BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 6b

Deletes on LSM Trees

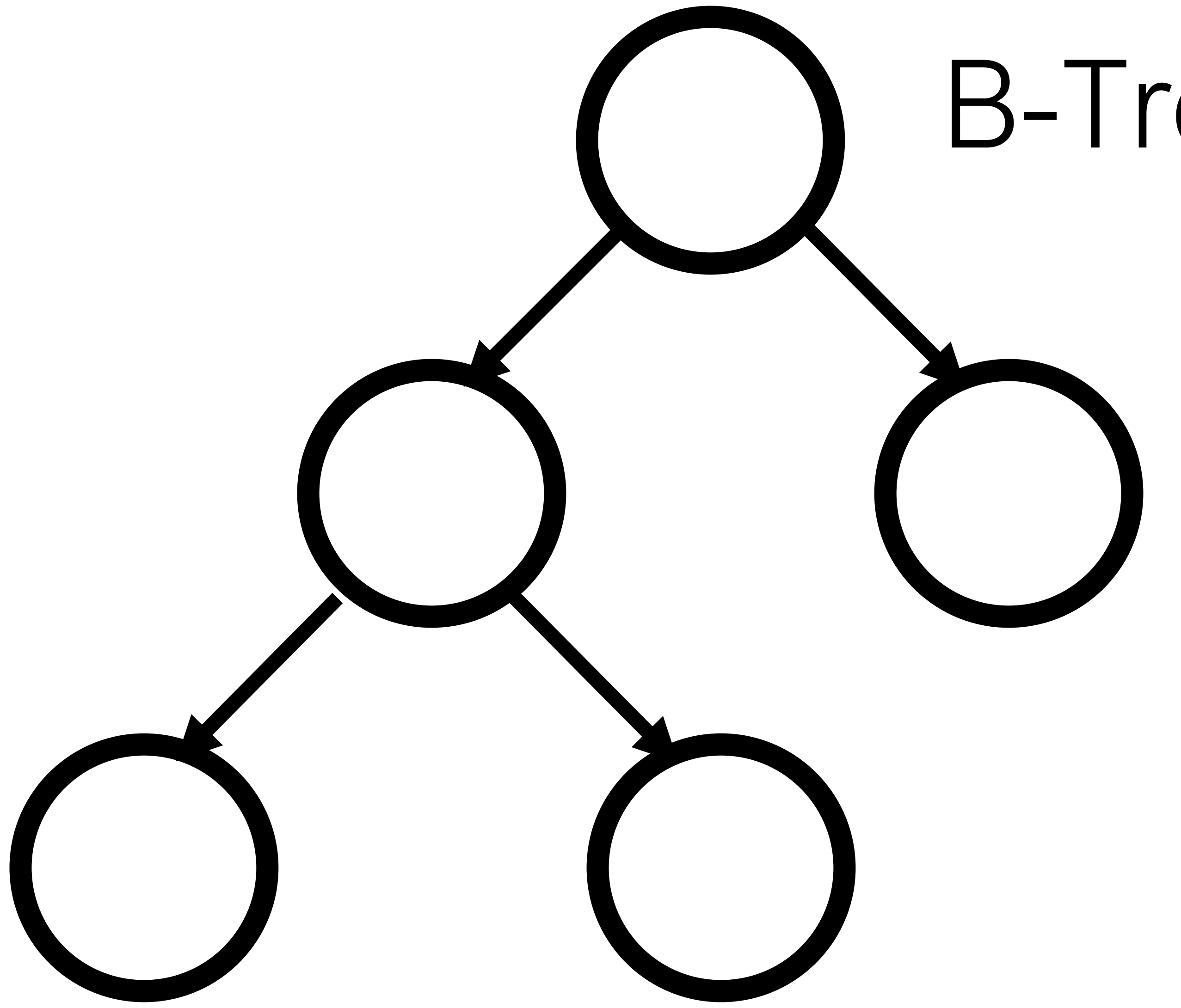
Prof. Manos Athanassoulis

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

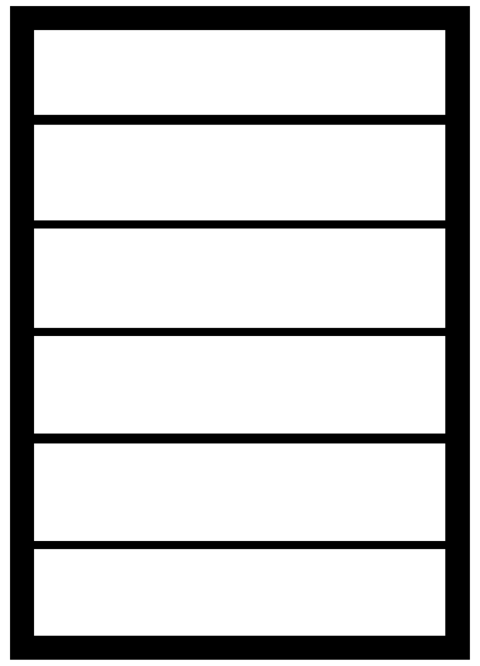
```
<your_favorite_data_structure>::delete (key)
{
  //todo
}
```

IN-PLACE

OUT-OF-PLACE



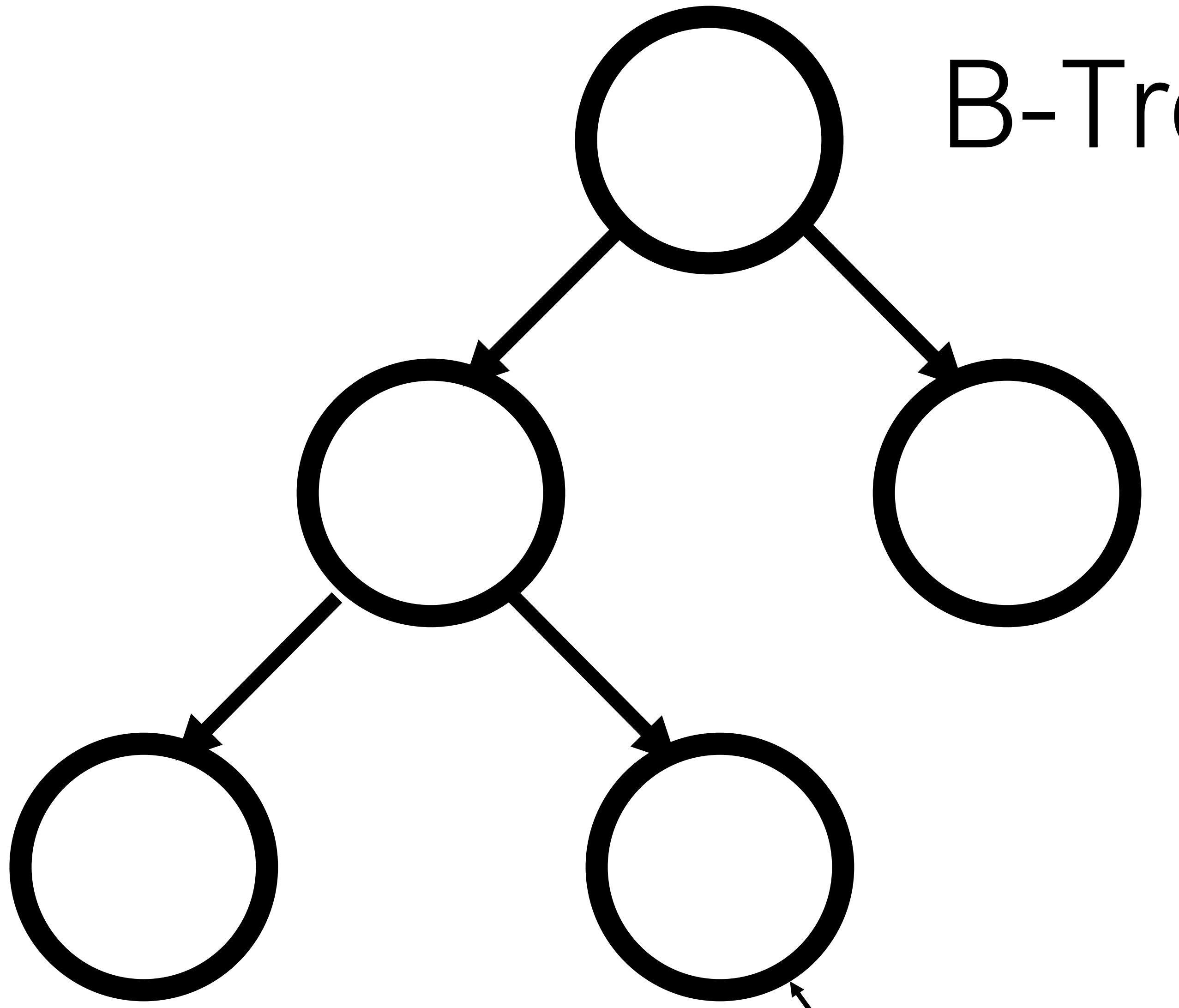
B-Trees



Heap Files
(slotted pages)

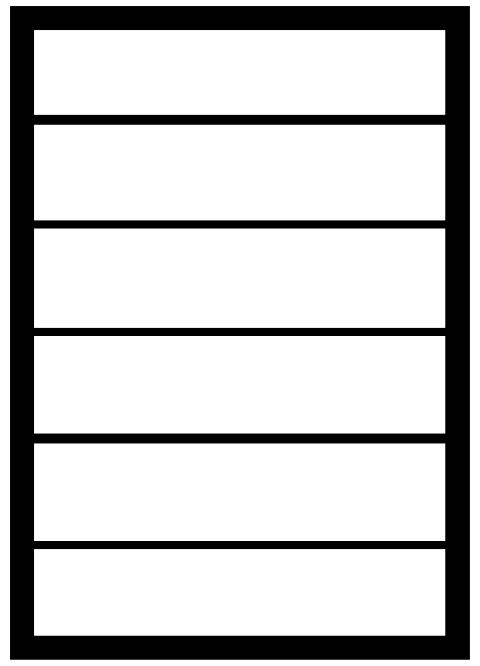
IN-PLACE

OUT-OF-PLACE



B-Trees

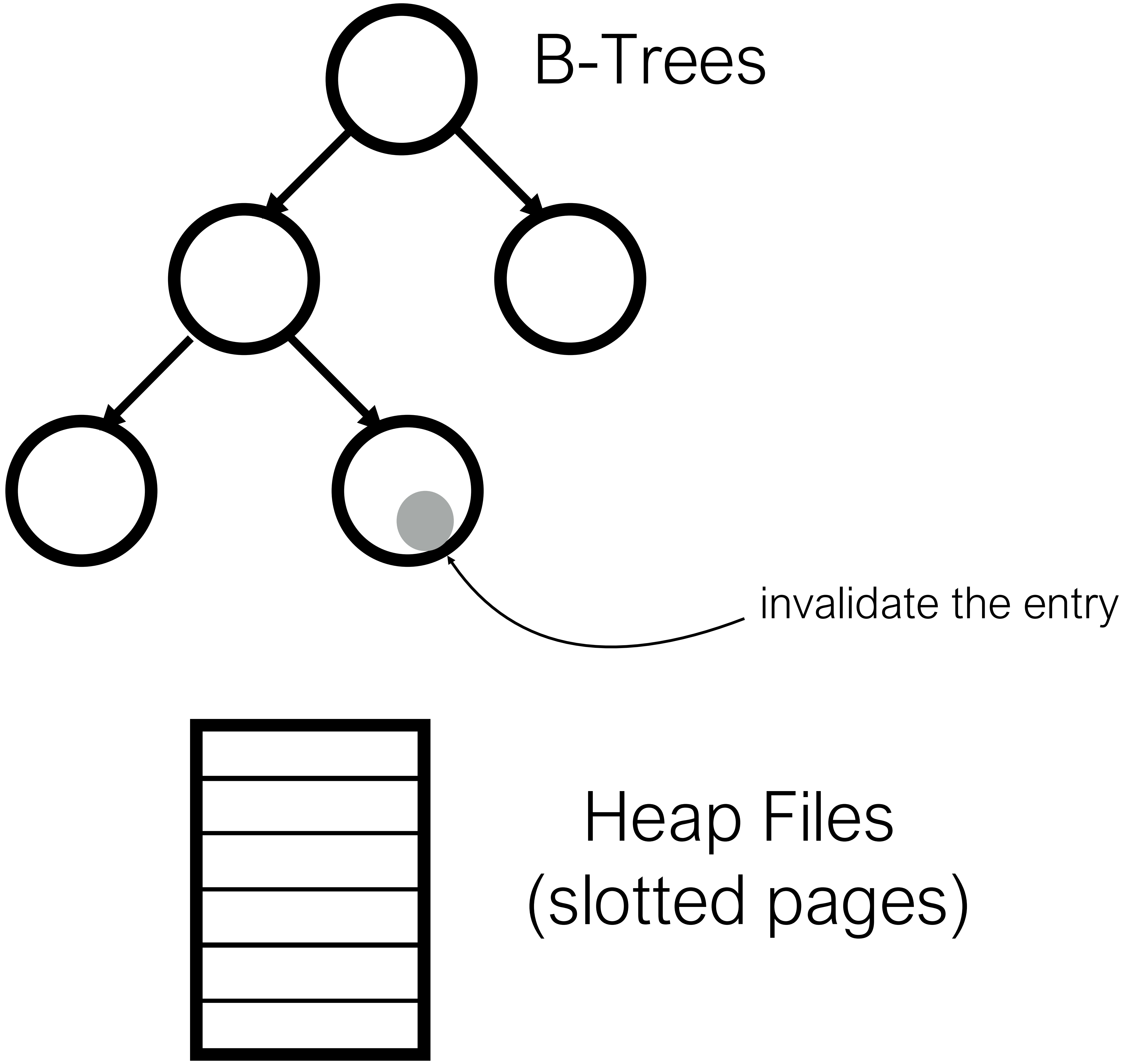
invalidate the entry



Heap Files
(slotted pages)

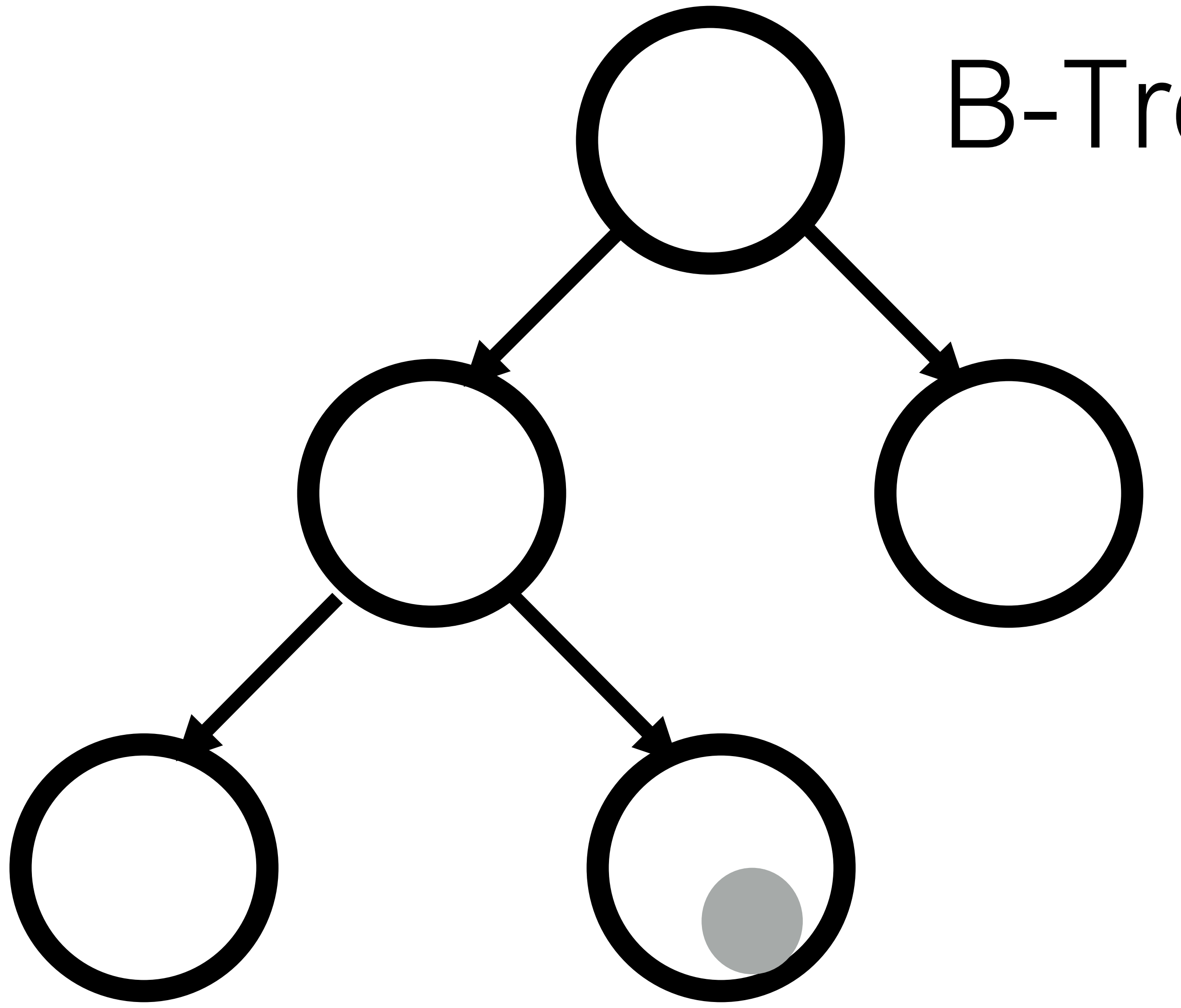
IN-PLACE

OUT-OF-PLACE

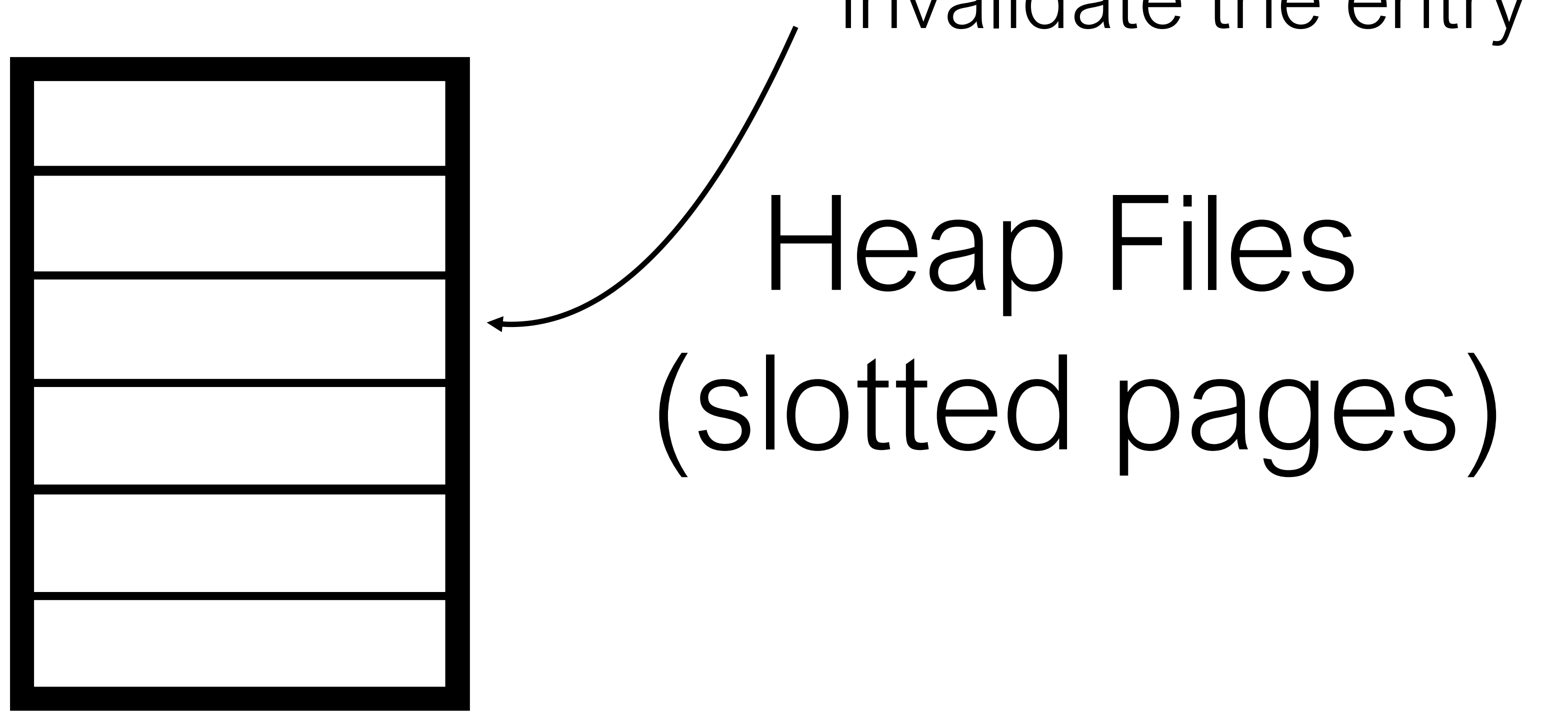


IN-PLACE

OUT-OF-PLACE



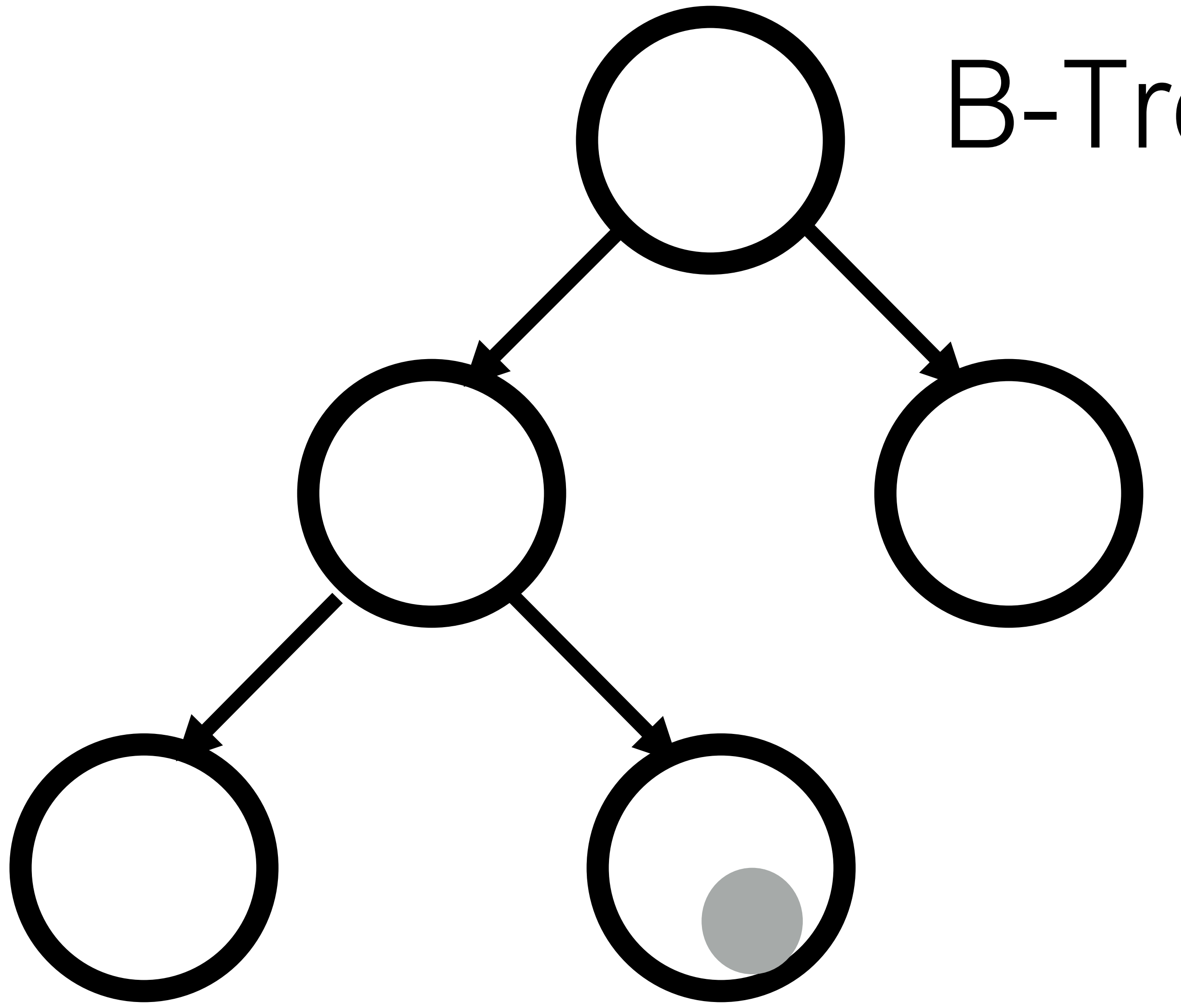
B-Trees

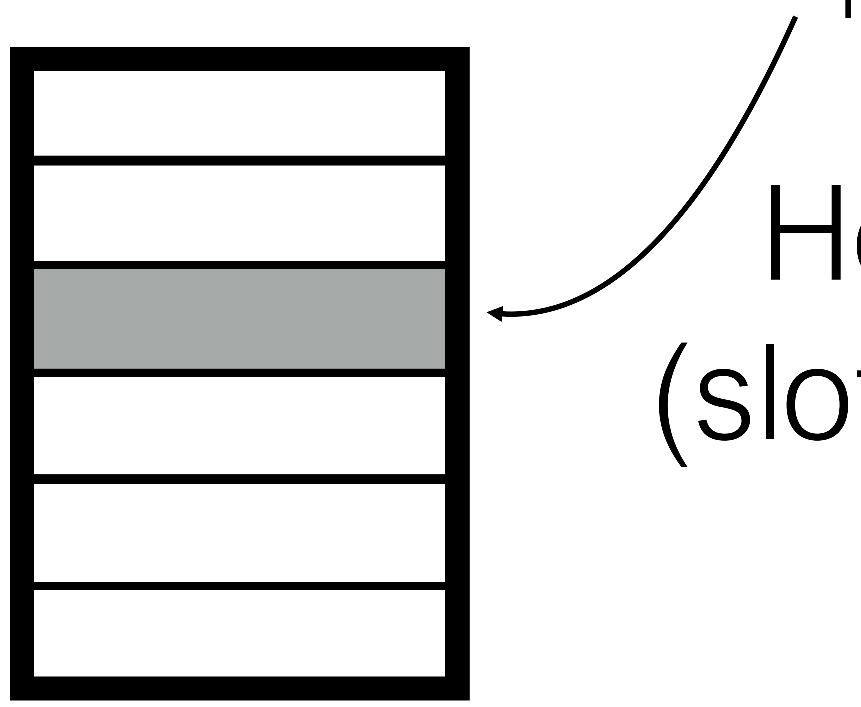


Heap Files
(slotted pages)

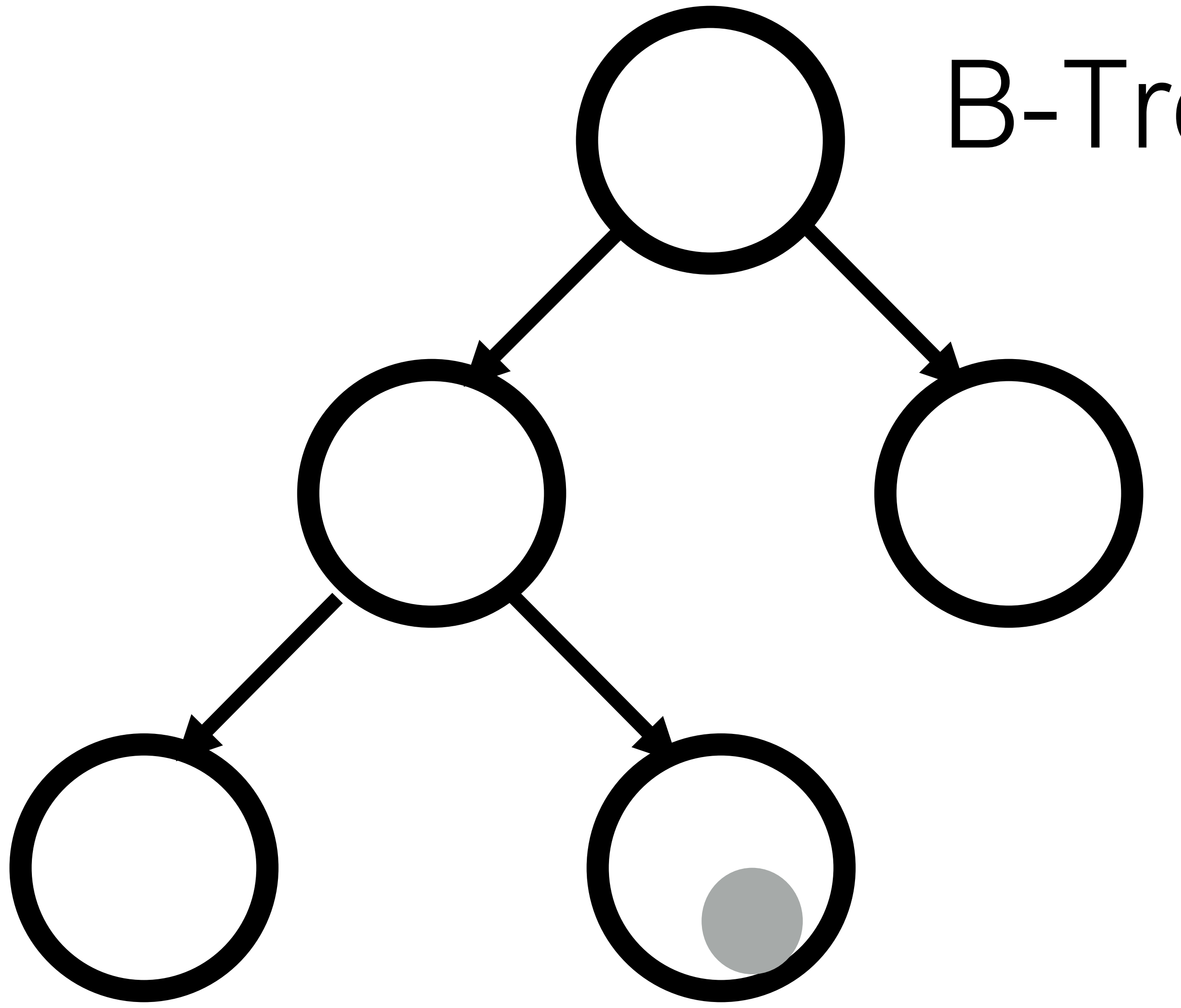
IN-PLACE

OUT-OF-PLACE





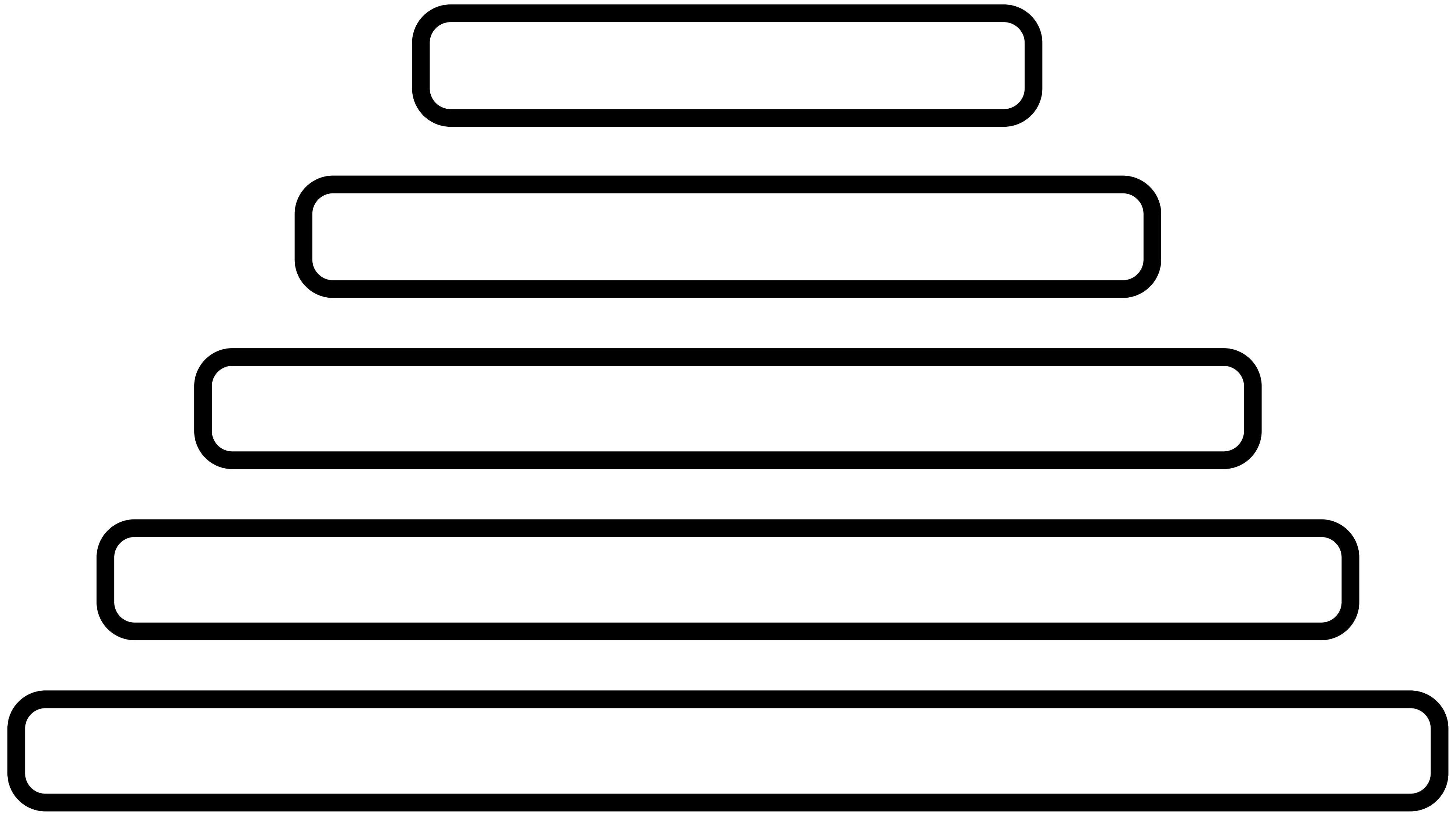
IN-PLACE



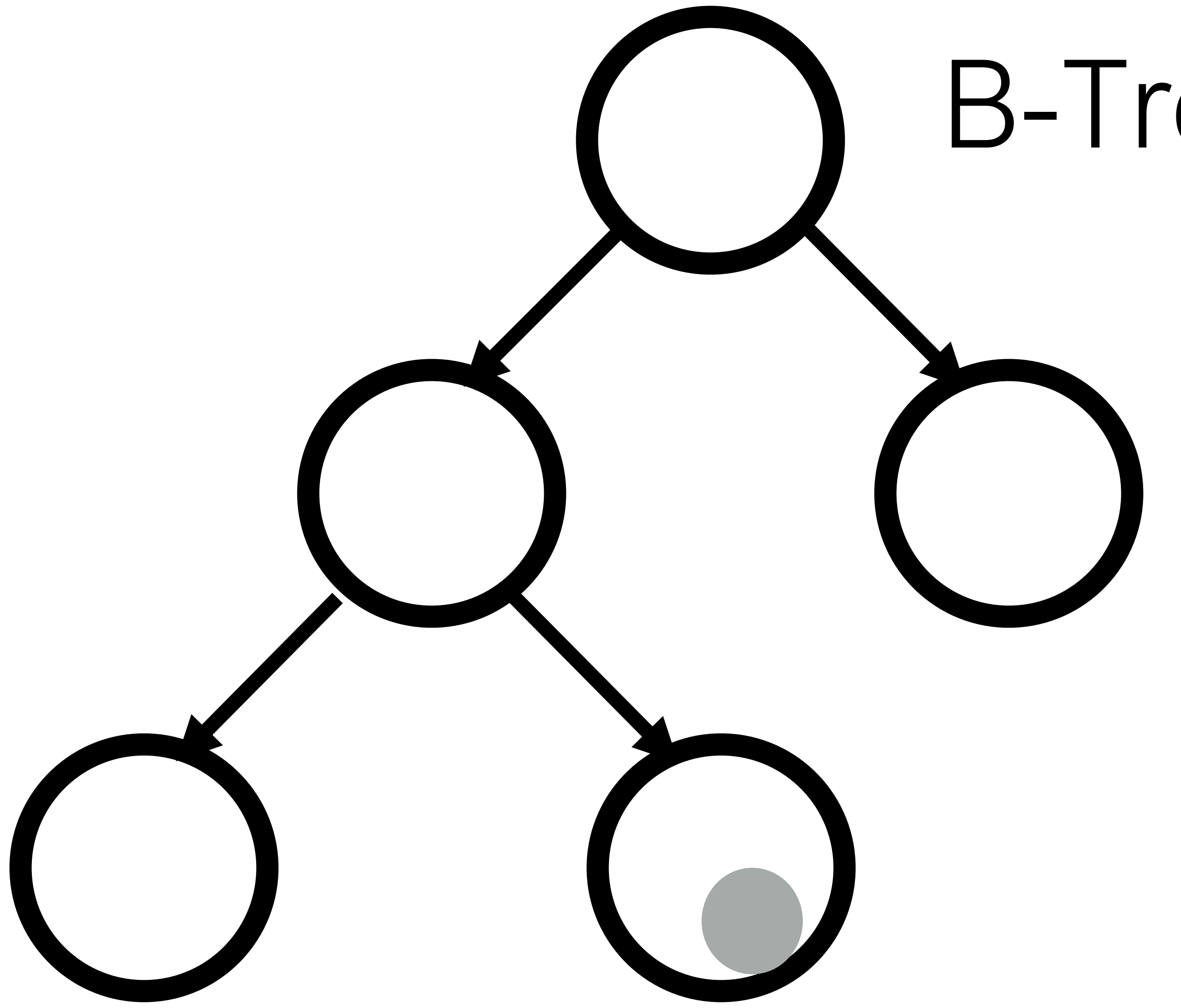
B-Trees

Heap Files
(slotted pages)

OUT-OF-PLACE



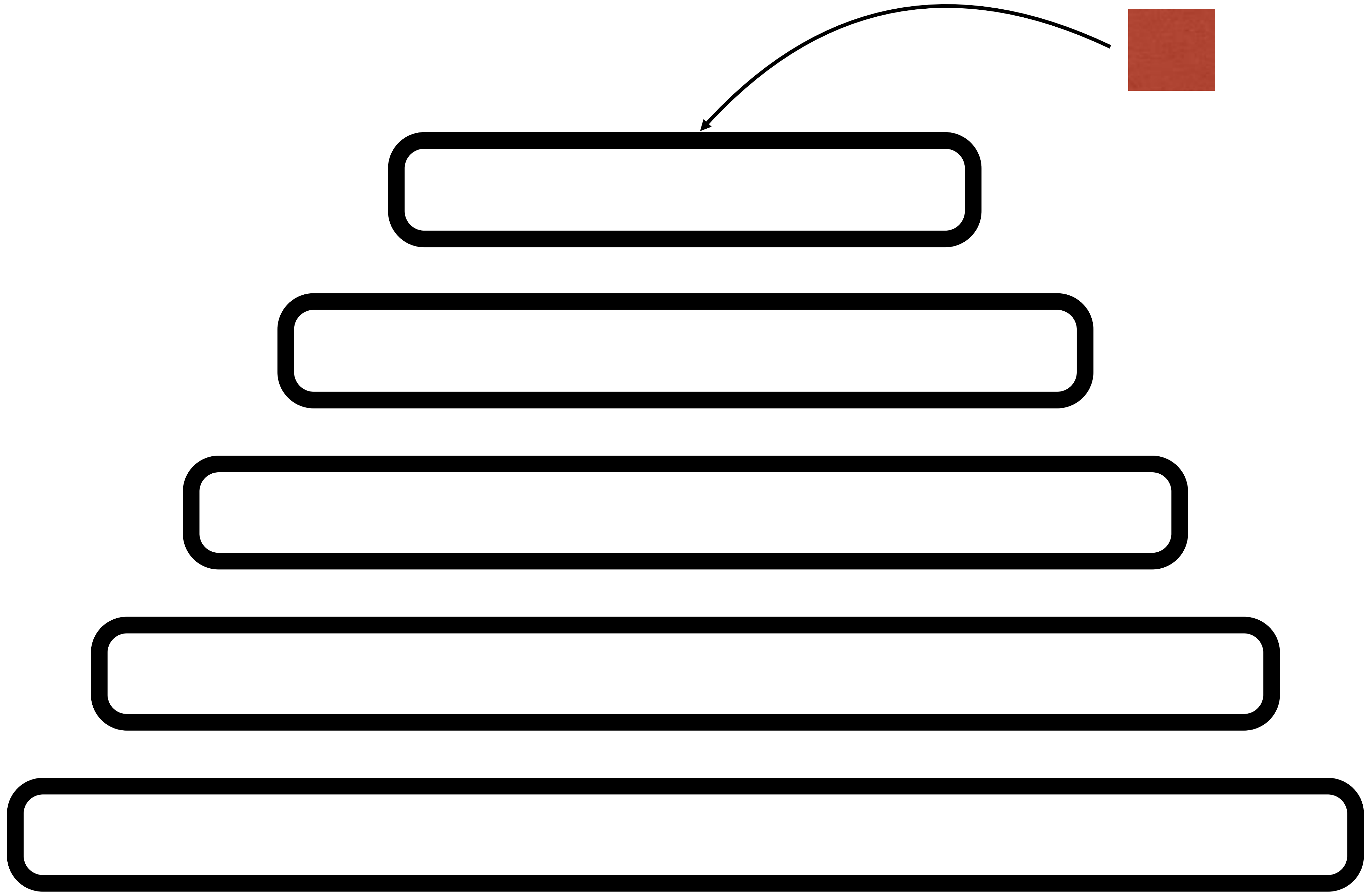
IN-PLACE



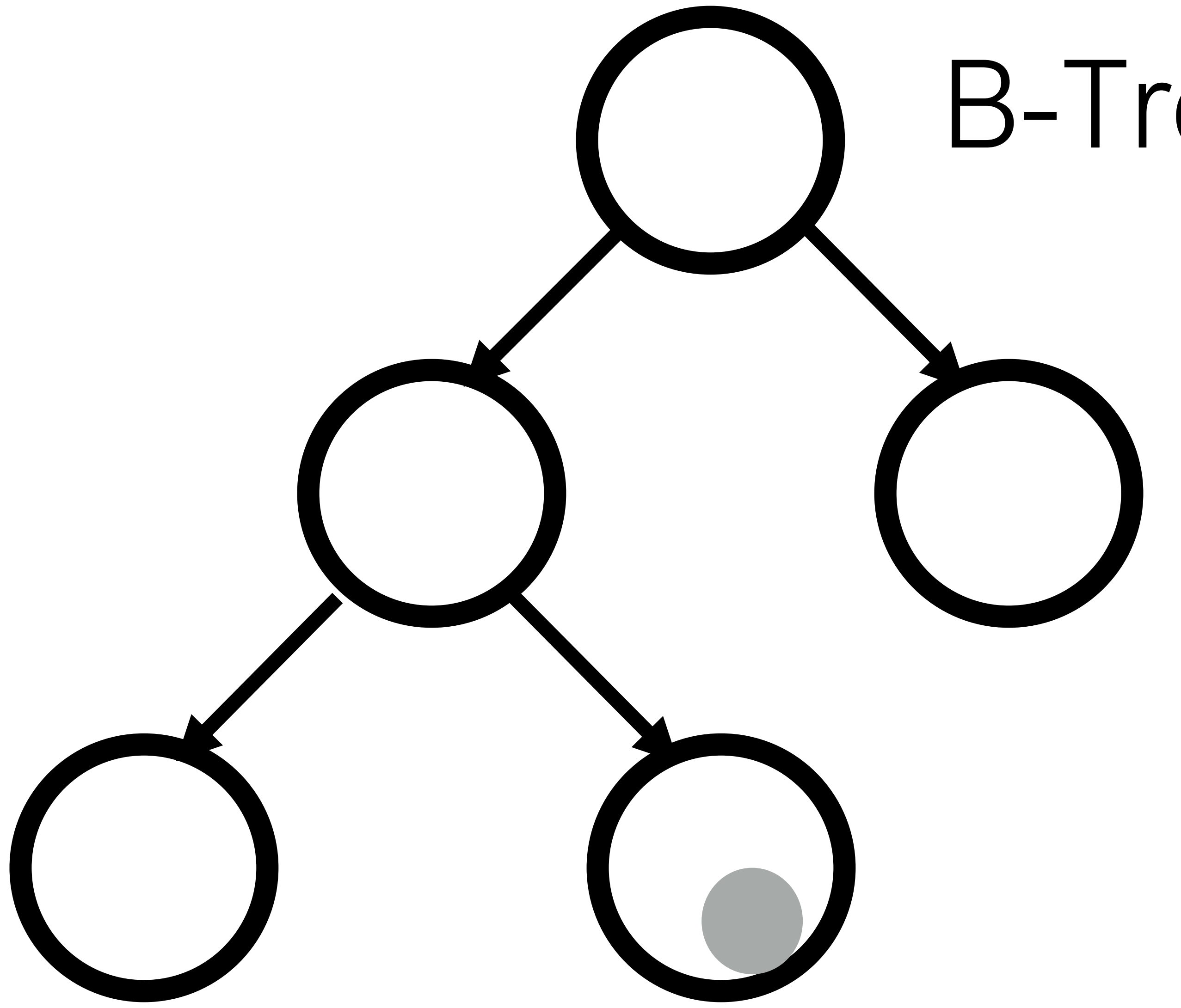
B-Trees

Heap Files
(slotted pages)

OUT-OF-PLACE



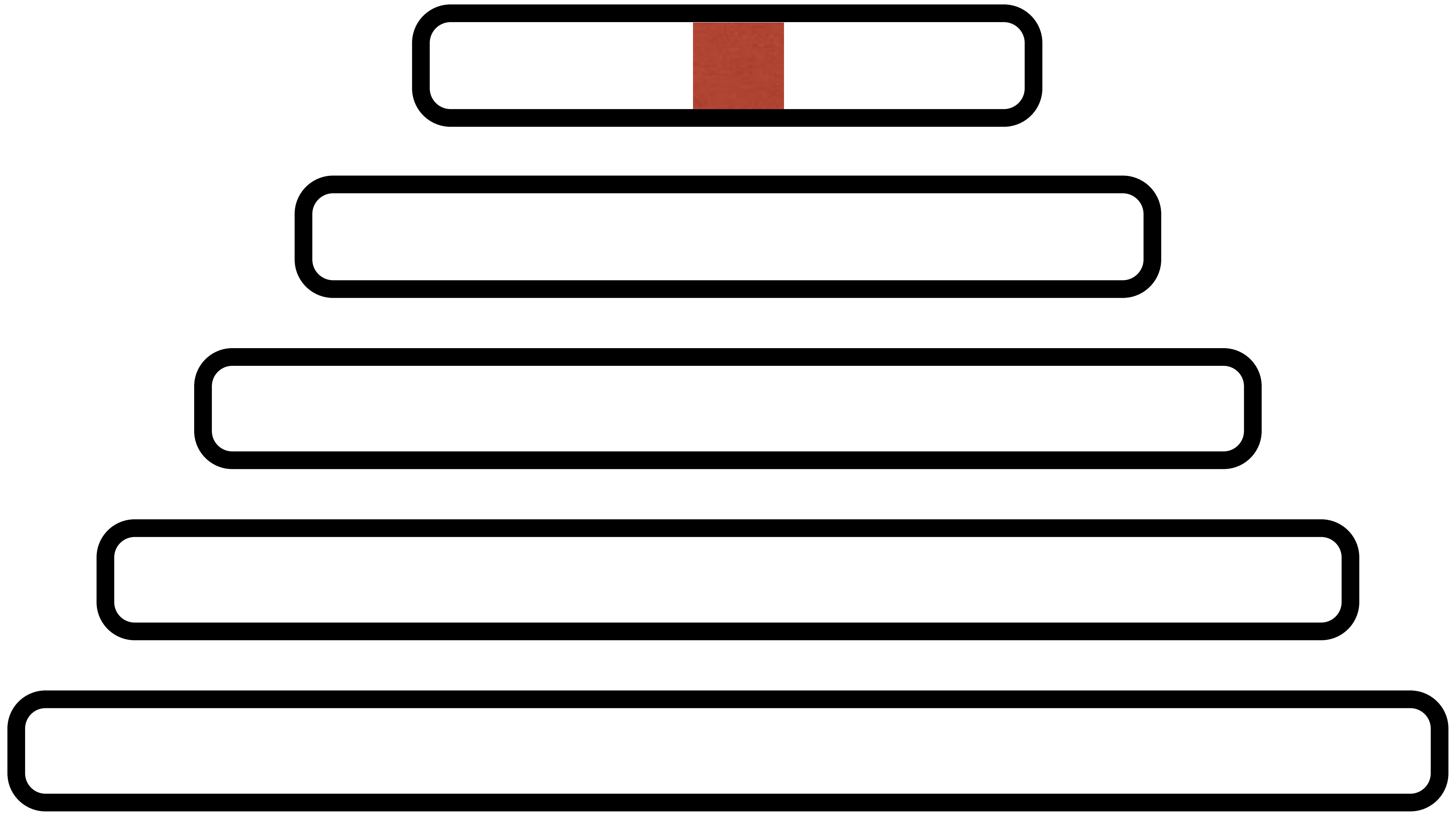
IN-PLACE



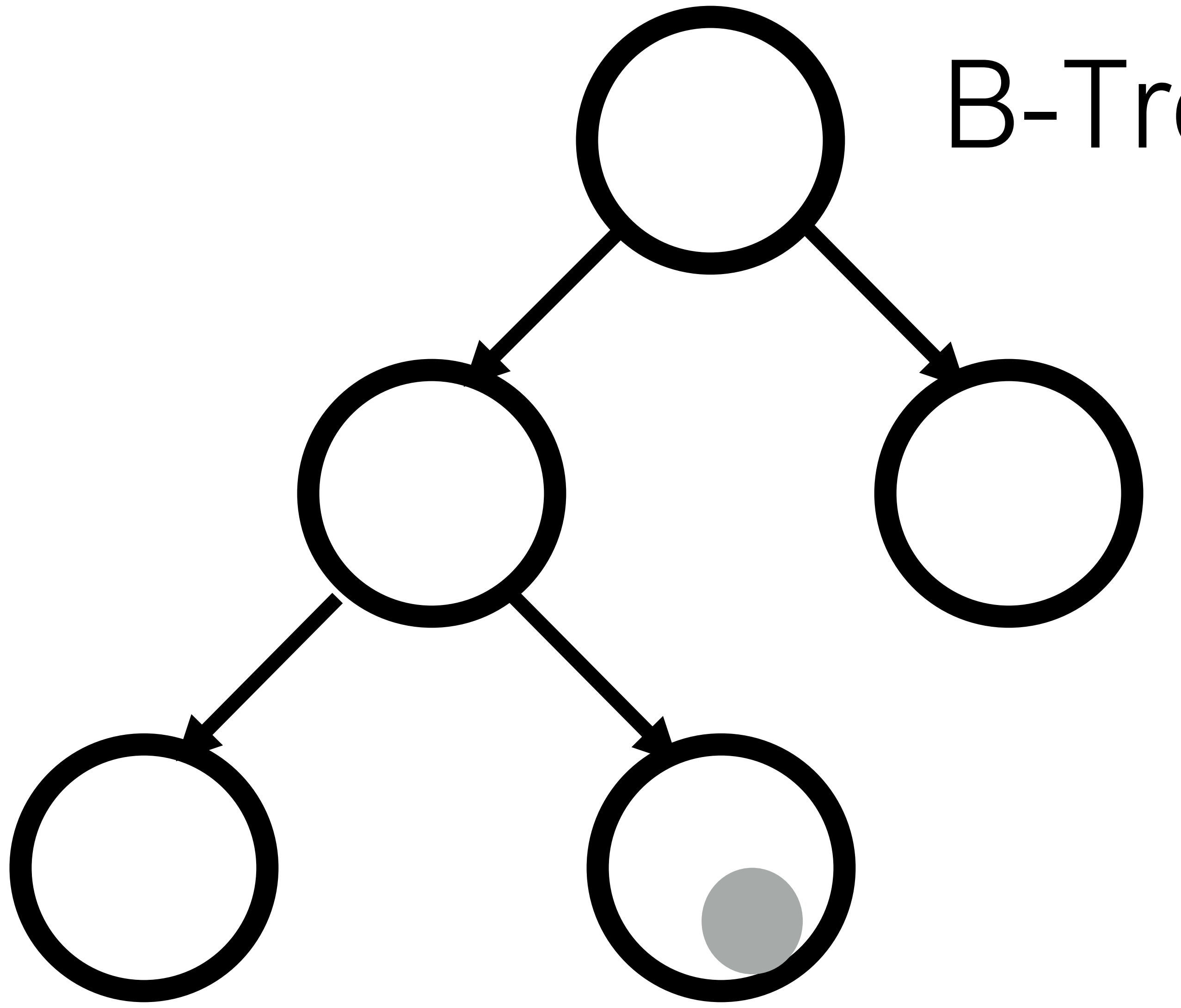
B-Trees

Heap Files
(slotted pages)

OUT-OF-PLACE



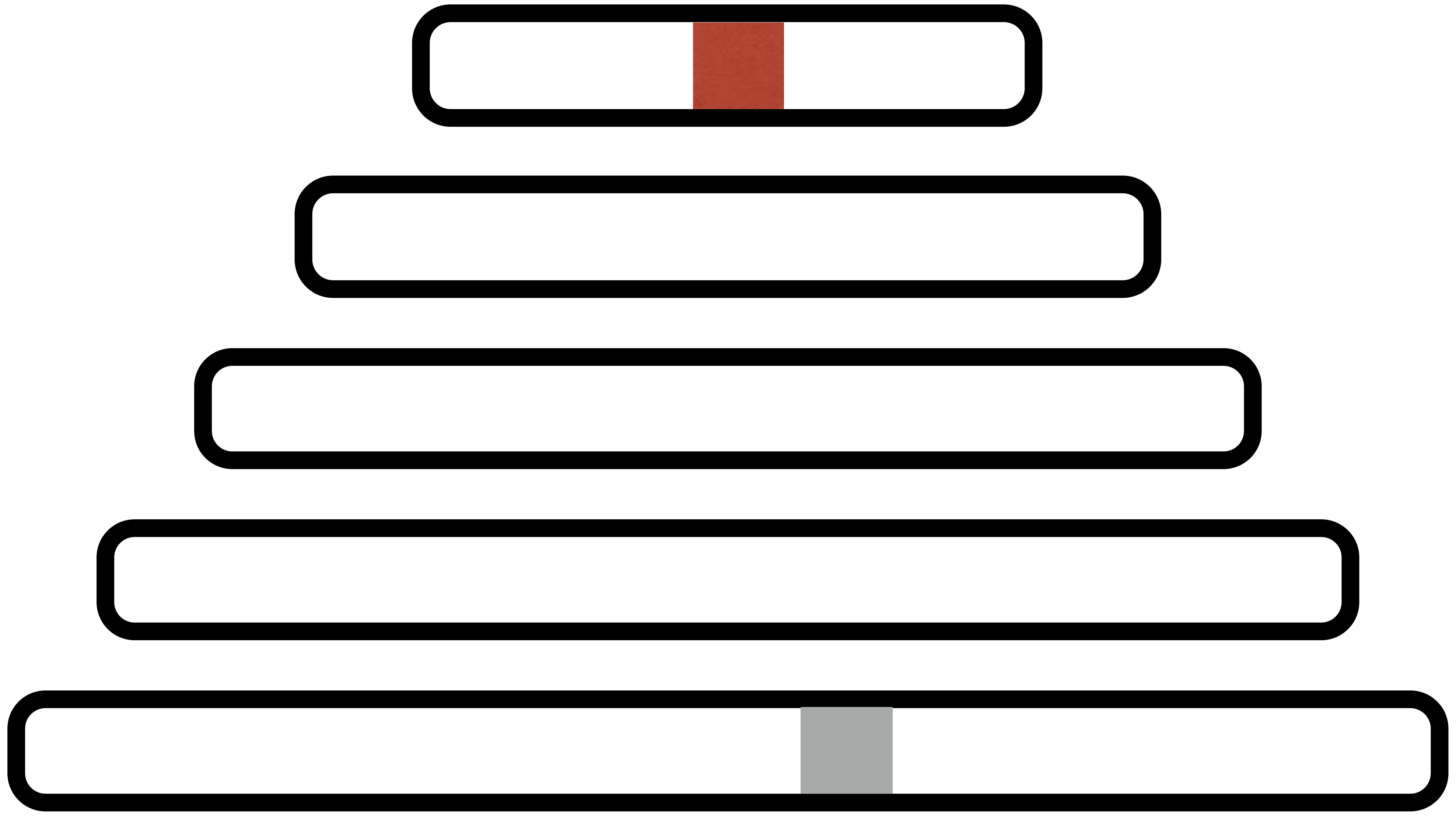
IN-PLACE



B-Trees

Heap Files
(slotted pages)

OUT-OF-PLACE



What is the delete tradeoff?

