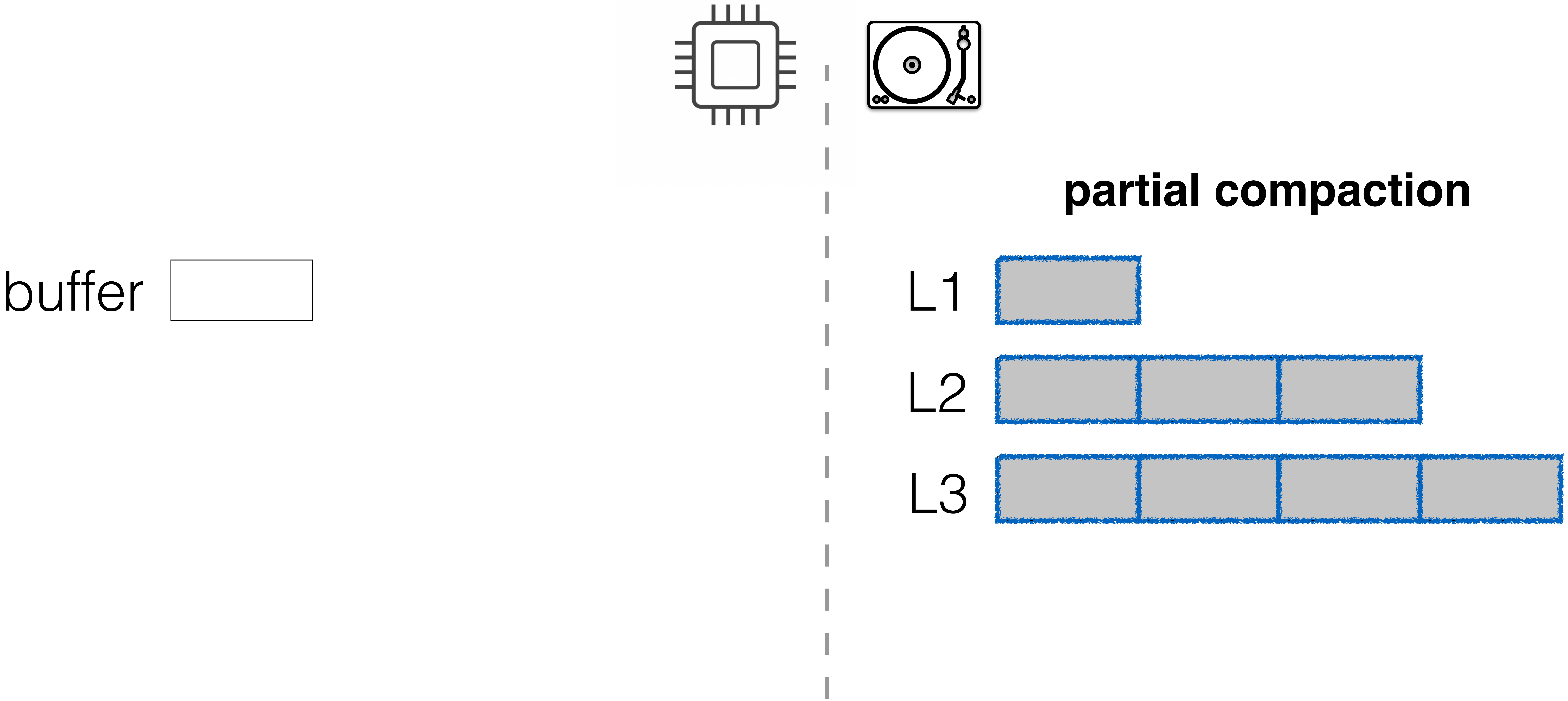
log-structured merge-tree



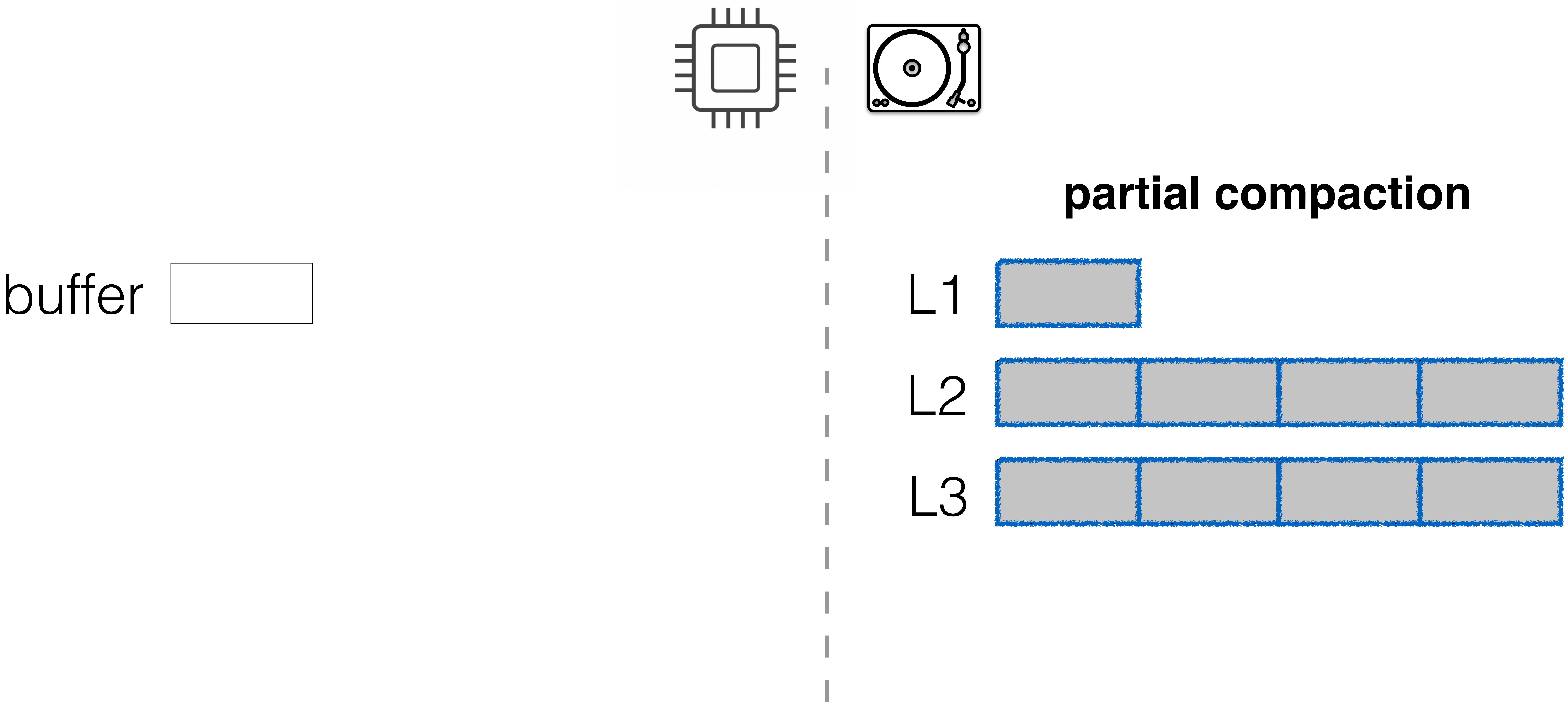
log-structured merge-tree



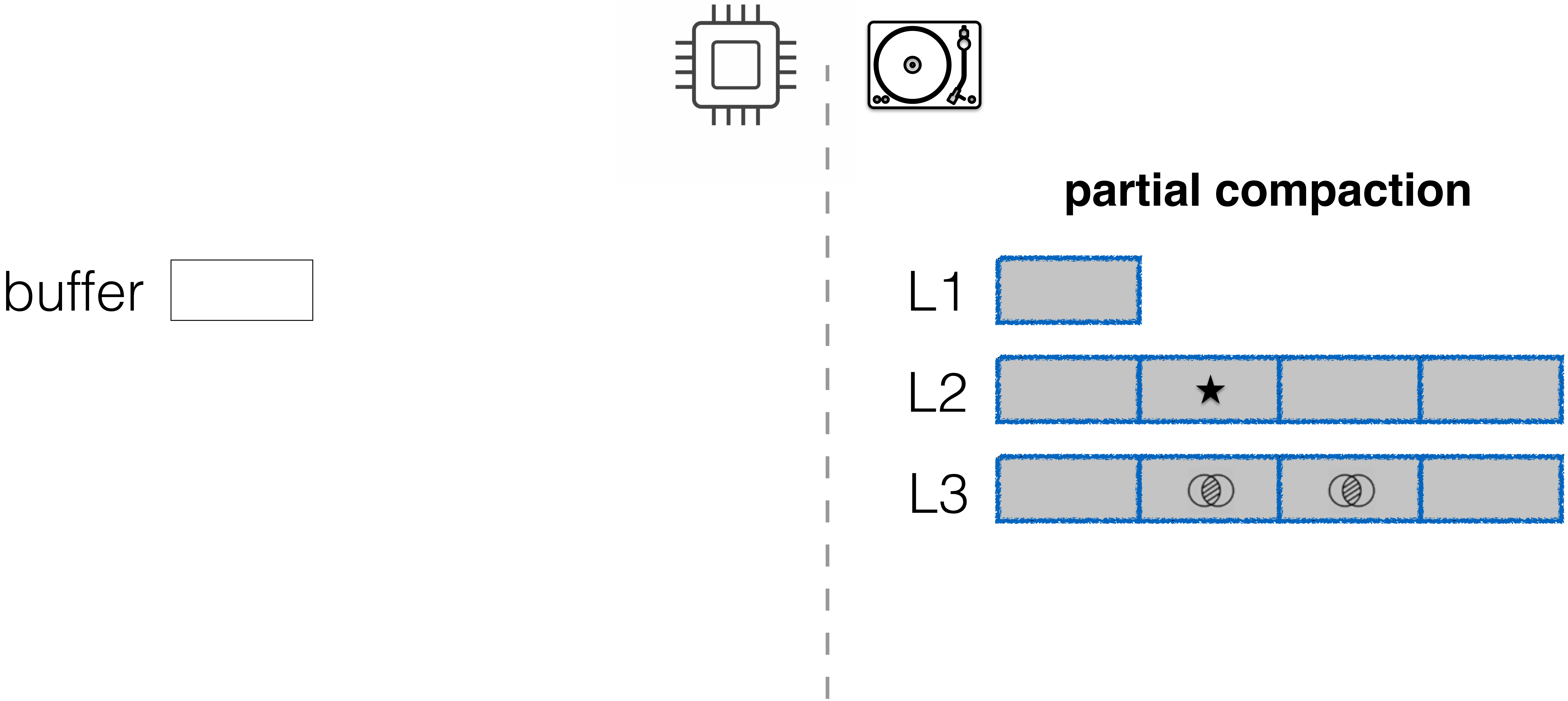
log-structured merge-tree



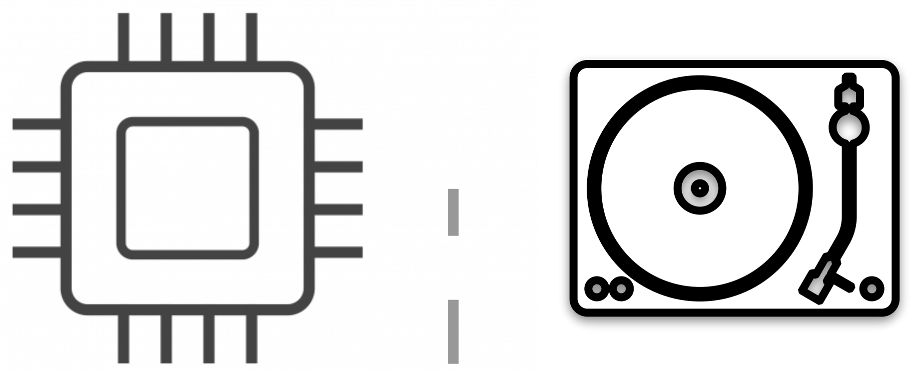
log-structured merge-tree



log-structured merge-tree

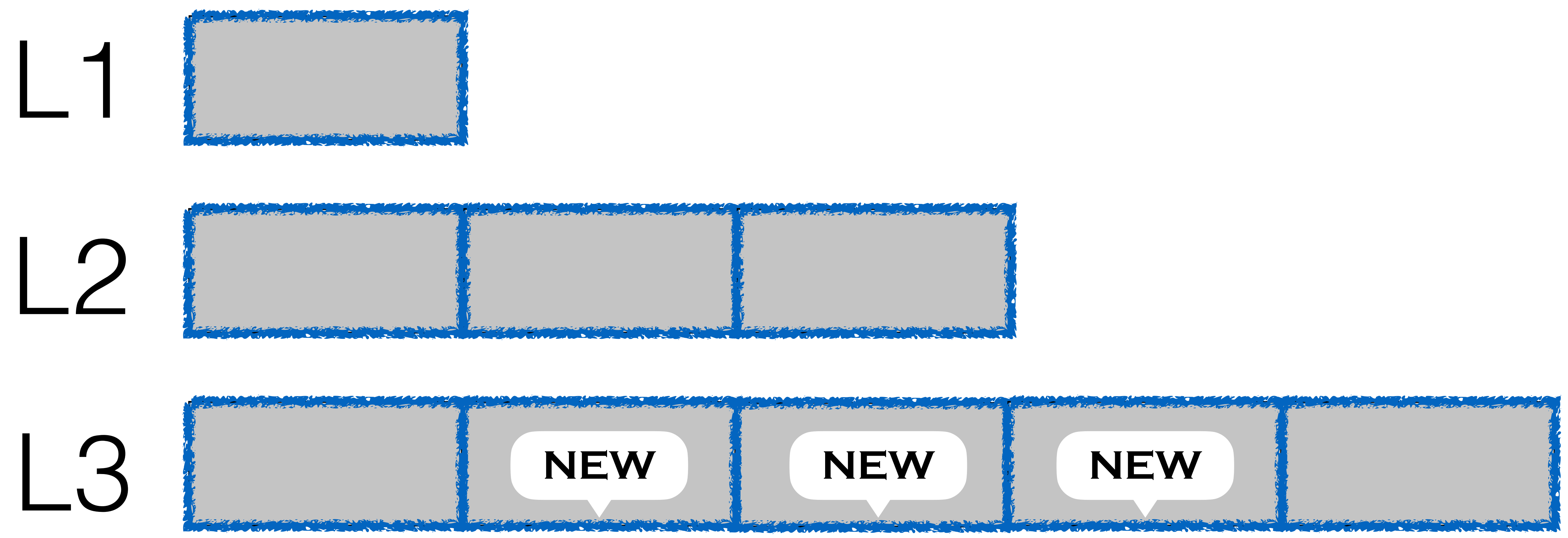


log-structured merge-tree



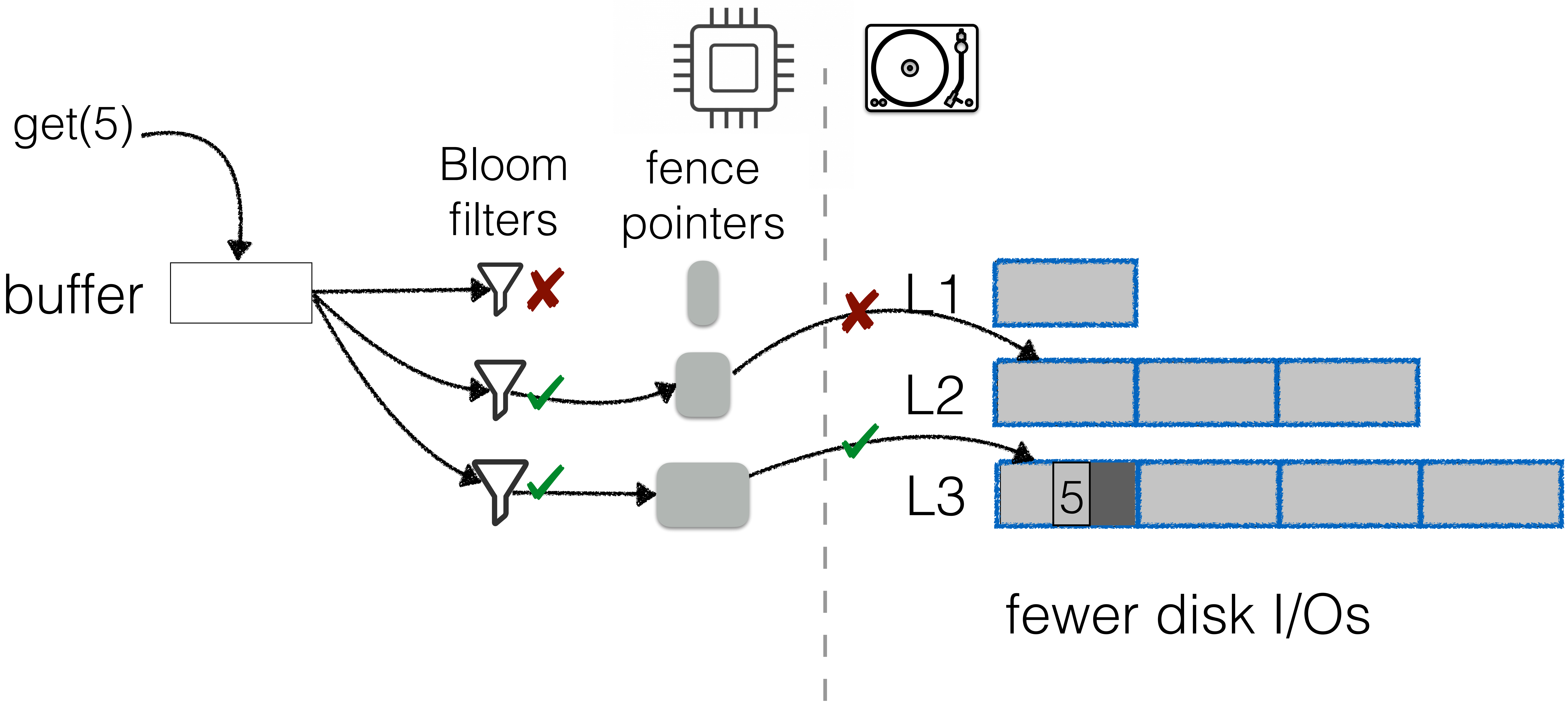
partial compaction

buffer



amortized compaction cost

log-structured merge-tree

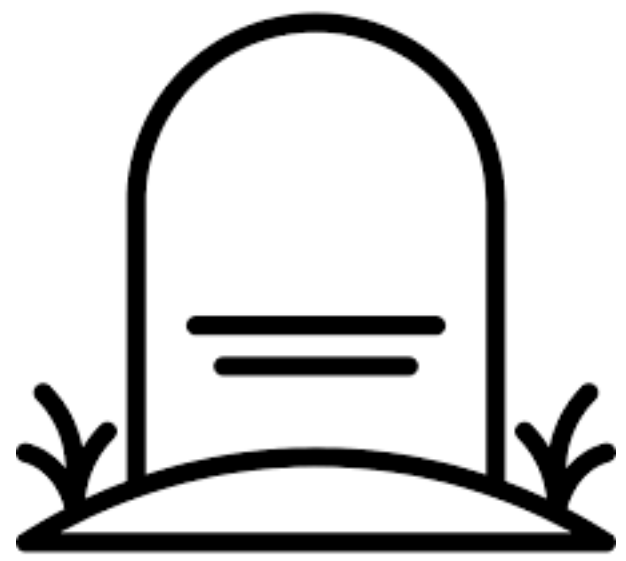


fewer disk I/Os

Now, let's talk about deletes!

