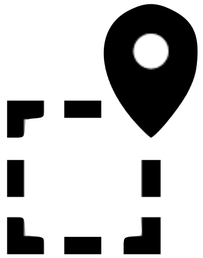
Advanced Storage

(Flash translation layers, SSD garbage-collection)

Niv Dayan - CSC2525: Research Topics in Database Management

SSDs &
FTLs



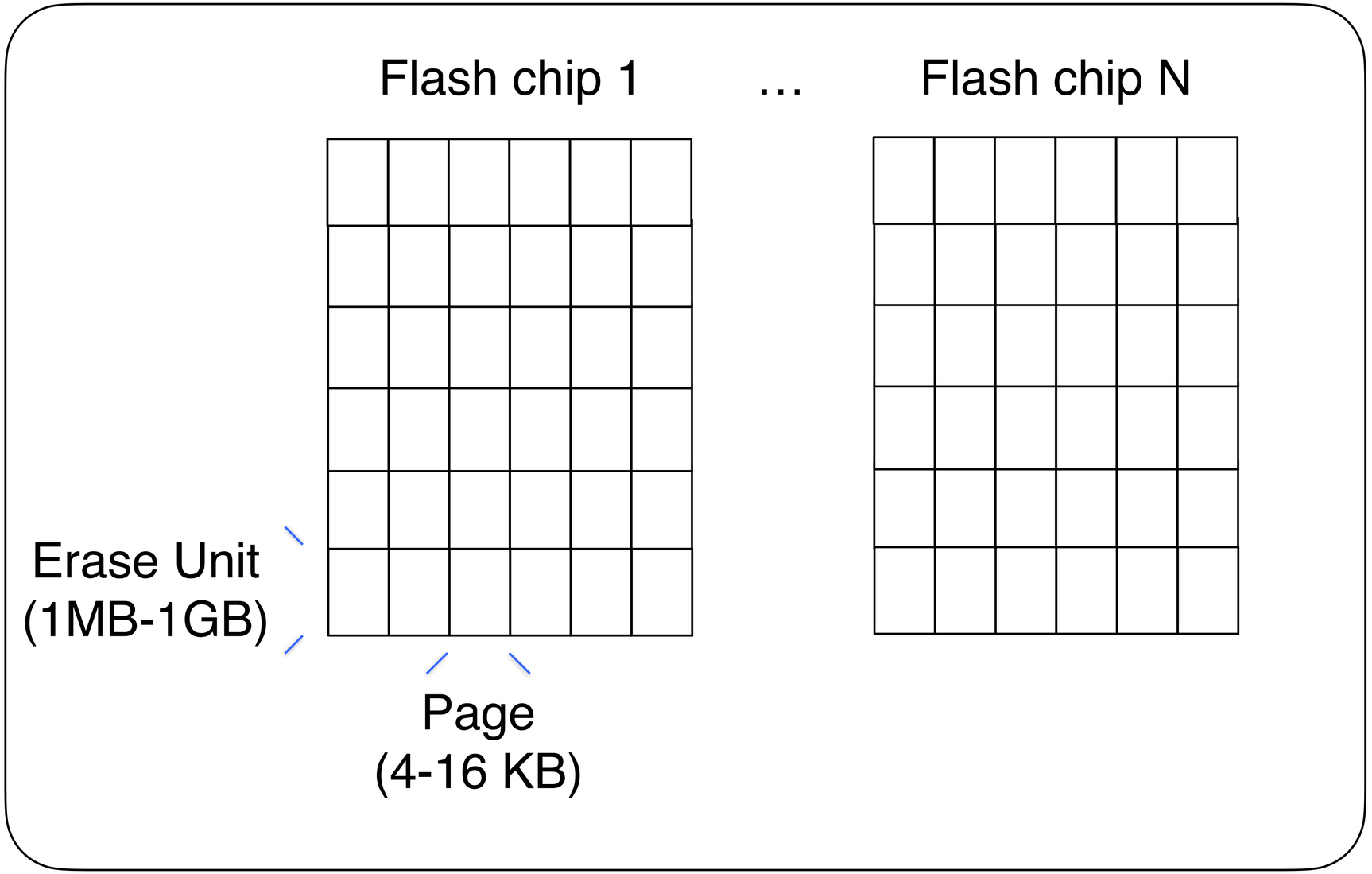
Zoned
Namespaces /
KV SSDs

Spooky

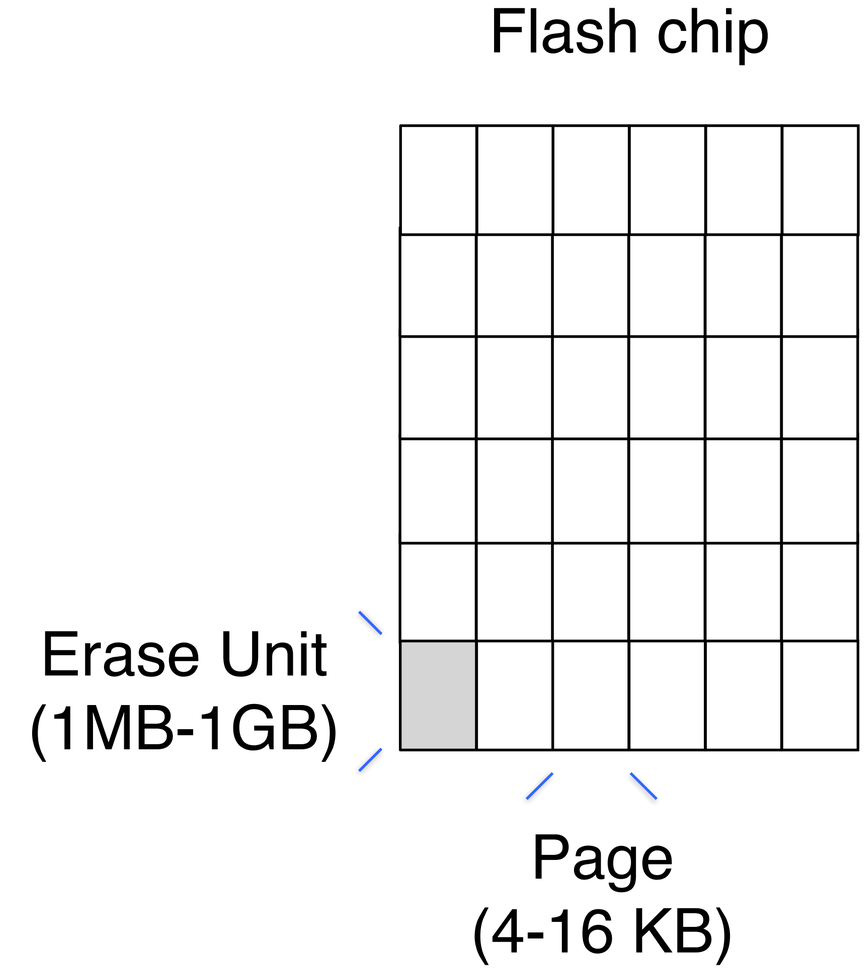
Shingled
Magnetic Disks

SSD Review

SSD Review

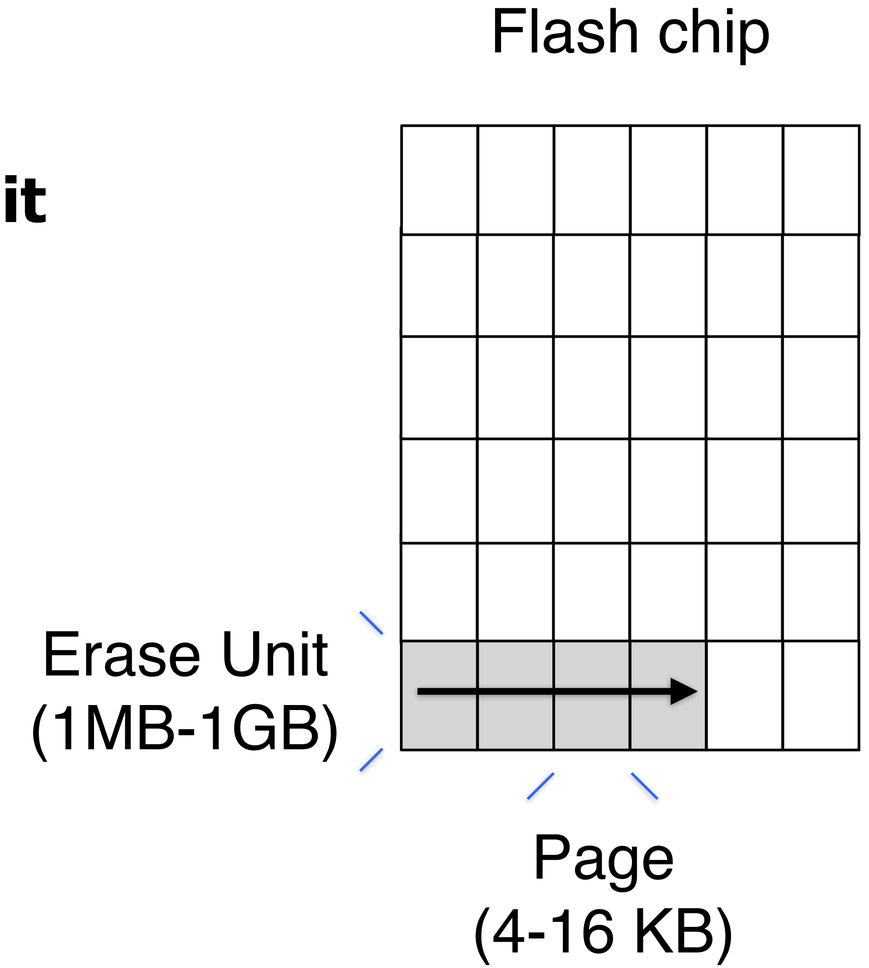


A page is the minimum read/write unit



A page is the minimum read/write unit

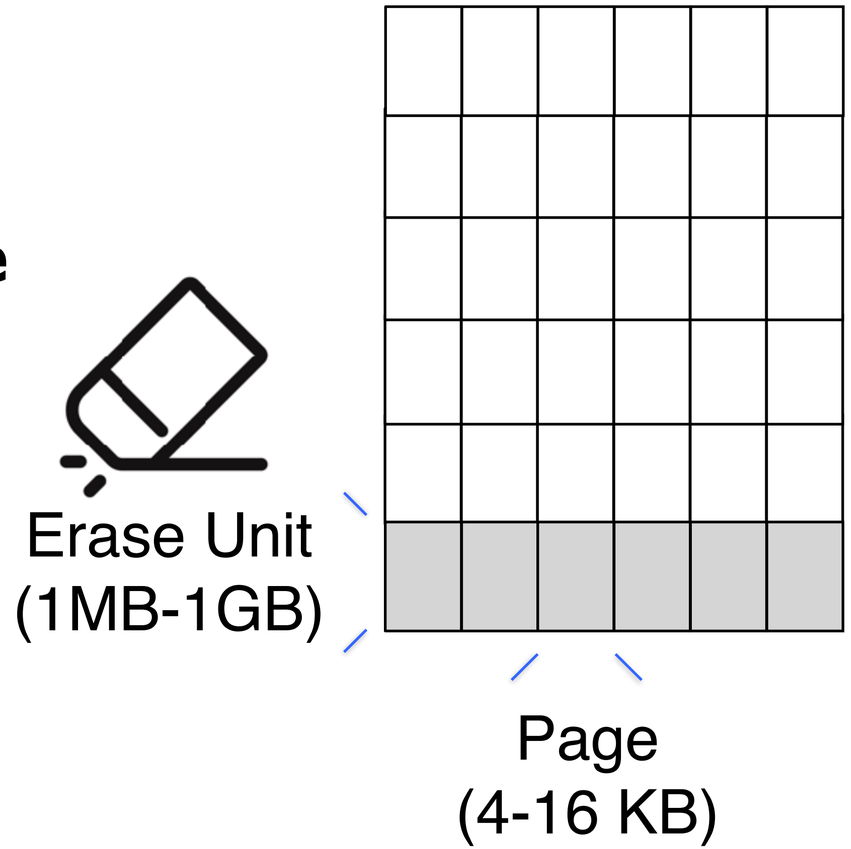
Pages must be written sequentially in an erase unit



A page is the minimum read/write unit

Pages must be written sequentially in an erase unit

All data in an erase unit is erased at the same time

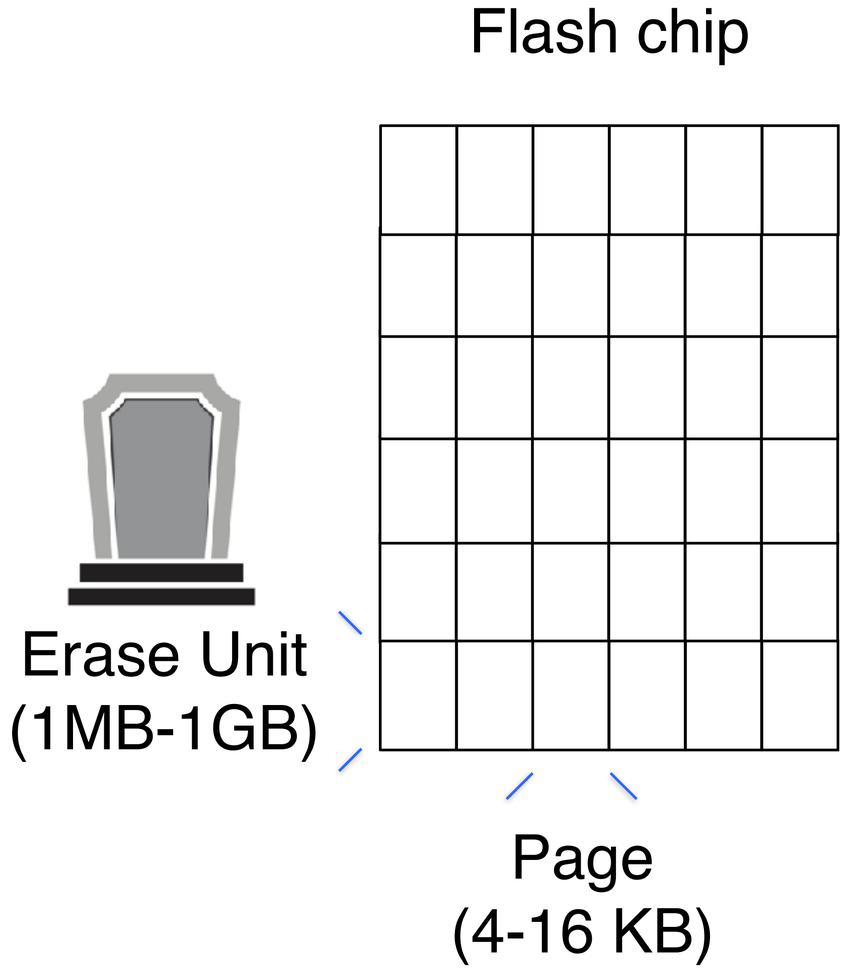


A page is the minimum read/write unit

Pages must be written sequentially in an erase unit

All data in an erase unit is erased at the same time

Each erase unit has a lifetime (1-10K erases)



A page is the minimum read/write unit

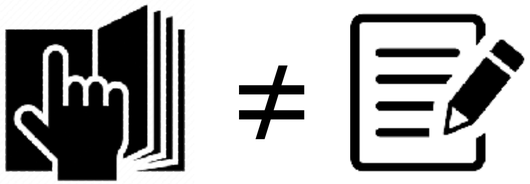
Pages must be written sequentially in an erase unit

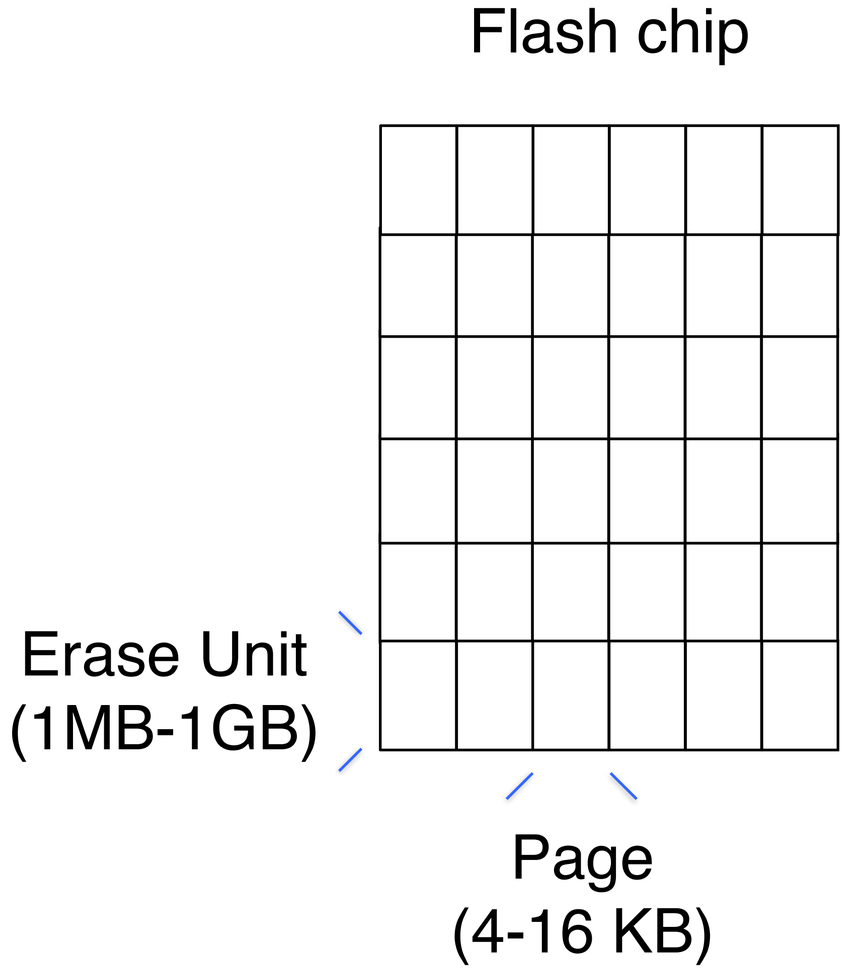
All data in an erase unit is erased at the same time

Each erase unit has a lifetime (1-10K erases)

Reading a page takes approx 50 us

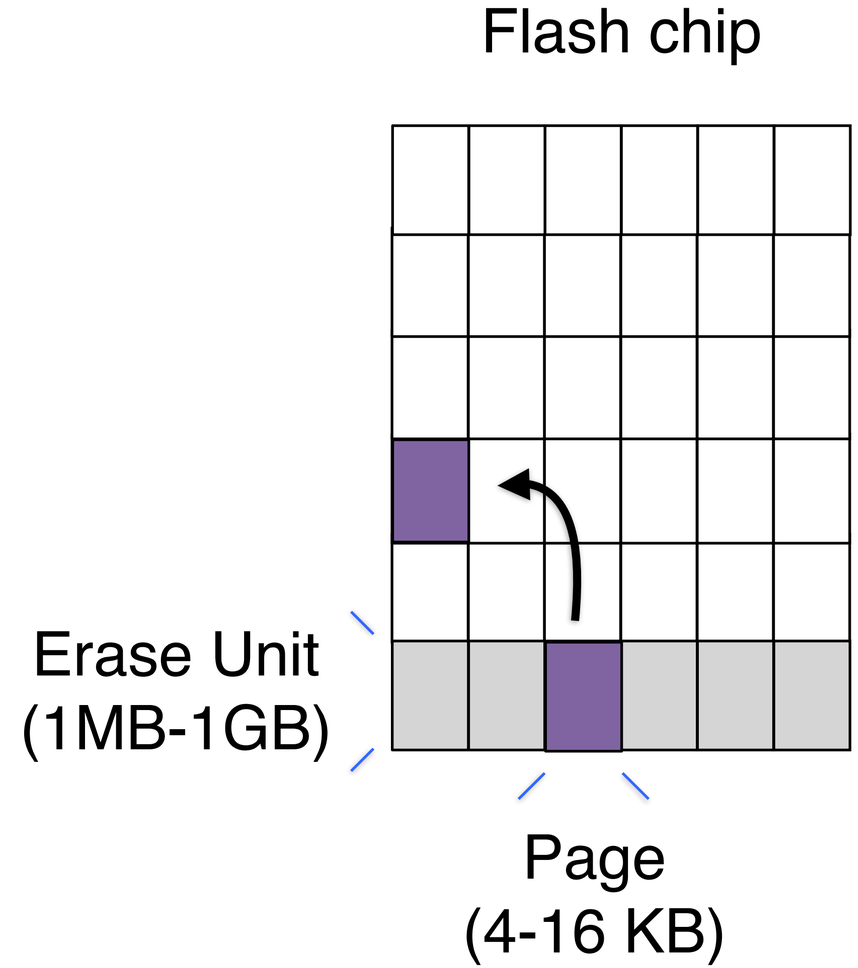
Writing a page takes approx 100-200 us





Updating a Page Out-of-Place

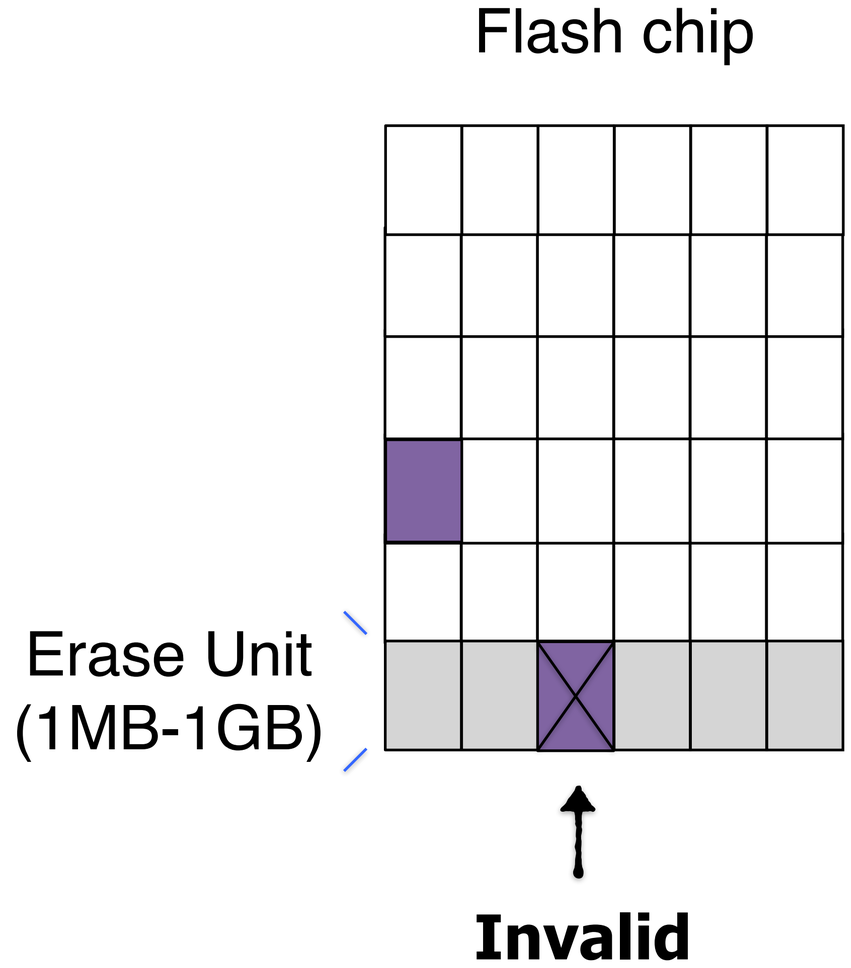
Copy updated page to erase unit with free space



Updating a Page Out-of-Place

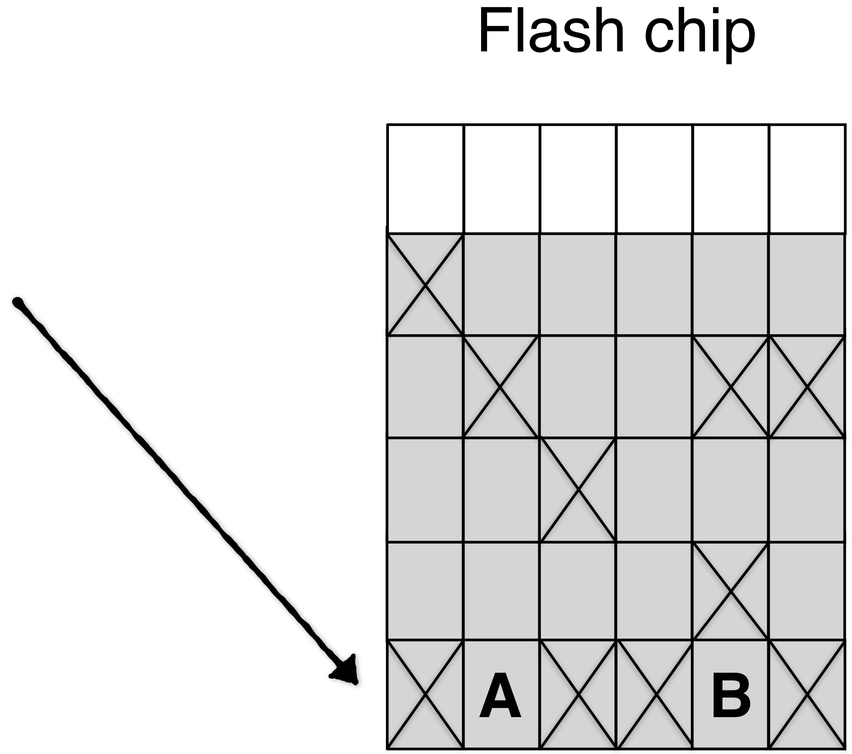
Copy updated page to erase unit with free space

Mark the original page as invalid (using a bitmap)



Garbage-Collection

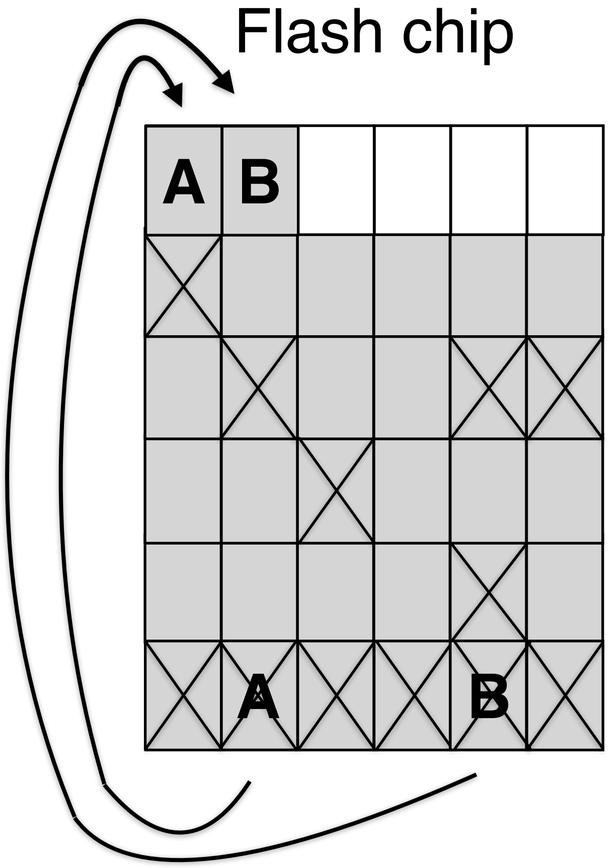
Find the erase unit with the least live data left.



Garbage-Collection

Find the erase unit with the least live data left.

Copy live pages to an erase unit with free space.

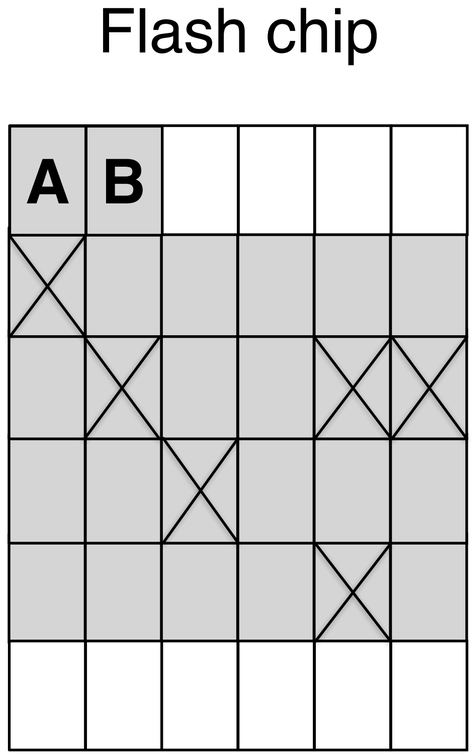


Garbage-Collection

Find the erase unit with the least live data left.

Copy live pages to an erase unit with free space.

Erase the block

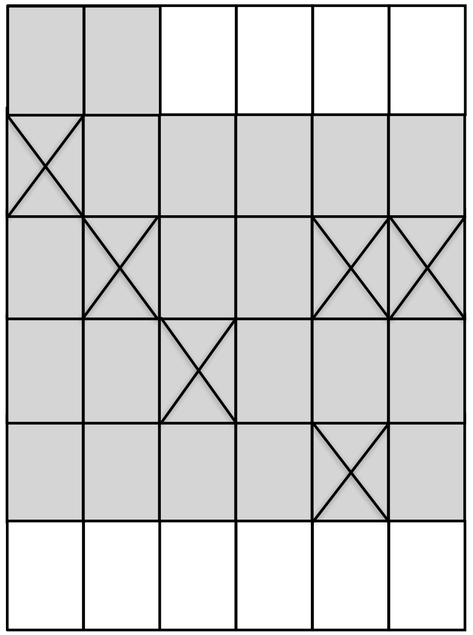


Over-Provisioning

SSD physical capacity must be greater than logical capacity

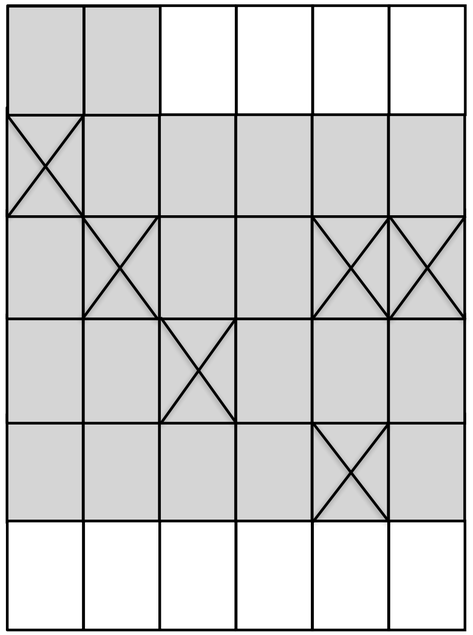
So invalid pages can accumulate to cheapen garbage-collection

Flash chip

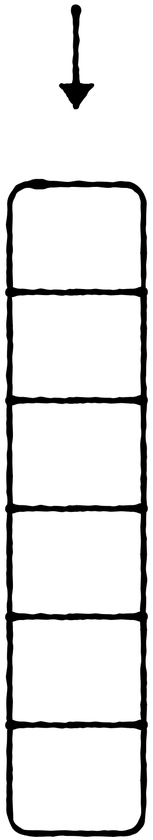


And now to new things :)

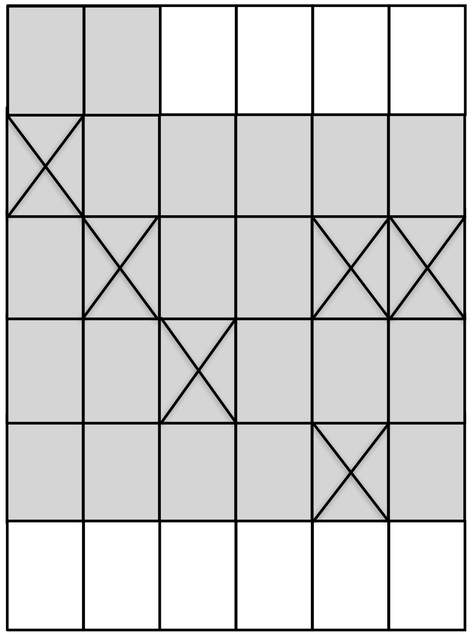
Flash chip



page mapping table



Flash chip



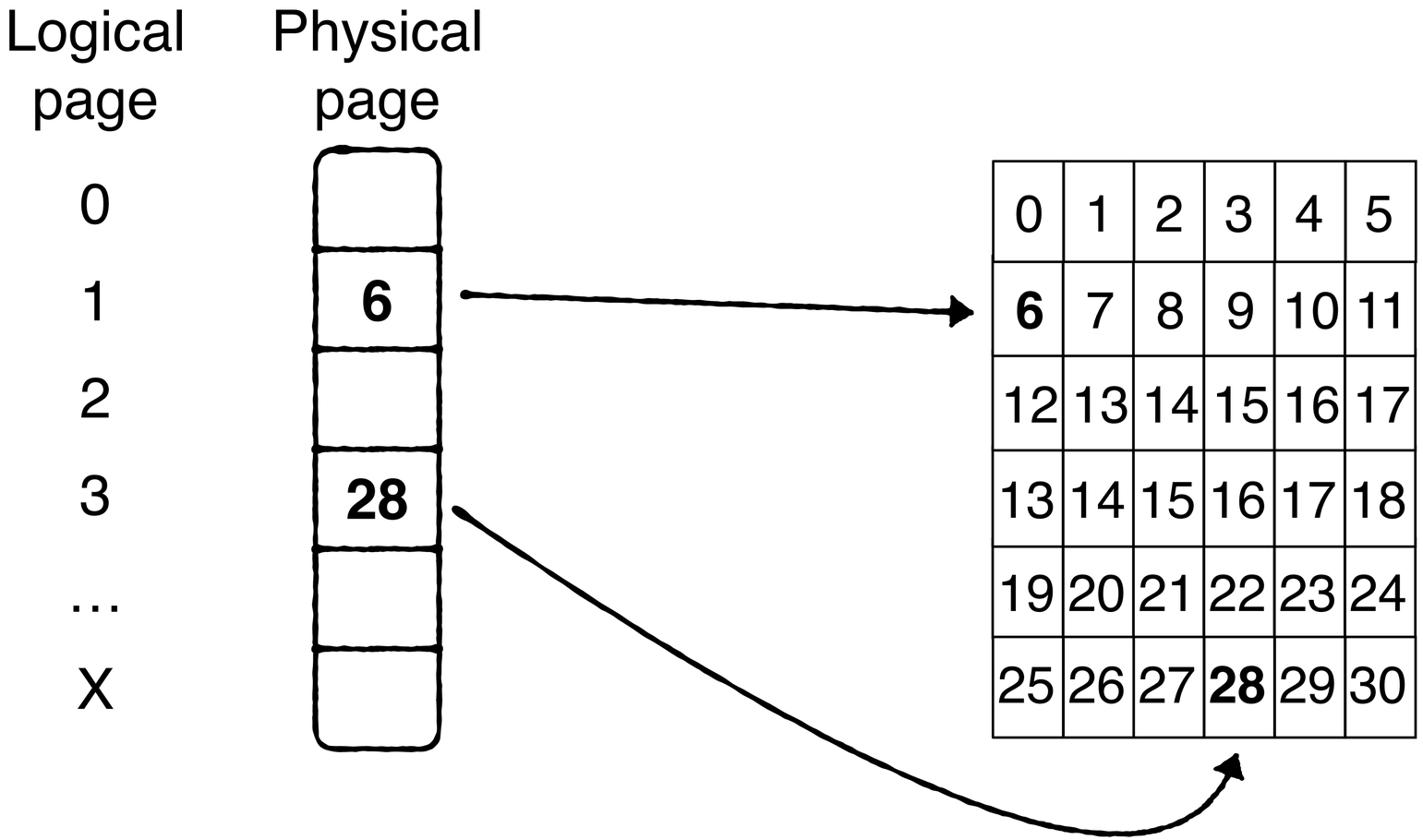
page mapping table

Logical page	Physical page
0	
1	
2	
3	
...	
X	

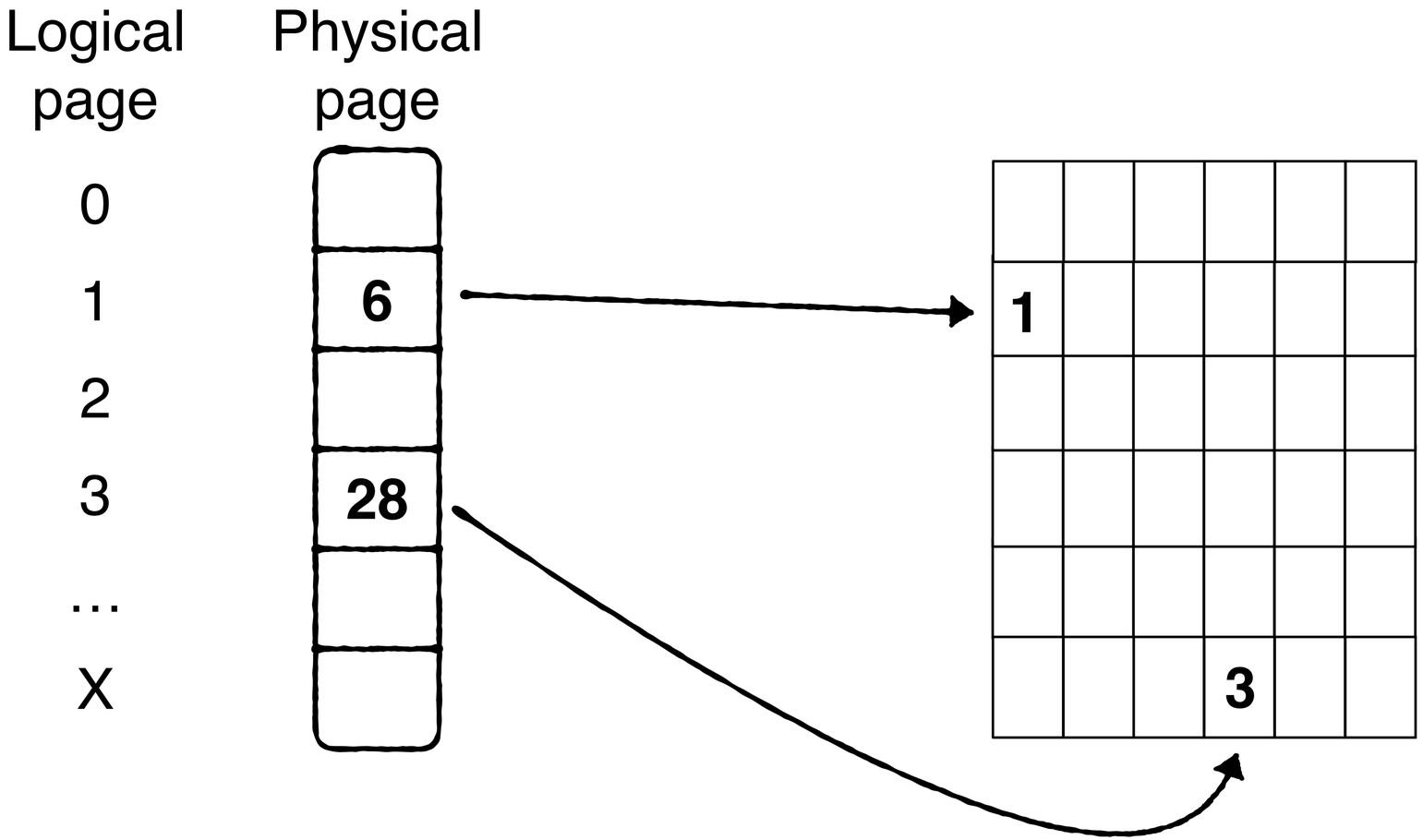
Flash chip

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

page mapping table



page mapping table



Flash translation layer (FTL)

