I/O, Files, & Storage Devices (Part I)

Professor Travis Peters
CSCI 460 Operating Systems
Fall 2019

Some slides & figures adapted from Stallings instructor resources.

Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

Some content adapted from the Disk Scheduling Algorithms tutorial
http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html

Today

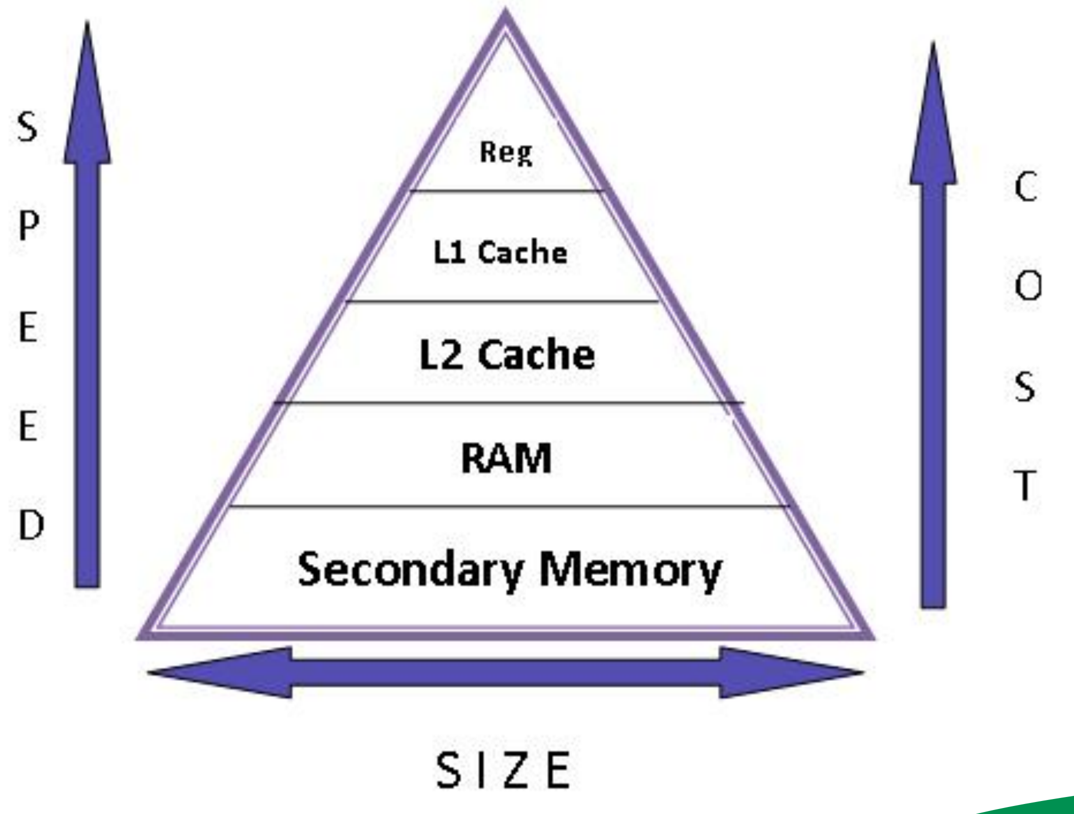
Announcements

- Project Proposals Due **FRIDAY**!

Goals & Learning Objectives

- Understand key concepts behind I/O devices and functions
- Understand some of the key issues in the design of OS support for I/O
- Understand basics of secondary storage (emphasis on disks and disk scheduling)
- Understand basics behind files and file systems

Memory Hierarchy: Respective Sizes & Access Times



LEVEL	ACCESS TIME	TYPICAL SIZE
Registers	"instantaneous"	under 1KB
Level 1 Cache	1-3 ns	64KB per core
Level 2 Cache	3-10 ns	256KB per core
Level 3 Cache	10-20 ns	2-20 MB per chip
Main Memory	30-60 ns	4-32 GB per system
Hard Disk	3,000,000-10,000,000 ns	over 1TB

I/O and Disks and Files, Oh My!

