

CS 561: Data Systems Architectures

class 6b

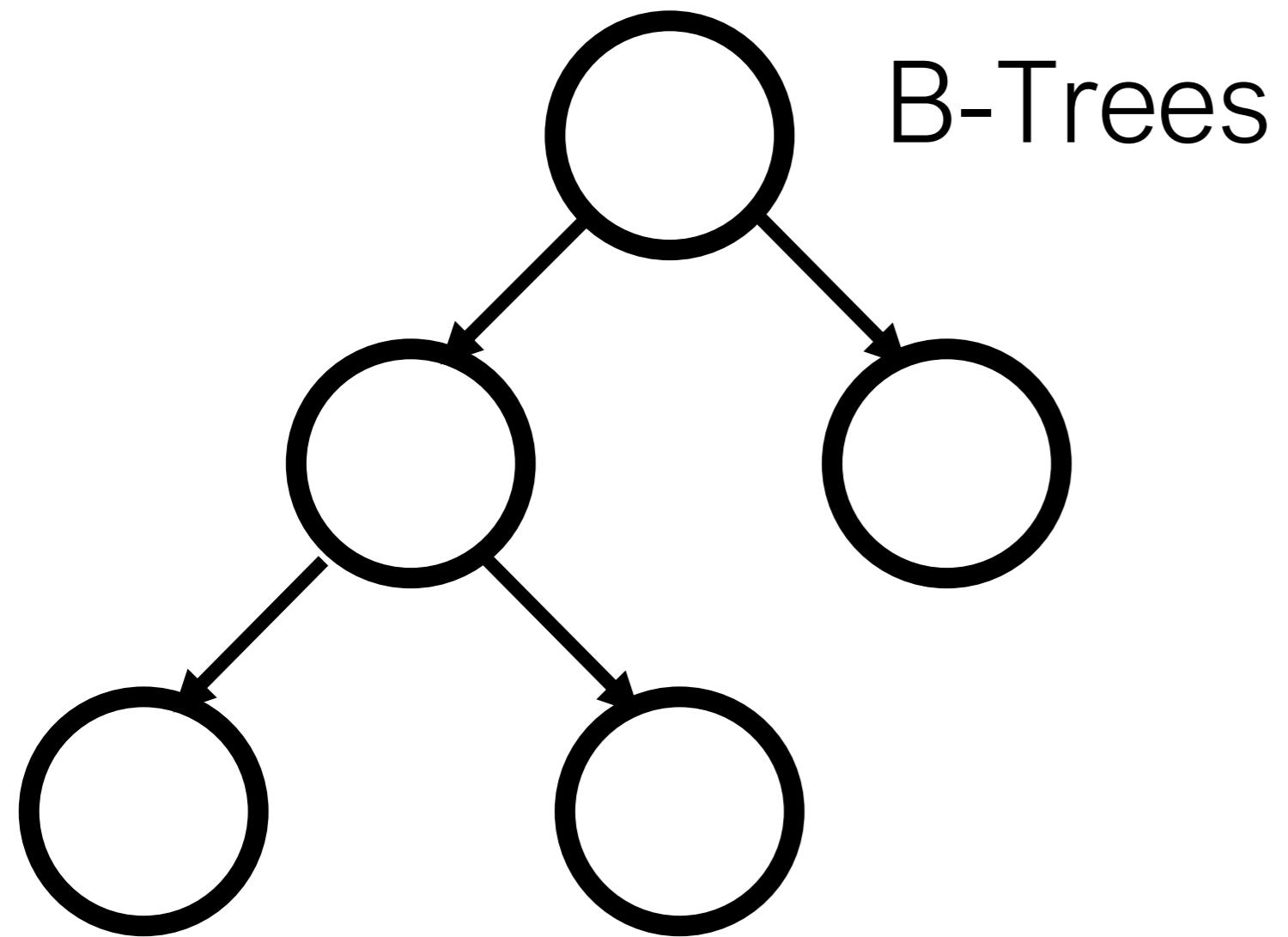
Deletes on LSM Trees

Prof. Manos Athanassoulis

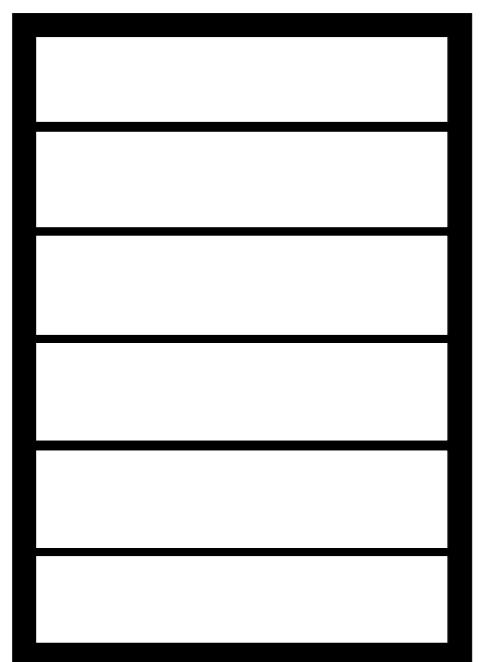
<https://bu-disc.github.io/CS561/>

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

IN-PLACE



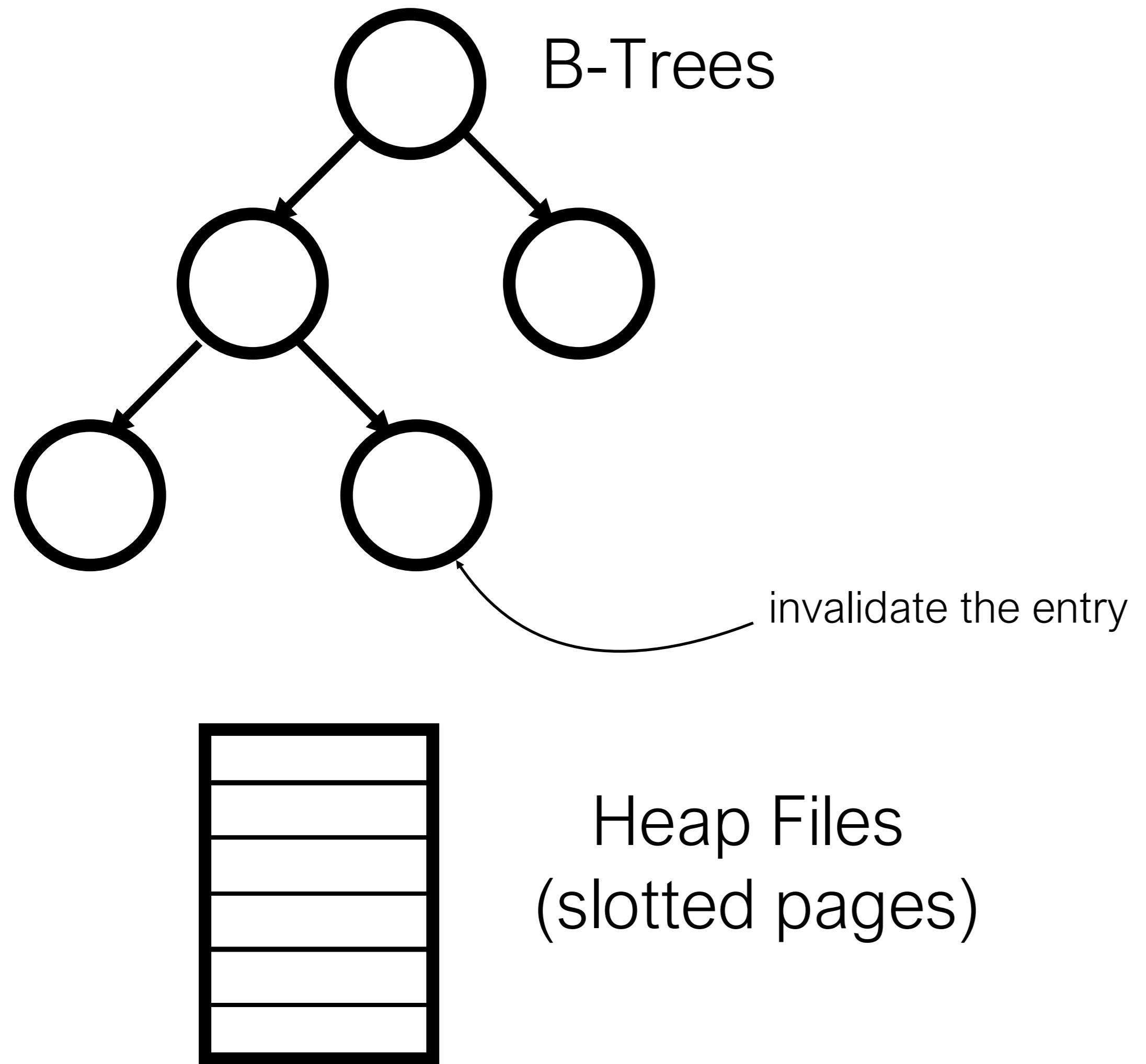
OUT-OF-PLACE



Heap Files
(slotted pages)

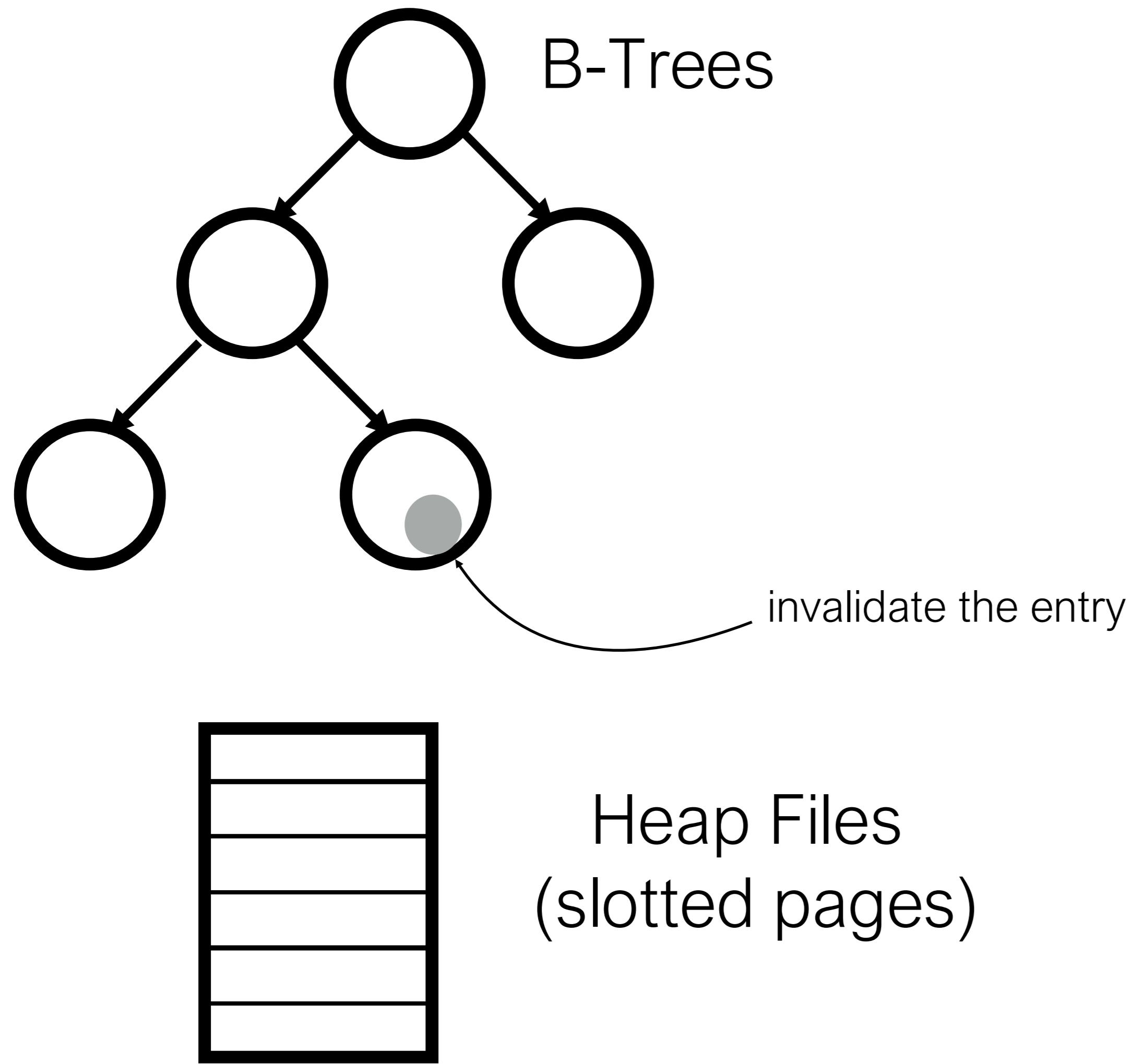
IN-PLACE

OUT-OF-PLACE



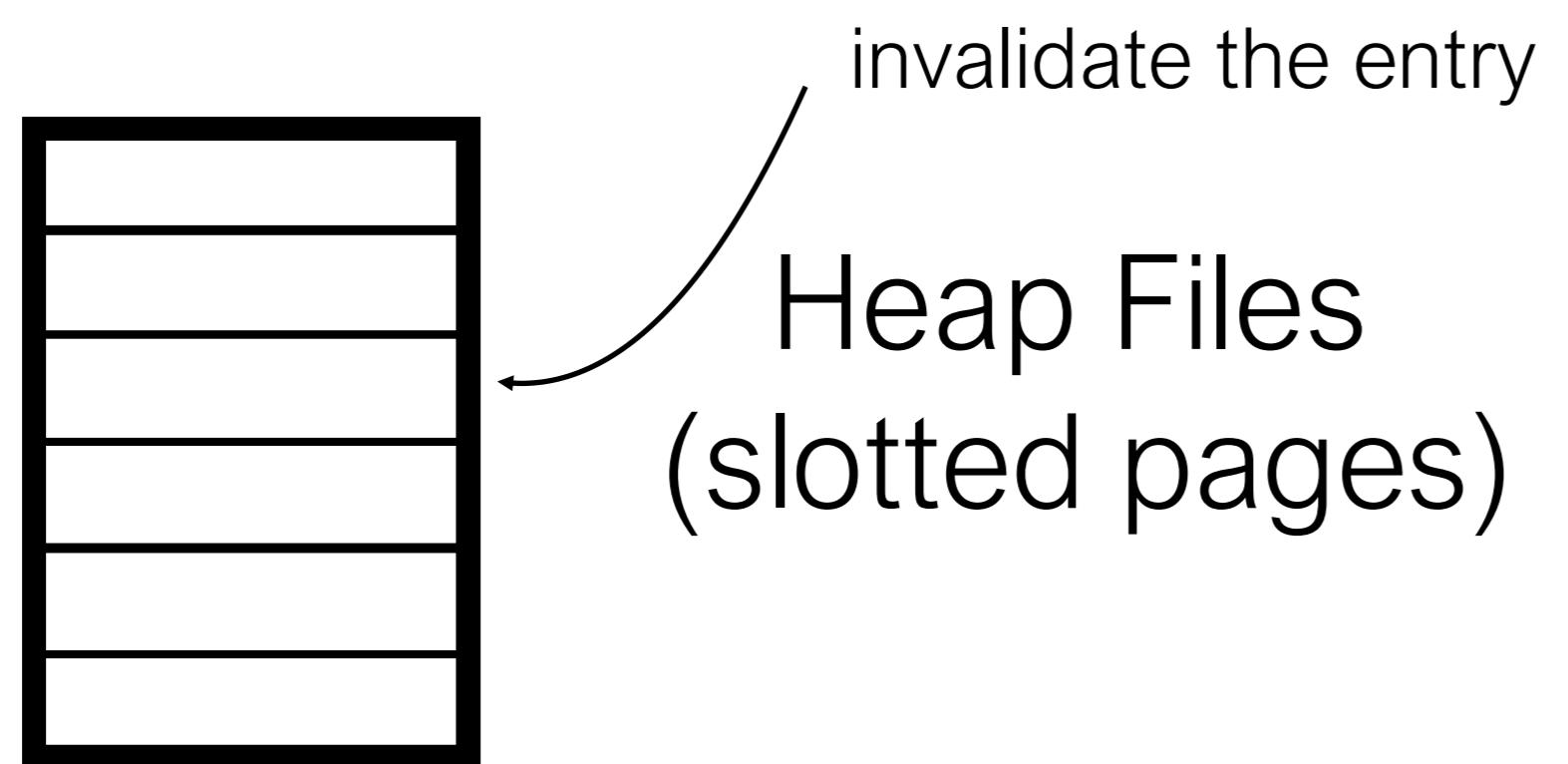
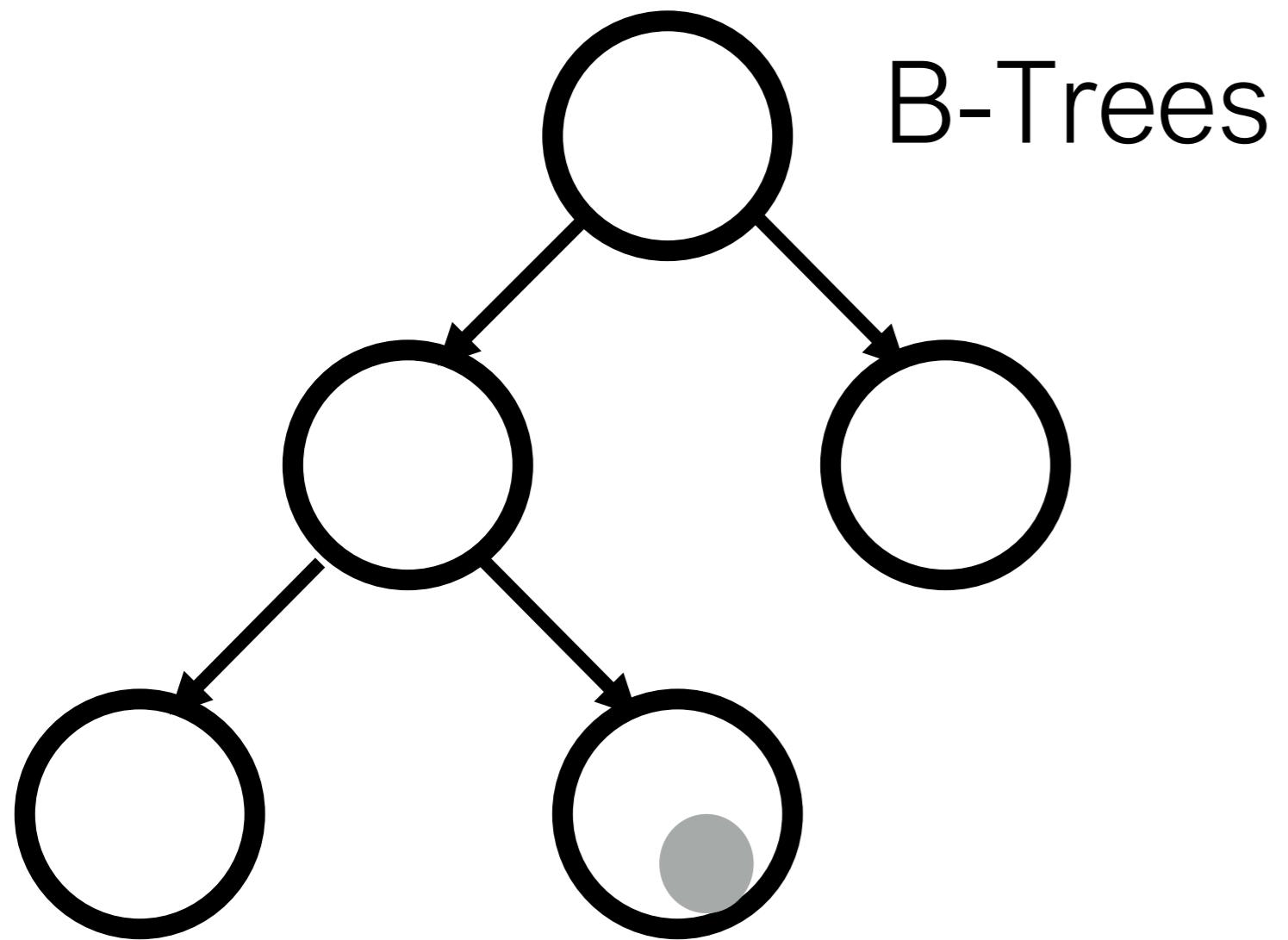
IN-PLACE

OUT-OF-PLACE



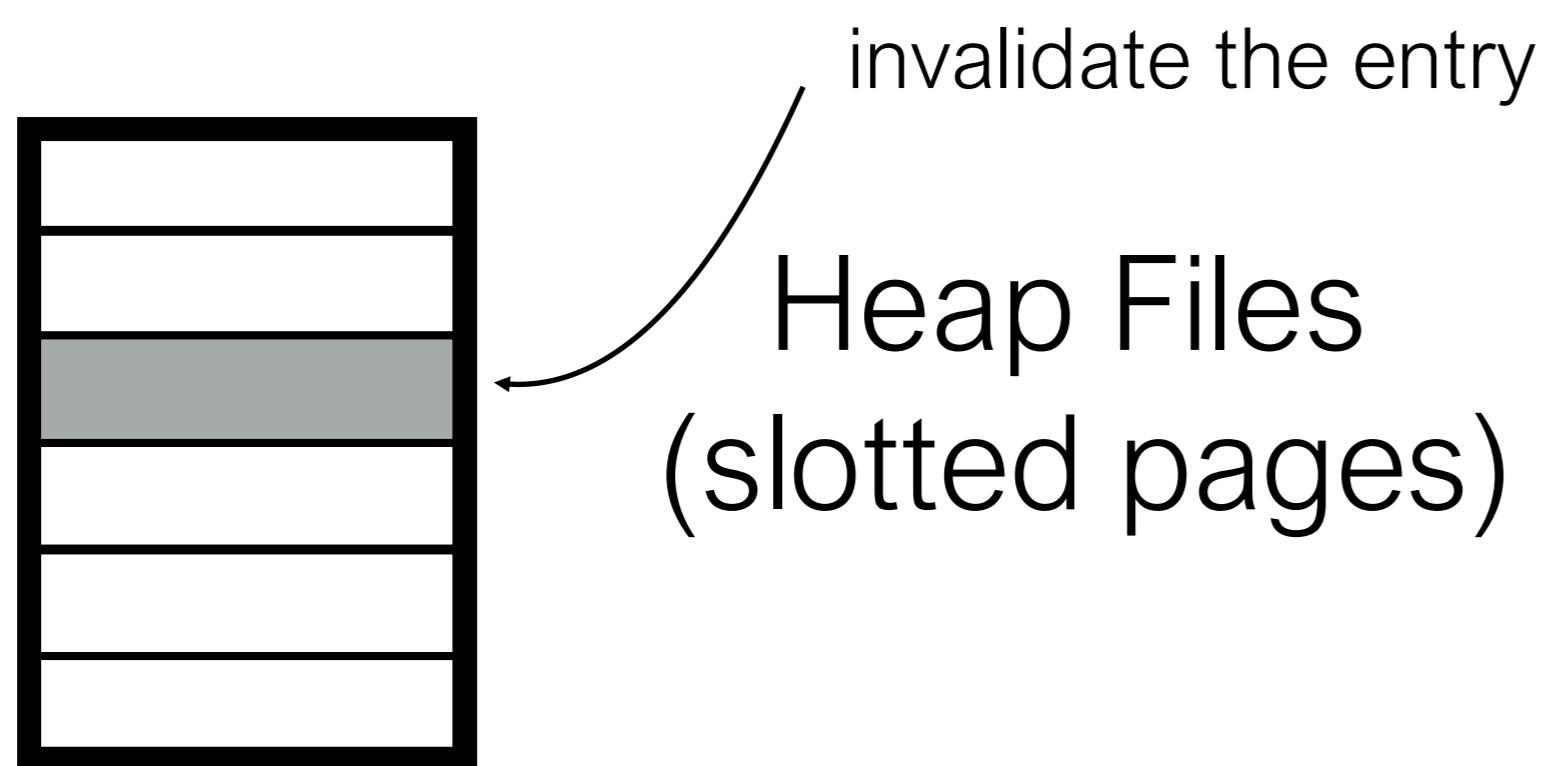
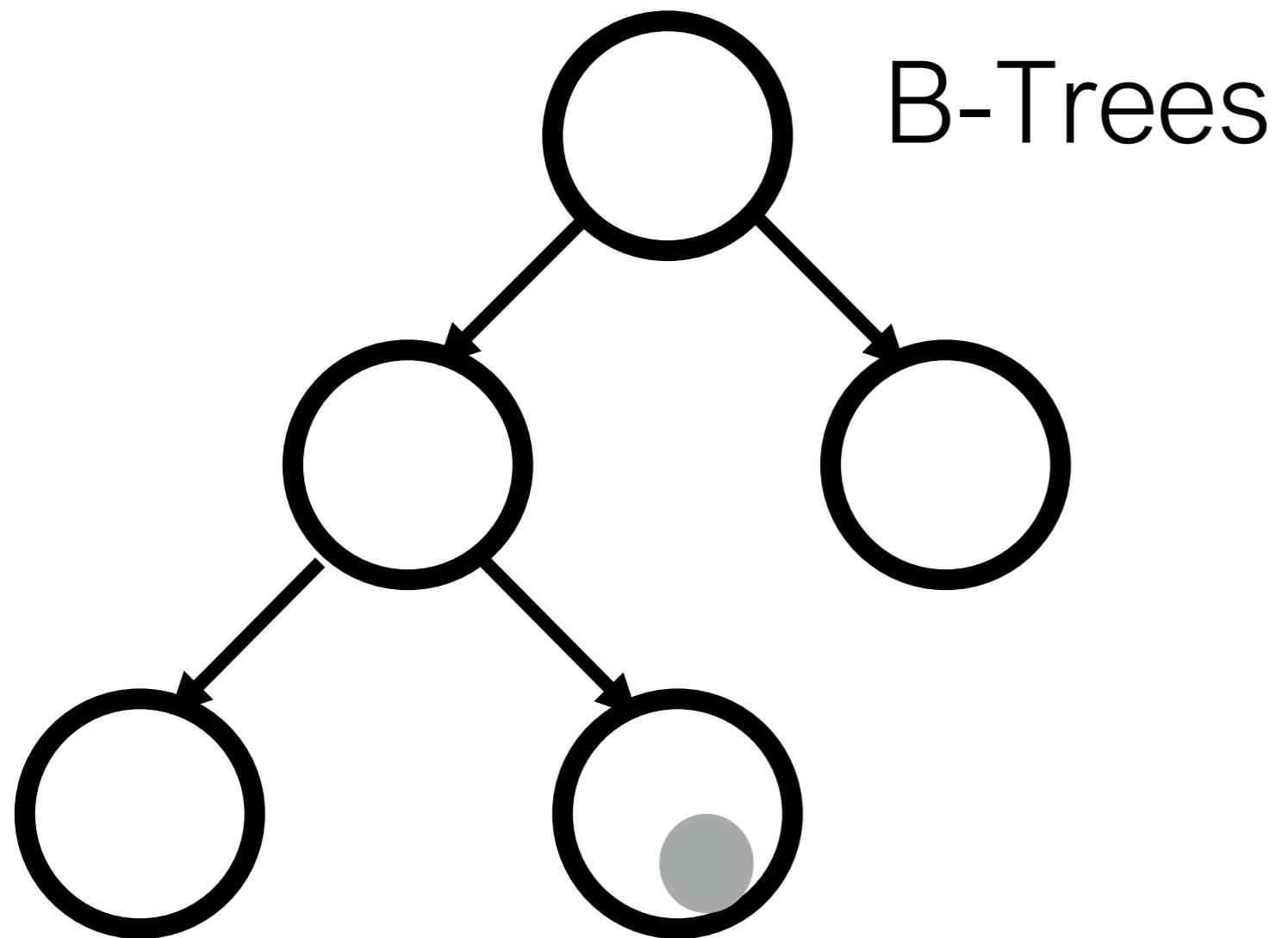
IN-PLACE

OUT-OF-PLACE

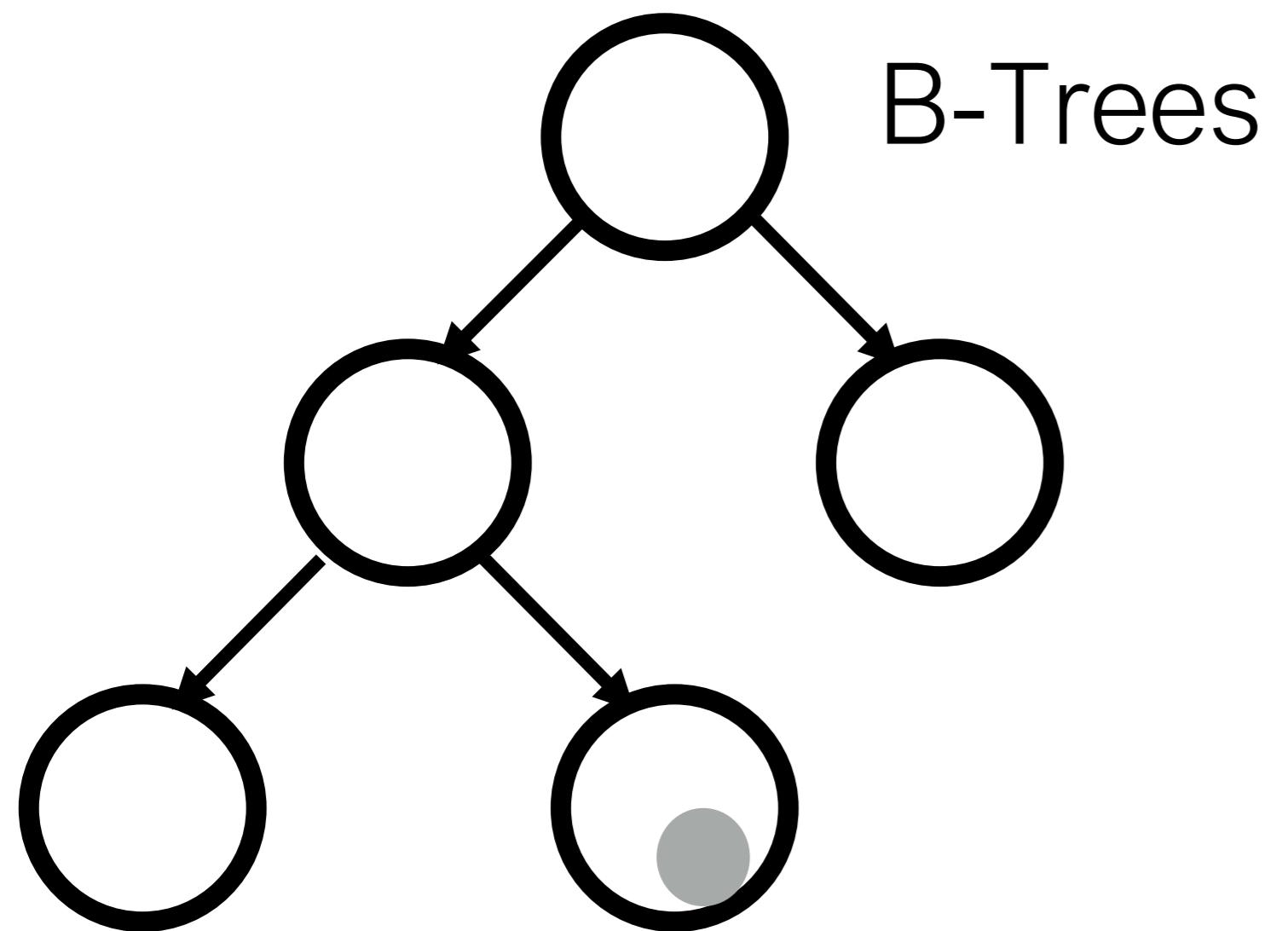


IN-PLACE

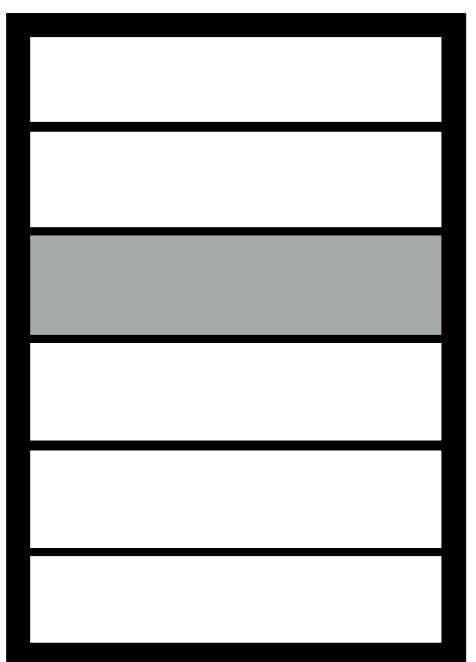
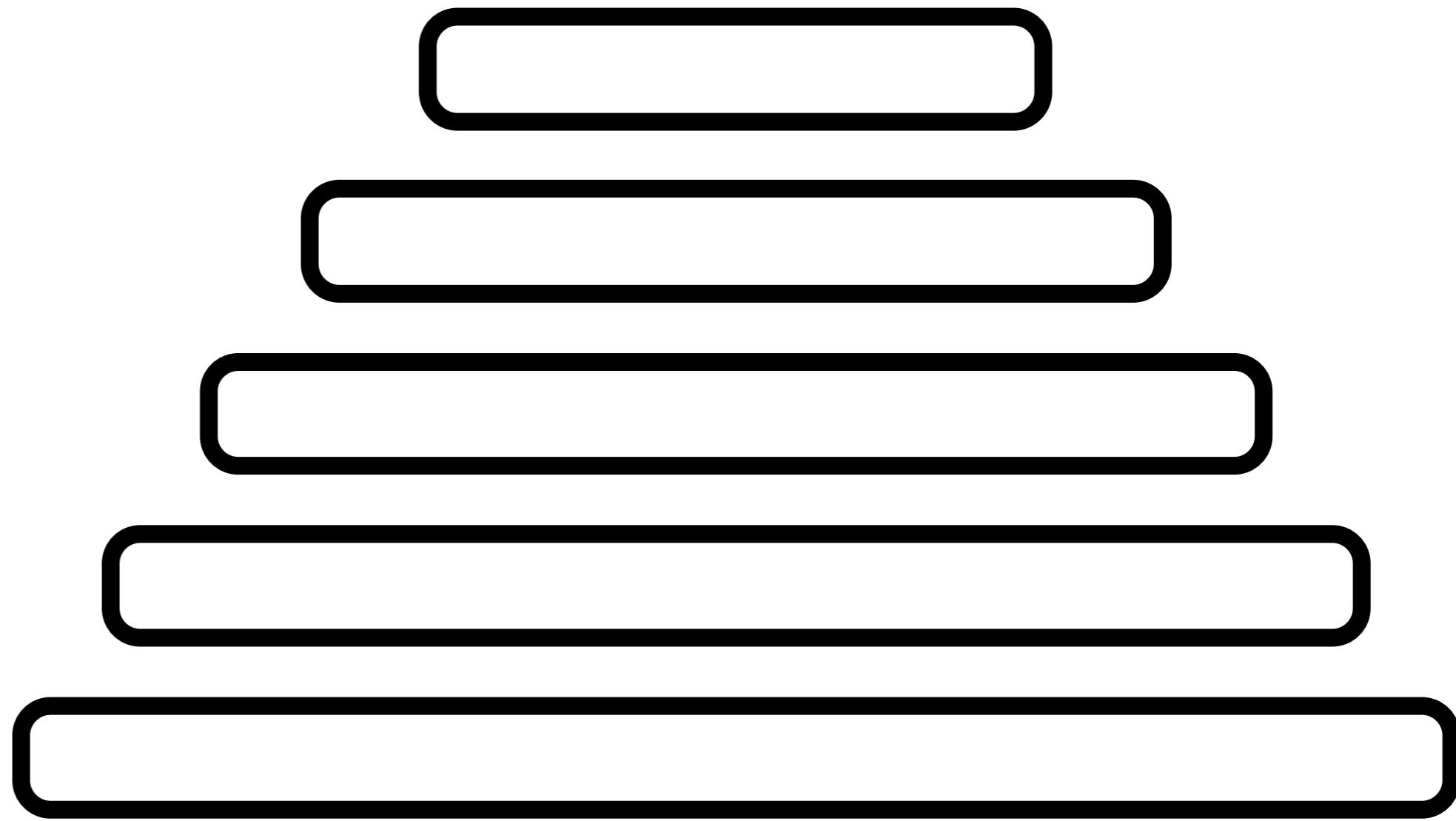
OUT-OF-PLACE



IN-PLACE

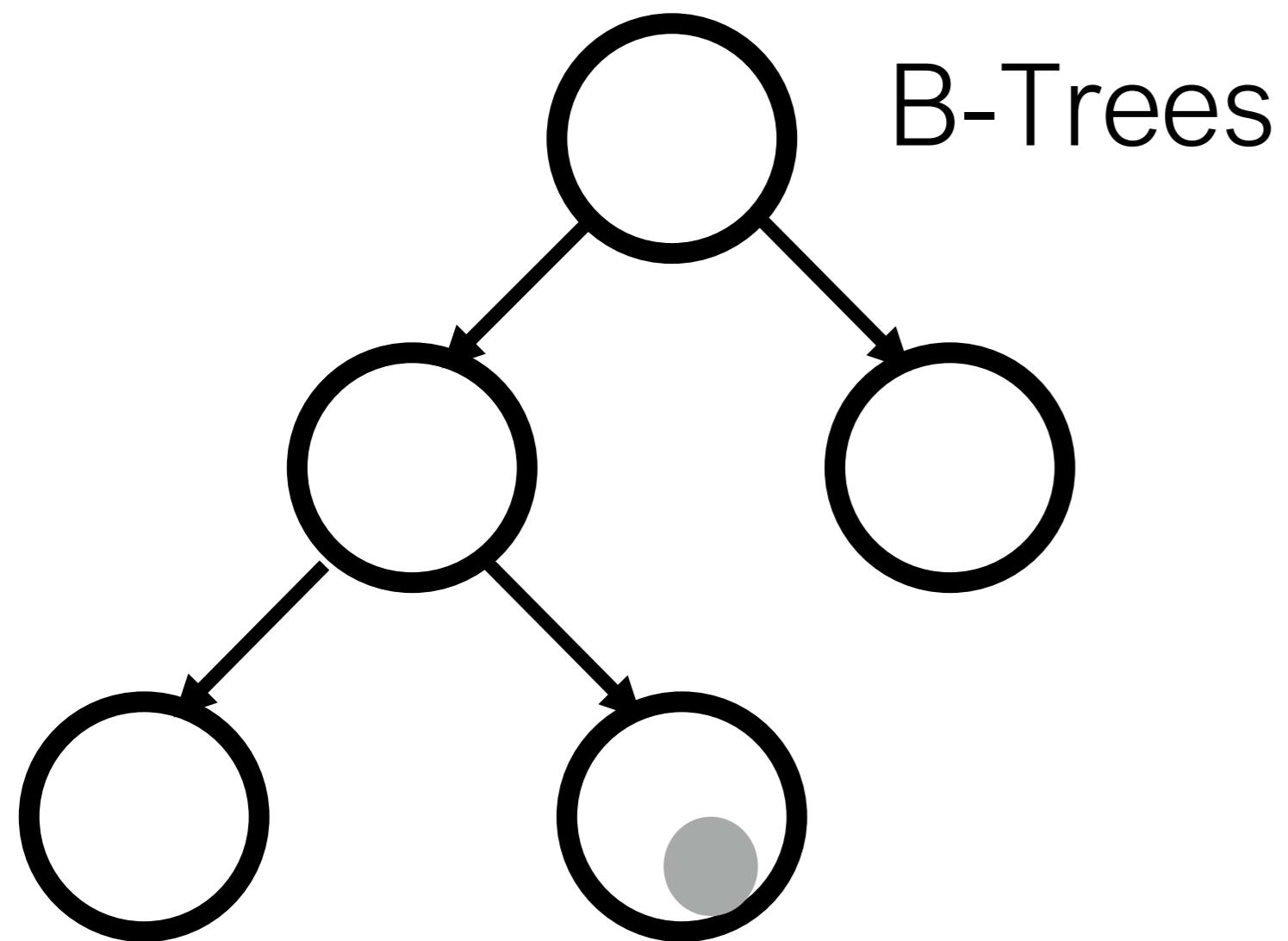


OUT-OF-PLACE



Heap Files
(slotted pages)

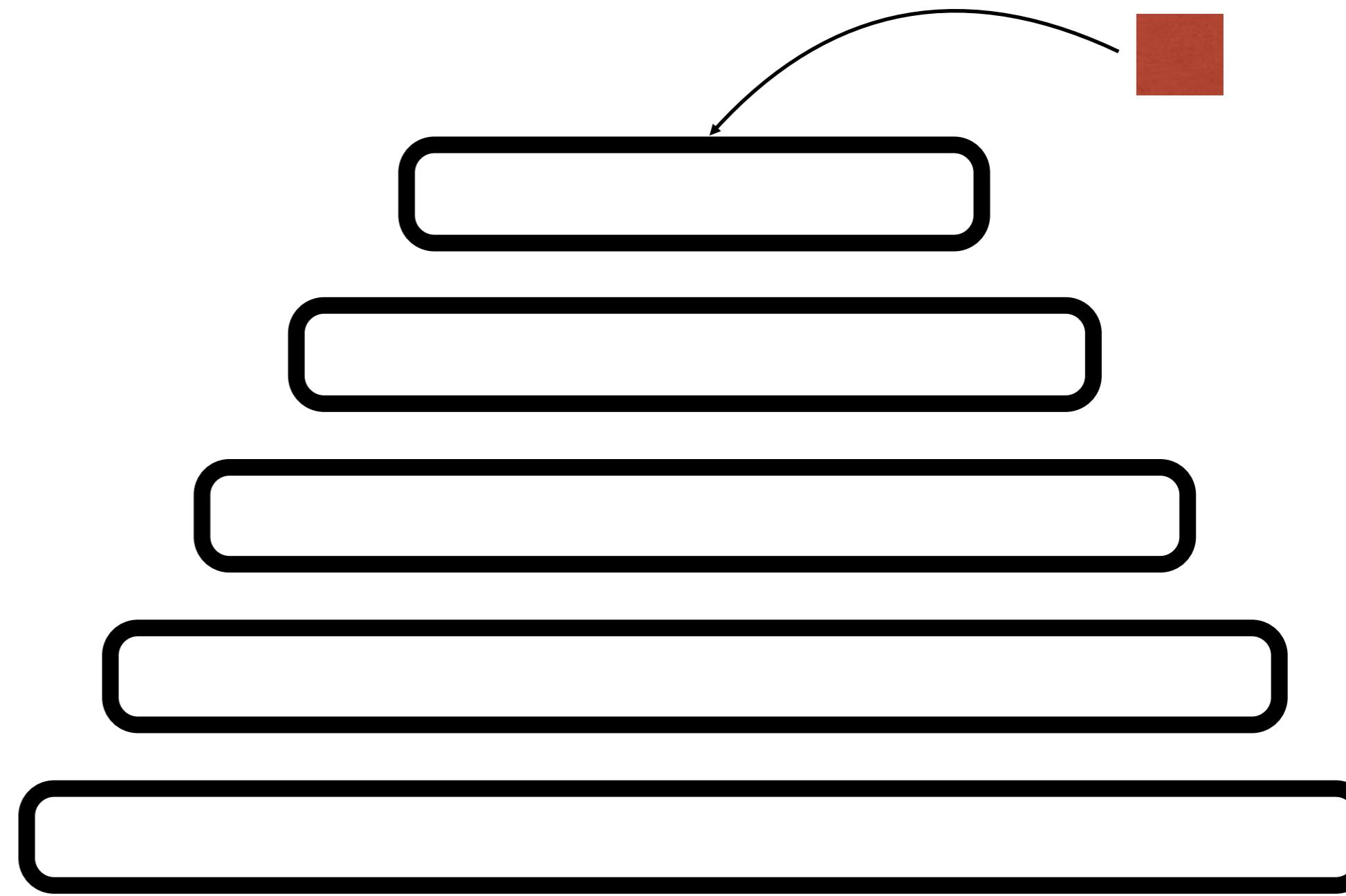
IN-PLACE



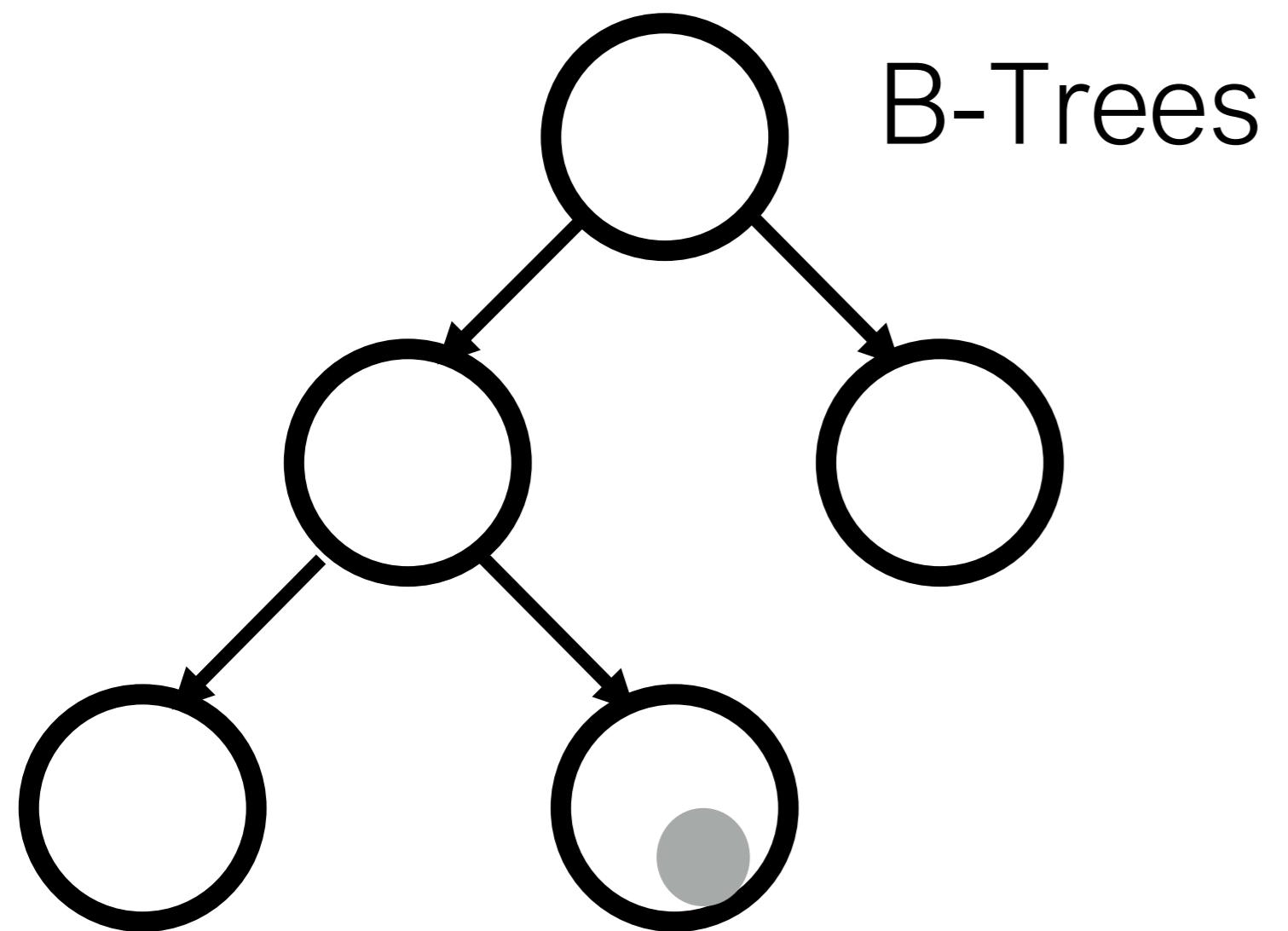
B-Trees
Heap Files
(slotted pages)



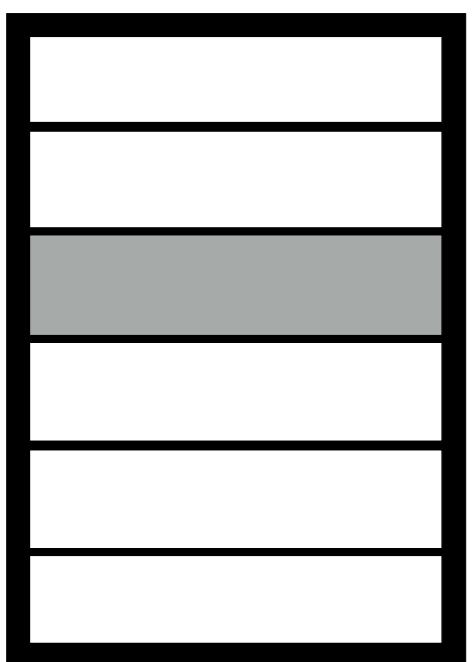
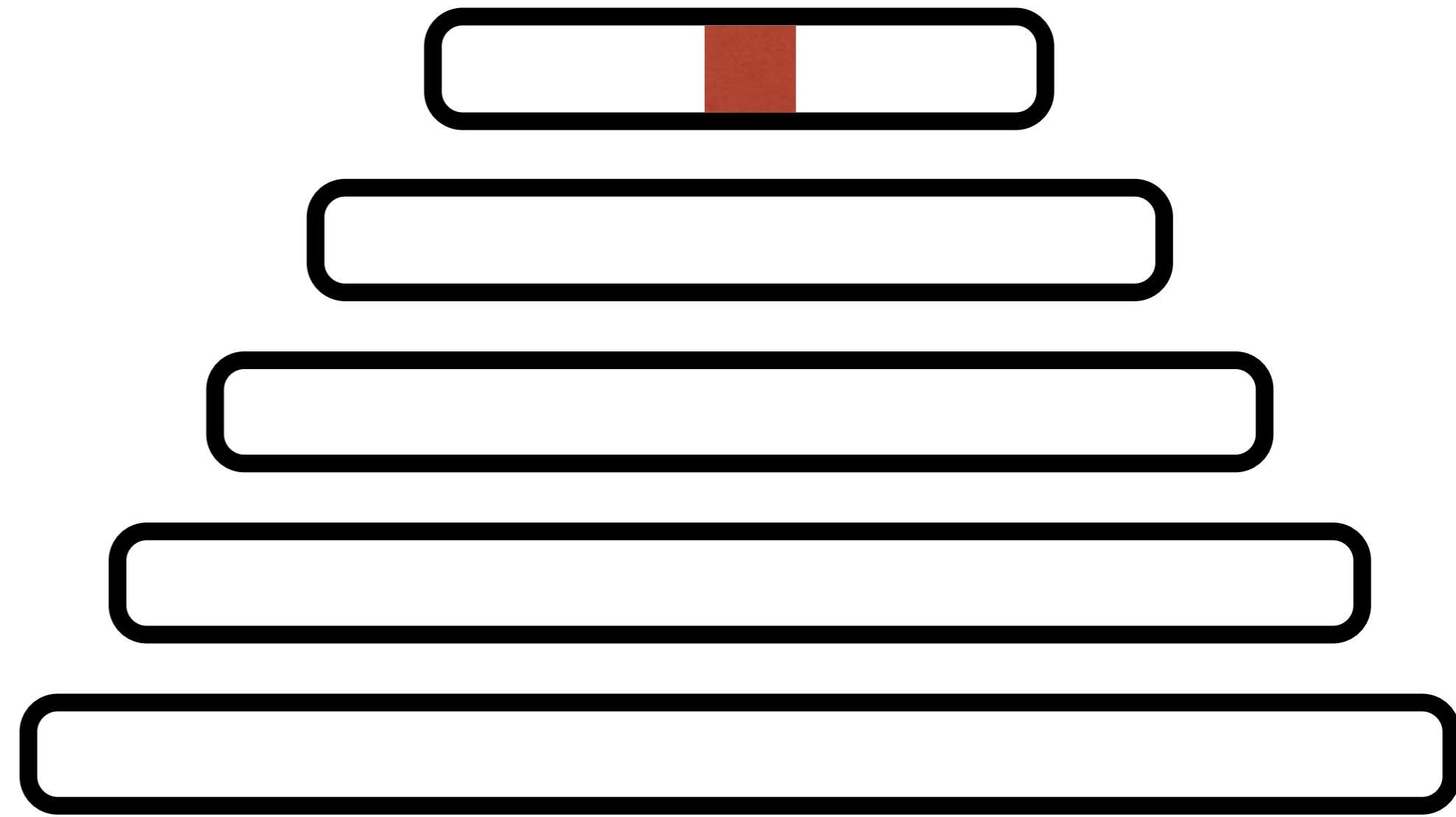
OUT-OF-PLACE



IN-PLACE

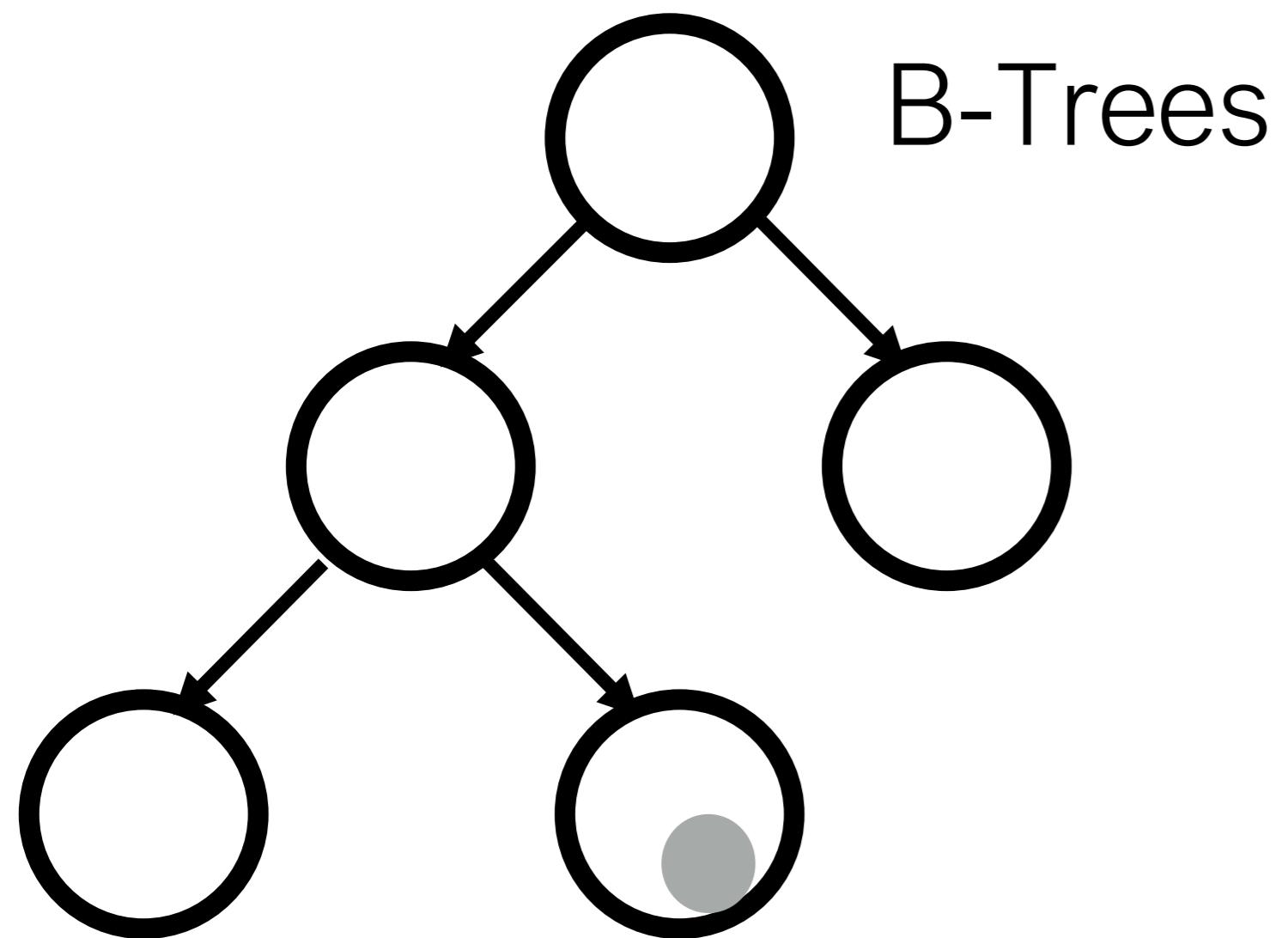


OUT-OF-PLACE

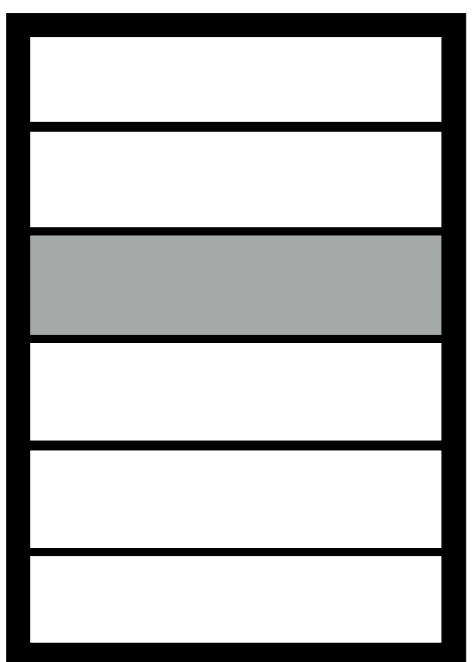
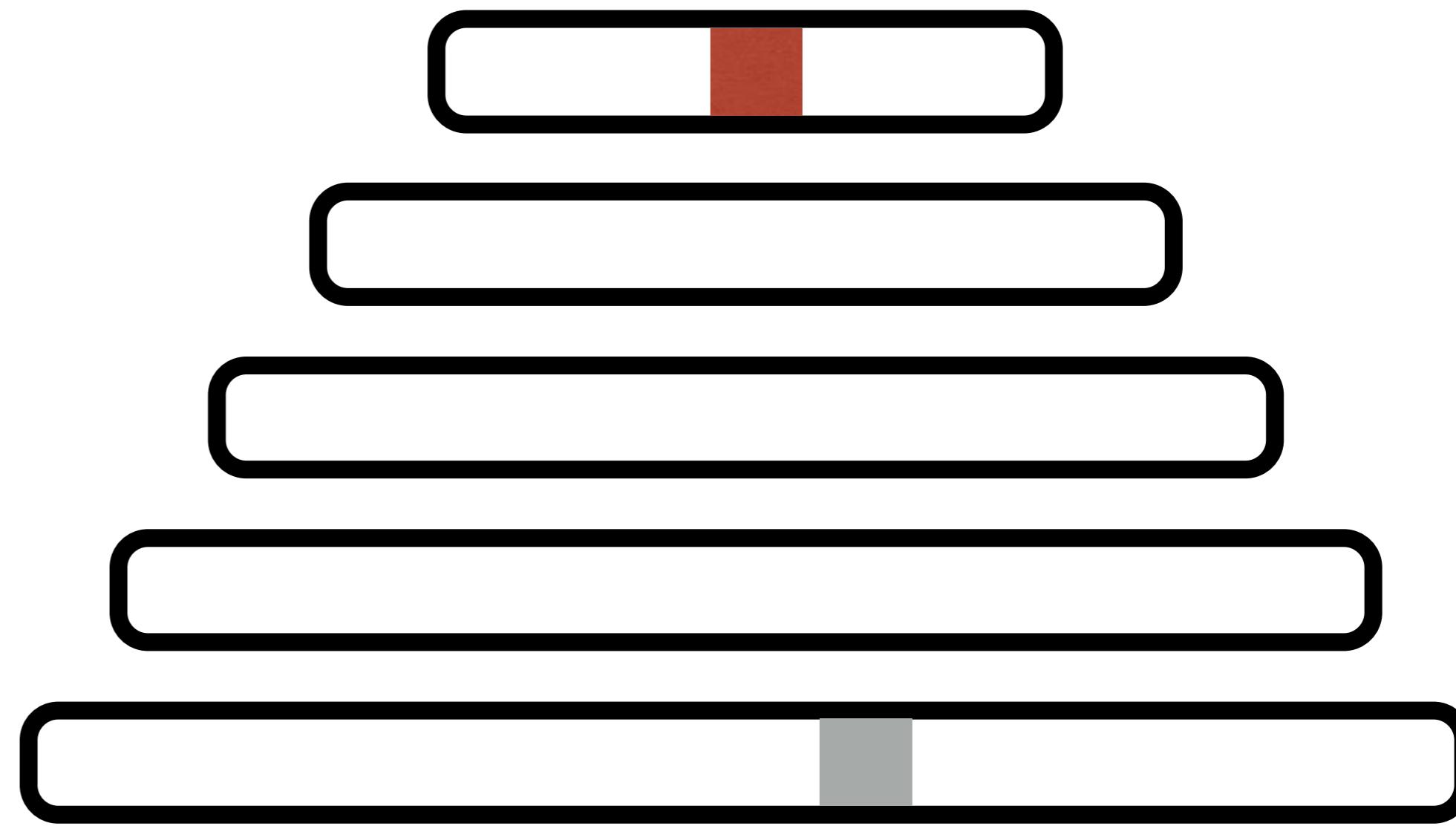


Heap Files
(slotted pages)

IN-PLACE



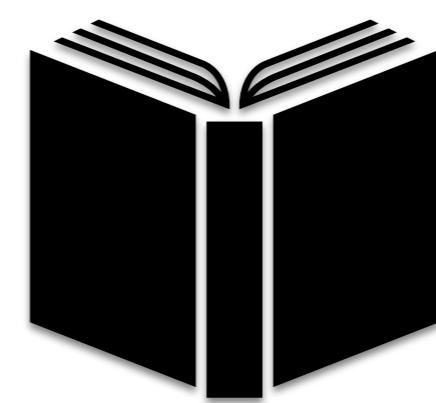
OUT-OF-PLACE



Heap Files
(slotted pages)

What is the delete tradeoff?

