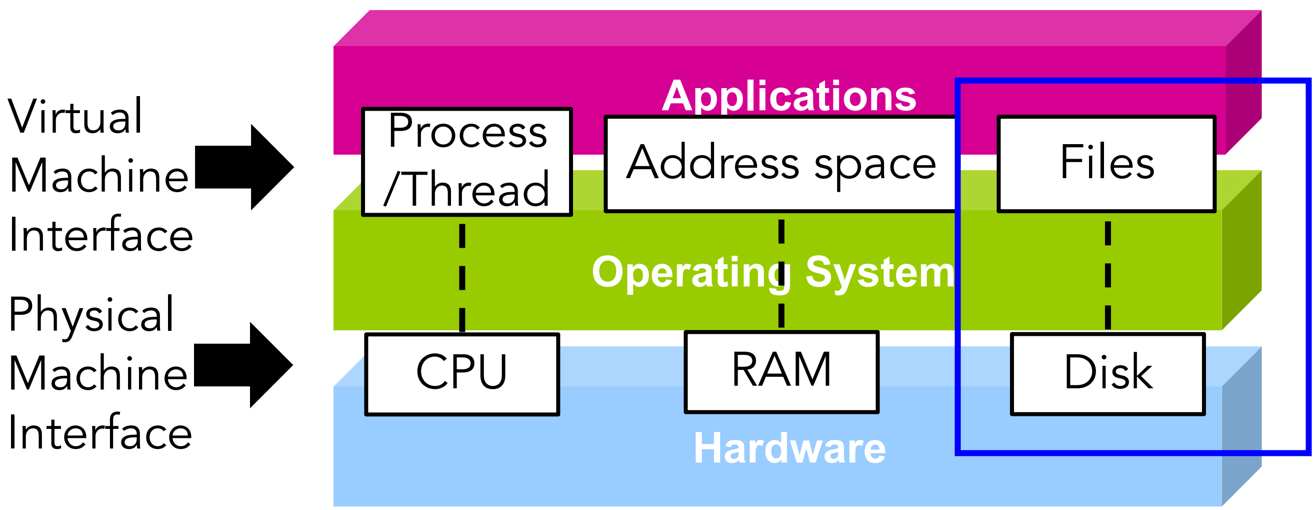
Winter 2025

EECS 482: Introduction to Operating Systems

Lecture 18: Disk

Prof. Ryan Huang

OS abstractions



File system abstractions

Hardware reality	**OS abstraction**
Heterogenous interfaces	Uniform interface
A few storage objects (disks)	Many storage objects (files)
Simple names (numeric)	Rich naming structure (symbolic, hierarchical, unified)
Slow	Fast
Inconsistent on crash	Consistent on crash

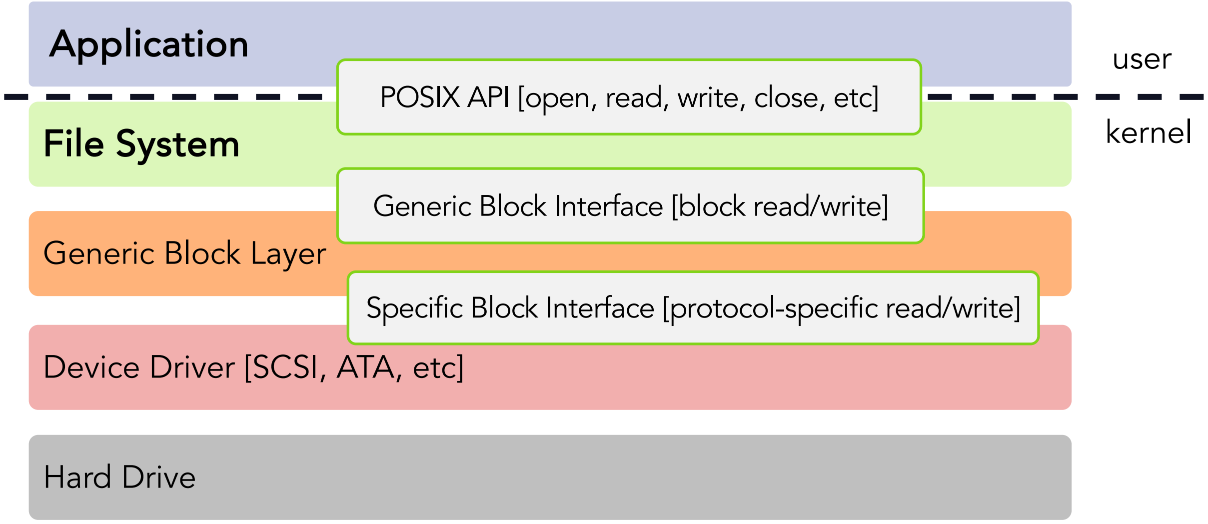
Dealing with heterogeneity

Many I/O devices, each has its own idiosyncrasy

- How to avoid writing a slightly different OS for each H/W?

