# A Tour to Computer Systems

## Introduction to Computer Systems

2023. Spring

Xi'an Jiaotong University

Danfeng Shan

# About me

- Danfeng Shan (https://dfshan.github.io)
- Contact
  - E-mail: dfshan@xjtu.edu.cn (I would read every email)
  - Telegram: antsshan
- Office
  - Room 4-6076, Hongli Building, iHabour

# Advices for the Course

- Manage your Time Wisely
  - ✗ Fall behind and try to catch up just before deadline ☹
  - ✓ Never fall behind 😄
  - We expect 2 weeks of concentrated effort for each assignment
- Don't cheat
  - The consequences will be much worse than not doing the assignment at all
- Ask for help
- Your health is most important

# Course Overview

What you are about to learn

Details of each component

How programs are executed?

Write a program

What you have known

# Preliminaries

- [Essential] C programming
- [Essential] Linux
  - [Essential] command line
  - [Essential] ssh
  - [Essential] gcc
  - [Helpful] vim
  - [Helpful] Make / Cmake / bazel
- [Helpful]  Git
- [Helpful] Google

# Overview

- Representing Program
- Translating Program
- Executing Program
  - Hardware Organization
- Memory Architecture
- Operating System
- Network

# Representing Program

*code/intro/hello.c*

```
1   #include <stdio.h>
2
3   int main()
4   {
5       printf("hello, world\n");
6   }
```

*code/intro/hello.c*

The `hello` program

# Representing Program

- Source program from the computer's perspective
  - A sequence of bits (0 or 1)
  - 8-bit chunks → bytes
  - Each byte represents some text character
  - ASCII standard

| # | i | n | c | l | u | d | e | <sp> | < | s | t | d | i | o | . |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35 | 105 | 110 | 99 | 108 | 117 | 100 | 101 | 32 | 60 | 115 | 116 | 100 | 105 | 111 | 46 |
| h | > | \n | \n | i | n | t | <sp> | m | a | i | n | ( | ) | \n | { |
| 104 | 62 | 10 | 10 | 105 | 110 | 116 | 32 | 109 | 97 | 105 | 110 | 40 | 41 | 10 | 123 |
| \n | <sp> | <sp> | <sp> | <sp> | p | r | i | n | t | f | ( | " | h | e | l |
| 10 | 32 | 32 | 32 | 32 | 112 | 114 | 105 | 110 | 116 | 102 | 40 | 34 | 104 | 101 | 108 |
| l | o | , | <sp> | w | o | r | l | d | \ | n | " | ) | ; | \n | } |
| 108 | 111 | 44 | 32 | 119 | 111 | 114 | 108 | 100 | 92 | 110 | 34 | 41 | 59 | 10 | 125 |

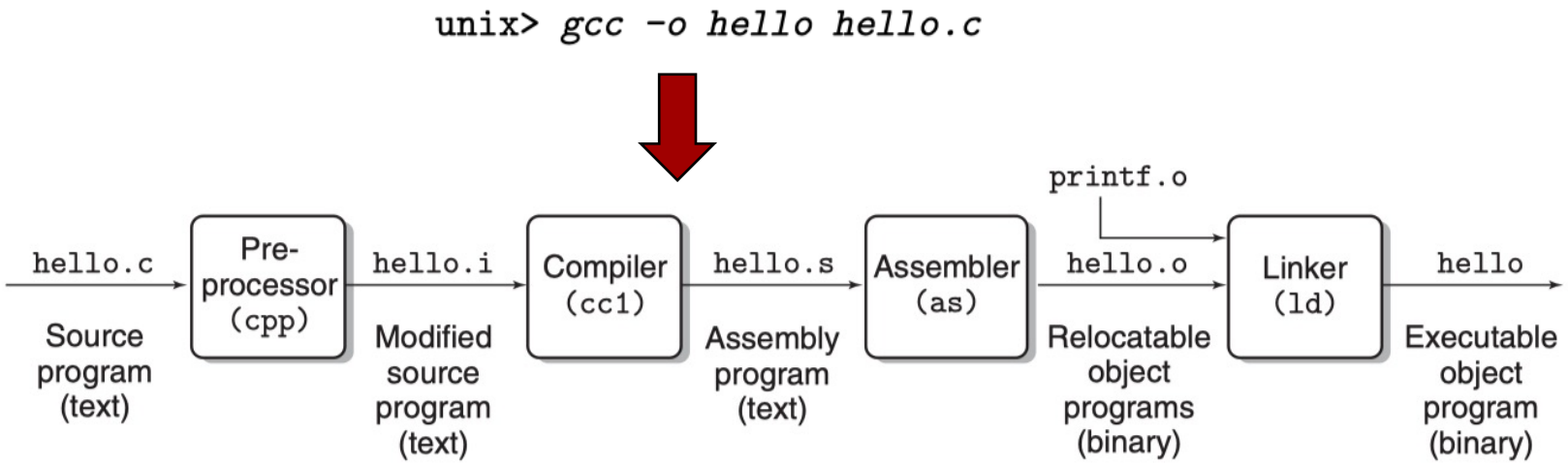
What about Chinese Character?

# Representing Program

- Source program from the computer's perspective
  - A sequence of bits (0 or 1)
  - 8-bit chunks → bytes
  - Each byte represents some text character
  - ASCII standard
- All information in a system is represented as a bunch of bits
  - Integer, floating number, text character, …
  - How to distinguish?
    - Contexts!
- Lessons Leaned
  - As a programmer, we need understand machine representations of numbers

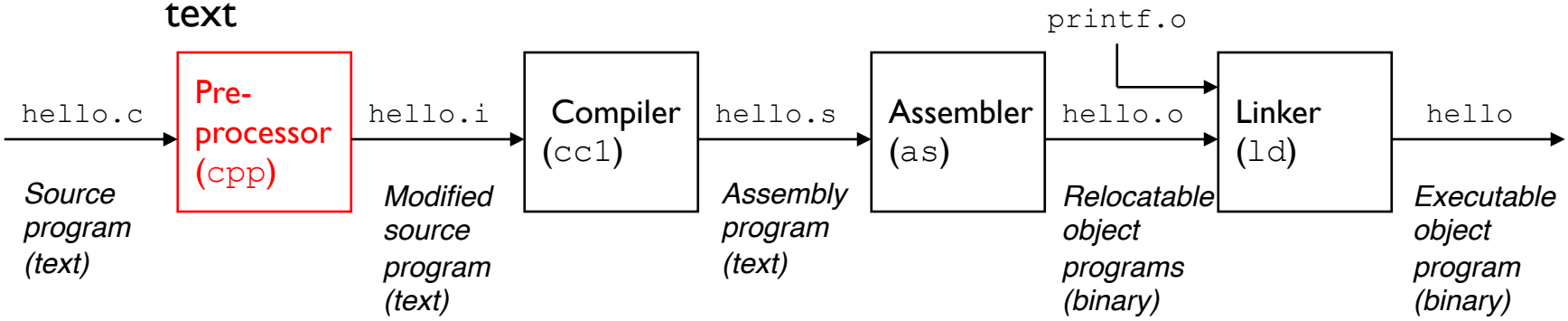
# Translating Program

- C program is a high-level language
  - Why high-level language: Easy to be understood by human
- Machine only execute instructions (i.e., low-level *machine language*)
  - A binary disk file (called executable object files)



