

# CS 318 Principles of Operating Systems

Fall 2022

## Lecture 14: I/O & Disks



JOHNS HOPKINS  
WHITING SCHOOL  
of ENGINEERING

**Prof. Ryan Huang**

# Overview

**We've covered OS abstractions for CPU and memory so far**

## Virtualization

Processes

Scheduling

Virtual Memory

## Concurrency

Threads

Synchronization

Semaphores and Monitors

## Persistence

I/O

Disks

File Systems

**I/O management is another major component of OS**

- Important aspect of computer operation
- I/O devices vary greatly: various methods to control them
- New types of devices

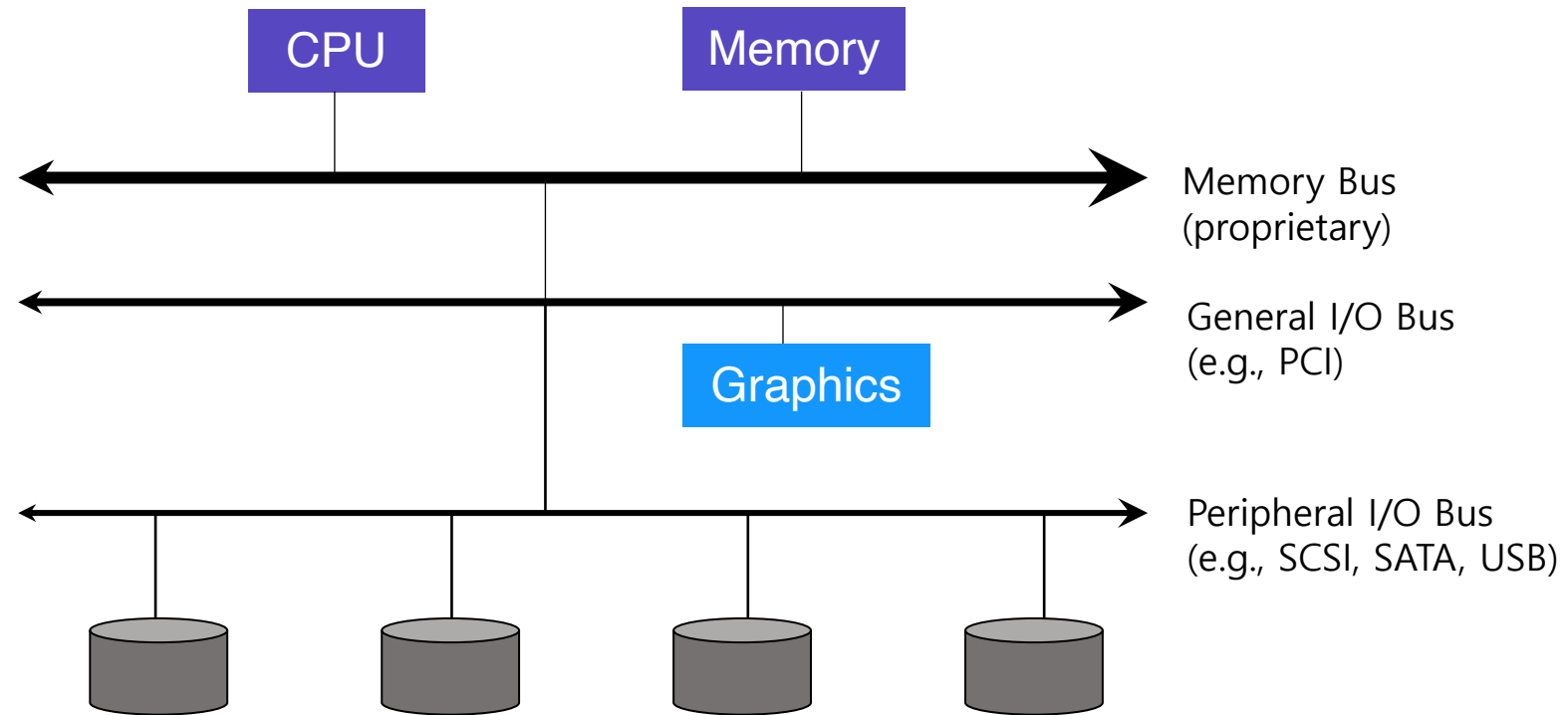
# I/O Devices



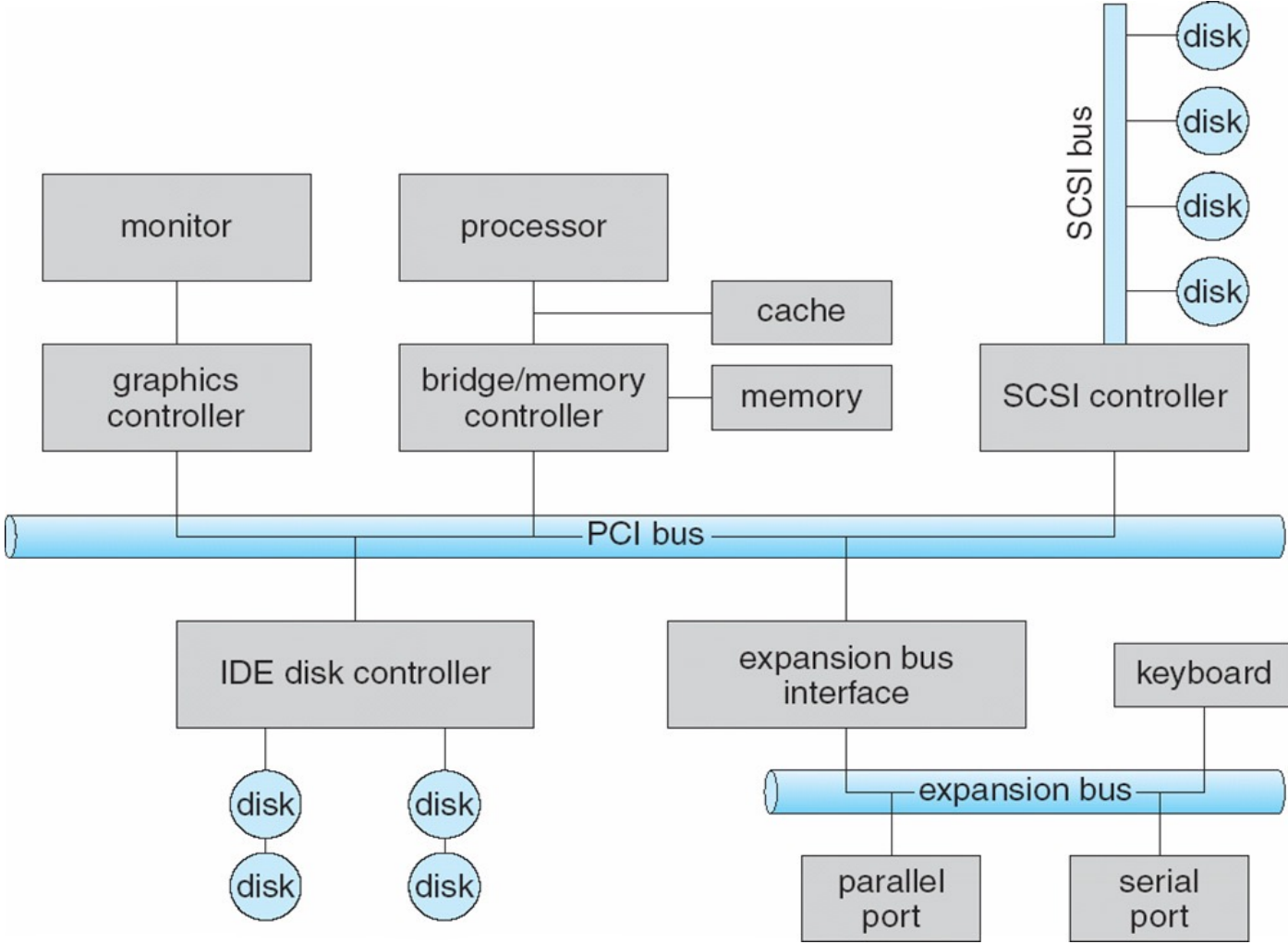
## Issues to address:

- How should I/O be integrated into systems?
- What are the general mechanisms?
- How can we manage them efficiently?

# Structure of Input/Output (I/O) Device

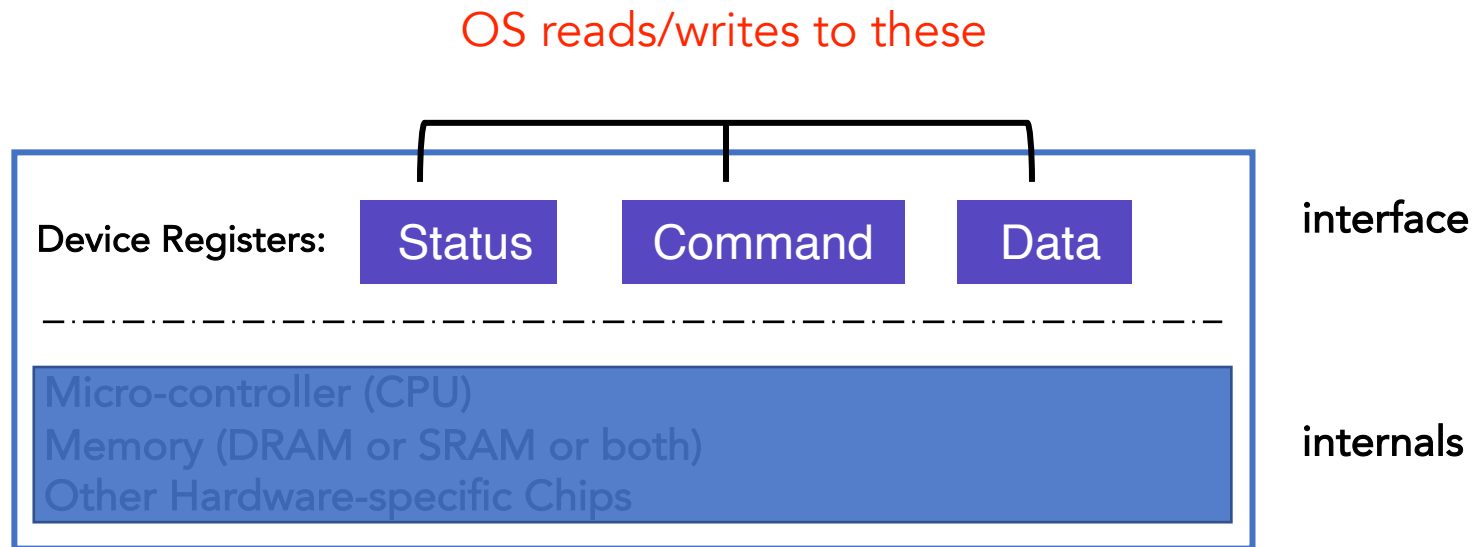


# Structure of Input/Output (I/O) Device



# Device Interaction

How does the OS communicate with an I/O device?



