BOSTON
UNIVERSITY

CS561: Data Systems Architectures

class 8

Efficient Deletes in Log-Structured Key-Value Storage

Prof. Manos Athanassoulis

BOSTON UNIVERSITY

CS561: Data Systems Architectures

class 8

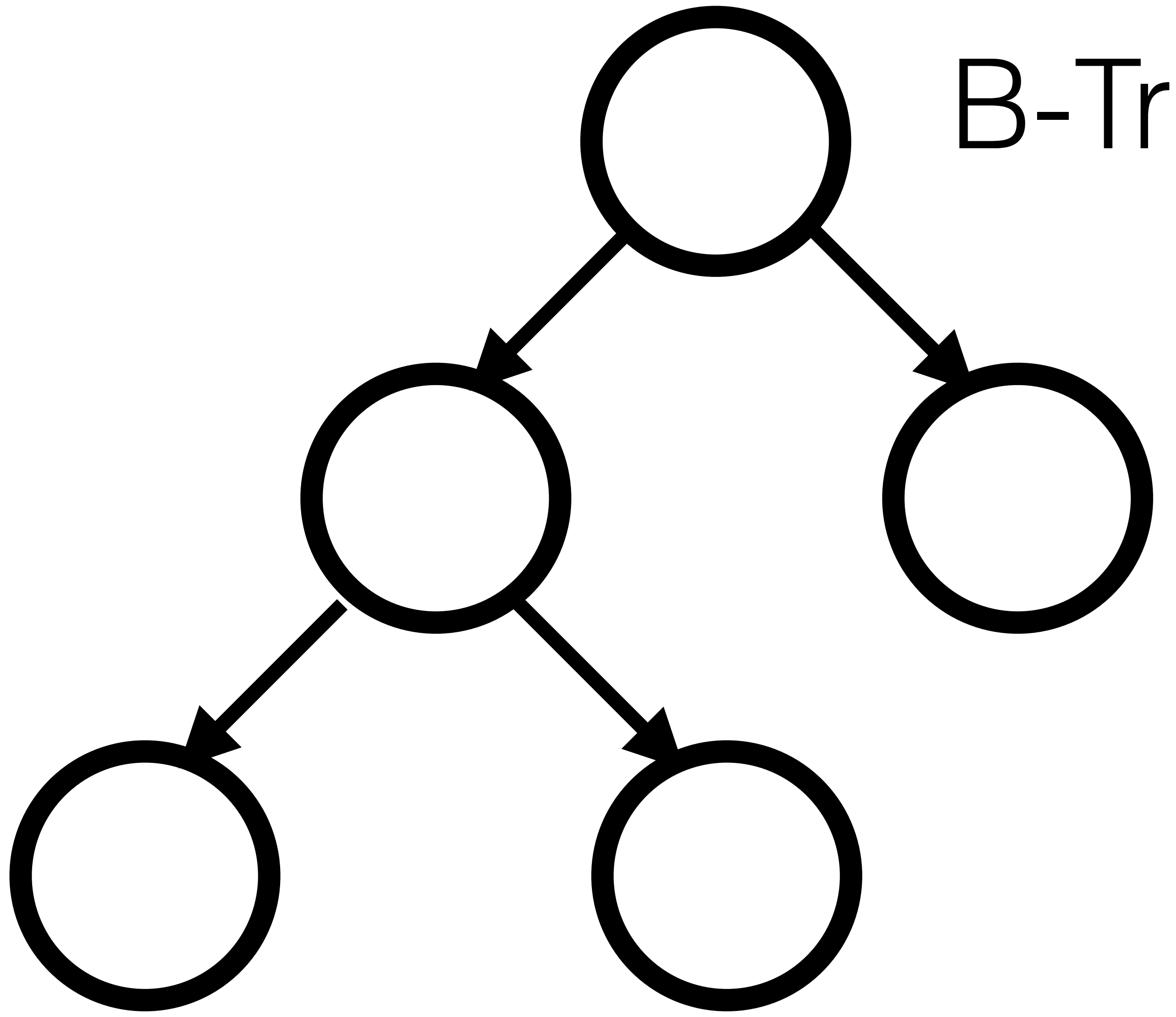
Delete: *the forgotten operator*

Prof. Manos Athanassoulis

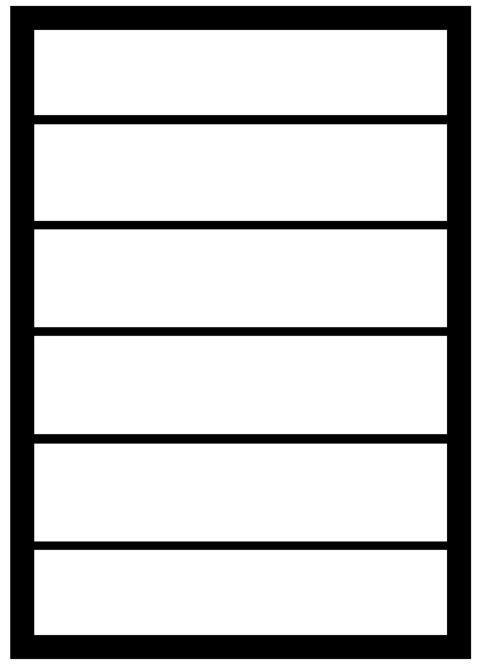
```
<your_favorite_data_structure>::delete (key)
{
  //todo
}
```

in-place

out-of-place



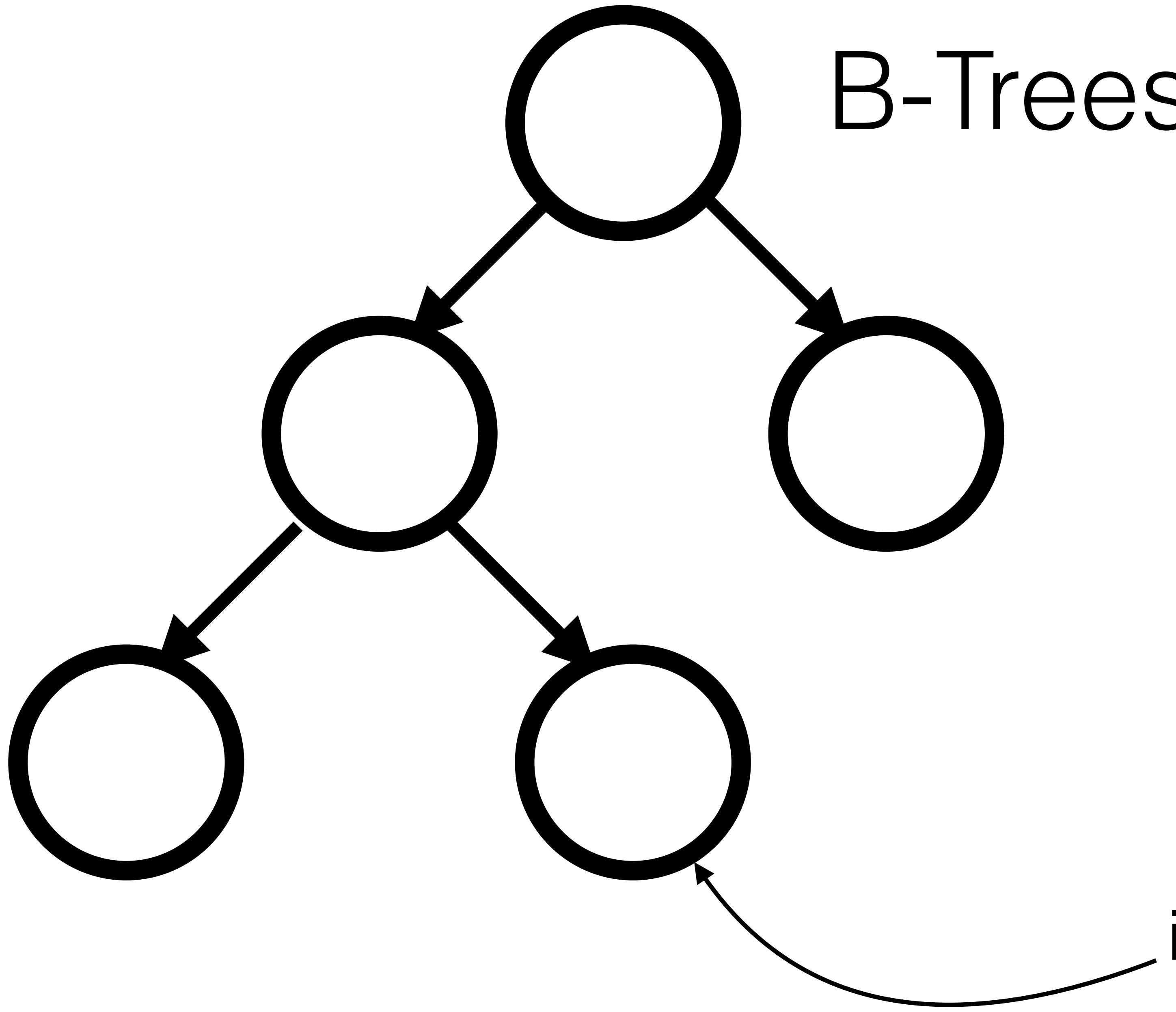
B-Trees



Heap Files
(slotted pages)

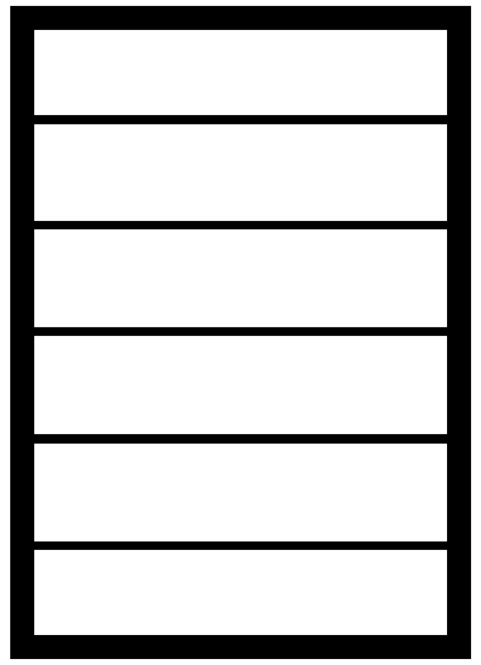
in-place

out-of-place



B-Trees

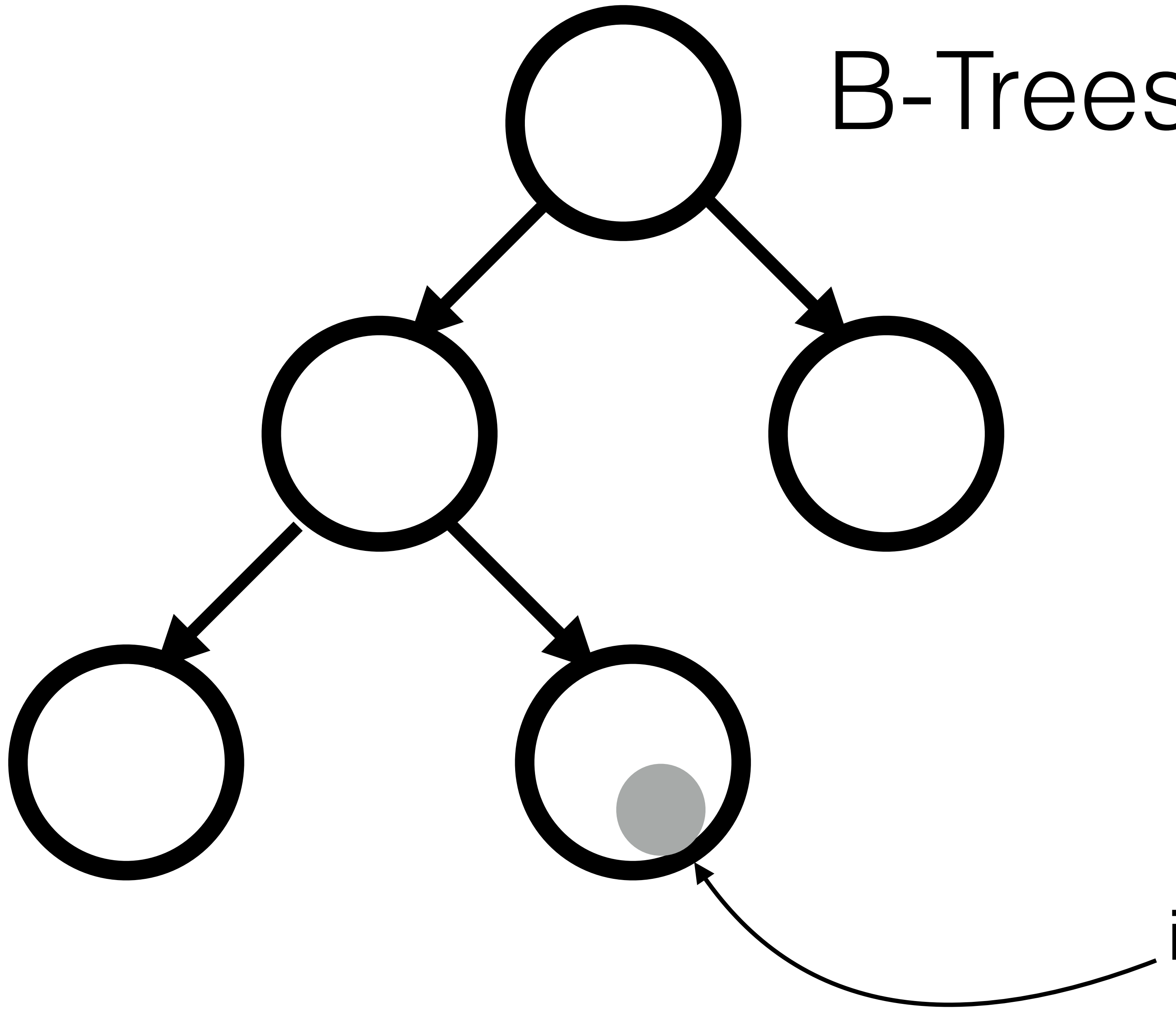
invalidate the entry



Heap Files
(slotted pages)

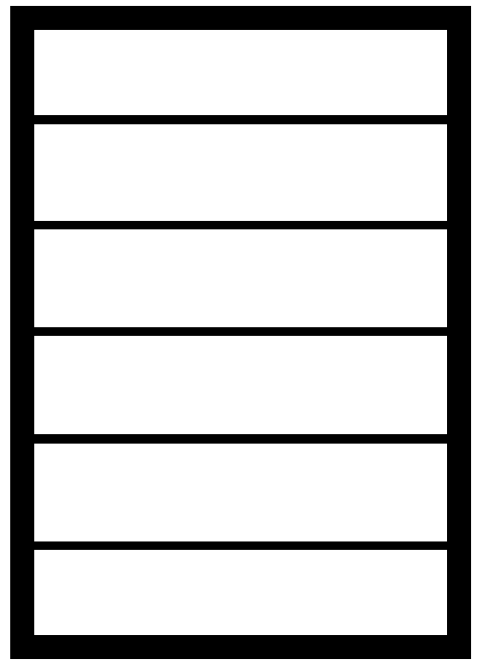
in-place

out-of-place



B-Trees

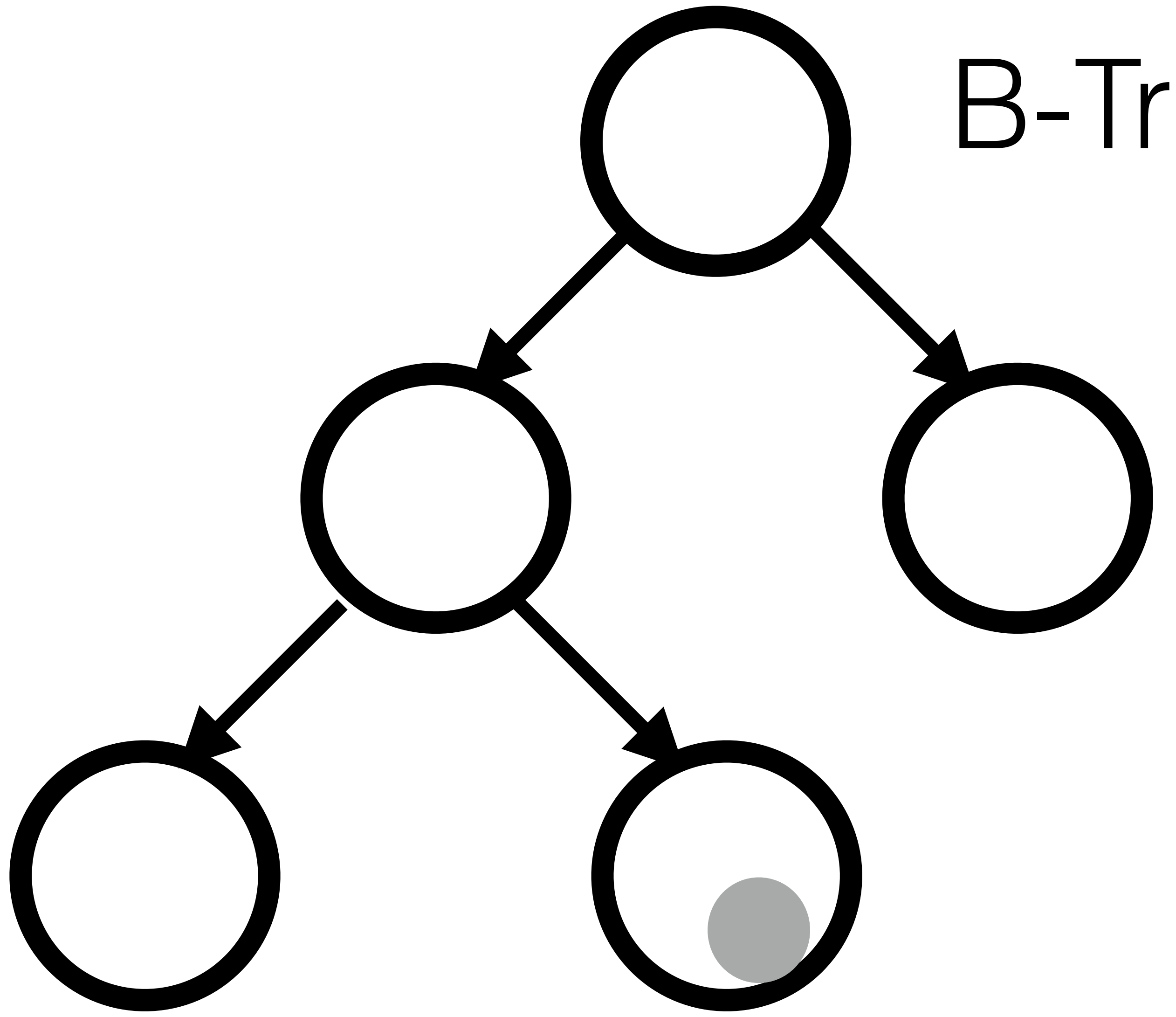
invalidate the entry



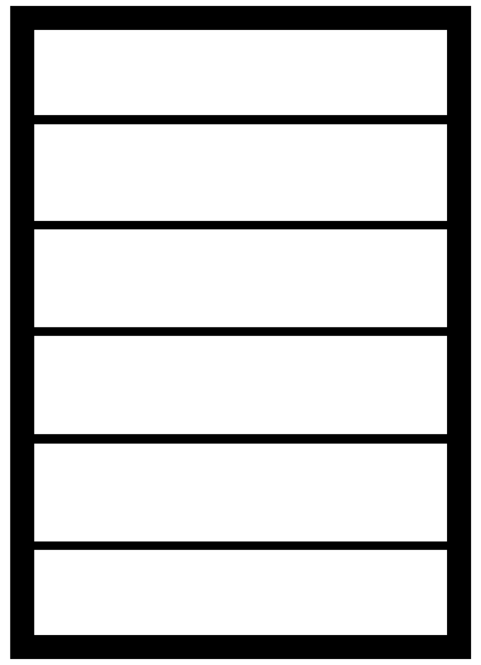
Heap Files
(slotted pages)

in-place

out-of-place



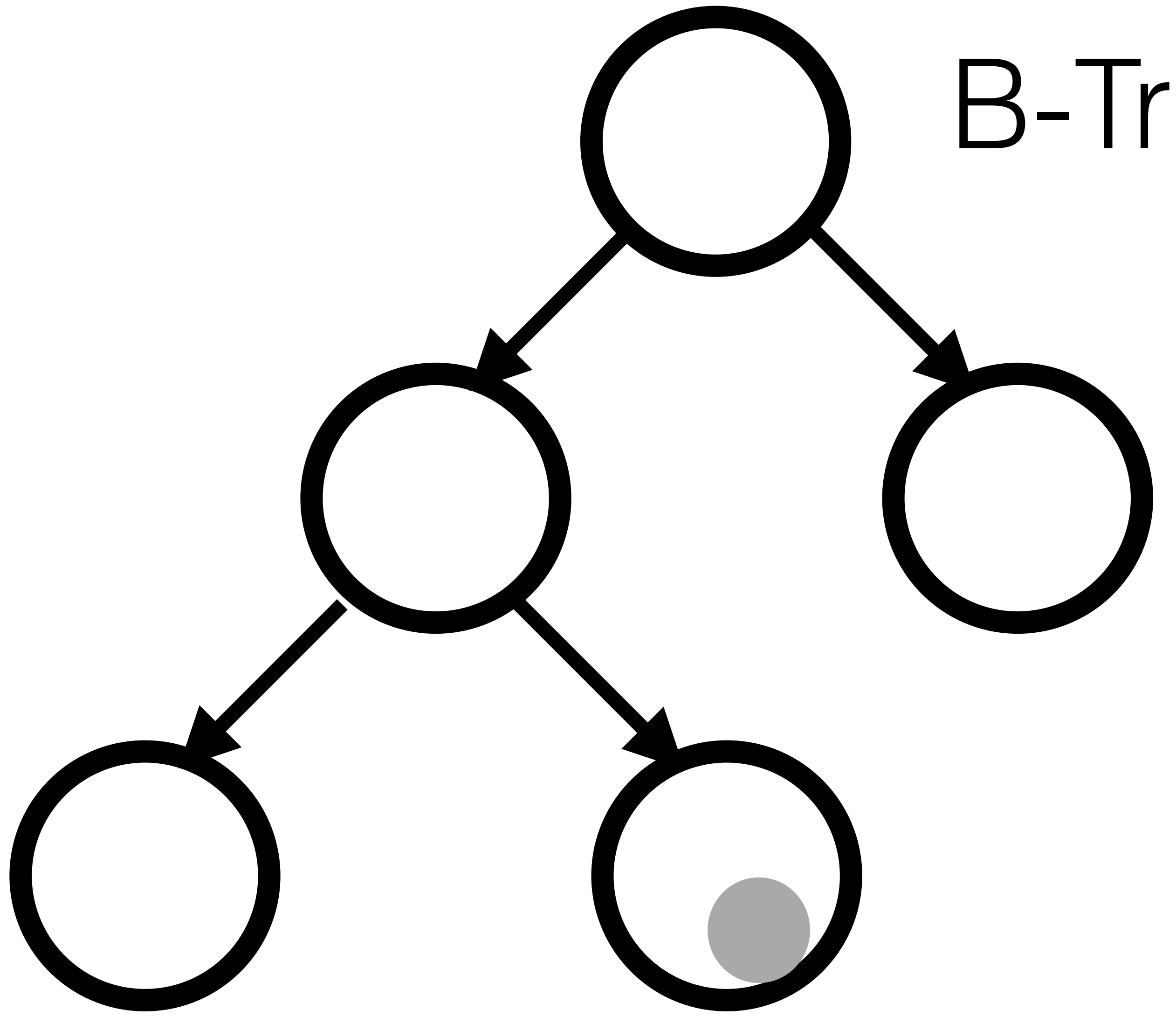
B-Trees



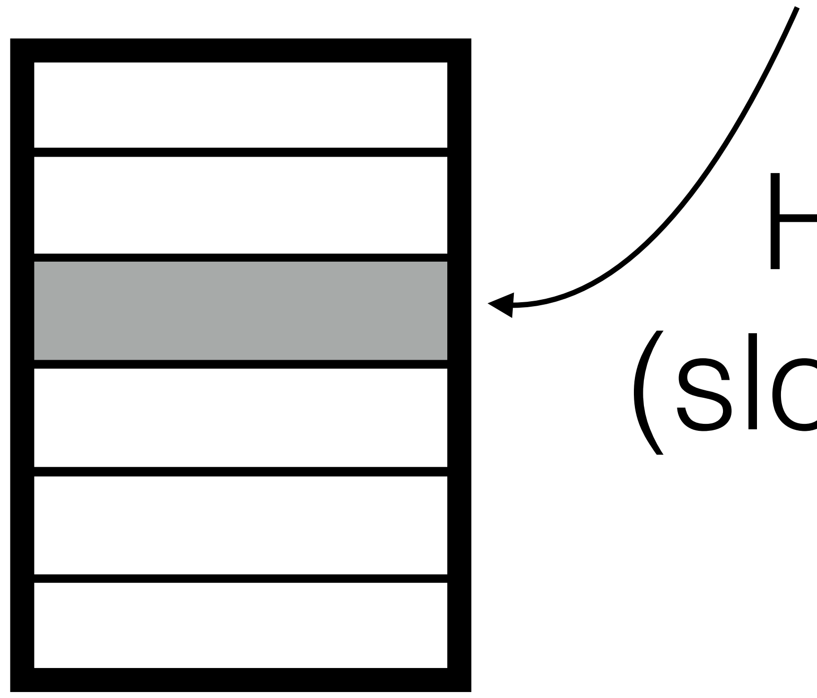
invalidate the entry

Heap Files
(slotted pages)

in-place



B-Trees



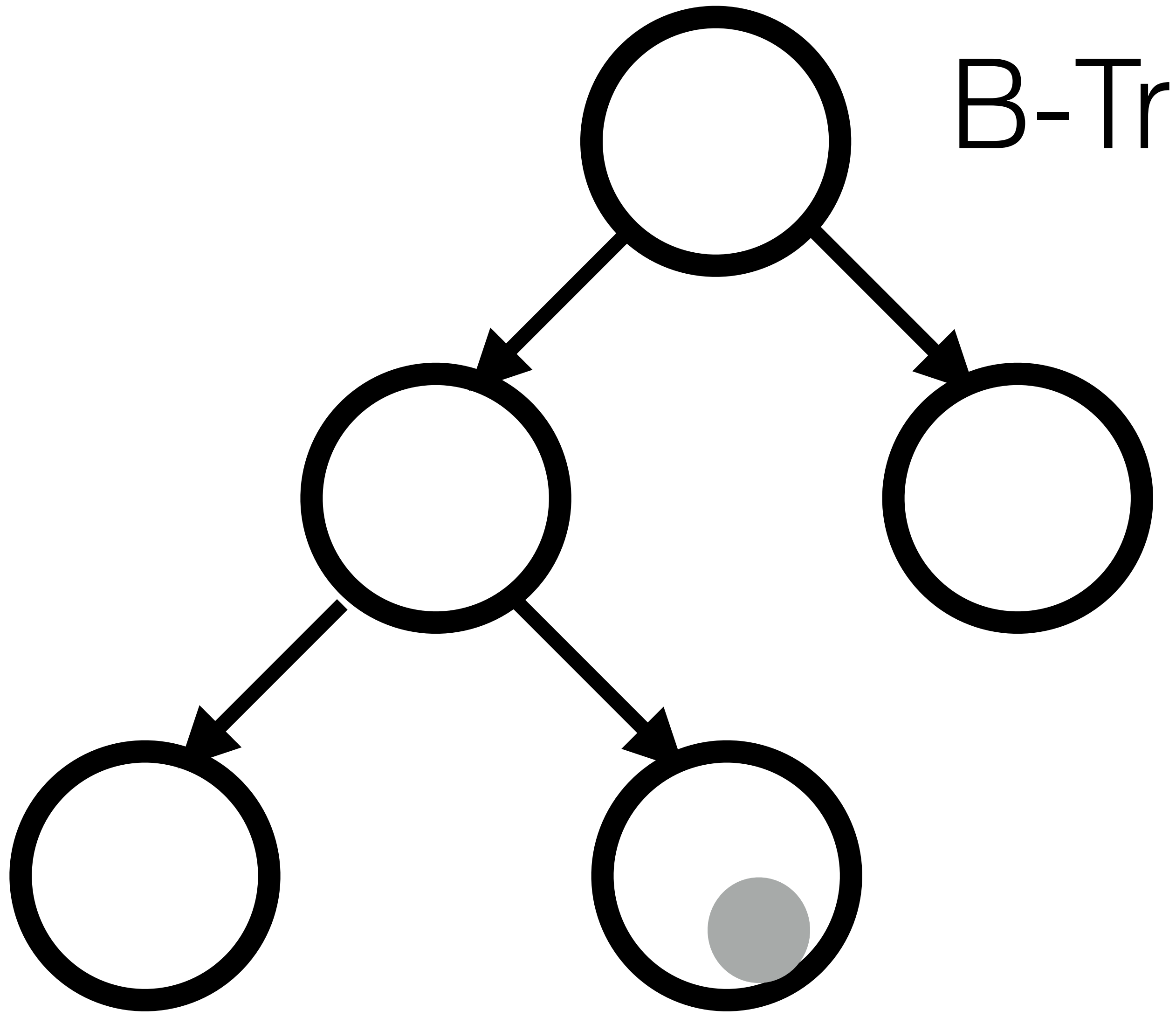
invalidate the entry

Heap Files
(slotted pages)

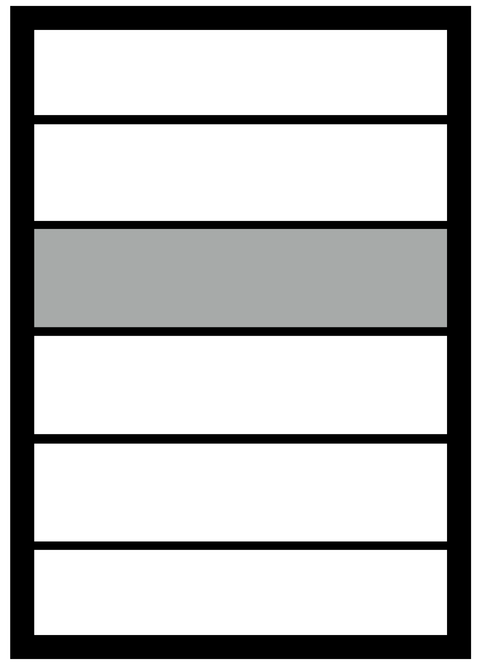
out-of-place

in-place

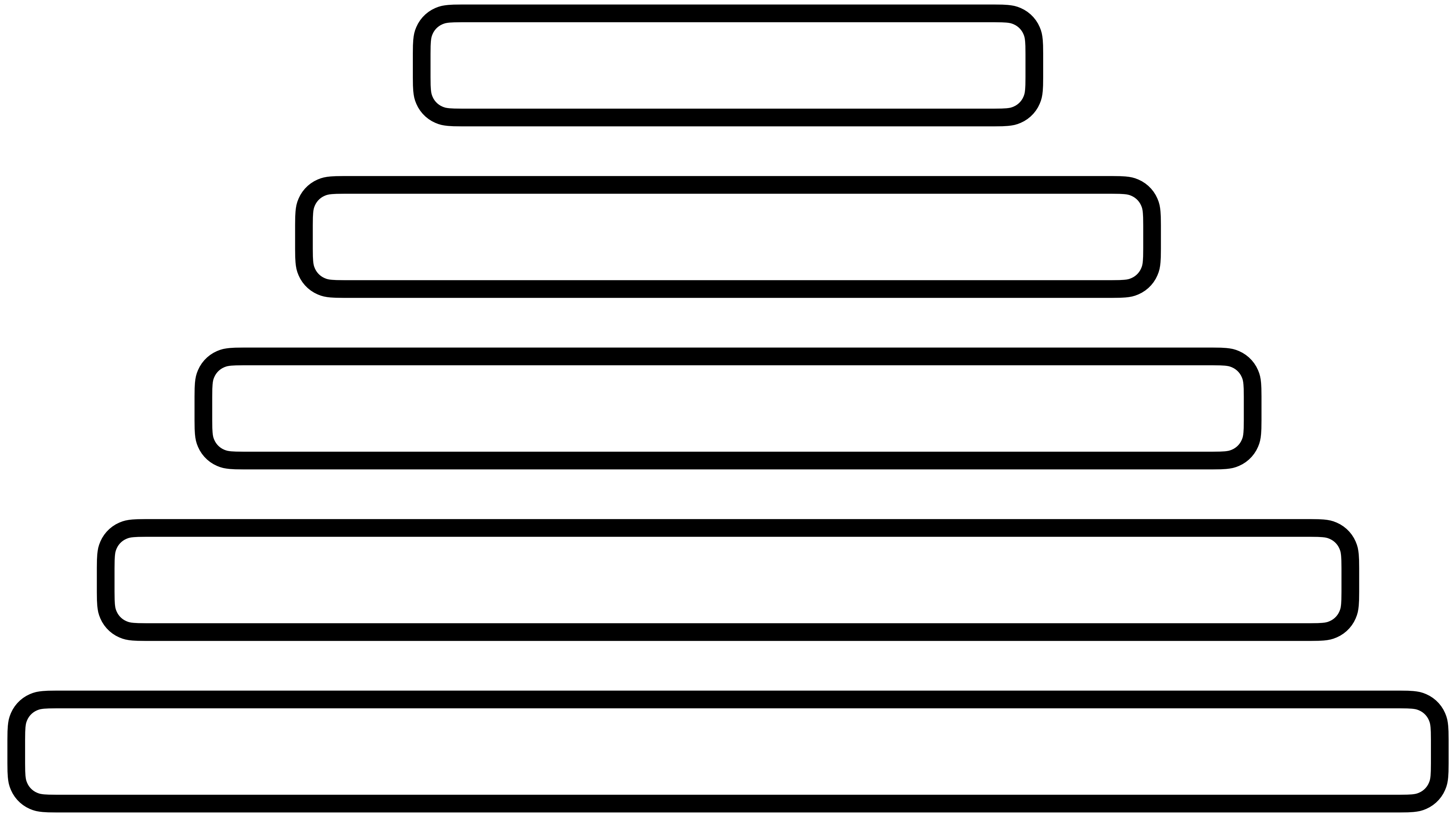
out-of-place



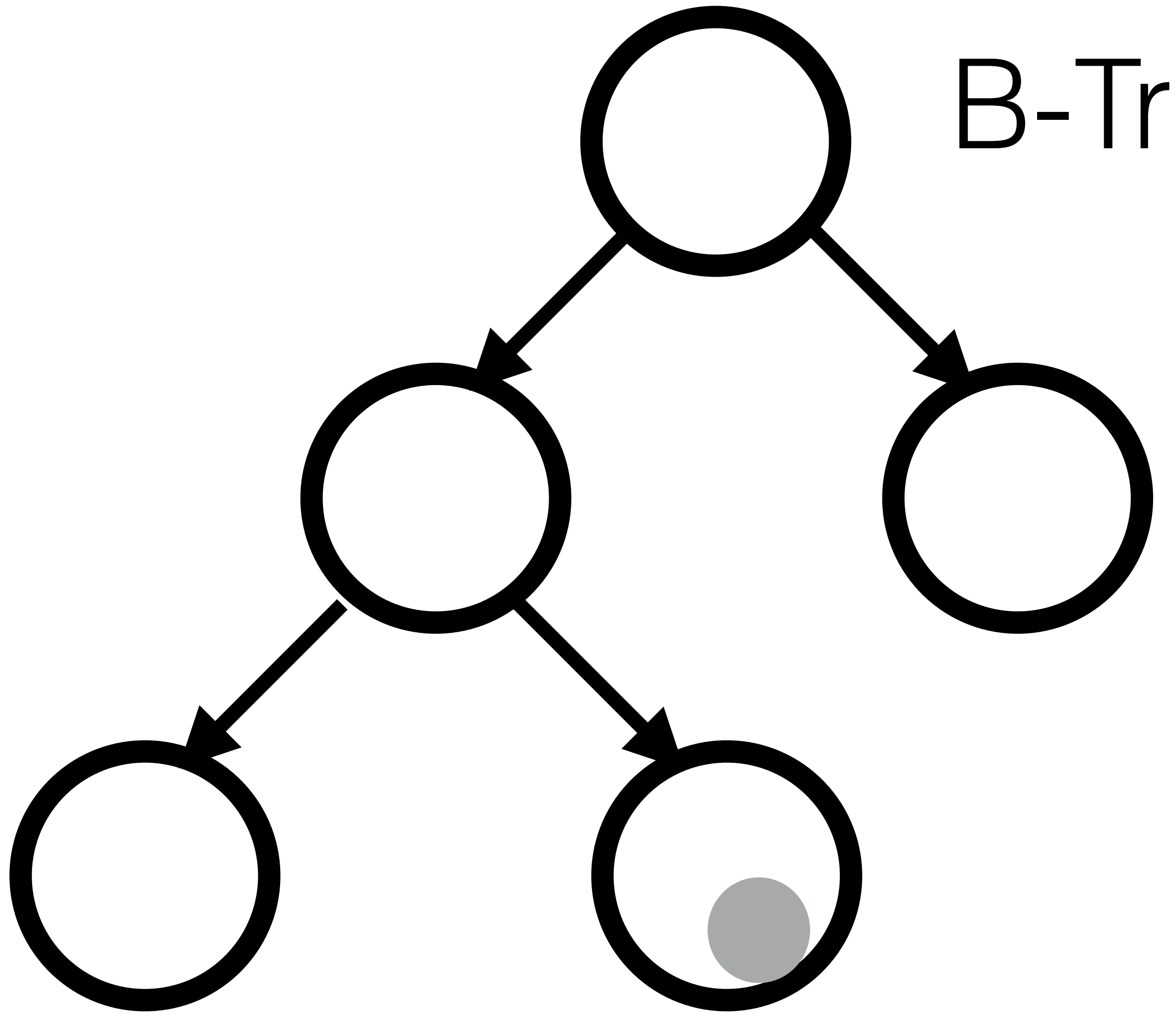
B-Trees



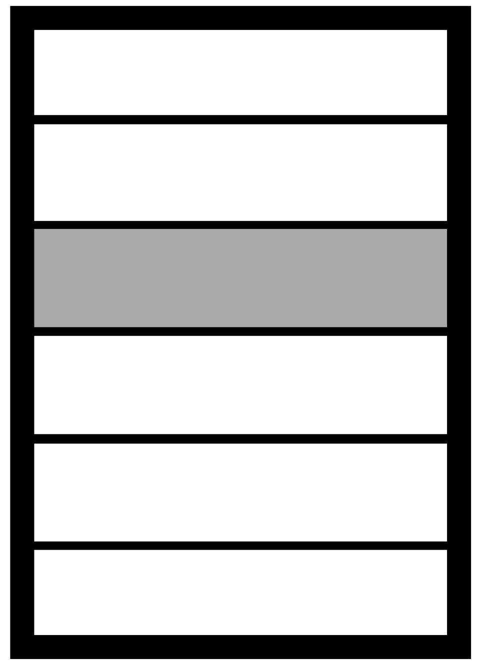
Heap Files
(slotted pages)



in-place

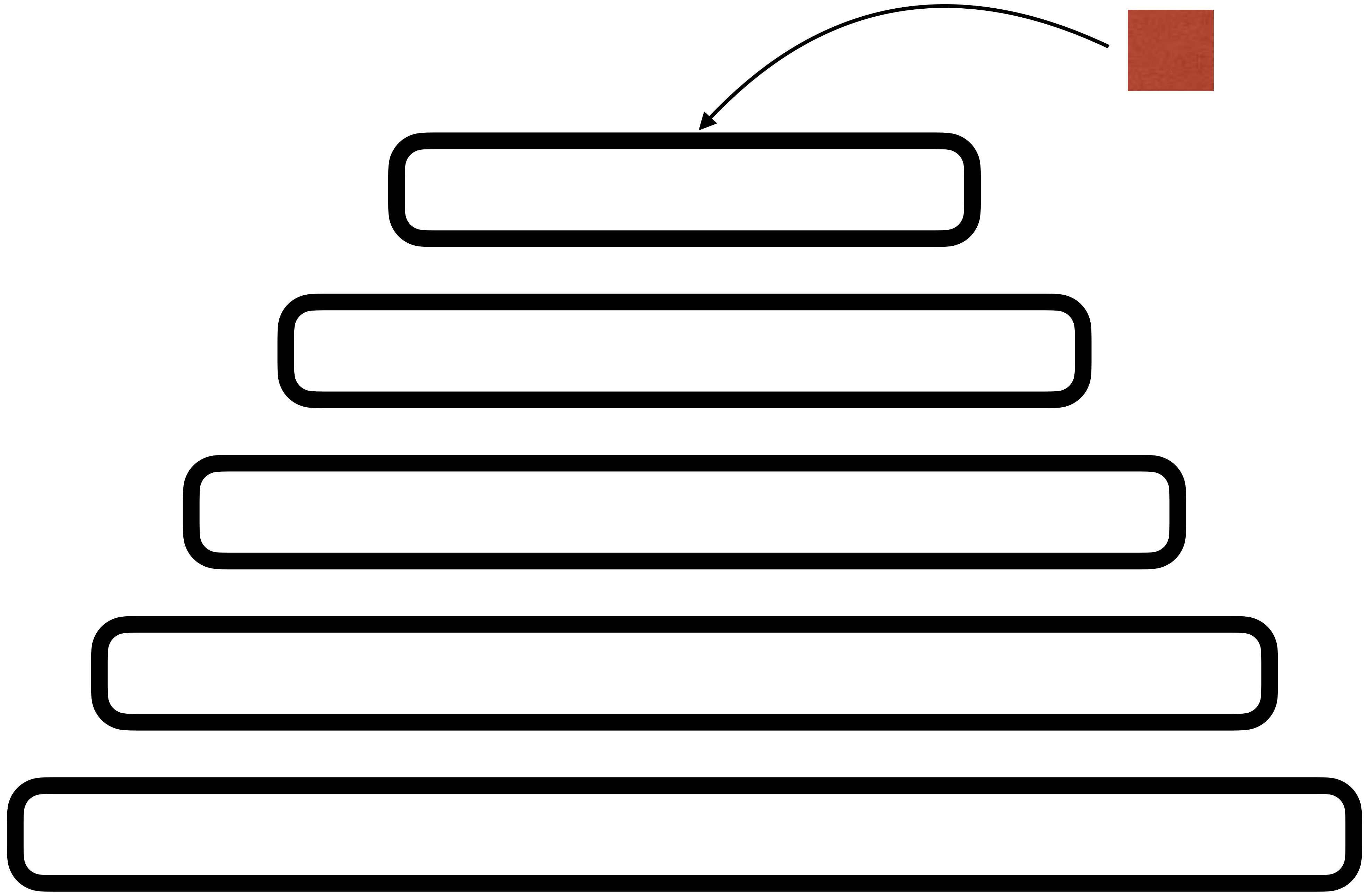


B-Trees

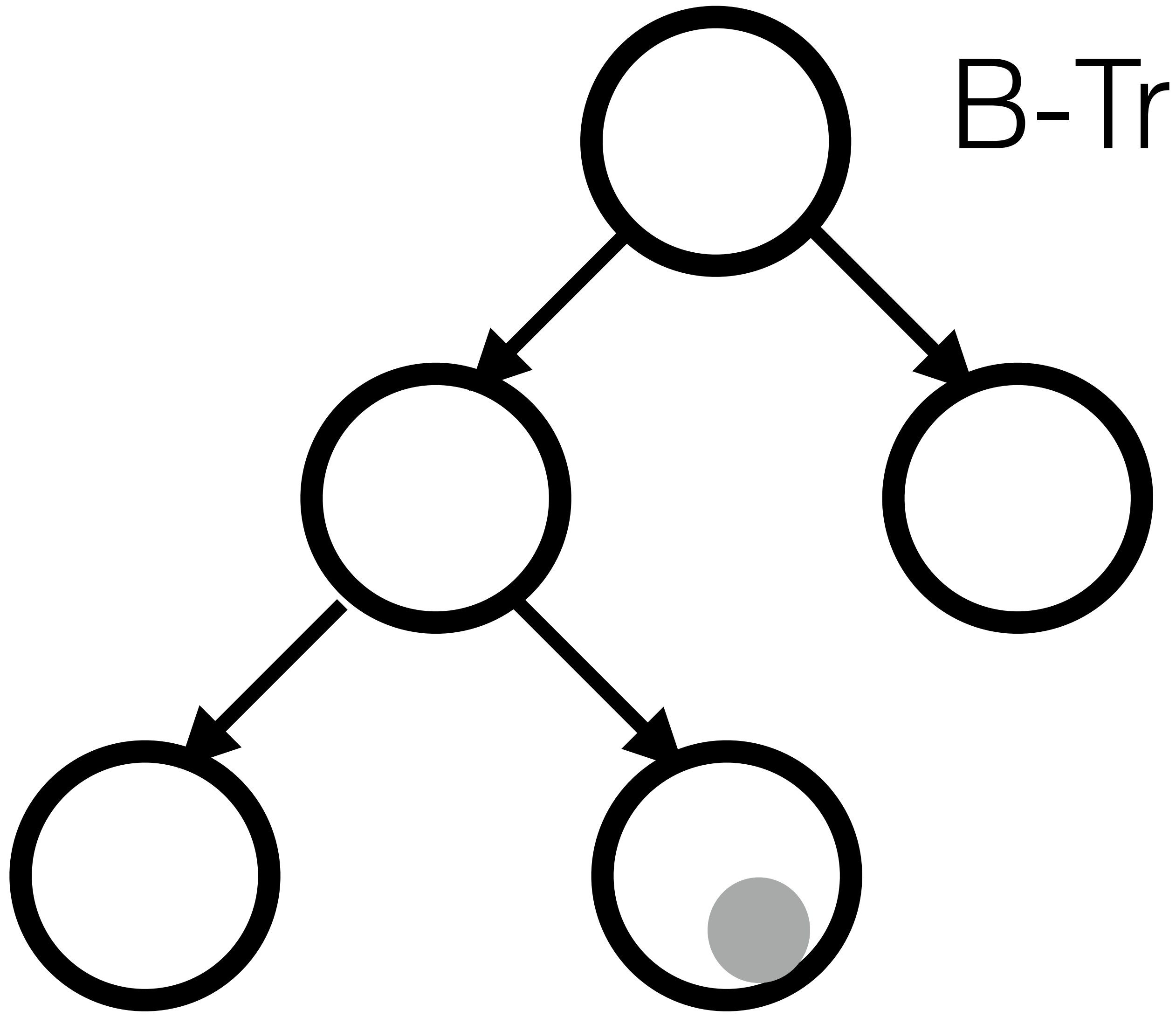


Heap Files
(slotted pages)

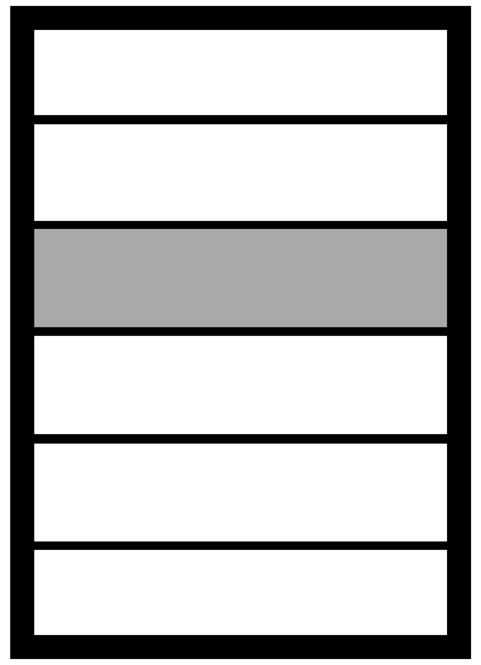
out-of-place



in-place

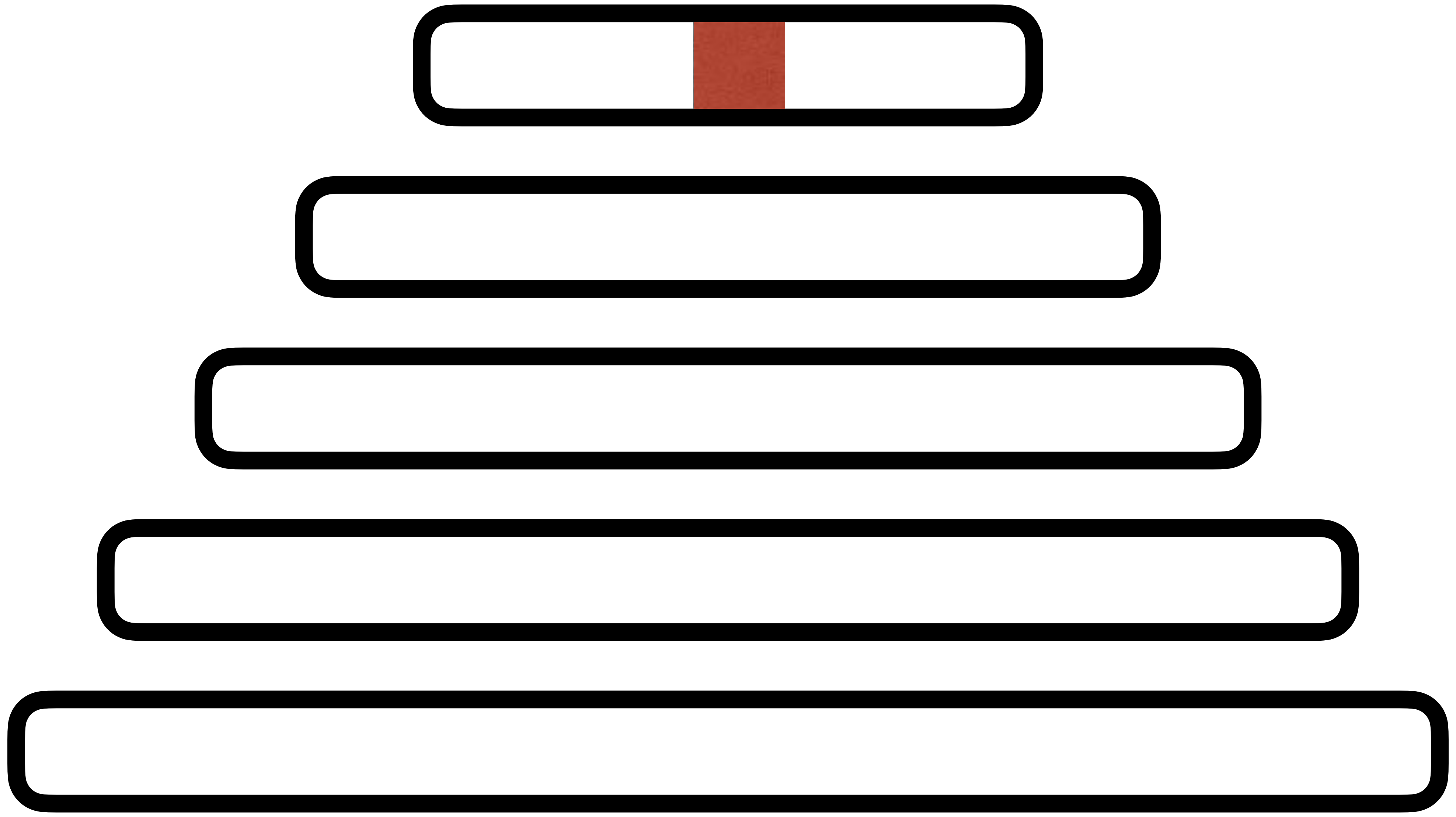


B-Trees

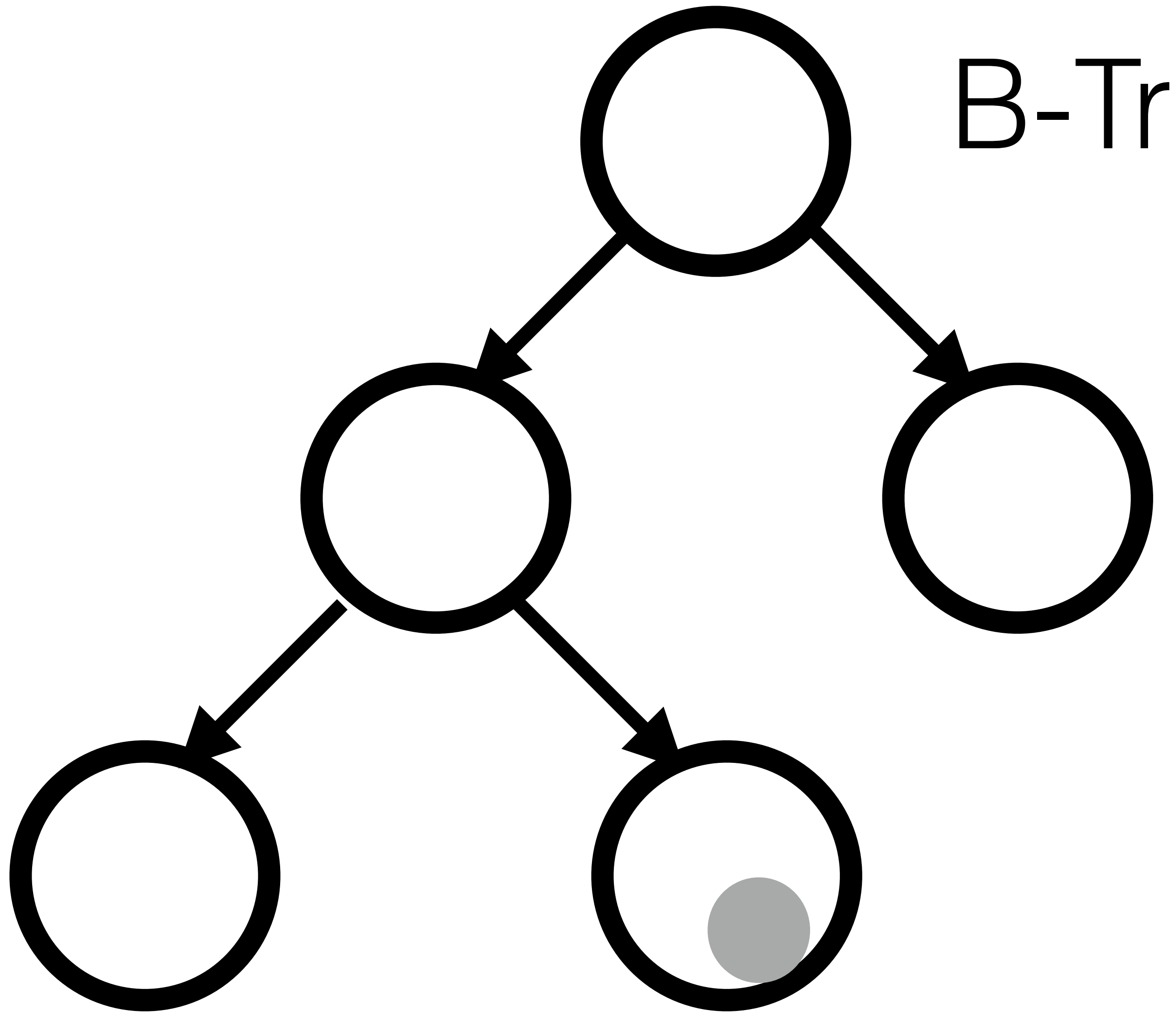


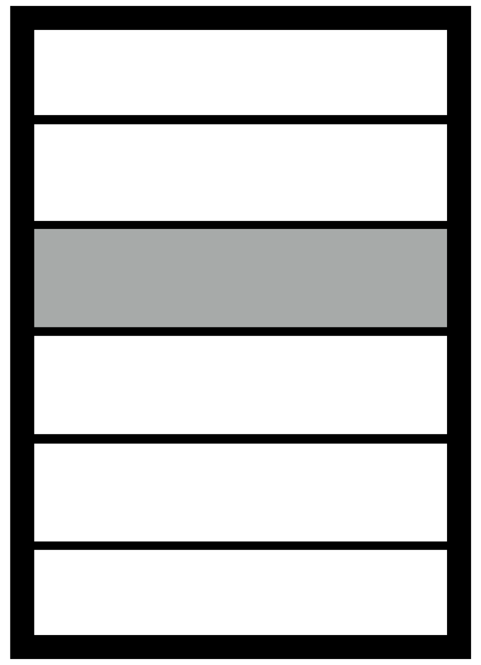
Heap Files
(slotted pages)

out-of-place

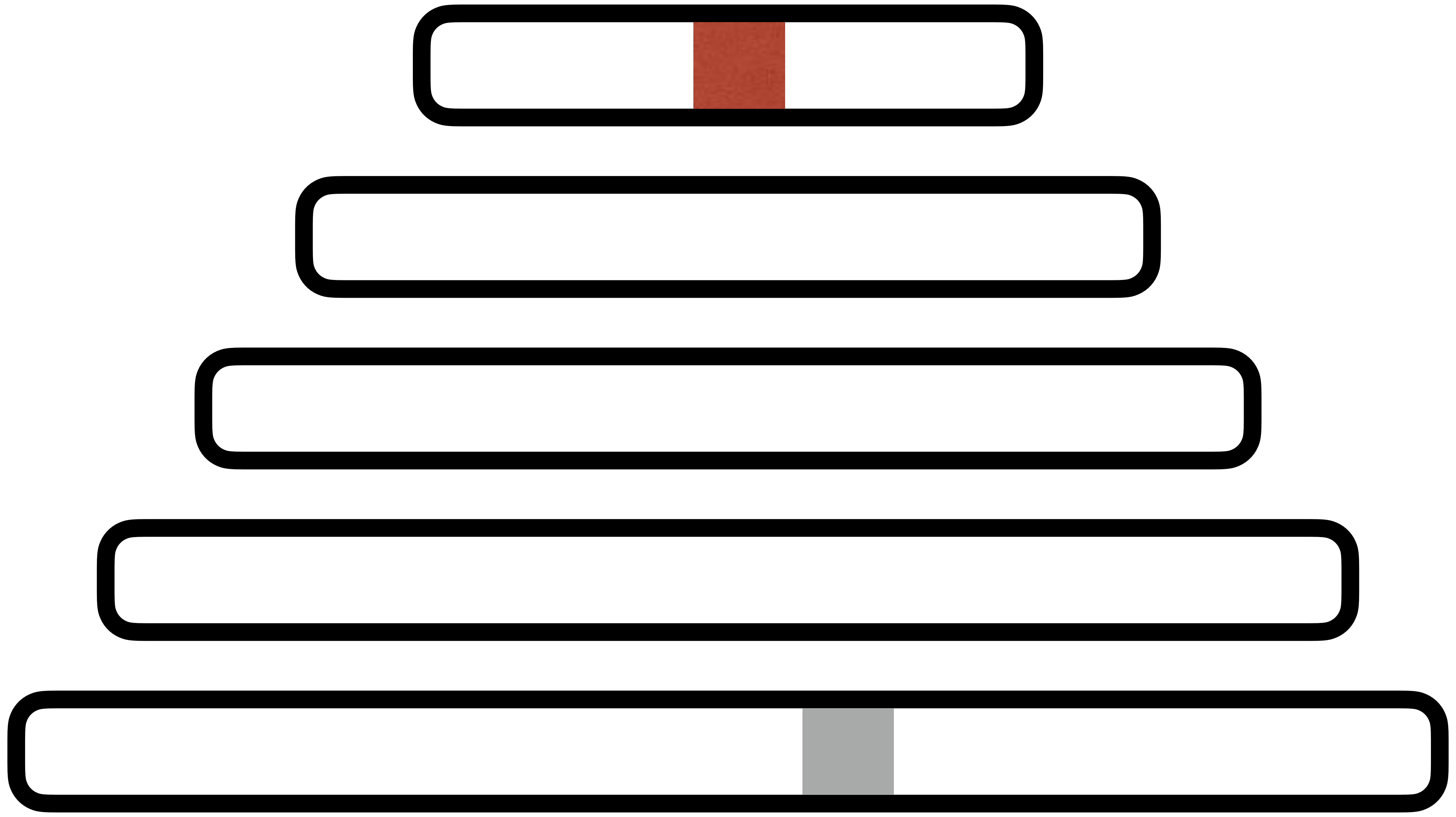


in-place


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

other tradeoffs?