What is the delete tradeoff?



read

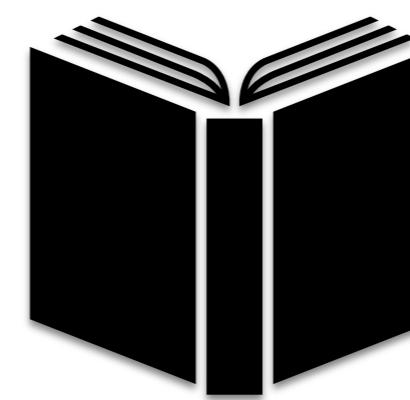


write

vs.

?

What is the delete tradeoff?



read



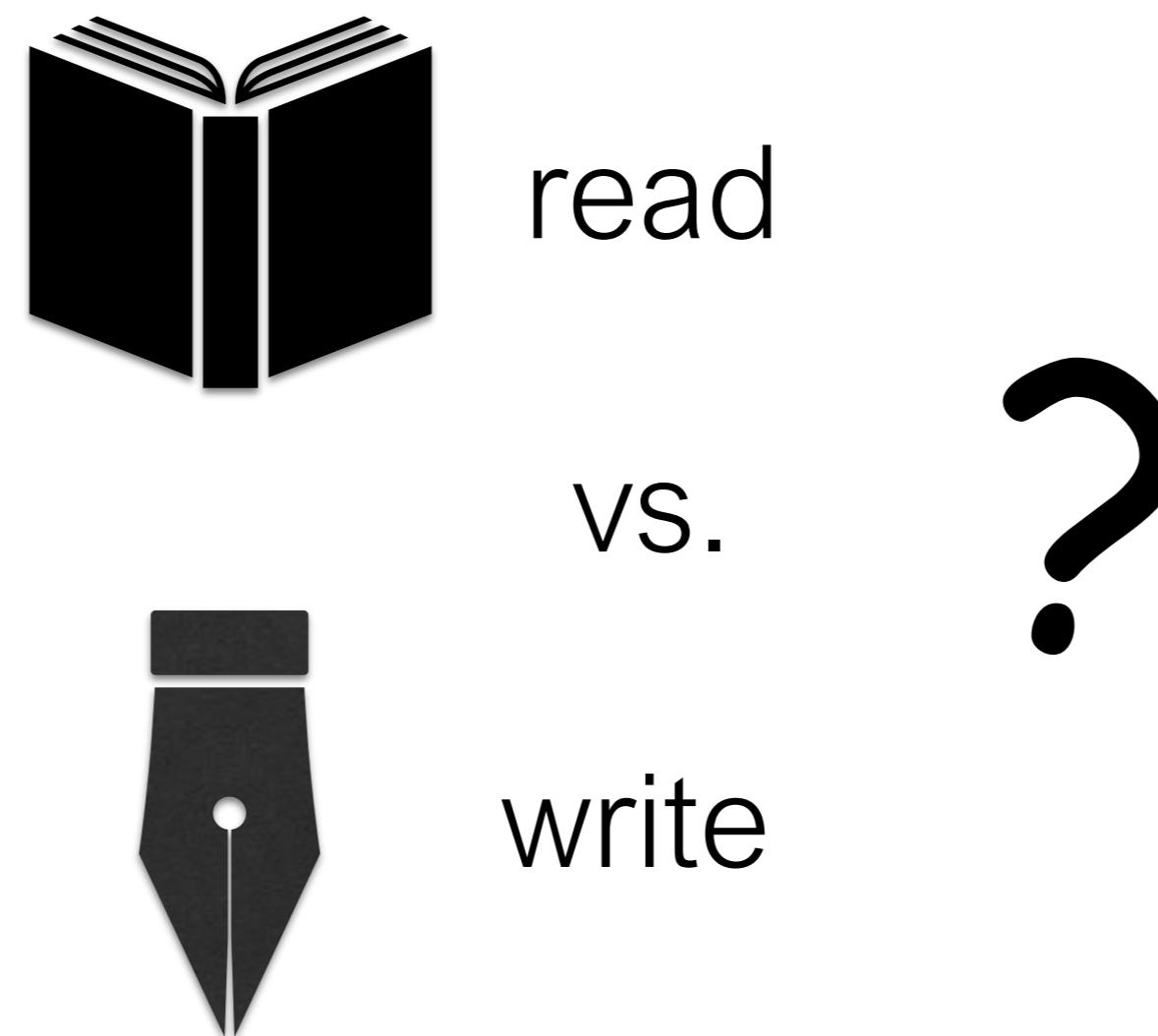
write

vs.

?

Deletes are almost **exclusively** *logical*

What is the delete tradeoff?



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?

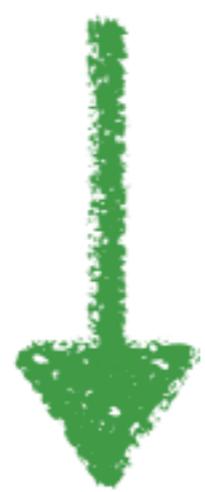
what if we persisted the
deletes immediately?

delete tradeoffs

delete latency vs. **persistent** delete latency

delete tradeoffs

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

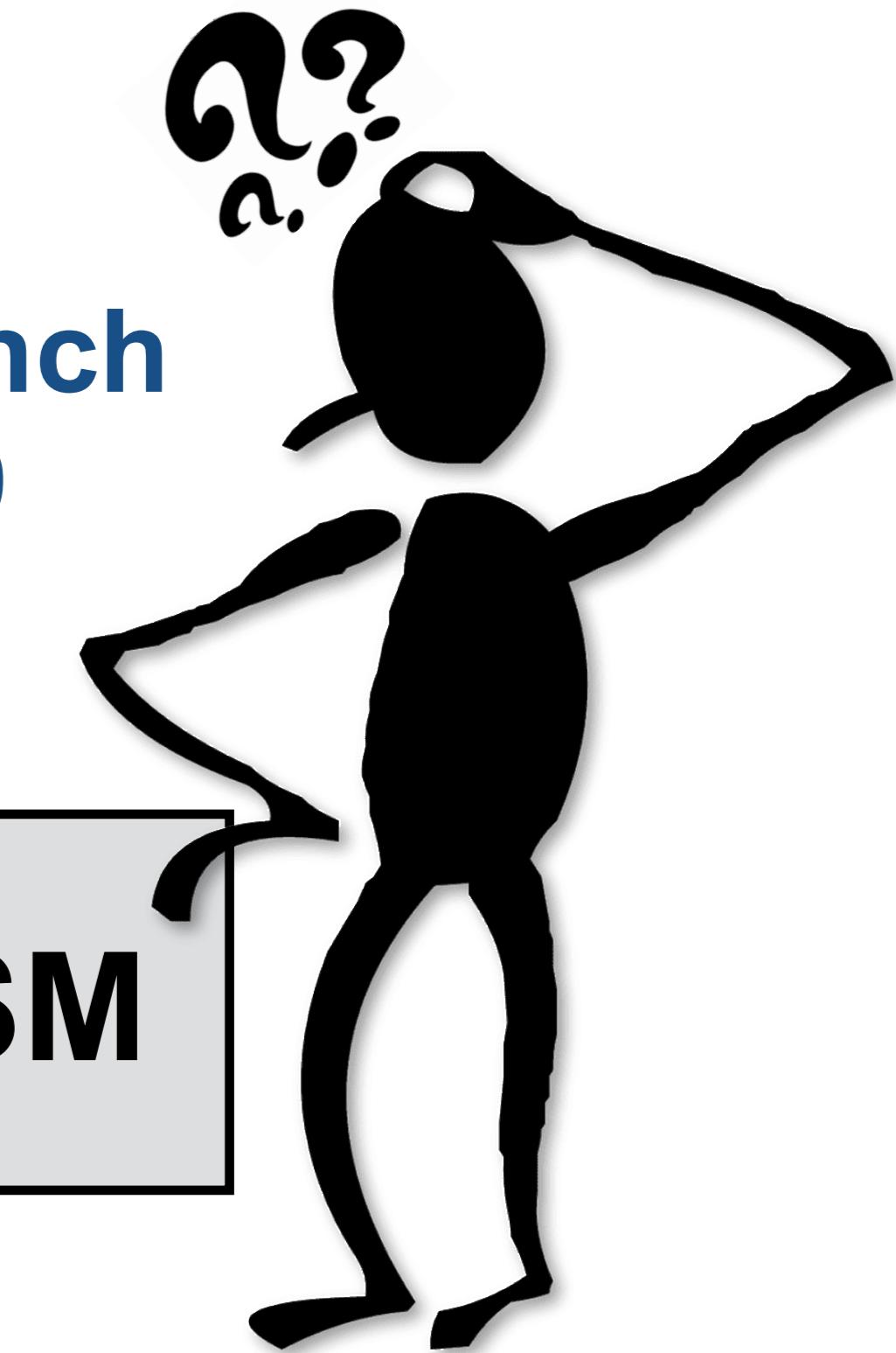


**Even years later, Twitter doesn't
delete your direct messages**

Small Datum
Jan '20

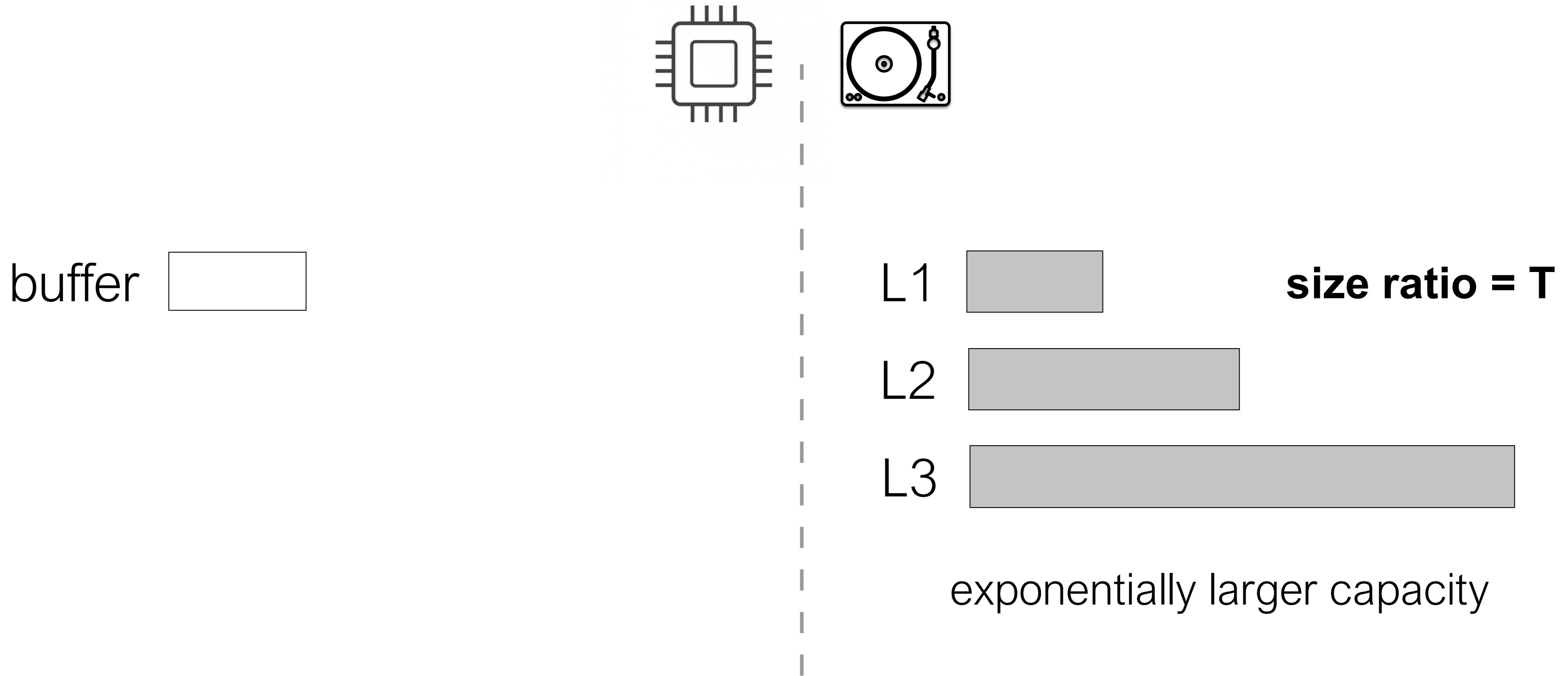
TechCrunch
Feb '19

Deletes are fast and slow in an LSM

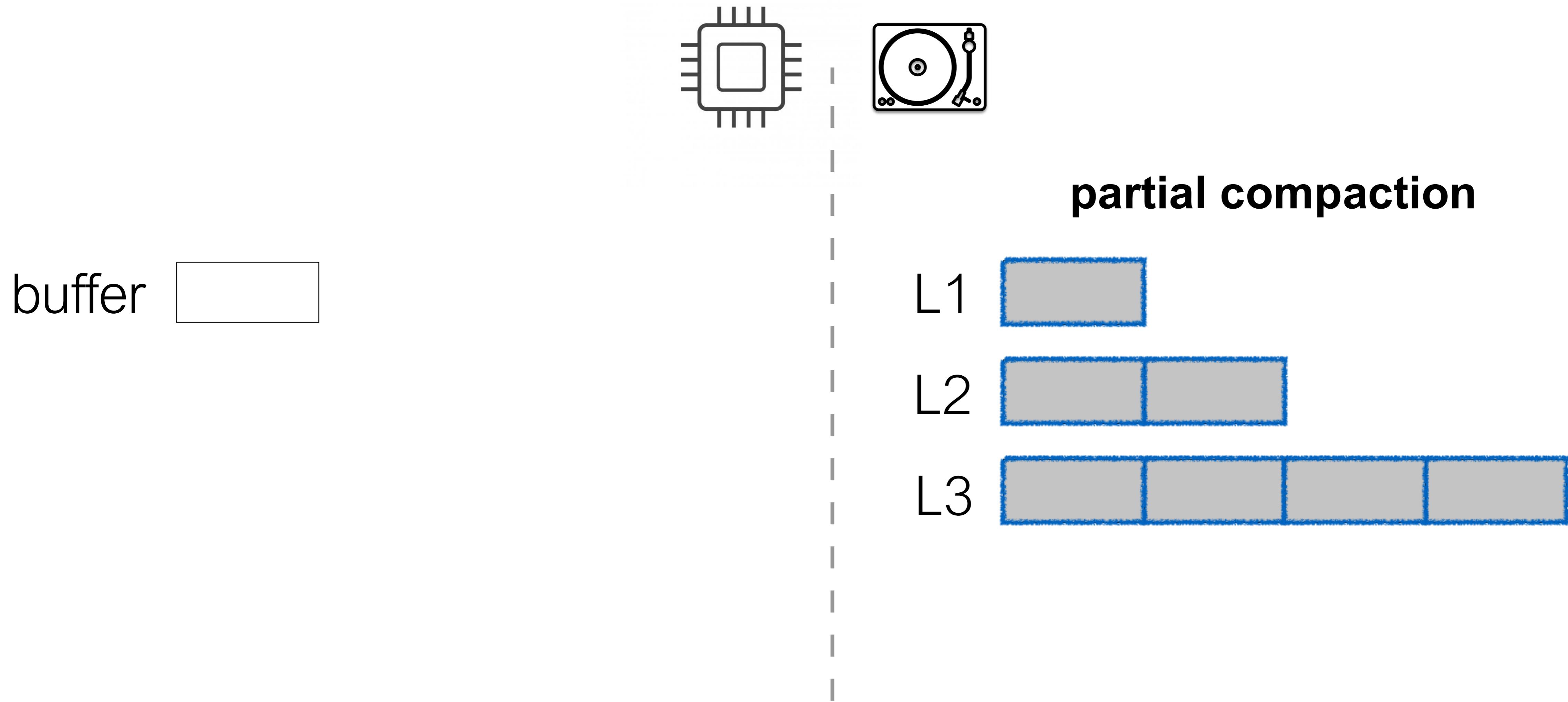


“LSM-based data stores perform suboptimally for workloads with deletes.”

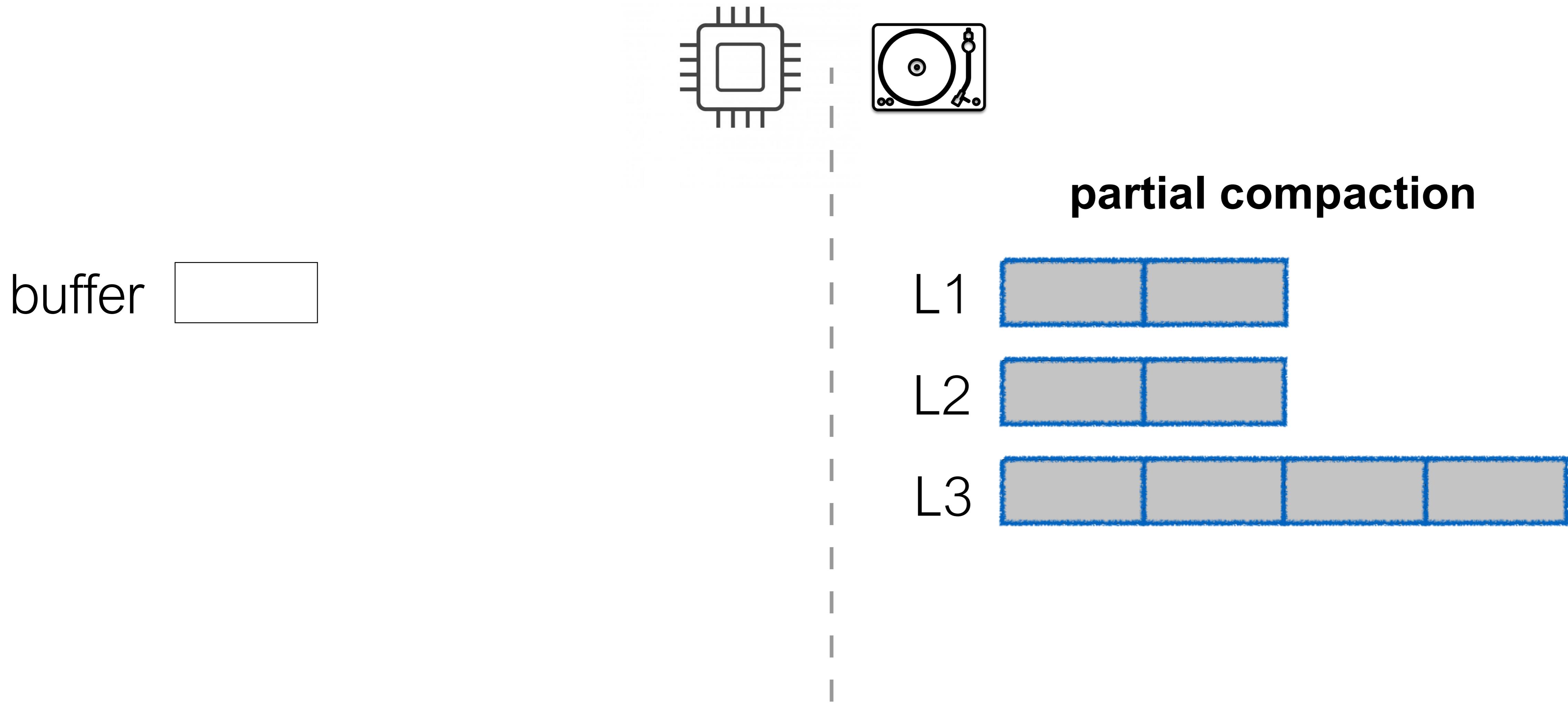
log-structured merge-tree



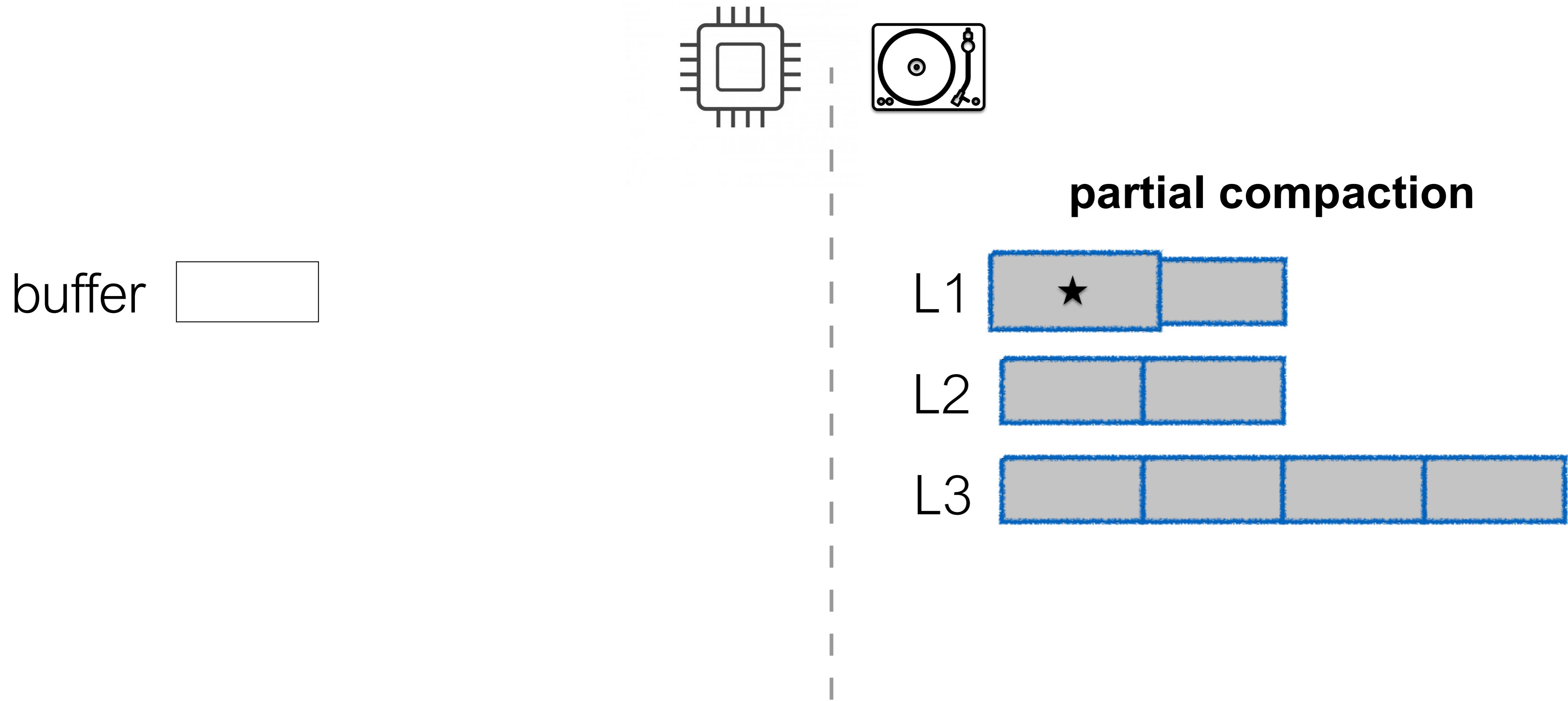
log-structured merge-tree



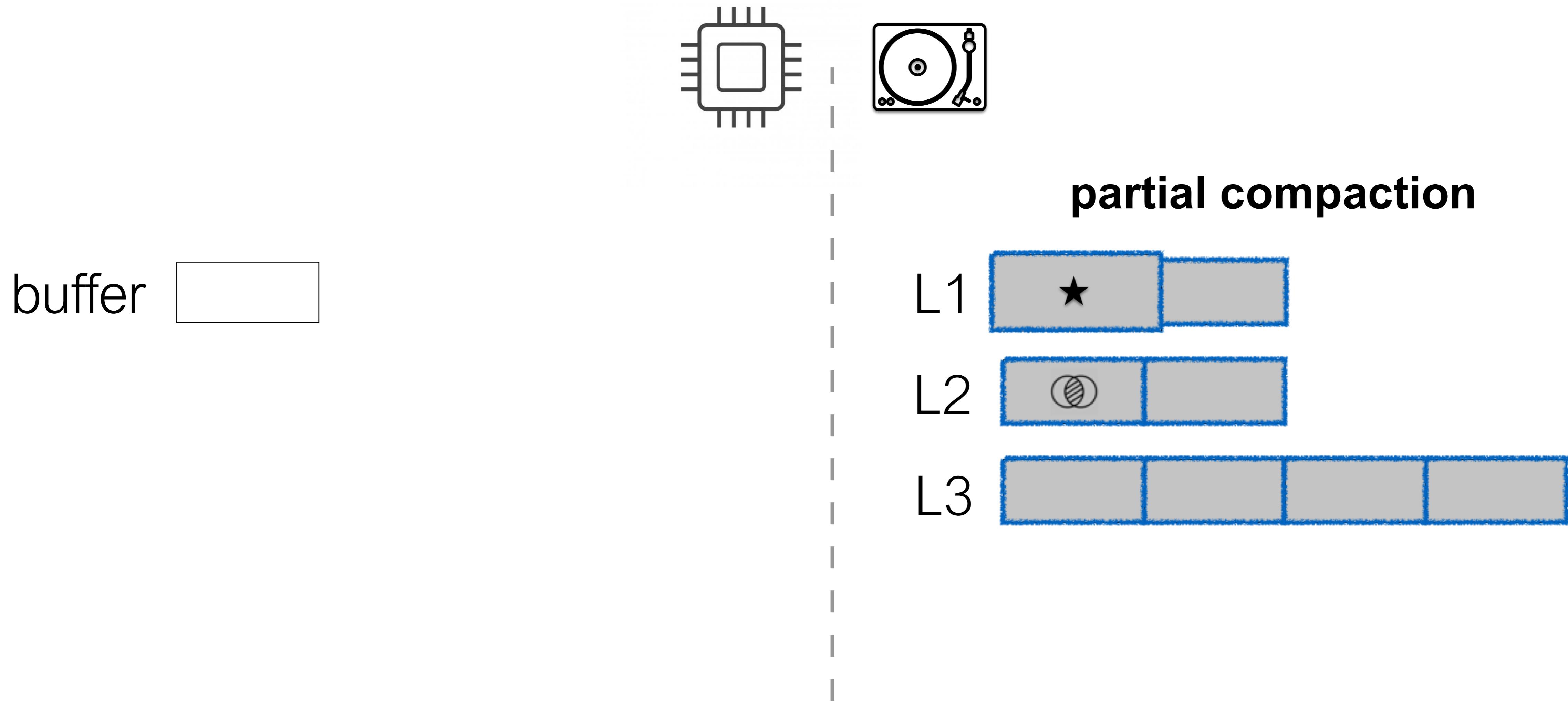
log-structured merge-tree



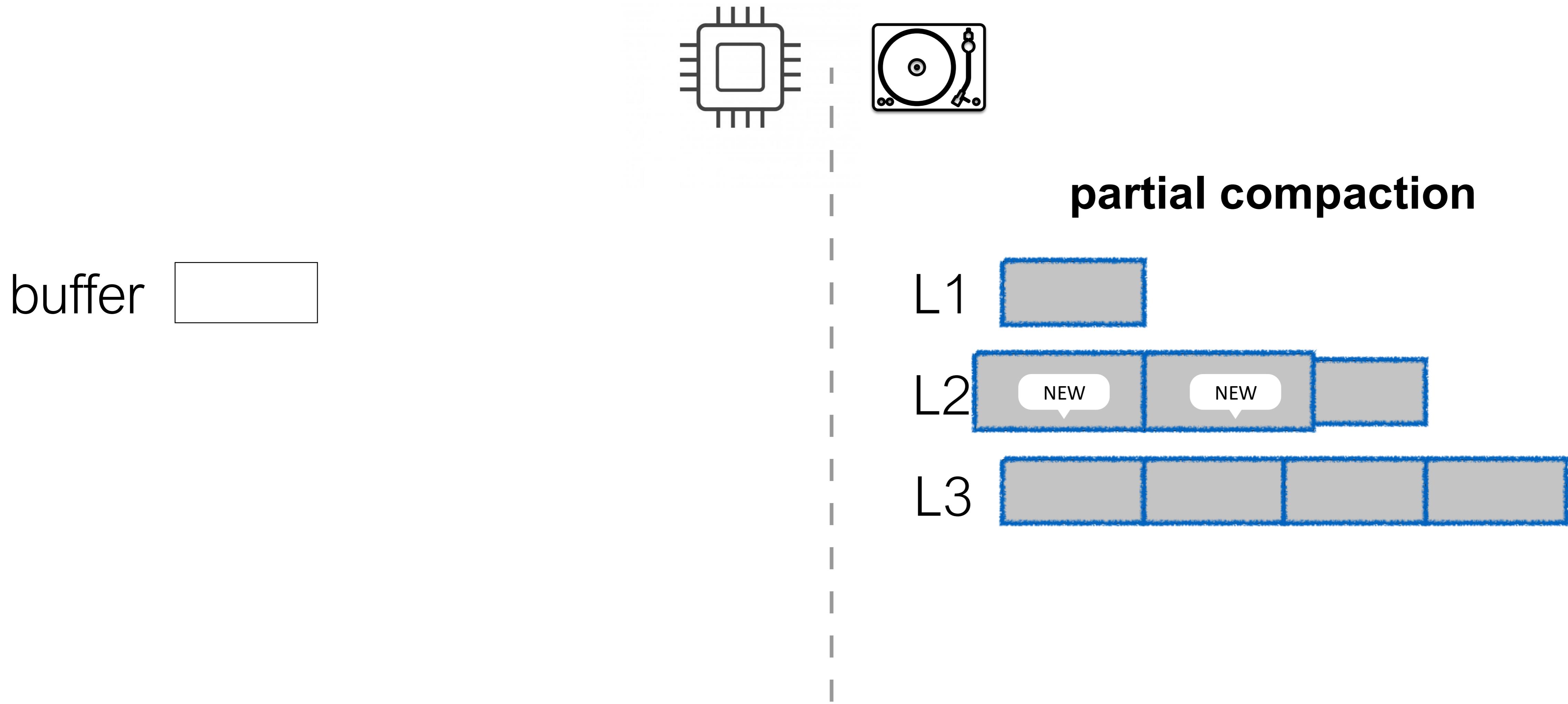
log-structured merge-tree



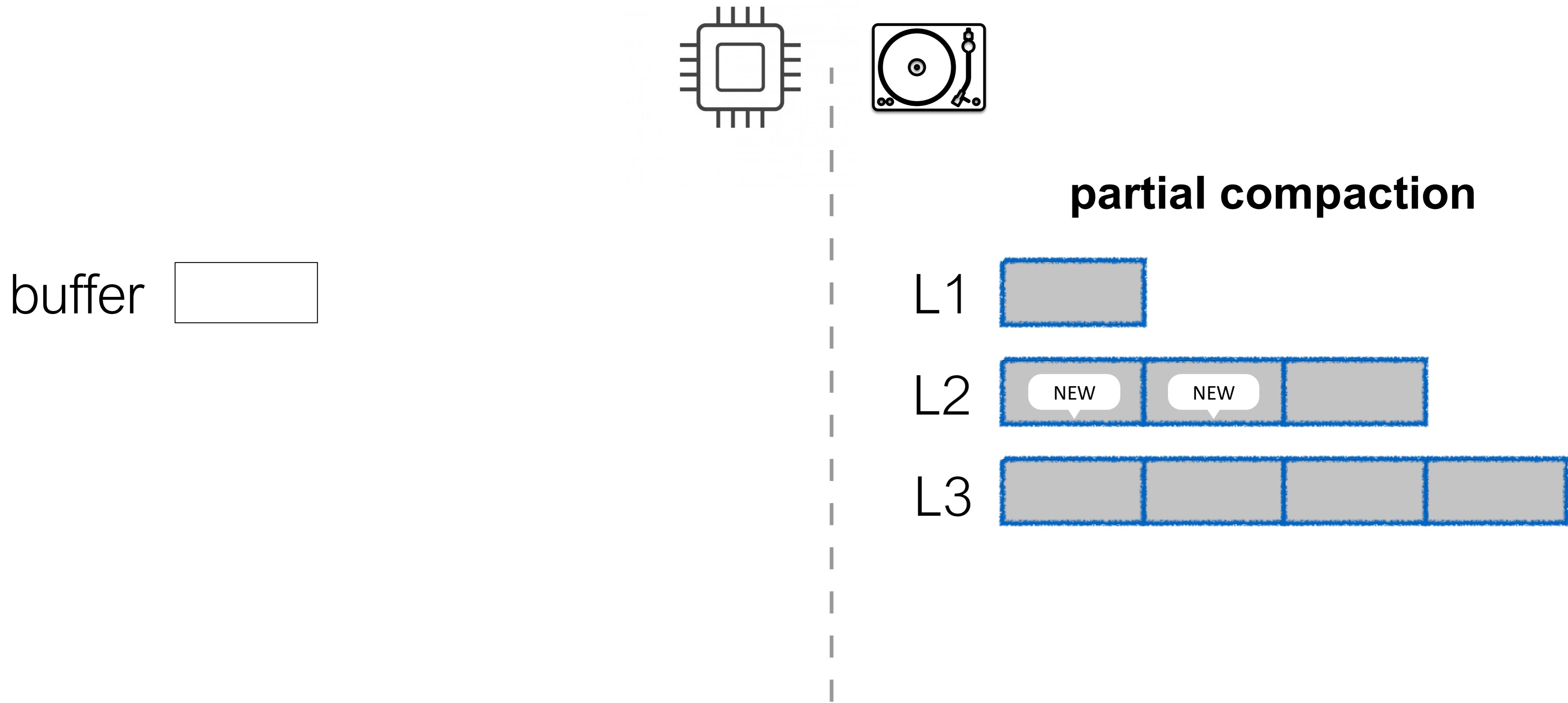
log-structured merge-tree



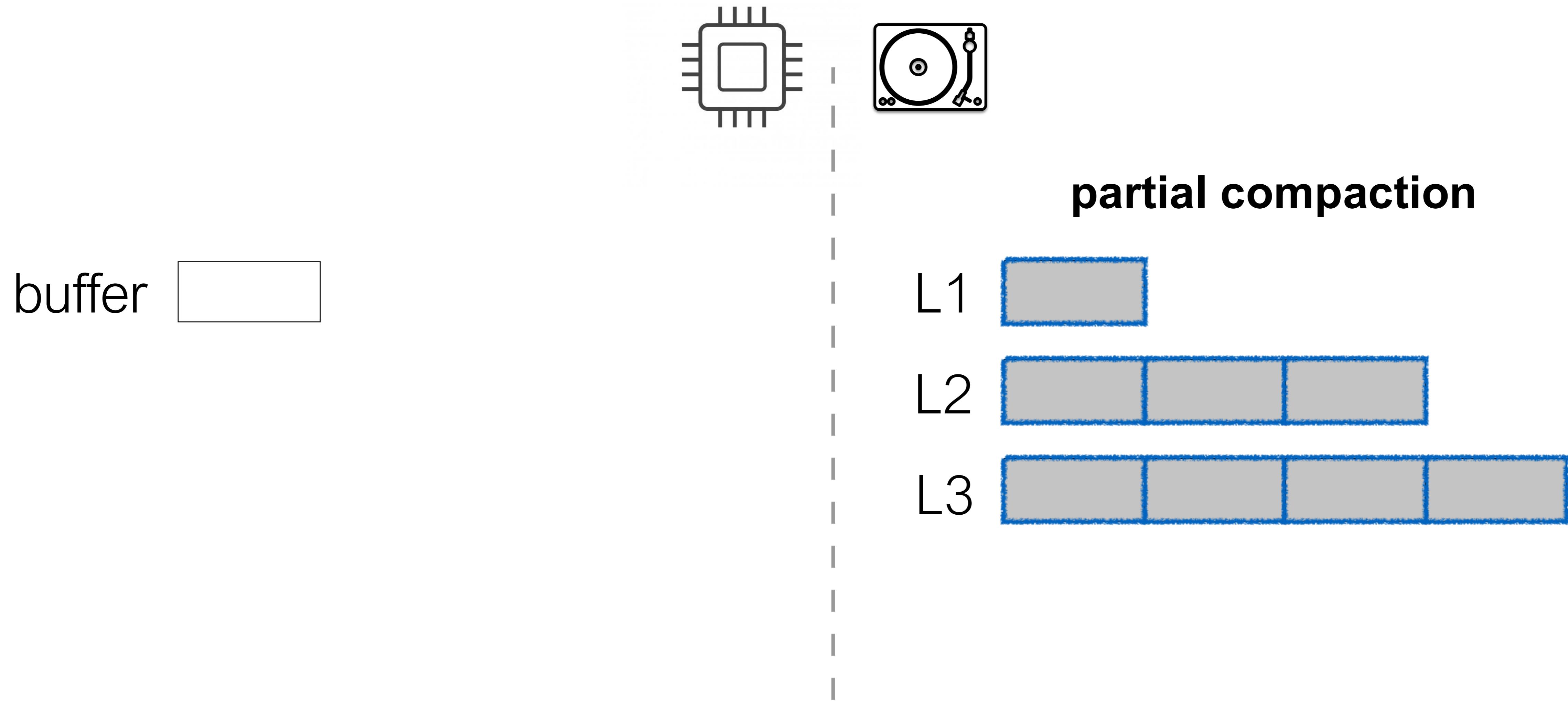
log-structured merge-tree



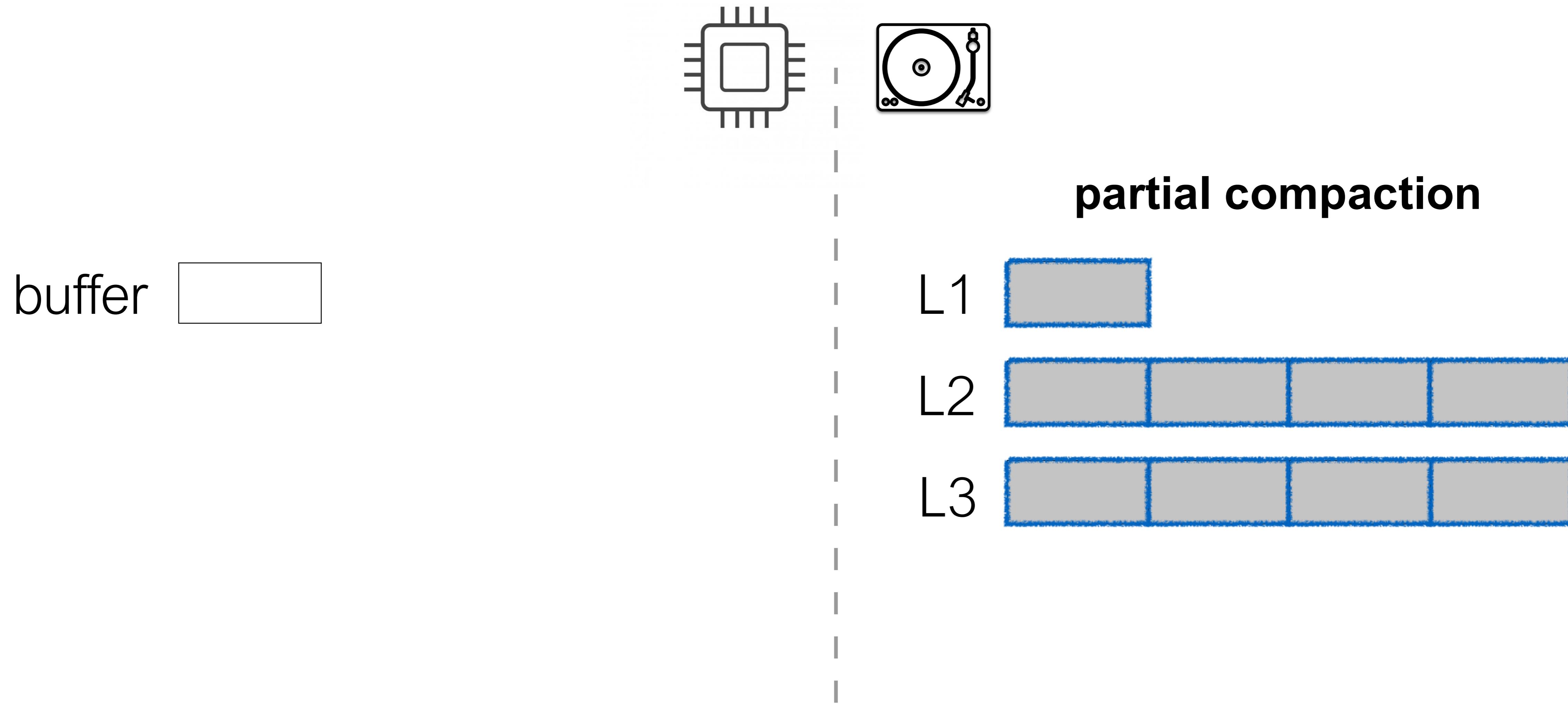
log-structured merge-tree



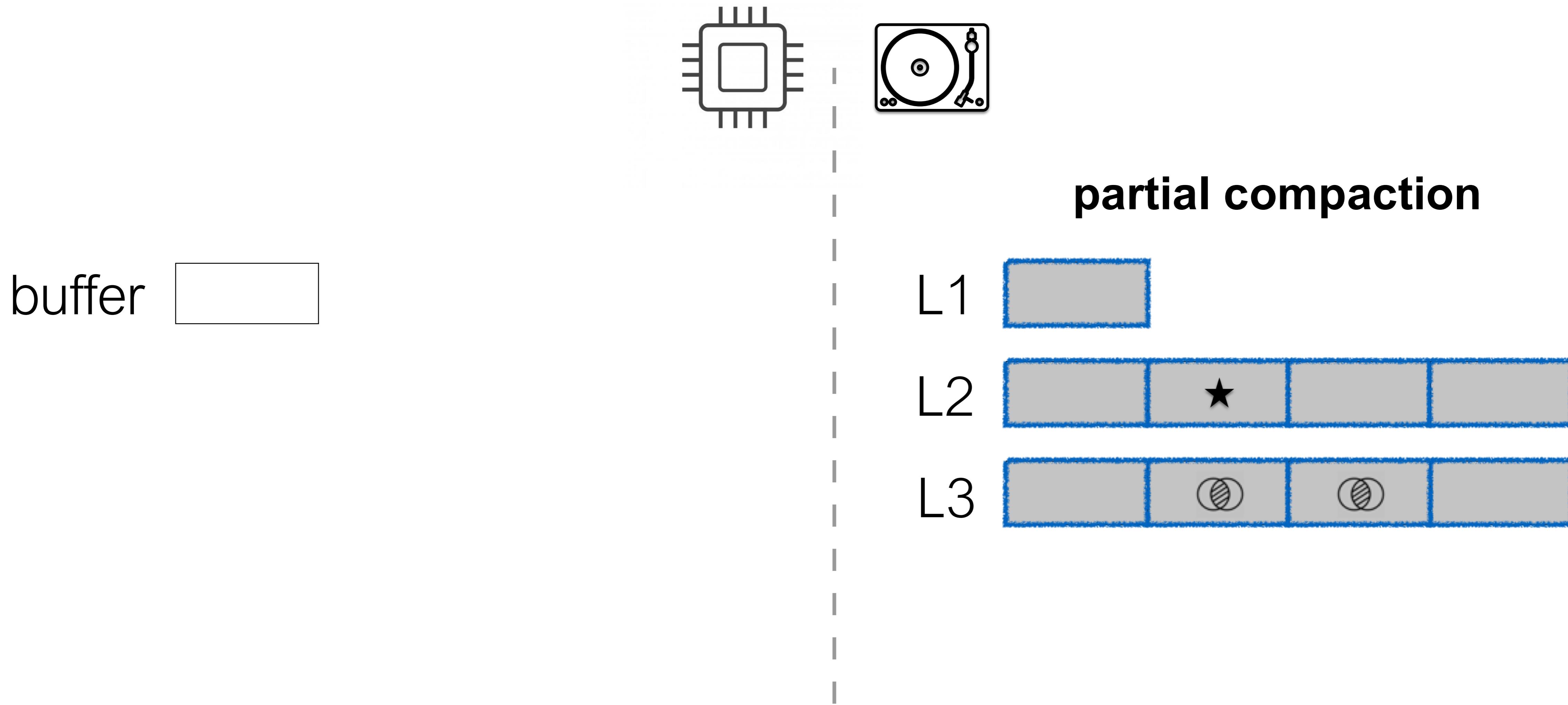
log-structured merge-tree



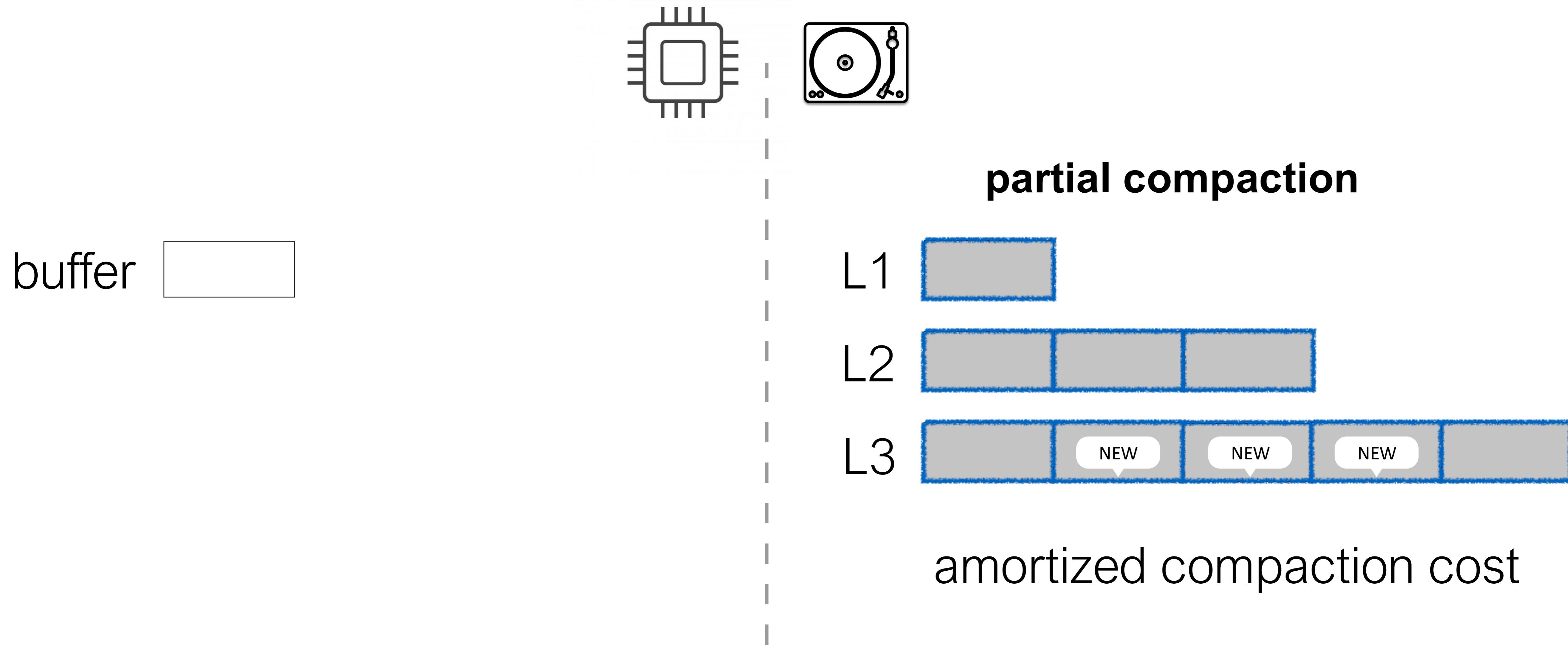
log-structured merge-tree



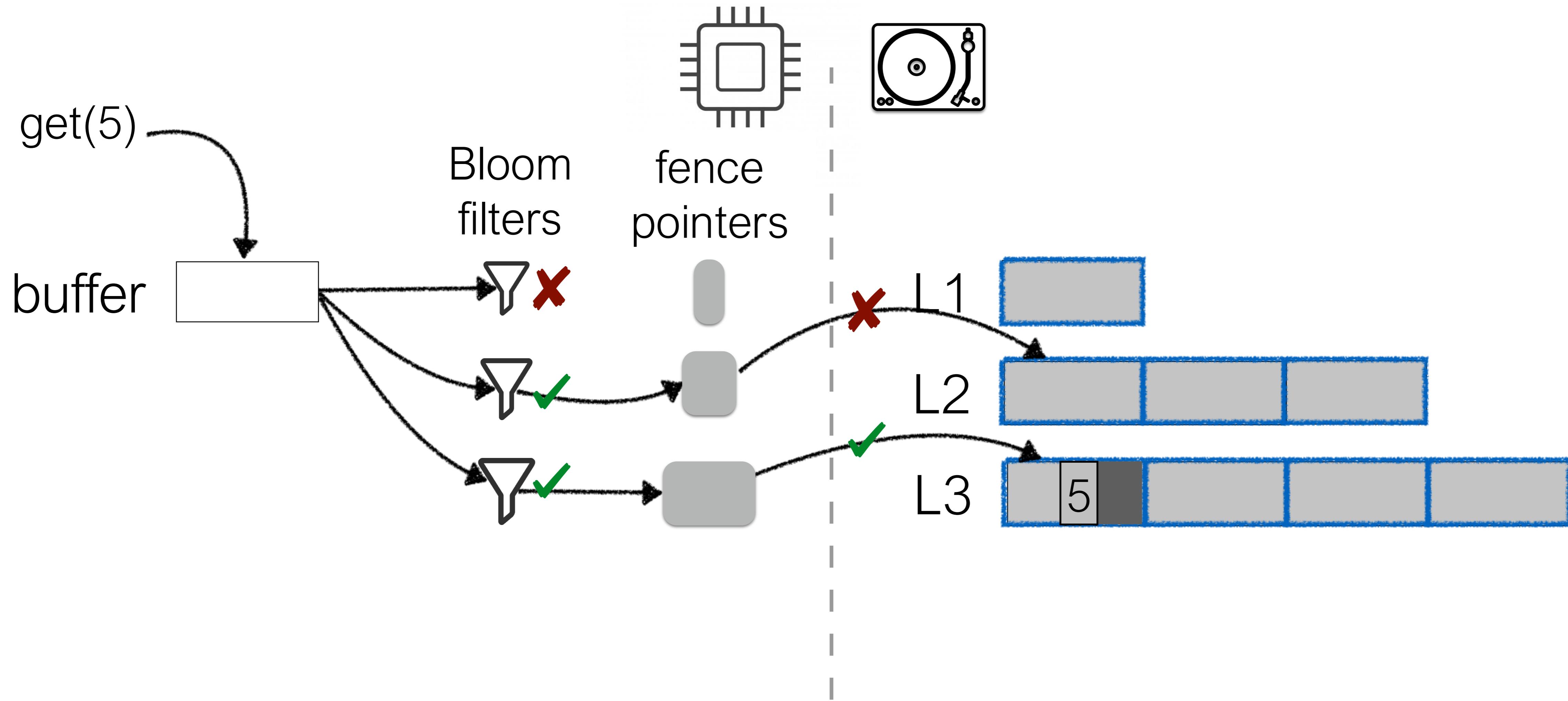
log-structured merge-tree



log-structured merge-tree



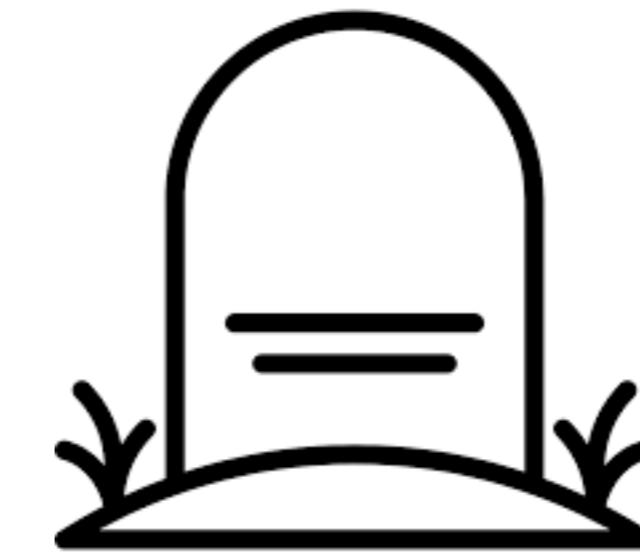
log-structured merge-tree



Now, let's talk about deletes!

