# Enabling Efficient Deletes in Log-Structured KV-Stores


**Subhadeep Sarkar**

# Enabling Efficient Deletes in Log-Structured KV-Stores


**Subhadeep Sarkar**

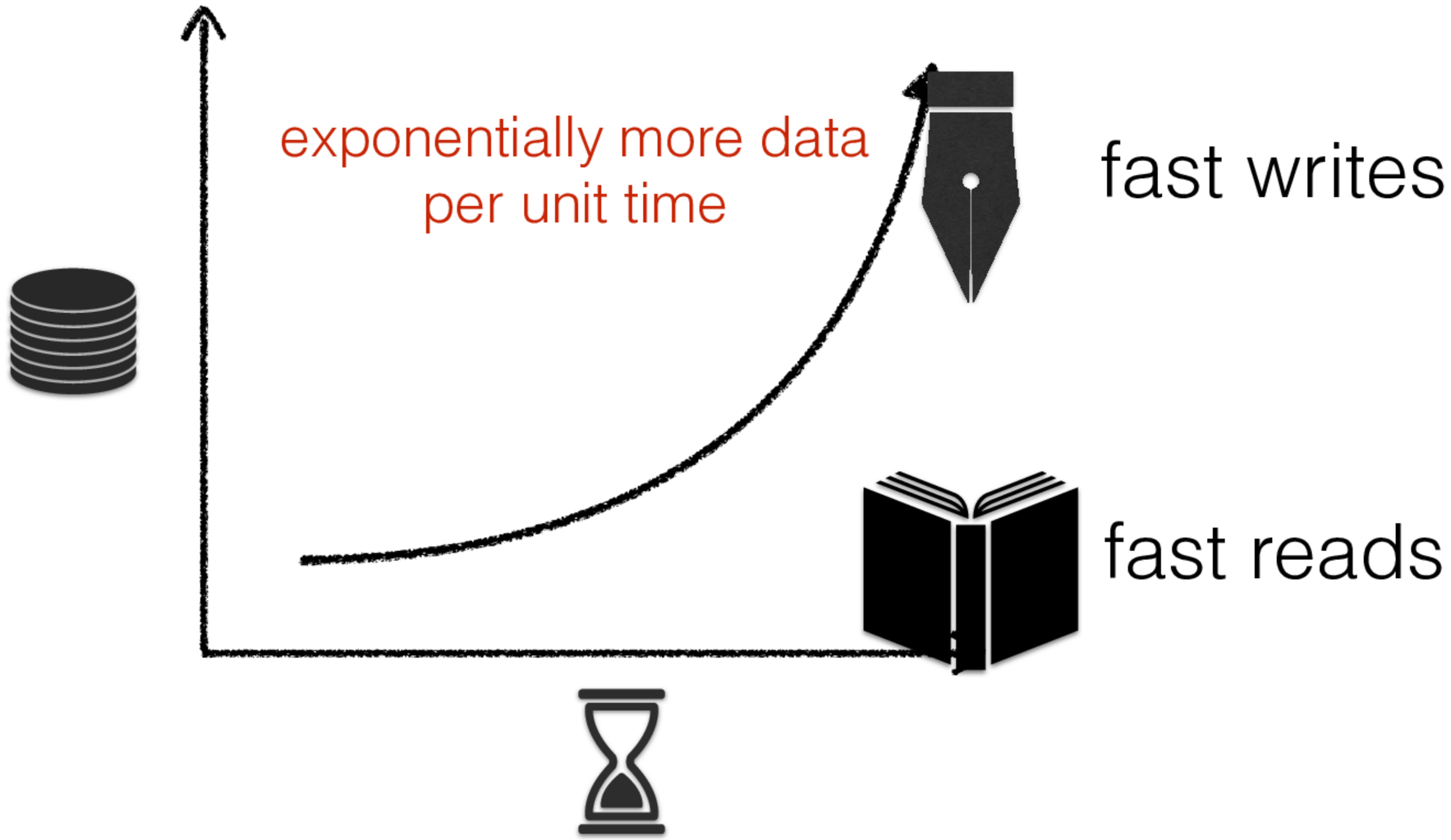
# Enabling Efficient Deletes in Log-Structured KV-Stores

# Enabling Efficient Deletes in Log-Structured KV-Stores

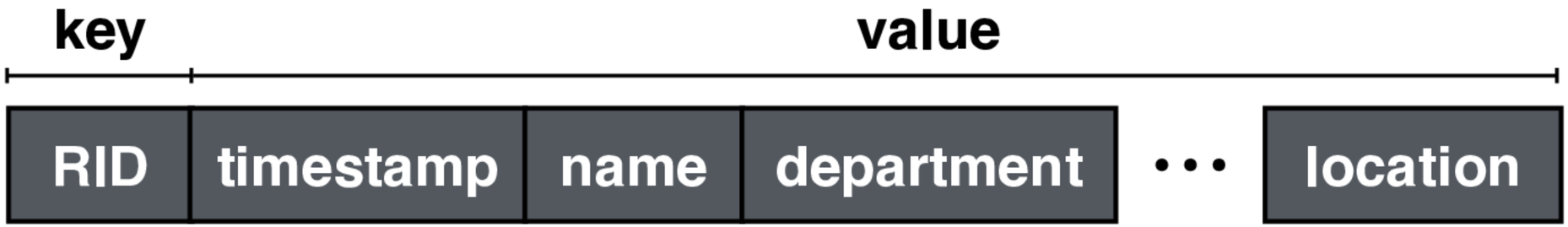
# Enabling Efficient Deletes in Log-Structured KV-Stores

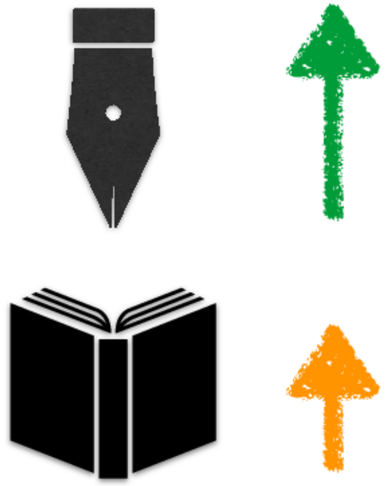
# Enabling Efficient Deletes in Log-Structured KV-Stores

**Lethe** [ \ˈlē-thē\ ]
n: the goddess of forgetfulness

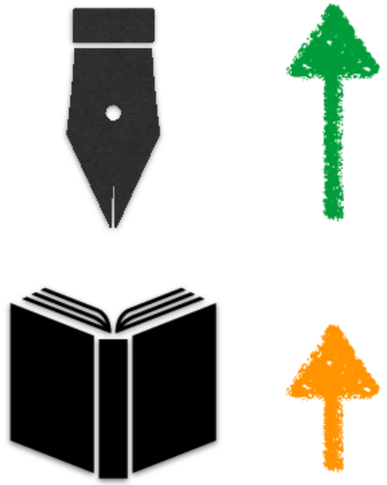

exponentially more data
per unit time
fast writes
fast reads

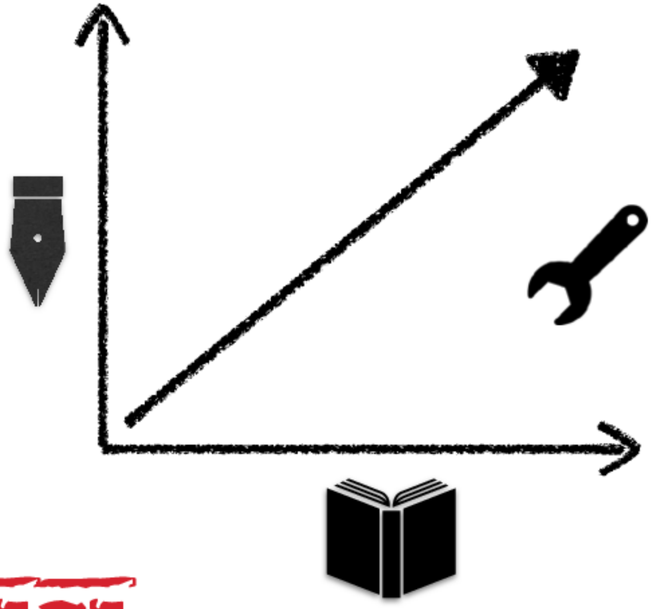
**key-value** pairs


key
value
RID
timestamp
name
department
• • •
location

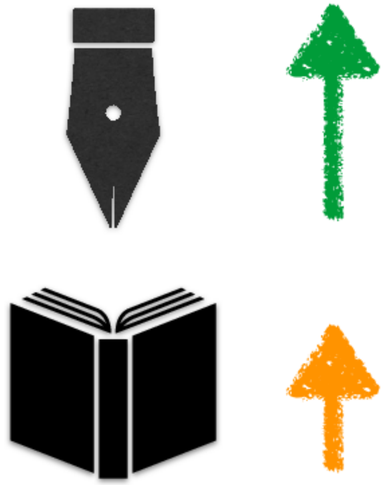


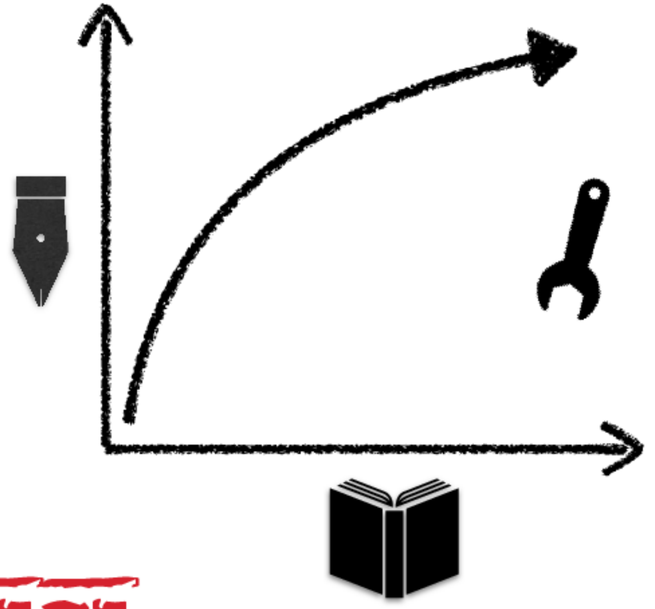
# LSM-TREE



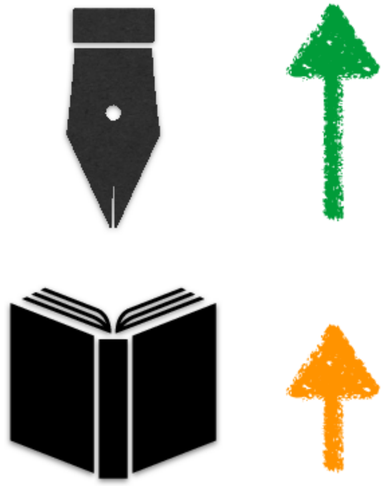


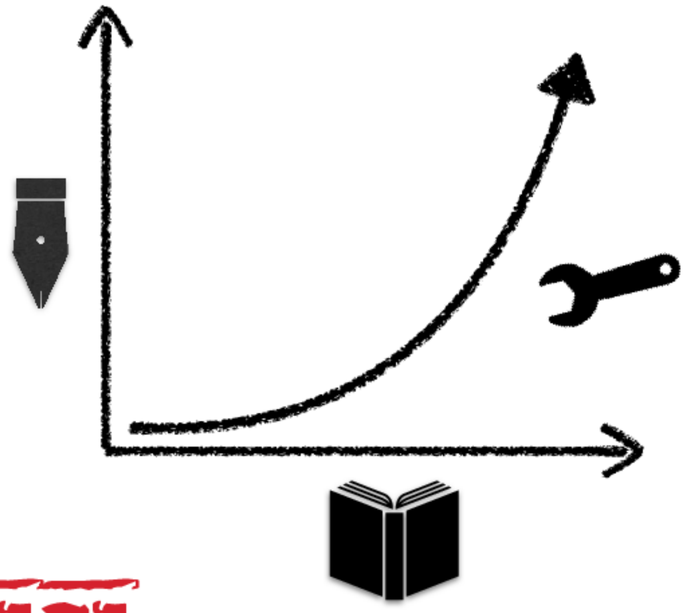
# LSM-TREE





# LSM-TREE





# LSM-TREE

cassandra

levelDB

RocksDB

WT

accumulo™

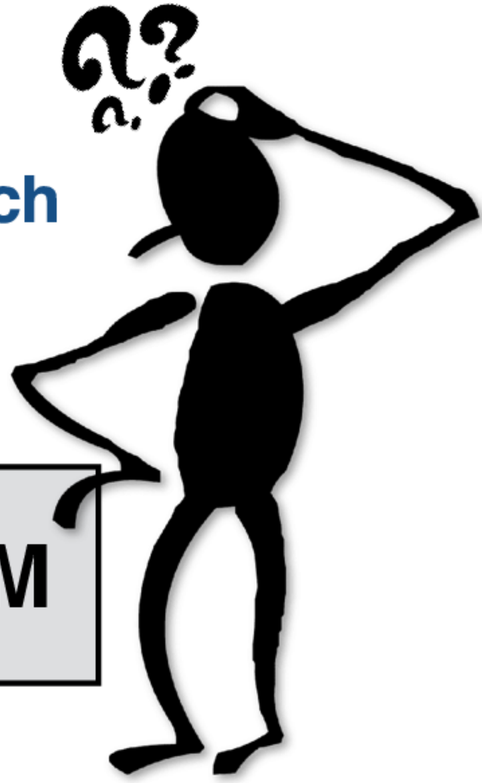
DynamoDB

SQLite

APACHE HBASE

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

**TechCrunch**
Feb '19

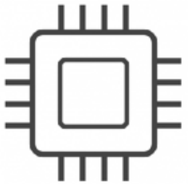


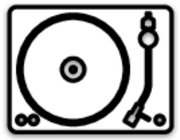
**Small Datum**
Jan '20

**Deletes are fast and slow in an LSM**

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

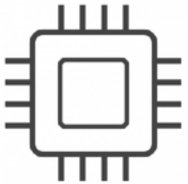
# log-structured merge-tree

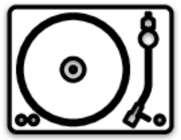




buffer

# log-structured merge-tree

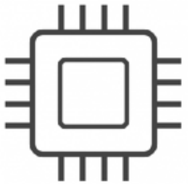


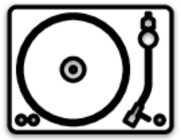


buffer

| 2 | 6 | 1 | 4 |
|---|---|---|---|

# log-structured merge-tree

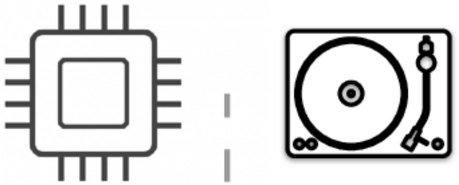




buffer

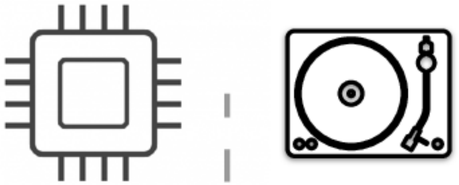
| 1 | 2 | 4 | 6 |
|---|---|---|---|

# log-structured merge-tree



buffer

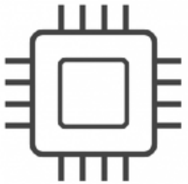
# log-structured merge-tree

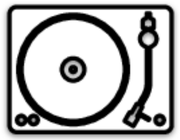


buffer

L1

# log-structured merge-tree

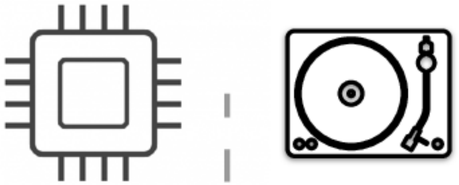




buffer

L1

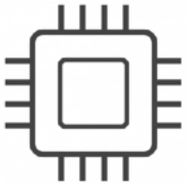
# log-structured merge-tree

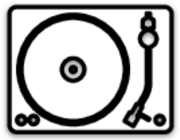


buffer

L1

# log-structured merge-tree





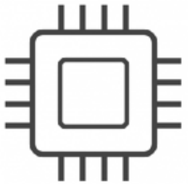
buffer

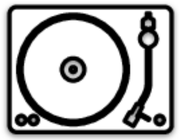
L1

L2

**compaction**

# log-structured merge-tree





buffer

L1

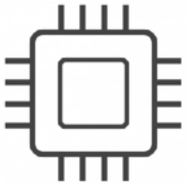
L2

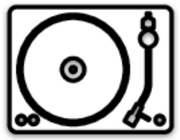
L3

**size ratio = T**

exponentially larger capacity

# log-structured merge-tree





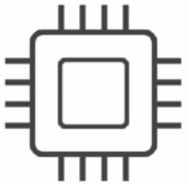
buffer

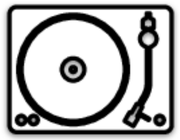
L1

L2

L3

# log-structured merge-tree





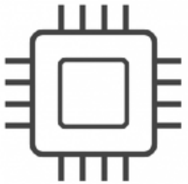
buffer

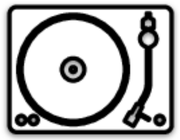
L1

L2

L3

# log-structured merge-tree





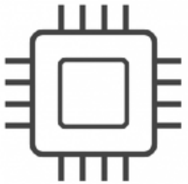
buffer

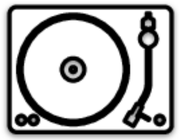
L1

L2

L3

# log-structured merge-tree





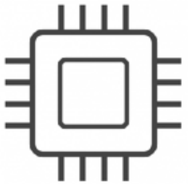
buffer

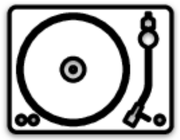
L1

L2

L3

# log-structured merge-tree





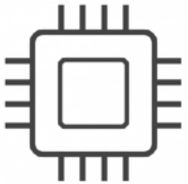
buffer

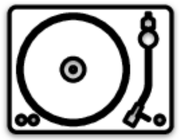
L1

L2

L3

# log-structured merge-tree





buffer 

L1

L2

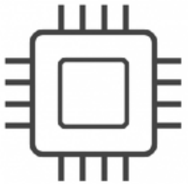
L3

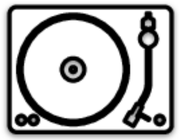
L4

burst of I/Os

prolonged write stalls

# log-structured merge-tree





buffer 

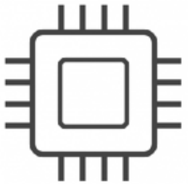
**partial compaction**

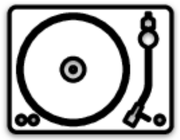
L1 

L2 

L3 

# log-structured merge-tree





buffer

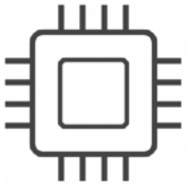
**partial compaction**

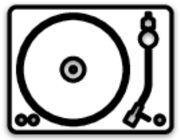
L1

L2

L3

# log-structured merge-tree





buffer

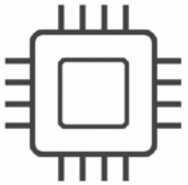
**partial compaction**

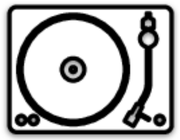
L1

L2

L3

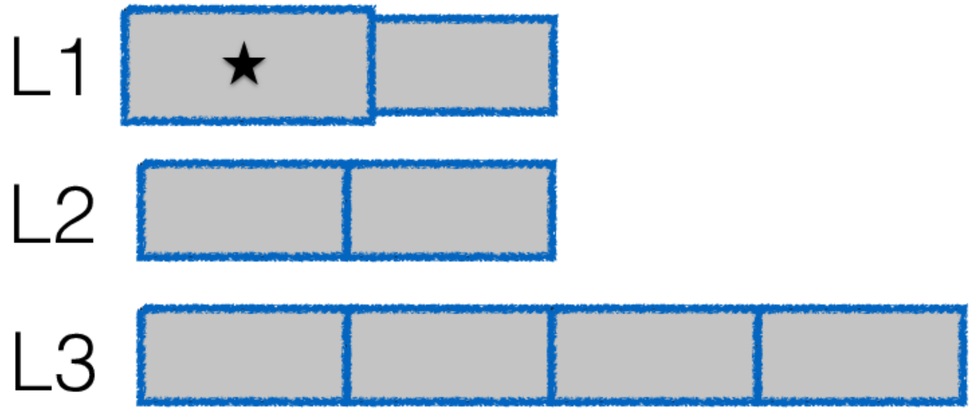
# log-structured merge-tree



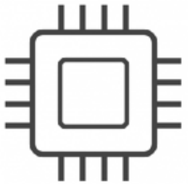


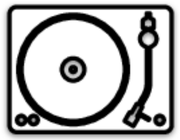
buffer

**partial compaction**



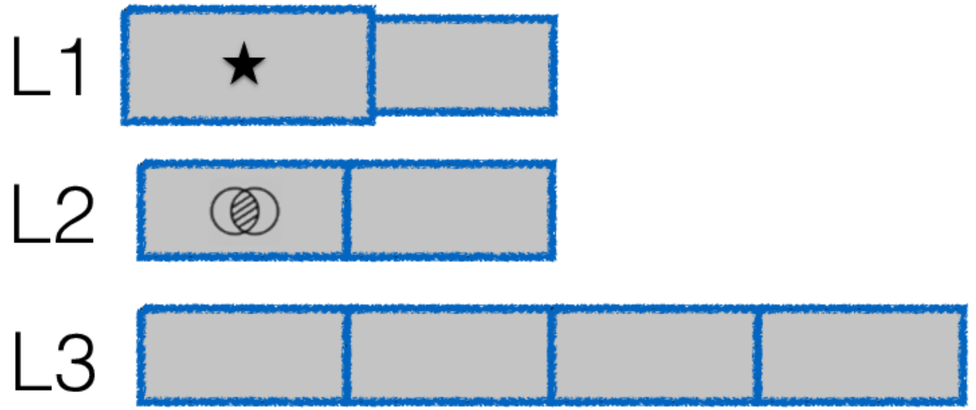
# log-structured merge-tree



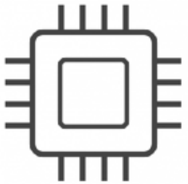


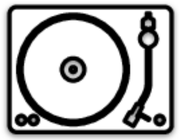
buffer

**partial compaction**



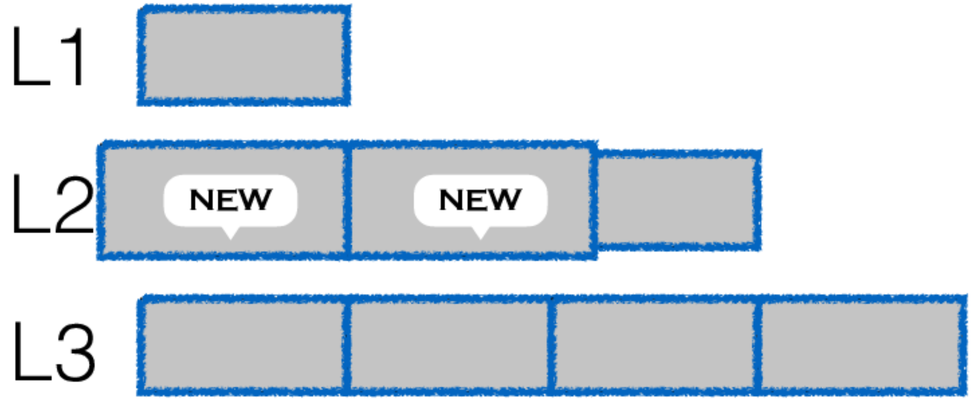
# log-structured merge-tree



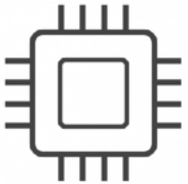


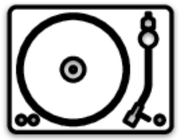
buffer

**partial compaction**



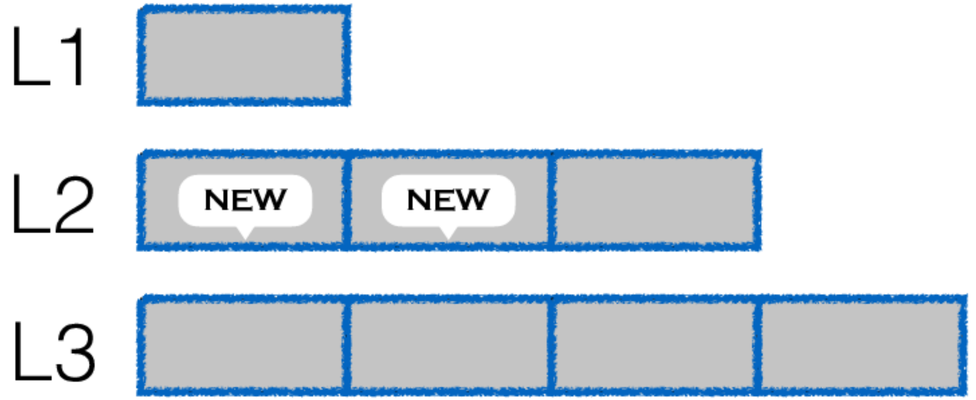
# log-structured merge-tree



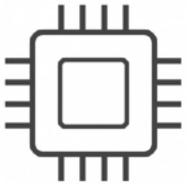


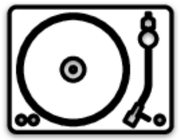
buffer

**partial compaction**



# log-structured merge-tree





buffer

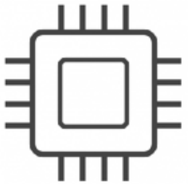
**partial compaction**

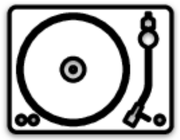
L1

L2

L3

# log-structured merge-tree





buffer

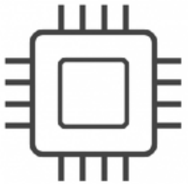
**partial compaction**

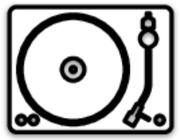
L1

L2

L3

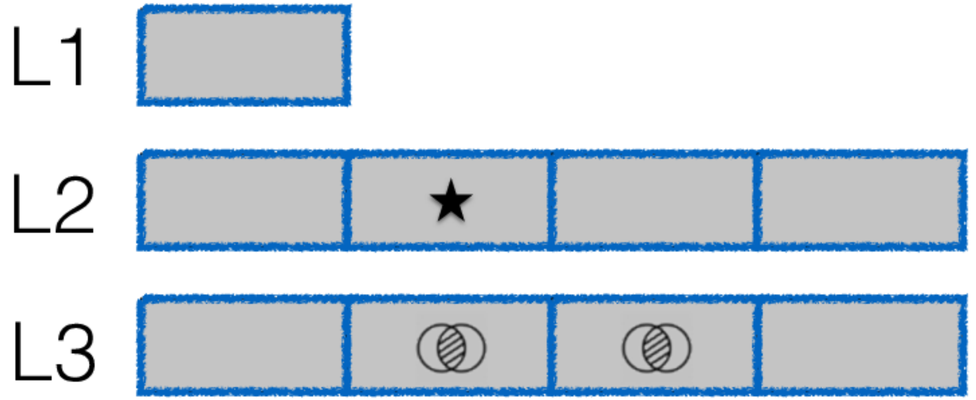
# log-structured merge-tree



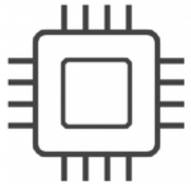


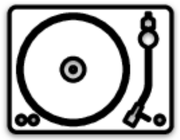
buffer 

**partial compaction**



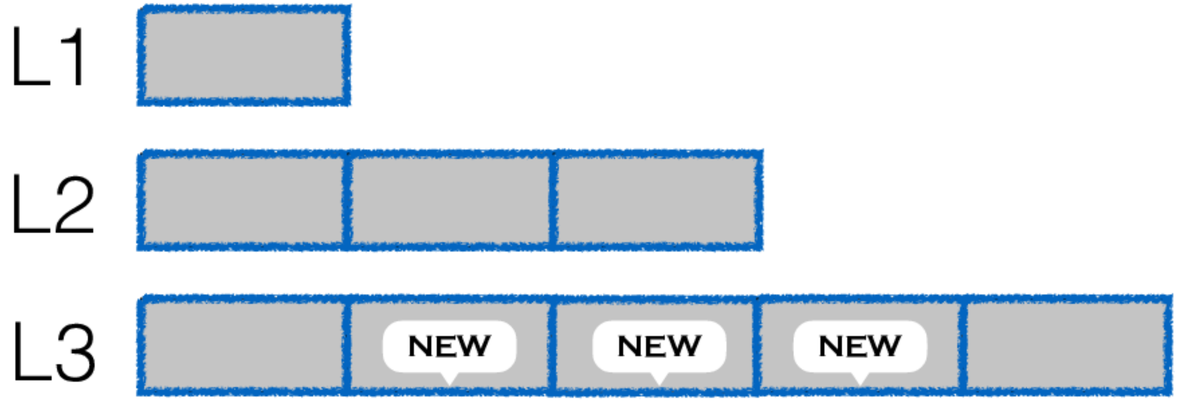
# log-structured merge-tree





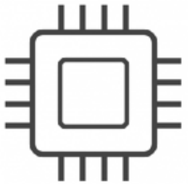
buffer

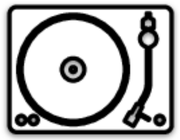
**partial compaction**




amortized compaction cost

# log-structured merge-tree





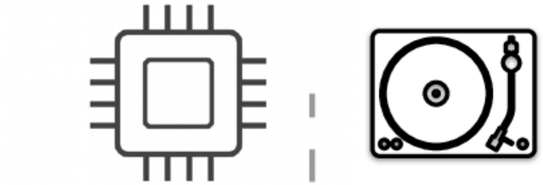
buffer

L1

L2

L3

# log-structured merge-tree



fence pointers

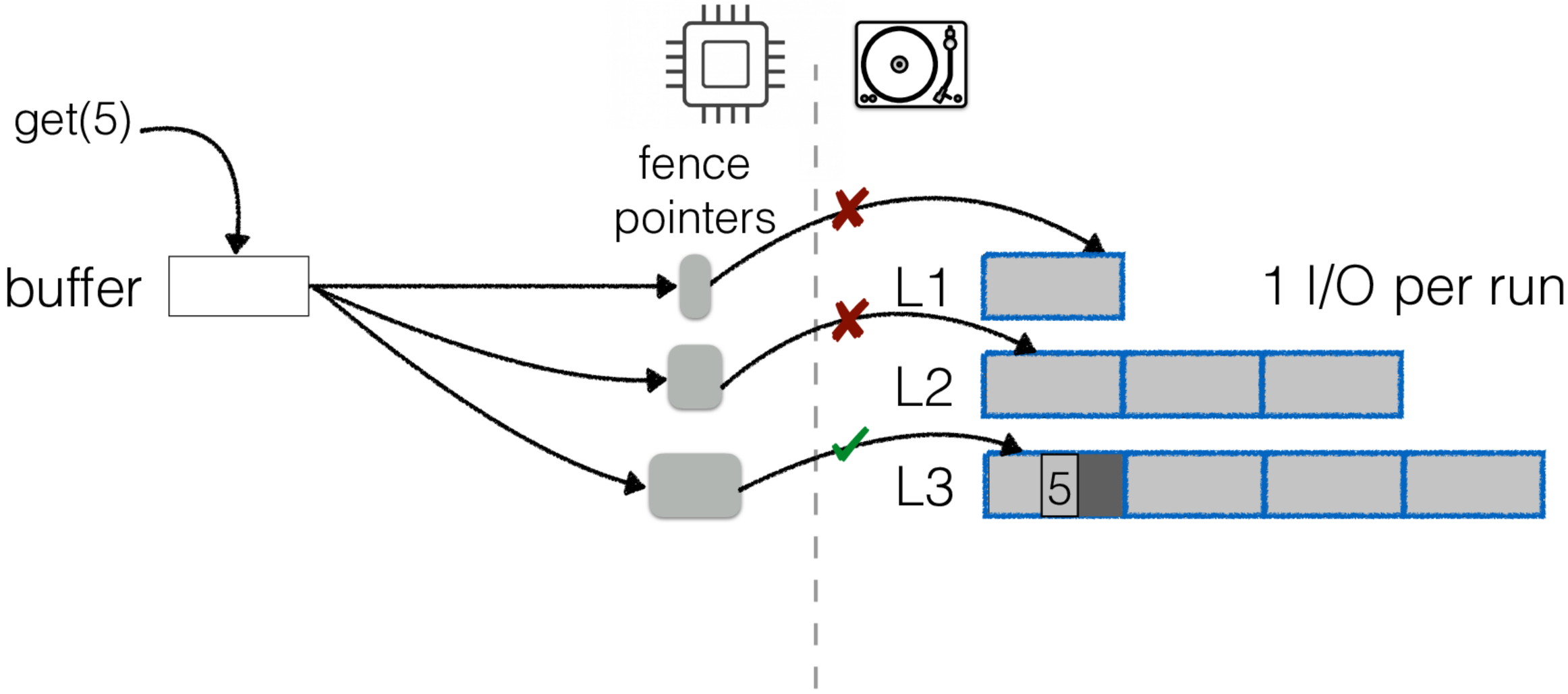
buffer

L1

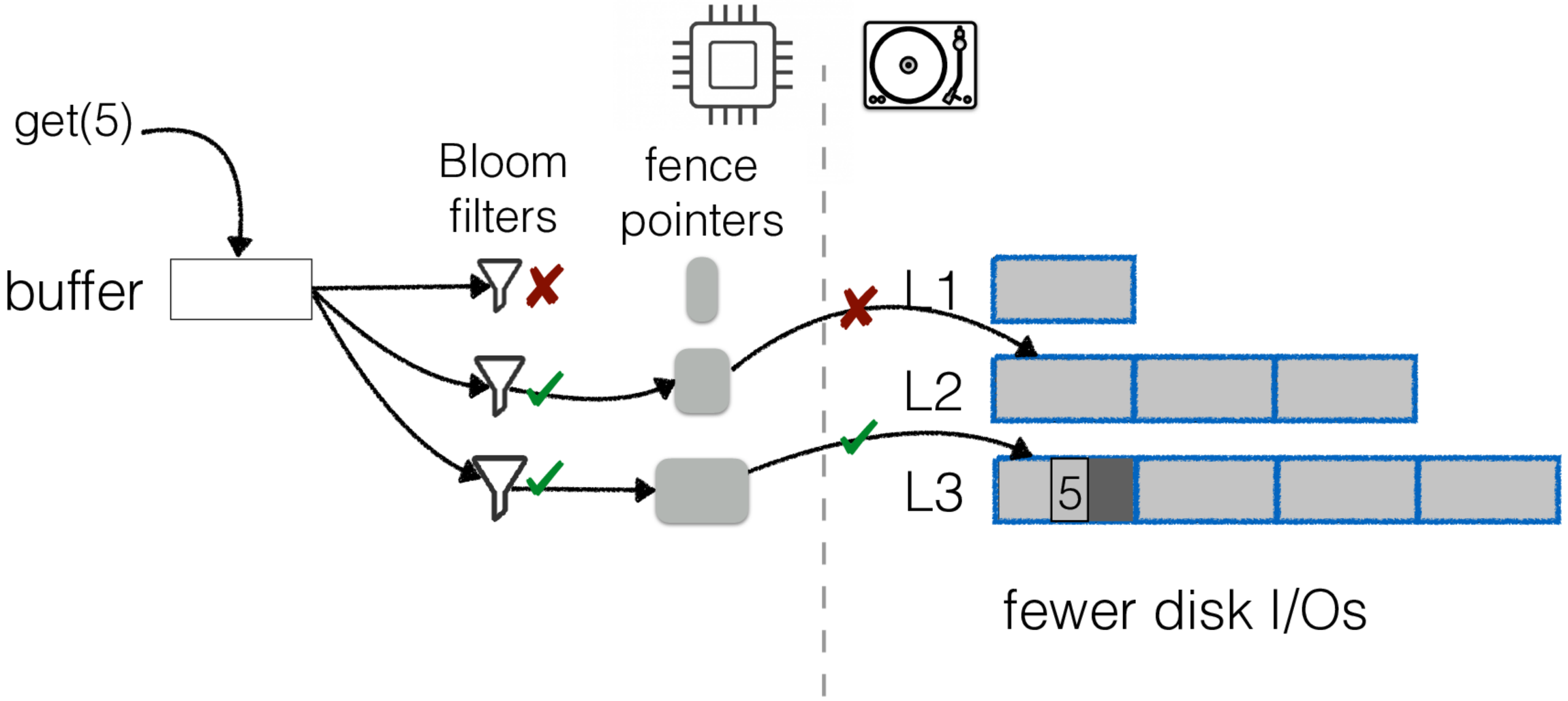
L2

L3

# log-structured merge-tree



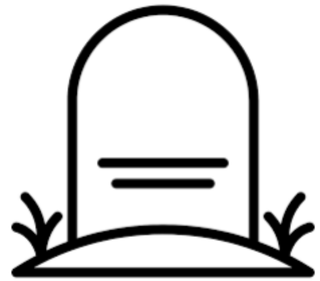
# log-structured merge-tree



# *Now, let's talk about deletes!*

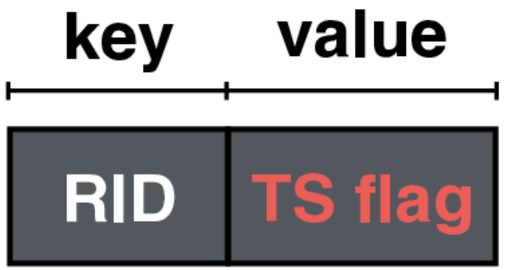
# deletes in LSM-tree

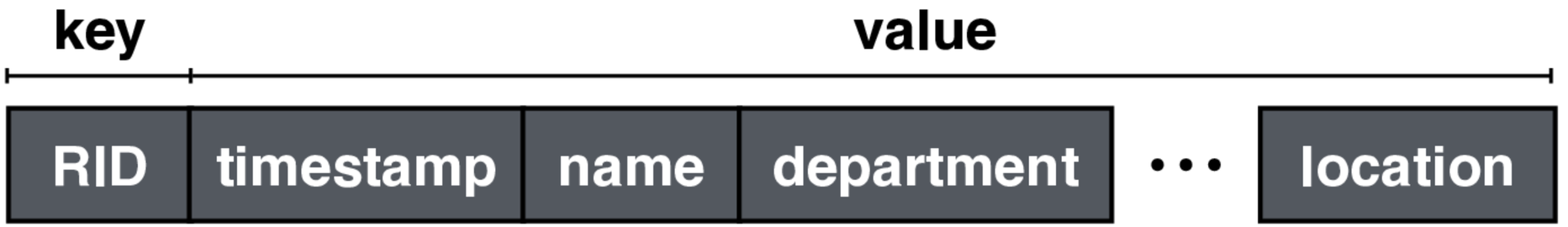
delete



# deletes in LSM-tree

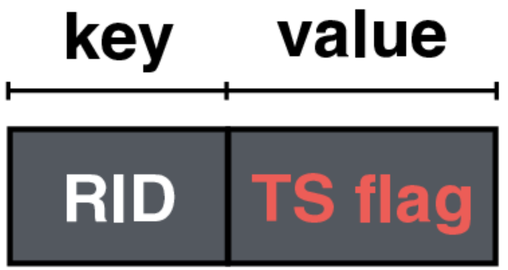
delete := insert tombstone

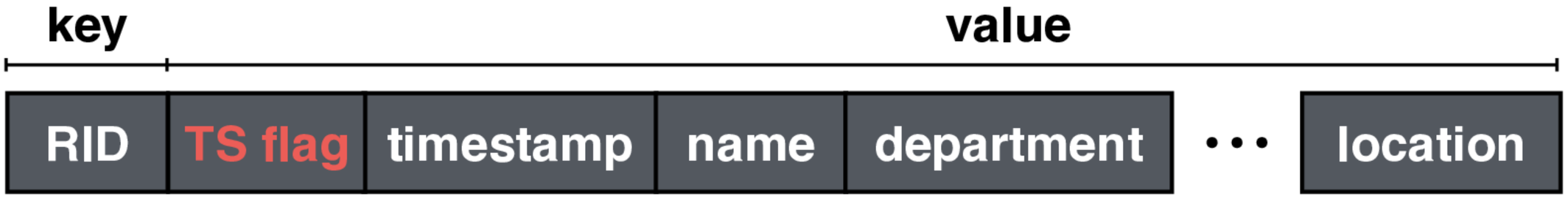


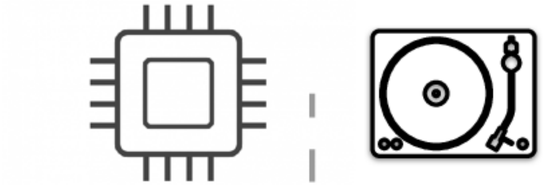
# deletes in LSM-tree

delete := insert tombstone



# deletes in LSM-tree



delete(5)