# Translating Program

- Preprocessing phase (cpp)
  - Modifies the original C program according to directives that begin with the # character
  - `hello.c`: Read the contents of `stdio.h` and insert it into the program text



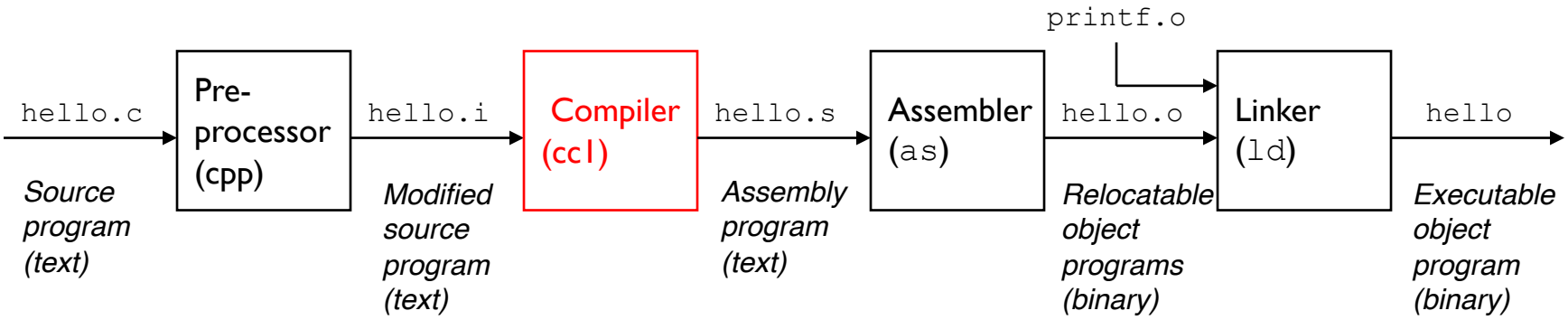

code/intro/hello.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("hello, world\n");
6  }
```

code/intro/hello.c

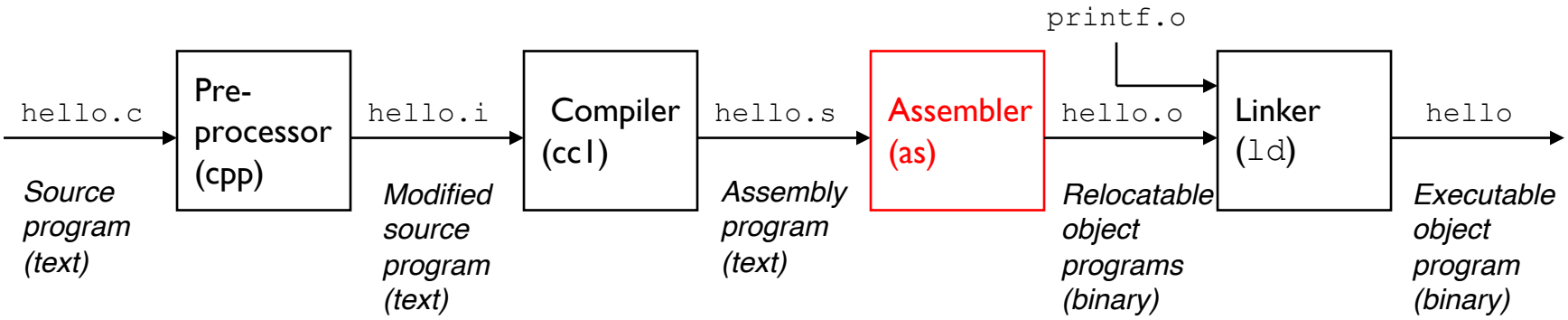
# Translating Program

- Compilation phase (`cc1`)
  - Translates the C to an assembly-language program
  - Assembly-language
    - Also in a standard text form
    - Each statement exactly describes one low-level machine-language instruction



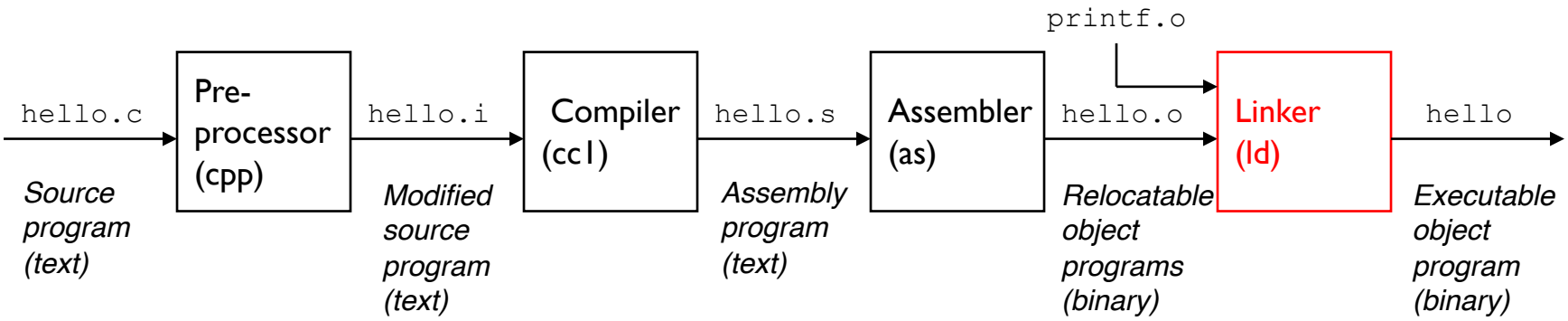
# Translating Program

- Assembly phase (`as`)
  - Translates `hello.s` into machine-language instructions
  - Package into *relocatable object program*



# Translating Program

- Linking phase
  - Where to find `prinf`?
    - `printf.o`
    - Provided by Standard C library
  - Merge `hello.o` and `prinf.o`
  - Result: `hello` (i.e., *executable object file*)



# Why we need to understand this

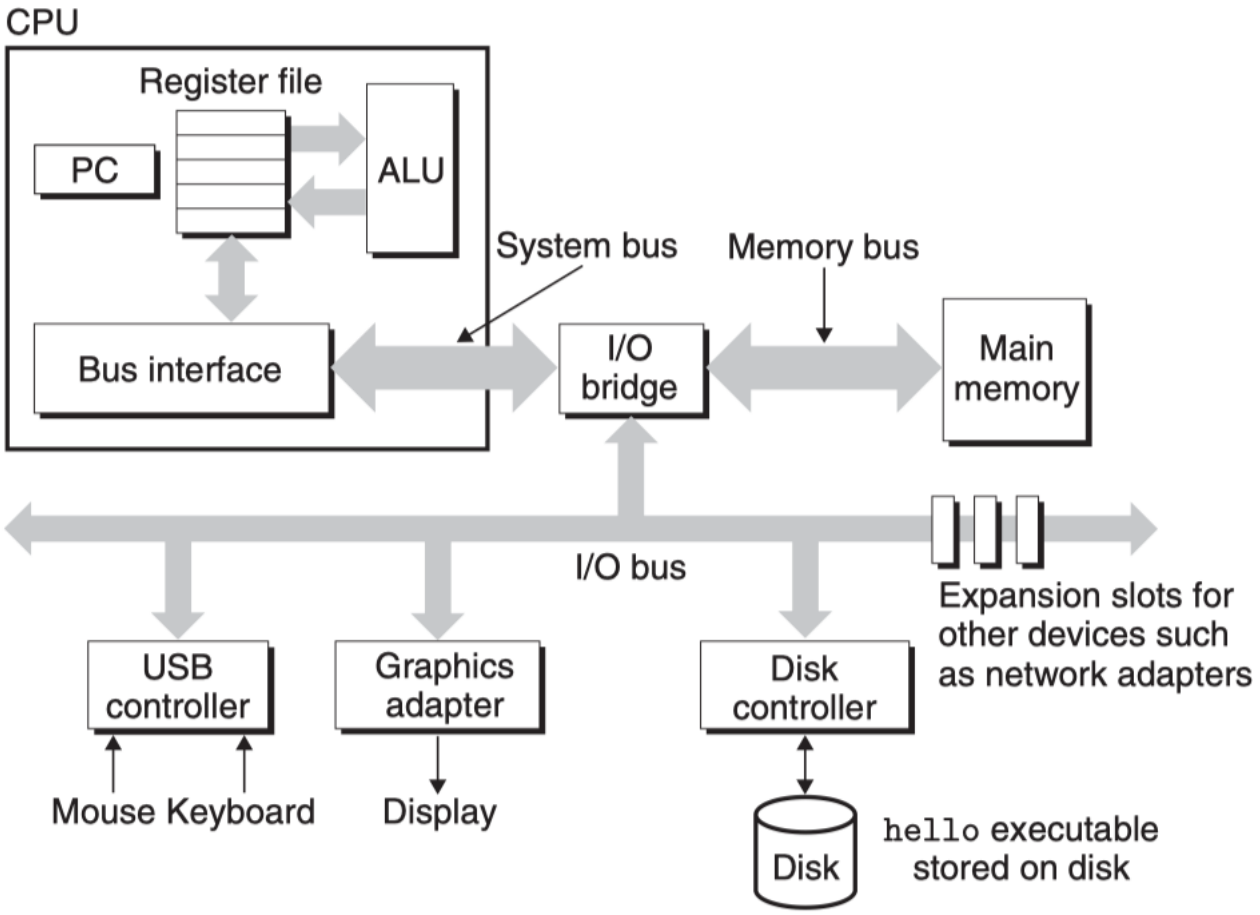
- Eliminating bugs
  - `#define min(x, y) x < y? x : y` [example01.c]
- Optimizing program performance
  - If-else vs. switch-case
  - `foo * 1024 → foo << 10`
- Understanding link-time errors
  - undefined reference to….
- Avoiding security holes
  - Buffer overflow

# Executing Program