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

↓ ↑

Today's talk:

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

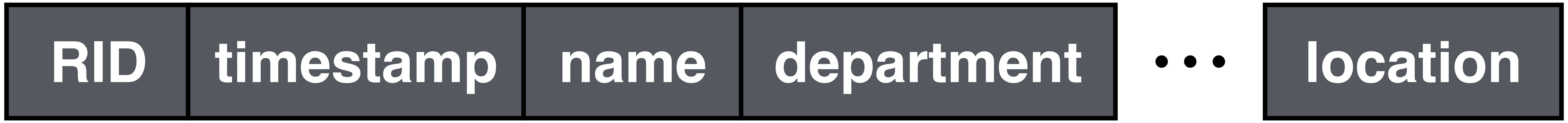
Presented at SIGMOD 2020

Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, Manos Athanassoulis

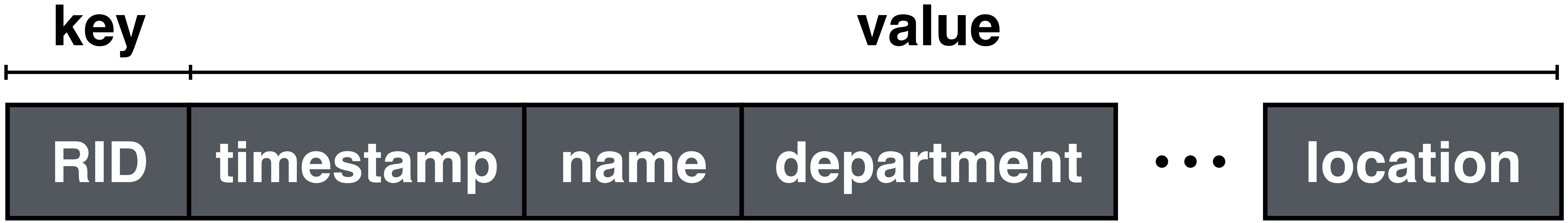
BOSTON UNIVERSITY

DiSC lab

key-value pairs



RID | timestamp | name | department | • • • | location



key

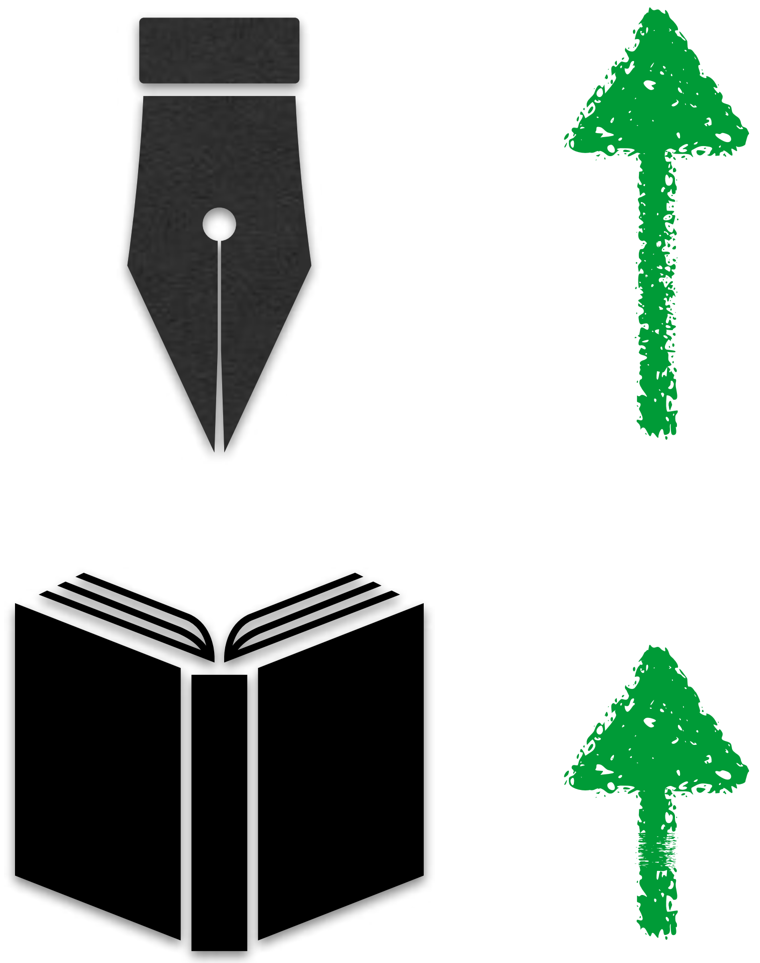
value

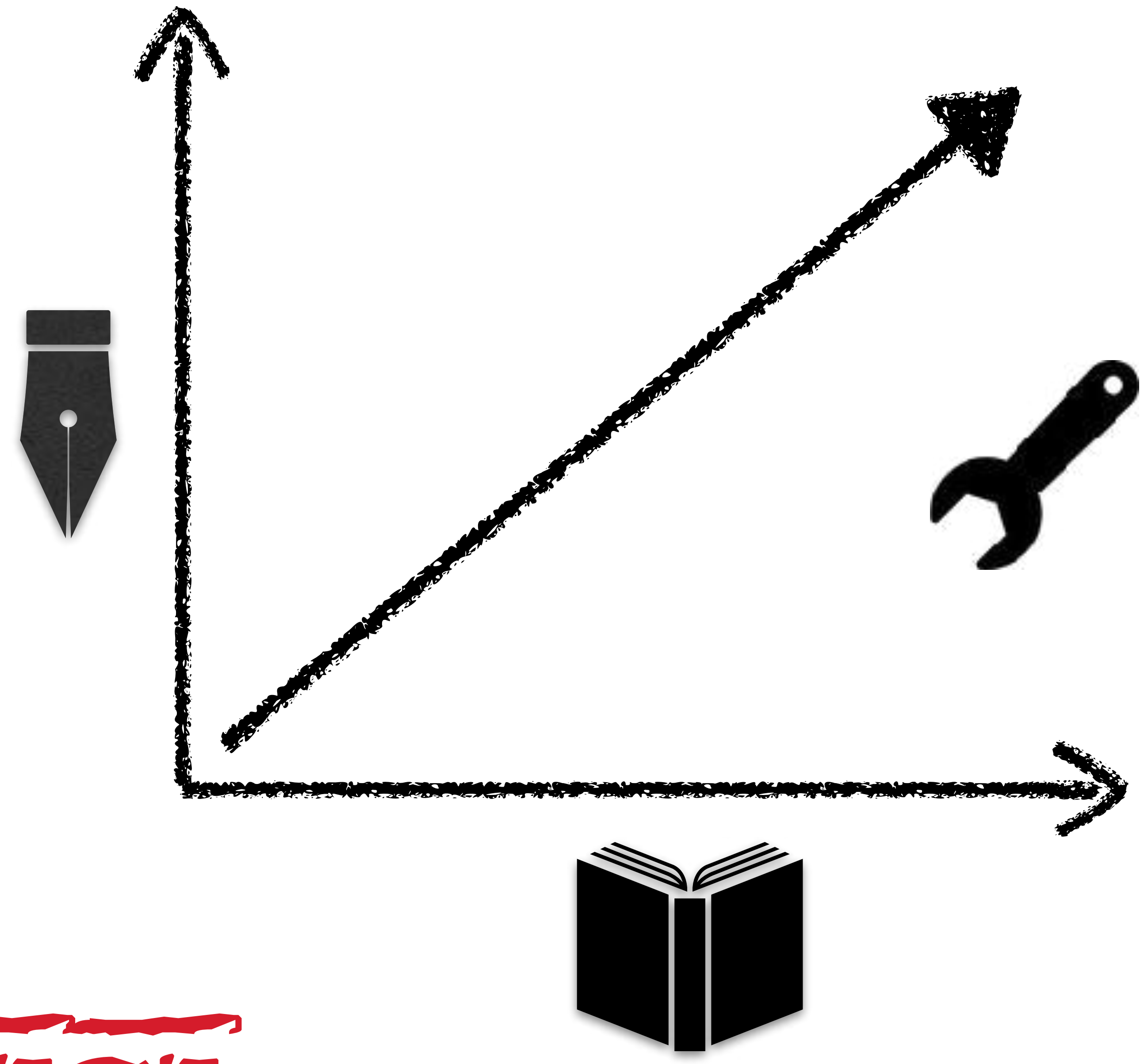
RID | timestamp | name | department | • • • | location

✒ ↑

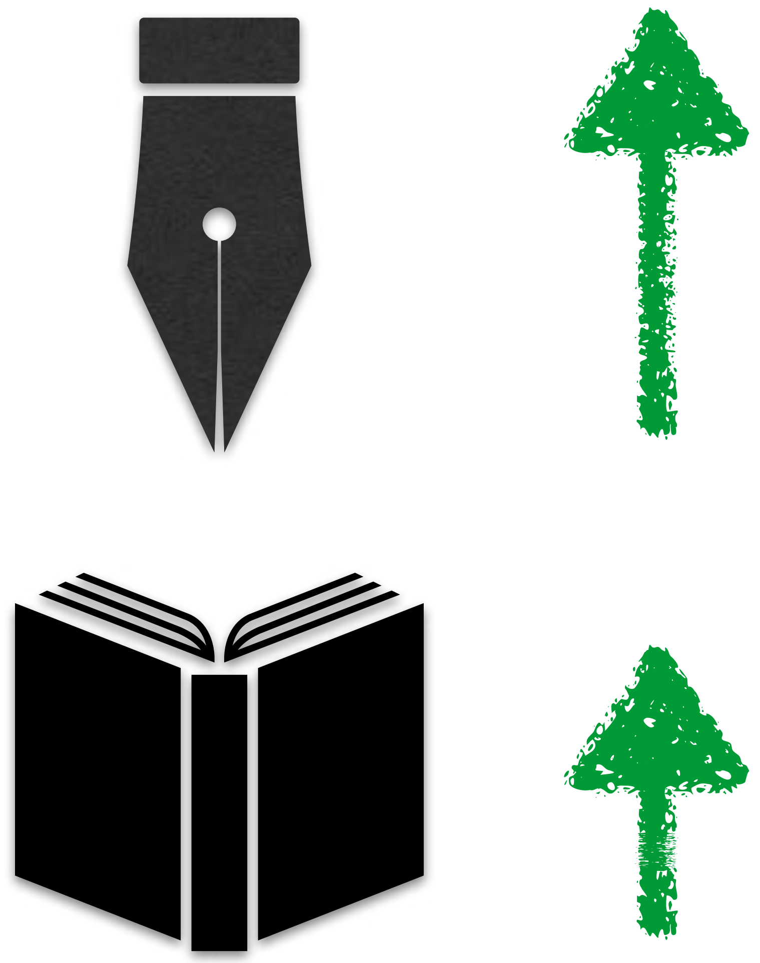
📖 ↑

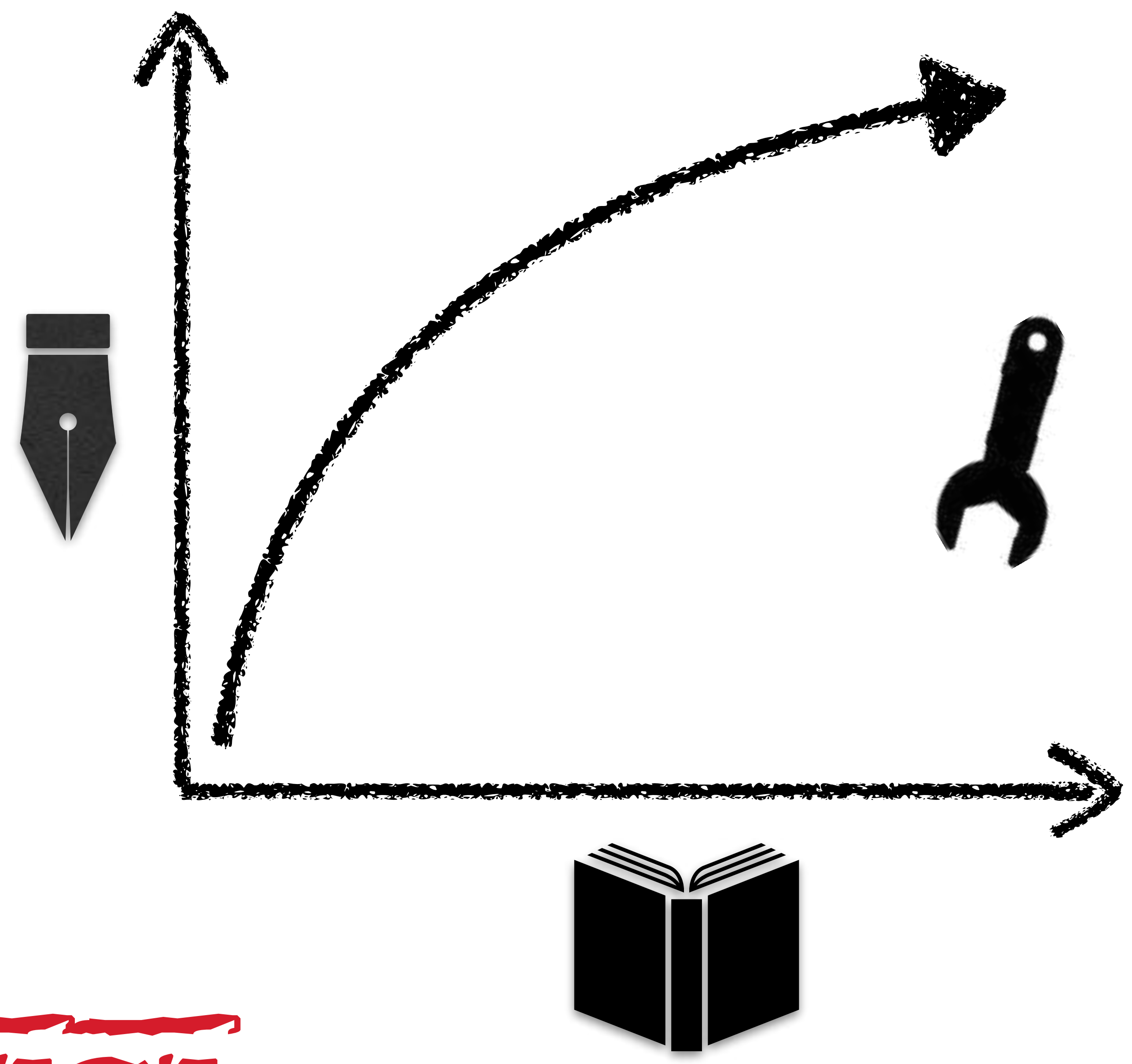
LSM-TREE



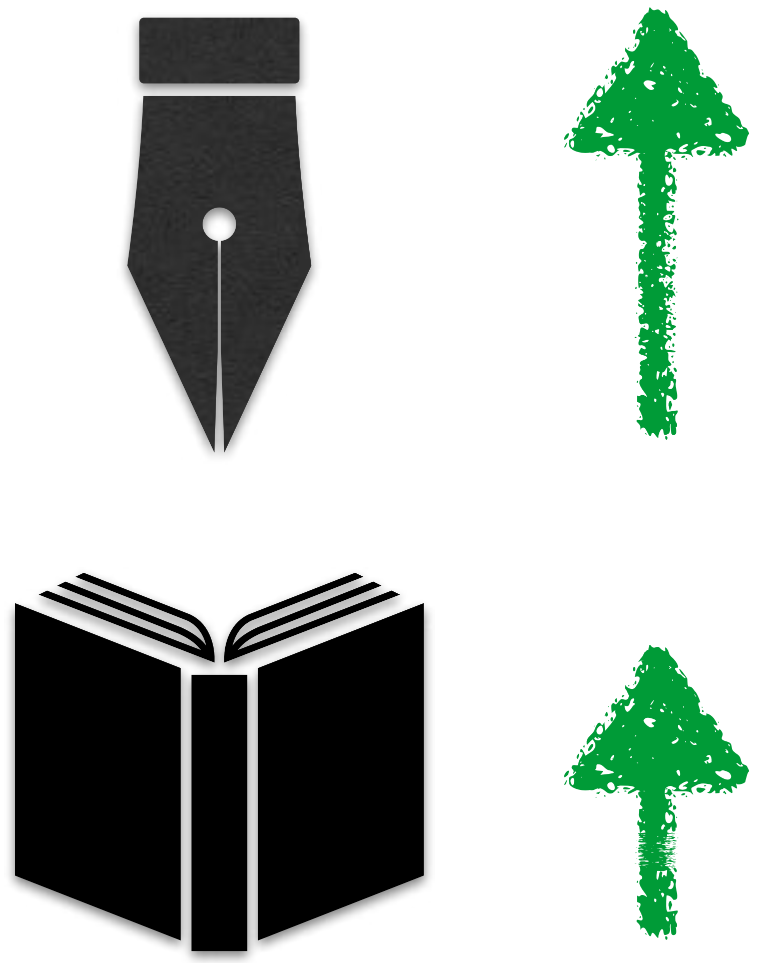


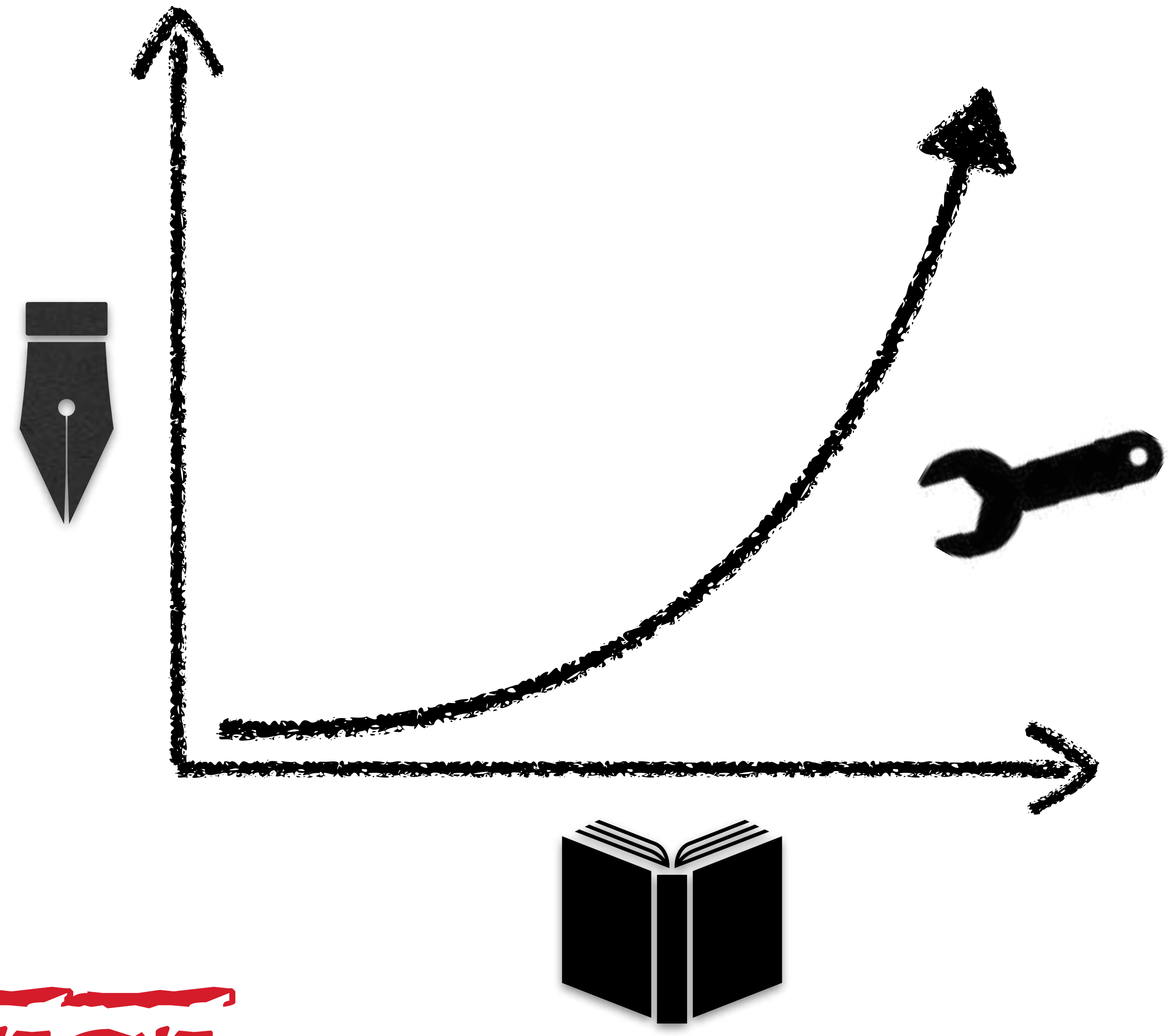
LSM-TREE





LSM-TREE





LSM-TREE

cassandra

levelDB

RocksDB

WT

accumulo™

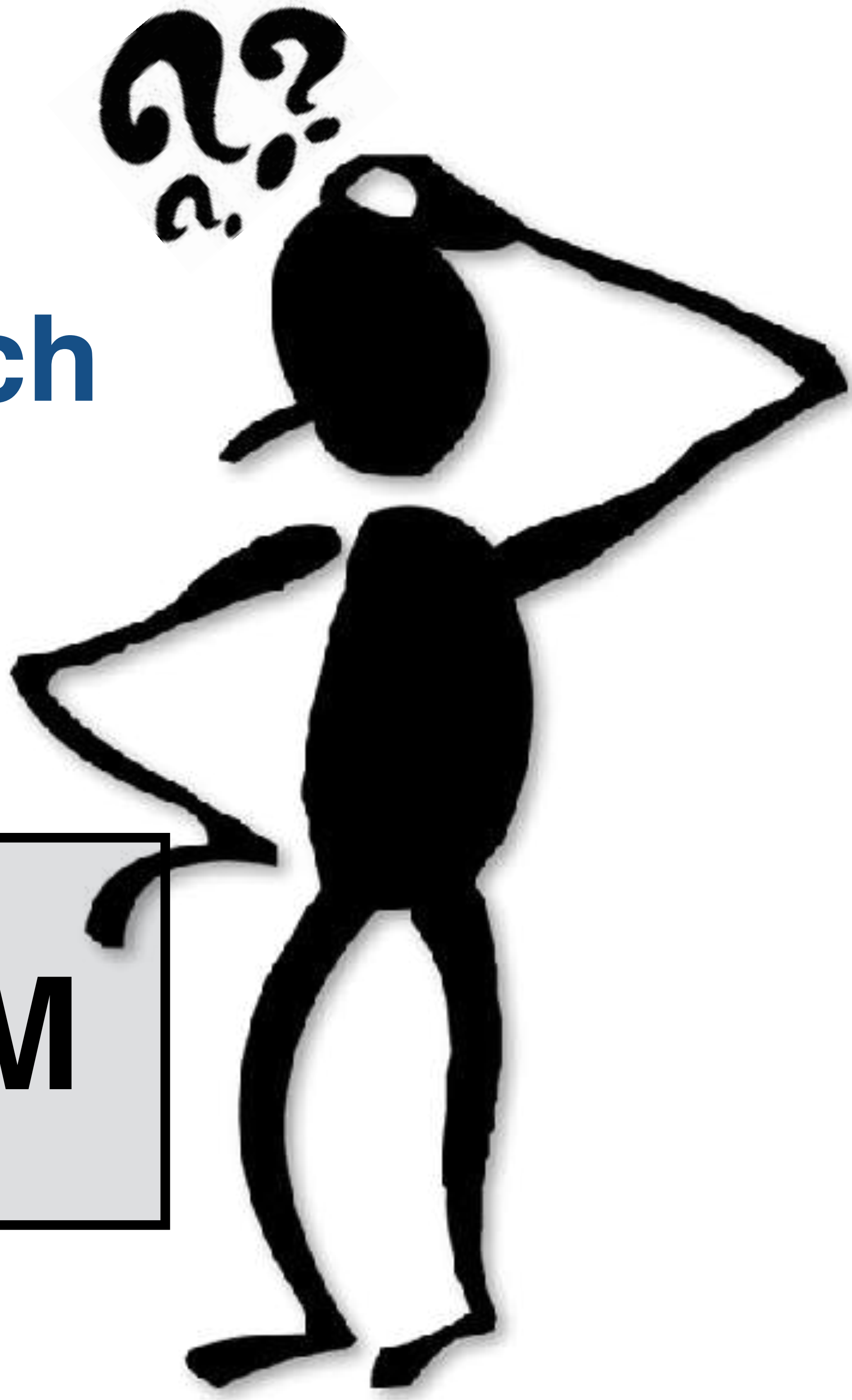
DynamoDB

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."

Large-scale production	Internal db ops	Privacy
ZippyDB **25.2M/day**	table drop	CCPA (California)
UP2X **92.5M merge through deletes**	data migration	GDPR (EU, UK)
	cleanup /gc	VCPDA (Virginia)

Large-scale production

- ZippyDB
  25.2M/day
- UP2X
  92.5M merge through deletes

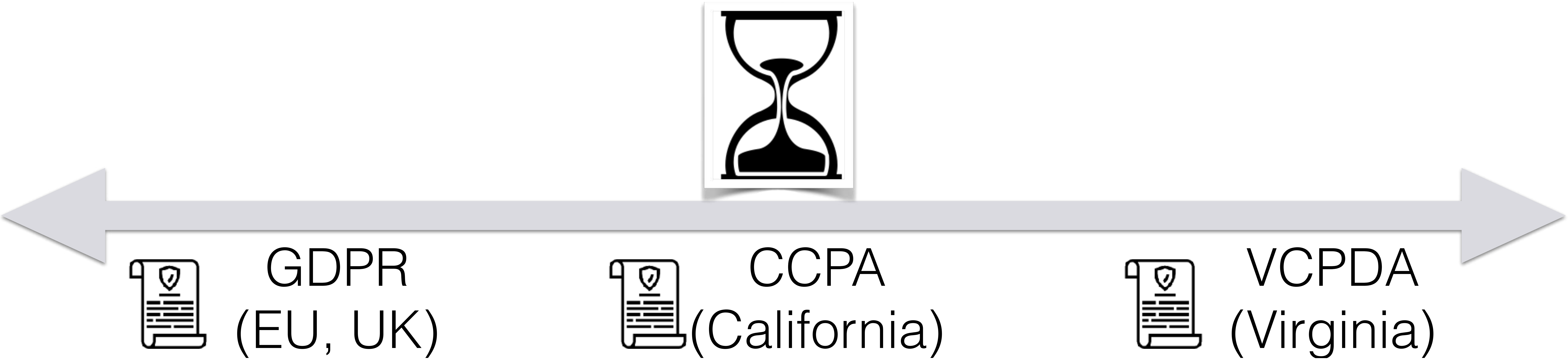
Internal db ops

- table drop
- data migration
- cleanup /gc

Privacy

- CCPA (California)
- GDPR (EU, UK)
- VCPDA (Virginia)

Large-scale production	Internal db ops	Privacy
ZippyDB **25.2M/day**	table drop	GDPR (EU, UK)
UP2X **92.5M merge through deletes**	data migration	CCPA (California)
	cleanup /gc	VCPDA (Virginia)



GDPR (EU, UK)

CCPA (California)

VCPDA (Virginia)

on-demand

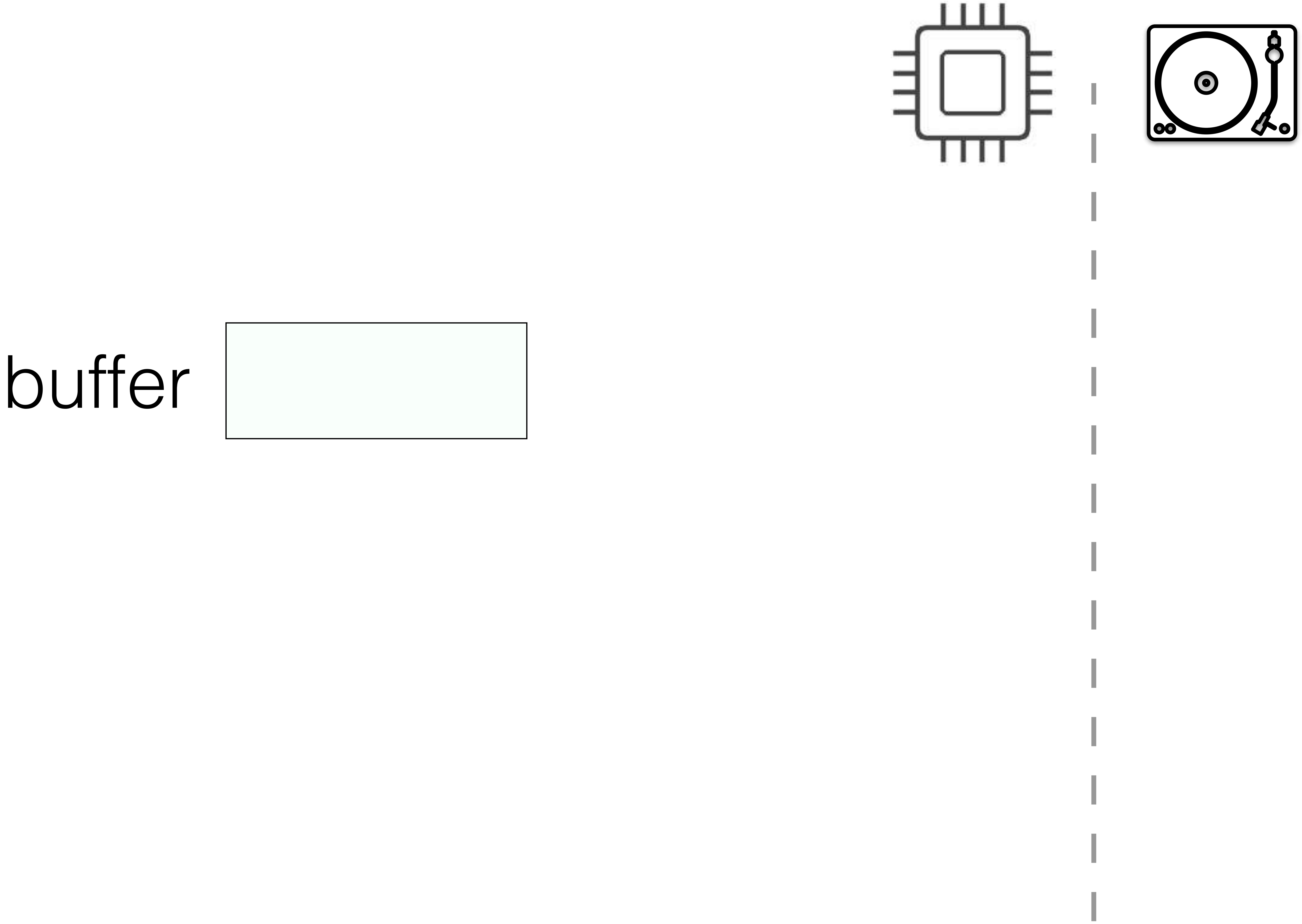
delete all data for user X within D days

rolling

keep deleting all data older than D days

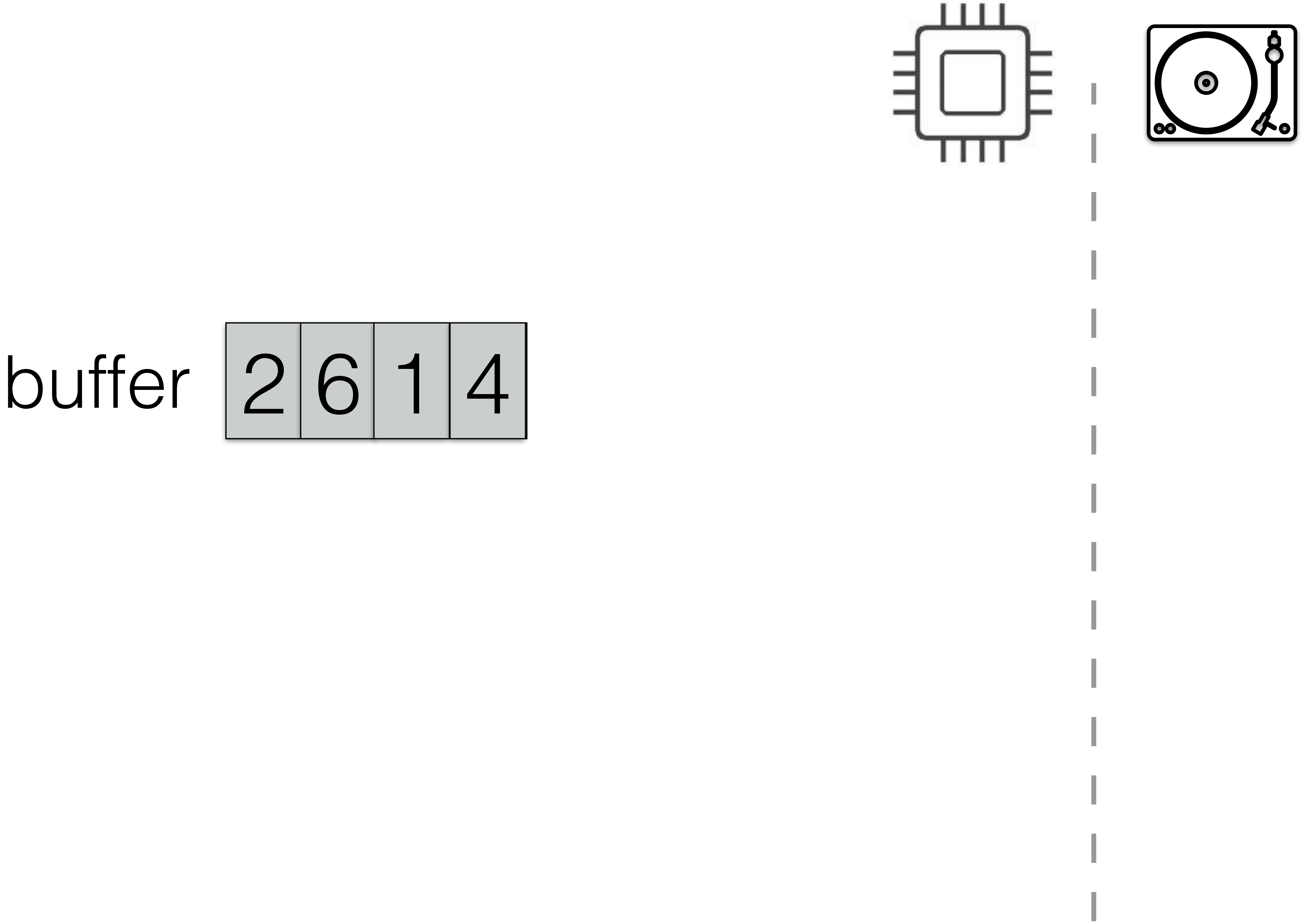
A reminder on how LSM-trees work!

