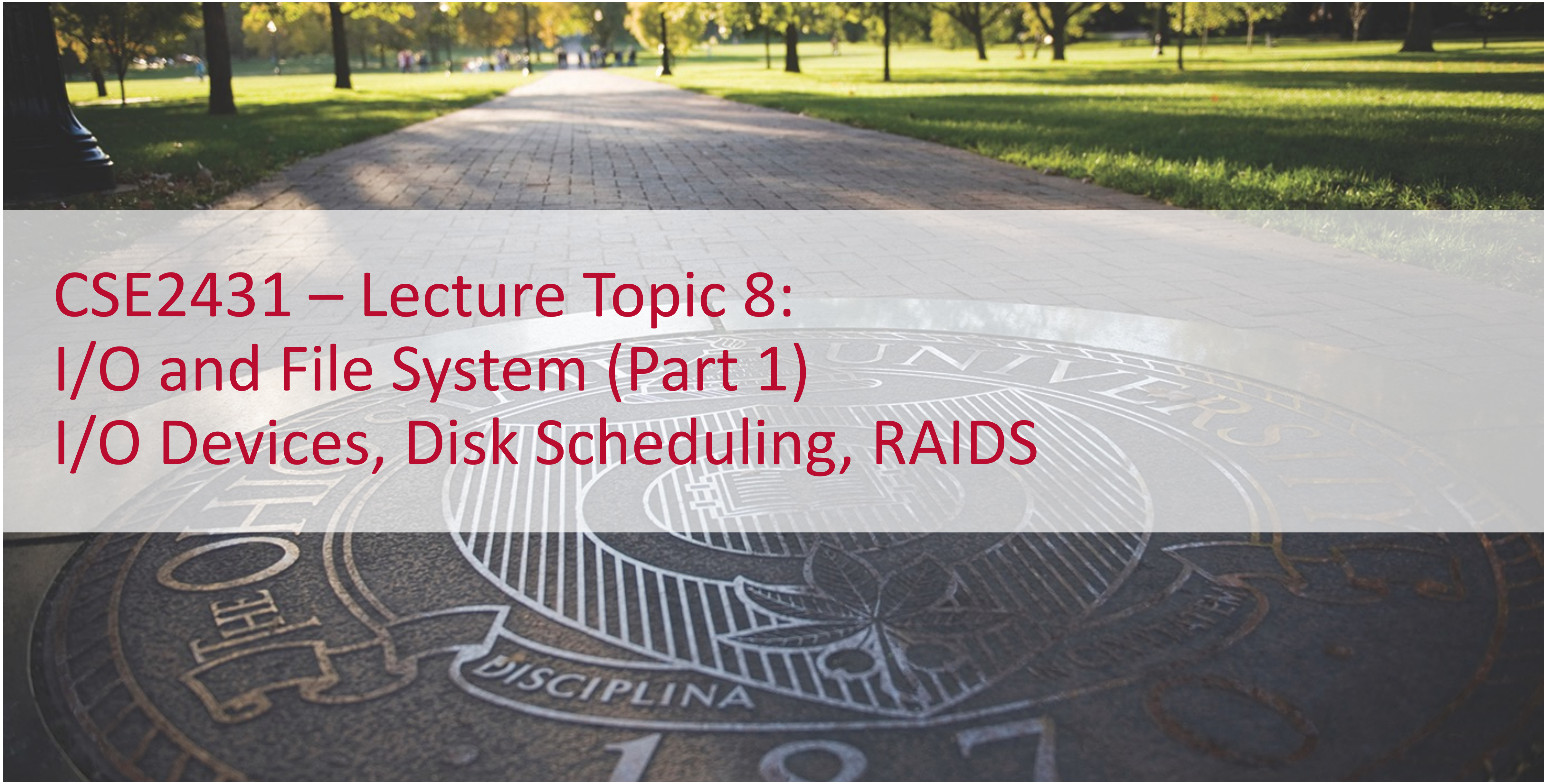


CSE2431 – Lecture Topic 8:
I/O and File System (Part 1)
I/O Devices, Disk Scheduling, RAIDS

THE OHIO STATE UNIVERSITY
COLLEGE OF ENGINEERING

I/O and File System (Part 1)

Instructor: Luan Duong, Ph.D.

CSE 2431: Introduction to Operating Systems

Reading: **Chap. 36, 37, and 38 [OSTEP]**

Outline: I/O and File System (part 1)

- I/O Systems
  - **What are I/O Devices?**
  - **Disks (again):**
    - **Disk Performance**
    - **Disk Scheduling**
  - I/O System
    - I/O System Architecture
    - I/O Devices Operating

What is I/O device?

- Apart from CPU and memory, almost everything else is an I/O Device.

- Examples: disk, graphic card, screen, mouse, keyboard, network card, printer, joystick, sound card, …

- How does an OS operate so many kinds of I/O devices?
  - To answer this question, let's first see how I/O devices work.

Disk Performance: Speed

- Disks are mechanical, so they're slow compared to CPUs
  - Seek: several msec (usually)
  - Rotation: typical/common speeds include 5400 RPM (**rotations per minute**), 7200 RPM, and 15000 RPM.
    - Average rotation time (seconds) = 1/ (RPM/60) / 2 (several msec)
  - Data transfer: usually 100 MB/sec – 150 MB/sec

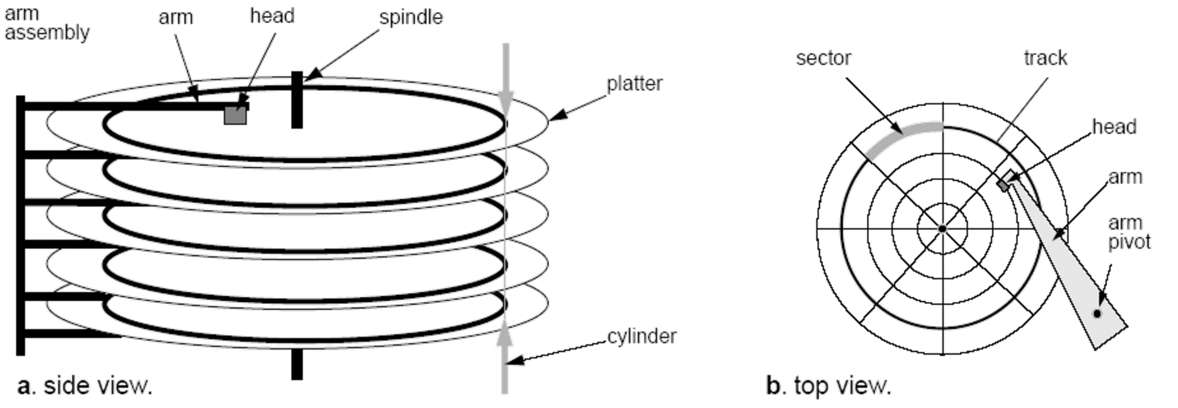


Figure 1: the mechanical components of a disk drive.

