CS 471 Operating Systems

Yue Cheng

George Mason University

Fall 2019

I/O Devices

Why I/O?

- I/O == Input/Output
- What good is a computer without any I/O devices?
  - Keyboard, display, disks…

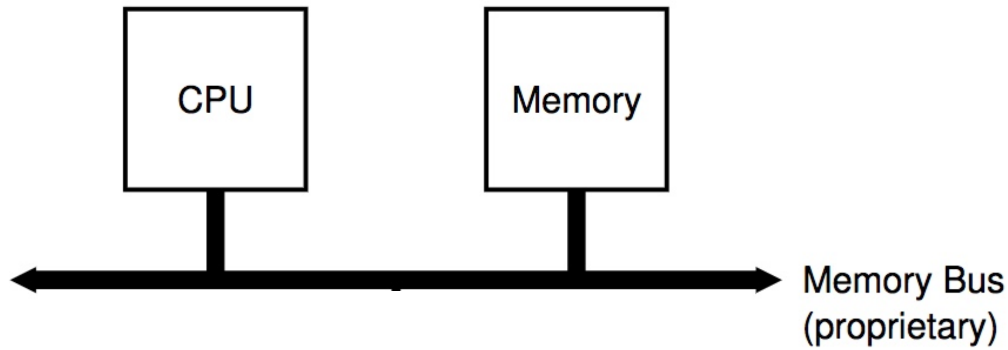
Why I/O?

- I/O == Input/Output

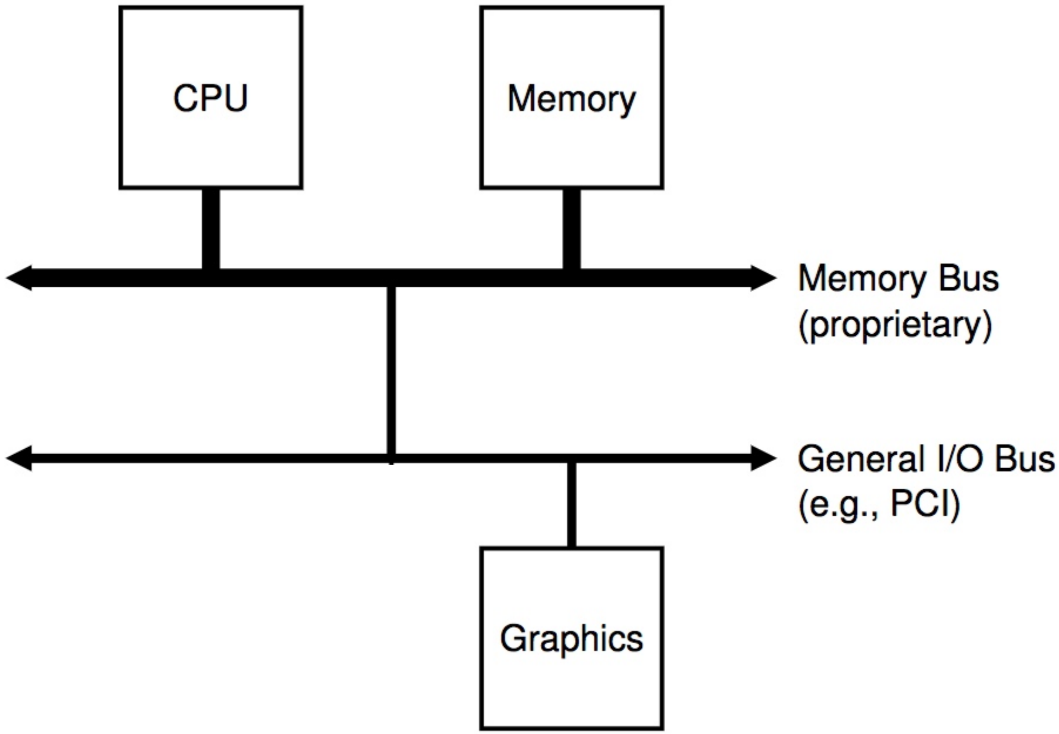
- What good is a computer without any I/O devices?
  - Keyboard, display, disks…

- We want
  - **Hardware**: which will provide direct physical interfaces
  - **OS**: which can interact with different combinations