Categories of I/O Devices

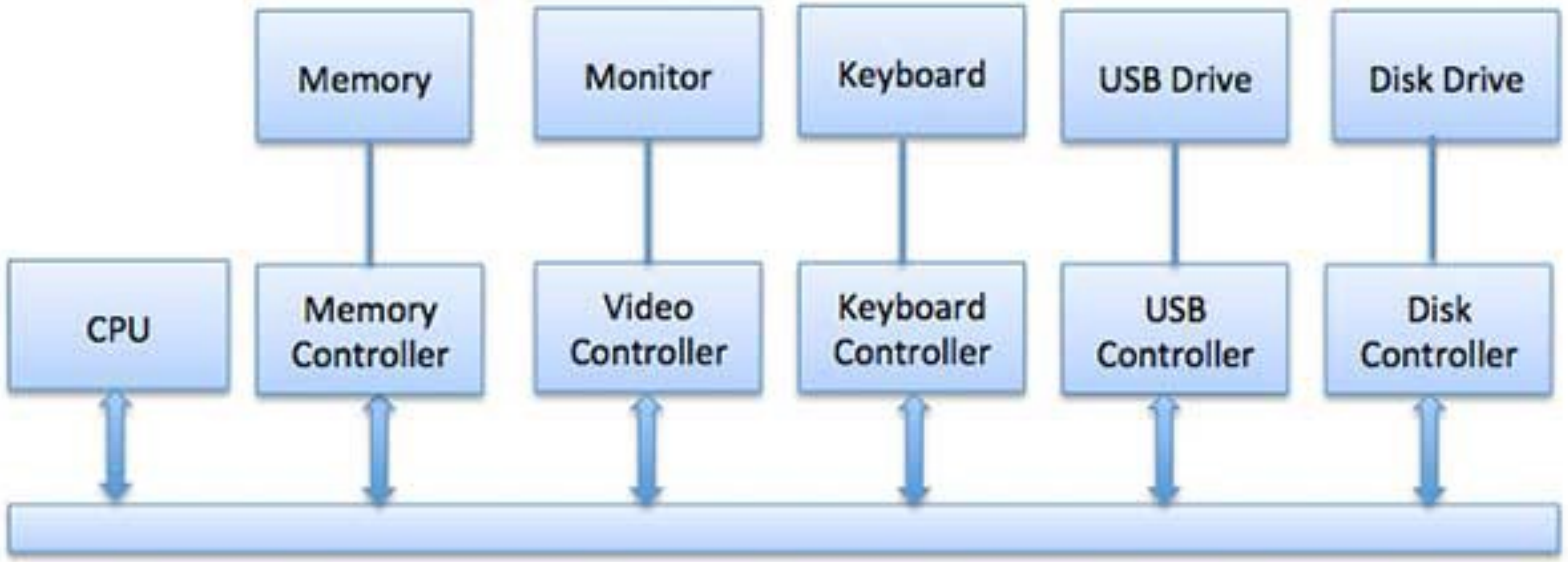
- Human Readable
  Computer-User Interactions (e.g., printers, terminals, keyboards, mice)
- Machine Readable
  Computer-Device Interactions (E.g., disk drives, controllers, sensors, actuators)
- Communication
  Computer-Computer Interactions (E.g., network devices, modems)

Block-oriented devices

Store information in (usually fixed-sized) blocks

Stream-oriented devices

Transfer data in/out as a stream of bytes (no block structure)



https://medium.com/cracking-the-data-science-interview/the-10-operating-system-concepts-software-developers-need-to-remember-480d0734d710

I/O and Disks and Files, Oh My!

