PERSISTENCE: I/O DEVICES

Shivaram Venkataraman

CS 537, Spring 2020

CHECK: 1,2,3

1. Can you hear me?
2. Can you see the slides and the annotations?
3. Can you write a question on the chat?

My notes look like this.

Videos will be available

ADMINISTRIVIA

Discussion: Sync
Async → Video posted → OH

Project 4a: Out today! Due April 2nd, 10pm → Individually or Groups of two

Grades: Project 3 grades out! → Check Piazza

Midterm grades end of the week (hopefully) → PDF with grades

Attendance: 24 hours after end of the class
2:15pm on Wed (Tue class)
Fri (Thu class)

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

Process

Address space

Mechanisms

Policies

1. Virtualization

2. Concurrency

Provide mutual exclusion, ordering

Locks, CVs

3. Persistence

↳ Devices
Store data

MOTIVATION

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

keyboard, display, disks

outputs (inputs)

mouse, Pen!

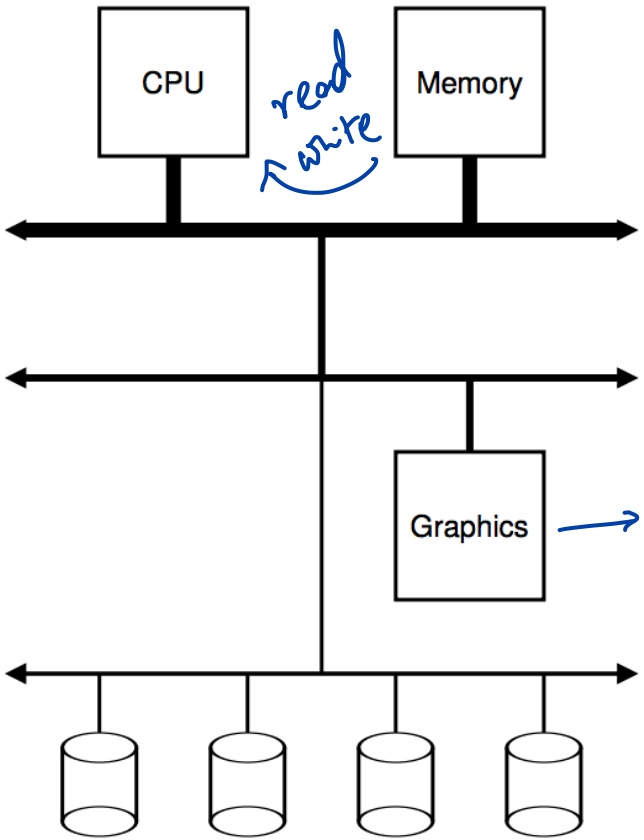
Output

We want:

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

hardware

HARDWARE SUPPORT FOR I/O



4 core
60 GB/sec

PCI bus
8-10 GB/s

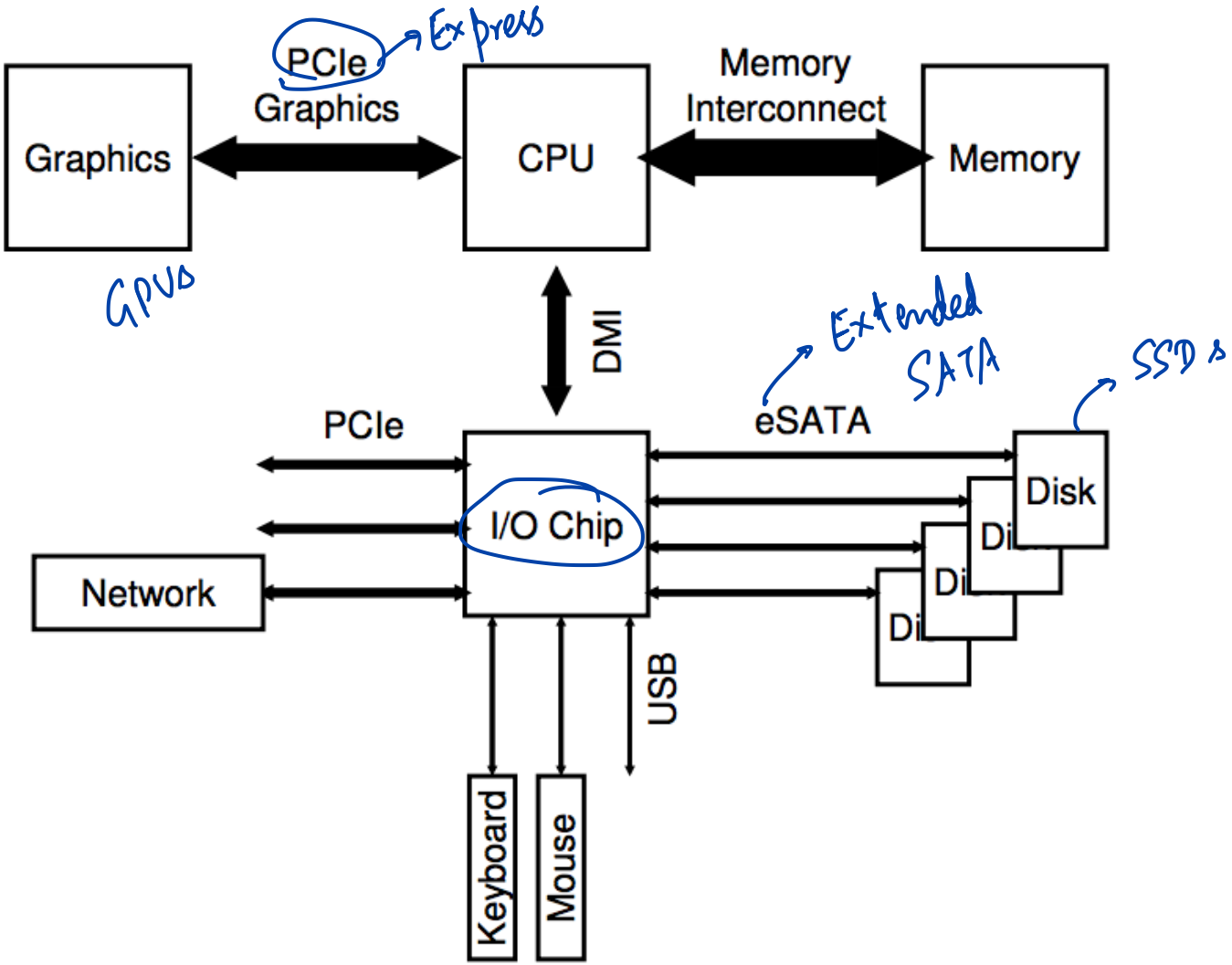
100-500 MB/s

Farther

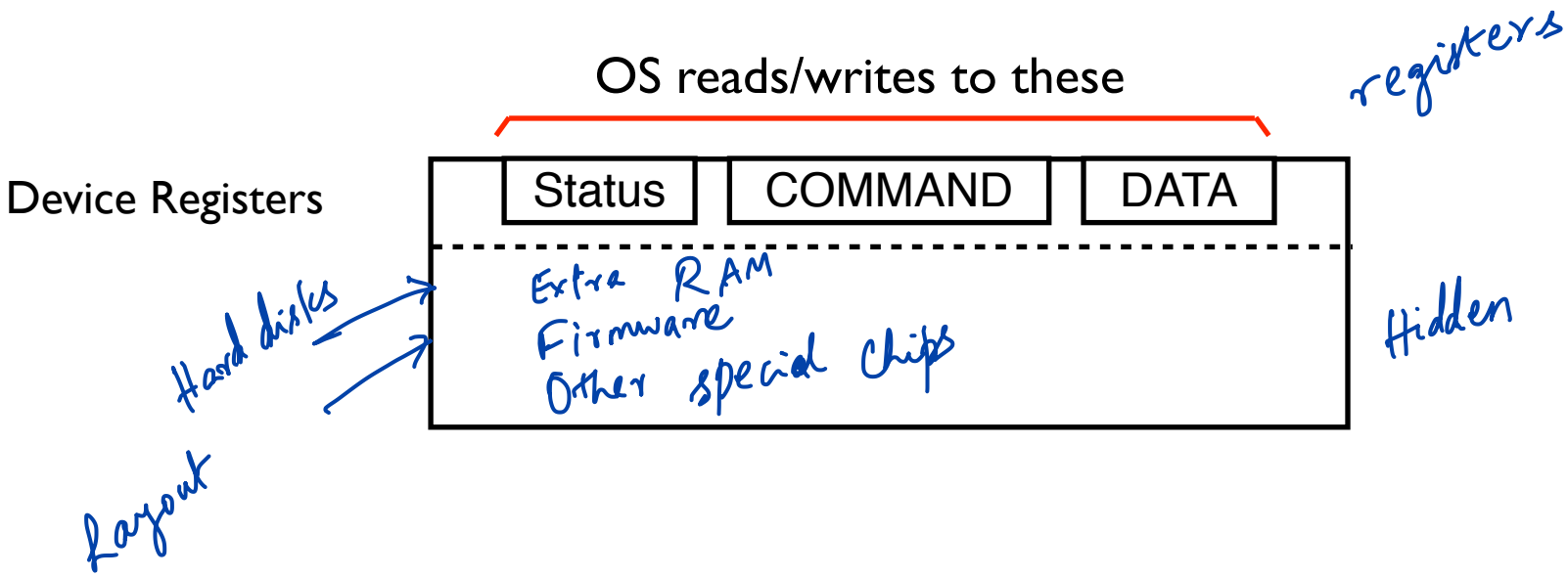
Hierarchy of devices connected

slower

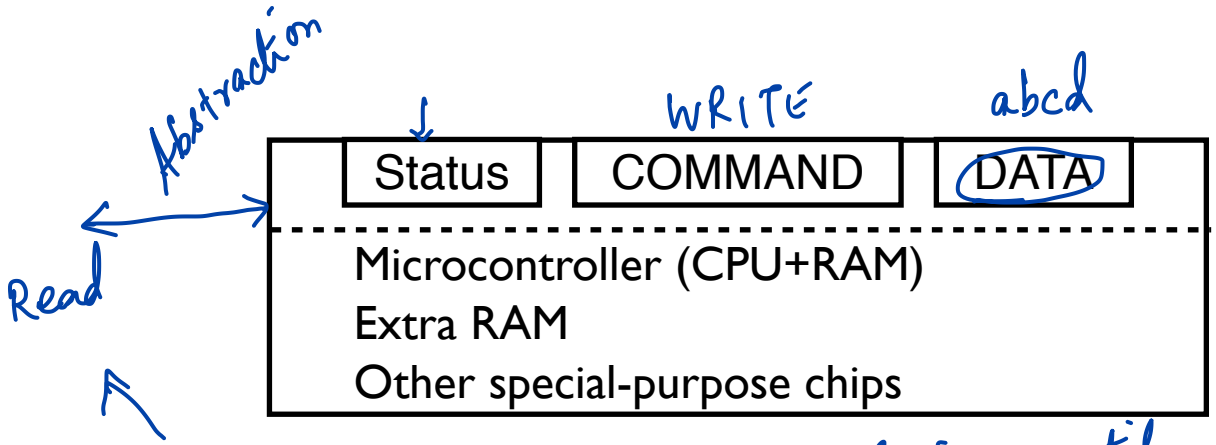
Modern computer



CANONICAL DEVICE



EXAMPLE WRITE PROTOCOL



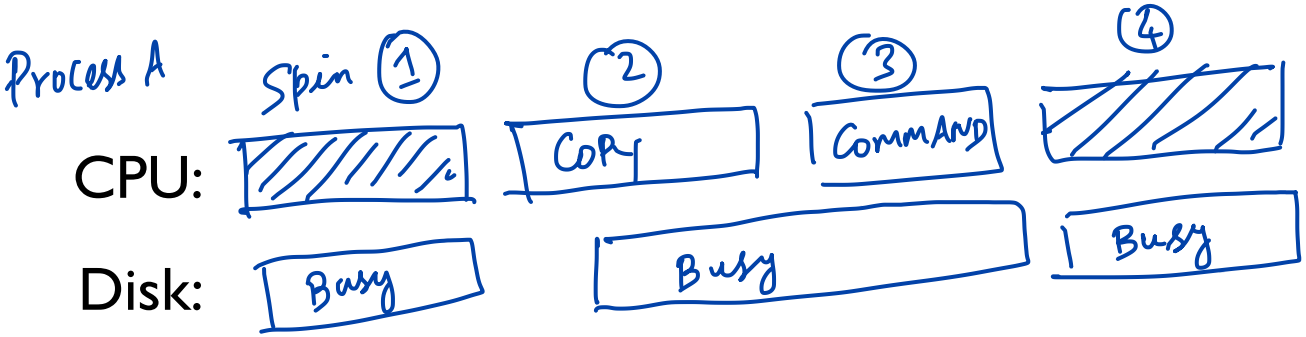

```
while (STATUS == BUSY)
   ; // spin
Write "data" to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
   ; // spin
```

Spin until device is free

Store or special instruction

WRITE

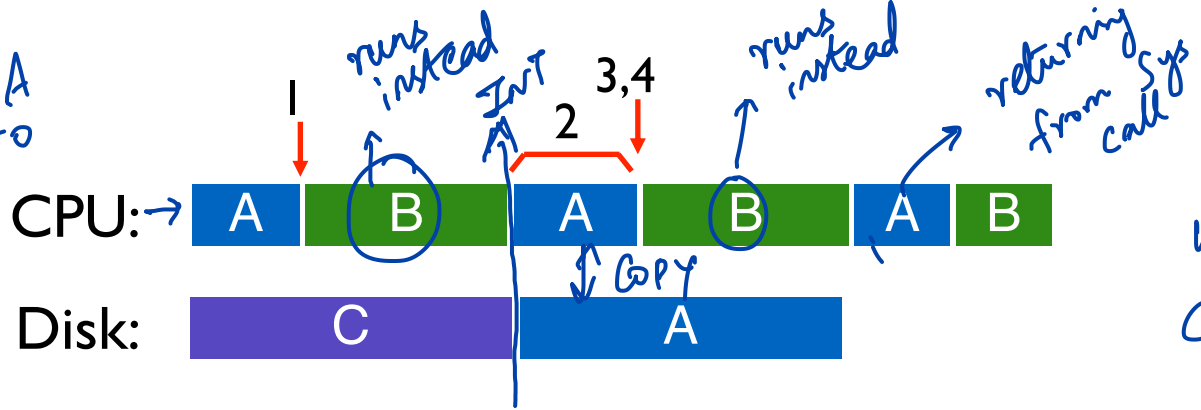
Spin until device is done with write



1 Shaded 4 regions CPU is wasteful as no useful work is done.

