# Overview

## Introduction to Computer Systems

2024. Spring

Xi'an Jiaotong University

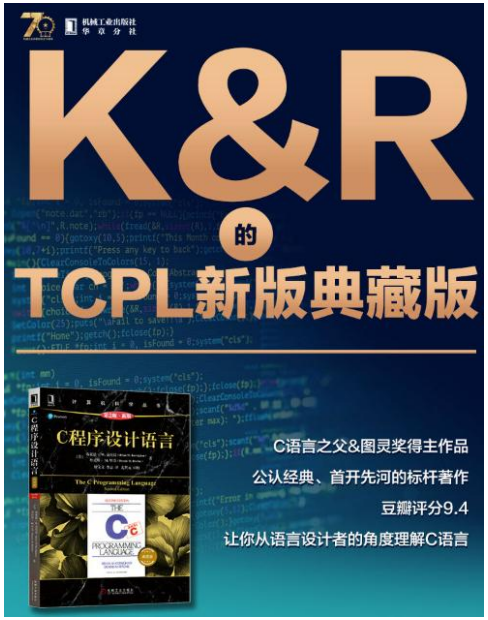
Danfeng Shan

# 教材与参考书

- 深入理解计算机系统（第三版）
- Computer Systems: A Programmer's Perspective (CSAPP 3e)


# 参考书

- C语言程序设计 第二版 （K&R）
- 操作系统导论（OSTEP）




# 教学规划——授课

| 内容 | 参考学时 |
| --- | --- |
| 信息的处理与表示 | 4 |
| 程序的机器级表示 | 14 |
| 存储器体系结构 | 4 |
| 程序优化 | 4 |
| 程序链接 | 4 |
| 异常控制流 | 6 |
| 虚拟存储器 | 10 |

# 教学规划——实践

| 内容 | 参考学时 |
| --- | --- |
| datalab | 2 |
| bomblab | 2 |
| attacklab | 2 |
| cachelab | 2 |
| optlab | 4 |
| loadderlab | 4 |

# 考核方法

- 平时成绩 10%
  - 考勤、课堂纪律、上课回答问题、课堂测验
- 项目实践 50%
  - Auto-Grading系统对代码自动打分
  - Anti-Cheating系统自动检测代码抄袭
  - 抄袭是高压线，一经核实双方项目实践分数为0
- 期末考试 40%
  - 以课堂和Lab内容为主

# 关于实验的几点补充

- 实验环境：Linux/GCC
  - 提供完整环境配置的服务器（ICSServer）
  - 提供预先写好的Makefile
  - 组织Bootcamp，带新人快速上手
- 实验发布：课程主页
  - https://ics.dfshan.net
  - 有详细的实验指导书
- 实验提交和分数公布：在线学习平台
  - http://class.xjtu.edu.cn/course/58865
  - 允许迟交，但会根据延后时间扣分

# 关于实验的几点补充

- 独立完成实验
- 做好时间管理
  - ✗ 跟不上节奏、在截止日期前疯狂弥补☹
  - ✓ 紧跟节奏
  - 预期：每个lab需要2周的时间全力以赴完成
- 实验所需预备知识

| 必要 | 有益 |
| --- | --- |
| • C编程<br>• Linux命令行<br>• ssh<br>• gcc/gdb | • vim<br>• Make/Cmake<br>• Git<br>• Google |

# 严禁作弊

- 什么是作弊
  - 从别人（同学/学长）复制/抄袭/借用代码
  - 从网上找代码（GitHub、博客）
  - 使用AI（ChatGPT）生成代码
  - 合作完成Lab
- 什么不是作弊
  - 帮助别人使用系统/工具
  - 请别人翻译一下题目的表面含义
  - 翻看通用手册/教材
- 作弊带来的后果比什么也不做更严重
- 作弊检测：MOSS


# 答疑

- 上策：XJTU Men（https://xjtu.men/c/academic/ics/58）
  - 匿名性好，可大胆提问
  - 知识共享：你遇到的问题往往别人也遇到过
- 中策：群里提问
- 下策：私下找助教/老师
- 下下策：憋着不说

- 《提问的智慧》

- 身体/心理健康是最最最重要的

# Course Overview

What you have known

Write a program

How programs are executed?

Details of each component

What you are about to learn

# Overview

- Representing Program (Chapter 2)
- Translating Program (Chapter 3&4)
- Executing Program (Chapter 7&8&9)
  - Hardware Organization
- Memory Architecture  (Chapter 6)
- 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 |

