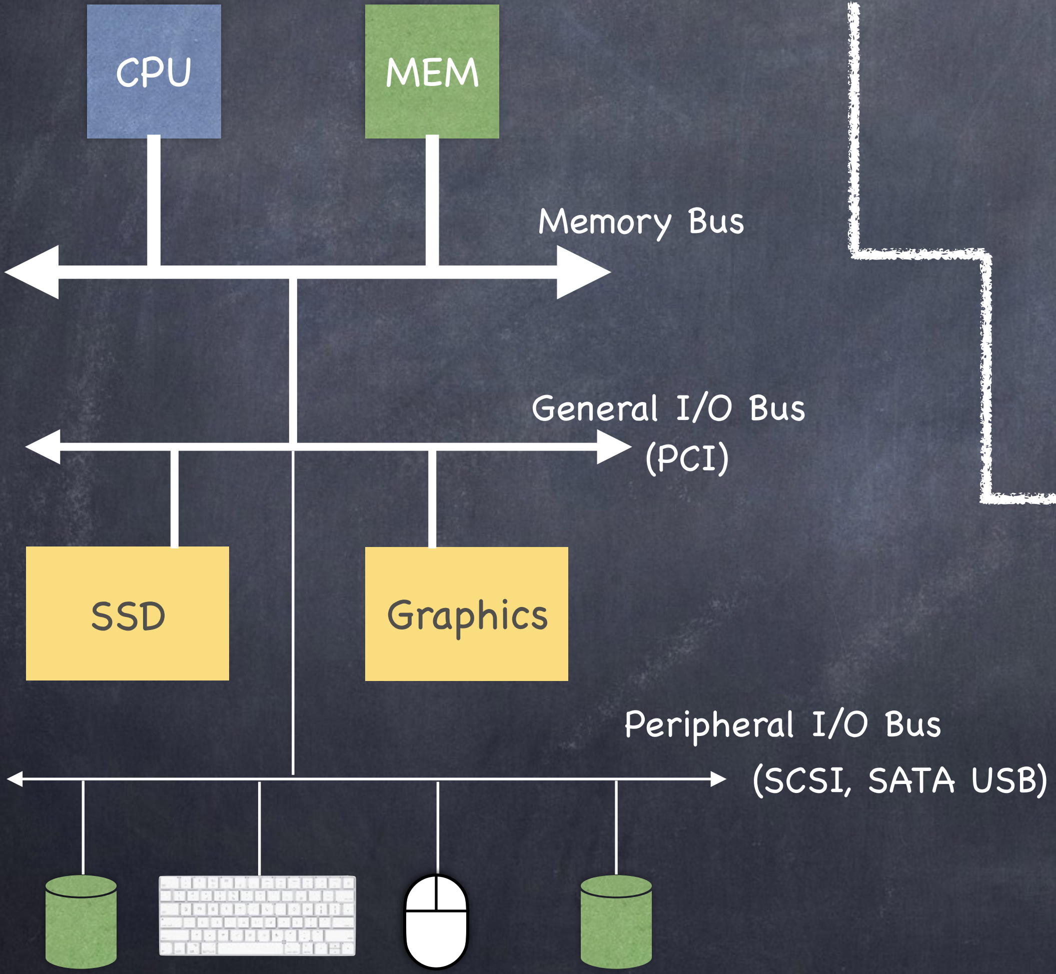
I/O Devices

Input/Output

- Mechanism to enable CPU to communicate with “outside” world
    - Memory
    - Persistent Storage
    - Remote resources/Network

Simplified I/O Architecture



CPU

MEM

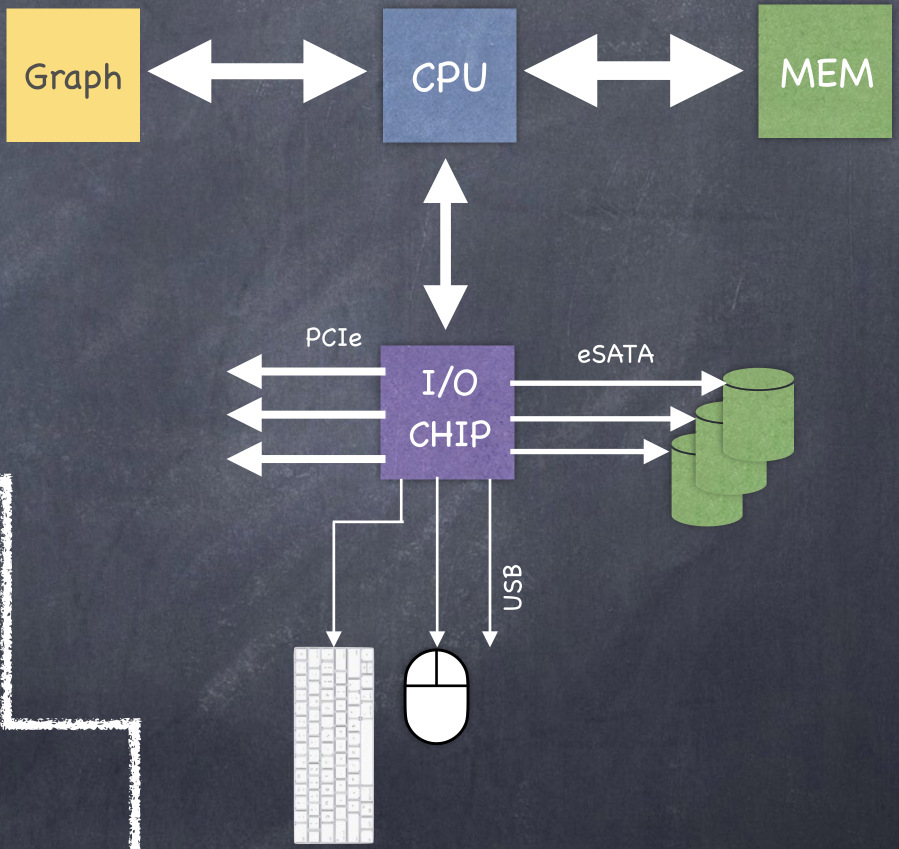
Memory Bus

General I/O Bus
(PCI)

SSD

Graphics

Peripheral I/O Bus
(SCSI, SATA USB)



Graph

CPU

MEM

PCIe

I/O
CHIP

eSATA

USB

Network within a server

- Memory bus, PCI bus, Peripheral I/O bus…
- What is a bus?
    - Common set of wires for communication among hardware devices
    - Allows connecting multiple devices over a singleton set of wires
- Split into three parts
    - Address bus: used to specify a physical address (e.g., memory location)
    - Control bus: used to carry commands from CPU, & signals from devices
    - Data bus: used for actual transfer of data
- Protocol:
    - Each bus has its own "protocol" for carrying out data transfer

Network within a server

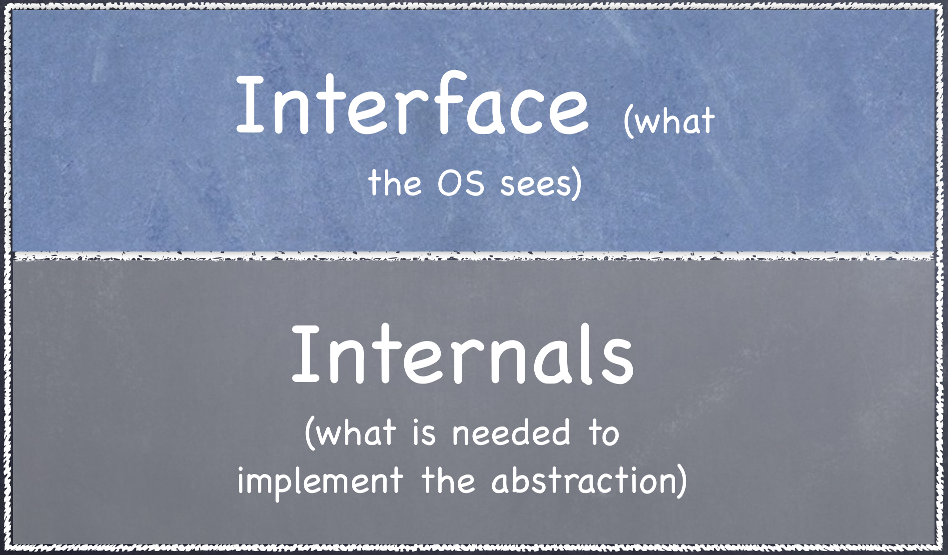
- Memory bus, PCI bus, Peripheral I/O bus…
- Memory bus
  - Connects the main memory (DRAM) to the memory controller
  - Enables communication between CPU and DRAM
- General I/O (e.g., PCI express) bus
  - Connects variety of other devices to the root complex (host)
  - E.g., GPUs, SSDs, NICs (network interface cards), etc.
- Peripheral bus:
  - Connects variety of peripheral devices (and mass storage devices)
  - E.g., printers, mouse, keyboard, etc.