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

Process A wants to write to disk



Interrupts!

We avoid wasting CPU cycles on spin

Polling

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

INTERRUPTS VS. POLLING

Are interrupts always better than polling?

→ 1 μs

Fast device: Better to spin than take interrupt overhead

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

→ 100 μs then go to sleep wait interrupts

Flood of interrupts arrive

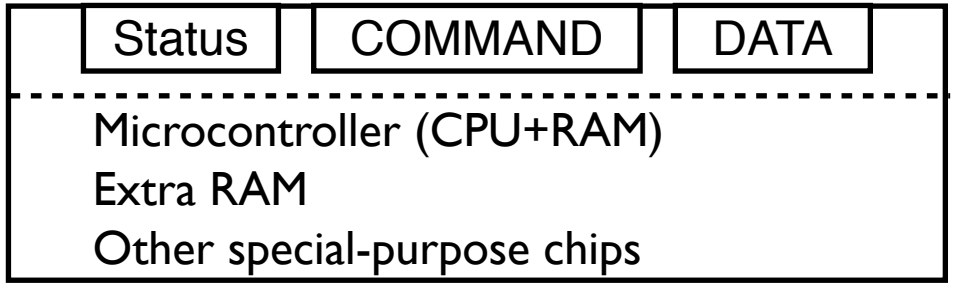
- Can lead to livelock (always handling interrupts)
- Better to ignore interrupts while make some progress handling them

Other improvement

- Interrupt coalescing (batch together several interrupts)

