PERSISTENCE: I/O DEVICES

Andrea Arpaci-Dusseau

CS 537, Fall 2019

ADMINISTRIVIA

Grades:

Project 3 (email TAs if problems)

Project 5 available now (xv6 Memory)

- Due Monday 11/4 (5 pm)
- Many lab hours through then
- Turn in any of 3 versions:
    - v1a (alloc alternating pages, all marked as UNKNOWN PID)
    - v1b (alternating pages, some marked UNKNOWN, most known PIDs)
    - v2 (contiguous allocations when possible, some marked UNKNOWN, most known PIDs

Midterm 2: Nov 11/6 (Wed) from 7:30-9:30pm

- Practice exams available
- Room details on Canvas
- Mostly Concurrency
  + Some Virtualization (usually repeated from Midterm1)

AGENDA / LEARNING OUTCOMES

How does the OS **interact** with I/O devices?

How can we optimize this?

What are the components of a **hard disk drive**?

How to calculate **sequential** and **random throughput** of a disk?

What algorithms are used to **schedule I/O** requests?

OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

1. Virtualization

2. Concurrency

3. Persistence

VIRTUALIZATION

Make each application believe it has each resource to itself

CPU and Memory

Abstraction: Process API, Address spaces

Mechanism:

- Limited direct execution, CPU scheduling
- Address translation (segmentation, paging, TLB)

Policy: MLFQ, LRU etc.

CONCURRENCY

Events occur simultaneously and may interact with one another

Need to

- Hide concurrency from independent processes
- Manage concurrency with interacting processes

Provide abstractions (locks, semaphores, condition variables etc.)

Correctness: mutual exclusion, ordering

Difficult to write multi-threaded applications!

OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

~~1. Virtualization~~

~~2. Concurrency~~

3. Persistence

PERSISTENCE

How to ensure data is available across reboots

- even after power outages, hardware failure, system crashes?

Topics:

- Persistent storage devices (HDDs, RAID, SSDs)
- File API for processes
- FS implementation (meta-data structures, allocation policies)
- Crash recovery (journaling)
- Advanced Topics: Distributed systems?

MOTIVATION: NEED INPUT + OUTPUT

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

keyboard, display, disks

what if no input?

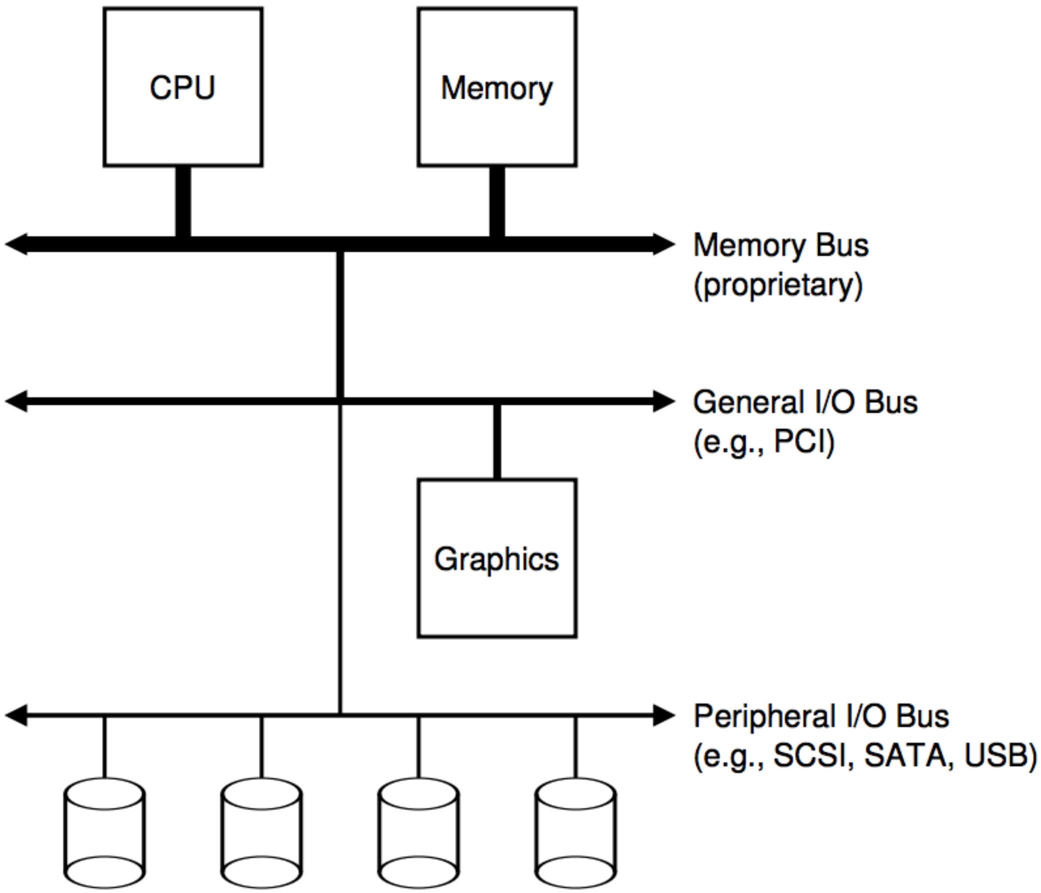
what if no output?

what if no state recorded between different computations?

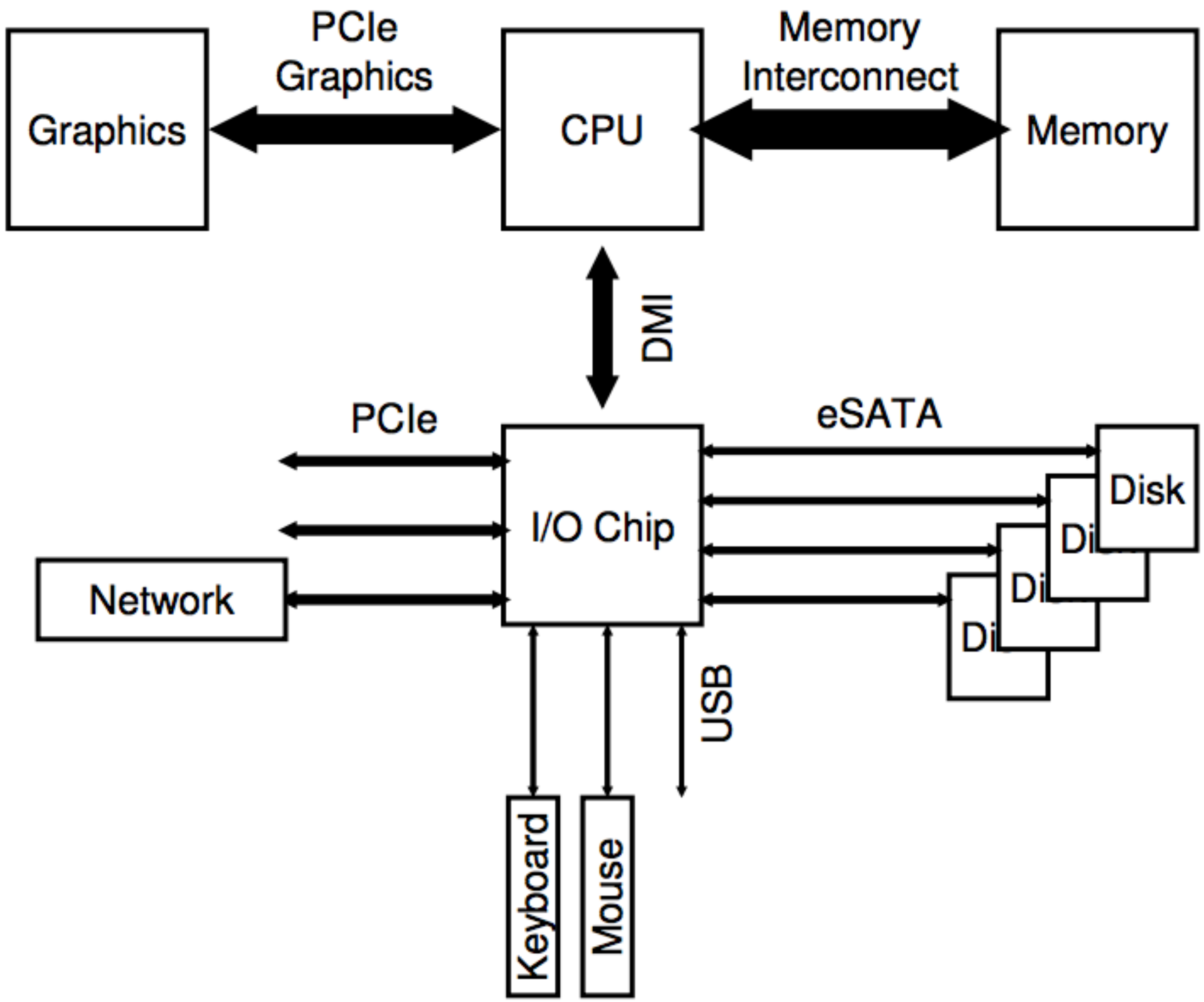
We want:

