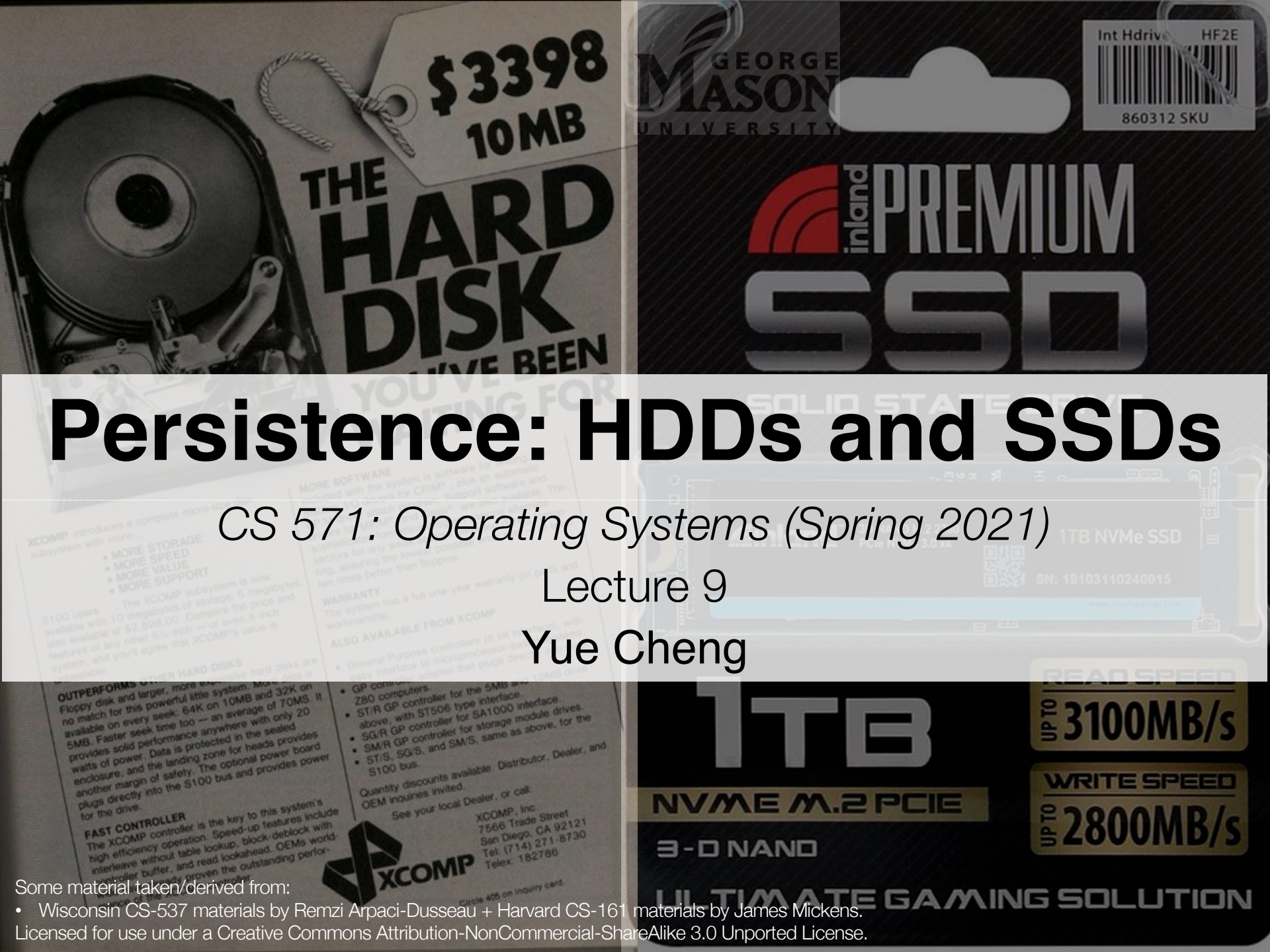


Persistence: HDDs and SSDs

CS 571: Operating Systems (Spring 2021)

Lecture 9

Yue Cheng

Some material taken/derived from:

- Wisconsin CS-537 materials by Remzi Arpaci-Dusseau + Harvard CS-161 materials by James Mickens.

Hard Disk Drives (HDDs)

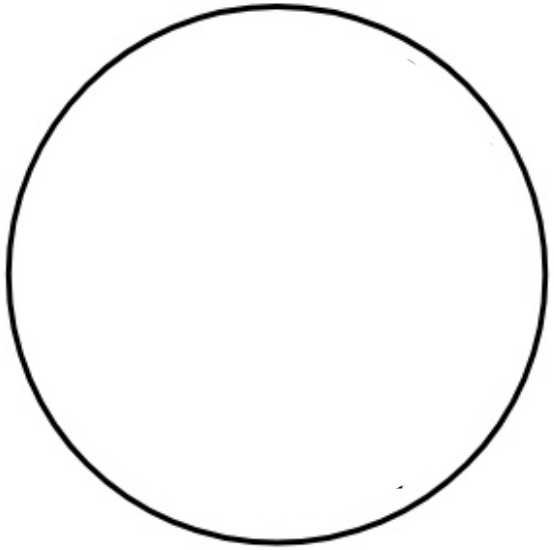
Basic Interface

- A magnetic disk has a sector-addressable address space
  - You can think of a disk as an array of sectors
  - Each sector (logical block) is the smallest unit of transfer
- Sectors are typically 512 or 4096 bytes
- Main operations
  - Read from sectors (blocks)
  - Write to sectors (blocks)

Disk Structure

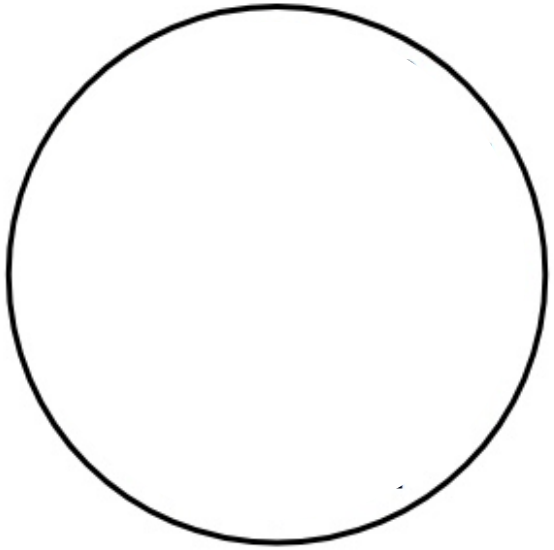
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
  - Logical to physical address should be easy
    - Except for bad sectors

Internals of Hard Disk Drive (HDD)



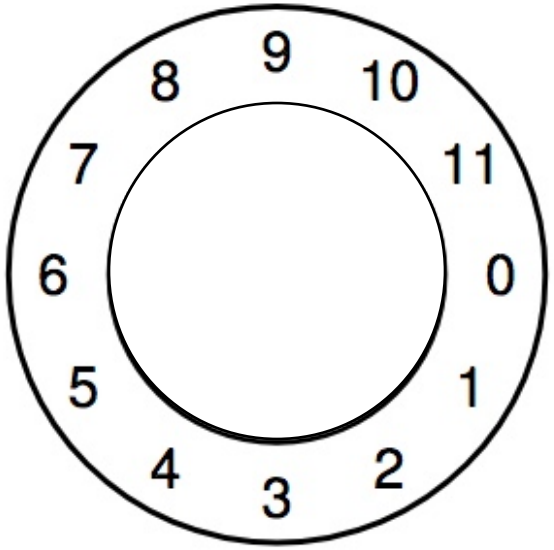
Internals of Hard Disk Drive (HDD)

Platter
Covered with a magnetic film



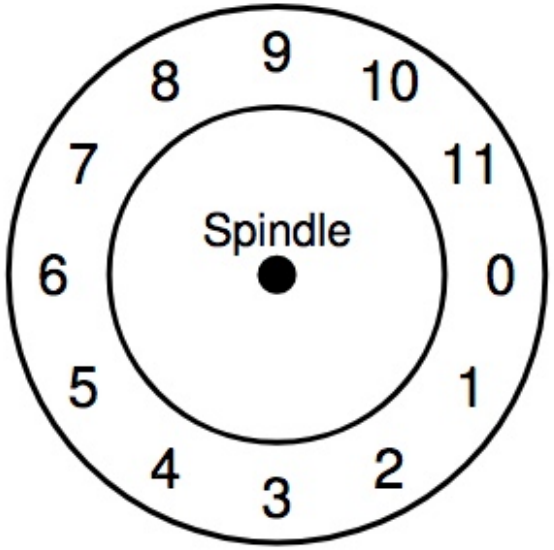
Internals of Hard Disk Drive (HDD)

A single track example



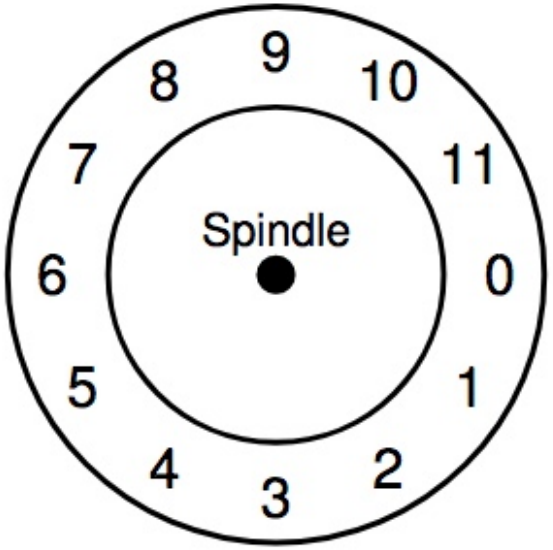
Internals of Hard Disk Drive (HDD)

Spindle in the center of the surface



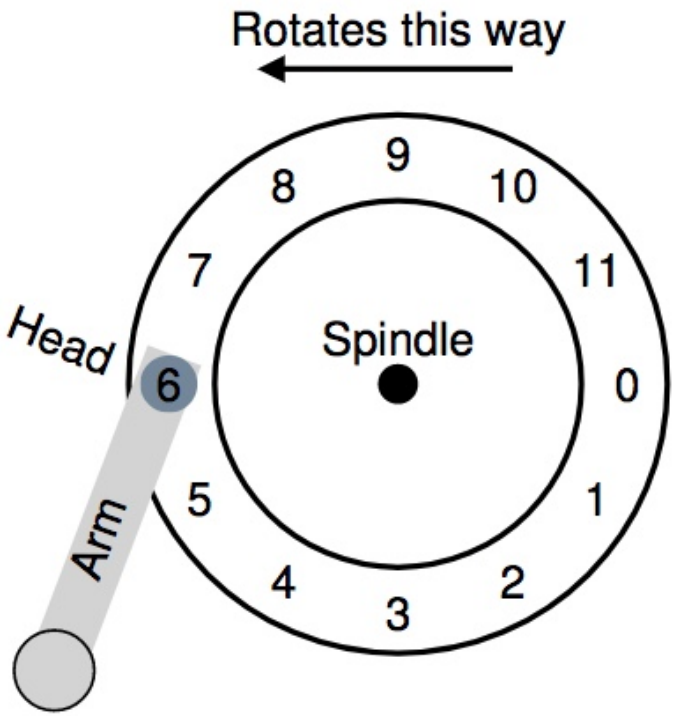
Internals of Hard Disk Drive (HDD)

The track is divided into numbered sectors

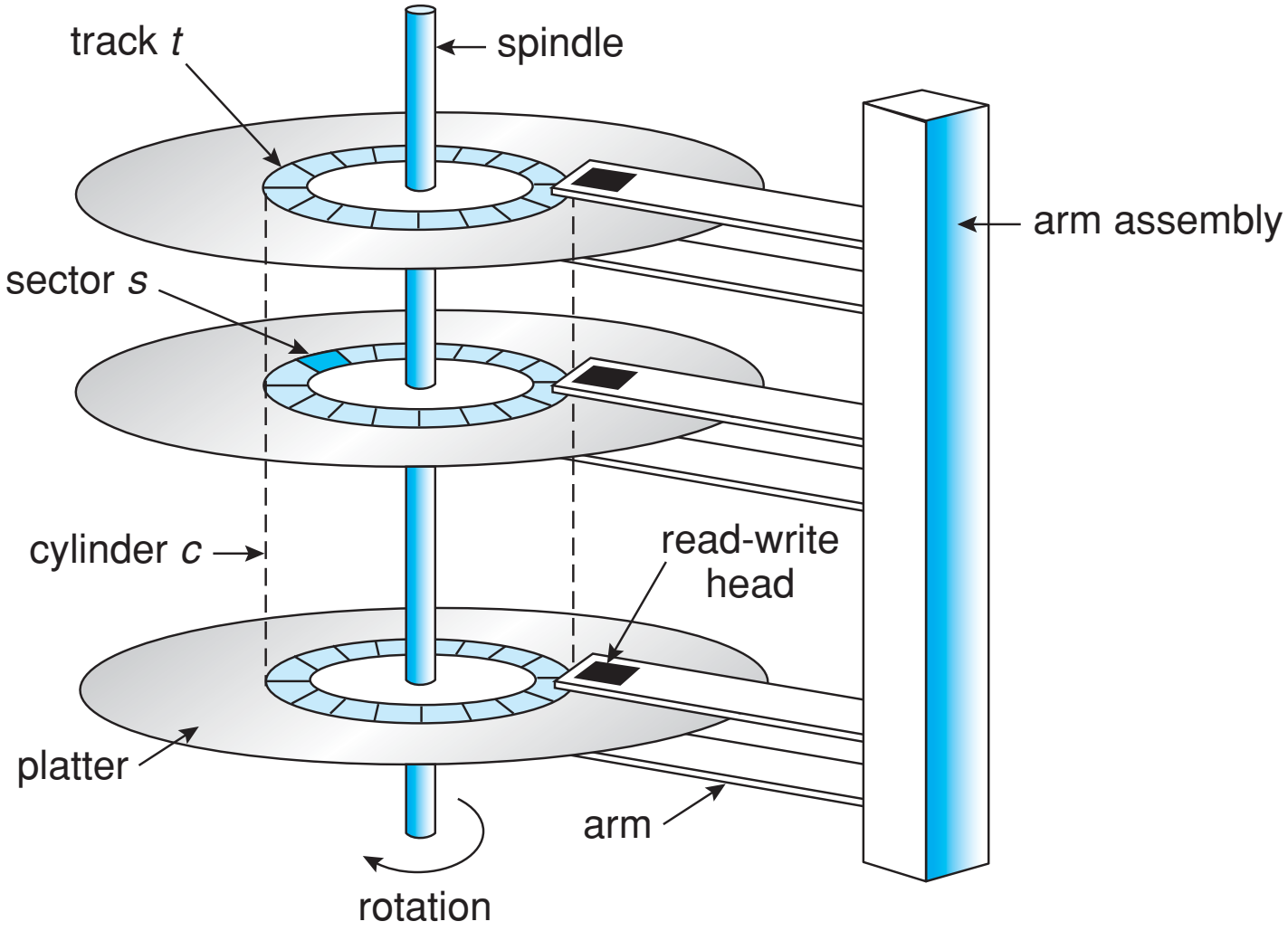


Internals of Hard Disk Drive (HDD)

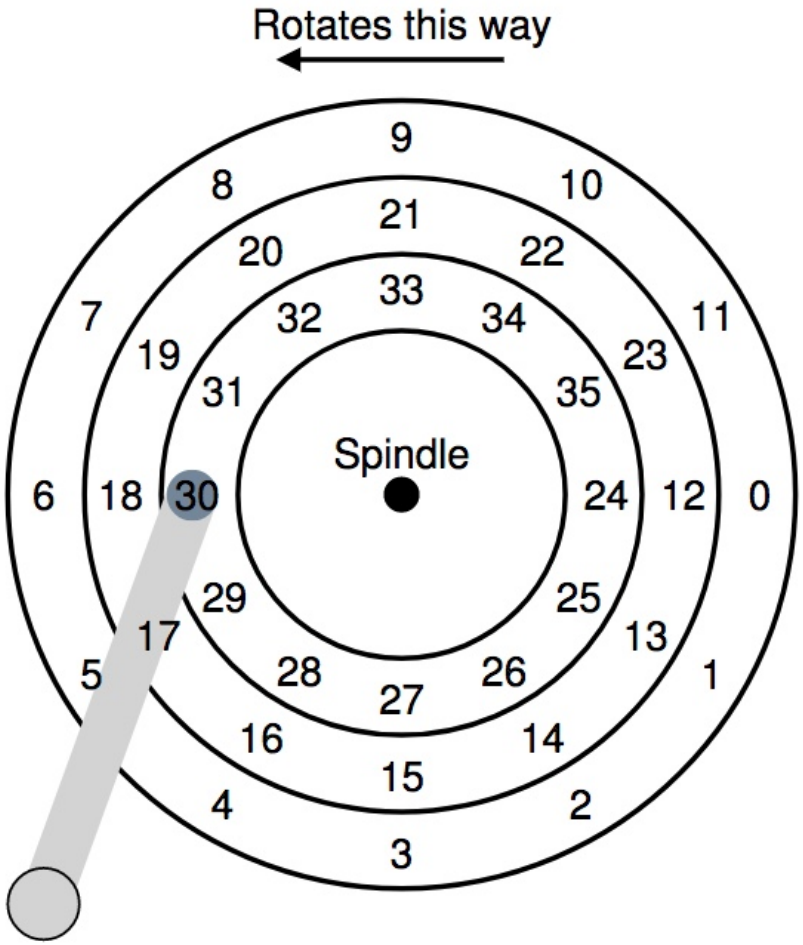
A single track + an arm + a head



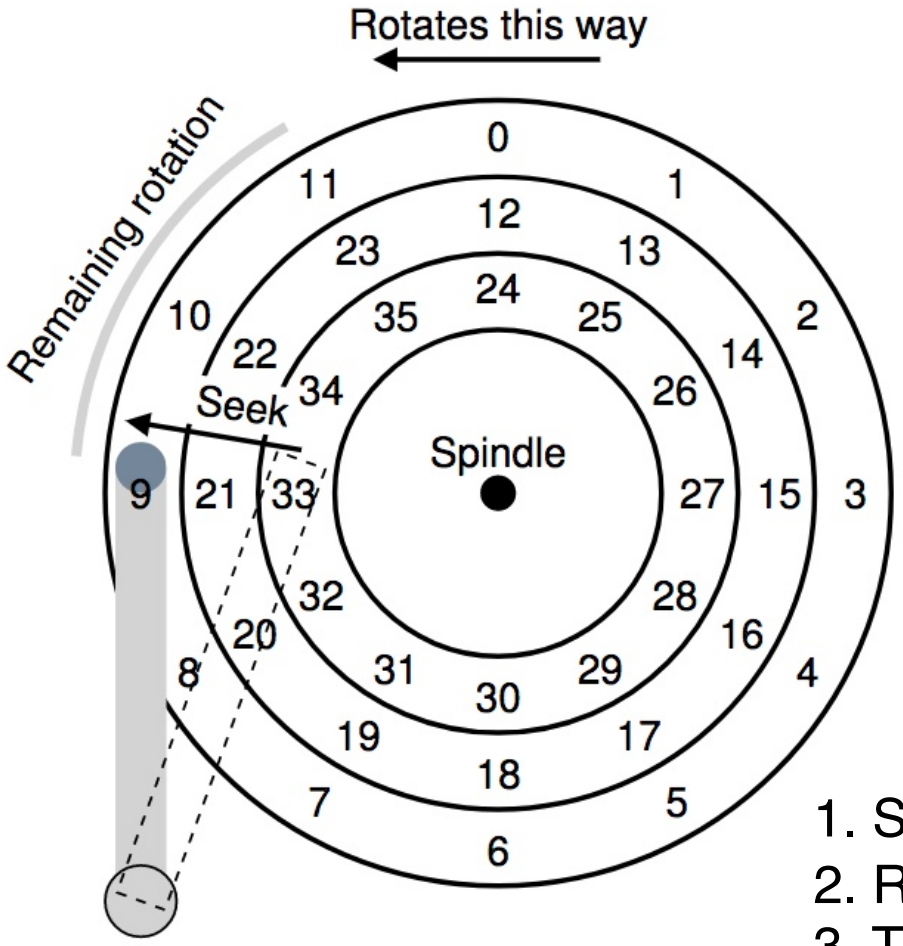
HDD Mechanism (3D view)



Let's Read Sector 0



Let's Read Sector 0



1. Seek for right track
2. Rotate (sector 9 → 0)
3. Transfer data (sector 0)

Don’t Try This at Home!

https://www.youtube.com/watch?v=9eMWG3fwiEU&feature=youtu.be&t=30s

Disk Performance

- I/O latency of disks

$$L_{I/O} = L_{seek} + L_{rotate} + L_{transfer}$$

- Disk access latency at millisecond level

Seek, Rotate, Transfer

- Seek may take several milliseconds (ms)
- Settling along can take 0.5 - 2ms
- Entire seek often takes 4 - 10ms

Seek, Rotate, Transfer

- Rotation per minute (RPM)
  - 7200 RPM is common nowadays
  - 15000 RPM is high end
  - Old computers may have 5400 RPM disks

- `1 / 7200 RPM = 1 minute / 7200 rotations =`

  `1 second / 120 rotations = 8.3 ms / rotation`

Seek, Rotate, Transfer

- Rotation per minute (RPM)
  - 7200 RPM is common nowadays
  - 15000 RPM is high end
  - Old computers may have 5400 RPM disks

- `1 / 7200 RPM = 1 minute / 7200 rotations =`

  `1 second / 120 rotations = 8.3 ms / rotation`

- So it may take 4.2 ms **on average** to rotate to target (0.5 * 8.3 ms)

Seek, Rotate, Transfer

- Relatively fast
  - Depends on RPM and sector density
- 100+ MB/s is typical for SATA I (1.5Gb/s max)
  - Up to 600MB/s for SATA III (6.0Gb/s)
- `1s / 100MB = 10ms / MB = 4.9us/sector`
  - Assuming 512-byte sector

Workloads

- Seeks and rotations are slow while transfer is relatively fast

- What kind of workload is best suited for disks?