Disk Performance: Access Time (1)

Average access time for a sector:

$$T_{access} = T_{avg\_seek} + T_{avg\_rotation} + T_{avg\_transfer}$$

Seek time (T_{avg_seek}):

- Time to position heads over cylinder
- Typical T_{avg_seek} is 3–9 msec, 20-msec maximum

Rotational latency ($T_{avg\_rotation}$):

- After head is positioned over track, the time it takes for the first bit of the sector to pass under the head
- Worst case: head just misses the sector and waits for the disk to rotate 360

$$T_{max\_rotation} = (1/RPM) \times (60 \text{ secs}/1 \text{ min})$$

- Average case is **half** of worst case:

$$T_{avg\_rotation} = (1/2) \times (1/RPM) \times (60 \text{ secs}/1 \text{ min})$$

- Typical $T_{avg\_rotation}$ = 5400–7200 RPM

Access Time (2)

Transfer time ($T_{avg\_transfer}$):

- Time to read bits in the sector
- Time depends on rotational speed, number of sectors per track.
- Estimate of the average transfer time:
  - $T_{avg\_transfer}$ = (1/RPM) x (1/(avg #sectors/track)) × (60 secs/1 min) × (1000 msec/1 sec)

Example:

- Rotational rate = 7200 RPM
- Average seek time = 9 msec
- Avg #sectors/track = 400

$T_{avg\_rotation}$ = 1/2 × (60 secs/7200 RPM) × (1000 msec/sec) = **4 msec**

$T_{avg\_transfer}$ = (60/7200 RPM) × (1/400 secs/track) × (1000 msec/sec) = **0.02 msec**

T_{access} = 9 msec + 4 msec + 0.02 msec = **13.02 msec**

Exercise

- Consider the Cheetah 15K.5 disk with these parameters:
  - 15,000 RPM
  - 4-msec average seek time
  - Max disk transfer: 125 MB/sec
- How long does it take to transfer 4 KB of data (assuming data, sectors are aligned)?
- How long does it take to transfer 4 MB of data?

Exercise: Solved

- Seek time = 4 msec
- Rotation time = 1/(15000/60)/2 = 2 msec

  1. For 4 KB of data: transfer time = 4 KB / (125 MB/sec) =0.031 msec
     - Transfer time is almost negligible compared to seek and rotation time
     - Disk access time = 4 msec + 2 msec + 0.031 msec = 6.031 msec
  2. For 4 MB data:, transfer time = 4 MB / (125 MB/sec) = 31 msec
     - Disk access time = 4 msec + 2 msec + 31 msec = 37 msec
     - Compared to case (1), we transfer 1,000× the amount of data in only 6× the amount of time!

- **Disks prefer sequential, large I/O operations**

Disks Prefer Sequential, Large I/O operations

- Sequential, large I/O operations **can be 100× faster** than small, random ones
  - OS has mechanisms to assemble small, sequential I/O operations into a large I/O one, so these two are somewhat equivalent.
  - The disk always spins!
    - Hence, if you issue an I/O operation to sector 1, wait awhile, and then issue another I/O operation to sector 2, they're *not* considered sequential I/O operations.
    - The disk and OS determine the acceptable interval between these.
- Reason: seek and rotation times dominate.
  - New devices like solid-state drives (SSDs) and non-volatile memory (NVM) have alleviated this problem
  - Yet, even on these devices, sequential I/O is much faster than random I/O

Unaligned I/O

- Thus far, we've assumed that I/O is aligned with sectors
- What do we do to read or write half a sector or 1.5 sectors?
  - Recall: a disk's minimal I/O data unit is a sector
- Read is simpler: just read all corresponding sectors that contain our targets, and then copy target data out
- Write: we must **read all sectors first**, **modify them**, then **write back**
  - This is called a **read-modify-write** procedure
  - It further slows down a disk, because a write needs an extra read

Lesson to Learn

- Avoid random I/O operations
- Avoid unaligned I/Os, in particular unaligned writes!
- These optimizations **significantly improve** the performance of your programs

Disk Performance Factor: Seeking

- Seeking: position the head to the desired cylinder (2–5 msec)
- Seek speed depends on:
  - Available power for pivot motor (0.5× seek time needs 4× power)
  - Arm stiffness (30–40g acceleration required for short seek time; flexible arms can twist ⟹ crash head into platter surface!)
- A seek is composed of
  - A speedup, a coast, a slowdown, a settle
  - For very short seeks, the settle time dominates (1–3 msec)
- Real-life analogy?

Questions

- Since seek time and rotation delay dominate, how do you improve I/O performance?

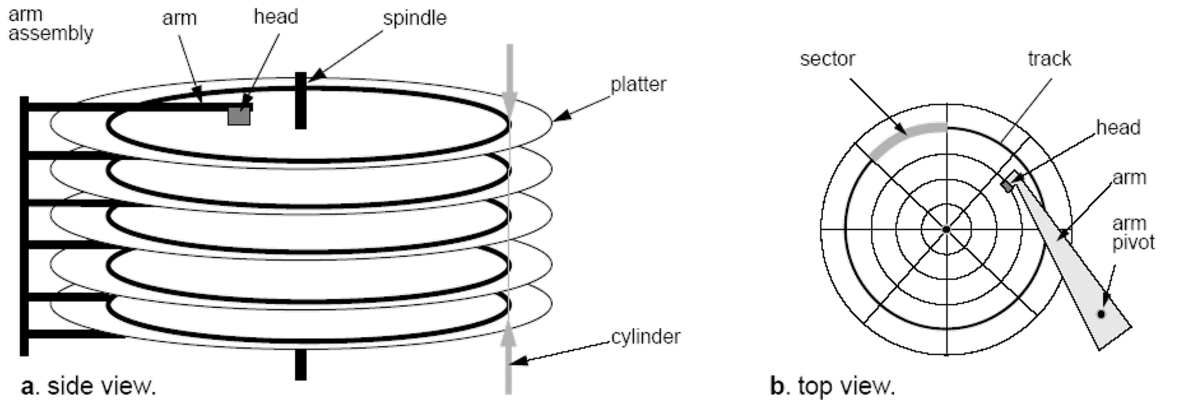


Figure 1: the mechanical components of a disk drive.