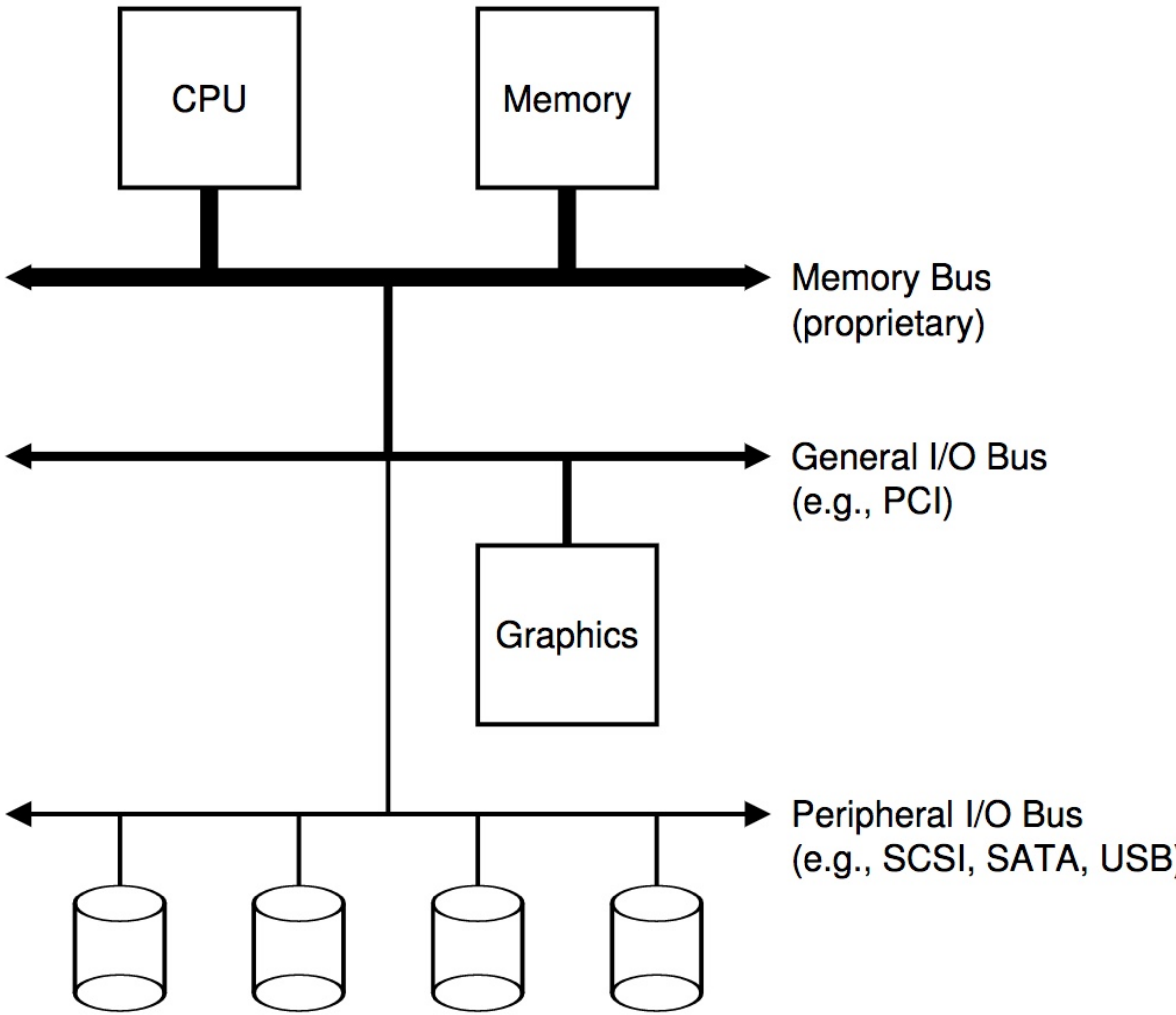
Prototypical System Architecture



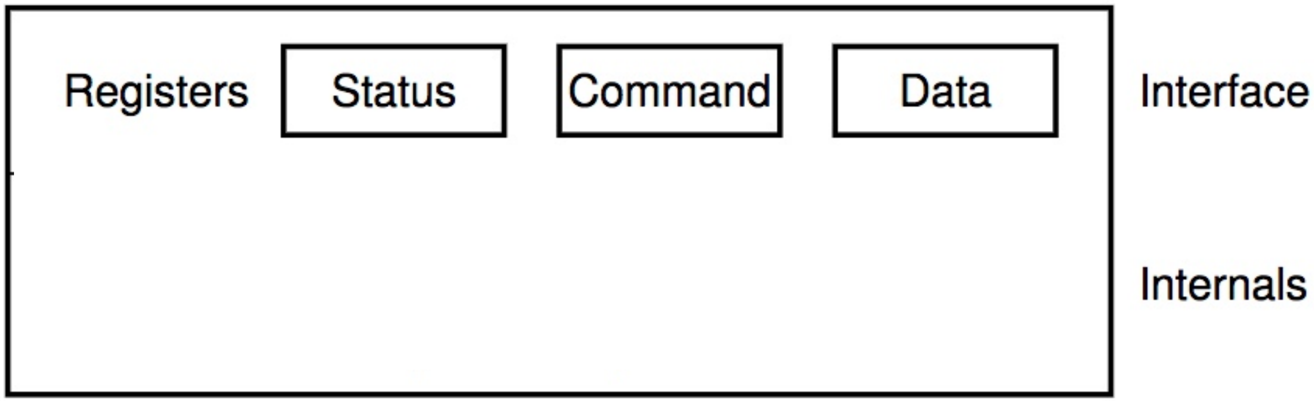
Prototypical System Architecture



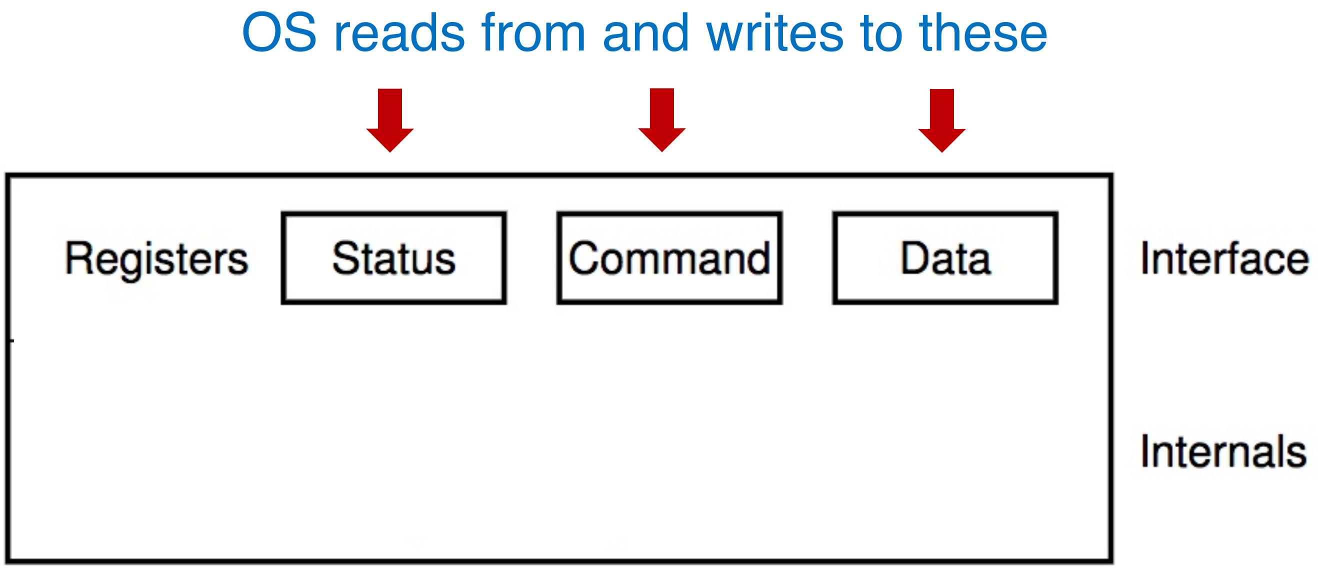
Prototypical System Architecture



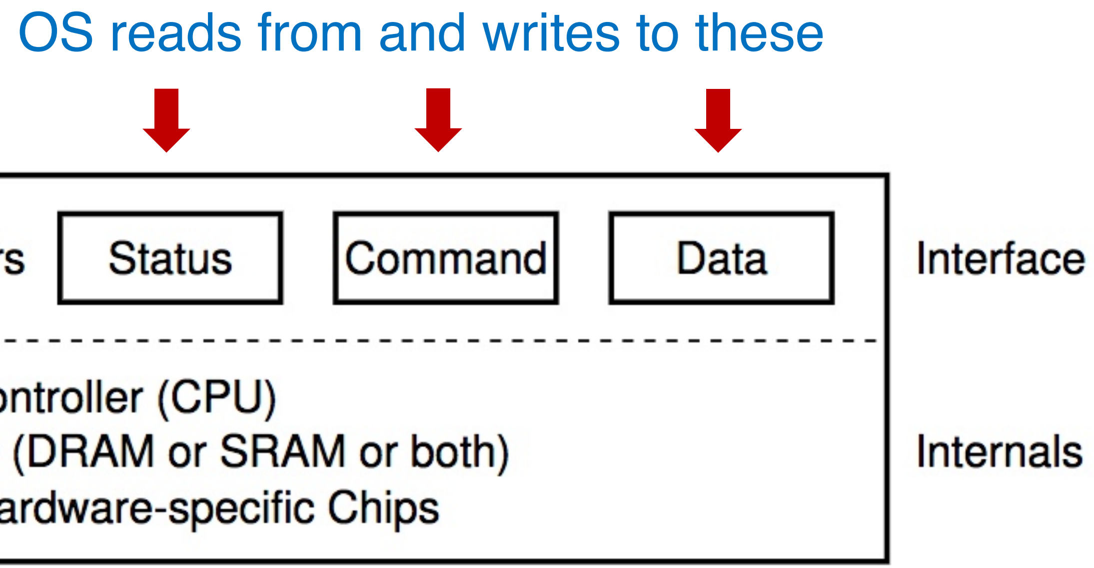
Canonical I/O Device



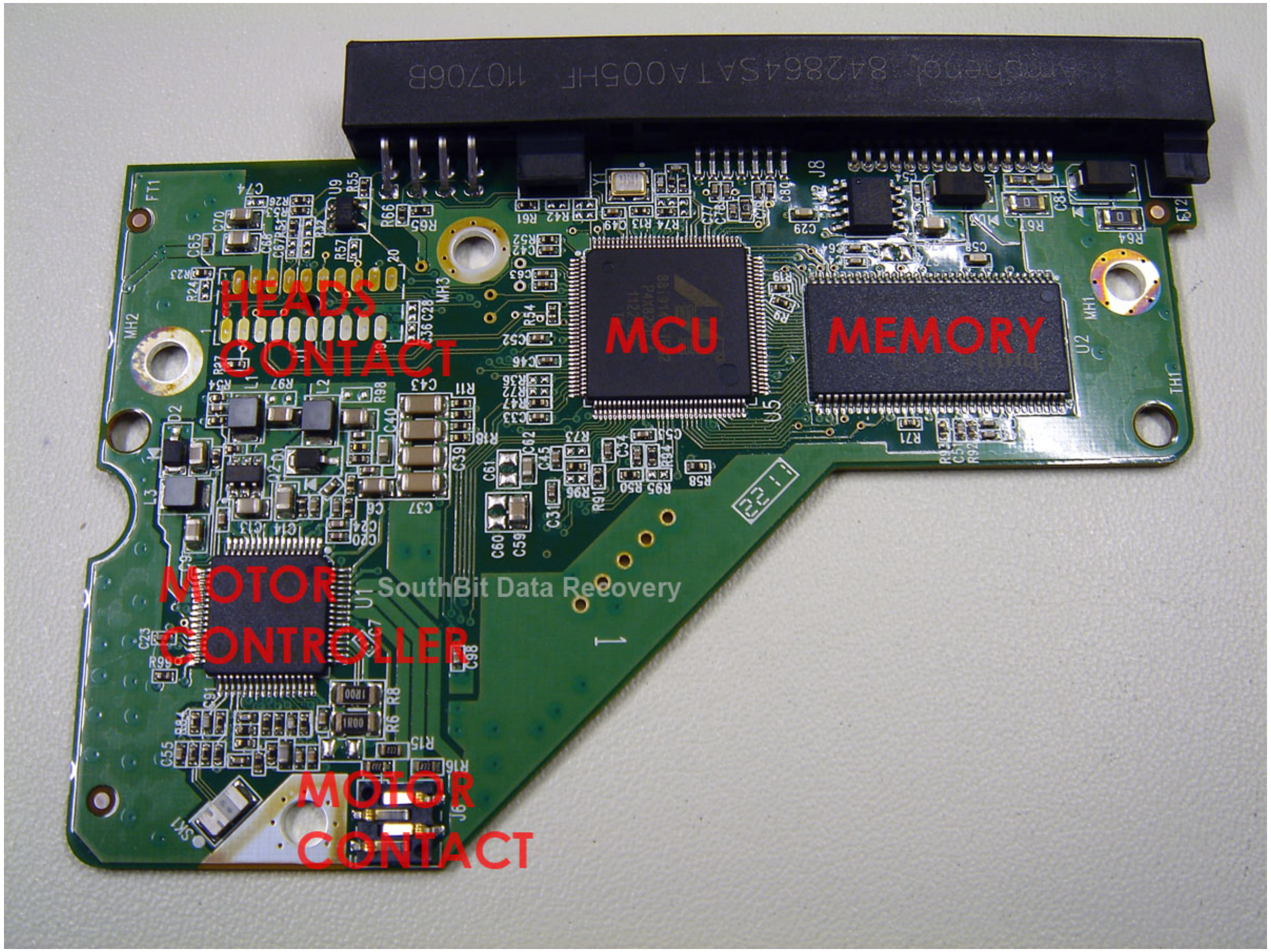
Canonical I/O Device



Canonical I/O Device



A Hard Disk Drive PCB Example



A Basic I/O Protocol

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

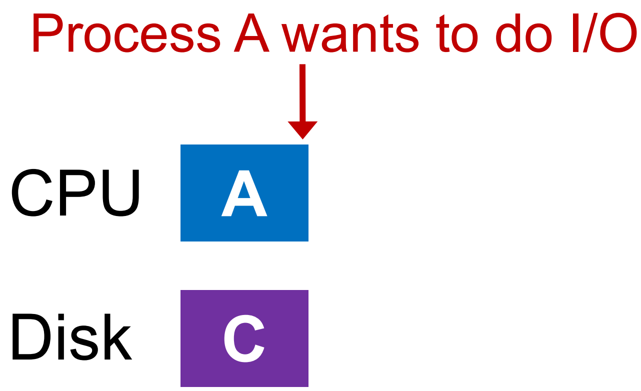
A Basic I/O Protocol

CPU **A**

Disk **C**

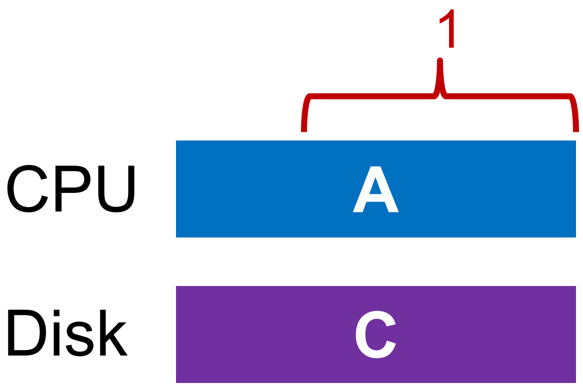
```
while (STATUS == BUSY)              //1
      ; // spin
Write data to DATA register         //2
Write command to COMMAND register //3
while (STATUS == BUSY)              //4
      ; // spin
```

A Basic I/O Protocol



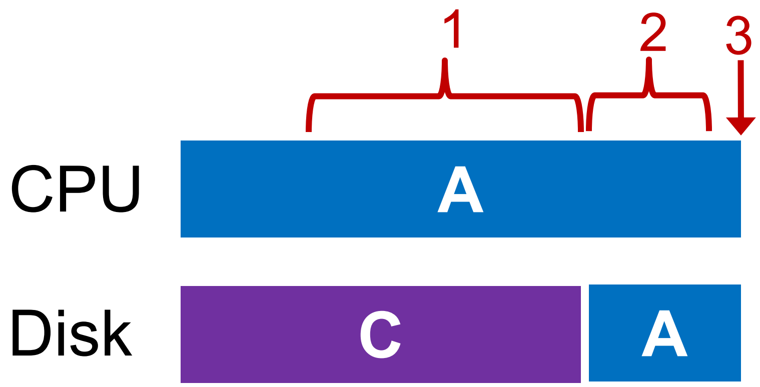

```
while (STATUS == BUSY)                 //1
      ; // spin