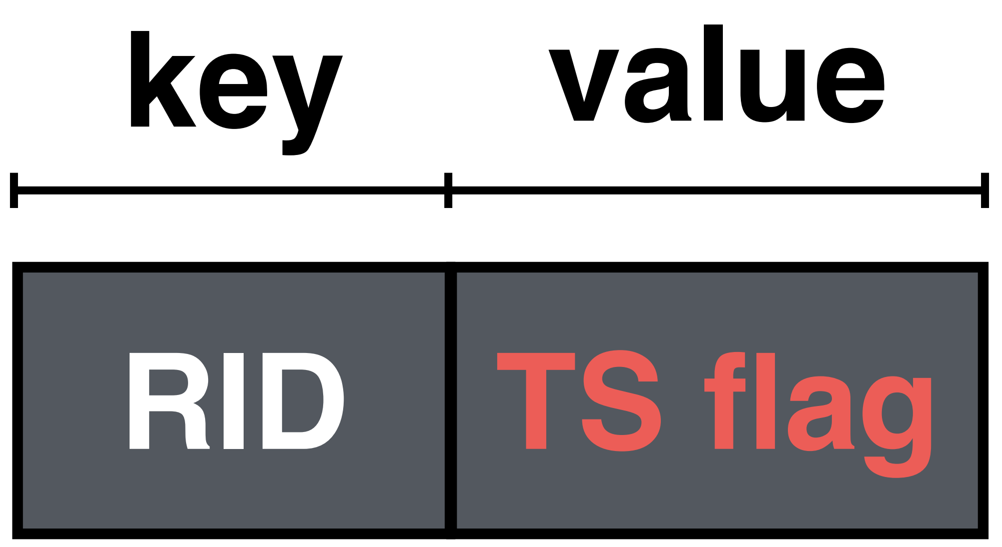
deletes in LSM-tree

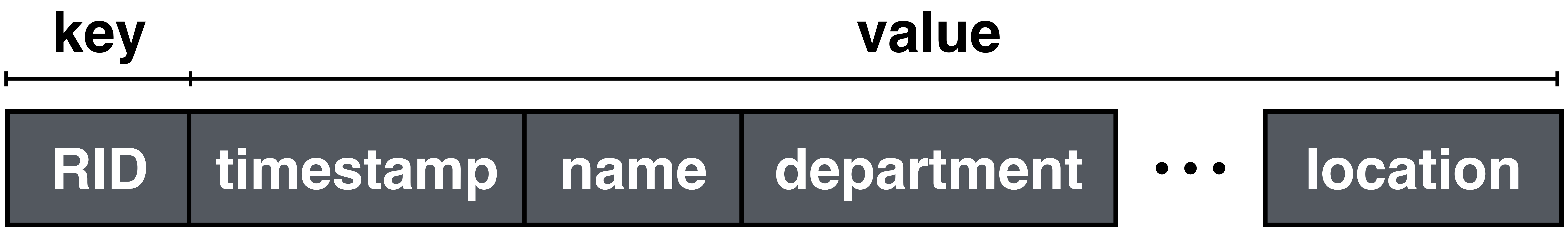
delete := insert tombstone



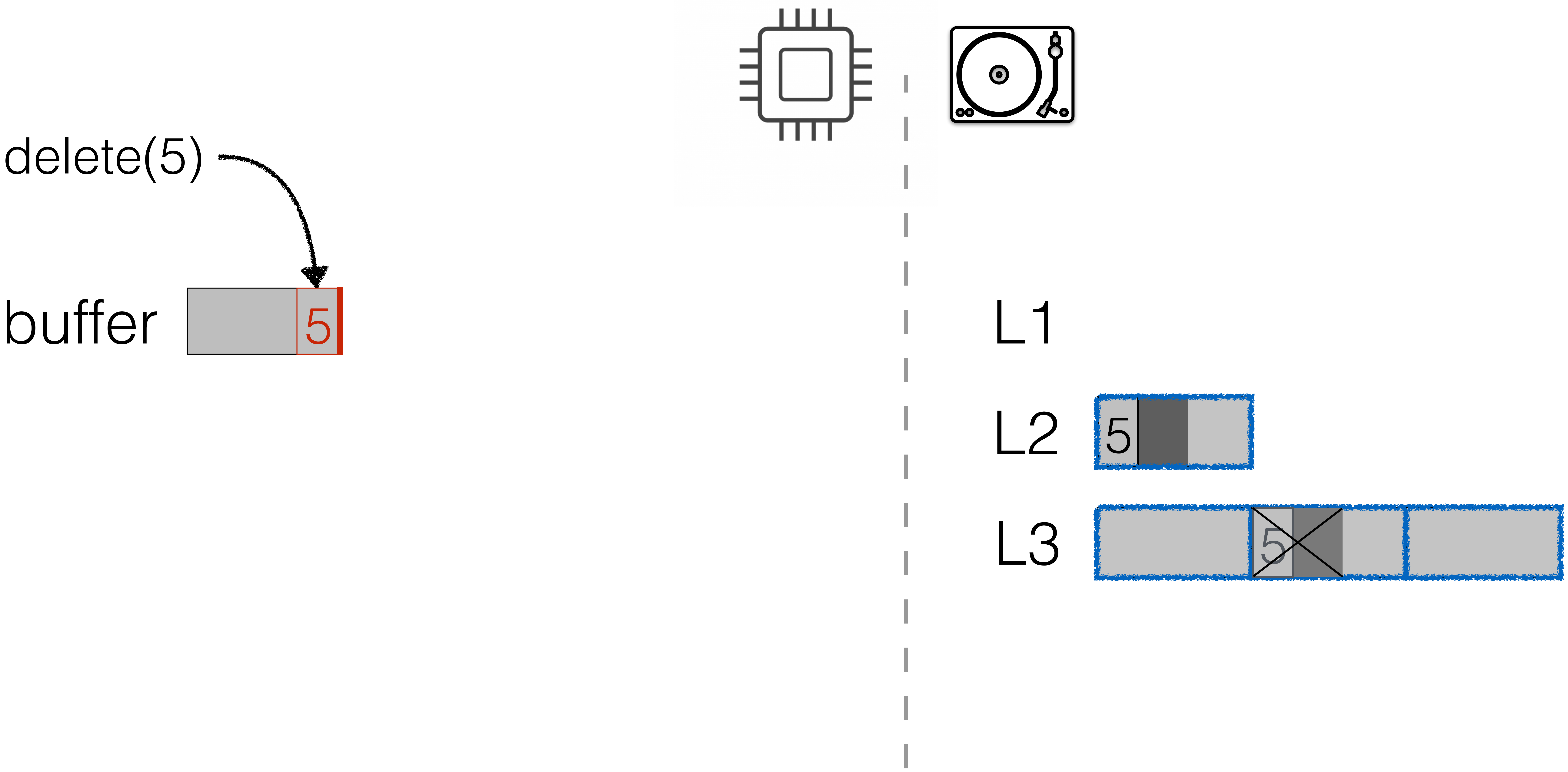
deletes in LSM-tree

delete := insert tombstone

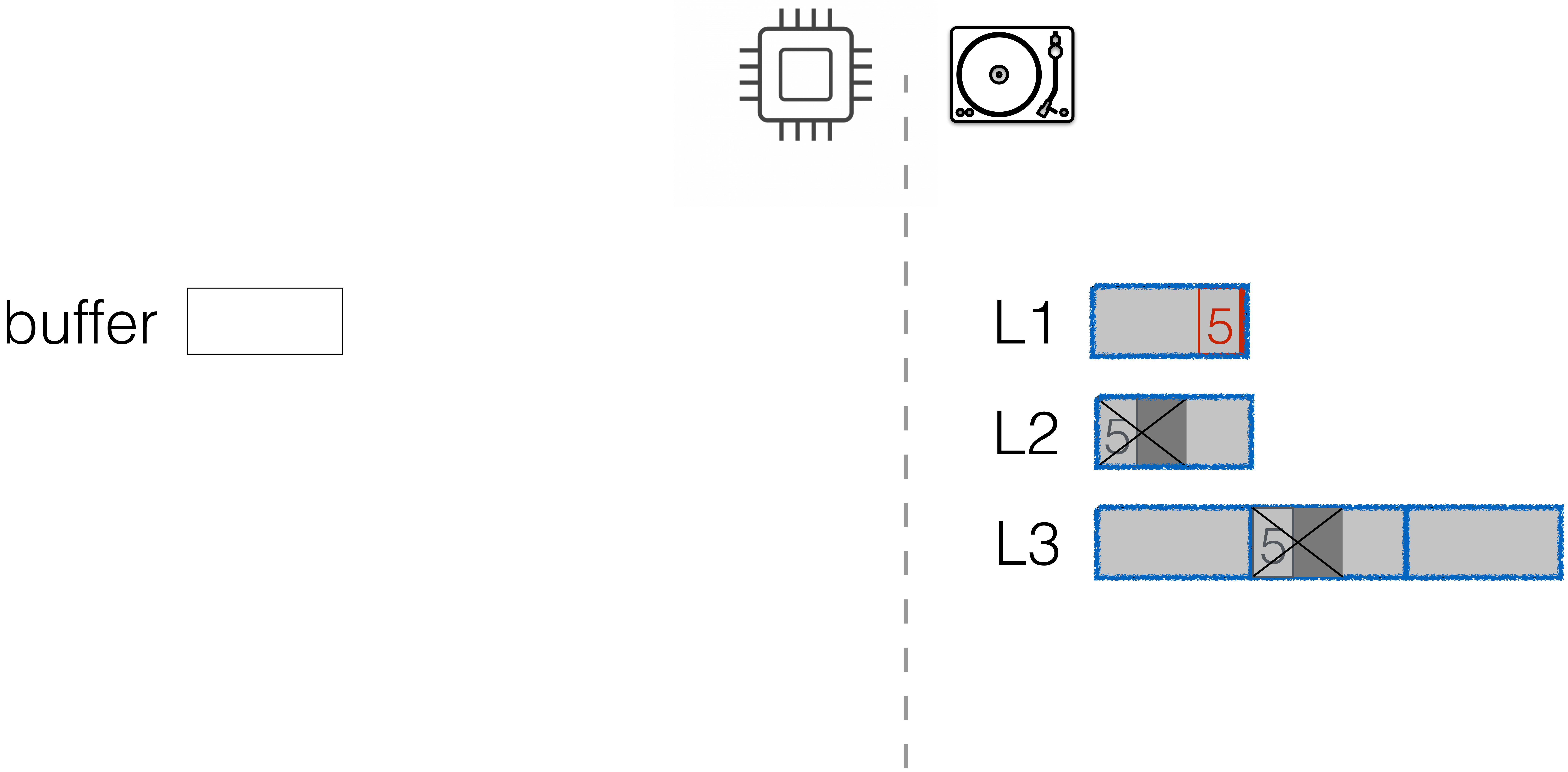




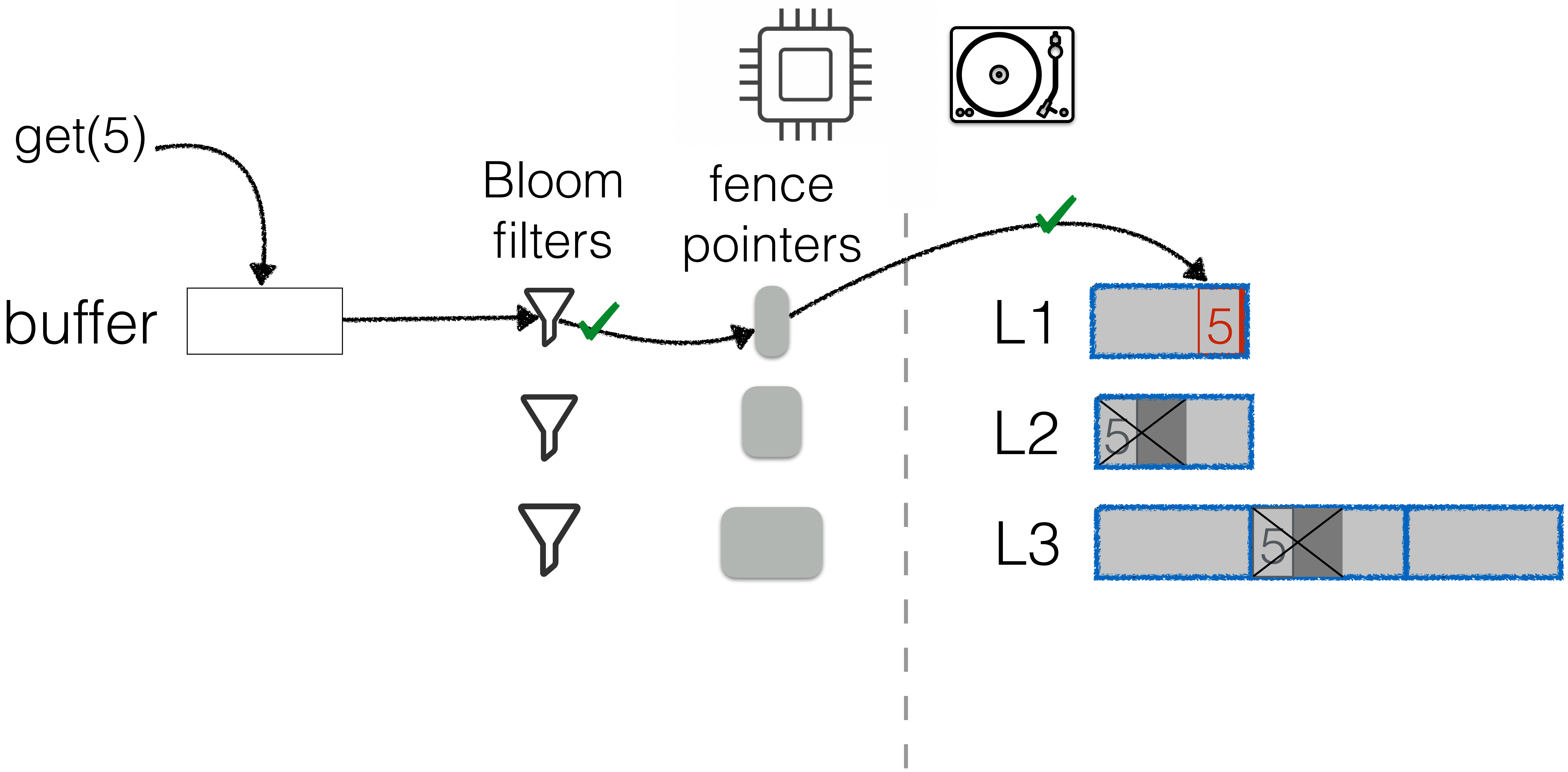
deletes in LSM-tree



deletes in LSM-tree



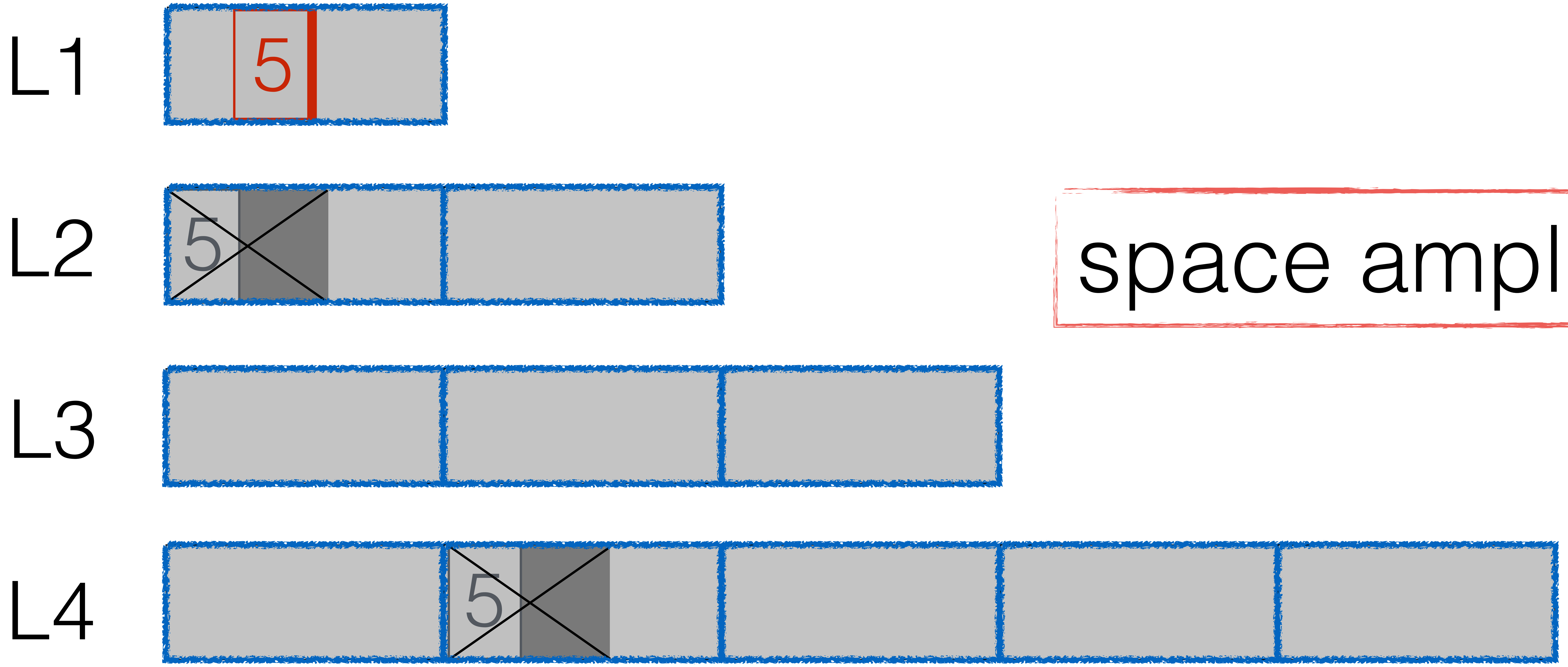
deletes in LSM-tree



the problems

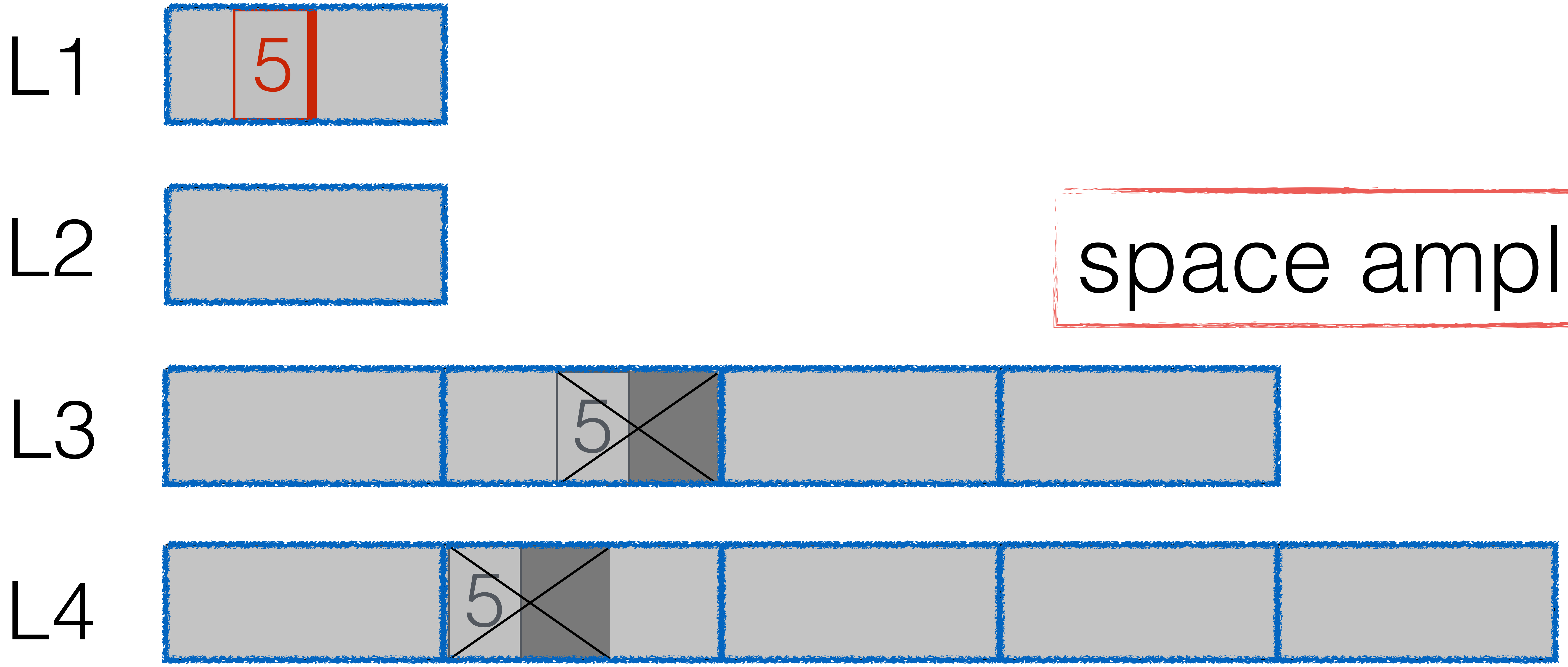
out-of-place deletes

out-of-place deletes



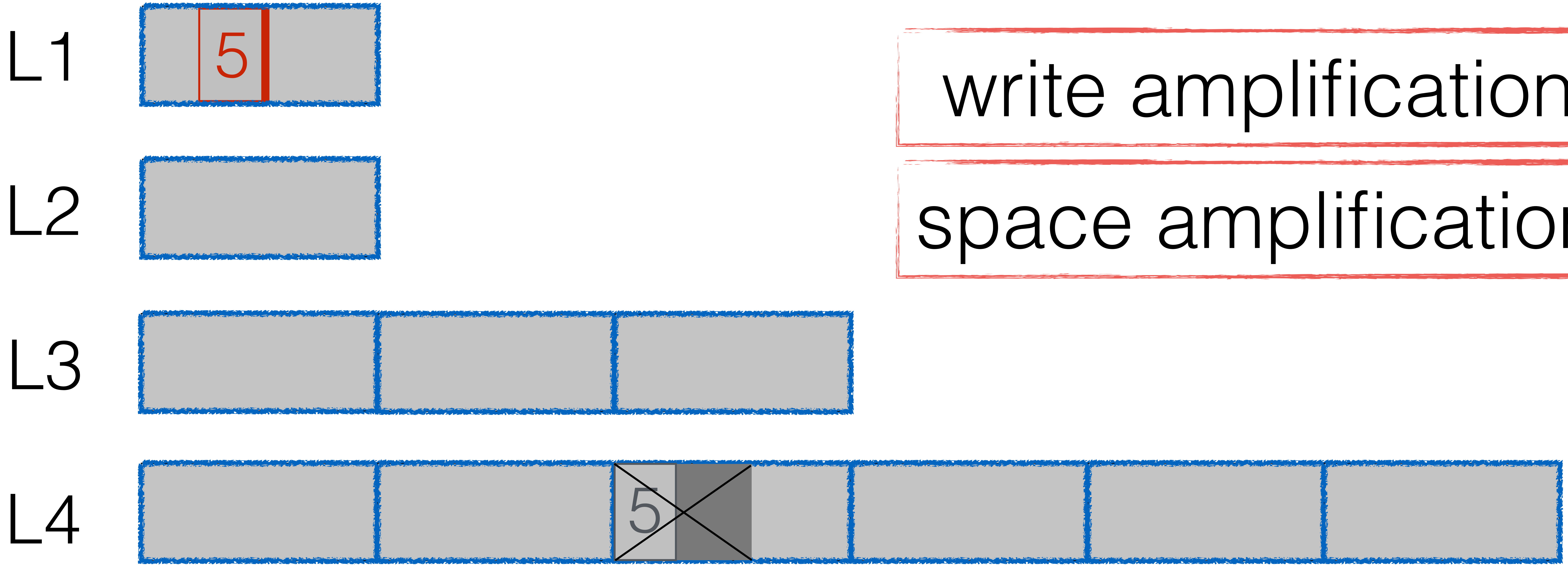
space amplification ✗

out-of-place deletes



space amplification ✗

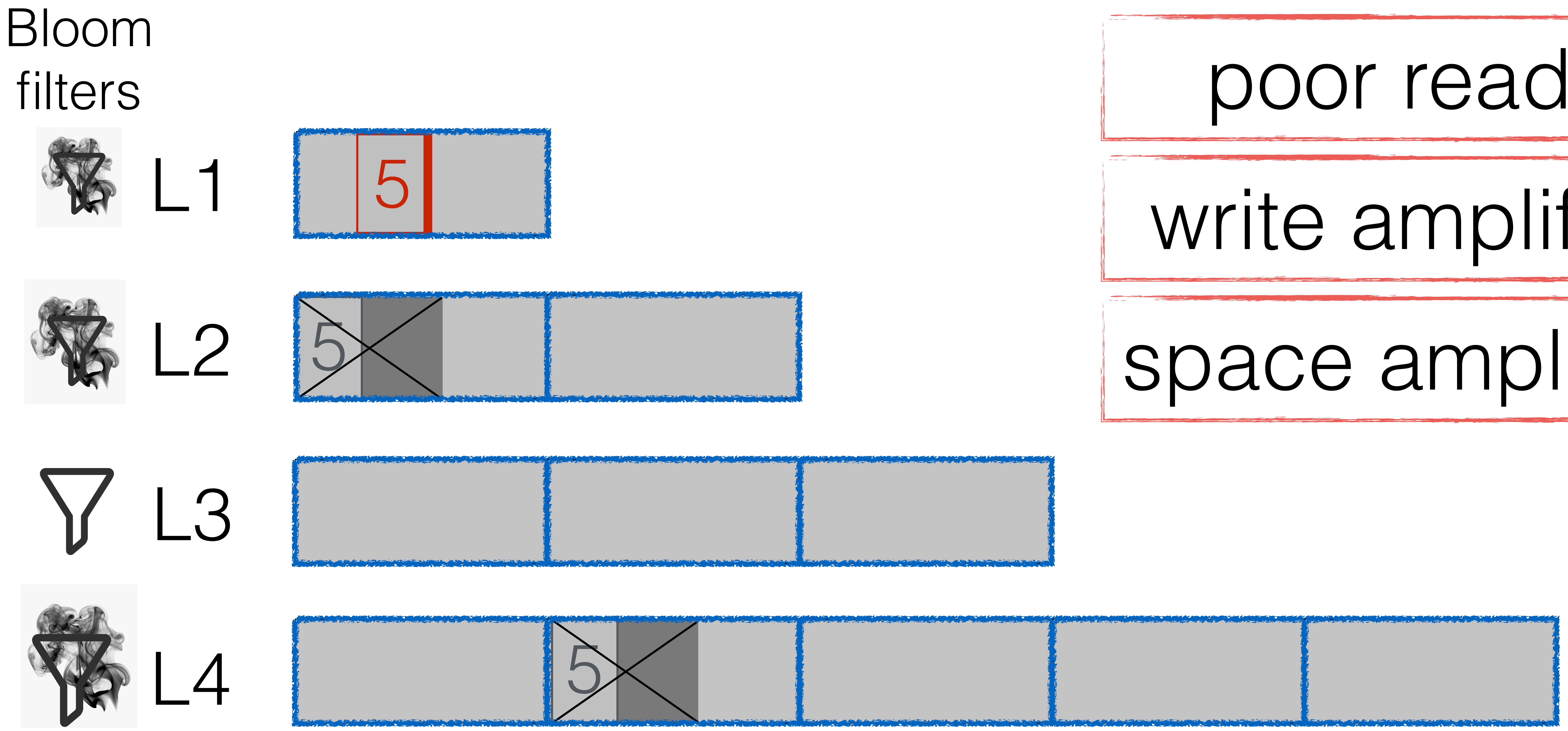
out-of-place deletes



write amplification

space amplification

out-of-place deletes



poor read perf. ✗

write amplification ✗

space amplification ✗

the problems

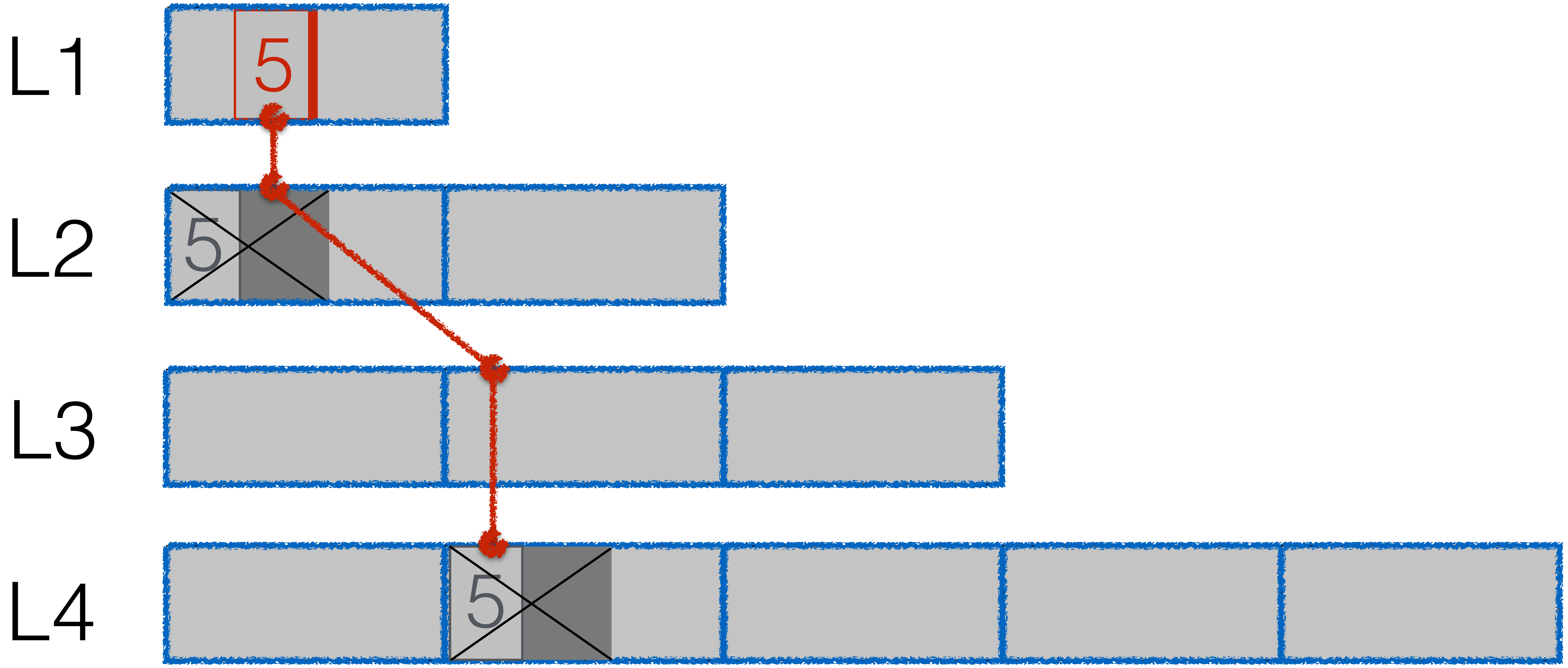
poor read perf.