log-structured merge-tree



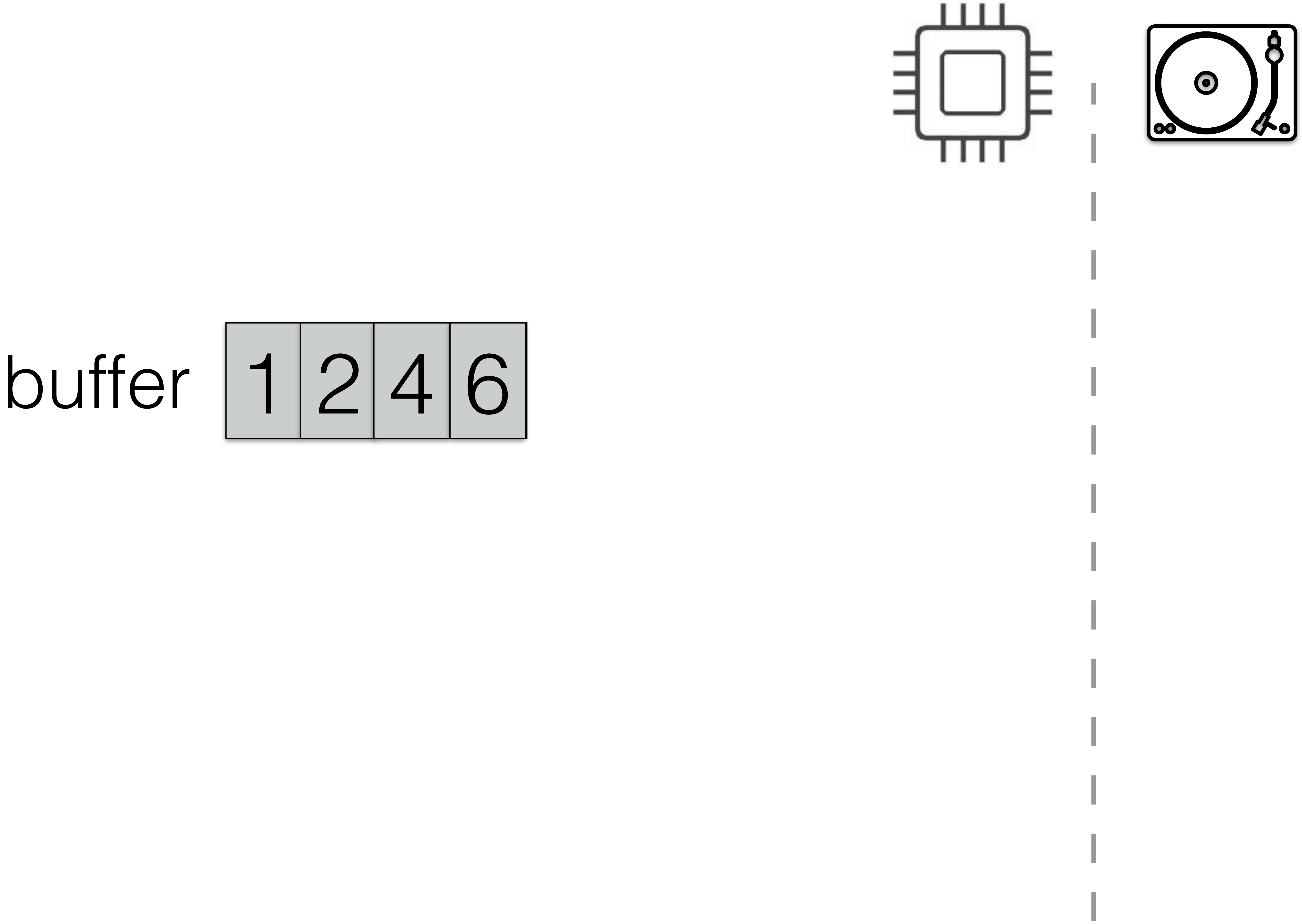
buffer

log-structured merge-tree



buffer	2	6	1	4

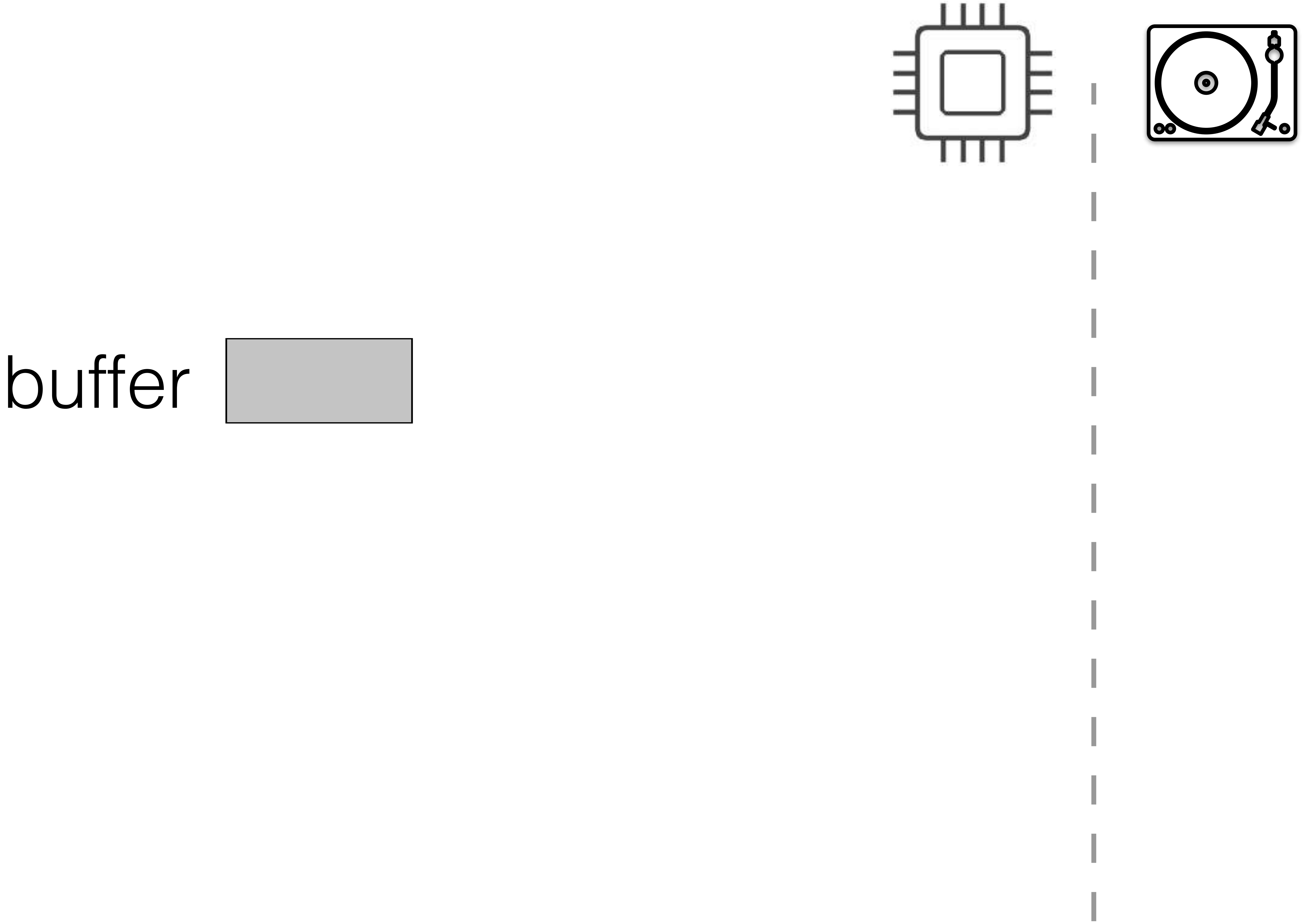
log-structured merge-tree



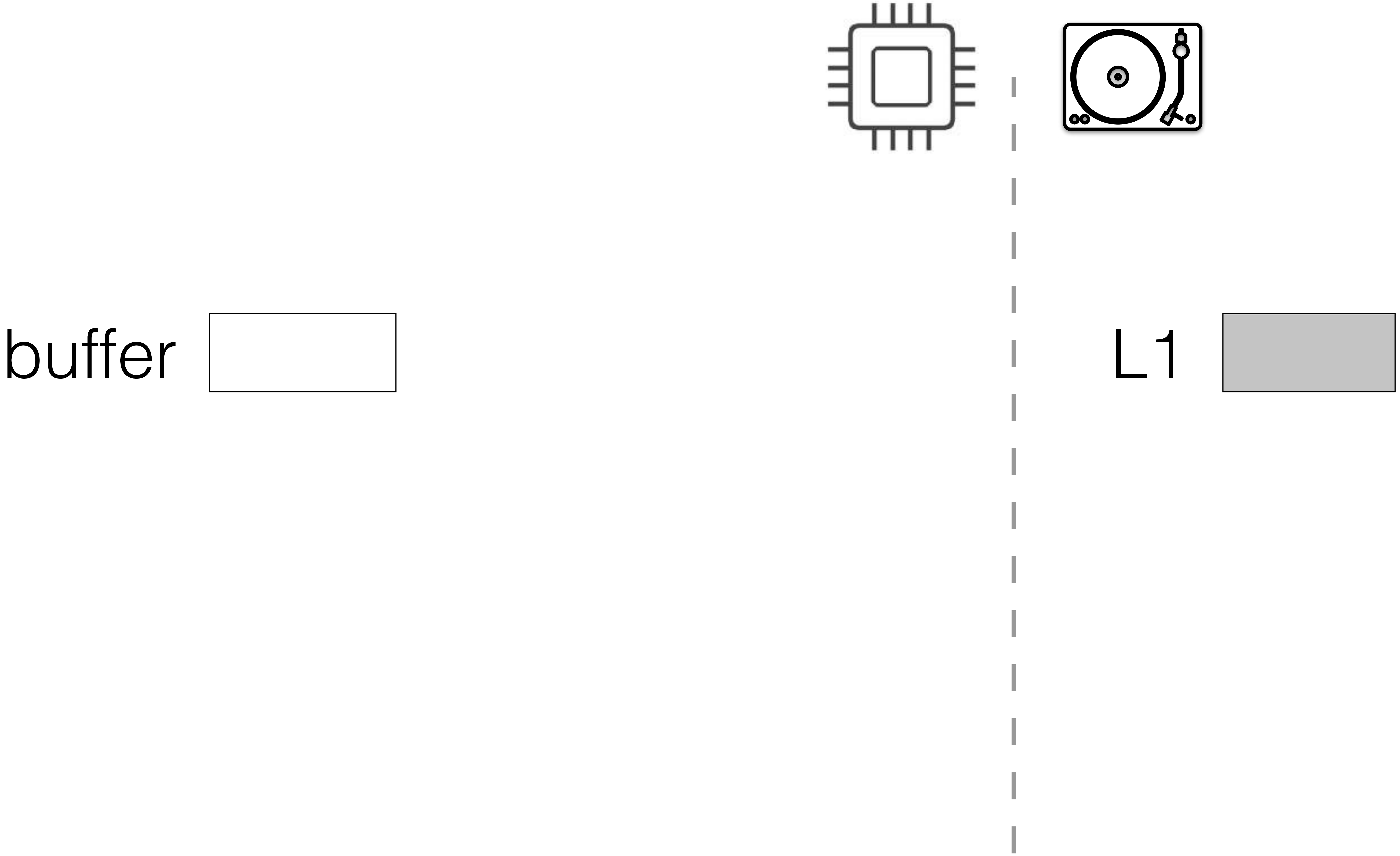
buffer

1	2	4	6

log-structured merge-tree



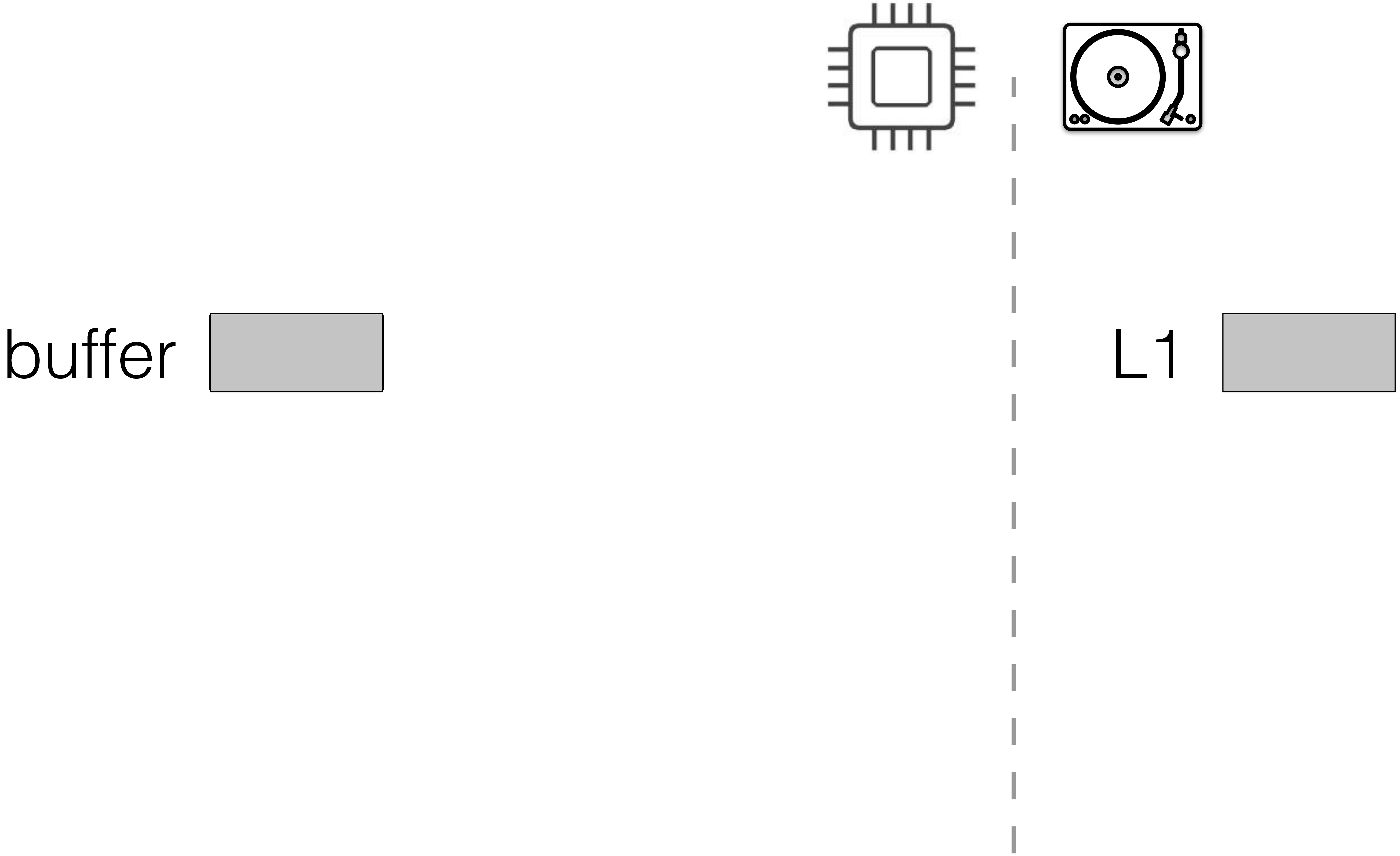
log-structured merge-tree



buffer

L1

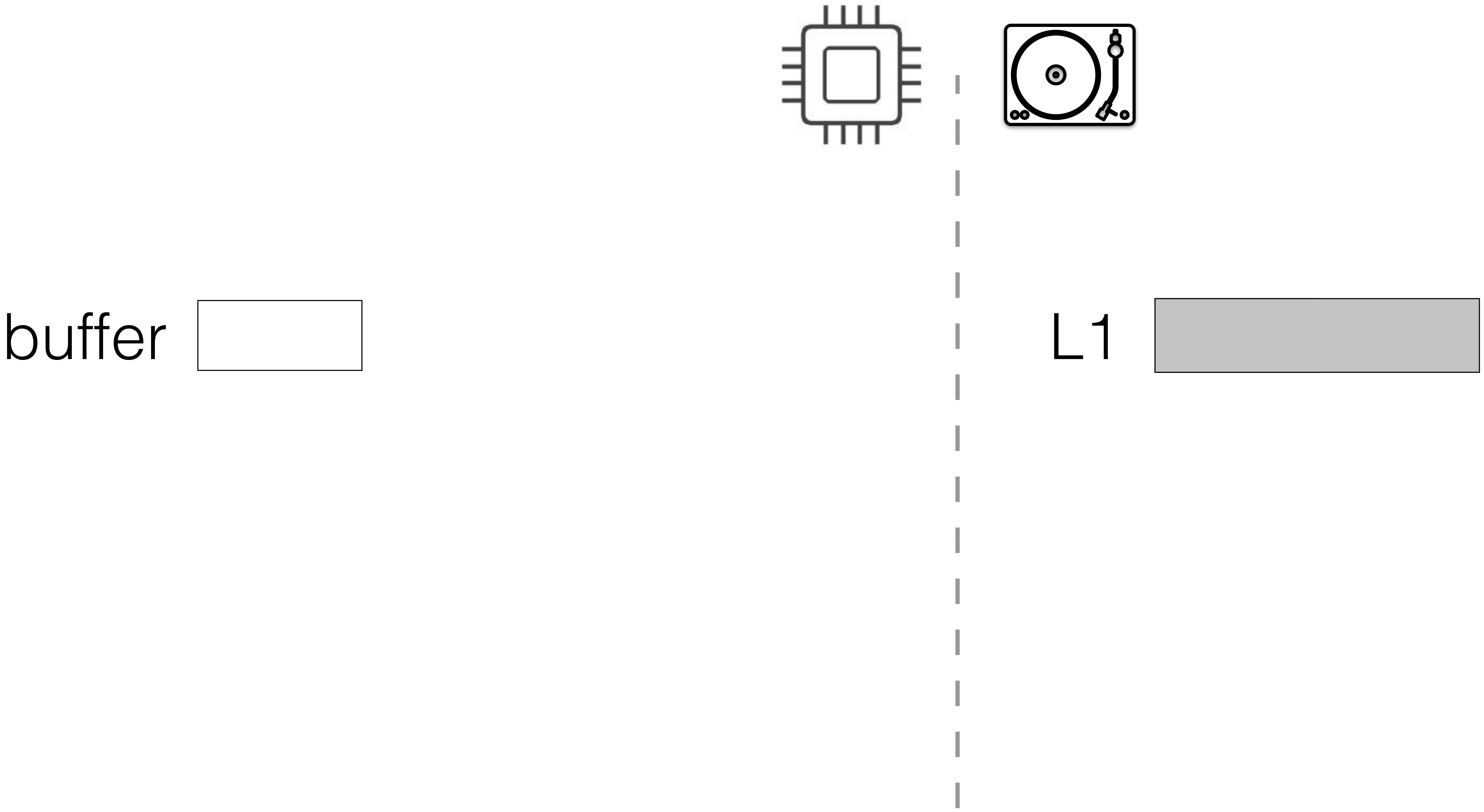
log-structured merge-tree



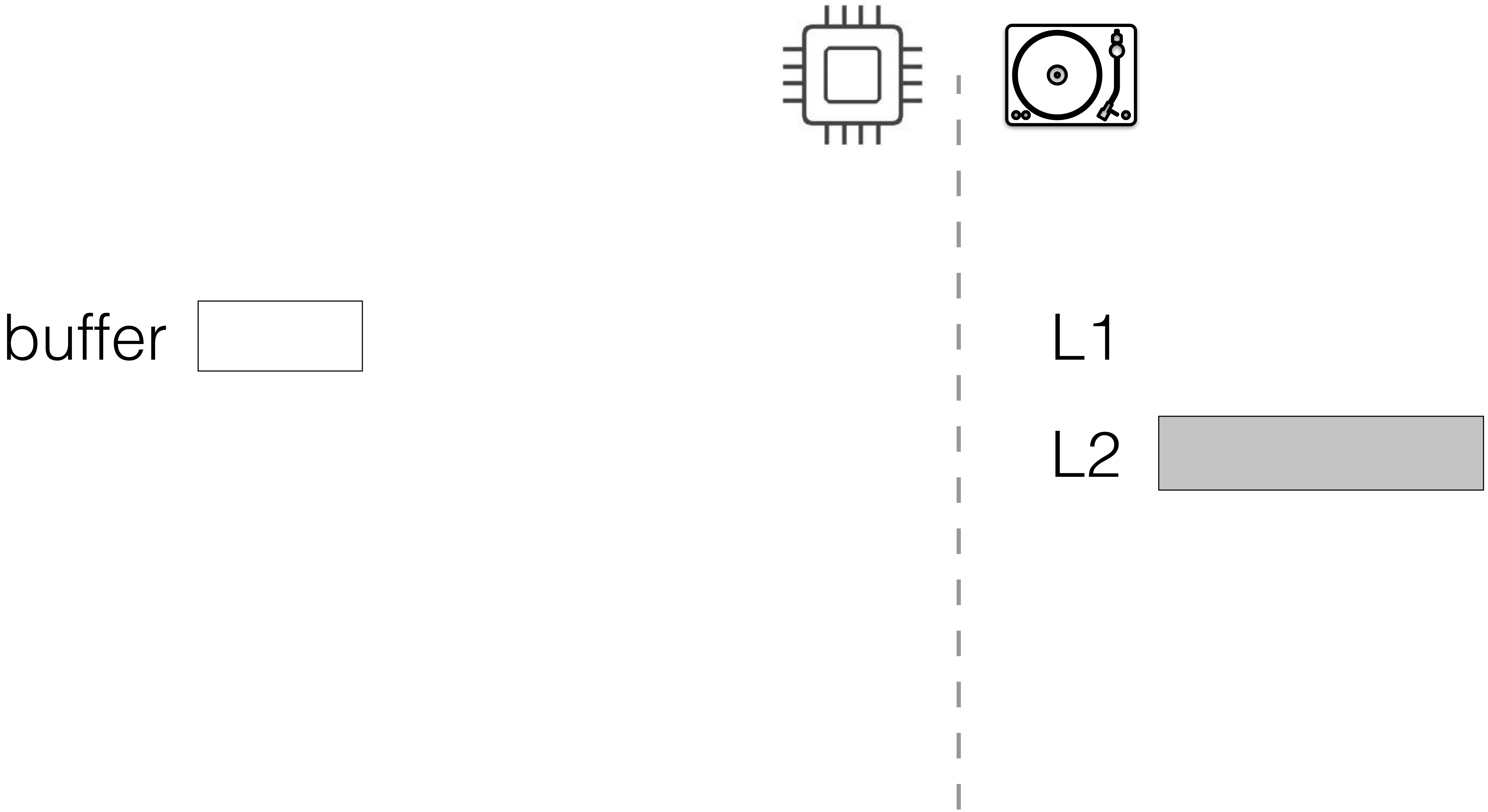
buffer

L1

log-structured merge-tree



log-structured merge-tree



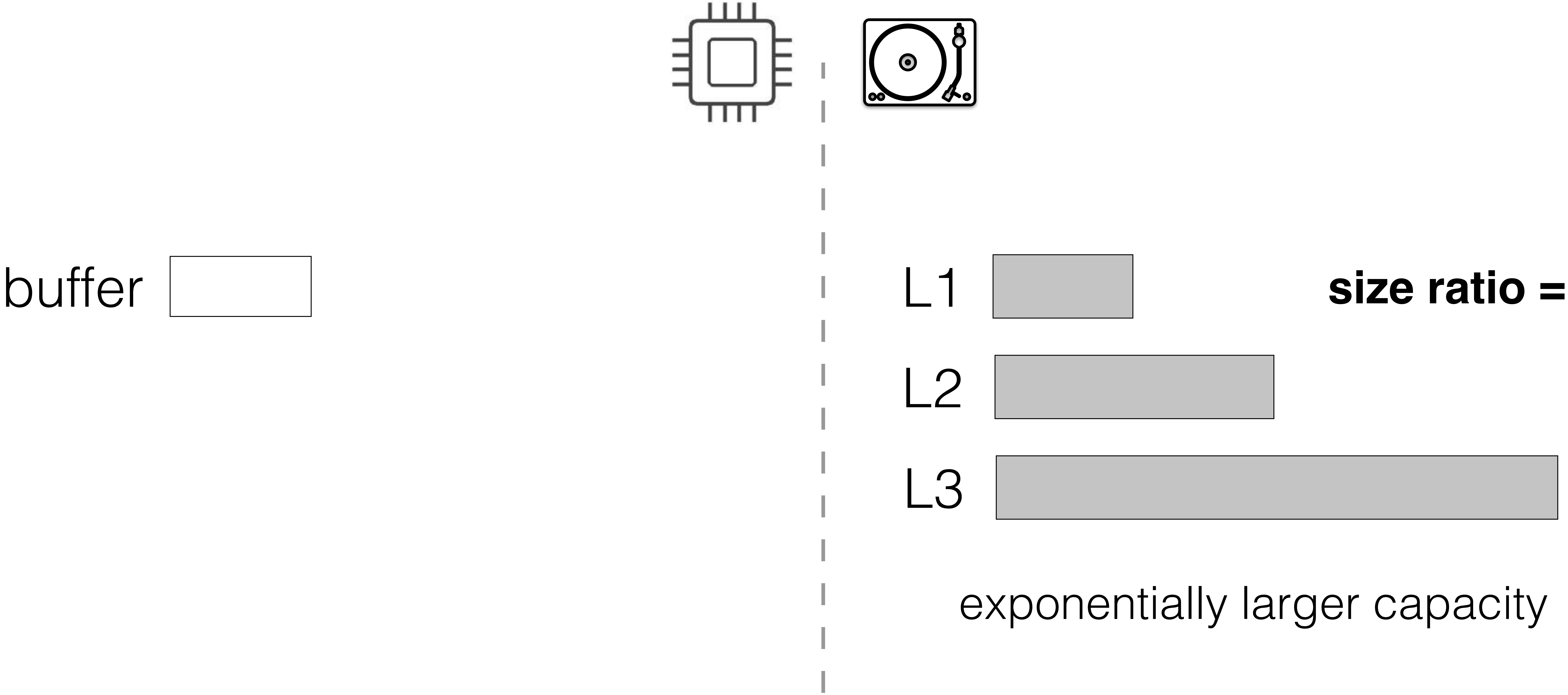
buffer

L1

L2

compaction

log-structured merge-tree



buffer

L1

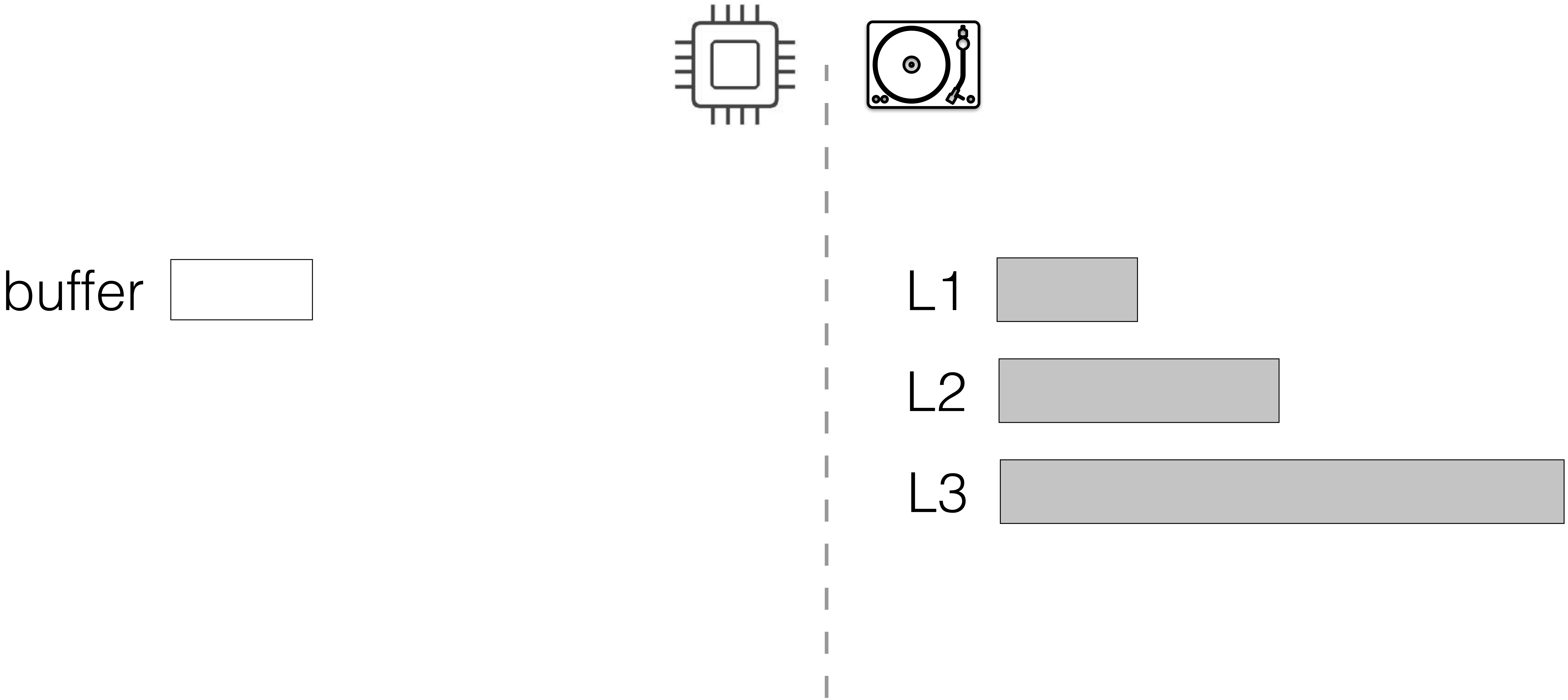
L2

L3

size ratio = T

exponentially larger capacity

log-structured merge-tree



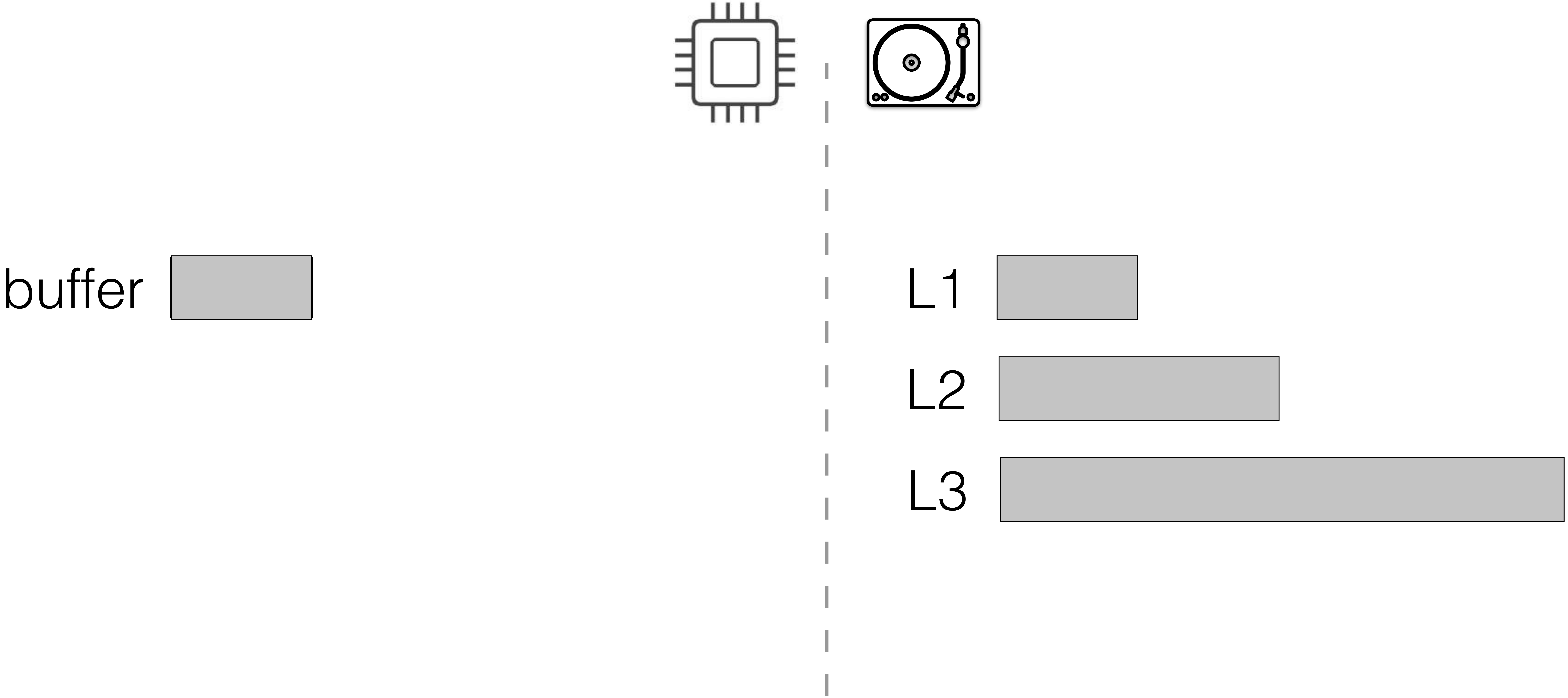
buffer

L1

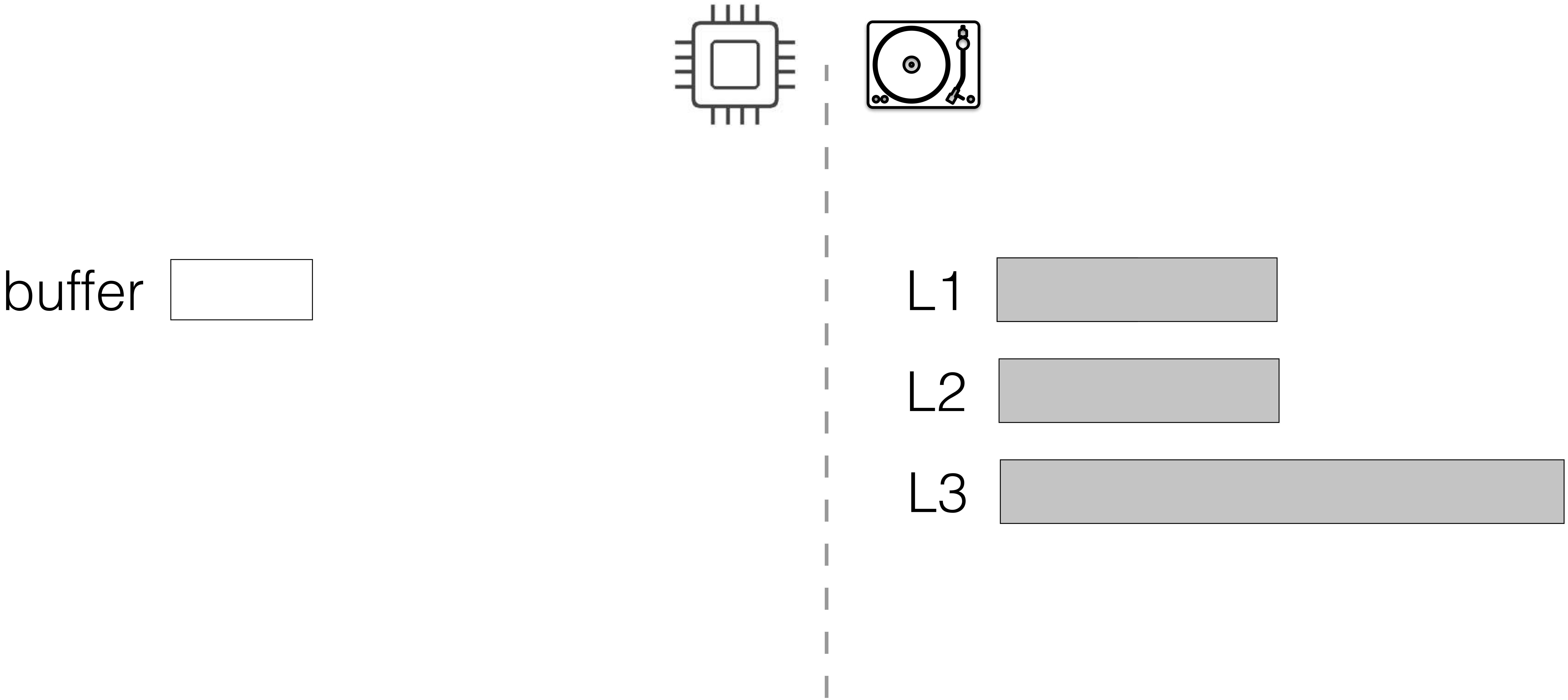
L2

L3

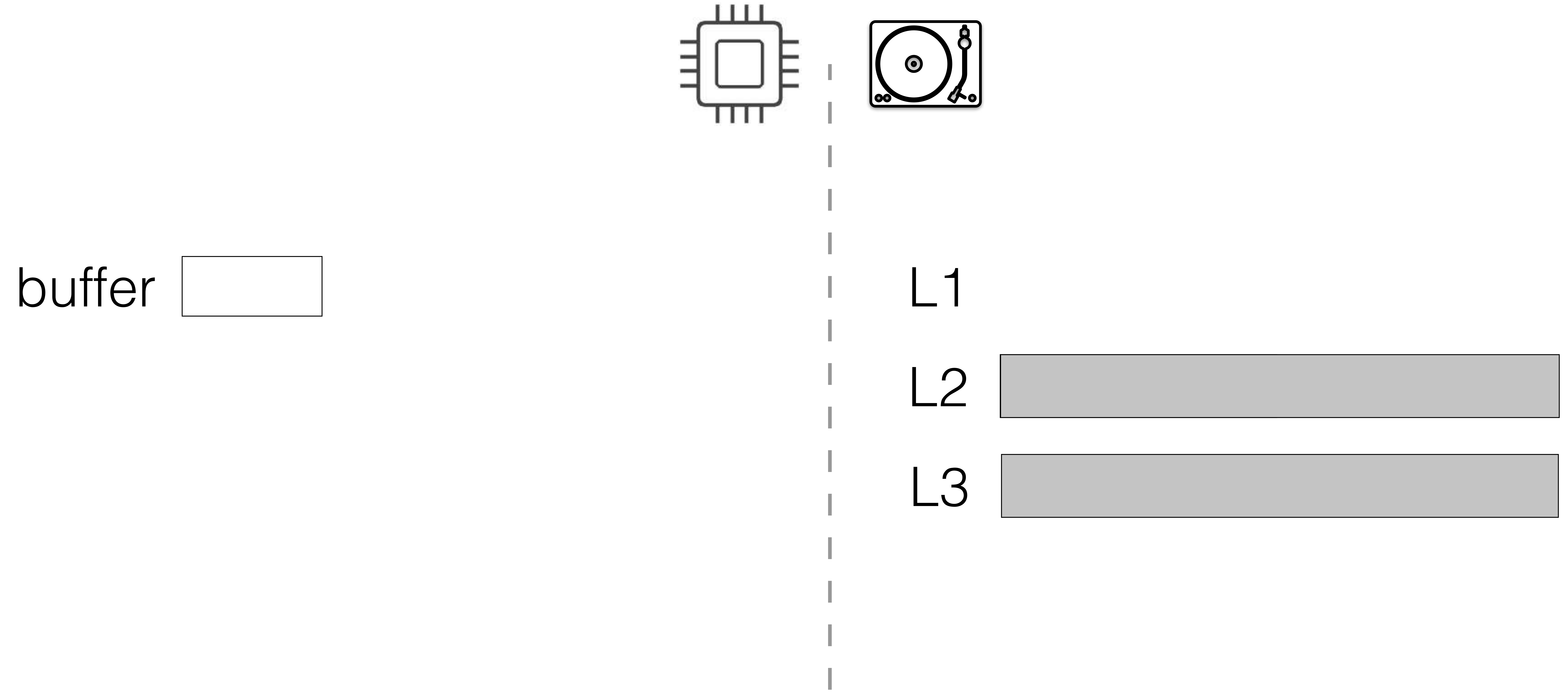
log-structured merge-tree



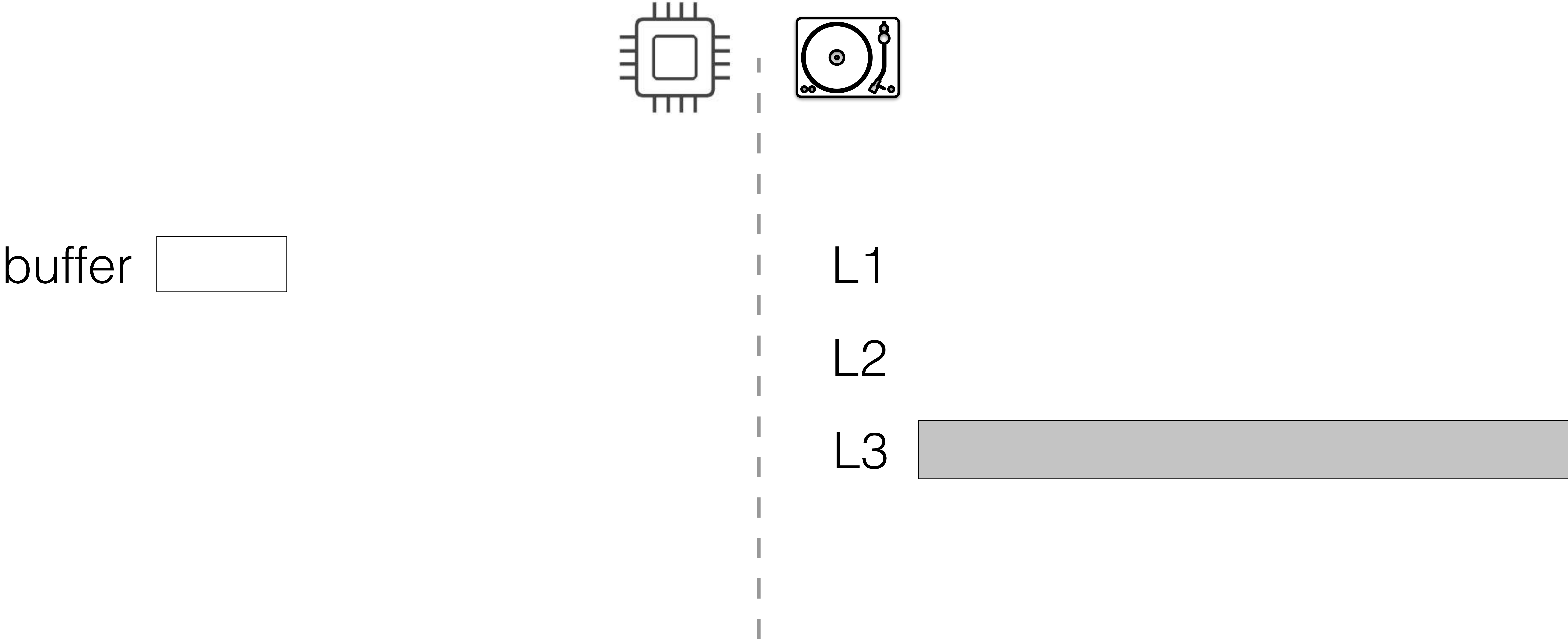
log-structured merge-tree



log-structured merge-tree



log-structured merge-tree



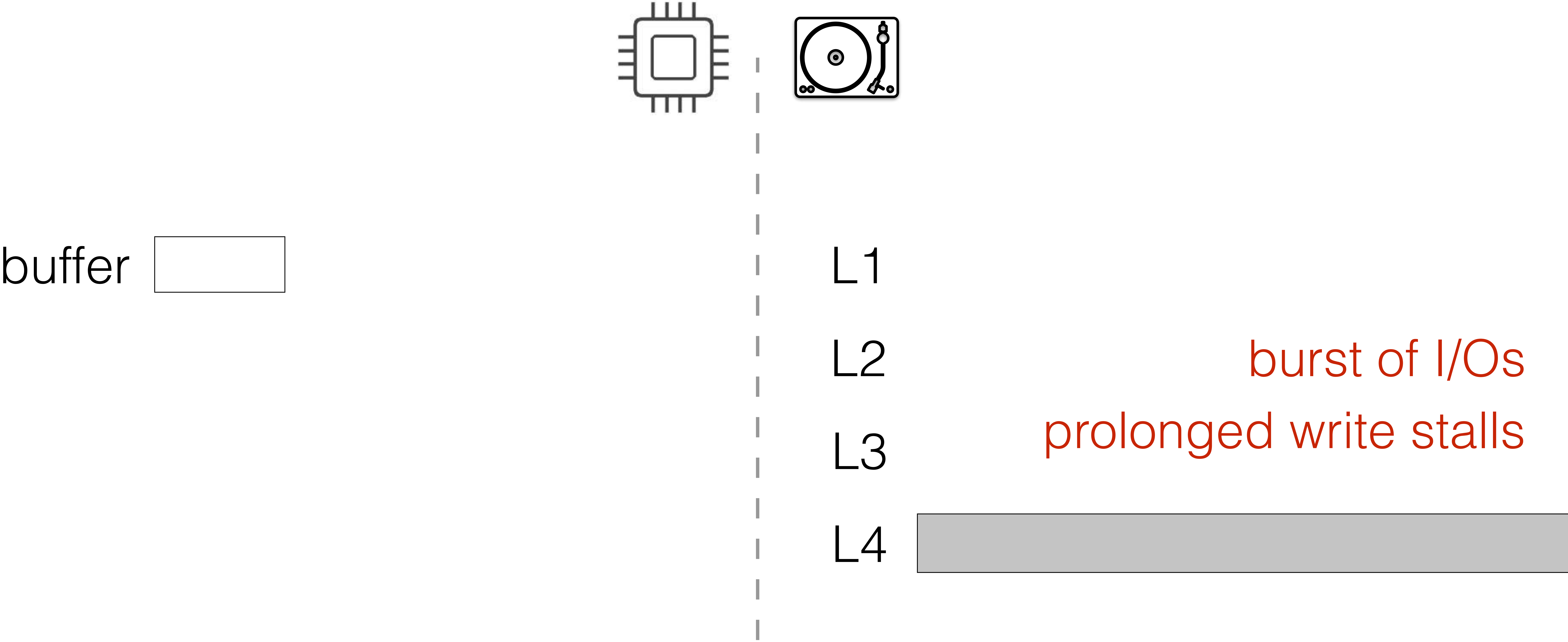
buffer

L1

L2

L3

log-structured merge-tree



buffer

L1

L2

L3

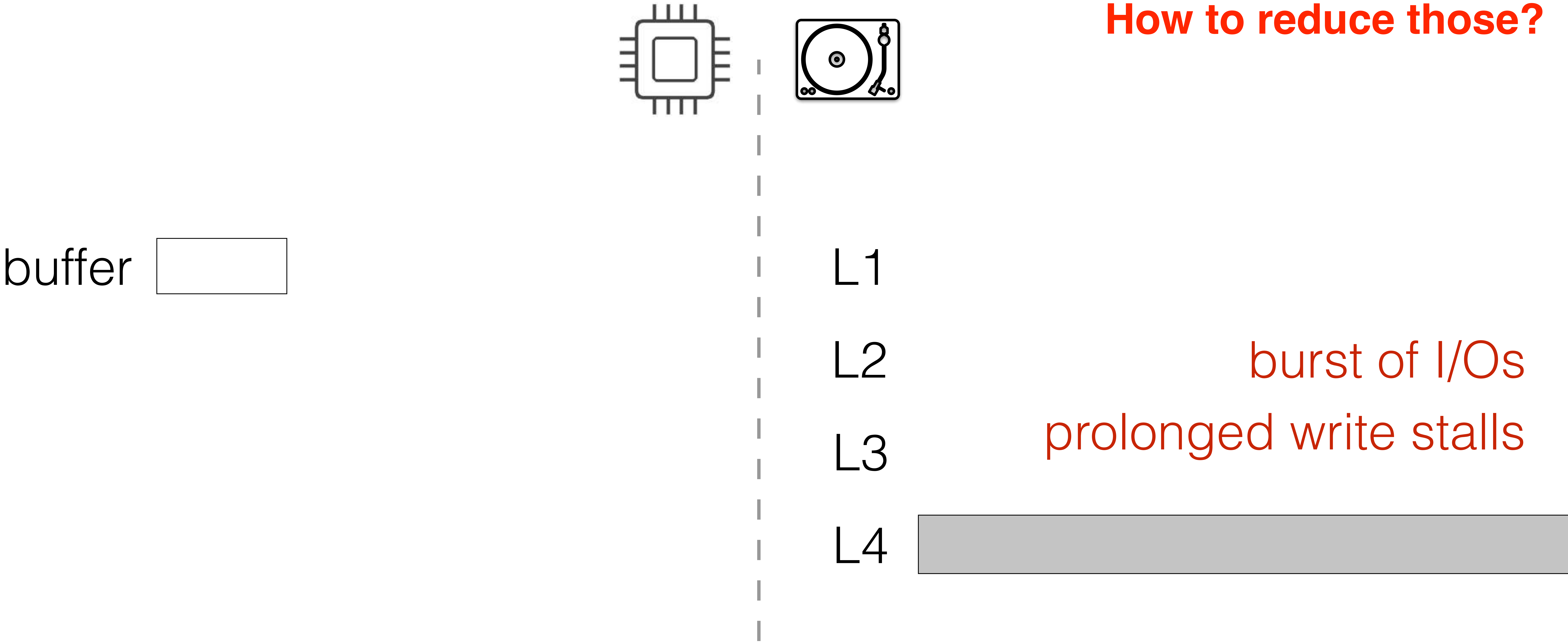
L4

burst of I/Os

prolonged write stalls

log-structured merge-tree

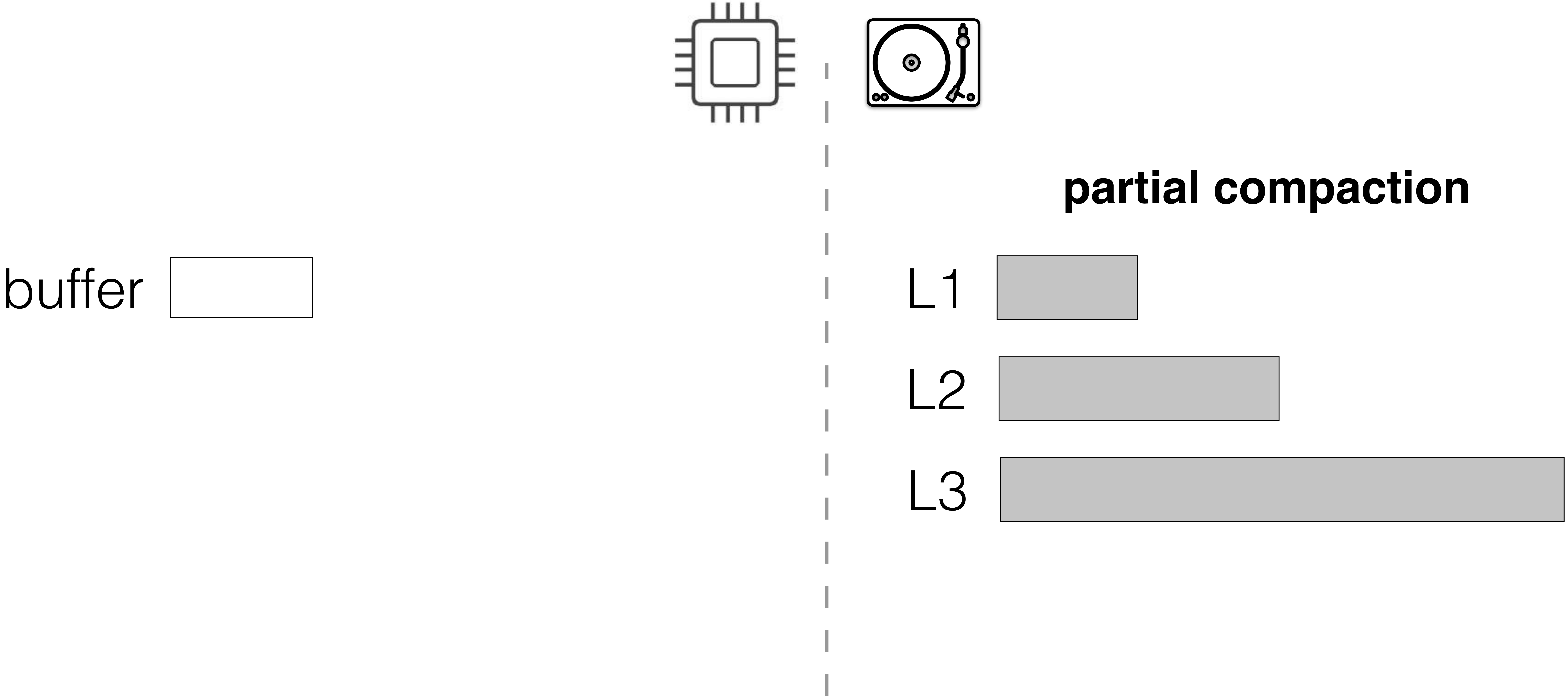
How to reduce those?



burst of I/Os

prolonged write stalls

log-structured merge-tree



partial compaction

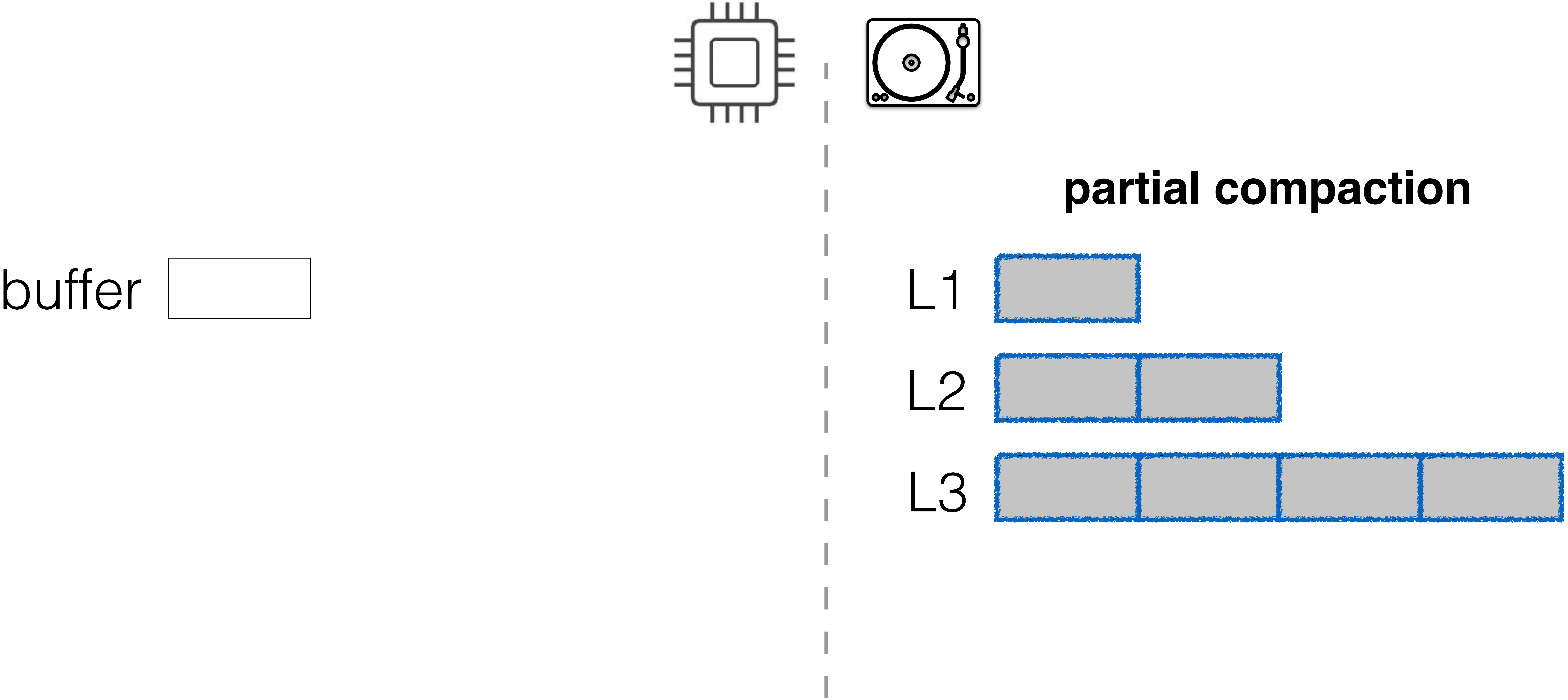
buffer

L1

L2

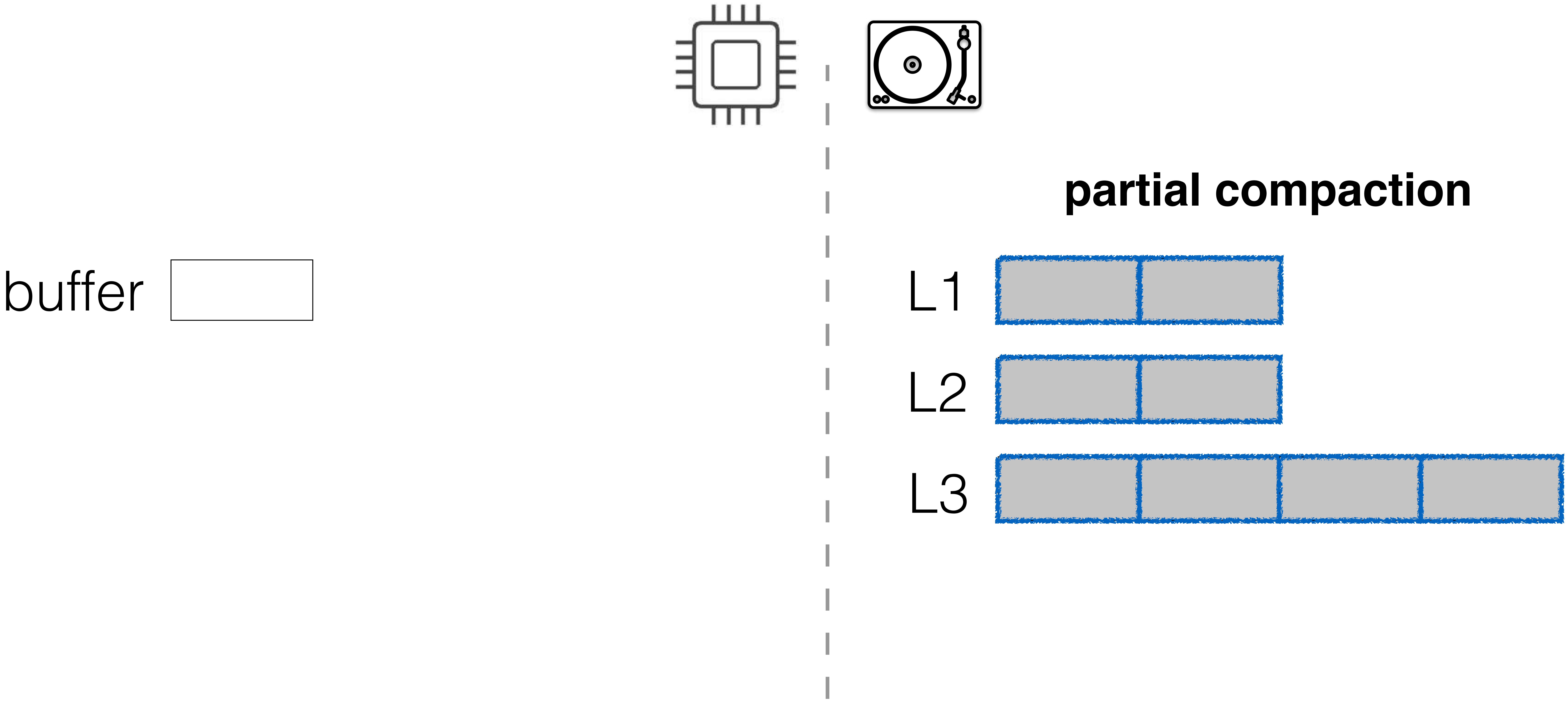
L3

log-structured merge-tree



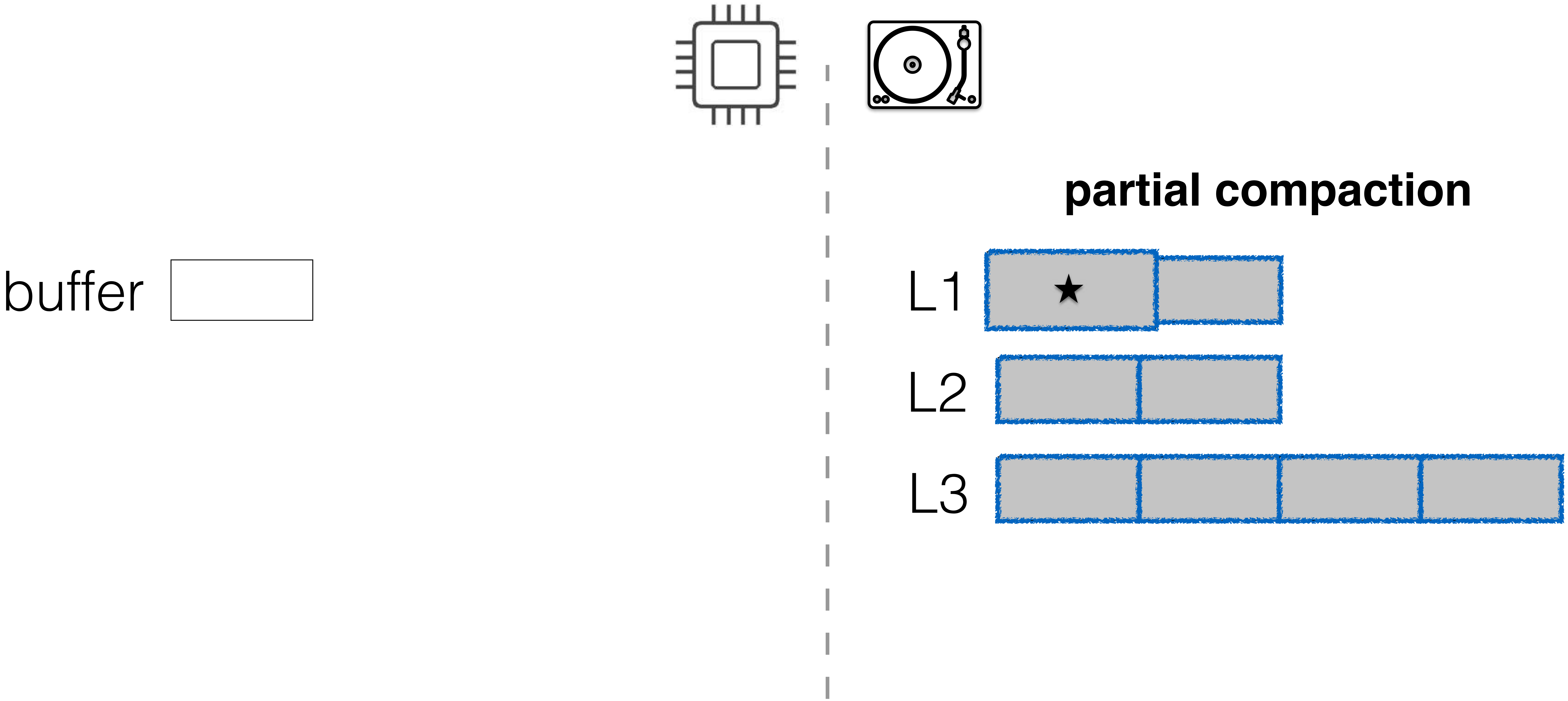
partial compaction

log-structured merge-tree



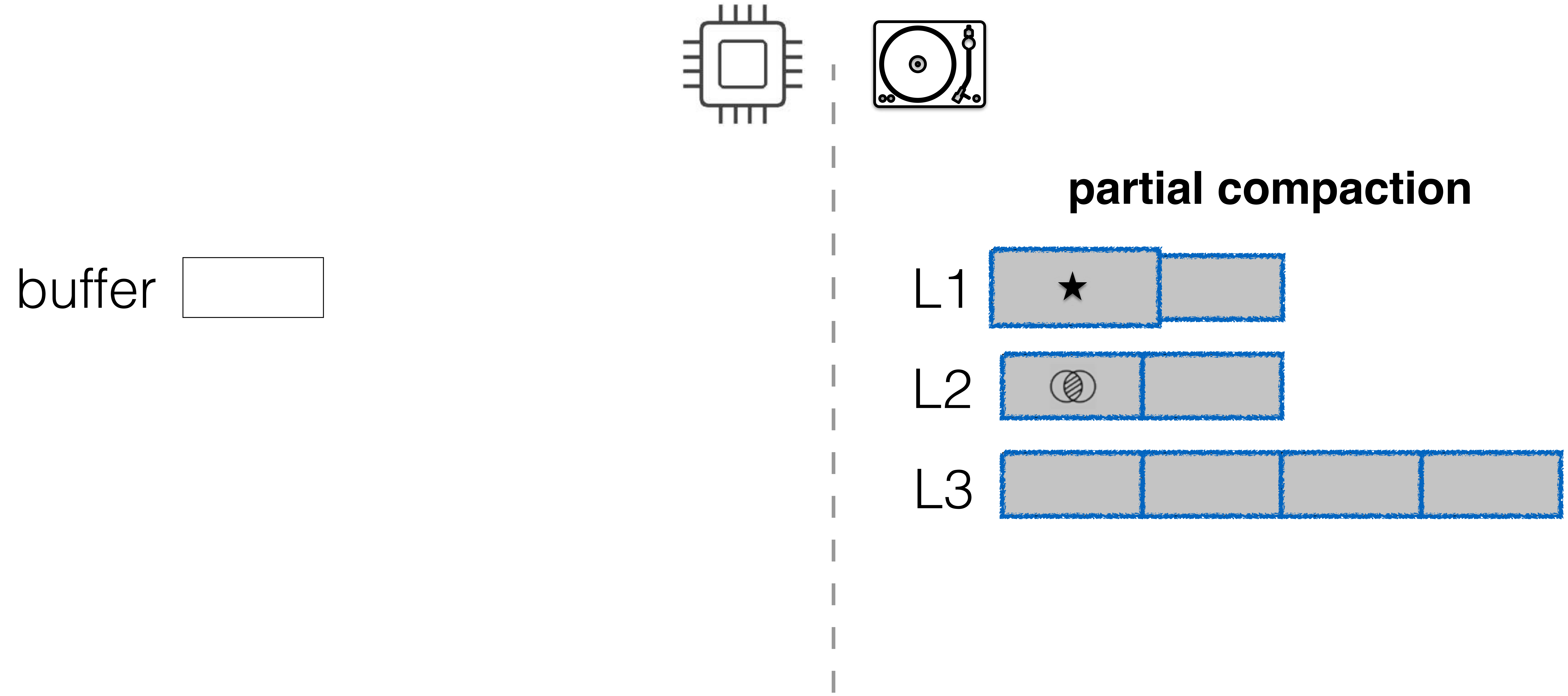
partial compaction

log-structured merge-tree

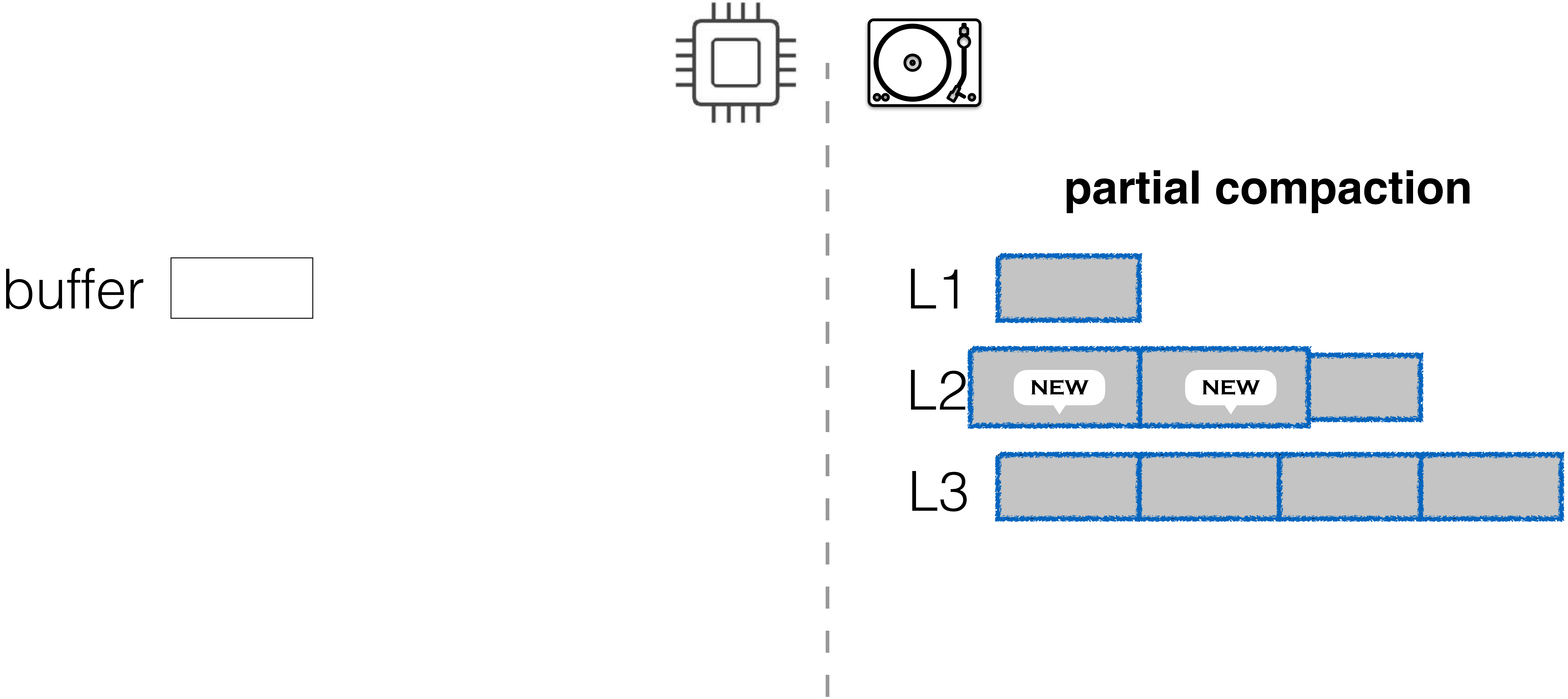


partial compaction

log-structured merge-tree



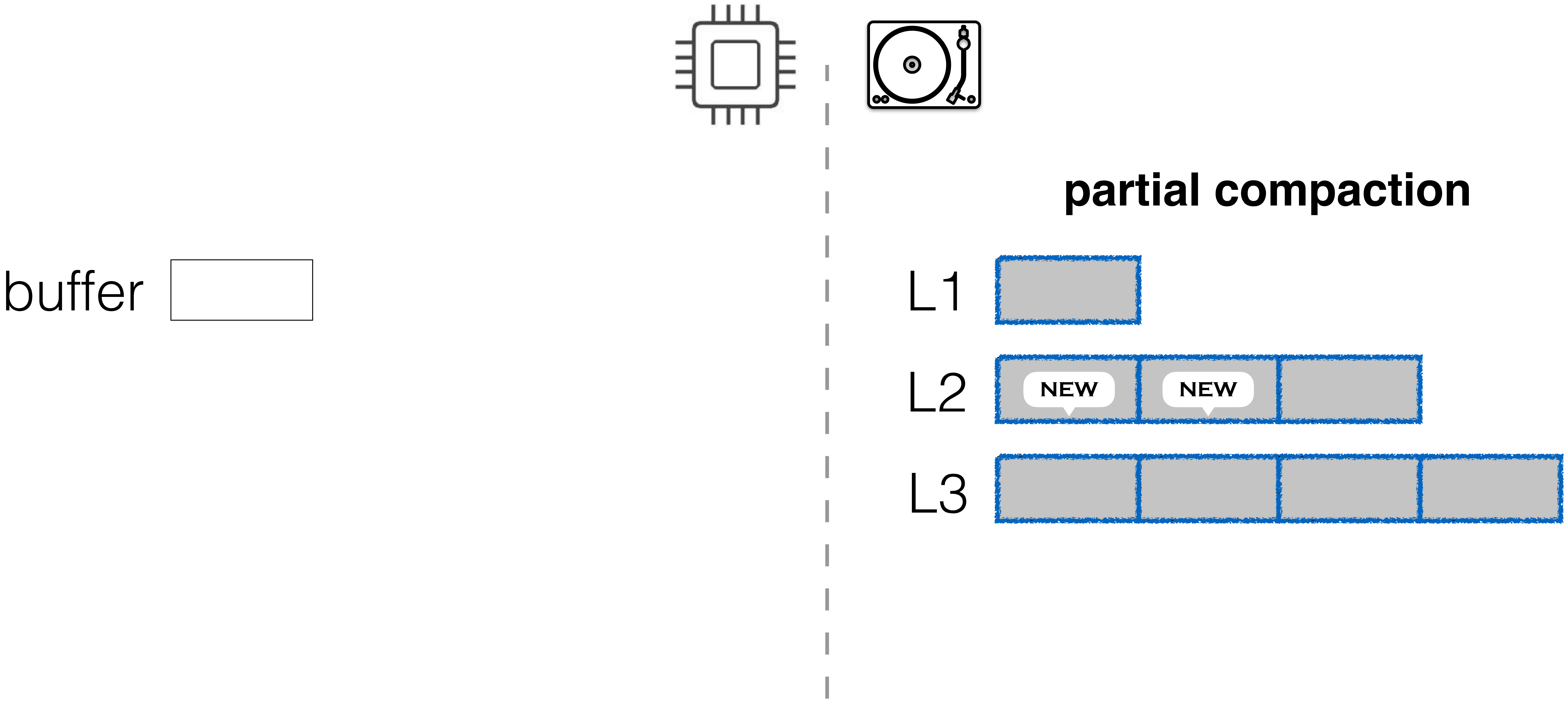
log-structured merge-tree



partial compaction

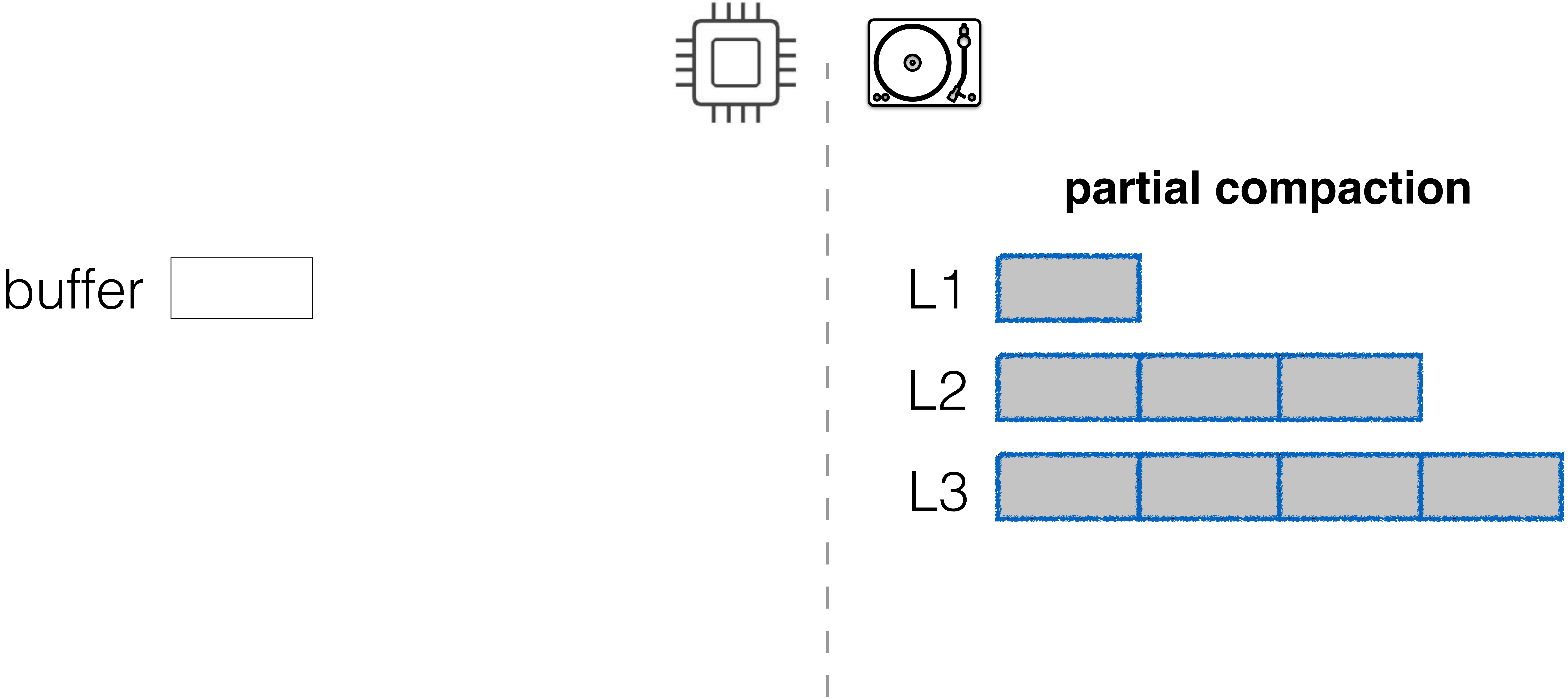
buffer

log-structured merge-tree



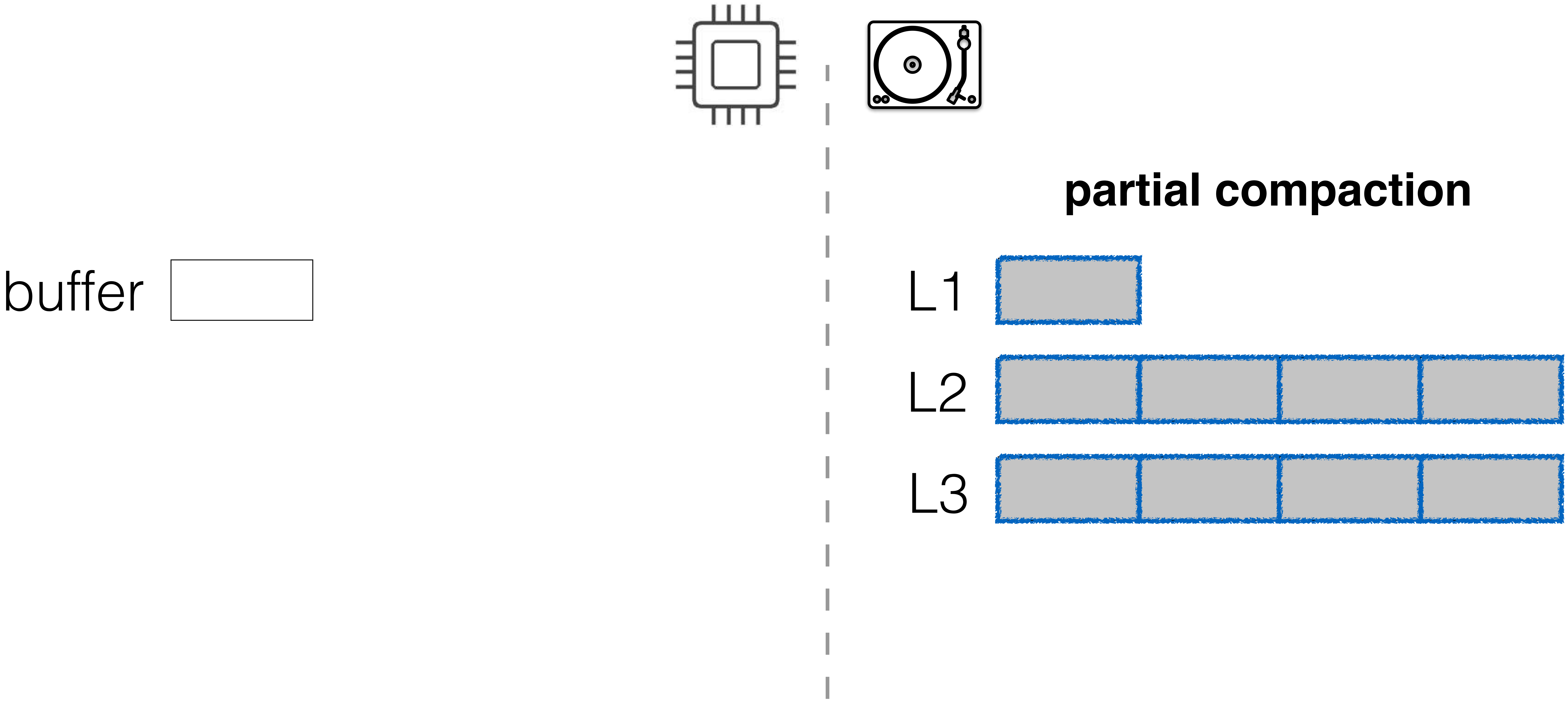
partial compaction

log-structured merge-tree



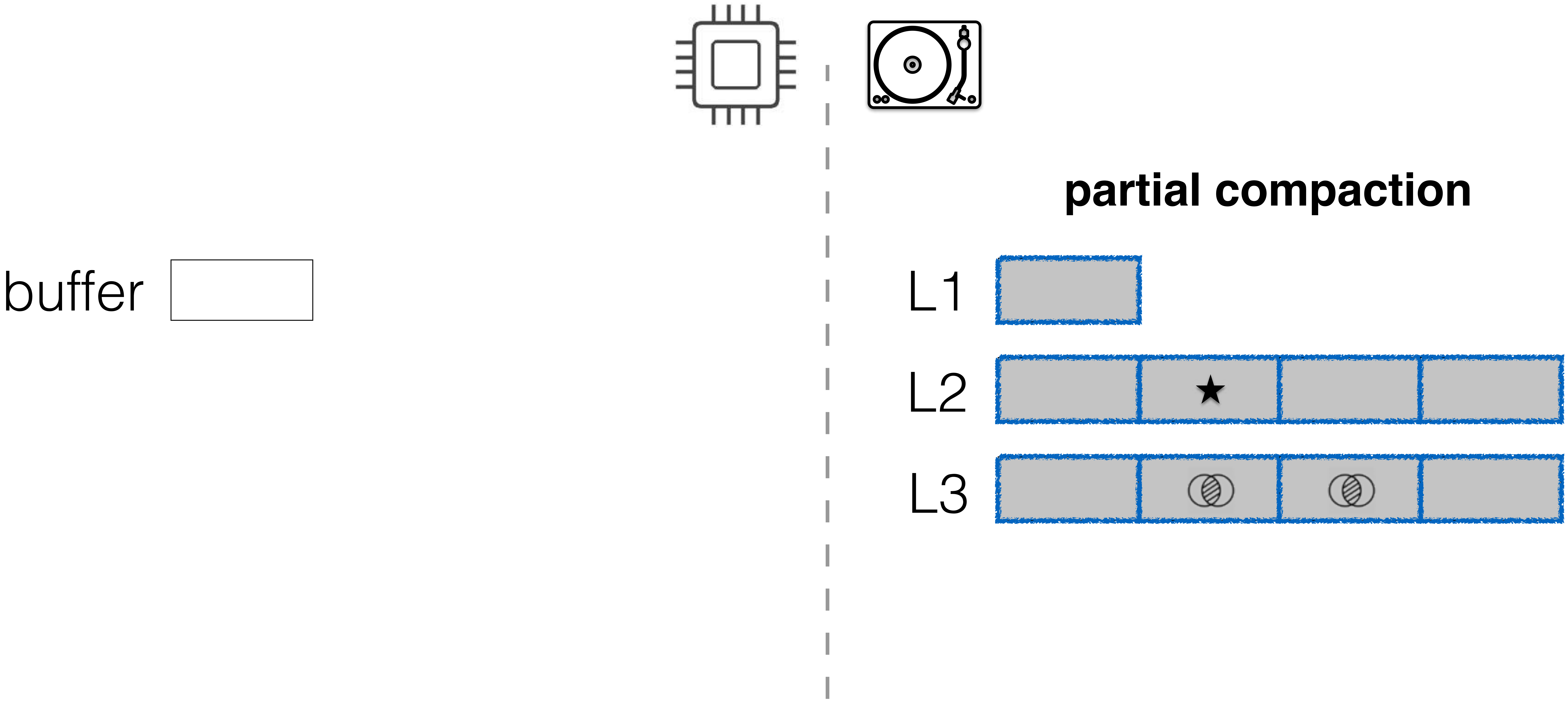
partial compaction

log-structured merge-tree



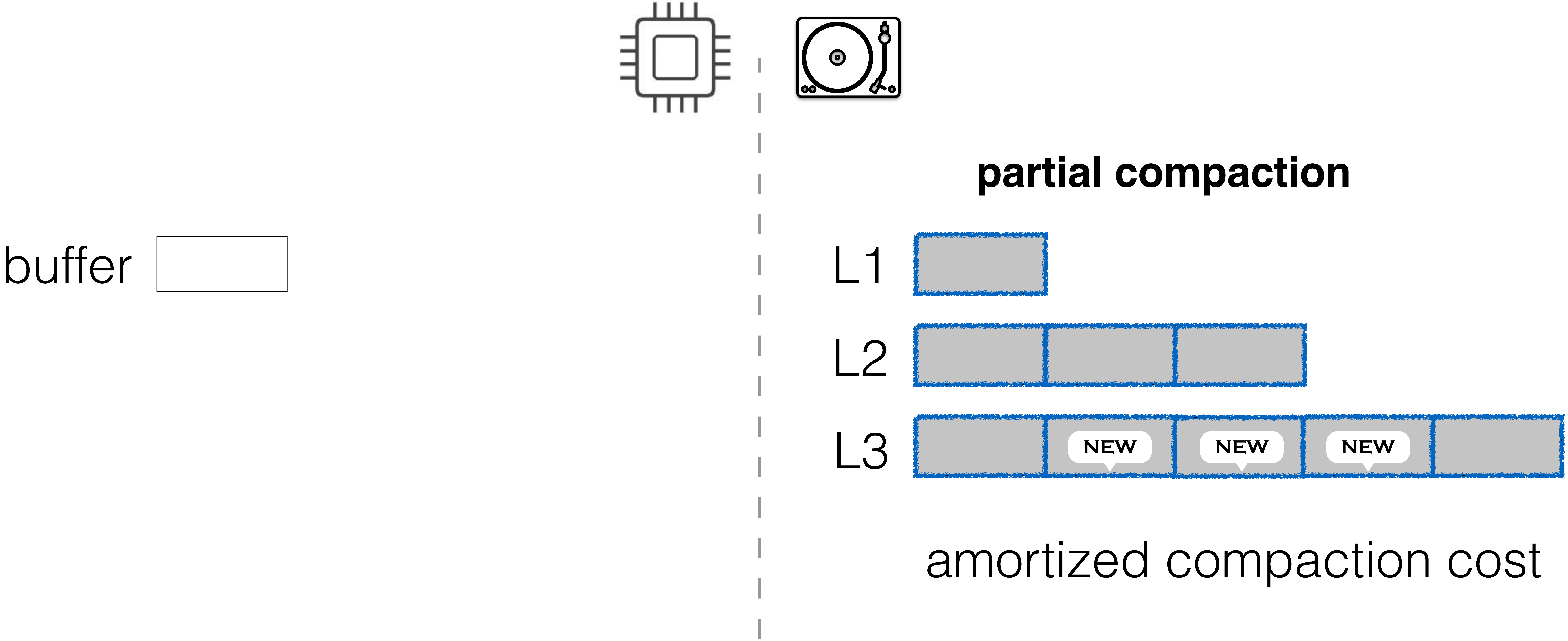
partial compaction

log-structured merge-tree



partial compaction

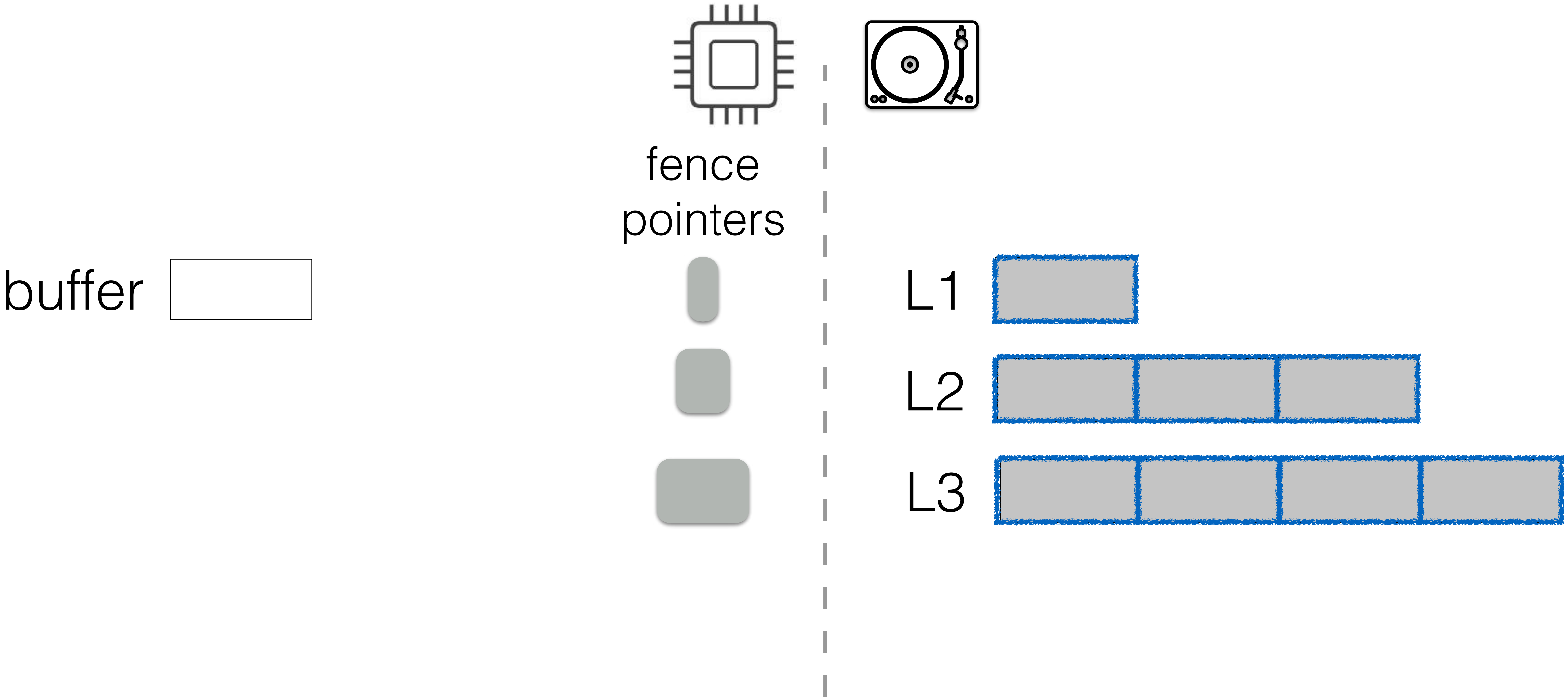
log-structured merge-tree



partial compaction

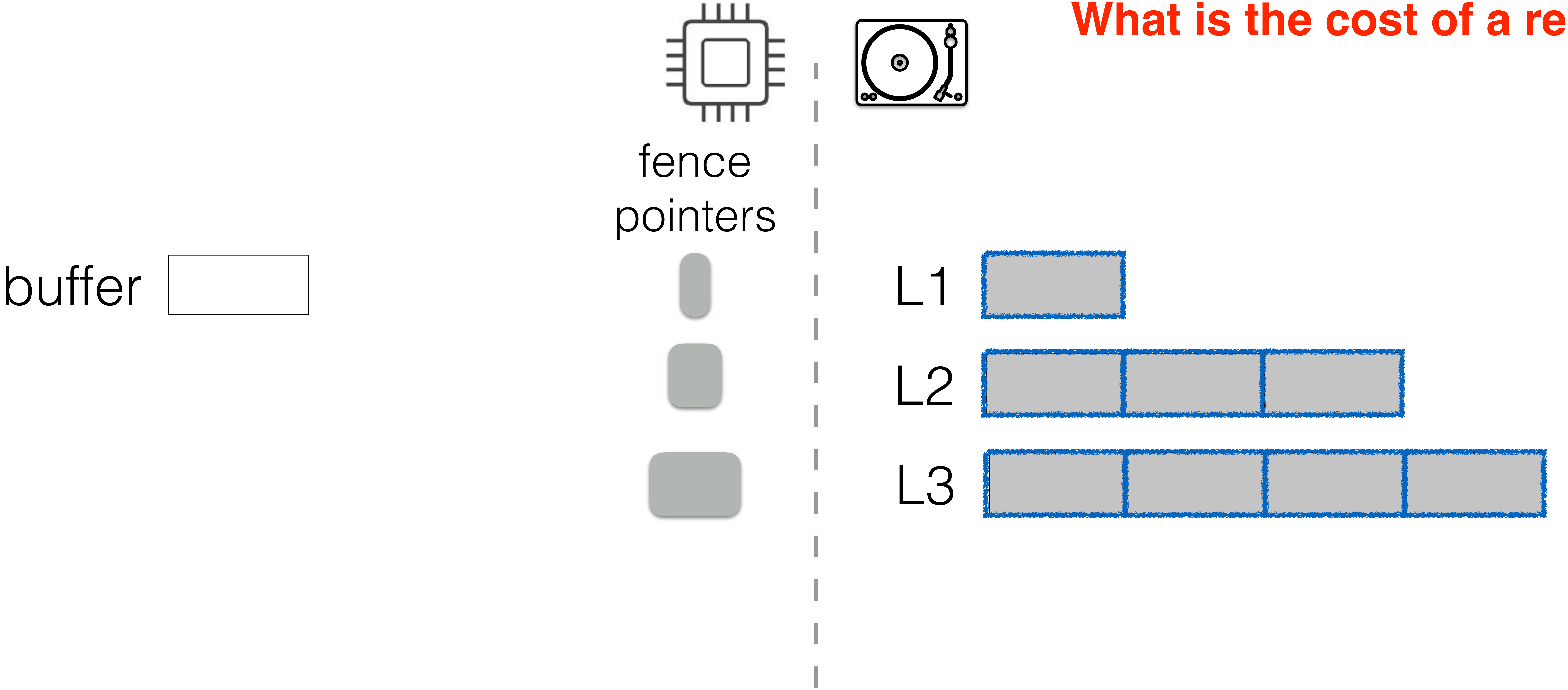
amortized compaction cost

log-structured merge-tree

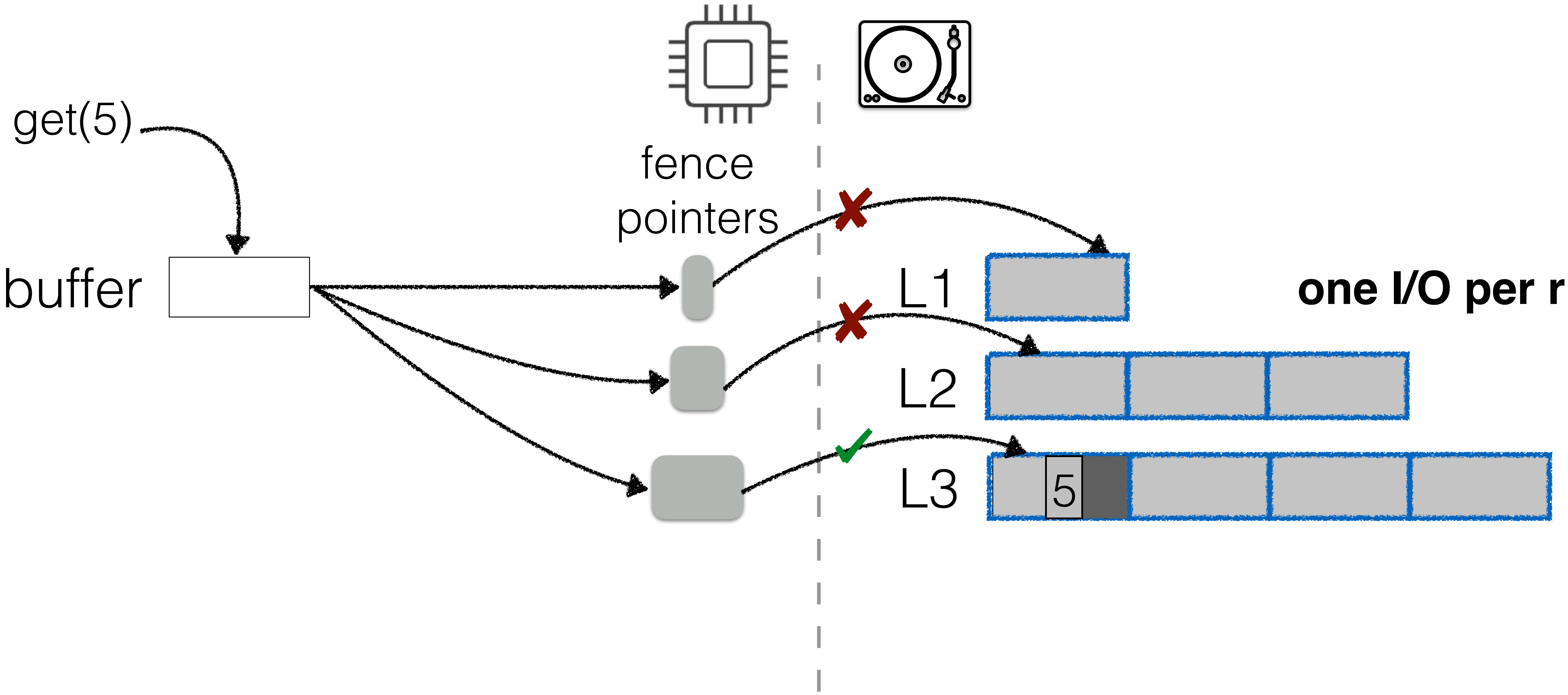


log-structured merge-tree

What is the cost of a read?