I/O Approaches

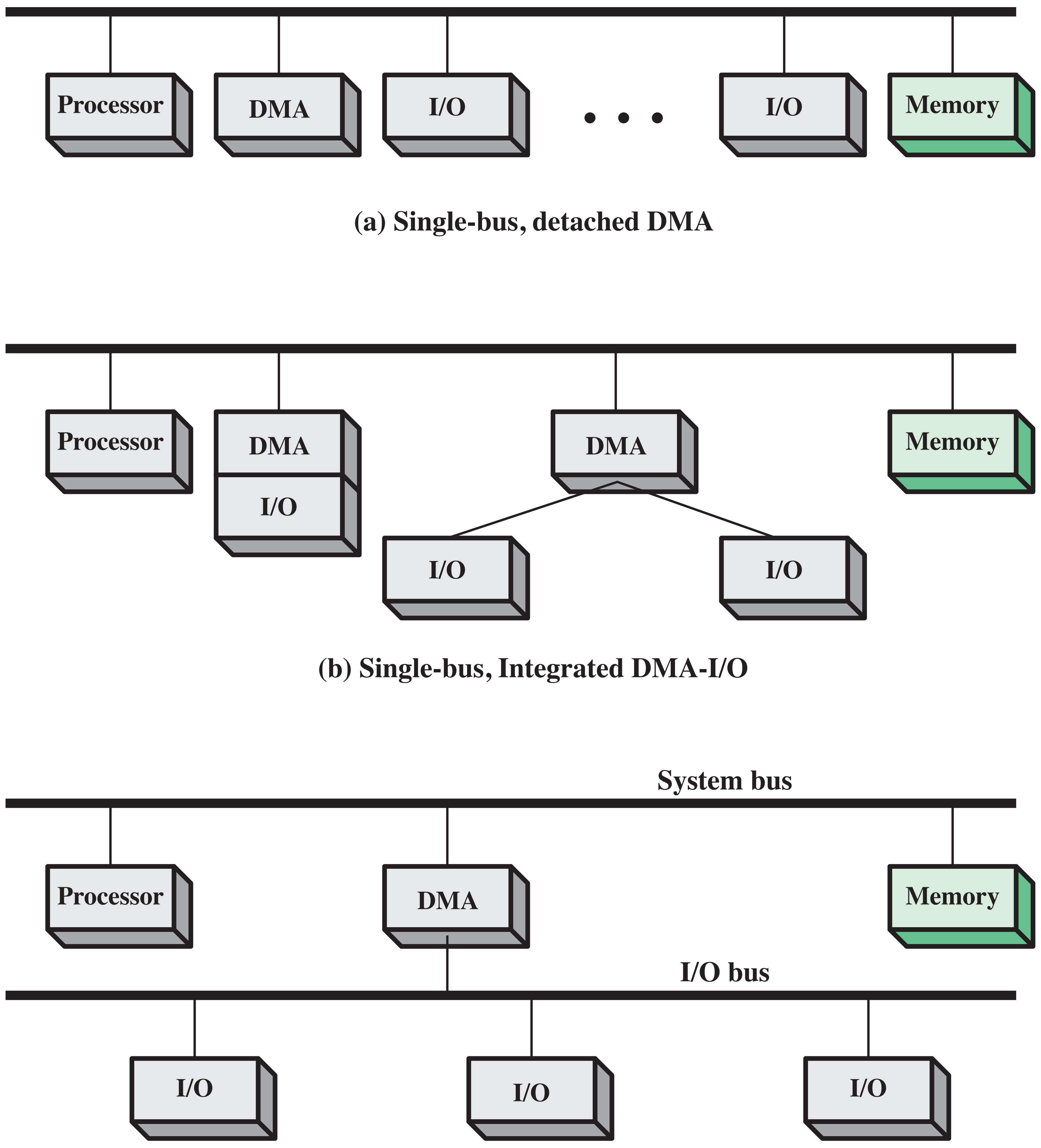
- Direct I/O
  Processor directly controls I/O device(s)
- Programmed I/O
  processor issues command; busy wait until complete; no interrupts
- Interrupt-Driven I/O
  processor issues command; continue running (or block); interrupt when done; uses interrupts, obviously
- Direct Memory Access (DMA)
  control exchanges between MM and I/O devices w/ minimal involvement of the processor (only needed at beginning/end)

Direct Memory Access (DMA)

- **Role of DMA:**
  transfer data to/from memory over the system bus

- I/O processor* (which realizes DMA) mimics the processor and its ability to take control over the system bus.

- A progression can show that more efficient implementations move all I/O off of the system bus onto a dedicated I/O bus



*I/O module more closely resembles a dedicated I/O processor for handling I/O

What should I/O devices provide?