Interacting with a Device

Abstraction

(what the user sees)

Interacting with a Device

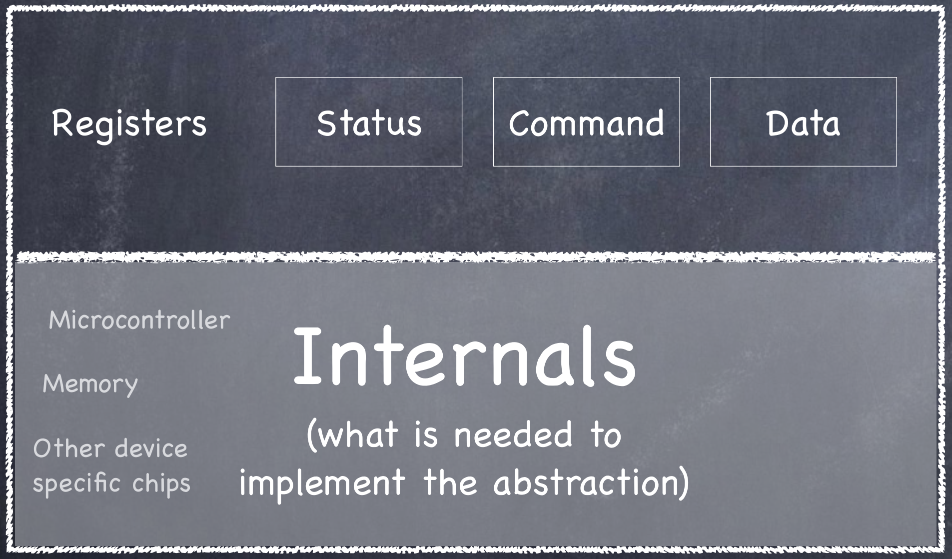


Interface (what the OS sees)

Internals

(what is needed to implement the abstraction)

Interacting with a Device



Registers

Status

Command

Data

Microcontroller

Memory

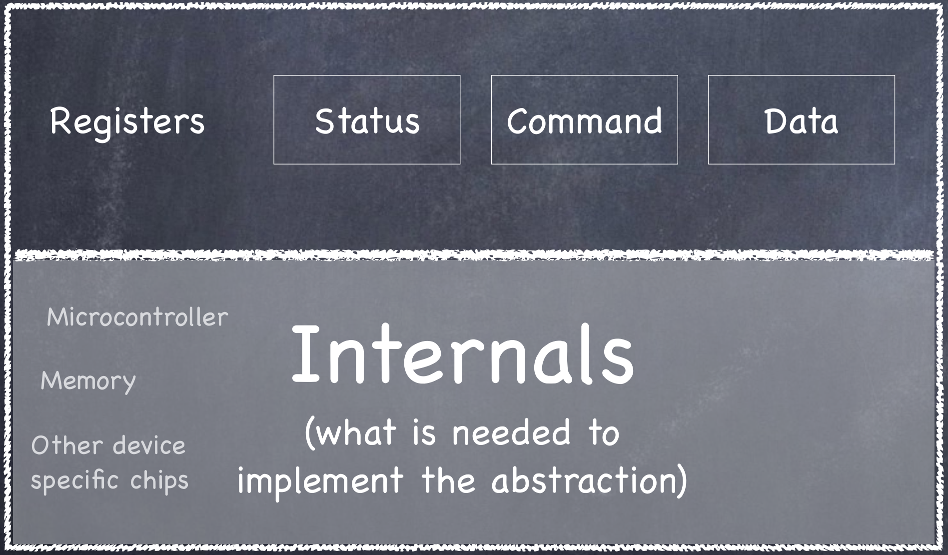
Other device specific chips

Internals

(what is needed to implement the abstraction)

Interacting with a Device

- OS controls device by reading/ writing registers



```
while (STATUS == BUSY)
          ; // wait until device is not busy
  write data to DATA register
write command to COMMAND register
      // starts device and executes command
while (STATUS == BUSY)
          ; // wait until device is done with request
```

Tuning It Up

- CPU is polling
    - use interrupts
    - run another process while device is busy
    - what if device returns very quickly?
- CPU is copying all the data to and from DATA
    - use Direct Memory Access (DMA)

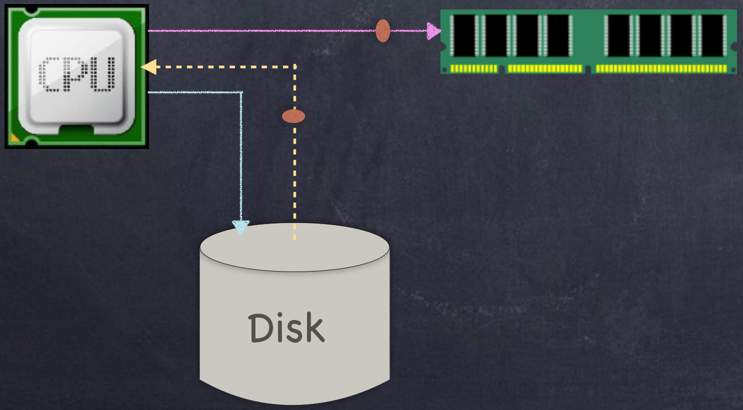
```
while (STATUS == BUSY)
          ; // wait until device is not busy
  write data to DATA register
write command to COMMAND register
      // starts device and executes command
while (STATUS == BUSY)
          ; // wait until device is done with request
```

From interrupt-driven I/O to DMA

- Interrupt driven I/O
  - Device ⟷ CPU ⟷ RAM

for $(i = 1 \ldots n)$

- CPU issues read request
- device interrupts CPU with data
- CPU writes data to memory



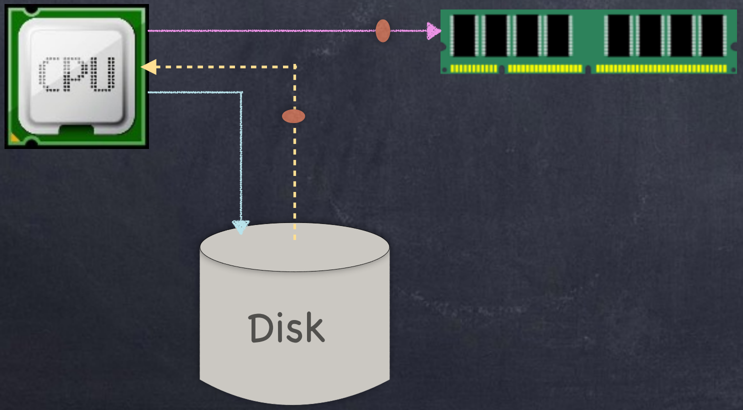
From interrupt-driven I/O to DMA

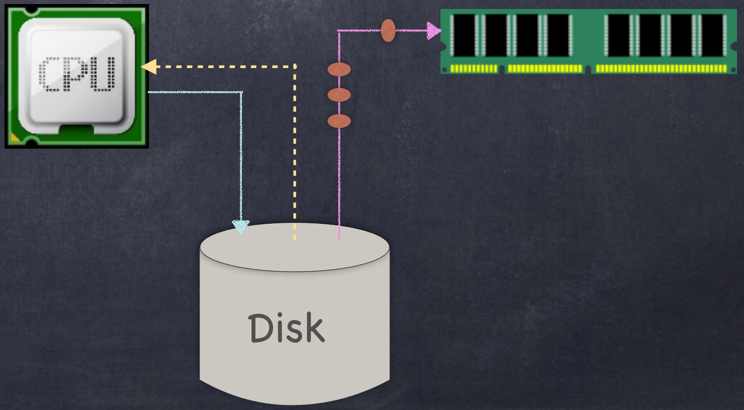
- Interrupt driven I/O
  - ☐ Device ↔ CPU ↔ RAM

for $(i = 1 \ldots n)$

- CPU issues read request
- device interrupts CPU with data
- CPU writes data to memory

- + Direct Memory Access
  - ☐ Device ↔ RAM
    - CPU sets up DMA request
    - Device puts data on bus & RAM accepts it
    - Device interrupts CPU when done