write amplification

space amplification

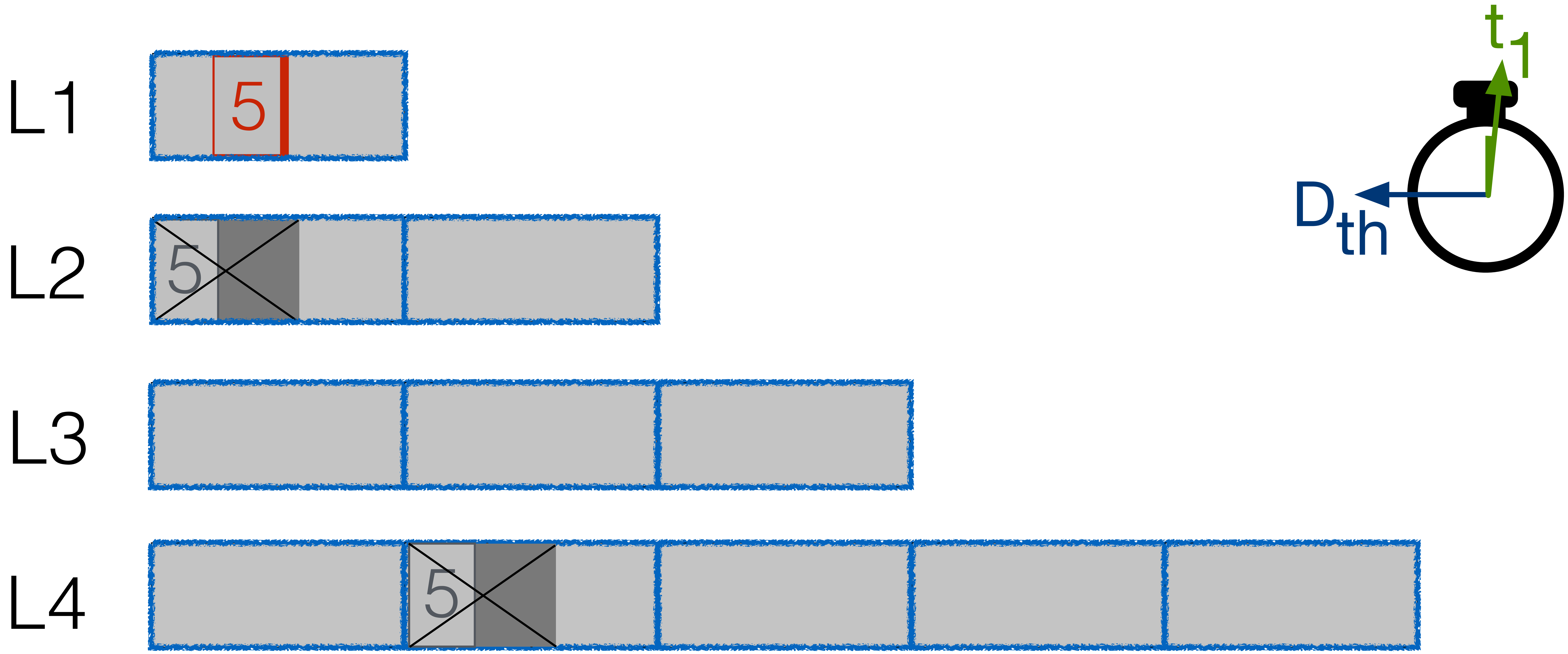
delete persistence latency

delete persistence latency



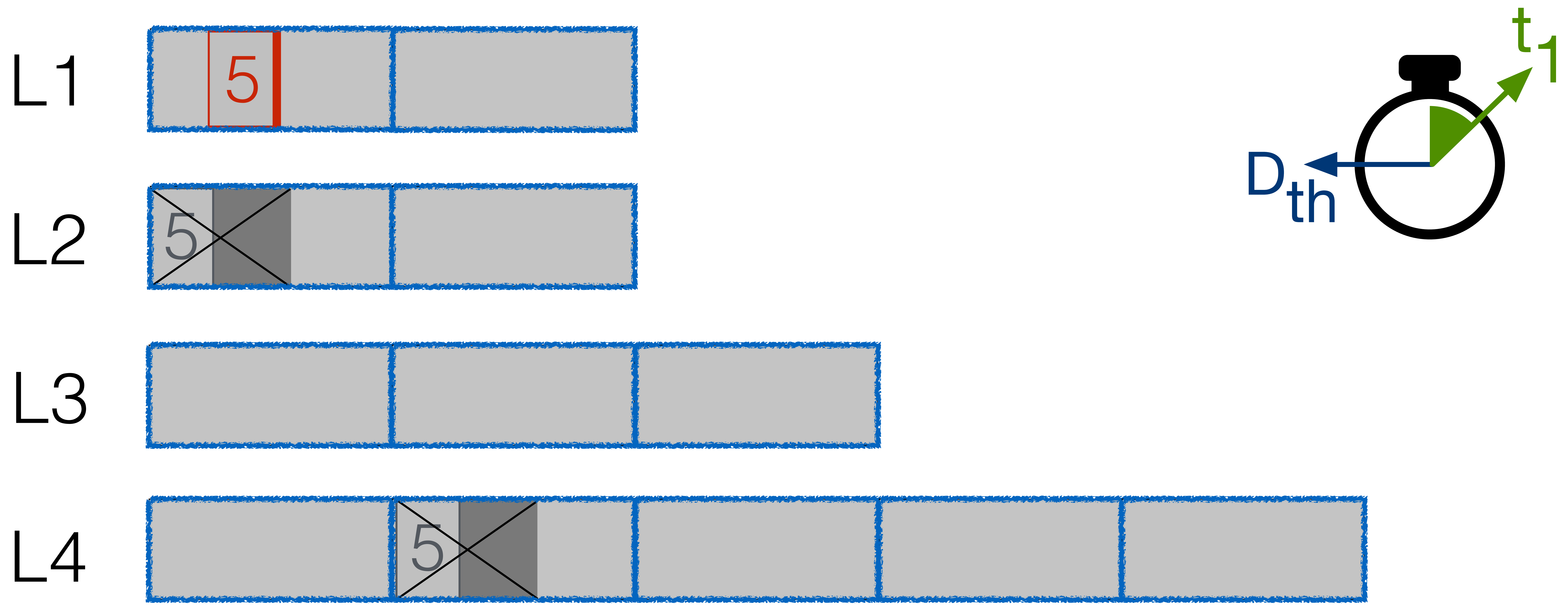
delete persistence latency

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



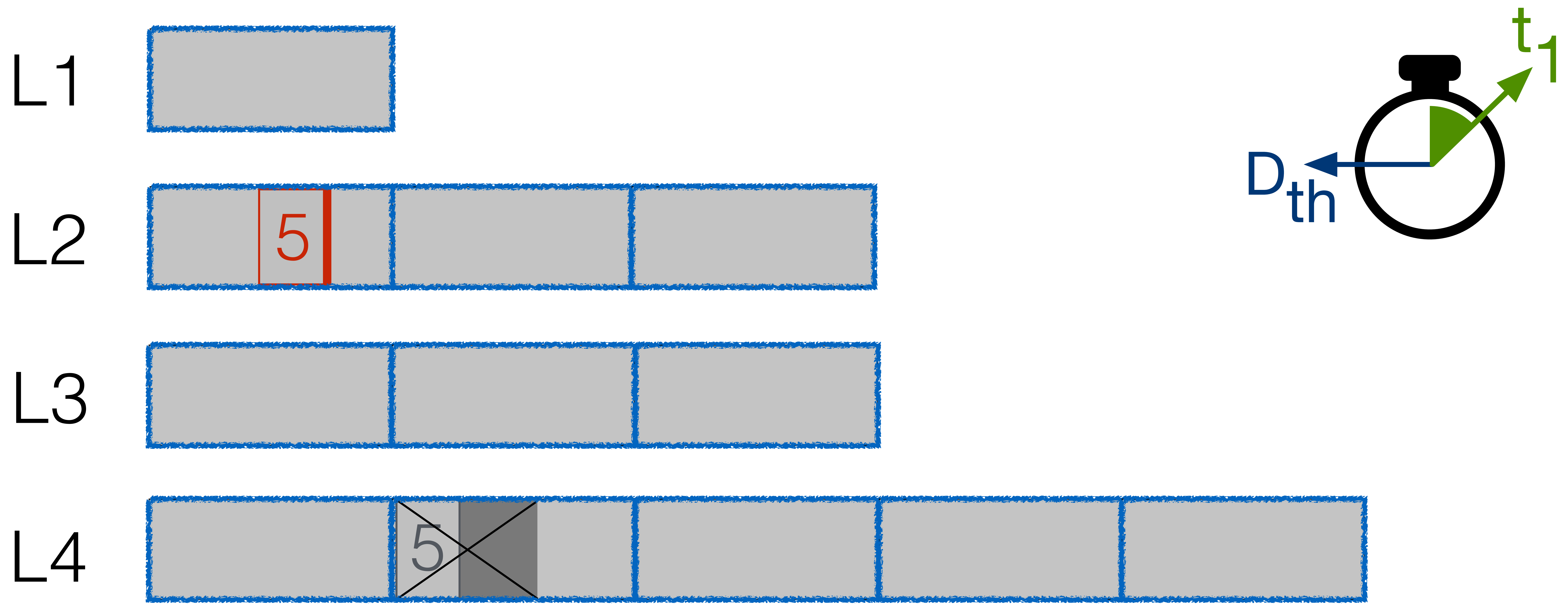
delete persistence latency

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



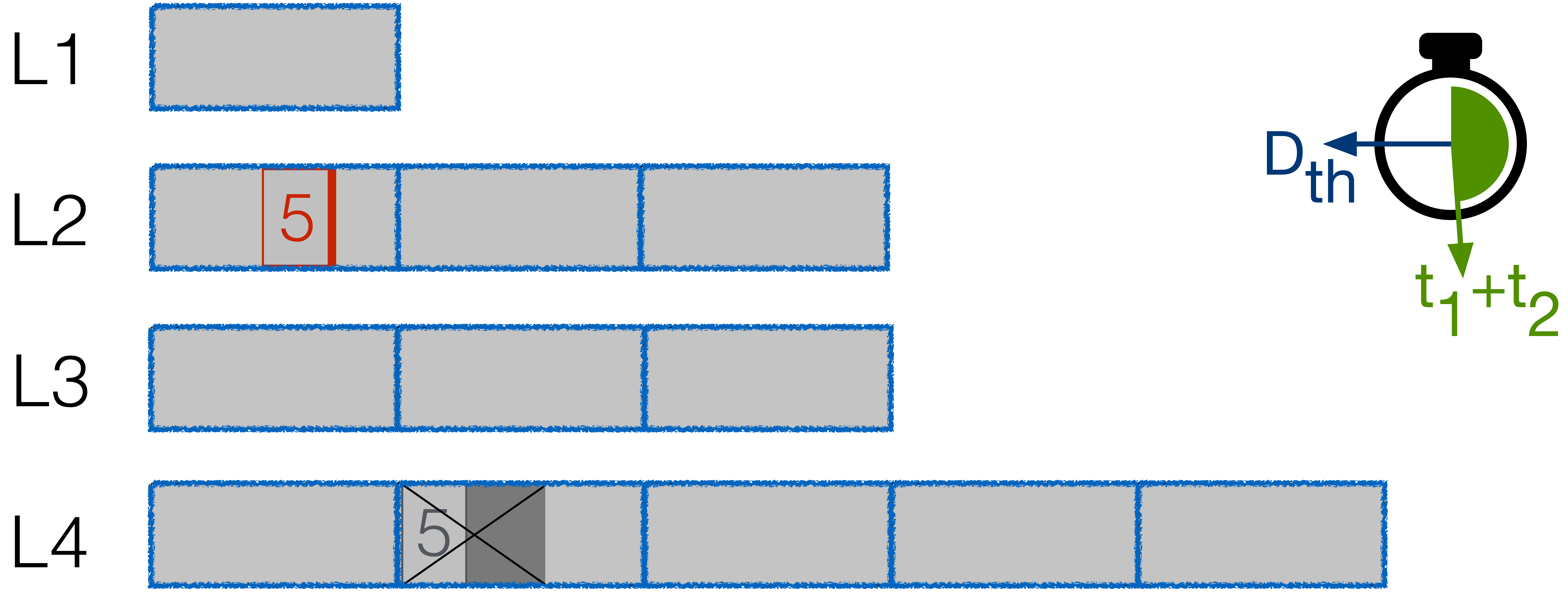
delete persistence latency

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



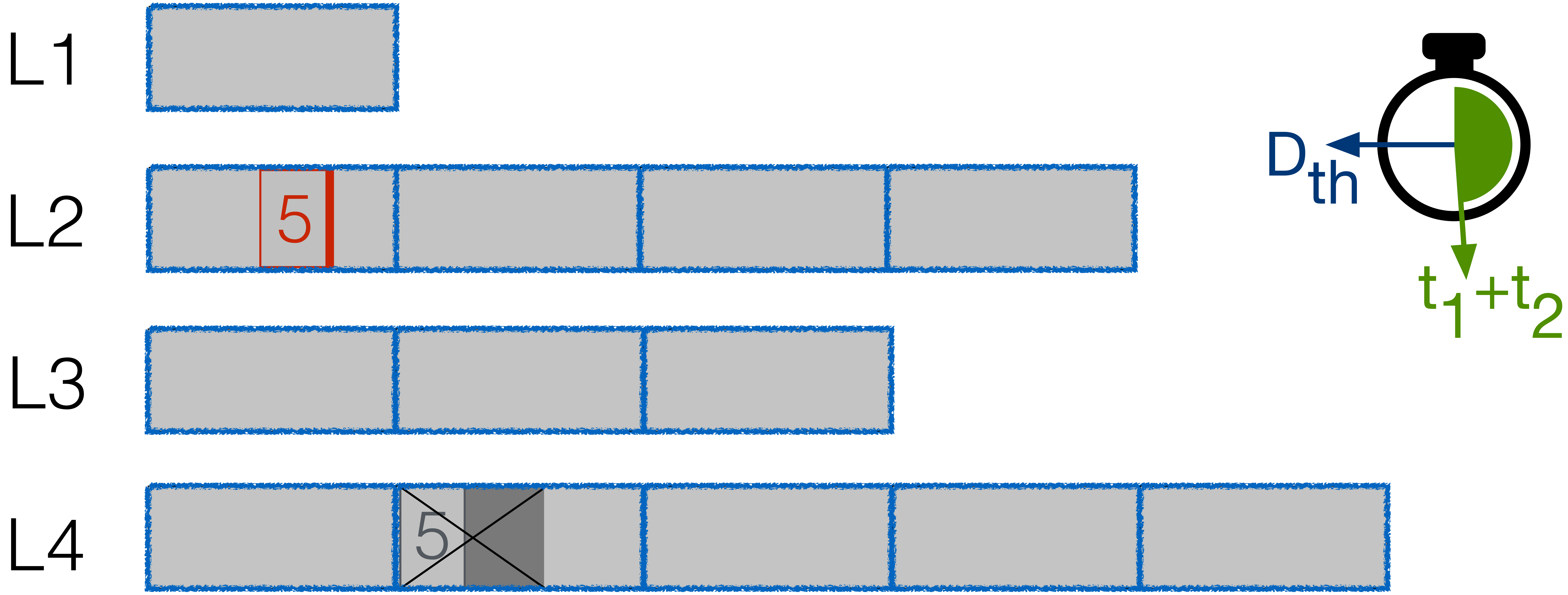
delete persistence latency

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



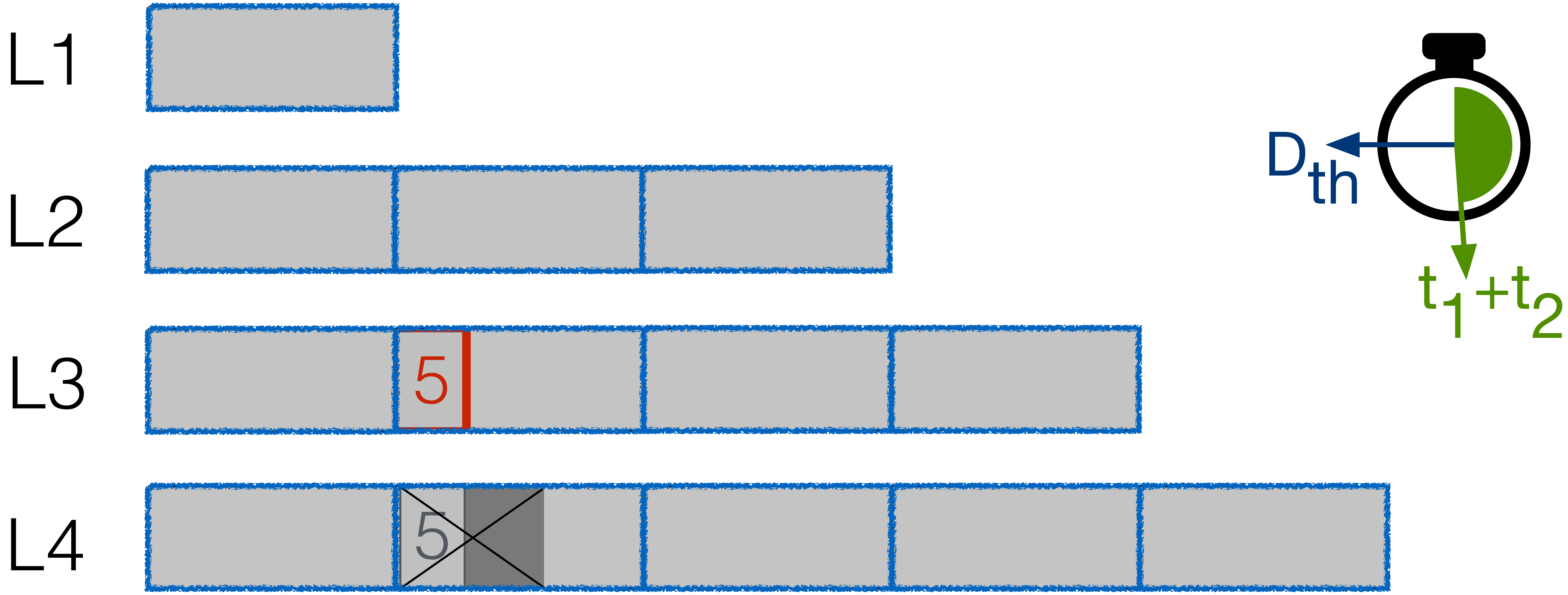
delete persistence latency

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



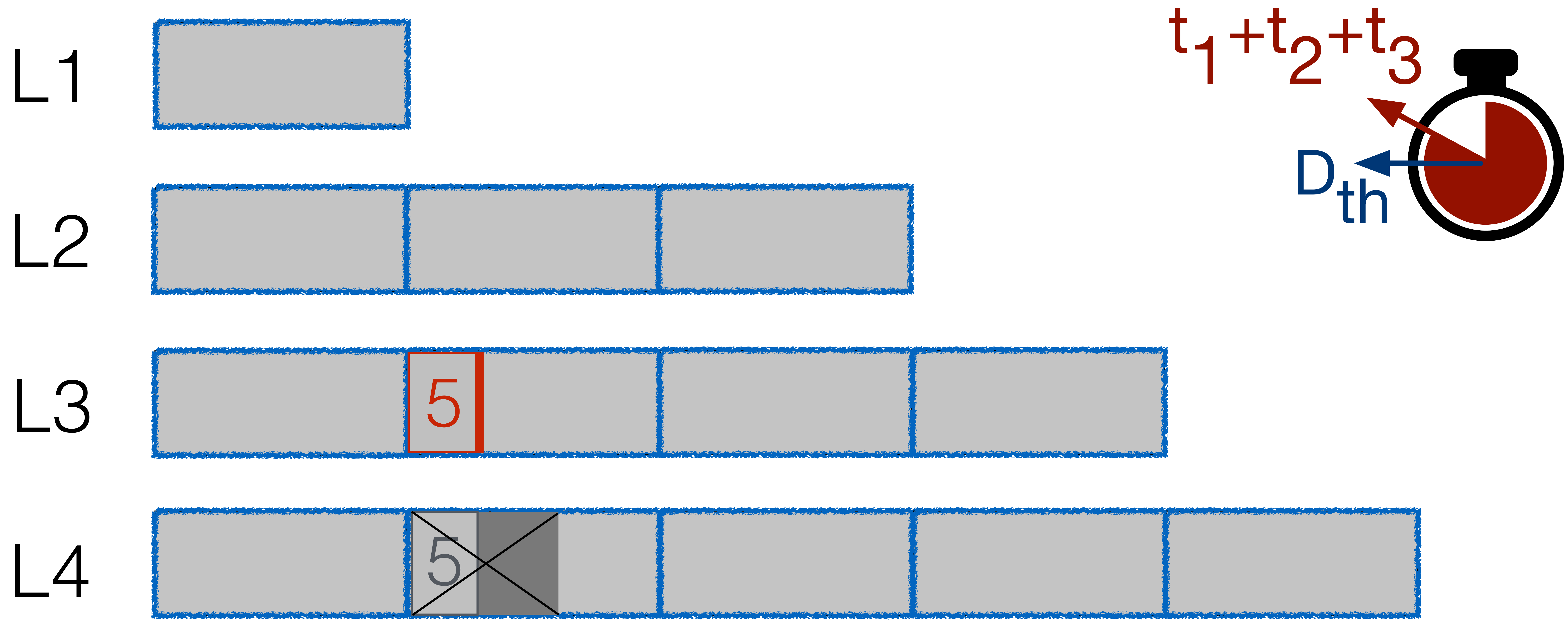
delete persistence latency

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



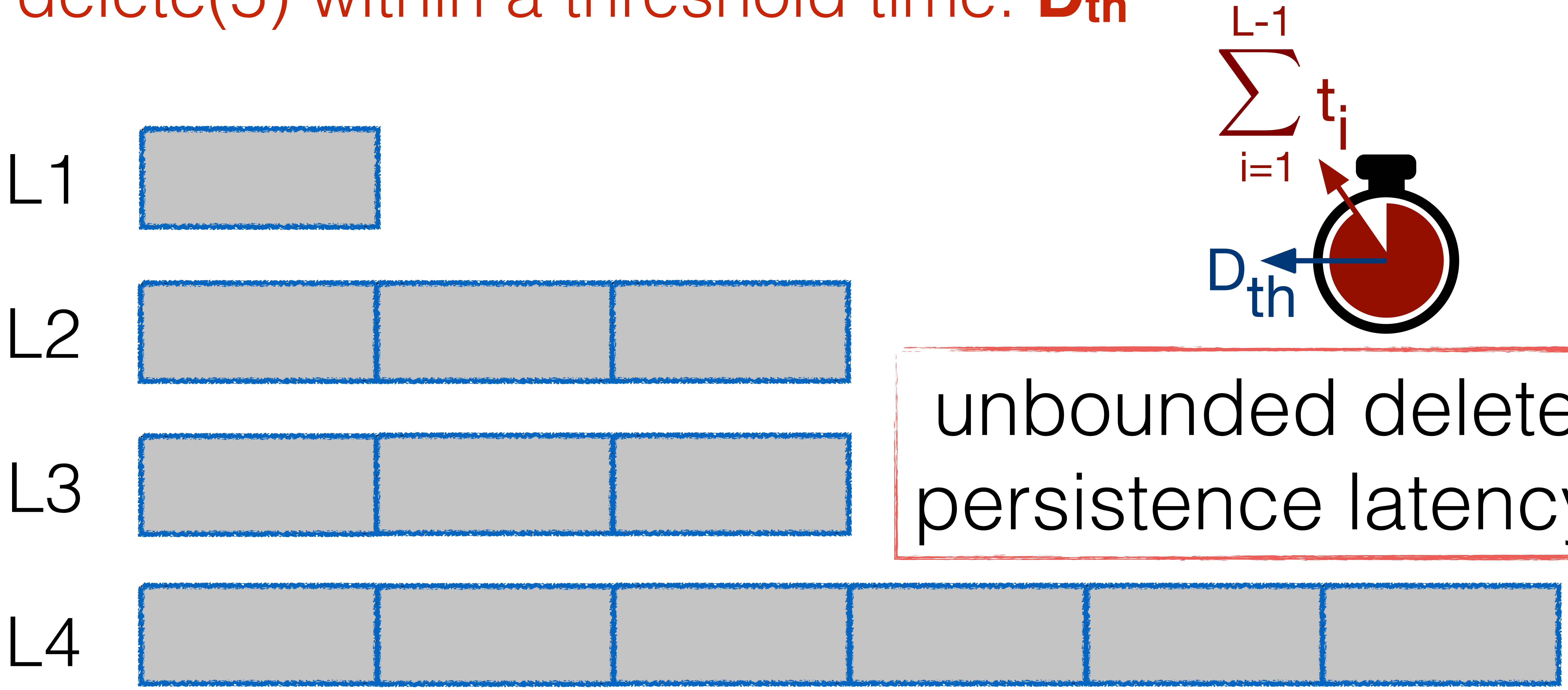
delete persistence latency

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



delete persistence latency

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



unbounded delete persistence latency

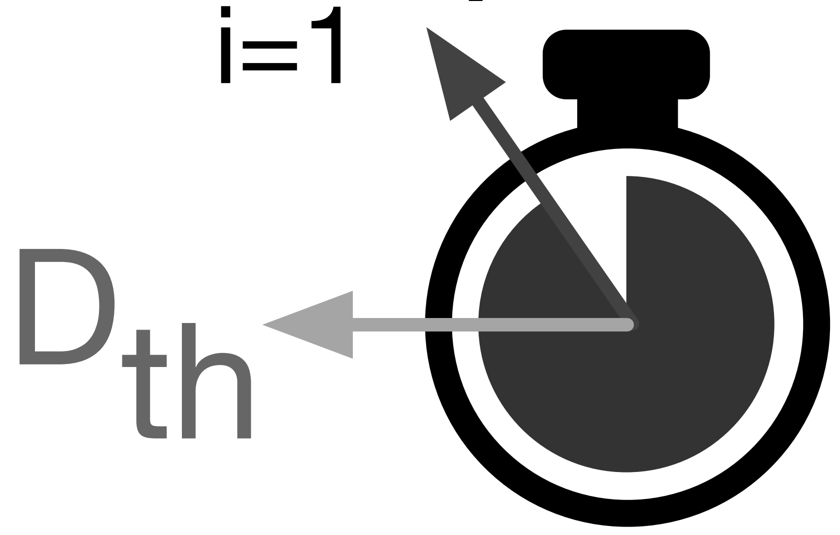
the problems

poor read perf.

write amplification

space amplification

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

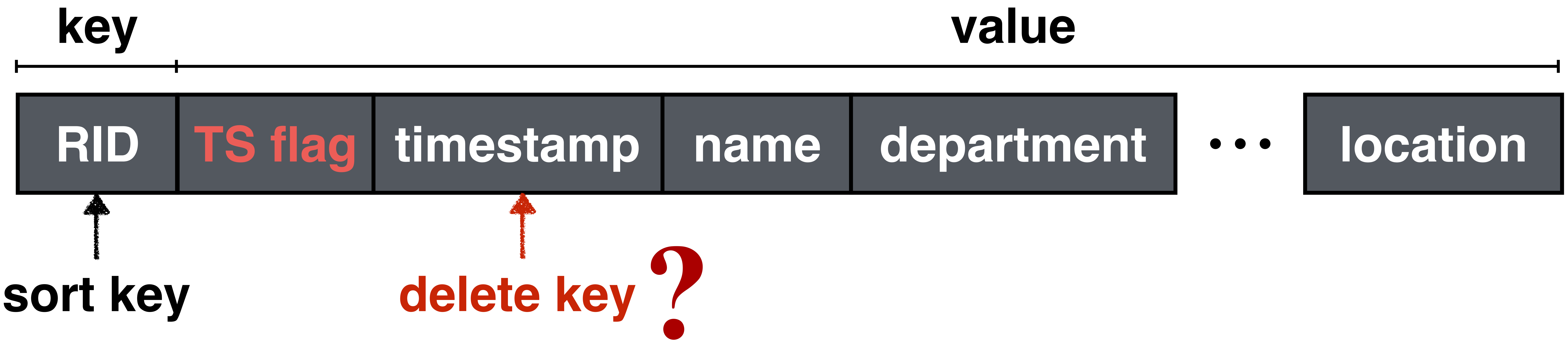


unbounded delete persistence latency

deletes on a secondary attribute

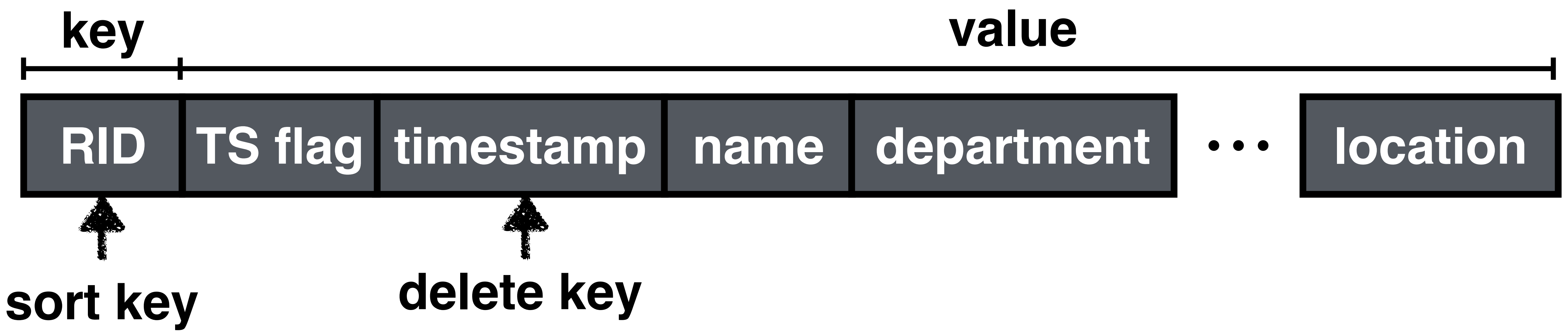
deletes on a secondary attribute

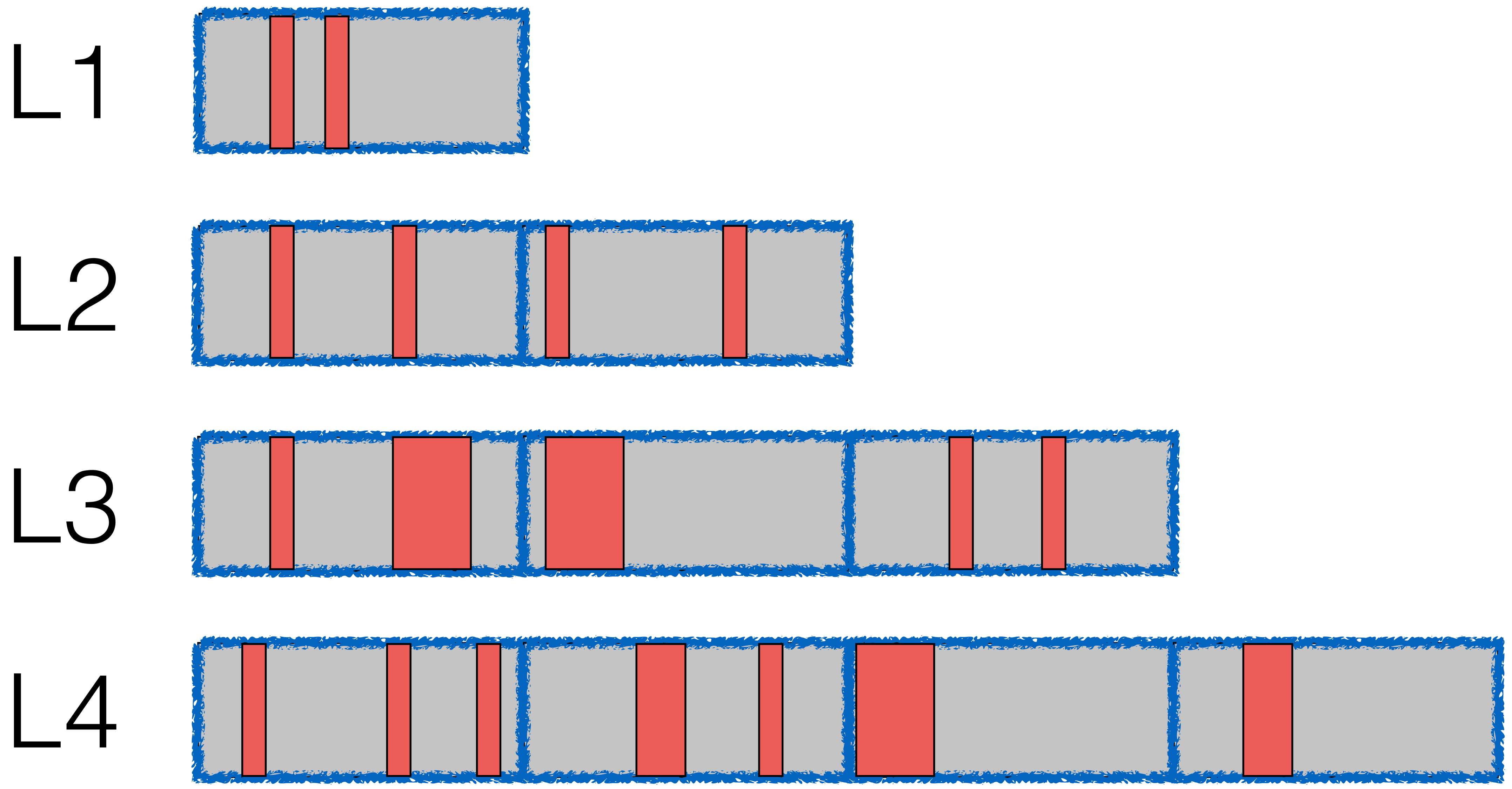
delete all entries older than: **D days**



deletes on a secondary attribute

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

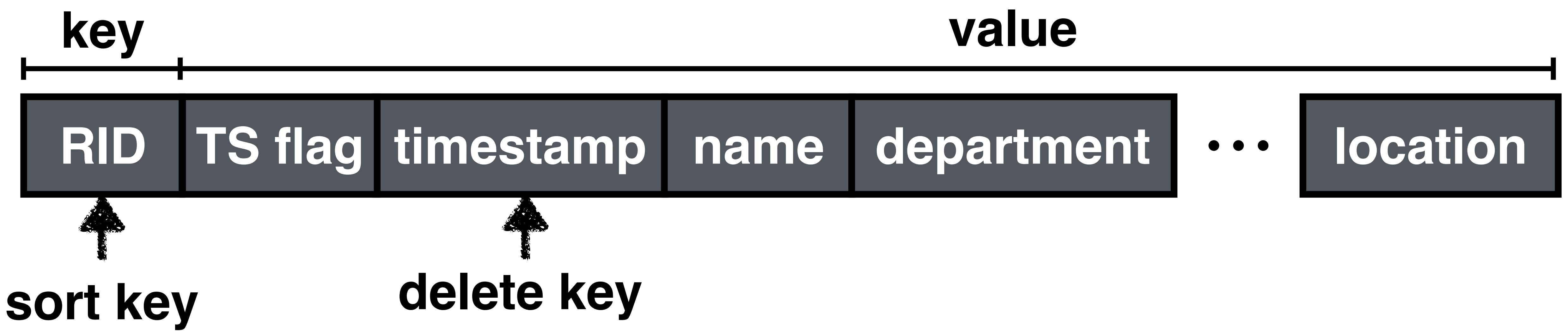


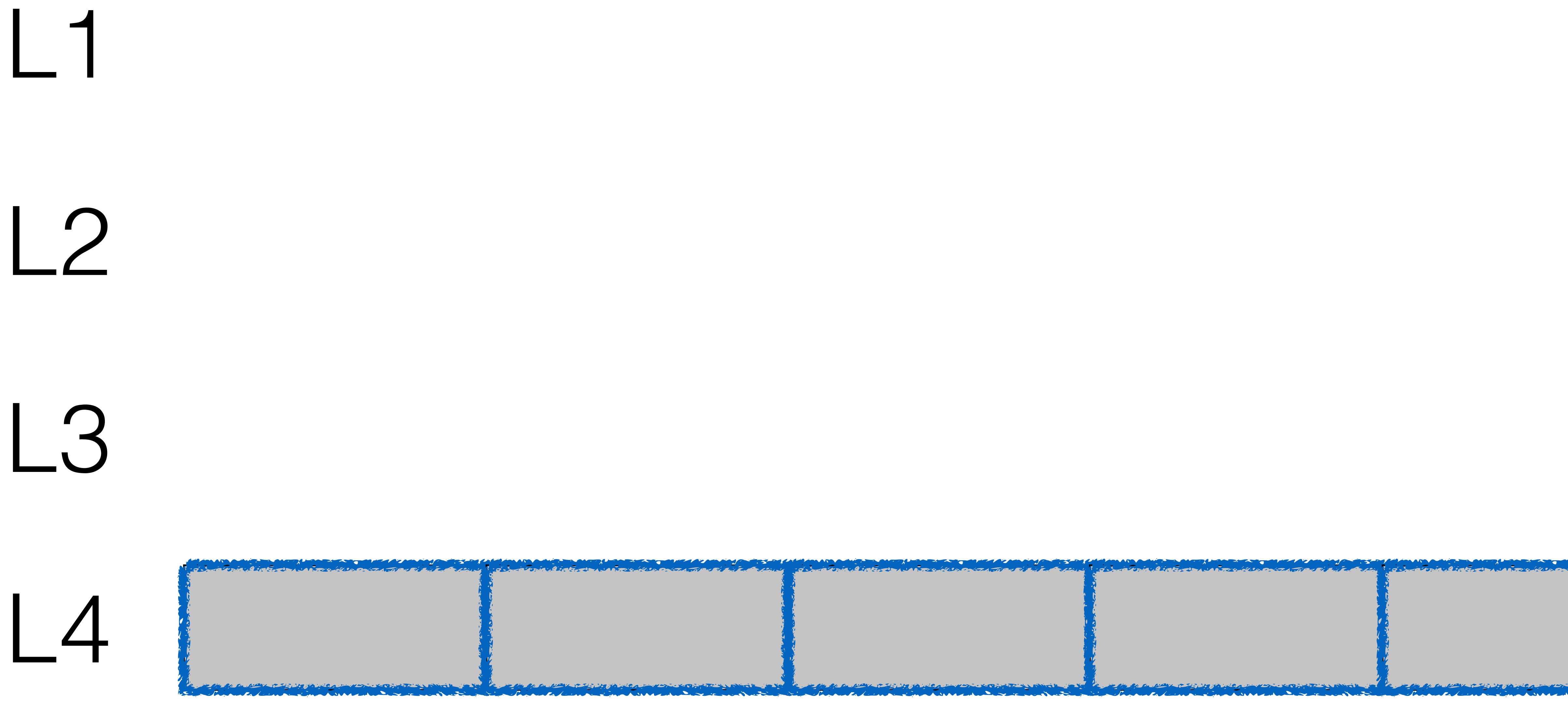


scattered occurrences

deletes on a secondary attribute

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





latency spikes ✗

superfluous I/Os ✗

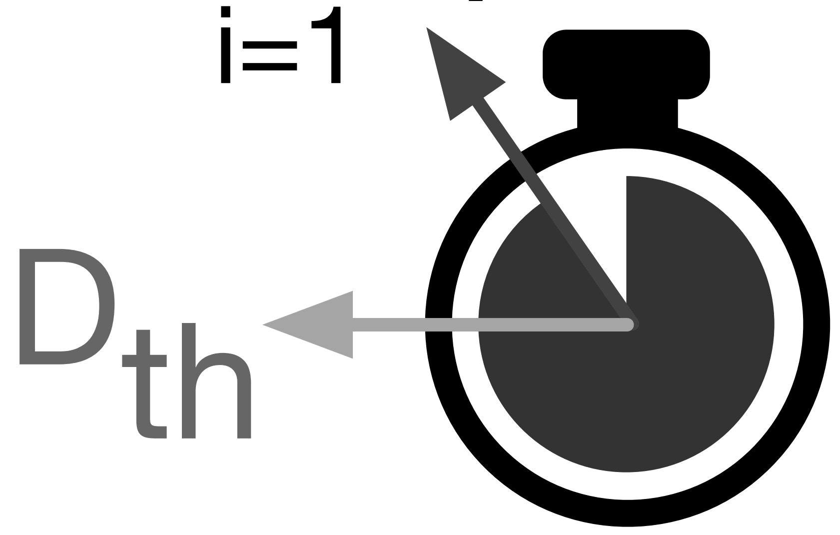
the problems

poor read perf.