Overview of I/O, Files, & Storage Devices

Overarching Objectives

Modern CPUs e.g., many BILLIONS of instructions per second

Modern memory systems e.g., 2-4GB/sec bandwidth

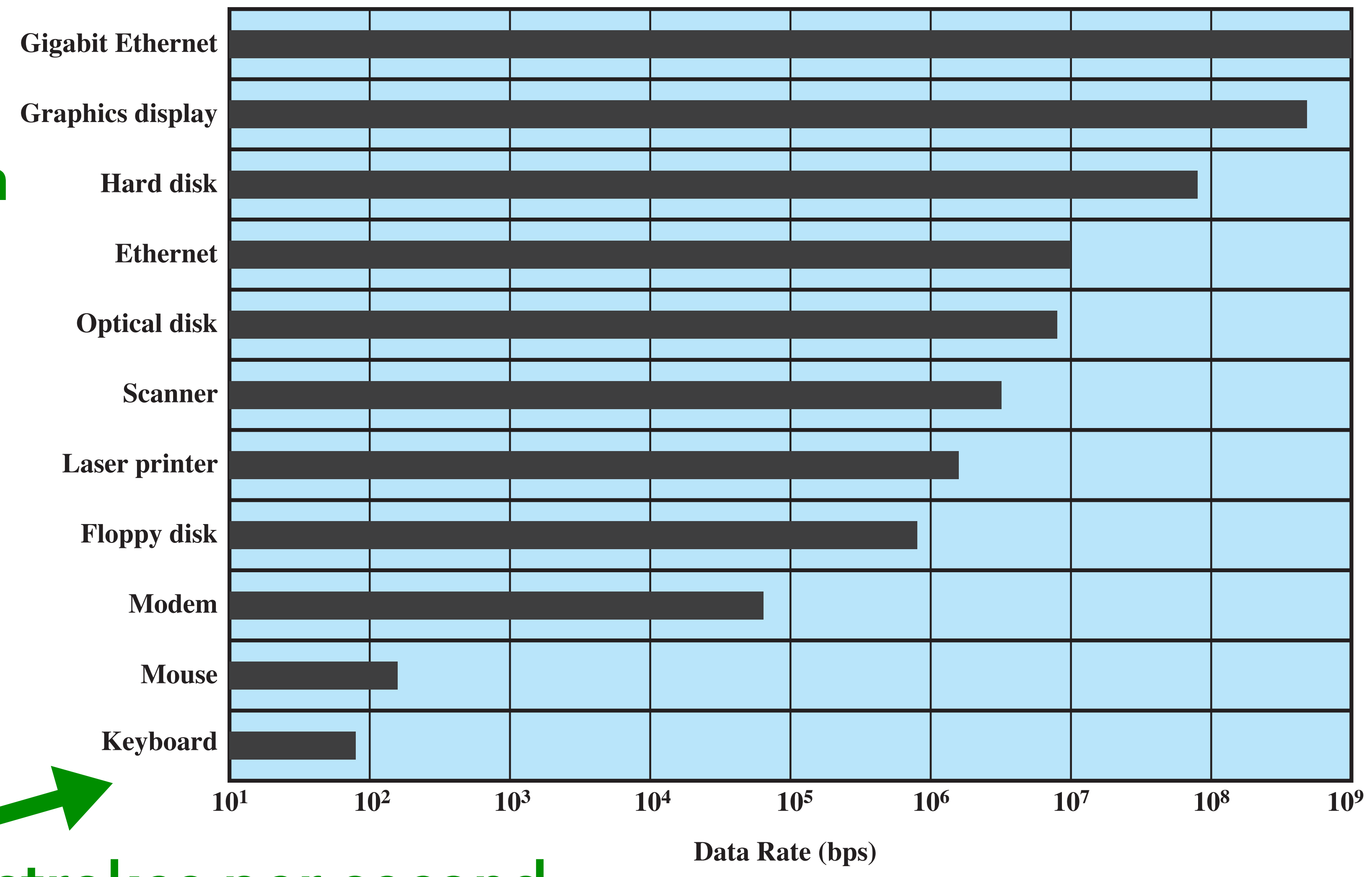
1. **Efficiency**

   I/O is slowww compared to modern processors/memory systems

   I/O is often the bottleneck

   Want I/O to be as efficient as possible!

