PERSISTENCE: I/O DEVICES

Shivaram Venkataraman

CS 537, Spring 2023

ADMINISTRIVIA

Project 4: Grades today (hopefully?)

Project 5: How is it going?

Midterm 2

- Date: April 4th, 2023
- Venue : Social Sciences 6210
- Time : 5.45pm to 7.15pm
- Practice exams: Check Canvas (Files → Old Exams)

AGENDA / LEARNING OUTCOMES

How does the OS interact with I/O devices?

What are the components of a hard disk drive?

RECAP

OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

Make each application believe it has each resource to itself CPU and Memory

1. Virtualization

2. Concurrency

Provide mutual exclusion, ordering

3. Persistence

MOTIVATION

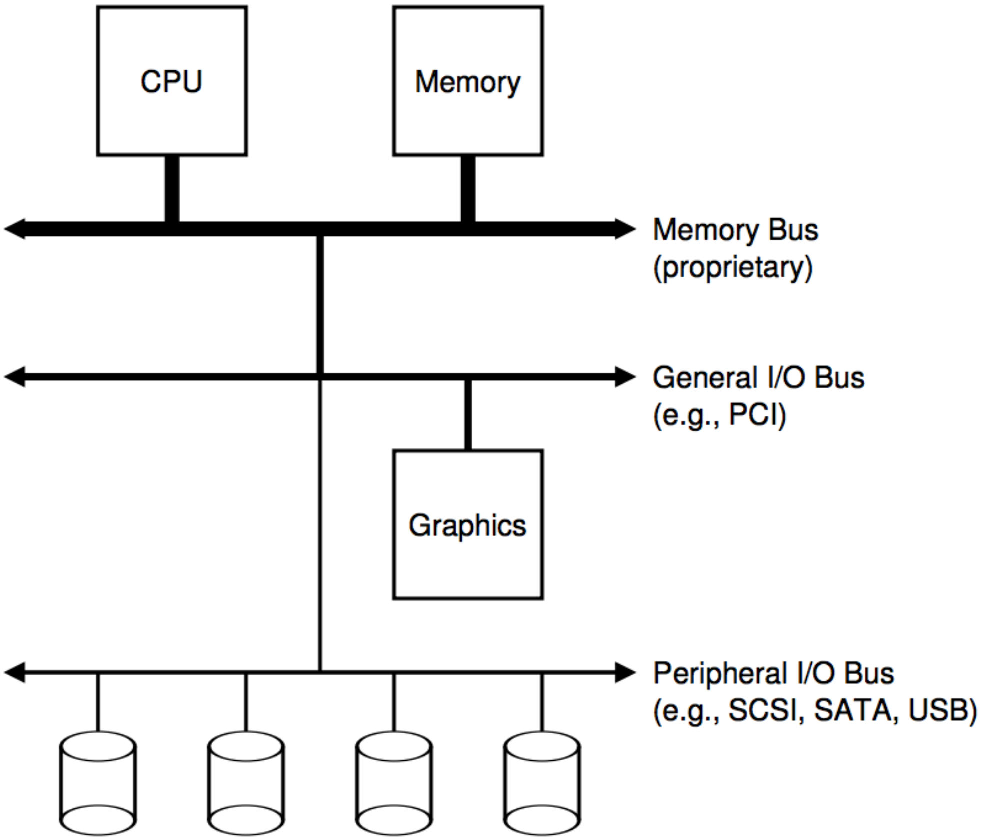
What good is a computer without any I/O devices?

