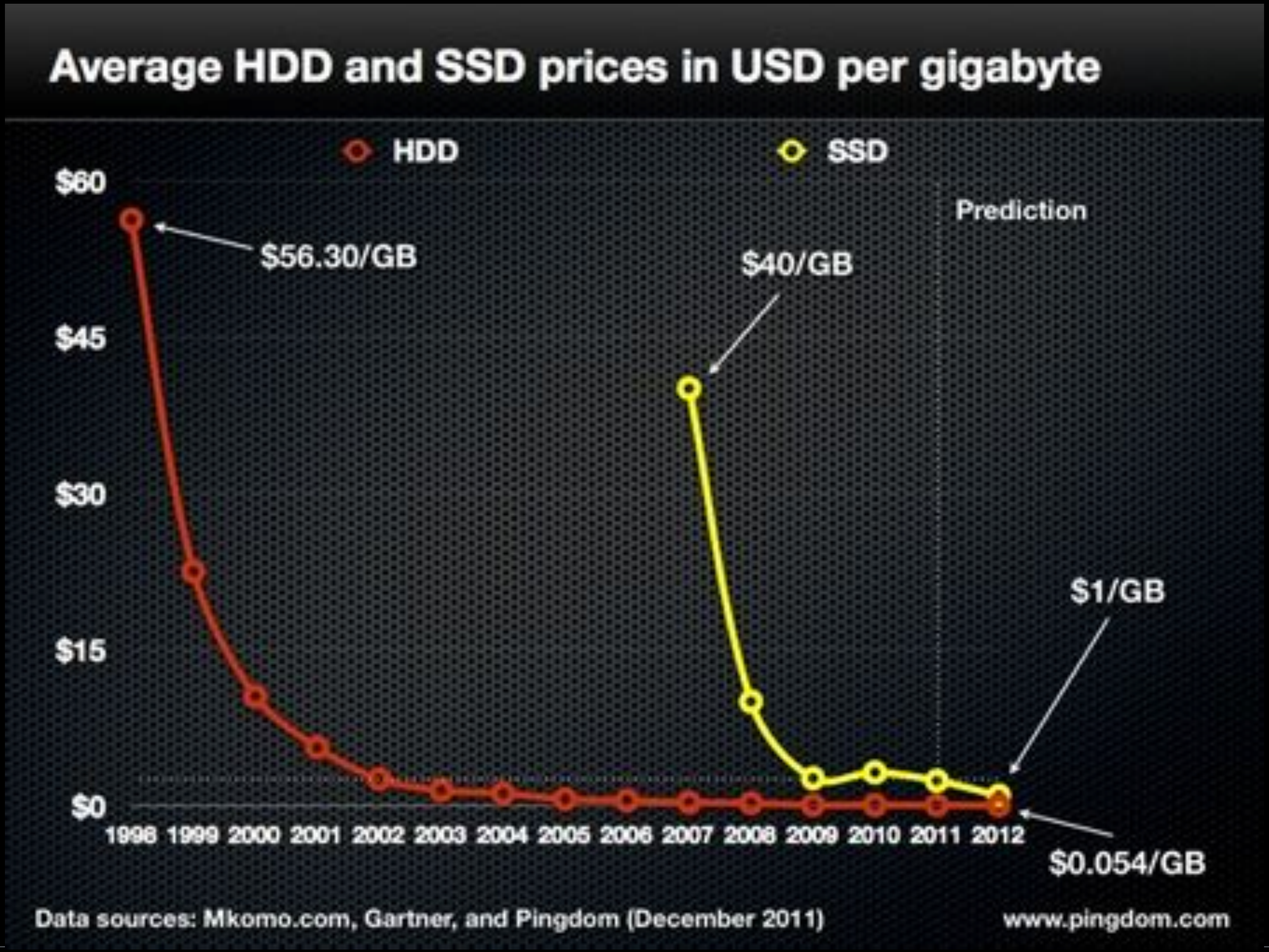
Flash-based SSDs

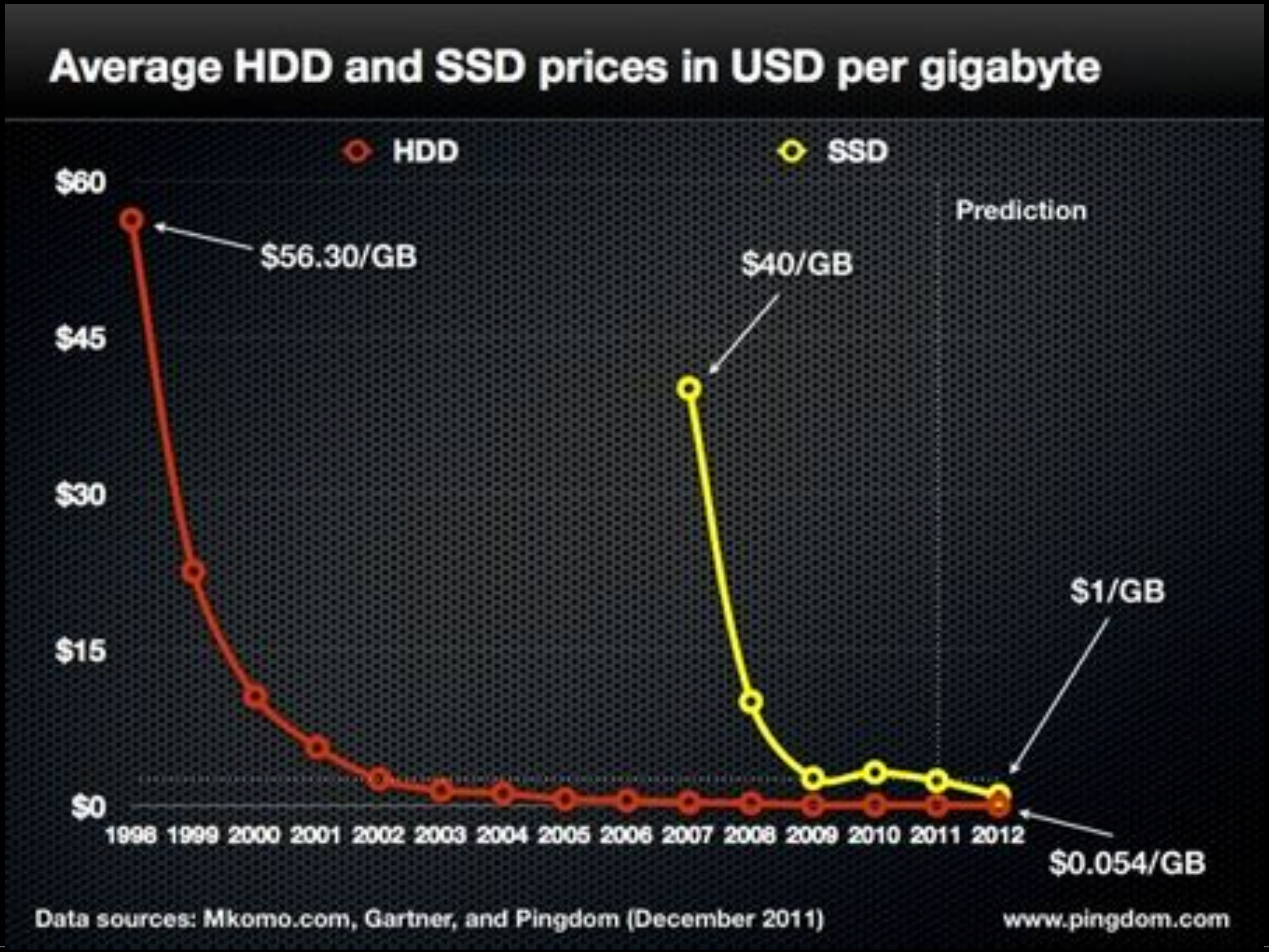
[Material based on slides from Tyler Caraza-Harter]
www: https://tyler.caraza-harter.com

Cost: HDD vs. SSD



Source: http://www.tomshardware.com/news/ssd-hdd-solid-state-drive-hard-disk-drive-prices,14336.html

Cost: HDD vs. SSD



Note: These are trends, not the most up-to-date data.

There are different classes of HDDs and SSDs which complicate this graph, but the thing to note is that there is a gap, but it is narrowing and all costs are trending downward.

Source: http://www.tomshardware.com/news/ssd-hdd-solid-state-drive-hard-disk-drive-prices,14336.html

Disk Overview

I/O requires: seek, rotate, transfer

Inherently:
- not parallel (only one head)
- slow (mechanical)
- poor random I/O (locality around disk head)

Random requests take 10ms+

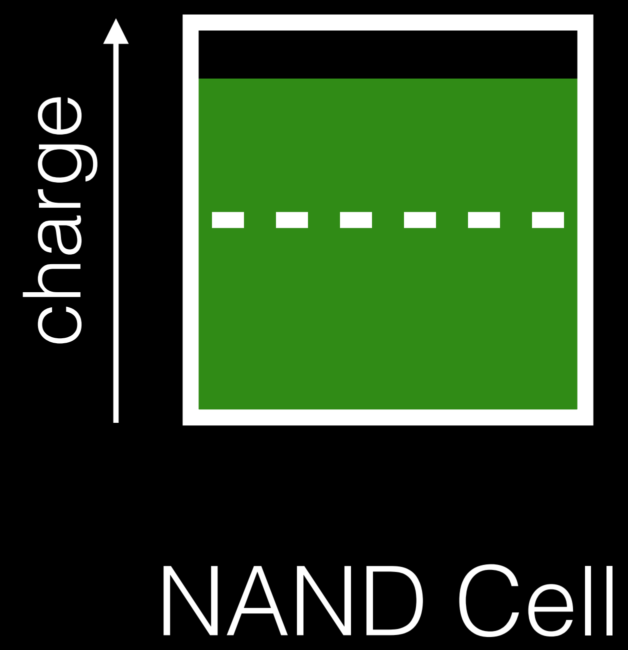
Flash

Hold charge in cells. No moving parts!

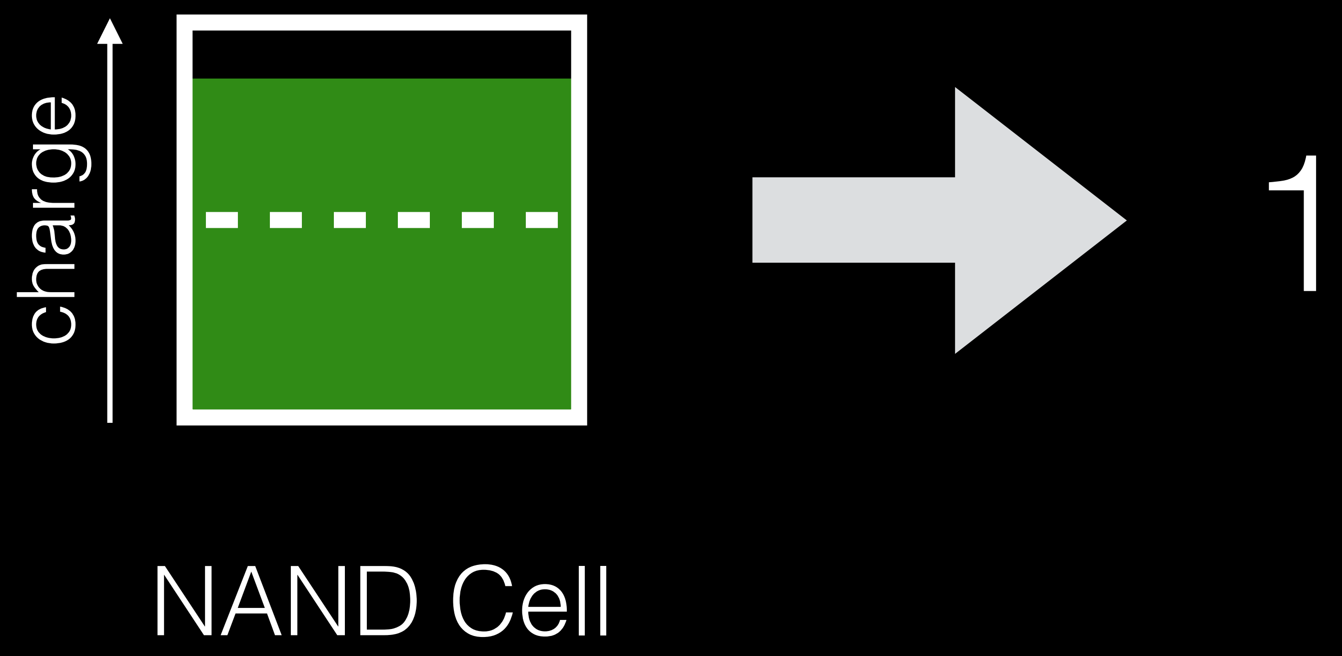
Inherently parallel.

No seeks!

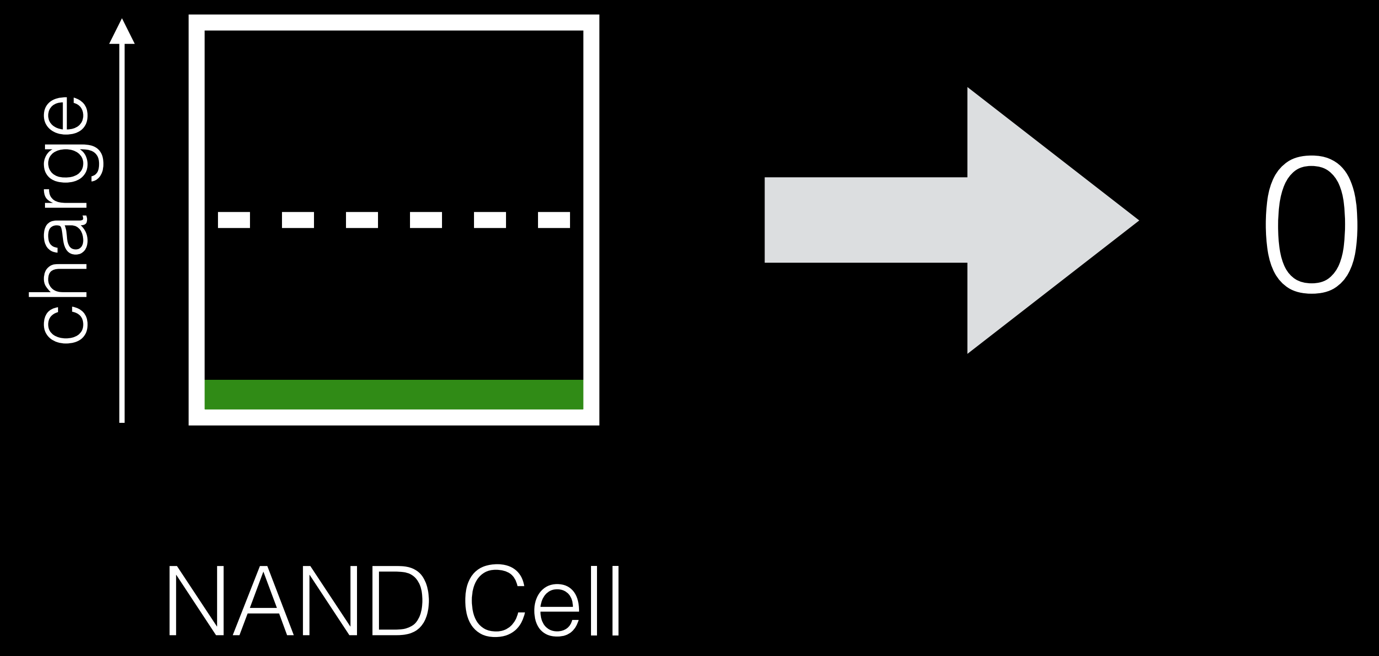
SLC: Single-Level Cell



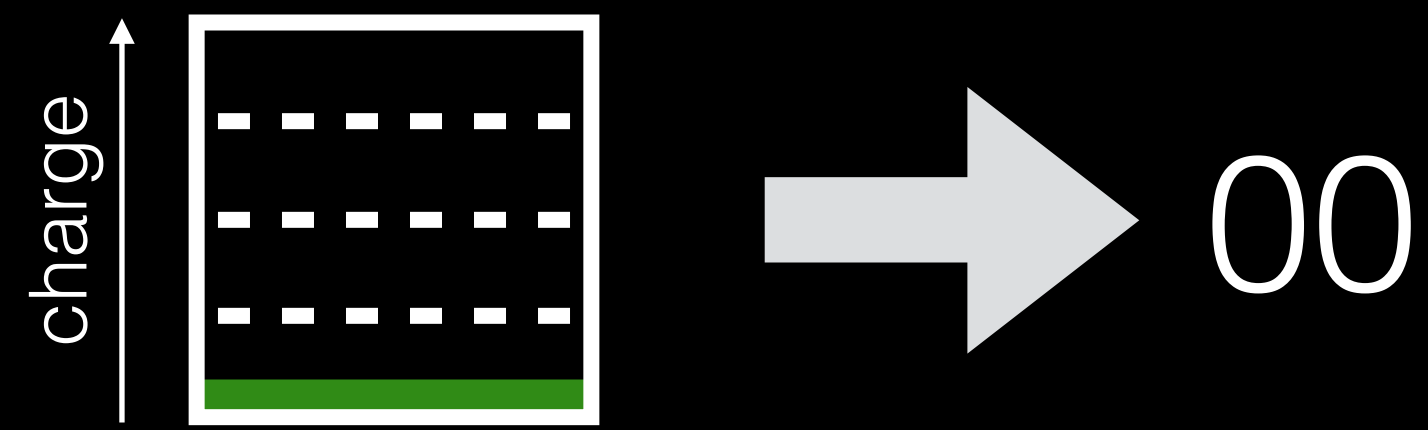
SLC: Single-Level Cell



SLC: Single-Level Cell

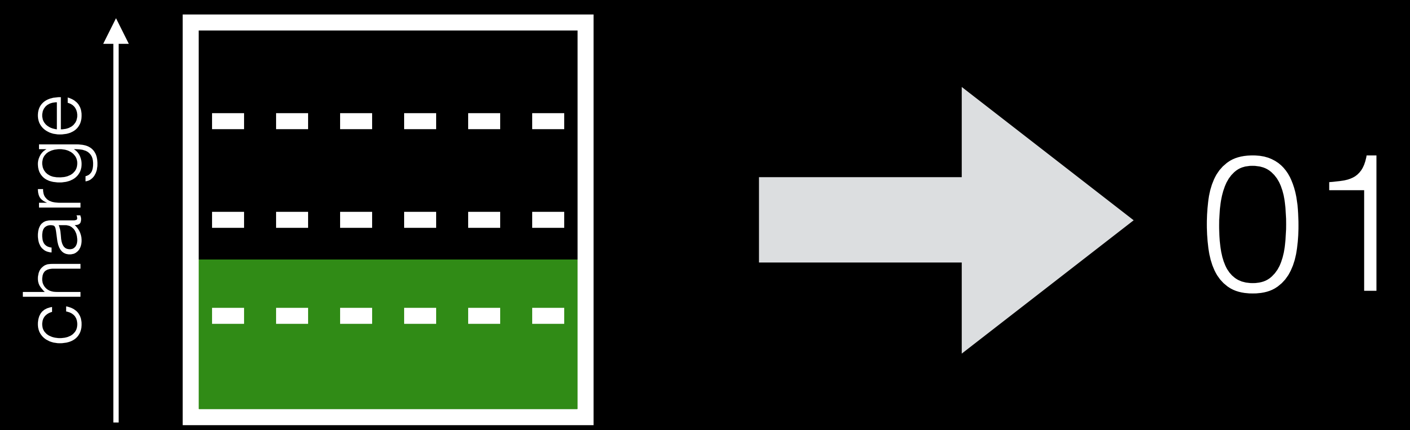


MLC: Multi-Level Cell



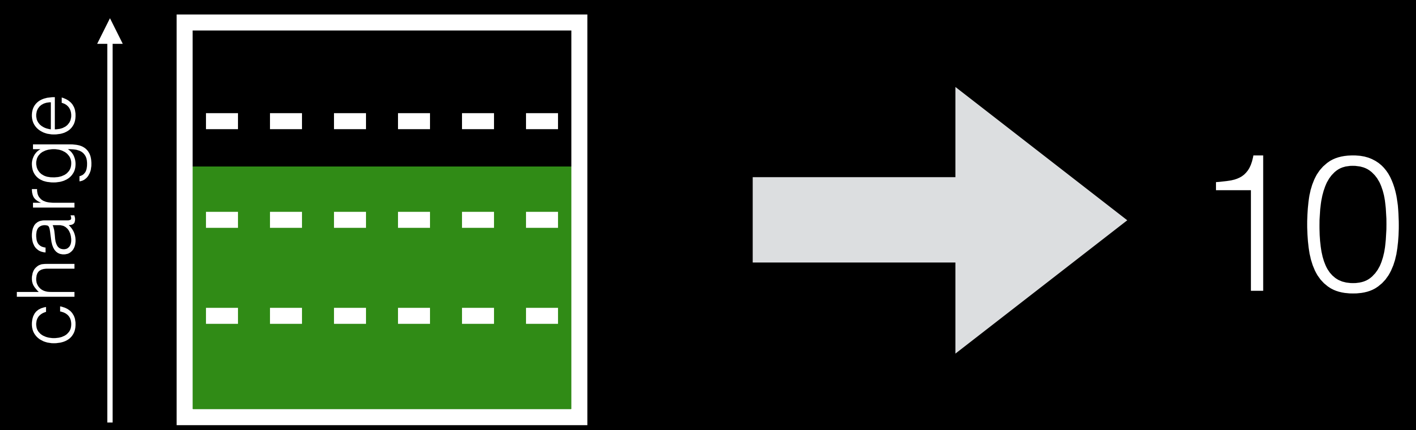
NAND Cell

MLC: Multi-Level Cell



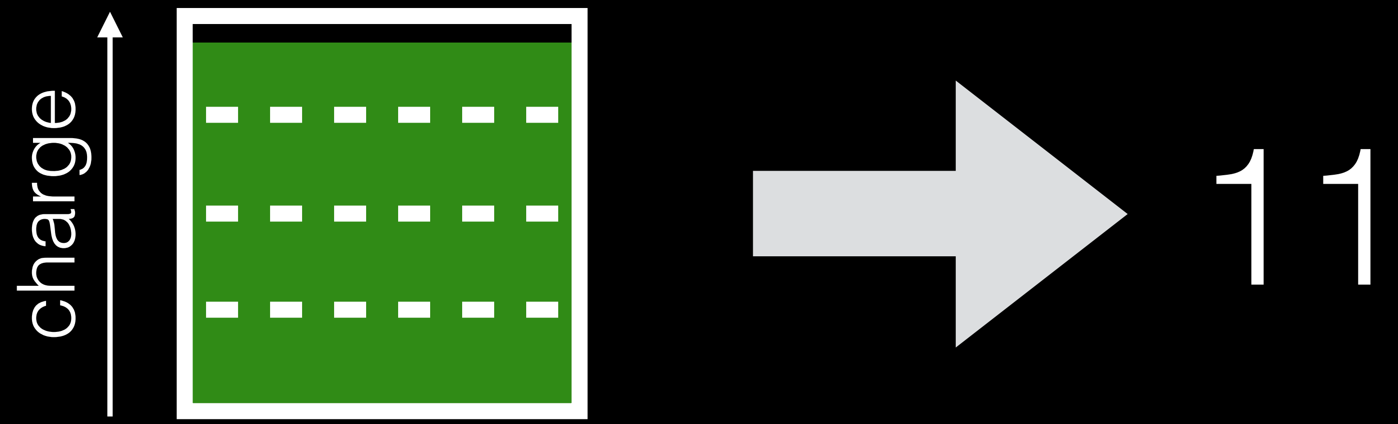
NAND Cell

MLC: Multi-Level Cell



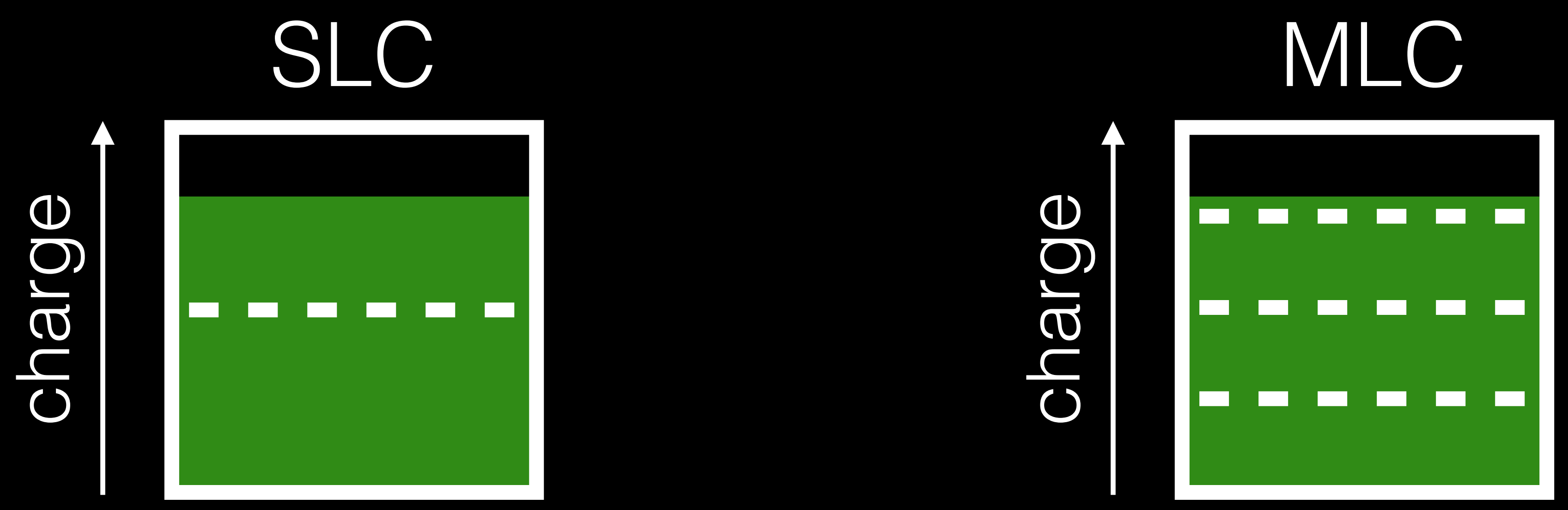
NAND Cell

MLC: Multi-Level Cell



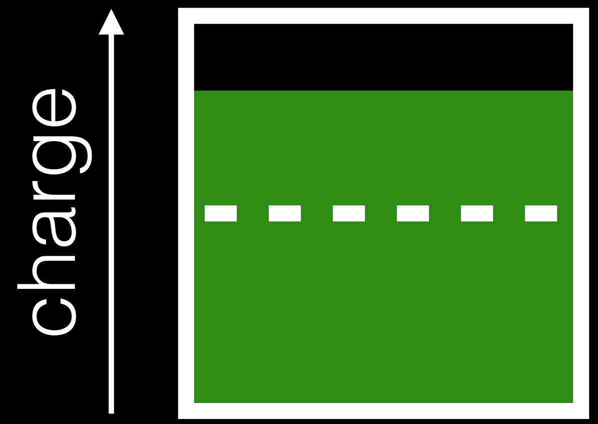
NAND Cell

Single- vs. Multi- Level Cell



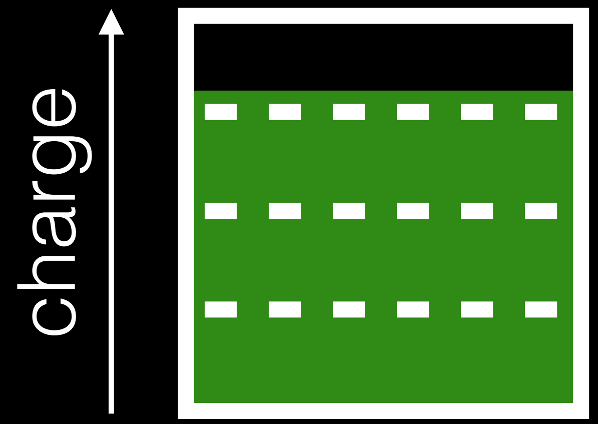
Single- vs. Multi- Level Cell

SLC



expensive
robust

MLC



cheap
sensitive

Wearout

Problem: flash cells wear out after being overwritten too many times.

MLC: ~10K times
SLC: ~100K times

Usage strategy:

Wearout

Problem: flash cells wear out after being overwritten too many times.

MLC: ~10K times
SLC: ~100K times

Usage strategy: wear leveling.
- prevents some cells from wearing out while others still fresh.

