香港中文大學
The Chinese University of Hong Kong

CSCI5550 Advanced File and Storage Systems

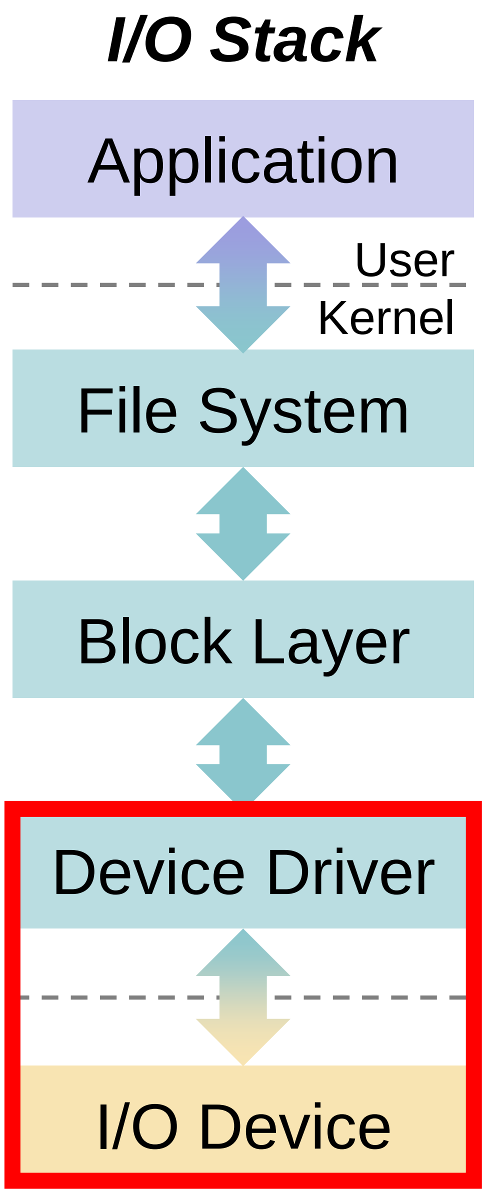
Lecture 01: I/O Devices

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

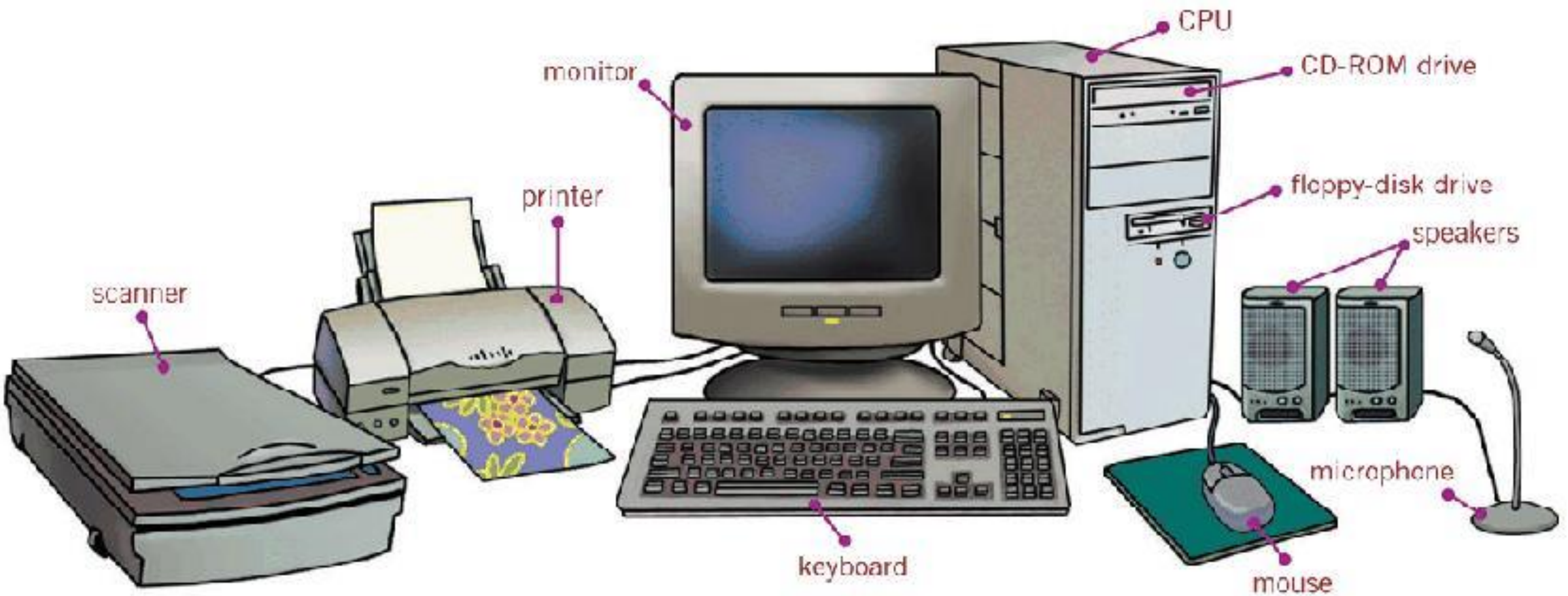
Outline

- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device
- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling



Input and Output (I/O)

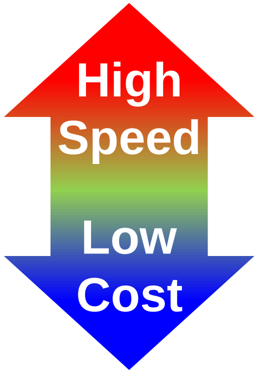
- Computers should have the ability to exchange information with a wide range of I/O devices.
  - E.g., keyboard, mouse, printer, disk drives, etc.

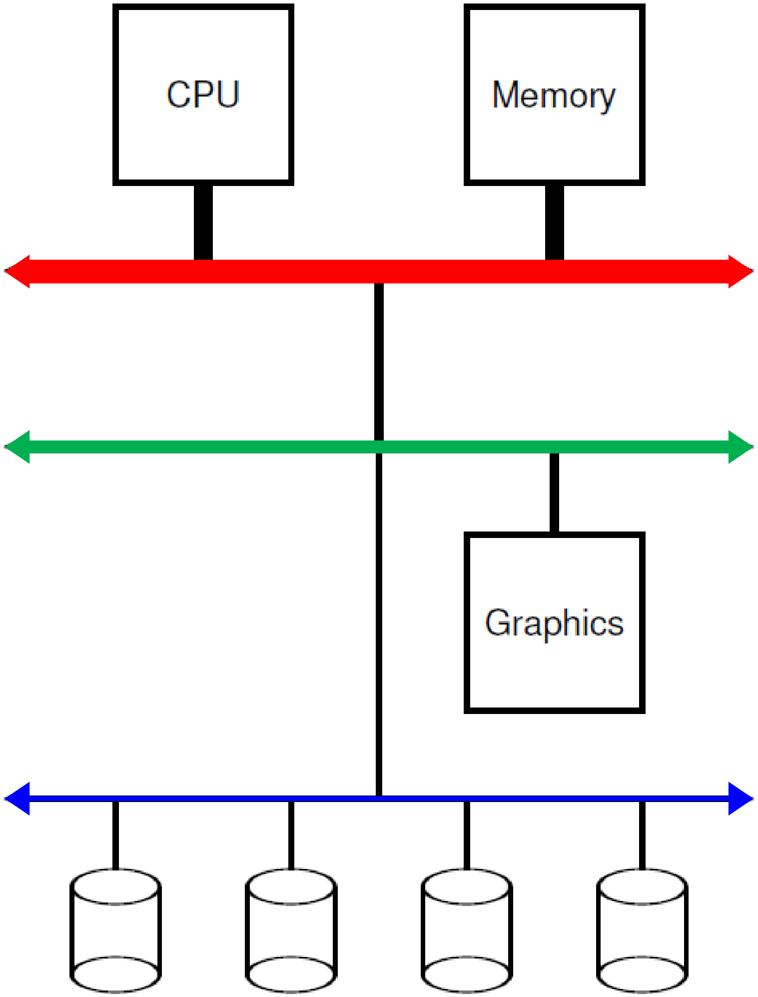


https://norizman.wordpress.com/notes/

Prototypical System Architecture

- Physics vs. Cost: **Hierarchical Bus** Structure
  - The *faster* a bus is, the *shorter* it should be!