Workloads

- Seeks and rotations are slow while transfer is relatively fast

- What kind of workload is best suited for disks?
  - Sequential I/O: access sectors in order (transfer dominated)

- Random workloads access sectors in a random order (seek+rotation dominated)
  - Typically slow on disks
  - Never do random I/O unless you must! E.g., Quicksort is a terrible algorithm for disk!

Disk Performance Calculation

- Seagate Enterprise SATA III HDD

Metric	Perf
RPM	7200
Avg seek	4.16ms
Max transfer	500MB/s

- How long does an average 4KB read take?

Disk Performance Calculation

- Seagate Enterprise SATA III HDD

Metric	Perf
RPM	7200
Avg seek	4.16ms
Max transfer	500MB/s

- How long does an average 4KB read take?

$$transfer = \frac{1\ sec}{500\ MB} \times 4\ KB \times \frac{1{,}000{,}000\ us}{1\ sec} = 8\ us$$

Disk Performance Calculation

- Seagate Enterprise SATA III HDD

Metric	Perf
RPM	7200
Avg seek	4.16ms
Max transfer	500MB/s

- How long does an average 4KB read take?

$$transfer = \frac{1\ sec}{500\ MB} \times 4\ KB \times \frac{1{,}000{,}000\ us}{1\ sec} = 8\ us$$

```
Latency = 4.16 ms + 4.2 ms + 8 us =
8.368 ms
```

Avg Seek

Avg Rotate

Q: Given a stream of I/O requests, in what order should they be served?

Disk Scheduling

Disk Scheduling

- OS is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and high disk bandwidth utilization

- Strategy: reorder requests to meet some goal
  - Performance (e.g., by making I/O sequential)
  - Fairness
  - Consistent latency

- Usually implemented in both OS and hardware

Disk Scheduling

- Performance objective: minimize seek+rotation time
  - Minimize the distance the head needs to go

- Disk bandwidth:
  - The total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling

- There are many sources of disk I/O requests:
  - OS
  - System processes
  - User processes
- I/O request:
  - Read/write mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device

- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms make sense only when a queue exists

Disk Scheduling

- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying "depth")

- Disk scheduling algorithms:
  - Algorithms that schedule the orders of disk I/O requests

Disk Scheduling

- Disk scheduling algorithms:
  - Algorithms that schedule the orders of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with an example request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

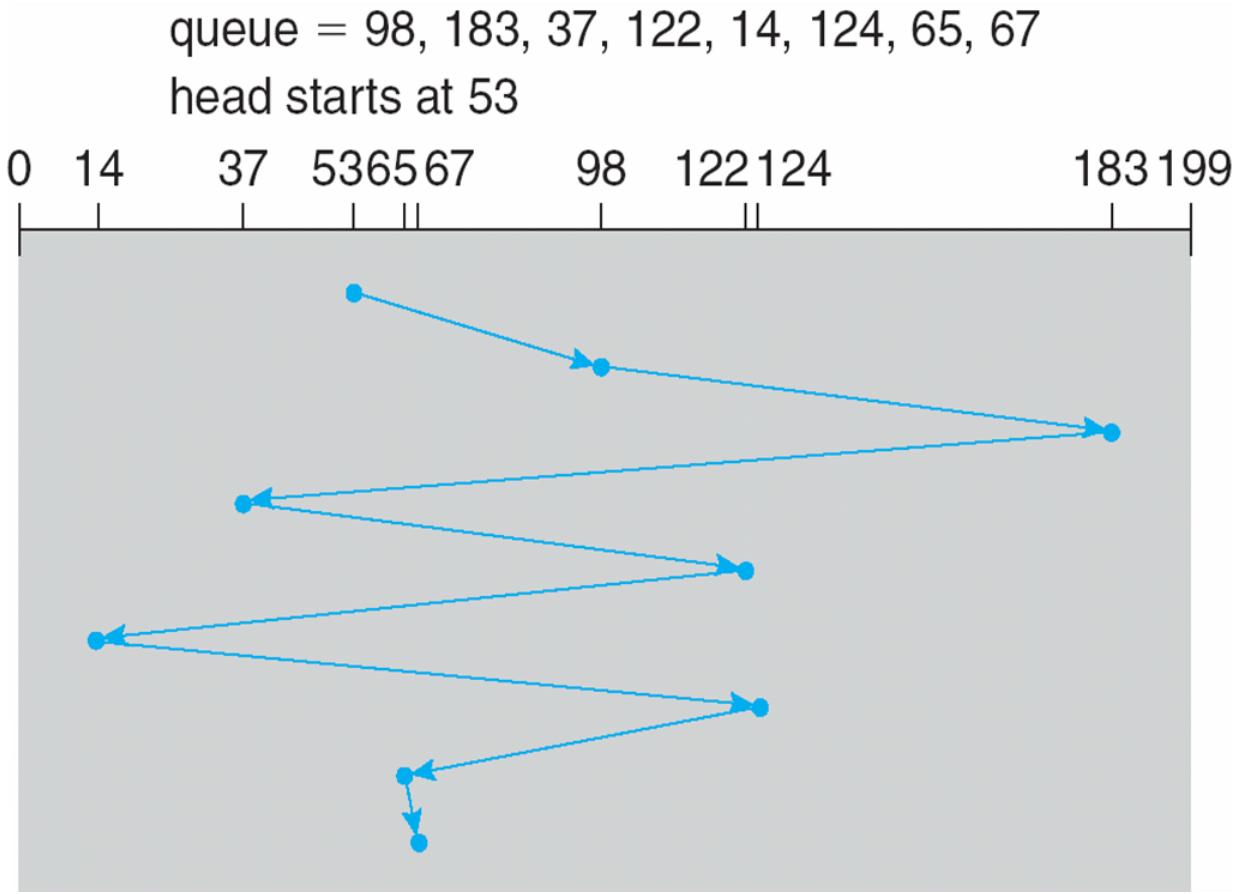
Initially, head pointer pointing to 53

FIFO

- Idea: Serve the I/O request in the order they arrive

FIFO

- Idea: Serve the I/O request in the order they arrive



FIFO

- Idea: Serve the I/O request in the order they arrive

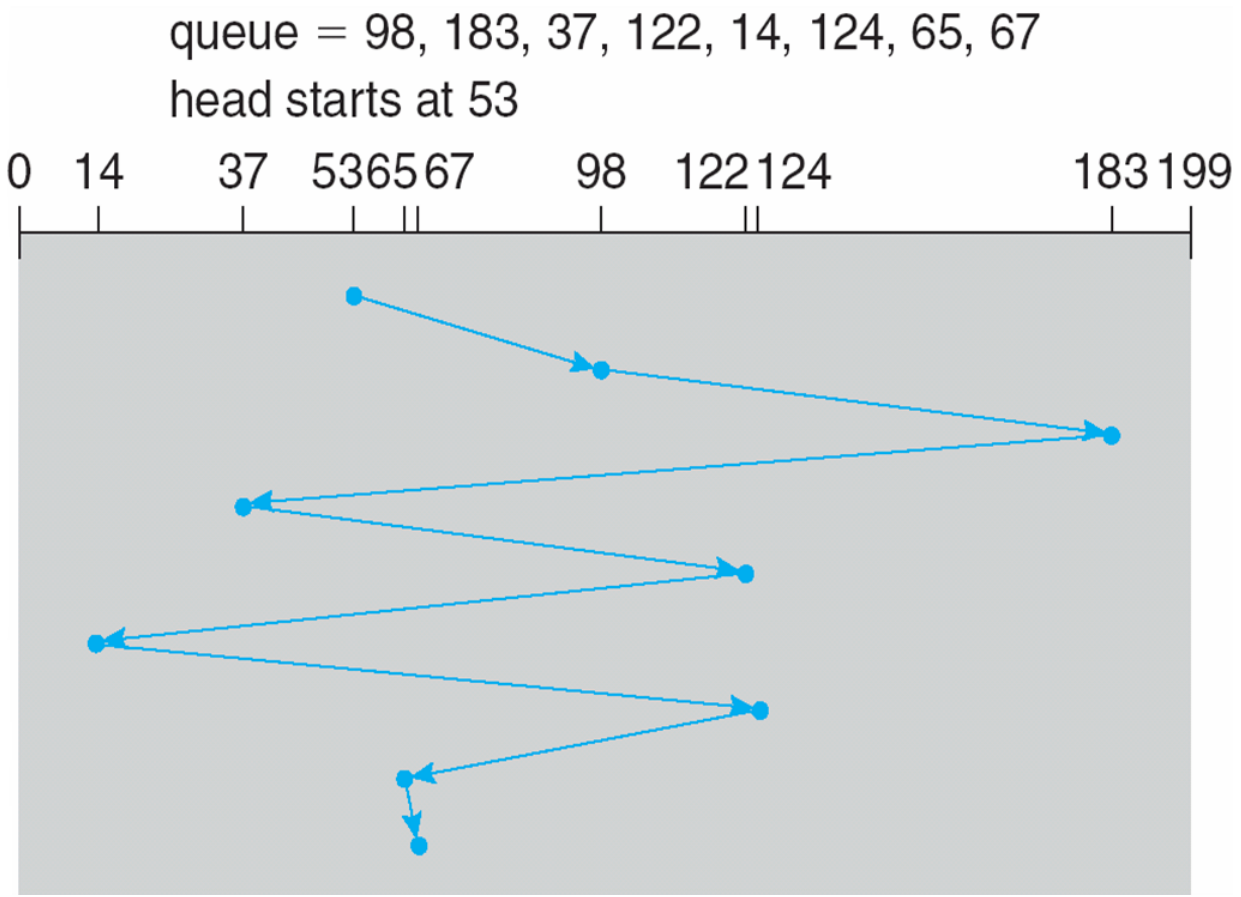


Illustration shows total head movement of **640** cylinders

Shortest Positioning Time First (SPTF)

- Idea: Selects the request that will take the least time for seeking and rotating

- Also called Shortest Seek Time First (SSTF) if rotational positioning is not considered

Shortest Positioning Time First (SPTF)

- Idea: Selects the request that will take the least time for seeking and rotating

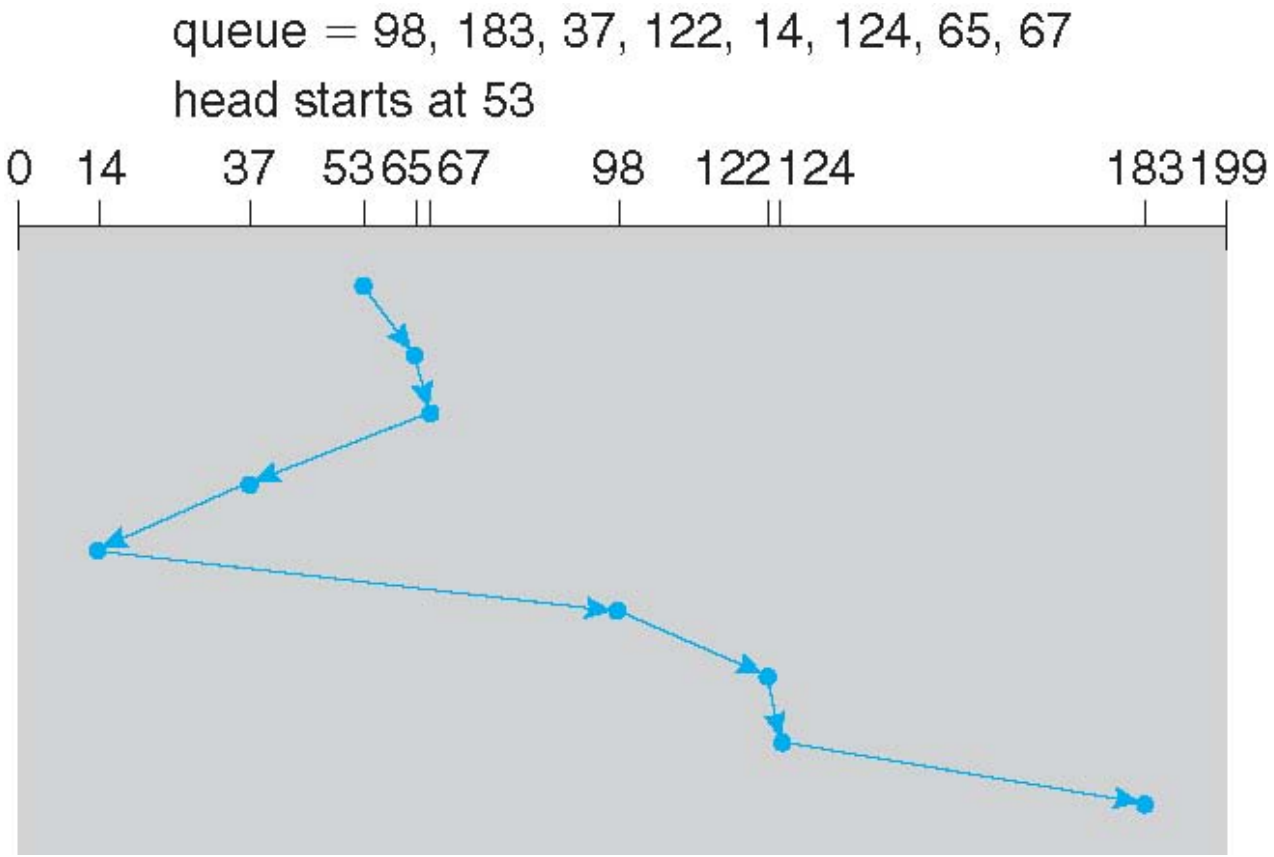


Illustration shows total head movement of **236** cylinders

Greedy algorithm: Similar to SJF: may cause starvation of some requests!

SCAN

- Idea: Sweep back and forth, from one end of disk to the other, serving requests as you go
  - The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues
  - AKA Elevator Algorithm

SCAN

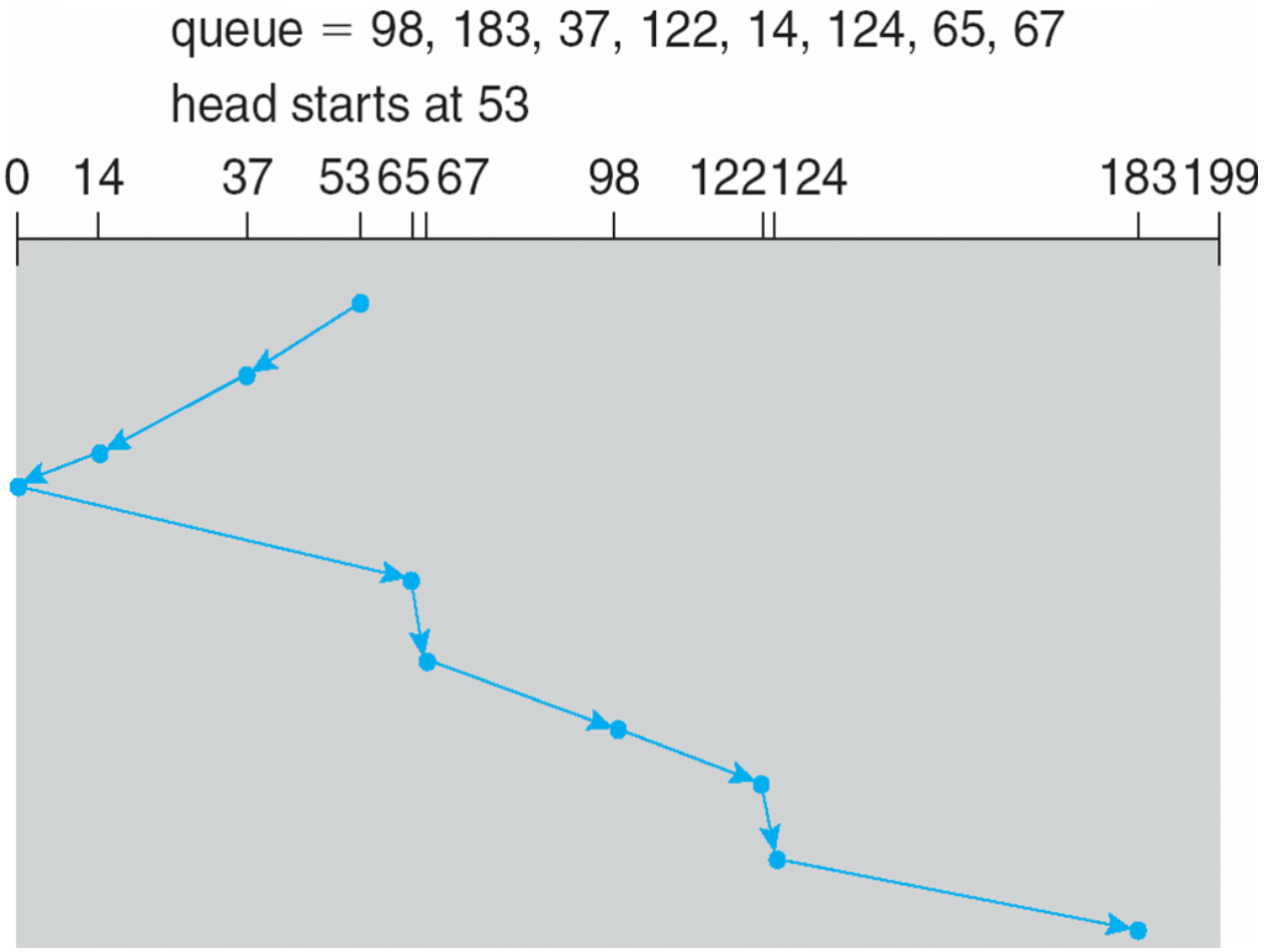


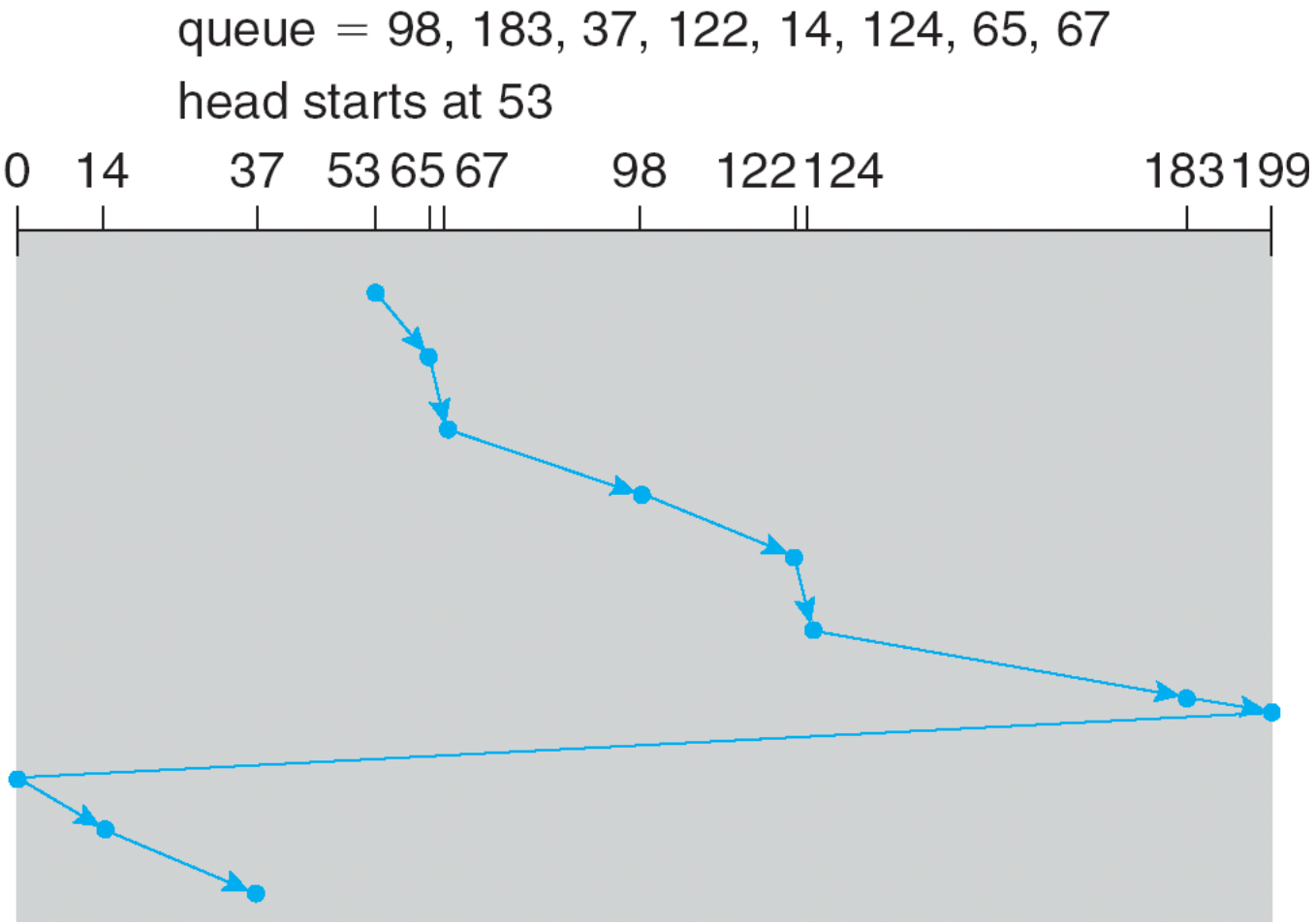
Illustration shows total head movement of **236** cylinders

Issue: Cylinders in the middle get better service;
Requests at the other end wait the longest!

