

Based on slides by Harsha V. Madhyastha

# EECS 482 Introduction to Operating Systems

Spring/Summer 2020

Lecture 16: Storage devices

Nicole Hamilton

[https://web.eecs.umich.edu/~nham/  
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

# Agenda

1. Grader2 is down.
2. Project 3, due July 27.
3. Storage devices.

# Agenda

1. Grader2 is down.
2. Project 3, due July 27.
3. Storage devices.

# Grader2 website is down.

That includes the autograder.

No estimated time when it will be fixed.

EECS tech support initially thought it was an expired SSL certificate.

Today, they reported the certificate is okay and there are no other obvious reasons why it's down.

Second level ITS support has been called in.

# Agenda

1. Grader2 is down.
2. **Project 3, due July 27.**
3. Storage devices.

# Project 3

Write test case for every transition in your state machine.

Even without handling `fork()`, test with multiple processes to test `vm_destroy`.

Run in multiple terminals or in background.

Call `fork()` before any calls to `vm_map`.

filename argument to `vm_map` is a user-level virtual address.

# Agenda

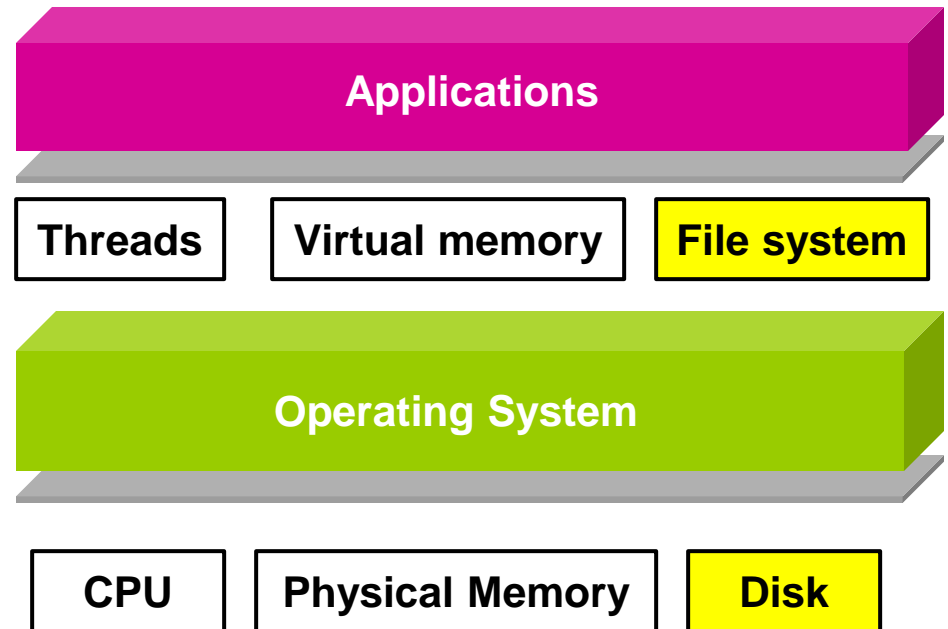
1. Grader2 is down.
2. Project 3, due July 27.
3. **Storage devices.**

# The file system

In this section of the course, we'll discuss how the filesystem works.

What interface does the file system export to applications?

How does file system interact with hardware?





# Reality vs. Abstraction

## OS abstraction

Large set of files, with rich naming convention.

Same interface to files, irrespective of hardware.

