[537] I/O Devices

Chapter 35 - 37
Tyler Harter
10/27/14

I/O Devices

Motivation

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

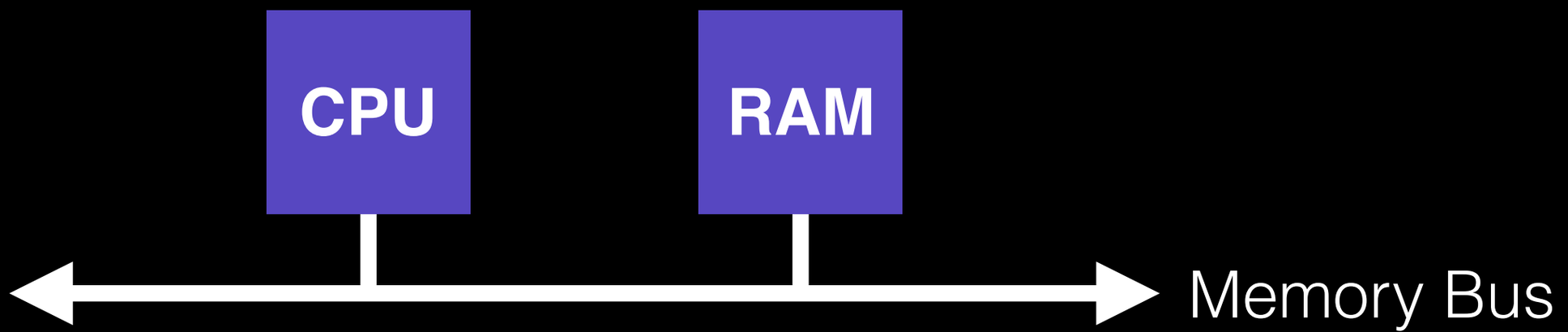
We want:
- **H/W** that will let us plug in different devices
- **OS** that can interact with different combinations

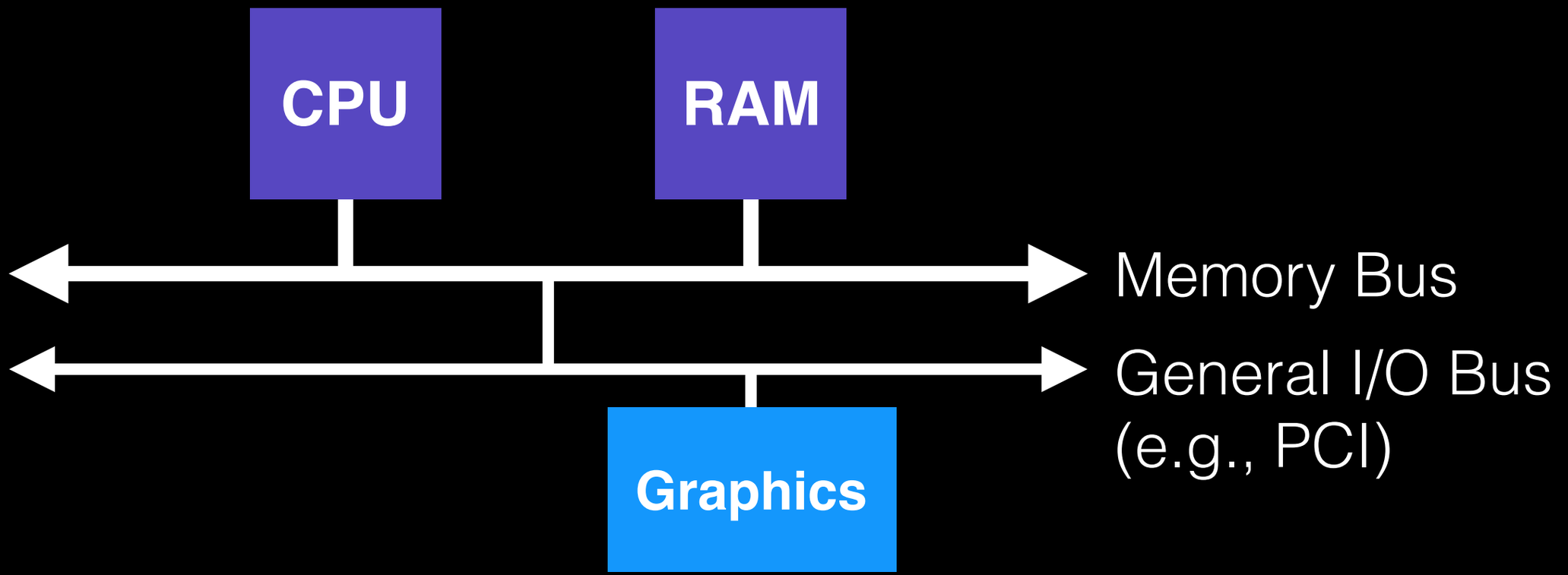
Motivation

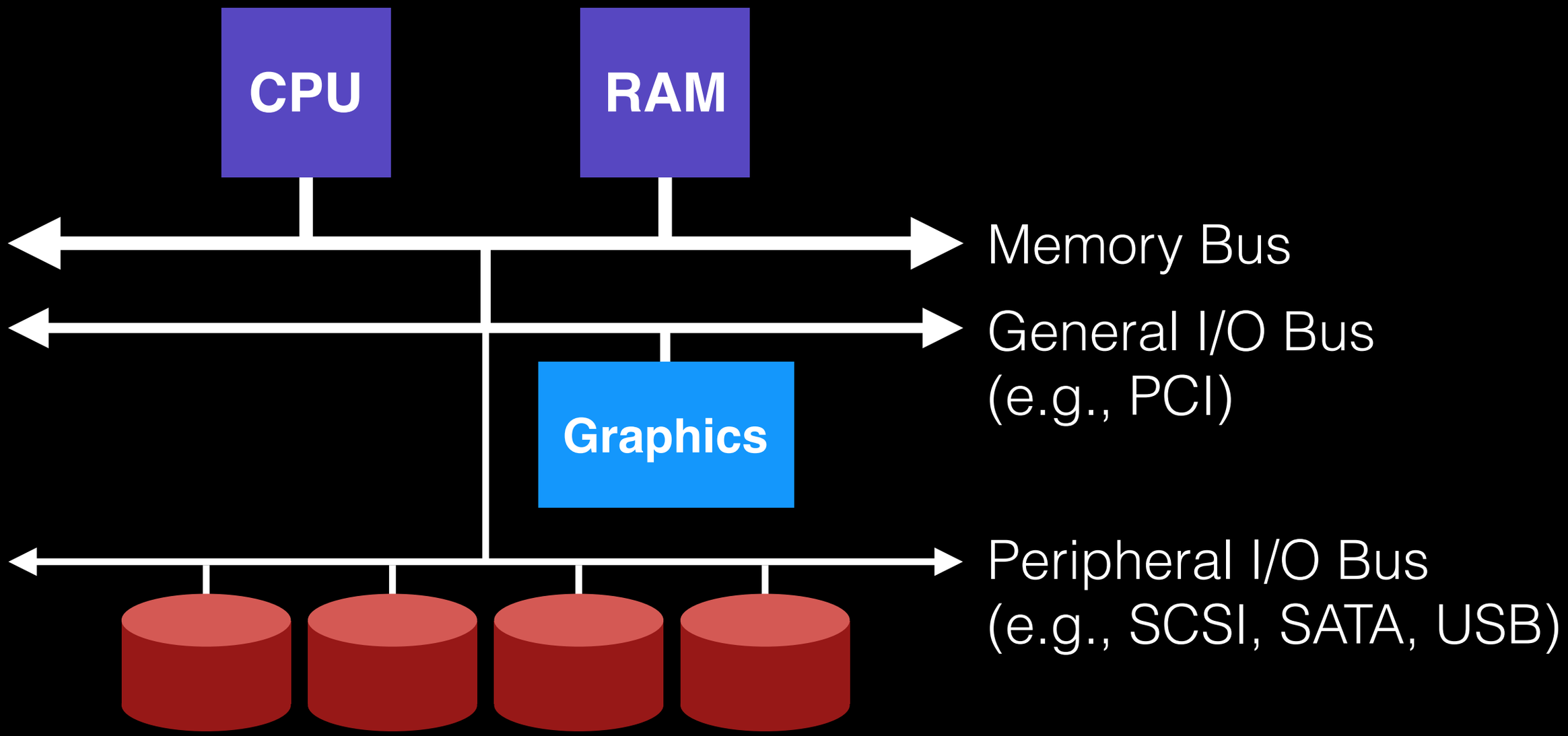
What good is a computer without any I/O devices?
- keyboard, display, disks

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

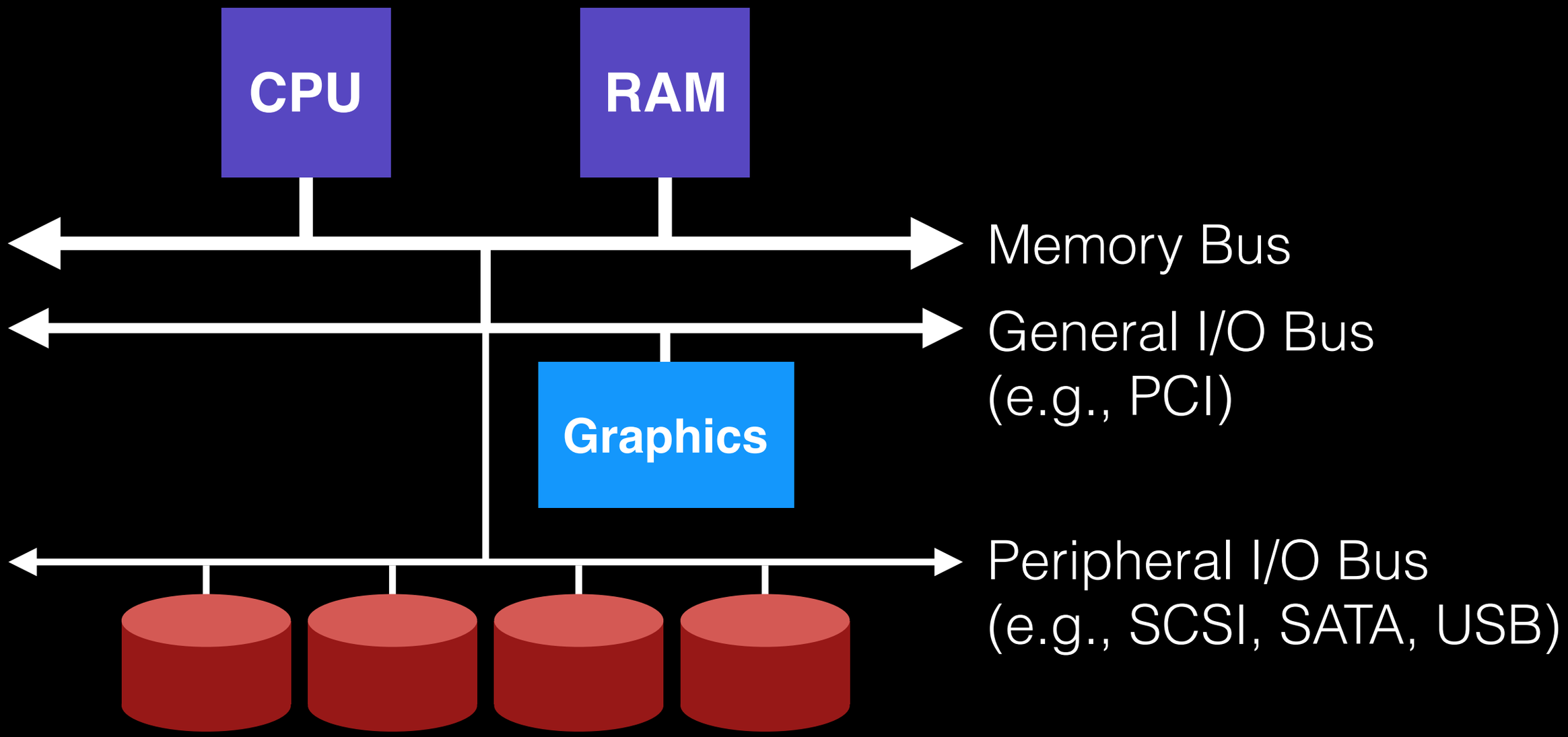
Largely a communication problem…


CPU
RAM
Memory Bus

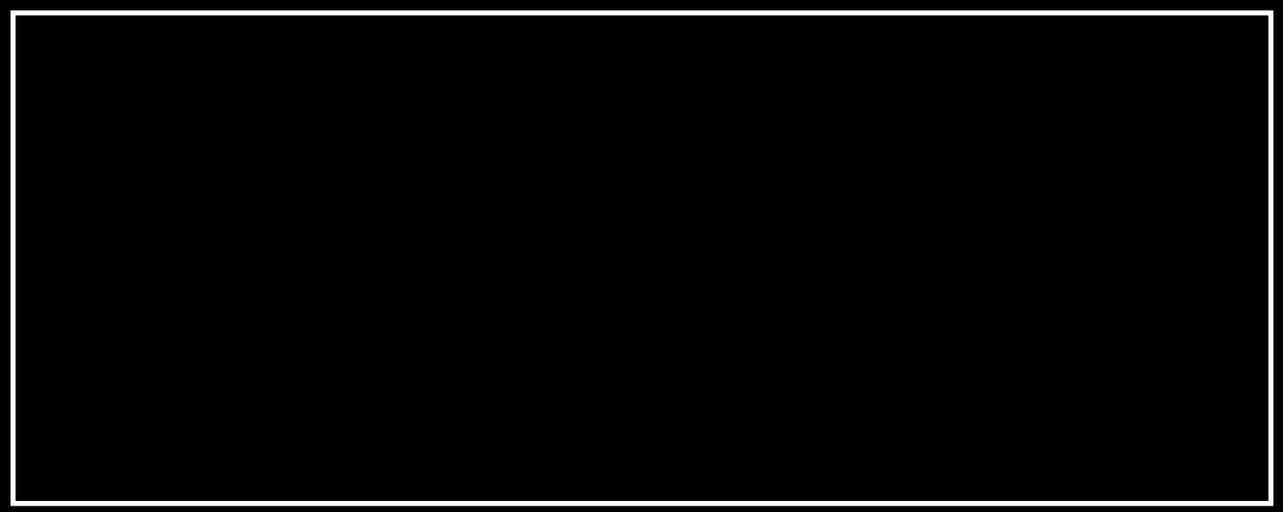

CPU
RAM
Memory Bus
General I/O Bus
(e.g., PCI)
Graphics


CPU
RAM
Memory Bus
General I/O Bus
(e.g., PCI)
Graphics
Peripheral I/O Bus
(e.g., SCSI, SATA, USB)

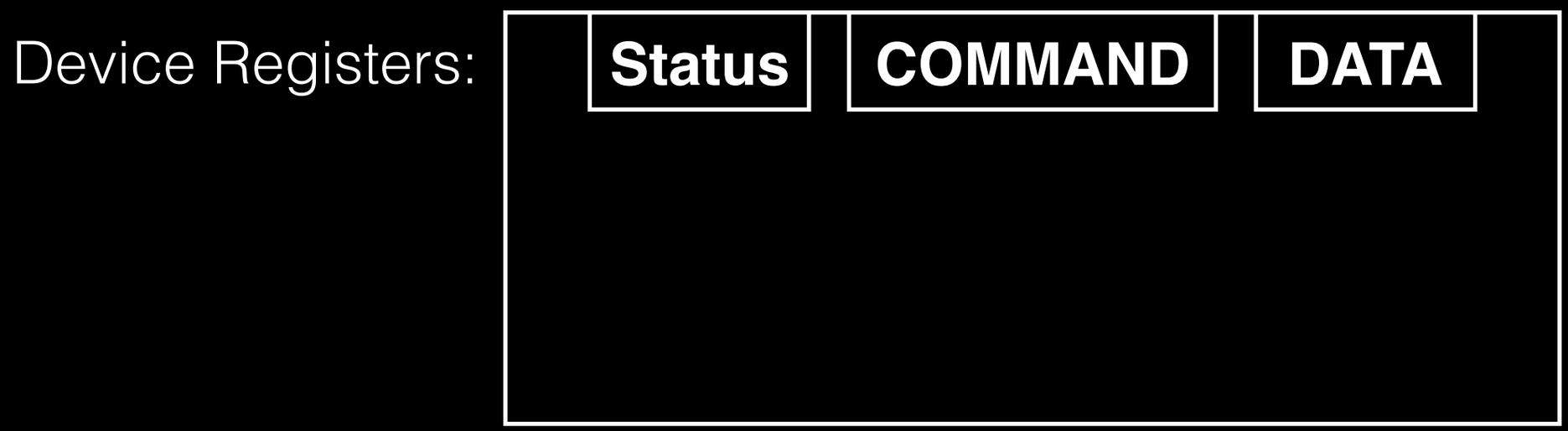
Why use hierarchical buses?