Canonical I/O Device

# Hardware Interface Of Canonical Device

## status register

- See the current status of the device

## command register

- Tell the device to perform a certain task

## data register

- Pass data to the device, or get data from the device

**By reading or writing the three registers, OS controls device behavior**

# Hardware Interface Of Canonical Device

## Typical interaction example

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



# Programming a device

## One approach: I/O instructions

- `in` and `out` instructions on x86
- Devices usually have registers
  - places commands, addresses, and data there to read/write registers
- How to identify (address) a device?
  - With a port location (I/O address range)

# Typical Device I/O Port Locations

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

# x86 I/O instructions

```
static inline uint8_t inb (uint16_t port)
{
    uint8_t data;
    asm volatile ("inb %w1, %b0" : "=a" (data) : "Nd" (port));
    return data;
}
```

```
static inline void outb (uint16_t port, uint8_t data)
{
    asm volatile ("outb %b0, %w1" : : "a" (data), "Nd" (port));
}
```

```
static inline void insw (uint16_t port, void *addr, size_t cnt)
{
    asm volatile ("rep insw" : "+D" (addr), "+c" (cnt)
                 : "d" (port) : "memory");
}
```

Pintos threads/io.h

# IDE Disk Driver

```
void IDE_ReadSector(int disk, int off,
    void *buf)
{
    // Select Drive
    outb(0x1F6, disk == 0 ? 0xE0 : 0xF0);
    IDEWait();
    // Read length (1 sector = 512 B)
    outb(0x1F2, 1);
    outb(0x1F3, off); // LBA low
    outb(0x1F4, off >> 8); // LBA mid
    outb(0x1F5, off >> 16); // LBA high
    outb(0x1F7, 0x20); // Read command
    insw(0x1F0, buf, 256); // Read 256 words
}
```

```
void IDEWait()
{
    // Discard status 4 times
    inb(0x1F7); inb(0x1F7);
    inb(0x1F7); inb(0x1F7);
    // Wait for status BUSY flag to clear
    while ((inb(0x1F7) & 0x80) != 0);
}
```

# Memory-mapped IO

## in/out instructions slow and clunky

- Instruction format restricts what registers you can use
- Only allows  $2^{16}$  different port numbers

## Another approach: Memory-mapped I/O

- Device registers available as if they were memory locations. `load` (to read) or `store` (to write) goes to the device instead of main memory.

```
volatile int32_t *device_control
    = (int32_t *) (0xc0100 + PHYS_BASE);
*device_control = 0x80;
int32_t status = *device_control;
```

- OS must map physical to virtual addresses, ensure non-cachable

# Polling

OS waits until the device is ready by repeatedly reading the **status register**

- Positive aspect is simple and working.
- However, it wastes CPU time just waiting for the device
  - Switching to another ready process is better utilizing the CPU.

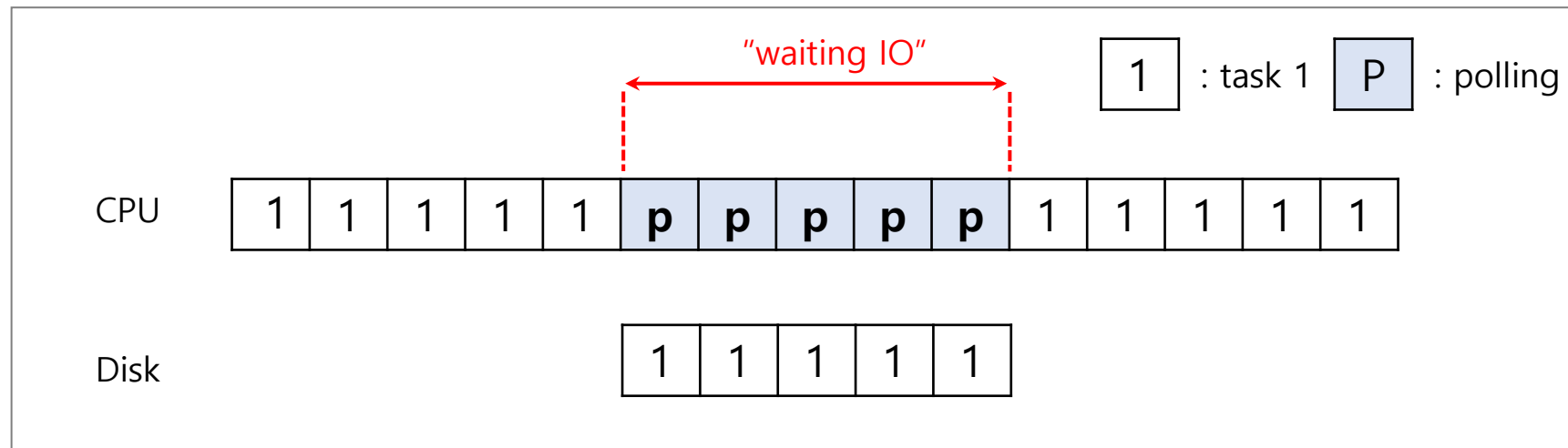


Diagram of CPU utilization by polling



# Polling vs Interrupts

**However, “interrupts is not always the best solution”**

- If, device performs very quickly, interrupt will “slow down” the system.

If a device is fast → **poll** is best  
If it is slow → **interrupt** is better

**E.g., high network packet arrival rate**

- Packets can arrive faster than OS can process them
- Interrupts are very expensive (context switch)
- Interrupt handlers have high priority
- In worst case, can spend 100% of time in interrupt handler and never make any progress

**Adaptive switching between interrupts and polling**



# One More Problem: Data Copying

CPU wastes a lot of time in copying large *data* from memory to a device register one byte a time (termed *programmed I/O, PIO*)

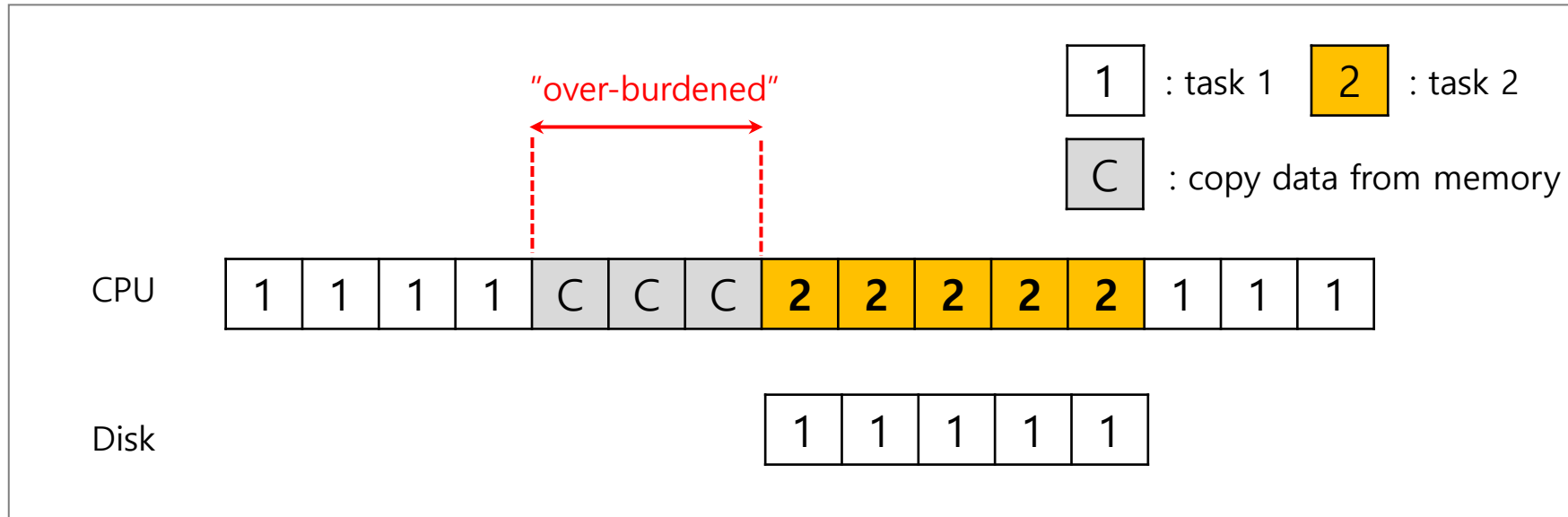
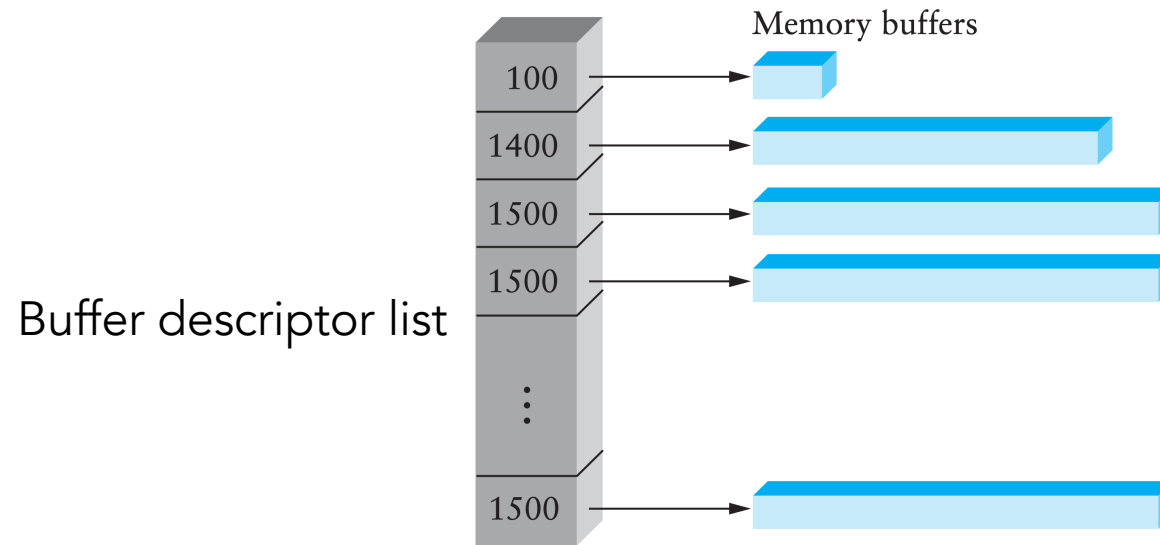


Diagram of CPU utilization

# DMA (Direct Memory Access)



**Idea:** only use CPU to transfer control requests, not data

**Include list of buffer locations in main memory**

- Device reads list and accesses buffers through DMA

# DMA (Direct Memory Access) Cont.

When completed, DMA raises an interrupt, I/O begins on Disk.