What is the delete tradeoff?

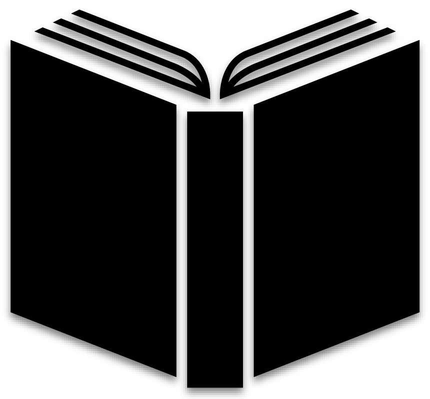
read

vs.

write

?

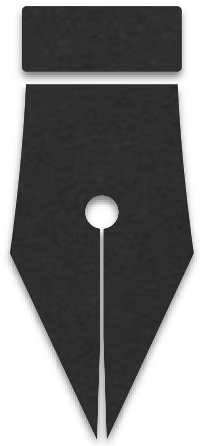
What is the delete tradeoff?



read

vs.

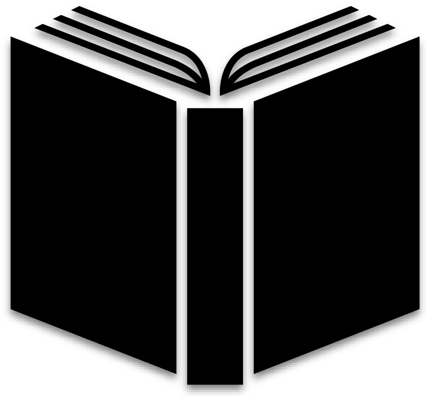
?



write

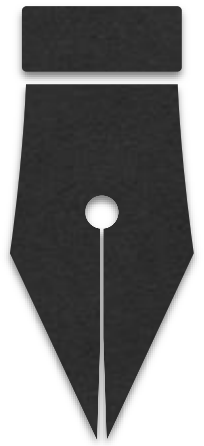
Deletes are almost **exclusively** *logical*

What is the delete tradeoff?



read

vs.



write

?

Deletes are almost **exclusively** *logical*

b-trees, slotted pages, LSM-trees **invalidate** the entry under deletion

delete tradeoffs

delete latency vs. ***future*** read performance

delete tradeoffs

delete latency vs. ***future*** read performance

e.g., tree re-org, page re-org, more metadata in LSM

delete tradeoffs

delete latency vs. ***future*** read performance

e.g., tree re-org, page re-org, more metadata in LSM

delete latency vs. data ***privacy***

delete tradeoffs

delete latency vs. ***future*** read performance

e.g., tree re-org, page re-org, more metadata in LSM

delete latency vs. data ***privacy***

logical deletes keep around deleted entries, what if they leak?

delete tradeoffs

delete latency vs. ***future*** read performance

e.g., tree re-org, page re-org, more metadata in LSM

delete latency vs. data ***privacy***

logical deletes keep around deleted entries, what if they leak?

delete latency vs. ***storage amplification***

delete tradeoffs

delete latency vs. ***future*** read performance

e.g., tree re-org, page re-org, more metadata in LSM

delete latency vs. data ***privacy***

logical deletes keep around deleted entries, what if they leak?

delete latency vs. ***storage amplification***

logical deletes keep around deleted entries and metadata!!

delete tradeoffs

delete latency vs. ***future*** read performance

e.g., tree re-org, page re-org, more metadata in LSM

delete latency vs. data ***privacy***

logical deletes keep around deleted entries, what if they leak?

`what if we persisted the deletes immediately?`

delete tradeoffs

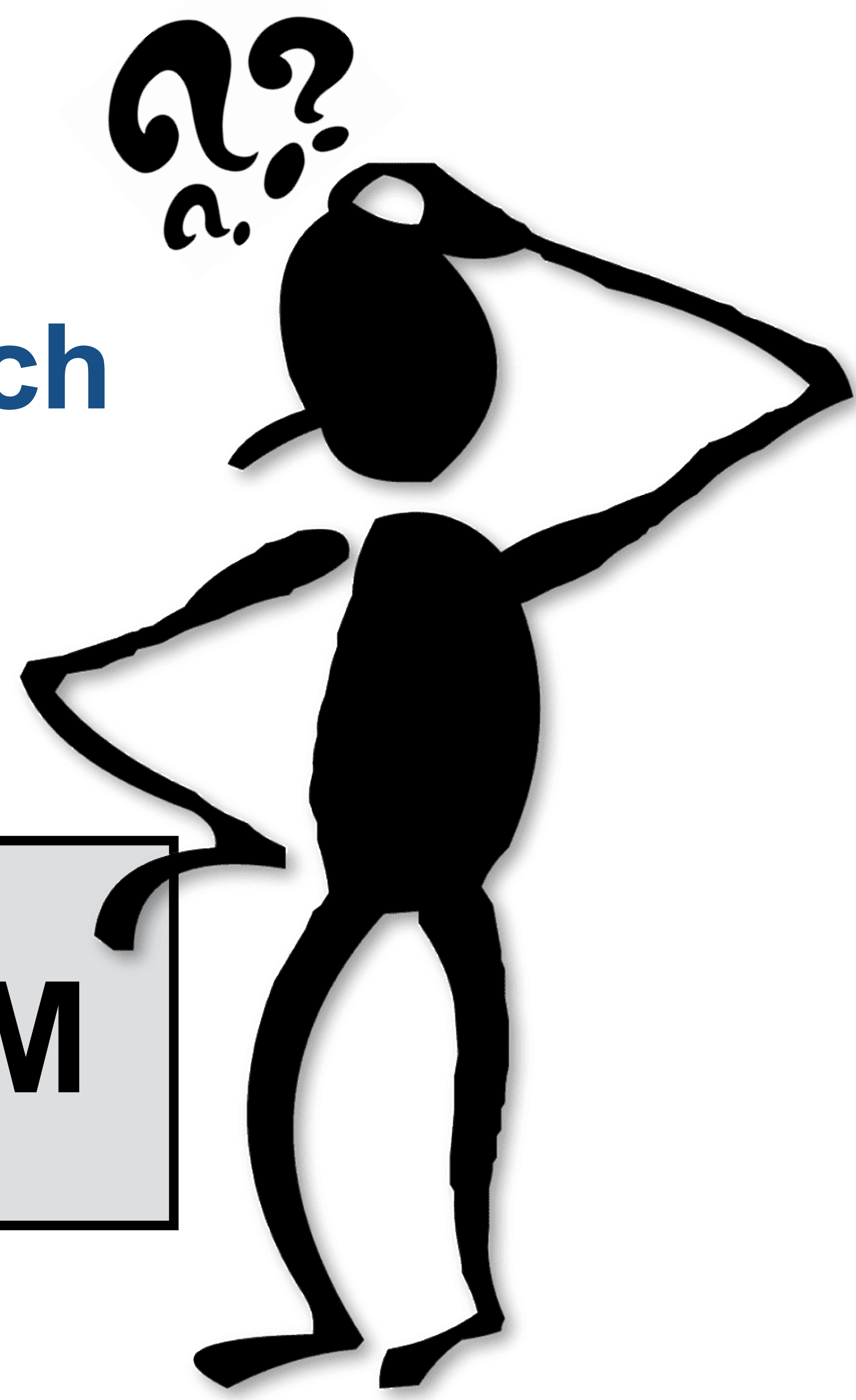
delete latency vs. **persistent** delete latency

delete tradeoffs

logical delete latency vs. **persistent** delete latency

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

TechCrunch
Feb '19

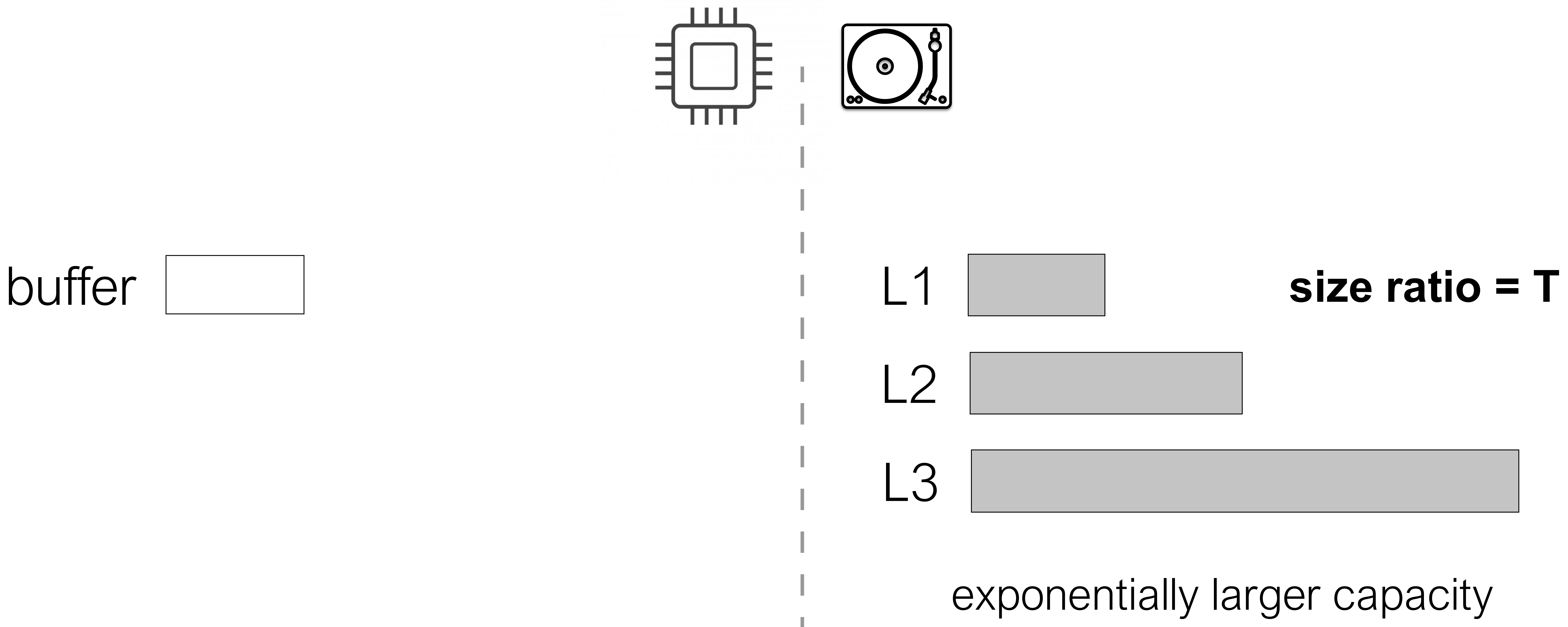


Small Datum
Jan '20

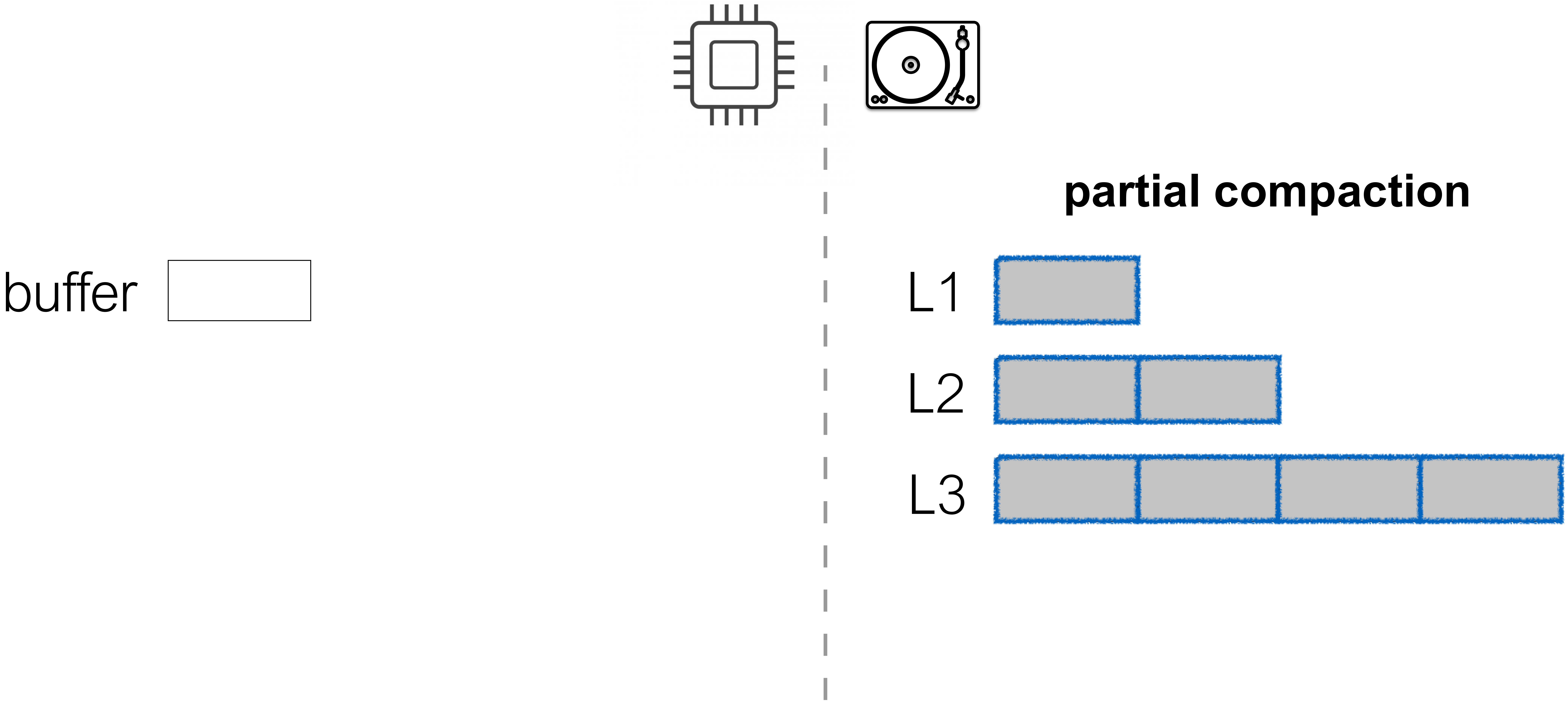
Deletes are fast and slow in an LSM

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

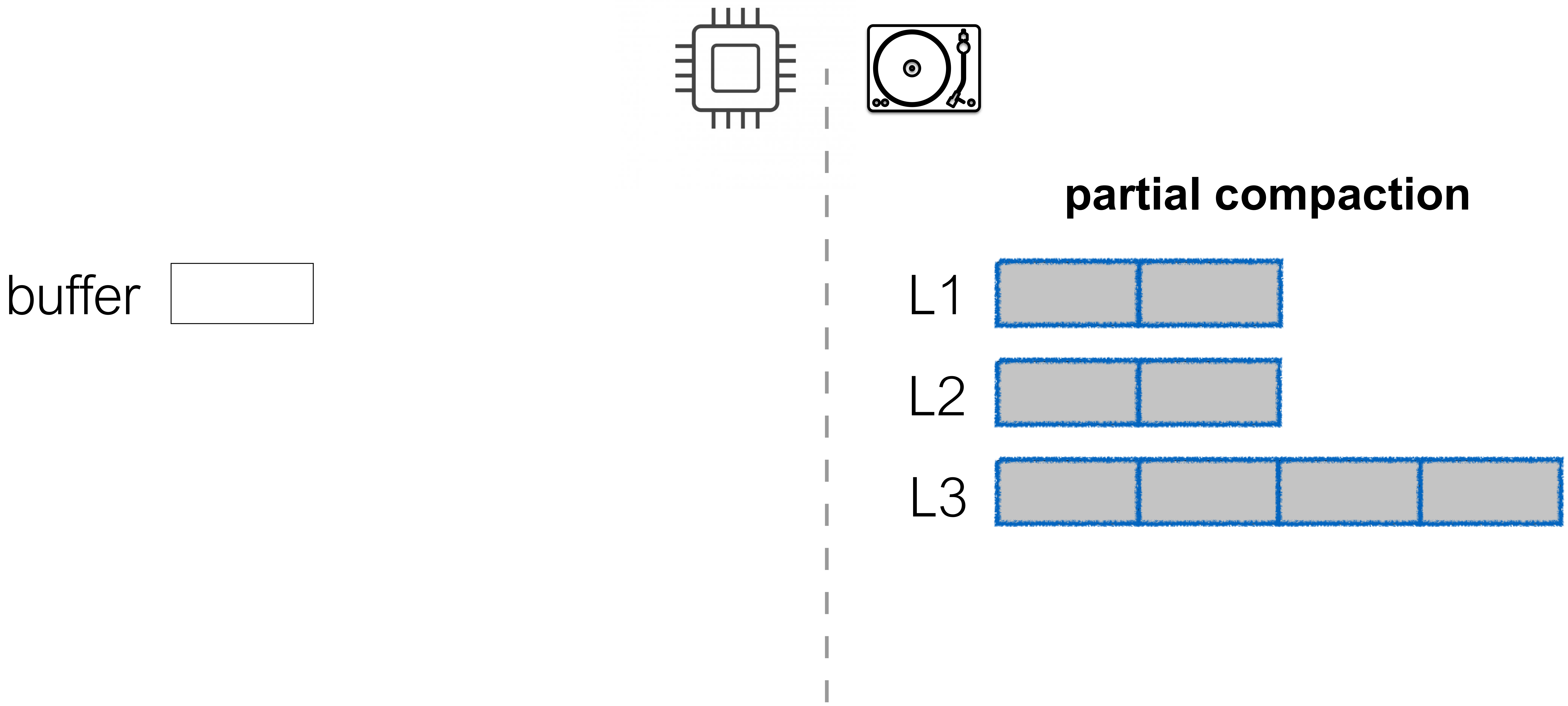
log-structured merge-tree



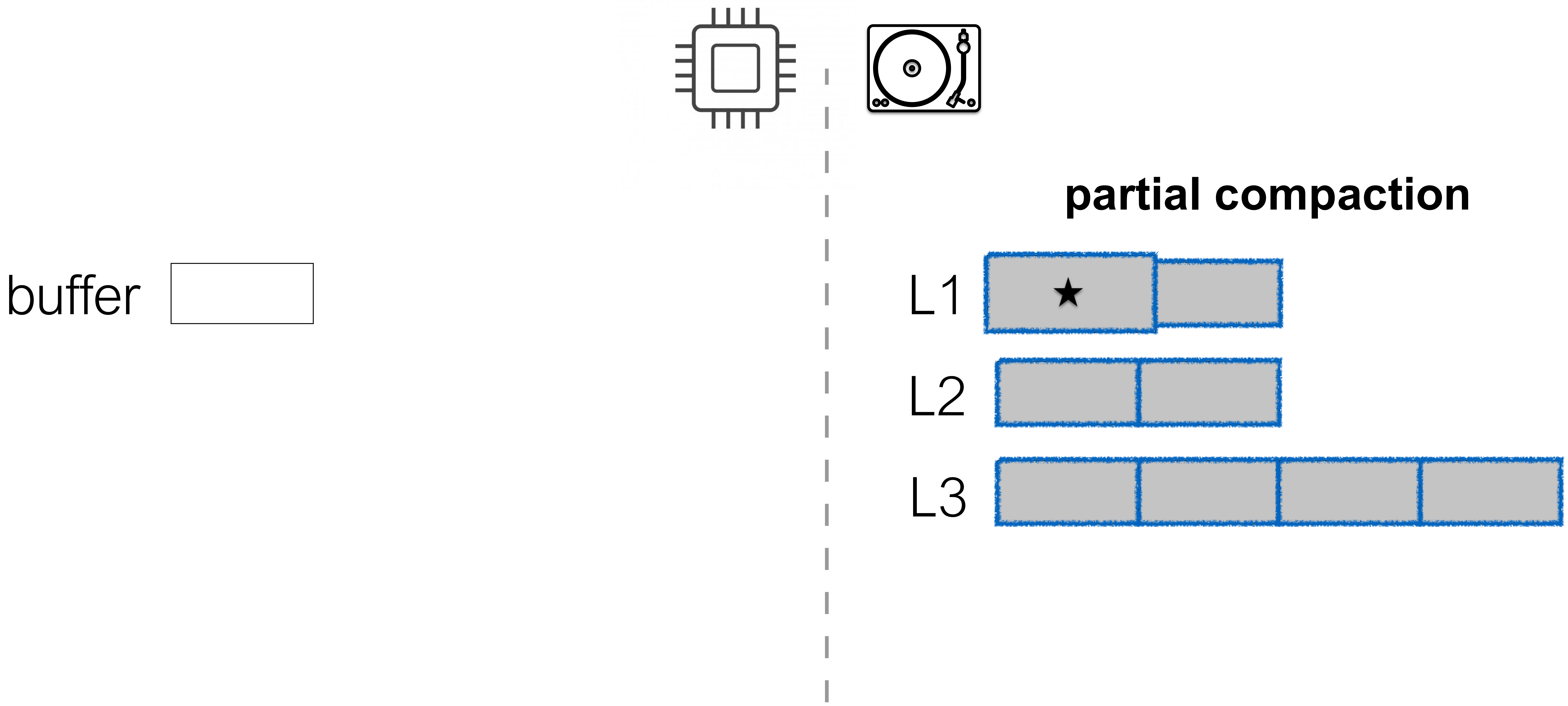
log-structured merge-tree



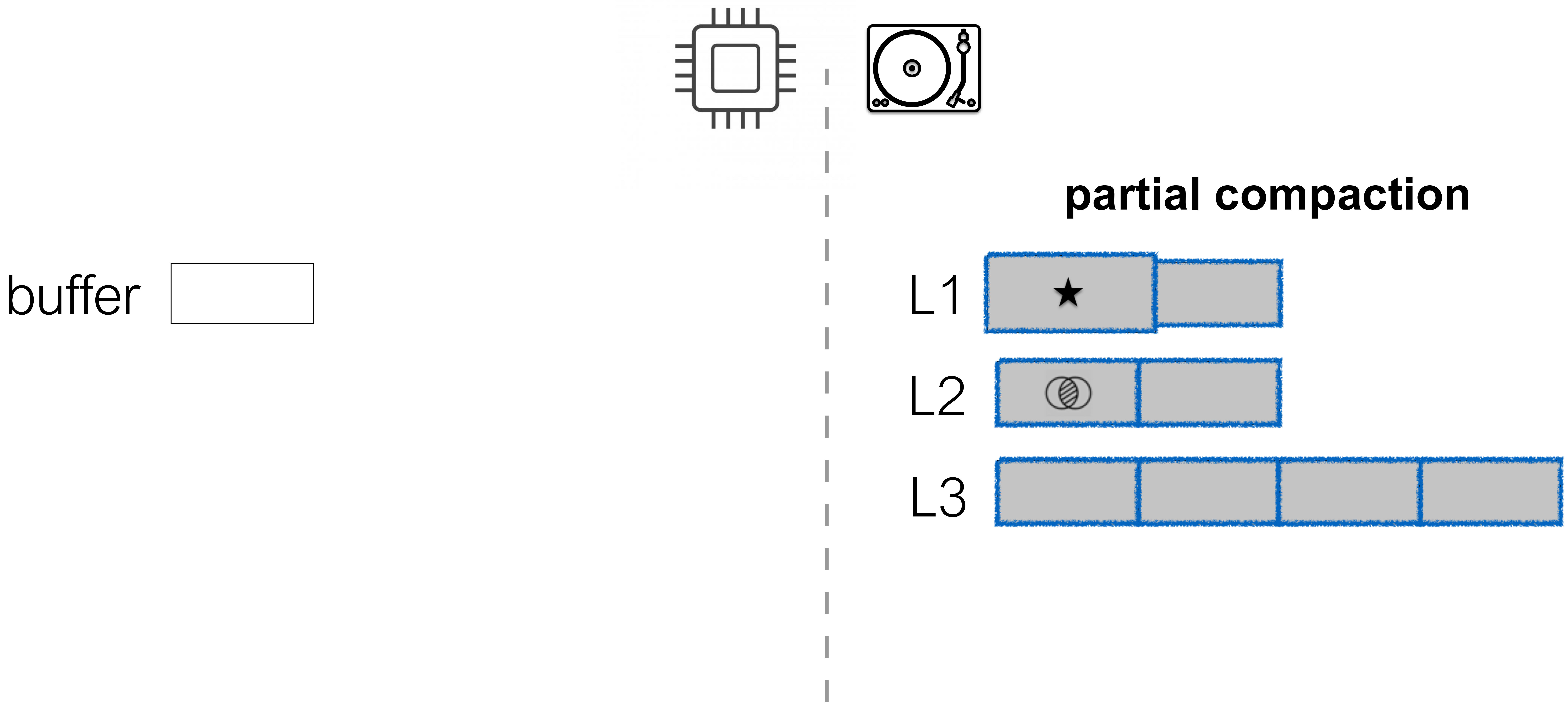
log-structured merge-tree



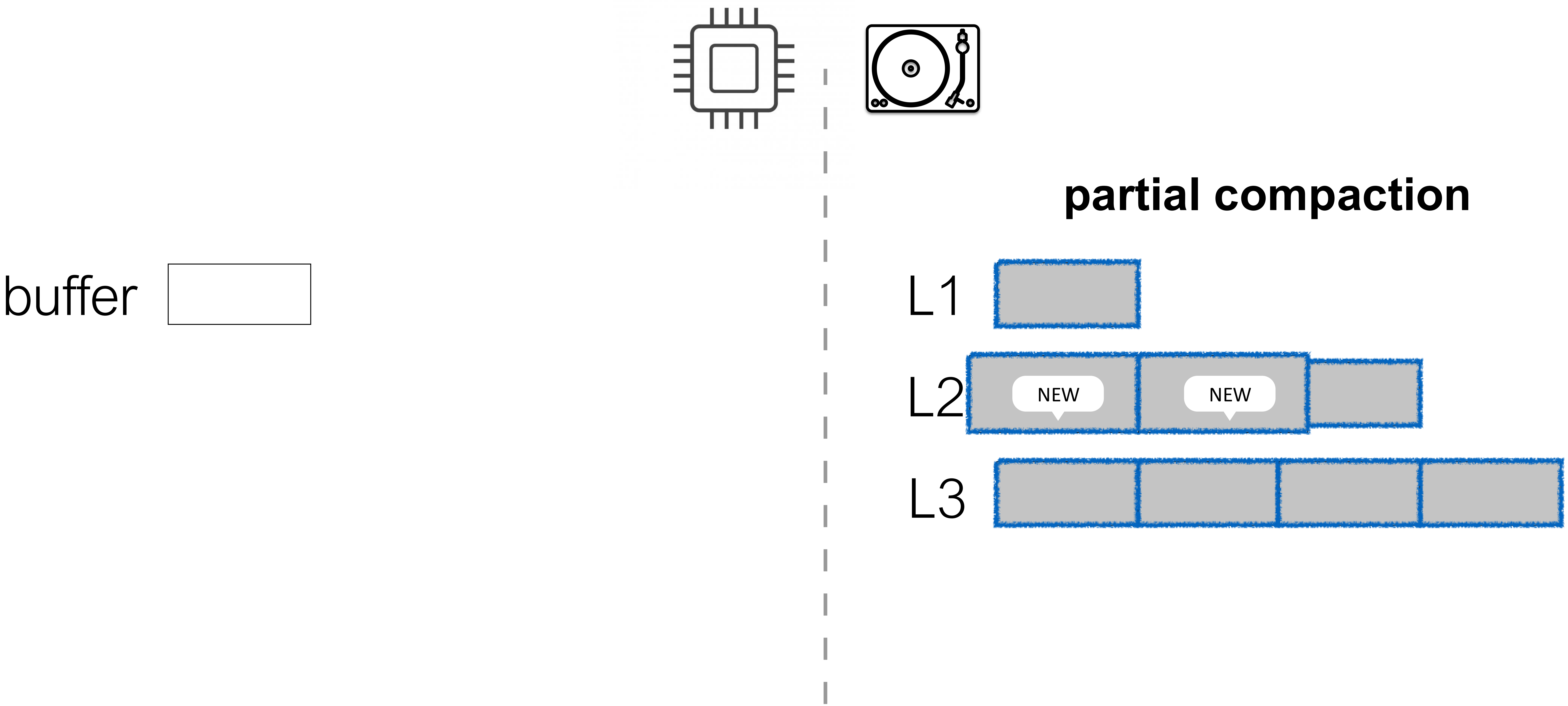
log-structured merge-tree



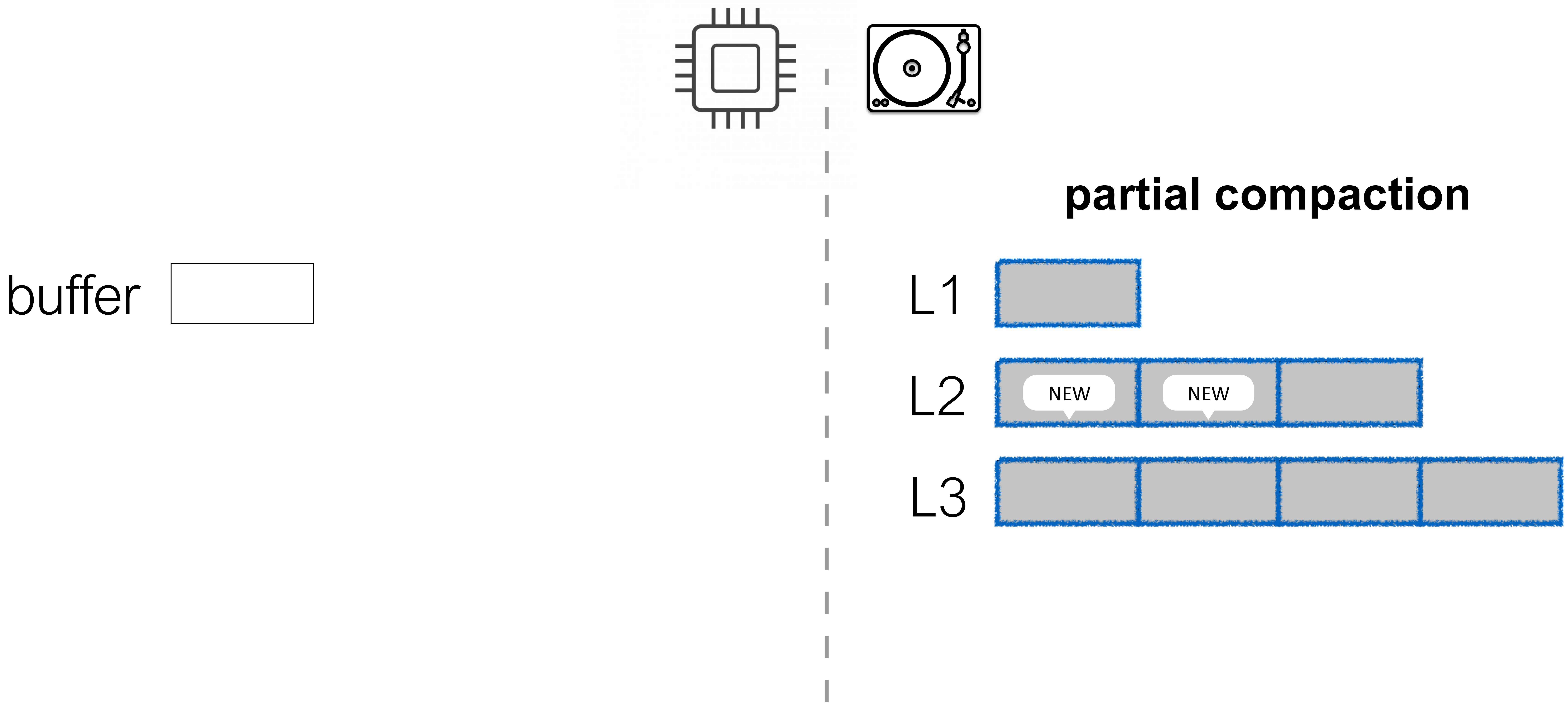
log-structured merge-tree



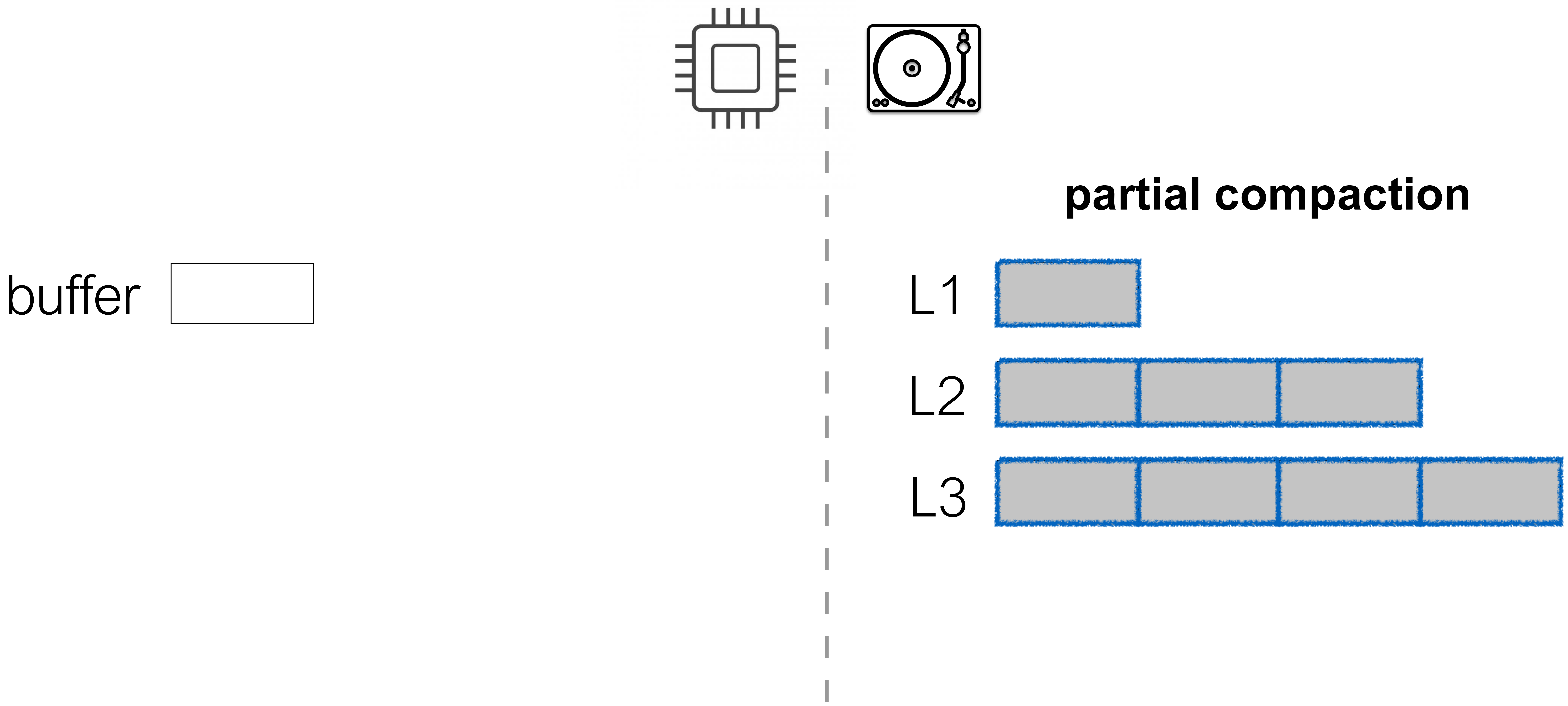
log-structured merge-tree



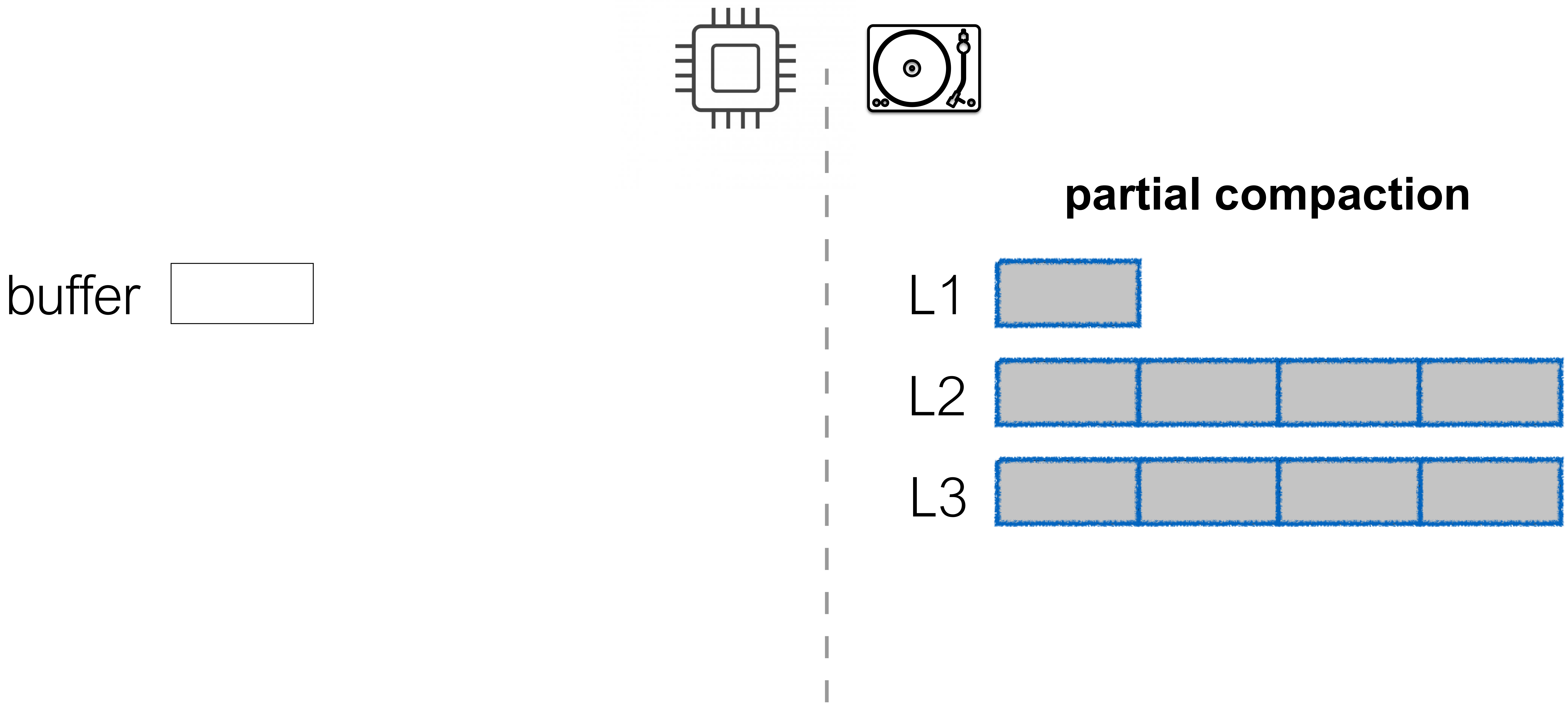
log-structured merge-tree



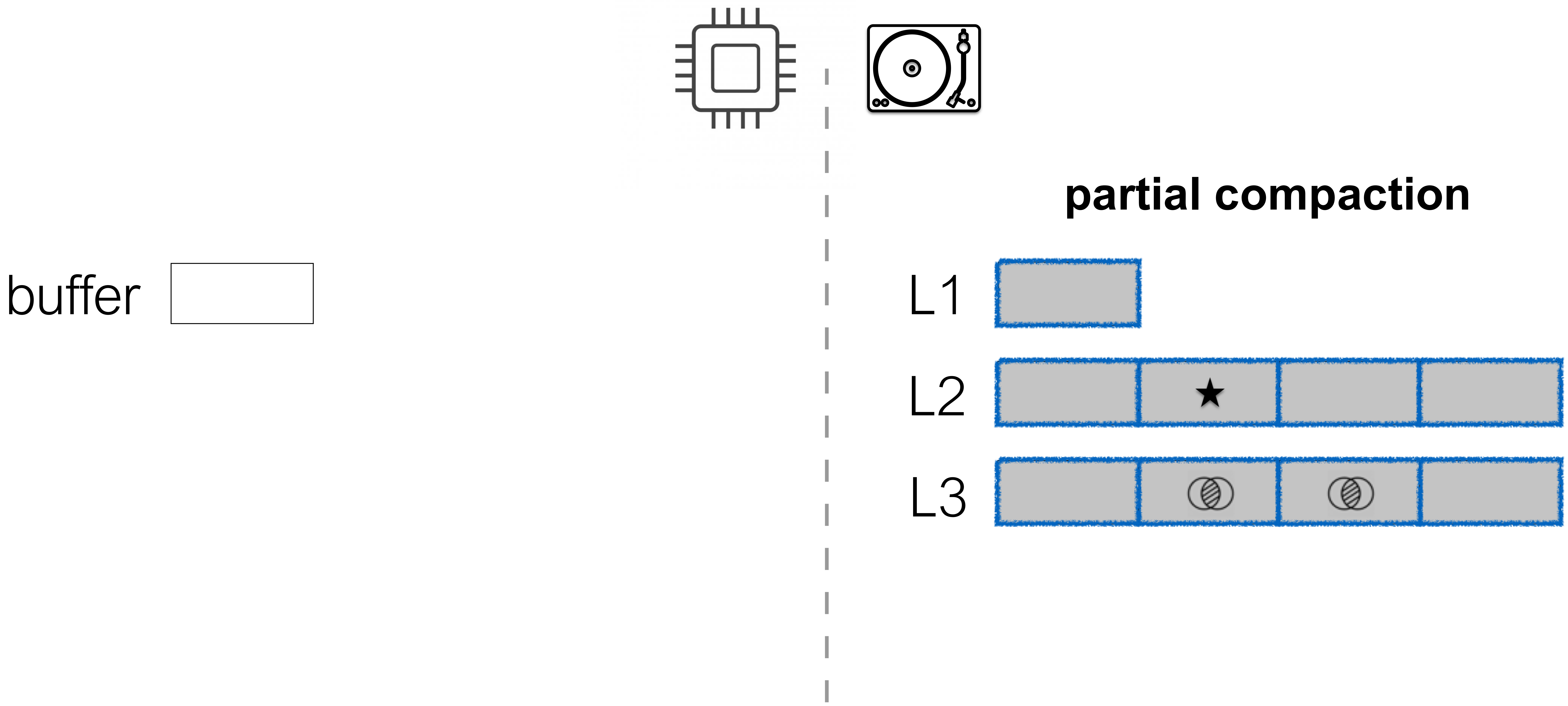
log-structured merge-tree



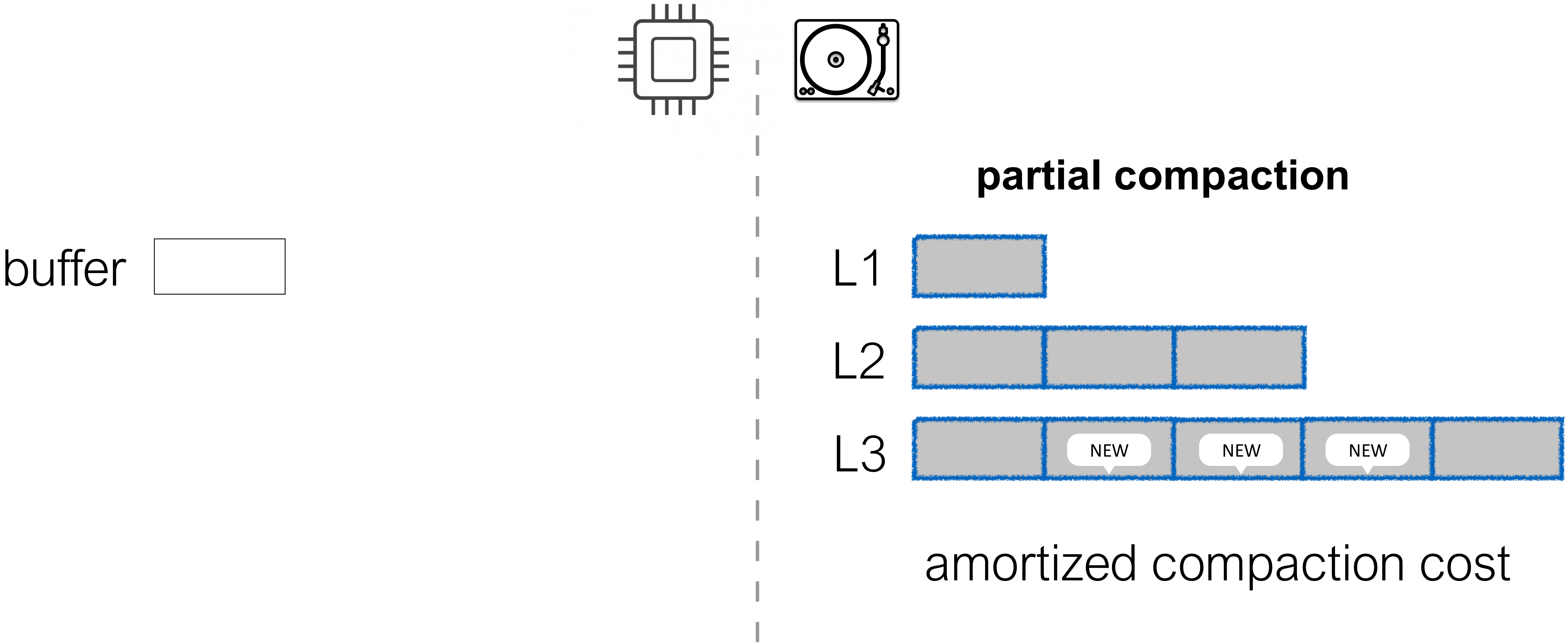
log-structured merge-tree



log-structured merge-tree

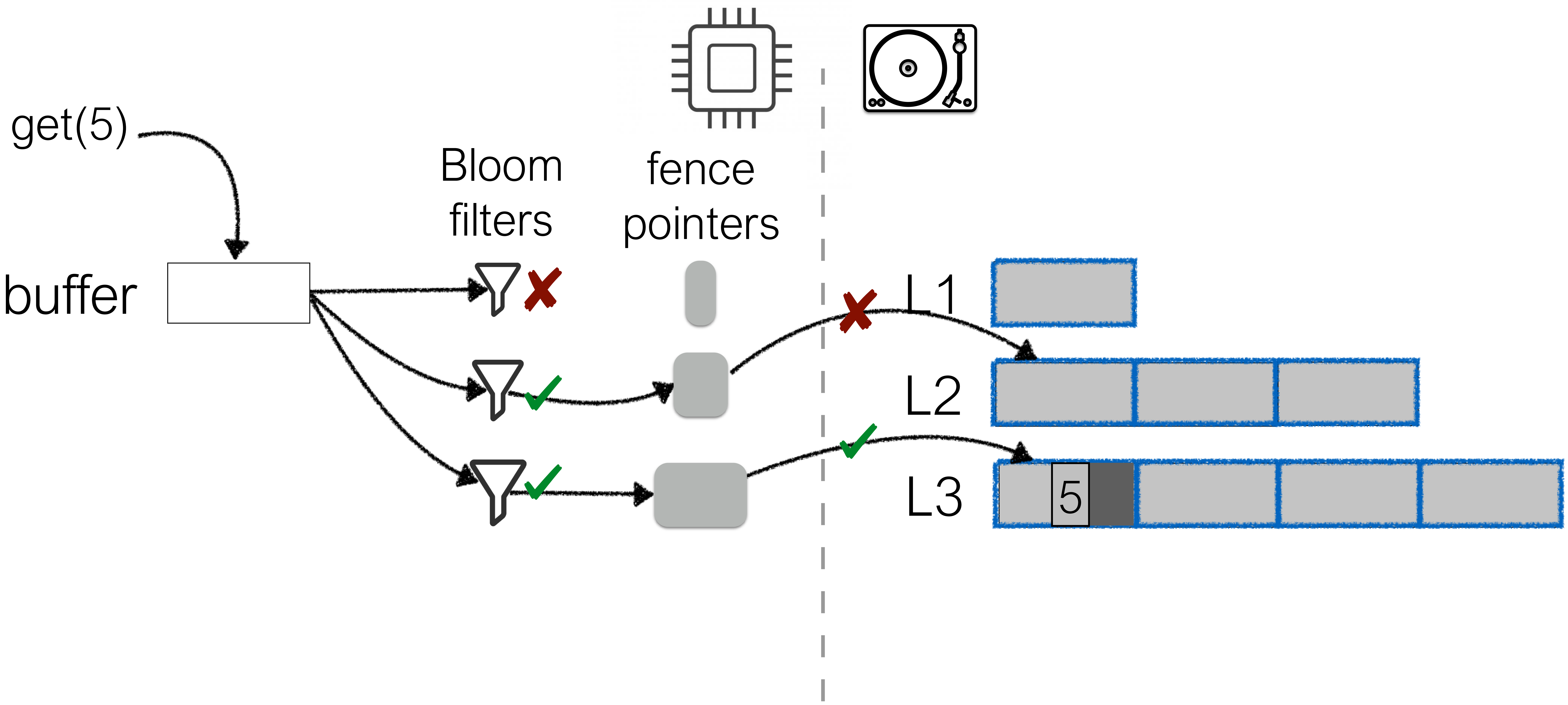


log-structured merge-tree



amortized compaction cost

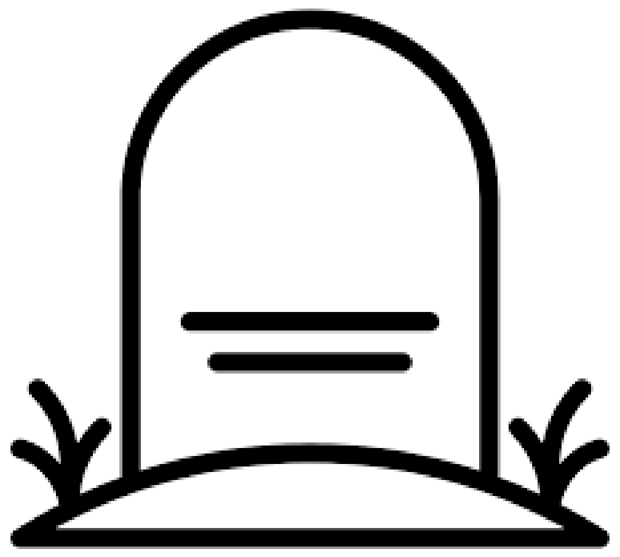
log-structured merge-tree



Now, let's talk about deletes!