Outline: I/O and File System (part 1)

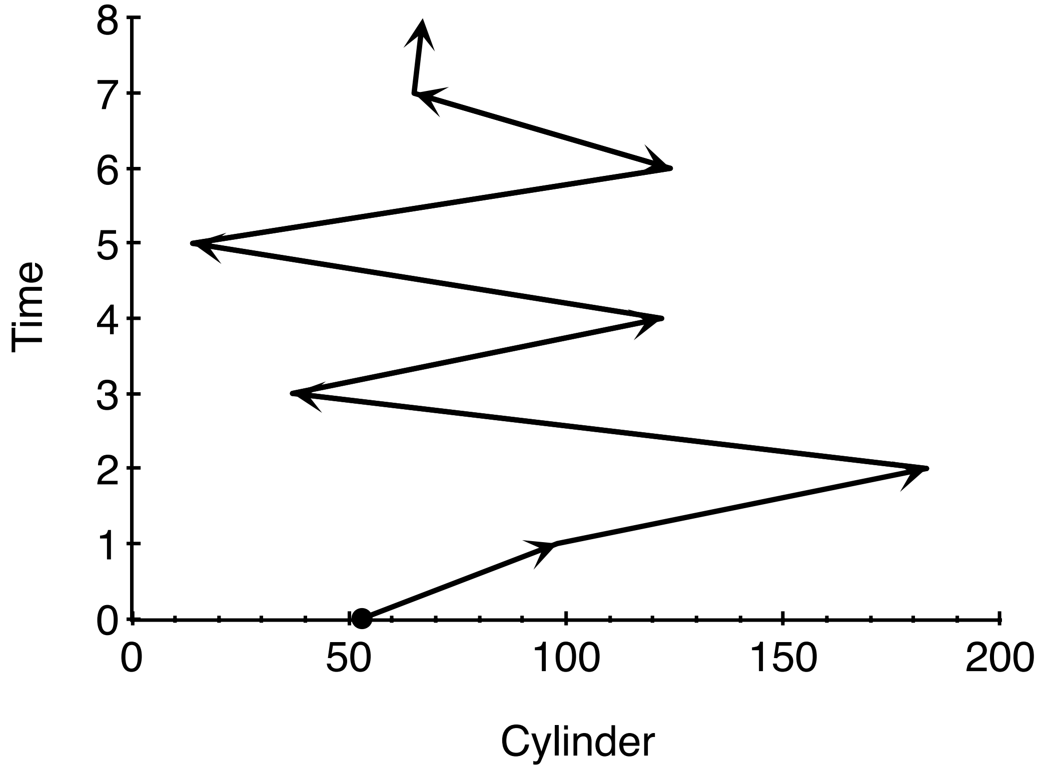
- I/O Systems
  - What are I/O Devices?
  - Disks (again):
    - Disk Performance
    - **Disk Scheduling (Hard Disk Drive: HDD)**
  - I/O System
    - I/O System Architecture
    - I/O Devices Operating

Hard Disk Scheduling

- Which disk request is serviced first?
  - FCFS
  - Shortest seek time first
  - Elevator (SCAN)
  - C-SCAN (Circular SCAN)
- Look familiar?

Hard Disk Scheduling: FIFO (FCFS)

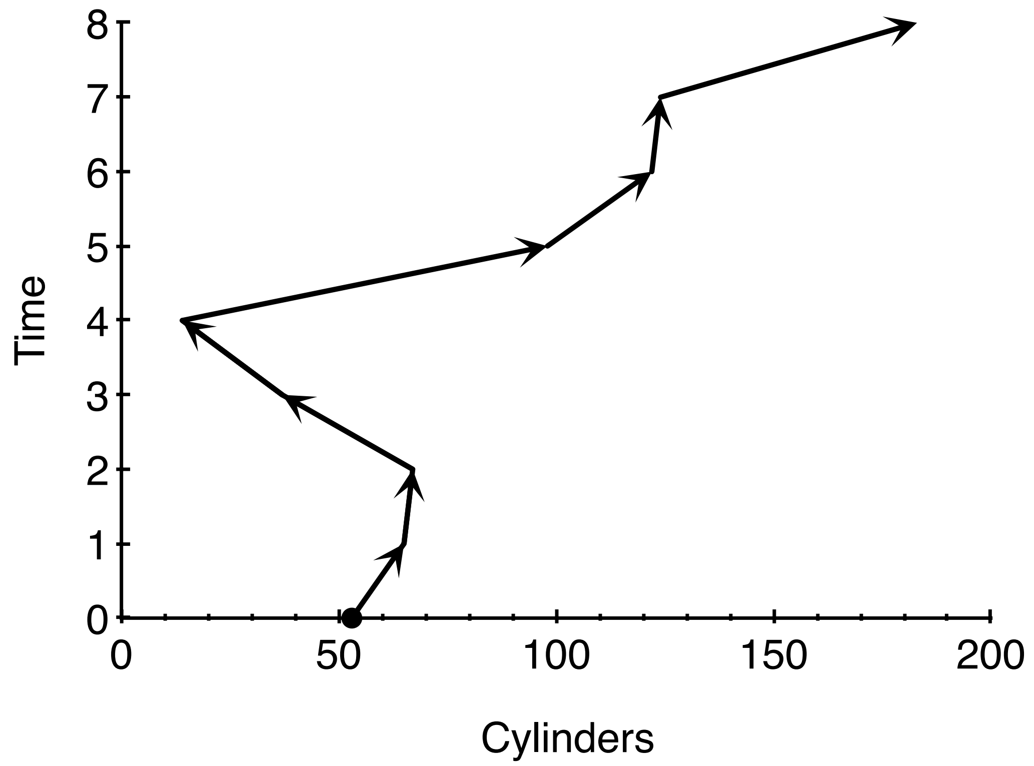
- **Method**
  - First come first serve
- **Pros**
  - Fairness among requests
  - In the order applications expect
- **Cons**
  - Arrival may be on random spots on the disk (long seeks)
  - Wild swing can happen
- **Analogy:**
  - Can elevator scheduling use FCFS?