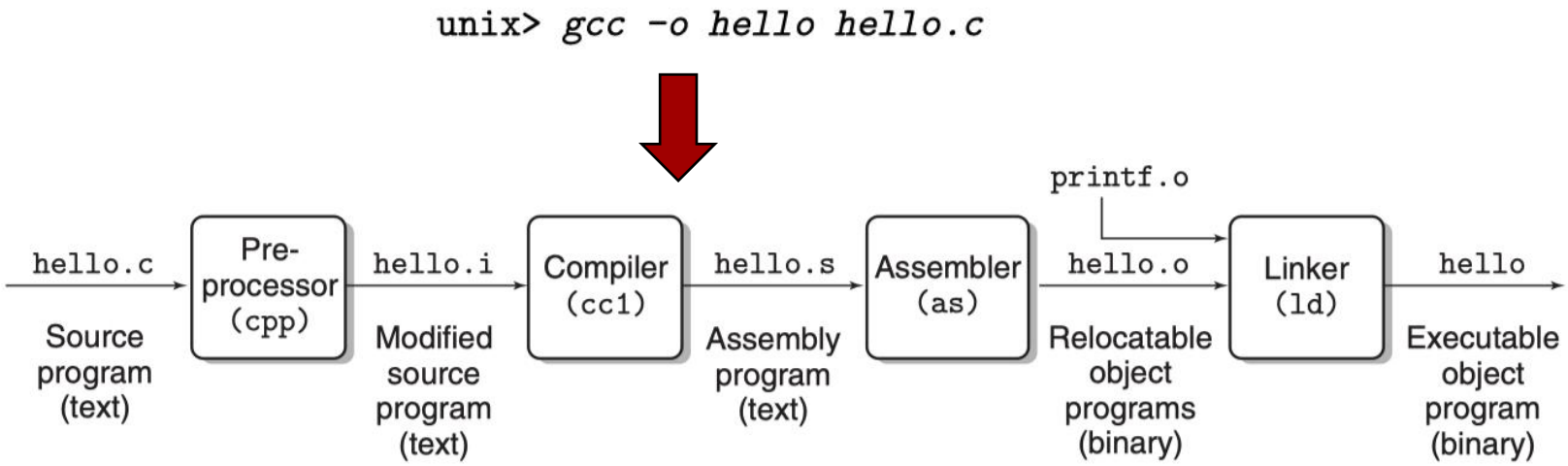
What about Chinese Character?

# 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

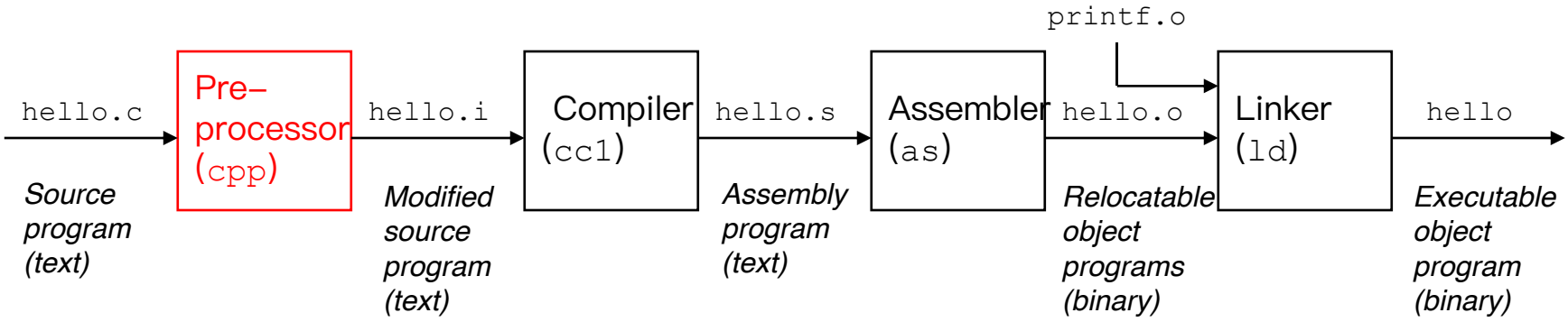
# Translating Program

- C program is a high-level language
  - Why: 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