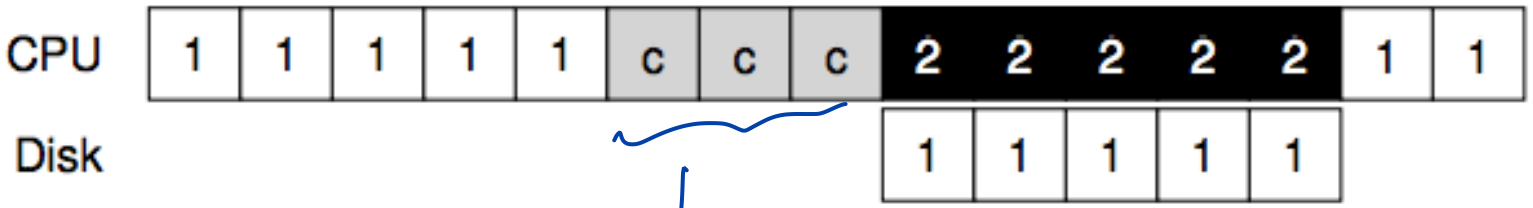
context switch

PROTOCOL VARIANTS



- ✓ Status checks: polling *vs.* interrupts

DATA TRANSFER COSTS



COPY Data

CPU is again not doing useful work

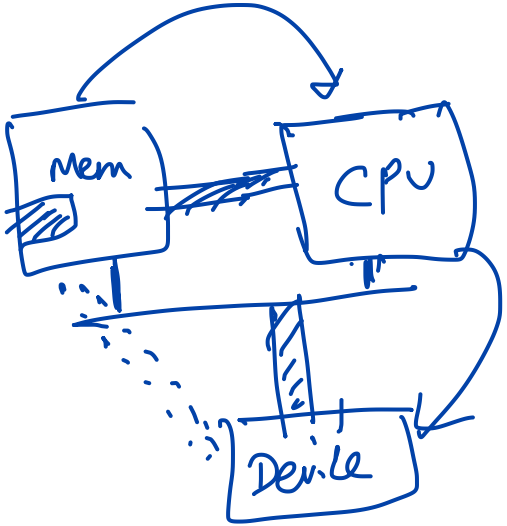
PROGRAMMED I/O VS. DIRECT MEMORY ACCESS

PIO (Programmed I/O):

- CPU directly tells device what the data is → Existing protocol

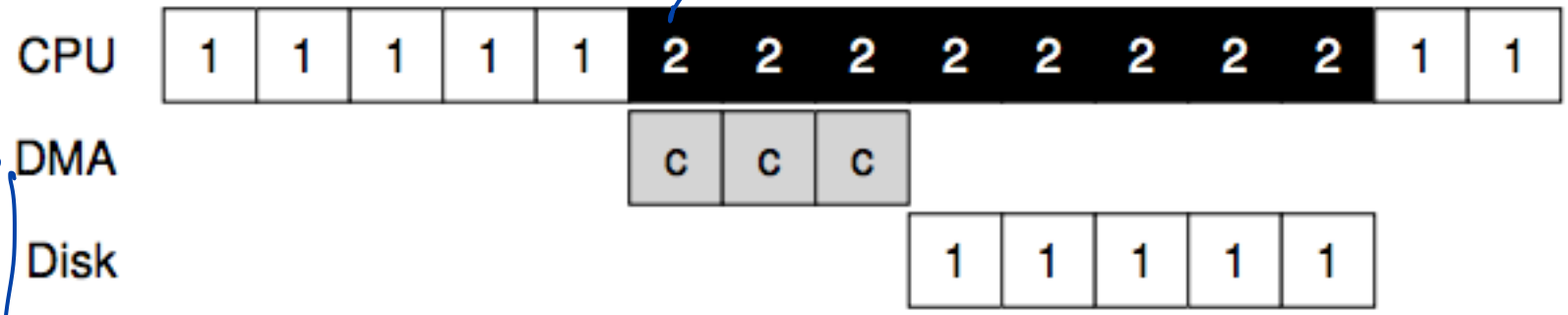
DMA (Direct Memory Access):

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

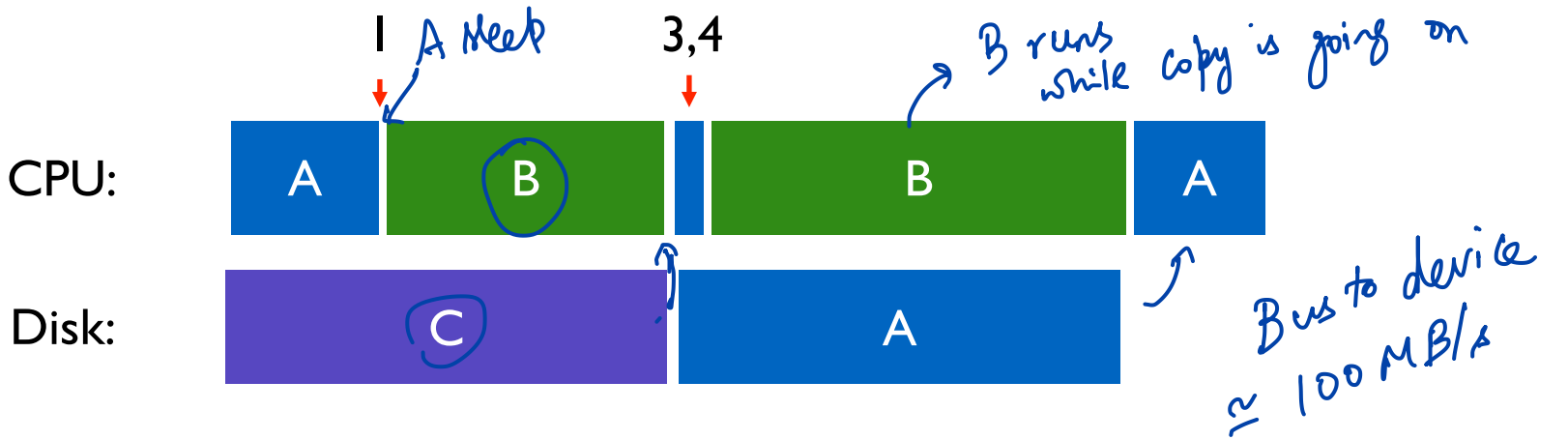


Hardware Component

CPU can now do useful work while copy is going on.