E.g., 10s-100s Mbits/sec



E.g., 9-10 keystrokes per second

Overview of I/O, Files, & Storage Devices

Overarching Objectives

1. **Efficiency**

   I/O is slowww compared to modern processors/memory systems

   I/O is often the bottleneck

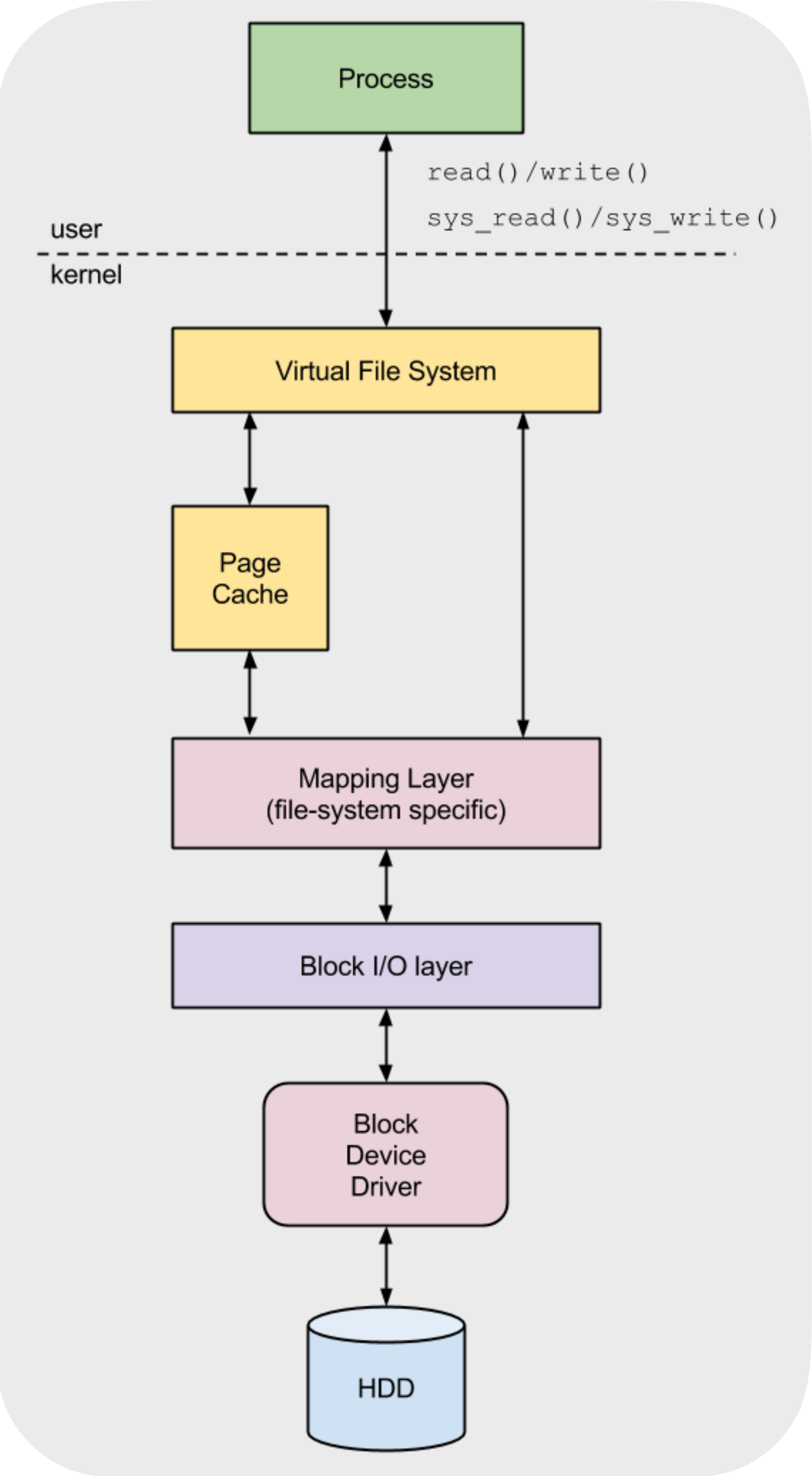
   Want I/O to be as efficient as possible!

2. **Generality**

   Want simplicity and correctness

   Want support for diverse range of devices

   *→Lots of abstractions — most things can be done with **read/write**, **open/close**, **lock/unlock**.*