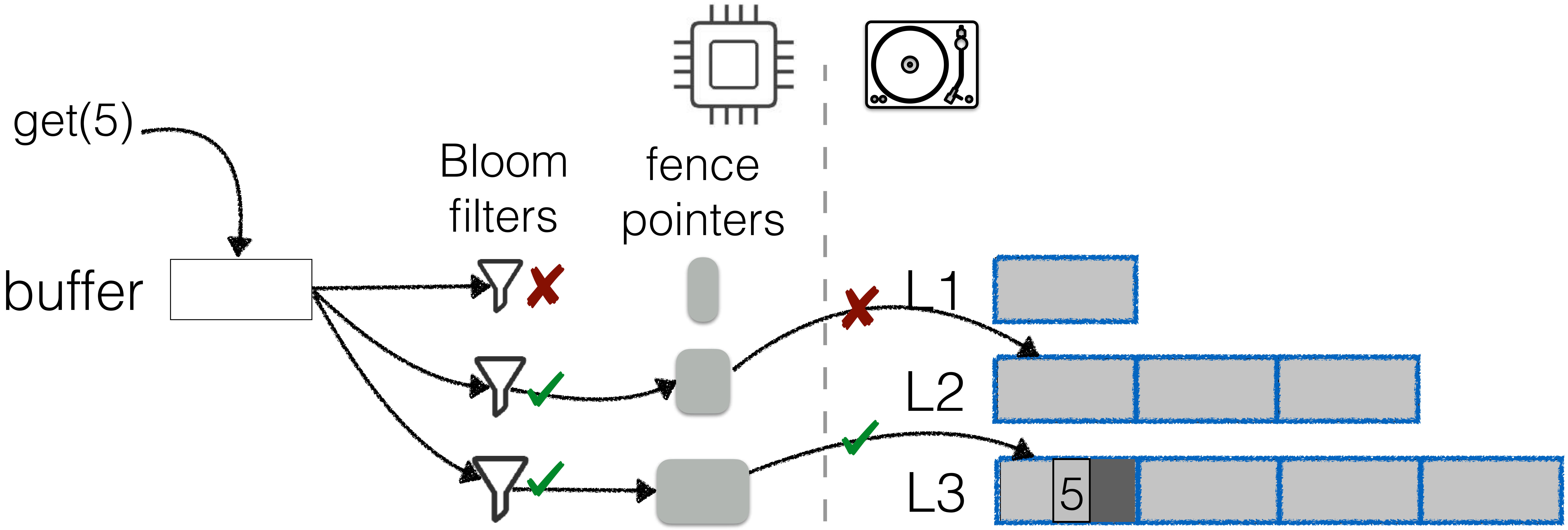


log-structured merge-tree



one I/O per run

log-structured merge-tree

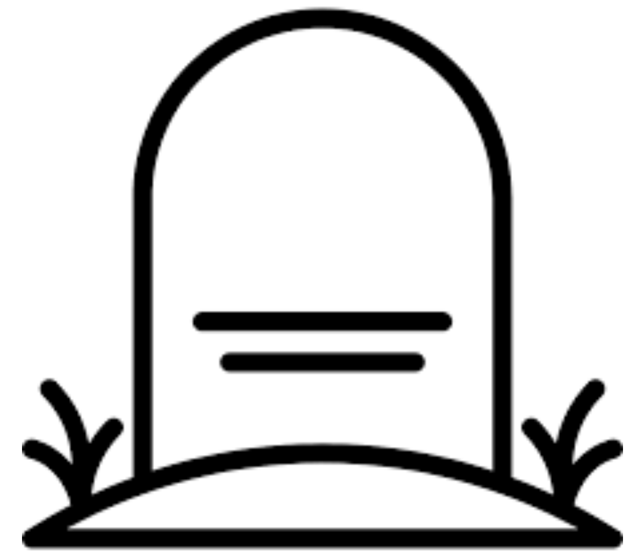


fewer disk I/Os

Now, let's talk about deletes!

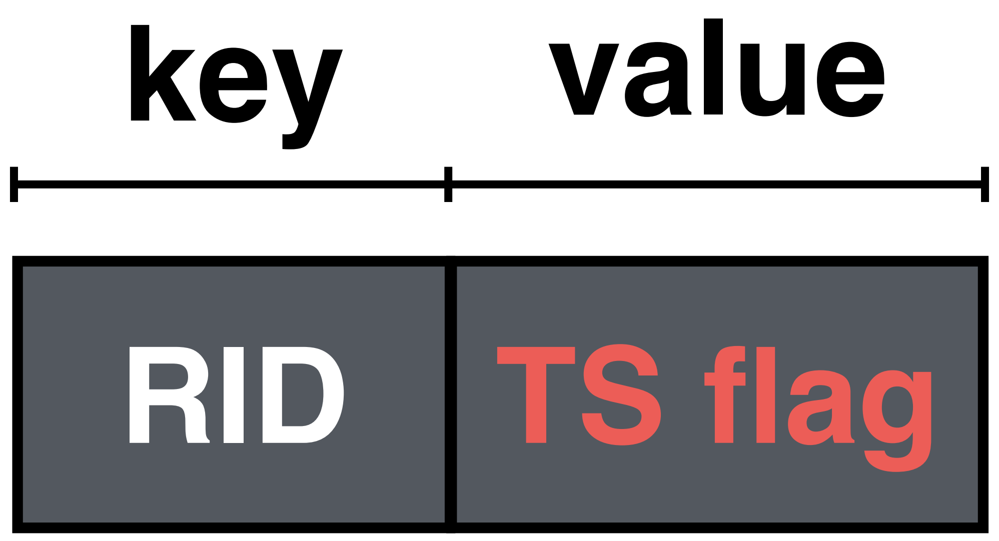
deletes in LSM-tree

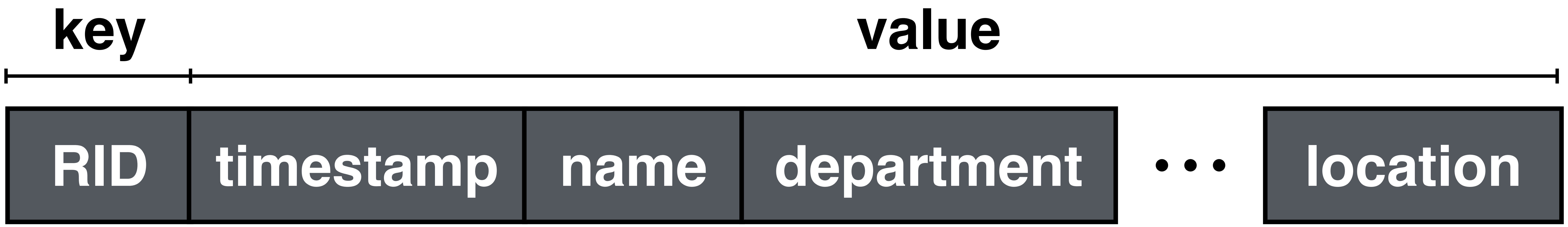
delete



deletes in LSM-tree

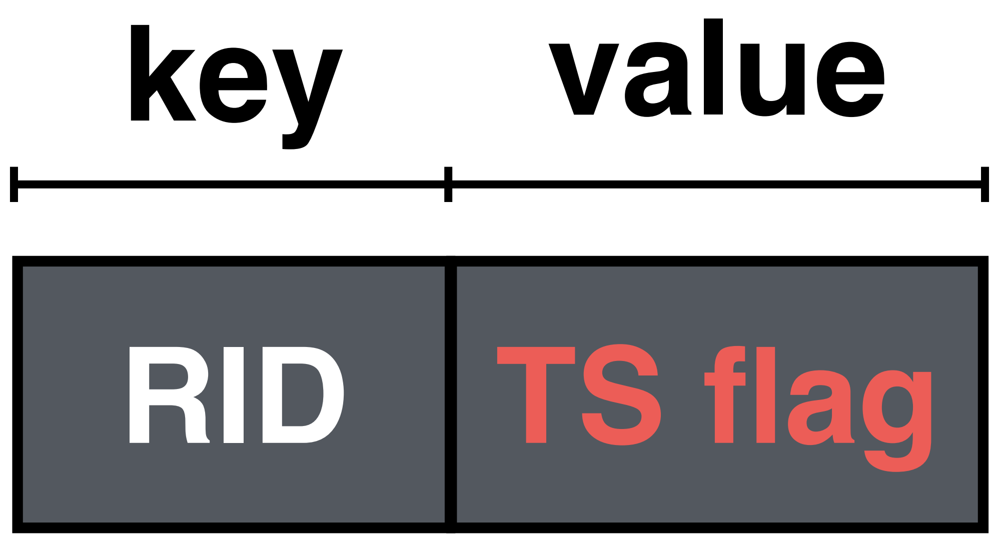
delete := insert tombstone





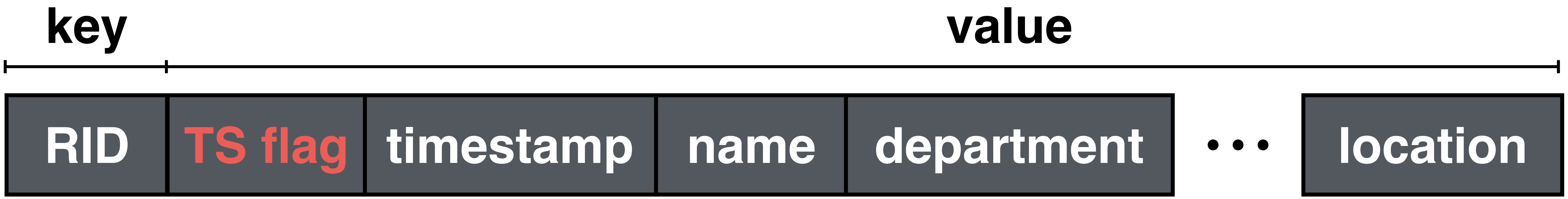
deletes in LSM-tree

delete := insert tombstone



key value

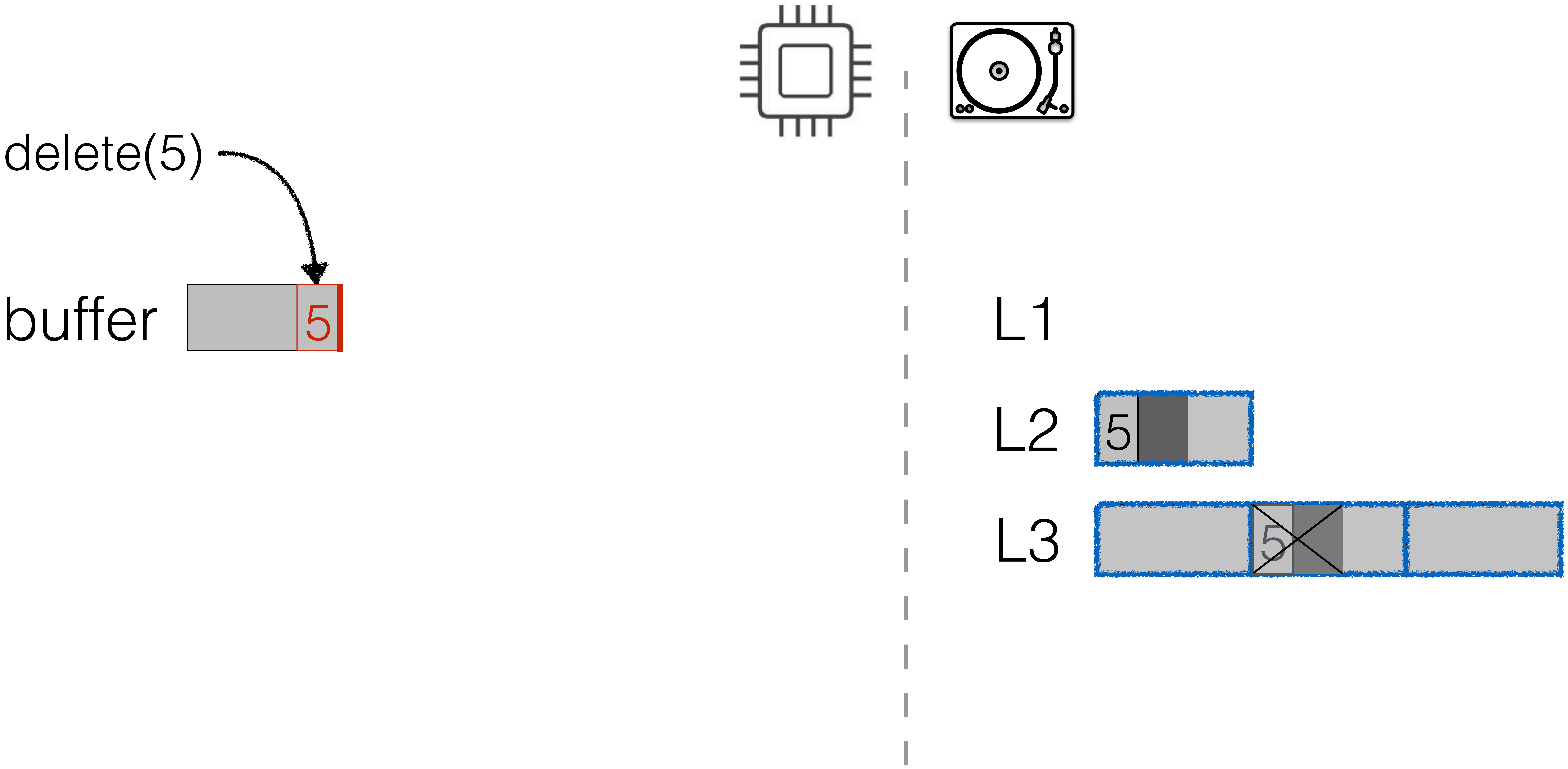
RID | TS flag



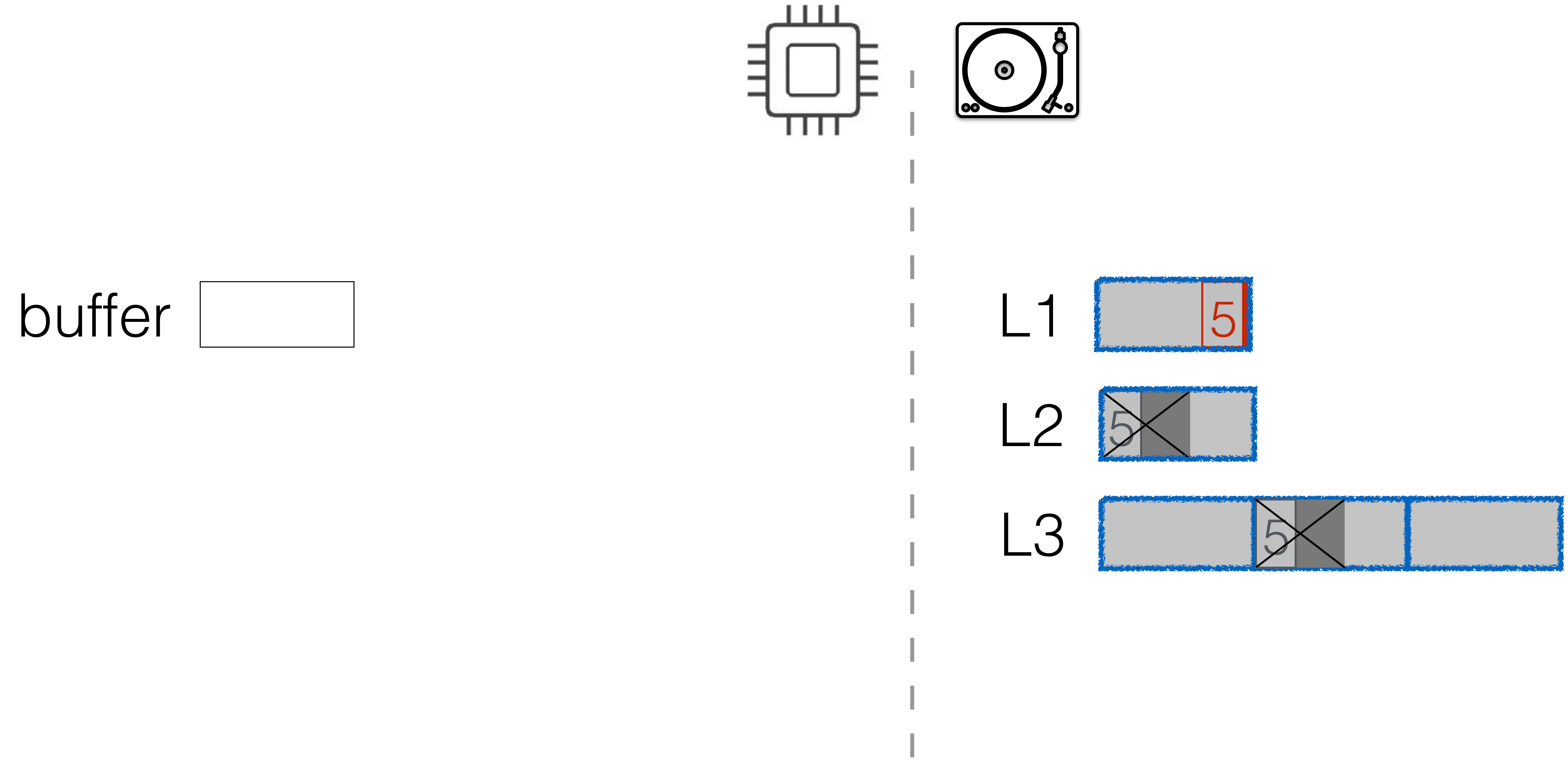
key value

RID | TS flag | timestamp | name | department | • • • | location

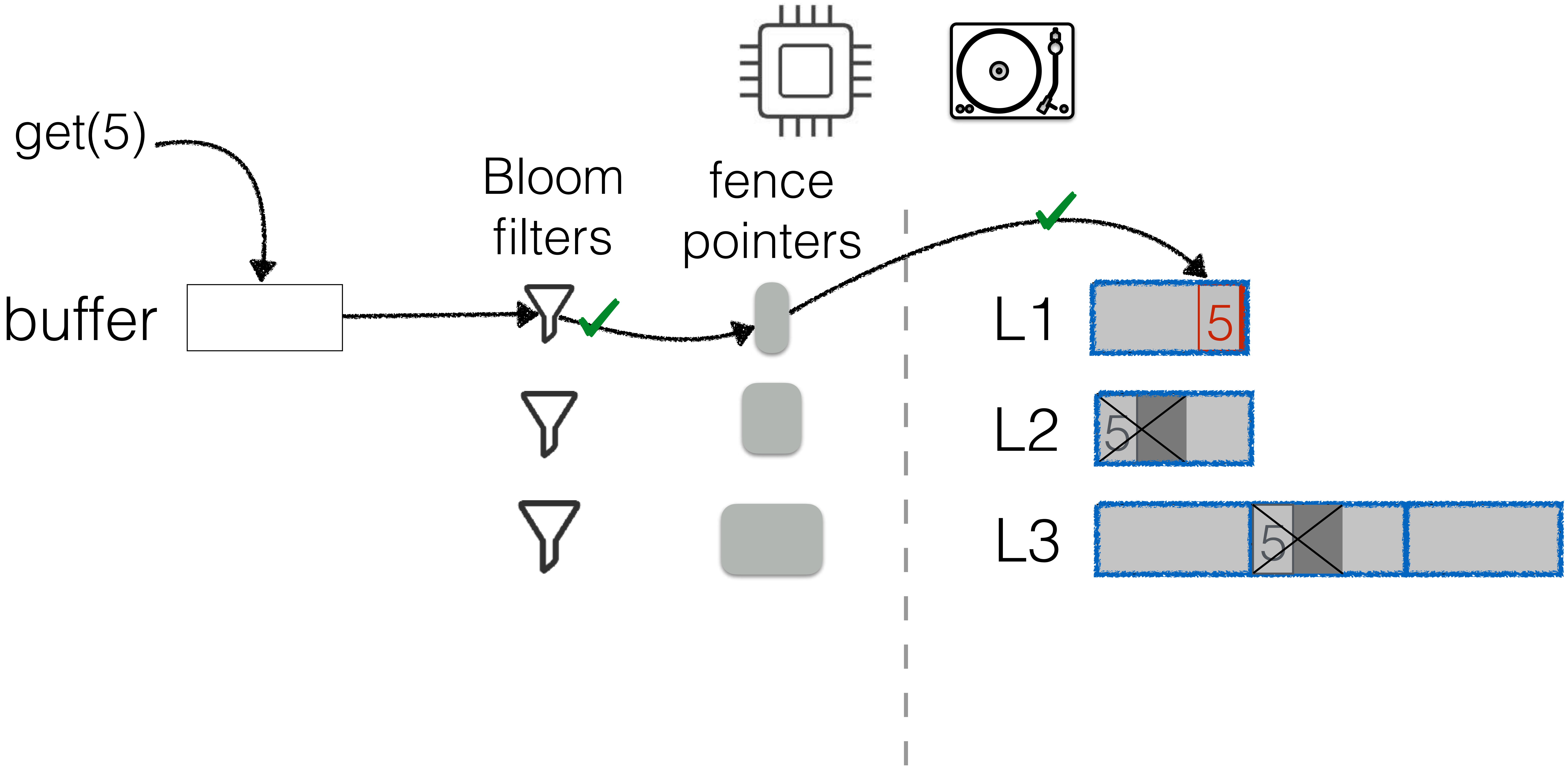
deletes in LSM-tree



deletes in LSM-tree



deletes in LSM-tree

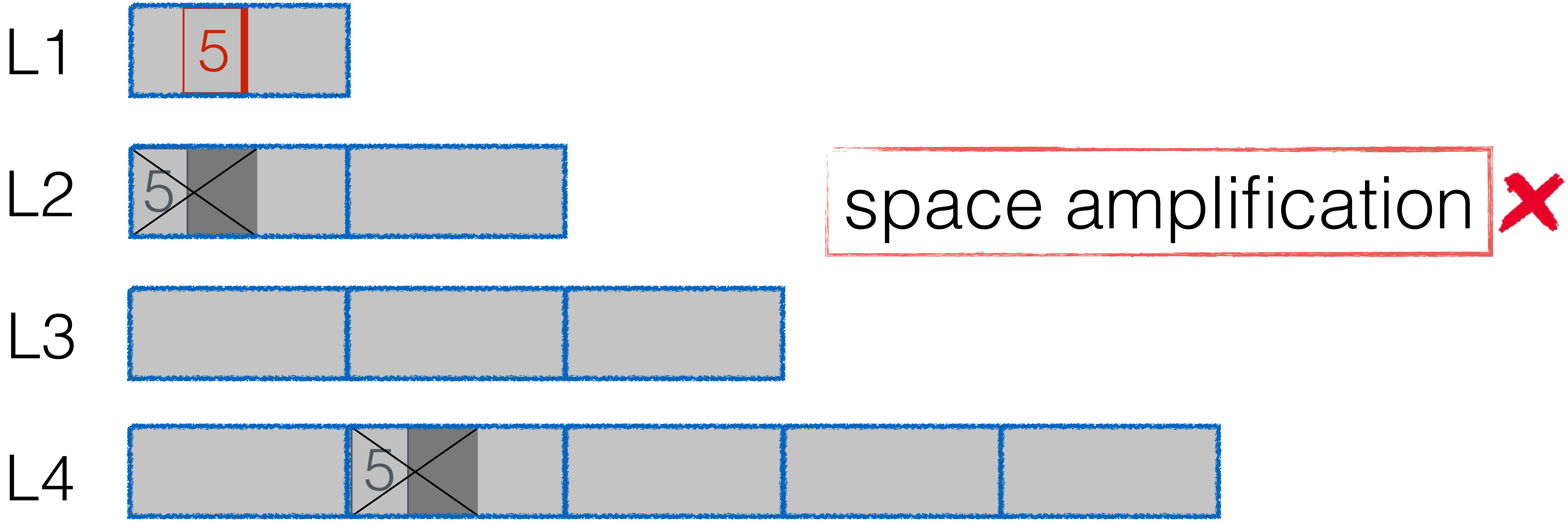


the problems

the problems

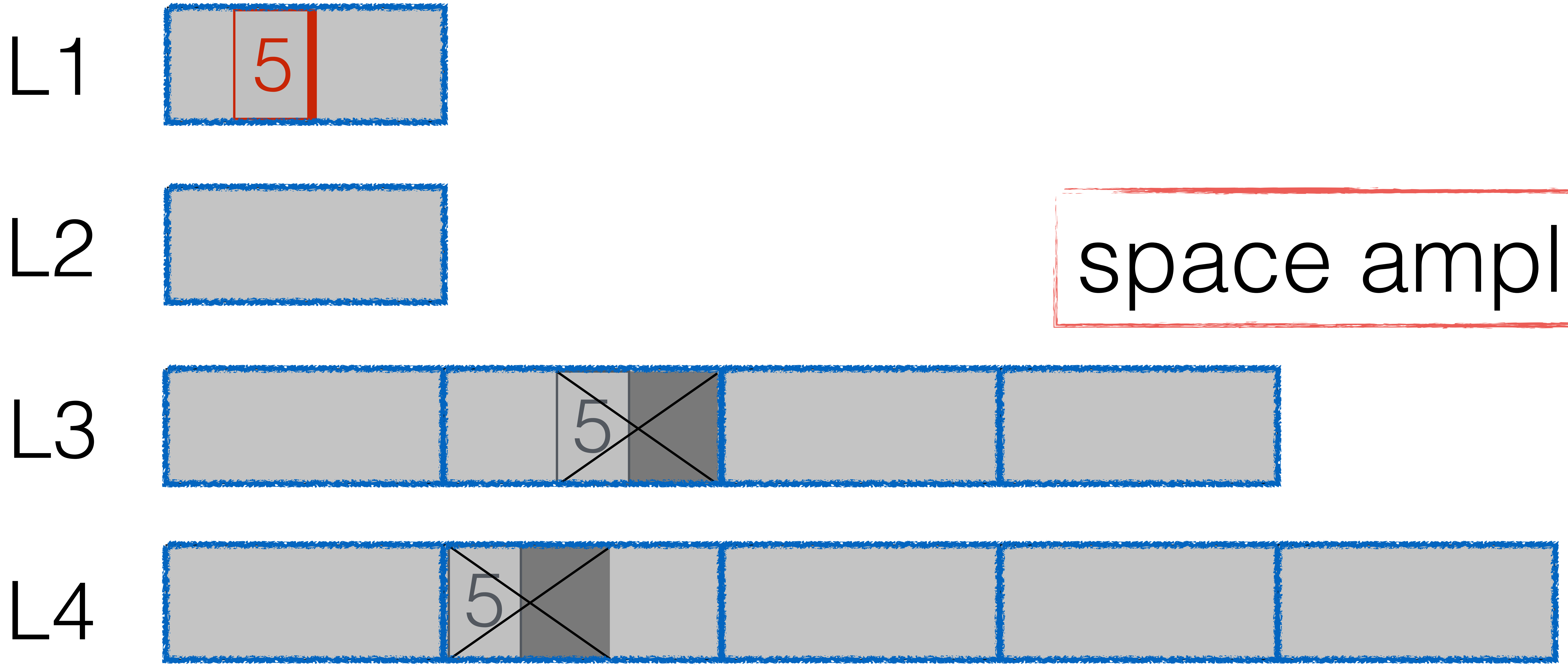
out-of-place deletes

out-of-place deletes



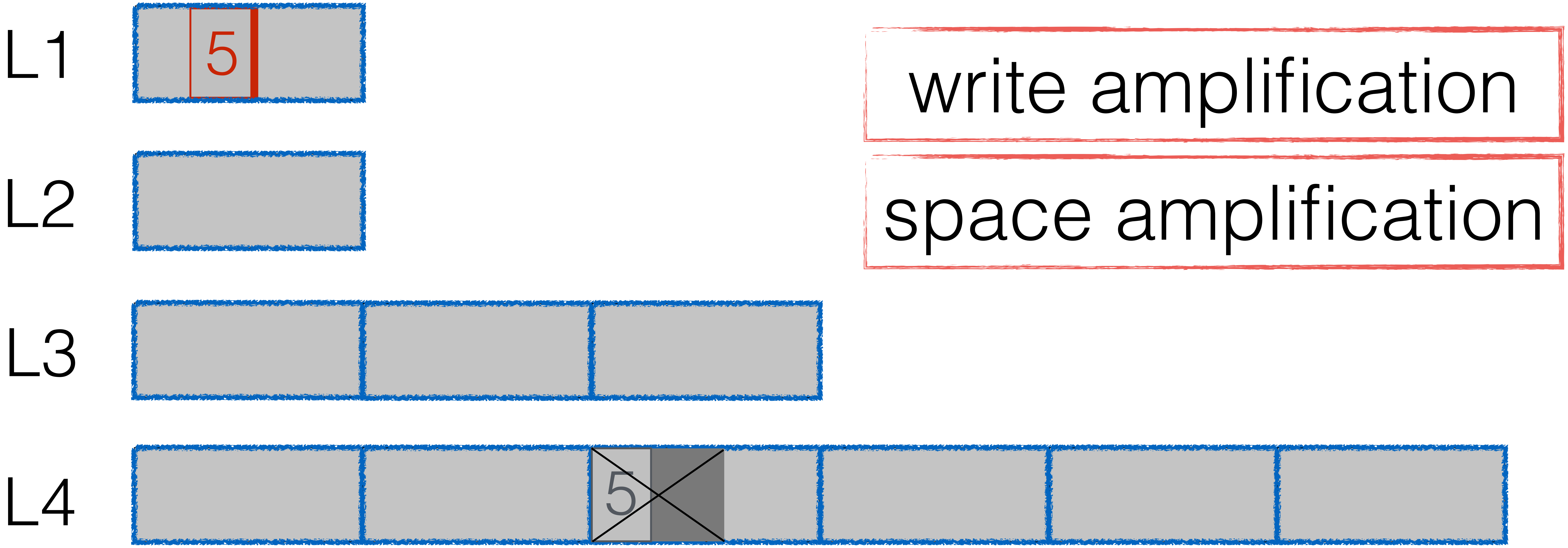
space amplification ✗

out-of-place deletes



space amplification ✗

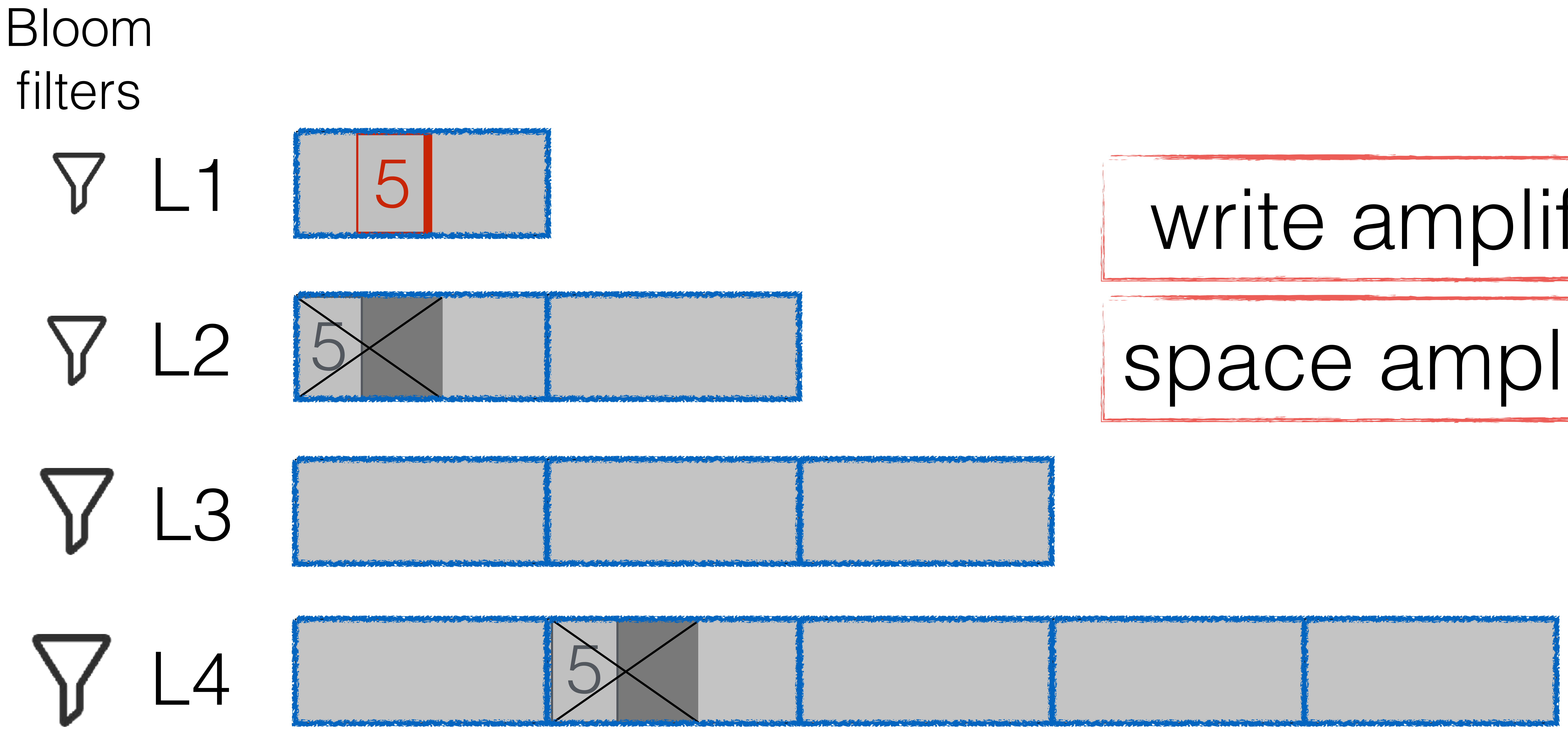
out-of-place deletes



write amplification ✗

space amplification ✗

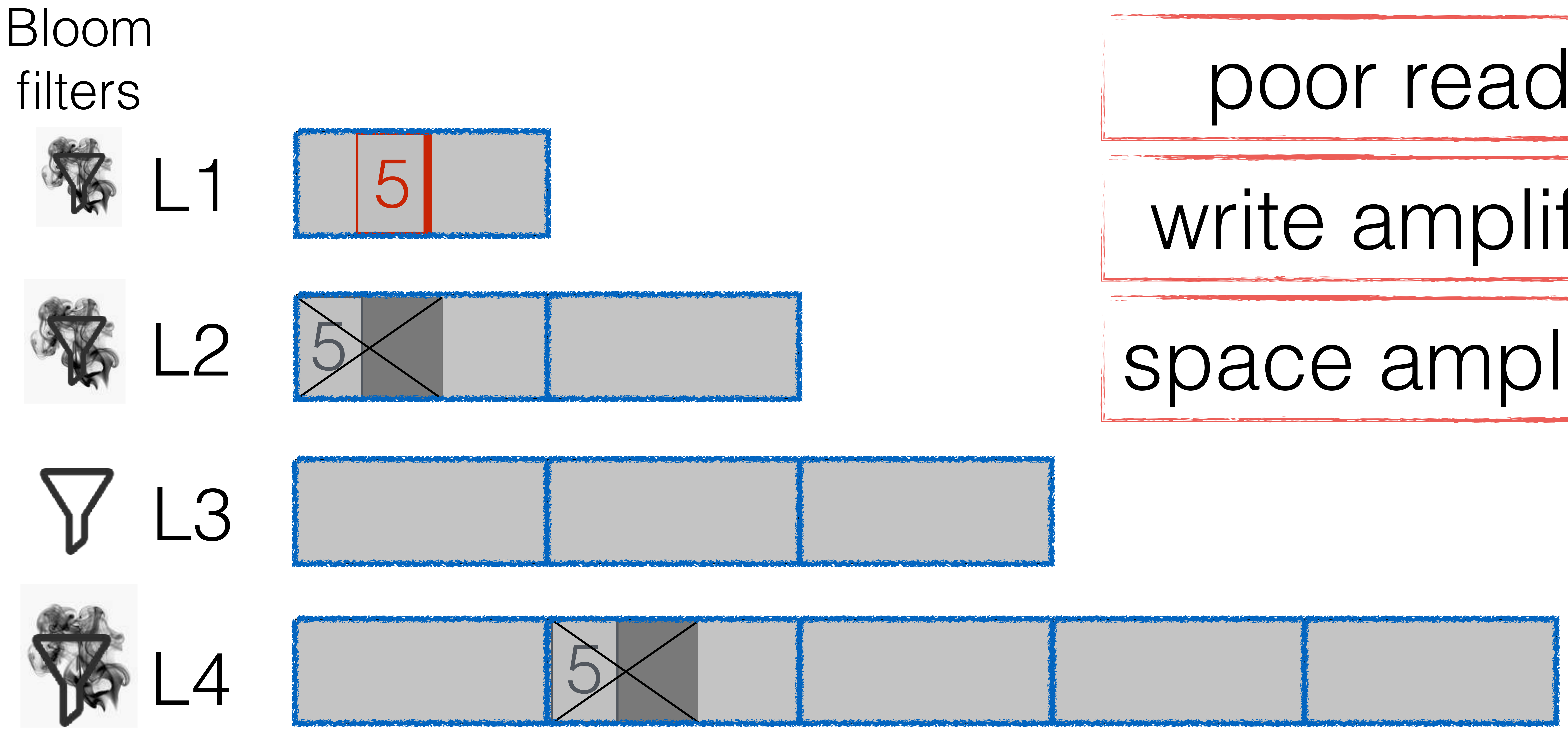
out-of-place deletes



write amplification ✗

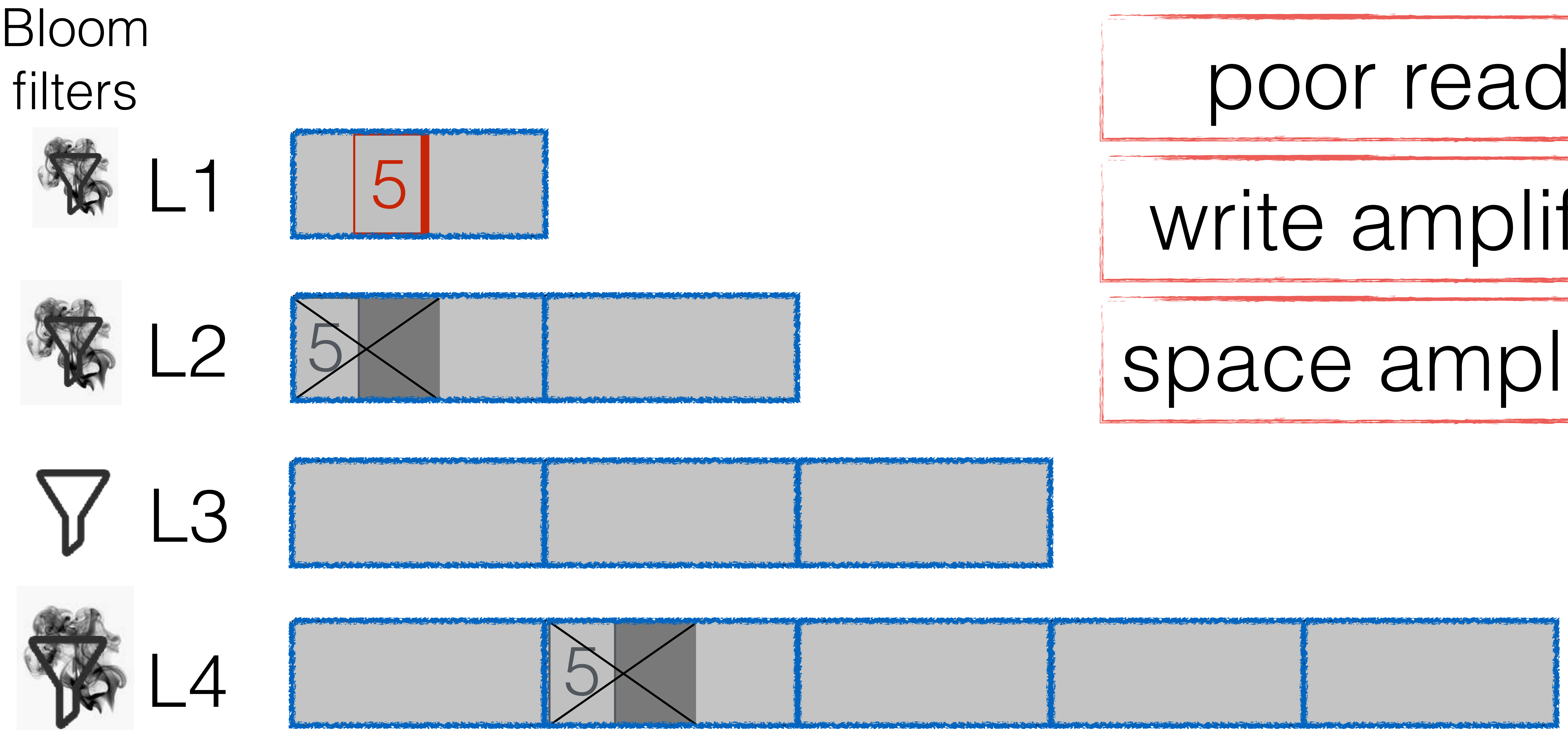
space amplification ✗

out-of-place deletes



- poor read perf. ✗
- write amplification ✗
- space amplification ✗

out-of-place deletes



poor read perf. ✗

write amplification ✗

space amplification ✗

the problems

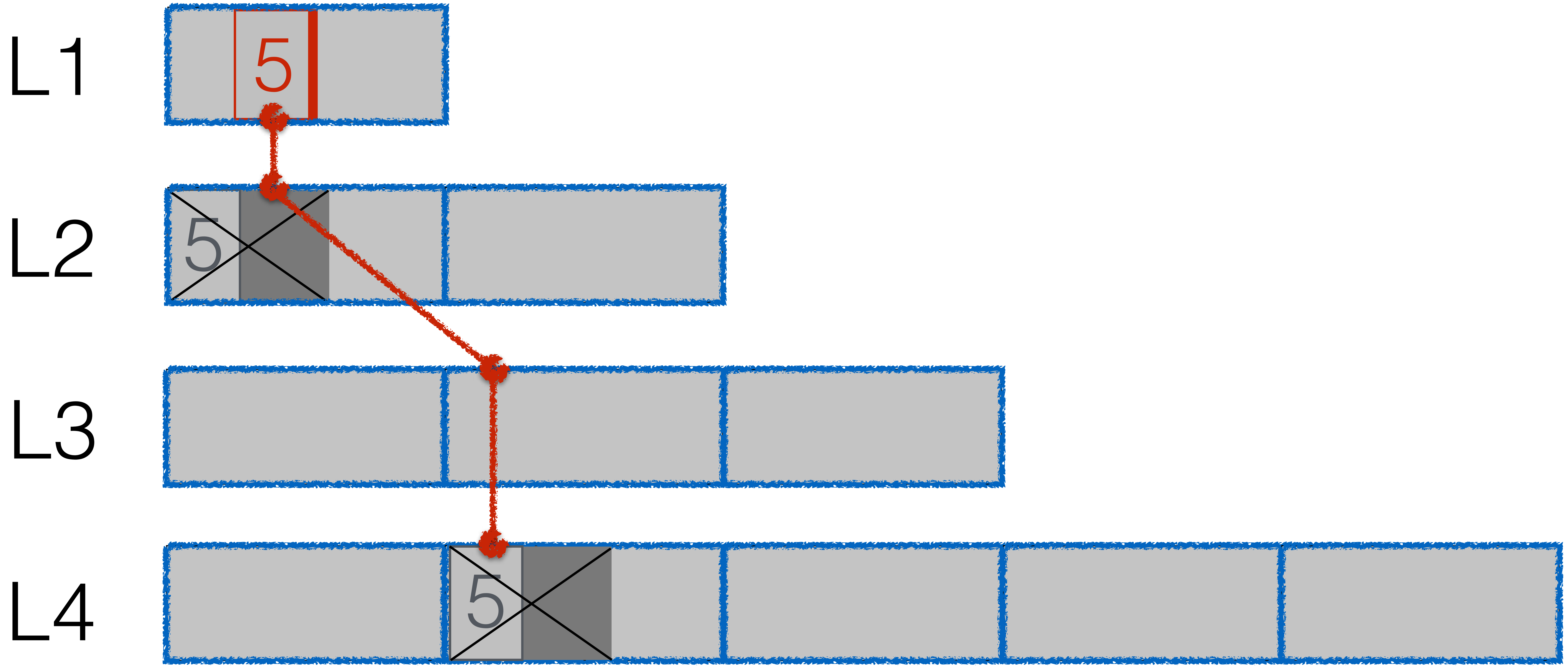
poor read perf.