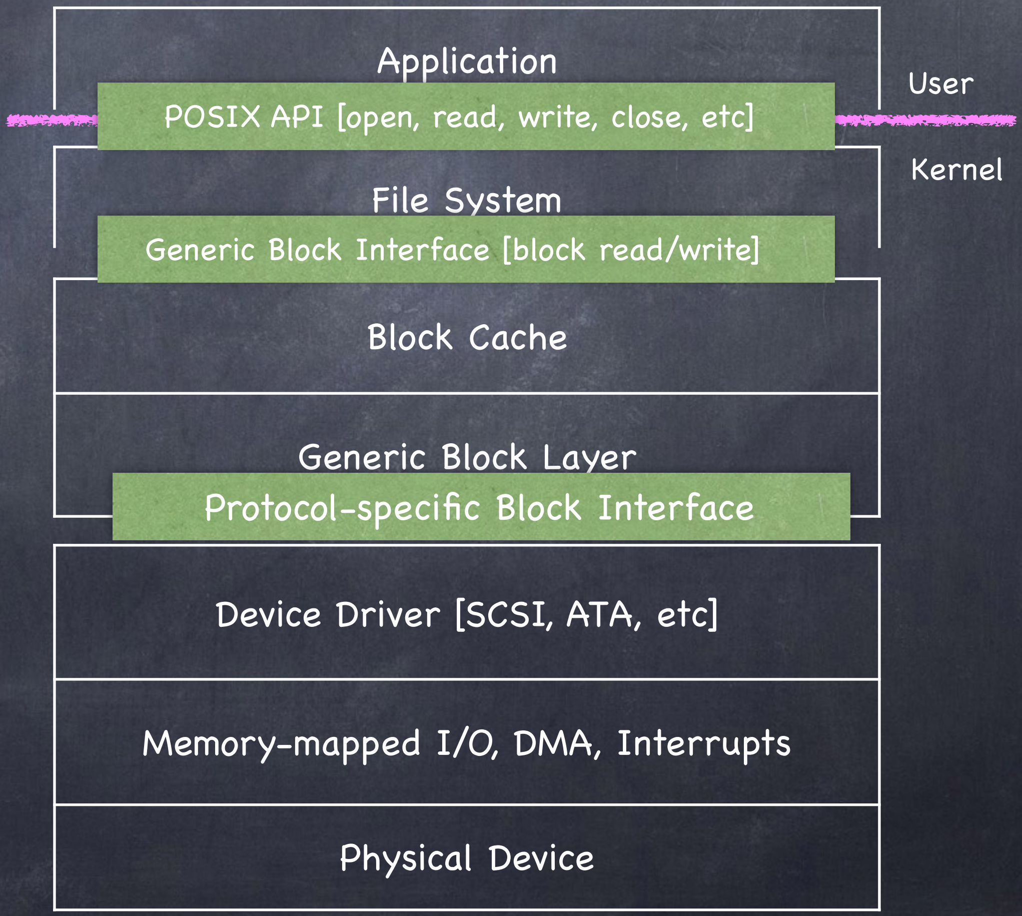




How can the OS handle a multitude of devices?

- Abstraction!
  - Encapsulate device specific interactions in a device driver
  - Implement device neutral interfaces above device drivers
- Humans are about 70% water...
  - ...OSs are about 70% device drivers!

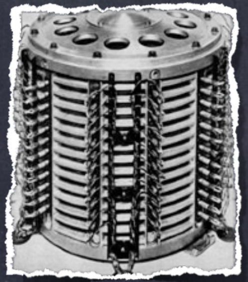
File System Stack (simplified)



Persistent Storage

Storage Devices

- We focus on two types of persistent storage
    - magnetic disks
        - servers, workstations, laptops
    - flash memory
        - smart phones, tablets, cameras, laptops
- Other exist(ed)
    - tapes
    - drums
    - clay tablets



Magnetic disks vs Flash

- Magnetic disks
  - Large capacity at low cost
  - Block-level random access
  - Poor random access performance
  - Better sequential access performance

- Flash memory
  - Capacity at intermediate cost
  - Block-level random access
  - Good performance for reads
  - Worse performance for writes
  - Wear issues

The SSD
Storage Hierarchy

Cell

1 to 4
bits

Page

2 to 8 KB

not to be
confused with
a VM page

Block

64 to 256
pages

not to be confused
with a disk block

Plane/Bank

Many blocks
(Several Ks)

Flash Chip

Several banks that
can be accessed
in parallel

Basic Flash Operations

- Read (a page)
  - 10s of μs, independent of the previously read page
    - great for random access!
- Erase (a block)
  - sets the entire block (with all its pages) to 1 (!)
  - very coarse way to write 1s…
  - 1.5 to 2 ms
- Program (a page)
  - can change some bits in a page of an erased block to 0
  - 100s of μs
  - changing a 0 bit back to 1 requires erasing the entire block!