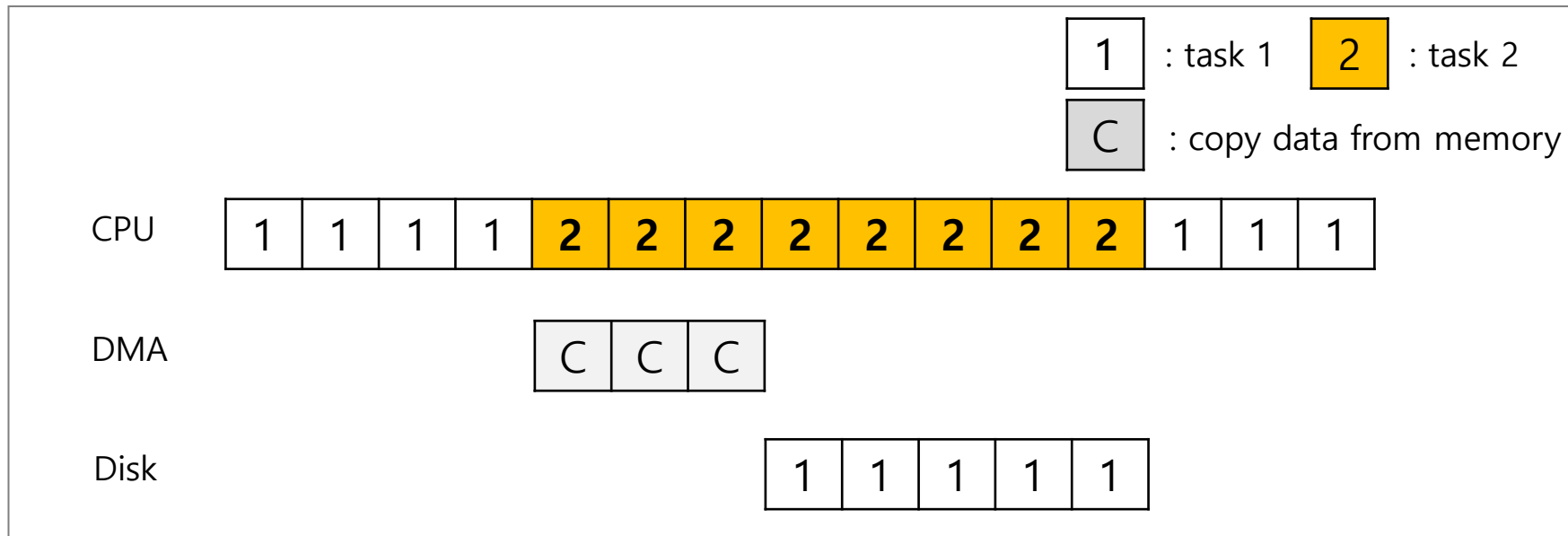


Diagram of CPU utilization by DMA

# Direct Memory Access

**Avoid programmed I/O for large data movement**

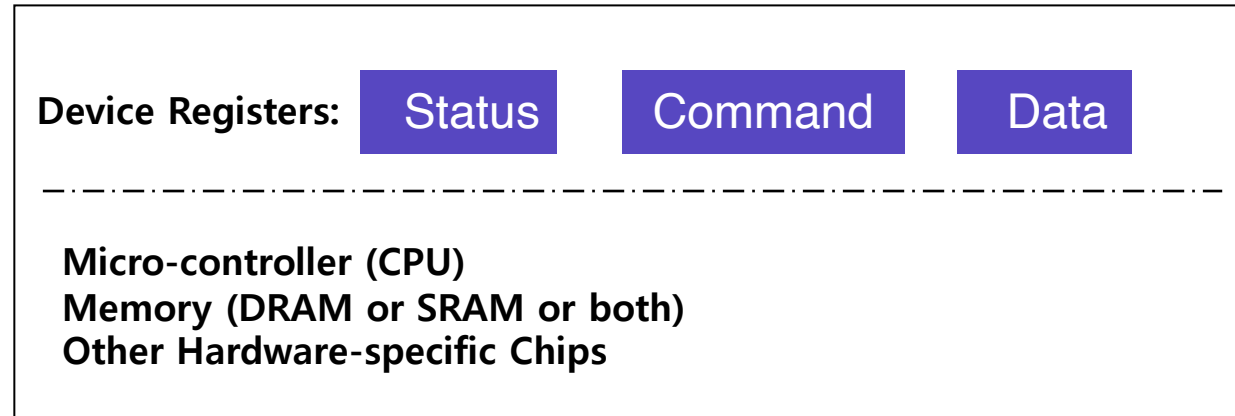
**Requires DMA controller**

**Bypasses CPU to transfer data directly between I/O device and memory**

**OS writes DMA command block into memory**

- Source and destination addresses
- Read or write mode
- Count of bytes
- Writes location of command block to DMA controller

# Device Protocol Variants



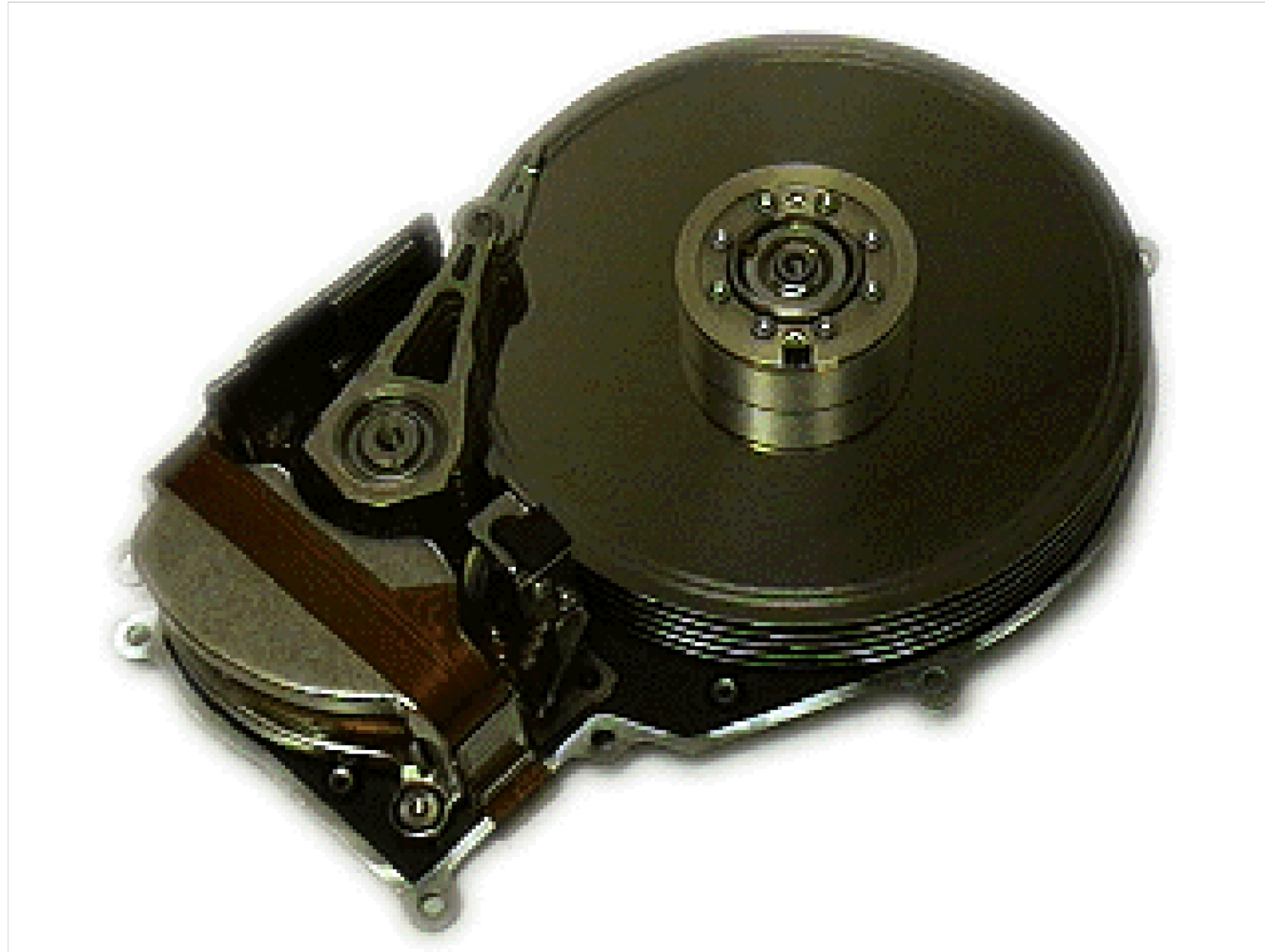
Canonical I/O Device

Status checks: *polling* vs. *interrupts*

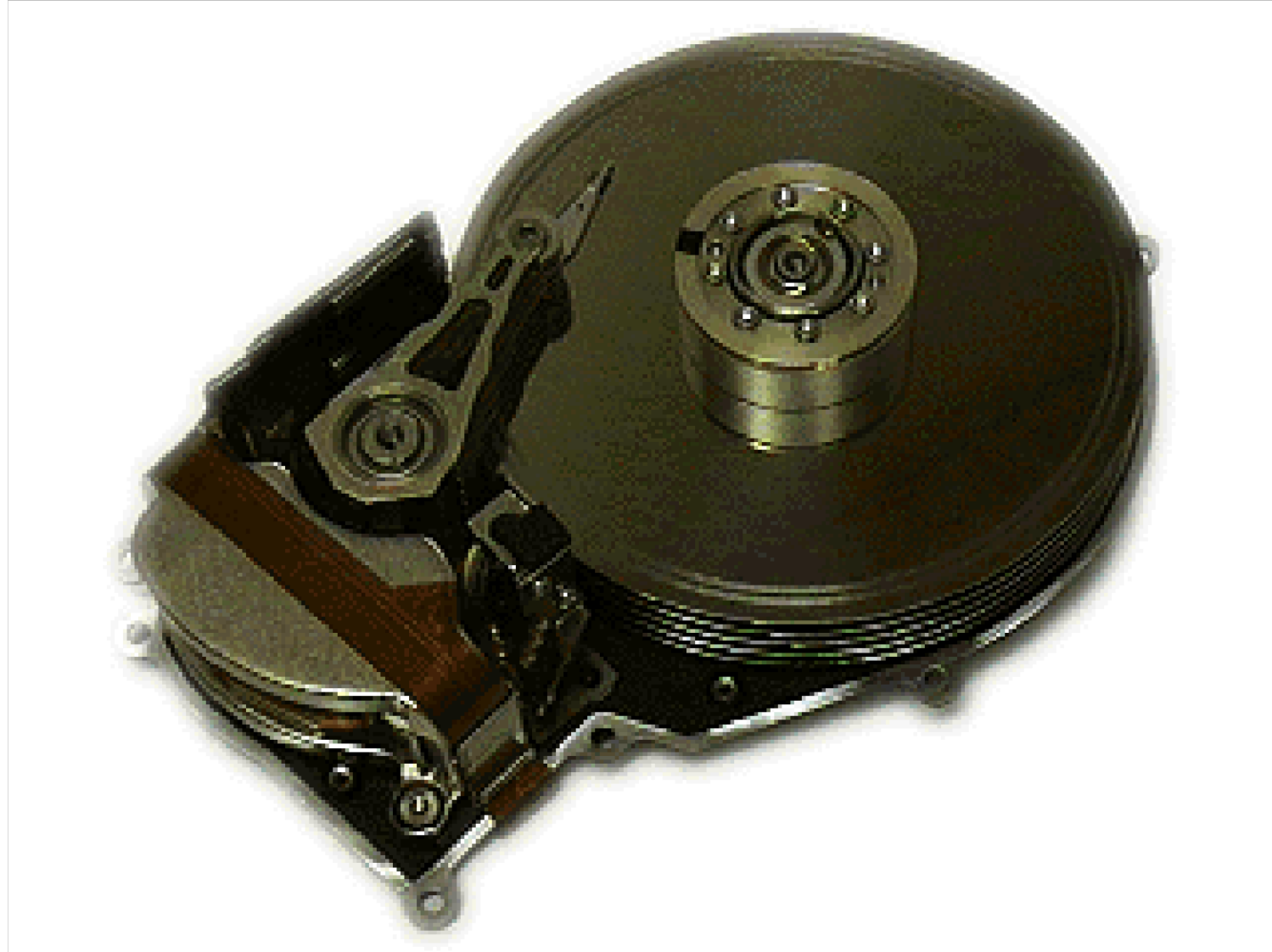
Command: *special instructions* vs. *memory-mapped I/O*

Data: *programmed I/O* (PIO) vs. *direct memory access* (DMA)