C-SCAN (Circular-SCAN)

- Idea: Only sweep in ONE direction
  - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Provides a more uniform wait time than SCAN
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

C-SCAN

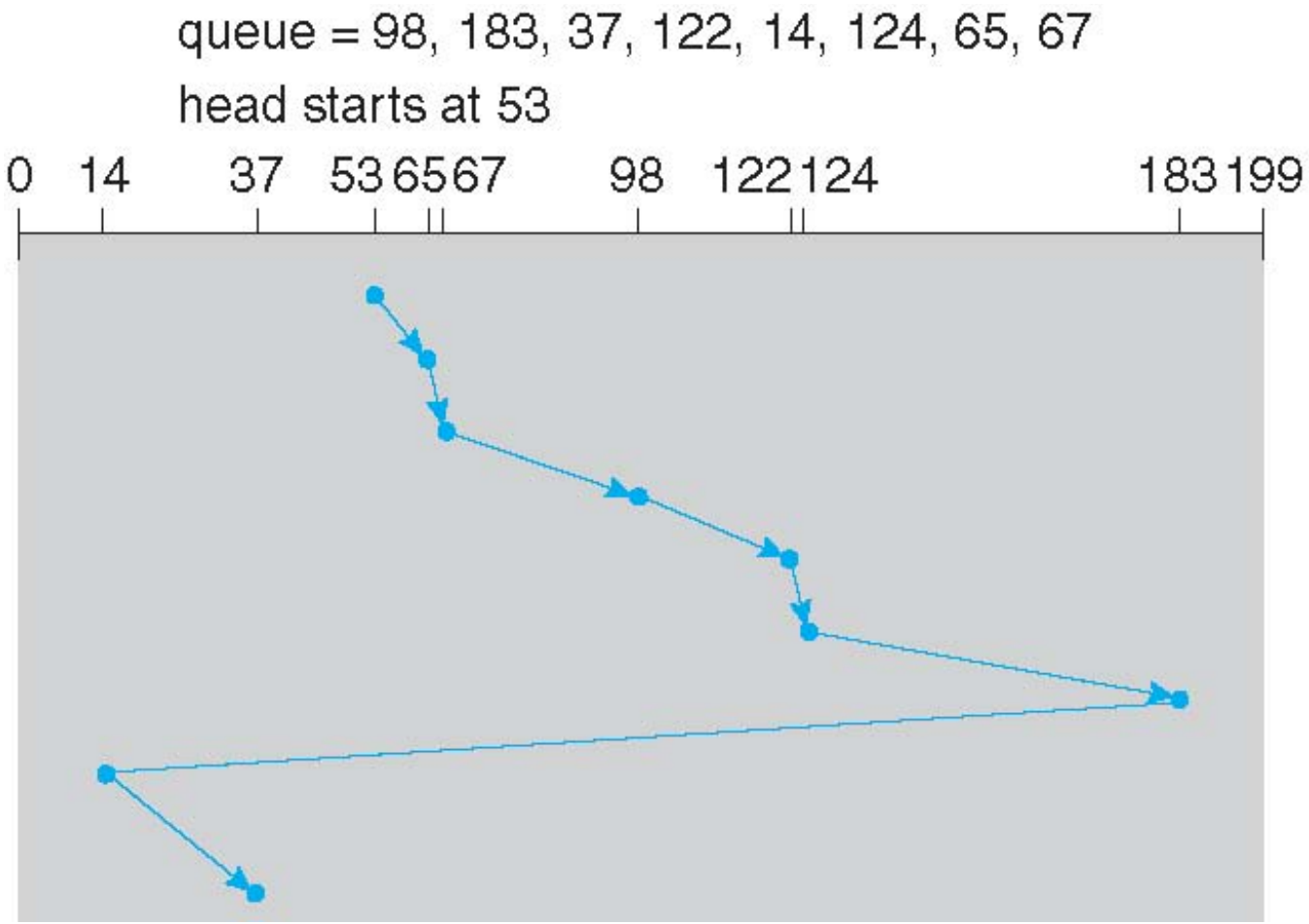


Total number of cylinders?

C-LOOK

- Idea: Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk
  - LOOK: A version of SCAN
  - C-LOOK: A version of C-SCAN

C-LOOK



Total number of cylinders?

Work Conservation

- Work conserving schedulers always try to do I/O if there's I/O to be done
- Sometimes, it's better to wait (delay) instead if you anticipate another request will appear nearby
- Such non-work-conserving schedulers are called anticipatory schedulers

Selecting A Disk Scheduling Algorithm

- SPTF is common and has a natural appeal
  - Starvation
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the workload (i.e., number and types of requests)
- The disk scheduling algorithm should be written as a separate OS module, allowing it to be replaced with a different algorithm if necessary
- Requests for disk service can be impacted by the file-allocation method/pattern
  - as well as metadata layout – topic of file systems ⟵ Next lecture

Solid State Drives (SSDs)

Disk Recap

- I/O requires: seek, rotate, transfer

- Inherently:
  - Not parallel (only one head)
  - Slow (mechanical)
  - Poor random I/O (locality around disk head)

- Random requests each taking ~10+ ms

SSD Overview

- Hold charge in cells. No moving (mechanical) parts (no seeks)!
  - SSDs use transistors (just like DRAM), but SSD data persists when the power goes out
  - NAND-based flash is the most popular technology, so we'll focus on it
- SSD is Inherently parallel!

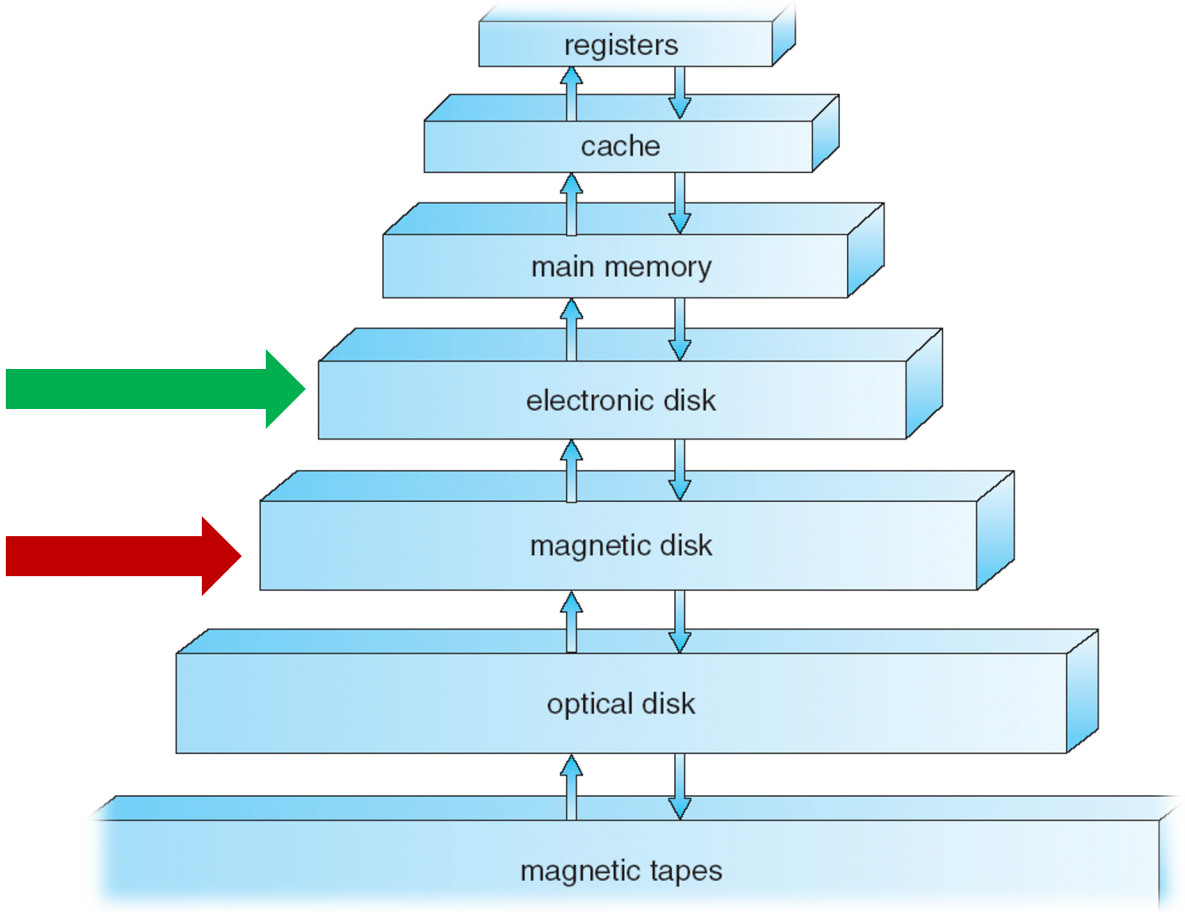
SSD Overview

- Hold charge in cells. No moving (mechanical) parts (no seeks)!
  - SSDs use transistors (just like DRAM), but SSD data persists when the power goes out
  - NAND-based flash is the most popular technology, so we'll focus on it
- SSD is Inherently parallel!
- High-level takeaways
  1. SSDs have a higher $/bit than HDDs, but better performance
  2. SSDs handle writes in a strange way; this has implications for file system design

Storage Hierarchy Overview

SSD:
Smaller capacity
Higher $/bit
Faster accesses

HDD:
Larger capacity
Lower $/bit
Way slower accesses



Disk vs. SSD: Performance

- Throughput
  - Disk: ~130MB/s (sequential)
  - Flash: ~400MB/s

- Latency
  - Disk: ~10ms (one op)
  - Flash:
    - Read: 10-50us
    - Program: 200-500us
    - Erase: 2ms

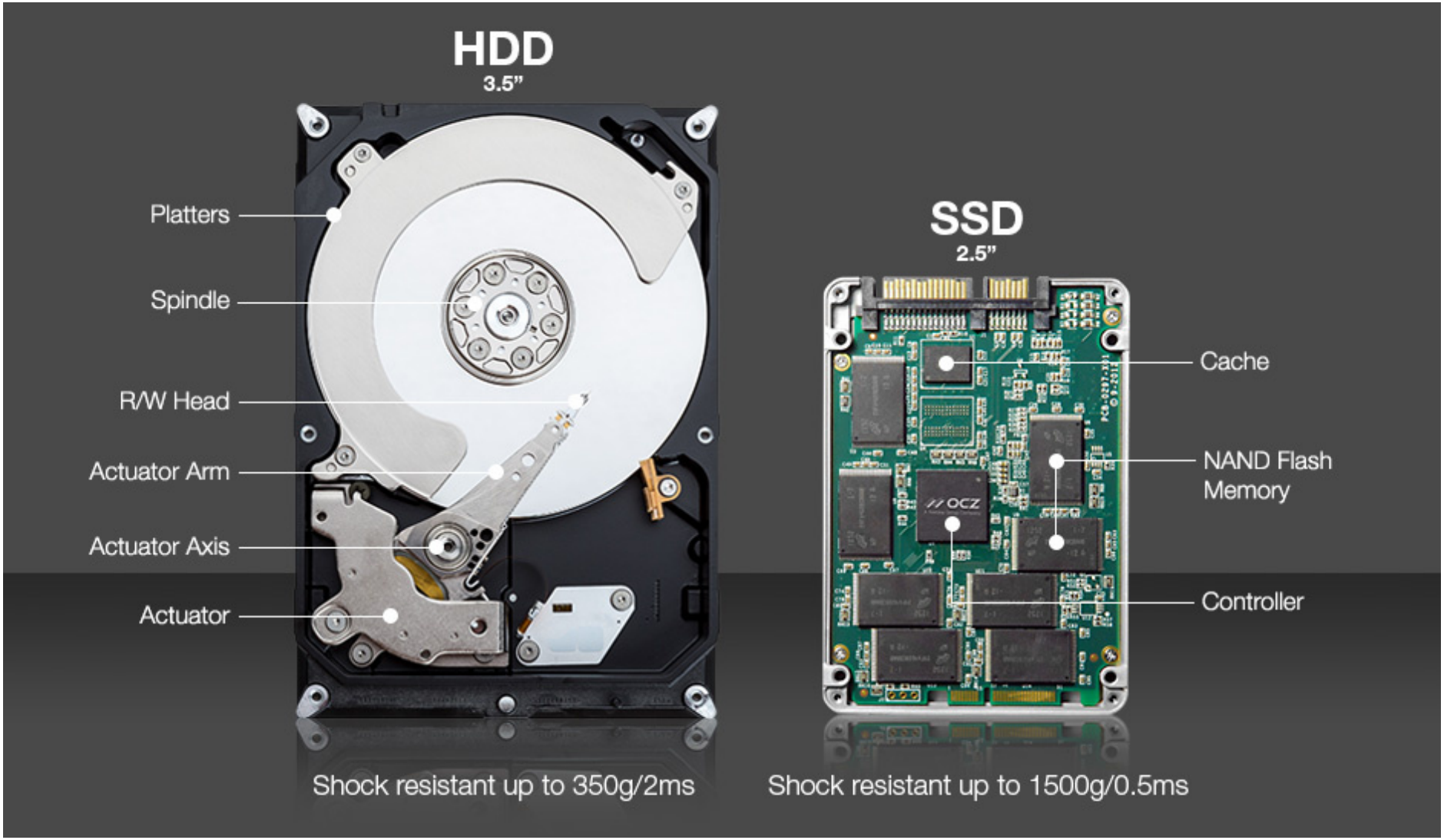
Disk vs. SSD: Performance

- Throughput
  - Disk: ~130MB/s (sequential)
  - Flash: ~400MB/s

- Latency
  - Disk: ~10ms (one op)
  - Flash:
    - Read: 10-50us
    - Program: 200-500us
    - Erase: 2ms

Types of write, more later…

Disk vs. SSD: Internal



* https://www.enterprisestorageforum.com/storage-hardware/ssd-vs-hdd.html

Disk vs. SSD: Capacity

"An obvious question is why are we talking about spinning disks at all, rather than SSDs, which have higher IOPS and are the "future" of storage. The root reason is that the cost per GB remains too high, and more importantly that the growth rates in capacity/$ between disks and SSDs are relatively close (at least for SSDs that have sufficient numbers of program-erase cycles to use in data centers), so that cost will not change enough in the coming decade. We do make extensive use of SSDs, but primarily for high performance workloads and caching, and this helps disks by shifting seeks to SSDs.

~ Eric Brewer et al. Google

Source: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44830.pdf

	SSD	HDD
Price	$0.25-$0.27 per GB average	$0.2-$0.03 per GB average

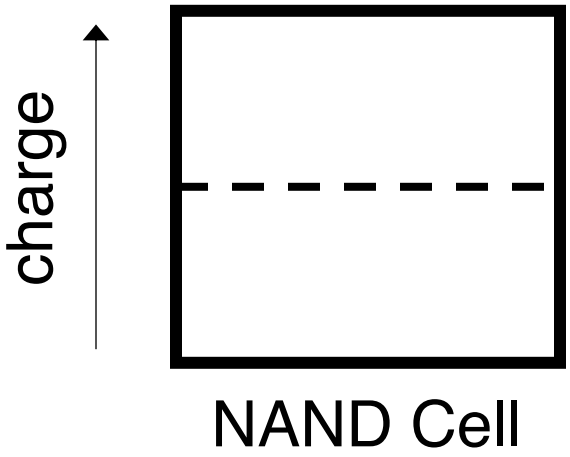
Disk vs. SSD: Summary

	SSD	HDD
Price	$0.25-$0.27 per GB average	$0.2-$0.03 per GB average
Lifespan	30-80% test developed bad block in their lifetime	3.5% developed bad sectors comparatively
Ideal for	High performance processing Residing in APA or Tier 0/1 media in hybrid arrays	High capacity nearline tiers Long-term retained data
Read/write speeds	200 MB/s to 2500 MB/s	up to 200 MB/s
Benefits	Higher performance for faster read/write operations and fast load times	Less expensive Mature technology and massive installed user base
Drawbacks	May not be as durable/reliable as HDDs Not good for long-term archival data	Mechanical components take longer to read-write than SSDs

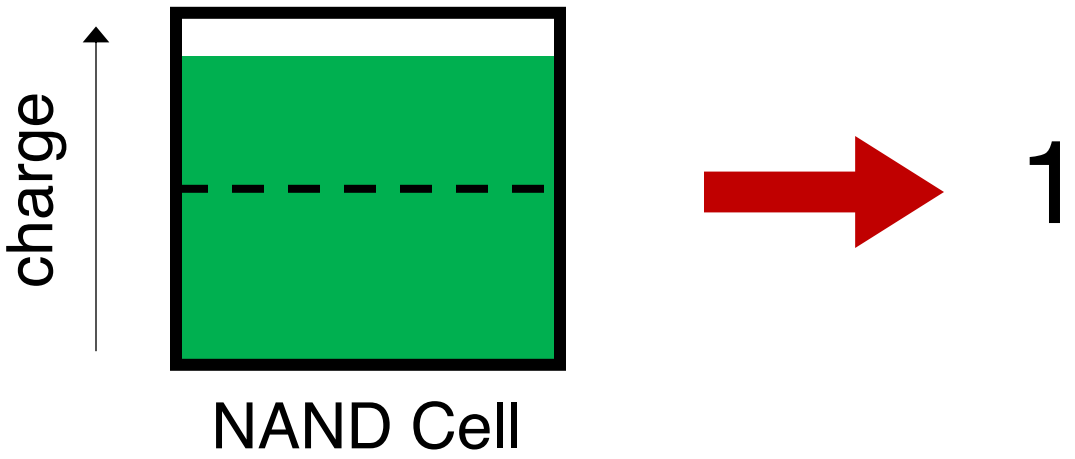
* https://www.enterprisestorageforum.com/storage-hardware/ssd-vs-hdd.html