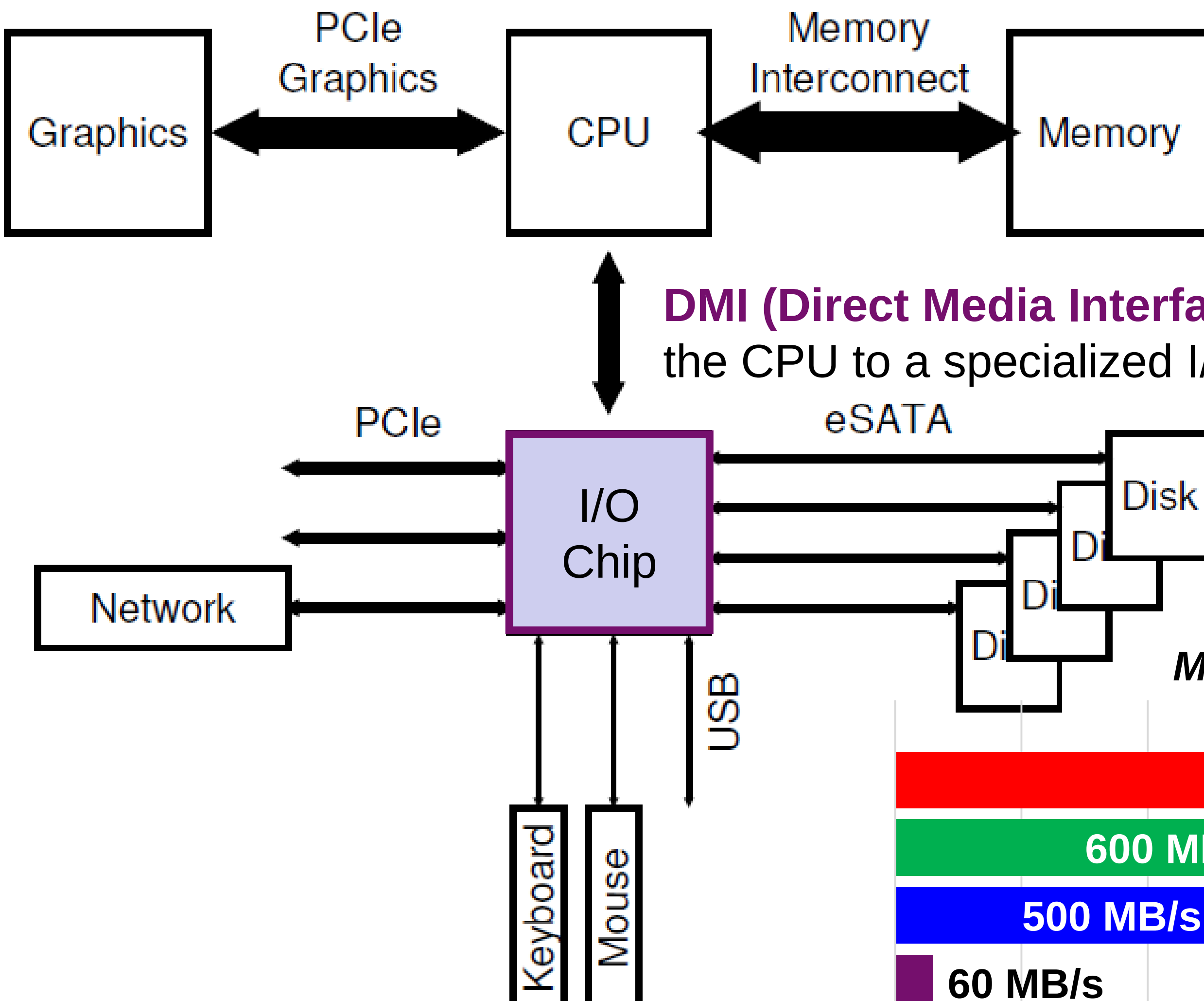
Memory Bus interconnects the fastest CPU and main memory of the system.

General I/O Bus (e.g., PCI) connects the graphics card and higher-performance I/O devices to the system.

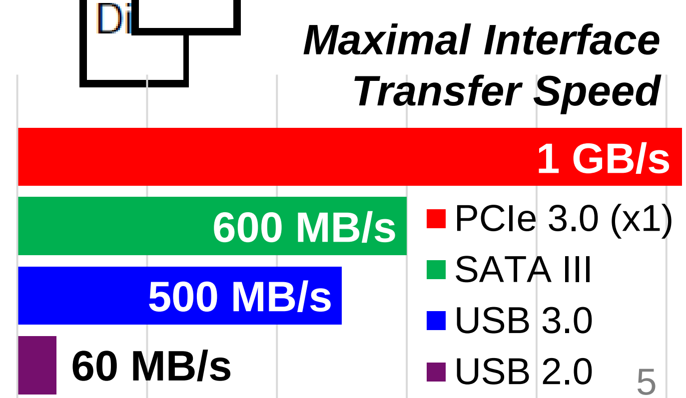
Peripheral I/O Bus (e.g., SCSI, SATA, or USB) connects the slow I/O devices, including disks, mice, and keyboards, to the system.

Modern System Architecture

- Modern systems use specialized chipsets and faster point-to-point interconnects to improve performance.



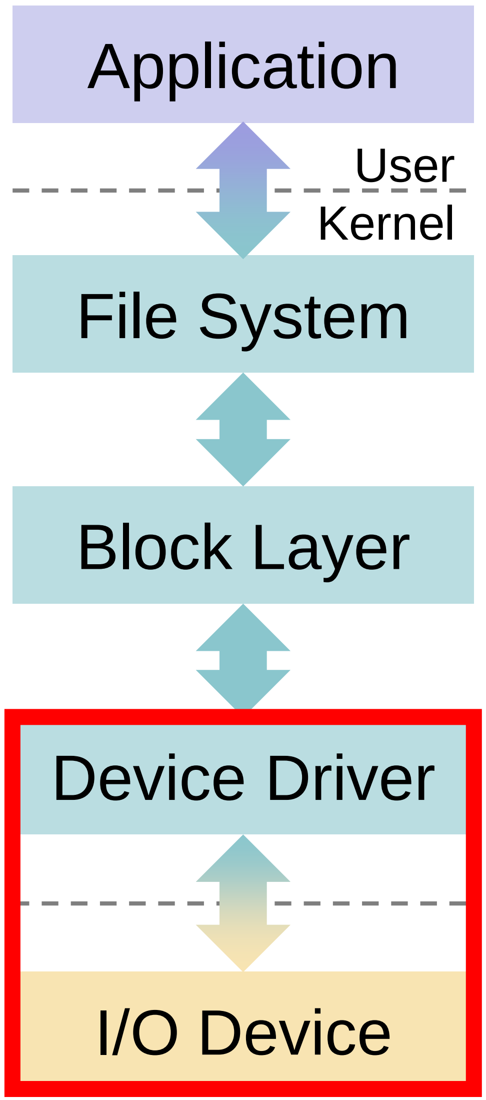
DMI (Direct Media Interface) connects the CPU to a specialized I/O chip.



Outline

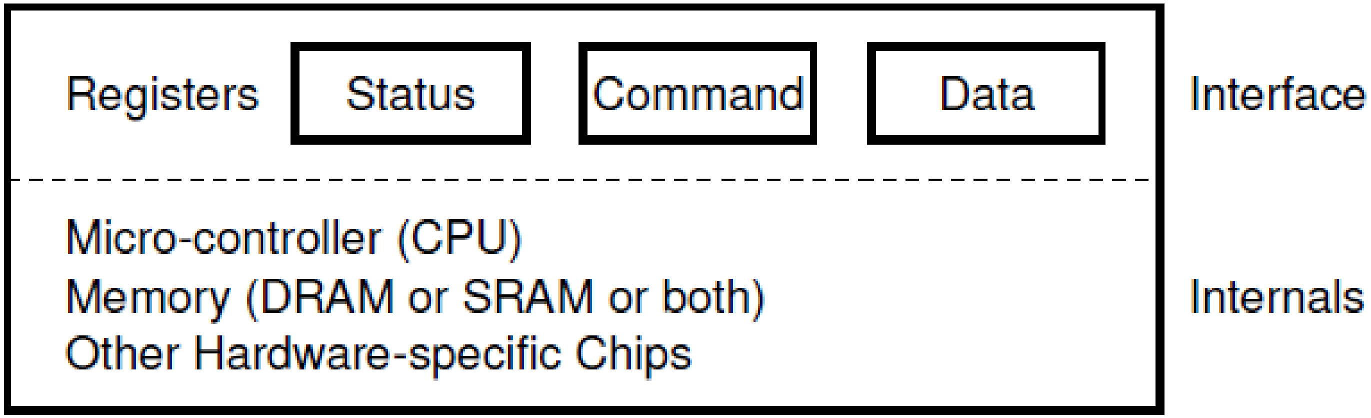
- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device
- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling

I/O Stack



Canonical Device

- A canonical (*not real*) device can be *abstracted* into two components:
  - **Interface:** allows system software to control its operation. by reading and writing the interface registers.
    - **Status**: holds current device status.
    - **Command**: tells device to perform certain tasks.
    - **Data**: pass data via device.
  - **Internal Structure:** contains firmware (software within a hardware device) to implement specific functionalities.



Canonical Protocol

- The canonical protocol is of the following four steps:
  1. The OS waits until the device is ready by **polling** the device (i.e., repeatedly reading the status register);
  2. The OS sends data down to the data register;
  3. The OS writes a command to the command register;
  4. The OS waits for the device to finish by again **polling** it.

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (Doing so starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

Class Discussion

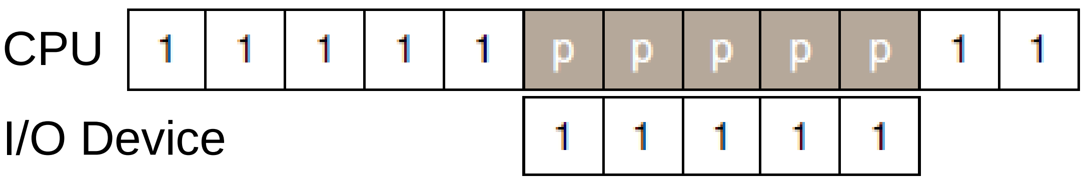
- Question: What are the potential inefficiencies in the canonical protocol?

- **Answer**: The **polling** mechanism wastes a great deal of CPU time on waiting the (usually slow) I/O device.

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (Doing so starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

Polling vs. Interrupt (1/2)

- Considering a system with two processes (indicated by `1` and `2`) and Process `1` issues an I/O request.
  - **Polling**
    - The OS simply waits on repeatedly checking the status register until the I/O is complete.

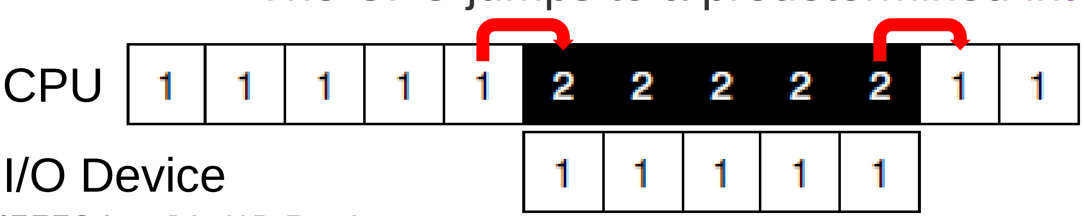


CPU needs to wait while Process `1` is polling the I/O device until it's completed.

  - **Interrupt**

    ① The OS can issue a request, put the calling process to sleep, and context switch to another task.

    ② The I/O device raises a hardware interrupt when I/O is complete.
      - The CPU jumps to a predetermined interrupt handler.