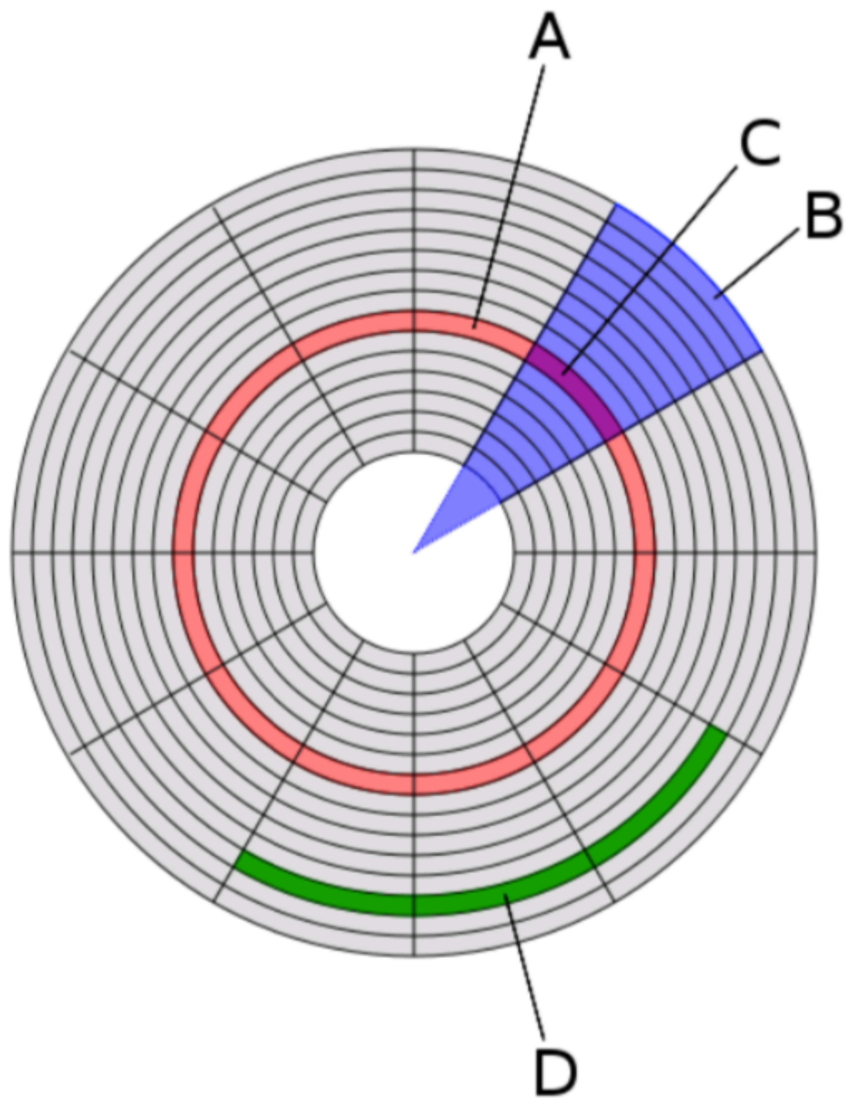
Disks & Disk Scheduling

What is a disk?

How does a disk work?

Disk Scheduling

A = Track / B = Sector / C = Sector of Track / D = File



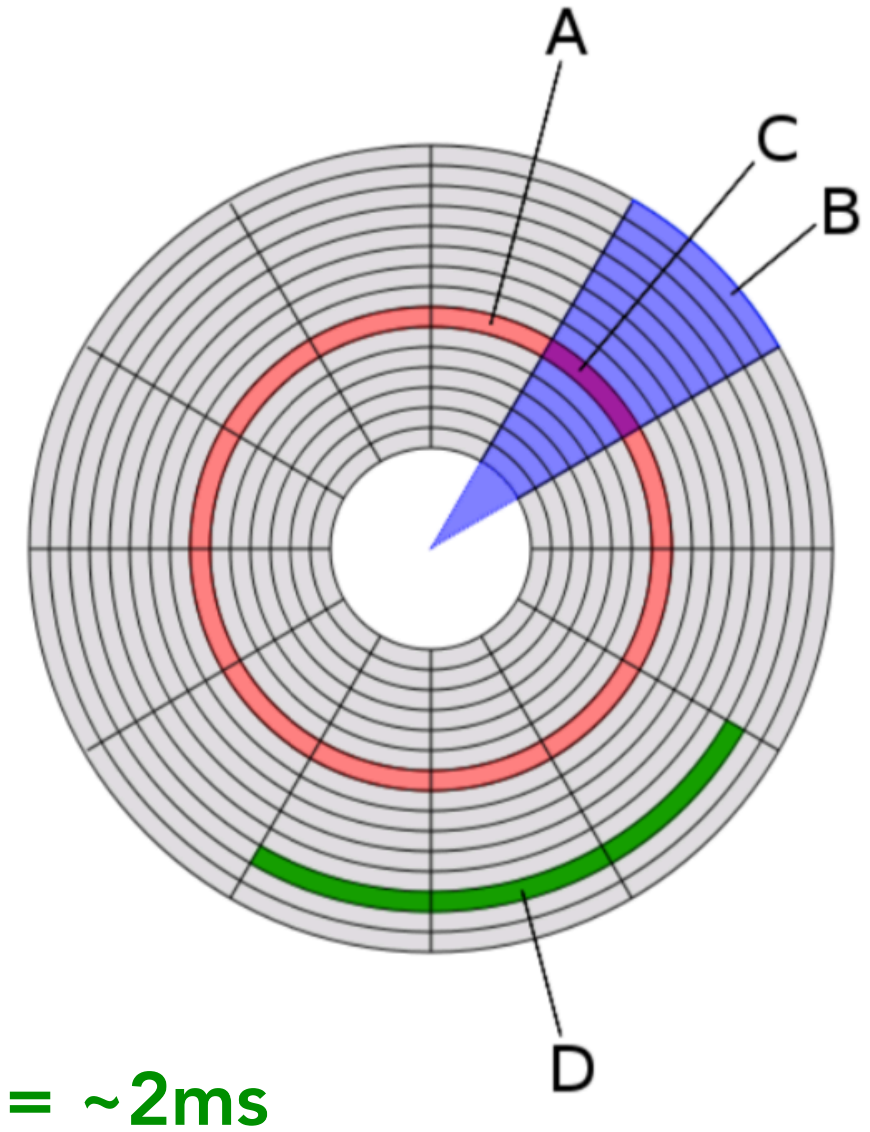
- **The Basics**
  - Reading/writing data to/from a disk is done with a read/write head
  - Disk rotates
  - Head must move (**seek**) to appropriate position (**track** / **sector**) on the disk

<u>Disk Scheduling Decision</u>

Given a series of access requests (reads/writes), on which track should the disk arm be placed next to maximize fairness, throughput, etc?