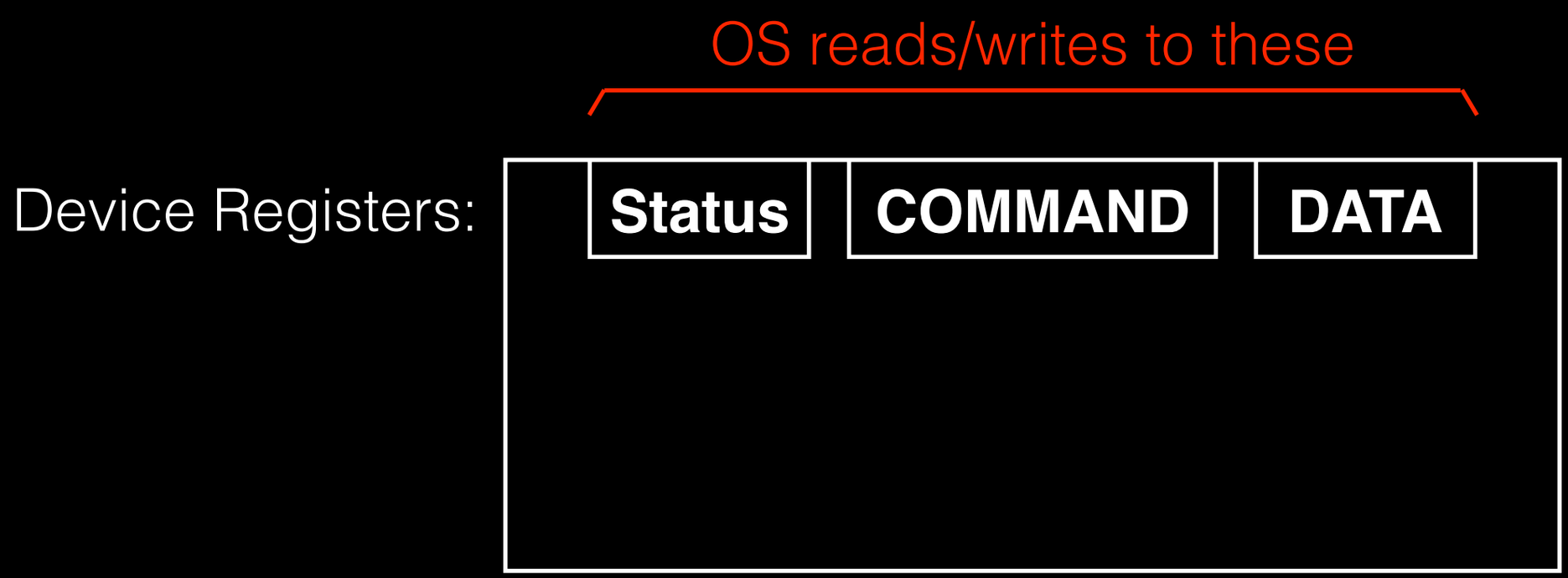
Canonical Device… Is a rectangle!



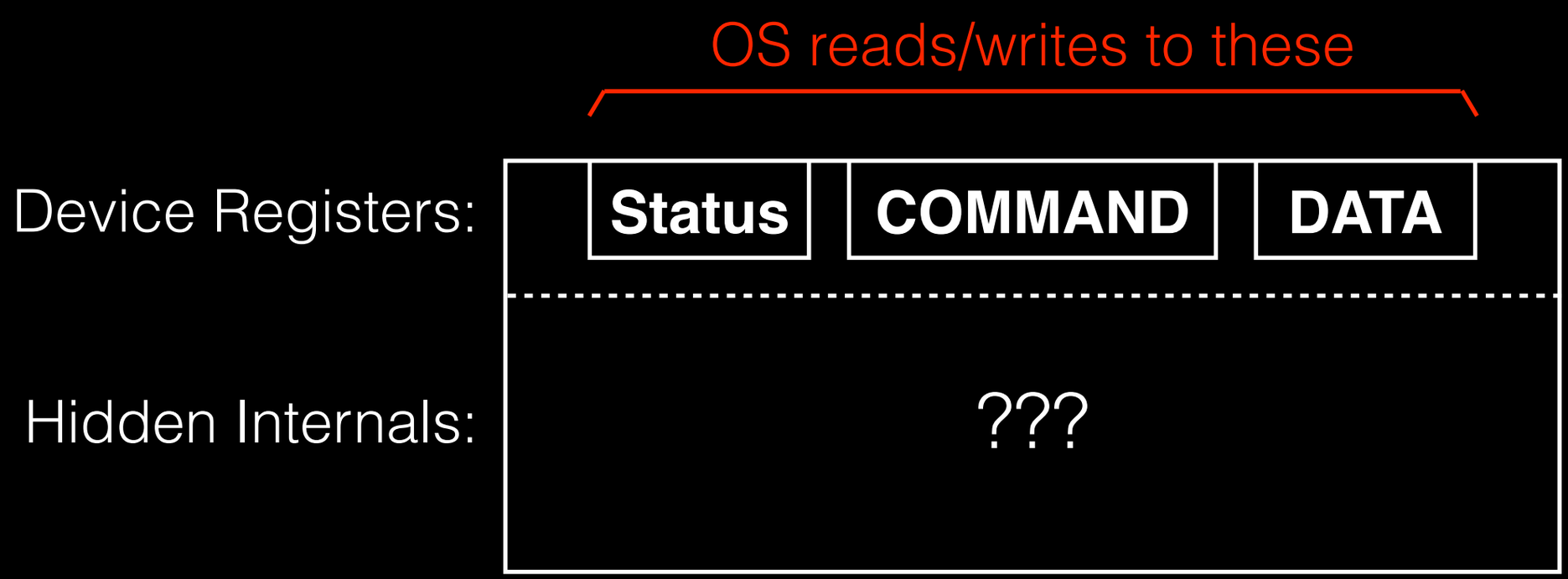
Canonical Device



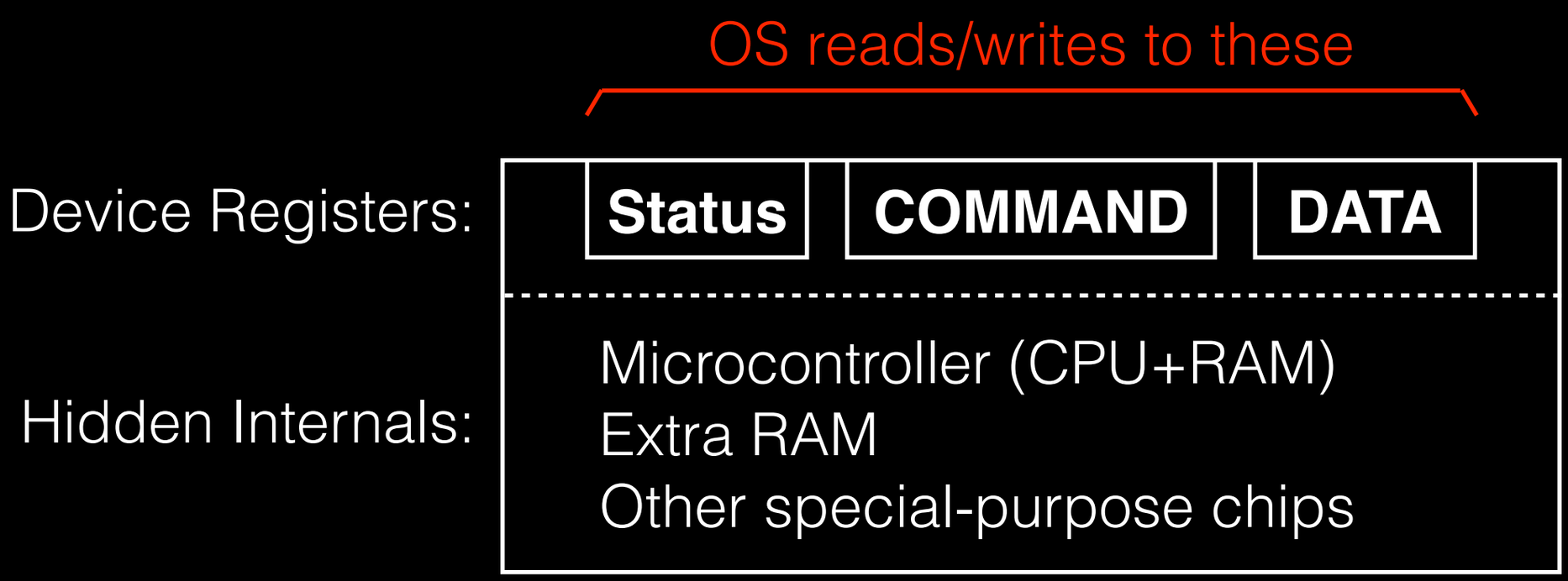
Canonical Device



Canonical Device



Canonical Device



Example Protocol

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

CPU:

Disk:

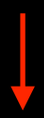
```
   while (STATUS == BUSY)                  // 1
      ;
   Write data to DATA register             // 2
   Write command to COMMAND register       // 3
   while (STATUS == BUSY)                  // 4
      ;
```

CPU: **A**

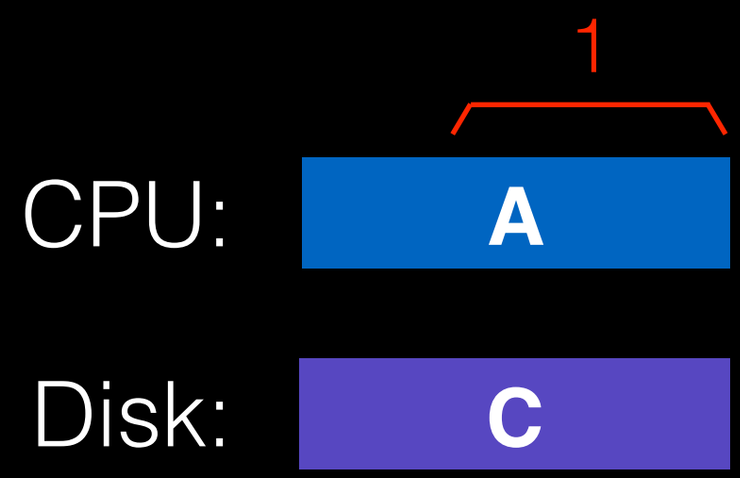
Disk: **C**

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

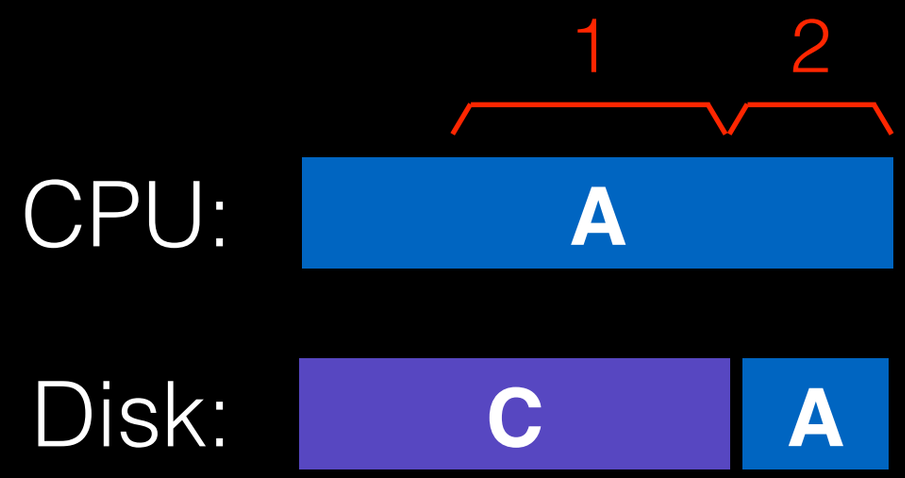
A wants to do I/O



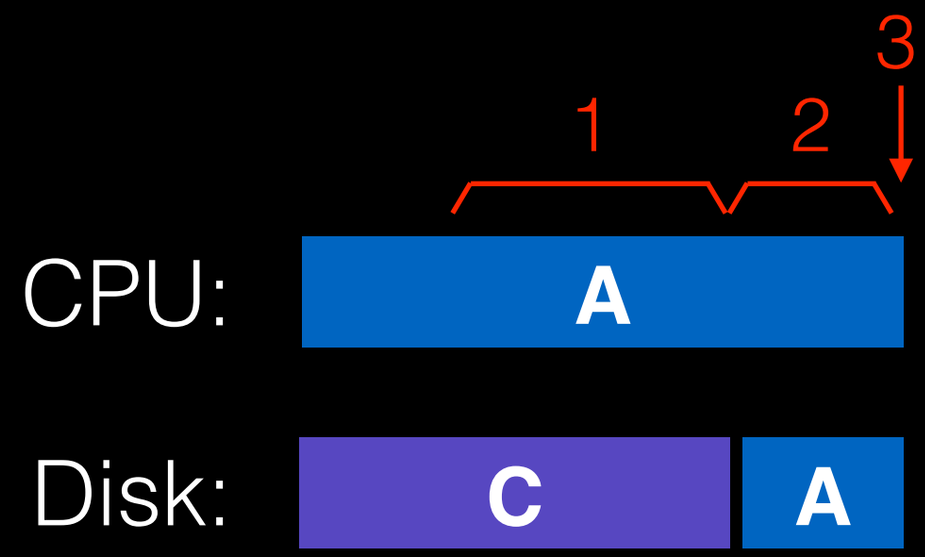

```
while (STATUS == BUSY)                  // 1
   ;
Write data to DATA register             // 2
Write command to COMMAND register       // 3
while (STATUS == BUSY)                  // 4
   ;
```



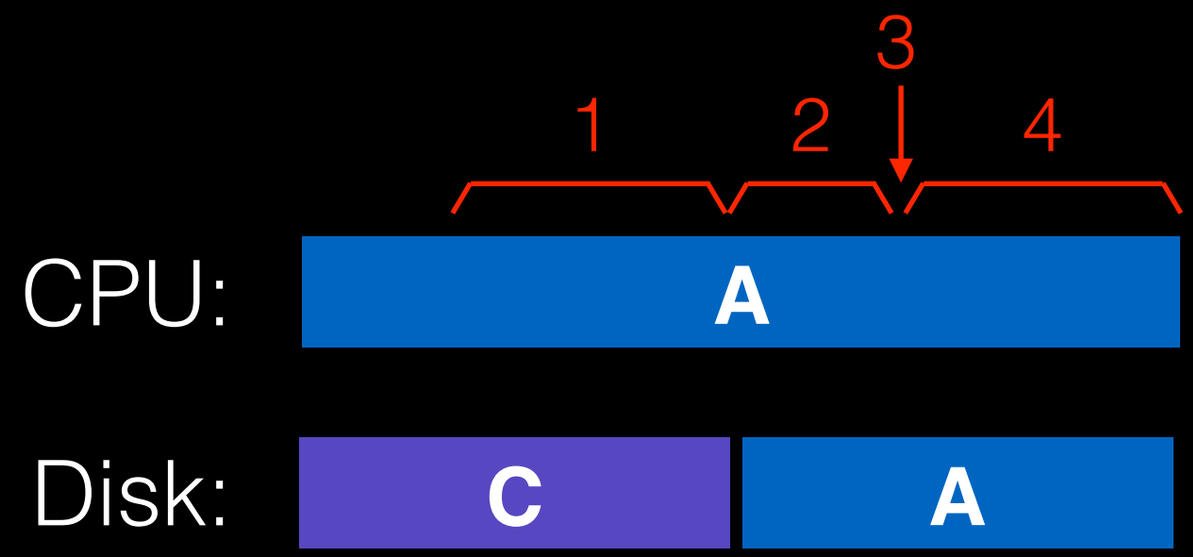

```
while (STATUS == BUSY)                    // 1
   ;
Write data to DATA register               // 2
Write command to COMMAND register         // 3
while (STATUS == BUSY)                    // 4
   ;
```



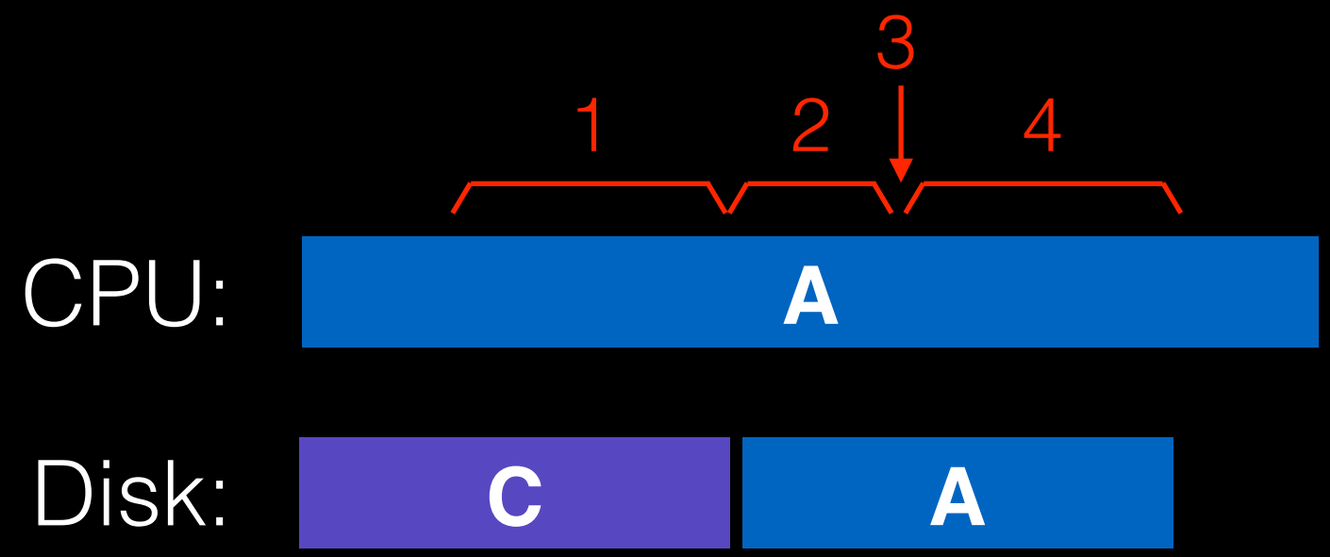

```
while (STATUS == BUSY)                  // 1
   ;
Write data to DATA register              // 2
Write command to COMMAND register        // 3
while (STATUS == BUSY)                   // 4
   ;
```