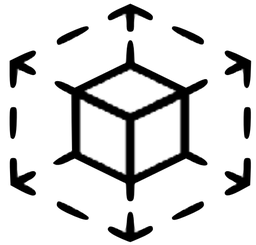
mapping table

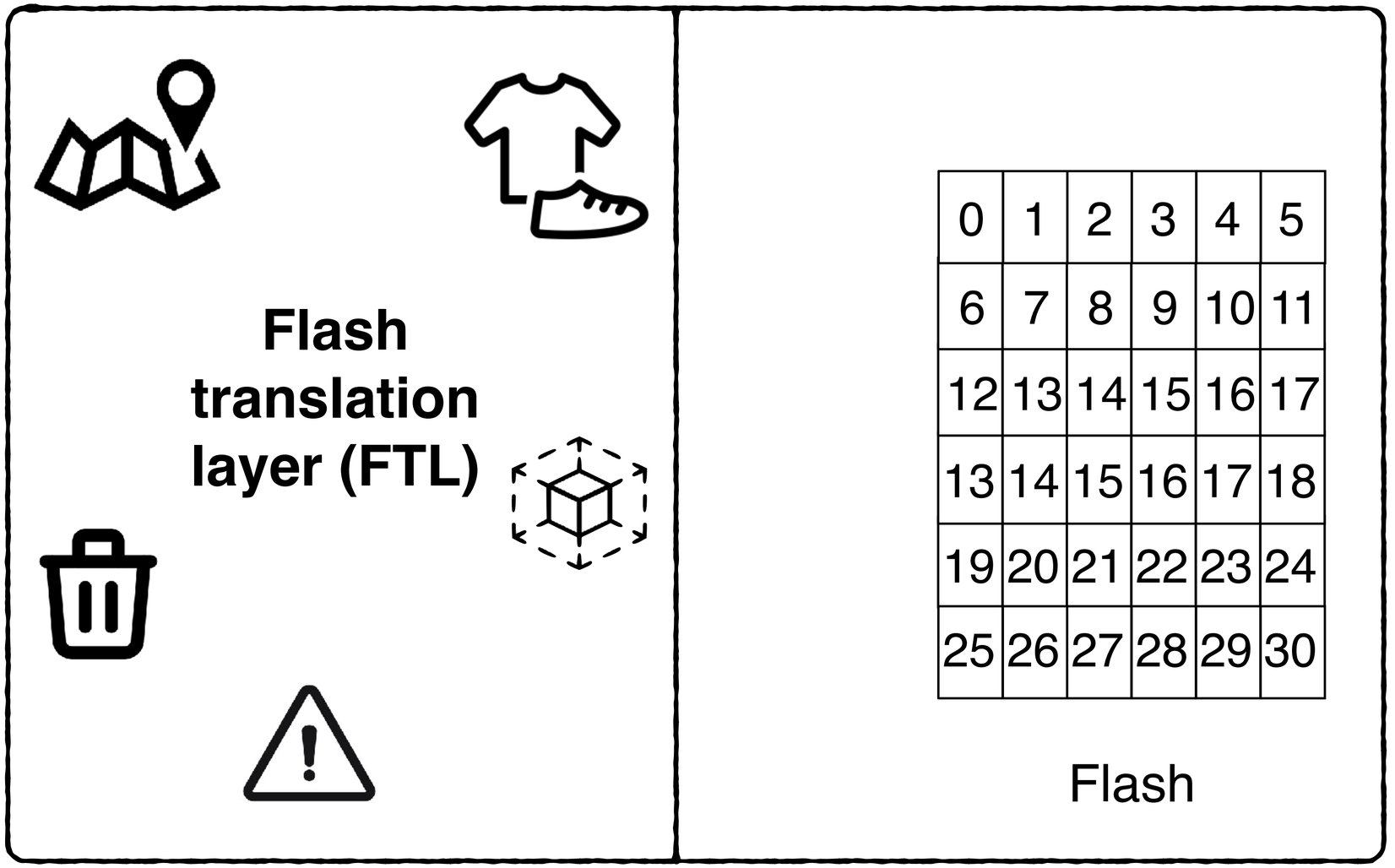
Garbage-collection

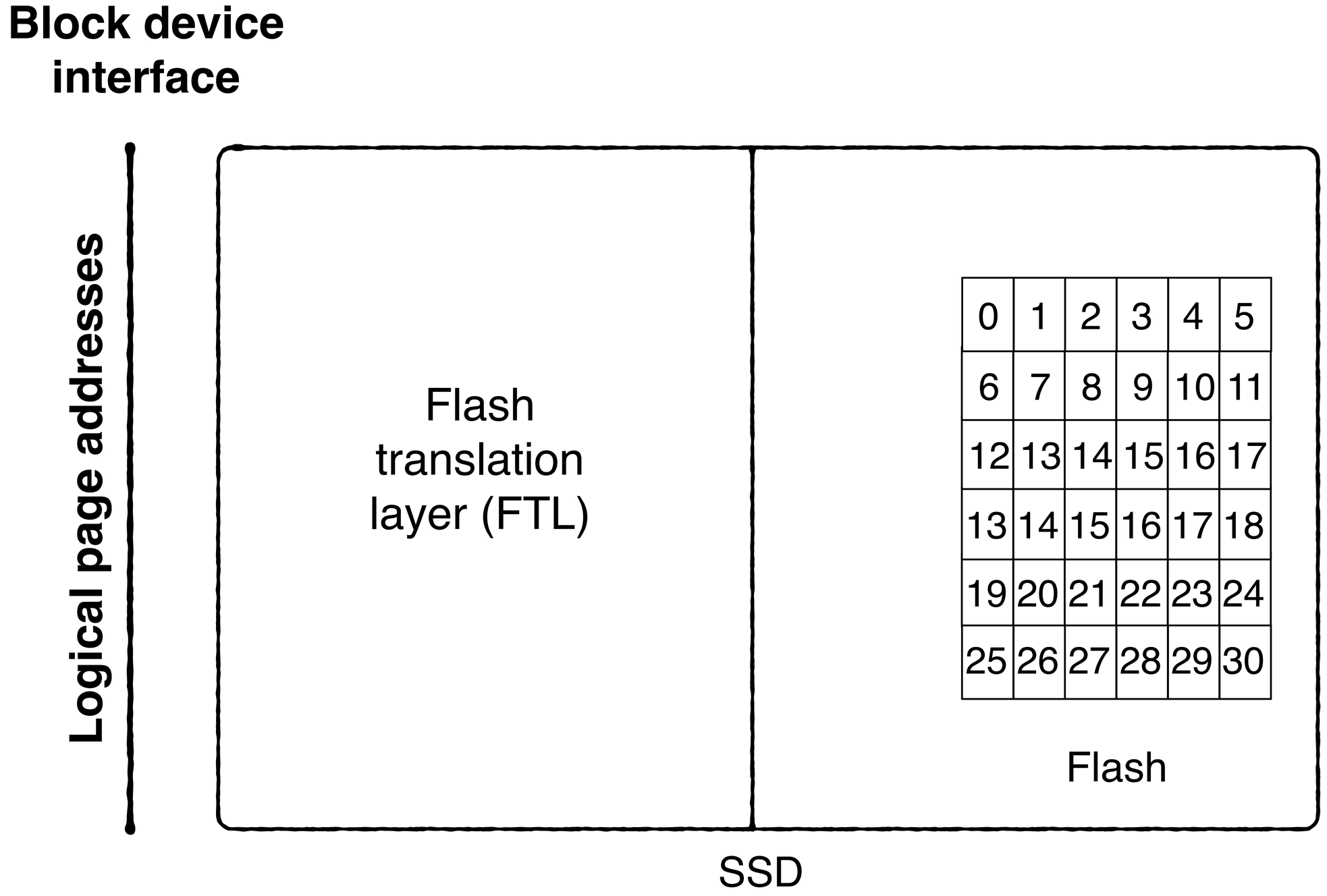
Wear-Leveling

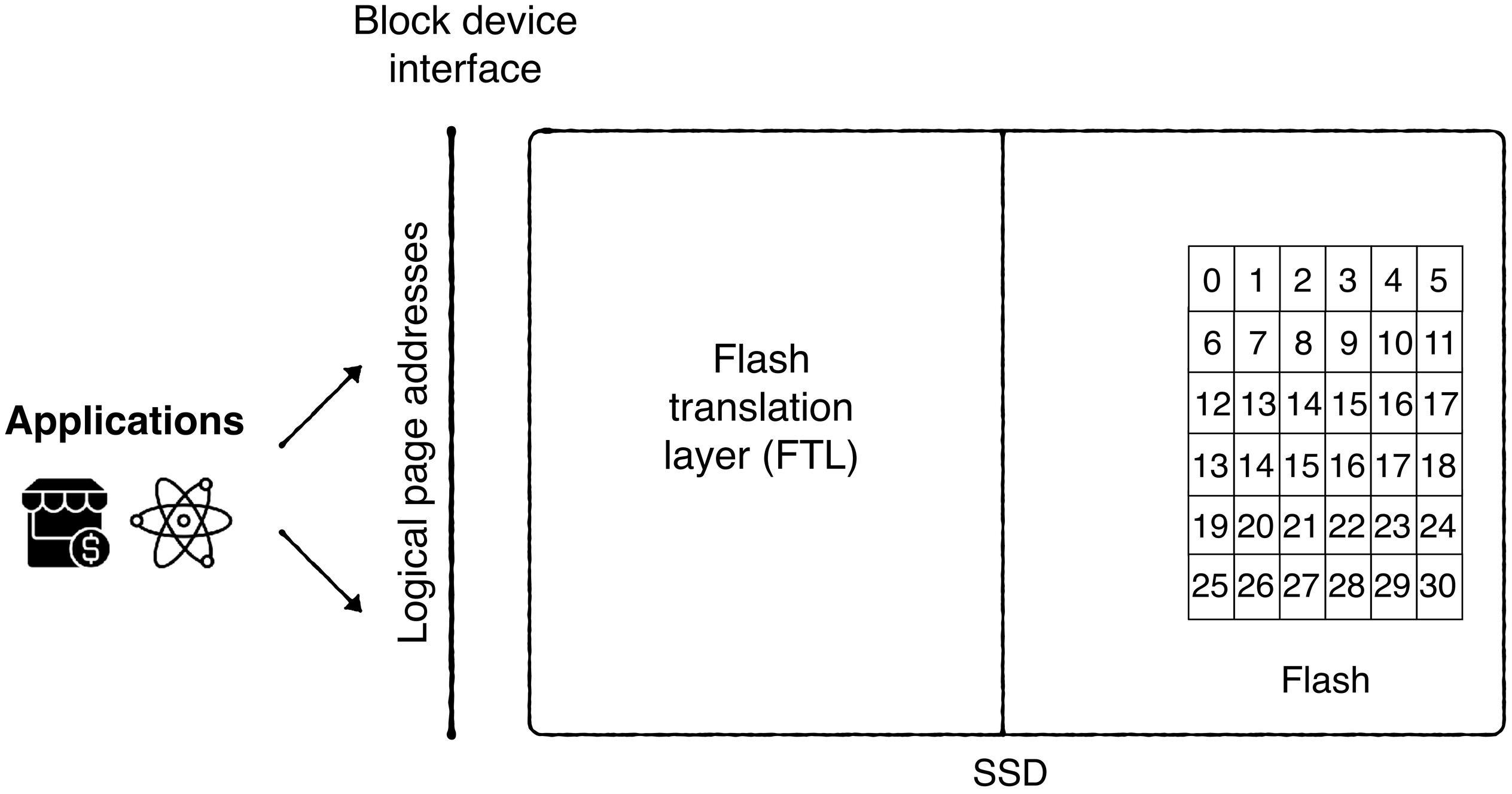
Error-Correction

Over-Provisioning

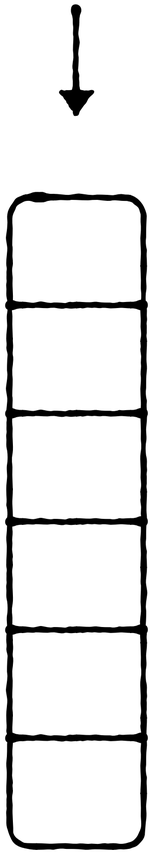



Flash
translation
layer (FTL)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
Flash
SSD


Block device interface
Logical page addresses
Flash translation layer (FTL)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
Flash
SSD


Block device
interface
Applications
Logical page addresses
Flash
translation
layer (FTL)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
Flash
SSD

Problem: the mapping table is large

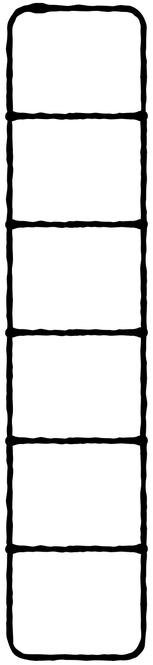


SRAM /
DRAM

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

Problem: the mapping table is large

e.g., 1TB SSD with 4KB pages requires 1GB mapping table



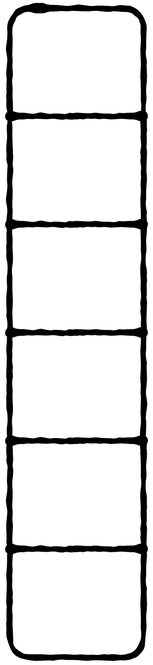
SRAM / DRAM

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

Problem: the mapping table is large

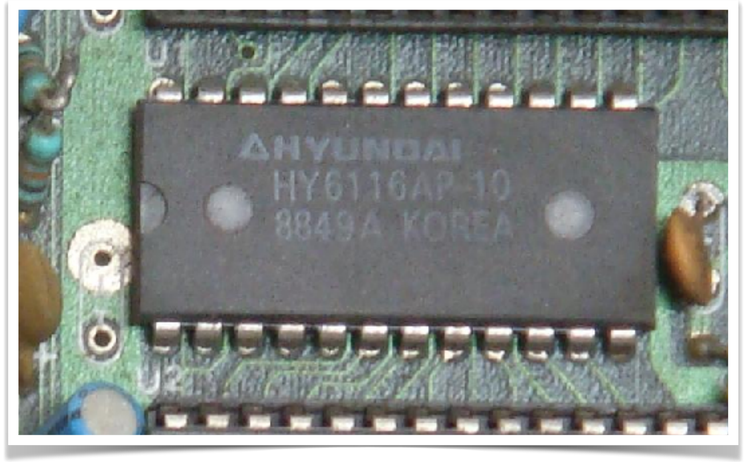
e.g., 1TB SSD with 4KB pages requires 1GB mapping table

Doesn’t sound like a lot but…

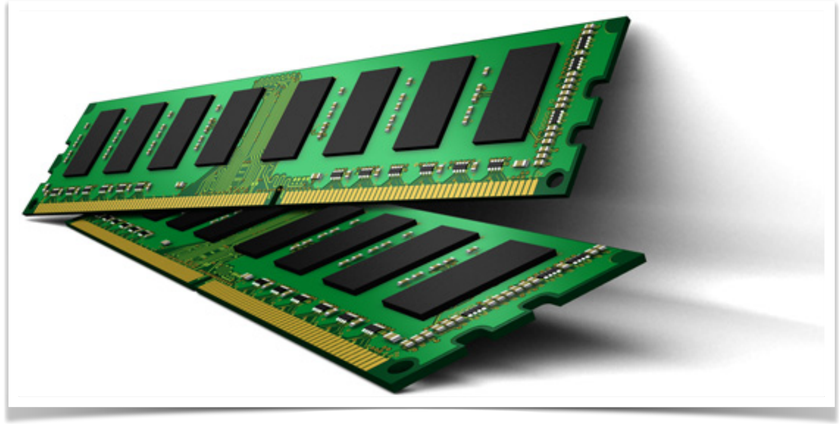


SRAM / DRAM

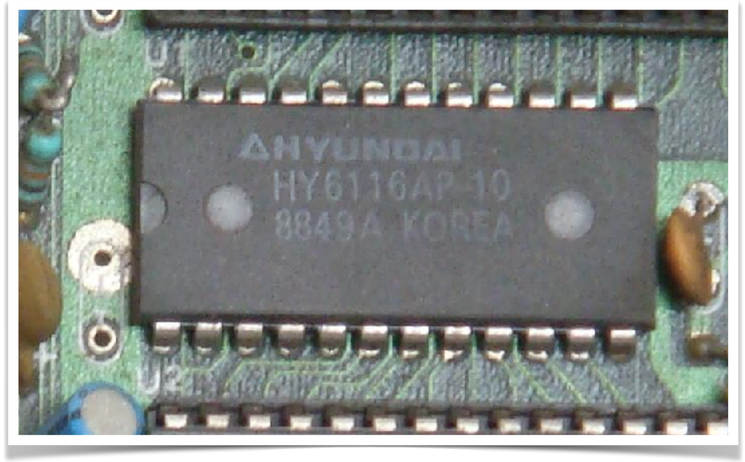
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30



Static RAM (SRAM)

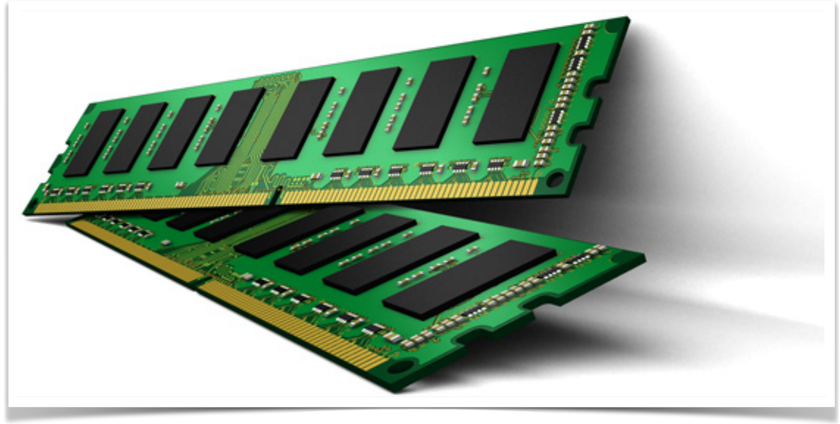


Dynamic RAM (DRAM)



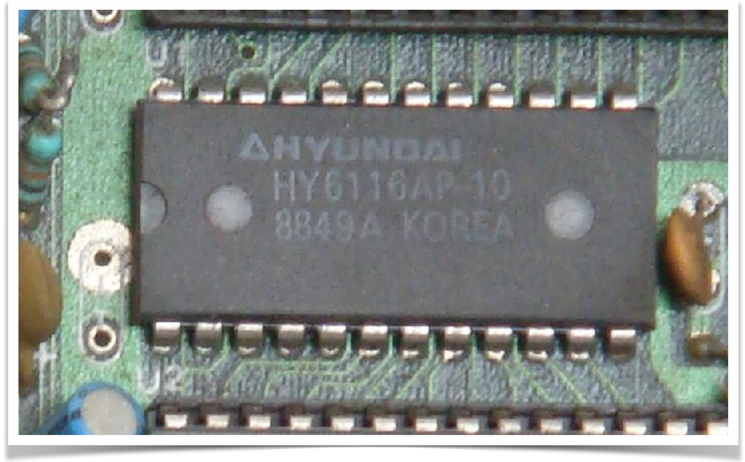
Static RAM (SRAM)

No refreshing



Dynamic RAM (DRAM)

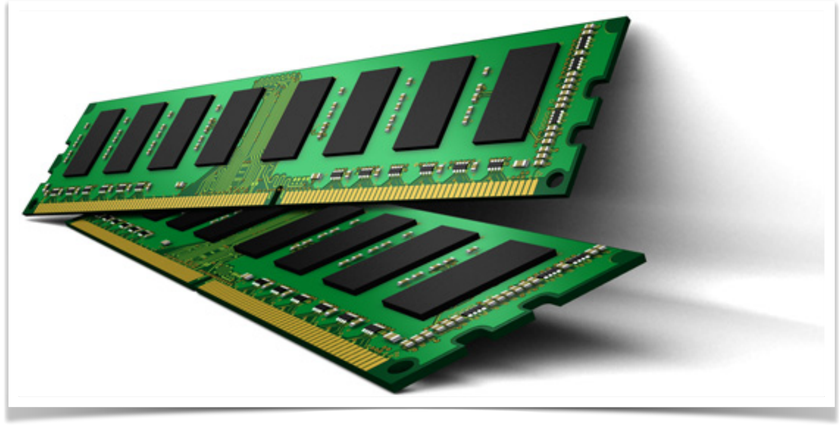
Requires refreshing even when idle



Static RAM (SRAM)

No refreshing

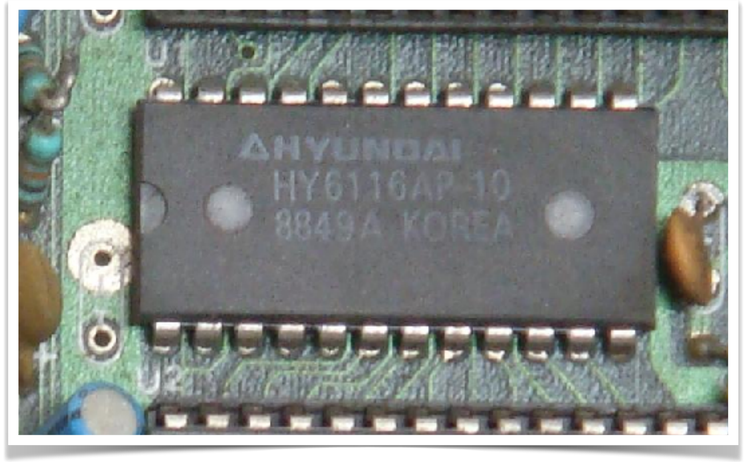
More energy efficient



Dynamic RAM (DRAM)

Requires refreshing even when idle

More power hungry

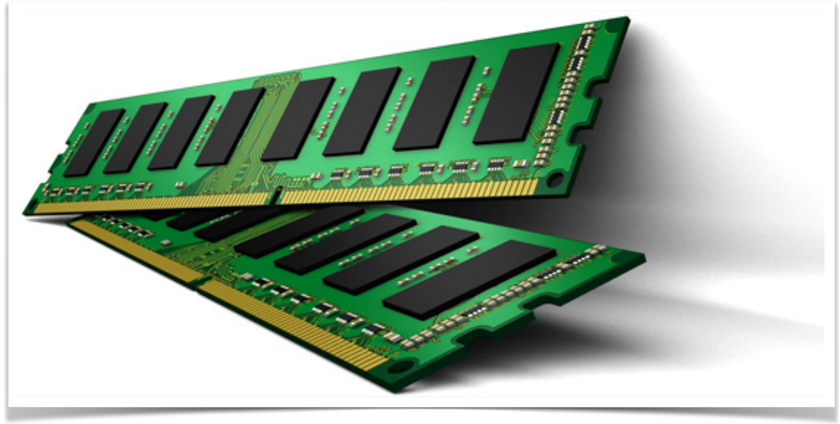


Static RAM (SRAM)

No refreshing

More energy efficient

Less error prone

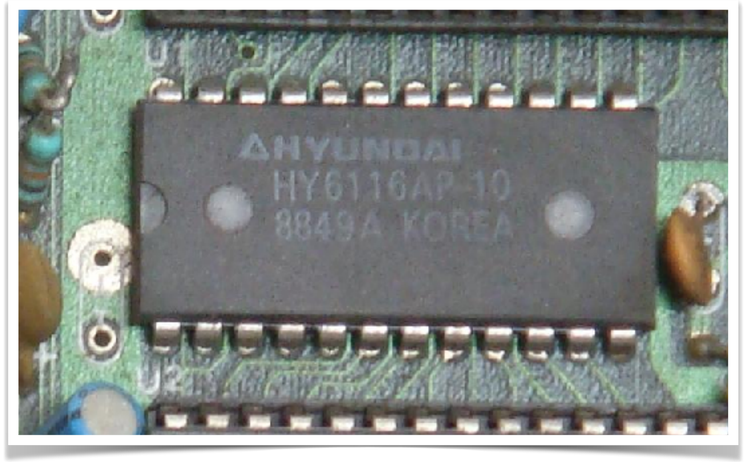


Dynamic RAM (DRAM)

Requires refreshing even when idle

More power hungry

bit flips due to charge leakage (requires error correction codes)



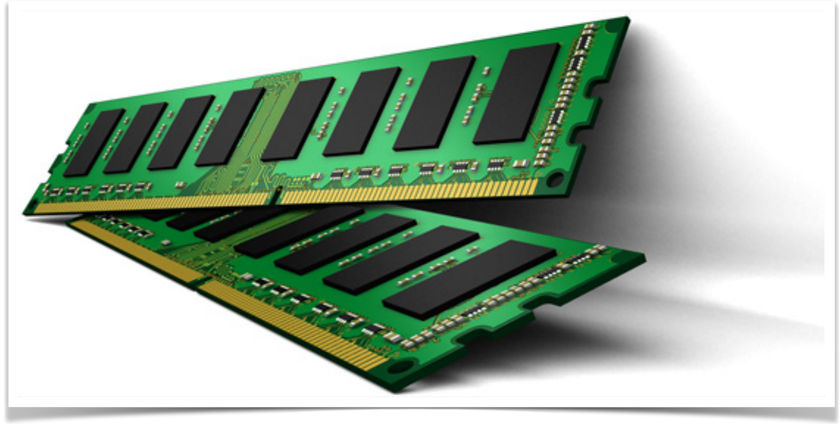
Static RAM (SRAM)

No refreshing

More energy efficient

Less error prone

Longer lifespan



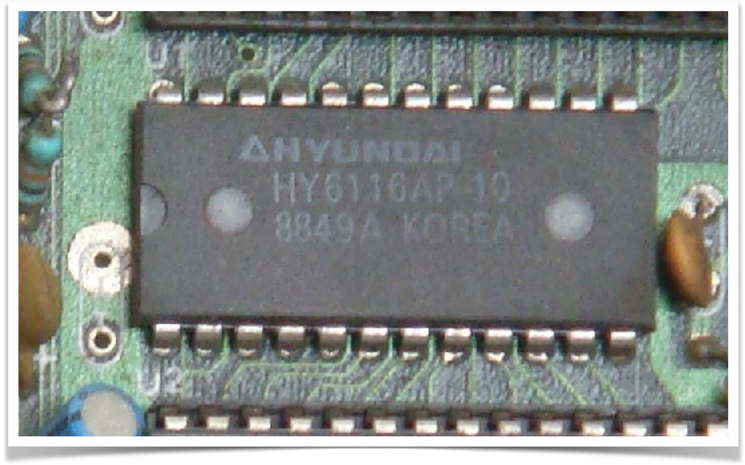
Dynamic RAM (DRAM)

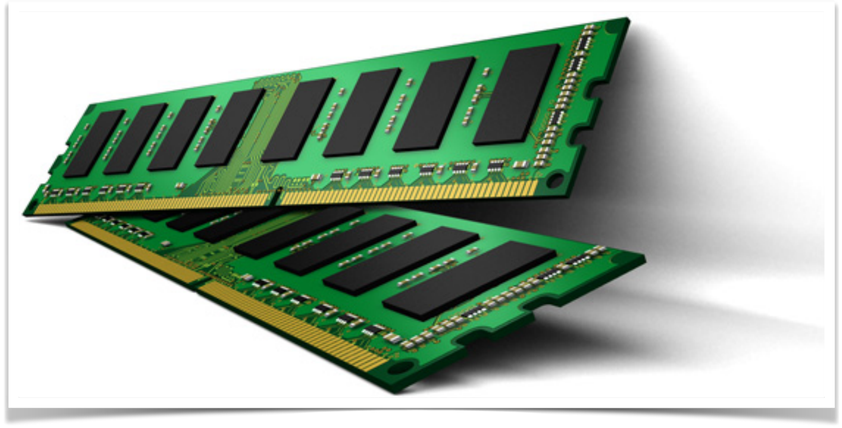
Requires refreshing even when idle

More power hungry

bit flips due to charge leakage
(requires error correction codes)

Shorter lifespan





Static RAM (SRAM)

No refreshing

More energy efficient

Less error prone

Longer lifespan

$1000 per GB

Dynamic RAM (DRAM)

Requires refreshing even when idle

More power hungry

bit flips due to charge leakage
(requires error correction codes)

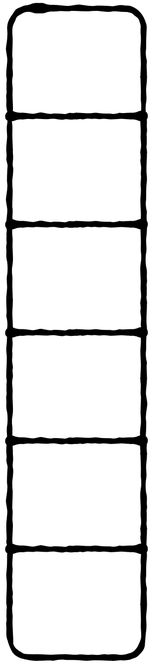
Shorter lifespan

$5 per GB

Problem: the mapping table is large

e.g., 1TB SSD with 4KB pages requires 1GB mapping table

Doesn’t sound like a lot but…

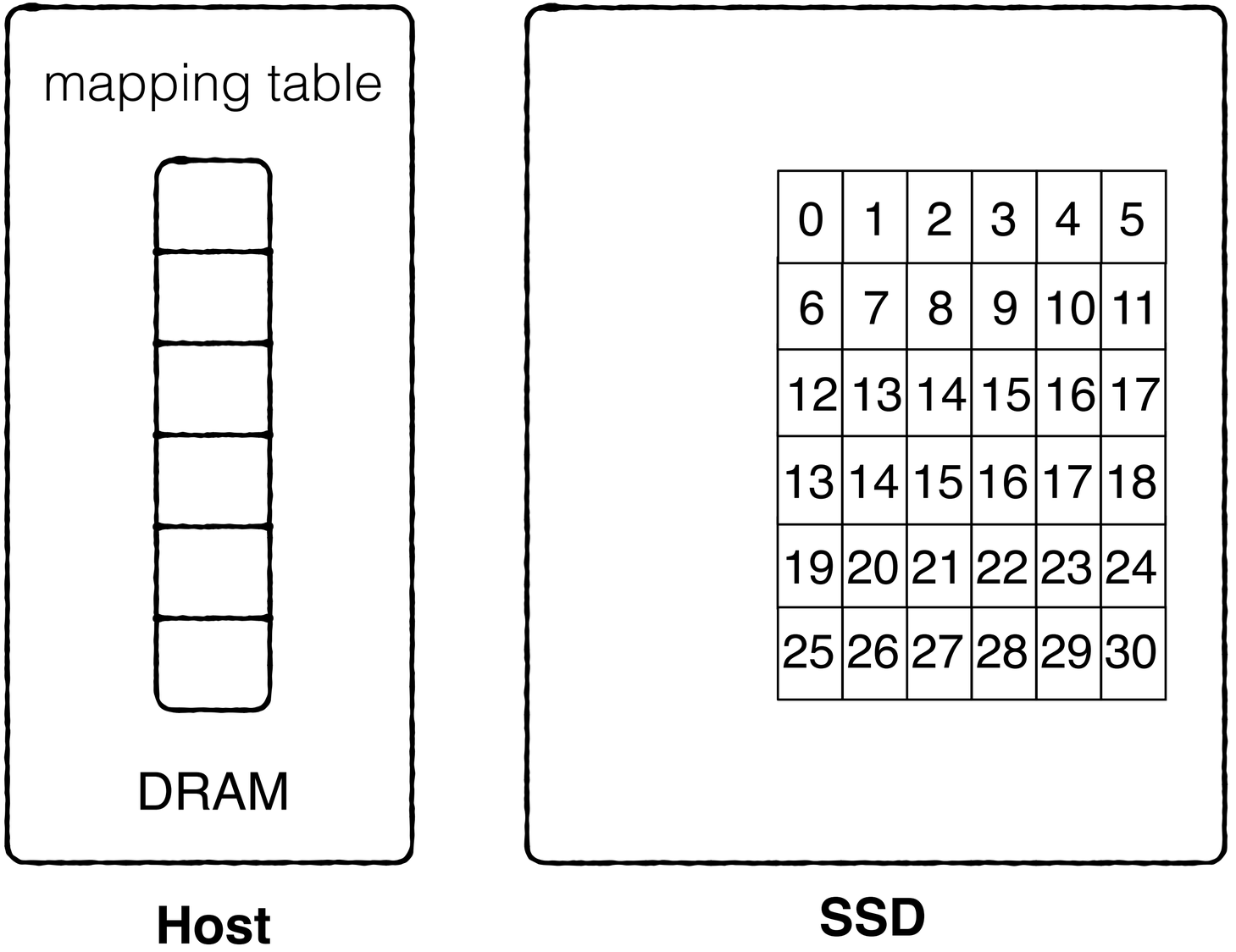


SRAM / DRAM

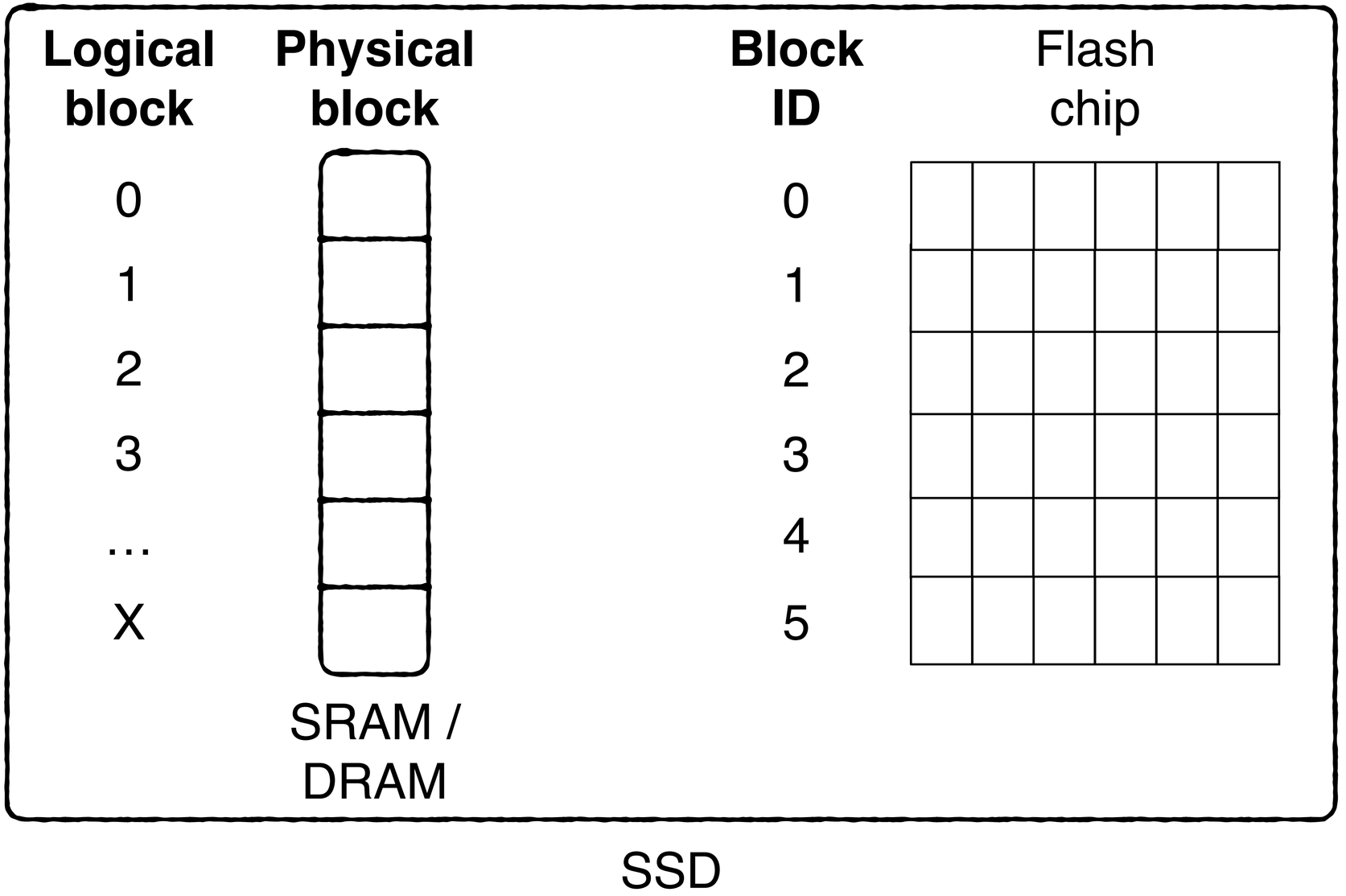
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

e.g., 1TB SSD with 4KB pages requires 1GB mapping table

Can store in host to save space (High-end SSDs)