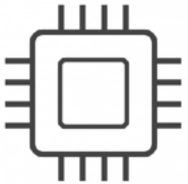
buffer 5

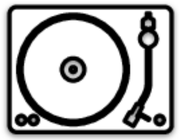
L1

L2 5

L3 5

# deletes in LSM-tree

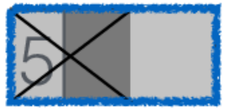




buffer

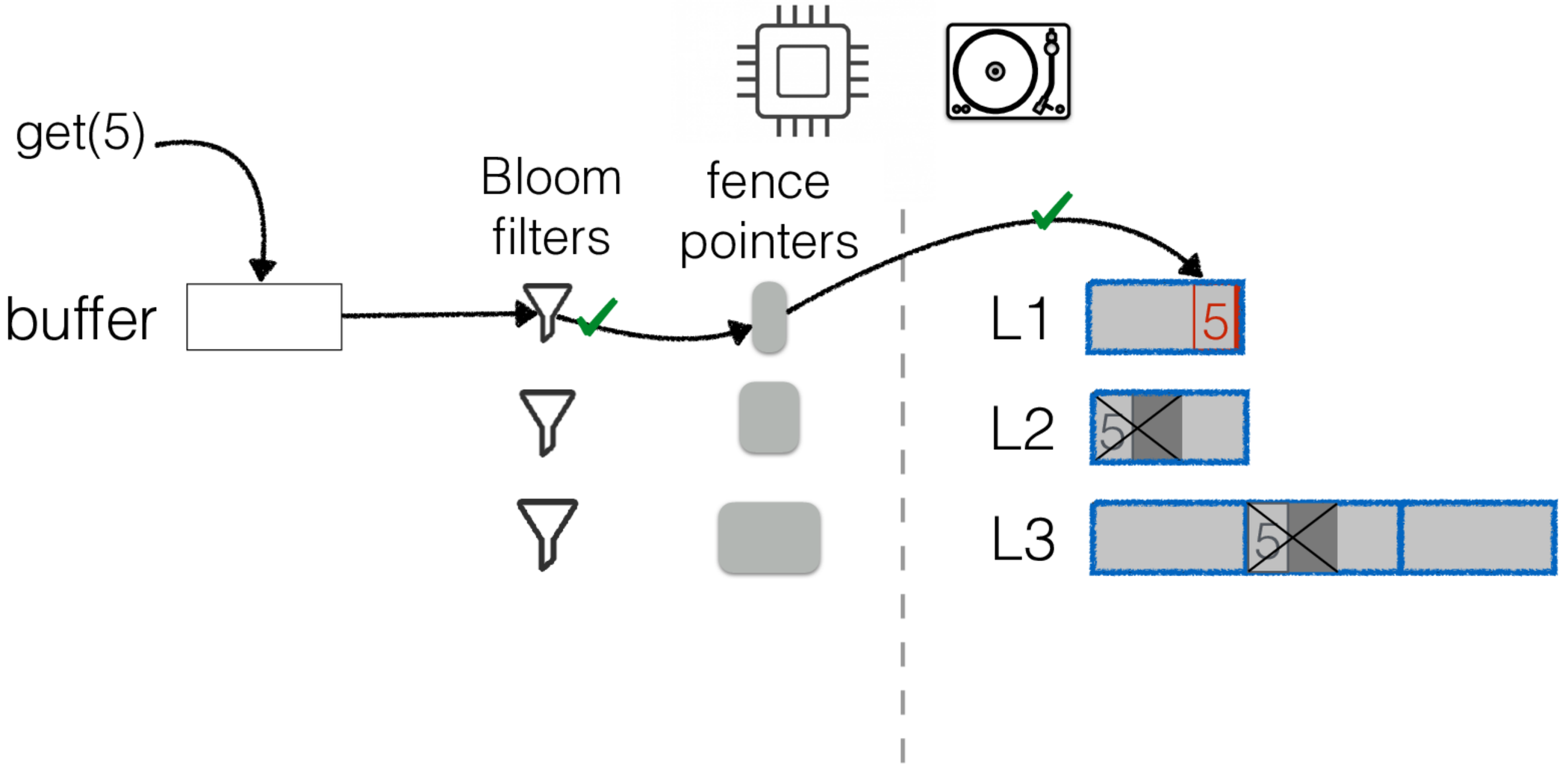
L1


L2




L3

# deletes in LSM-tree

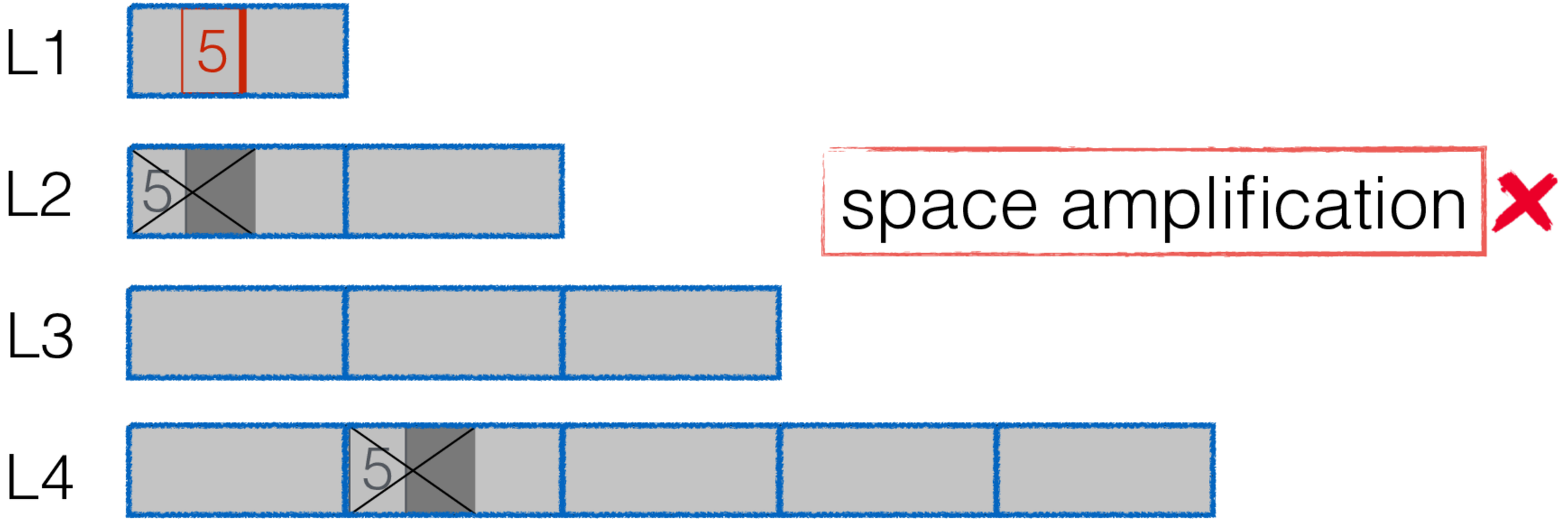


# the problems

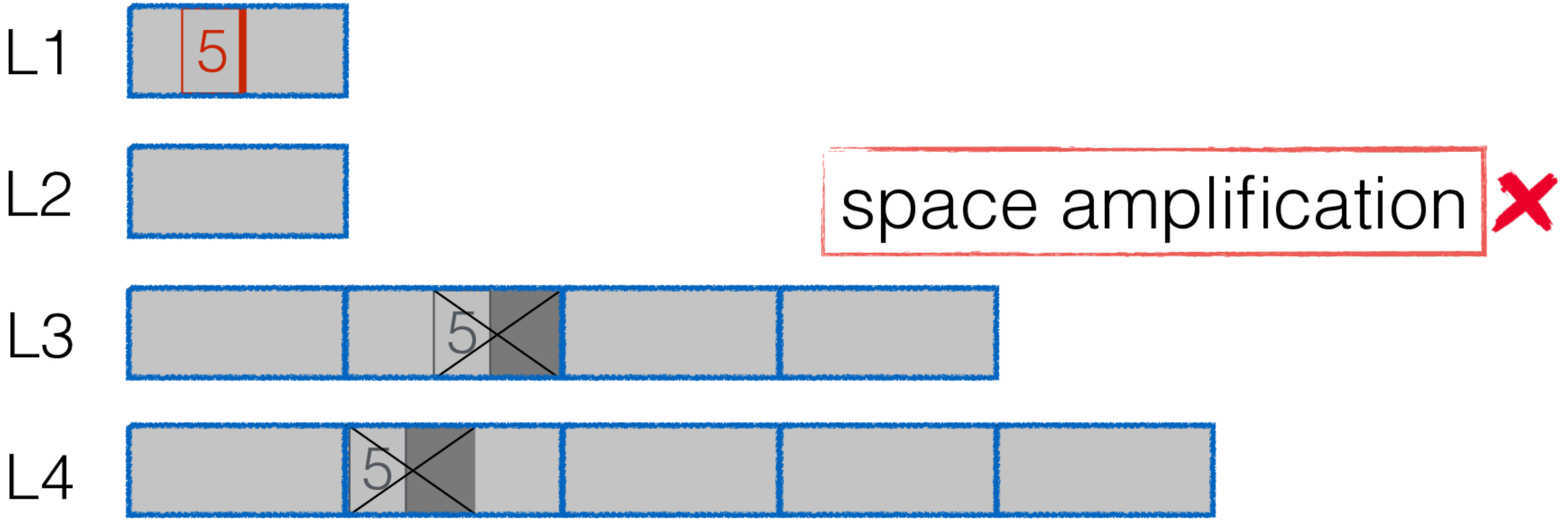
# the problems

# out-of-place deletes

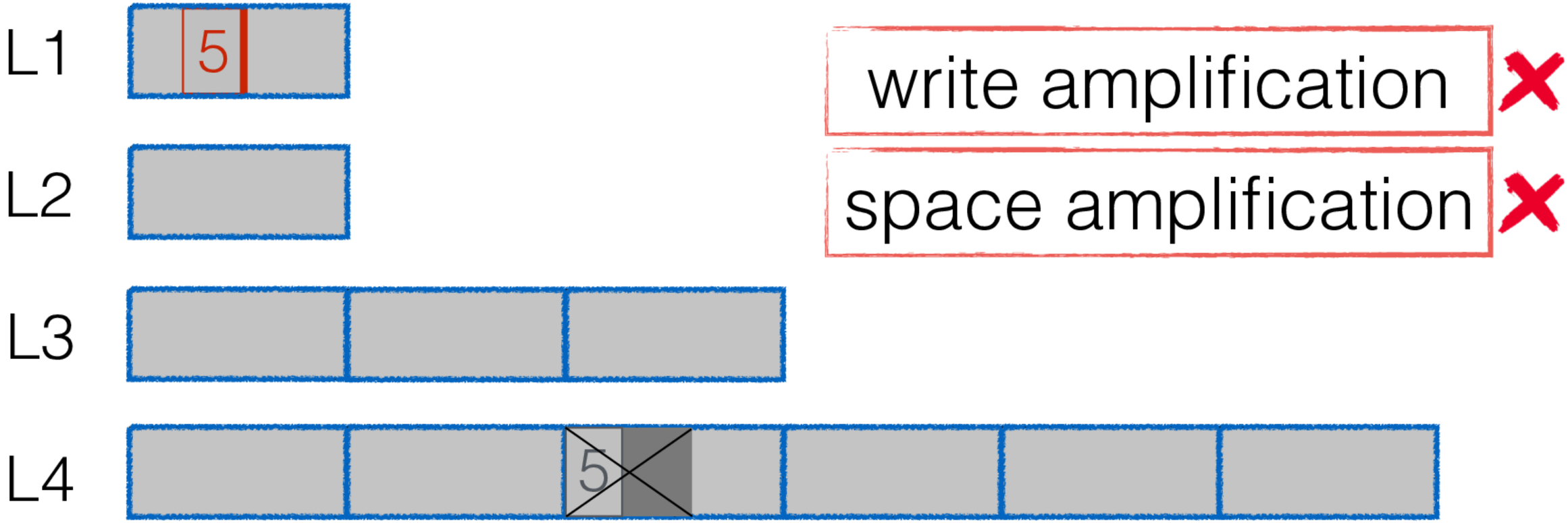
# out-of-place deletes



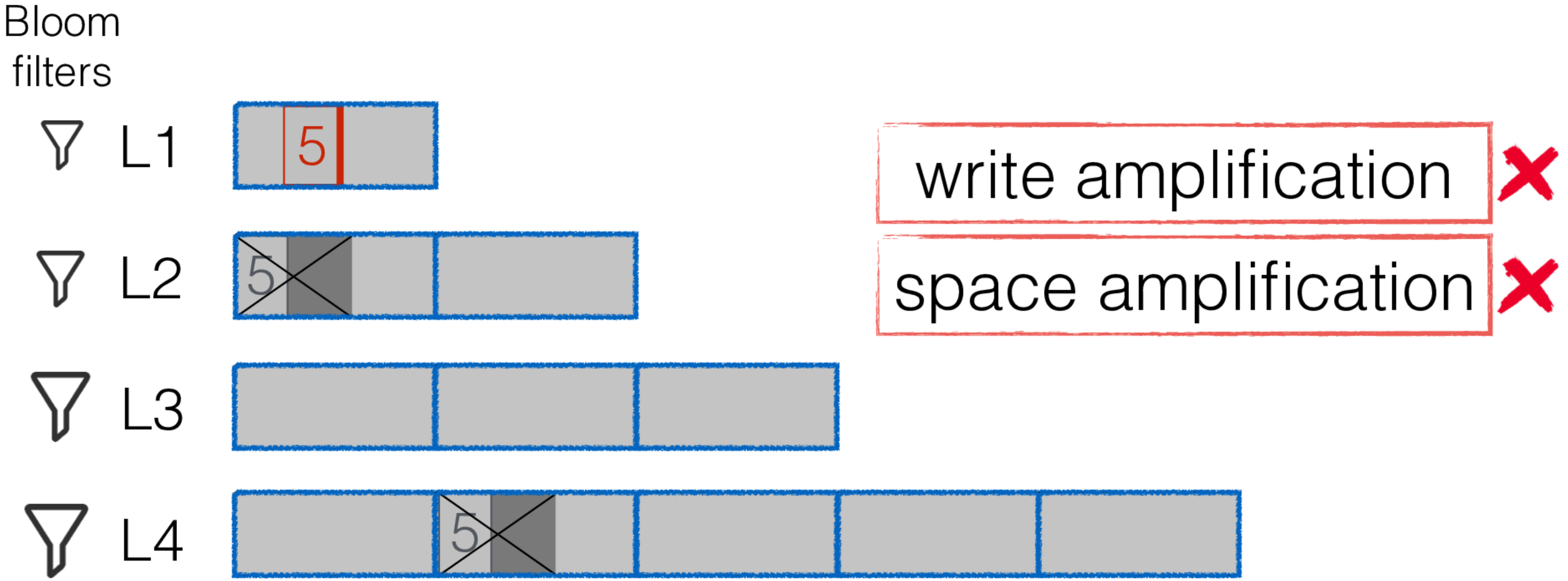
# out-of-place deletes



# out-of-place deletes



# out-of-place deletes



# out-of-place deletes

Bloom filters

L1


L2


L3

L4


poor read perf. ✗

write amplification ✗

space amplification ✗

# out-of-place deletes

Bloom filters

L1


L2


L3

L4


poor read perf. ✗

write amplification ✗

space amplification ✗

# the problems

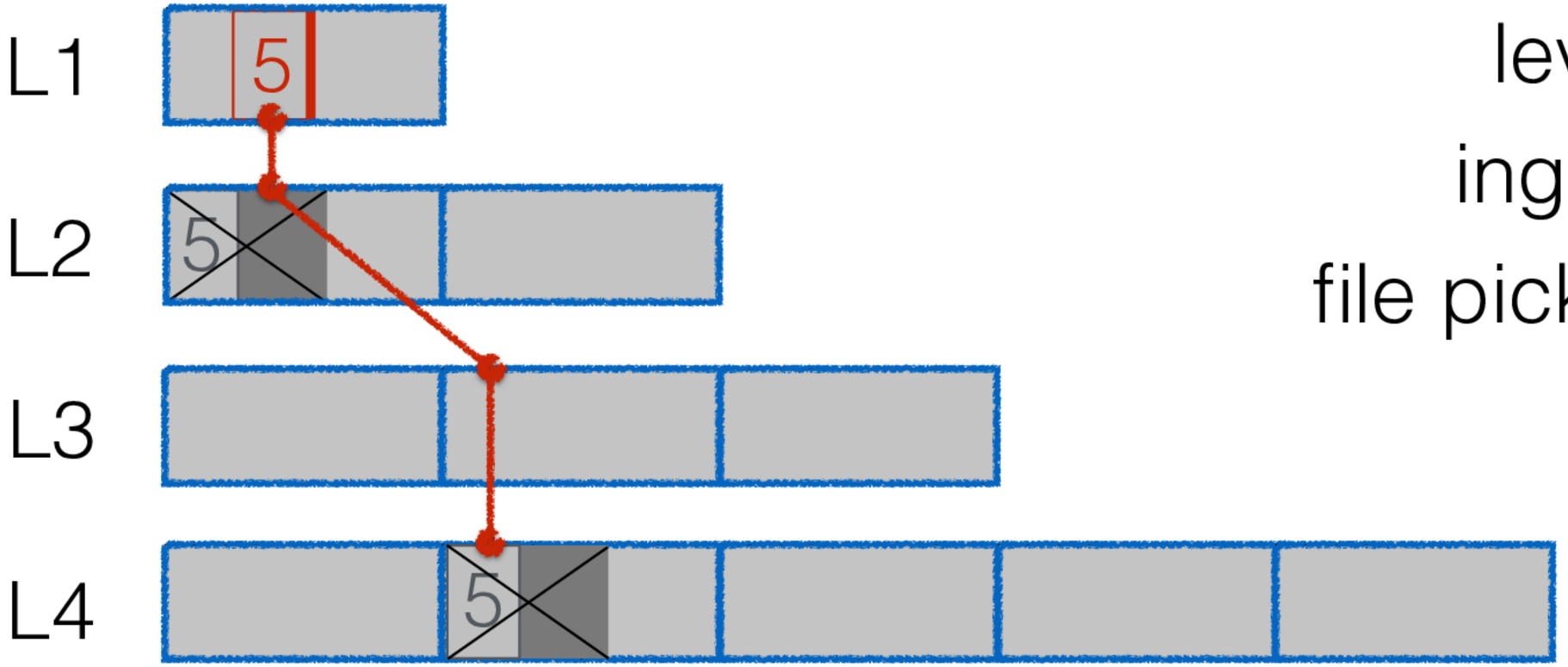
poor read perf.

write amplification

space amplification

# delete persistence latency

# delete persistence latency




size ratio

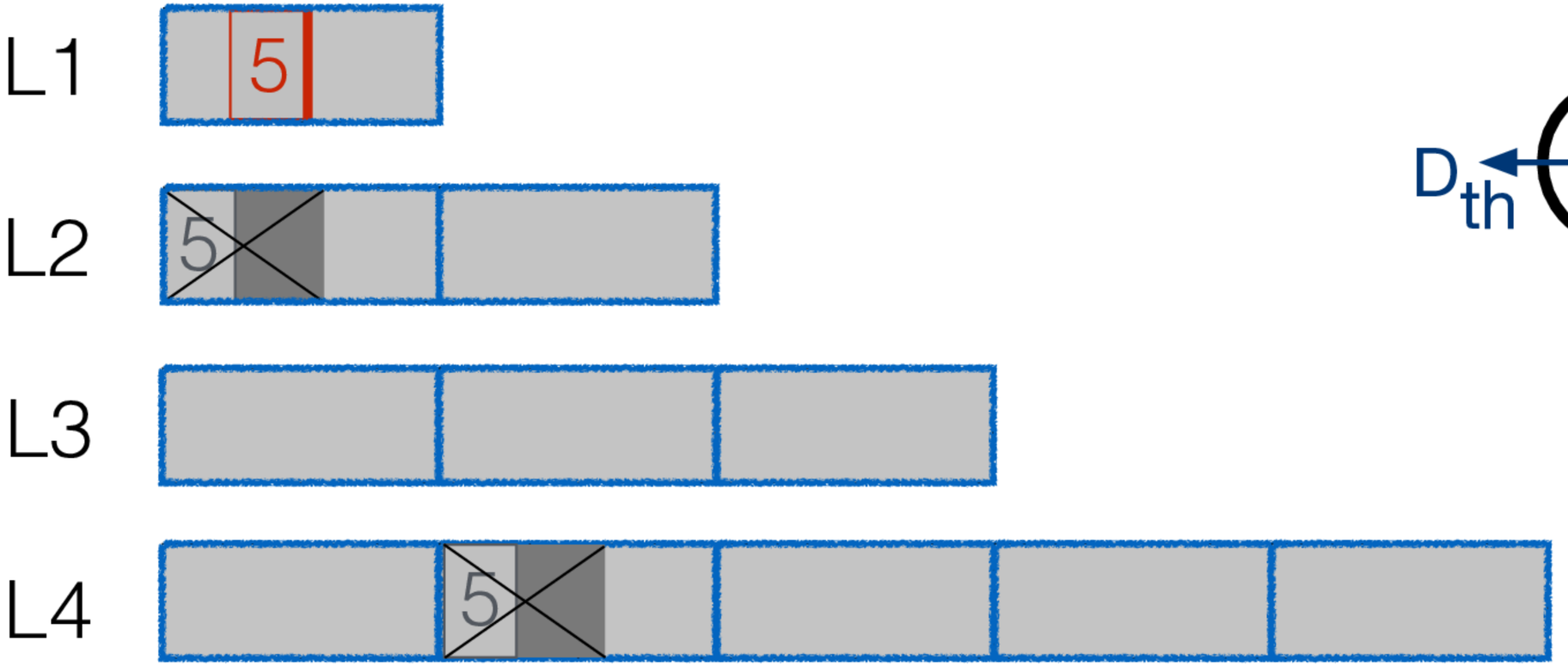
levels in tree

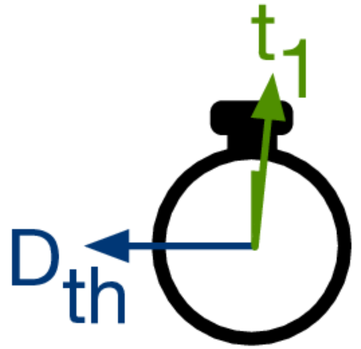
ingestion rate

file picking policy

# delete persistence latency

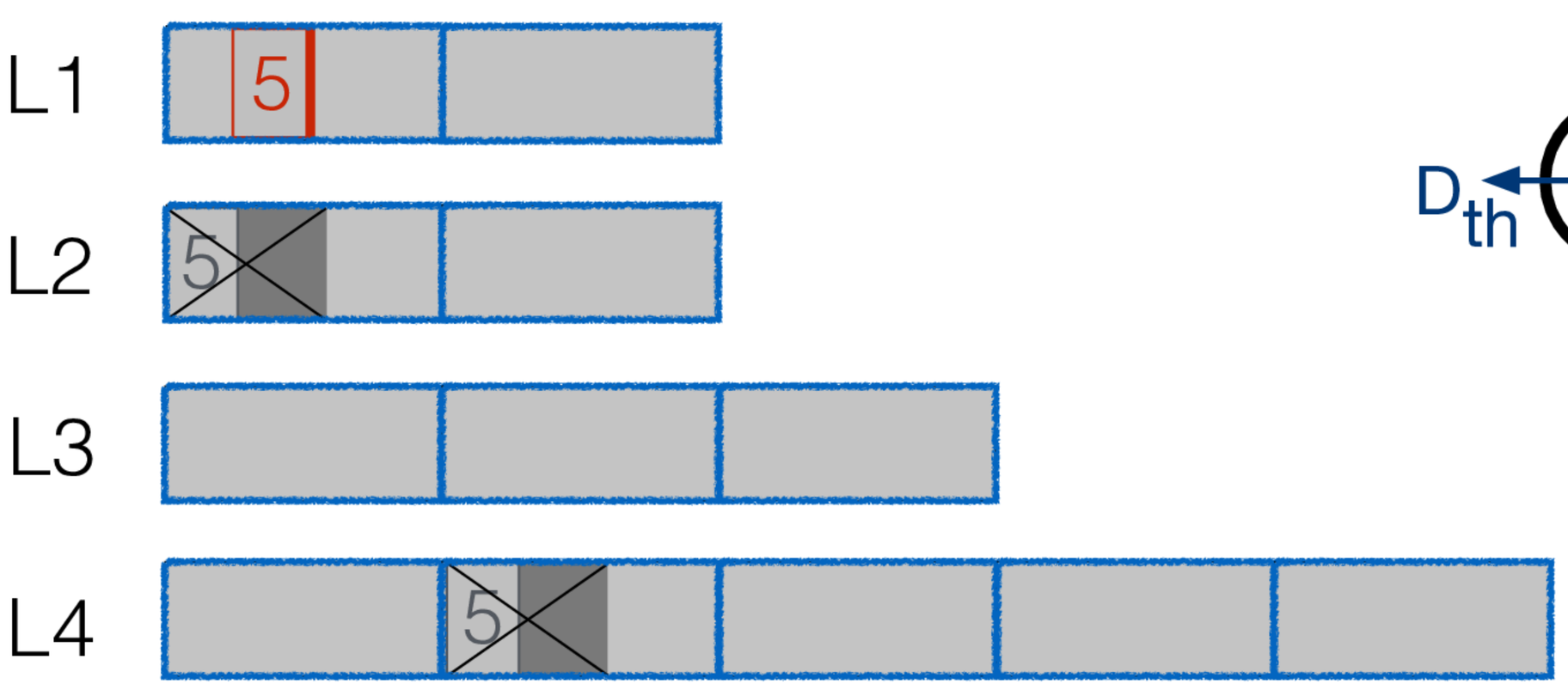
delete(5) within a threshold time: **$D_{th}$**

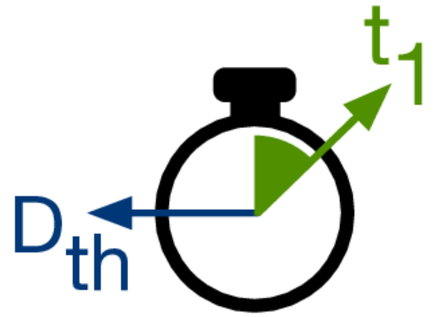


# delete persistence latency

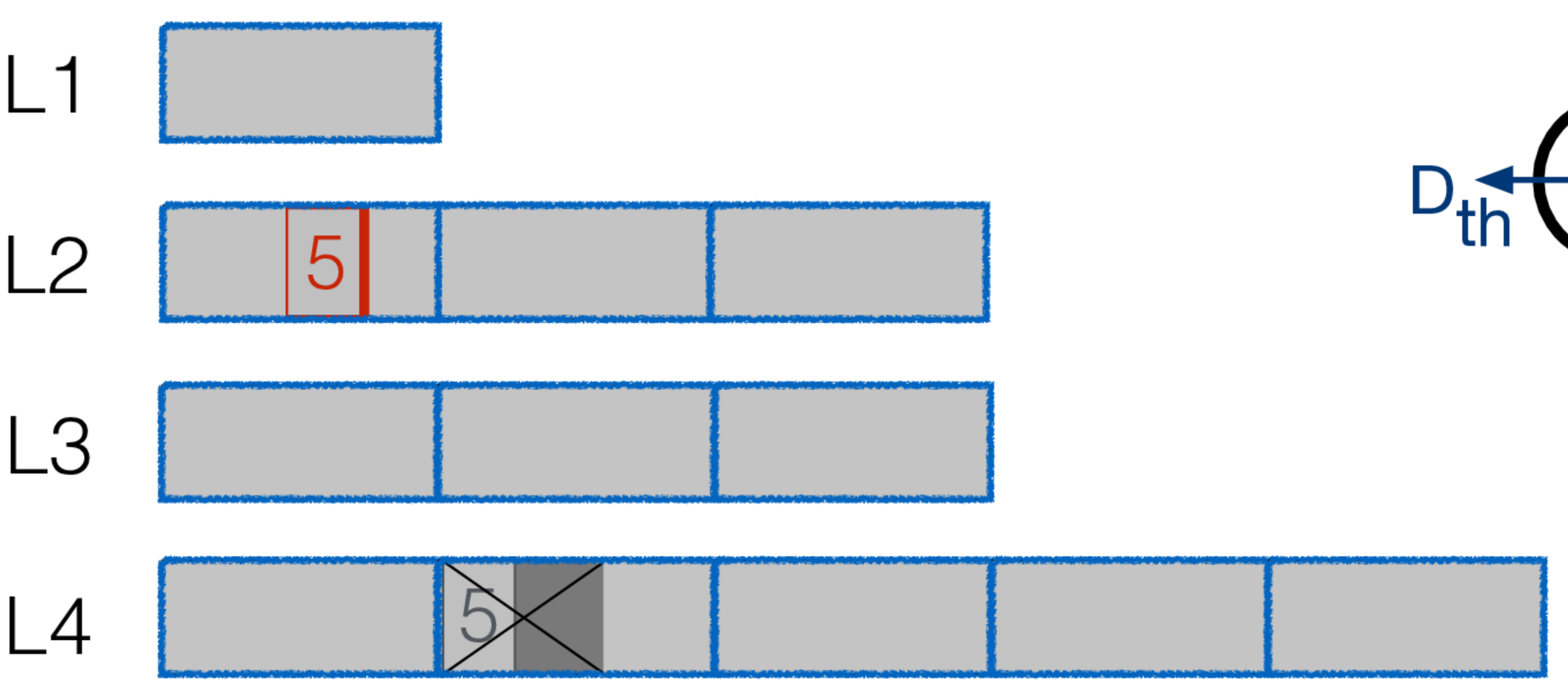
delete(5) within a threshold time: $\mathbf{D_{th}}$

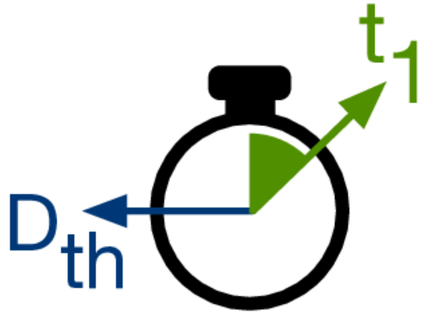


# delete persistence latency

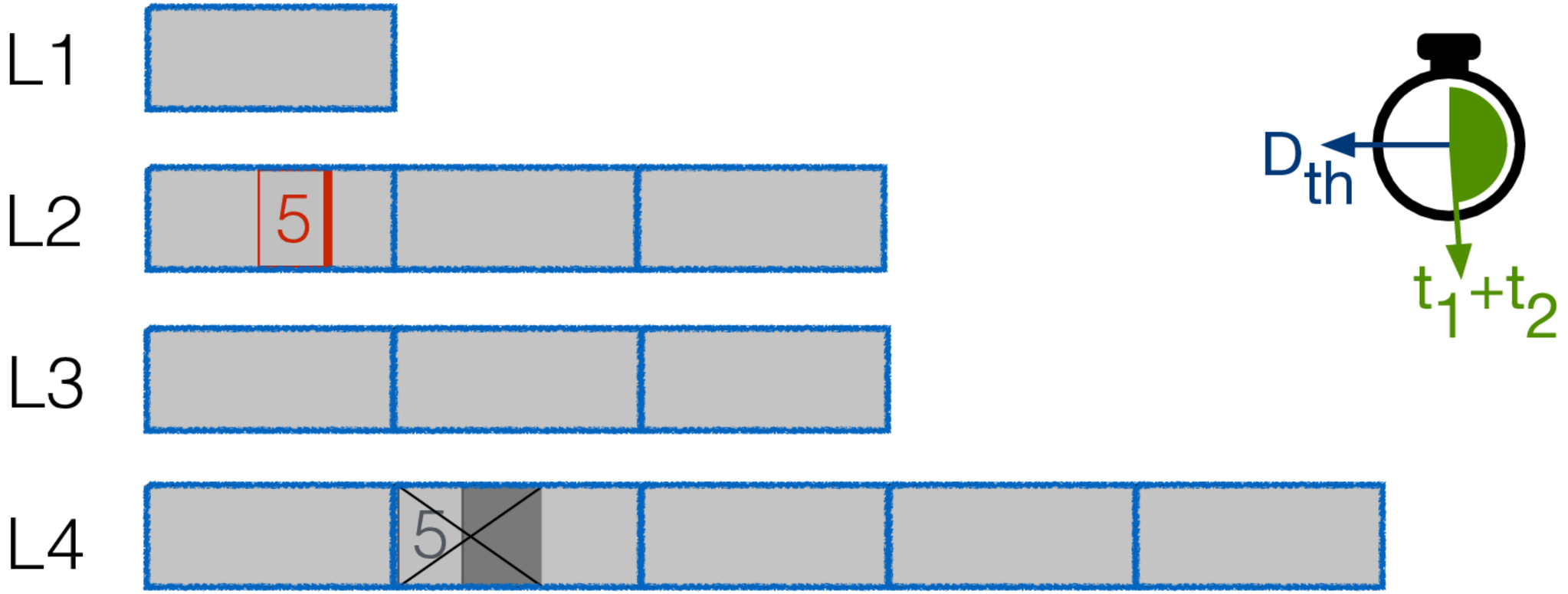
delete(5) within a threshold time: **$D_{th}$**