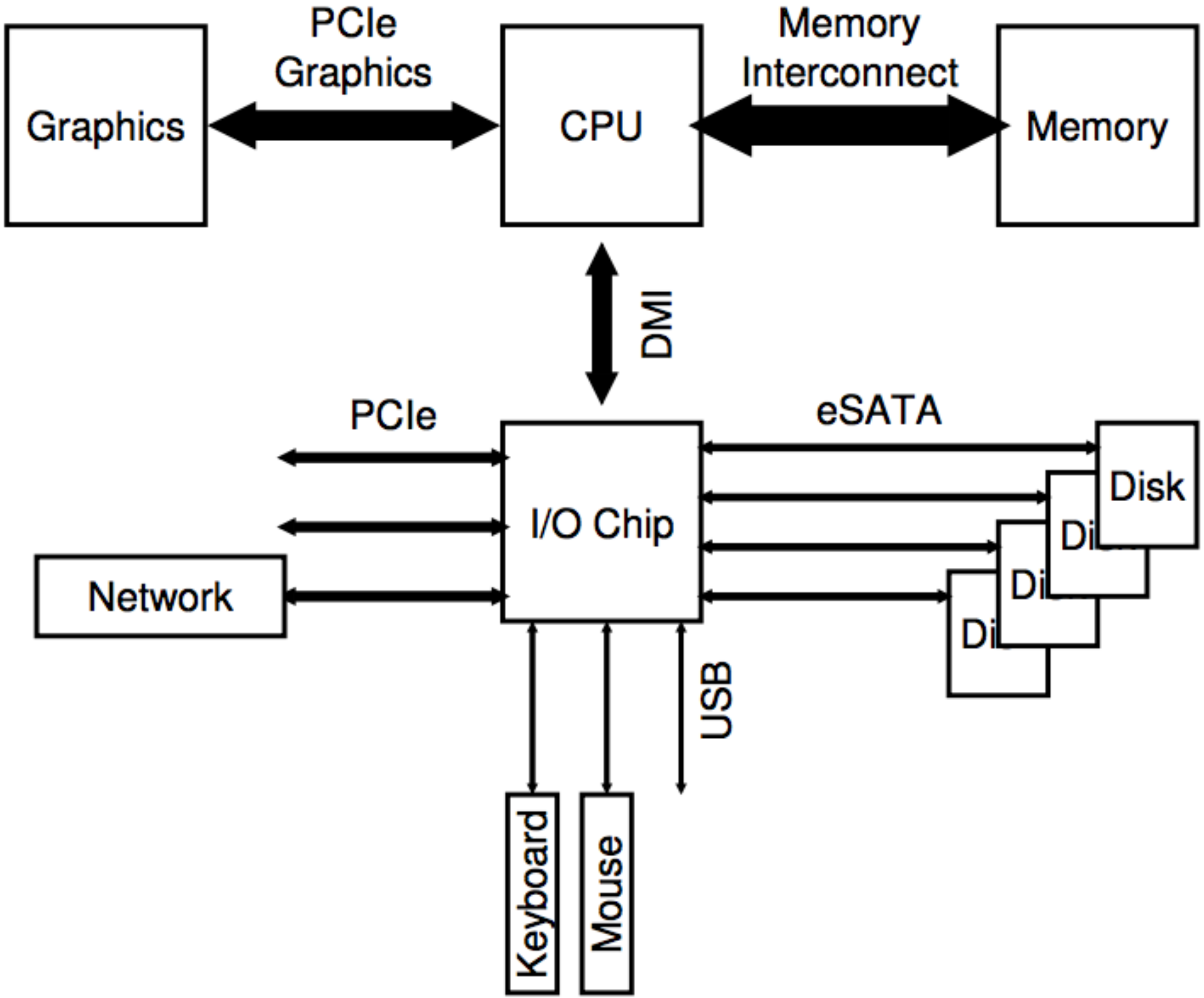
keyboard, display, disks

We want:

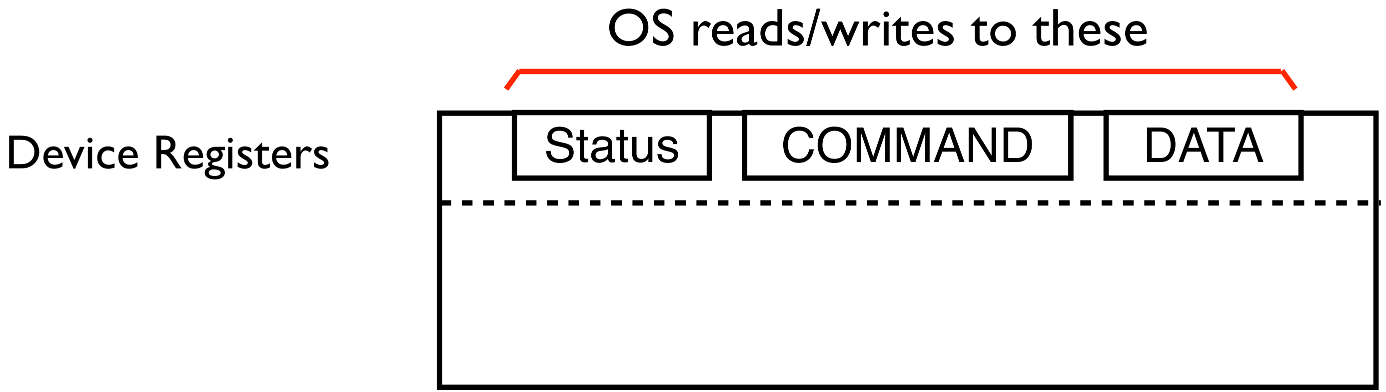
- **H/W** that will let us plug in different devices
- **OS** that can interact with different combinations

HARDWARE SUPPORT FOR I/O




PCIe
Graphics
Memory
Interconnect
Graphics
CPU
Memory
DMI
PCIe
eSATA
I/O Chip
Disk
Network
USB
Keyboard
Mouse