Write data to DATA register            //2
Write command to COMMAND register //3
while (STATUS == BUSY)                 //4
      ; // spin
```

A Basic I/O Protocol



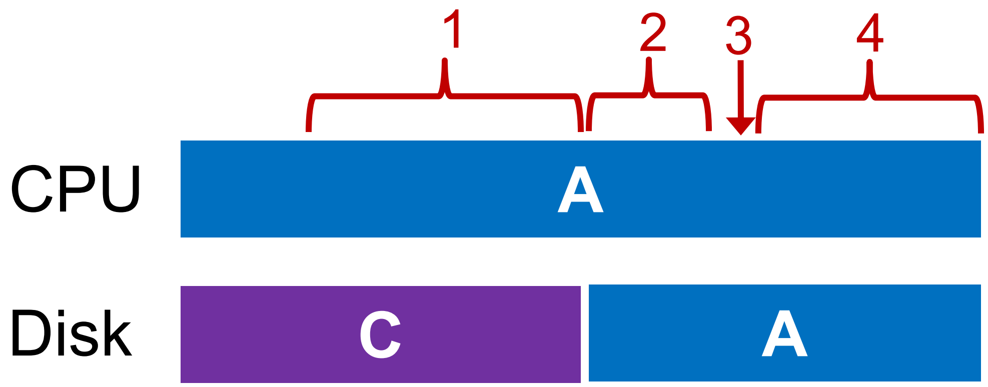

```
while (STATUS == BUSY)              //1
      ; // spin
Write data to DATA register         //2
Write command to COMMAND register   //3
while (STATUS == BUSY)              //4
      ; // spin
```

A Basic I/O Protocol



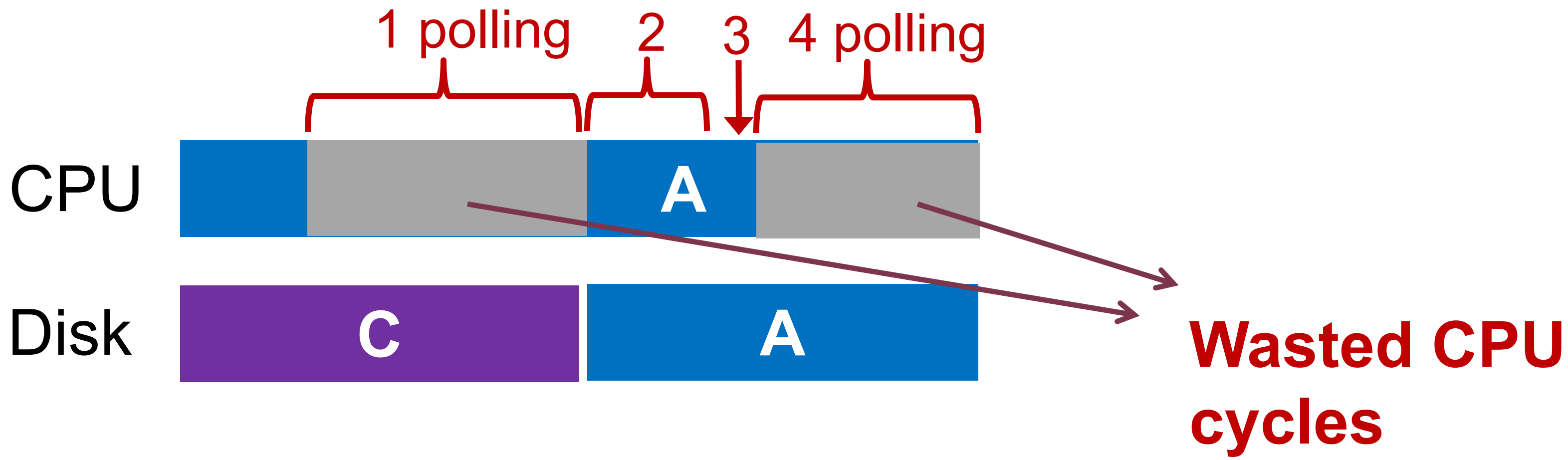

```
while (STATUS == BUSY)              //1
      ; // spin
Write data to DATA register         //2
Write command to COMMAND register //3
while (STATUS == BUSY)              //4
      ; // spin
```

A Basic I/O Protocol



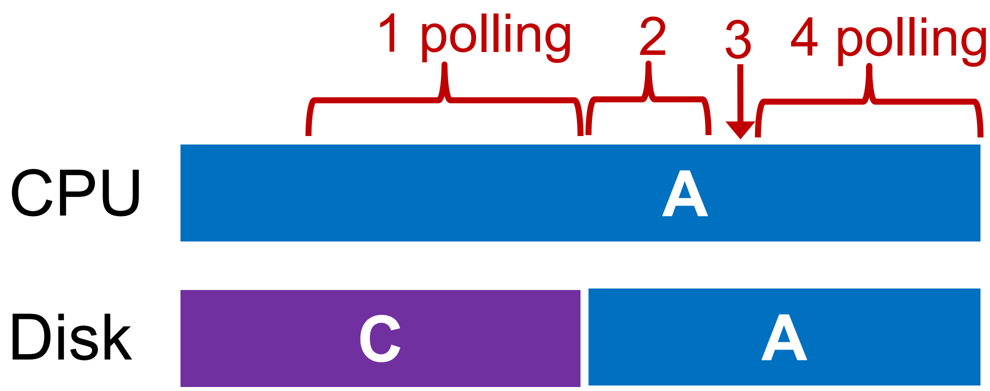

```
while (STATUS == BUSY)              //1
      ; // spin
Write data to DATA register         //2
Write command to COMMAND register //3
while (STATUS == BUSY)              //4
      ; // spin
```