SSD Architecture

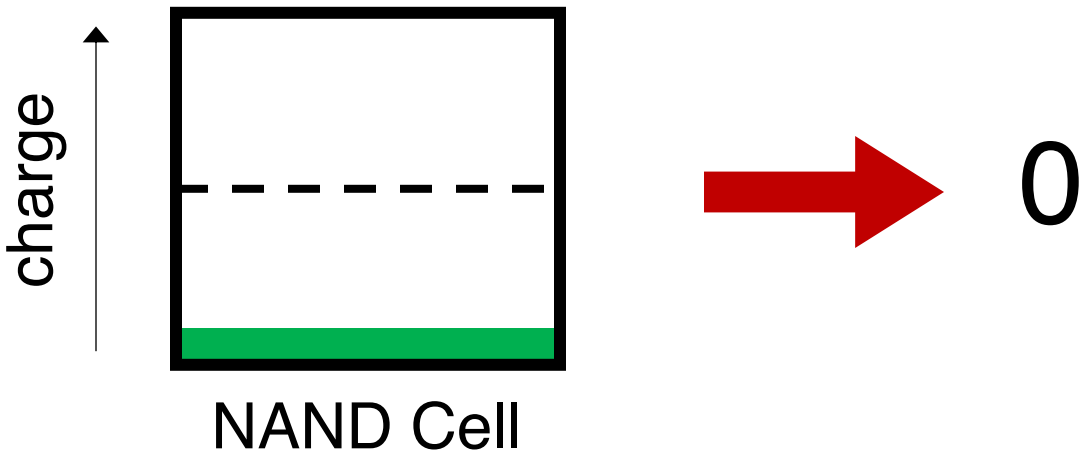
SLC: Single-Level Cell



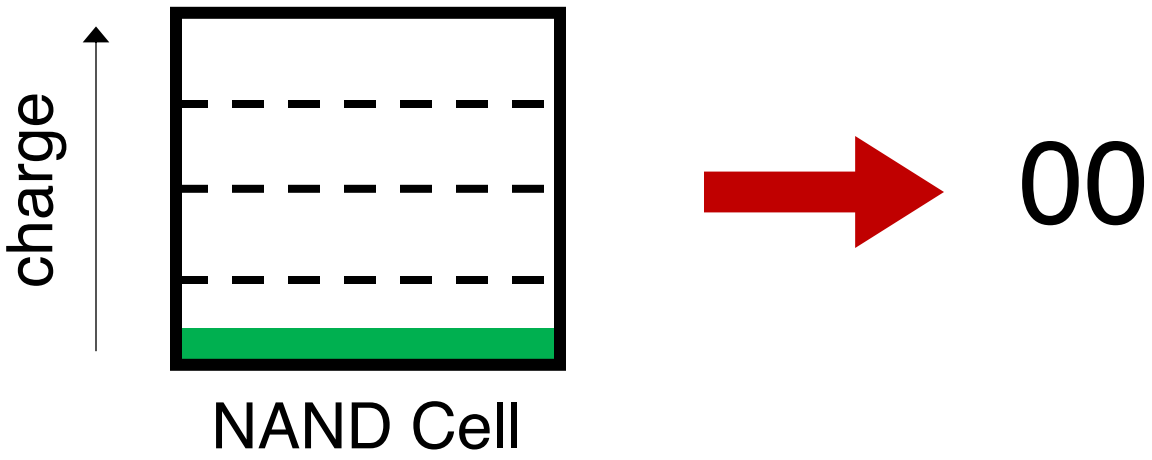
SLC: Single-Level Cell



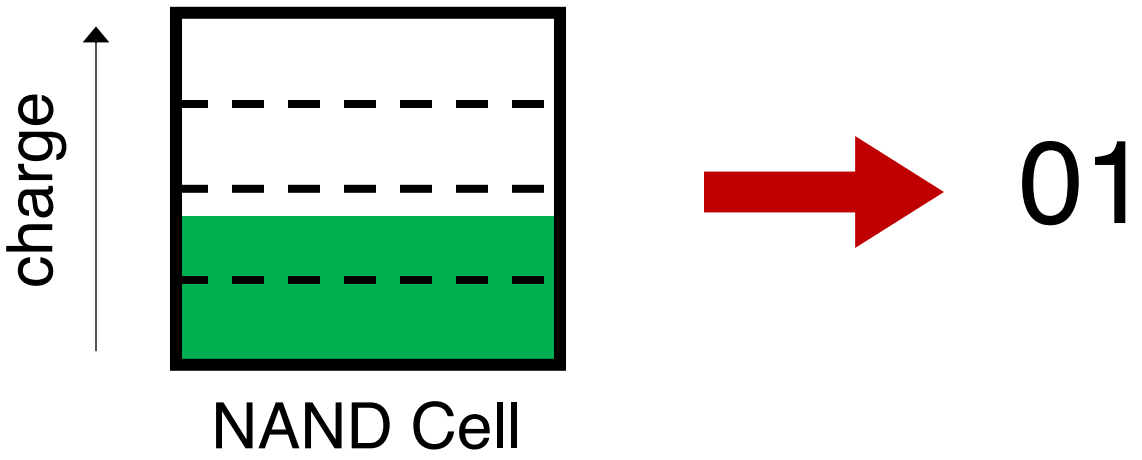
SLC: Single-Level Cell



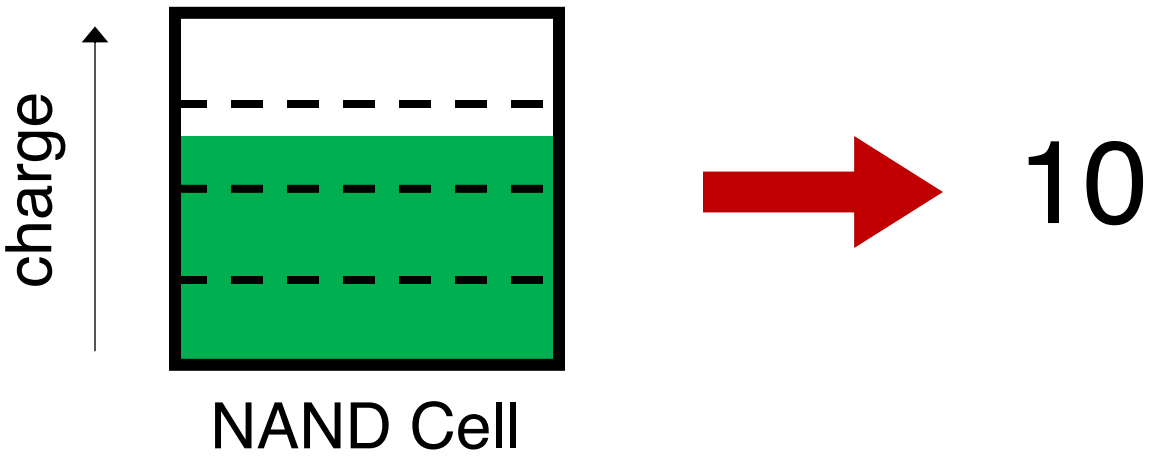
MLC: Multi-Level Cell



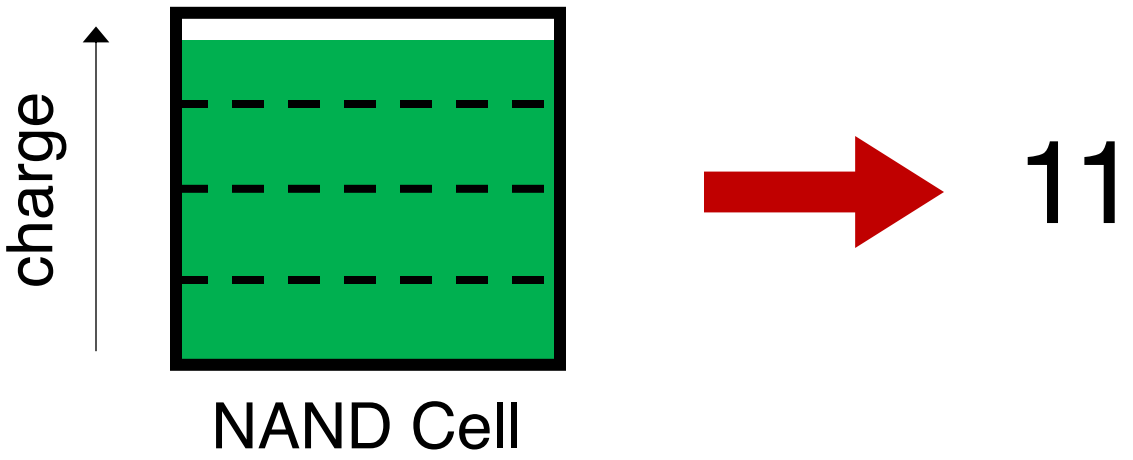
MLC: Multi-Level Cell



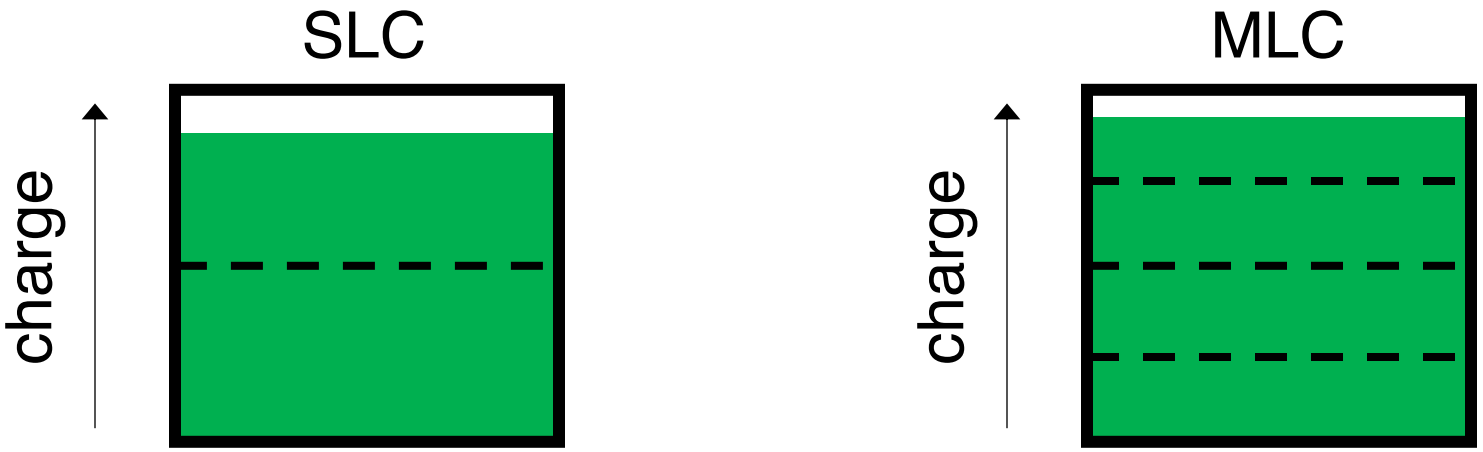
MLC: Multi-Level Cell



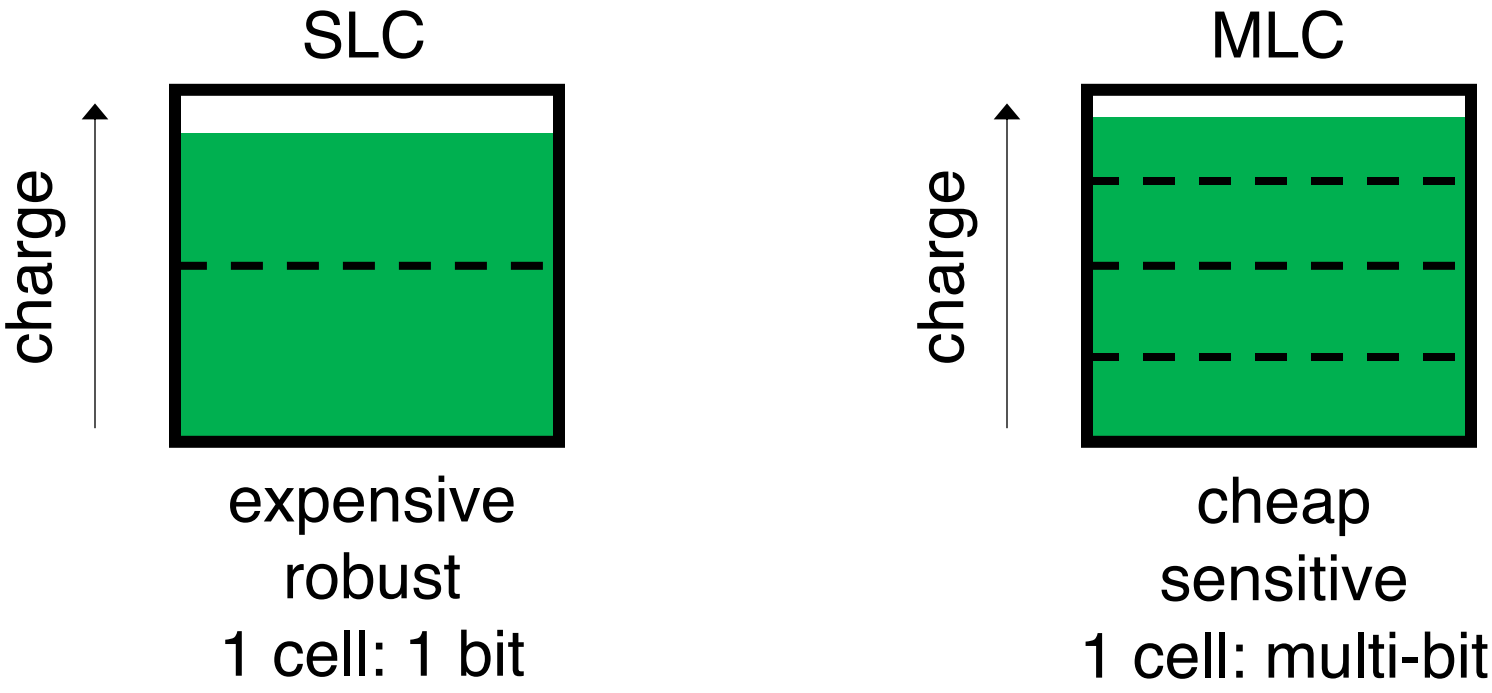
MLC: Multi-Level Cell



Single- vs. Multi-Level Cell



Single- vs. Multi-Level Cell



Wearout

- Problem: flash cells wear out after being erased too many times

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

- Usage strategy: ???

Wearout

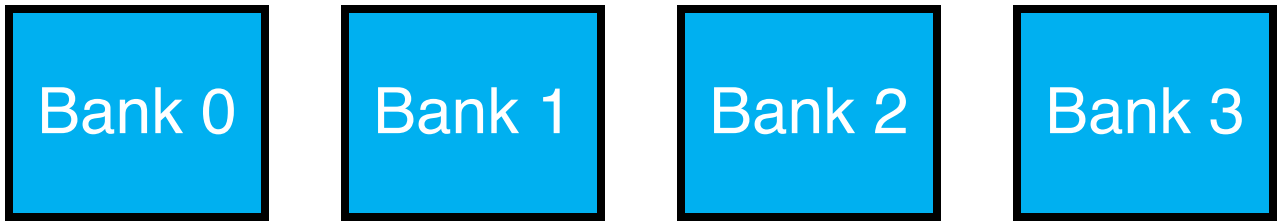
- Problem: flash cells wear out after being erased too many times

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

- Usage strategy: wear leveling
  - Prevents some cells from being wornout while others still fresh

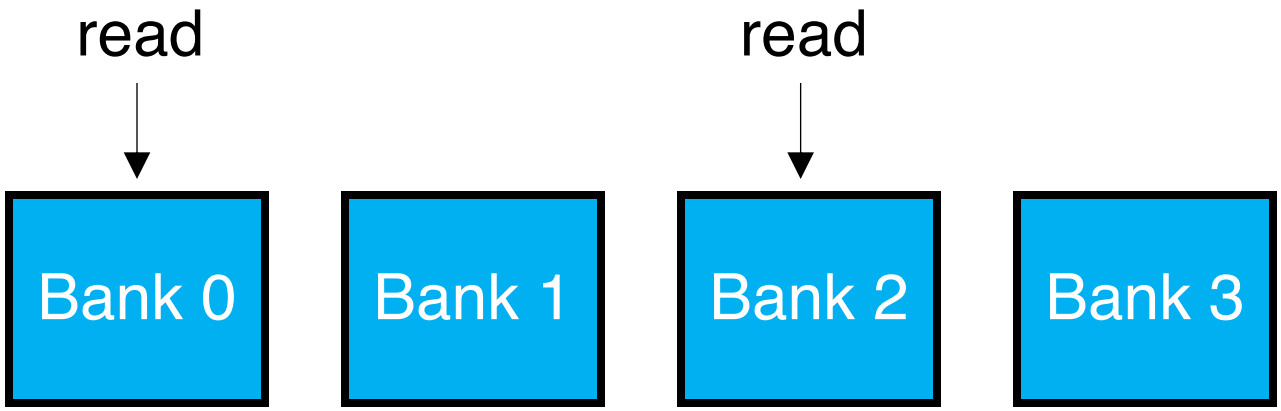
Banks

- SSD devices are divided into banks (aka. planes)
- Banks can be accessed in parallel



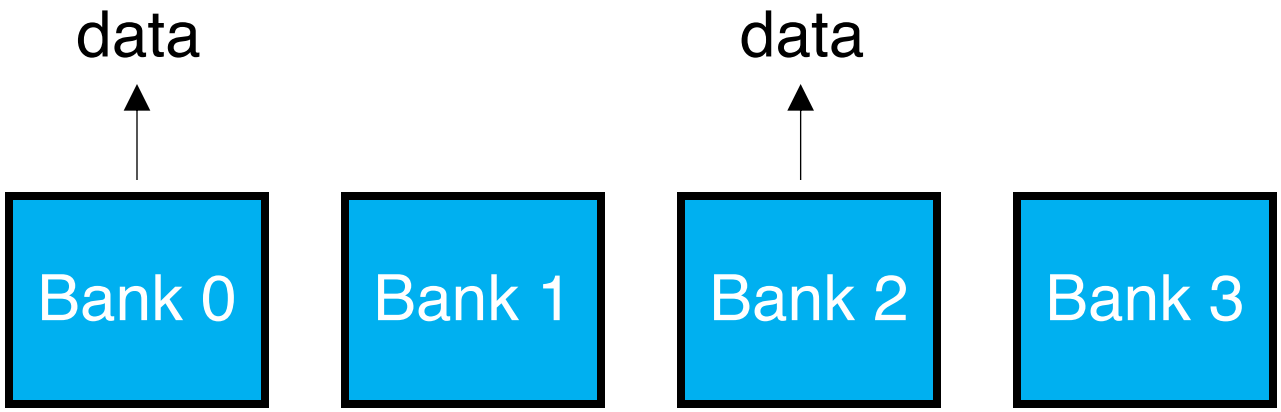
Banks

- SSD devices are divided into banks (aka. planes)
- Banks can be accessed in parallel



Banks

- SSD devices are divided into banks (aka. planes)
- Banks can be accessed in parallel



SSD Writes

- Writing 0's
  - Fast, fine-grained
- Writing 1's
  - Slow, coarse-grained

SSD Writes

- Writing 0's
  - Fast, fine-grained
  - called "program"
- Writing 1's
  - Slow, coarse-grained
  - called "erase"

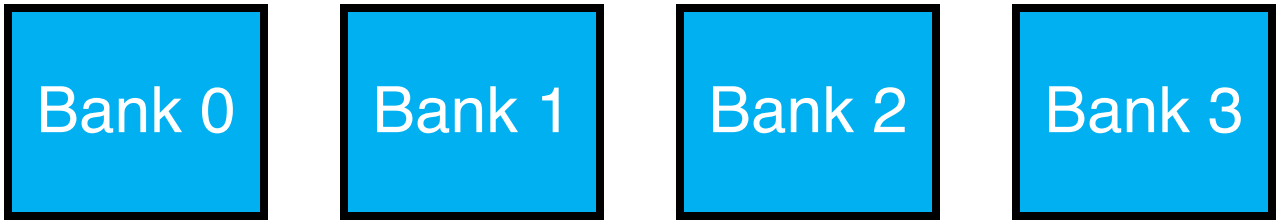
SSD Writes

- Writing 0's
  - Fast, fine-grained [page-level]
  - called "program"
- Writing 1's
  - Slow, coarse-grained [block-level]
  - called "erase"

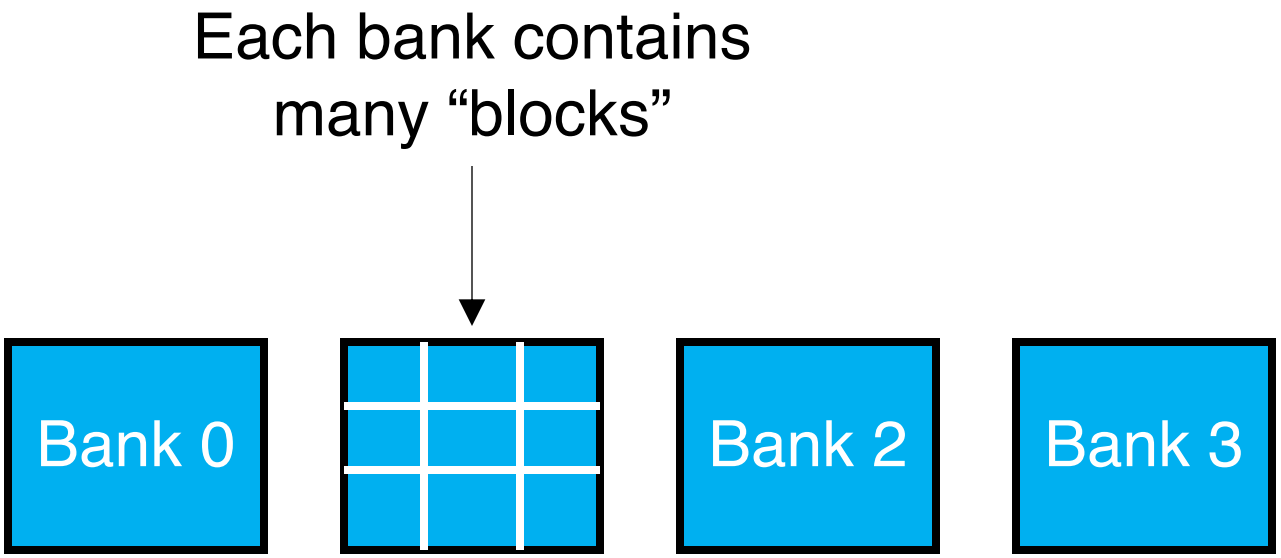
SSD Writes

- Writing 0's
  - Fast, fine-grained [page-level]
  - called "program"
- Writing 1's
  - Slow, coarse-grained [block-level]
  - called "erase"
- Flash can only "write" (program) into clean pages
  - "clean": pages containing all 1's (pages that have been erased)
  - Flash does not support in-place overwrite!