write amplification

space amplification

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



unbounded delete persistence latency

latency spikes

superfluous I/Os

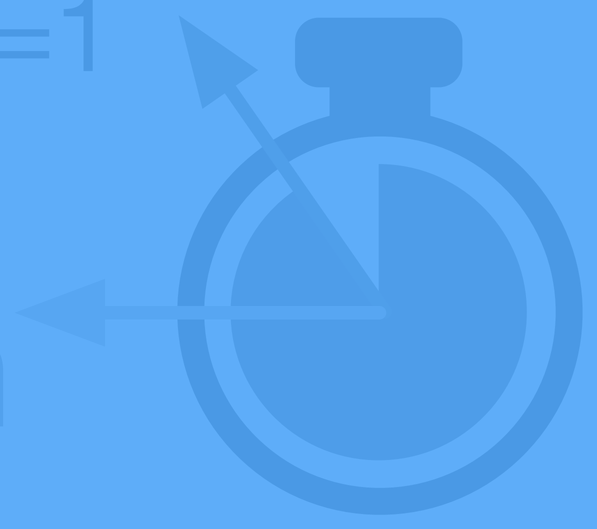
the solution

poor read perf.

write amplification

space amplification

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



D_{th}

unbounded delete persistence latency

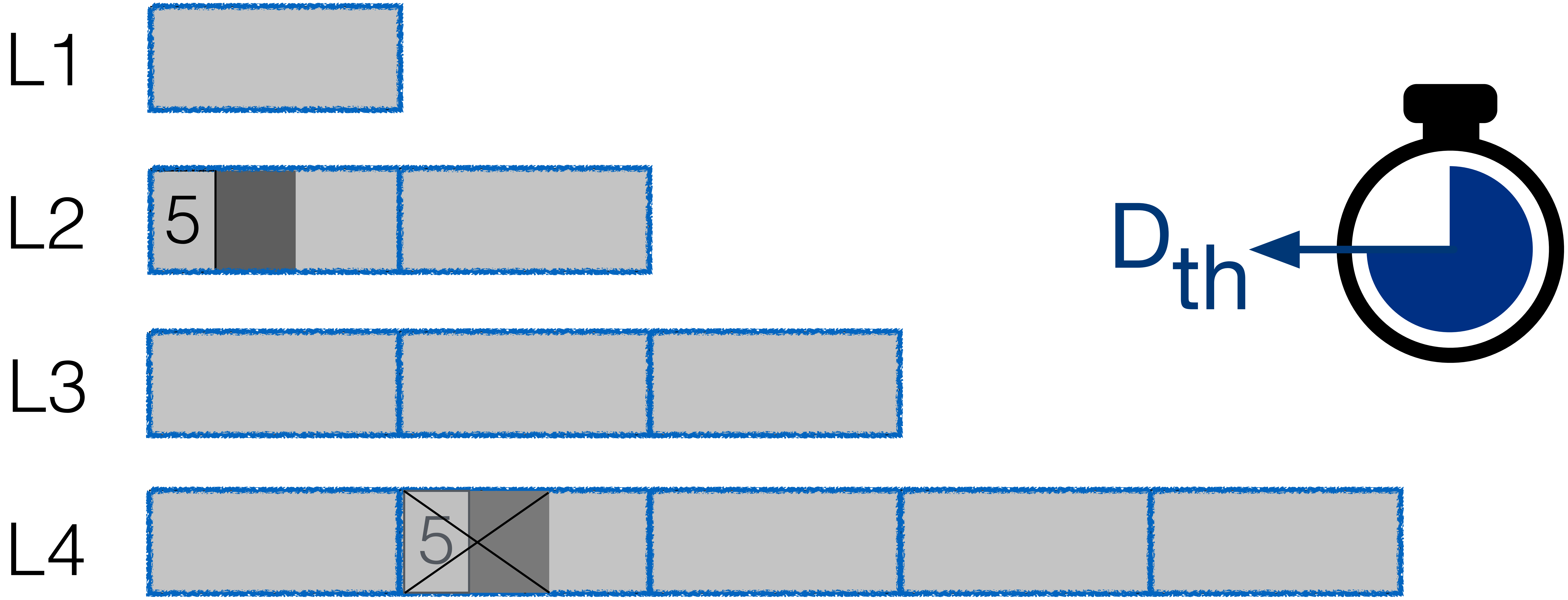
FADE

latency spikes

superfluous I/Os

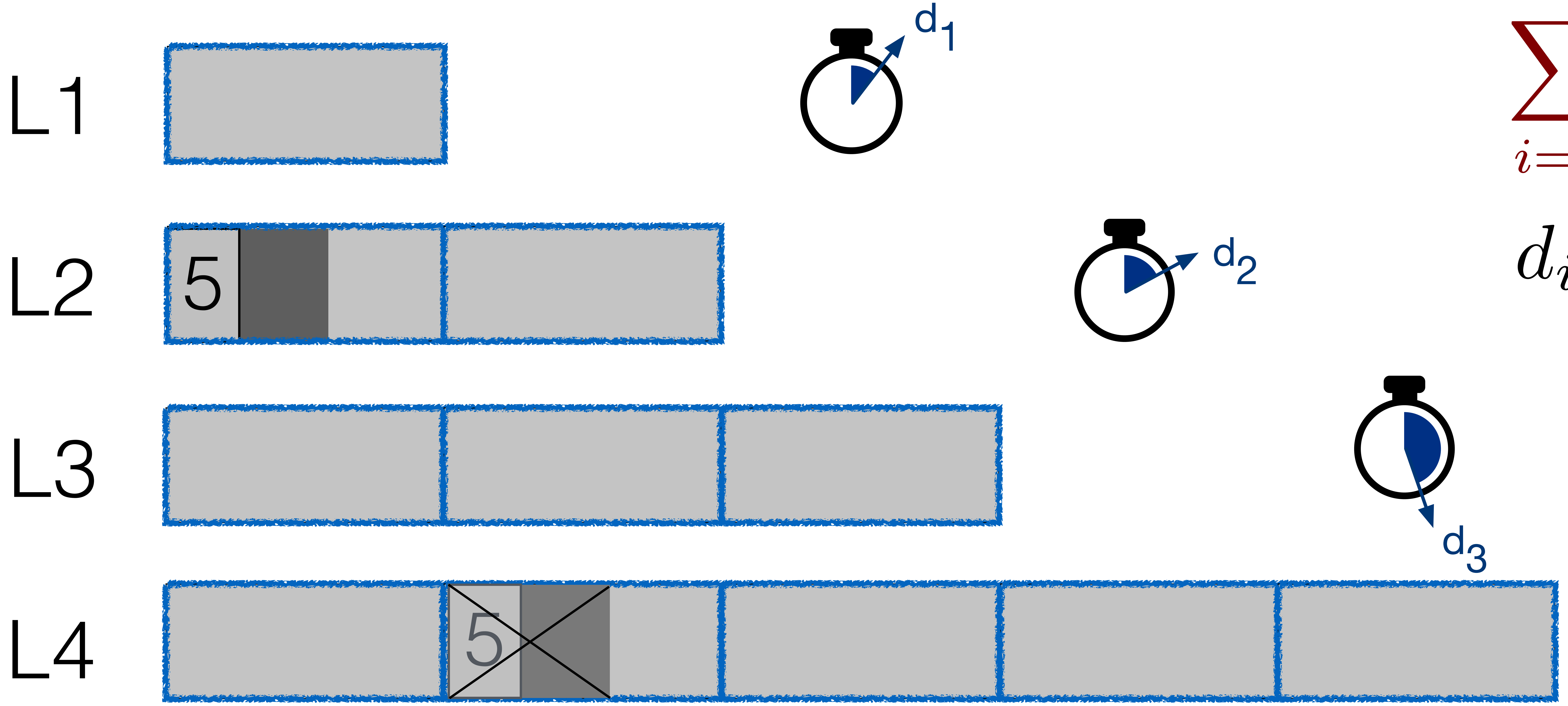
FAst DElete

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



FAst DElete

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

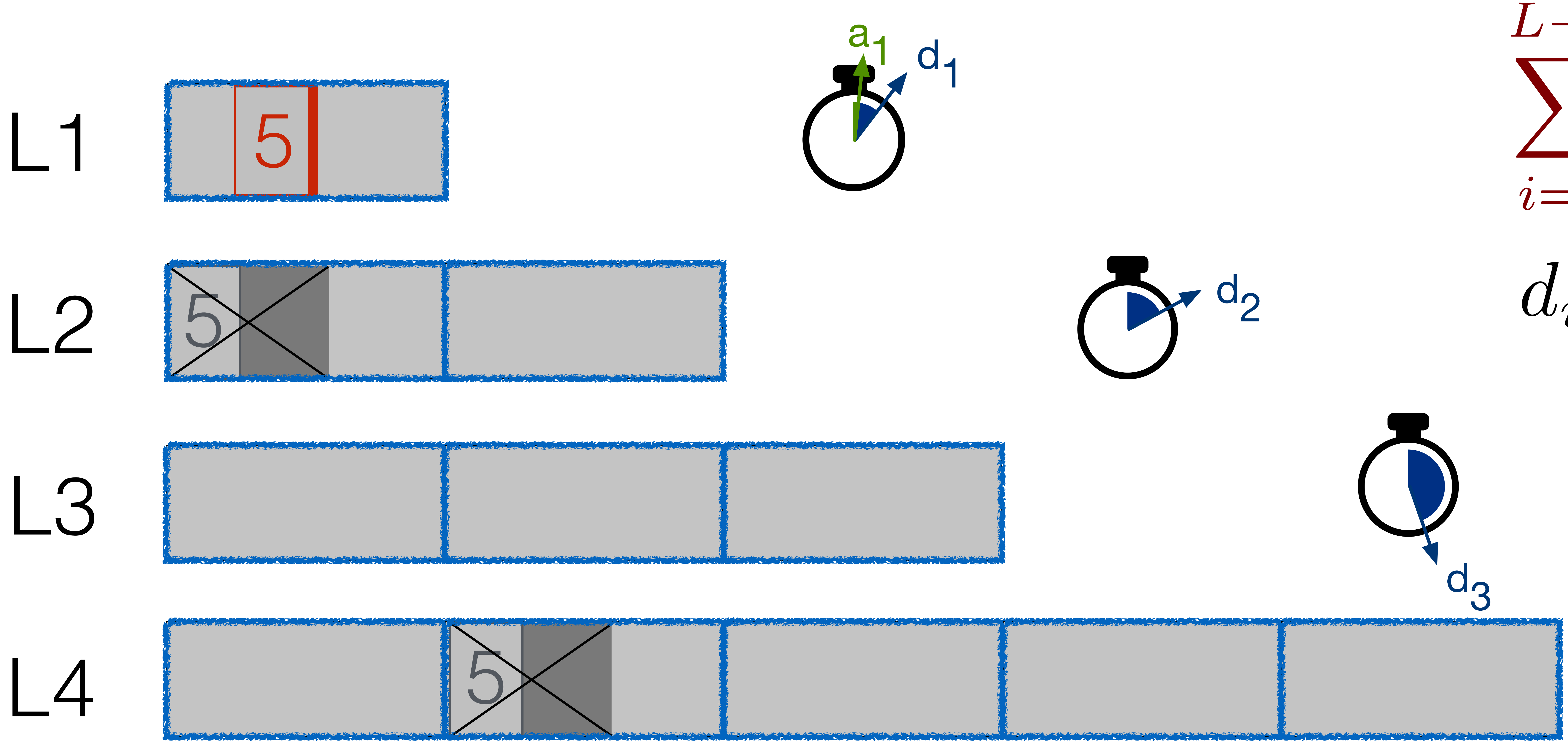


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

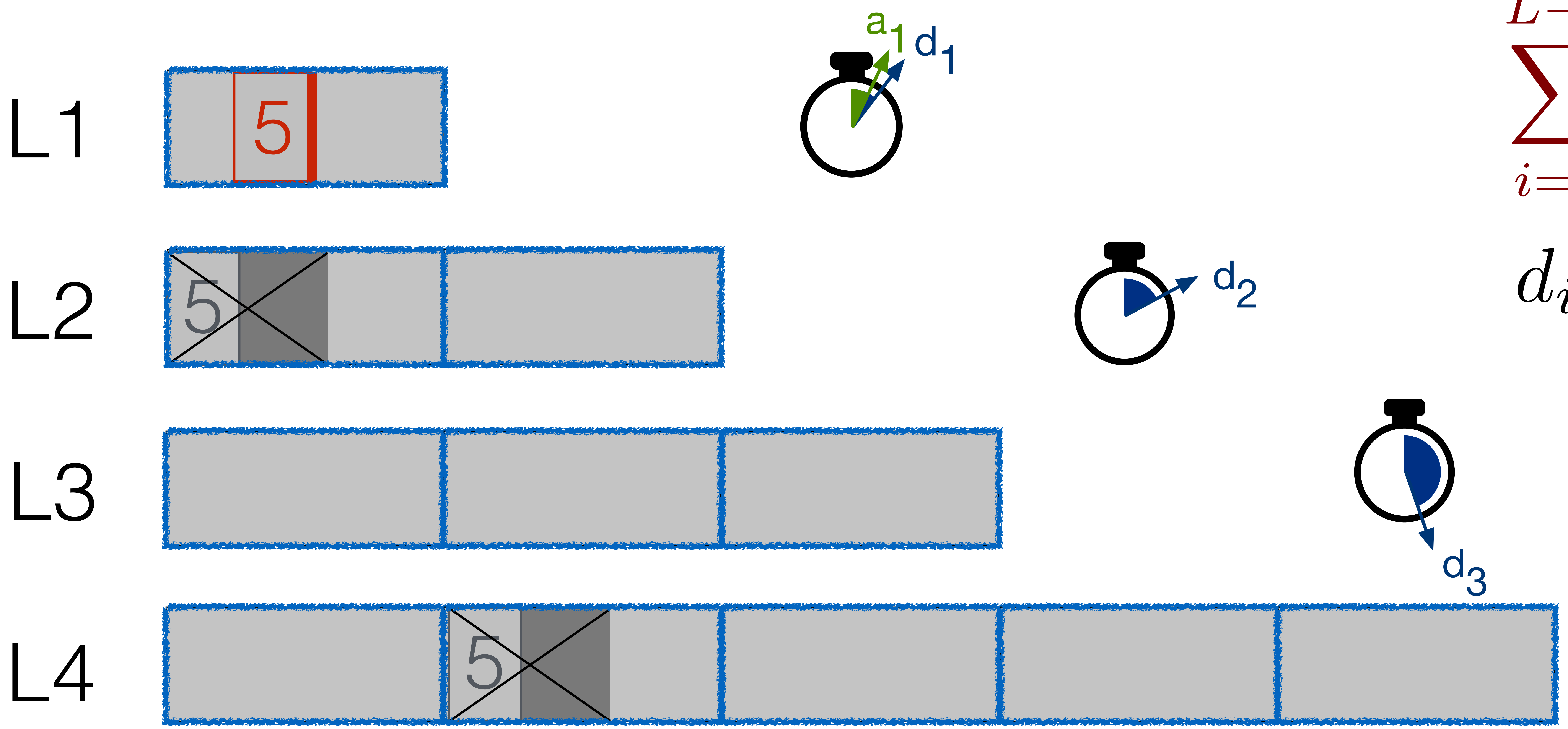


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

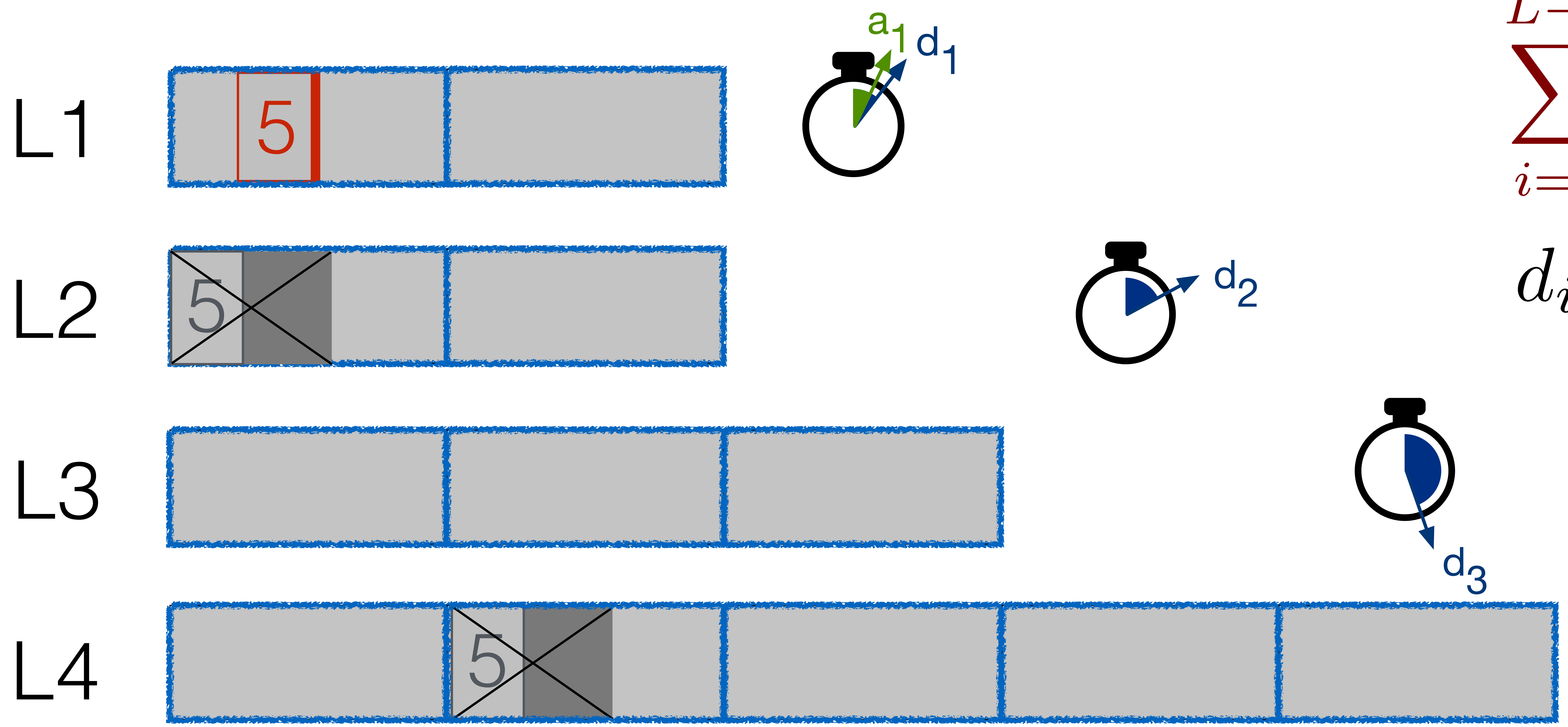


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

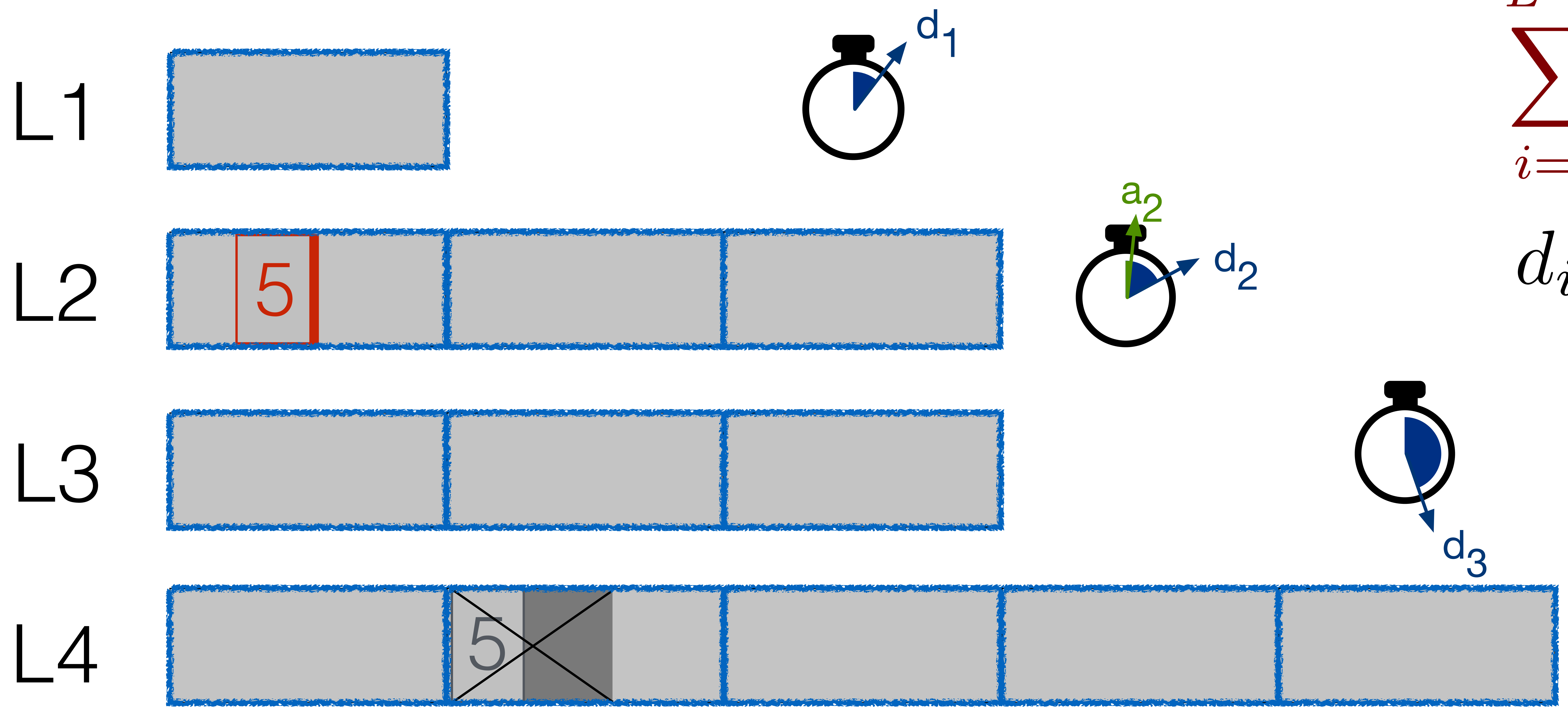


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