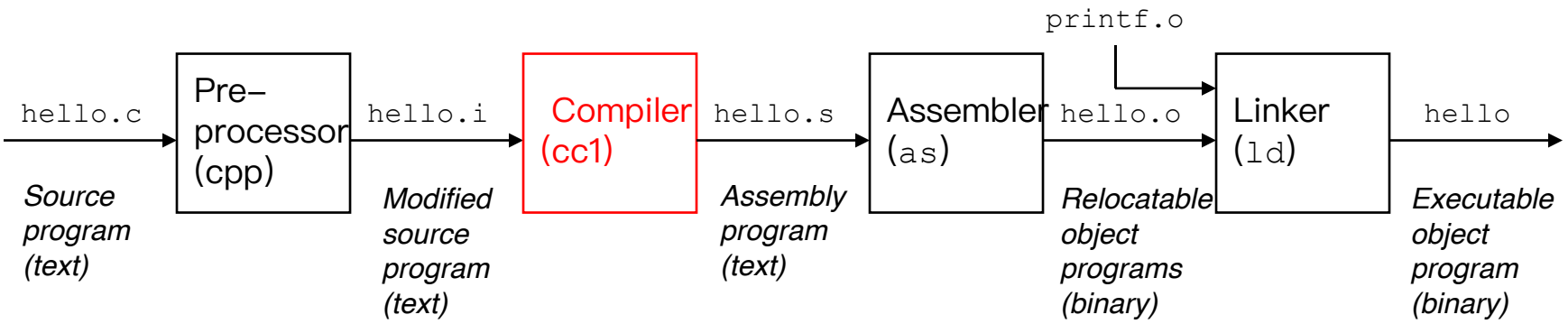

code/intro/hello.c

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

code/intro/hello.c

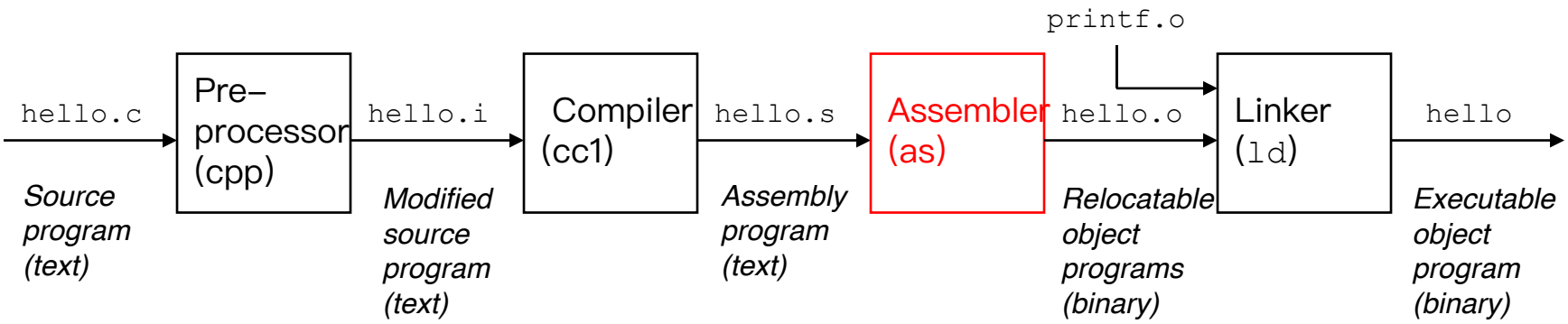
# Translating Program

- 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



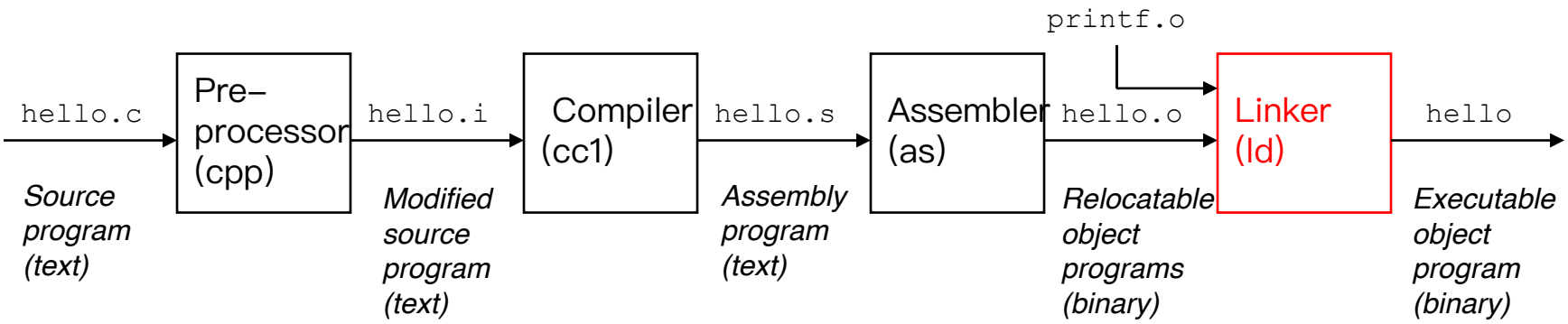
# Translating Program

- 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