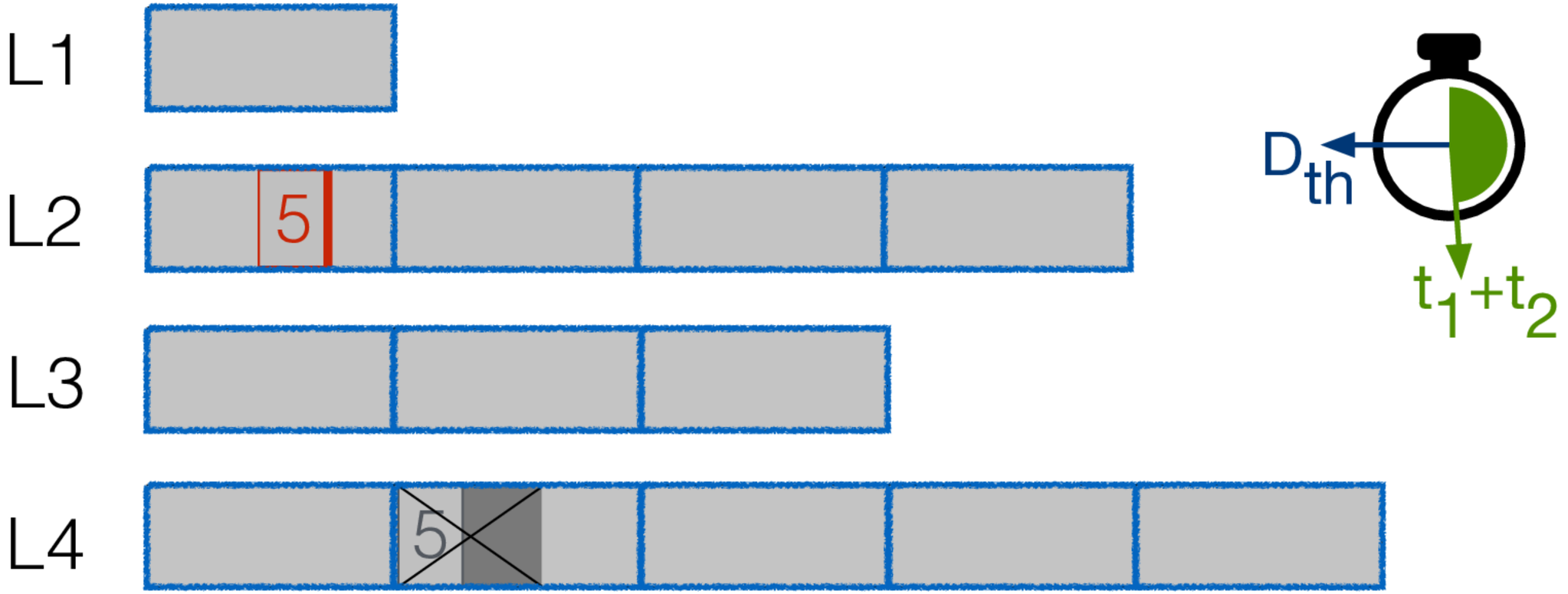
# delete persistence latency

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



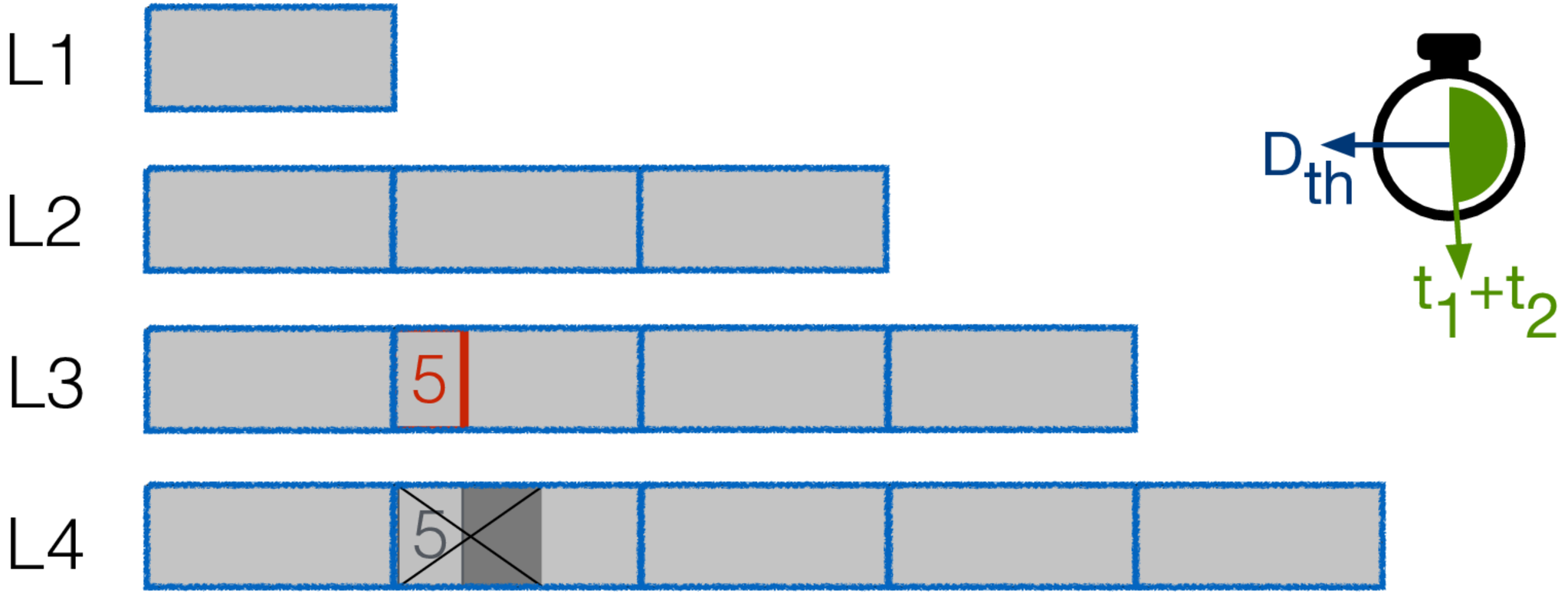
# delete persistence latency

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



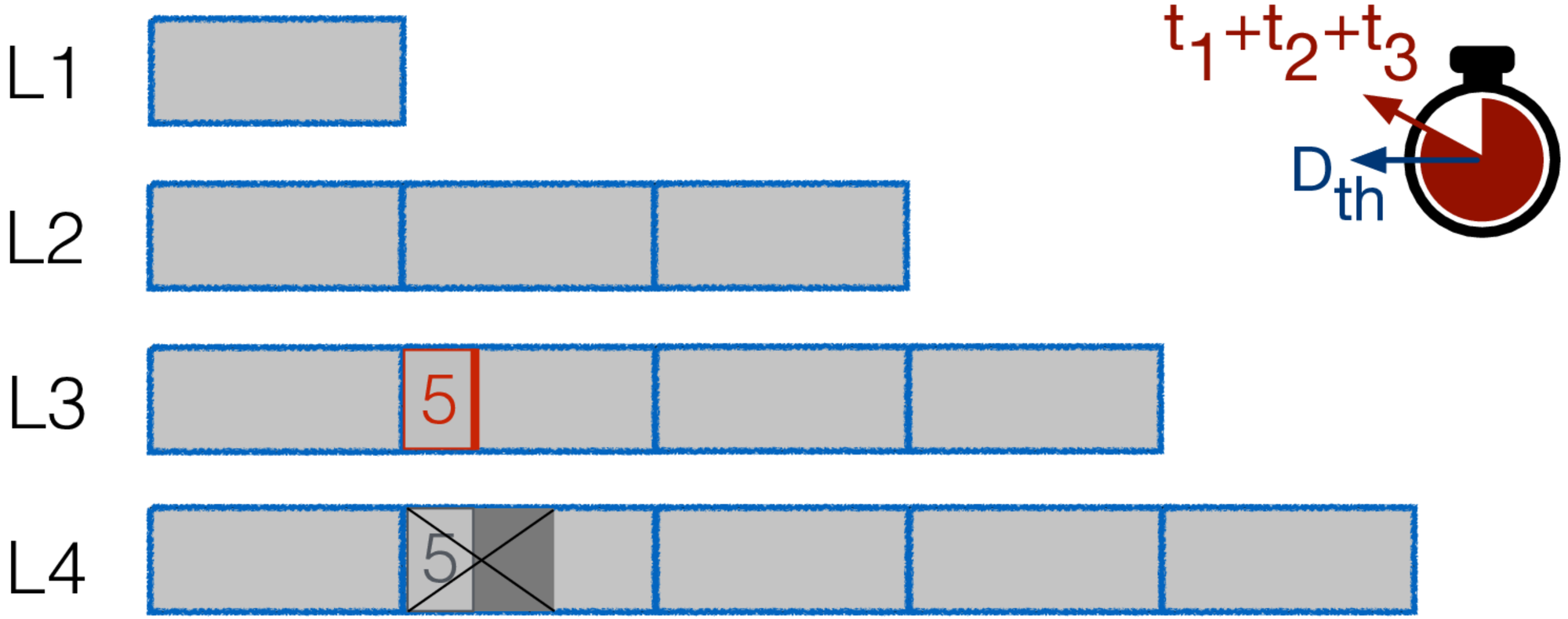
# delete persistence latency

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



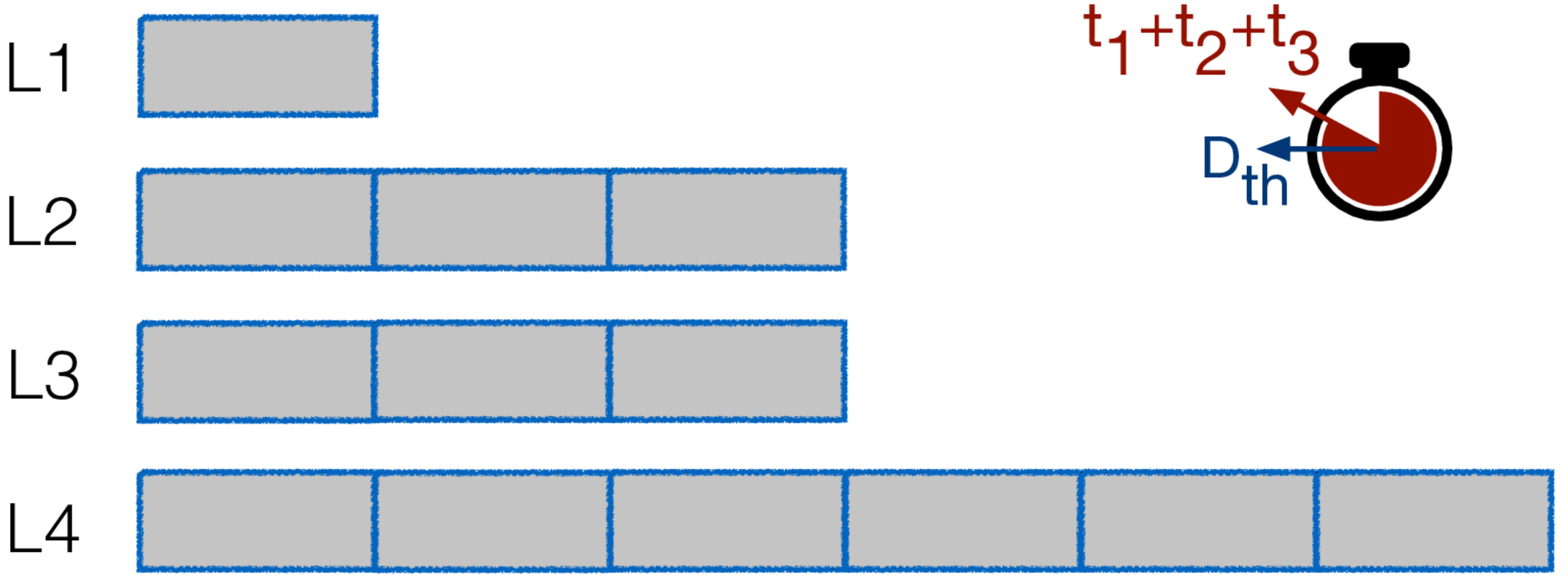
# delete persistence latency

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



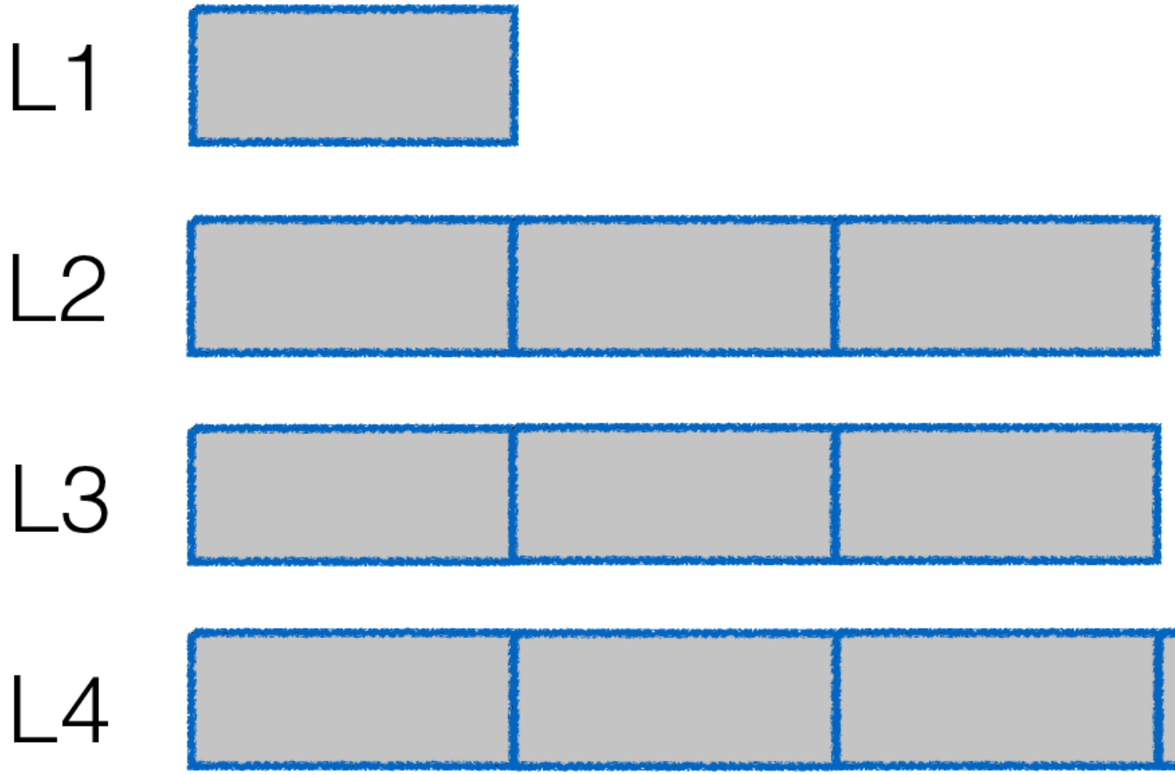
# delete persistence latency

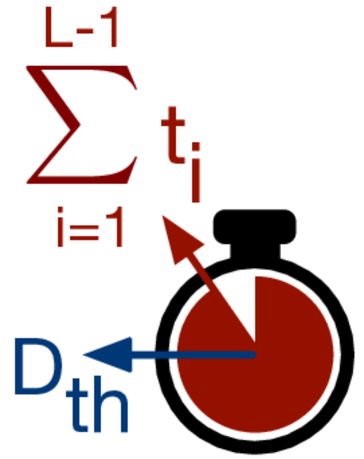
delete(5) within a threshold time: **$D_{th}$**



# delete persistence latency

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



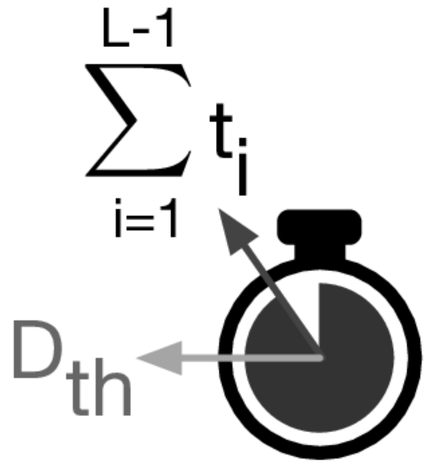

unbounded delete persistence latency ✗

# the problems

poor read perf.

write amplification

space amplification

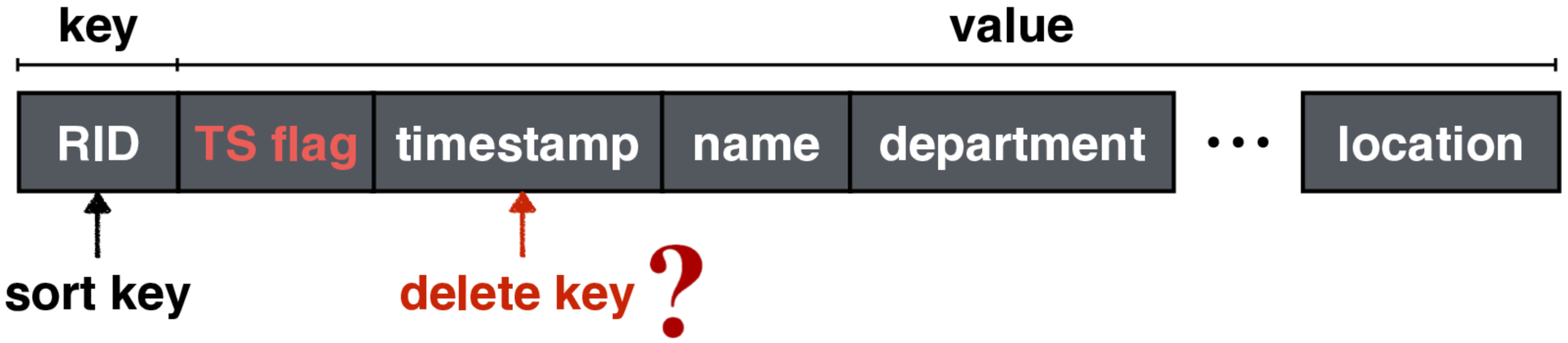



unbounded delete persistence latency

# deletes on a secondary attribute

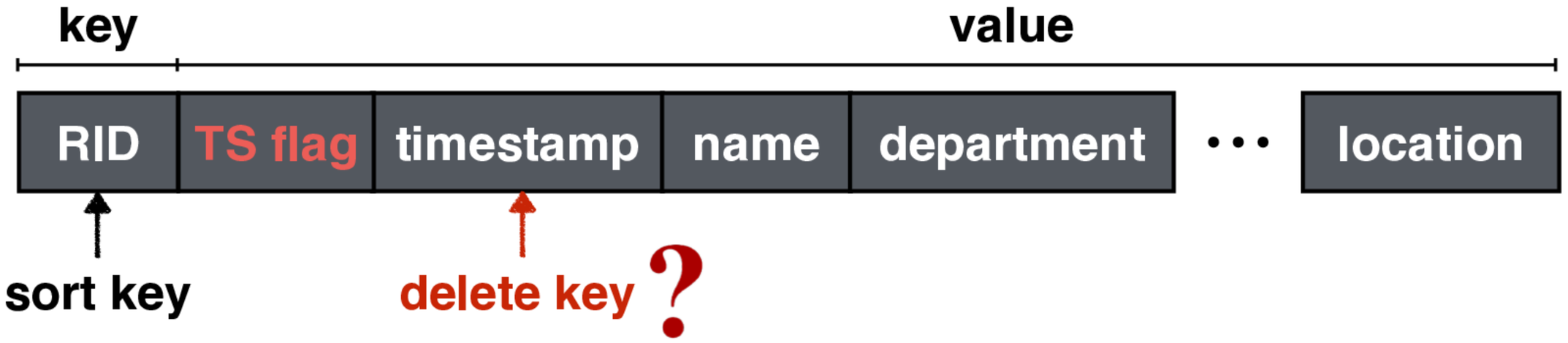
# deletes on a secondary attribute

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



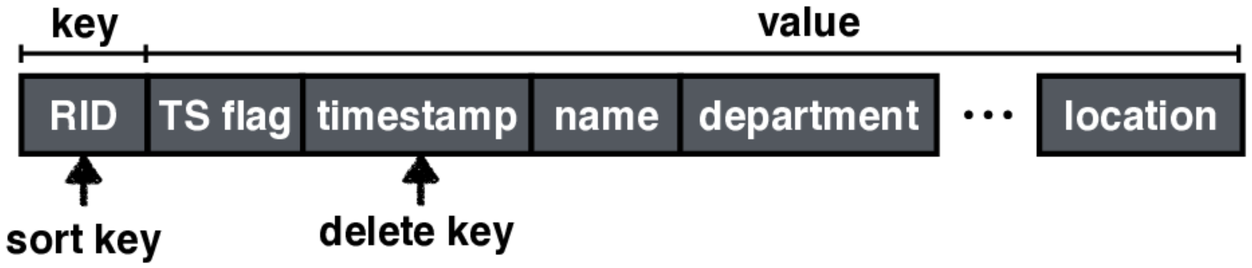
# deletes on a secondary attribute

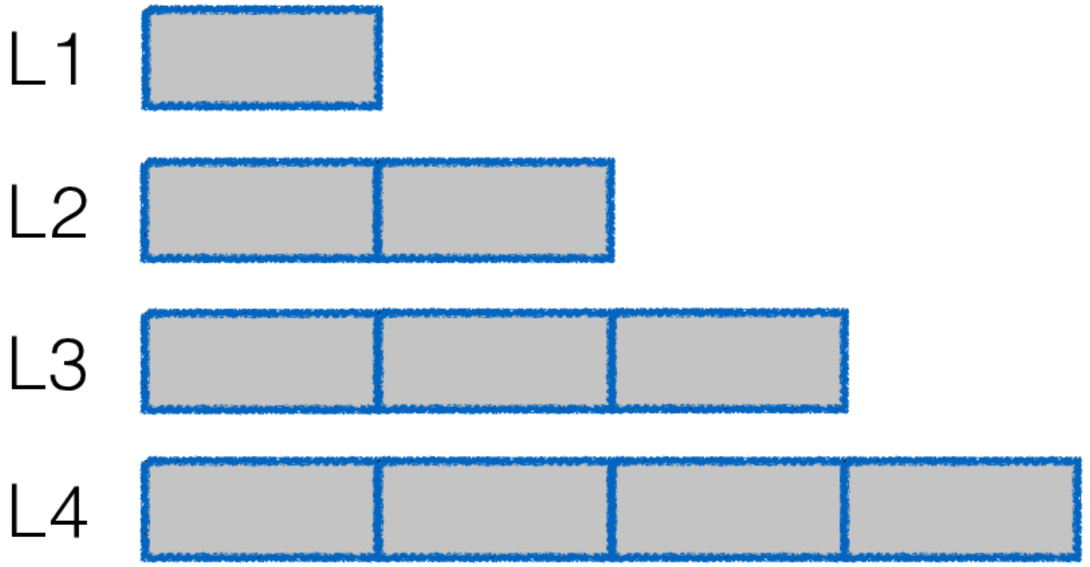
delete all entries older than: **D days**



# deletes on a secondary attribute

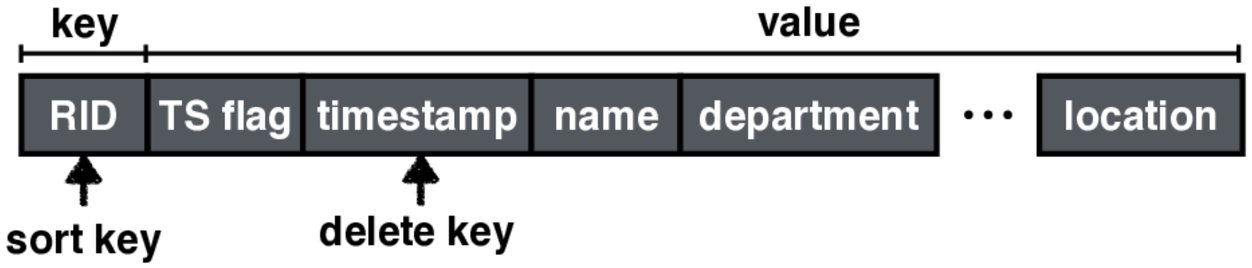
delete all entries older than: **D days**

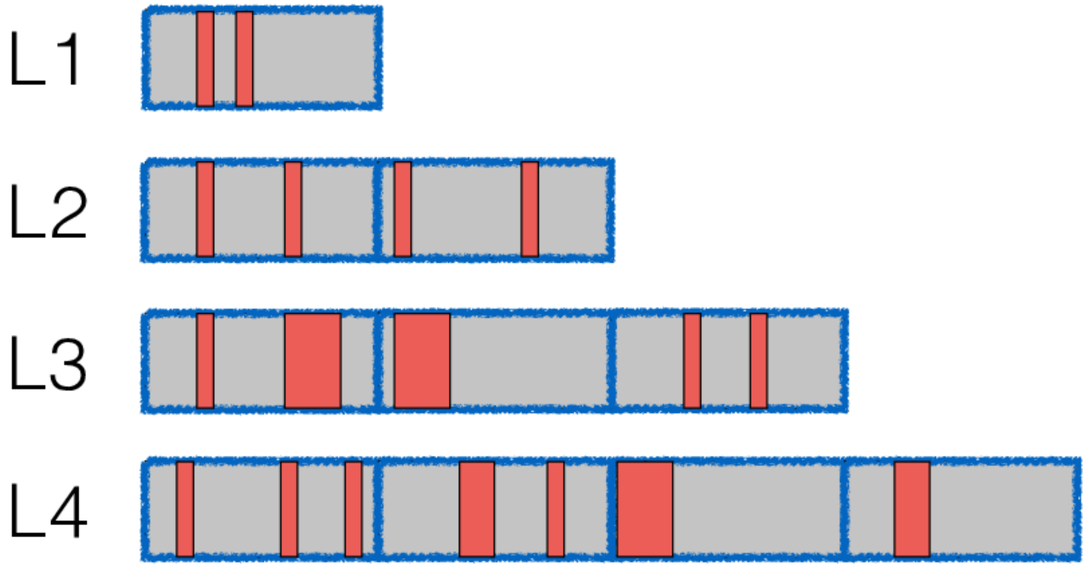


# deletes on a secondary attribute

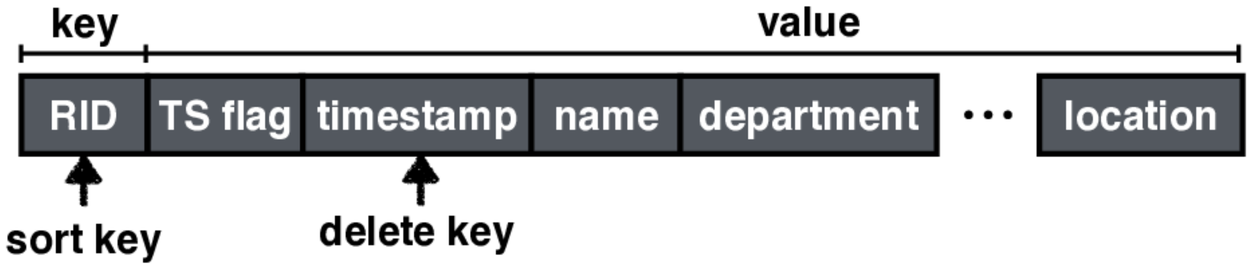
delete all entries older than: **D days**

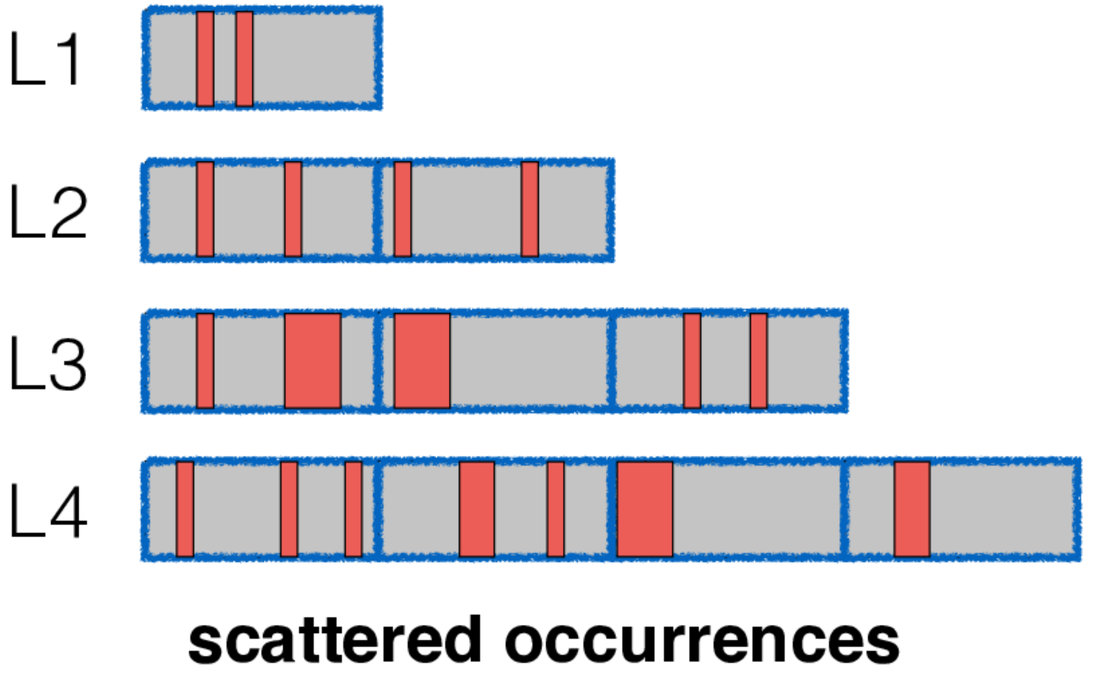


# deletes on a secondary attribute

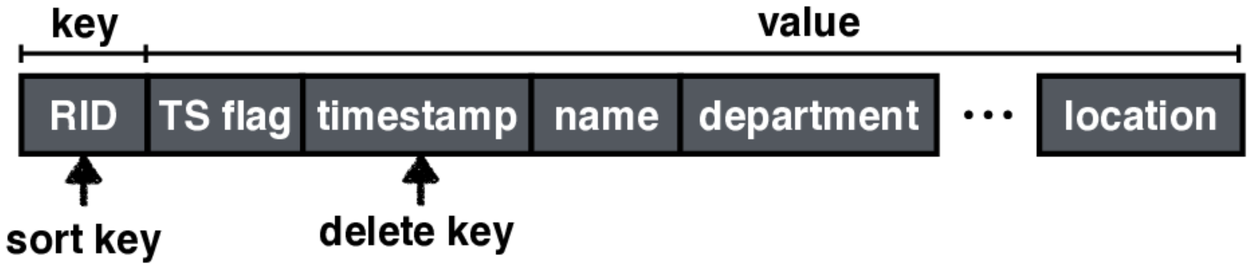
delete all entries older than: **D days**