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

HARDWARE SUPPORT FOR I/O

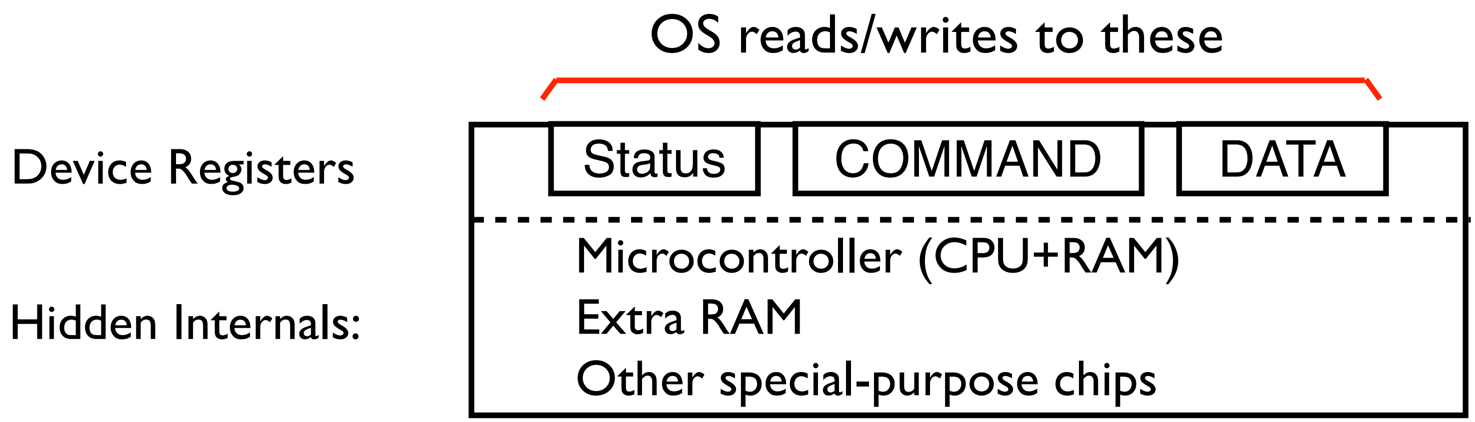


Hierarchical buses

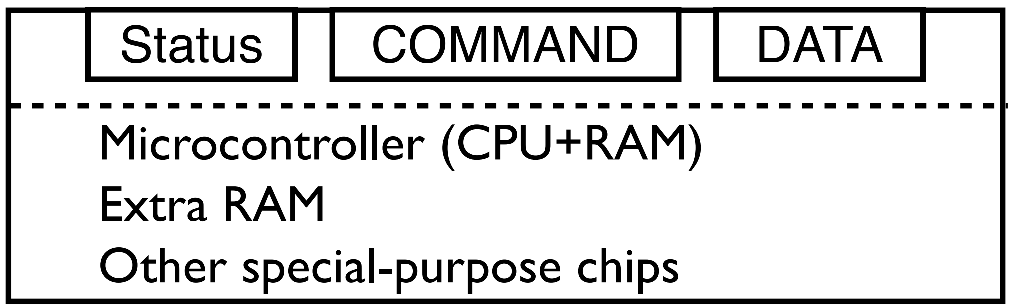


Graphics
PCIe
Graphics
CPU
Memory
Interconnect
Memory
DMI
PCIe
I/O Chip
Network
eSATA
Disk
Di
Di
Di
USB
Keyboard
Mouse

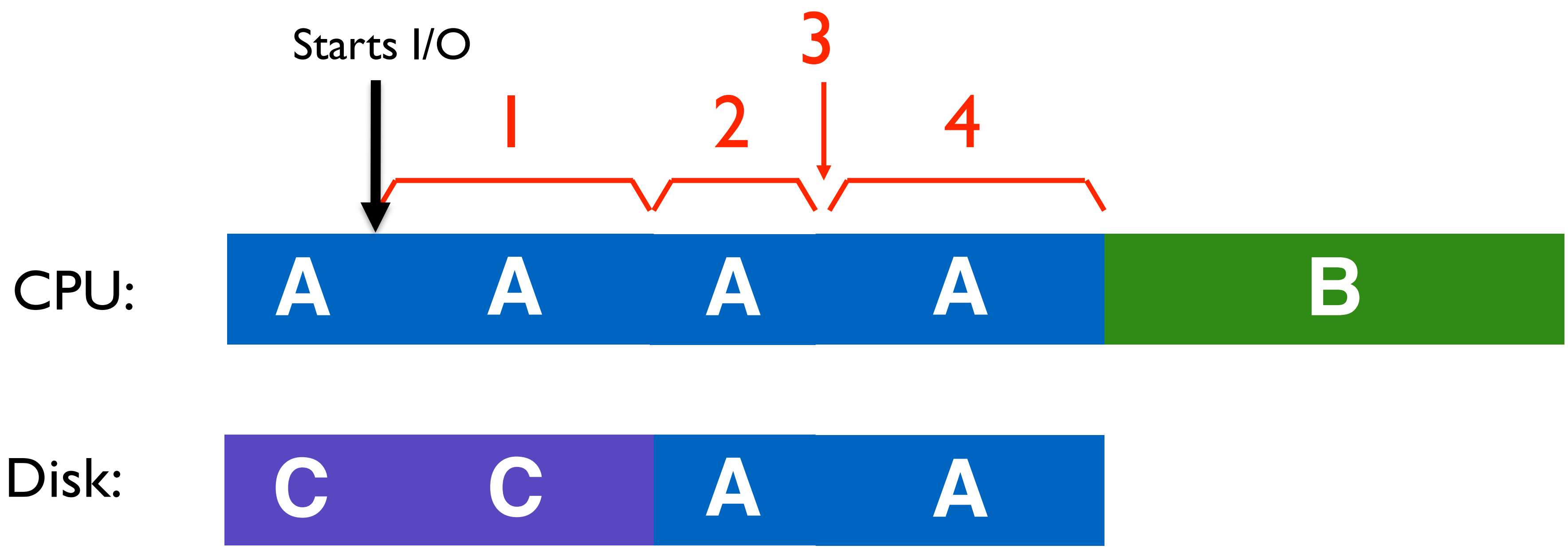
CANONICAL DEVICE



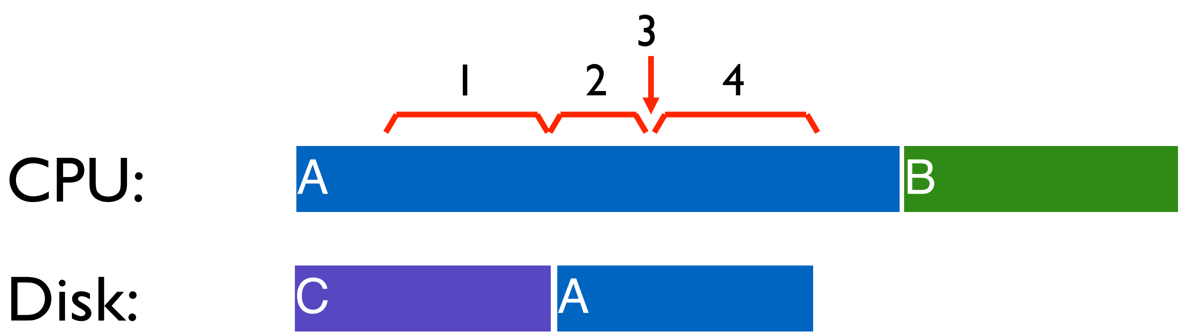
EXAMPLE WRITE PROTOCOL: STARTING POINT




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



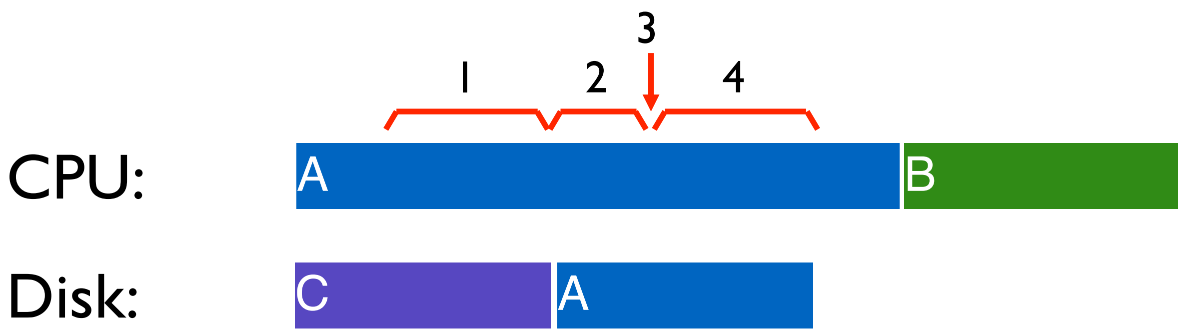

```
while (STATUS == BUSY)            // 1
    ;
Write data to DATA register       // 2
Write command to COMMAND register // 3
while (STATUS == BUSY)            // 4
    ;
```