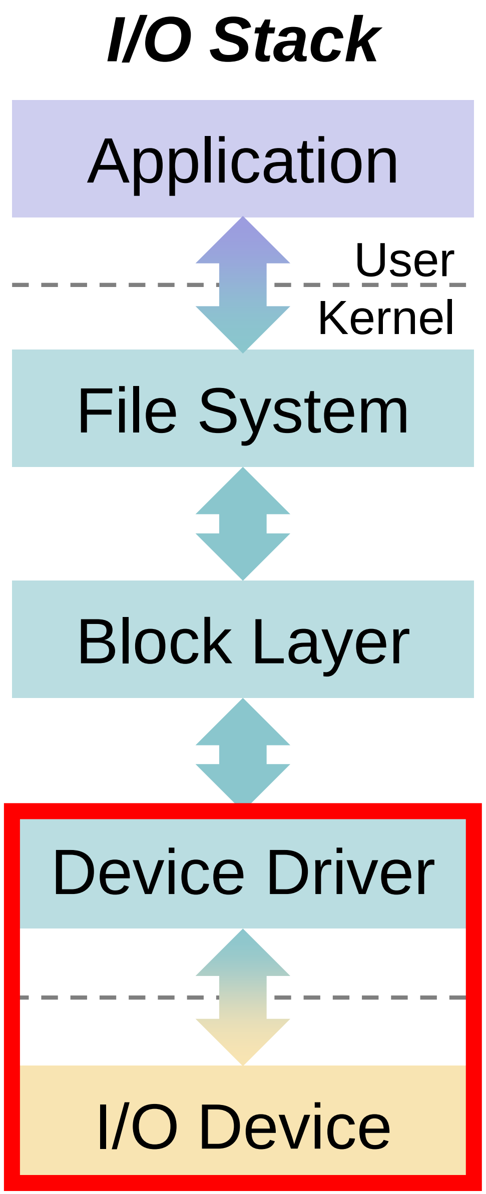
Process `2` runs on the CPU while the I/O device is servicing Process `1`.

Polling vs. Interrupt (2/2)

- Interrupt is **not always** the best.
  - Reason: Context switch is expensive (e.g., 1~10us).
    - Imagine a device that performs its tasks very quickly: the first poll usually finds the device to be done with task.
    - Using an interrupt in this case will actually slow down the system: ① switching to another process, ② handling the interrupt, and ③ switching back to the issuing process is expensive.
- Simple Rule: If the device is fast, it may be best to poll; if the device is slow, interrupts are the best.
- What if the speed of the device is *unknown* or *sometimes fast and sometimes slow*?
  - A hybrid approach may achieve the best of both worlds!
    - It may be best to use a hybrid that polls for a little while and then, if the device is not yet finished, uses interrupts.

Outline

- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device
- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling



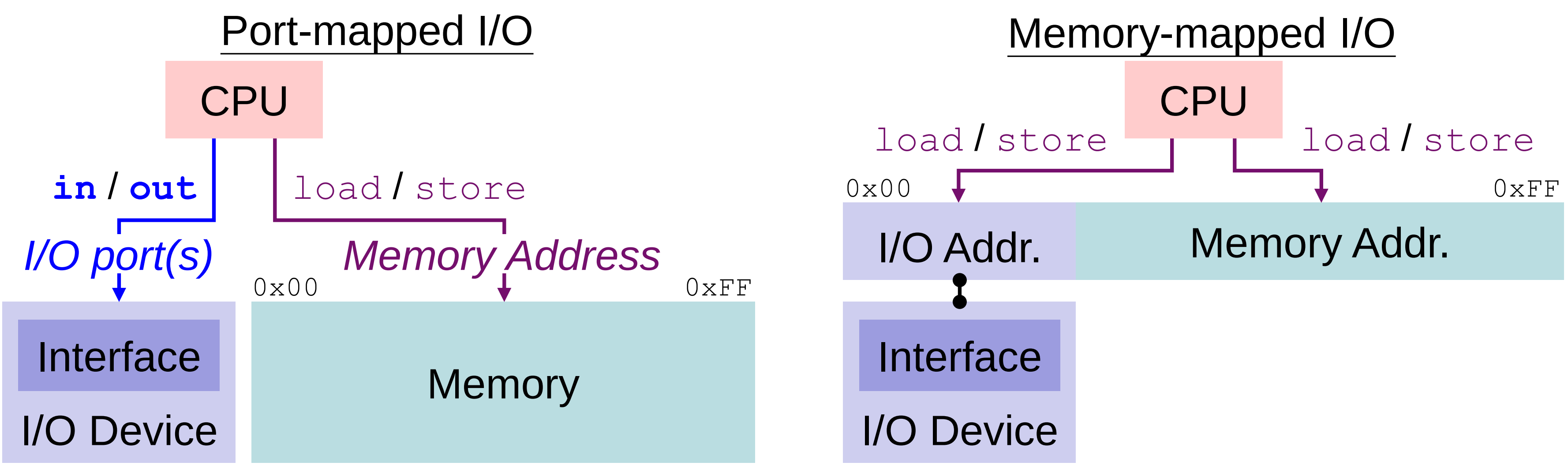
Device Interaction Methods (1/3)

- The canonical protocol is of the following four steps:
  ① The OS waits until the device is ready by polling the device (i.e., repeatedly reading the status register);
  ② The OS sends data down to the data register;
  ③ The OS writes a command to the command register;
  ④ The OS waits for the device to finish by again polling it.

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (Doing so starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

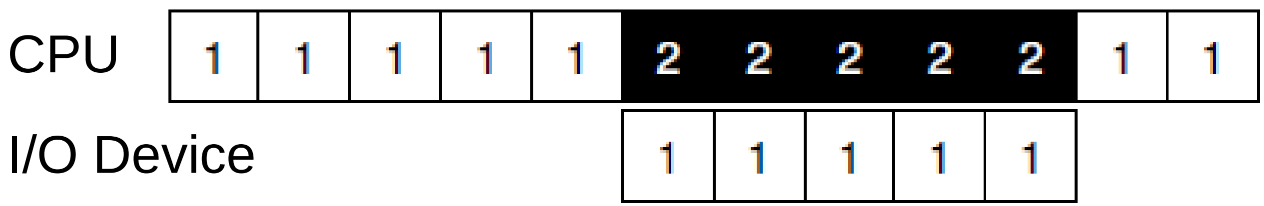
Device Interaction Methods (2/3)

- **Method 1) Programmed I/O (PIO)**: The CPU is involved with the data transfer.
  - **Port-mapped I/O**: Data are explicitly transferred, through specific *port* (which names the device), to *device register*.
    - Specialized I/O instructions (such as `in` and `out` on x86) are used.
  - **Memory-mapped I/O**: The data transfer is through the *shared* memory (i.e., device registers as memory locations).
    - General memory instructions (such as `load` and `store`) are used.

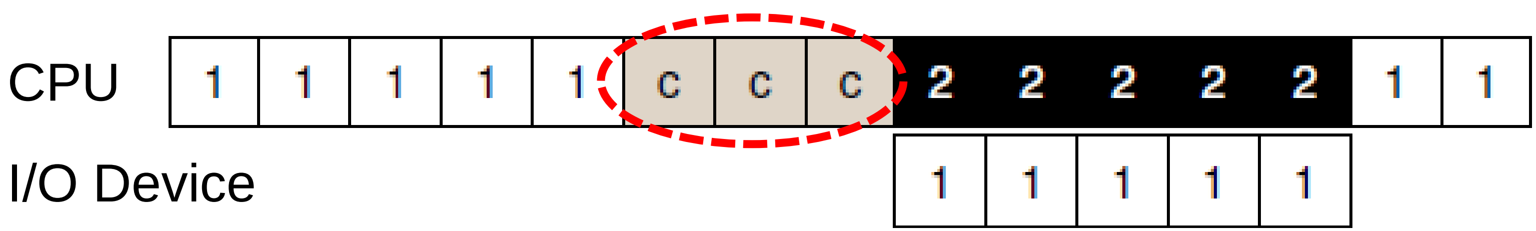


Class Discussion

- Question: What is the potential inefficiency of Programmed I/O even though the interrupt is used?

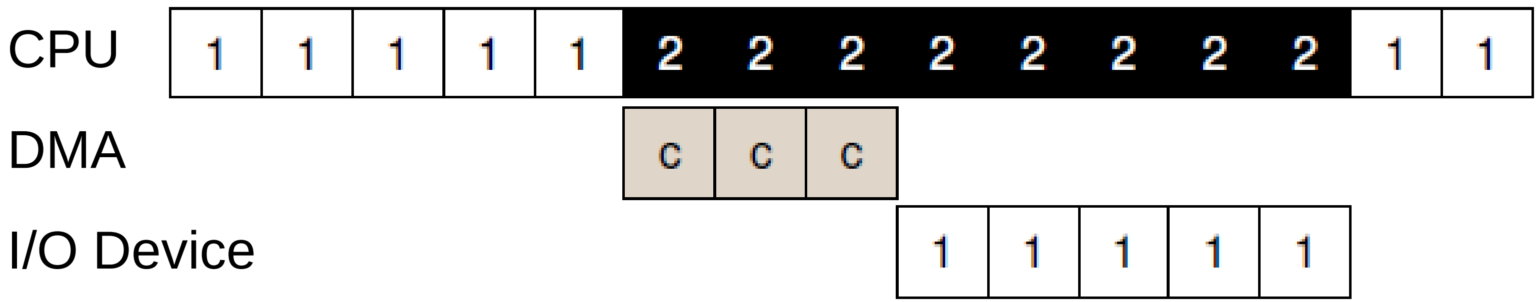


- **Answer**: The CPU could be still overburdened when transferring a large chunk of data to a device.