A Basic I/O Protocol



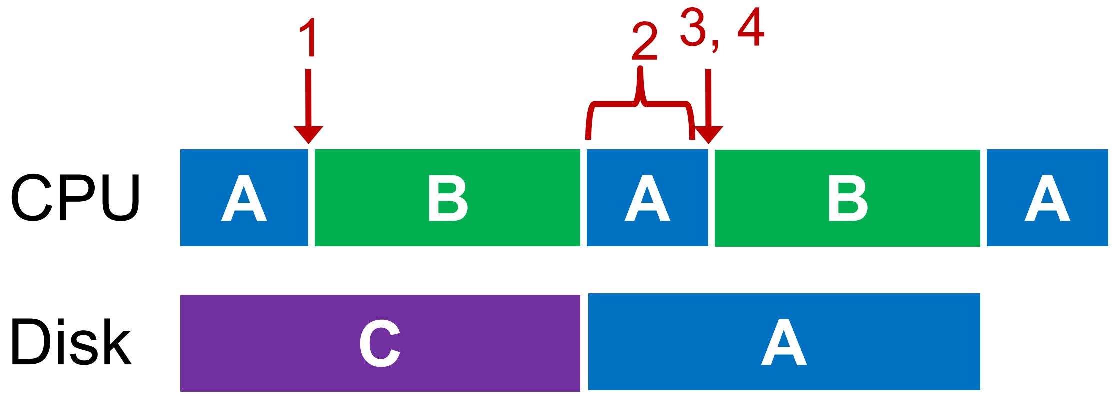

```
while (STATUS == BUSY)                 //1
      ; // spin
Write data to DATA register            //2
Write command to COMMAND register //3
while (STATUS == BUSY)                 //4
      ; // spin
```

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




```
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

- Any potential issues for interrupts?

Interrupts vs. Polling

- Any potential issues for interrupts?

- Interrupts can lead to **livelock**
  - E.g., flood of network packets

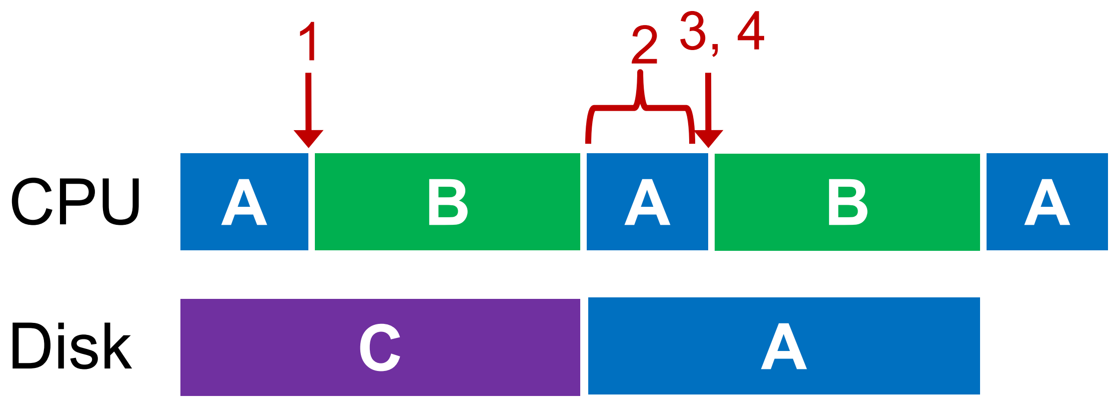
Interrupts vs. Polling

- Any potential issues for interrupts?

- Interrupts can lead to **livelock**
  - E.g., flood of network packets

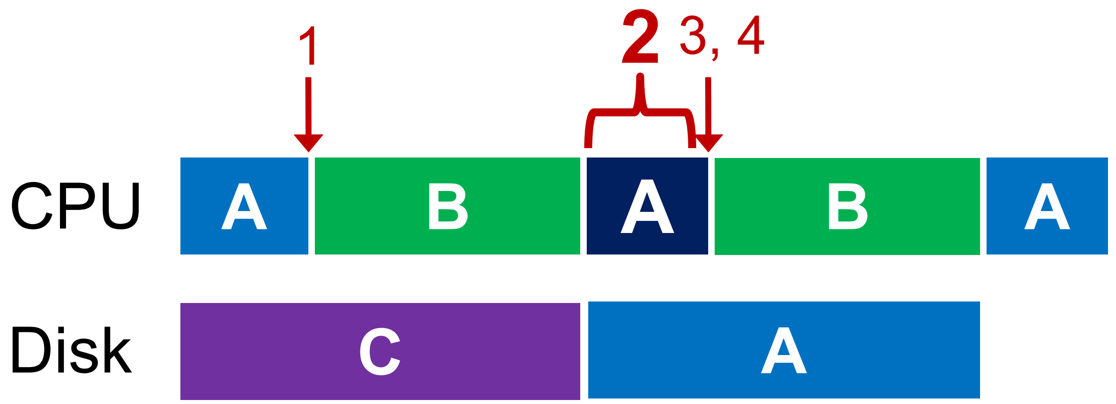
- Techniques
  - Hybrid approach: polling + interrupts
  - Interrupt coalescing: batching a bunch interrupts in one go

Where else Can We Optimize?




```
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;
```

Data Transfer



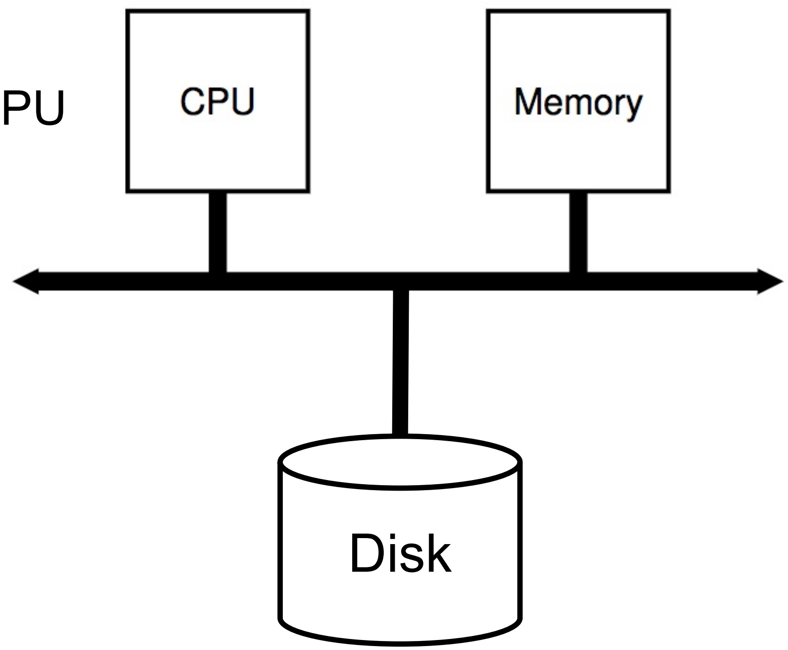

