I/O and Storage:
I/O Basics

CS 571: Operating Systems (Spring 2020)

Lecture 9a

Yue Cheng

Some material taken/derived from:

- Wisconsin CS-537 materials created by Remzi Arpaci-Dusseau.

I/O Devices

Why I/O?

- I/O == Input/Output

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

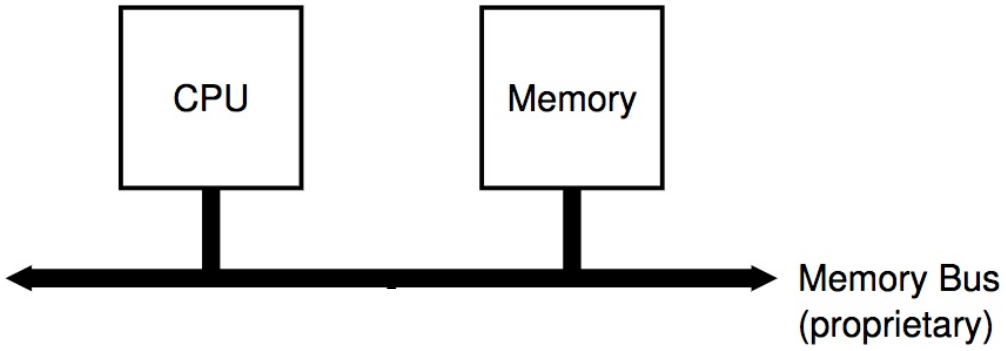
Why I/O?

- I/O == Input/Output

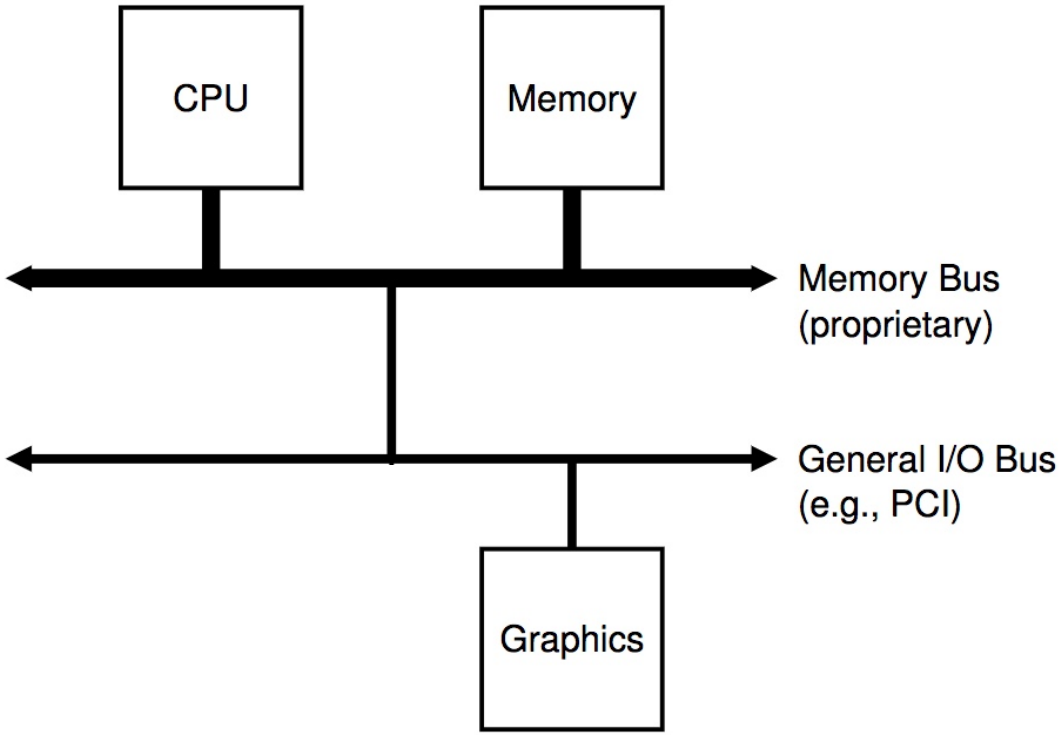
- What good is a computer without any I/O devices?
  - Keyboard, display, disks…