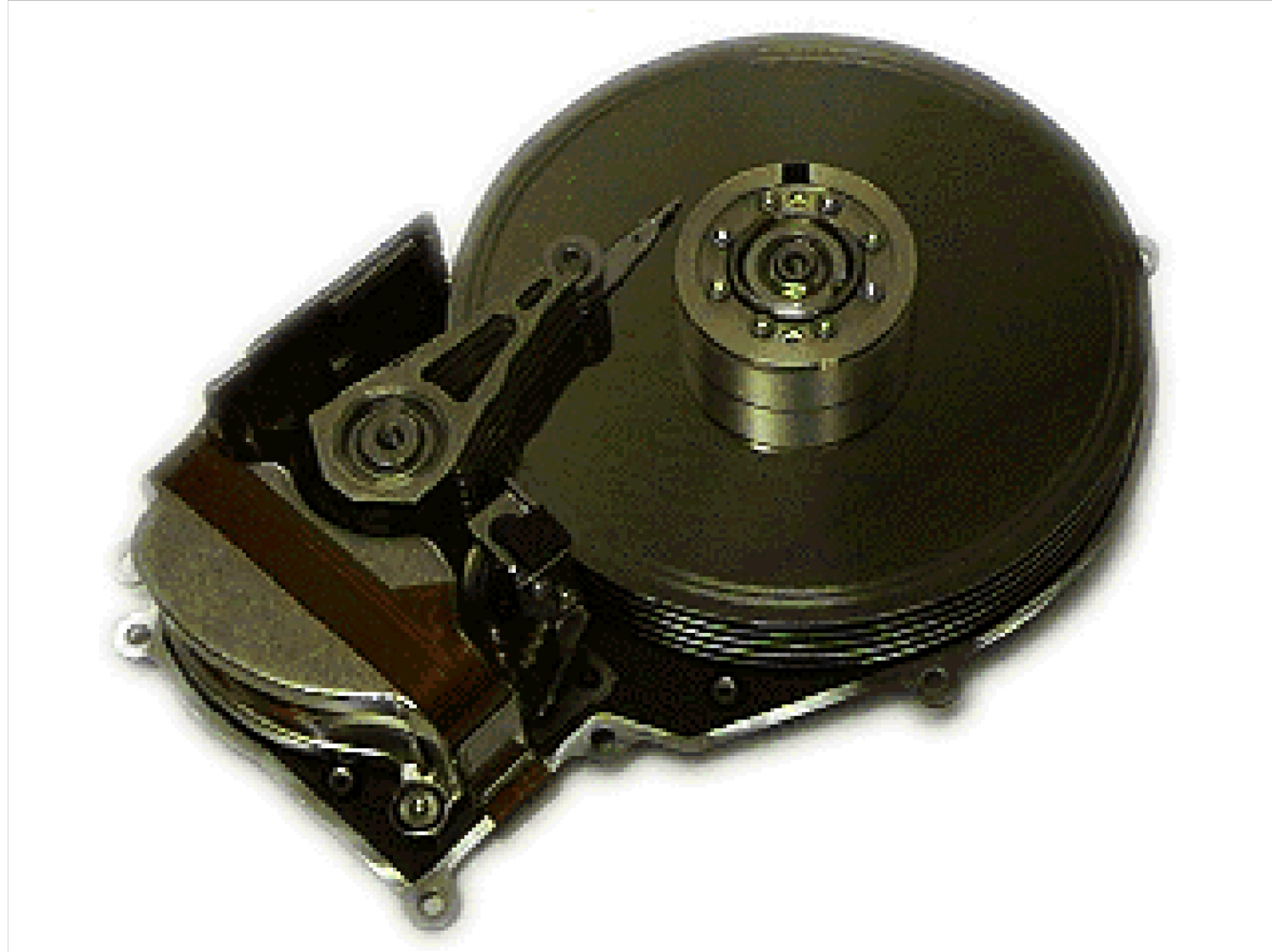
# Hard Disks



# Hard Disks



# Hard Disks





# Basic Interface

## Disk interface presents linear array of sectors

- Historically **512 Bytes**
- Written atomically (even if there is a power failure)
- 4 KiB in “advanced format” disks
  - Torn write: If an untimely power loss occurs, only a portion of a larger write may complete

## Disk maps logical sector #s to physical sectors

## OS doesn't know logical to physical sector mapping

# Basic Geometry



## **Platter** (Aluminum coated with a thin magnetic layer)

- A circular hard surface
- Data is stored persistently by inducing magnetic changes to it
- Each platter has 2 sides, each of which is called a **surface**

# Basic Geometry (Cont.)

## Spindle

- Spindle is connected to a motor that spins the platters around
- The rate of rotations is measured in **RPM** (Rotations Per Minute)
  - Typical modern values : 7,200 RPM to 15,000 RPM.

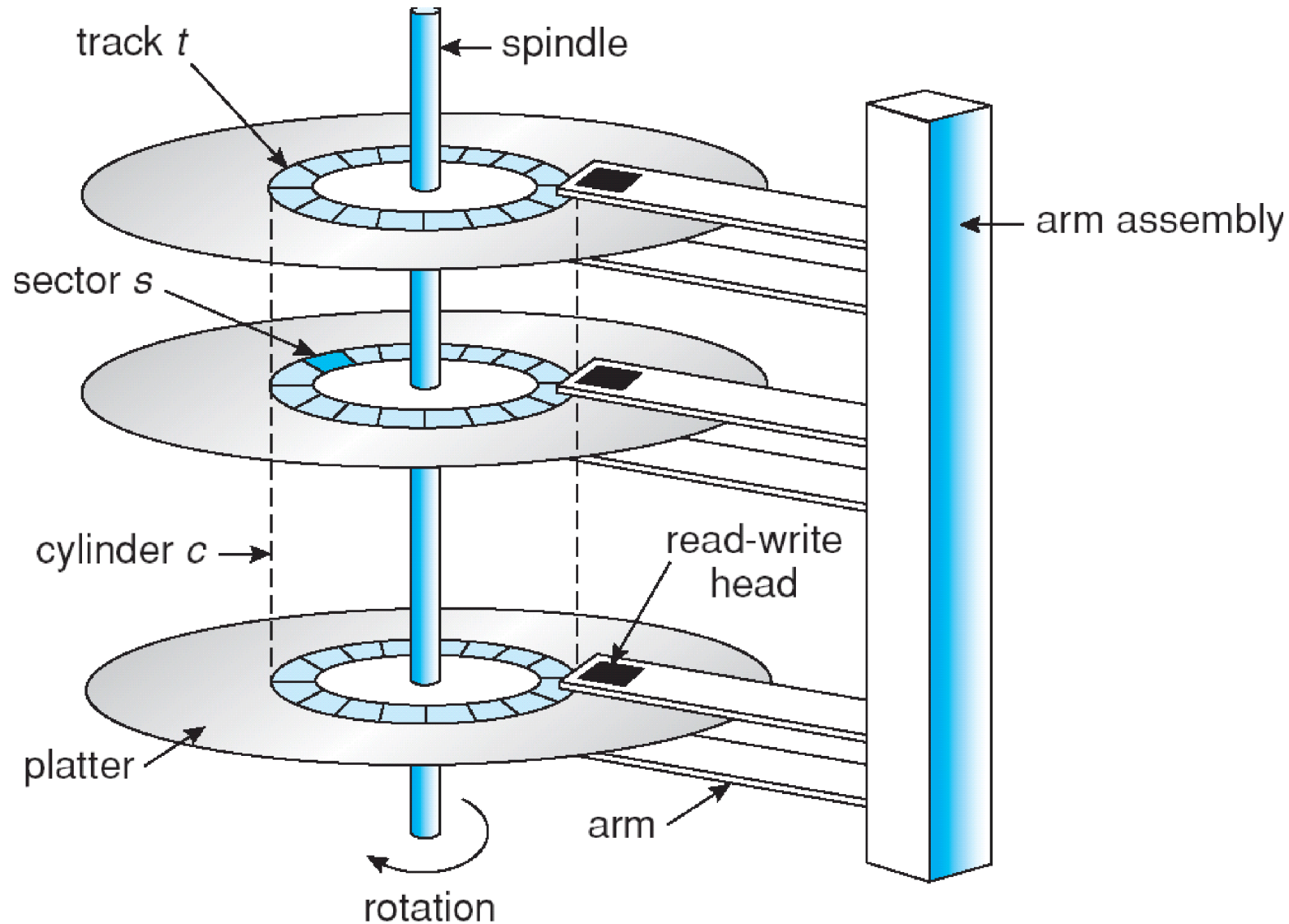
## Track

- Concentric circles of **sectors**
- Data is encoded on each surface in a track
- A single surface contains many thousands and thousands of tracks

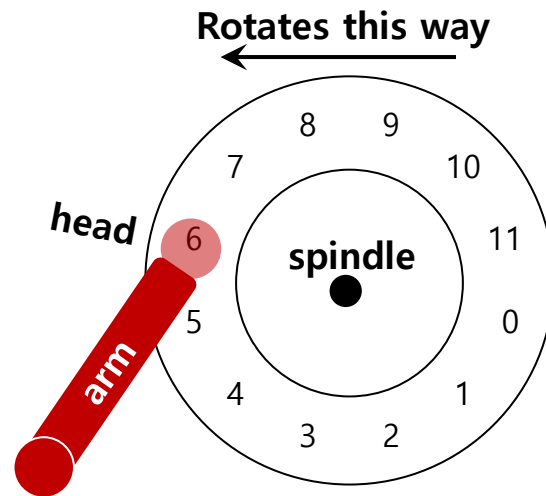
## Cylinder

- A stack of tracks of fixed radius
- Heads record and sense data along cylinders
- Generally only one head active at a time

# Cylinders, Tracks, & Sectors



# A Simple Disk Drive

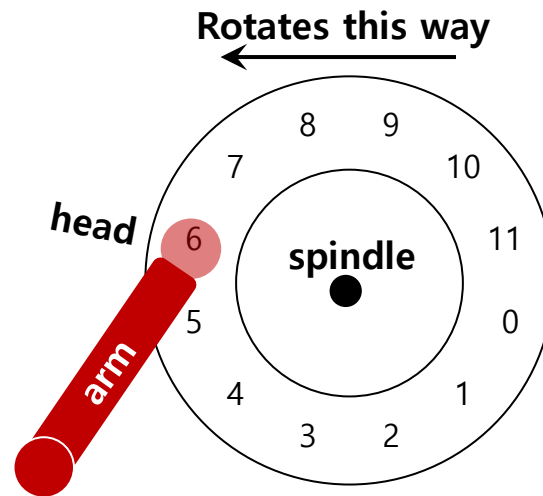


A Single Track Plus A Head

## Disk head (one head per surface of the drive)

- The process of reading and writing is accomplished by the disk head
- Attached to a single disk arm, which moves across the surface

# Single-track Latency



A Single Track Plus A Head

**Rotational delay:** Time for the desired sector to rotate

- Ex) Full rotational delay is  $R$  and we start at sector 6
  - Read sector 0: Rotational delay =  $\frac{R}{2}$
  - Read sector 5: Rotational delay =  $R-1$  (worst case.)

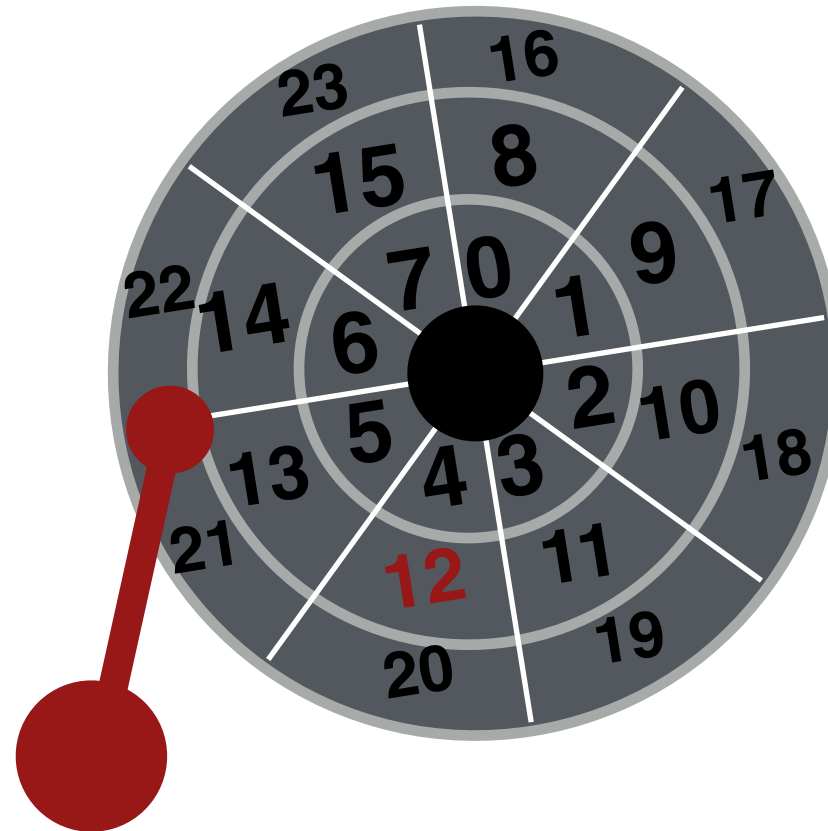
# Multiple Tracks

Let's Read 12!