CANONICAL DEVICE

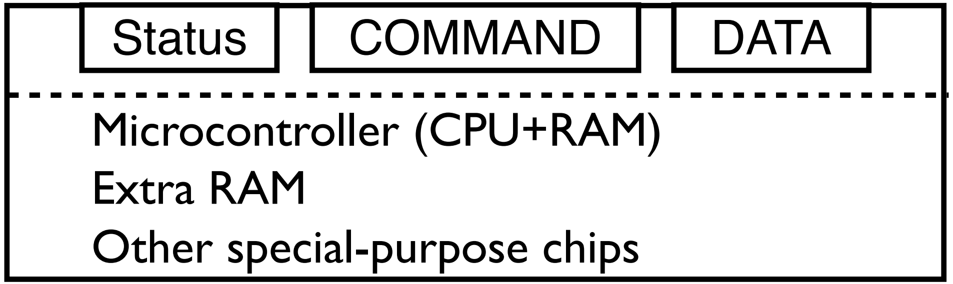


Status checks: polling *vs.* interrupts

Data transfer

Control: Invoking I/O

EXAMPLE WRITE PROTOCOL



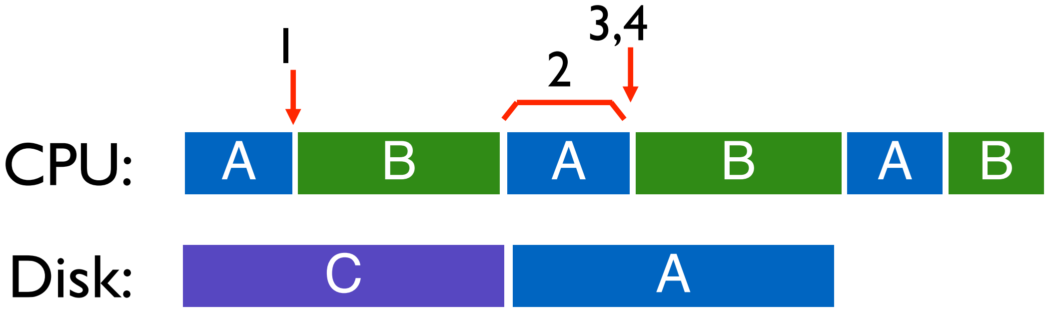

```
while (STATUS == BUSY)
  ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
  ; // spin
```

CPU:

Disk:

```
while (STATUS == BUSY)                   // 1
    ;
Write data to DATA register              // 2
Write command to COMMAND register        // 3
while (STATUS == BUSY)                   // 4
    ;
```

Interrupts!




```
while (STATUS == BUSY)              // 1
      wait for interrupt;
Write data to DATA register         // 2
Write command to COMMAND register   // 3
while (STATUS == BUSY)              // 4
      wait for interrupt;
```

INTERRUPTS VS. POLLING

Are interrupts always better than polling?

Fast device: Better to spin than take interrupt overhead

- Device time unknown? Hybrid approach (spin then use interrupts)

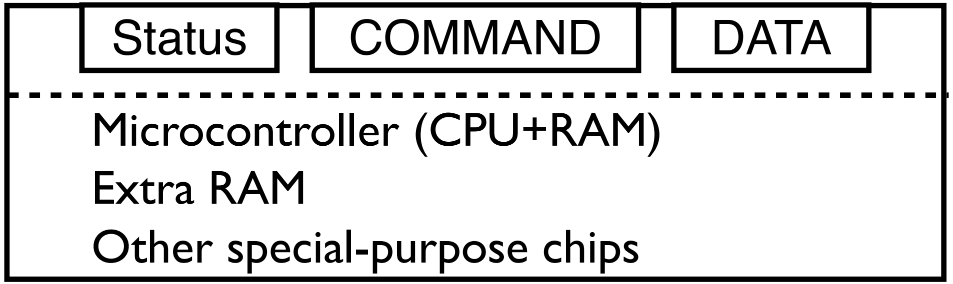
Flood of interrupts arrive

- Can lead to livelock (always handling interrupts)
- Better to ignore interrupts while make some progress handling them

Other improvement

- Interrupt coalescing (batch together several interrupts)