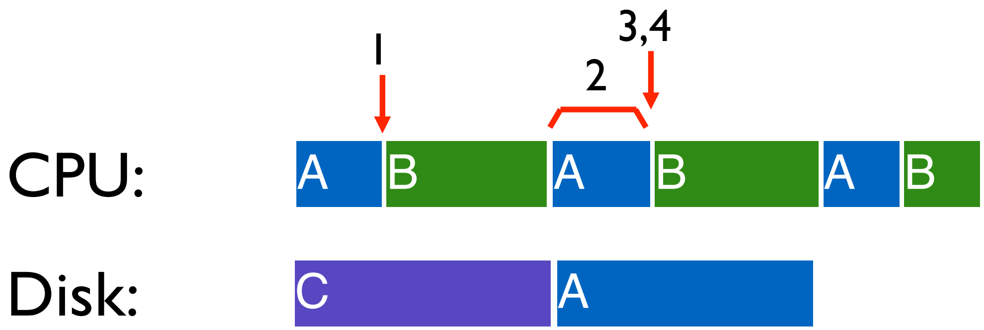

```
while (STATUS == BUSY)                  // 1
  ;
Write data to DATA register             // 2
Write command to COMMAND register       // 3
while (STATUS == BUSY)                  // 4
  ;
```

how to avoid spinning?

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;
```




```
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;
```

EXAMPLE WRITE PROTOCOL: INTERRUPTS

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)

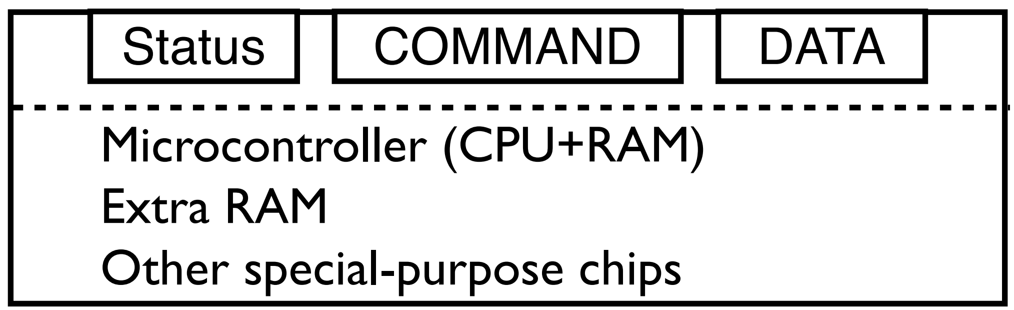
Flood of interrupts arrive

- 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

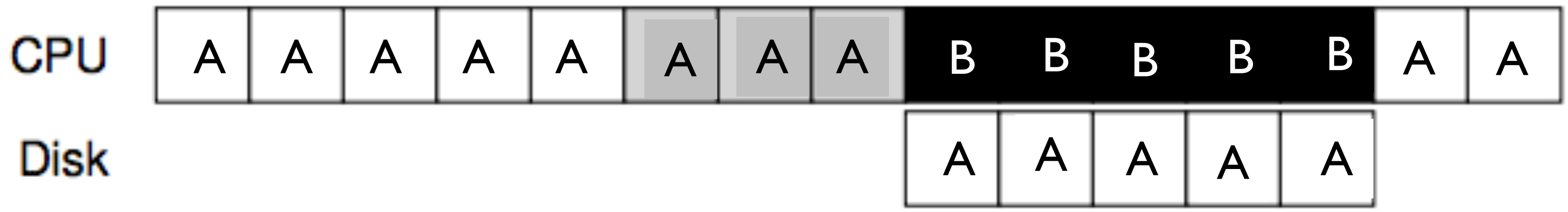


~~Status checks: polling vs. interrupts~~

PIO vs DMA

Special instructions vs. Memory mapped I/O

DATA TRANSFER COSTS WITH PIO




```
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

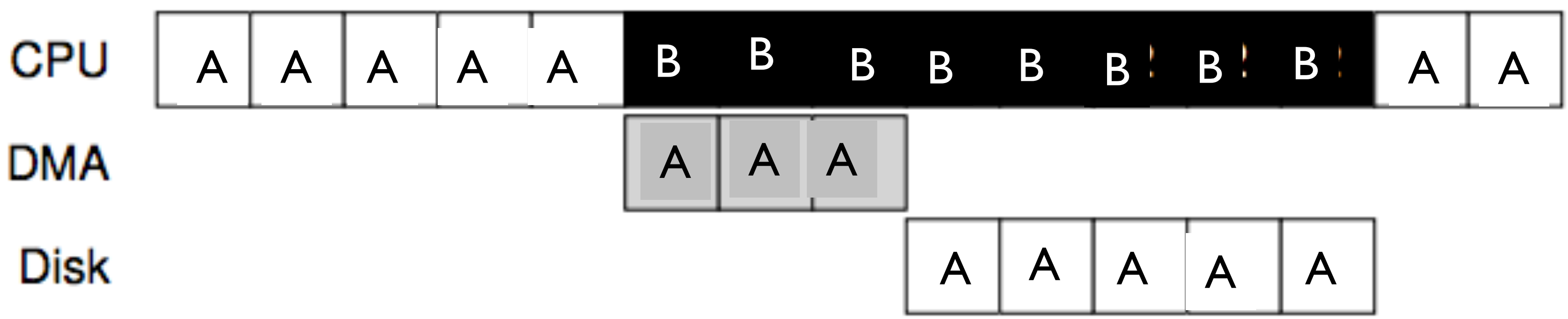
PIO (Programmed I/O):

- CPU directly tells device what the data is

DMA (Direct Memory Access):

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

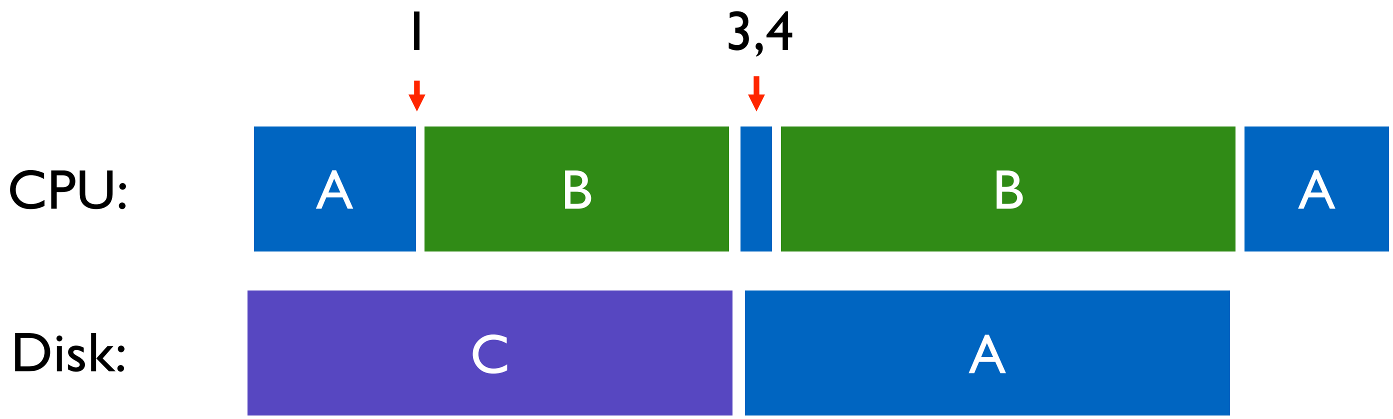
DATA TRANSFER WITH DMA




```
while (STATUS == BUSY)              // 1
      wait for interrupt;
```