Banks

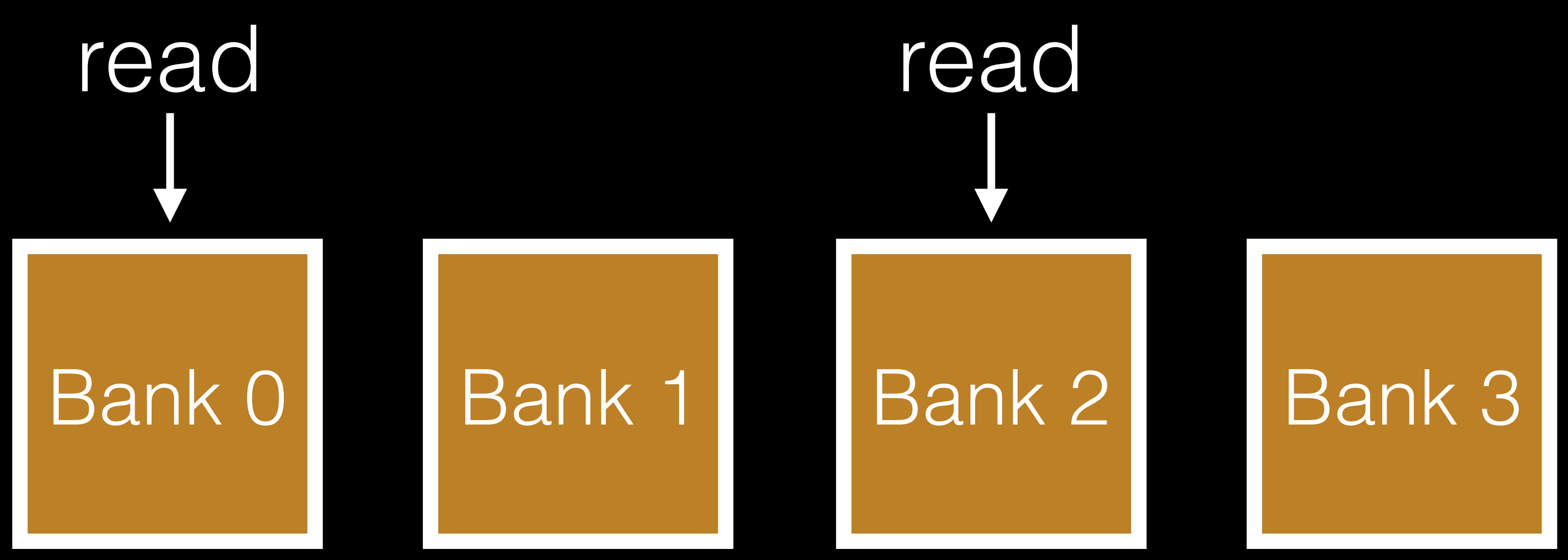
Flash devices are divided into banks (aka, planes).

Banks can be accessed in parallel.

Banks

Flash devices are divided into banks (aka, planes).

Banks can be accessed in parallel.



Banks

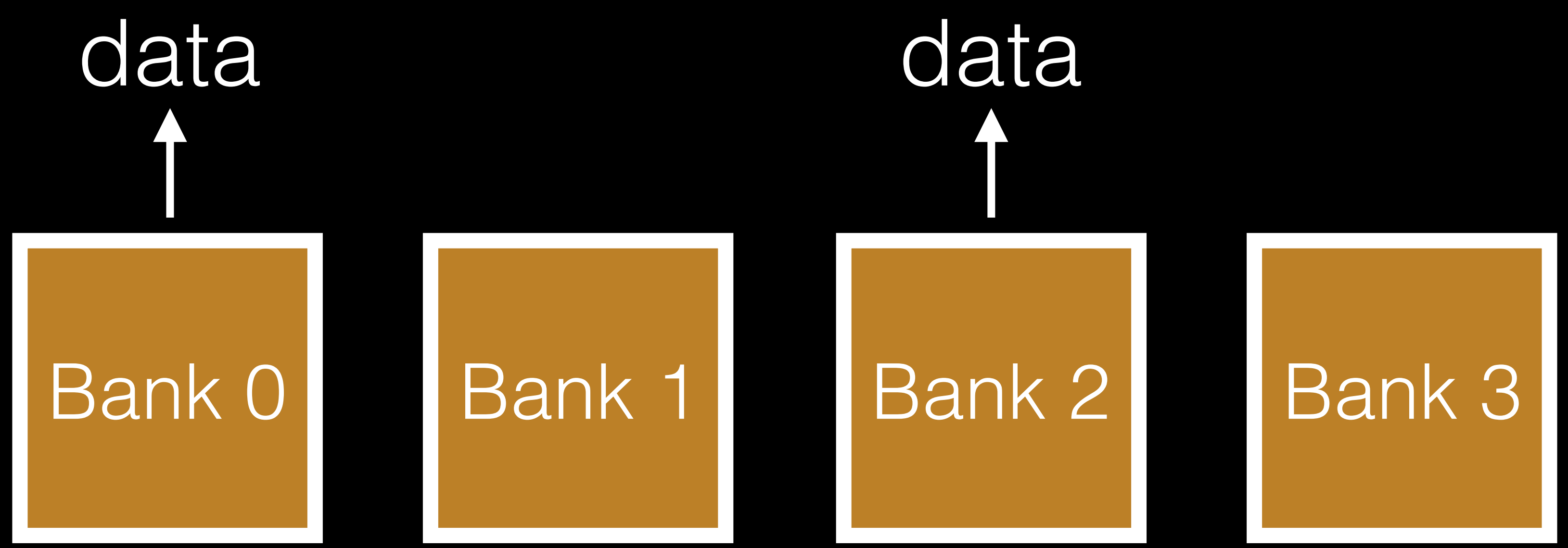
Flash devices are divided into banks (aka, planes).

Banks can be accessed in parallel.

Banks

Flash devices are divided into banks (aka, planes).

Banks can be accessed in parallel.



Banks

Flash devices are divided into banks (aka, planes).

Banks can be accessed in parallel.

Flash Writes

Writing 0’s:
- fast, fine-grained

Writing 1’s:
- slow, course-grained

Flash Writes

Writing 0's:
- fast, fine-grained
- called "program"

Writing 1's:
- slow, course-grained
- called "erase"

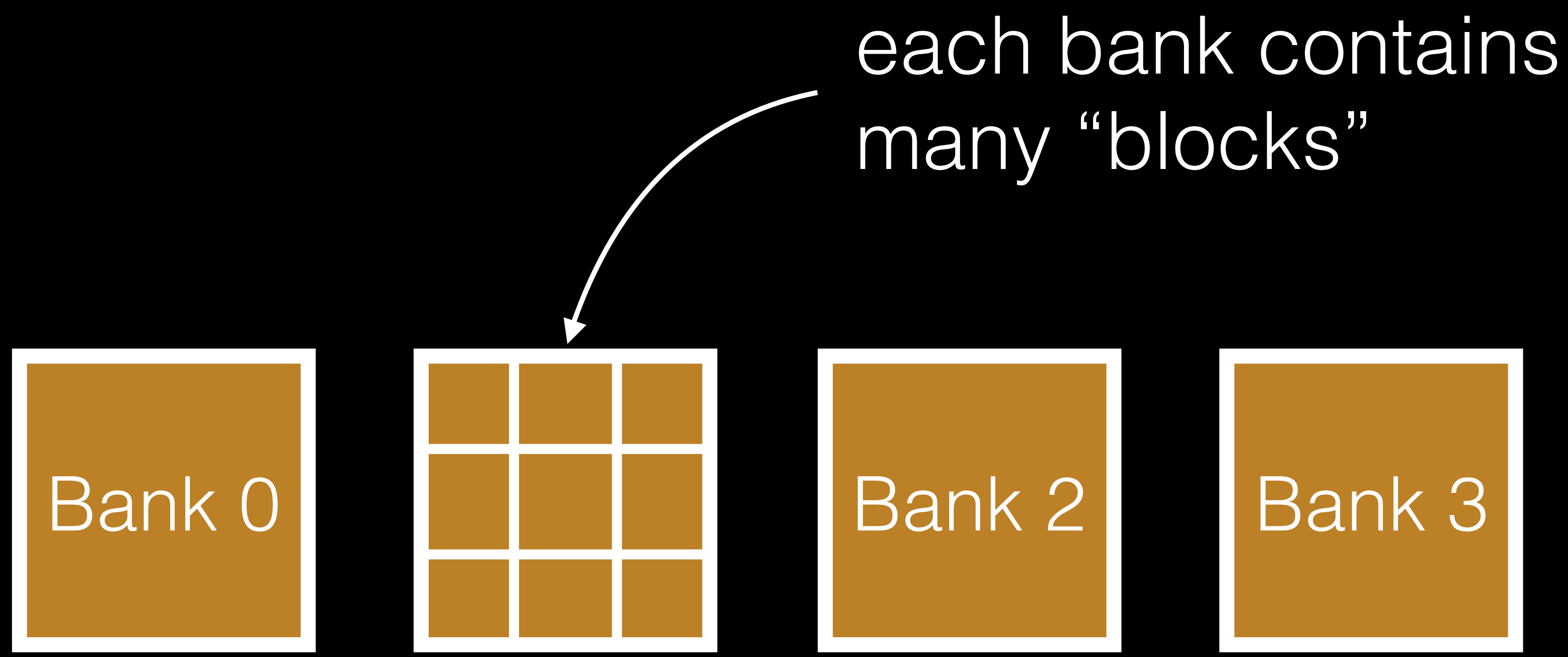
Flash Writes

Writing 0’s:
- fast, fine-grained [pages]
- called “program”

Writing 1’s:
- slow, course-grained [blocks]
- called “erase”

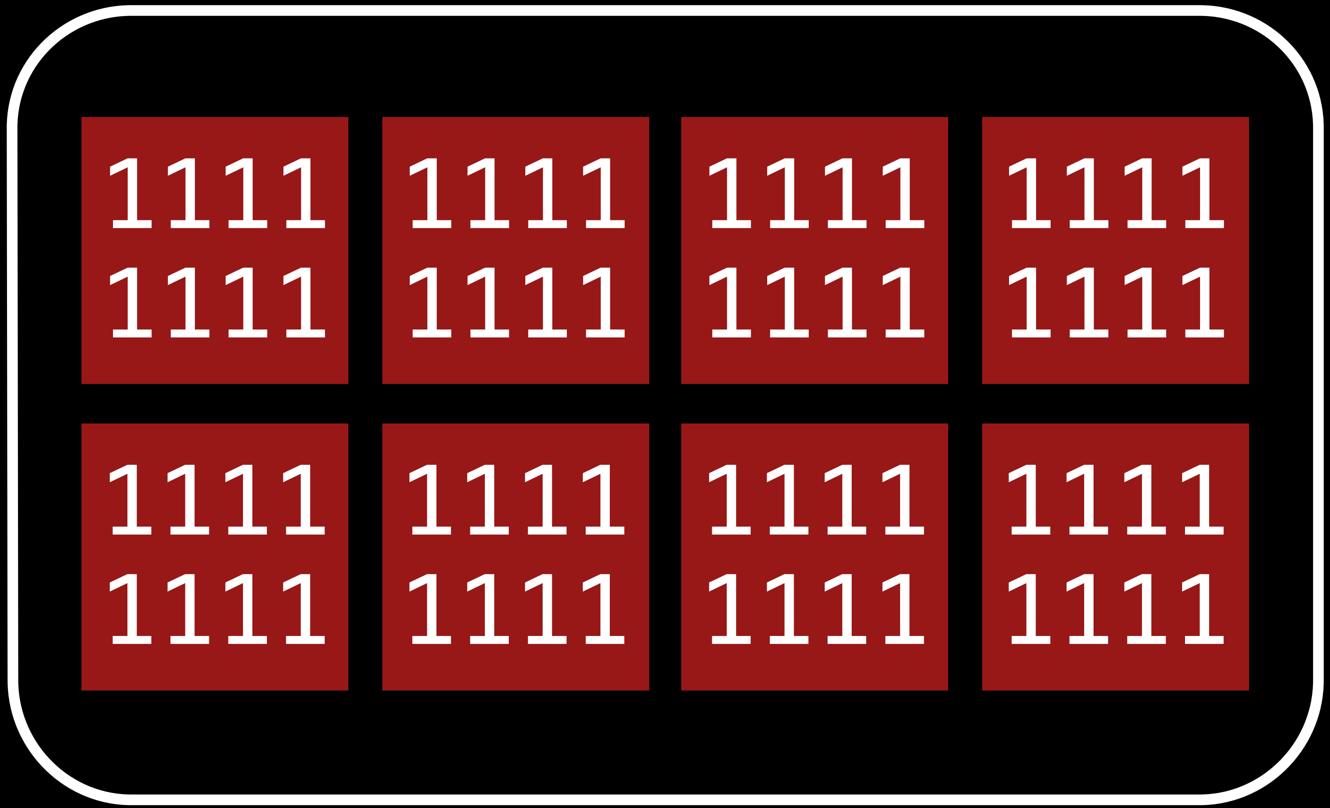
A Bank Consists of Blocks

A Bank Consists of Blocks



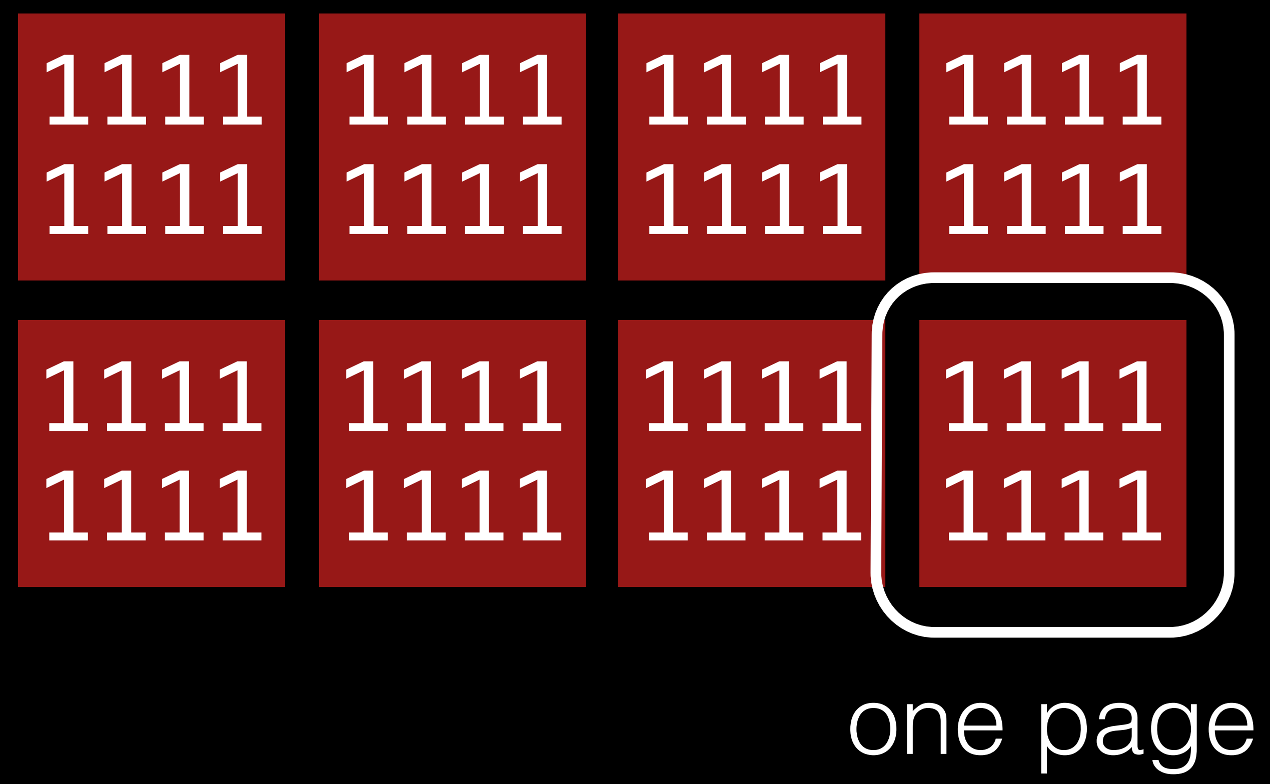
A Block Consists of Pages

A Block Consists of Pages

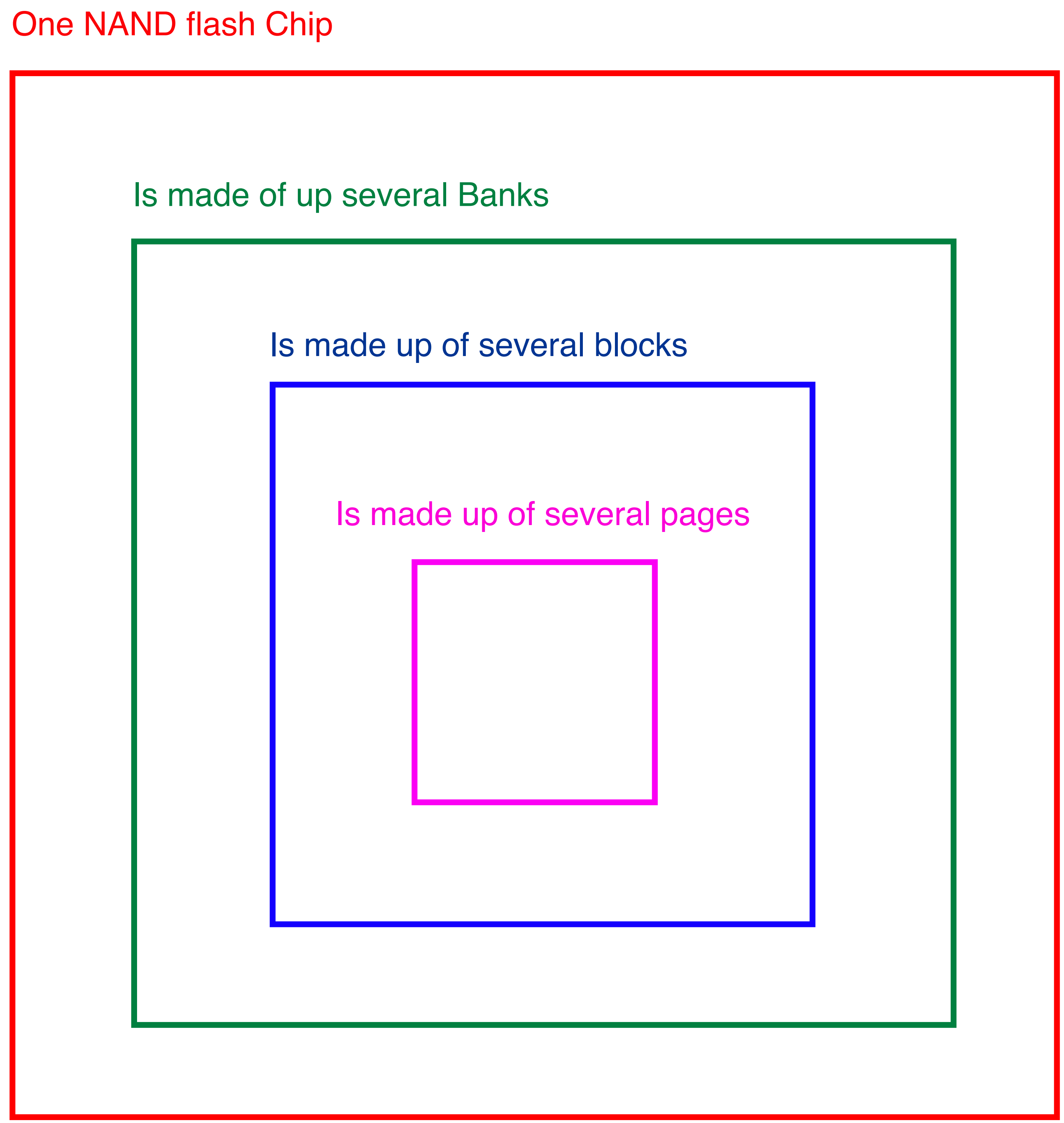


one block

Block


1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
one page

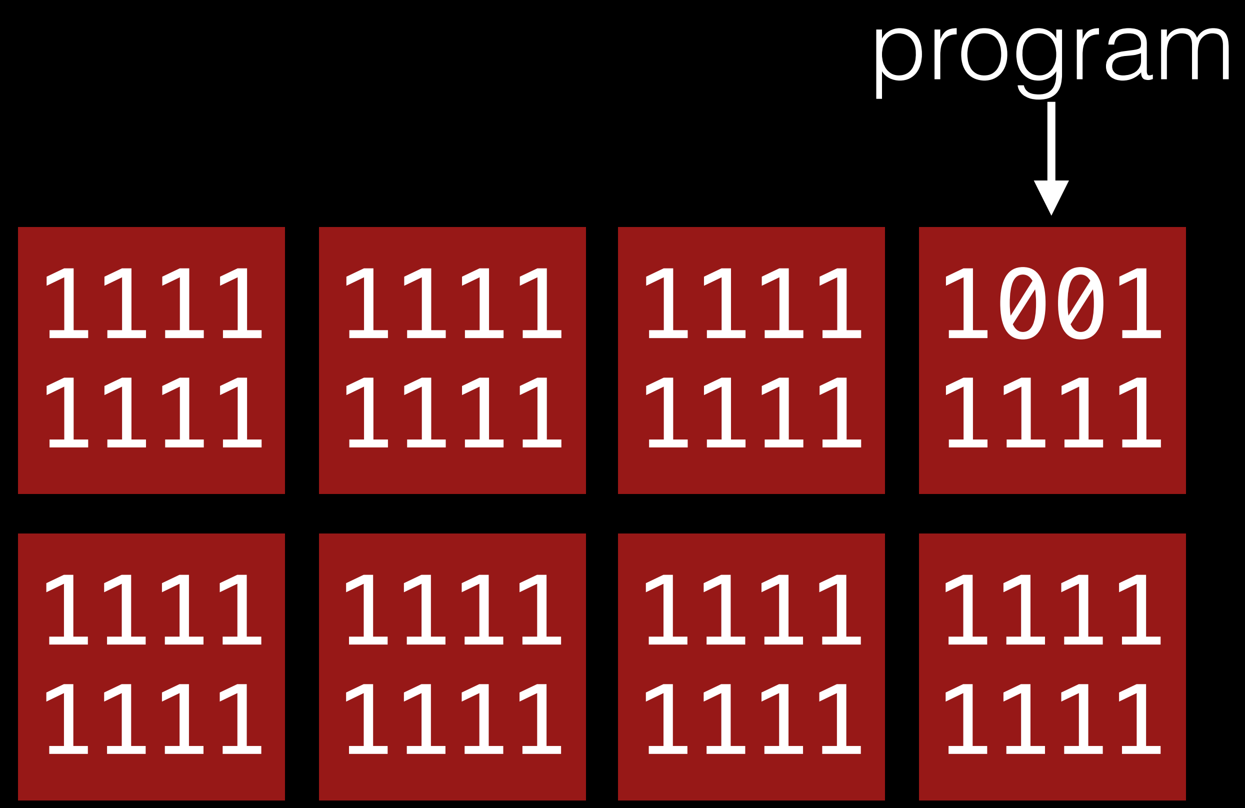
The Heirarchy of SSD components:



Block


1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111

Block


program
1111 1111
1111 1111
1111 1111
1001 1111
1111 1111
1111 1111
1111 1111
1111 1111

Block

1111 1111	1111 1111	1111 1111	1001 1111
1111 1111	1111 1111	1111 1111	1111 1111

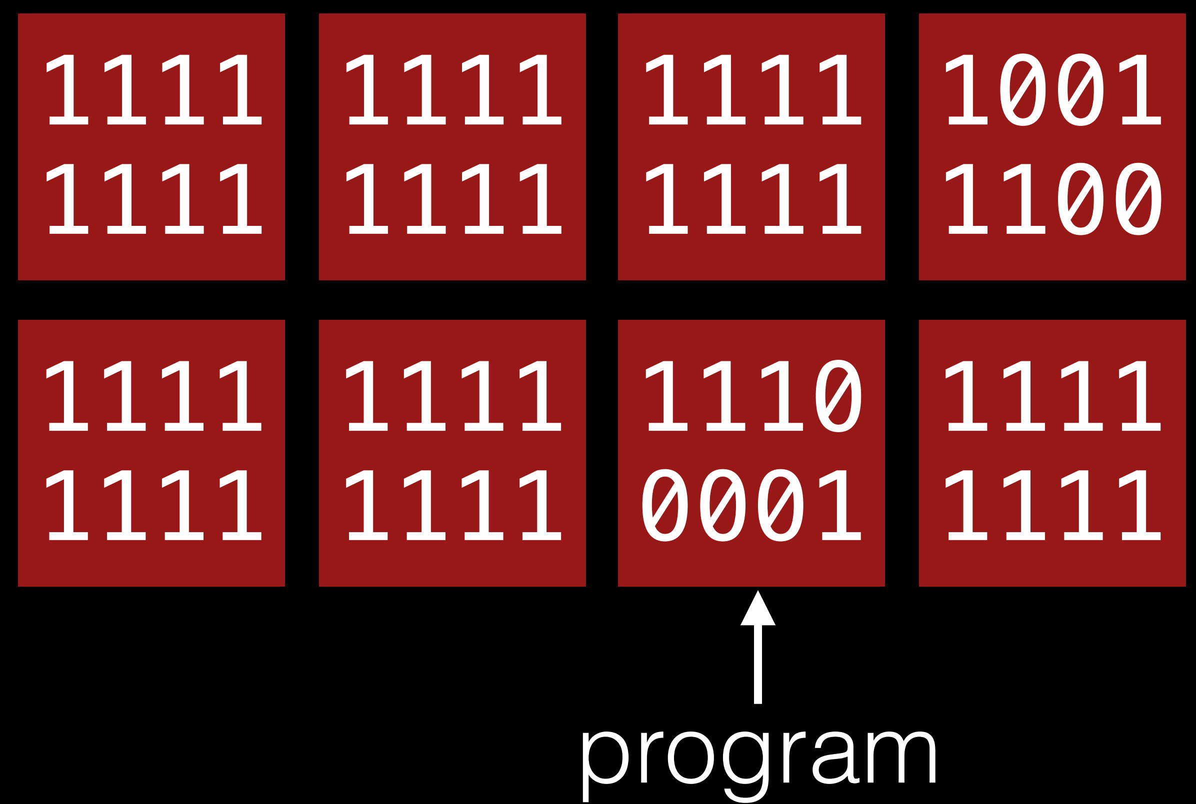
Block


program
1111
1111
1111
1111
1111
1111
1001
1100
1111
1111
1111
1111
1111
1111
1111
1111

Block

1111 1111	1111 1111	1111 1111	1001 1100
1111 1111	1111 1111	1111 1111	1111 1111

Block


1111 1111
1111 1111
1111 1111
1001 1100
1111 1111
1111 1111
1110 0001
1111 1111
program

Block

1111 1111	1111 1111	1111 1111	1001 1100
1111 1111	1111 1111	1110 0001	1111 1111

Block

1111 1111	1111 1111	1111 1111	1001 1100
1111 1111	1111 1111	1110 0001	1111 1111

erase

Block


1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111

erase

Block

1111 1111	1111 1111	1111 1111	1111 1111
1111 1111	1111 1111	1111 1111	1111 1111

APIs

	disk	flash
read		
write		

APIs

	disk	flash
read	read sector	read page
write		

APIs

	disk	flash
read	read sector	read page
write	write sector	program page (0's) erase block (1's)

Flash Hierarchy

Plane: 1024 to 4096 blocks
- planes accessed in parallel

Block: 64 to 256 pages
- unit of erase

Page: 2 to 8 KB
- unit of read and program

Disk vs. Flash Performance

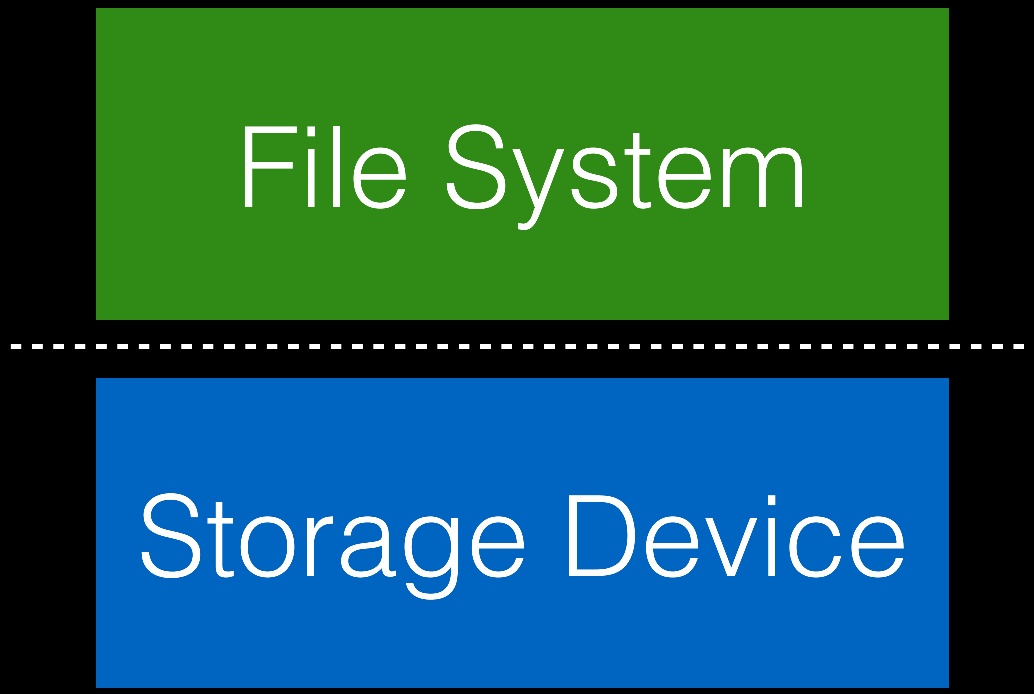
Throughput:
- disk: ~130 MB/s (sequential)
- flash: ~200 MB/s - 550 MB/s