Cylinders: 98, 183, 37, 122, 14, 124, 65, 67
Start position: cylinder 53
Direction: heading toward cylinder 200
Traversal order: 53, 98, 193, 37, 122, 14, 124, 65, 67

SSTF (Shortest Seek Time First)

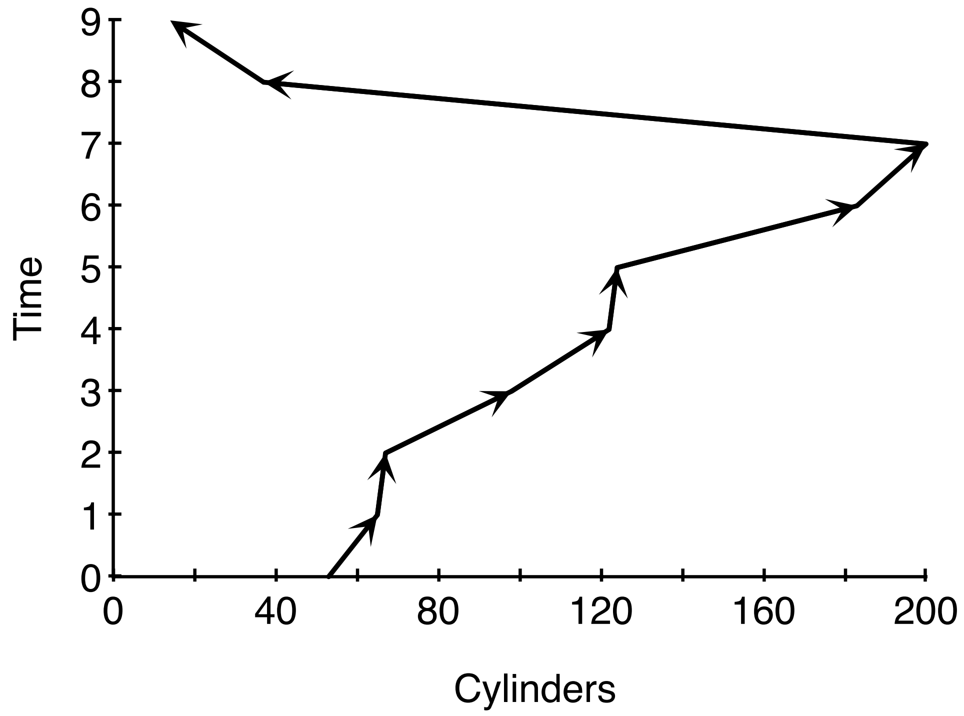
- **Method**
  - Pick the one **closest on disk**
  - Rotational Delay is in calculation
- **Pros**
  - Aims to minimize seek time
- **Cons**
  - Starvation
- **Analyses:**
  - Is SSTF optimal?
  - Can we avoid starvation?



Cylinders: 98, 183, 37, 122, 14, 124, 65, 67
Start position: cylinder 53
Direction: heading toward cylinder 200
Traversal order: 53, 65, 67, 37, 14, 98, 122, 124, 183

Elevator (SCAN)

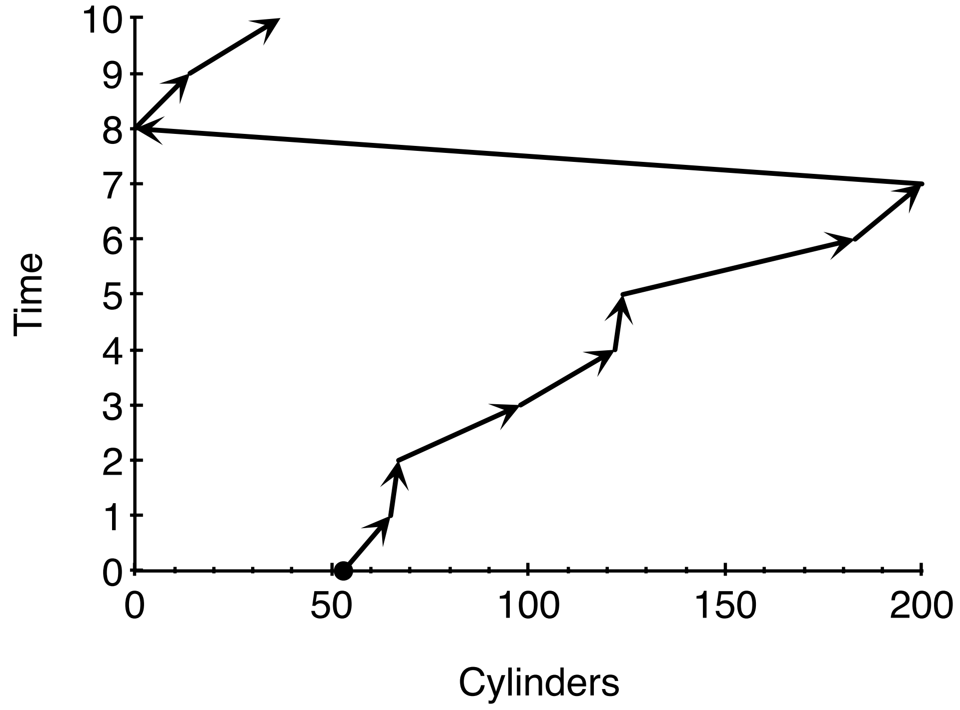
- **Method**
  - Take the **closest request in the direction of** travel
  - Travel to end, change direction
- **Pros:** Bounded time for each request
- **Cons:** Request at the other end will take time
- **LOOK algorithm**
  - Do not go to the end
  - Service last request, then change direction



Cylinders: 98, 183, 37, 122, 14, 124, 65, 67
Start position: cylinder 53
Direction: heading toward cylinder 200
Traversal order: 53, 65, 67, 98, 122, 124, 183, 200, 37, 14

C-SCAN (Circular SCAN)

- **Method:** like SCAN, but wrap around
- **Pros:** Uniform service time
- **Cons:** Does nothing on return
- C-LOOK
  - Do not go to the end
  - Service the last request, then wrap around



Cylinders: 98, 183, 37, 122, 14, 124, 65, 67
Start position: cylinder 53
Direction: heading toward cylinder 200
Traversal order: 53, 65, 67, 98, 122, 124, 183, 200, 0, 14, 37

SPTF: Shortest Positioning Time First

- So far, we've only considered track (seek time), but rotation also takes time.
- Accessing a sector with a farther track could be faster, due to less required rotation.
- Computing both track and rotation is too hard for OS
  - Hard disk devices often use SPTF internally

Other Issues

- Where to schedule disk I/O operations?
  - Both disks and the OS do that.
- I/O merging: OS can merge small I/Os into big ones
- How long should an OS wait before issuing an I/O?
  - Longer wait times may yield more merges, greater scheduling efficiency
  - But if no new I/O operations arrive, this wastes time!