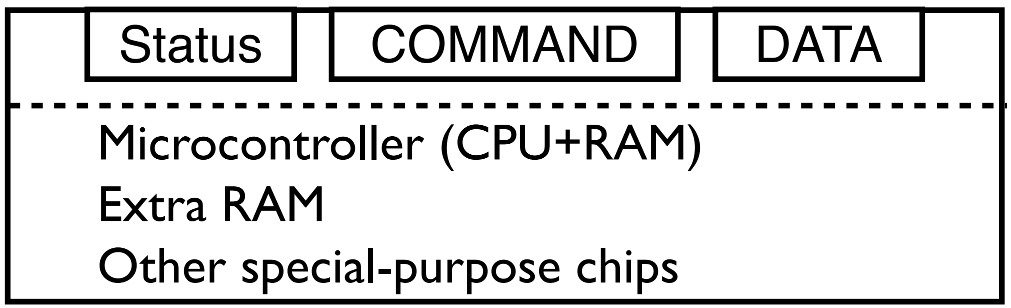
~~Write data to DATA register // 2~~

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




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

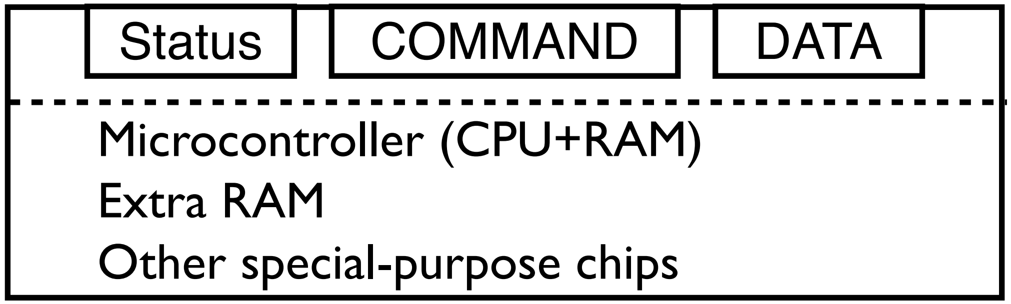
PROTOCOL VARIANTS



~~Status checks: polling vs. interrupts~~

~~PIO vs DMA~~

Special instructions vs. Memory mapped 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

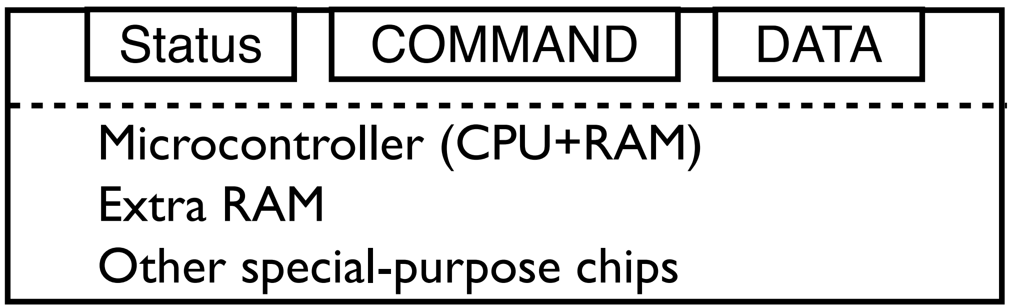
- each device has a port
- in/out instructions (x86) communicate with device

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~~

VARIETY IS A CHALLENGE

Problem:

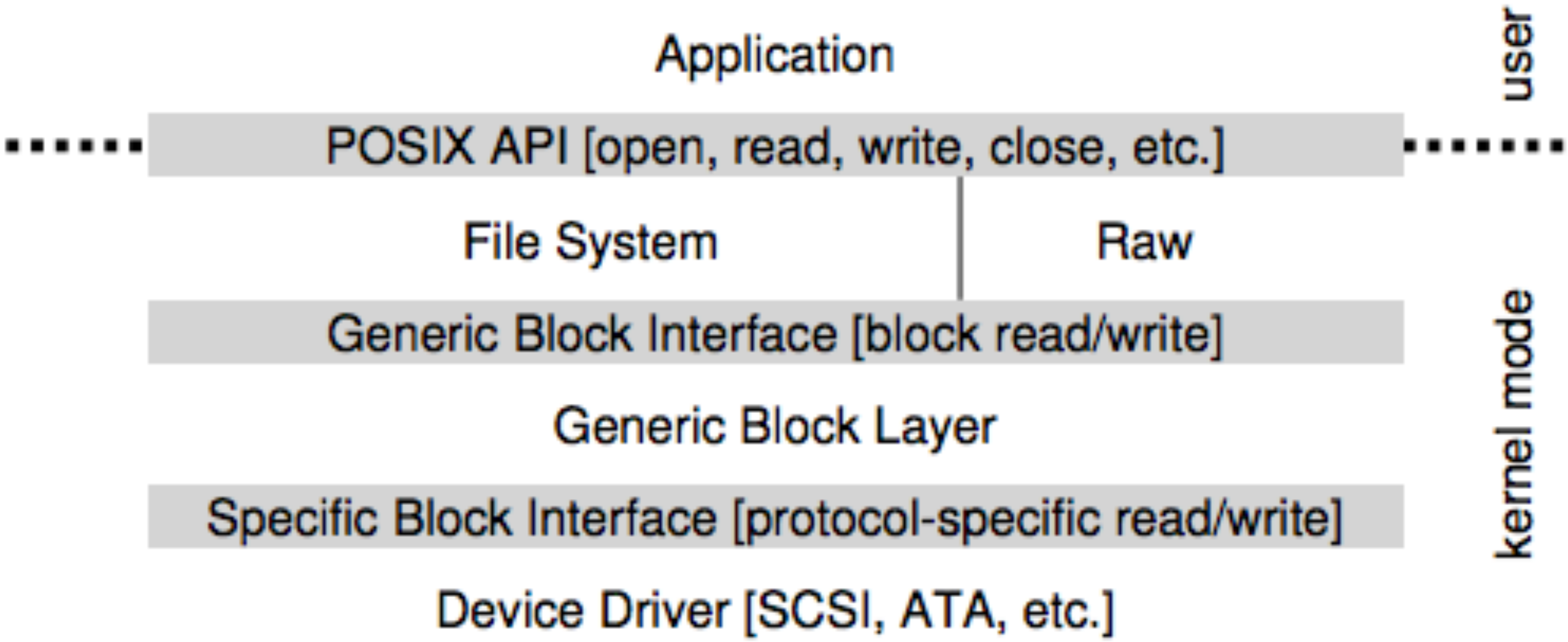
- many, many devices
- each has its own protocol

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

DEVICE DRIVERS



HARD DISKS

HARD DISK INTERFACE

Mechanical (slow) nature of HDDs makes management "interesting"

Disk has a sector-addressable address space

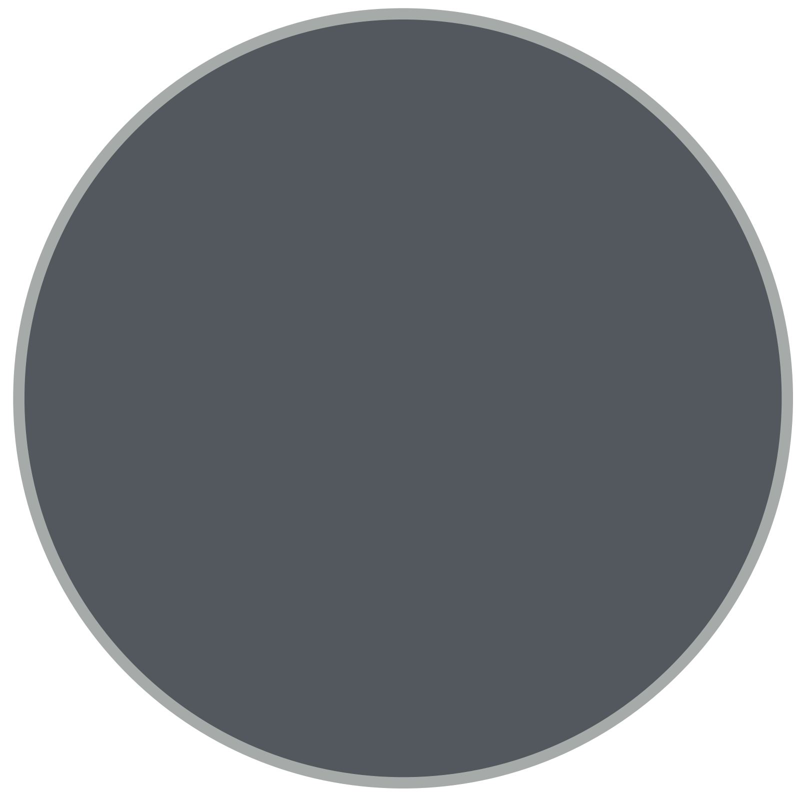
- Appears as an array of sectors
- Sectors are typically 512 bytes

Main operations:

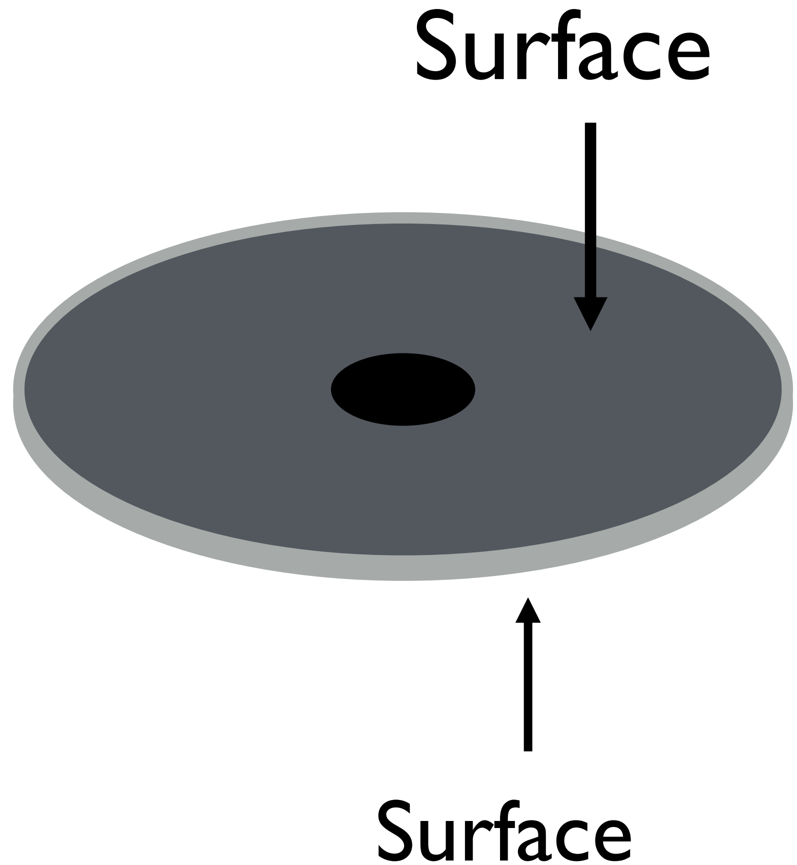
- reads + writes to sectors

DISK COMPONENTS

Platter



Spindle



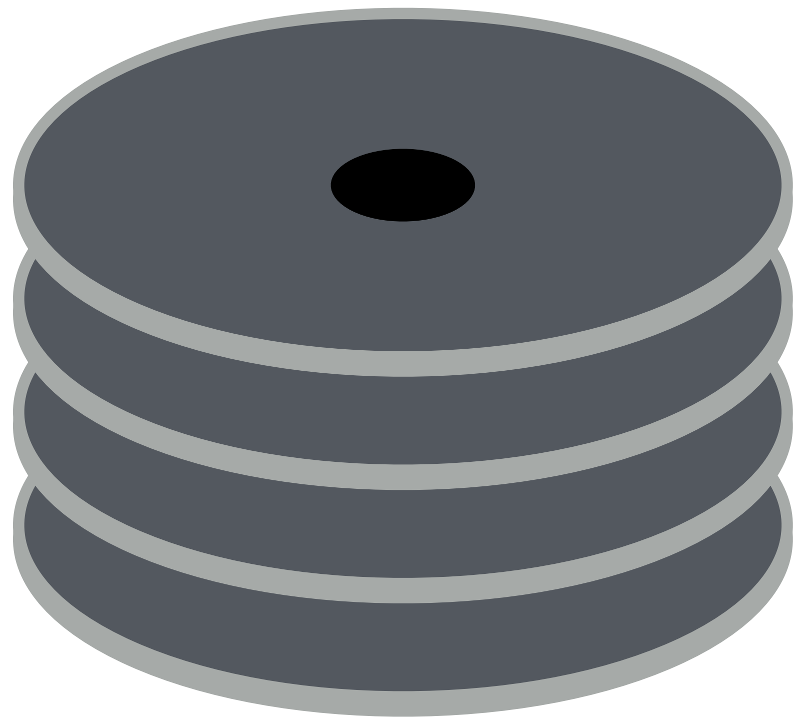
RPM?