Solution: abstraction

- Build a common interface
- Write device driver for each device
  - Drivers are 70% of Linux source code



Disk interface

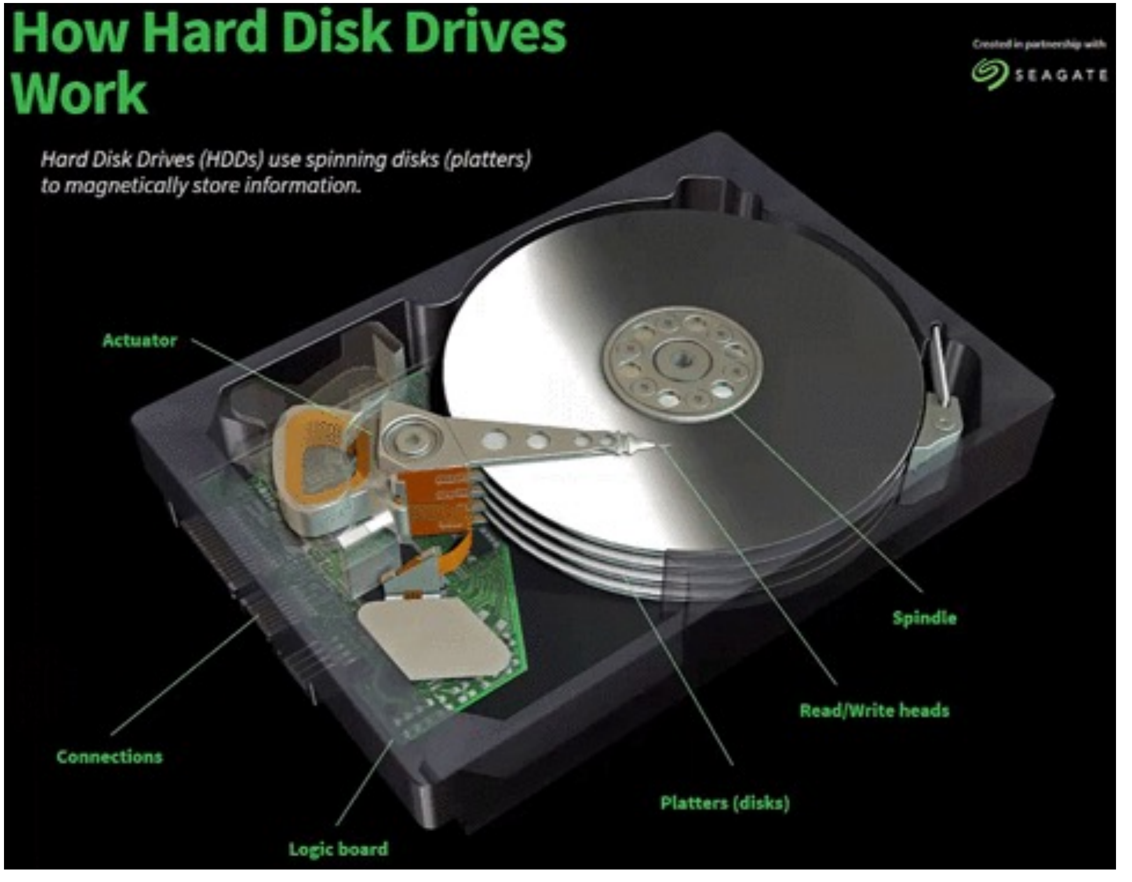
Disk interface presents linear array of sectors

- A sector is typically 512 bytes in size
- Written atomically (even if there is a power failure)
- 4 KiB in "advanced format" disks