Disk vs. Flash Performance

Throughput:
- disk: ~130 MB/s (sequential)
- flash: ~200 MB/s **- 550 MB/s**

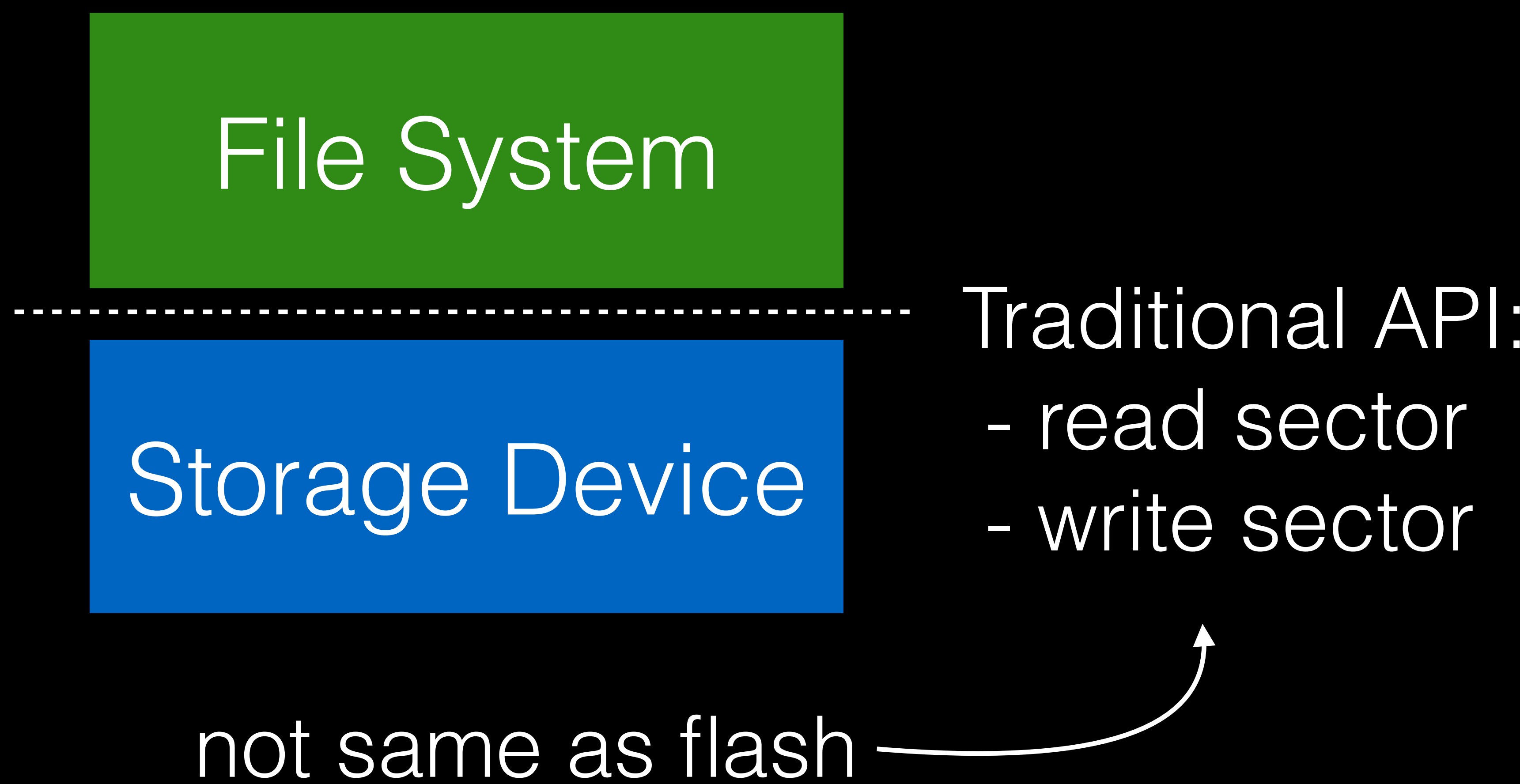
Latency
- disk: ~10 ms (one op)
- flash
  - read: 10-50 us
  - program: 200-500 us
  - erase: 2 ms

Traditional File Systems



Traditional API:
- read sector
- write sector

Traditional File Systems



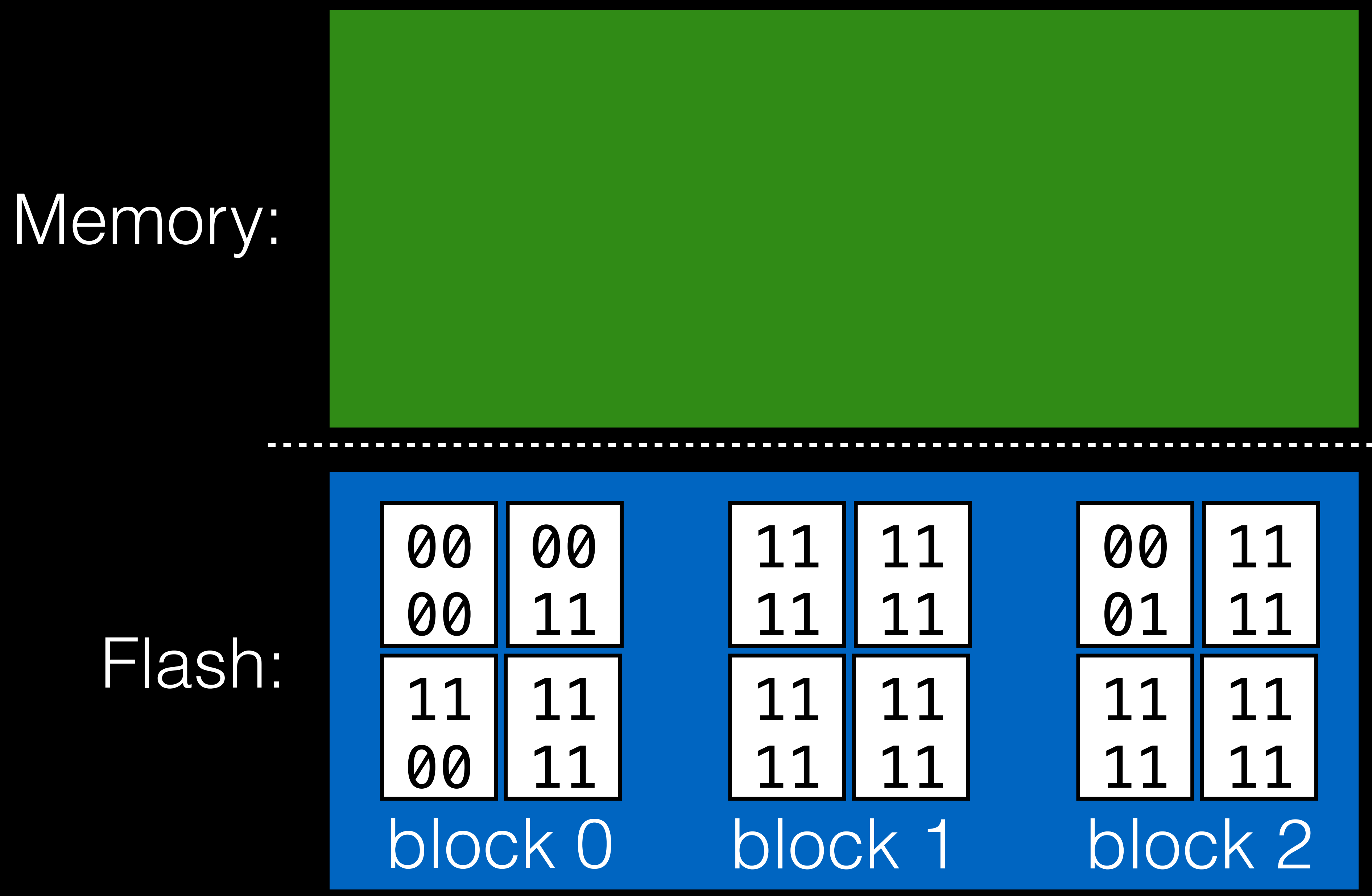
Options

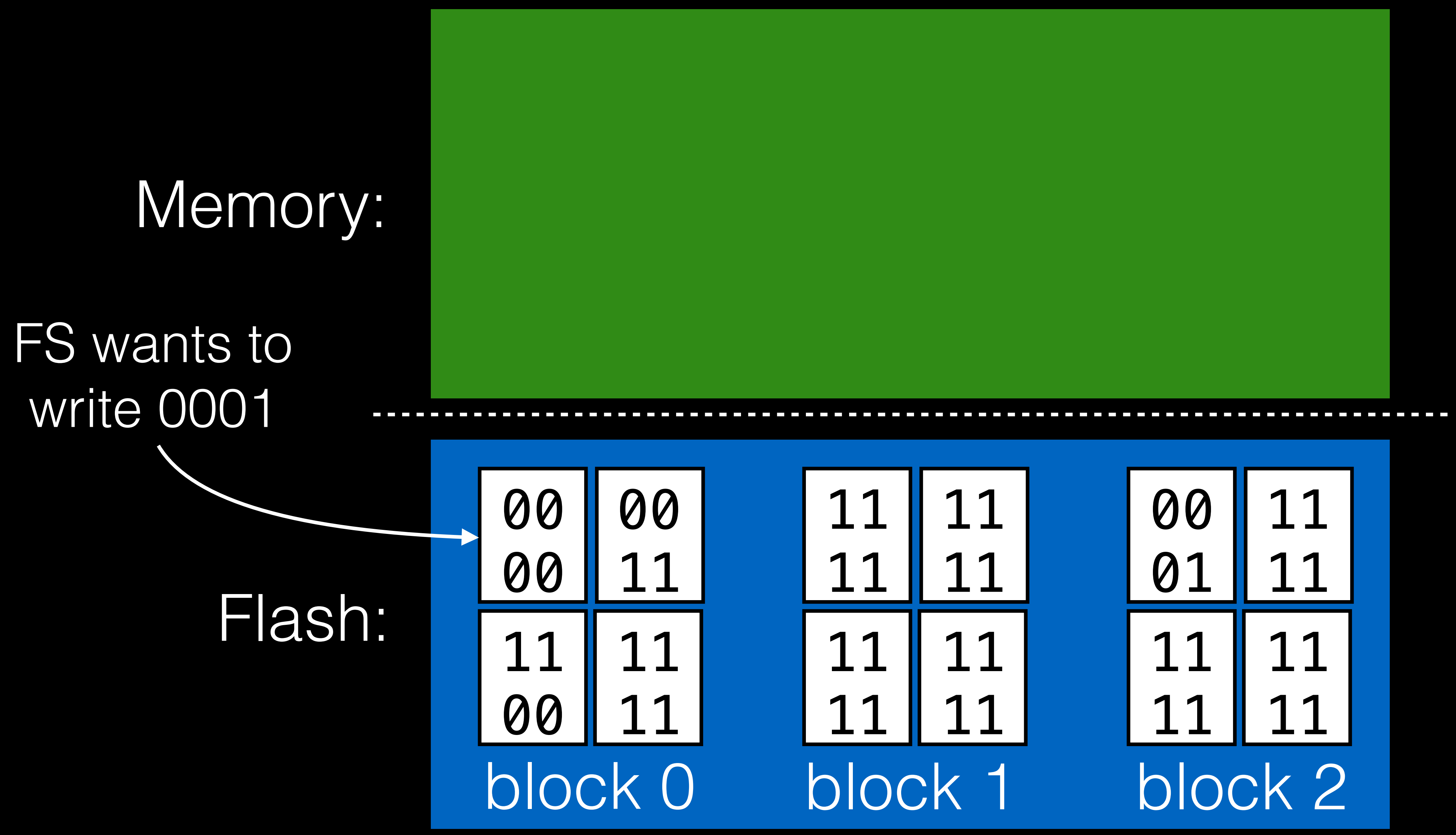
1. Build/use new file systems for flash
 - JFFS, YAFFS
 - lot of work!

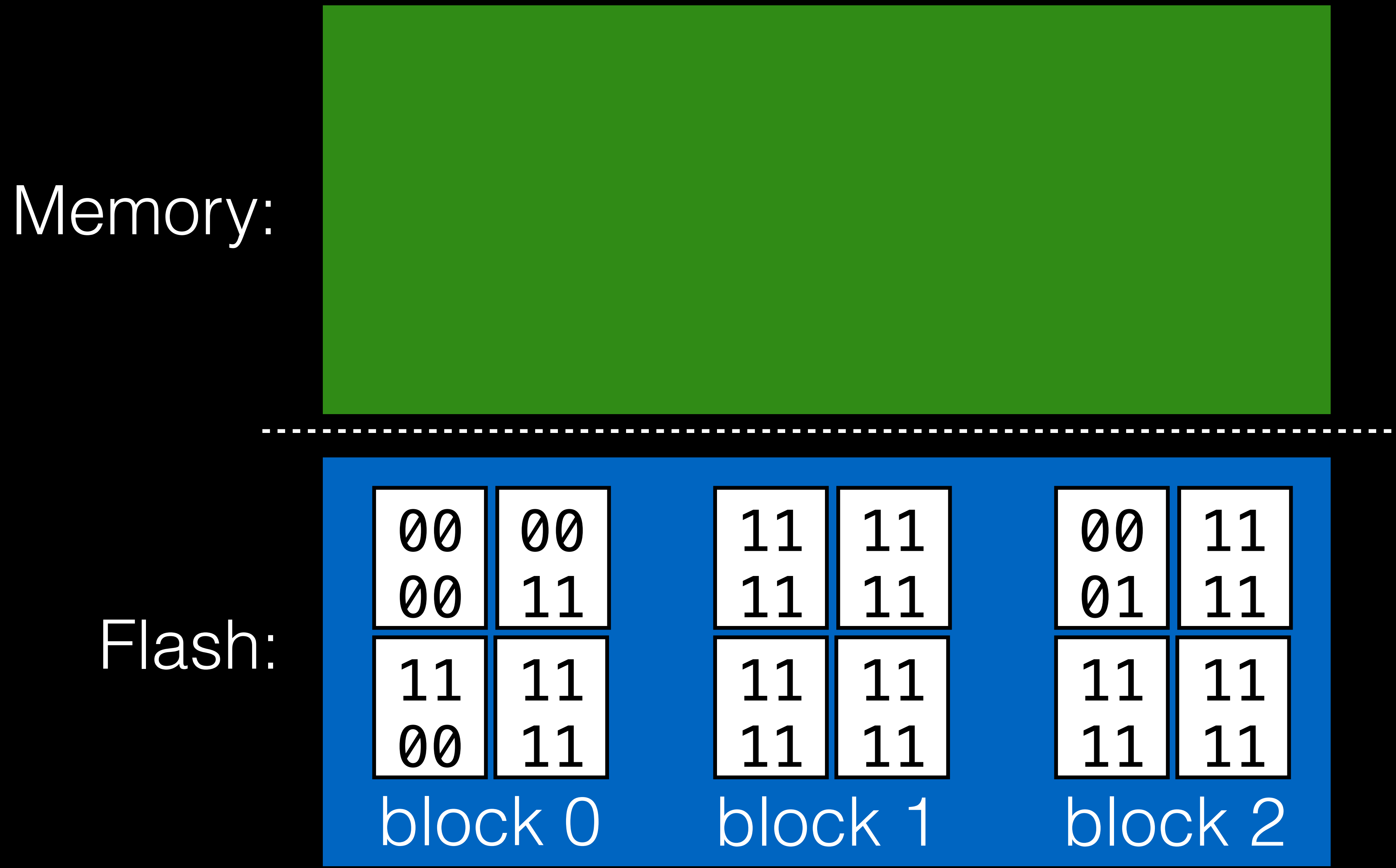
2. Build traditional API over flash API.
 - use FFS, LFS, whatever we want

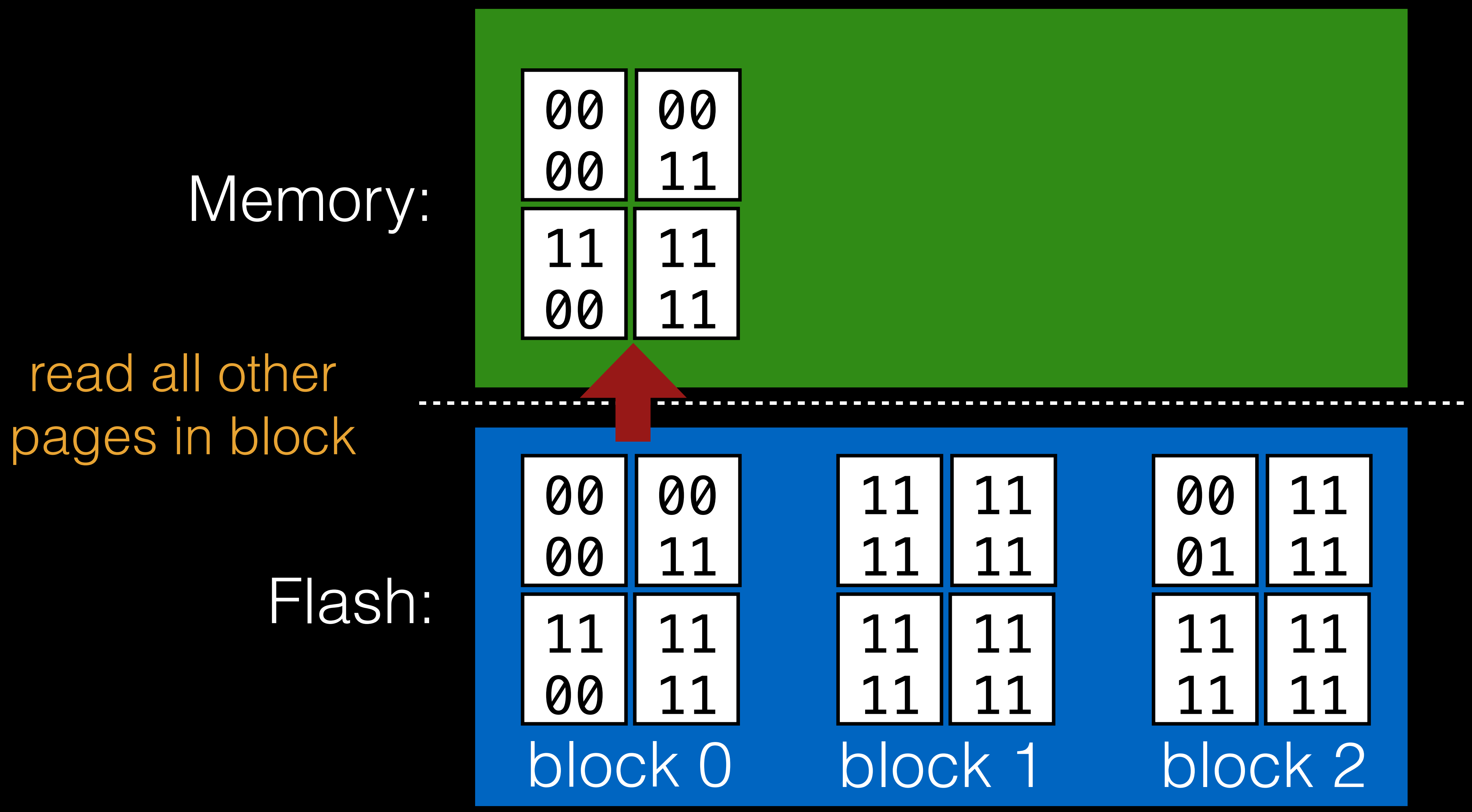
Traditional API with Flash

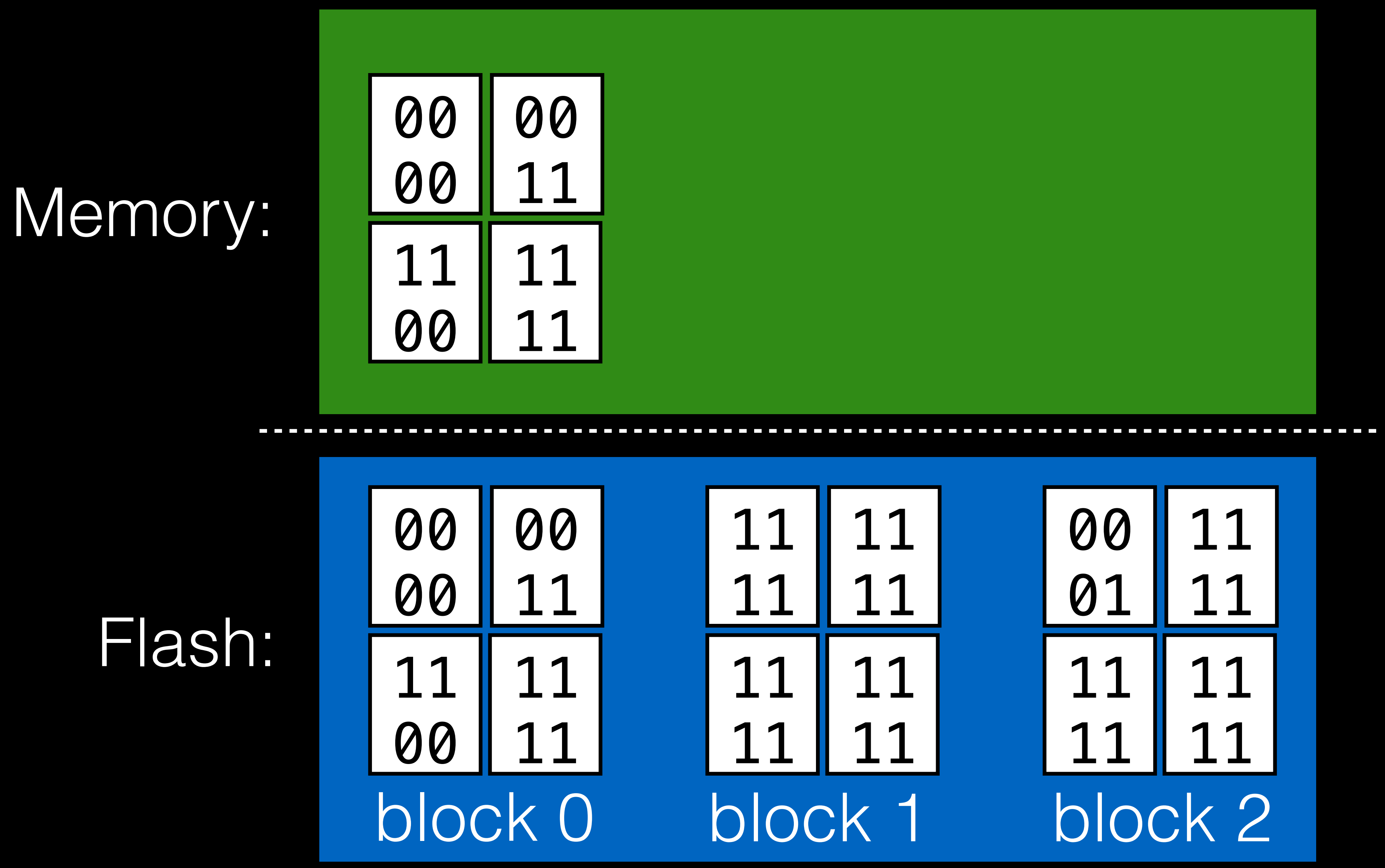
```
read(addr):
    return flash_read(addr)

write(addr, data):
    block_copy = flash_read(block of addr)
    modify block_copy with data
    flash_erase(block of addr)
    flash_program(block of addr, block_copy)
```

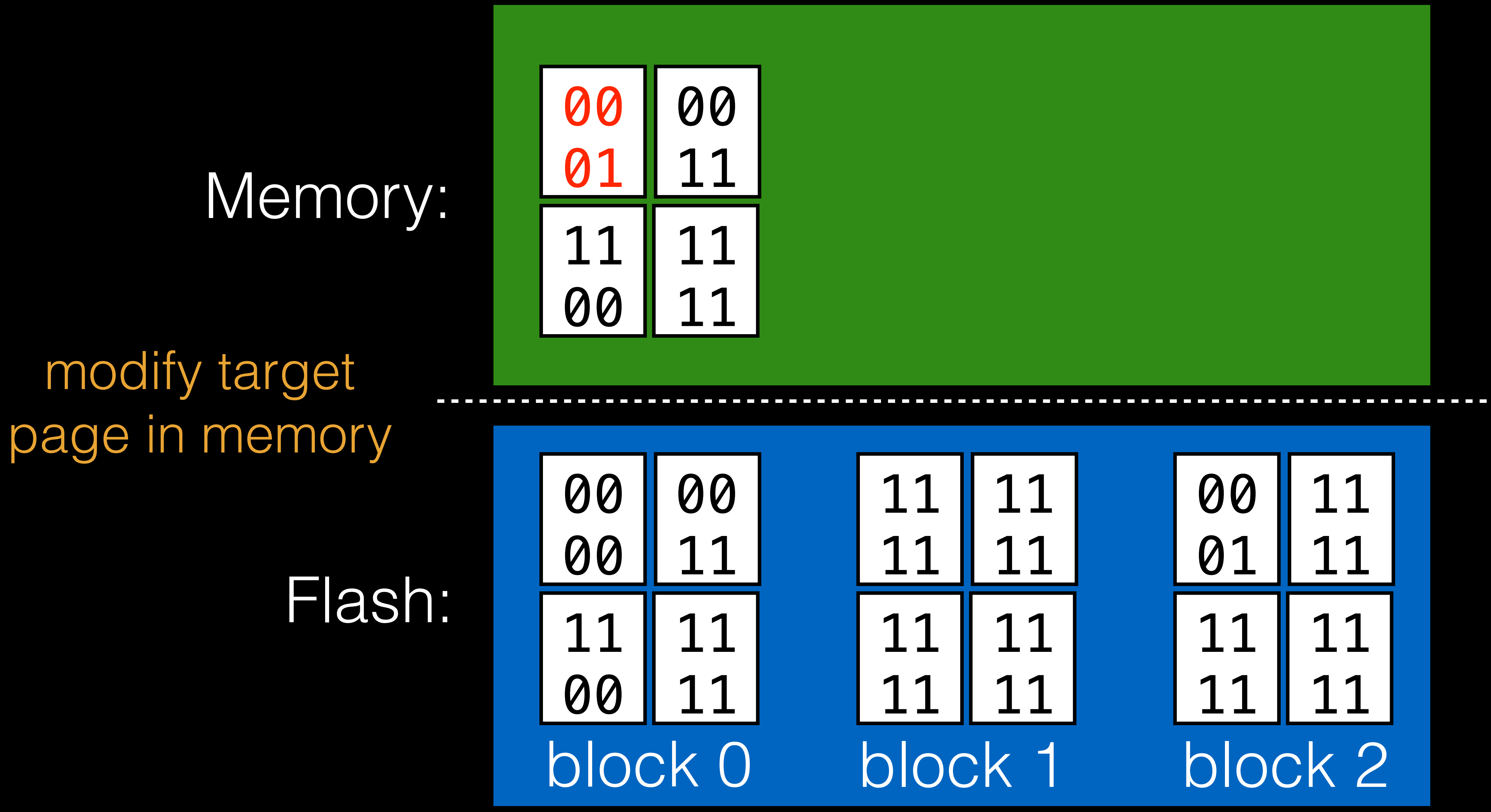

Memory:
Flash:
00
00
00
11
11
00
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2

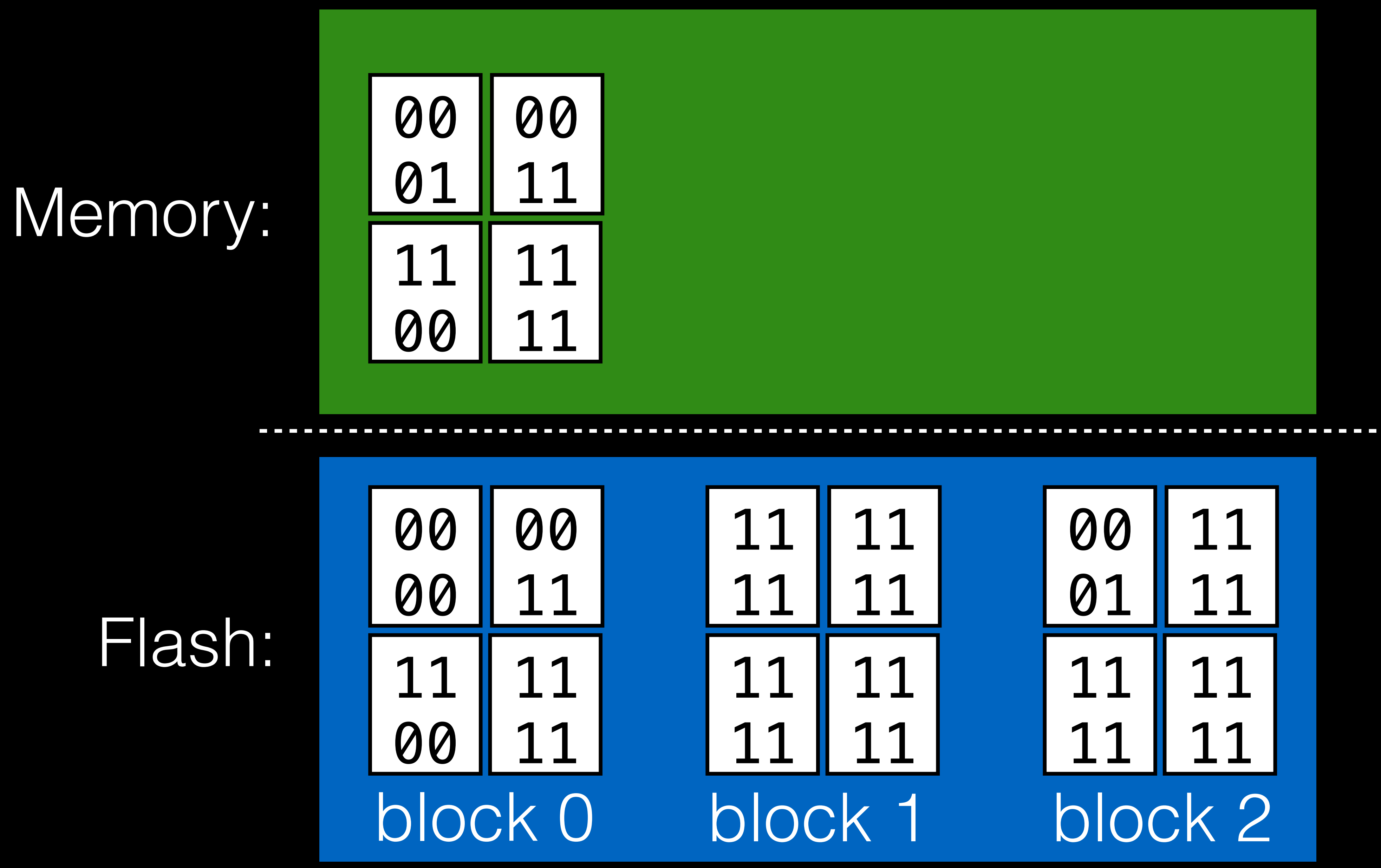

Memory:
FS wants to
write 0001
Flash:
00
00
00
11
11
00
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2