Many platters may be bound to spindle

Motor connected to spindle spins platters

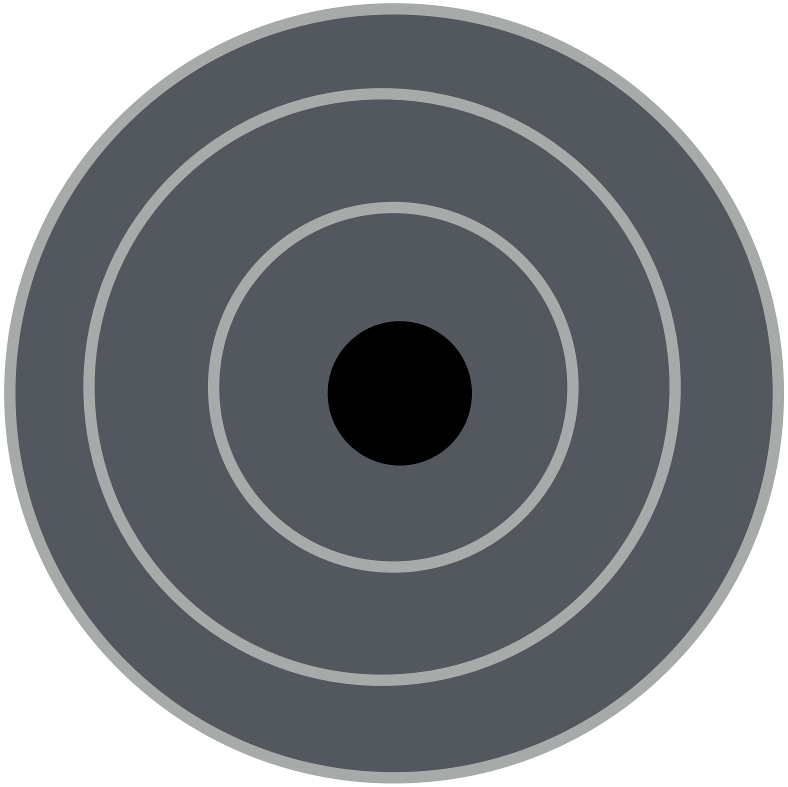
Rate of rotation: RPM

10000 RPM → single rotation is 6 ms



Surface is divided into rings: tracks

Stack of tracks(across surfaces): cylinder

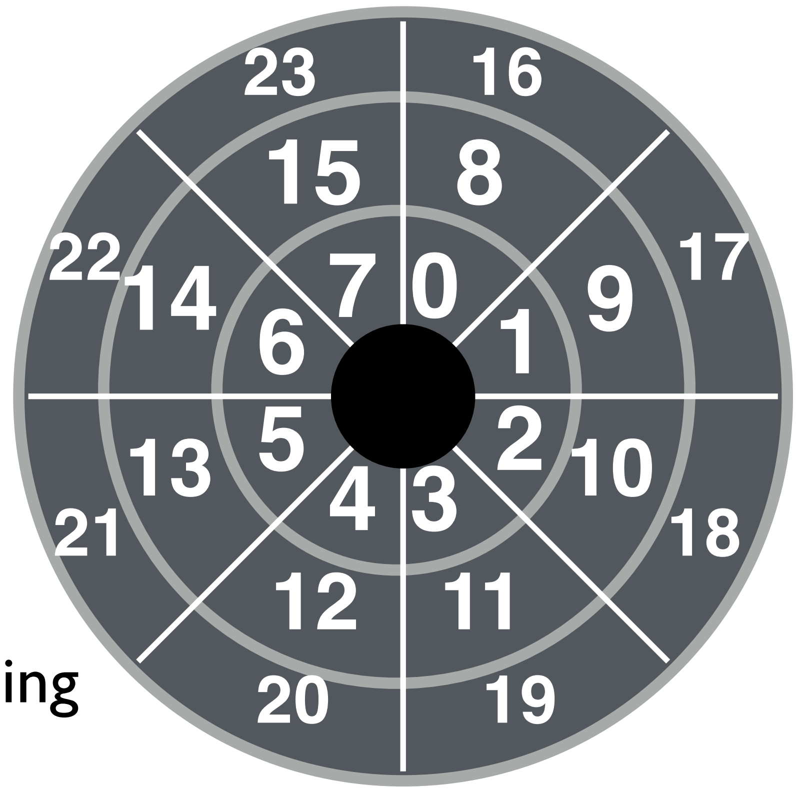


Tracks are divided into numbered sectors

- OS views as linear array

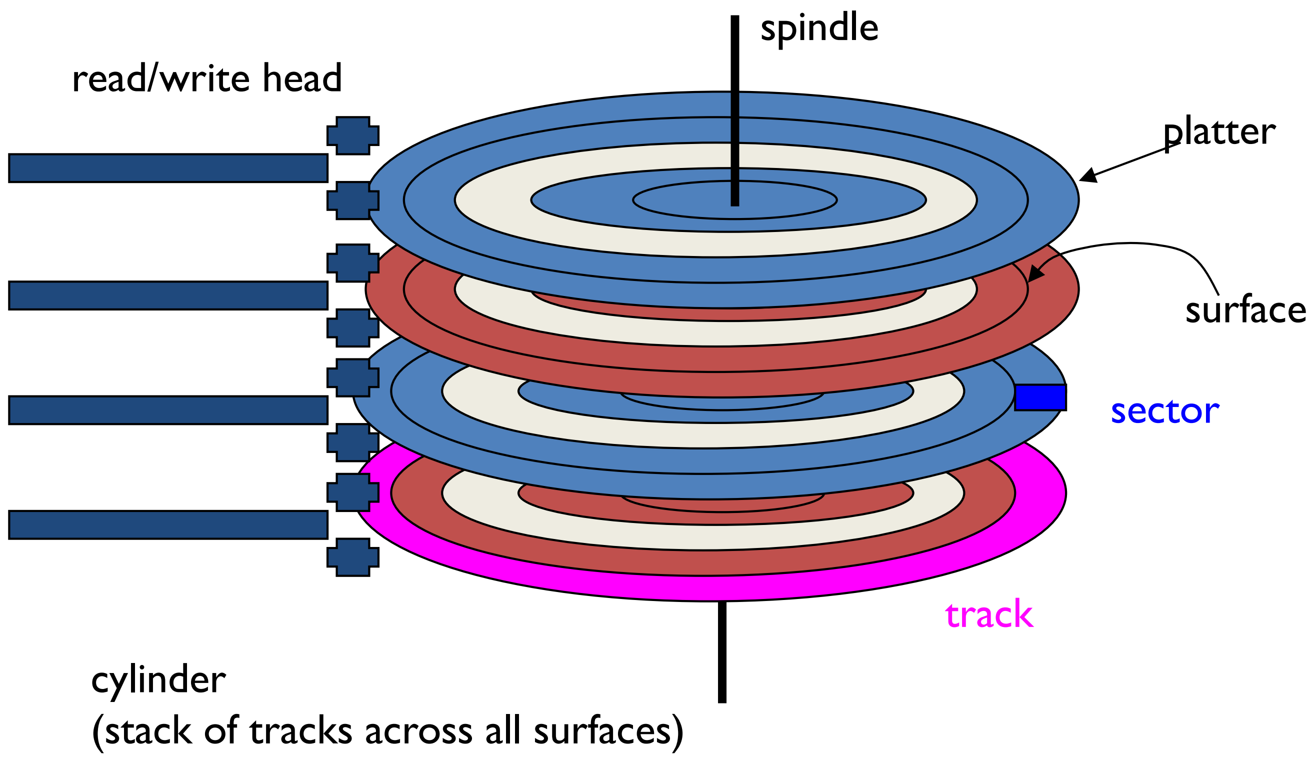
Actual mapping varies,

- uses knowledge of disk details,
- OS doesn't need to know mapping



Heads on a moving arm can read from each surface

DISK TERMINOLOGY

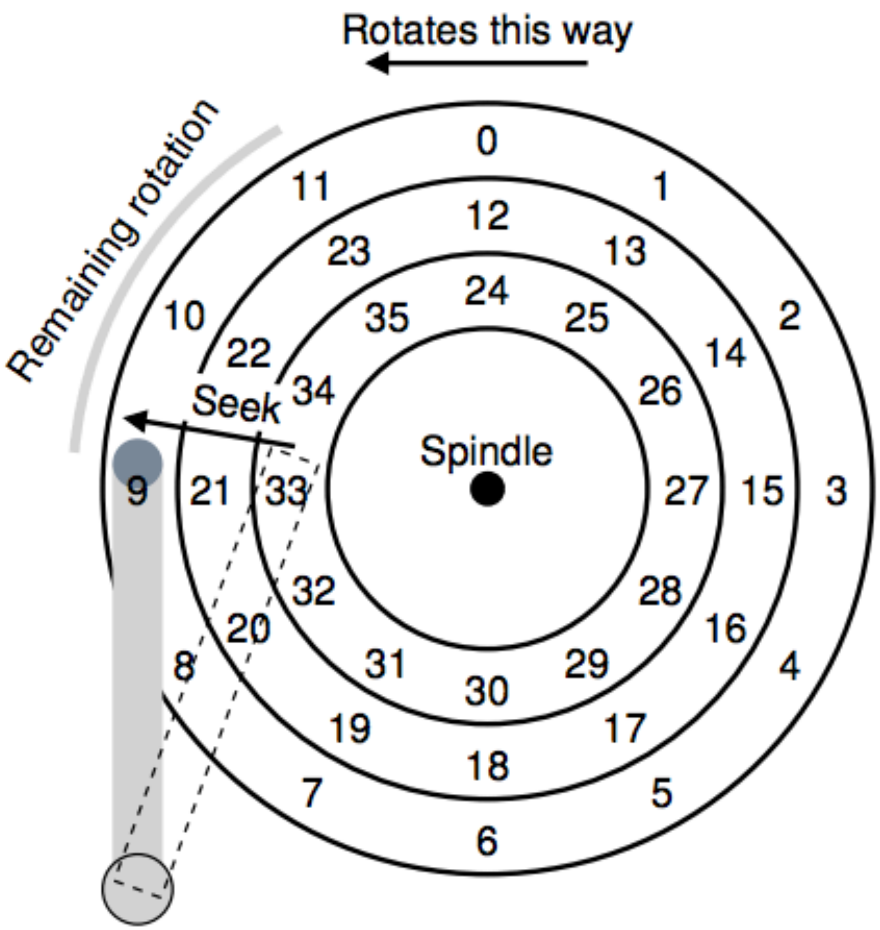


TIME TO READ/WRITE

Three components:

Time = seek + rotation + transfer time

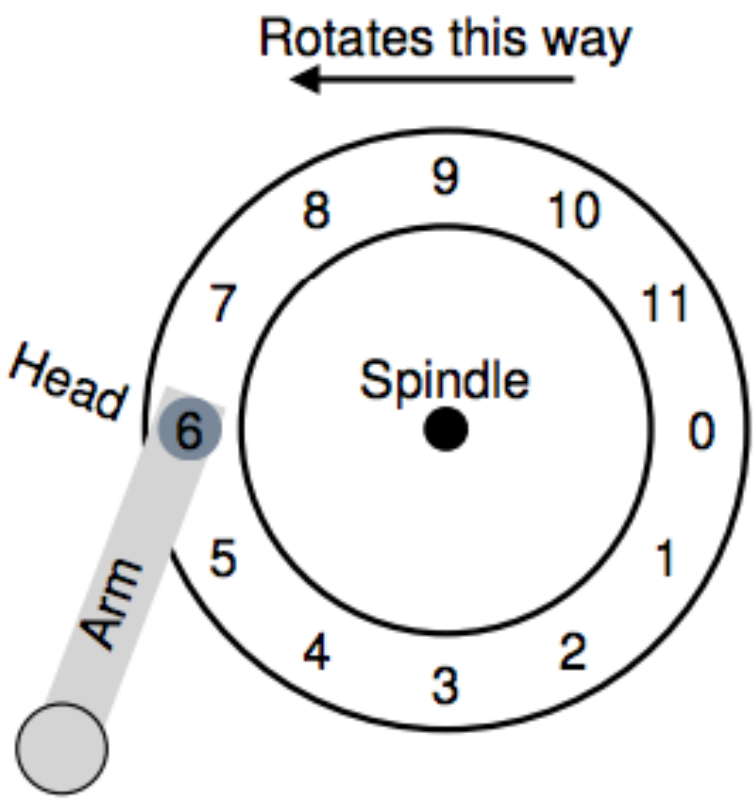
READING DATA FROM DISK



Example: Read sector 1

1) Seek

READING DATA FROM DISK



Example: Read sector 1

2) Rotational delay

3) Transfer time

SEEK, ROTATE, TRANSFER

Calculate average seek cost for random I/O

Seek cost: Function of cylinder distance

- Not purely linear cost
- (but can do a linear model for calculations in this course)

Must accelerate, coast, decelerate, settle

- Settling alone can take 0.5 - 2 ms

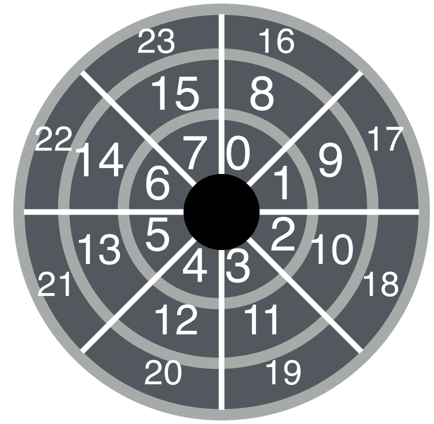
Entire seeks often takes several milliseconds

- 4 - 10 ms

Approximate average seek distance?

= 1/3 max seek distance

(derivation in text book)



SEEK, ROTATE, TRANSFER

Calculate average rotate cost for random I/O

Depends on rotations per minute (RPM)

- 7200 RPM is common, 15000 RPM is high end

With 7200 RPM, how long to rotate around?

1 / 7200 RPM = 1 minute / 7200 rotations =

1 second / 120 rotations =

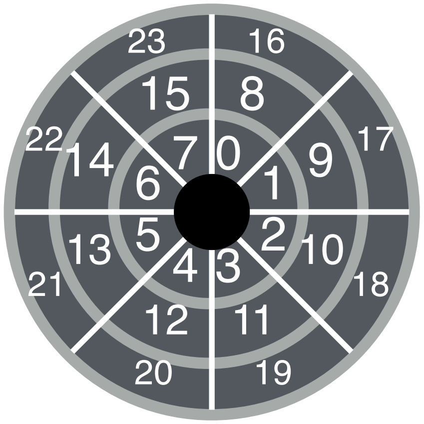
8.3 ms / rotation

Average rotation distance?