Direct memory Access

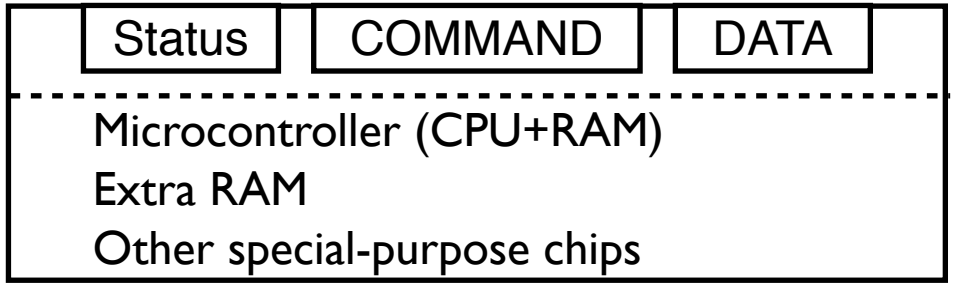



```
while (STATUS == BUSY)                  // 1
   ;
Write data to DATA register             // 2
Write command to COMMAND register       // 3
while (STATUS == BUSY)                  // 4
   ;
```

setup DMA

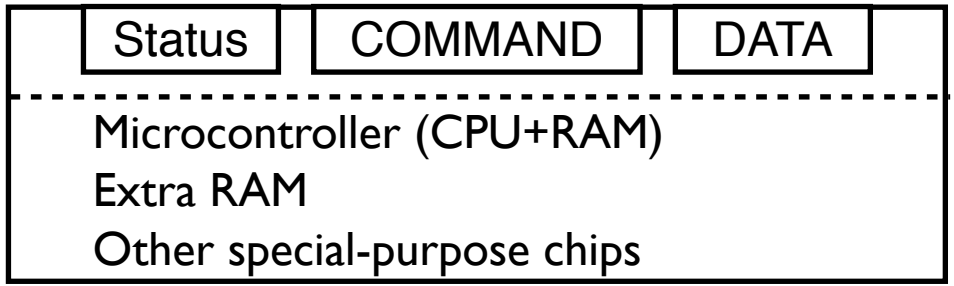
much smaller

PROTOCOL VARIANTS



Status checks: polling *vs.* interrupts

PIO vs DMA




```
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 → Privileged instructions

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

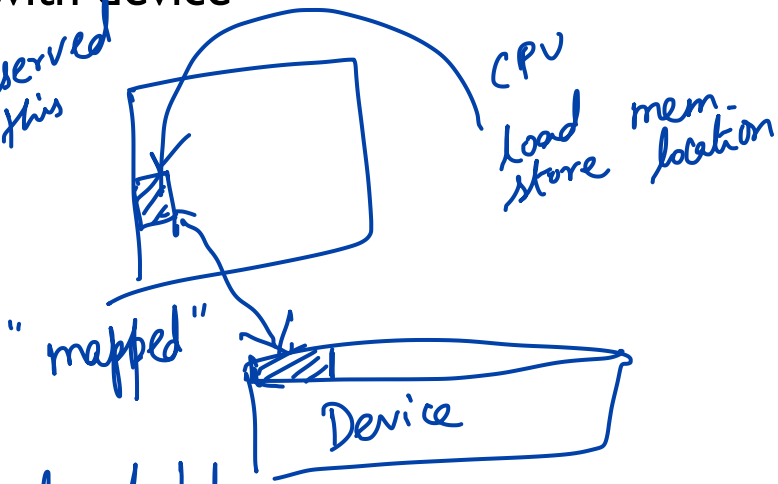
→ IN REG PORT
OUT REG PORT
→ Device

Memory-Mapped I/O → Part of Physical addr space reserved for this

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