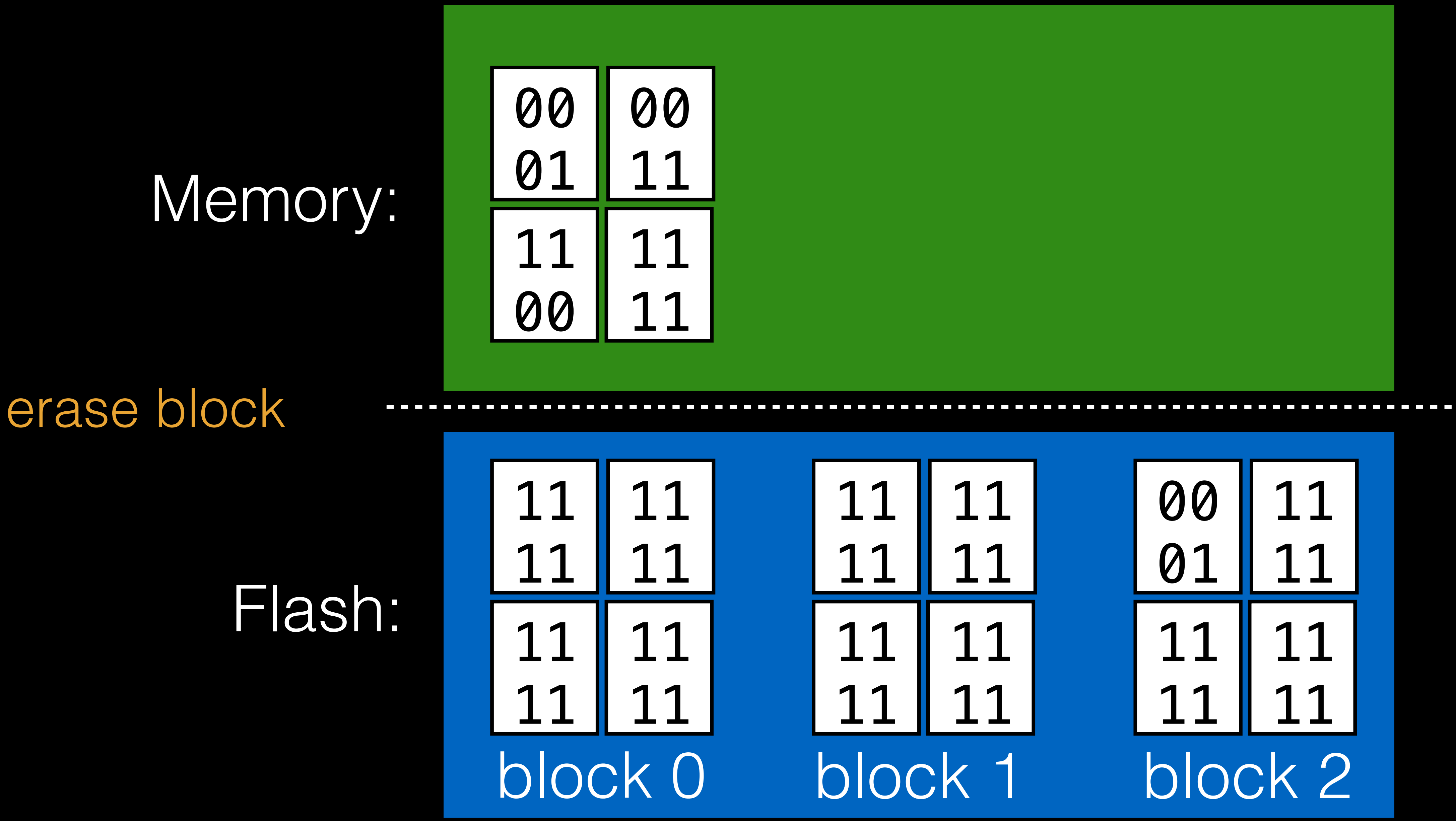

Memory:
Flash:
00 00
00 11
11 00
11 11
block 0
11 11
11 11
11 11
11 11
block 1
00 01
11 11
11 11
11 11
block 2

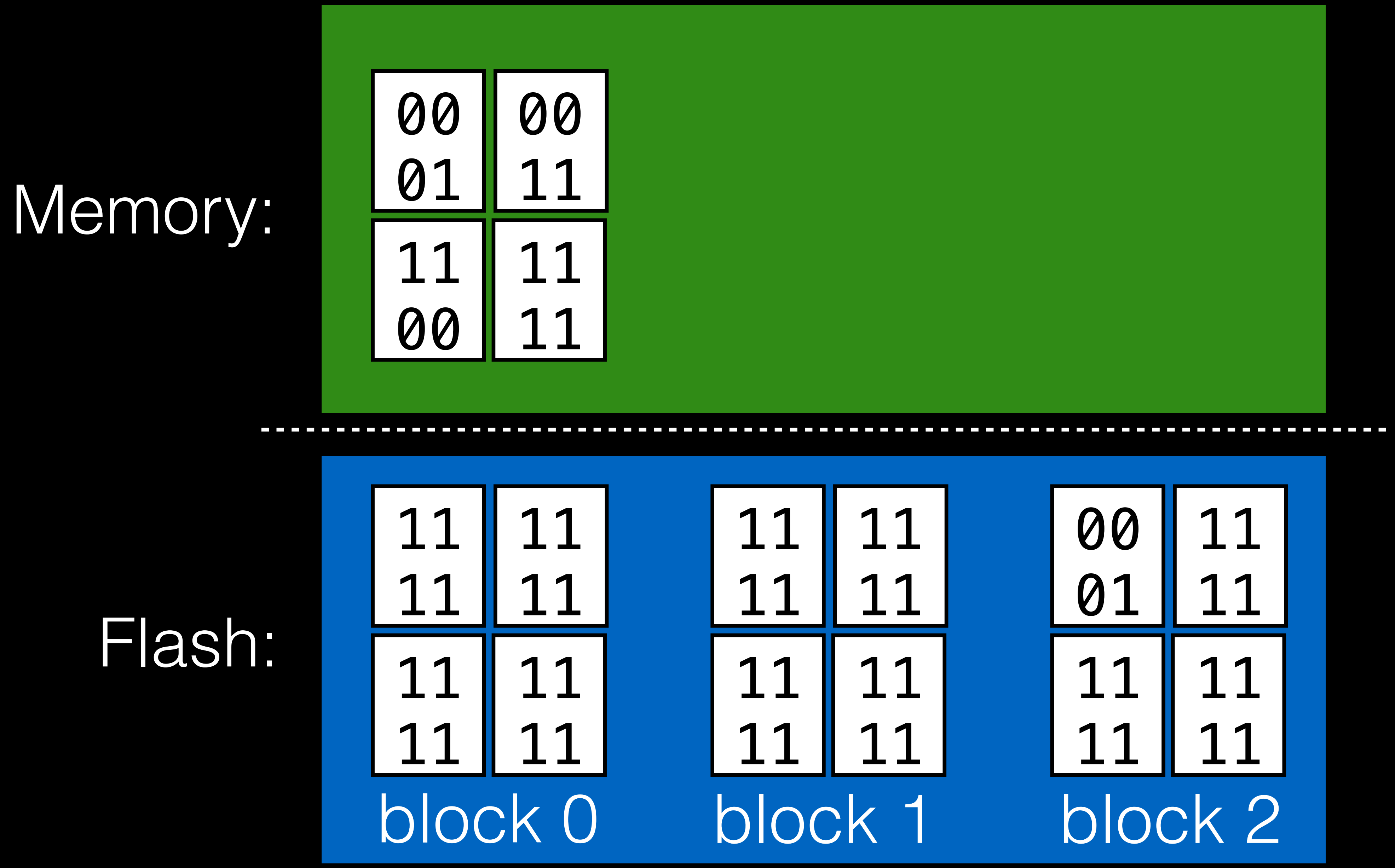

Memory:
00
00
00
11
11
00
11
11
read all other
pages in block
Flash:
00
00
00
11
11
00
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2

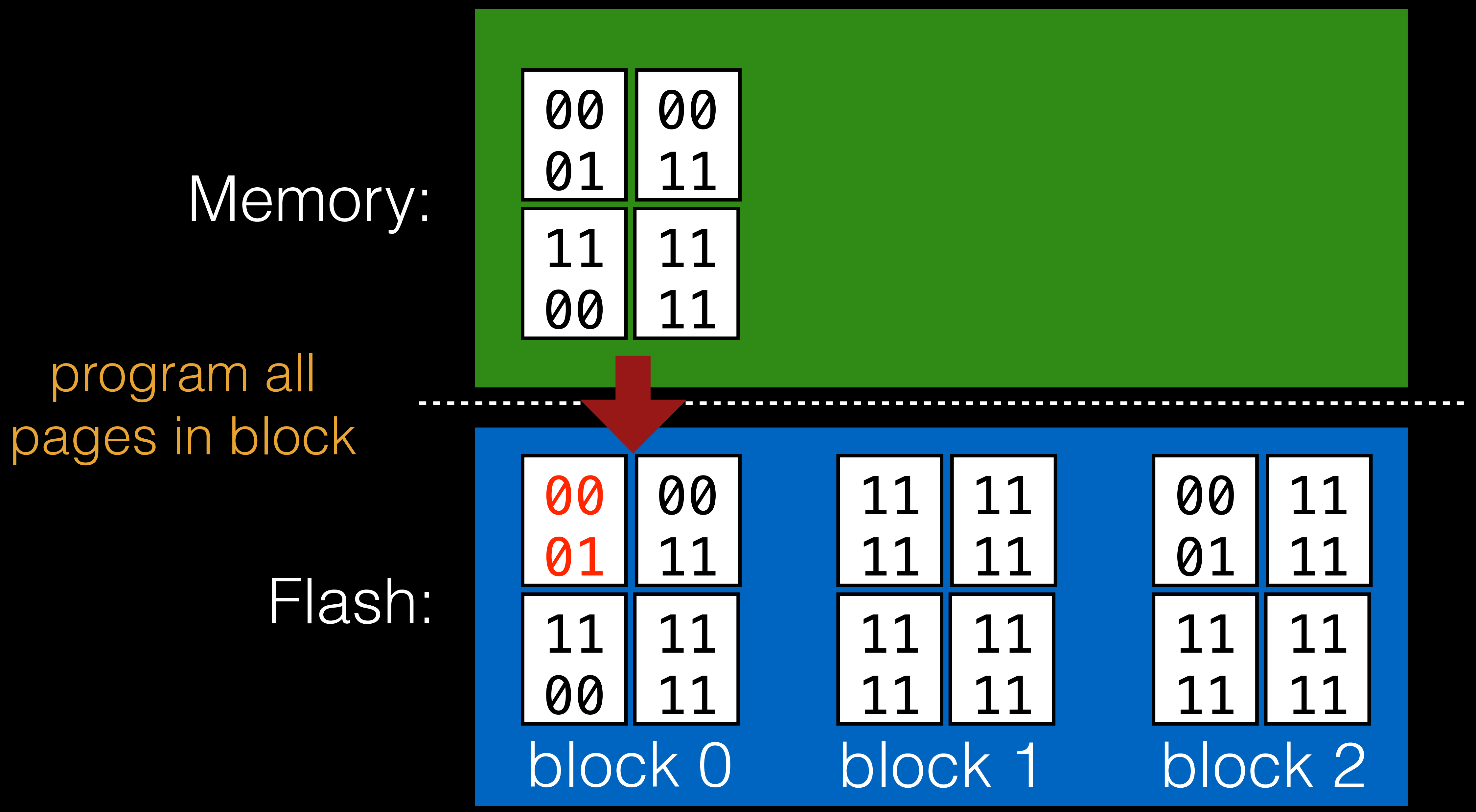

Memory:
00
00
00
11
11
00
11
11
Flash:
00
00
00
11
11
00
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2

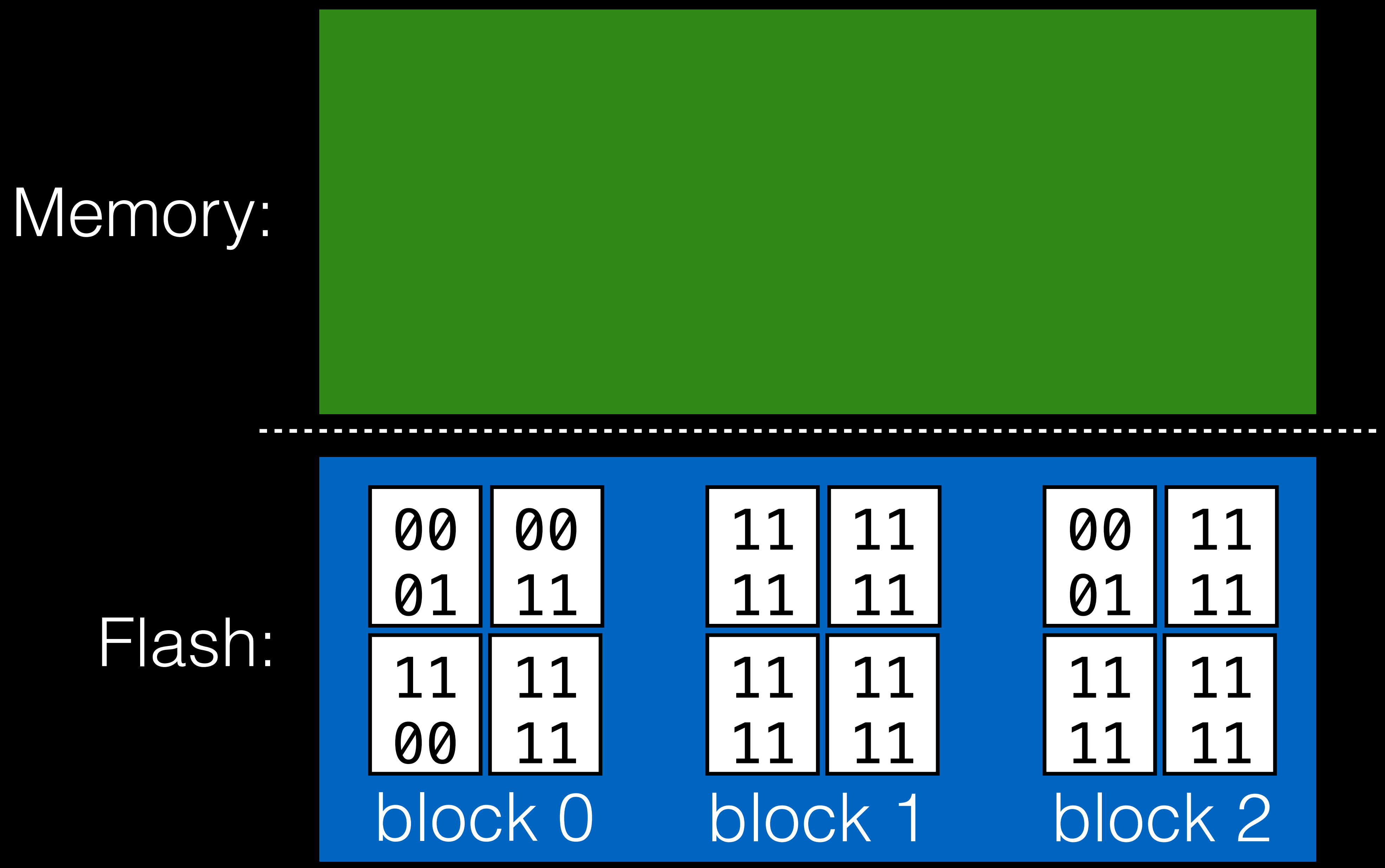

Memory:
00
01
00
11
11
00
11
11
modify target
page in memory
Flash:
00
00
00
11
11
00
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2


Memory:
00 01
00 11
11 00
11 11
Flash:
00 00
00 11
11 00
11 11
11 11
11 11
11 11
11 11
00 01
11 11
11 11
11 11
block 0
block 1
block 2


Memory:
00
01
00
11
11
00
11
11
erase block
Flash:
11
11
11
11
11
11
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2


Memory:
00
01
00
11
11
00
11
11
Flash:
11
11
11
11
11
11
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2


Memory:
00
01
00
11
11
00
11
11
program all
pages in block
Flash:
00
01
00
11
11
00
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2


Memory:
Flash:
00
01
00
11
11
00
11
11
block 0
11
11
11
11
11
11
11
11
block 1
00
01
11
11
11
11
11
11
block 2

Write Amplification

Random writes are extremely expensive!

Writing one 2KB page may cause:
- `read`, `erase`, and `program` of 256KB block.

Write Amplification

Random writes are extremely expensive!

Writing one 2KB page may cause:
- `read`, `erase`, and `program` of 256KB block.

Would FFS or LFS be better with flash?

File Systems over Flash

Copy-On-Write FS may prevent some expensive random writes.

What about wear leveling? LFS won't do this.

What if we want to use some other FS?

Perhaps some other FS has features or APIs our applications rely on, so we must use it

Better Solution

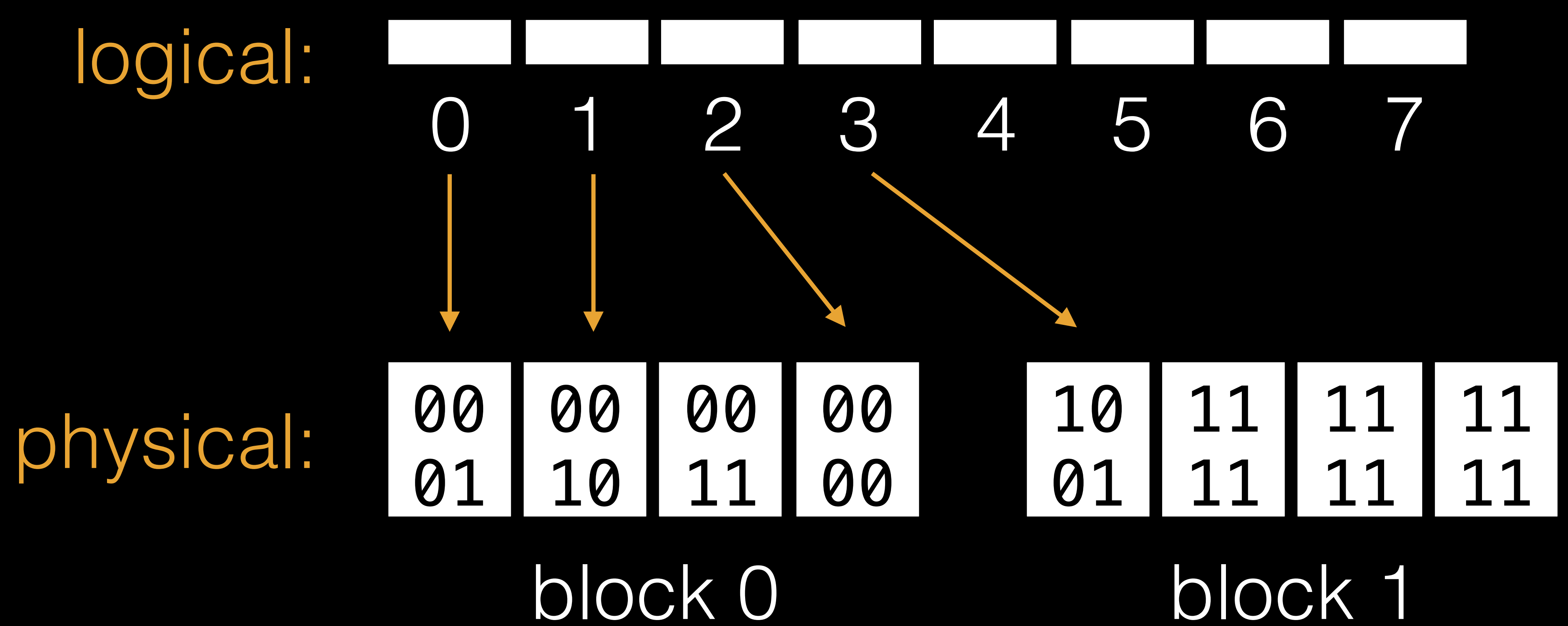
Add copy-on-write layer between FS and flash.
Avoids RMW (read-modify-write) cycle.

Translate logical device addrs to physical addrs.

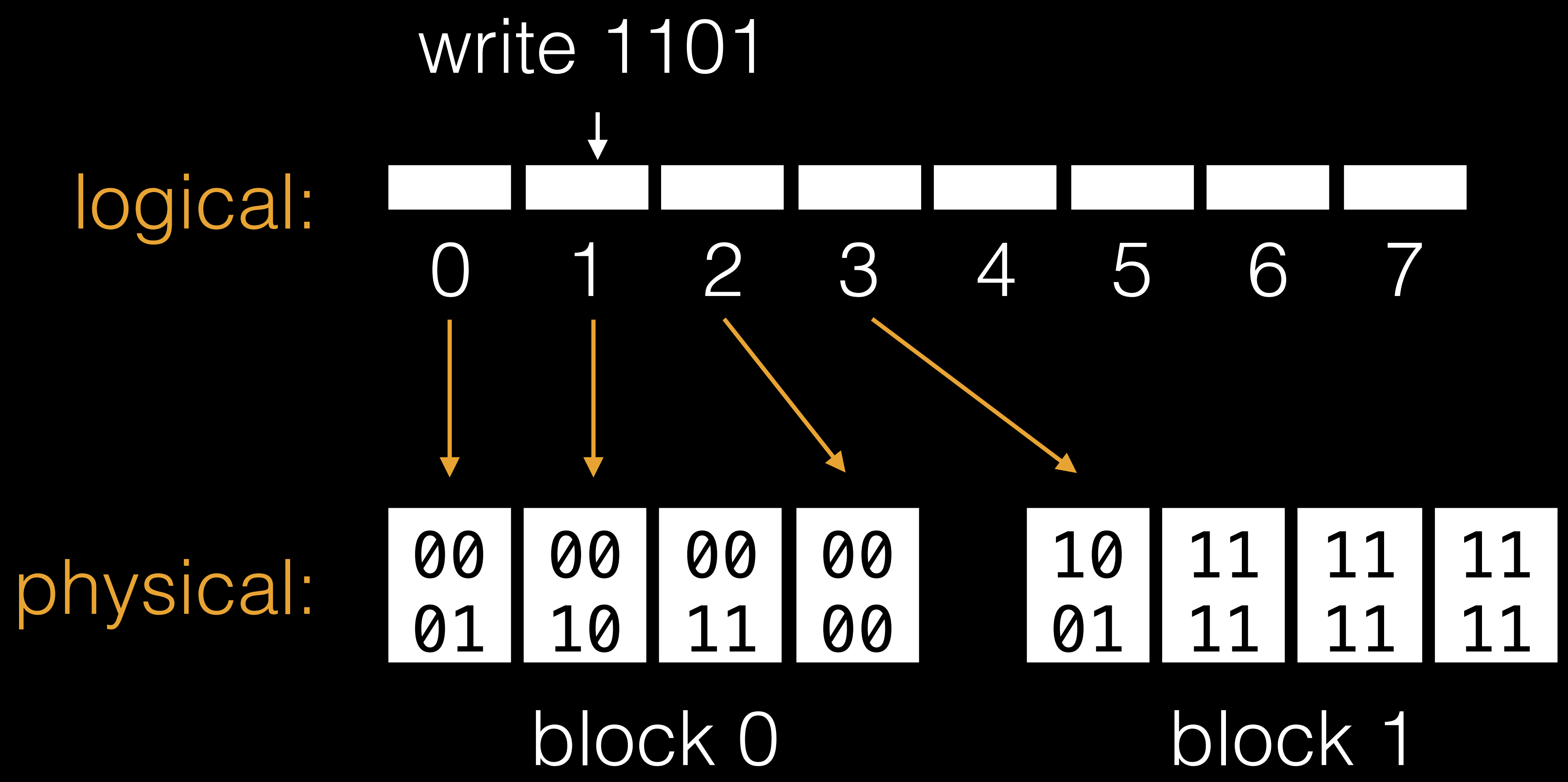
FTL: Flash Translation Layer.

Should translation use math or data structure?