# deletes on a secondary attribute

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




L1

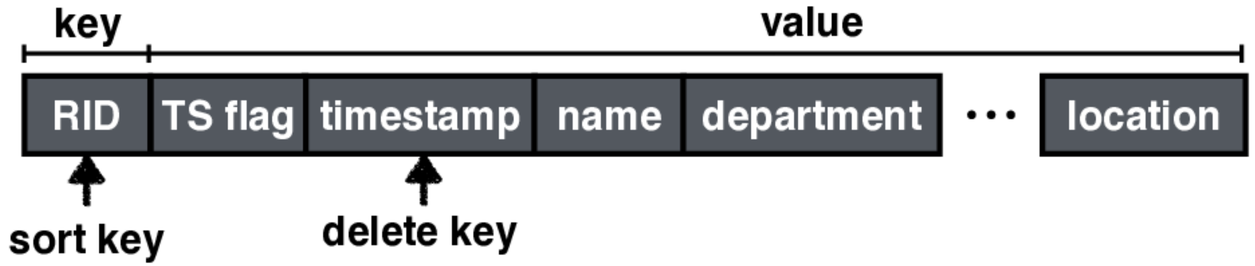
L2

L3

L4

# deletes on a secondary attribute

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

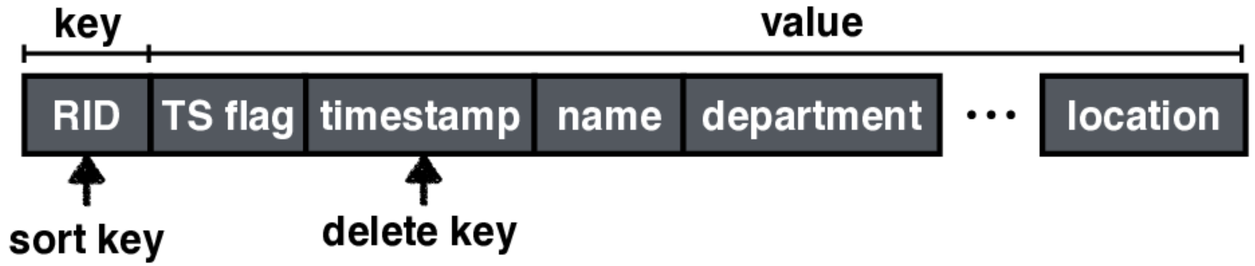



latency spikes ✗

superfluous I/Os ✗

# deletes on a secondary attribute

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




latency spikes ✗

superfluous I/Os ✗

L1

L2

L3

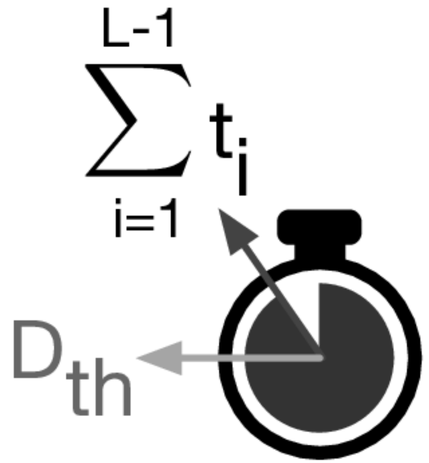
L4

# 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


unbounded delete persistence latency

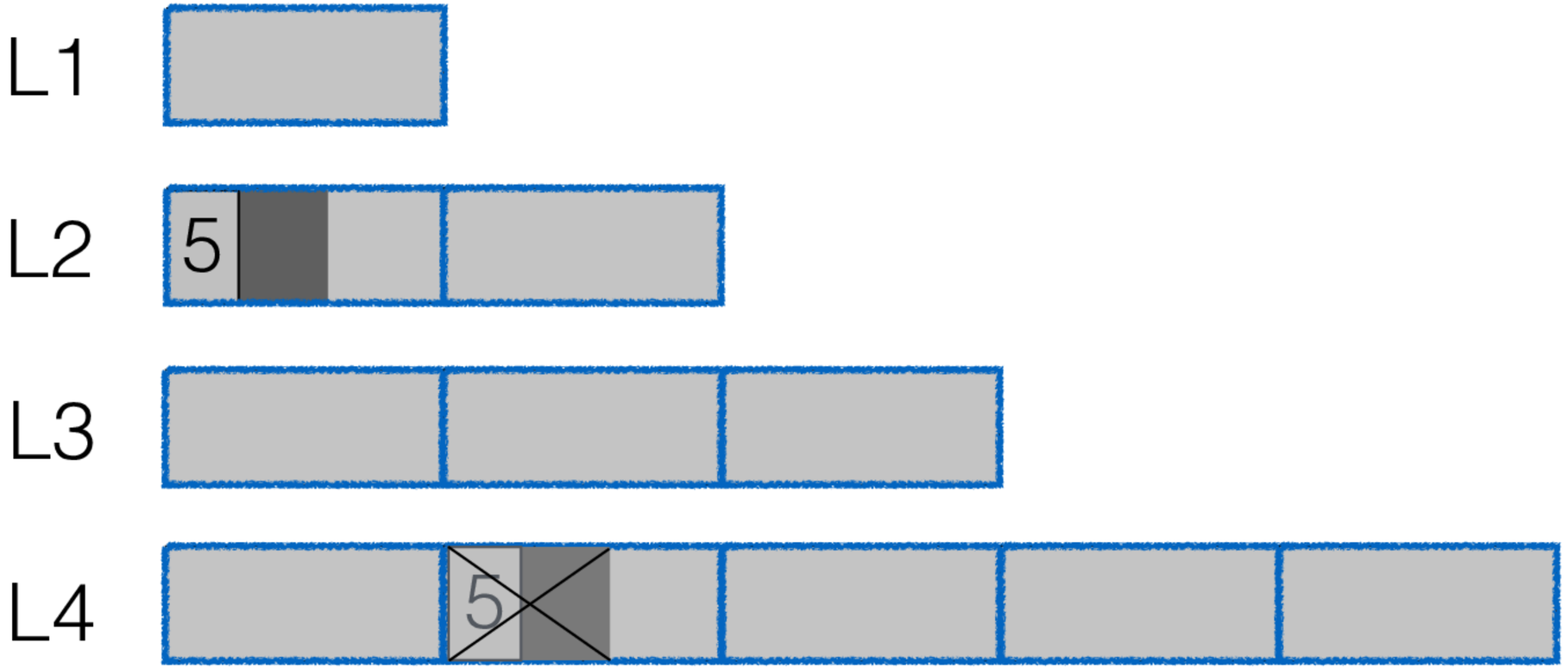
**FADE**

latency spikes

superfluous I/Os

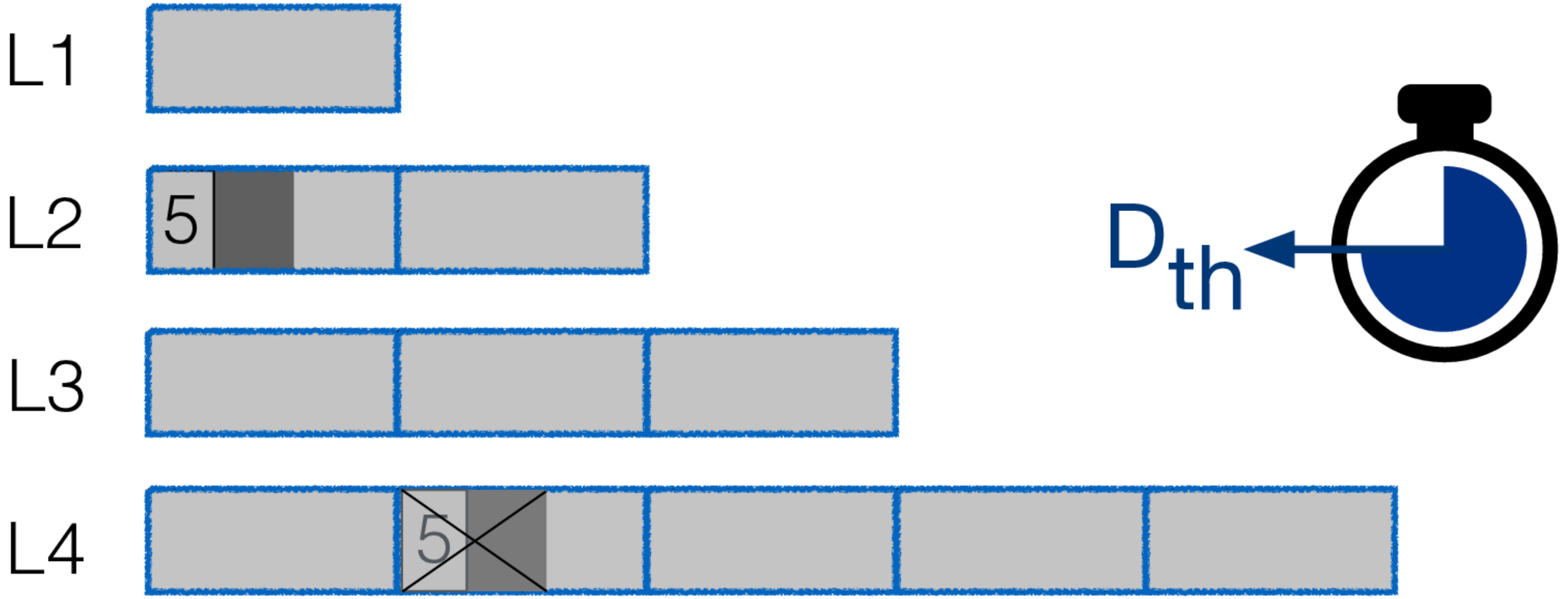
# FAst DElete

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



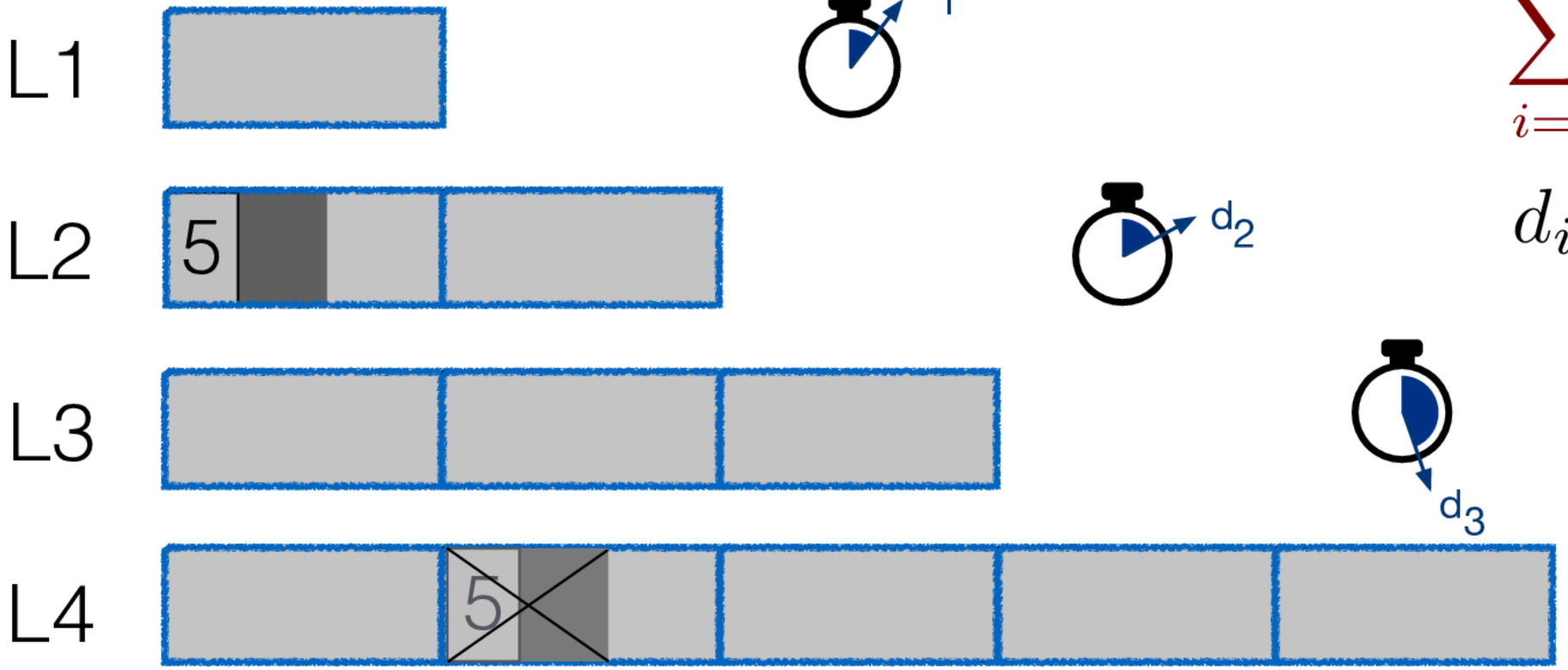
# FAst DElete

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



# FAst DElete

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

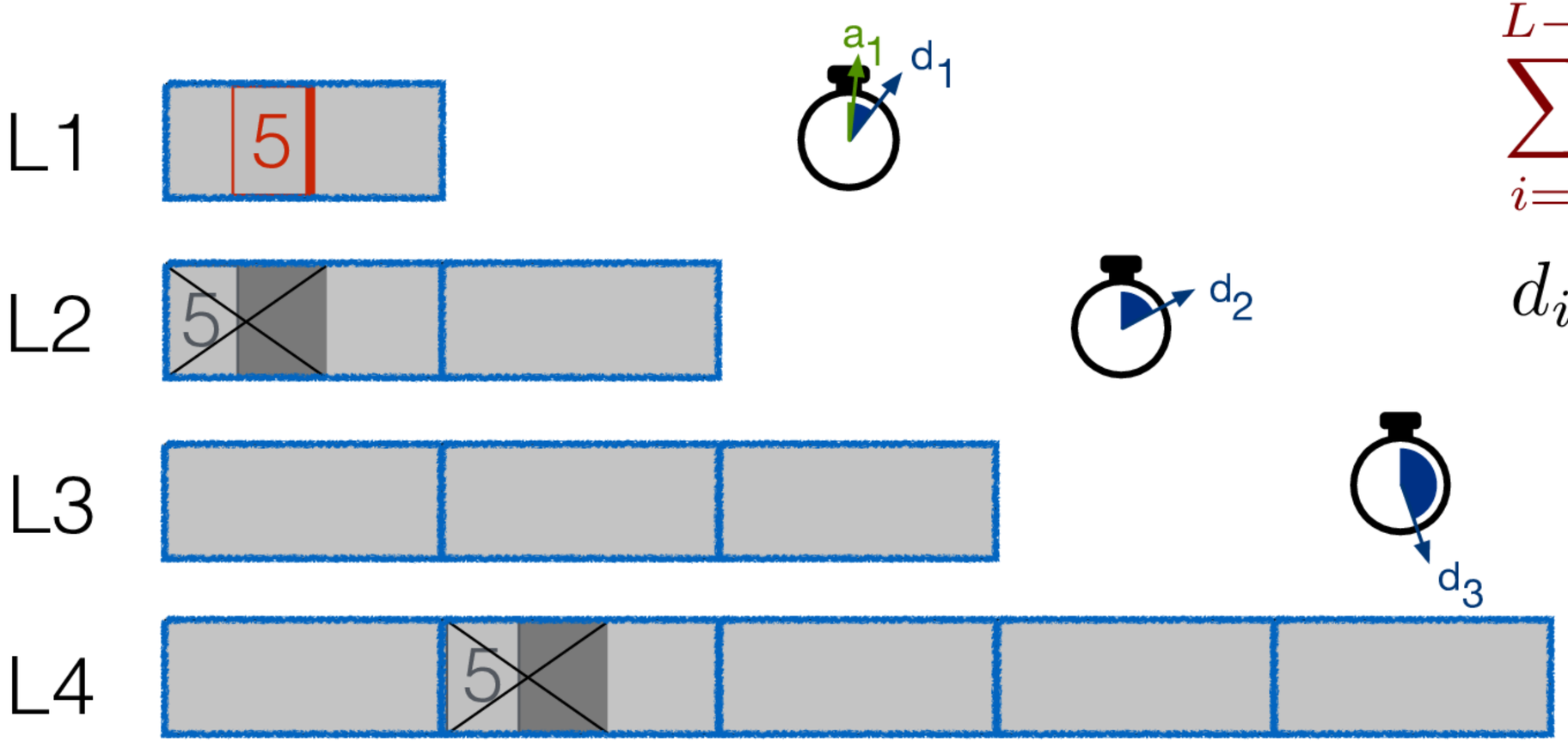



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

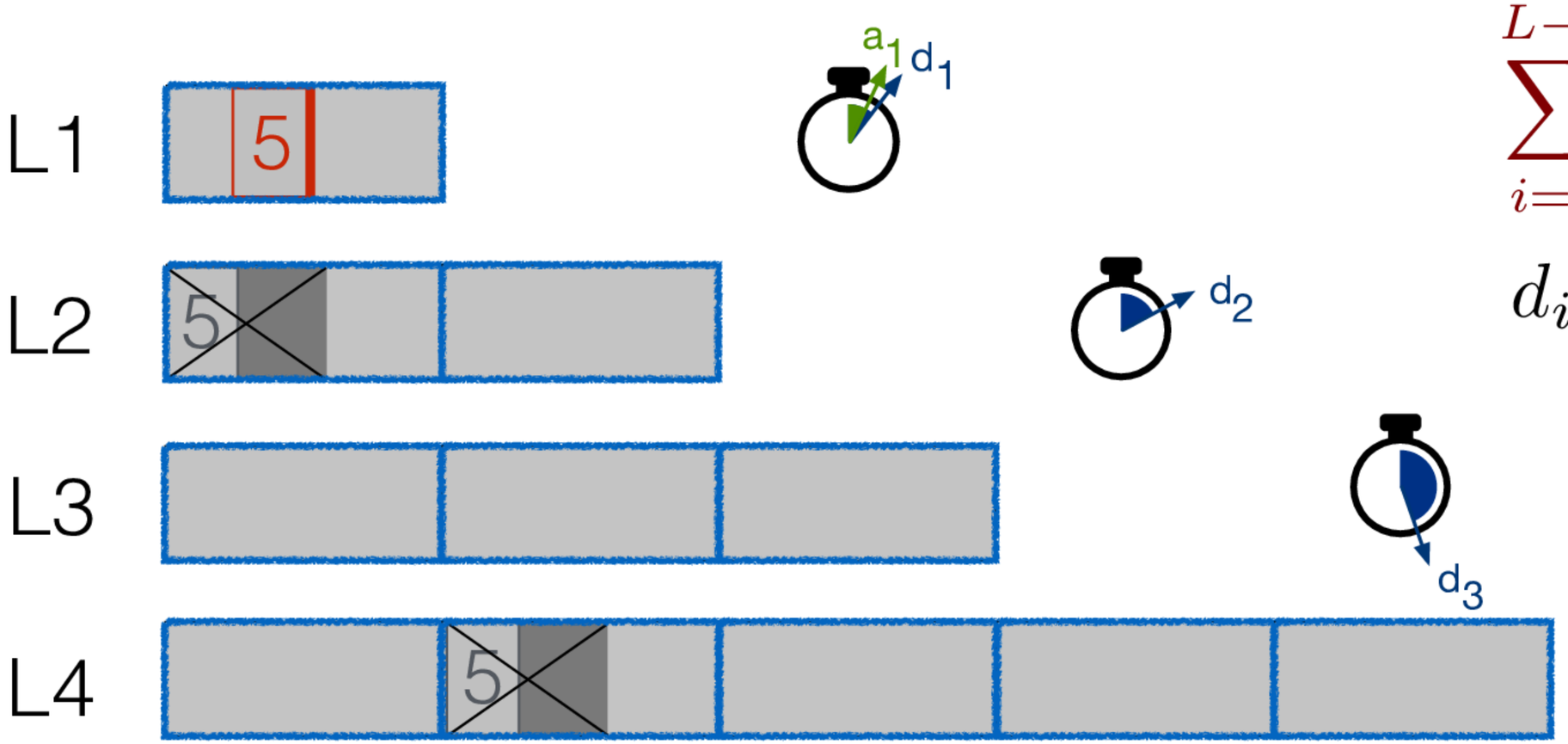



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

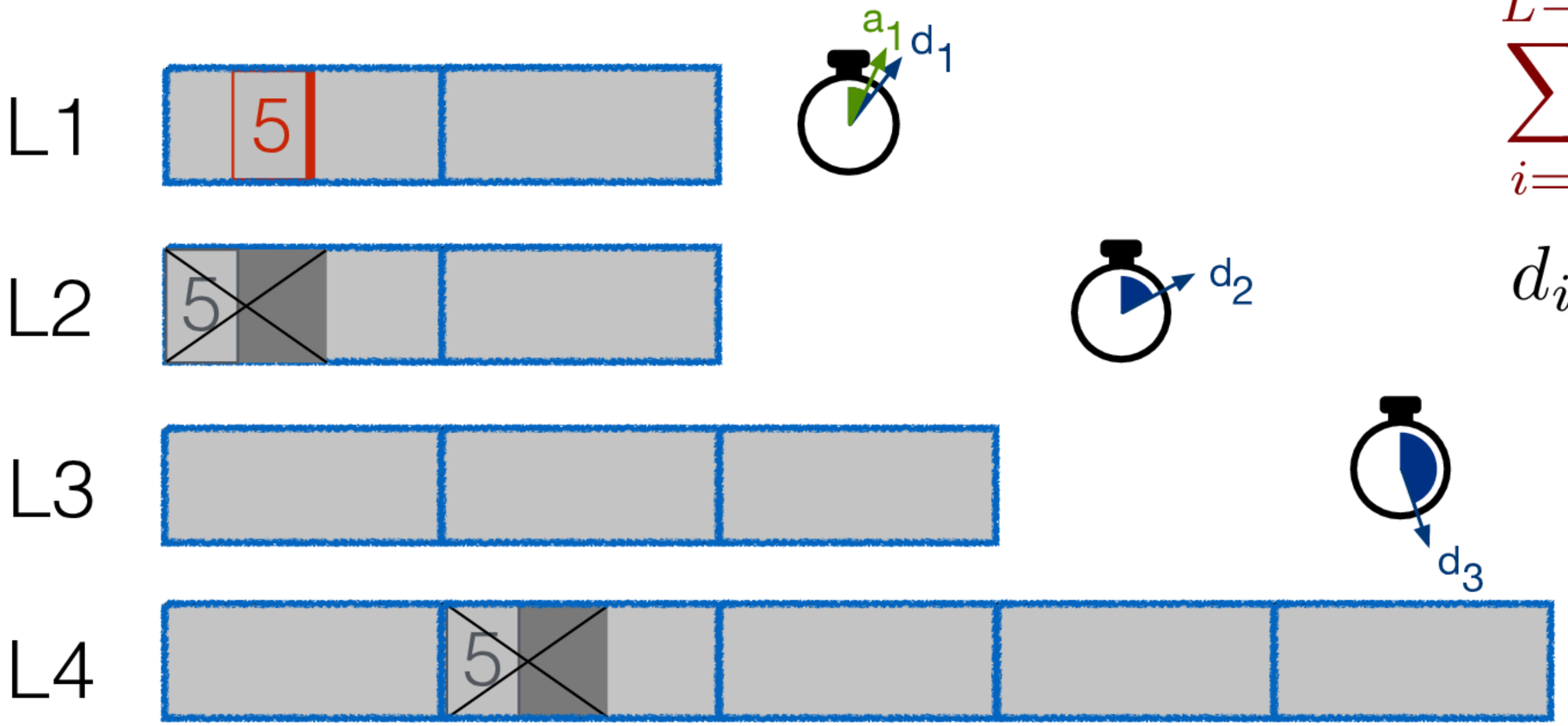



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

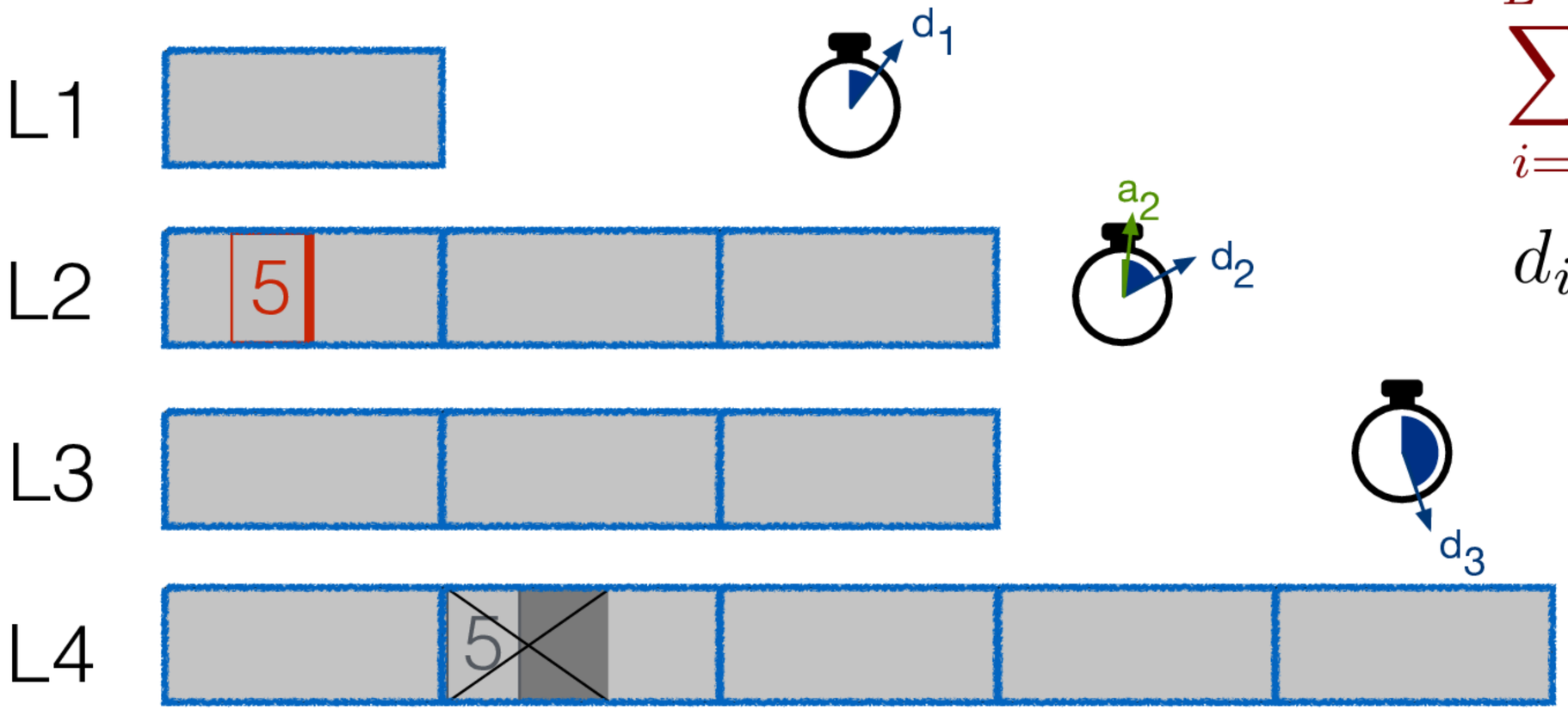



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

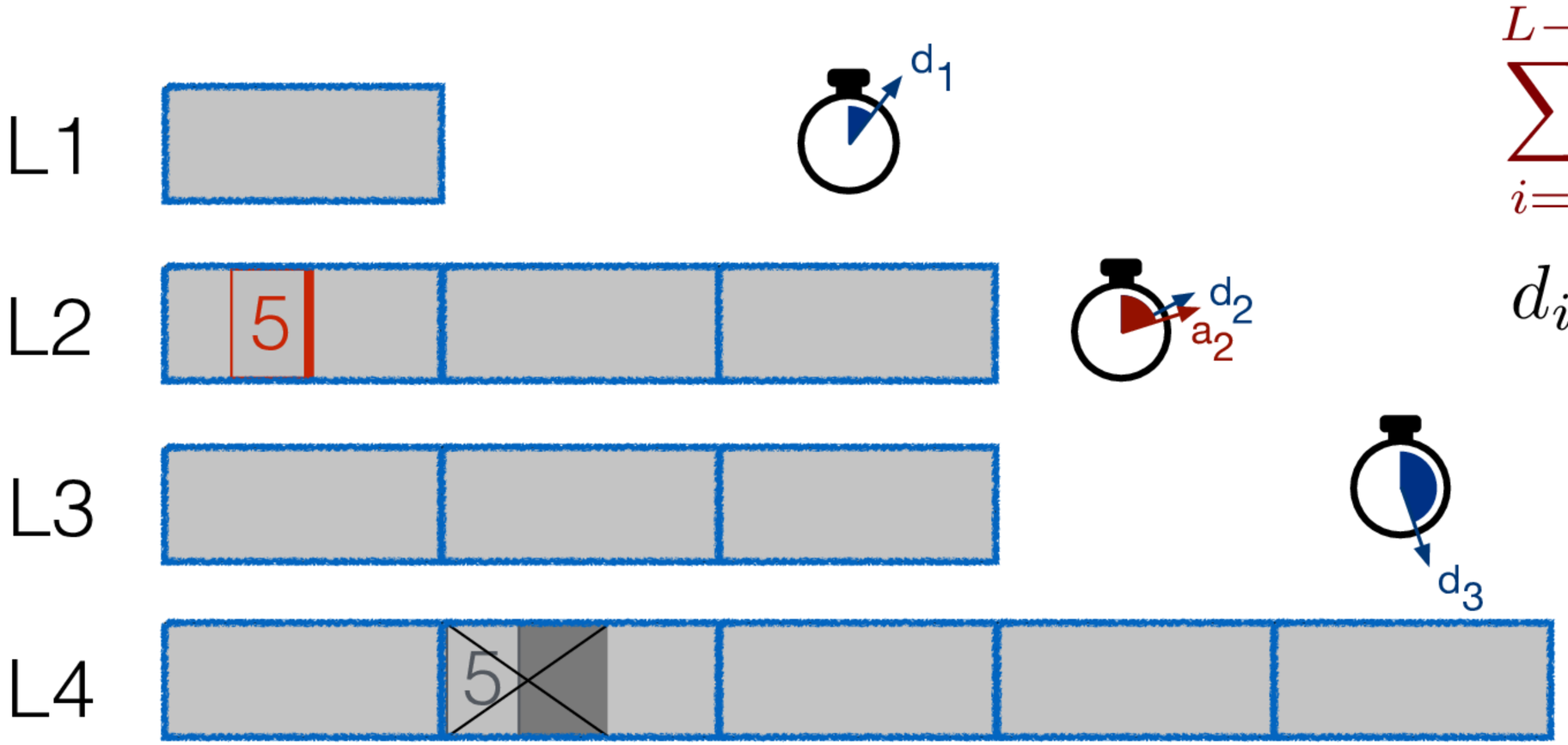



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

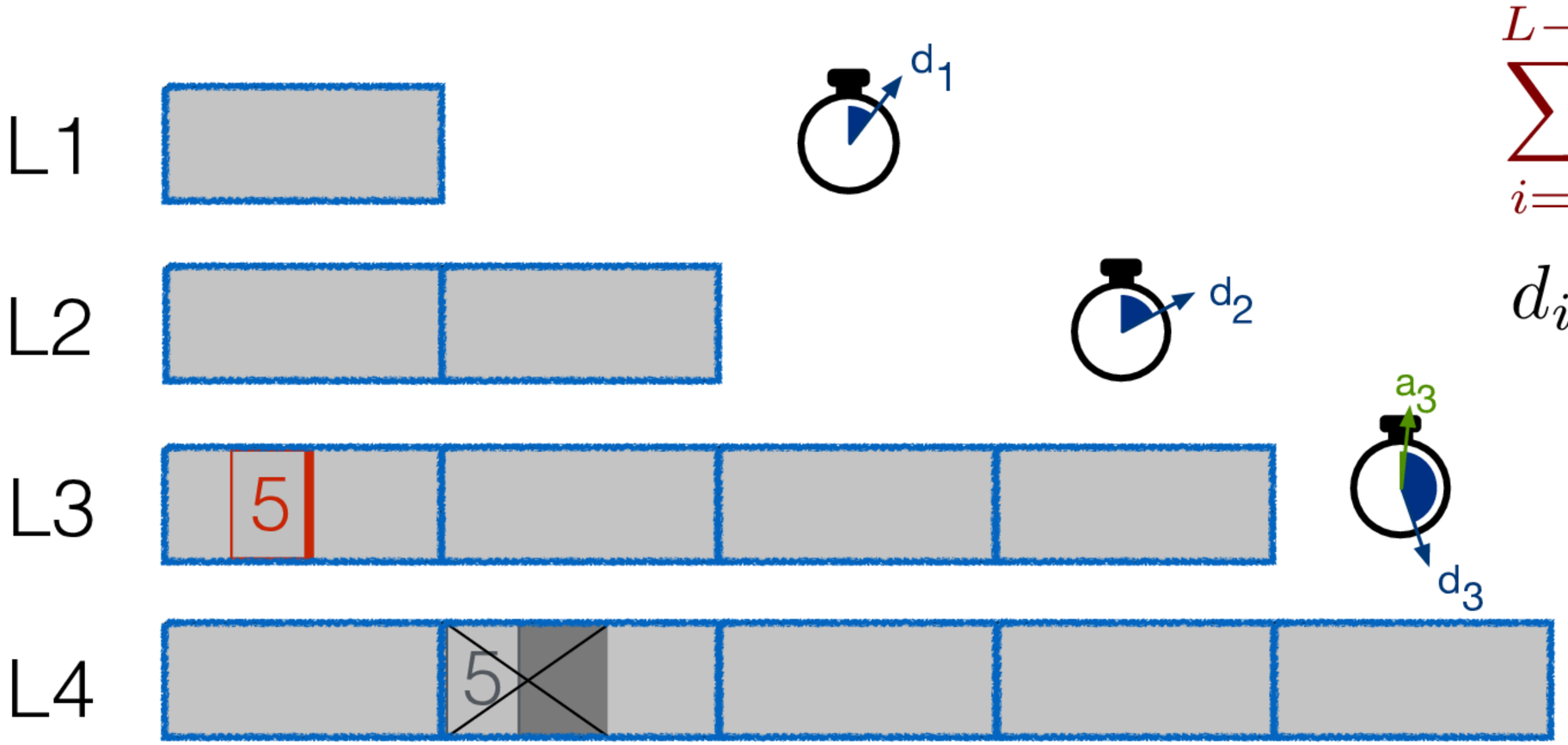



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

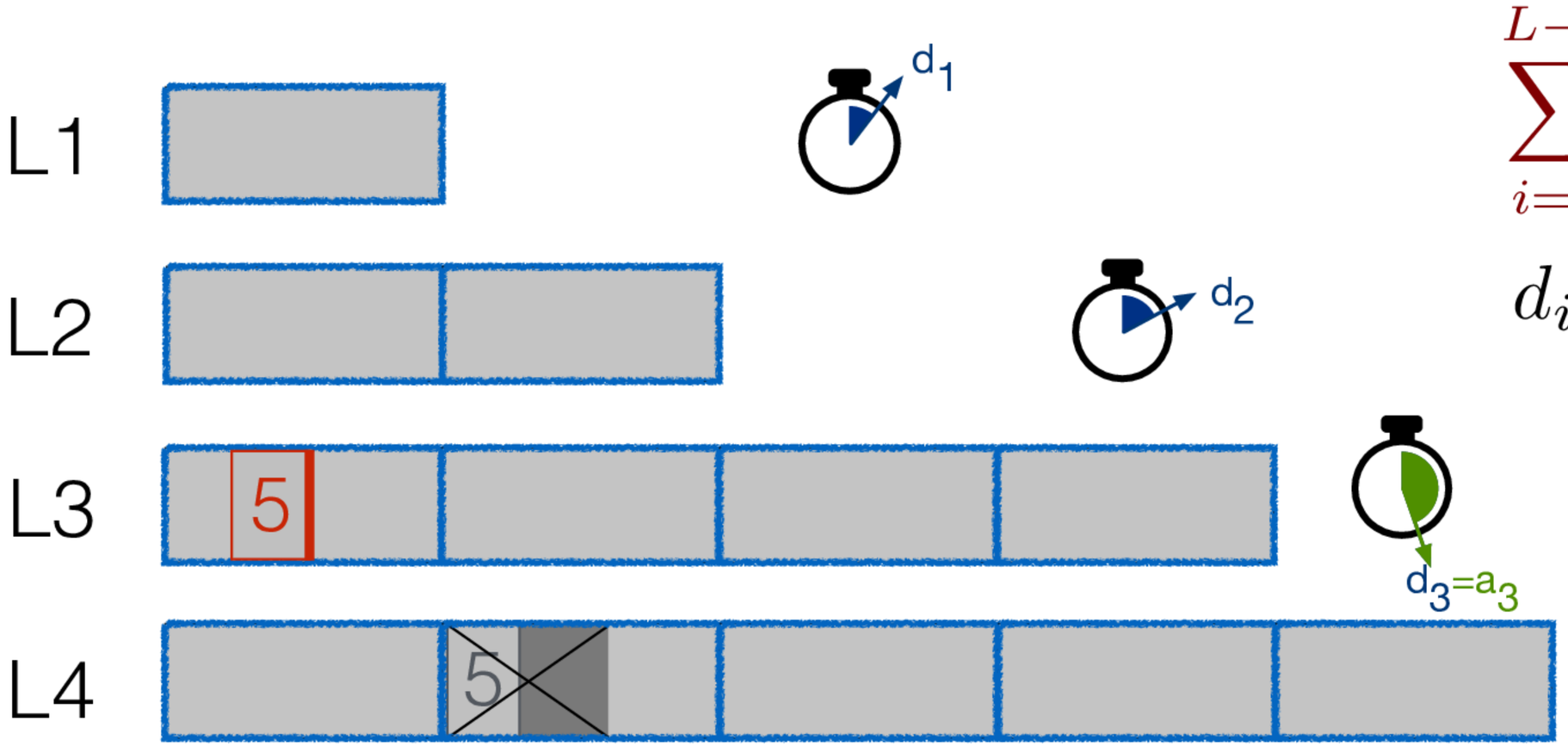



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

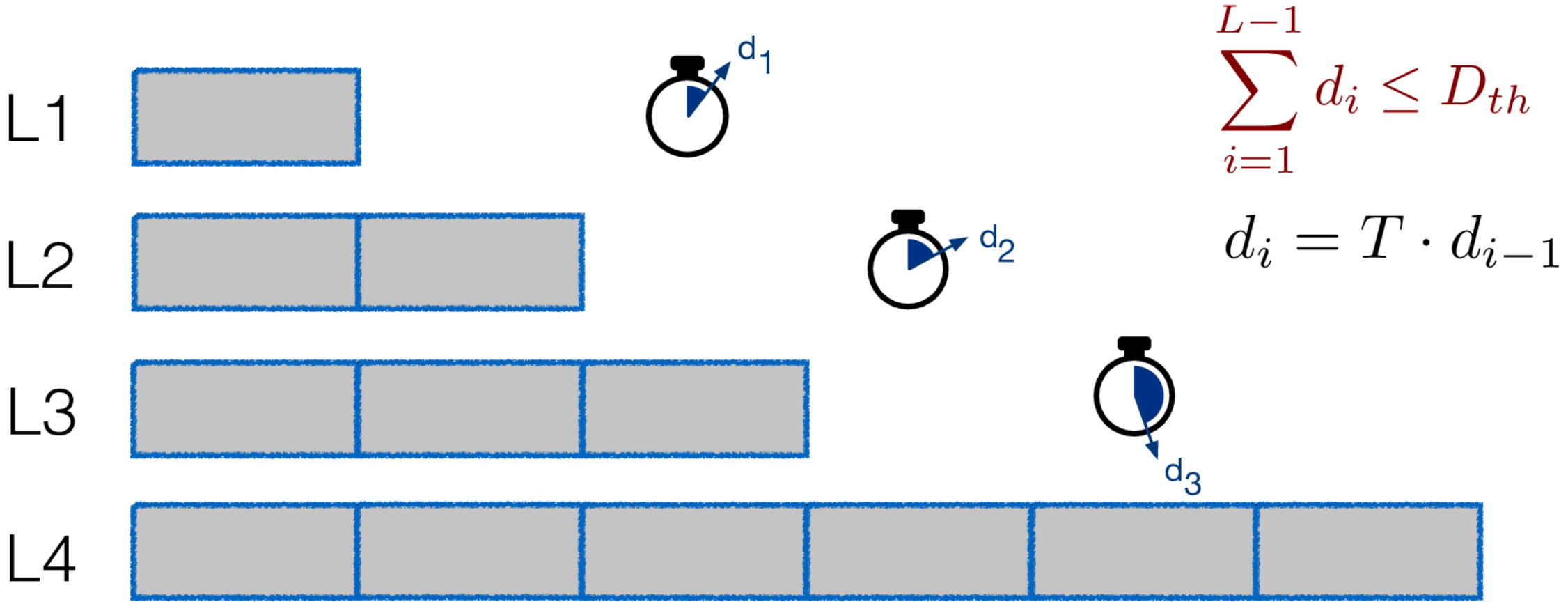



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

# FAst DElete

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

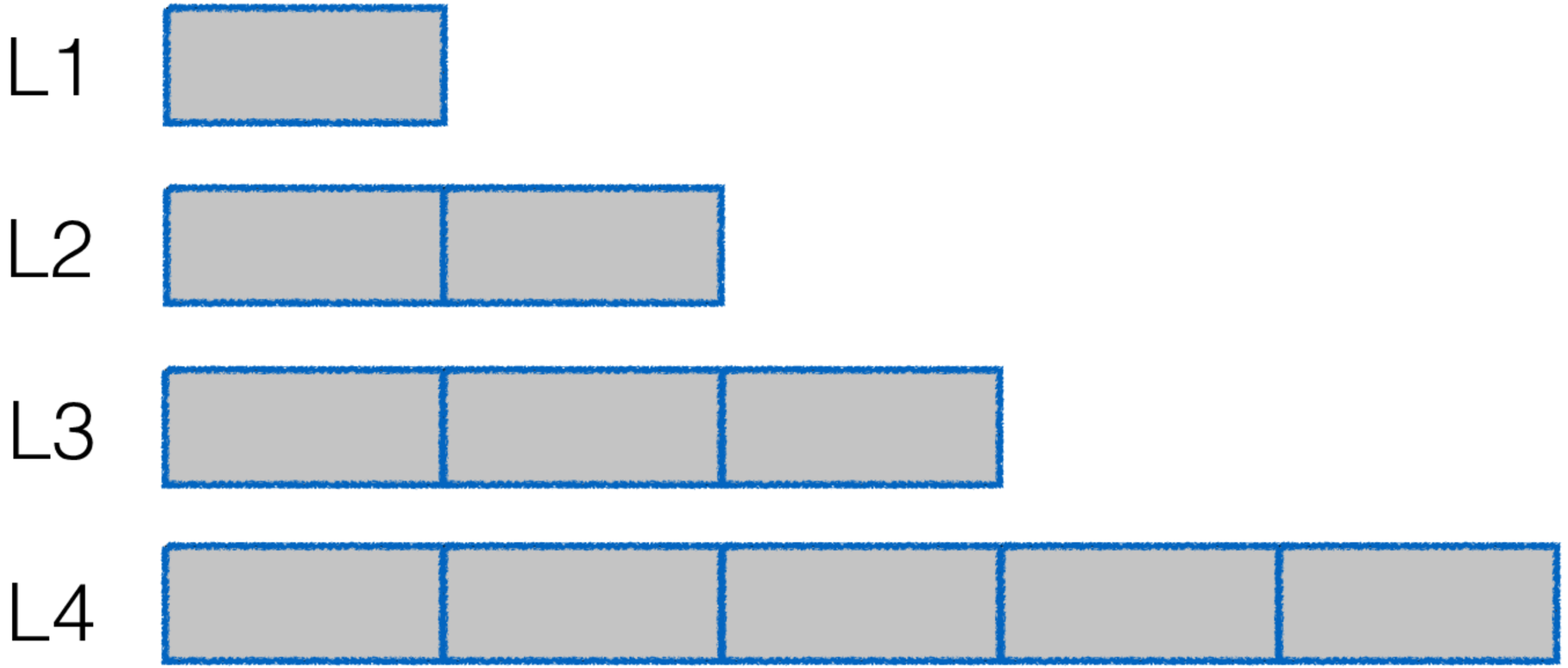



$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

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

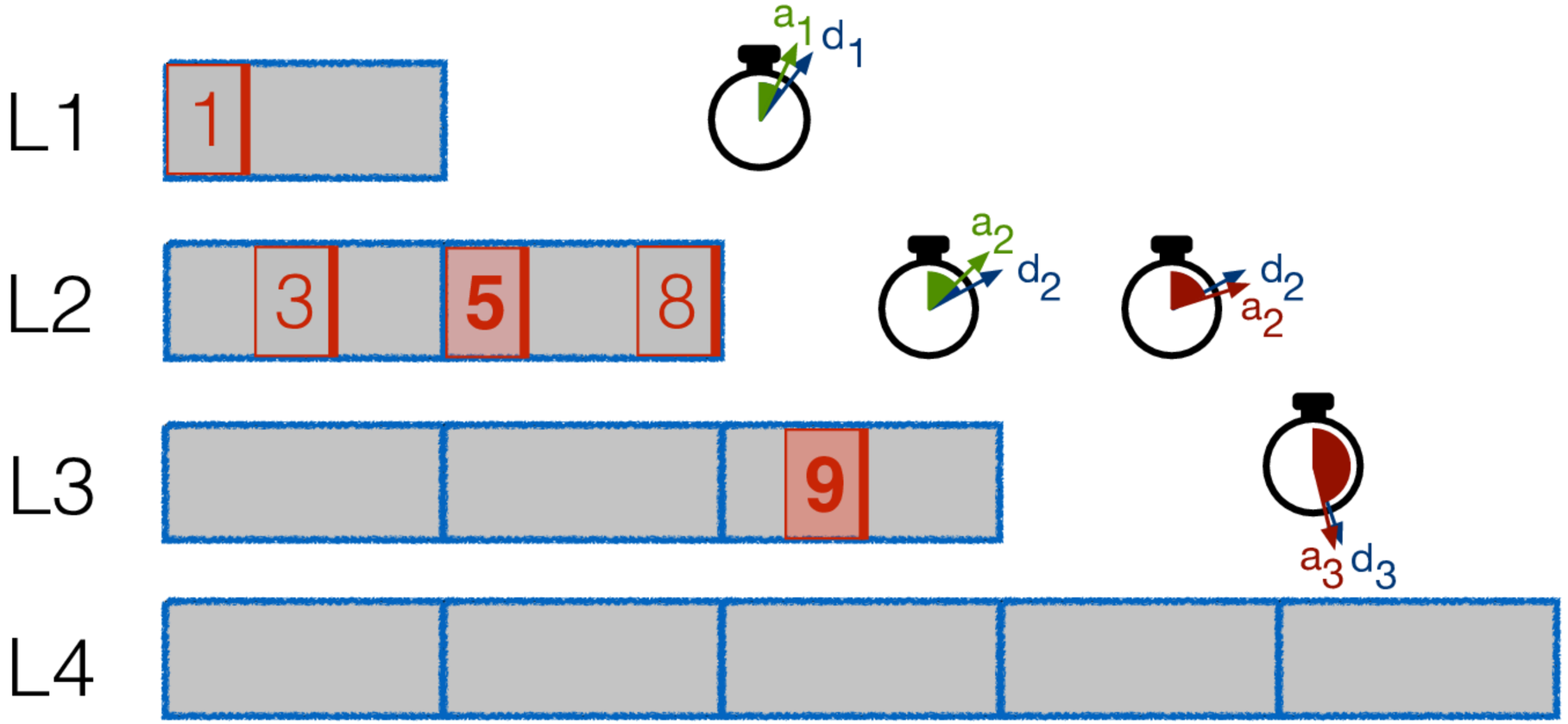
# FAst DElete

breaking ties in practical workloads



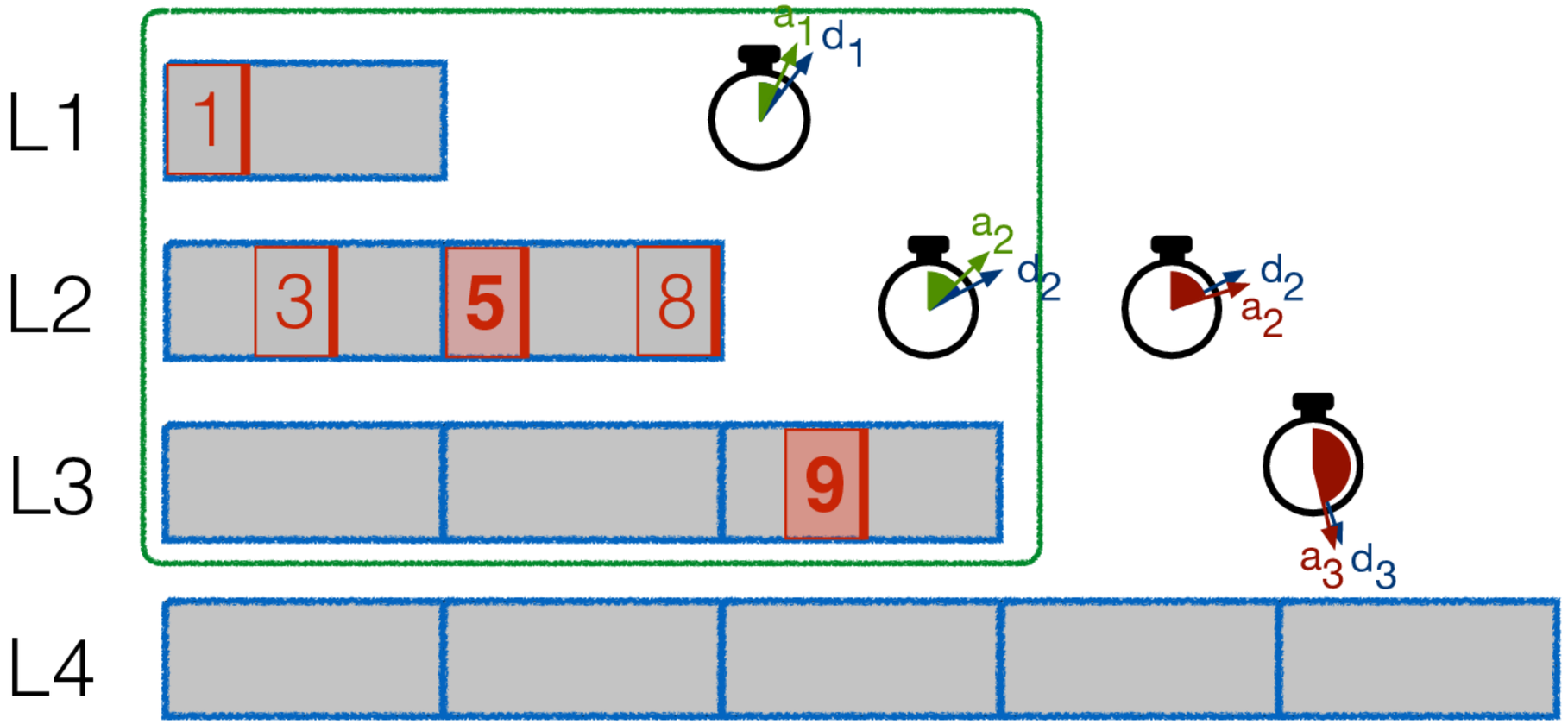