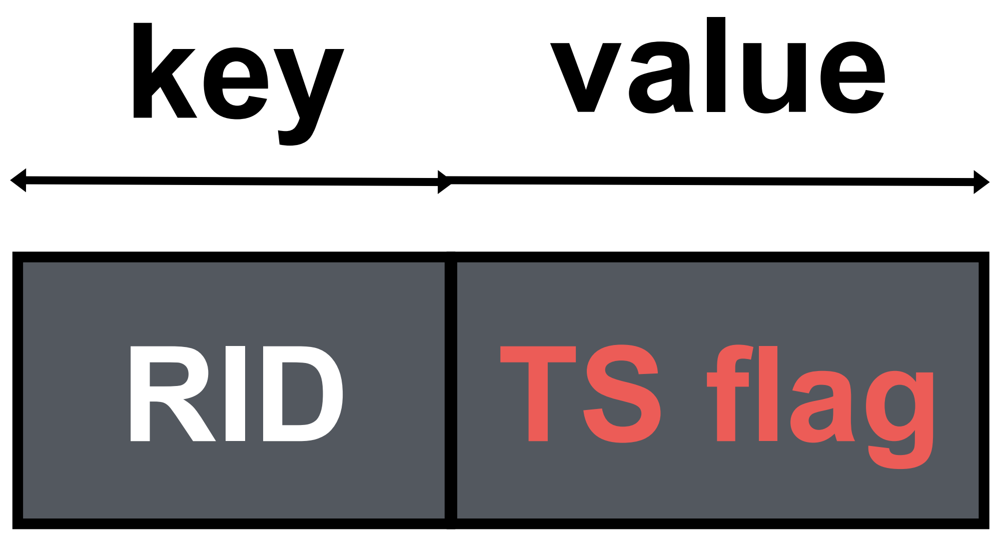
deletes in LSM-tree

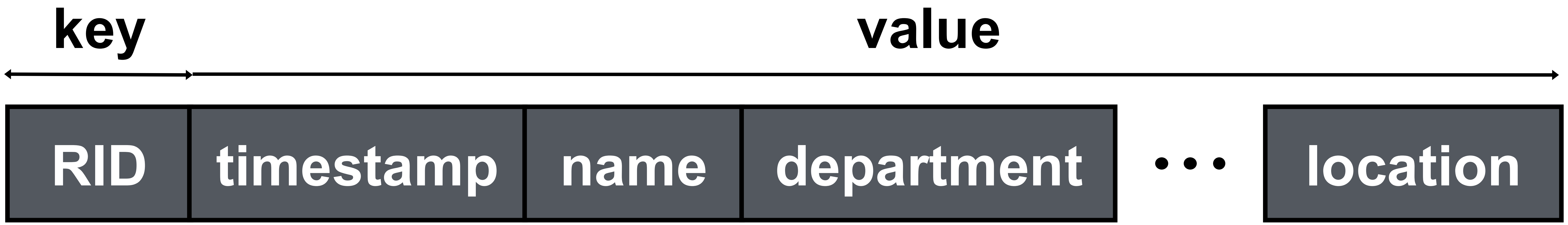
delete



deletes in LSM-tree

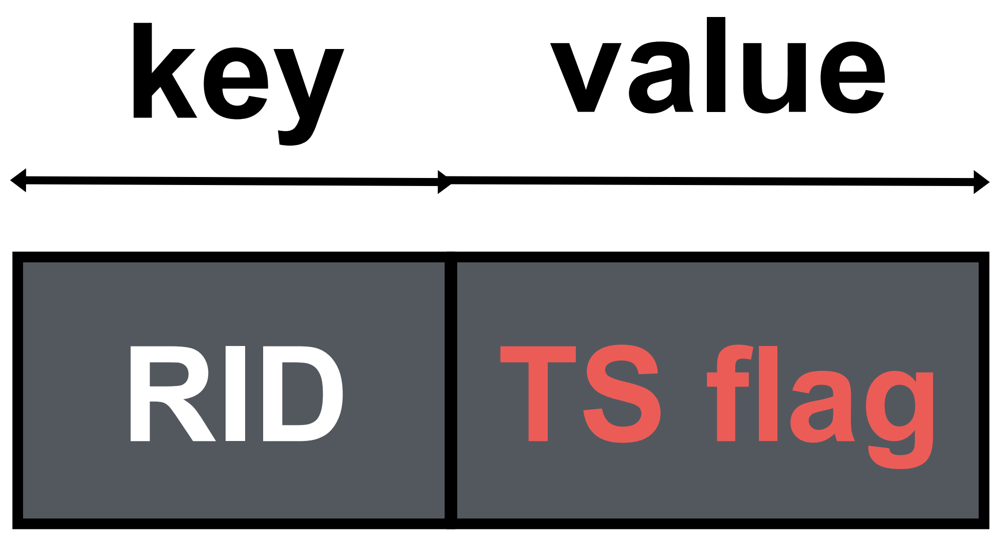
delete := insert tombstone

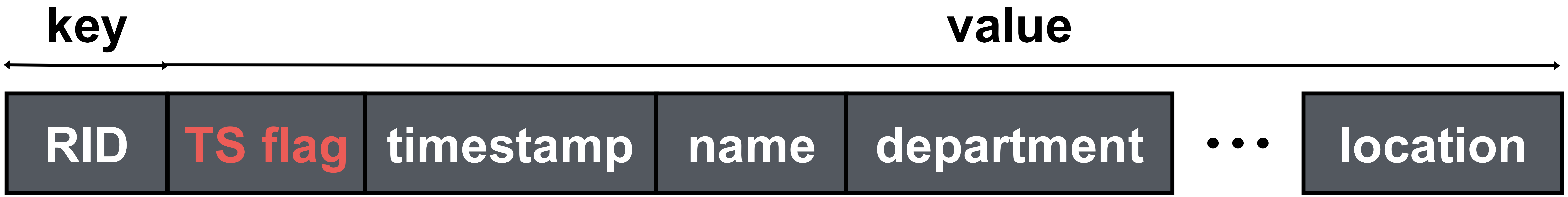




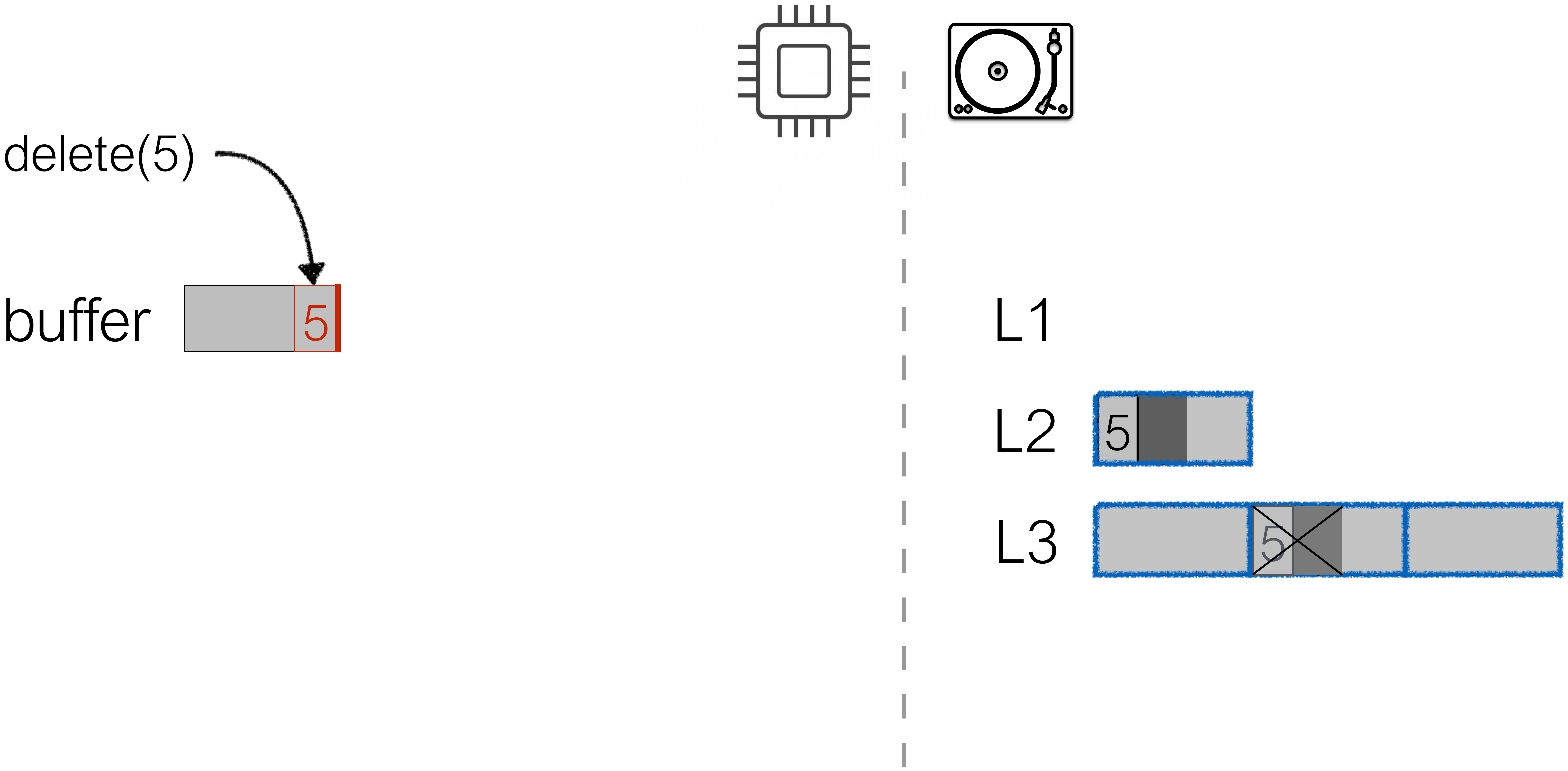
deletes in LSM-tree

delete := insert tombstone

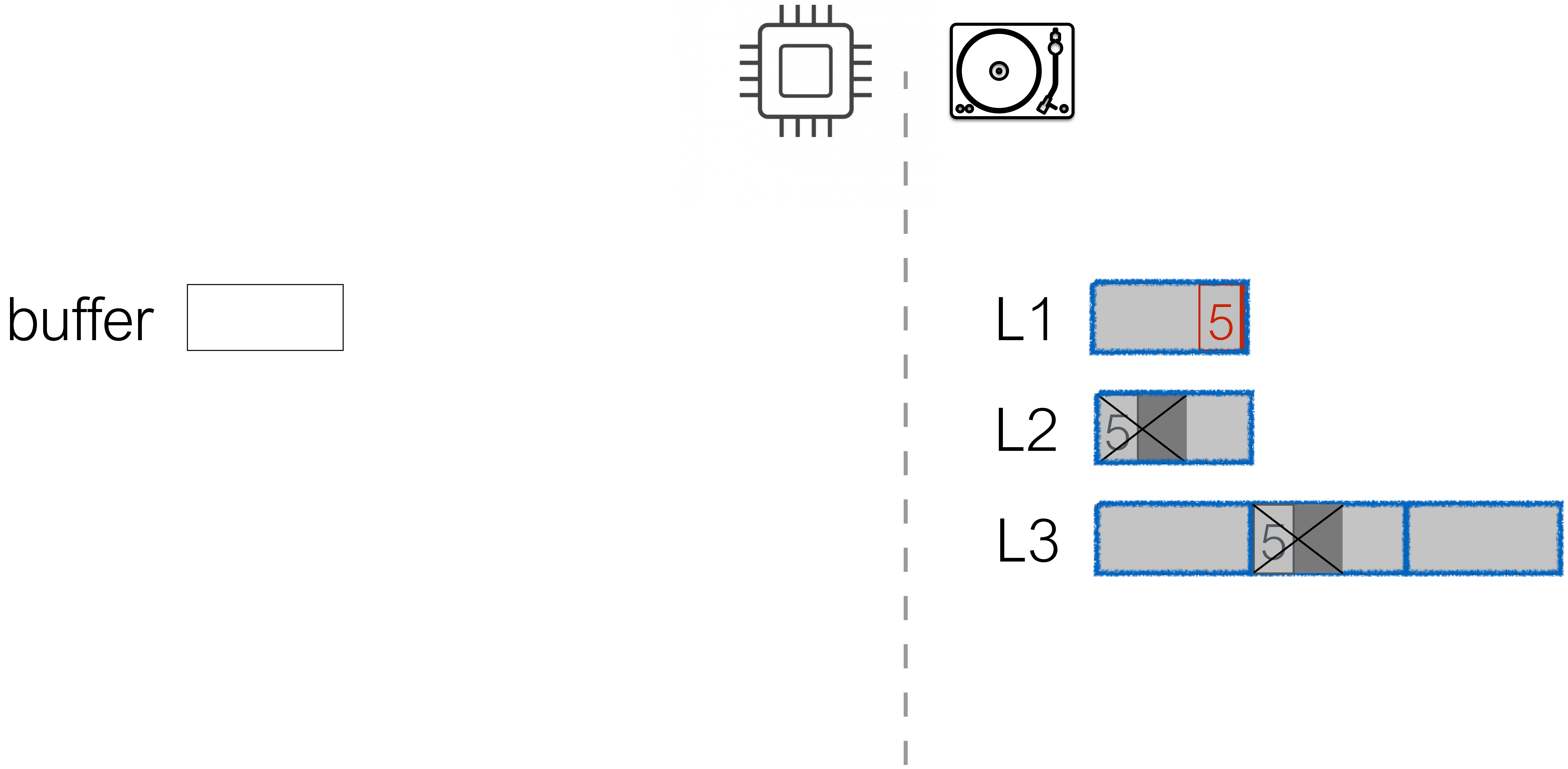




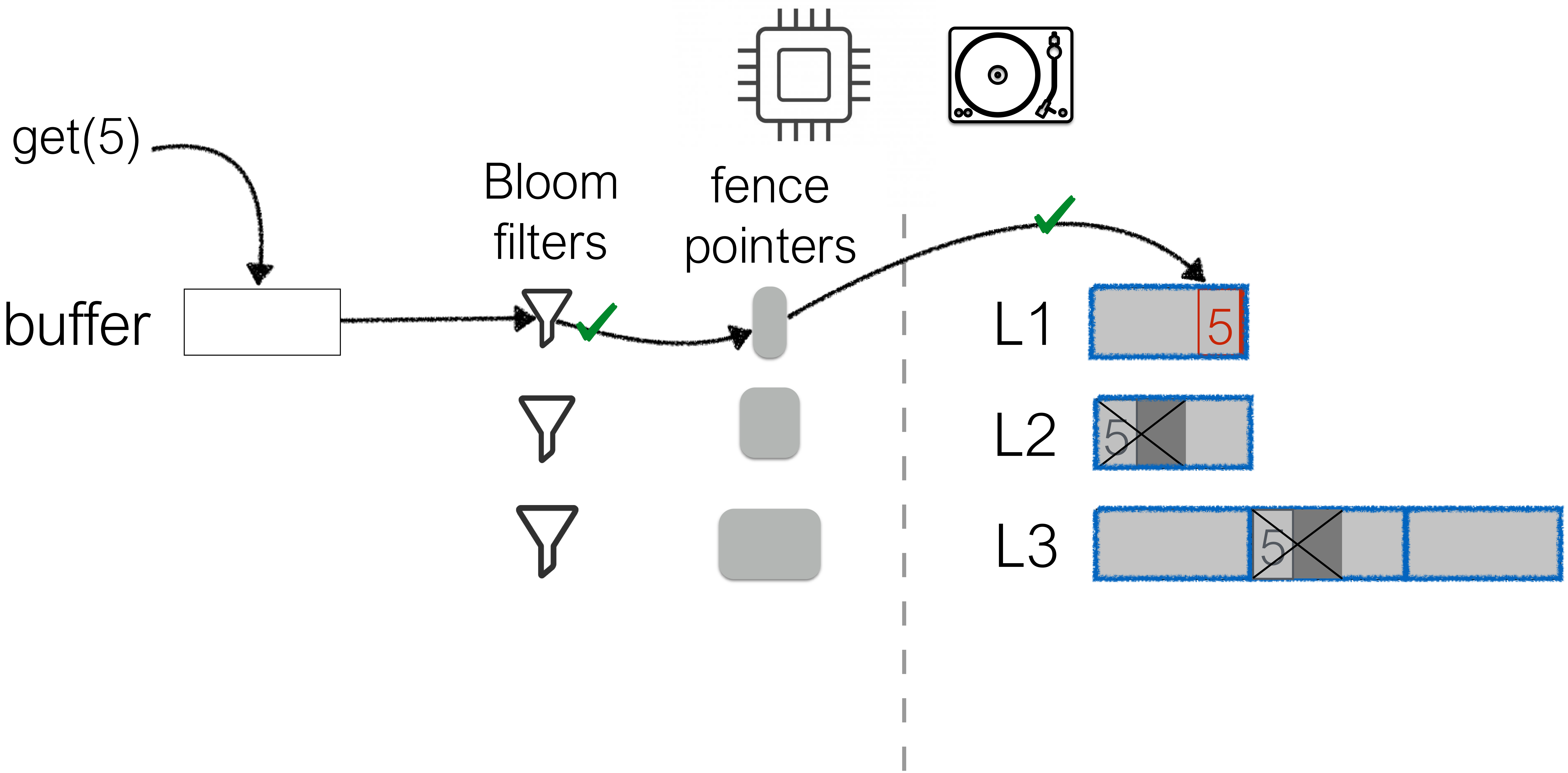
deletes in LSM-tree



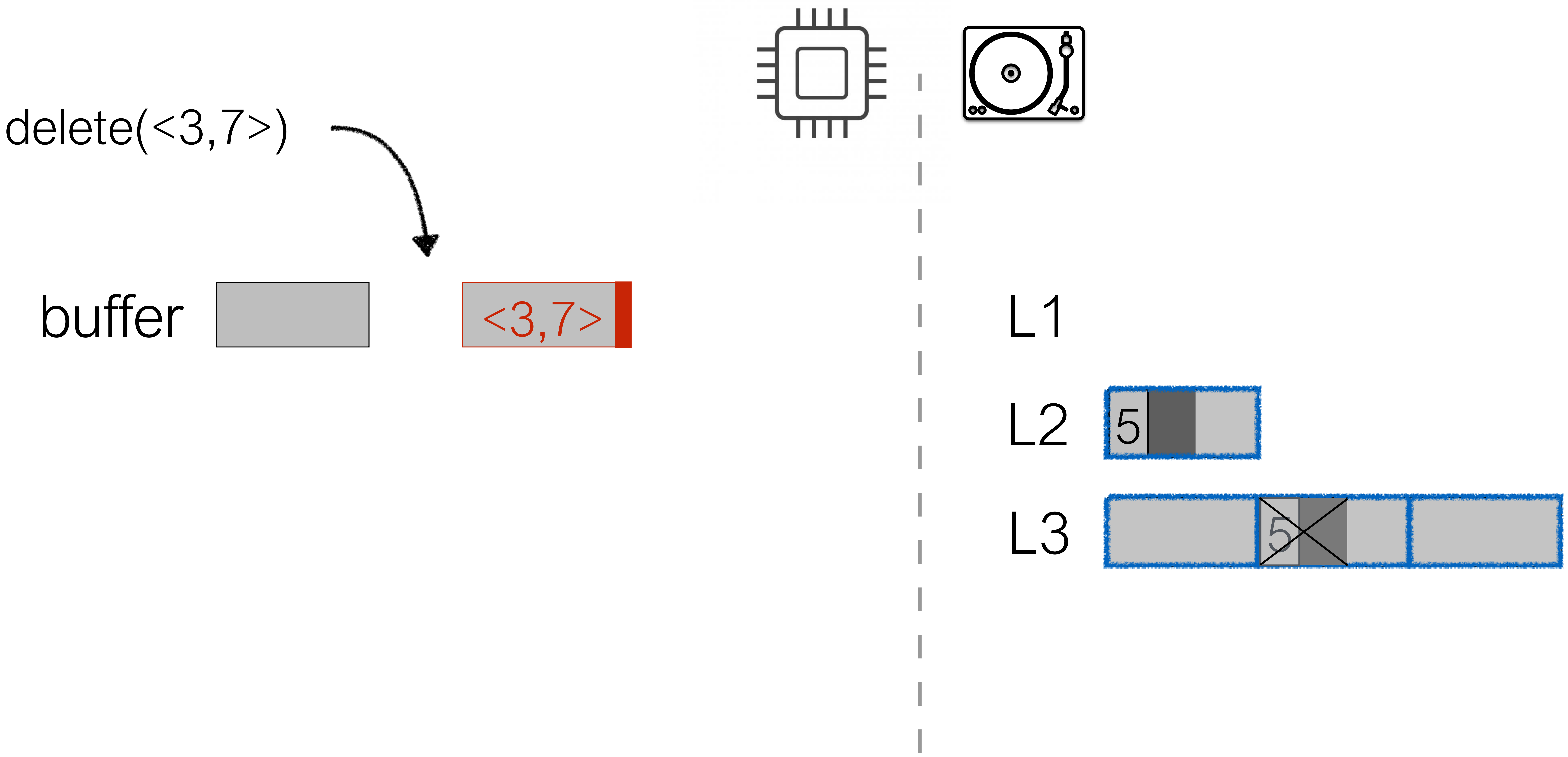
deletes in LSM-tree



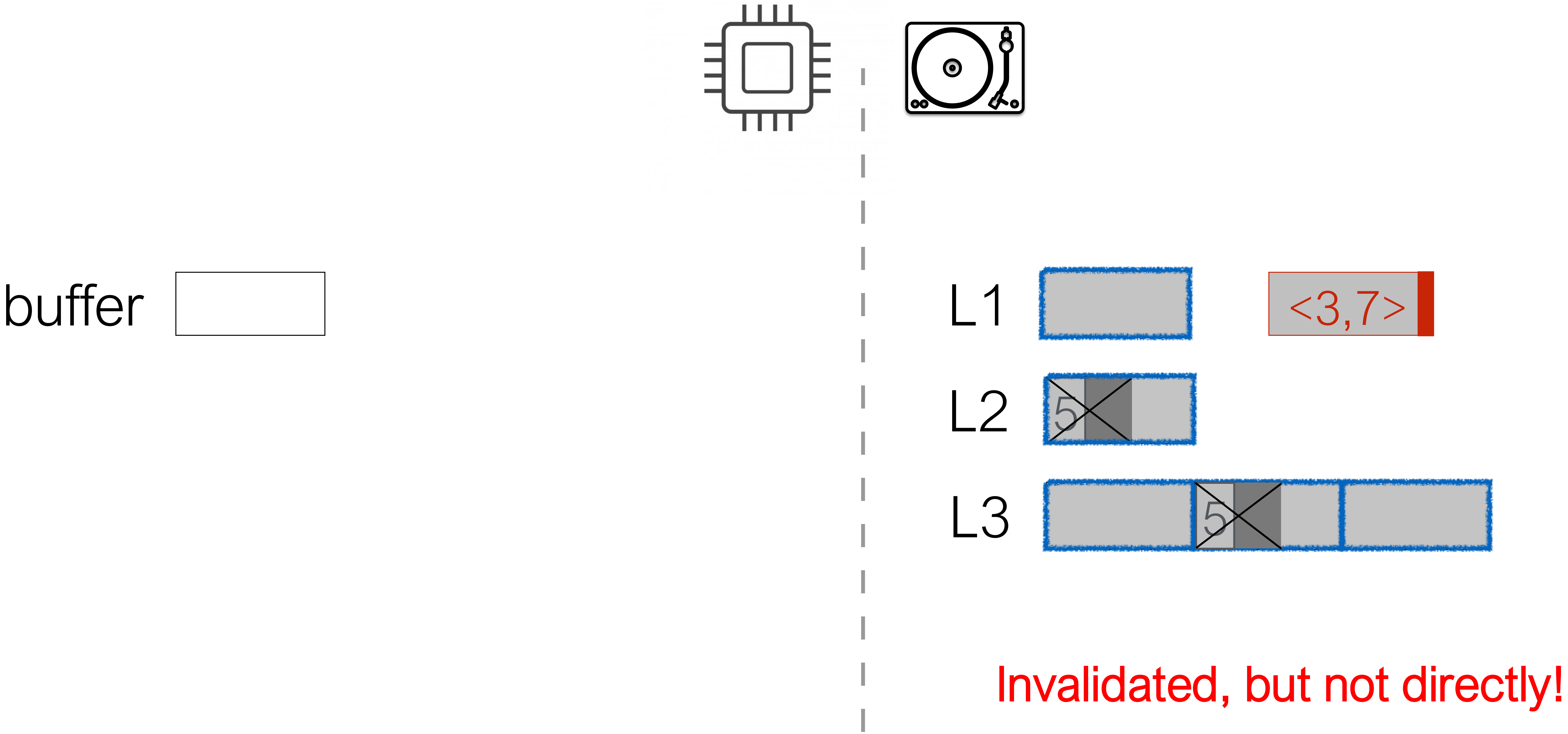
deletes in LSM-tree



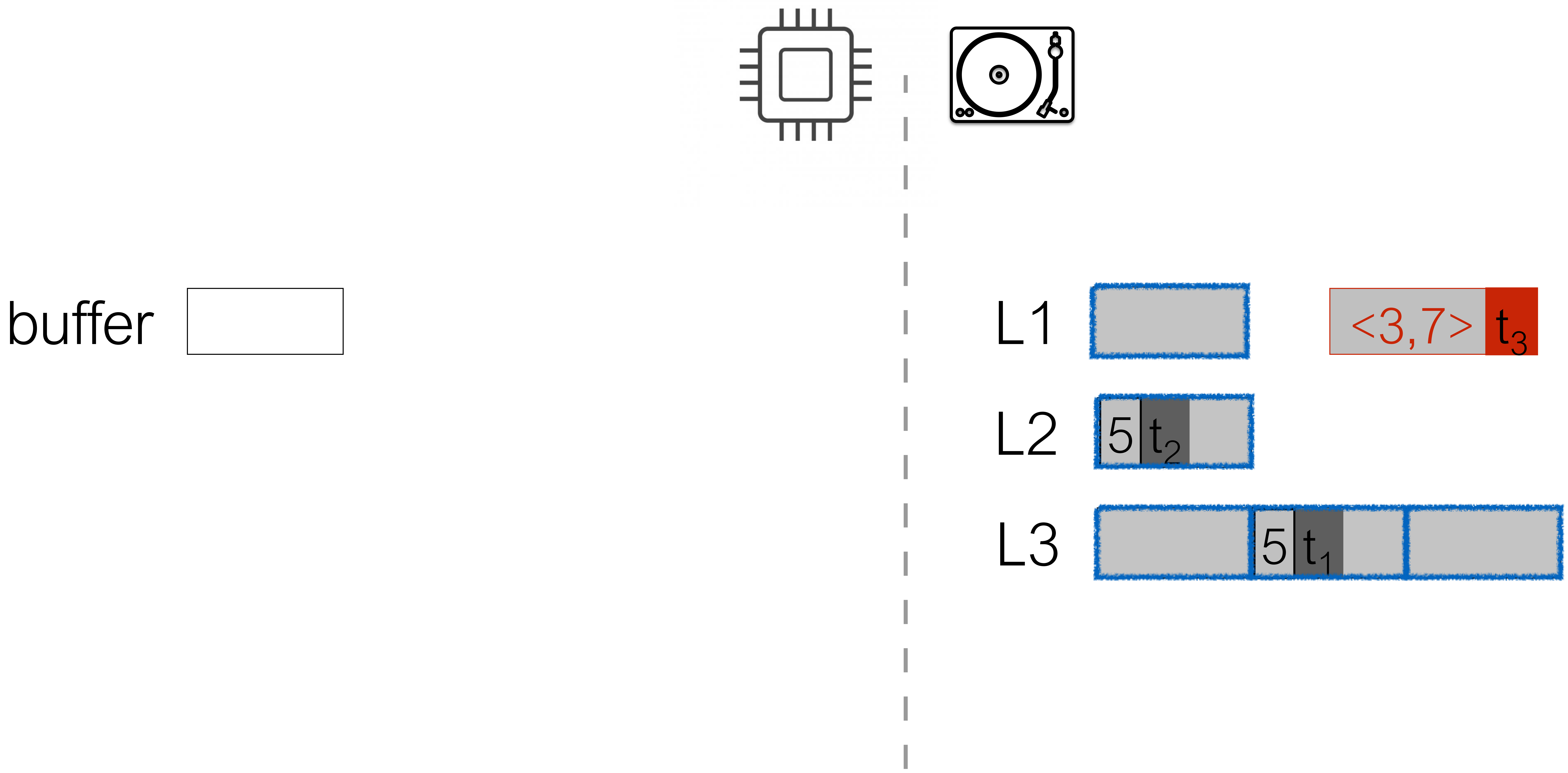
range deletes in LSM-tree



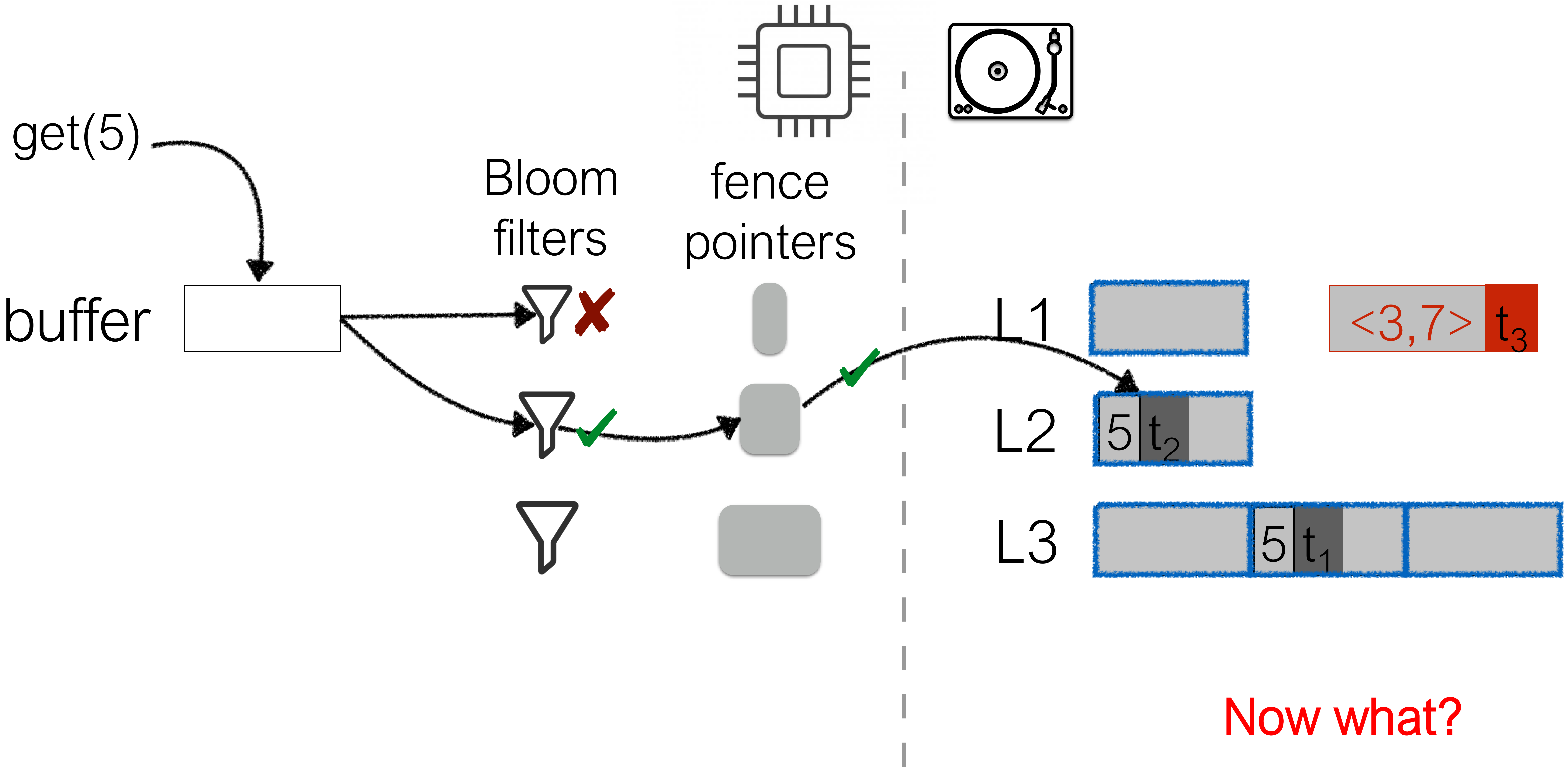
range deletes in LSM-tree



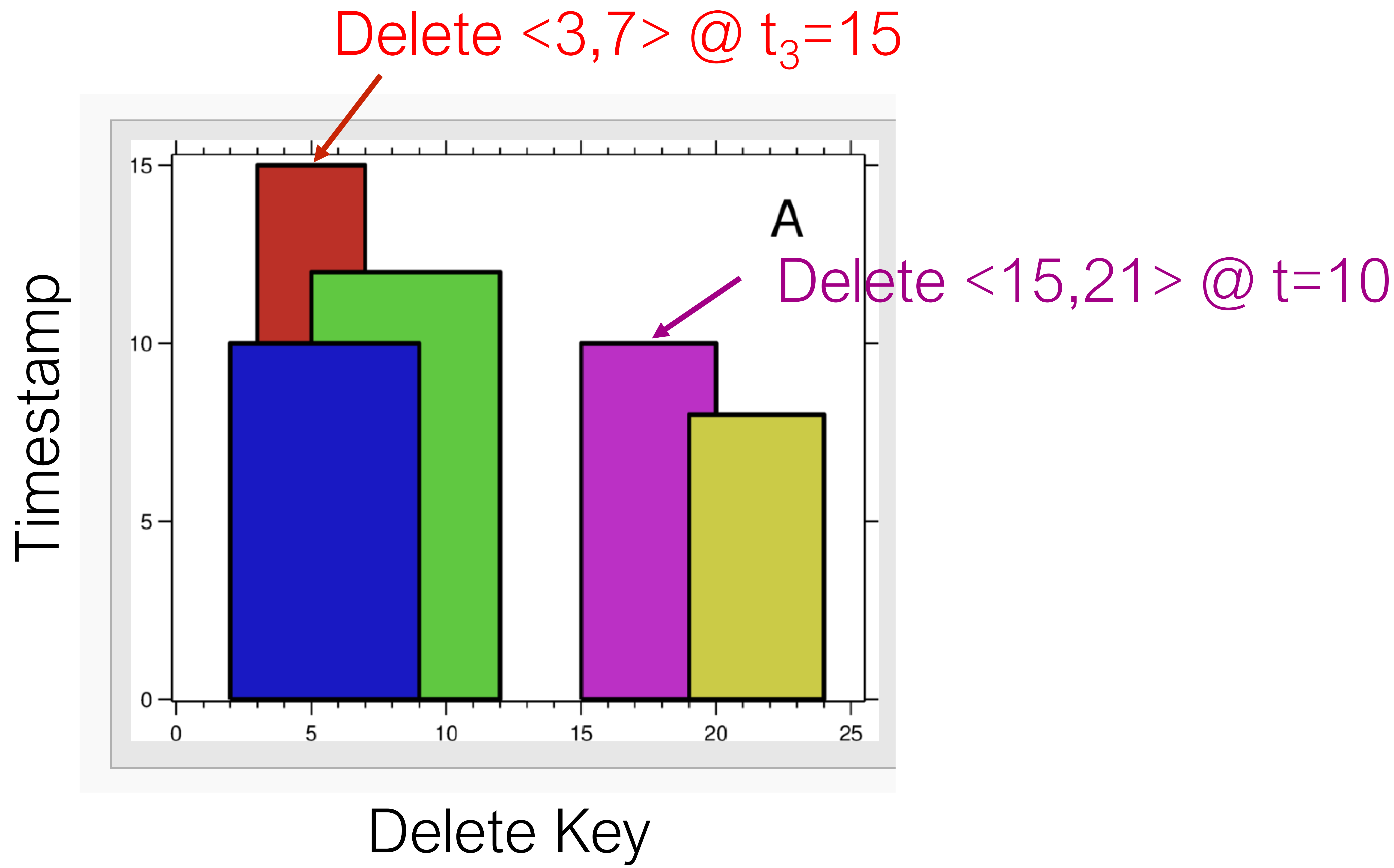
range deletes in LSM-tree



range deletes in LSM-tree

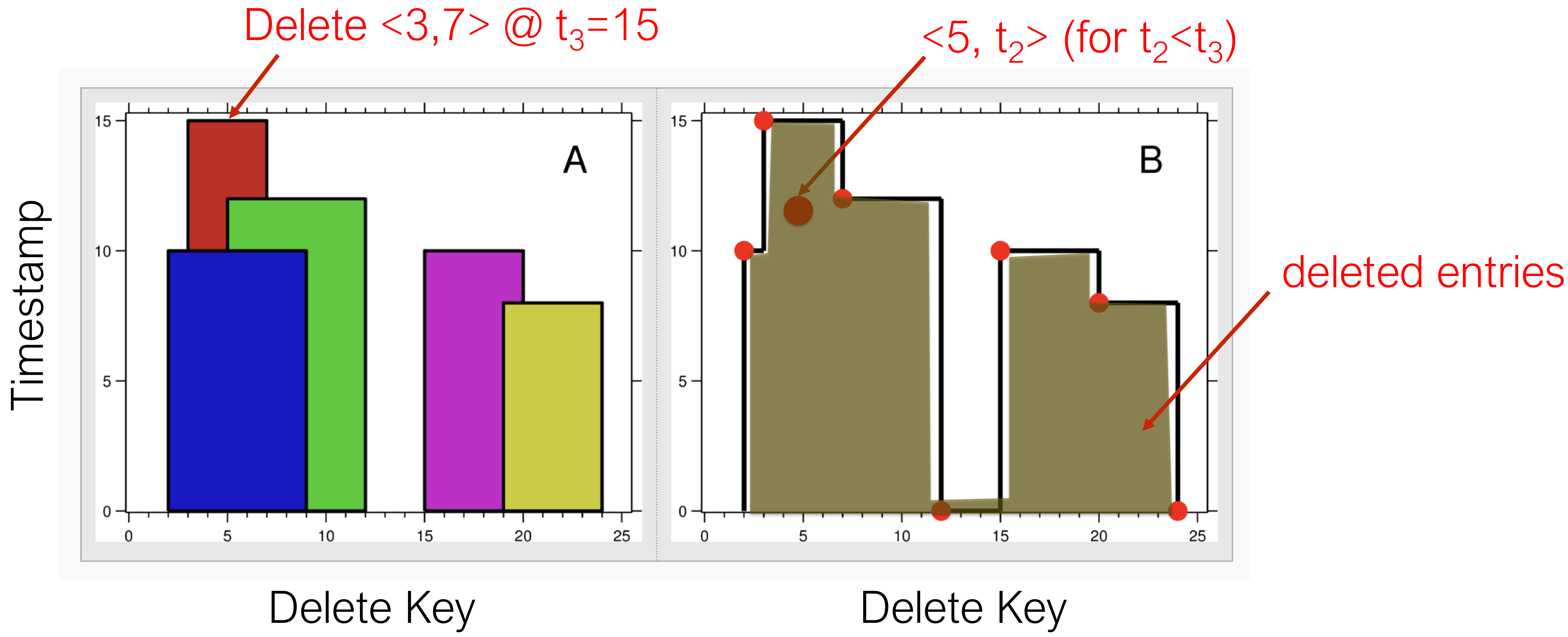


Now what?



Is $\boxed{5 \mid t_2}$ valid?

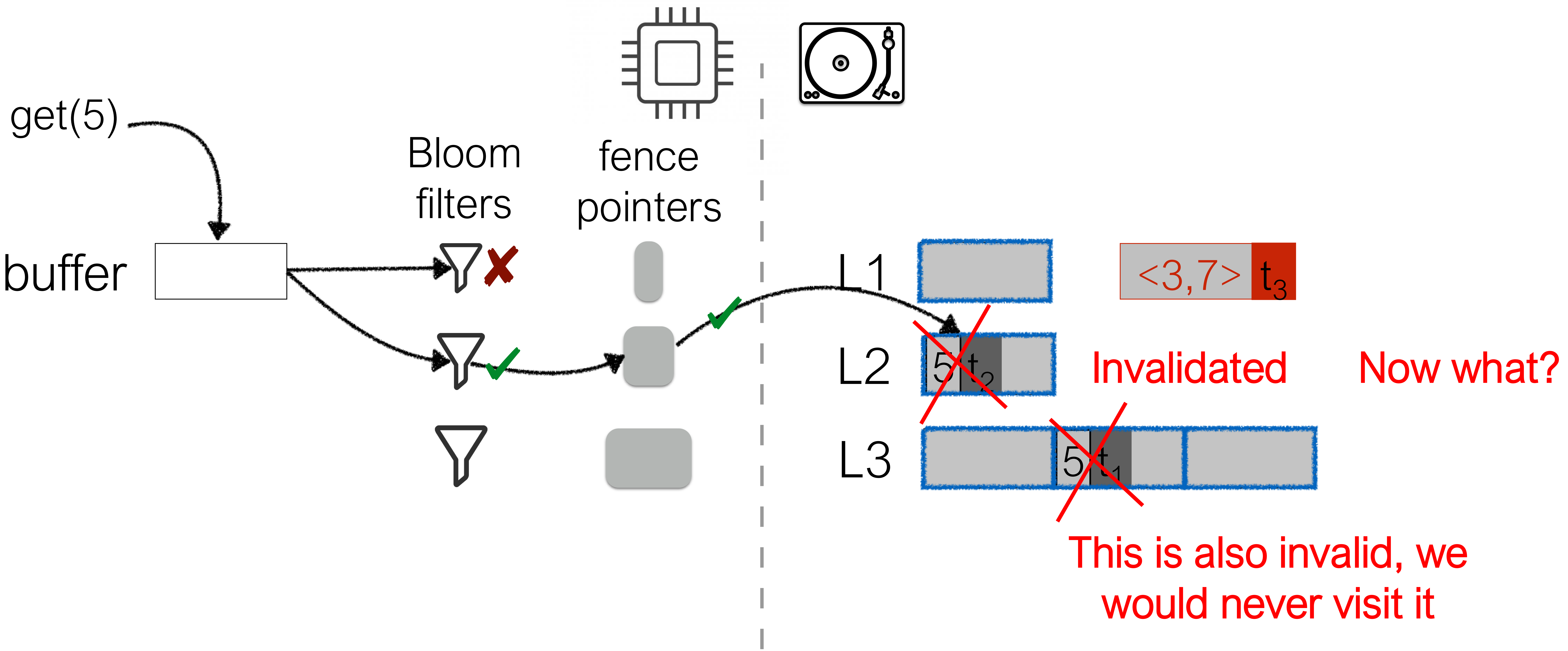
We know t_2 is before t_3 (i.e., $t_2 < t_3$, that is, $t_2 < 15$)



Is $\boxed{5 \mid t_2}$ valid?