write amplification

space amplification

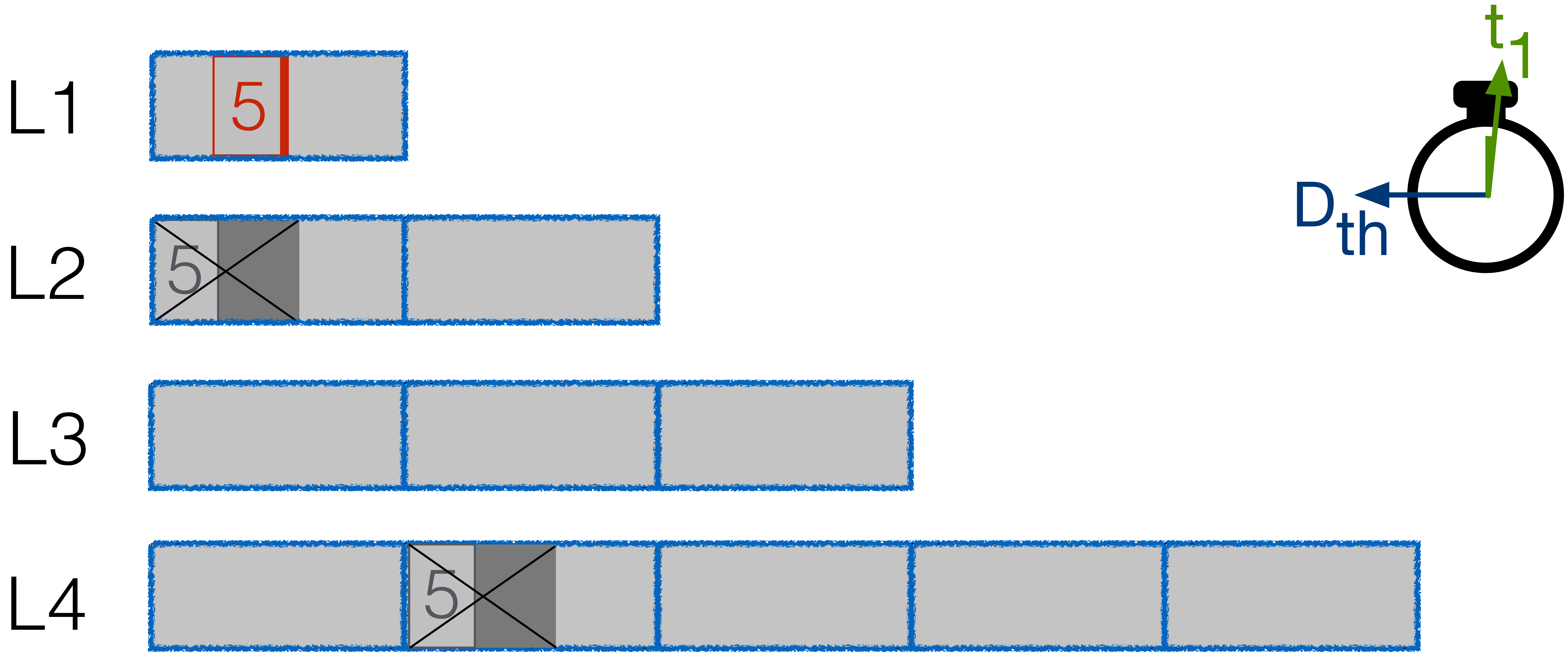
delete persistence latency

delete persistence latency



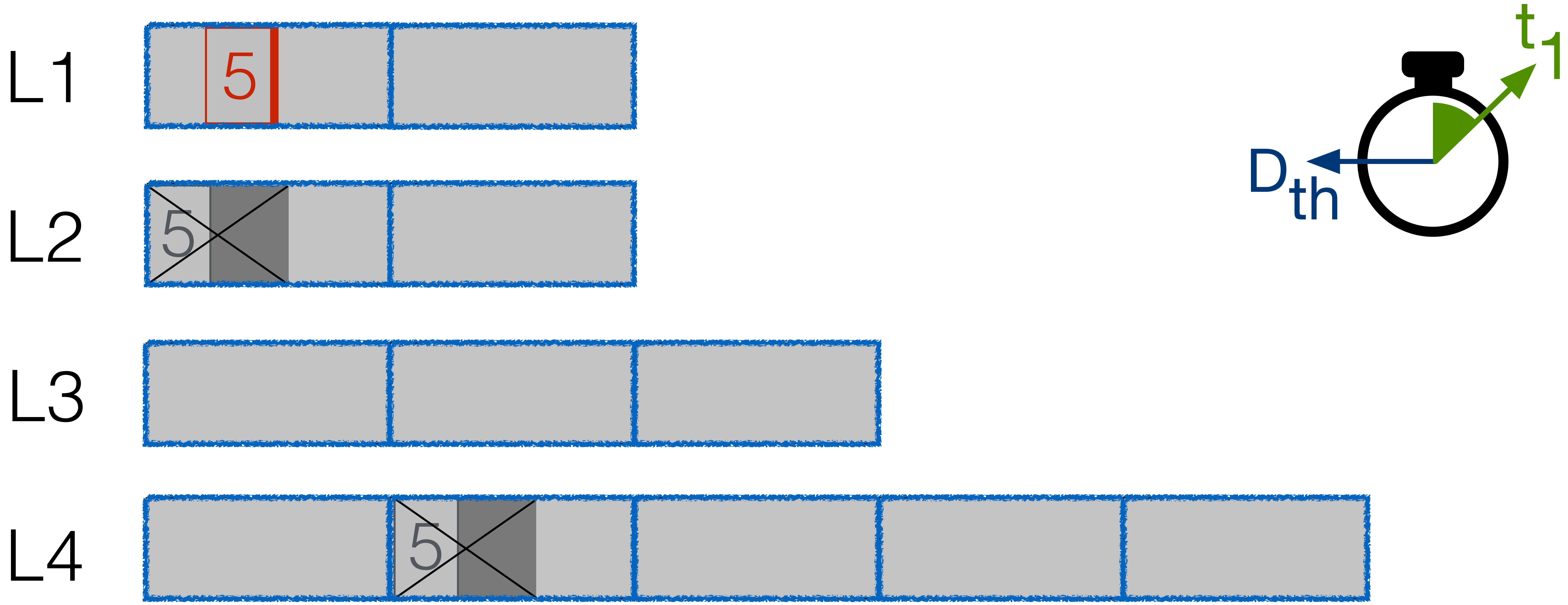
delete persistence latency

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



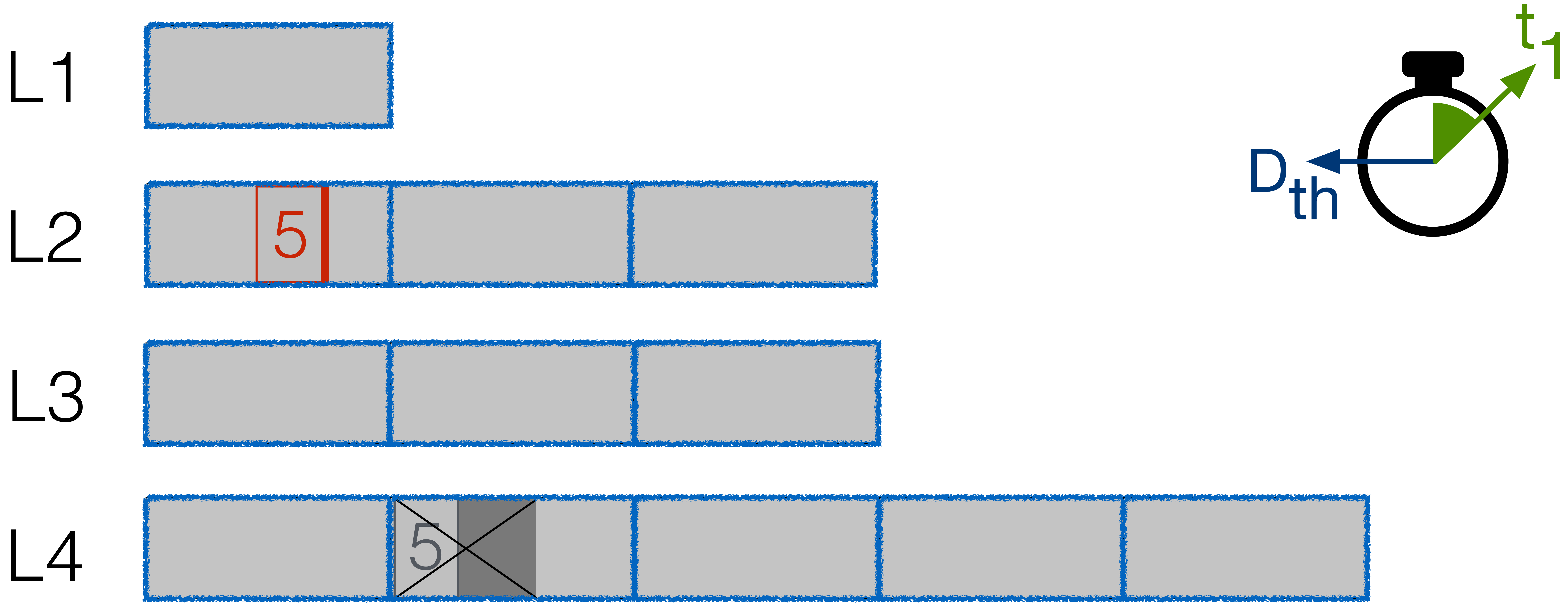
delete persistence latency

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



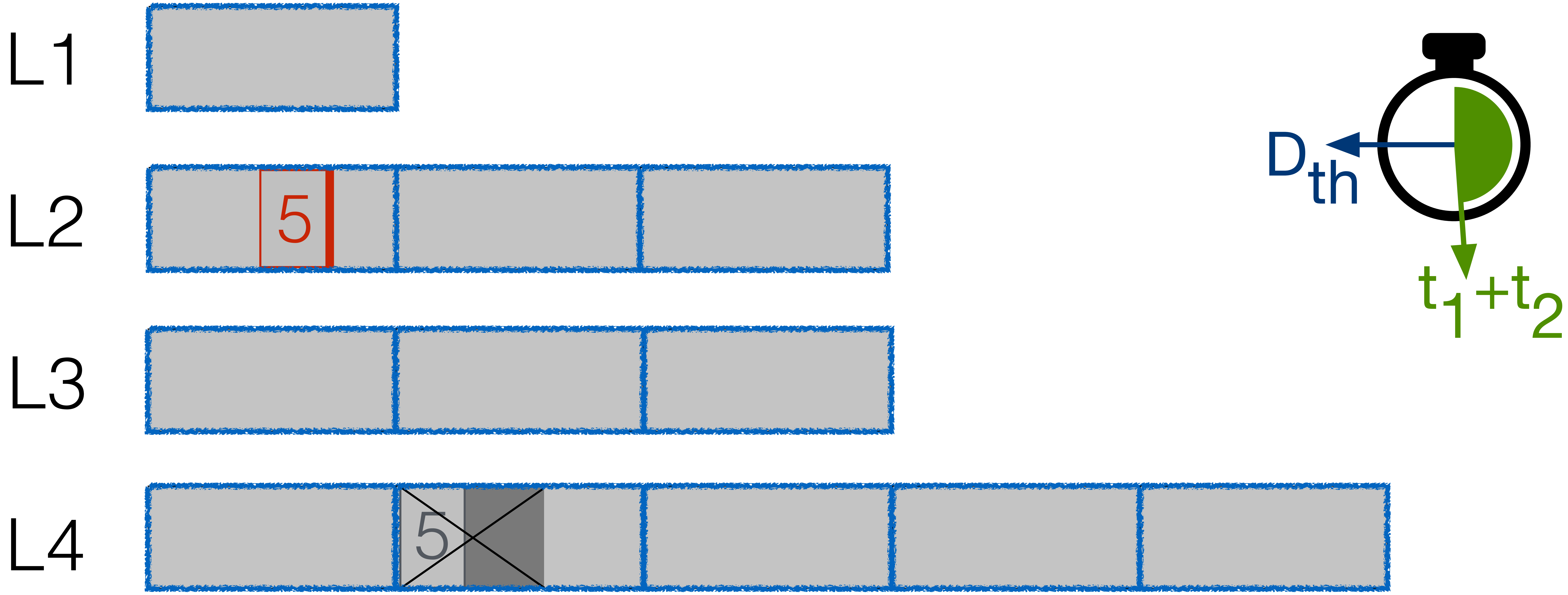
delete persistence latency

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



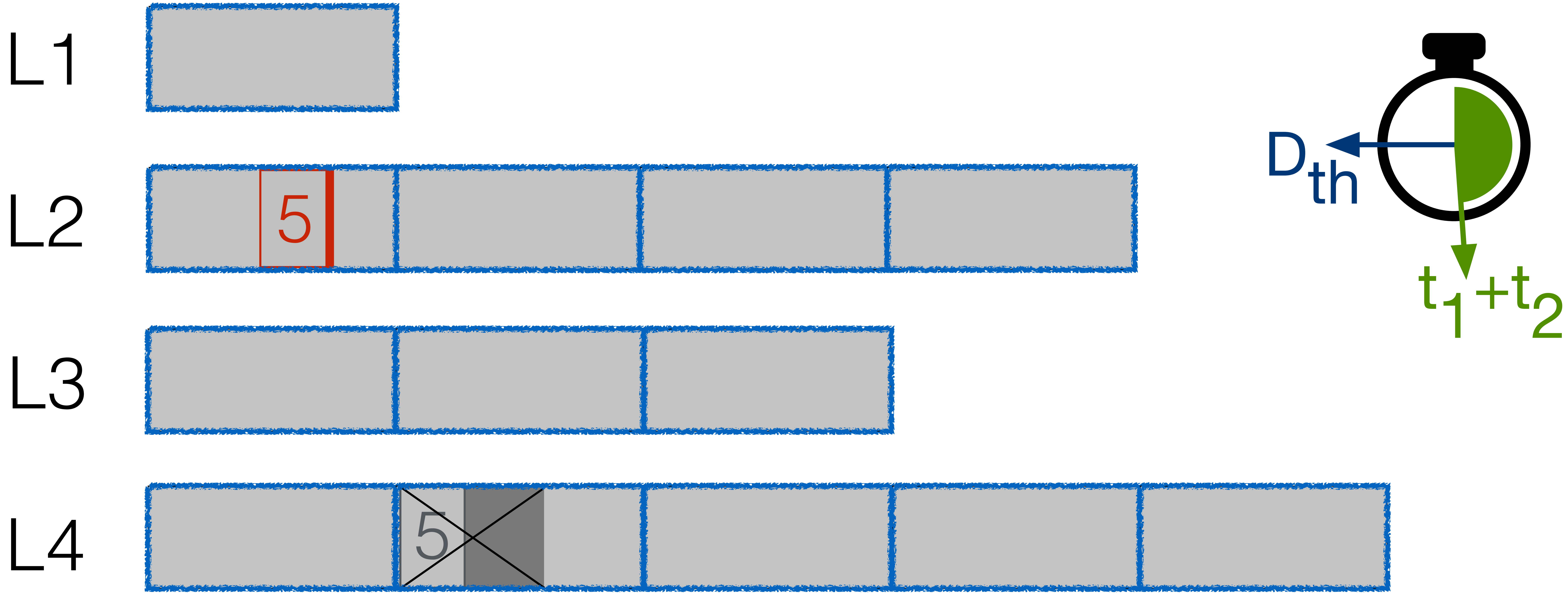
delete persistence latency

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



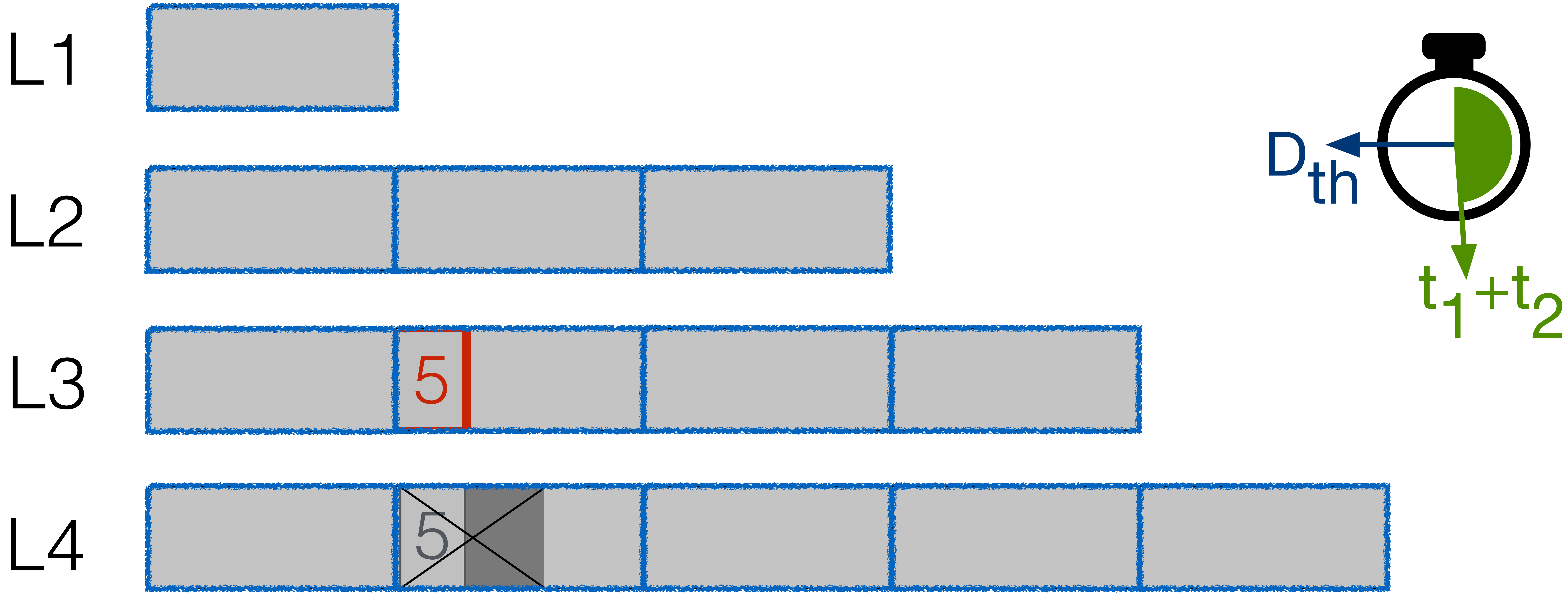
delete persistence latency

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



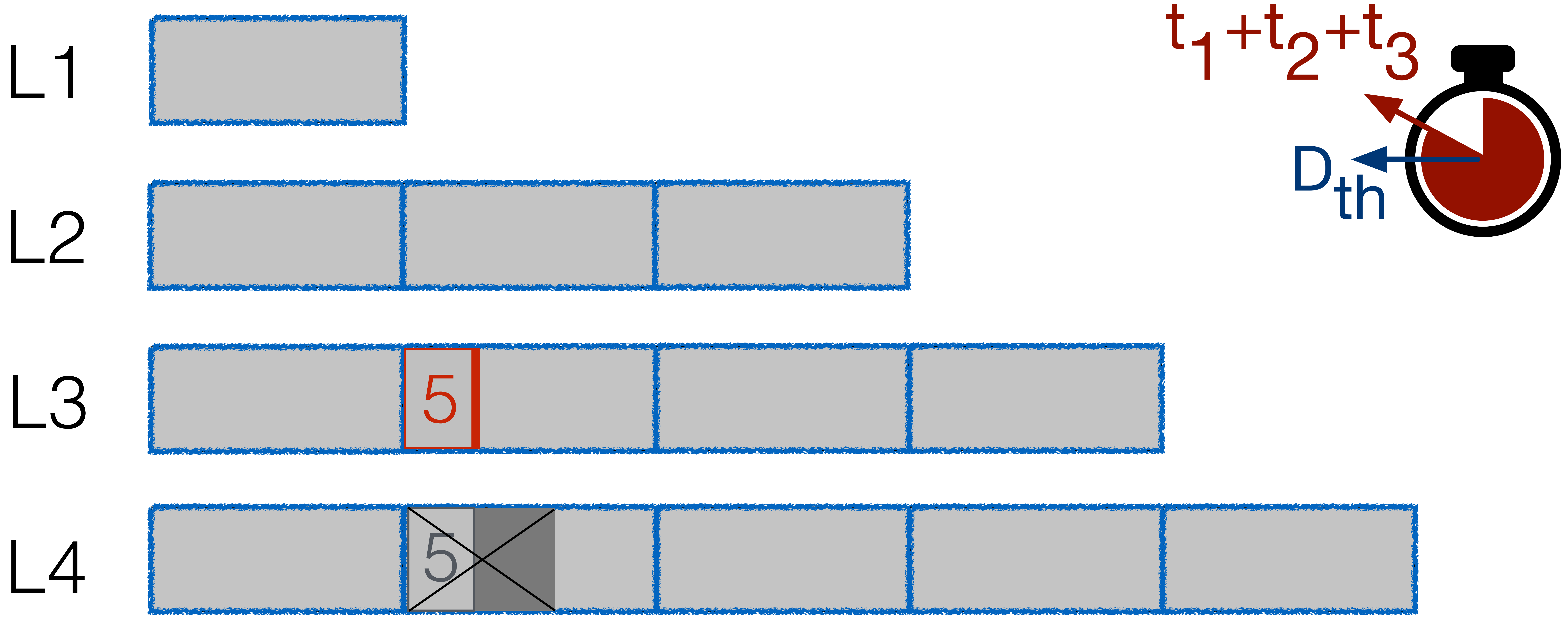
delete persistence latency

delete(5) within a threshold time: **D**$_{\textbf{th}}$



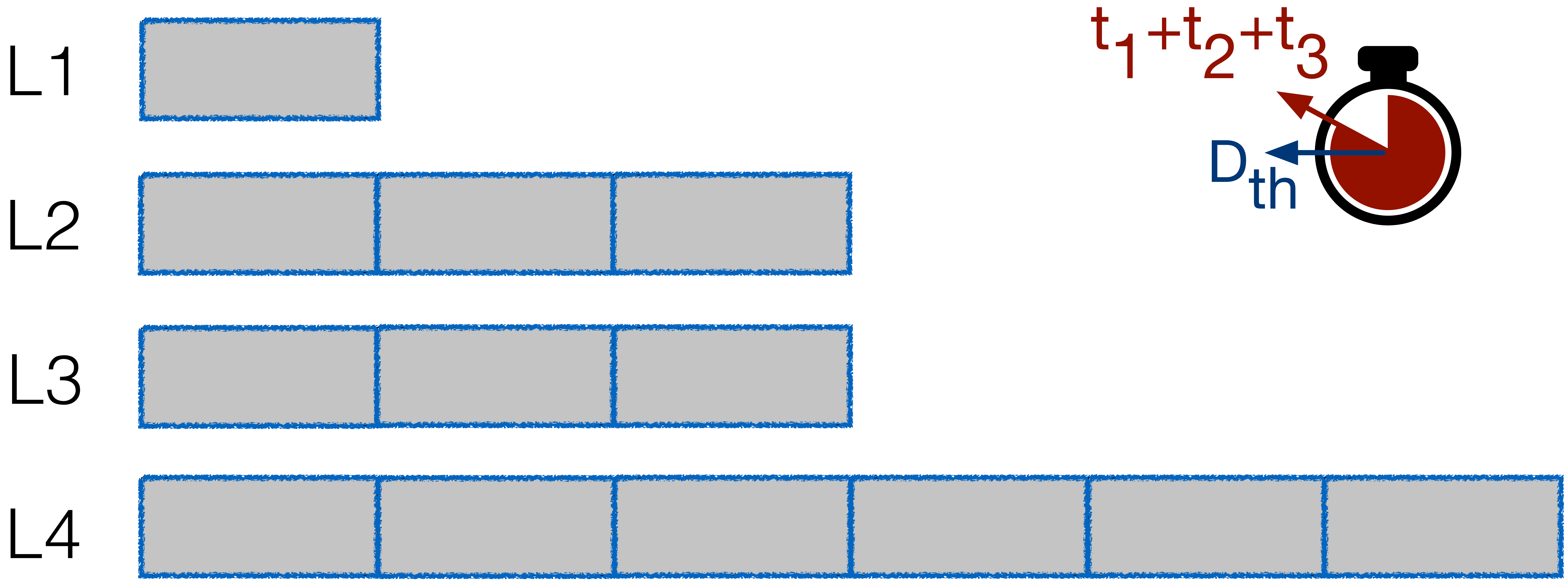
delete persistence latency

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



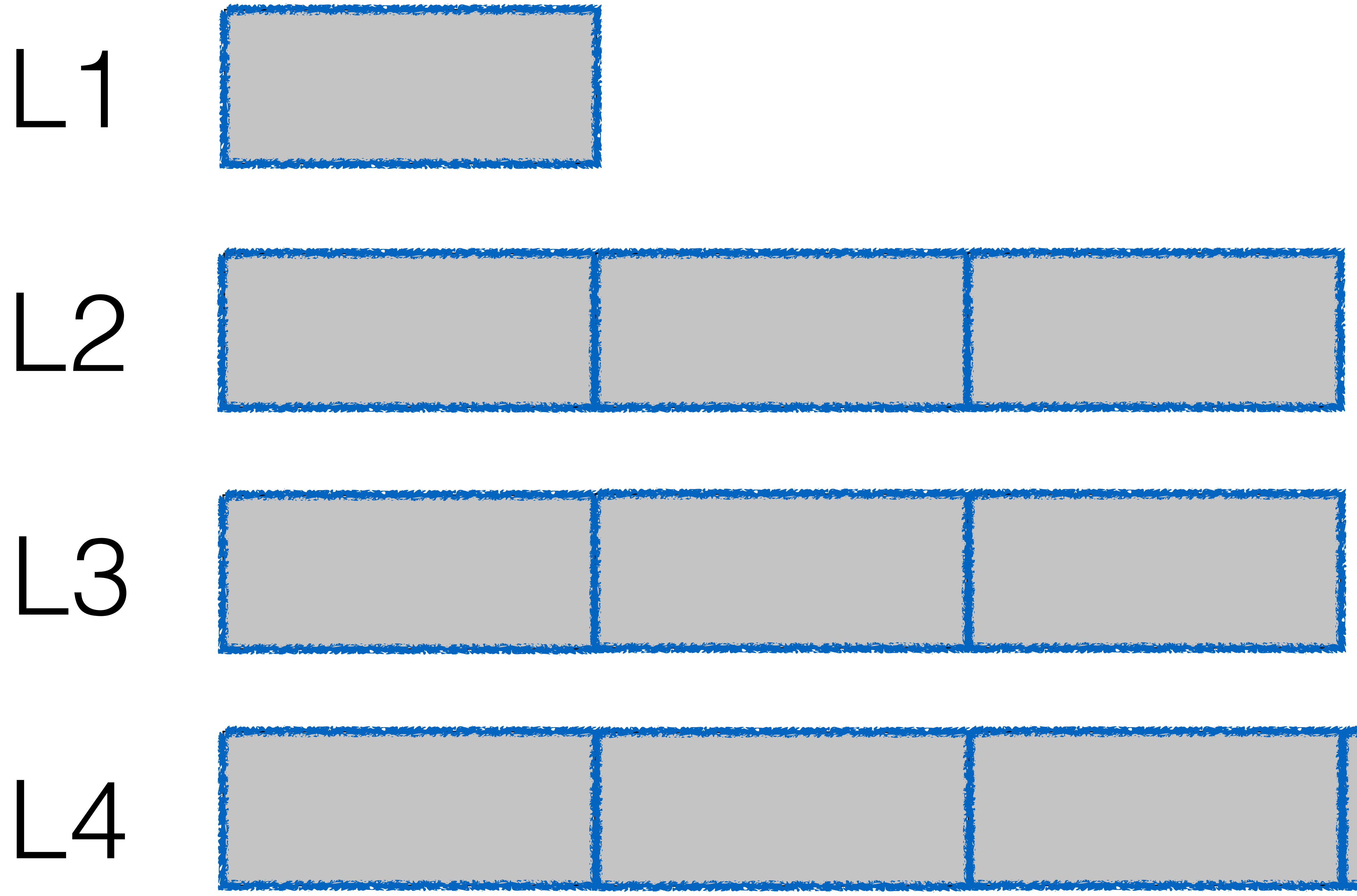
delete persistence latency

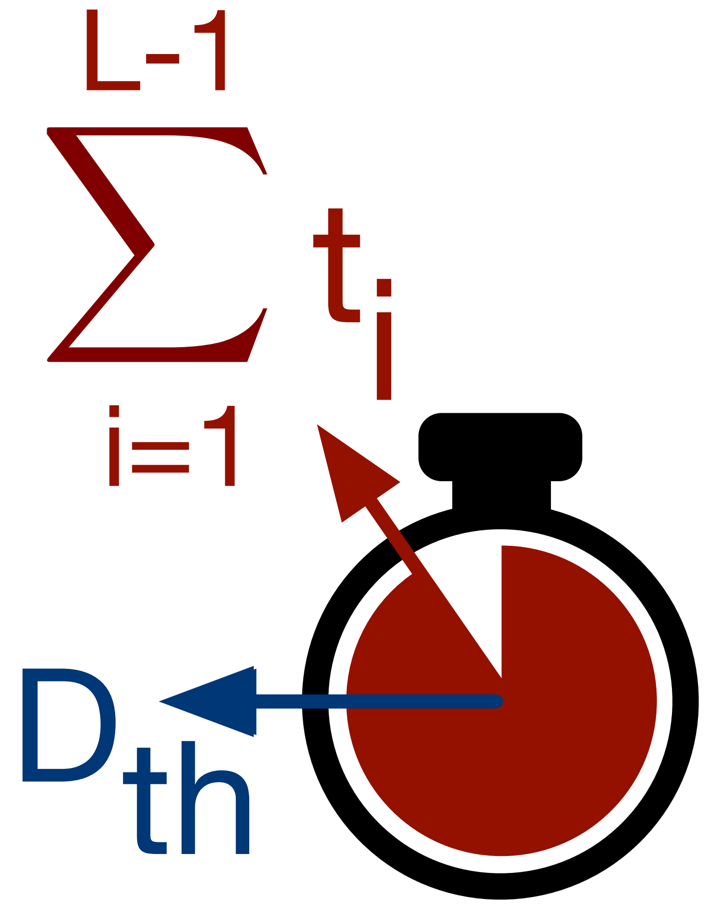
delete(5) within a threshold time: **D_{th}**



delete persistence latency

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





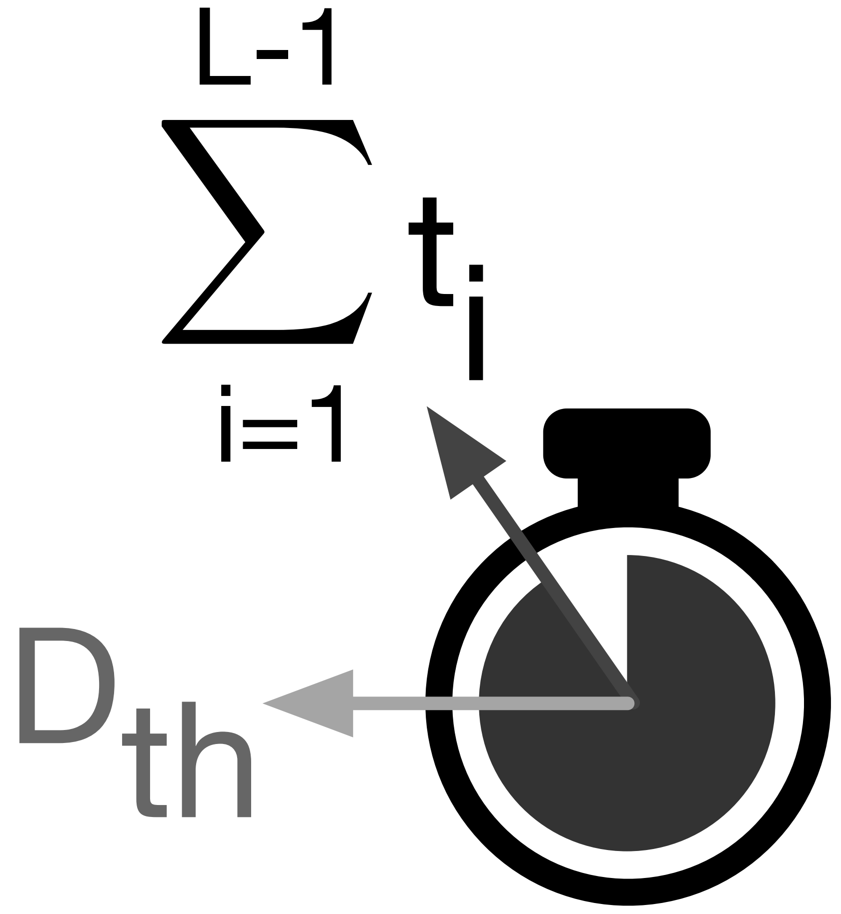
unbounded delete persistence latency ✗

the problems

poor read perf.

write amplification

space amplification



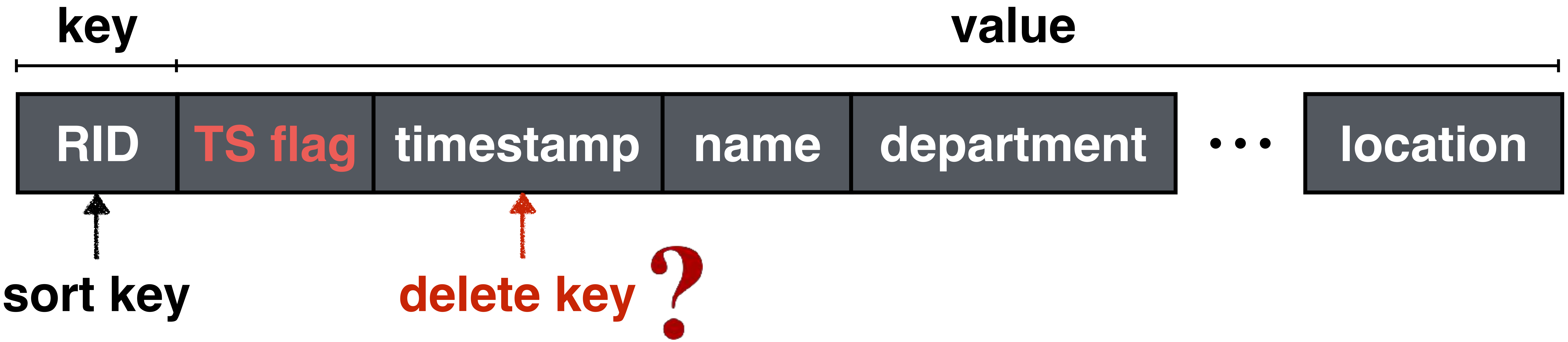
$$\sum_{i=1}^{L-1} t_i$$

unbounded delete persistence latency

deletes on a secondary attribute

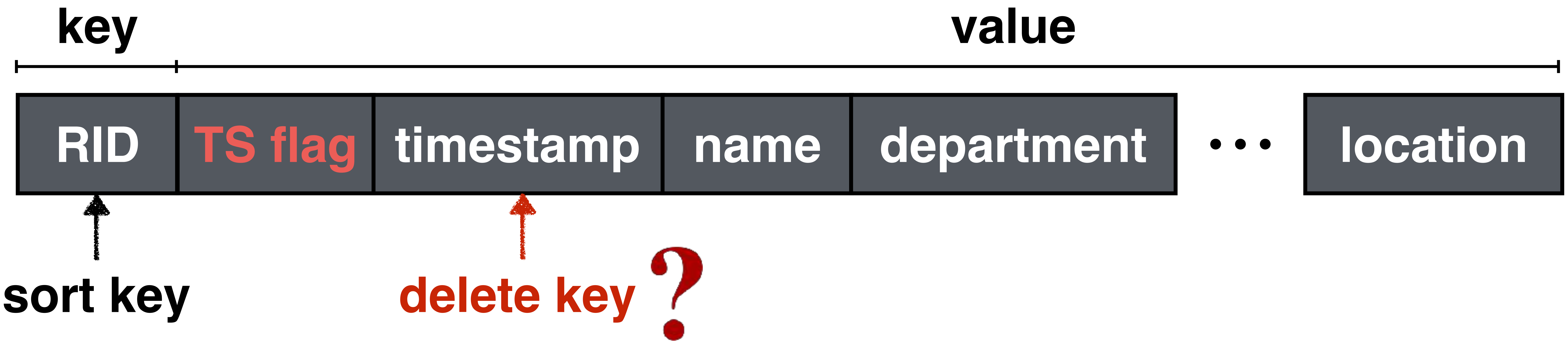
deletes on a secondary attribute

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



deletes on a secondary attribute

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



key

value

RID

TS flag

timestamp

name

department

• • •

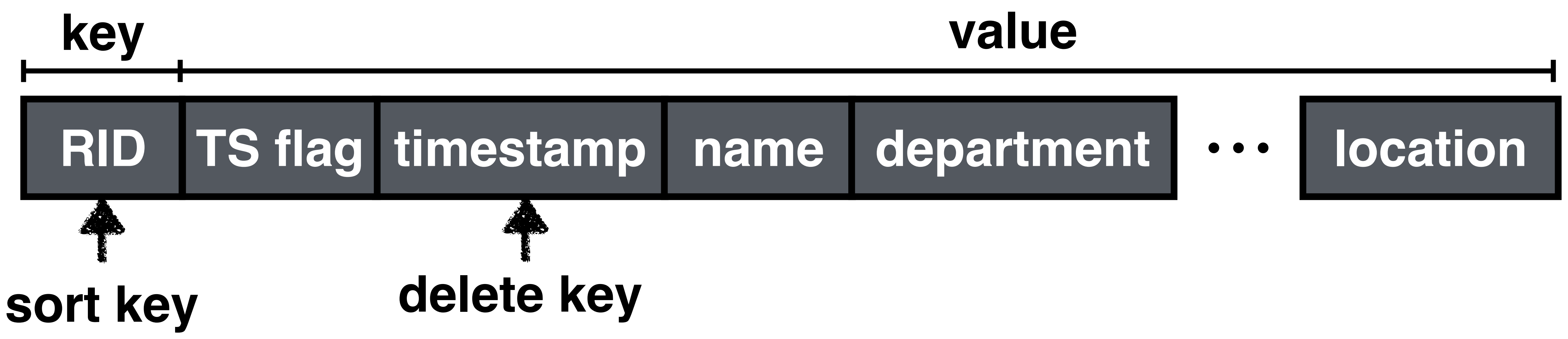
location

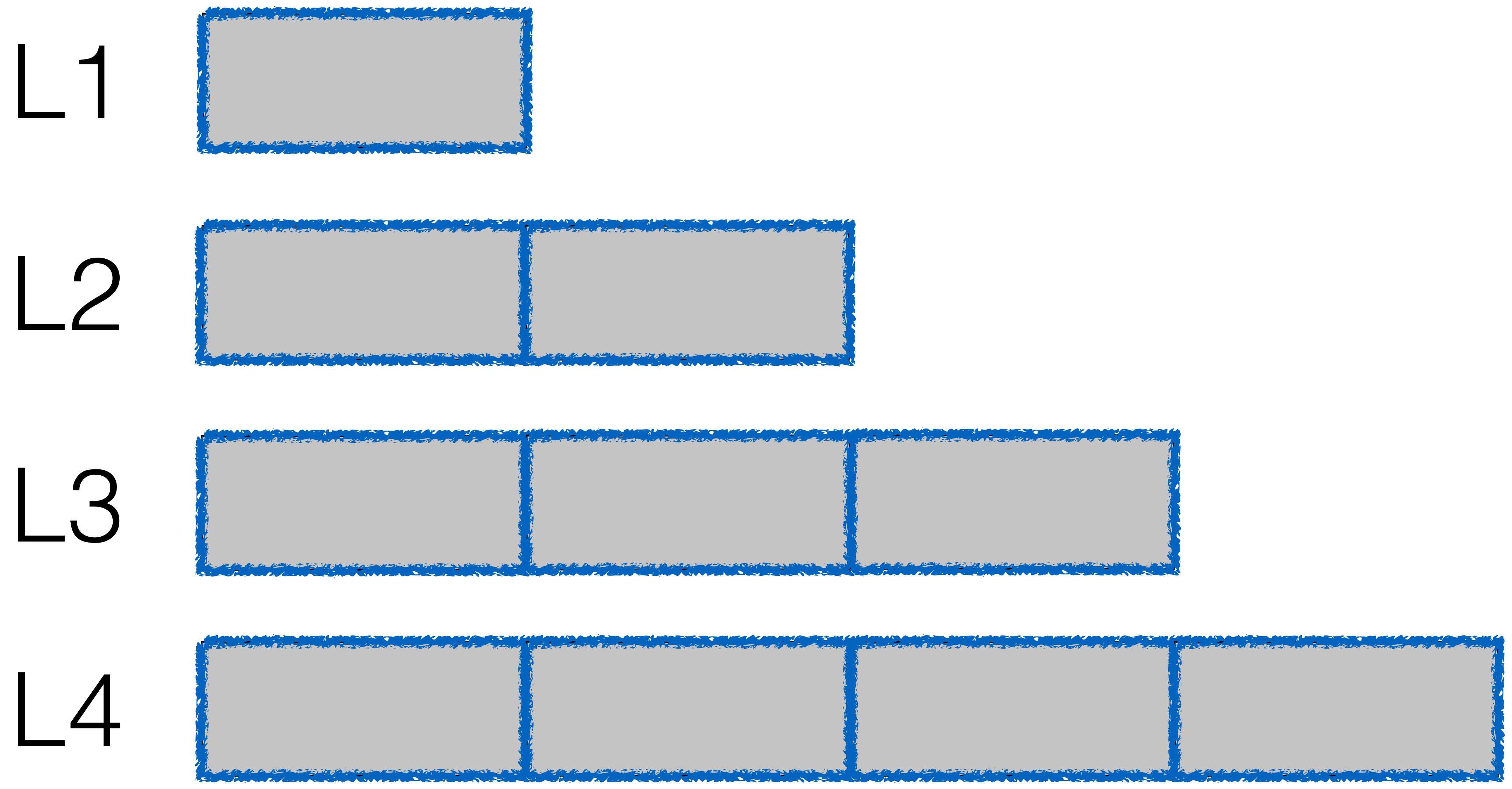
sort key

delete key ?

deletes on a secondary attribute

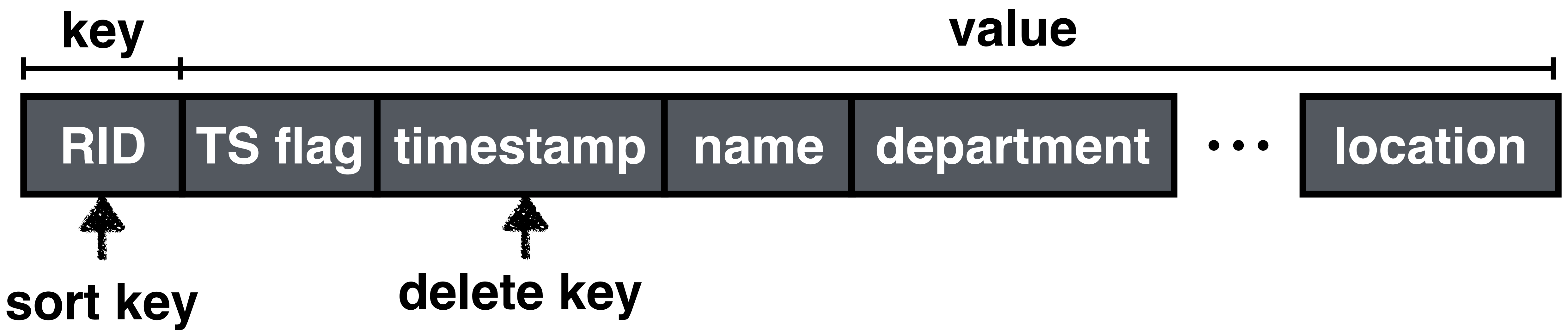
delete all entries older than: **D days**

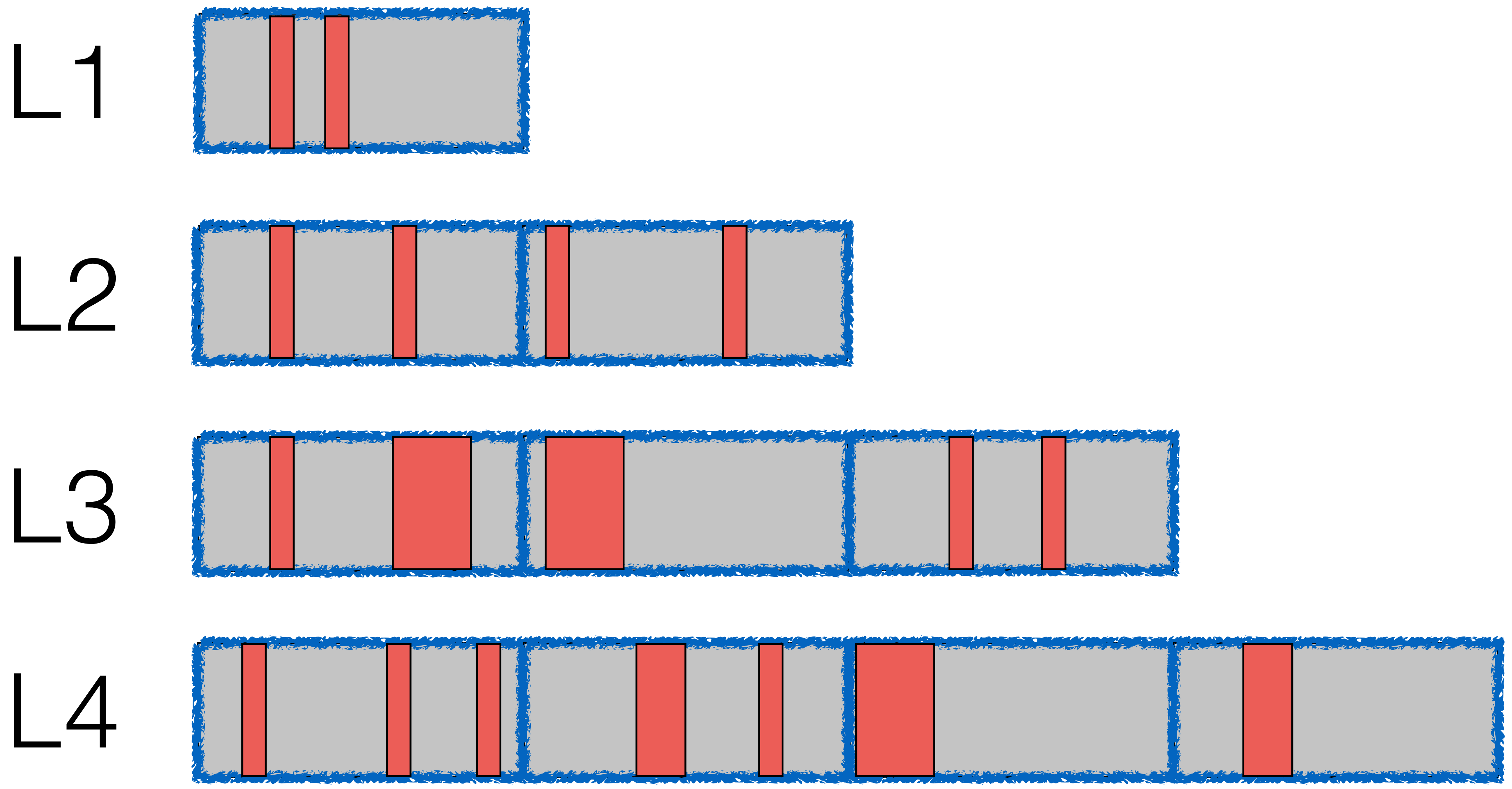




deletes on a secondary attribute

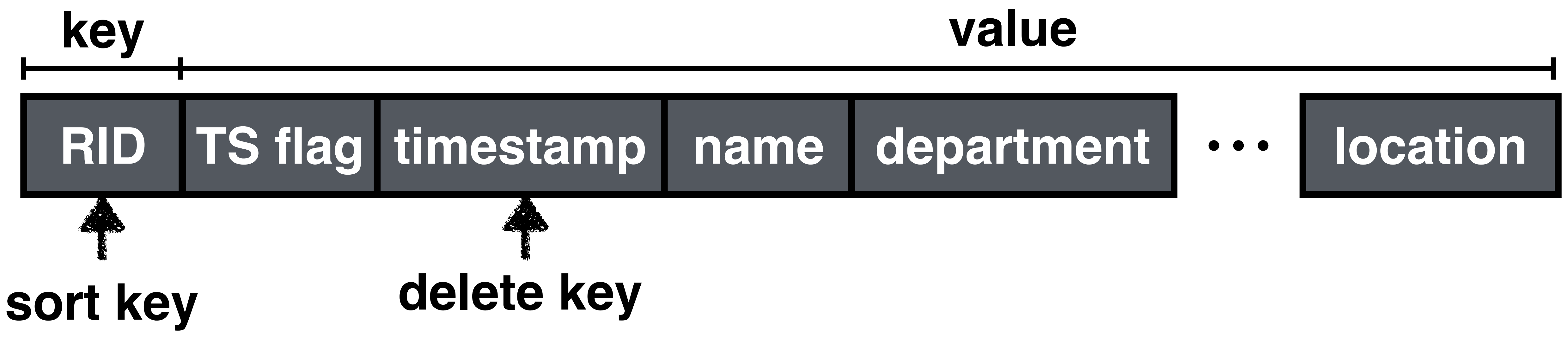
delete all entries older than: **D days**





deletes on a secondary attribute

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


key

value

RID

TS flag

timestamp

name

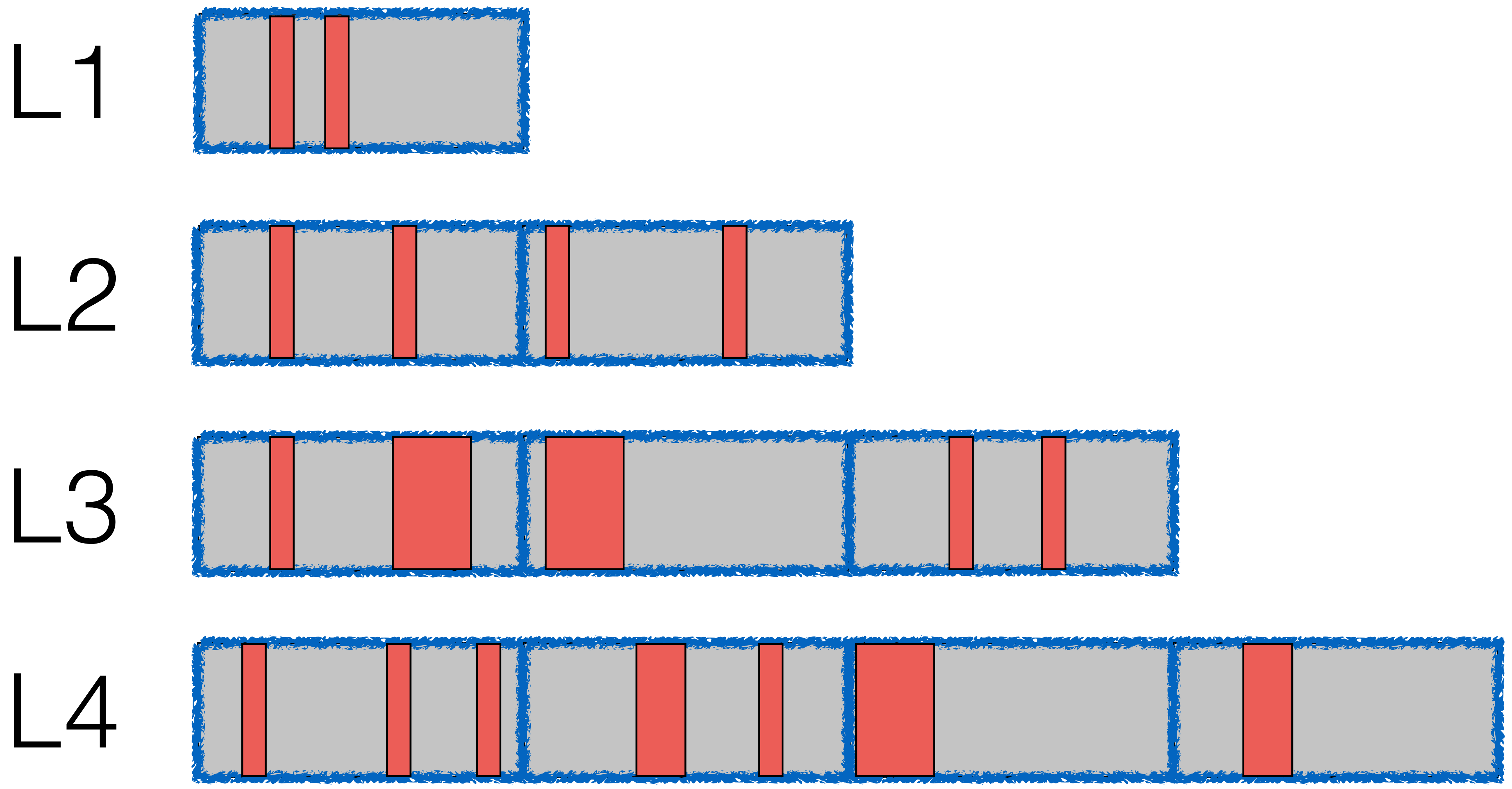
department

···

location

sort key

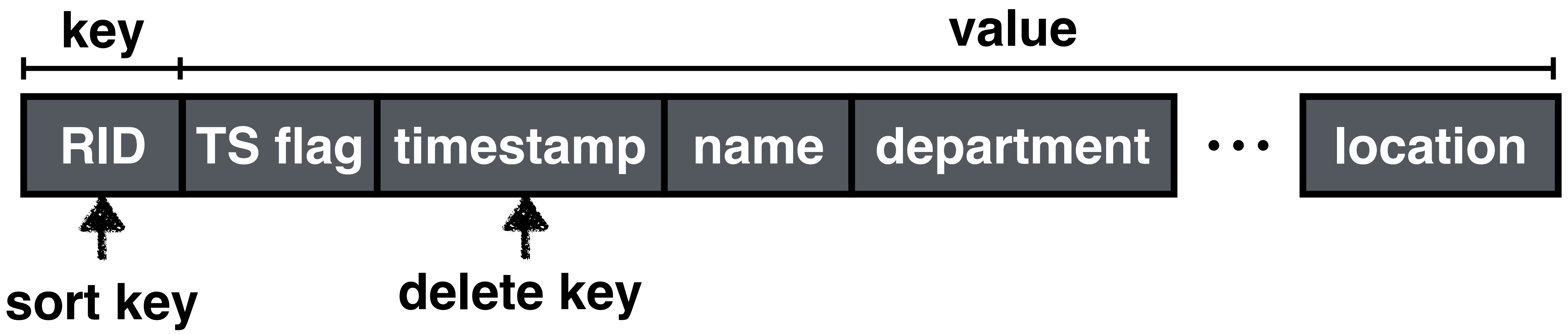
delete key



scattered occurrences

deletes on a secondary attribute

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



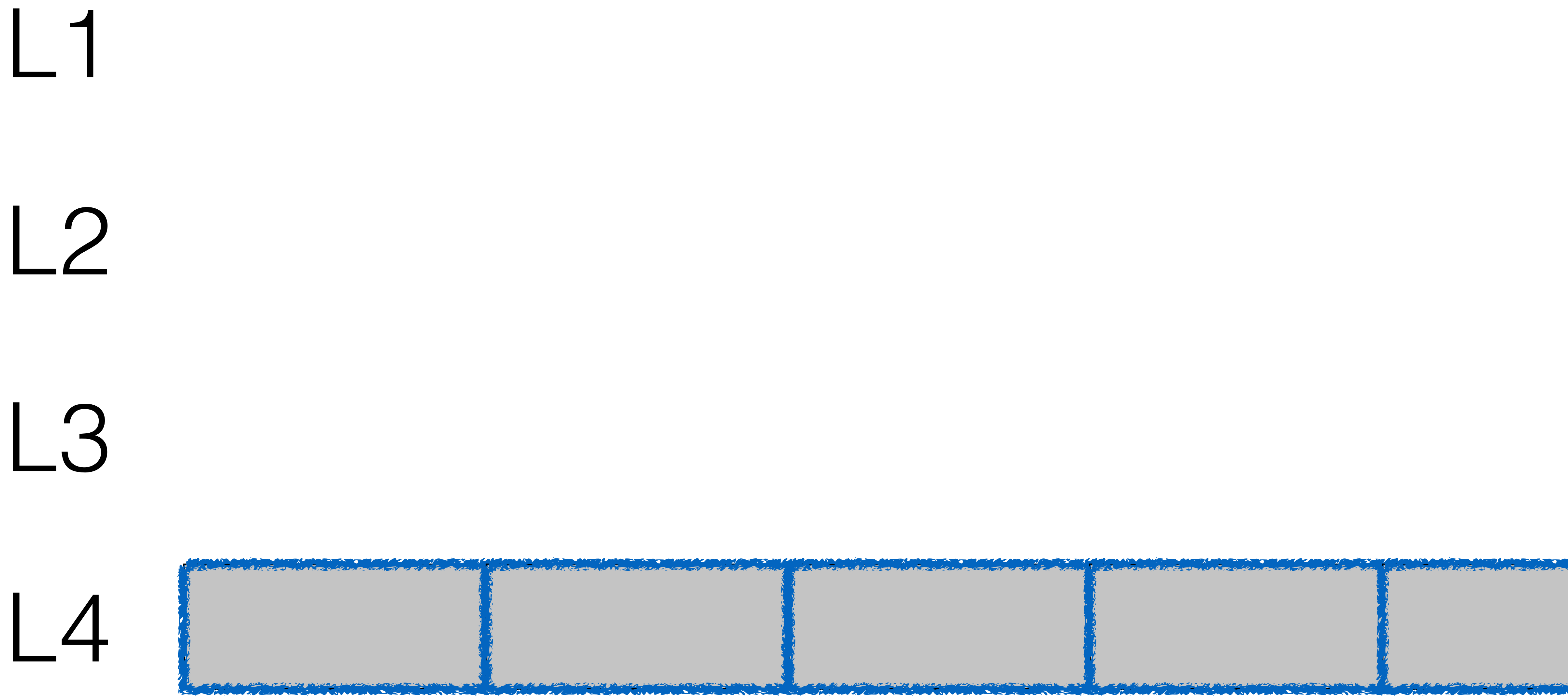
key

value

RID	TS flag	timestamp	name	department	···	location

sort key

delete key



L1

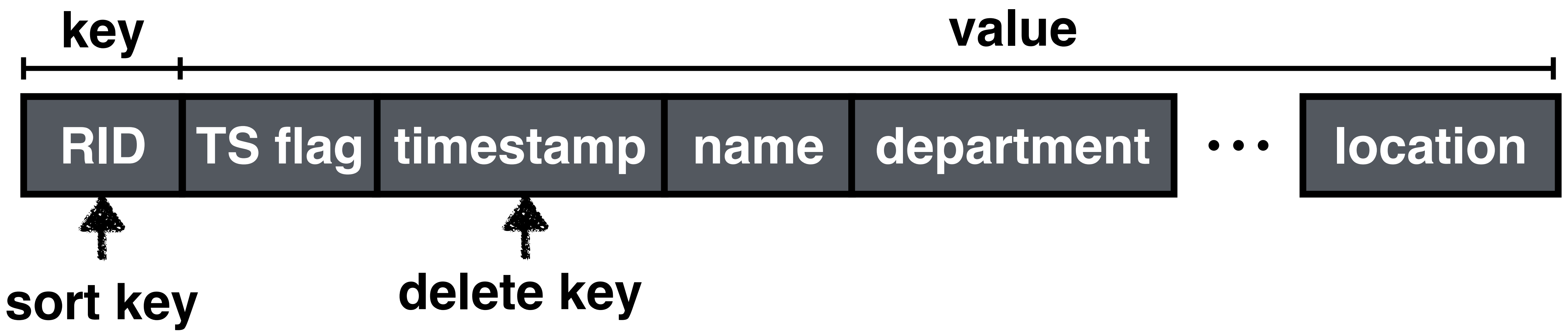
L2

L3

L4

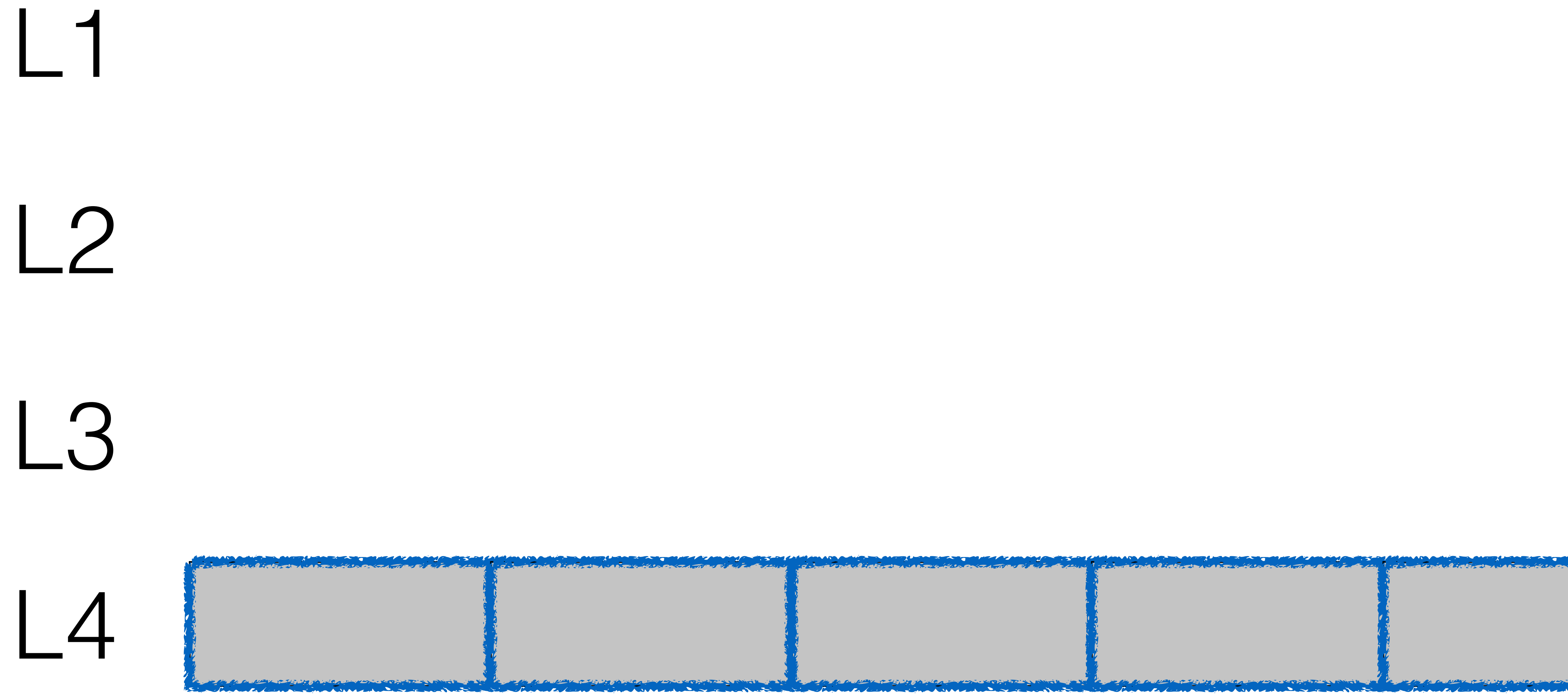
deletes on a secondary attribute

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



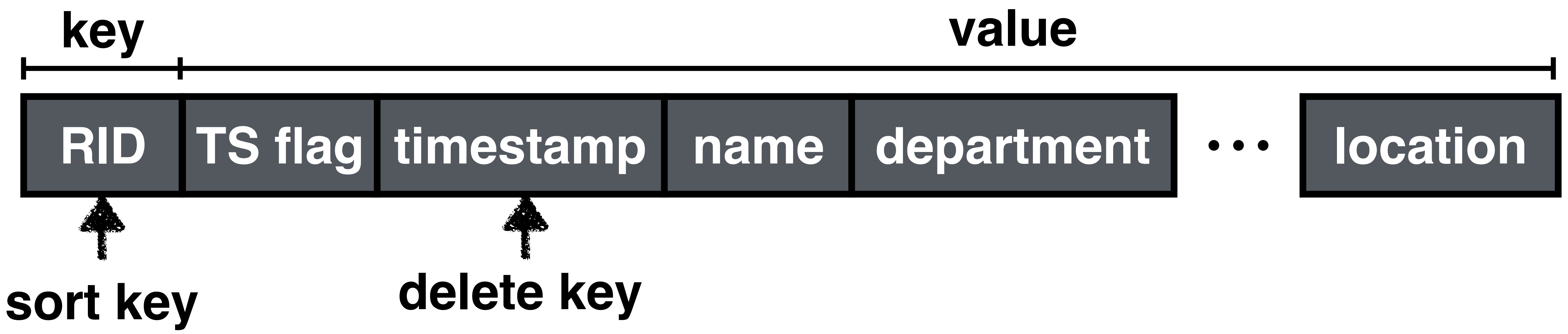
latency spikes ✗

superfluous I/Os ✗



deletes on a secondary attribute

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

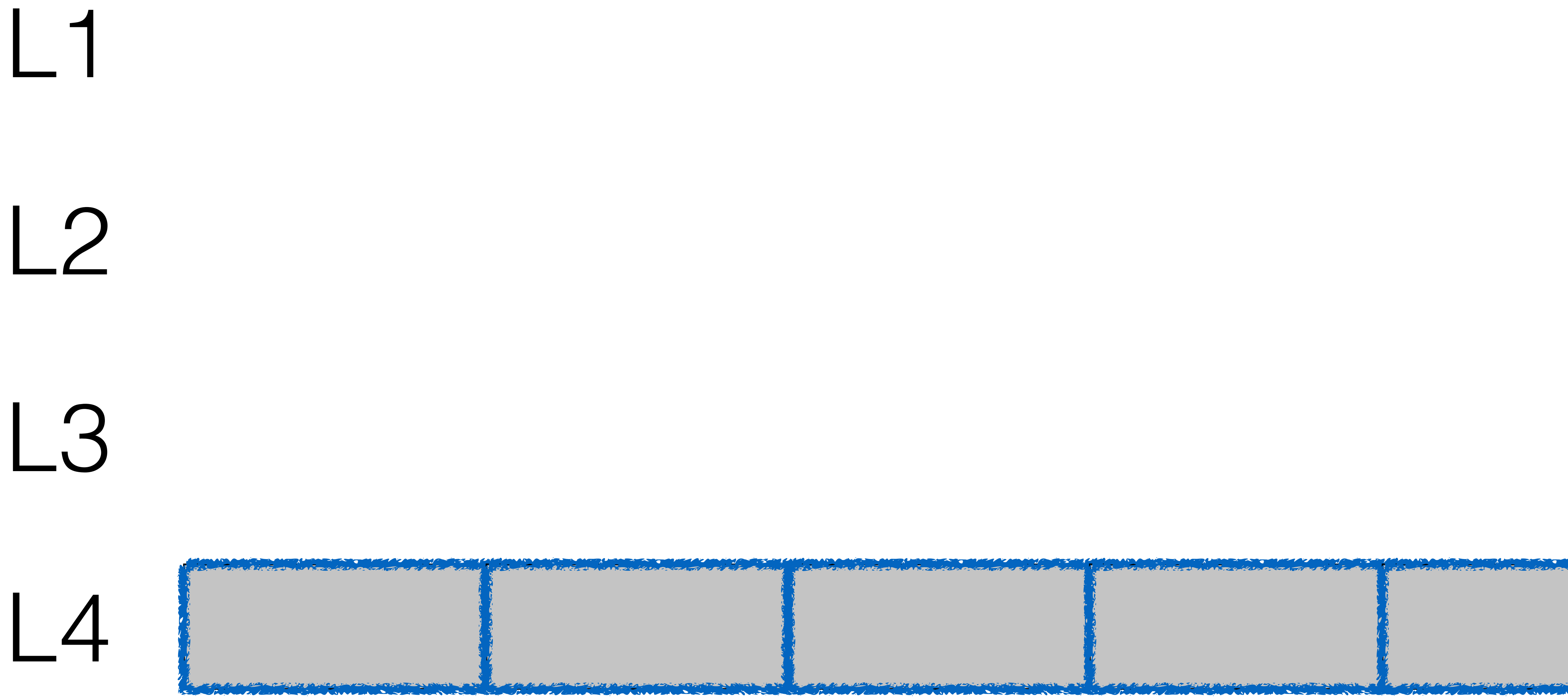


key | value

RID	TS flag	timestamp	name	department	…	location

sort key

delete key



latency spikes ✗

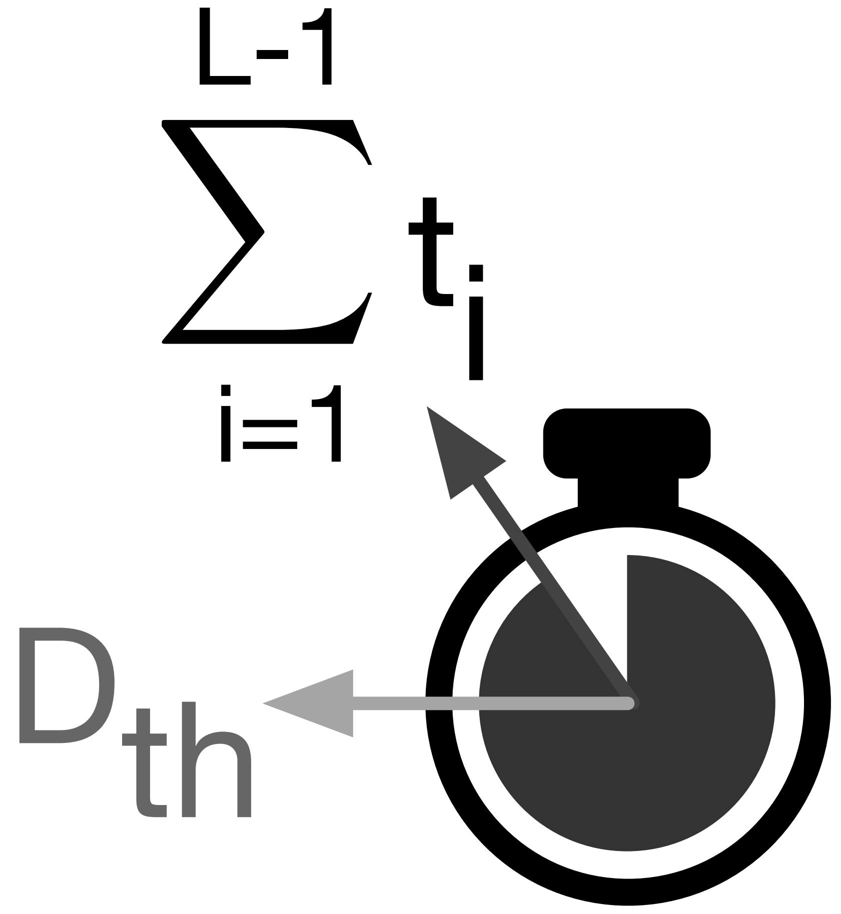
superfluous I/Os ✗

the problems

poor read perf.