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

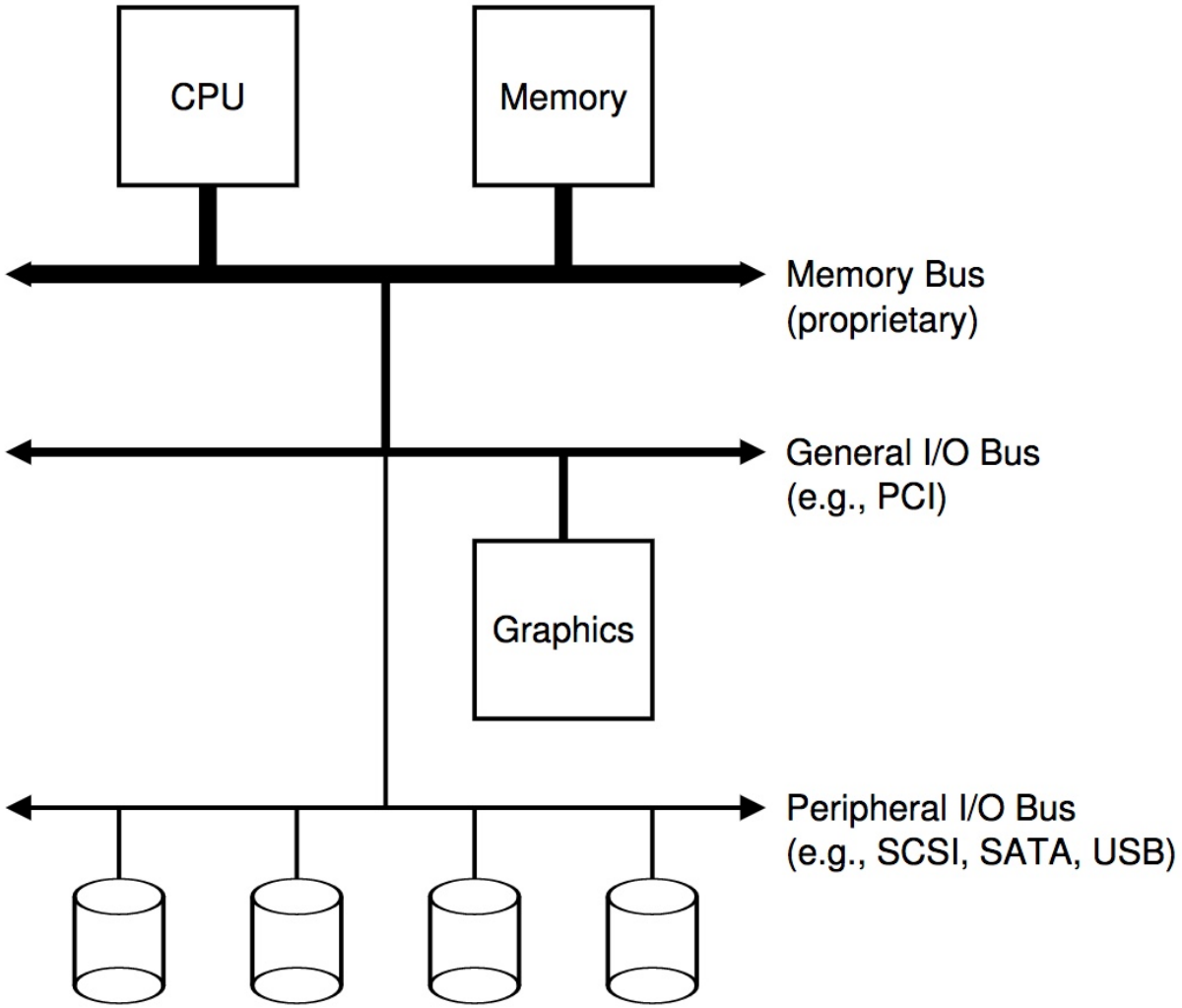
Prototypical System Architecture



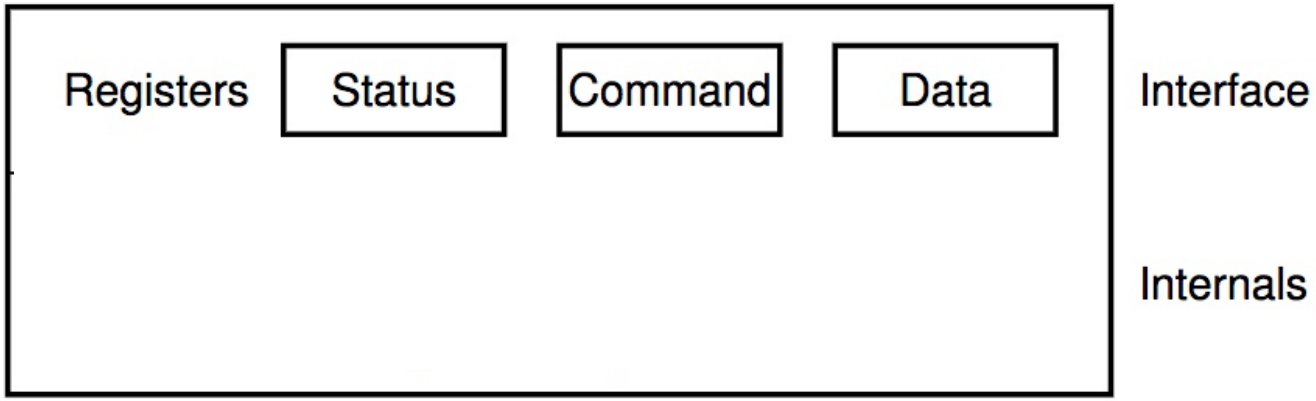
Prototypical System Architecture



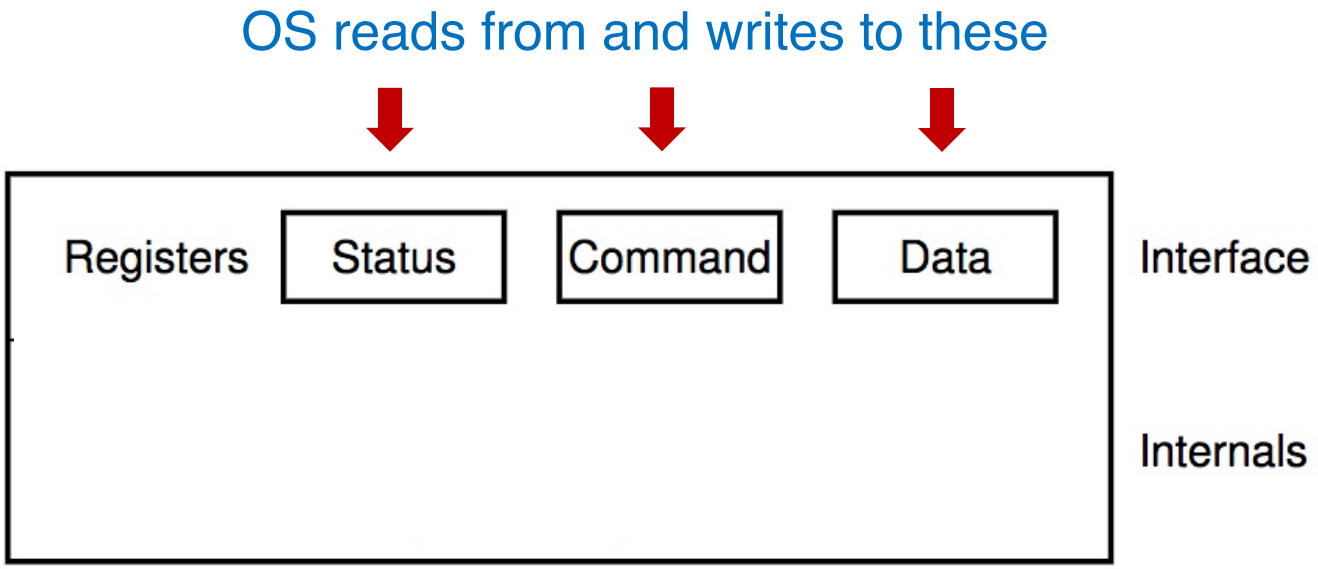
Prototypical System Architecture



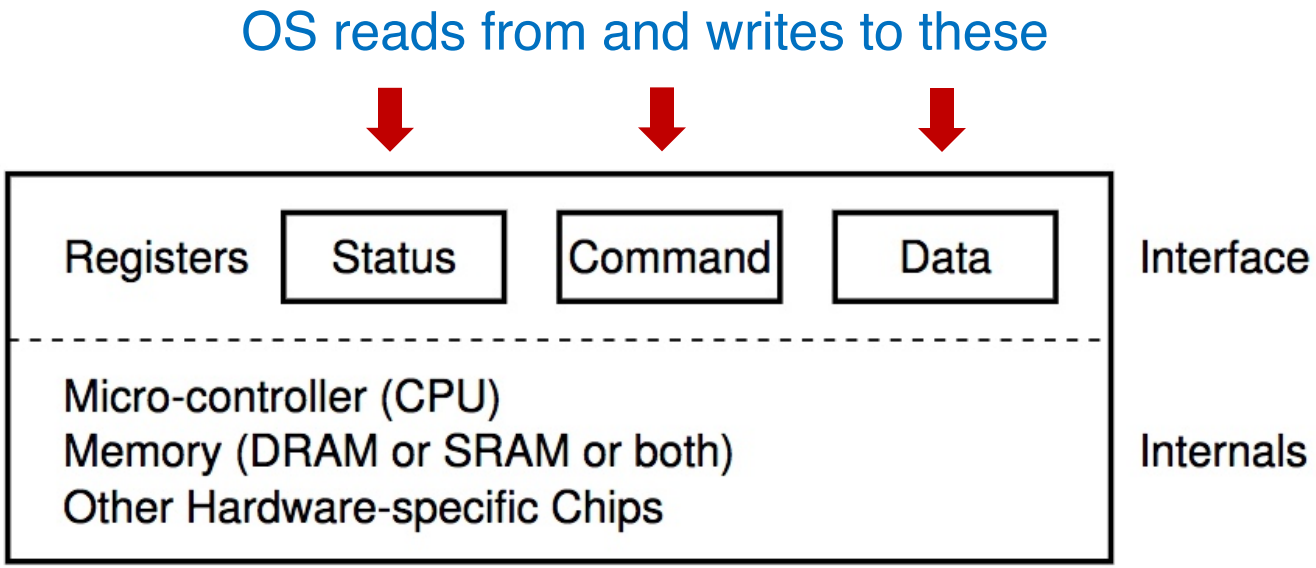
Canonical I/O Device



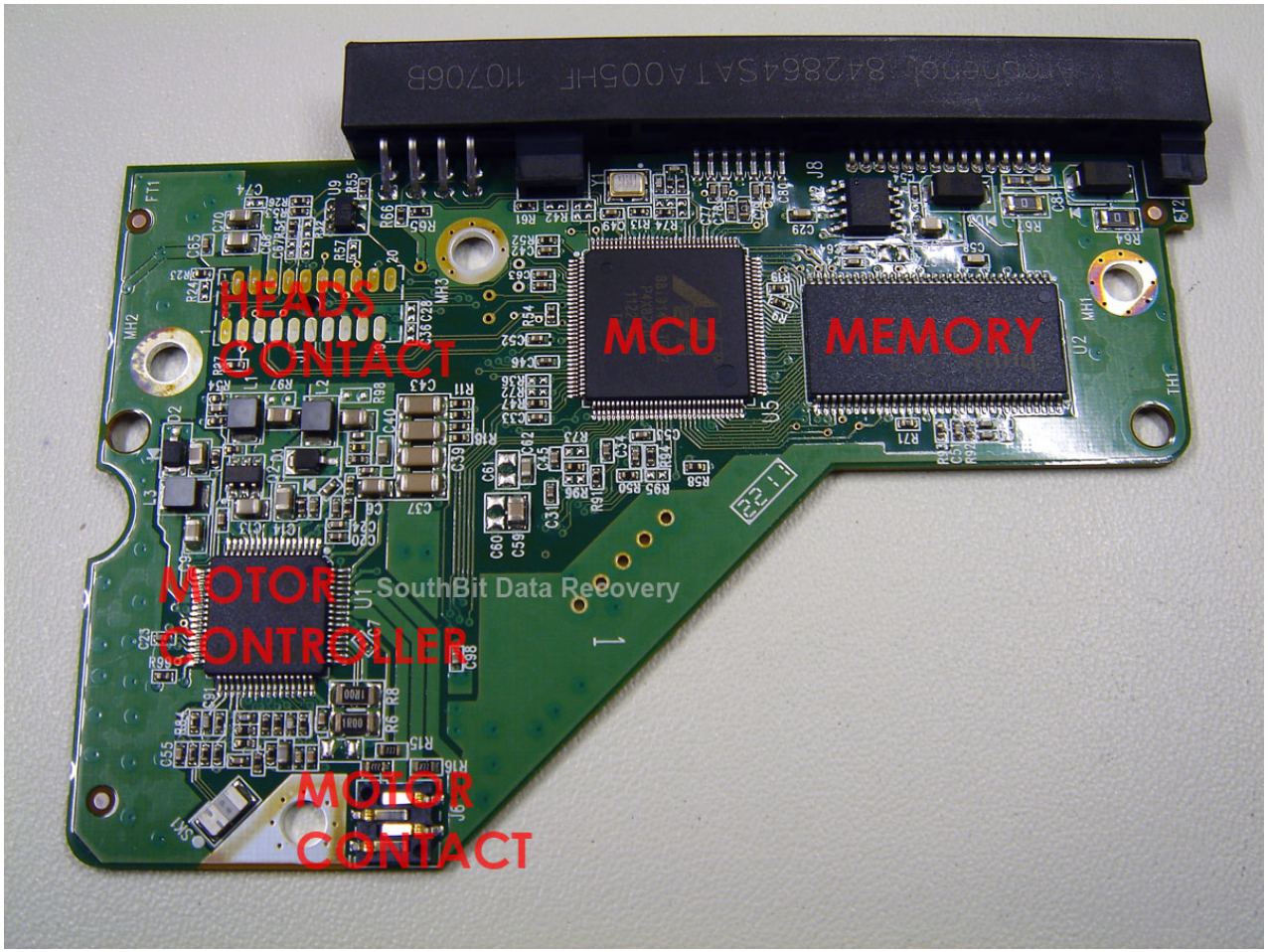
Canonical I/O Device



Canonical I/O Device



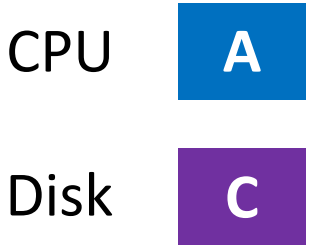