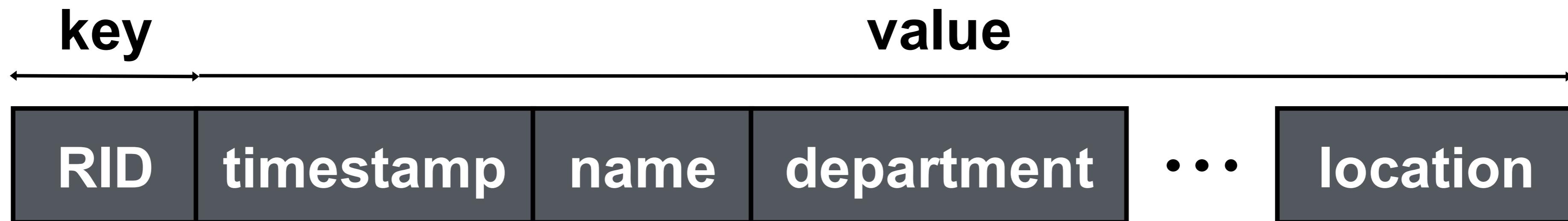
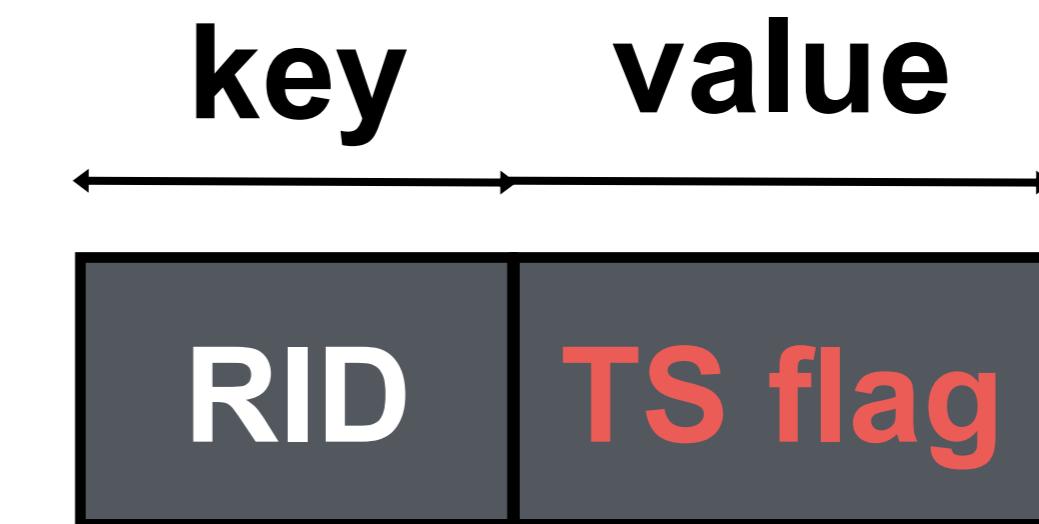
deletes in LSM-tree

delete



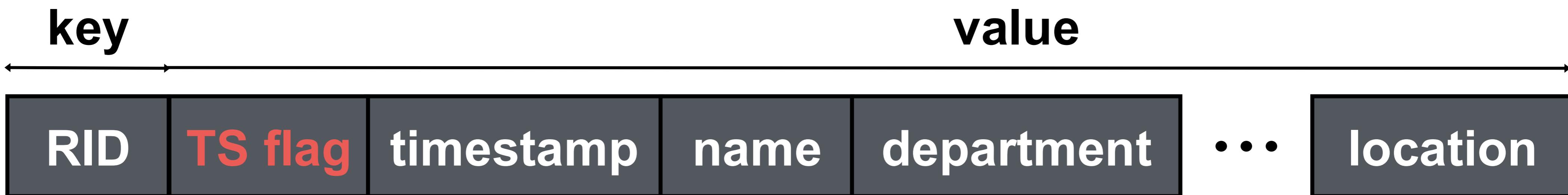
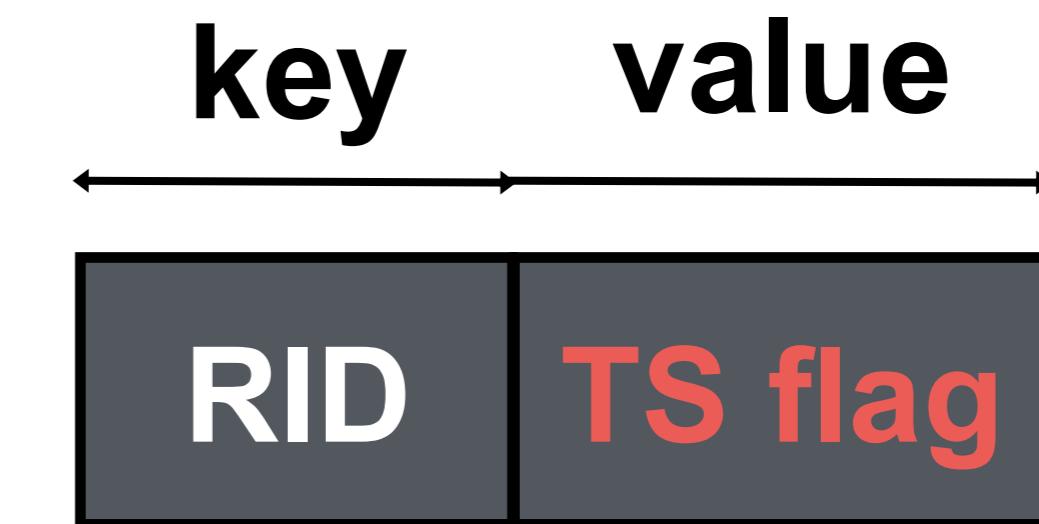
deletes in LSM-tree

delete := insert tombstone

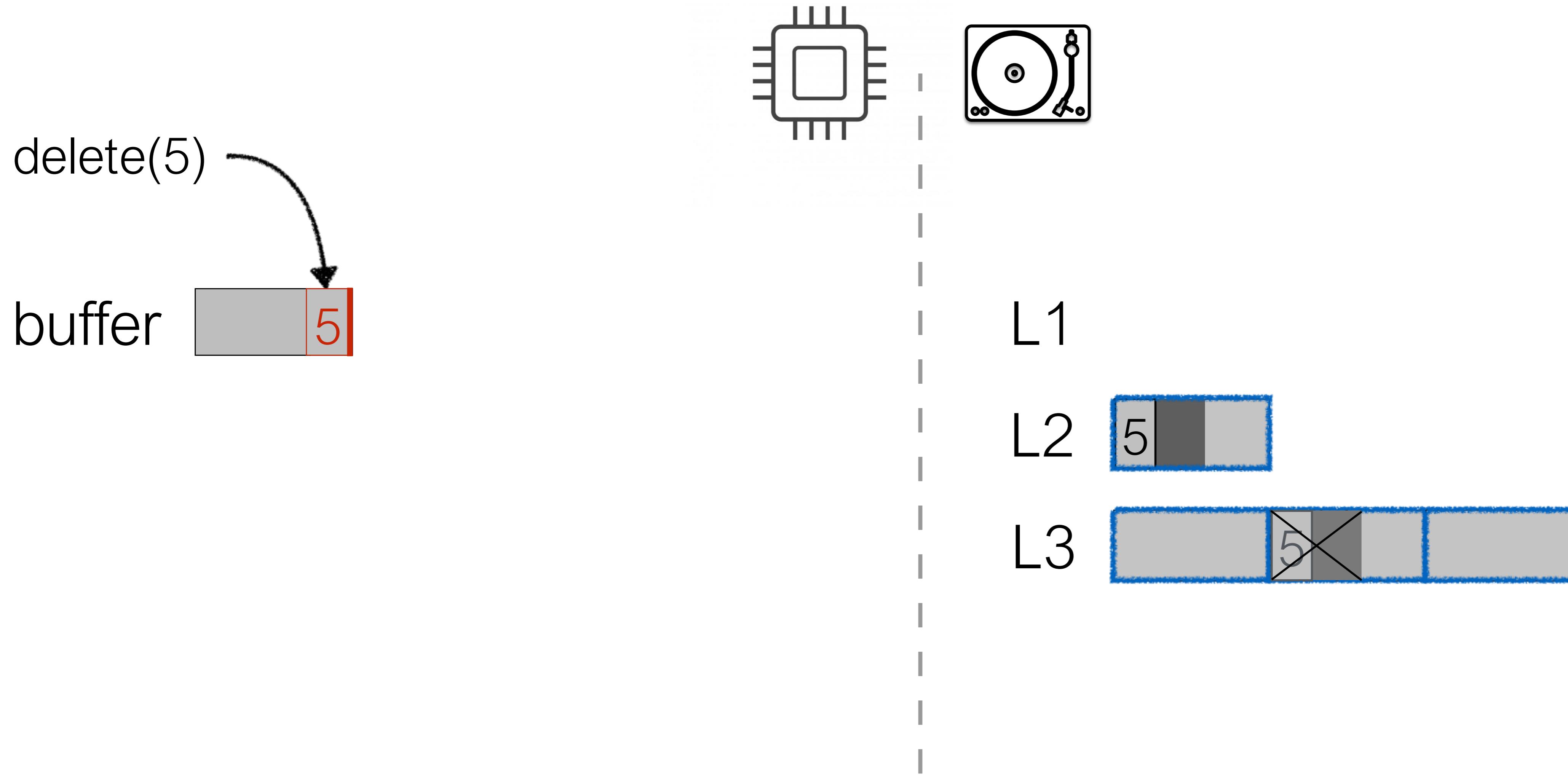


deletes in LSM-tree

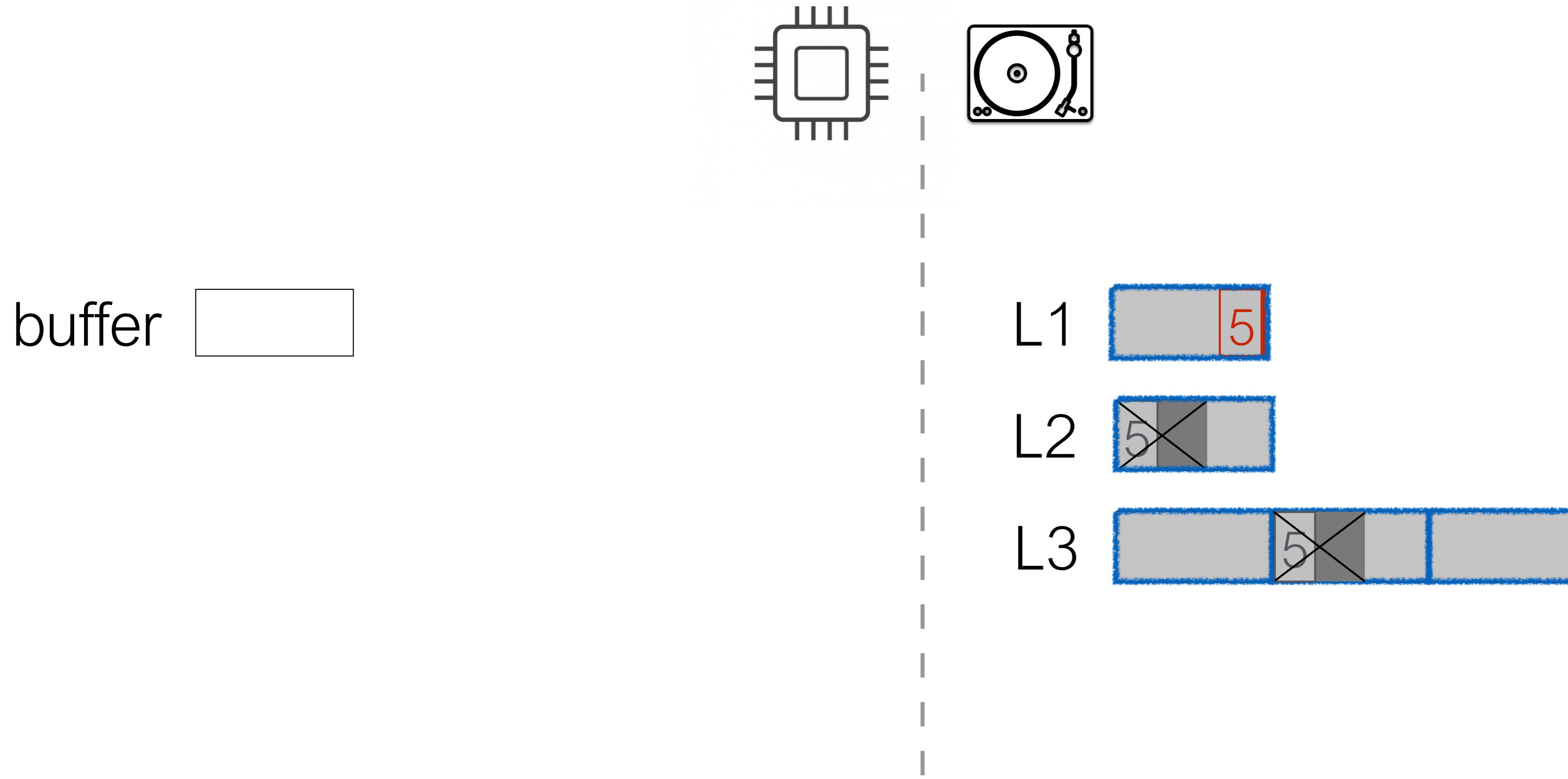
delete := insert tombstone



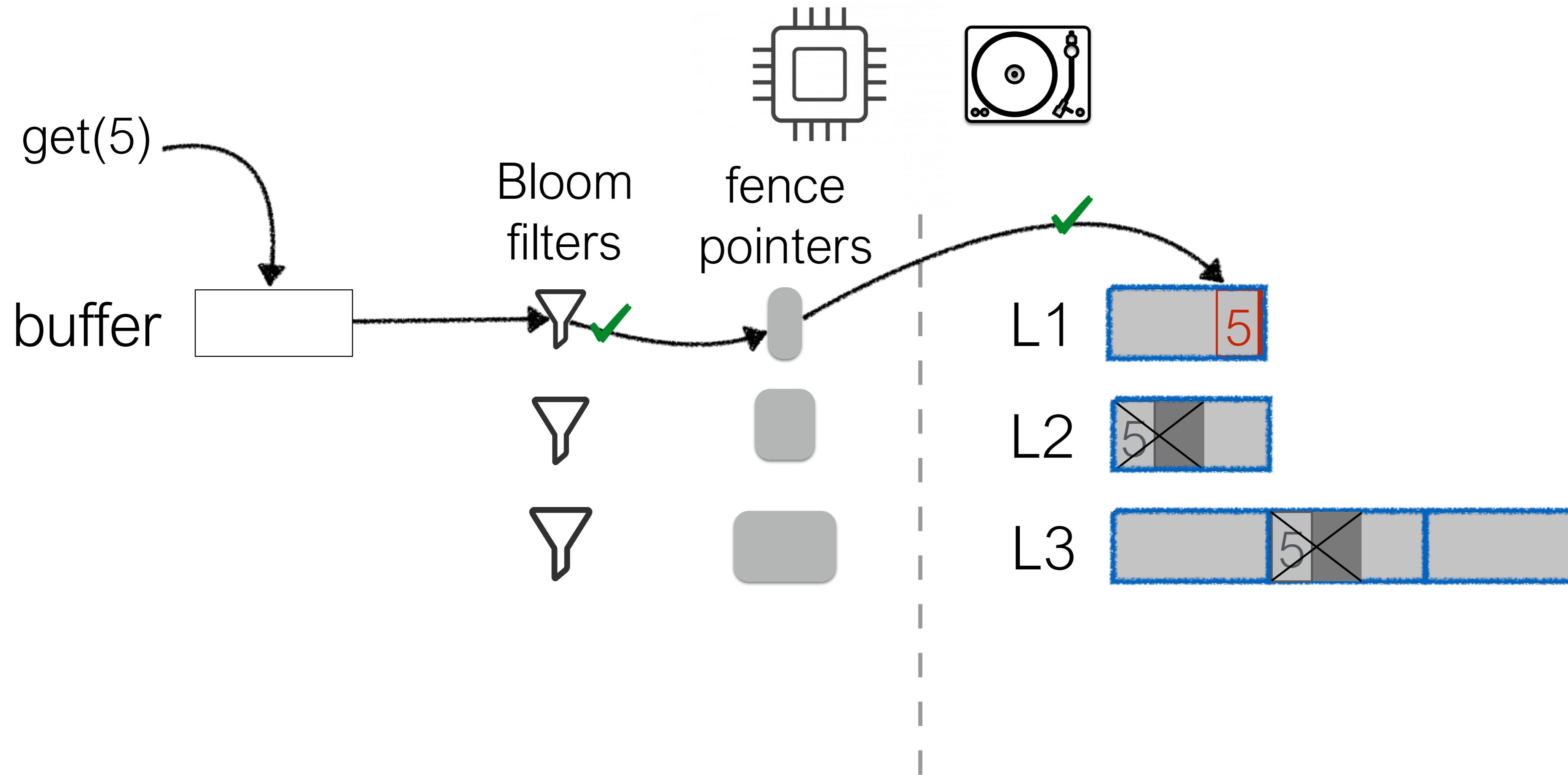
deletes in LSM-tree



deletes in LSM-tree



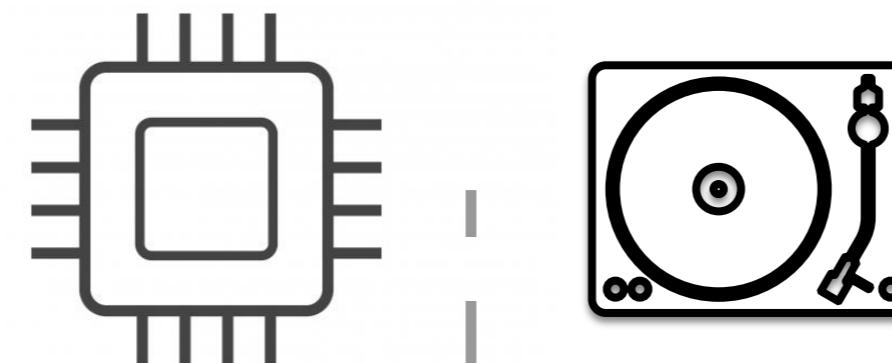
deletes in LSM-tree



range deletes in LSM-tree

delete(<3,7>)

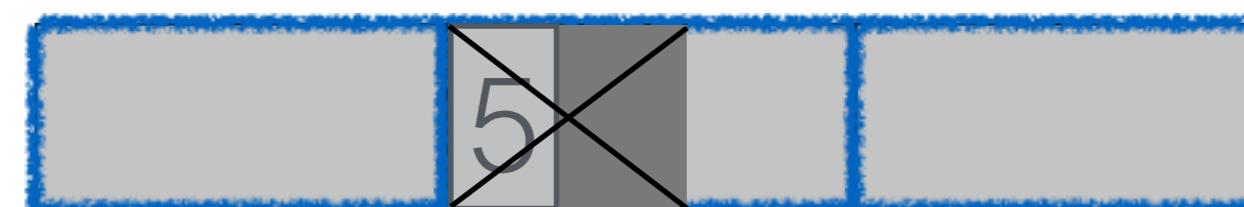
buffer



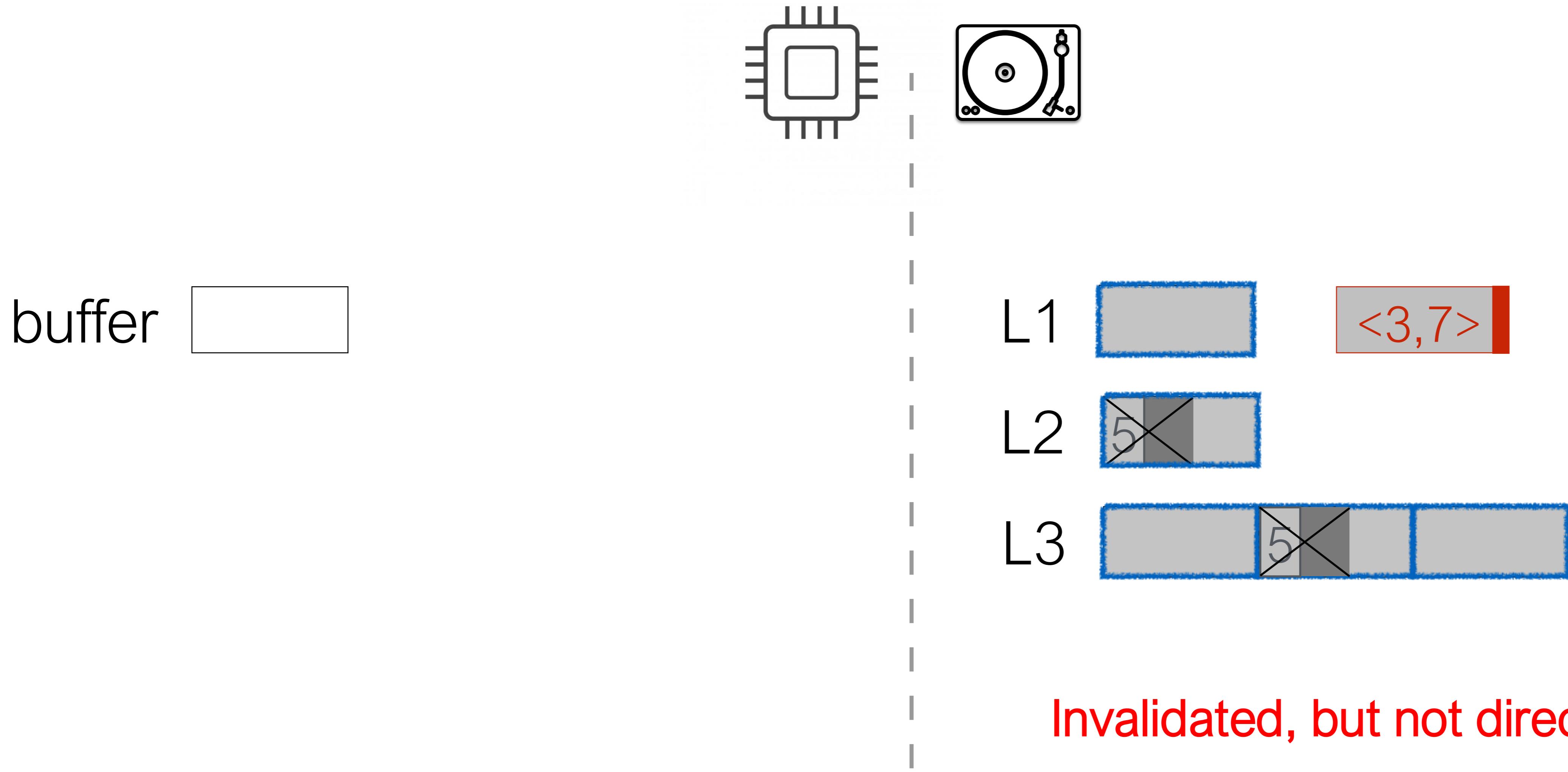
L1

L2

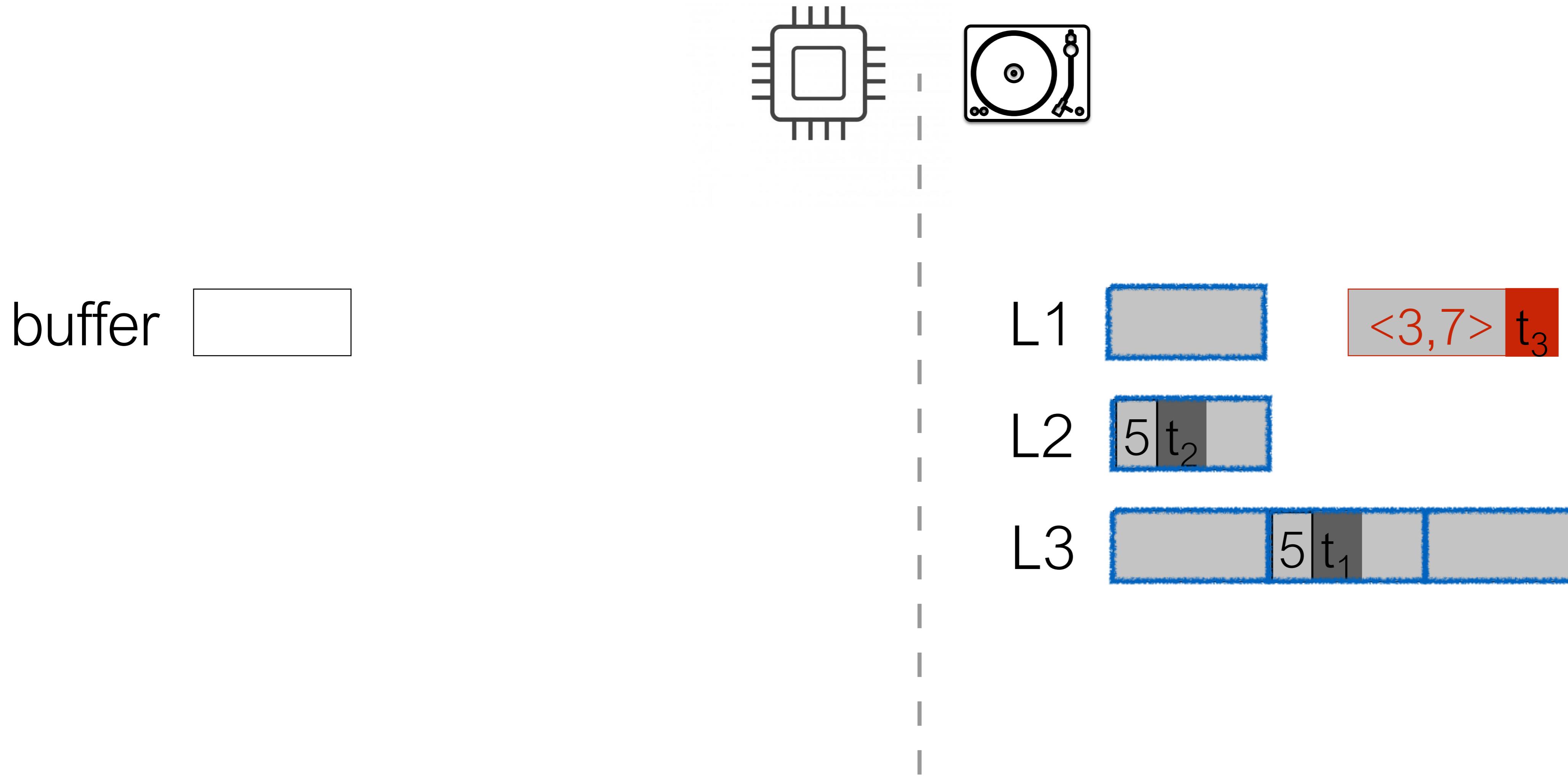
L3



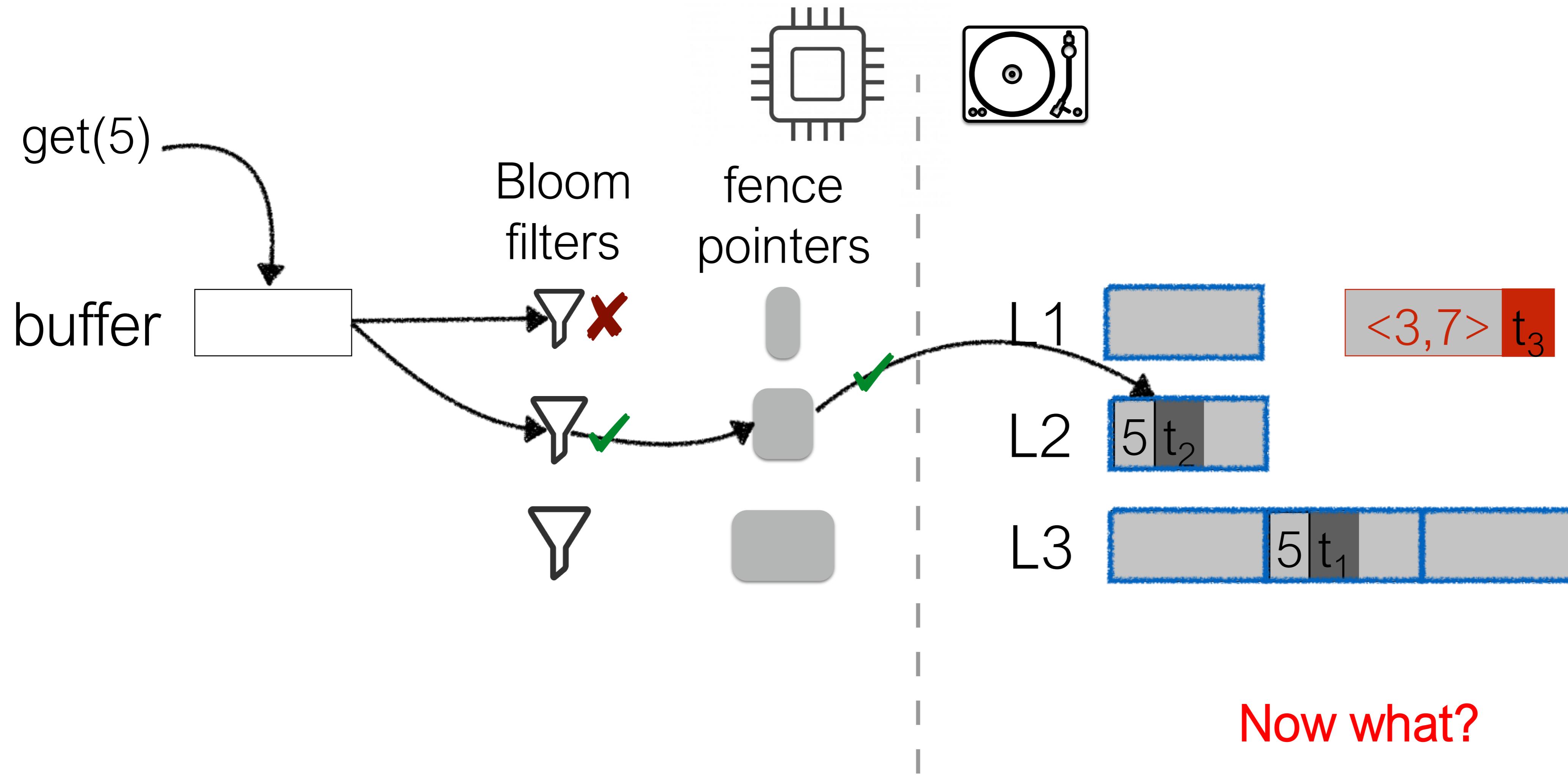
range deletes in LSM-tree



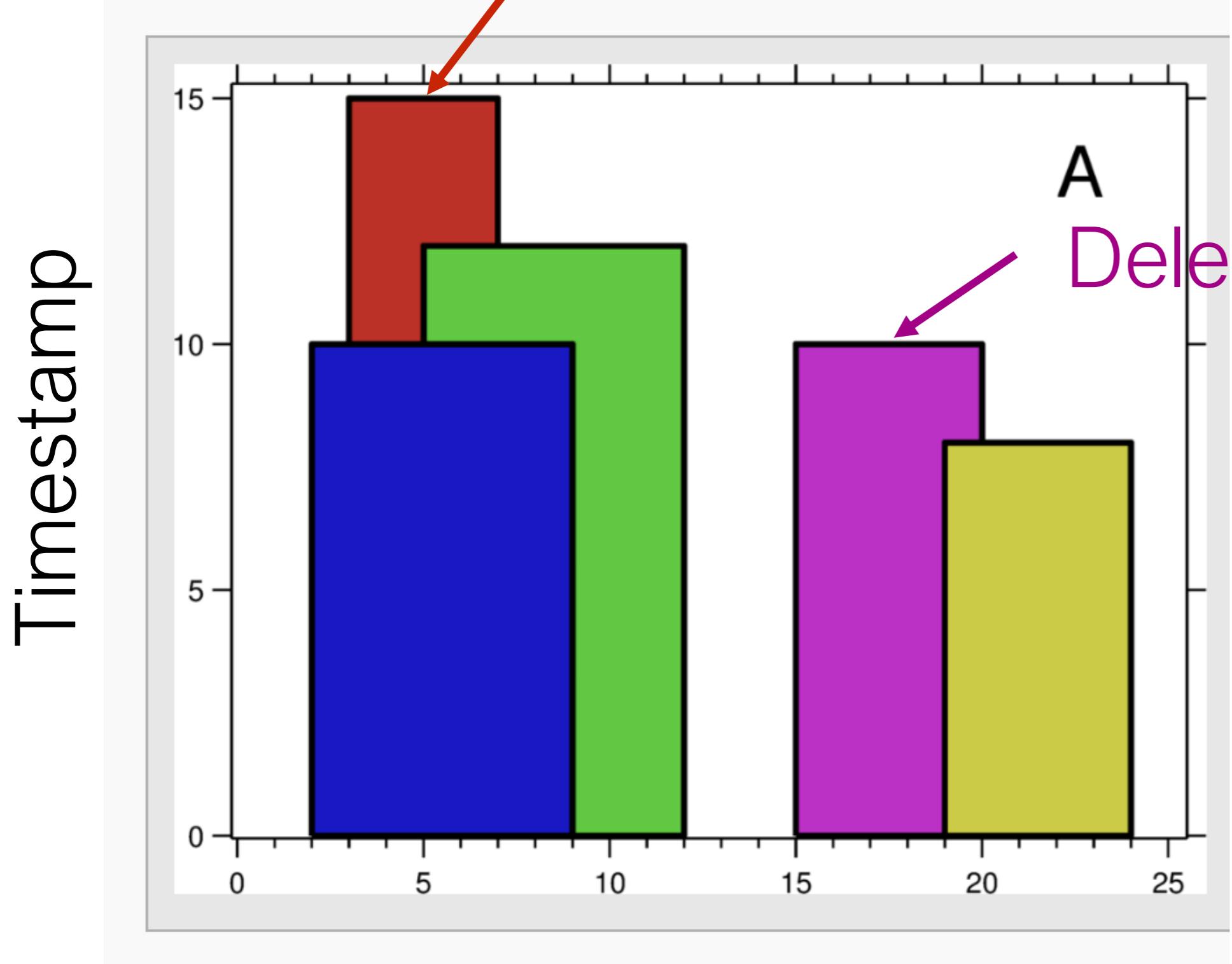
range deletes in LSM-tree



range deletes in LSM-tree



Delete <3,7> @ $t_3=15$



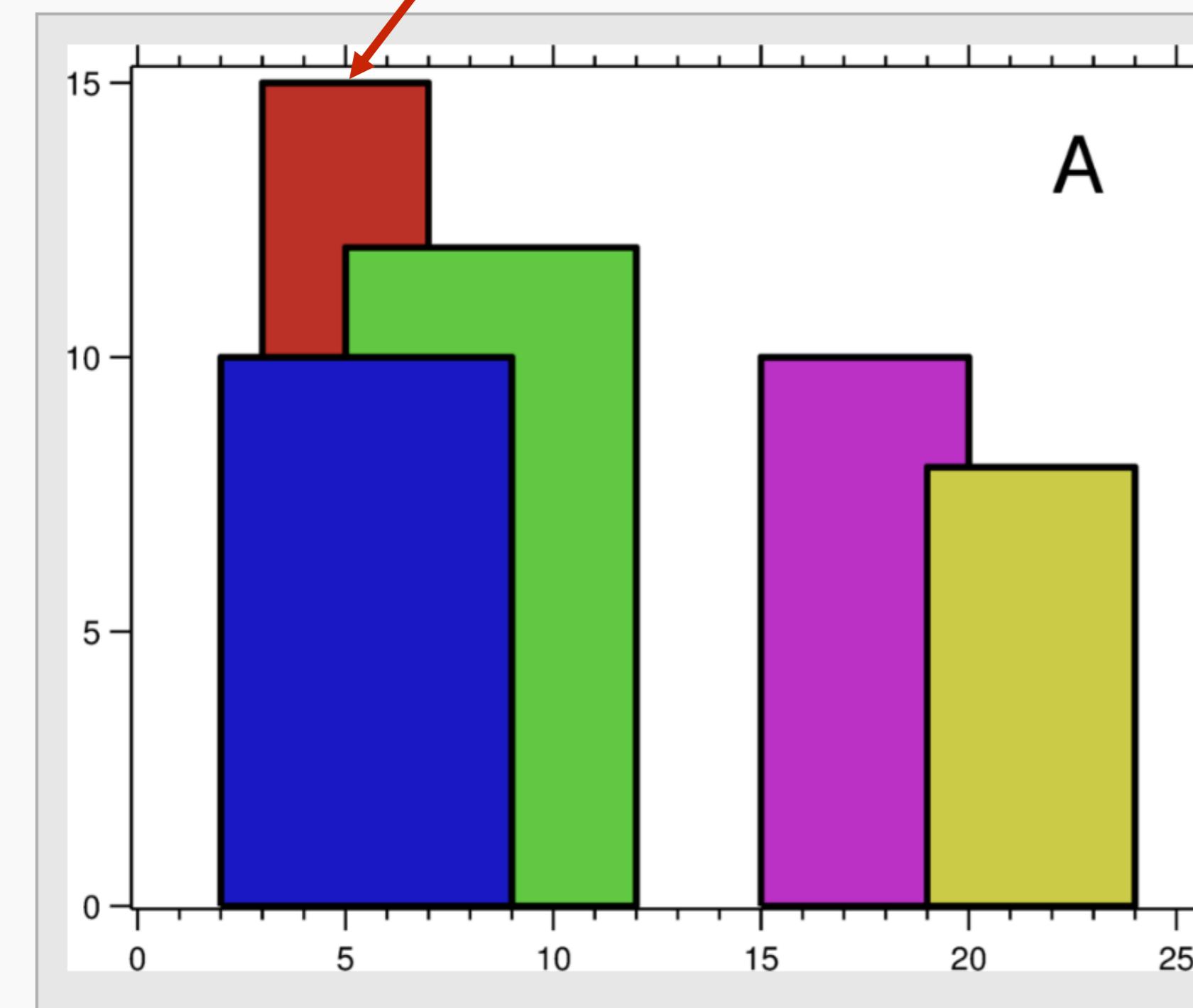
Delete Key

Is $\boxed{5}t_2$ valid?

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

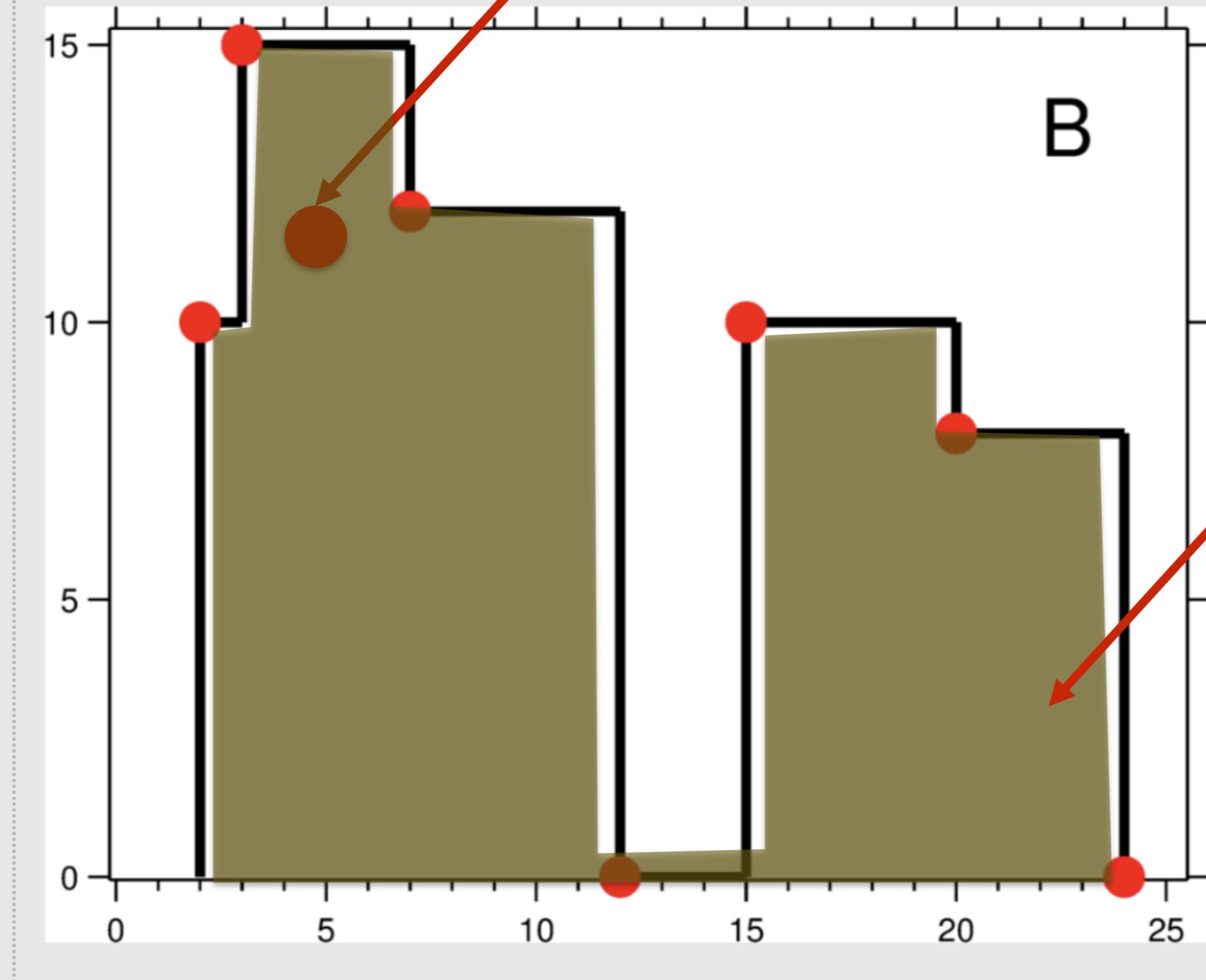
Delete $\langle 3, 7 \rangle$ @ $t_3=15$

Timestamp



Delete Key

$\langle 5, t_2 \rangle$ (for $t_2 < t_3$)



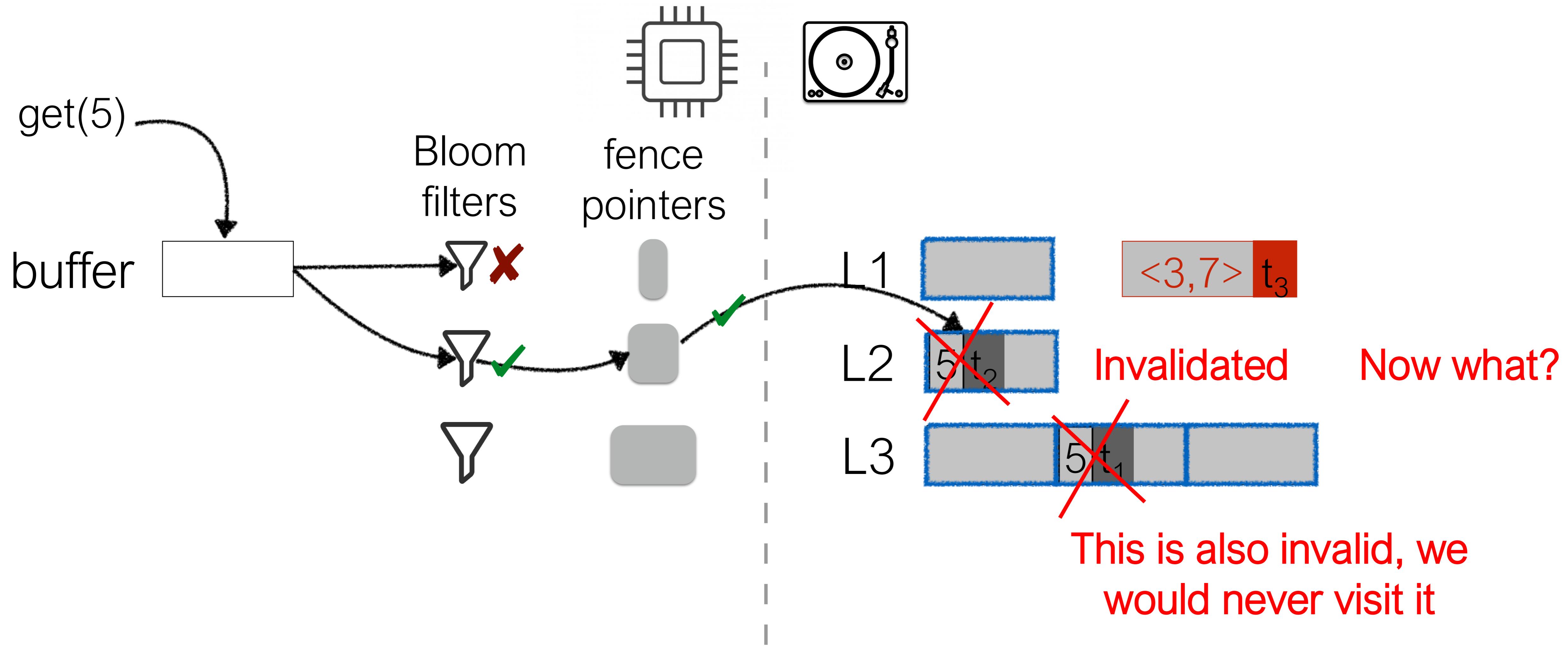
Delete Key

deleted entries

Is $\boxed{5}t_2$ 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

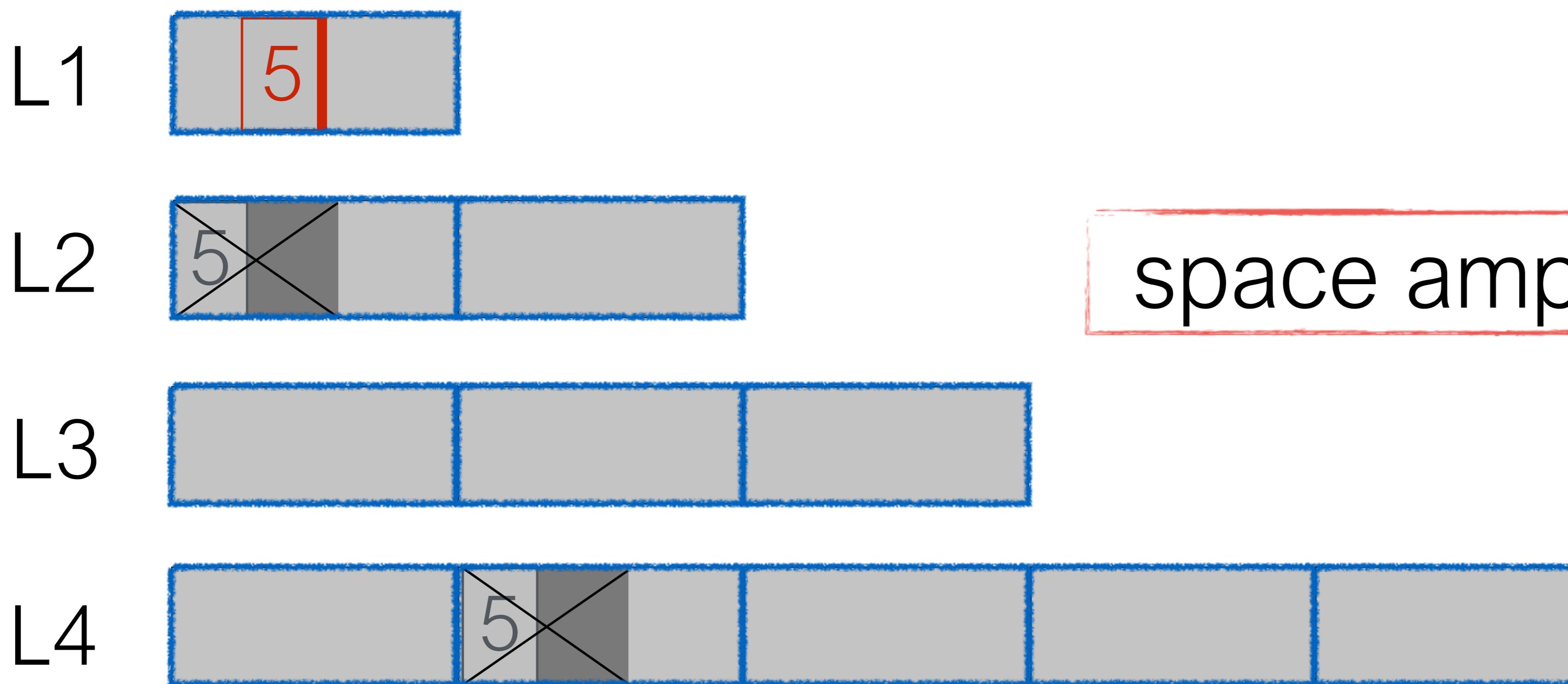


the problems



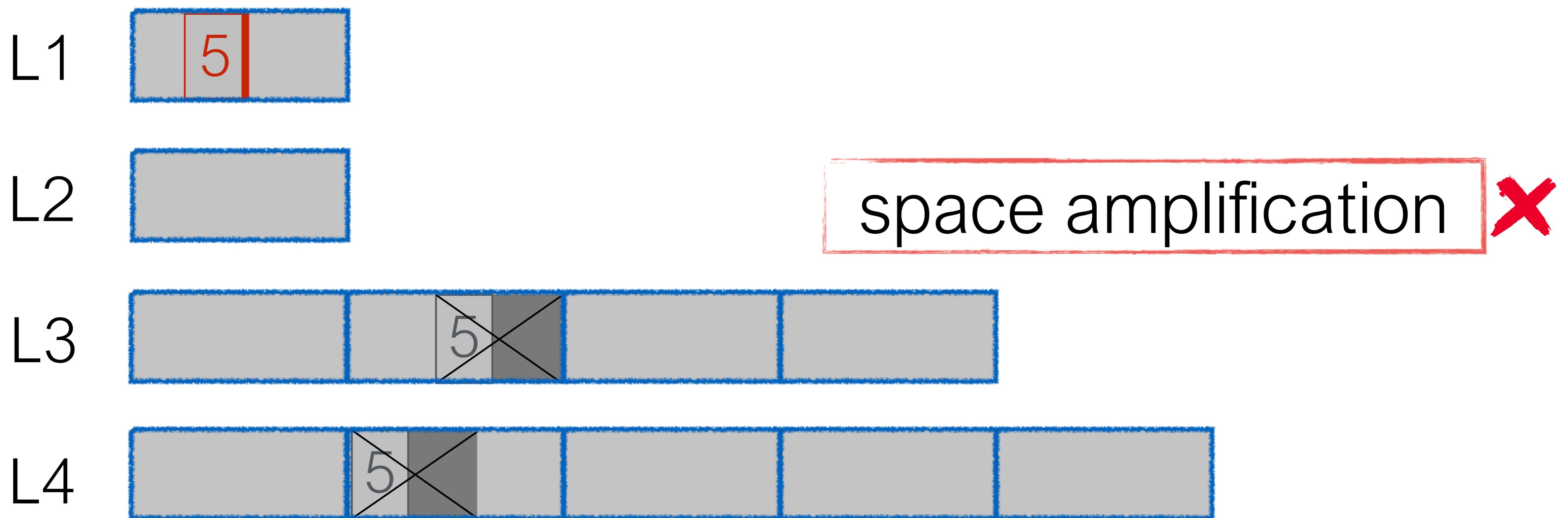
out-of-place deletes

out-of-place deletes

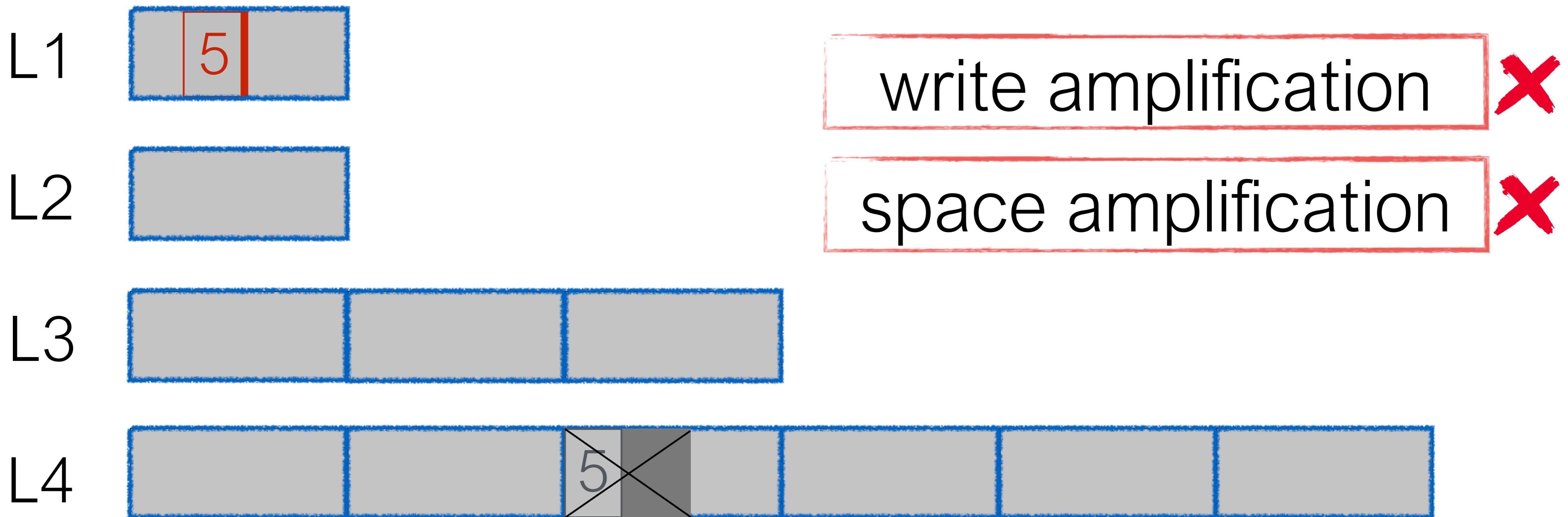


space amplification X

out-of-place deletes

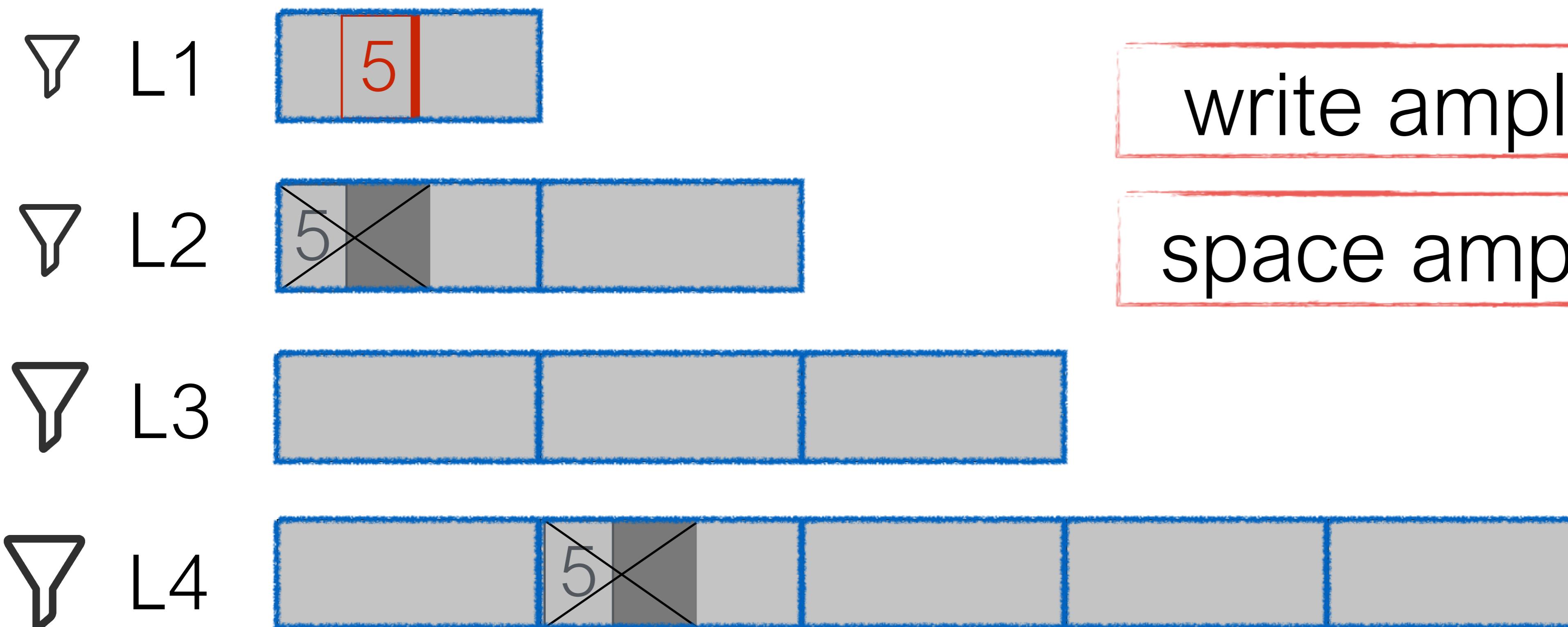


out-of-place deletes



out-of-place deletes

Bloom
filters



write amplification X

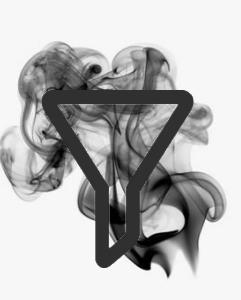
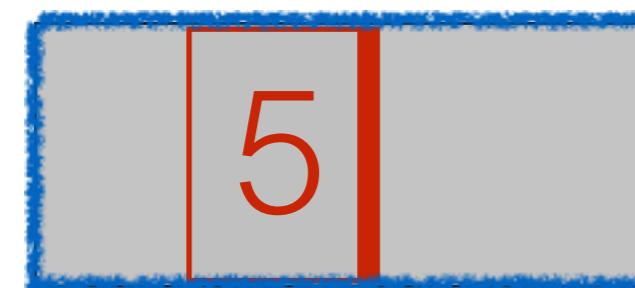
space amplification X

out-of-place deletes

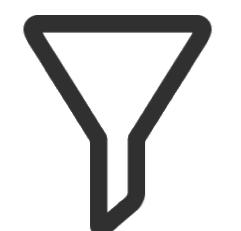
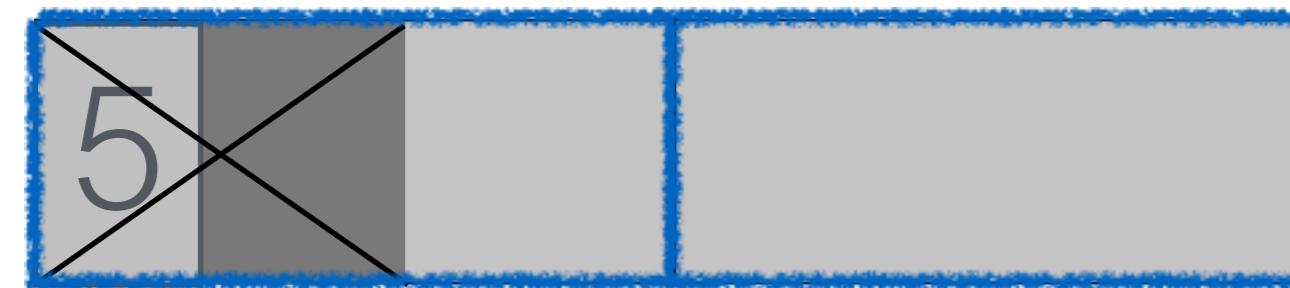
Bloom
filters