PROTOCOL VARIANTS

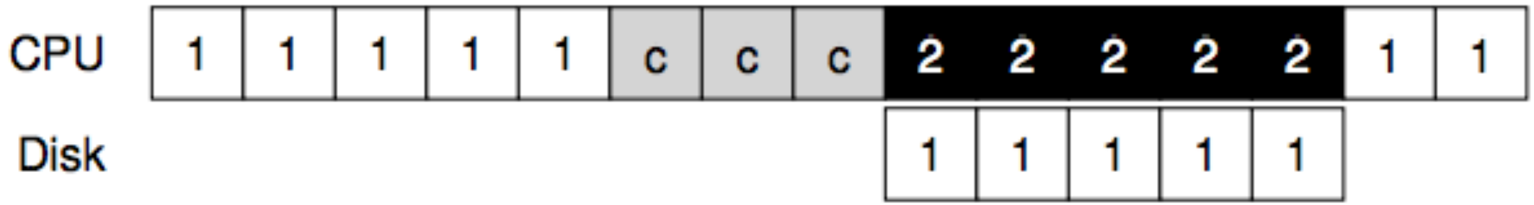


~~Status checks: polling vs. interrupts~~

Data transfer

Control: Invoking I/O

DATA TRANSFER COSTS



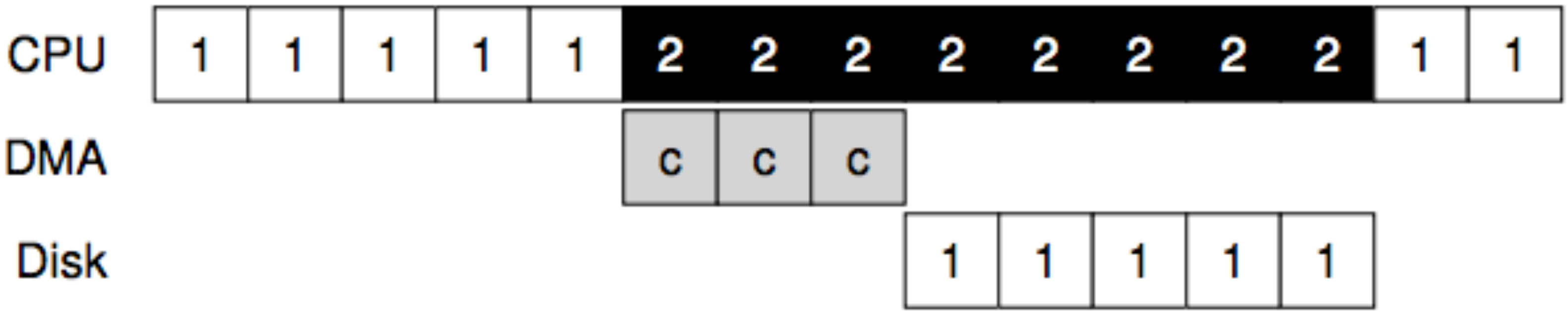
PROGRAMMED I/O VS. DIRECT MEMORY ACCESS

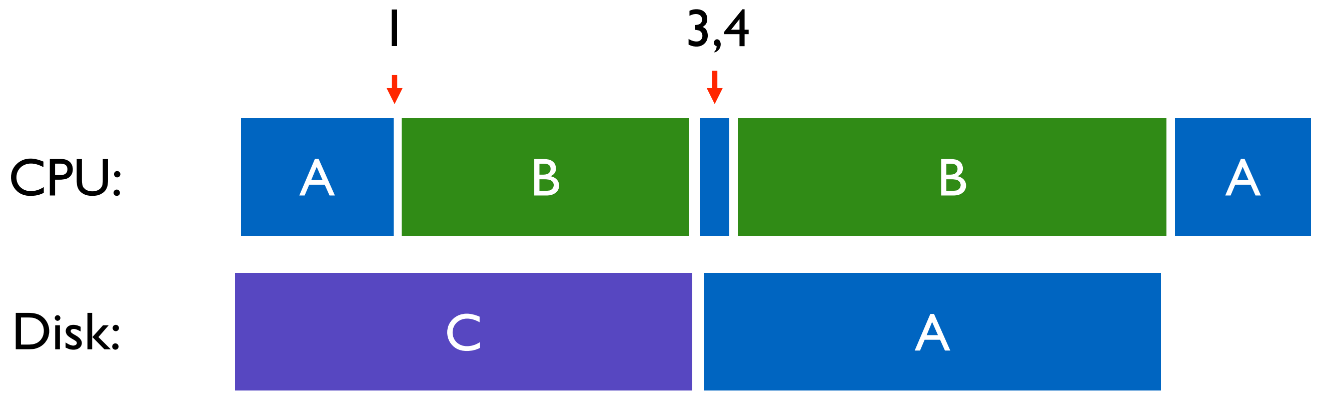
PIO (Programmed I/O):

- CPU directly tells device what the data is

DMA (Direct Memory Access):

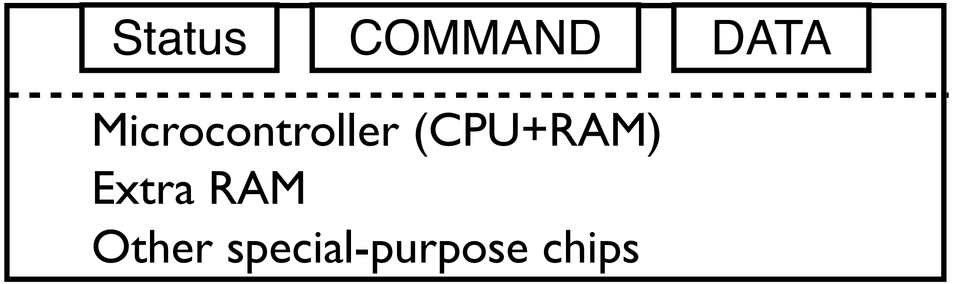
- CPU leaves data in memory
- Device reads data directly from memory


CPU
1 1 1 1 1 2 2 2 2 2 2 2 2 1 1
DMA
c c c
Disk
1 1 1 1 1




```
while (STATUS == BUSY)               // 1
  ;
Write data to DATA register          // 2
Write command to COMMAND register    // 3
while (STATUS == BUSY)               // 4
  ;
```