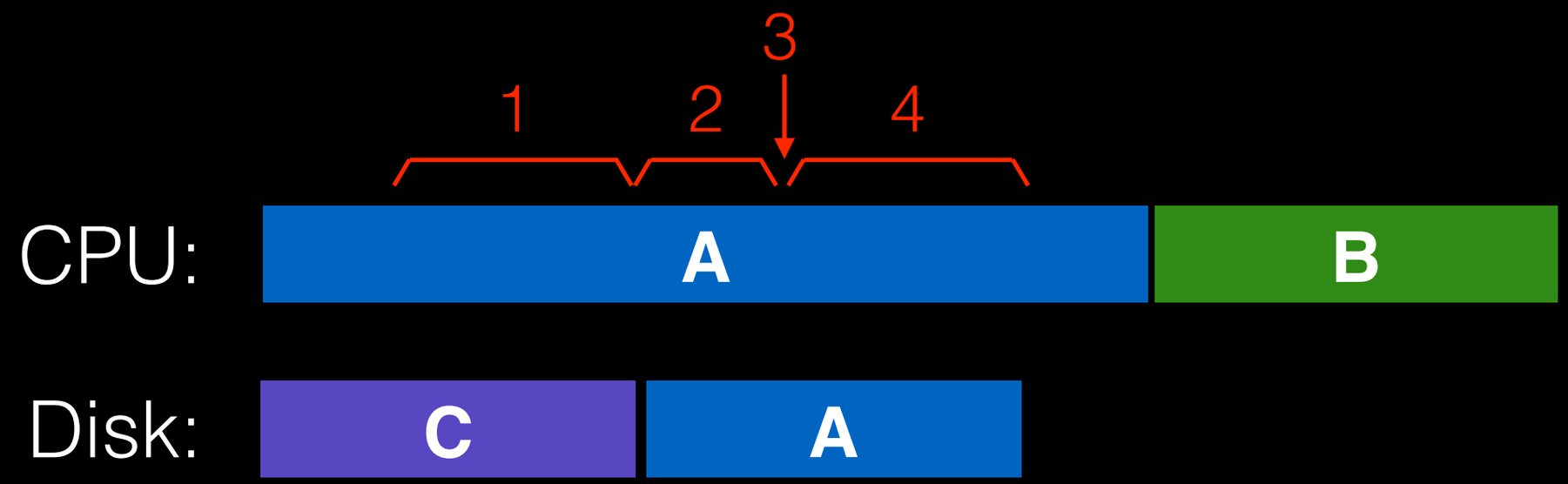

```
while (STATUS == BUSY)                    // 1
   ;
Write data to DATA register               // 2
Write command to COMMAND register         // 3
while (STATUS == BUSY)                    // 4
   ;
```




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

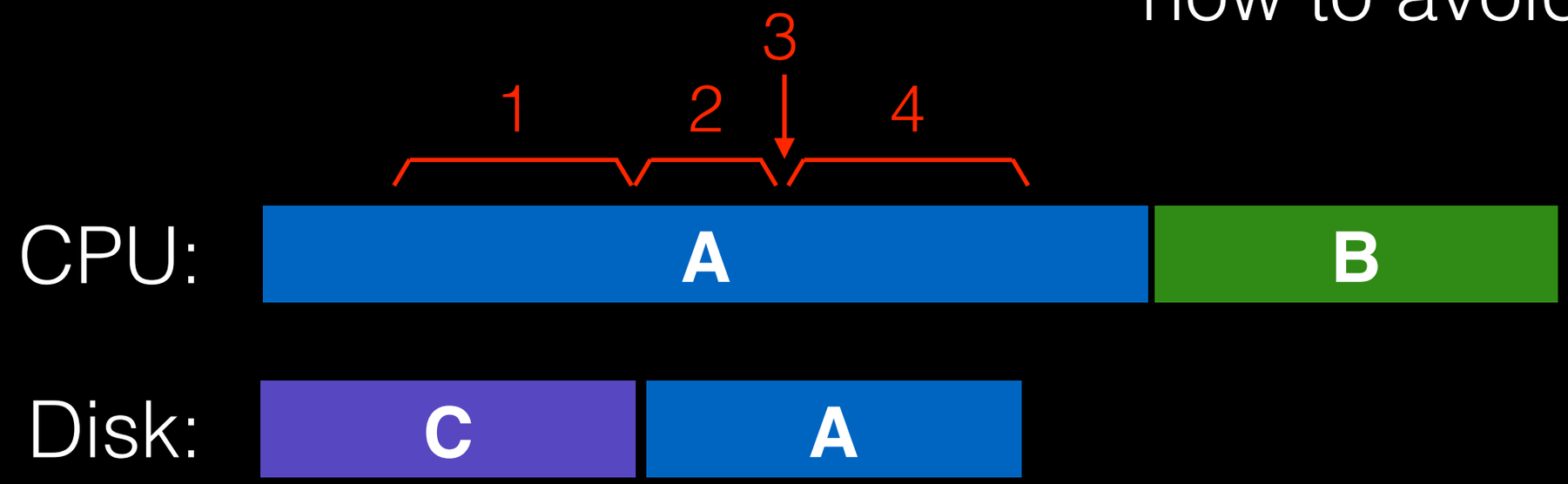



```
while (STATUS == BUSY)                  // 1
   ;
Write data to DATA register              // 2
Write command to COMMAND register        // 3
while (STATUS == BUSY)                   // 4
   ;
```



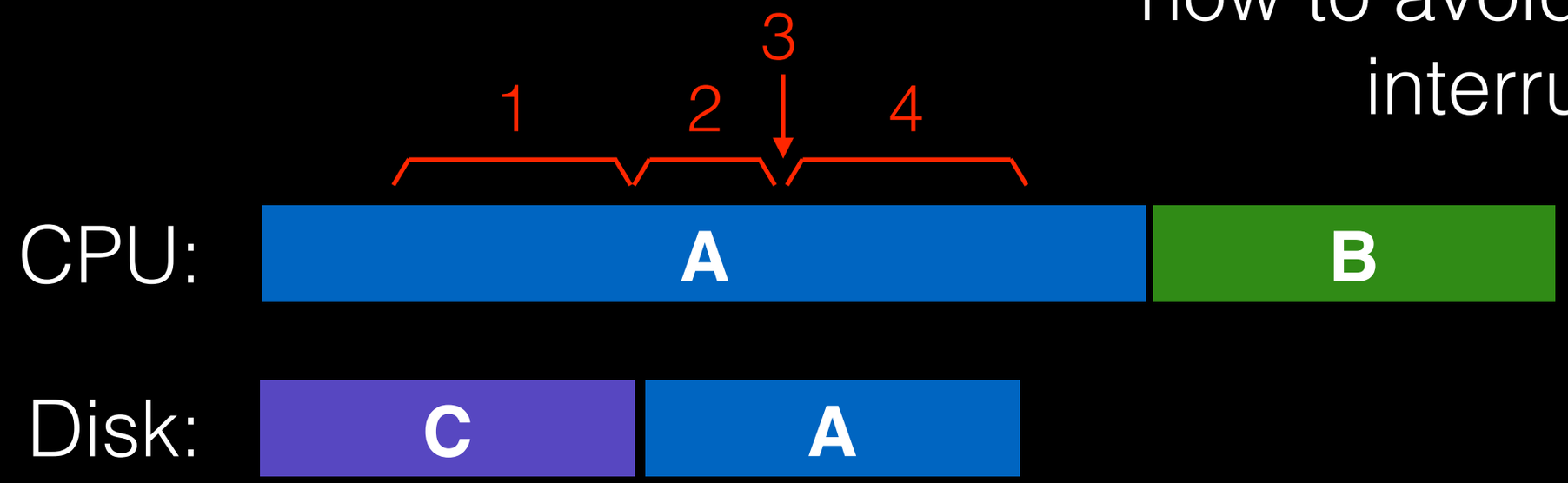

```
while (STATUS == BUSY)                  // 1
   ;
Write data to DATA register              // 2
Write command to COMMAND register        // 3
while (STATUS == BUSY)                   // 4
   ;
```

how to avoid spinning?



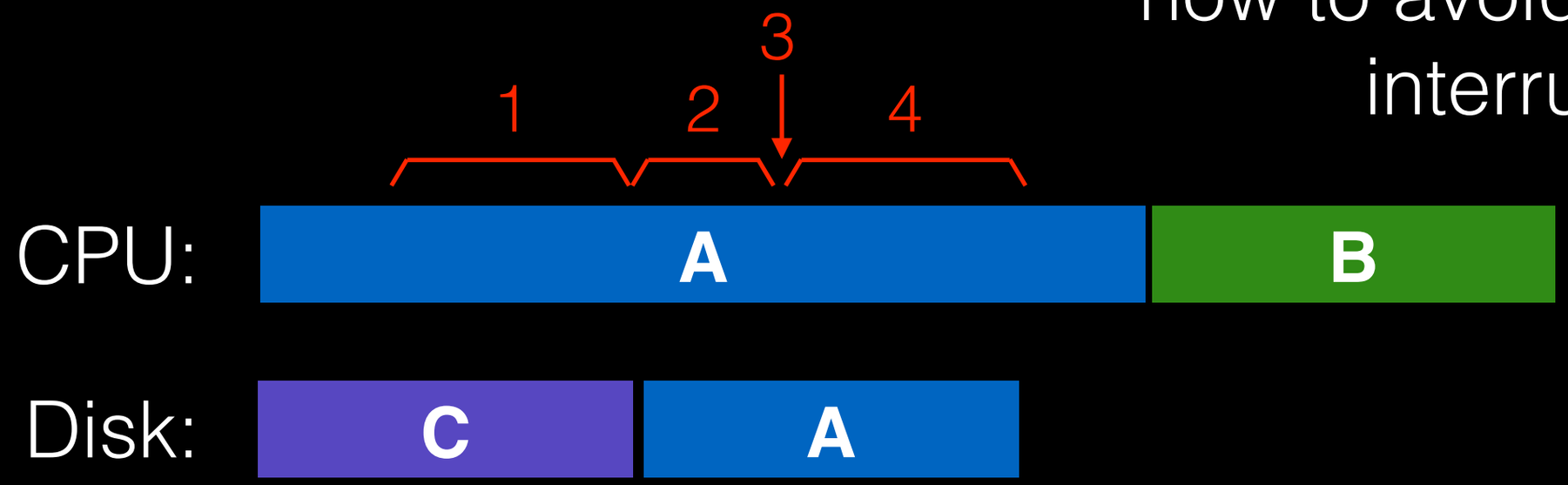

```
while (STATUS == BUSY)                    // 1
   ;
Write data to DATA register               // 2
Write command to COMMAND register         // 3
while (STATUS == BUSY)                    // 4
   ;
```

how to avoid spinning?
interrupts!



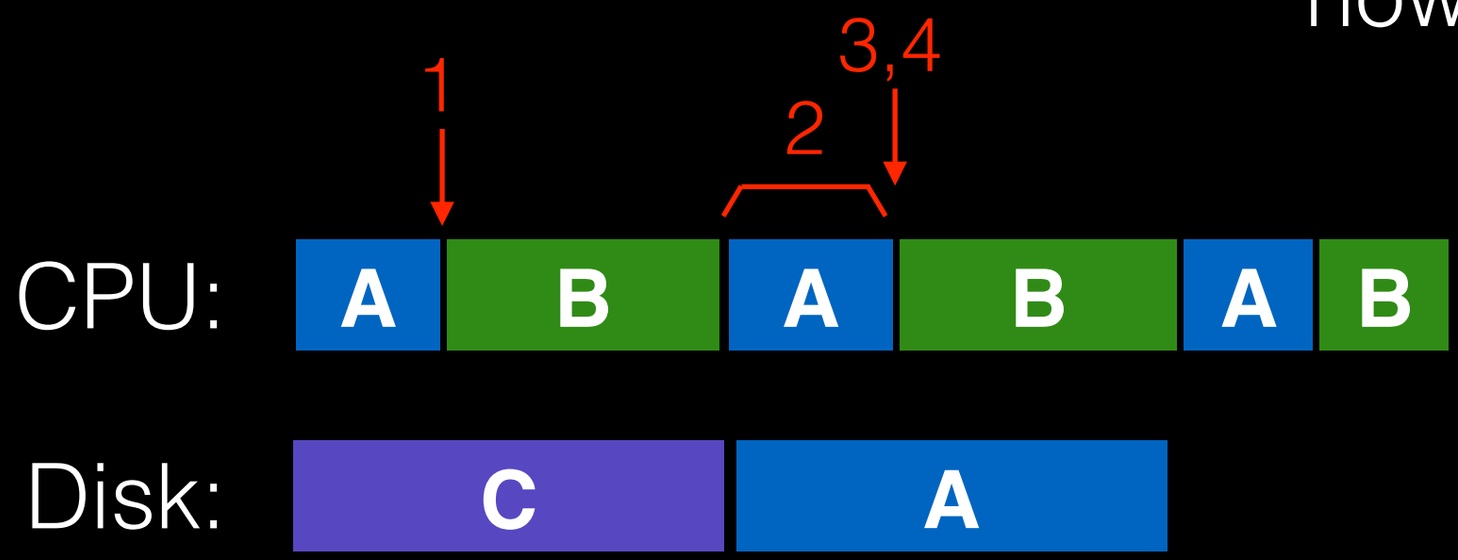

```
while (STATUS == BUSY)                    // 1
   ;
Write data to DATA register               // 2
Write command to COMMAND register         // 3
while (STATUS == BUSY)                    // 4
   ;
```