Device Interaction Methods (3/3)

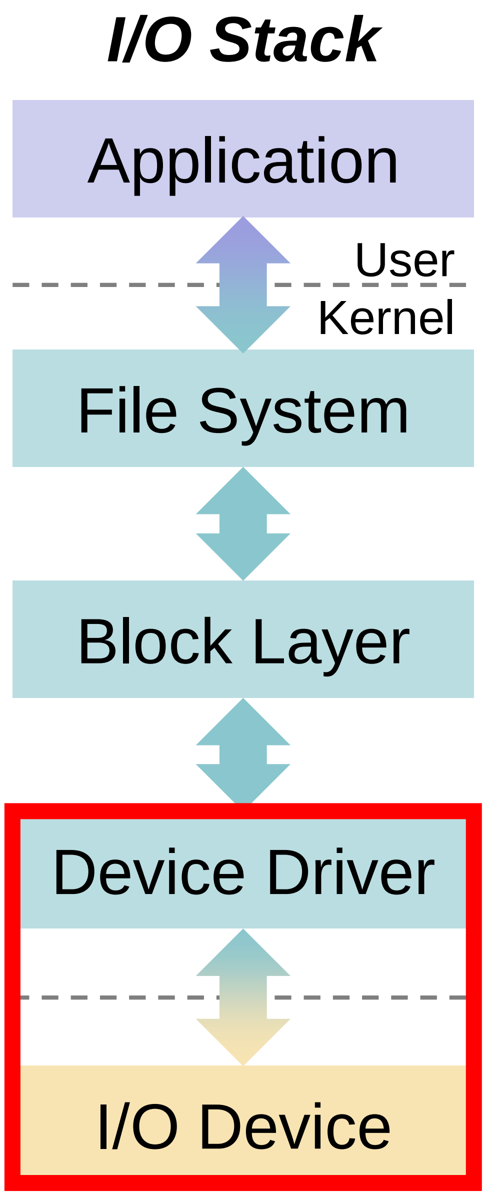
- **Method 2) Direct Memory Access (DMA)**: The data transfer is conducted without much CPU intervention.
  - The OS offloads the data transfer (from the device to the memory) to the **DMA hardware** for freeing the CPU.
  - When complete, the DMA raises an interrupt to notify OS.



- Both methods are still in use today.
  - Method 1) Programmed I/O (with CPU involved)
    - Port-mapped I/O (e.g., ARM) & Memory-mapped I/O (e.g., Intel)
  - Method 2) Direct Memory Access (without CPU involved)

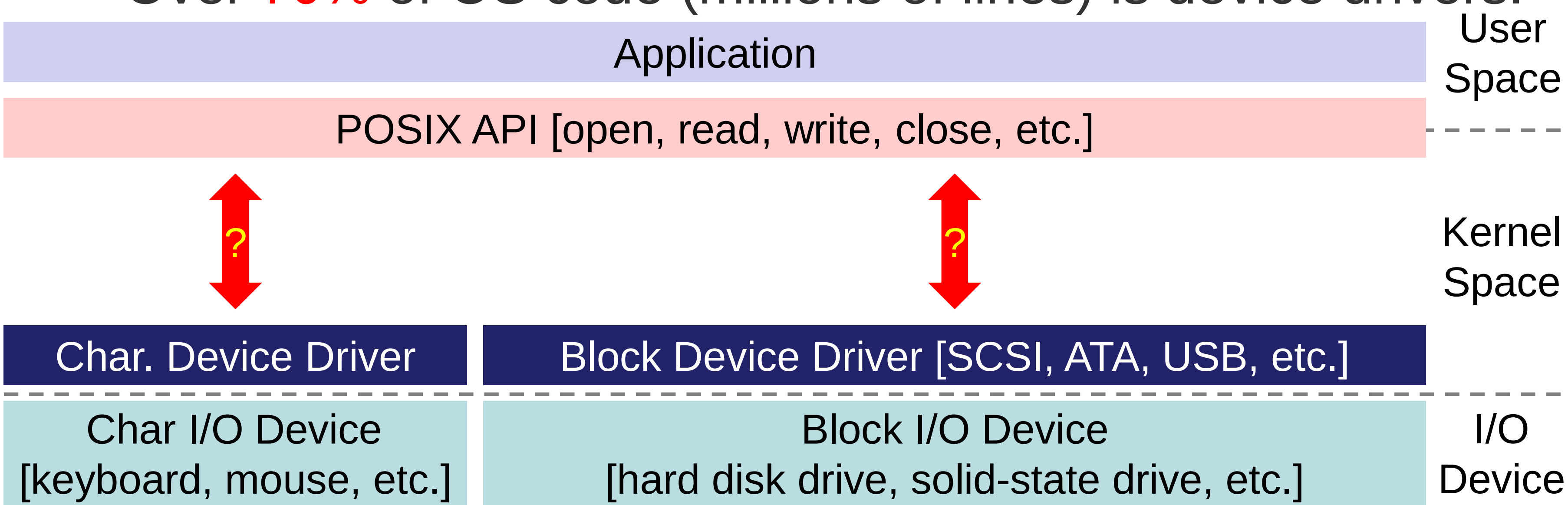
Outline

- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device
- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling



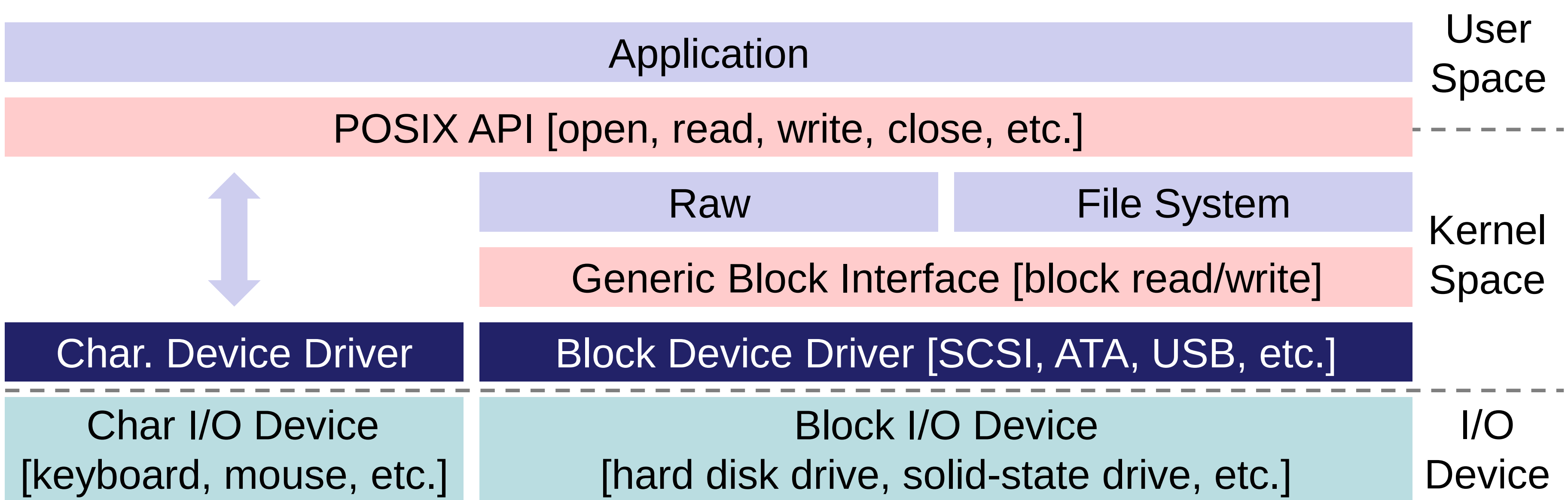
Device Driver (1/2)

- In reality, each device may have its specific interface and internal structures.
- **Device Driver**: encapsulates device-specific details.
  - Applications issue I/O requests to device via POSIX API.
  - The specific device driver handles the I/O request.
    - Manufacturers implement the device driver for each device.
  - Over 70% of OS code (millions of lines) is device drivers.



Device Driver (2/2)

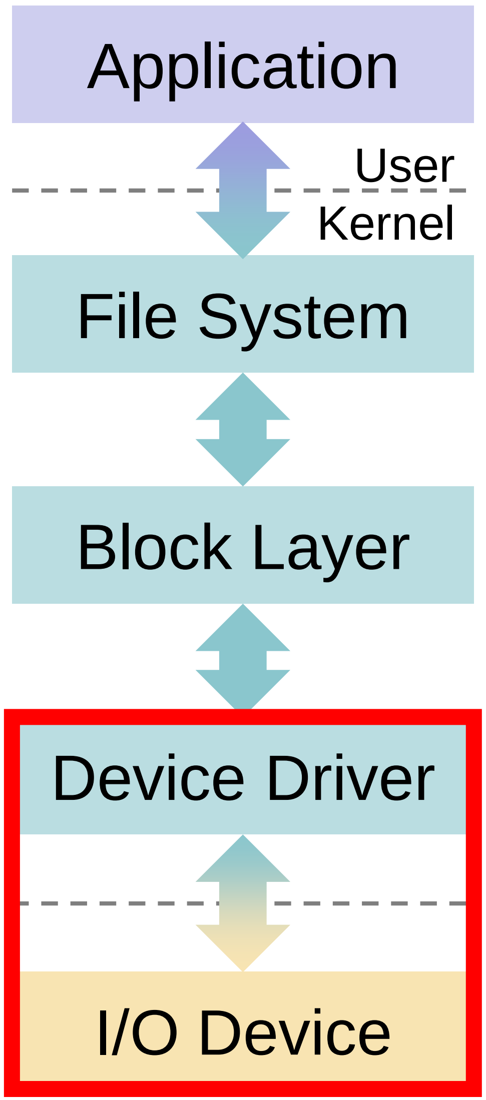
- **Character I/O Device:** transfers I/O in bytes serially.
- **Block-oriented I/O Device**: transfers I/O in blocks.
  - Some block devices (e.g., HDD or SSD), which can persist the stored data, are also referred to as data storage.
    - A block device can be directly read/written via the raw interface.
    - A file system can further provide *file abstraction* to manage the data.
    - A block interface (and bio structure) unifies the block accesses.



Outline

- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device
- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling

I/O Stack



Why Hard Disk Drive (HDD)?