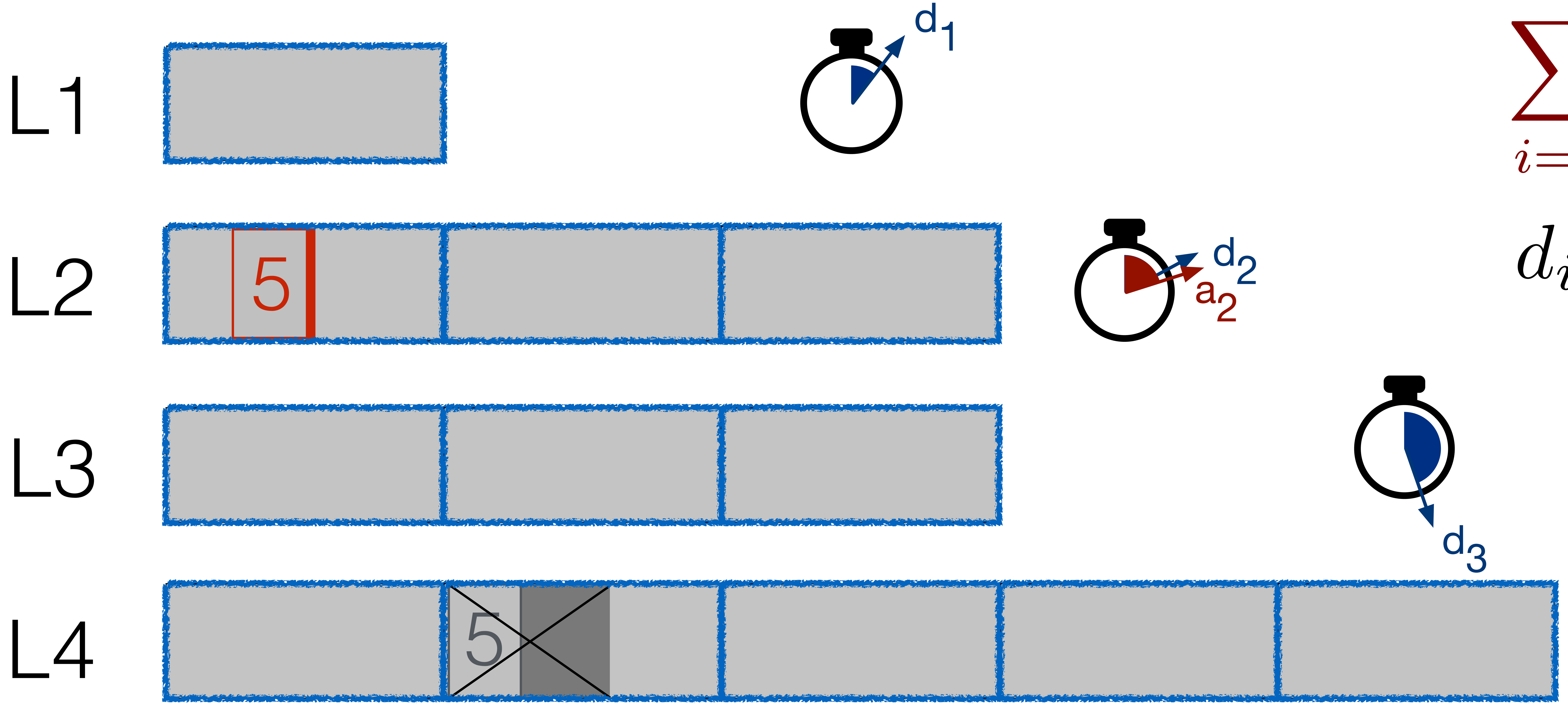


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

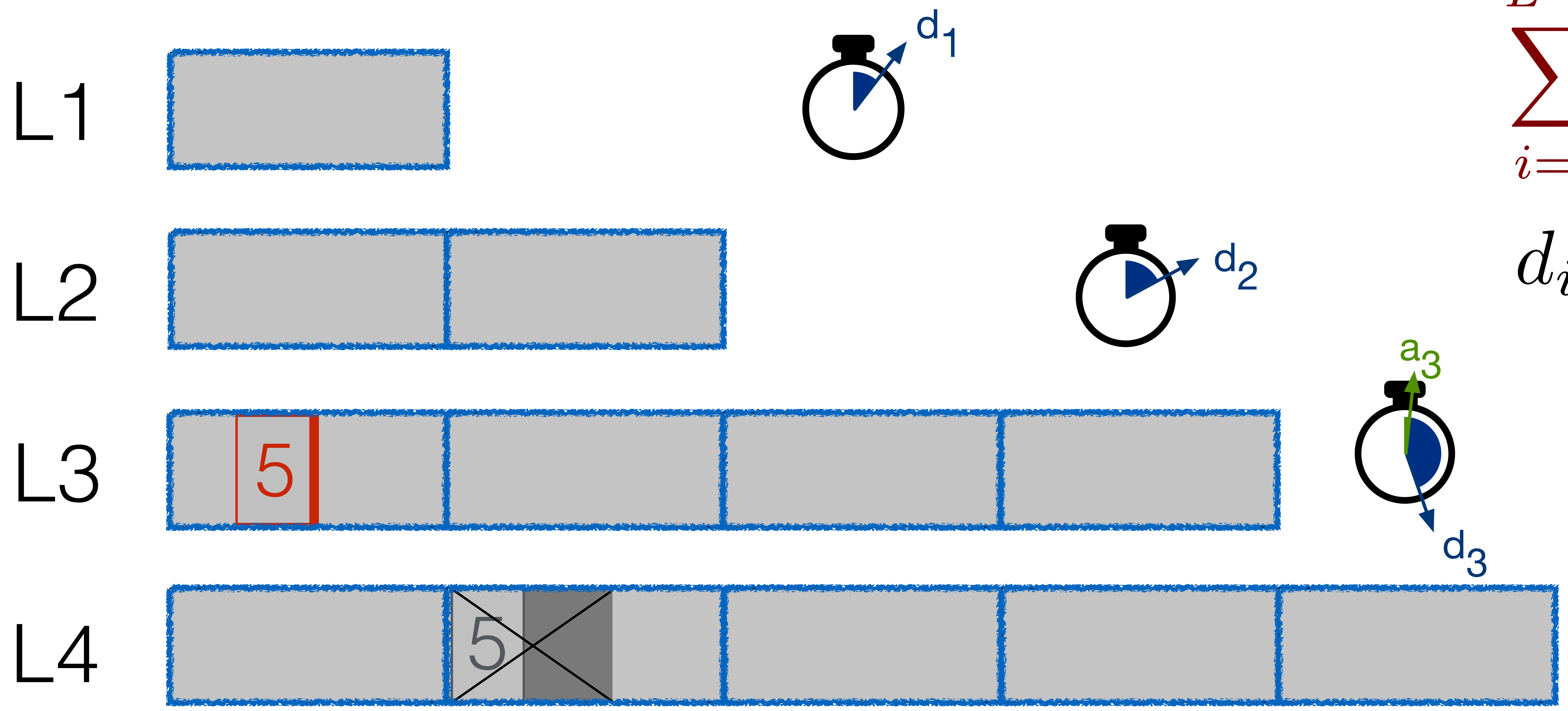


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

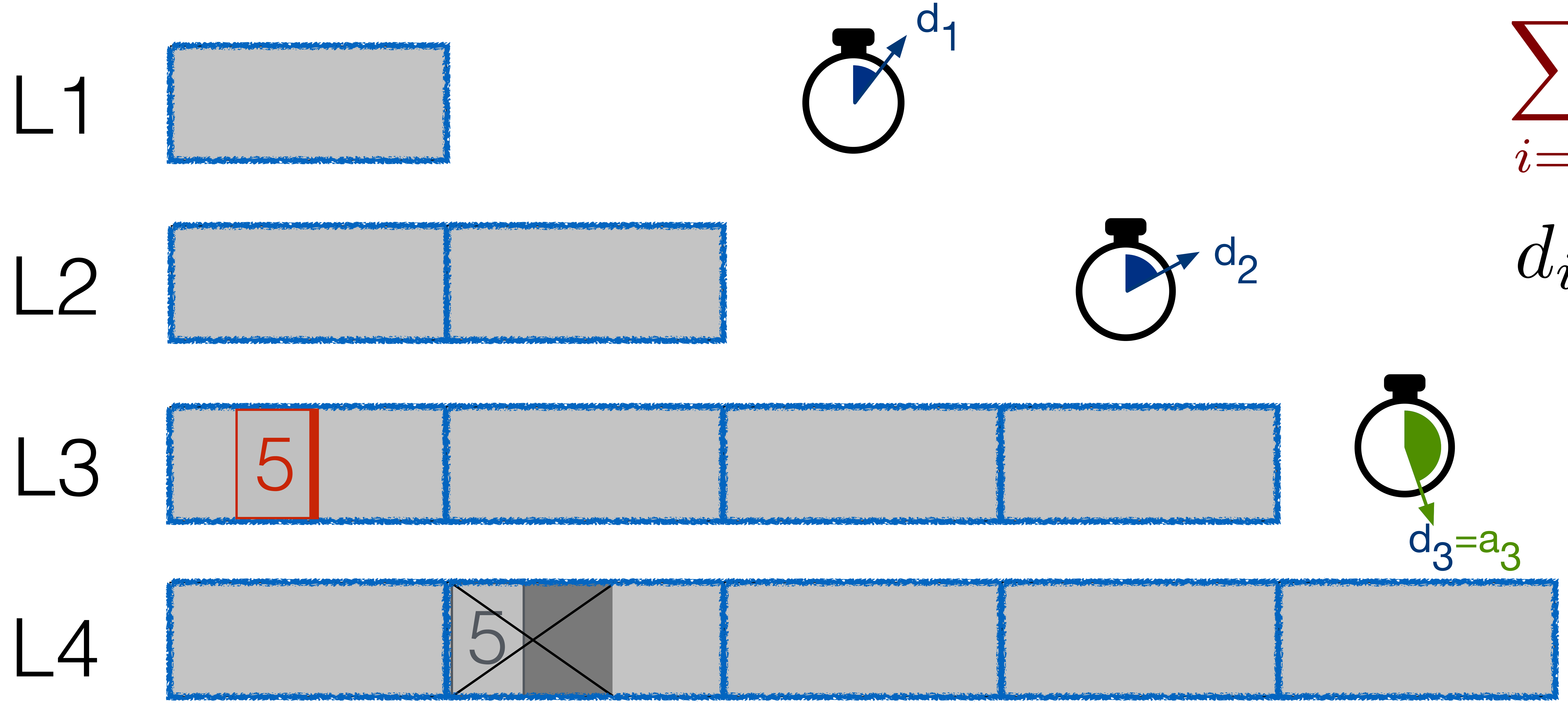


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

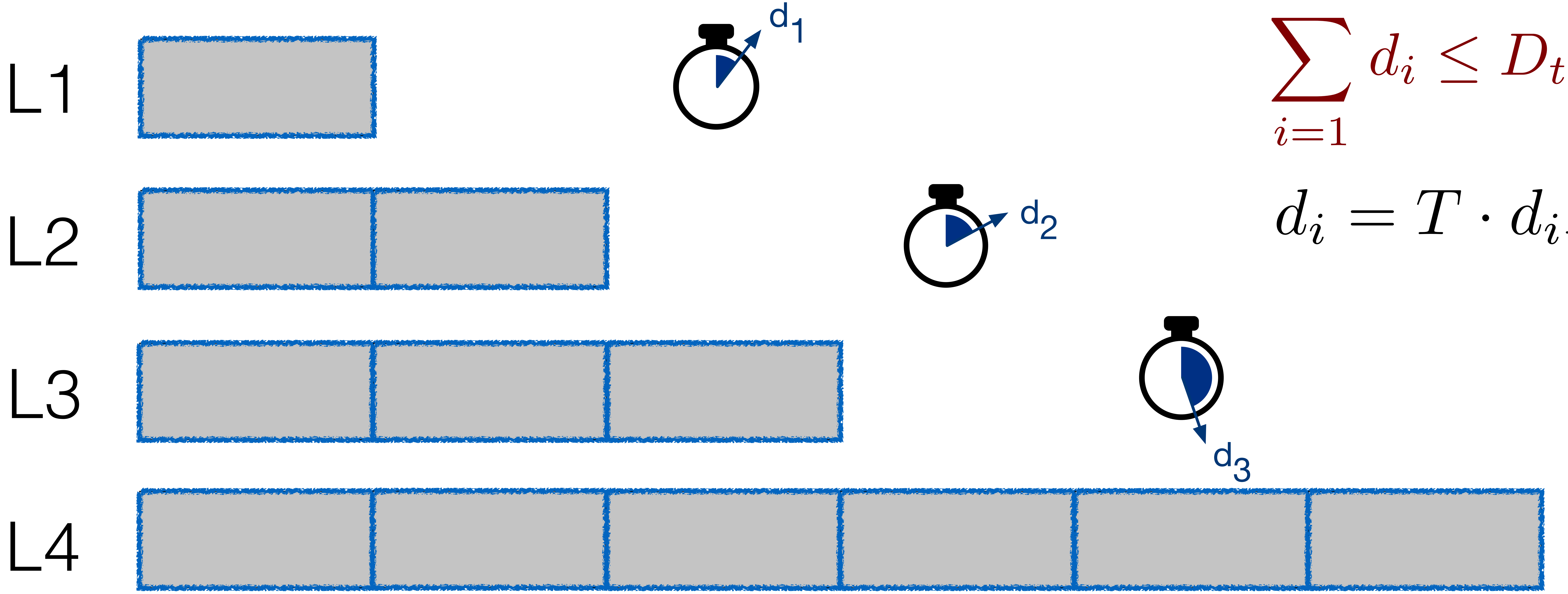


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

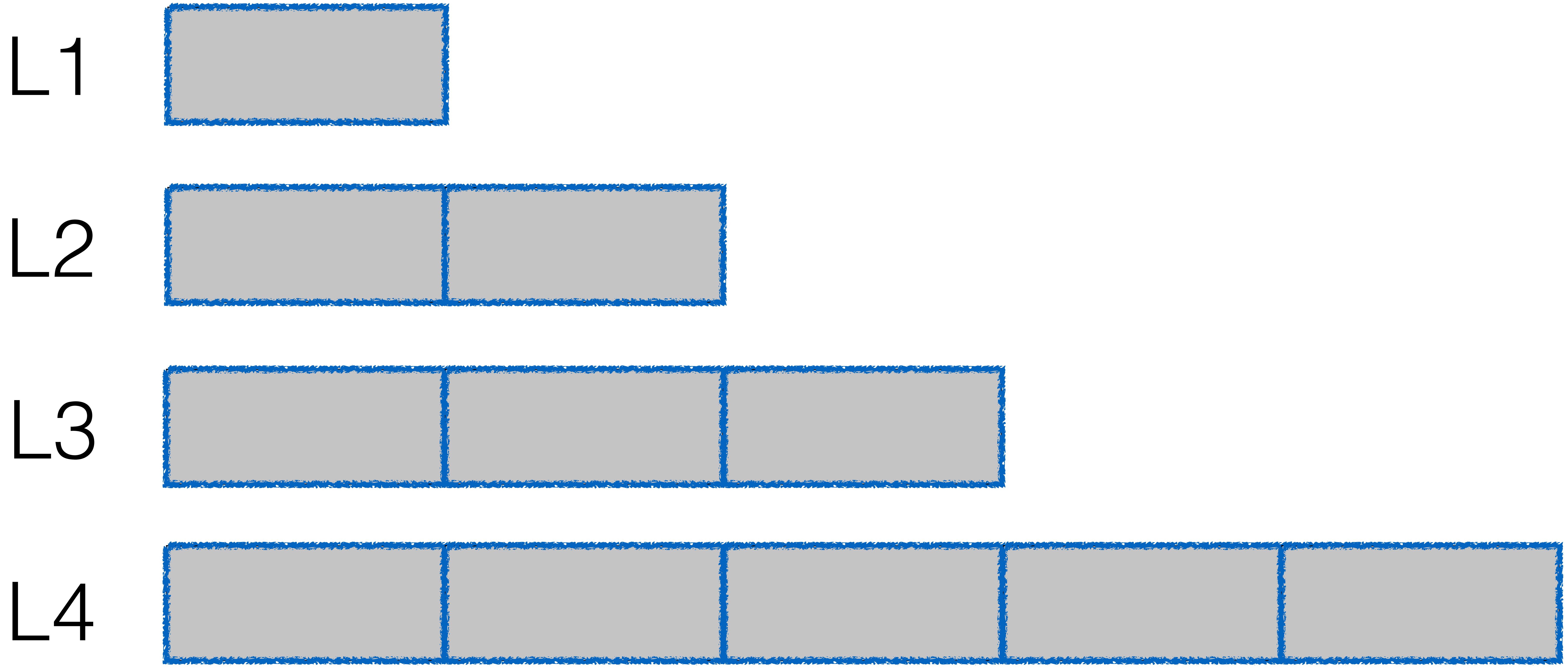


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

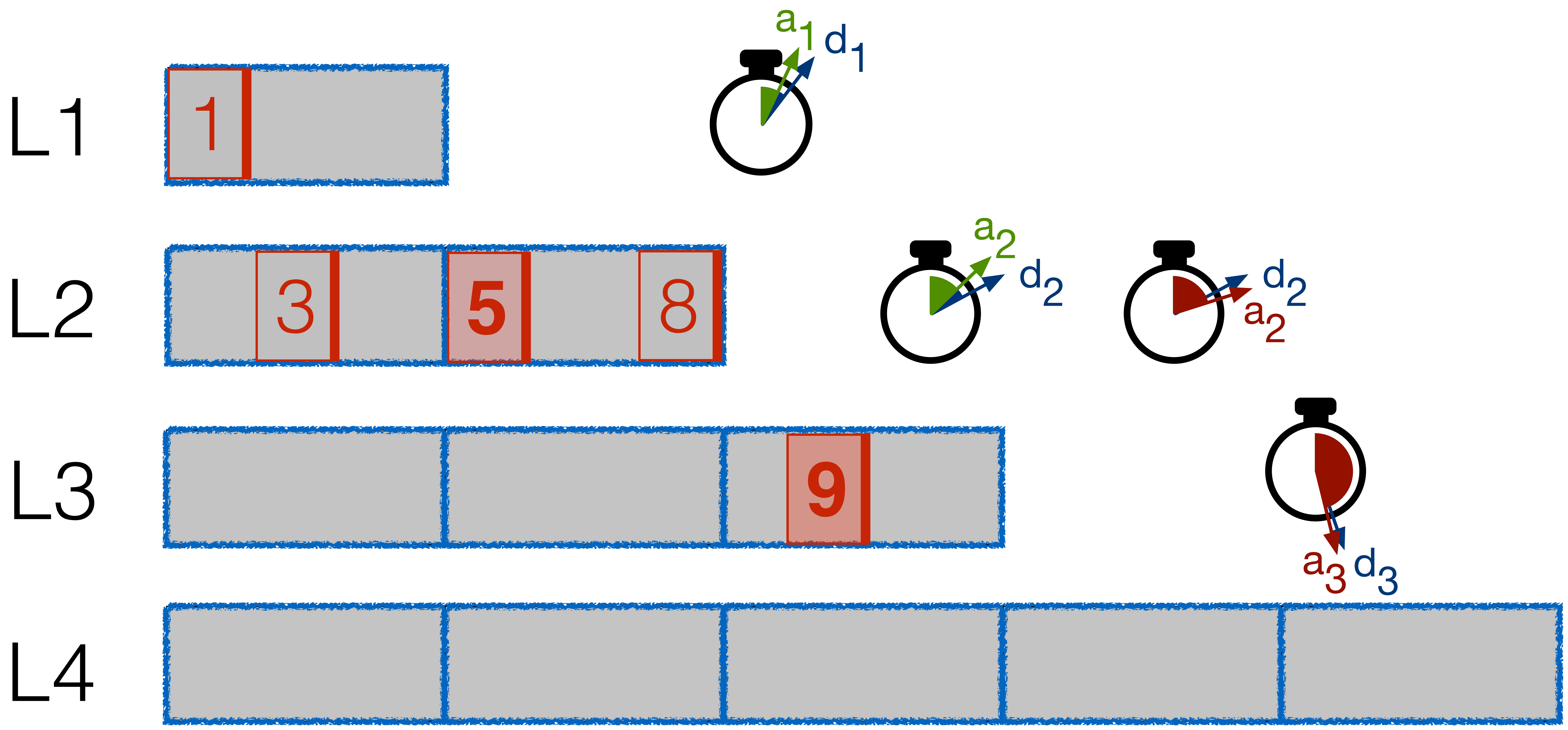
FAst DElete

breaking ties in practical workloads



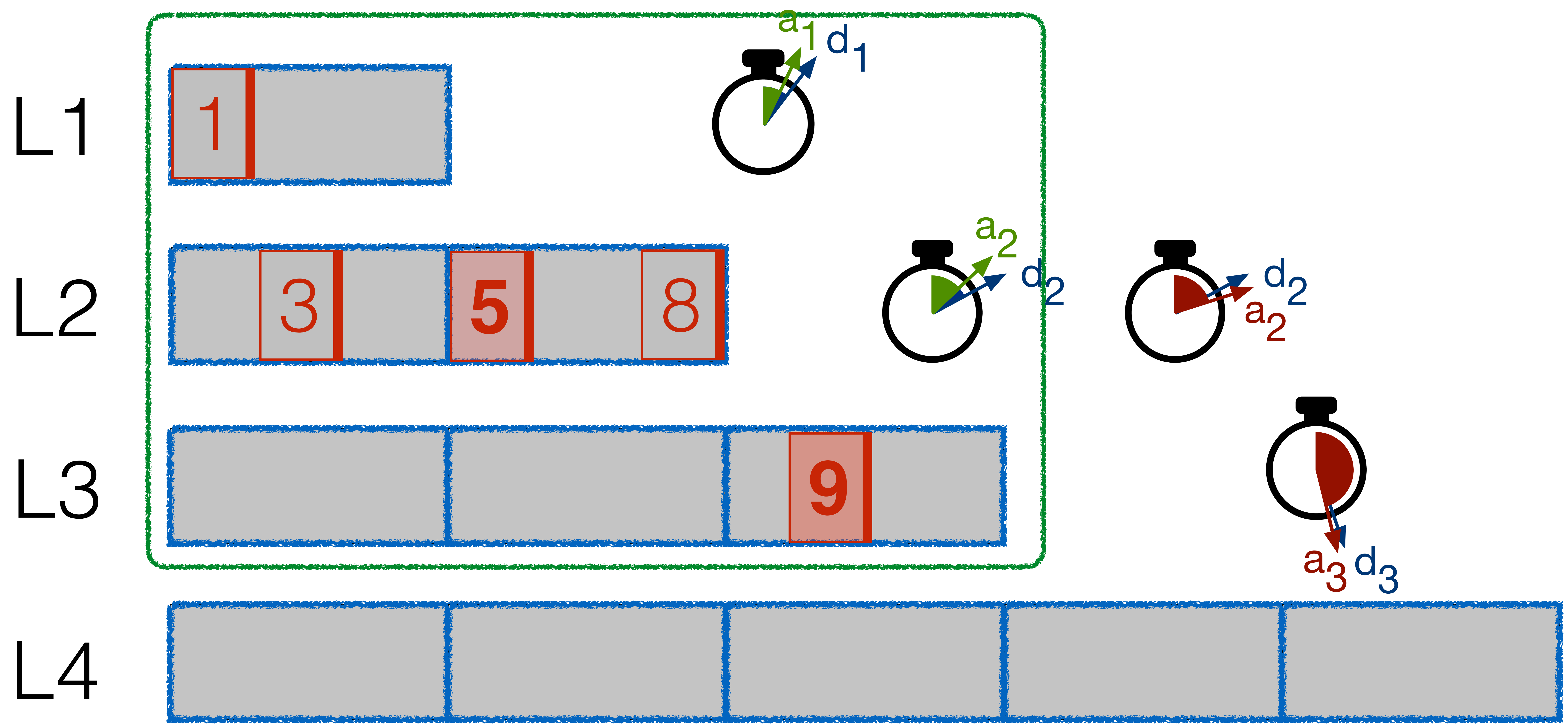
FAst DElete

breaking ties in practical workloads



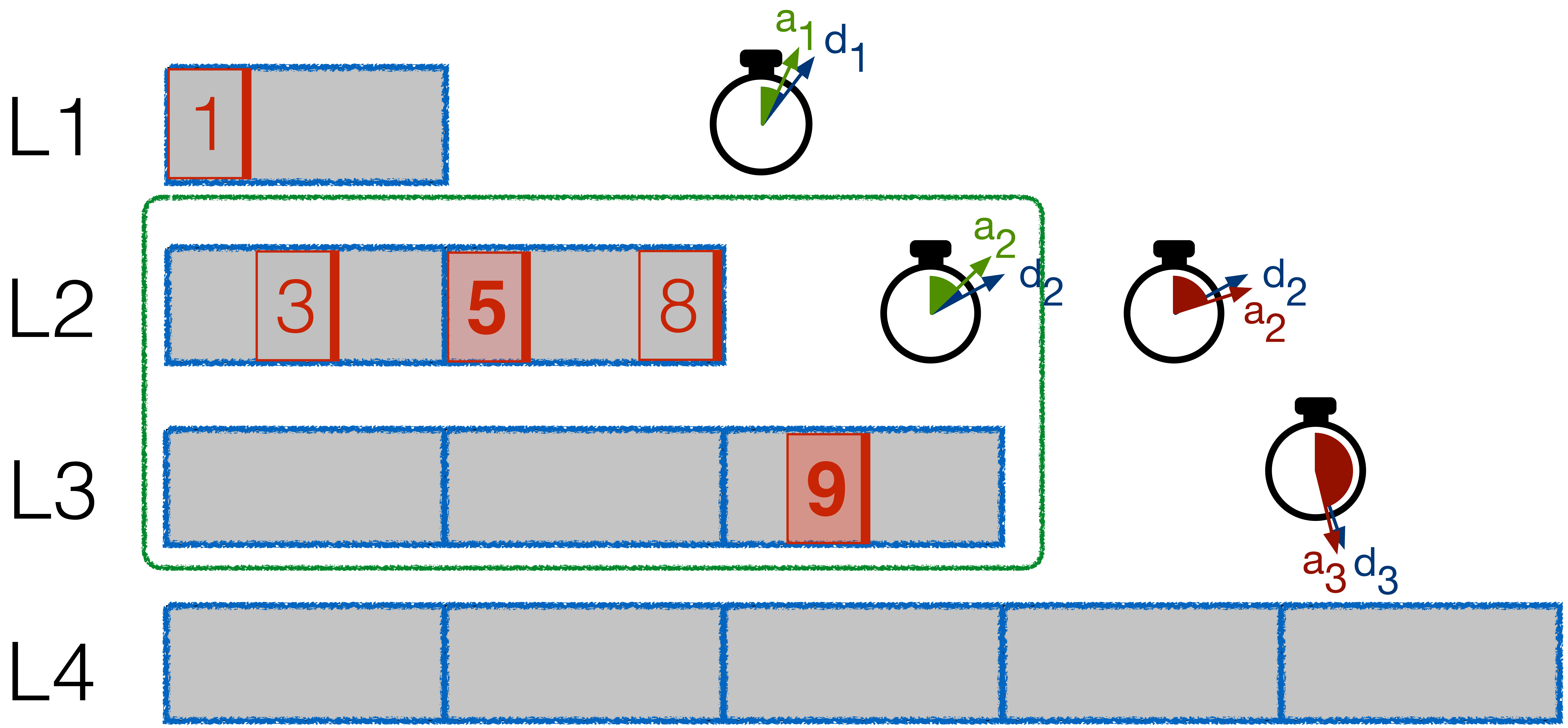
FAst DElete

breaking ties in practical workloads



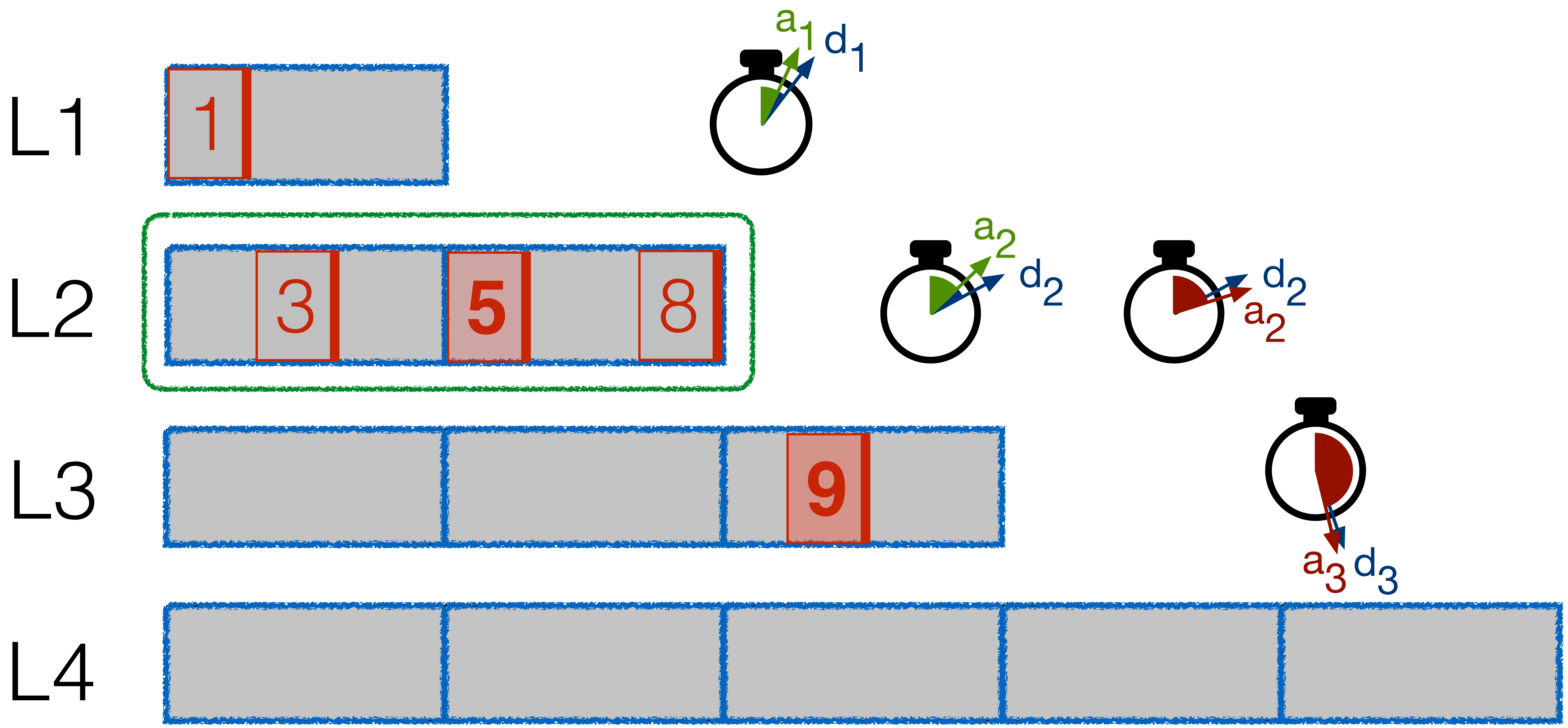
FAst DElete

breaking ties in practical workloads



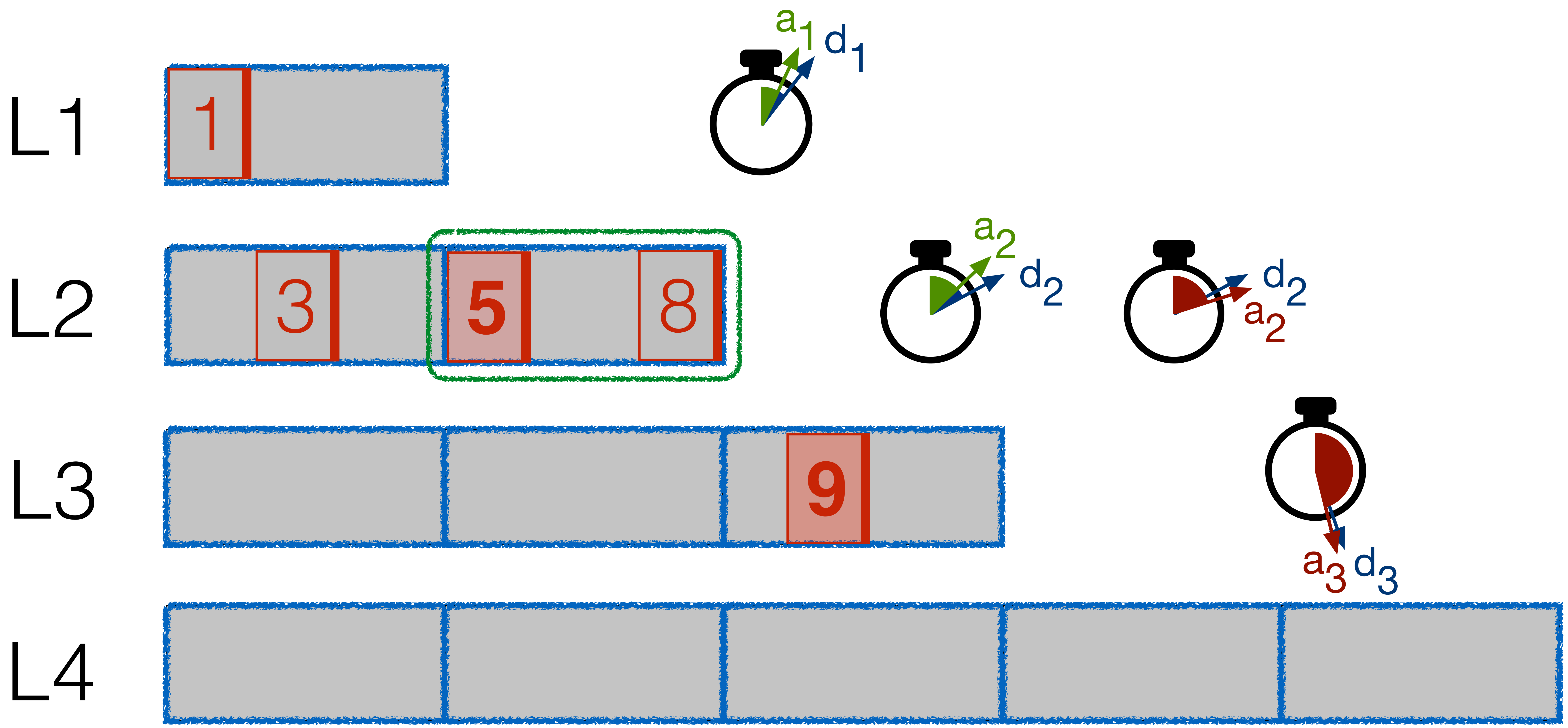
FAst DElete

breaking ties in practical workloads



FAst DElete