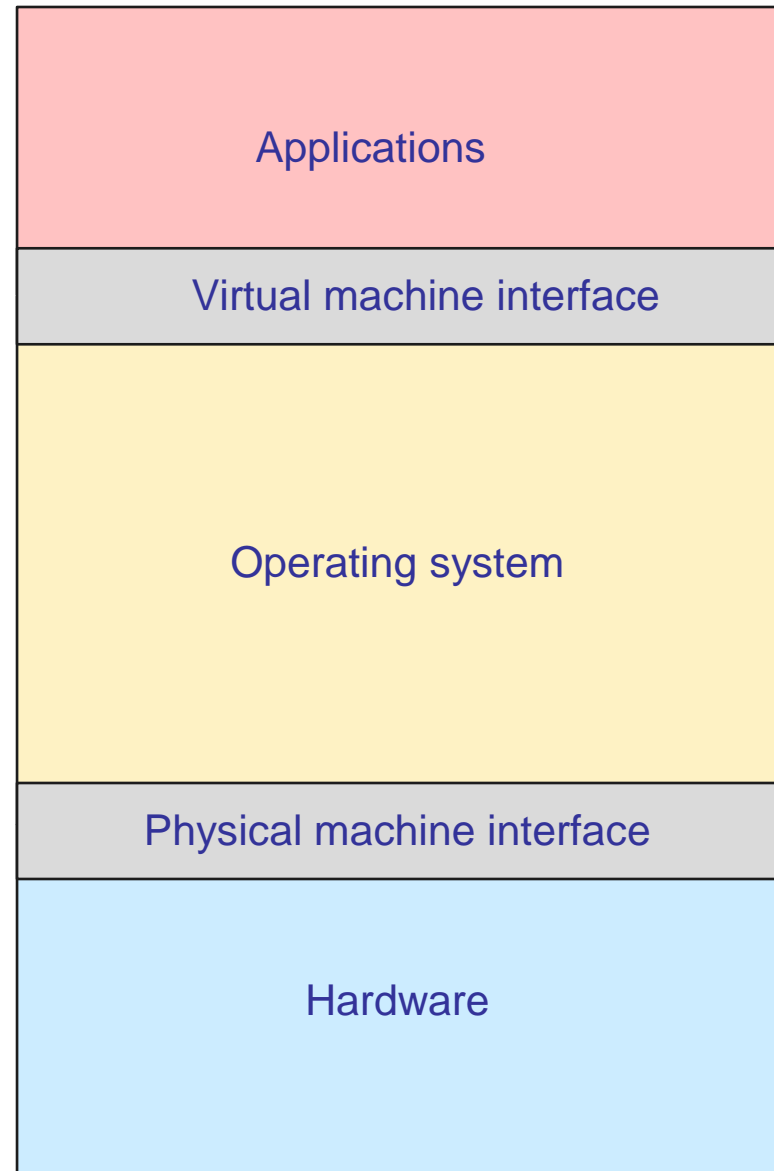
Fast, and crash consistent.

## Hardware interface

Small set of disks, with array of blocks on each disk.

Interface varies across disks.

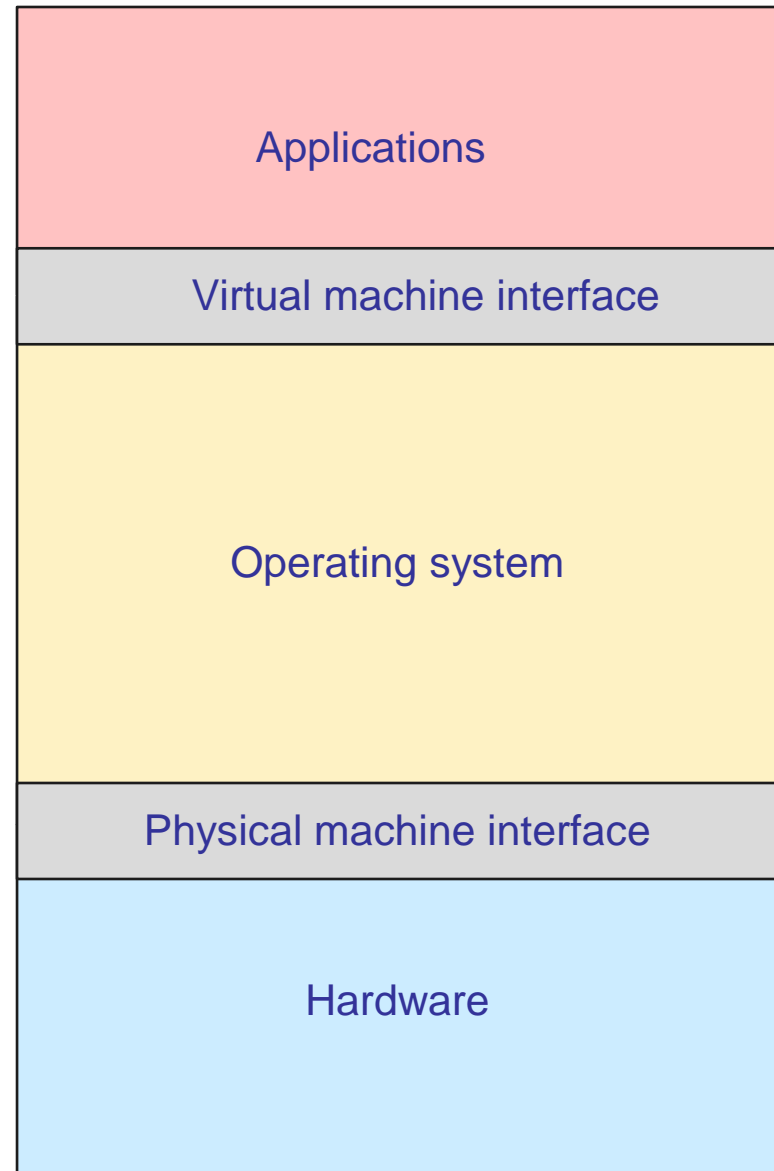
Slow, and potentially inconsistent on crash.