½

Average rotation?

8.3 ms / 2 = 4.15 ms



SEEK, ROTATE, TRANSFER

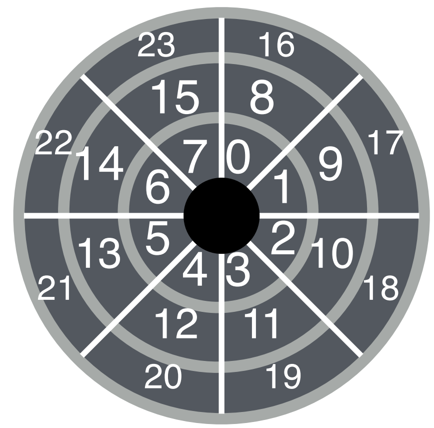
Calculate average transfer cost for random I/O (transfer cost always the same)

Pretty fast — depends on RPM and sector density

100+ MB/s is typical for **maximum transfer rate**

How long to transfer 512-bytes?

512 bytes * (1s / 100 MB) = 5 us



WORKLOAD PERFORMANCE

So…

- seeks are slow (ms)
- rotations are slow (ms)
- transfers are fast (us)

What kind of workload is fastest for disks?

Sequential (access sectors in order) vs. Random (access sectors in random order)?

Sequential: fast (no seek or rotation; transfer dominated)

Random: slow (seek+rotation dominated)

DISK SPEC: SEQ VS RANDOM THROUGHPUT

	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?

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

DISK SPEC: RANDOM THROUGHPUT

	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?)

What is size of each random read?
Assume 16-KB reads

Throughput for average **random 16-KB** read w/ Cheetah?

	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

Time = Seek + rotation + transfer

Average seek? Seek = 4 ms

Average rotation in ms?

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

Transfer of 16 KB?

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

Throughput for average **random 16-KB** read w/ Cheetah?

	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

Time = Seek + rotation + transfer

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

Random Throughput? (MB/s)

$$\text{throughput} = \frac{16\text{ KB}}{6.1\text{ms}} \times \frac{1\text{ MB}}{1024\text{ KB}} \times \frac{1000\text{ ms}}{1\text{ sec}} = 2.5\text{ MB/s}$$

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

Throughput for average random 16-KB read on Barracuda?

Time = seek + rotation + transfer

Avg seek = 9ms

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

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

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

Throughput for average random 16-KB read on Barracuda?

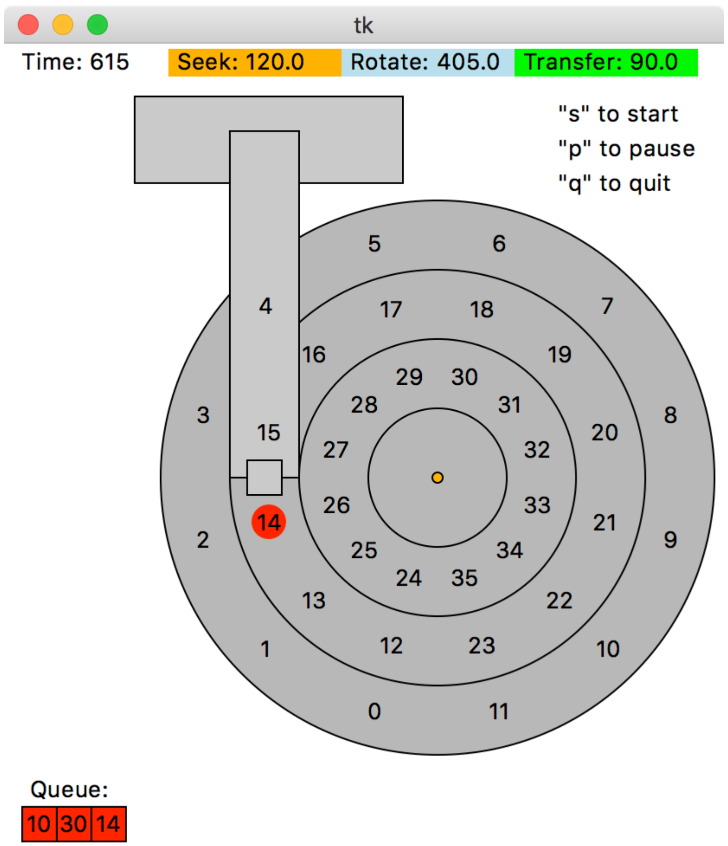
Barracuda time = 9ms + 4.1ms + 149us = 13.2ms

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

	Cheetah	Barracuda
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

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

DISK SIMULATOR



./example-rand.csh
./example-seq.csh

OTHER IMPROVEMENTS

Track Skew

Zones

Cache

PROBLEM

What if sequential request spans multiple tracks?

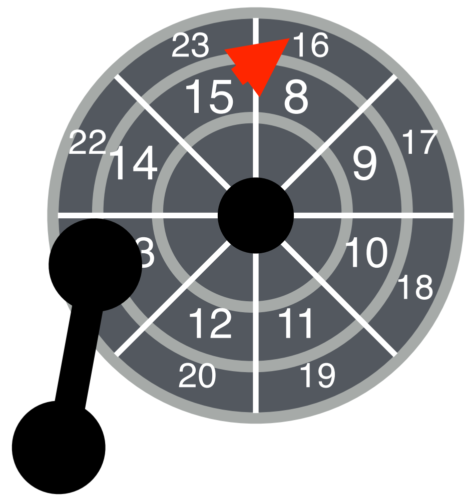
./example-skew.csh

./example-skew-fixed.csh

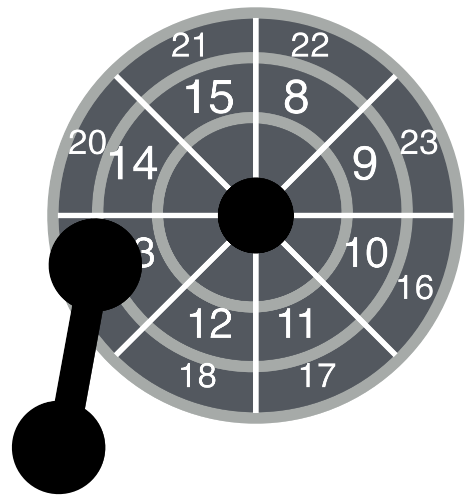
TRACK SKEW

Imagine sequential reading (start with sector 8…)
How should sectors numbers be laid out on disk?