breaking ties in practical workloads



FAst DElete

higher write amplification ↑

4 - 25 %

improved read performance ✓

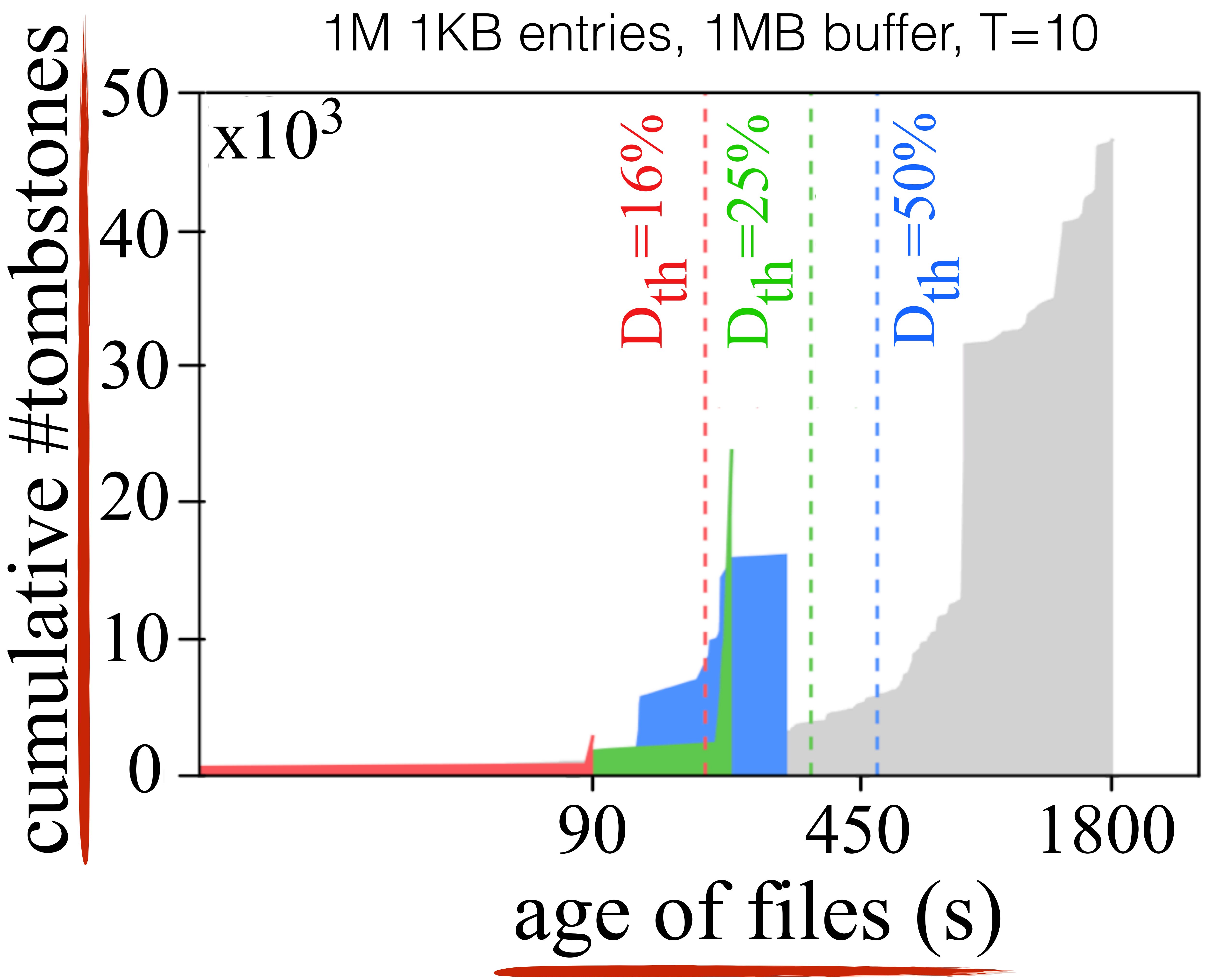
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

within D_{th}



FAst DElete

higher write amplification

0.7%

improved read performance

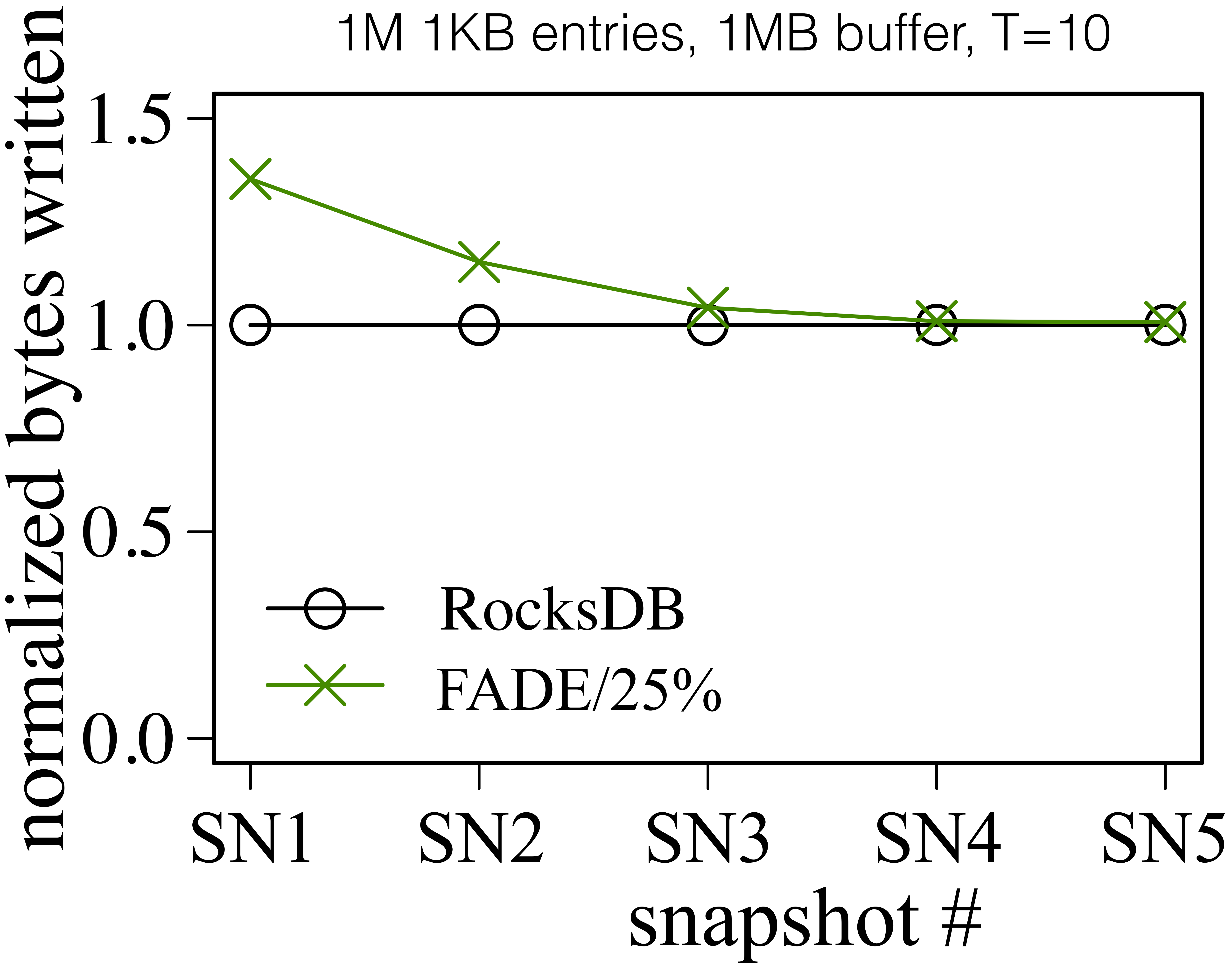
1.2 - 1.4x

reduced space amplification

2.1 - 9.8x

timely delete persistence

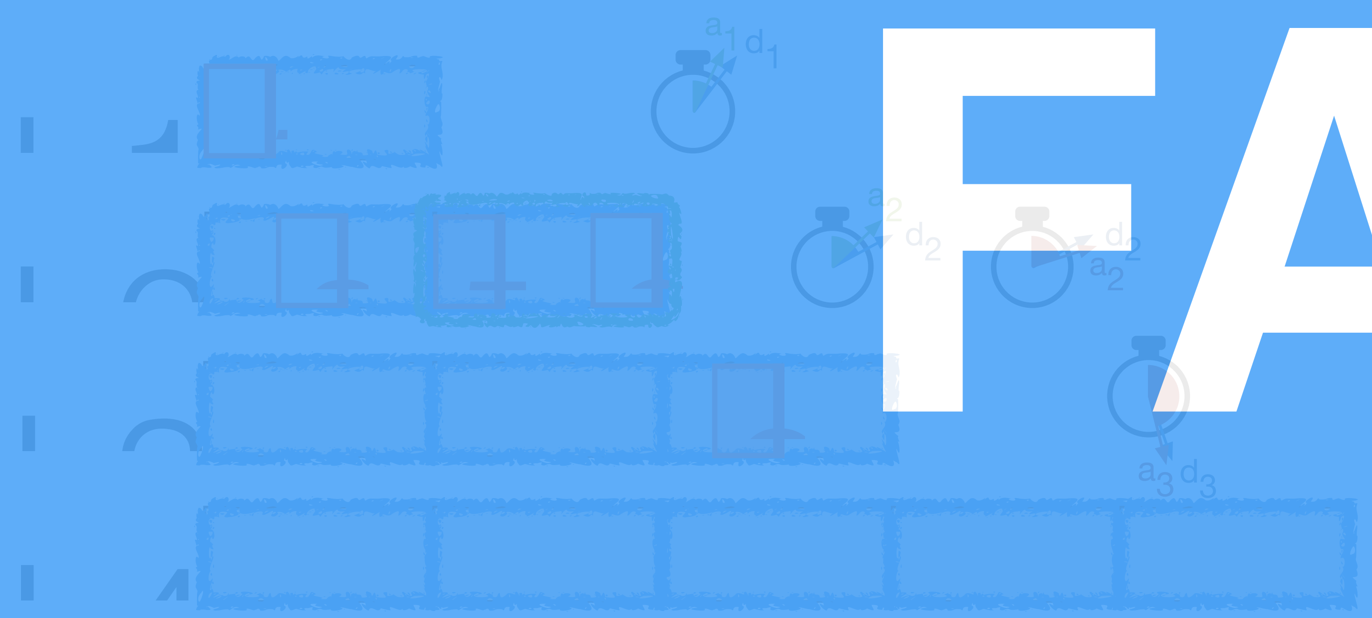
within D_{th}



the solution

FADE

FAst DElete



higher write amplification

improved read performance ✓

reduced space amplification ✓

timely delete persistence ✓

KiWi

latency spikes

superfluous I/Os

the solution

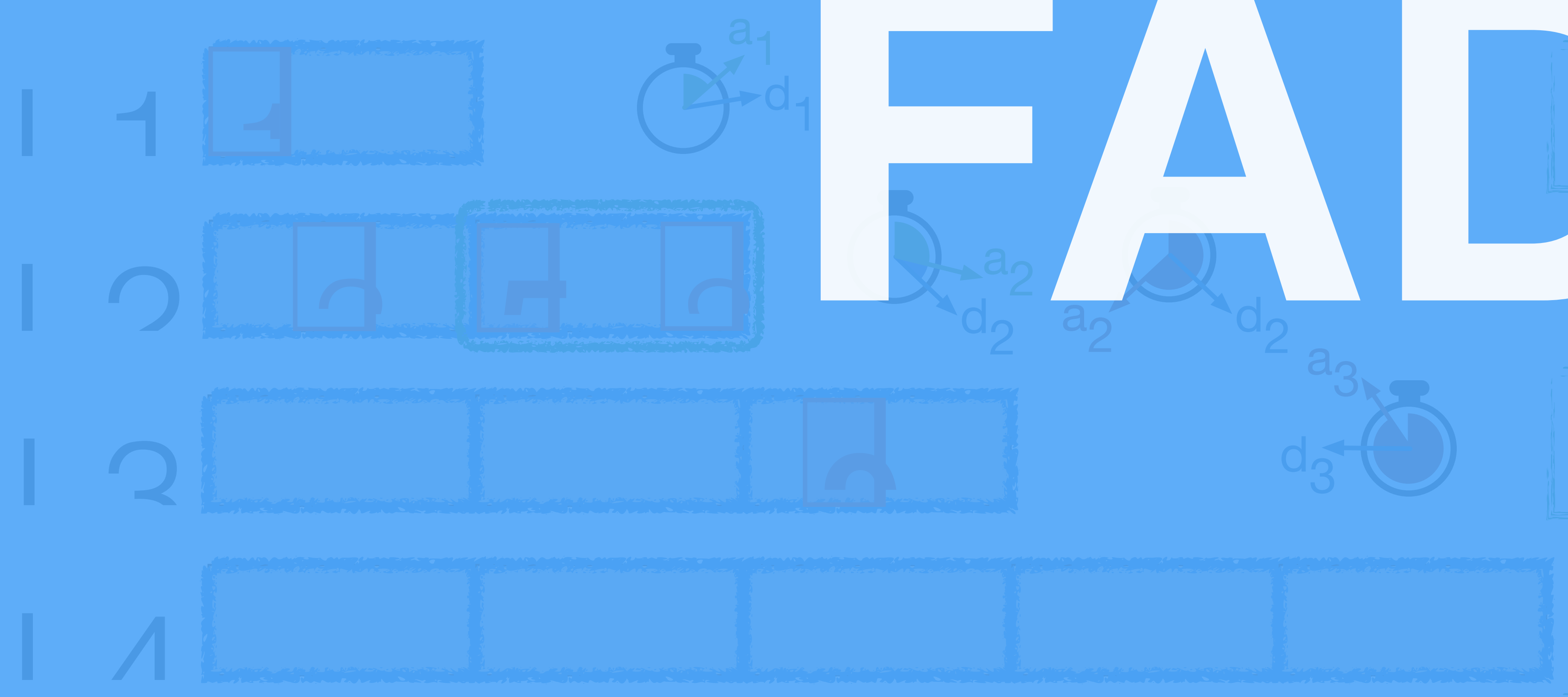
FAst DElete

higher write amplification

read performance

reduced space amplification

timely delete persistence



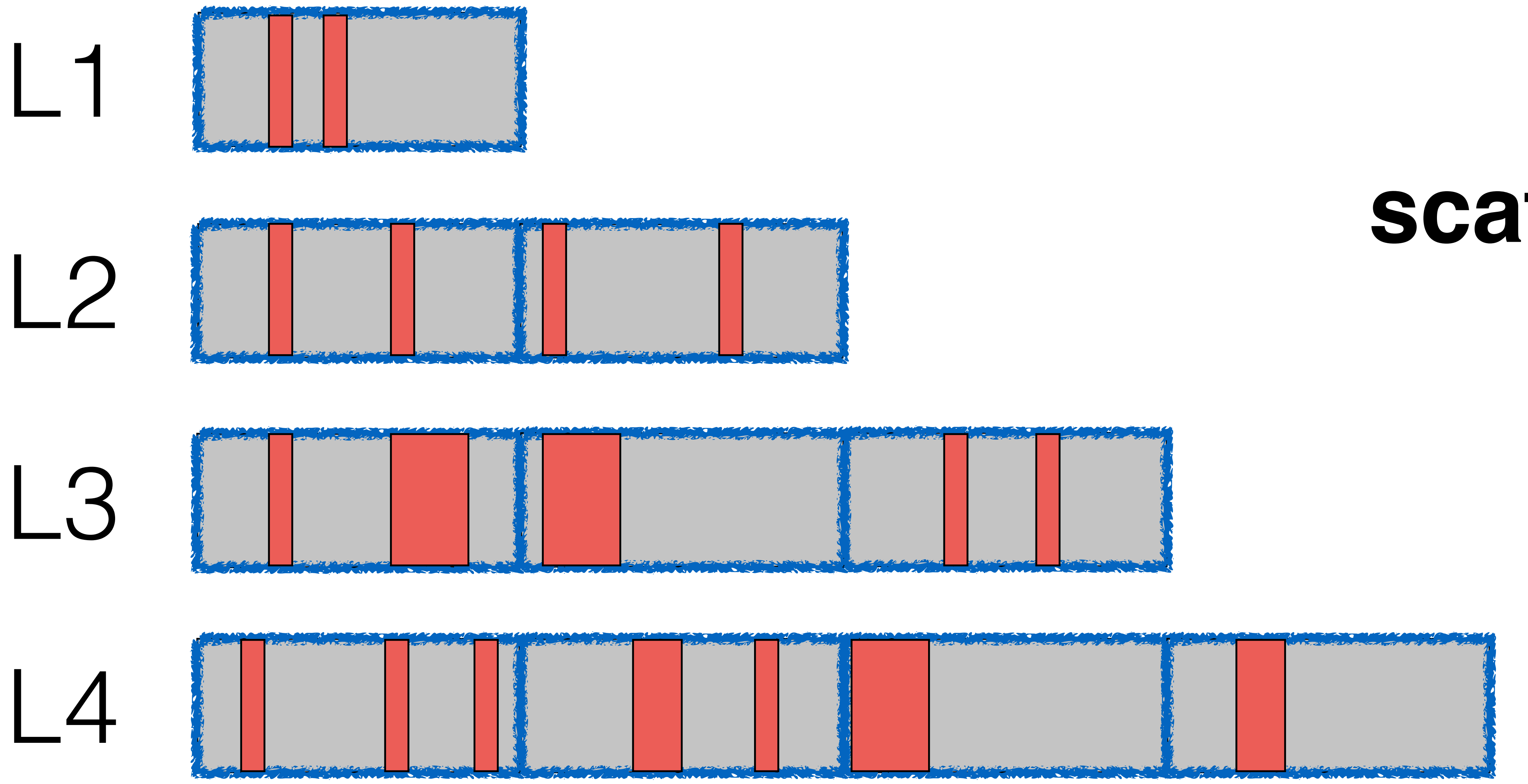
FADE

latency spikes

KiWi

Key Weaving storage layout

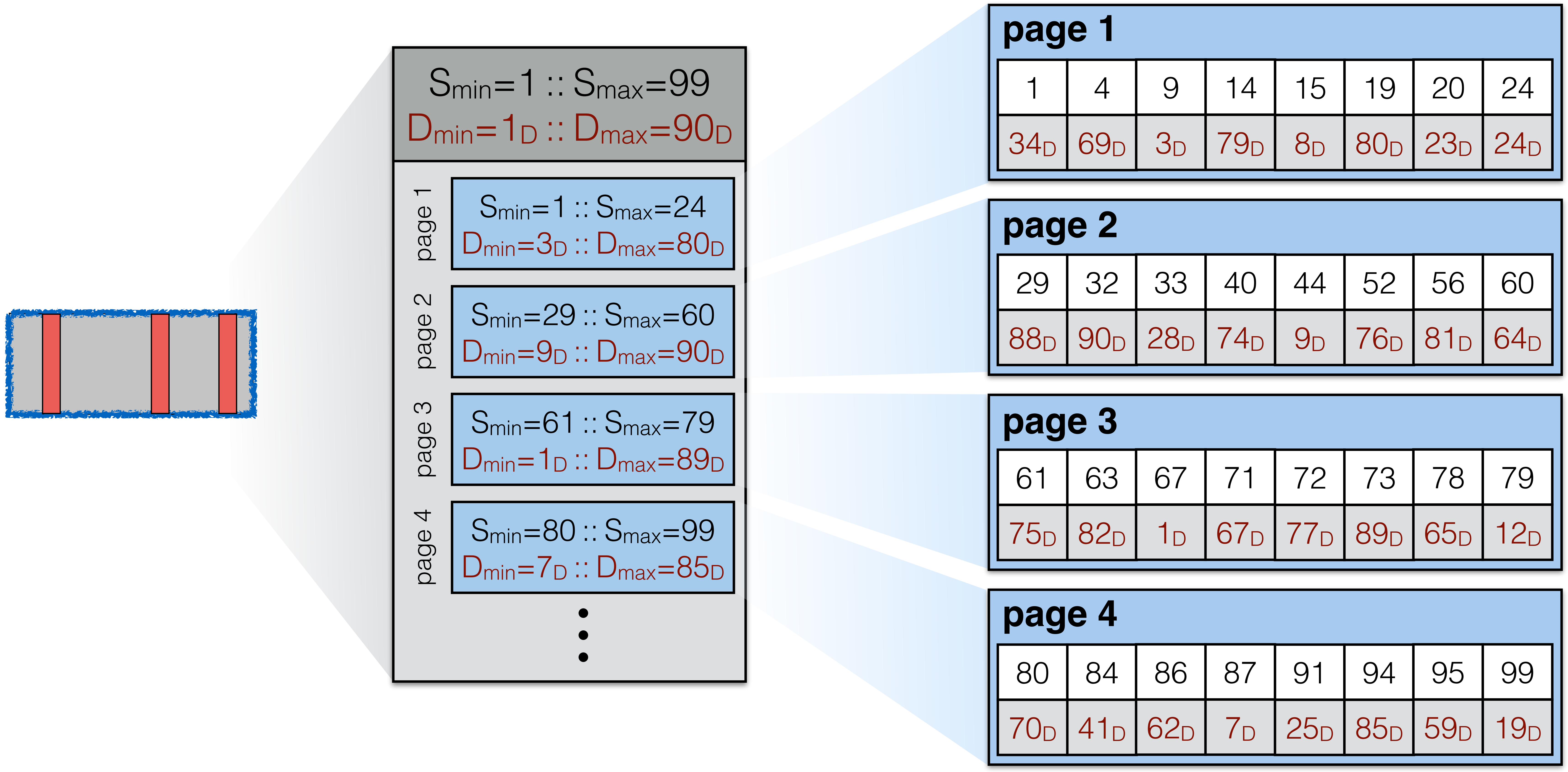
delete all entries older than: **D days**