```
while (STATUS == BUSY)                  //1
      wait for interrupt;
Write data to DATA register             //2
Write command to COMMAND register //3
while (STATUS == BUSY)                  //4
      wait for interrupt;
```

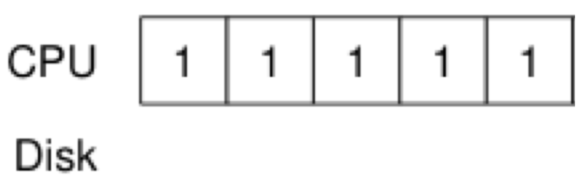
Programmed I/O vs. Direct Memory Access

- PIO (Programmed I/O)
  - CPU directly tells device what data is
  - CPU involved in data transfer
- DMA (Direct Memory Access)
  - CPU leaves data in memory
  - DMA hardware does data copy

PIO Data Flow

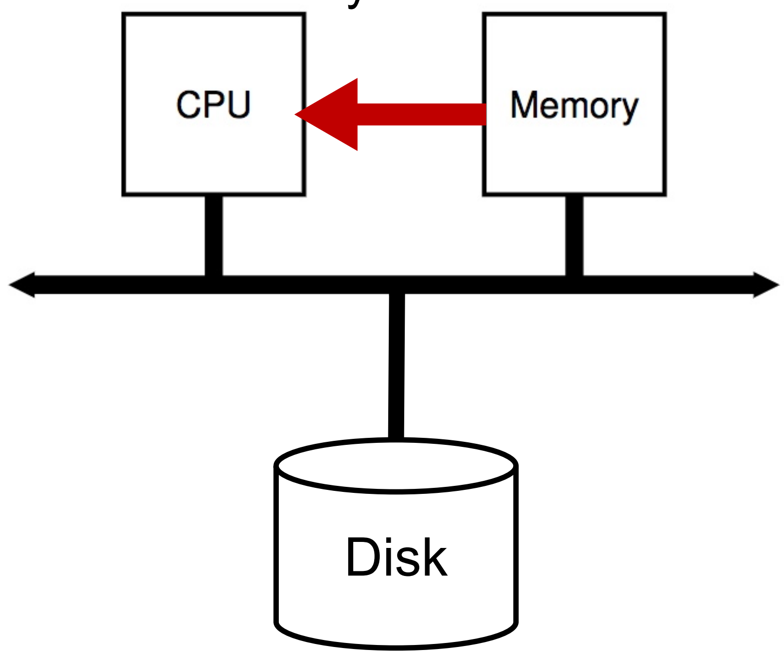
1. Executing P1 on CPU

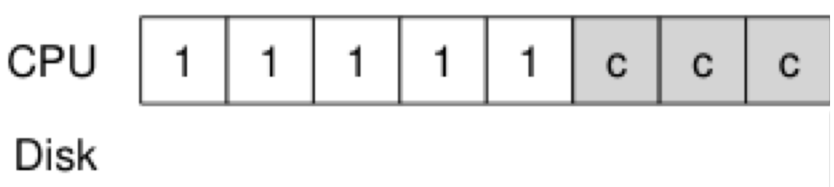




PIO Data Flow

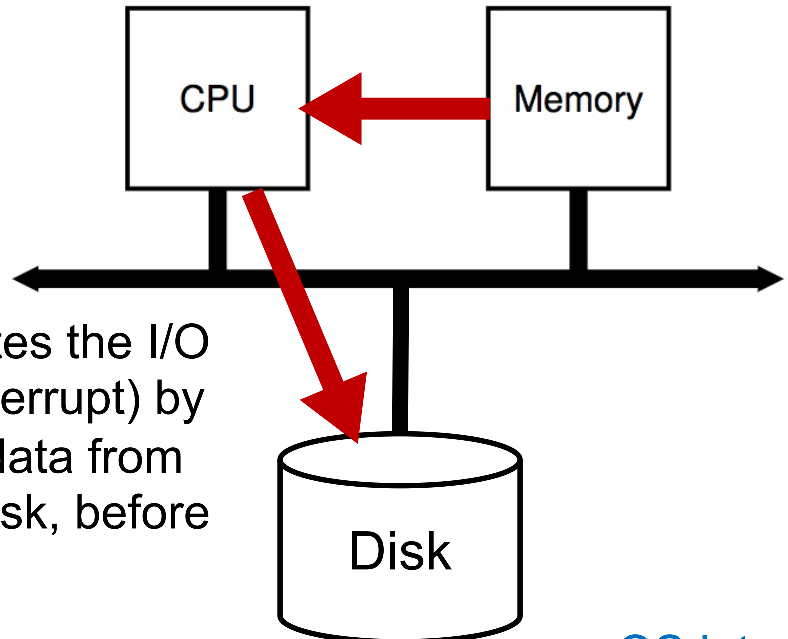
2. Copy data from memory via CPU





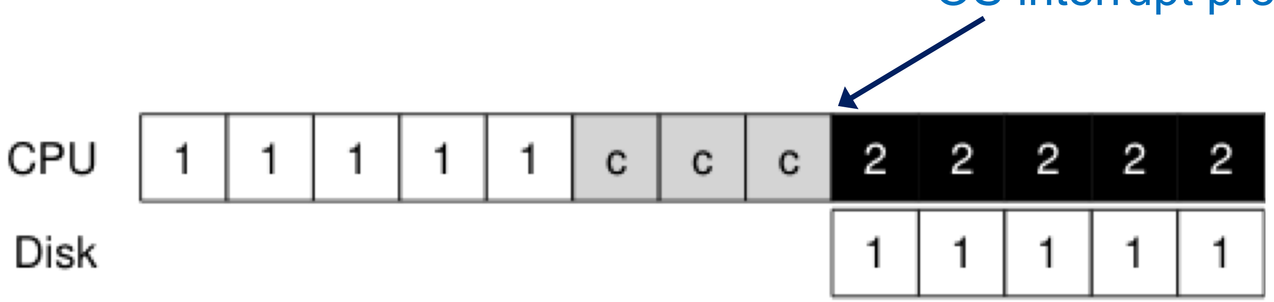
Note: c == copy memory words

PIO Data Flow



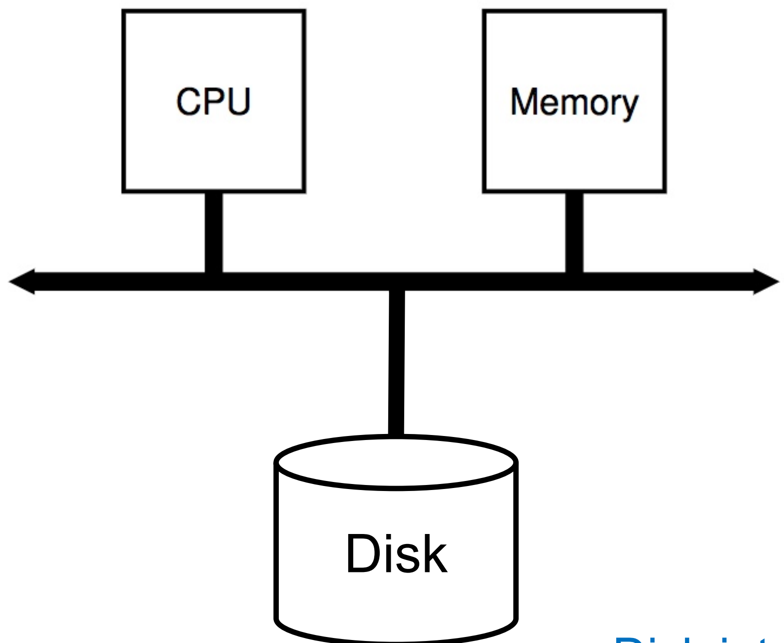
3. CPU initiates the I/O (w/ an OS interrupt) by copying the data from memory to disk, before running P2