Banks

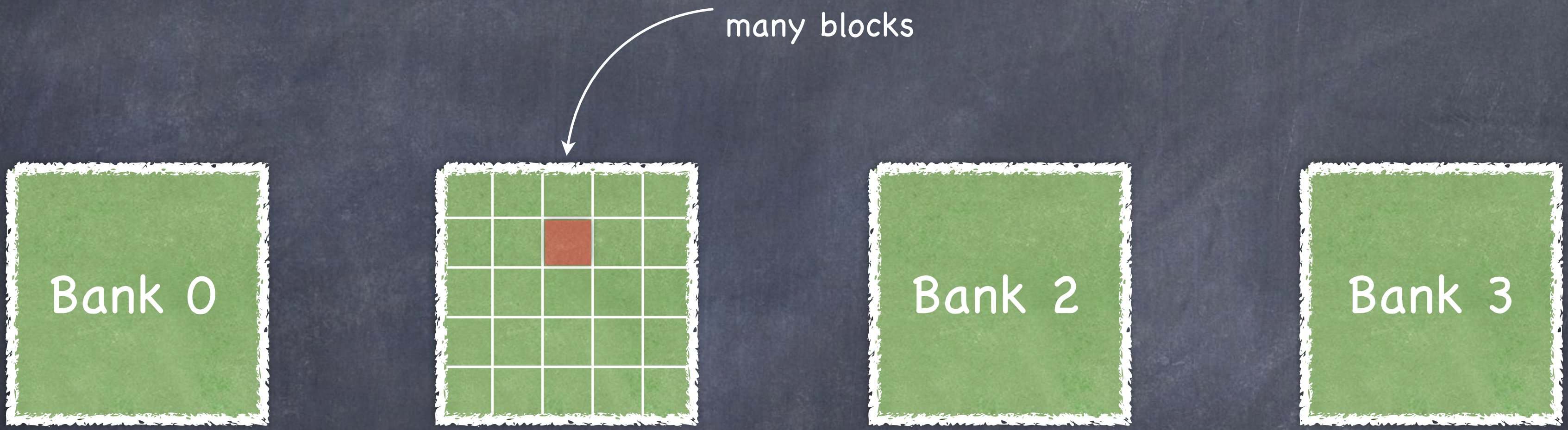
Bank 0

Bank 1

Bank 2

Bank 3

Banks



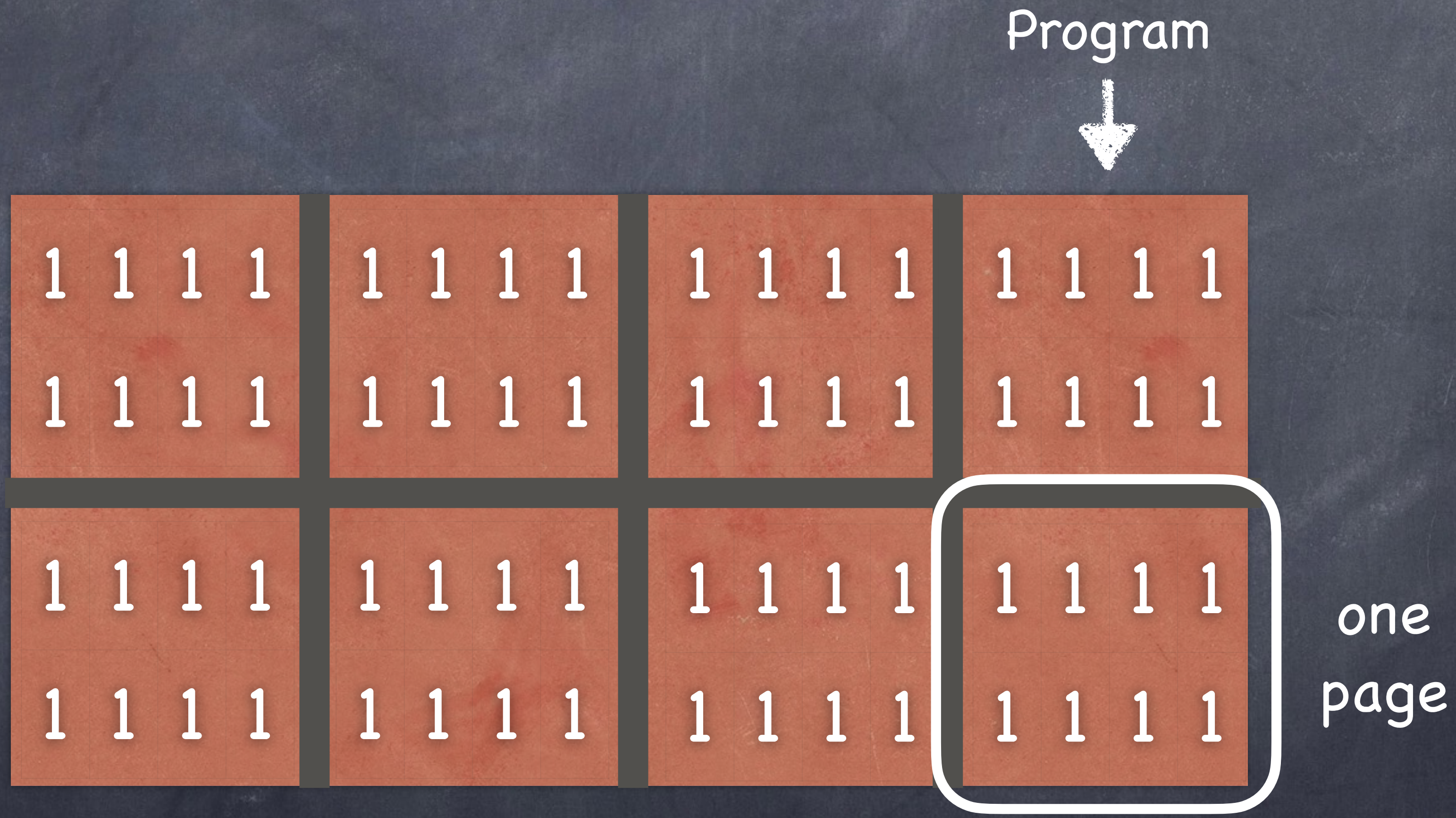
Each bank contains many blocks

Bank 0

Bank 2

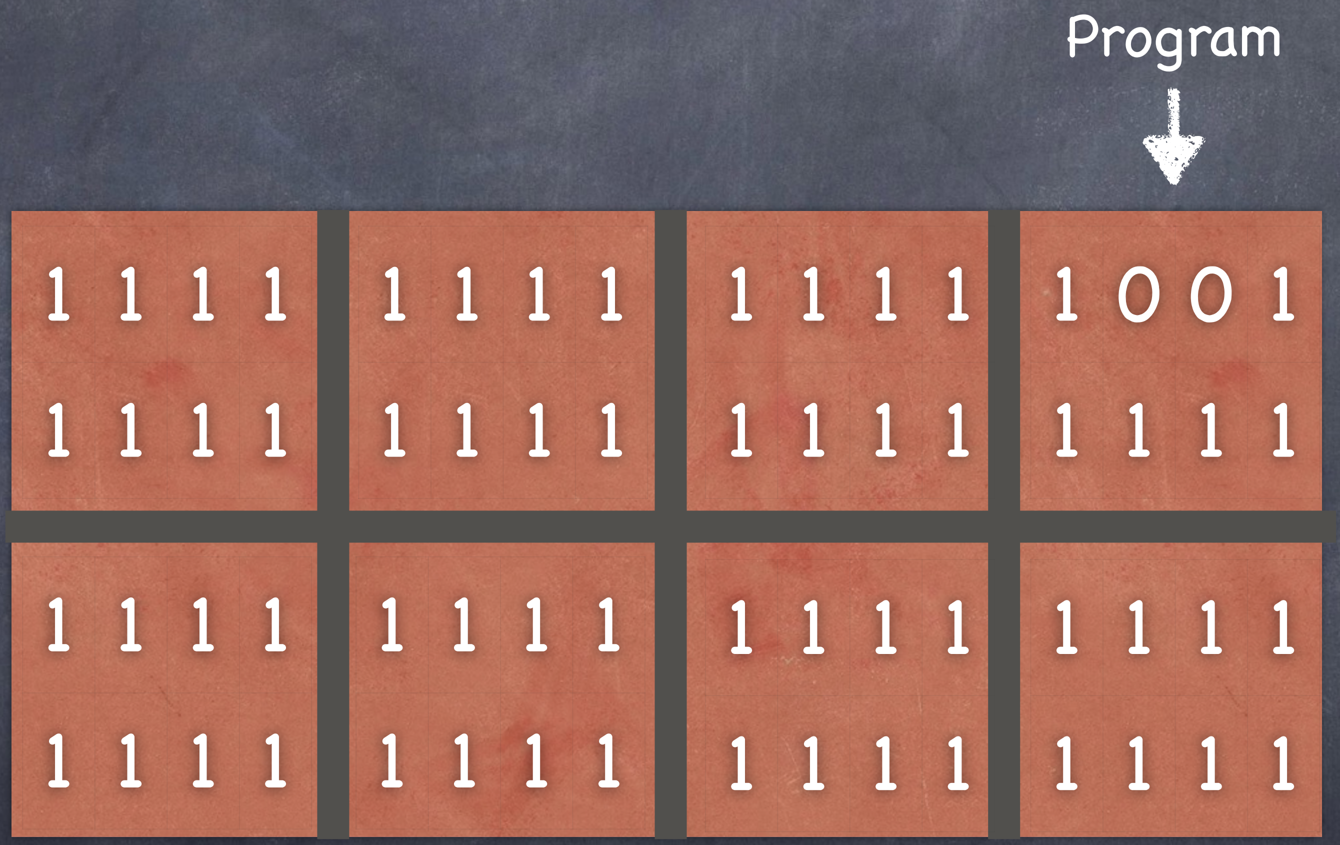
Bank 3

Block



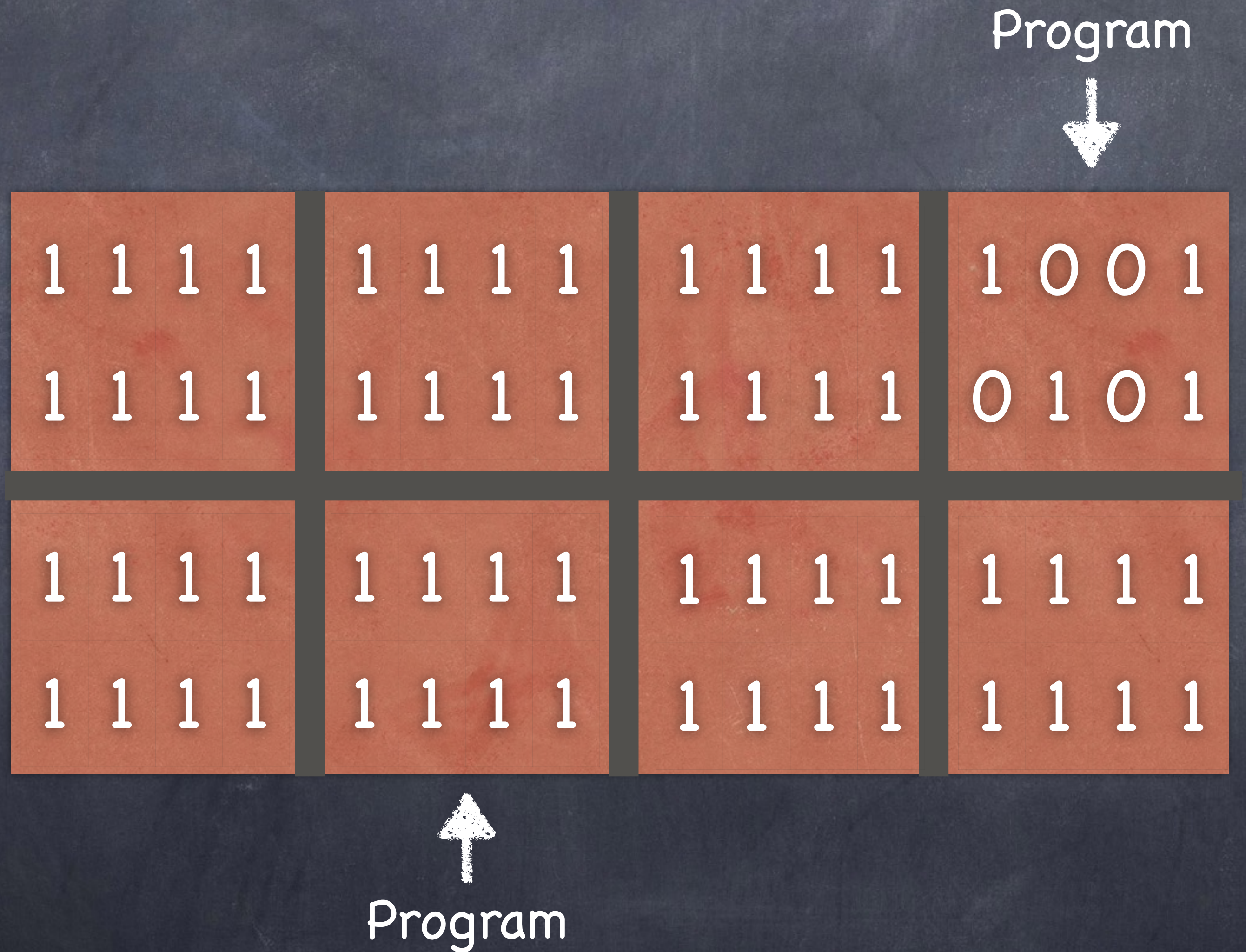
After an Erase, all cells are discharged (i.e., store 1s)