scattered occurrences

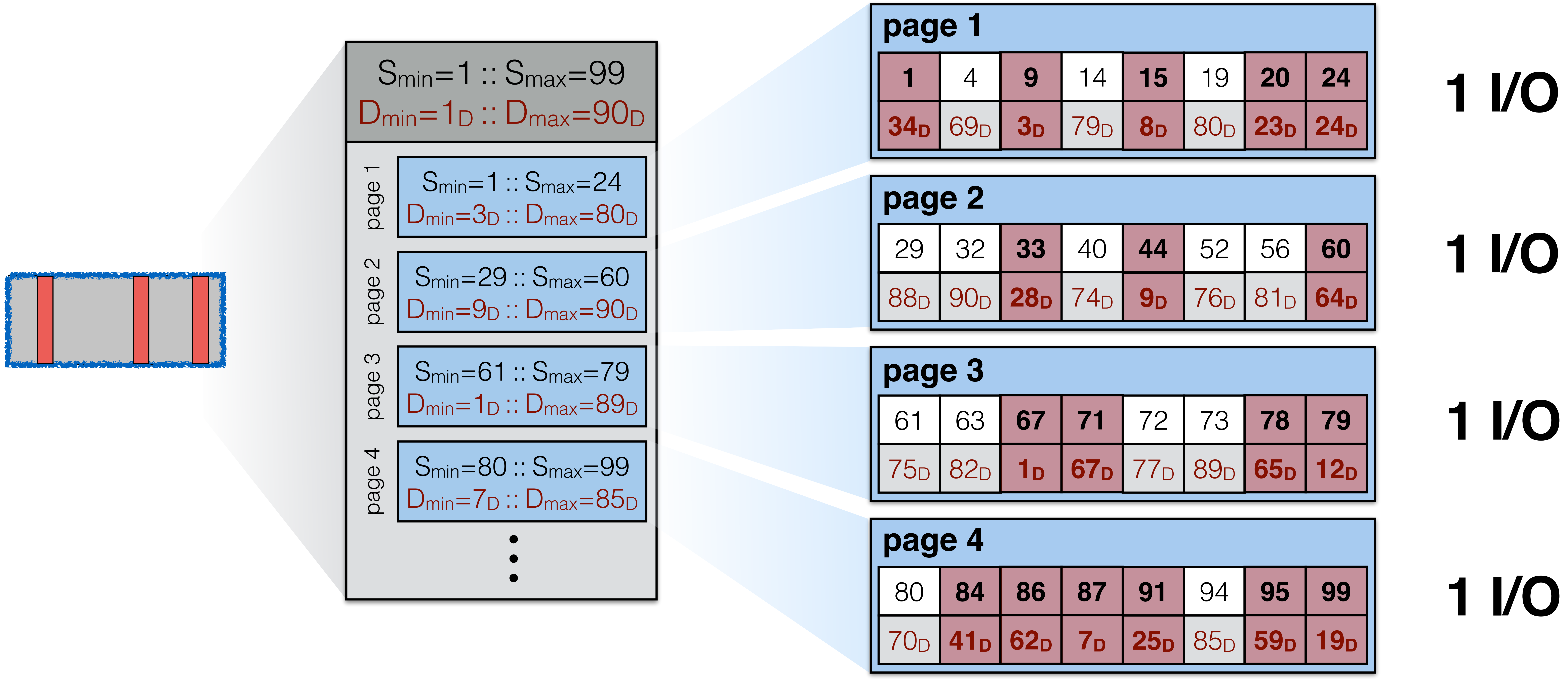
Key Weaving storage layout

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



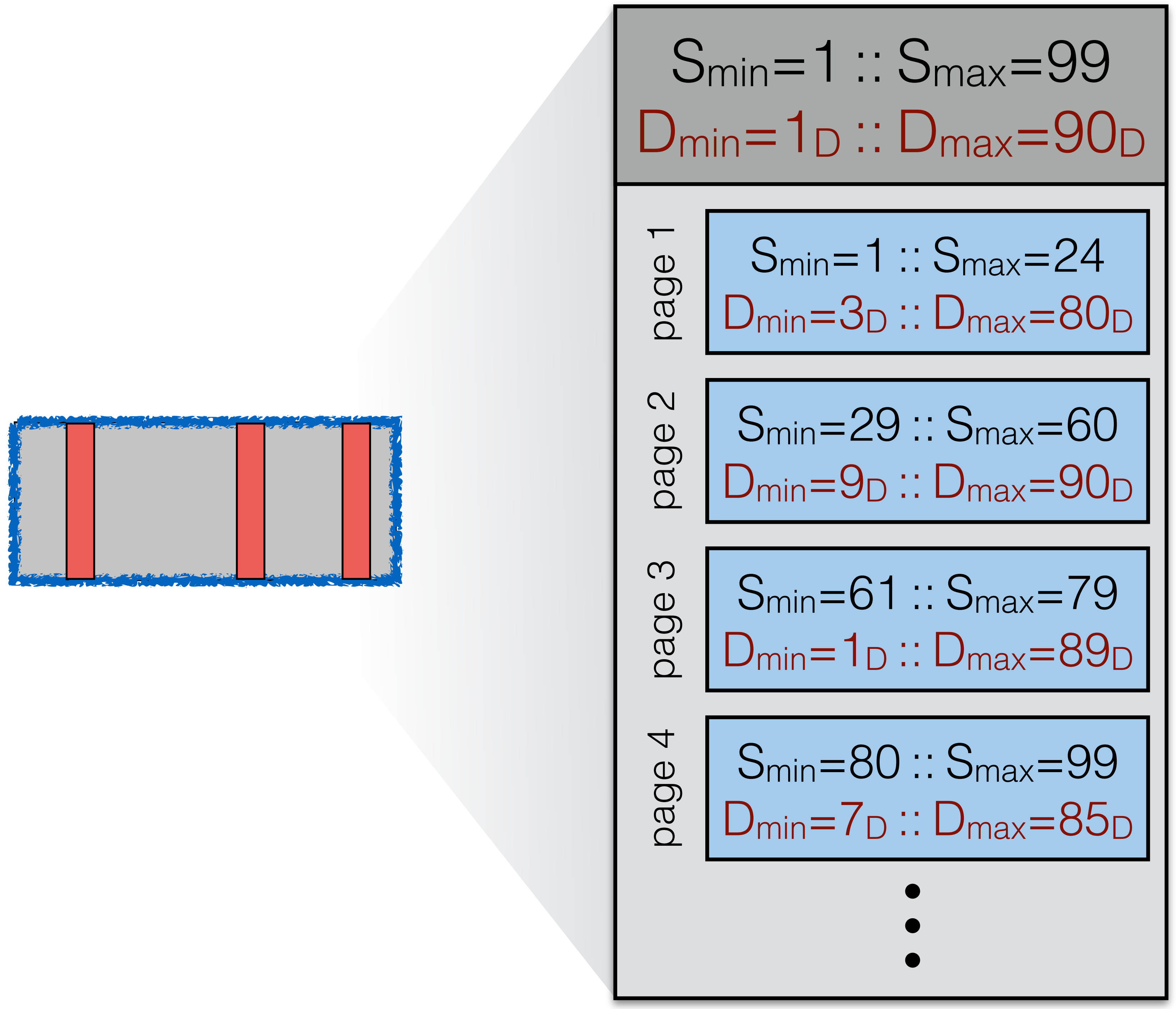
Key Weaving storage layout

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



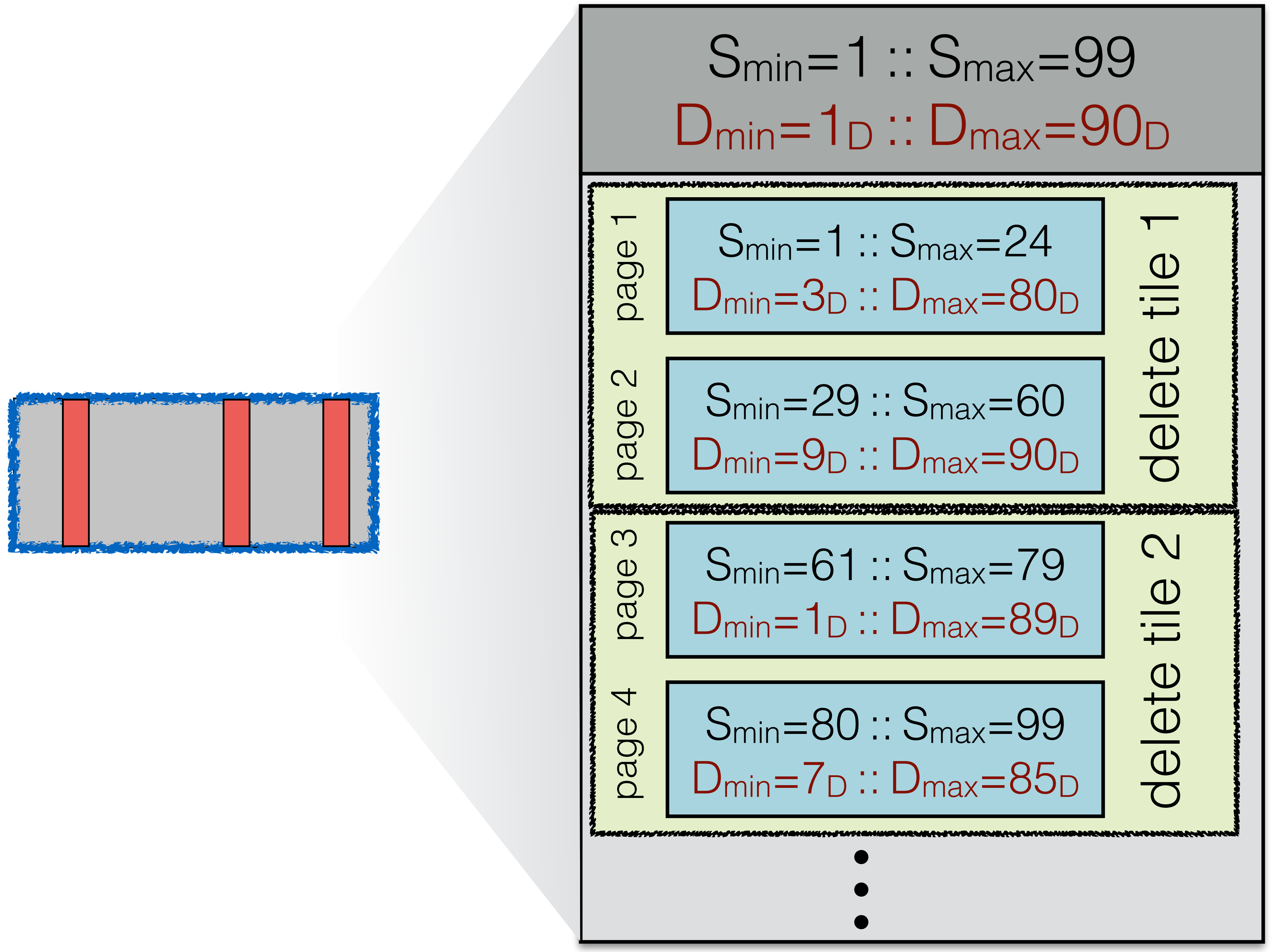
Key Weaving storage layout

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



Key Weaving storage layout

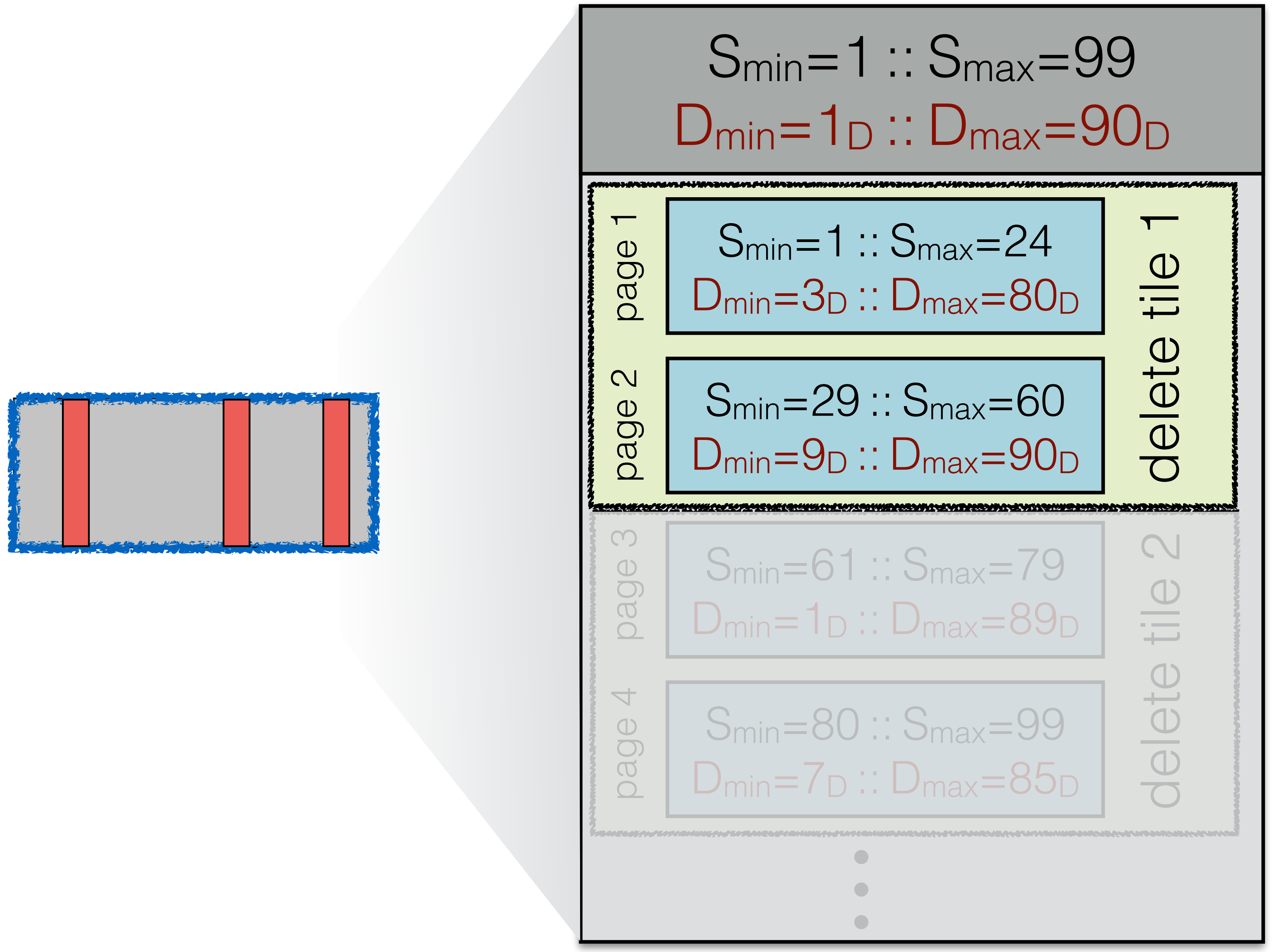
delete all entries with timestamp **<= 65$_D$**



partitioned on S

Key Weaving storage layout

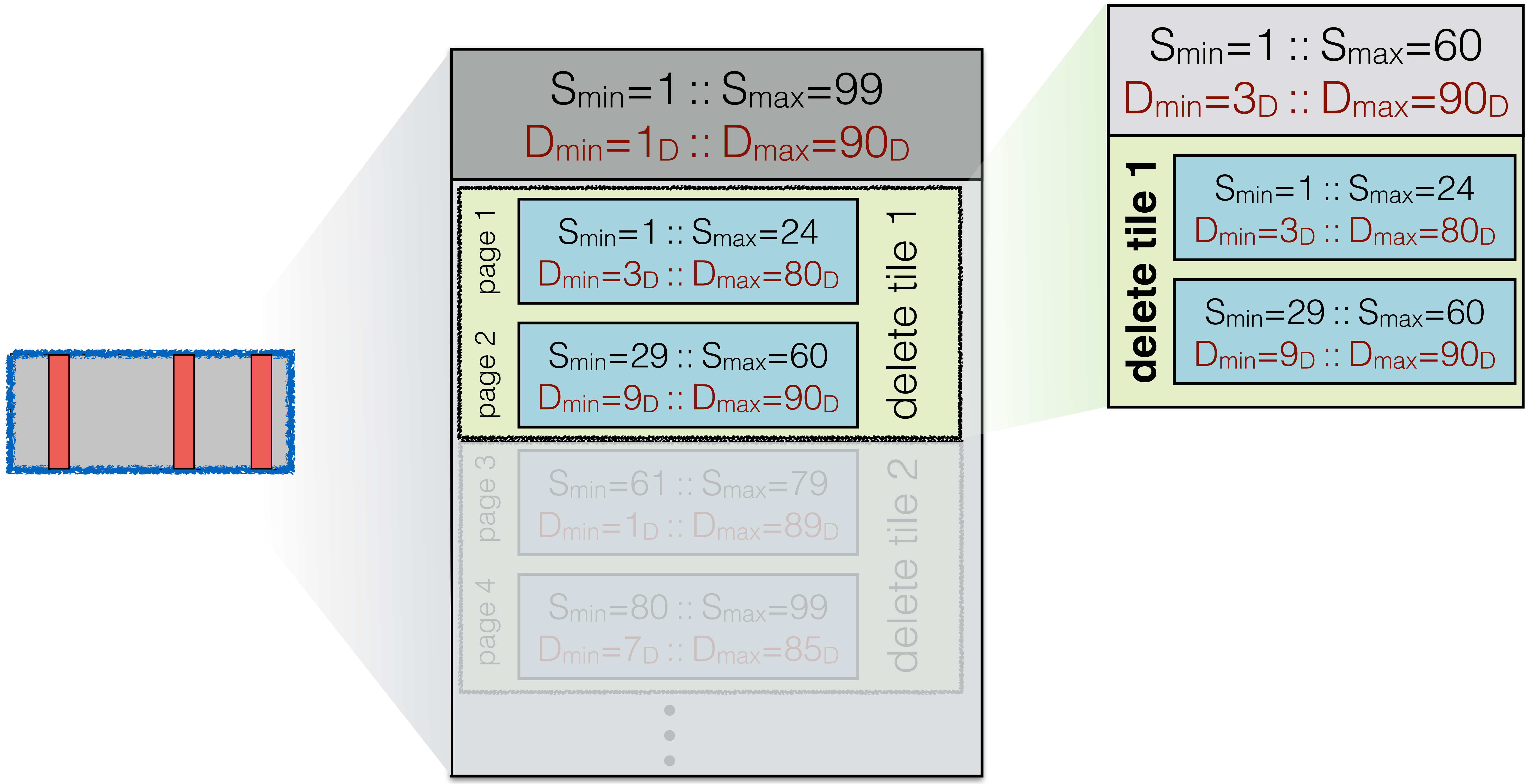
delete all entries with timestamp **<= 65D**



partitioned on S

Key Weaving storage layout

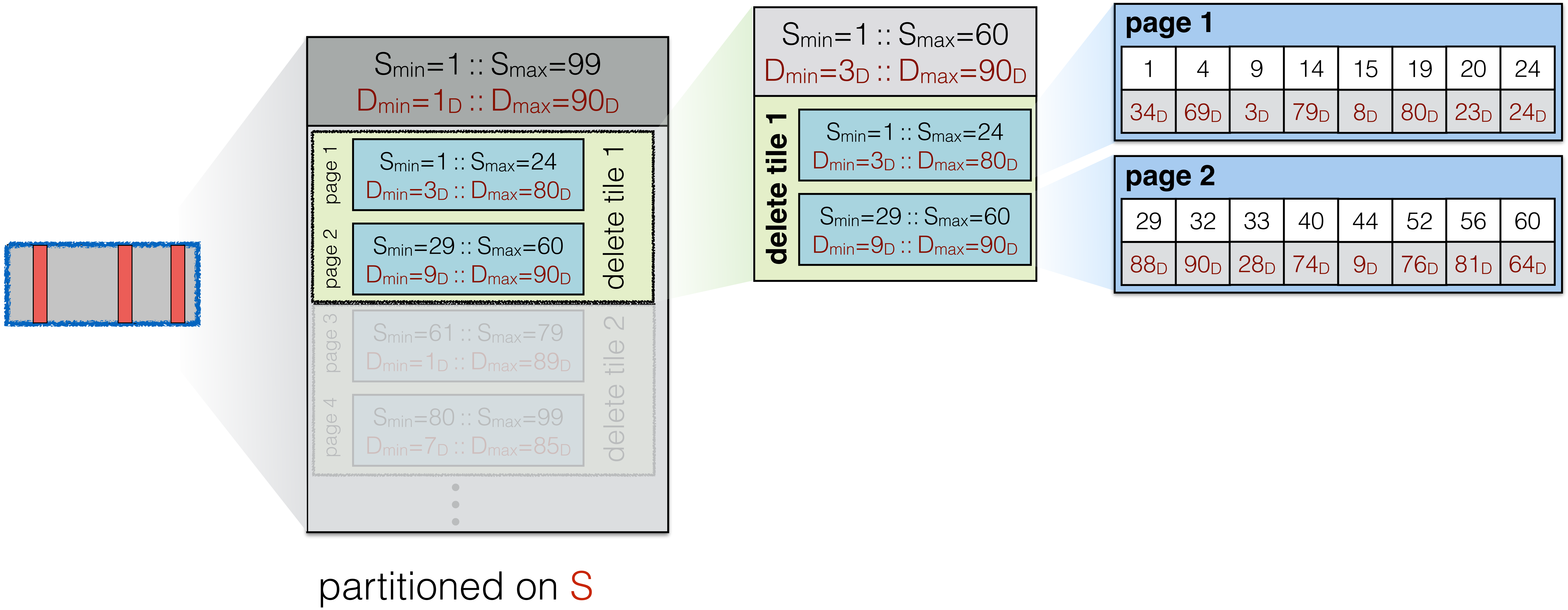
delete all entries with timestamp **<= 65D**



partitioned on S

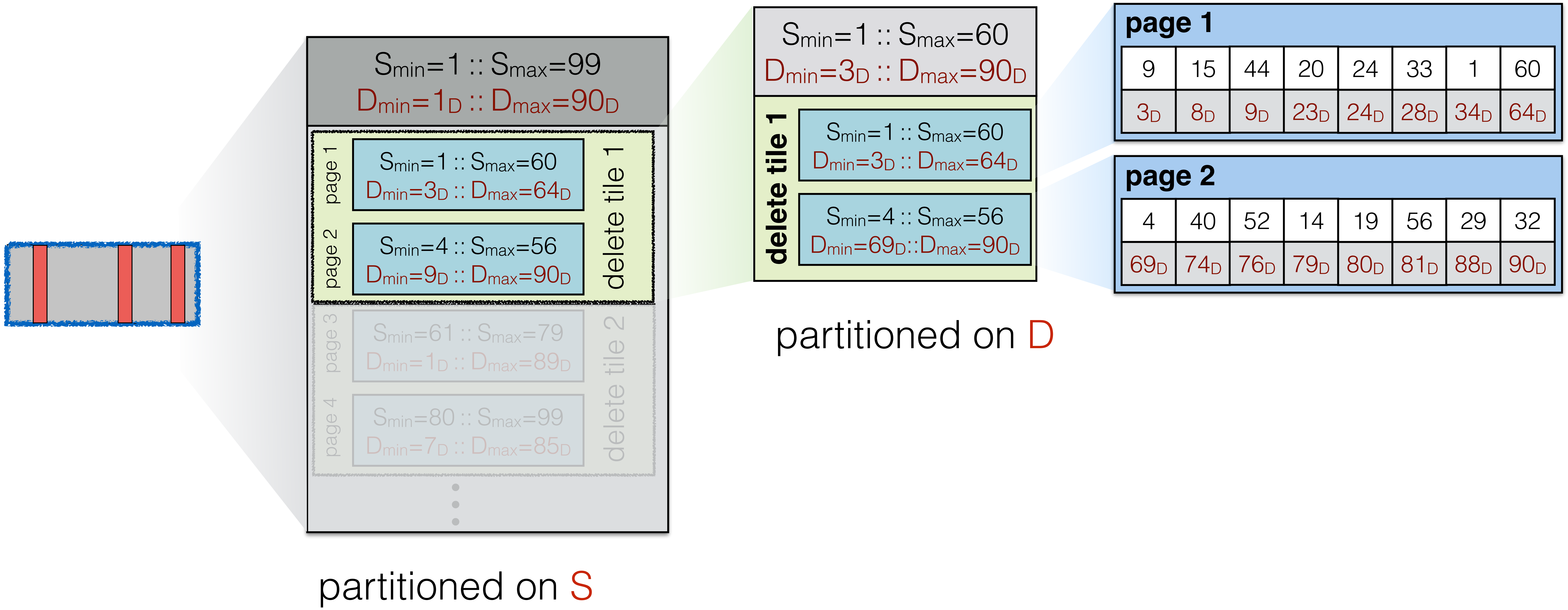
Key Weaving storage layout

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



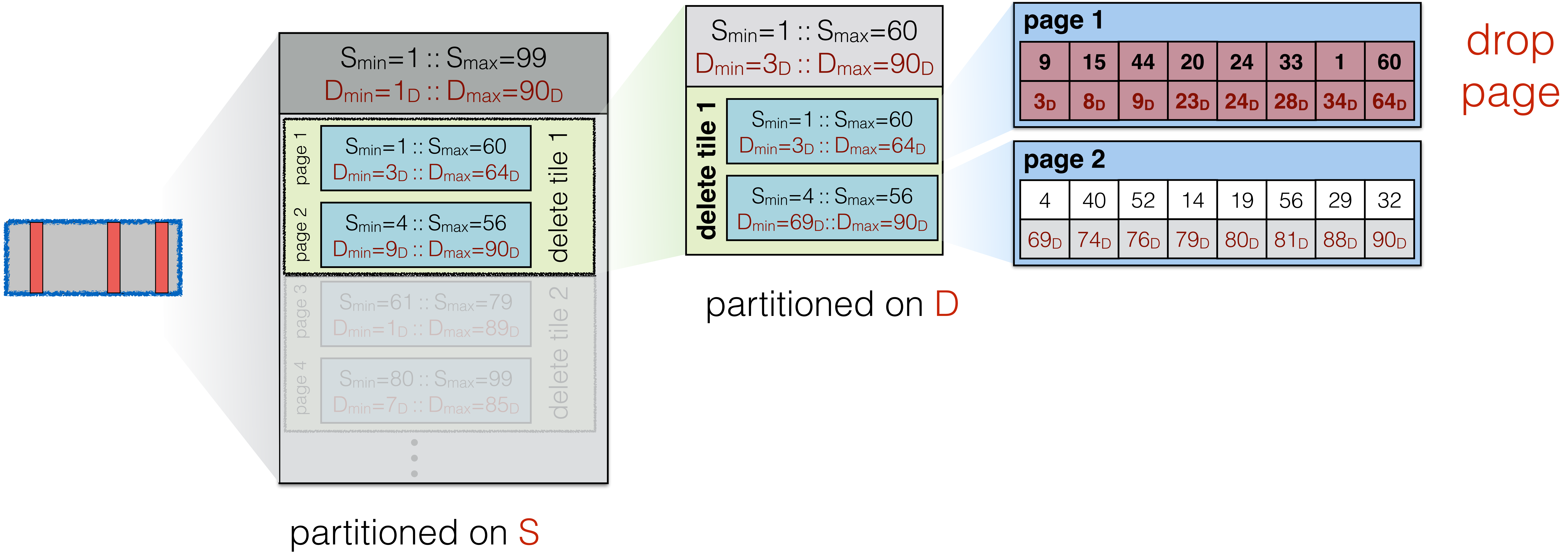
Key Weaving storage layout

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



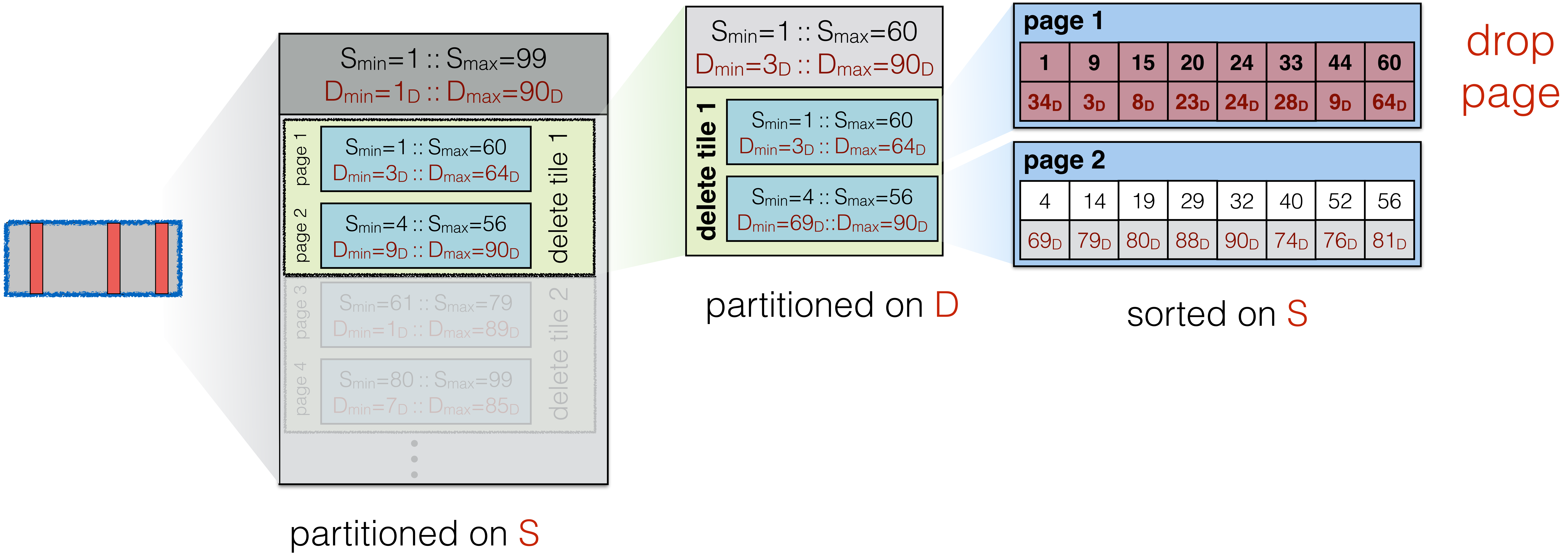
Key Weaving storage layout

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



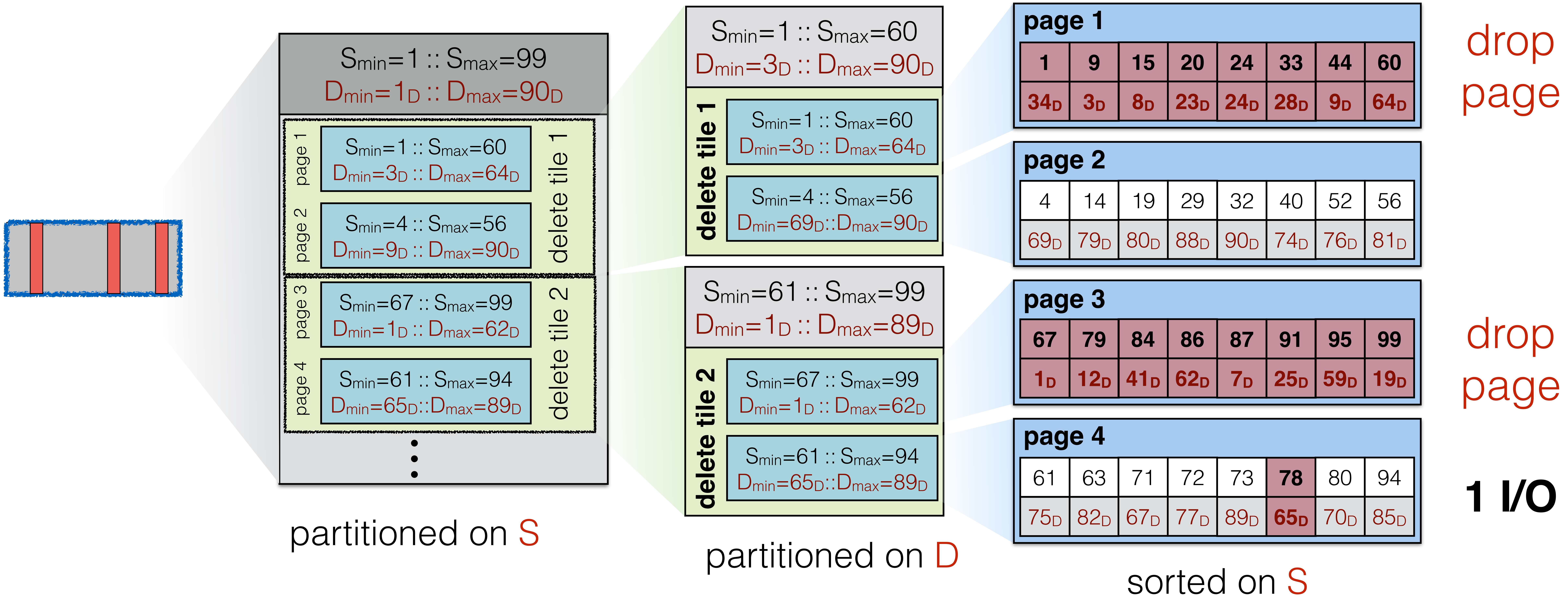
Key Weaving storage layout

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



Key Weaving storage layout

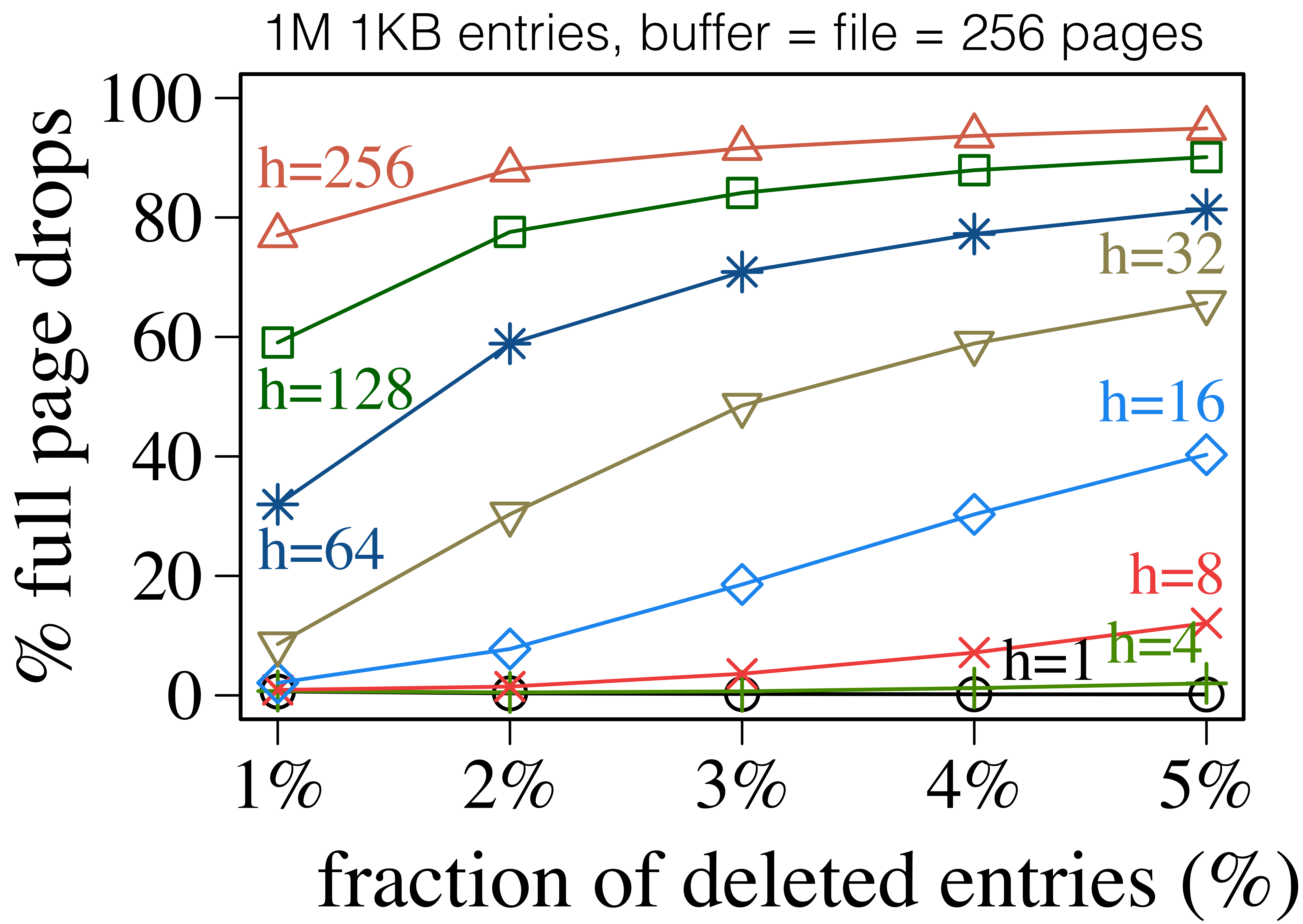
delete all entries with timestamp **<= 65$_D$**