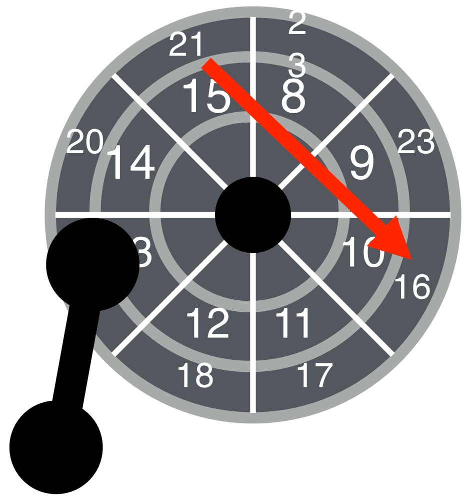
TRACK SKEW



TRACK SKEW



TRACK SKEW



OTHER IMPROVEMENTS

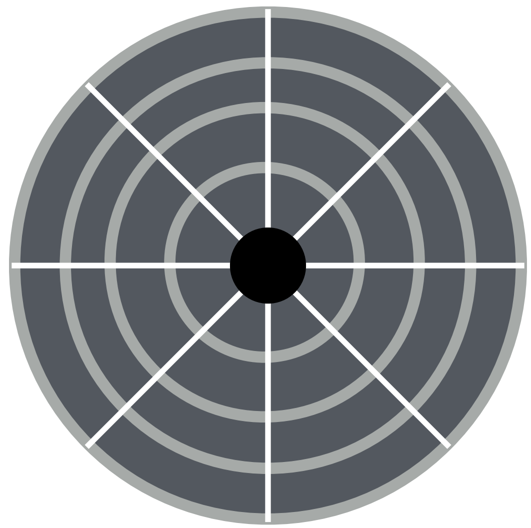
~~Track Skew~~

Zones

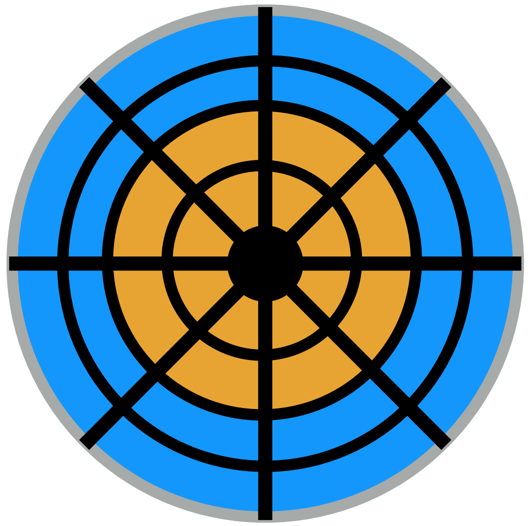
Cache

ZONES

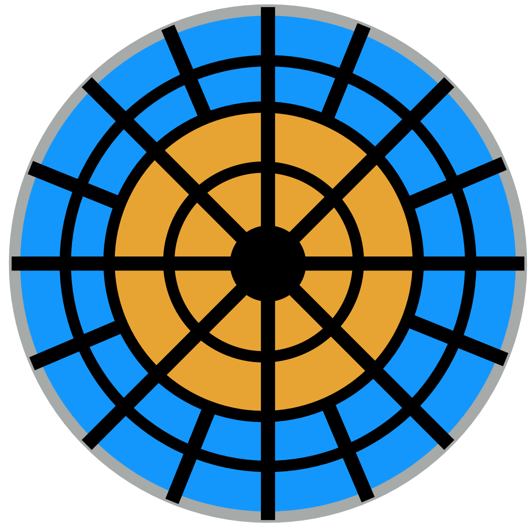
ZONES



ZONES



ZONES



ZBR (Zoned bit recording): More sectors on outer tracks
Goal: Constant density across tracks

DISK SIMULATOR: ZONES

Performance characteristics of ZBR?

Where do you want your data?

./example-zones-outer.csh

./example-zones-inner.csh

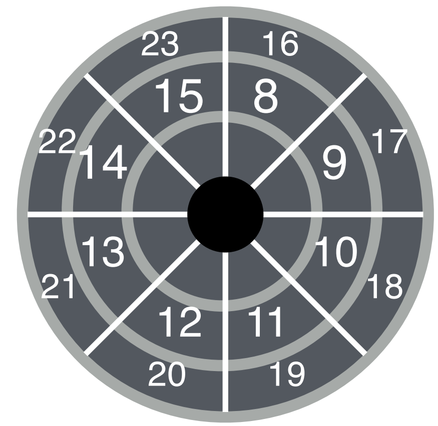
OTHER IMPROVEMENTS

~~Track Skew~~

~~Zones~~

Cache

DRIVE CACHE



Drives may cache both reads and writes

- OS caches file data too (later lecture)
- Disks contain ~16MB used as cache

What advantage does caching in **drive** have for reads?

Spatial locality -- Read-ahead: "Track buffer"

- Read contents of entire track into track buffer during rotational delay

What advantage does caching in **drive** have for writes?

Write caching with volatile memory (i.e., not persistent)

- Immediate reporting: Claim written to disk when not
- Data could be lost on power failure

Drive Cache: Buffering

Tagged command queueing (TCQ)

- Have multiple outstanding requests to the disk
- Disk can reorder (schedule) requests for better performance

I/O SCHEDULERS

I/O SCHEDULERS

Given stream of I/O requests, in what order should they be served?

Goal: Minimize seek + rotation time

Much different than CPU scheduling

Position of disk head relative to request position matters more than length of job

IMPACT OF DISK SCHEDULING?

Assume seek+rotate = 10 ms for random request

How long (roughly) does the below workload take?

- Requests are given in sector numbers

300001, 700001, 300002, 700002, 300003, 700003

FCFS:

Best possible:

SIMULATOR

./example-sched-fifo.csh

What would be a better algorithm than FIFO?

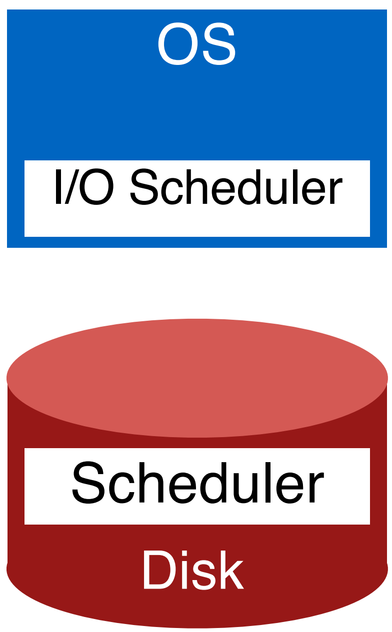
./example-sched-sstf.csh

./example-rotate.csh

./example-rotate-sptf.csh

./example-rotate-question.csh

I/O SCHEDULERS



Where should the I/O scheduler go?

SPTF (SHORTEST POSITIONING TIME FIRST)

Strategy: choose request w/ least **positioning time** (seek + rotation)

How to implement in **disk**?

- Greedy algorithm (just looks for best NEXT decision)

How to implement in **OS**?

- Use Shortest Seek Time First (SSTF) instead
- Approximate by scheduling by sector number

Easy for far away requests to starve

STARVATION

./example-starve.csh

AVOID STARVATION: SCAN

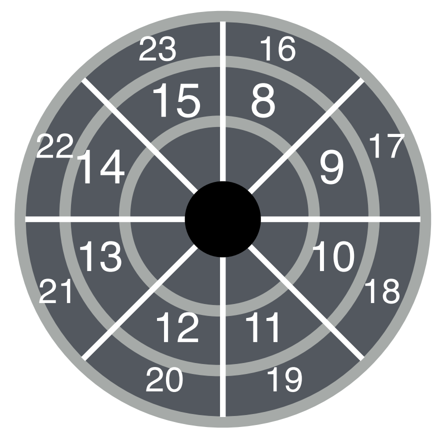
Elevator Algorithm (or Scan)

- Sweep back and forth, from one end of disk to other, and back
- Serve requests as pass that cylinder in each direction
- Sorts by cylinder number; ignores rotation delays

Disadvantage?

Better: C-SCAN (circular scan)

- Only sweep in one direction



ANOTHER APPROACH: BOUNDED WINDOW

Much schedule all requests in one window before moving to next

./example-starve-bsatf.csh

What is the impact of different window sizes?

./example-bsatf-w1.csh

./example-bsatf-wall.csh

IS SPTF BEST POSSIBLE?

Compare time for identical workloads…

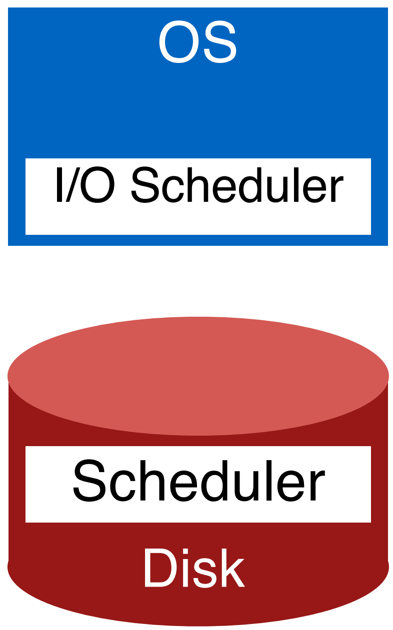
./example-greedy-satf.csh

./example-optimal.csh

Not computationally feasible to determine optimal schedule (even if know all future requests)

Even worse since don't know which requests will arrive next…

I/O SCHEDULERS

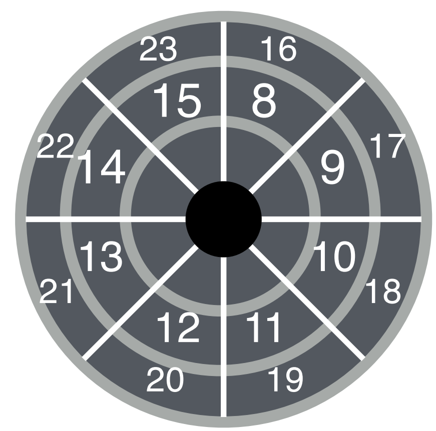


Where should the I/O scheduler go?

WHAT HAPPENS AT OS LEVEL?

Assume 2 processes A and B each call read() with 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);
    }
}
```



Stream of requests seen by disk: ABABABA

WORK CONSERVATION

Work conserving schedulers always do work if work exists

- Principle applies to any type of scheduler (CPU too)

Could be better to wait if can **anticipate** another request will arrive

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

- Keeps resource idle while waiting for future requests

Better stream of requests for OS to give disk: AAAAABBBBBAAAAAA

I/O DEVICE SUMMARY

Overlap I/O and CPU whenever possible!

- Use interrupts, DMA

Storage devices provide common **block interface**

On a disk: Never do random I/O unless you must!

- Quicksort is a terrible algorithm on disk

Spend time to schedule on slow, stateful devices

Next: Other storage devices (RAIDs and SSDs/flash)