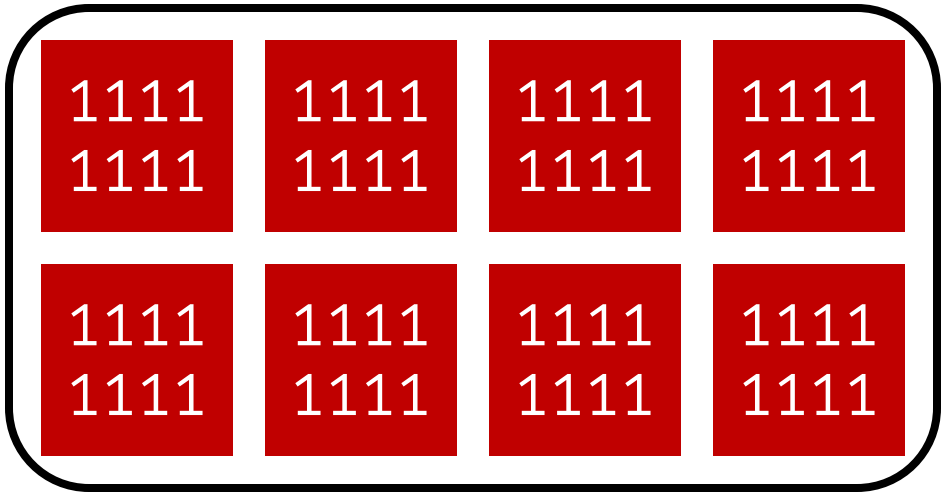
Banks and Blocks



Banks and Blocks

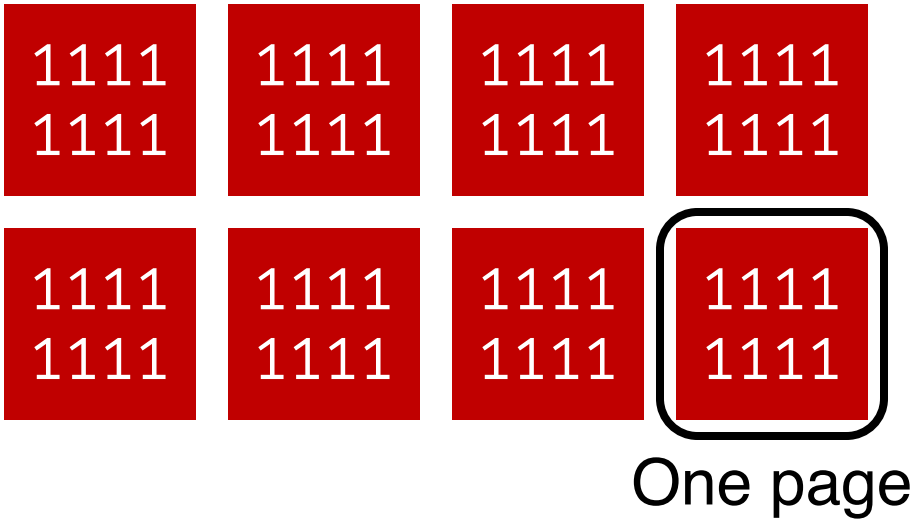


Block and Pages

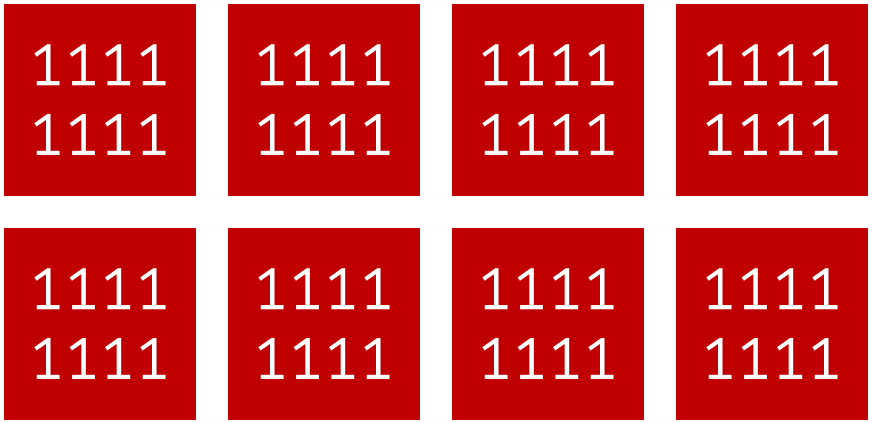


One block

Block and Pages

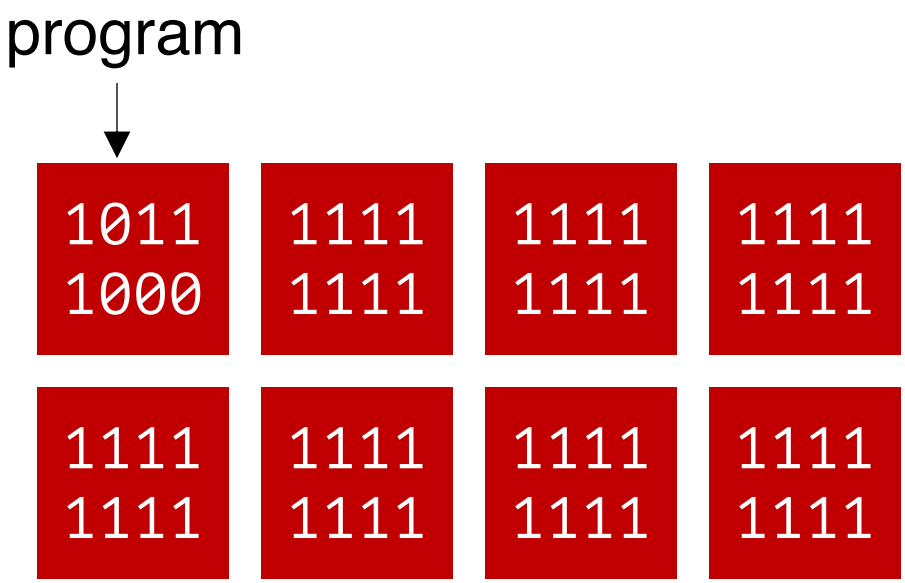


Block and Pages

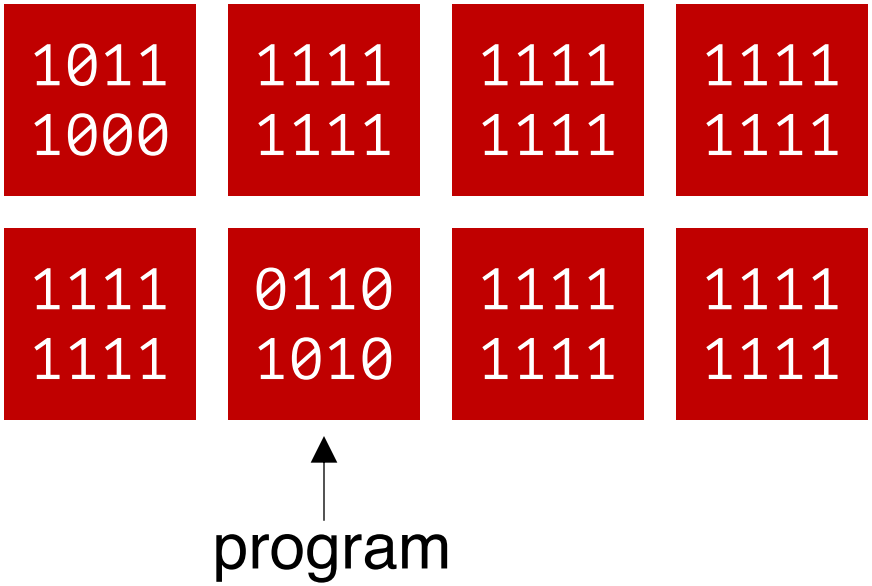


All pages are clean
("programmable")

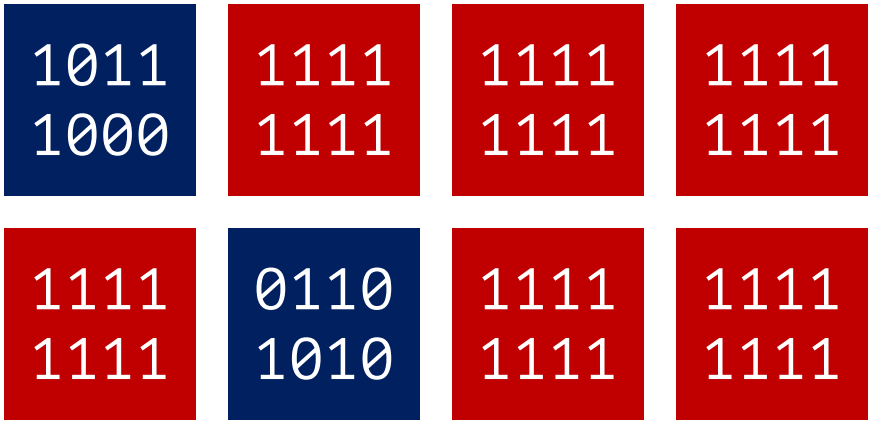
Block


program
1011
1000
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111

Block


1011
1000
1111
1111
1111
1111
1111
1111
1111
1111
0110
1010
1111
1111
1111
1111
program

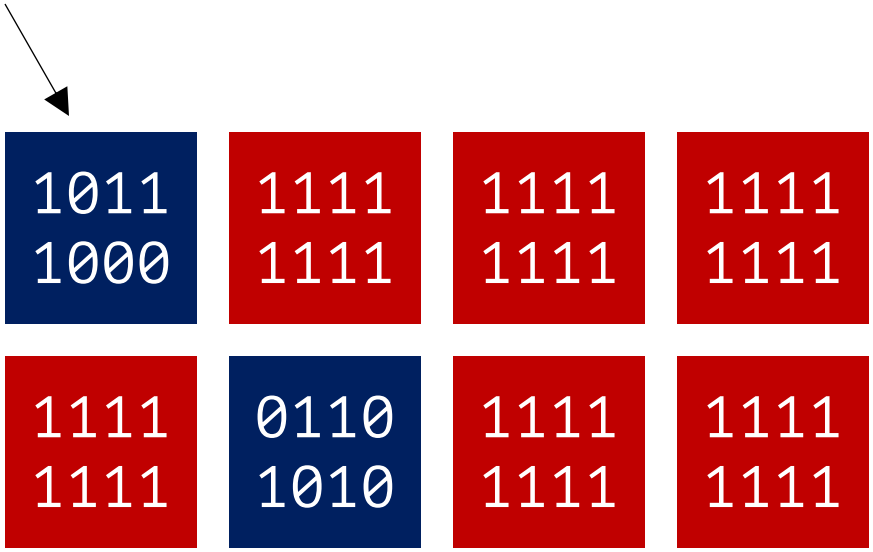
Block



Two pages hold data
(**cannot be overwritten**)

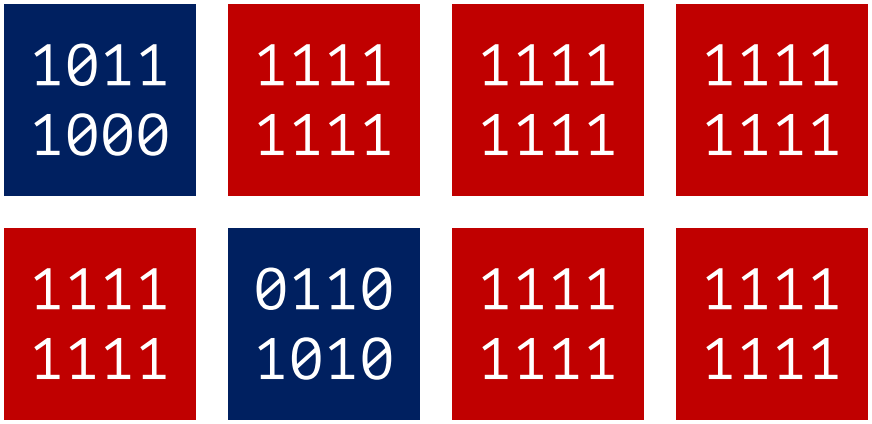
Block

still want to write data into this page???



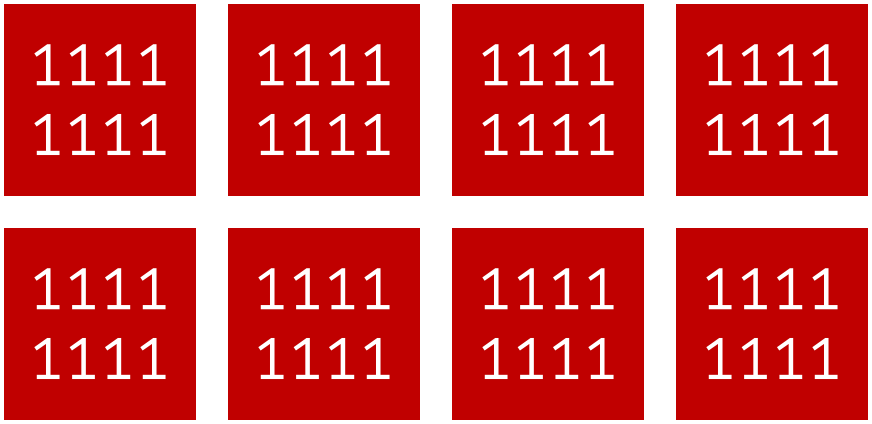
Two pages hold data
(**cannot be overwritten**)

Block


1011
1000
1111
1111
1111
1111
1111
1111
1111
1111
0110
1010
1111
1111
1111
1111

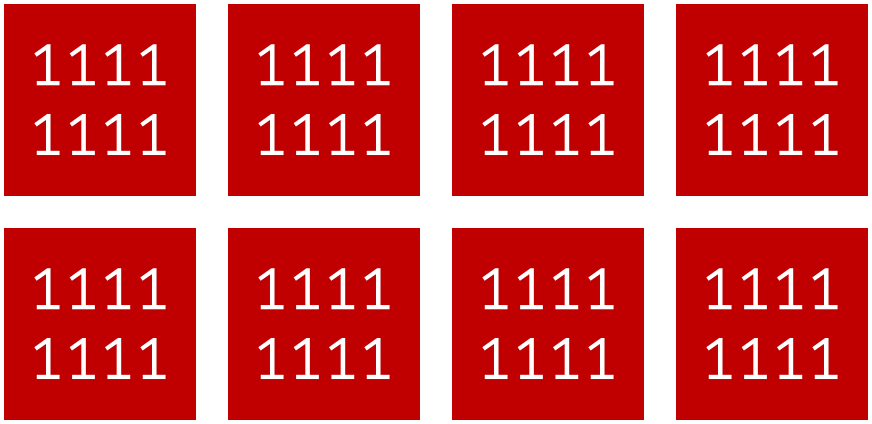
erase

Block



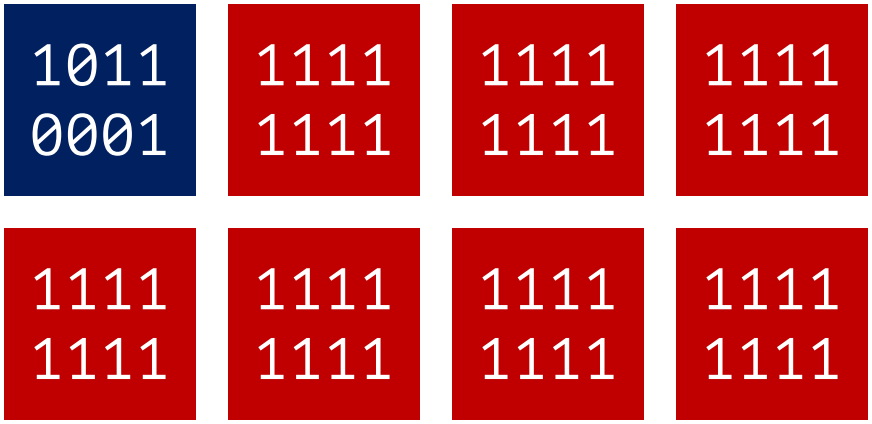
erase
(the whole block)

Block



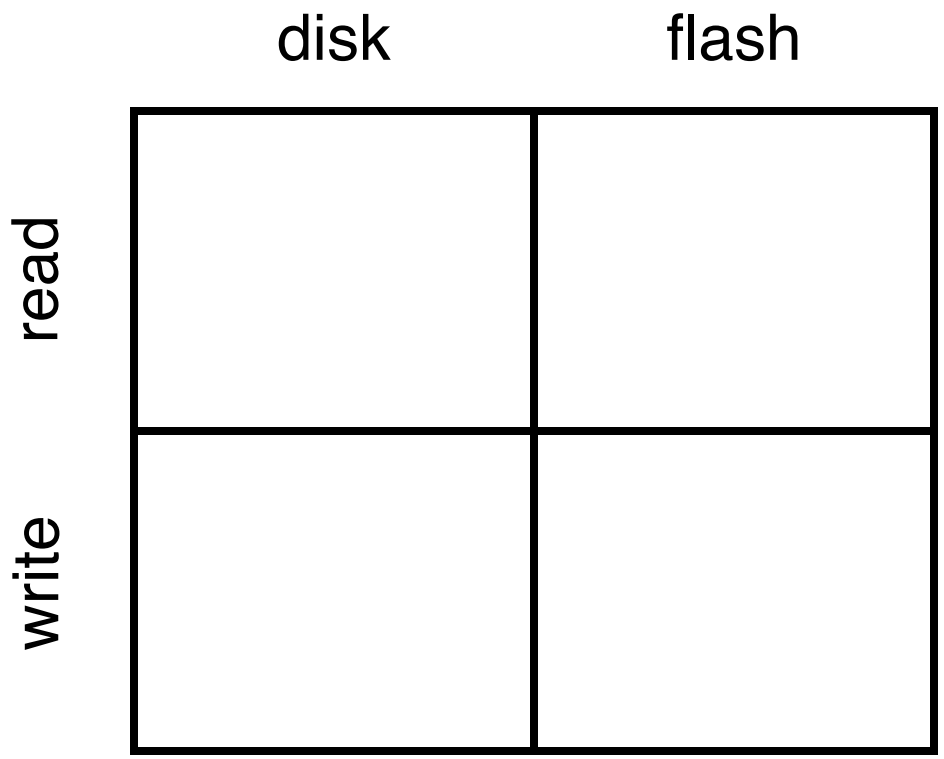
After erase, again, **free state**
(can write new data in any page)

Block

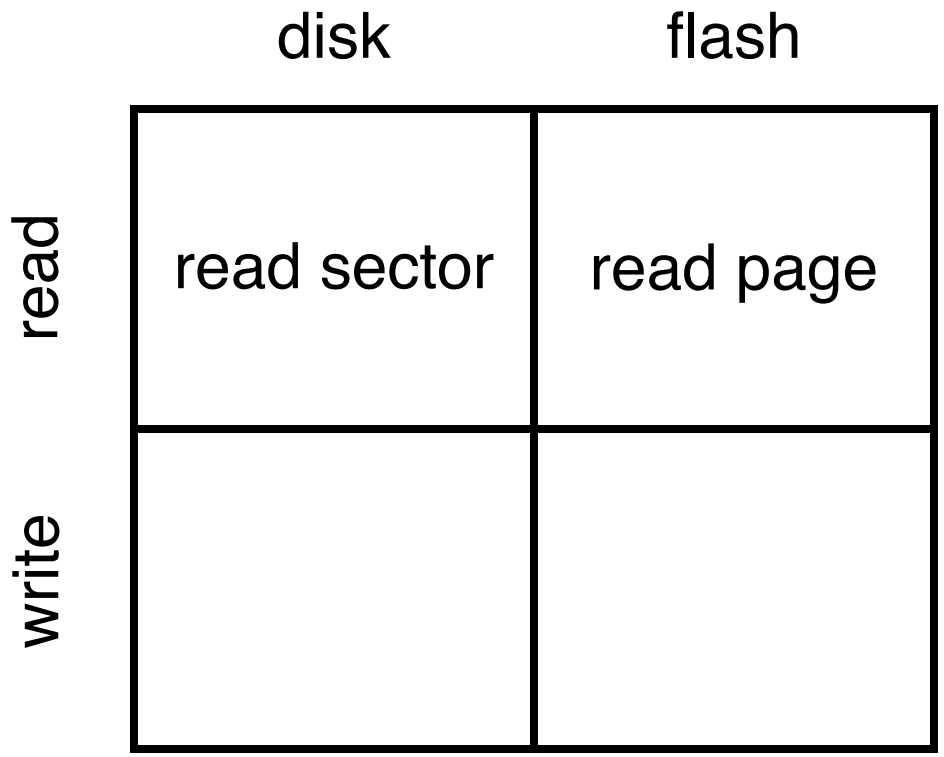


This dark blue page holds data

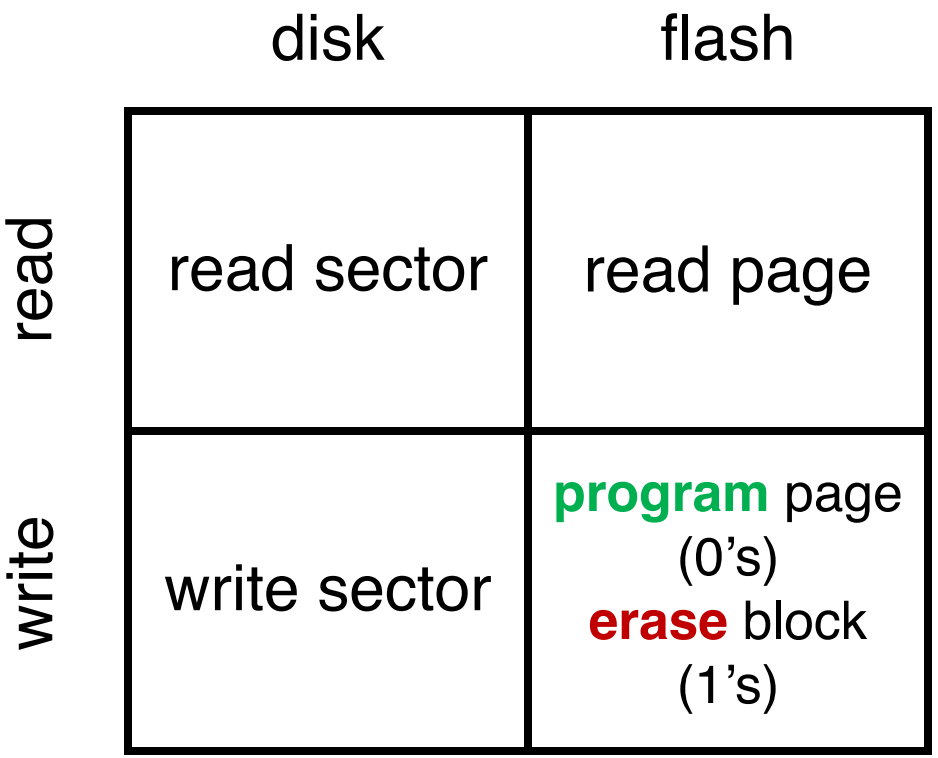
SSD vs. Disk: APIs



SSD vs. Disk: APIs



SSD vs. Disk: APIs



SSD Architecture

- Bank/plane: 1024 to 4096 blocks
  - Banks accessed in parallel
- Block: 64 to 256 pages
  - Unit of erase
- Page: 2 to 8 KB
  - Unit of read and program