# Multiple Tracks: Seek To Right Track

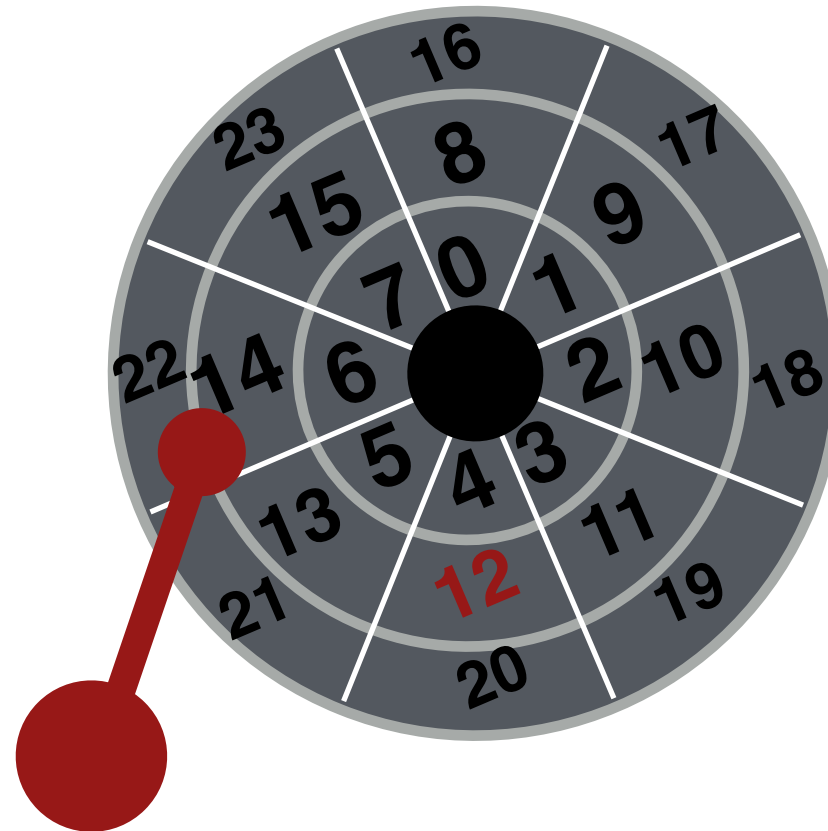
Let's Read 12!





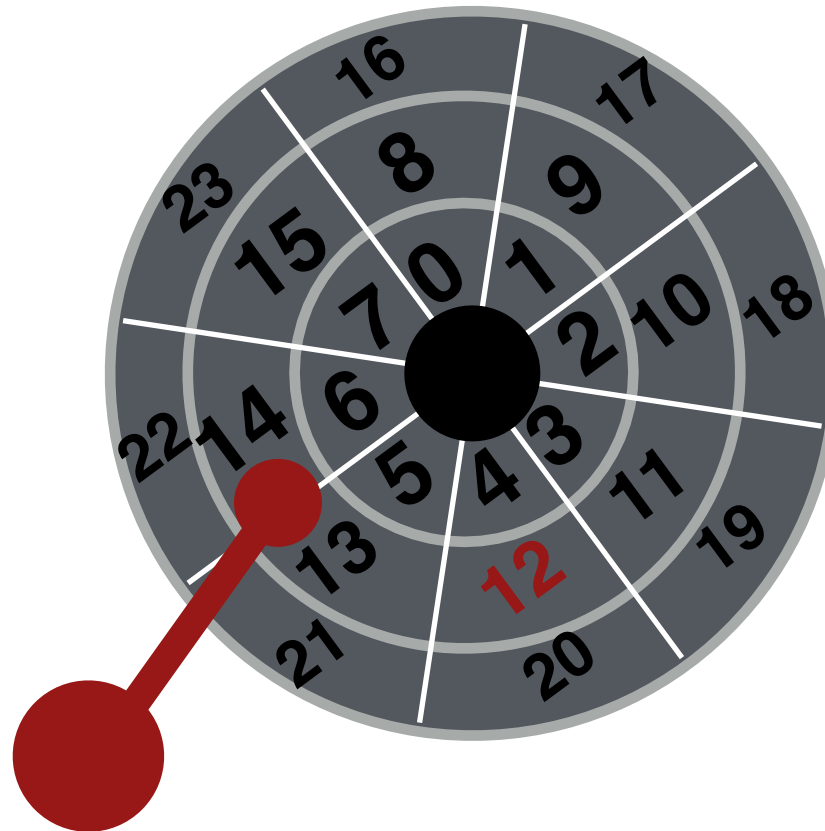
# Multiple Tracks: **Seek To Right Track**

Let's Read 12!



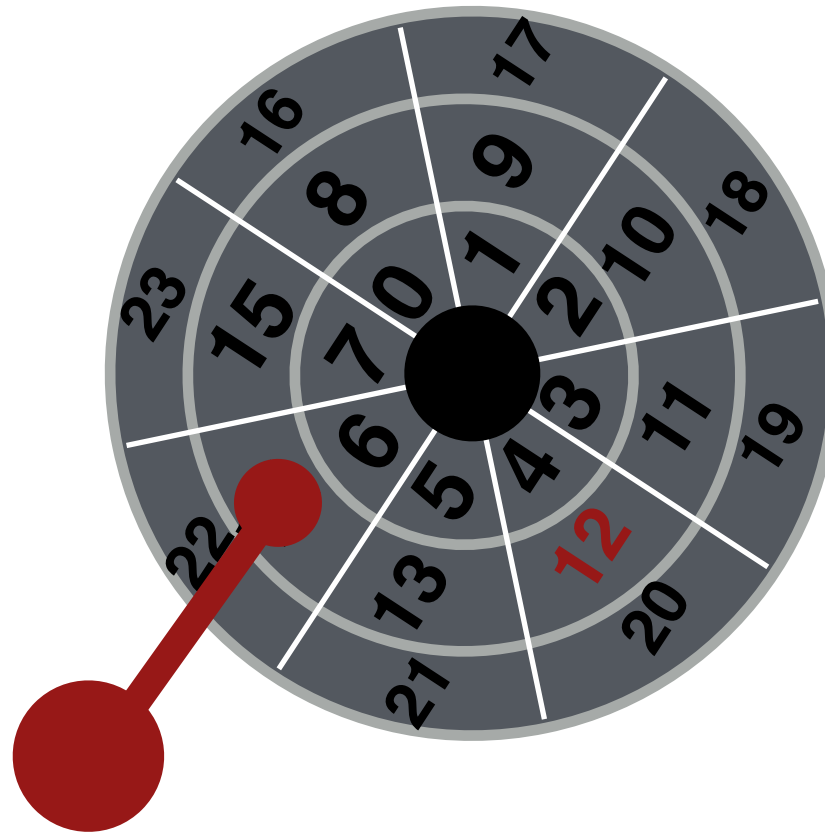
# Multiple Tracks: **Seek To Right Track**

Let's Read 12!



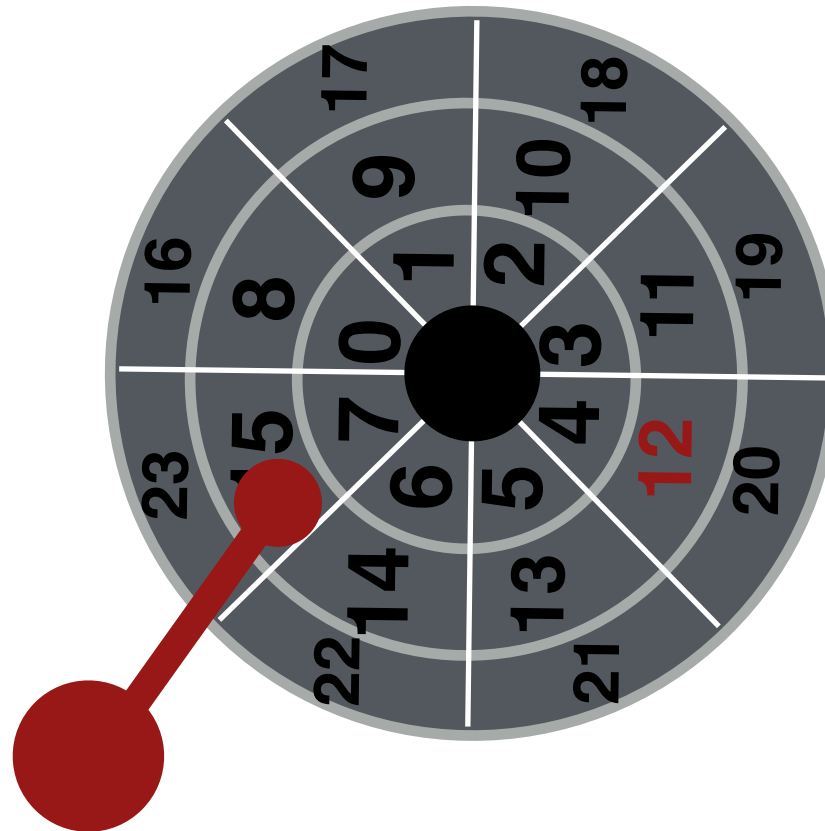
# Multiple Tracks: Wait for Rotation

Let's Read 12!



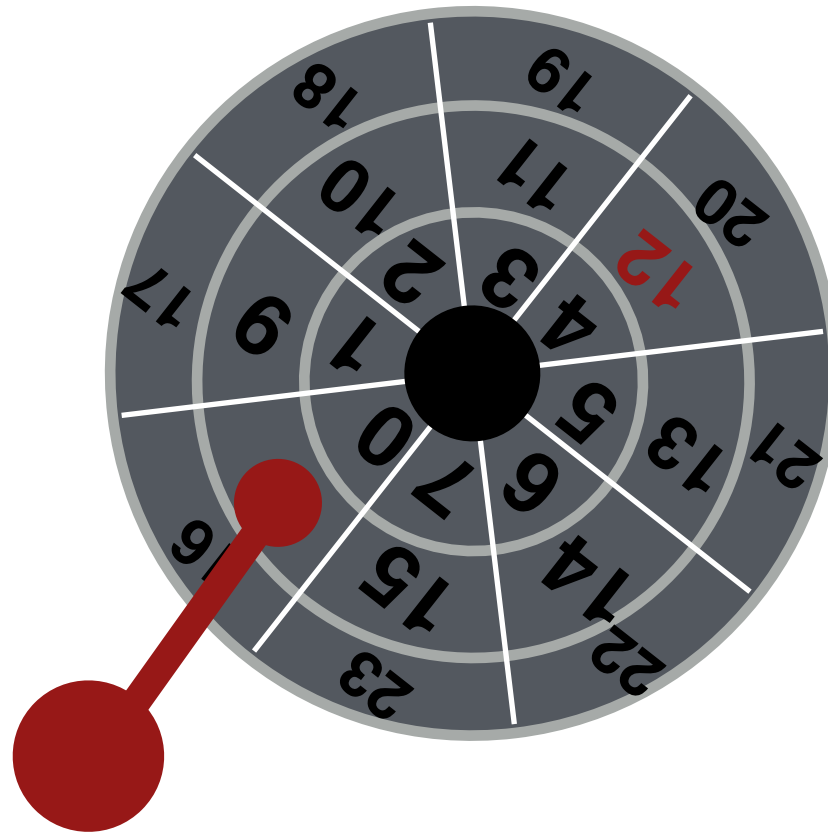
# Multiple Tracks: Wait for Rotation

Let's Read 12!