Disk Scheduling

A = Track / B = Sector / C = Sector of Track / D = File



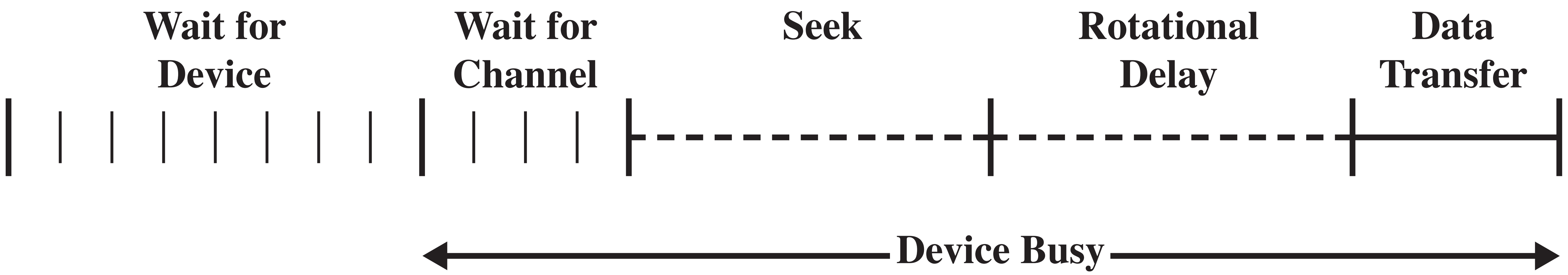
- **The Basics**
  - Reading/writing data to/from a disk is done with a read/write head
  - Disk rotates
  - Head must move (**seek**) to appropriate position (**track** / **sector**) on the disk

- **Timing of Disk I/O Transfer**
  - **Seek Time**
    time taken to move disk arm to a specified track
  - **Rotational Latency / Rotational Delay**
    time to wait for sector to reach the head
  - **Transfer Time**
    time to transfer (read/write) data
  - **Access Time**
    = seek time
    + rotational latency
    (+ transfer time)

E.g., RD = ~2ms
(assuming ~15K rpm)