Disk maps logical sector #s to physical sectors

- OS doesn't know logical to physical sector mapping

Magnetic hard drive



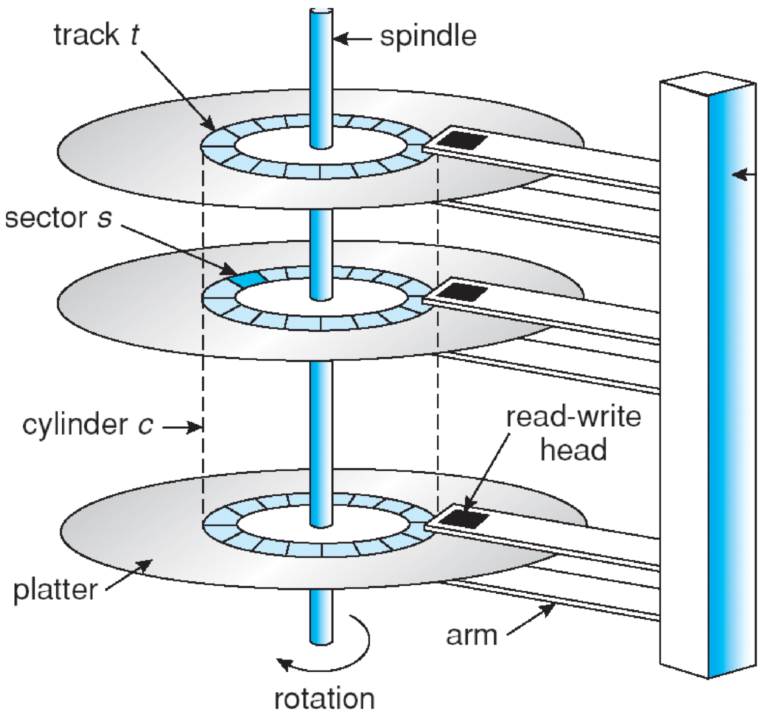
Disk geometry/structure

Platter
- Data is stored by inducing magnetic changes to it
- Each platter has 2 sides, each called a surface

Spindle
- Spins the platters around
- The rate of rotations is measured in RPM (Rotations Per Minute)

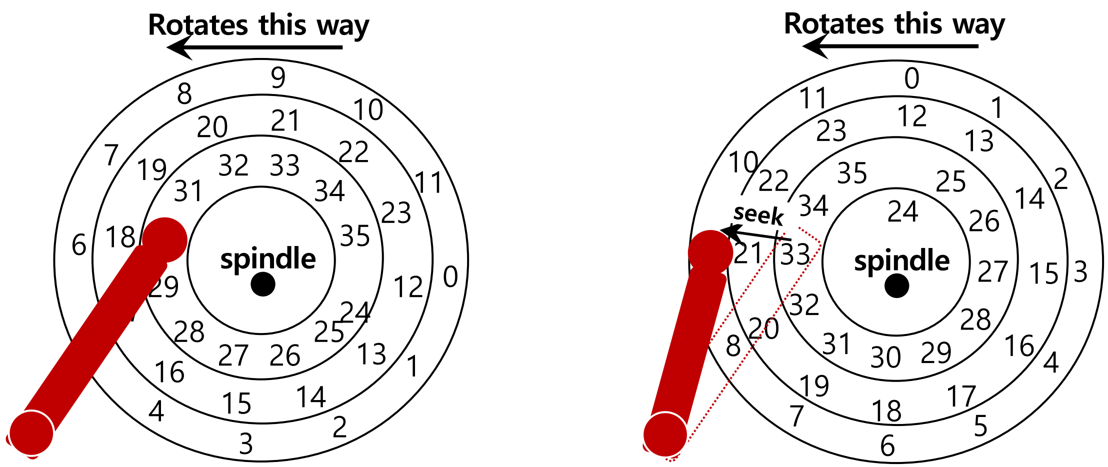
Track
- Concentric circles of sectors
- A single surface contains many thousands and thousands of tracks

Cylinder
- A stack of tracks of fixed radius
- Heads record and sense data along cylinders
- Generally only one head active at a time



Access data: seek, rotate, transfer

Seek: move the disk arm to the correct track

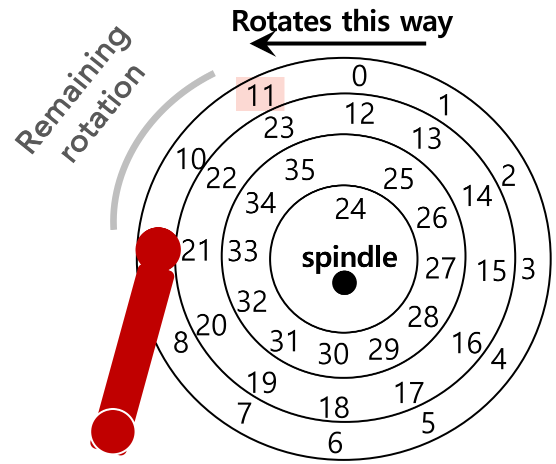


numbers represent sector no.