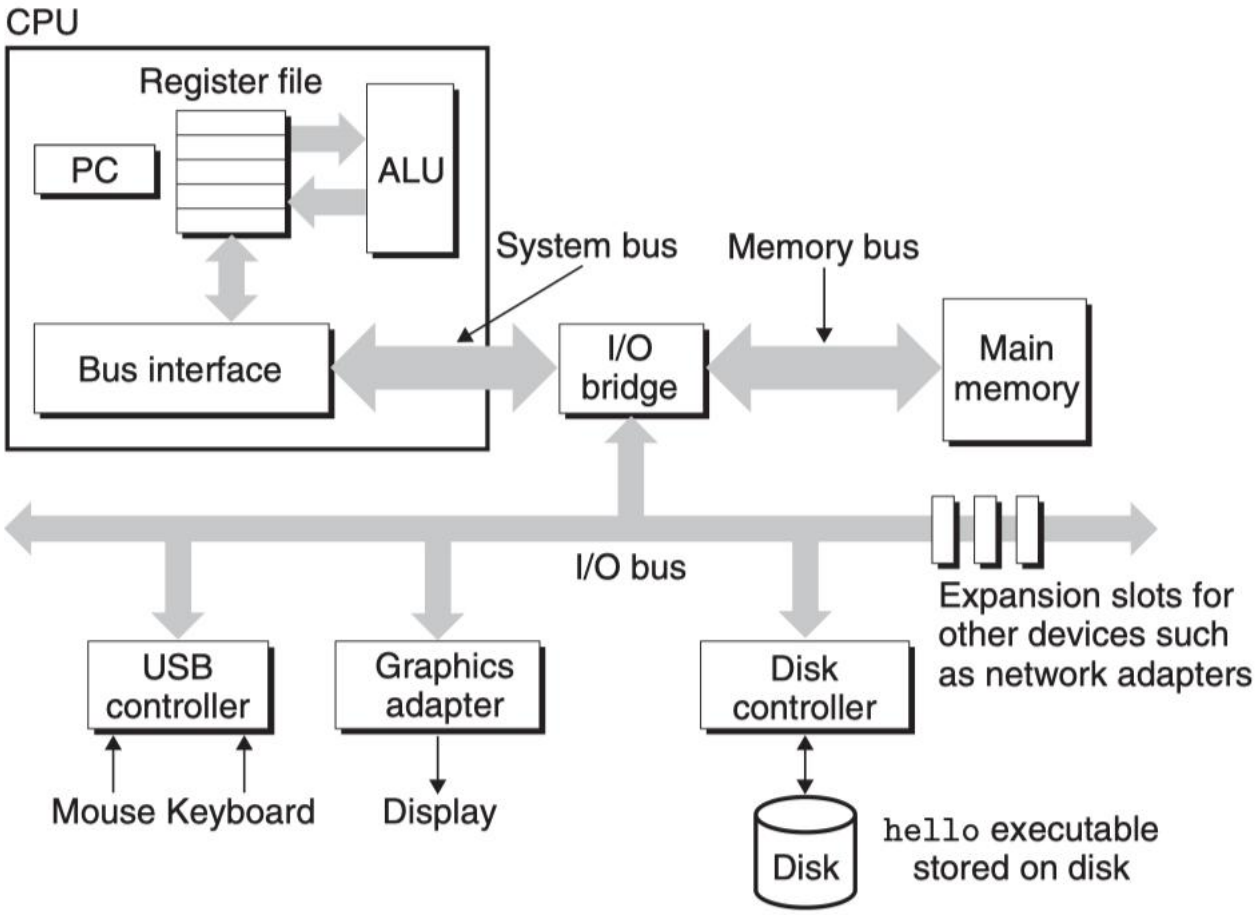
- Eliminating bugs
  - `#define min(x, y) x < y ? x : y` [example01.c]
- Optimizing program performance
  - If-else vs. switch-case
  - `foo * 1024 → foo << 10`
- Understanding link-time errors
  - undefined reference to….
- Avoiding security holes
  - Buffer overflow