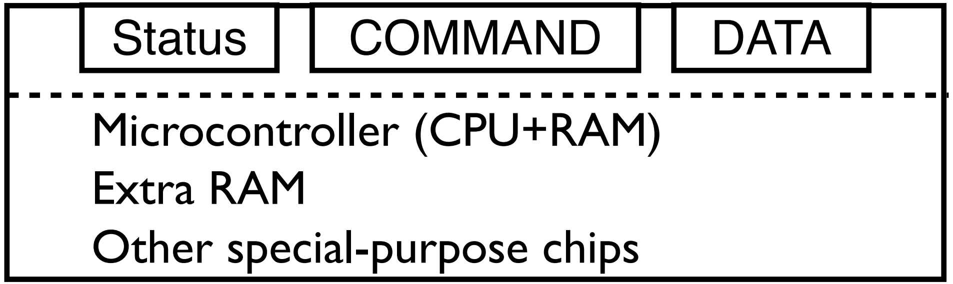
PROTOCOL VARIANTS



Status checks: polling *vs.* interrupts

PIO vs DMA

Control: Invoking I/O




```
while (STATUS == BUSY)              // 1
  ;
Write data to DATA register         // 2
Write command to COMMAND register   // 3
while (STATUS == BUSY)              // 4
  ;
```

SPECIAL INSTRUCTIONS VS. MEM-MAPPED I/O

Special instructions

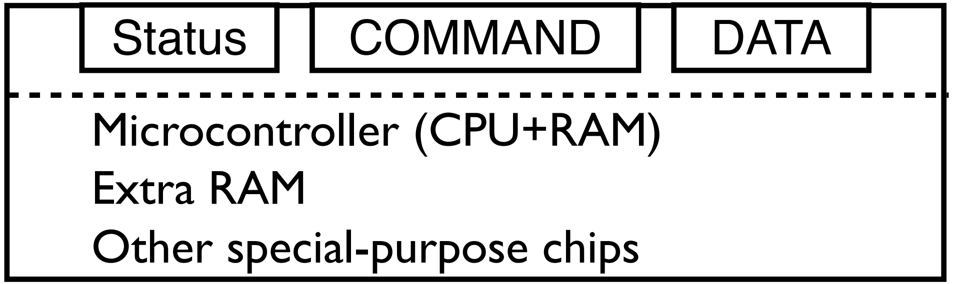
- each device has a port
- in/out instructions (x86) communicate with device

Memory-Mapped I/O

- H/W maps registers into address space
- loads/stores sent to device

Doesn’t matter much (both are used)

PROTOCOL VARIANTS

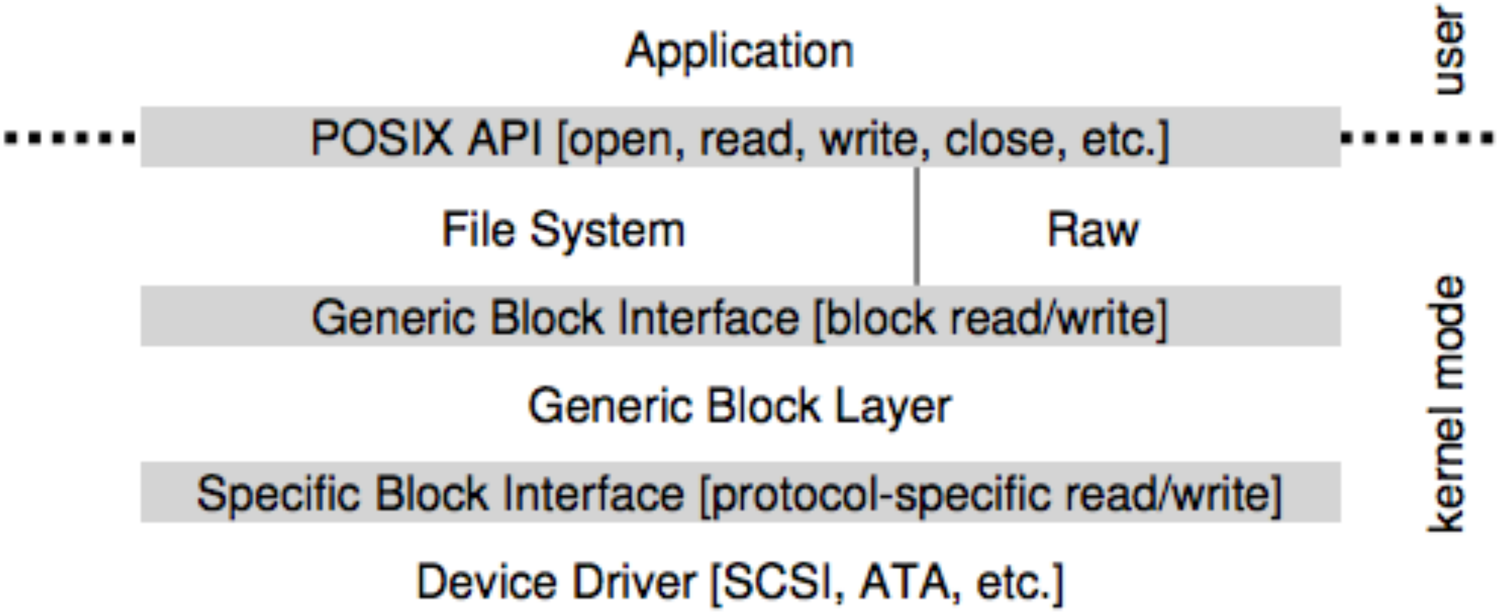


Status checks: polling *vs.* interrupts

PIO vs DMA

Special instructions vs. Memory mapped I/O

DEVICE DRIVERS



VARIETY IS A CHALLENGE

Problem:

- many, many devices
- each has its own protocol

How can we avoid writing a slightly different OS for each H/W combination?