L1



L2



L3



L4



poor read perf. X

write amplification X

space amplification X

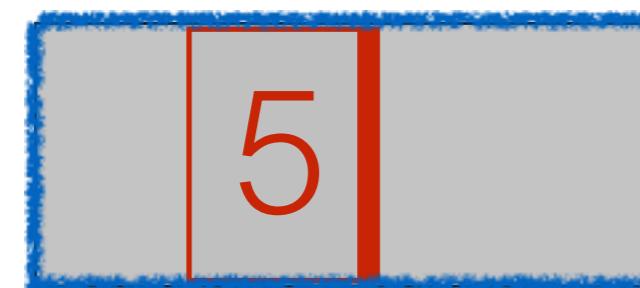


out-of-place deletes

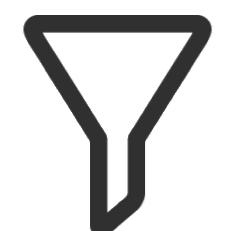
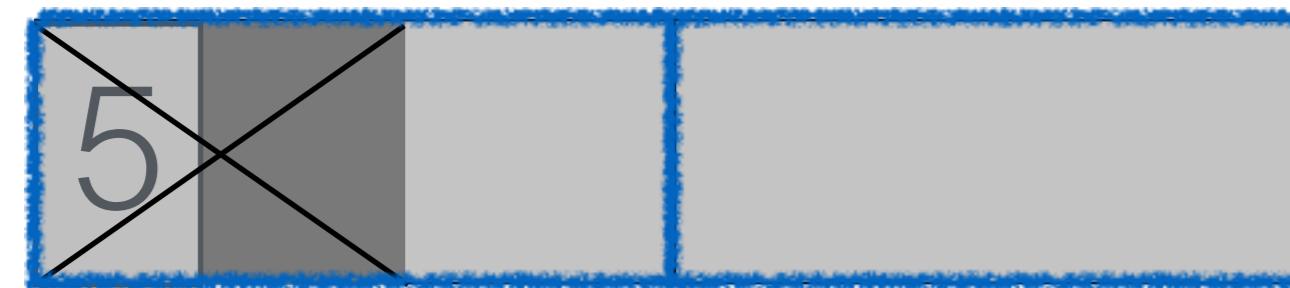
Bloom
filters



L1



L2



L3



L4



poor read perf. X

write amplification X

space amplification X

X

X

X

the problems

poor read perf.

write amplification

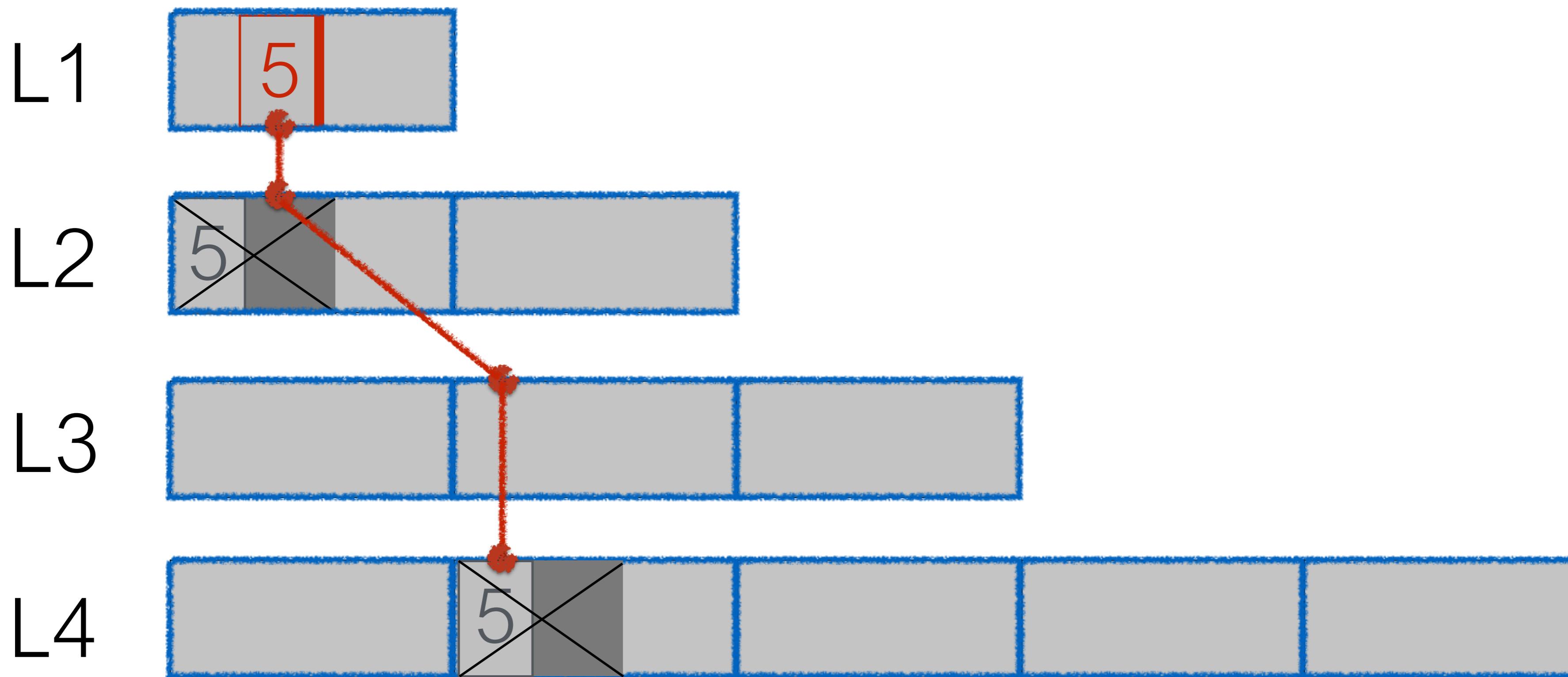
space amplification





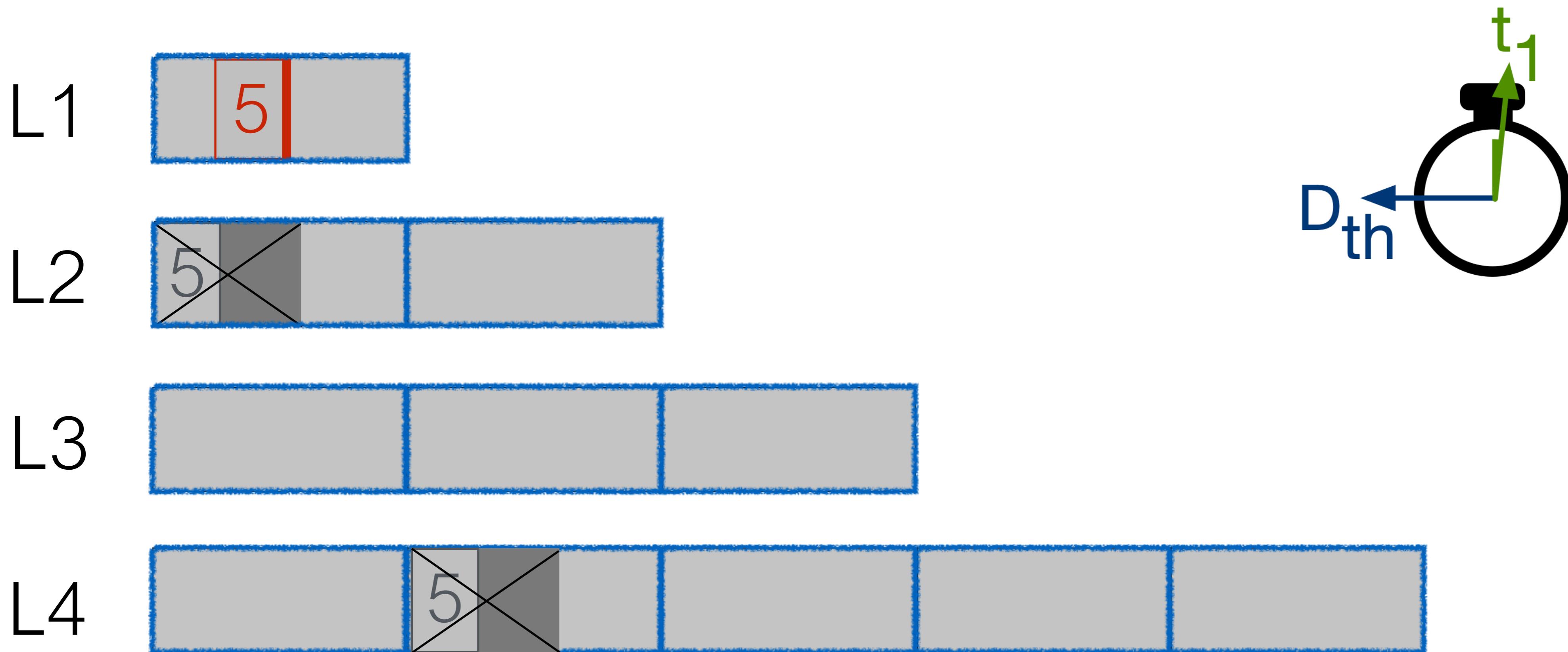
delete persistence latency

delete persistence latency



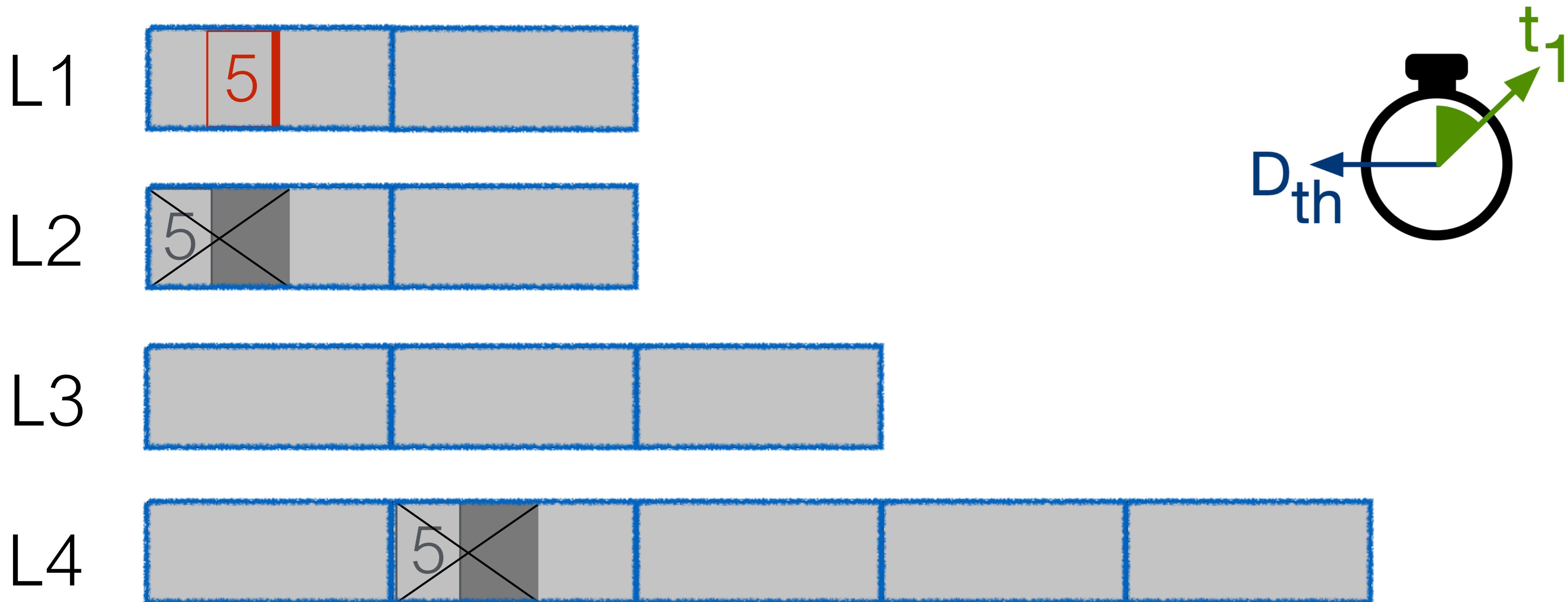
delete persistence latency

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



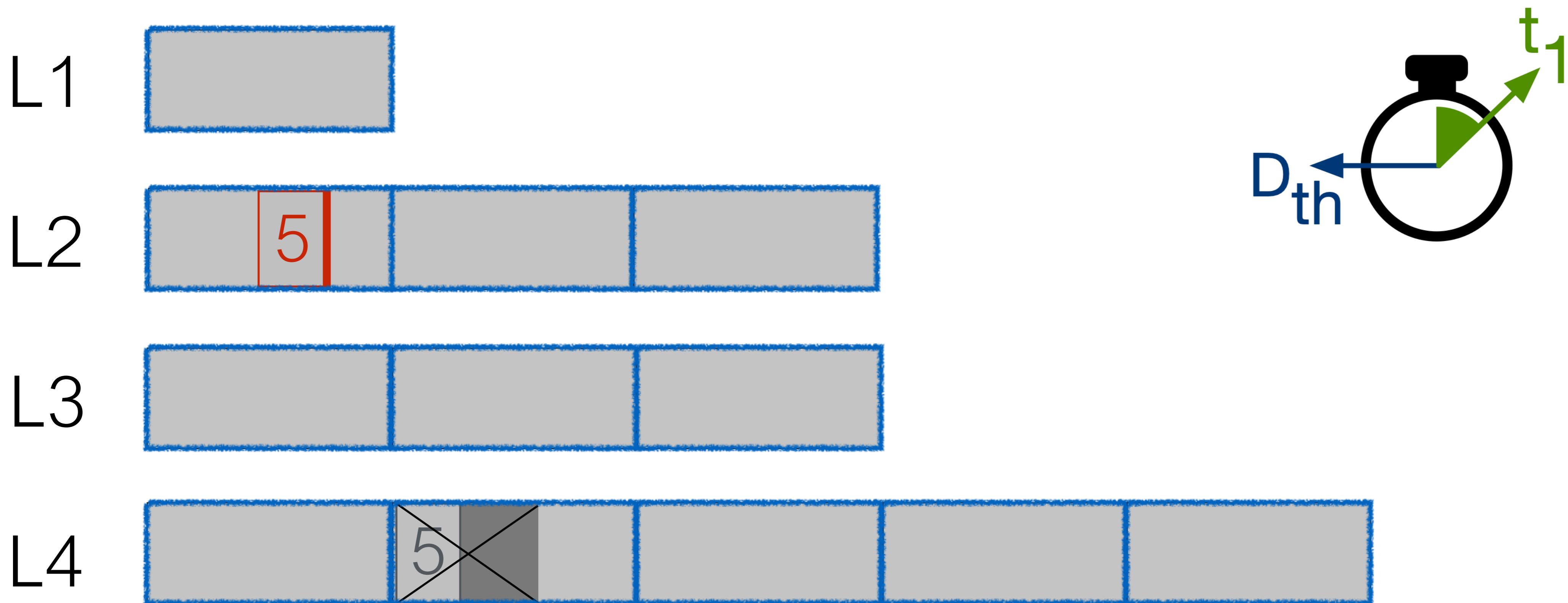
delete persistence latency

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



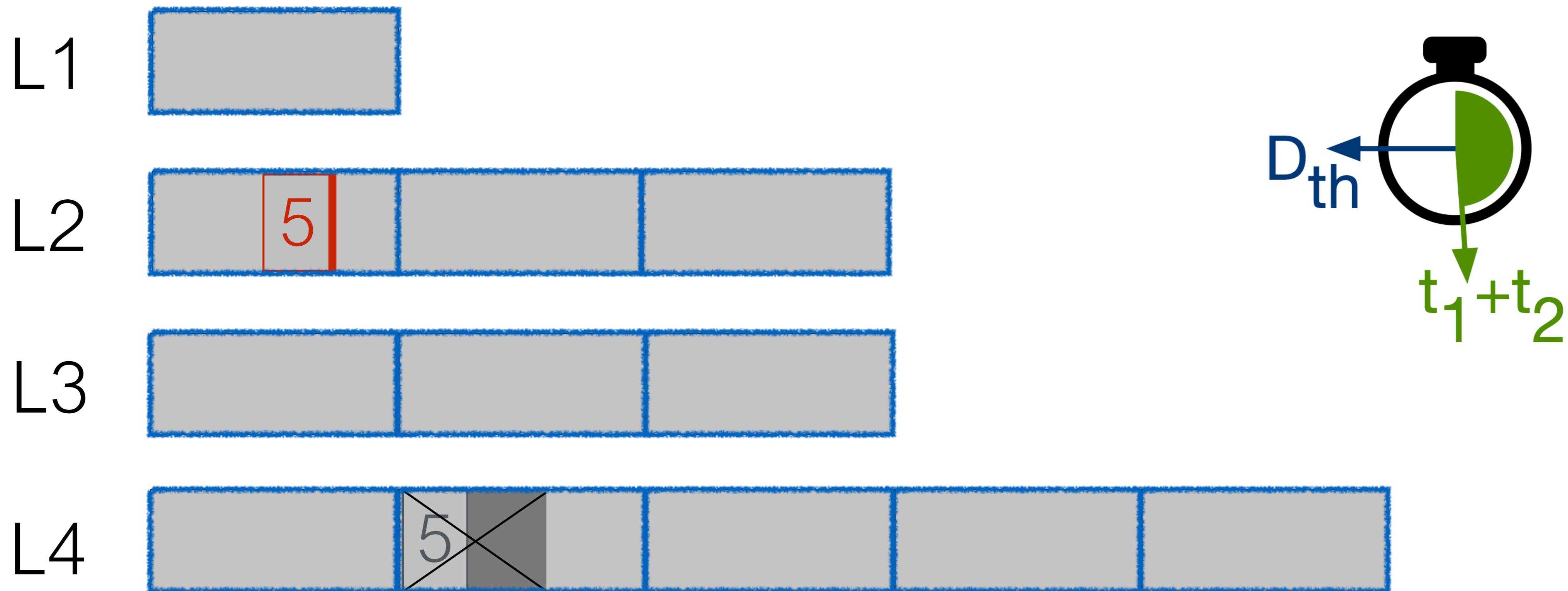
delete persistence latency

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



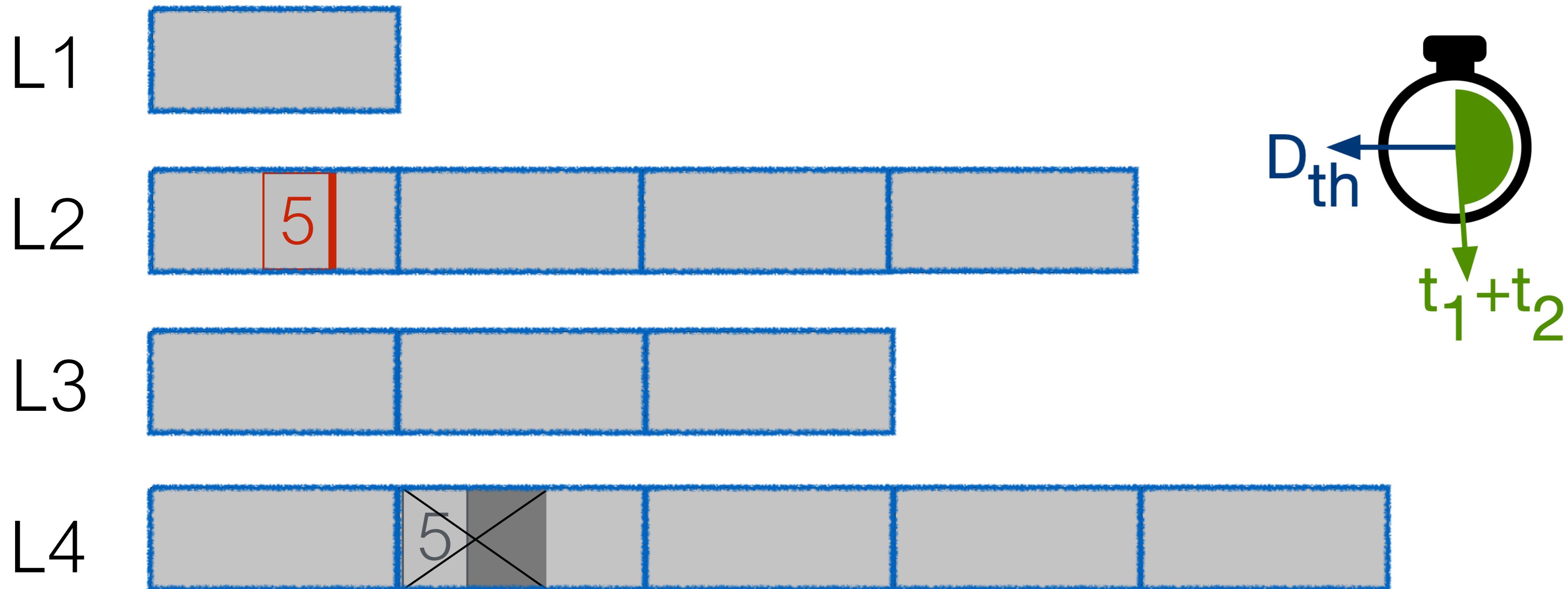
delete persistence latency

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



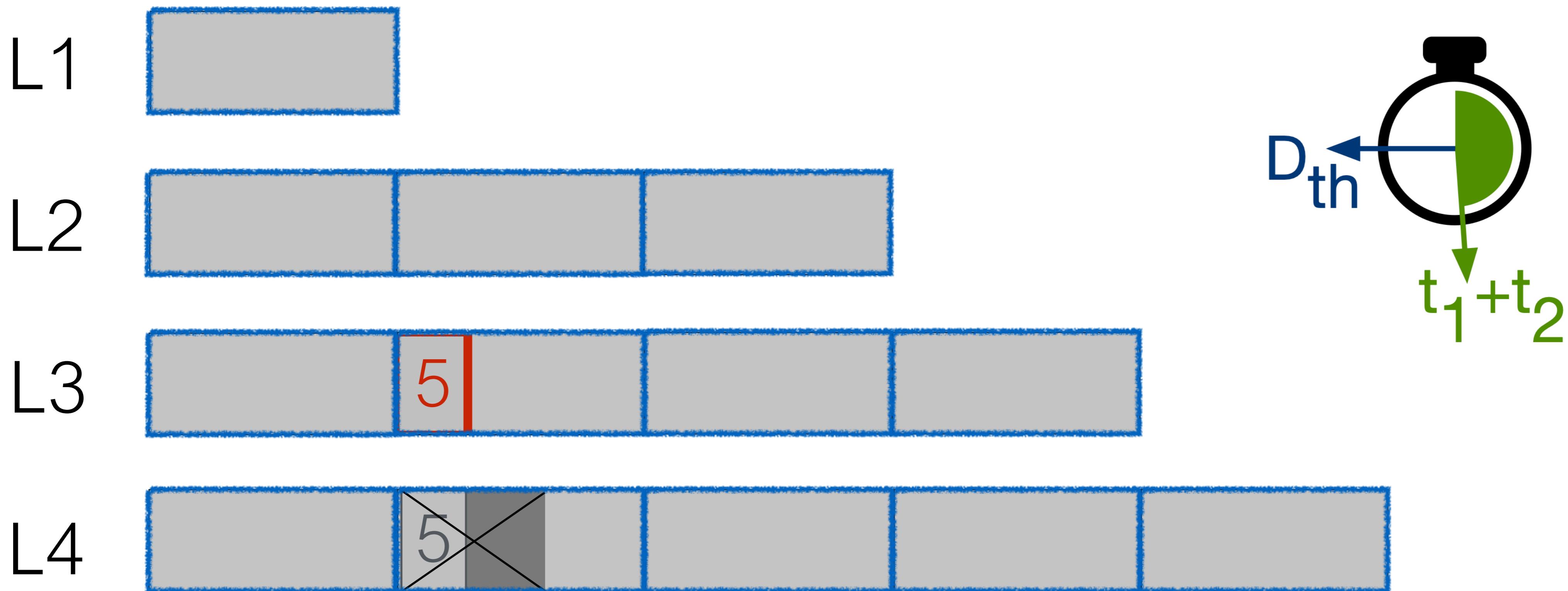
delete persistence latency

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



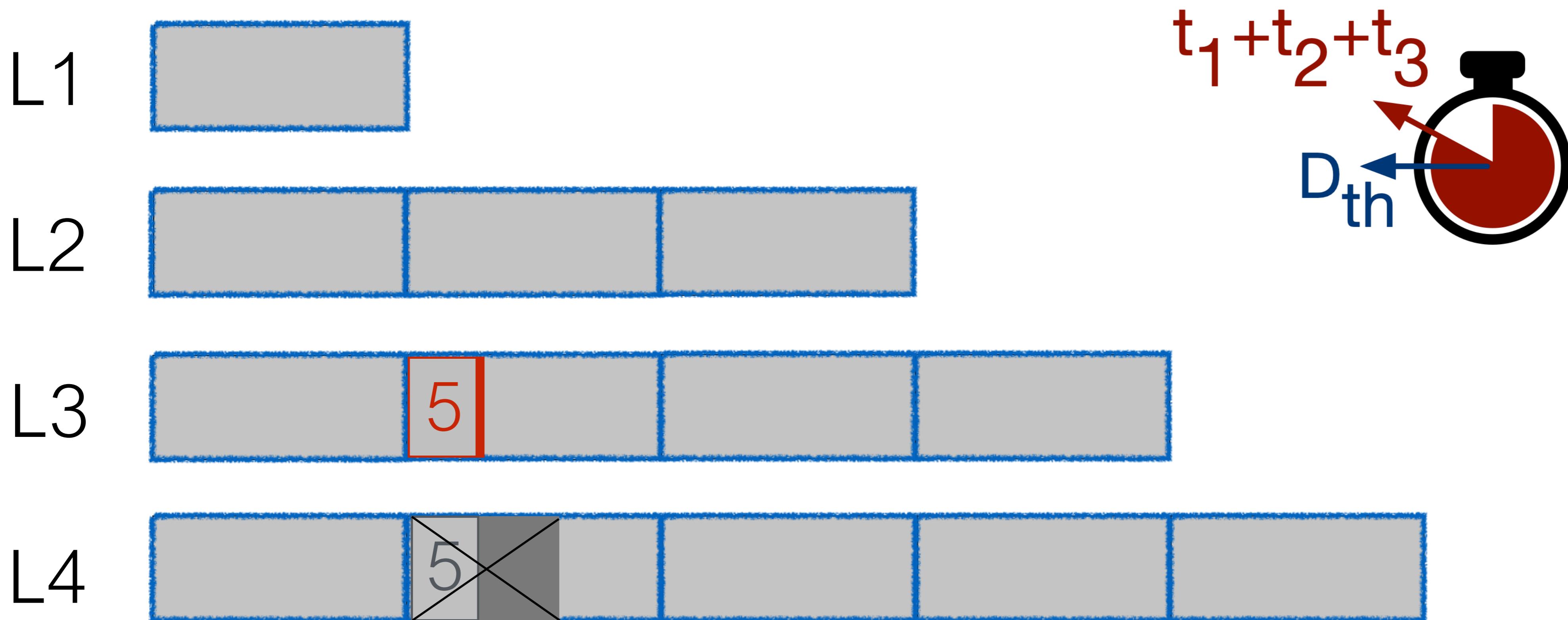
delete persistence latency

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



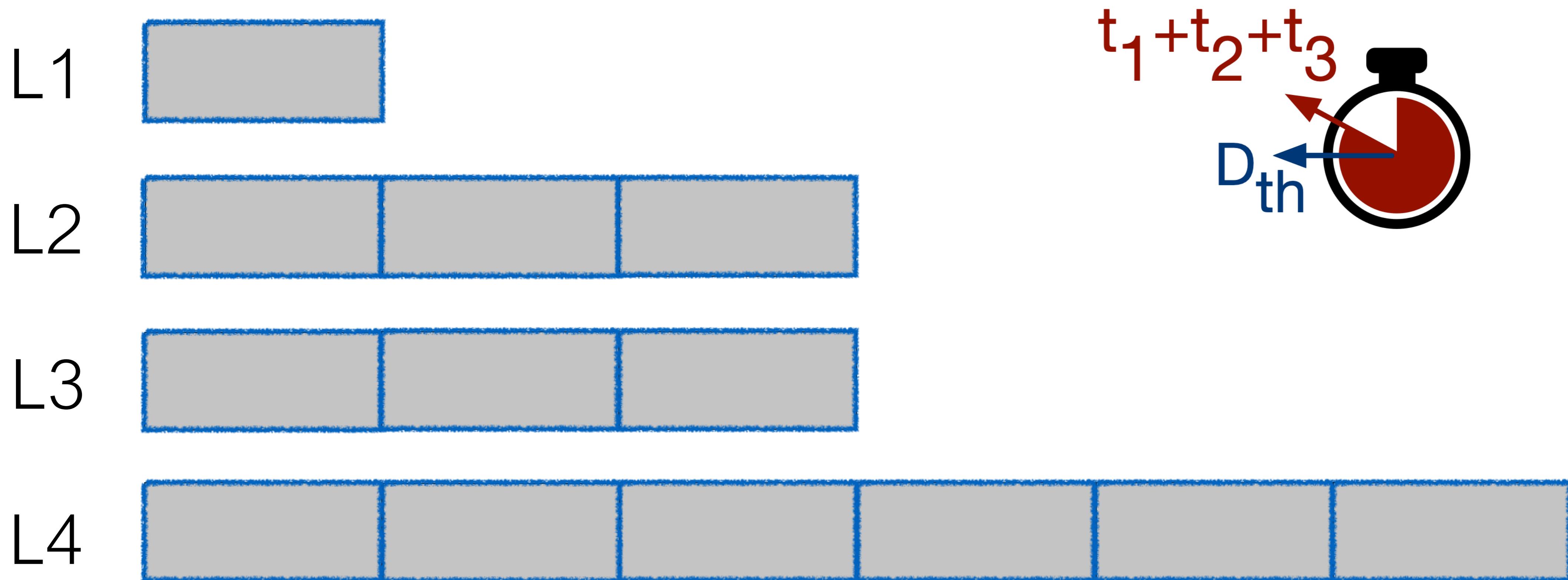
delete persistence latency

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



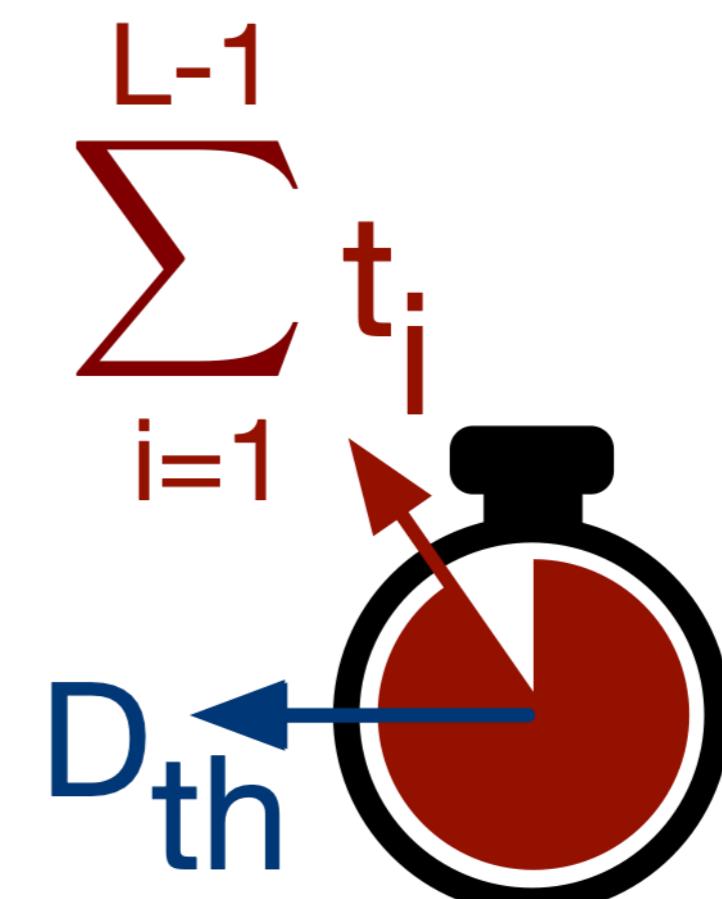
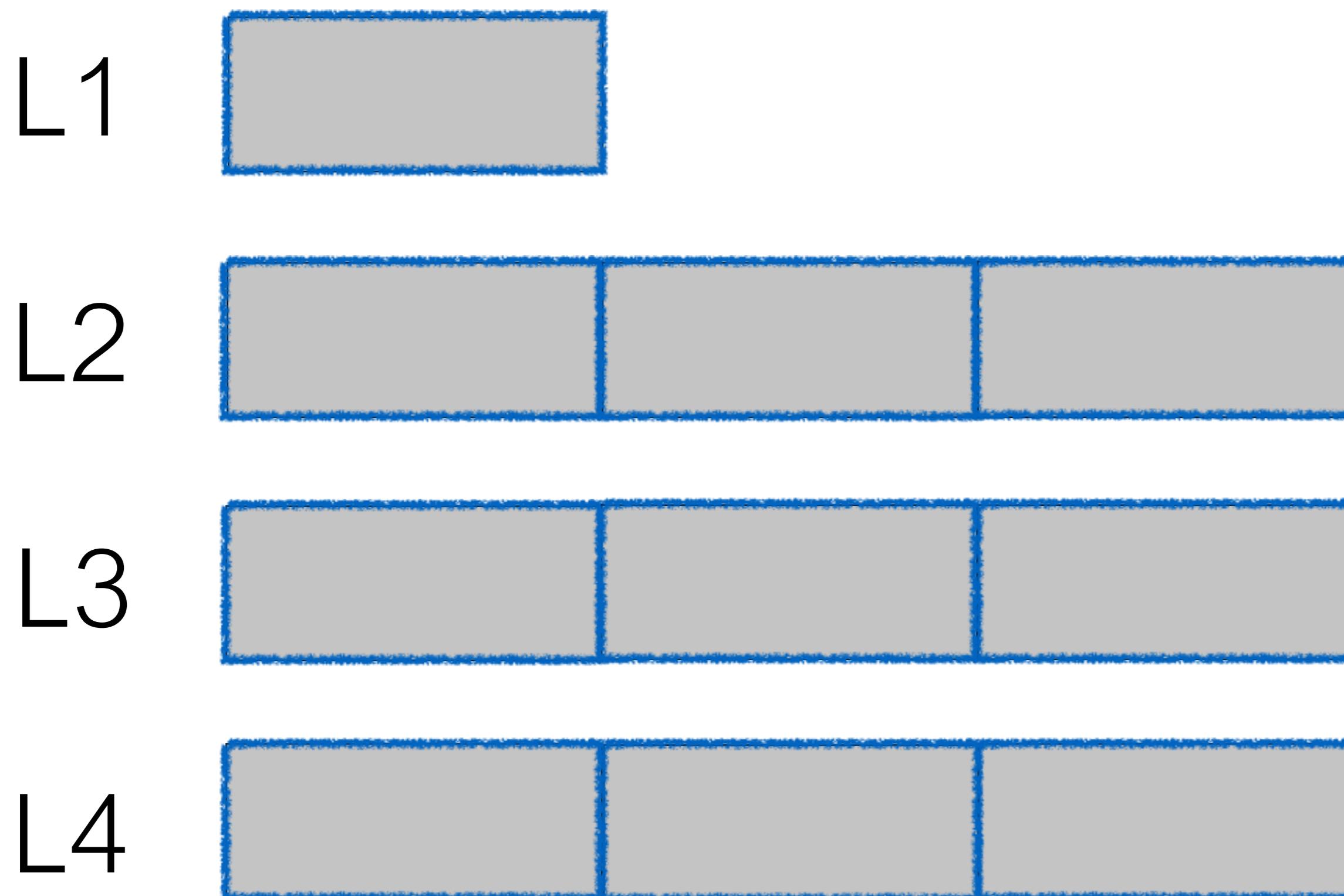
delete persistence latency

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



delete persistence latency

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



unbounded delete
persistence latency

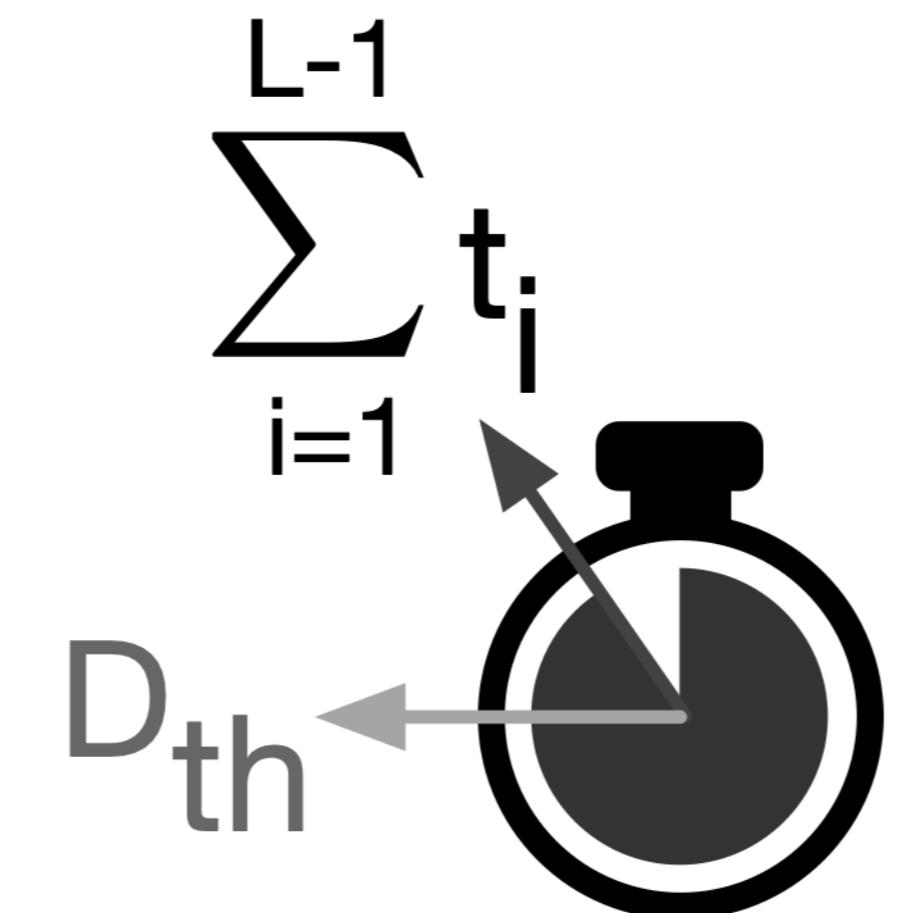
X

the problems

poor read perf.