```
unix> ./hello
hello, world
unix>
```

Shell loads and runs the program

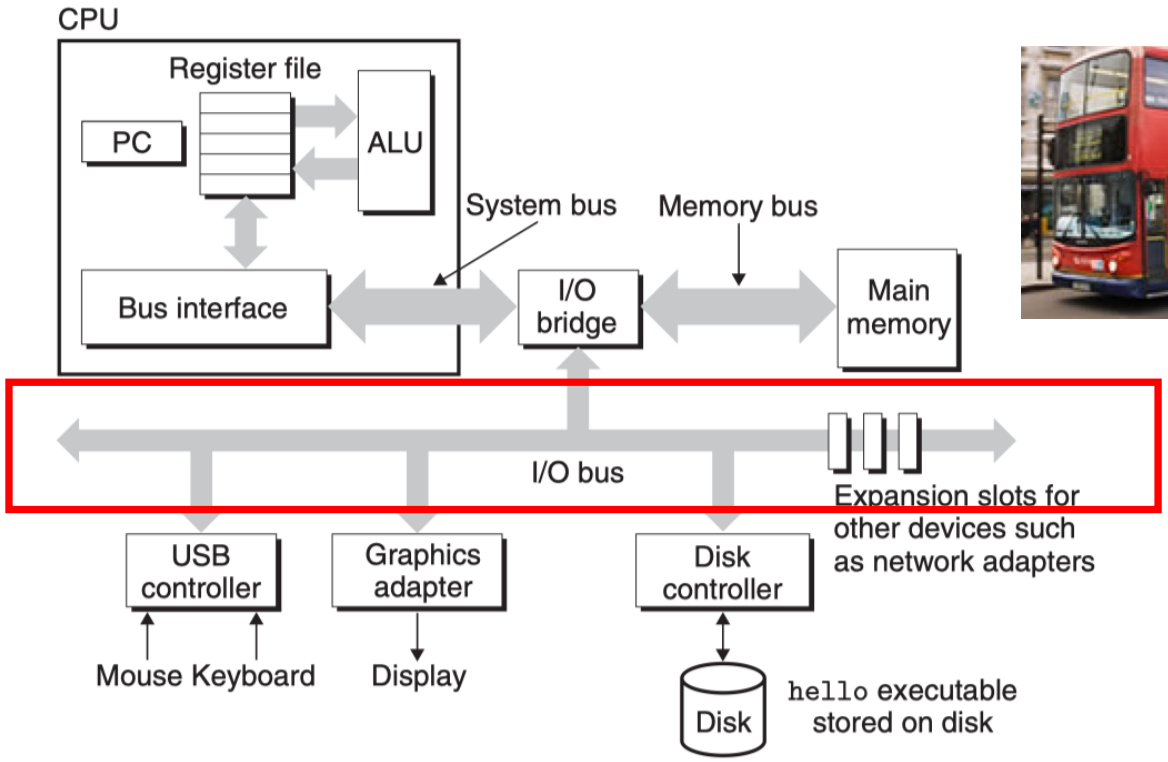
# Hardware Organization




Hardware organization of a typical system

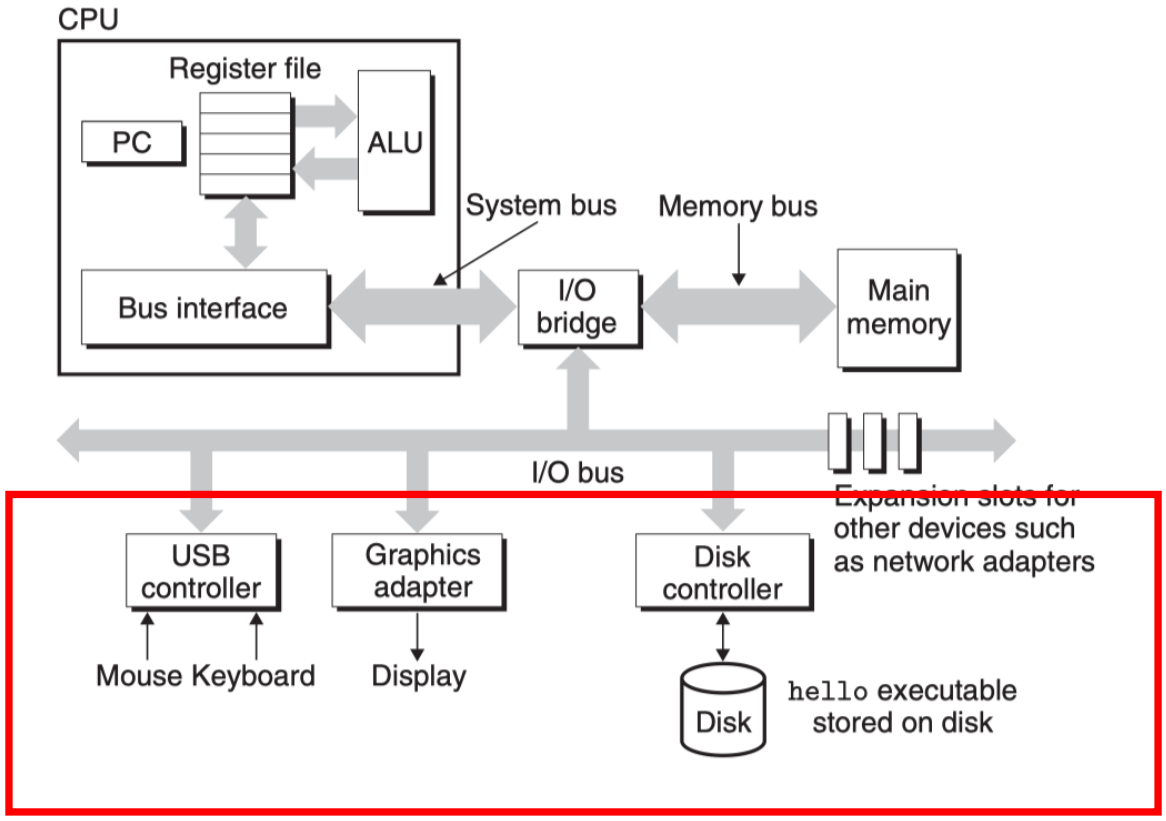