We know t_2 is before t_3 (i.e., $t_2<t_3$, that is, $t_2<15$)

range deletes in LSM-tree

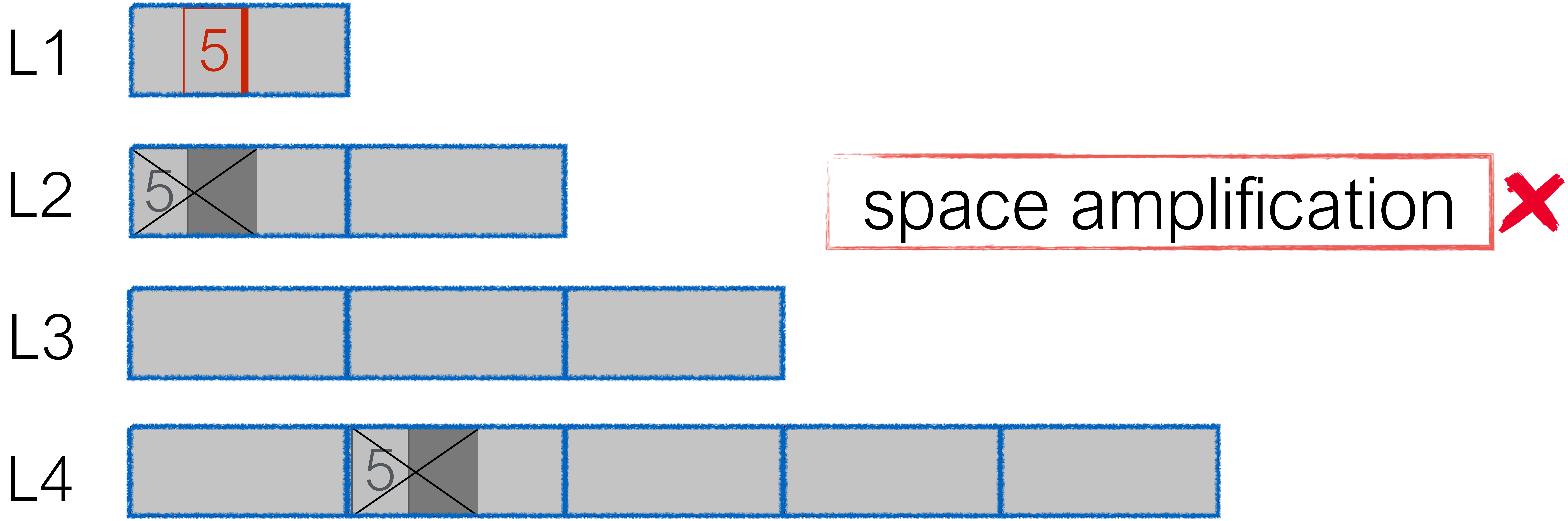


the problems

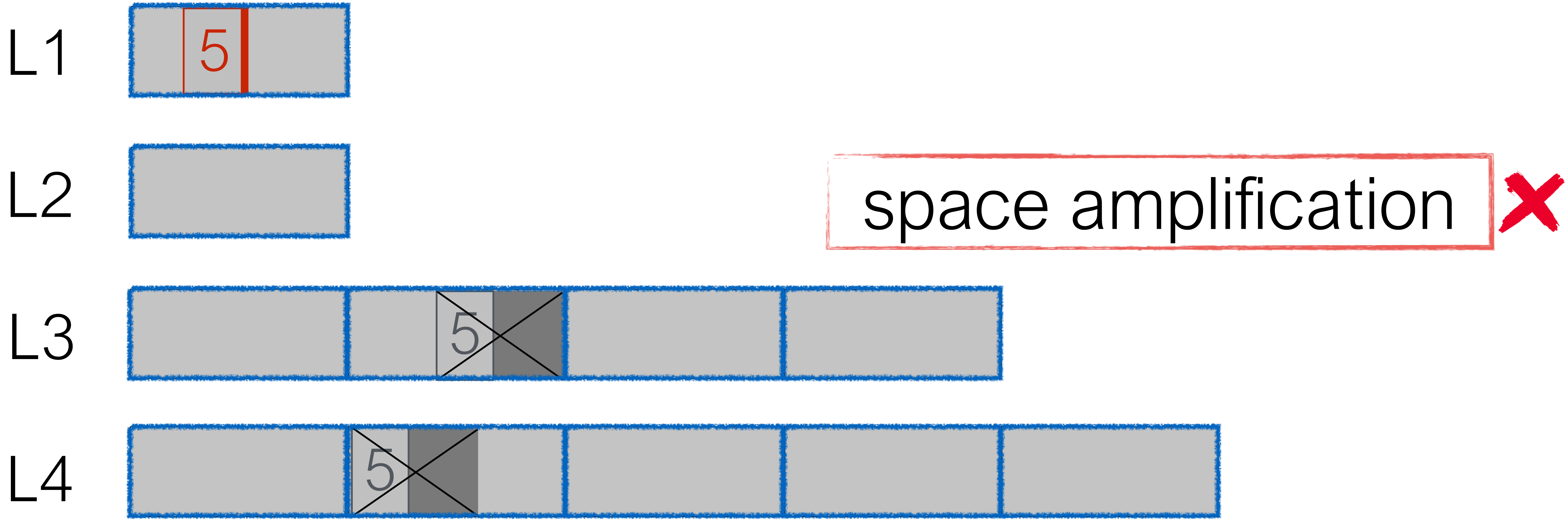
the problems

out-of-place deletes

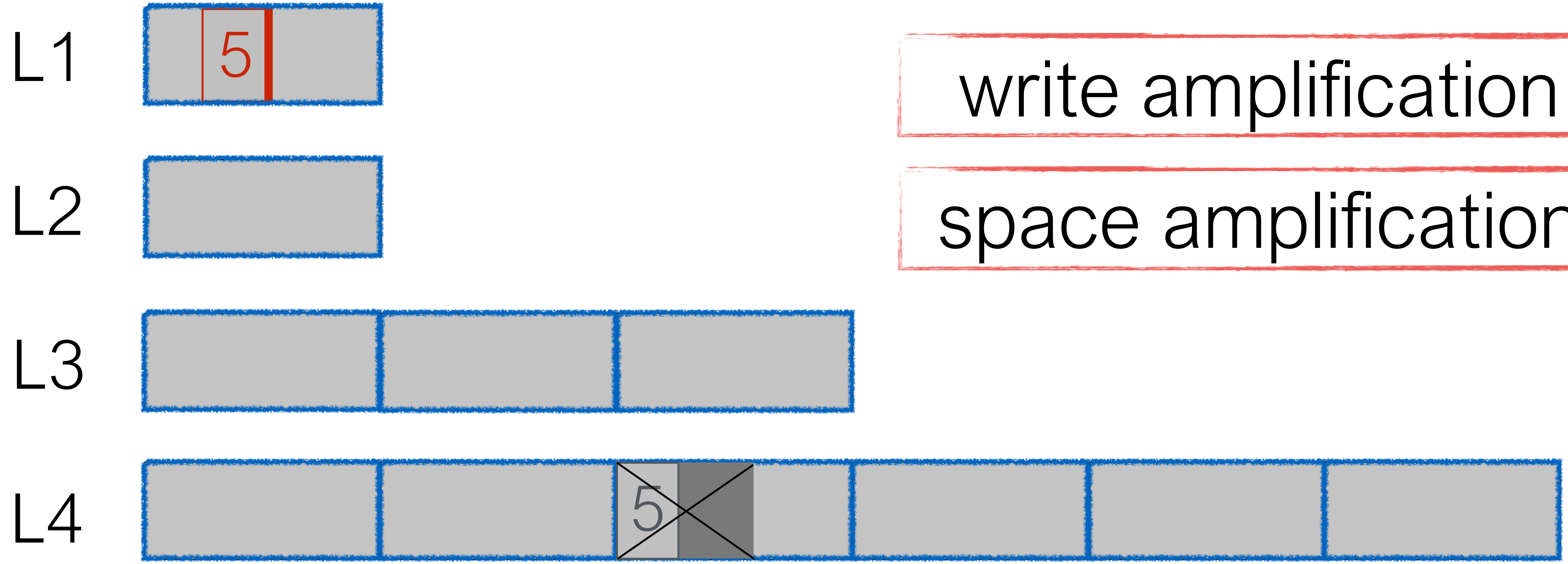
out-of-place deletes



out-of-place deletes



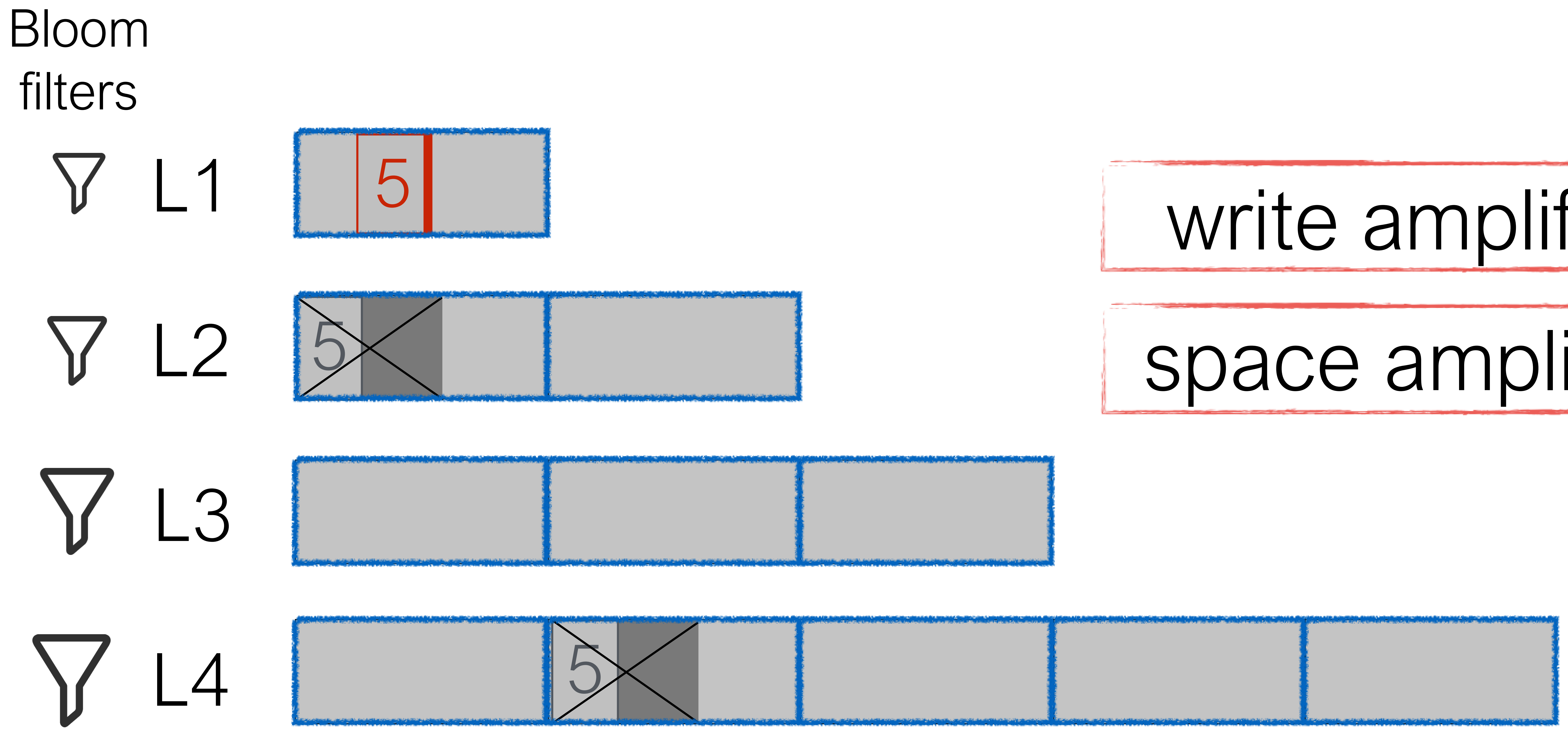
out-of-place deletes



write amplification ✗

space amplification ✗

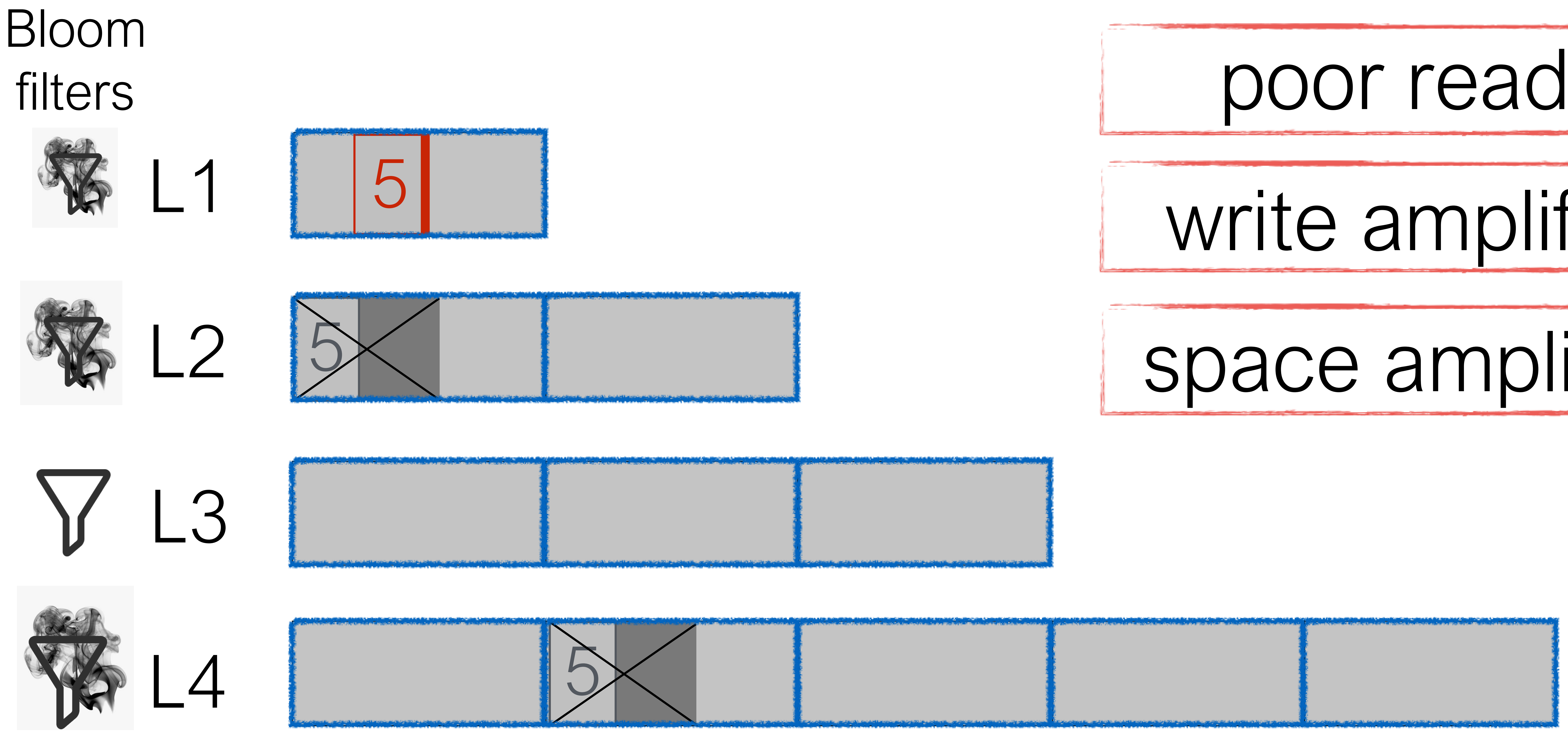
out-of-place deletes



write amplification

space amplification

out-of-place deletes

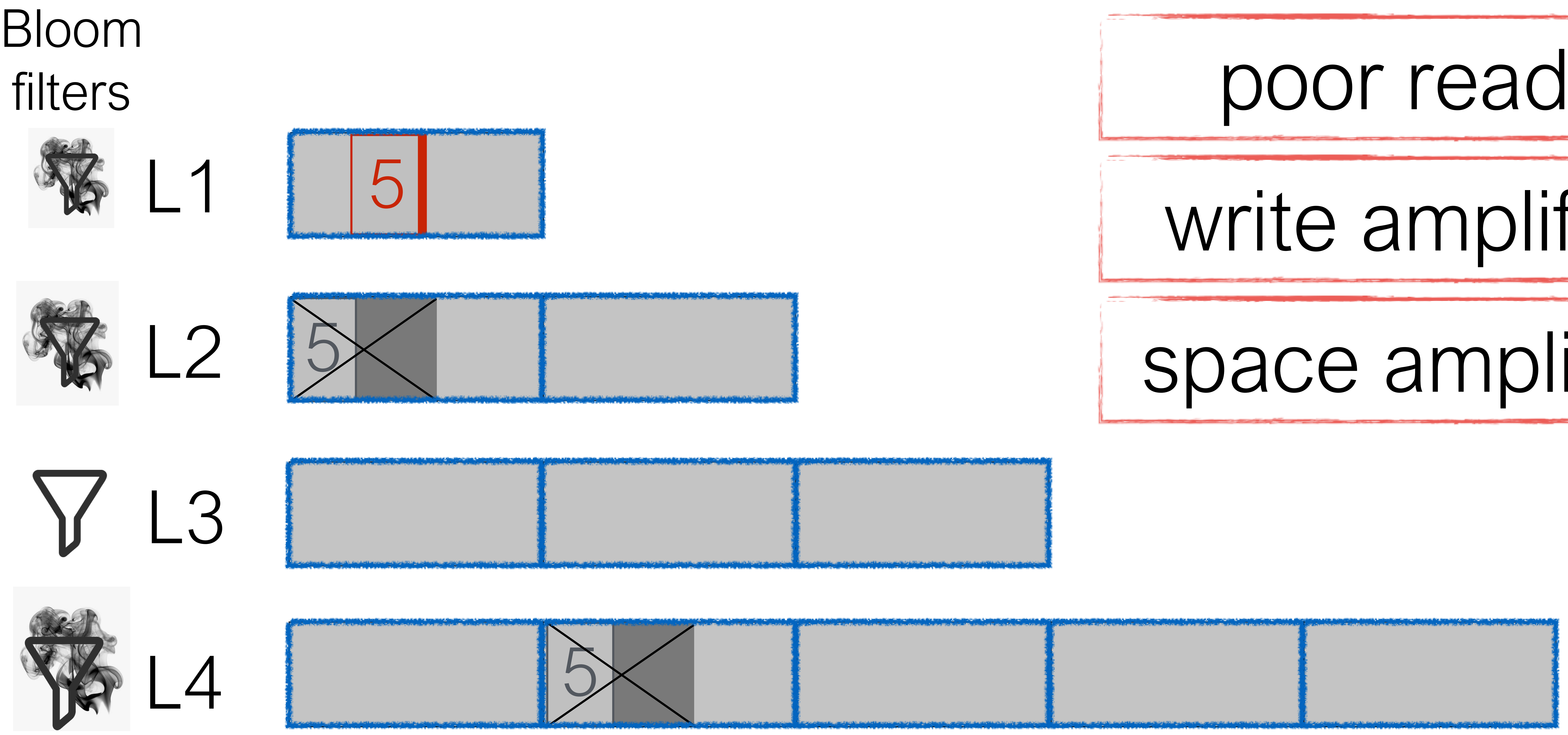


poor read perf. ✗

write amplification ✗

space amplification ✗

out-of-place deletes



poor read perf. ✗

write amplification ✗

space amplification ✗

the problems

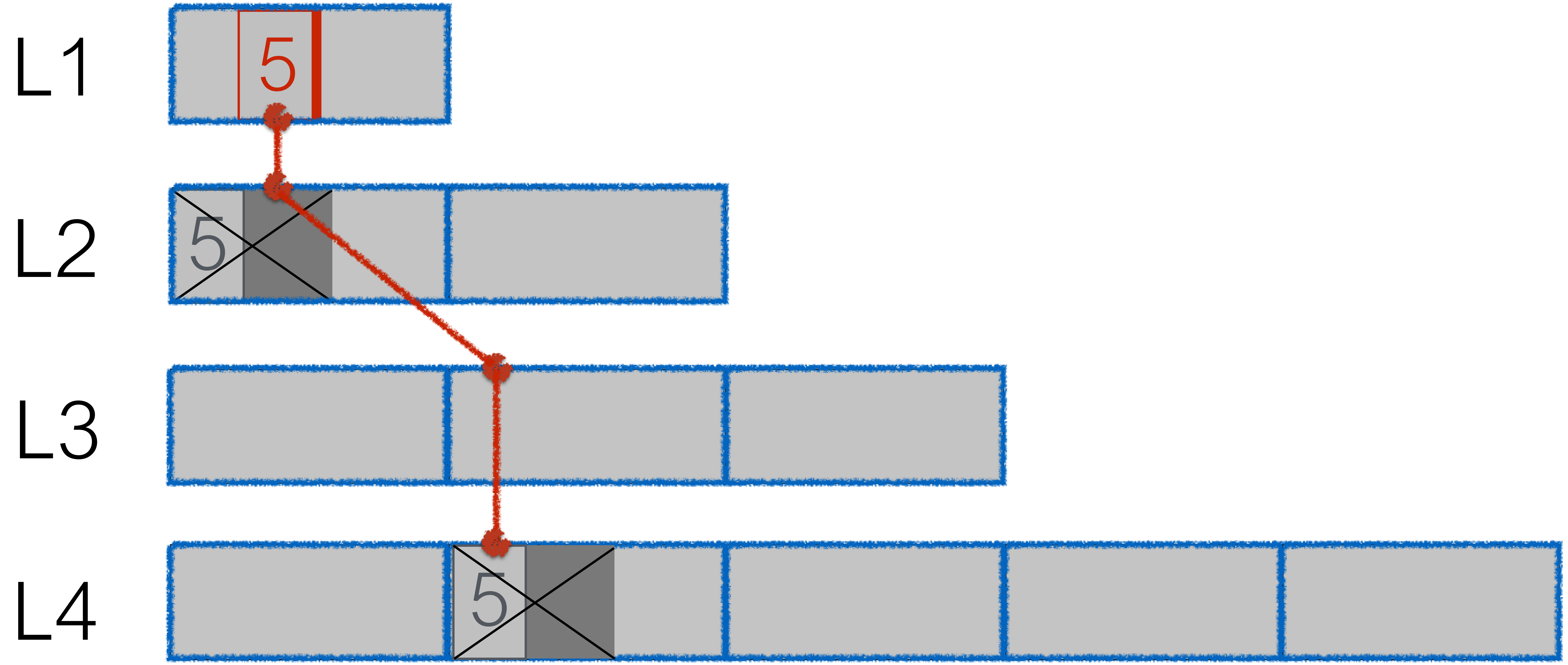
poor read perf.

write amplification

space amplification

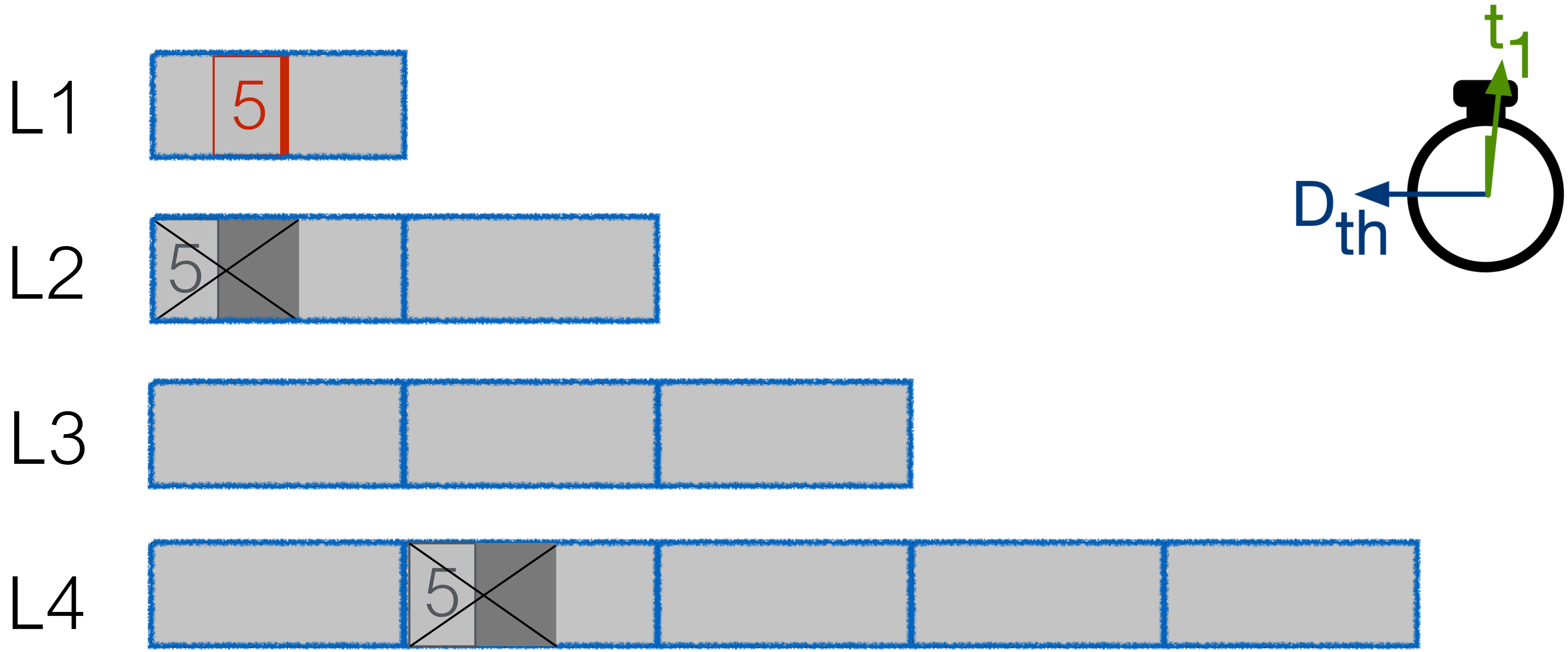
delete persistence latency

delete persistence latency



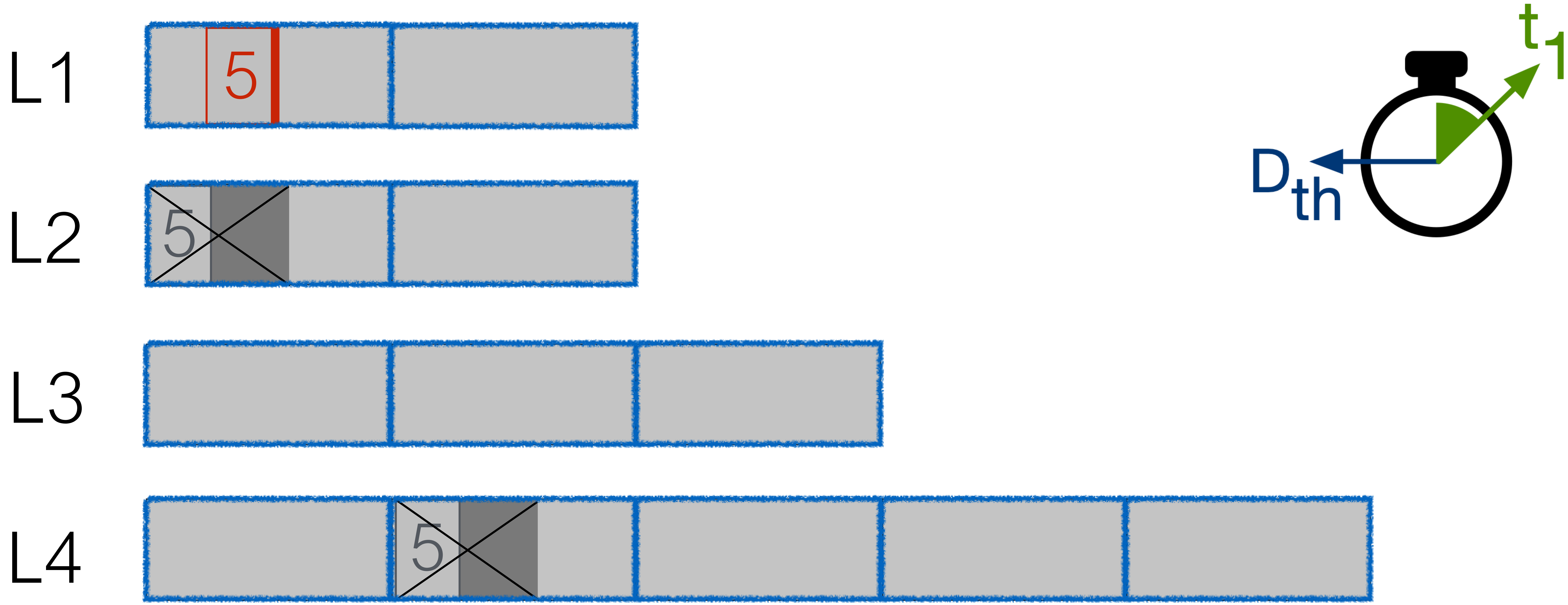
delete persistence latency

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



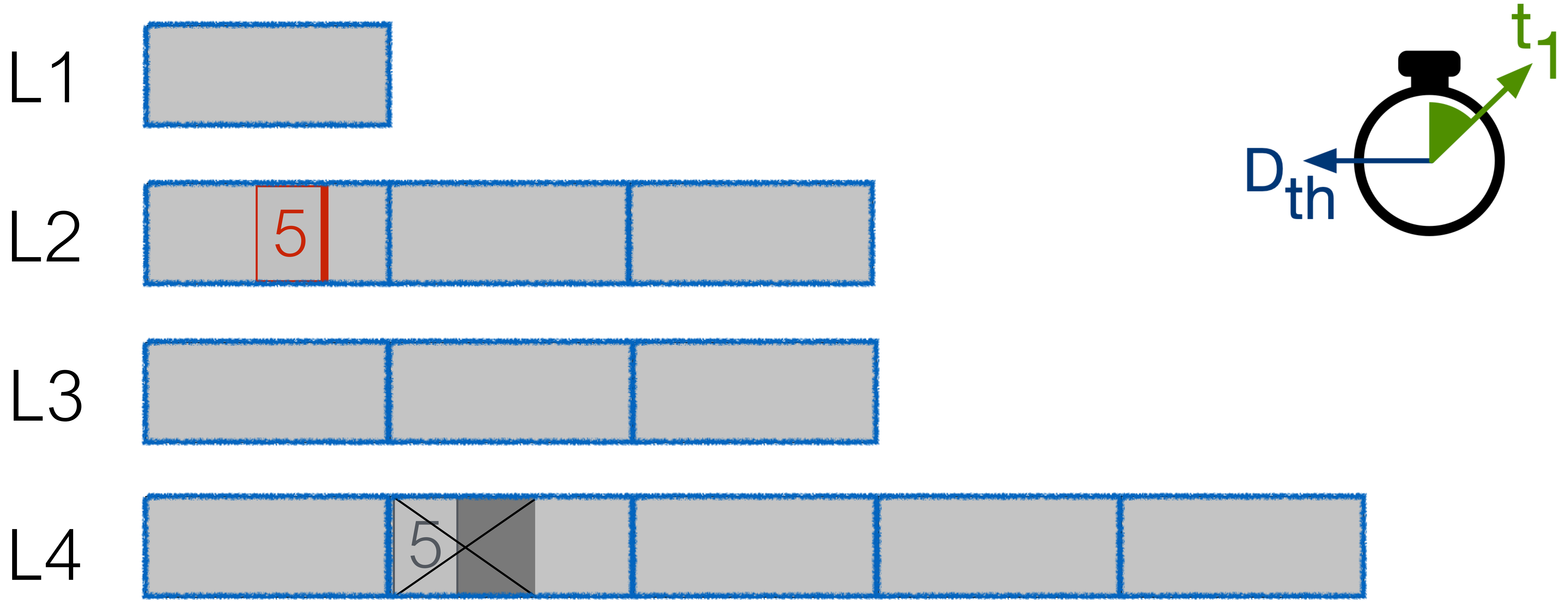
delete persistence latency

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



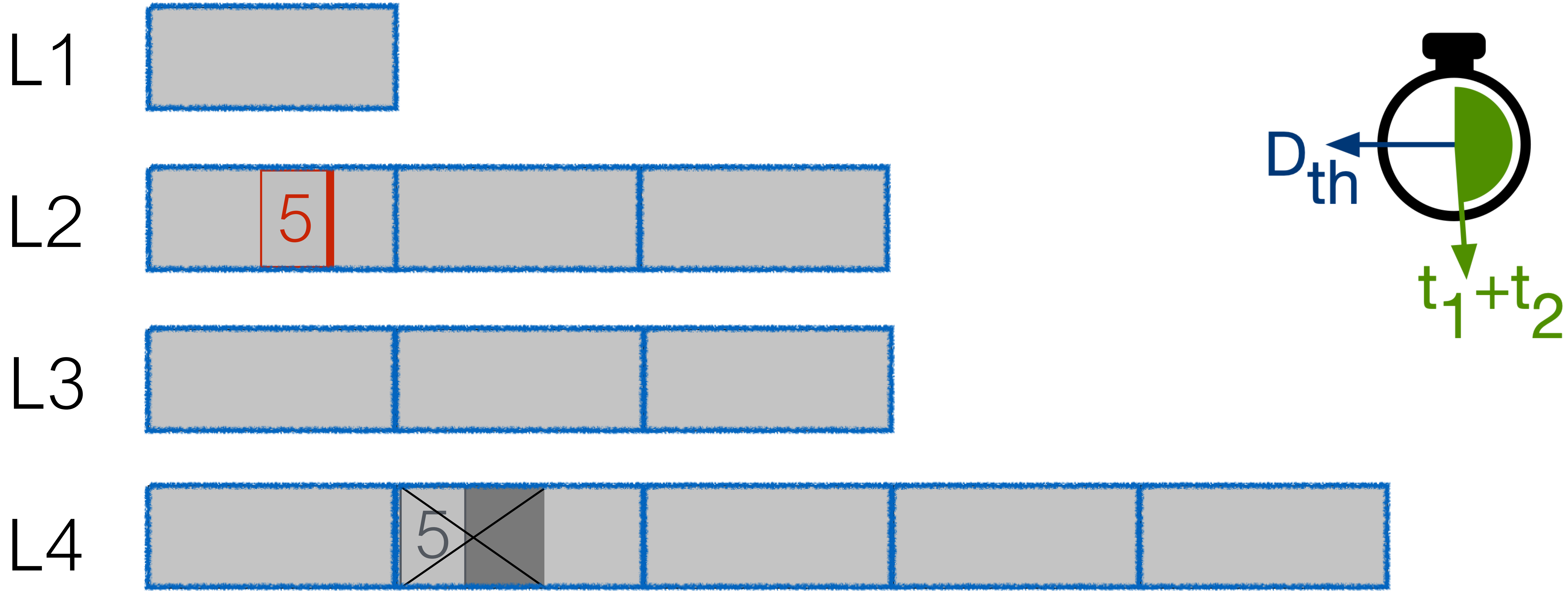
delete persistence latency

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



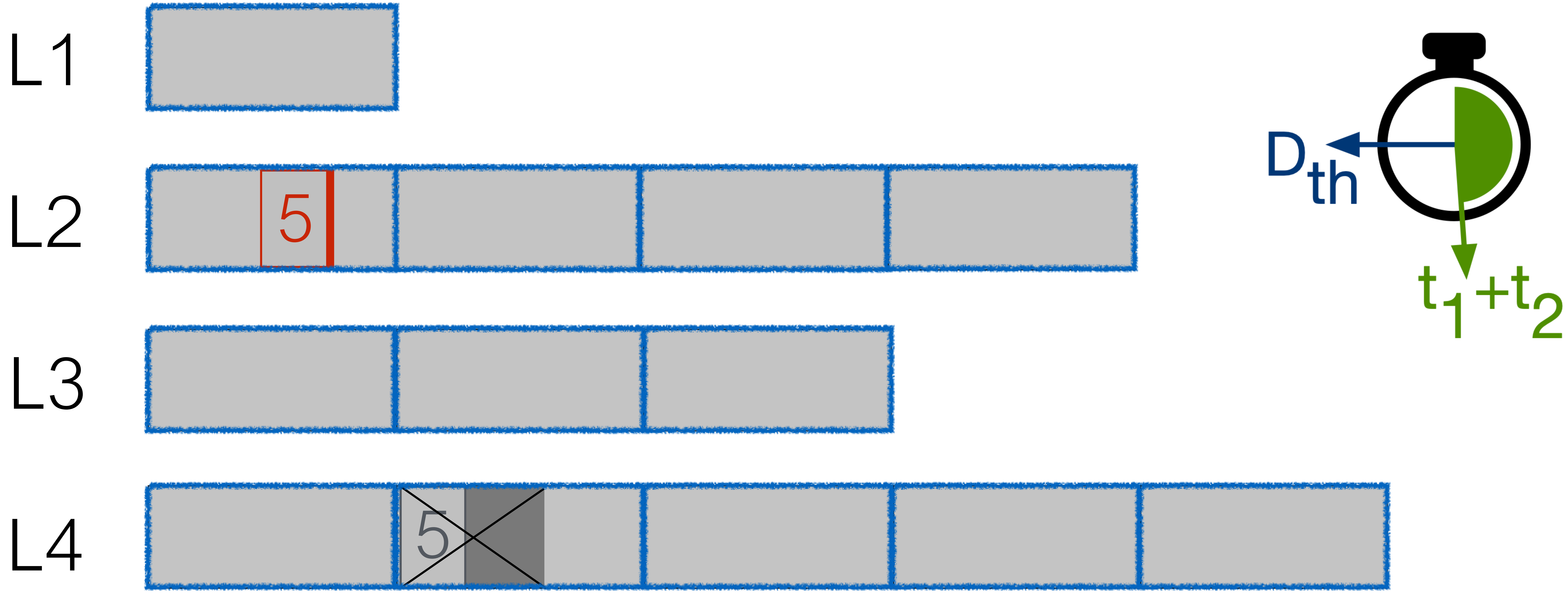
delete persistence latency

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



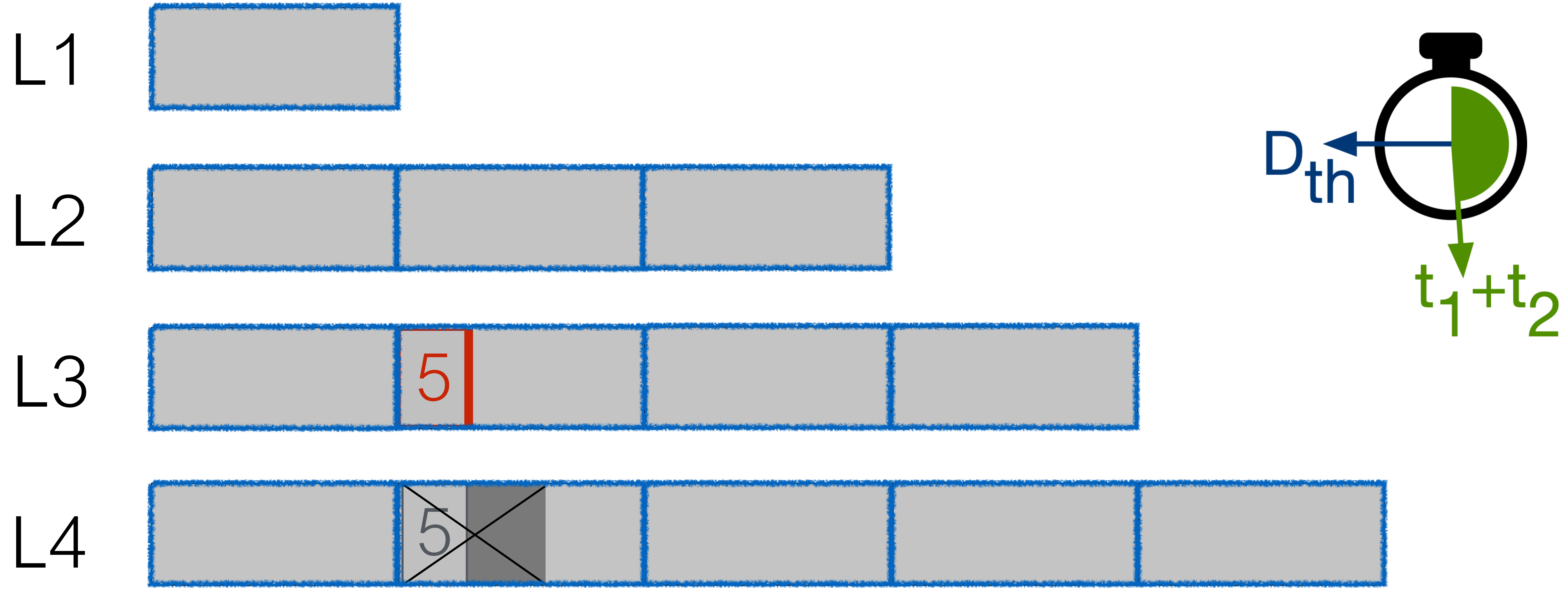
delete persistence latency

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



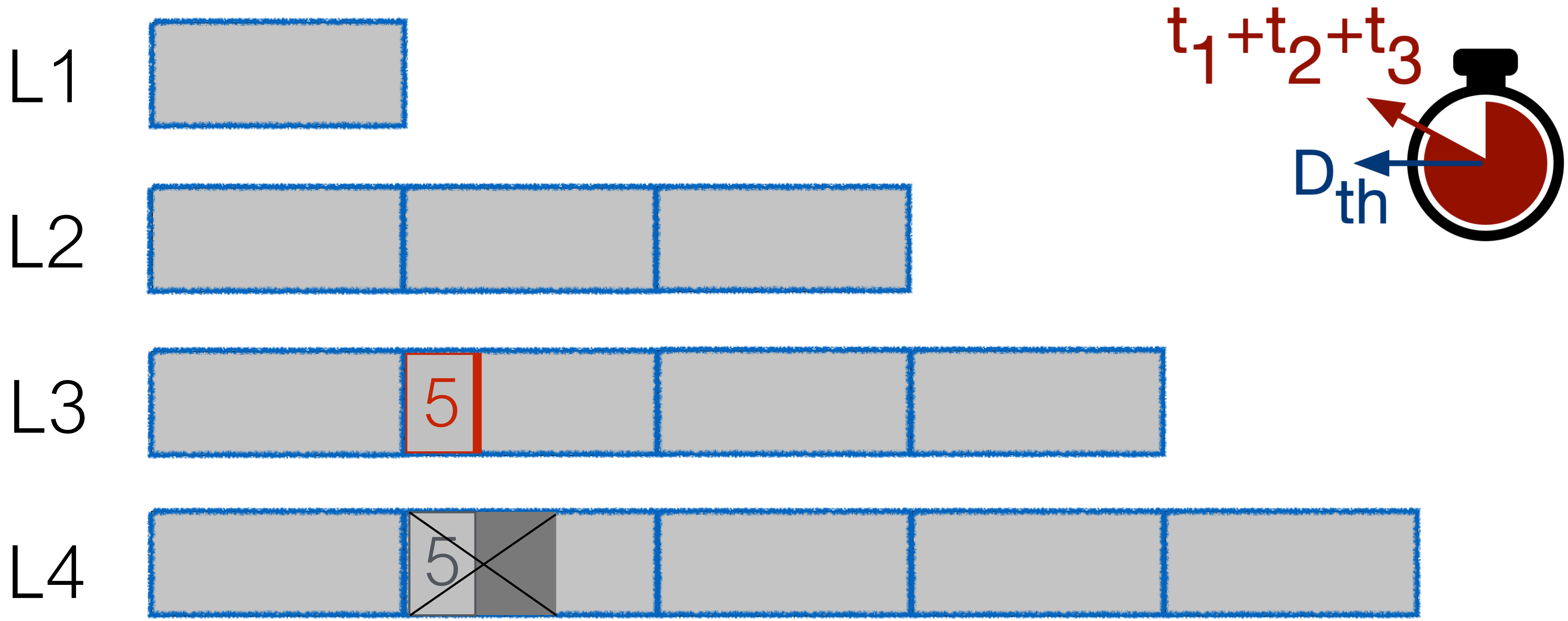
delete persistence latency

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



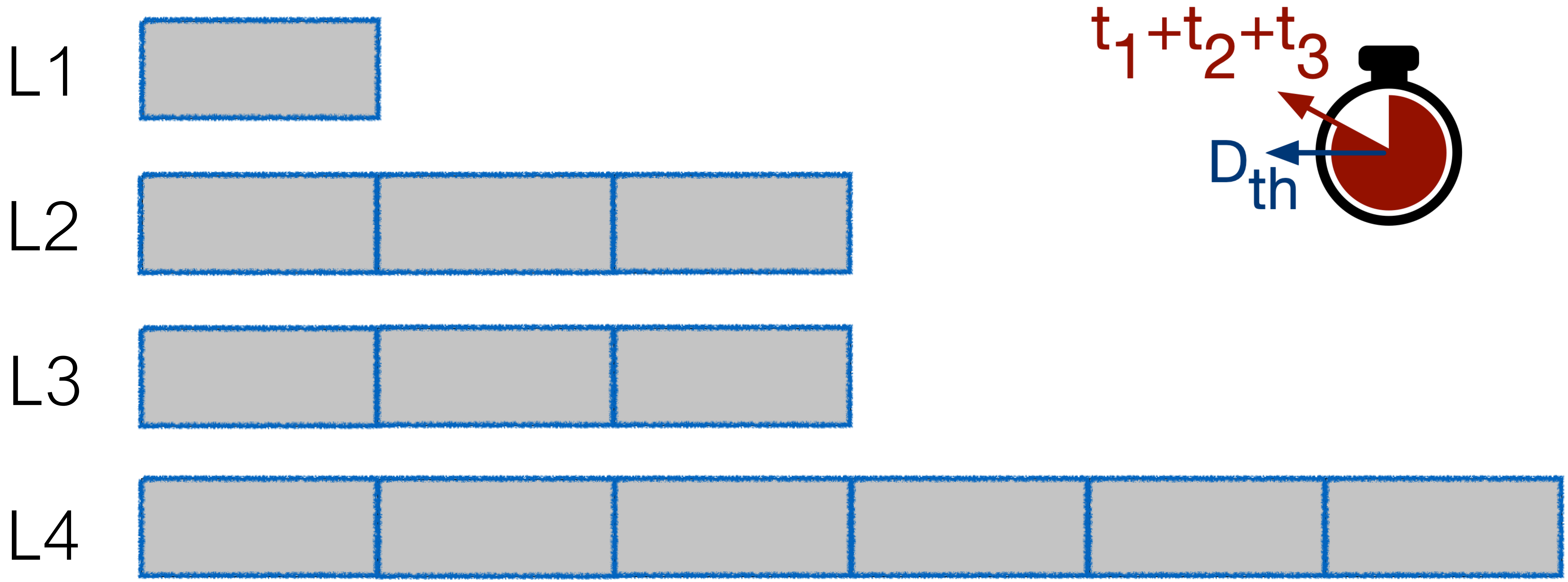
delete persistence latency

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



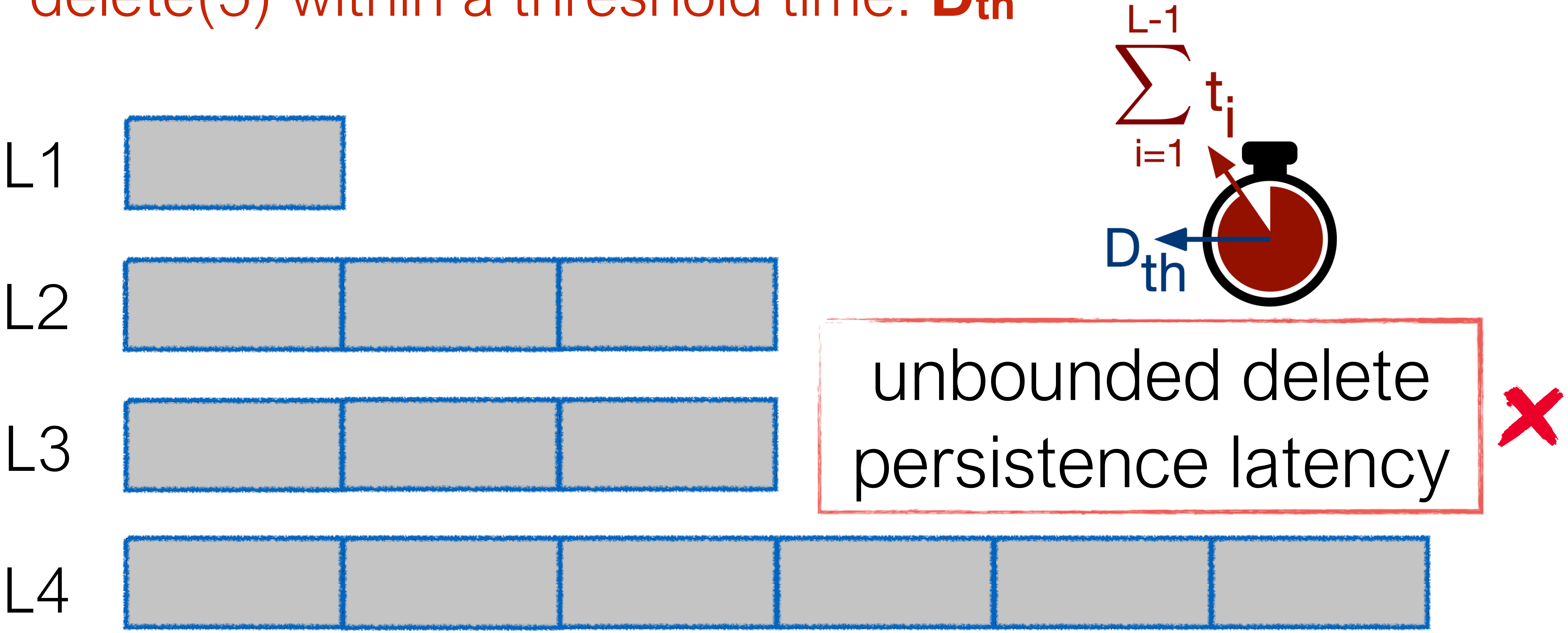
delete persistence latency

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



delete persistence latency

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



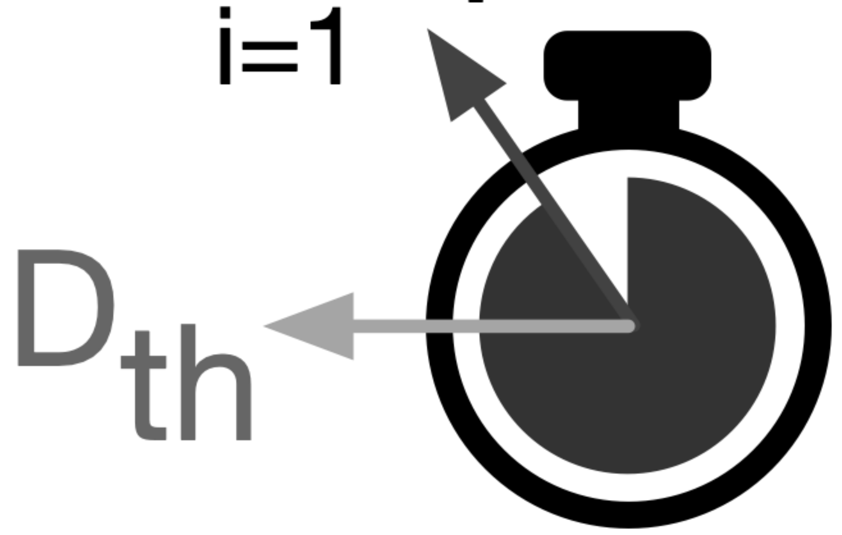
the problems

poor read perf.