A Hard Disk Drive PCB Example



A Basic I/O Protocol

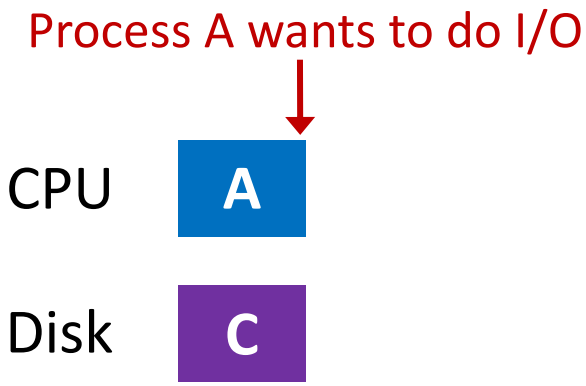
```
while (STATUS == BUSY)
      ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
      ; // spin
```

A Basic I/O Protocol



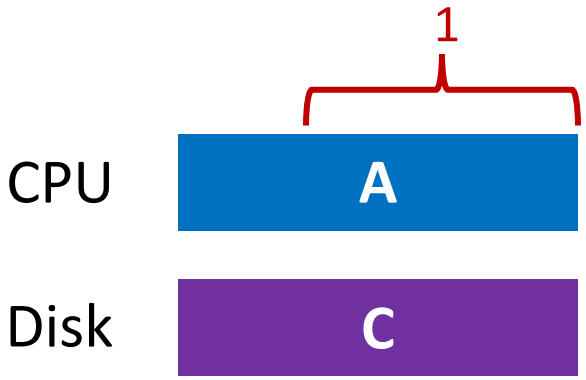

```
while (STATUS == BUSY)                    //1
      ; // spin
Write data to DATA register               //2
Write command to COMMAND register         //3
while (STATUS == BUSY)                    //4
      ; // spin
```