# Hardware Organization

- Buses



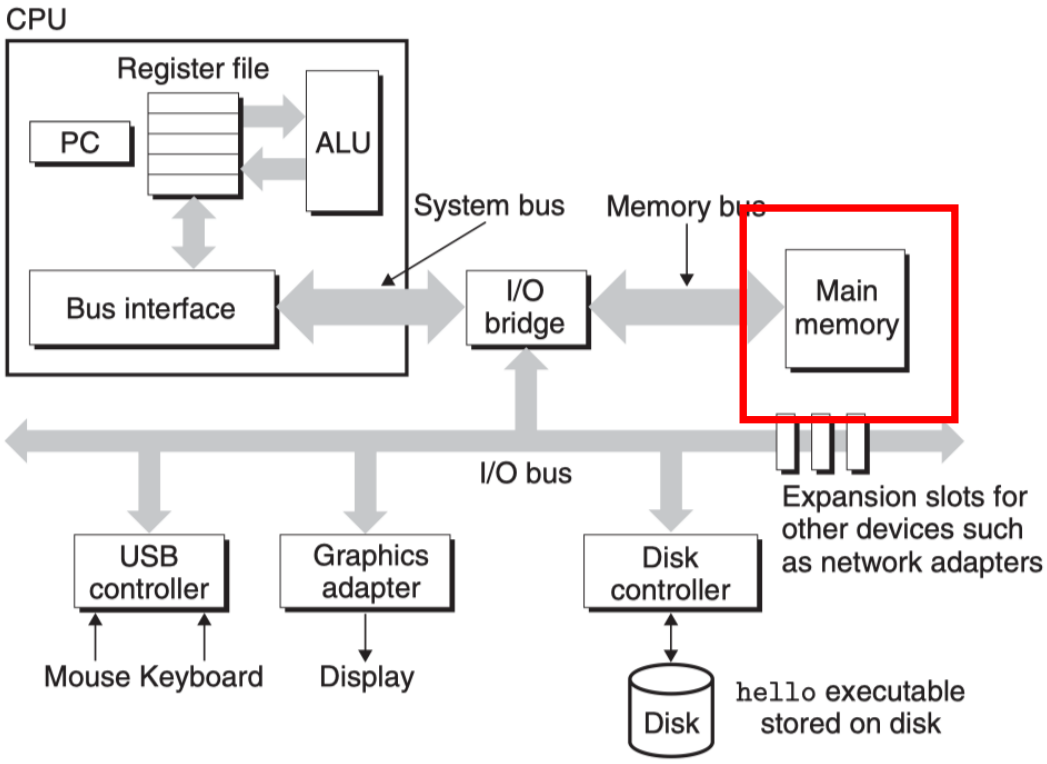

# Hardware Organization

- I/O Devices (Chapter 6&10)



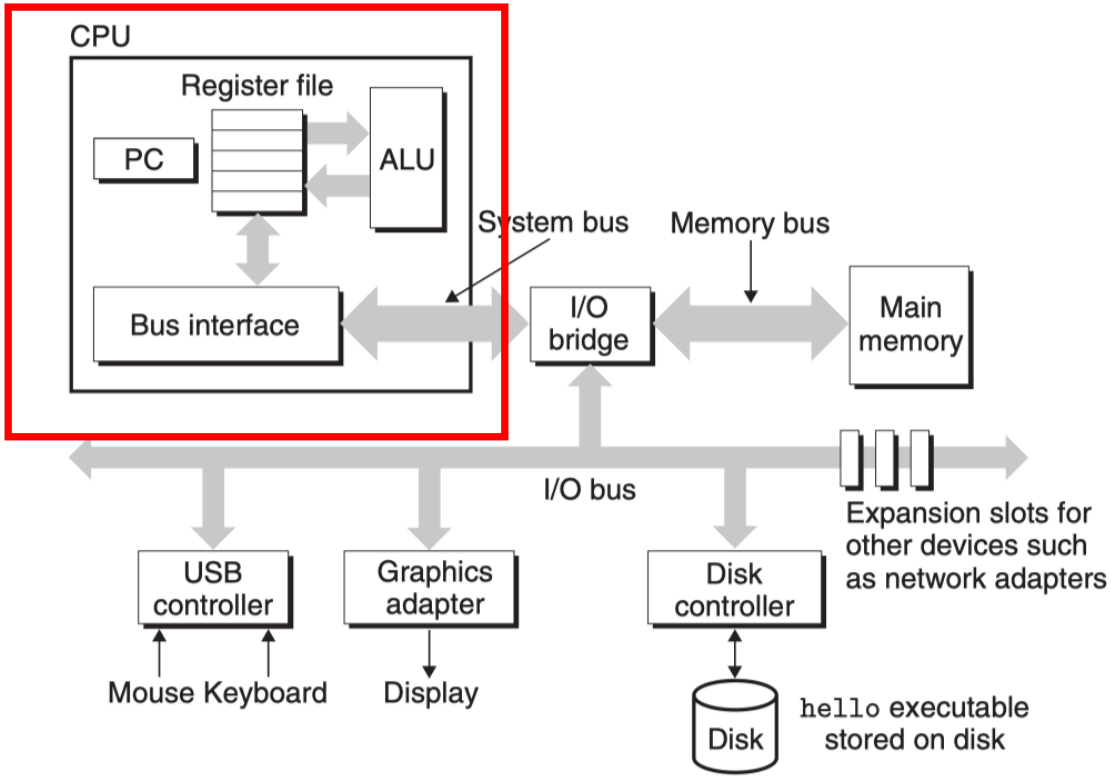
# Hardware Organization

- Main Memory (Chapter 6)
  - DRAM