→ Only OS can access

Doesn't matter much (both are used)



PROTOCOL VARIANTS

Port typical ranges of memory used for memory mapped devices

Status	COMMAND	DATA

Microcontroller (CPU+RAM)
Extra RAM
Other special-purpose chips

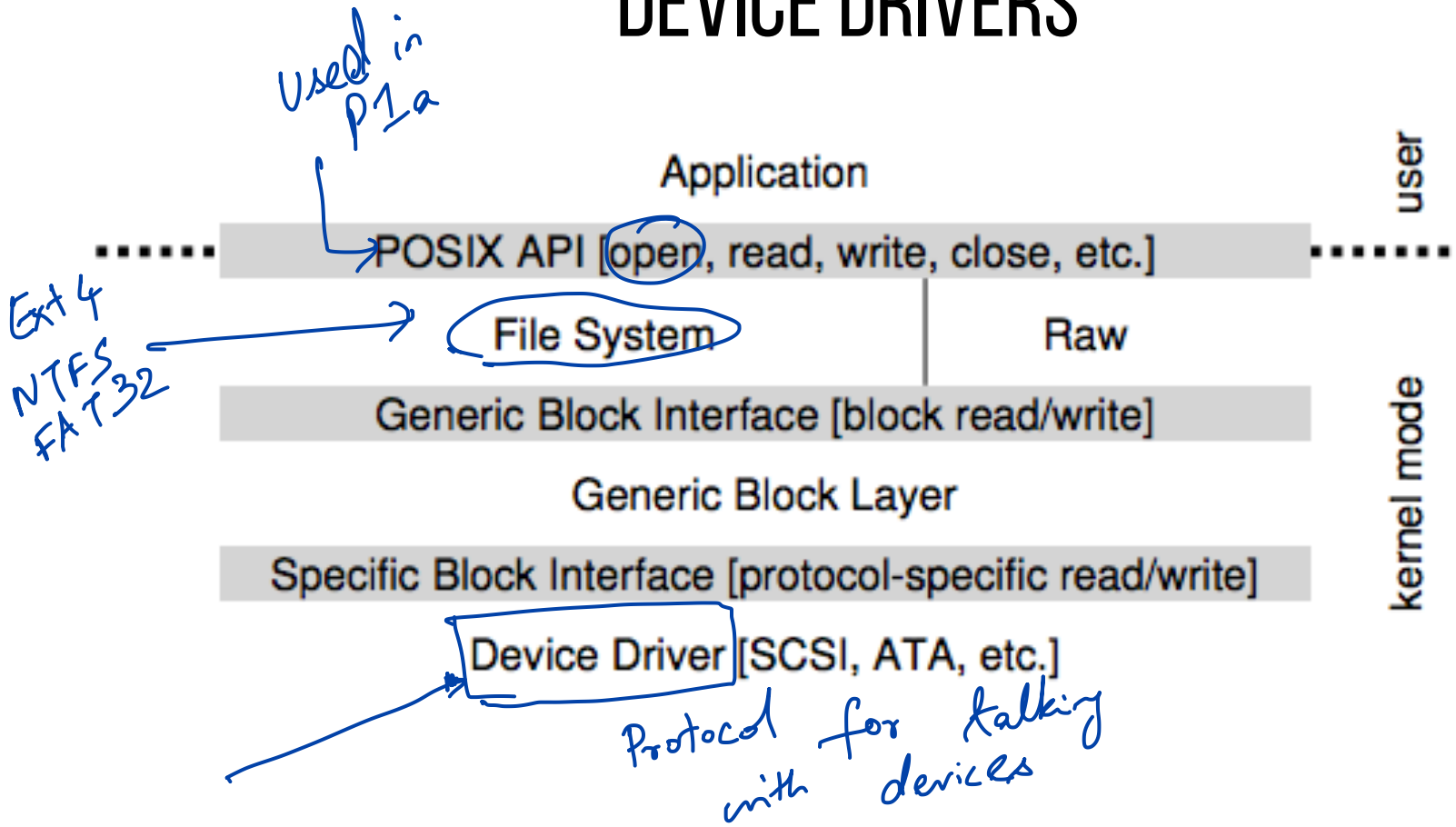
Status checks: polling *vs.* interrupts 1,4

PIO vs DMA → 2

Special instructions vs. Memory mapped I/O
↳ 3

1 → status of a device
2 → Copy data
3 → Copy Command
4 → wait for device

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 → Installed a device driver

Modular / Pluggable

Drivers are **70%** of Linux source code → Dont typically use all!

QUIZ 20

https://tinyurl.com/cs537-sp20-quiz20

~ micro second read/write

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