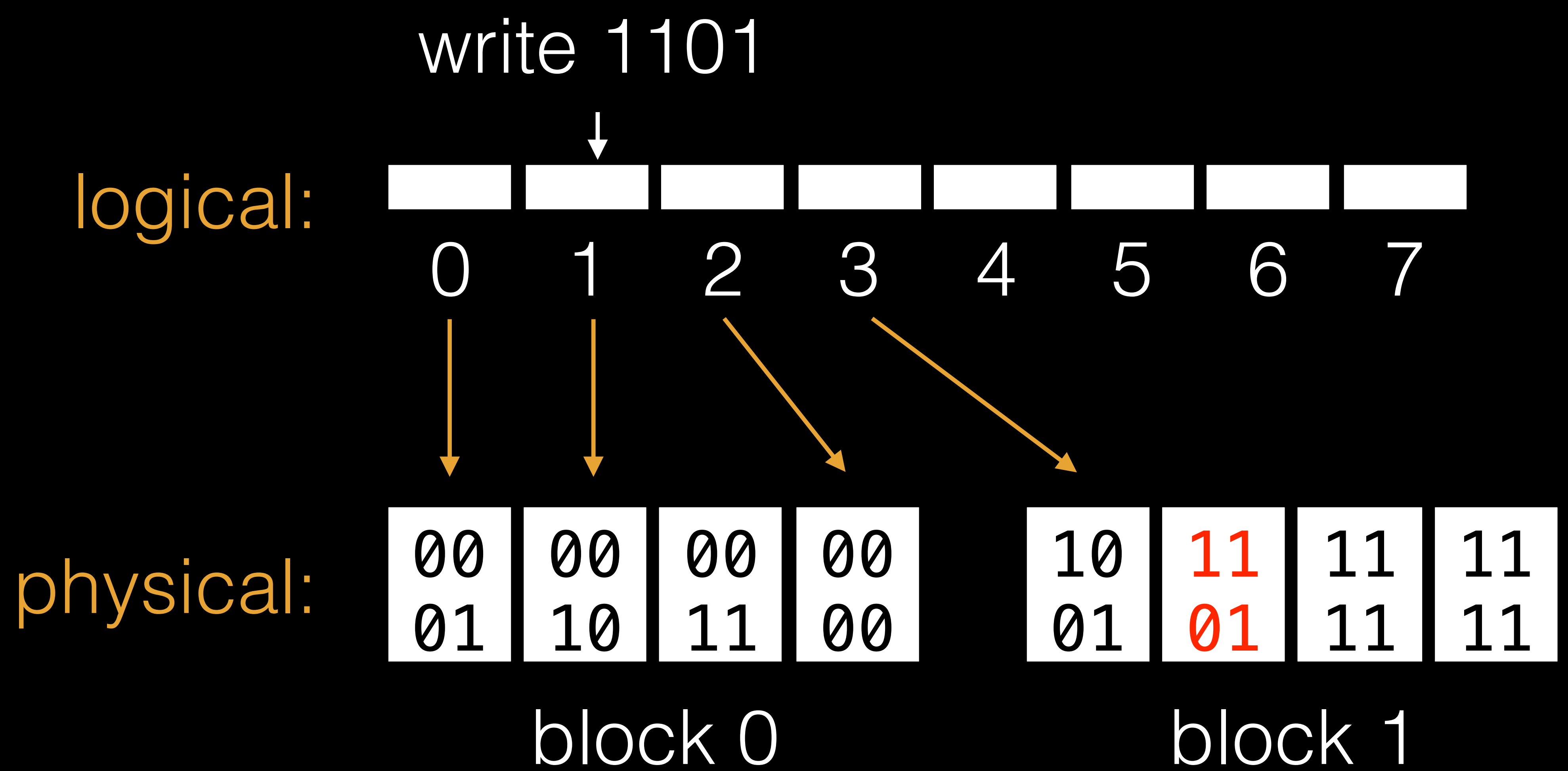
Flash Translation Layer



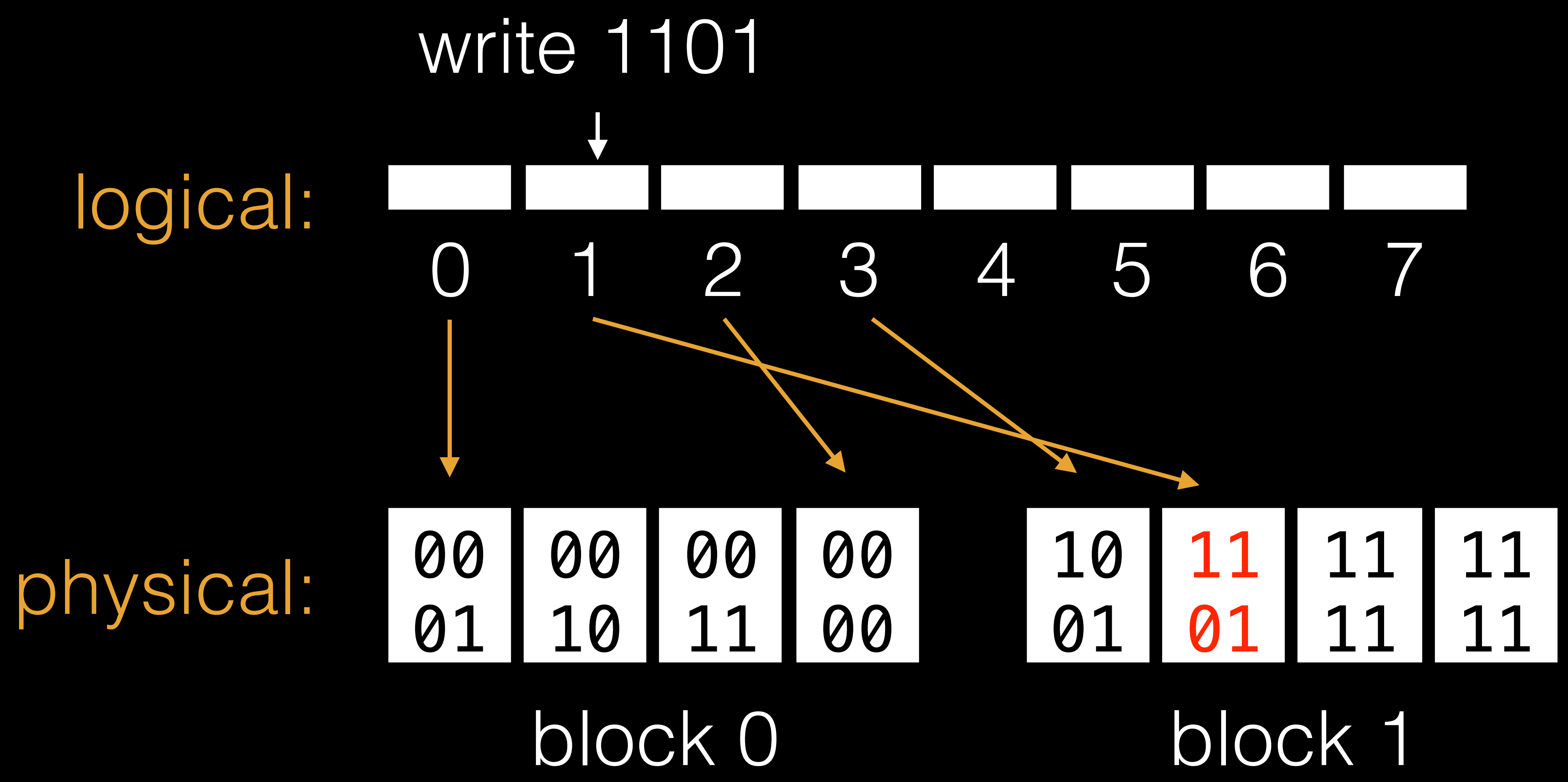
Flash Translation Layer



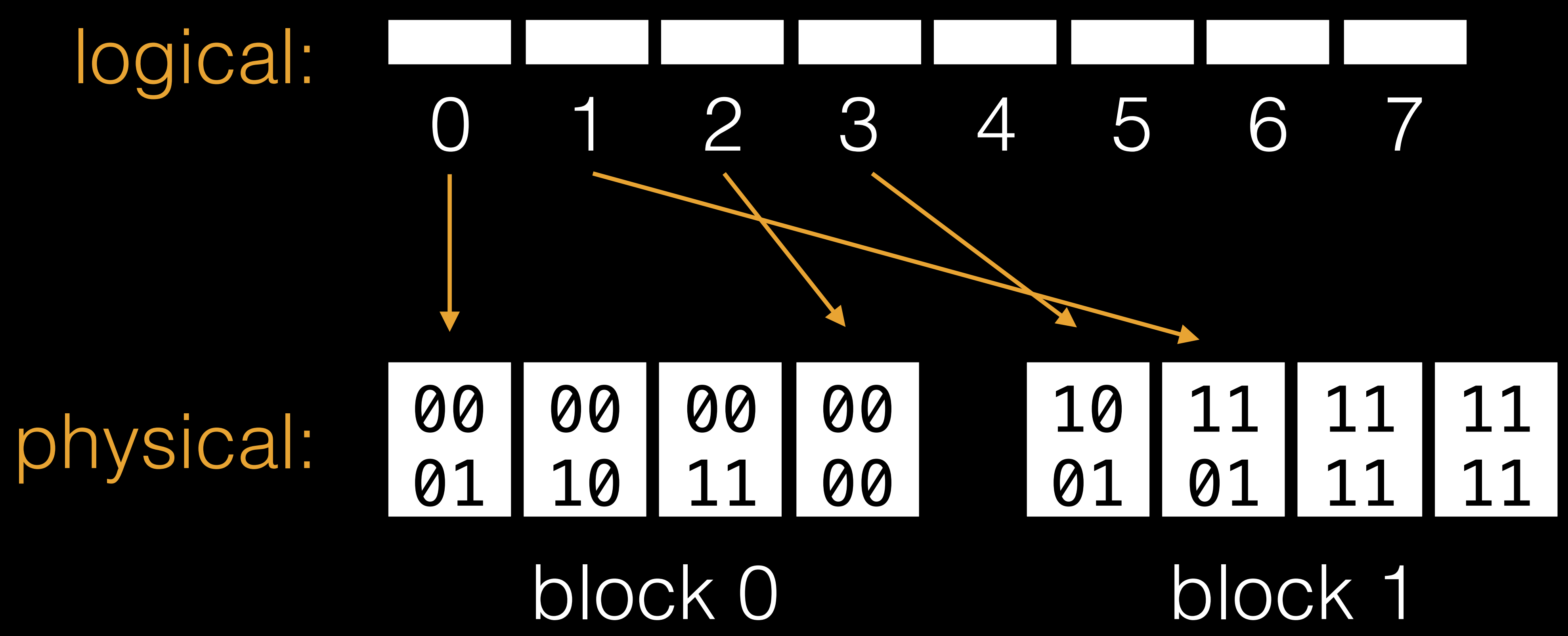
Flash Translation Layer



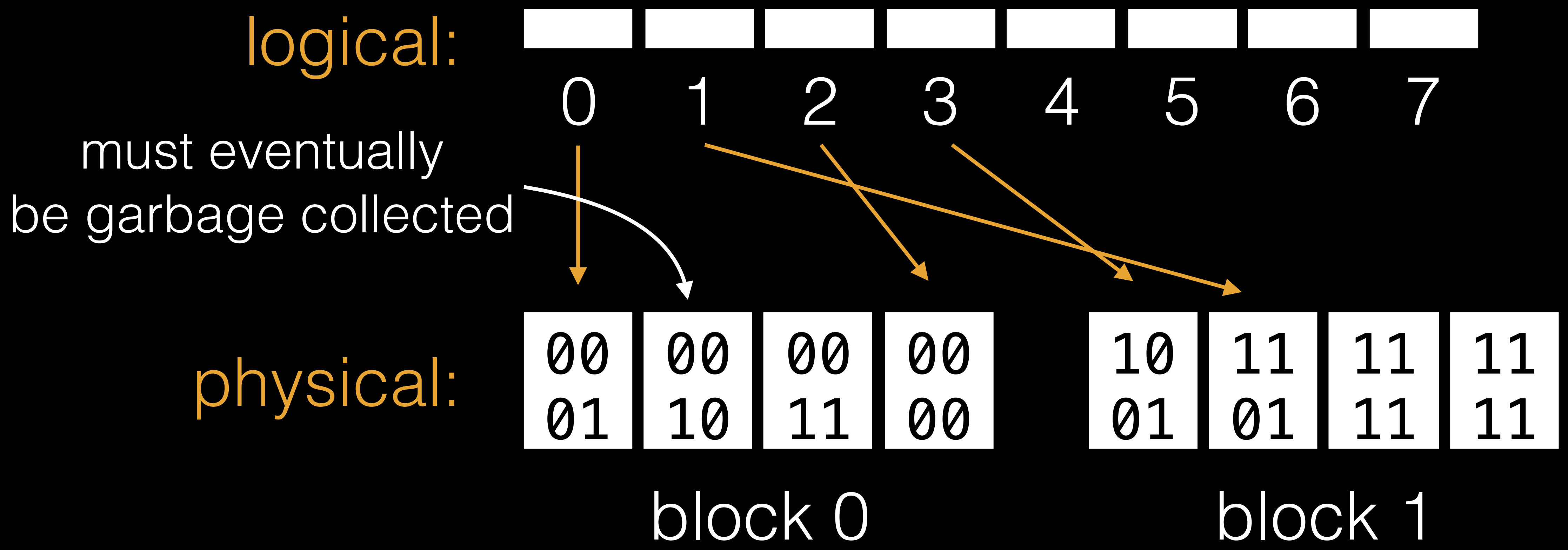
Flash Translation Layer



Flash Translation Layer



Flash Translation Layer



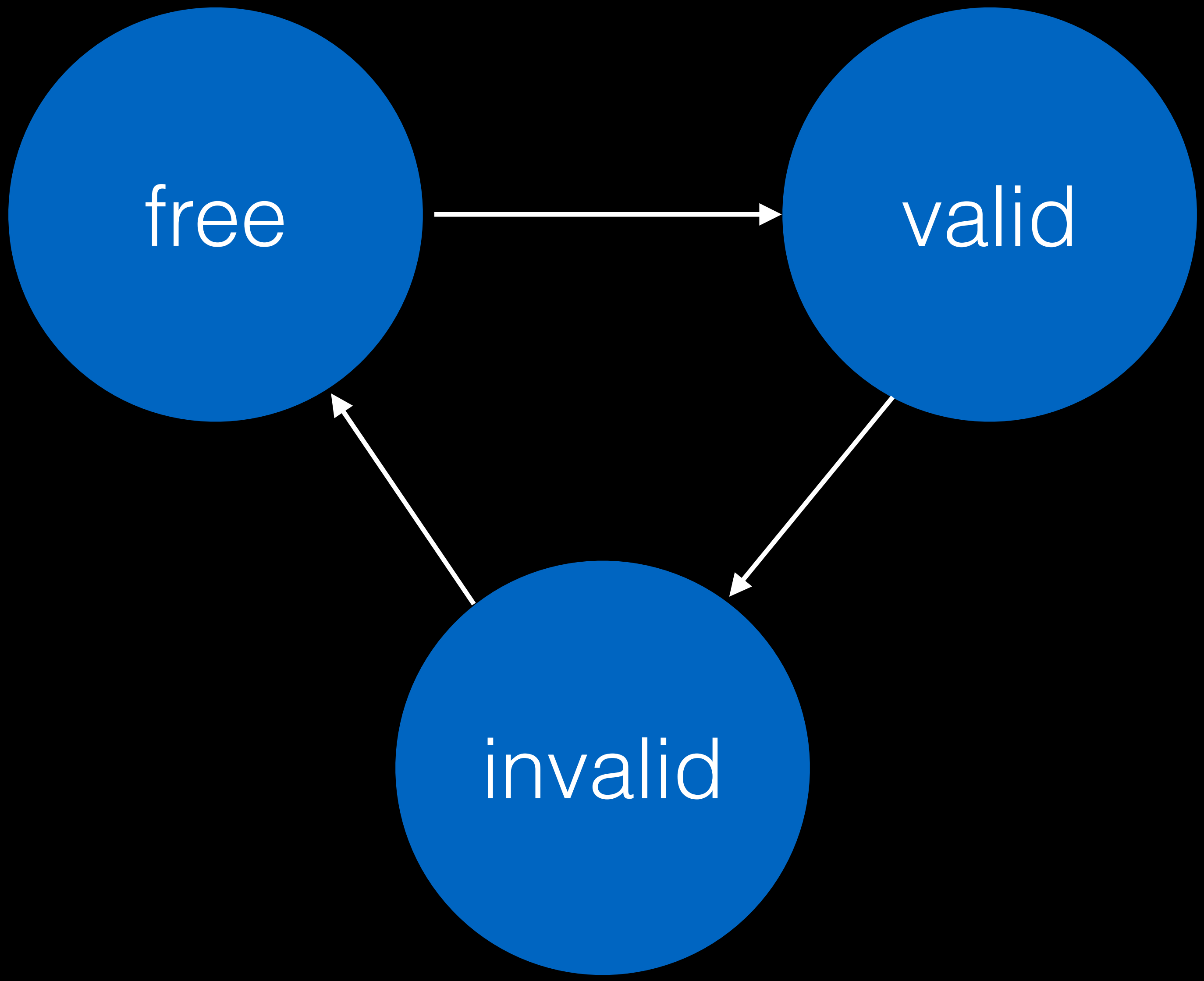
FTL

Could be implemented as device driver or in firmware (usually the latter).

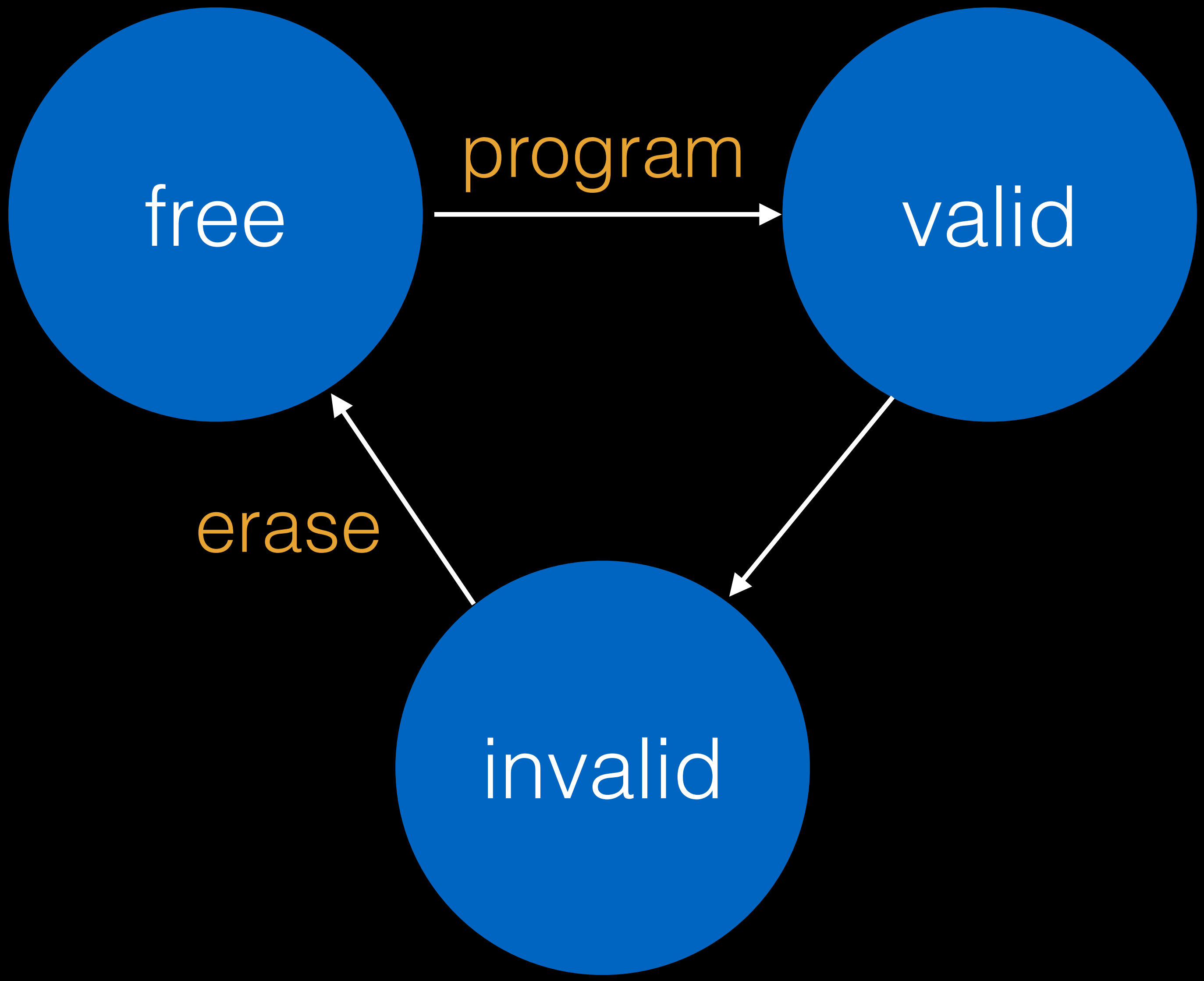
Where to store mapping? SRAM.

Physical pages can be in three states:
- valid, invalid, free

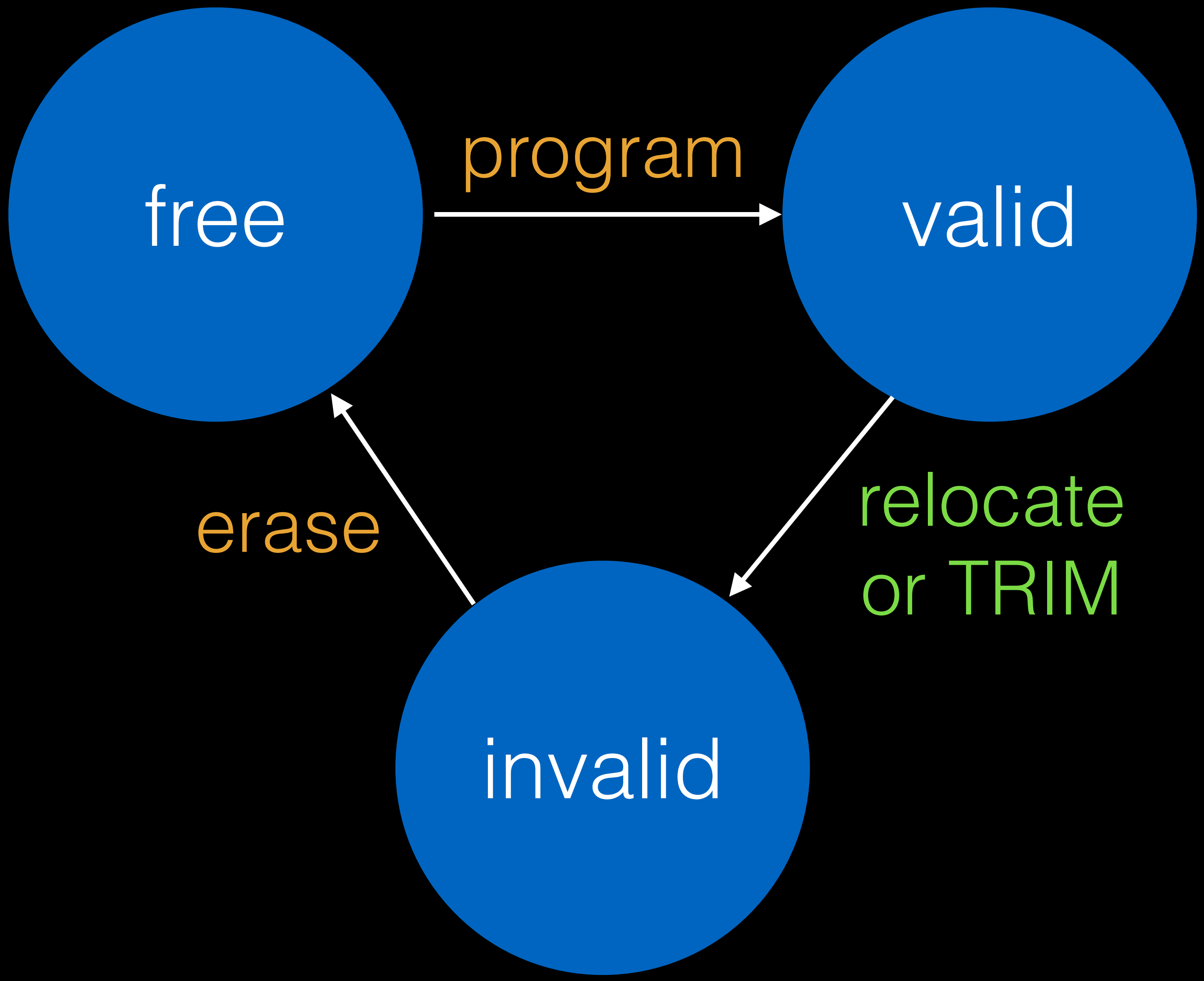
States



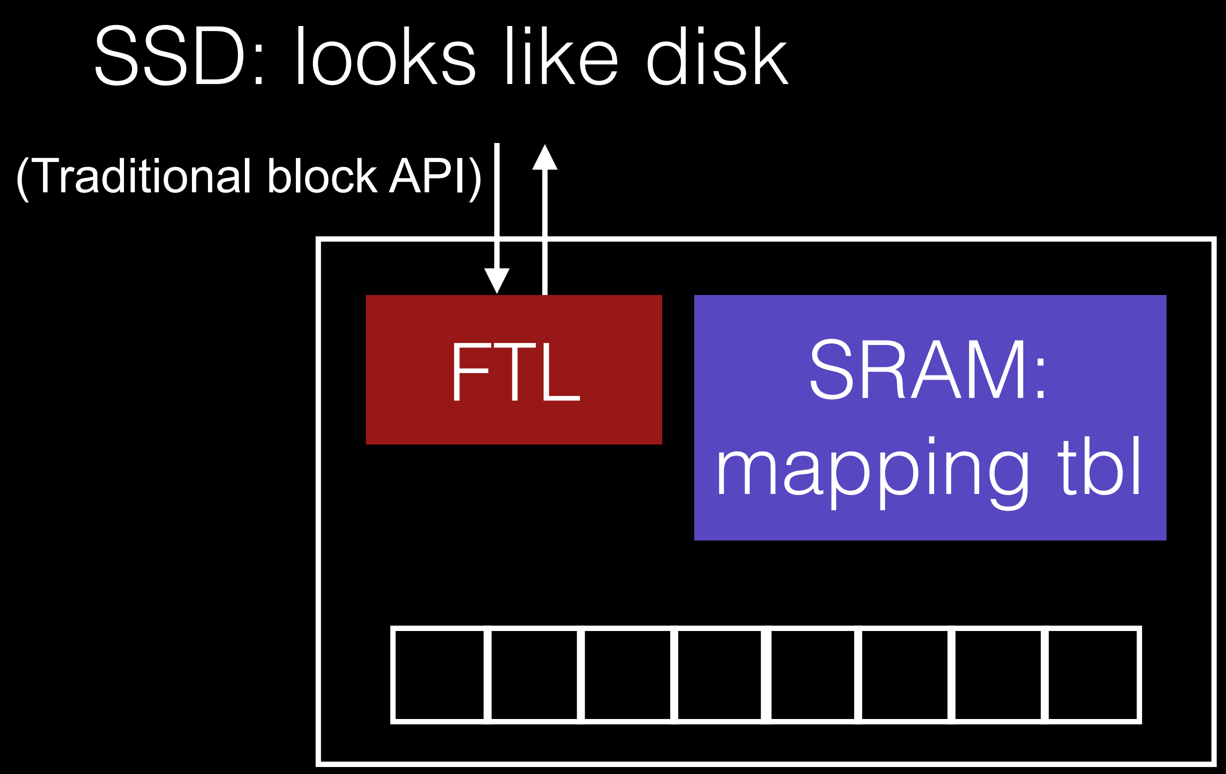
States



States



SSD Architecture



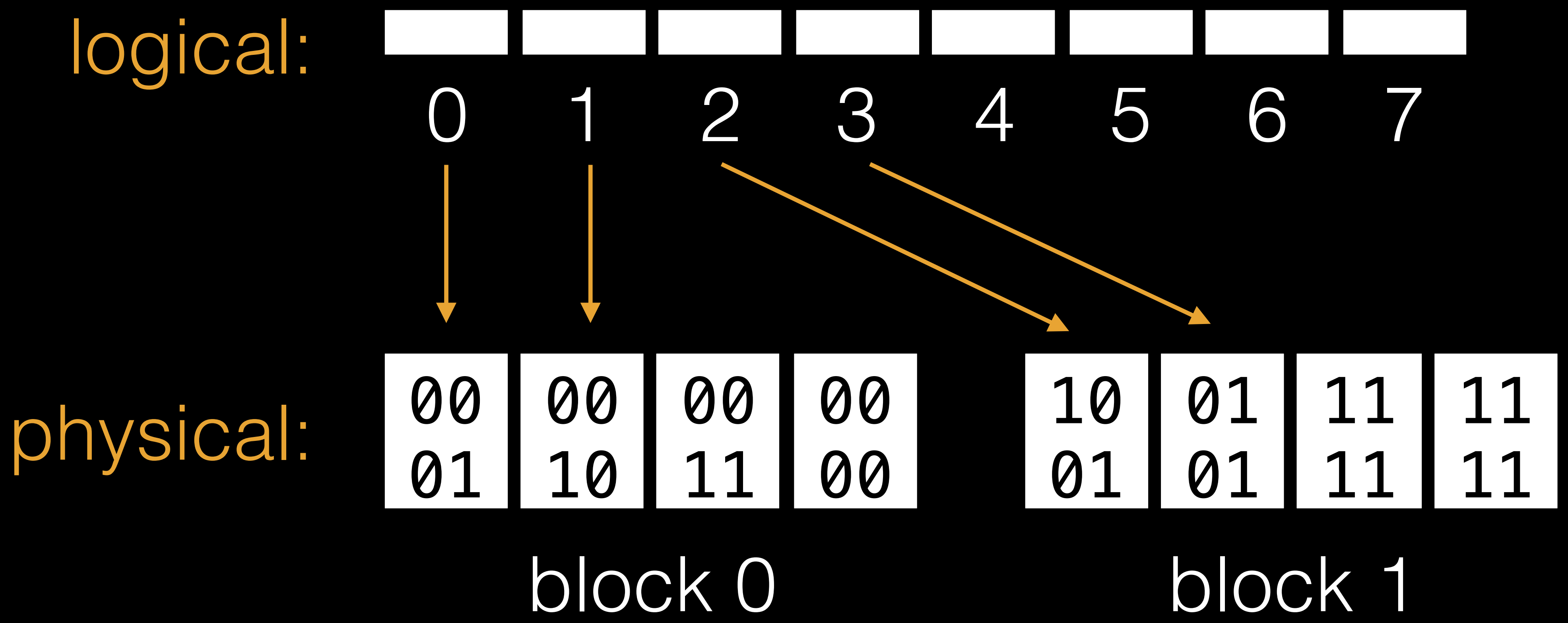
Problem: Big Mapping Table

Assume 200GB device, 2KB pages, 4-byte entries.

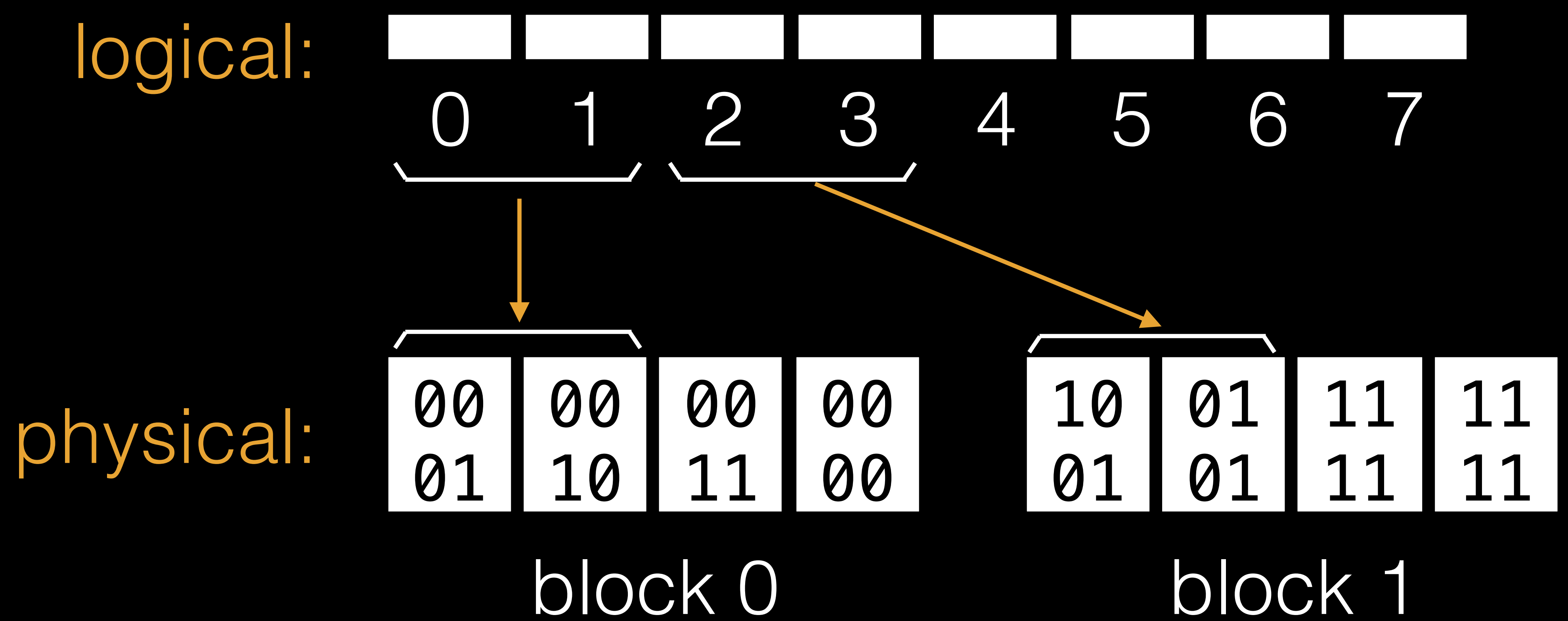
SRAM needed: (200GB / 2KB) * 4 bytes = 400 MB.