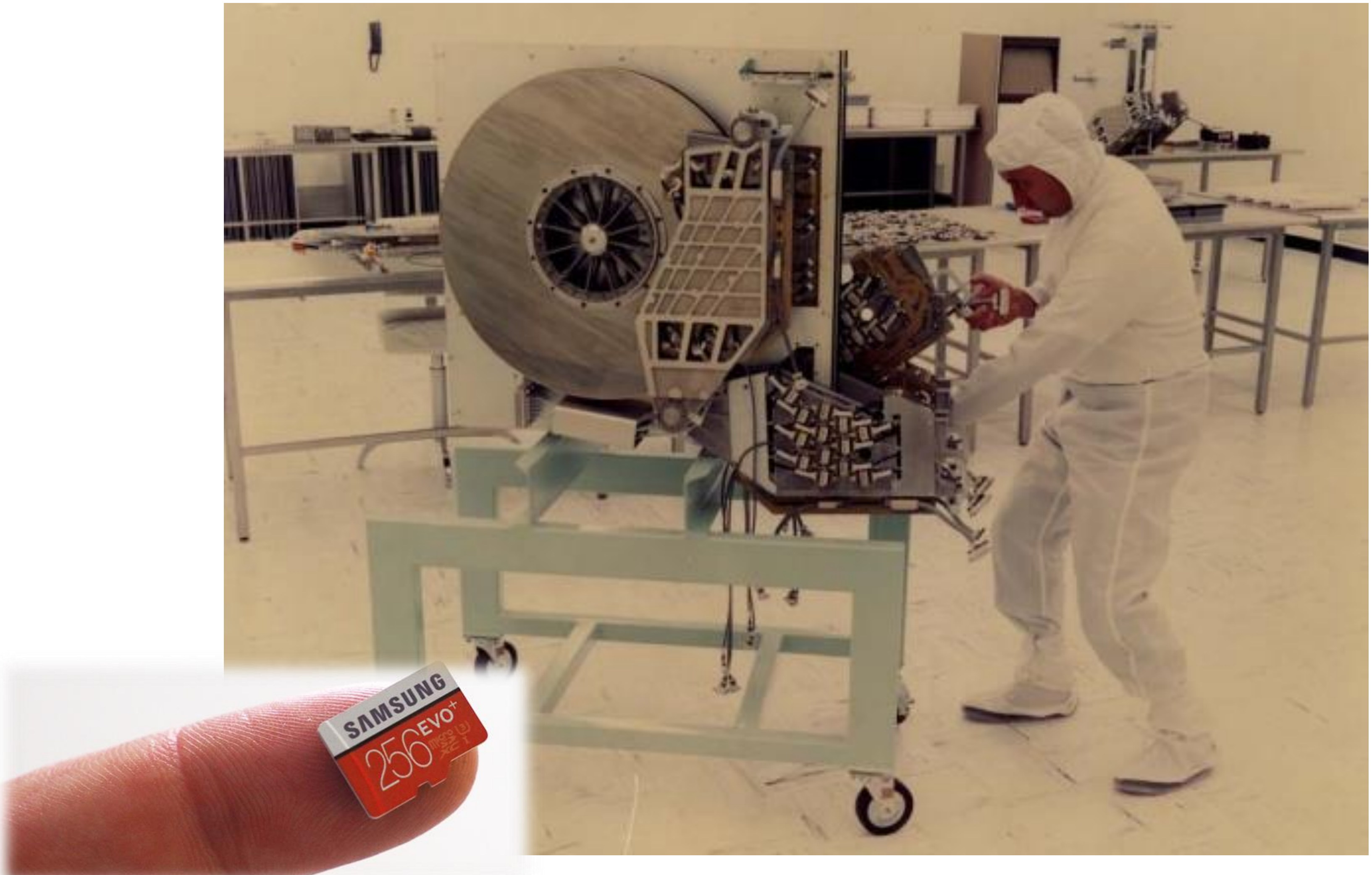
- Let’s focus on one specific I/O device: **hard disk drive (HDD)**.

- HDDs have been the main form of persistent data storage in computer systems for decades.
  - In 1953, IBM recognized the urgent need.
  - The first commercial usage of HDD began in 1957.

- Many file and storage systems are designed and optimized based on HDD characteristics.

Amazing Photos about HDD

- Below is a 250 MB hard disk drive in 1979 …

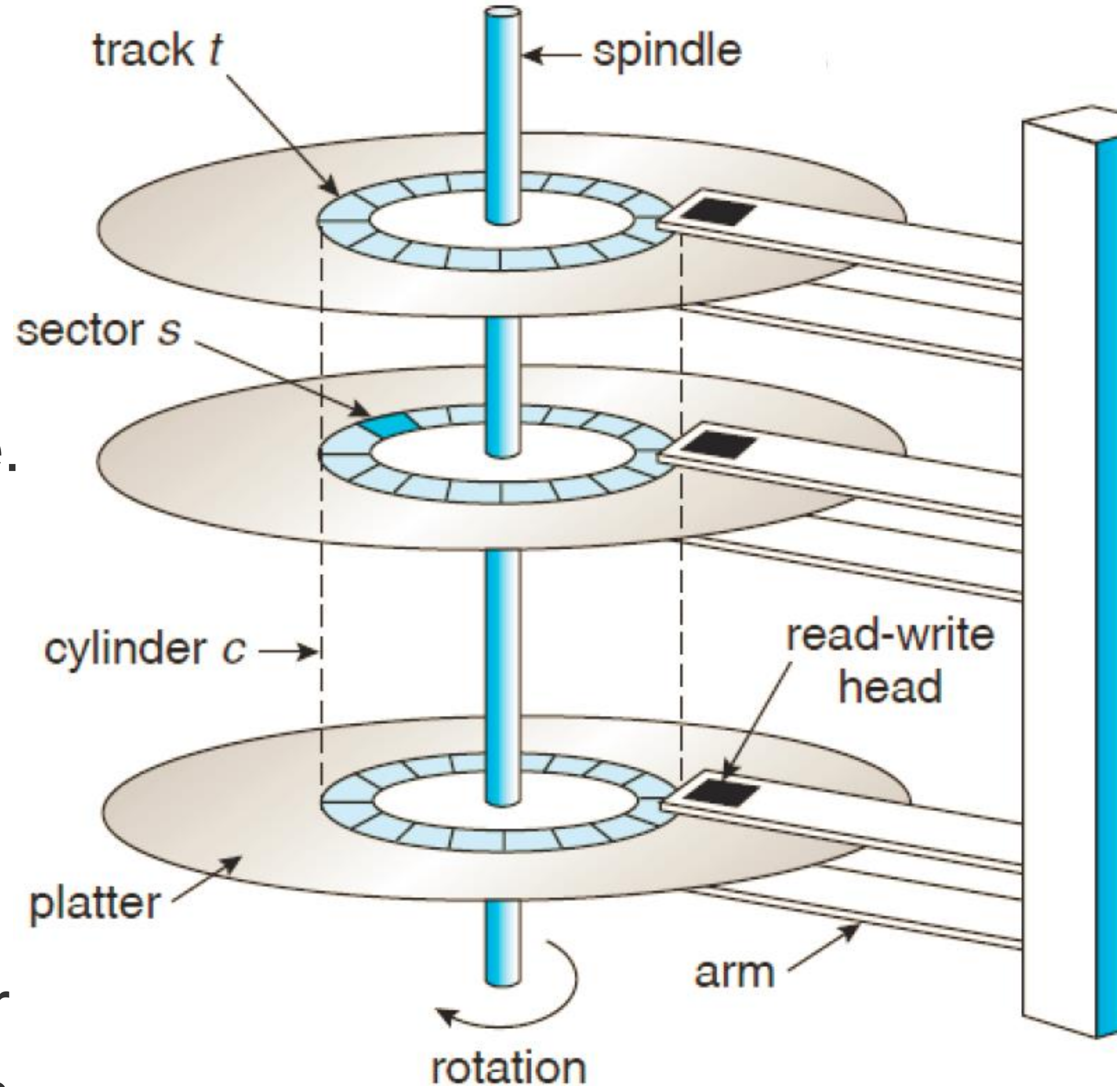


Disk Organization: Logical View

- HDD: Accessed in **blocks** but organized in **sectors**
  - **Sector**
    - The most common sector size is 512 bytes.
      - The sector size is fixed on an HDD.
    - All sectors are numbered from 0 to $n - 1$ (i.e., the **address space**).
      - The disk can be logically viewed as **an array of n sectors**.
  - **Block**
    - Disk I/Os are in units of **blocks**.
    - A **block** may refer to one or multiple sectors.
- In an HDD, only a single 512-byte write is **atomic**.
  - It will either <u>complete in entirety</u> or <u>fail at all</u>.
    - Torn Write: Only a portion of a larger write may complete.

Disk Organization: Physical View

- A hard disk has one or multiple **platters**.
  - Each platter has 2 sides (**surfaces**).
  - Platters are bound together by a **spindle**.
- Each surface has multiple *concentric circles* called **tracks**.
  - A track is further divided into **sectors**.
- A **disk head** reads or writes data of sectors.

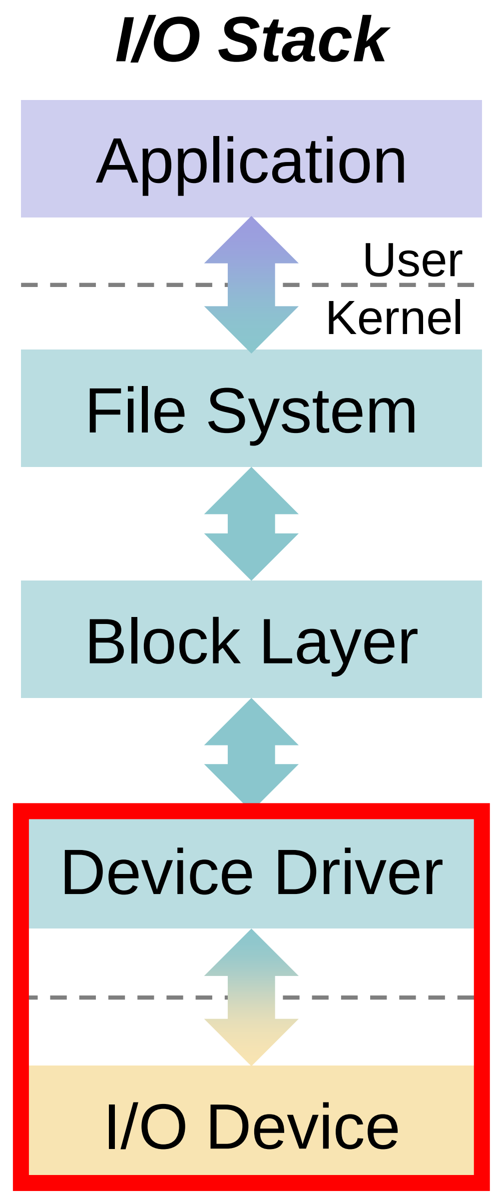


Silberschatz et al., "Operating System Concepts Essential".