write amplification

space amplification

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

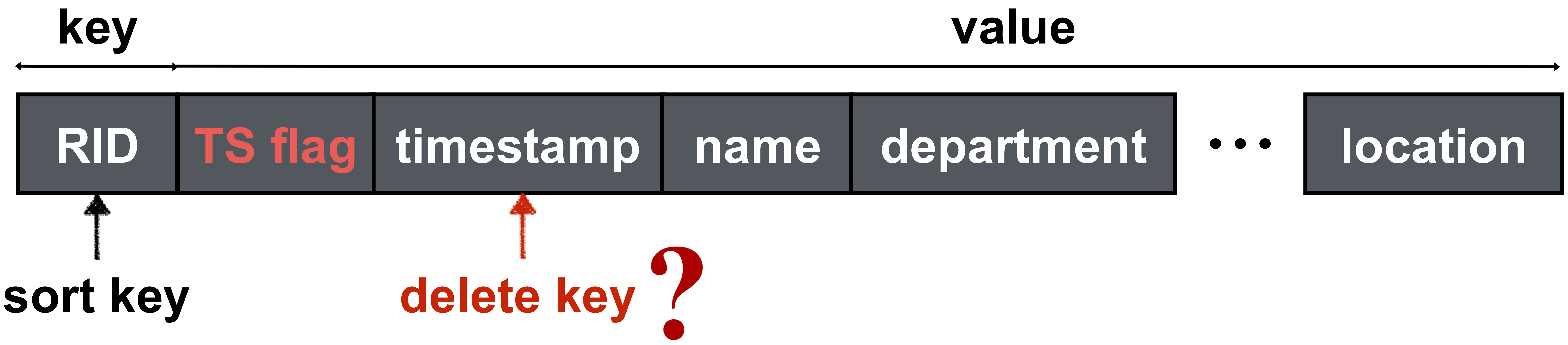


unbounded delete persistence latency

deletes on a secondary attribute

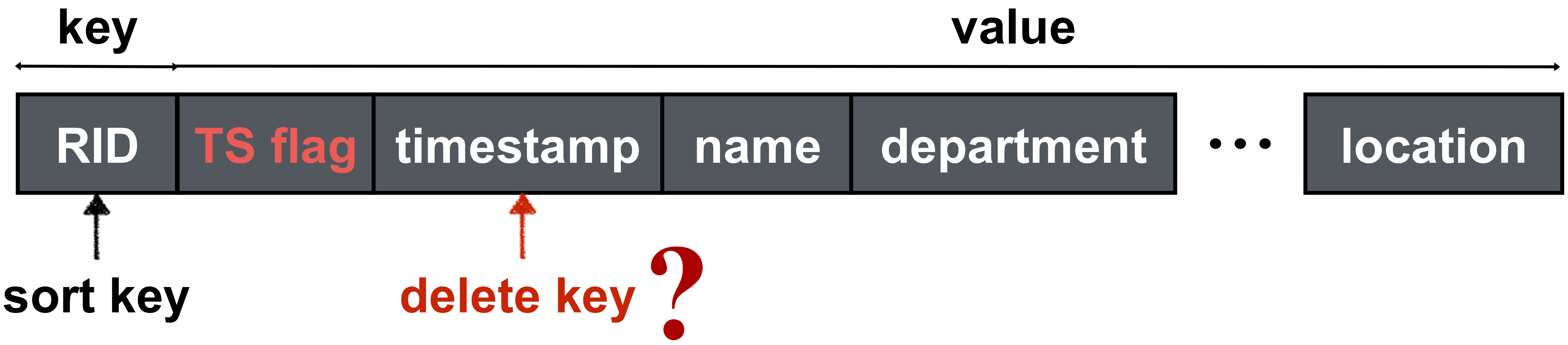
deletes on a secondary attribute

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



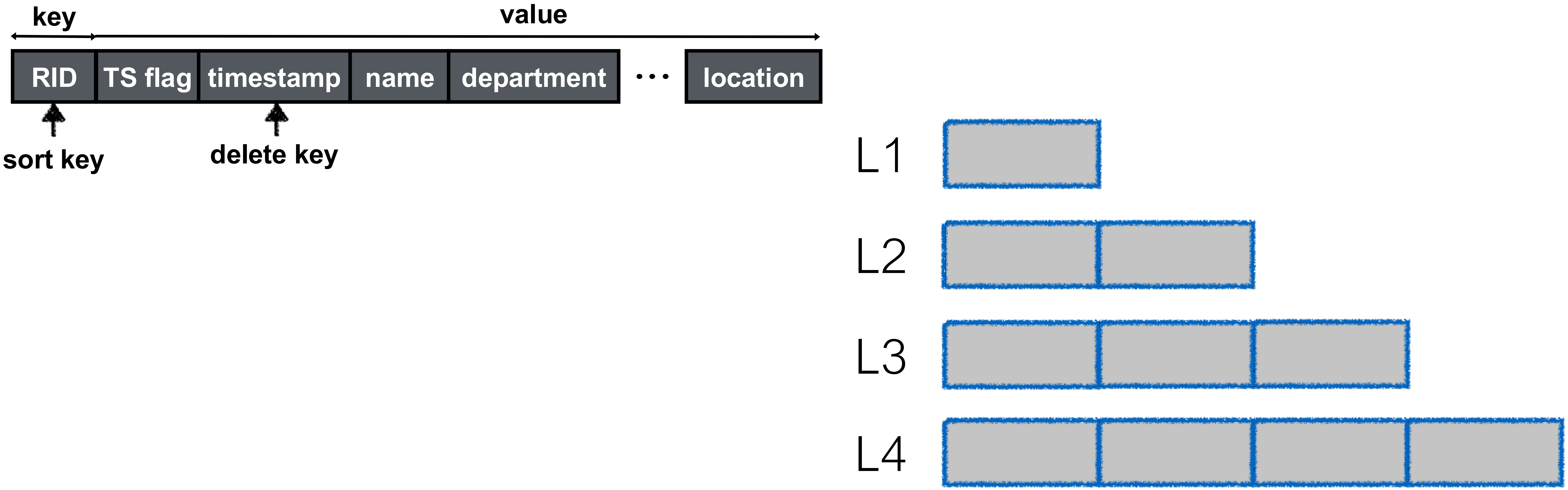
deletes on a secondary attribute

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



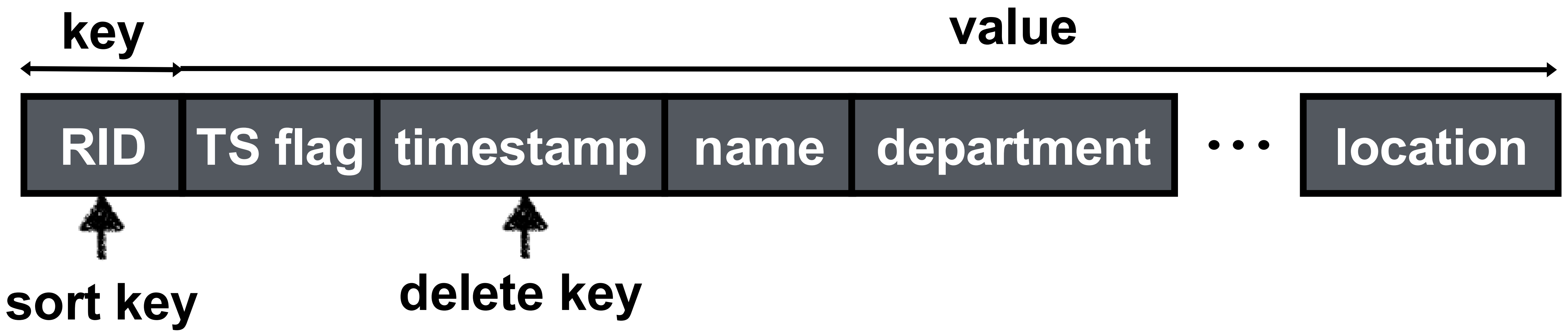
deletes on a secondary attribute

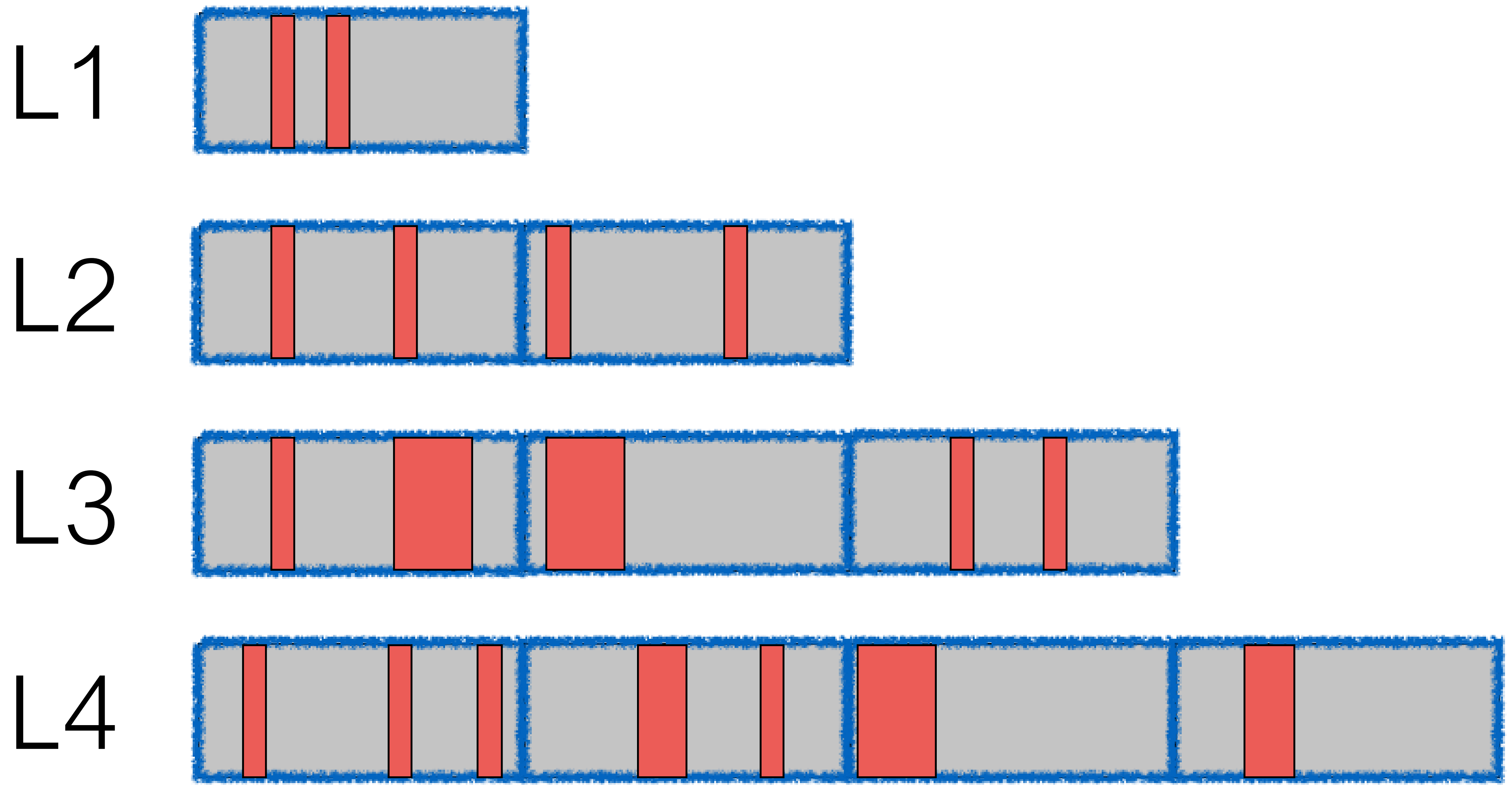
delete all entries older than: **D days**



deletes on a secondary attribute

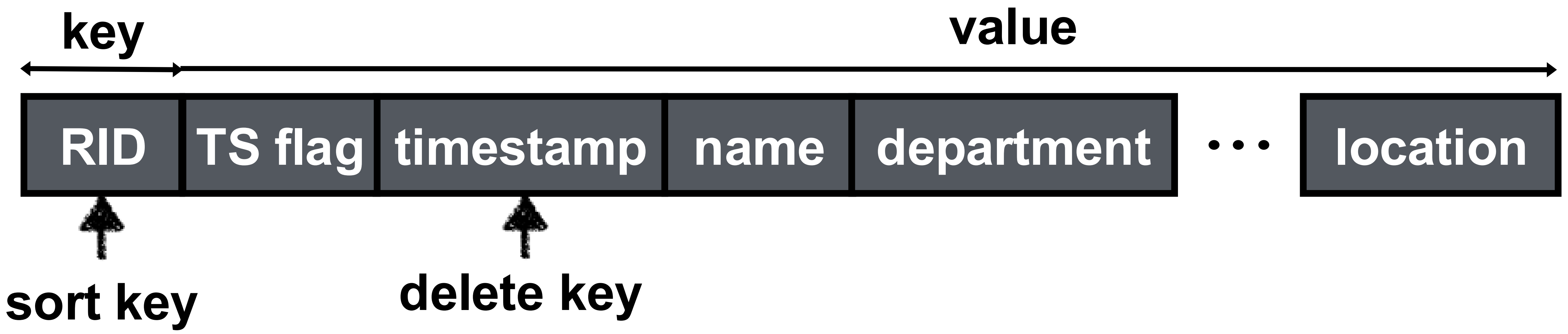
delete all entries older than: **D days**

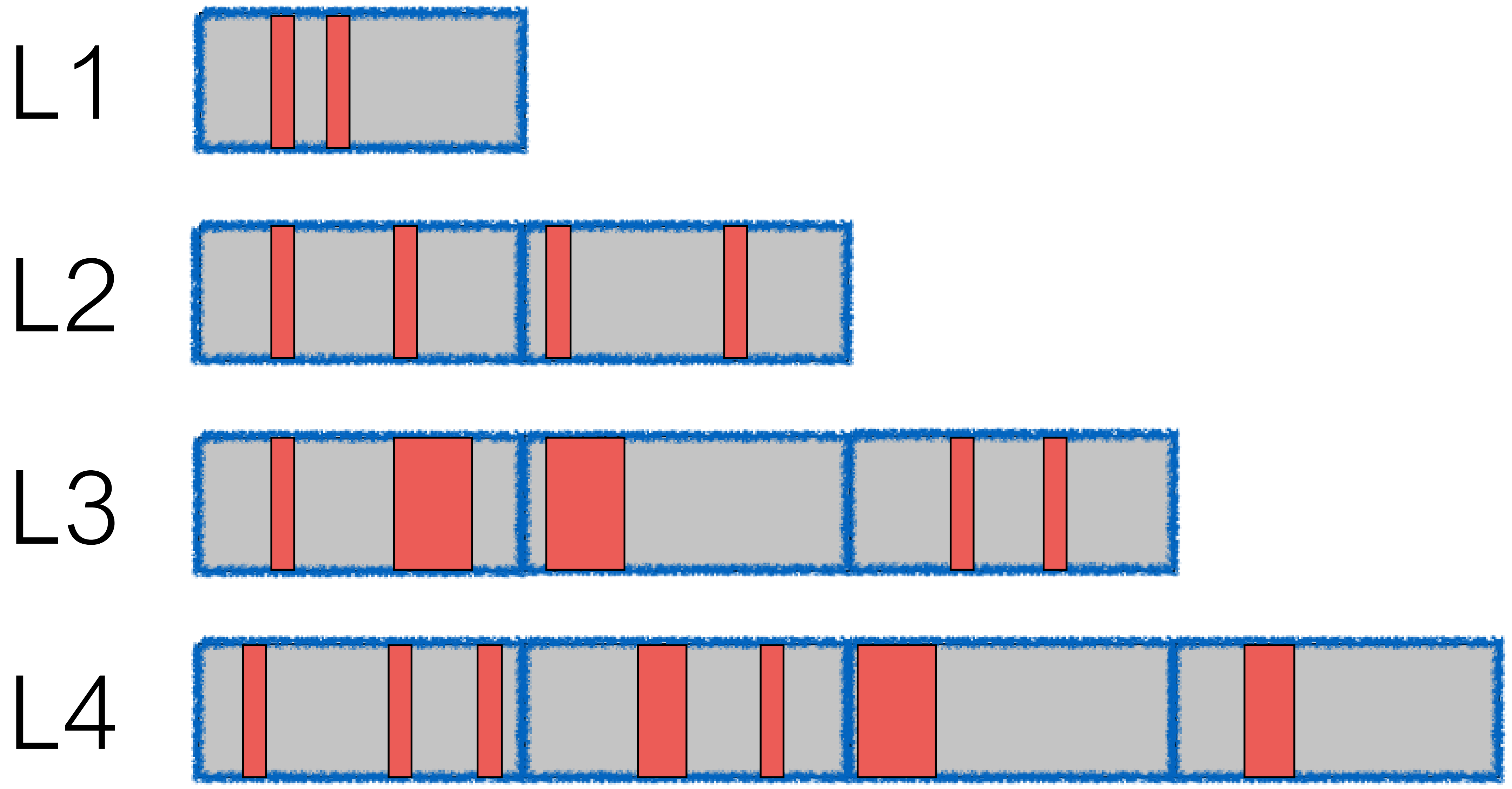




deletes on a secondary attribute

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

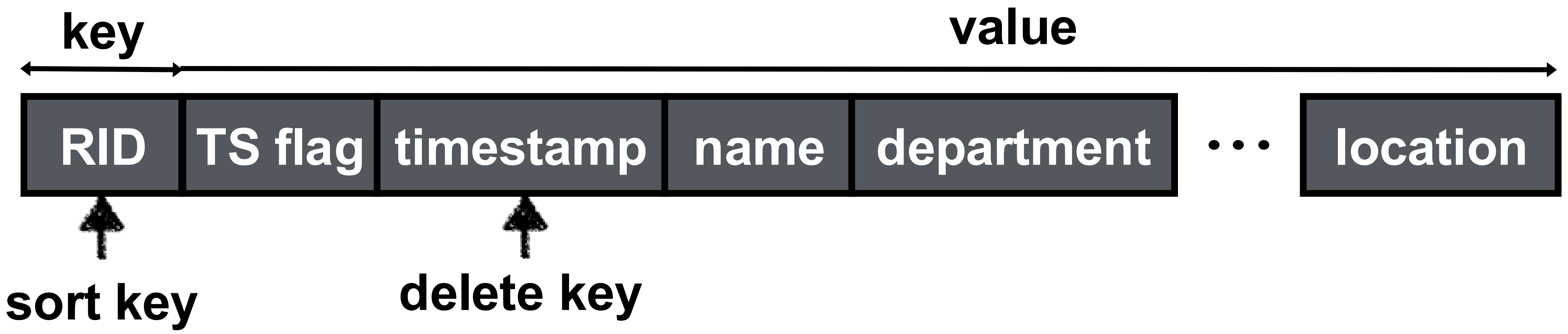




scattered occurrences

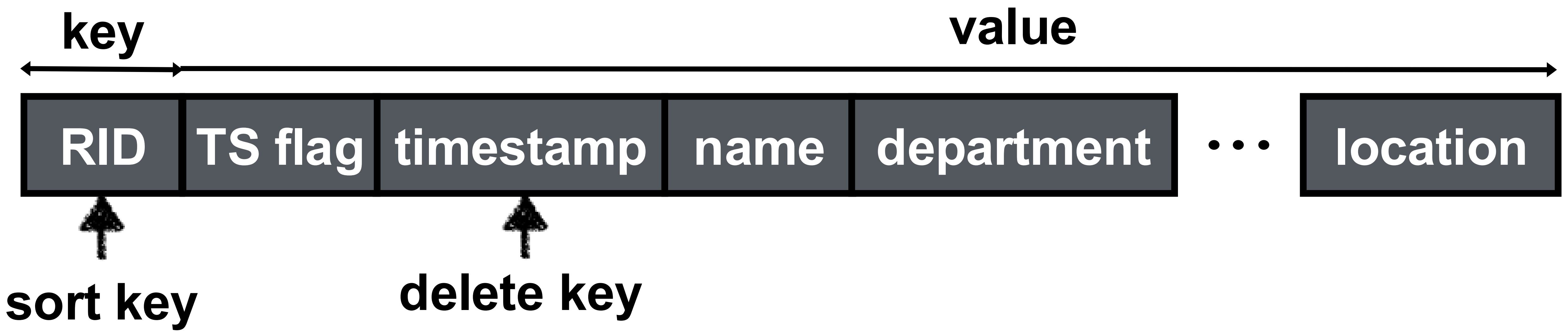
deletes on a secondary attribute

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



deletes on a secondary attribute

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

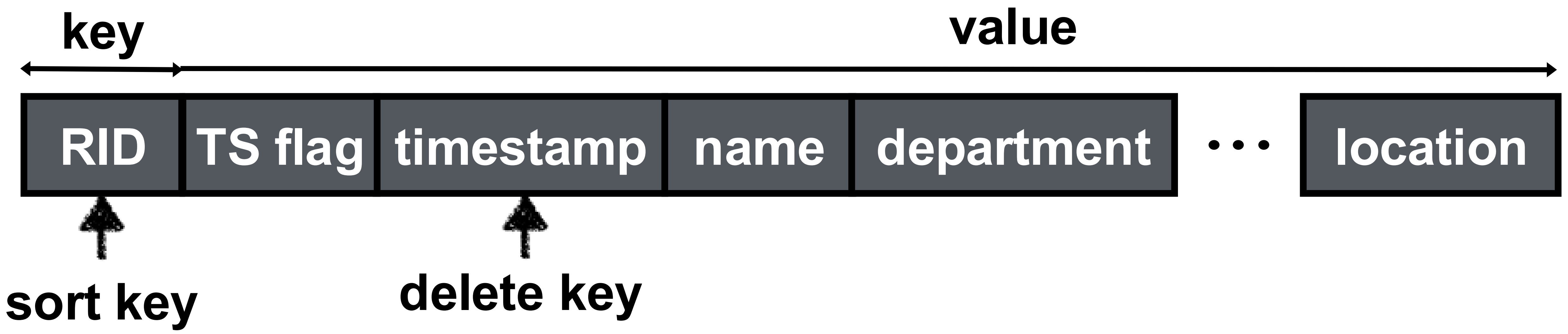


latency spikes ✗

superfluous I/Os ✗

deletes on a secondary attribute

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



latency spikes ✗

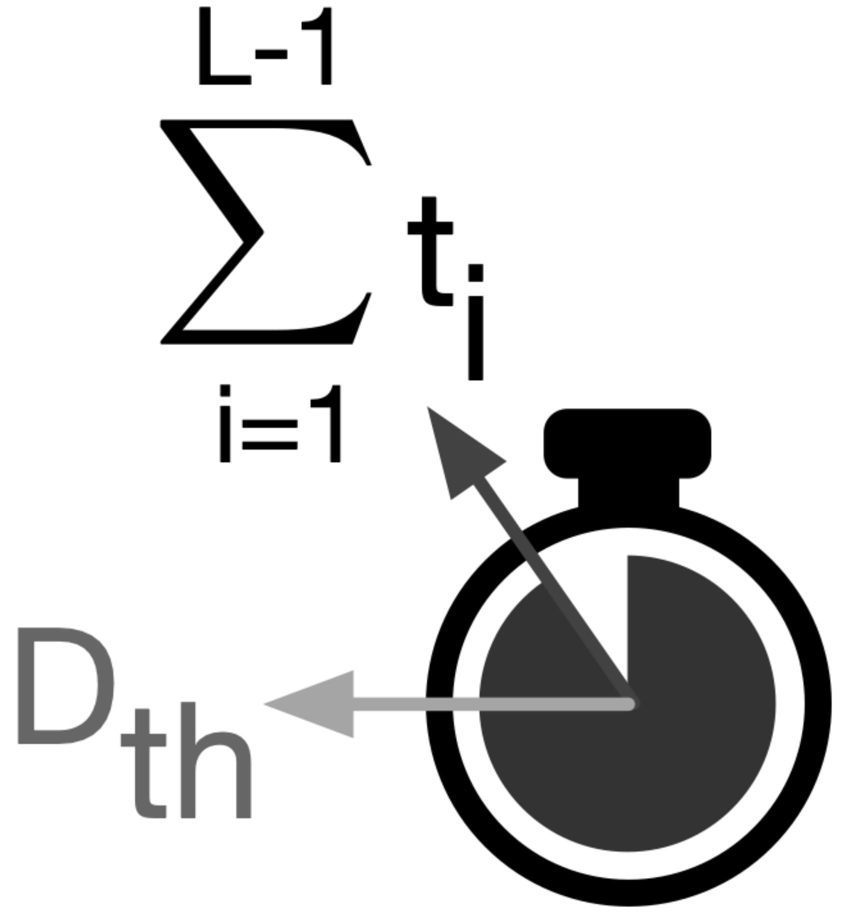
superfluous I/Os ✗

the problems

poor read perf.

write amplification

space amplification



unbounded delete persistence latency

latency spikes

superfluous I/Os

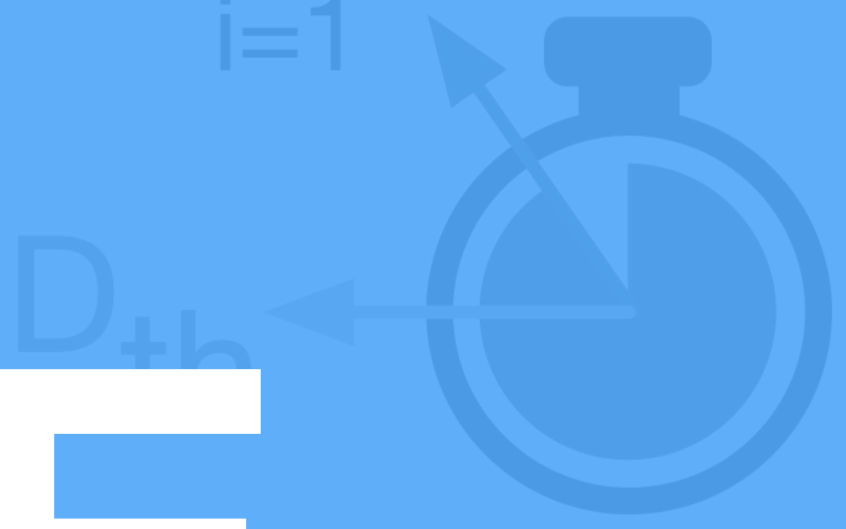
the solution

poor read perf.