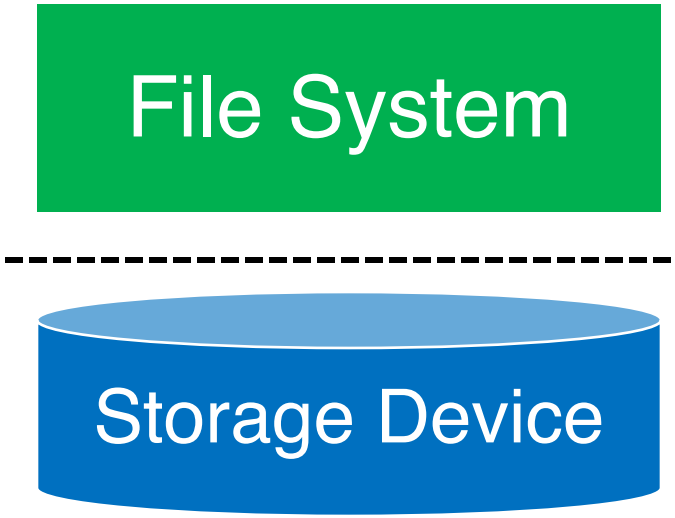
Disk vs. SSD: Performance

- Throughput
  - Disk: ~130MB/s (sequential)
  - Flash: ~400MB/s

- Latency
  - Disk: ~10ms (one op)
  - Flash:
    - Read: 10-50us
    - Program: 200-500us
    - Erase: 2ms

Working with File System

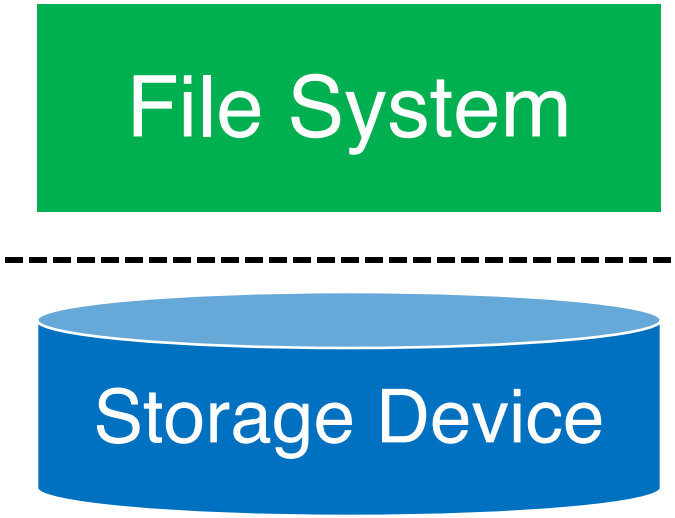
Traditional File Systems



Traditional API:

- read sector
- write sector

Traditional File Systems



Traditional API:

- read sector
- write sector

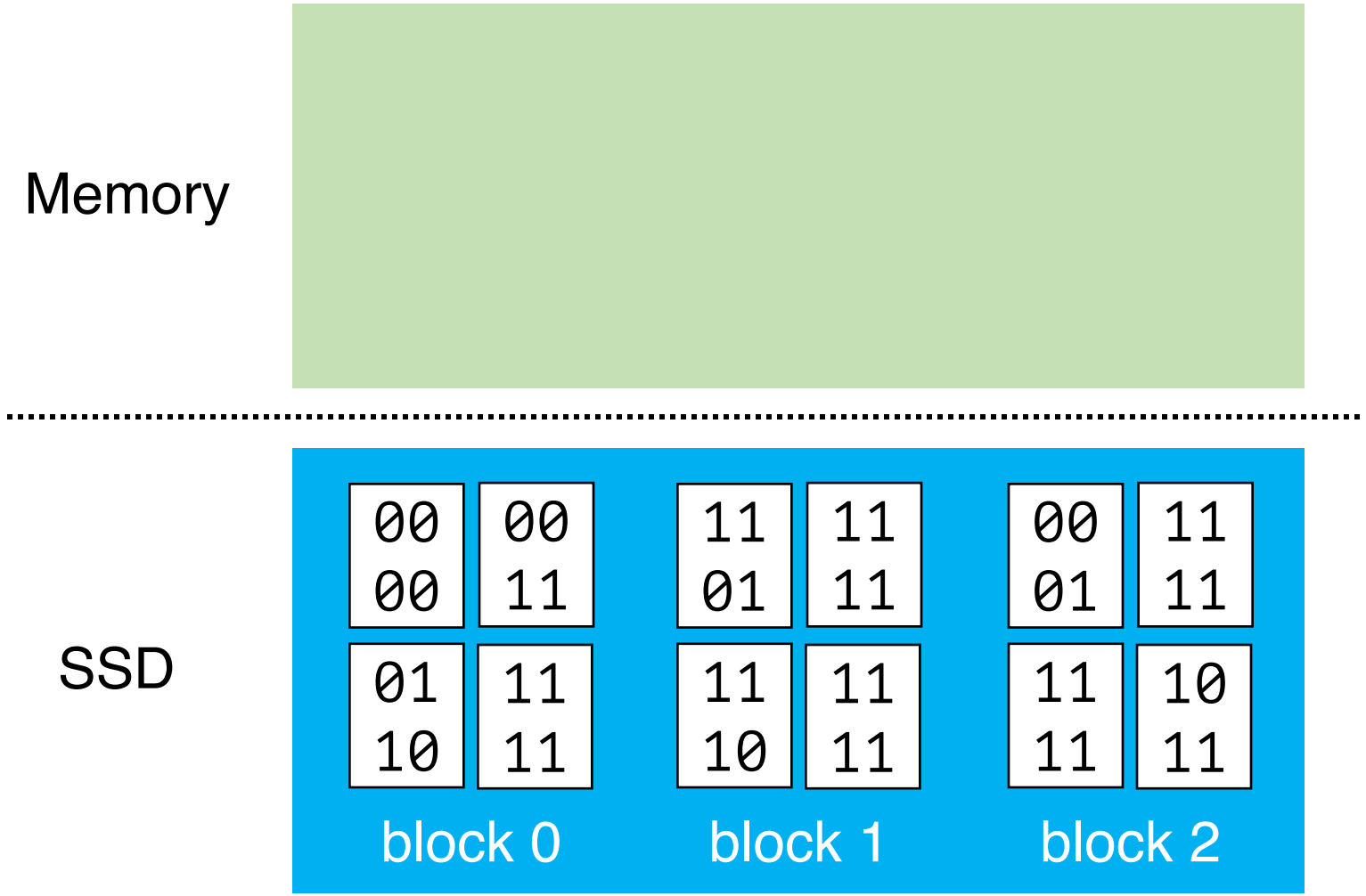
Mismatch with flash!