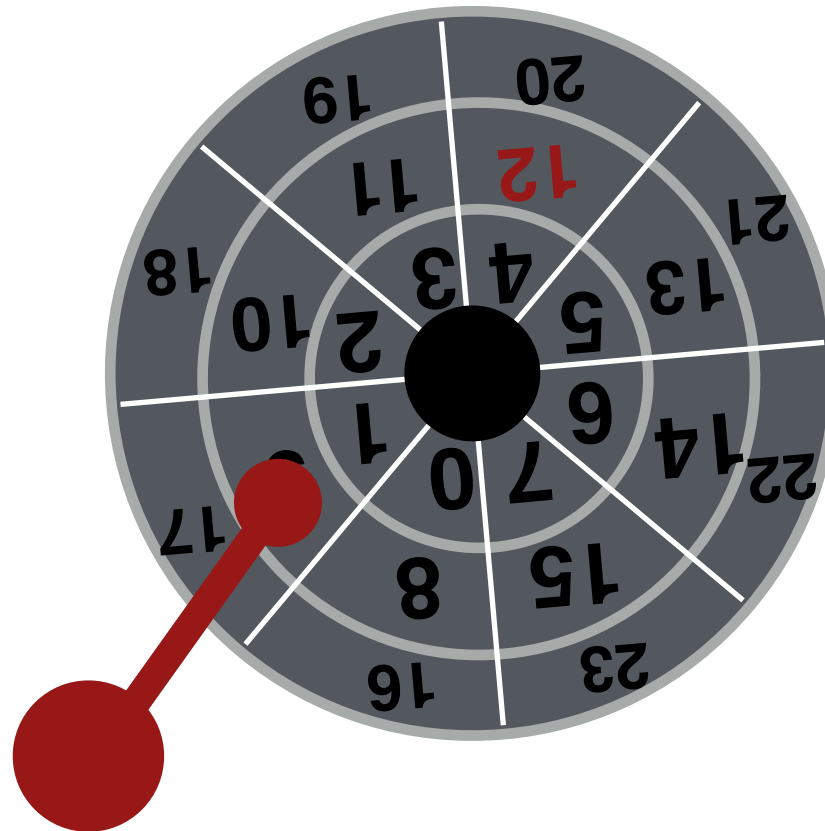
# Multiple Tracks: Wait for Rotation

Let's Read 12!



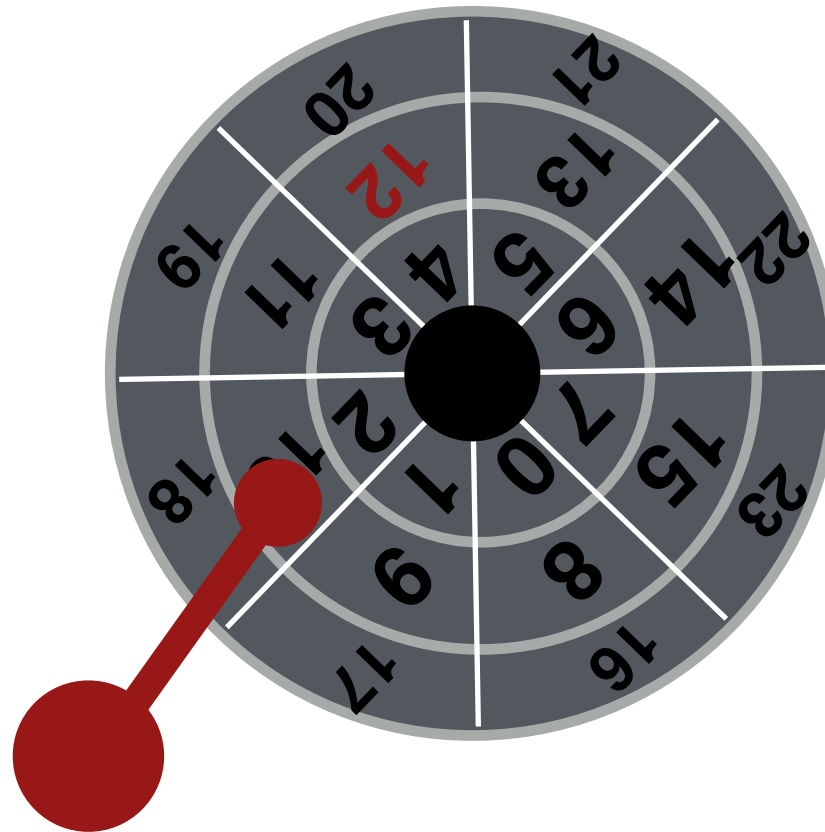
# Multiple Tracks: **Wait for Rotation**

Let's Read 12!



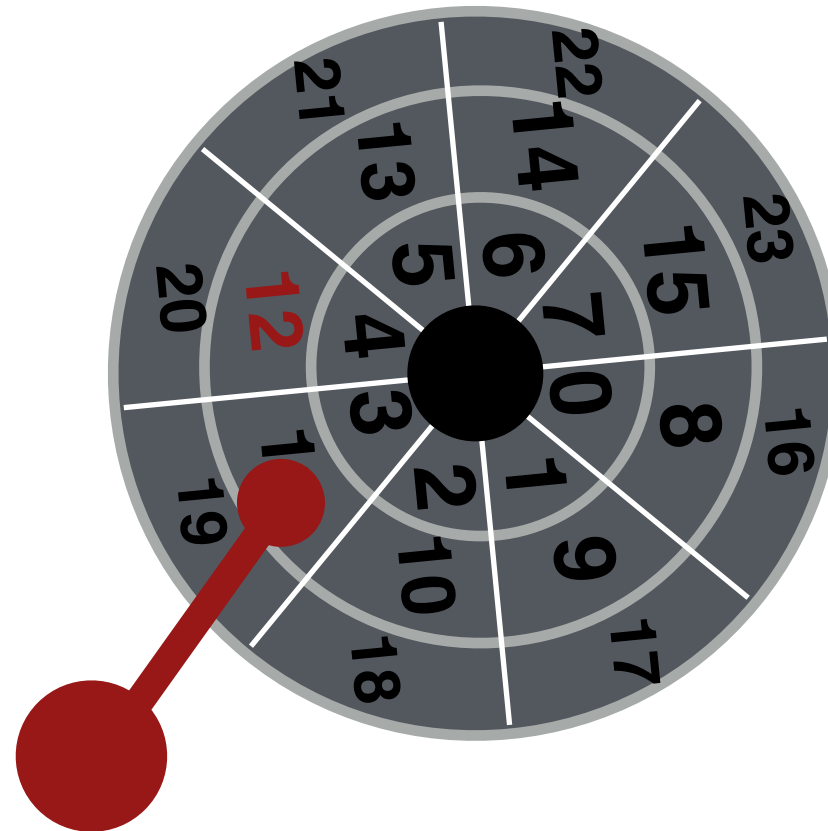
# Multiple Tracks: Wait for Rotation

Let's Read 12!



# Multiple Tracks: Wait for Rotation

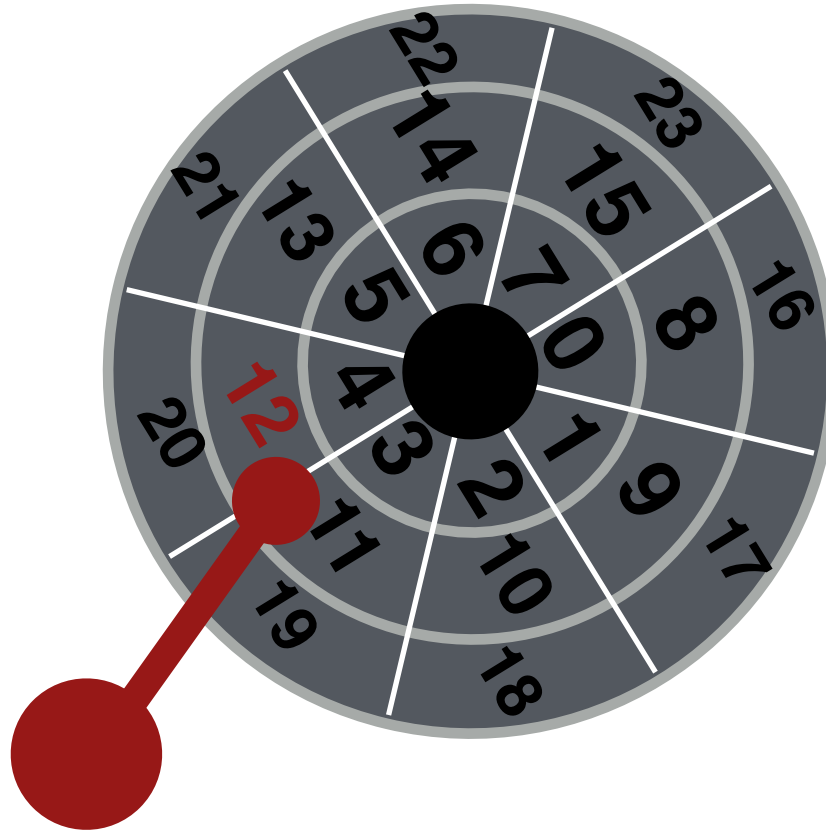
Let's Read 12!





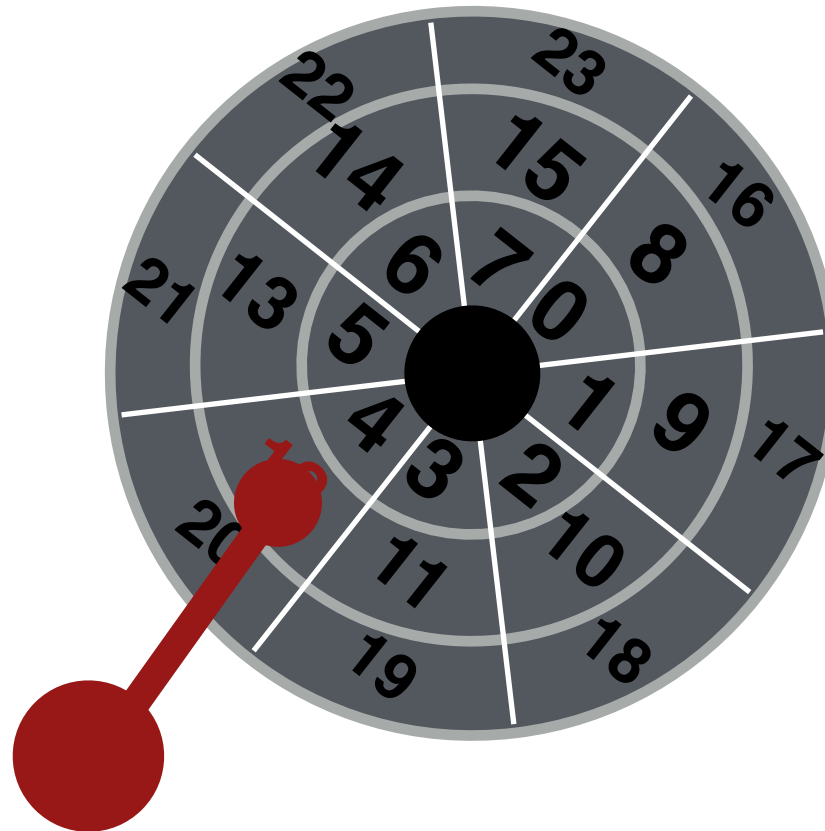
# Multiple Tracks: Transfer Data

Let's Read 12!