Write device driver for each device

Drivers are **70%** of Linux source code

QUIZ 20

https://tinyurl.com/cs537-sp23-quiz20

If you have a fast non-volatile memory based storage device, which approach would work better?

What part of a device protocol is improved by using DMA ?

HARD DISKS

HARD DISK INTERFACE

Disk has a sector-addressable address space

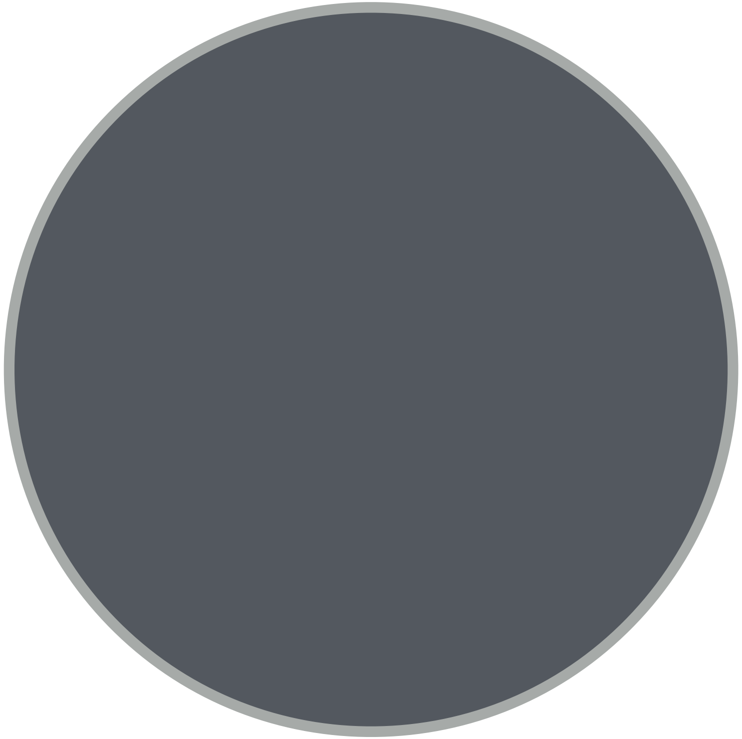
Appears as an array of sectors

Sectors are typically 512 bytes

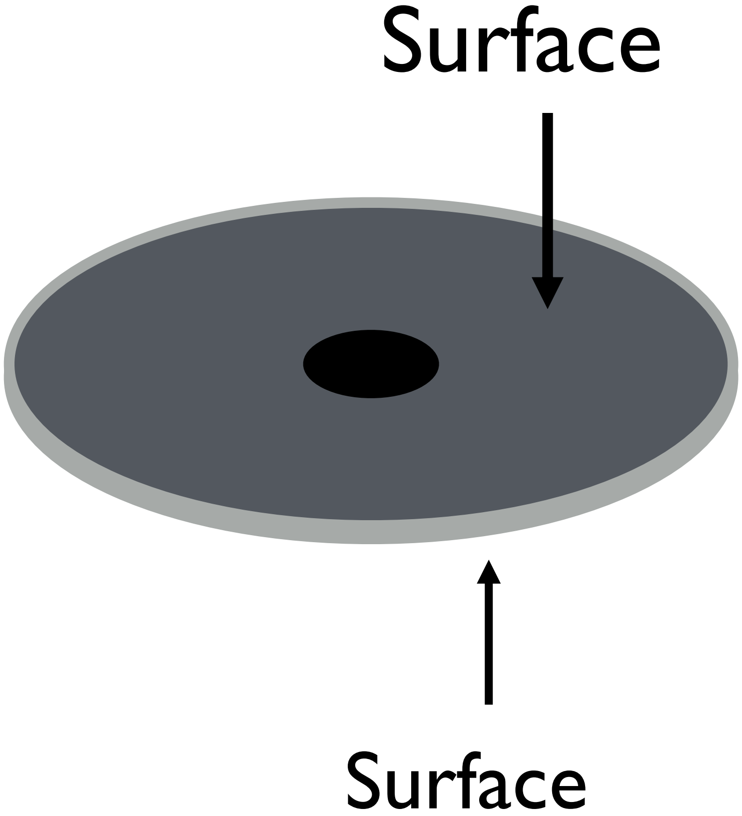
Main operations: reads + writes to sectors

Mechanical and slow (?)