Seeks often take several milliseconds!

- one of the most costly disk operations
- entire seek often takes 4 - 10 ms

Access data: seek, rotate, transfer



Wait for the desired sector to rotate

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 min / 7200 rotations = 1 sec / 120 rotations = 8.3 ms / rotation

Average rotation?

- 8.3 ms / 2 = 4.15 ms

Access data: seek, rotate, transfer

The final phase of I/O

- Data is either *read from* or *written* to the surface.

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 μs

Optimizing I/O performance

General strategies

- Avoid doing I/O (disks are slow!)
- Reduce overhead (particularly seek time)
- Amortize overhead over larger requests

How to avoid doing I/O?

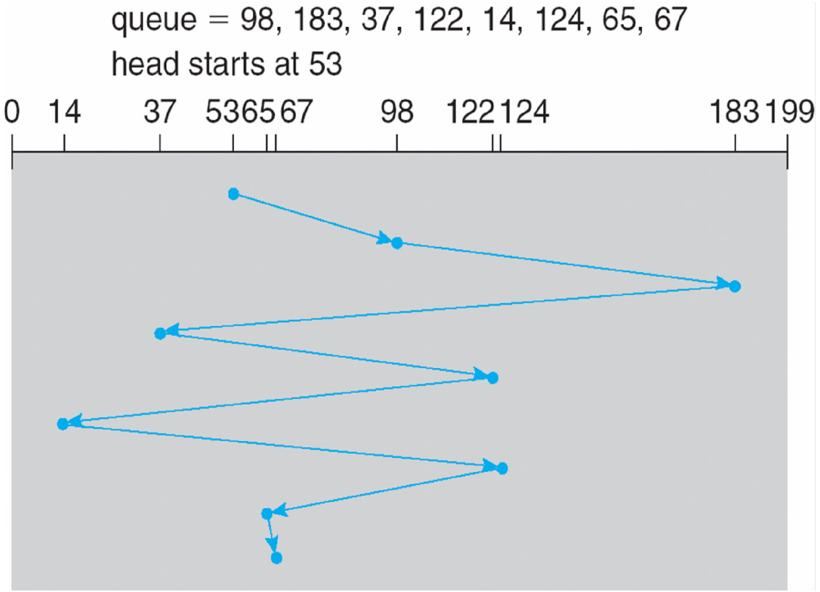
How to reduce positioning time?

Disk scheduling

Reduce overhead by reordering requests

First-come, first-served (FIFO)

- Problems?

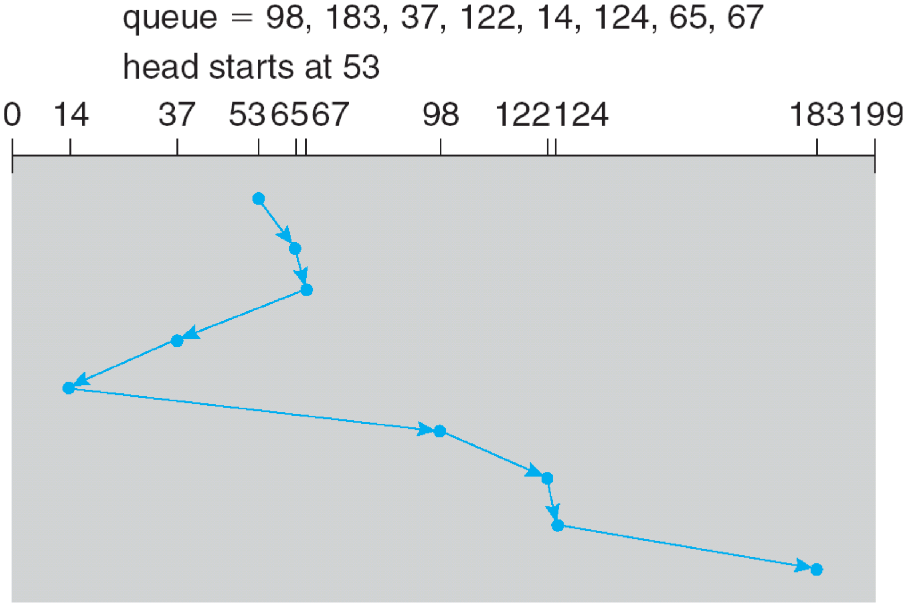


number represents the track no.