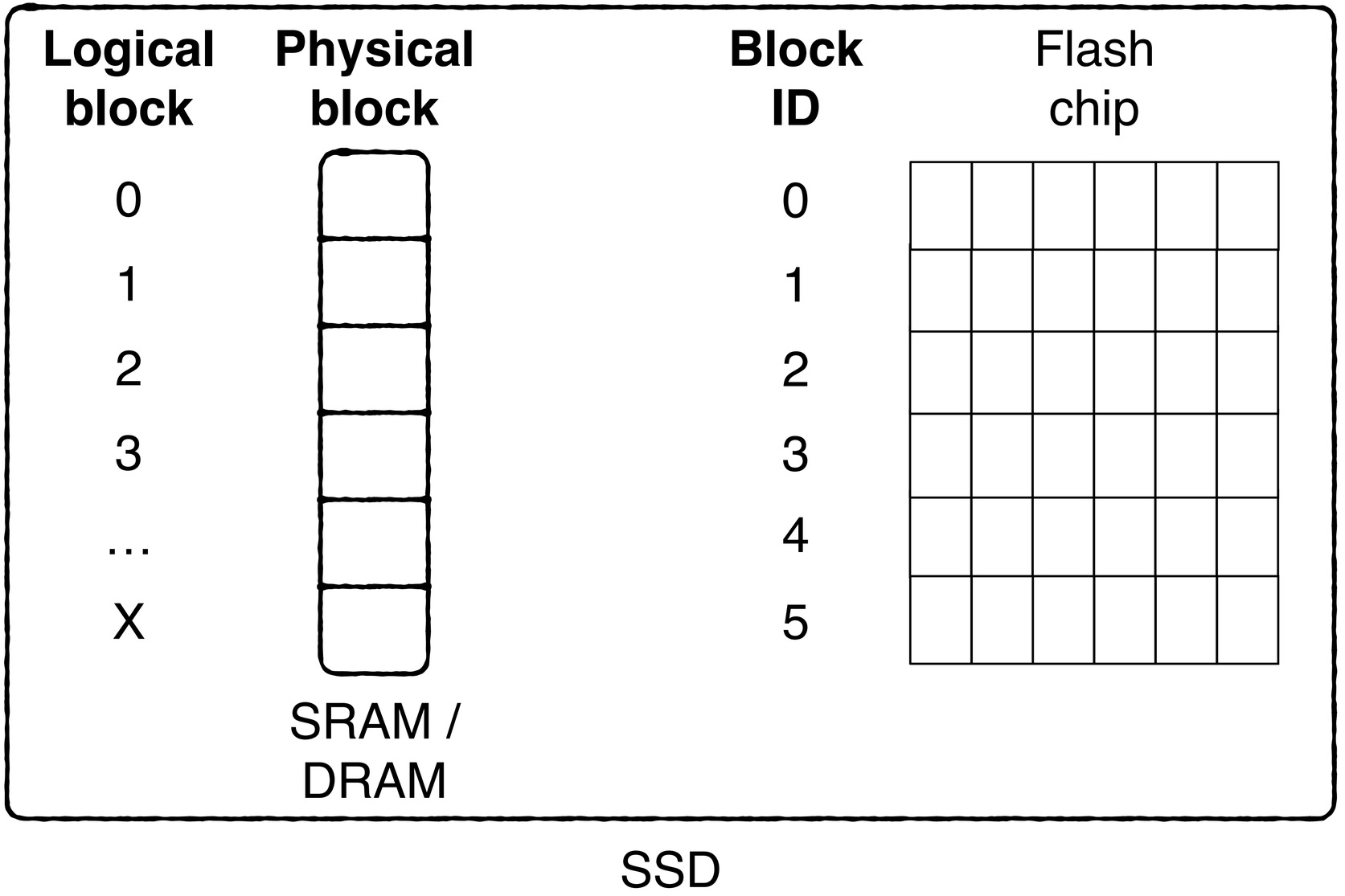
Alternative 1: use block mapping

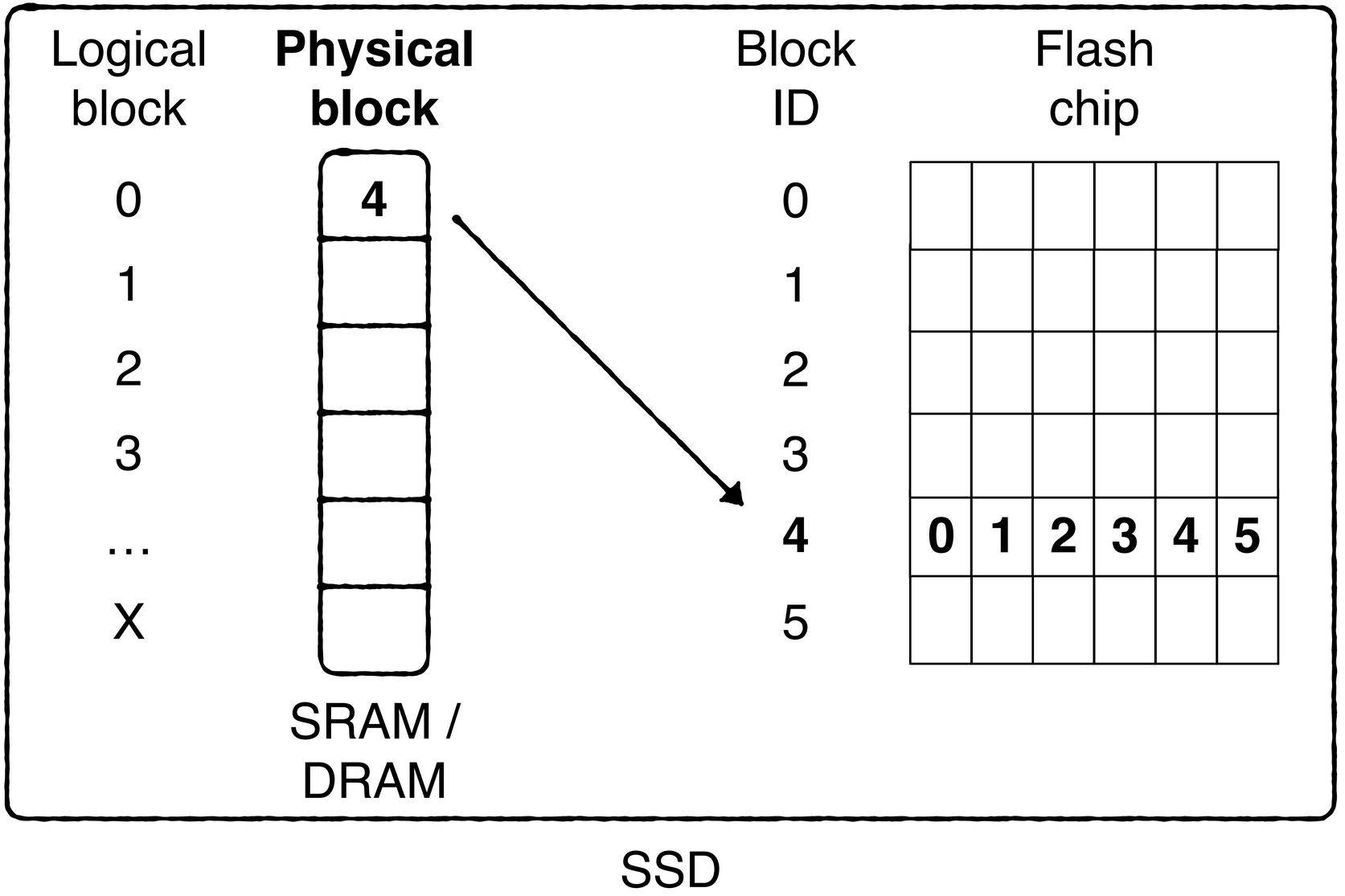


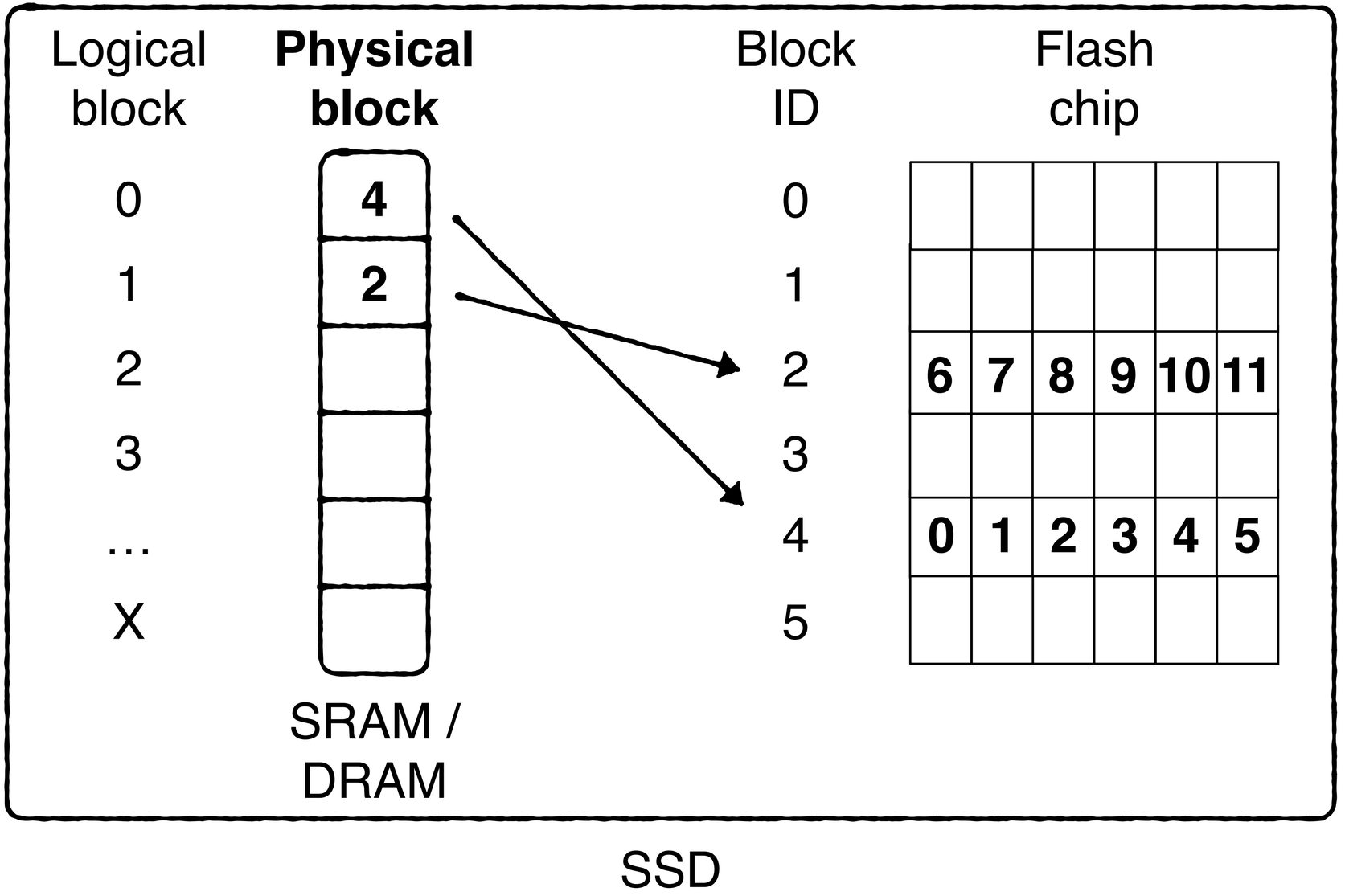
Alternative 1: use block mapping

FAST: A log buffer-based flash translation layer using fully-associative sector translation

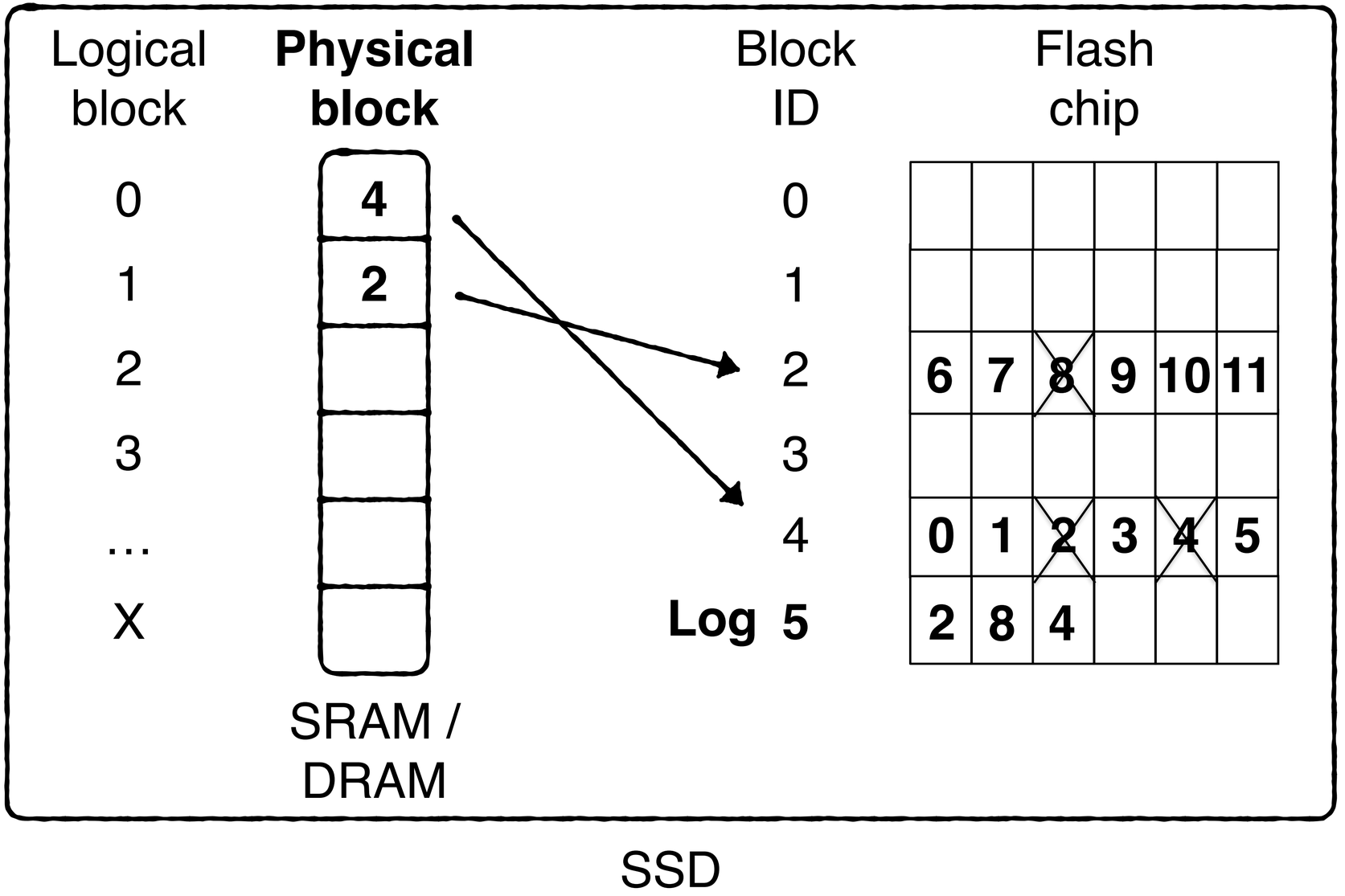
TECS 2007.



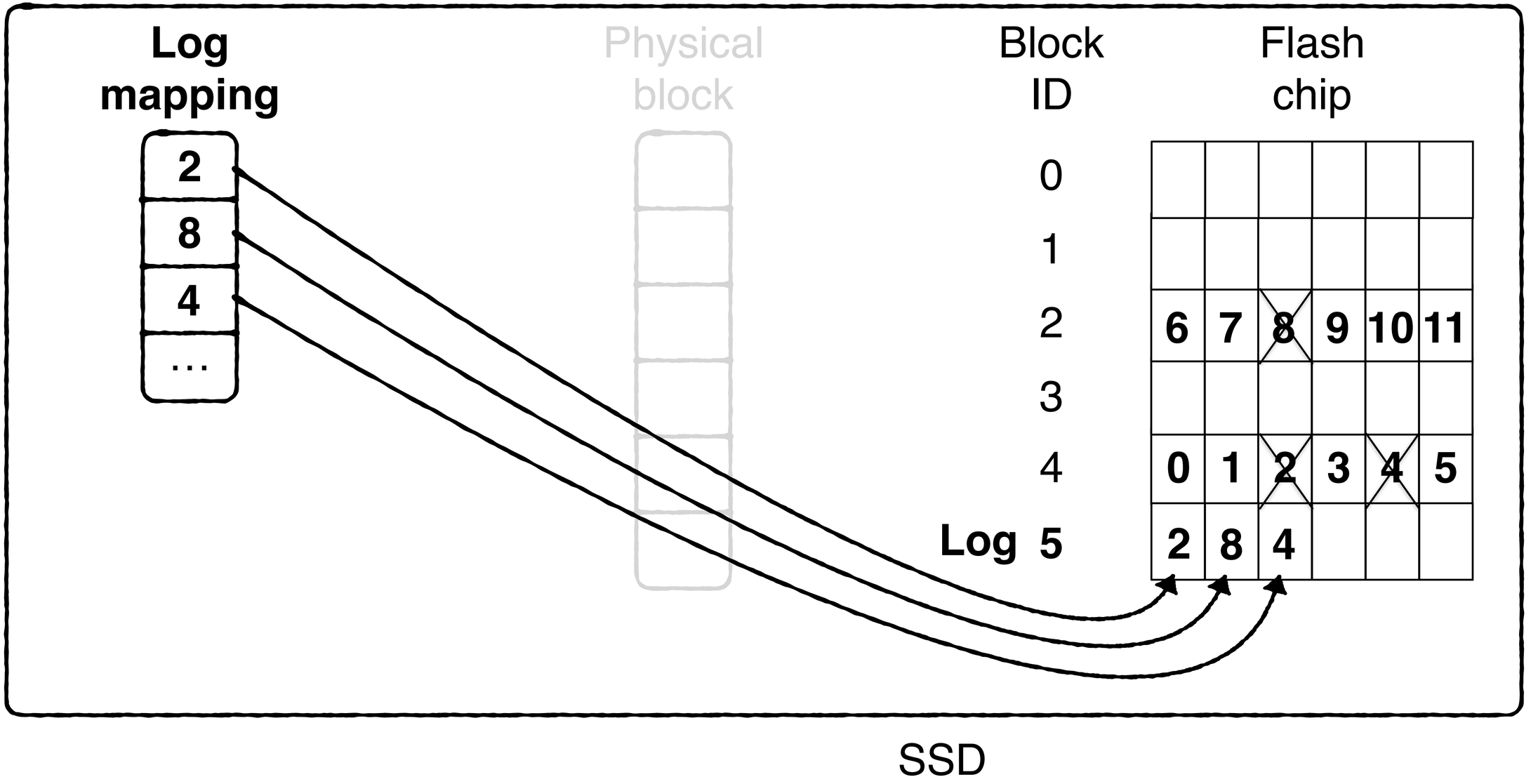

Logical block
Physical block
Block ID
Flash chip
0
1
2
3
...
X
4
0
1
2
3
4
5
0 1 2 3 4 5
SRAM / DRAM
SSD


Logical block
Physical block
Block ID
Flash chip
0
1
2
3
...
X
4
2
0
1
2
3
4
5
6 7 8 9 10 11
0 1 2 3 4 5
SRAM / DRAM
SSD

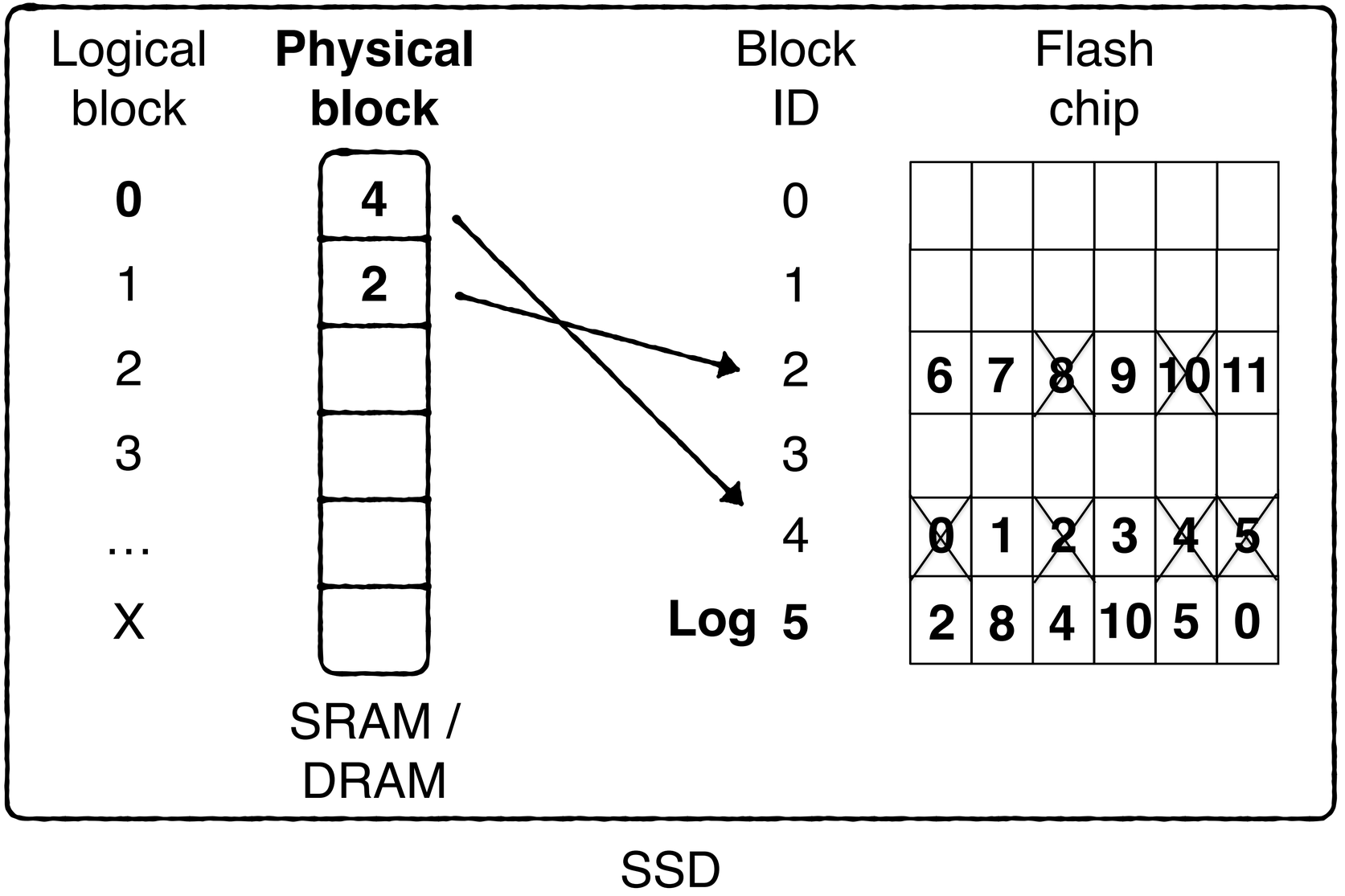
Store updates in log blocks



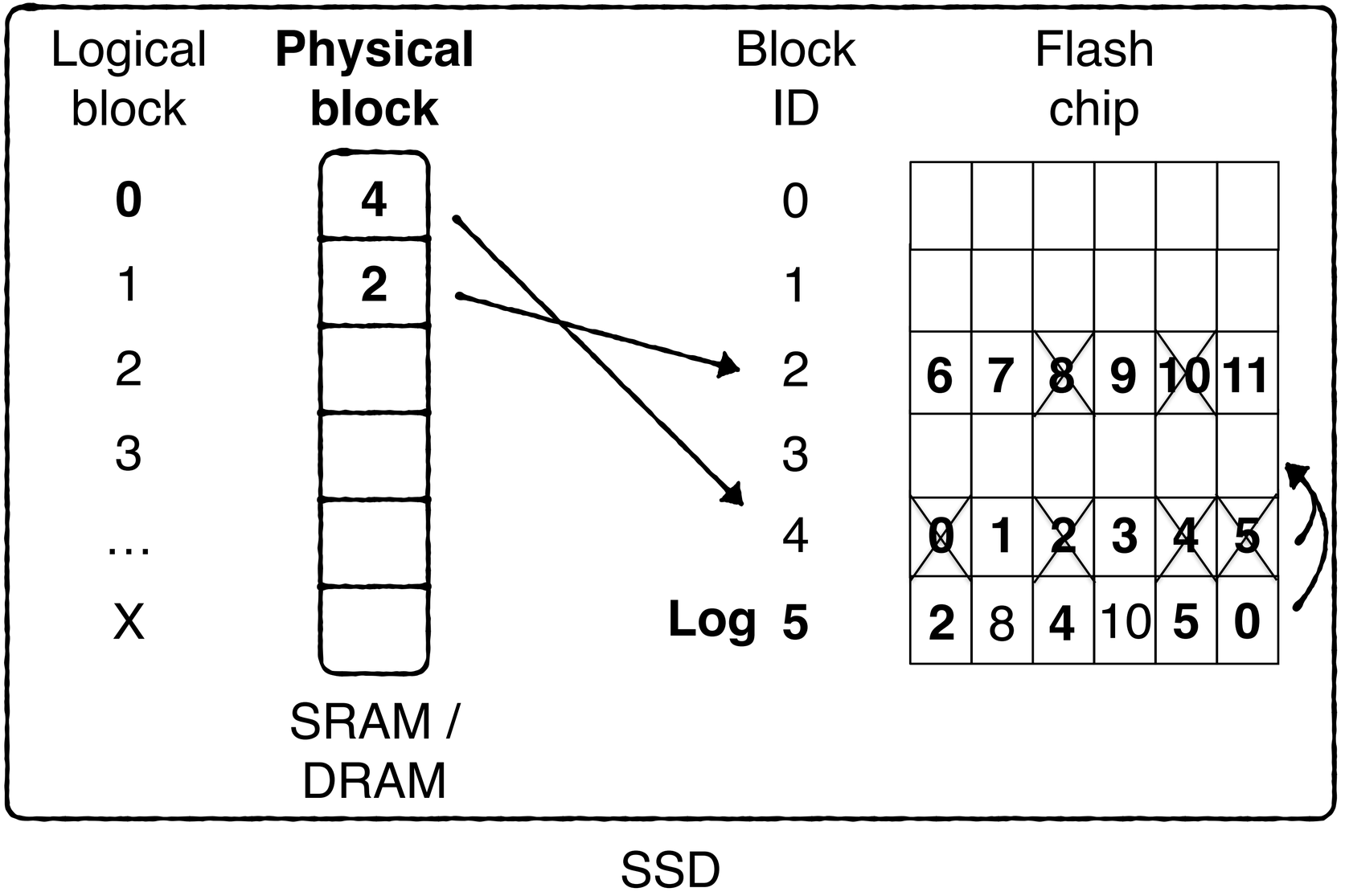
log blocks use page mapping. There are few of them