write amplification

space amplification



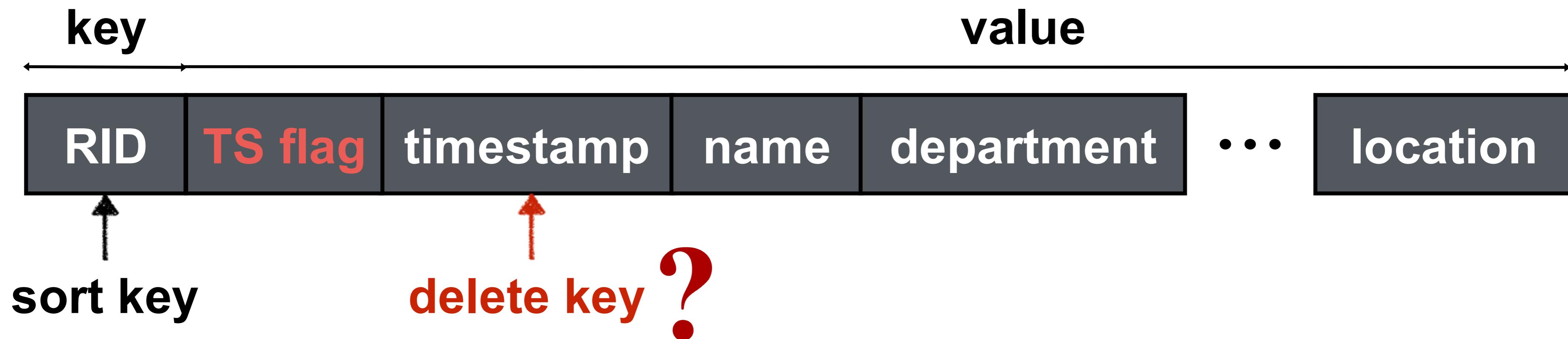
unbounded delete
persistence latency



deletes on a secondary attribute

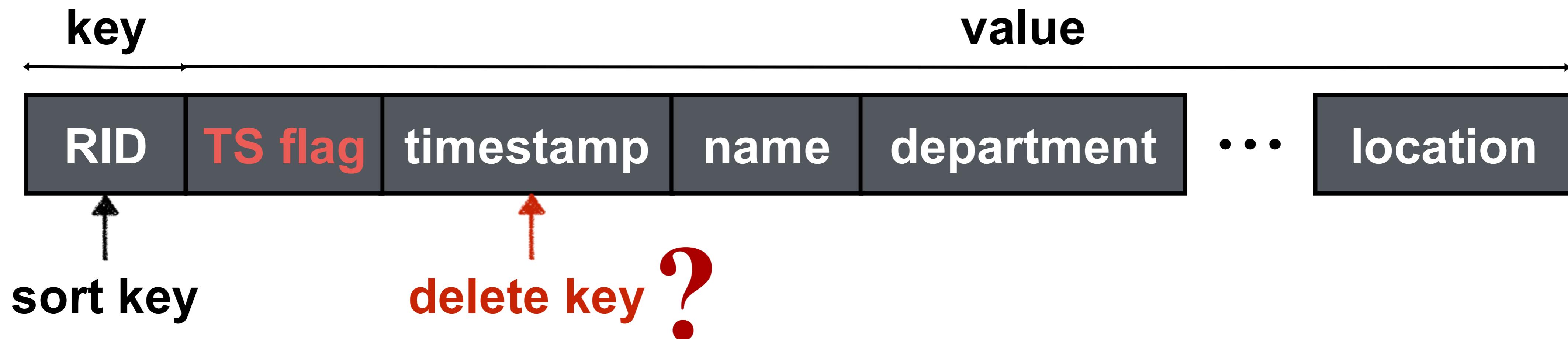
deletes on a secondary attribute

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



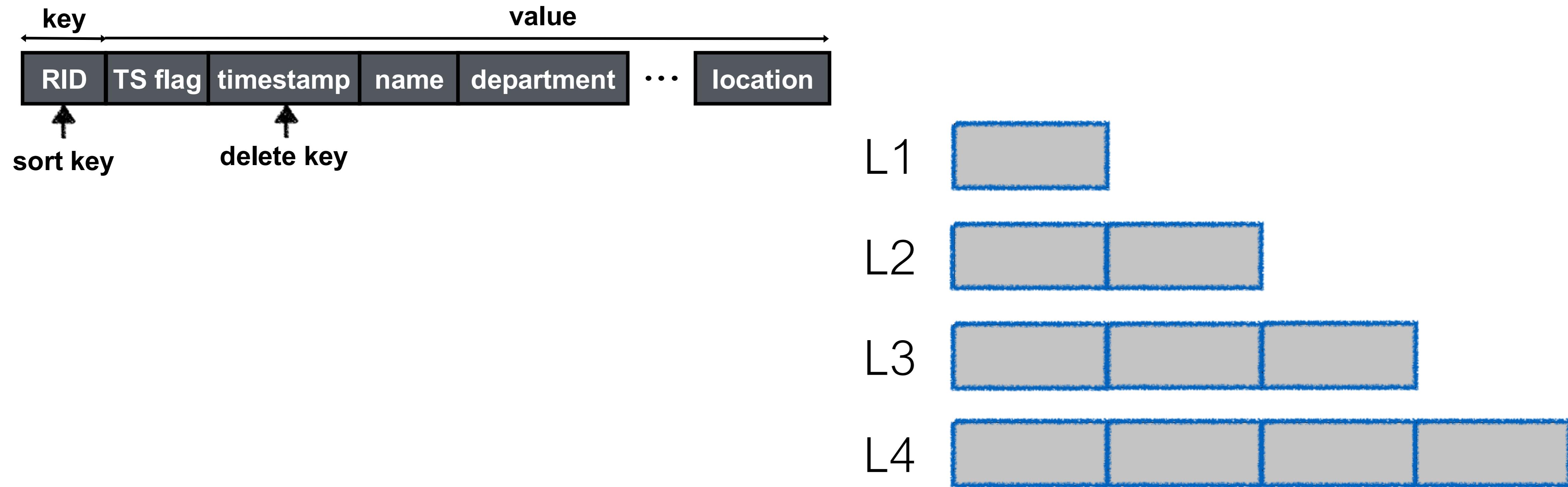
deletes on a secondary attribute

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



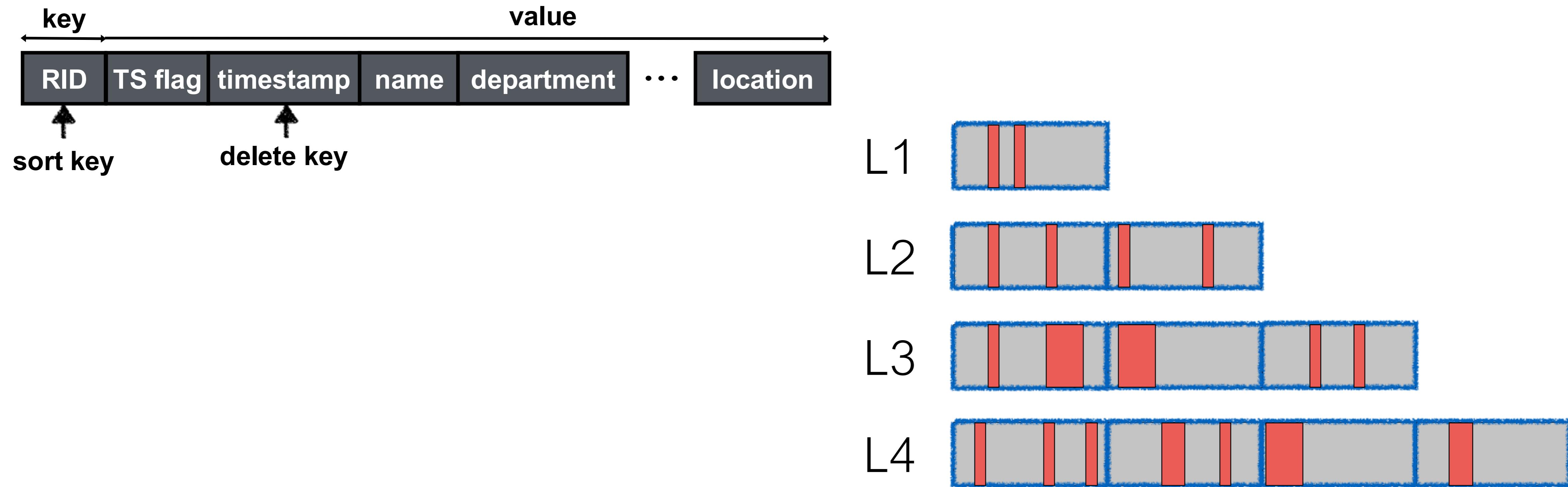
deletes on a secondary attribute

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



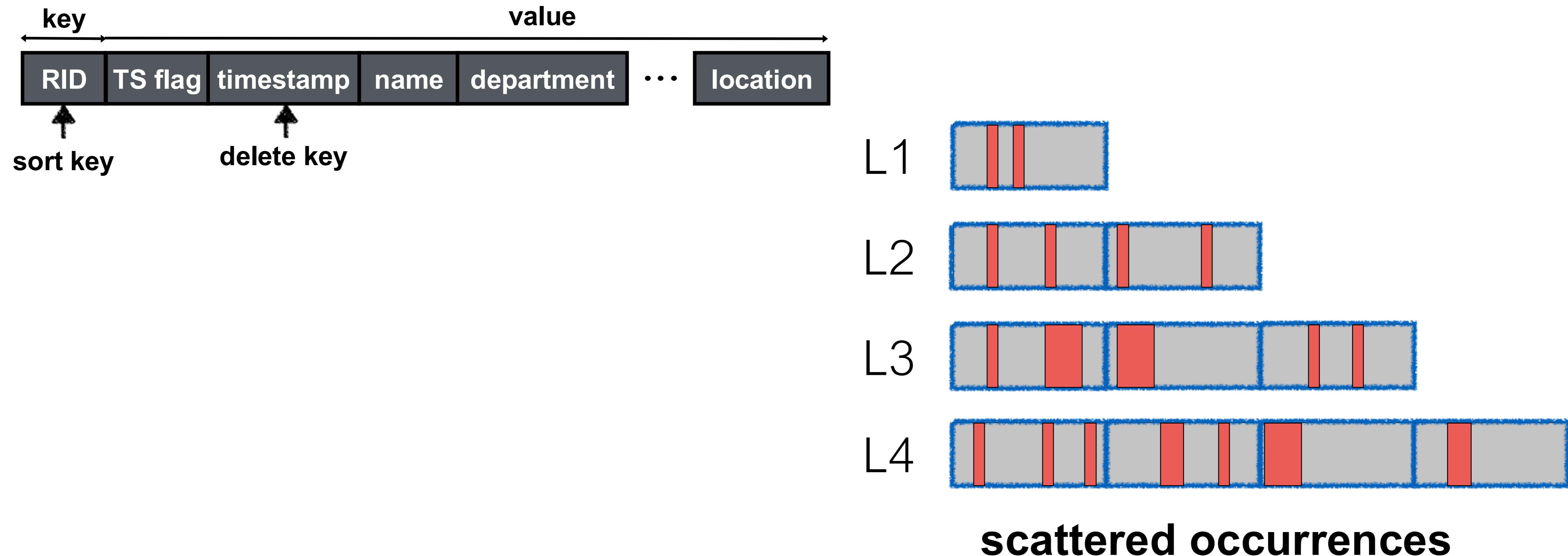
deletes on a secondary attribute

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



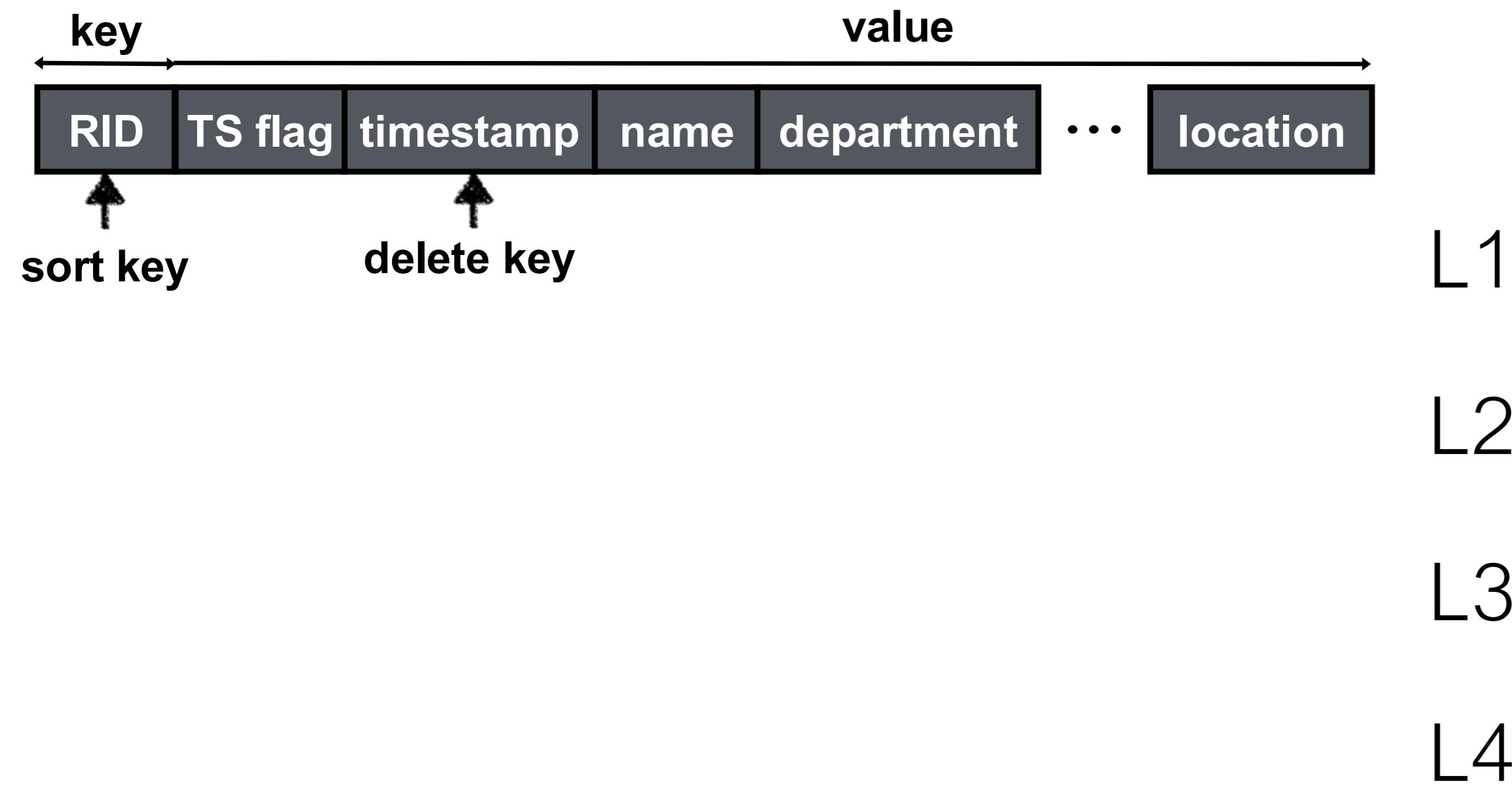
deletes on a secondary attribute

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



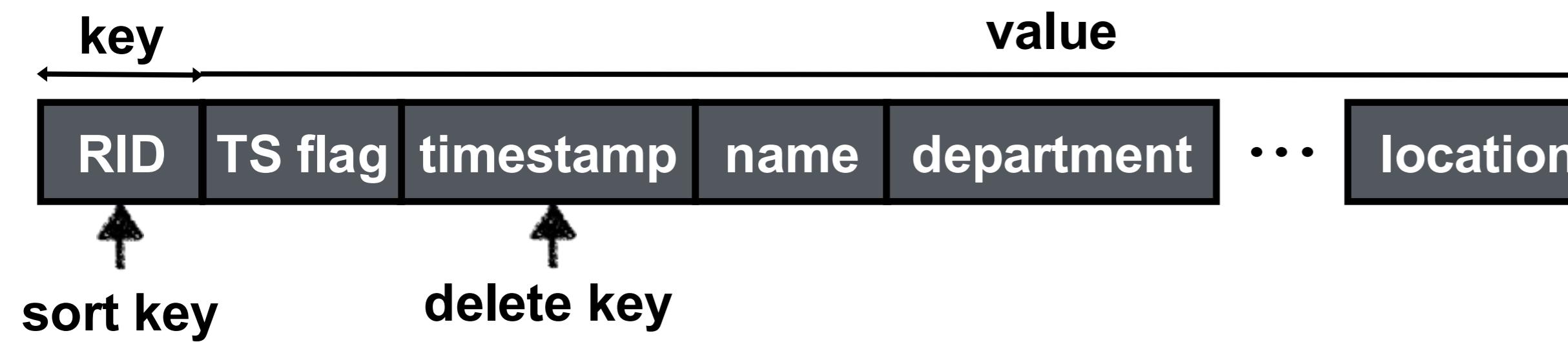
deletes on a secondary attribute

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



deletes on a secondary attribute

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



L1

L2

L3

L4

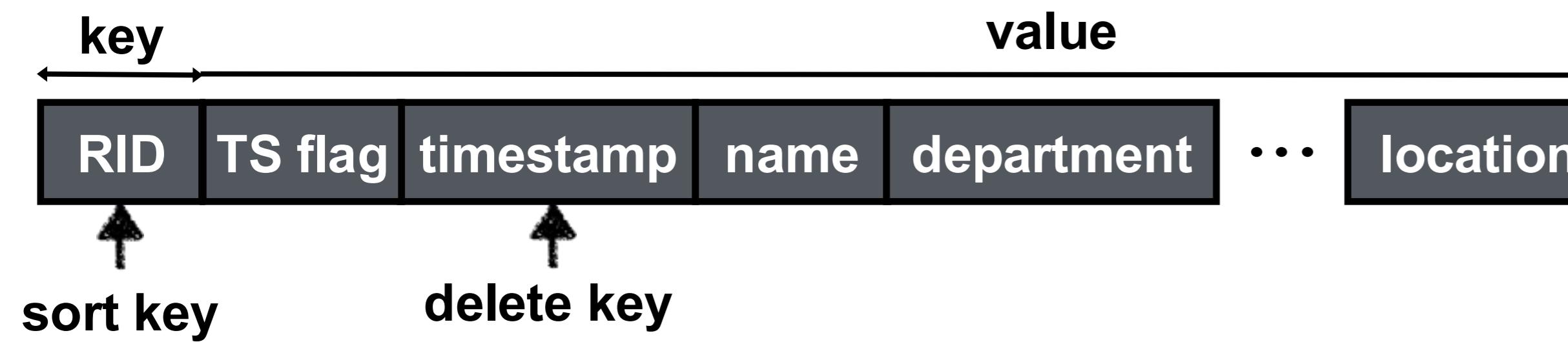
latency spikes X

superfluous I/Os X



deletes on a secondary attribute

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



L1

L2

L3

L4

latency spikes X

superfluous I/Os X

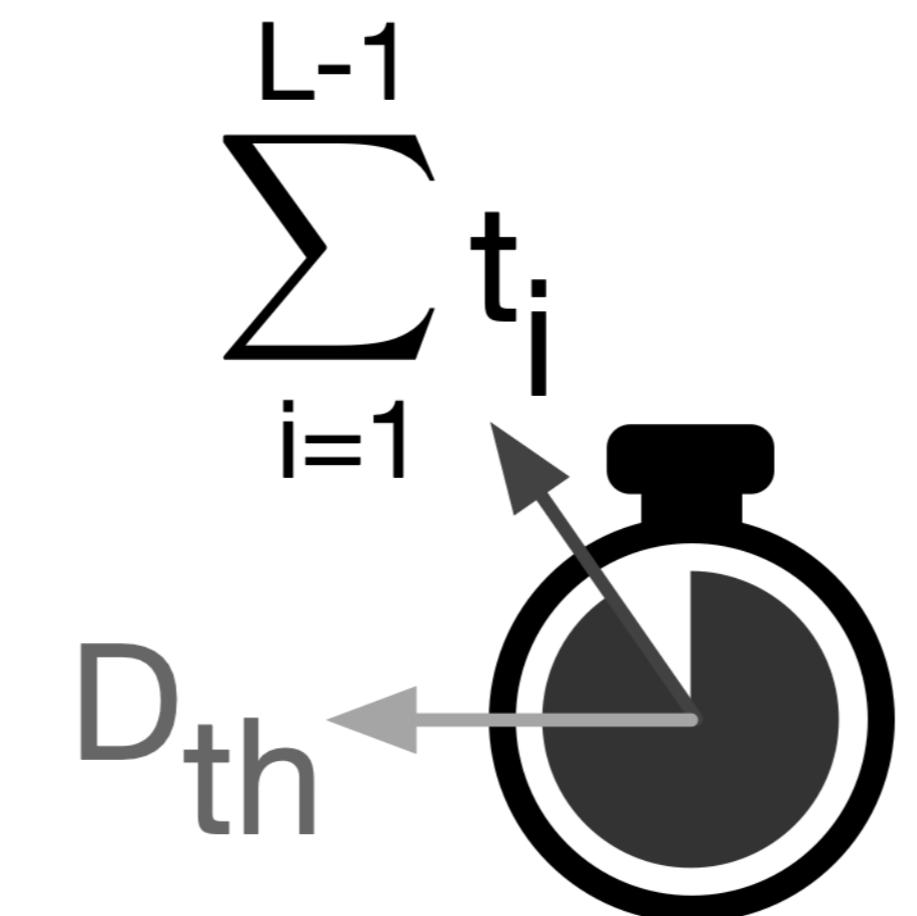


the problems

poor read perf.

write amplification

space amplification

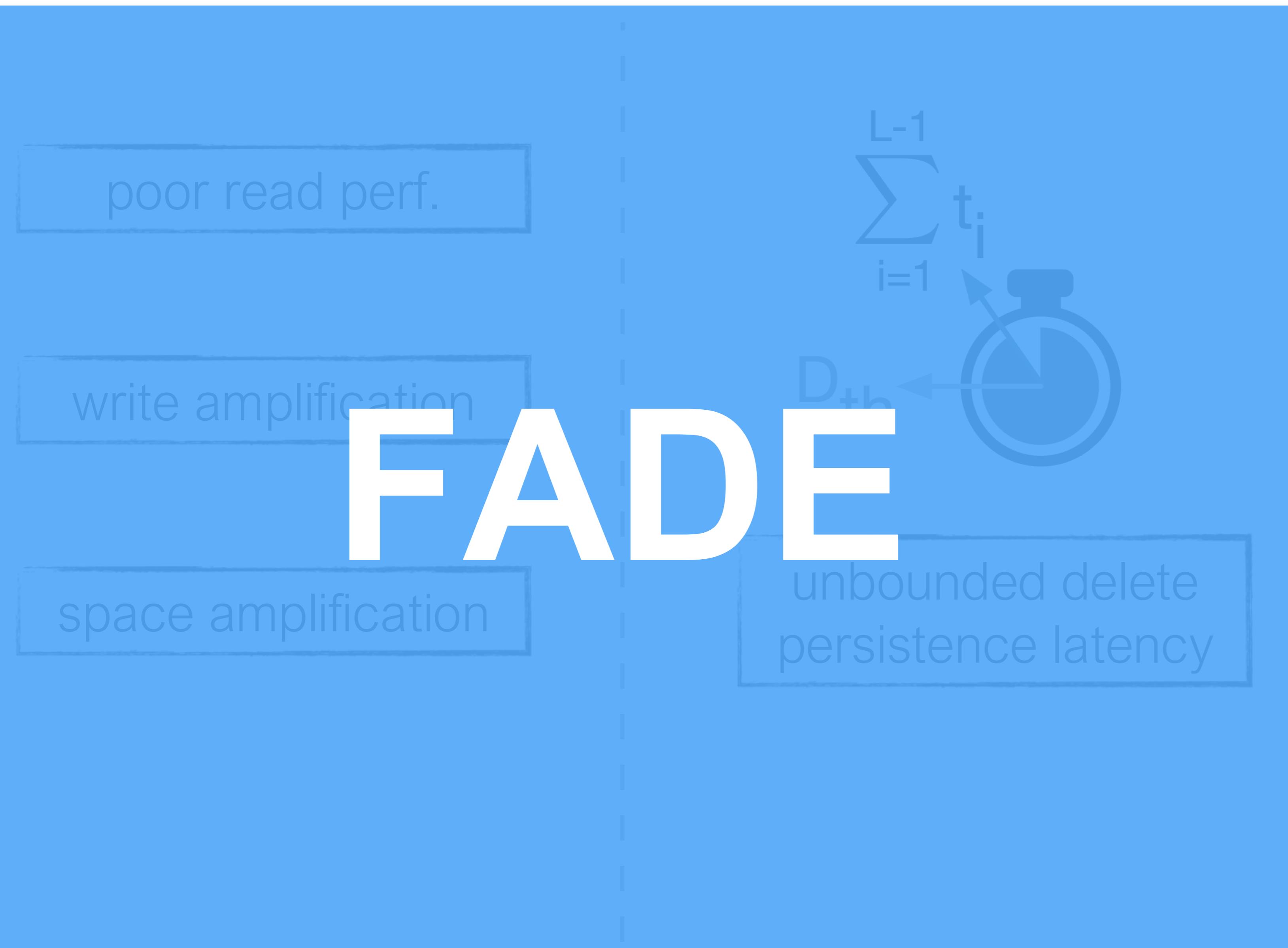


unbounded delete
persistence latency

latency spikes

superfluous I/Os

the solution

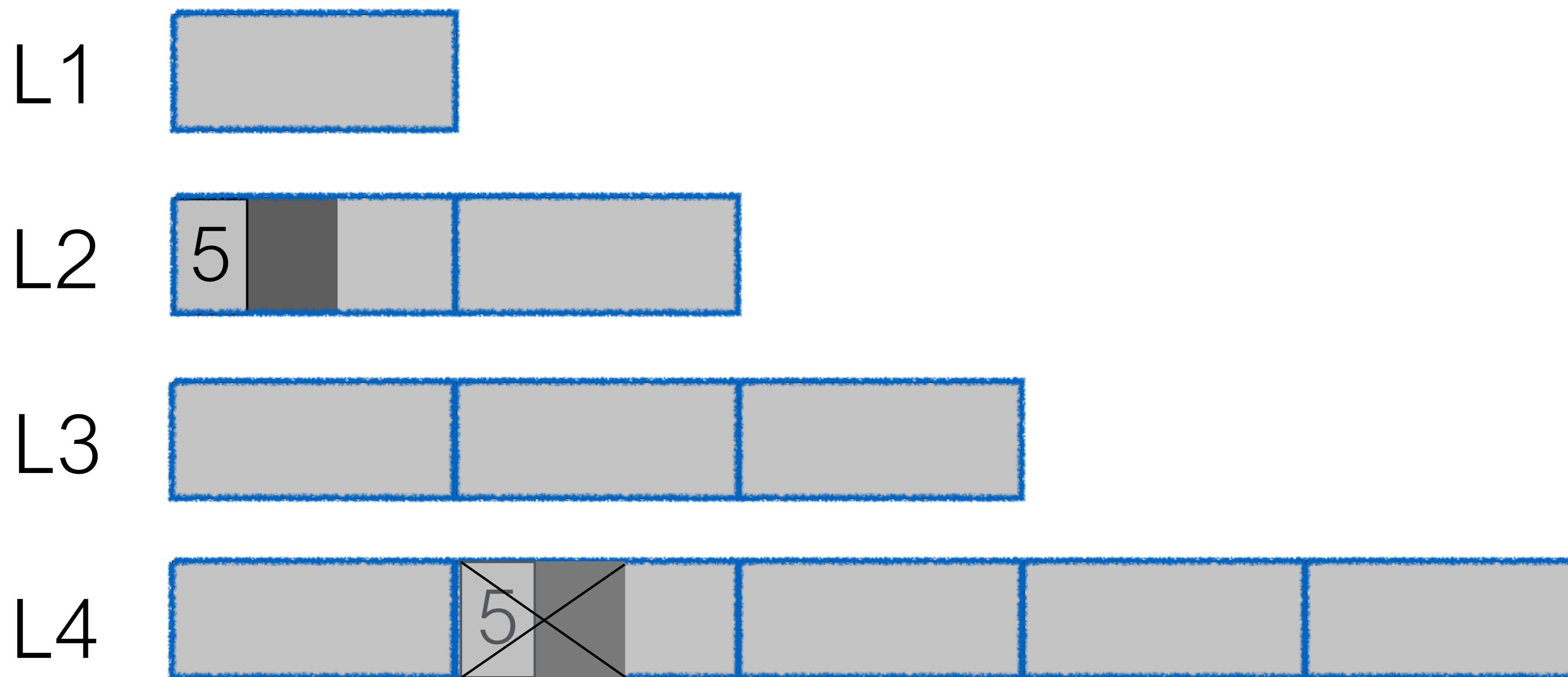


latency spikes

superfluous I/Os

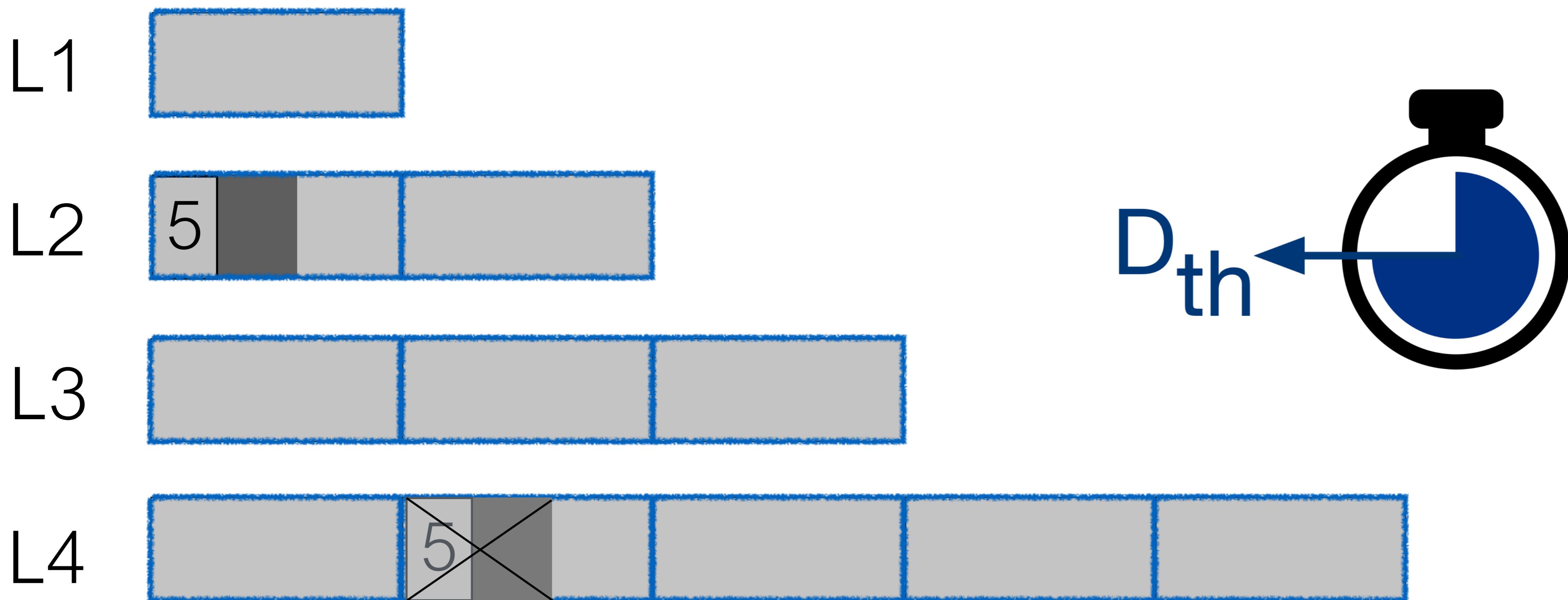
FAst DElete

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



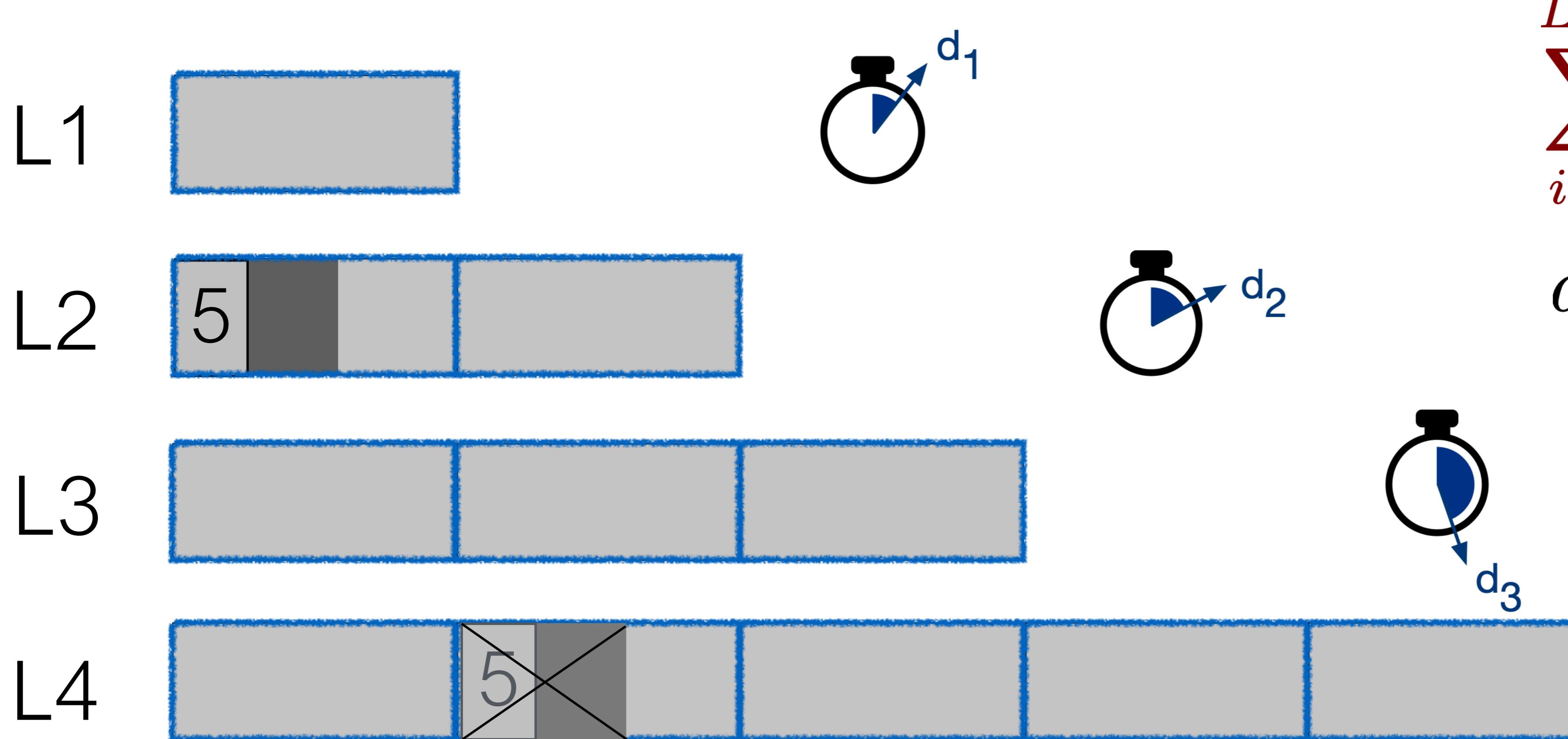
FAst DElete

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



FAst DElete

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



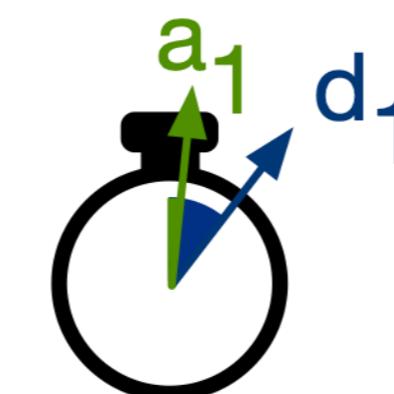
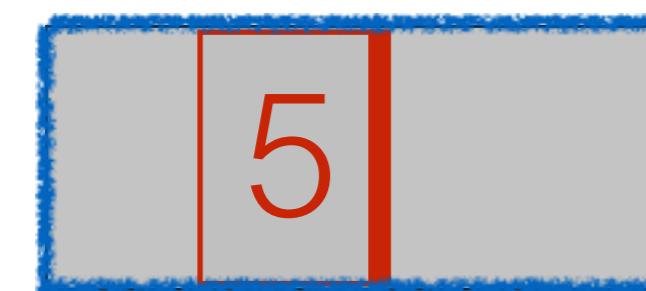
$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

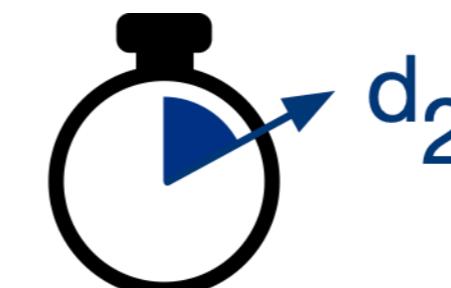
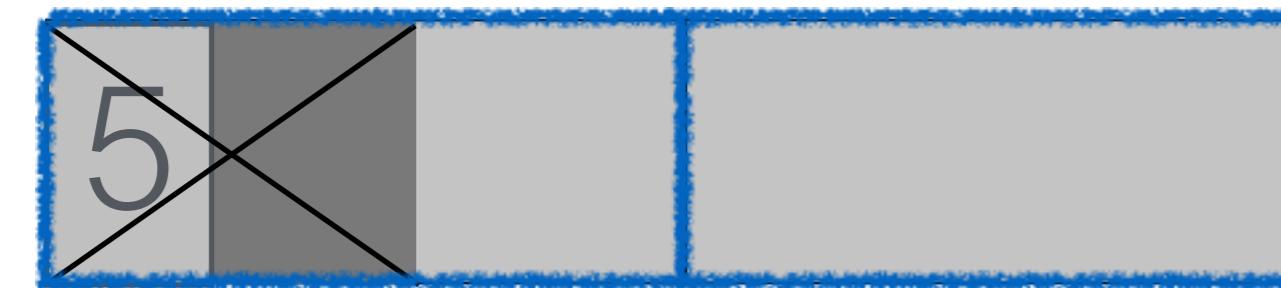
delete(5) within a threshold time: D_{th}

L1



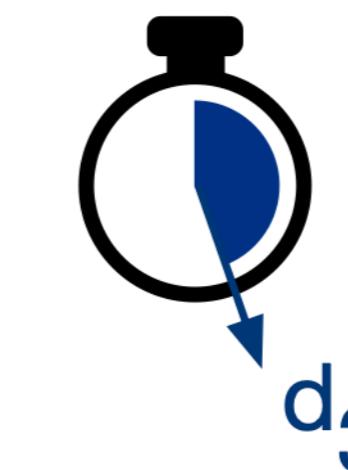
$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

L2



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

L3



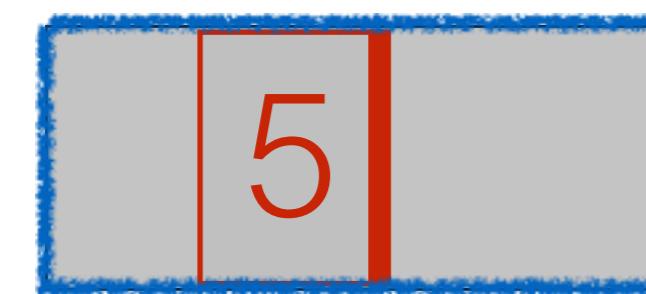
L4



FAst DElete

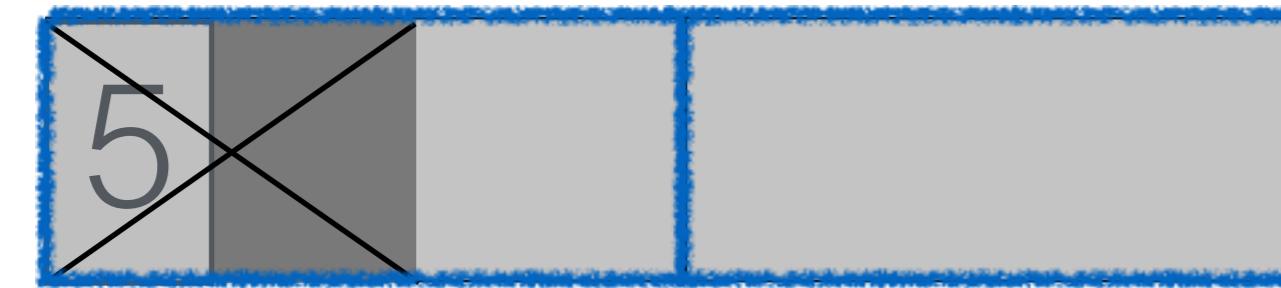
delete(5) within a threshold time: D_{th}

L1



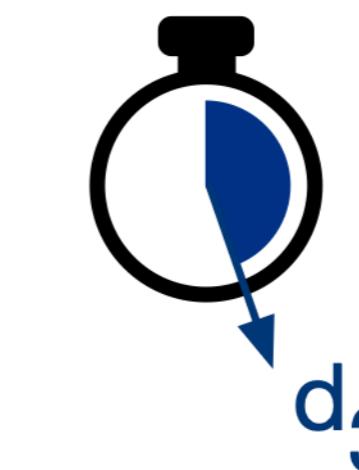
$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

L2



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

L3



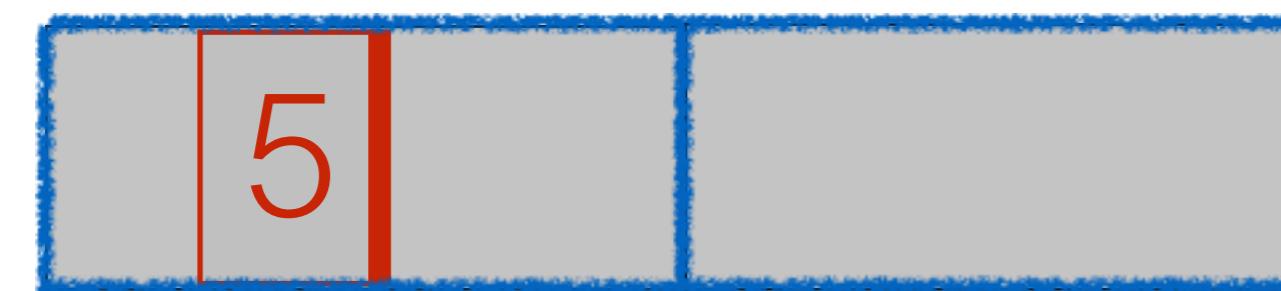
L4



FAst DElete

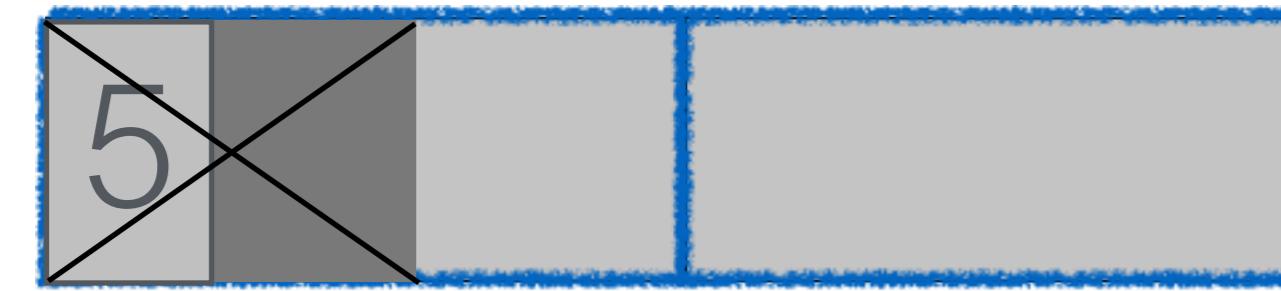
delete(5) within a threshold time: D_{th}