# Dealing with heterogeneity

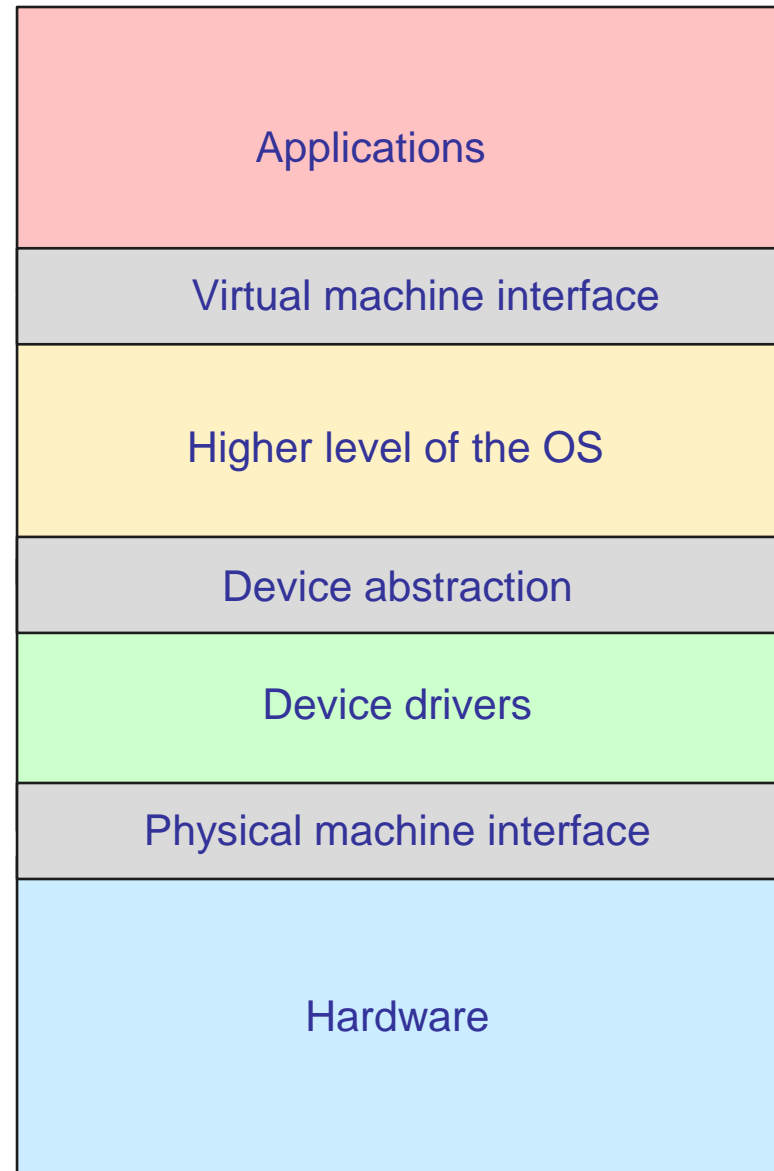
Many different types of disks and other devices and lots of different interfaces, e.g., ESDI, USB, SCSI, SATA, Fiber channel, m.2.