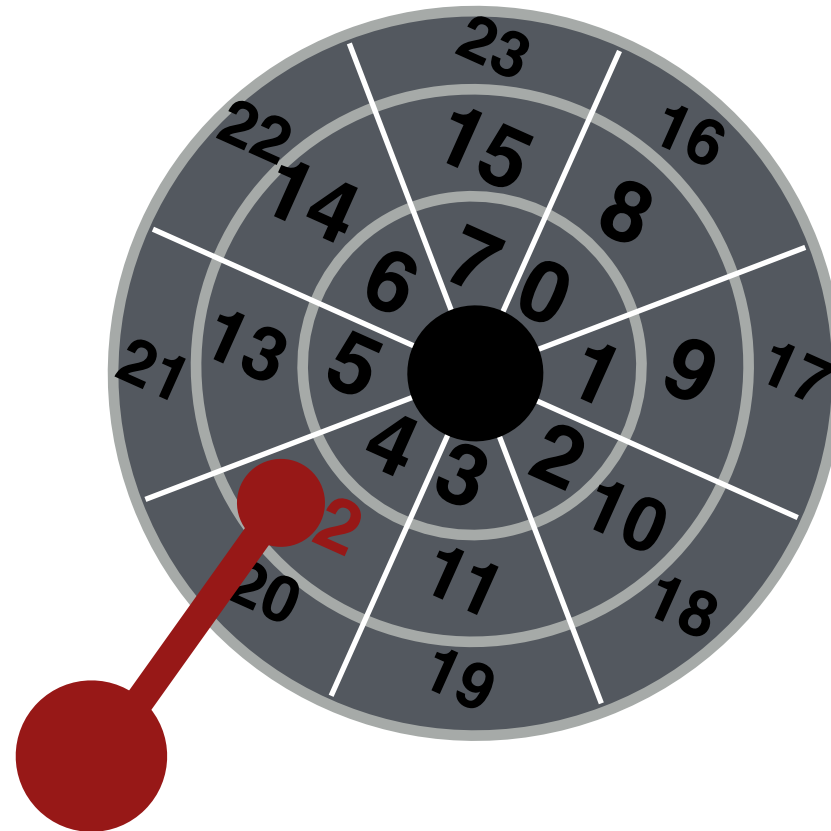
# Multiple Tracks: Transfer Data

Let's Read 12!

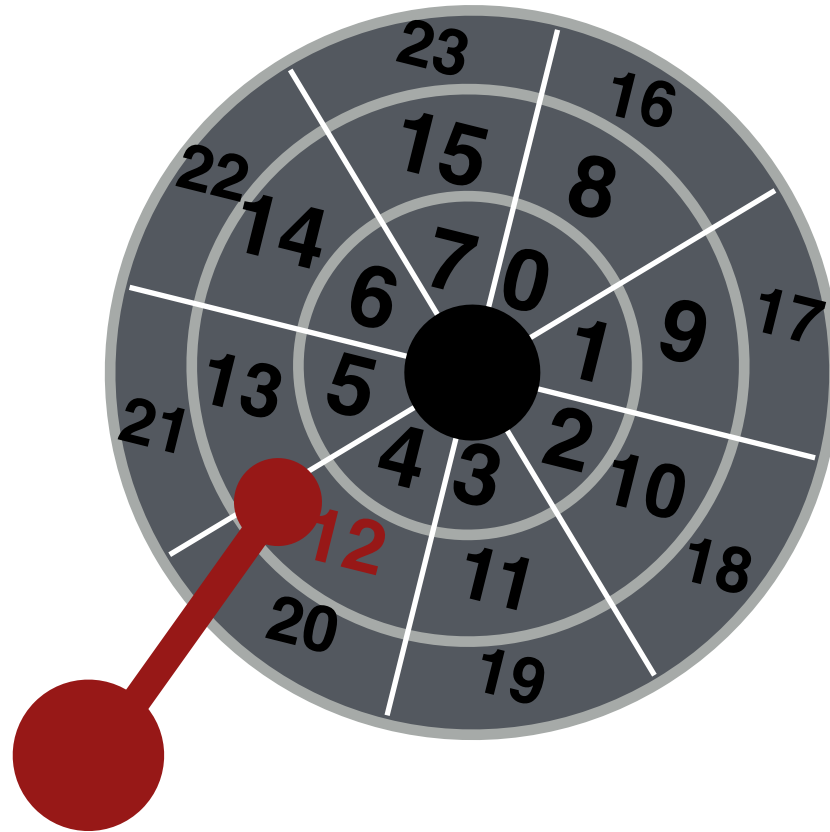


# Multiple Tracks: **Transfer Data**

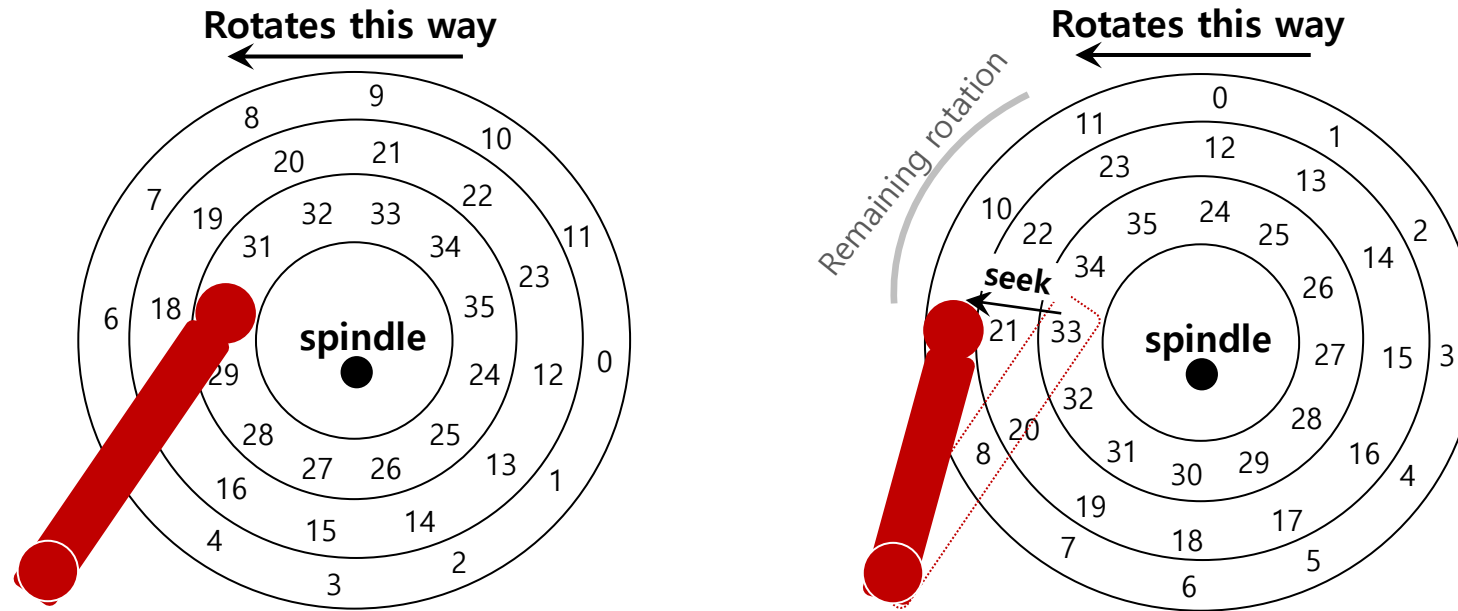
Let's Read 12!



# Yay!



# Multiple Tracks: Seek Time



## Seek: Move the disk arm to the correct track

- **Seek time:** Time to move head to the track contain the desired sector.
- One of the most costly disk operations.

# Seek, Rotate, Transfer

## Acceleration → Coasting → Deceleration → Settling

- **Acceleration:** The disk arm gets moving.
- **Coasting:** The arm is moving at full speed.
- **Deceleration:** The arm slows down.
- **Settling:** The head is *carefully positioned* over the correct track.

## Seeks often take several milliseconds!

- settling alone can take 0.5 to 2ms.
- entire seek often takes 4 - 10 ms.

# Seek, Rotate, Transfer

## Depends on rotations per minute (RPM)

- 7200 RPM is common, 15000 RPM is high-end.

## With 7200 RPM, how long to rotate around?

- $1 / 7200 \text{ RPM} = 1 \text{ minute} / 7200 \text{ rotations} = 1 \text{ second} / 120 \text{ rotations} = 8.3 \text{ ms} / \text{rotation}$

## Average rotation?

- $8.3 \text{ ms} / 2 = 4.15 \text{ ms}$

# Seek, Rotate, Transfer

## The final phase of I/O

- Data is either *read from* or *written* to the surface.

Pretty fast — depends on RPM and sector density

100+ MB/s is typical for maximum transfer rate

How long to transfer 512-bytes?

- 512 bytes \* (1s / 100 MB) = 5  $\mu$ s



# Workload

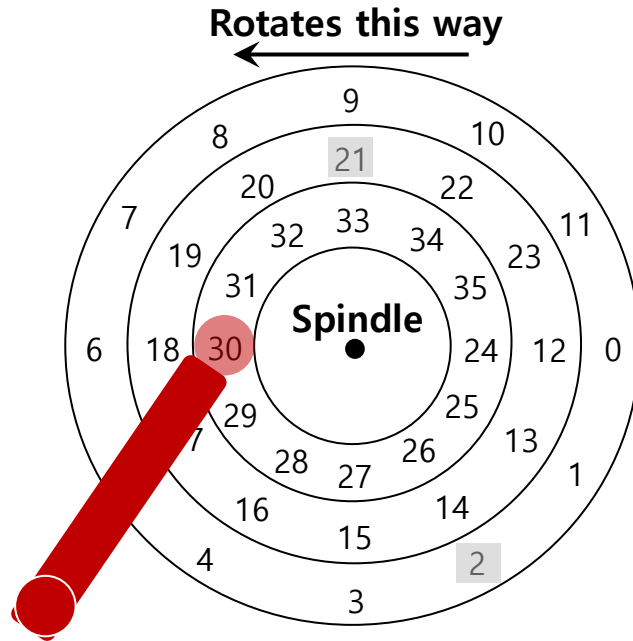
So...