# Executing Program

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

Shell loads and runs the program

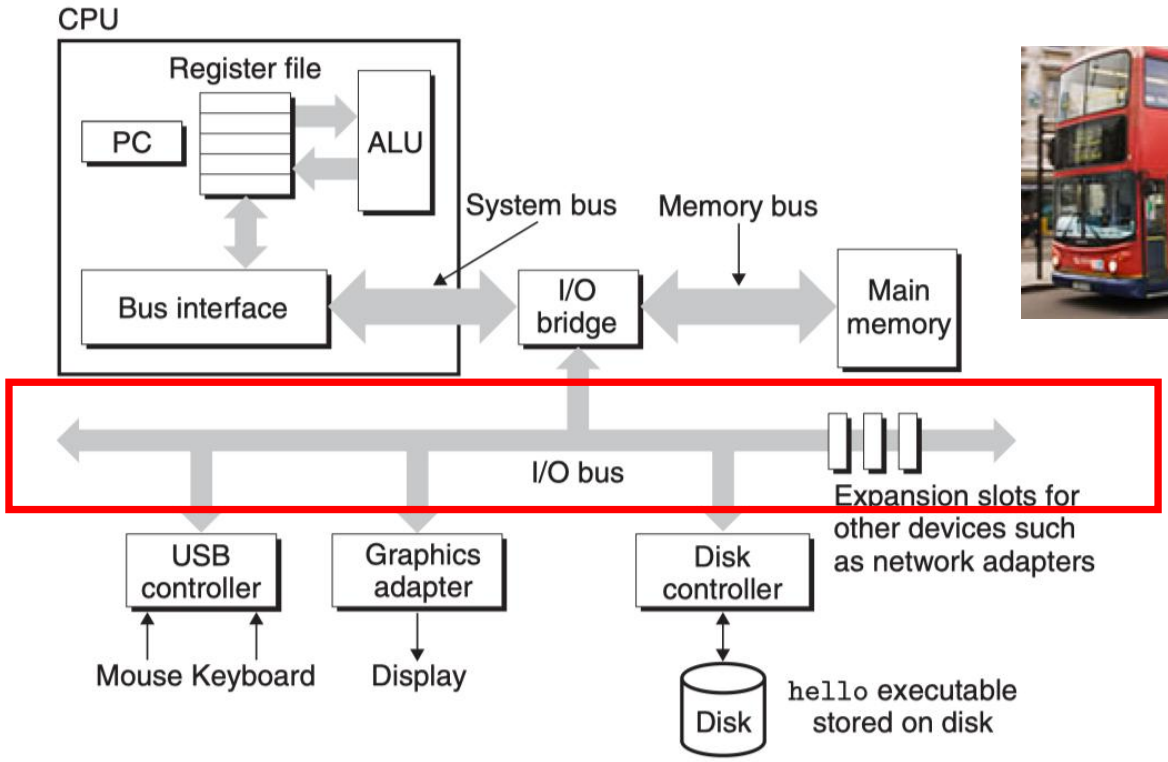
# Hardware Organization




Hardware organization of a typical system