Solid-State Disk Scheduling

- SSD scheduling algorithms simpler than HDDs
- Linux uses **NOOP (no scheduling)**, but nearby logical block requests combined
  - *Write amplification* (one write leads to garbage collection, many I/O ops.) hurts performance
  - File system can notify about empty blocks to be erased (TRIM on SATA SSDs)
- More info: J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Disk Schedulers for Solid State Drivers," in *Proc. ACM EMSOFT*, 2009.

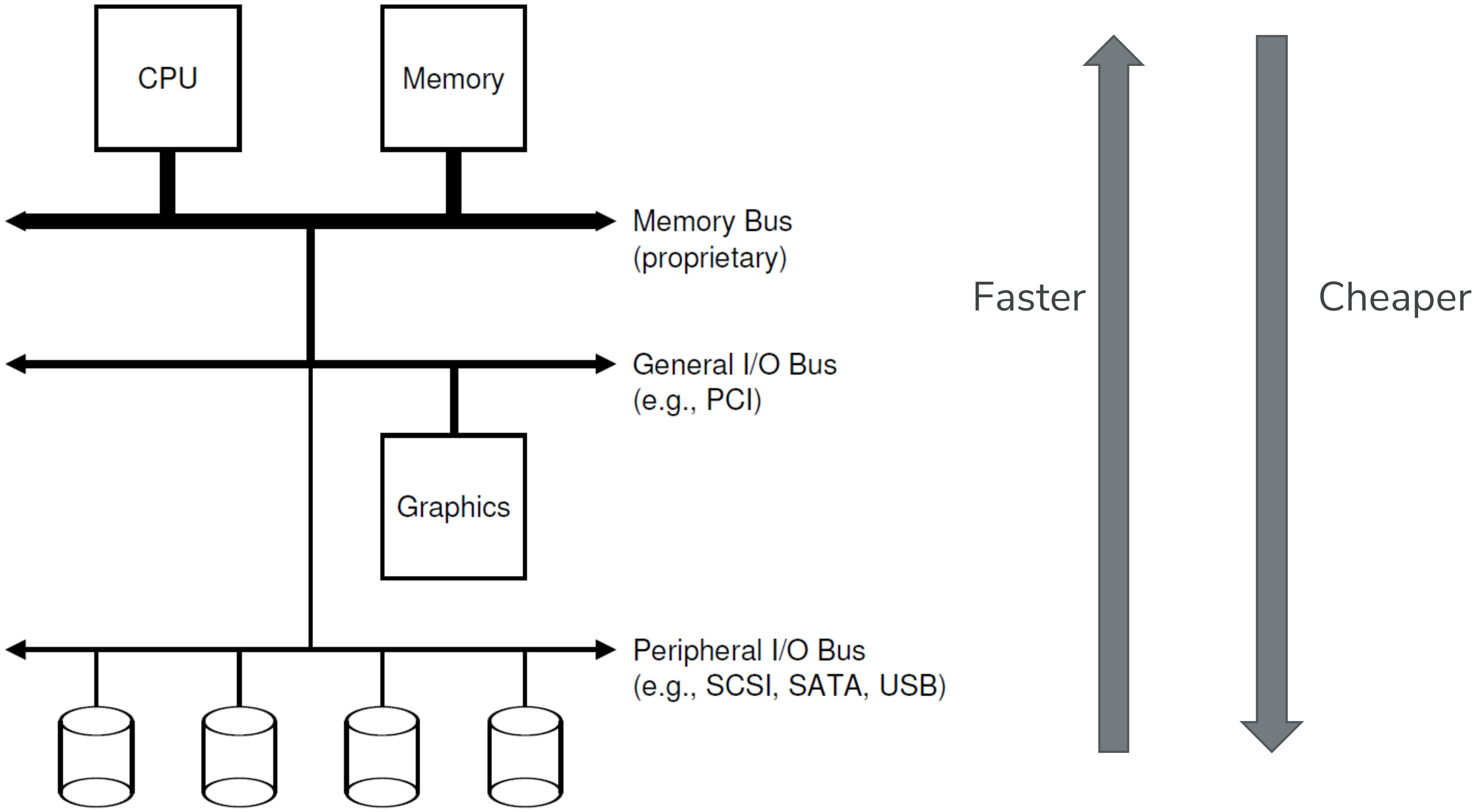
History of Disk-related Concerns

- When memory was expensive, minimize bookkeeping
- When disks were expensive, maximize number of usable sectors
- When disks became more common, increase reliability
- When processors got faster, make them appear faster
- **Key point:** find, improve **bottlenecks, single points of failure**