Outline

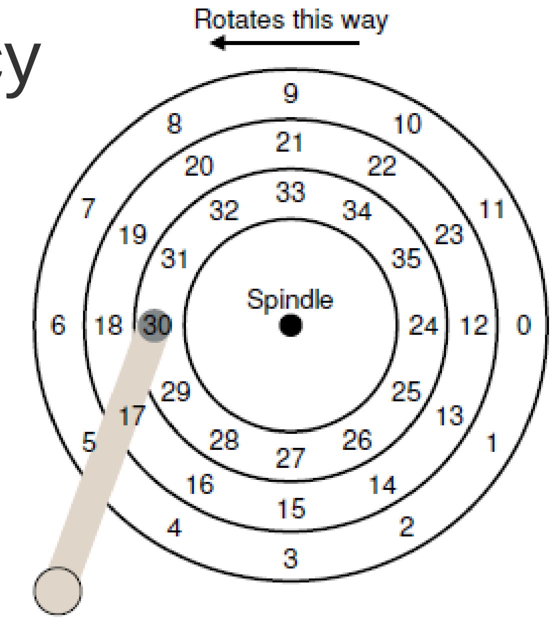
- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device

- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling

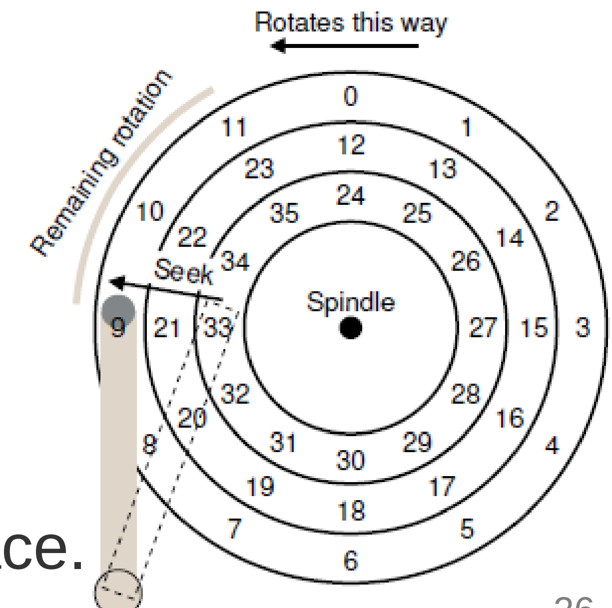


Disk I/O Performance (1/2)

- **Rotational Delay**: Single-track latency
  - The time to rotate a disk to move the desired sector under the disk head.
    - Full rotational delay: R
      - E.g., 10,000 **RPM** disk → R = 4ms.
      - **Rotations per Minute:** The rate of rotation.
    - Average rotational delay: R/2



- **Seek Time**: Multiple-track latency
  - The time to move the disk arm to the correct track.
  - Average seek time: The order of ***ms***.



- **Transfer Time**
  - The time to read/write data to the surface.

Disk I/O Performance (2/2)

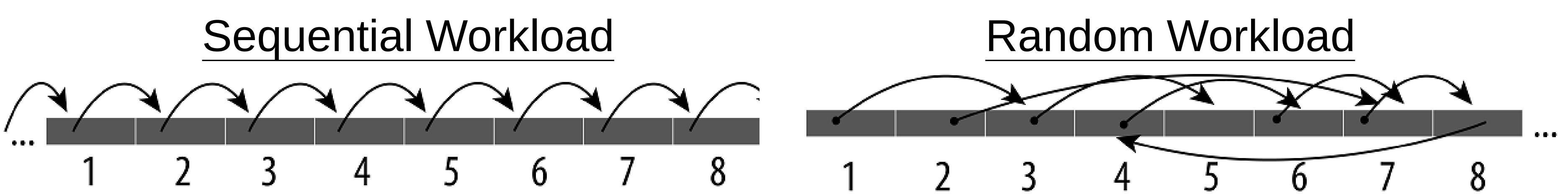
- **I/O Time**: The sum of three major components.

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

- **I/O Rate**: Divide the size of data transfer by the time.

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$$

- I/O performance heavily depends on **workloads**.
  - **Sequential Workload**: Large reads/writes (e.g., 100MB) to a number of contiguous sectors.
  - **Random Workload**: Small reads/writes (e.g., 4KB) to random sector locations on the disk.



Disk I/O Performance: An Example

- Let's consider two modern disks from Seagate.

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Left: "high performance" disk vs. Right: "large capacity" disk

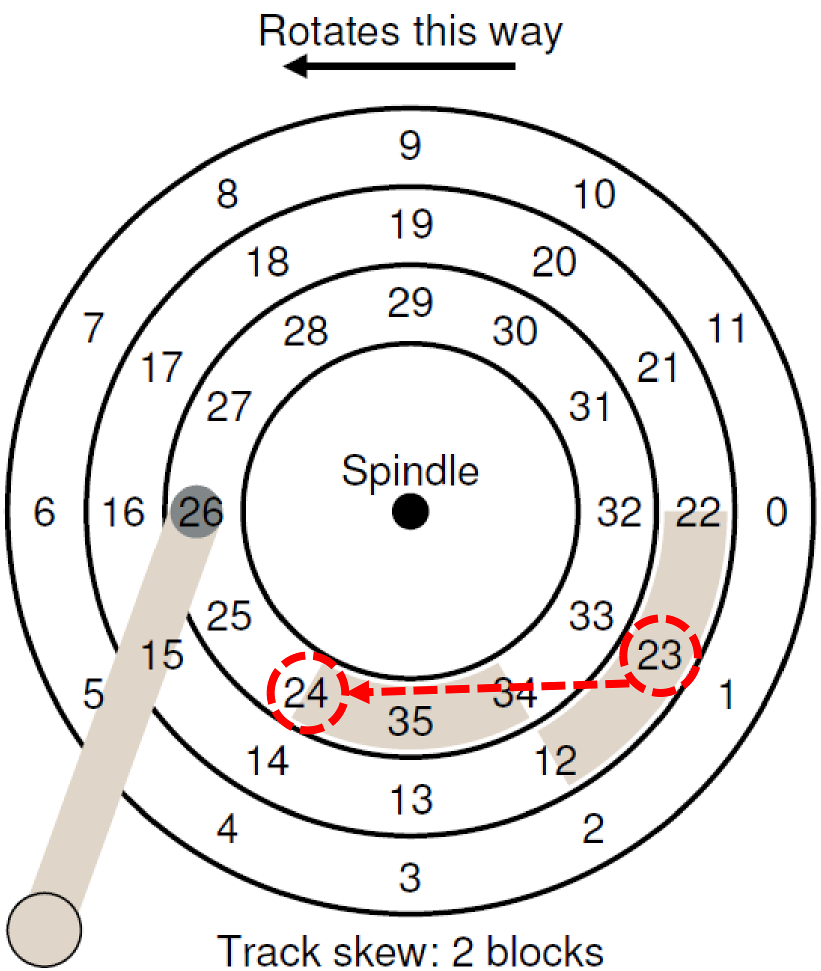
- Performance under random/sequential workloads:

	Cheetah	Barracuda
$R_{I/O}$ Random	0.66 MB/s	0.31 MB/s
$R_{I/O}$ Sequential	125 MB/s	105 MB/s

  - **Sequential I/O**: Determined by transfer performance.
  - **Random I/O**: Determined by rotation and seek time.

More Details: Track Skew

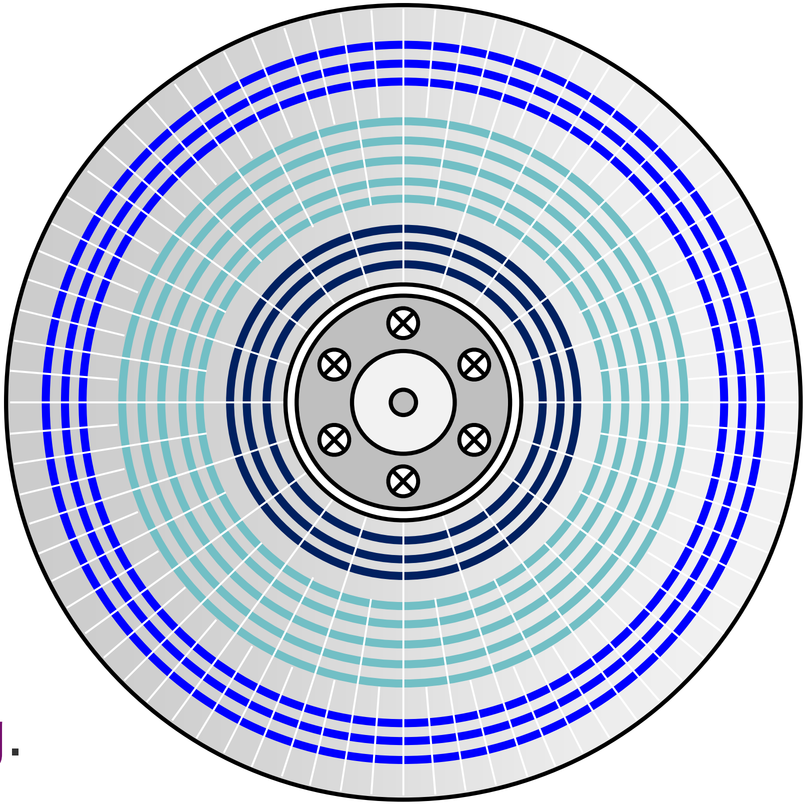
- **Track Skew**: ensures that sequential reads can be properly serviced when crossing track boundaries.
  - The disk needs time to re-position the head.