Block



1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1

Block



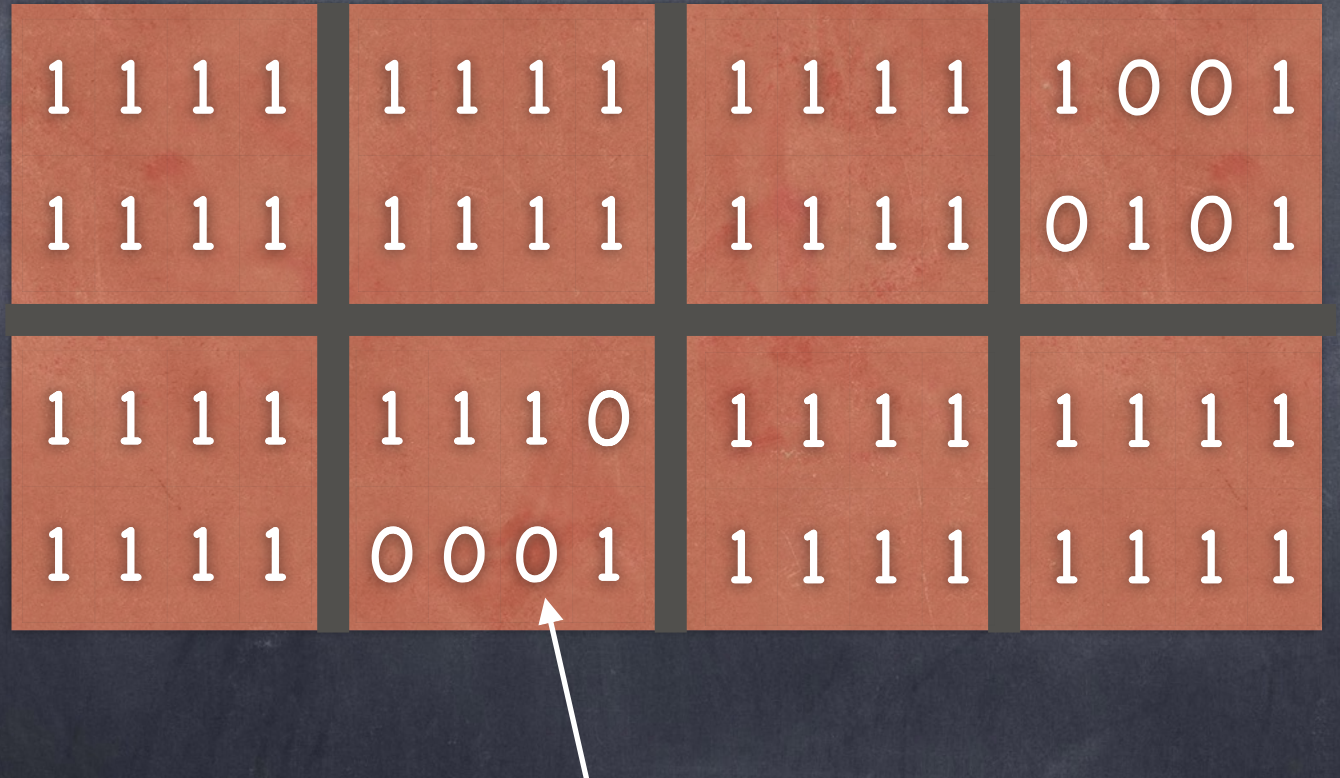
Program

1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 1 0 1 0 1
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1

Program

Block

Erase (!)



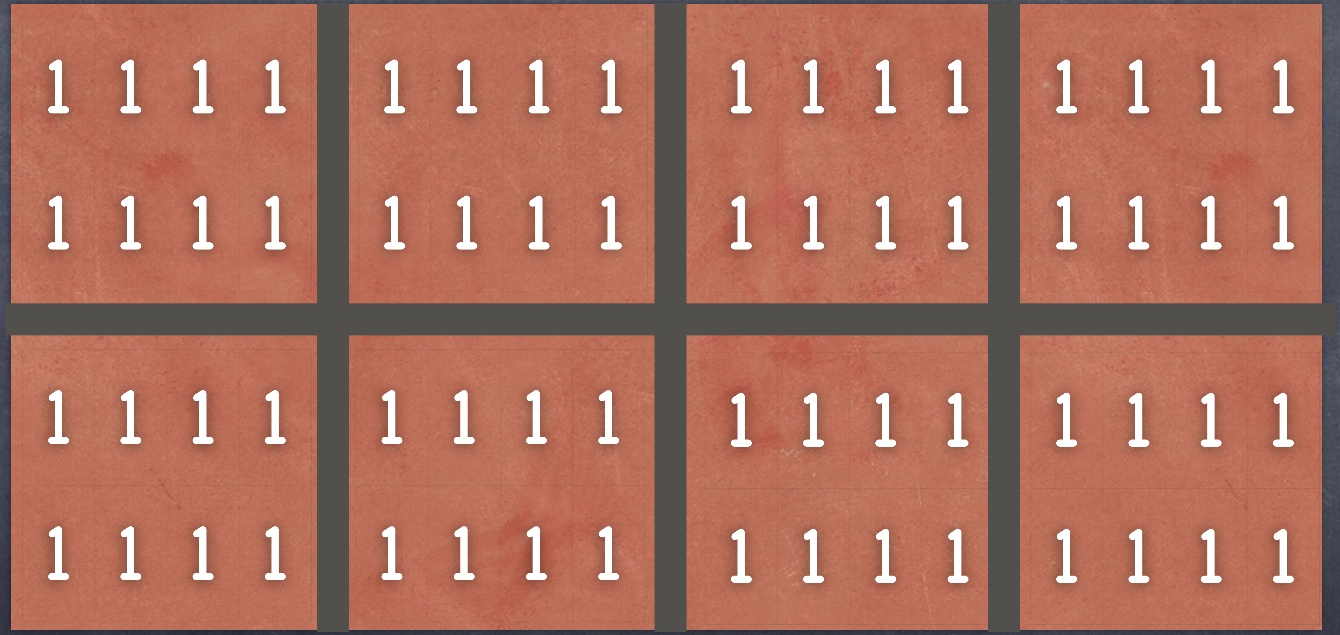
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 1 0 1 0 1
1 1 1 1 1 1 1 1	1 1 1 0 0 0 0 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1

If now we want to set this bit to 1,
we need to erase the entire block!

Modified pages must be
copied elsewhere, or lost!

Block

Erase



1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Wear Out

Every erase/program cycle adds some charge to a block; over time, hard to distinguish 1 from 0!

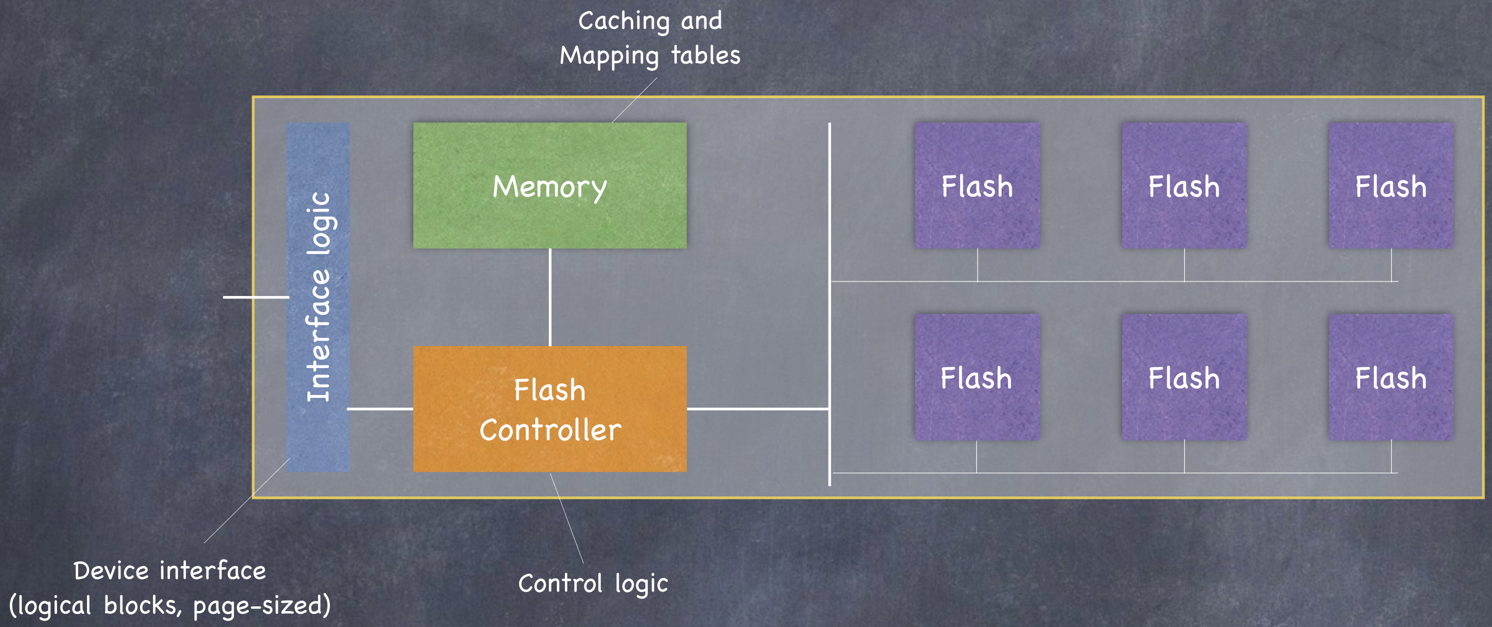
Using Flash Memory

- Need to map reads and writes to logical blocks to read, program, and erase operations on flash