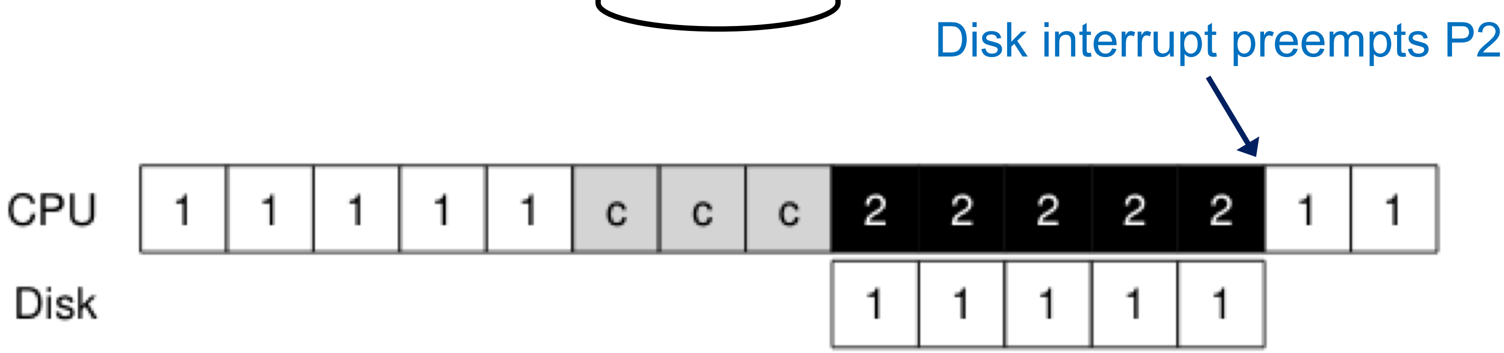
OS interrupt preempts P1



PIO Data Flow

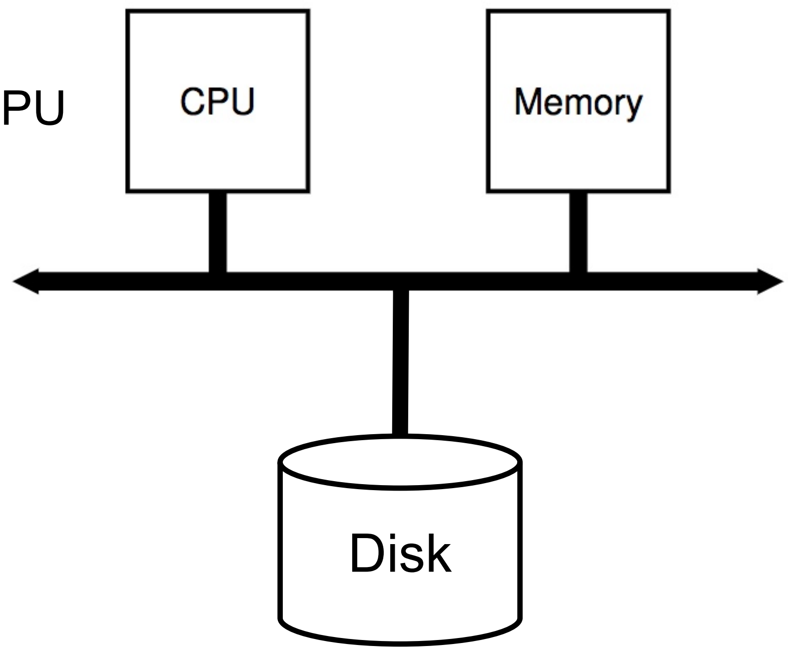
4. Done with I/O, Disk interrupts P2 and re-schedules P1 on CPU





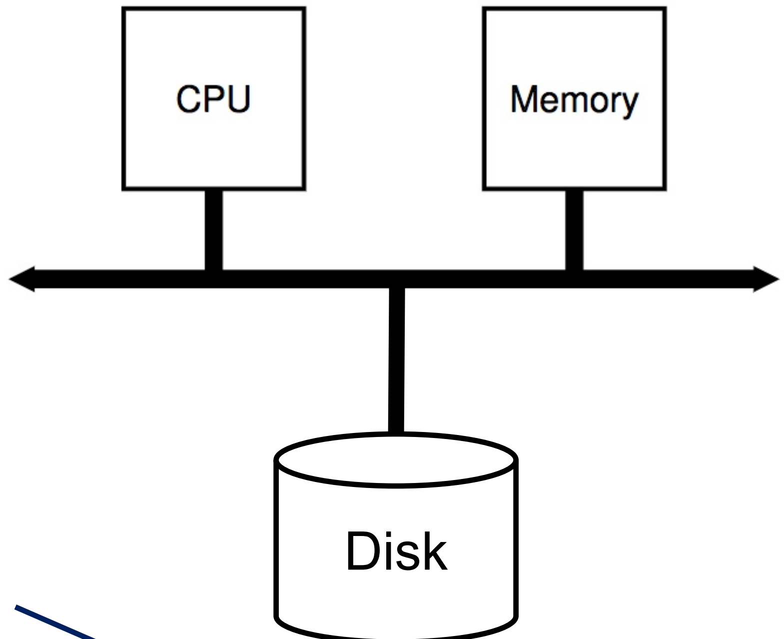
DMA Data Flow

1. Executing P1 on CPU



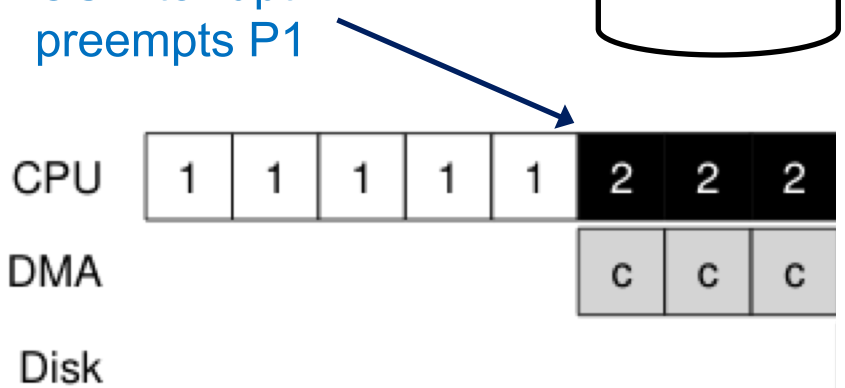
CPU	1	1	1	1	1
DMA					
Disk					

DMA Data Flow

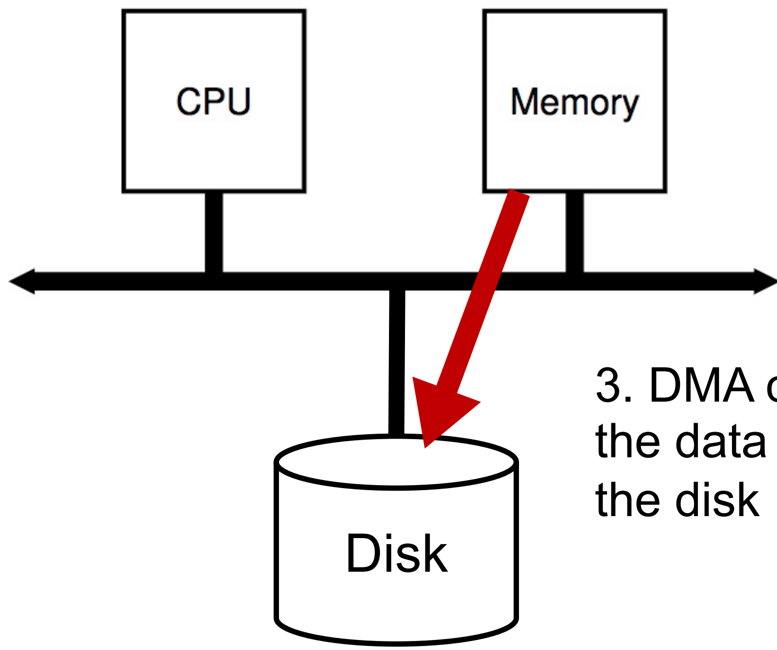


2a. OS initiates DMA by telling the DMA engine where data lives in memory, how much to copy, and which device to send it to;
2b. DMA then copies the data from memory

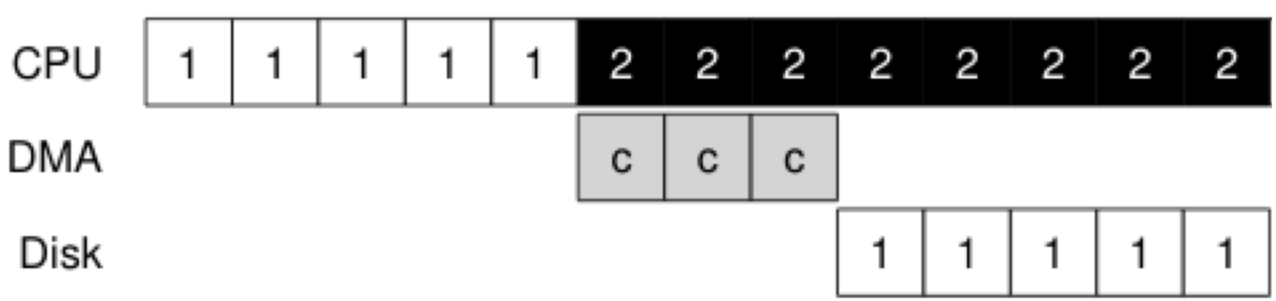
OS interrupt preempts P1



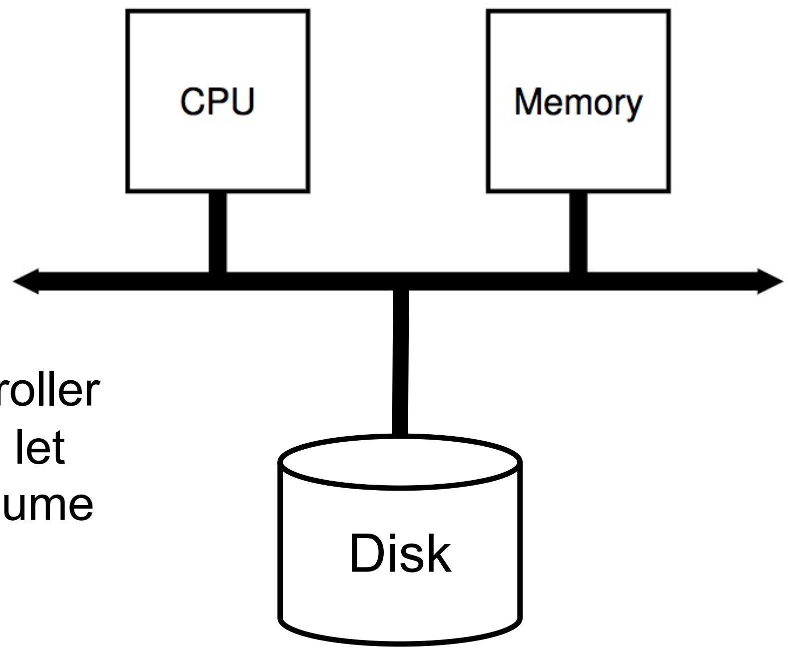
DMA Data Flow



3. DMA controller copies the data from memory to the disk

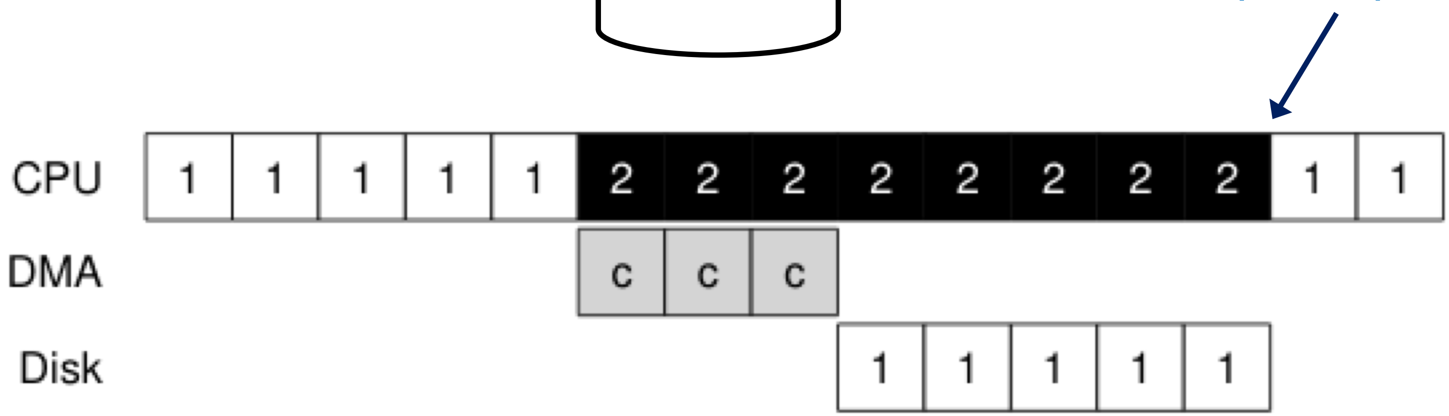


DMA Data Flow

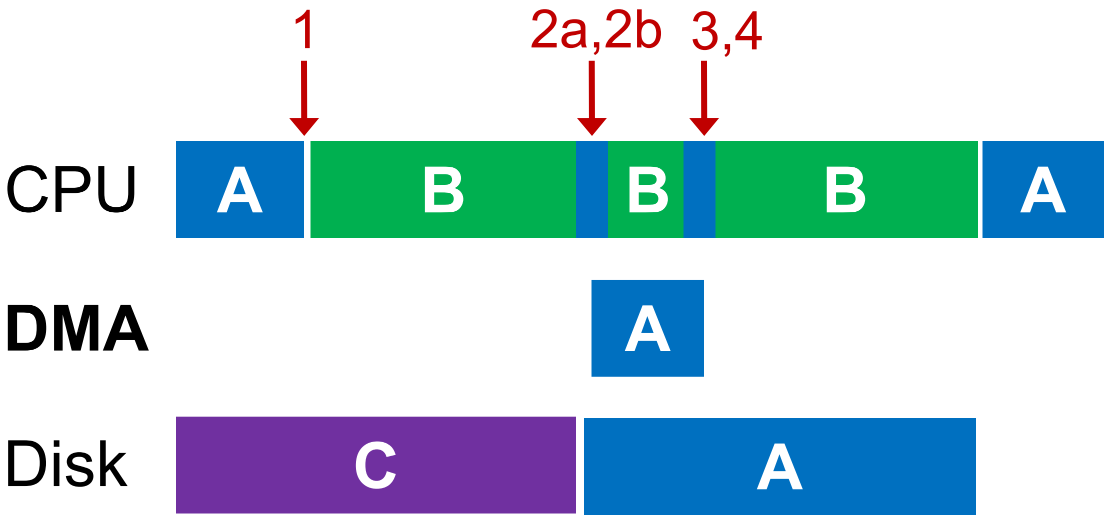


4. When DMA is complete, DMA controller raises an interrupt to let OS know P1 can resume

DMA interrupt preempts P2



DMA