More Details: Zones

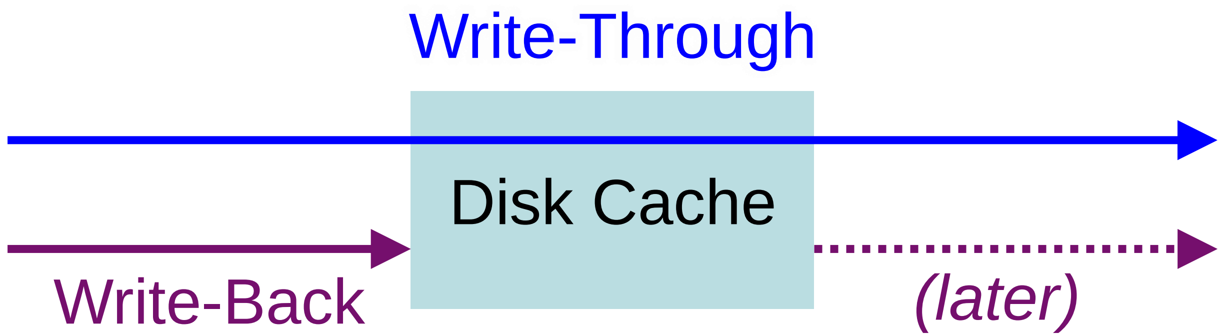
- Another reality is that outer tracks tend to have more sectors than inner tracks.
  - It is nothing special but a result of geometry: there is simply more room out there.

- **Multi-zoned HDDs** are organized into multiple zones.
  - A zone is consecutive set of tracks on a surface.
  - In a zone, every track has *the same number of sectors*.
  - Outer zones have more sectors than inner zones.
  - Also known as zone bit recoding.



More Details: Disk Caching

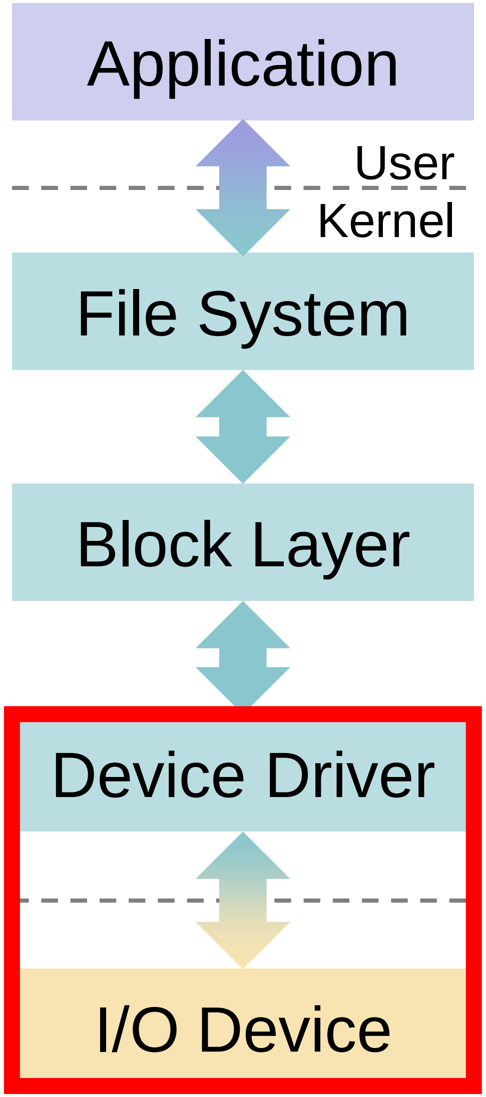
- **Disk Cache** (or **Track Buffer**): A small amount of in-disk memory (usually around 8 or 16 MB) to hold data read from or written to disk.
- On **reads**, the drive may cache all sectors of a track to quickly respond to subsequent reads to same track.
- On **writes**, the drive has a choice:
  - **Write-Through**: Write is completed when data is on disk.
  - **Write-Back**: Write is completed when data is in cache.



Outline

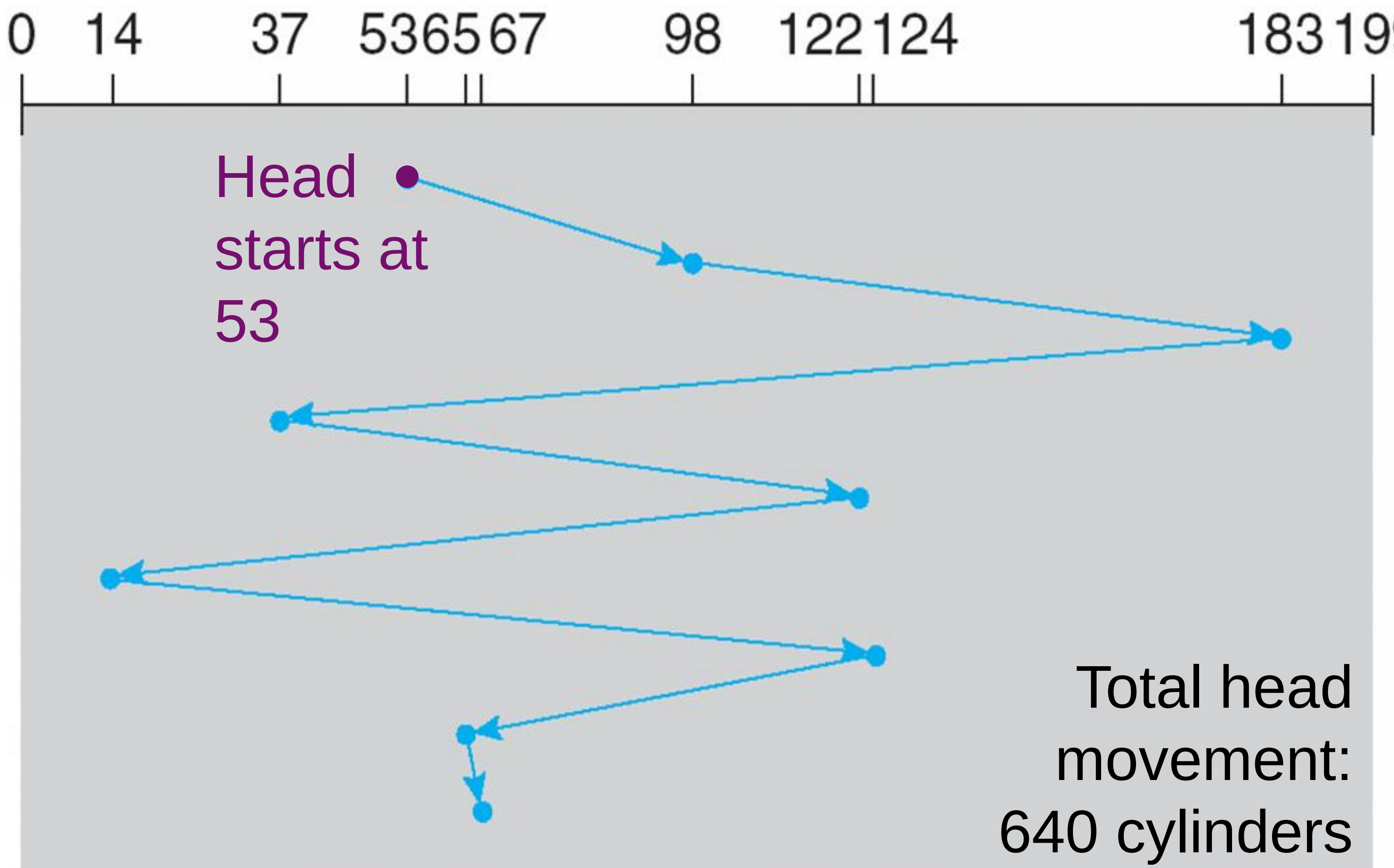
- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device
- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling

I/O Stack



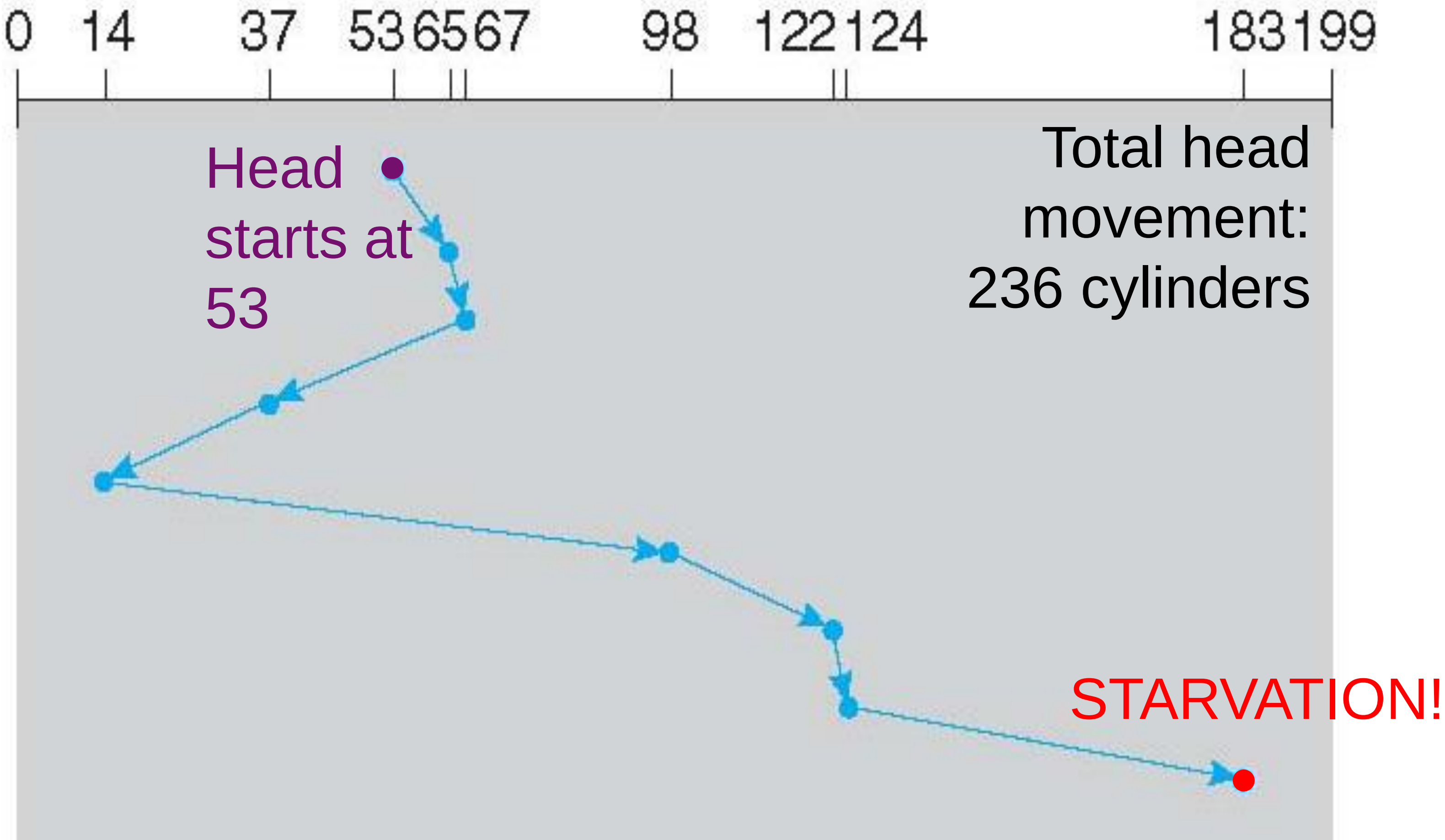
Disk Scheduling: Decides the order of I/Os