# Hardware Organization

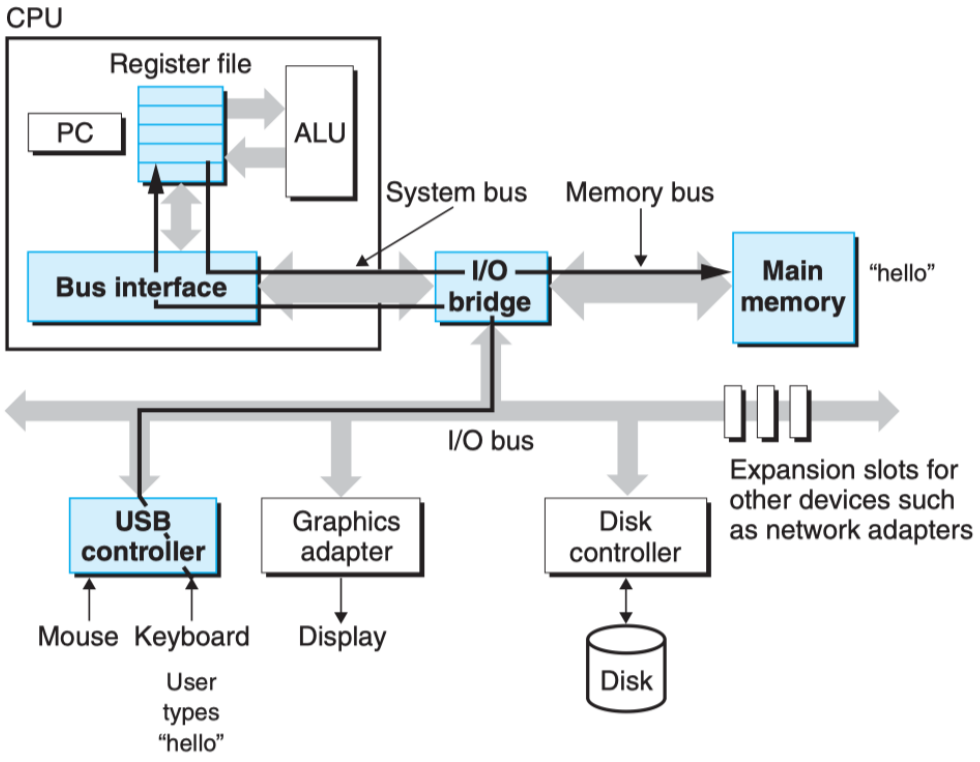
- Processor (Chapter 4)



# Executing Program

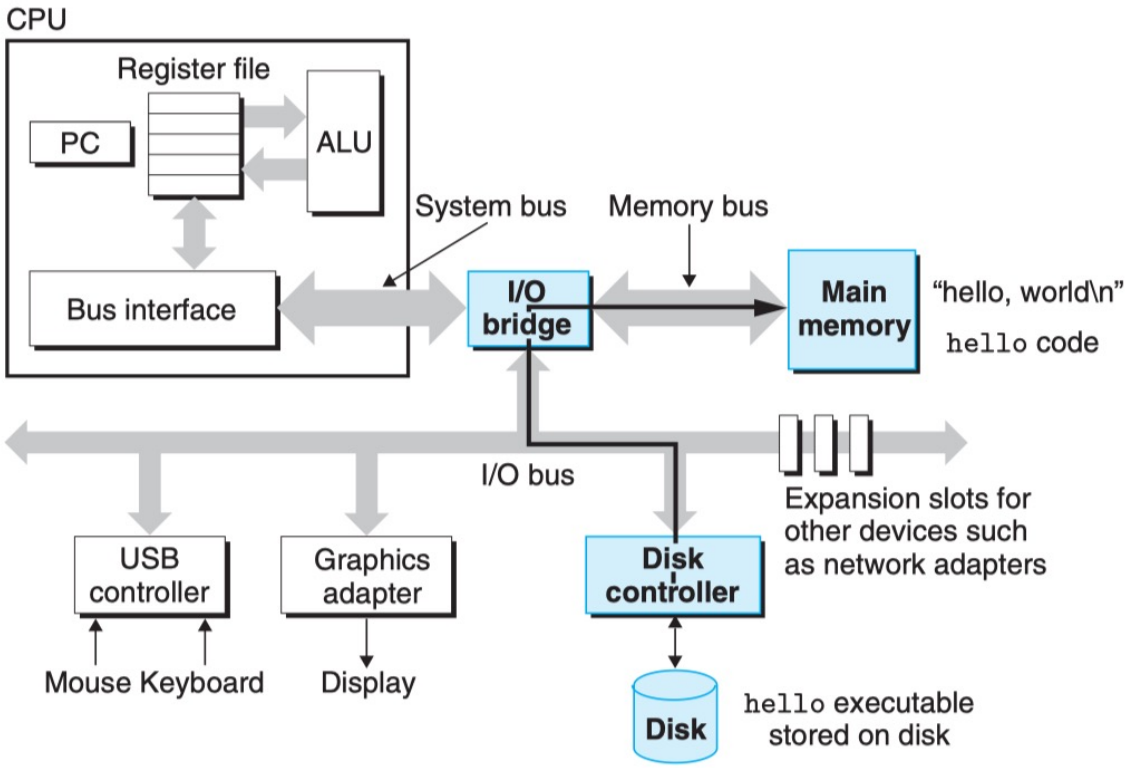
- Shell read "./hello" from keyboard into a register
- Store it into memory