Too big, SRAM is expensive!

Page Translations



2-Page Translations

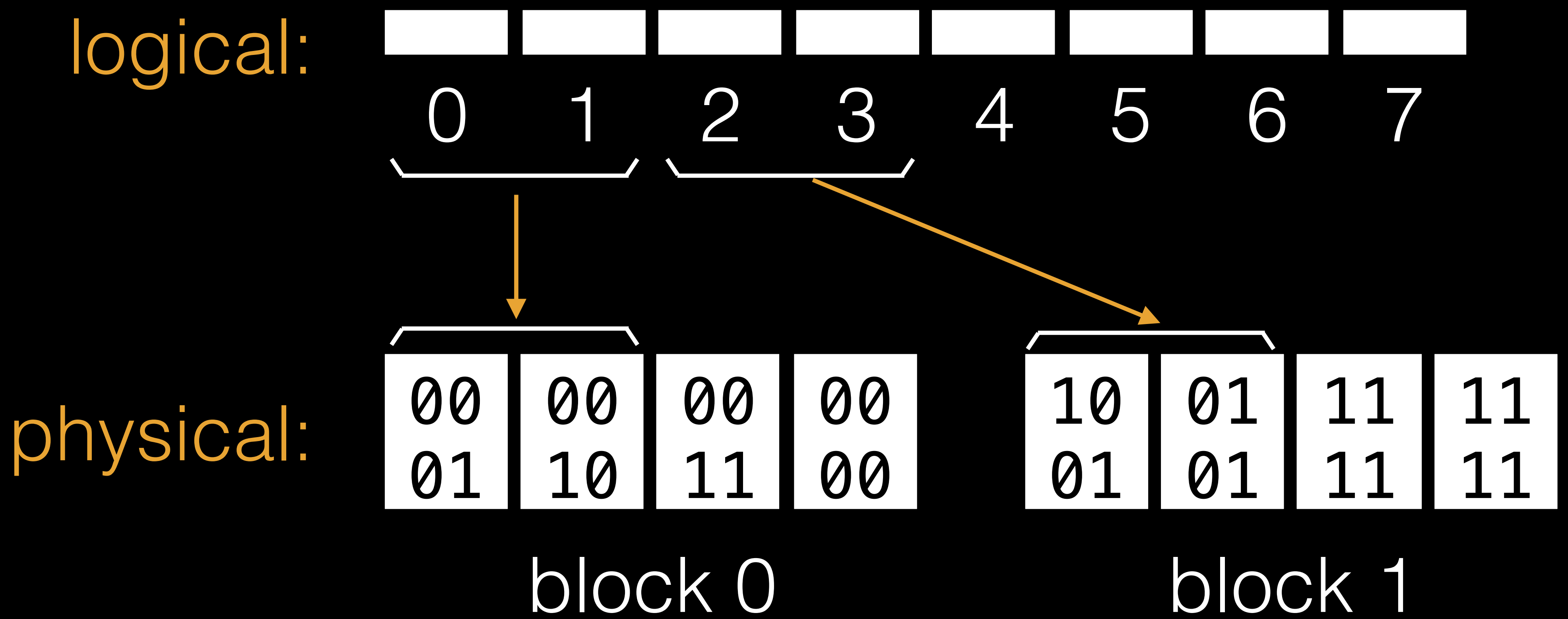


Larger Mappings

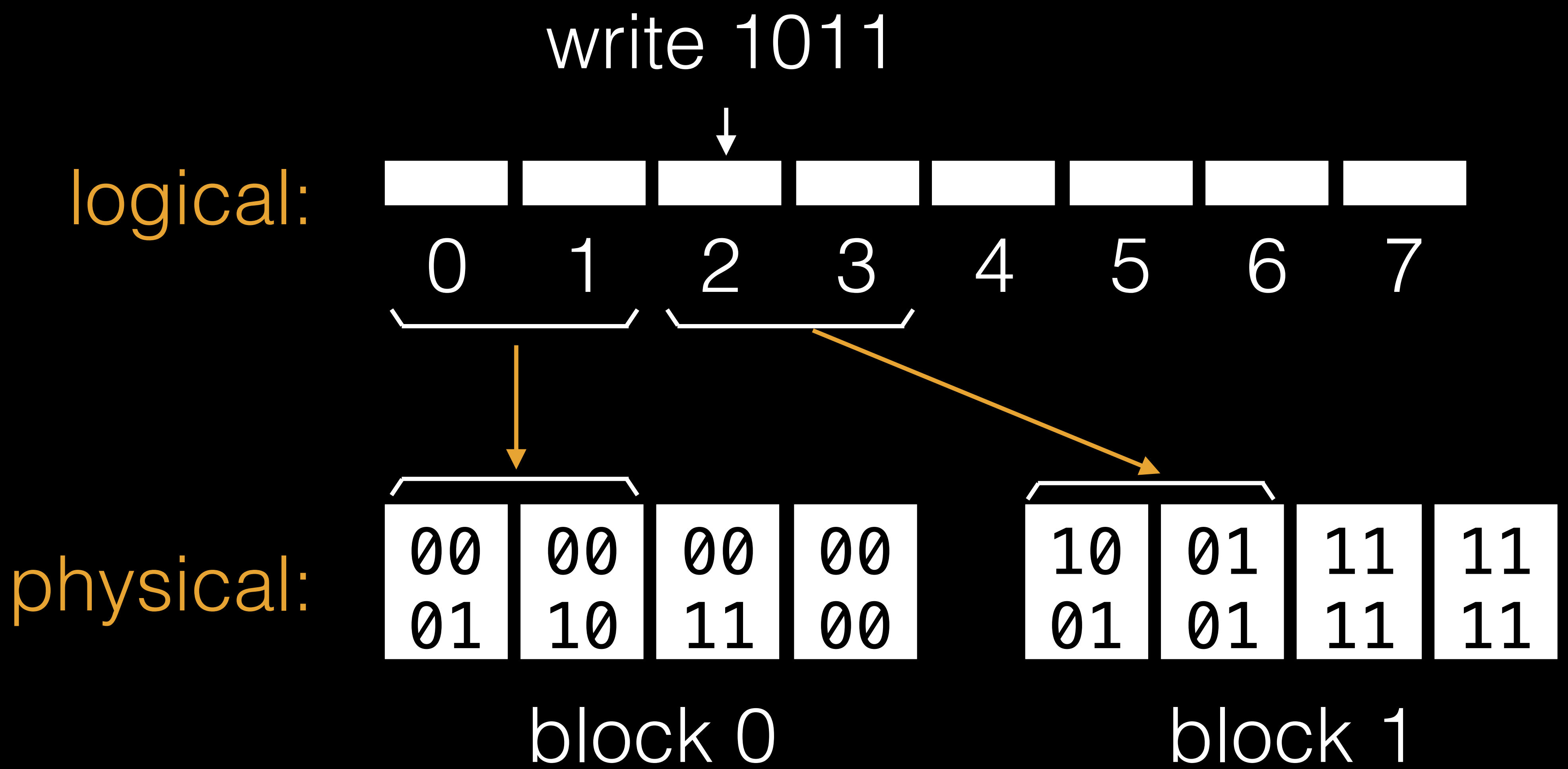
Advantage: larger mappings decrease table size.

Disadvantage?

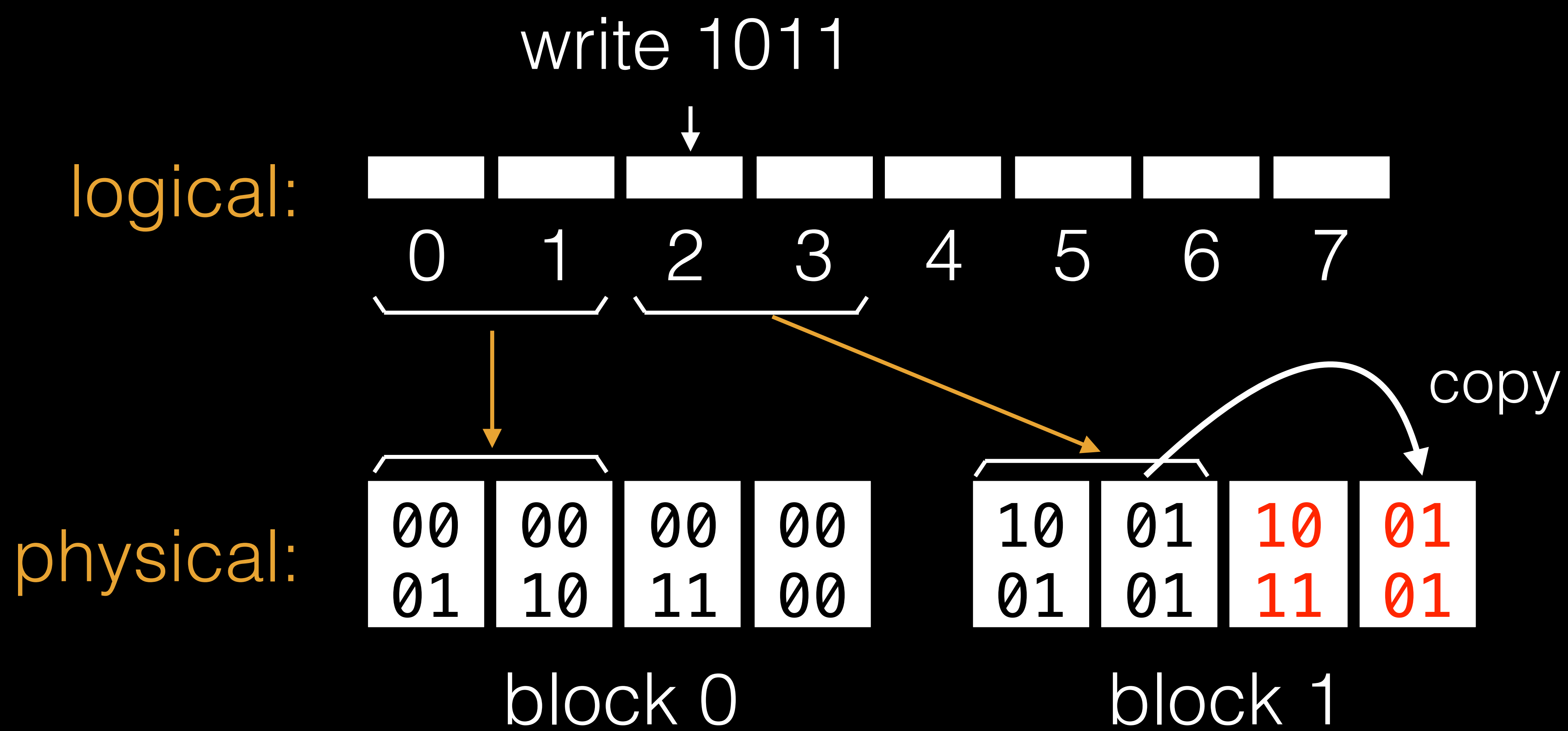
2-Page Translations



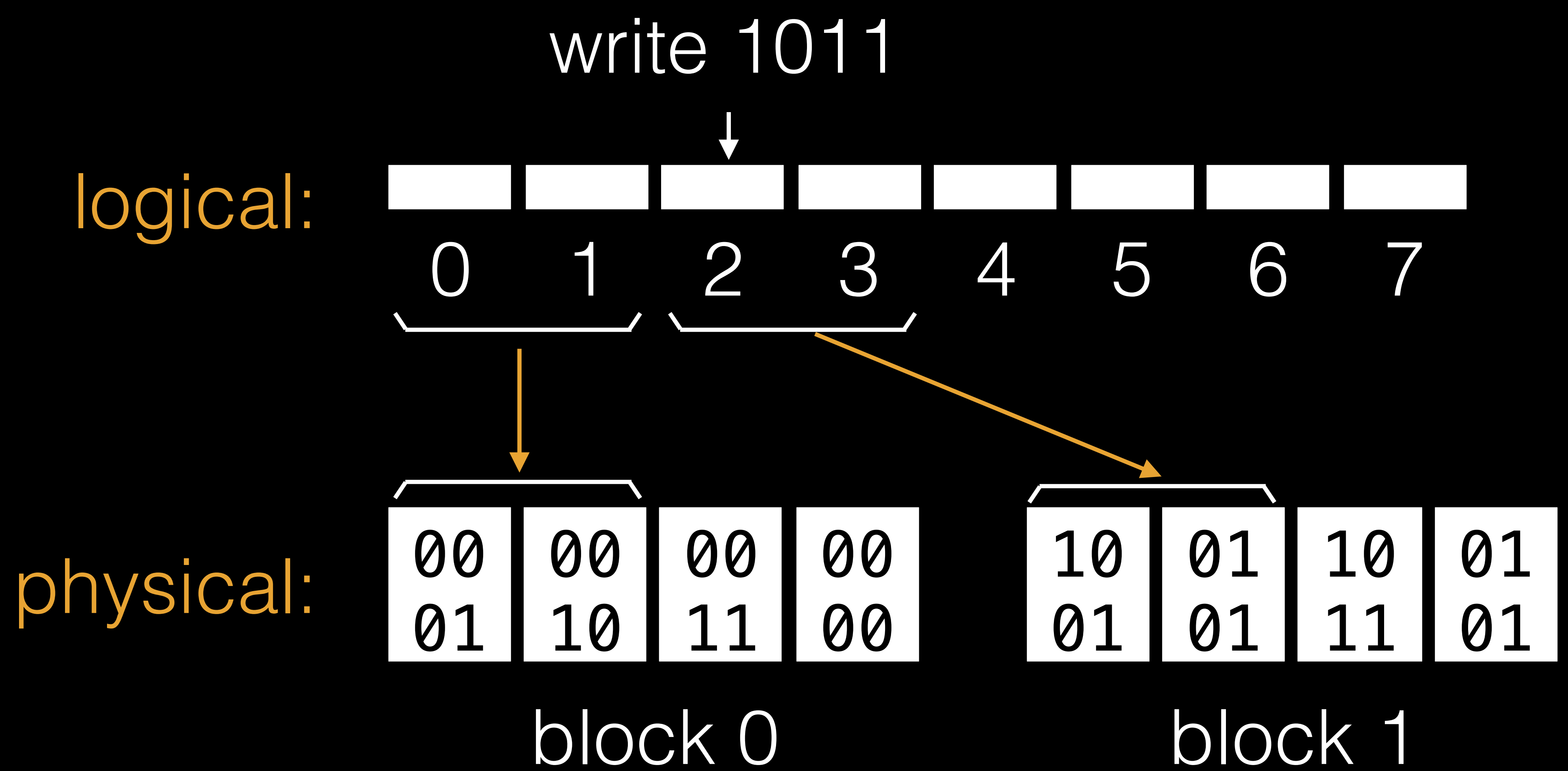
2-Page Translations



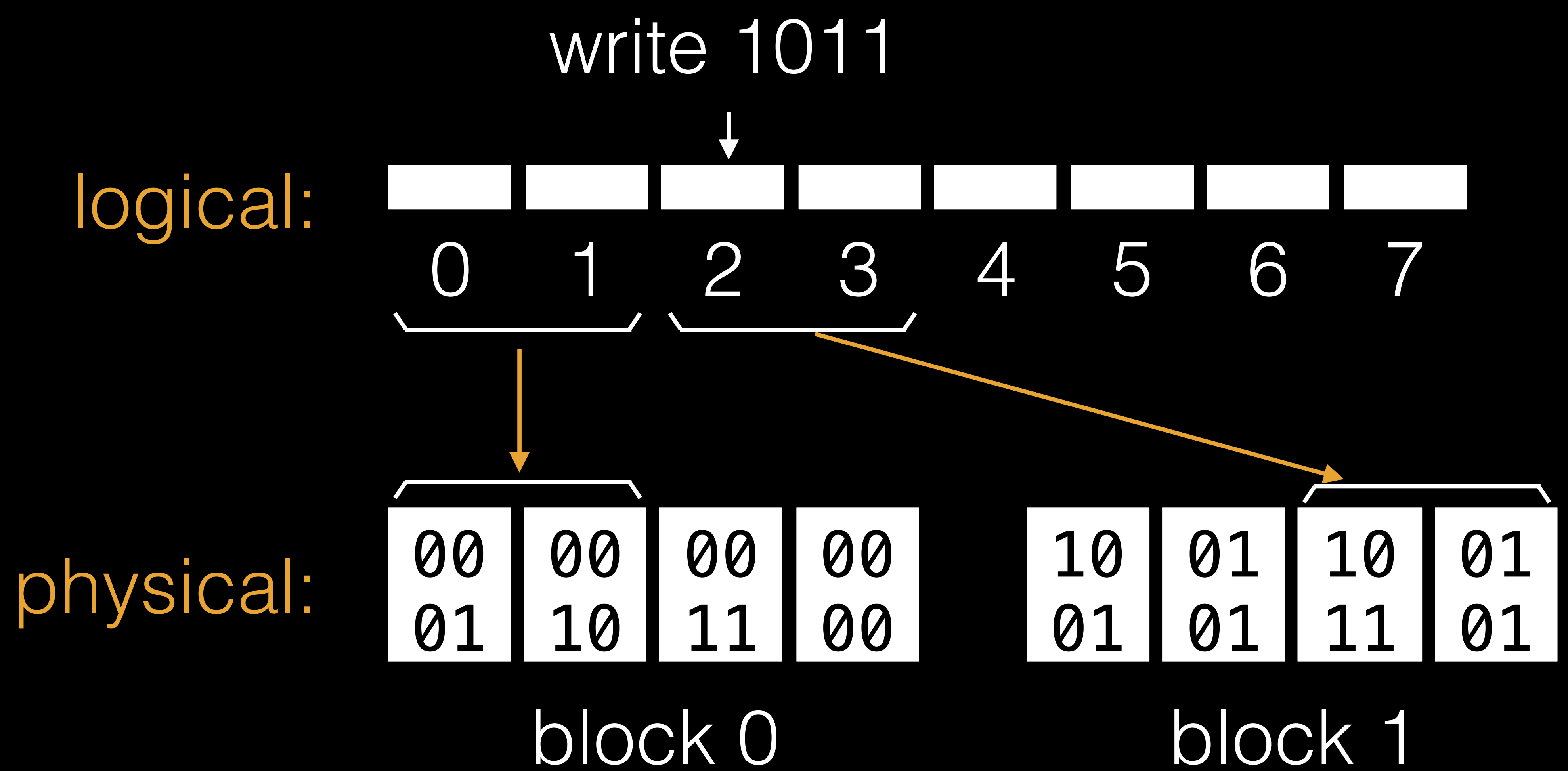
2-Page Translations



2-Page Translations



2-Page Translations



Larger Mappings

Advantage: larger mappings decrease table size.

Disadvantages?
- more read-modify-write updates
- more garbage
- less flexibility for placement

Hybrid FTL

Use course-grained mapping for most (e.g., 95%) of data. Map at block level.

Use fine-grained mapping for recent data. Map at page level.

Log Blocks

Write changed pages to designated log blocks.

After blocks become full, merge changes with old data.

Eventually garbage collect old pages.

Merging

Merging technique depends on I/O pattern.

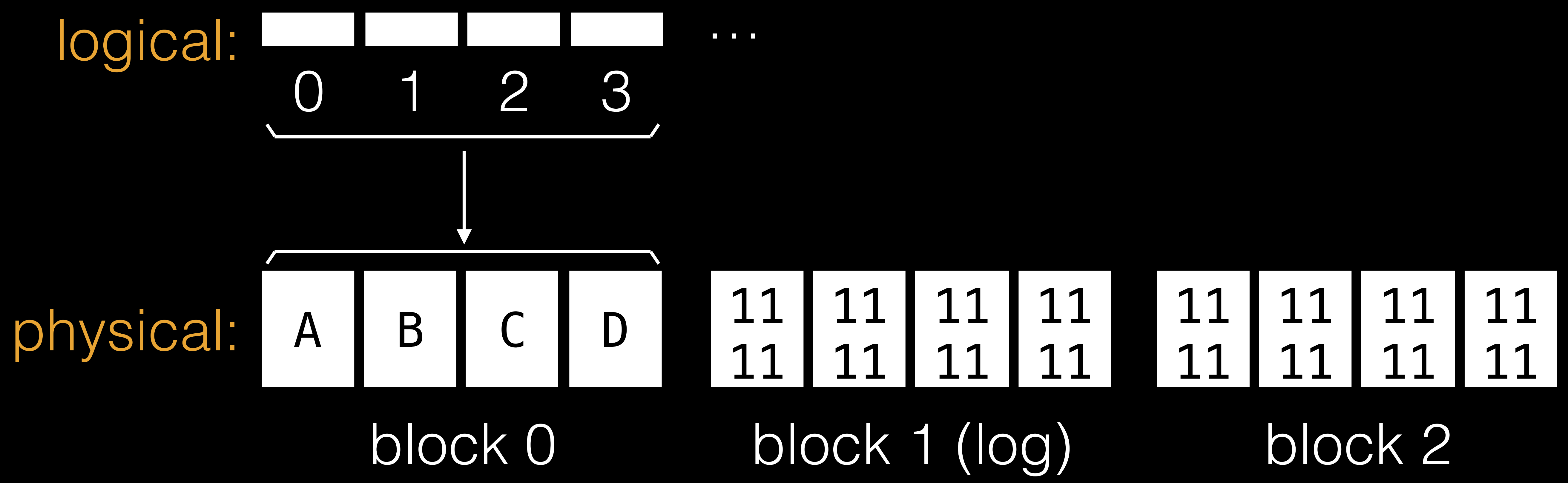
Three merge types:
- full merge
- partial merge
- switch merge

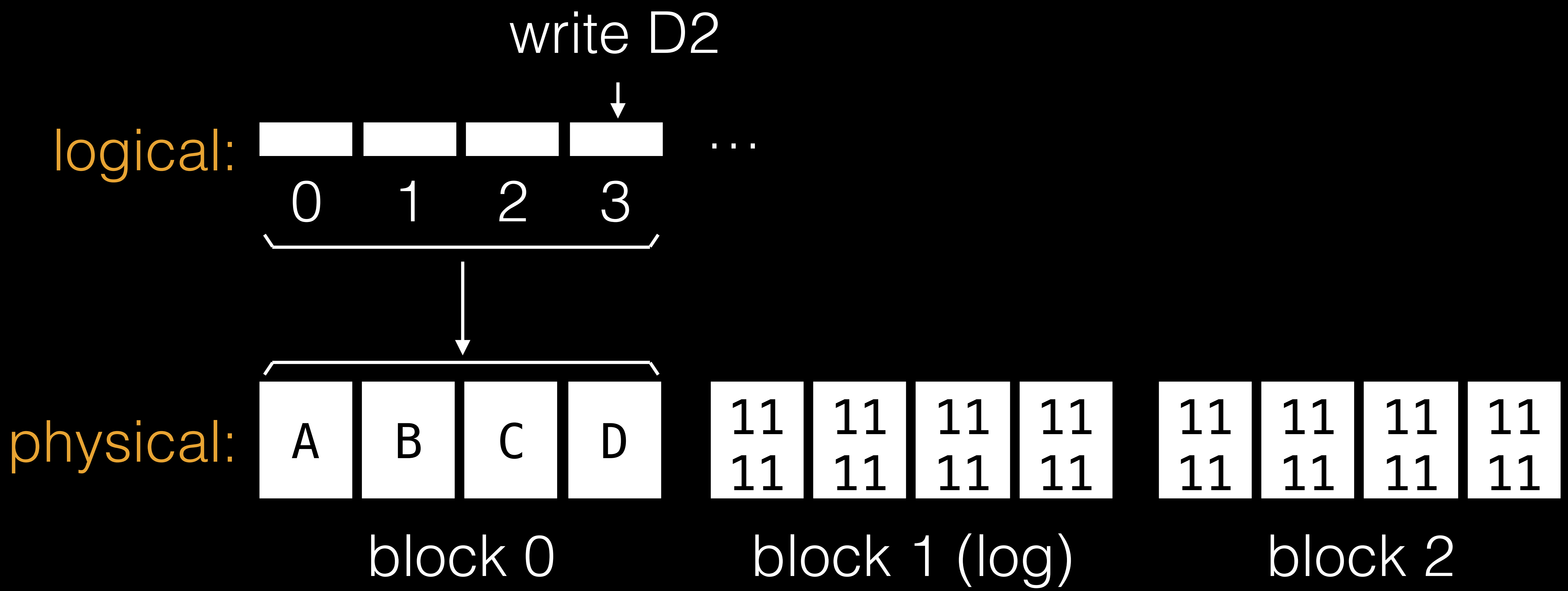
Merging

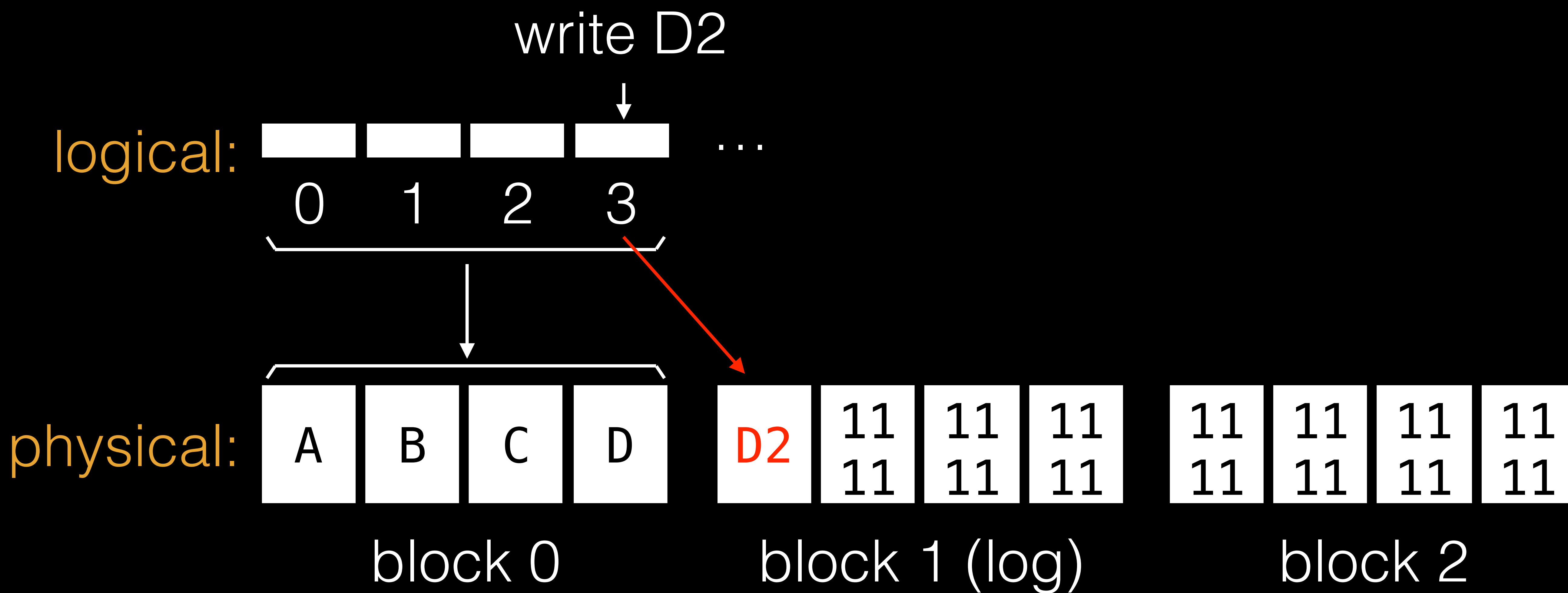
Merging technique depends on I/O pattern.