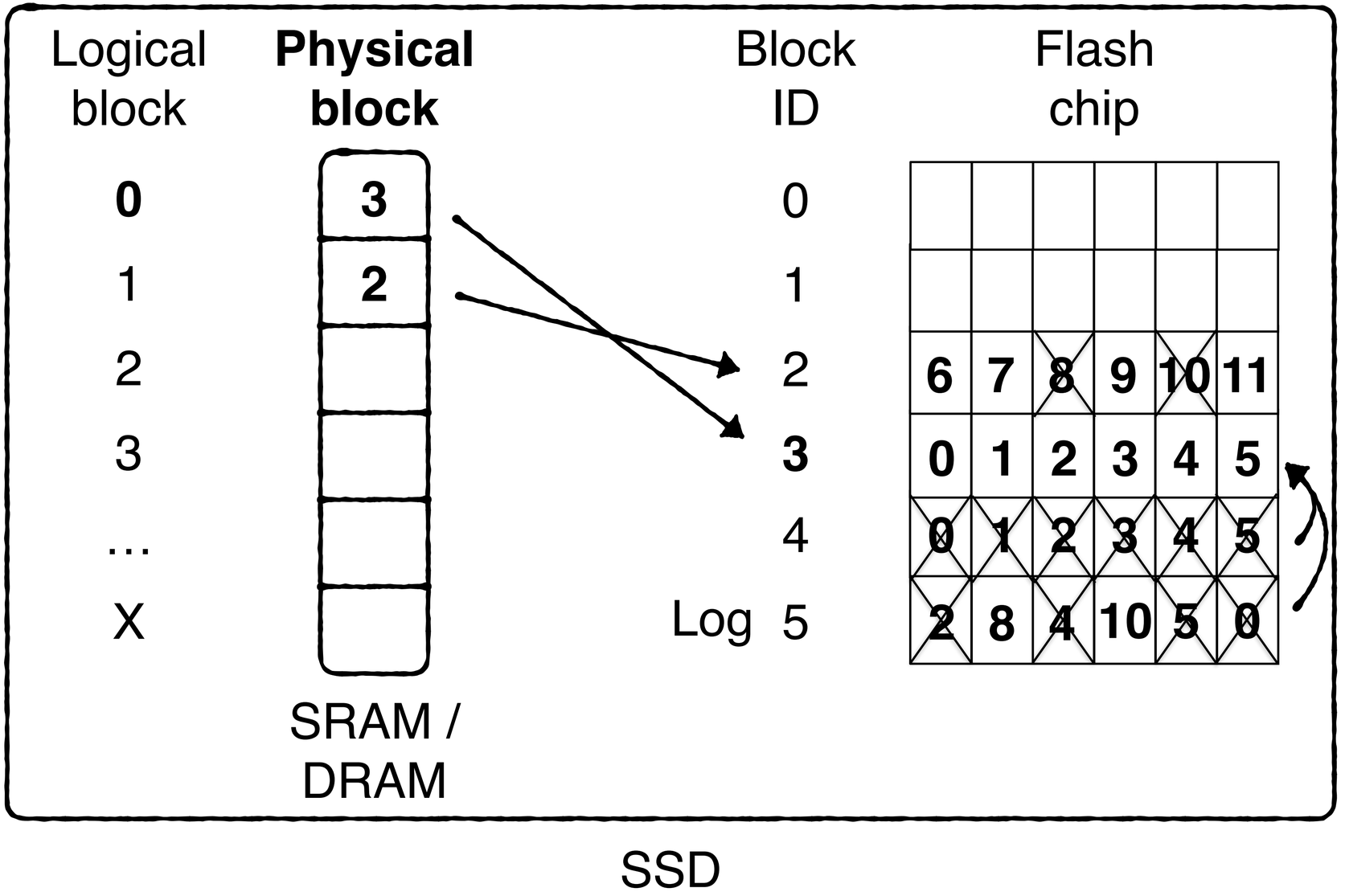
When log block fills up, merge



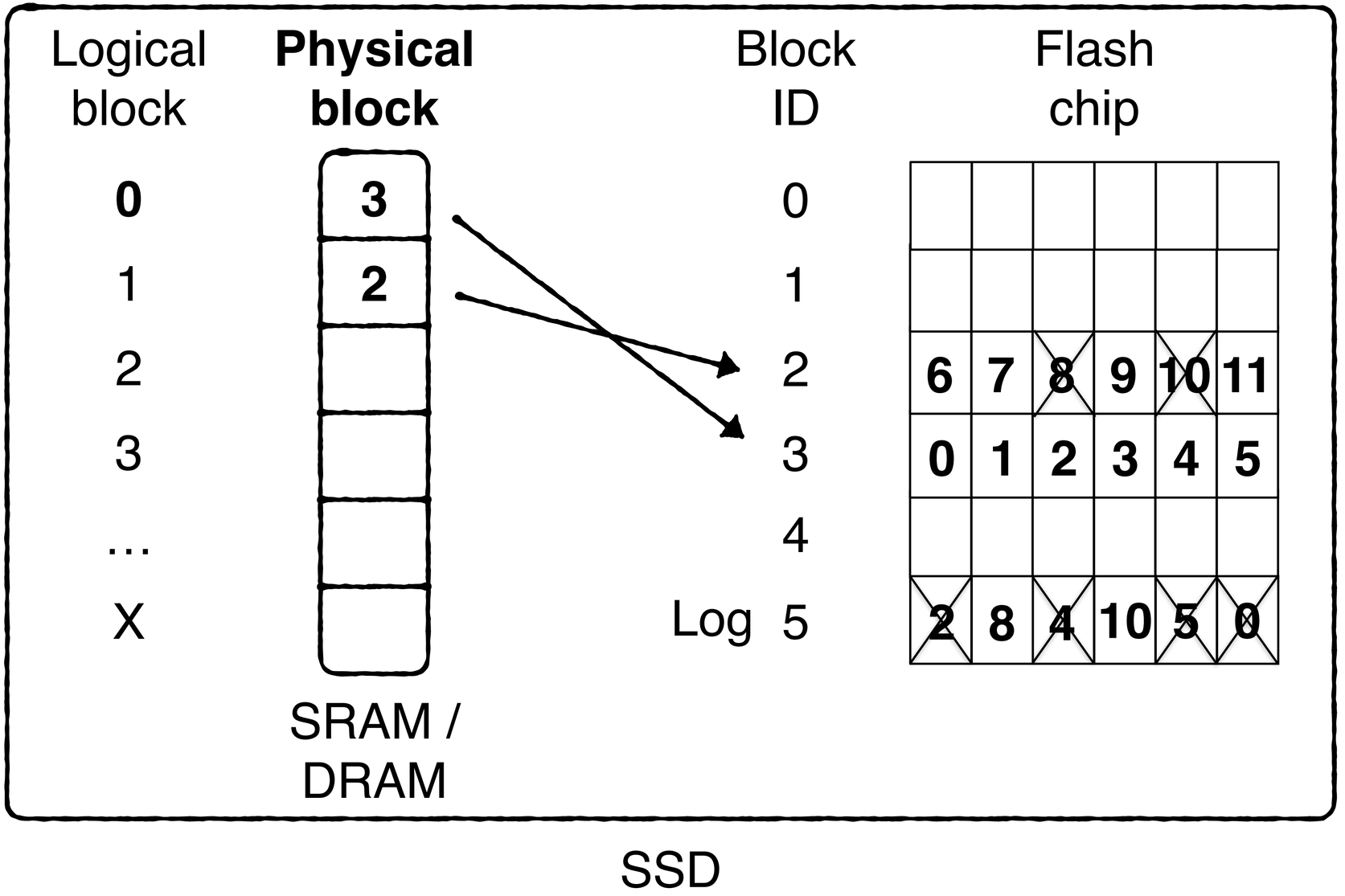
When log block fills up, merge



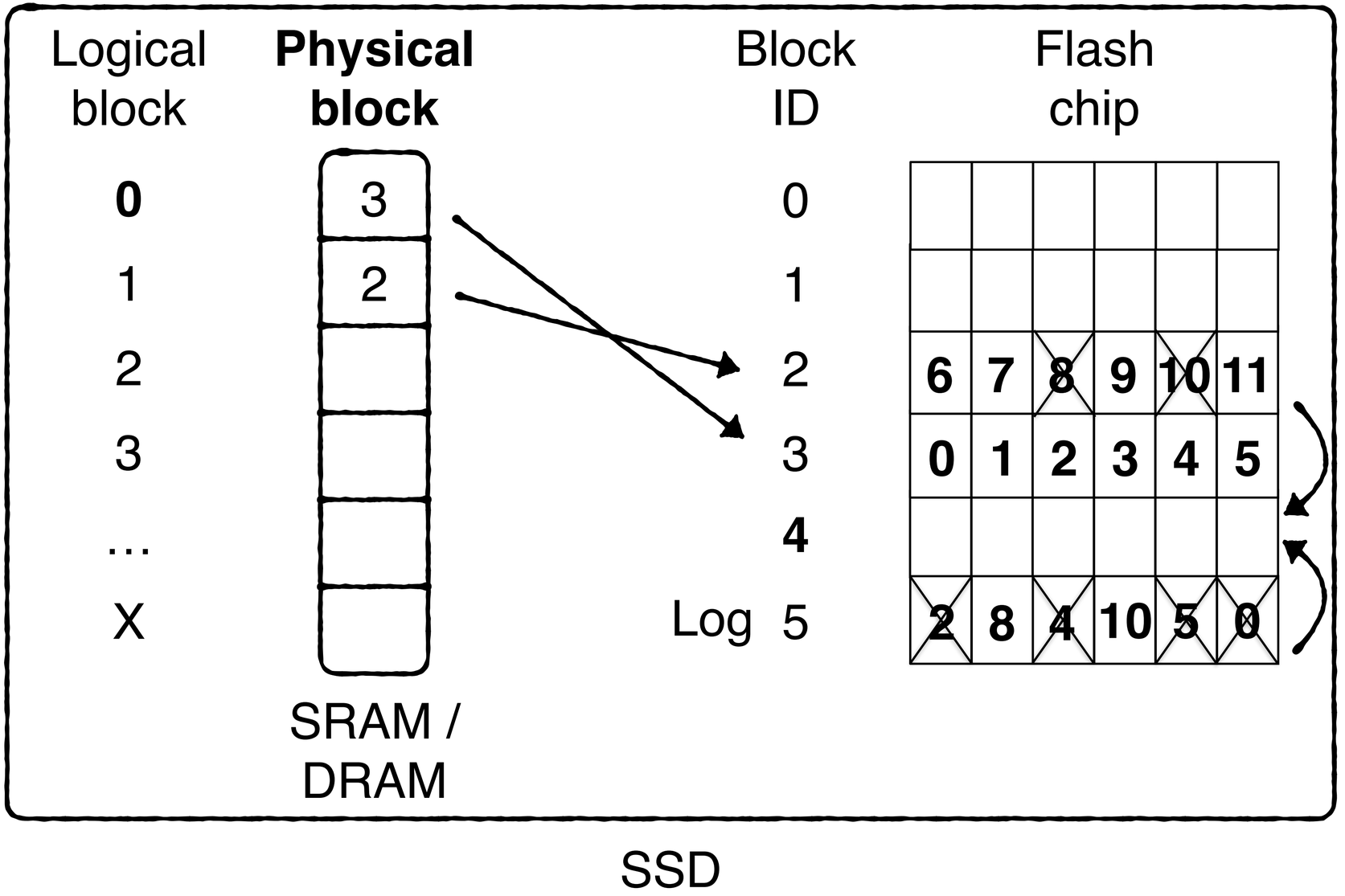
When log block fills up, merge



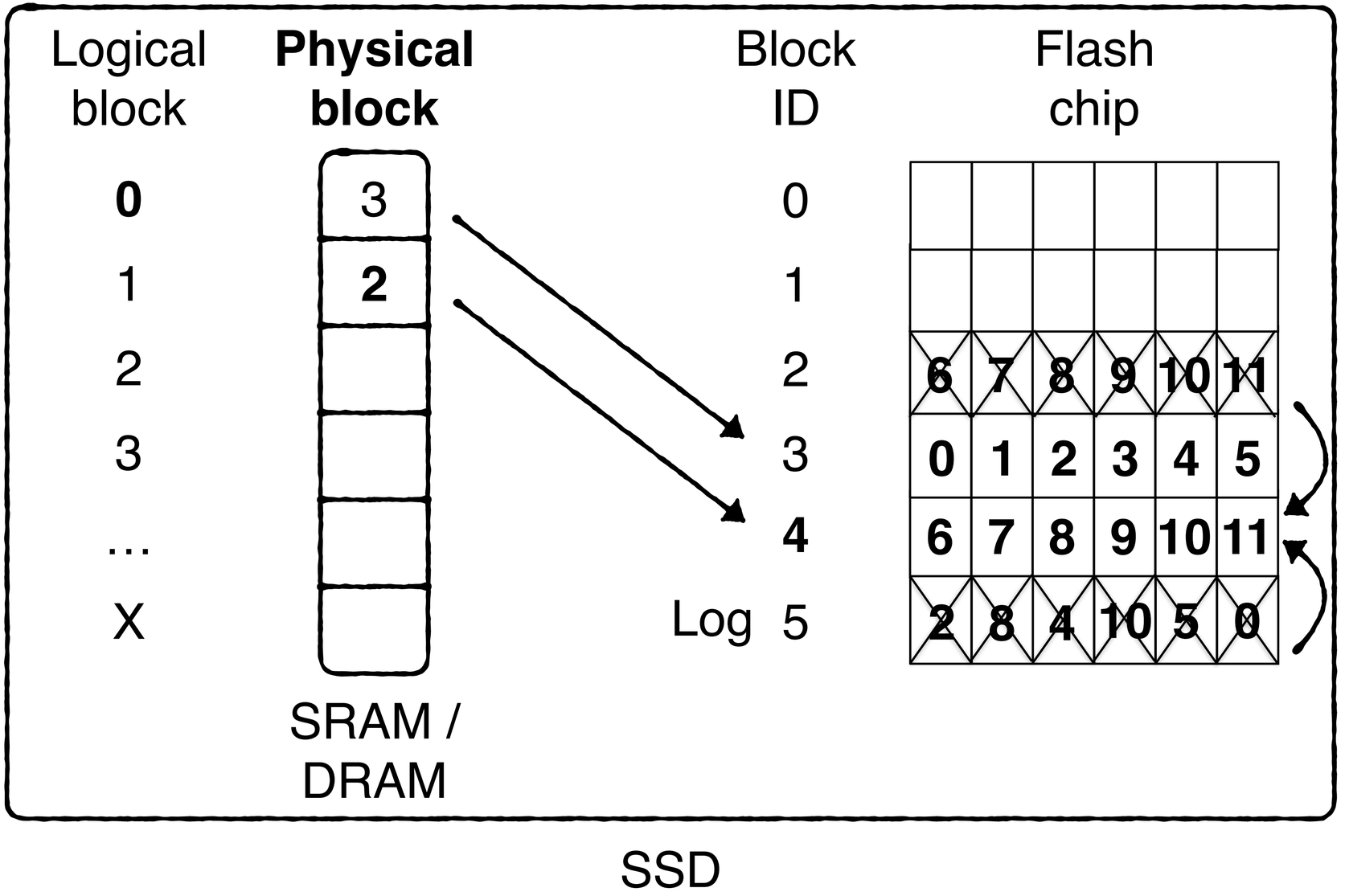
When log block fills up, merge



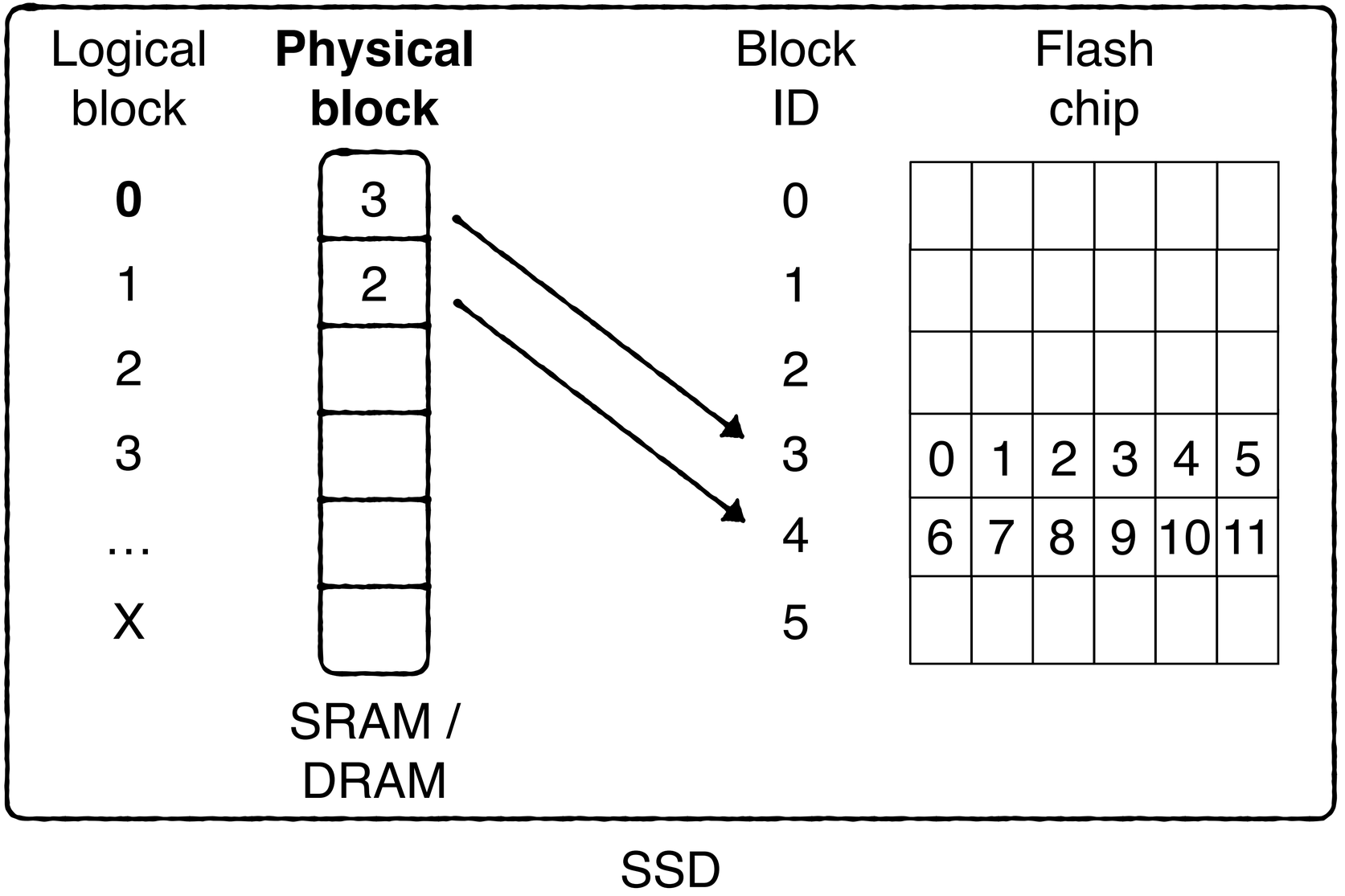
When log block fills up, merge



When log block fills up, merge

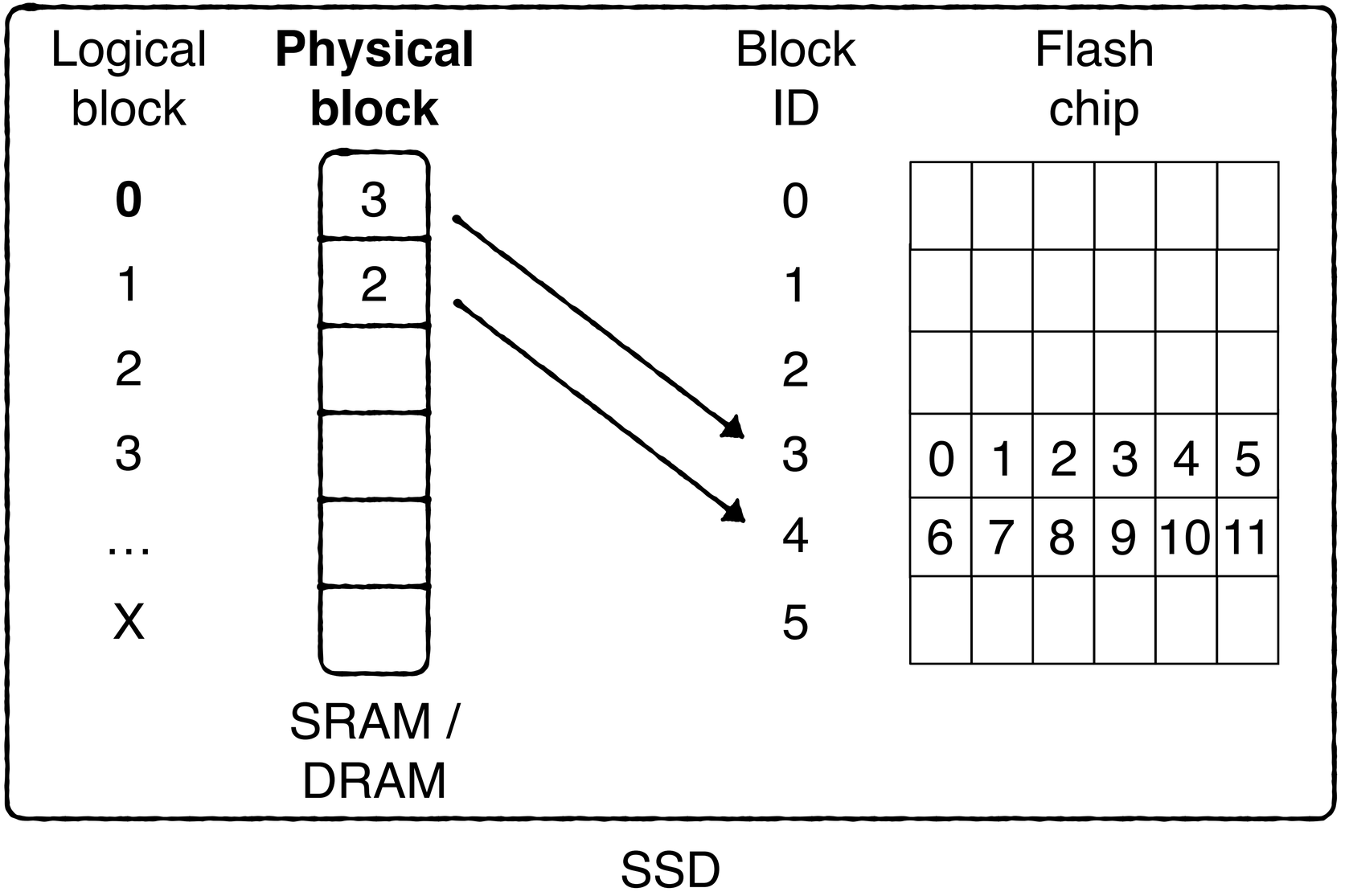


Downsides?


Logical block
Physical block
Block ID
Flash chip
0
1
2
3
...
X
3
2
0
1
2
3
4
5
0 1 2 3 4 5
6 7 8 9 10 11
SRAM / DRAM
SSD

Downsides?

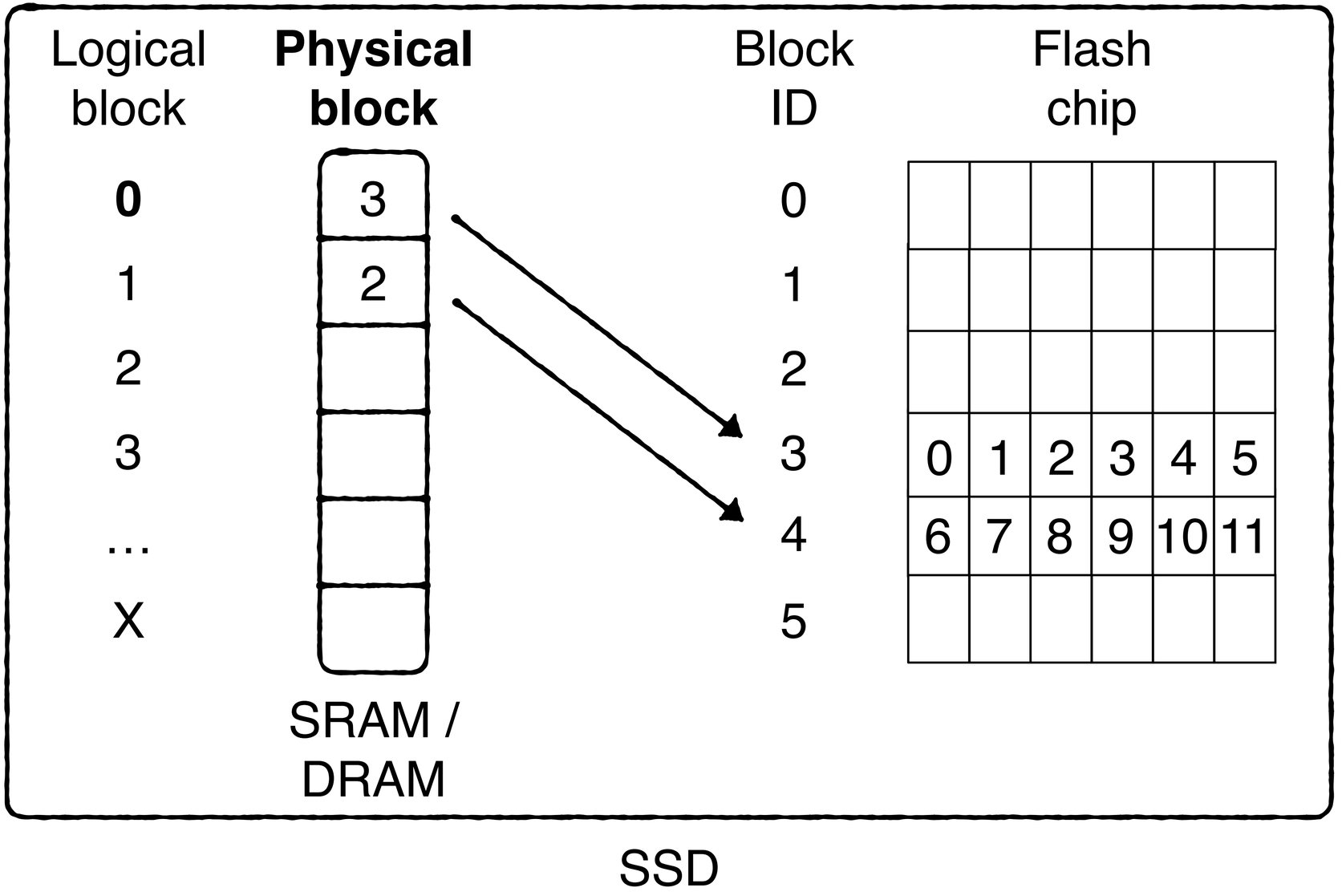
Coupling between mapping table and GC.



Downsides?

Coupling between mapping table and GC.

One log block can have updates from different data blocks, leading to long GC process that rewrites many blocks

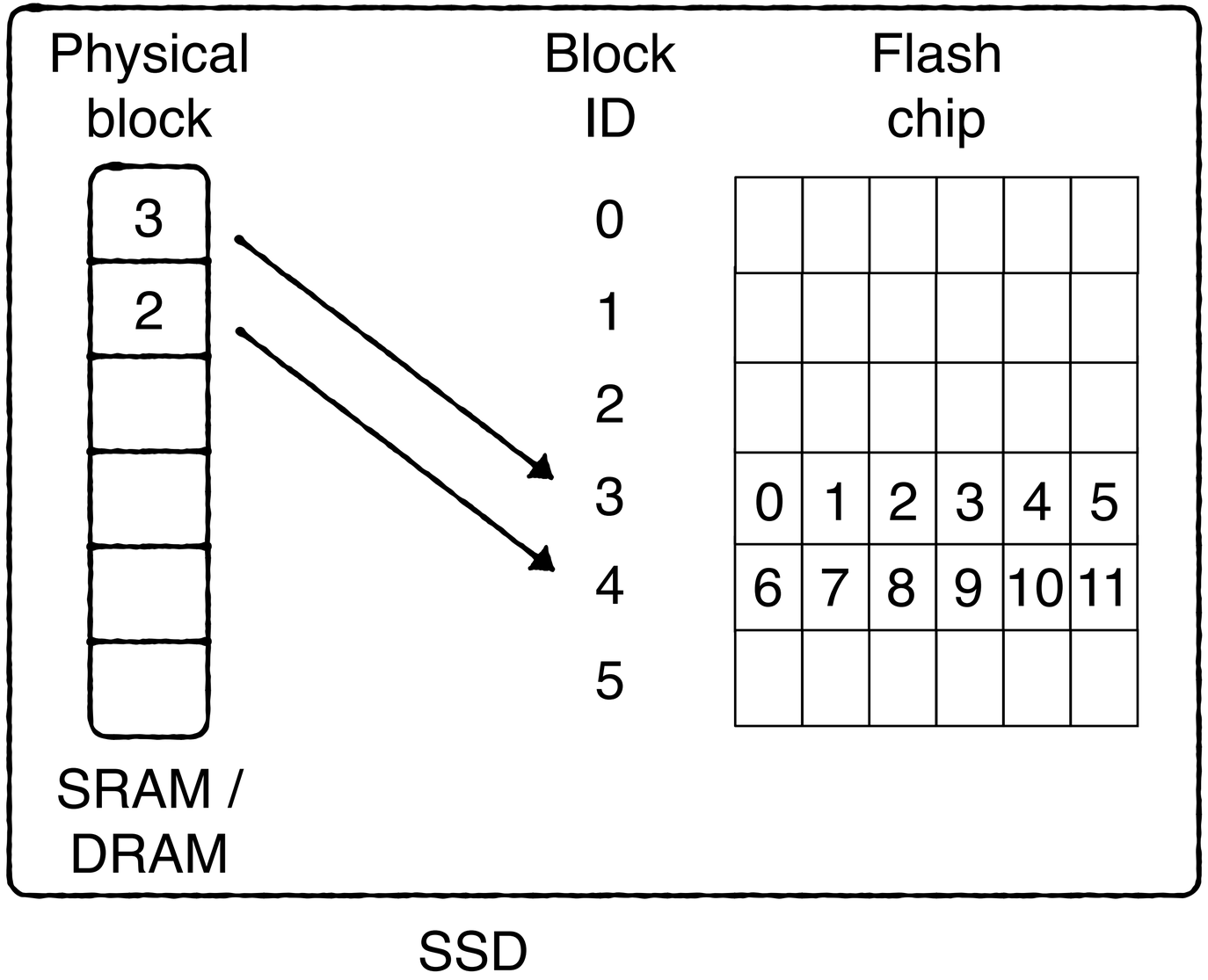


Downsides?

Coupling between mapping table and GC.

Sequential writes

Random writes



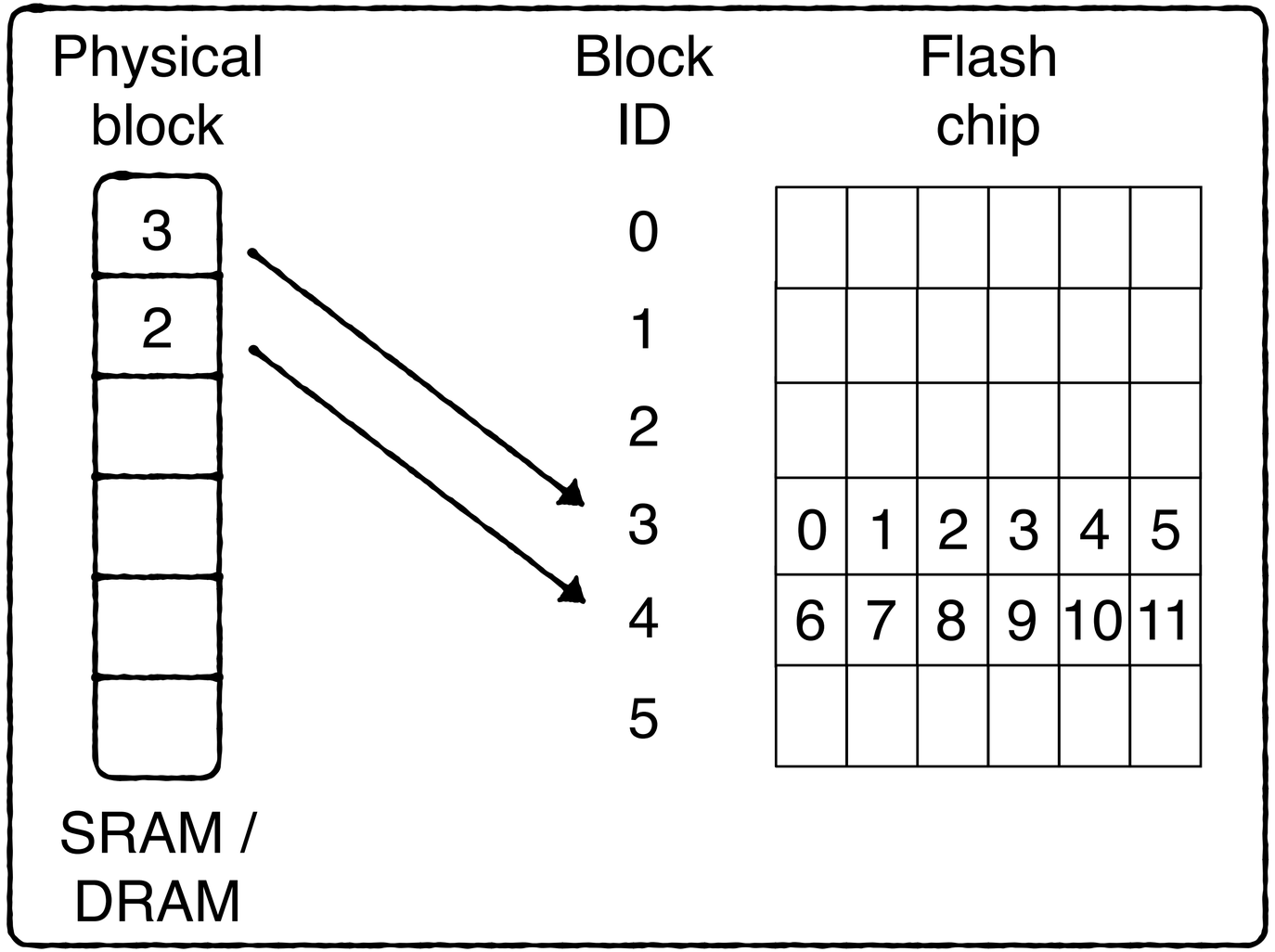
Better ideas? :)

Downsides?

Coupling between mapping table and GC.