Need a way of managing this diversity.



# Dealing with heterogeneity

Solution is to add a device driver abstraction inside the operating system to hide the differences between similar classes of devices.

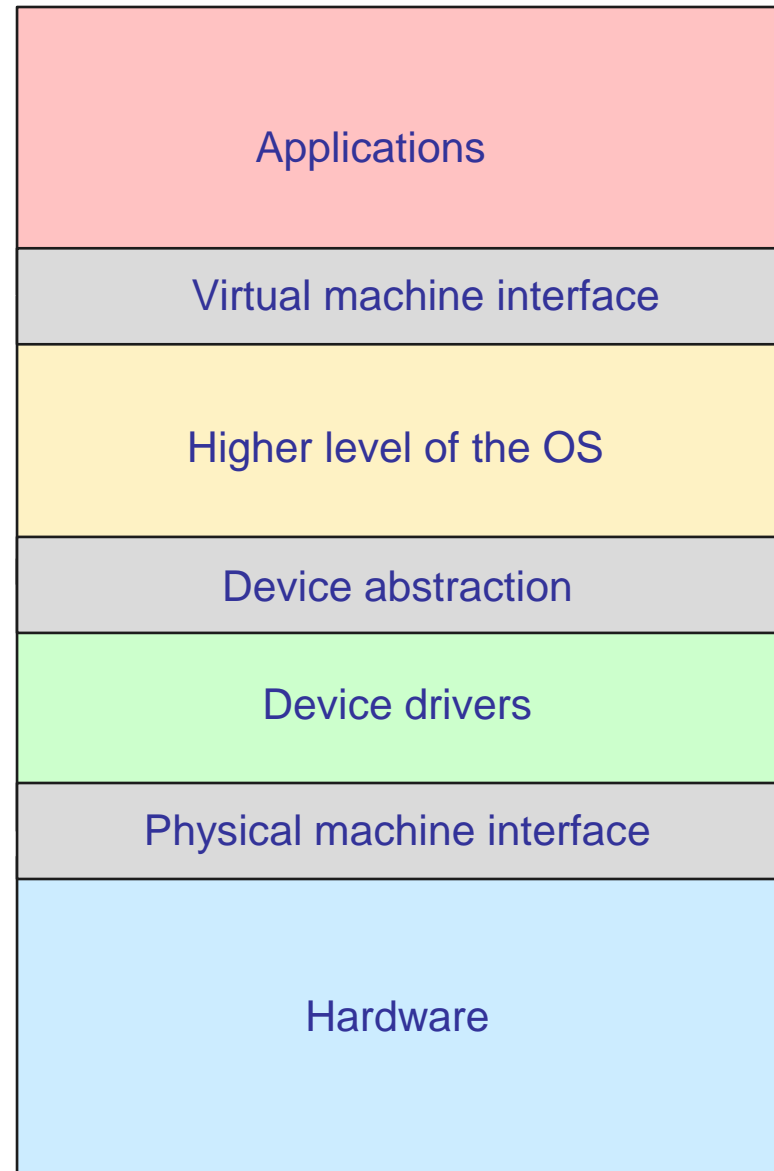


# Dealing with heterogeneity

Device drivers are usually supplied by the device manufacturers.

Because they run as a trusted part of the kernel, in the past, they've been a major reason for Windows crashes.