how to avoid spinning?
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;
```

how to avoid spinning? 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

Discuss: are interrupts ever worse?

Interrupts vs. Polling

Discuss: are interrupts ever worse?

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

Interrupts vs. Polling

Discuss: are interrupts ever worse?

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

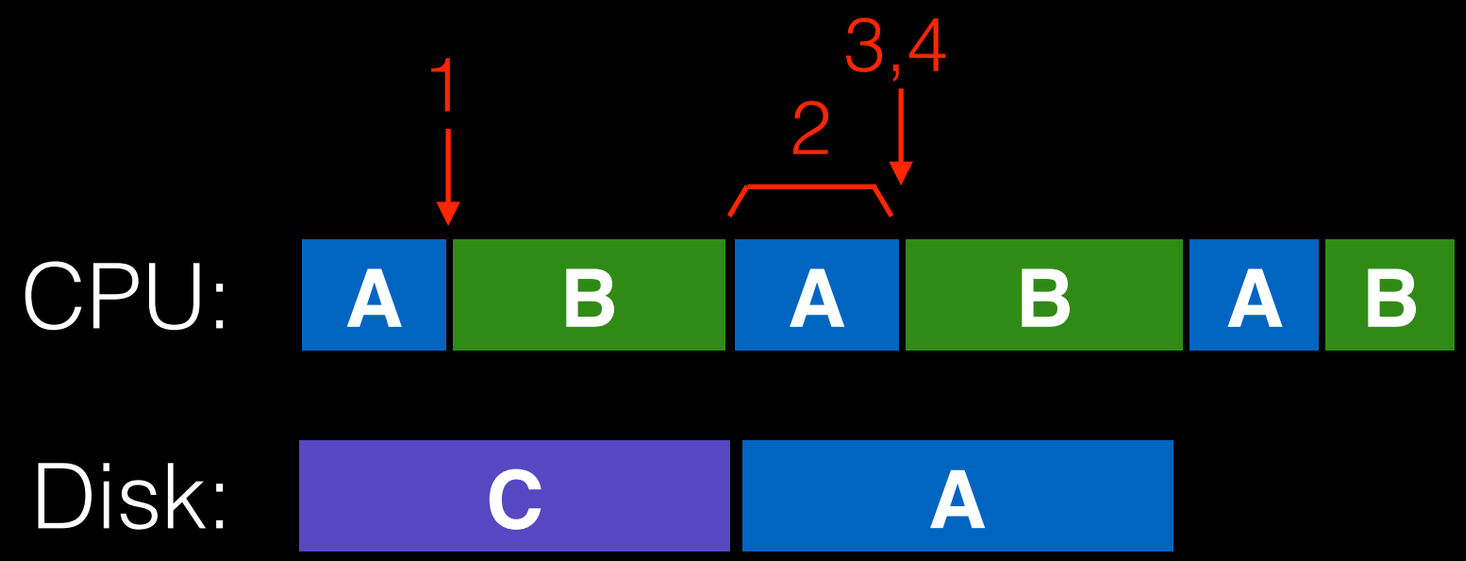
Techniques:
 - hybrid approach
 - interrupt coalescing

Protocol Variants

Status checks: polling *vs.* interrupts

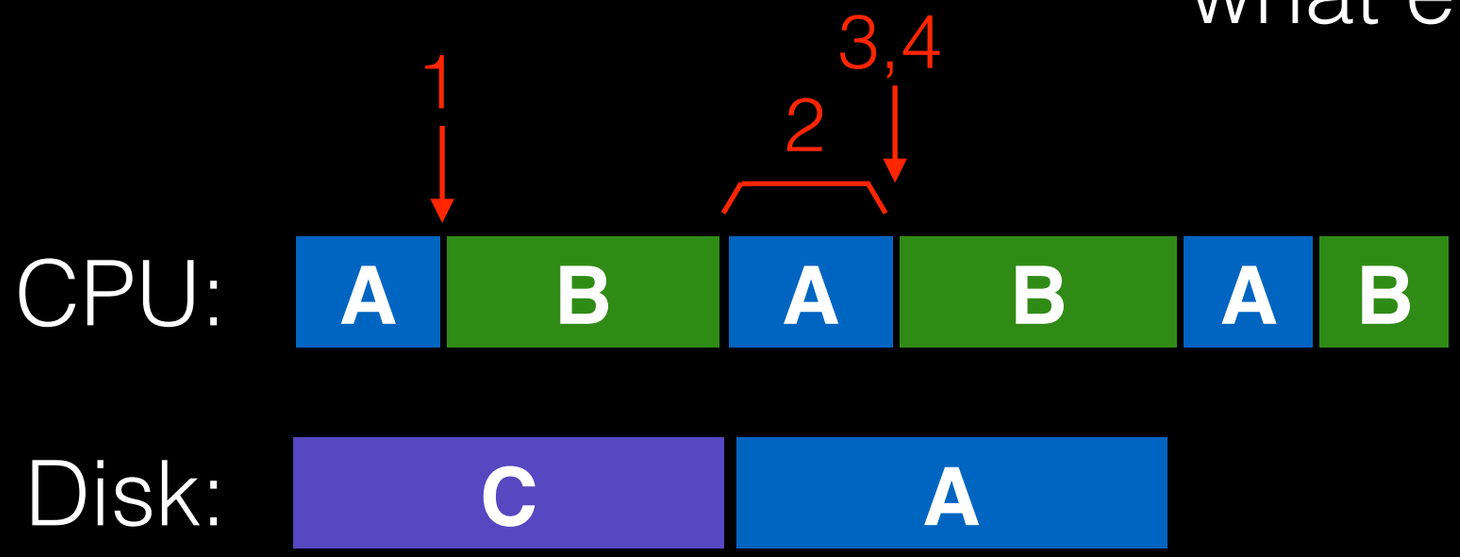
Data: PIO *vs.* DMA

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



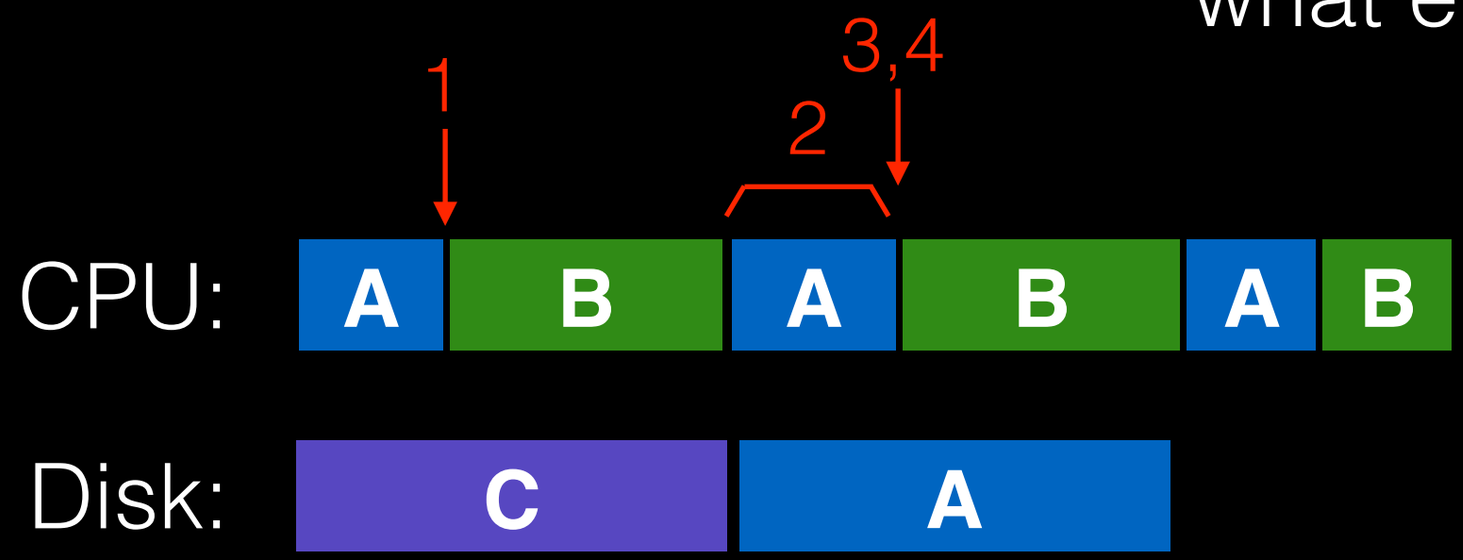

```
while (STATUS == BUSY)                    // 1
   wait for interrupt;
Write data to DATA register               // 2
Write command to COMMAND register         // 3
while (STATUS == BUSY)                    // 4
   wait for interrupt;
```

what 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;
```

what else can we optimize?
data transfer!

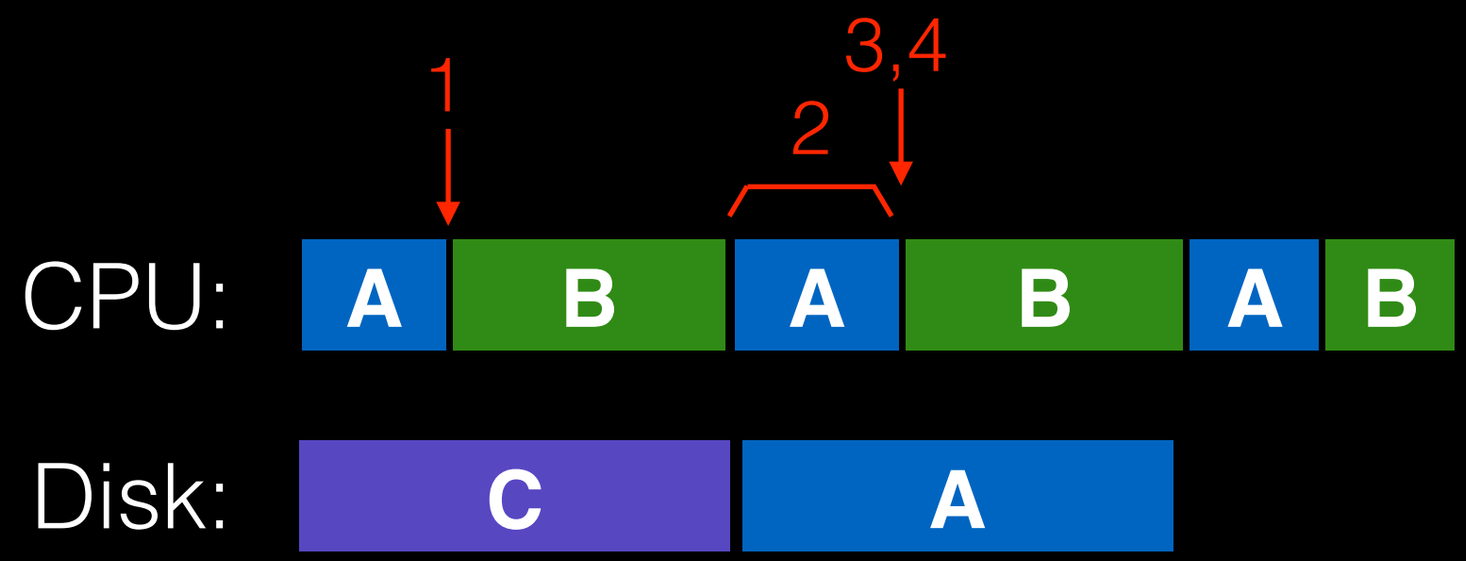



```
while (STATUS == BUSY)                    // 1
   wait for interrupt;
Write data to DATA register               // 2
Write command to COMMAND register         // 3
while (STATUS == BUSY)                    // 4
   wait for interrupt;
```

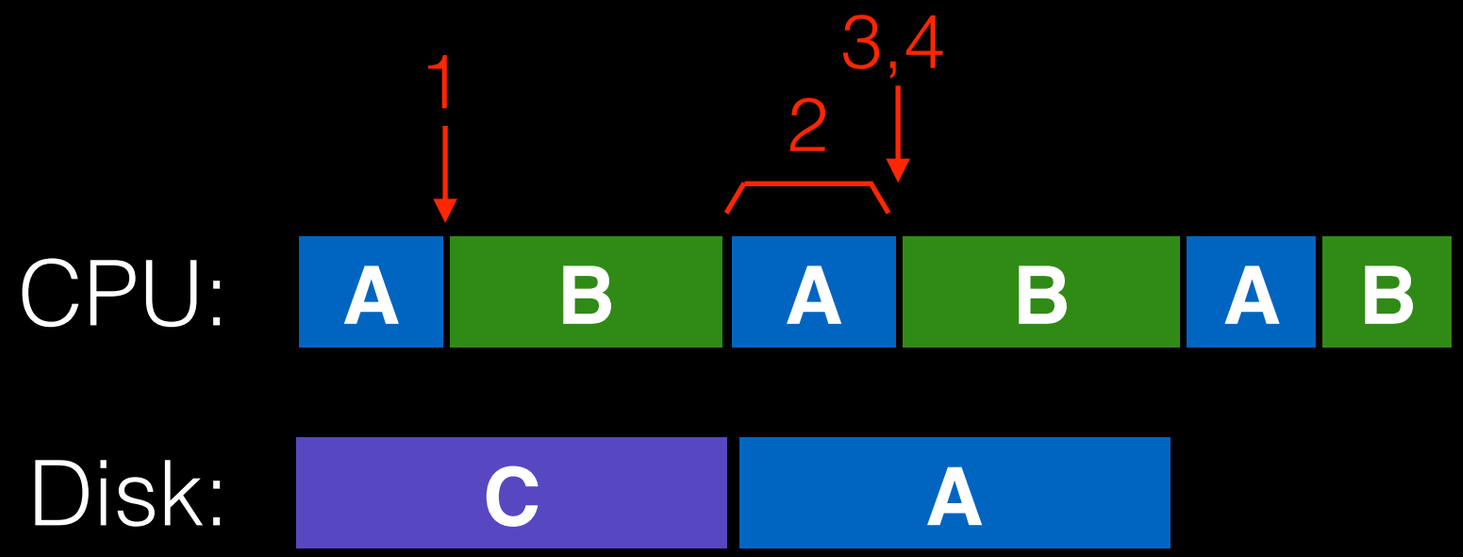
Programmed I/O vs. Direct Memory Access

PIO (Programmed I/O):
- CPU directly tells device what data is

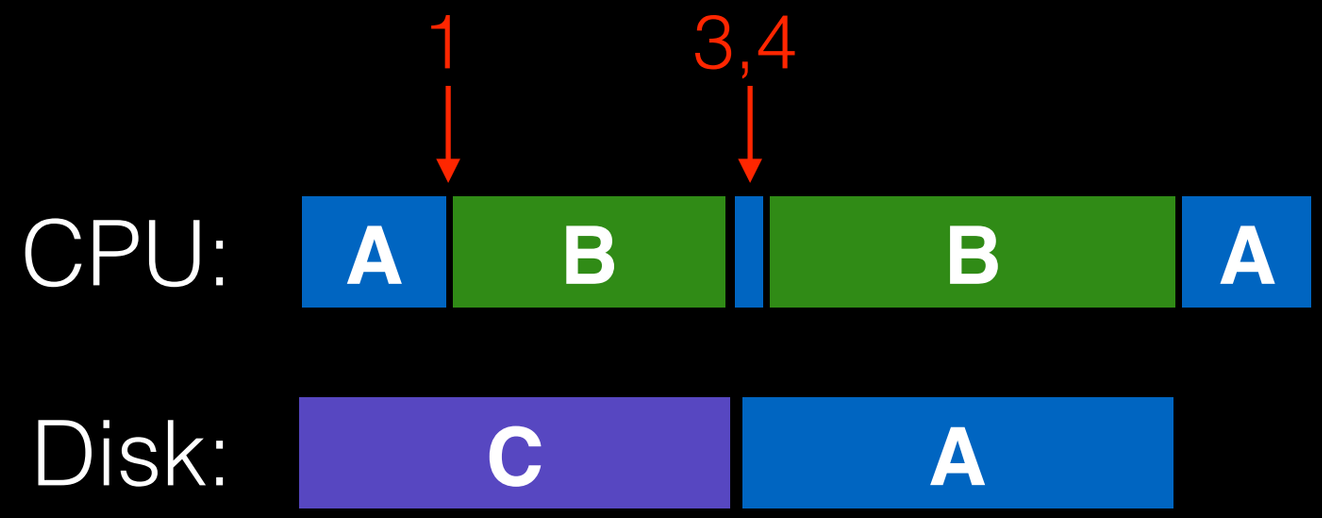
DMA (Direct Memory Access):
- CPU leaves data in memory
- Device reads it directly




```
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;
```




```
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;
```



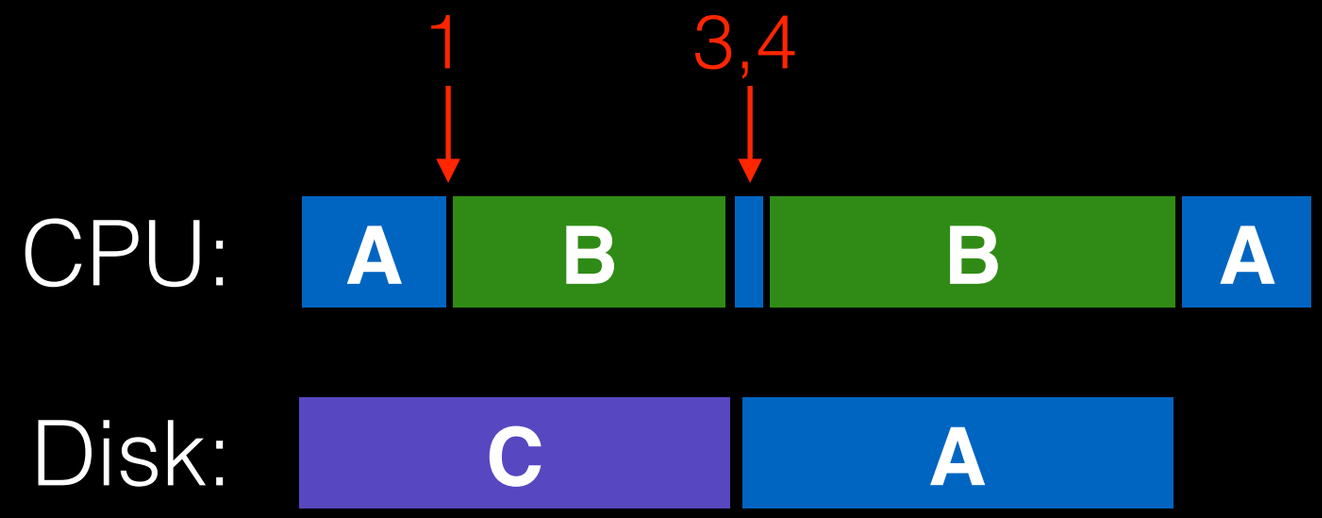

```
while (STATUS == BUSY)                    // 1
   wait for interrupt;
Write data to DATA register               // 2
Write command to COMMAND register         // 3
while (STATUS == BUSY)                    // 4
   wait for interrupt;
```

Protocol Variants

Status checks: polling *vs.* interrupts

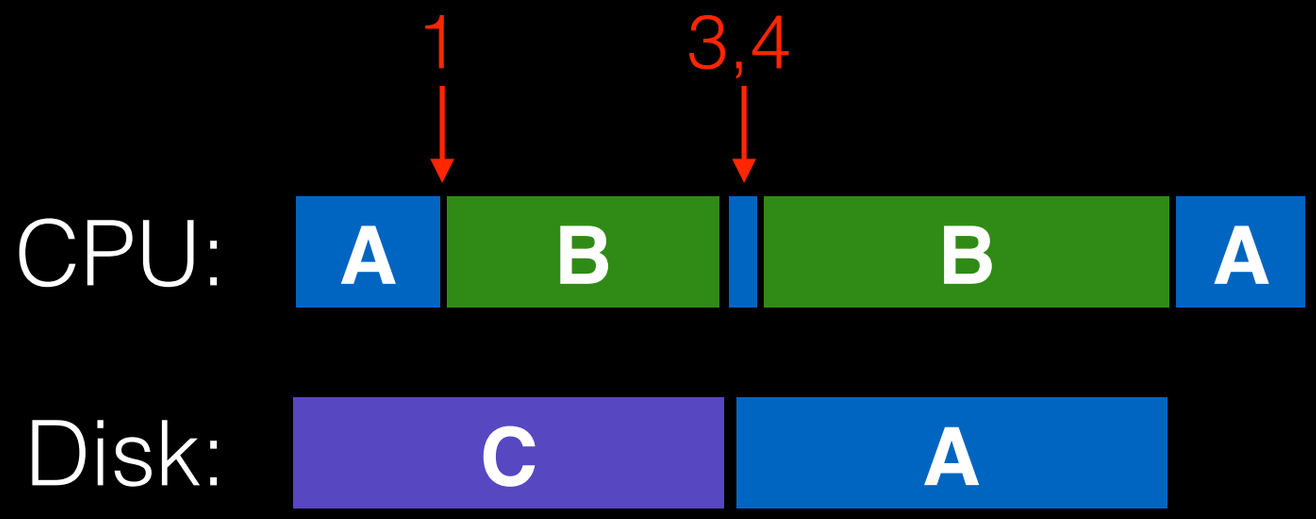
Data: PIO *vs.* DMA

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




```
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;
```

how does OS read and write registers?




```
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;
```

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

Tradeoffs?