$T = \frac{b}{rN}$

E.g., ST = ~10ms

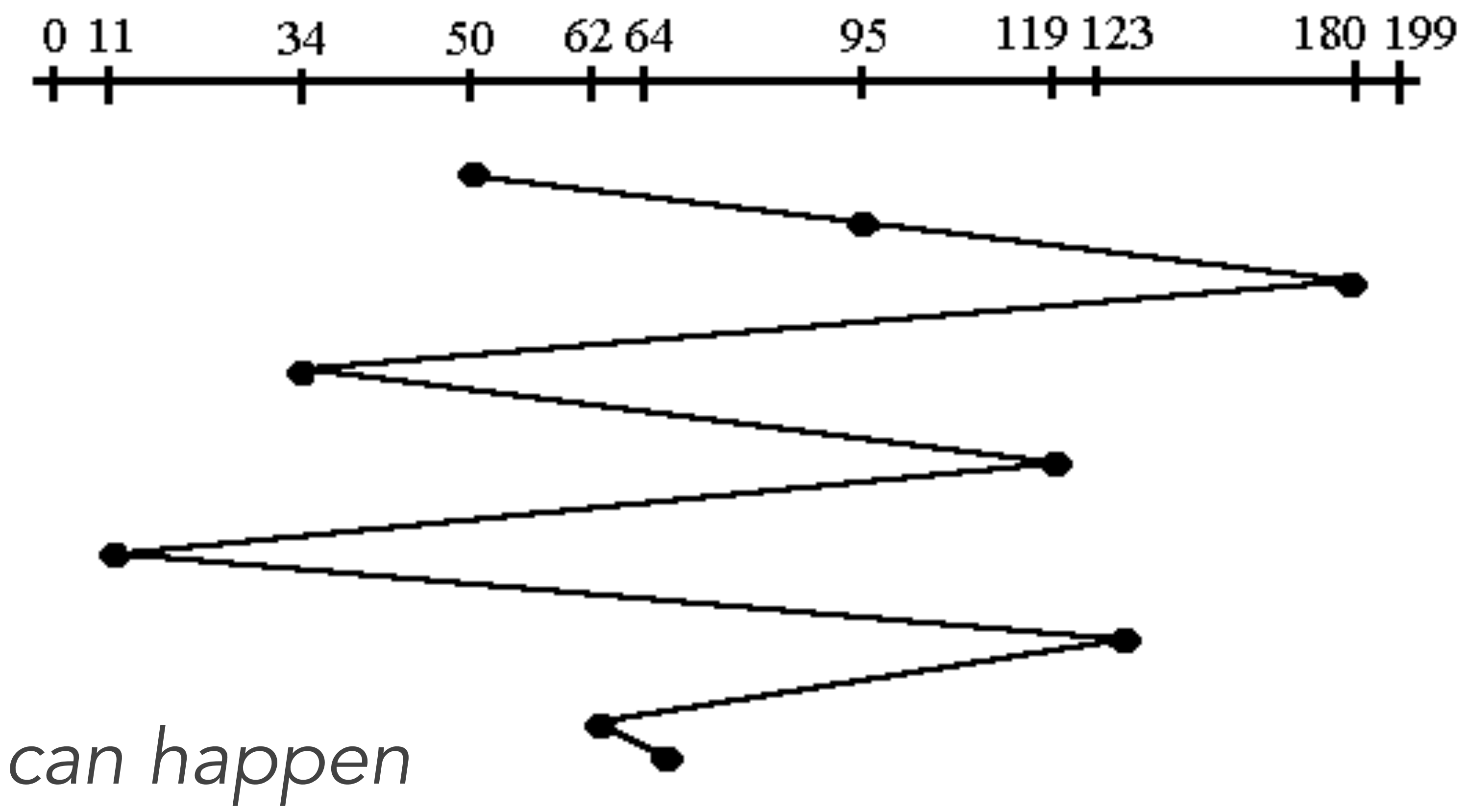


$\rightarrow T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$

Disk Scheduling Policies

Order of ops matter! → seek time can have tremendous impact on access time

Consider multiple processes queued up on various I/O requests…



how to schedule I/O ops?

- **Random Scheduling** — only really useful as a benchmark/baseline…
- **FIFO/FCFS** — approx. equivalent to random scheduling!
  - ***Pros?*** *fairness among requests, requests handled in well-defined order*
  - ***Cons?*** *arrival may be on random spots on the disk (long seek times); wild "swings" can happen*
  - ***Analogy?*** *FCFS elevator scheduling?*
- **Priority** — not targeted at optimal disk utilization (rather, more concerned w/ OS objectives)
- **LIFO** — always service the most recent process
  - **Why might this be good/bad?**
  - → results in very little arm movement *(principle of locality!)*
  - → could lead to starvation…

Can we do better?

What if the scheduler knew about the current track / position of the head?

Extras

Things alluded to in class but not fully covered

Disk Cache

- **Delivery Strategies**
  - DMA to user process memory
  - Shared memory w/ the disk cache

- **Replacement Strategies**
  - *Recall*: LRU
  - Also, *Least Frequently Used (LFU), Frequency-based Replacement…*

    → *only shown to be slightly better than LRU*

Flash Memory *(1 Slide Only…)*

- **EEPROM** (Electronically Erasable Programmable Read Only Memory)
  - **Ex:** NAND Flash
  - READ performance
    - Random READ: 25μs, Sequential READ: 25ns
  - WRITE performance
    - PROGRAM PAGE: 25μs, BLOCK ERASE: 1.5ms
  - Endurance: 100,000 PROGRAM/ERASE cycles

- **In The News**

*Tesla's cars use **Linux** and do **a huge amount of logging**. According to 057 Technology's Jason Hughes, "The information logged here is pretty much useless on production vehicles. Unless a developer has a specific reason for enabling it, it does the customer no good. These logs are also rarely downloaded by Tesla." That **excessive logging is causing the flash memory in the Media Control Unit (MCU) v1 to fail**, which causes the car to **lose touch screen functionality, which in a Tesla controls most everything**. V1 units are in model S and X Teslas made before 2018.*
— https://www.tomshardware.com/news/flash-memory-wear-killing-older-teslas-due-to-excessive-data-logging-report

- **GANGRENE: Exploring the Mortality of Flash Memory**
  https://www.usenix.org/system/files/conference/hotsec12/hotsec12-final4.pdf