Polling avoids Context switch

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

Writing data to the device

HARD DISKS

HARD DISK INTERFACE

Per byte most cost efficient storage

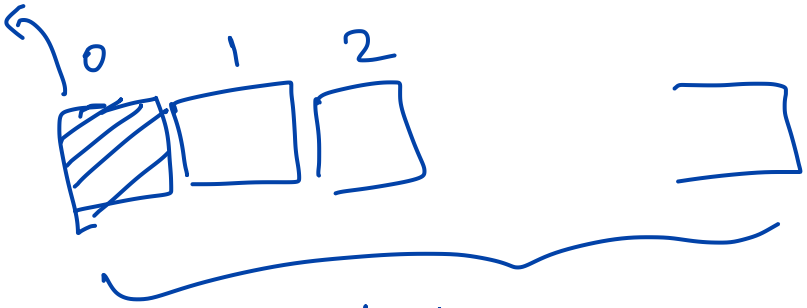
Disk has a sector-addressable address space

Appears as an array of sectors

Sectors are typically 512 bytes

Main operations: reads + writes to sectors

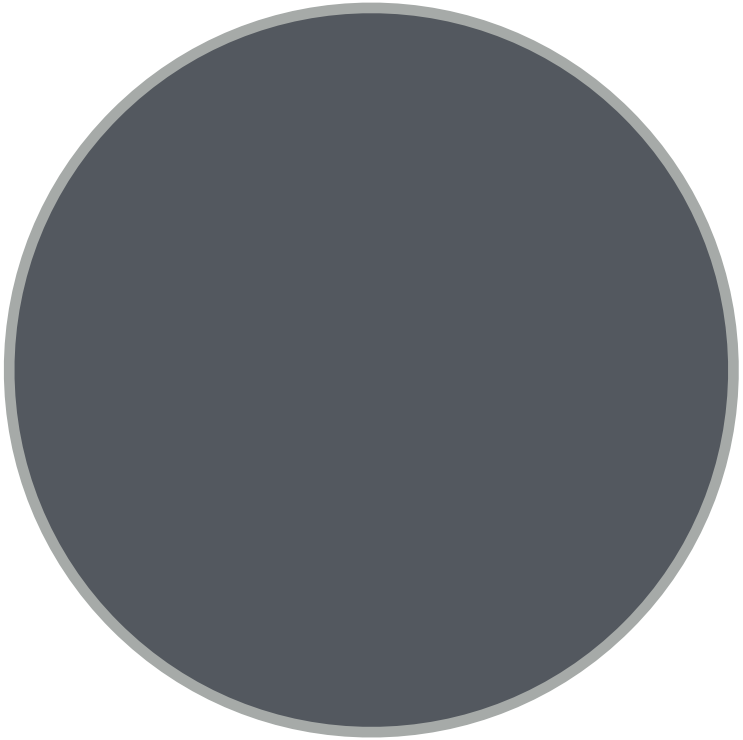
Mechanical and slow (?)

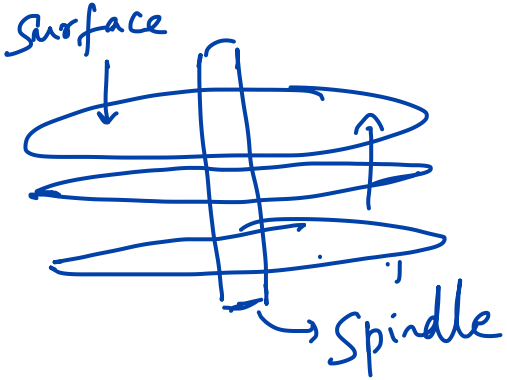


read from sector 2

media that can persist data

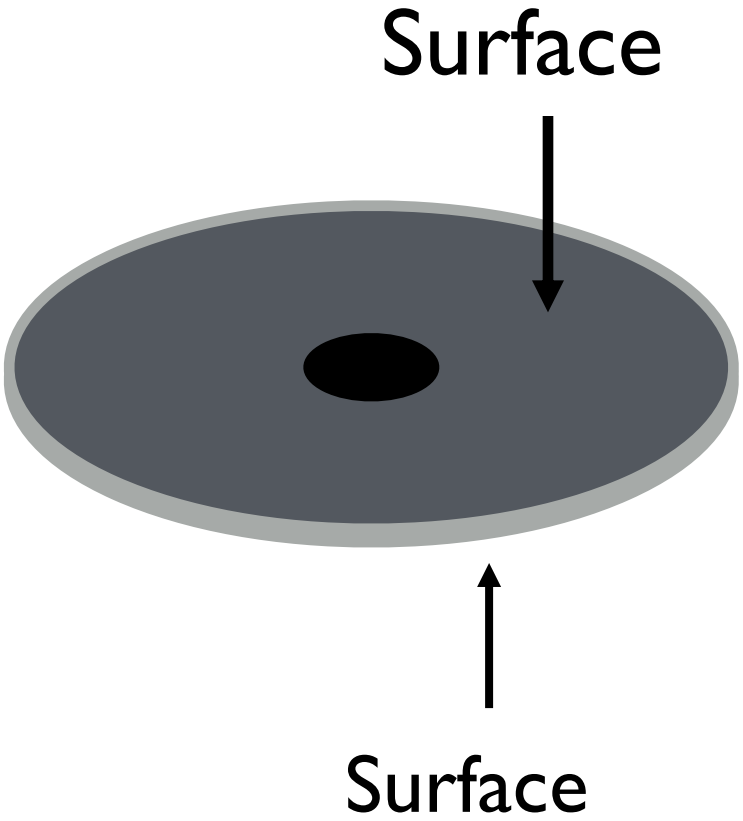
Platter





Both sides are used

Spindle



RPM?

Motor connected to spindle spins platters

Rotations per minute

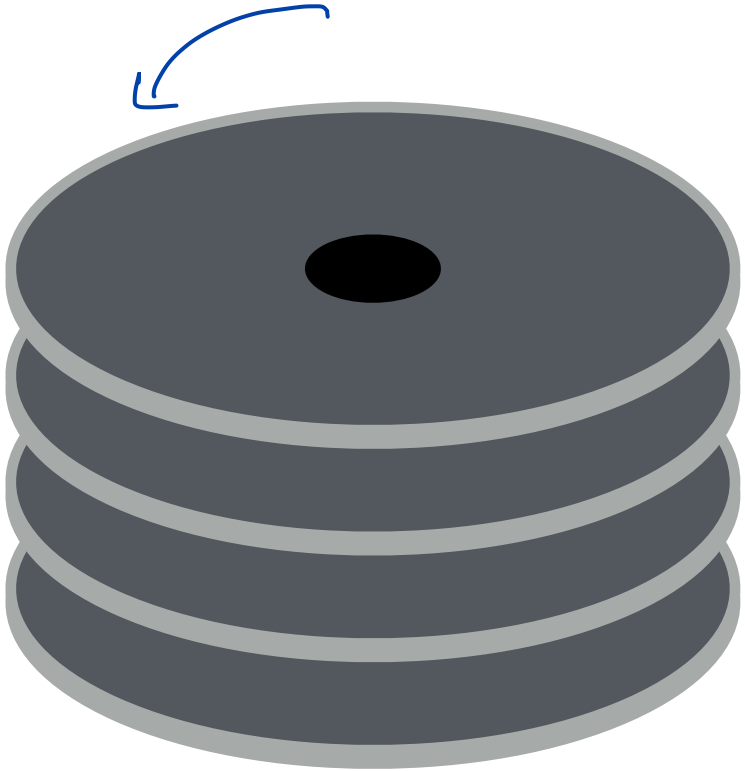