write amplification

space amplification

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



FADE

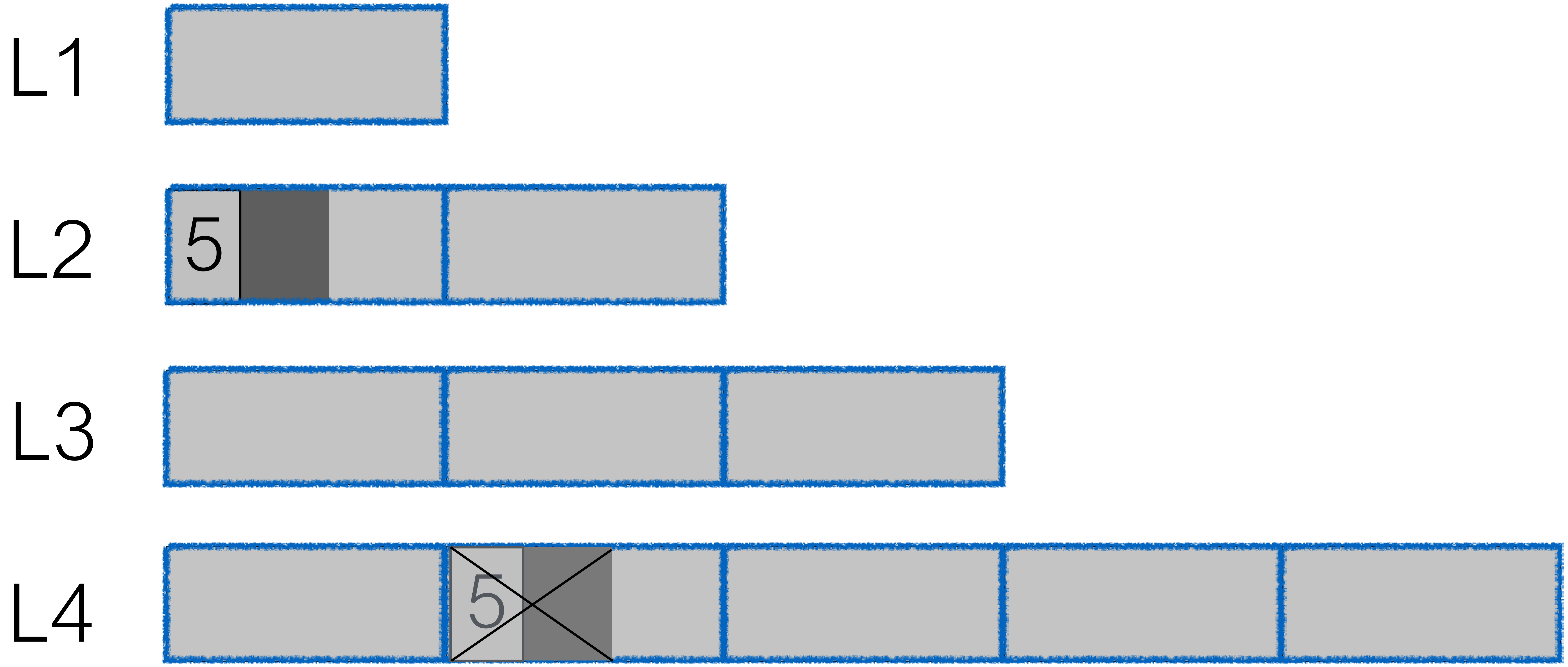
unbounded delete persistence latency

latency spikes

superfluous I/Os

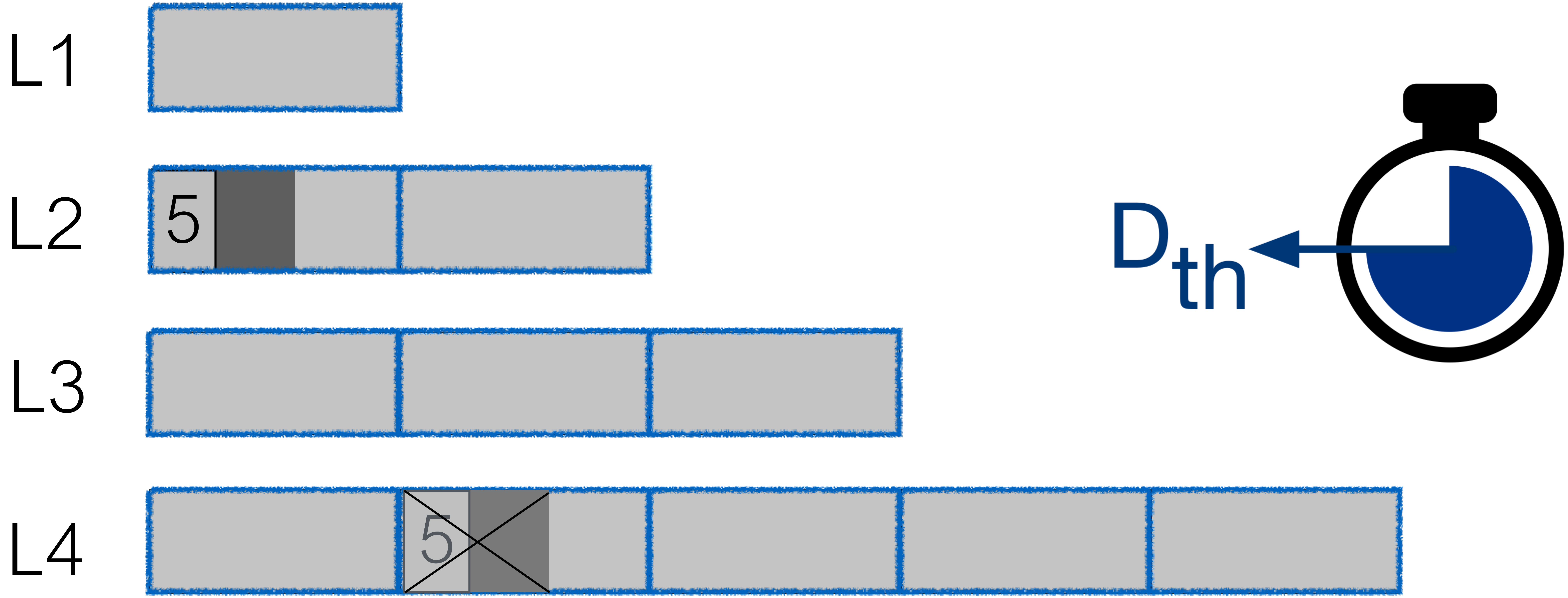
FAst DElete

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



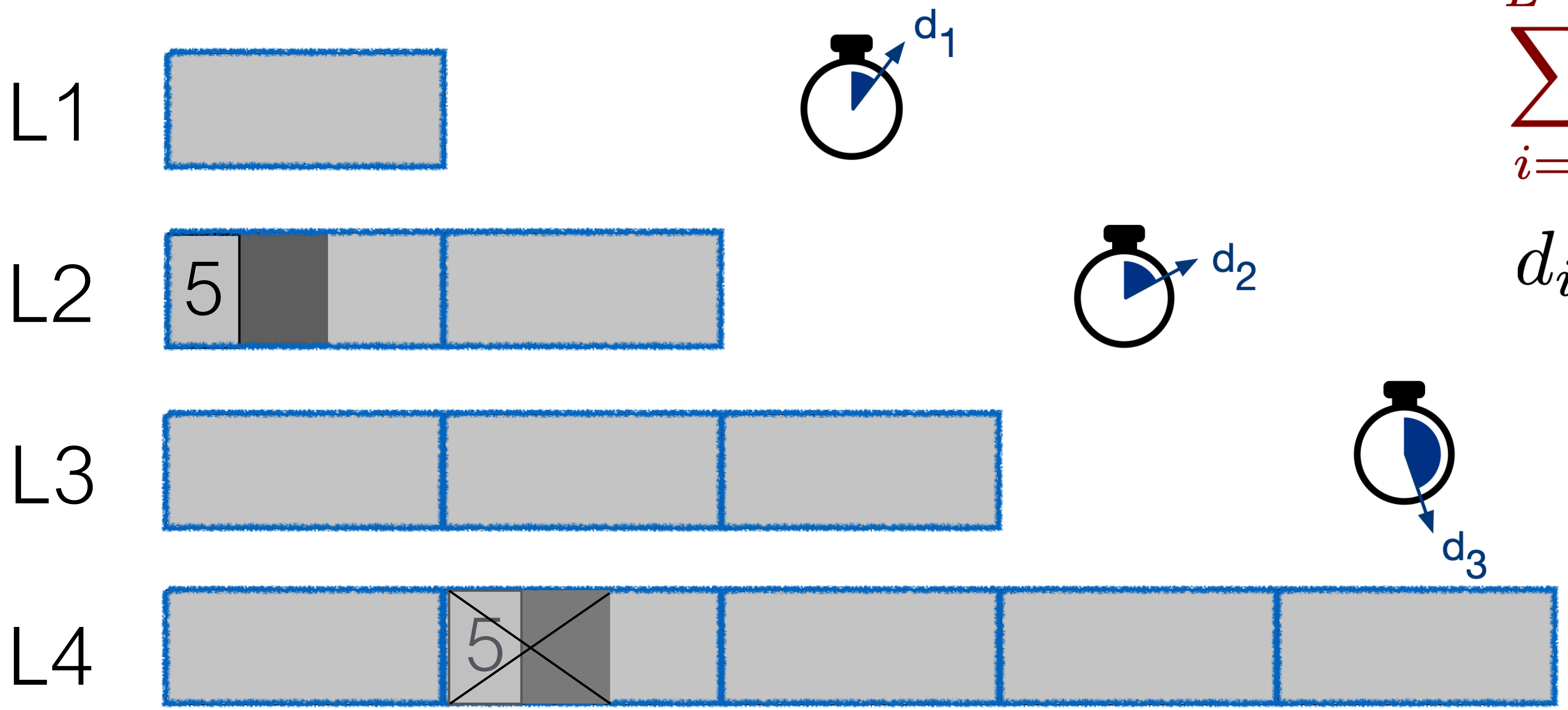
FAst DElete

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



FAst DElete

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

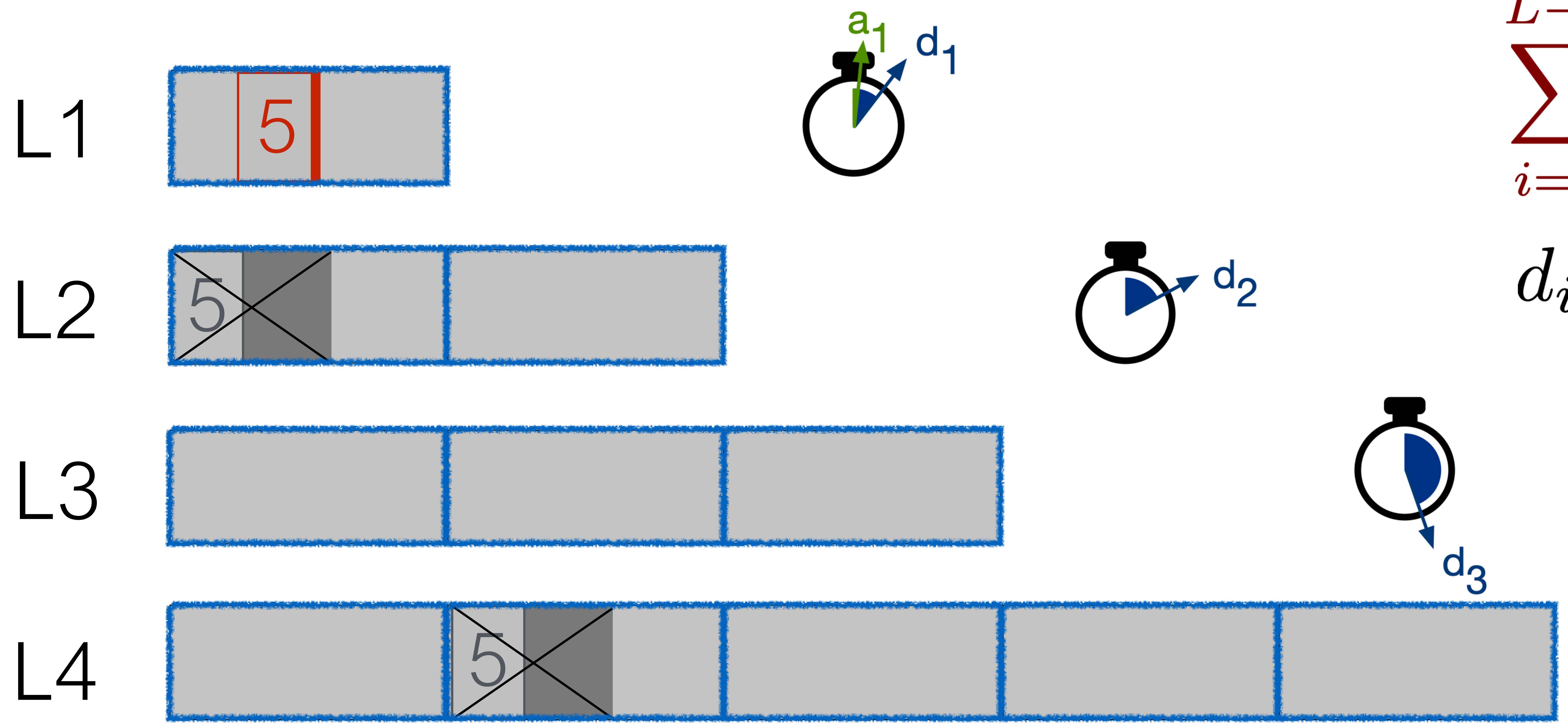


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

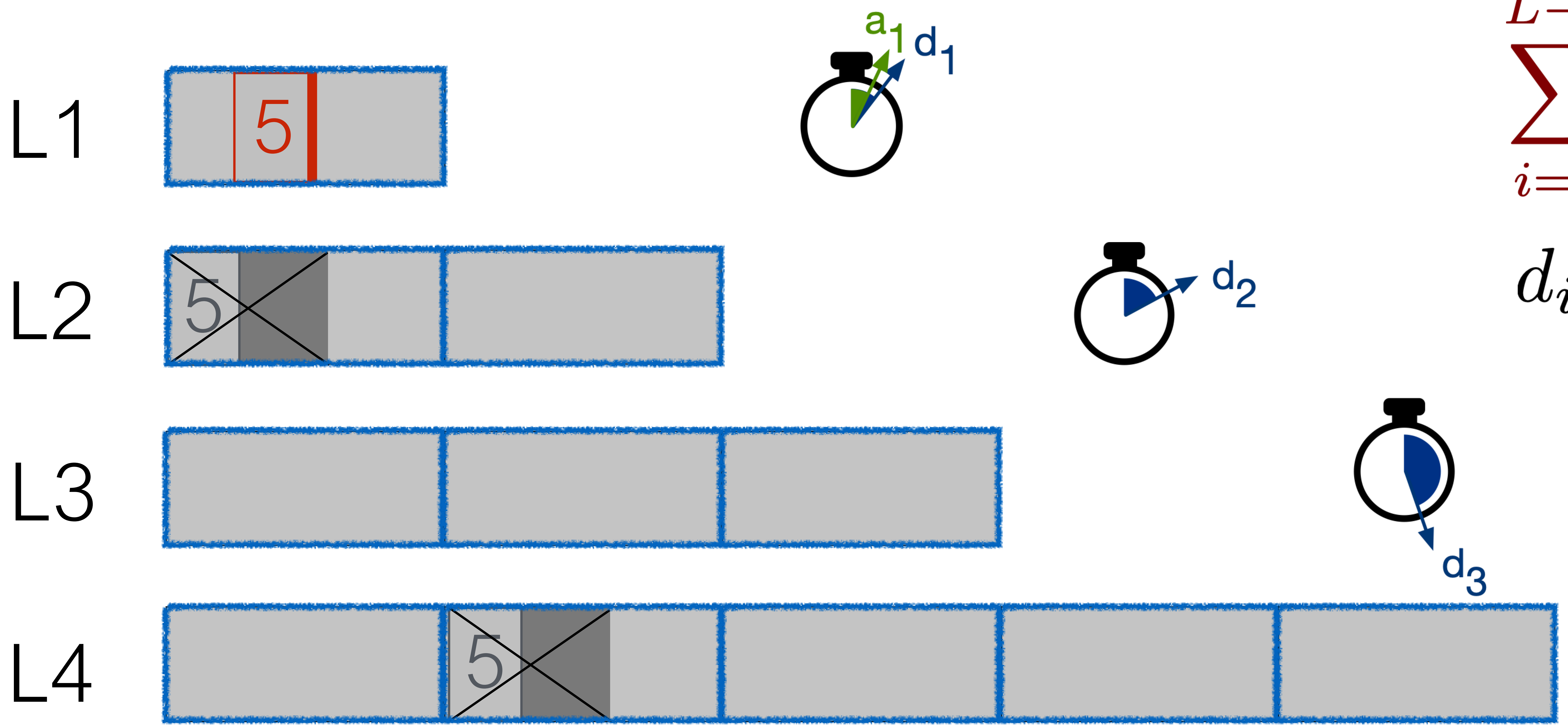


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

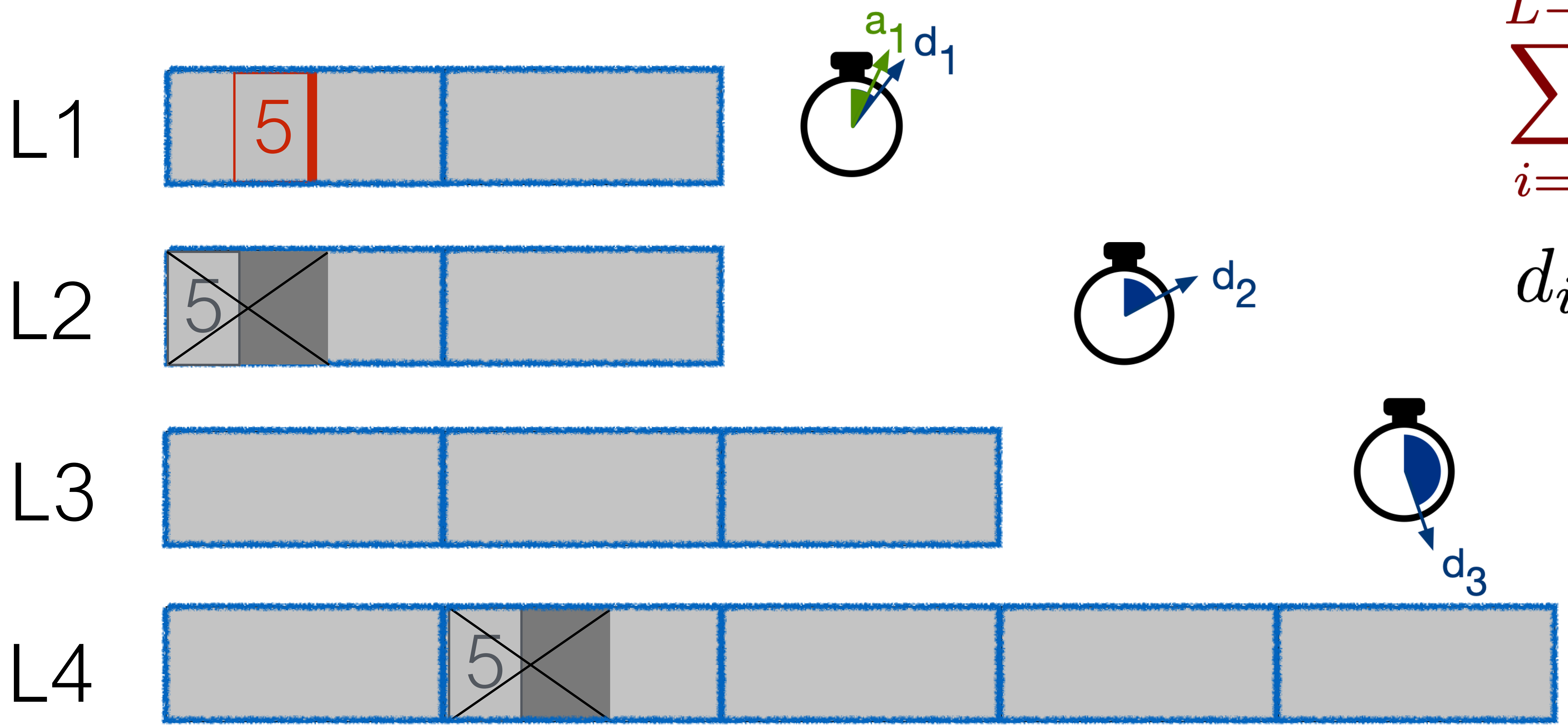


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

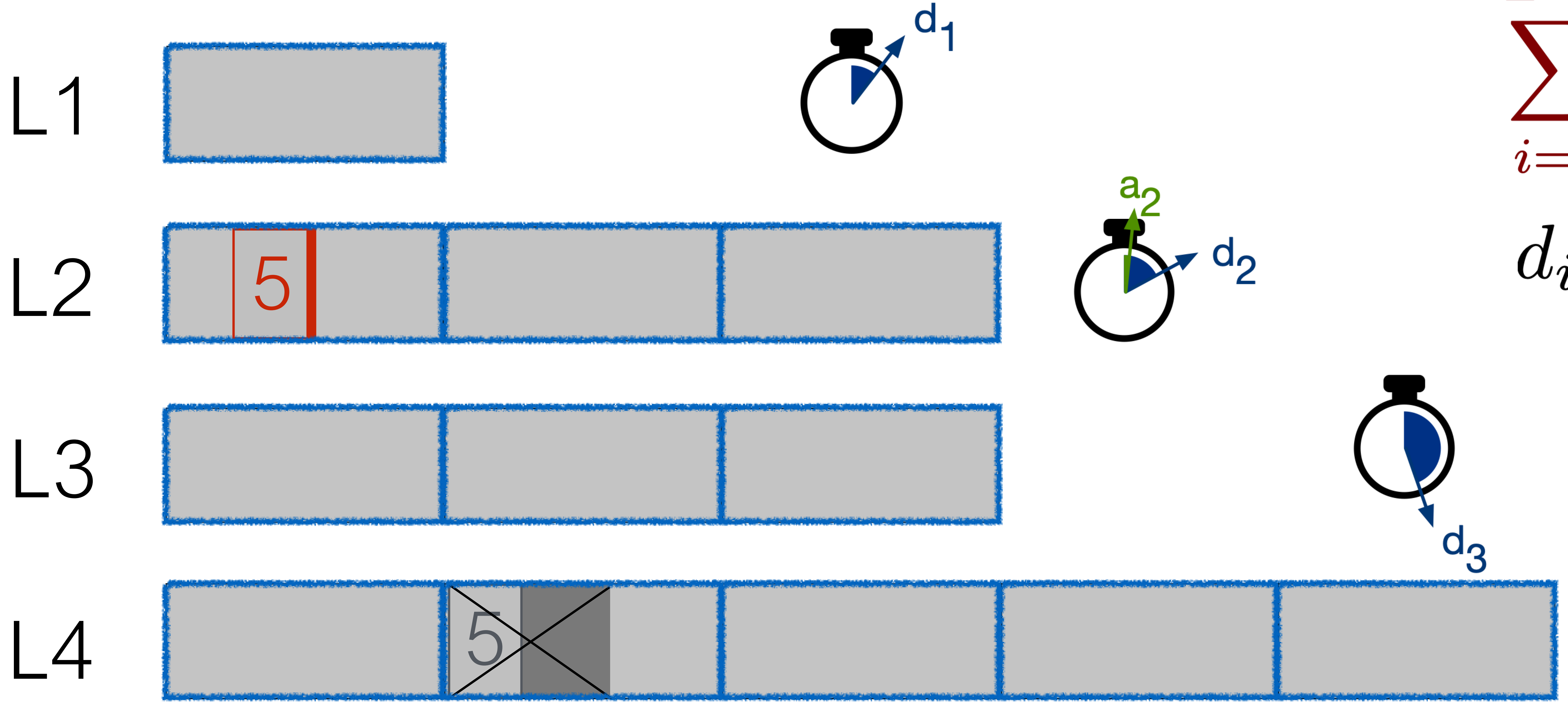


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

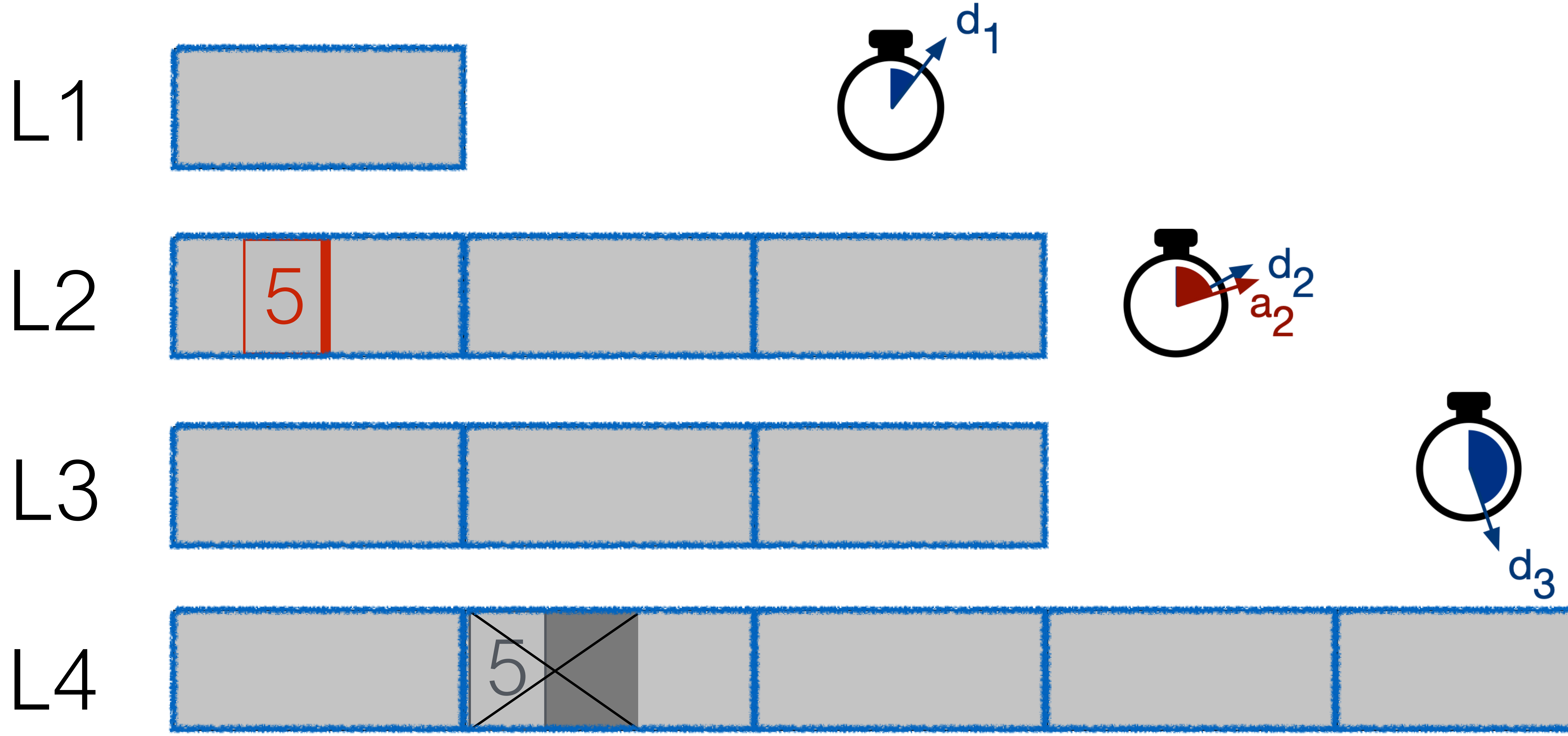


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

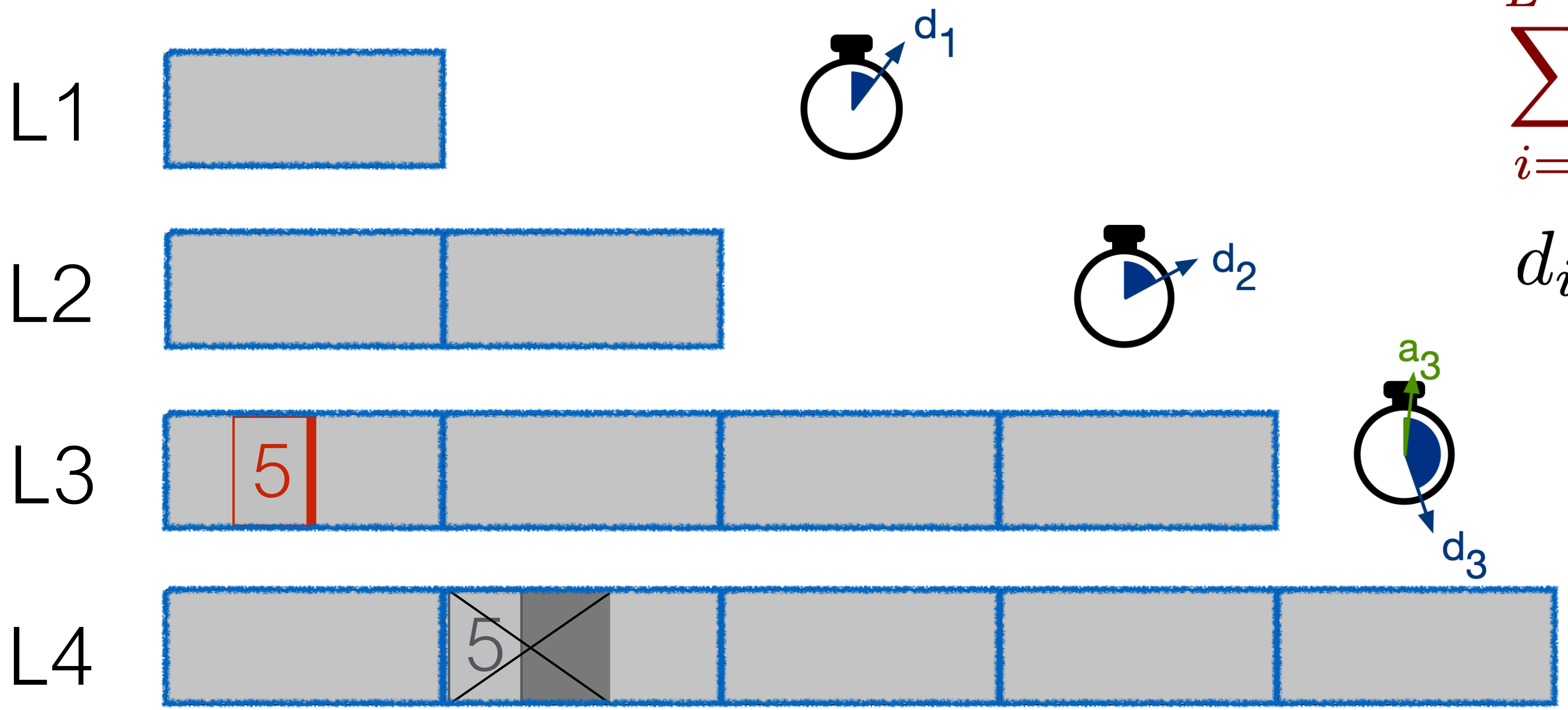


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

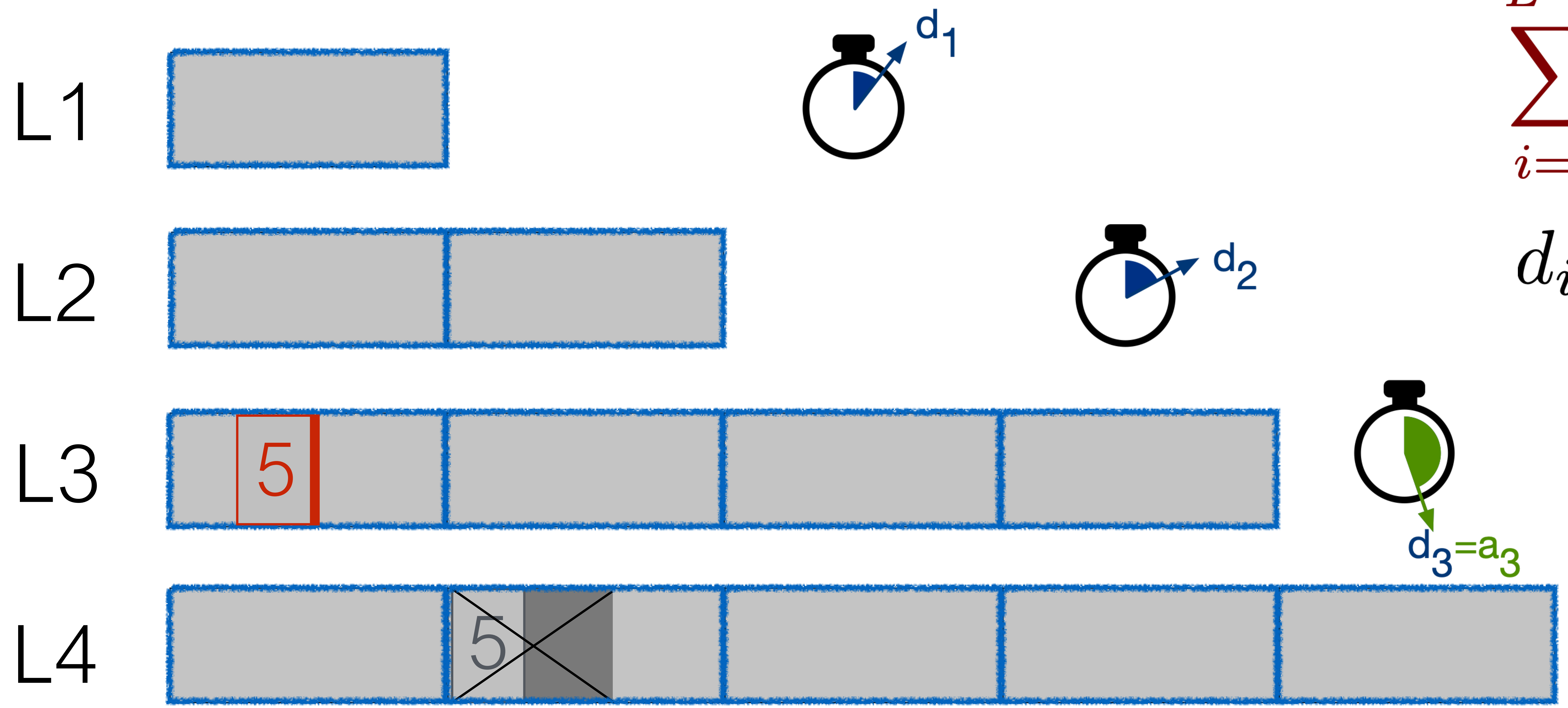


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

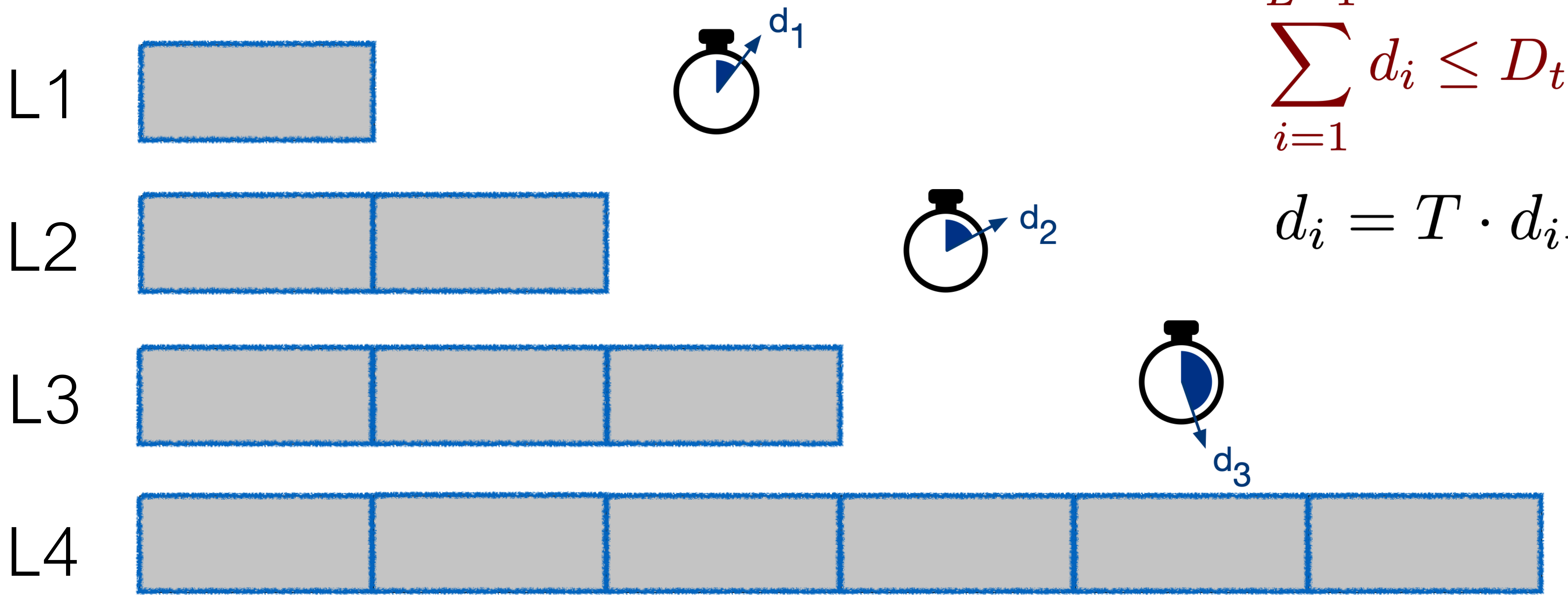


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

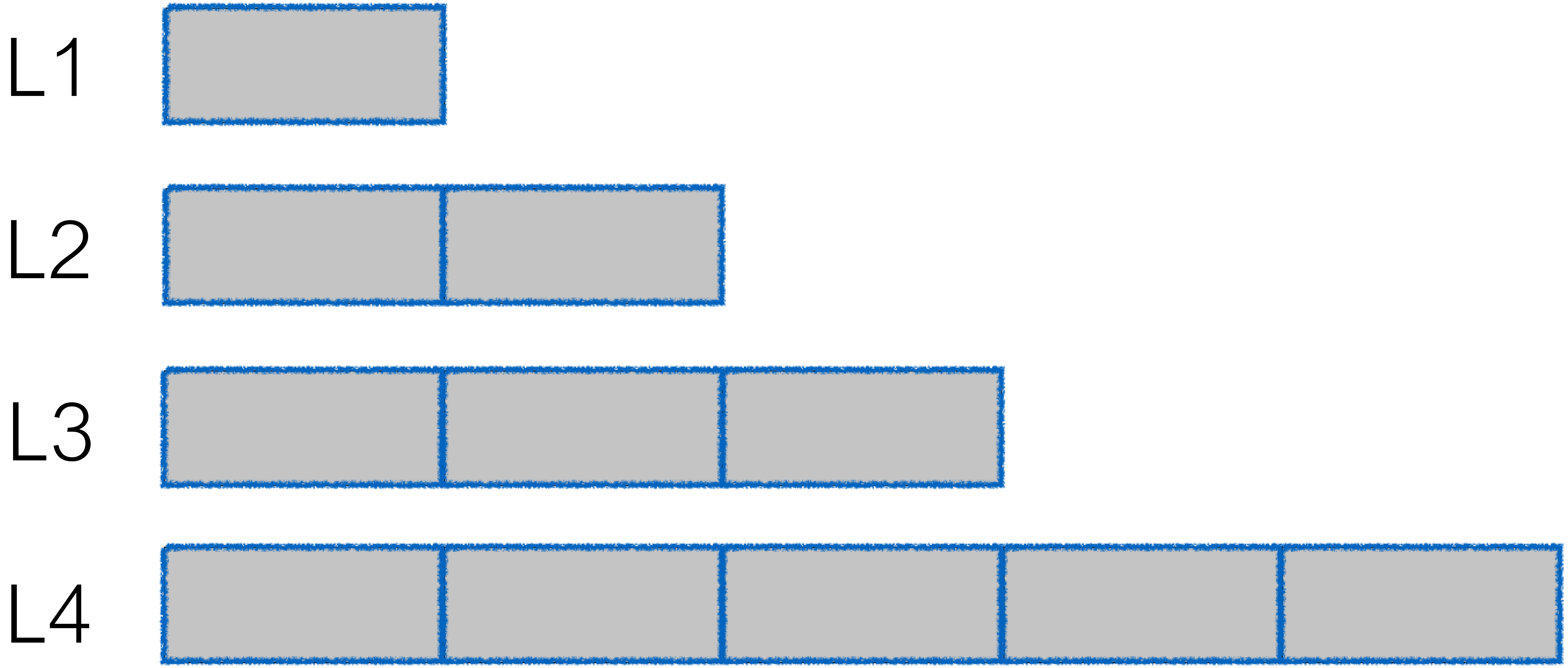


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

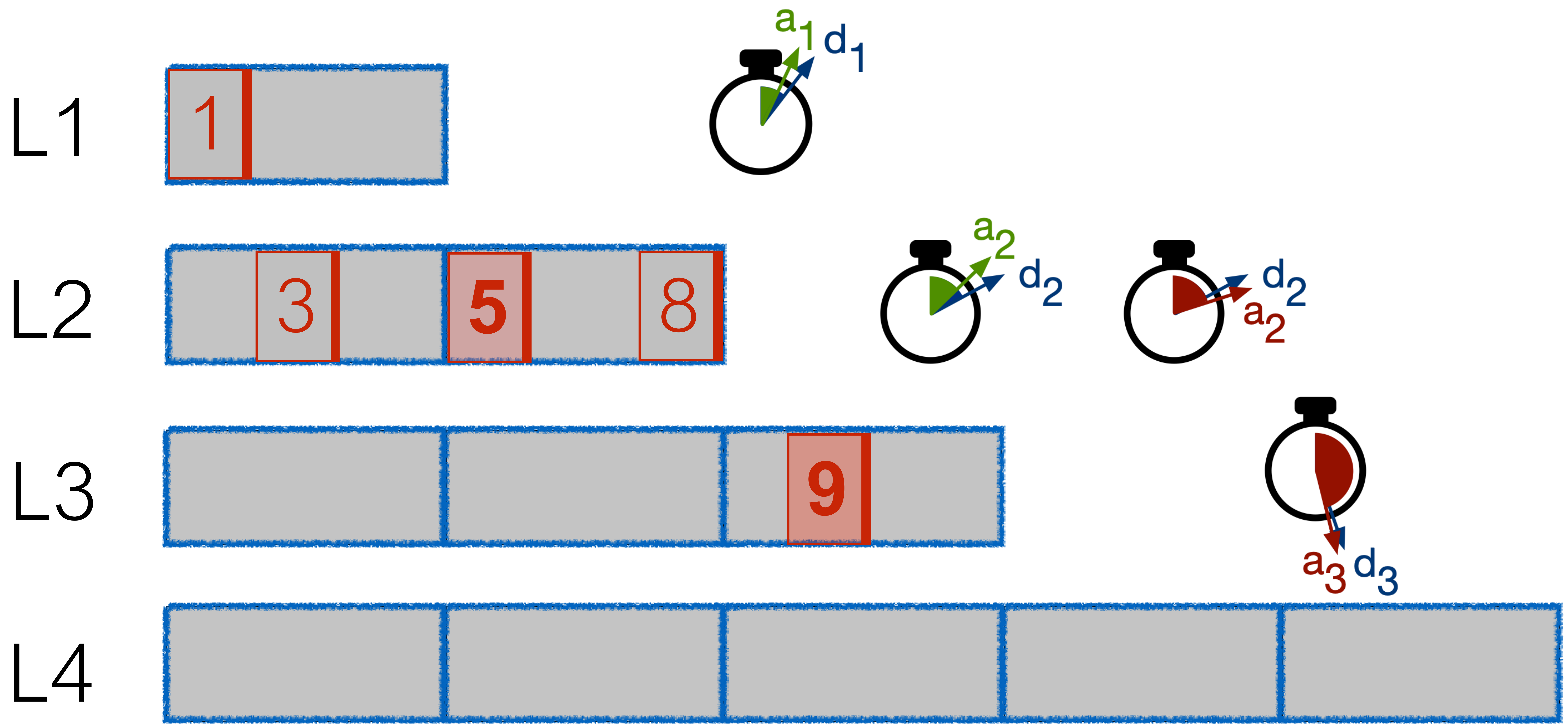
FAst DElete

breaking ties in practical workloads



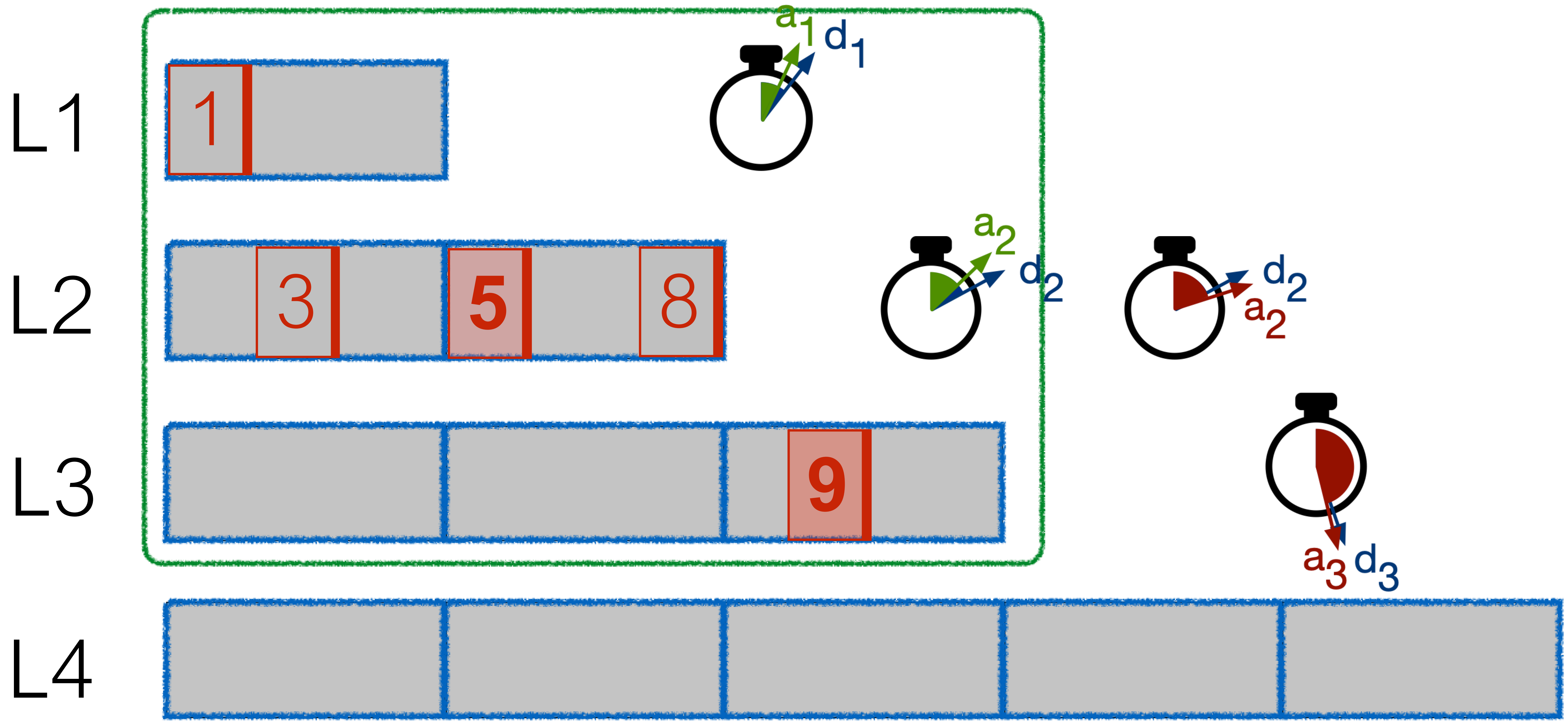
FAst DElete

breaking ties in practical workloads



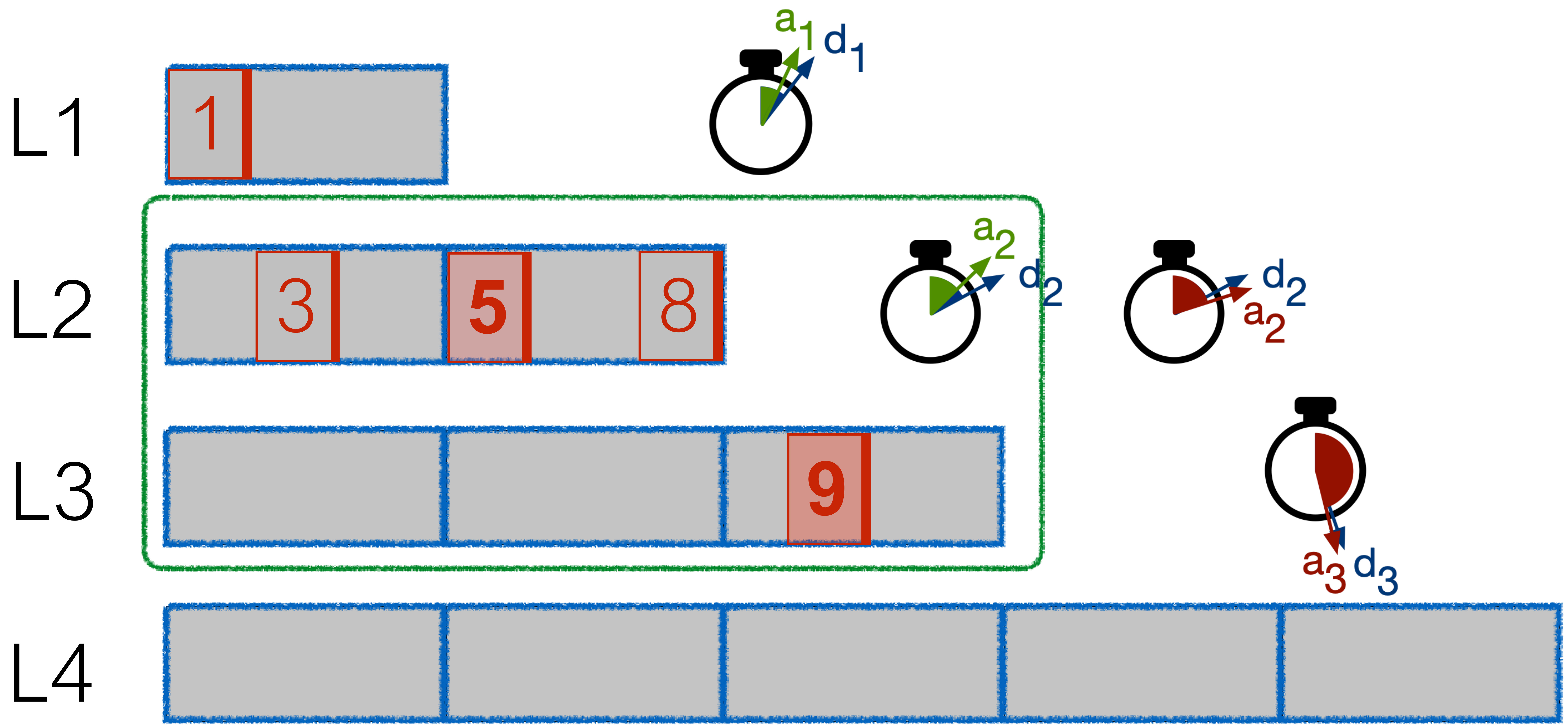
FAst DElete

breaking ties in practical workloads



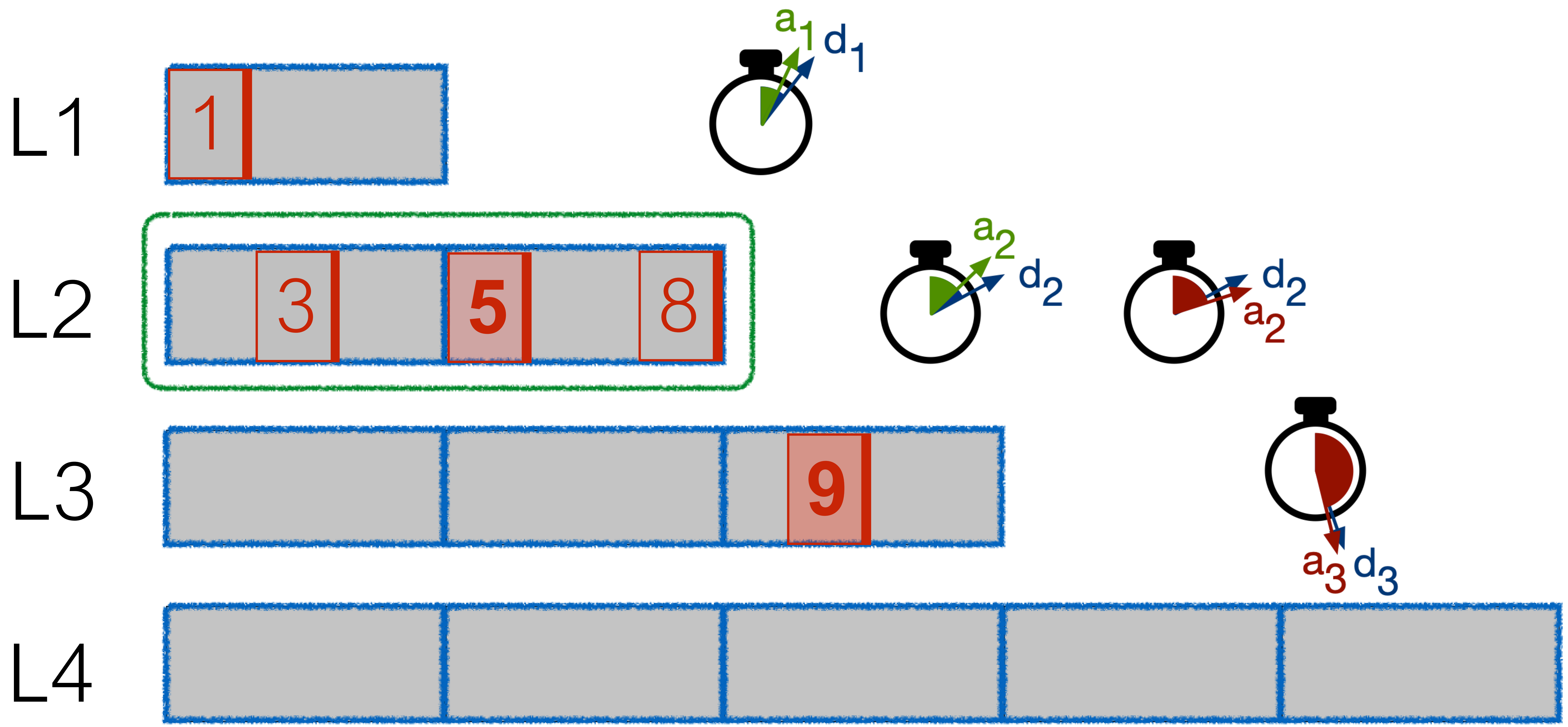
FAst DElete

breaking ties in practical workloads



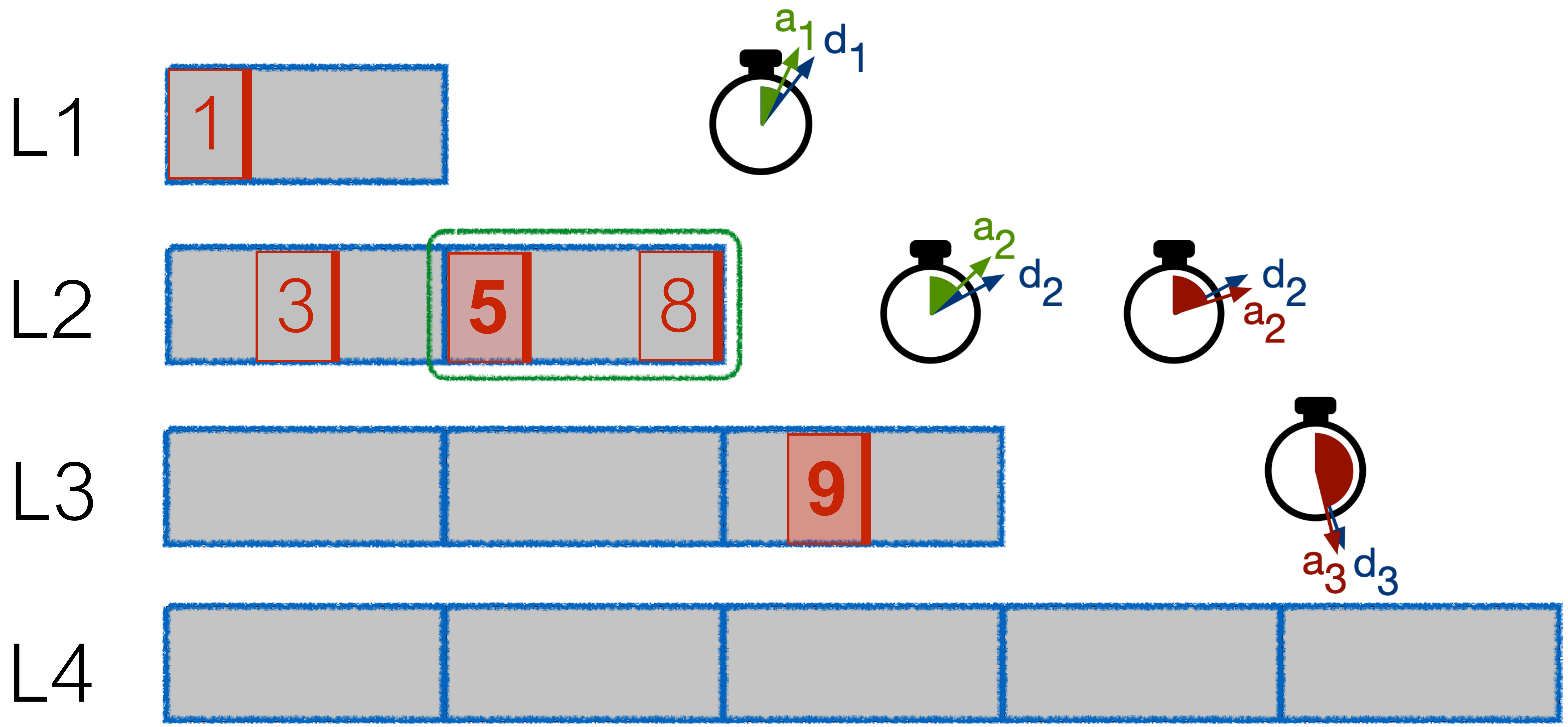
FAst DElete

breaking ties in practical workloads



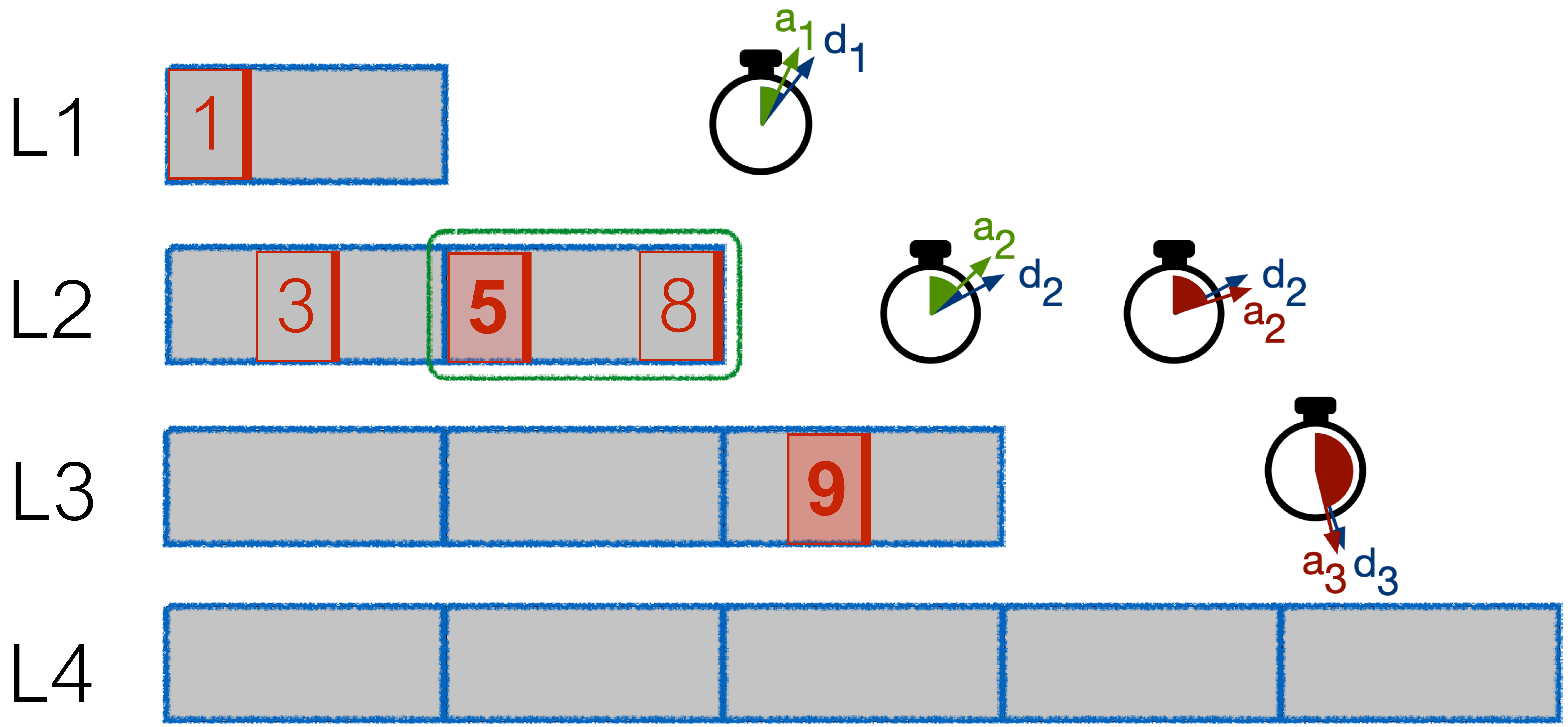
FAst DElete

breaking ties in practical workloads

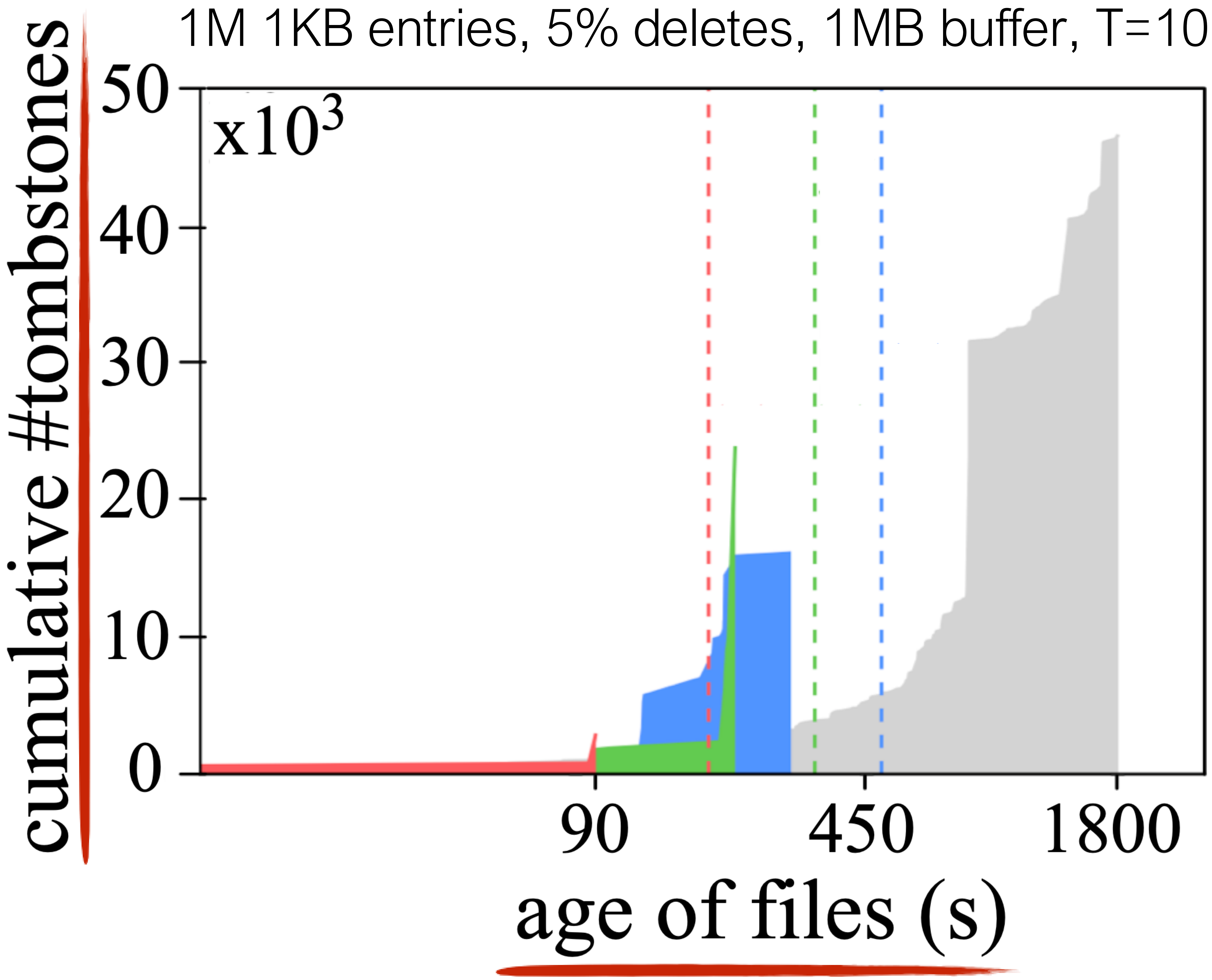


FAst DElete

breaking ties in practical workloads



FAst DElete



timely delete persistence ✔

within D_th

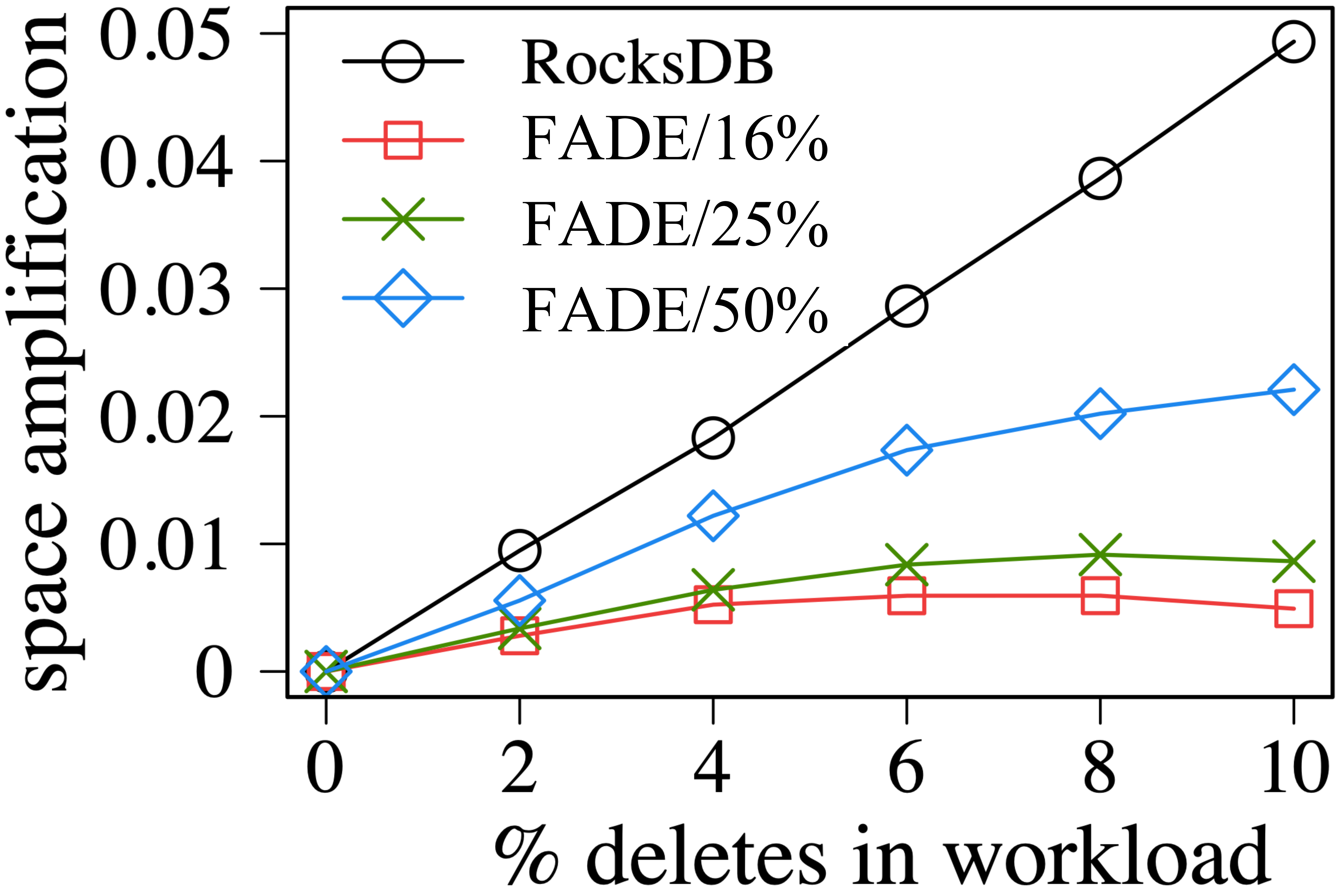
FAst DElete

reduced space amplification

2.1 - 9.8 x

timely delete persistence

within D_{th}



FAst DElete

improved read performance ✓

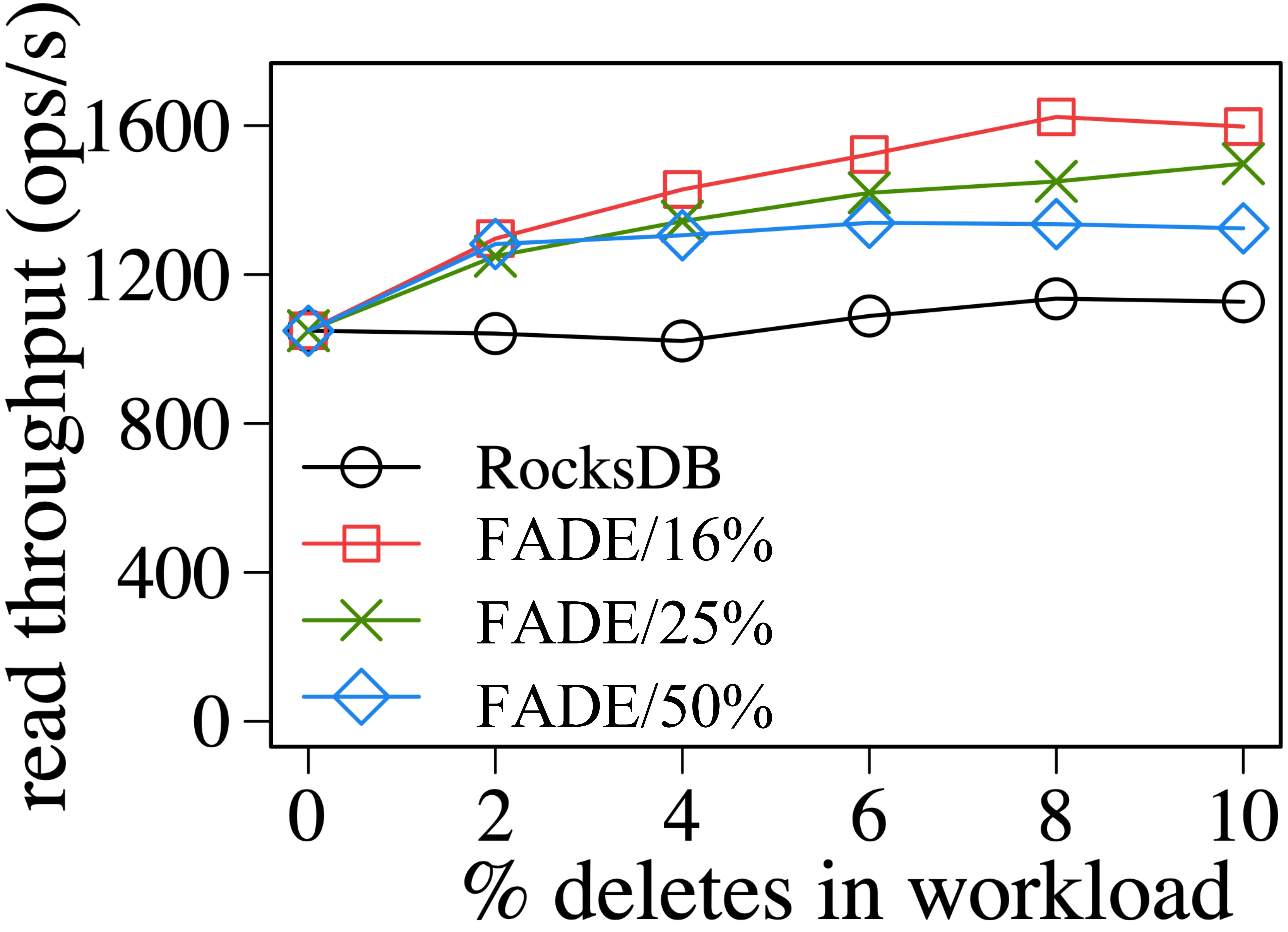
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

within D_{th}



FAst DElete

higher write amplification

4 - 25 %

improved read performance

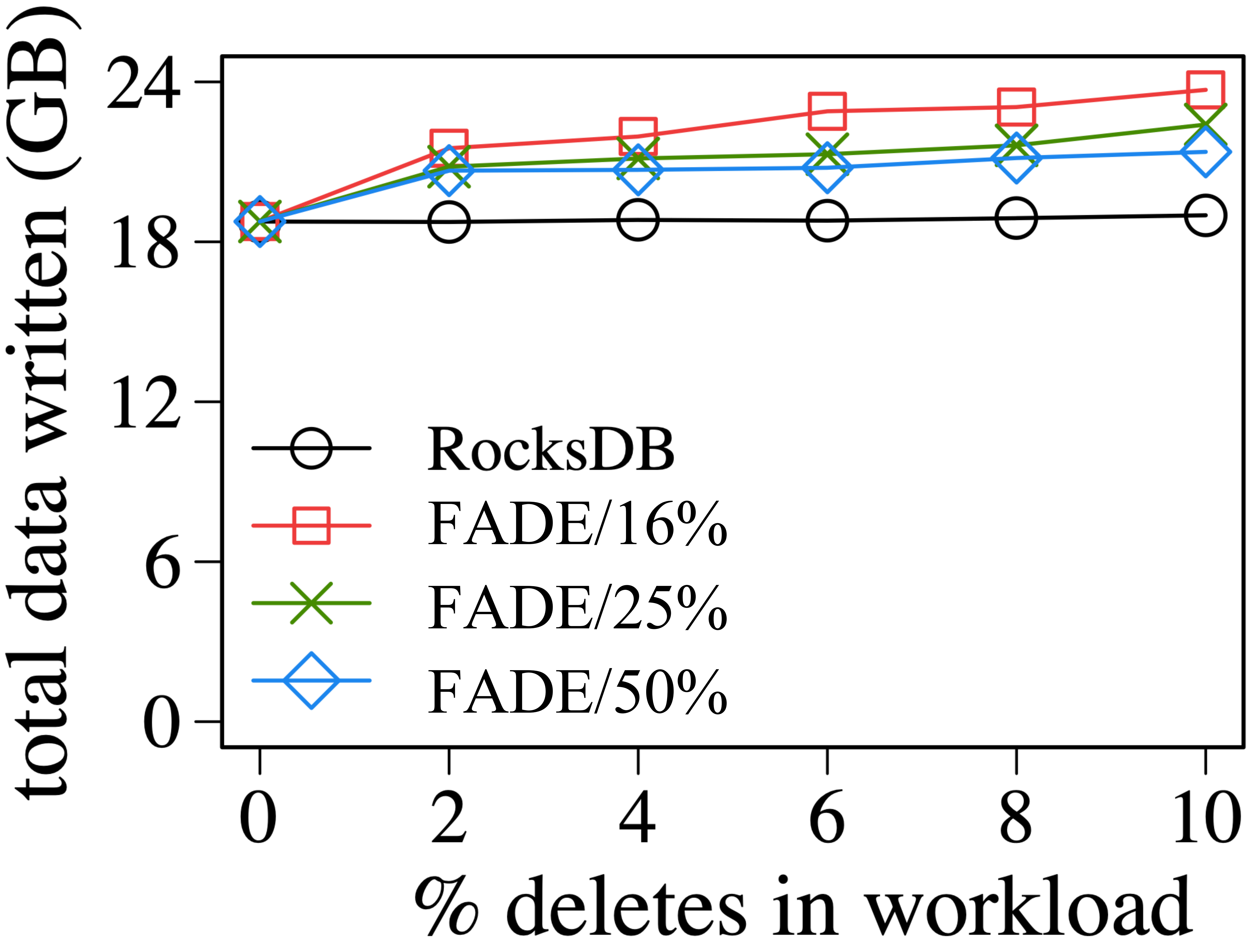
1.2 - 1.4 x

reduced space amplification

2.1 - 9.8 x

timely delete persistence

within D_{th}



FAst DElete

higher write amplification

4 - 25 %

improved read performance

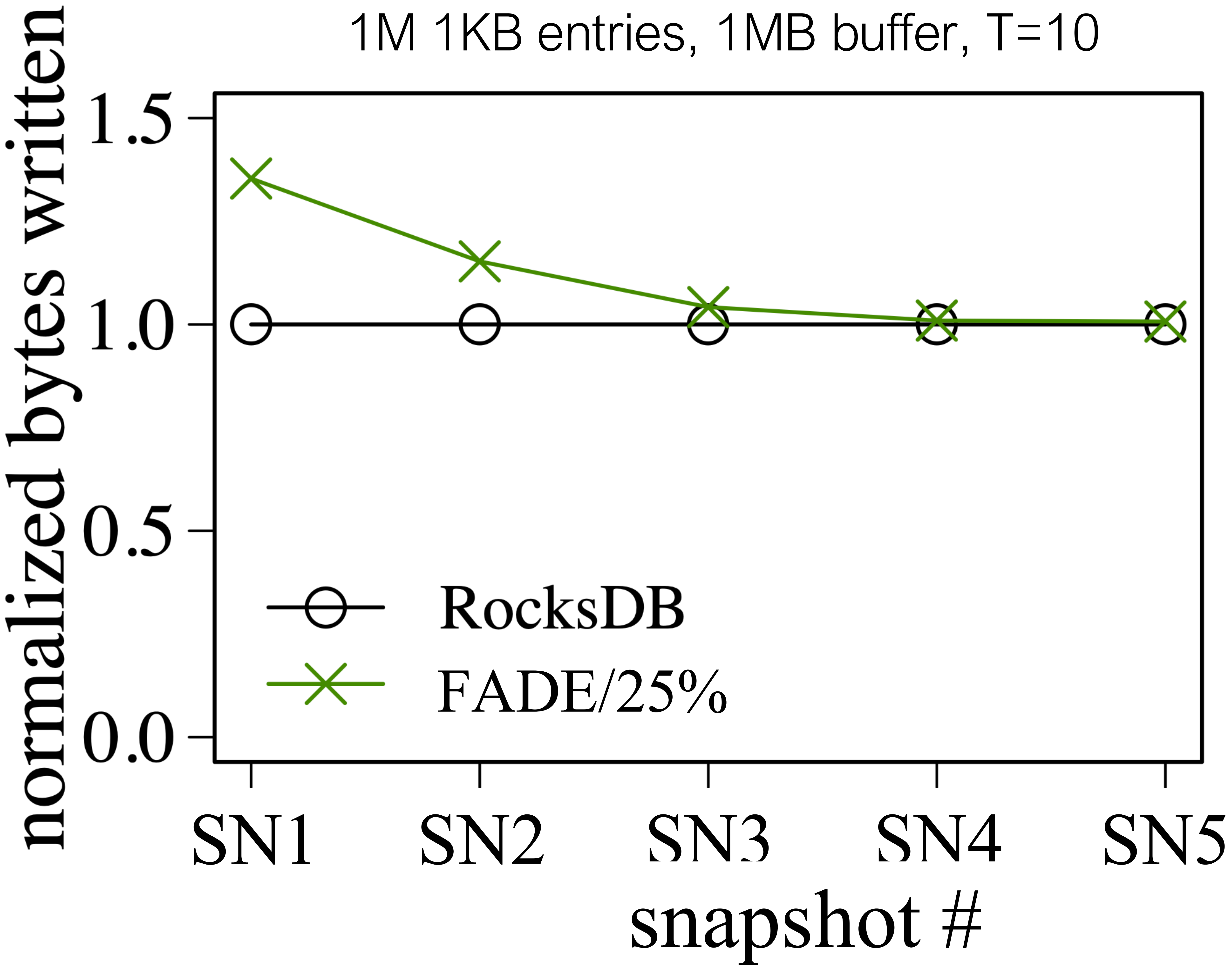
1.2 - 1.4 x

reduced space amplification

2.1 - 9.8 x

timely delete persistence

within D_{th}



FAst DElete

higher write amplification

0.7 %

improved read performance ✓

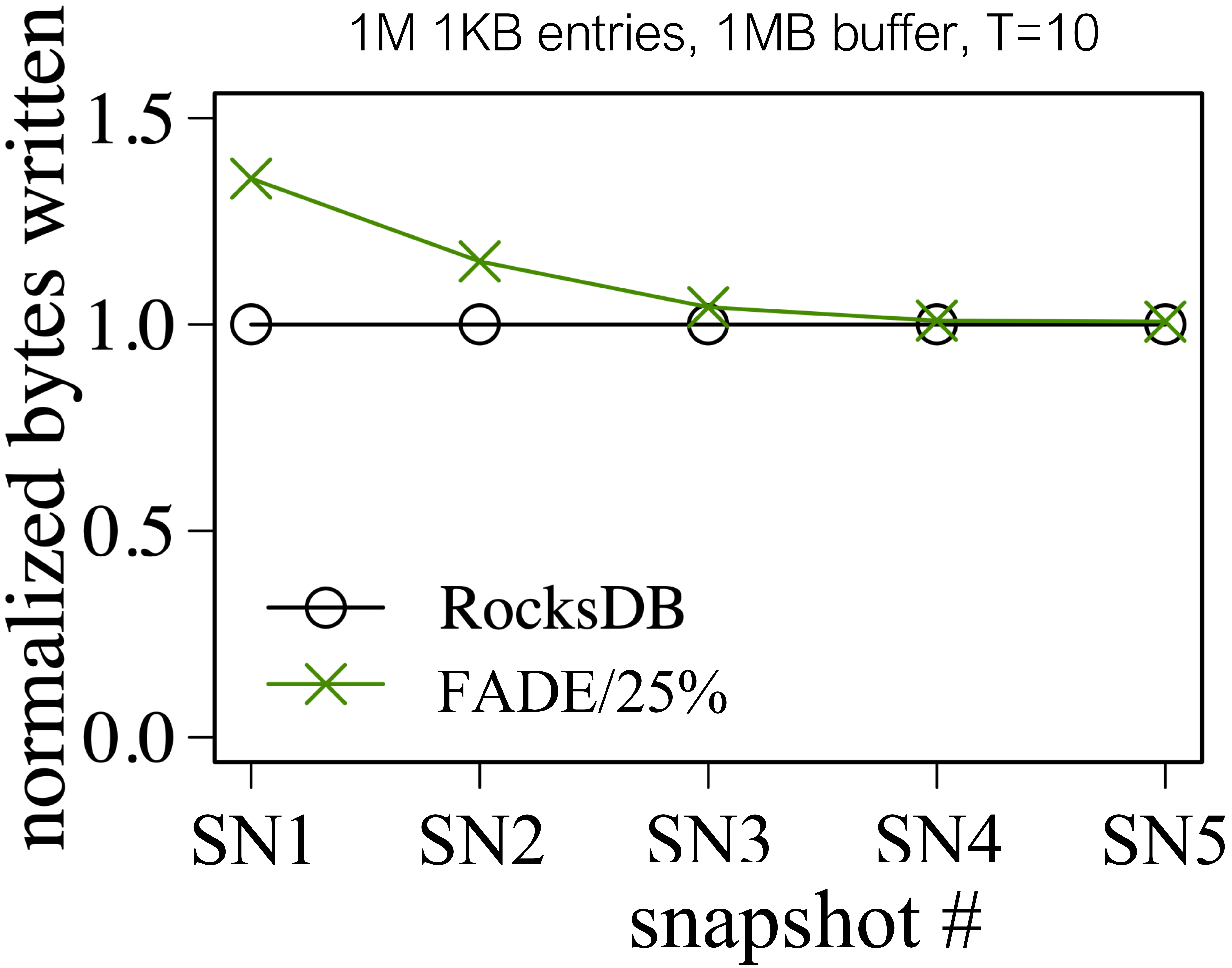
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