write amplification

space amplification



unbounded delete persistence latency

latency spikes

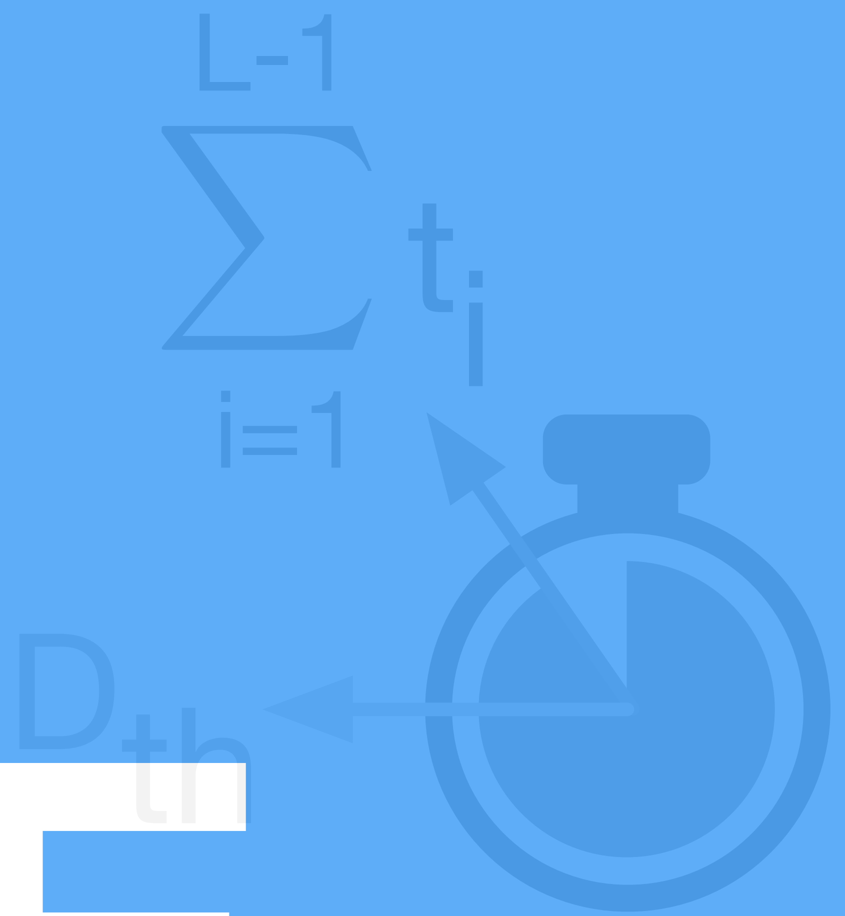
superfluous I/Os

the solution

poor read perf.

write amplification

space amplification



FADE

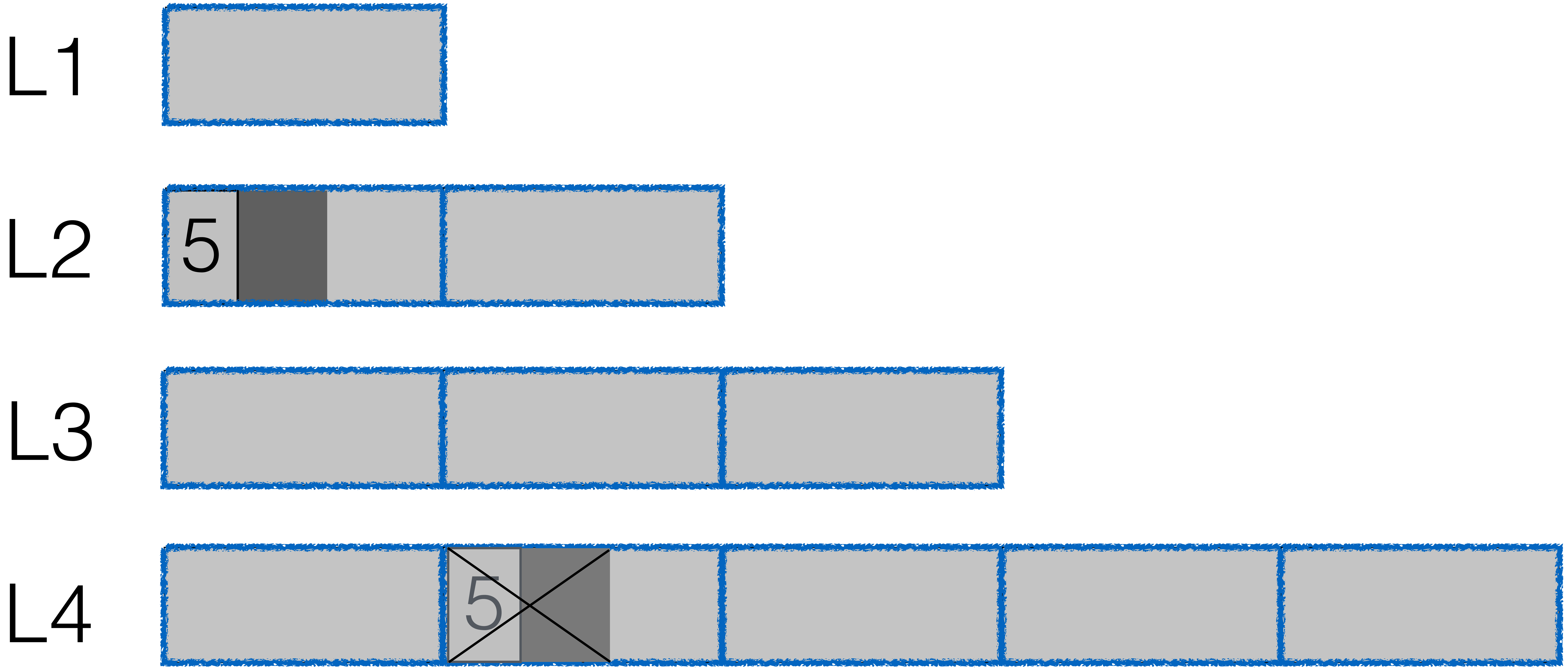
unbounded delete persistence latency

latency spikes

superfluous I/Os

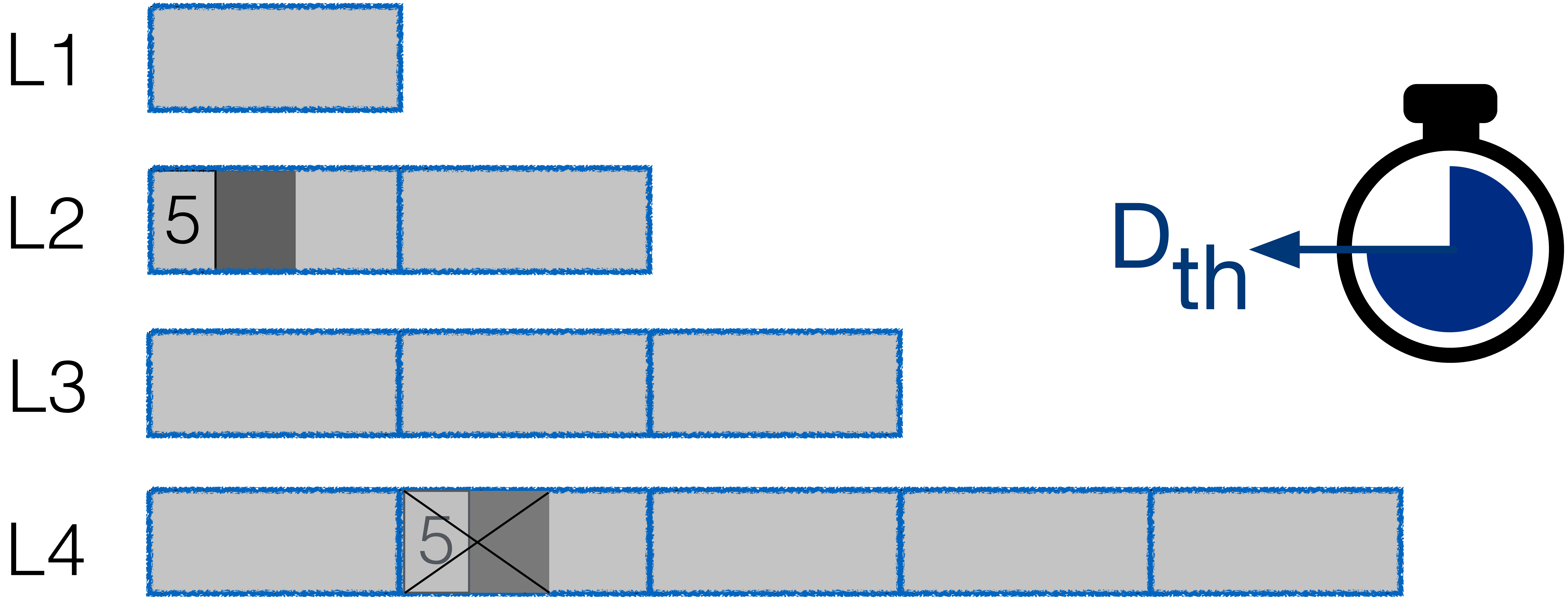
FAst DElete

delete(5) within a threshold time: **D**$_{\mathbf{th}}$



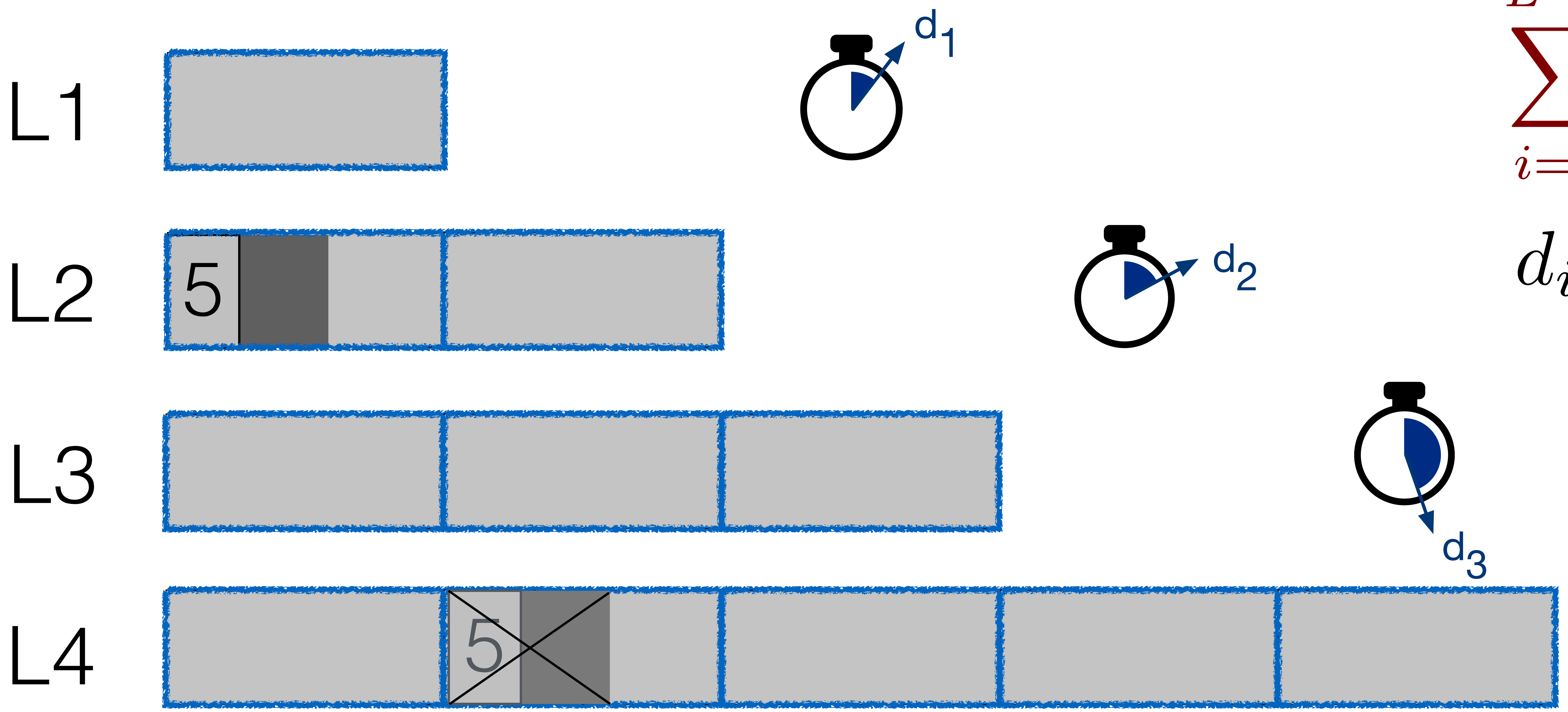
FAst DElete

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



FAst DElete

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

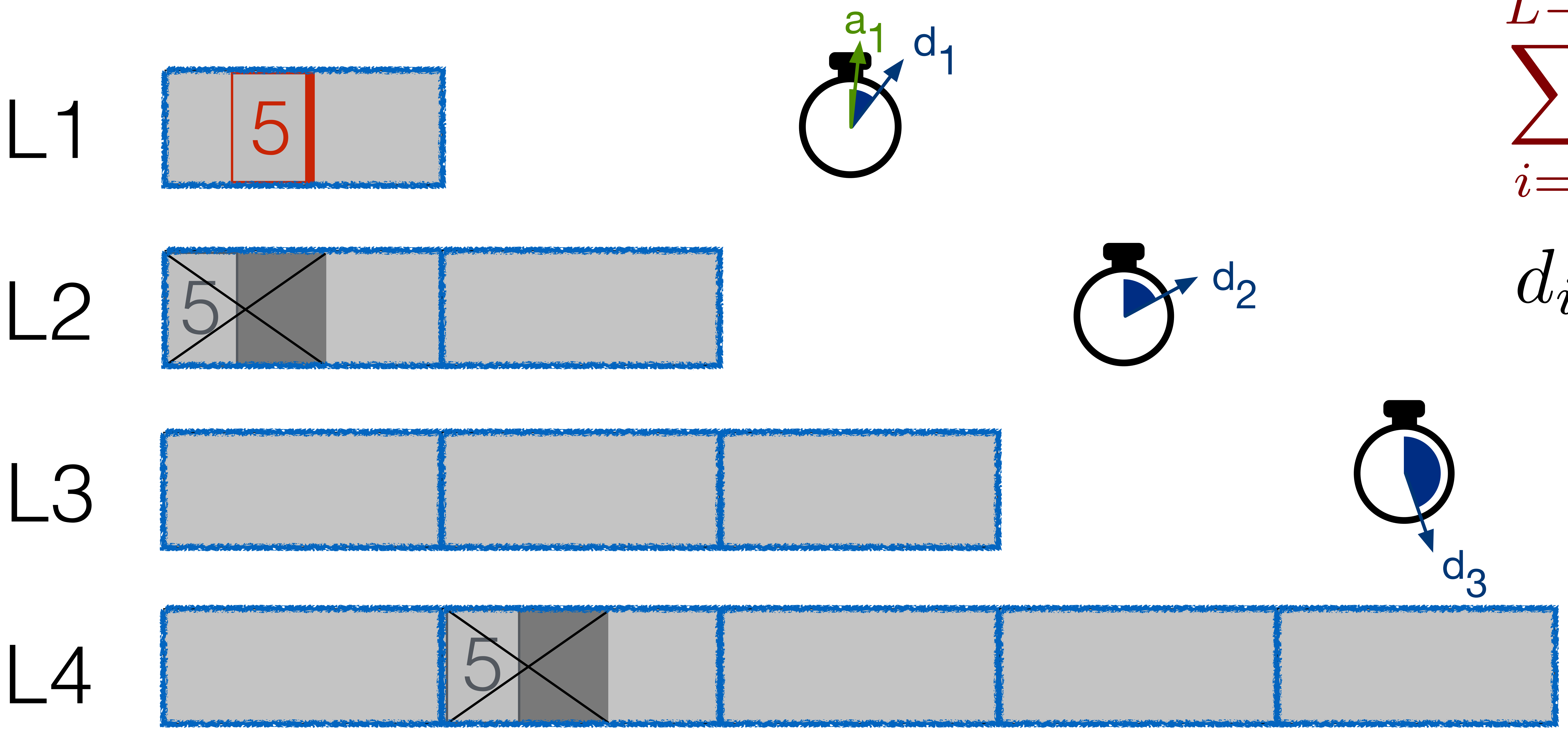


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

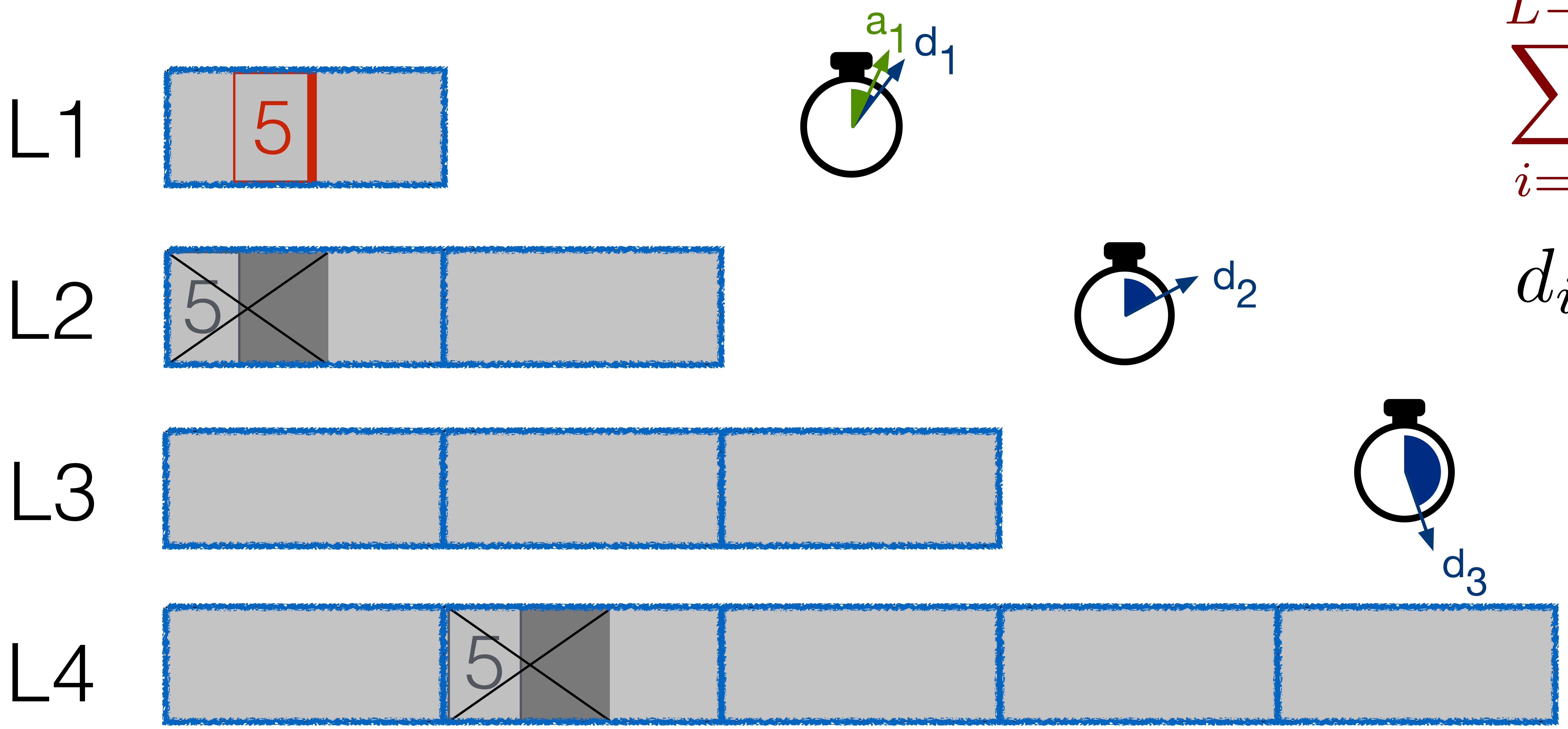


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

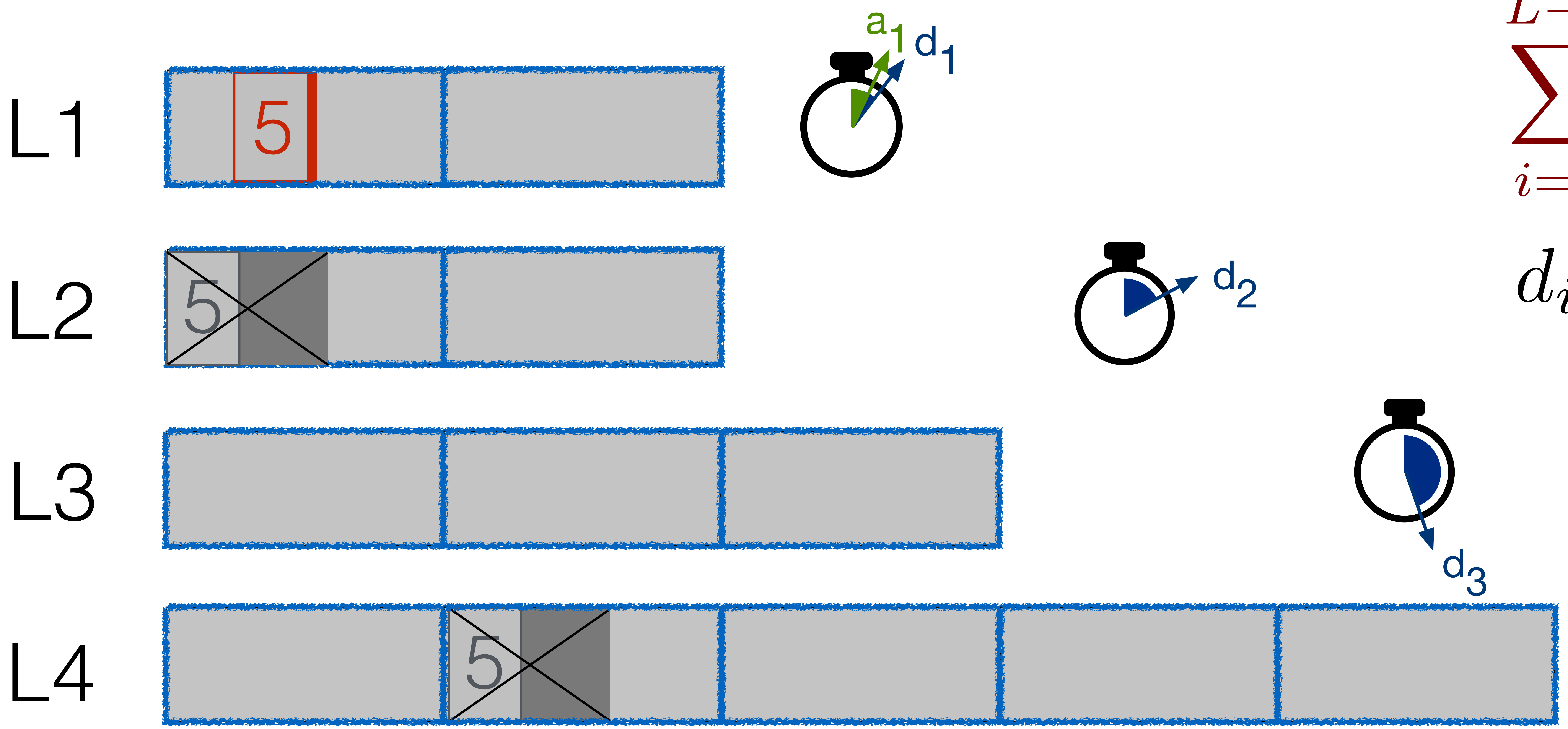


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

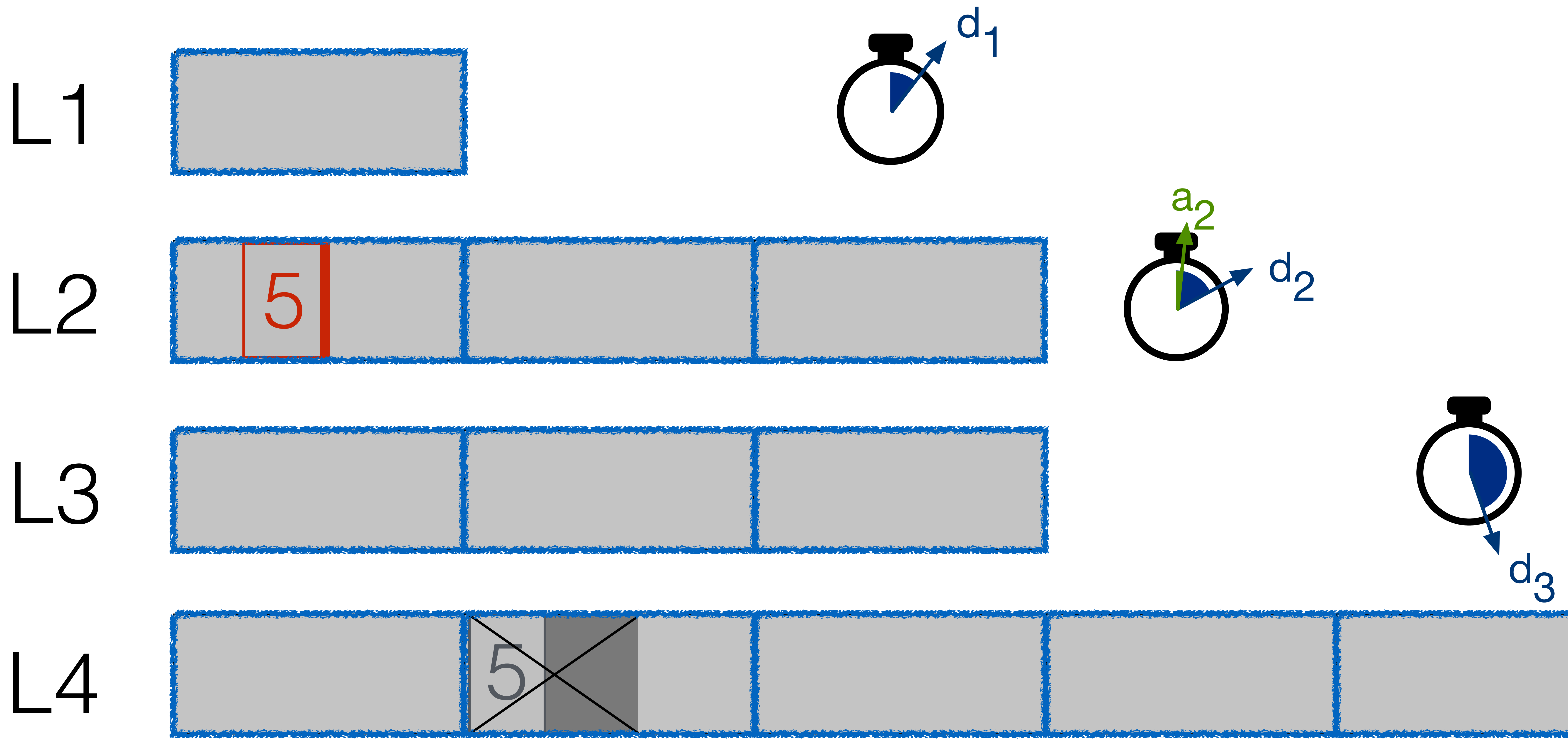


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

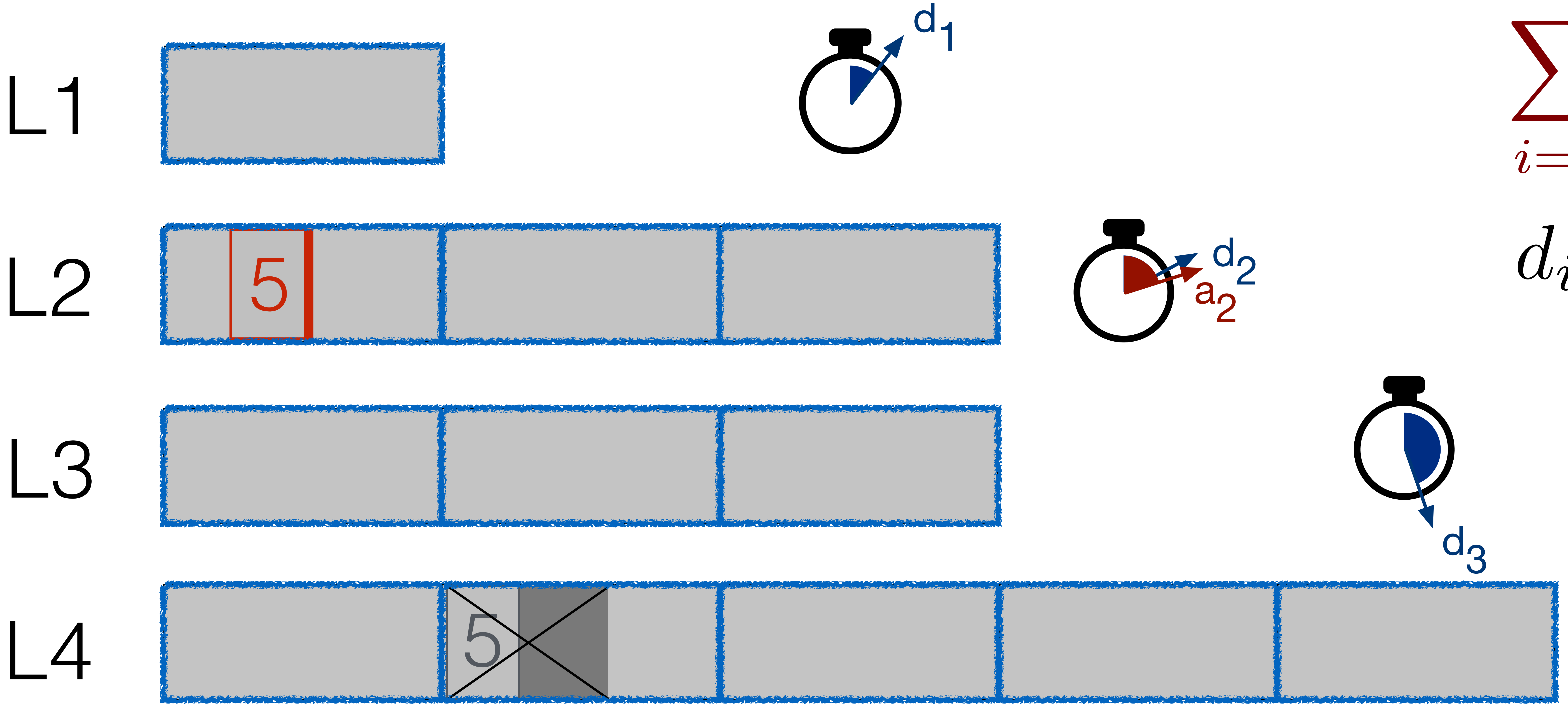


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

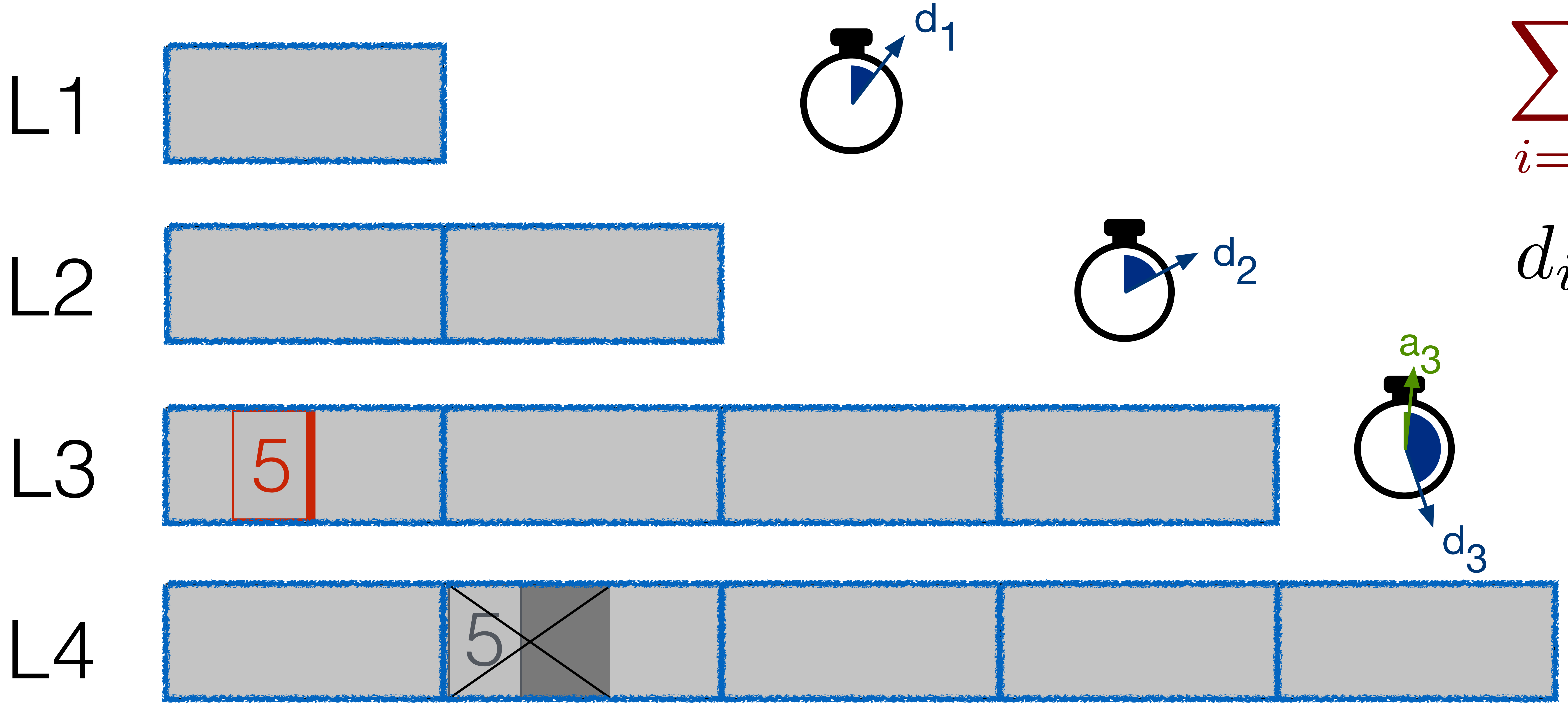


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

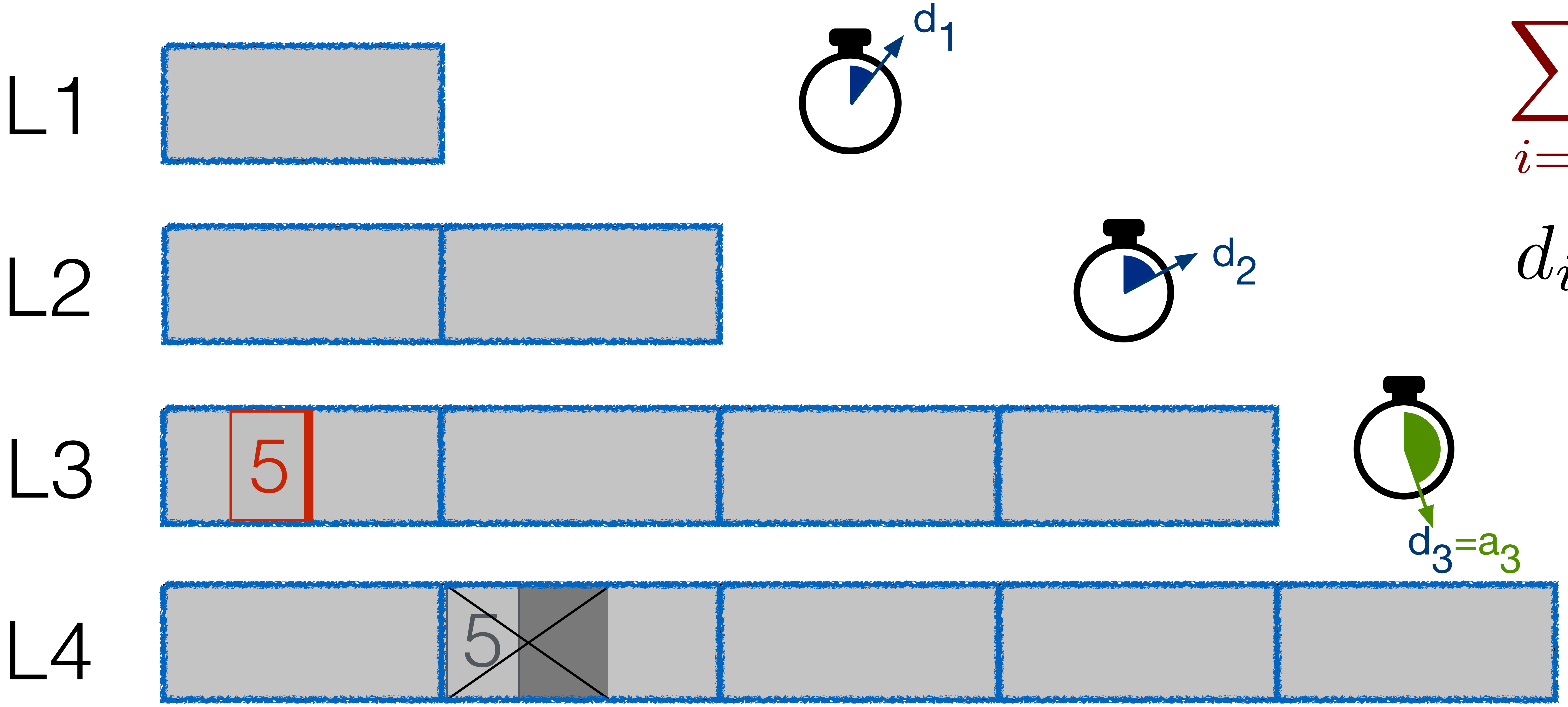


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

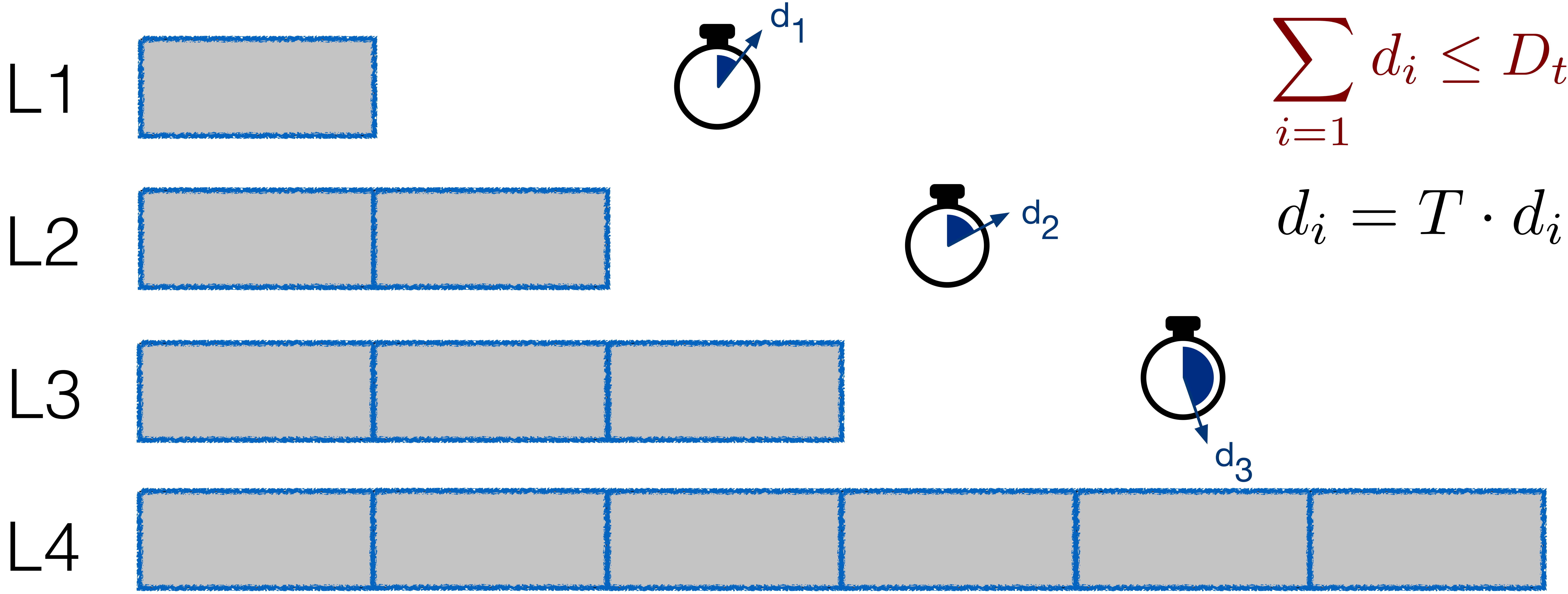


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

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

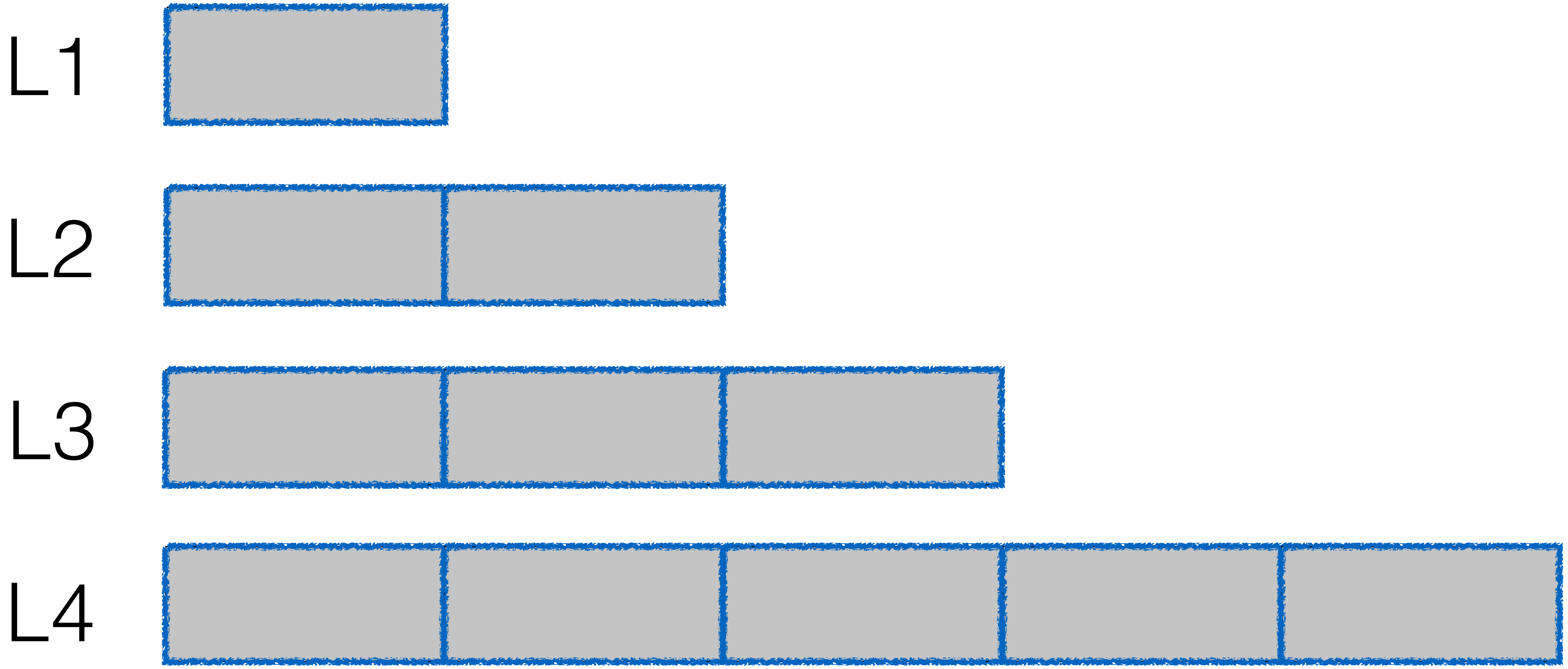


$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

FAst DElete

breaking ties in practical workloads



L1

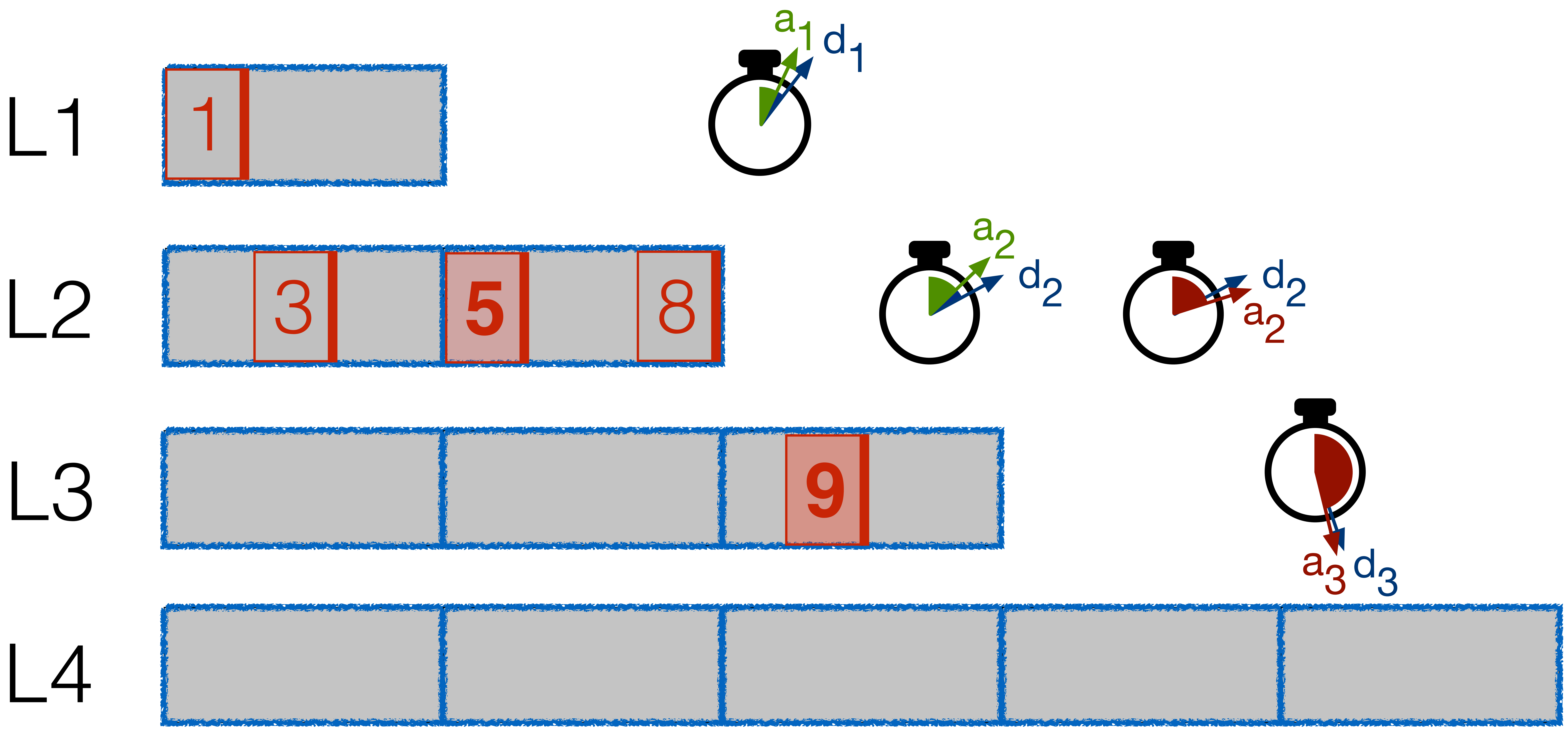
L2

L3

L4

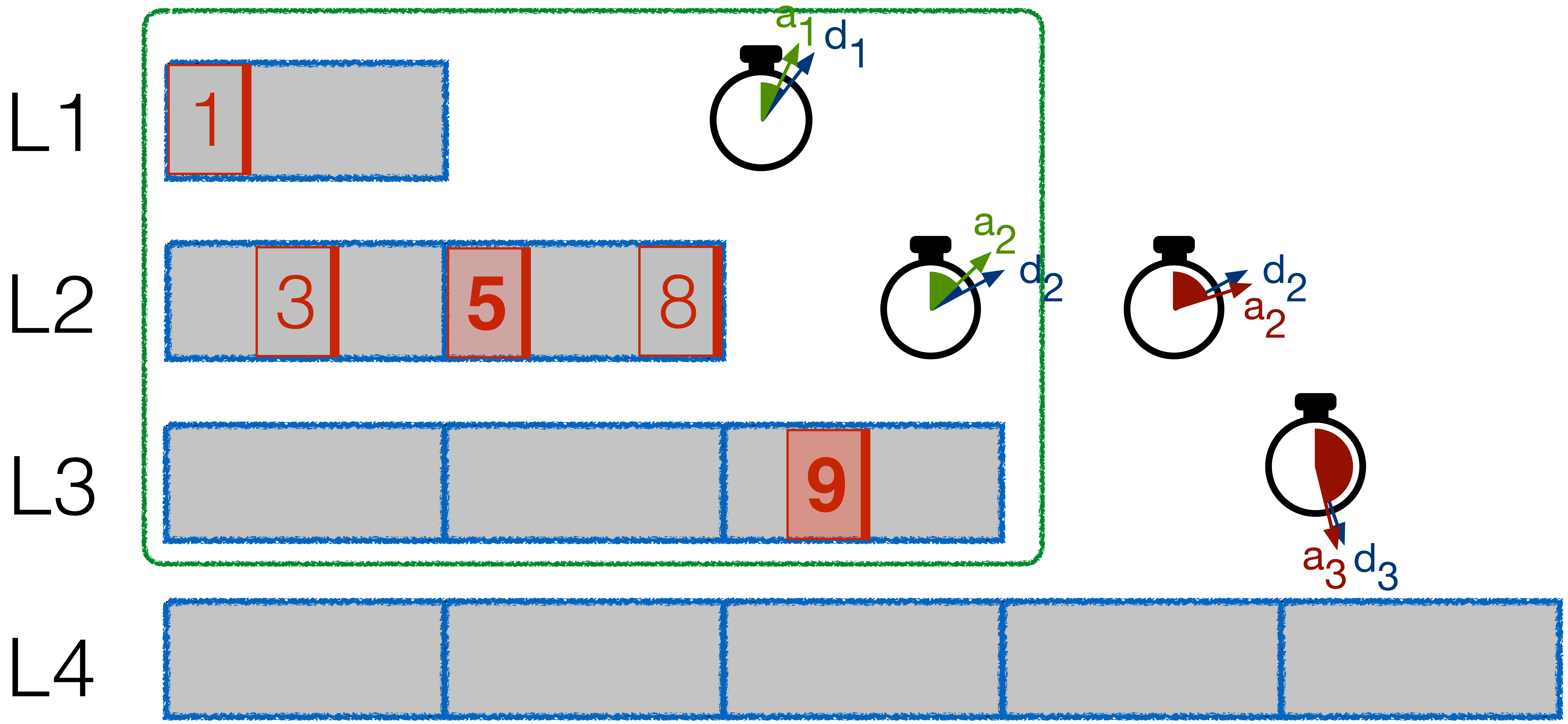
FAst DElete

breaking ties in practical workloads



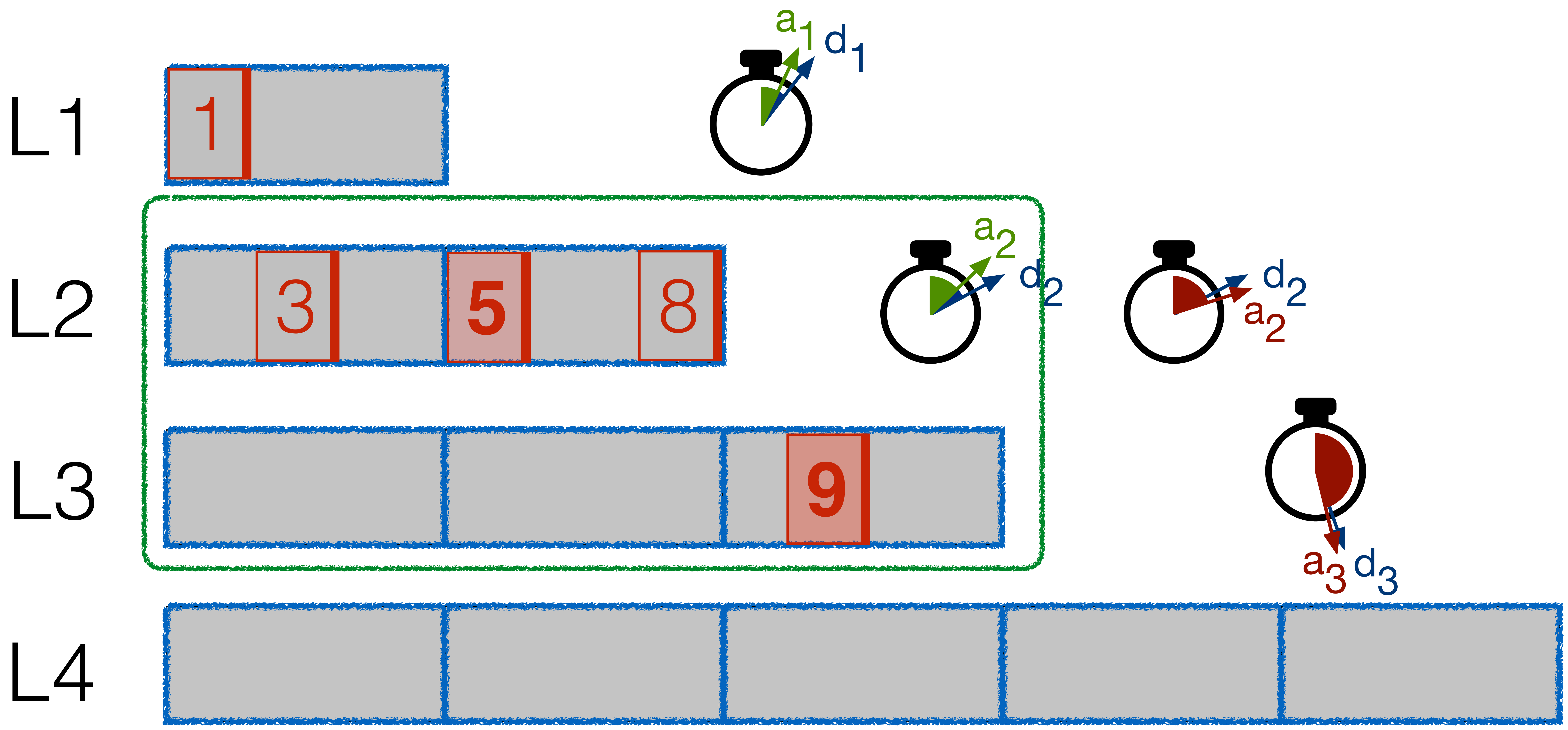
FAst DElete

breaking ties in practical workloads



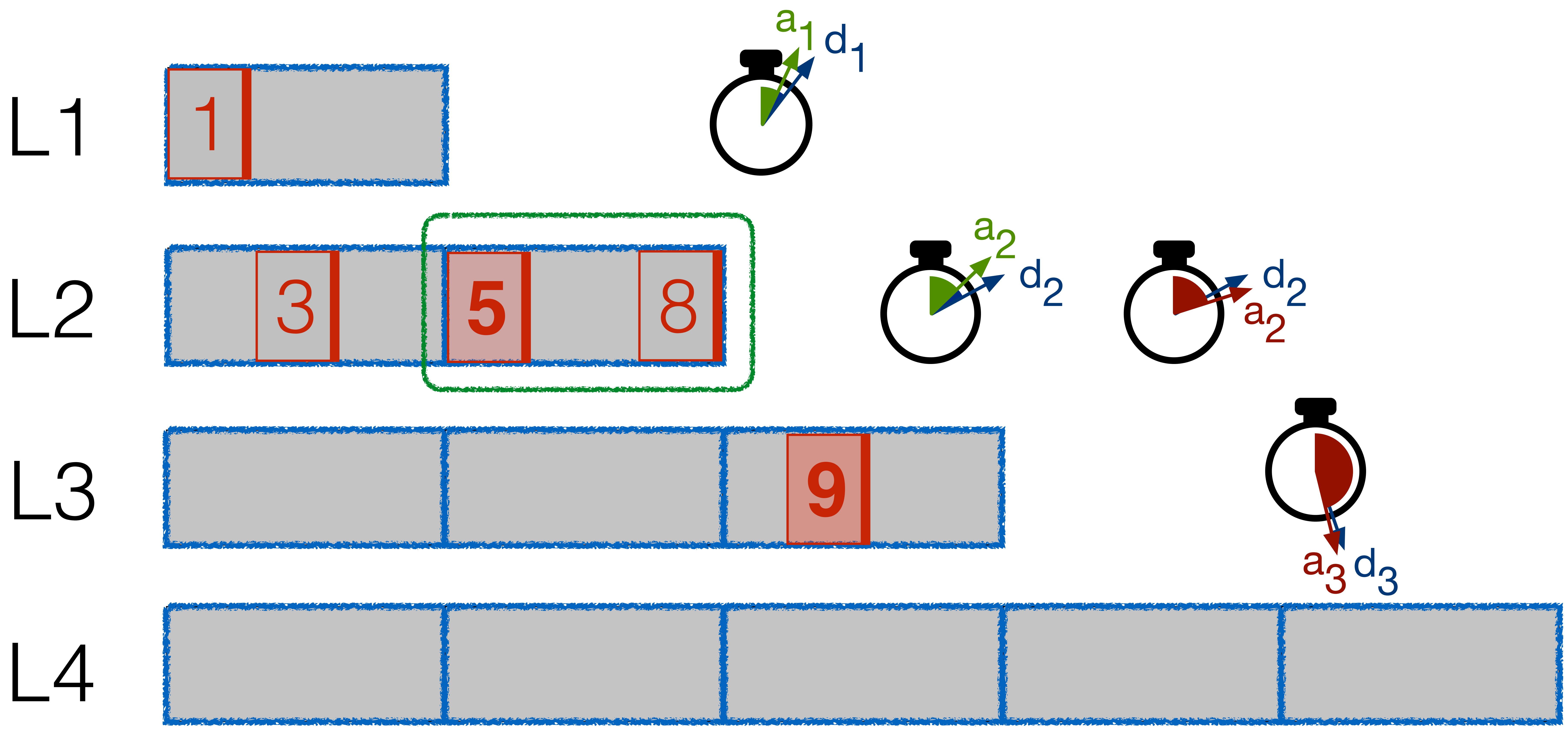
FAst DElete

breaking ties in practical workloads

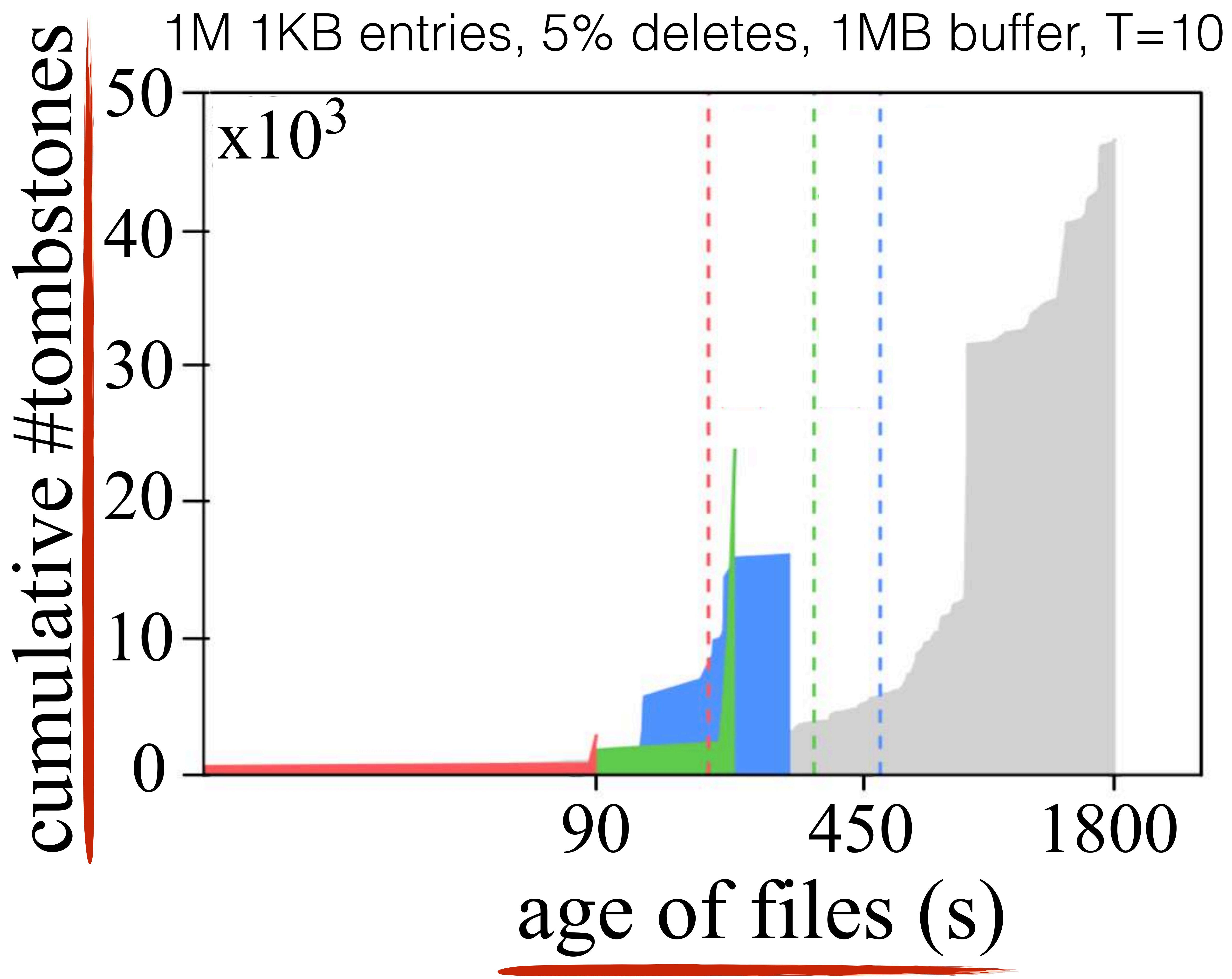


FAst DElete

breaking ties in practical workloads



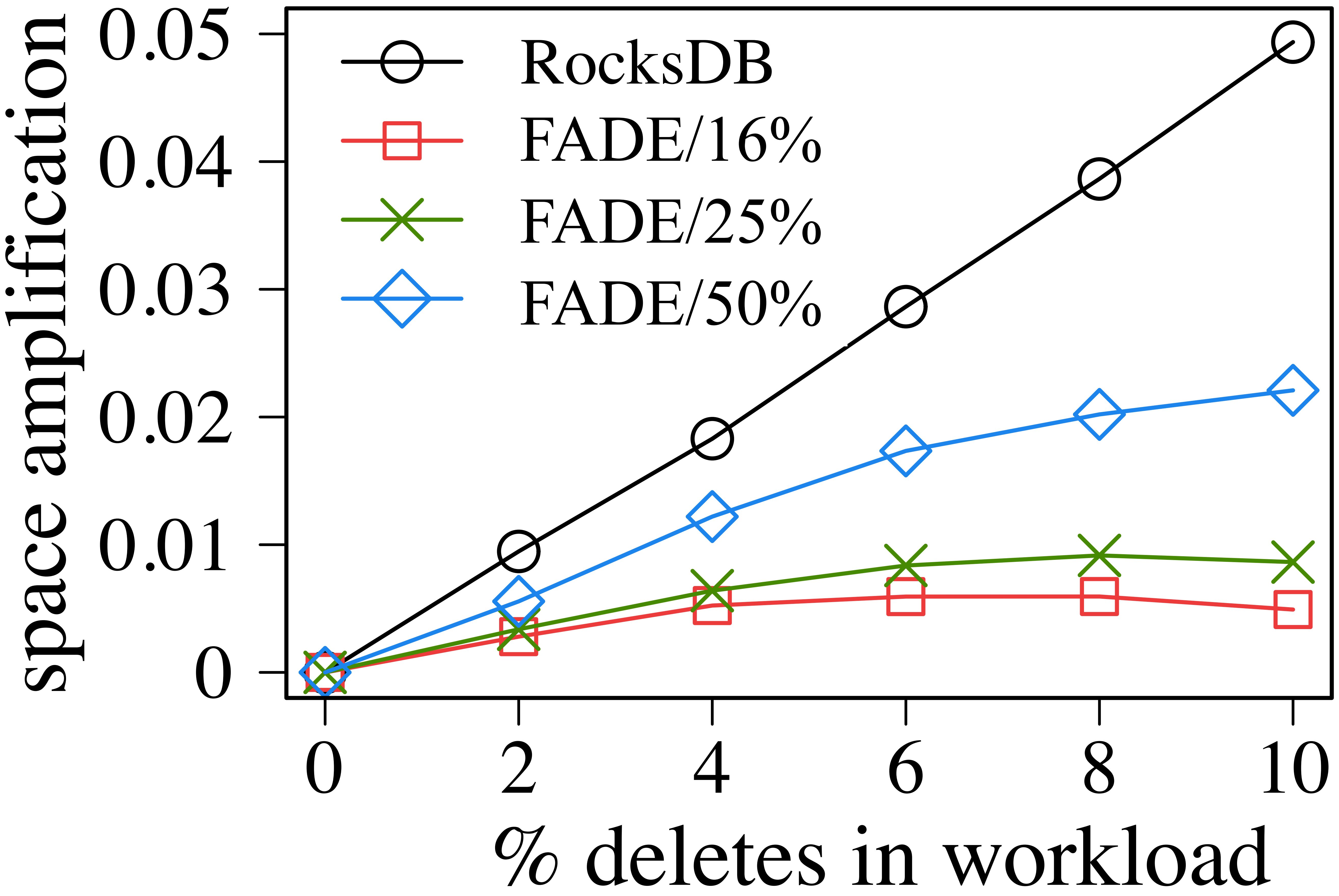
FAst DElete



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 ✓

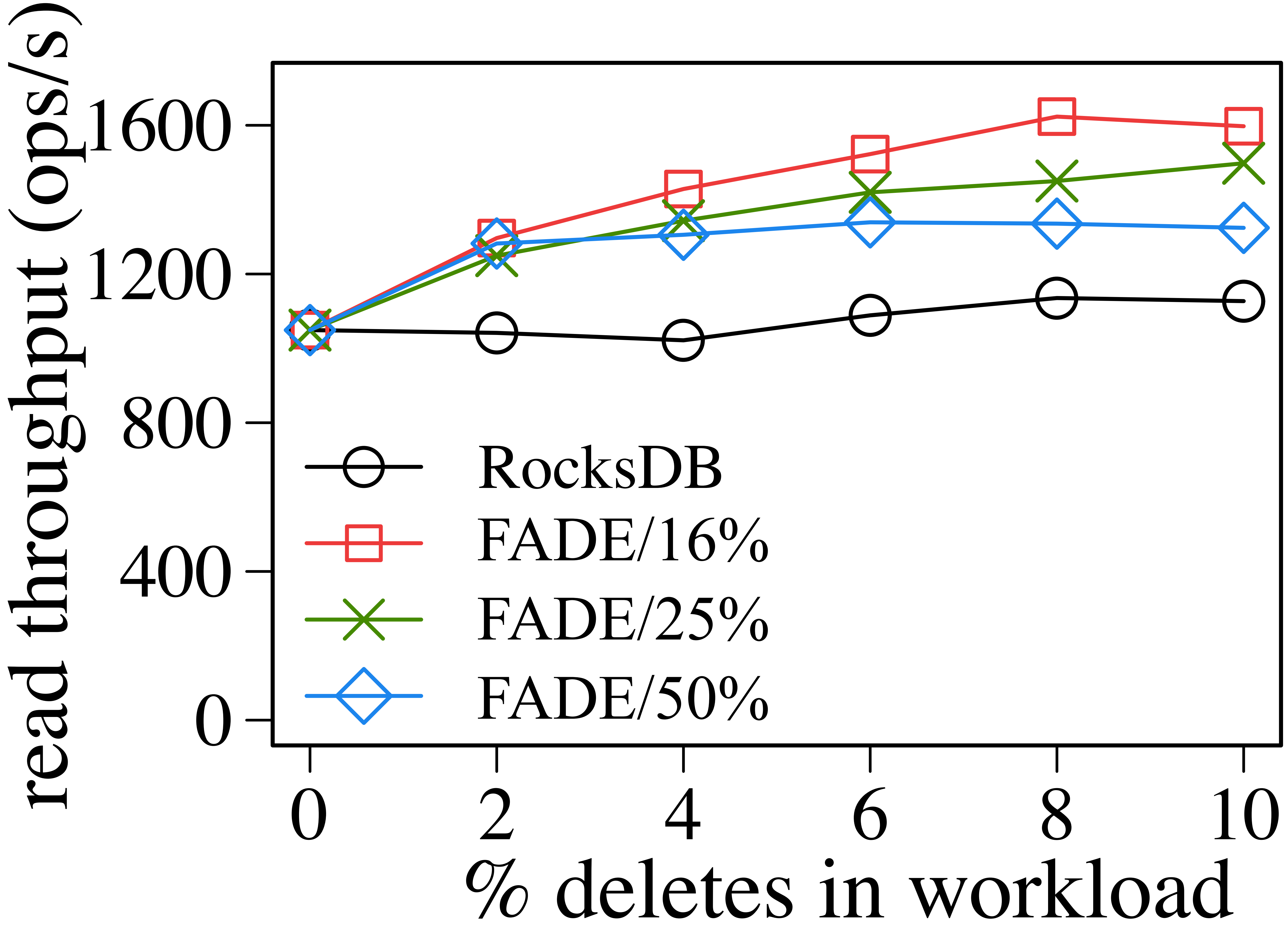
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