Rate of rotation: RPM

10000 RPM → single rotation is 6 ms

10,000 rotations 60 s

60 000 ms

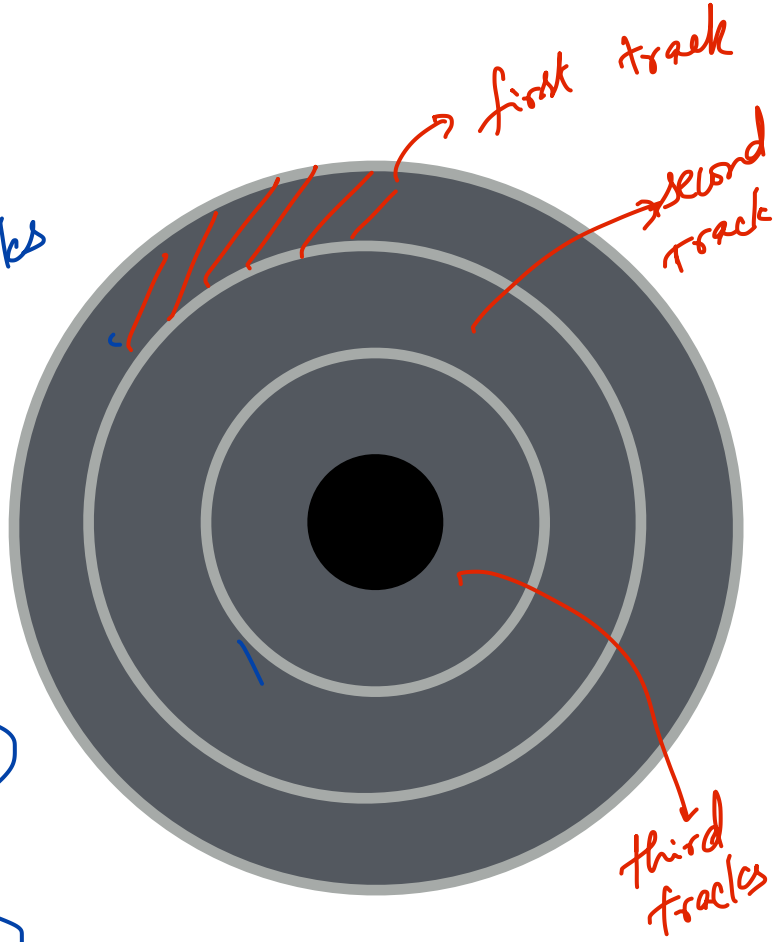
⇒ 1 rotation takes 6 ms



More storage if you have lots of tracks

Surface is divided into rings: tracks

Stack of tracks(across platters): cylinder



Project 4a released today

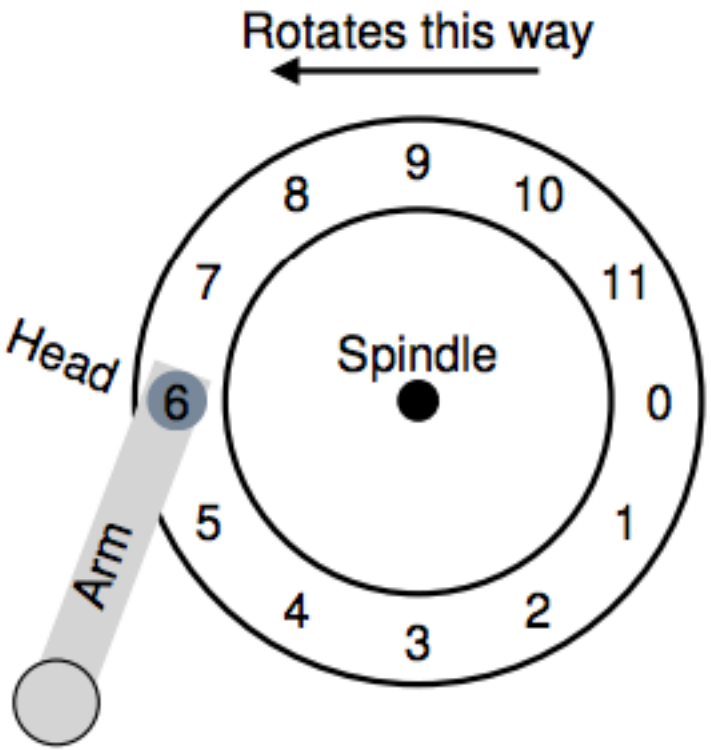
Survey • how did today's class go?

Tracks are divided into numbered sectors

8 sectors in a track

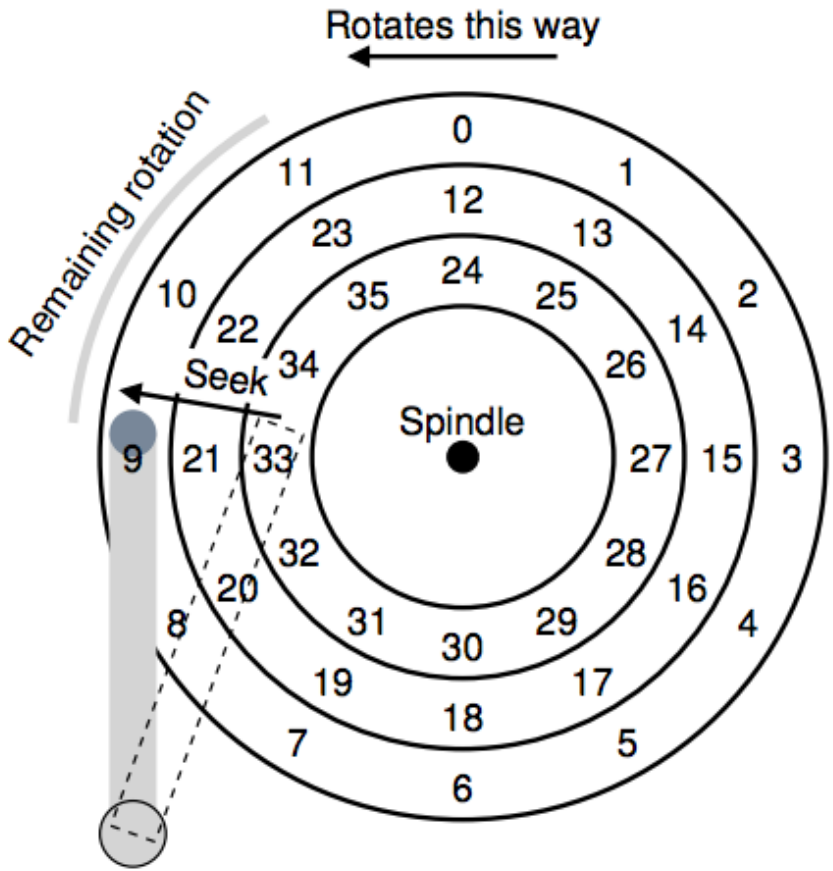
Heads on a moving arm can read from each surface.

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-sp20-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

Project 4a: Out tonight

Grades: Project 2b, 3, midterm by tomorrow!