↓

Flash Translation Layer (FTL)

From Flash to SSD



- Flash Translation Layer
    - tries to minimize
        - write amplification: $\left[\frac{\text{write traffic (bytes) to flash chips}}{\text{write traffic (bytes) from client to SSD}}\right]$
        - wear out: practices wear leveling
        - disturbance: writes pages in a block in order, low to high

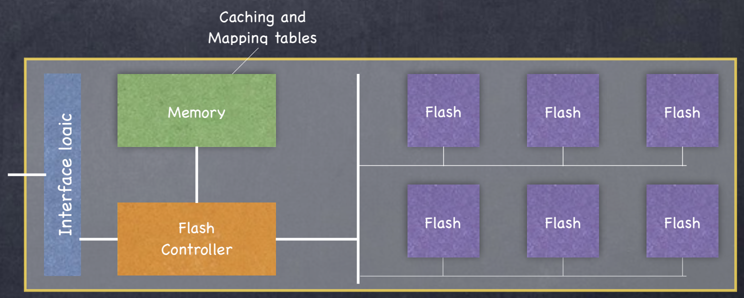
Log Structured FTL

- Think of flash storage as implementing a log

Log Structured FTL

- Think of flash storage as implementing a log

- On a write, program next available page of physical block being currently written
  - i.e., "append" the write to your log
- On a read, find in the log the page storing the logical block
  - don't want to scan the whole log...
  - keep an in-memory map from logical blocks to pages!



Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Log

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content												
State	i	i	i	i	i	i	i	i	i	i	i	i

Flash Chip

invalid state

- Client operations { Write (a1, 100)

1) Erase(00)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content												
State	E	E	E	E	i	i	i	i	i	i	i	i

Flash Chip

- Client operations { Write (a1, 100)

2) Program(00)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00

Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1												Flash Chip
State	V	E	E	E	i	i	i	i	i	i	i	i	

- Client operations { Write (a1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00

Memory

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content	a1											
State	V	E	E	E	i	i	i	i	i	i	i	i

Flash Chip

- Client operations
  - Write (a1, 100)
  - Write (a2, 101)

3) Program(01)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 Memory

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content	a1	a2										
State	V	V	E	E	i	i	i	i	i	i	i	i

Flash Chip

- Client operations
  - Write (a1, 100)
  - Write (a2, 101)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2									Flash Chip
State	V	V	V	V	i	i	i	i	i	i	i	i	

- Client operations

Write (a1, 100)
Write (a2, 101)
Write (b1, 2000)
Write (b2, 2001)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2									Flash Chip
State	V	V	V	V	i	i	i	i	i	i	i	i	

- Client operations { Write (c1, 100)

Erase(01)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2									Flash Chip
State	V	V	V	V	E	E	E	E	i	i	i	i	

- Client operations { Write (c1, 100)

Program(04)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1								Flash Chip
State	V	V	V	V	V	E	E	E	i	i	i	i	

- Client operations { Write (c1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02				Flash Chip
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1								
State	V	V	V	V	V	E	E	E	i	i	i	i	

- Client operations { Write (c1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 05 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1	c2							Flash Chip
State	V	V	V	V	V	V	E	E	i	i	i	i	

- Client operations
  - Write (c1, 100)
  - Write (c2, 101)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 05 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1	c2							Flash Chip
State	V	V	V	V	V	V	E	E	i	i	i	i	

- Client operations

Write (c1, 100)
Write (c2, 101)

APIs

	HDD	Flash
read	read sector	read page
write	write sector	program page (0's) erase block (1's)

Performance

	HDD	Flash
Throughput	≈ 130MB/s (sequential)	≈200MB/s (random or sequential)
Latency	≈ 10ms	read 25μs program 200-300μs erase 1.5-2 ms

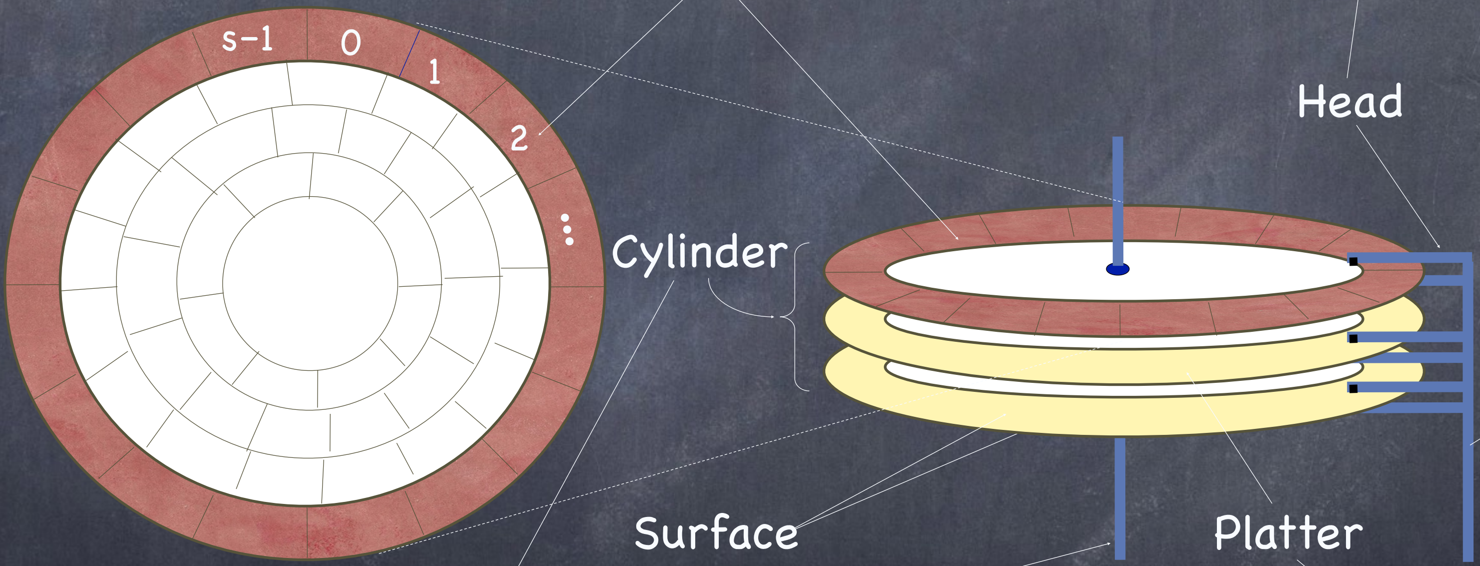
Input/Output

- But, the OS must support all hardware devices ….
- Devices have different internal architecture/design:
  - OS: how can we standardize the interfaces to these devices?
  - Answer: OS File system, OS block layer (1 lecture)
- Devices are unreliable:
  - OS: how can we give the illusion of a reliable device?
  - Answer: RAID design (1 lecture)
- Devices have unpredictable performance:
  - OS: how can we manage them if we don't know device behavior
  - Answer: OS schedulers (1 lecture)

Magnetic disk

- Store data magnetically on thin metallic film bonded to rotating disk of glass, ceramic, or aluminum

Disk Drive Schematic



Track

data on a track can be read without moving arm

track skewing staggers logical address 0 on adjacent one to account for time to move head