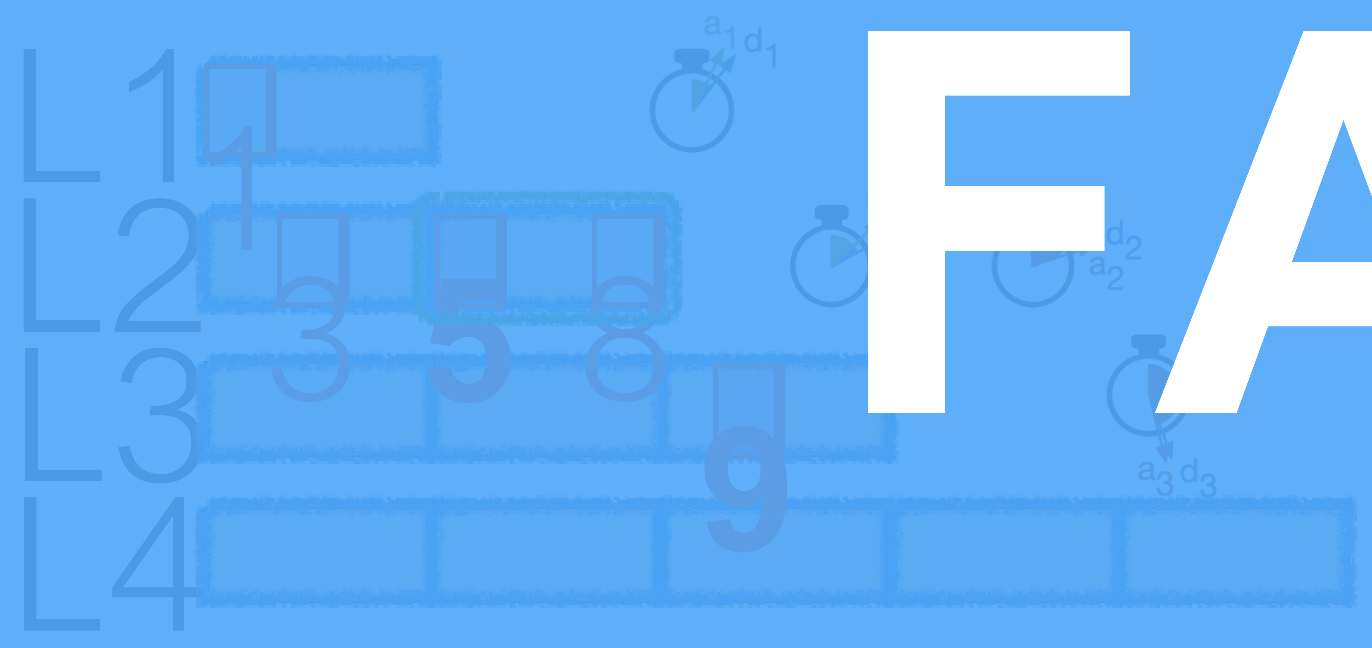
within D_{th}



the solution

FADE

FAst DElete



higher write amplification

reduced space amplification

timely delete persistence

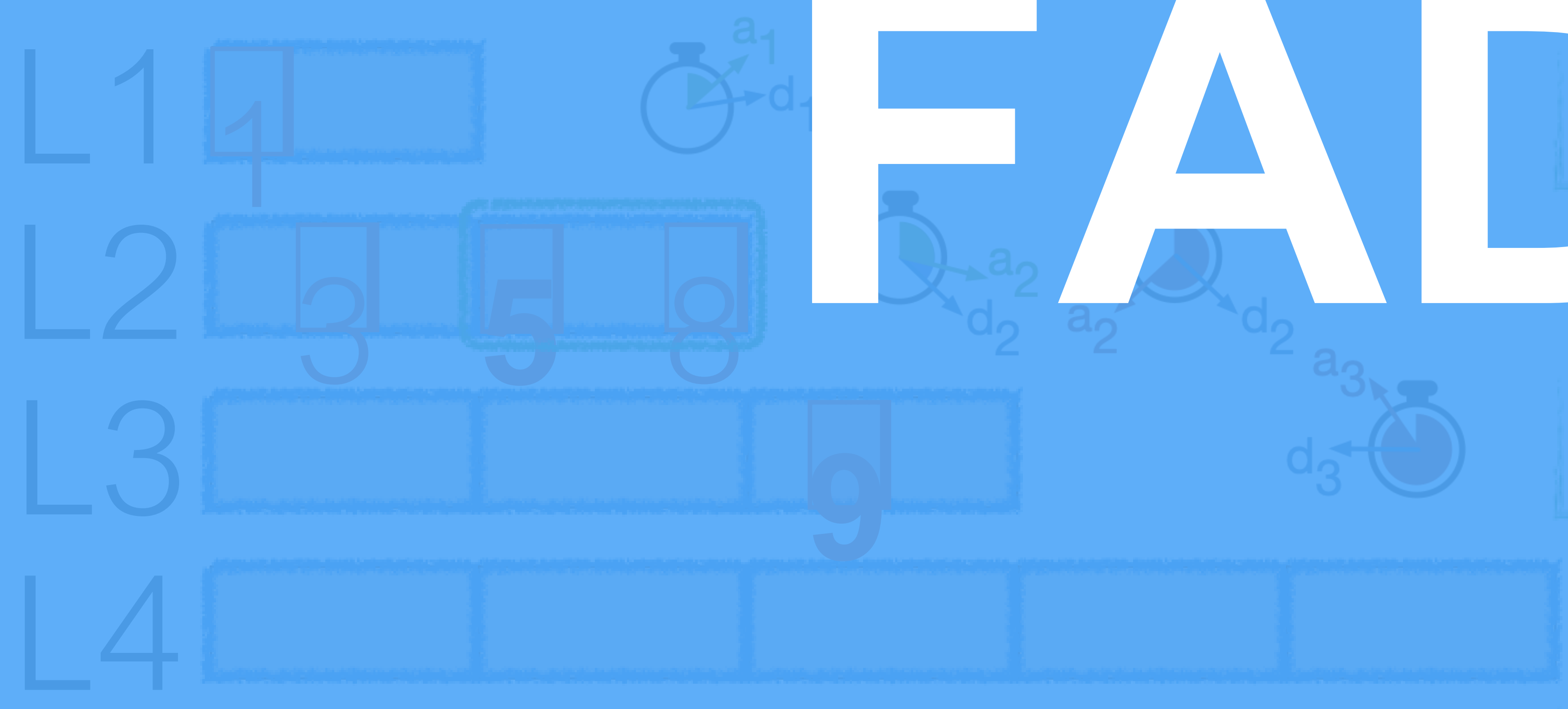
KiWi

latency spikes

the solution

FADE

FAst DElete



higher write amplification

improved read performance

reduced space amplification

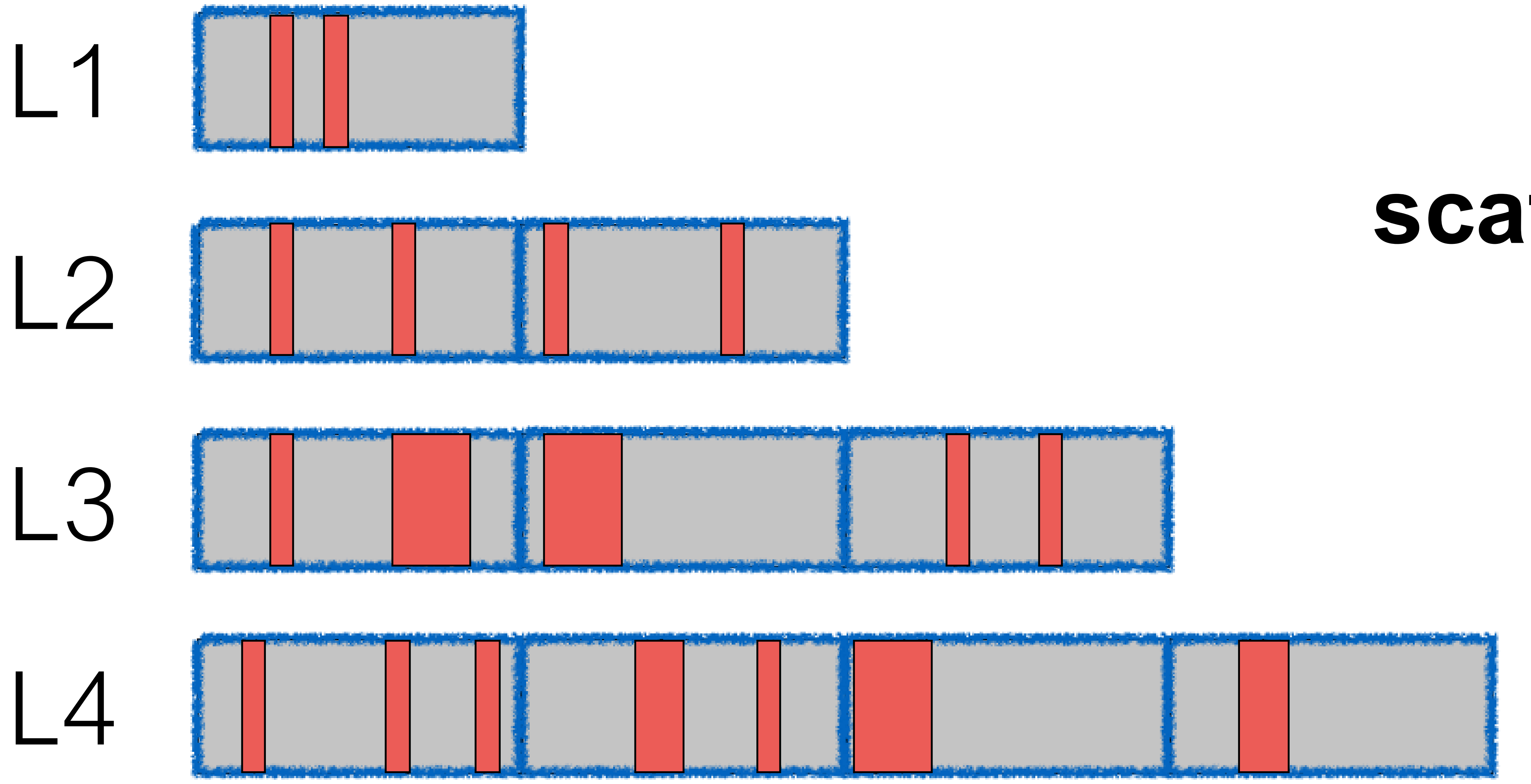
timely delete persistence

KiWi

latency spikes

Key Weaving storage layout

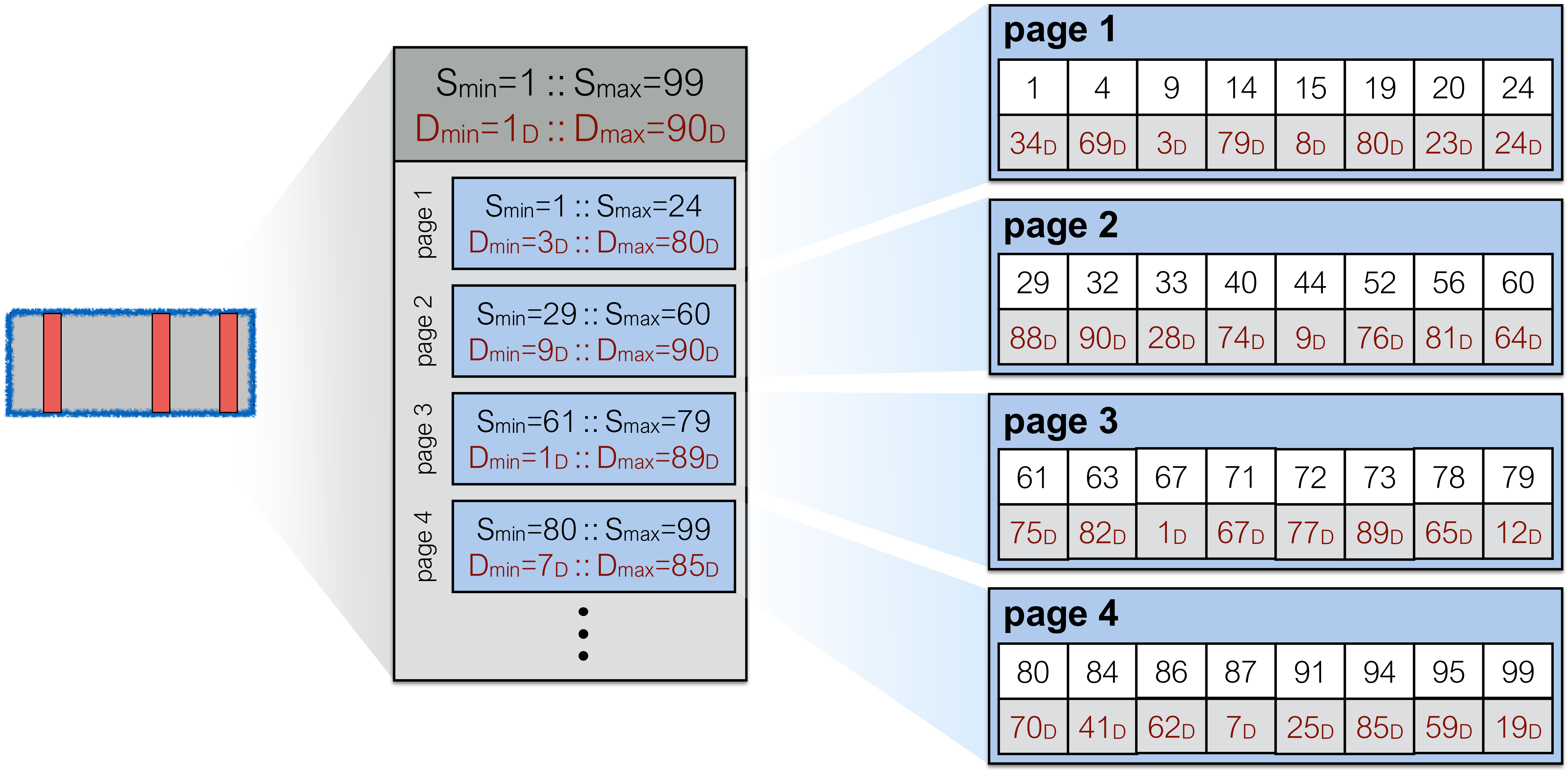
delete all entries older than: **D days**



scattered occurrences

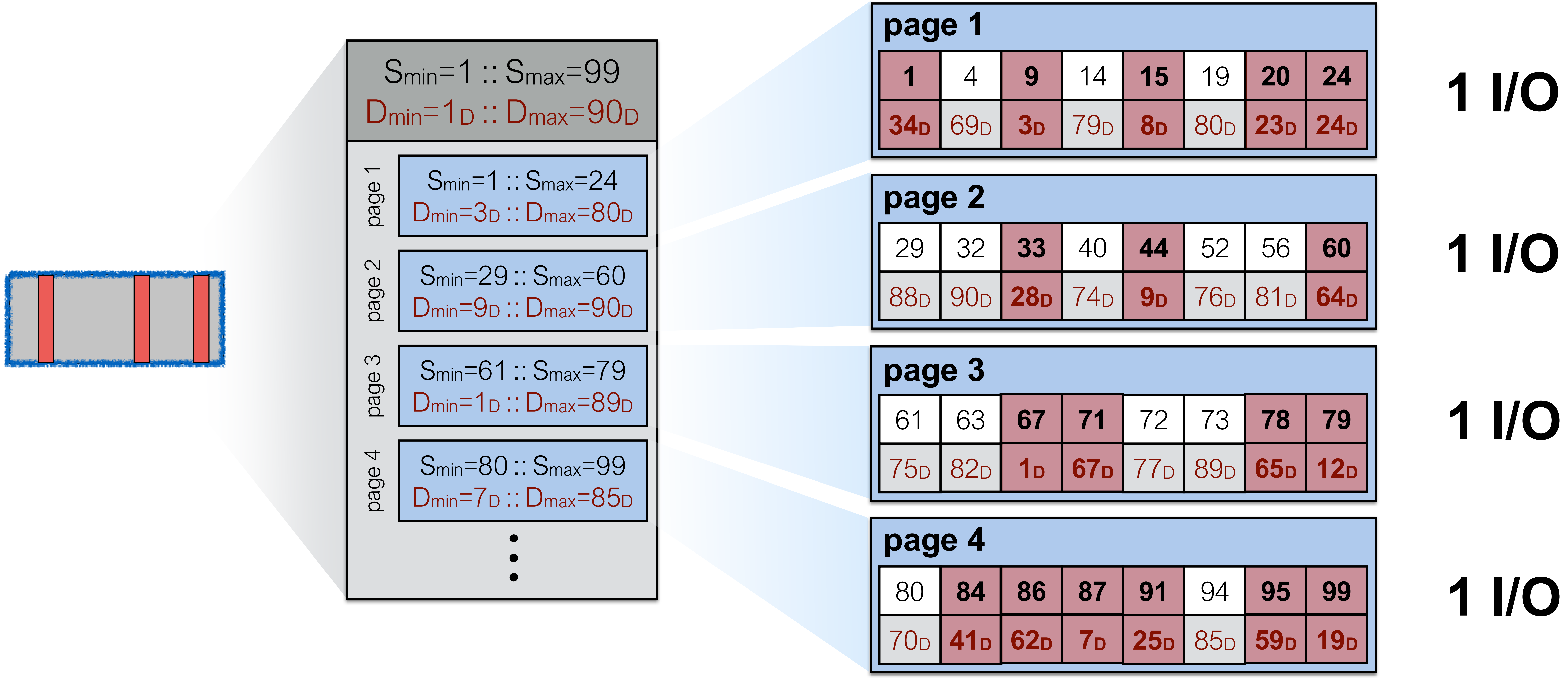
Key Weaving storage layout

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



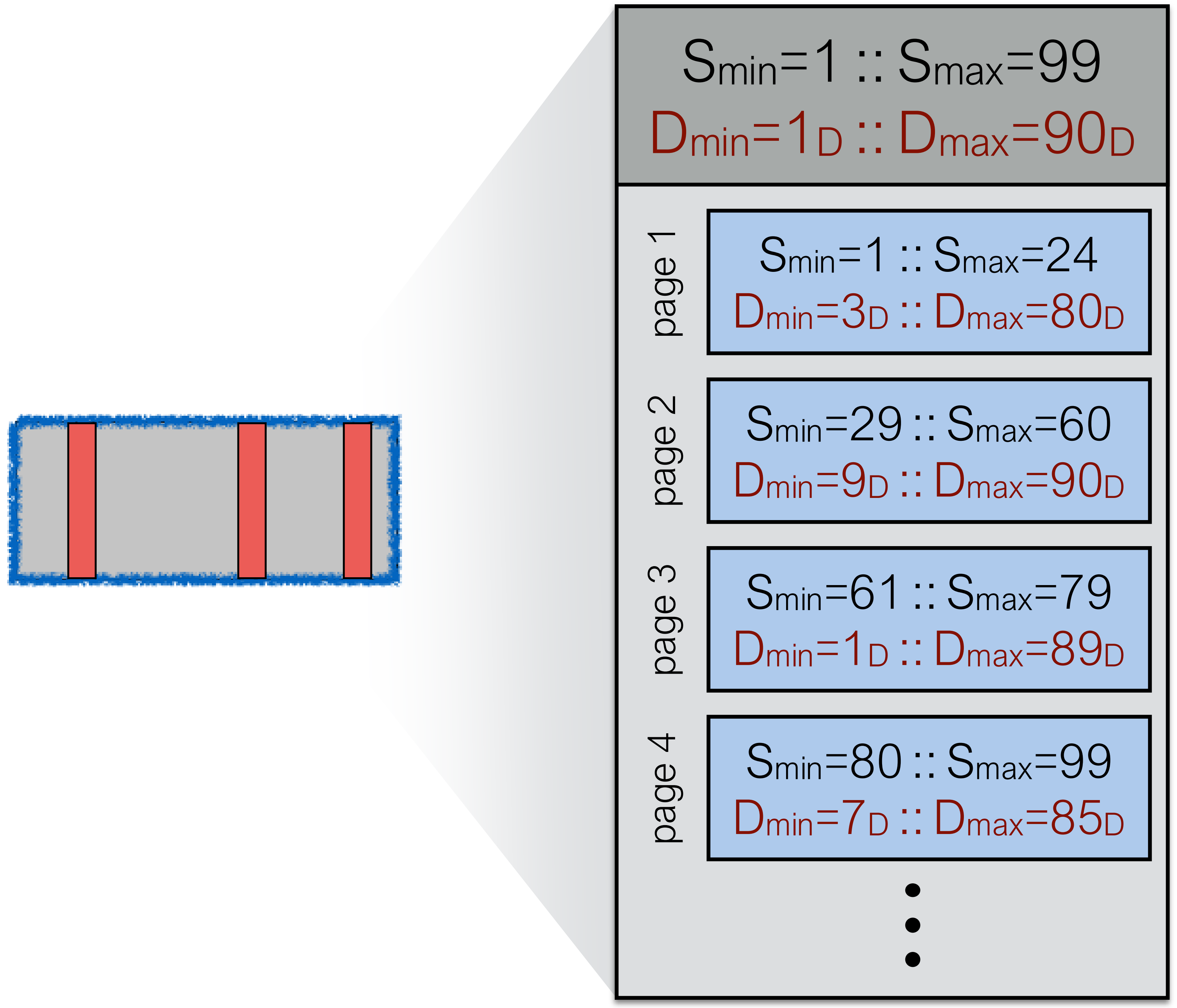
Key Weaving storage layout

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



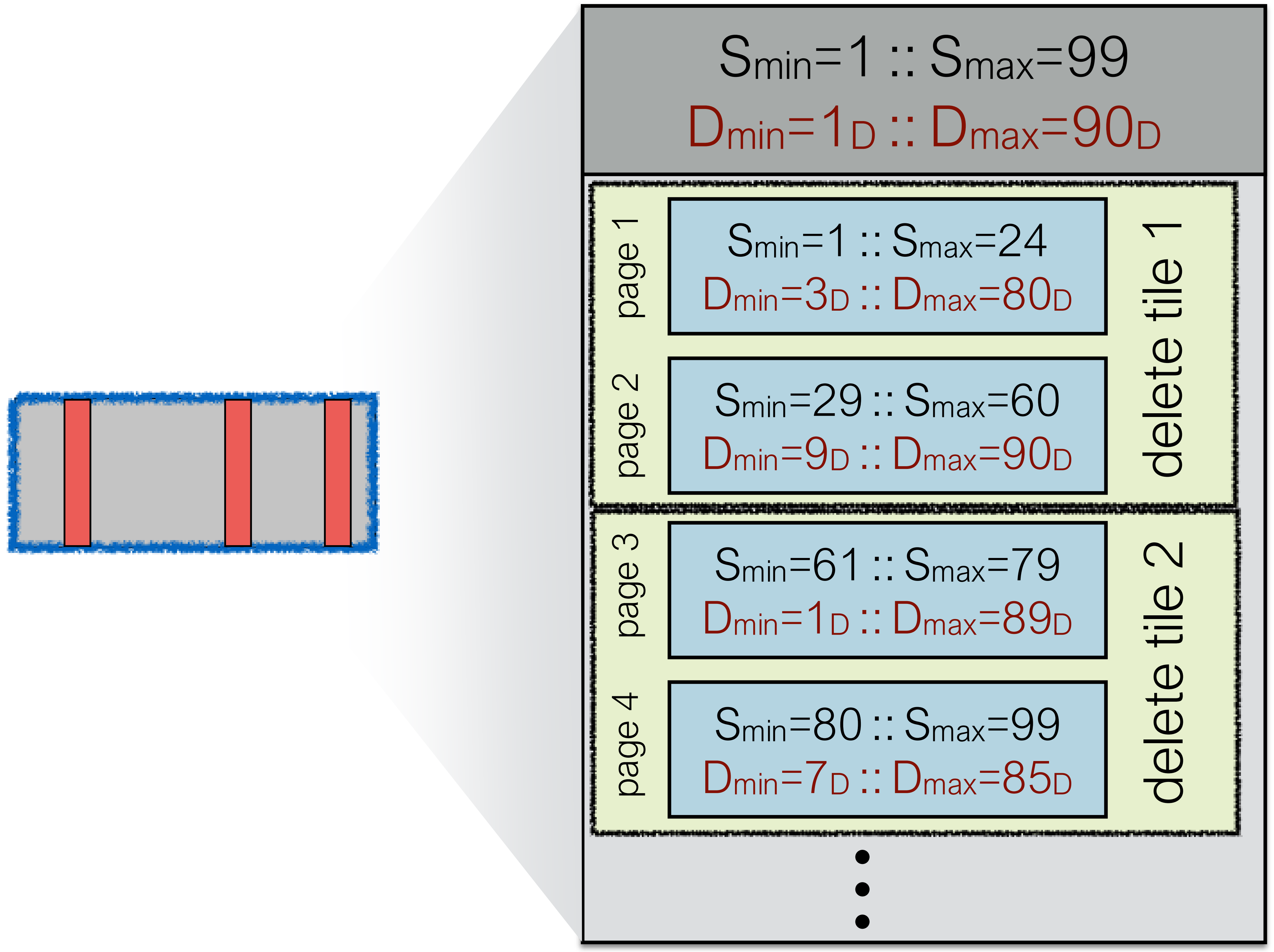
Key Weaving storage layout

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



Key Weaving storage layout

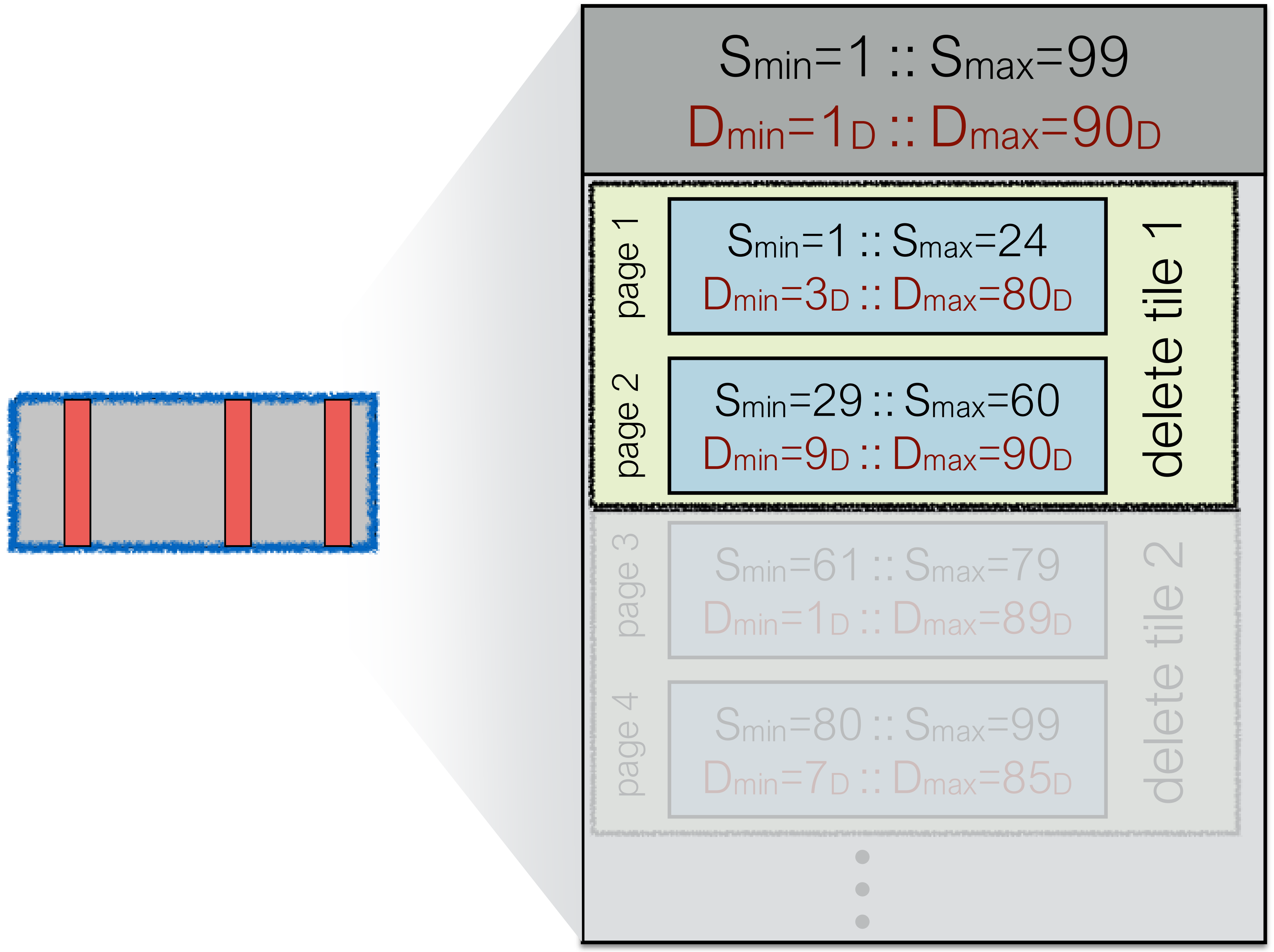
delete all entries with timestamp **<= 65D**



partitioned on S

Key Weaving storage layout

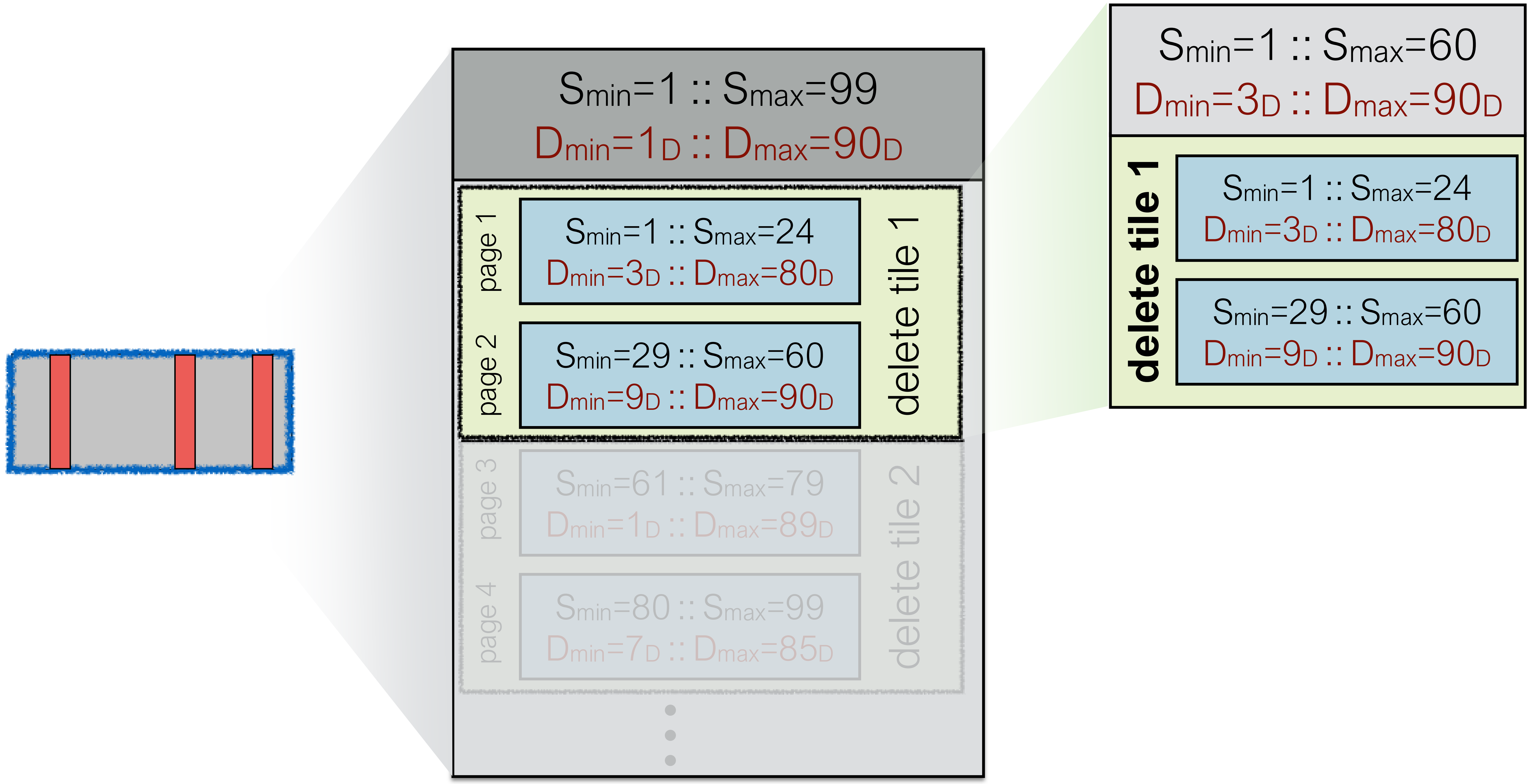
delete all entries with timestamp **<= 65D**



partitioned on S

Key Weaving storage layout

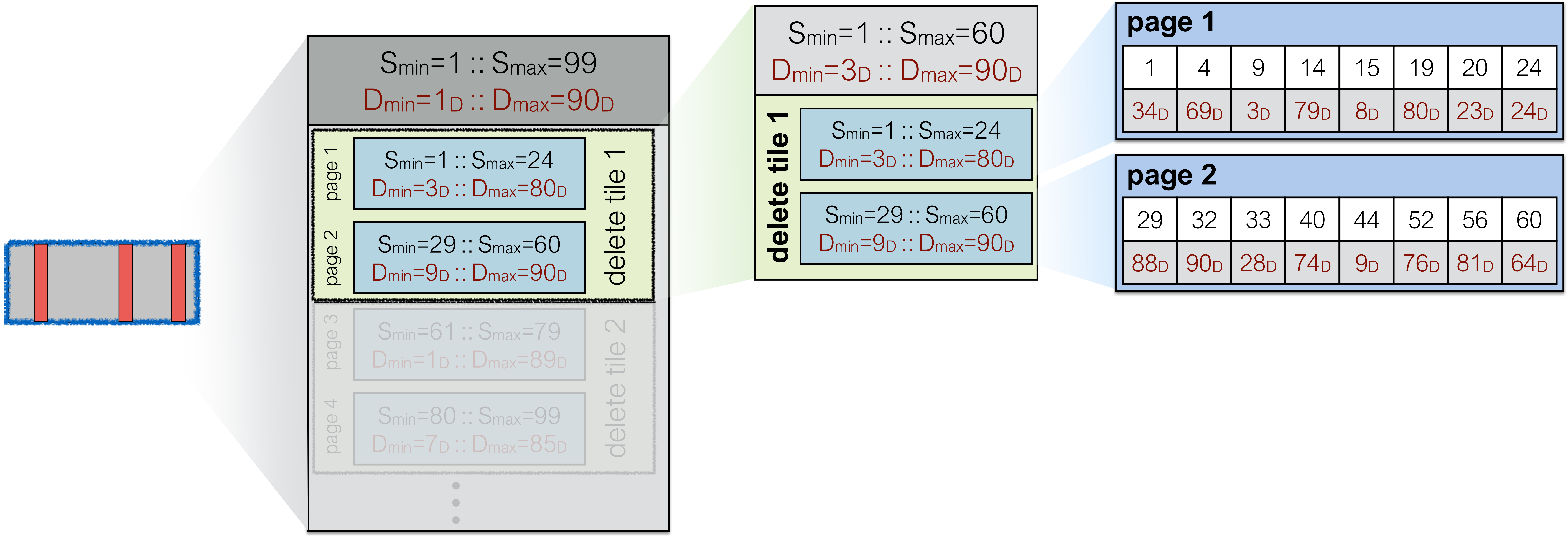
delete all entries with timestamp **<= 65D**



partitioned on S

Key Weaving storage layout

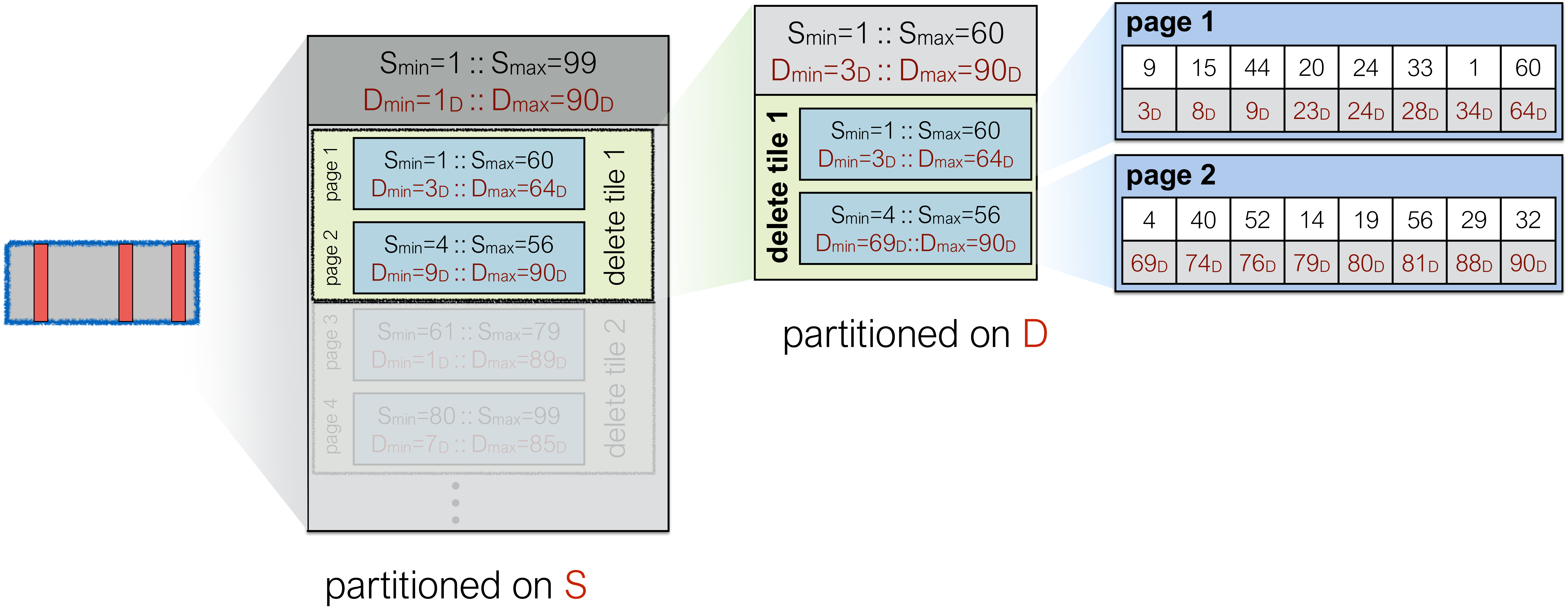
delete all entries with timestamp **<= 65D**