A Basic I/O Protocol



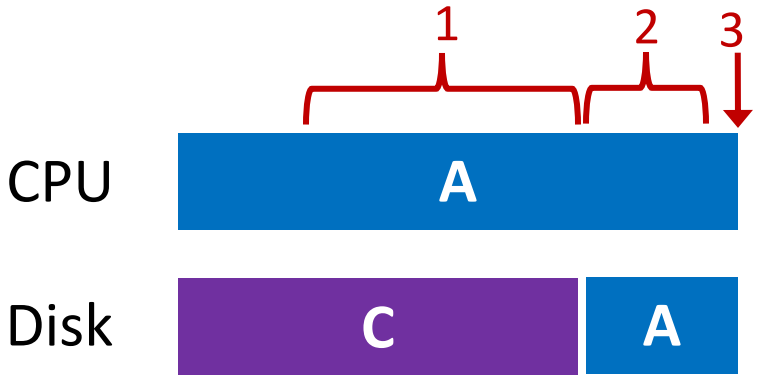

```
while (STATUS == BUSY)                    //1
      ; // spin
Write data to DATA register               //2
Write command to COMMAND register         //3
while (STATUS == BUSY)                    //4
      ; // spin
```

A Basic I/O Protocol



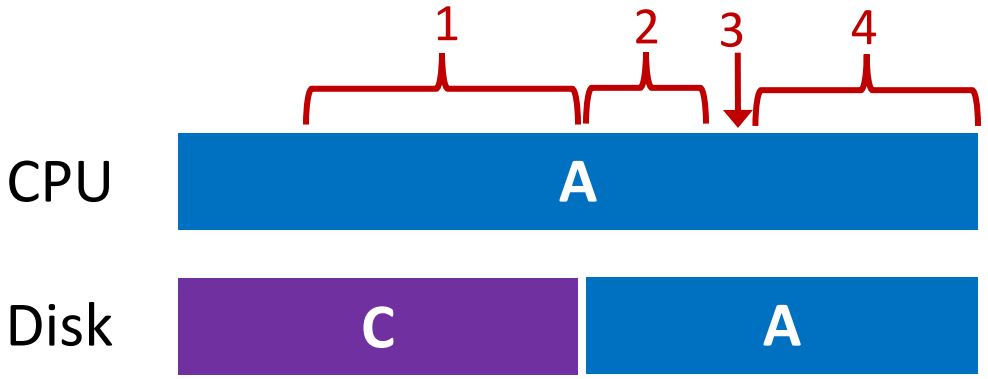

```
while (STATUS == BUSY)                    //1
      ; // spin
Write data to DATA register               //2
Write command to COMMAND register         //3
while (STATUS == BUSY)                    //4
      ; // spin
```

A Basic I/O Protocol



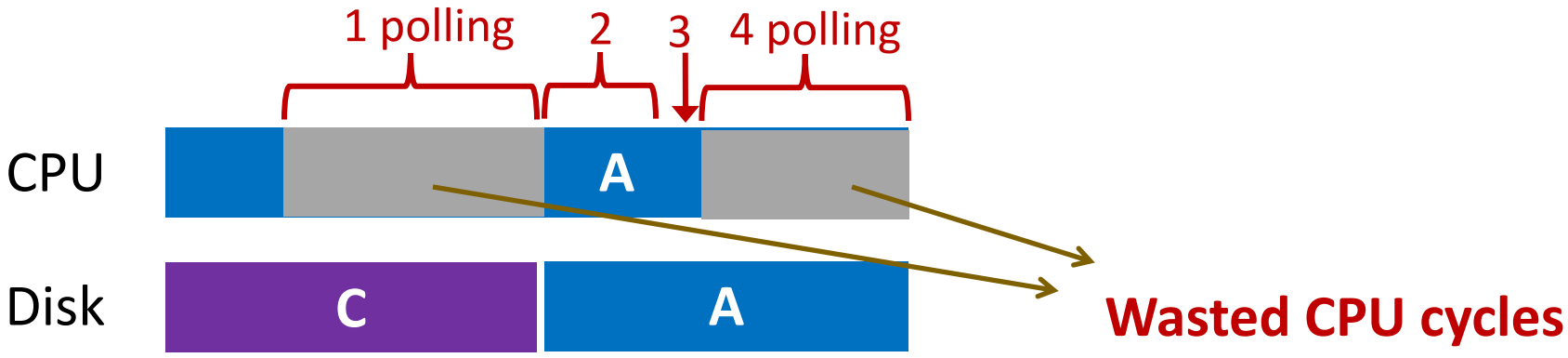

```
while (STATUS == BUSY)                    //1
      ; // spin
Write data to DATA register               //2
Write command to COMMAND register         //3
while (STATUS == BUSY)                    //4
      ; // spin
```

A Basic I/O Protocol



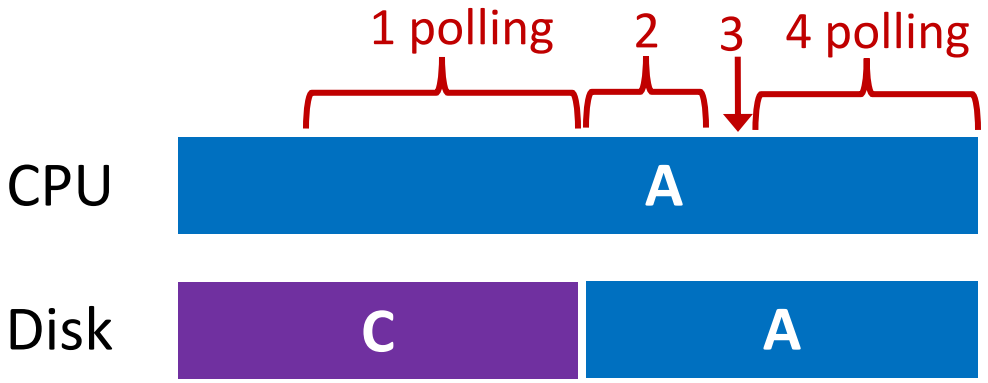

```
while (STATUS == BUSY)                    //1
      ; // spin
Write data to DATA register               //2
Write command to COMMAND register         //3
while (STATUS == BUSY)                    //4
      ; // spin
```

A Basic I/O Protocol



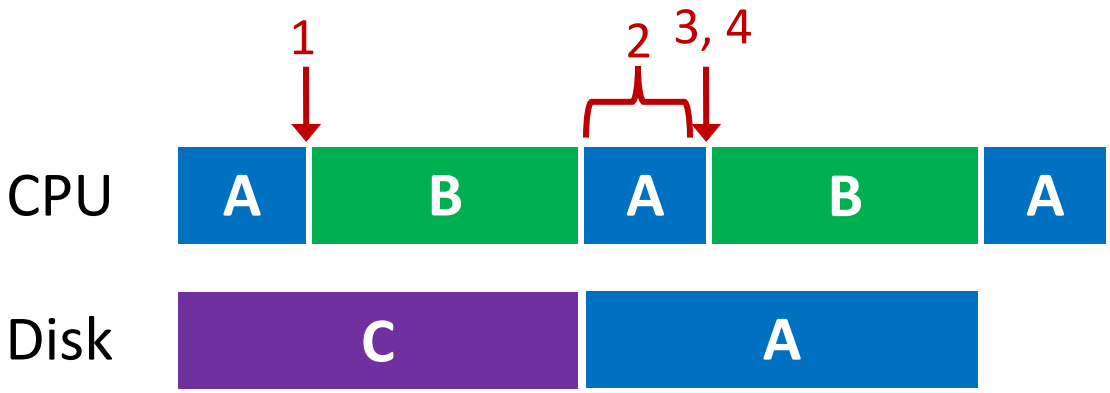

```
while (STATUS == BUSY)                    //1
      ; // spin
Write data to DATA register               //2
Write command to COMMAND register         //3
while (STATUS == BUSY)                    //4
      ; // spin
```