# FAst DElete

breaking ties in practical workloads



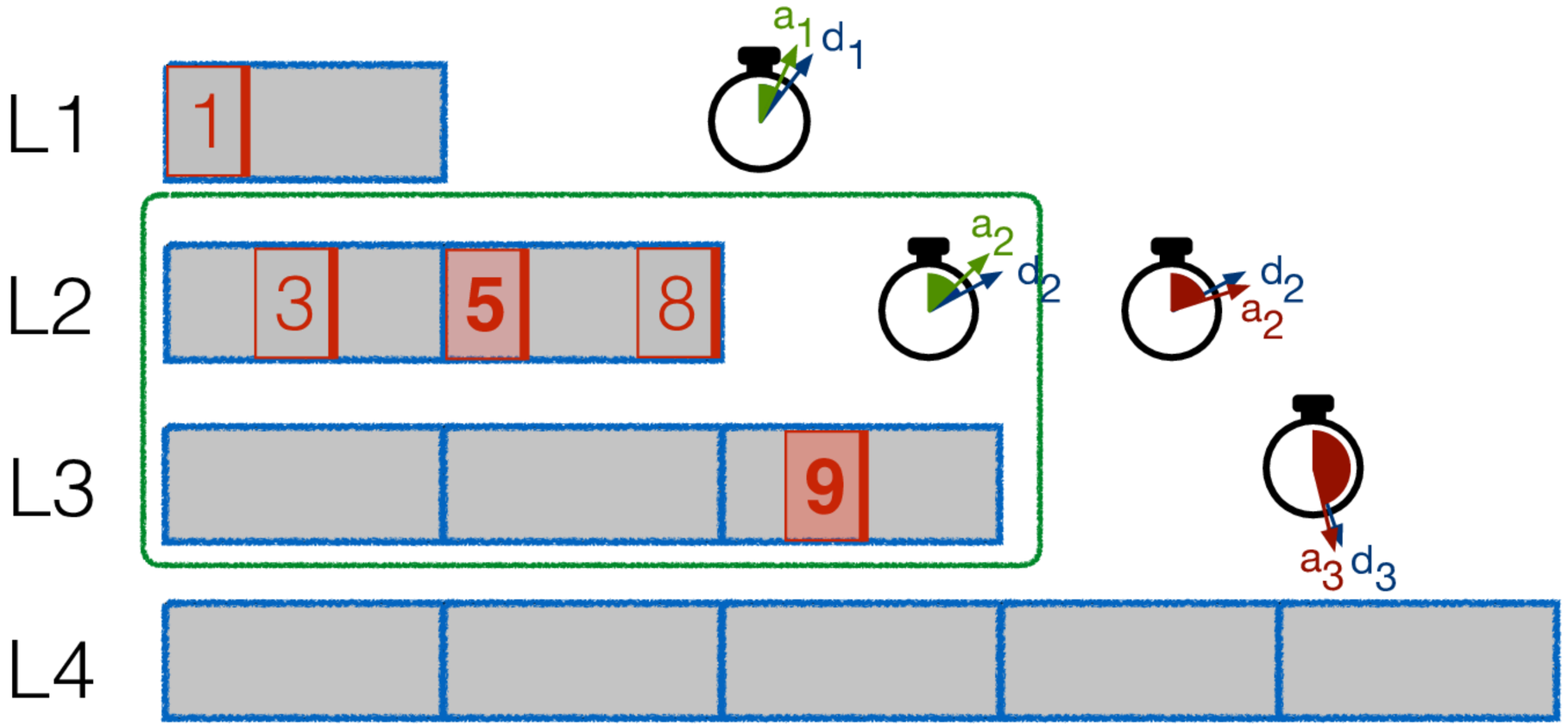
# FAst DElete

breaking ties in practical workloads



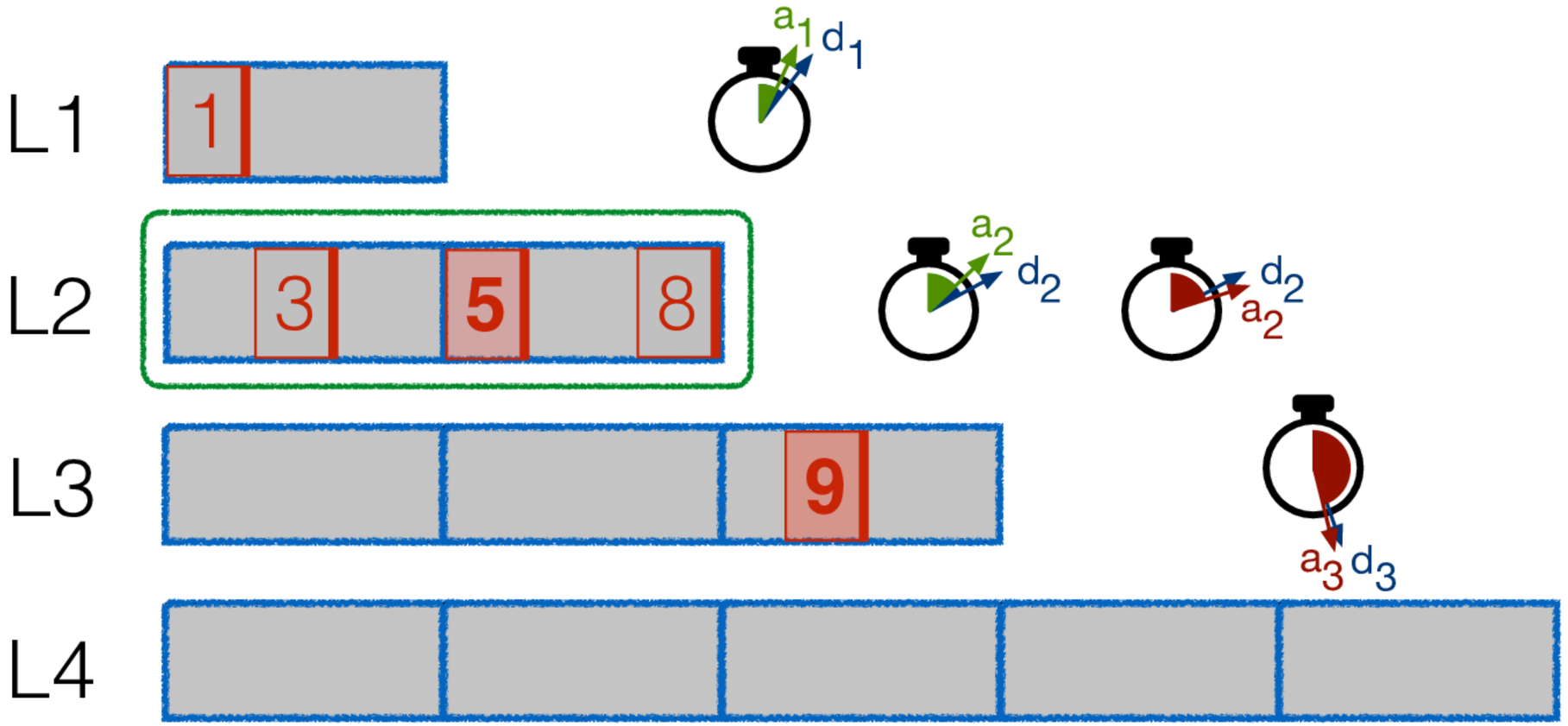
# FAst DElete

breaking ties in practical workloads



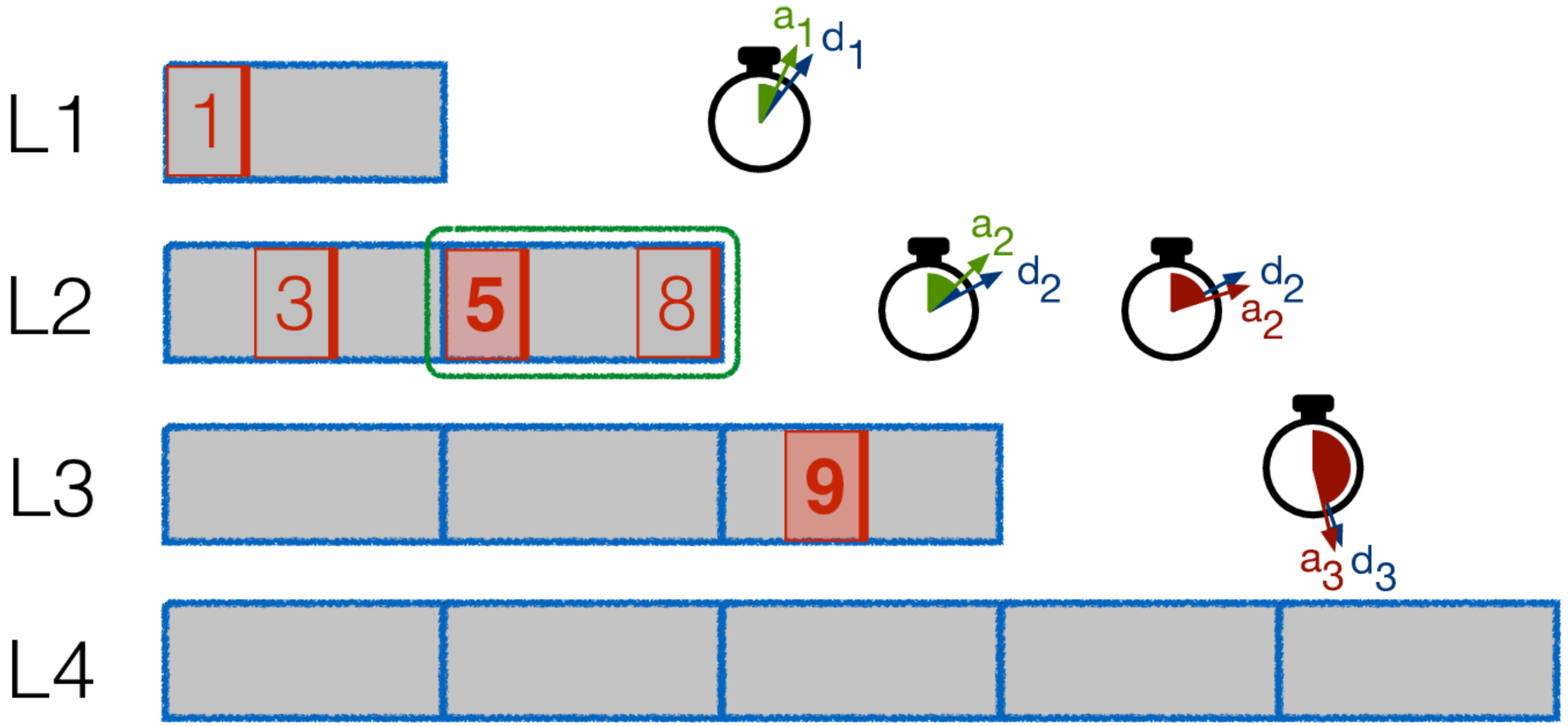
# FAst DElete

breaking ties in practical workloads



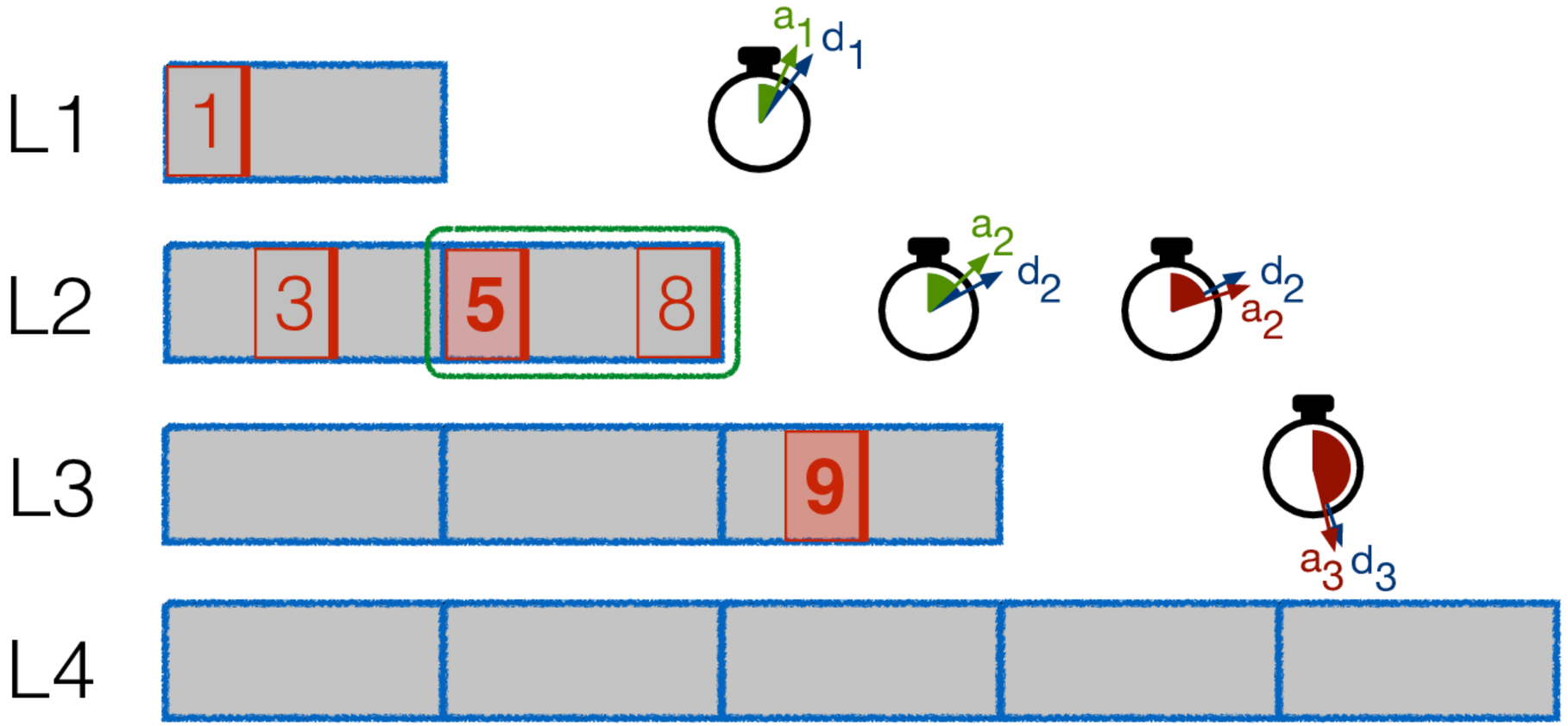
# FAst DElete

## breaking ties in practical workloads



# FAst DElete

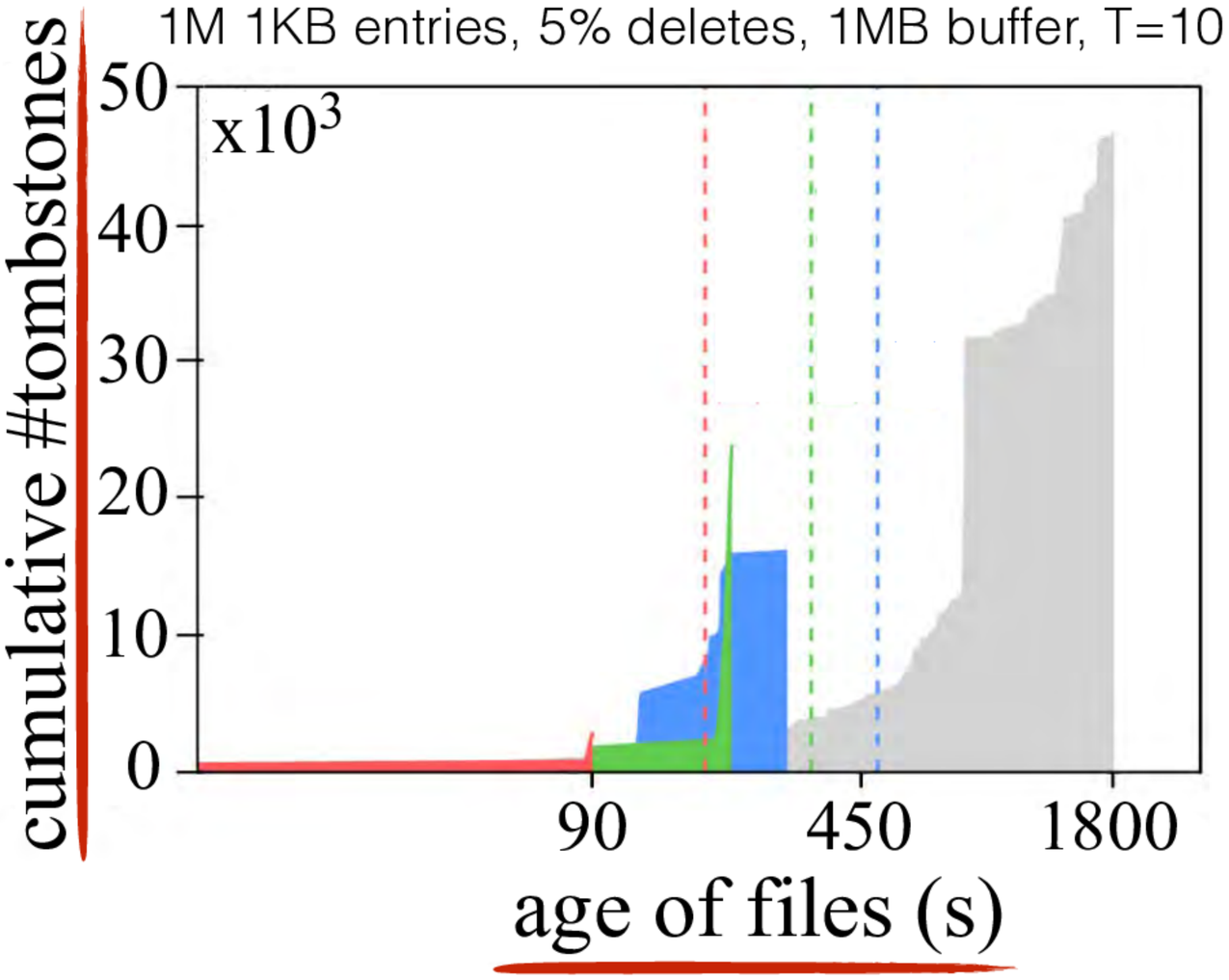
## breaking ties in practical workloads



# FAst DElete

timely delete persistence ✓

**within $D_{th}$**



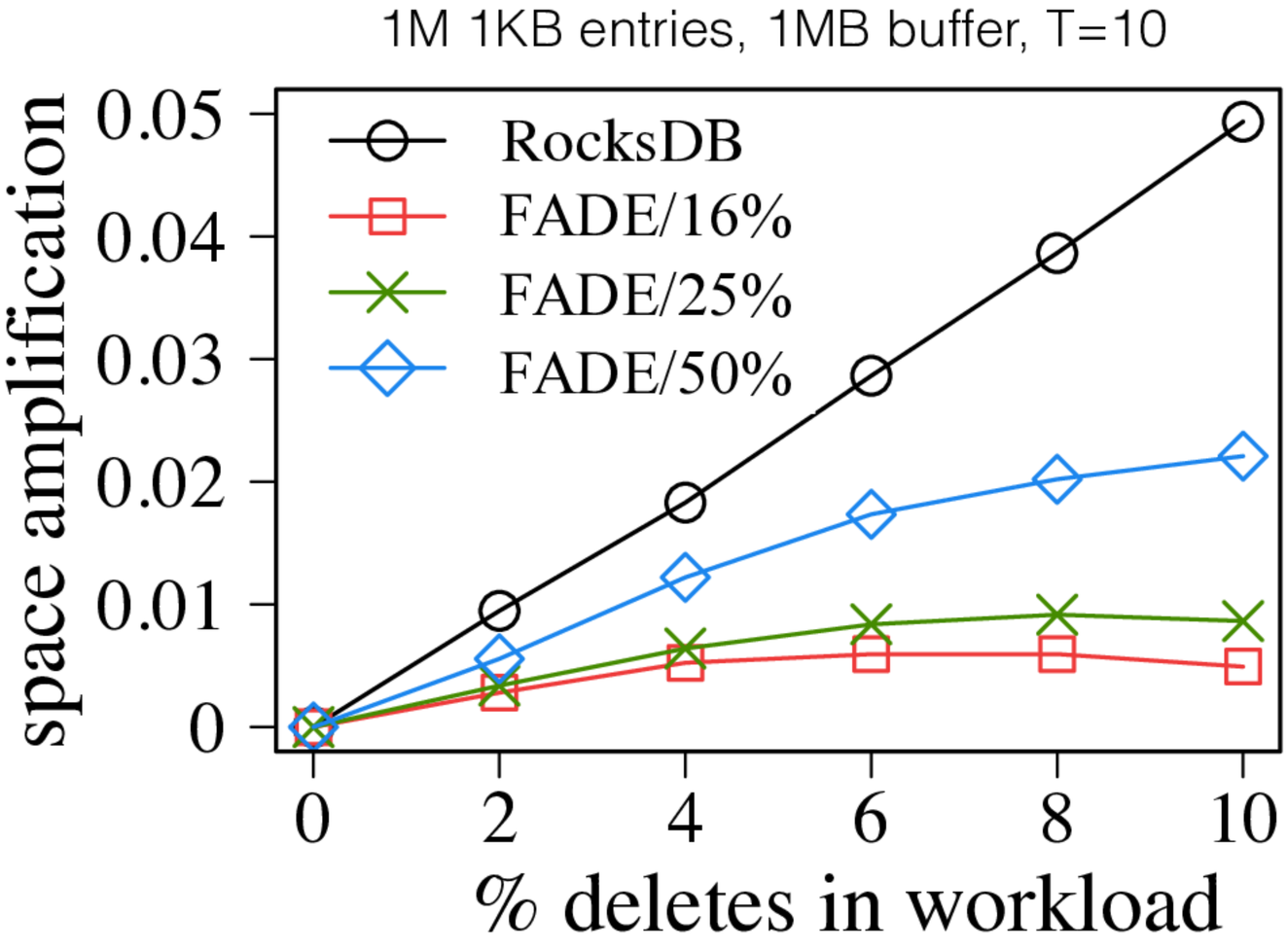
# FAst DElete

reduced space amplification ✓

**2.1 - 9.8 x**

timely delete persistence ✓

**within $D_{th}$**



# FAst DElete

improved read performance ✓

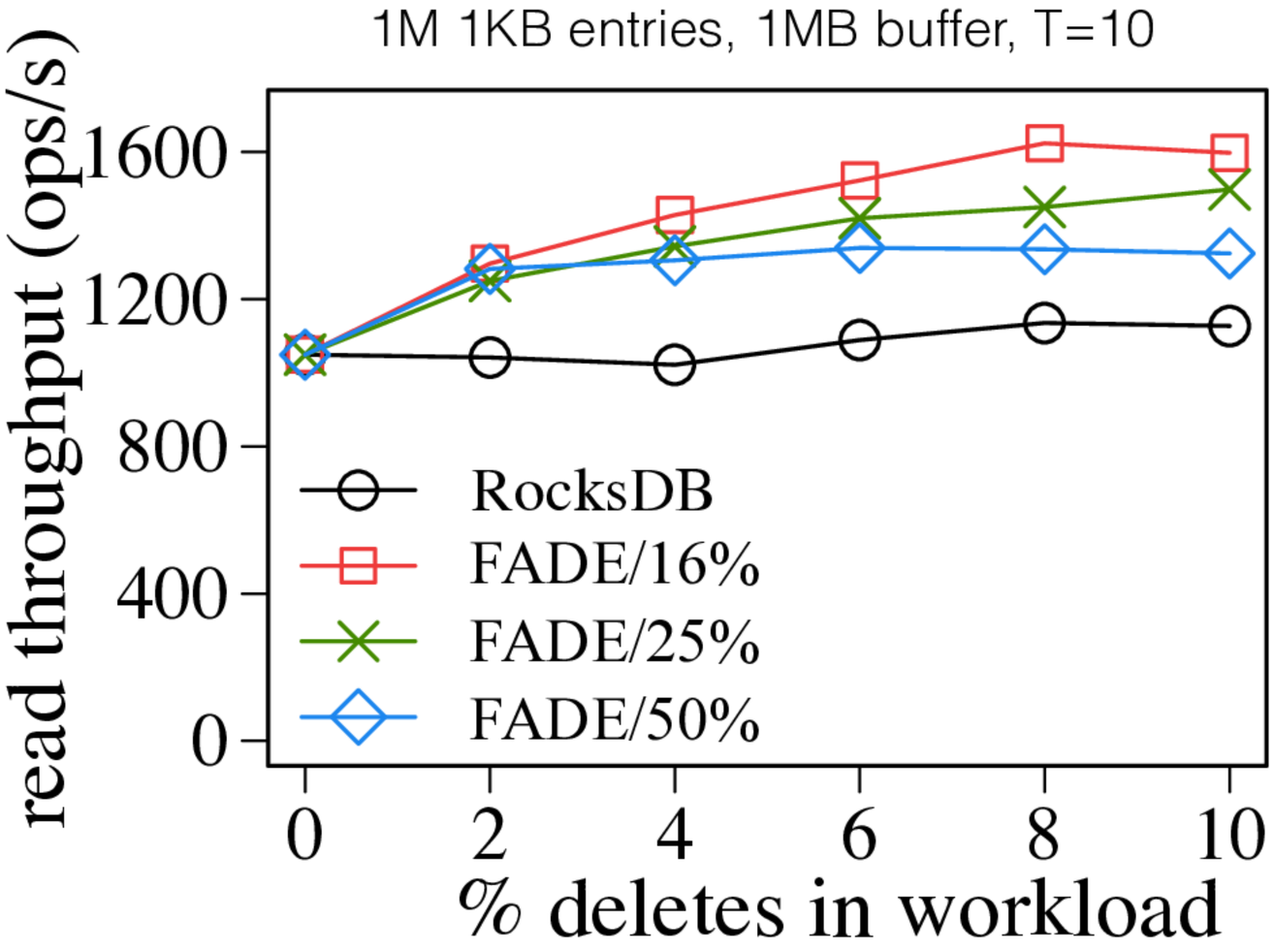
**1.2 - 1.4 x**

reduced space amplification ✓

**2.1 - 9.8 x**

timely delete persistence ✓

**within $D_{th}$**



# FAst DElete

higher write amplification

**4 - 25%**

improved read performance ✓

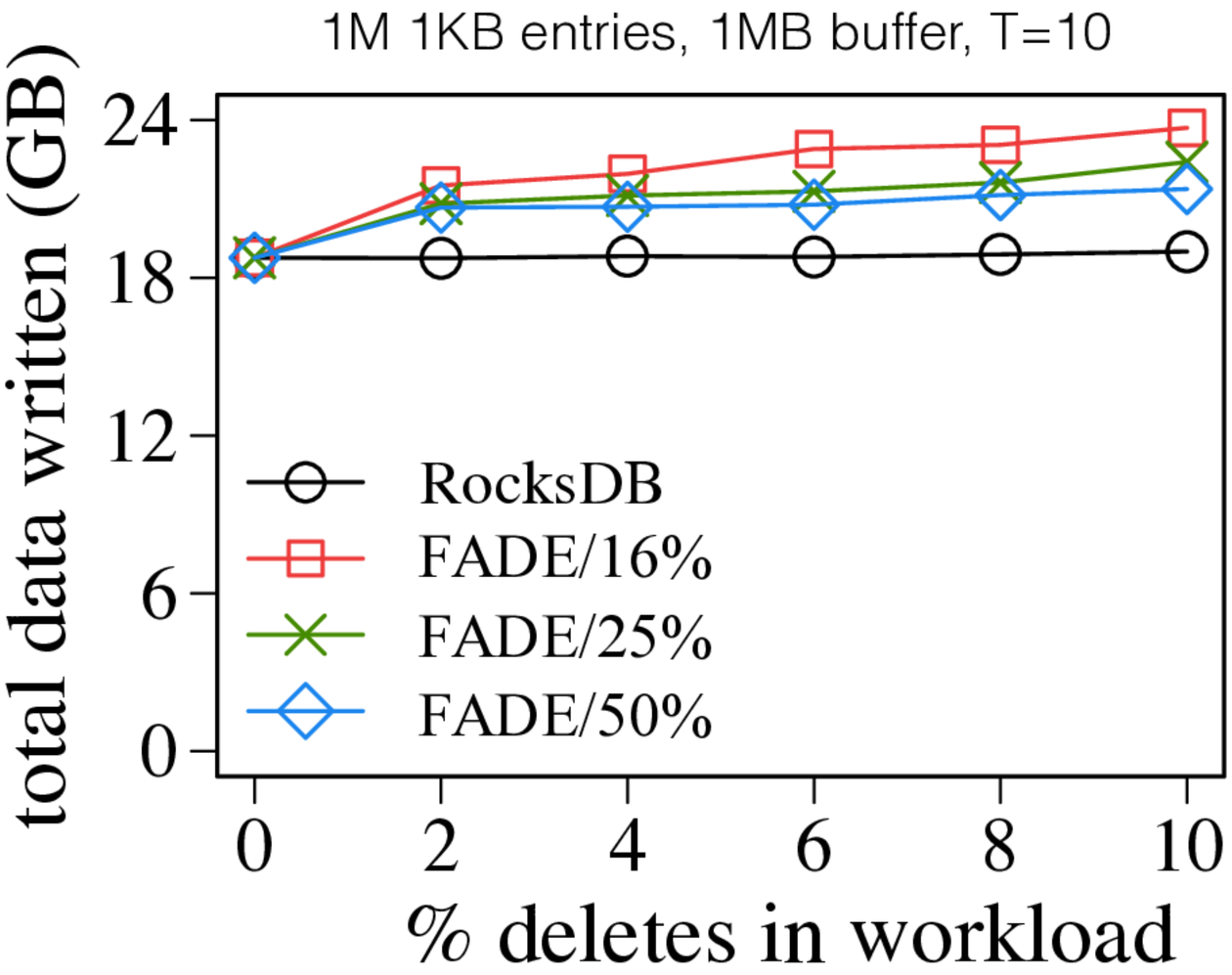
**1.2 - 1.4x**

reduced space amplification ✓

**2.1 - 9.8x**

timely delete persistence ✓

**within $D_{th}$**



# FAst DElete

higher write amplification

**4 - 25%**

improved read performance ✓

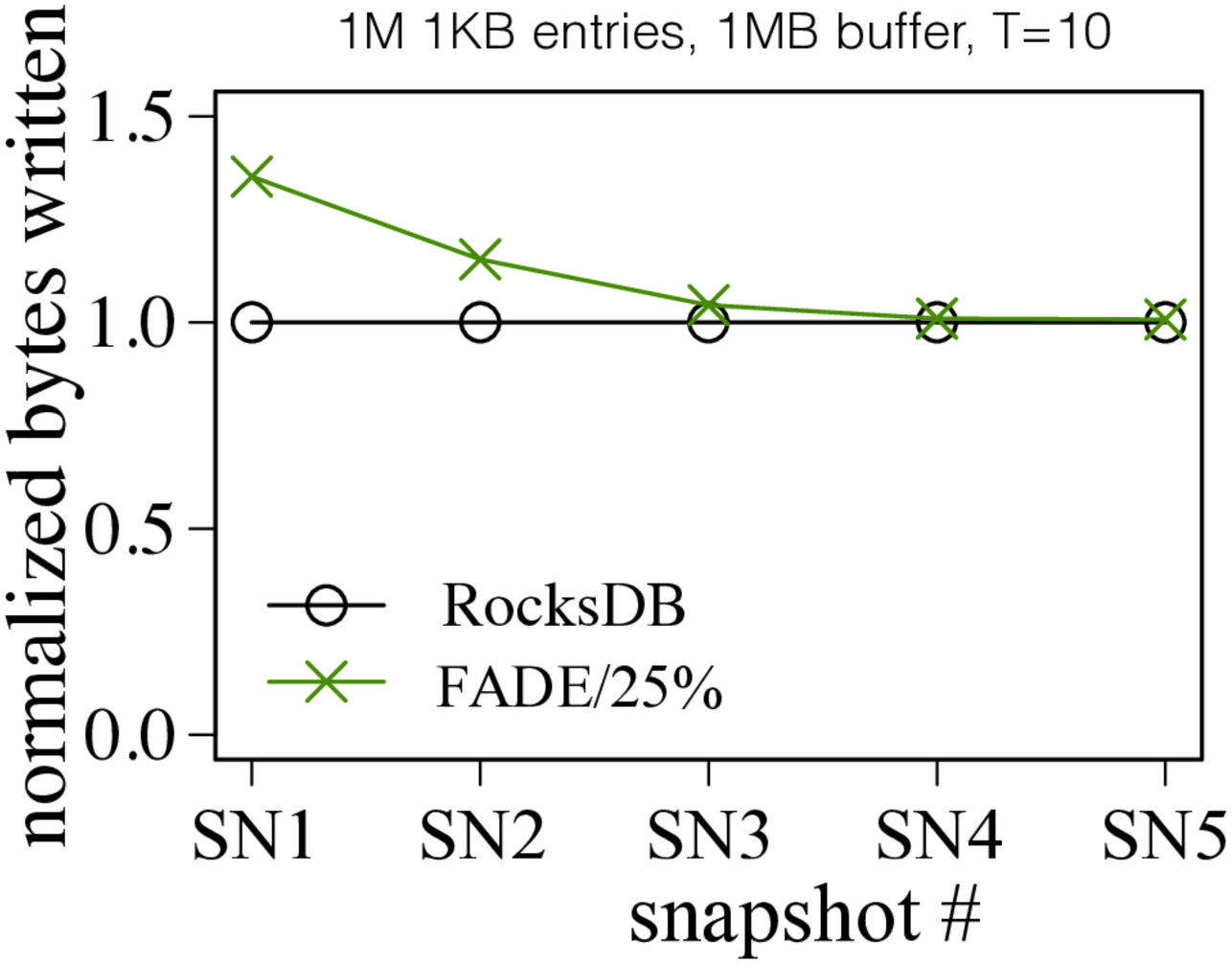
**1.2 - 1.4x**

reduced space amplification ✓

**2.1 - 9.8x**

timely delete persistence ✓

**within $D_{th}$**



# FAst DElete

higher write amplification

0.7 %

improved read performance ✓

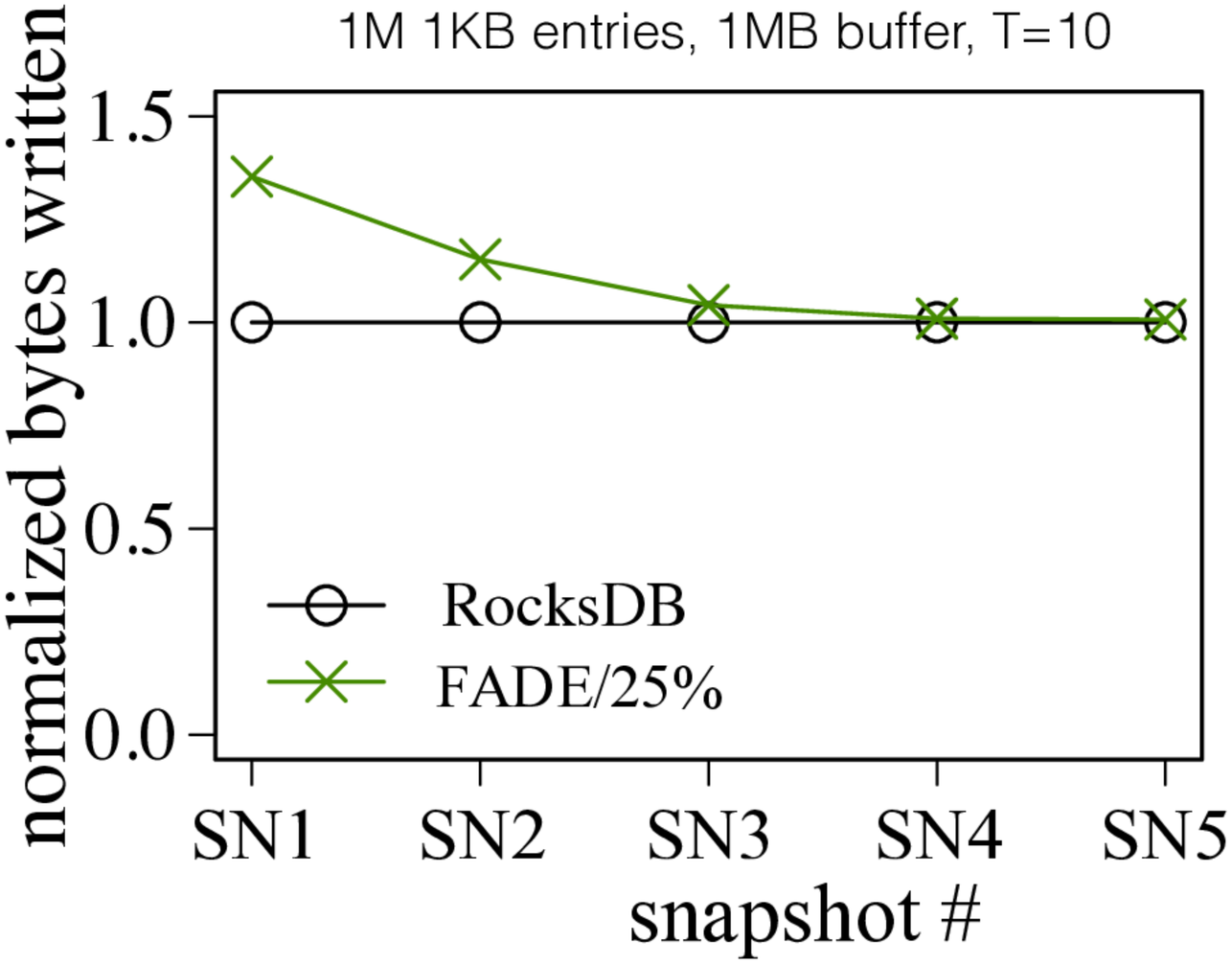
1.2 - 1.4 x

reduced space amplification ✓

2.1 - 9.8 x

timely delete persistence ✓

within $D_{th}$



# the solution

FAst DElete

higher write amplification

improved read performance

reduced space amplification

timely delete persistence

**FADE**

latency spikes

superfluous IOs

**KiWi**

# the solution

FAst DElete

higher write amplification

read performance

reduced space amplification

timely delete persistence

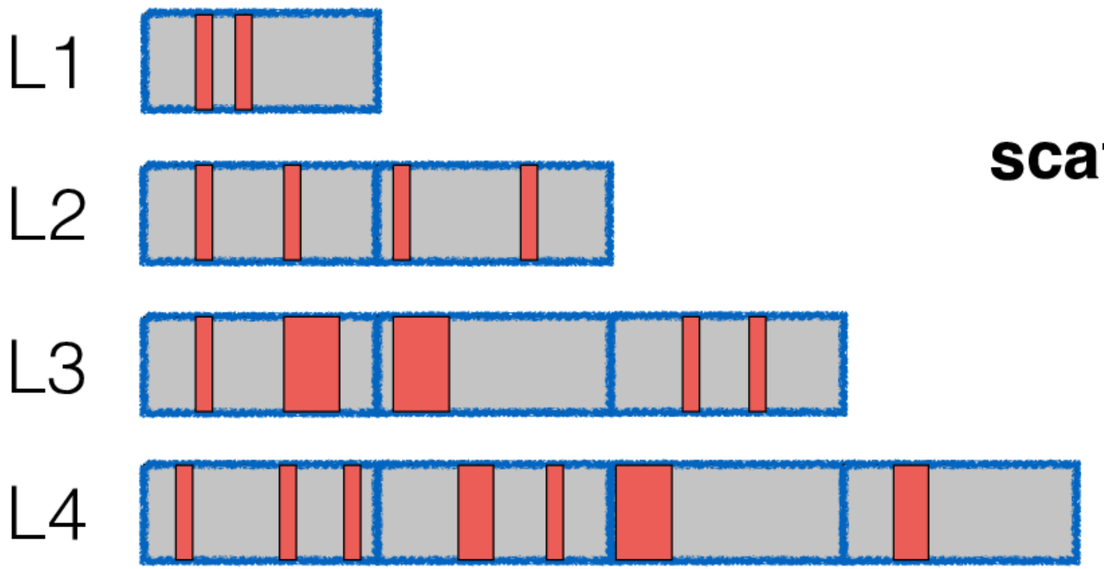
latency spikes

FADE

KiWi

# Key Weaving storage layout

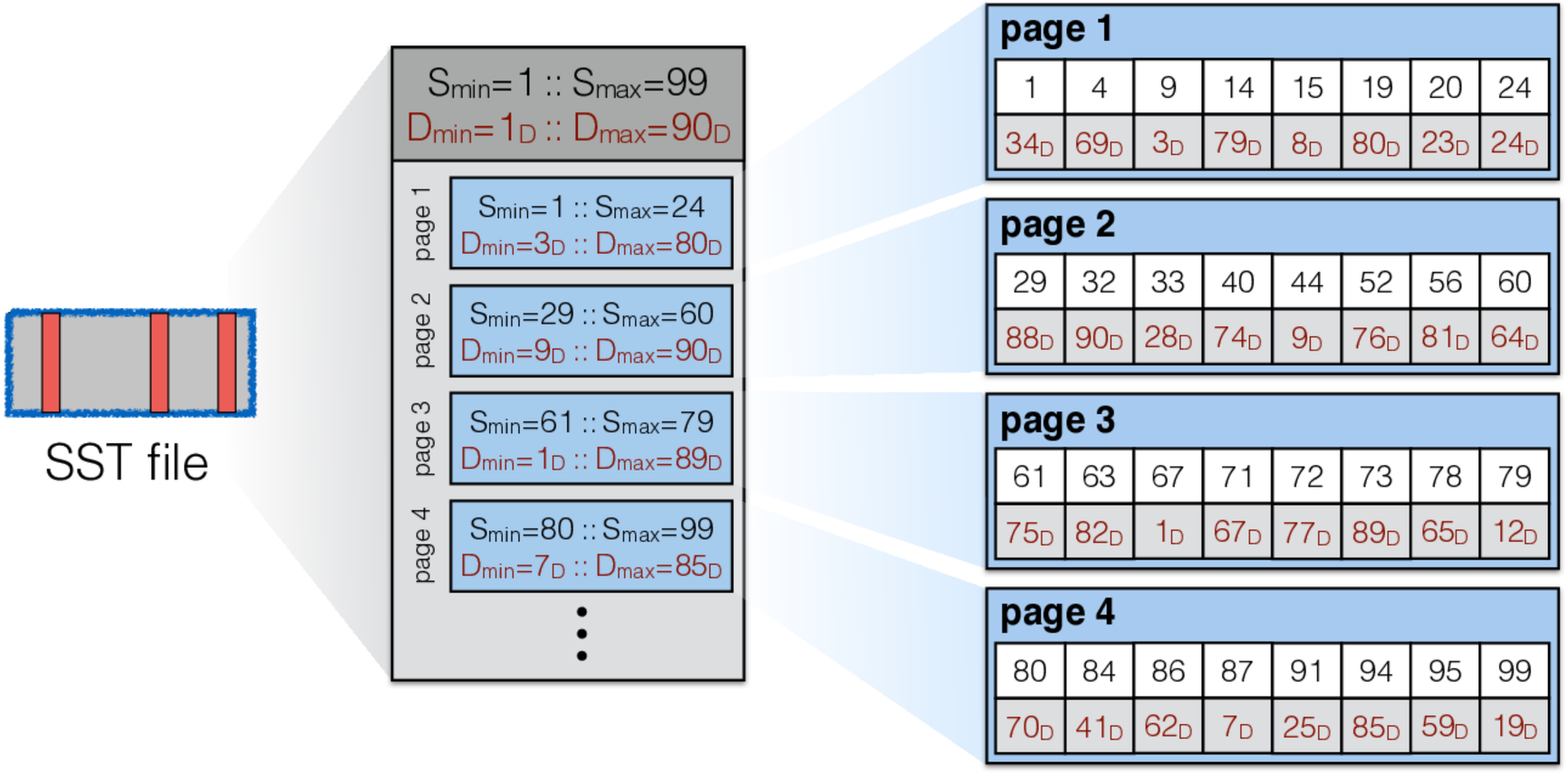
delete all entries older than: **D days**