partitioned on S

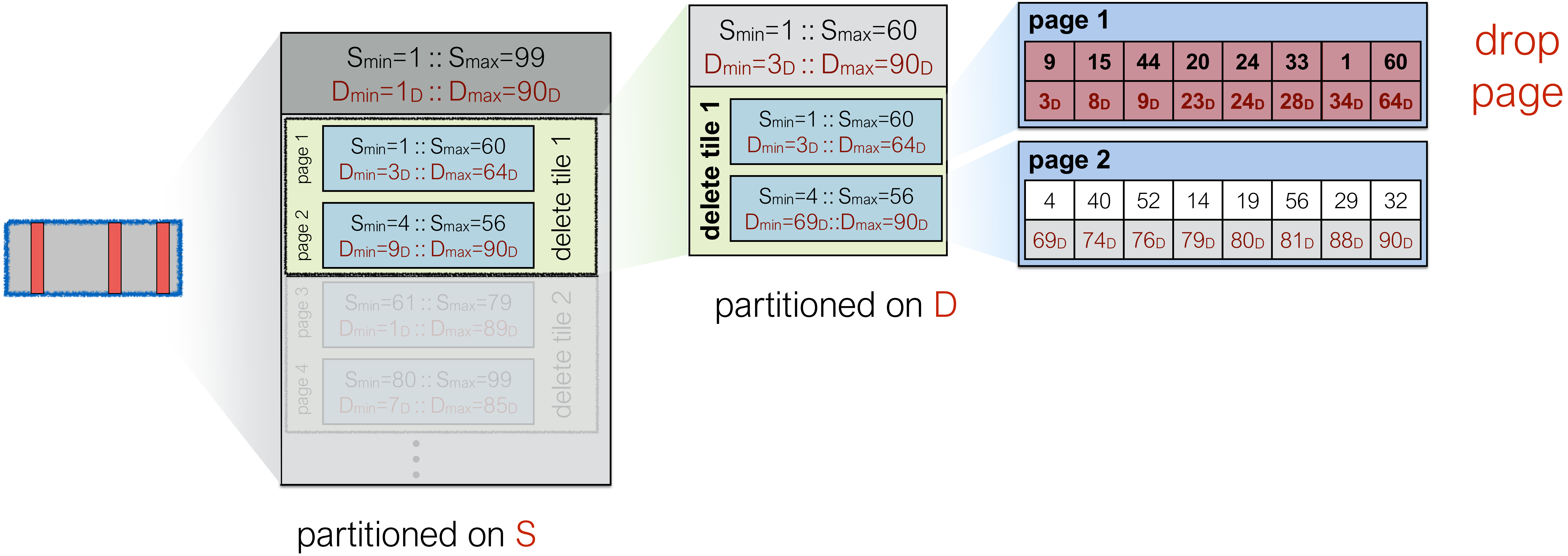
Key Weaving storage layout

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



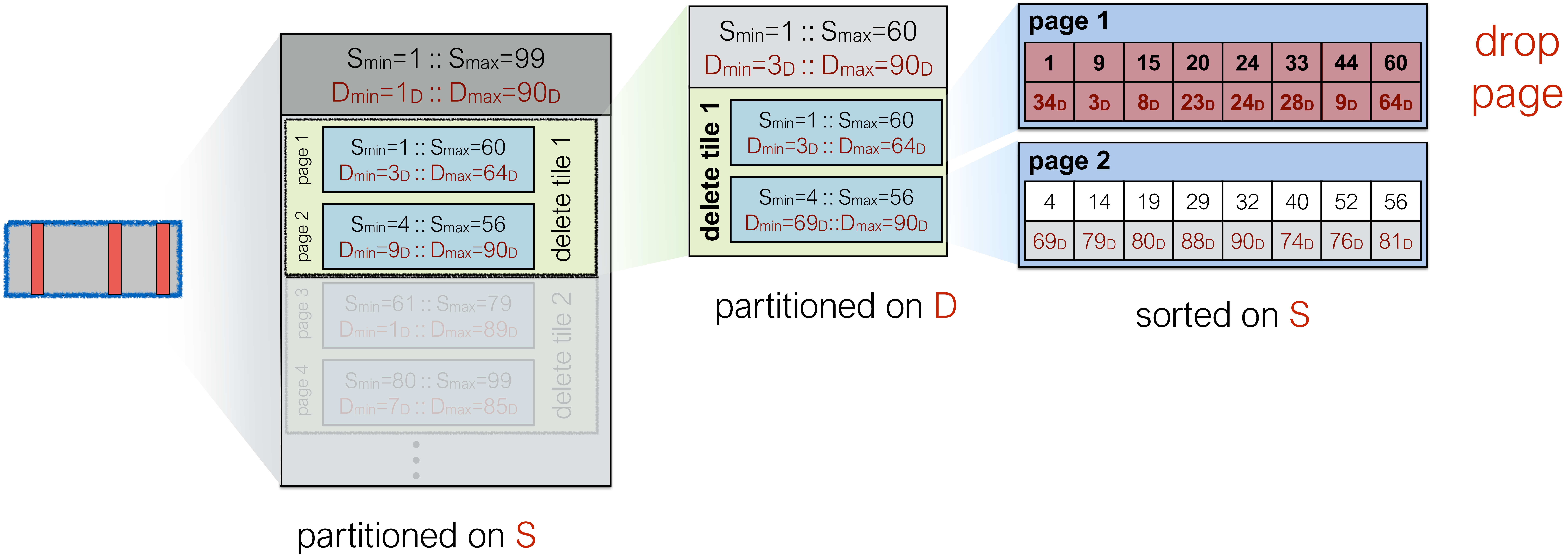
Key Weaving storage layout

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



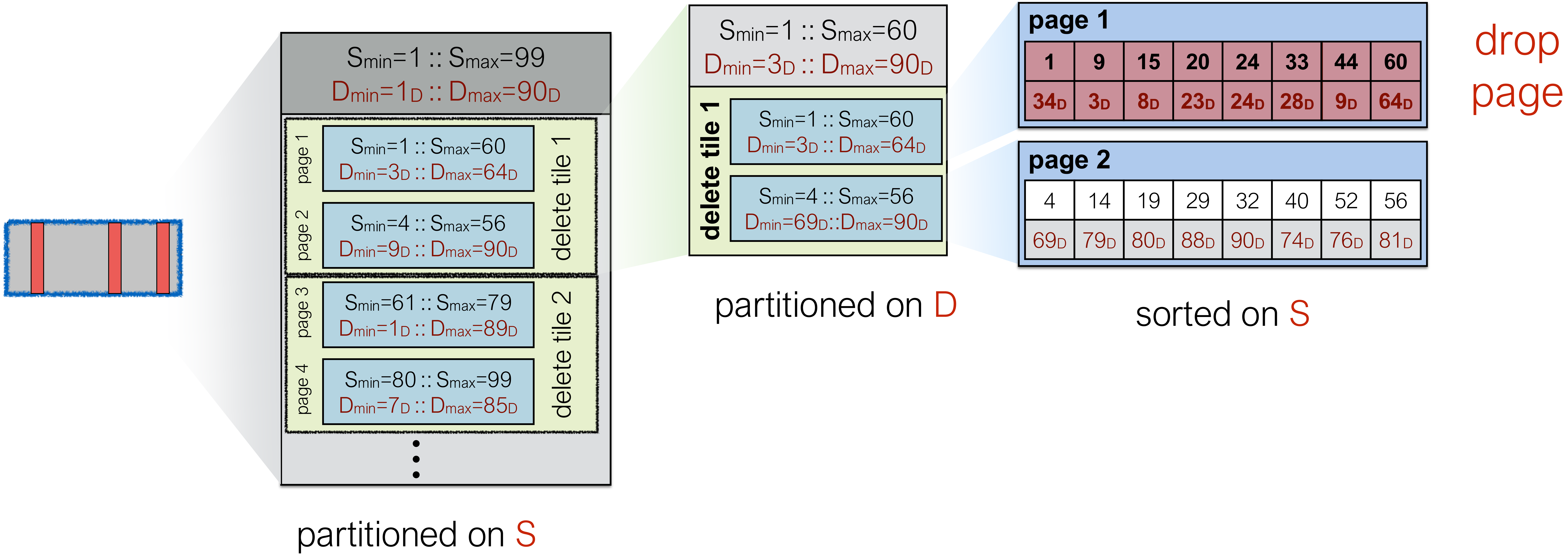
Key Weaving storage layout

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



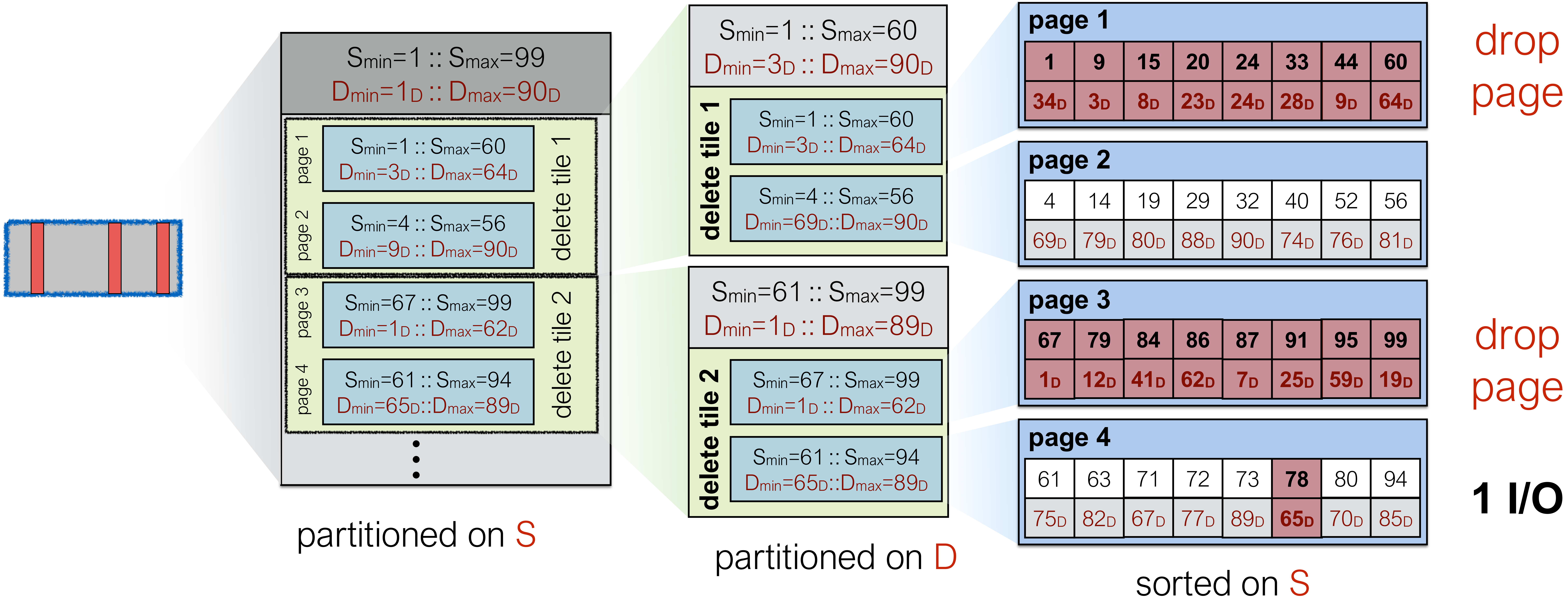
Key Weaving storage layout

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

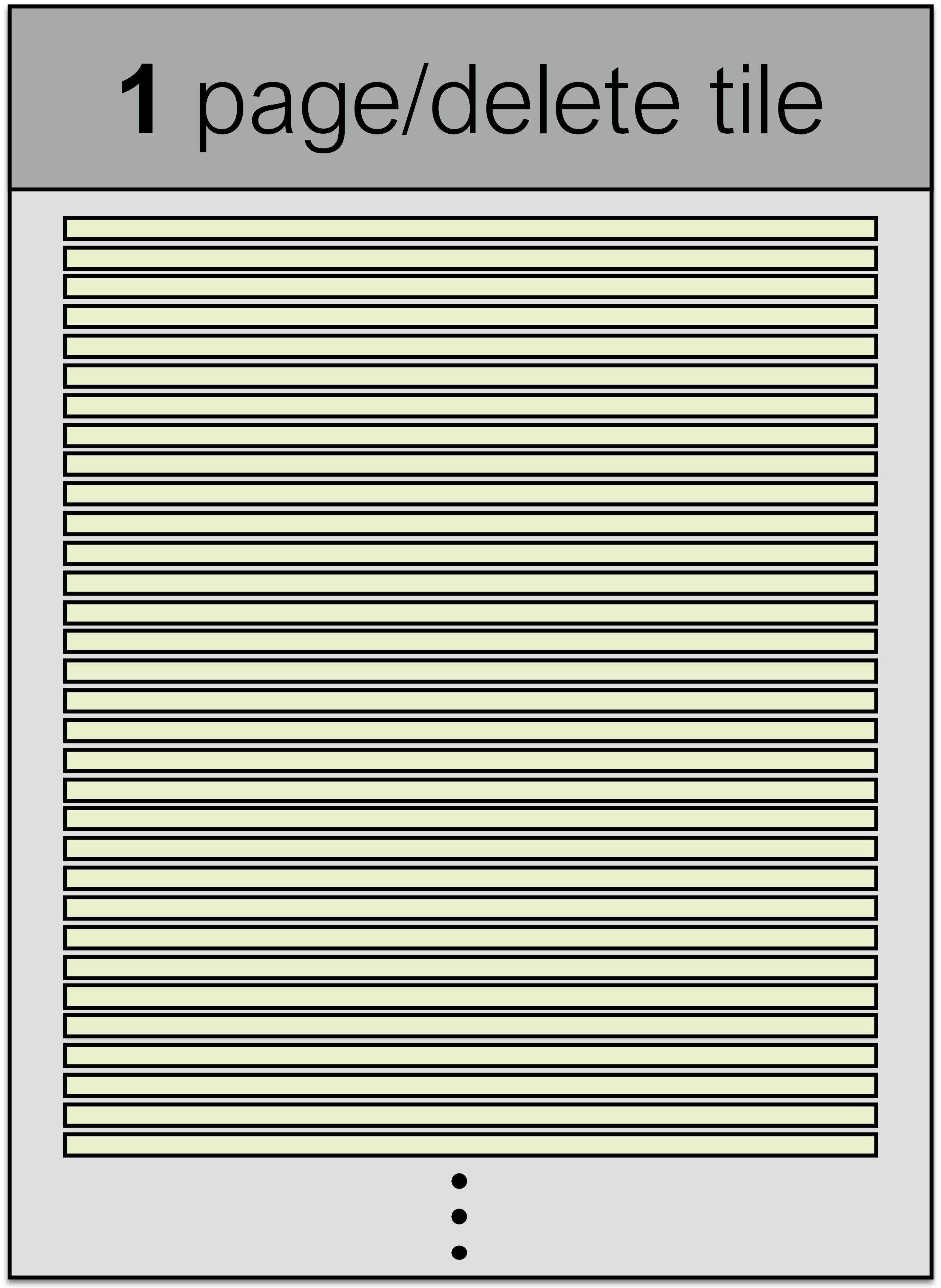


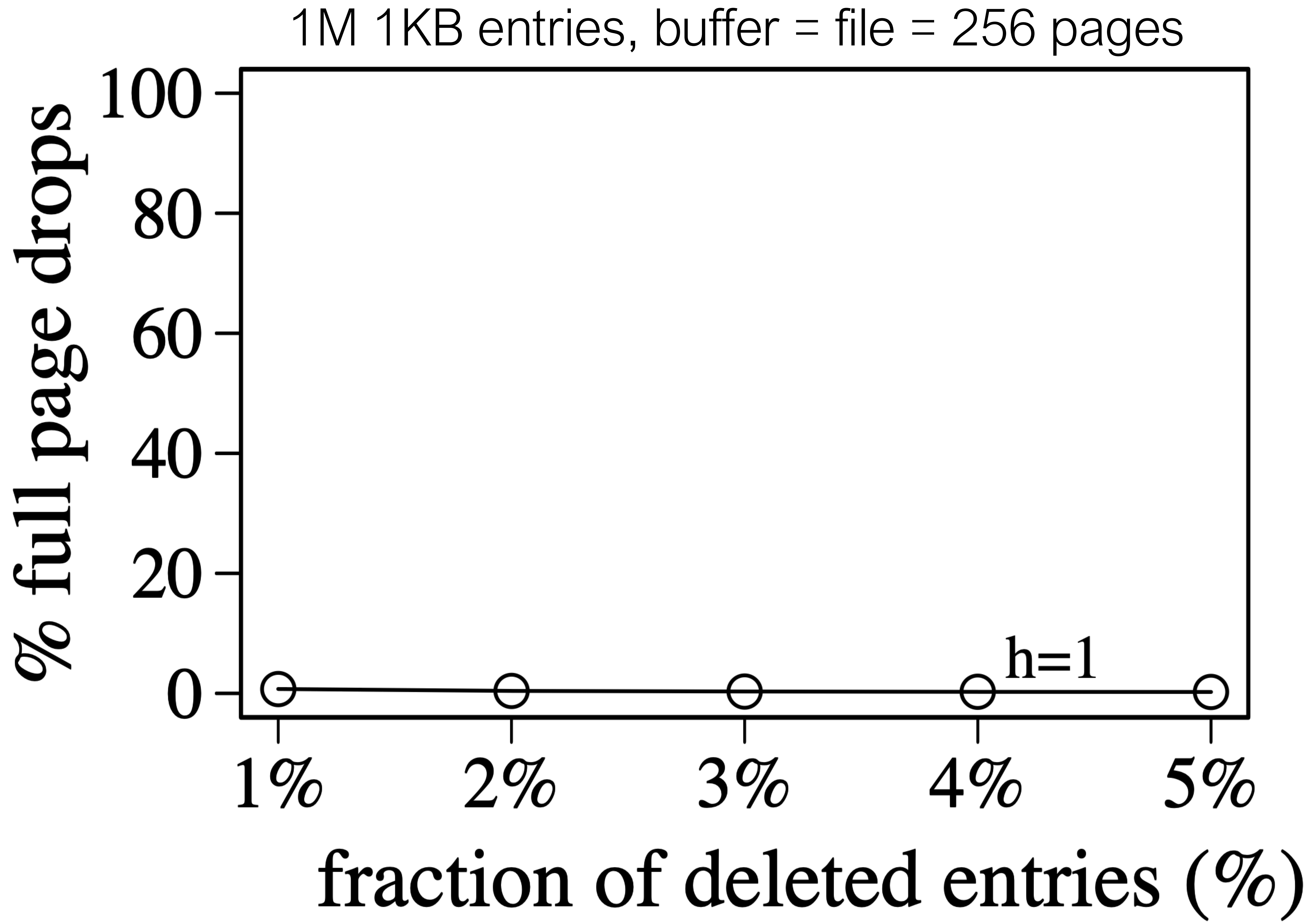
Key Weaving storage layout

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

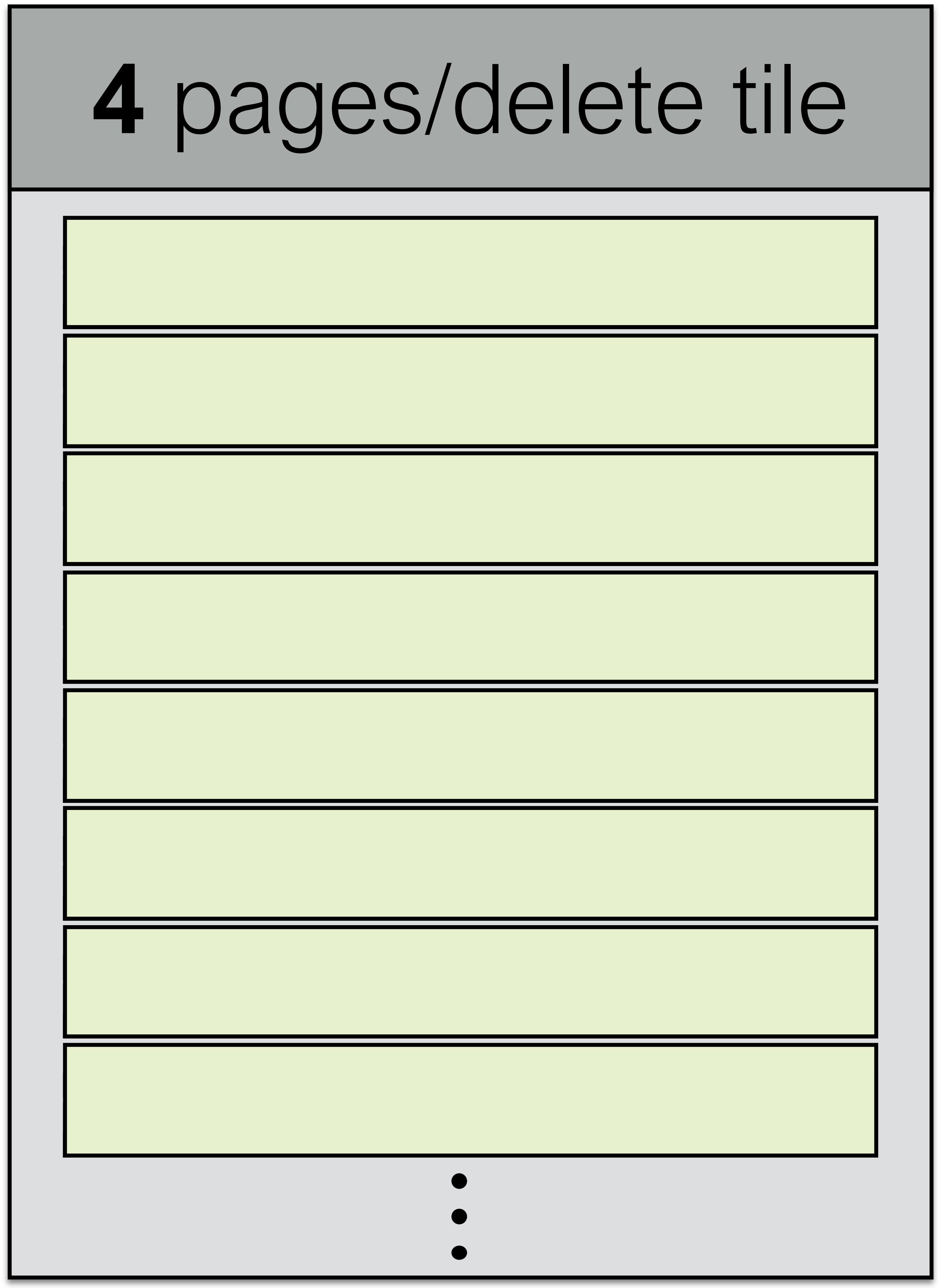


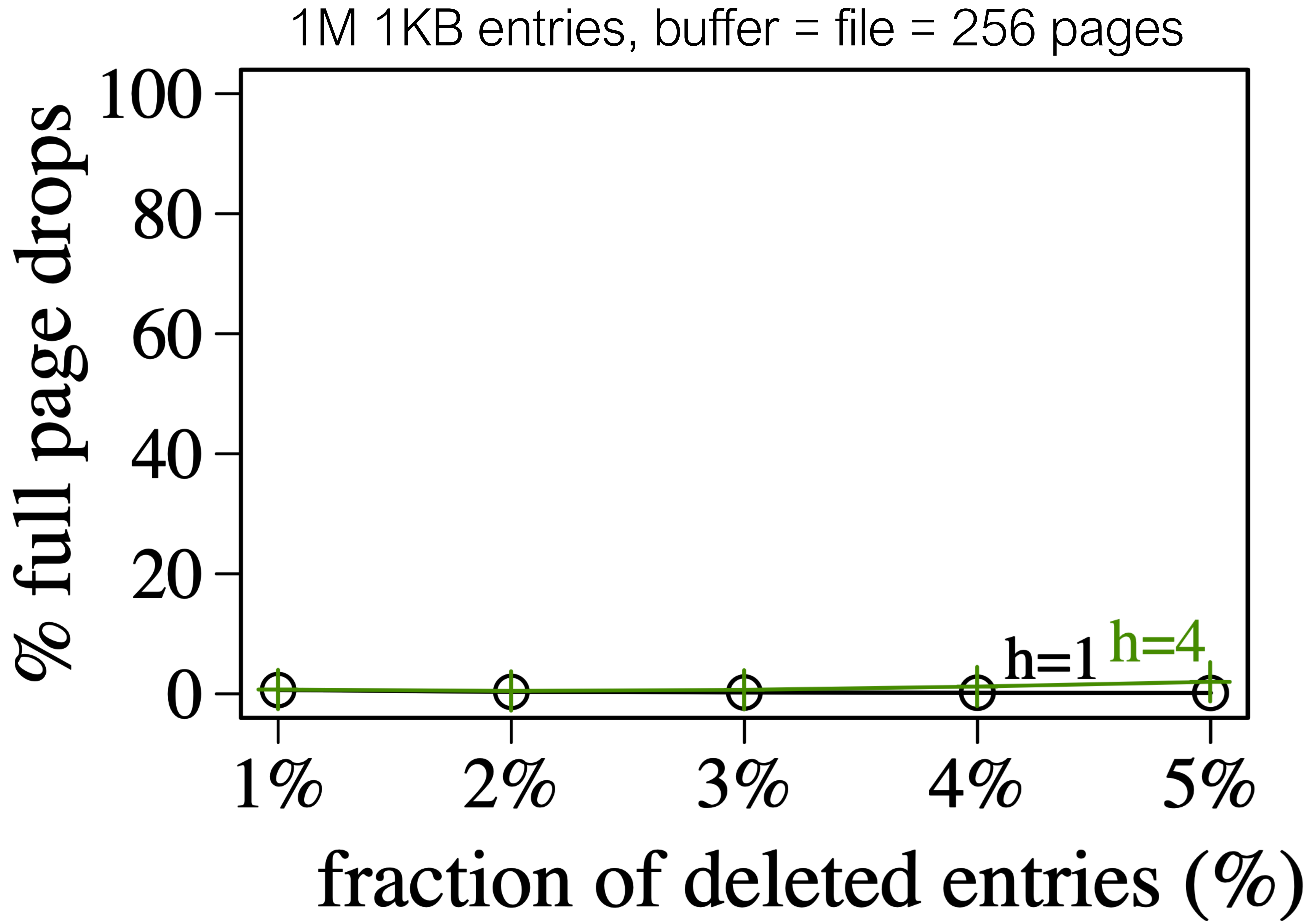
Key Weaving storage layout



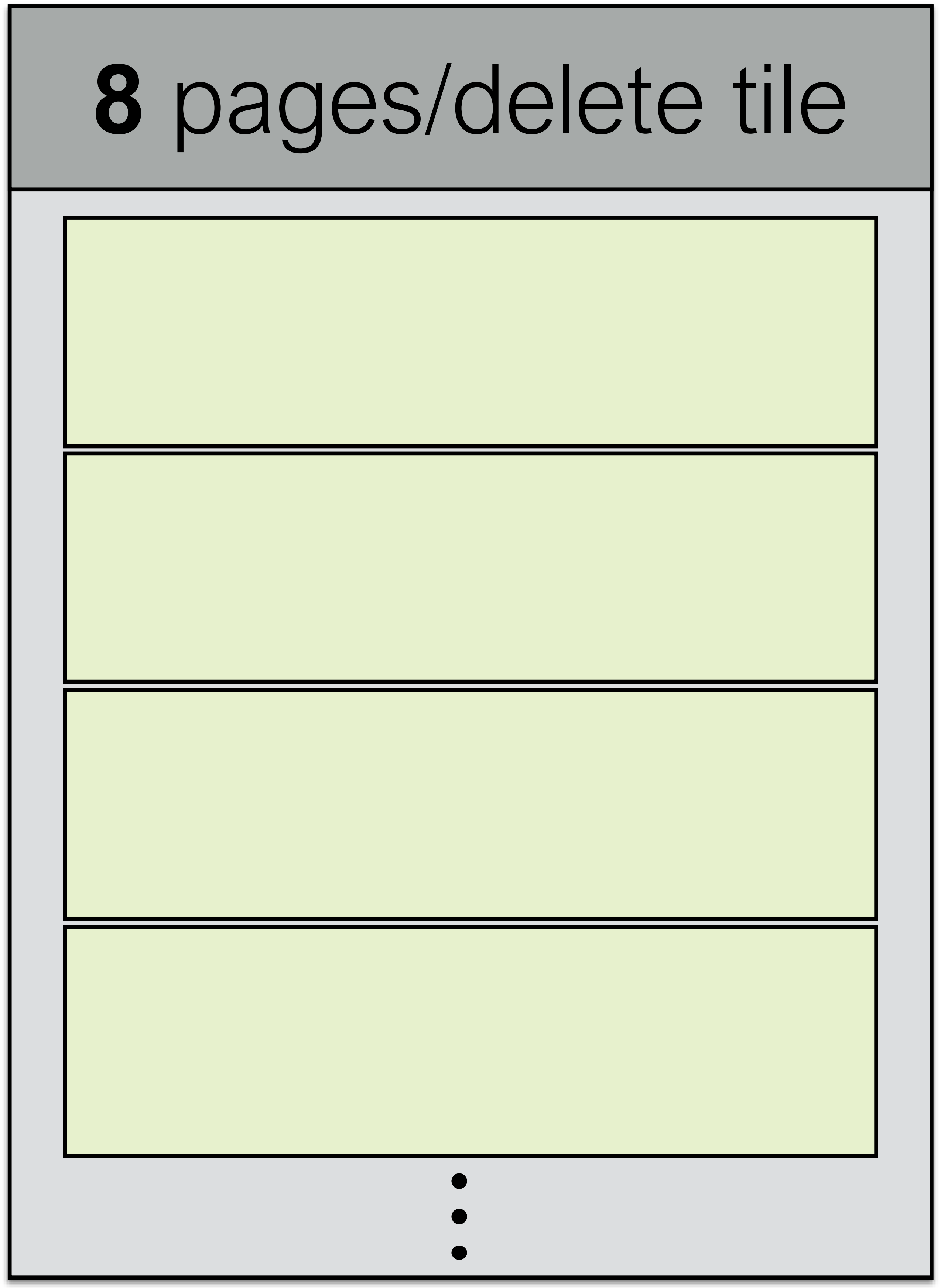


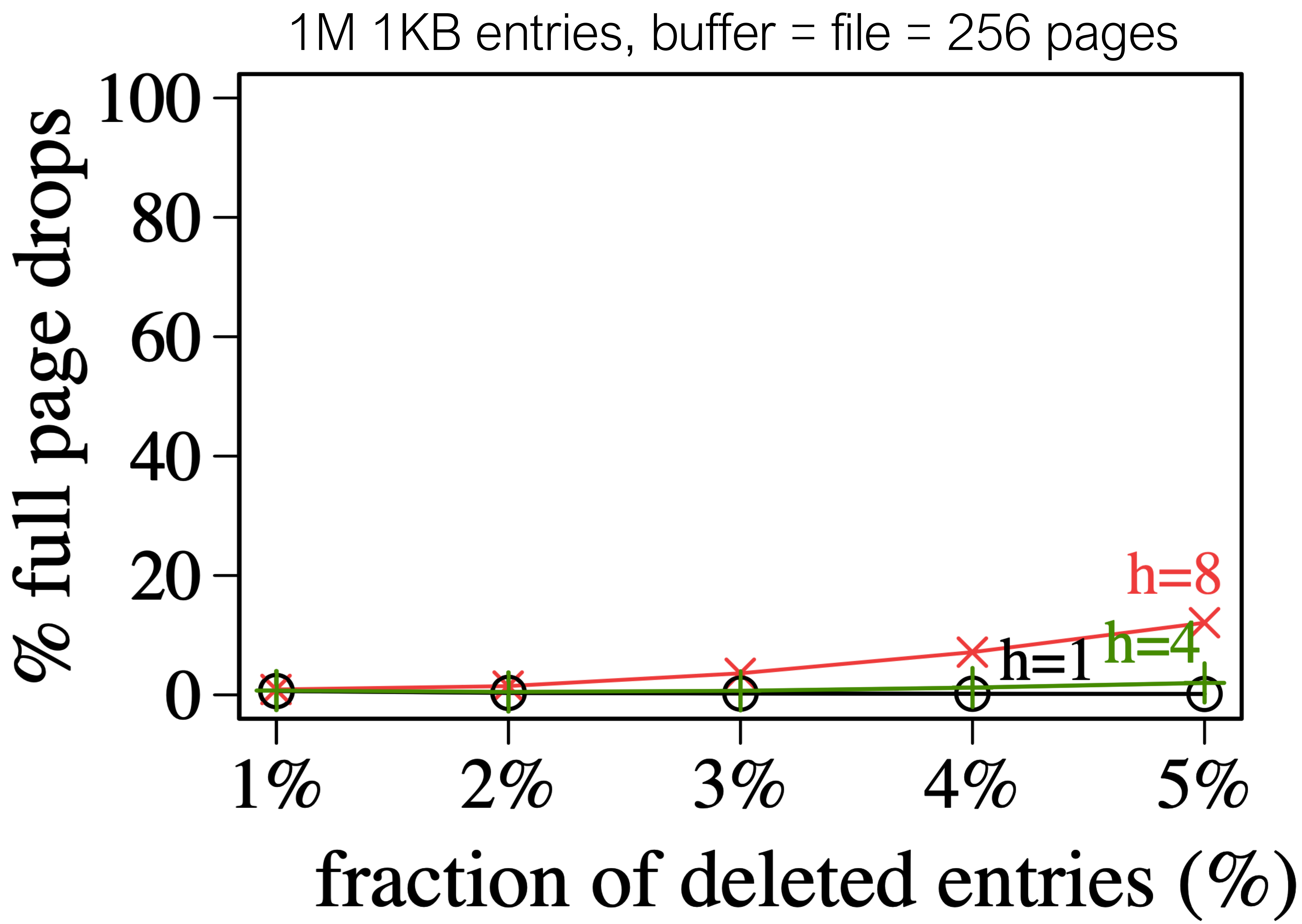
Key Weaving storage layout



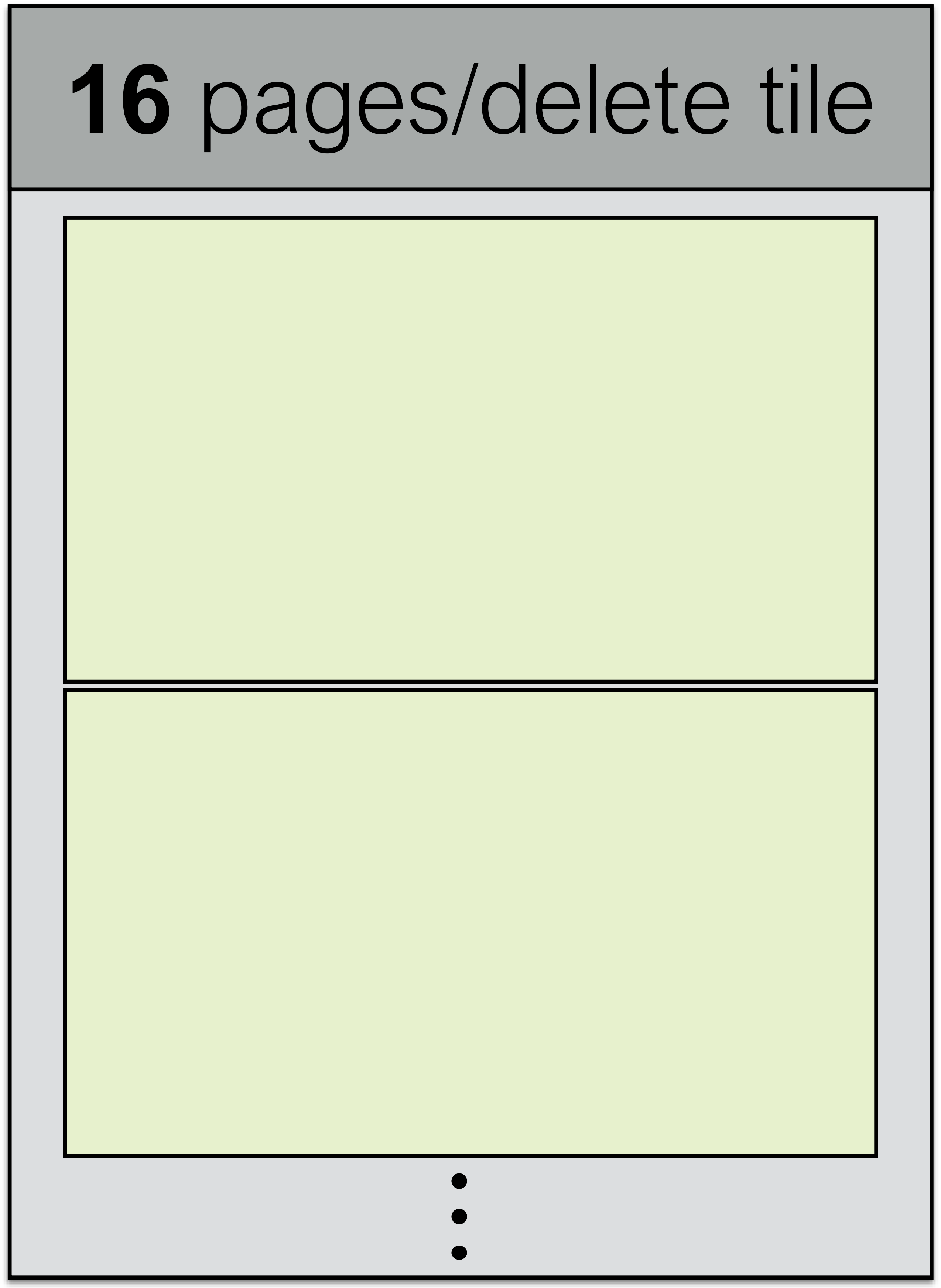


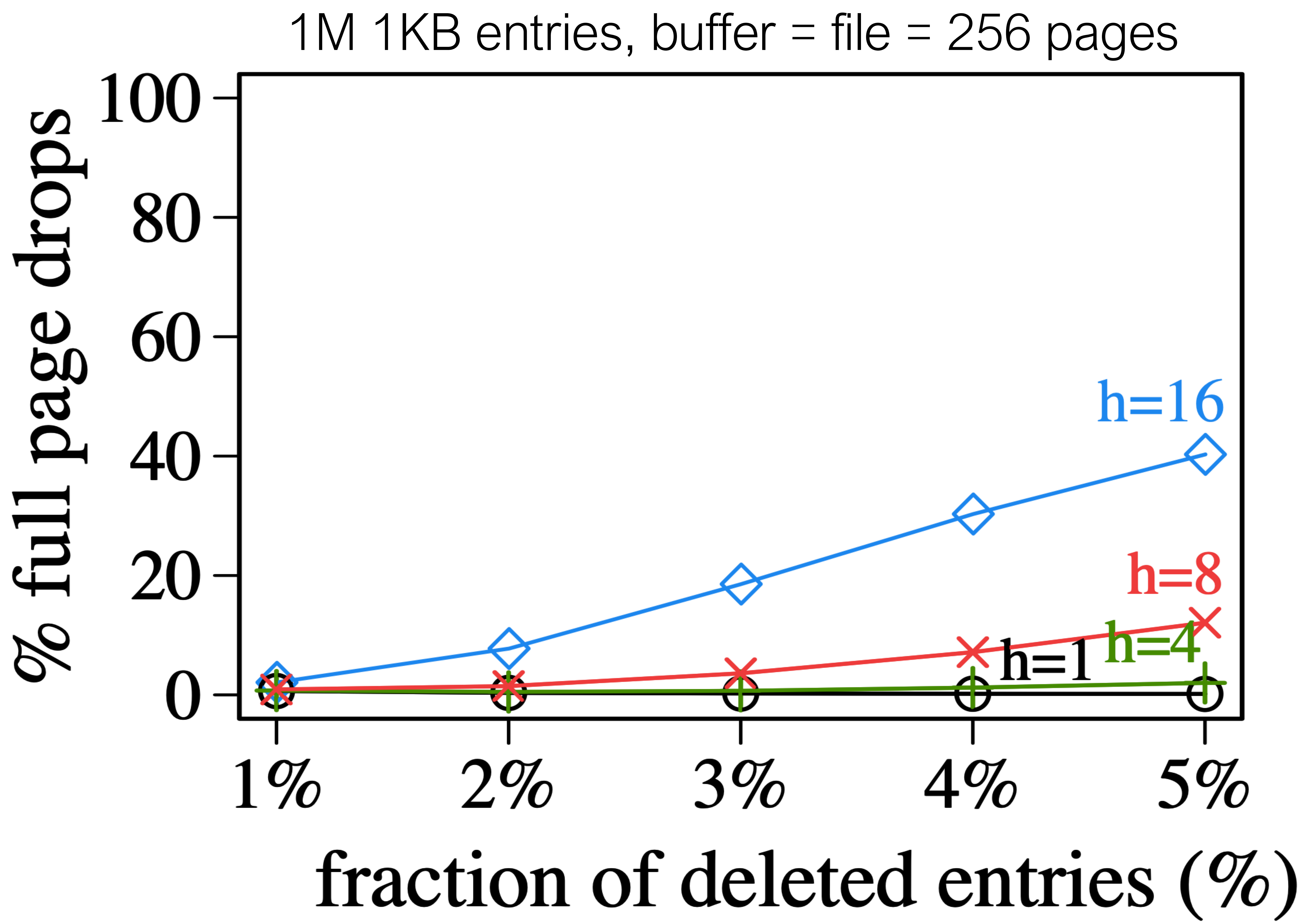
Key Weaving storage layout



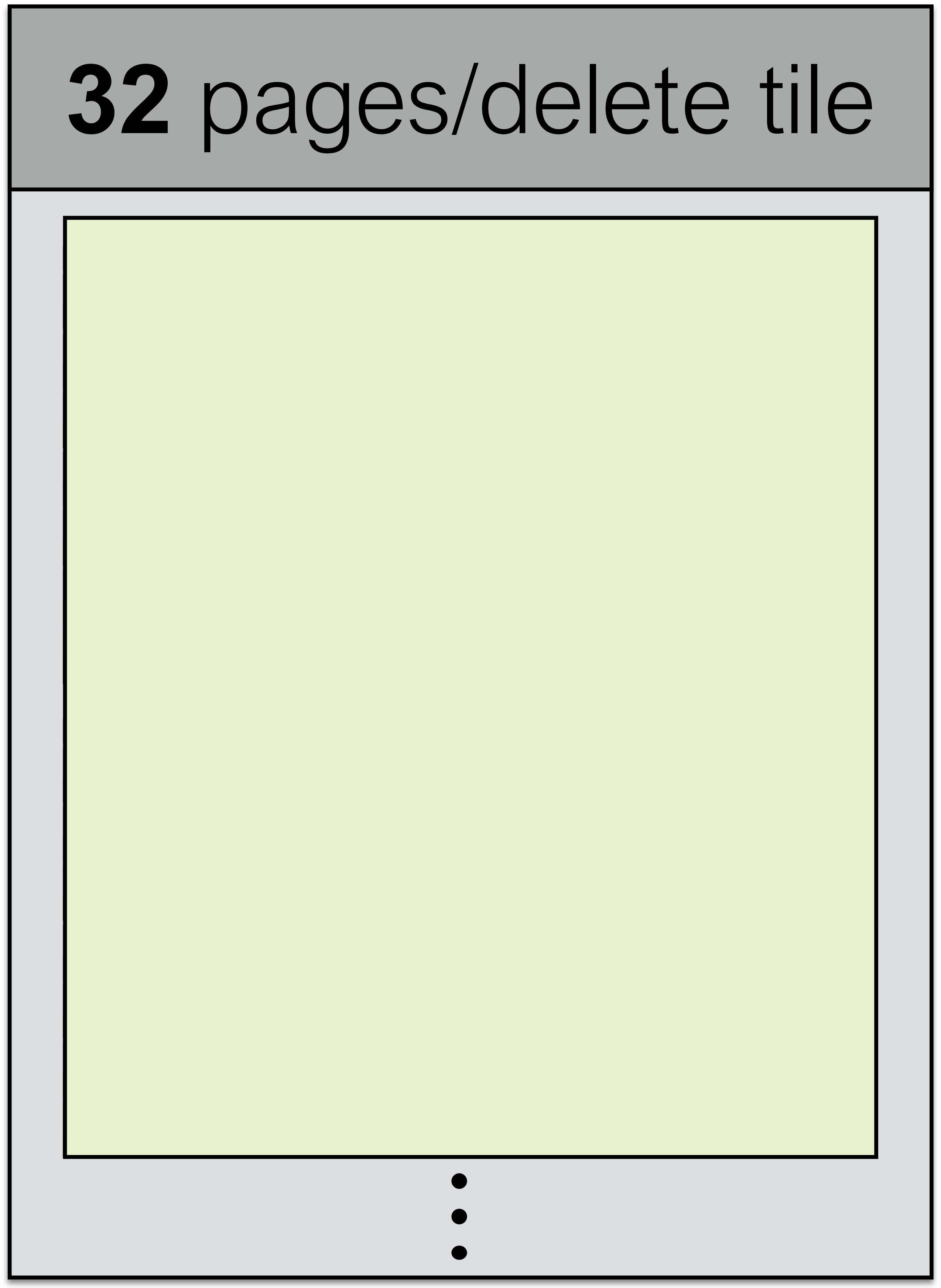


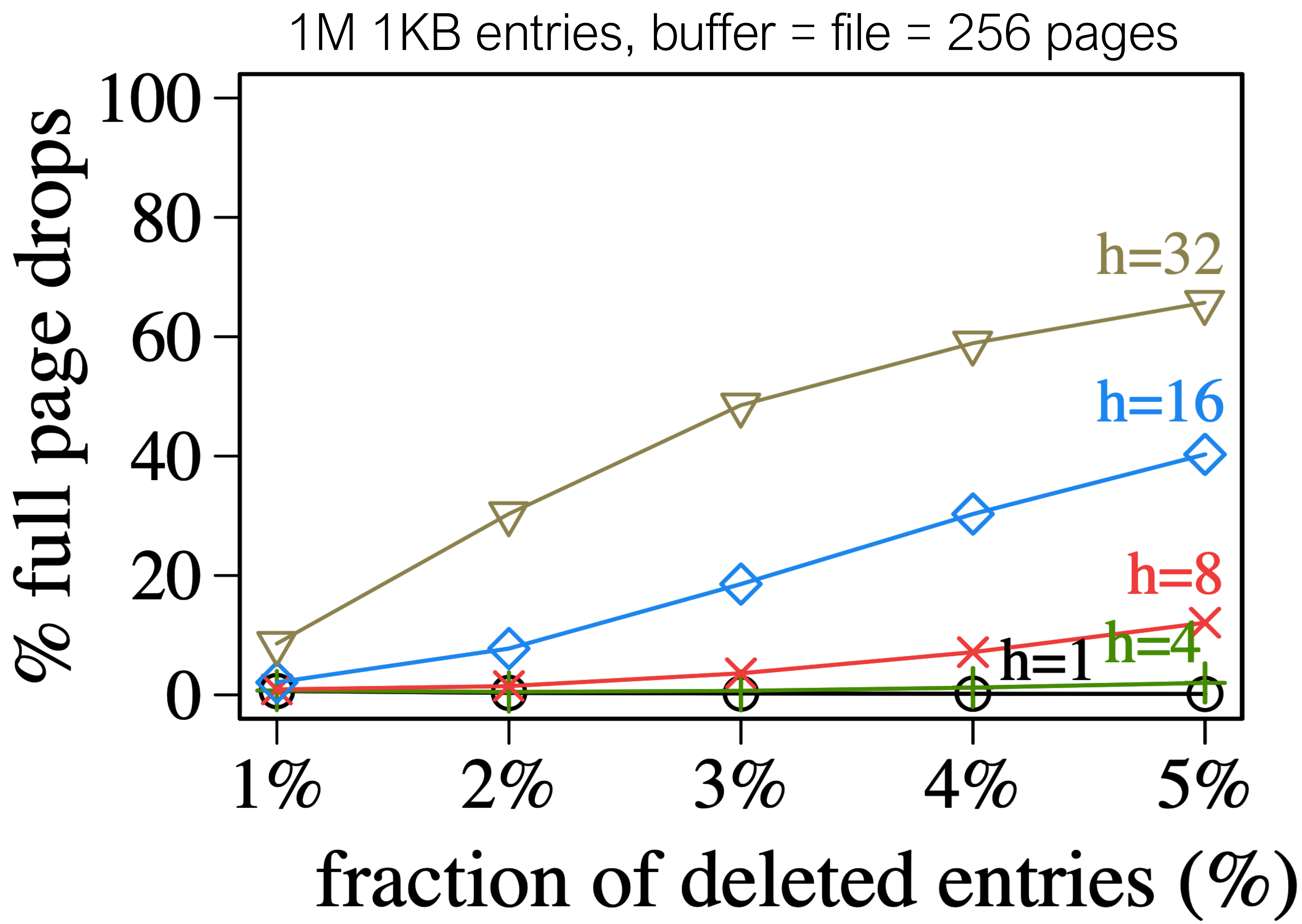
Key Weaving storage layout





Key Weaving storage layout

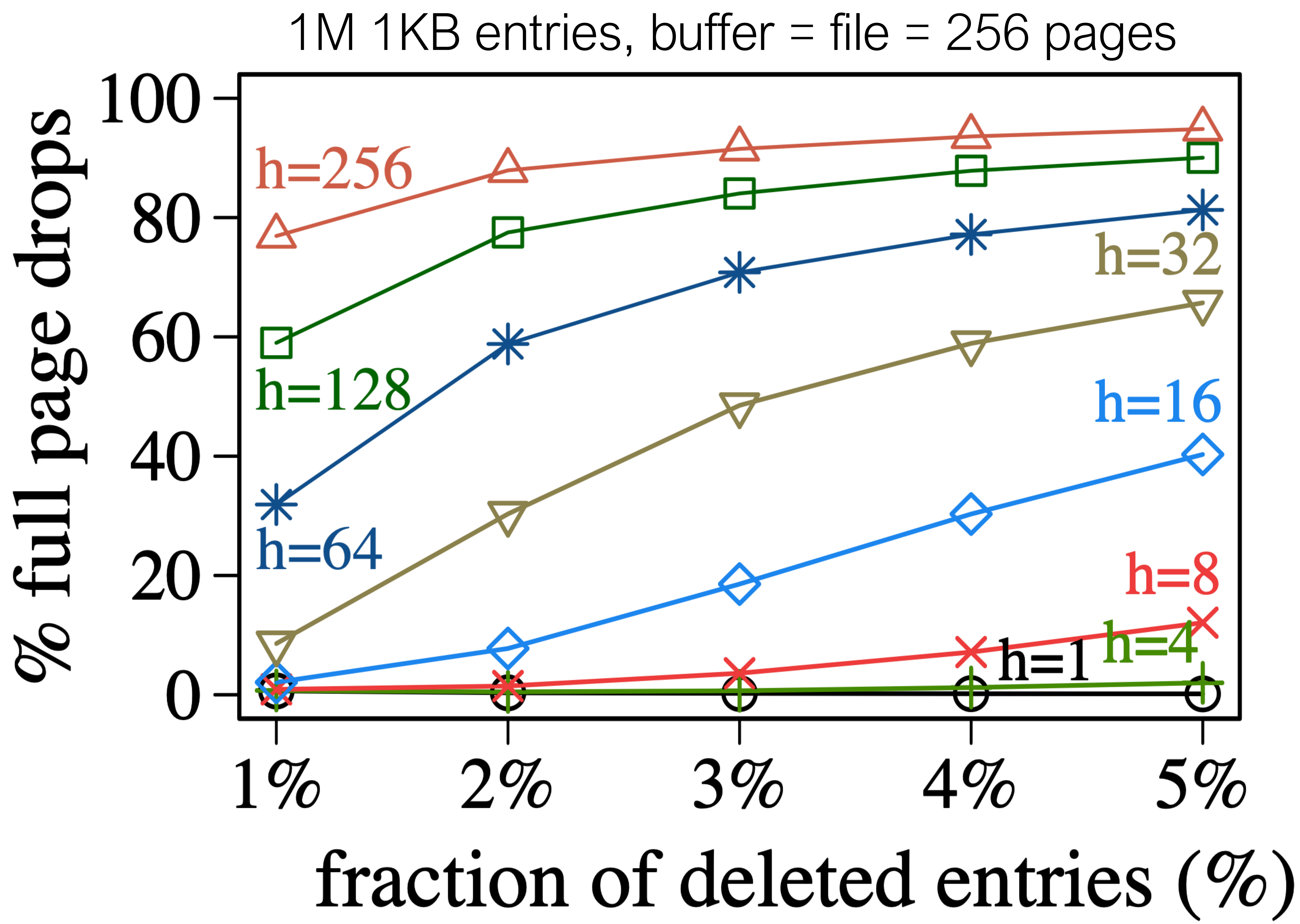




Key Weaving storage layout

reduced latency spikes ✓

full page drops reduces superfluous I/Os ✓

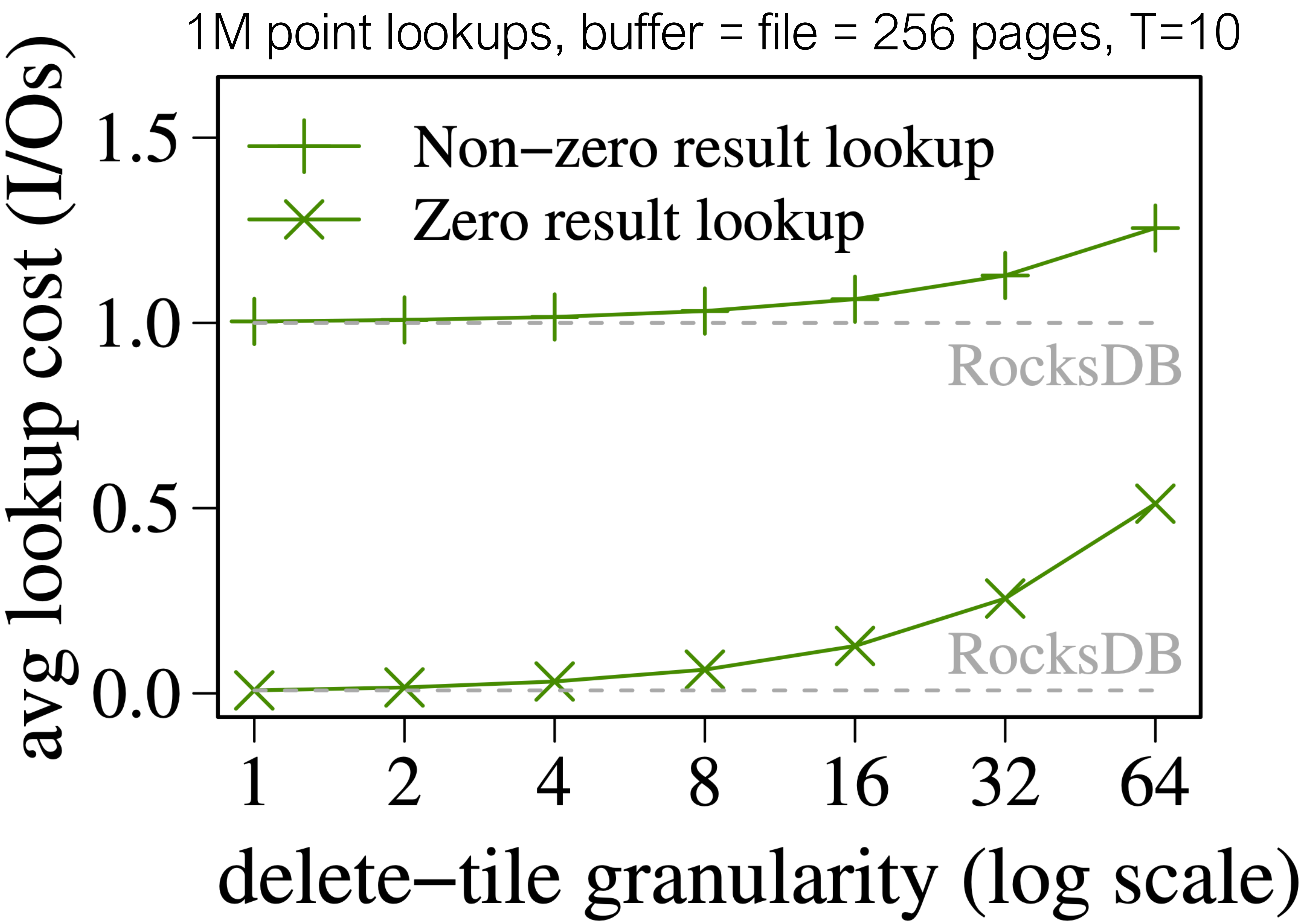


Key Weaving storage layout

higher lookup cost ↑

reduced latency spikes ✓

full page drops reduces superfluous I/Os ✓



the solution

FADE

FAst DElete

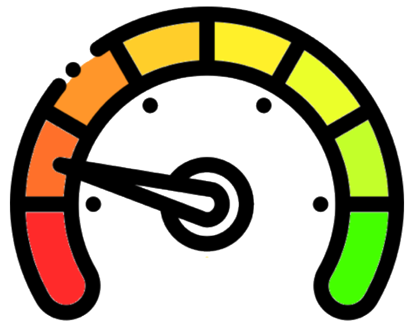
amortized write amplification

timely delete persistence

KiWi

higher lookup cost

full page drops reduces superfluous I/Os



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

BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 6b

Deletes on LSM Trees

Prof. Manos Athanassoulis

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