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

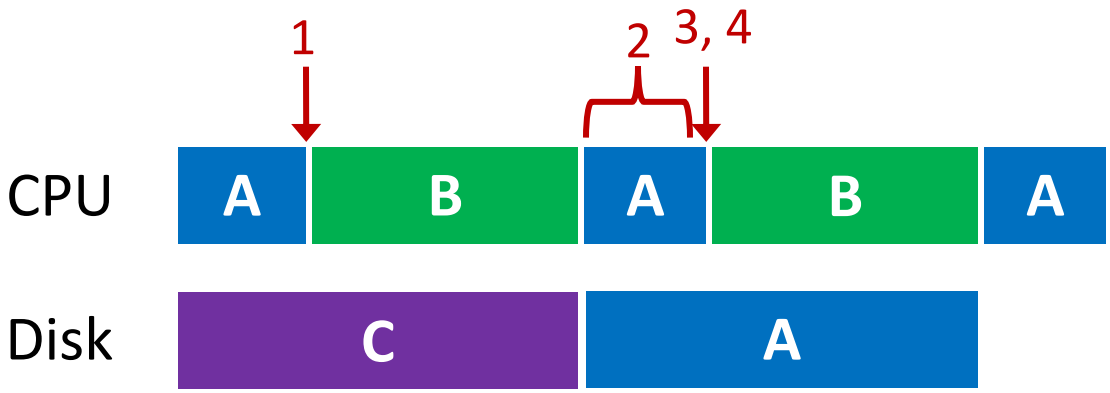
Interrupts vs. Polling

- Any potential issues for interrupts?

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

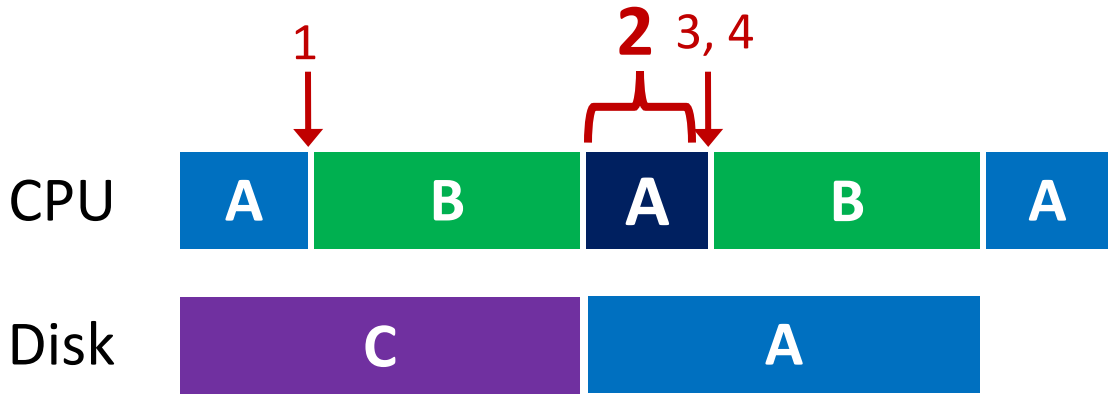
- Techniques
  - Hybrid approach: polling + interrupts
  - Interrupt coalescing: batching a bunch interrupts in one go