Block/Sector

Typically 512 bytes
spare sectors added for fault tolerance

Head

reads by sensing a magnetic field

writes by creating one

floats on air cushion created by spinning disk

Cylinder

set of tracks on different surfaces with same track index

Surface

Spindle

2018: 4200-15000 RPM

Platter

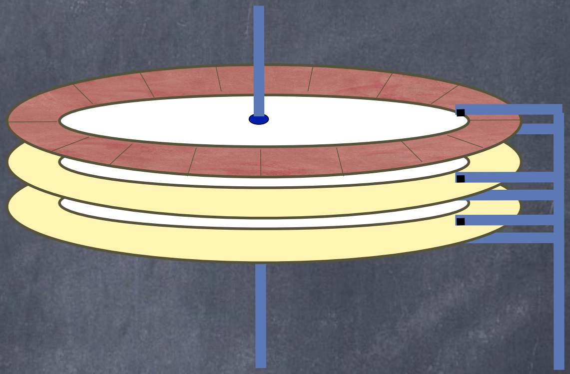
thin cylinder that holds magnetic material

each platter has two surfaces

Arm assembly

Disk Read/Write

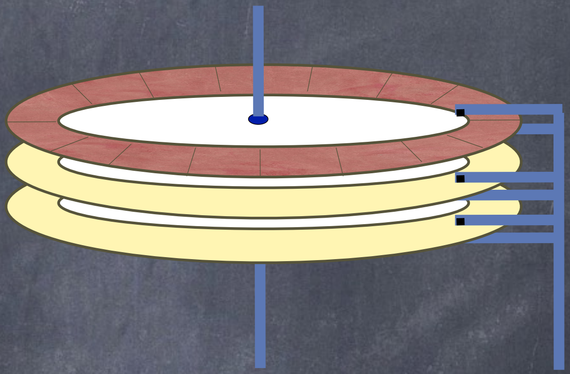
- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek
  - settle
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time



Disk access time:

Disk Read/Write

- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek (and though shalt approximately find)
  - settle (fine adustments)
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time

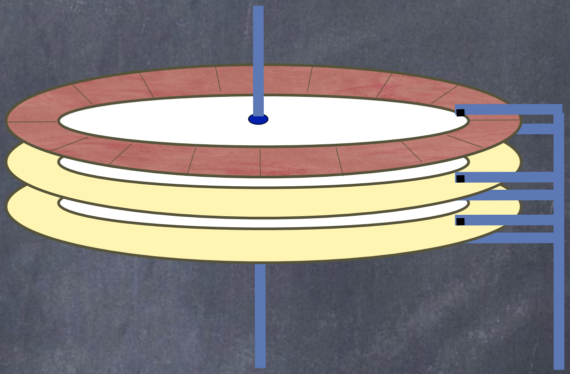


Disk access time:

seek time +

Disk Read/Write

- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek (and though shalt approximately find)
  - settle (fine adustments)
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time



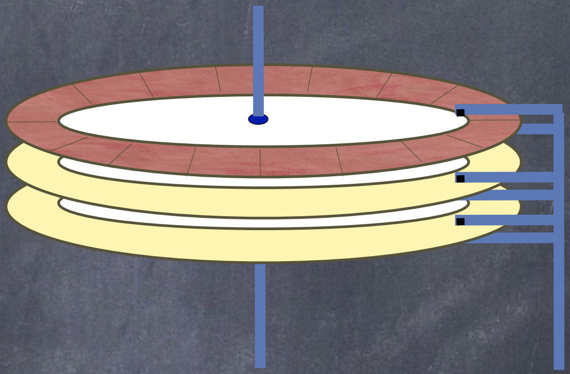
Disk access time:

seek time +

rotation time +

Disk Read/Write

- Present disk with a sector address
  - Old: CHS = (cylinder, head, sector)
  - New abstraction: Logical Block Address (LBA)
    - linear addressing 0…N-1
- Heads move to appropriate track
  - seek (and though shalt approximately find)
  - settle (fine adustments)
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotational latency
- Read/Write sector
  - transfer time



Disk access time:

seek time +

rotation time +

transfer time

Seek time:
A closer look

- Minimum: time to go from one track to the next
  - 0.3-1.5 ms
- Maximum: time to go from innermost to outermost track
  - more than 10ms; up to over 20ms
- Average: average across seeks between each possible pair of tracks
  - approximately time to seek 1/3 of the way across disk

Seek time: A closer look

- Minimum: time to go from one track to the next
  - 0.3-1.5 ms
- Maximum: time to go from innermost to outermost track
  - more than 10ms; up to over 20ms
- Average: average across seeks between each possible pair of tracks
  - approximately time to seek 1/3 of the way across disk
- Head switch time: time to move from track on one surface to the same track on a different surface
  - range similar to minimum seek time

Rotation time:
A closer look

- Today most disk rotate at 4200 to 15,000 RPM
  - ≈15ms to 4ms per rotation
  - good estimate for rotational latency is half that amount
- Head starts reading as soon as it settles on a track
  - track buffering to avoid "shoulda coulda" if any of the sectors flying under the head turn out to be needed

Transfer time: A closer look

- Surface transfer time
  - Time to transfer one or more sequential sectors to/from surface after head reads/writes first sector
  - Much smaller that seek time or rotational latency
    - 512 bytes at 100MB/s ≈ 5μs (0.005 ms)
  - Lower for outer tracks than inner ones
    - same RPM, but more sectors/track: higher bandwidth!
- Host transfer time
  - time to transfer data between host memory and disk buffer
    - 60MB/s (USB 2.0); 640 MB/s (USB 3.0); 25.GB/s (Fibre Channel 256GFC)

Buffer Memory

- Small cache [``Track buffer”, 8 to 16 MB] holds data
  - read from disk
  - about to be written to disk
- On write
  - write back (return from write as soon as data is cached)
  - write through (return once it is on disk)

Computing I/O time

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

- The rate of I/O is computed as

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/0}}$$

Example: Toshiba MK3254GSY (2008)

Size	
Platters/Heads	2/4
Capacity	320GB
Performance	
Spindle speed	7200 RPM
Avg. seek time R/W	10.5/12.0 ms
Max. seek time R/W	19 ms
Track-to-track	1 ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer memory	16MB
Power	
Typical	16.35 W
Idle	11.68 W

500 Random Reads

Size	
Platters/Heads	2/4
Capacity	320GB
Performance	
Spindle speed	7200 RPM
Avg. seek time R/W	10.5/12.0 ms
Max. seek time R/W	19 ms
Track-to-track	1 ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer memory	16MB
Power	
Typical	16.35 W
Idle	11.68 W

- Workload
  - 500 read requests, randomly chosen sector
  - served in FIFO order
- How long to service them?
  - 500 times (seek + rotation + transfer)
  - seek time: 10.5 ms (avg)
  - rotation time:
    - 7200 RPM = 120 RPS
    - rotation time 8.3 ms
    - on average, half of that: 4.15 ms
  - transfer time
    - at least 54 MB/s
    - 512 bytes transferred in (.5/54,000) seconds = 9.26μs
  - Total time:
    - 500 x (10.5 + 4.15 + 0.009) ≈ 7.33 sec

$$R_{I/O} = \frac{500 \times .5 \times 10^{-3} MB}{7.33\, s} = 0.034\, MB/s$$

500 Sequential Reads

Size	
Platters/Heads	2/4
Capacity	320GB
Performance	
Spindle speed	7200 RPM
Avg. seek time R/W	10.5/12.0 ms
Max. seek time R/W	19 ms
Track-to-track	1 ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer memory	16MB
Power	
Typical	16.35 W
Idle	11.68 W

- Workload
  - 500 read requests for sequential sectors on the same track
  - served in FIFO order
- How long to service them?
  - seek + rotation + 500 times transfer
  - seek time: 10.5 ms (avg)
  - rotation time:
    - 4.15 ms, as before
  - transfer time
    - outer track: 500 x (.5/128000) ≈ 2ms
    - inner track: 500 x (.5/54000) seconds ≈ 4.6ms
  - Total time is between:
    - outer track: (2 + 4.15 + 10.5) ms ≈ 16.65 ms

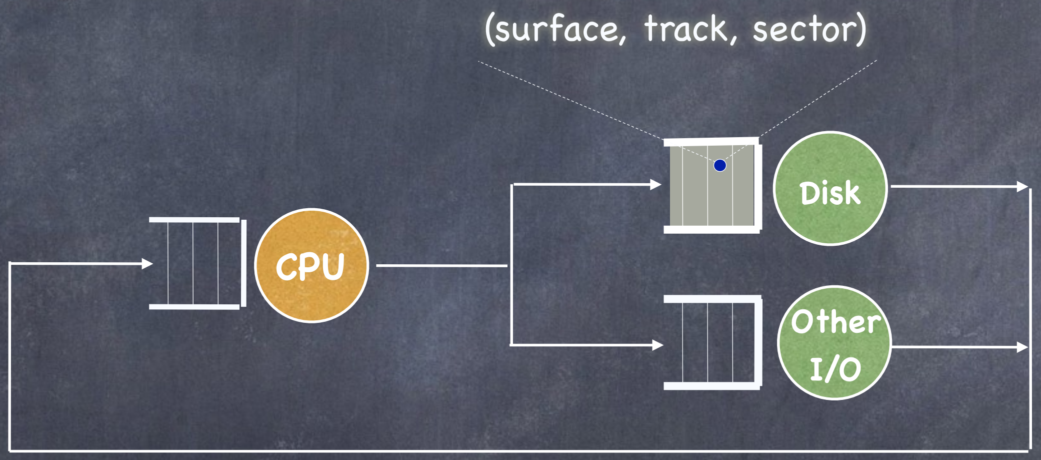
$$R_{I/O} = \frac{500 \times .5 \times 10^{-3} MB}{16.65\, ms} = 15.02\, MB/s$$