- seeks are slow
- rotations are slow
- transfers are fast

**What kind of workload is fastest for disks?**

- **Sequential**: access sectors in order (transfer dominated)
- **Random**: access sectors arbitrarily (seek+rotation dominated)

# Disk Scheduling



**Disk Scheduler decides which I/O request to schedule next**

# Disk Scheduling: FCFS

## “First Come First Served”

- Process disk requests in the order they are received

## Advantages

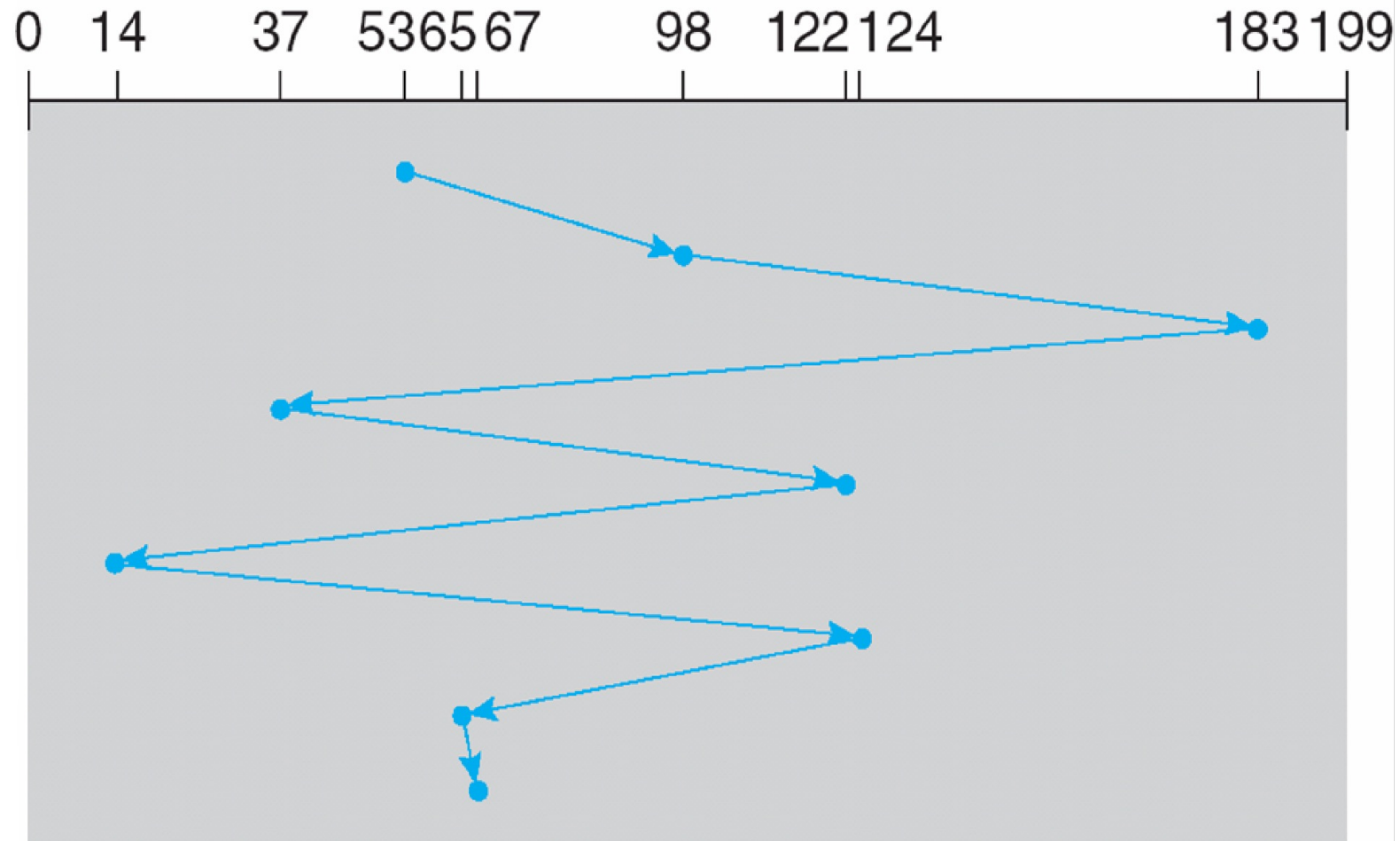
- Easy to implement
- Good fairness

## Disadvantages

- Cannot exploit request locality
- Increases average latency, decreasing throughput

# FCFS Example

queue = 98, 183, 37, 122, 14, 124, 65, 67  
head starts at 53



# SSTF (Shortest Seek Time First)

Order the queue of I/O request by track

Pick requests on the nearest track to complete first

- Also called shortest positioning time first (SPTF)

## Advantages

- Exploits locality of disk requests
- Higher throughput

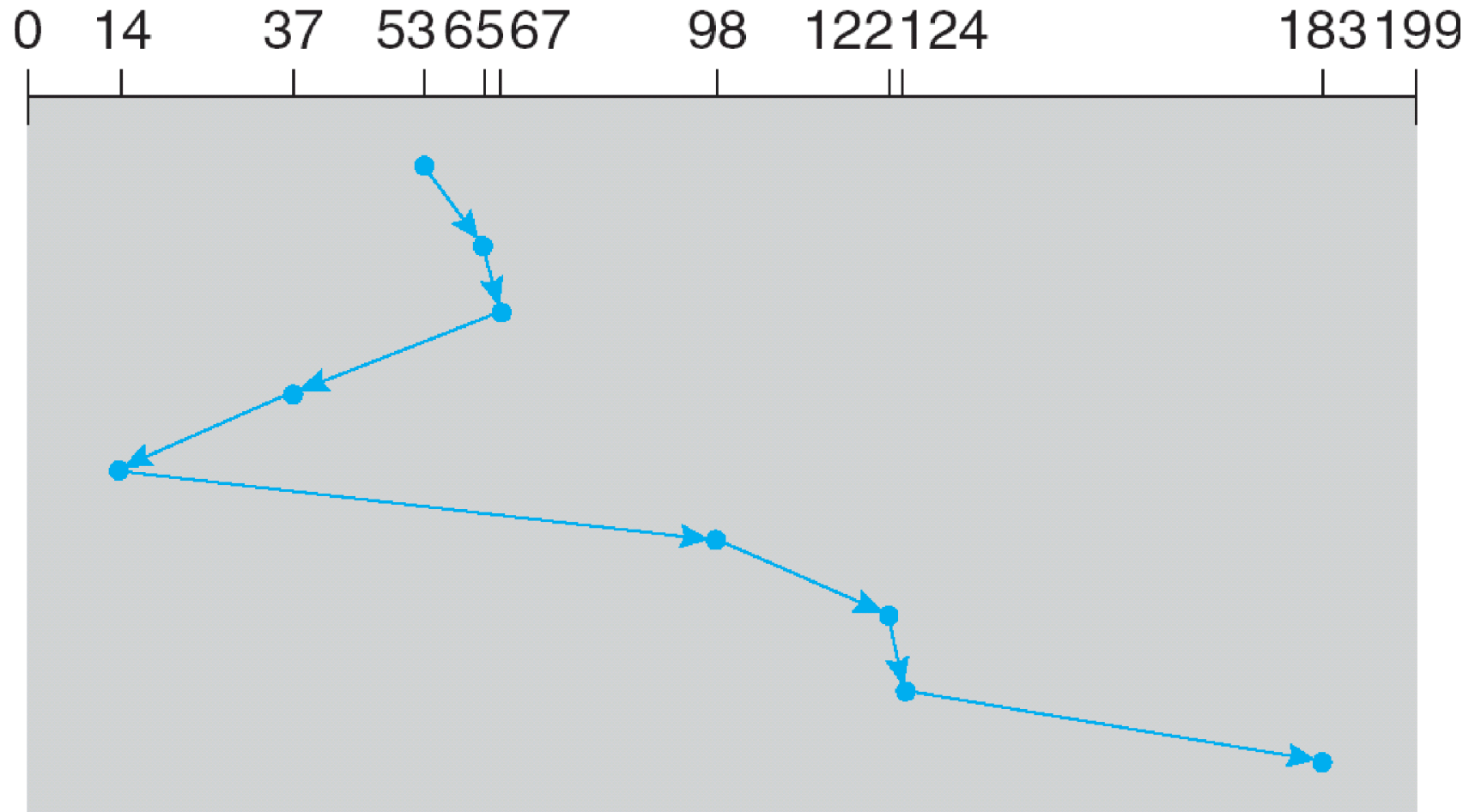
## Disadvantages

- Starvation
- Don't always know what request will be fastest

# SSTF Example

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

head starts at 53



# “Elevator” Scheduling (SCAN)

## Sweep across disk, servicing all requests passed

- Like SSTF, but next seek must be in same direction
- Switch directions only if no further requests

## Advantages

- Takes advantage of locality
- Bounded waiting

## Disadvantages

- Cylinders in the middle get better service
- Might miss locality SSTF could exploit

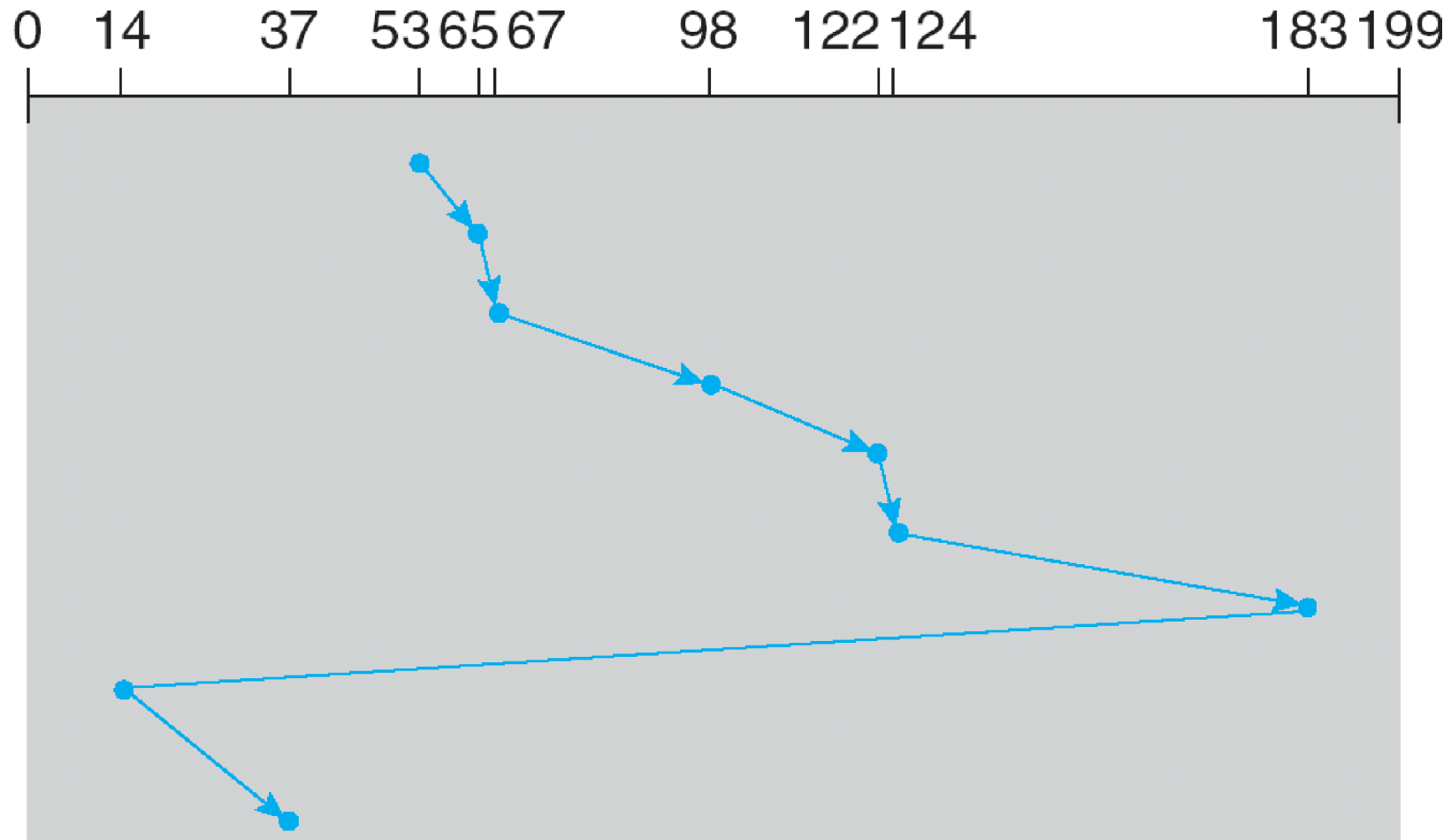
## CSCAN: Only sweep in one direction

- **Very commonly used algorithm in Unix**

# CSCAN example

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

head starts at 53





# Flash Memory

Today, people increasingly using flash memory

## Completely solid state (no moving parts)

- Remembers data by storing charge
- Lower power consumption and heat
- No mechanical seek times to worry about

## Limited # overwrites possible

- Blocks wear out after 10,000 (MLC) – 100,000 (SLC) erases
- Requires flash translation layer (FTL) to provide wear leveling, so repeated writes to logical block don't wear out physical block
- FTL can seriously impact performance

## Limited durability

- Charge wears out over time
- Turn off device for a year, you can potentially lose data!

# Next Time...

**Read Chapter 39, 40**