L1



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

L2



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

L3

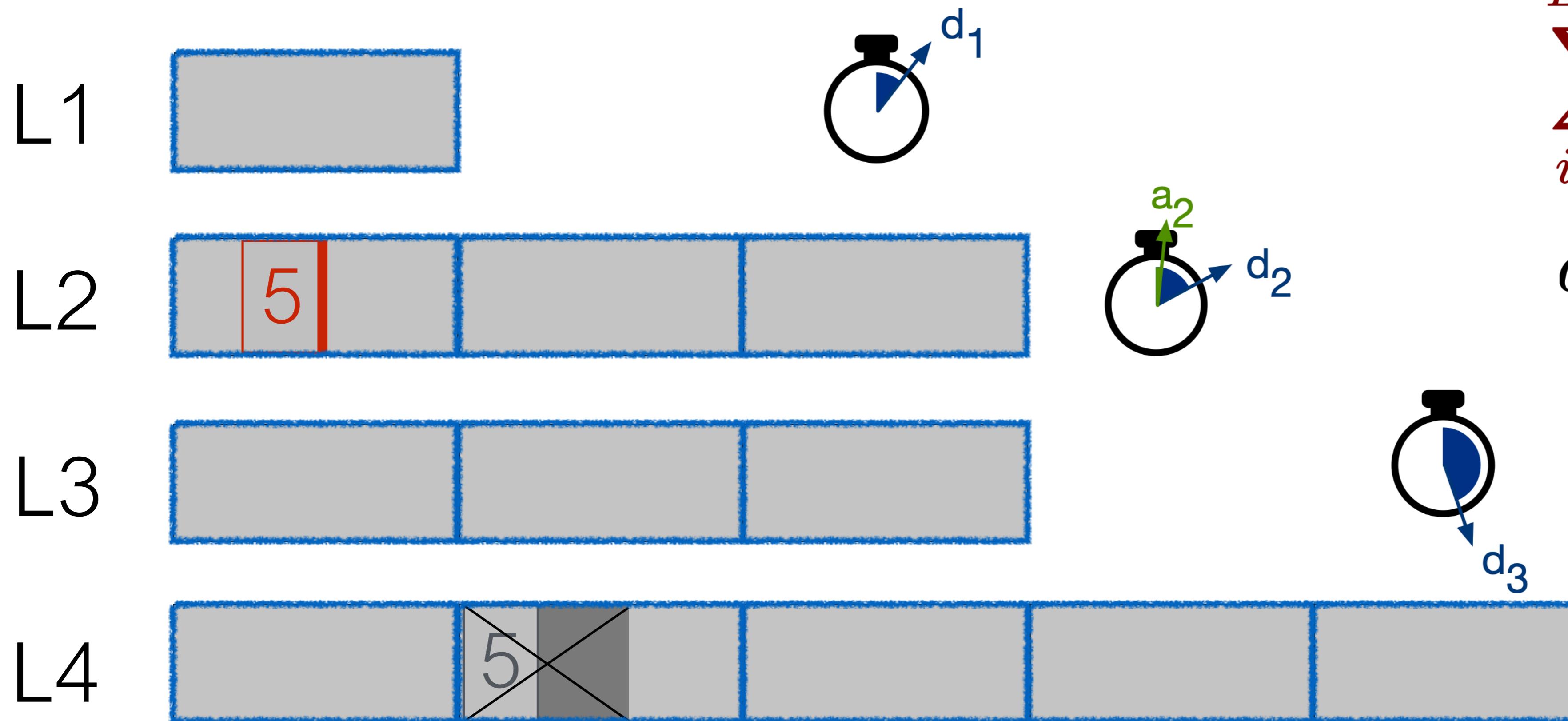


L4



FAst DElete

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

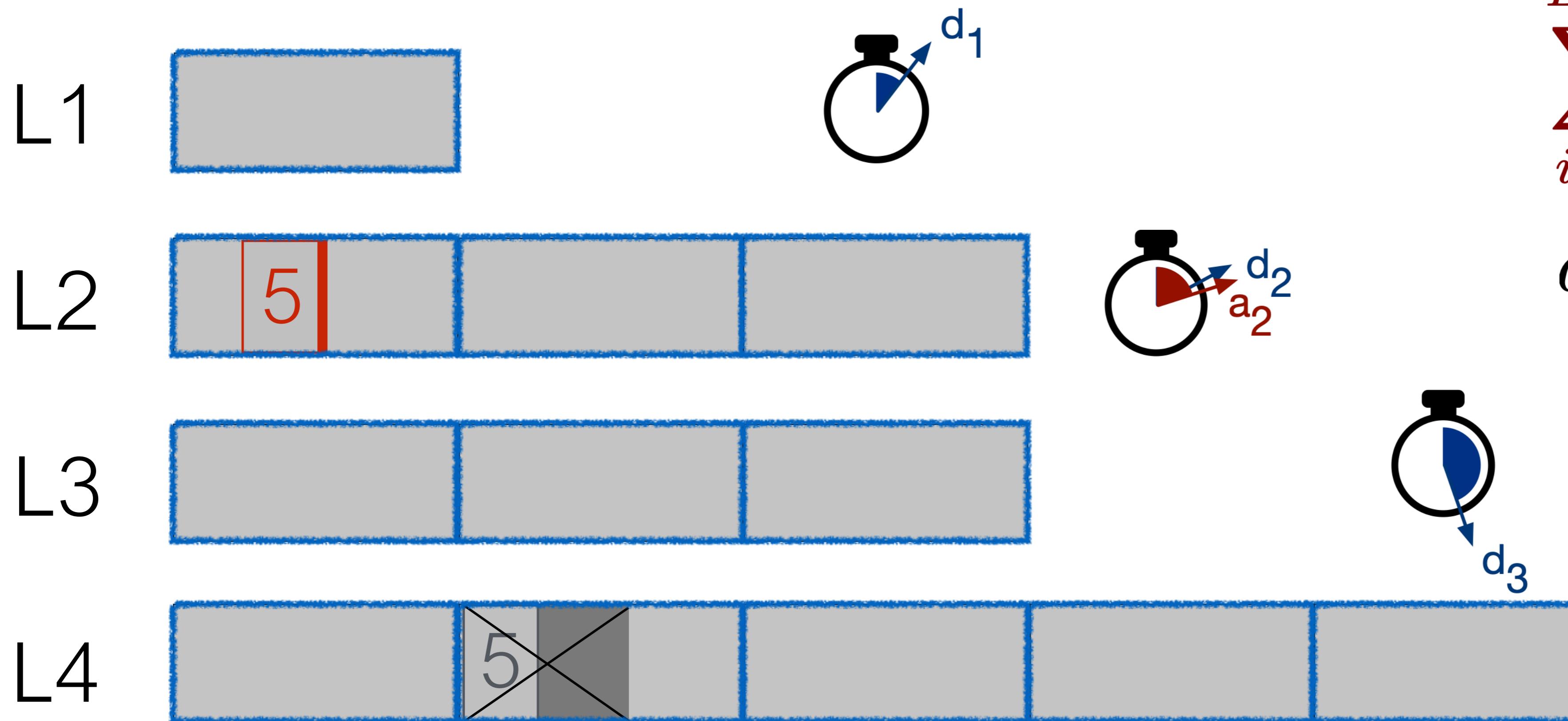


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

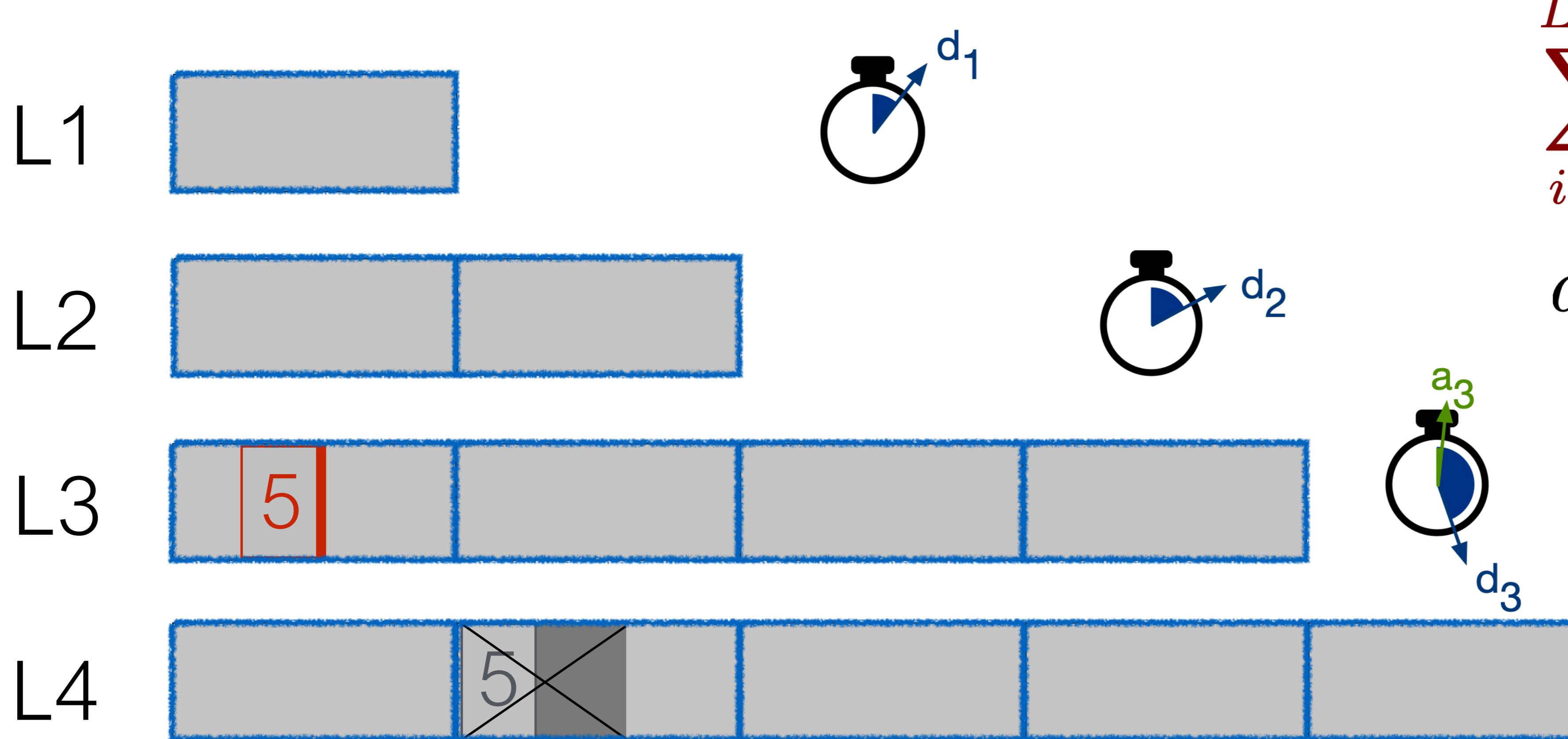


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

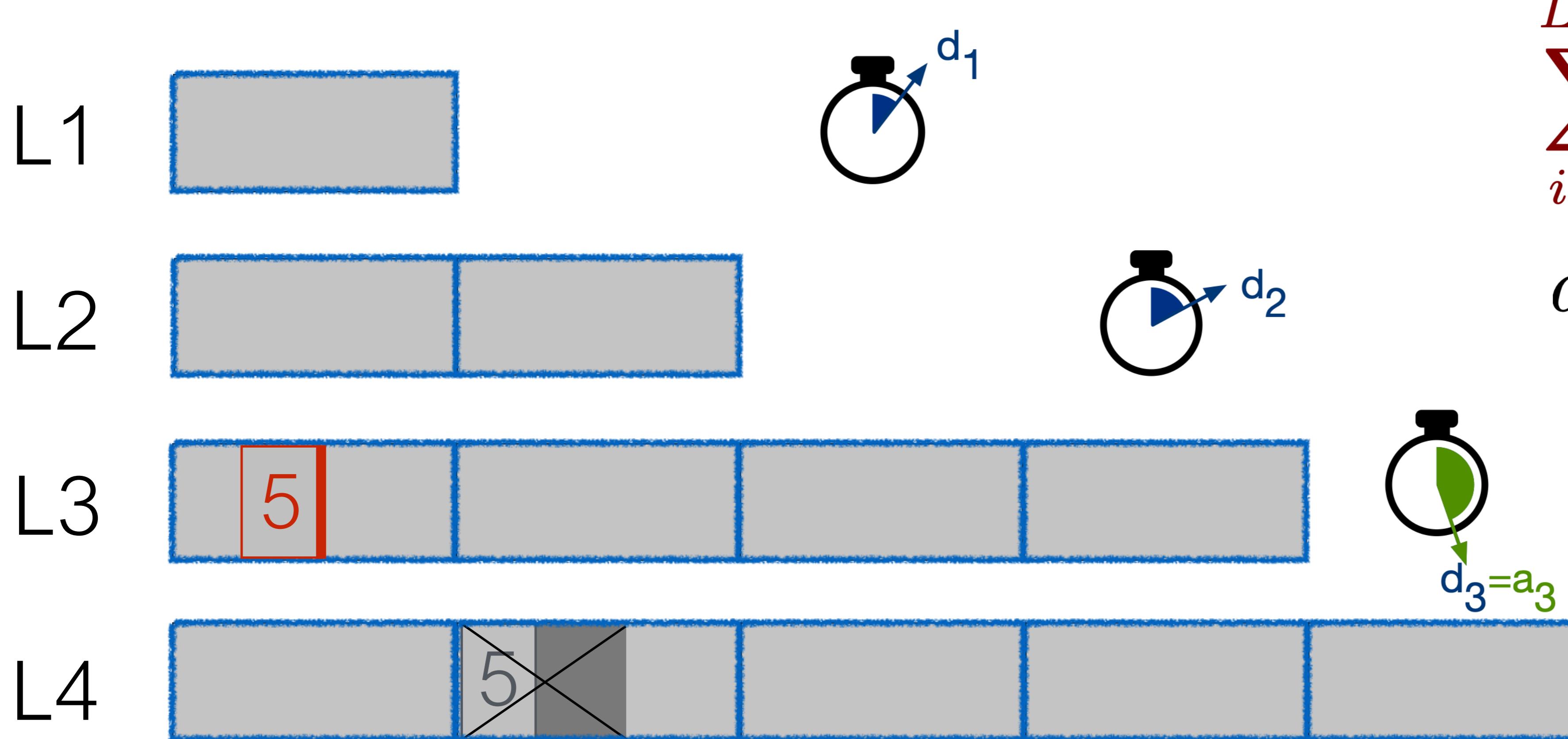


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

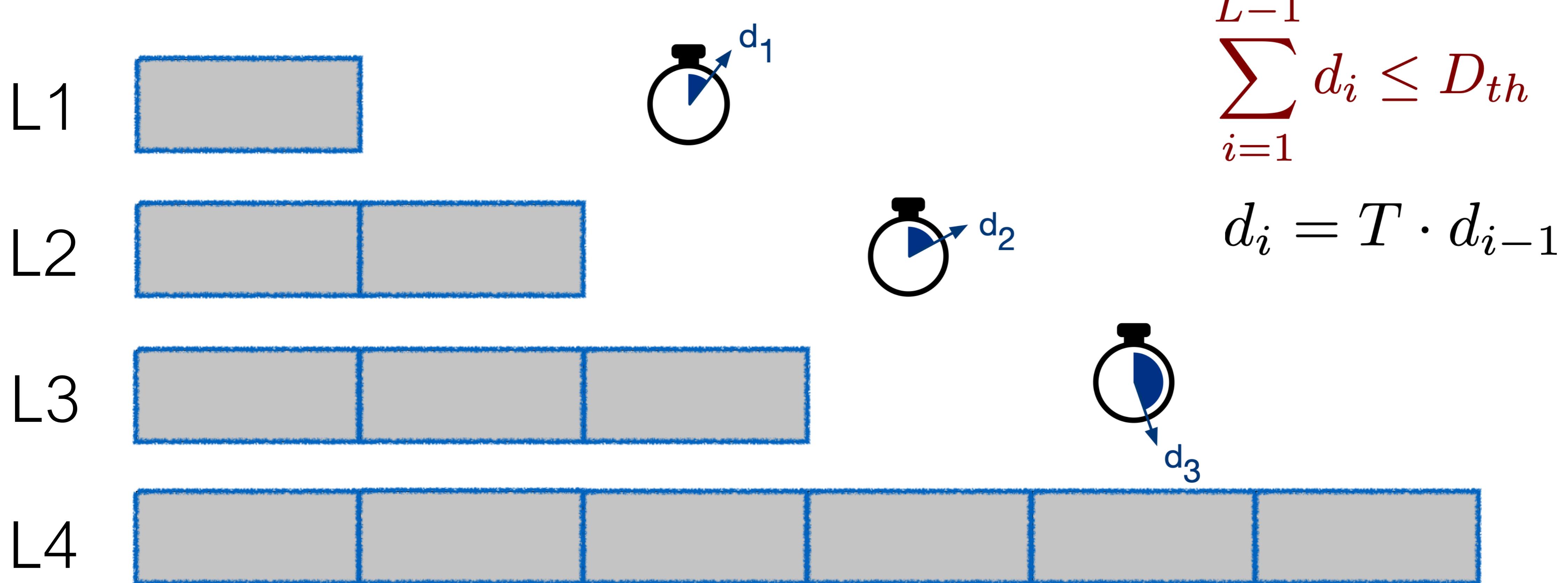


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

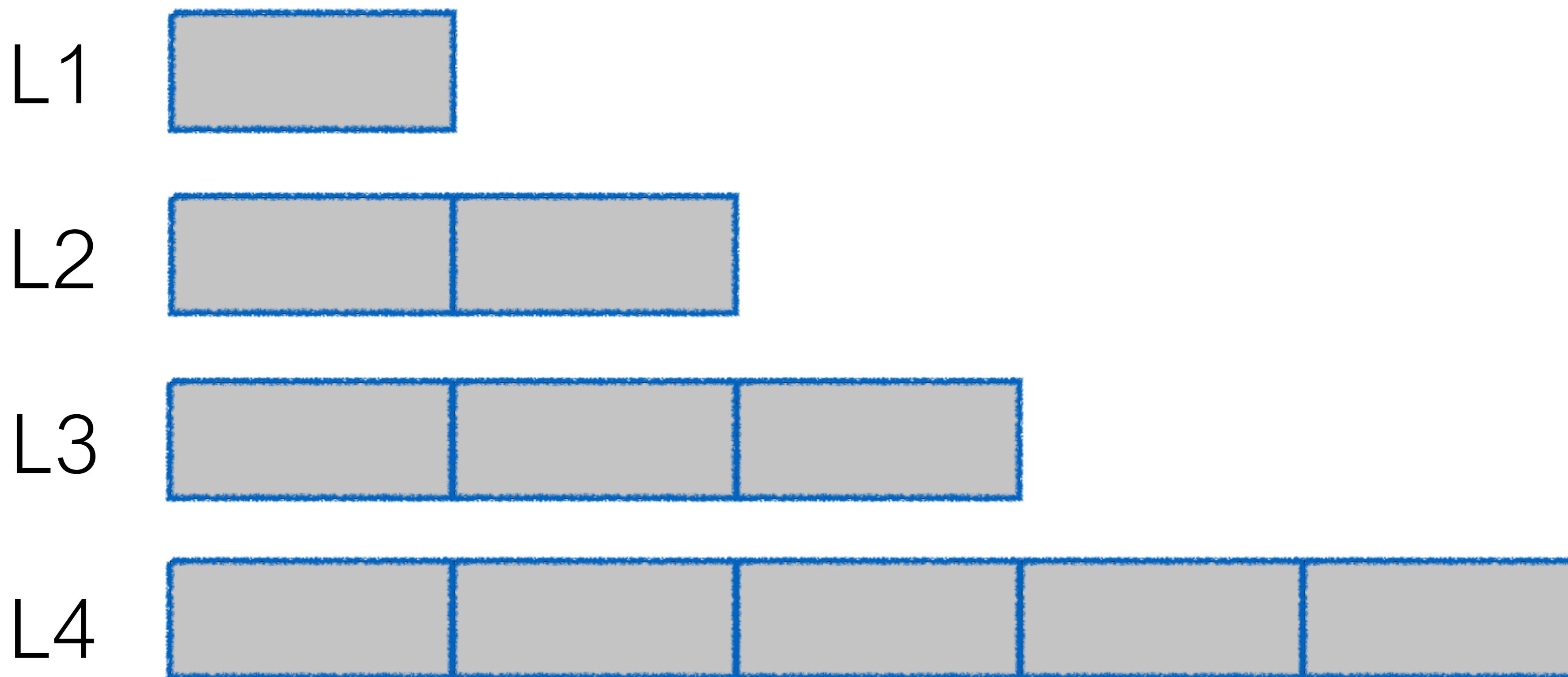
FAst DElete

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



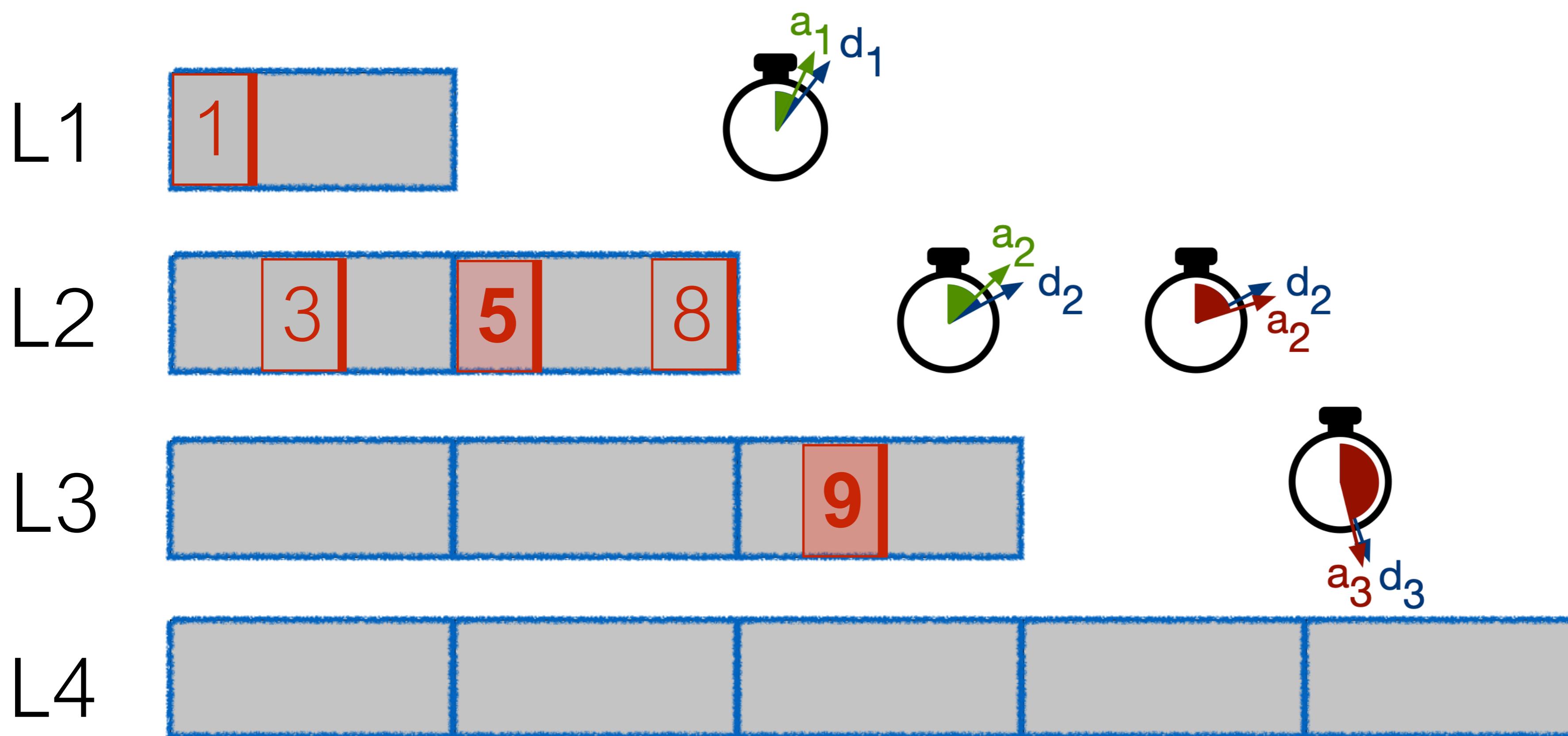
FAst DElete

breaking ties in practical workloads



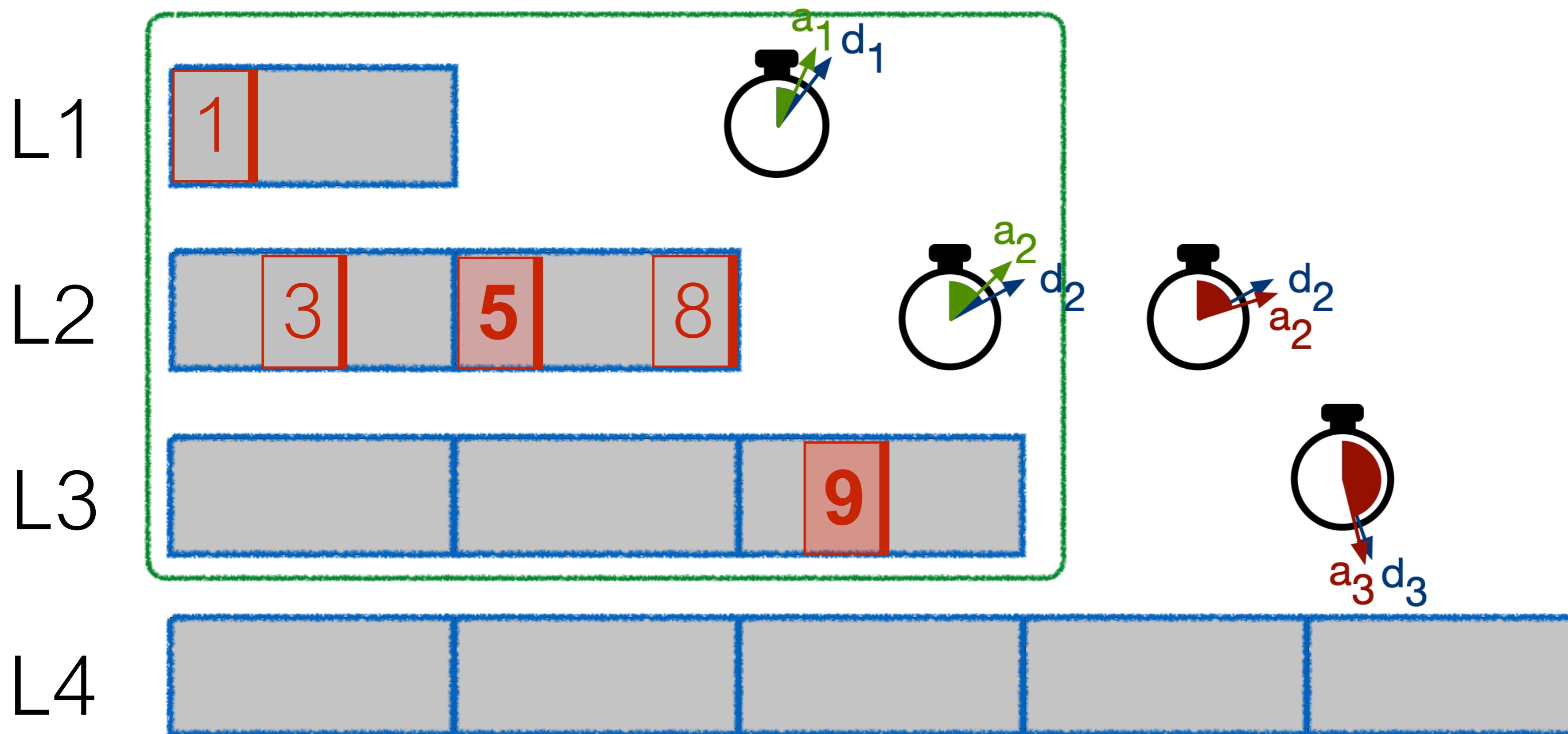
FAst DElete

breaking ties in practical workloads



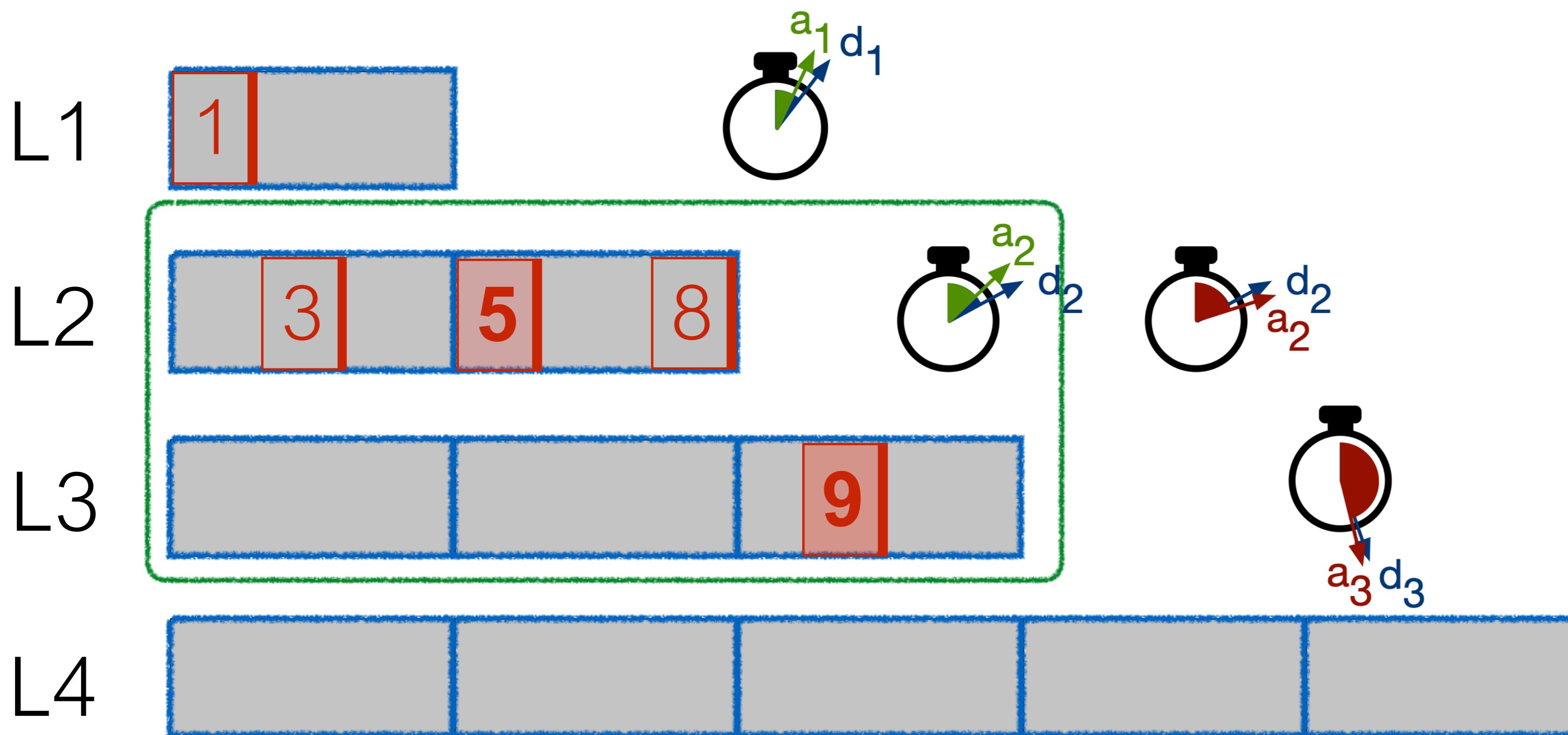
FAst DElete

breaking ties in practical workloads



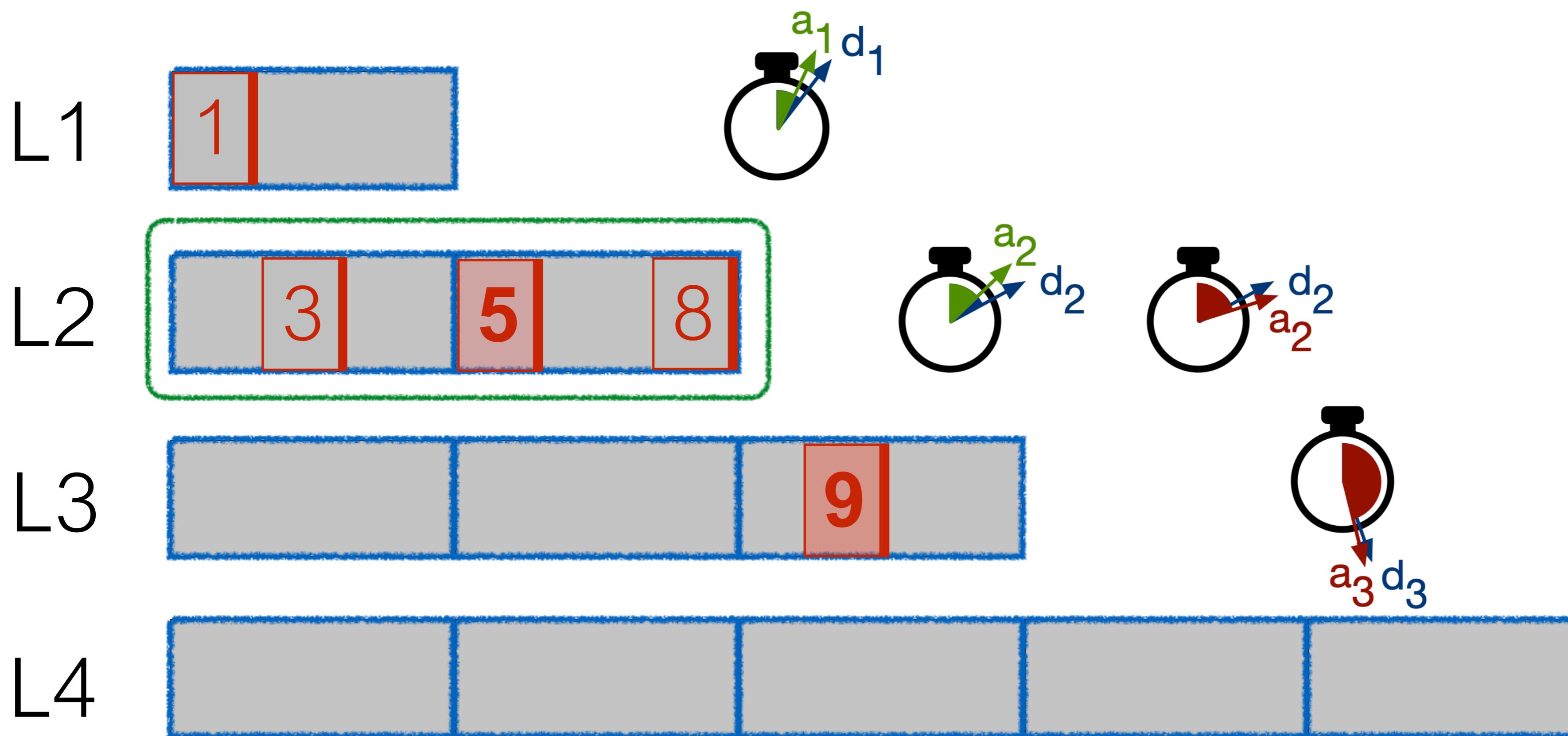
FAst DElete

breaking ties in practical workloads



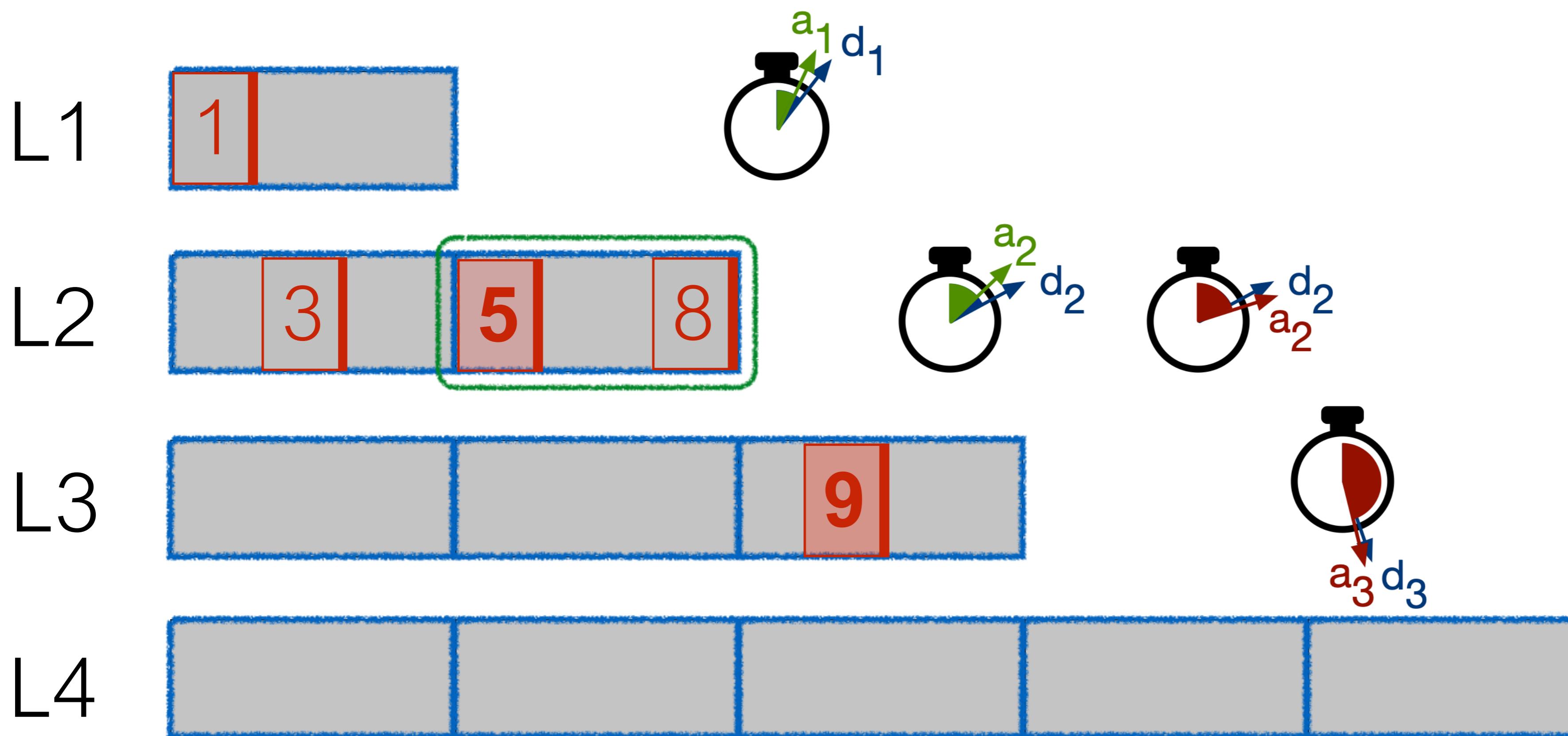
FAst DElete

breaking ties in practical workloads



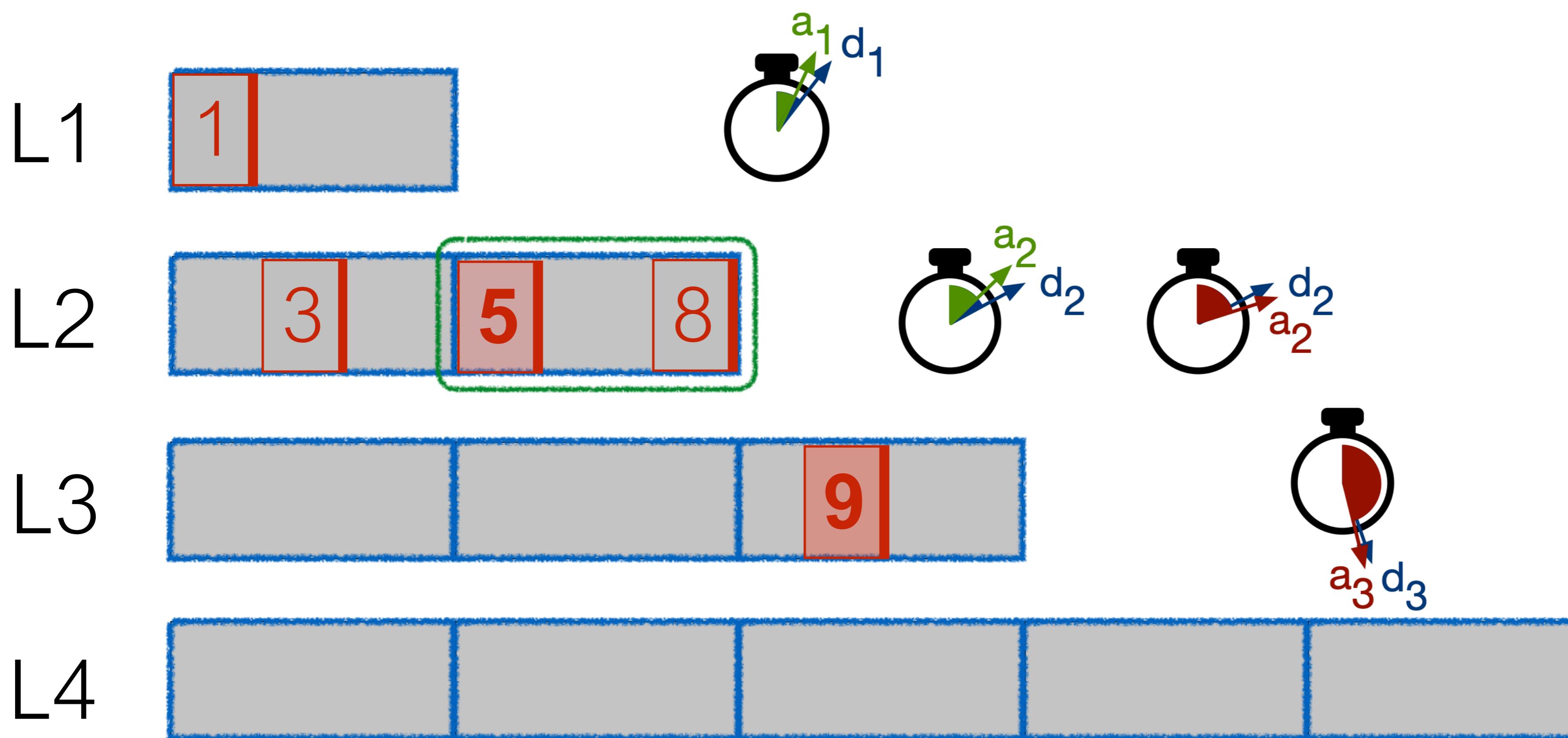
FAst DElete

breaking ties in practical workloads



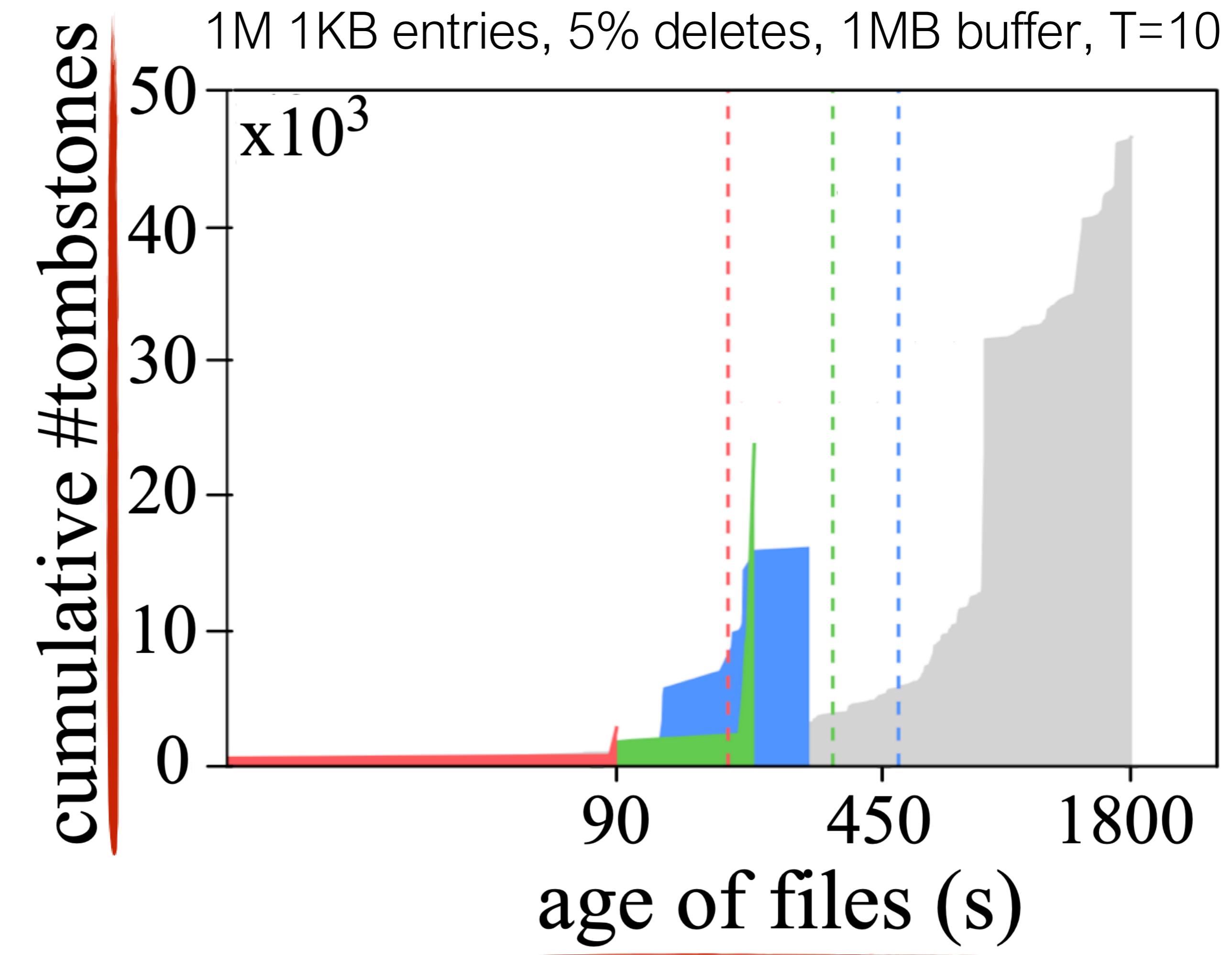
FAst DElete

breaking ties in practical workloads



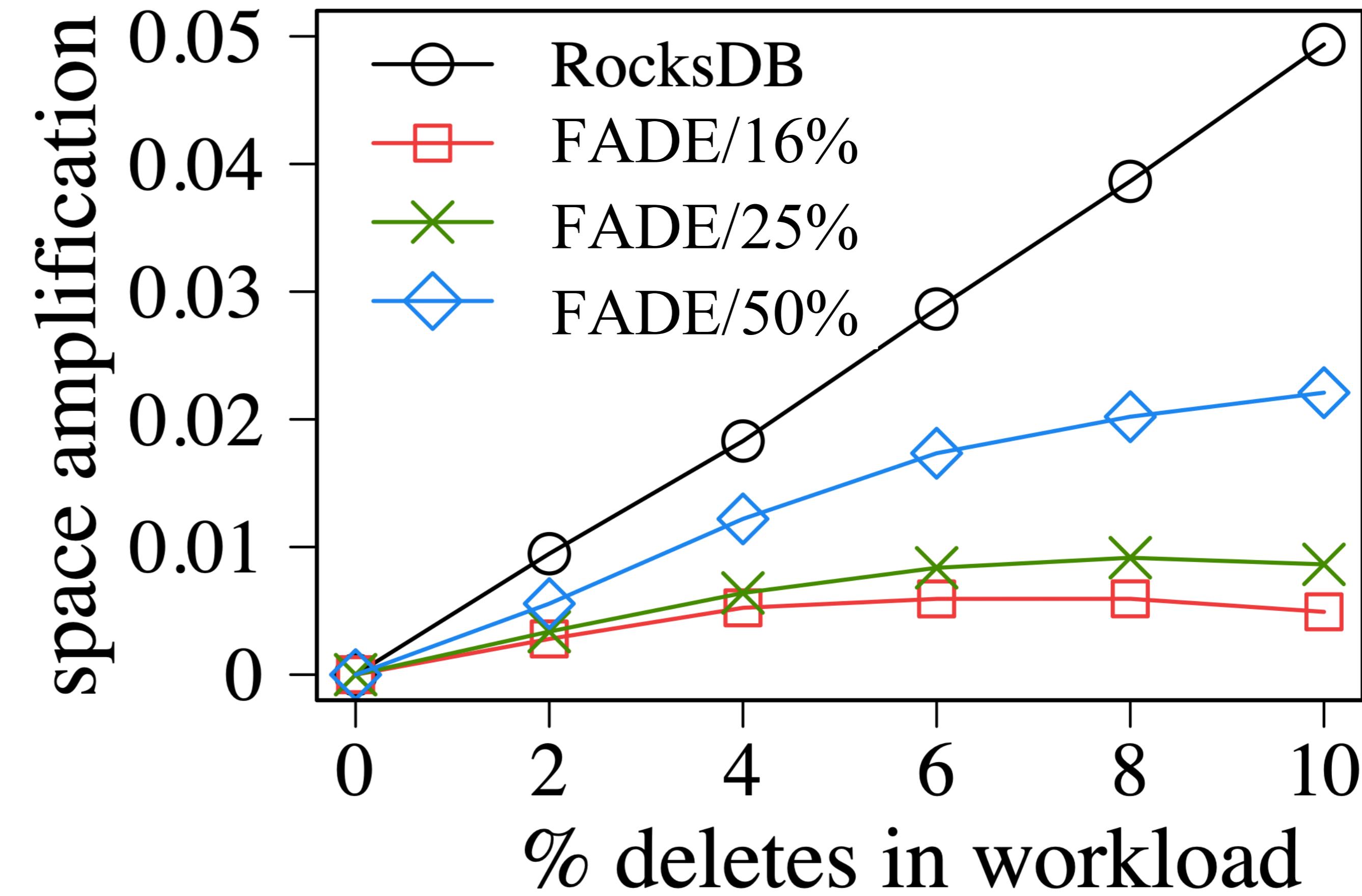
FAst DElete

timely delete persistence
within D_{th} ✓



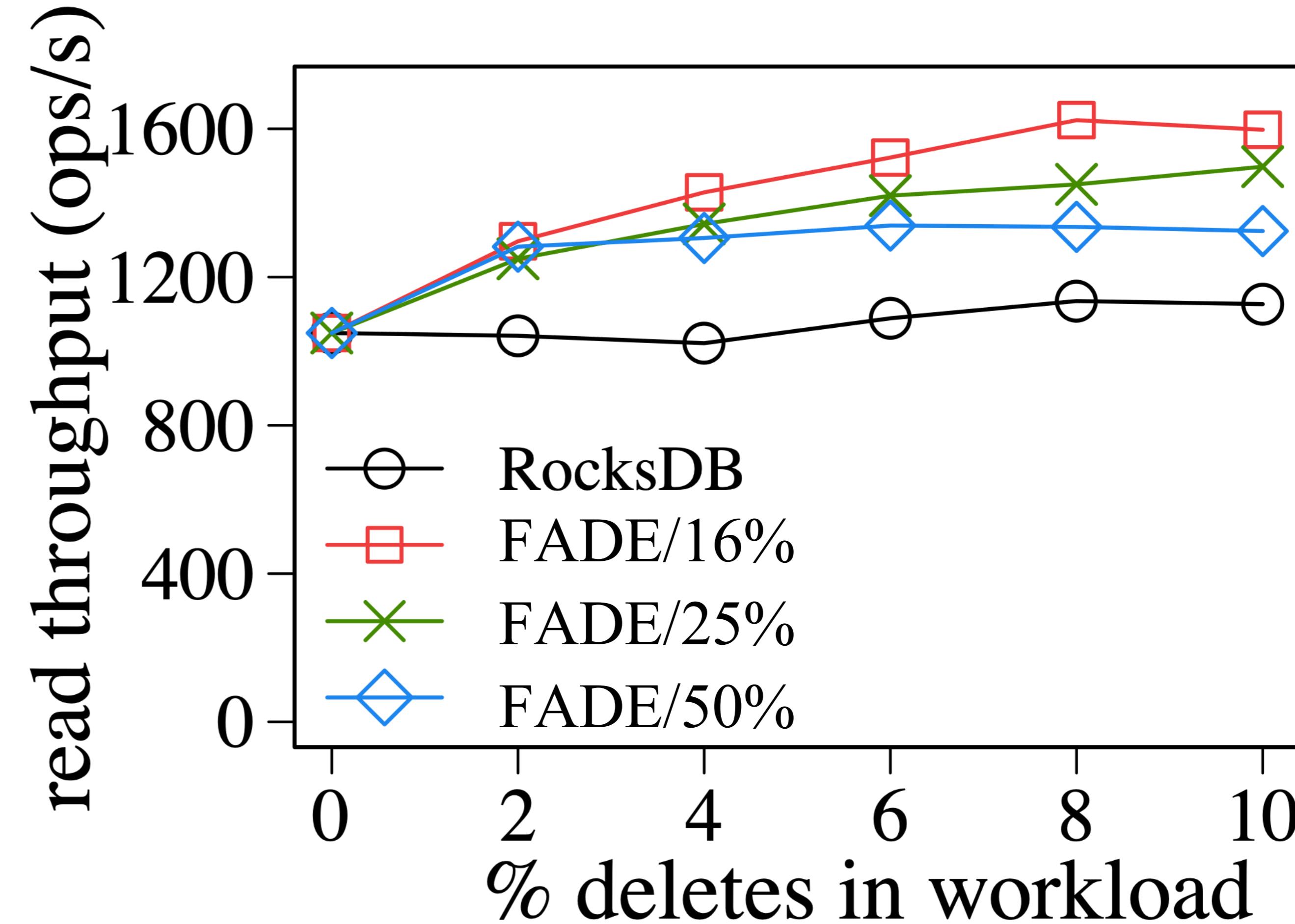
FAst DElete

- reduced space amplification
2.1 - 9.8x ✓
- timely delete persistence
within D_{th} ✓



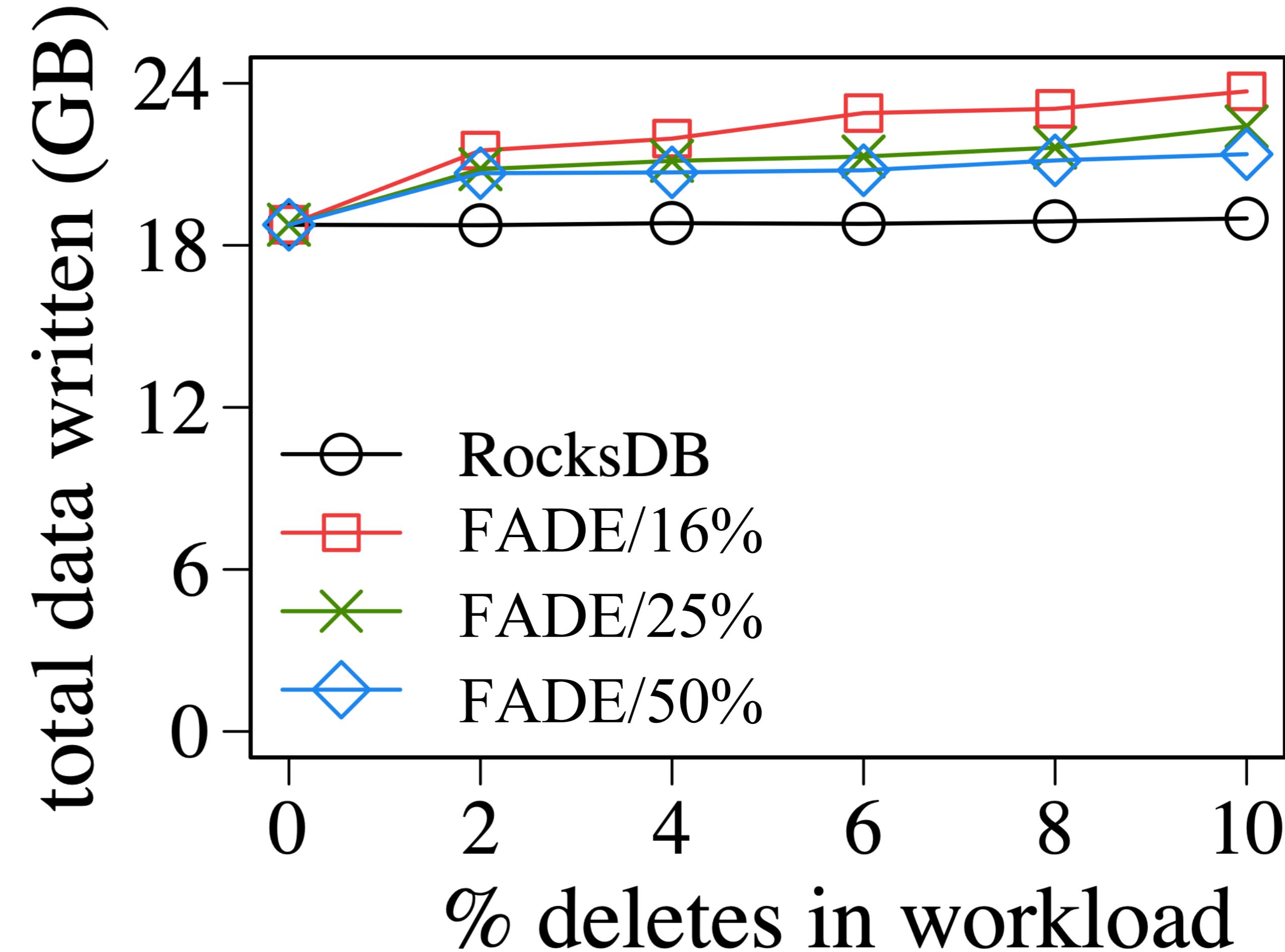
FAst DElete

- improved read performance **1.2 - 1.4x** ✓
- reduced space amplification **2.1 - 9.8x** ✓
- timely delete persistence **within D_{th}** ✓



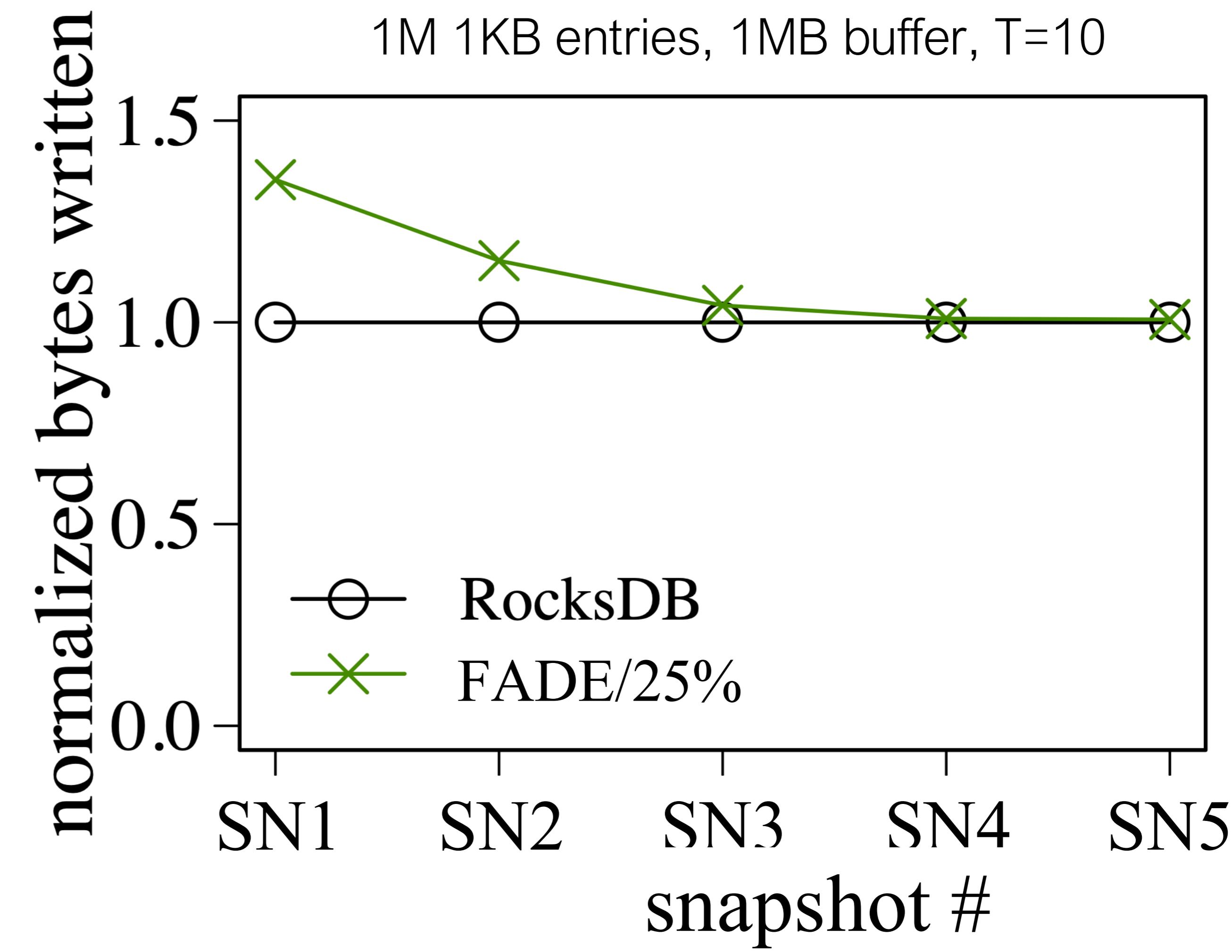
FAst DElete

- higher write amplification
4 - 25% ↑
- improved read performance
1.2 - 1.4x ✓
- reduced space amplification
2.1 - 9.8x ✓
- timely delete persistence
within D_{th} ✓



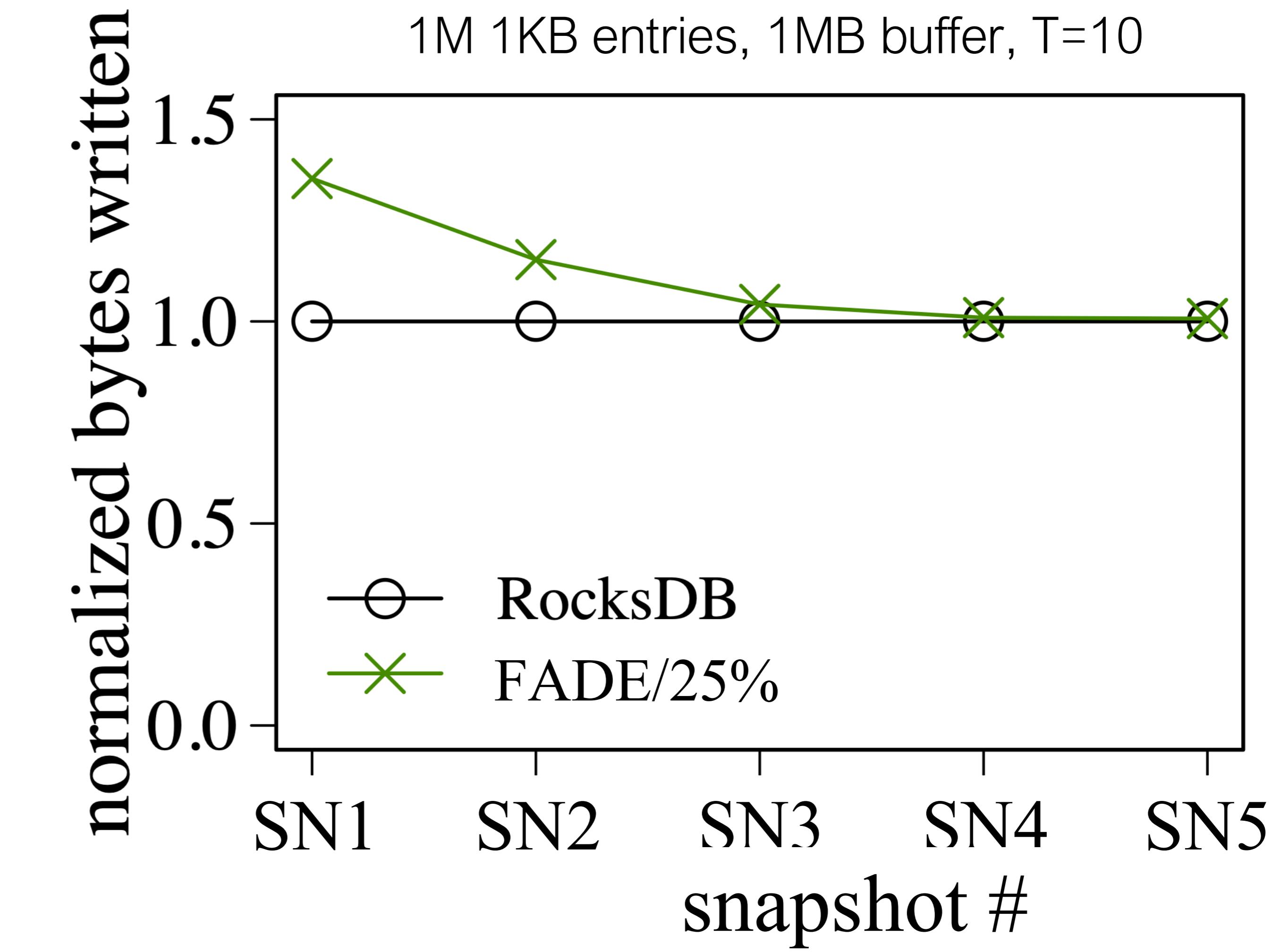
FAst DElete

- higher write amplification
4 - 25%
 - improved read performance
1.2 - 1.4x
 - reduced space amplification
2.1 - 9.8x
 - timely delete persistence
within D_{th}
- ↑ ✓ ✓ ✓

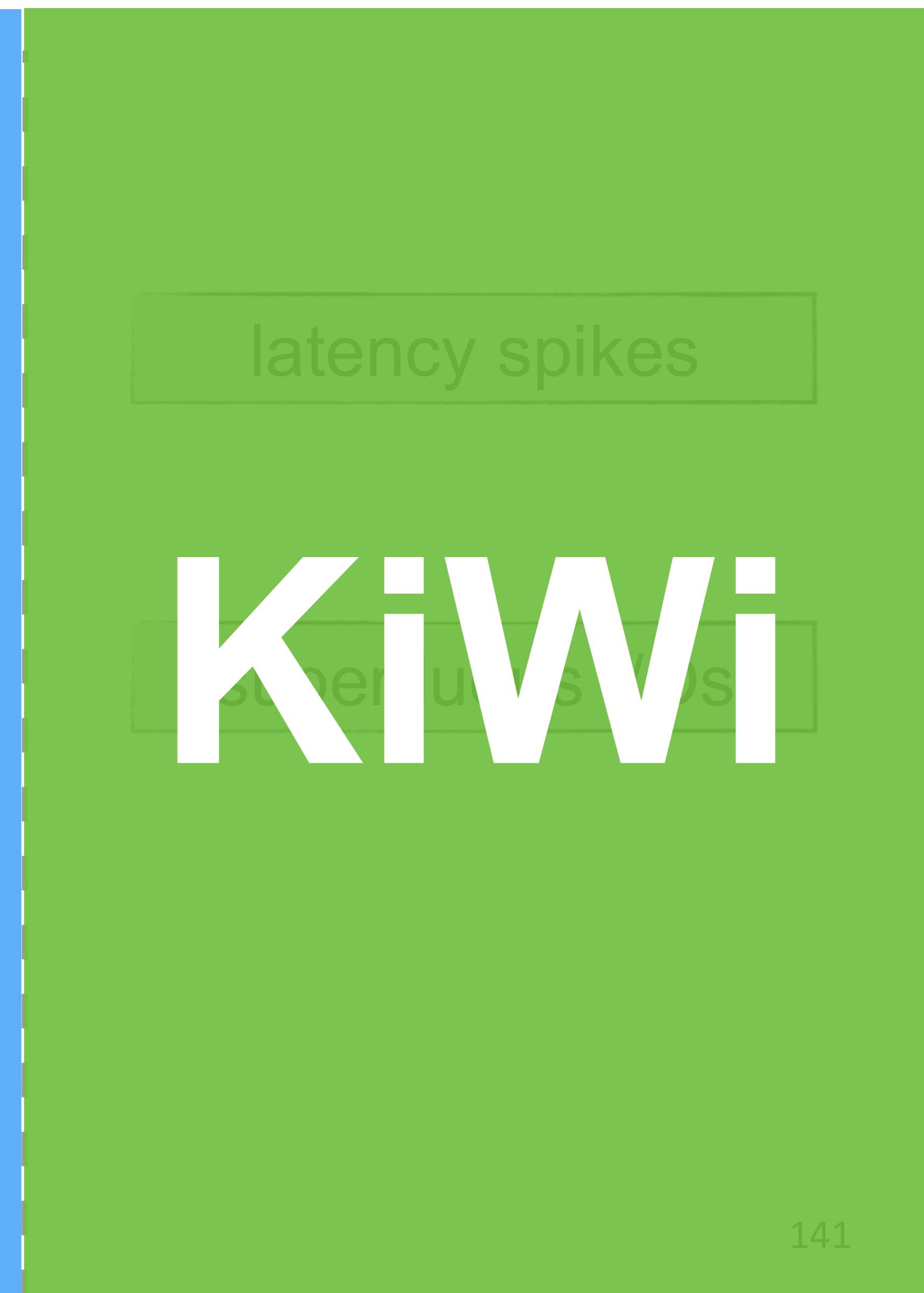
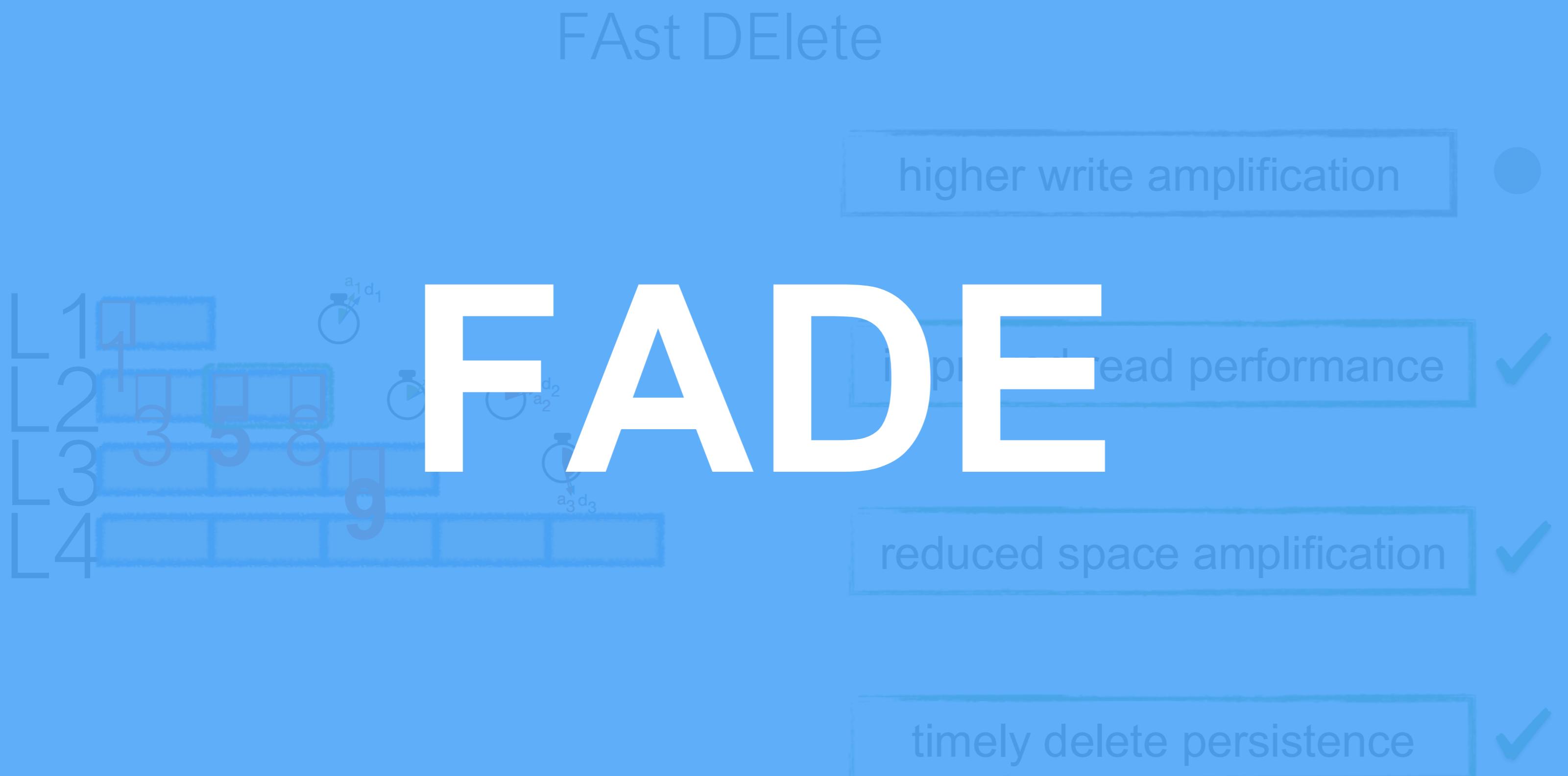