within D_{th}



FAst DElete

higher write amplification ↑

4 - 25 %

improved read performance ✓

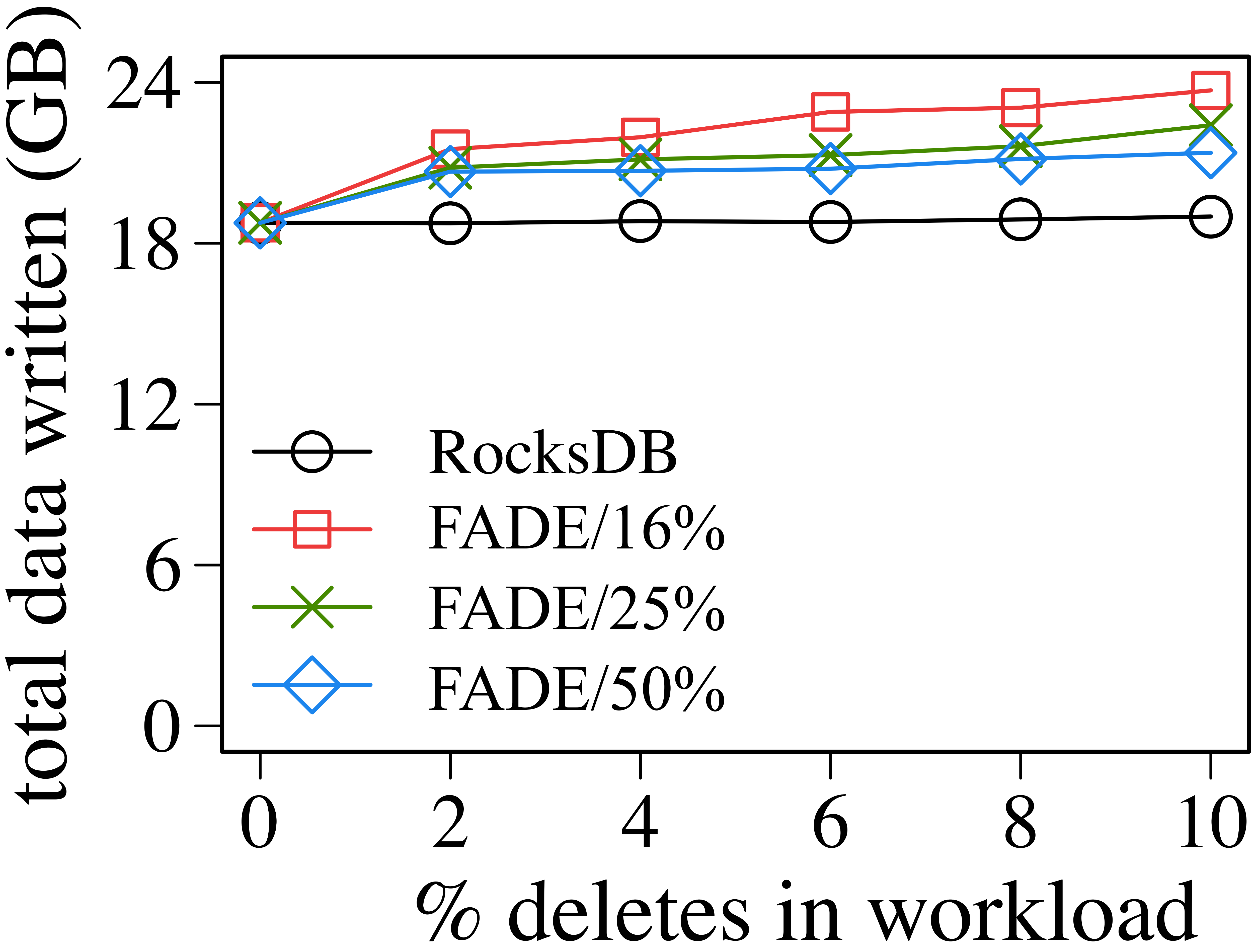
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

within D_{th}



FAst DElete

higher write amplification ↑

4 - 25 %

improved read performance ✓

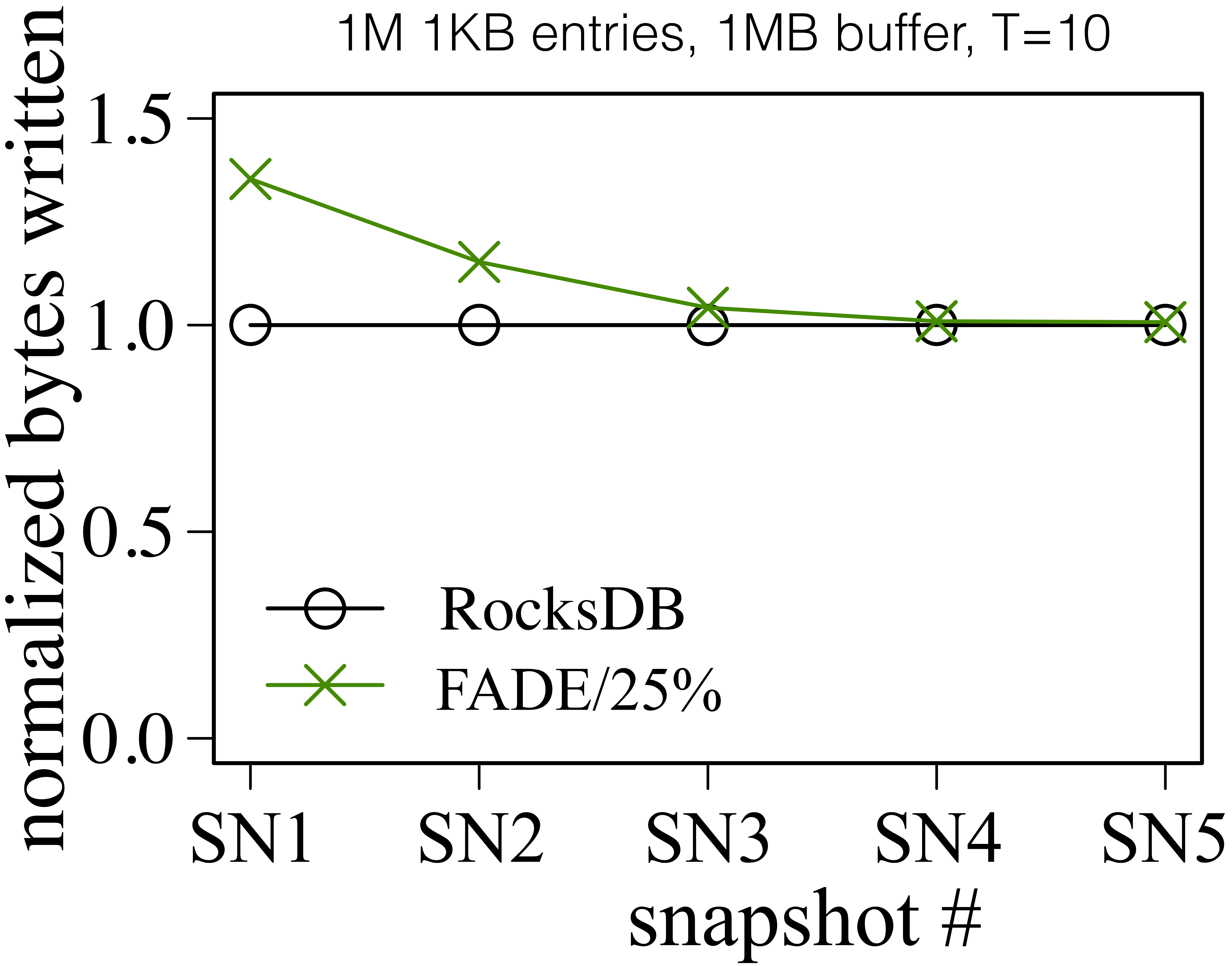
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

within D_{th}



FAst DElete

higher write amplification ●

0.7 %

improved read performance ✓

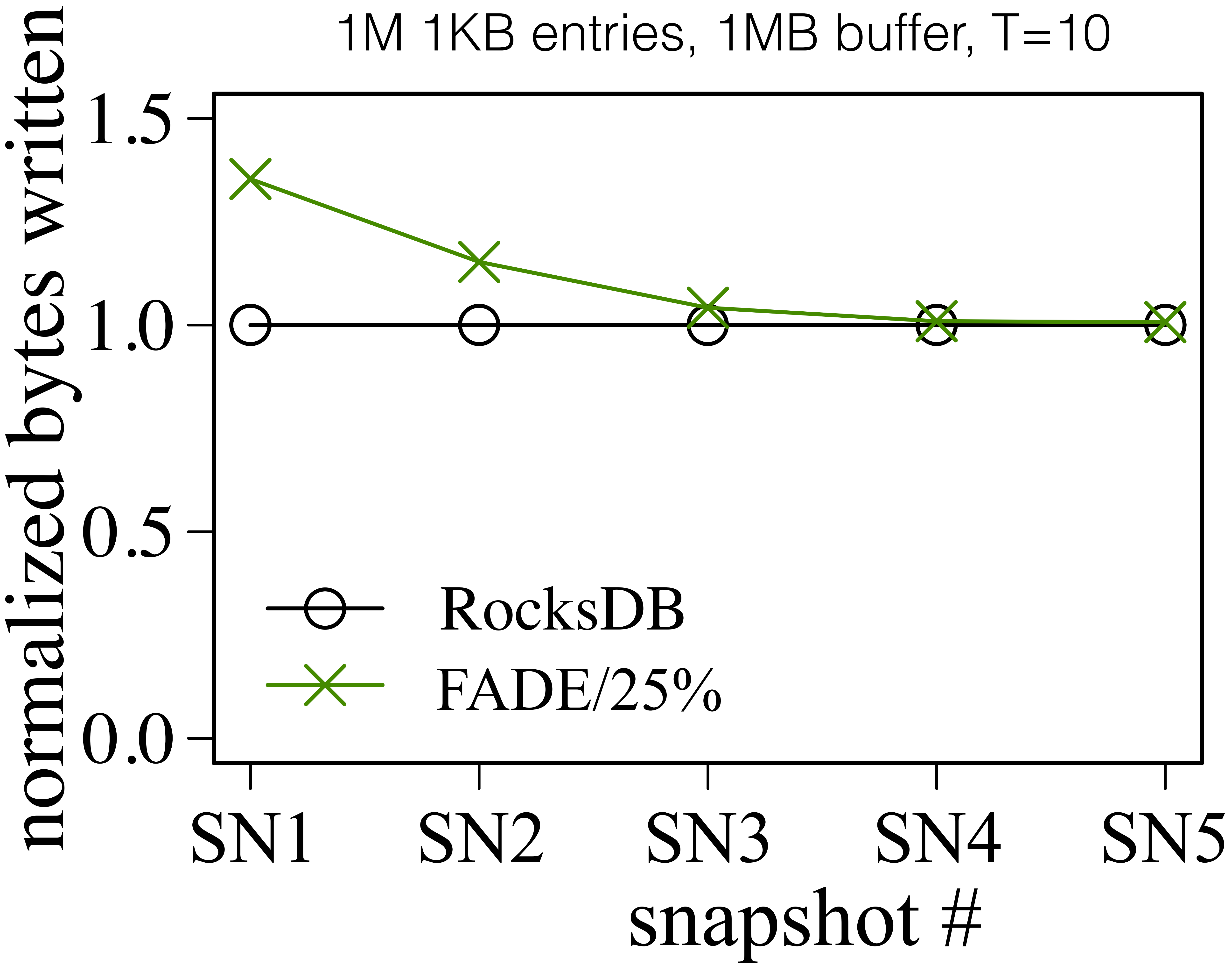
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

within D_{th}

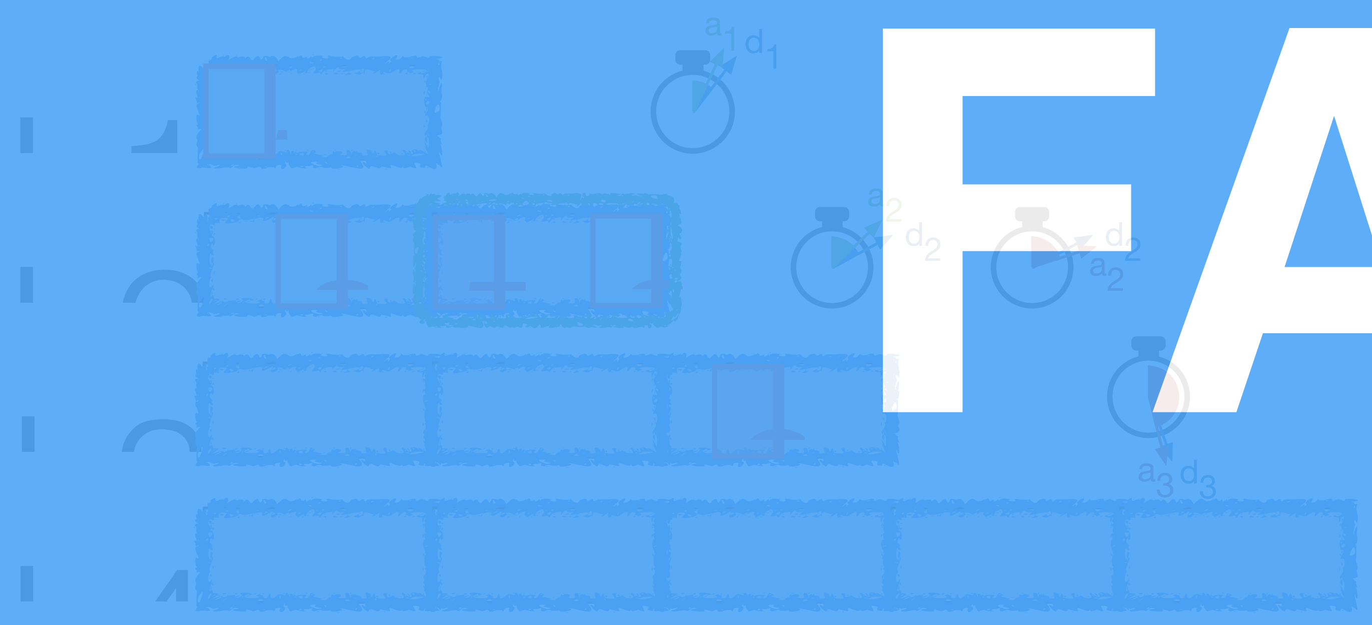


the solution

FADE

FAst DElete

higher write amplification ●



improved read performance ✓

reduced space amplification ✓

timely delete persistence ✓

KiWi

latency spikes

superfluous I/Os

the solution

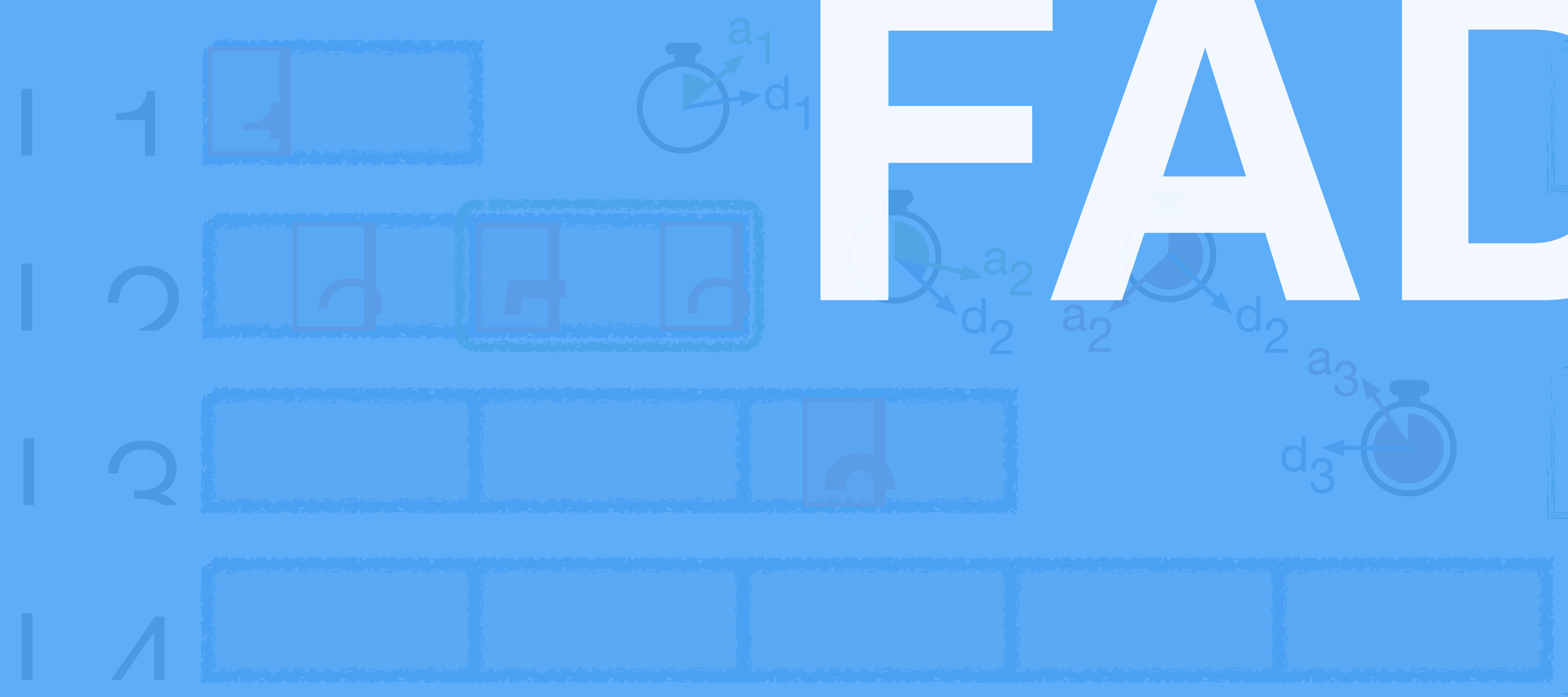
FAst DElete

higher write amplification ●

improved read performance ✓

reduced space amplification ✓

timely delete persistence ✓



FADE

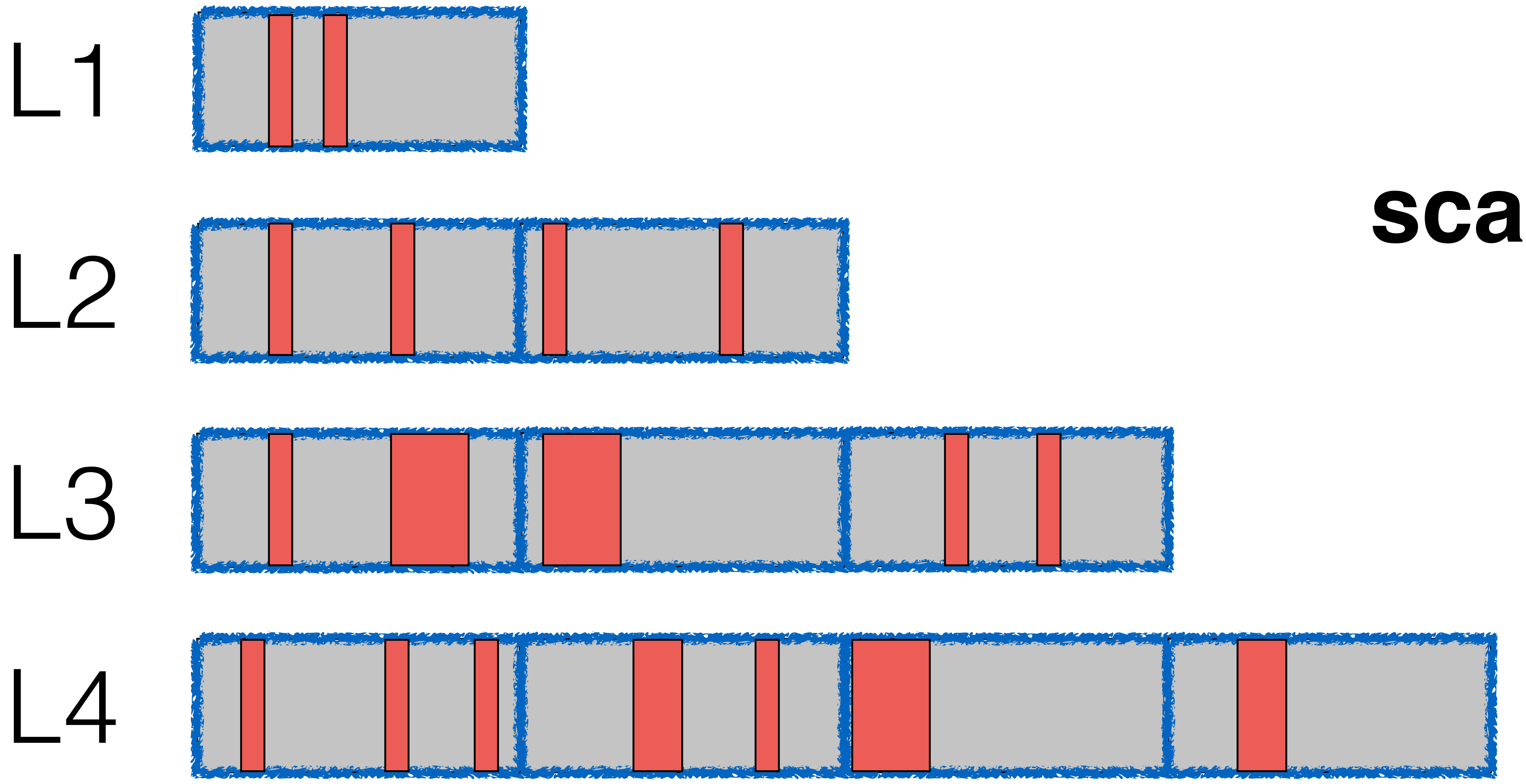
KiWi

latency spikes

superfluous I/Os

Key Weaving storage layout

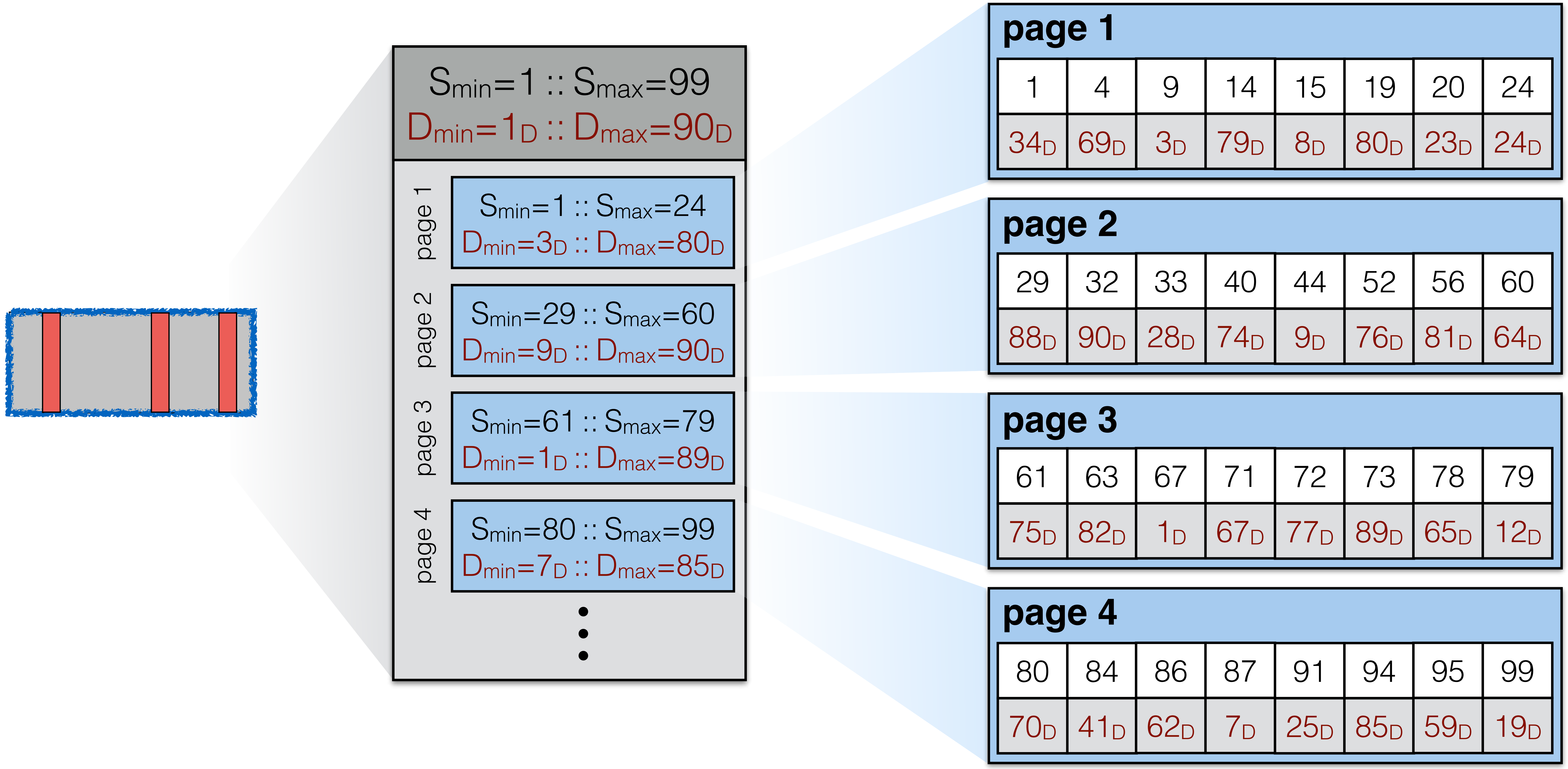
delete all entries older than: **D days**



scattered occurrences

Key Weaving storage layout

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



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

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

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

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

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

⋮

page 1

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

page 2

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

page 3

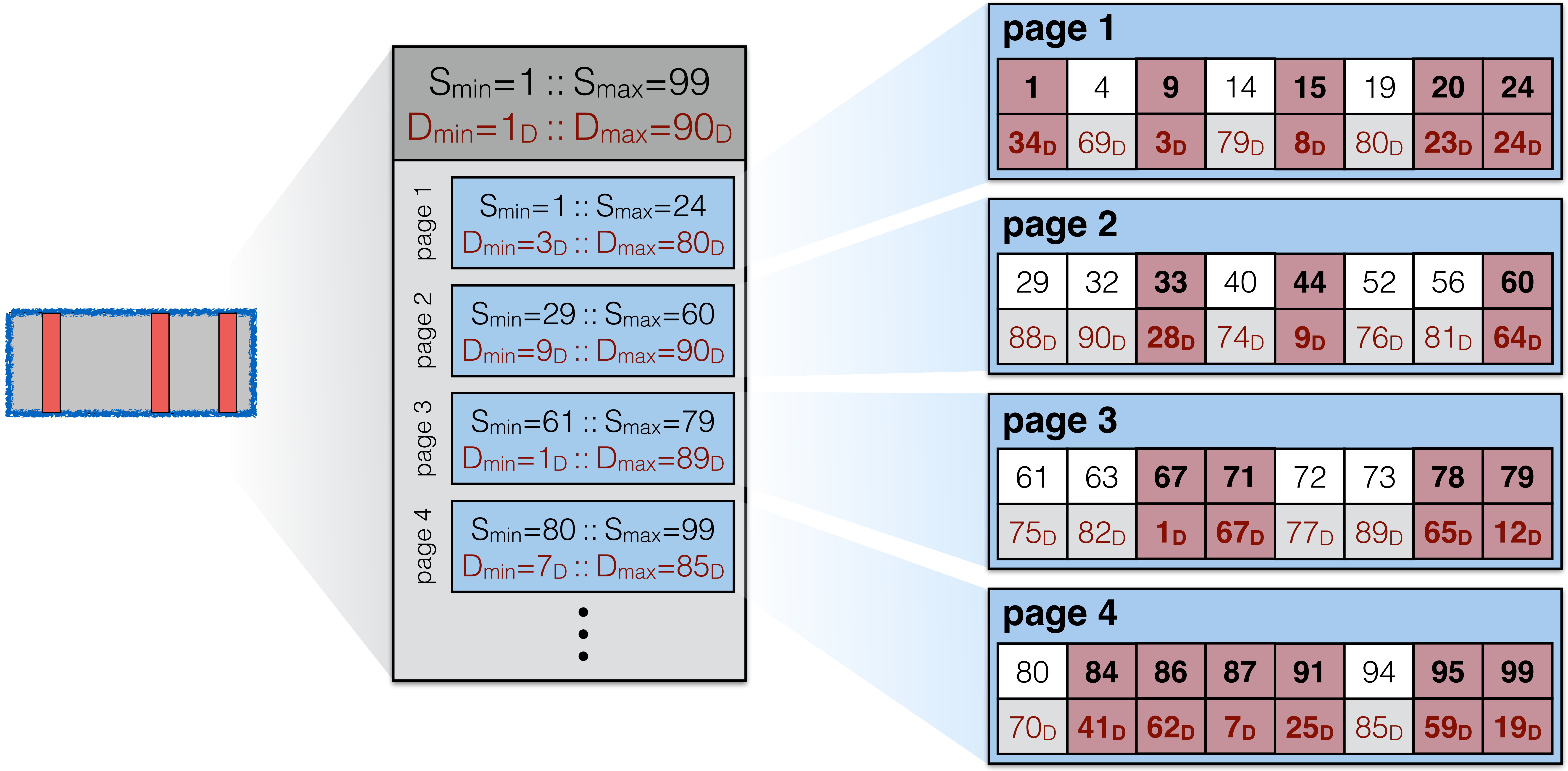
61	63	67	71	72	73	78	79
75_D	82_D	1_D	67_D	77_D	89_D	65_D	12_D

page 4

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

Key Weaving storage layout

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



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

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

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

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

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

⋮

page 1

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

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

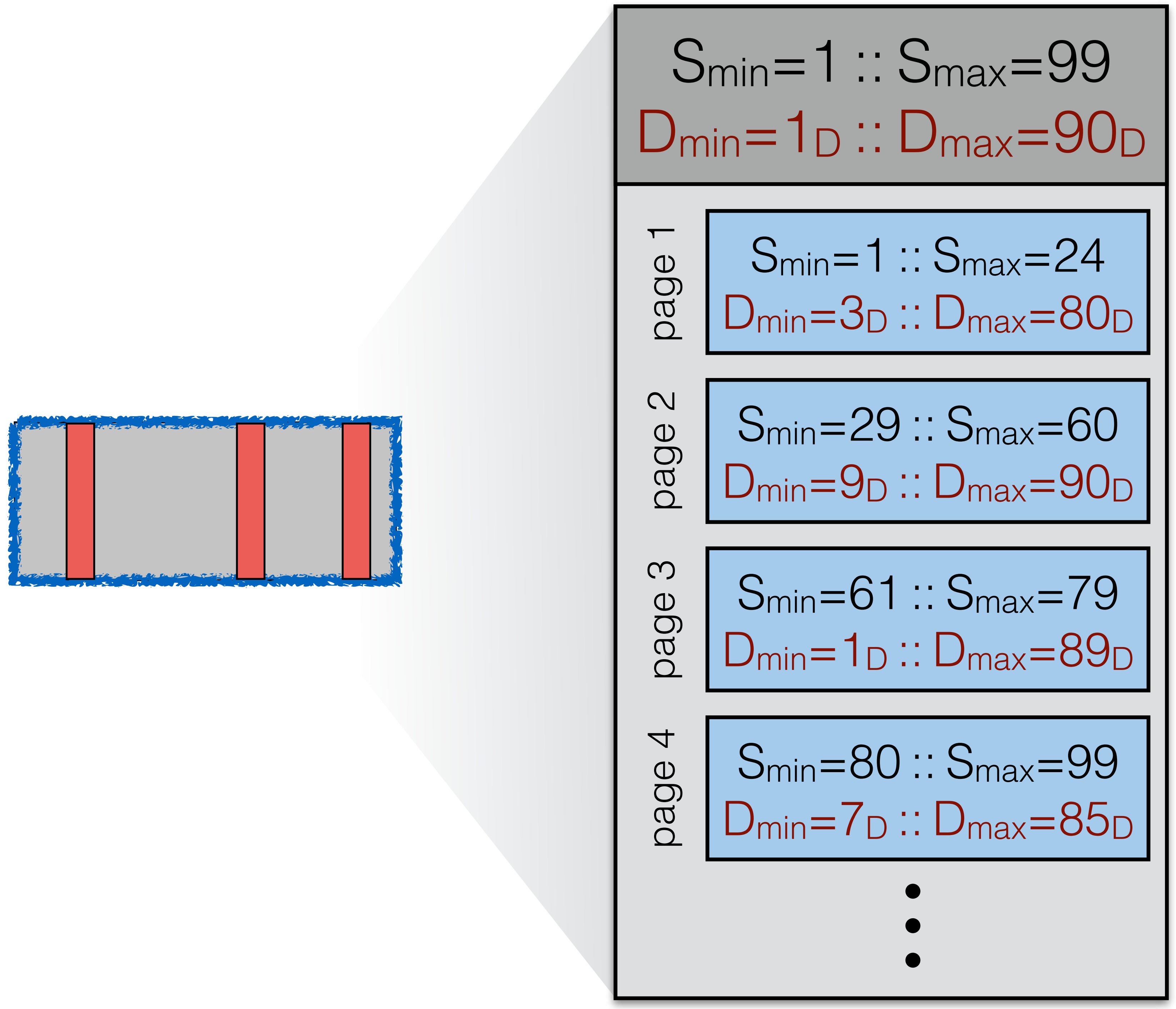
page 4

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 **<= 65$_D$**



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

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

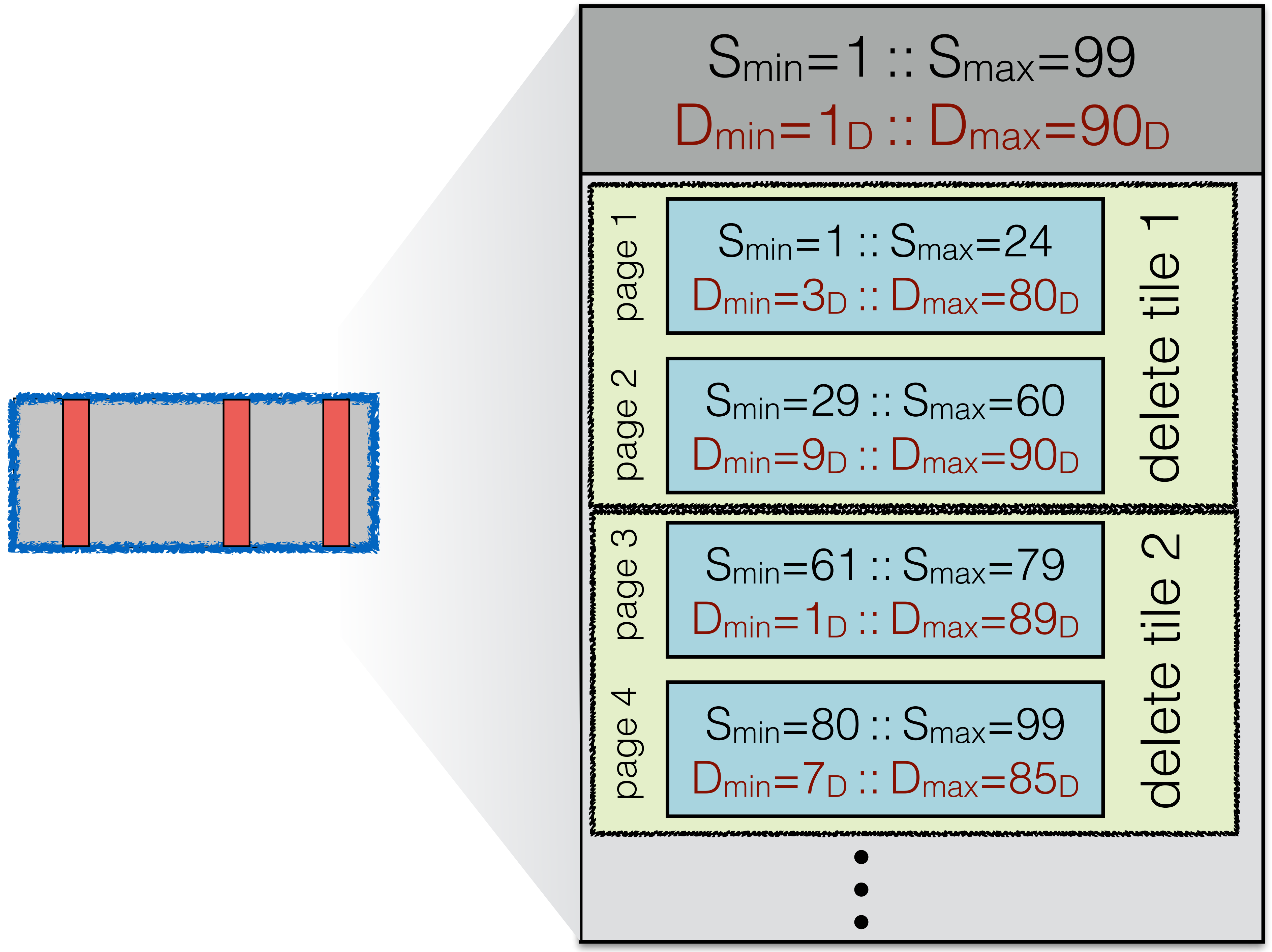
page 2
$S_{min}=29$ :: $S_{max}=60$
$D_{min}=9_D$ :: $D_{max}=90_D$

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

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

Key Weaving storage layout

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



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

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

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

delete tile 1

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

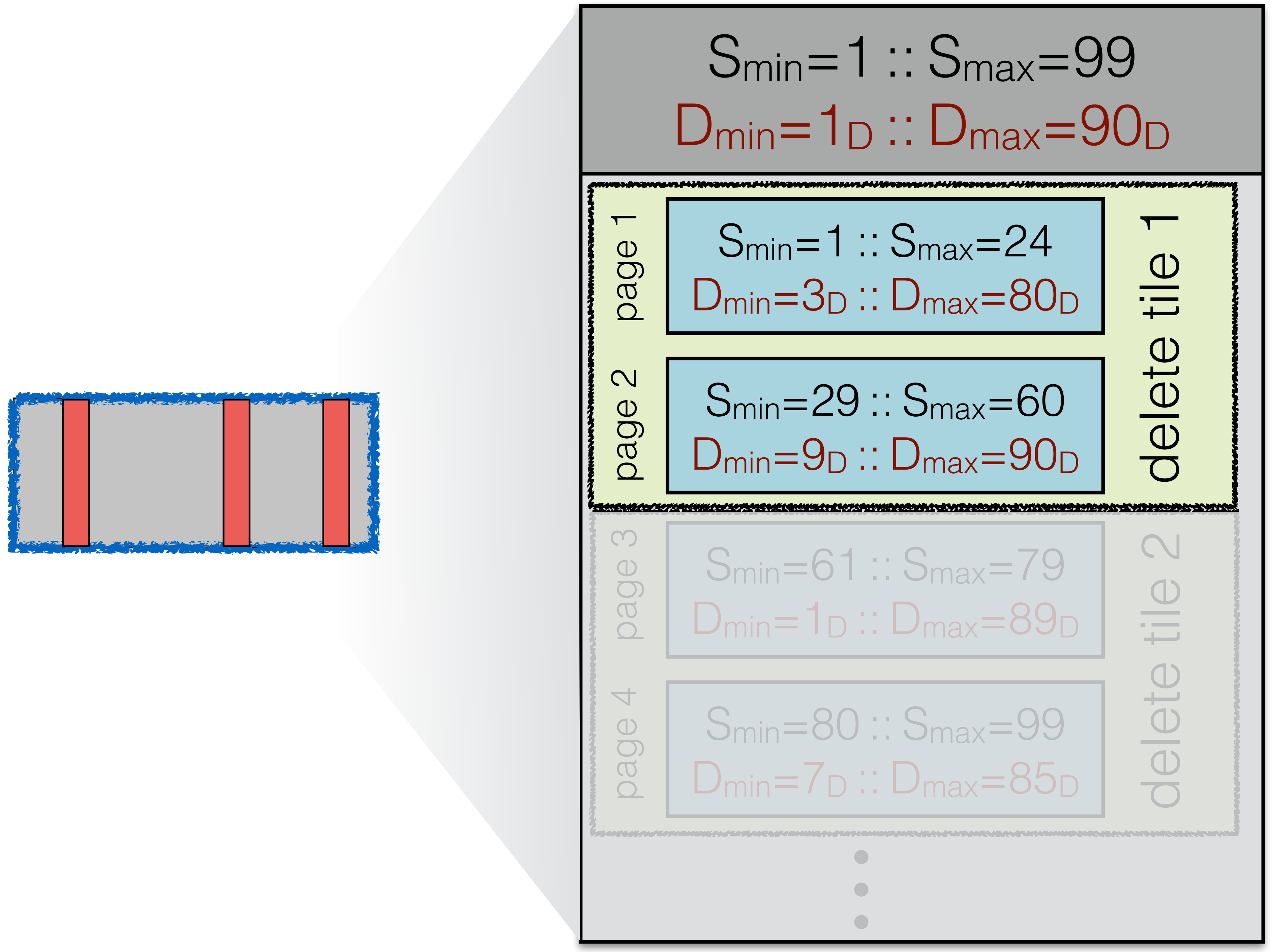
page 4
$S_{min}=80$ :: $S_{max}=99$
$D_{min}=7_D$ :: $D_{max}=85_D$

delete tile 2

partitioned on S

Key Weaving storage layout

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



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

delete tile 1

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

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

delete tile 2

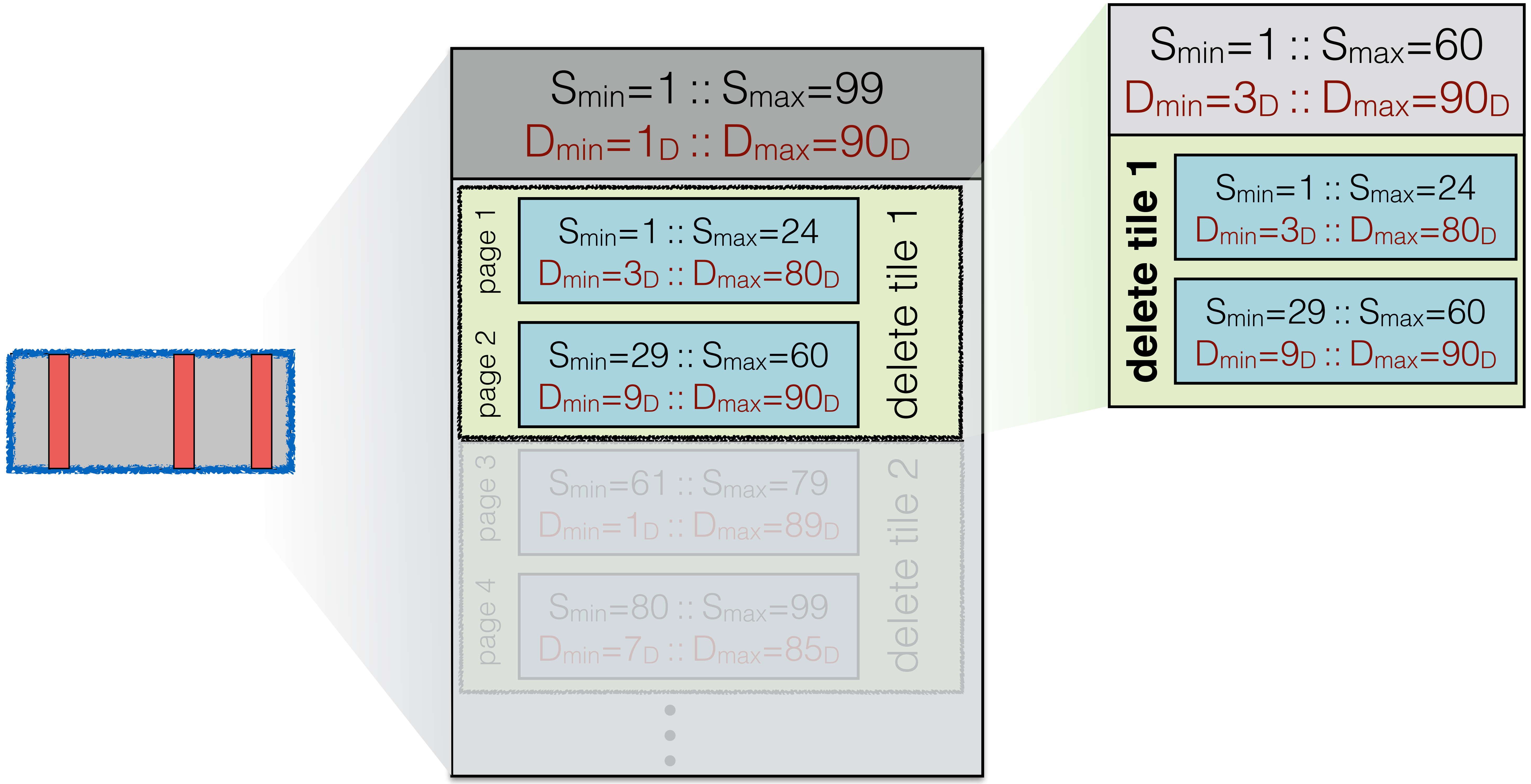
page 3
$S_{min}=61$ :: $S_{max}=79$
$D_{min}=1_D$ :: $D_{max}=89_D$

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

partitioned on S

Key Weaving storage layout

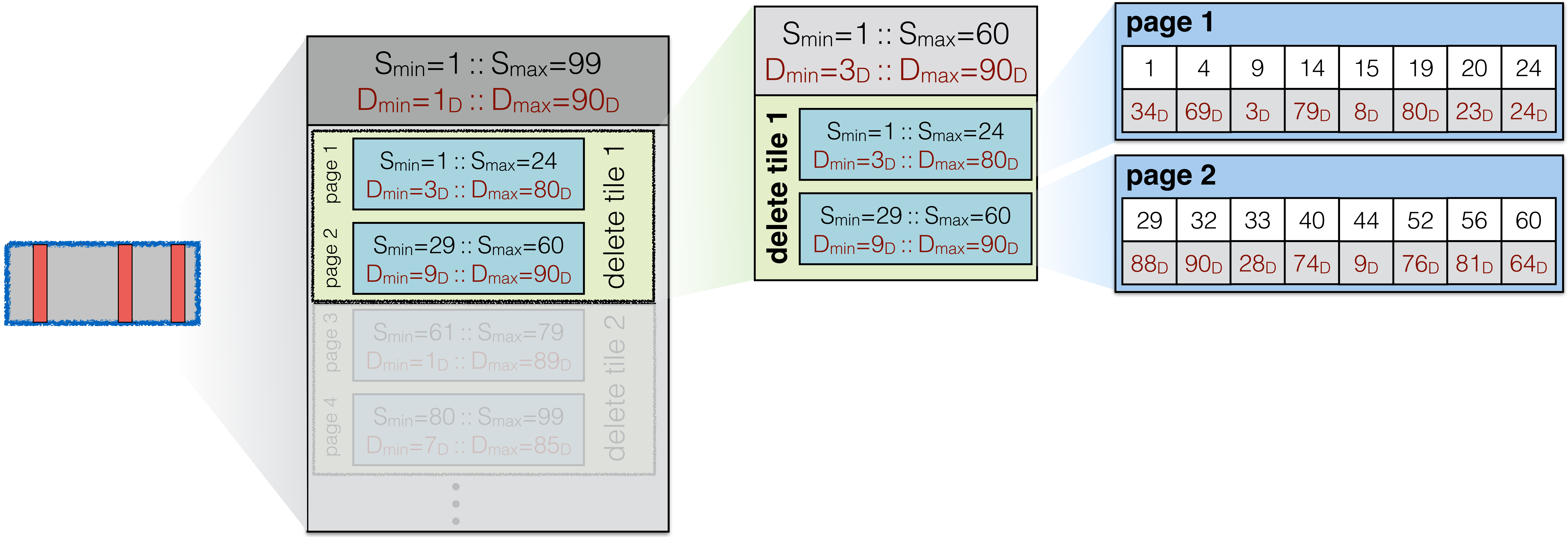
delete all entries with timestamp **<= 65$_D$**



partitioned on S

Key Weaving storage layout

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



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

delete tile 1

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

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

delete tile 2

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

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

delete tile 1

$S_{min}=1 :: S_{max}=60$
$D_{min}=3_D :: D_{max}=90_D$

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

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

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

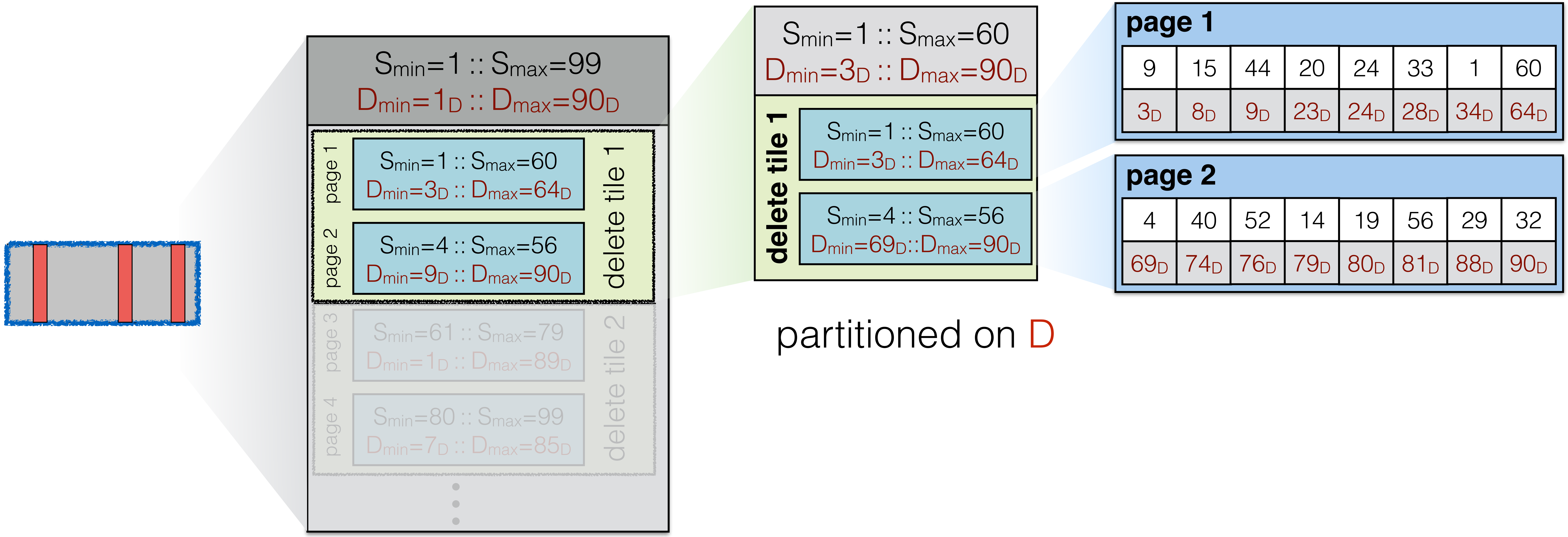
page 2

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

partitioned on S

Key Weaving storage layout

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



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

delete tile 1

page 1: $S_{min}=1 :: S_{max}=60$, $D_{min}=3_D :: D_{max}=64_D$

page 2: $S_{min}=4 :: S_{max}=56$, $D_{min}=9_D :: D_{max}=90_D$

delete tile 2

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

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

partitioned on S

$S_{min}=1 :: S_{max}=60$
$D_{min}=3_D :: D_{max}=90_D$

delete tile 1

$S_{min}=1 :: S_{max}=60$, $D_{min}=3_D :: D_{max}=64_D$

$S_{min}=4 :: S_{max}=56$, $D_{min}=69_D::D_{max}=90_D$

partitioned on D

page 1

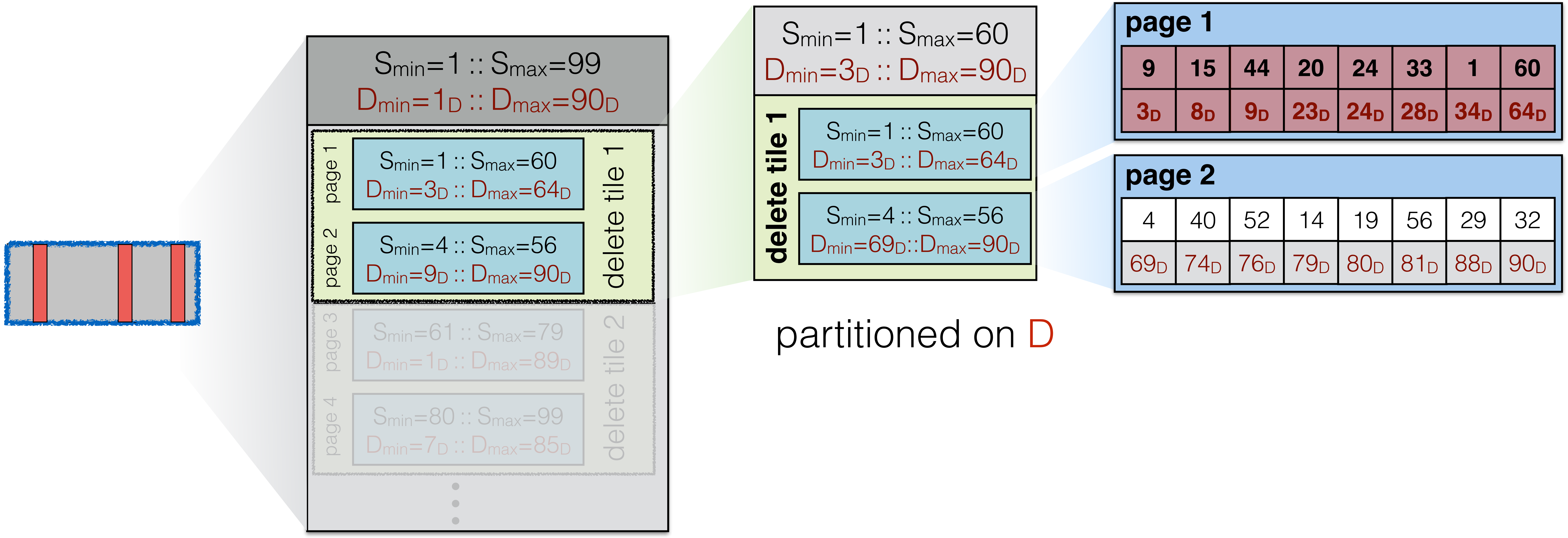
9	15	44	20	24	33	1	60
3_D	8_D	9_D	23_D	24_D	28_D	34_D	64_D