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;
```

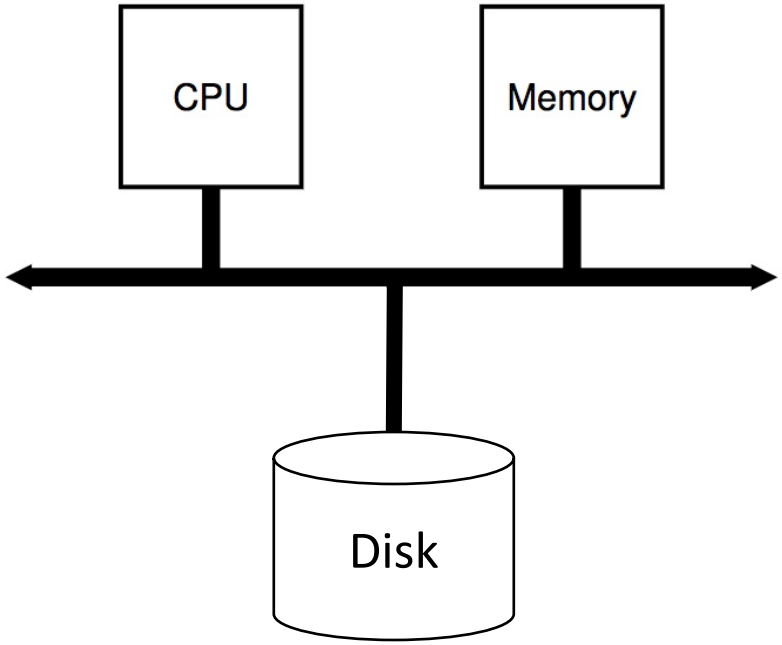
Programmed I/O vs. Direct Memory Access

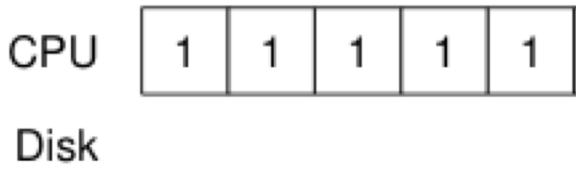
- PIO (Programmed I/O)
  - CPU directly tells device what data is
  - CPU involved in data transfer

- DMA (Direct Memory Access)
  - CPU leaves data in memory
  - DMA hardware does data copy

PIO Data Flow

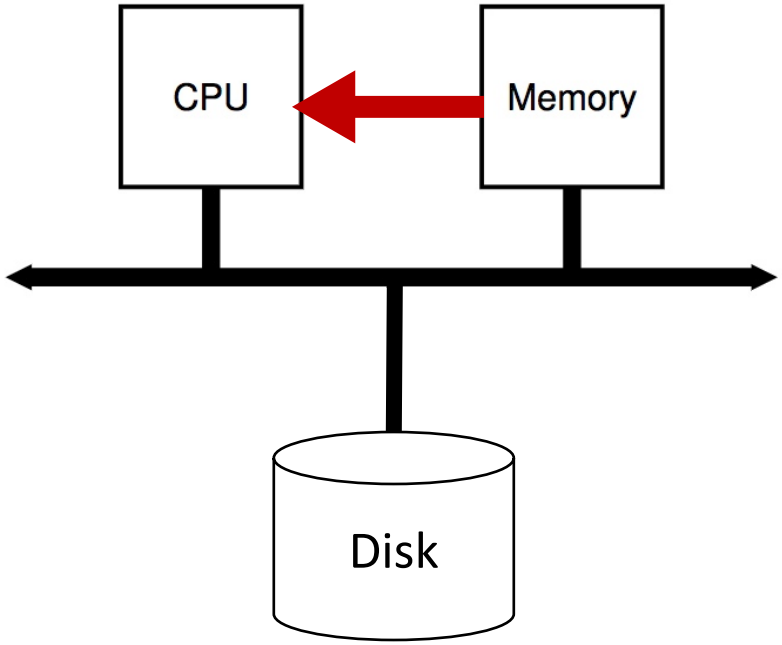
1. Executing P1 on CPU

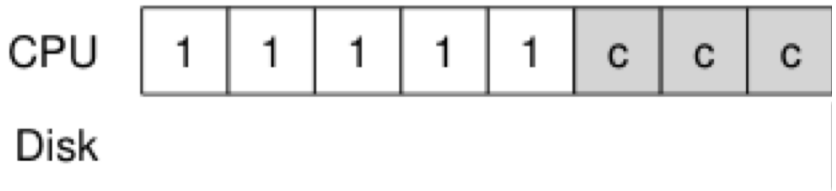




PIO Data Flow

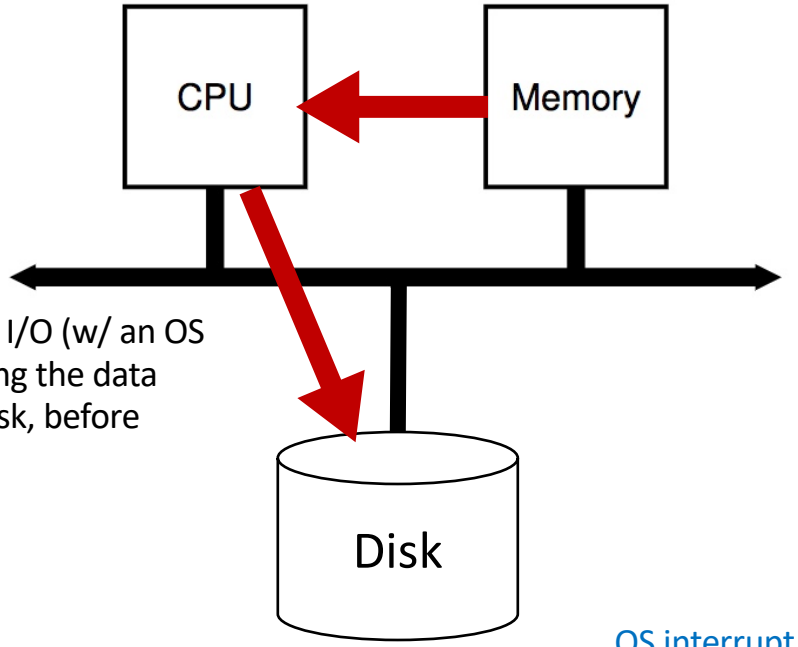
2. Copy data from memory via CPU