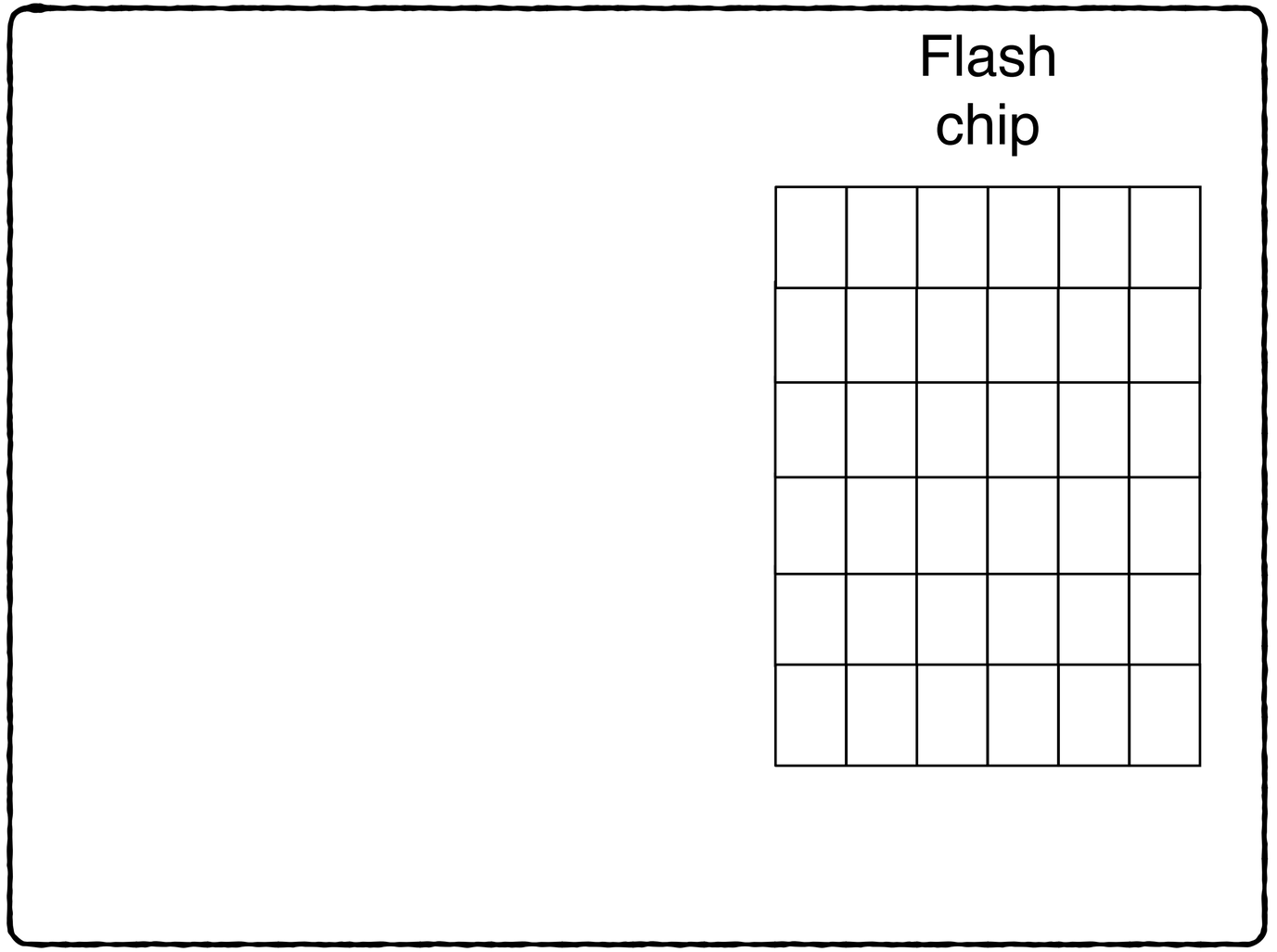
Sequential writes

Random writes



SSD

Alternative 2: Store Page Mapping in Flash & Cache Parts

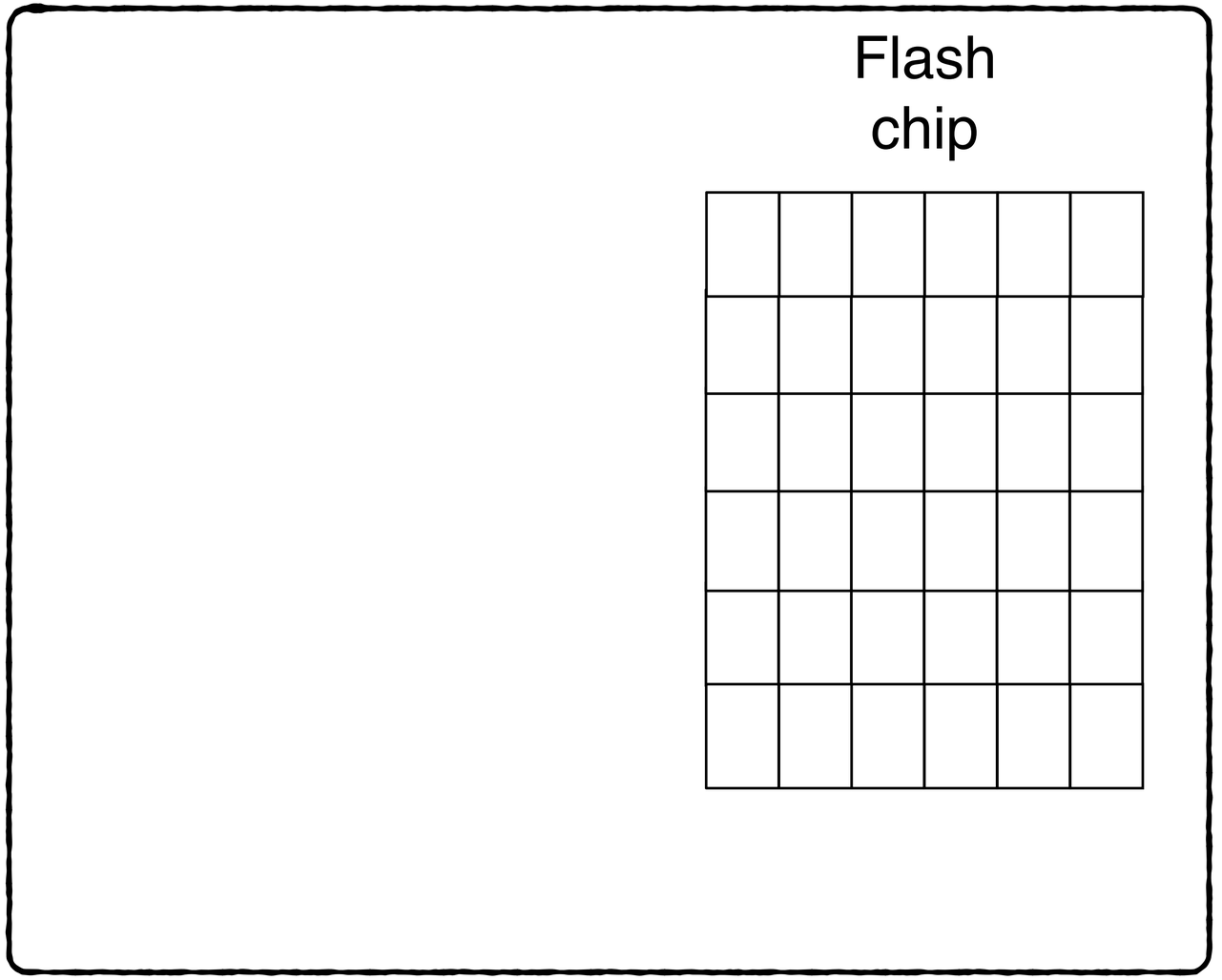


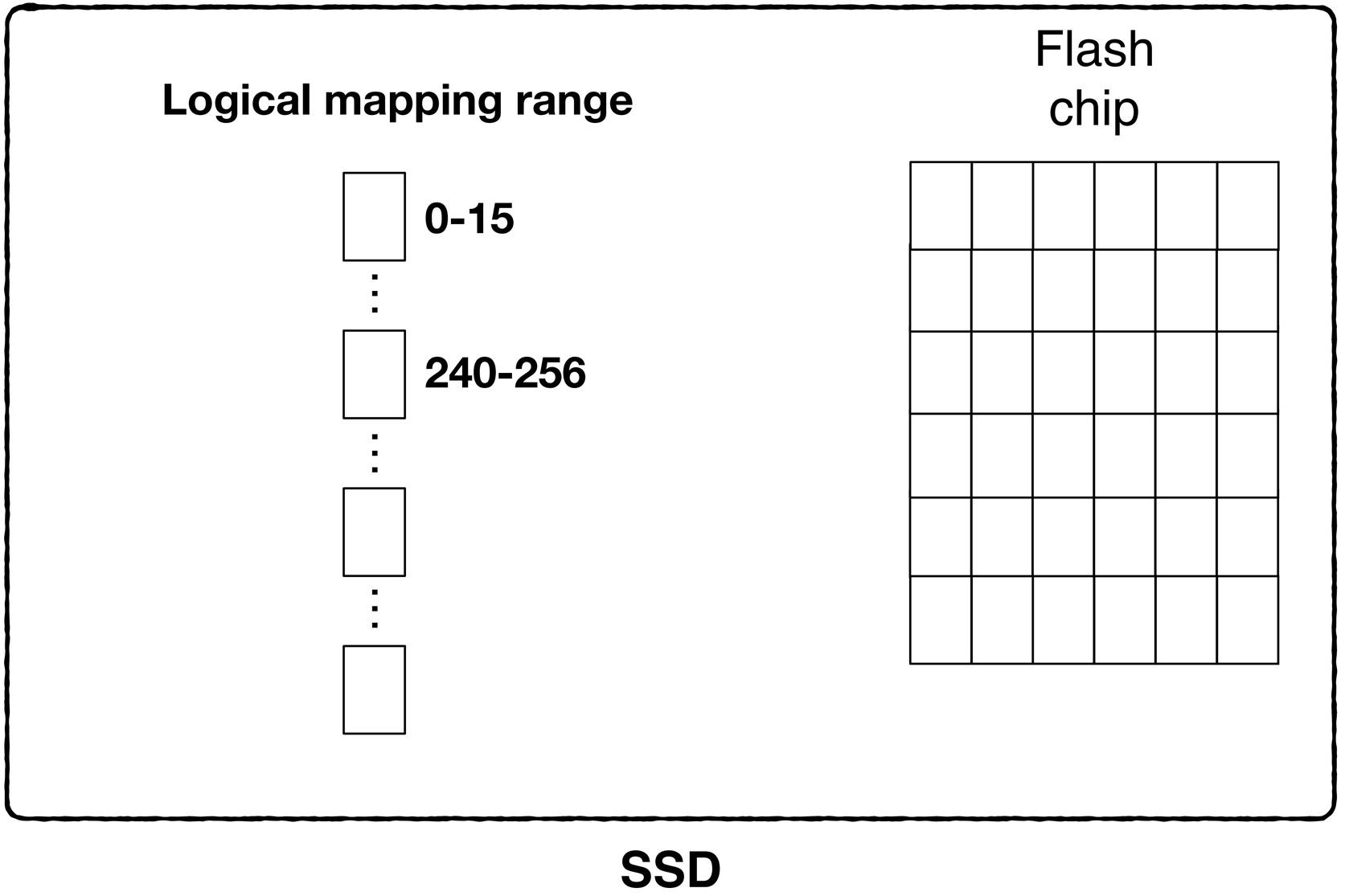
SSD

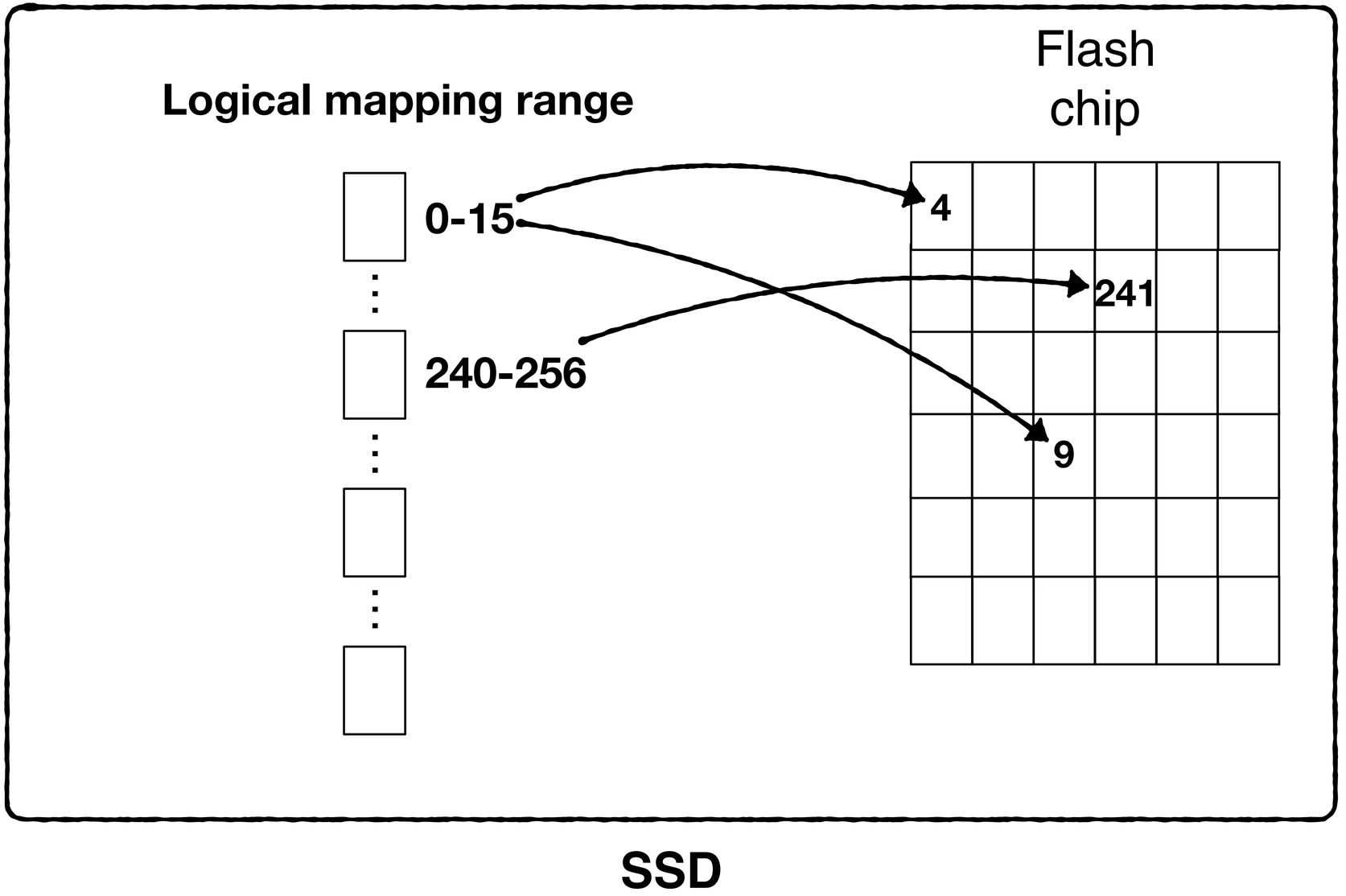
Alternative 2: Store Page Mapping in Flash & Cache Parts

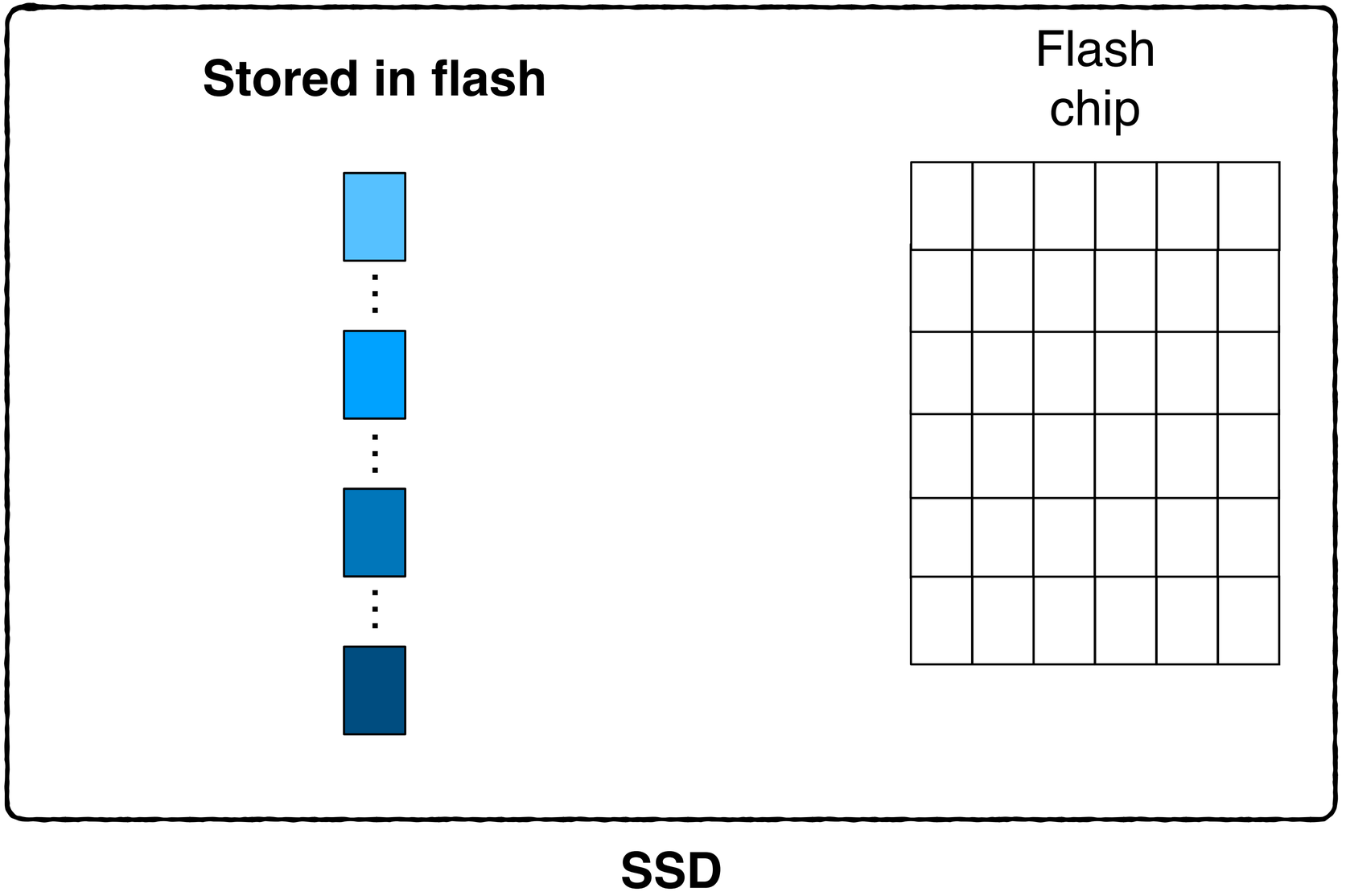
DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings

ACM SIGPLAN Notices 2009

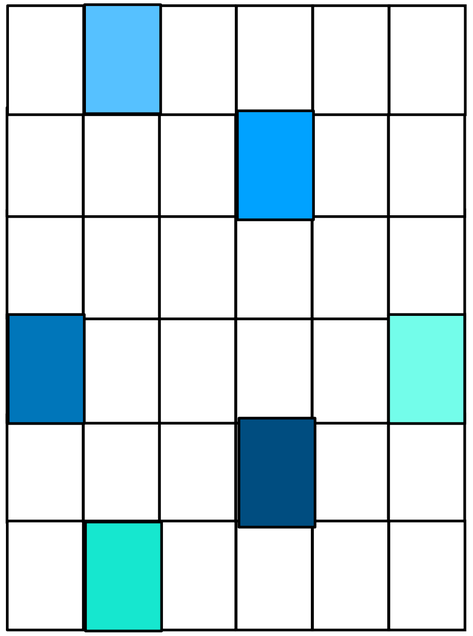



Logical mapping range
0-15
240-256
Flash
chip
SSD

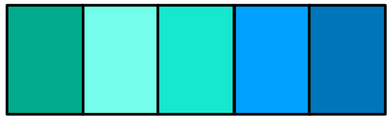

Logical mapping range
Flash chip
0-15
240-256
4
241
9
SSD


Stored in flash
Flash chip
SSD

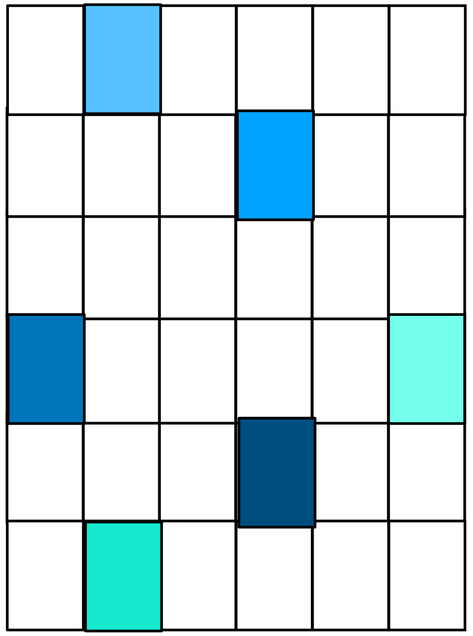
Flash
chip



Buffer pool
frequently accessed mapping pages
(LRU/Clock)

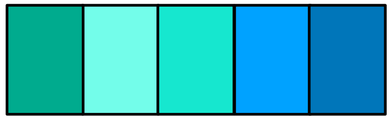


Flash
chip

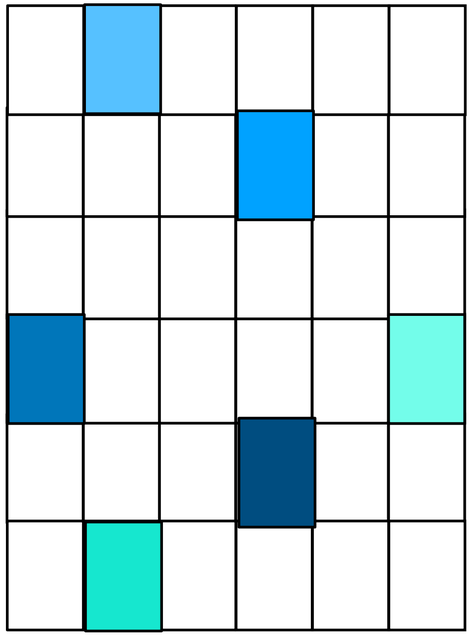


Downsides?

Buffer pool
(LRU/Clock)



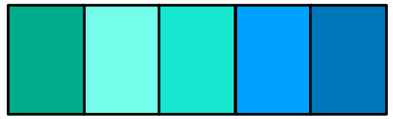
Flash
chip



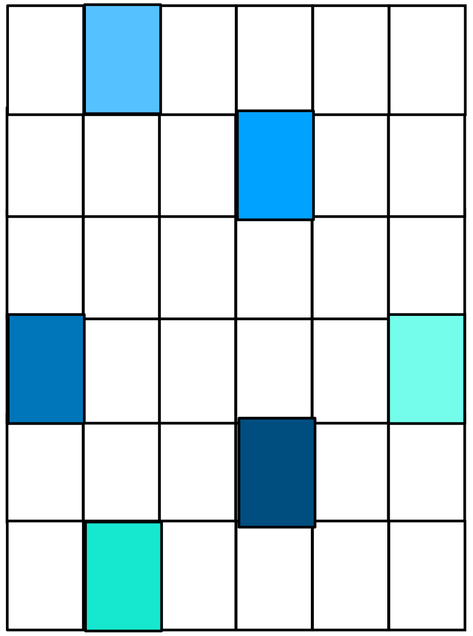
Downsides?

(1) read costs 2 I/Os if mapping page isn't cached

Buffer pool
(LRU/Clock)



Flash
chip

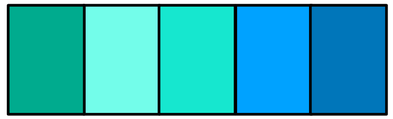


Downsides?

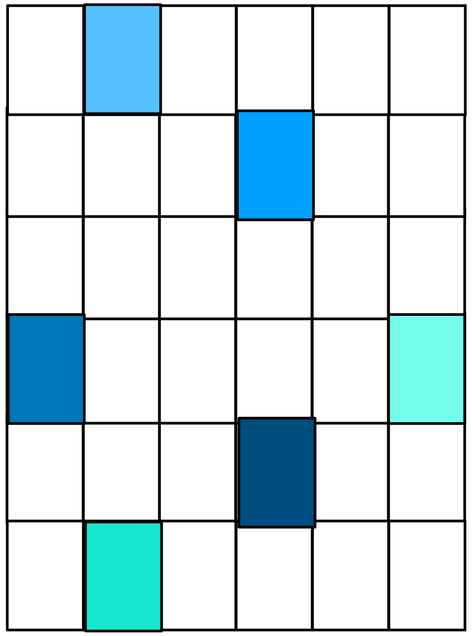
(1) read costs 2 I/Os if mapping page isn't cached

(2) writes may cost more due to buffer pool evictions

Buffer pool
(LRU/Clock)



Flash
chip



Downsides?

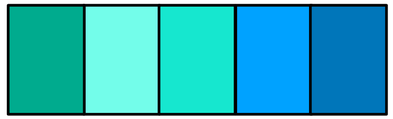
(1) read costs 2 I/Os if mapping page isn't cached

(2) writes may cost more due to buffer pool evictions

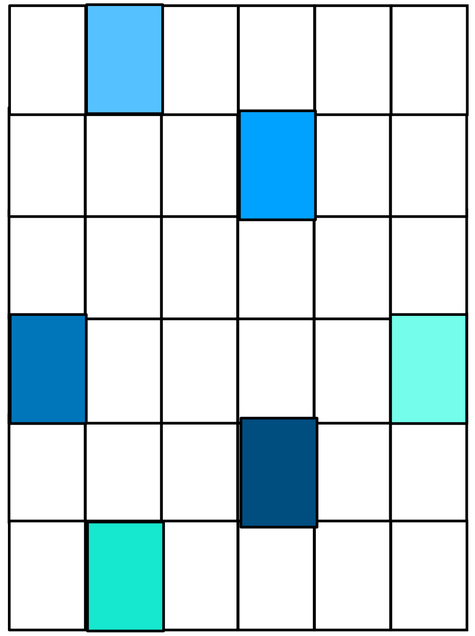
Sequential writes

Random I/Os

Buffer pool (LRU/Clock)



Flash chip



FTL implementation is opaque

FTL implementation is opaque

Hard to reason about performance outcomes given random SSD

FTL Downsides

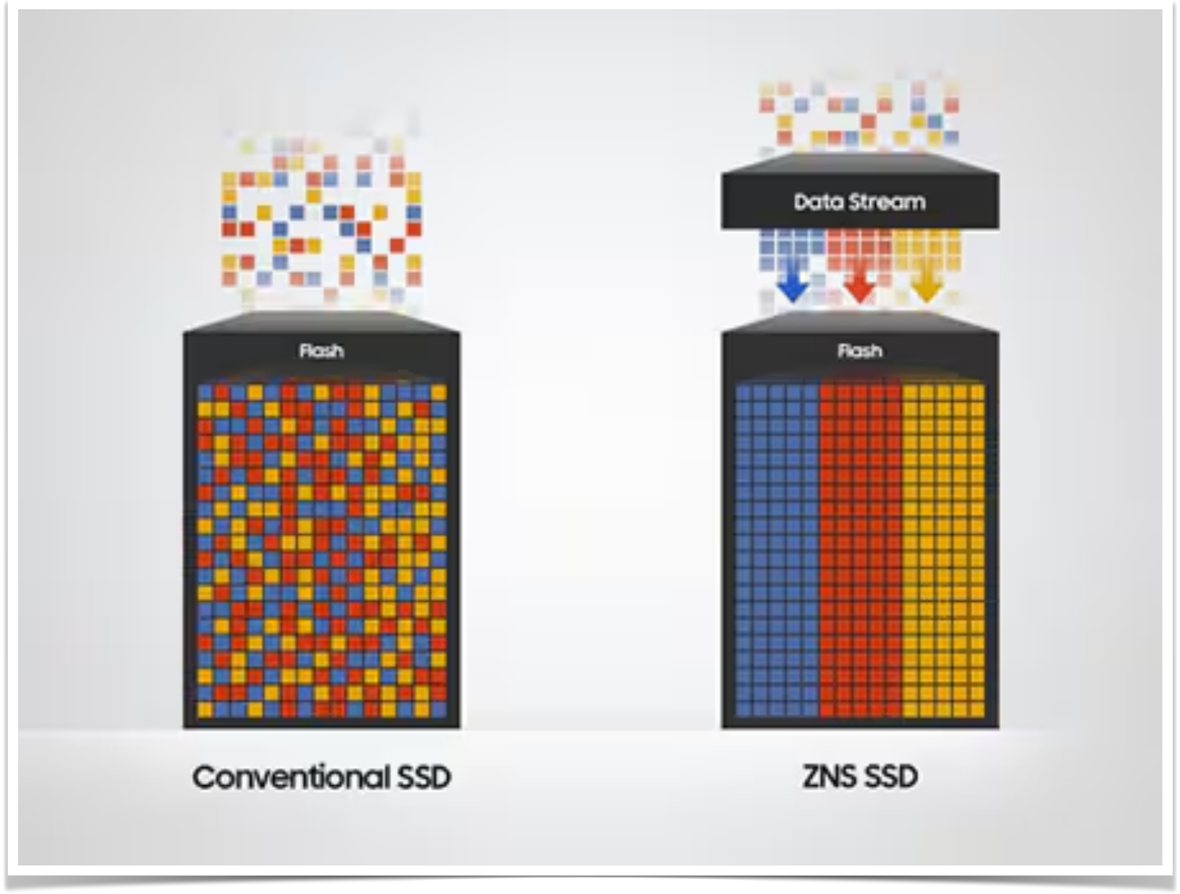
Memory overhead for mapping table

Storage space for over-provisioning

I/O & lifetime for garbage-collection

Zoned Namespaces (ZNS)

Zoned Namespaces (ZNS)



Zoned Namespaces (ZNS)

Zoned Namespaces (ZNS)

Applications write a zone sequentially

& later deletes it

Zoned Namespaces (ZNS)

Applications write a zone sequentially

& later deletes it

SSD performs no garbage-collection, mapping is small, and no over-provisioning

Zoned Namespaces (ZNS)

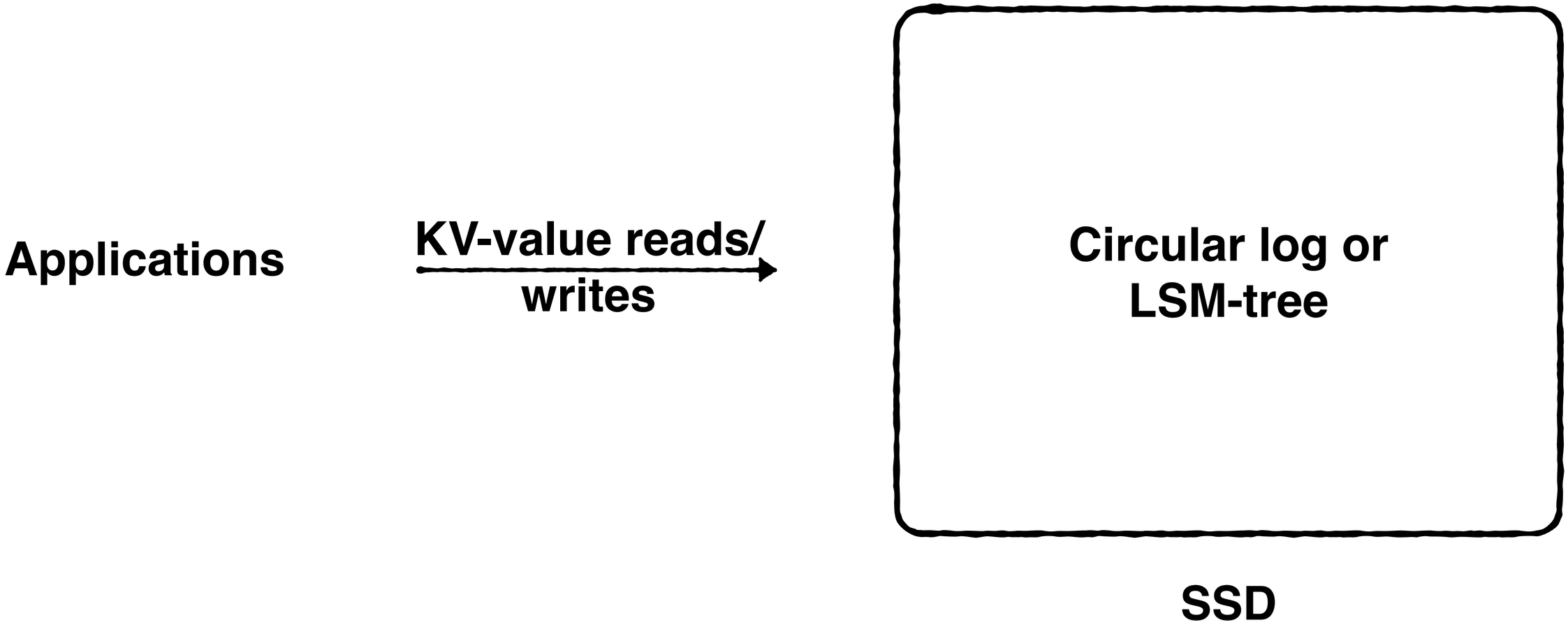
Applications write a zone sequentially

& later deletes it

SSD performs no garbage-collection, mapping is small, and no over-provisioning

Restrictive & application has to essentially implement FTL

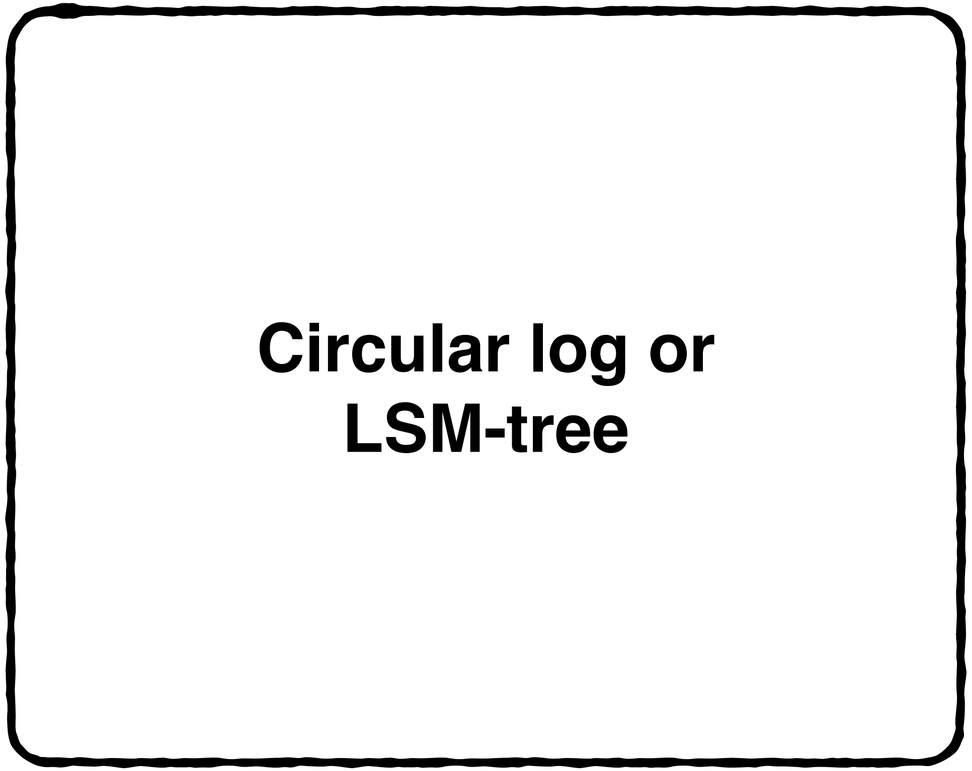
Key-Value SSDs



Key-Value SSDs

No data movement or CPU for compaction or circular log garbage-collection

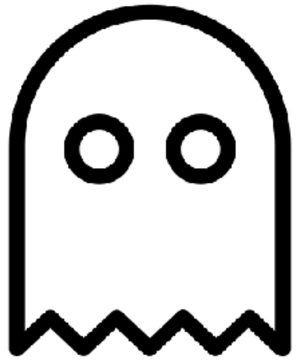
Reduces SSD over-provisioning



SSD

Spooky

Granulating LSM-Tree Compactions Correctly



Niv Dayan*
University of Toronto

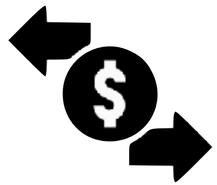
Tamar Weiss, Shmuel Dashevsky, Michael Pan,
Edward Bortnikov, Moshe Twitto *Pliops*