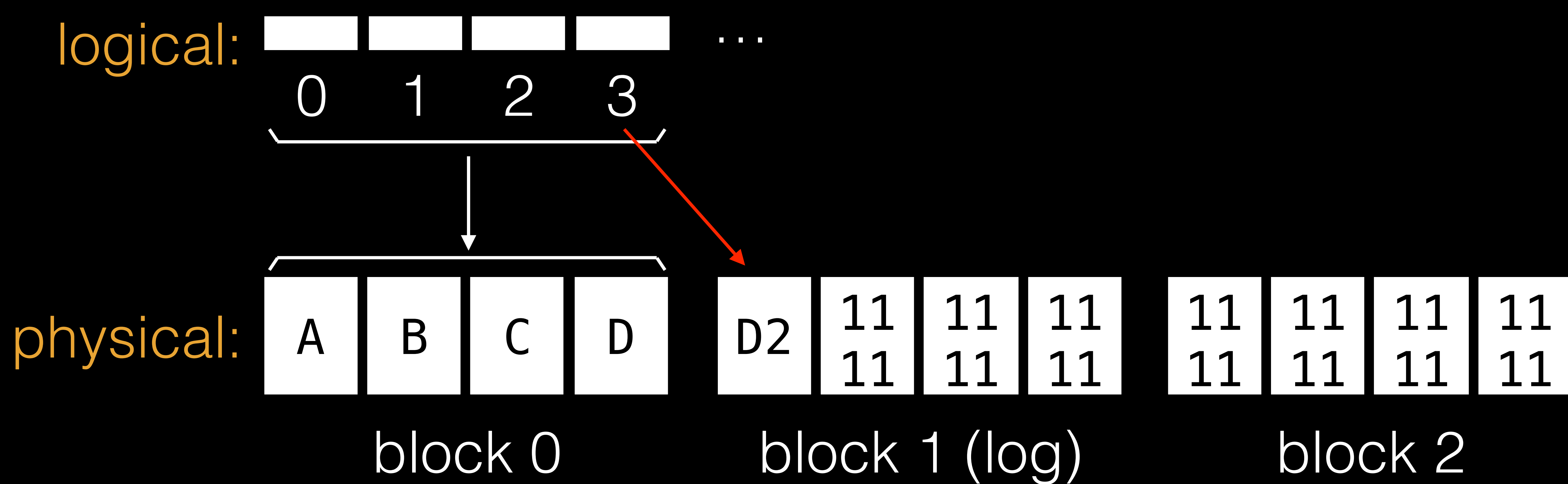
Three merge types:

- full merge
- partial merge
- switch merge

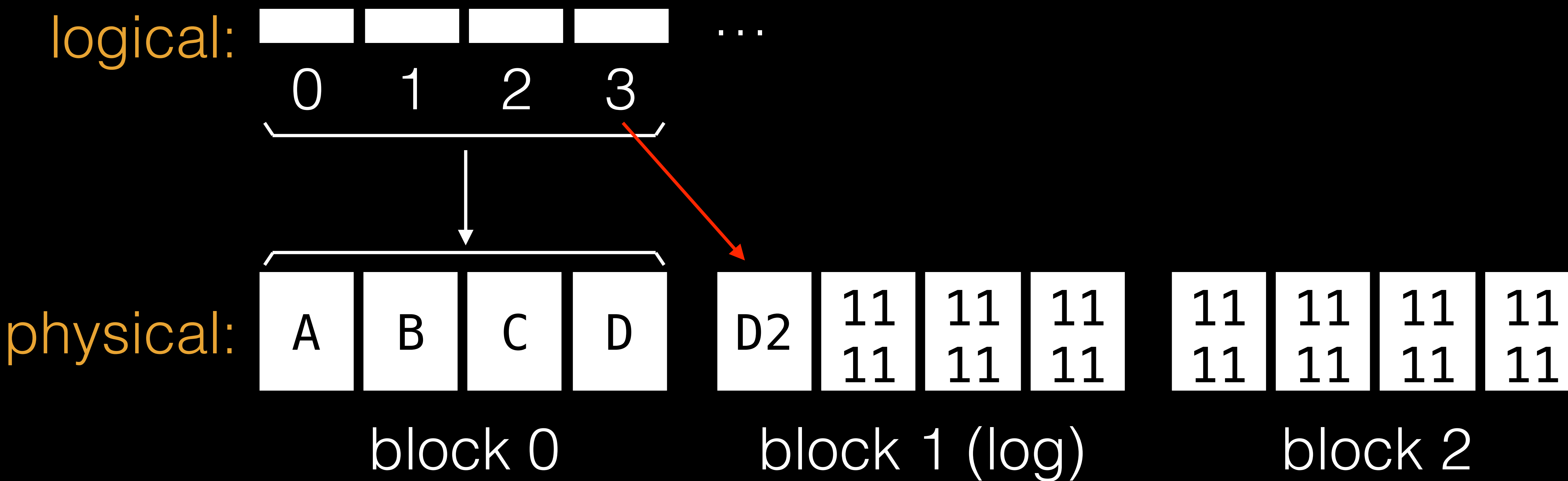

logical:
0
1
2
3
...
physical:
A
B
C
D
11 11
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2


write D2
logical:
0
1
2
3
...
physical:
A
B
C
D
11 11
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

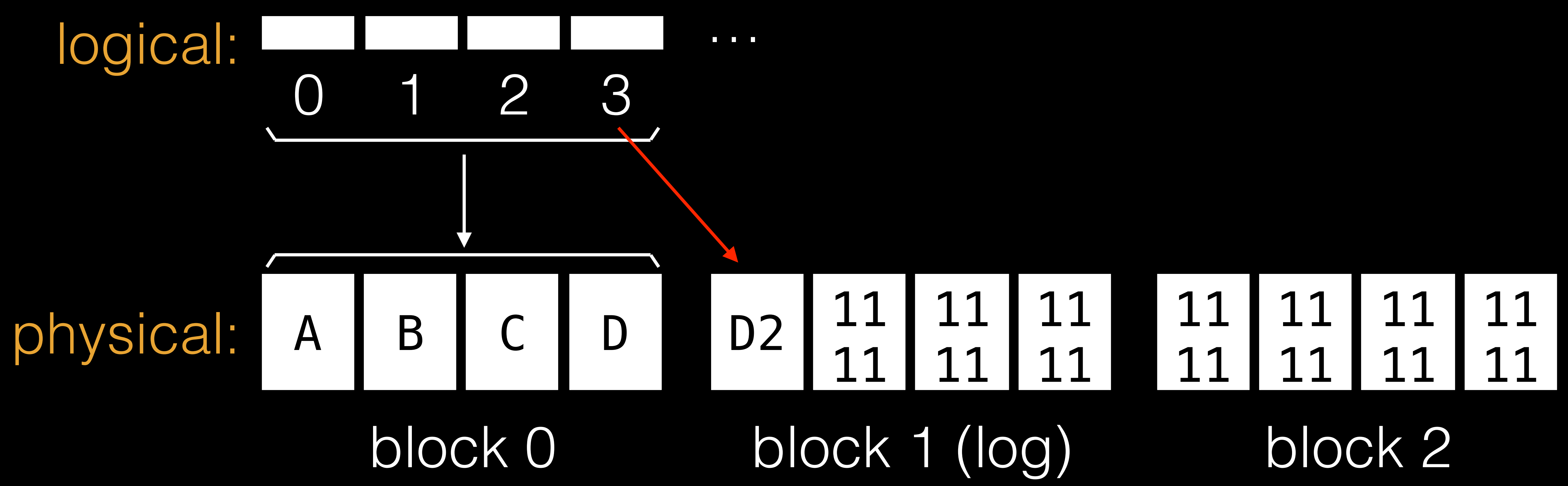

write D2
logical:
0
1
2
3
...
physical:
A
B
C
D
D2
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

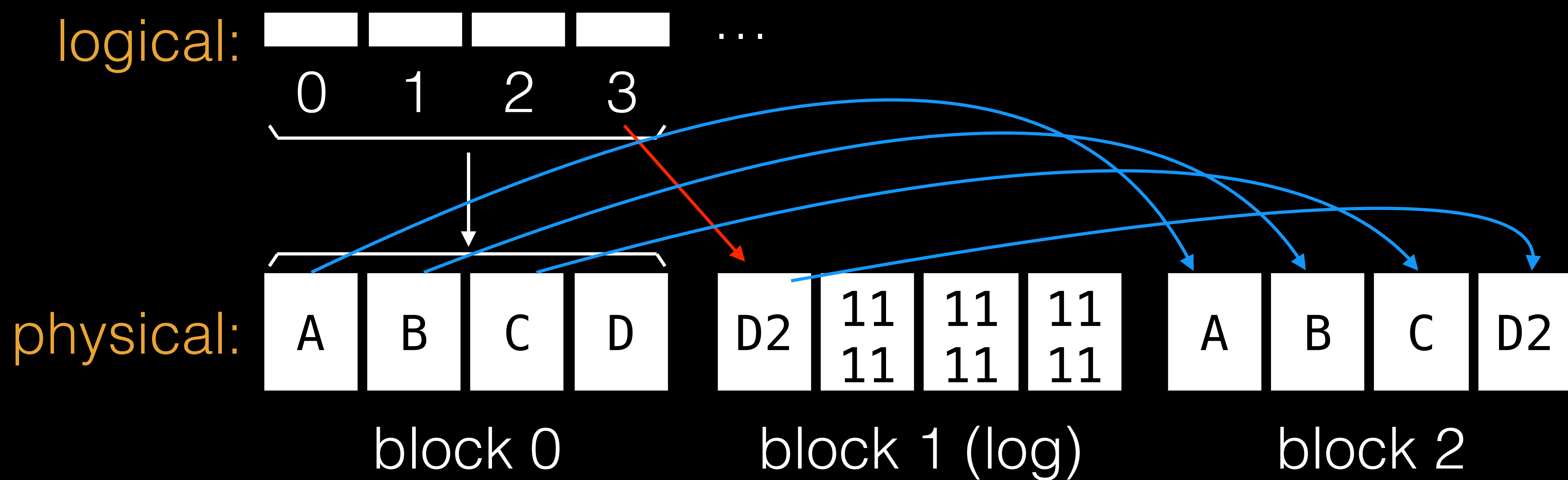


logical:
0
1
2
3
...
physical:
A
B
C
D
D2
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

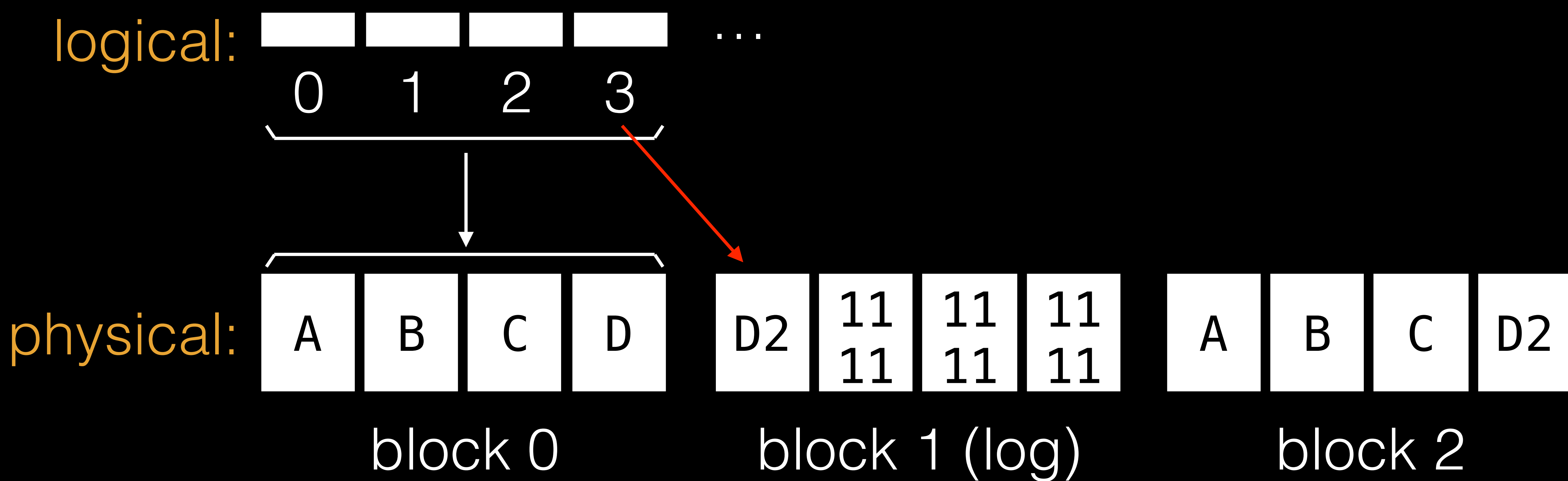


eventually, we need to get rid of red arrows,
as these represent expensive mappings

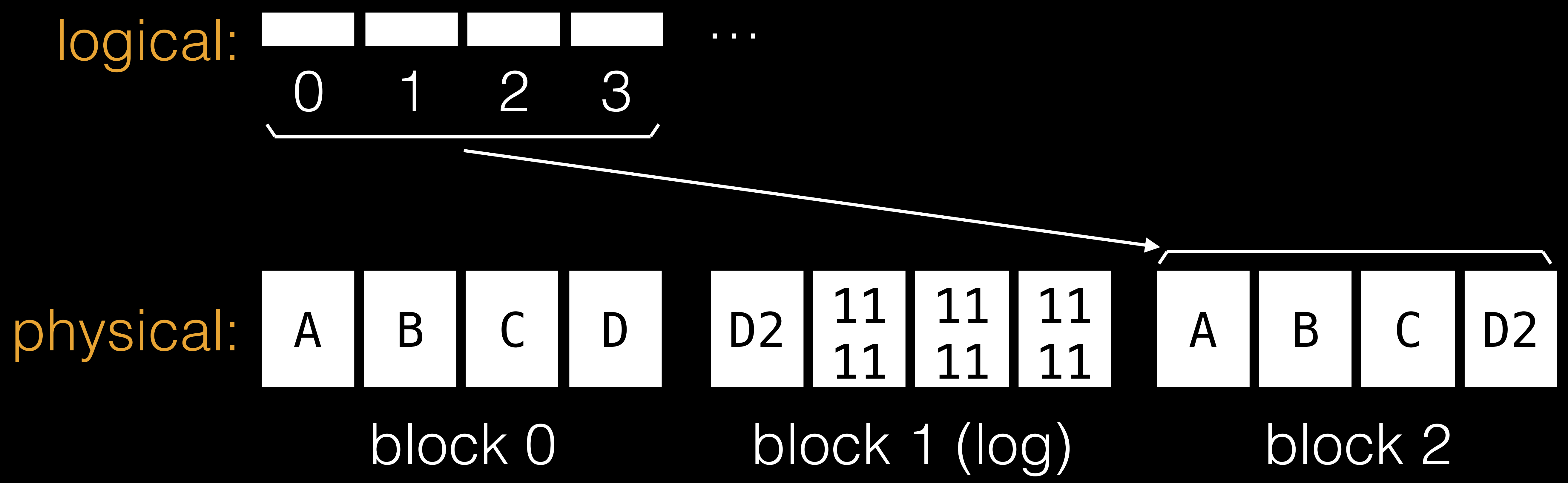

logical:
...
0
1
2
3
physical:
A
B
C
D
D2
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

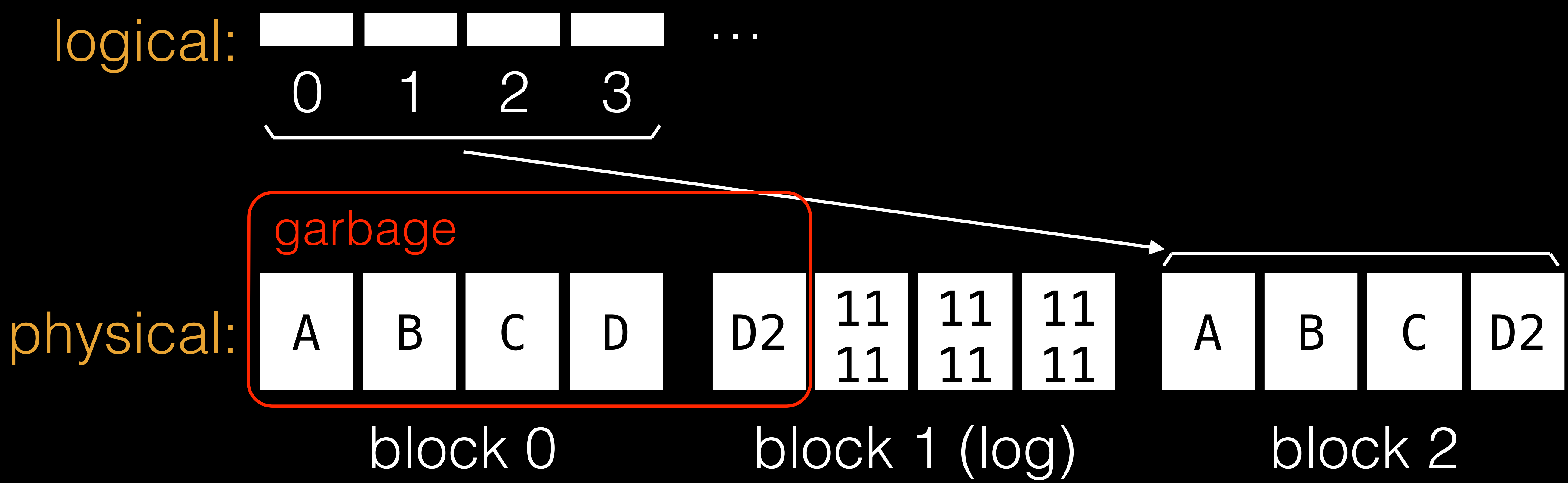


logical:
...
0
1
2
3
physical:
A
B
C
D
D2
11 11
11 11
11 11
A
B
C
D2
block 0
block 1 (log)
block 2



logical:
...
0
1
2
3
physical:
A
B
C
D
D2
11 11
11 11
11 11
A
B
C
D2
block 0
block 1 (log)
block 2

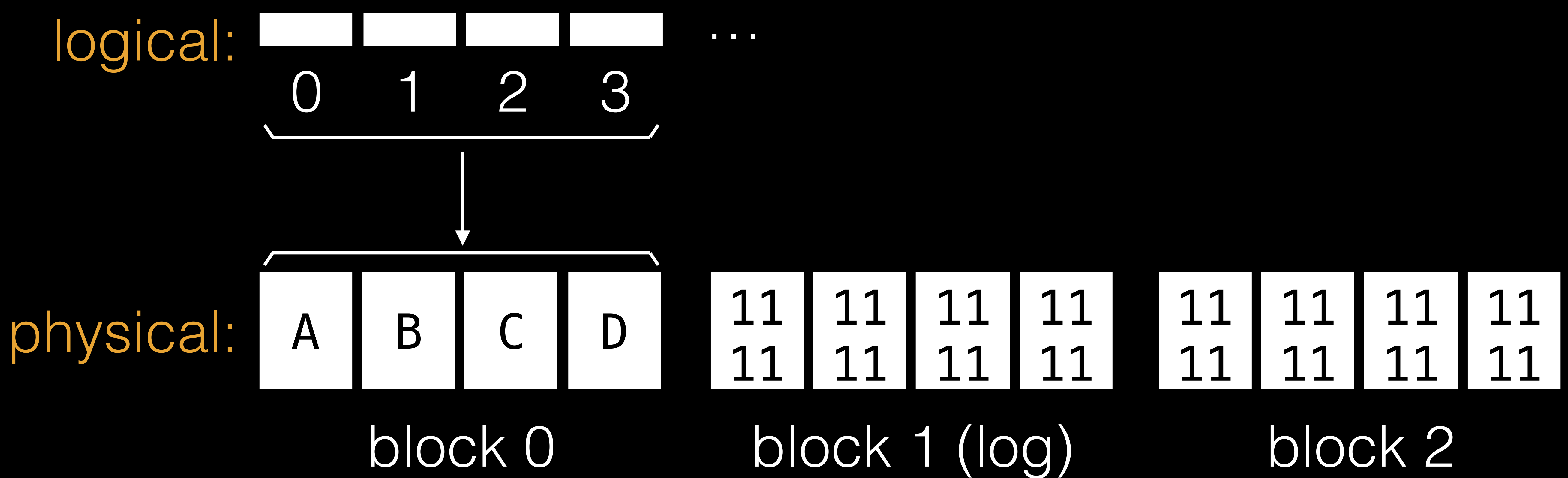

logical:
0
1
2
3
...
physical:
A
B
C
D
D2
11 11
11 11
11 11
A
B
C
D2
block 0
block 1 (log)
block 2

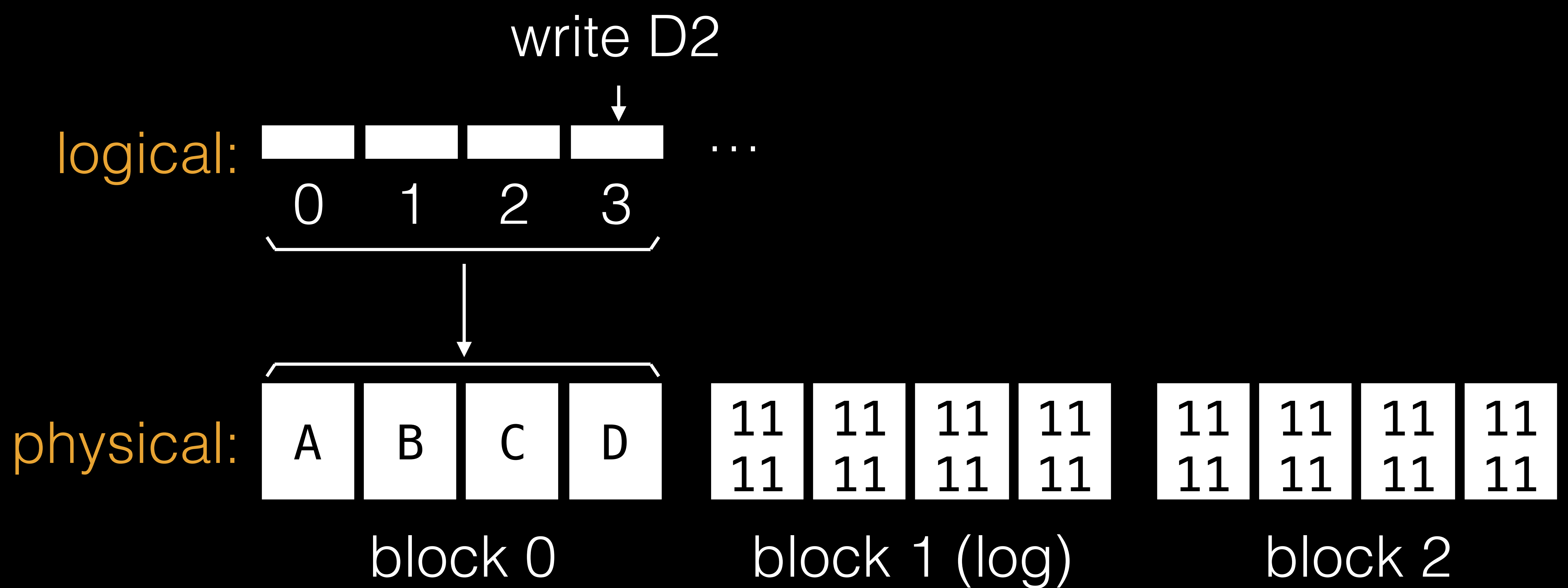

logical:
...
0
1
2
3
garbage
physical:
A
B
C
D
D2
11 11
11 11
11 11
A
B
C
D2
block 0
block 1 (log)
block 2

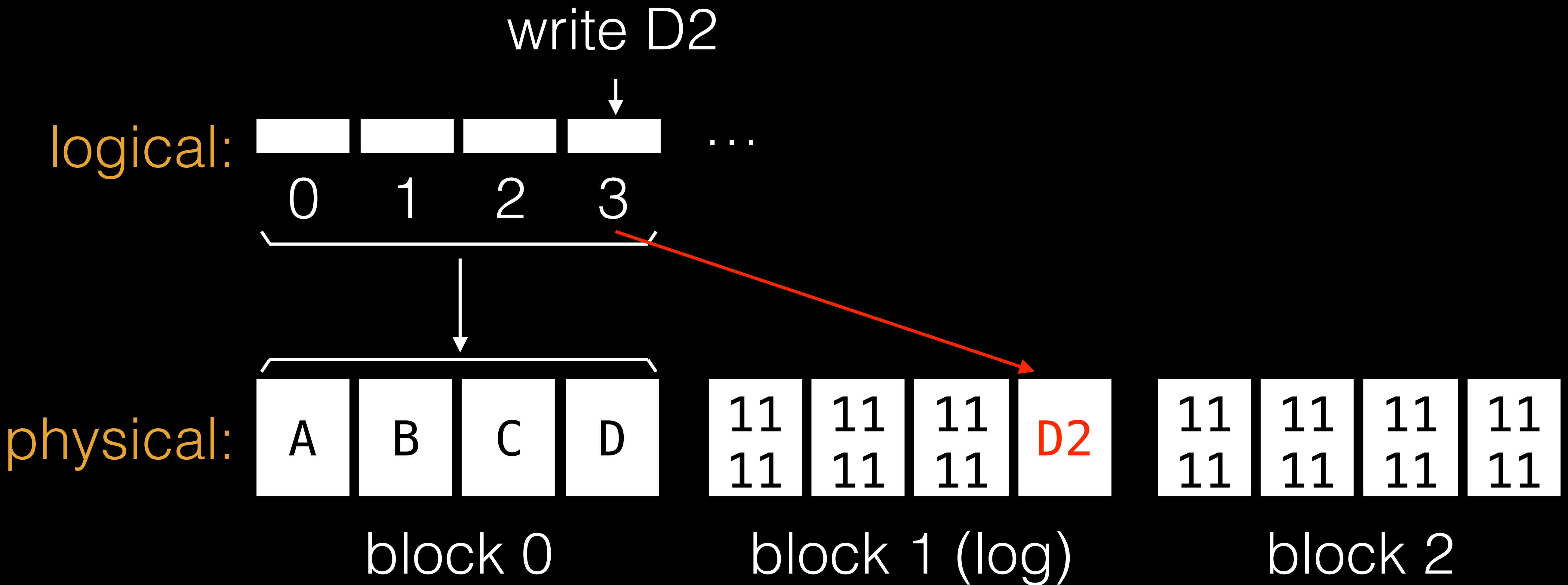
Merging

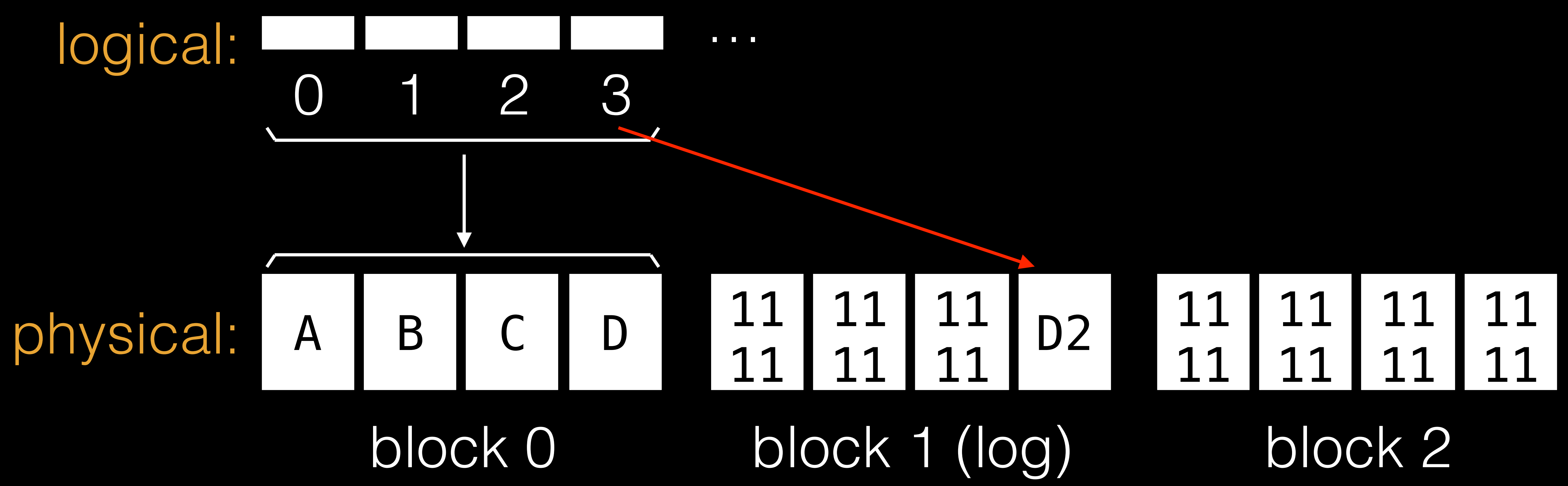
Merging technique depends on I/O pattern.