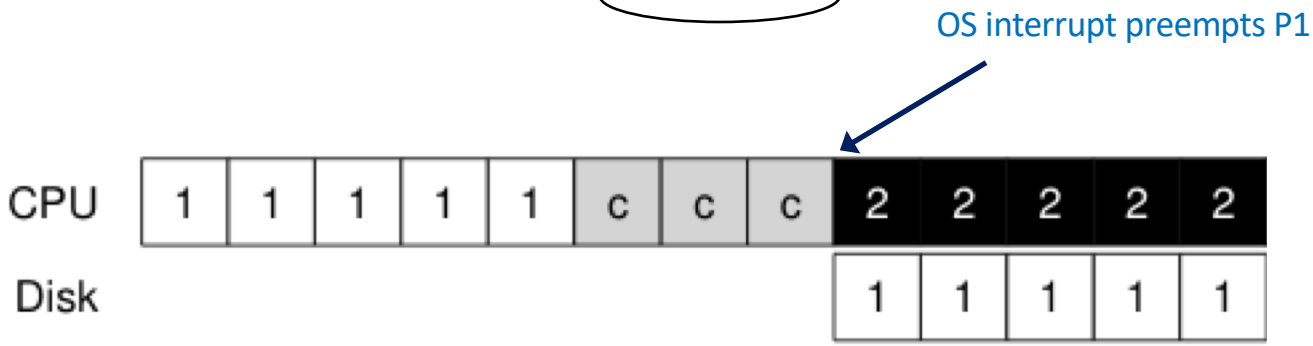


Note: c == copy memory words

PIO Data Flow

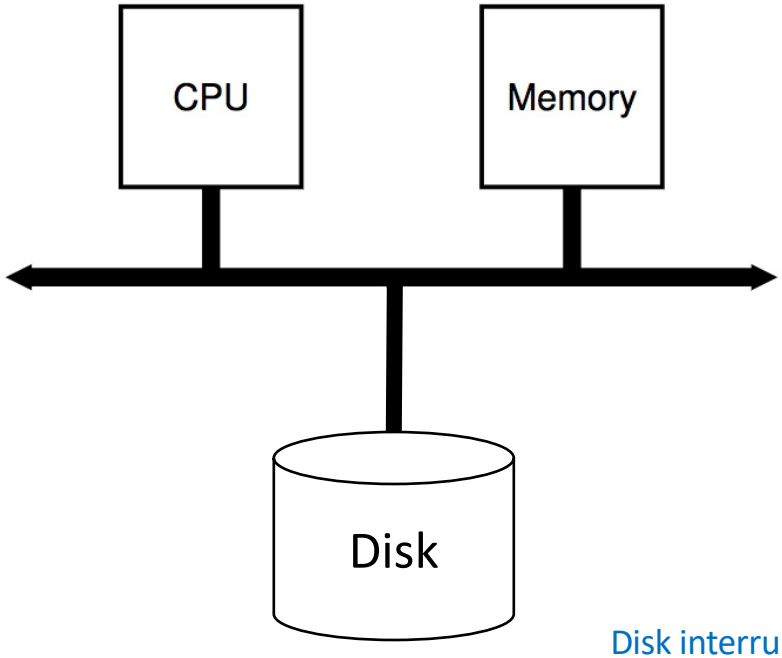


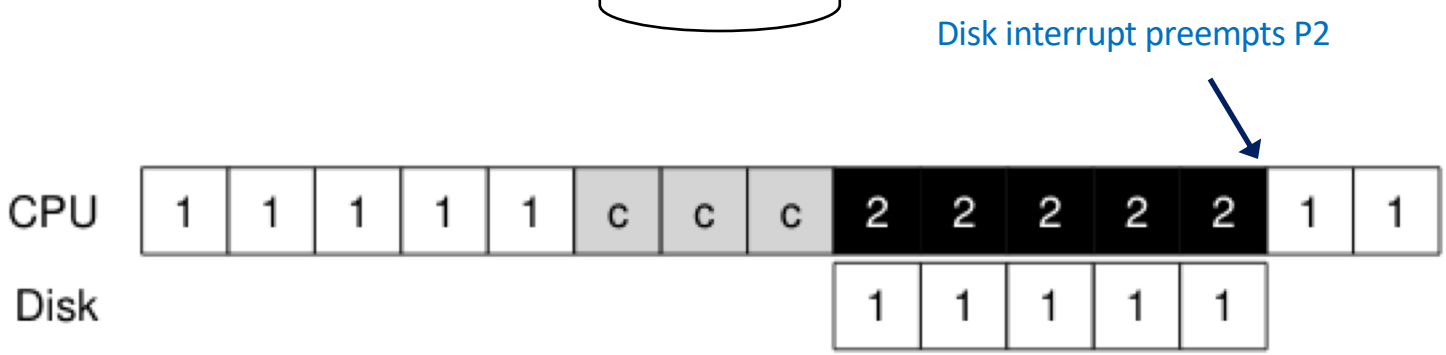
3. CPU initiates the I/O (w/ an OS interrupt) by copying the data from memory to disk, before running P2



PIO Data Flow

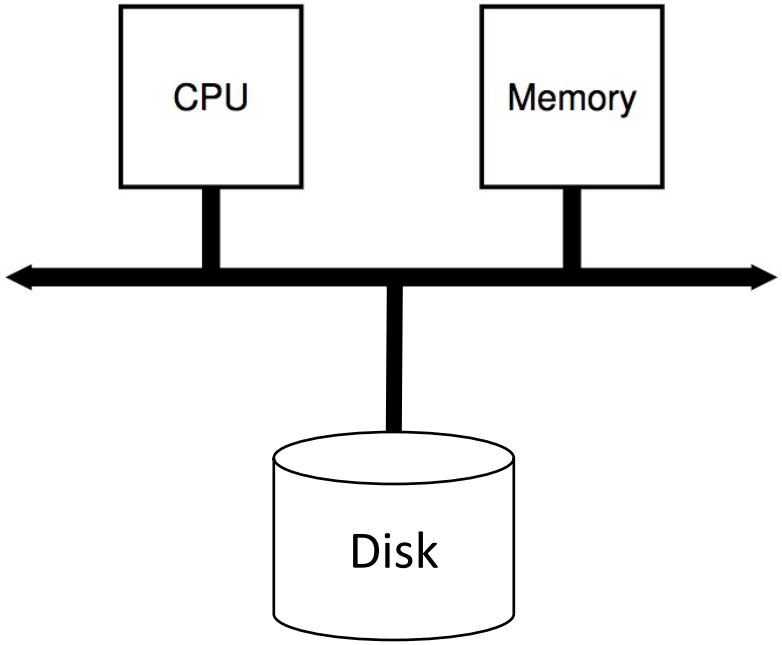
4. Done with I/O, Disk interrupts P2 and re-schedules P1 on CPU





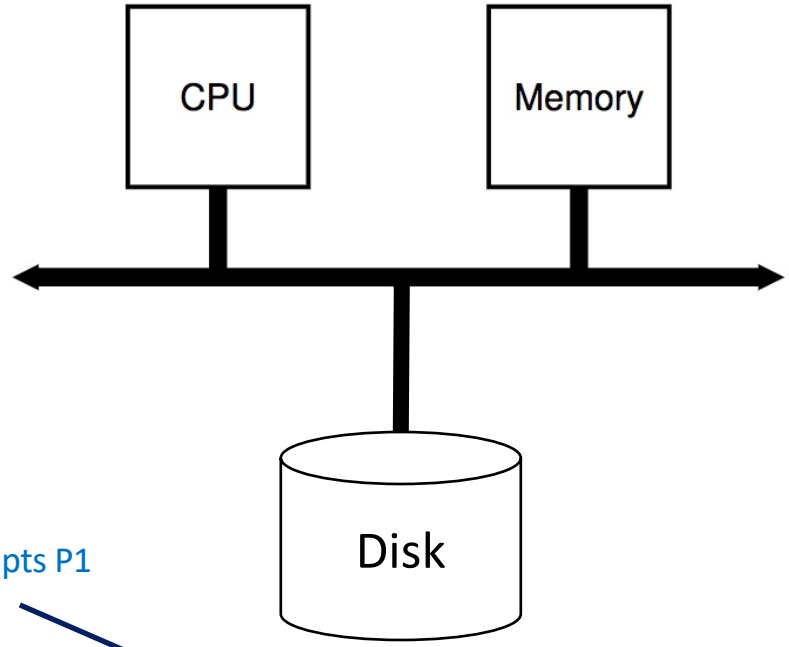
DMA Data Flow

1. Executing P1 on CPU

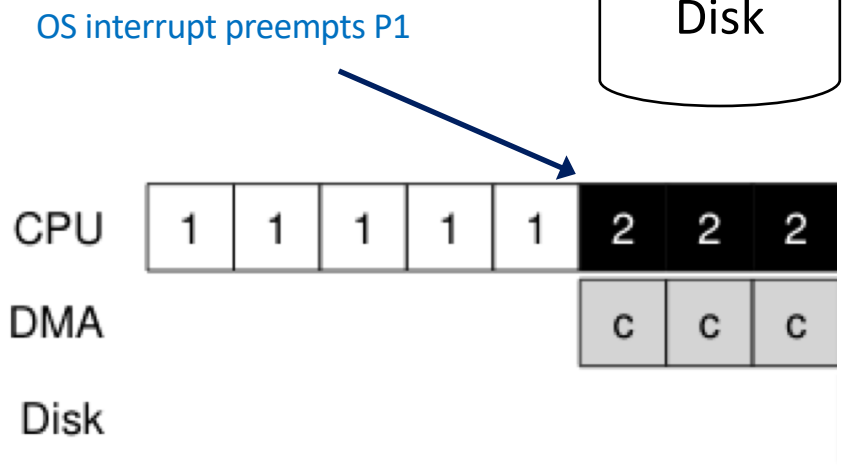


CPU	1	1	1	1	1
DMA					
Disk					

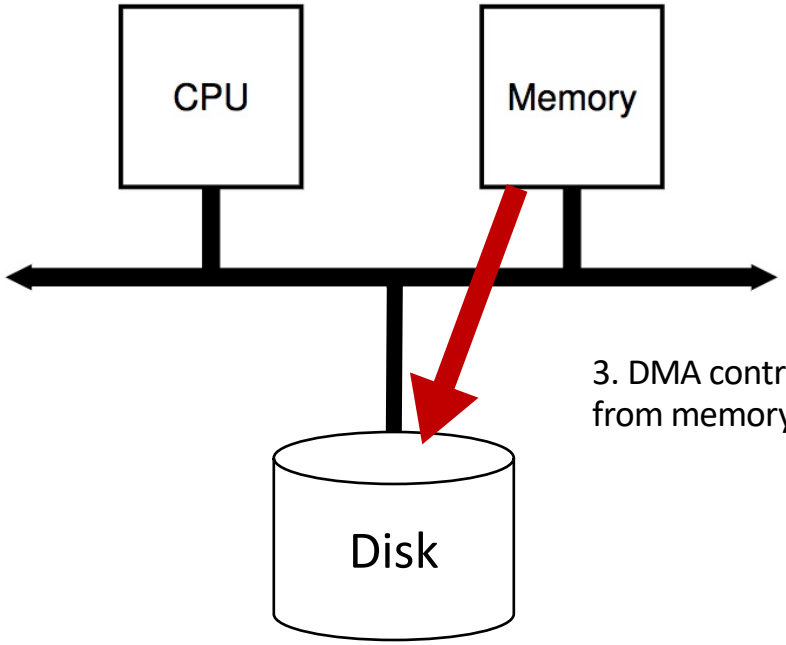