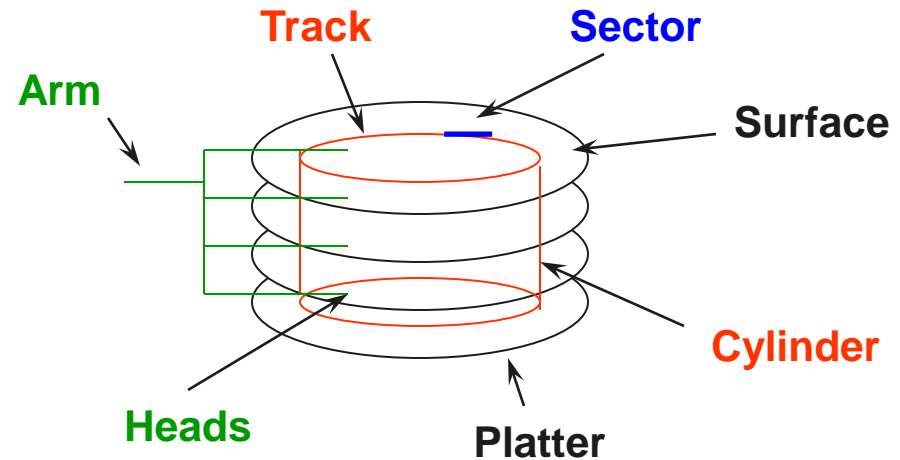
Active area of research into how to contain device driver problems.



# Physical Hard Disk Structure

## Disk components

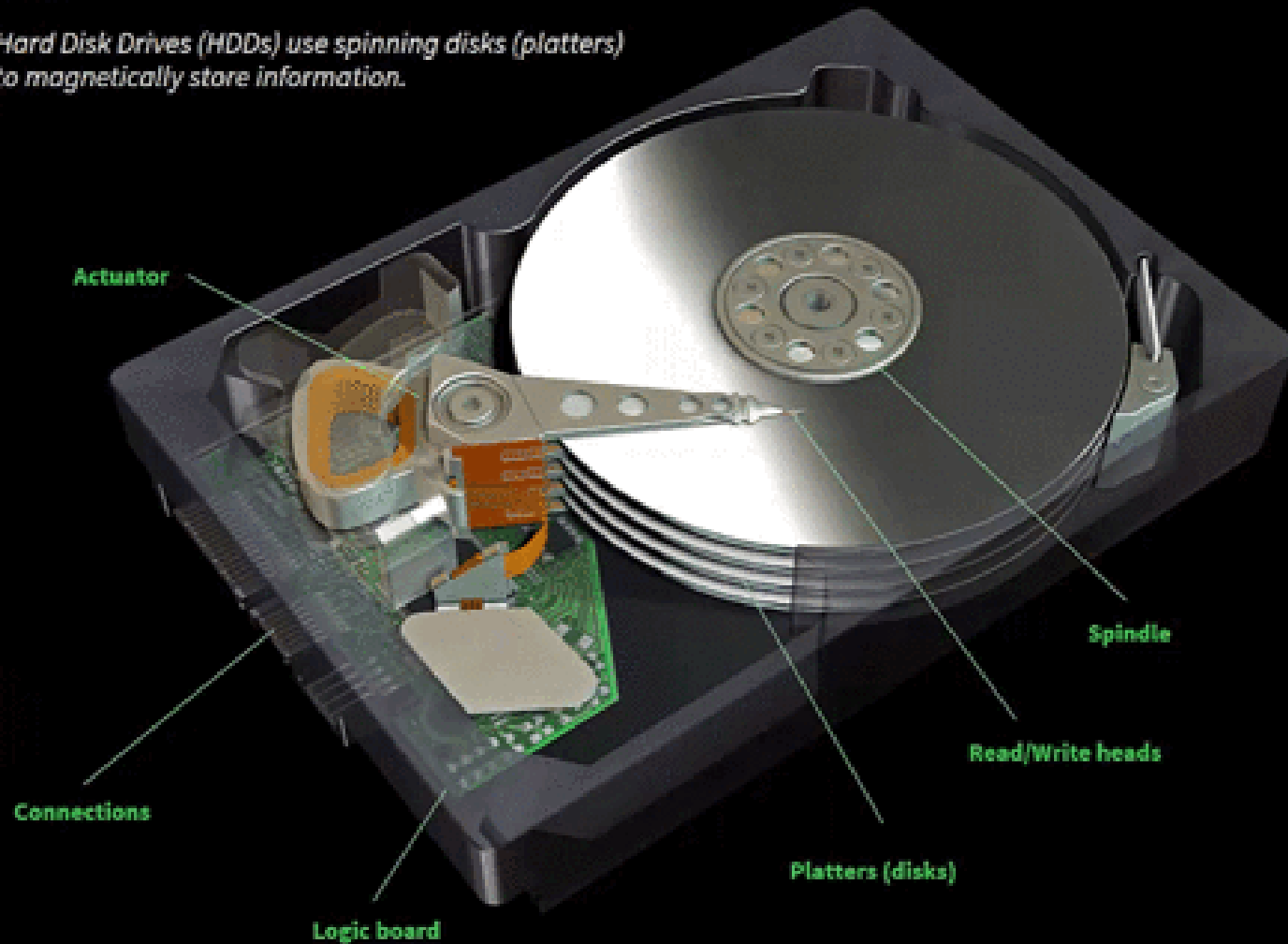
- ◆ Platters
- ◆ Surfaces
- ◆ Tracks
- ◆ Sectors
- ◆ Cylinders
- ◆ Arm
- ◆ Heads