**scattered occurrences**

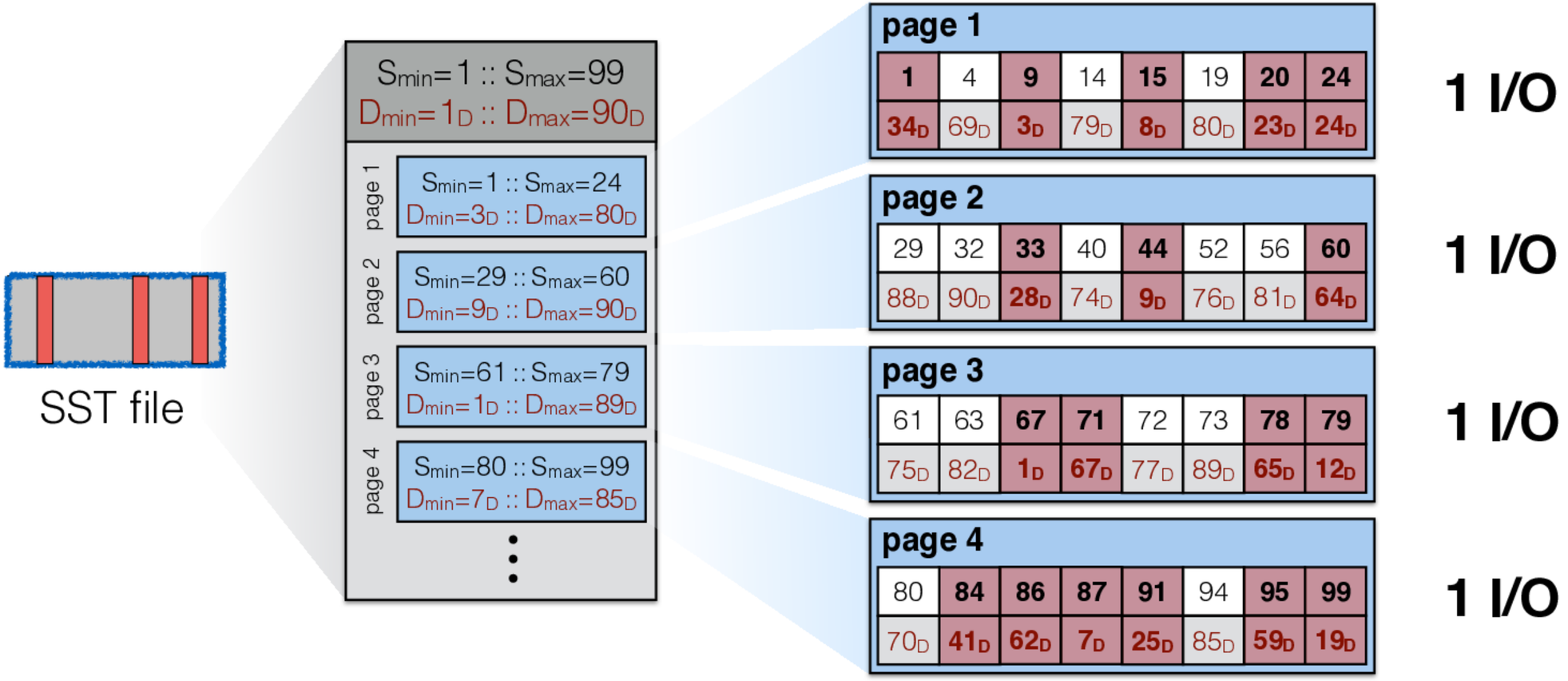
# Key Weaving storage layout

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



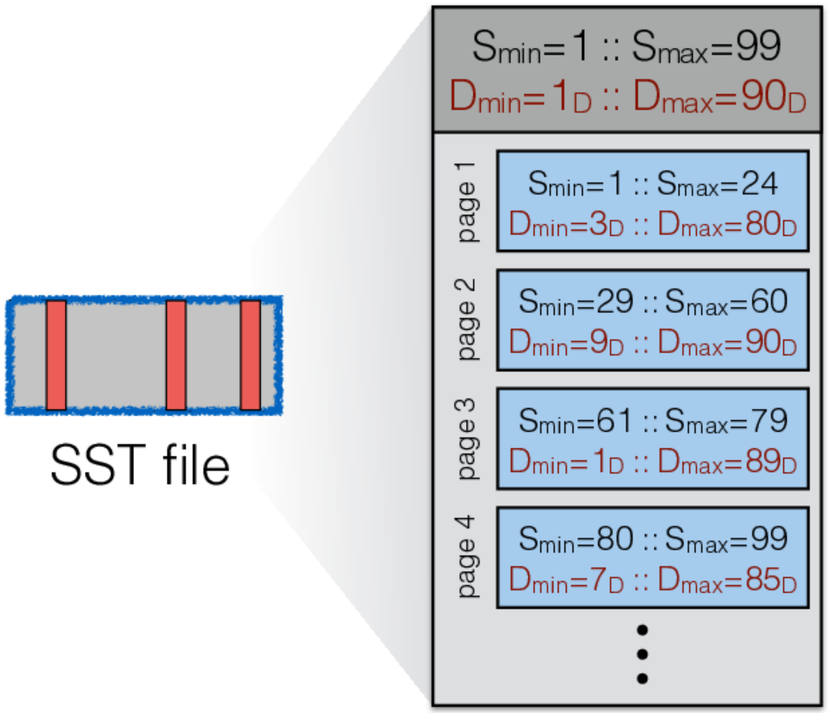
# Key Weaving storage layout

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



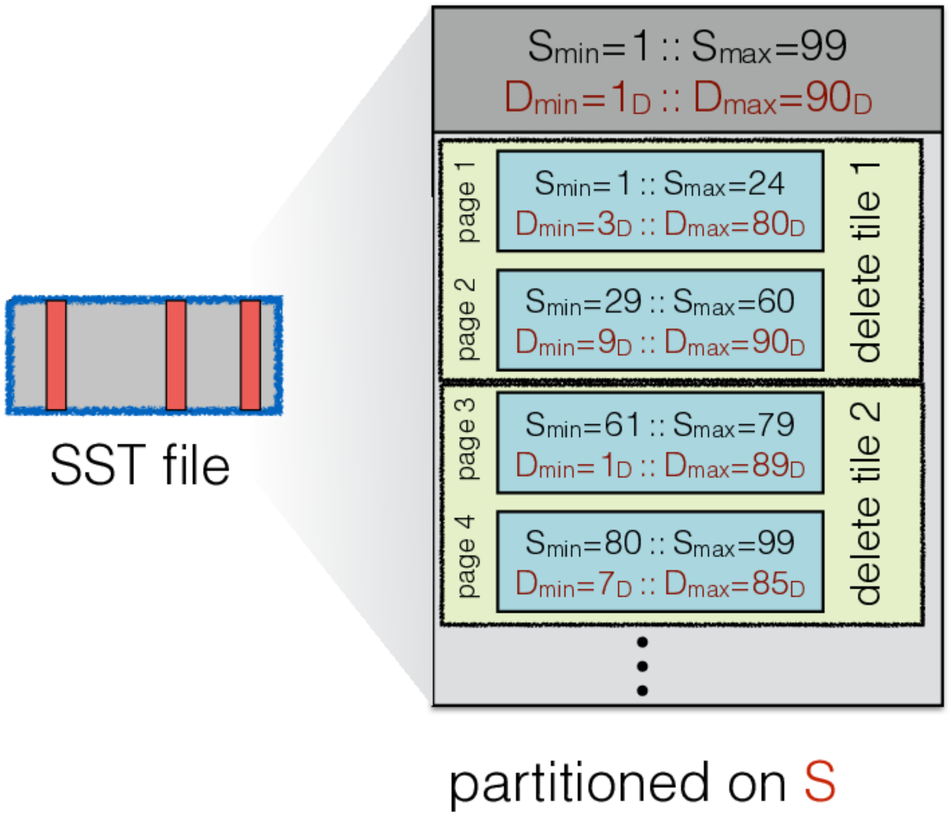
# Key Weaving storage layout

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



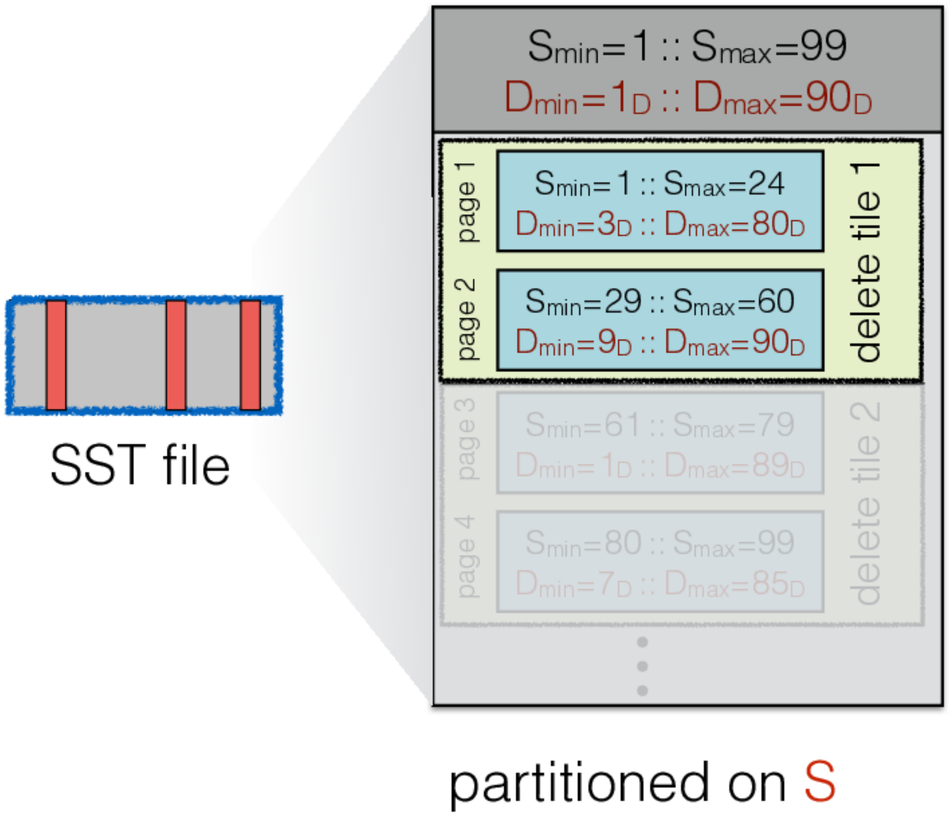
# Key Weaving storage layout

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



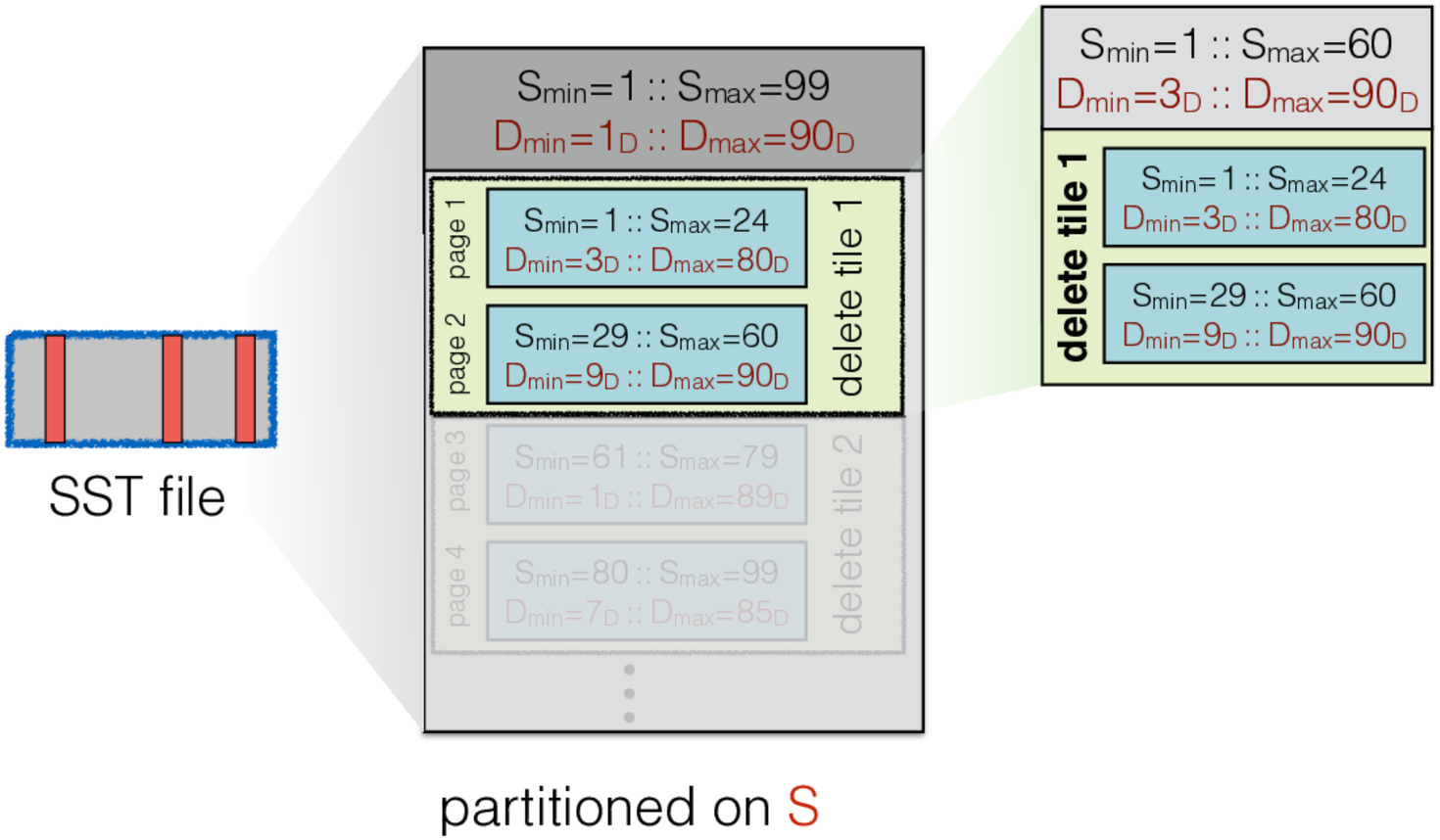
# Key Weaving storage layout

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



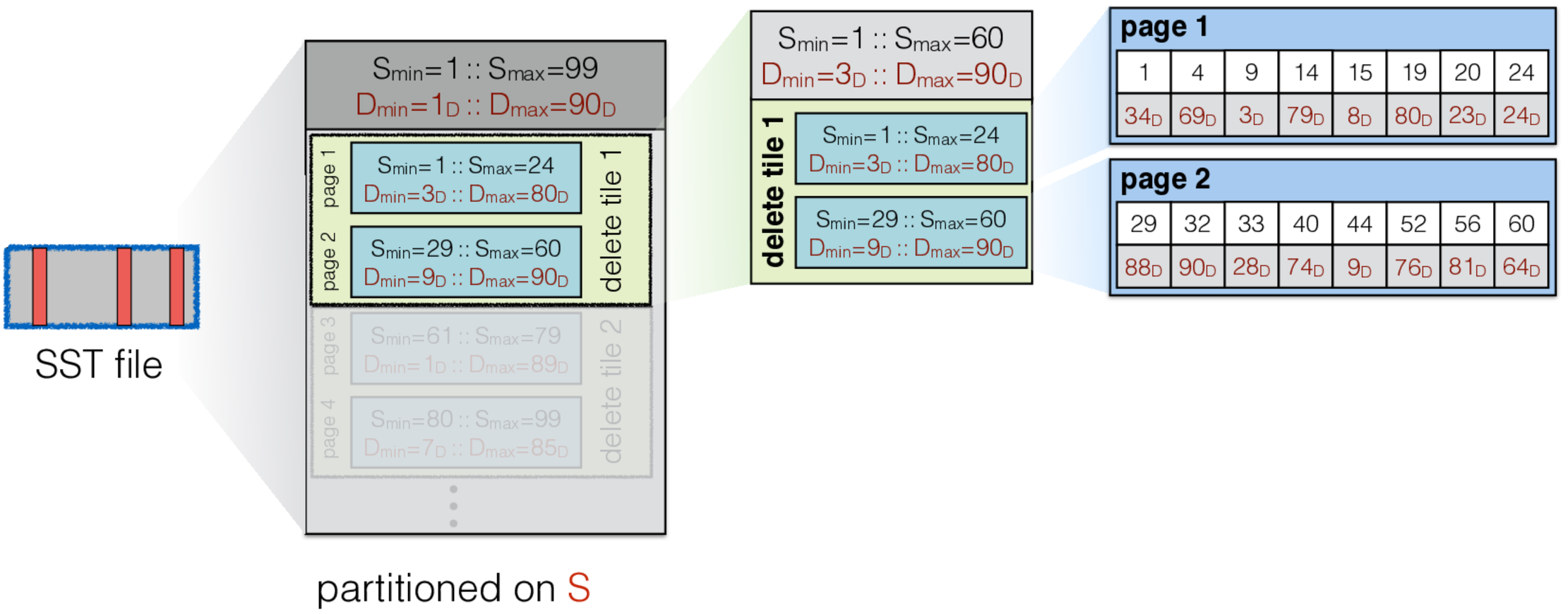
# Key Weaving storage layout

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



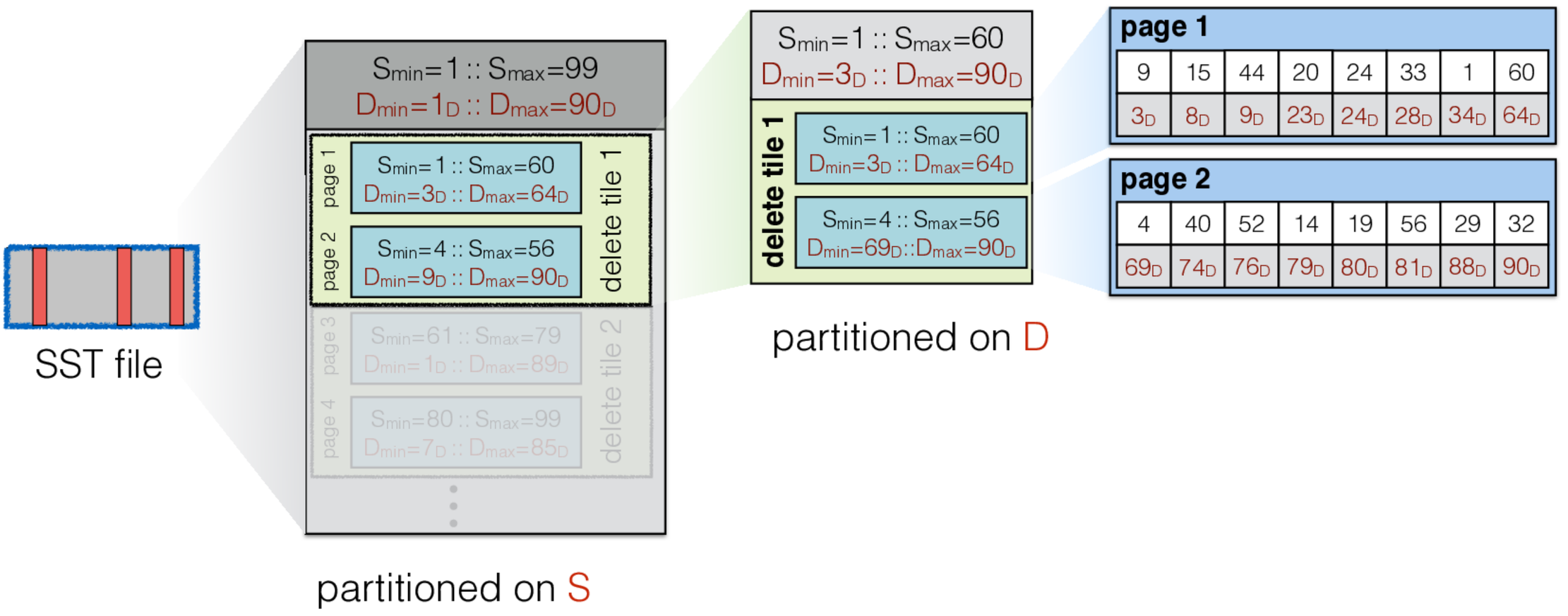
# Key Weaving storage layout

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



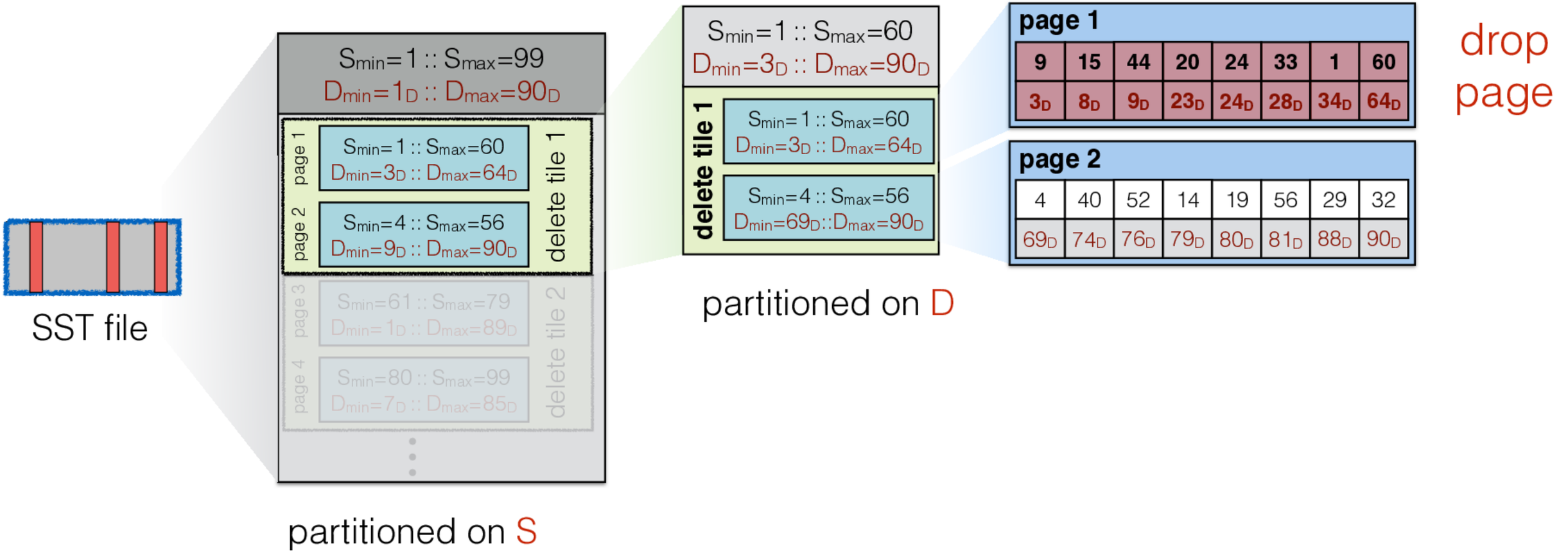
# Key Weaving storage layout

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



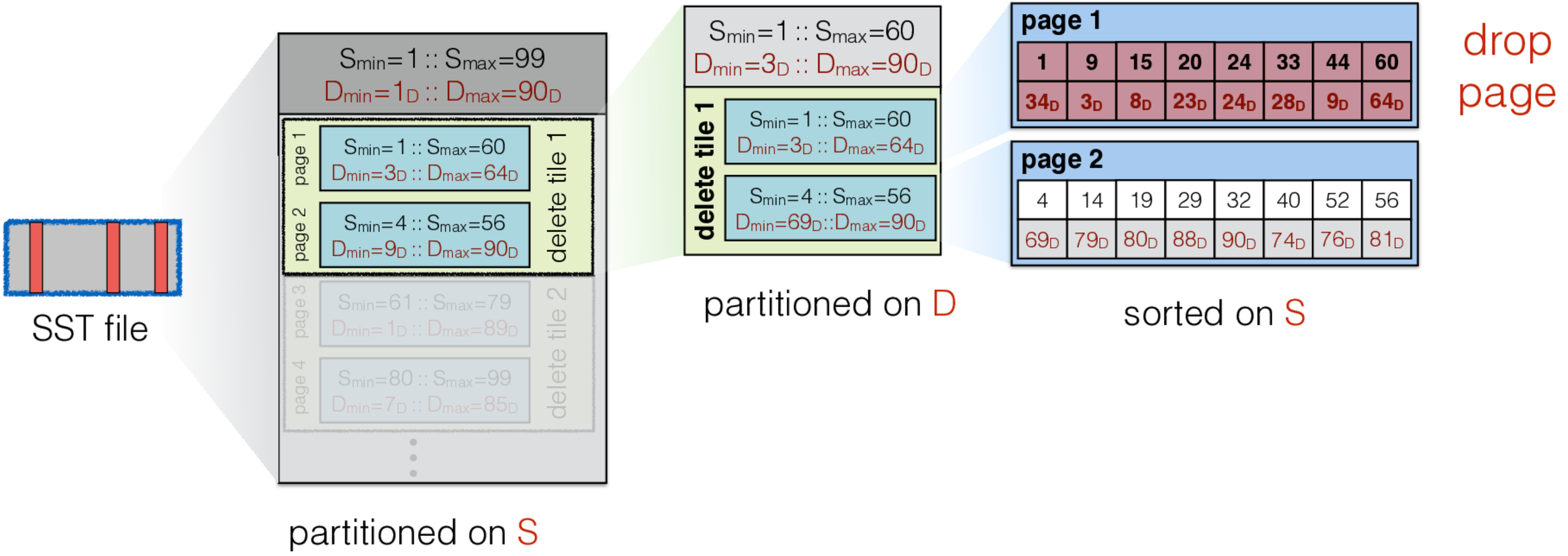
# Key Weaving storage layout

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



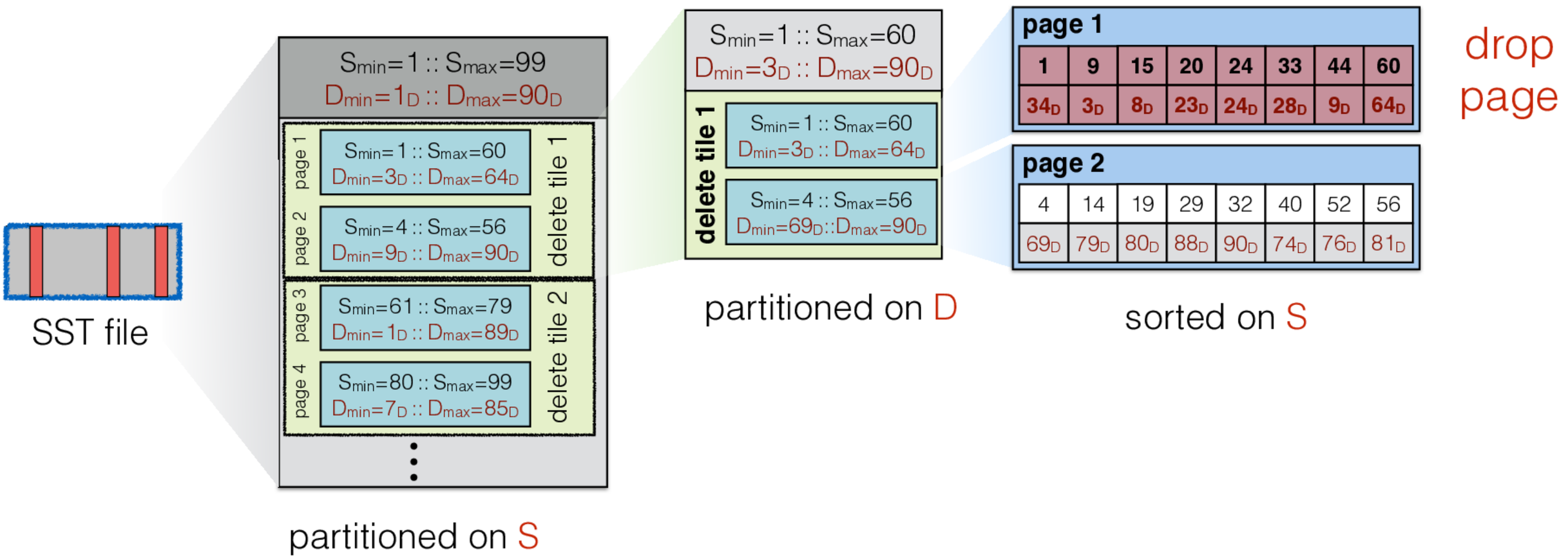
# Key Weaving storage layout

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



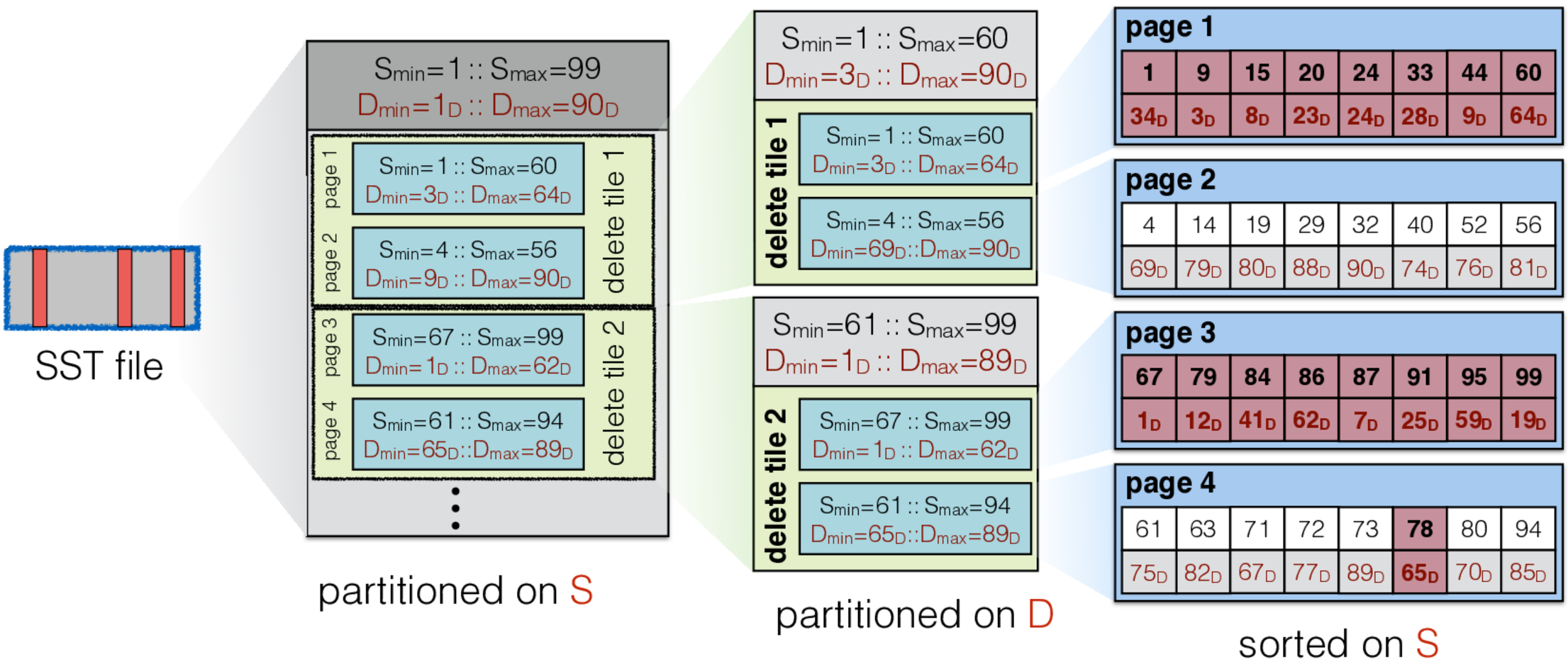
# Key Weaving storage layout

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

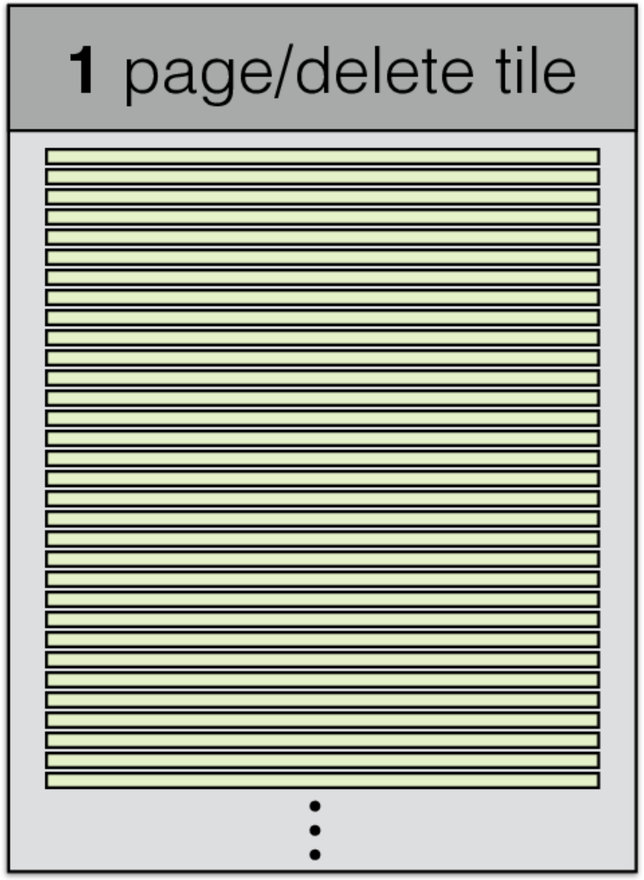


# Key Weaving storage layout

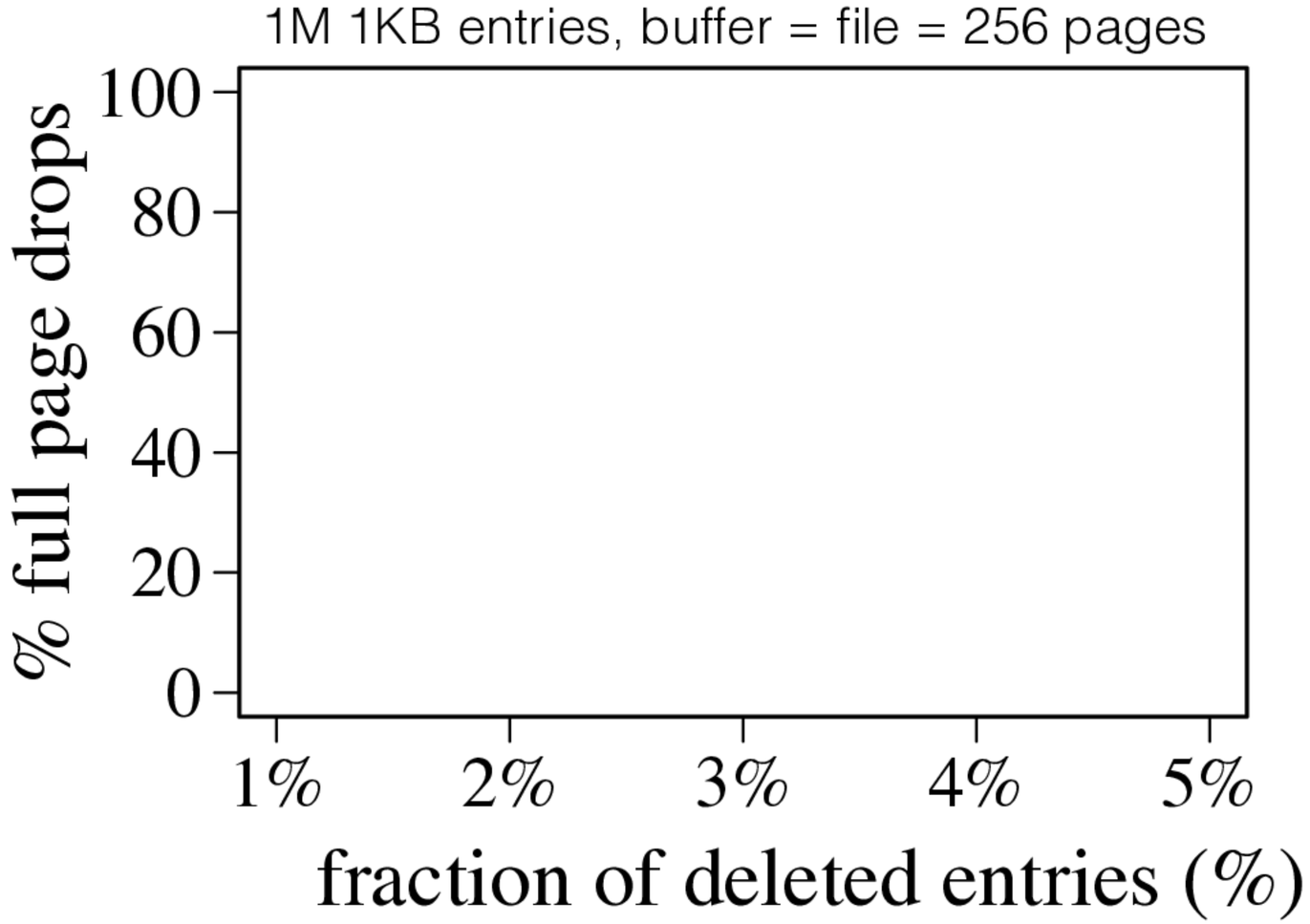
## delete all entries with timestamp <= $65_D$