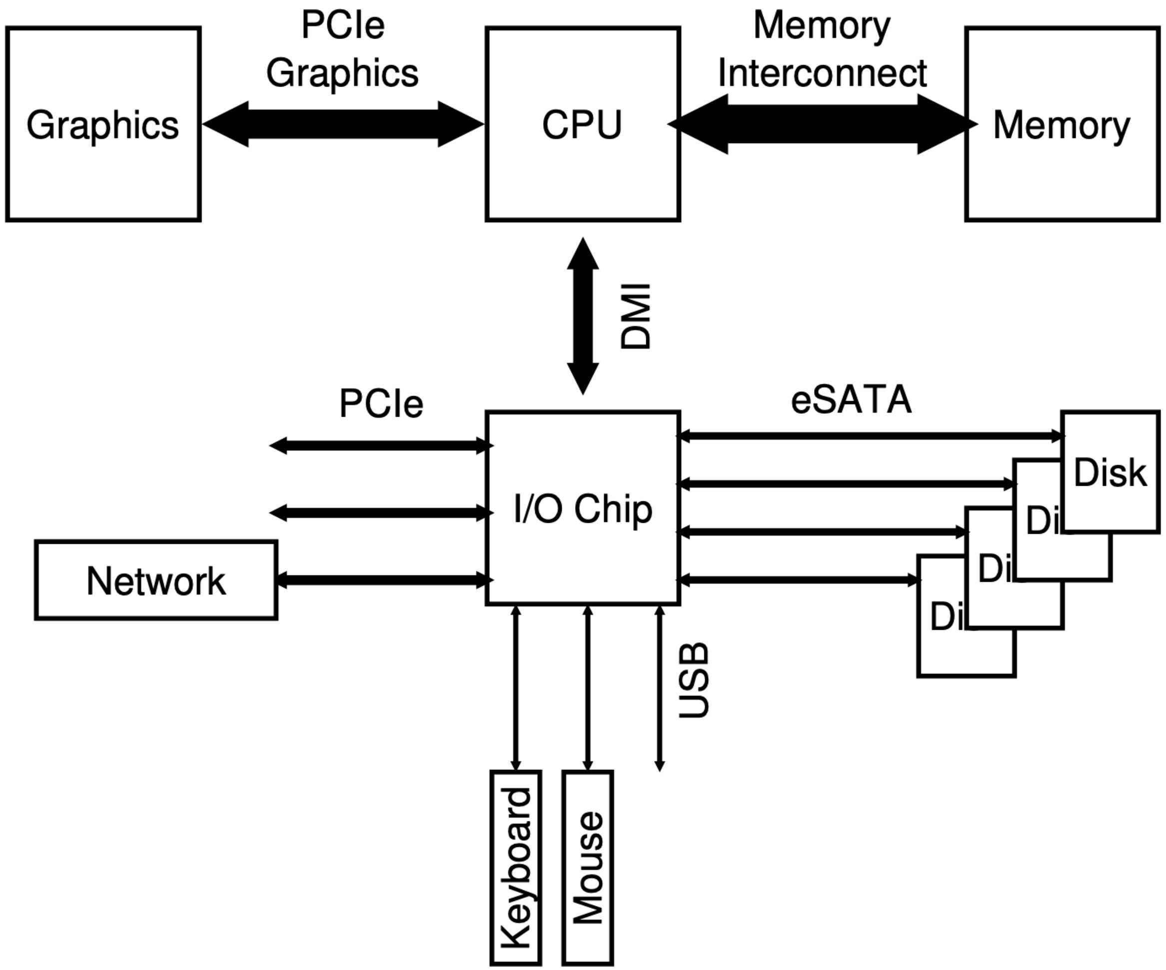
Outline: I/O and File System (part 1)

- I/O Systems
  - What are I/O Devices?
  - Disks (again):
    - Disk Performance
    - Disk Scheduling
  - **I/O System**
    - **I/O System Architecture**
    - **I/O Devices Operating**

Architecture

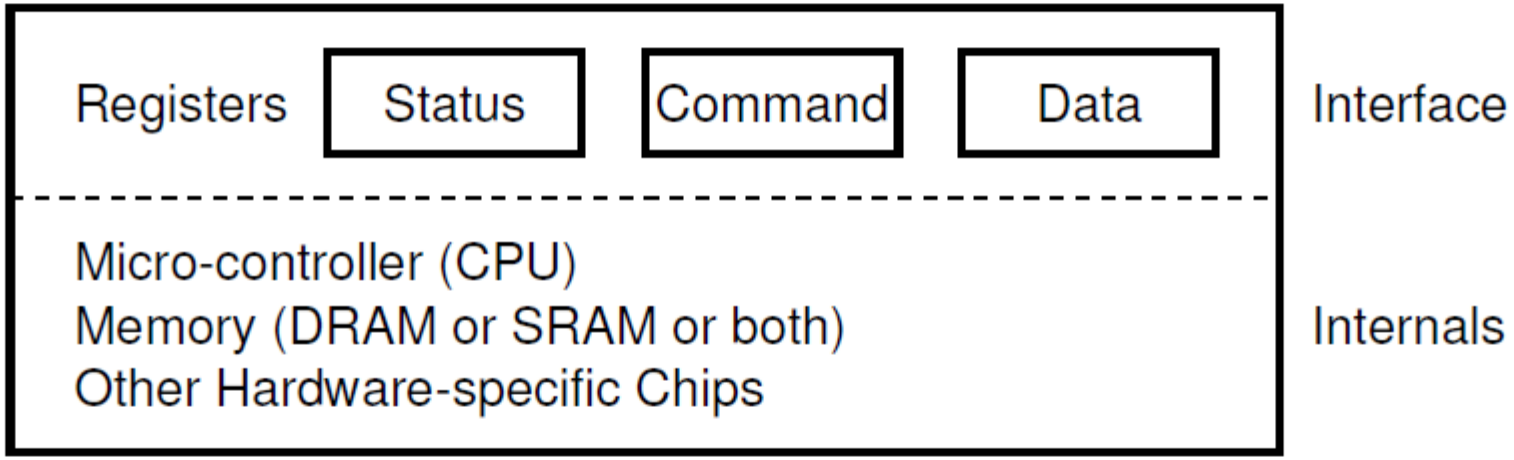


Architecture (Modern)



A canonical device

- Interface: usually a set of registers
- Internal structure: vary a lot



Operating the device

- A typical piece of pseudo code

```
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
```