```
unix> ./hello
hello, world
unix>
```



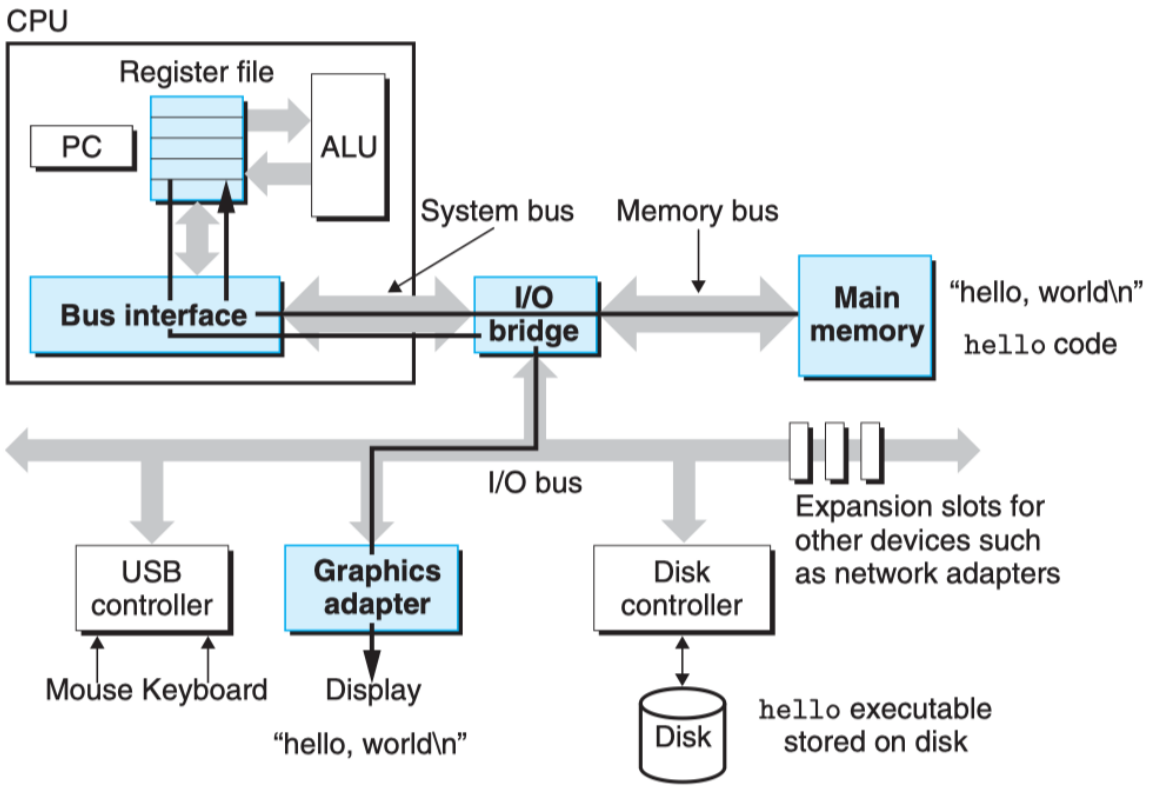
# Executing Program

- Load "hello" into main memory
  - Copies the code and data from disk to main memory
  - DMA



# Executing Program

- Execute the machine-language instructions
  - Copy "hello, world\n" from memory to the registers
  - Copy from registers to the display device

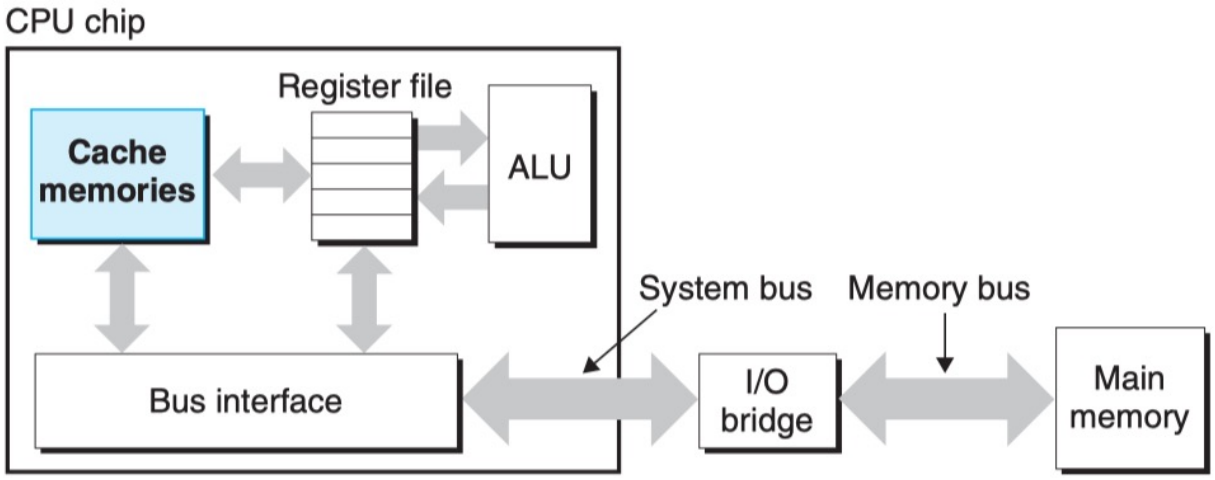


# Memory Architecture

- Spends a lot of time moving information from one place to another!
- Disk vs. Main memory
  - 1,000x larger
  - 10,000,000x slower
- Registers vs. Main memory
  - 100s bytes vs. 10s GB