VLDB 2022

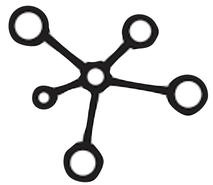
Spooky

Getting LSM-trees right on SSDs

OLTP

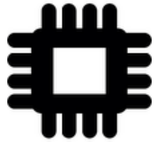


Graphs



Time series

Analytics



buffer

merge

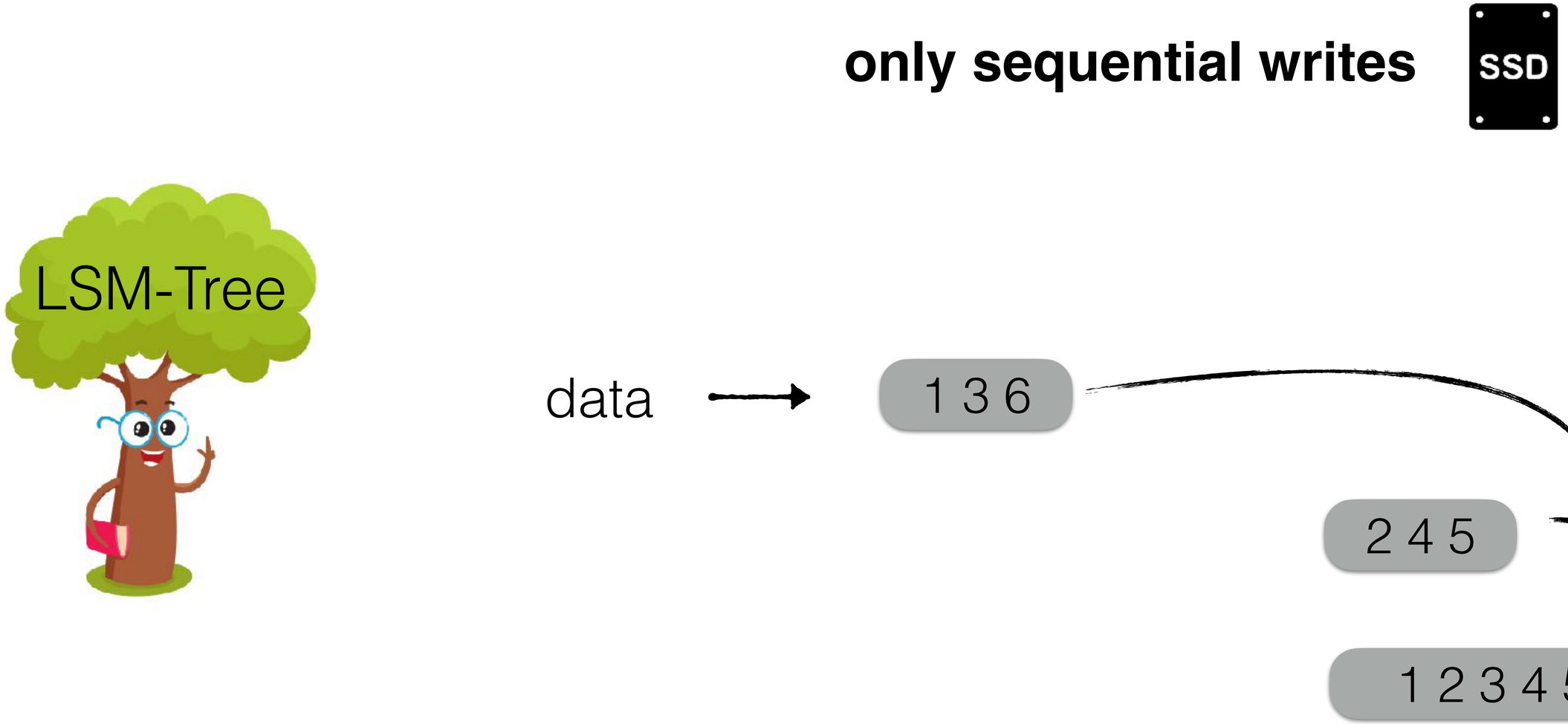
data

1 3 6

2 4 5

1 2 3 4 5 6

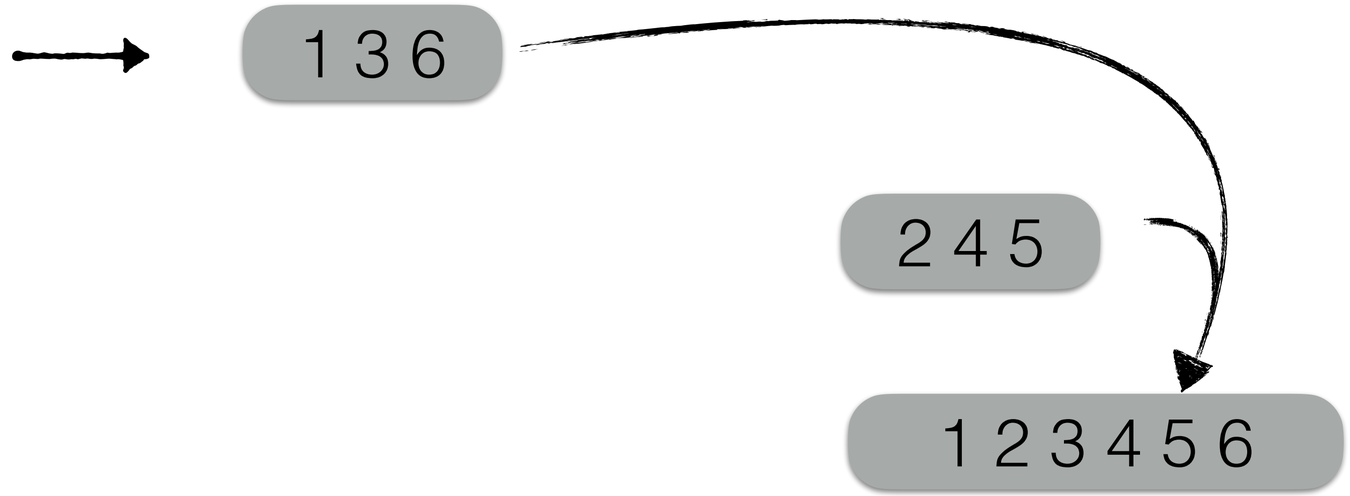
only sequential writes



only sequential writes

write-amplification

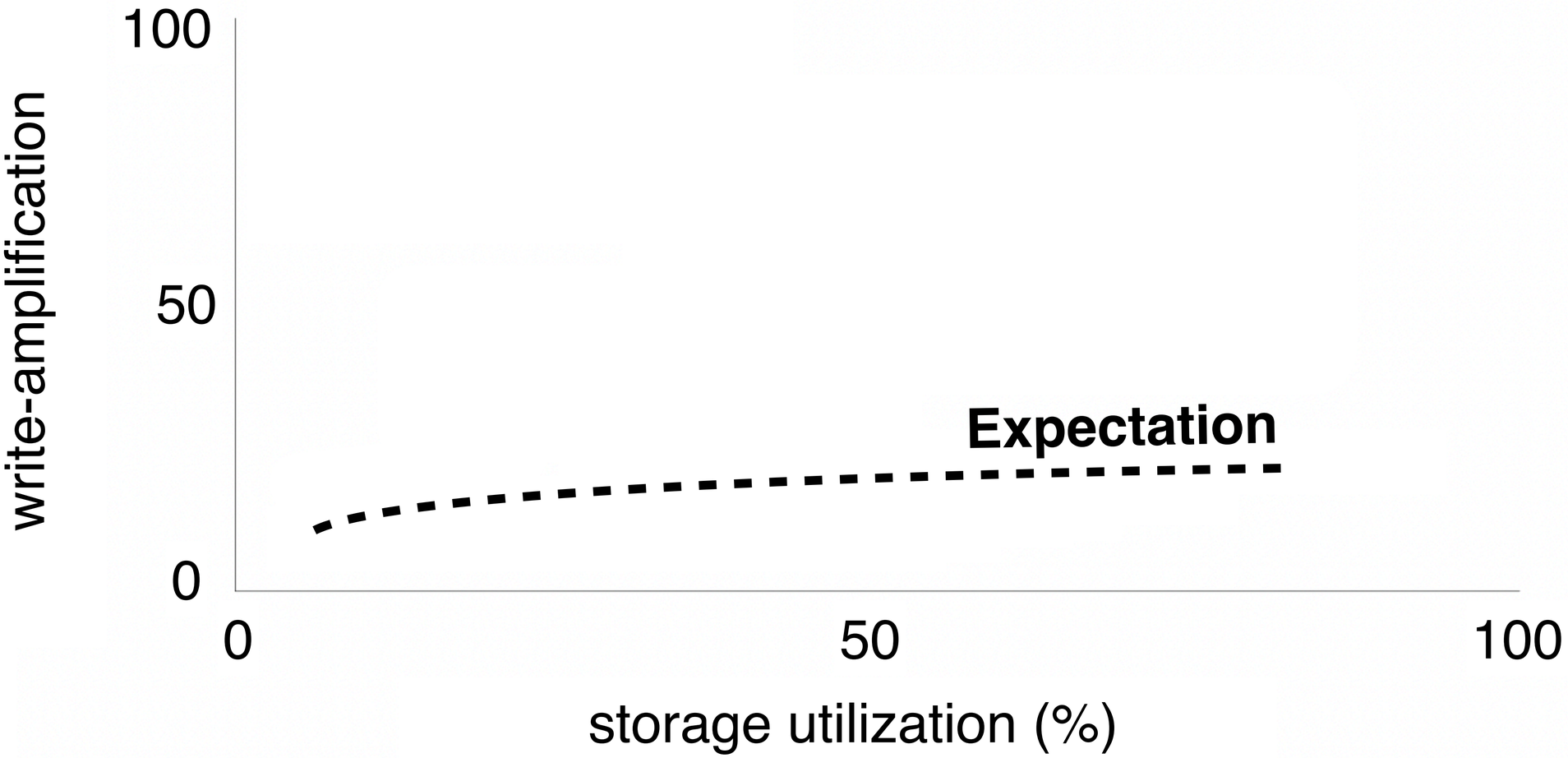
data

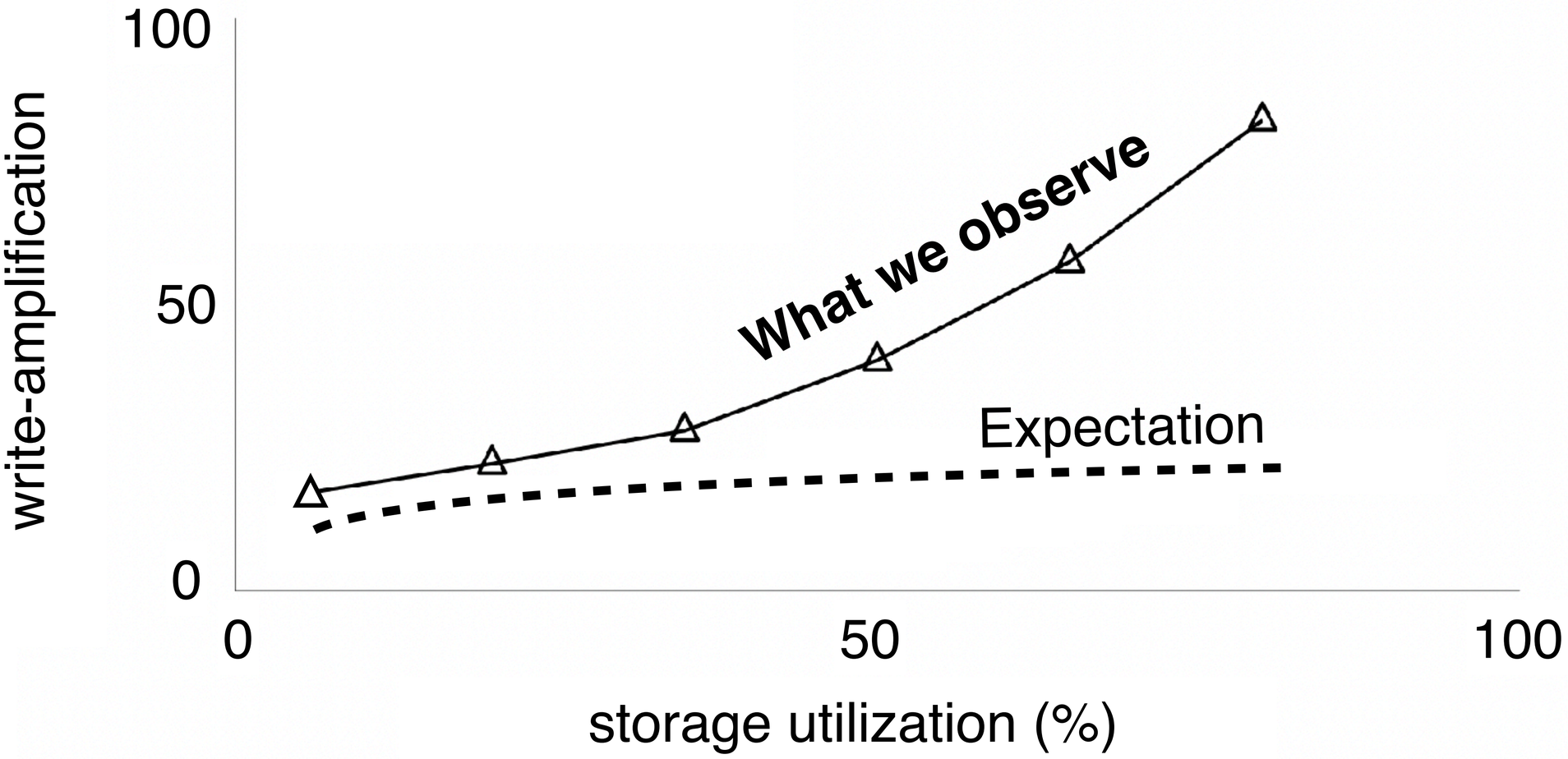


only sequential writes

write-amplification

O(Log N)


100
50
0
write-amplification
Expectation
0
50
100
storage utilization (%)


100
50
0
write-amplification
What we observe
Expectation
0
50
100
storage utilization (%)

LSM-Tree

key-value pairs

↓

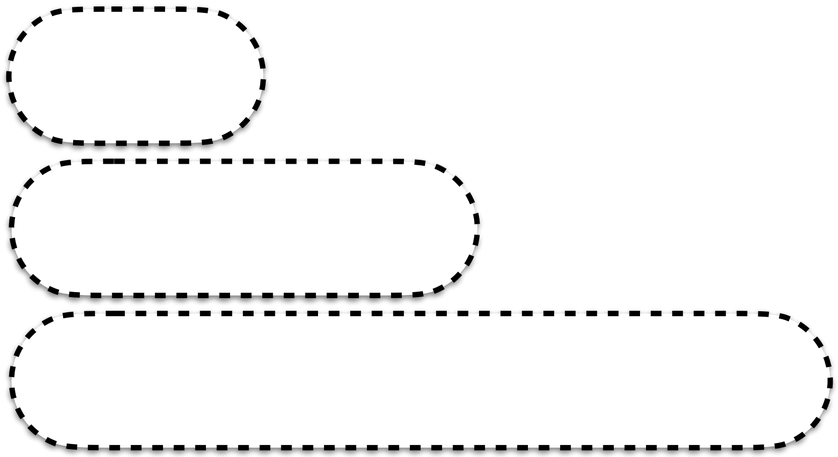
buffer

LSM-Tree

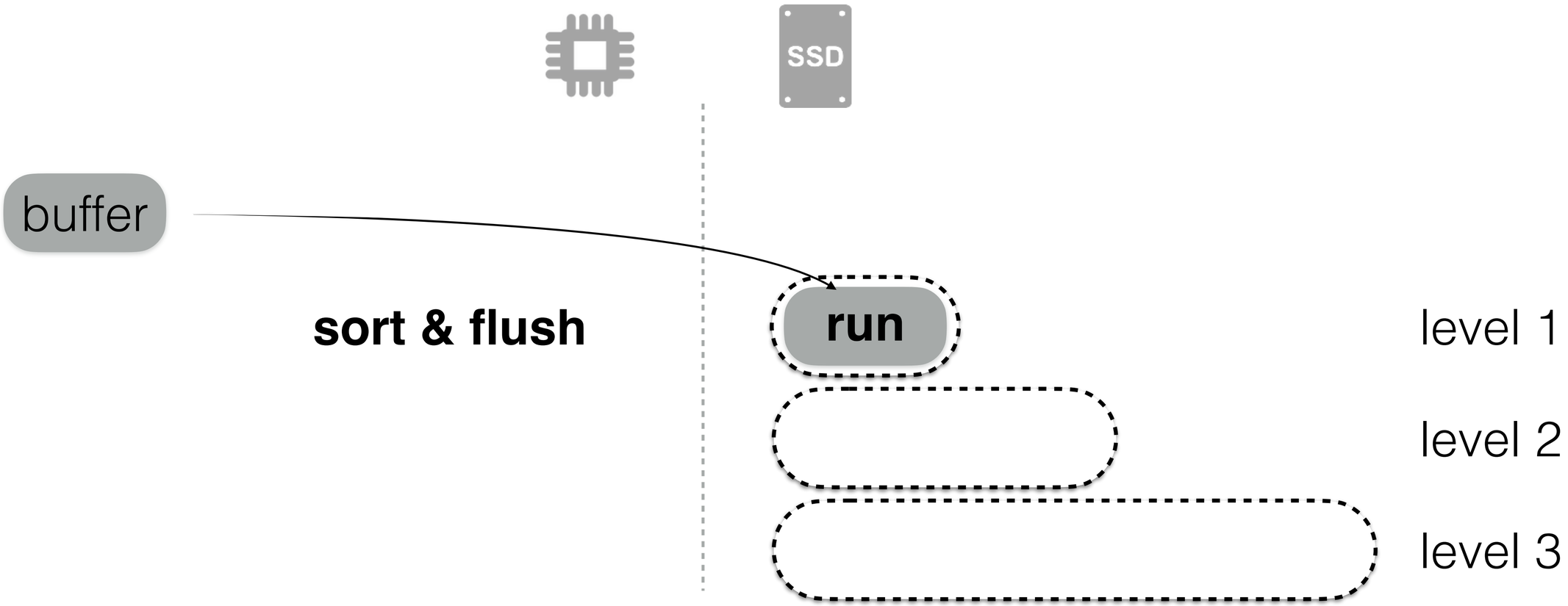
key-value pairs

↓

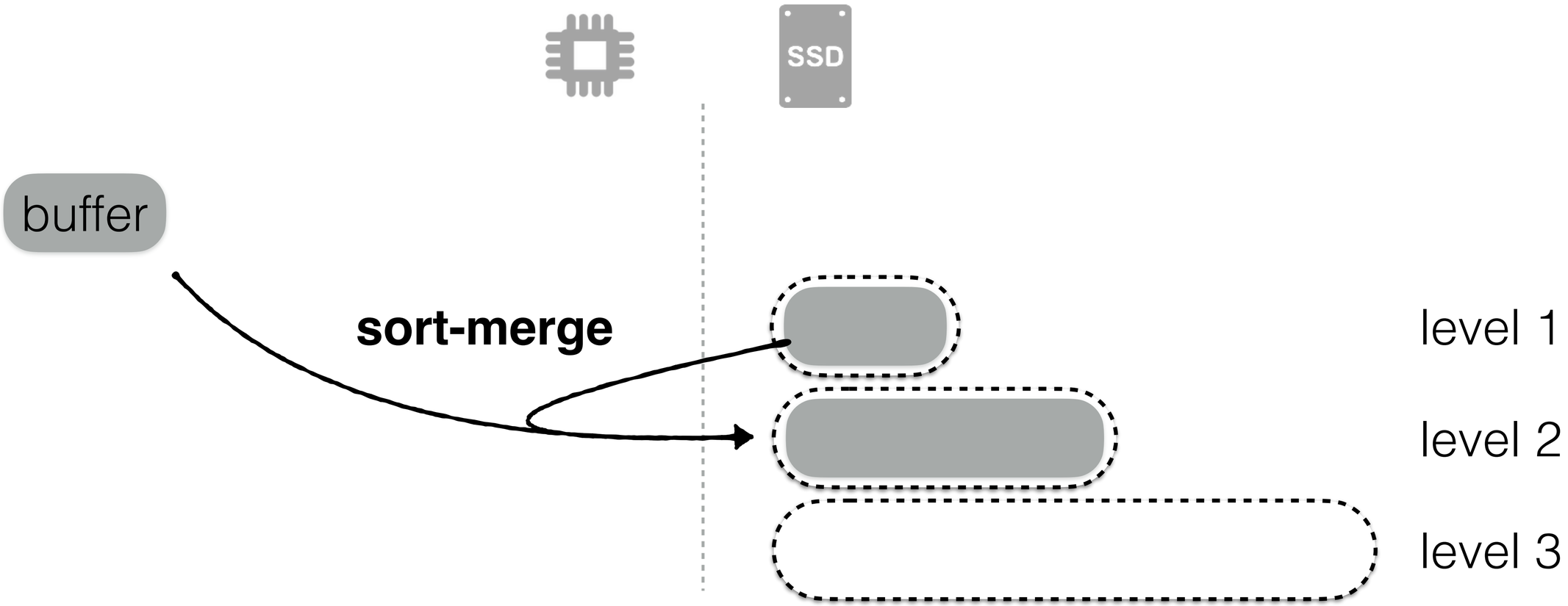
buffer

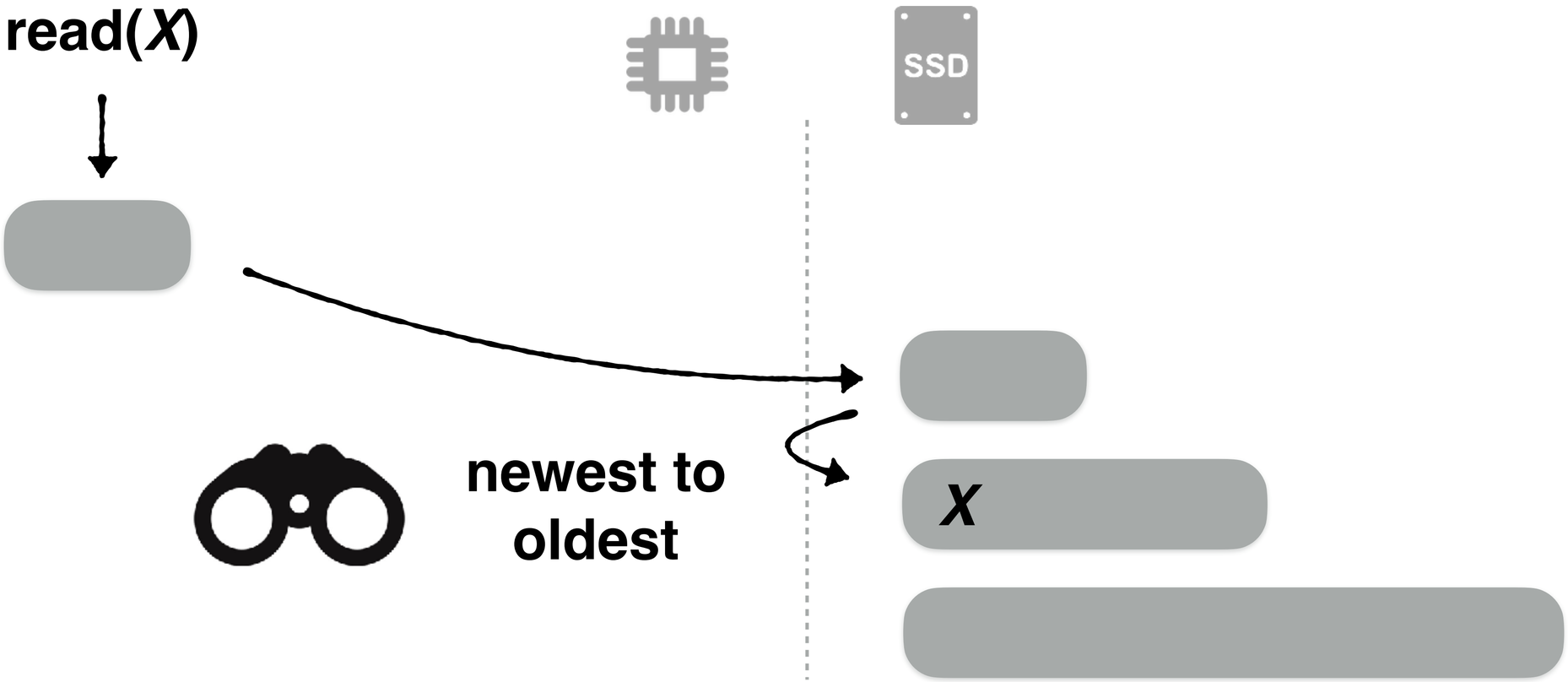


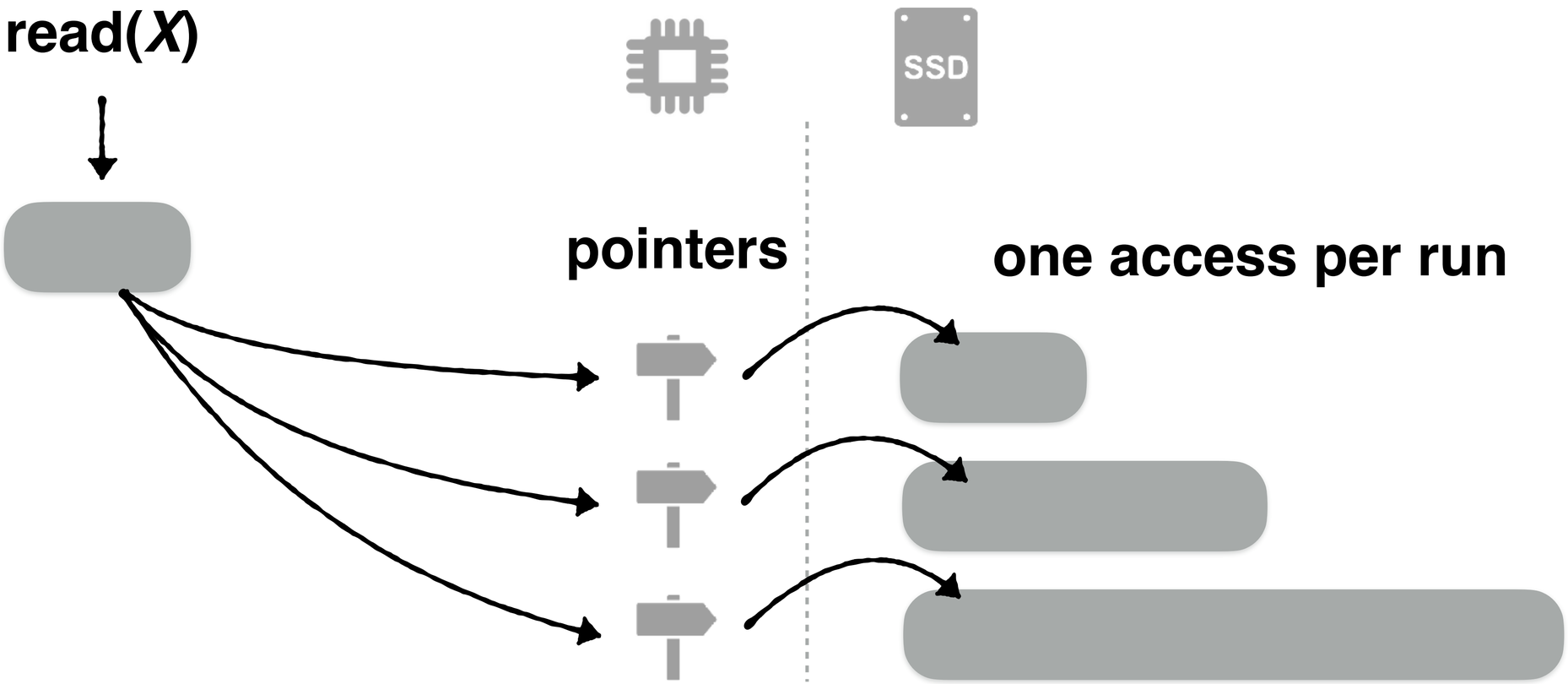
LSM-Tree

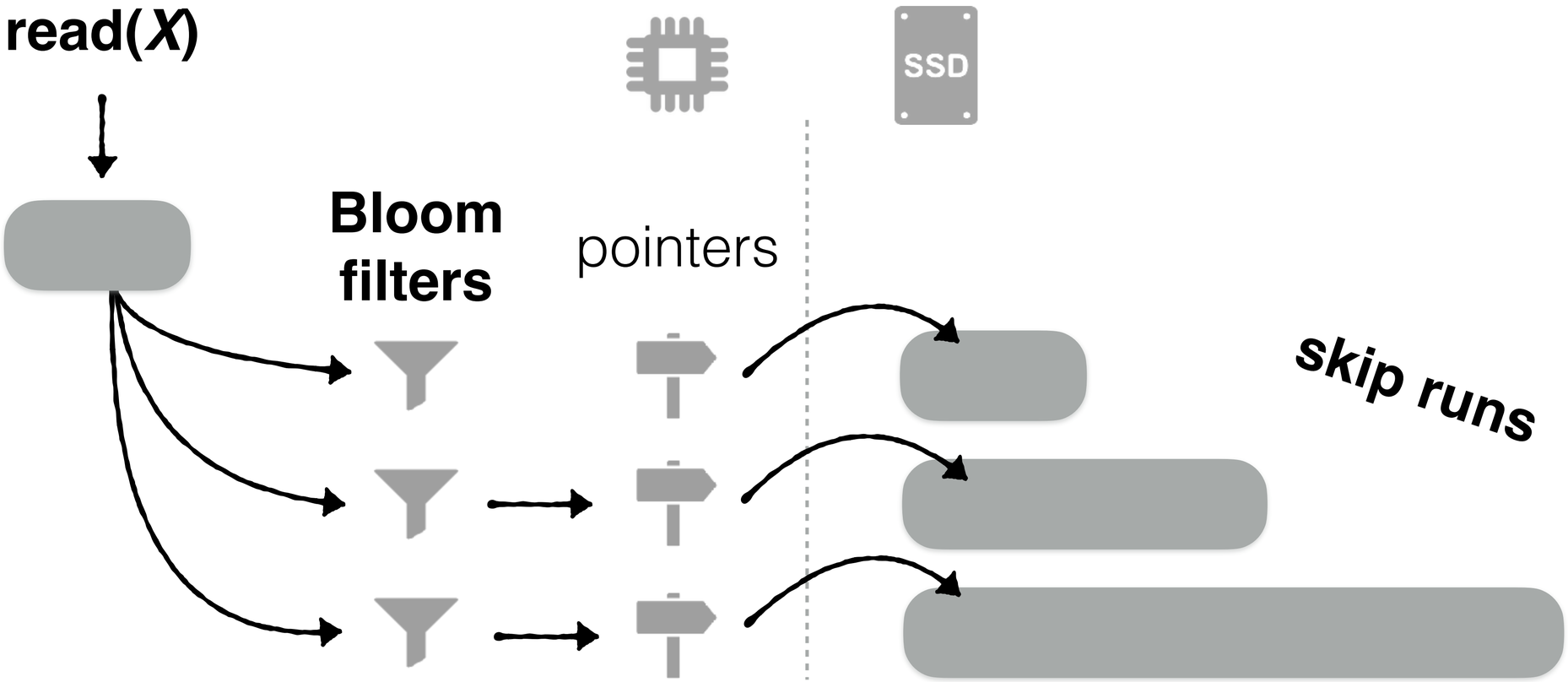


LSM-Tree

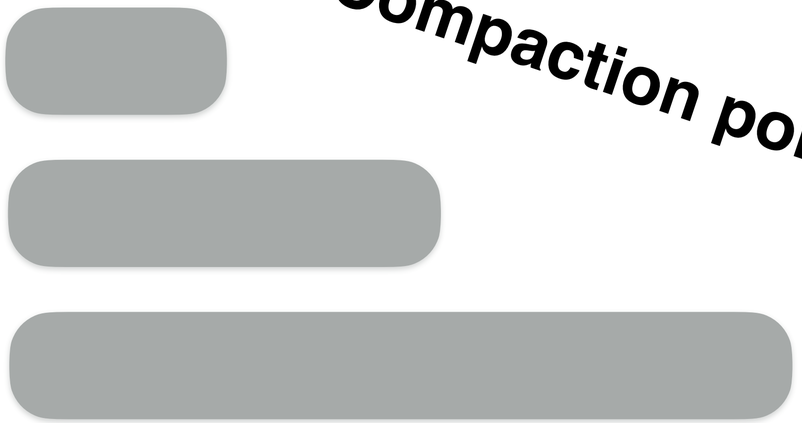



read(X)
SSD
newest to
oldest
X


read(X)
SSD
pointers
one access per run


read(X)
SSD
Bloom filters
pointers
skip runs


SSD


Compaction policy

Compaction policy

Write-optimized

Eagerness

Read-optimized

Compaction policy

Write-optimized

Eagerness

Read-optimized

3 1 **Tiering**

2, 7 1, 5

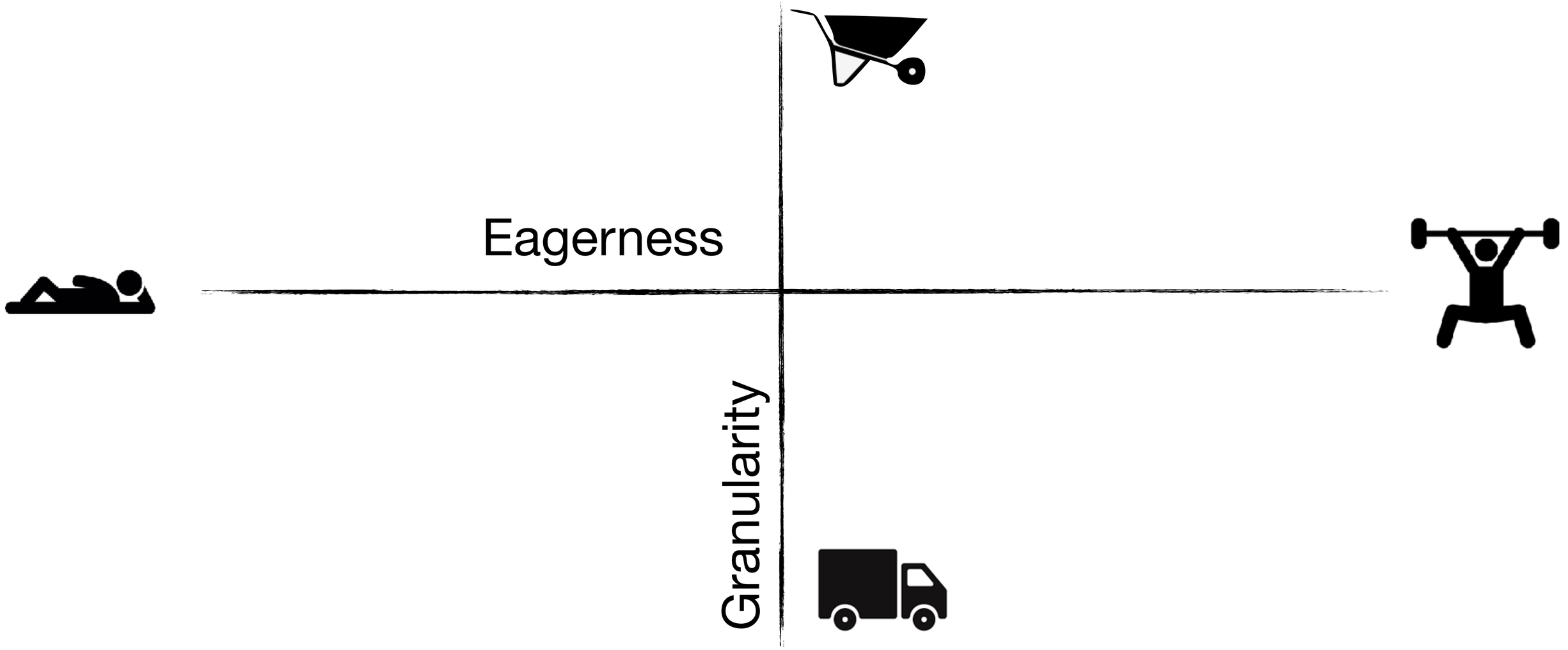
1, 3, 6, 7 2, 3, 5, 9

2, 5 **Leveling**

1, 3, 6, 8

1, 2, 3, 5, 6, 8, 9

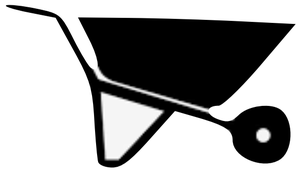
Compaction policy



Compaction Granularity

Full Merge

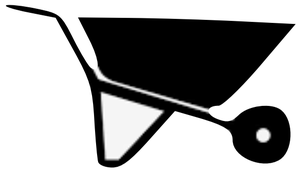
Partial Merge



Compaction Granularity

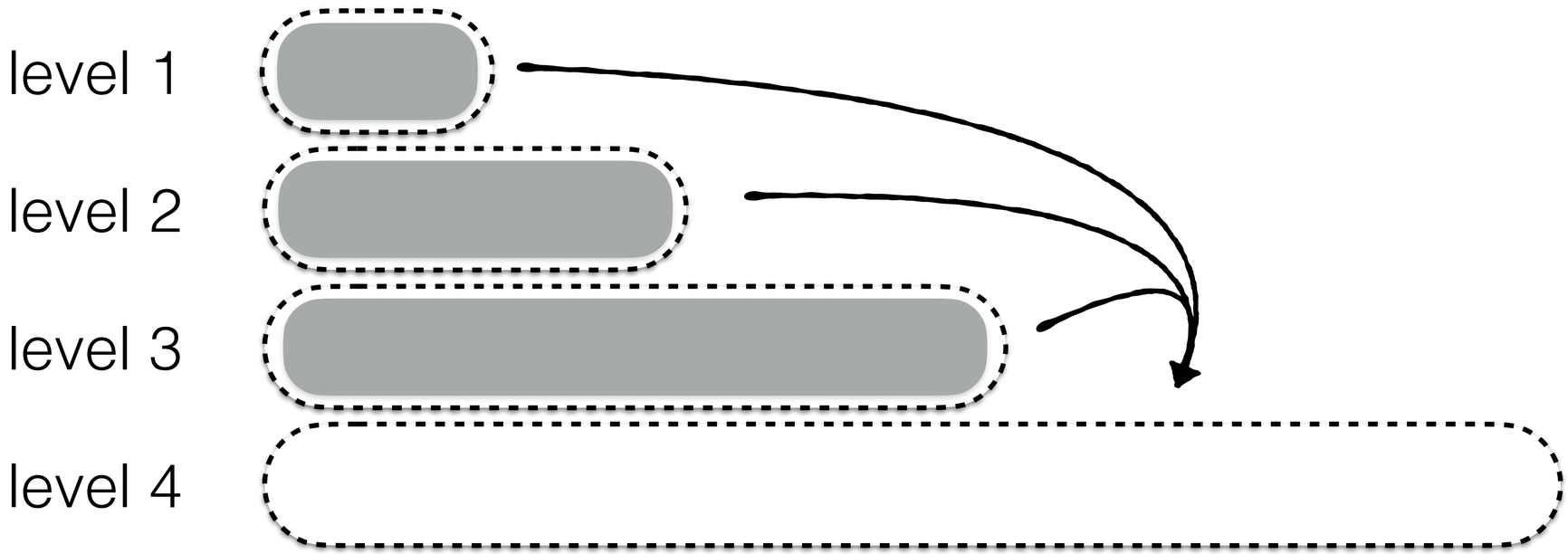
Full Merge

Partial Merge



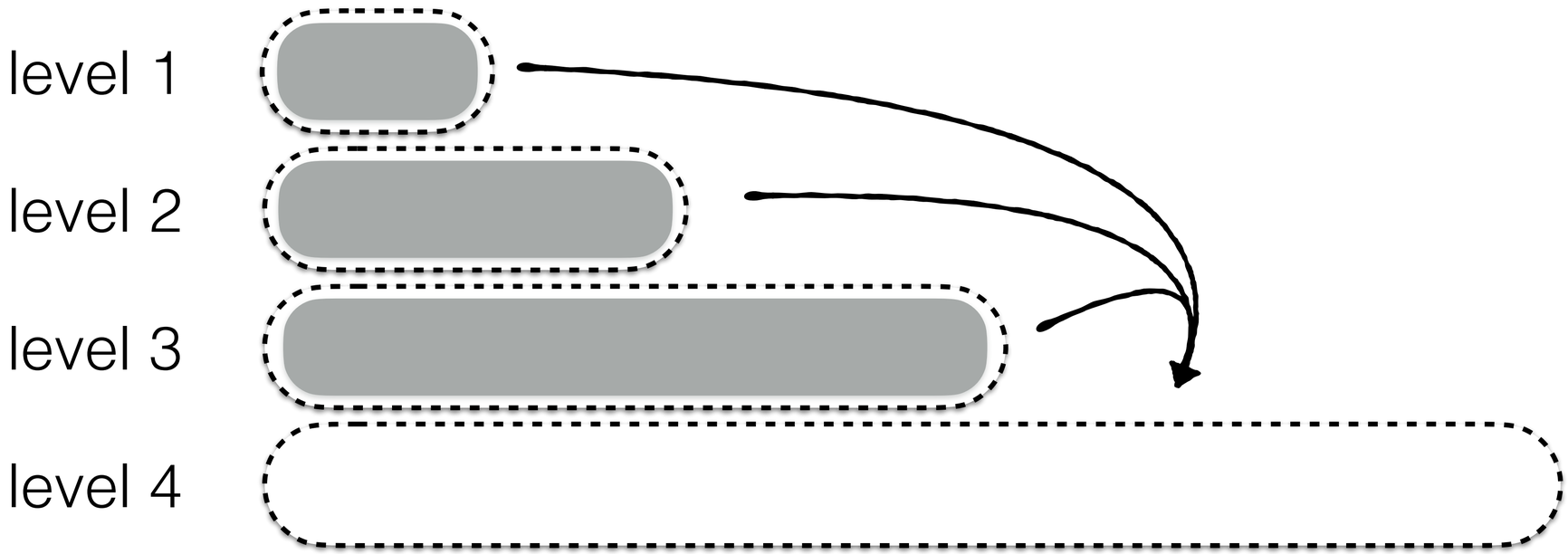
Full Merge

Merge consecutive full levels into first non-full level



Full Merge

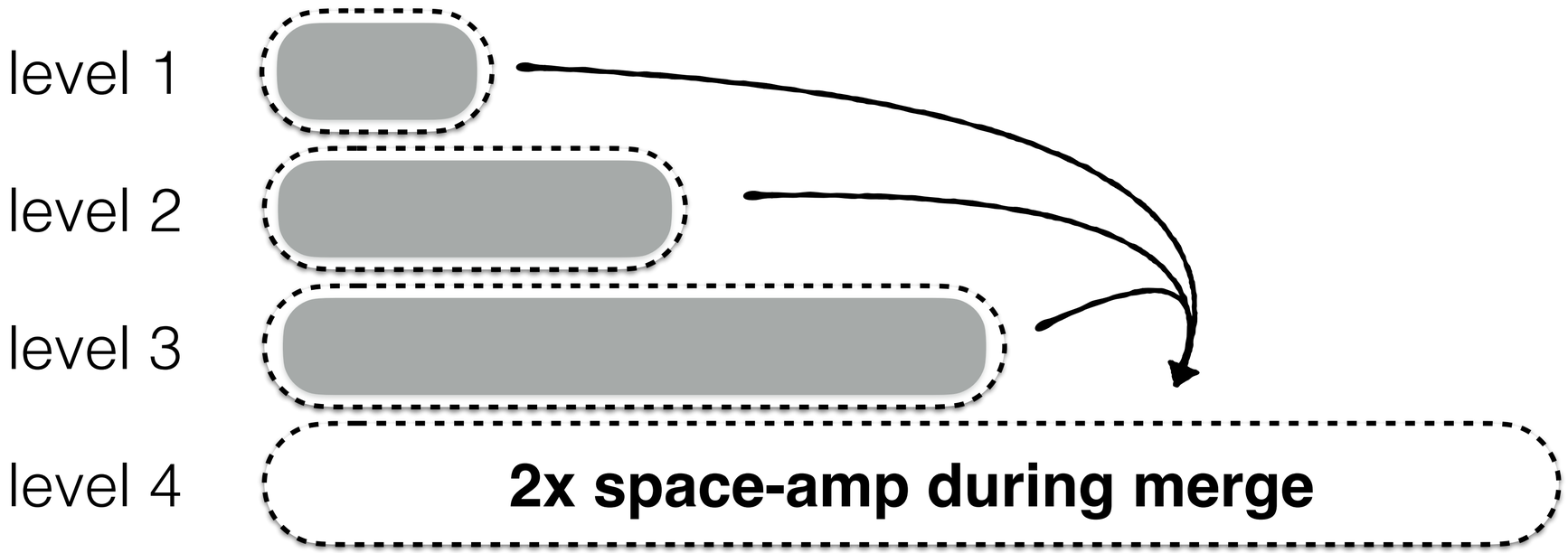
Merge consecutive full levels into first non-full level



Problem?