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 **<= 65_D**



$S_{min}=1$ :: $S_{max}=99$
$D_{min}=1_D$ :: $D_{max}=90_D$
page 1
$S_{min}=1$ :: $S_{max}=60$
$D_{min}=3_D$ :: $D_{max}=64_D$
page 2
$S_{min}=4$ :: $S_{max}=56$
$D_{min}=9_D$ :: $D_{max}=90_D$
delete tile 1
page 3
$S_{min}=61$ :: $S_{max}=79$
$D_{min}=1_D$ :: $D_{max}=89_D$
page 4
$S_{min}=80$ :: $S_{max}=99$
$D_{min}=7_D$ :: $D_{max}=85_D$
delete tile 2
$S_{min}=1$ :: $S_{max}=60$
$D_{min}=3_D$ :: $D_{max}=90_D$
delete tile 1
$S_{min}=1$ :: $S_{max}=60$
$D_{min}=3_D$ :: $D_{max}=64_D$
$S_{min}=4$ :: $S_{max}=56$
$D_{min}=69_D$::$D_{max}=90_D$

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**

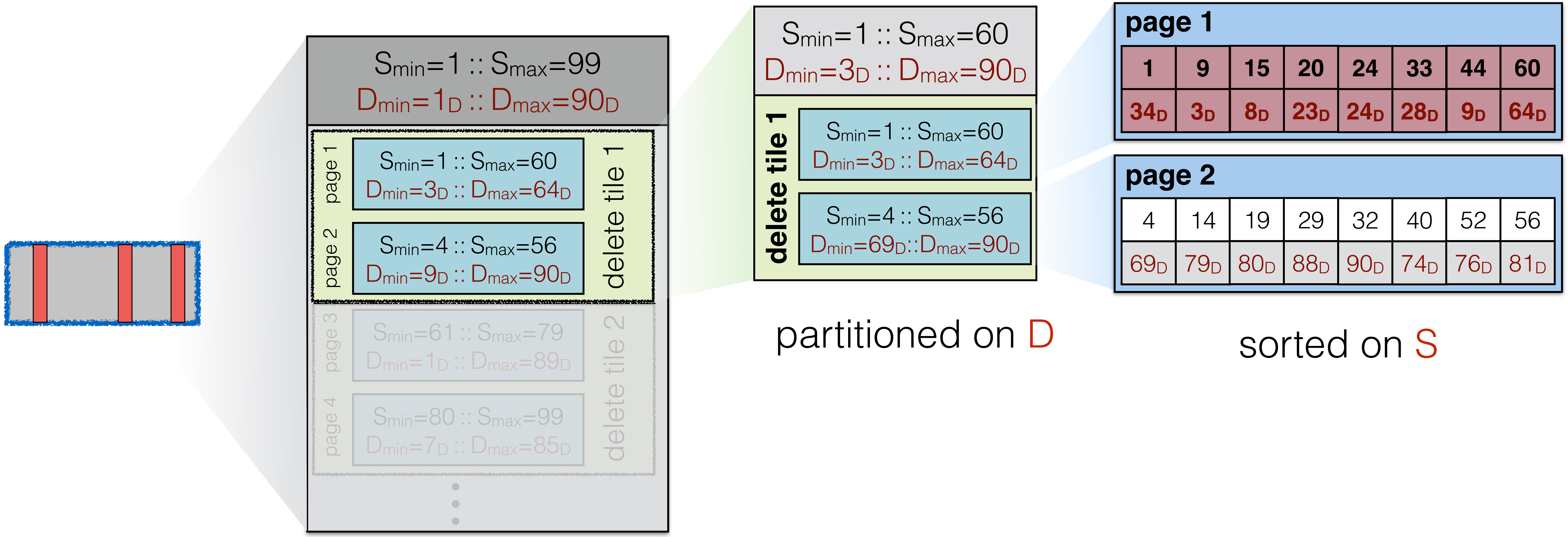
page 2

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

drop page

Key Weaving storage layout

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



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

delete tile 1

page 1: $S_{min}=1$ :: $S_{max}=60$ $D_{min}=3_D$ :: $D_{max}=64_D$

page 2: $S_{min}=4$ :: $S_{max}=56$ $D_{min}=9_D$ :: $D_{max}=90_D$

delete tile 2

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

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

partitioned on S

$S_{min}=1$ :: $S_{max}=60$
$D_{min}=3_D$ :: $D_{max}=90_D$

delete tile 1

$S_{min}=1$ :: $S_{max}=60$ $D_{min}=3_D$ :: $D_{max}=64_D$

$S_{min}=4$ :: $S_{max}=56$ $D_{min}=69_D$::$D_{max}=90_D$

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

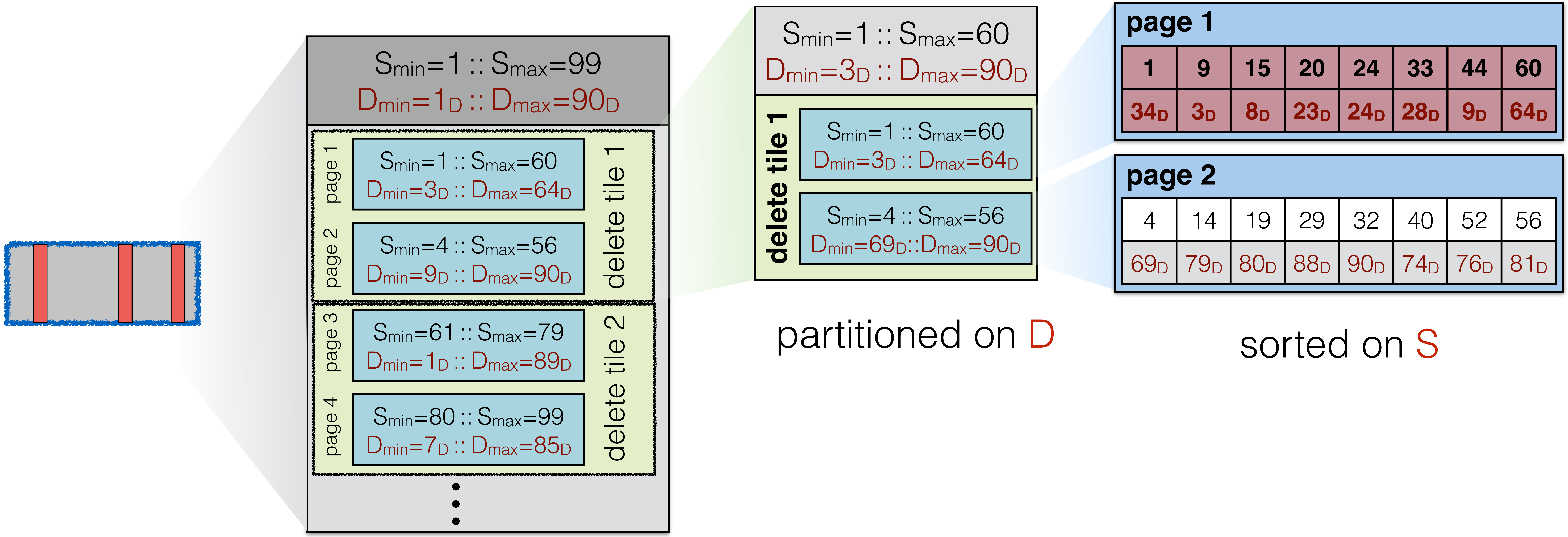
4	14	19	29	32	40	52	56
69$_D$	79$_D$	80$_D$	88$_D$	90$_D$	74$_D$	76$_D$	81$_D$

sorted on S

drop
page

Key Weaving storage layout

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



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

delete tile 1

page 1: $S_{min}=1 :: S_{max}=60$, $D_{min}=3_D :: D_{max}=64_D$

page 2: $S_{min}=4 :: S_{max}=56$, $D_{min}=9_D :: D_{max}=90_D$

delete tile 2

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

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

⋮

partitioned on S

$S_{min}=1 :: S_{max}=60$
$D_{min}=3_D :: D_{max}=90_D$

delete tile 1

$S_{min}=1 :: S_{max}=60$, $D_{min}=3_D :: D_{max}=64_D$

$S_{min}=4 :: S_{max}=56$, $D_{min}=69_D :: D_{max}=90_D$

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

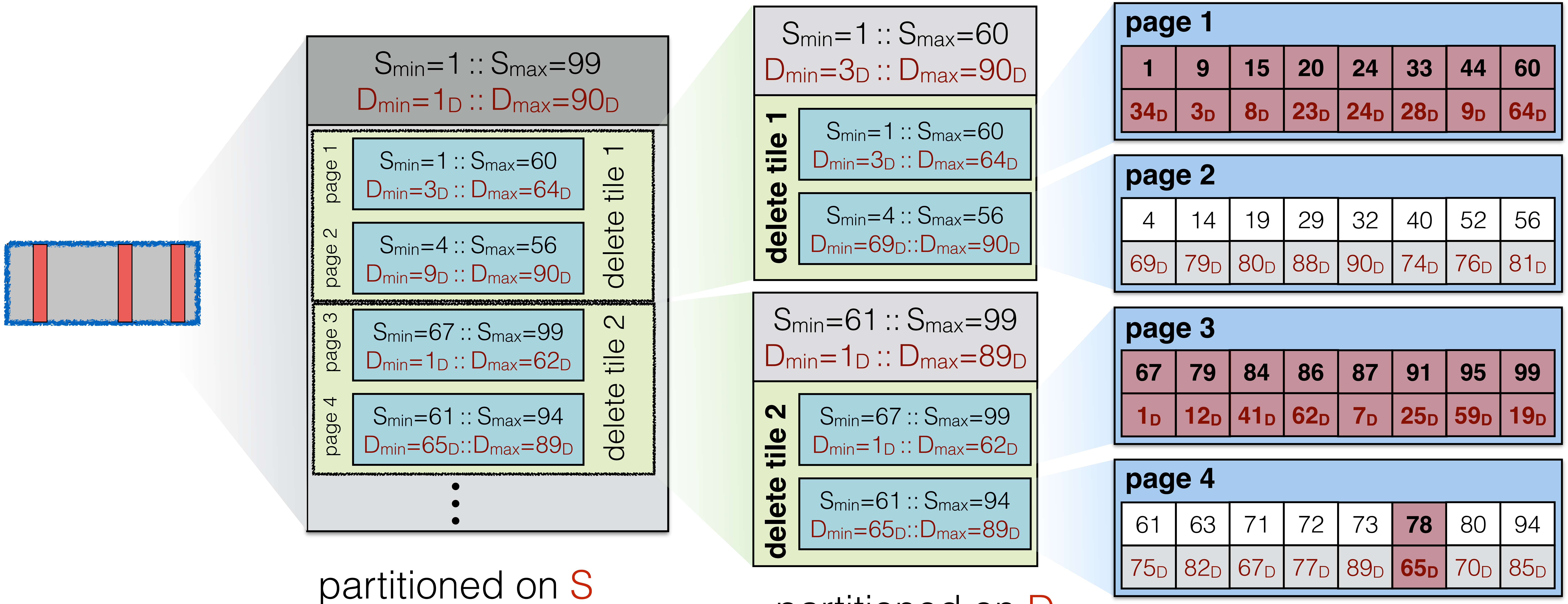
4	14	19	29	32	40	52	56
69$_D$	79$_D$	80$_D$	88$_D$	90$_D$	74$_D$	76$_D$	81$_D$

sorted on S

drop page

Key Weaving storage layout

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



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

delete tile 1

page 1: $S_{min}=1$ :: $S_{max}=60$, $D_{min}=3_D$ :: $D_{max}=64_D$

page 2: $S_{min}=4$ :: $S_{max}=56$, $D_{min}=9_D$ :: $D_{max}=90_D$

delete tile 2

page 3: $S_{min}=67$ :: $S_{max}=99$, $D_{min}=1_D$ :: $D_{max}=62_D$

page 4: $S_{min}=61$ :: $S_{max}=94$, $D_{min}=65_D$::$D_{max}=89_D$

⋮

partitioned on S

$S_{min}=1$ :: $S_{max}=60$
$D_{min}=3_D$ :: $D_{max}=90_D$

delete tile 1

$S_{min}=1$ :: $S_{max}=60$, $D_{min}=3_D$ :: $D_{max}=64_D$

$S_{min}=4$ :: $S_{max}=56$, $D_{min}=69_D$::$D_{max}=90_D$

$S_{min}=61$ :: $S_{max}=99$
$D_{min}=1_D$ :: $D_{max}=89_D$

delete tile 2

$S_{min}=67$ :: $S_{max}=99$, $D_{min}=1_D$ :: $D_{max}=62_D$

$S_{min}=61$ :: $S_{max}=94$, $D_{min}=65_D$::$D_{max}=89_D$

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**

drop page

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**

drop page

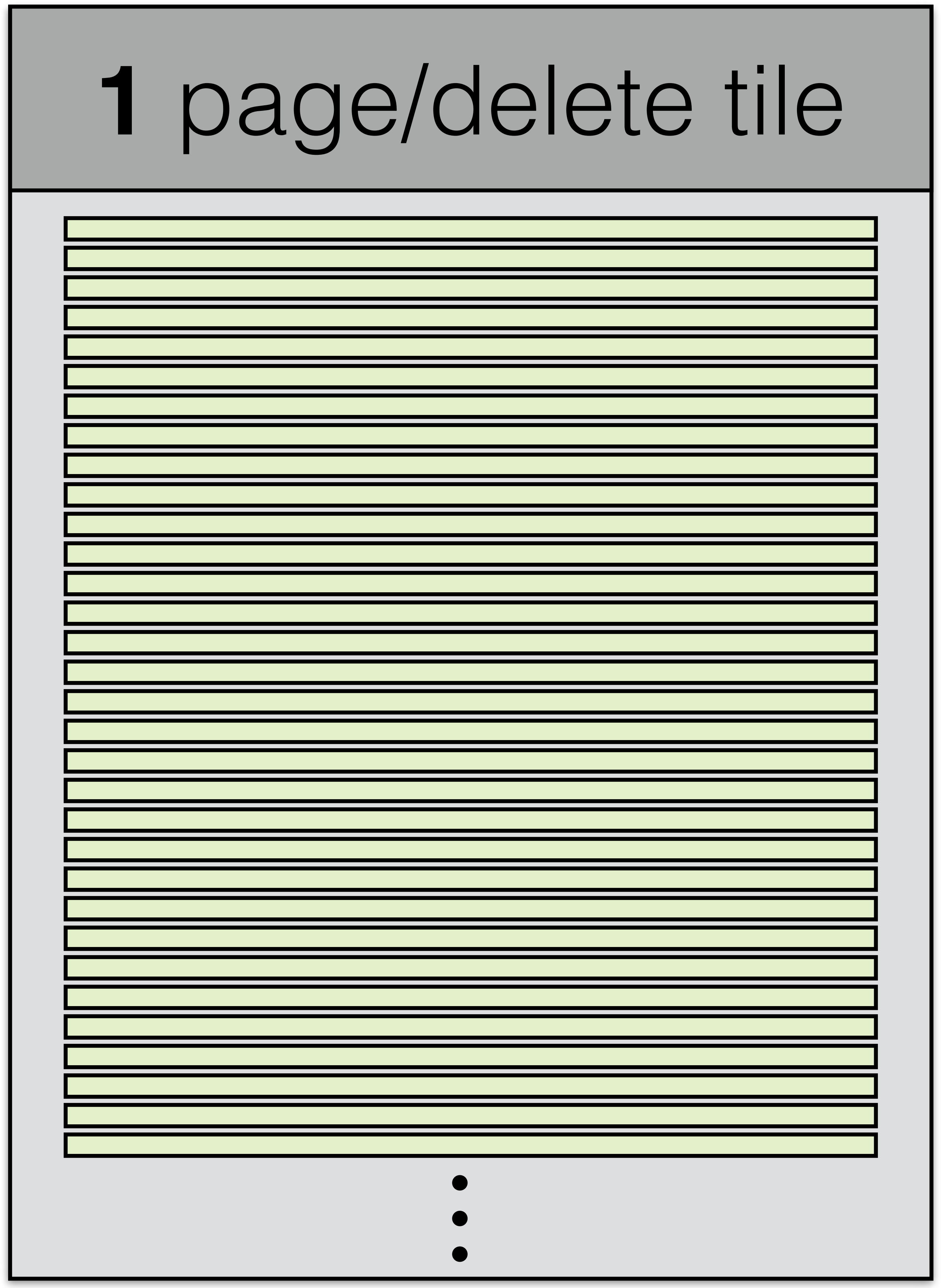
page 4

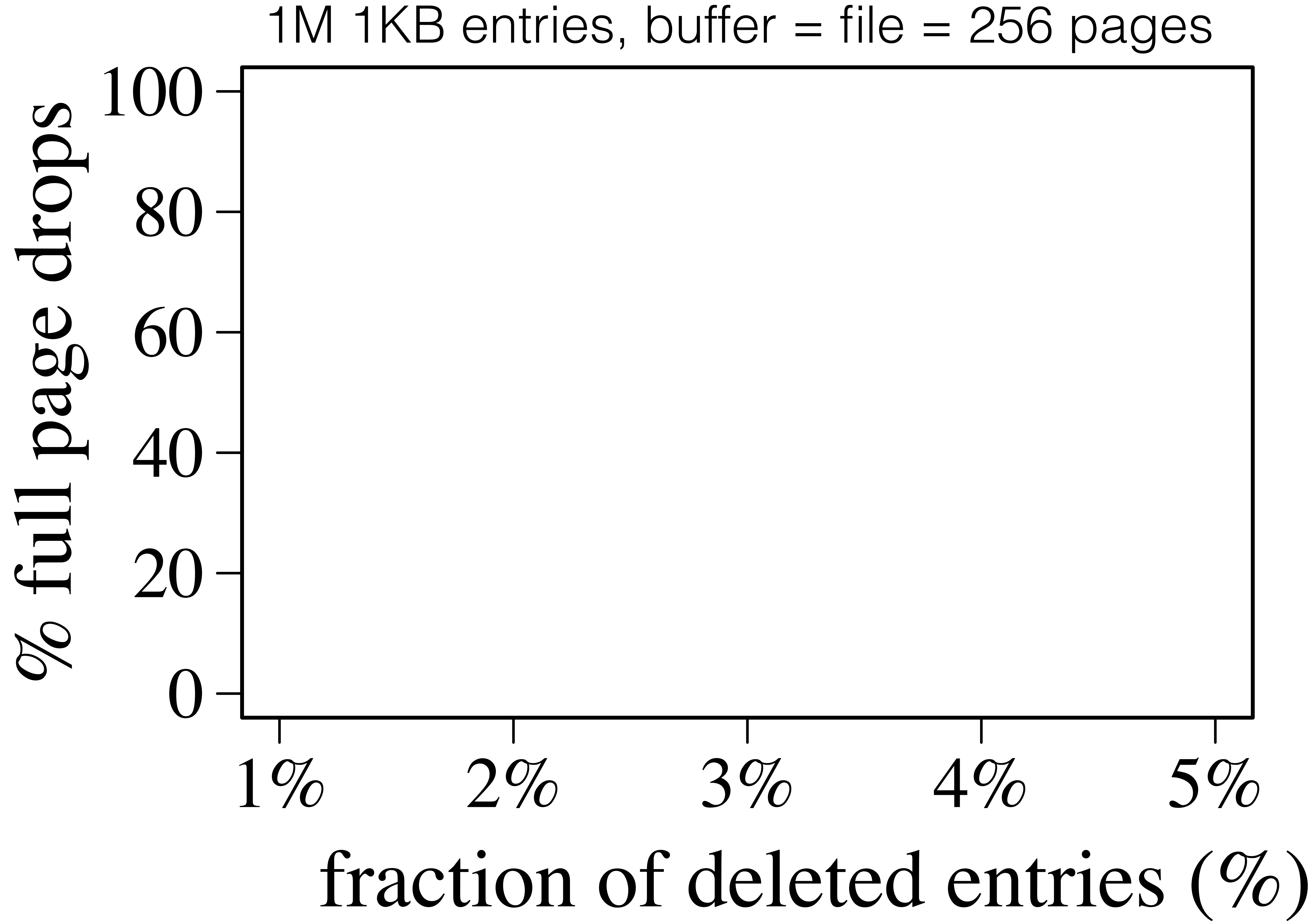
61	63	71	72	73	**78**	80	94
75_D	82_D	67_D	77_D	89_D	**65_D**	70_D	85_D

1 I/O

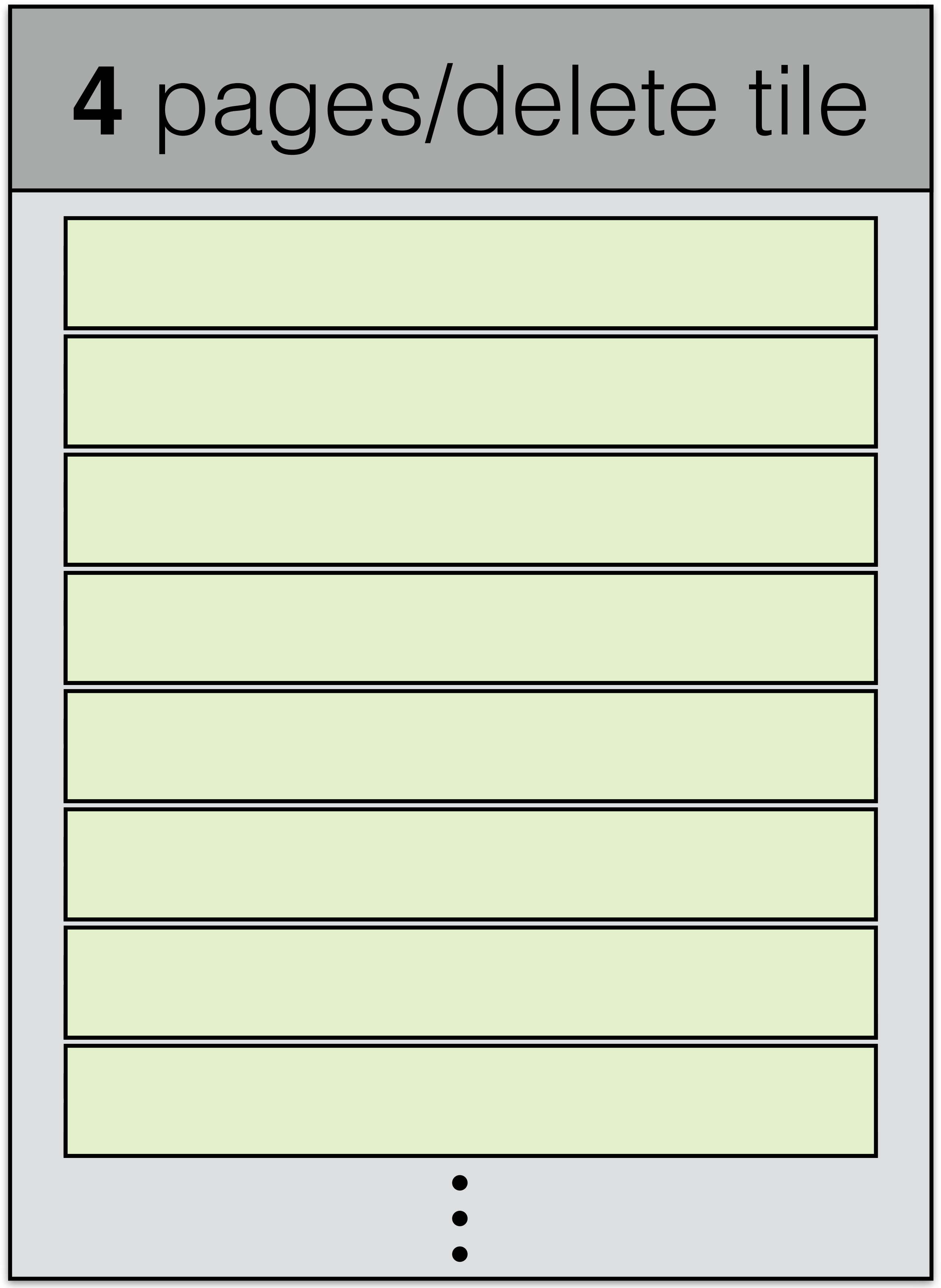
sorted on S

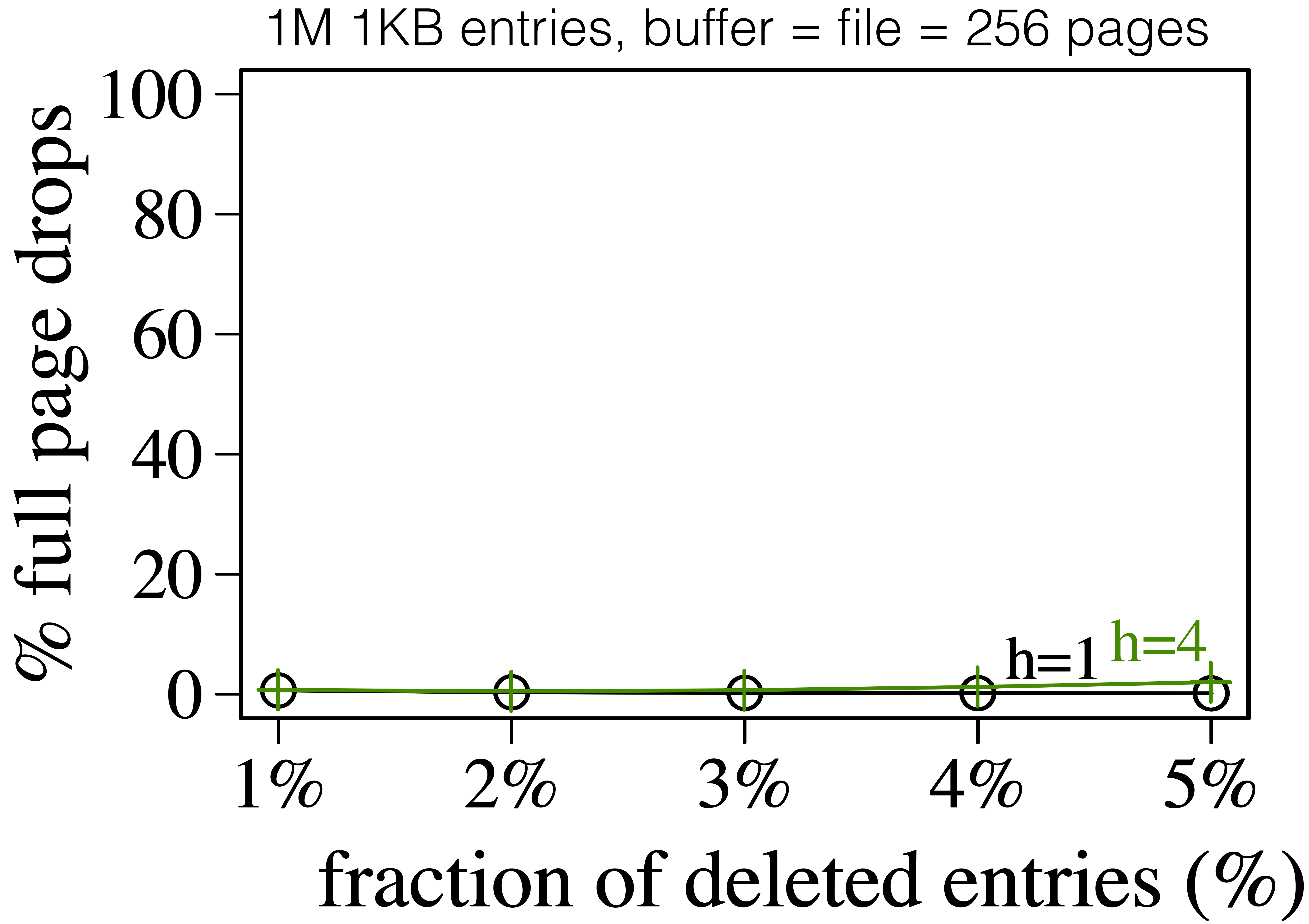
Key Weaving storage layout



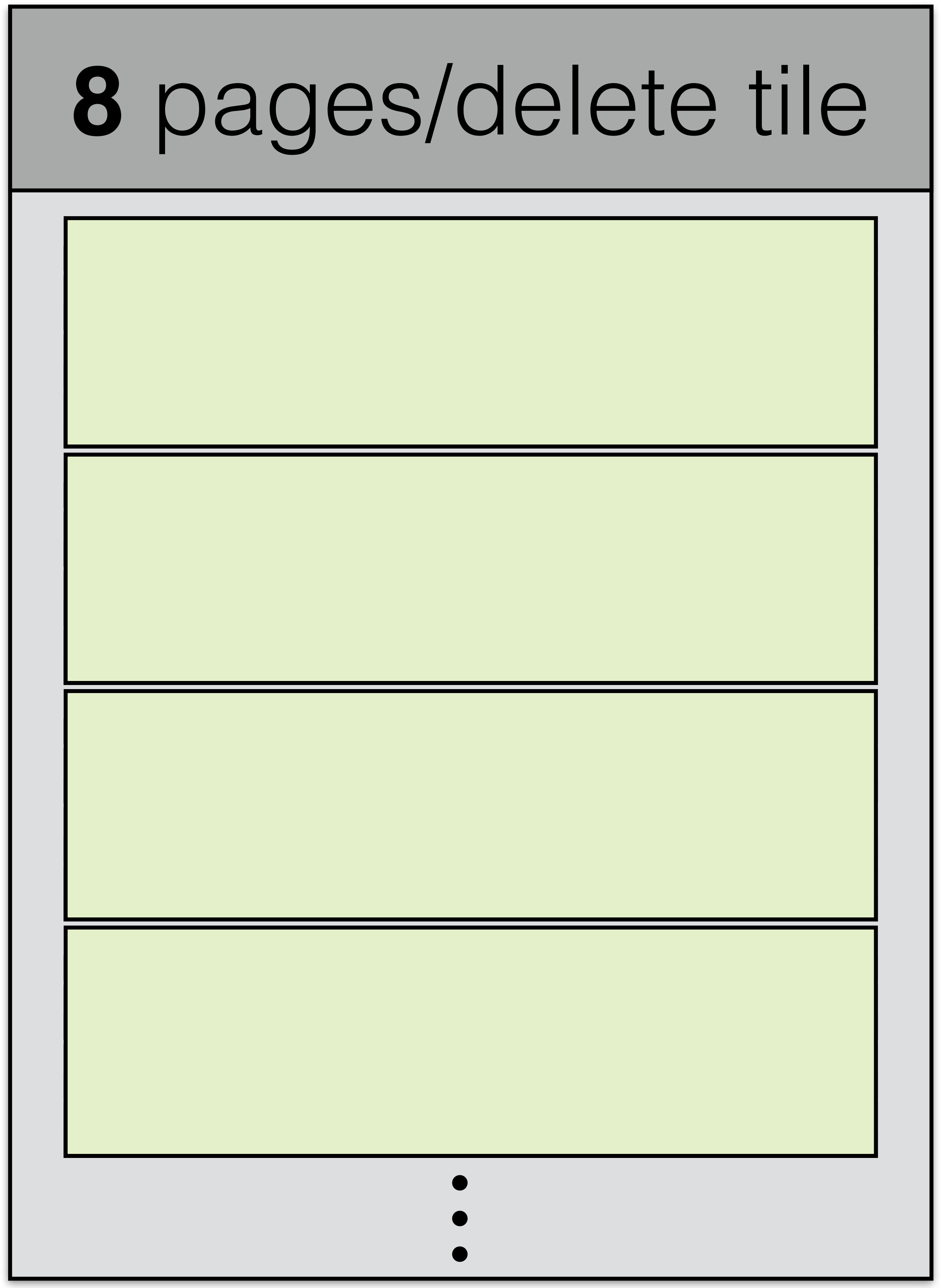


Key Weaving storage layout

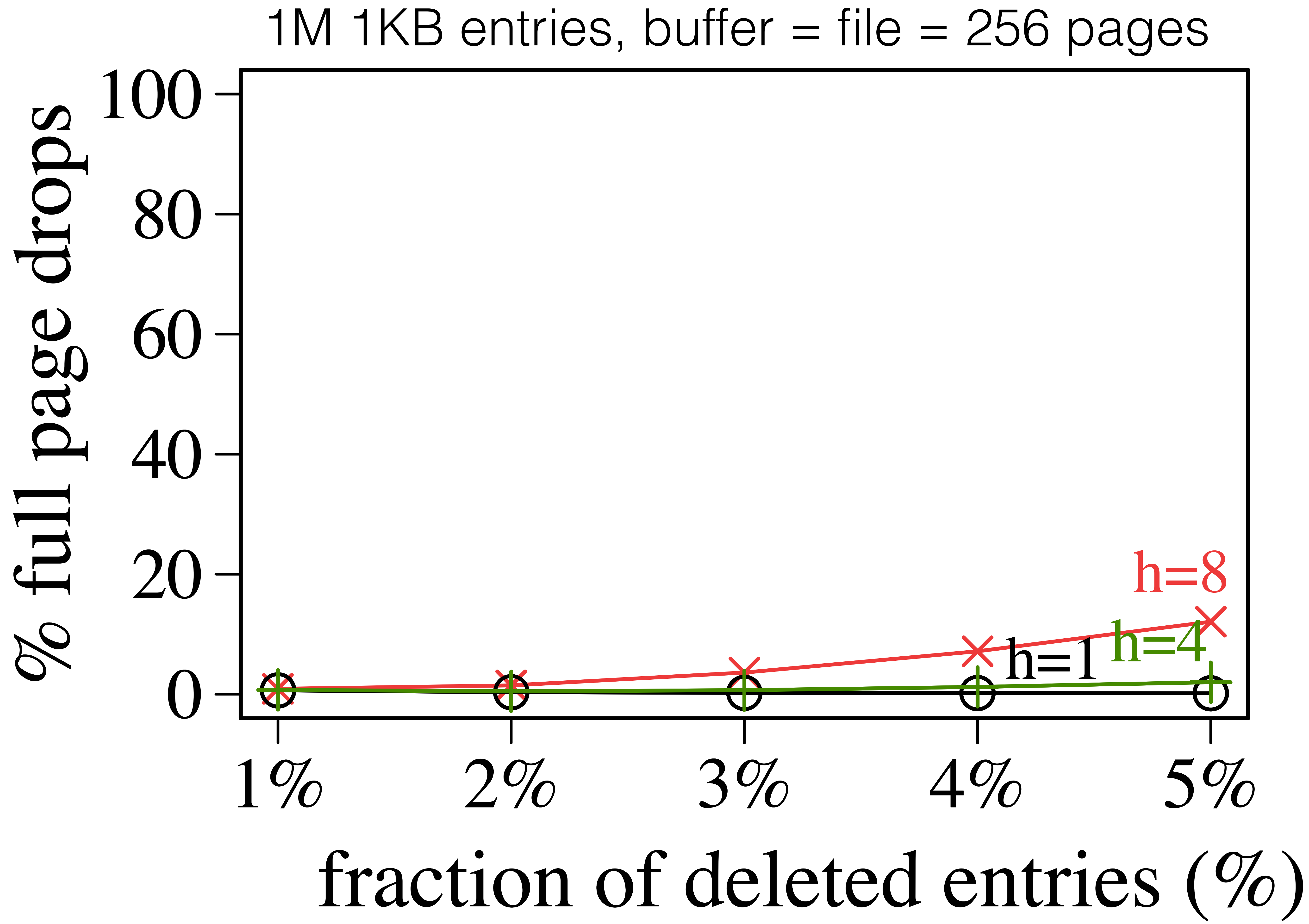




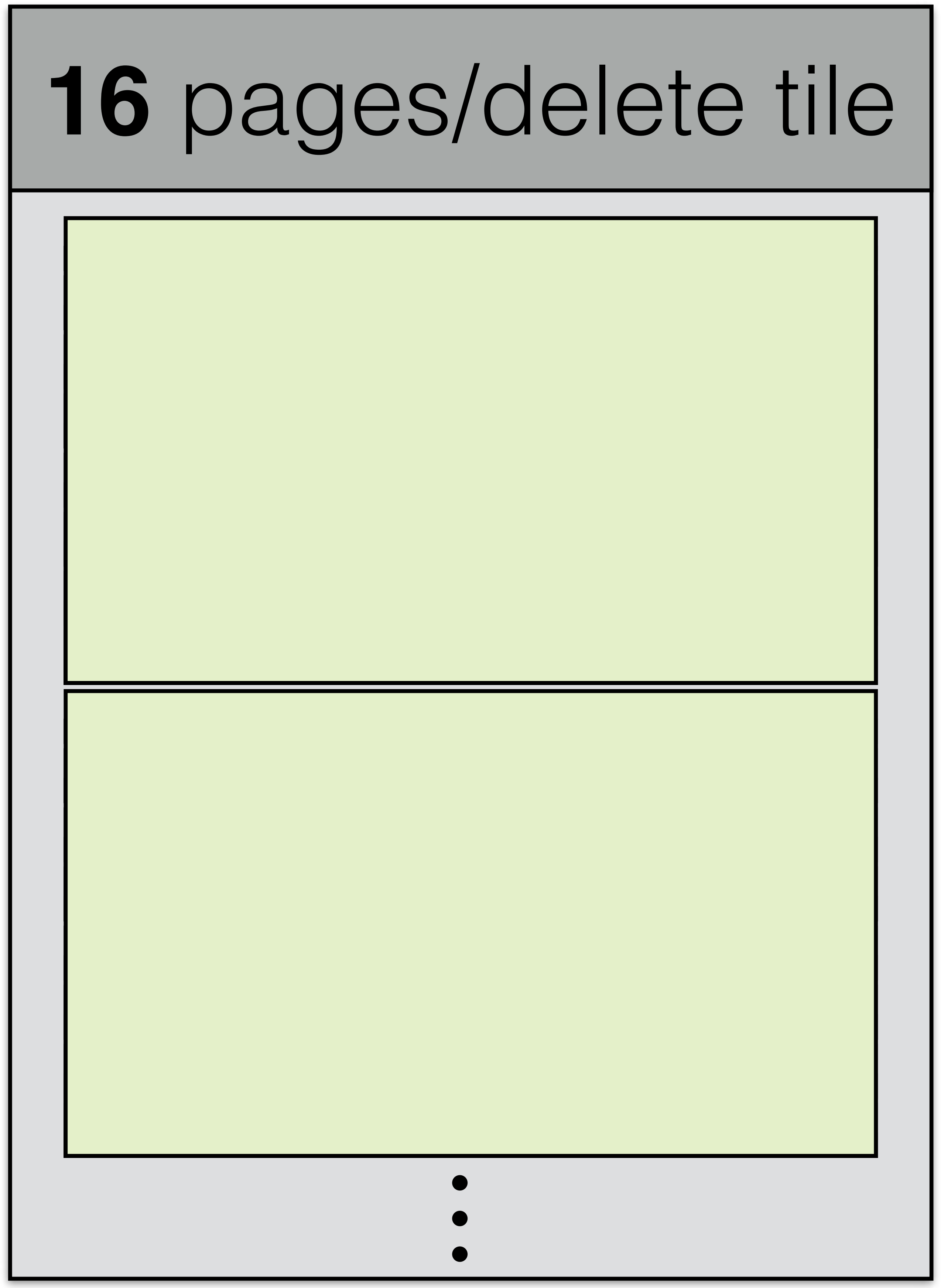
Key Weaving storage layout

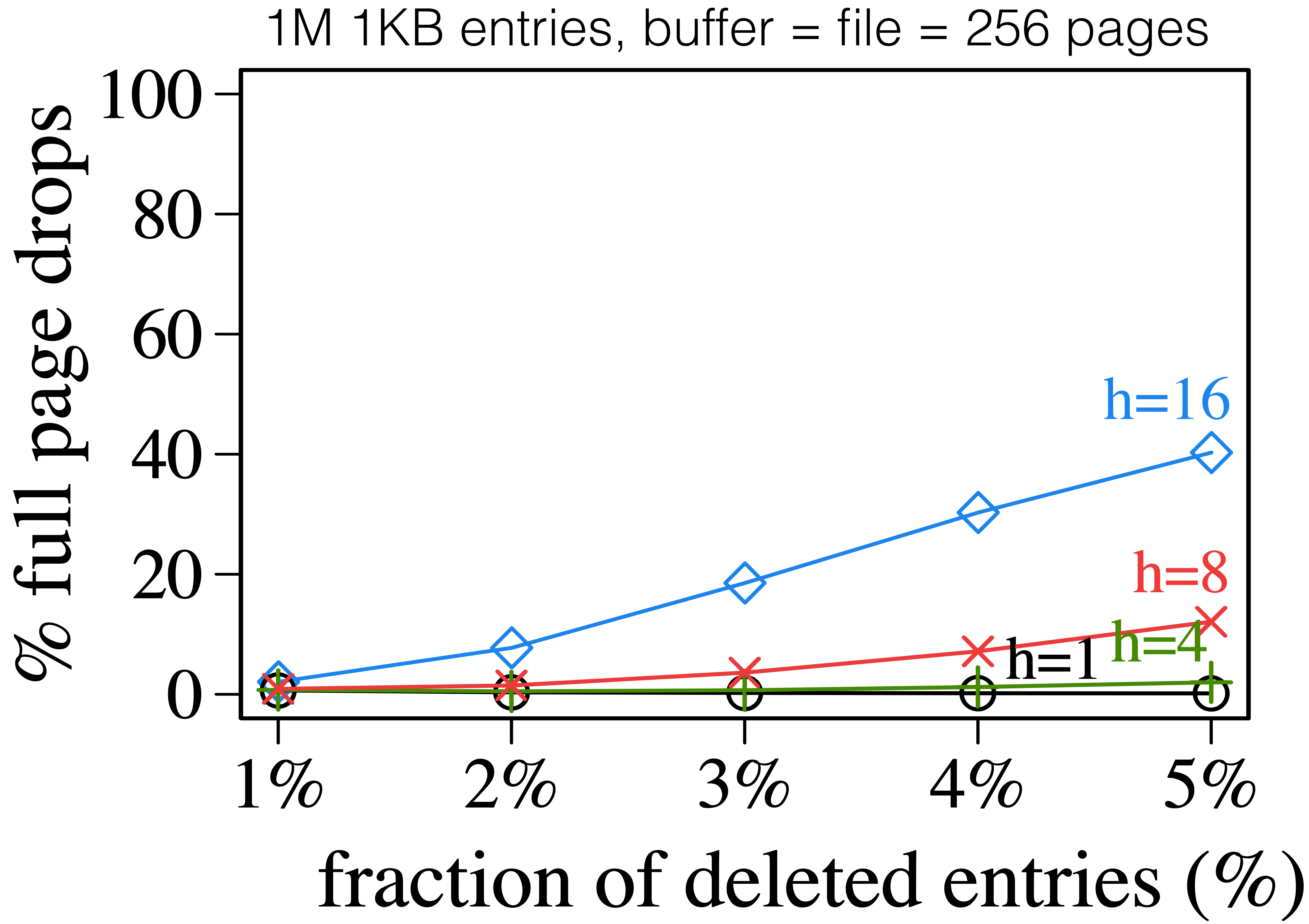


8 pages/delete tile

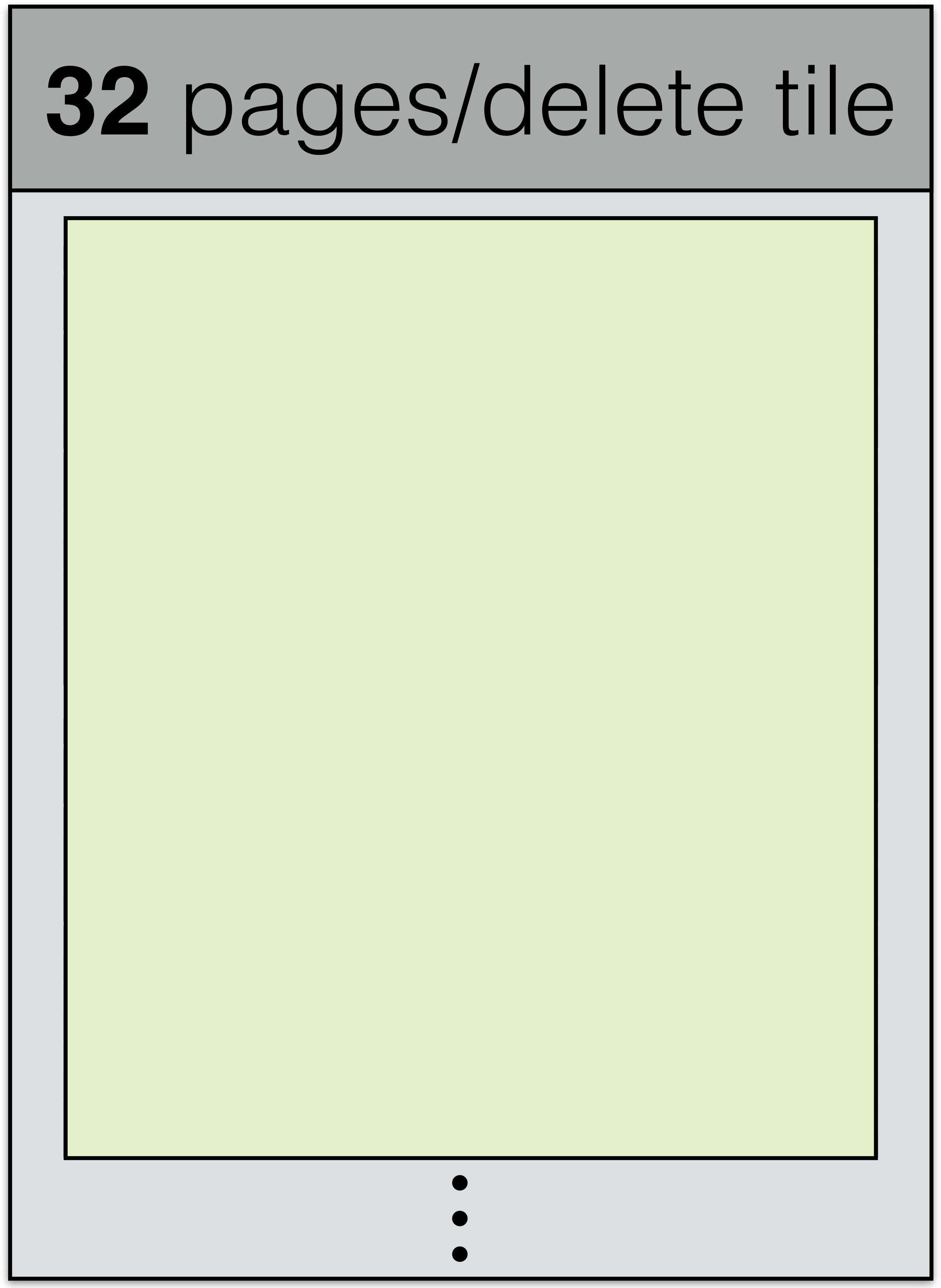


Key Weaving storage layout

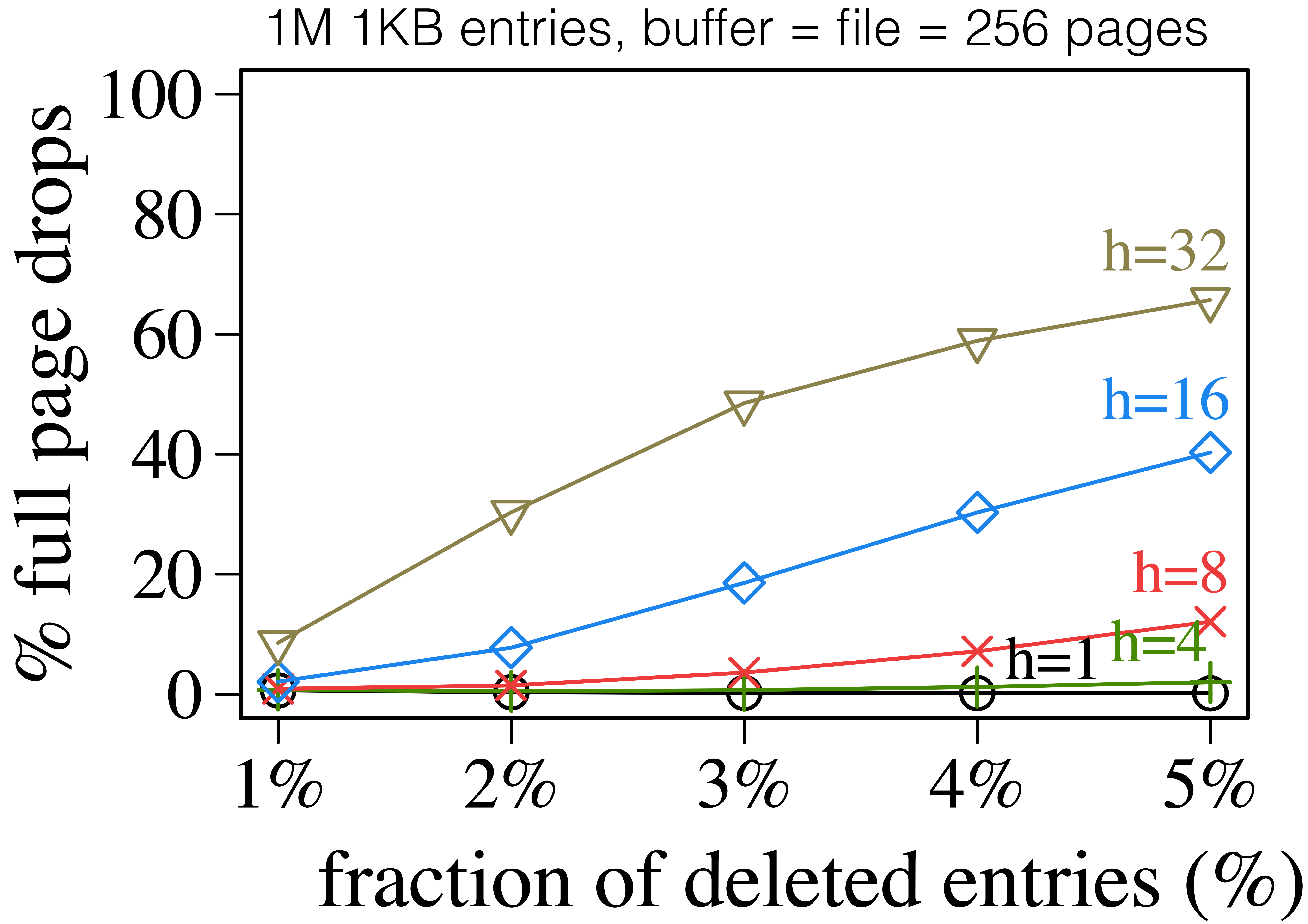




Key Weaving storage layout



32 pages/delete tile



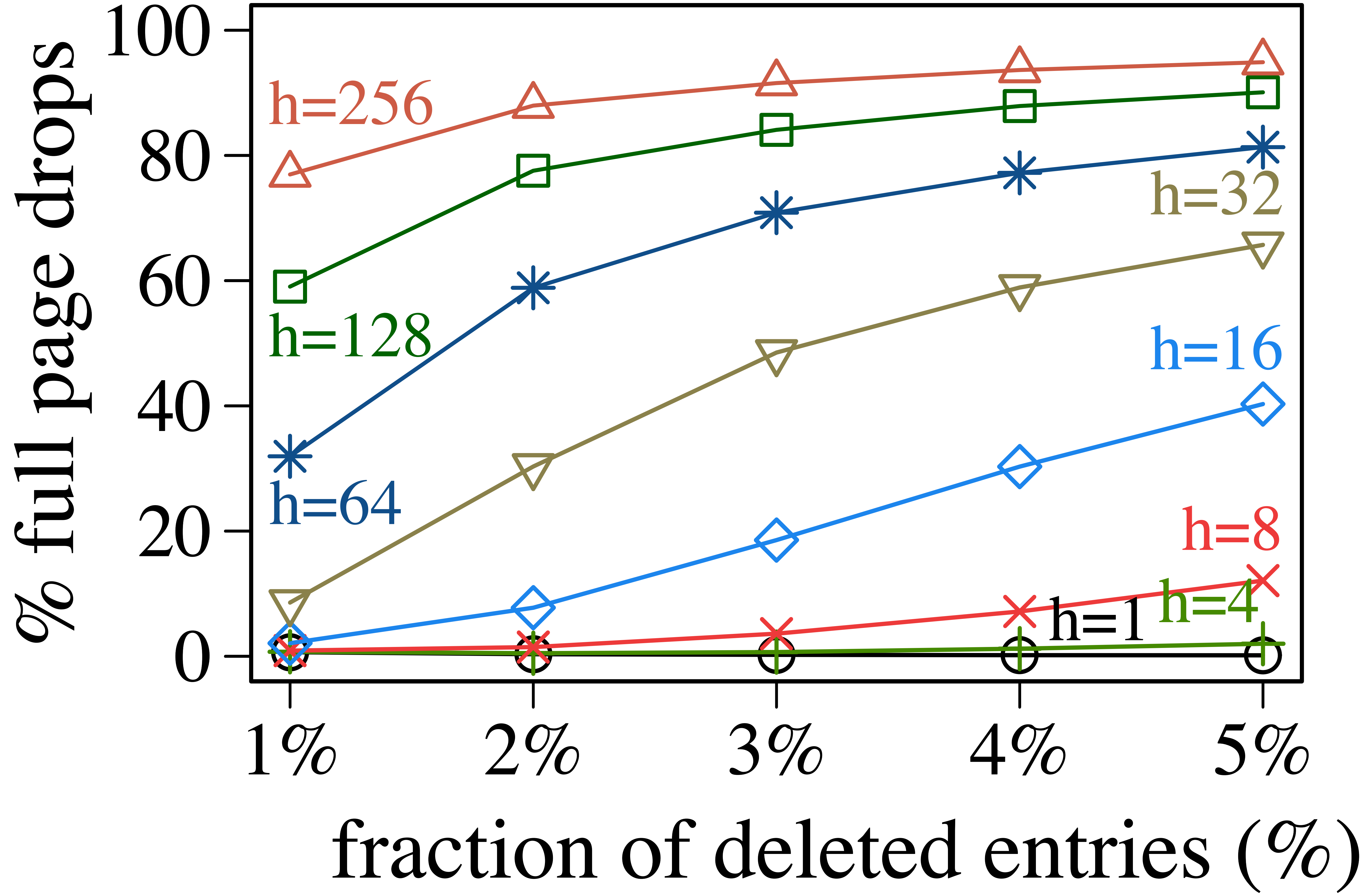
1M 1KB entries, buffer = file = 256 pages

Key Weaving storage layout

reduced latency spikes ✓

full page drops reduces superfluous I/Os ✓

1M 1KB entries, buffer = file = 256 pages

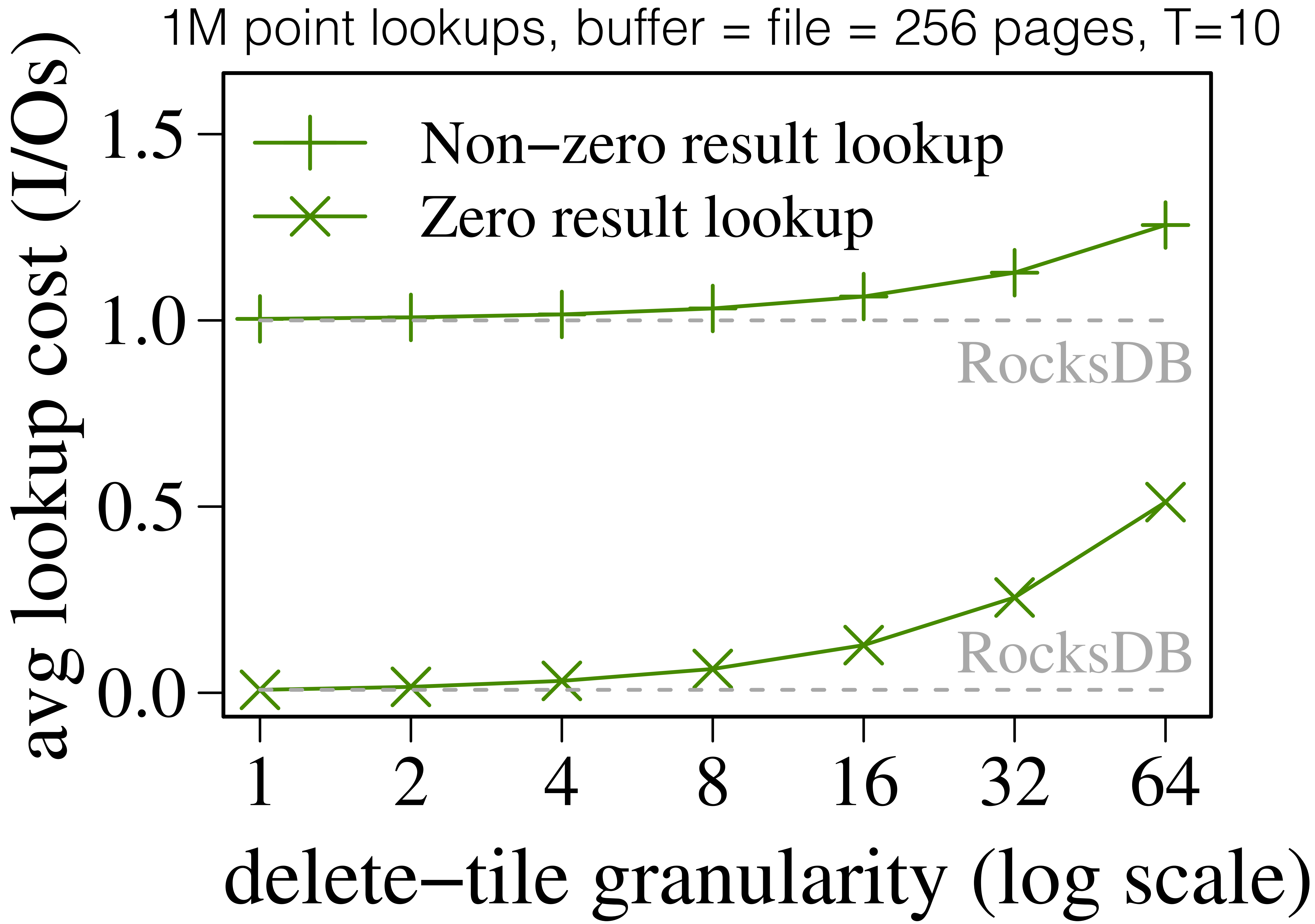


Key Weaving storage layout

higher lookup cost ↑

reduced latency spikes ✓

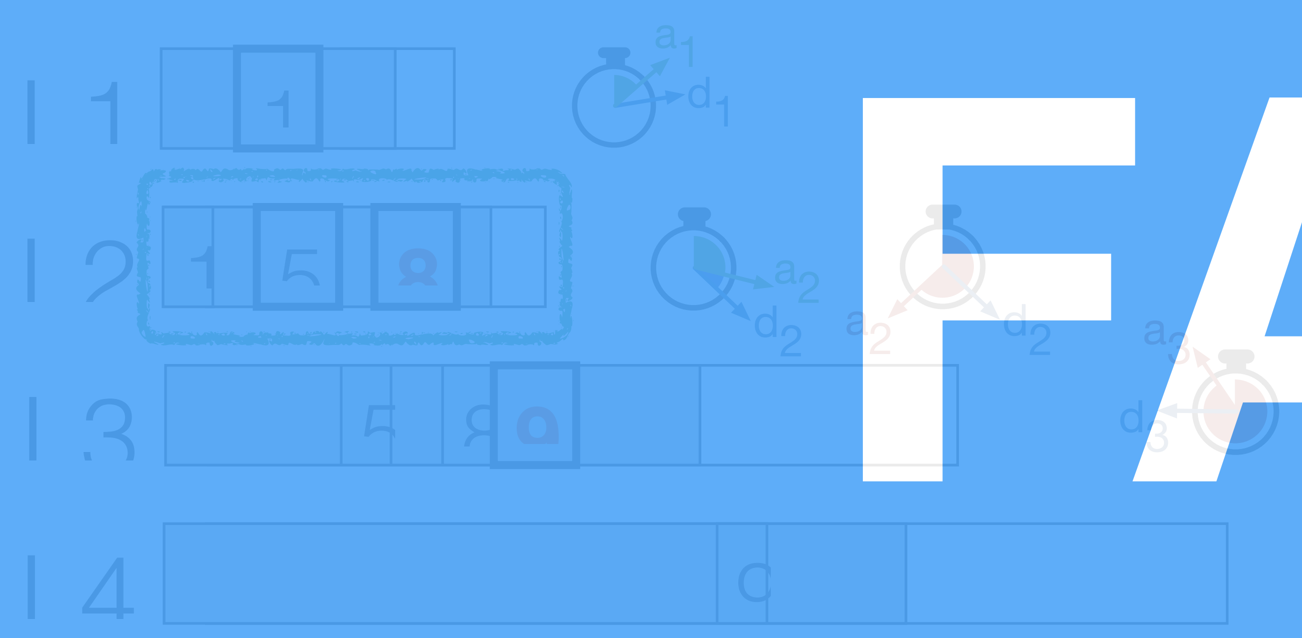
full page drops reduces superfluous I/Os ✓



the solution

FADE

FAst DElete



- amortized write ●
- reduced space ✓
- improved read ✓
- timely delete persistence ✓

KiWi

- higher lookup cost ↑
- reduced latency spikes ✓
- 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

CS561: Data Systems Architectures

class 8

Efficient Deletes in Log-Structured Key-Value Storage

Prof. Manos Athanassoulis