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

Data: PIO *vs.* DMA

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

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?

Solution

Encapsulation!

Write driver for each device.

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

Solution

Encapsulation!

Write driver for each device.

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

Encapsulation also enables us to mix-and-match devices, schedulers, and file systems.

Storage Stack

application

file system

scheduler

driver

hard drive

Storage Stack

application

file system

scheduler

driver

hard drive

build common interface
on top of all HDDs

Storage Stack

what about special capabilities?

application

file system

scheduler

driver

hard drive

build common interface on top of all HDDs

Hard Disks

Basic Interface

Disk has a sector-addressable address space (so a disk is like an array of sectors).

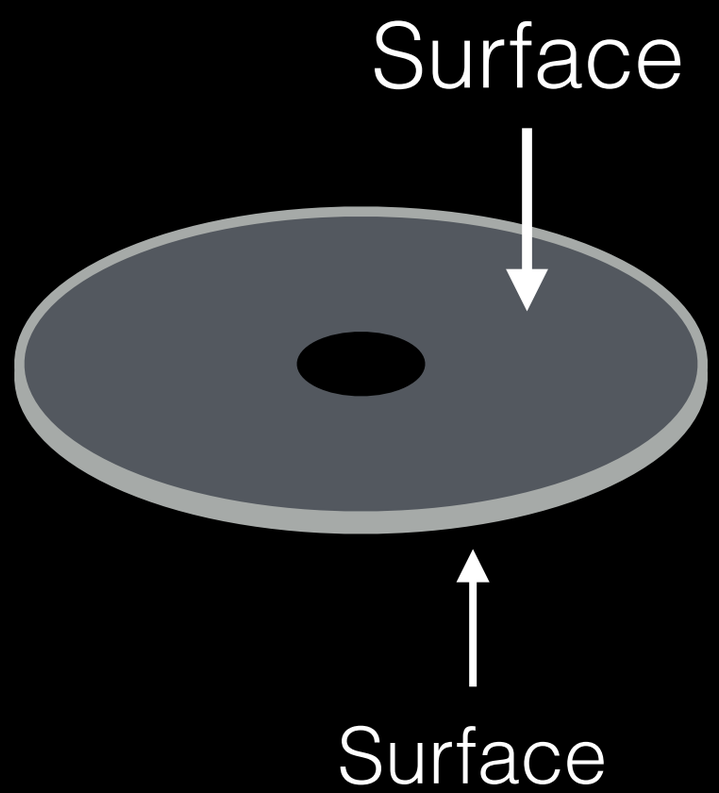
Sectors are typically 512 bytes or 4096 bytes.

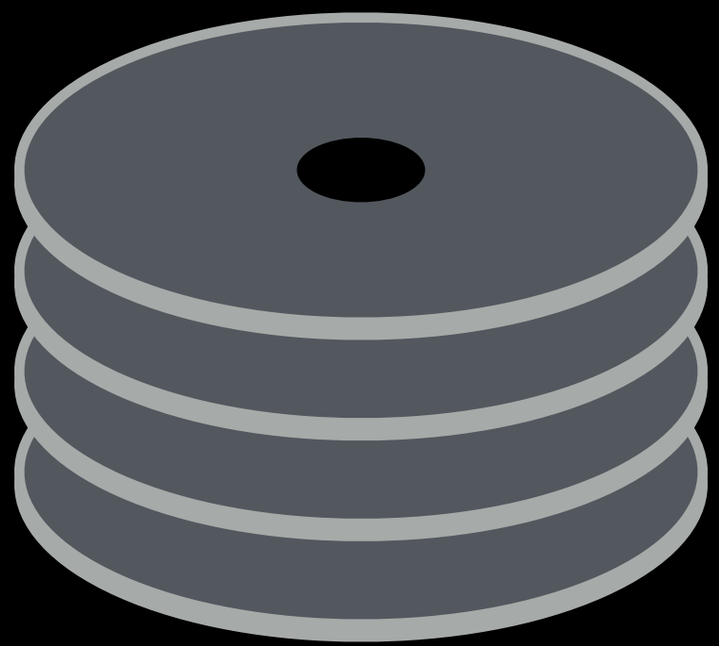
Main operations: reads + writes to sectors.

Platter

Platter is covered with a magnetic film.

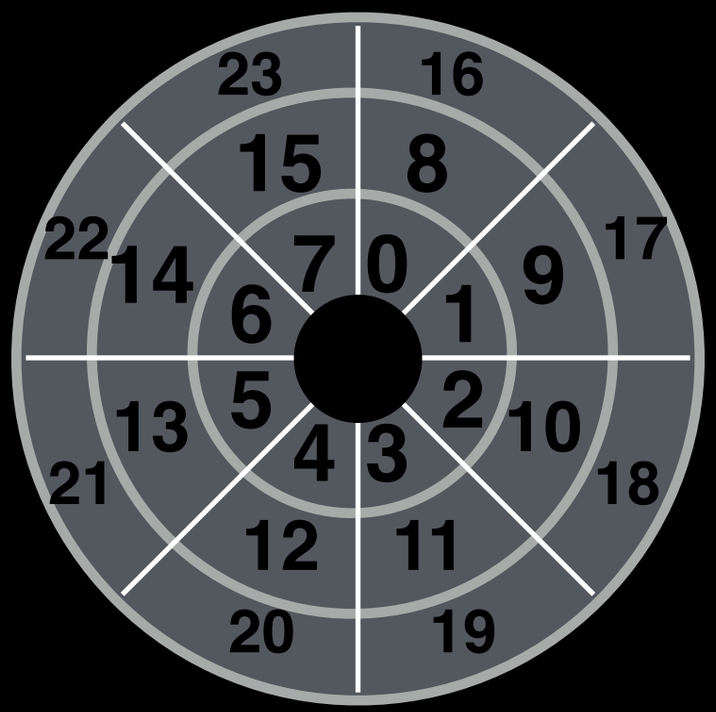
Spindle


Surface
Surface

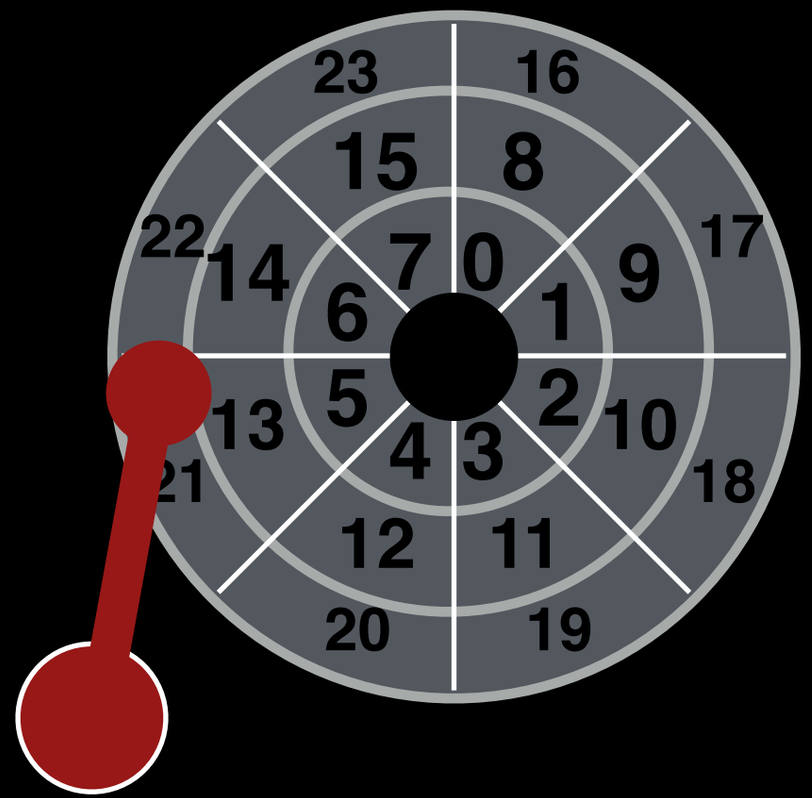


Many platters may be bound to the spindle.

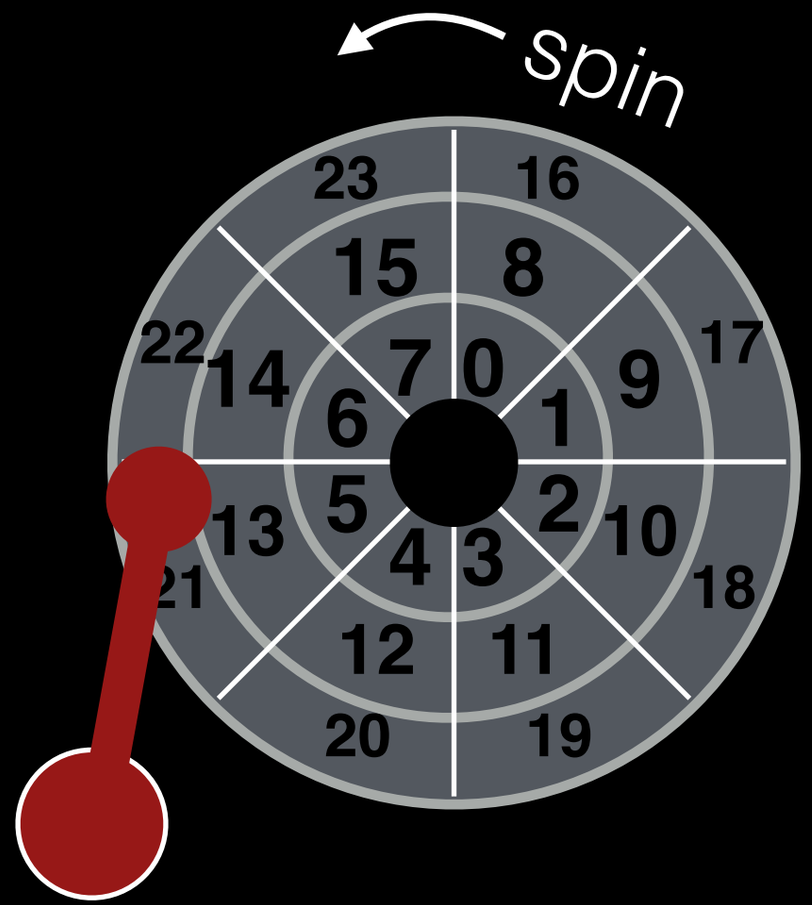
Each surface is divided into rings called tracks.
A stack of tracks (across platters) is called a cylinder.



The tracks are divided into numbered sectors.



Heads on a moving arm can read from each surface.

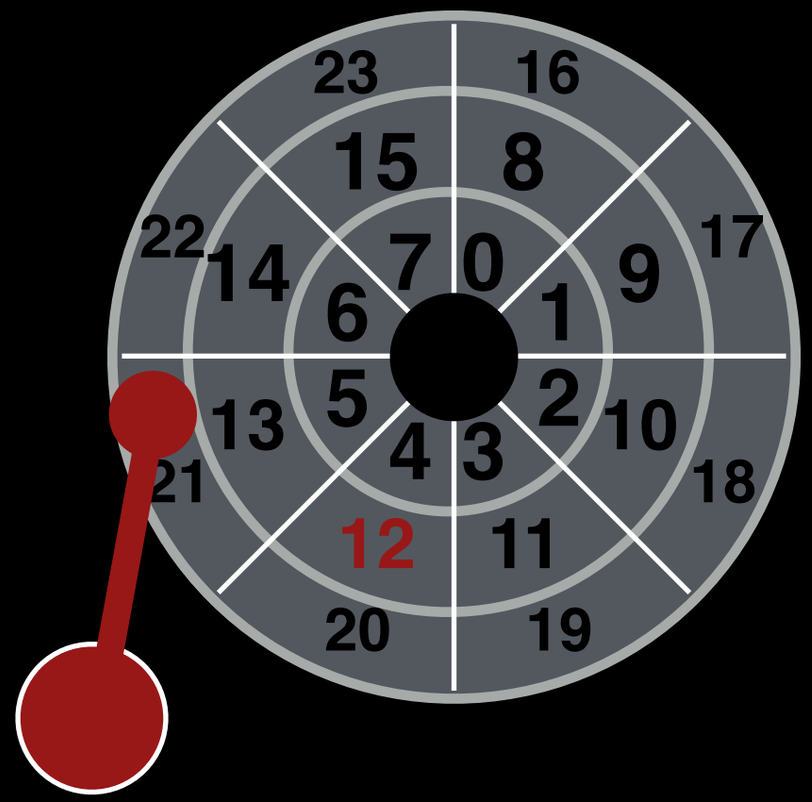


Spindle/platters rapidly spin.

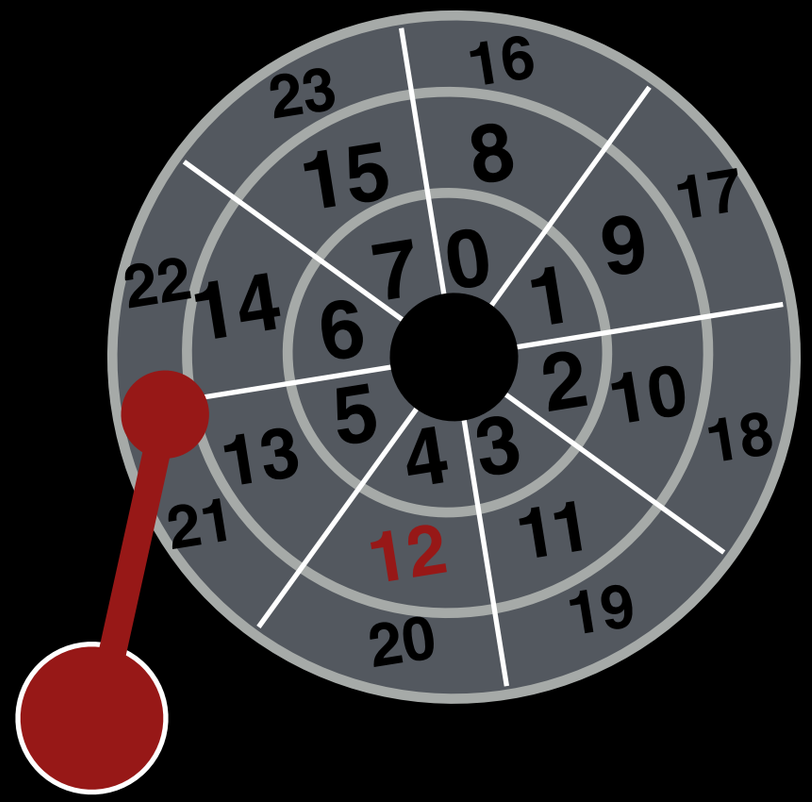
Don’t try this at home!

http://youtu.be/9eMWG3fwiEU?t=30s

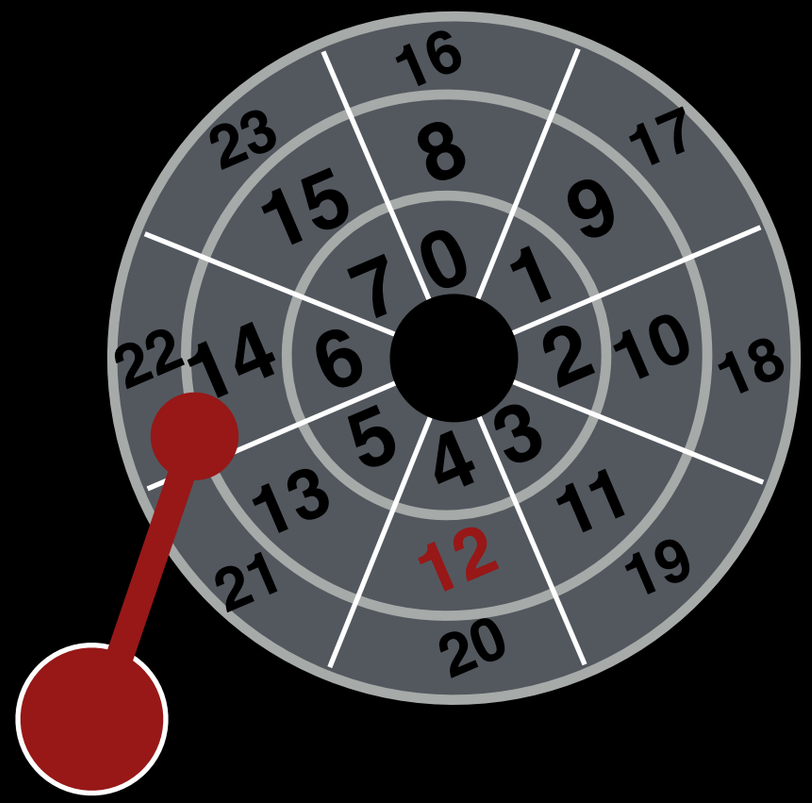
Let’s Read 12!