    - inner track: (4.6 + 4.15 + 10.5) ms ≈ 19.25 ms

$$R_{I/O} = \frac{500 \times .5 \times 10^{-3} MB}{19.25\, ms} = 12.99\, MB/s$$

Disk Head Scheduling

- In a multiprogramming/time sharing environment, a queue of disk I/Os can form



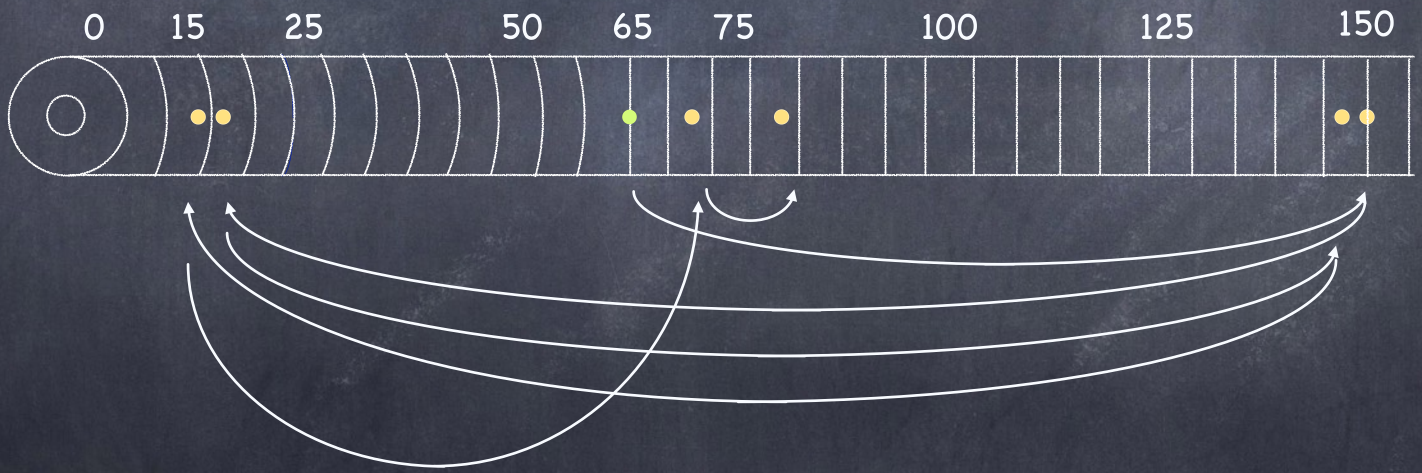
- OS maximizes disk I/O throughput by minimizing head movement through disk head scheduling
  - and this time we have a good sense of the length of the task!

FCFS

- Assume a queue of request exists to read/write tracks

.....	83	72	14	147	16	150

and the head is on track 65



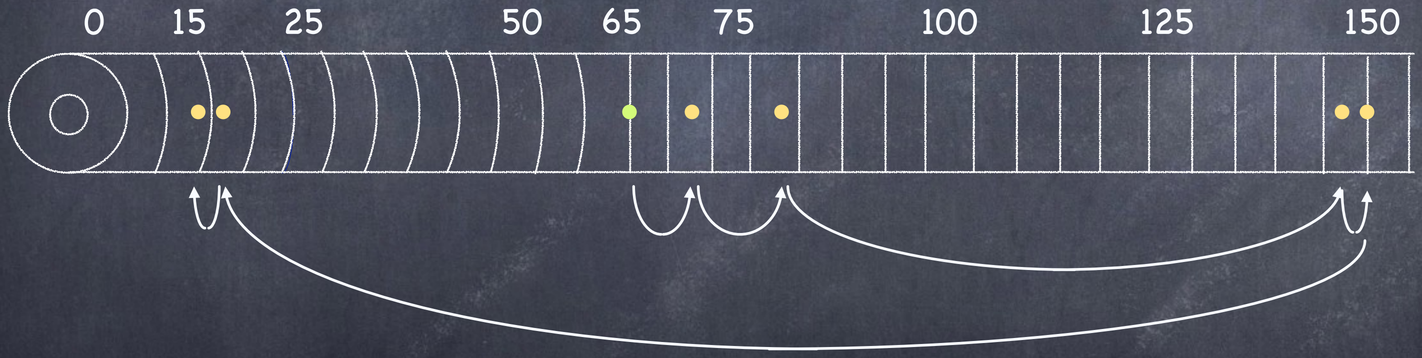
FCFS scheduling results in disk head moving 550 tracks

and makes no use of what we know about the length of the tasks!

SSTF:
Shortest Seek Time First

- Greedy scheduling

Rearrange queue from:	.....	83	72	14	147	16	150
to:	.....	14	16	150	147	83	72



Head moves 221 tracks **BUT**

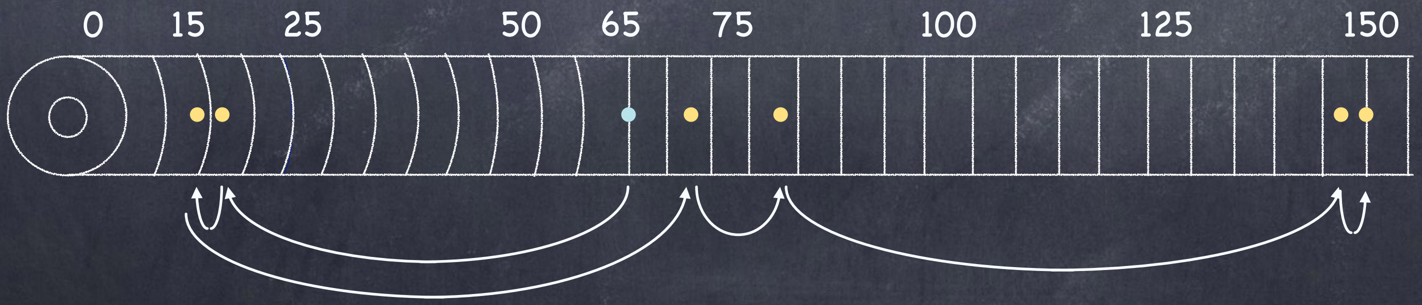
☐ OS knows blocks, not tracks (easily fixed)

☐ starvation

SCAN Scheduling
“Elevator”

- Move the head in one direction until all requests have been serviced, and then reverse
    - sweeps disk back and forth

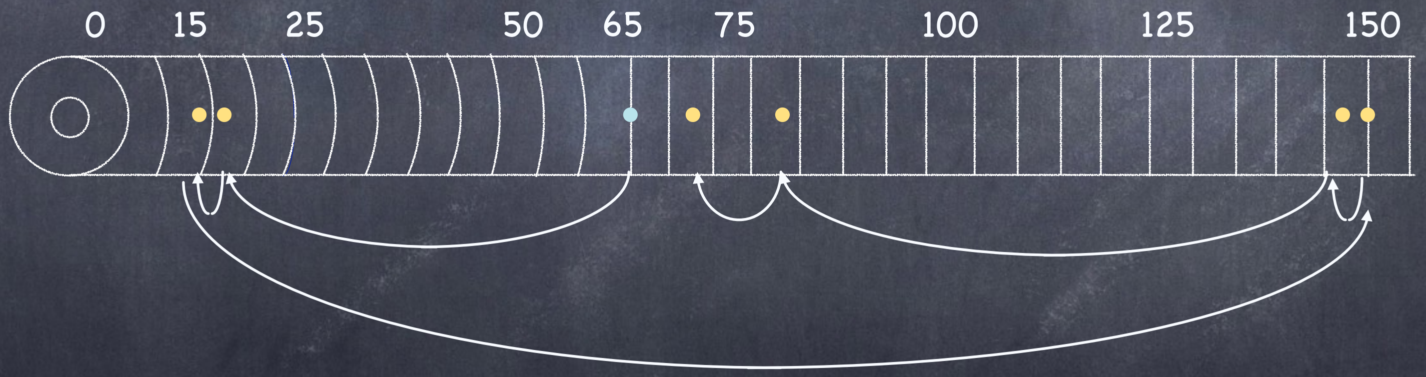
Rearrange queue from:	.....	83	72	14	147	16	150
to:	.....	150	147	83	72	14	16



Head moves 187 tracks.

C-SCAN scheduling

- Circular SCAN
  - sweeps disk in one direction (from outer to inner track), then resets to outer track and repeats



- More uniform wait time than SCAN
  - moves head to serve requests that are likely to have waited longer

OS Outsources Scheduling Decisions

- Selecting which track to serve next should include rotation time (not just seek time!)
  - SPTF: Shortest Positioning Time First
- Hard for the OS to estimate rotation time accurately
  - Hierarchical decision process
    - OS sends disk controller a batch of "reasonable" requests
    - disk controller makes final scheduling decisions