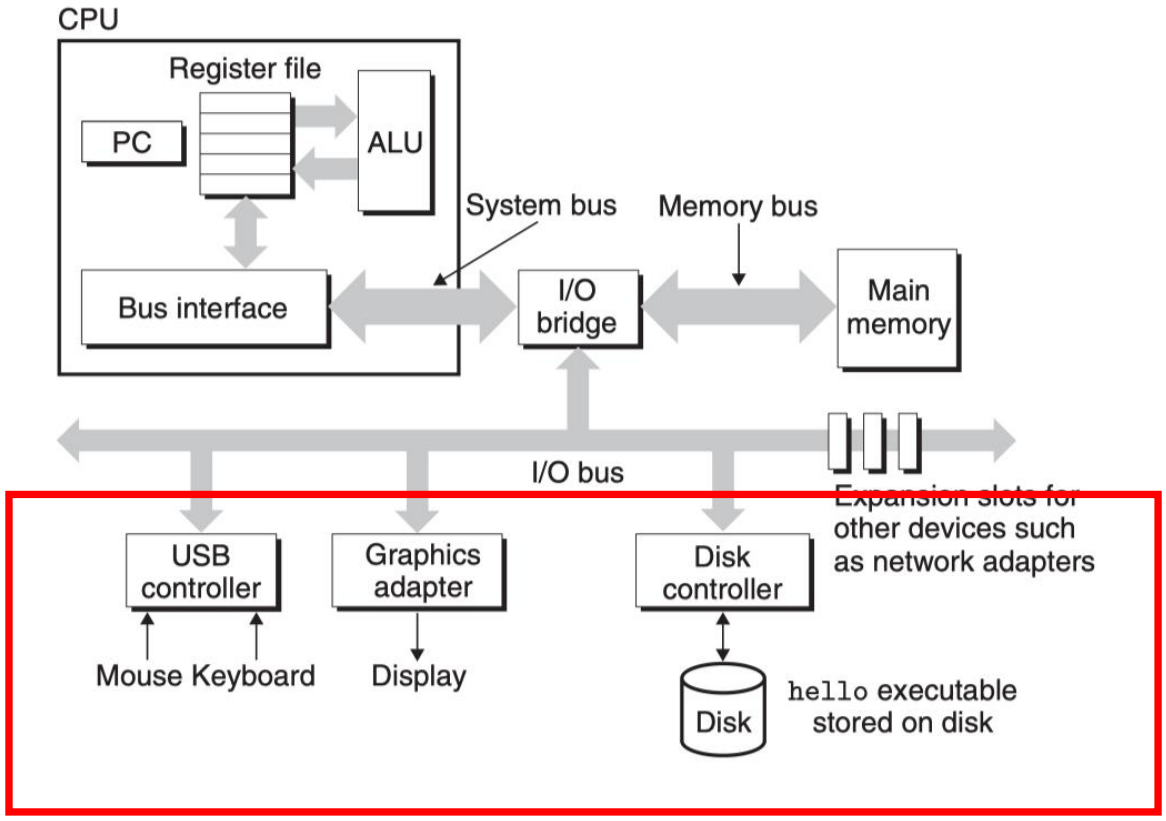
# Hardware Organization

- Buses



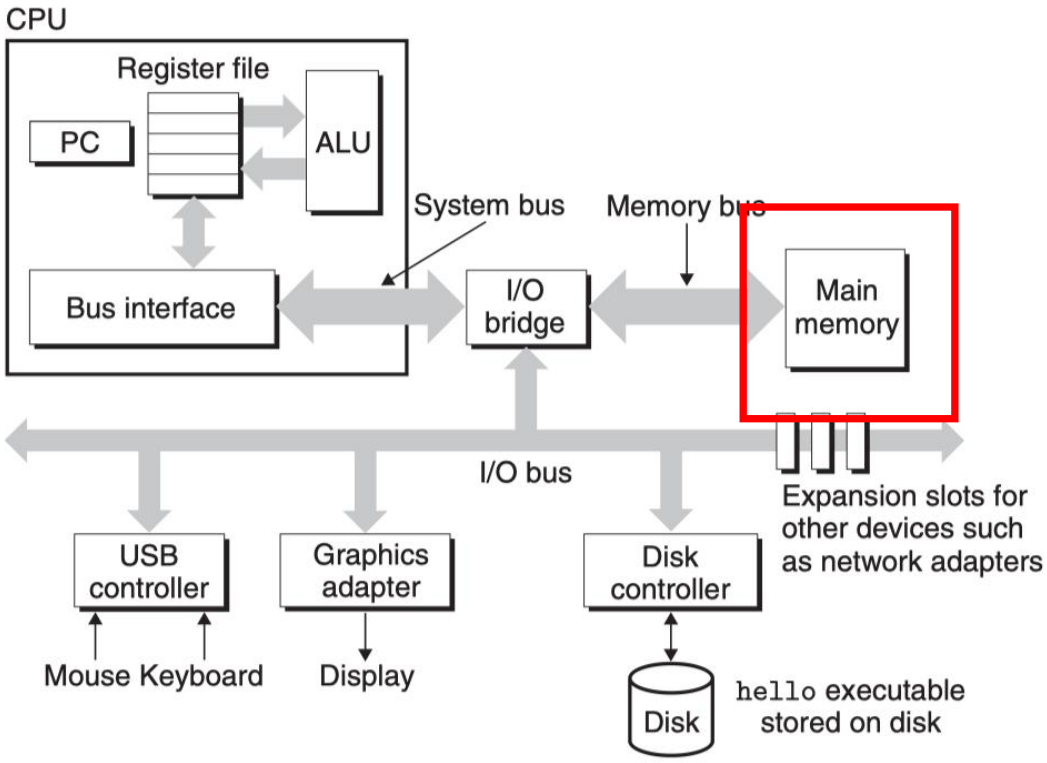

# Hardware Organization

- I/O Devices (Chapter 6&10)