# How Hard Disk Drives Work

Created in partnership with  
 SEAGATE

*Hard Disk Drives (HDDs) use spinning disks (platters) to magnetically store information.*




Shouting in the Datacenter - You x +

youtube.com/watch?v=tDacjrSCeq4

Seattle PI WSJ NY Times WaPost Canvas Course Info Center Crappy Scheduler BofA Other bookmarks

YouTube shouting in the datacenter



0:00 / 1:59

Shouting in the Datacenter

1,732,871 views • Dec 31, 2008

10K 122 SHARE SAVE

The image shows a screenshot of a YouTube video player. The video is titled "Shouting in the Datacenter" and has 1,732,871 views as of December 31, 2008. The video content shows a man with dark hair and a goatee, wearing a yellow t-shirt, standing in a server room. He has a frustrated or angry expression on his face. The background is filled with server racks, cables, and other equipment. The video player interface includes a search bar with the text "shouting in the datacenter", a play button, a progress bar at 0:00 / 1:59, and interaction buttons for likes (10K), comments (122), share, and save.

<https://www.youtube.com/watch?v=tDacjrSCeq4>

# Hard Disk Performance

What does disk performance depend upon?

**Queue** Wait for the disk to be free.

**Positioning** Move the disk arm to the correct cylinder and rotate to the right sector.

**Access** Transfer data from/to disk.

For given load, performance depends on

Positioning overhead, called seek time, ~1 to 10 ms.

Transfer time, ~100 MBps.



# Disks Heterogeneity

## Seagate Barracuda 3.5" ([workstation](#))

capacity: 250 - 750 GB

rotational speed: 7,200 RPM

sequential read performance: 78 MB/s (outer) - 44 MB/s (inner)

seek time (average): 8.1 ms

## Seagate Cheetah 3.5" ([server](#))

capacity: 73 - 300 GB

rotational speed: 15,000 RPM

sequential read performance: 135 MB/s (outer) - 82 MB/s (inner)

seek time (average): 3.8 ms

## Seagate Savvio 2.5" ([smaller form factor](#))

capacity: 73 GB

rotational speed: 10,000 RPM

sequential read performance: 62 MB/s (outer) - 42 MB/s (inner)

seek time (average): 4.3 ms

# Optimizing I/O performance

To increase performance of slow I/O devices:

Avoid doing I/O (Disks are *sloooooow!*)

Reduce overhead (minimize positioning time)

Amortize overhead over larger requests

Efficiency = transfer time / (seek time + transfer time)

Rule of thumb: Achieve at least 50% efficiency

Example: 5 ms average seek time and 100MBps transfer rate → Read at least 500KB

# Disk scheduling

Reduce overhead by reordering requests.

Can be implemented in OS or hardware.

## Tradeoffs?

The hardware knows more about the device itself, bad blocks, error handling, what's really likely to be faster, and can offload work from the OS.