Disk scheduling (cont'd)

Shortest seek time first (SSTF)

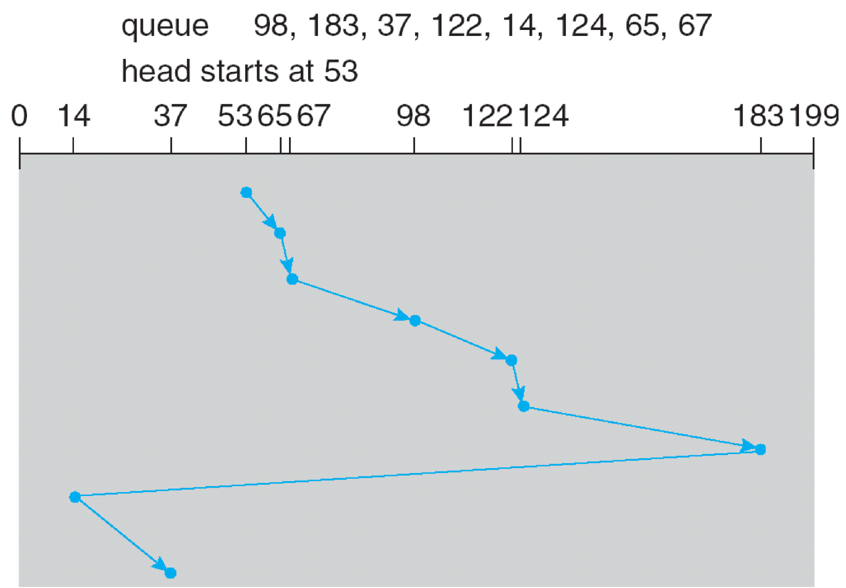
- Pick requests on the nearest track to complete first
- Reduces response time
- Problems?



Disk scheduling (cont'd)

"Elevator" Scheduling (SCAN)

- Sweep across disk, servicing all requests passed
- Like SSTF, but next seek must be in same direction
- Switch directions only if no further requests
- CSCAN: Only sweep in one direction



Amortizing overhead

Efficiency = transfer time / (overhead + transfer time)

E.g., to achieve 50% efficiency, overhead should equal transfer time

overhead = transfer time

➔ overhead = size / transfer rate

➔ size = overhead * transfer rate. In networking, this is called the bandwidth-delay product

E.g., 8 ms overhead, 100 MB/s transfer rate ➔ 800 KB transfer to achieve 50% efficiency

Solid state disks (e.g., flash memory)

Increasingly popular storage medium

Remembering data by storing charge

- Lower power consumption
- No mechanical seek times (better random read performance)
- Better shock resistance

Limited # overwrites possible

- Blocks wear out after 10,000 (MLC) – 100,000 (SLC) erases
- Requires flash translation layer (FTL) to provide wear leveling
- FTL can seriously impact performance

Limited durability

- Charge wears out over time

File abstraction

Hardware reality: a few storage objects (disks)

OS abstraction: numerous storage objects (files)

- Created and destroyed on demand
- Named and organized for user convenience

Challenges:

- How to name files?
- How to find and organize files?
- How to keep file data consistent in the presence of crashes?

File systems

File system: a persistent data structure

Persistent across what?

- Process creation/exit
- Machine crashes/reboots
- Power outages

How to enable persistence across these events?

- Use persistent storage medium
- Use persistent pointers (addresses that are stable across reboot), e.g., disk block number
- Write data in a careful order

Interface to file system

Create file

Delete file

Read <file, offset>

Write <file, offset>

Other (e.g., list files in a directory)

Other persistent data structures: database (accessed via SQL or key-value)

File system workloads

Optimize data structure for the common case

Some general rules of thumb

- Most file accesses are reads
- Most programs access files sequentially and entirely
- Most files are small, but most bytes belong to large files