```
while (STATUS == BUSY)                   //1
      wait for interrupt;
Initiate DMA transfer                    //2a
Wait for interrupt                       //2b
Write command to COMMAND register        //3
while (STATUS == BUSY)                   //4
      wait for interrupt;
```

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

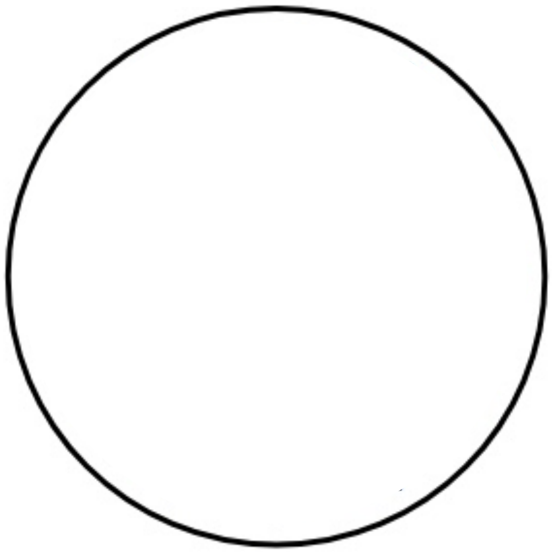
- Sectors are typically 512 or 4096 bytes

- Main operations
  - Read from sectors (blocks)
  - Write to sectors (blocks)

Disk Structure

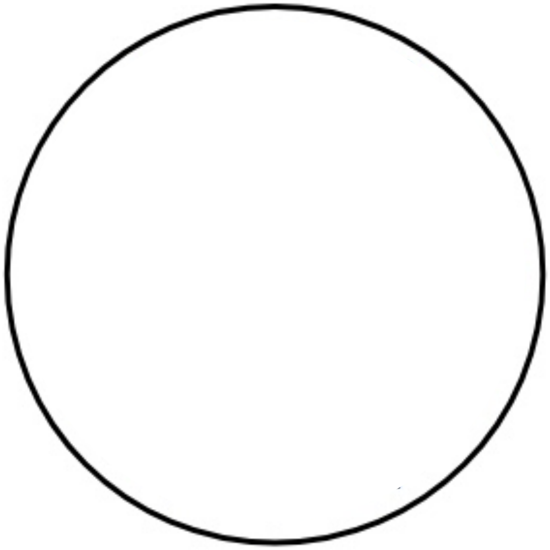
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the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

Internals of Hard Disk Drive (HDD)



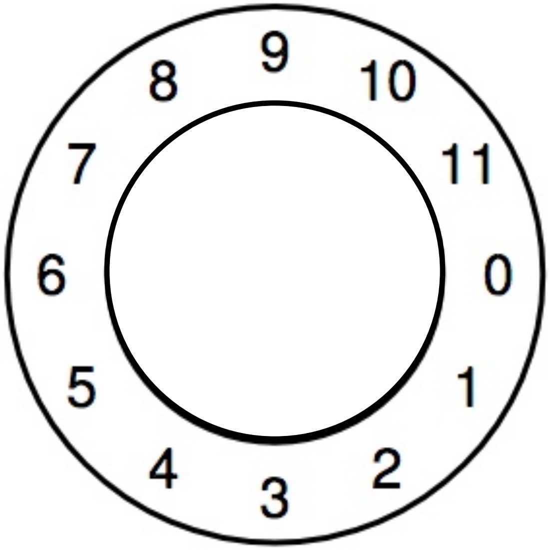
Internals of Hard Disk Drive (HDD)

Platter