The OS knows more about the application needs.

# FCFS

Pick 1 of n requests in queue:

Example: 98, 183, 37, 122, 14, 124, 65, 67

Start track is 53

**FCFS** (first come, first served)

98, 183, 37, 122, 14, 124, 65, 67

Total head movement: 640 tracks

# SSTF (STCF)

Pick 1 of n requests in queue:

Example: 98, 183, 37, 122, 14, 124, 65, 67

Start track is 53

**SSTF** (shortest seek time first)

65, 67, 37, 14, 98, 122, 124, 183

Total head movement: 236 tracks

**Any drawbacks?**

Potential starvation. Some tracks may never get served.

# SCAN (Elevator)

Pick 1 of n requests in queue:

Example: 98, 183, 37, 122, 14, 124, 65, 67

Start track is 53

**SCAN** (like windshield wipers)

37, 14, 65, 67, 98, 122, 124, 183

Total head movement: 208 tracks

**Drawbacks and fix?**

Blocks in the middle served more often than at ends.

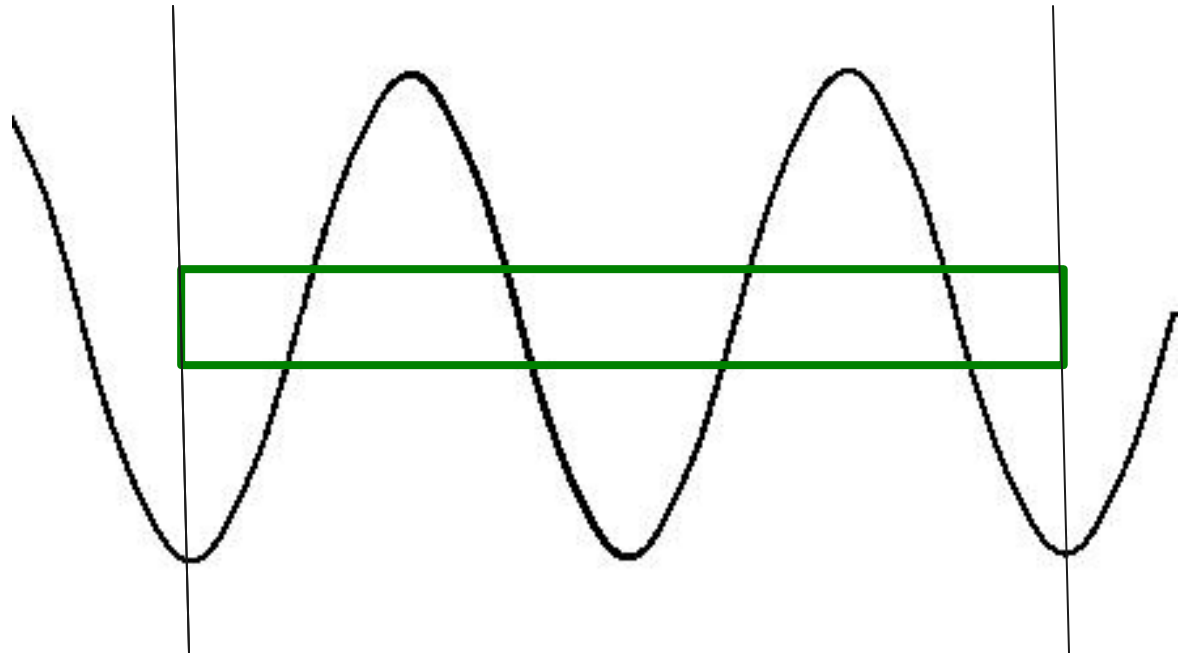
**Circular SCAN** (C-SCAN): Serve requests only in one direction

# SCAN

Head position

Consider two cycles.

Over 2 cycles, the head enters the middle 4 times.