- Without disk scheduling, the requests are served in a **First Come First Serve (FCFS)** basis.
- Given accesses: 98, 183, 37, 122, 14, 124, 65, 67



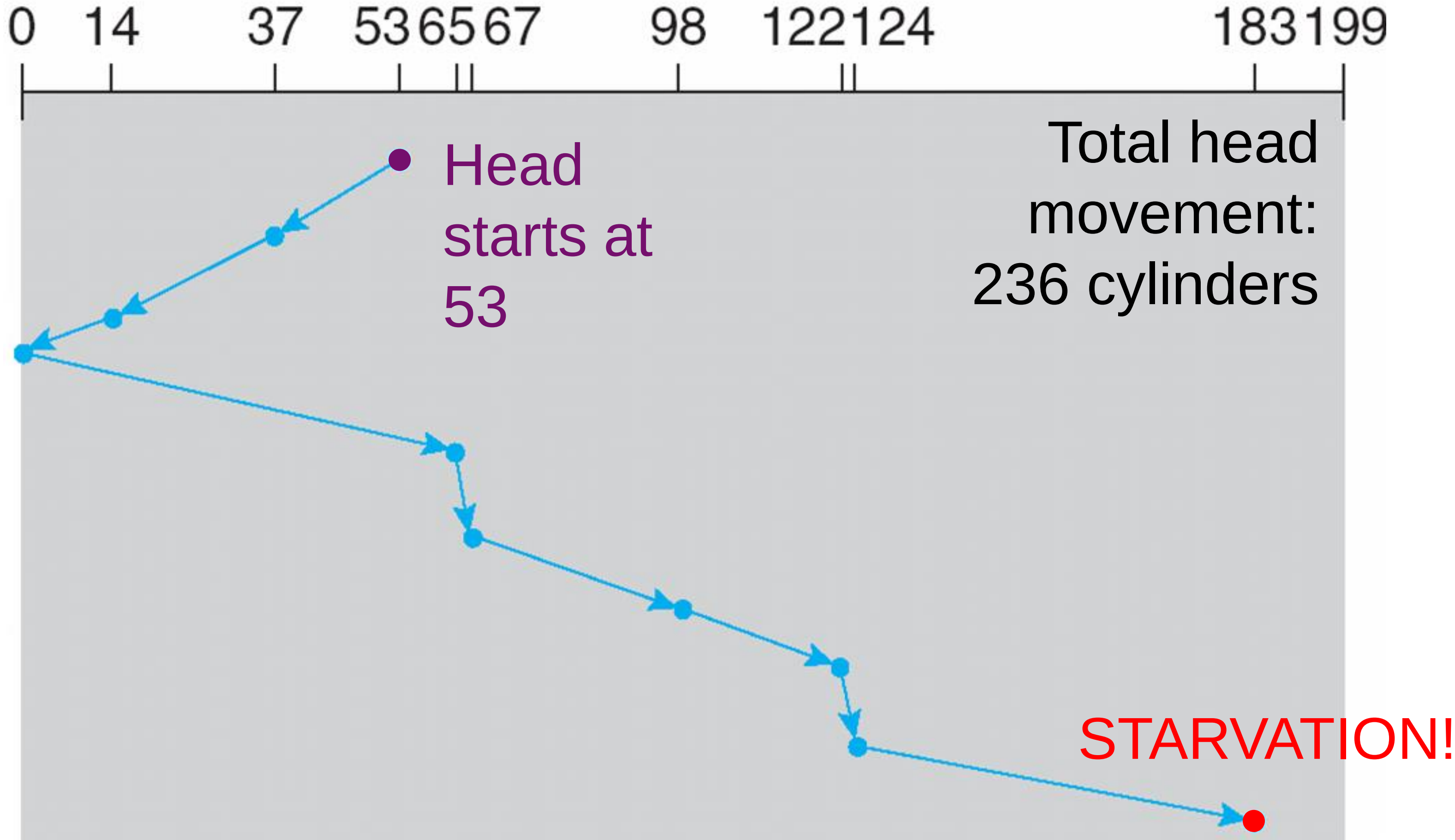
Shortest Seek Time First (SSTF)

- **SSTF** selects the request with the minimum seek time from the current head position.
- Given accesses: 98, 183, 37, 122, 14, 124, 65, 67



SCAN (a.k.a. Elevator)

- **SCAN** starts at one end of the disk, moves toward the other end, reverses until reaching any end.
- Given accesses: 98, 183, 37, 122, 14, 124, 65, 67



Variants of SCAN

- **SCAN** (a.k.a. **Elevator**)
  - Moves the head across tracks of the disk from one end to another end, and services requests in order.
  - Reverses the direction of the head at the other end.
- **F-SCAN**
  - Freezes the queue of requests during a sweep.
  - Avoids starvation of far-away requests, by delaying the servicing of late-arriving (but nearer by) request.
- **C-SCAN**
  - When the head reaches the end, immediately returns to the beginning and services the requests from beginning to end (i.e., always serves in one direction).
  - Avoids favoring the middle locations.

Class Discussion

- Question: Why SSTF, SCAN and its variants cannot deliver the best scheduling results?

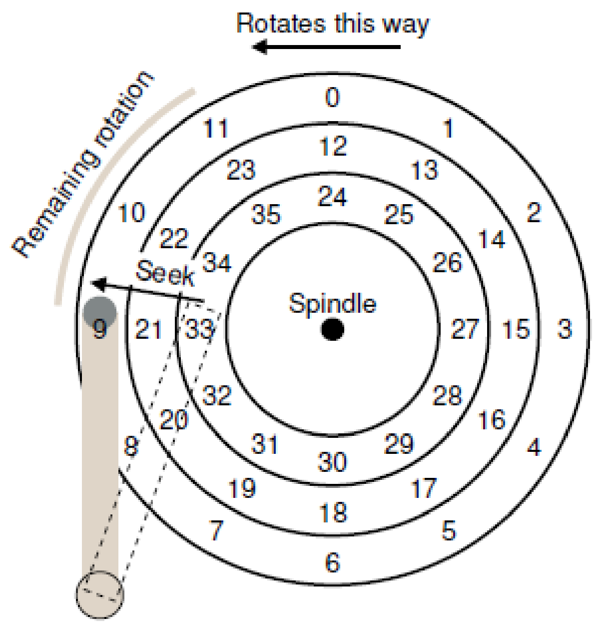
- **Answer**: They don't consider the rotation delay.

I/O Time: The sum of three major components.

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

Shortest Positioning Time First (SPTF)

- **Shortest Position Time First (SPTF)**
  - Estimates the sum of the seek time and rotation delay from the disk head to the read location.

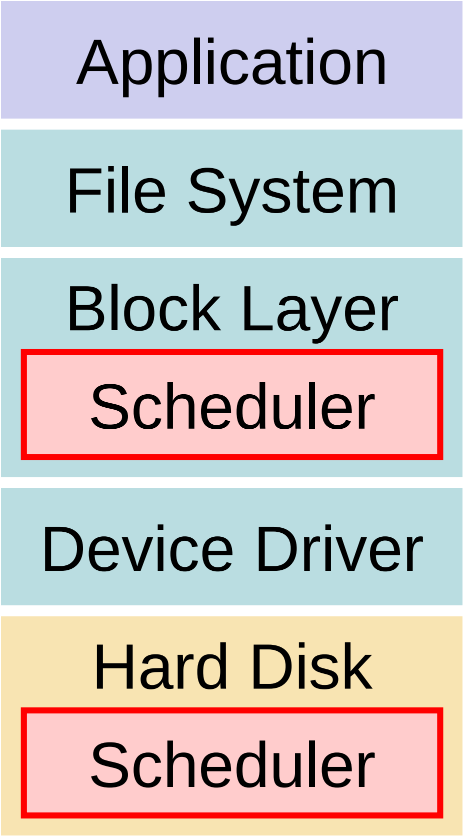


  - Limitation: Usually performed within a drive, but not in OS.
    - Why? The OS generally does not have a good idea where track boundaries are or where the disk head currently is.

Other Disk Scheduling Issues

Q1. **Where** is disk scheduling performed?

- At both OS (in block layer) and disk.
  - The OS scheduler picks a few best requests;
  - The disk scheduler selects the best possible order (e.g., SPTF).



Q2. Can we **merge I/O** to further improve performance?

- Consider a series of requests: 33, 8, 34.
- The scheduler should merge requests for blocks 33 and 34 into a single request.

Q3. **How long** should the OS wait before issuing an I/O?

- By waiting, a new and "better" request may arrive at the disk, and thus overall efficiency is increased.
- It is tricky to decide "when" and "how long" to wait.

Summary

- Basics of I/O Devices
  - System Architecture
  - Canonical Device and Canonical Protocol
    - Polling vs. Interrupt
  - Device Interaction Methods
    - Programmed I/O vs. Direct Memory Access
  - Device Driver
    - Char Device vs. Block Device

- Case Study of Block I/O Device: HDD
  - Disk Organization
  - Disk I/O Performance
  - Disk Scheduling

I/O Stack