Covered with a magnetic film



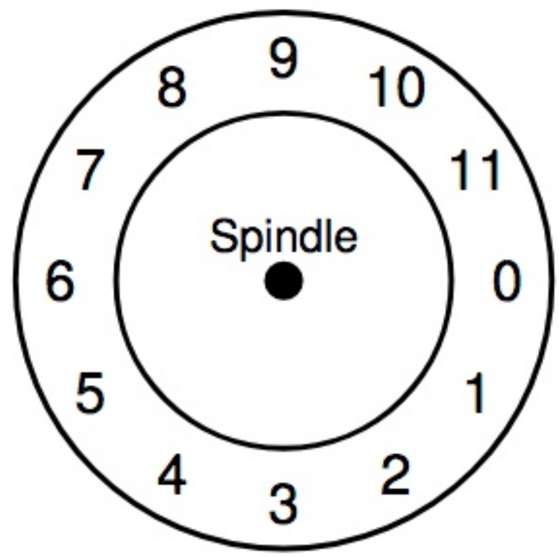
Internals of Hard Disk Drive (HDD)

A single track example



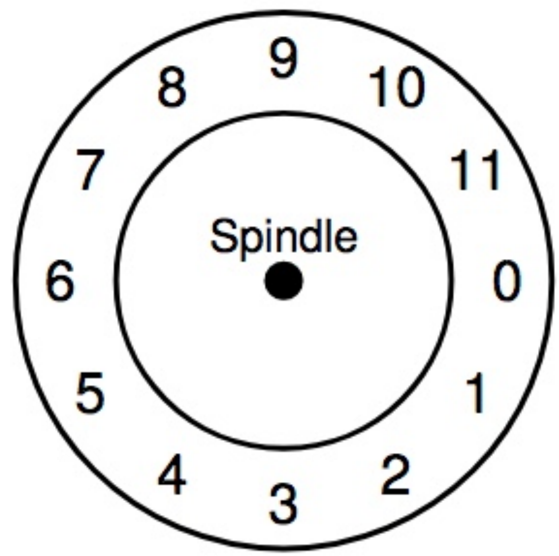
Internals of Hard Disk Drive (HDD)

Spindle in the center of the surface



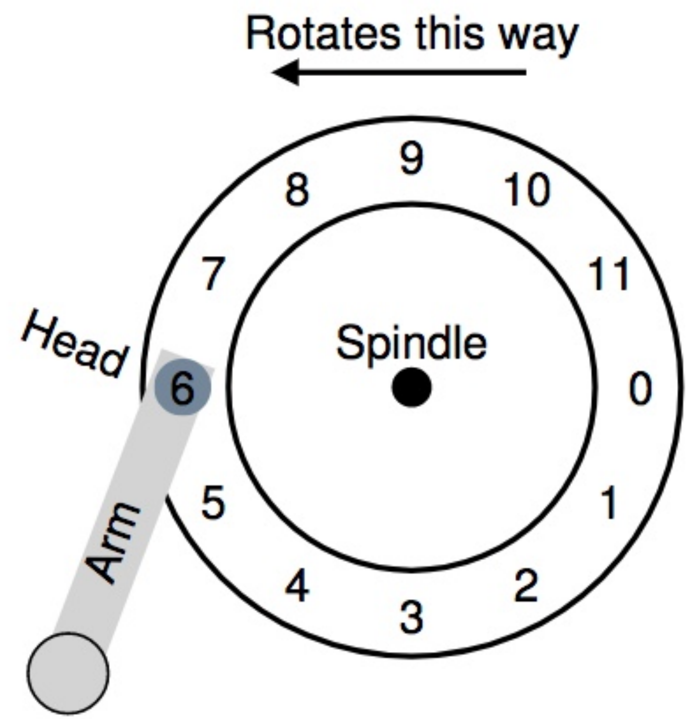
Internals of Hard Disk Drive (HDD)

The track is divided into numbered sectors

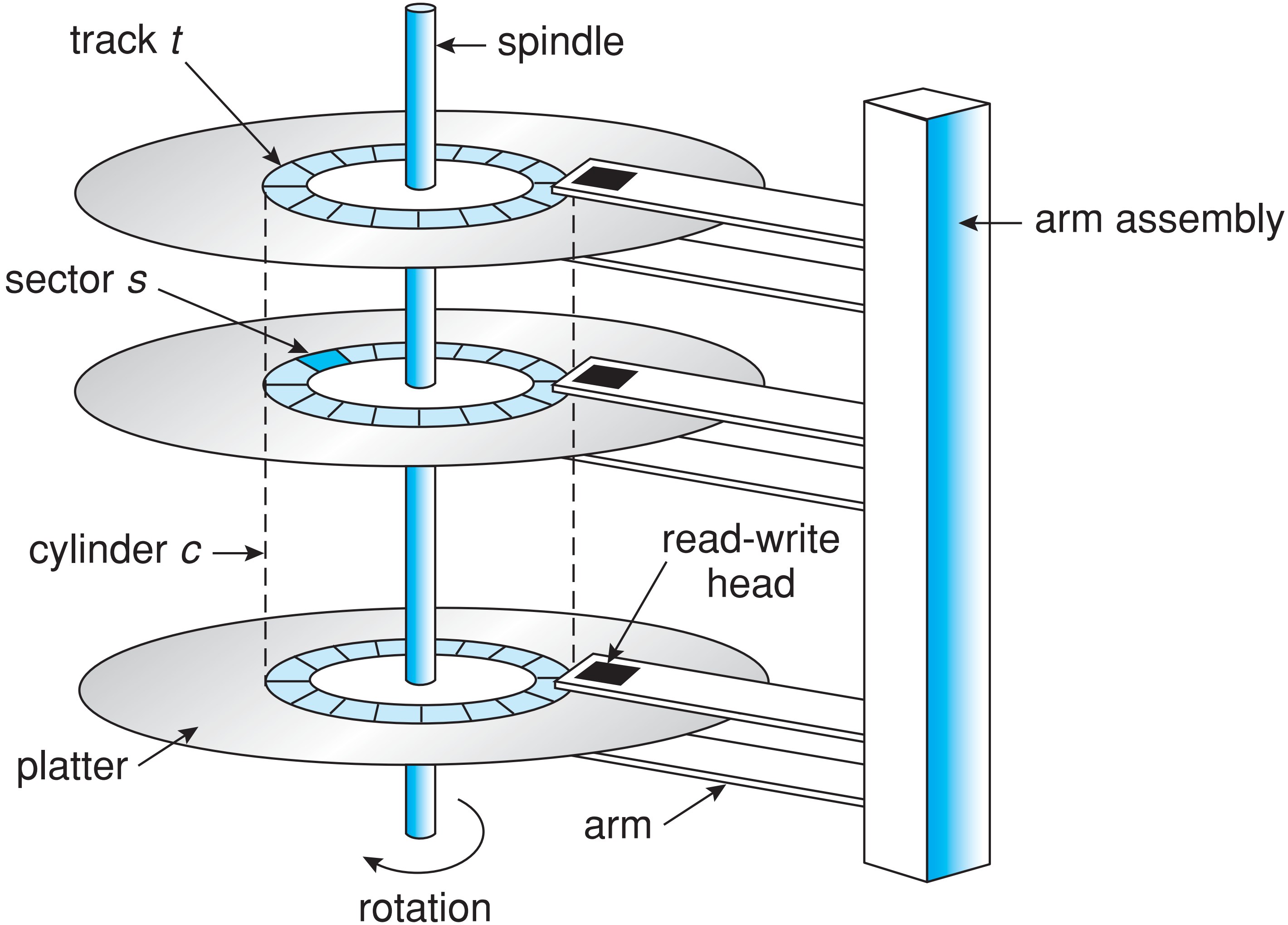


Internals of Hard Disk Drive (HDD)

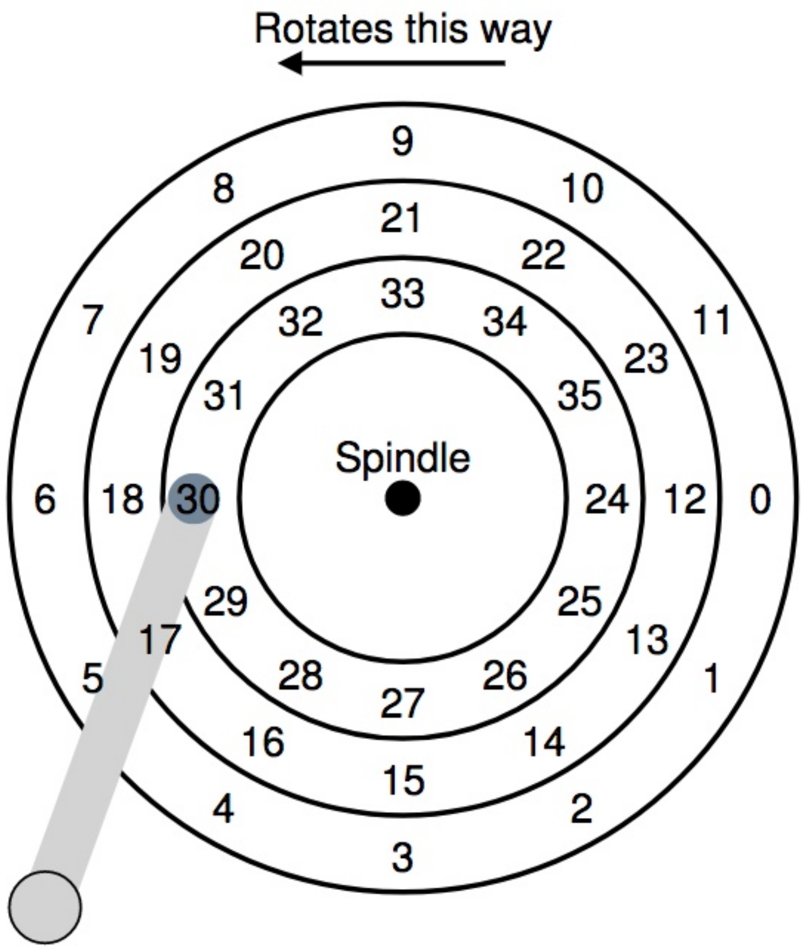
A single track + an arm + a head



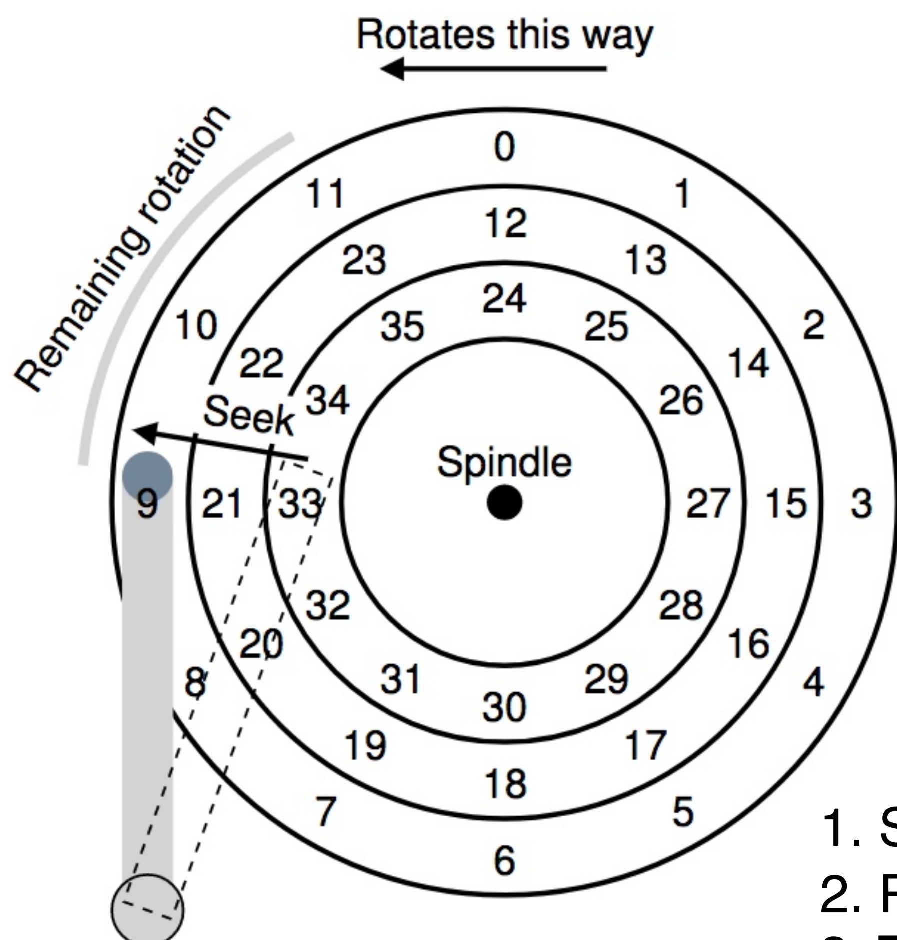
HDD Mechanism (3D view)



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 (4.16 ms) — Avg Rotate (4.2 ms)

The First Commercial Disk Drive

- 1956 IBM RAMDAC computer
  - 5M (7-bit) characters
  - 50 x 24” platters
  - Access time <= 1 sec