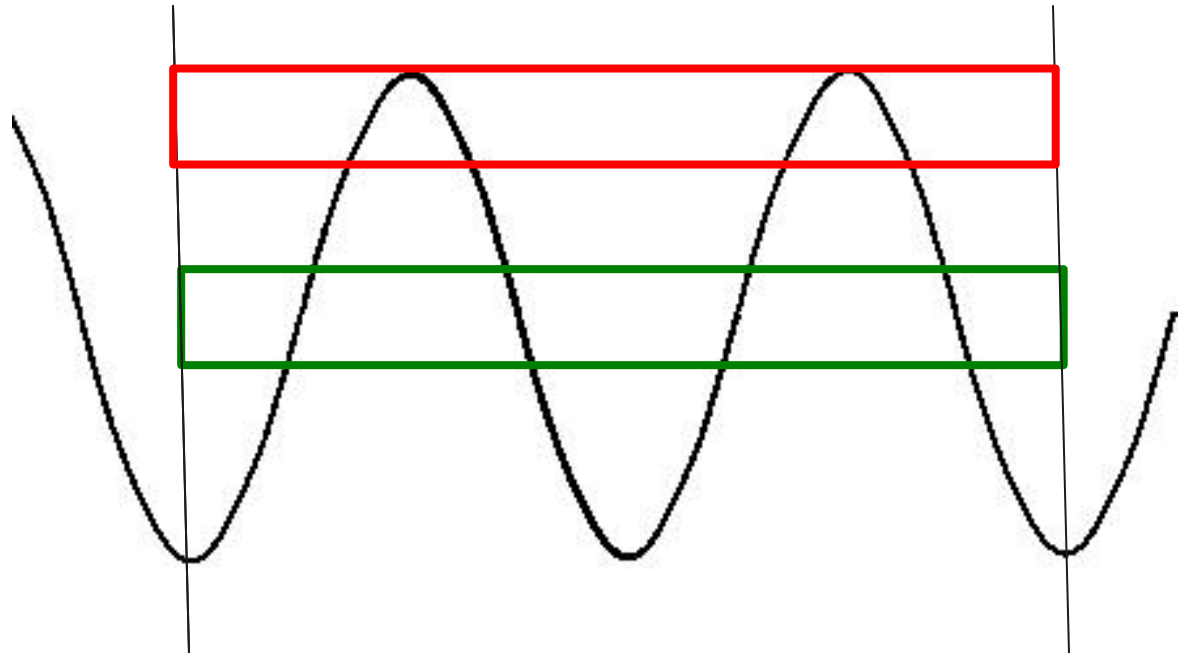
# SCAN

Head position

Consider two cycles.

Over 2 cycles, the head enters the middle 4 times.

But it enters a given peak only twice.





# SCAN vs. SSTF

SCAN typically has better throughput

Minimizing total head movement

SSTF may have better response time

Servicing fastest request first

But, poor throughput can cause longer queue waits

Disk scheduling: Queueing vs. positioning

Does CPU scheduling affect throughput?

# Anticipatory scheduling

Consider two processes with disk locality

P1: read 1, compute, read 2, compute, read 3, ....

P2: read 1001, compute, read 1002, compute, ....

What behavior will SSTF give?

1, 1001, 2, 1002, 3, ... (suboptimal)

Key idea: wait for a bit for new request

1, 2, 3, (timer expires), 1001, 1002

Can improve throughput **and** response time

# Optimizing data layout

Keep related items together on disk, e.g., on the same track or same cylinder.

**What items will be accessed together?**

Can guess based on general usage patterns.

- Blocks in same file often accessed together.

- Files in same directory often accessed together.

- Files often accessed with their directory.

Can guess based on past accesses of data.

- Learn patterns and reorganize data on disk.

# Flash (solid state disks)

Optimizations depend on specifics of a device.

Flash differs from magnetic disk.

1. Better random read performance.
2. Lower positioning overhead.
3. Much faster transfer rates (m.2).
4. Starting to yield more read parallelism.
5. Lower power.
6. Better shock resistance.
7. But they experience *wearout*, a limited number of times a cell can be rewritten.

OS hides physical characteristics of device from applications.

# Optimizing for Flash

Move data blocks to do wear leveling.

Write data in big blocks.

Asynchronously erase blocks.

Prefer to read data rather than write.

# Next time ...

File system interface and structure