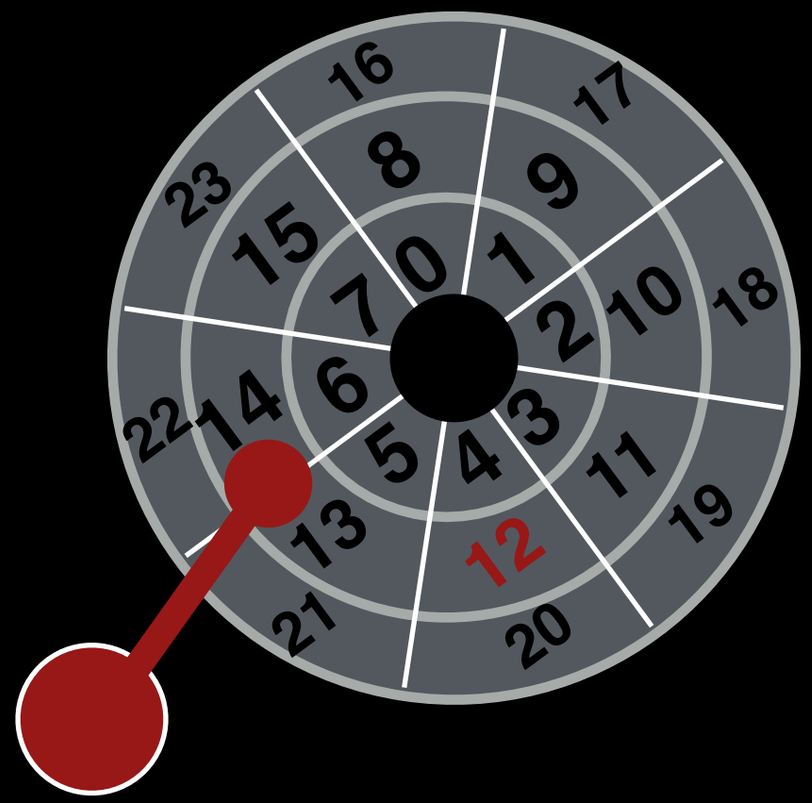
Seek to right track.



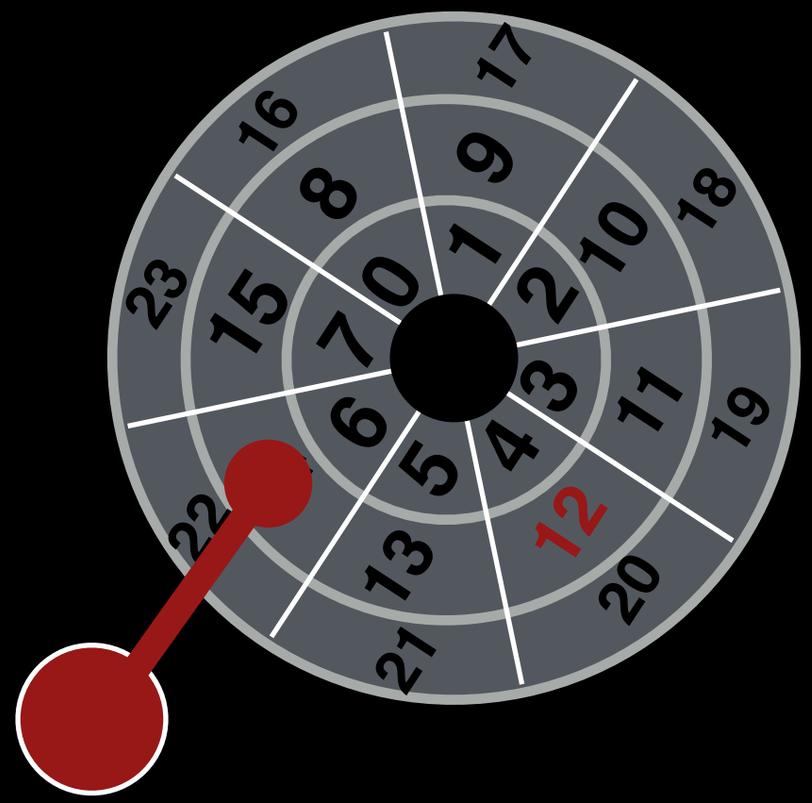
Seek to right track.



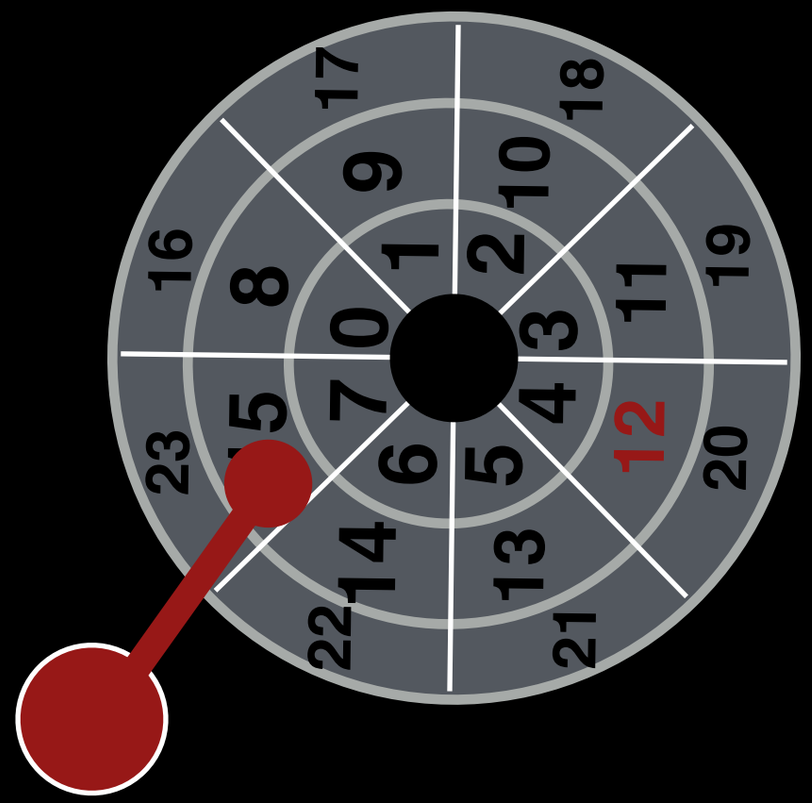
Seek to right track.



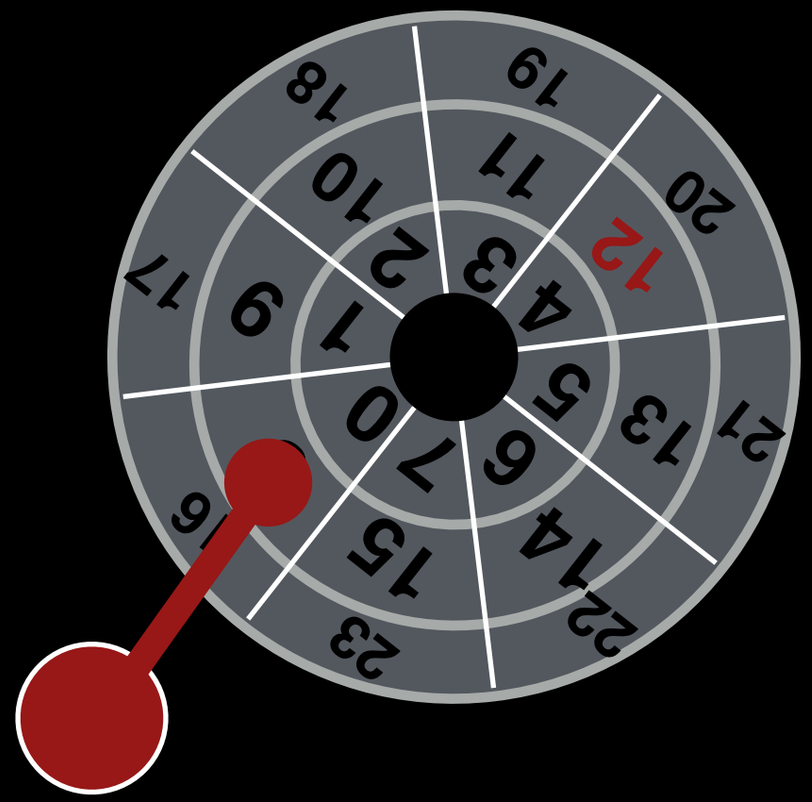
Wait for rotation.

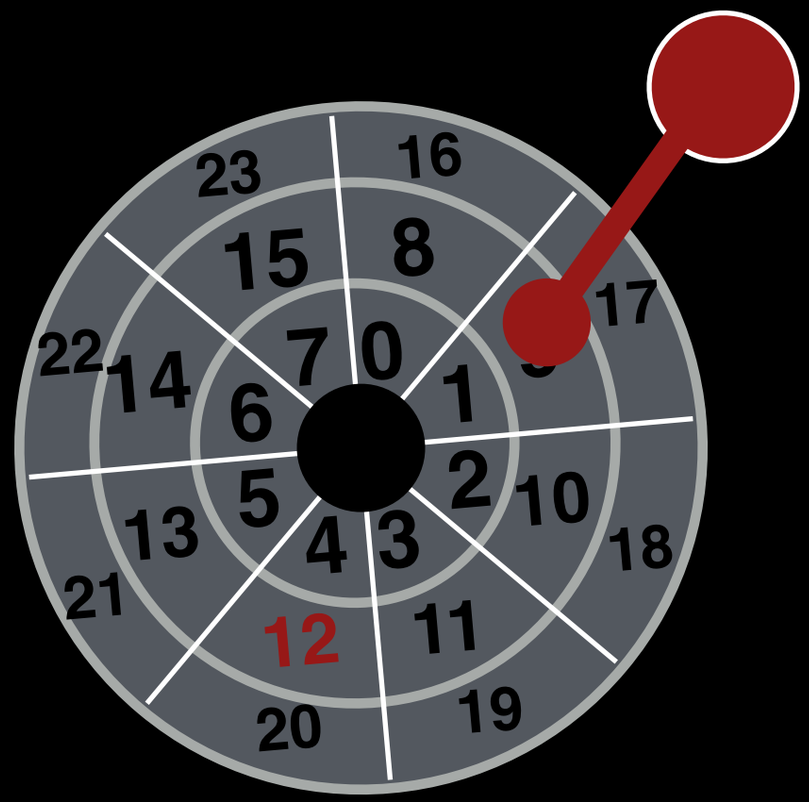


Wait for rotation.



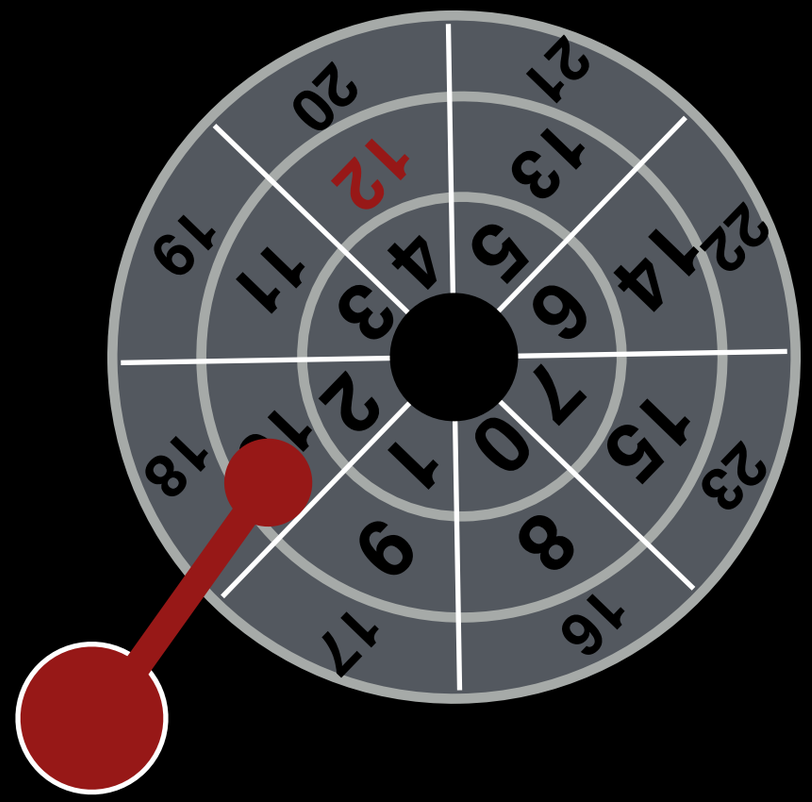
Wait for rotation.



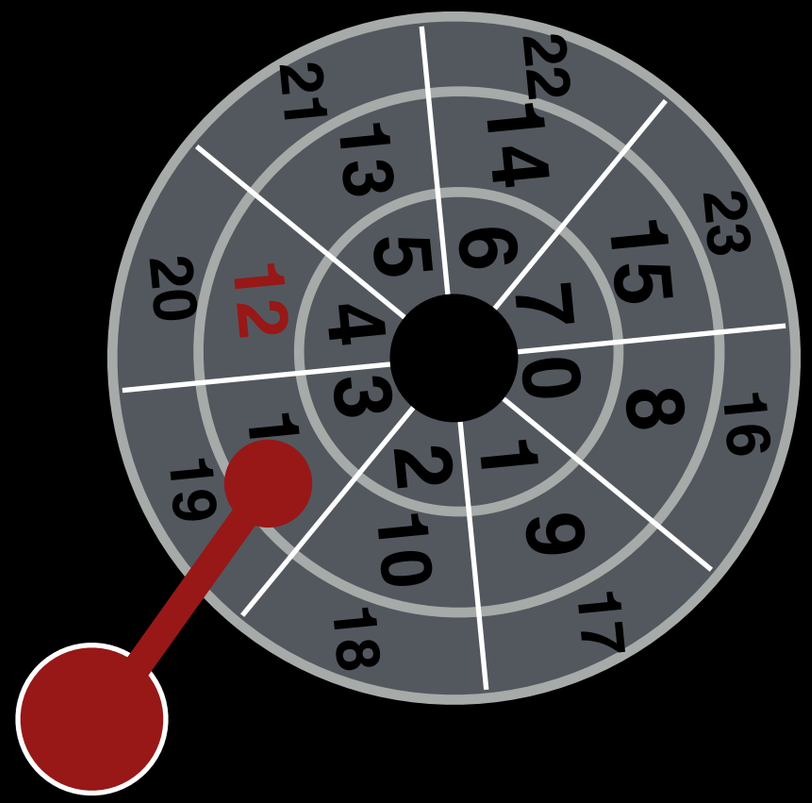


Wait for rotation.

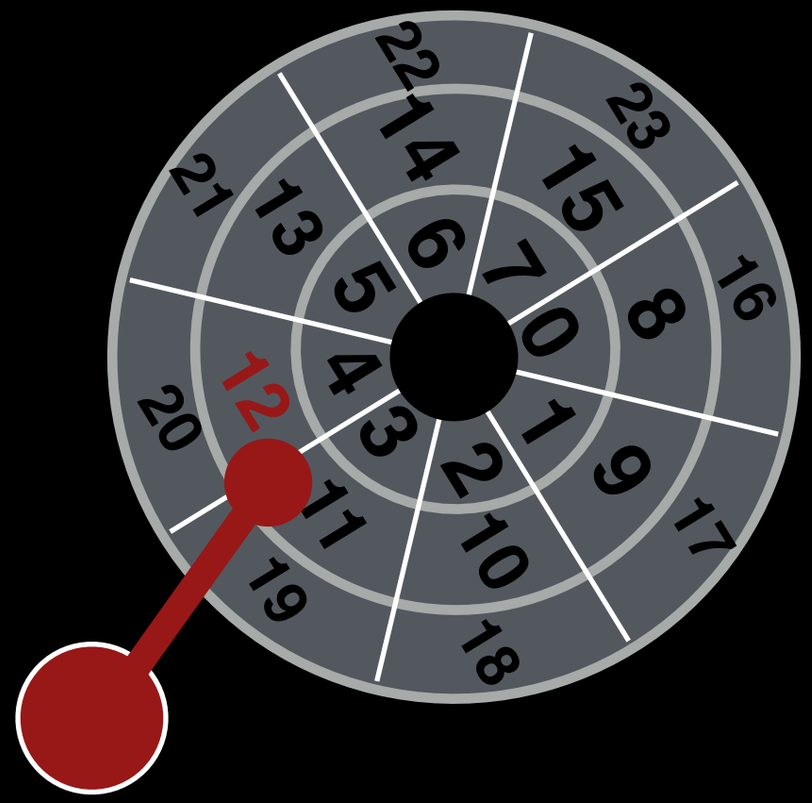
Wait for rotation.