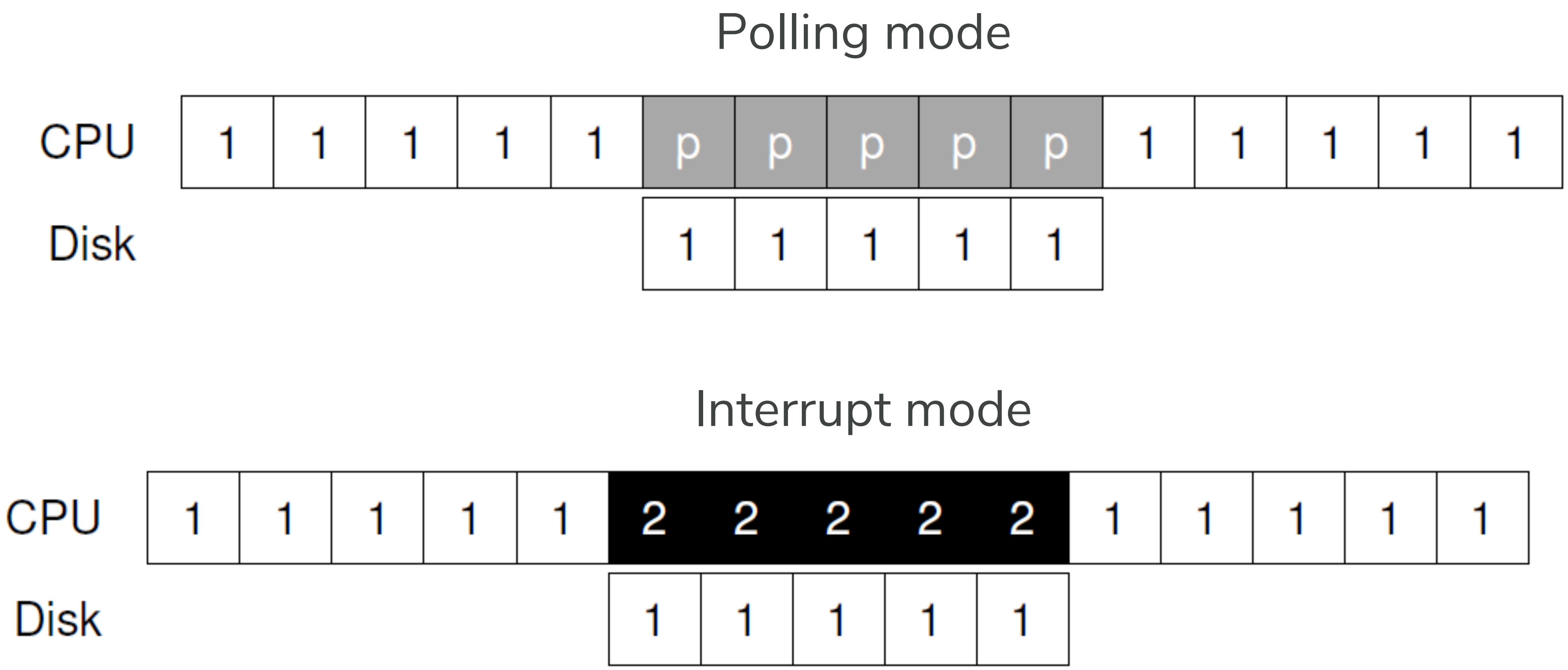
- It continuously polls registers to check state (polling mode)
- It requires CPU to write data to registers (PIO mode)

Polling vs. Interrupts

- Polling consumes CPU, so polling for long is not efficient

- Alternative solution: interrupt
  - OS registers an interrupt handler to CPU
  - Instead of polling, OS puts the process that is performing I/O to sleep
  - OS may switch to another process
  - When the I/O device is ready or has completed its job, it issues an interrupt
  - CPU calls interrupt handler and may wake up the original process

Polling vs. Interrupts



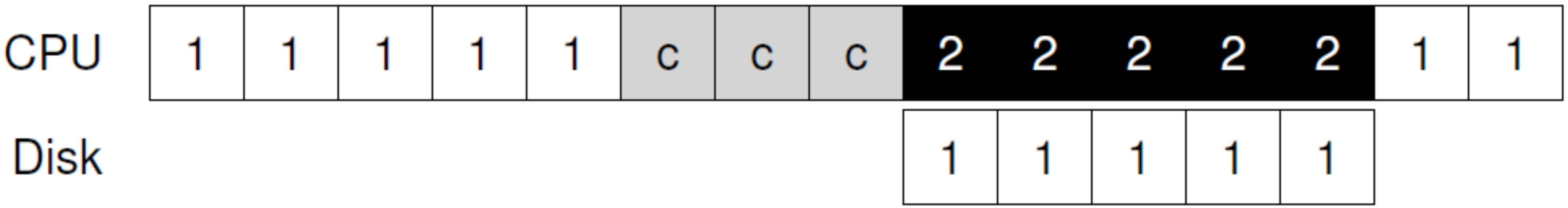
Polling vs. Interrupts

- Interrupt has its own overhead: it needs to perform two context switches

- For fast devices, polling may be a better solution. Why?
  - If every polling takes short, its overhead may be smaller than two context switches

Programmed I/O vs Direct Memory Access

- PIO (programmed I/O): CPU moves data from its own register to I/O devices' register (may need to move data from memory first)
  - CPU cannot do other things when moving data



- For fast I/O devices, PIO can consume a lot of CPU.

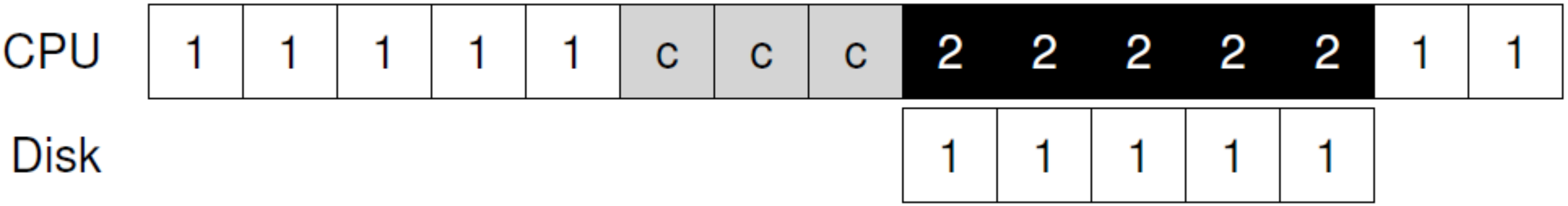
Programmed I/O vs Direct Memory Access

- Alternative solution: **DMA (Direct Memory Access)**. Modern computer has some special hardware (DMA engine) that can transfer data across devices

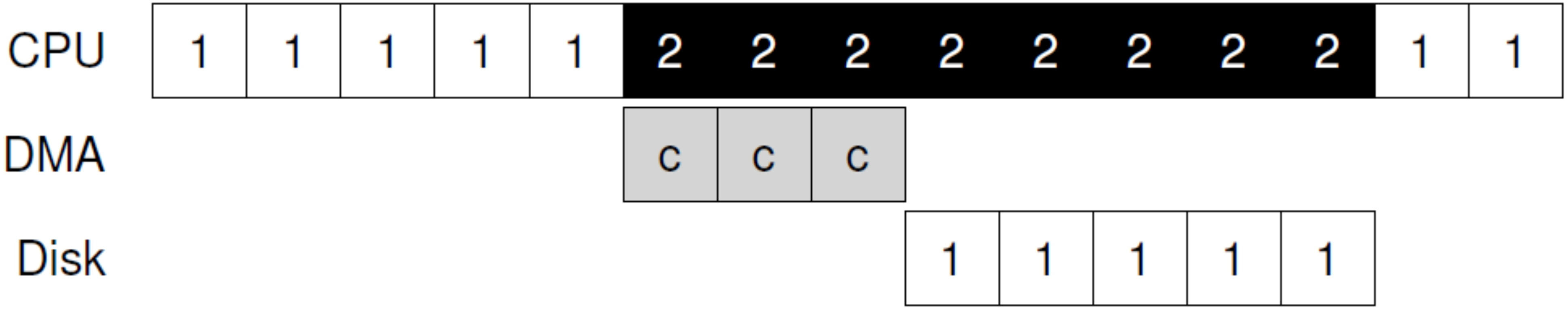
- DMA usage:
  - OS tells DMA engine location of data, size of data, and destination of data
  - OS is done and can switch to other work; DMA engine copies data in the meantime
  - When DMA is complete, DMA engine raises an interrupt
  - OS realizes data copy is complete

Programmed I/O vs Direct Memory Access

PIO



DMA



How to read/write device registers?