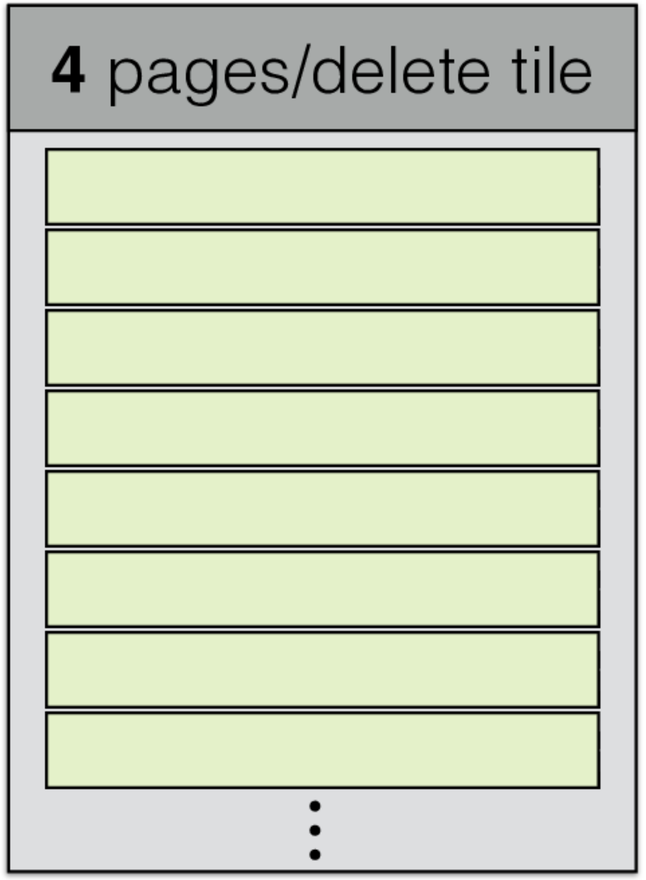
# Key Weaving storage layout



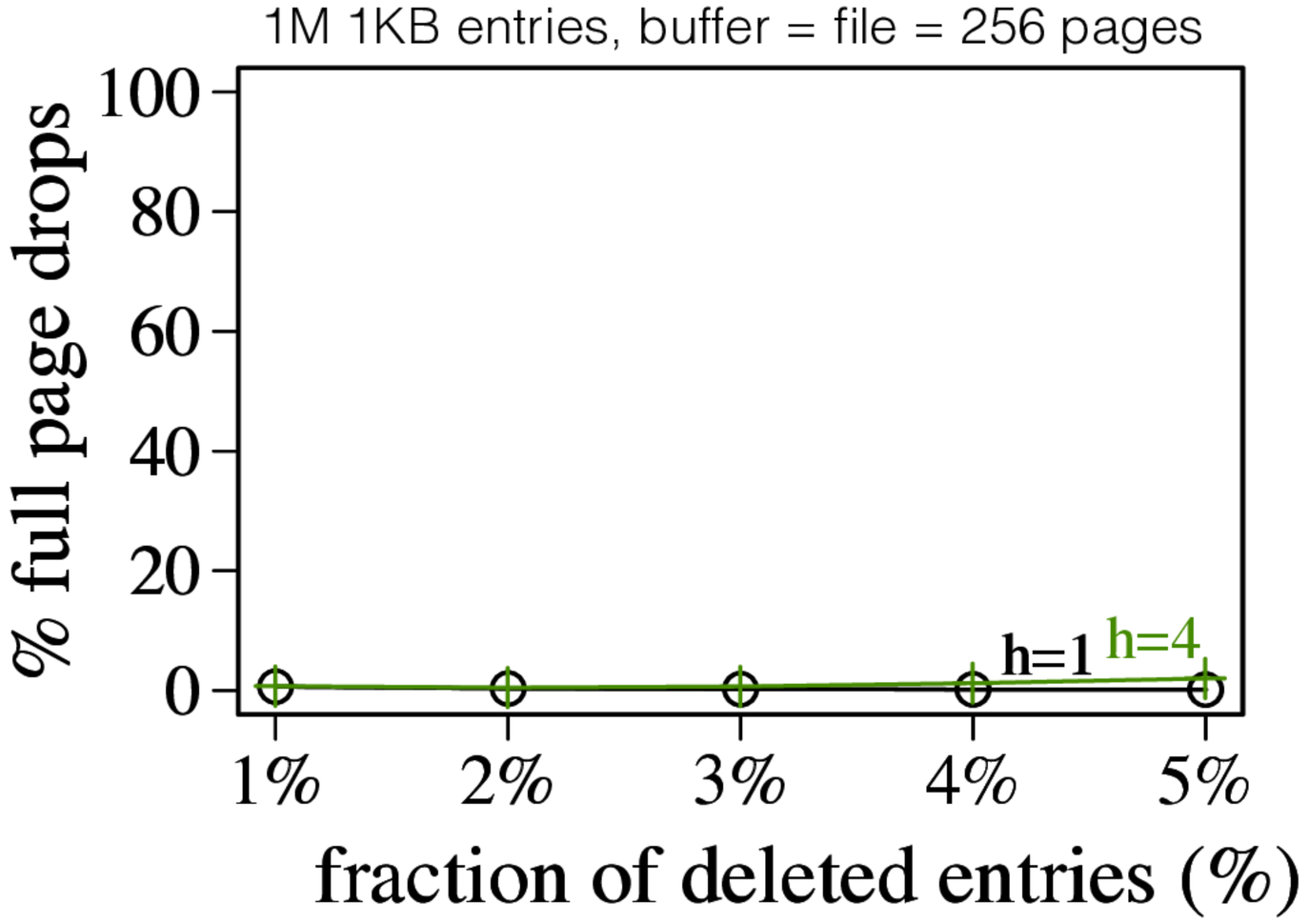

Internals of an SST file in KiWi



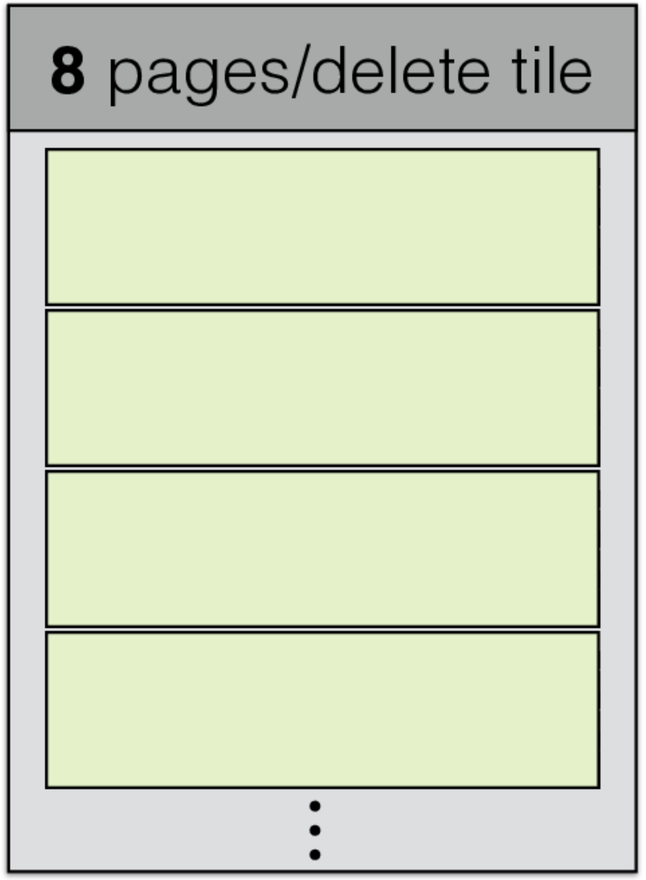
# Key Weaving storage layout



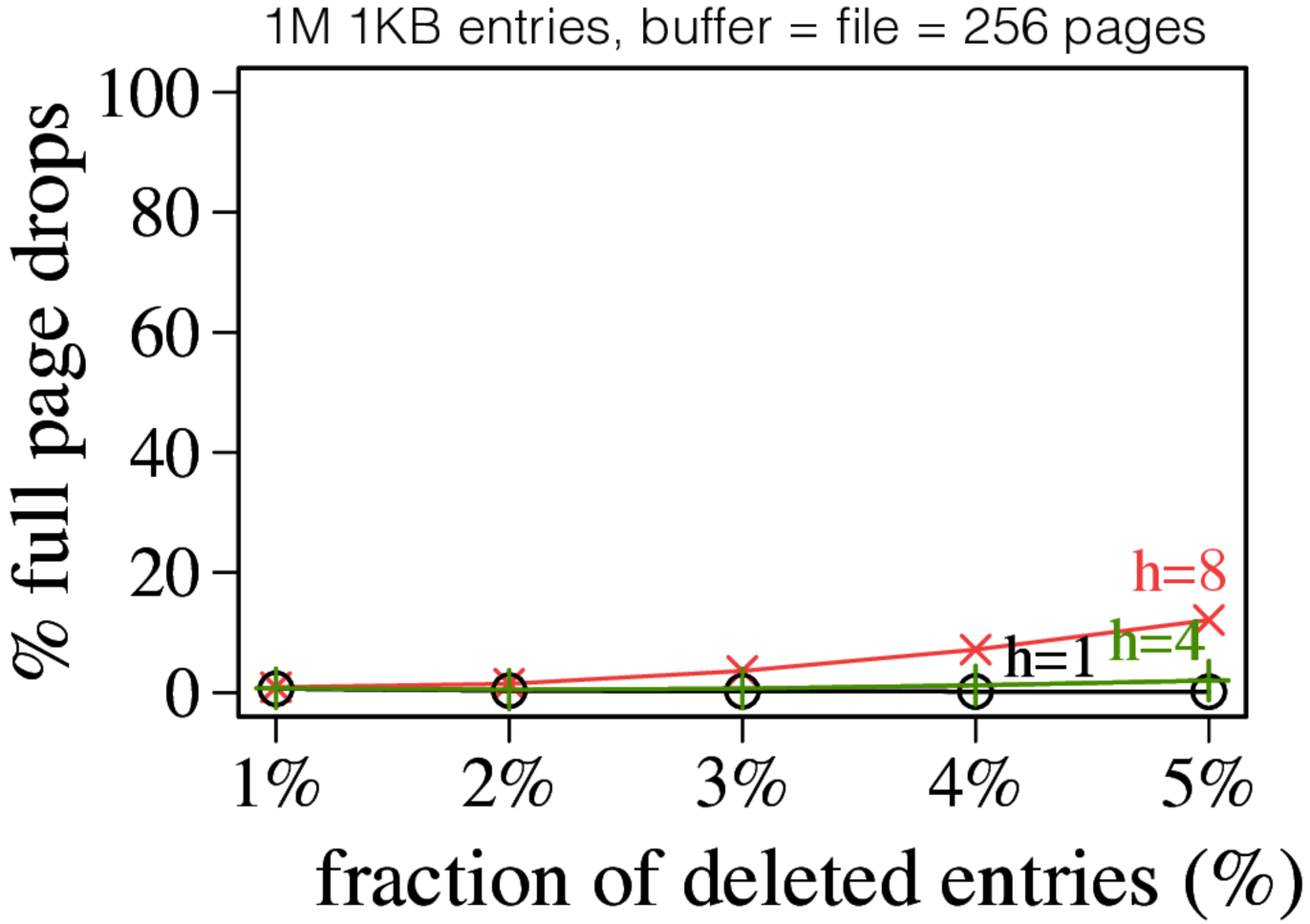

Internals of an SST file in KiWi



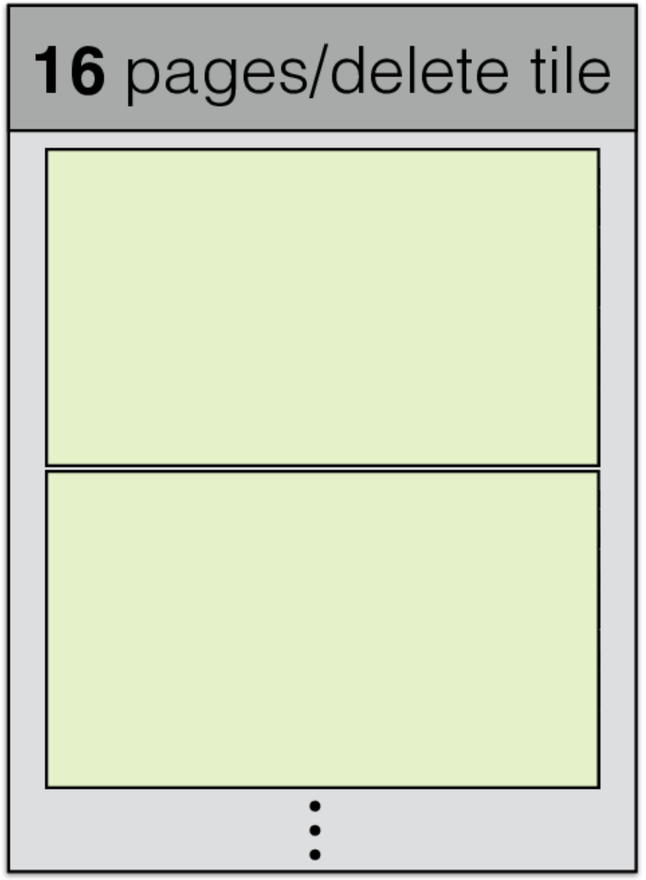
# Key Weaving storage layout



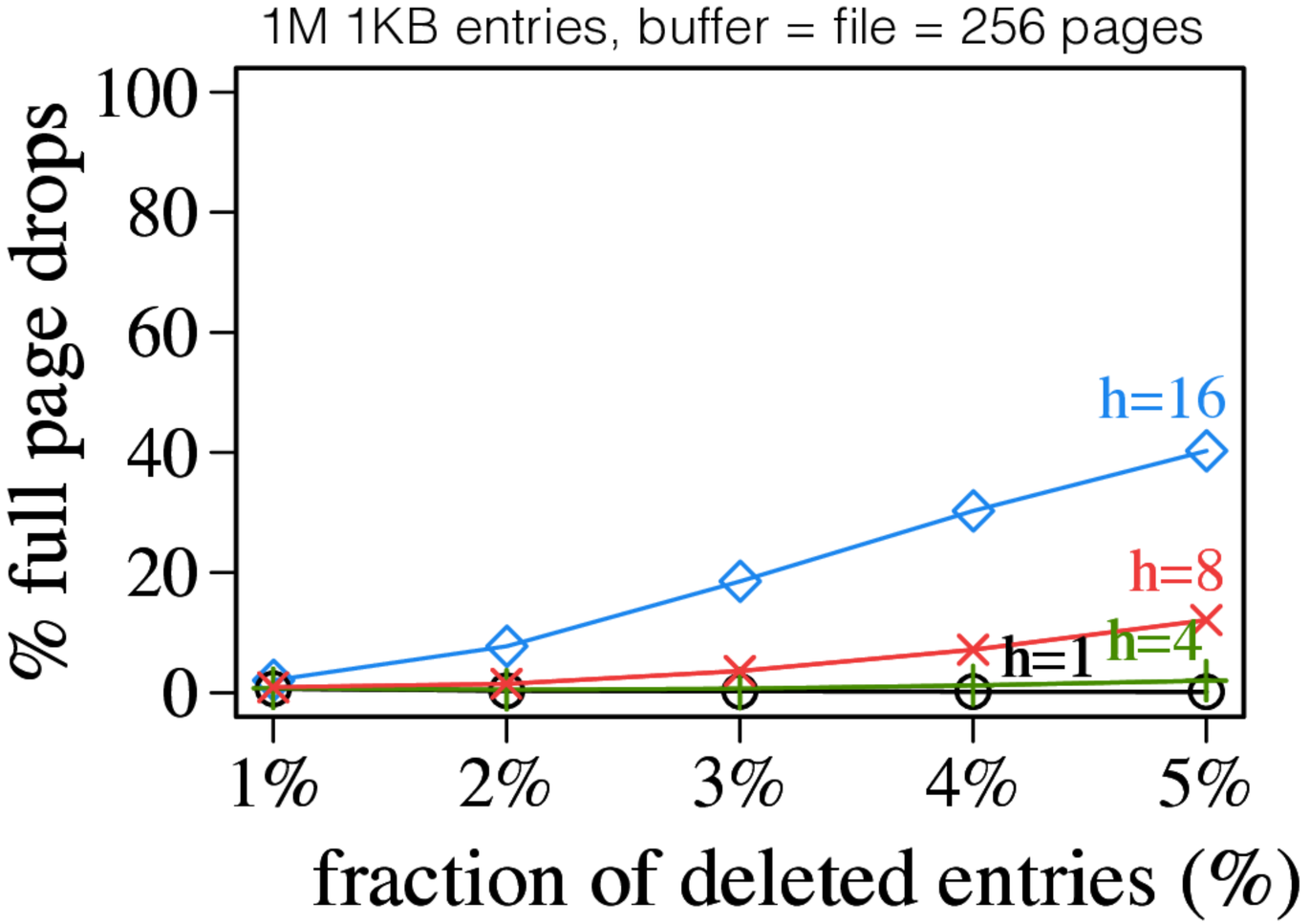

Internals of an SST file in KiWi



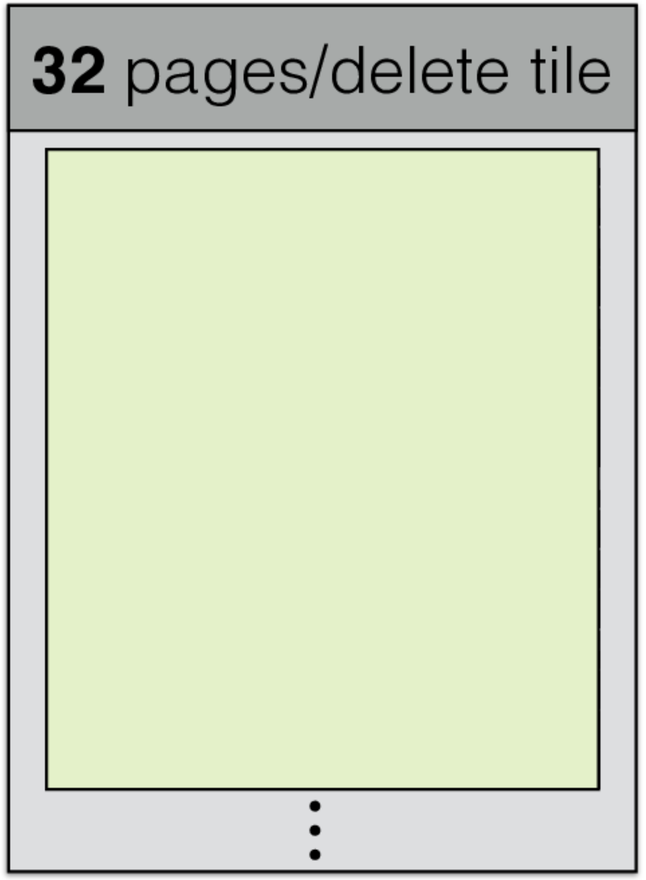
# Key Weaving storage layout



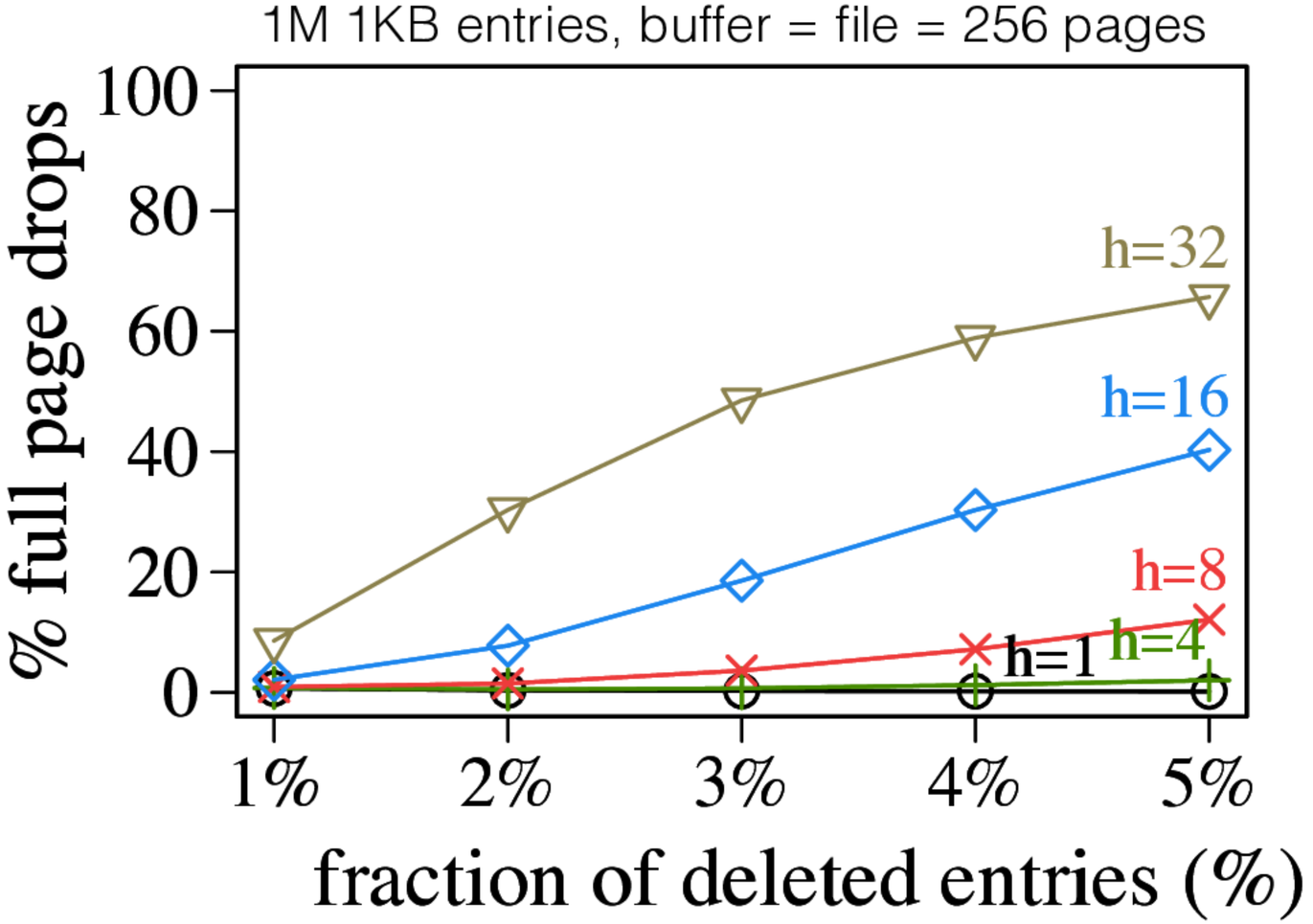

Internals of an SST file in KiWi



# Key Weaving storage layout



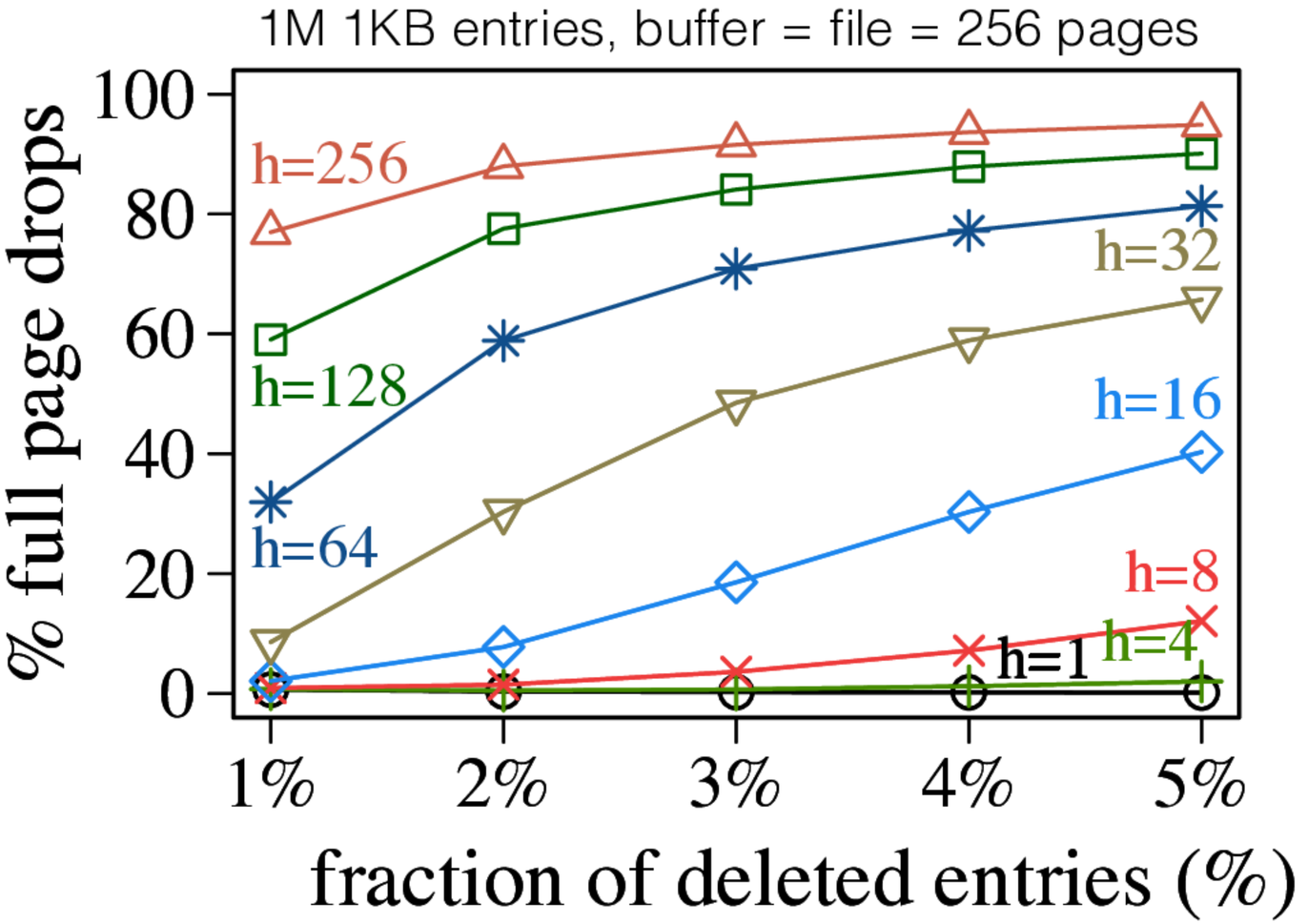

Internals of an SST file in KiWi



# Key Weaving storage layout

reduced latency spikes ✔

full page drops reduces superfluous I/Os ✔

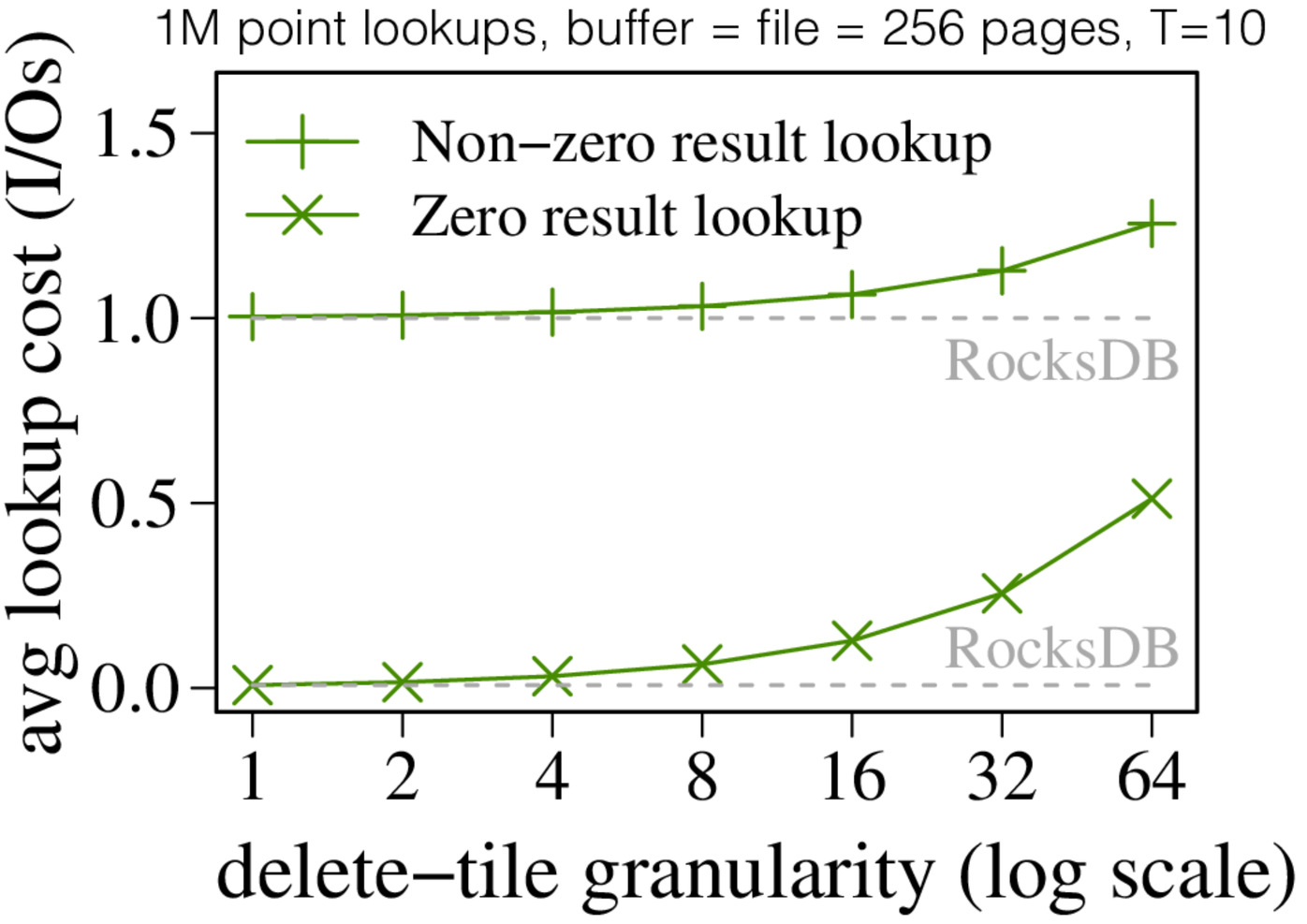


# Key Weaving storage layout

higher lookup cost ↑

reduced latency spikes ✓

full page drops reduces superfluous I/Os ✓



# the solution

FAst DElete

amortized write

reduced space

improved read

timely delete persistence

**FADE**

higher lookup cost

reduced latency spikes

full page drops reduces superfluous I/Os

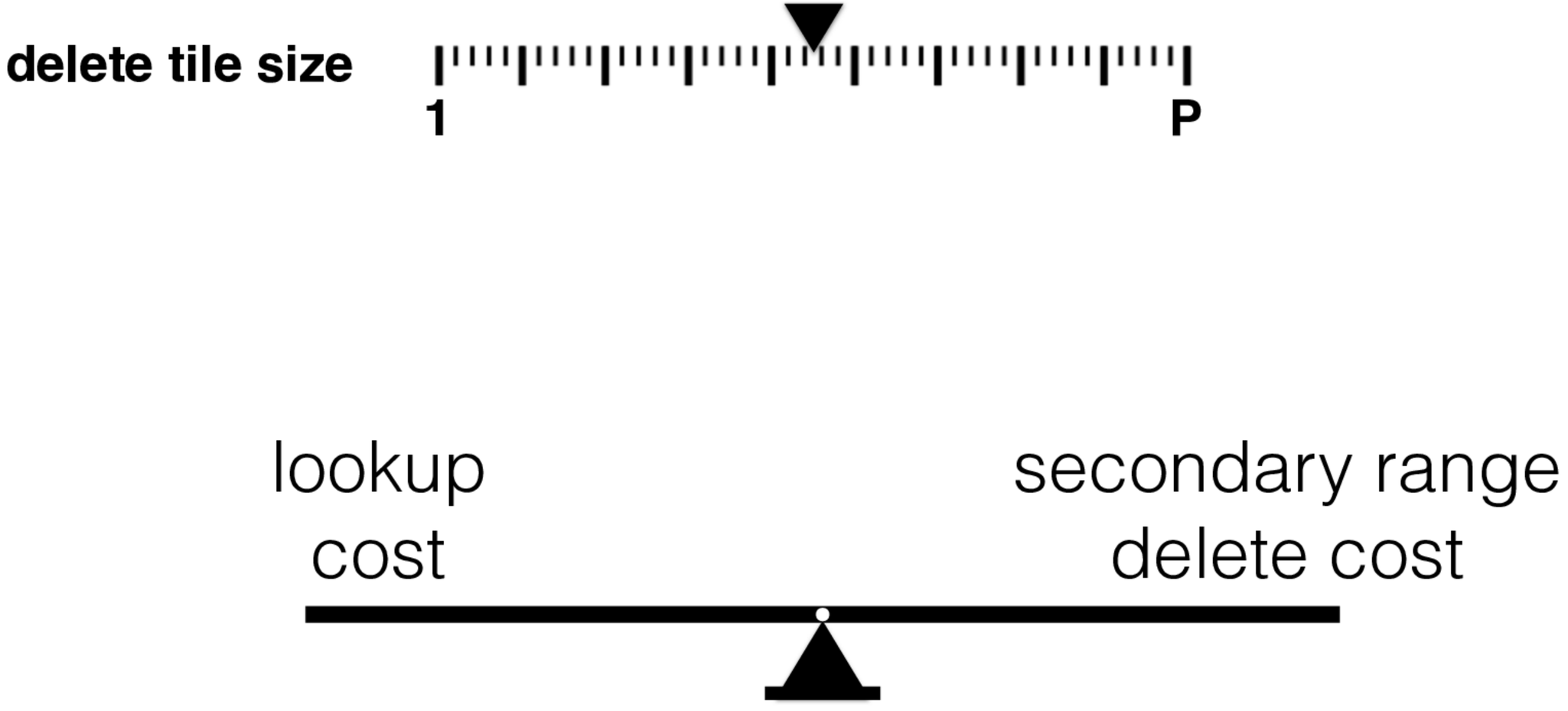
**KiWi**

# the solution

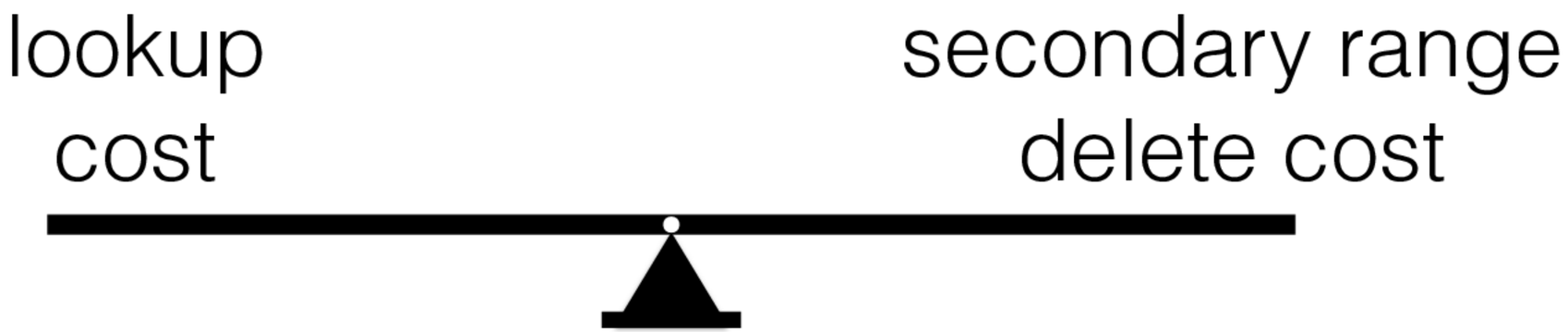
FADE

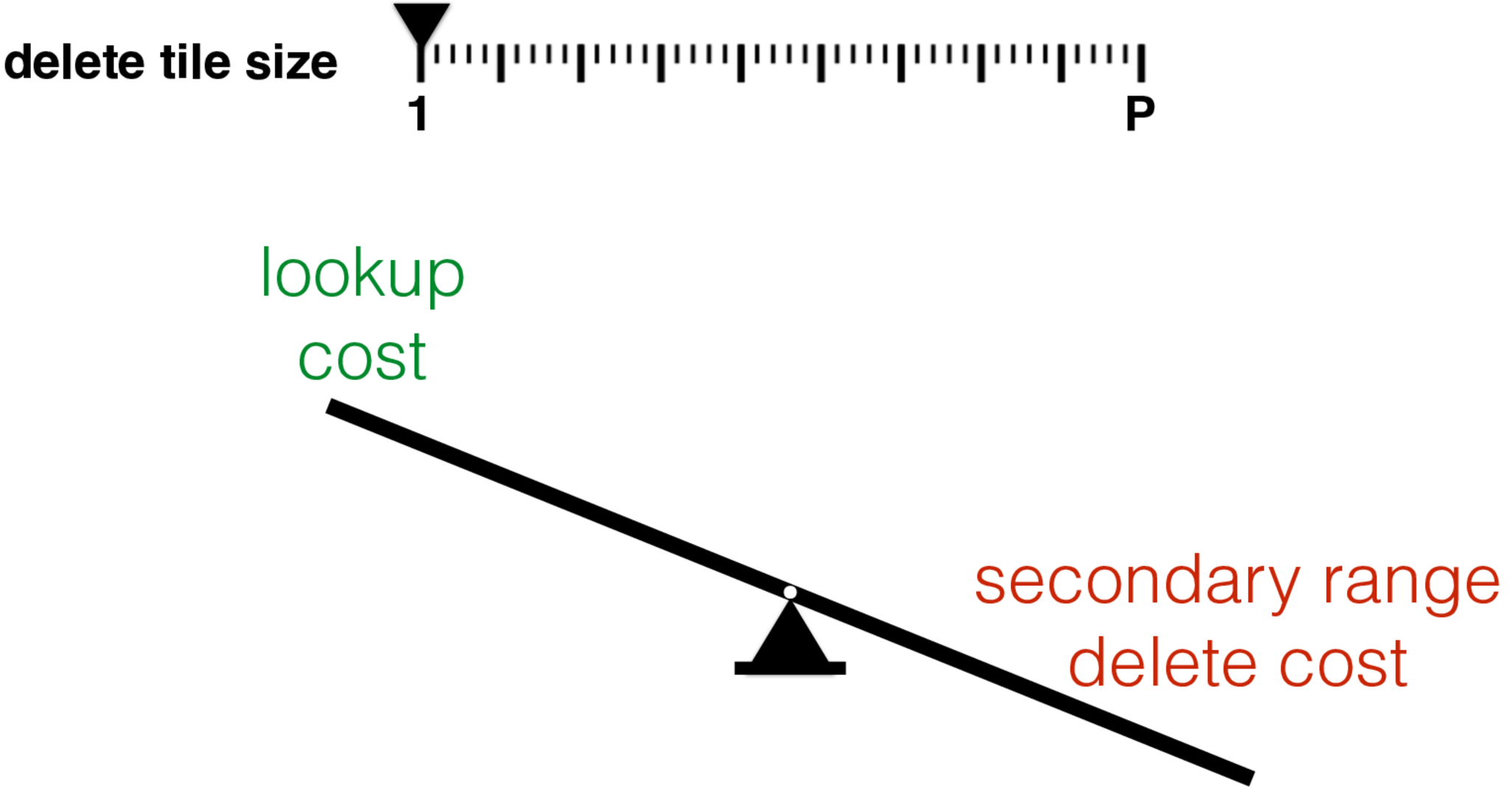
KiWi

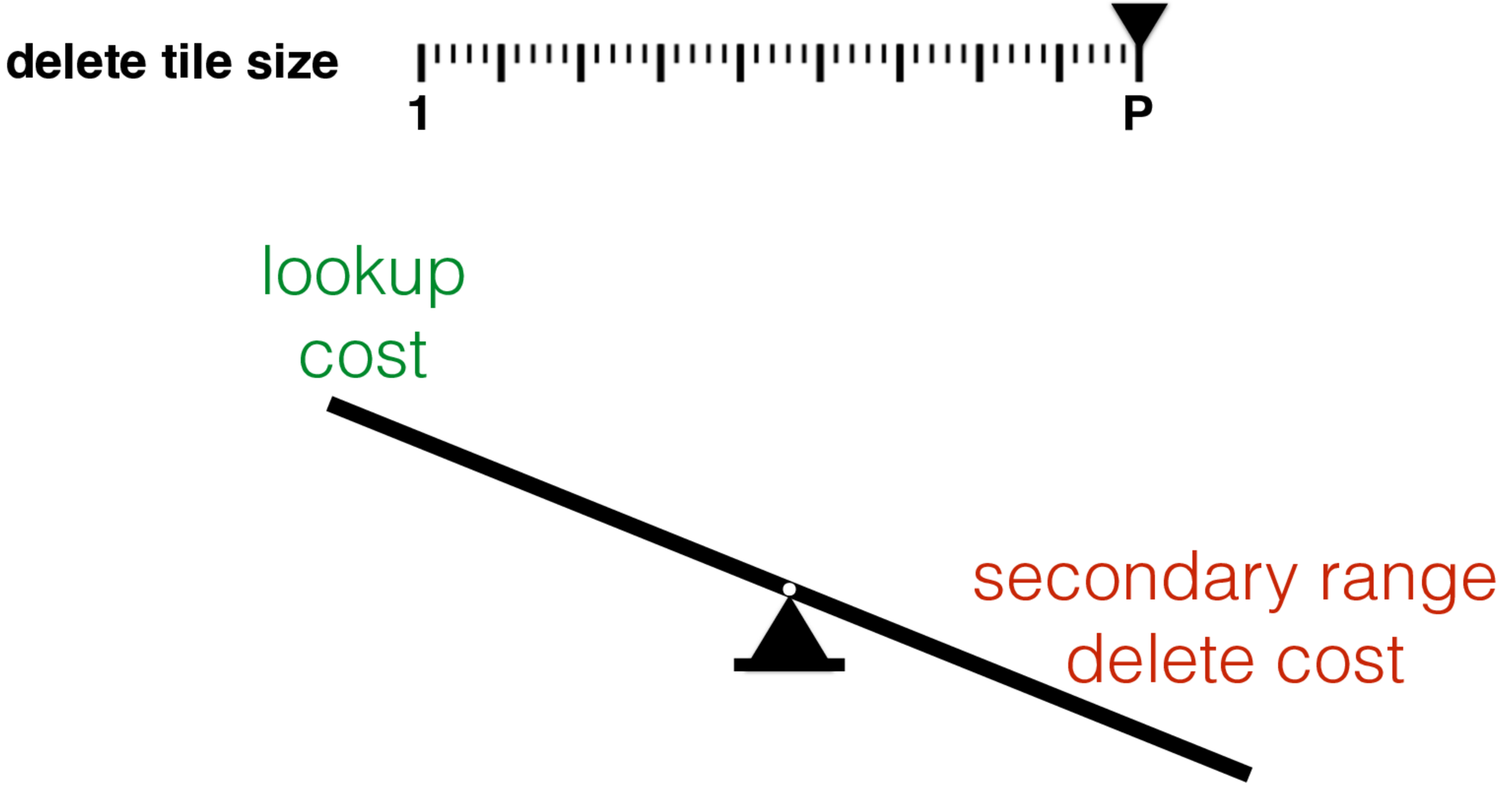
**Lethe**

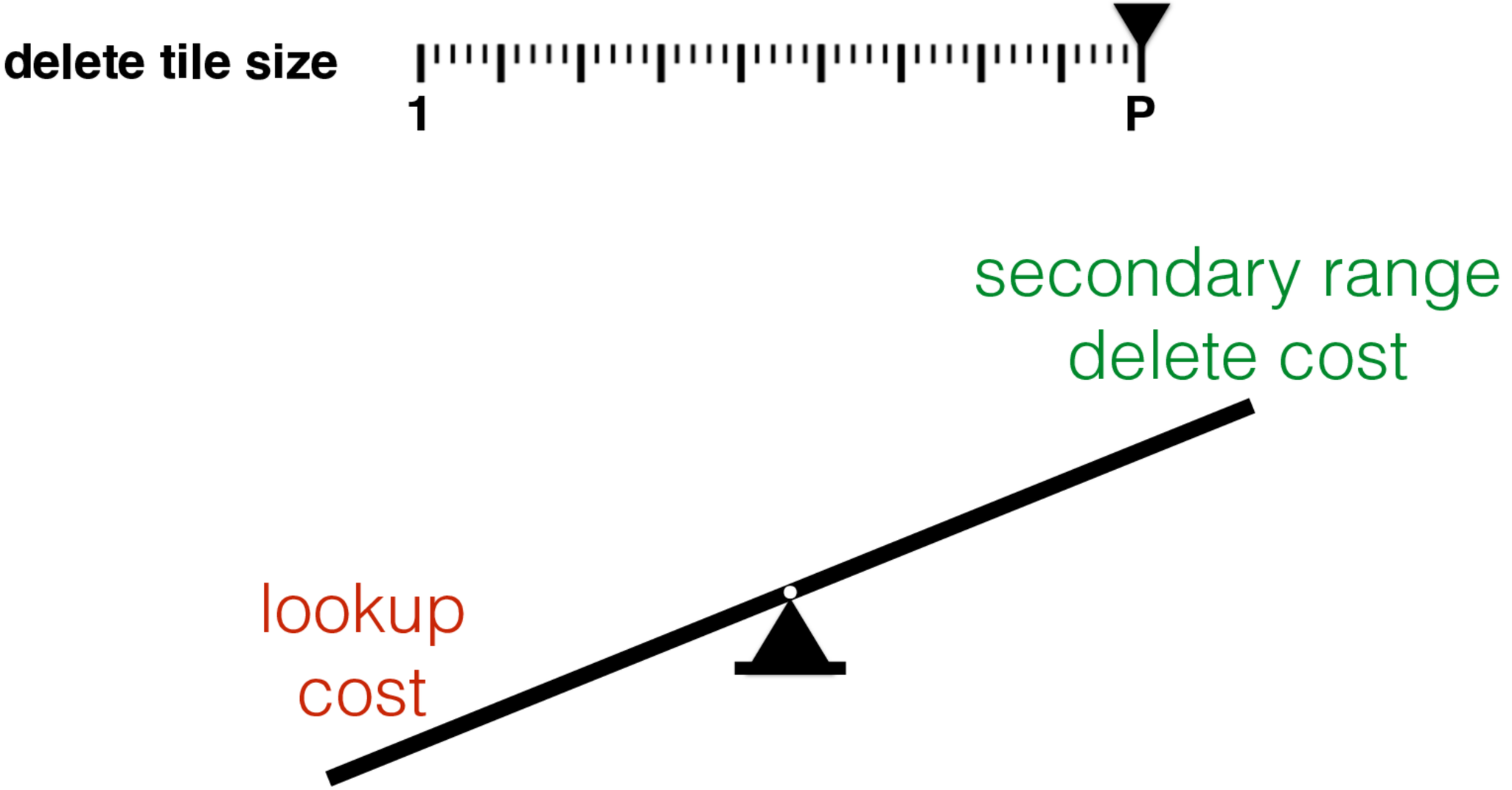