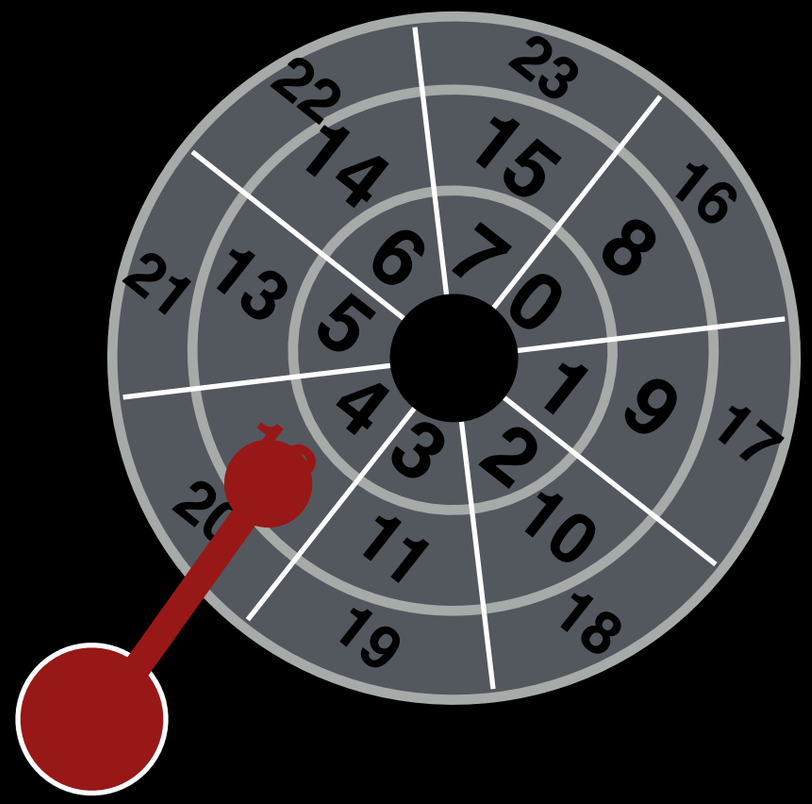
Wait for rotation.



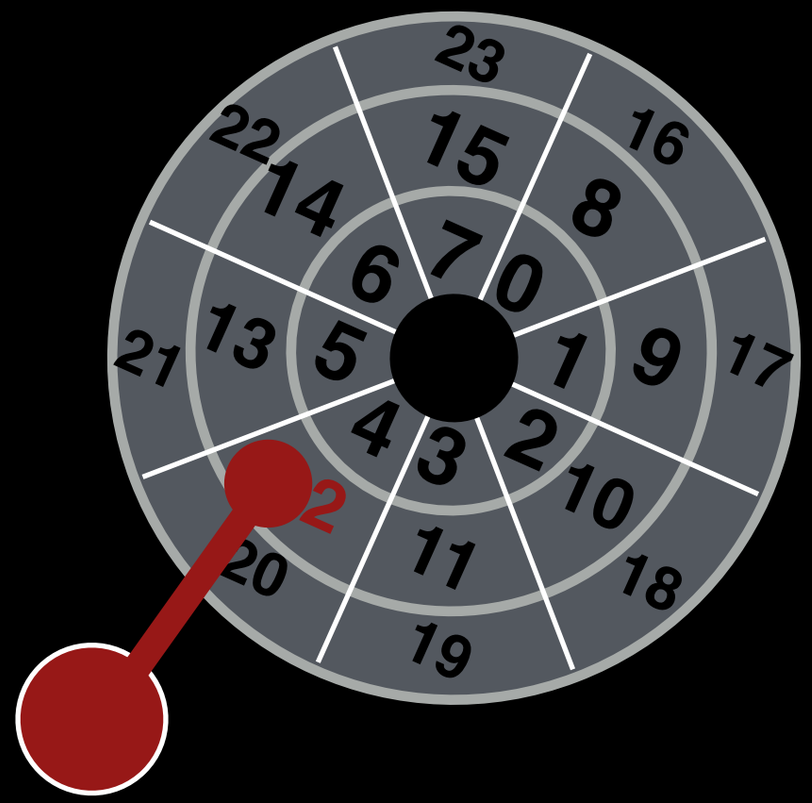
Transfer data.



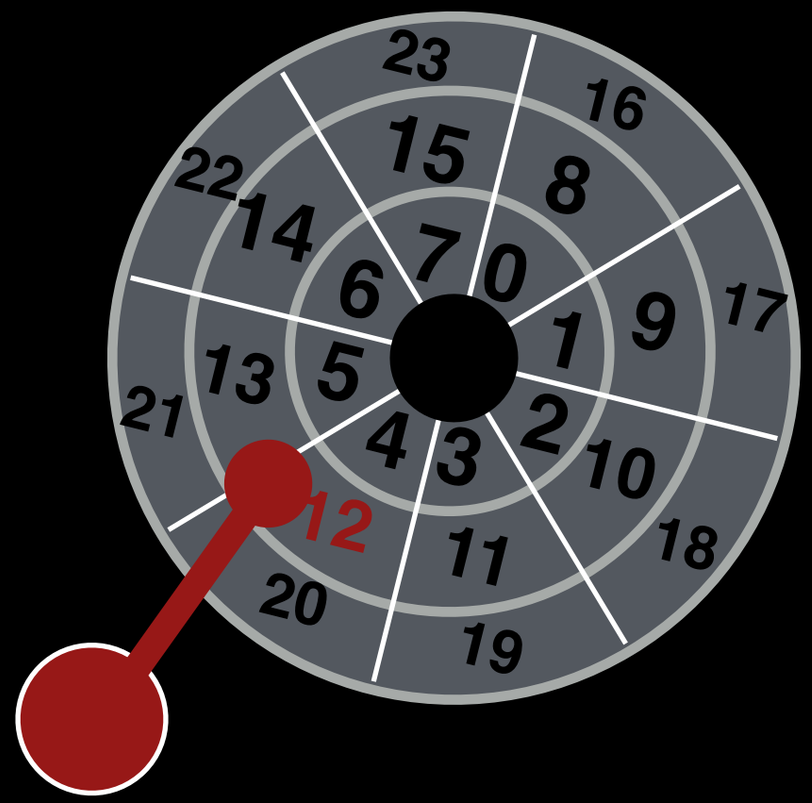
Transfer data.



Transfer data.



Yay!



Seek, Rotate, Transfer

Must accelerate, coast, decelerate, settle

Seeks often take several milliseconds!

Settling alone can take 0.5 - 2 ms.

Entire seek often takes 4 - 10 ms.

Seek, Rotate, Transfer

Depends on rotations per minute (RPM).
 - 7200 RPM is common, 1500 RPM is high end.

1 / 7200 RPM =
1 minute / 7200 rotations =
1 second / 120 rotations =
12 ms / rotation

Seek, Rotate, Transfer

Depends on rotations per minute (RPM).
- 7200 RPM is common, 1500 RPM is high end.

1 / 7200 RPM =
1 minute / 7200 rotations =
1 second / 120 rotations =
12 ms / rotation

so it may take **6 ms** on avg to rotate to target (0.5 * 12 ms)

Seek, Rotate, Transfer

Pretty fast — depends on RPM and sector density.

100+ MB/s is typical.

1s / 100 MB = 10 ms / MB = 4.9 us / sector
(assuming 512-byte sector)

Workload

So…
- seeks are slow
- rotations are slow
- transfers are fast

What kind of workload is fastest for disks?

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 Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

Disk Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

Sequential workload: what is throughput for each?

Disk Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	**125 MB/s**	**105 MB/s**
Platters	4	4
Cache	16 MB	32 MB

Cheeta: 125 MB/s.
Barracuda: 105 MB/s.

Disk Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

Random workload: what is throughput for each?
(what else do you need to know?)

Disk Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

Random workload: what is throughput for each?
Assume 16-KB reads.

Disk Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

Random workload: what is throughput for each?
Assume 16-KB reads.

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

$$\text{avg rotation} = \frac{1}{2} \times \frac{1\ \text{min}}{15000}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

$$\text{avg rotation} = \frac{1}{2} \times \frac{1\ \text{min}}{15000} \times \frac{60\ \text{sec}}{1\ \text{min}} \times \frac{1000\ \text{ms}}{1\ \text{sec}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

$$\text{avg rotation} = \frac{1}{2} \times \frac{1\ \text{min}}{15000} \times \frac{60\ \text{sec}}{1\ \text{min}} \times \frac{1000\ \text{ms}}{1\ \text{sec}} = 2\ \text{ms}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

$$\text{transfer} = \frac{1 \text{ sec}}{125 \text{ MB}} \times 16 \text{ KB}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

$$\text{transfer} = \frac{1 \text{ sec}}{125 \text{ MB}} \times 16 \text{ KB} \times \frac{1{,}000{,}000 \text{ us}}{1 \text{ sec}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

$$\text{transfer} = \frac{1 \text{ sec}}{125 \text{ MB}} \times 16 \text{ KB} \times \frac{1{,}000{,}000 \text{ us}}{1 \text{ sec}} = 125 \text{ us}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms

$$\text{throughput} = \frac{16\ \text{KB}}{6.1\text{ms}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms

$$\text{throughput} = \frac{16\ \text{KB}}{6.1\text{ms}} \times \frac{1\ \text{MB}}{1024\ \text{KB}} \times \frac{100\ \text{ms}}{1\ \text{sec}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Cheetah?

Cheetah time = 4ms + 2ms + 125us = 6.1ms

$$\text{throughput} = \frac{16\ \text{KB}}{6.1\text{ms}} \times \frac{1\ \text{MB}}{1024\ \text{KB}} \times \frac{100\ \text{ms}}{1\ \text{sec}} = 2.5\ \text{MB/s}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

$$\text{avg rotation} = \frac{1}{2} \times \frac{1 \text{ min}}{7200}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

$$\text{avg rotation} = \frac{1}{2} \times \frac{1\ \text{min}}{7200} \times \frac{60\ \text{sec}}{1\ \text{min}} \times \frac{1000\ \text{ms}}{1\ \text{sec}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

$$\text{avg rotation} = \frac{1}{2} \times \frac{1\ \text{min}}{7200} \times \frac{60\ \text{sec}}{1\ \text{min}} \times \frac{1000\ \text{ms}}{1\ \text{sec}} = 4.1\ \text{ms}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

$$\text{transfer} = \frac{1 \text{ sec}}{105 \text{ MB}} \times 16 \text{ KB}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

$$\text{transfer} = \frac{1 \text{ sec}}{105 \text{ MB}} \times 16 \text{ KB} \times \frac{1{,}000{,}000 \text{ us}}{1 \text{ sec}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

$$\text{transfer} = \frac{1 \text{ sec}}{105 \text{ MB}} \times 16 \text{ KB} \times \frac{1{,}000{,}000 \text{ us}}{1 \text{ sec}} = 149 \text{ us}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

$$\text{throughput} = \frac{16\ \text{KB}}{13.2\text{ms}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

$$\text{throughput} = \frac{16\ \text{KB}}{13.2\text{ms}} \times \frac{1\ \text{MB}}{1024\ \text{KB}} \times \frac{1000\ \text{ms}}{1\ \text{sec}}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

How long does an average 16-KB read take w/ Barracuda?

Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

$$\text{throughput} = \frac{16\ \text{KB}}{13.2\text{ms}} \times \frac{1\ \text{MB}}{1024\ \text{KB}} \times \frac{1000\ \text{ms}}{1\ \text{sec}} = 1.2\ \text{MB/s}$$

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

	Cheetah	Barracuda
Sequential	125 MB/s	105 MB/s
Random	2.5 MB/s	1.2 MB/s

Other Improvements

Track Skew

Zones

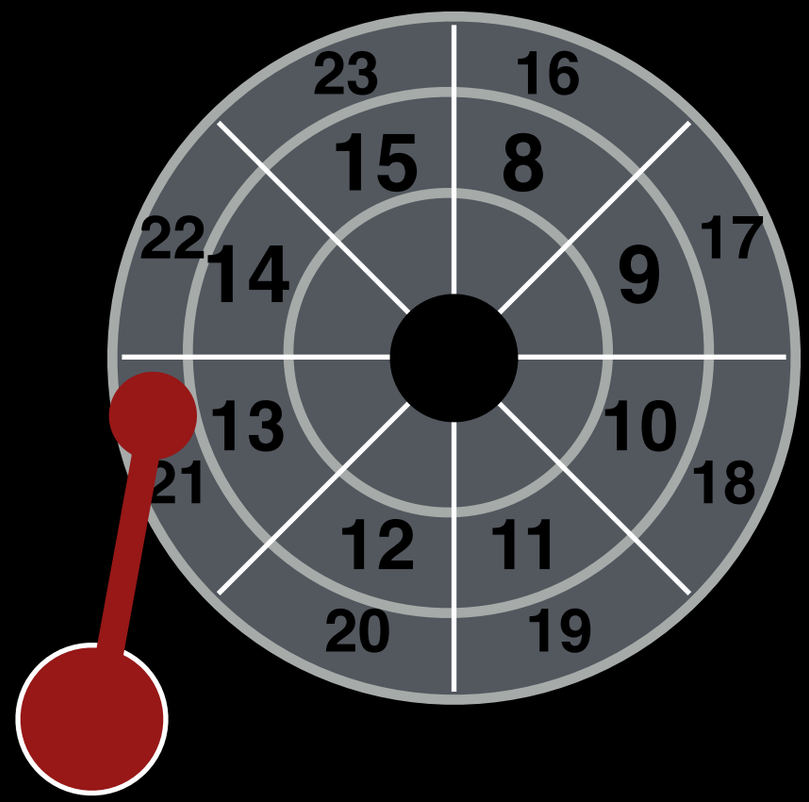
Cache

Other Improvements

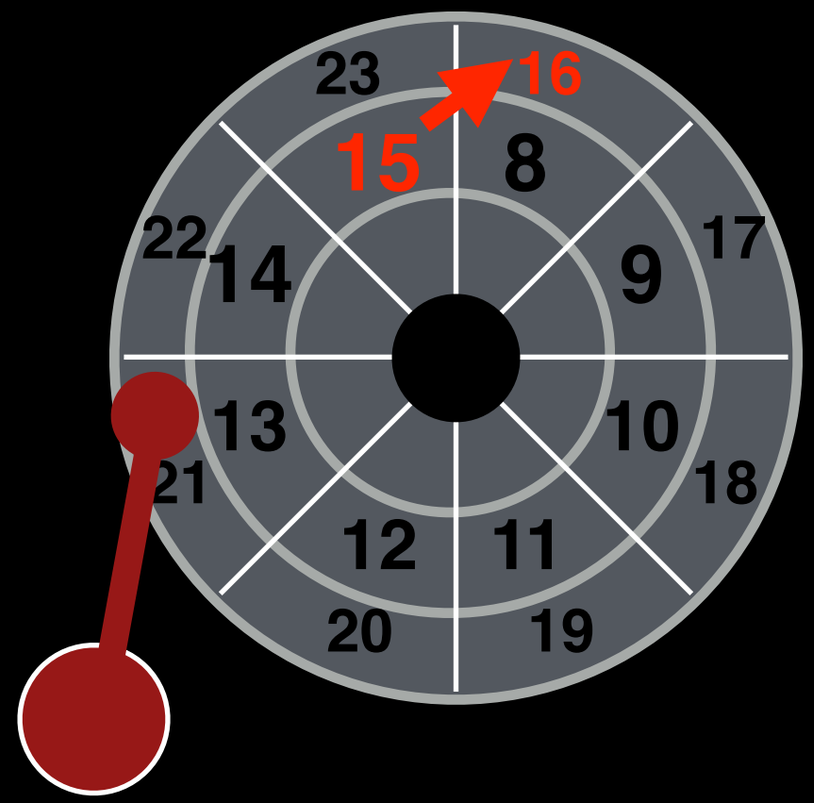
Track Skew

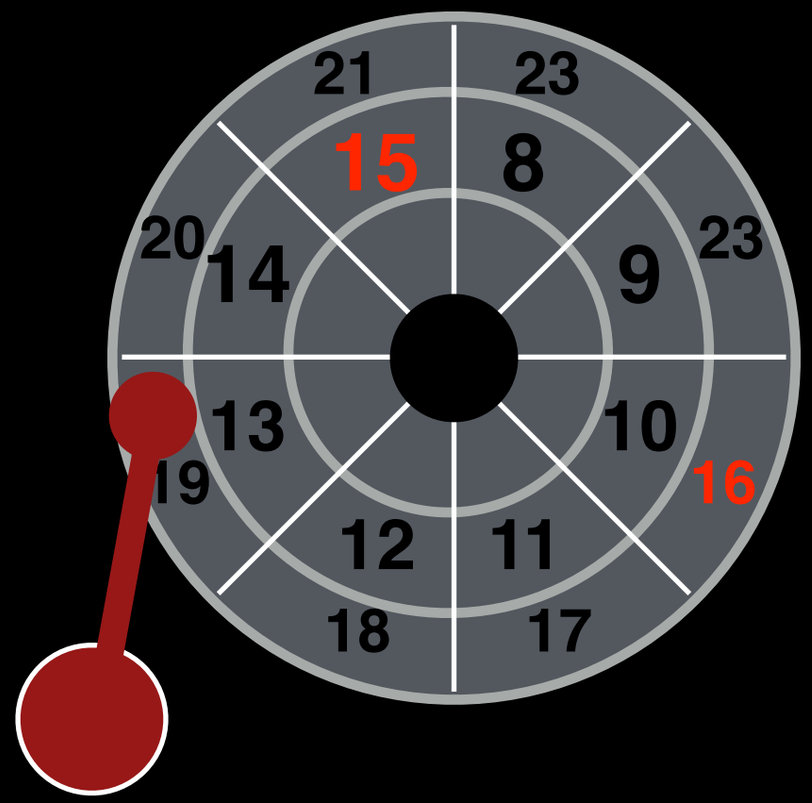
Zones

Cache

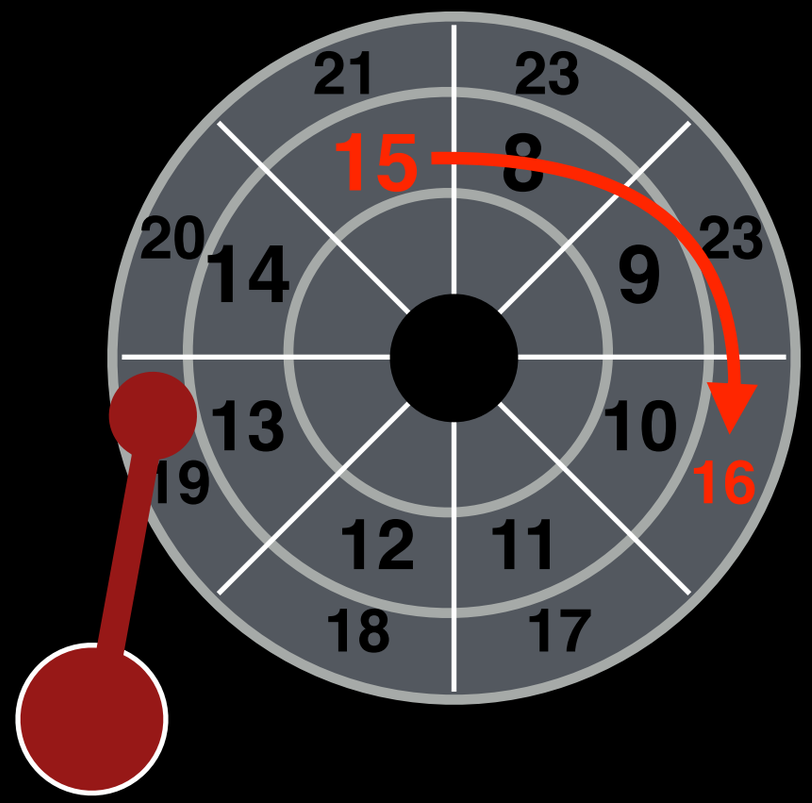

23
16
15
8
22
14
9
17
13
10
21
18
12
11
20
19

When reading 16 after 15, the head won't settle quick enough, so we need to do a rotation.