DMA Data Flow



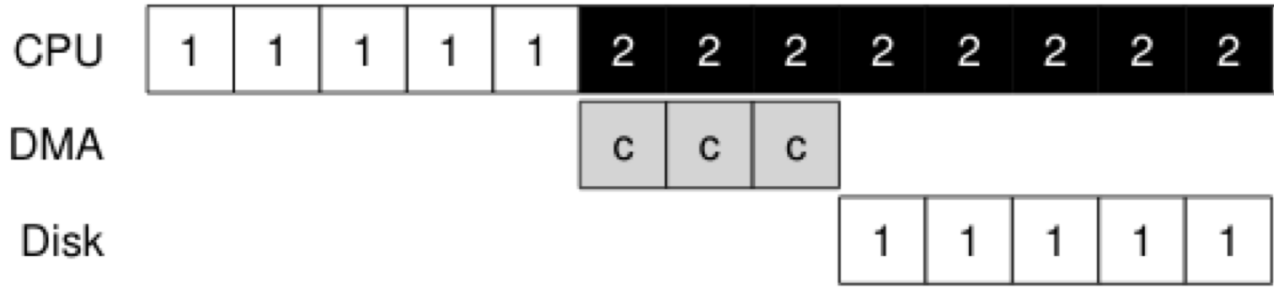
2a. OS initiates DMA by telling the DMA engine where data lives in memory, how much to copy, and which device to send it to;
2b. DMA then copies the data from memory



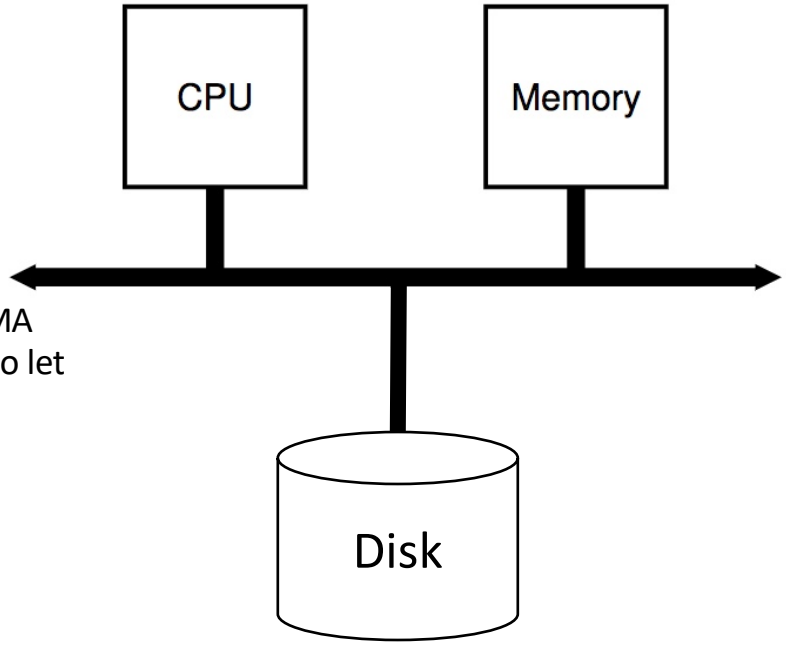
DMA Data Flow



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

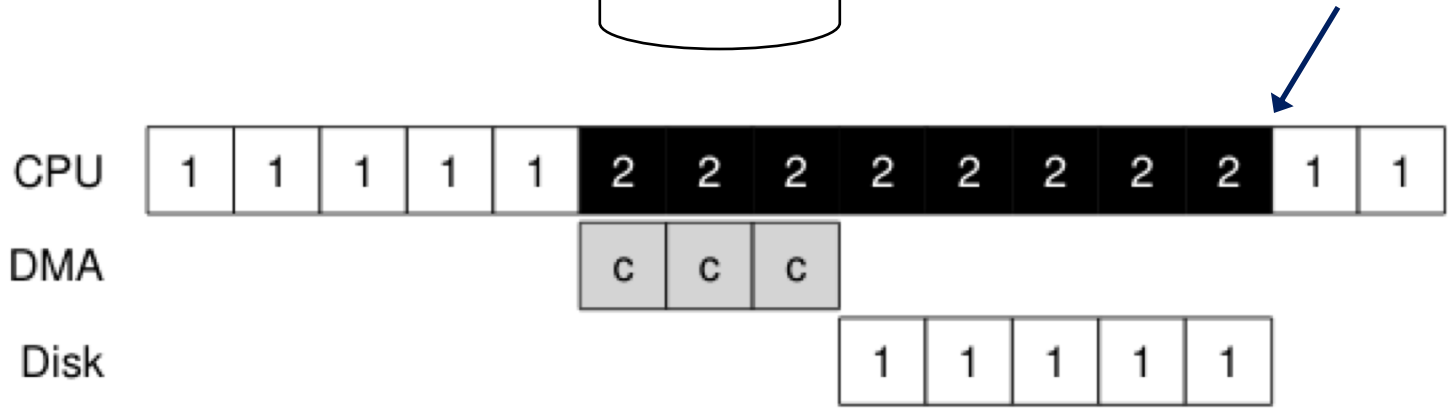


DMA Data Flow

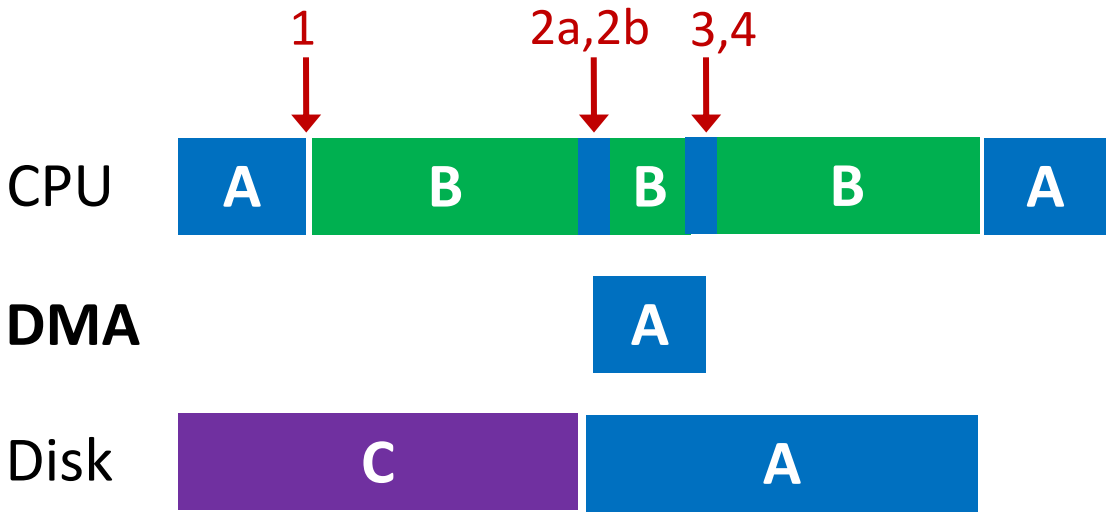


4. When DMA is complete, DMA controller raises an interrupt to let OS know P1 can resume

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;
```