delete tile size
1
P
lookup
cost
secondary range
delete cost

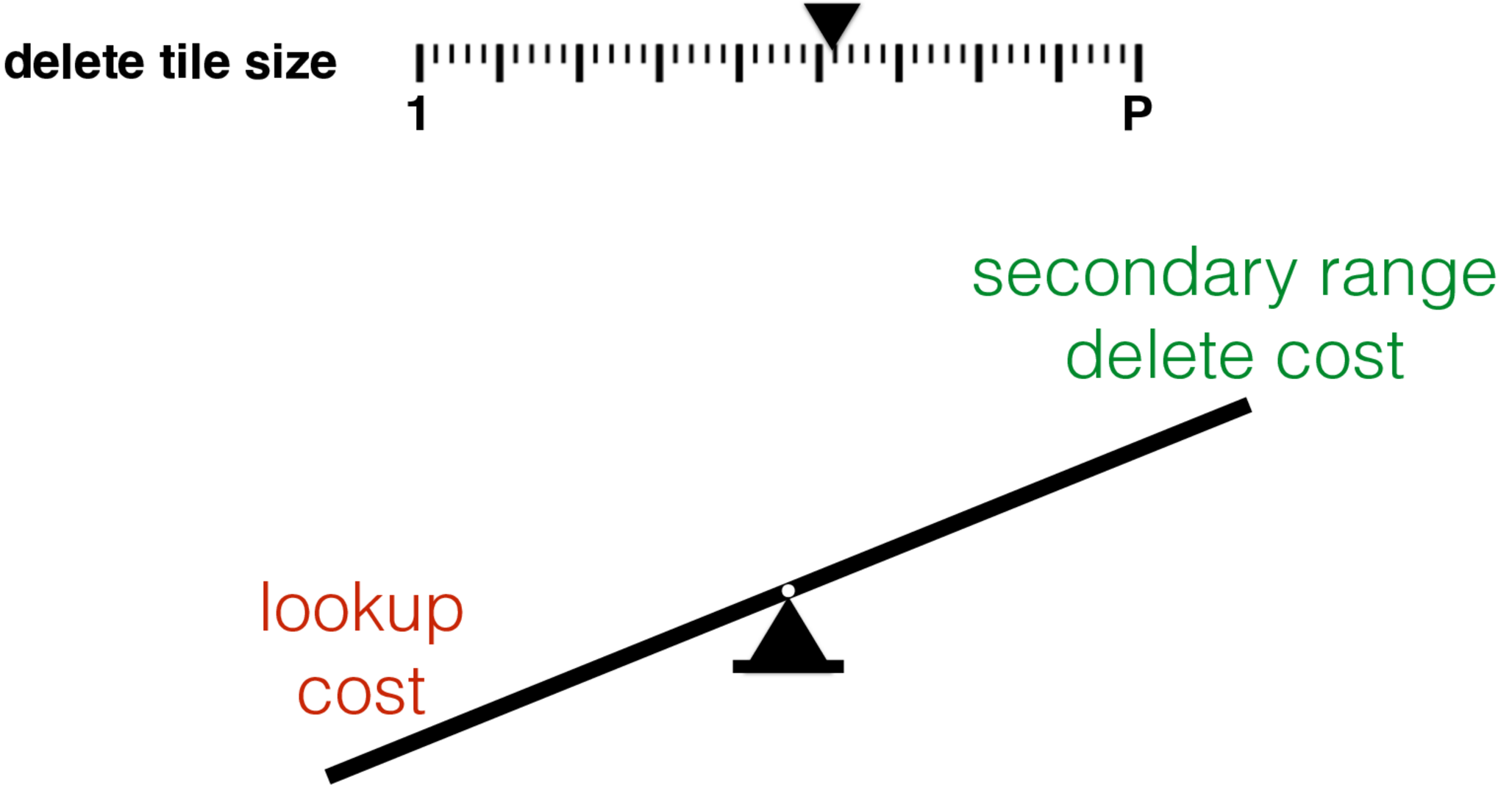

delete tile size
1
P

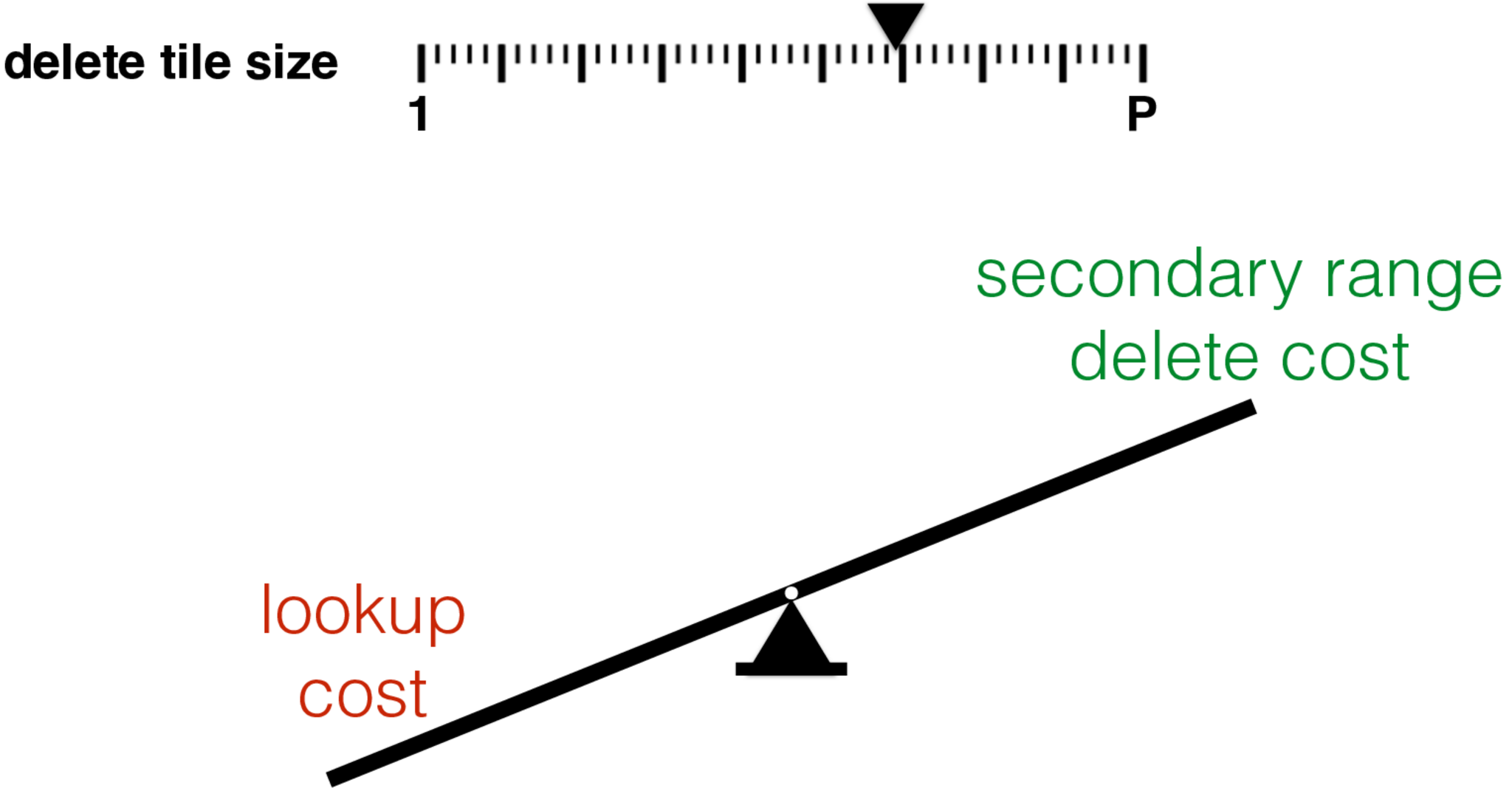

lookup
cost
secondary range
delete cost

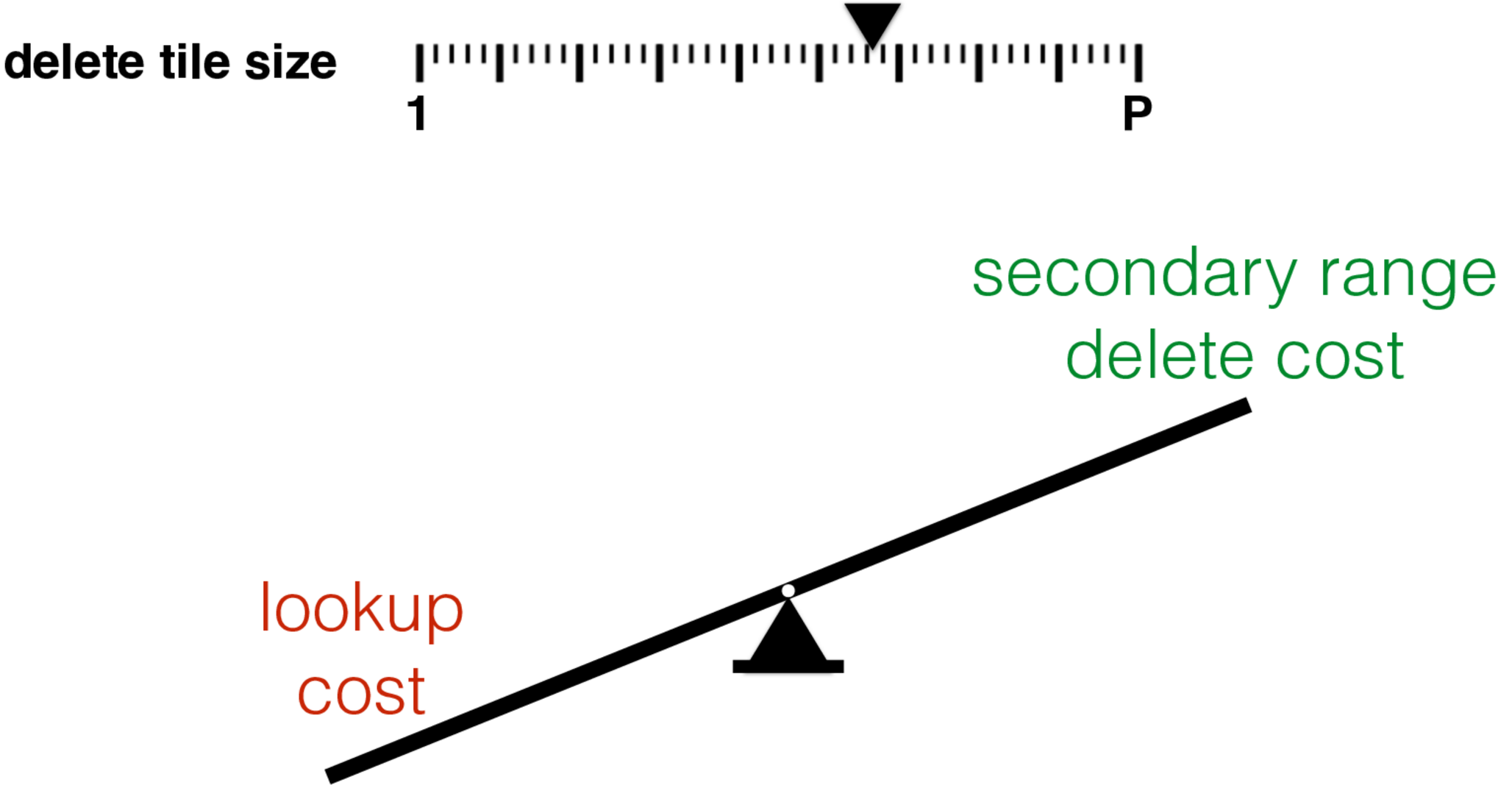

delete tile size
1
P
lookup
cost
secondary range
delete cost

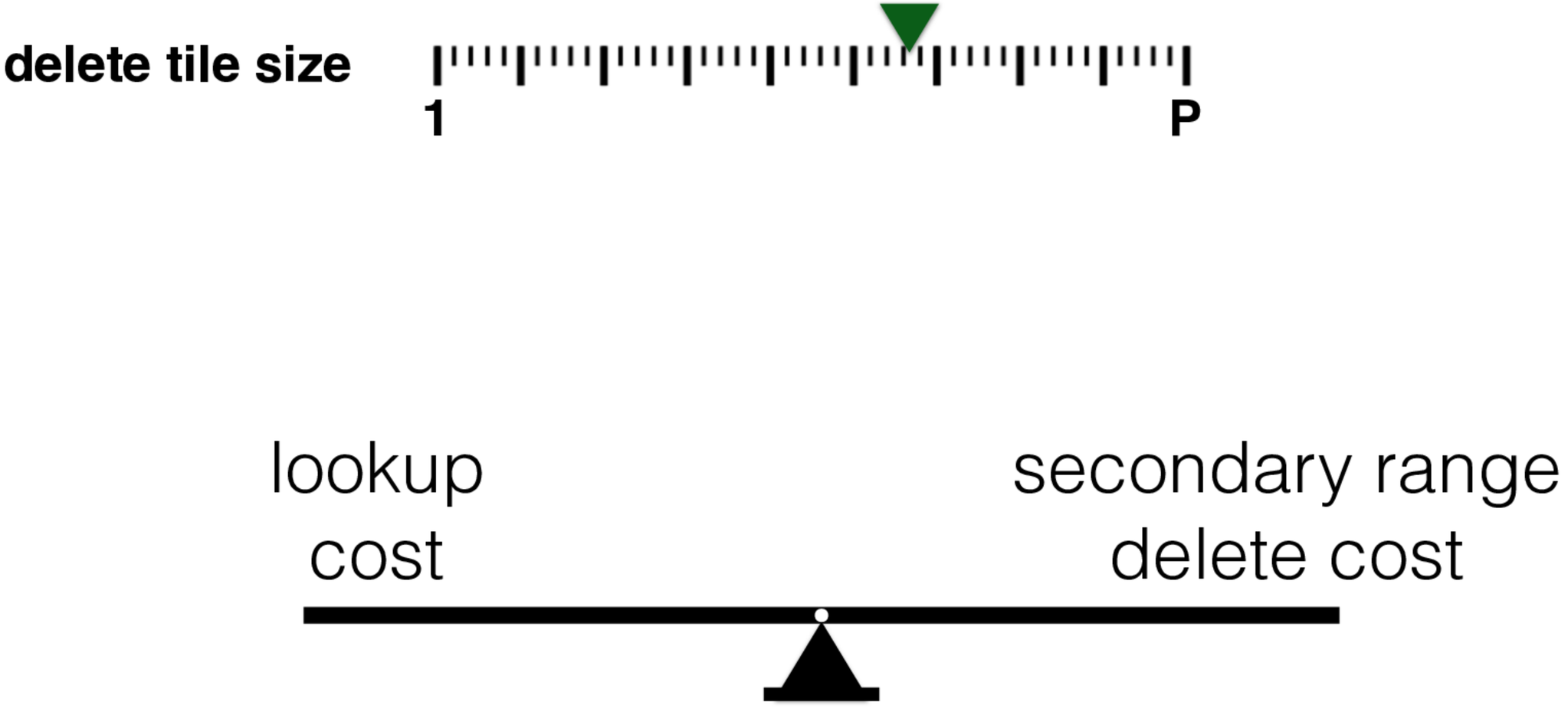

delete tile size
1
P
lookup
cost
secondary range
delete cost

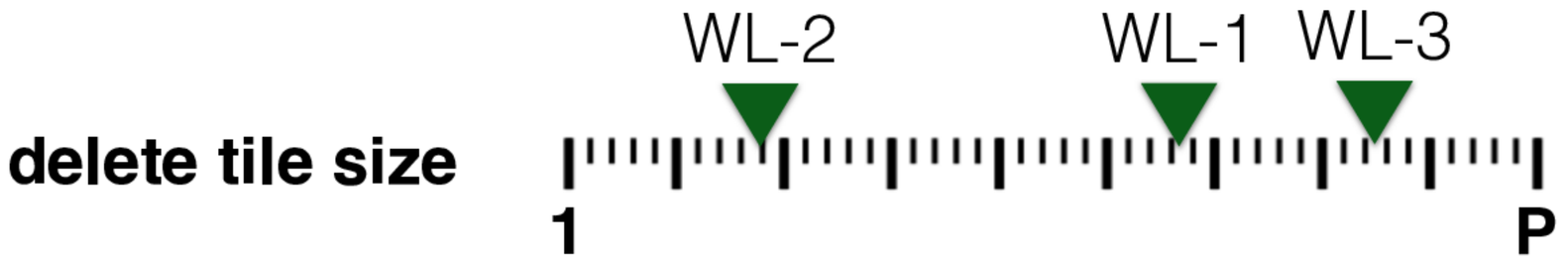

delete tile size
1
P
secondary range
delete cost
lookup
cost

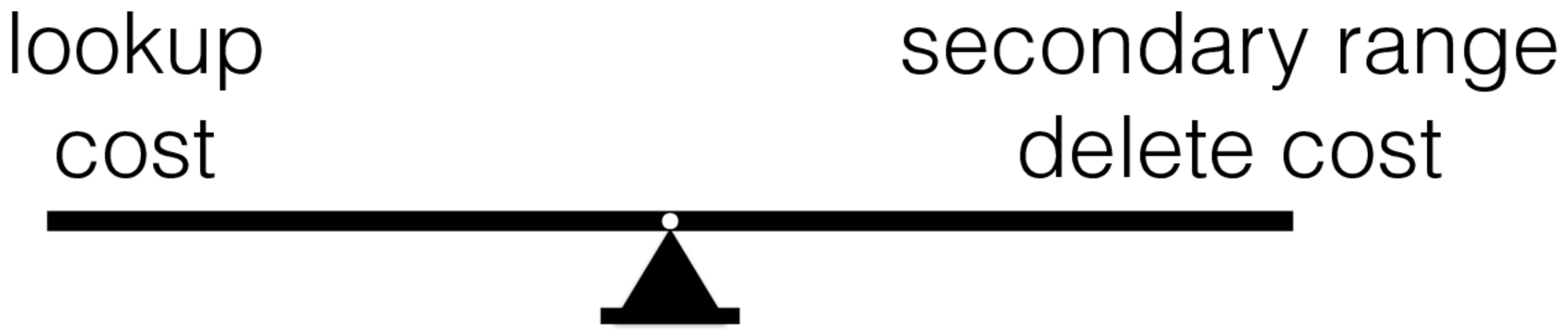

delete tile size
1
P
secondary range
delete cost
lookup
cost


delete tile size
1
P
secondary range
delete cost
lookup
cost


delete tile size
1
P
secondary range
delete cost
lookup
cost


delete tile size
1
P
lookup
cost
secondary range
delete cost


WL-2
WL-1
WL-3
delete tile size
1
P


lookup
cost
secondary range
delete cost

suboptimal state-of-the-art design

for workloads with deletes

FADE persists deletes timely

using latency-driven compactions

KiWi supports efficient
secondary range deletes

using key-interweaved data storage

suboptimal state of the art design

for workloads with deletes

FADE persists deletes timely

using latency-driven compactions

KiWi supports efficient secondary range deletes

by key-interweaved data layout

Lethe strikes balance between cost, performance, and latency

Thank You!

disc-projects.bu.edu/lethe/