Platter

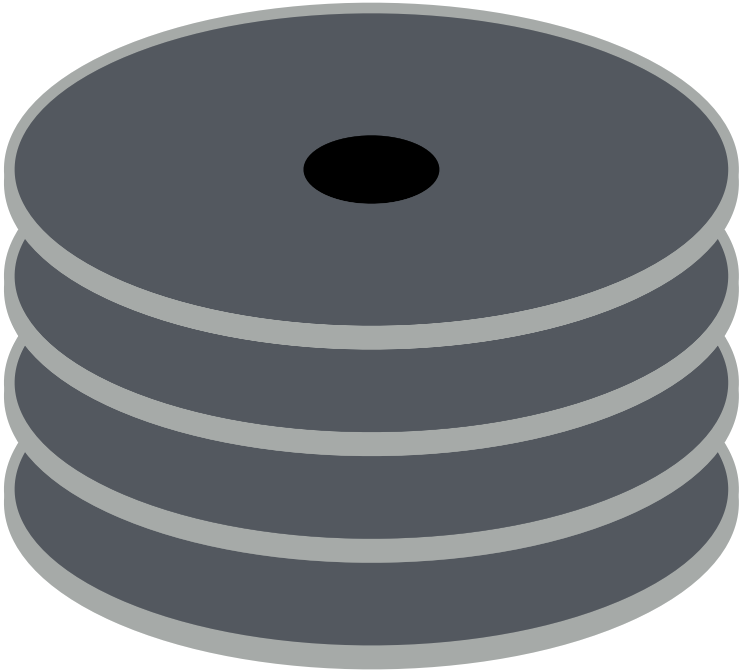


Spindle



RPM?