Three merge types:
- full merge
- partial merge
- switch merge

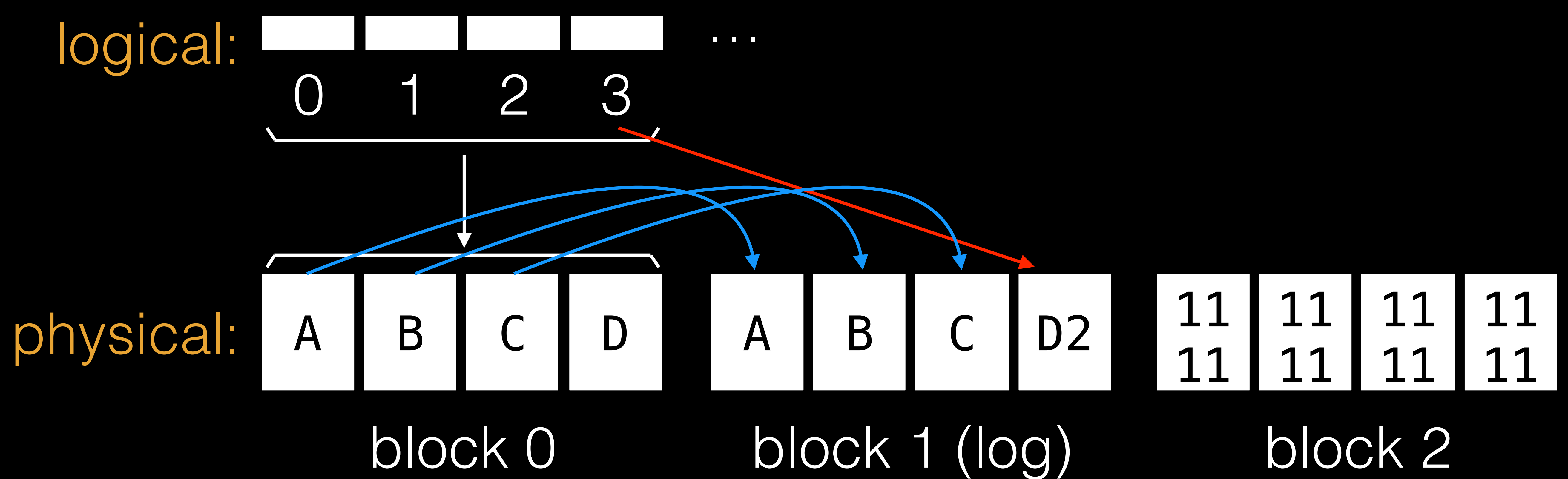

logical:
0 1 2 3
...
physical:
A B C D
11 11 11 11
11 11 11 11
11 11 11 11
11 11 11 11
block 0
block 1 (log)
block 2

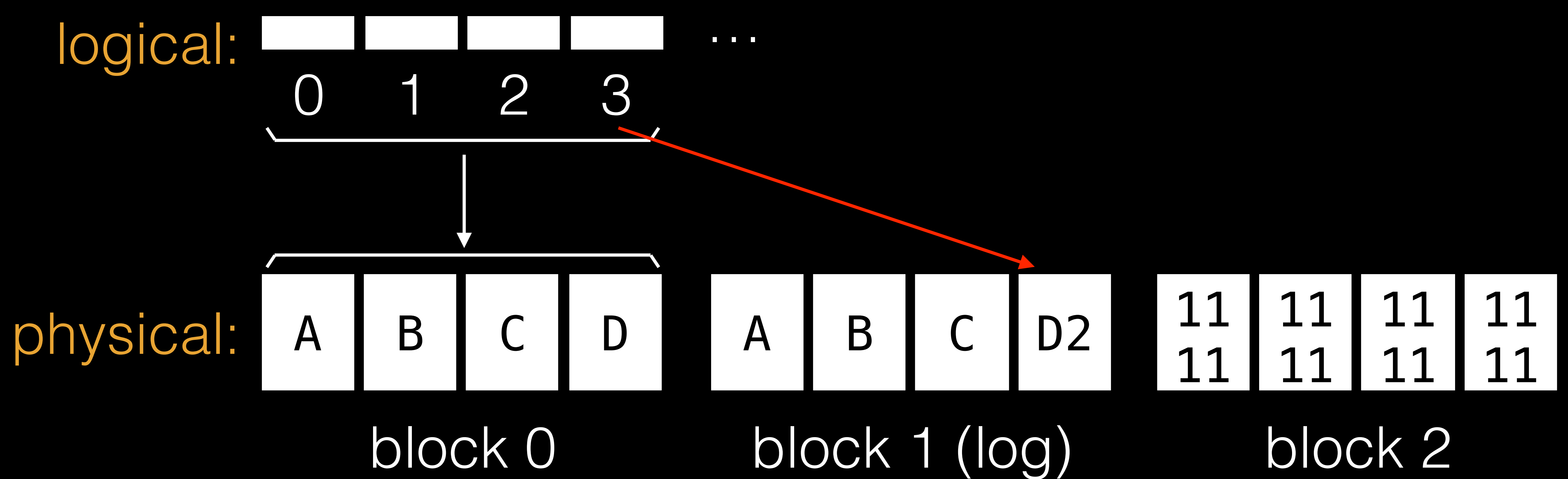

write D2
logical:
0
1
2
3
...
physical:
A
B
C
D
11 11
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

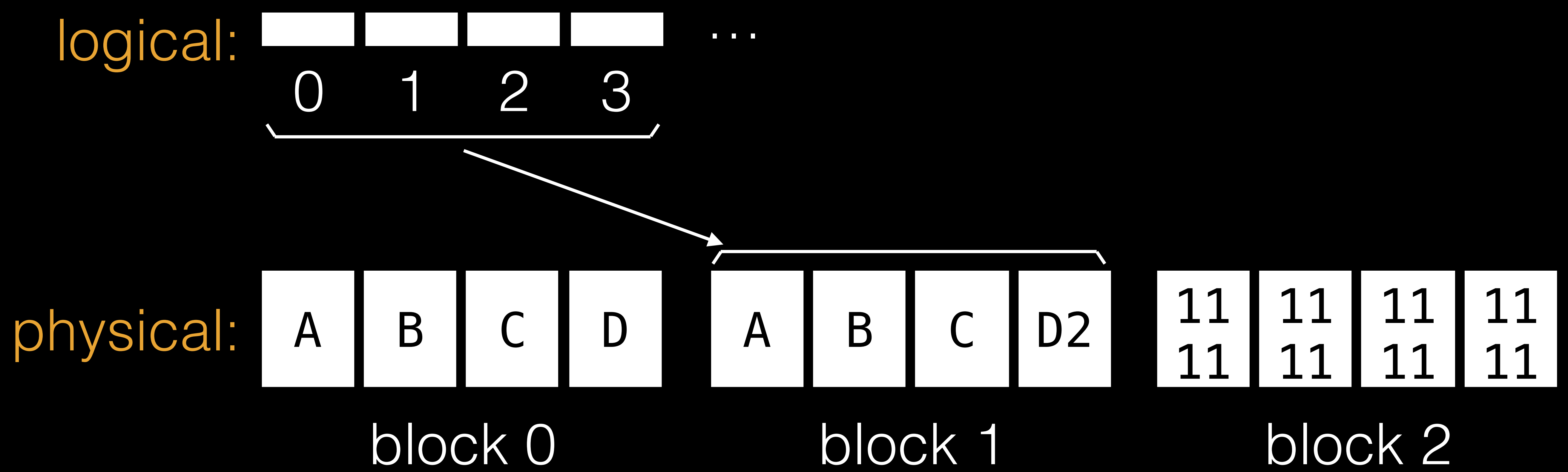

write D2
logical:
0
1
2
3
...
physical:
A
B
C
D
11 11
11 11
11 11
D2
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

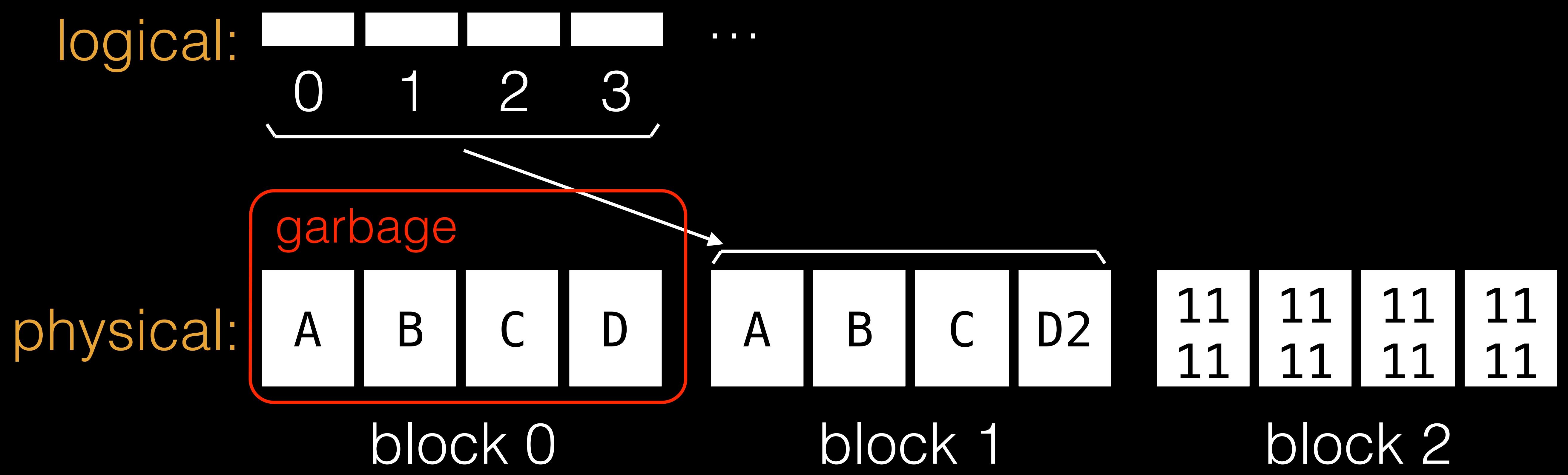


logical:
...
0
1
2
3
physical:
A
B
C
D
11 11
11 11
11 11
D2
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2


logical:
...
0
1
2
3
physical:
A
B
C
D
A
B
C
D2
11
11
11
11
11
11
11
11
block 0
block 1 (log)
block 2


logical:
0
1
2
3
...
physical:
A
B
C
D
A
B
C
D2
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

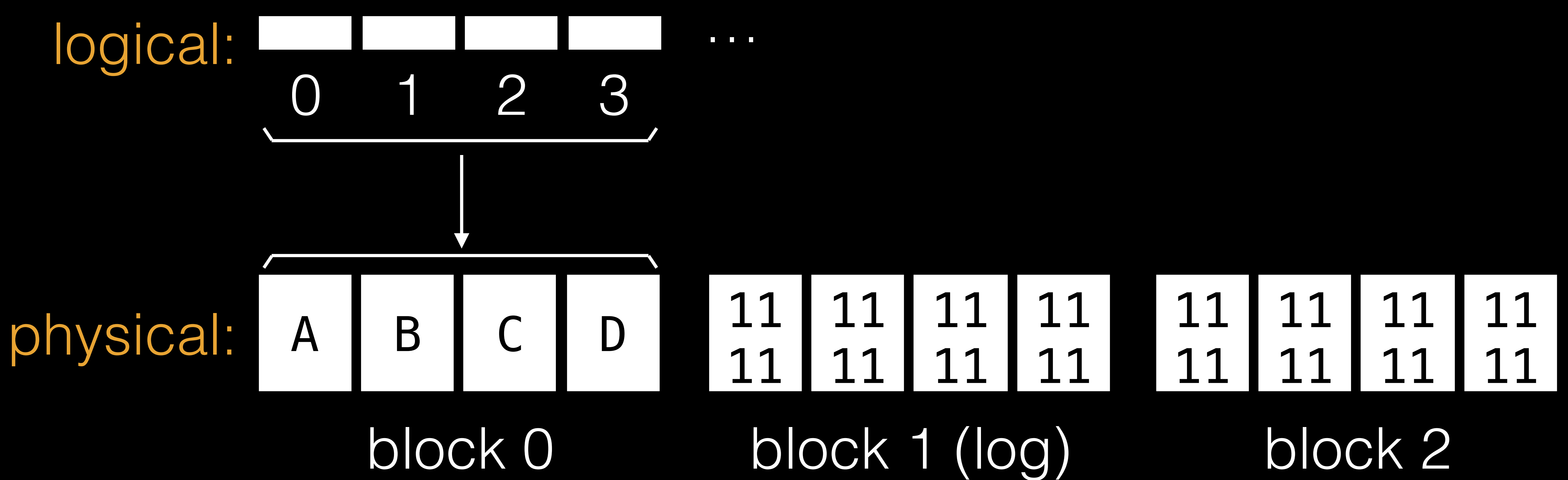

logical:
0
1
2
3
...
physical:
A
B
C
D
A
B
C
D2
11 11
11 11
11 11
11 11
block 0
block 1
block 2

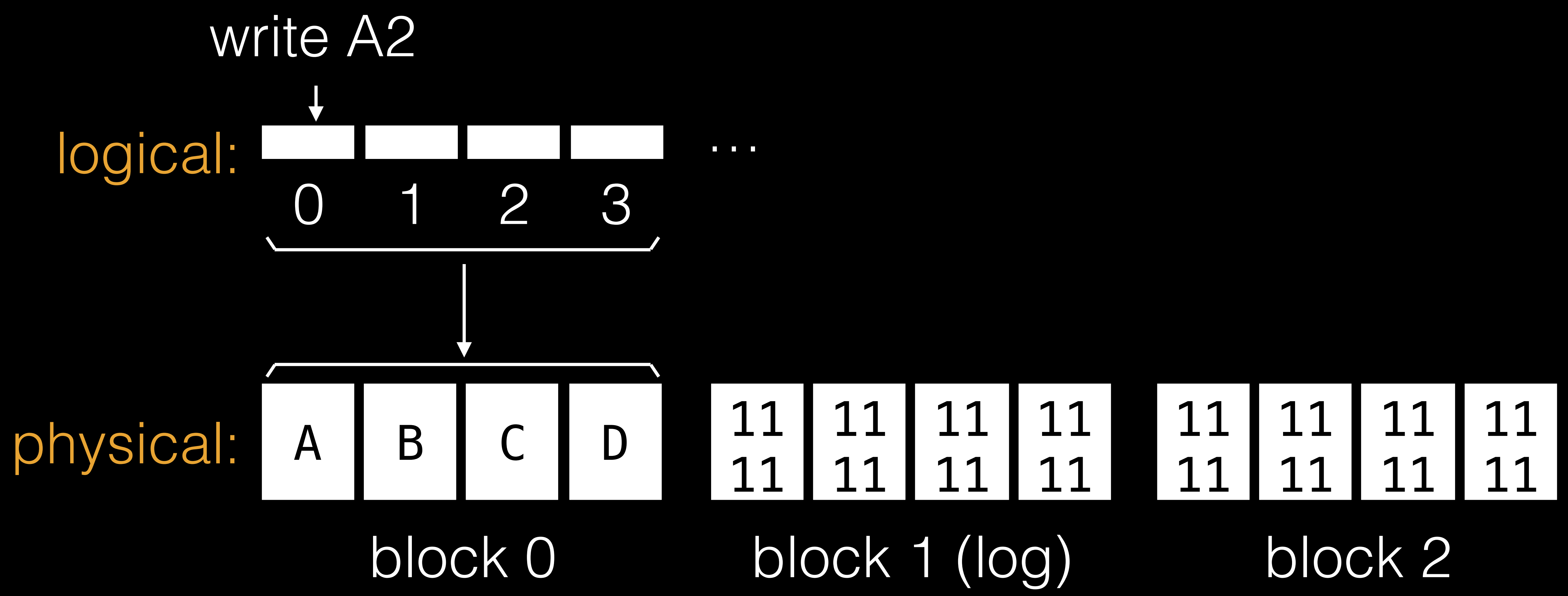

logical:
0
1
2
3
...
garbage
physical:
A
B
C
D
A
B
C
D2
11 11
11 11
11 11
11 11
block 0
block 1
block 2

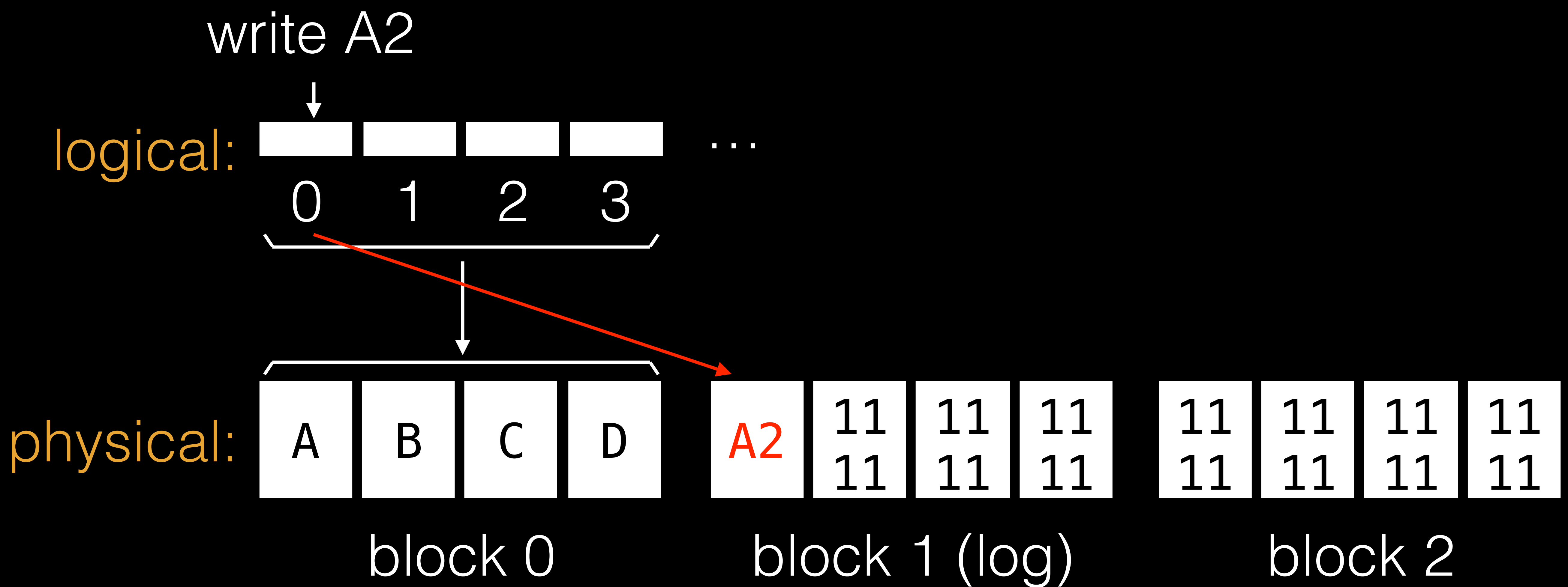
Merging

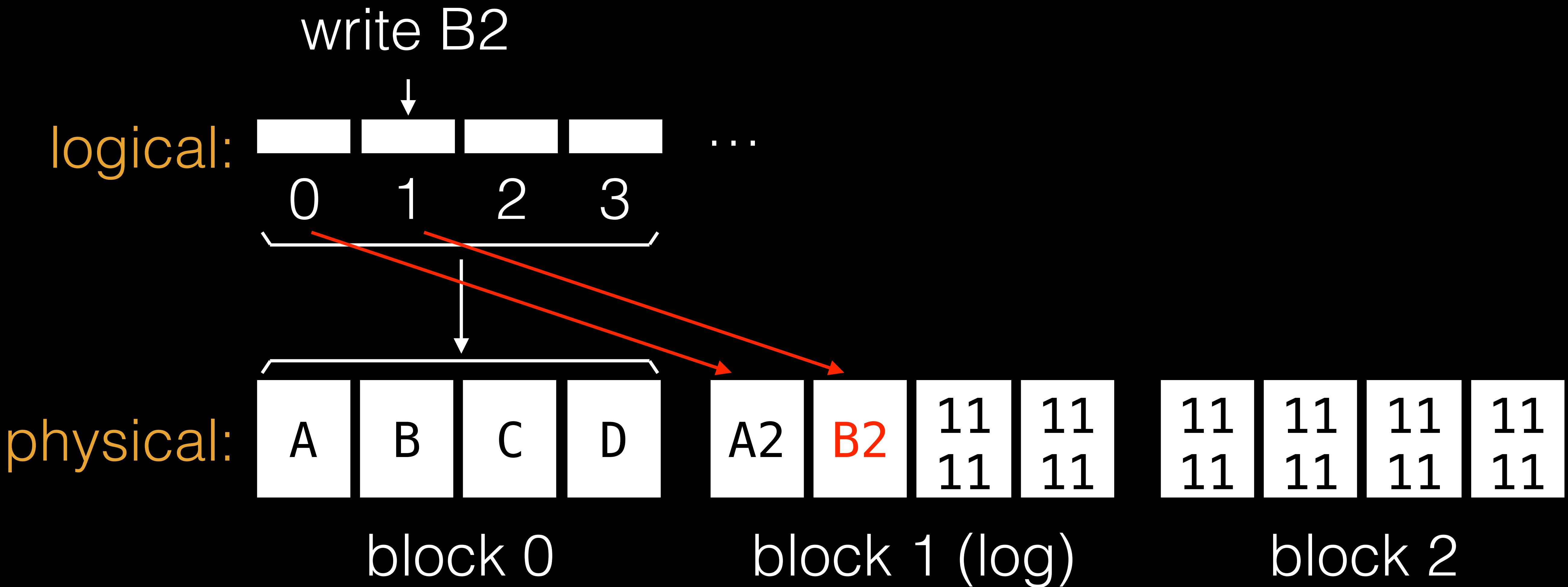
Merging technique depends on I/O pattern.

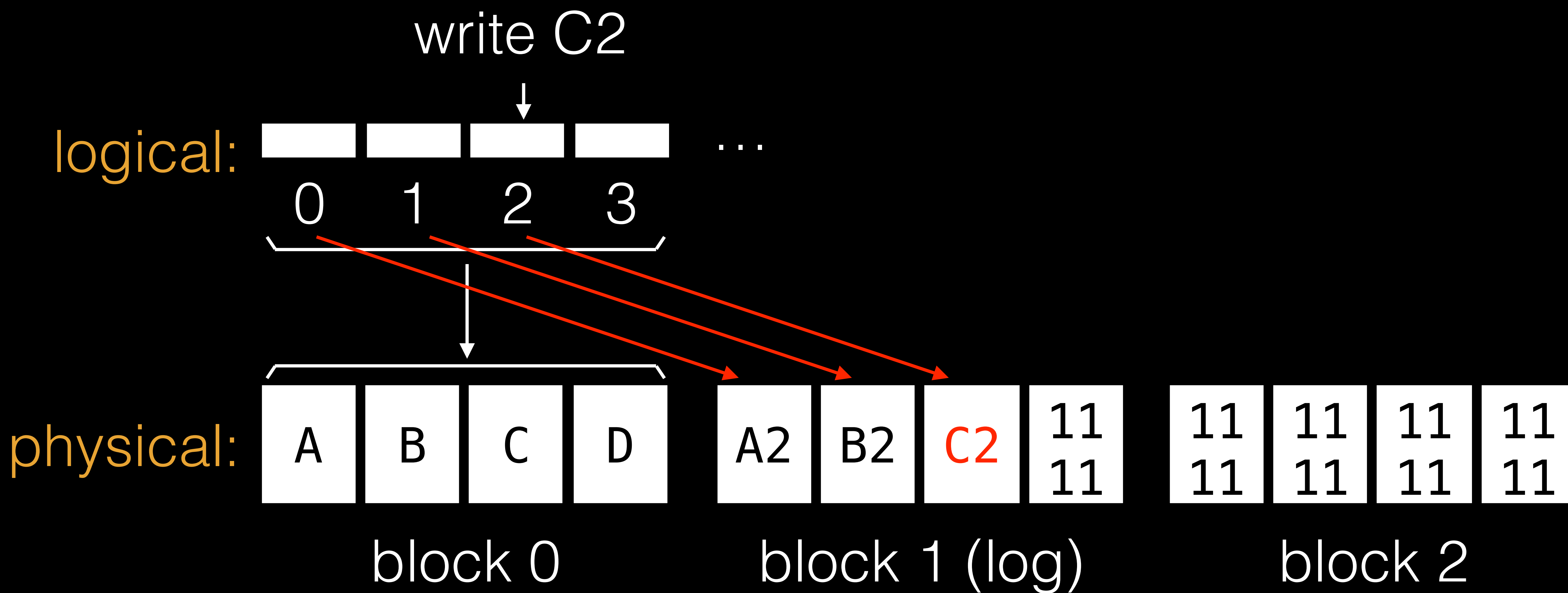
Three merge types:
- full merge
- partial merge
- switch merge

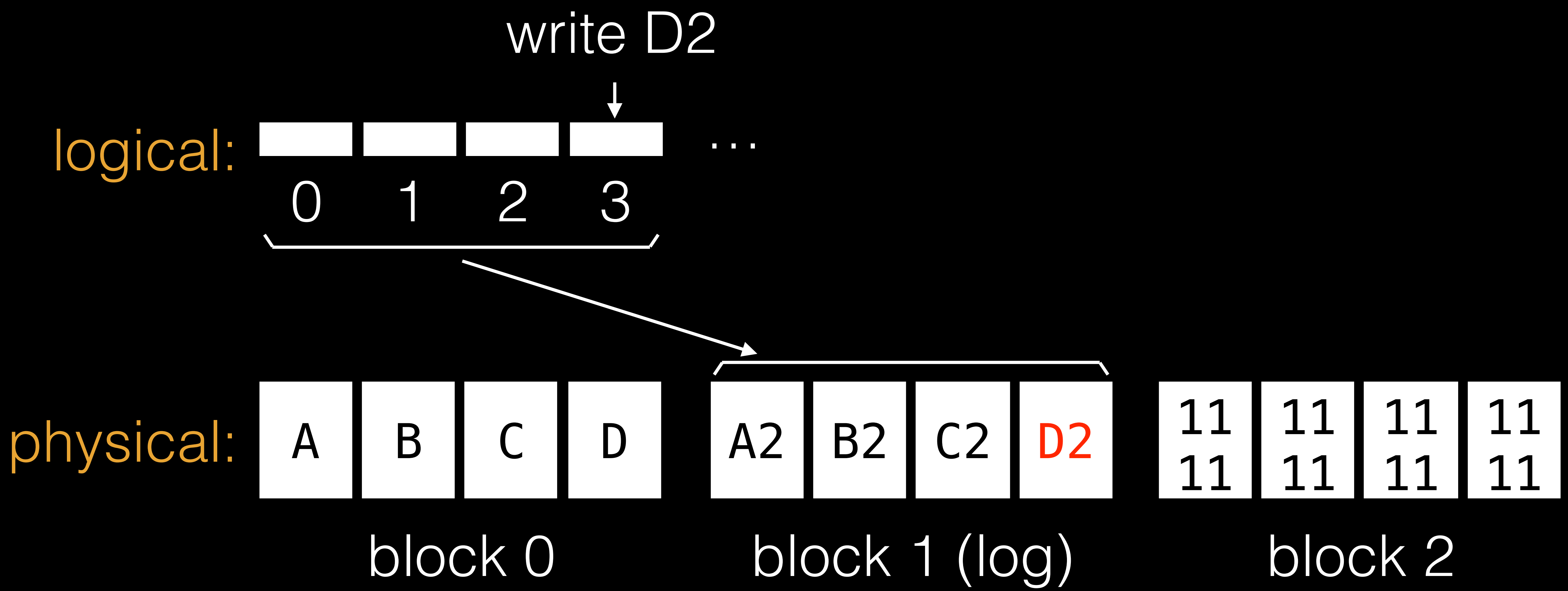

logical:
0
1
2
3
...
physical:
A
B
C
D
11 11
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

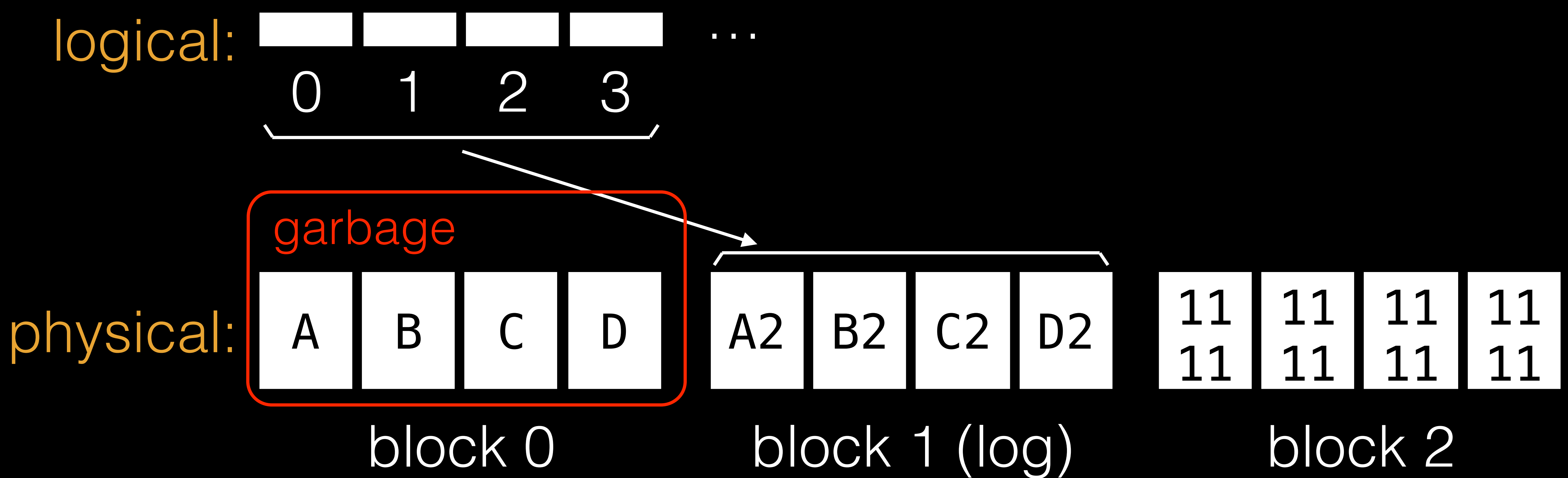

write A2
logical:
0
1
2
3
...
physical:
A
B
C
D
11 11
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2


write A2
logical:
0
1
2
3
...
physical:
A
B
C
D
A2
11 11
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2


write B2
logical:
0
1
2
3
...
physical:
A
B
C
D
A2
B2
11 11
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2


write C2
logical:
0
1
2
3
...
physical:
A
B
C
D
A2
B2
C2
11 11
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2


write D2
logical:
0
1
2
3
...
physical:
A
B
C
D
A2
B2
C2
D2
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2


logical:
0
1
2
3
...
garbage
physical:
A
B
C
D
A2
B2
C2
D2
11 11
11 11
11 11
11 11
block 0
block 1 (log)
block 2

Merging

Merging technique depends on I/O pattern.

Three merge types:
- full merge
- partial merge
- switch merge

Summary

Flash is much faster than disk, but…

It is more expensive.

It's not a drop-in replacement beneath an FS without a complex layer for emulating hard disk API.