21
23
15
8
20
14
9
23
13
10
19
16
12
11
18
17

enough time to settle now

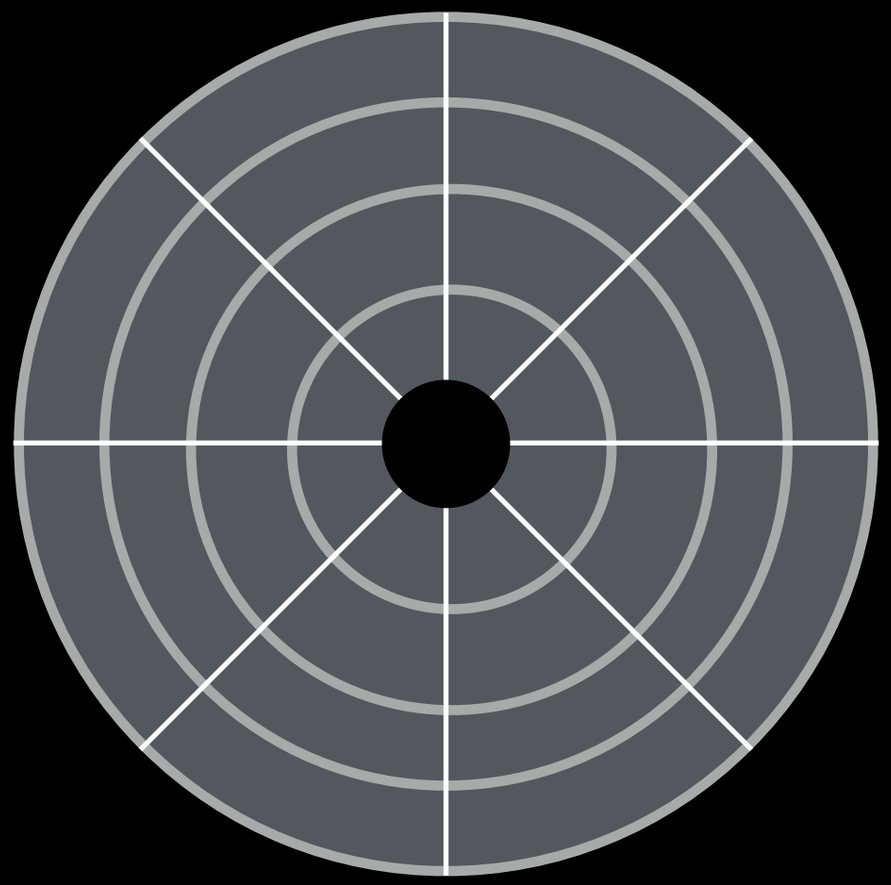


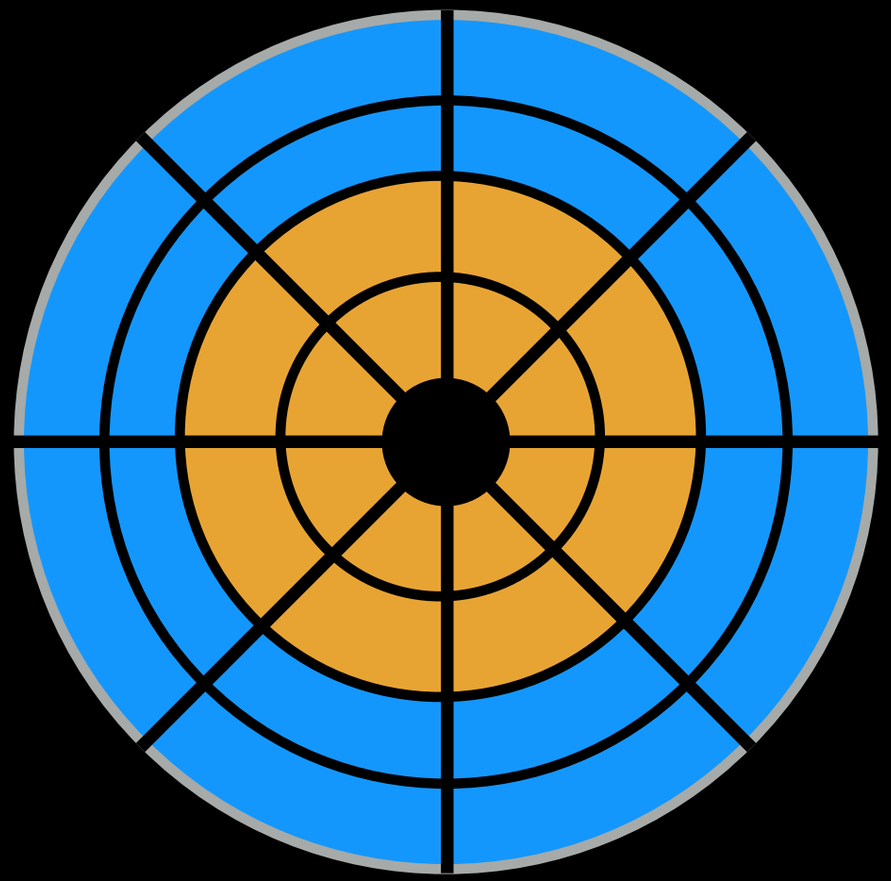
Other Improvements

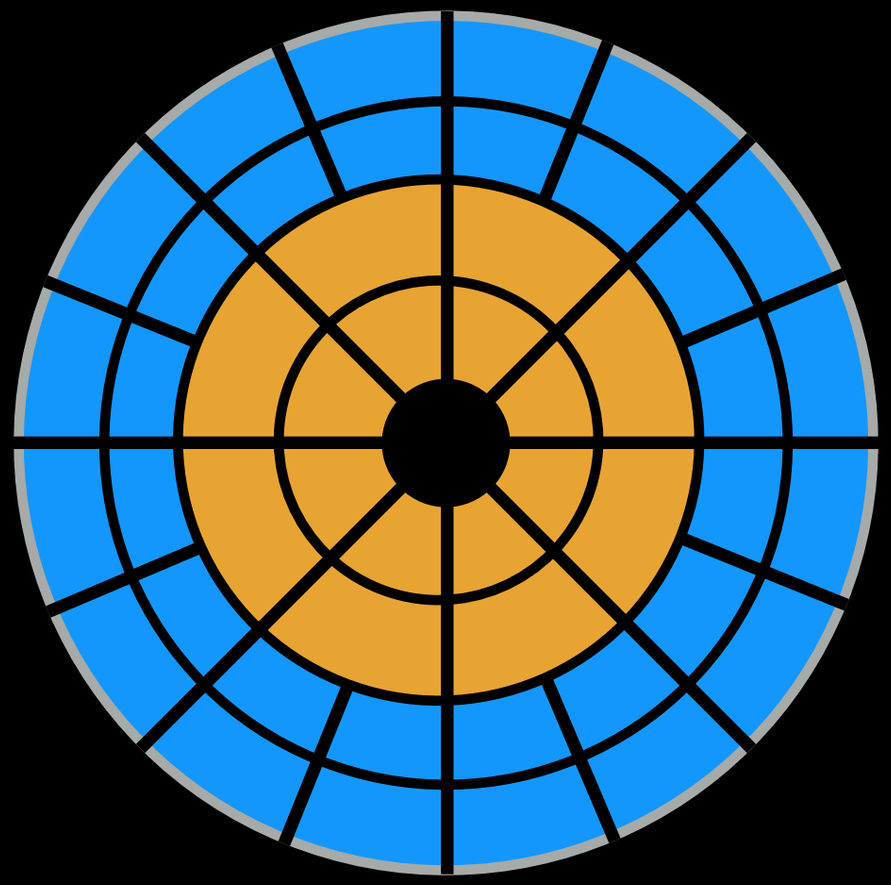
Track Skew

Zones

Cache







Other Improvements

Track Skew

Zones

Cache

Drive Cache

Drives may cache both reads and writes.

OS does this to.

What advantage does drive have for reads?

What advantage does drive have for writes?

Schedulers

Schedulers

Given a stream of requests, in what order should they be served?

FCFS (First-Come-First-Serve)

Assume seek+rotate = 10 ms on average.
Assume transfer = 100 MB/s.

How long (roughly) does the below workload take?
The integers are sector numbers.

300001, 700001, 300002, 700002, 300003, 700003

FCFS (First-Come-First-Serve)

Assume seek+rotate = 10 ms on average.
Assume transfer = 100 MB/s.

How long (roughly) does the below workload take?
The integers are sector numbers.

300001, 700001, 300002, 700002, 300003, 700003 (~60ms)

FCFS (First-Come-First-Serve)

Assume seek+rotate = 10 ms on average.
Assume transfer = 100 MB/s.

How long (roughly) do the below workloads take?
The integers are sector numbers.

300001, 700001, 300002, 700002, 300003, 700003 (~60ms)
300001, 300002, 300003, 700001, 700002, 700003

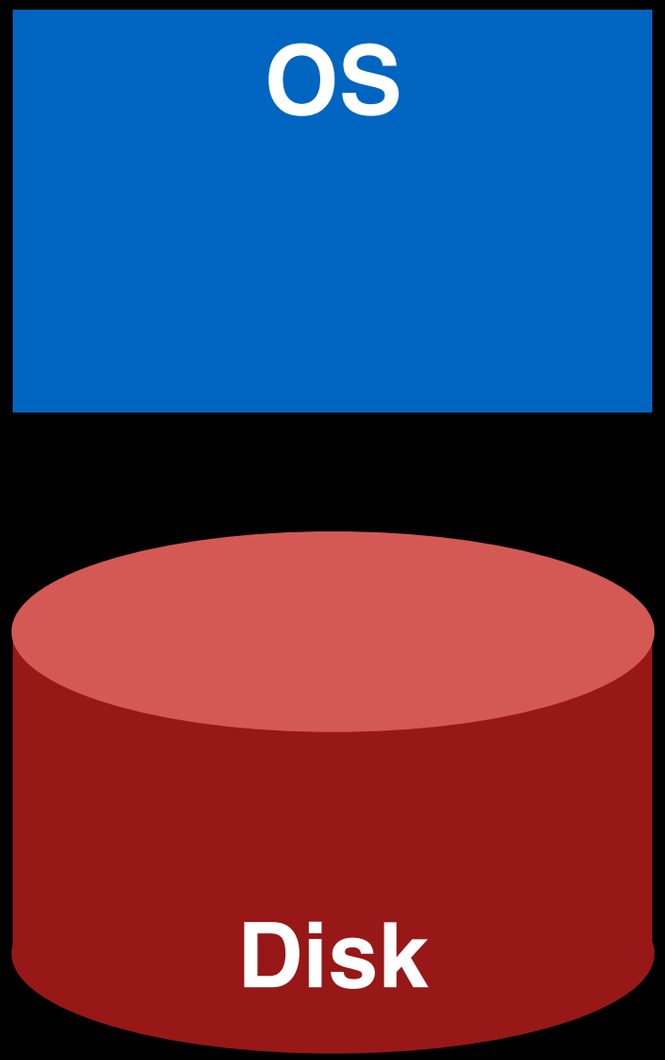
FCFS (First-Come-First-Serve)

Assume seek+rotate = 10 ms on average.
Assume transfer = 100 MB/s.

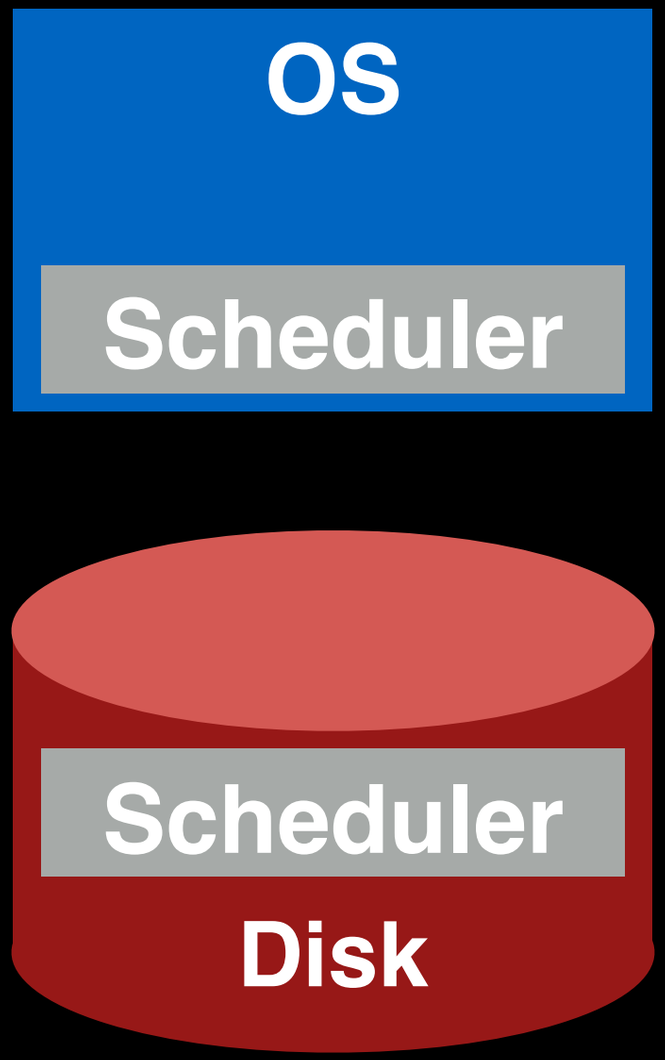
How long (roughly) do the below workloads take?
The integers are sector numbers.

300001, 700001, 300002, 700002, 300003, 700003 (~60ms)
300001, 300002, 300003, 700001, 700002, 700003 (~20ms)

Schedulers



Schedulers



Where should the scheduler go?

SPTF (Shortest Positioning Time First)

Strategy: always choose the request that will take the least time for seeking and rotating.

How to implement in disk?
How to implement in OS?

SPTF (Shortest Positioning Time First)

Strategy: always choose the request that will take the least time for seeking and rotating.

How to implement in disk?
How to implement in OS?

Disadvantages?

SCAN

Sweep back and forth, from one end of disk to the other, serving requests as you go.

Pros/Cons?

SCAN

Sweep back and forth, from one end of disk to the other, serving requests as you go.

Pros/Cons?

Better: C-SCAN (circular scan)

- only sweep in one direction

What happens?

Assume 2 processes, and C-SCAN.

```
void reader(int fd) {
   char buf[1024];
   int rv;
   while((rv = read(buf)) != 0) {
      assert(rv);
      // takes short time, e.g., 1ms
      process(buf, rv);
   }
}
```

Work Conservation

Work conserving schedulers always try to do I/O if there's I/O to be done.

Sometimes, it's better to wait instead if you anticipate another request will appear nearby.

Such non-work-conserving schedulers are called anticipatory schedulers.

CFQ (Linux Default)

Completely Fair Queueing.

Queue for each process.

Do weighted round-robin between queues, with slice time proportional to priority.

Optimize order within queue.

Yield slice only if idle for a given time (anticipation).

Summary

Overlap I/O and CPU whenever possible!
 - use interrupts, DMA

Never do random I/O unless you must!
 - e.g., Quicksort is a terrible algorithm on disk

Announcements

Office hours: after class, in lab.

p3b due Friday.