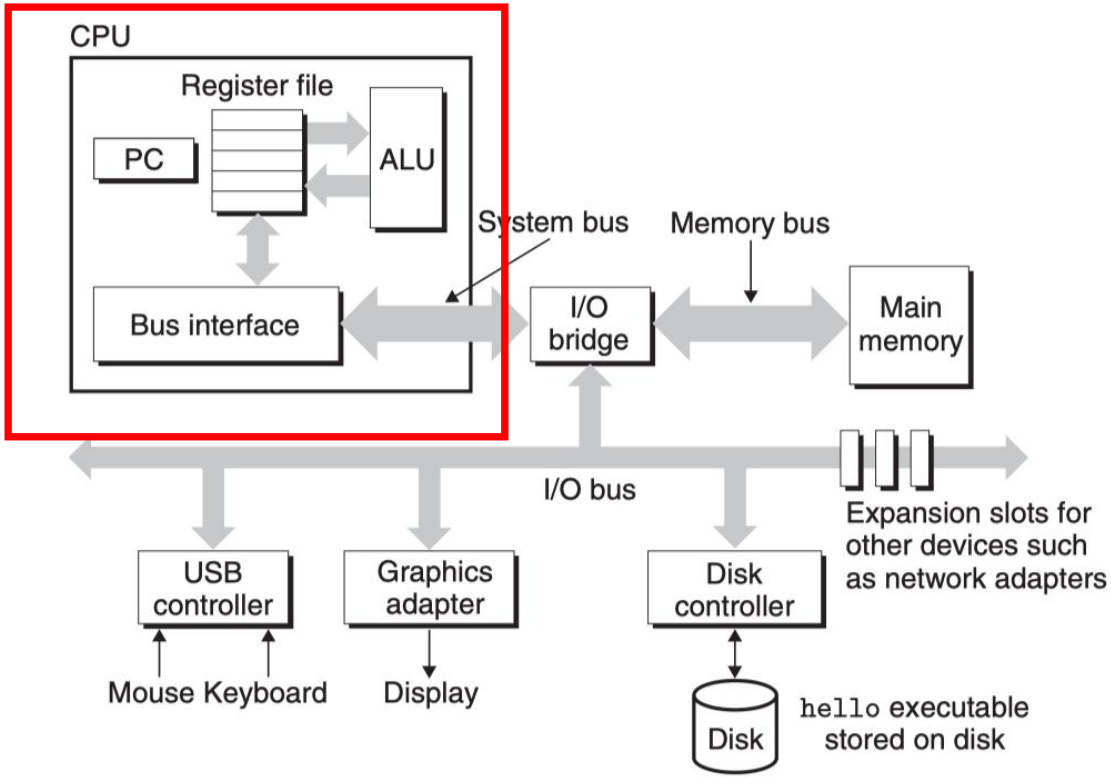
# Hardware Organization

- Main Memory (Chapter 6)
  - Temporary storage



# Hardware Organization

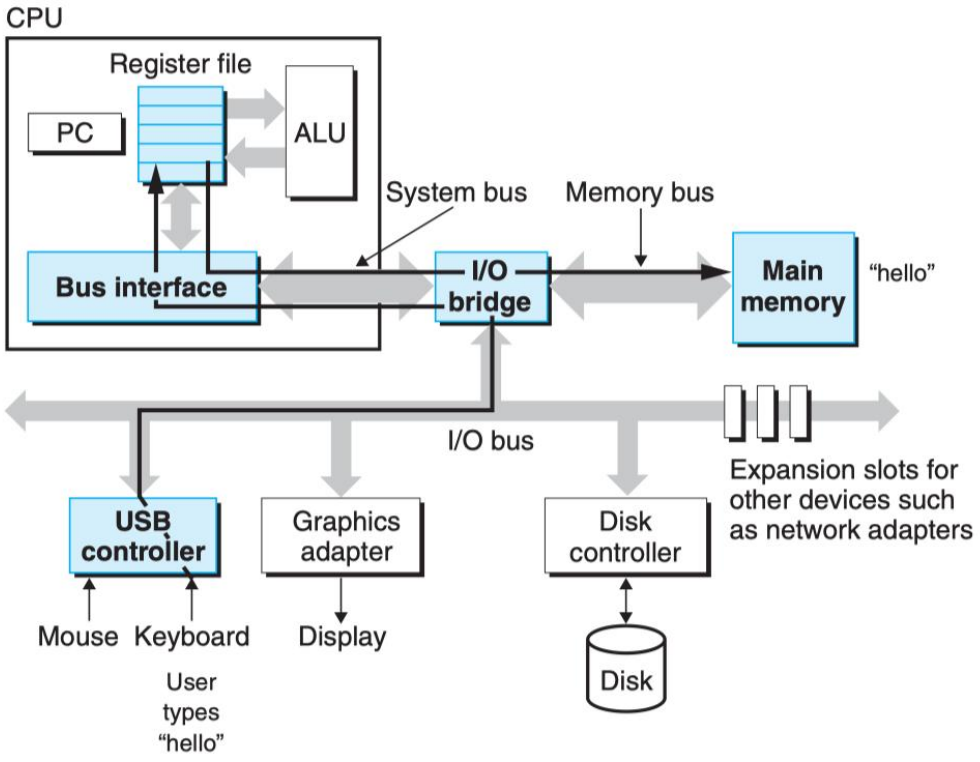
- Processor (Chapter 4)



# Executing Program

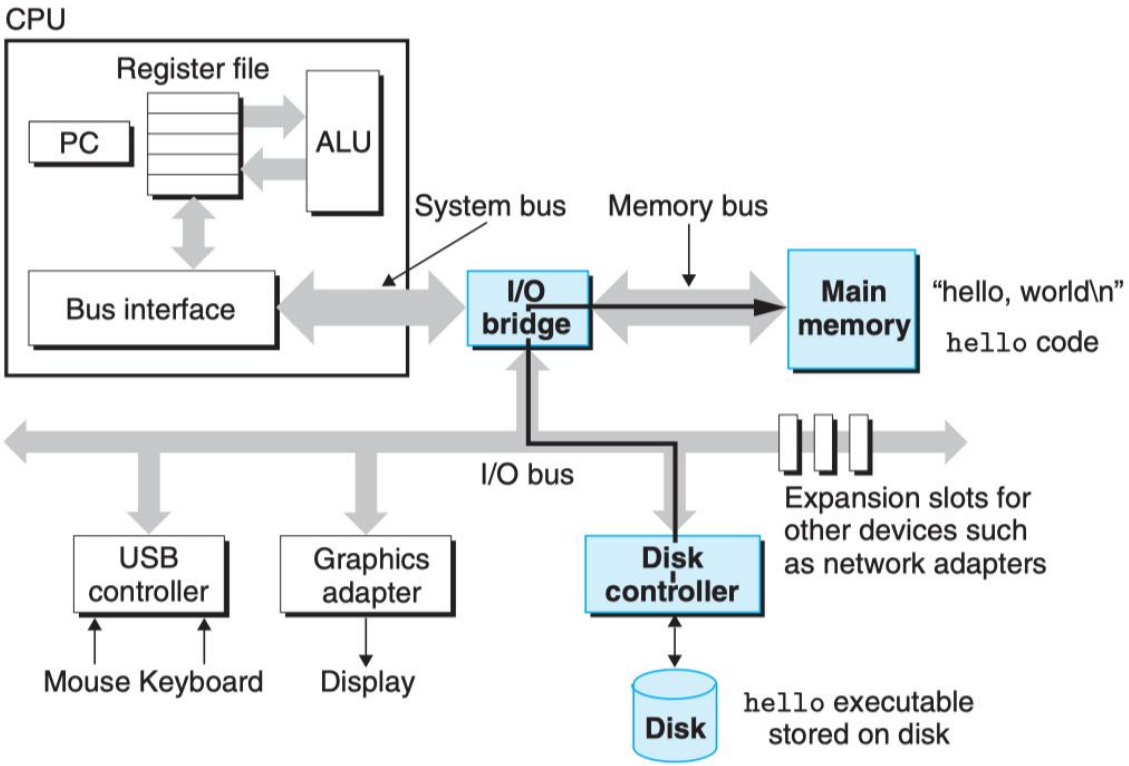
- Shell read "./hello" from keyboard into a register
- Store it into memory