Motor connected to spindle spins platters

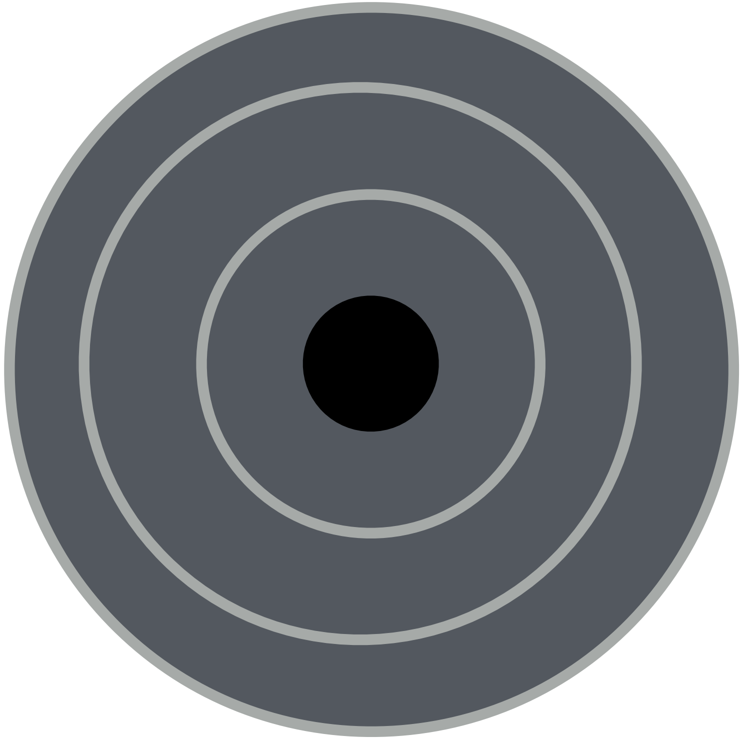


Rate of rotation: RPM

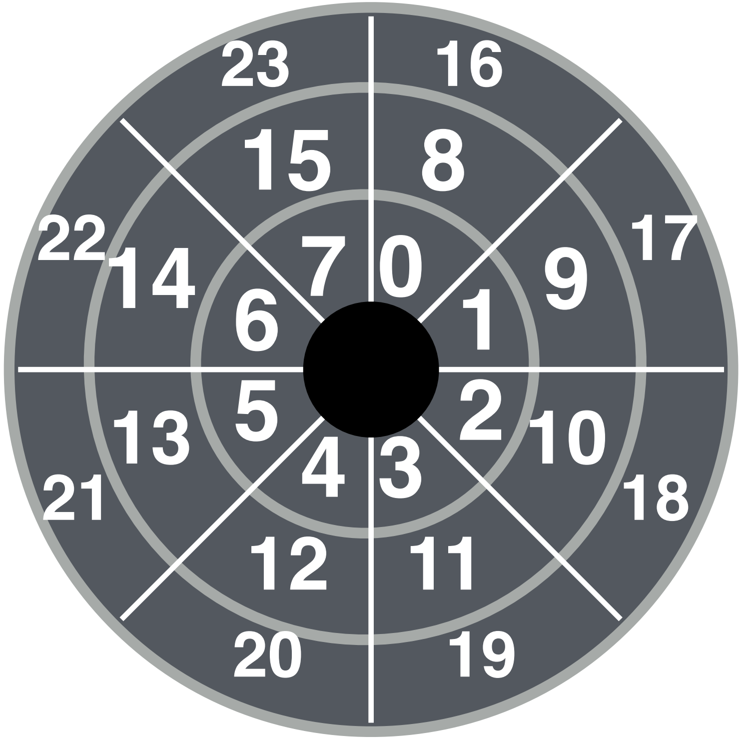
10000 RPM → single rotation is 6 ms

Surface is divided into rings: tracks

Stack of tracks(across platters): cylinder

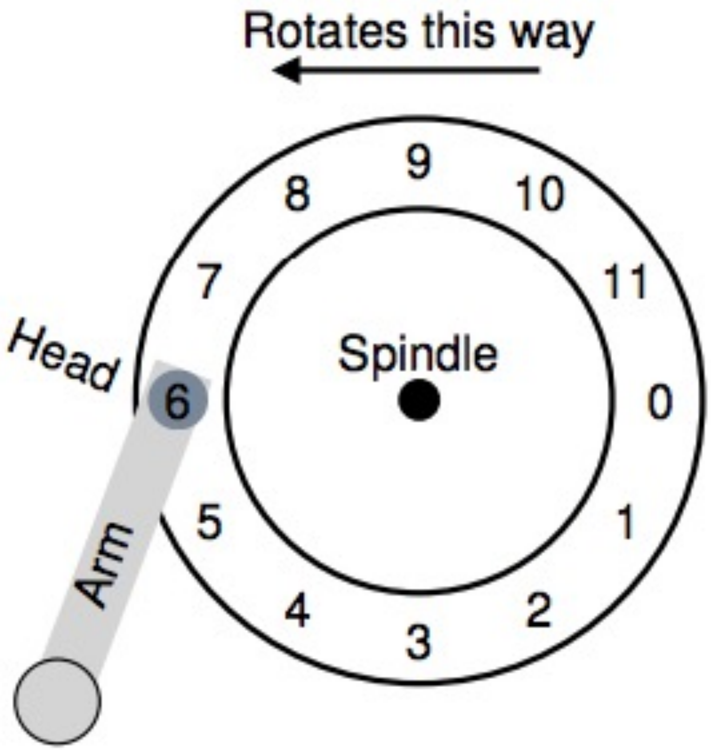


Tracks are divided into numbered sectors



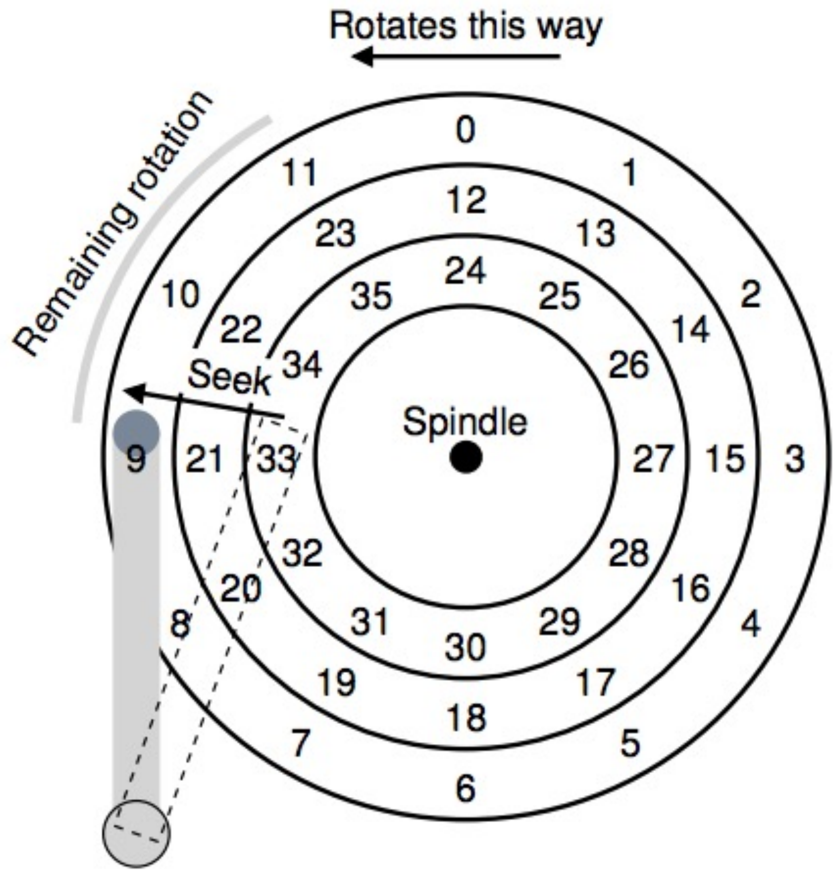
Heads on a moving arm can read from each surface.

READING DATA FROM DISK



Rotational delay

READING DATA FROM DISK



Seek Time

TIME TO READ/WRITE

Three components:

Time = seek + rotation + transfer time

SEEK, ROTATE, TRANSFER

Seek cost: Function of cylinder distance

- Not purely linear cost
- Must accelerate, coast, decelerate, settle
- Settling alone can take 0.5 - 2 ms

Entire seeks often takes 4 - 10 ms

Average seek = 1/3 of max seek

Depends on rotations per minute (RPM)

- 7200 RPM is common, 15000 RPM is high end

Average rotation?

Pretty fast: depends on RPM and sector density.

100+ MB/s is typical for maximum transfer rate

QUIZ 21

https://tinyurl.com/cs537-sp23-quiz21

What is the time for 4KB random read?

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

NEXT STEPS

Advanced disk features

Scheduling disk requests

Midterm 2 soon!