FAst DElete

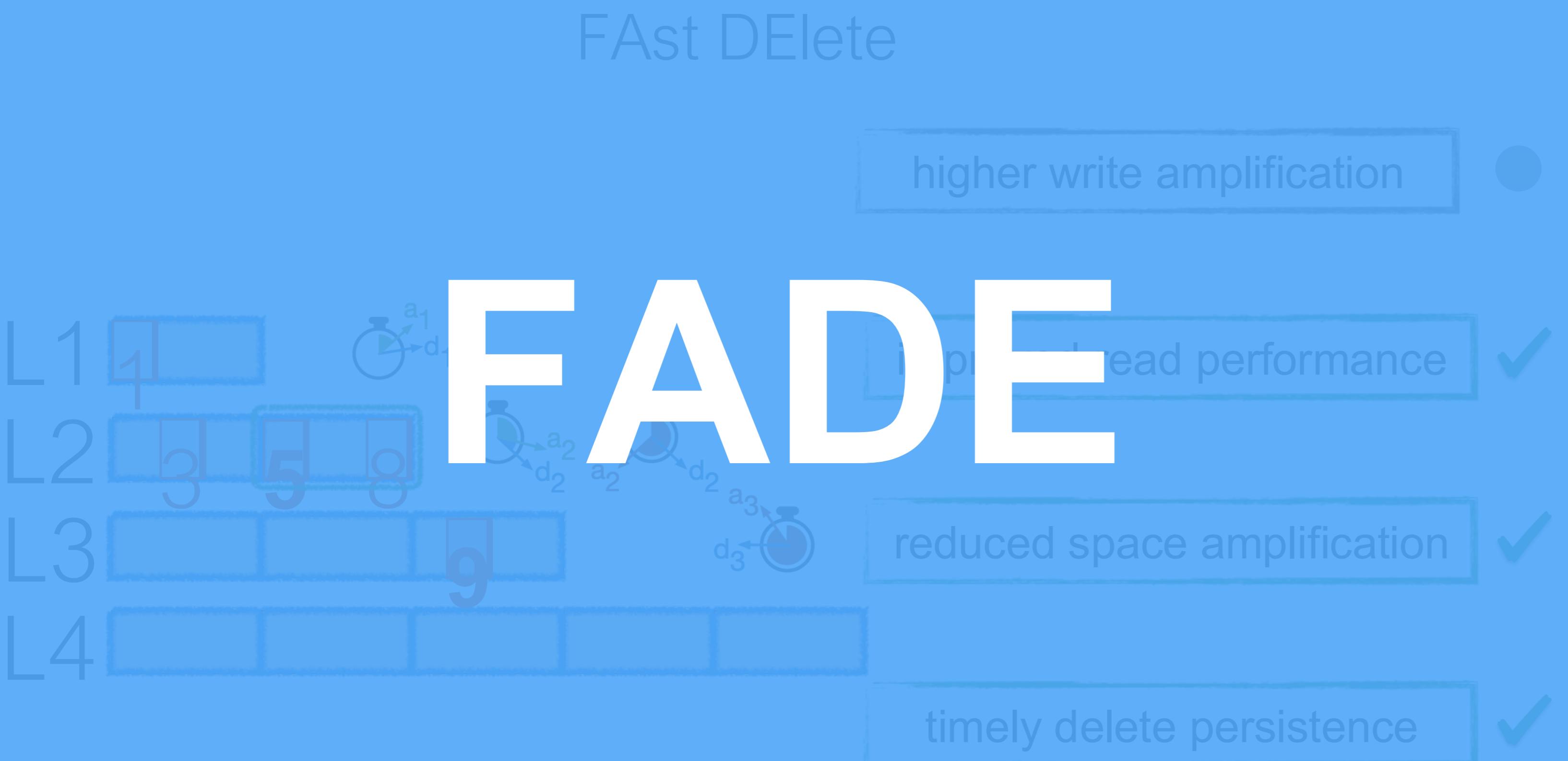
- higher write amplification
0.7 %
- improved read performance
1.2 - 1.4x
- reduced space amplification
2.1 - 9.8x
- timely delete persistence
within D_{th}



the solution



the solution

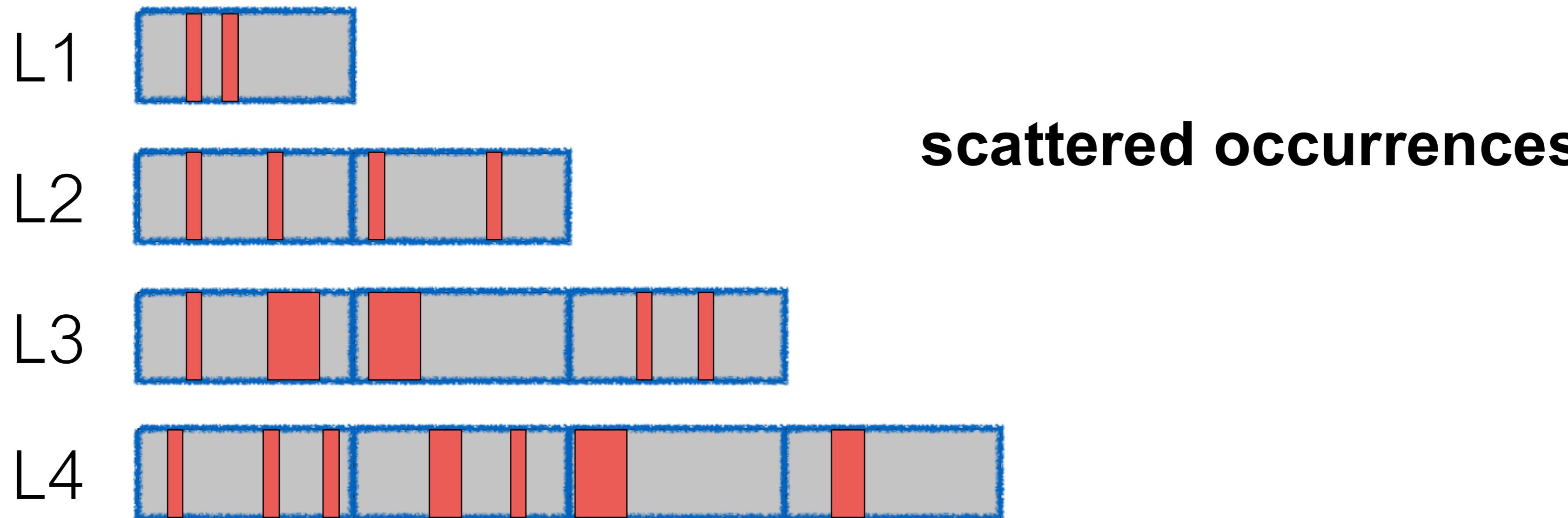


latency spikes

KiWi

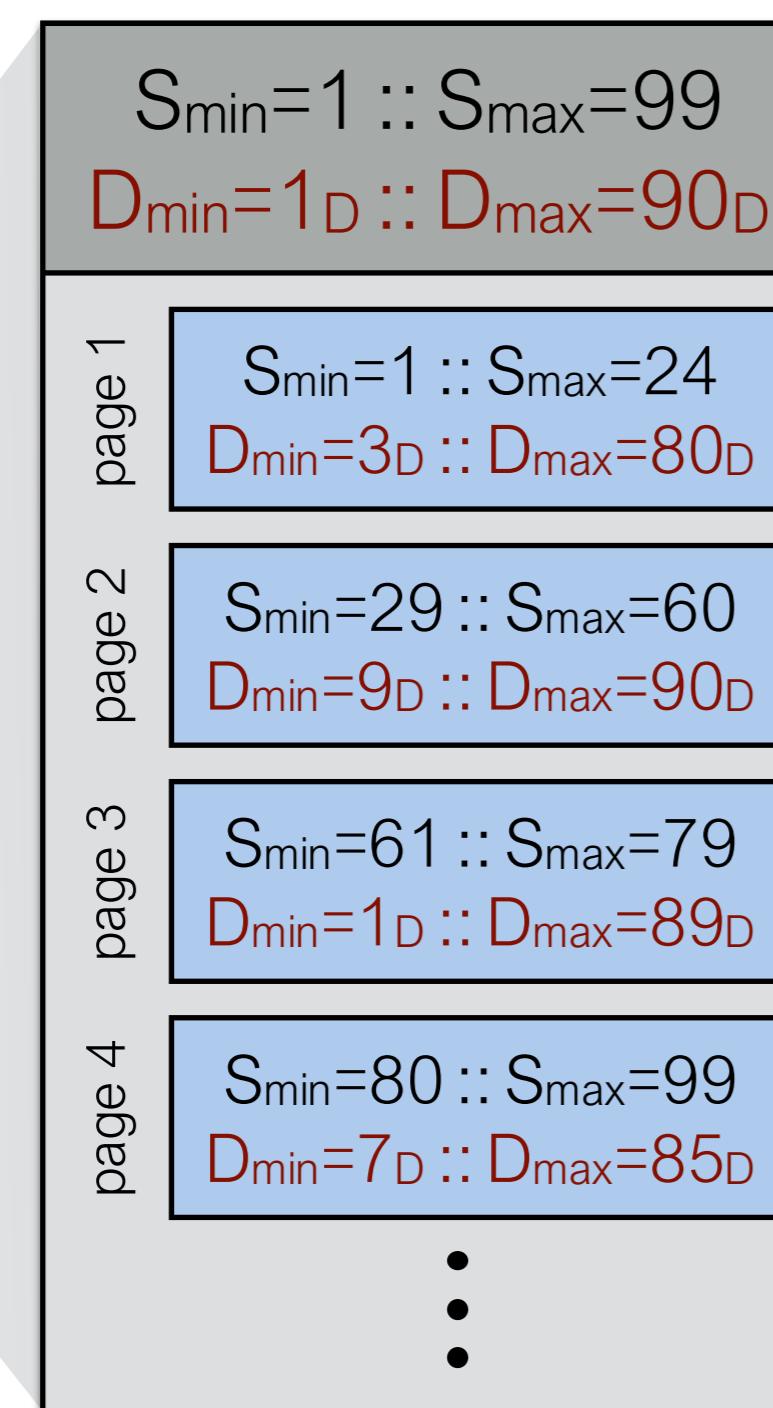
Key Weaving storage layout

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



Key Weaving storage layout

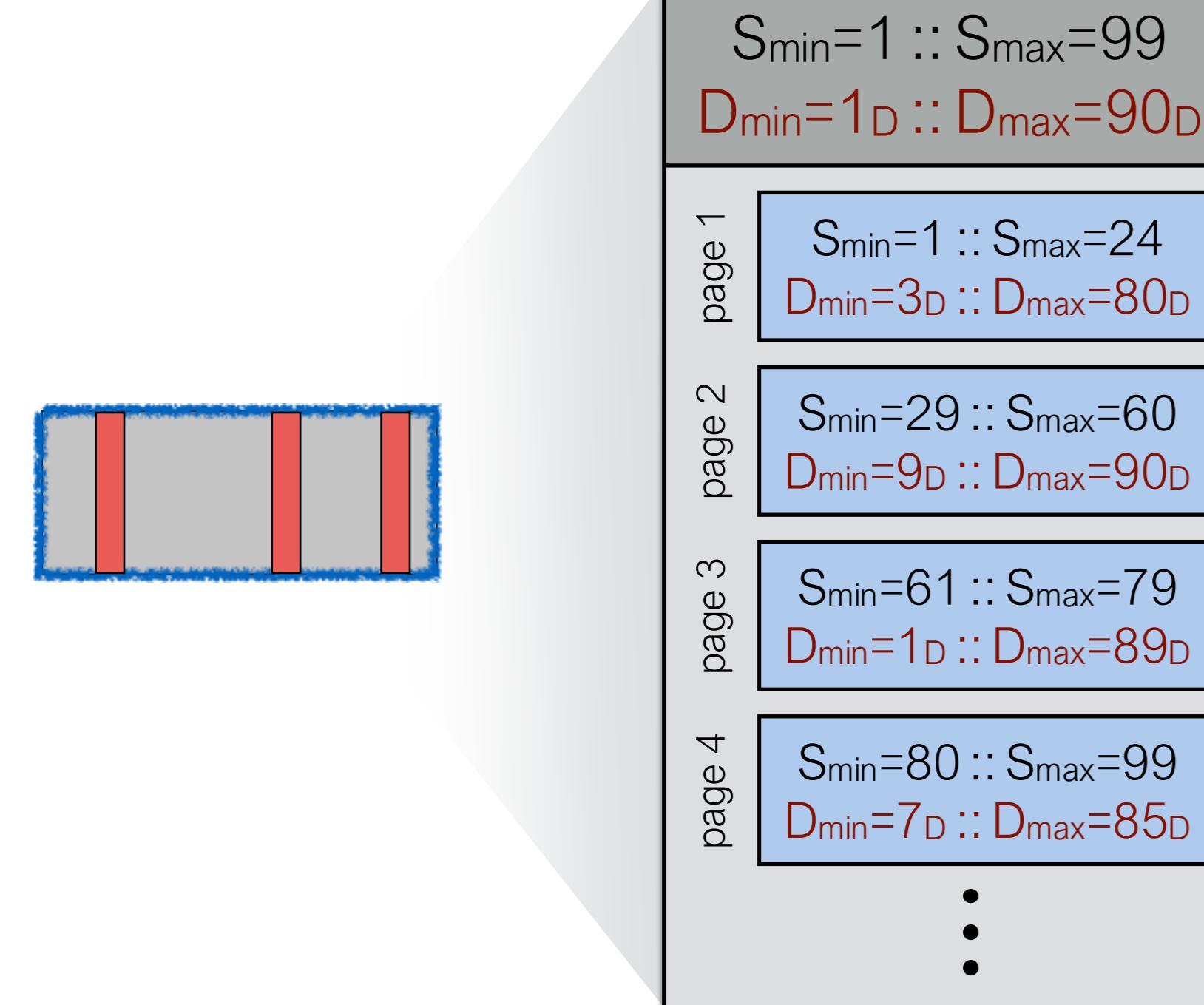
delete all entries with timestamp $\leq 65_D$



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 $\leq 65_D$



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

1 I/O

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

1 I/O

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

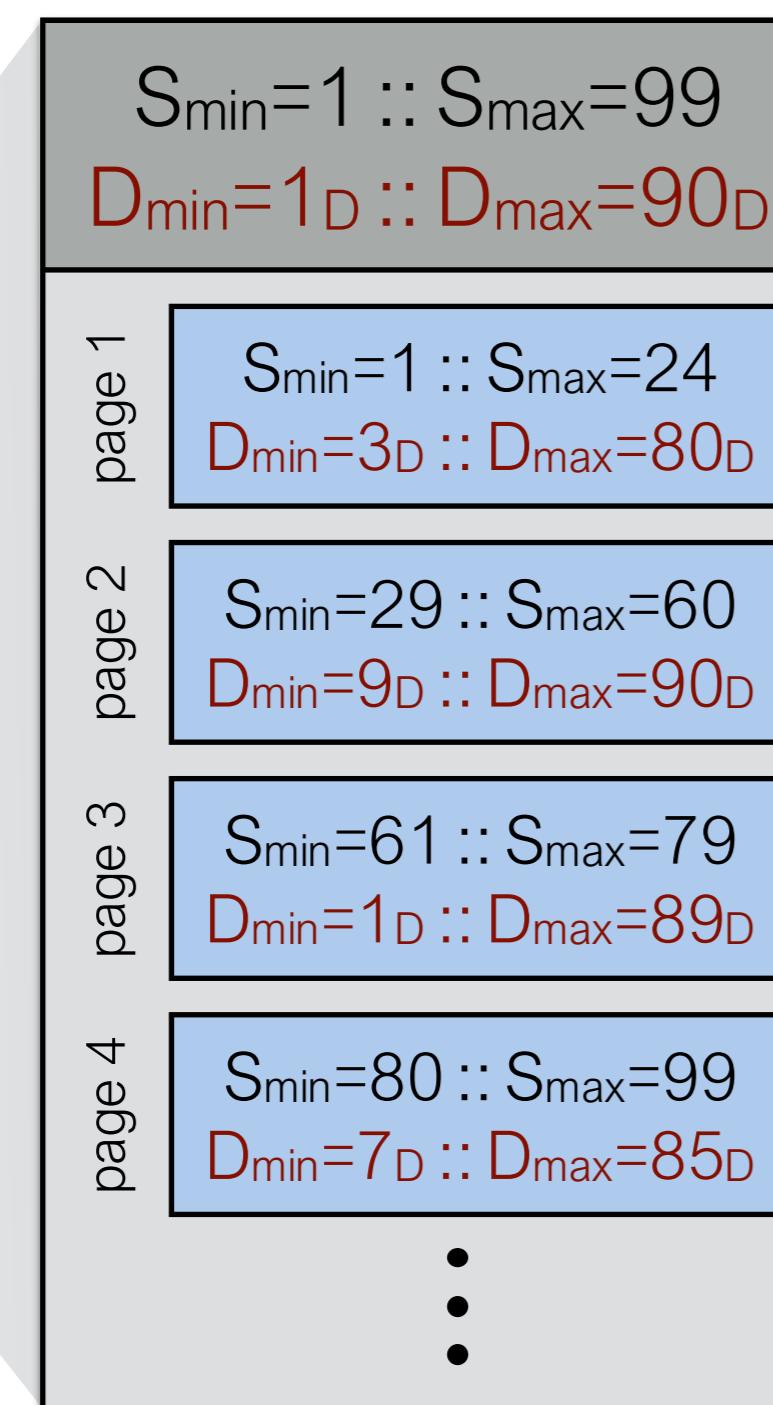
1 I/O

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

1 I/O

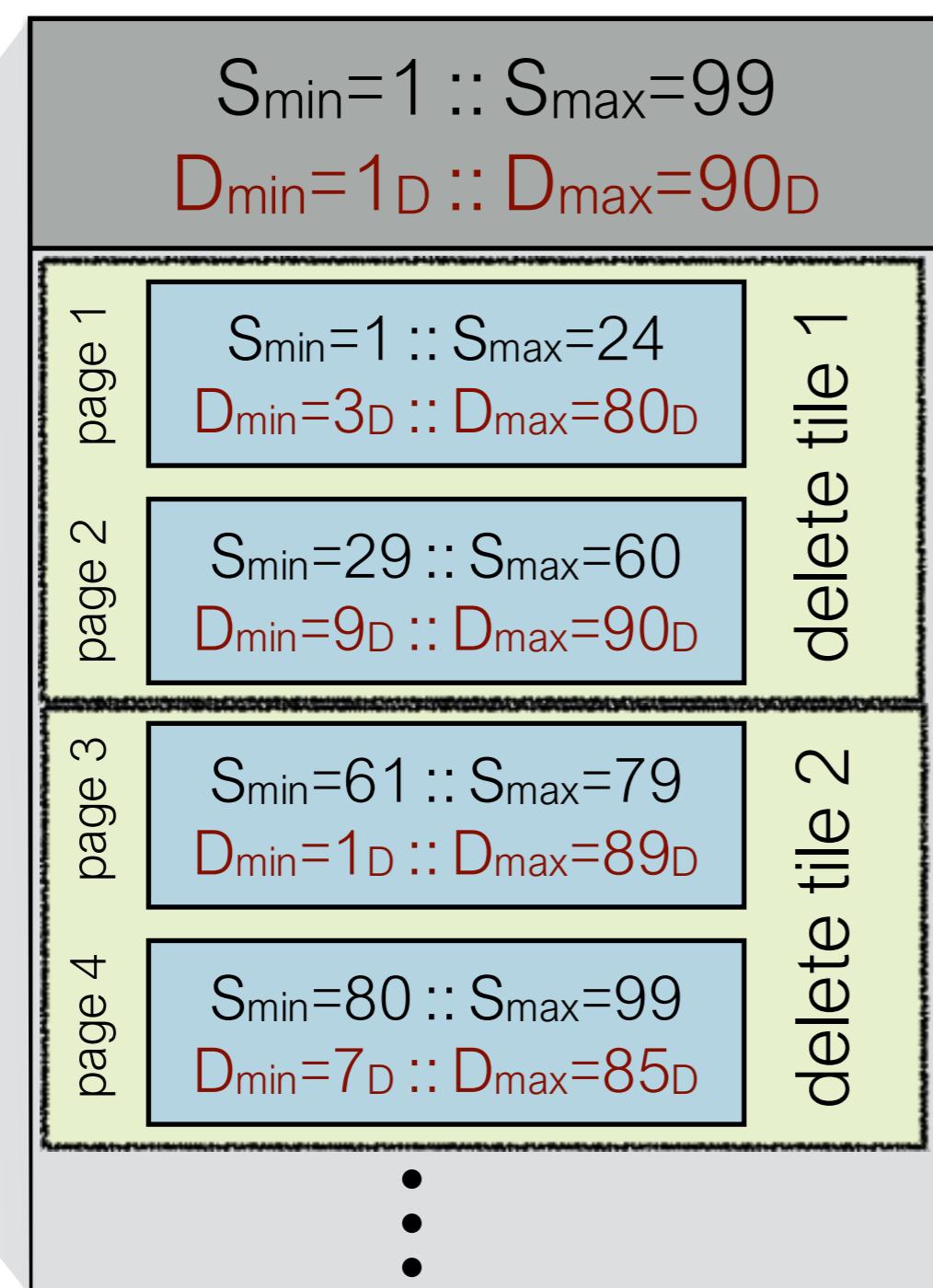
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$



Key Weaving storage layout

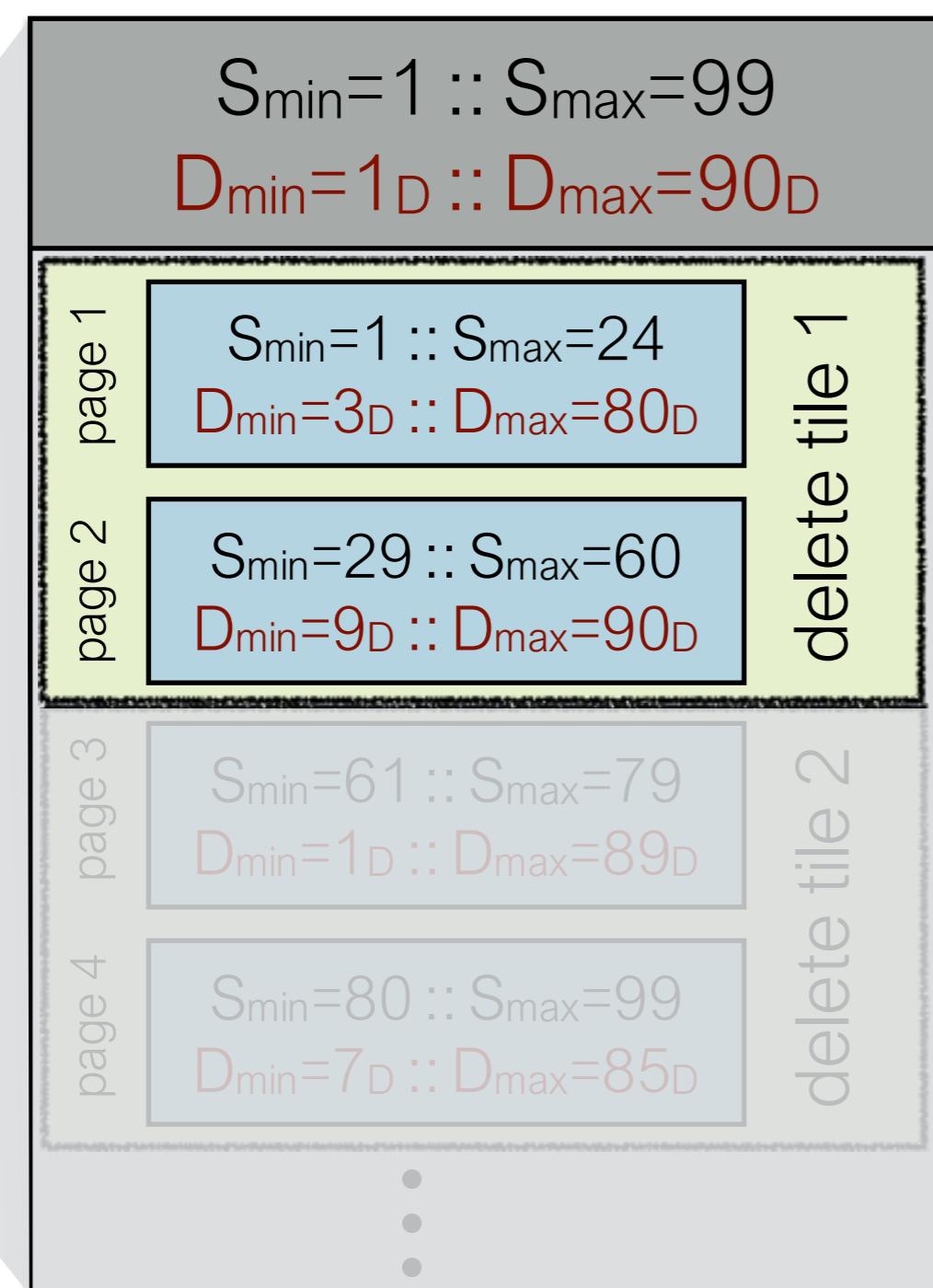
delete all entries with timestamp $\leq 65_D$



partitioned on S

Key Weaving storage layout

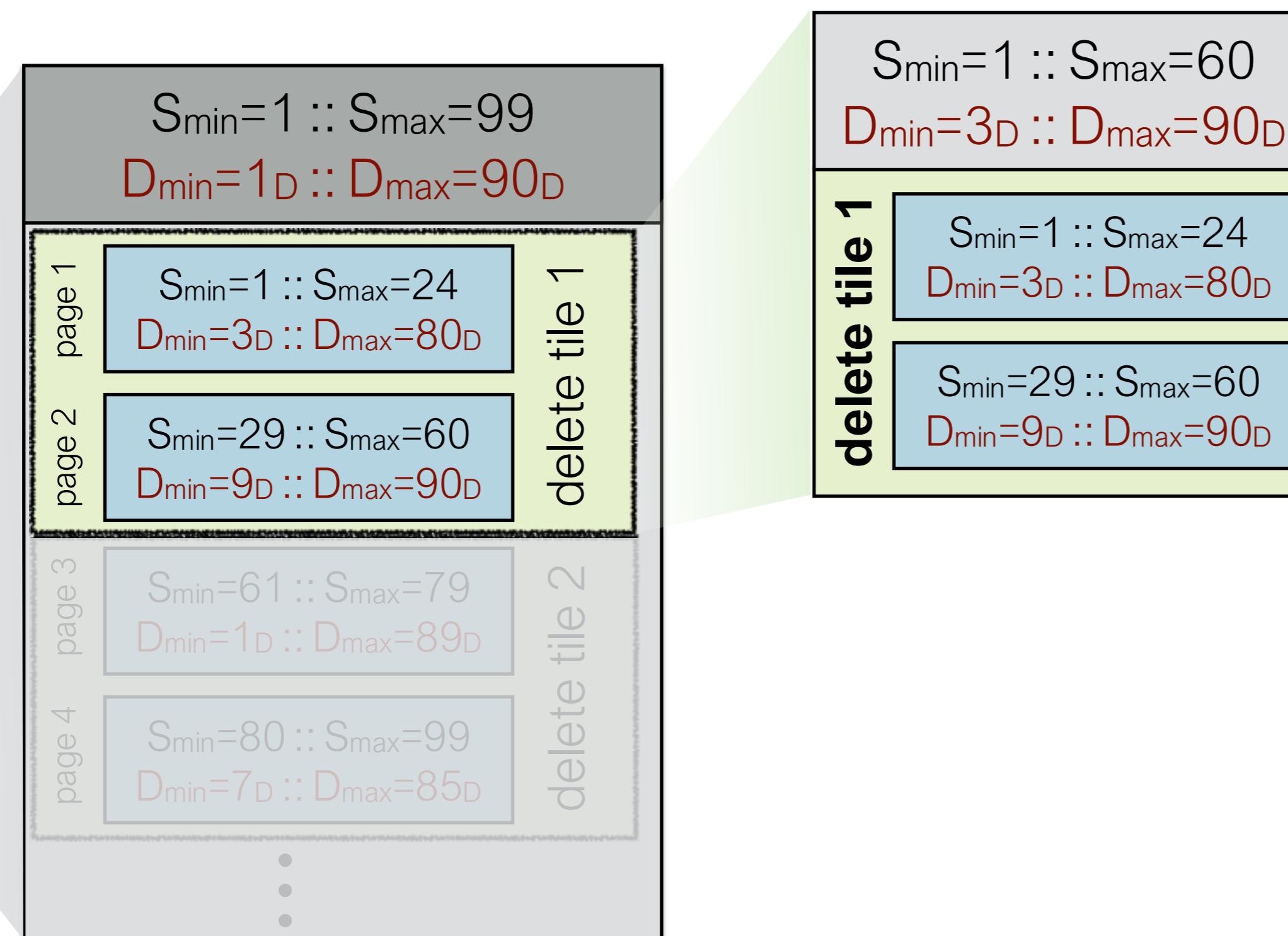
delete all entries with timestamp $\leq 65D$



partitioned on S

Key Weaving storage layout

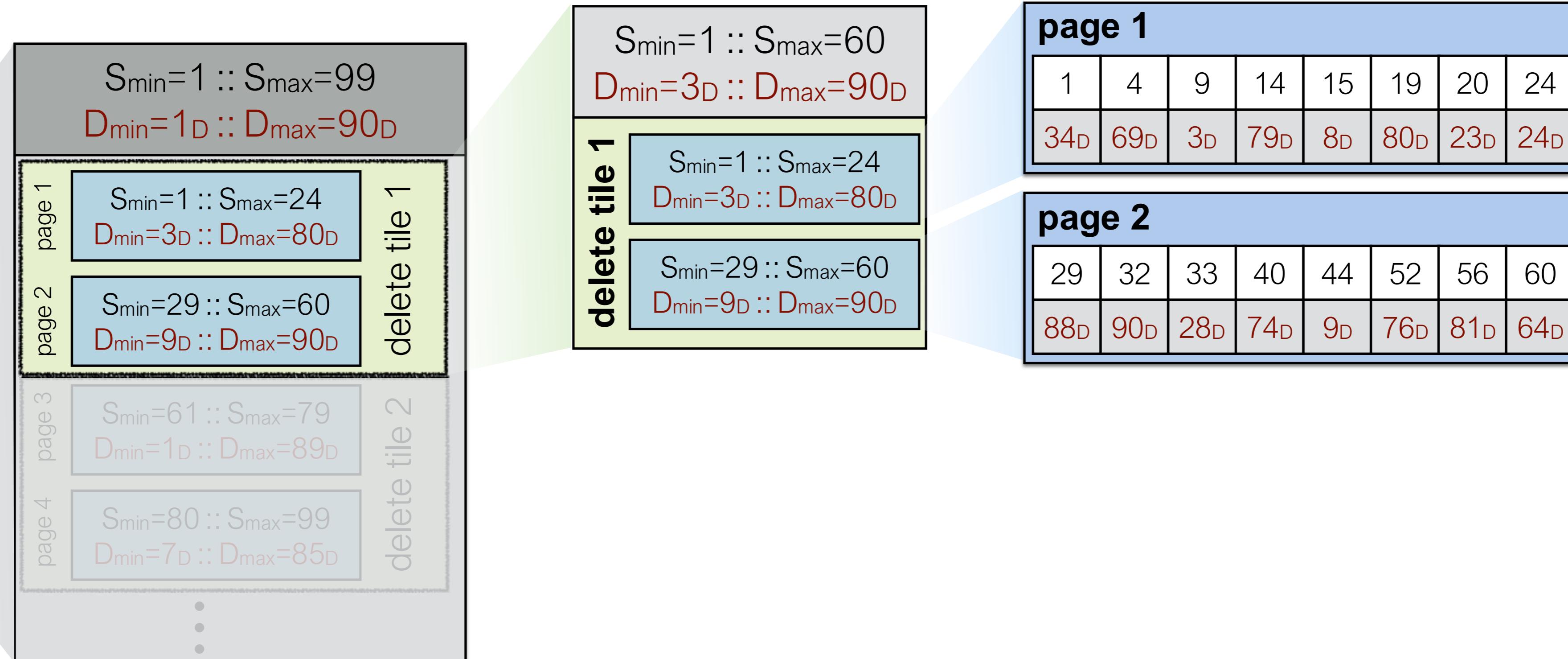
delete all entries with timestamp $\leq 65_D$



partitioned on S

Key Weaving storage layout

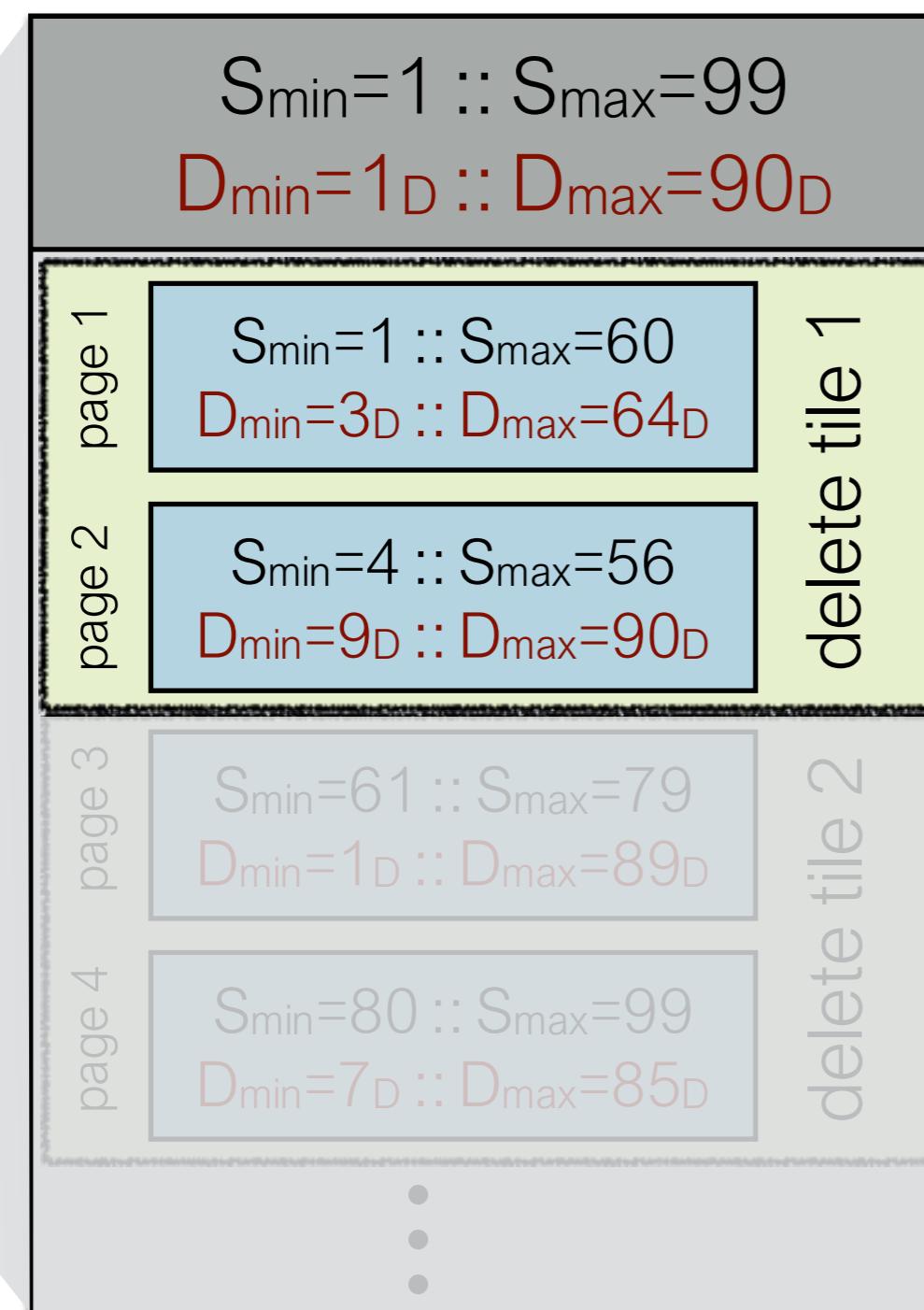
delete all entries with timestamp $\leq 65_D$



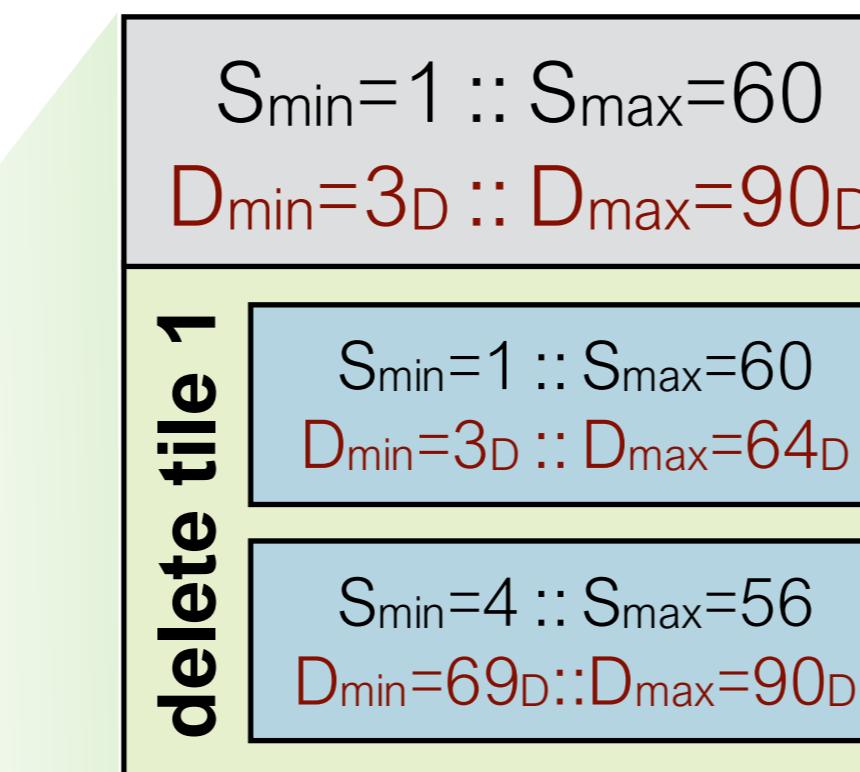
partitioned on S

Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$



partitioned on S

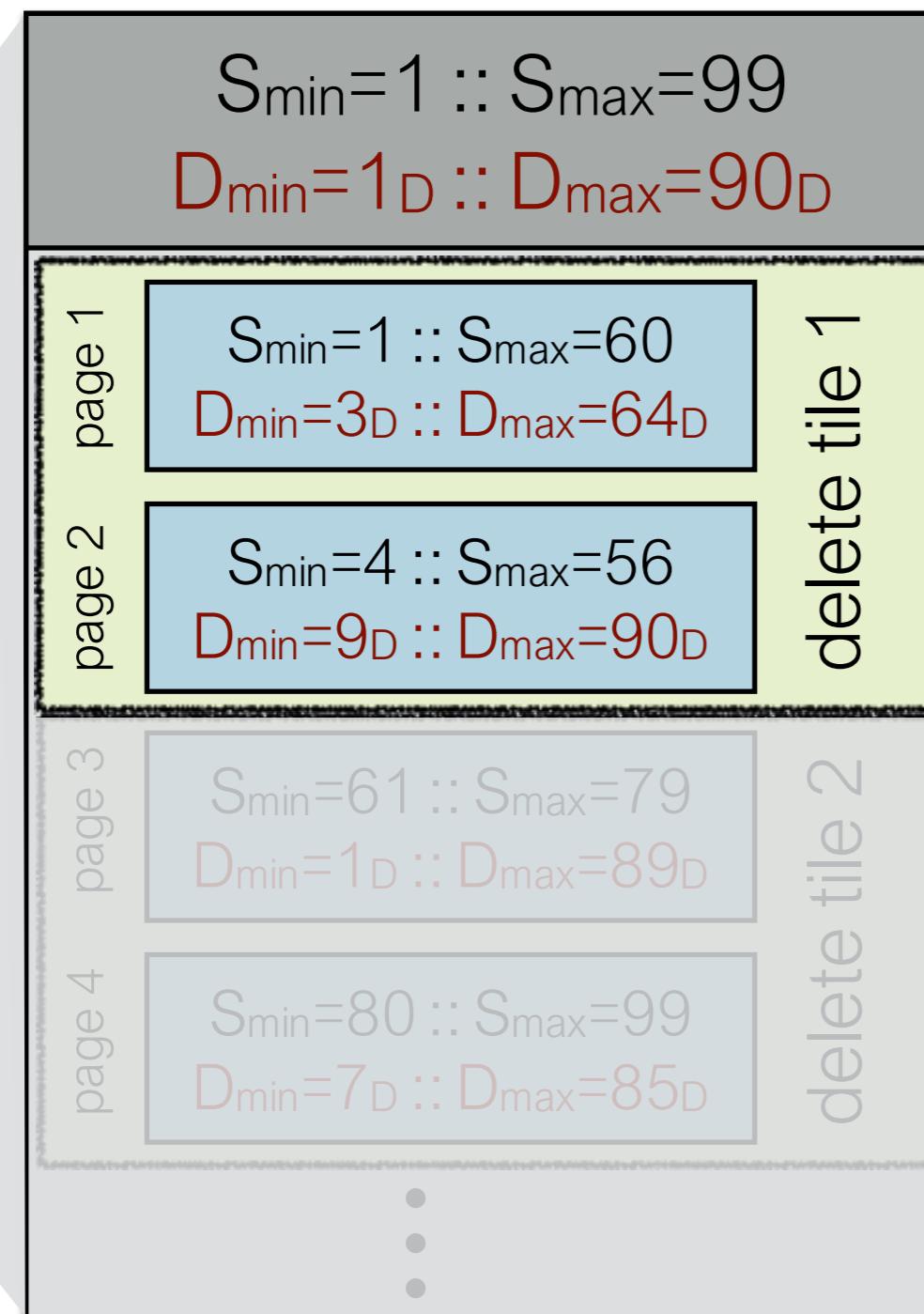


partitioned on D

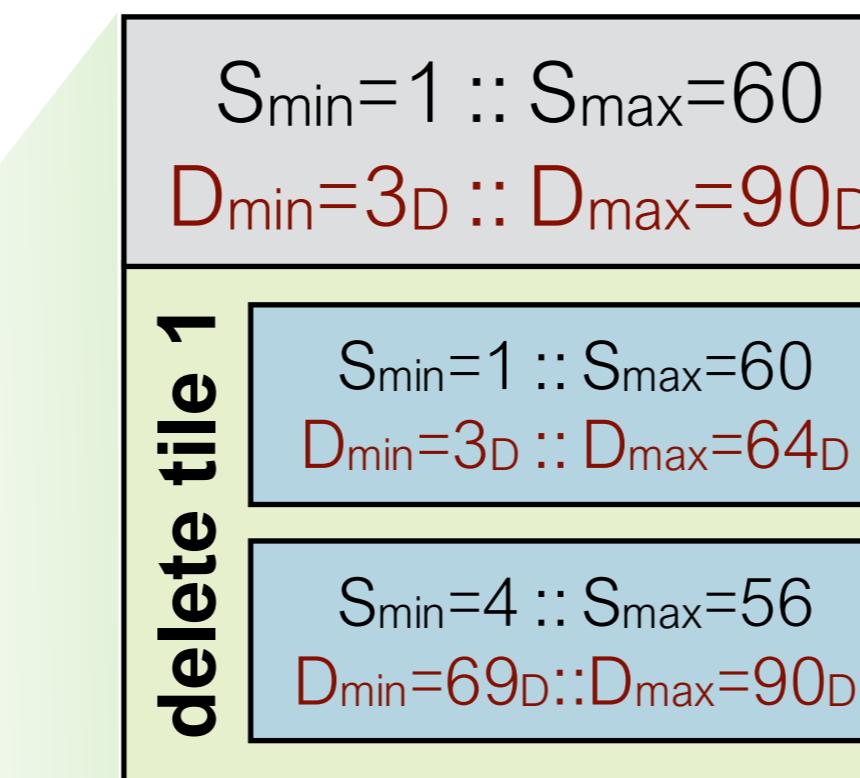
page 1							
9	15	44	20	24	33	1	60
3_D	8_D	9_D	23_D	24_D	28_D	34_D	64_D
page 2							
4	40	52	14	19	56	29	32
69_D	74_D	76_D	79_D	80_D	81_D	88_D	90_D

Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$



partitioned on S



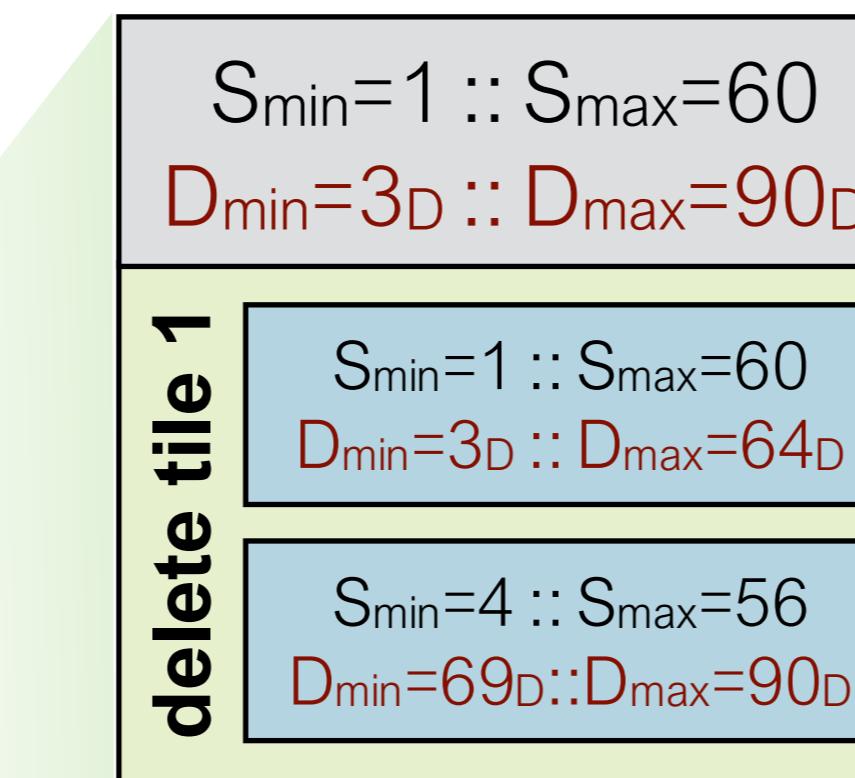
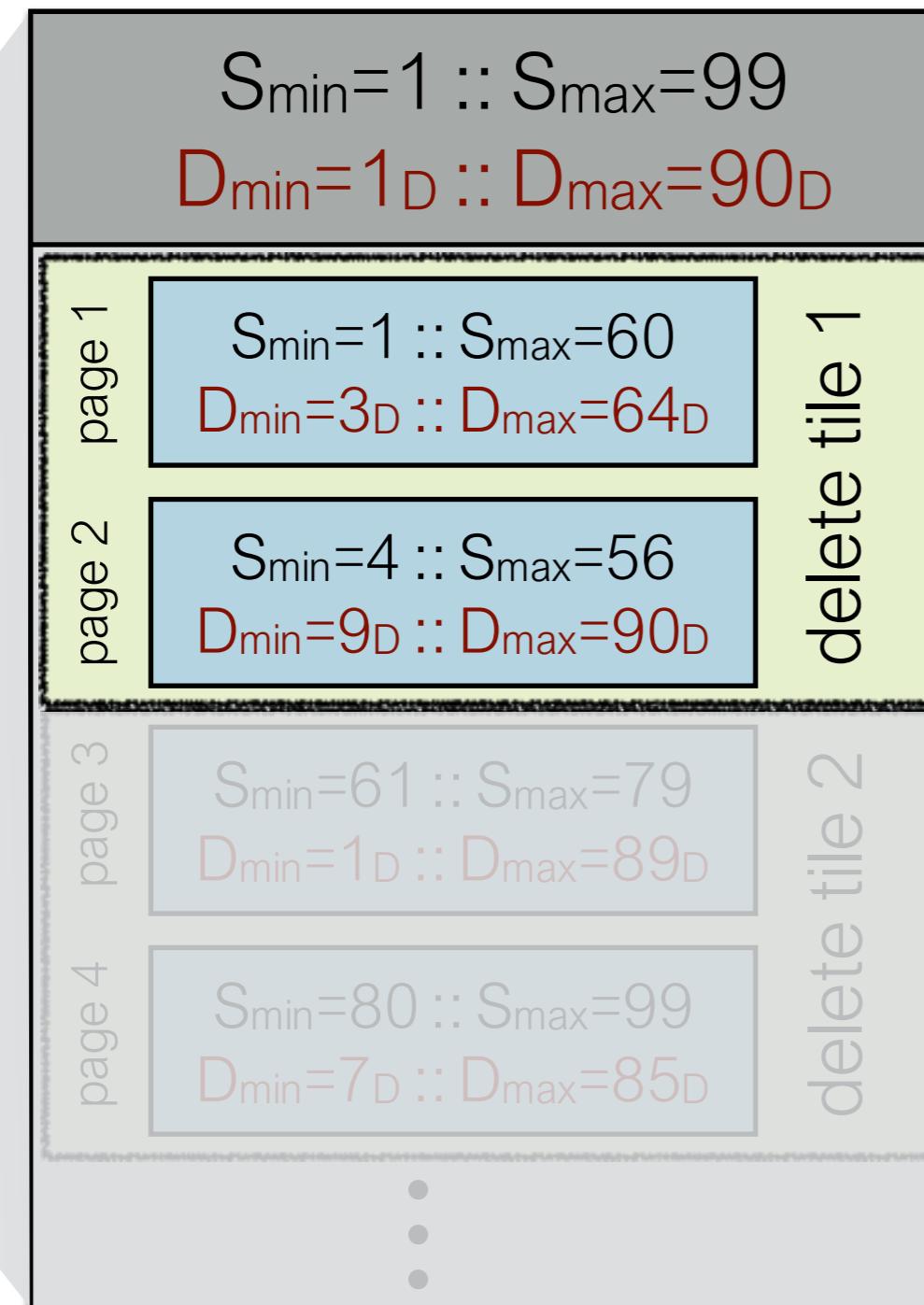
partitioned on D

Two pages are shown. **page 1** has a blue header and contains 8 slots. The first slot (row 1, col 1) contains 9 (red) and 3_D (red). The second slot (row 1, col 2) contains 15 (red) and 8_D (red). The third slot (row 1, col 3) contains 44 (black) and 9_D (red). The fourth slot (row 1, col 4) contains 20 (black) and 23_D (red). The fifth slot (row 1, col 5) contains 24 (black) and 24_D (red). The sixth slot (row 1, col 6) contains 33 (black) and 28_D (red). The seventh slot (row 1, col 7) contains 1 (black) and 34_D (red). The eighth slot (row 1, col 8) contains 60 (black) and 64_D (red). **page 2** has a blue header and contains 8 slots. The first slot (row 1, col 1) contains 4 (black) and 69_D (red). The second slot (row 1, col 2) contains 40 (black) and 74_D (red). The third slot (row 1, col 3) contains 52 (black) and 76_D (red). The fourth slot (row 1, col 4) contains 14 (black) and 79_D (red). The fifth slot (row 1, col 5) contains 19 (black) and 80_D (red). The sixth slot (row 1, col 6) contains 56 (black) and 81_D (red). The seventh slot (row 1, col 7) contains 29 (black) and 88_D (red). The eighth slot (row 1, col 8) contains 32 (black) and 90_D (red).

drop
page

Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$



sorted on S

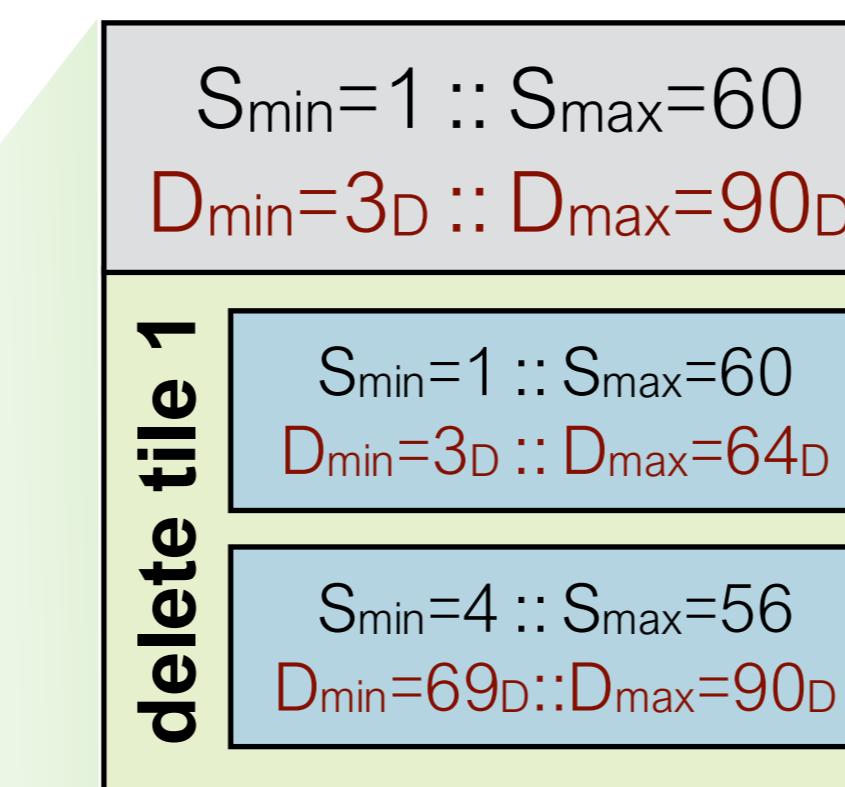
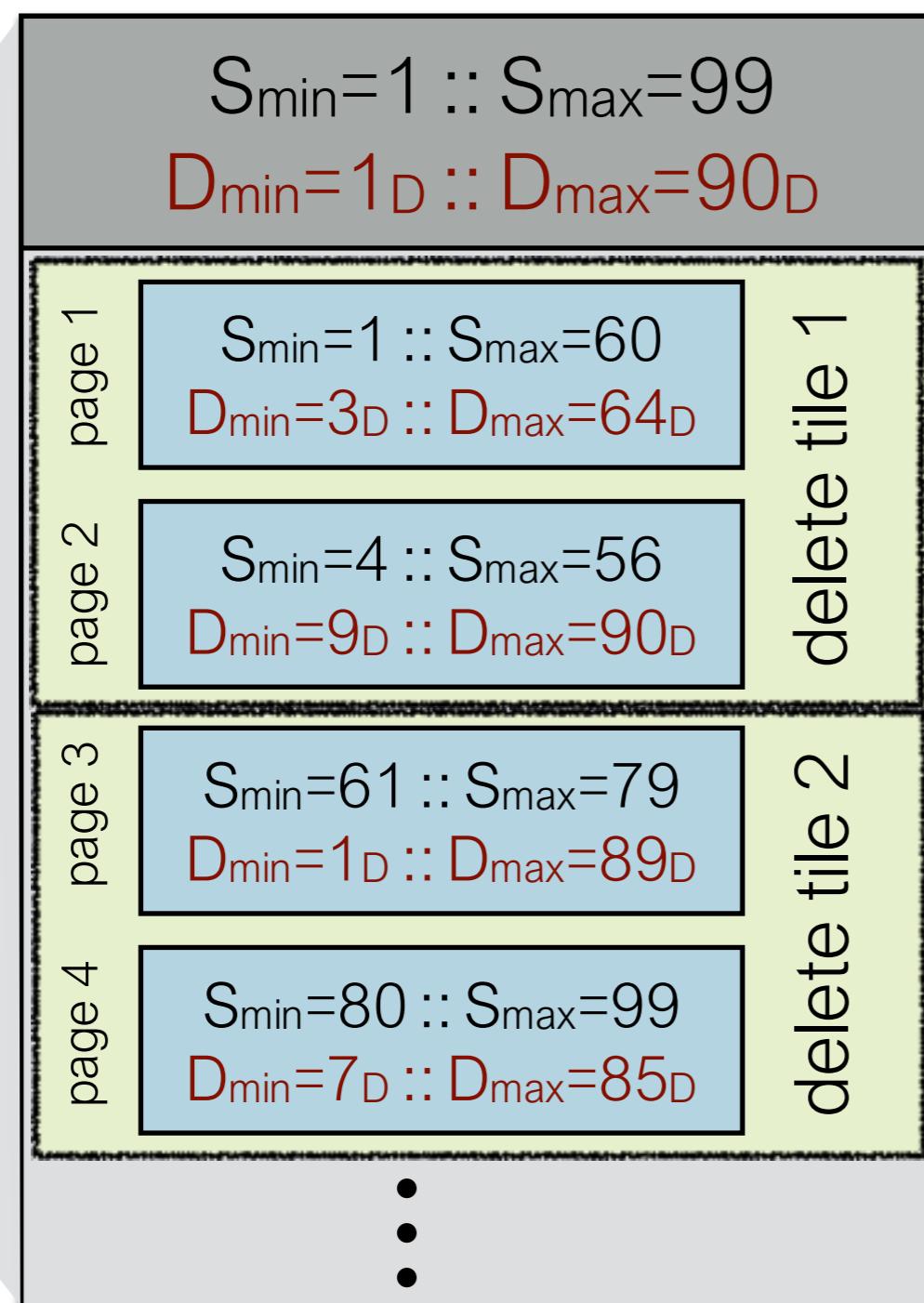
page 1							
1	9	15	20	24	33	44	60
34 _D	3 _D	8 _D	23 _D	24 _D	28 _D	9 _D	64 _D

page 2							
4	14	19	29	32	40	52	56
69 _D	79 _D	80 _D	88 _D	90 _D	74 _D	76 _D	81 _D

drop page

Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$



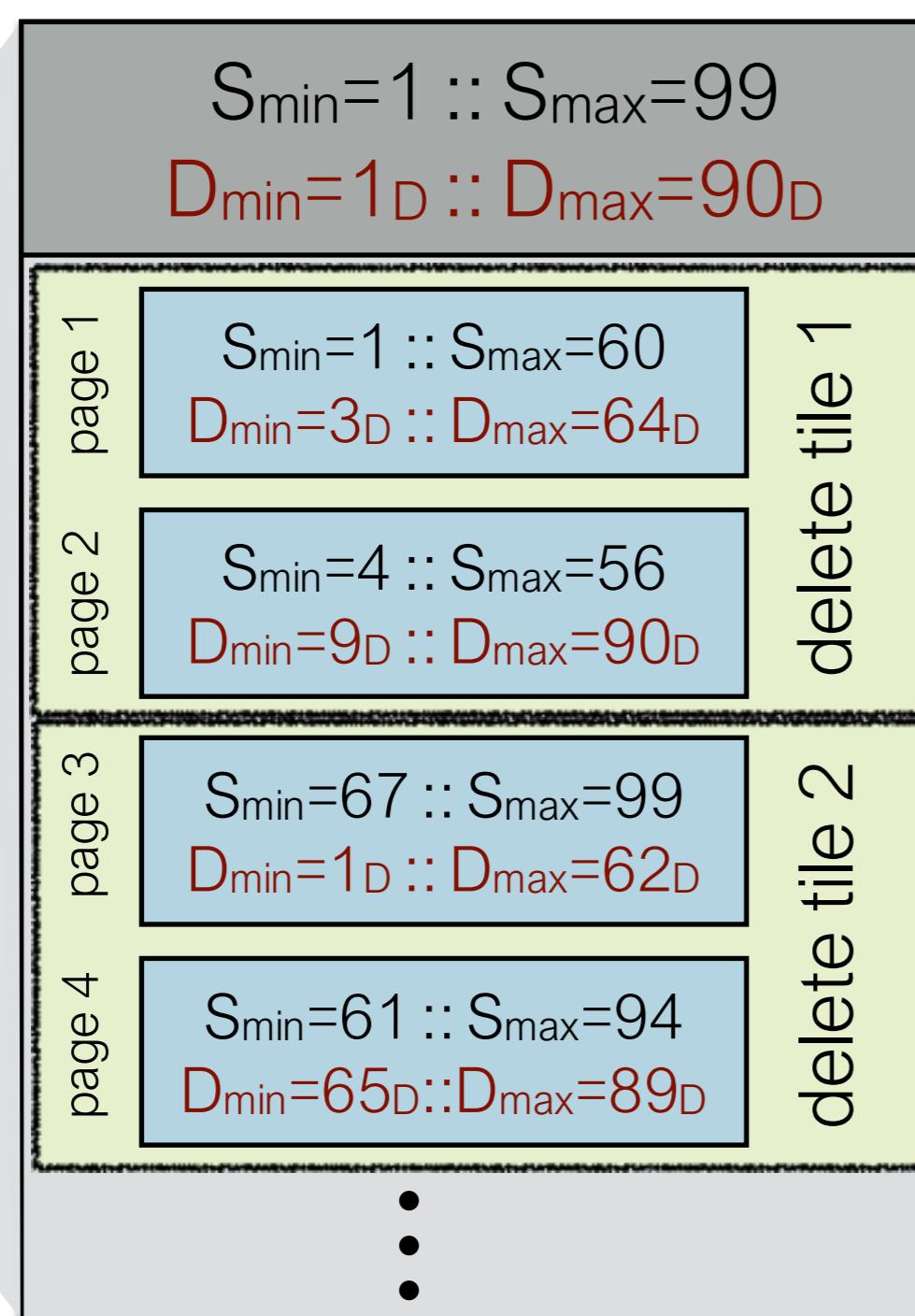
sorted on S

drop page

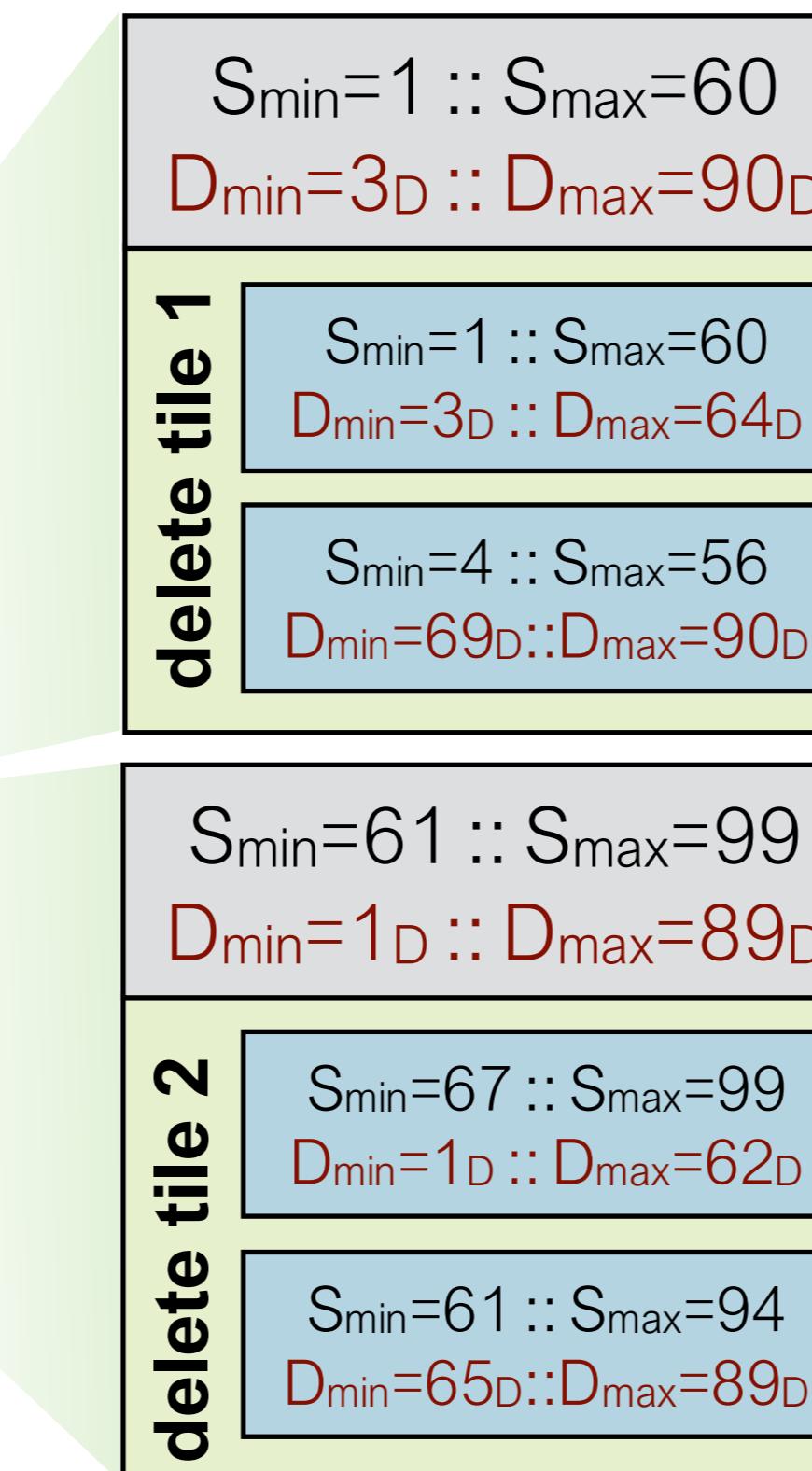
page	entry	value
1	1	1
1	9	9
1	15	15
1	20	20
1	24	24
1	33	33
1	44	44
1	60	60
1	34 _D	34 _D
1	3 _D	3 _D
1	8 _D	8 _D
1	23 _D	23 _D
1	24 _D	24 _D
1	28 _D	28 _D
1	9 _D	9 _D
1	64 _D	64 _D
2	4	4
2	14	14
2	19	19
2	29	29
2	32	32
2	40	40
2	52	52
2	56	56
2	69 _D	69 _D
2	79 _D	79 _D
2	80 _D	80 _D
2	88 _D	88 _D
2	90 _D	90 _D
2	74 _D	74 _D
2	76 _D	76 _D
2	81 _D	81 _D

Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$



partitioned on S



partitioned on D

page 1							
1	9	15	20	24	33	44	60
34_D	3_D	8_D	23_D	24_D	28_D	9_D	64_D

page 2							
4	14	19	29	32	40	52	56
69_D	79_D	80_D	88_D	90_D	74_D	76_D	81_D

page 3							
67	79	84	86	87	91	95	99
1_D	12_D	41_D	62_D	7_D	25_D	59_D	19_D

page 4							
61	63	71	72	73	78	80	94
75_D	82_D	67_D	77_D	89_D	65_D	70_D	85_D

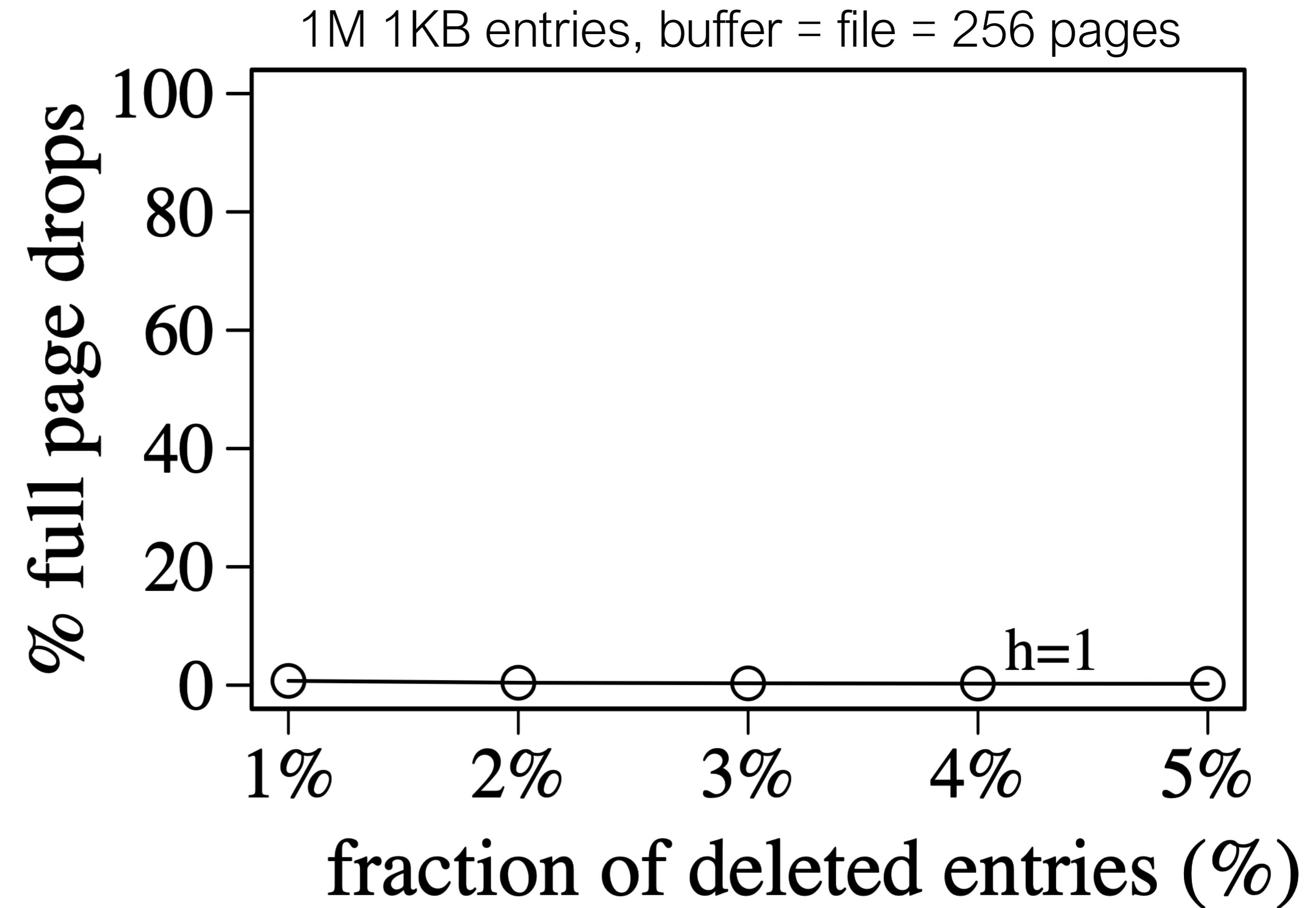
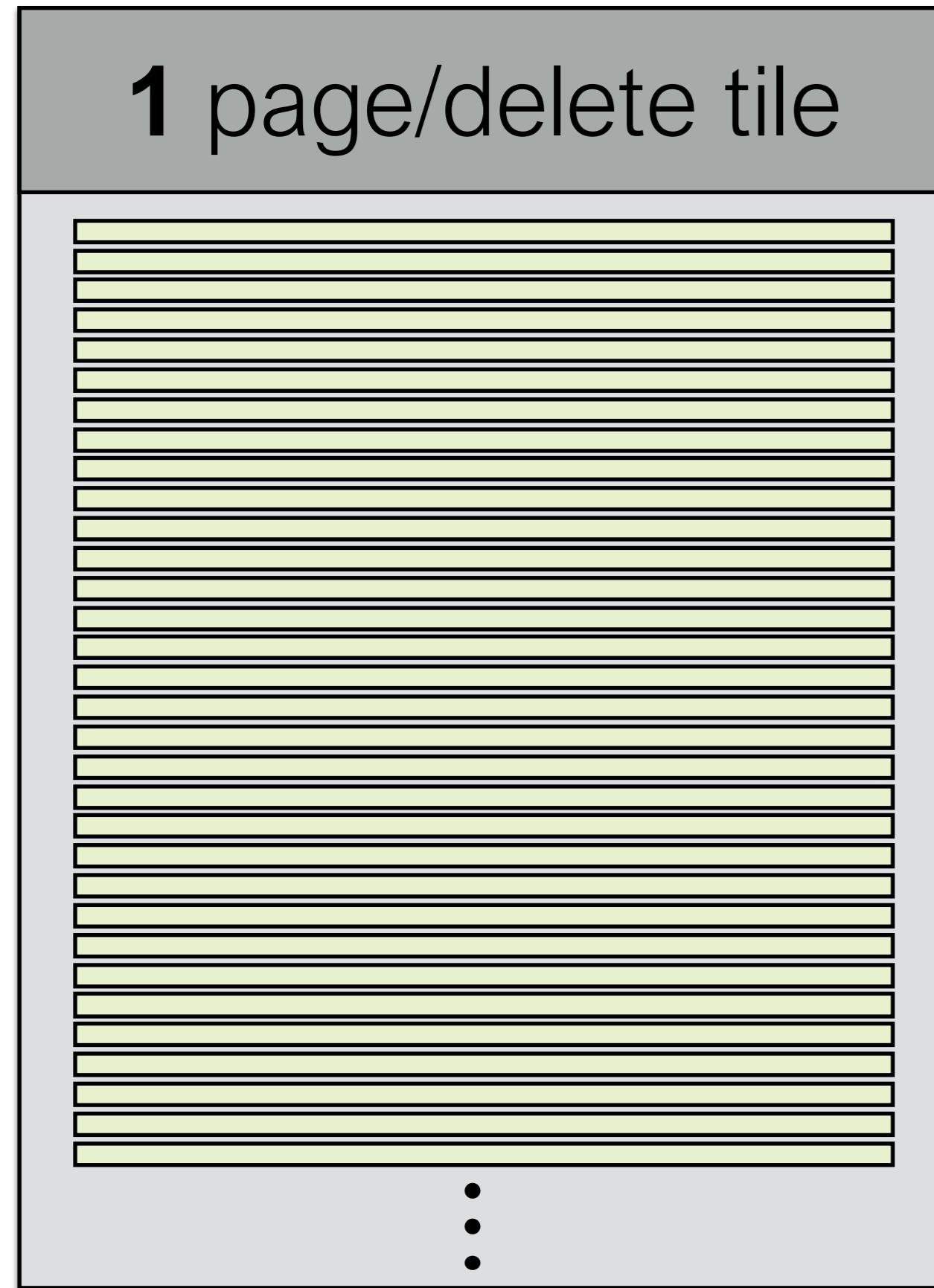
sorted on S

drop
page

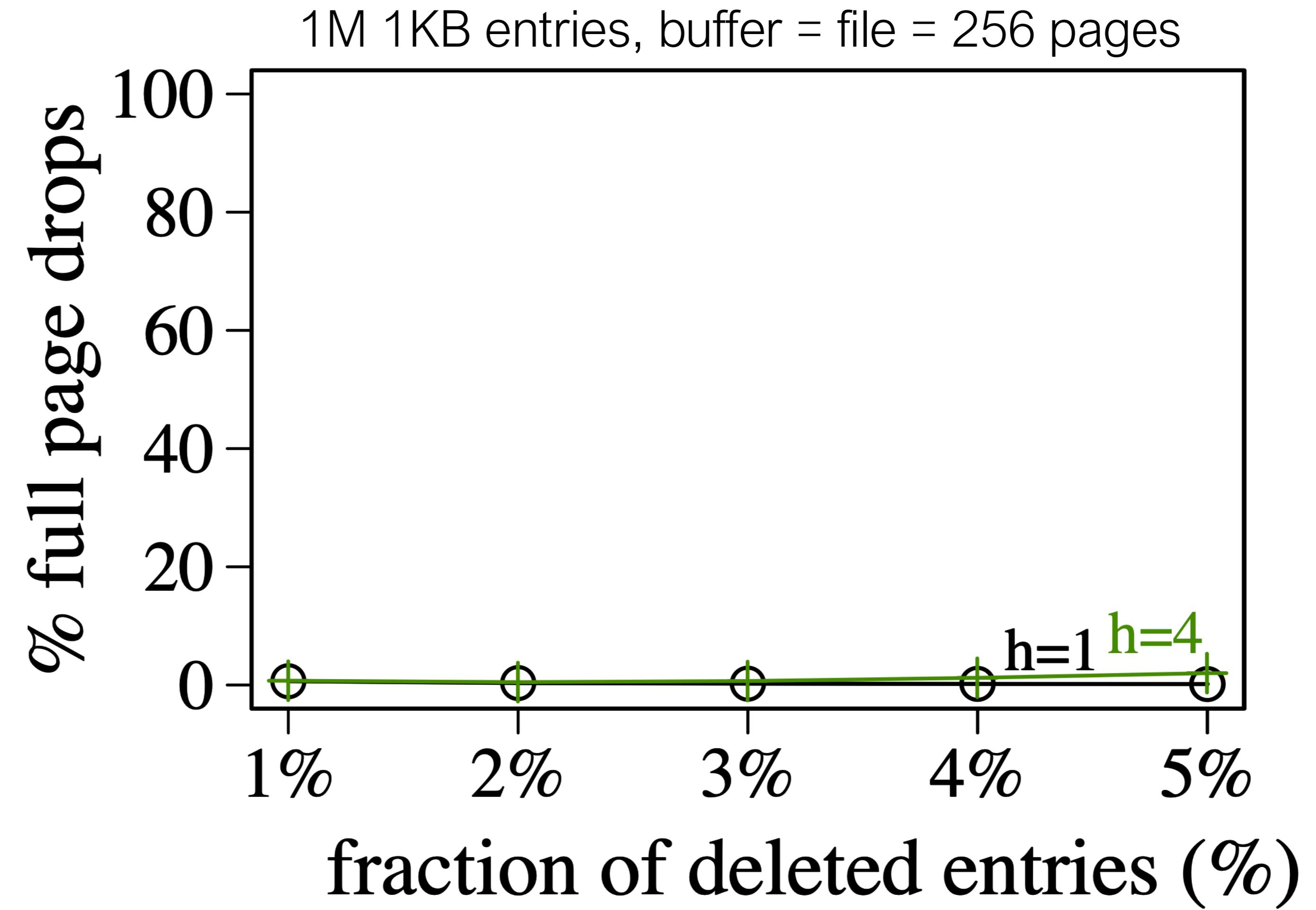
drop
page

1 I/O

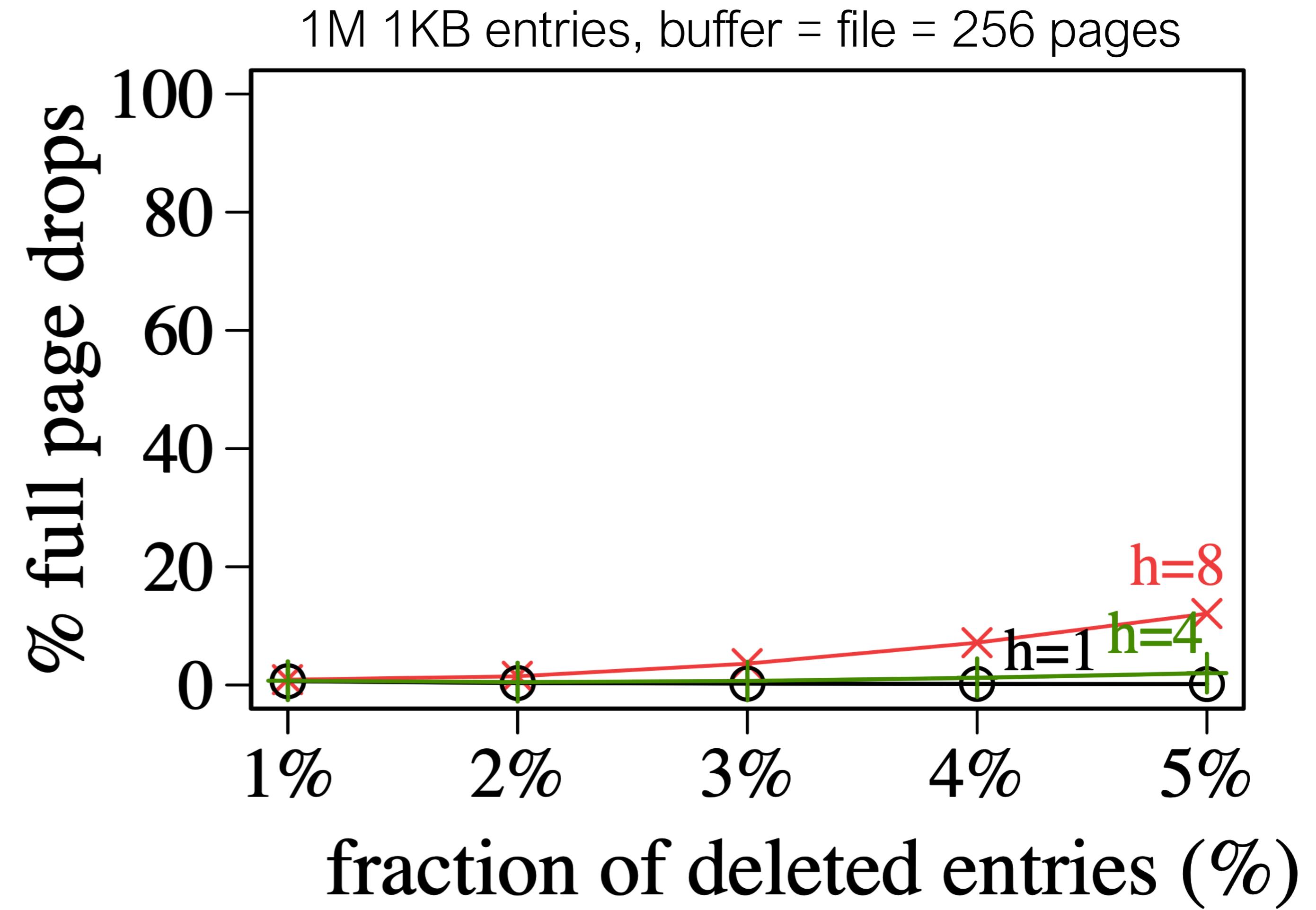
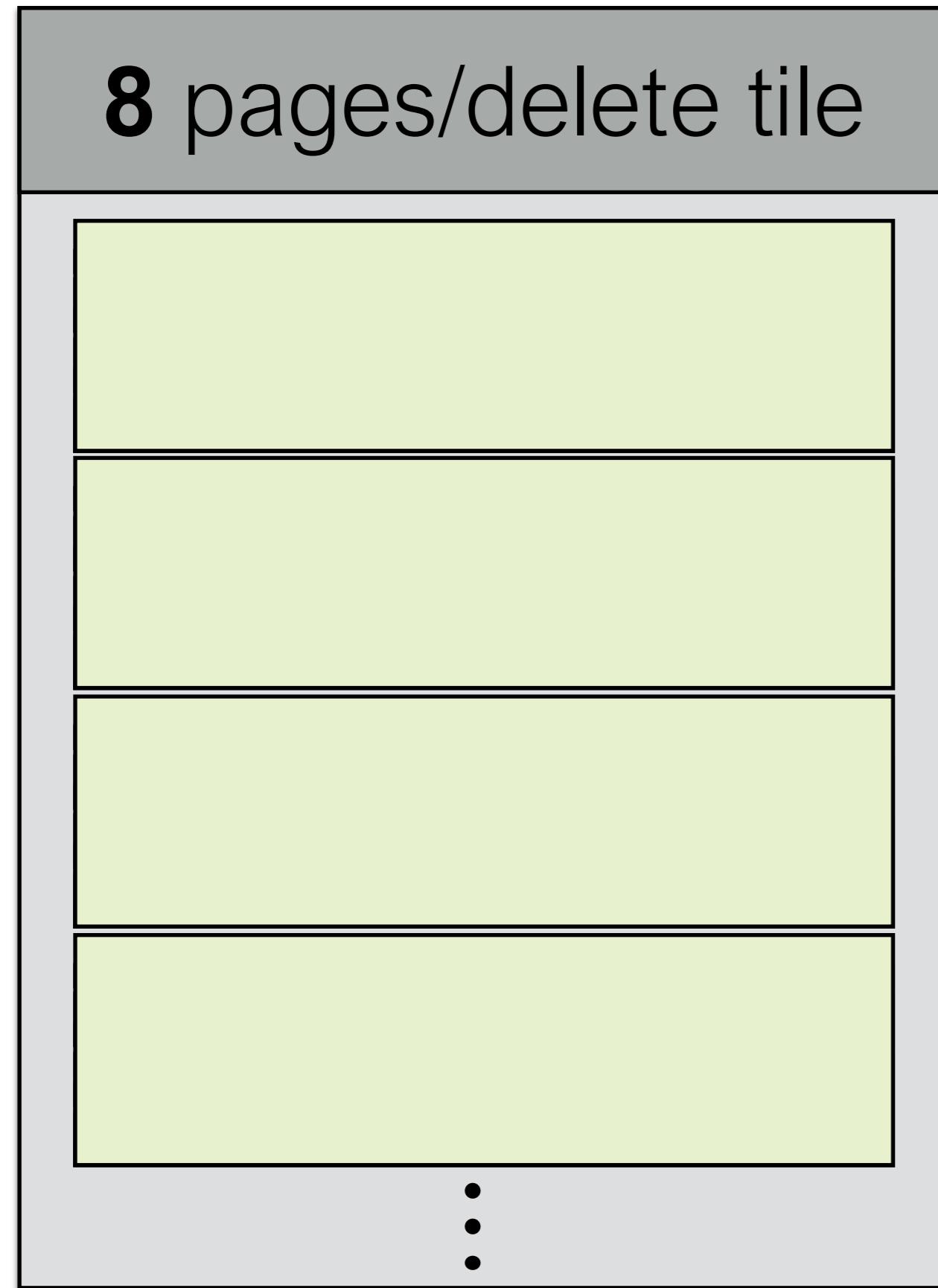
Key Weaving storage layout



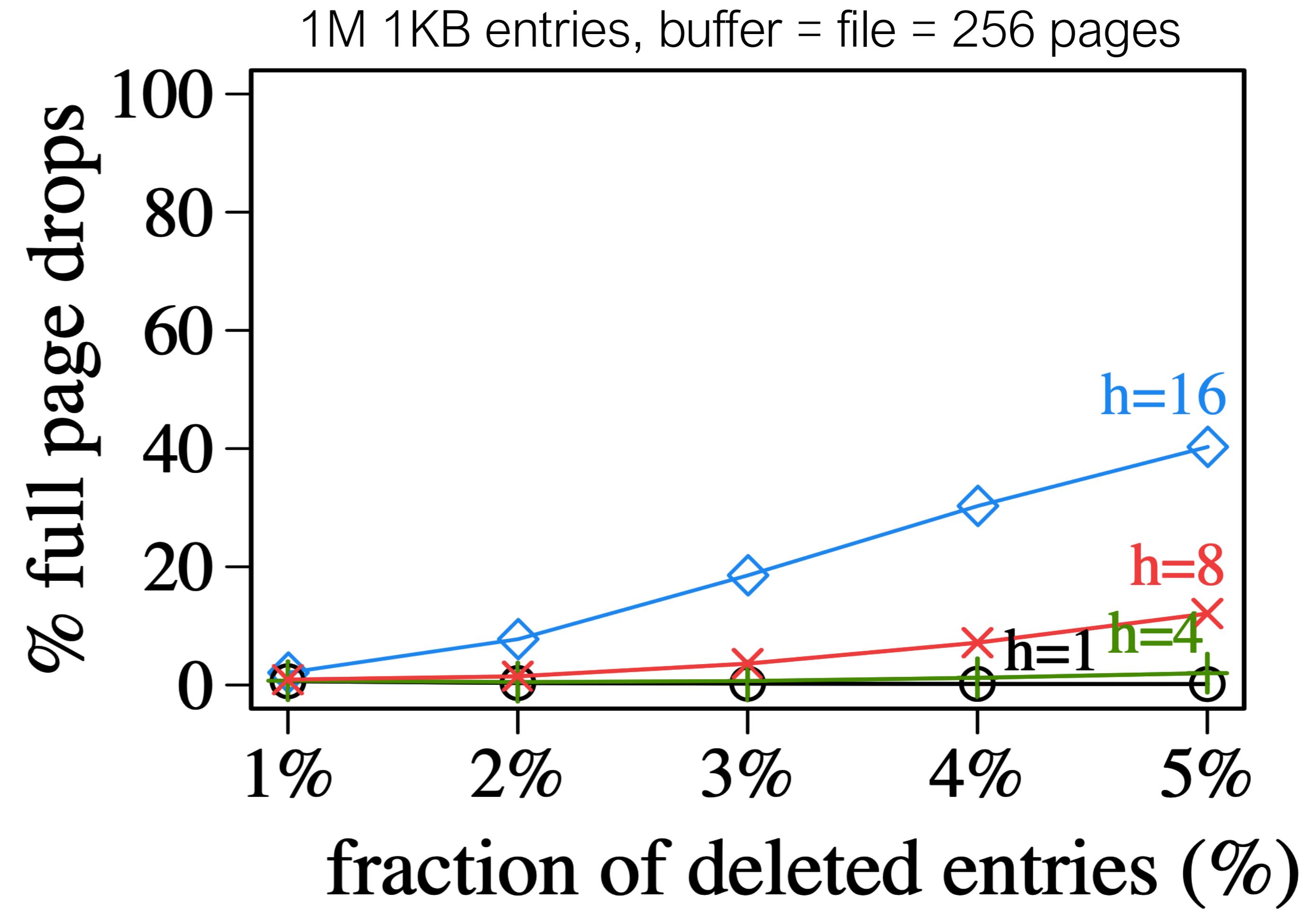
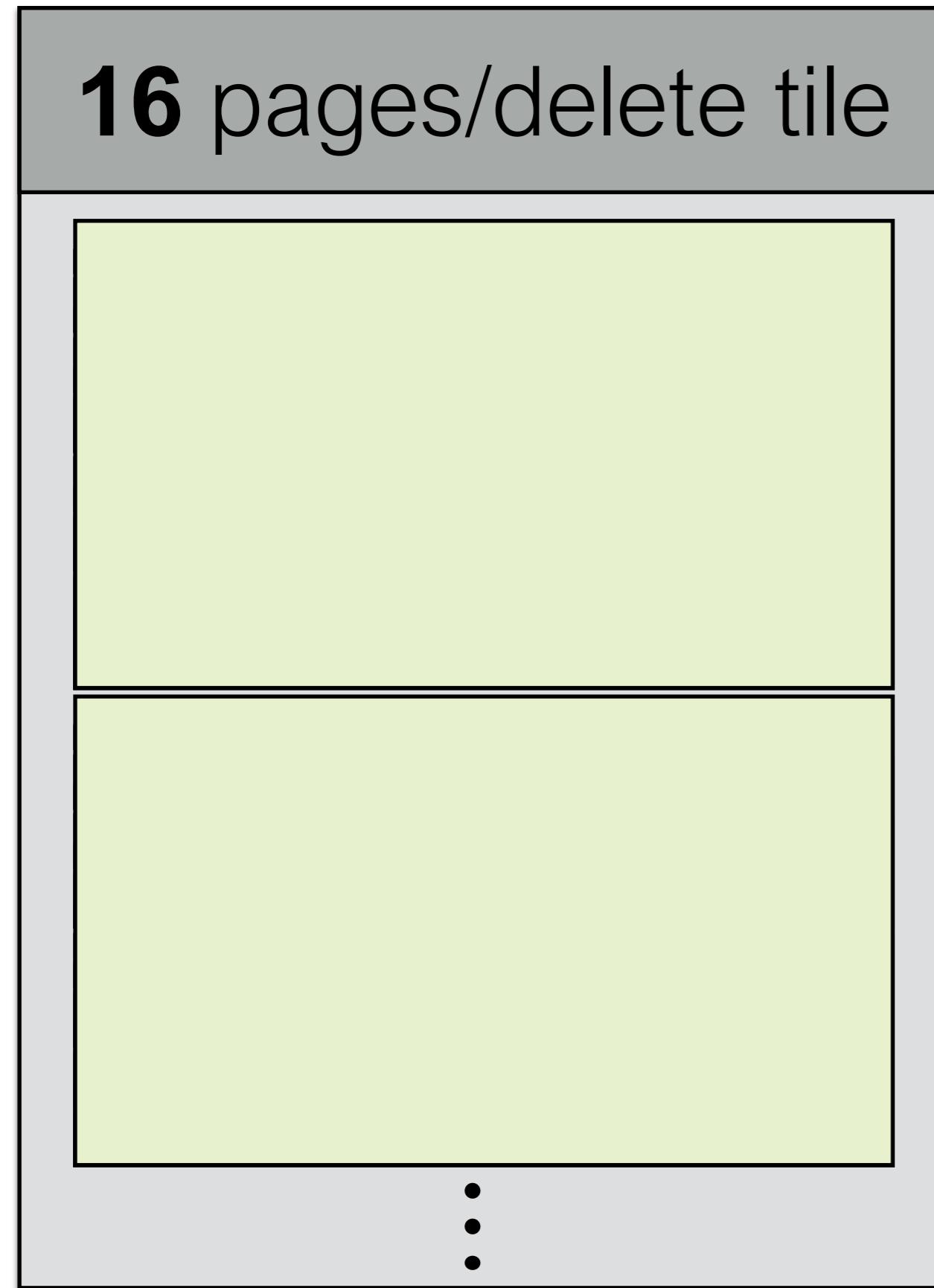
Key Weaving storage layout



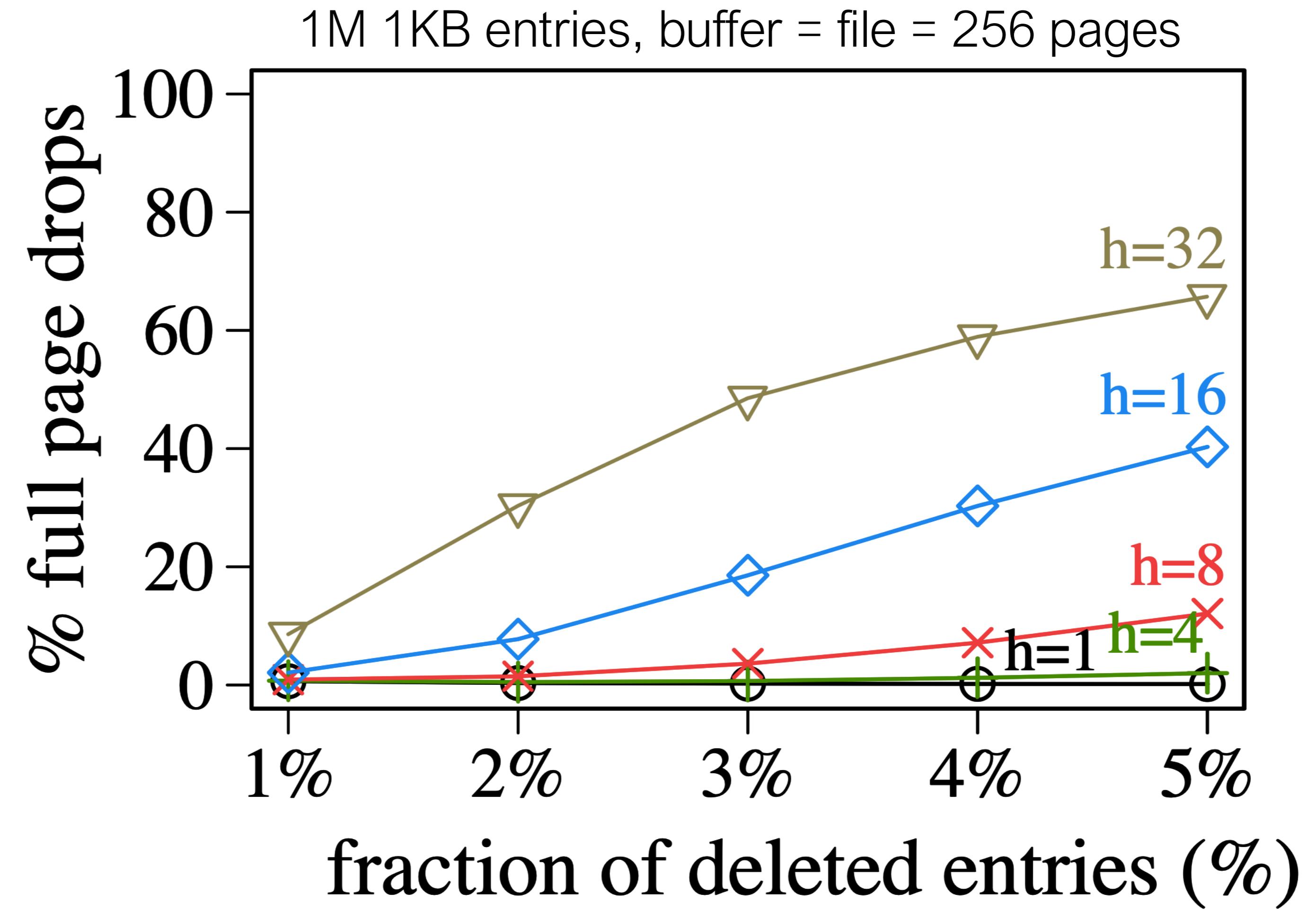
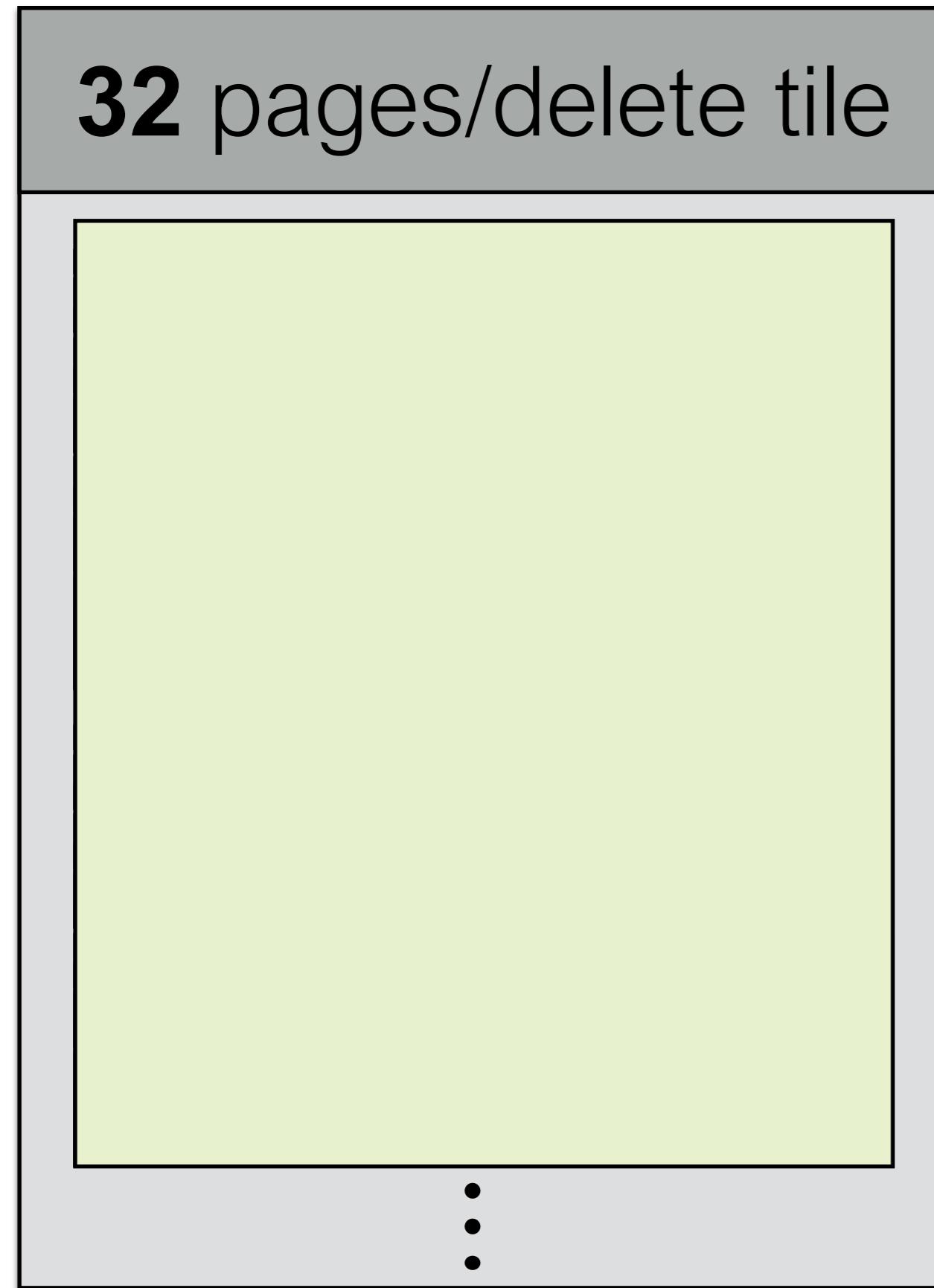
Key Weaving storage layout



Key Weaving storage layout

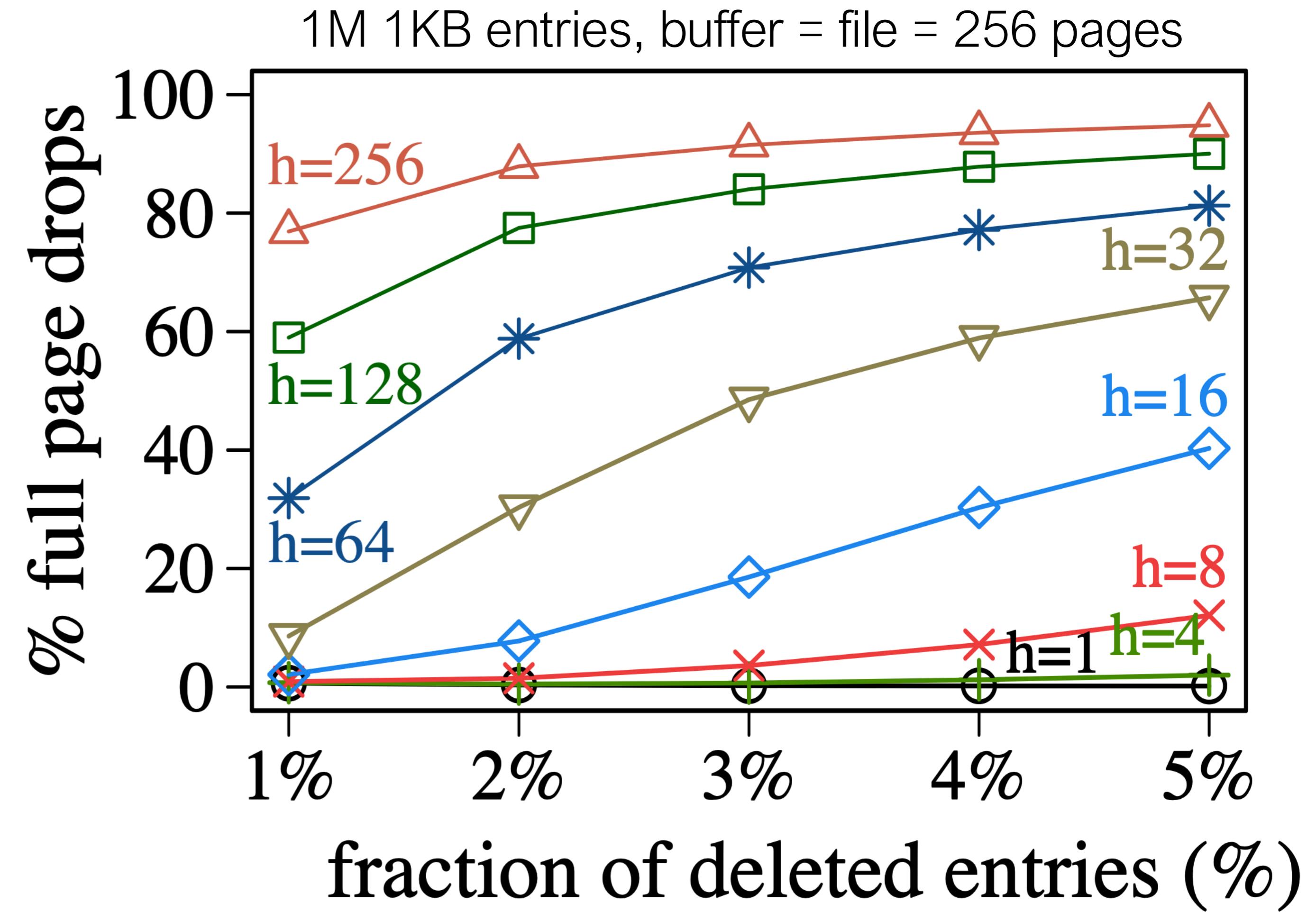


Key Weaving storage layout



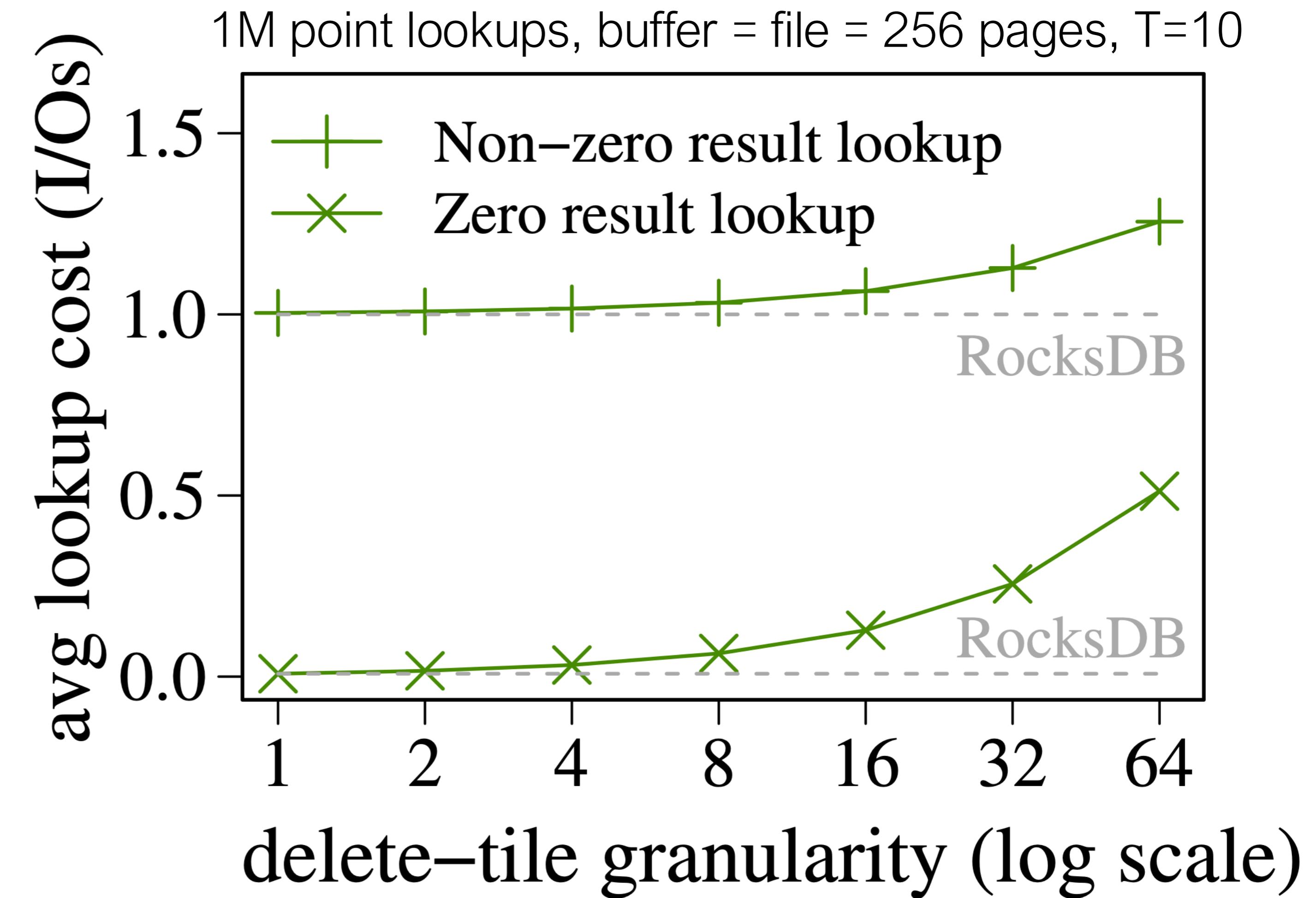
Key Weaving storage layout

- reduced latency spikes ✓
- full page drops reduces superfluous I/Os ✓

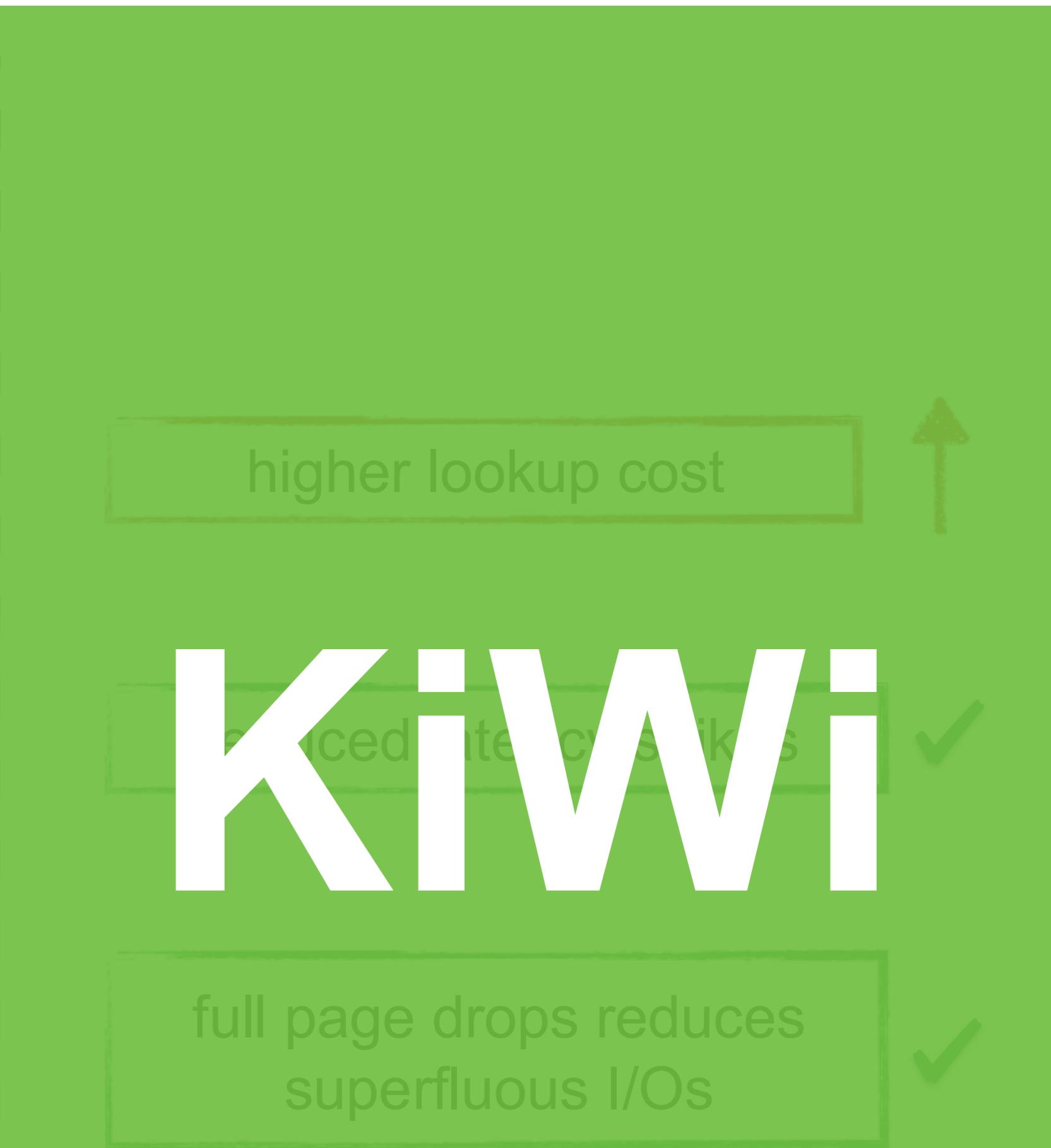
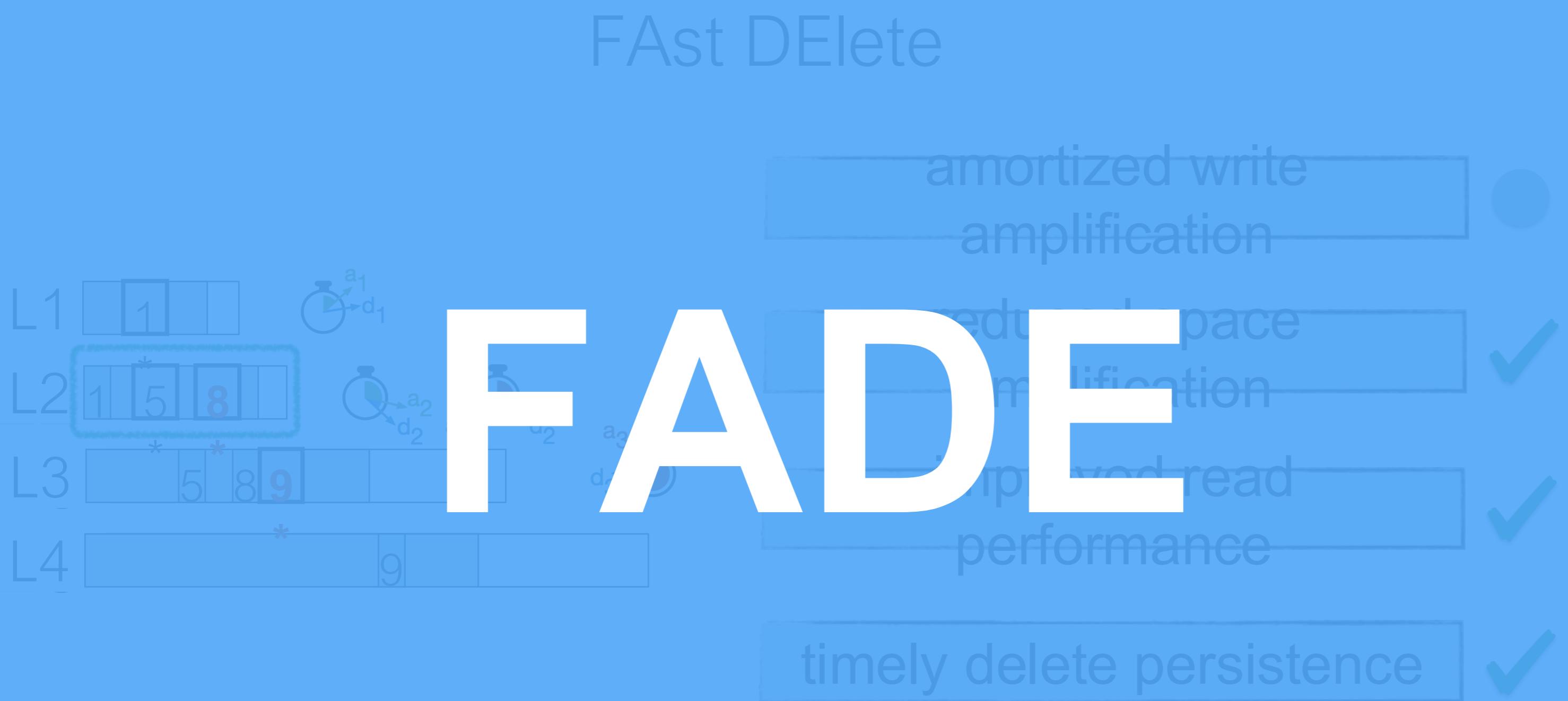


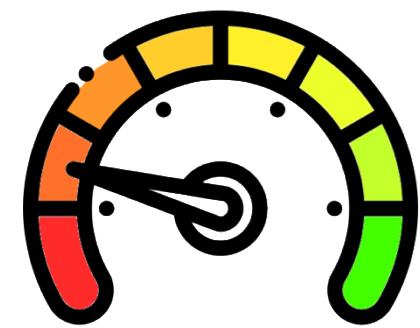
Key Weaving storage layout

- higher lookup cost ↑
- reduced latency spikes ✓
- full page drops reduces superfluous I/Os ✓

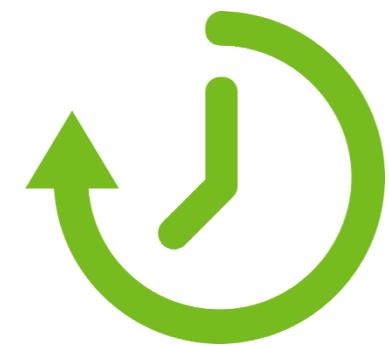


the solution





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 Architectures

class 6b

Deletes on LSM Trees

Prof. Manos Athanassoulis

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