Full Merge

Merge consecutive full levels into first non-full level

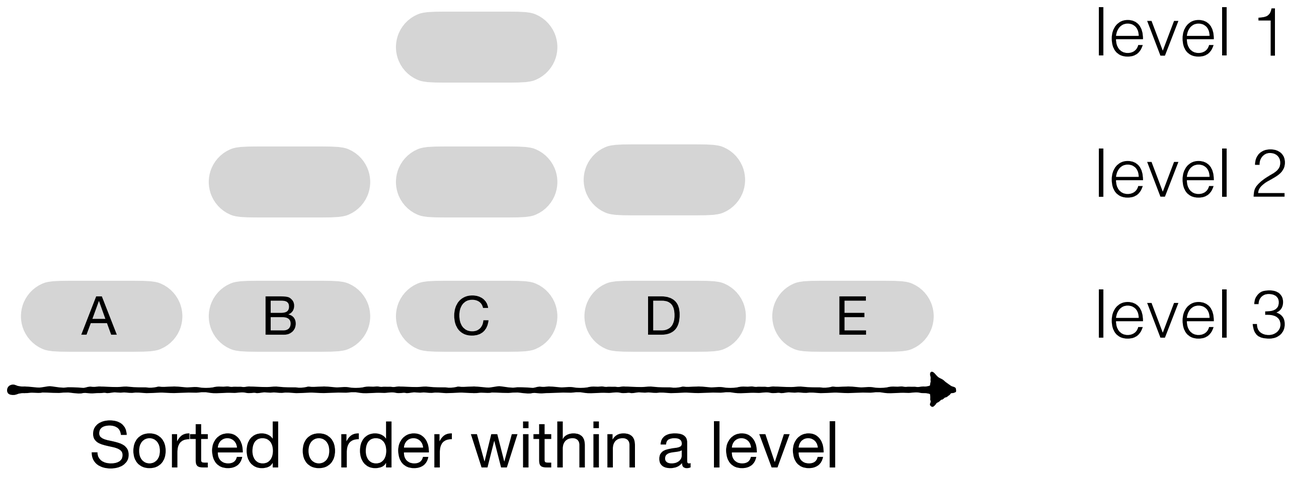


Full Merge

Partial Merge

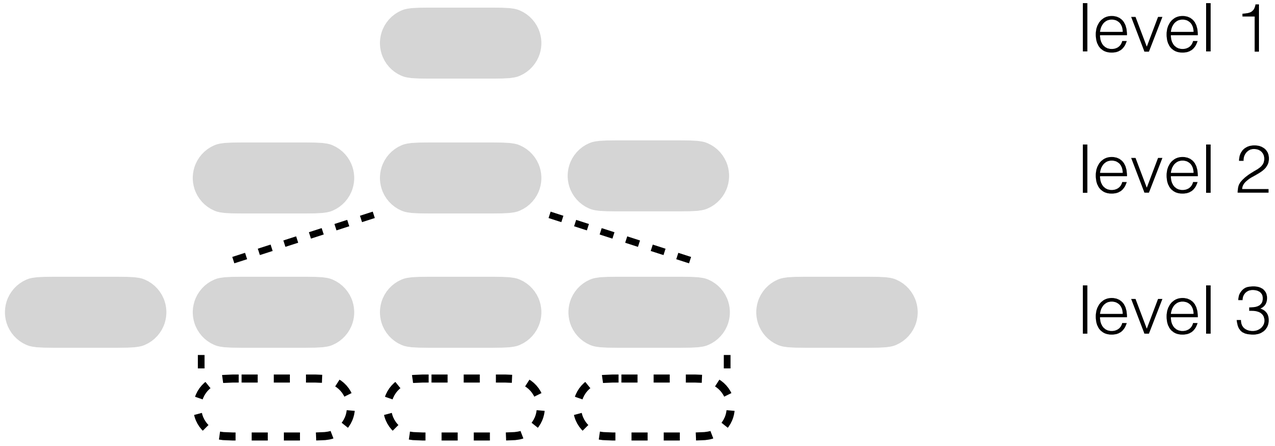
Partial Merge

1. split runs into many files (SSTs) in each level



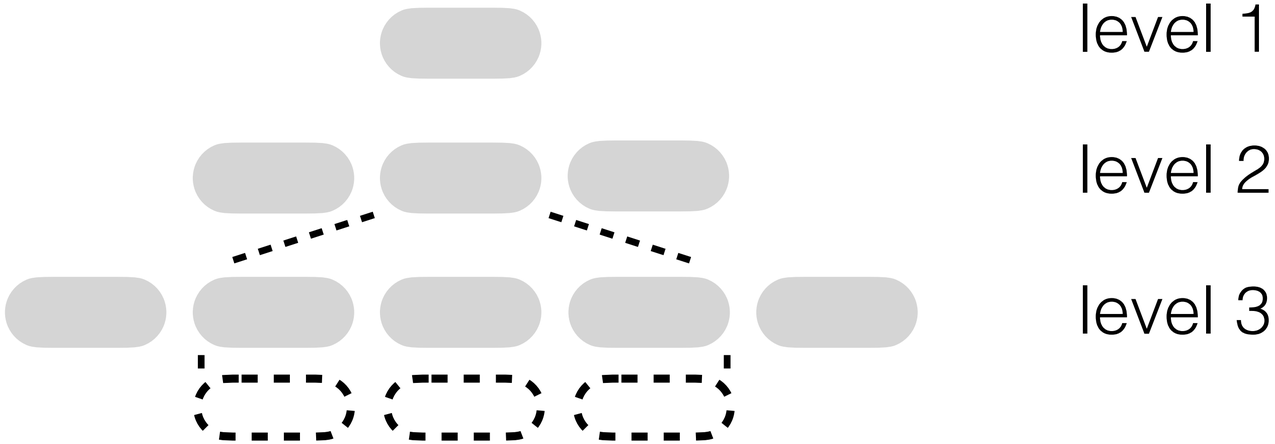
Partial Merge

1. split runs into many files (SSTs) in each level
2. When a level is full, pick SST with smallest intersection into next level



Partial Merge

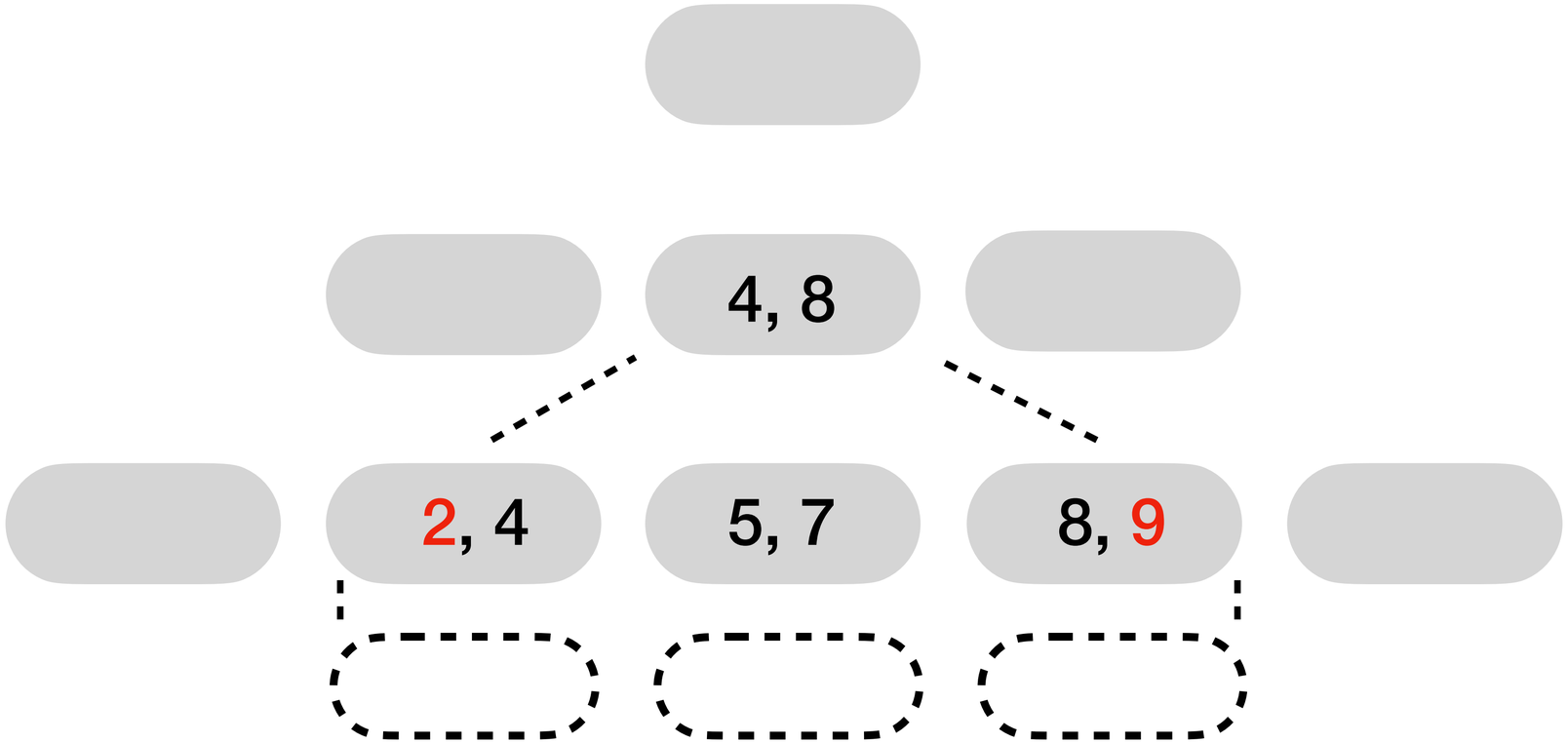
1. split runs into many files (SSTs) in each level
2. When a level is full, pick SST with smallest intersection into next level



Problem?

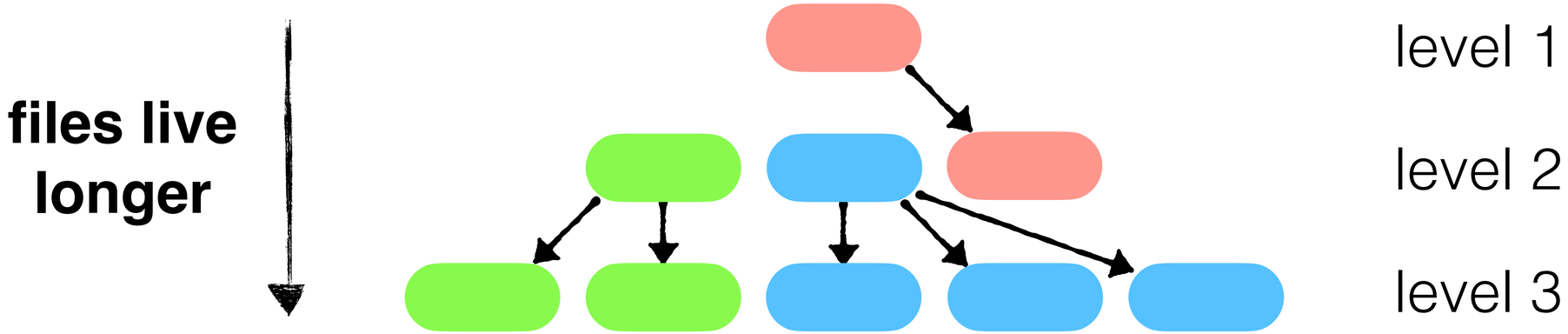
Partial Merge

Problem 1: non-intersecting entries increase write-amplification



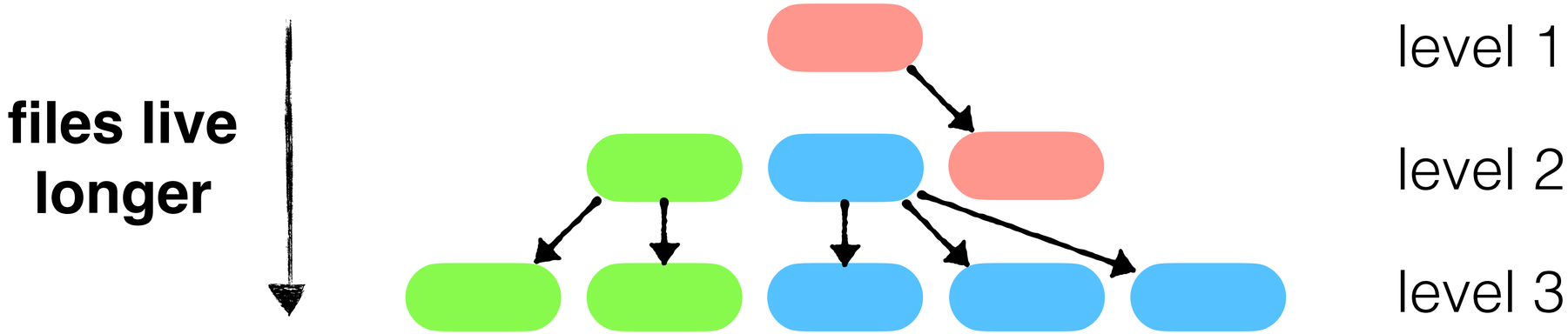
Partial Merge

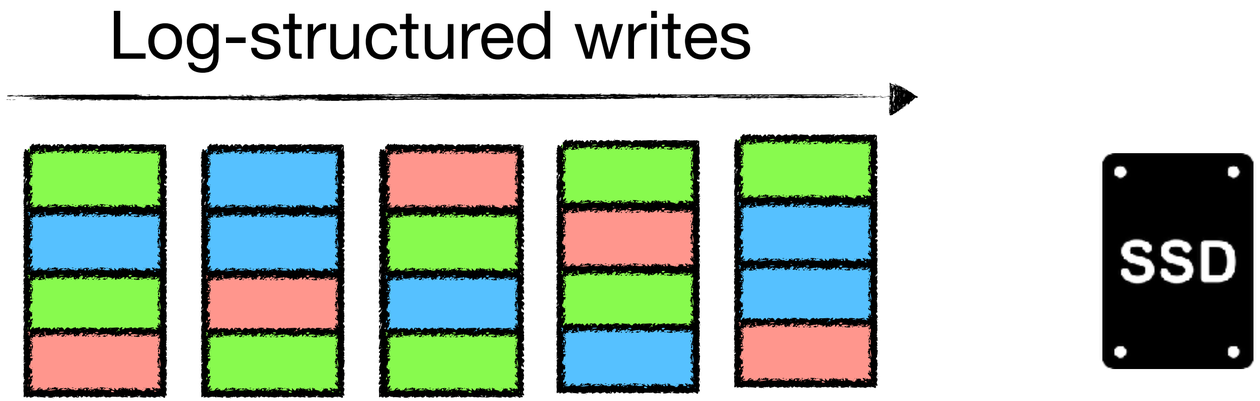
Problem 2: many small simultaneous compactions



Partial Merge

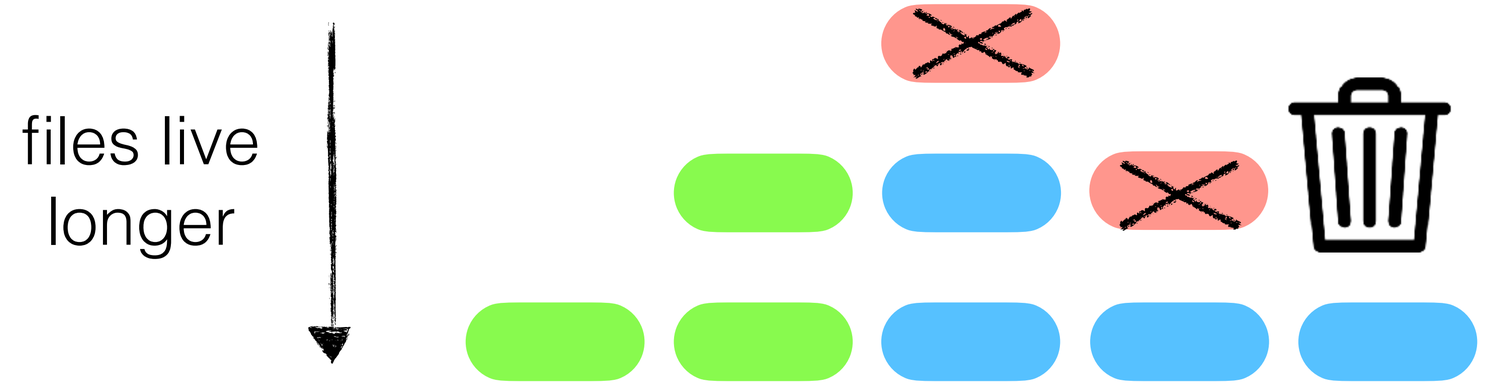
Problem 2: many small simultaneous compactions

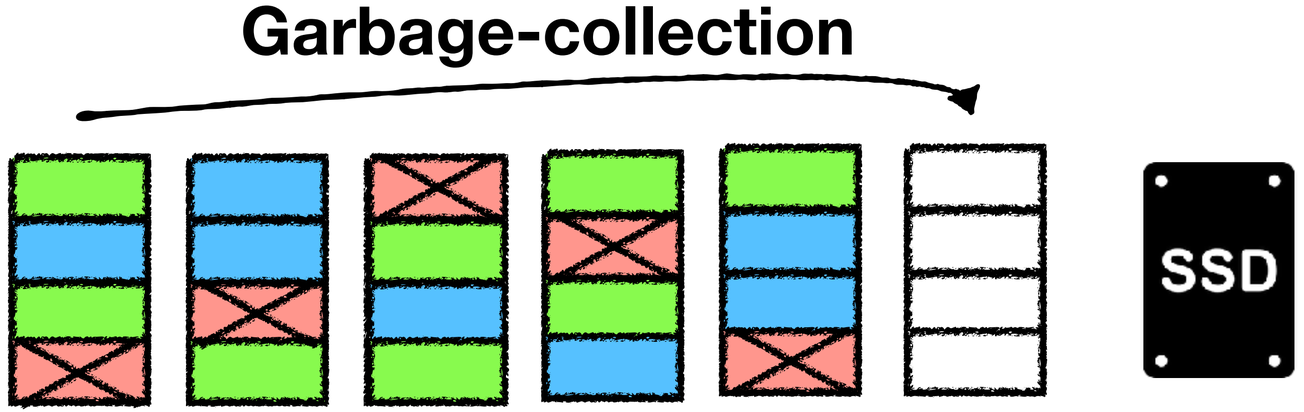




Partial Merge

Problem 2: many small simultaneous compactions

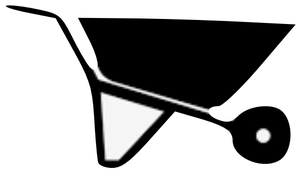




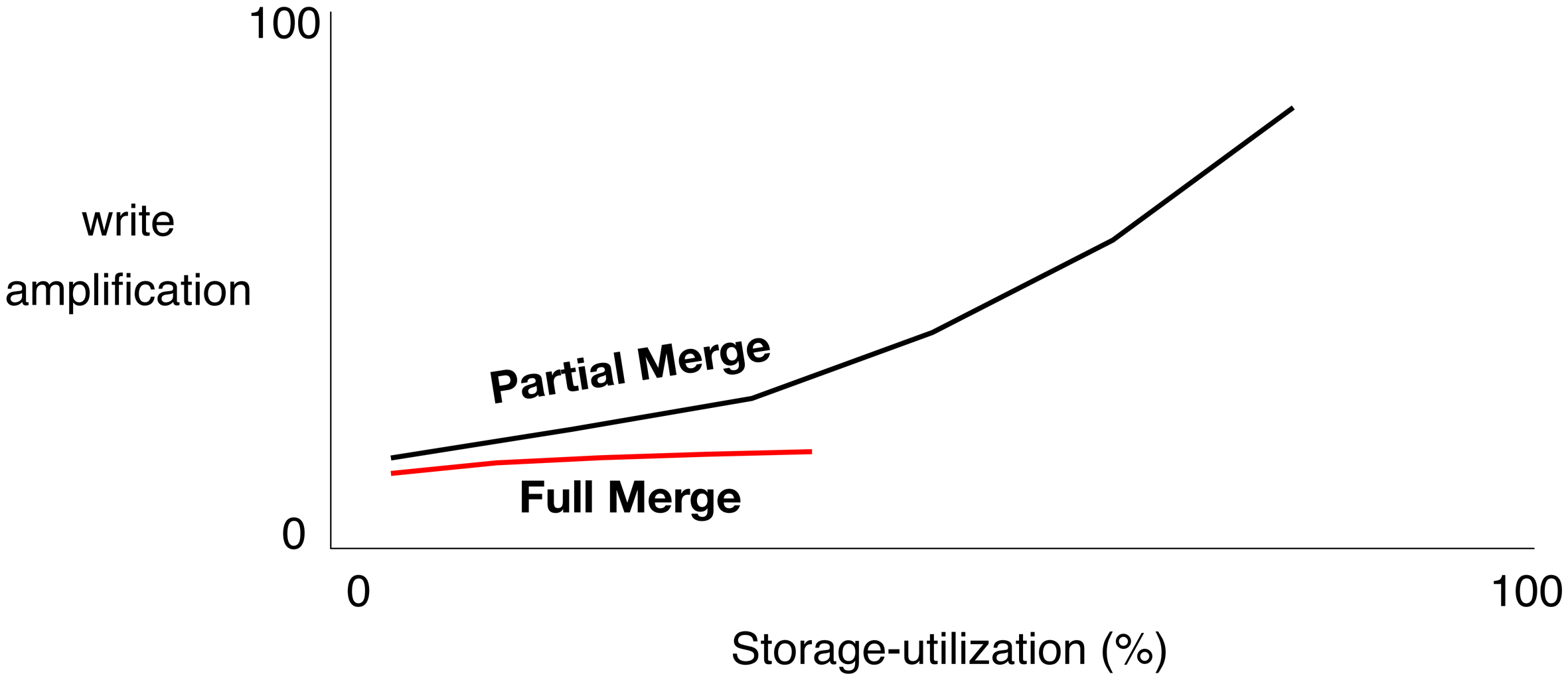
Full Merge

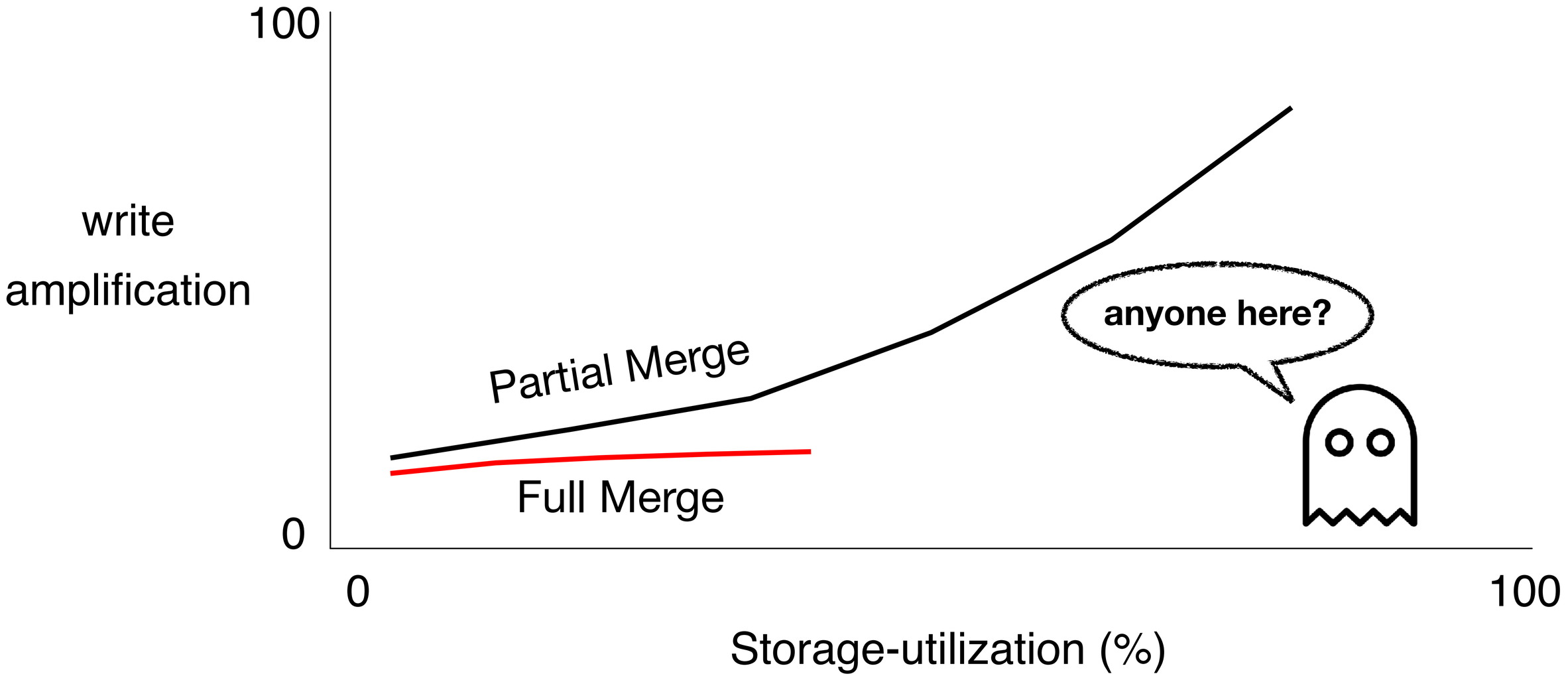
space amplification

Partial Merge



write amplification


100
write
amplification
Partial Merge
Full Merge
0
0
100
Storage-utilization (%)


100
write
amplification
Partial Merge
anyone here?
Full Merge
0
0
100
Storage-utilization (%)

Spooky

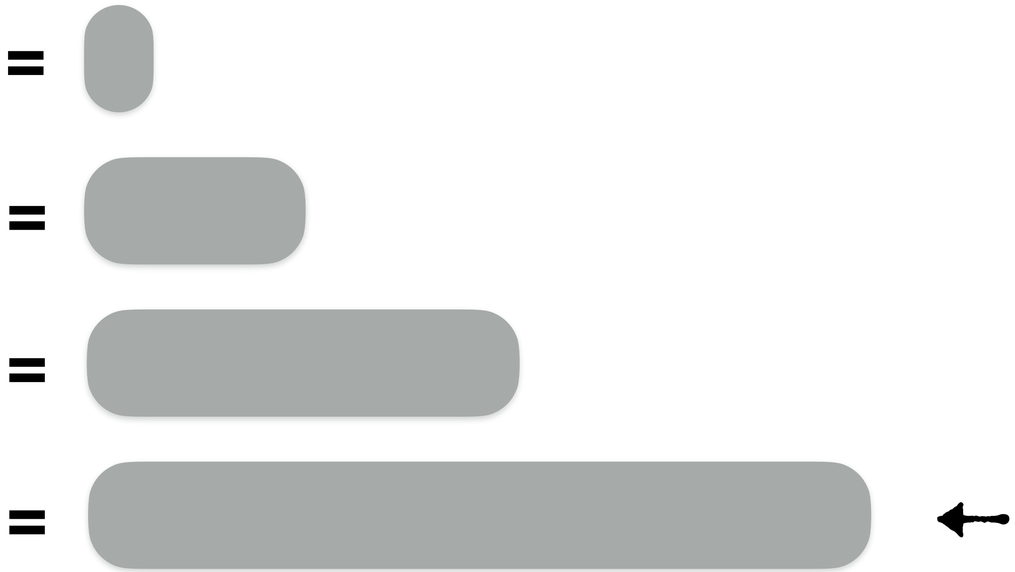
Spooky: **p**artitioned c**o**mpaction f**o**r **k**e**y**-value stores

Spooky's intuition

transient
space amplification

Spooky's intuition

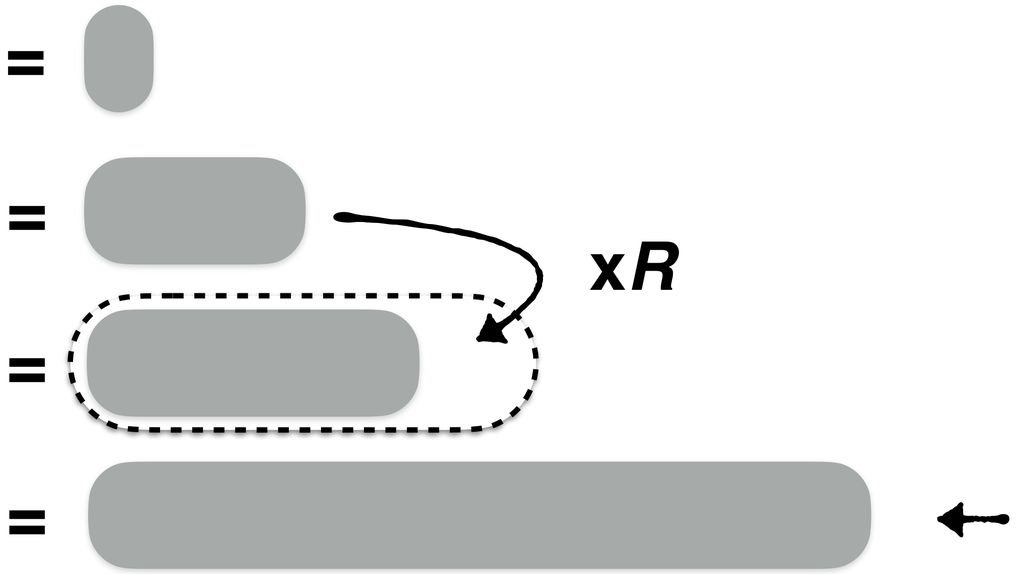
write amplification



transient
space amplification

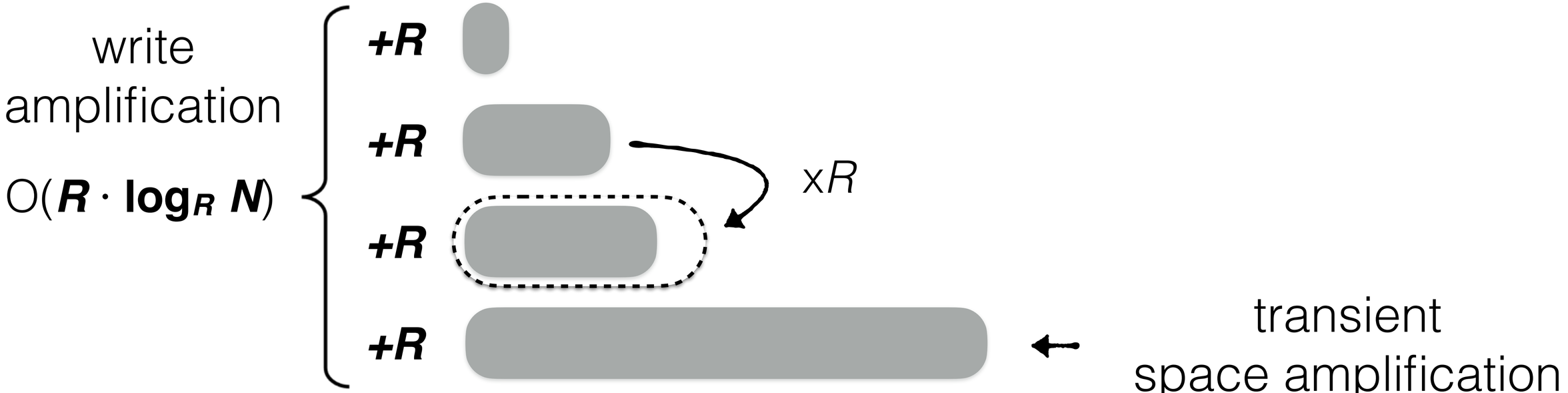
Spooky's intuition

write amplification



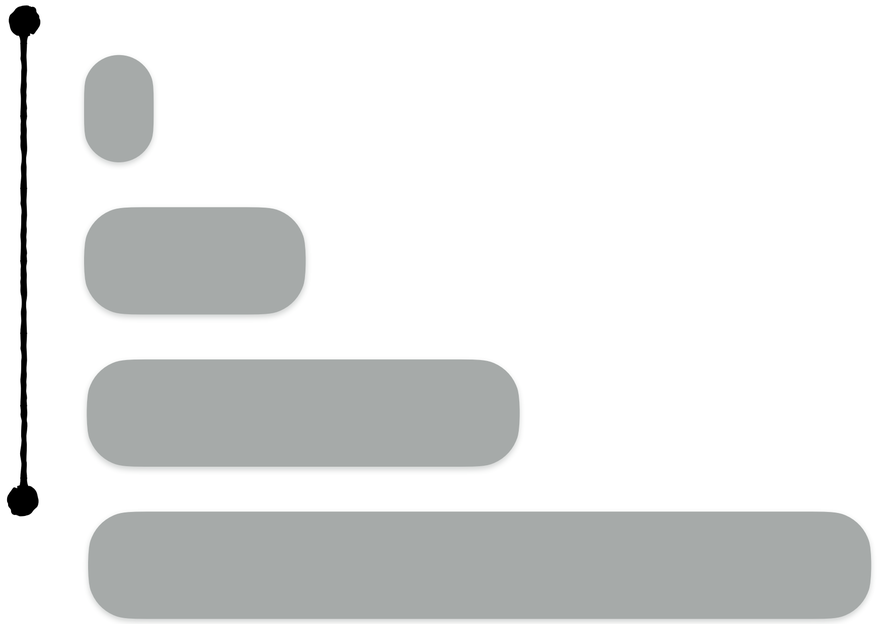
transient
space amplification

Spooky's intuition



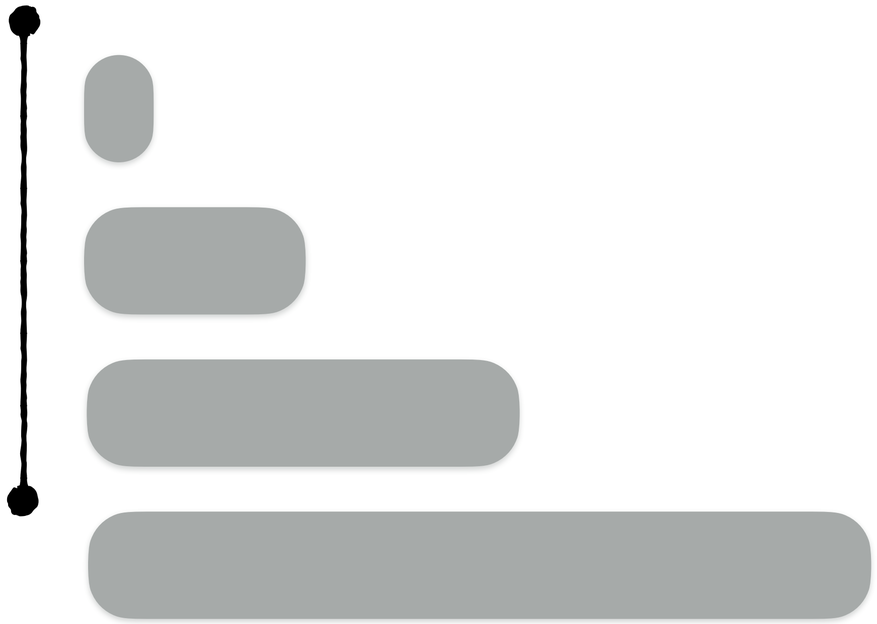
Spooky’s intuition

Full merge
(write optimized)