- One solution: I/O instructions (e.g. “in” and “out” on x86)
  - They are privileged instructions

- Another solution: memory-mapped I/O
  - Map device registers to some specific memory locations
  - Then operate them with “load” and “store” instructions
  - These memory locations are not visible to user programs.

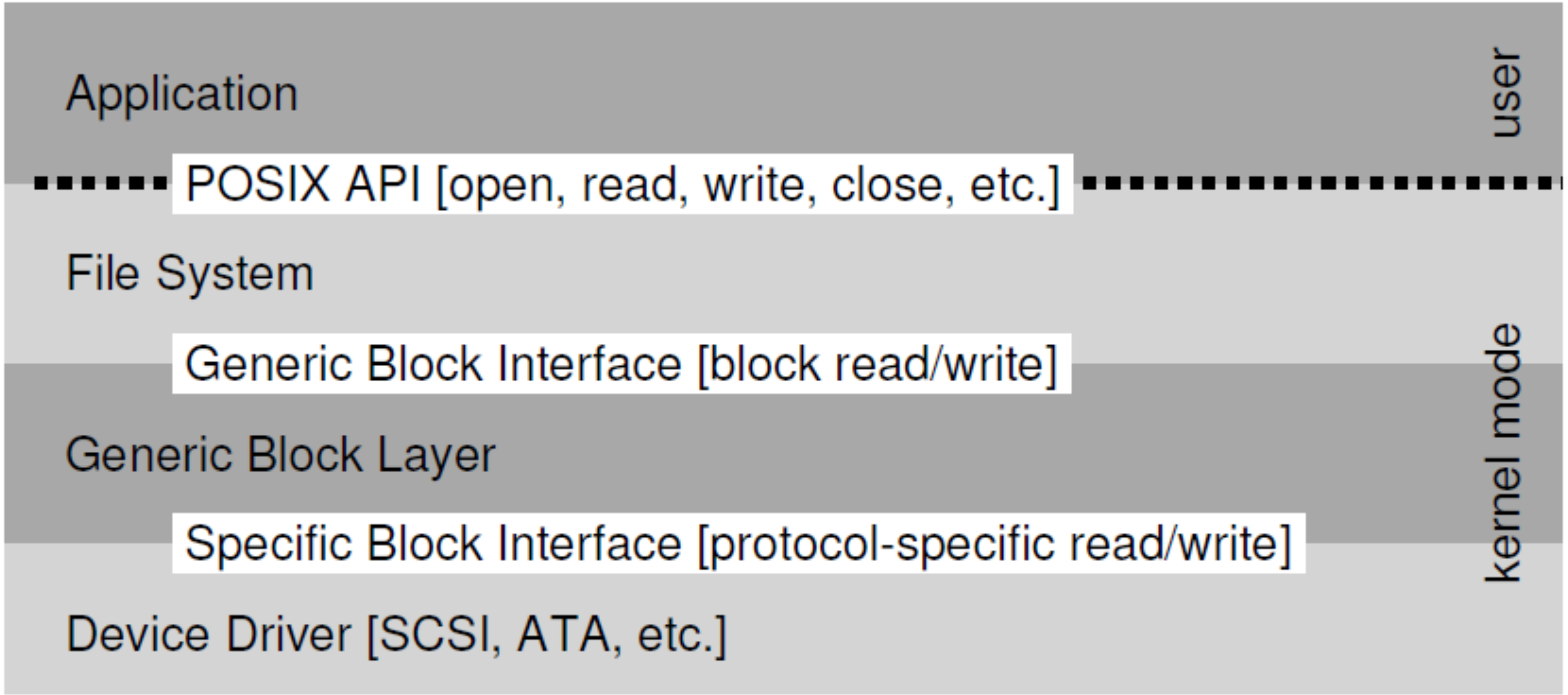
- No fundamental differences between these two solutions

Go back to our original question

- How can OS operate so many kinds of I/O devices?
  - Each device has different registers, commands, and data formats.
  - New devices may be released after OS.
- After you learn to drive, you can drive different kinds of cars. How does this happen? Cars may have different engines, transmissions, etc.
  - Reason: all cars follow a similar specification (steering wheel, brake, etc).
- OS uses a similar solution
  - OS defines a specification for each type of I/O device (e.g. disk must implement read/write operations)
  - Device manufacturer implements this specification in a device driver (e.g. how to do read/write by operating disk registers)
  - After you install a new device, you must also install its device driver.

Device drivers

- Multi-level abstractions



Facts about Device Drivers

- 70% of Linux kernel code is actually for device drivers
  - Windows probably has a similar number
- Compared to OS core code, device drivers are often not written by kernel programming experts and are often not well tested
- As a result, they are a primary contributor to bugs and kernel crashes
  - It may be a mistake to blame Microsoft for “blue screen”

Case study: A simple IDE disk driver

- High-level abstraction: OS requires all block devices (disk is a kind of block device) must implement two core functions: readBlock and writeBlock
- Low-level abstraction: a disk has multiple registers, including control, command block, status, and error. OS does not know such information. Only disk manufacturer knows.
- Disk manufacturer publishes a disk driver that implements readBlock and writeBlock by manipulating those registers
  - Read details in the book

Summary (Key points)

- Disk Performance (how to calculate seek time, rotational time, disk transfer time, as well as the total/average time, and throughput)
- Disk Scheduling (FCFS, SSTF, SCAN, CSCAN, SPTF)
- Polling vs. Interrupt
- Programmed I/O vs. DMA