Key Weaving storage layout

reduced latency spikes ✔

full page drops reduces superfluous I/Os ✔

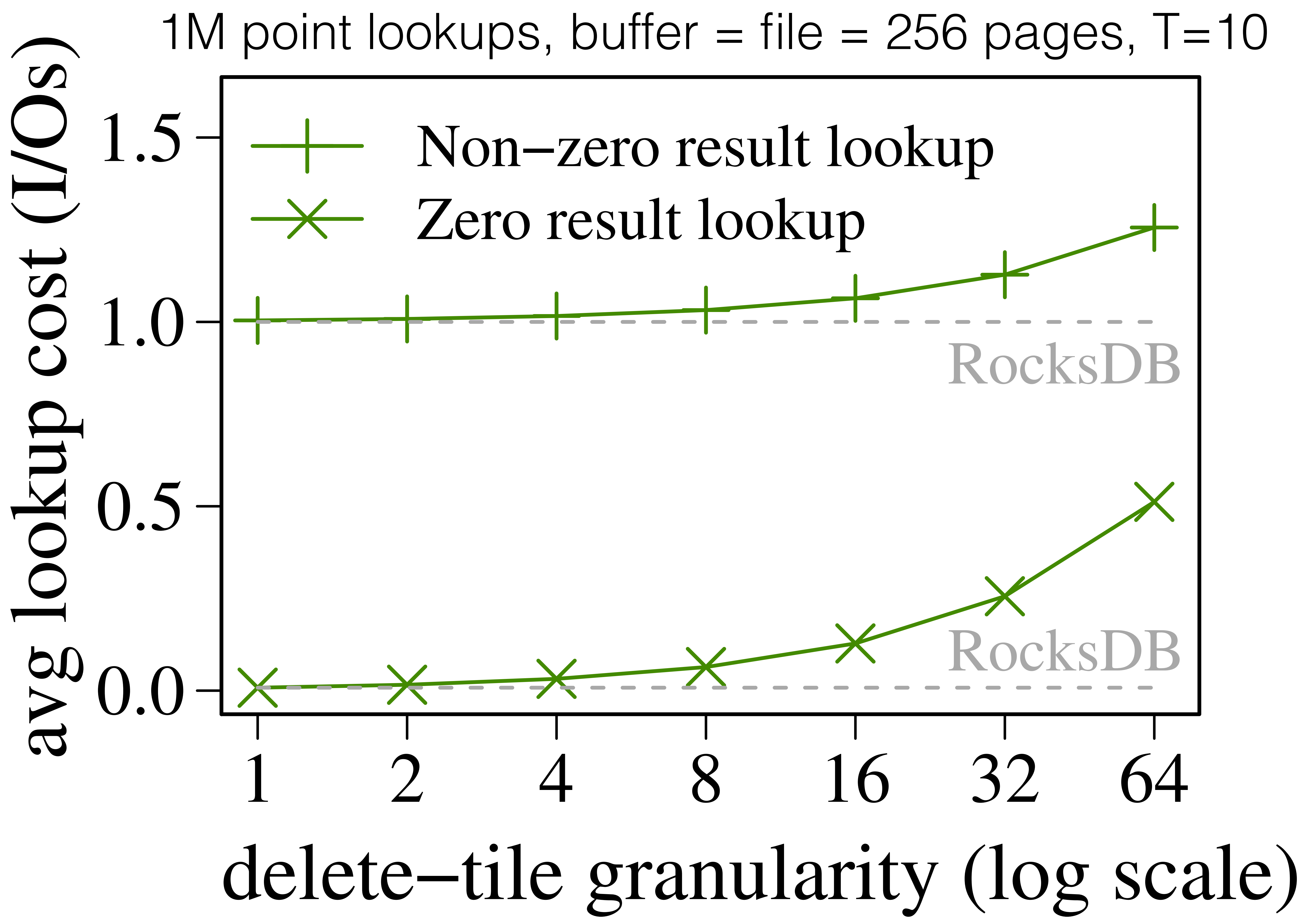


Key Weaving storage layout

higher lookup cost

reduced latency spikes

full page drops reduces superfluous I/Os



the solution

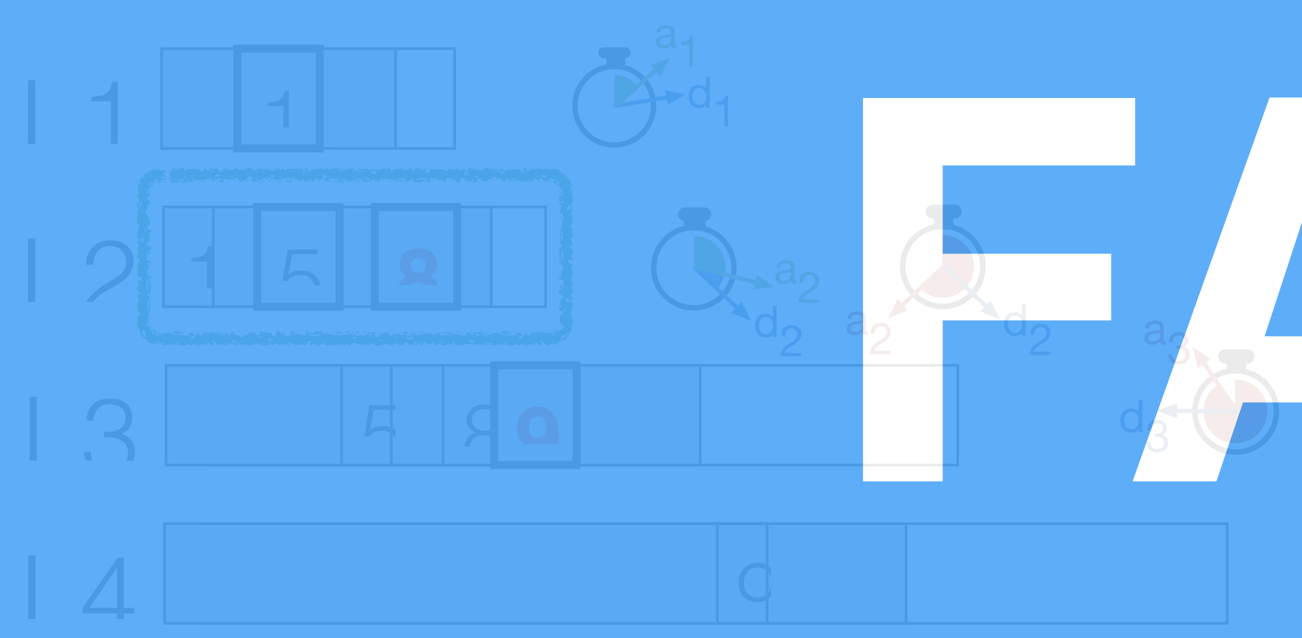
FAst DElete

amortized write

reduced space

improved read

timely delete persistence



FADE

KiWi

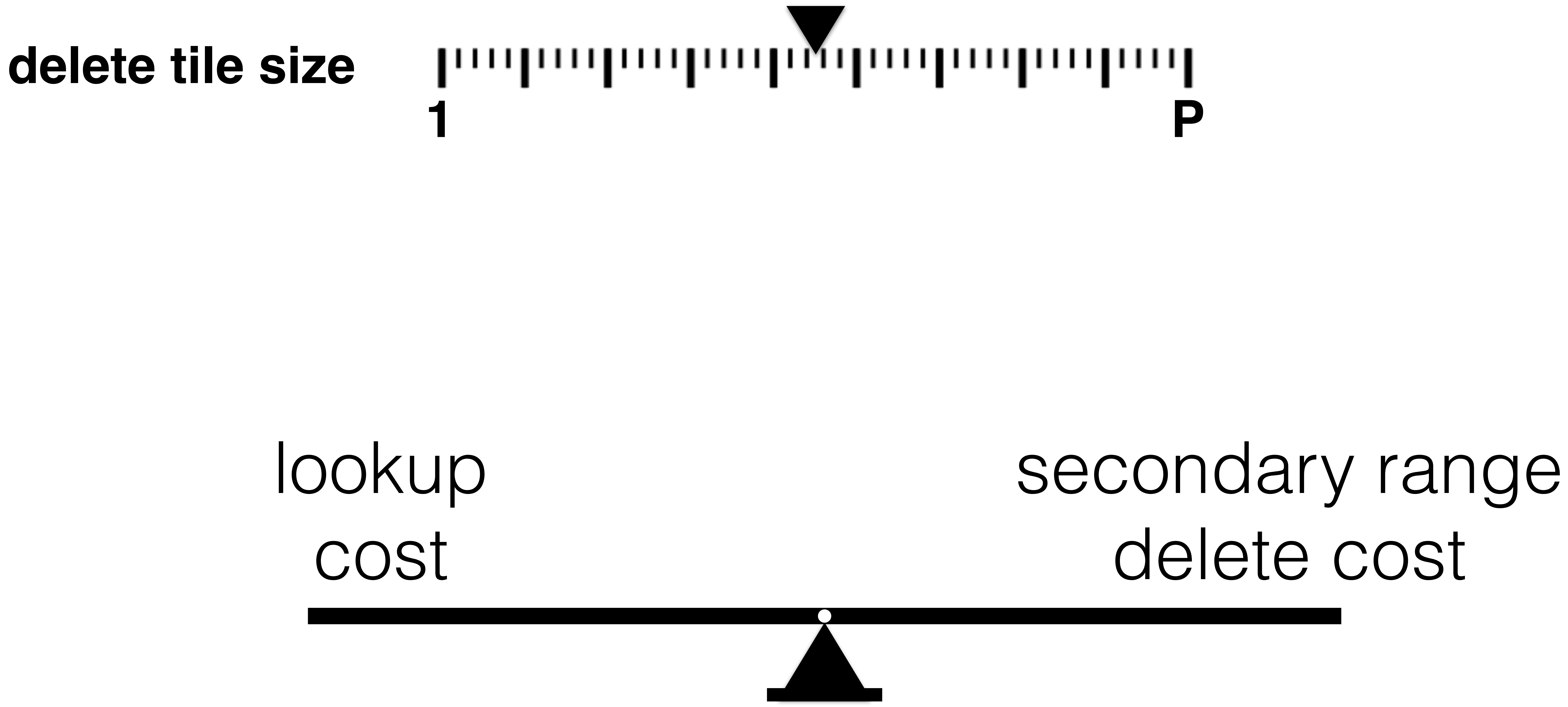
higher lookup cost

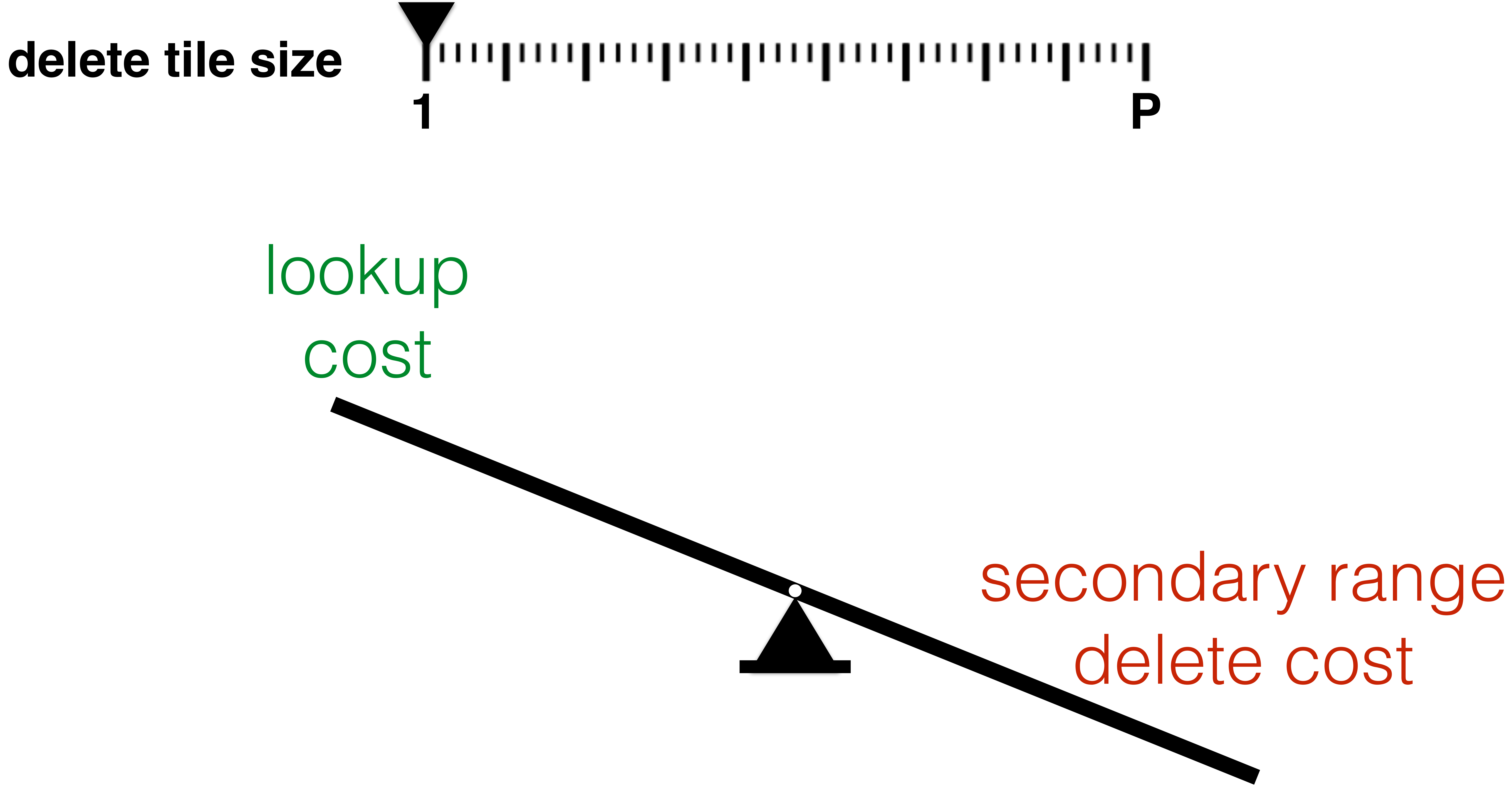
reduced latency spikes

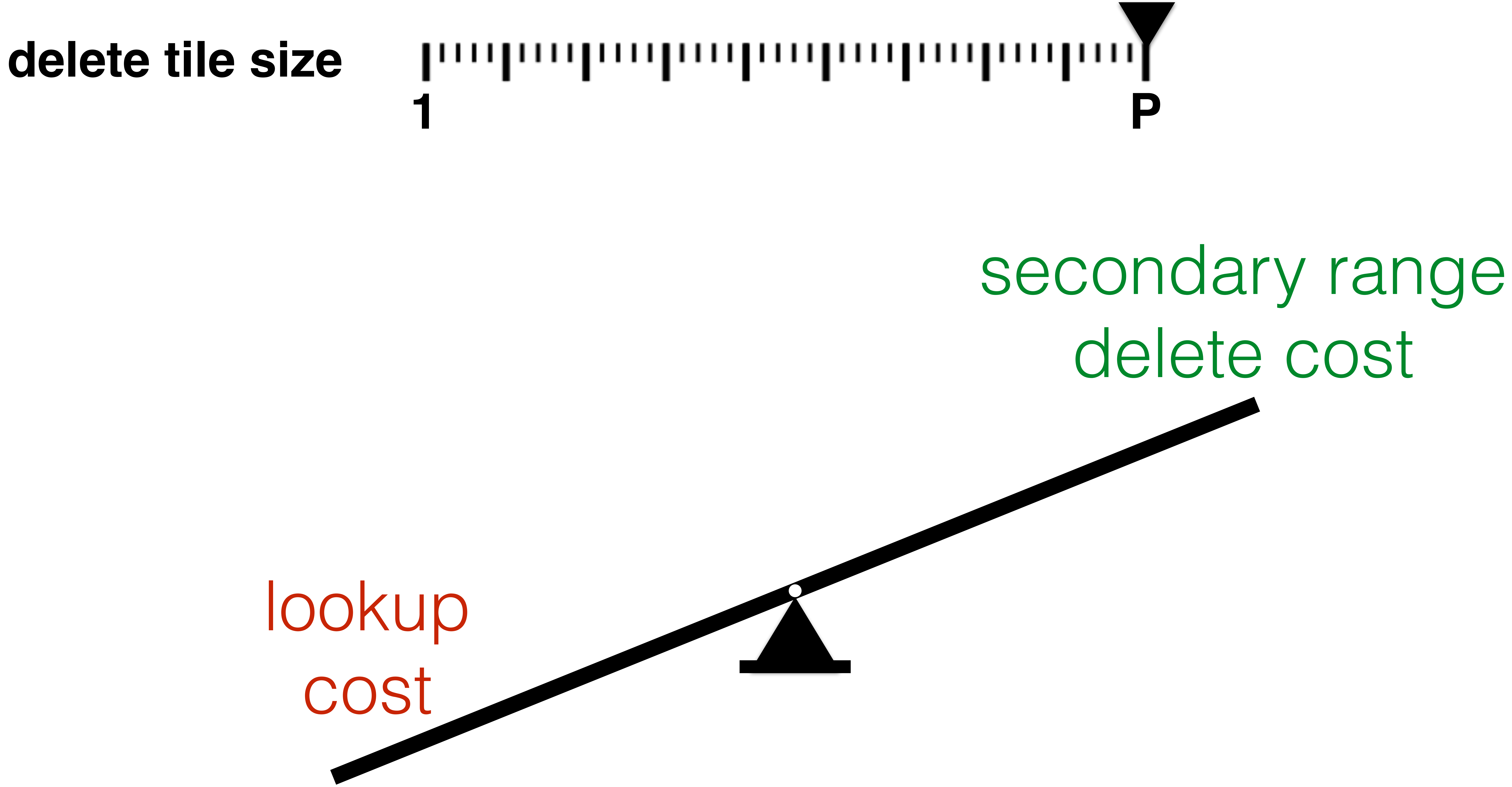
full page drops reduces superfluous I/Os the solution

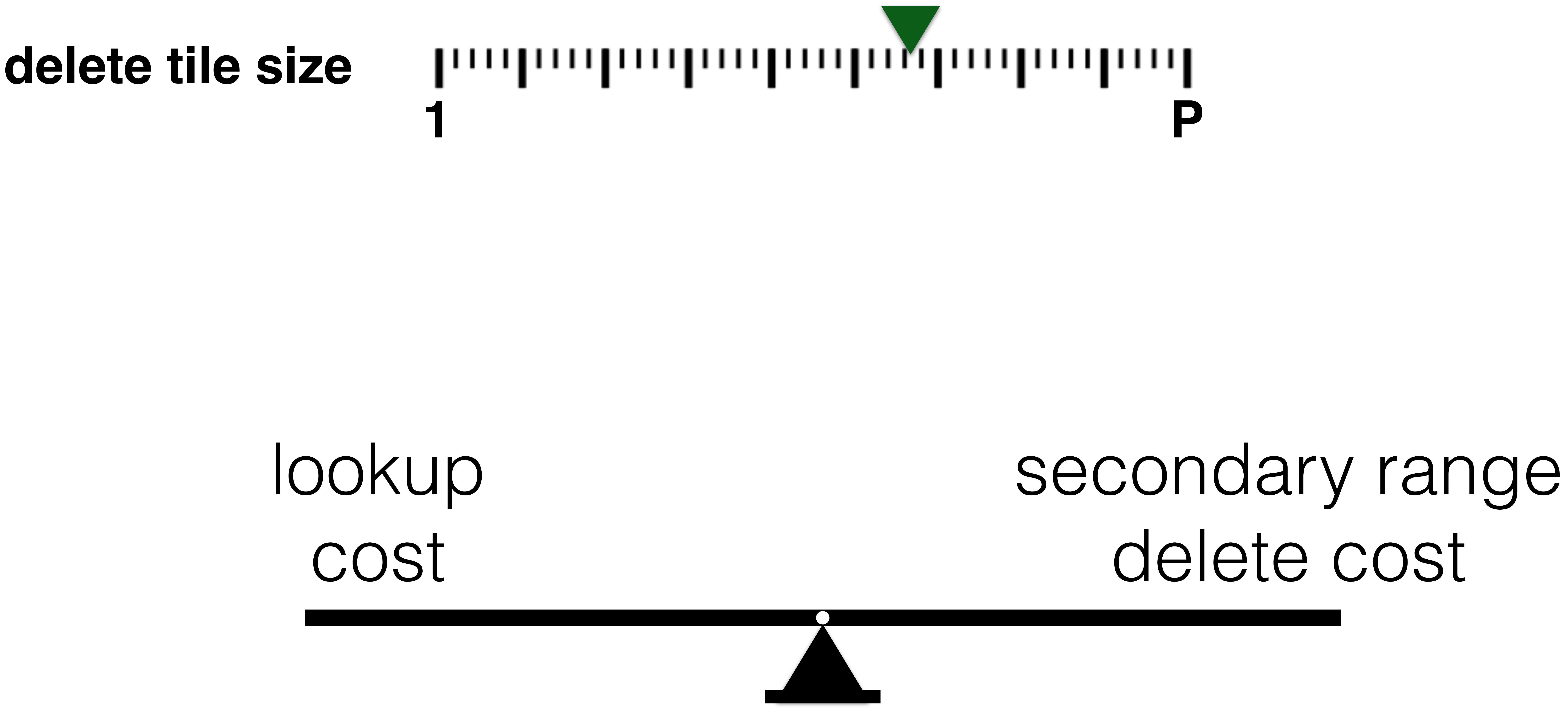
FADE

KiWi

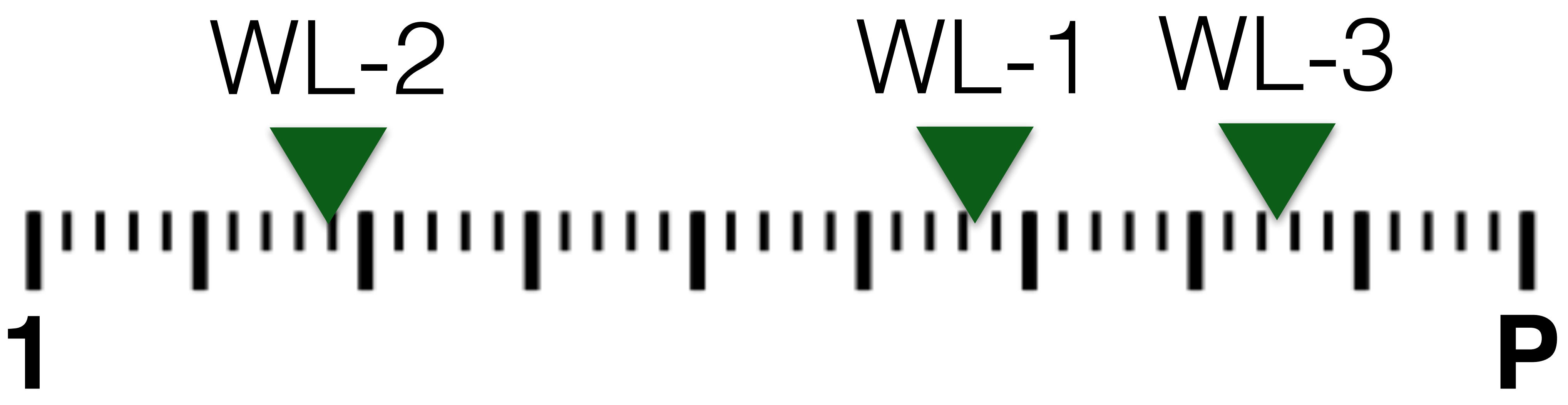

delete tile size
1
P
lookup cost
secondary range delete cost


delete tile size
1
P
lookup
cost
secondary range
delete cost


delete tile size
1
P
secondary range
delete cost
lookup
cost

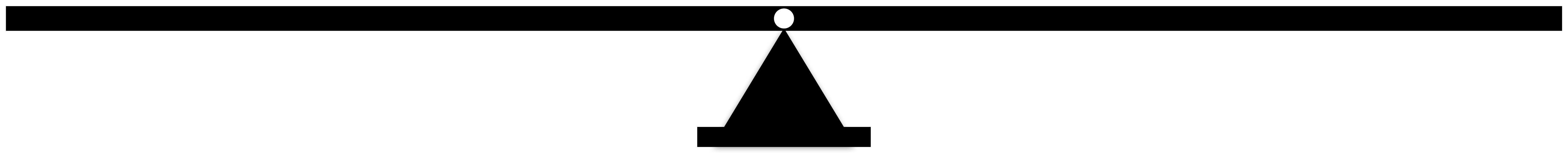

delete tile size
1
P
lookup
cost
secondary range
delete cost

delete tile size



lookup cost

secondary range delete cost



suboptimal state of the art design

for workloads with deletes

FADE persists deletes timely

using latency-driven compactions

KiWi supports efficient secondary range deletes

by key-interweaved data layout

Lethe strikes balance between cost, performance, and latency

Thank You!

BOSTON UNIVERSITY

MiDAS

midas.bu.edu/lethe