Traditional APIs wrapping around SSD APIs

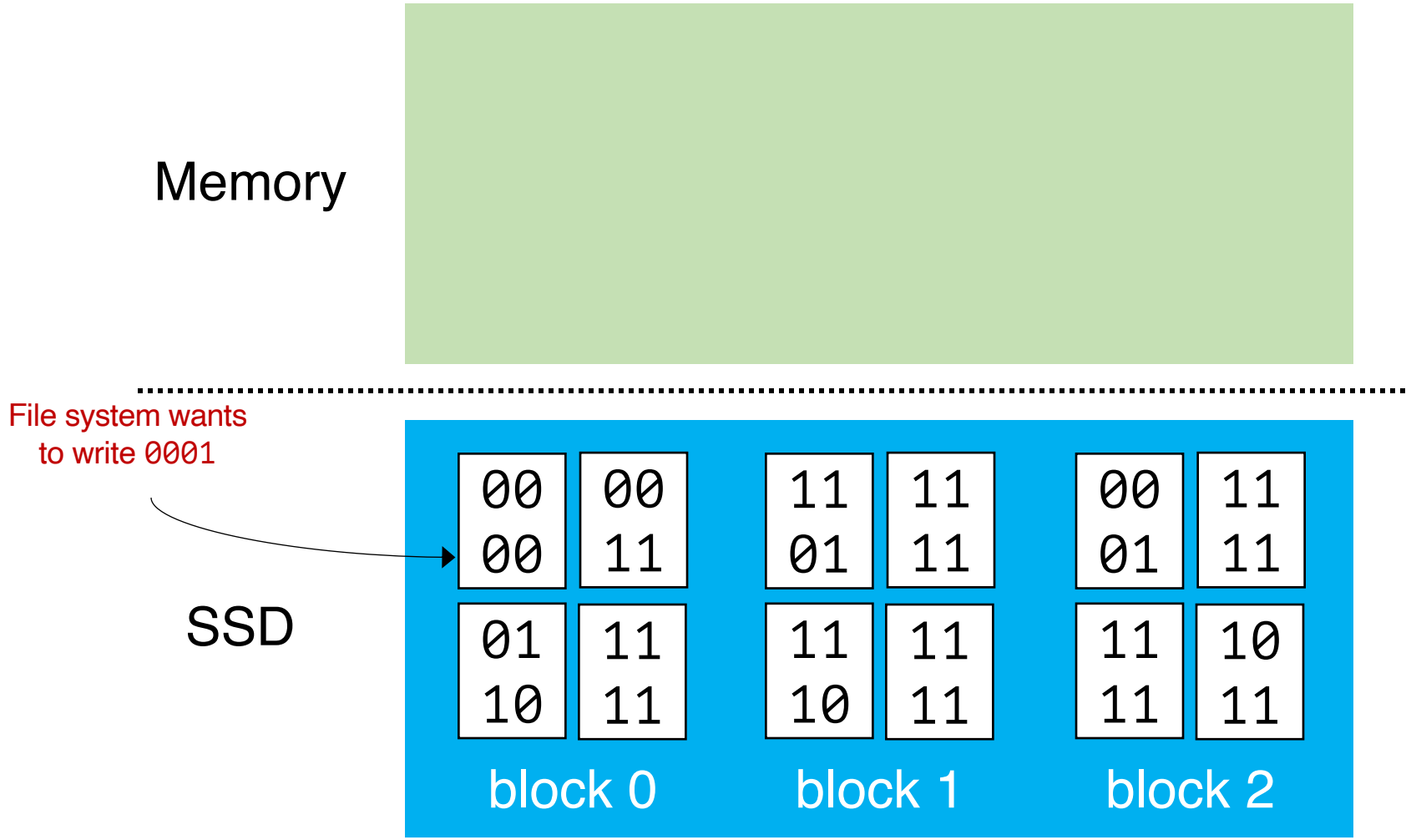
```
read(addr):
    return flash_read(addr)

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

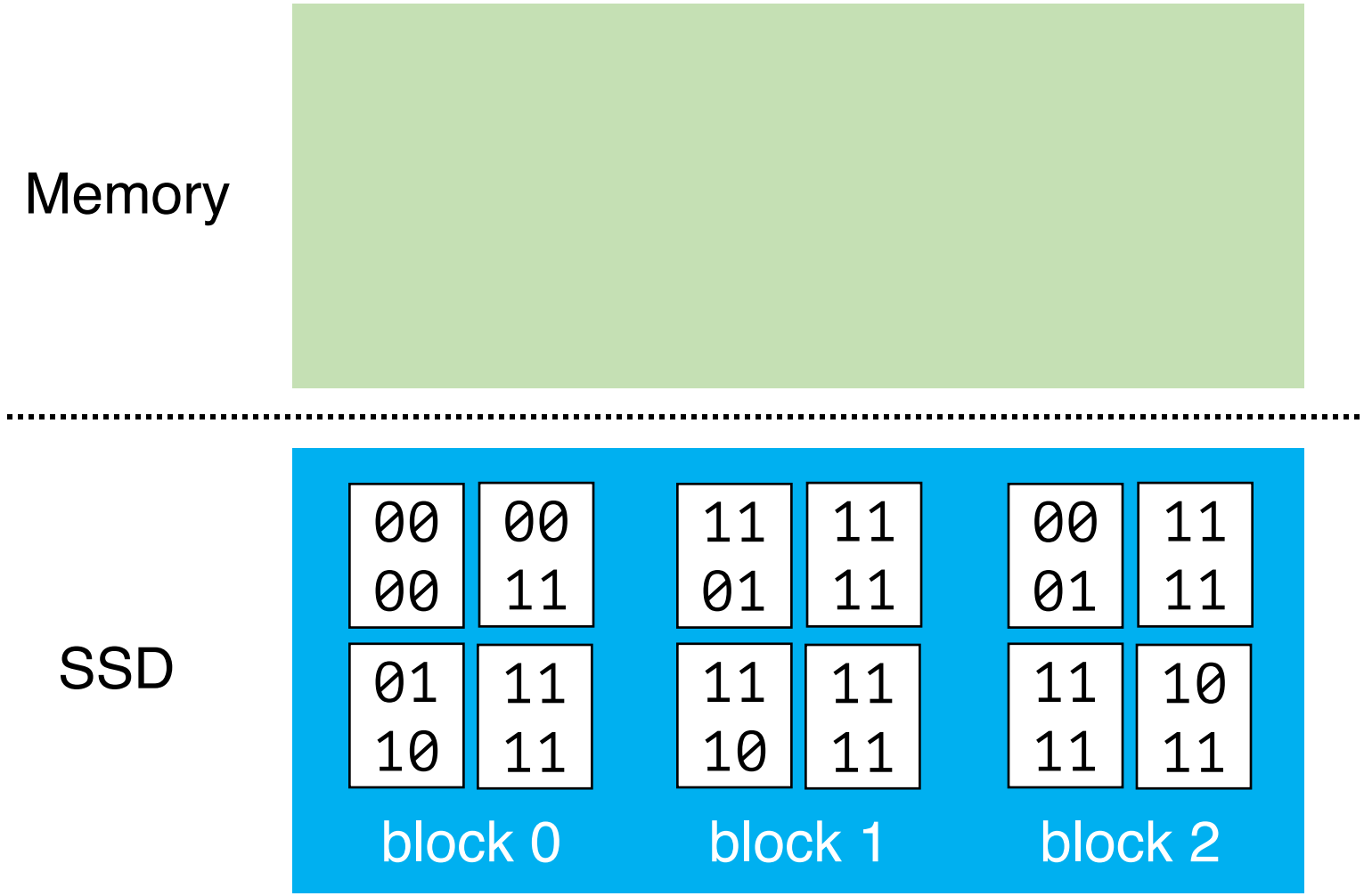
Awkward SSD Write



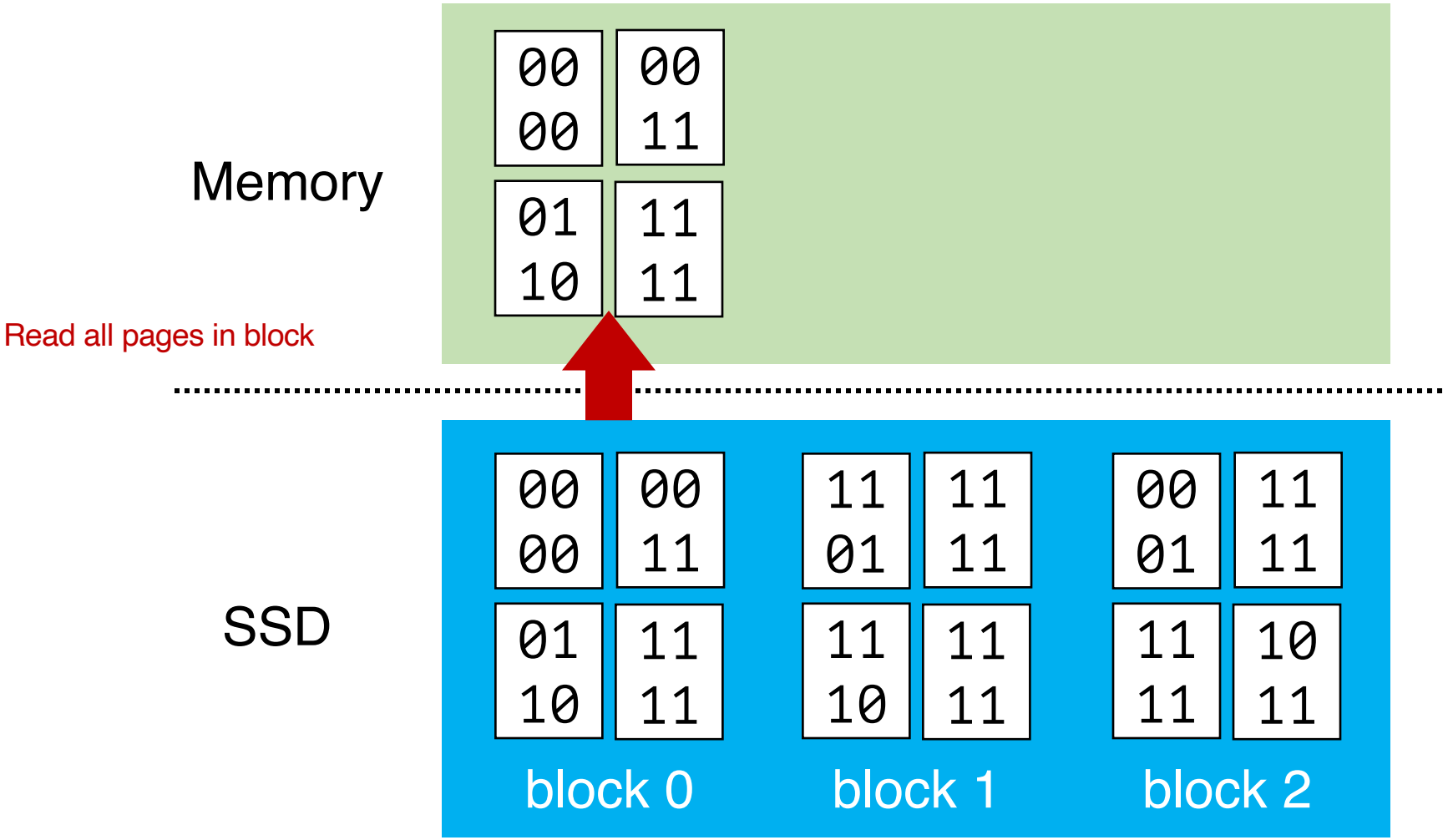
Awkward SSD Write



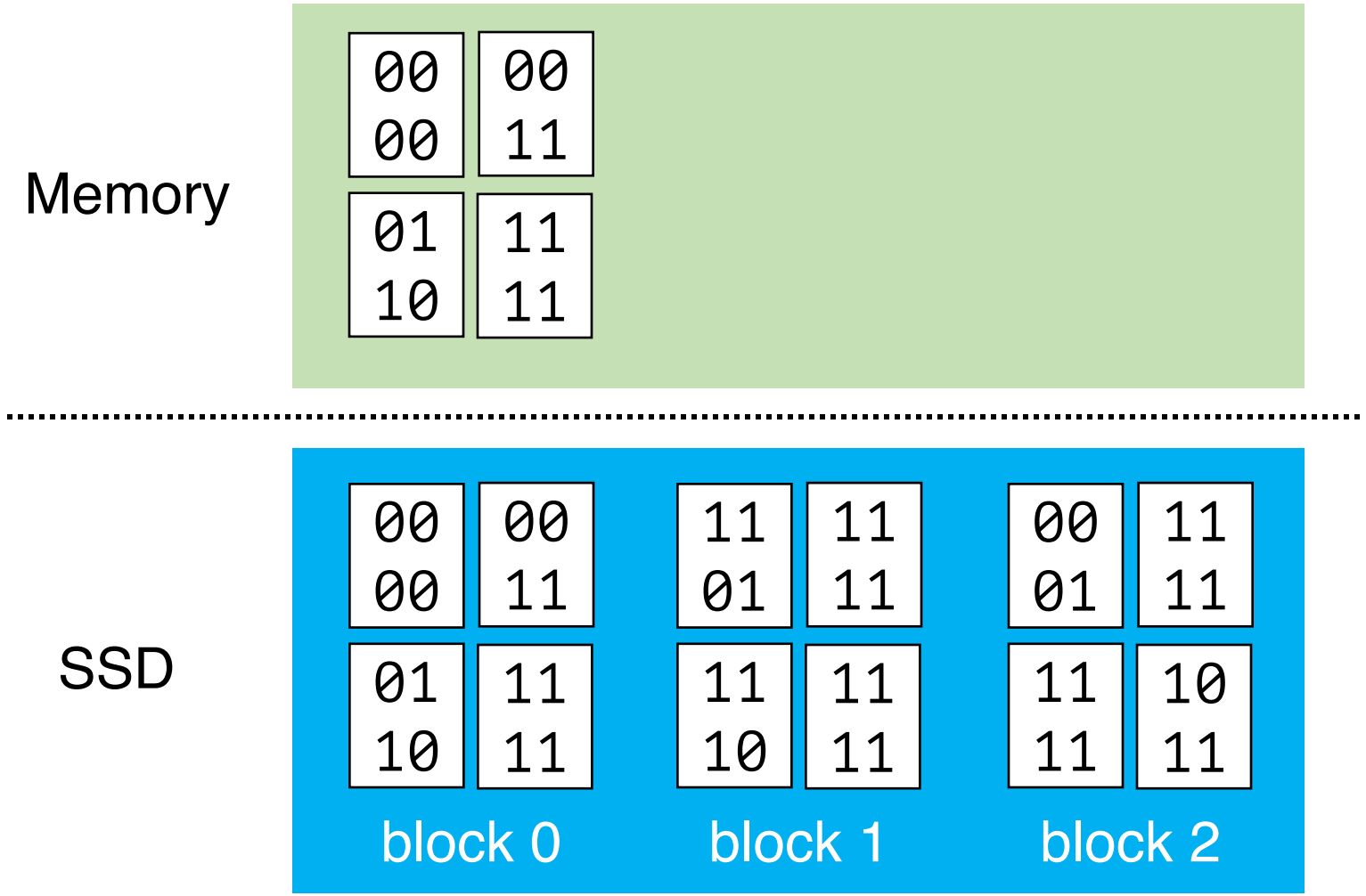
Awkward SSD Write



Awkward SSD Write

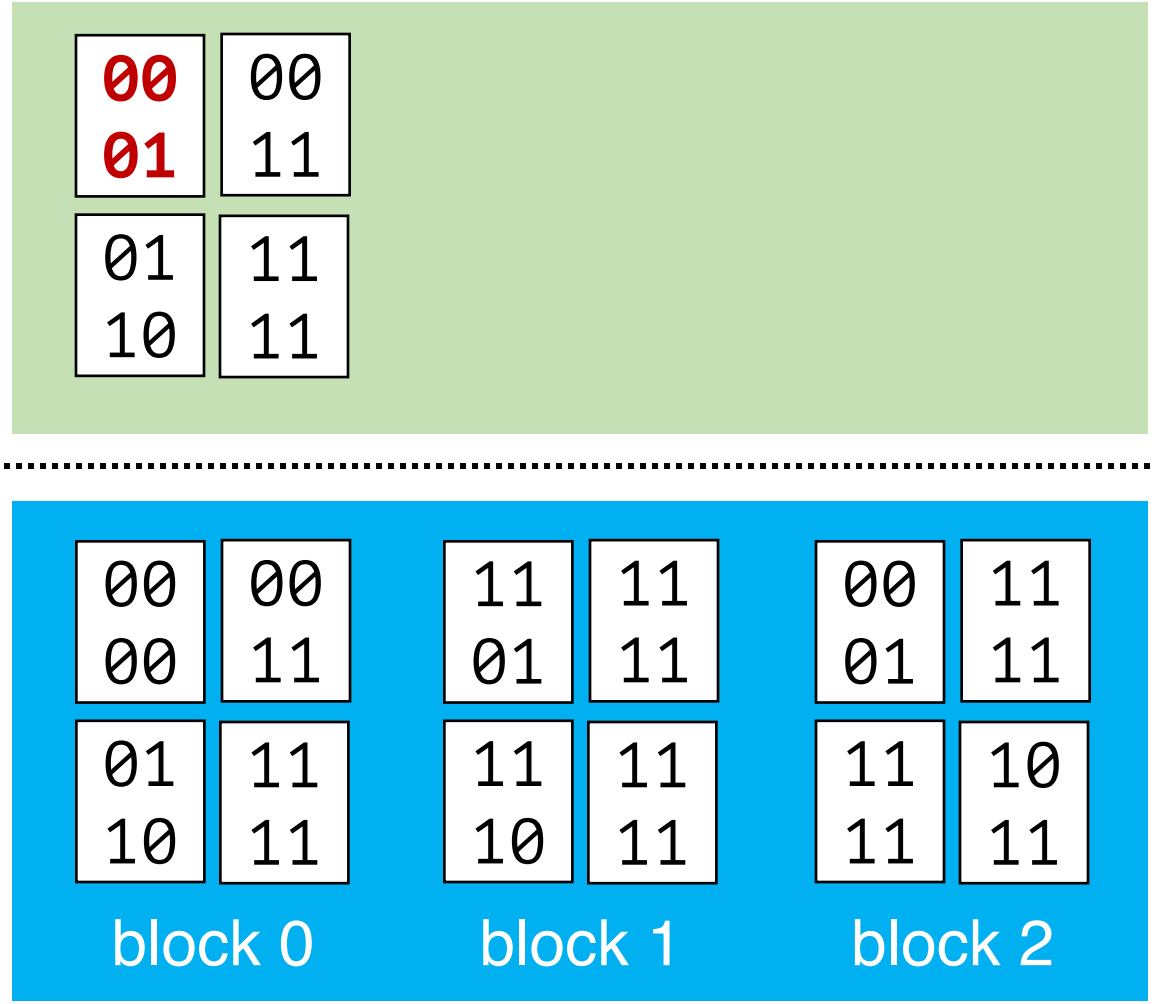


Awkward SSD Write

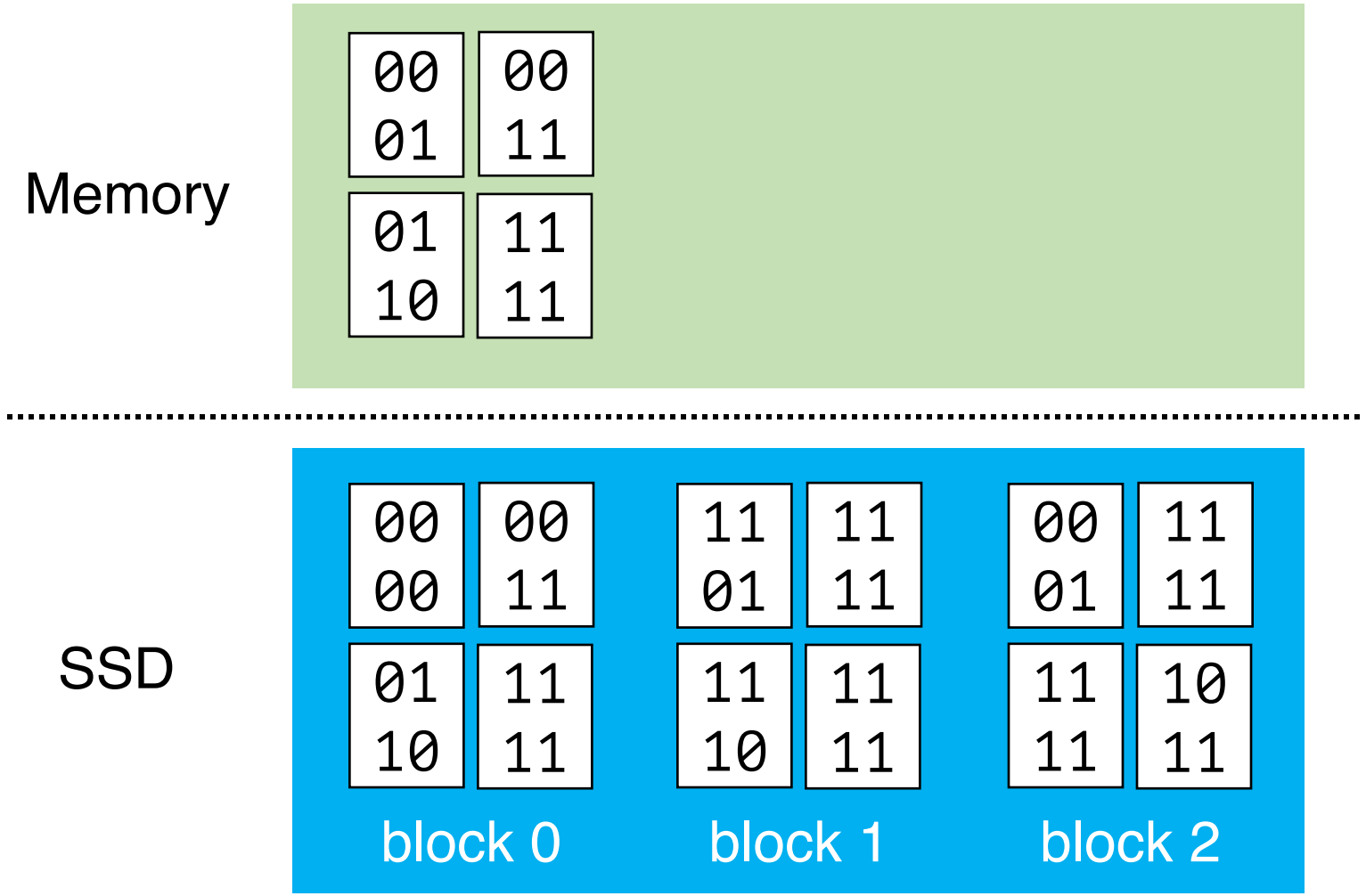


Awkward SSD Write

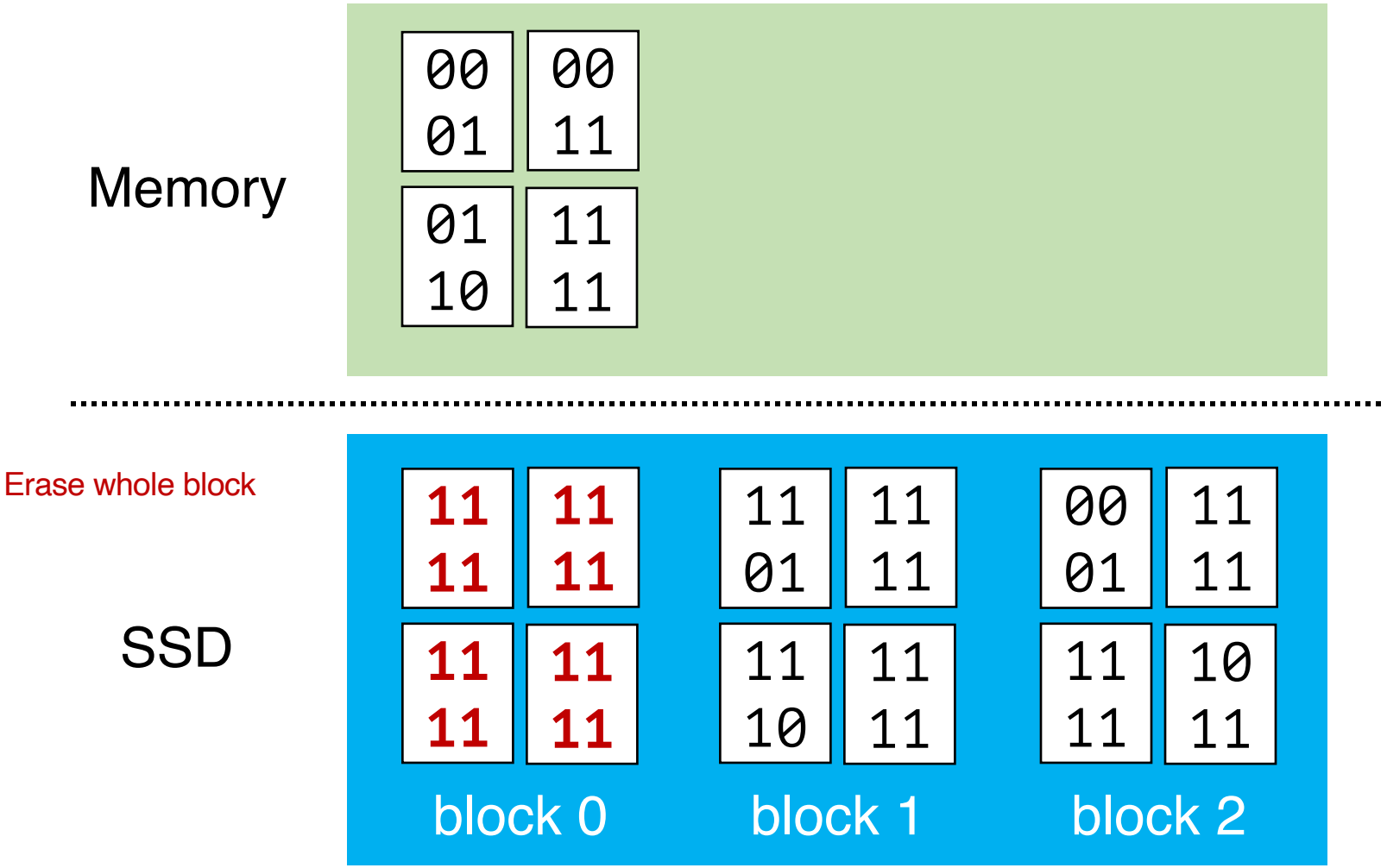
Modify target page in memory



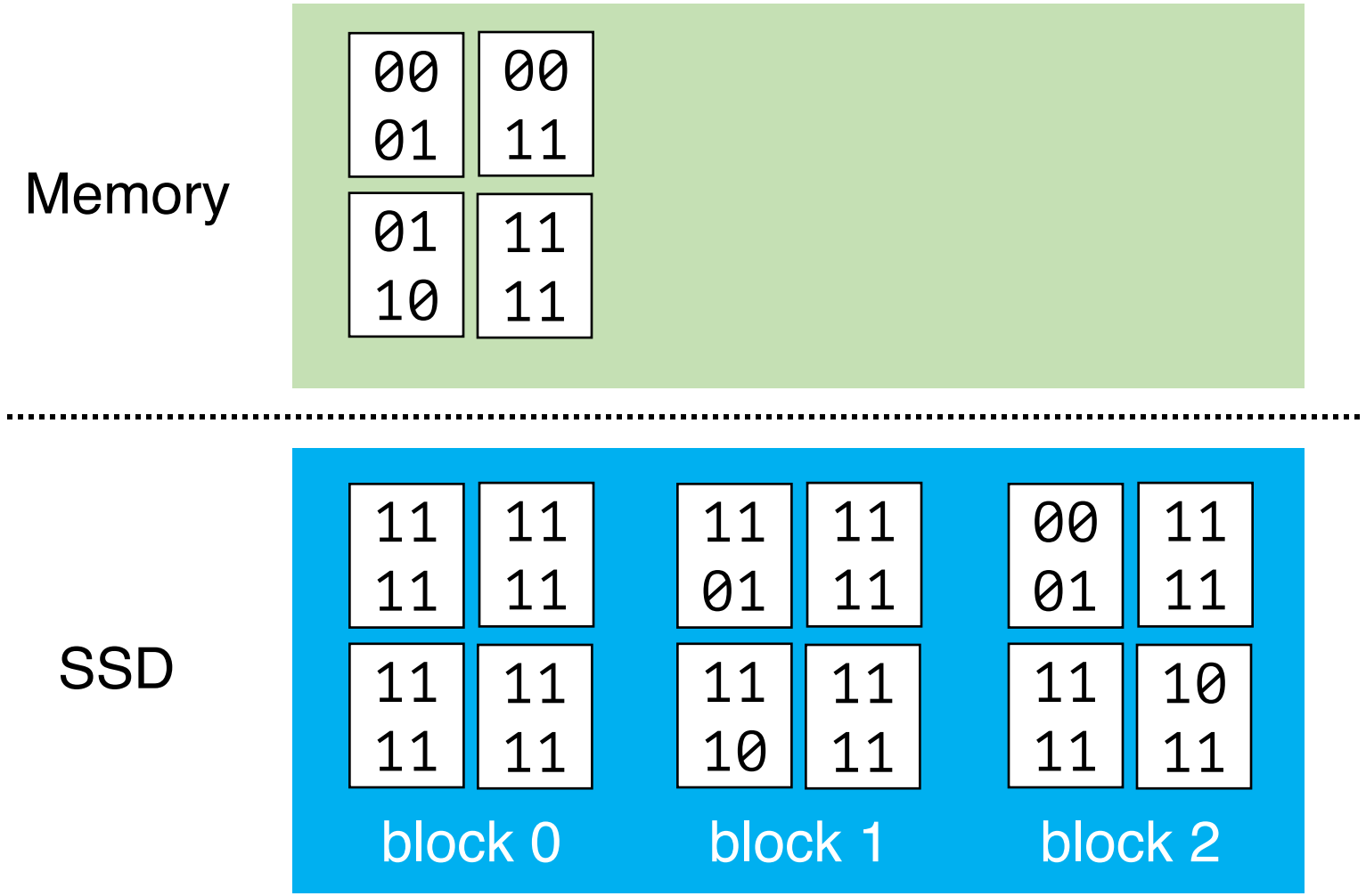
Awkward SSD Write



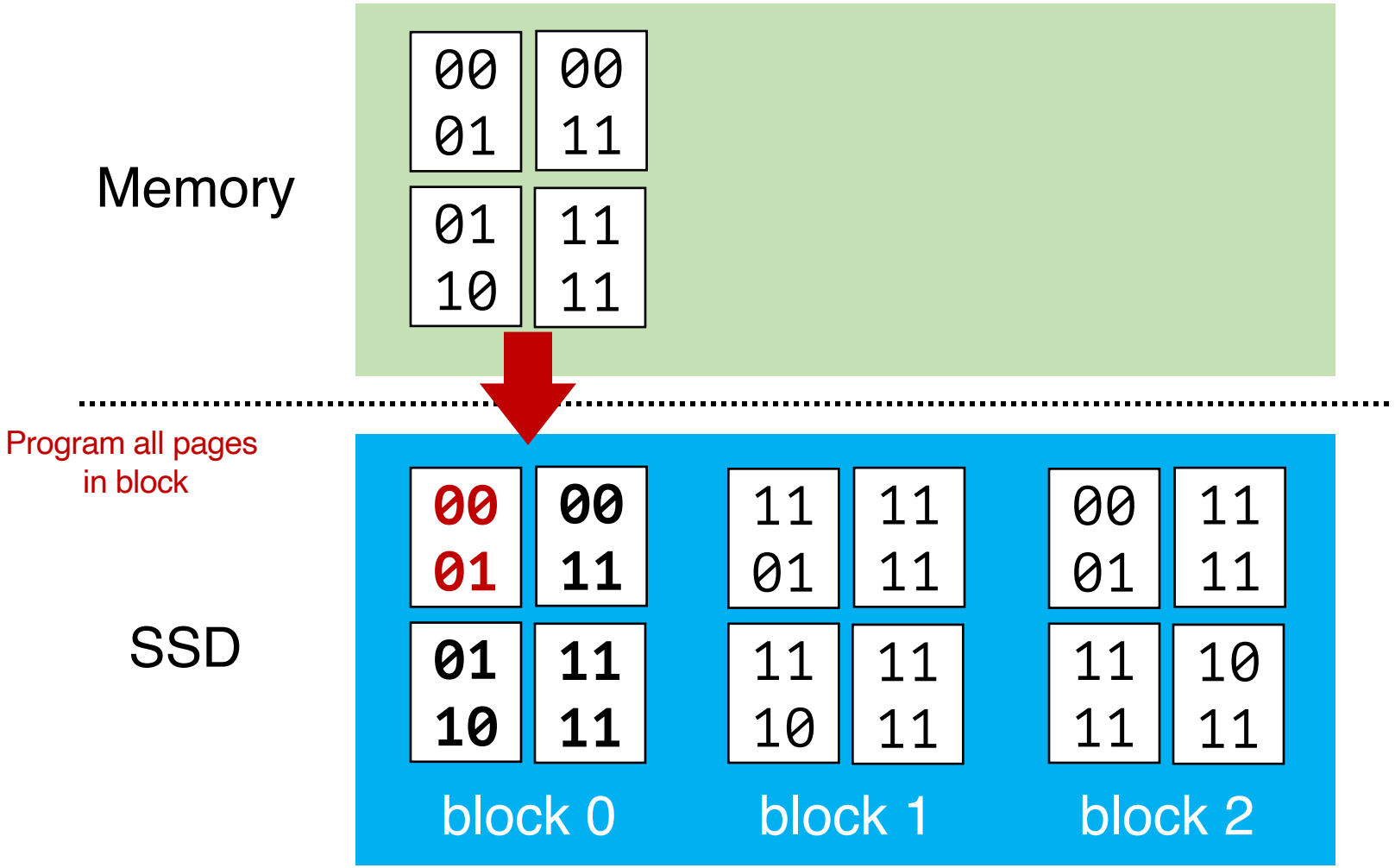
Awkward SSD Write



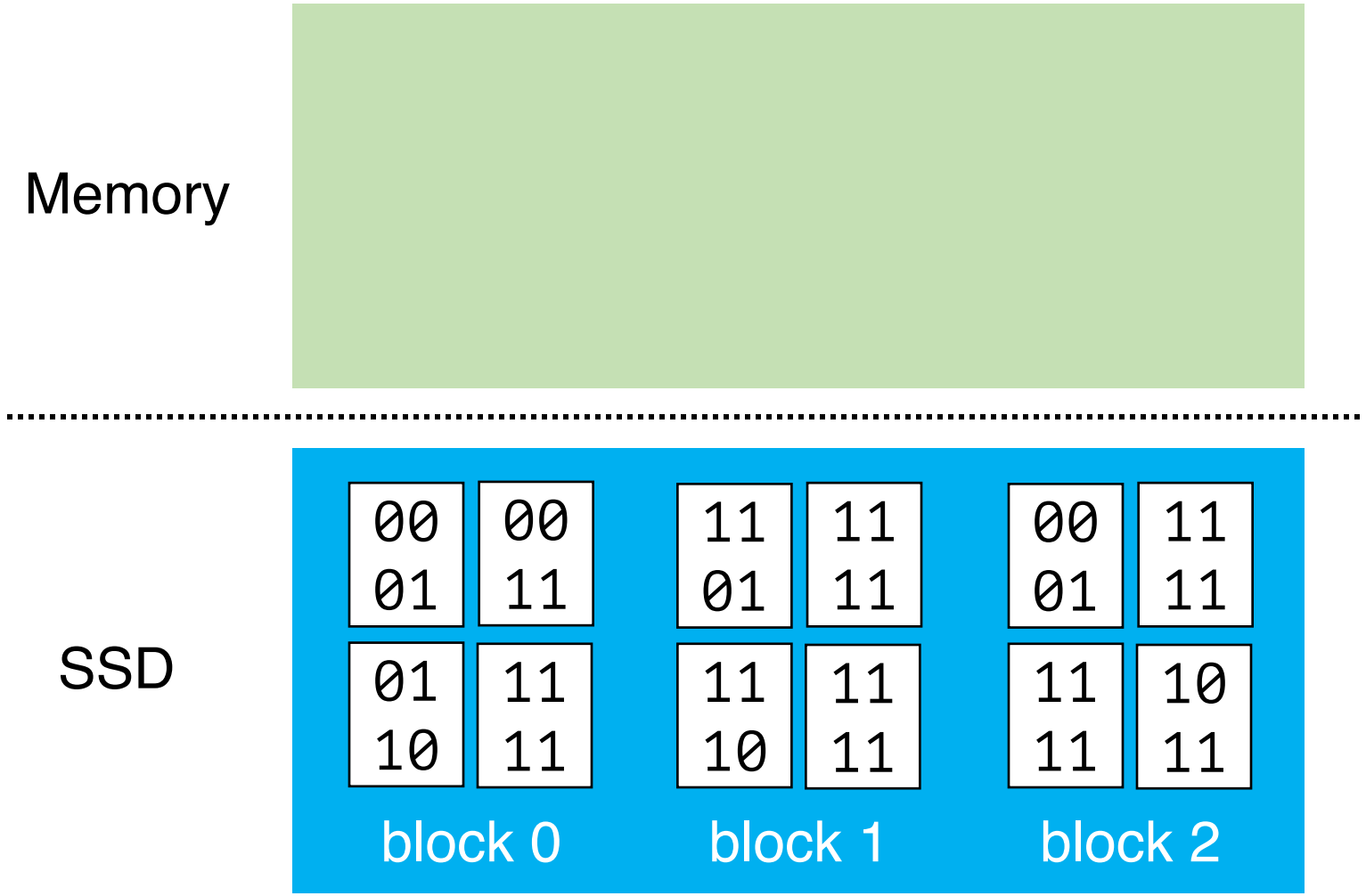
Awkward SSD Write



Awkward SSD Write



Awkward SSD Write



Issue: Write Amplification

- Random writes are expensive for flash!

- Writing one 4KB page may cause:
  - `read`, `erase`, and `program` of the whole 256KB block

Flash Translation Layer (FTL)

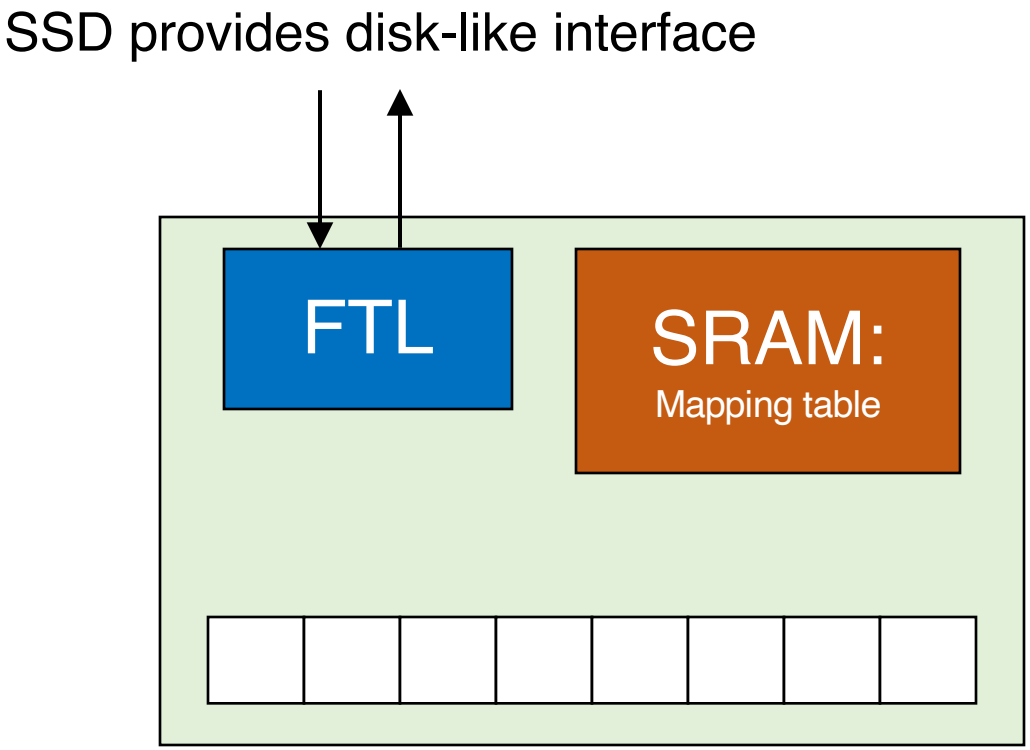
Flash Translation Layer (FTL)

- Add an address translation layer between upper-level file system and lower-level flash
  - Translate logical device addresses to physical addresses
  - Convert in-place write into append-write (log-structured)
  - Essentially, a virtualization & optimization layer

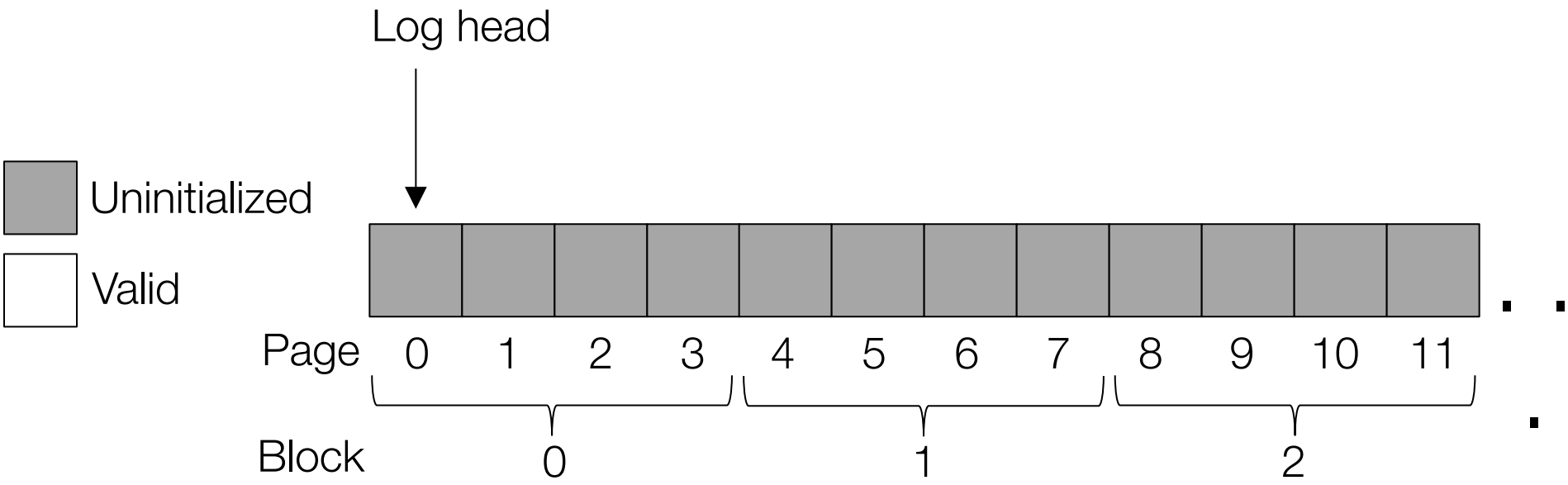
Flash Translation Layer (FTL)

- Usually implemented in SSD device's firmware (hardware)
  - But is also implemented in software for some SSDs
- Where to store mapping?
  - SRAM
- Physical pages can be in three states
  - uninitialized, valid, invalid

SSD Architecture with FTL



Logical-to-physical map