  - 100x faster
- Laws
  - Larger: slower
  - Faster : more expensive

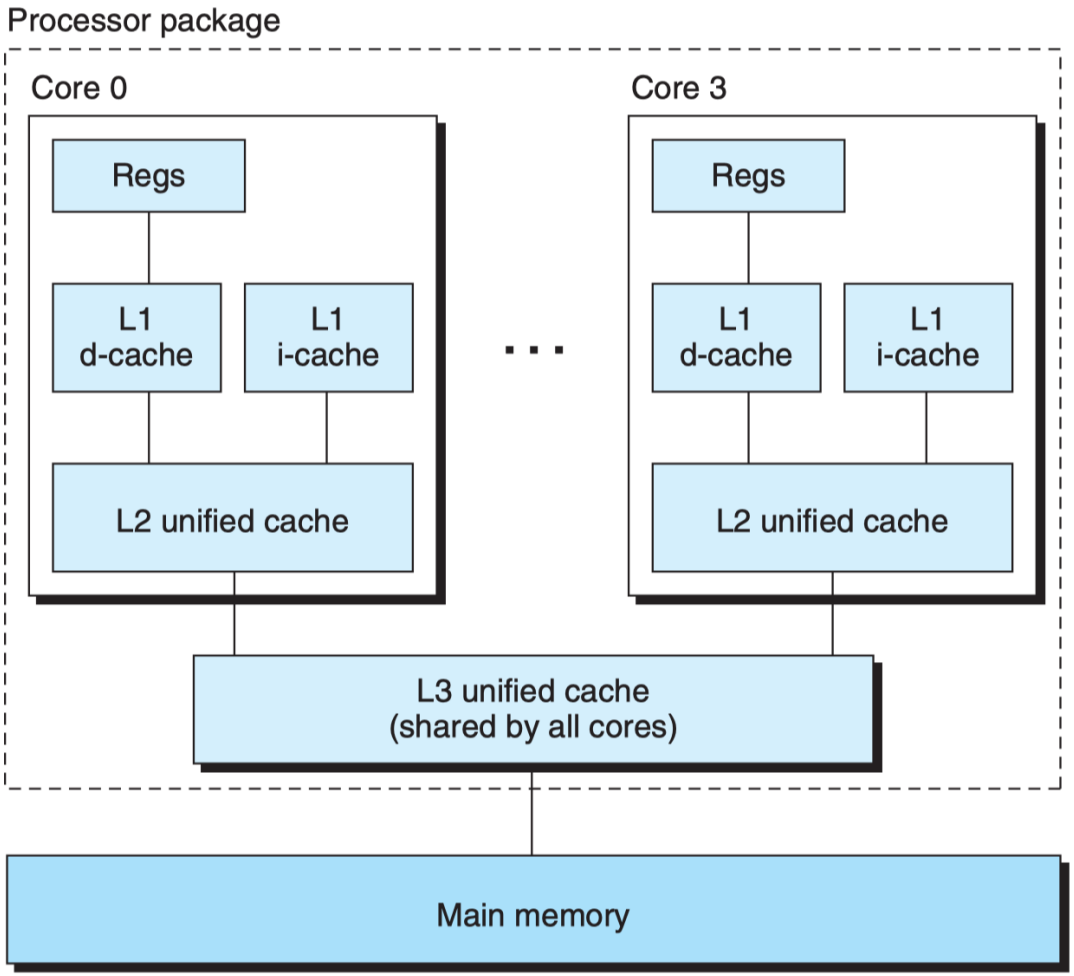
# Memory Architecture

- Cache
  - SRAM
  - 10s MB (Intel i7-11700, 16MB Cache)
  - 5x slower than registers
  - 5-10x faster than main memory



# Memory Architecture

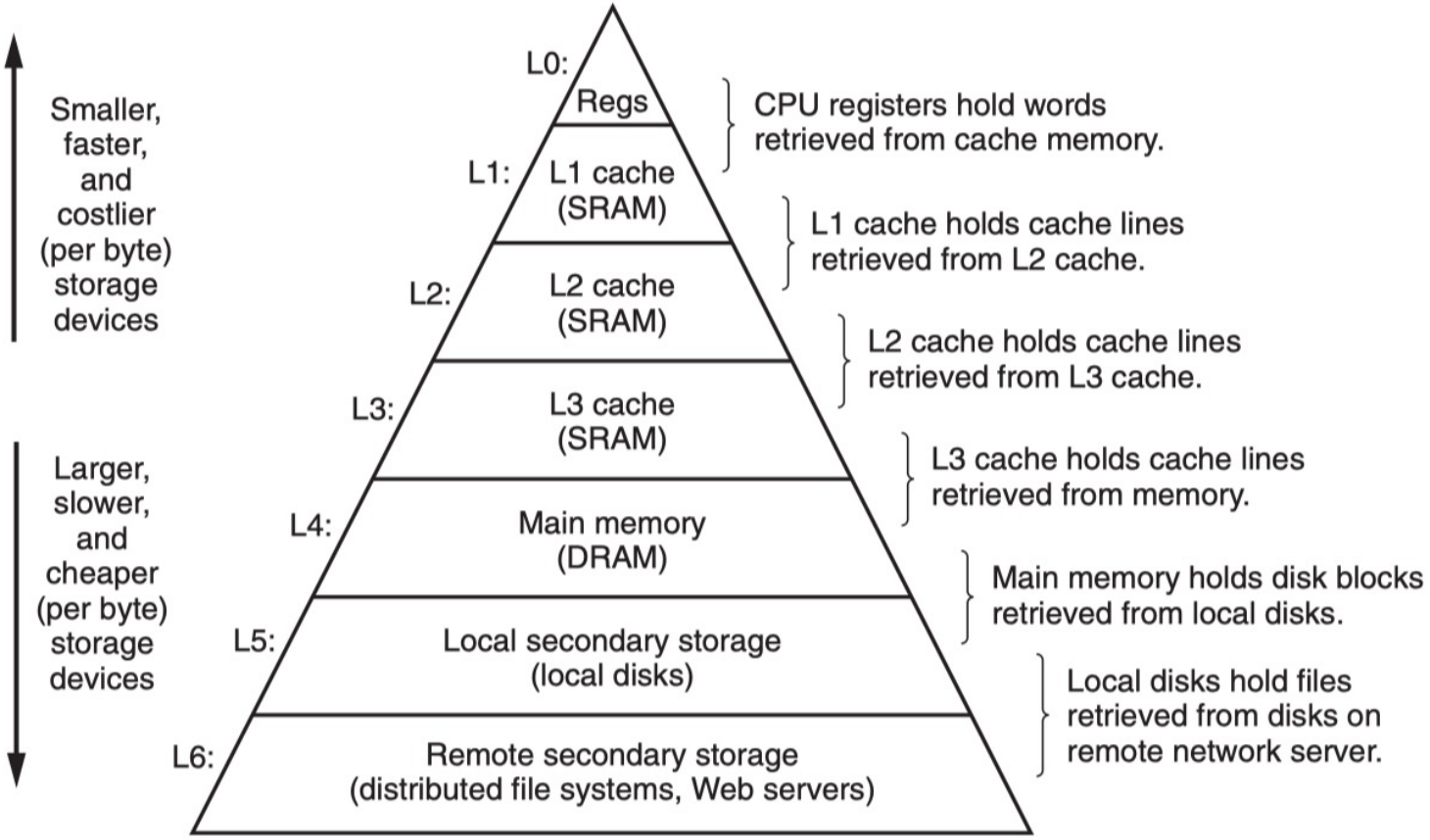
- Cache




Intel Core i7

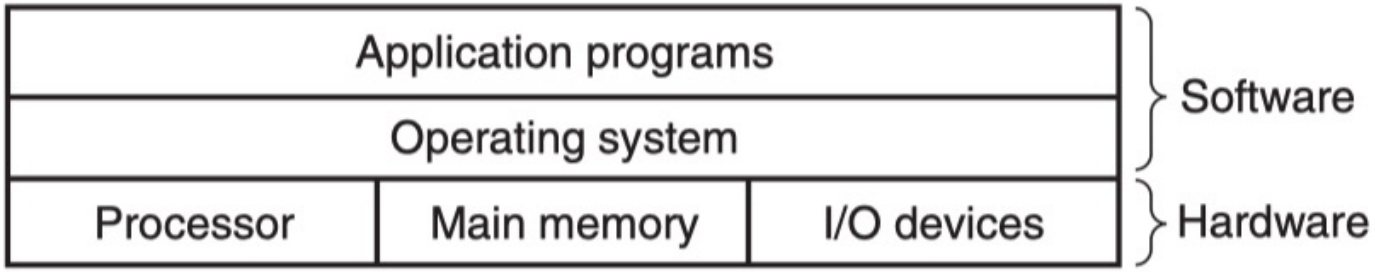
# Memory Architecture

- Memory hierarchy
  - Speed → Registers/Cache
  - Size → Disks



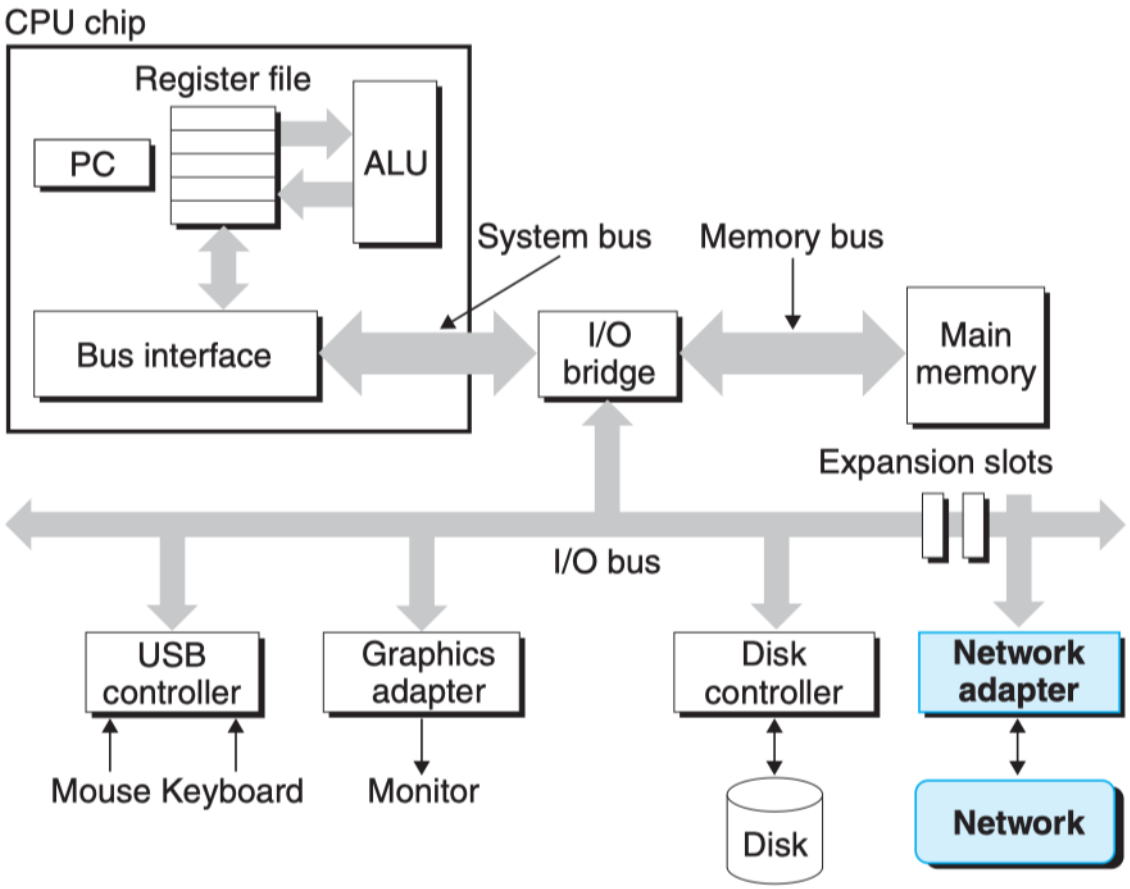
# Operating System

- Manages the hardware
  - Protect the hardware
    - Applications can be evil and vulnerable
  - Provide applications with simple and uniform mechanisms
    - Low-level hardware devices are quite different from each other



# Network

- Communication between Systems (Chapter 11)



# Summary

- Information = bits + context
- Programs are translated by compilers
  - From ASCII text to binary executable file
- Memory: store binary instructions
- Processor: execute binary instructions
- Memory is important
  - Computers spend most of their time copying data
  - Memory hierarchy
    - Speed: register/cache
    - Size: disk
- Operating System: managing hardware
- Network: communicate between hosts