Spooky's intuition

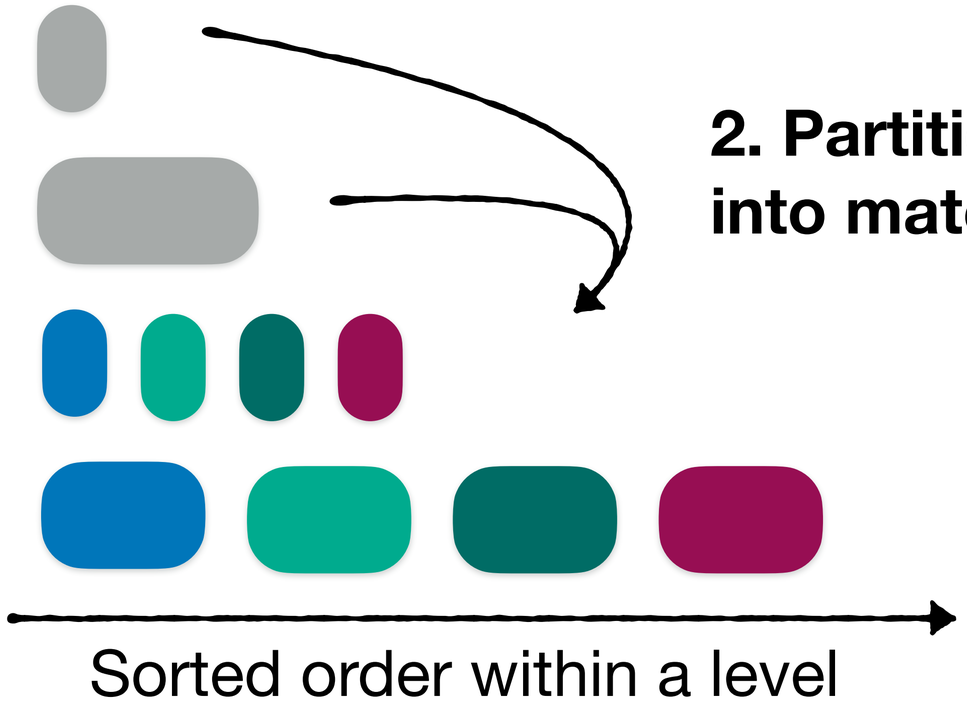
Full merge
(write optimized)



Partial Merge
(space optimized)

Spooky

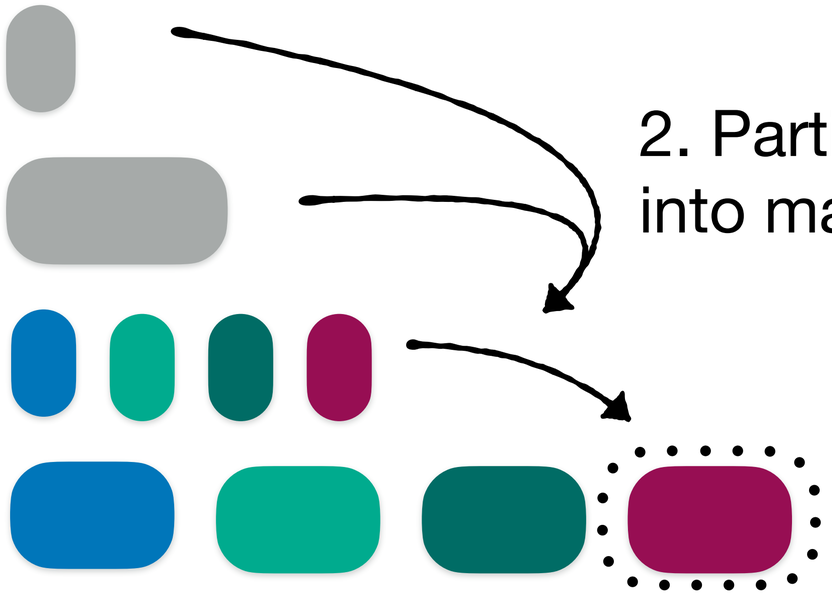
1. Full merge at up to second largest level



2. Partition largest two level into matching pairs of files

Spooky

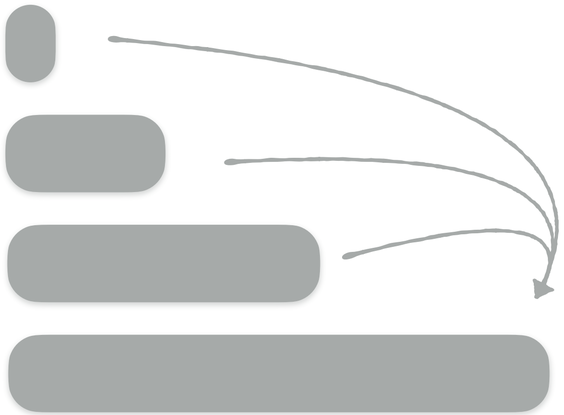
1. Full merge at up to second largest level



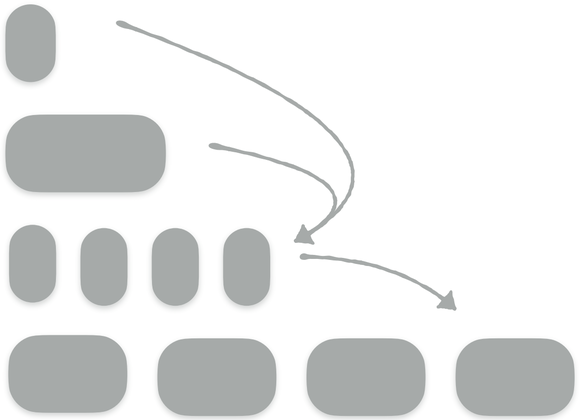
2. Partition largest two level into matching pairs of files

3. Merge one partition at a time

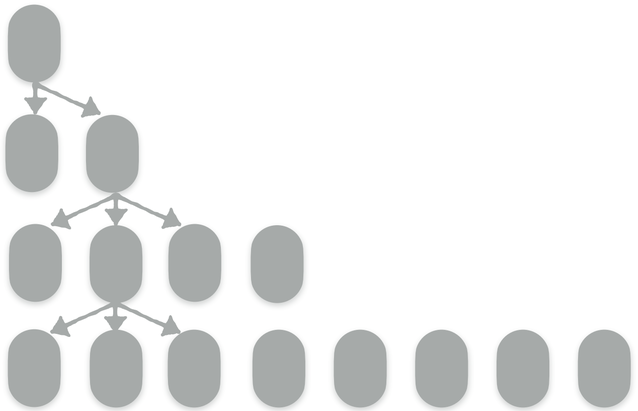
Full



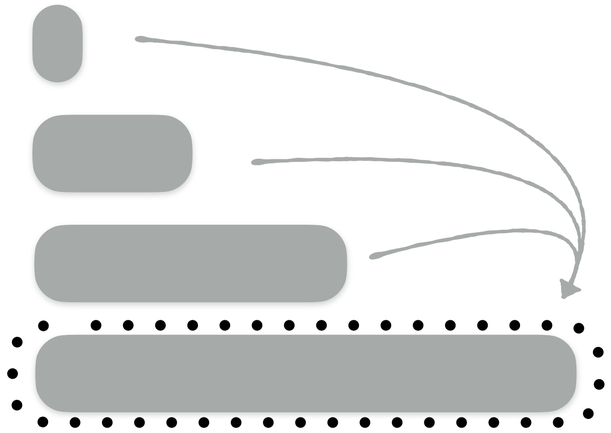
Spooky



Partial

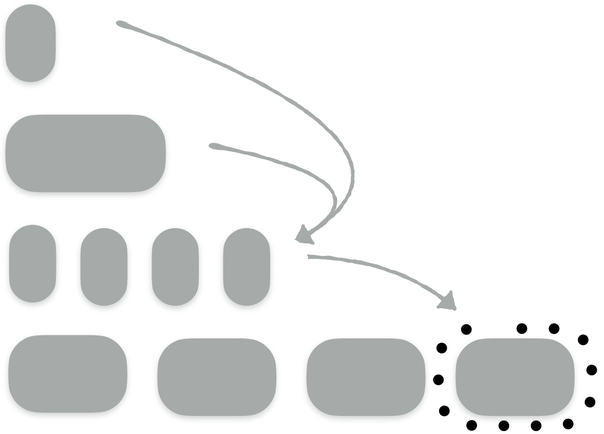


Full



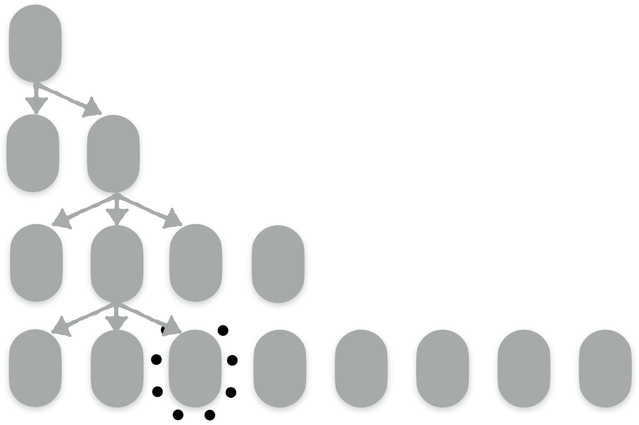
High

Spooky



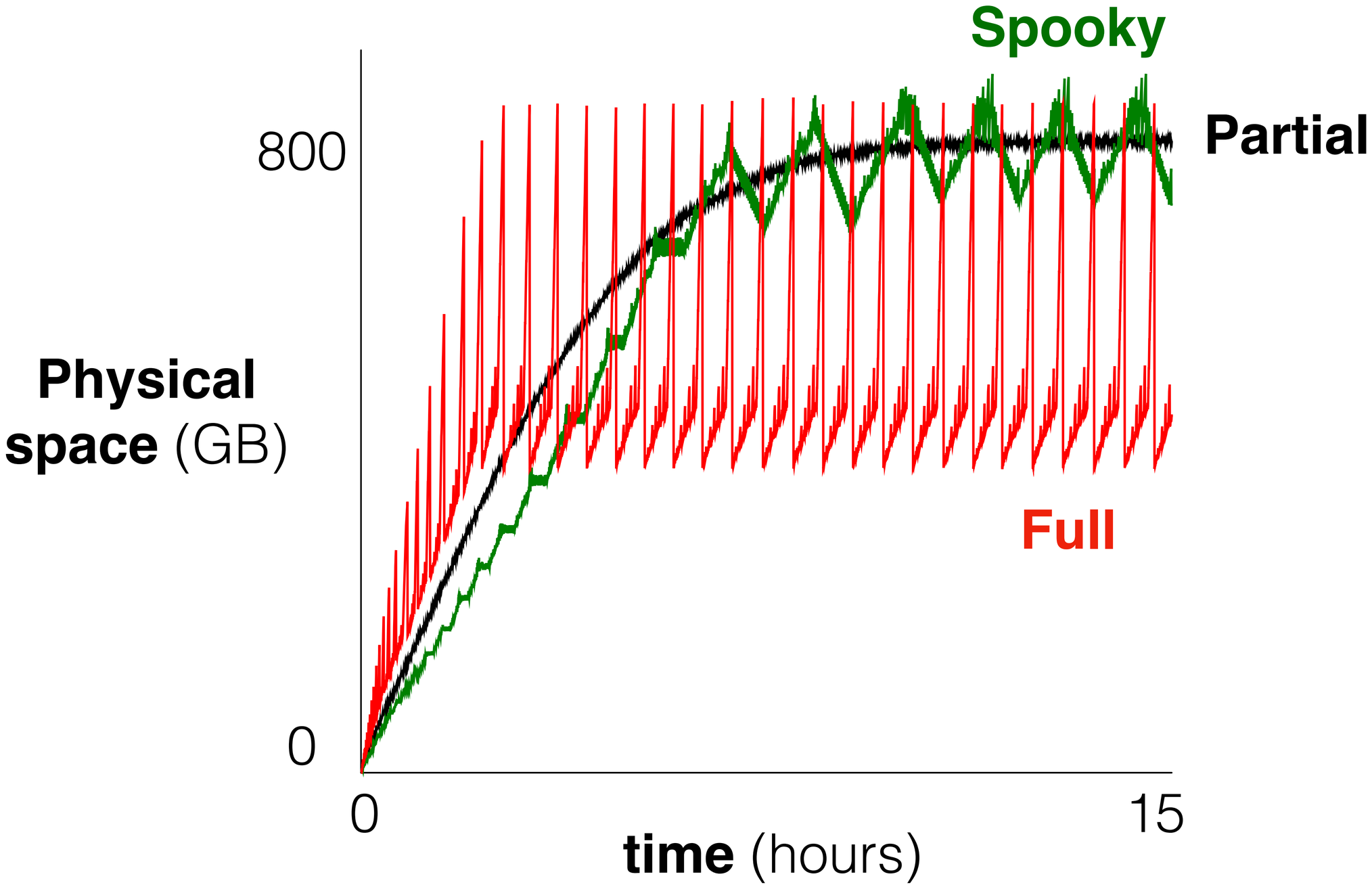
Modest

Partial

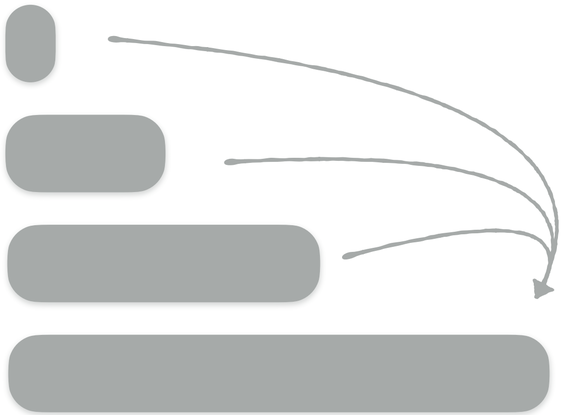


Negligible

space-amplification

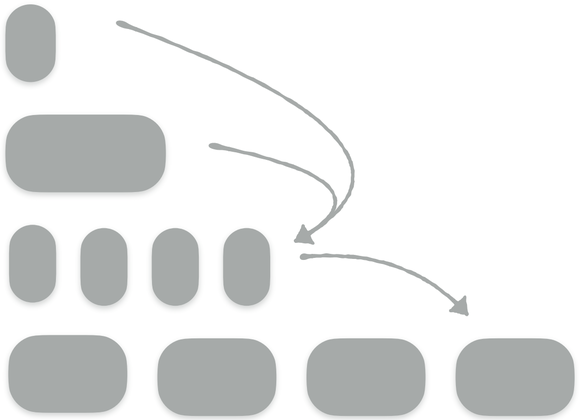

Spooky
Partial
Full
800
0
Physical space (GB)
0
15
time (hours)

Full



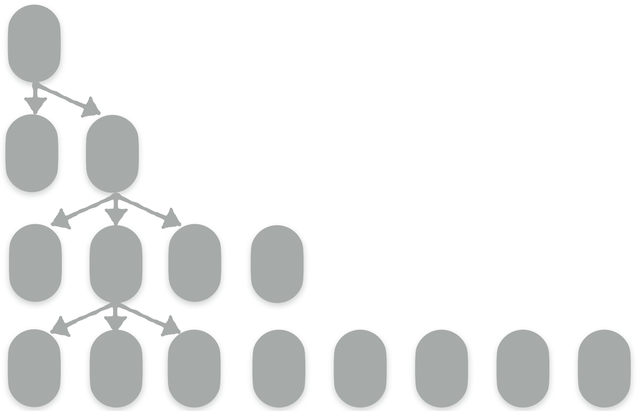
Lowest

Spooky



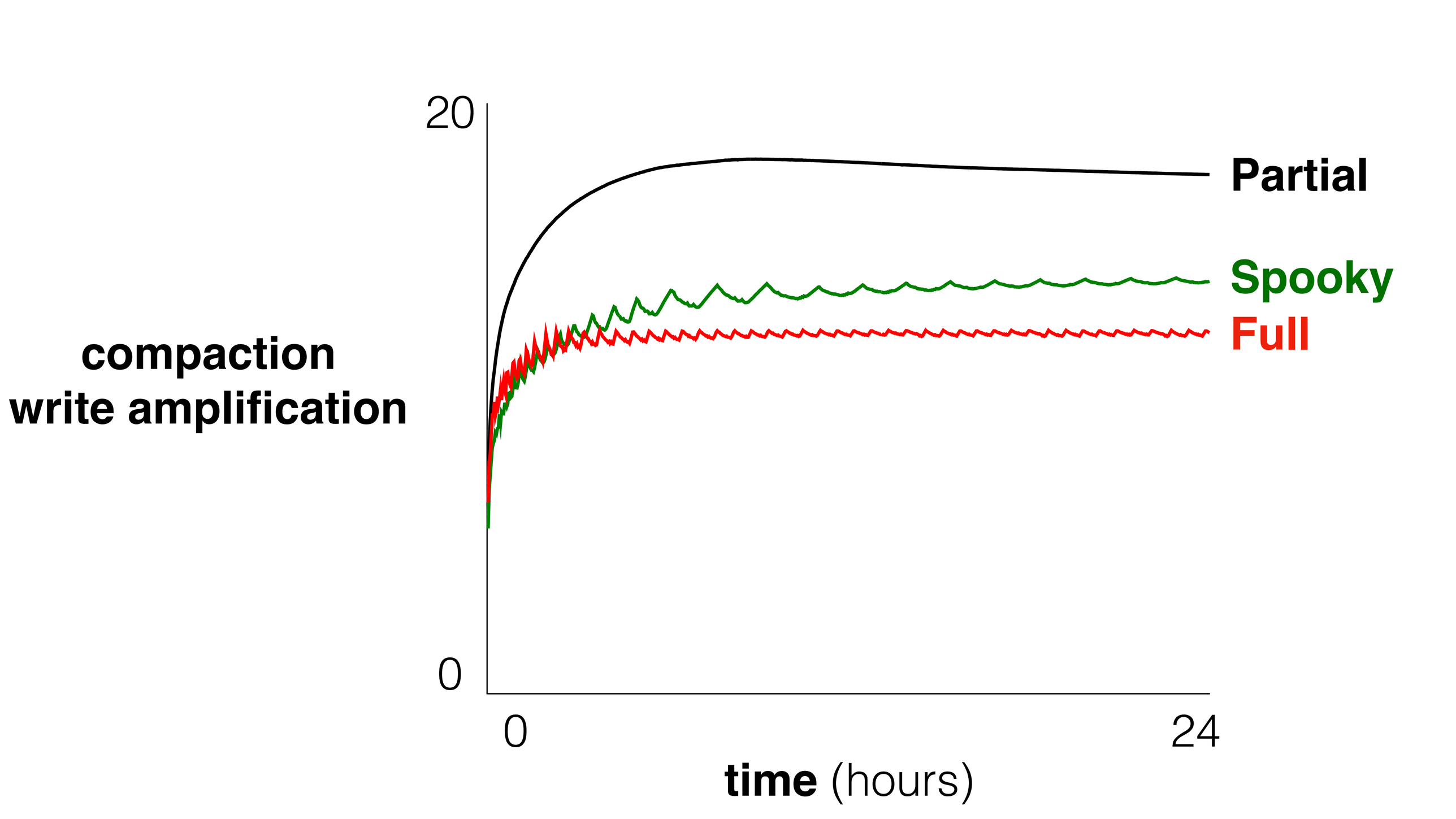
Modest

Partial

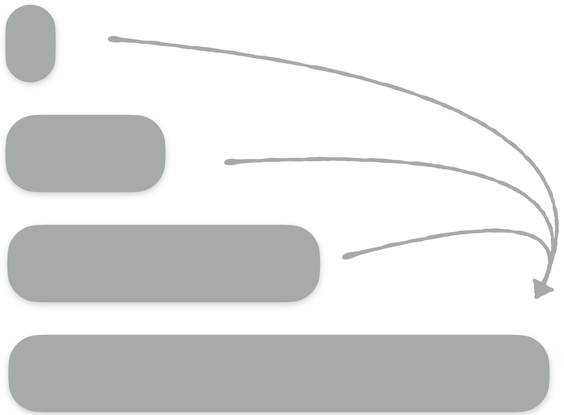


Highest

compaction
write amplification

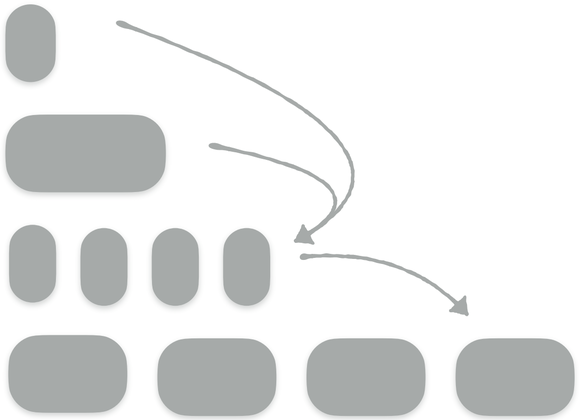

compaction
write amplification
20
0
0
24
time (hours)
Partial
Spooky
Full

Full



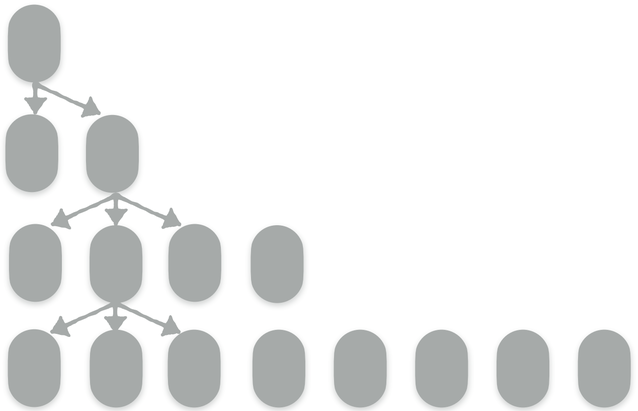
1

Spooky



2-3

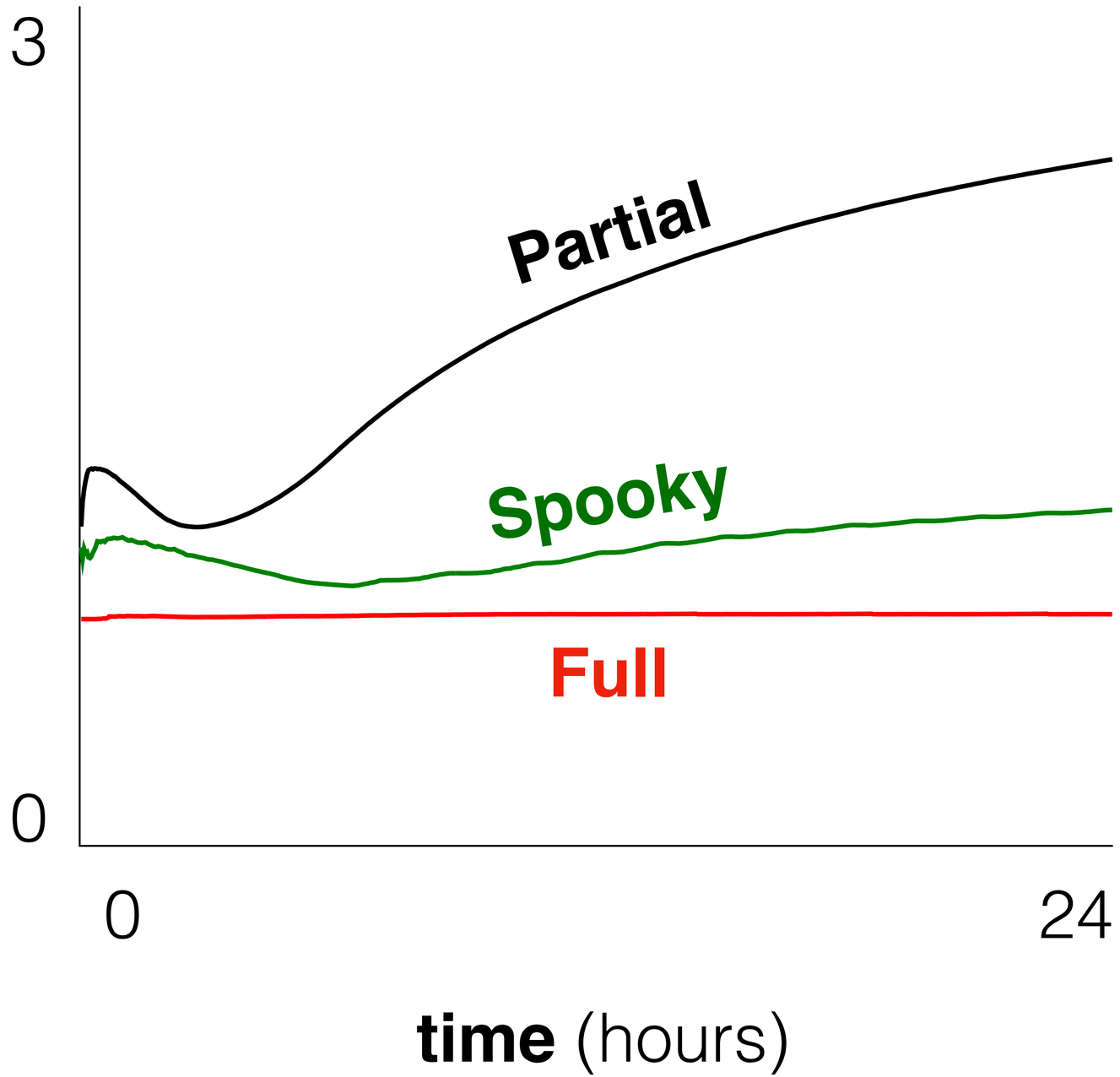
Partial



Unbounded

Simultaneous compactions

Garbage Collection
write amplification



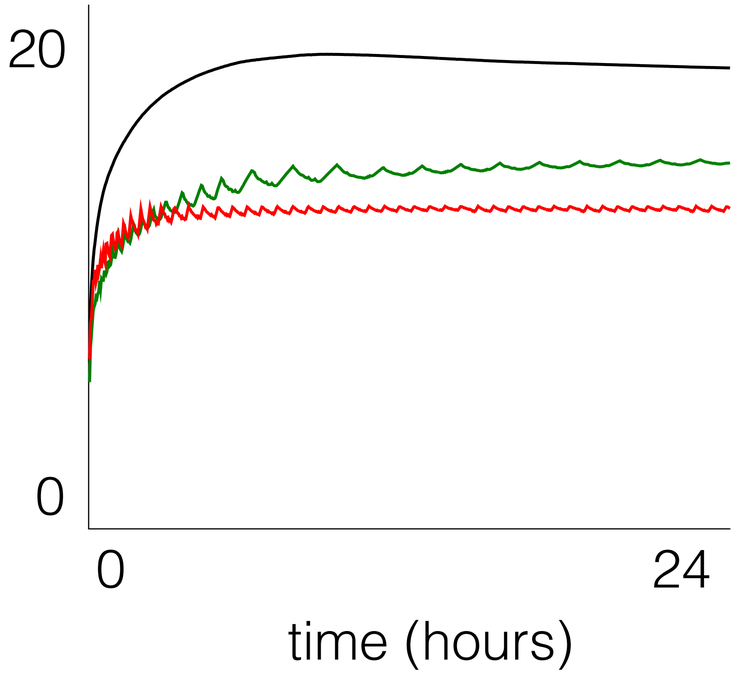
Compaction
write amplification

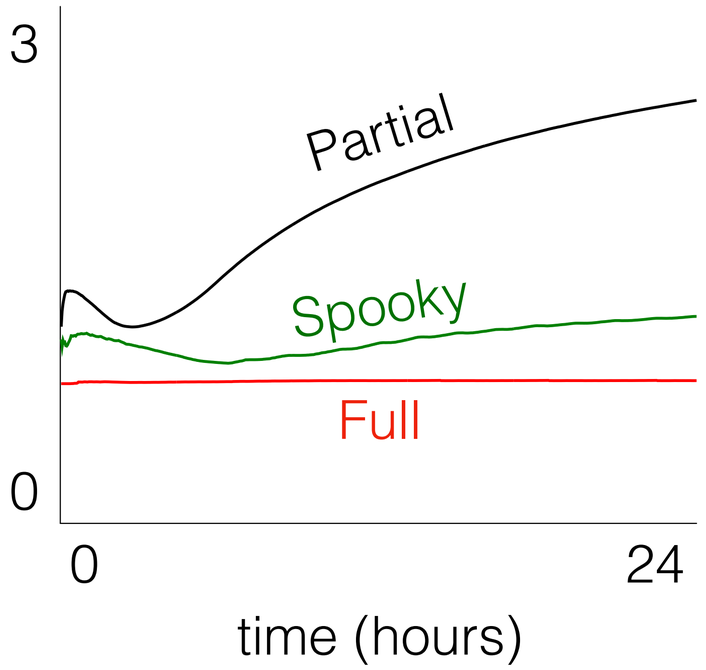
X

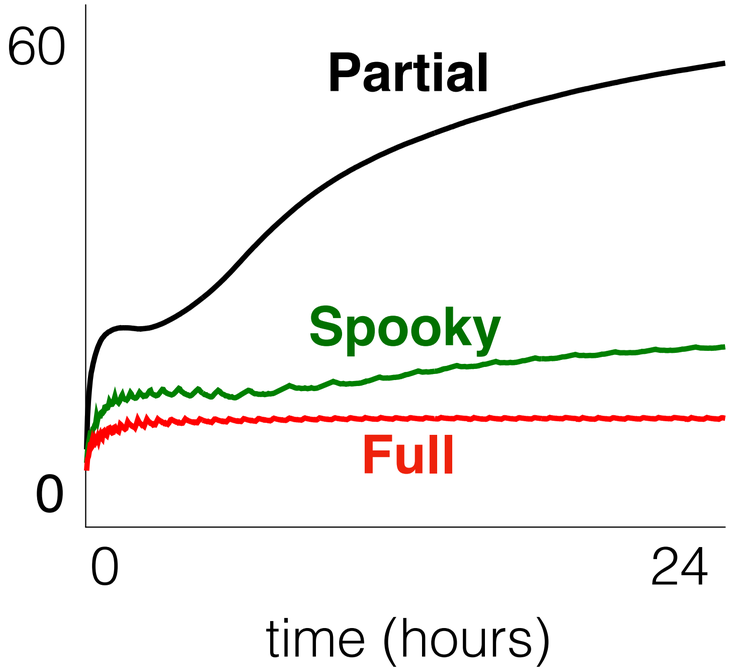
Garbage Collection
write amplification

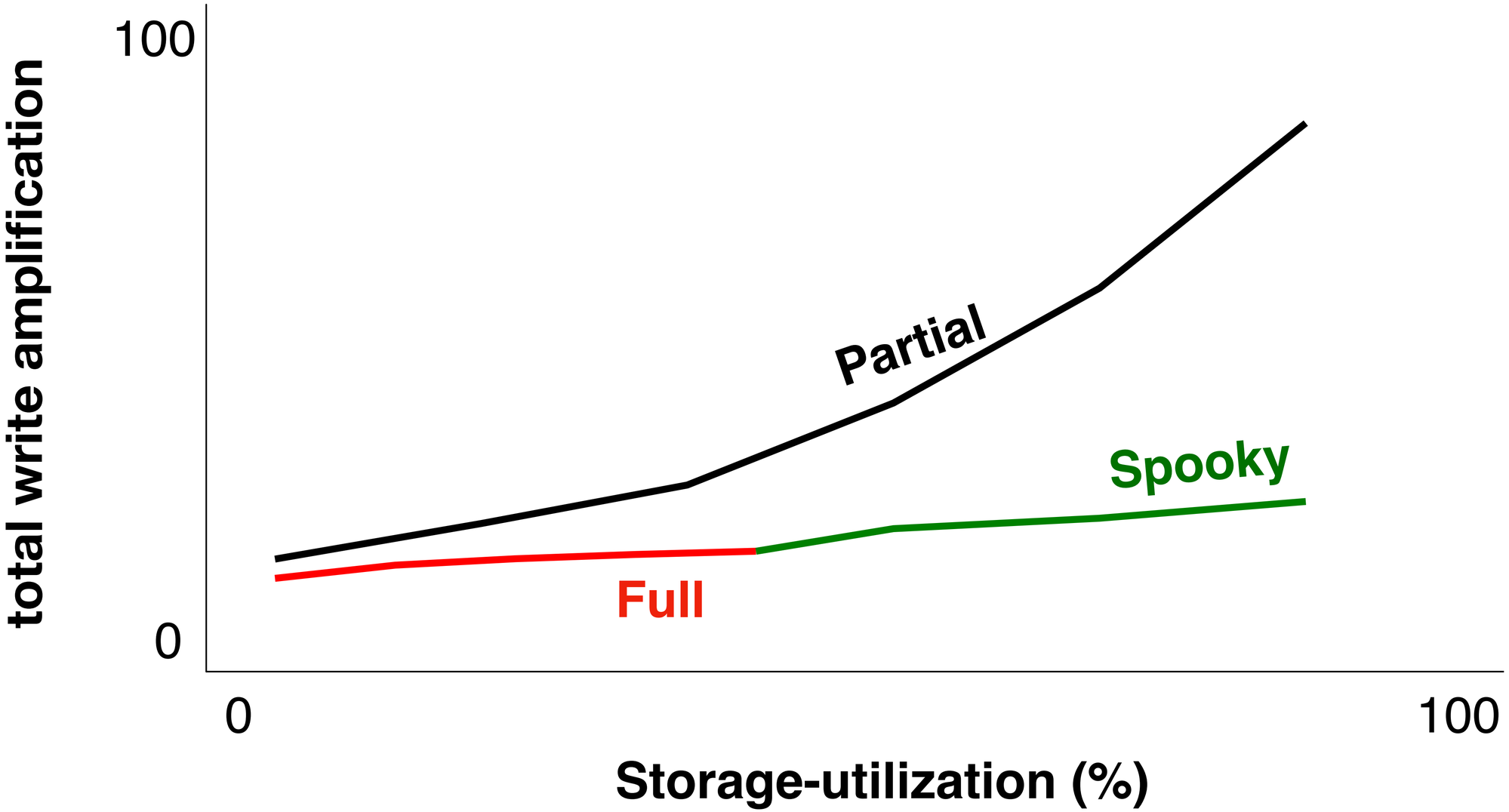
=

Total
write amplification

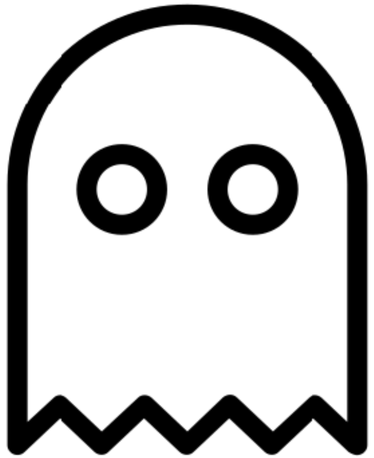






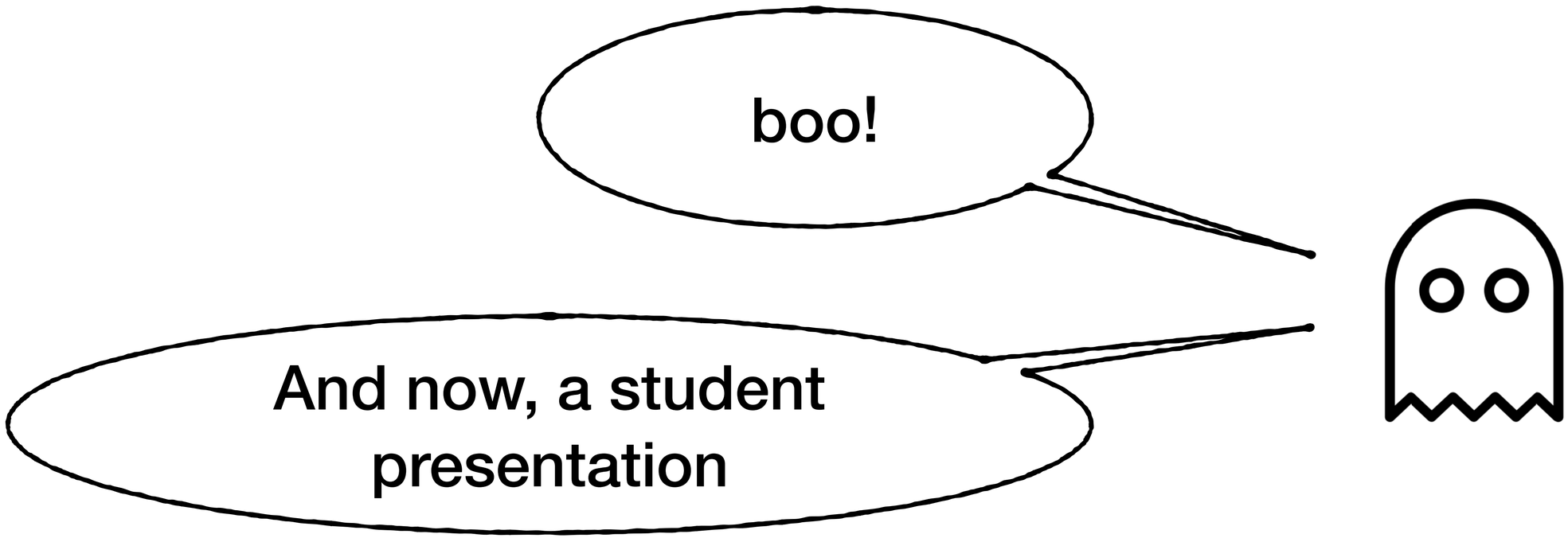

100
0
total write amplification
Partial
Spooky
Full
0
100
Storage-utilization (%)

Spooky



write-amplification

space-amplification


boo!
And now, a student presentation