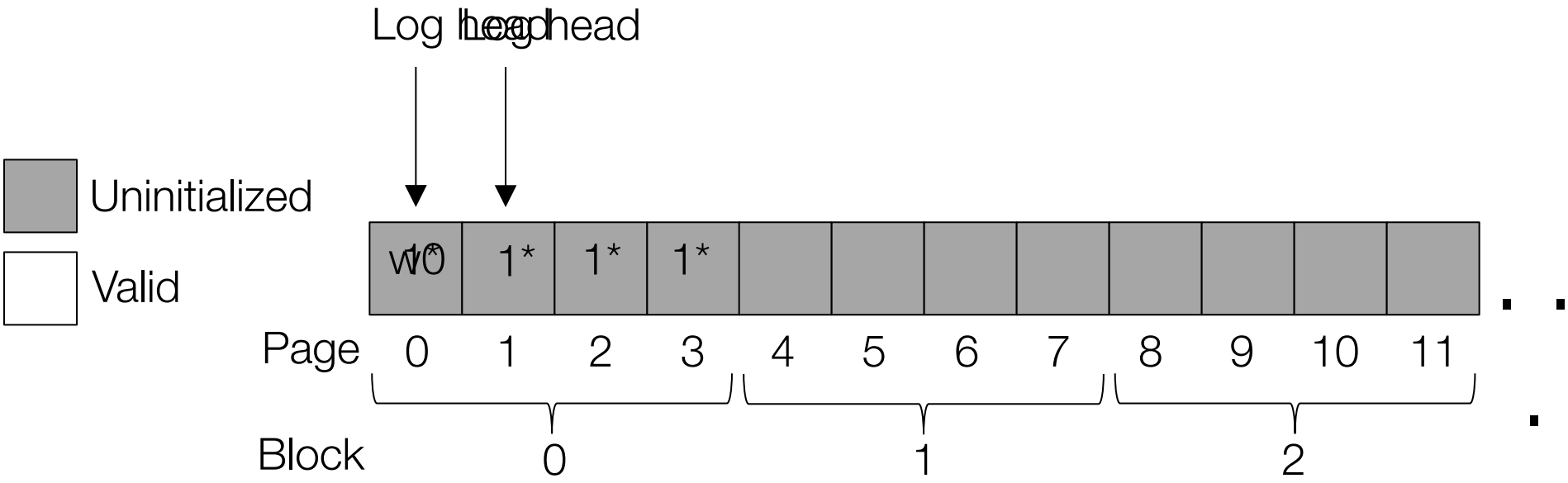


write(page=92, data=w0)

- erase(block0)
- program(page0, w0)
- logHead++

Logical-to-physical map

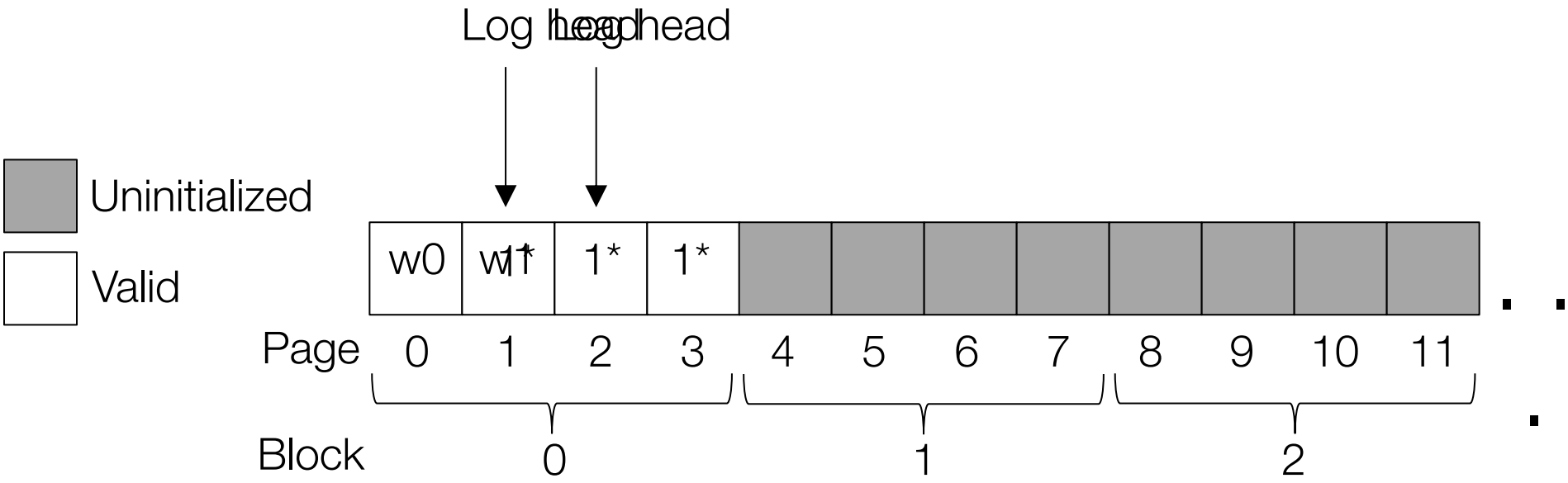
92 --> 0



write(page=92, data=w0)
- erase(block0)
- program(page0, w0)
- logHead++

write(page=17, data=w1)
- program(page1, w1)
- logHead++

Logical-to-physical map
92 --> 0
17 --> 1



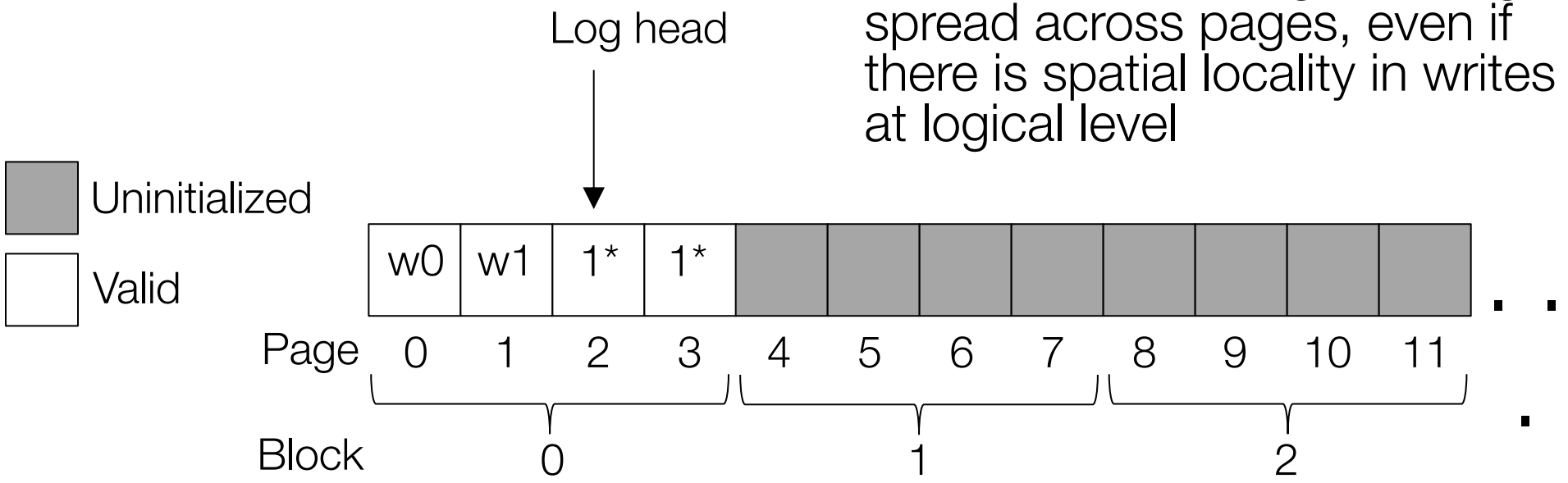
write(page=92, data=w0)
- erase(block0)
- program(page0, w0)
- logHead++

write(page=17, data=w1)
- program(page1, w1)
- logHead++

Logical-to-physical map
92 --> 0
17 --> 1

Advantages w.r.t. direct mapping

- Avoids expensive read-modify-write behavior
- Better wear levelling: writes get spread across pages, even if there is spatial locality in writes at logical level



write(page=92, data=w4)

- erase(block1)
- program(page4, w4)
- logHead++

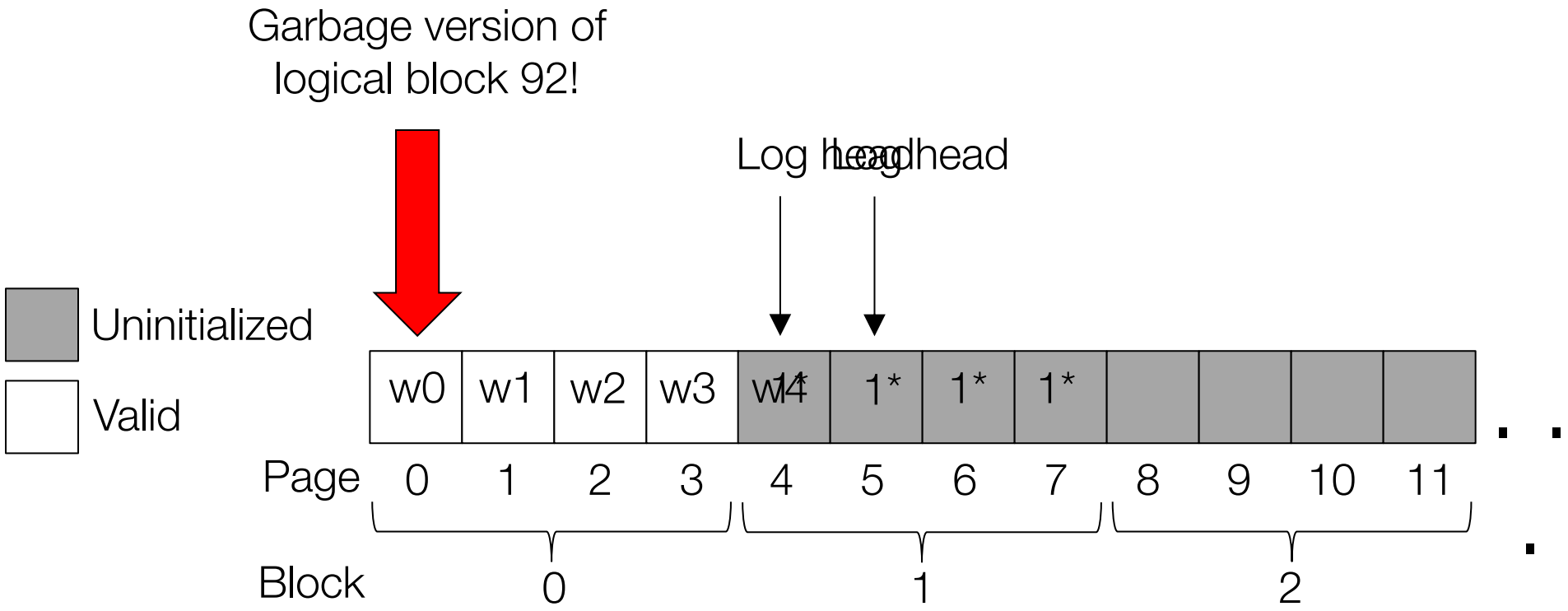
Logical-to-physical map

~~92 --> 0~~ 92 --> 4

17 --> 1

33 --> 2

68 --> 3



At some point, FTL must:

- Read all pages in physical block 0
- Write out the second, third, and fourth pages to the end of the log
- Update logical-to-physical map

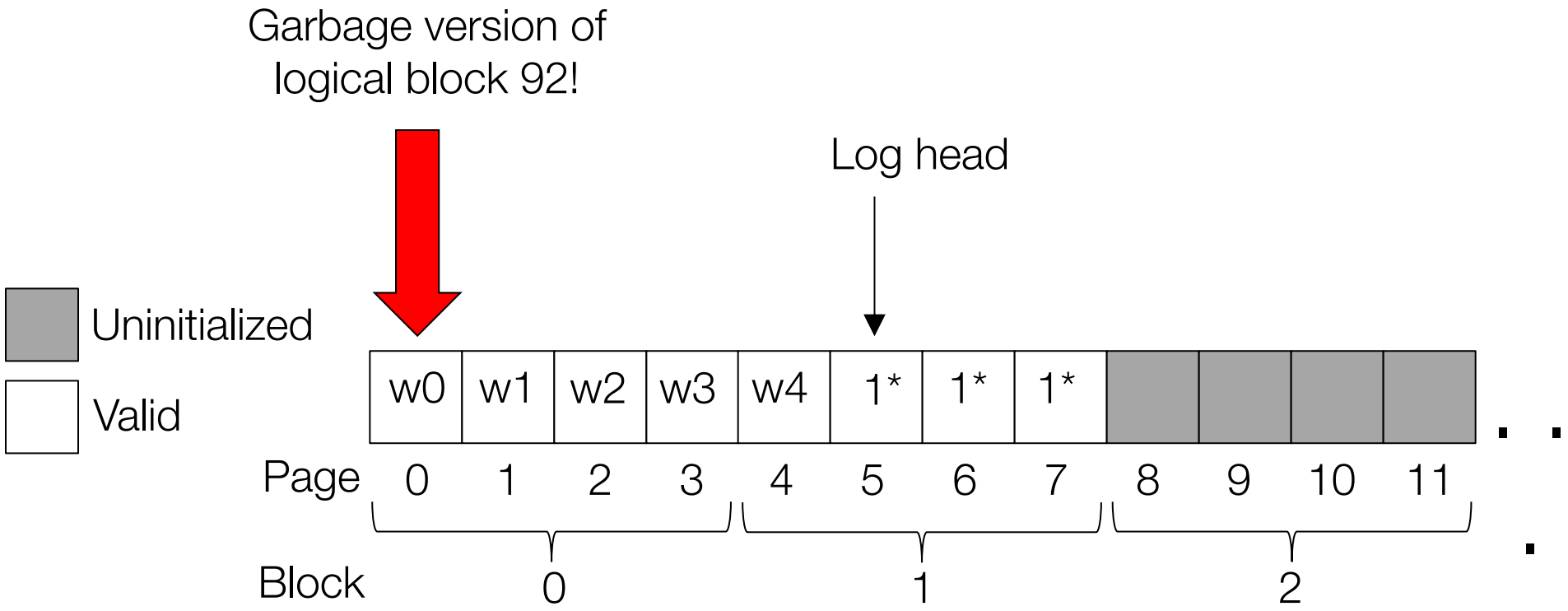
Logical-to-physical map

~~92 --> 0~~ 92 --> 4

17 --> 1

33 --> 2

68 --> 3



Trash Day is the Worst Day

- Garbage collection requires extra read+write traffic
- Overprovisioning makes GC less painful
  - SSD exposes a logical page space that is smaller than the physical page space
  - By keeping extra, “hidden” pages around, the SSD tries to defer GC to a background task (thus removing GC from critical path of a write)
- SSD will occasionally shuffle live (i.e., non-garbage) blocks that never get overwritten
  - Enforces wear leveling