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



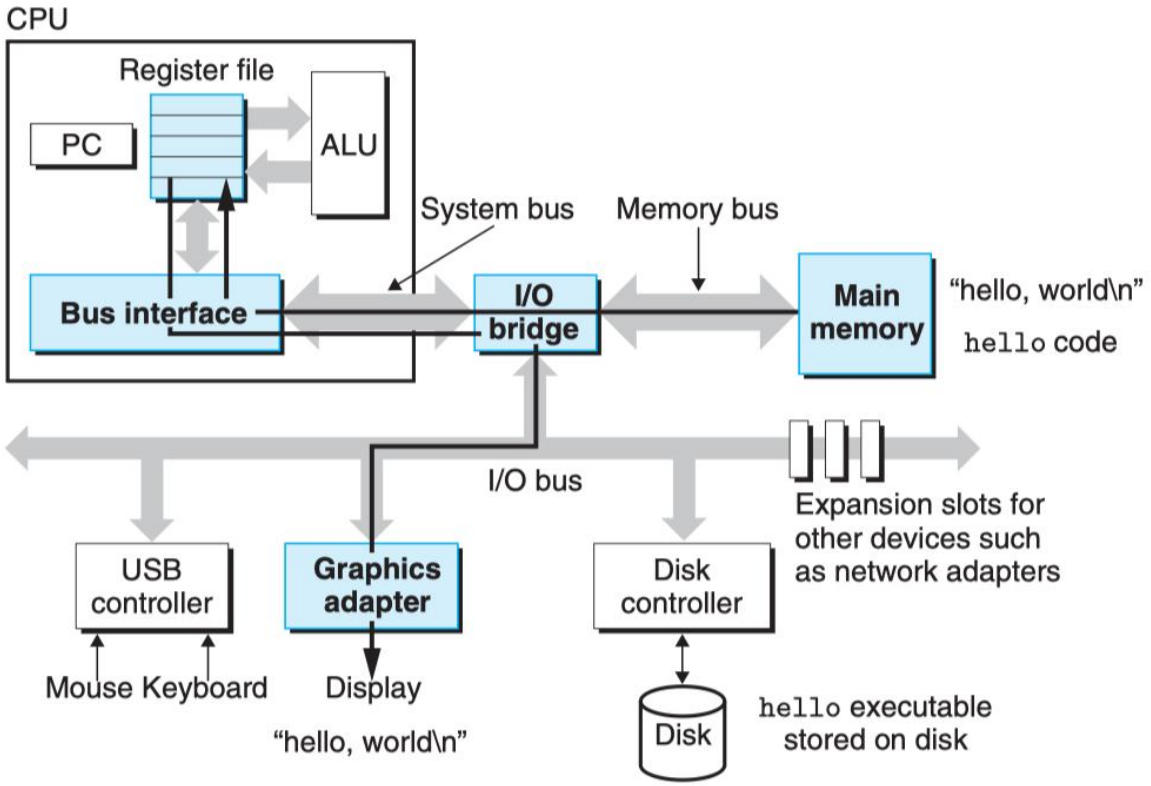
# Executing Program

- 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

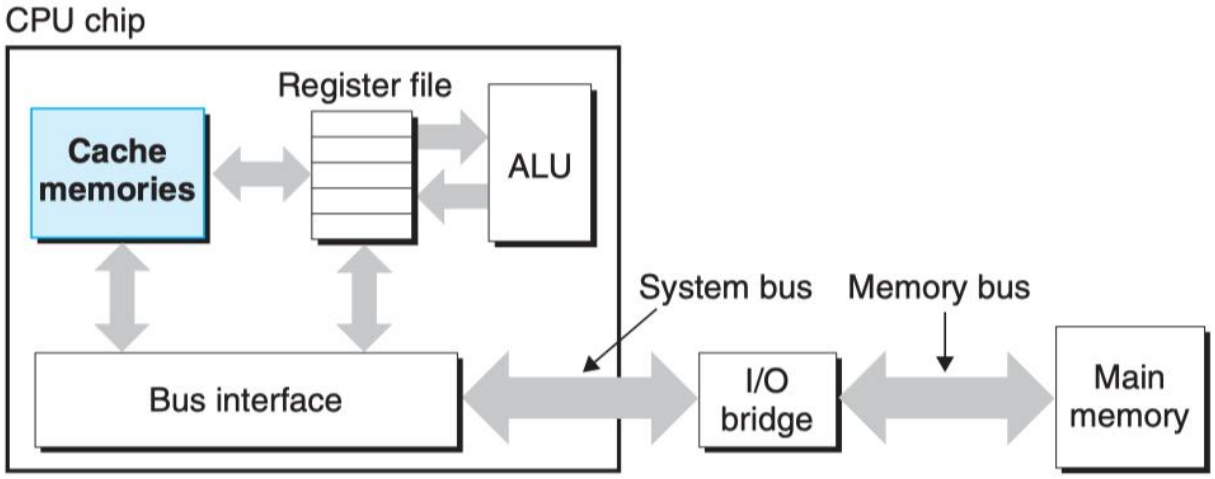


# Memory Architecture

- 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

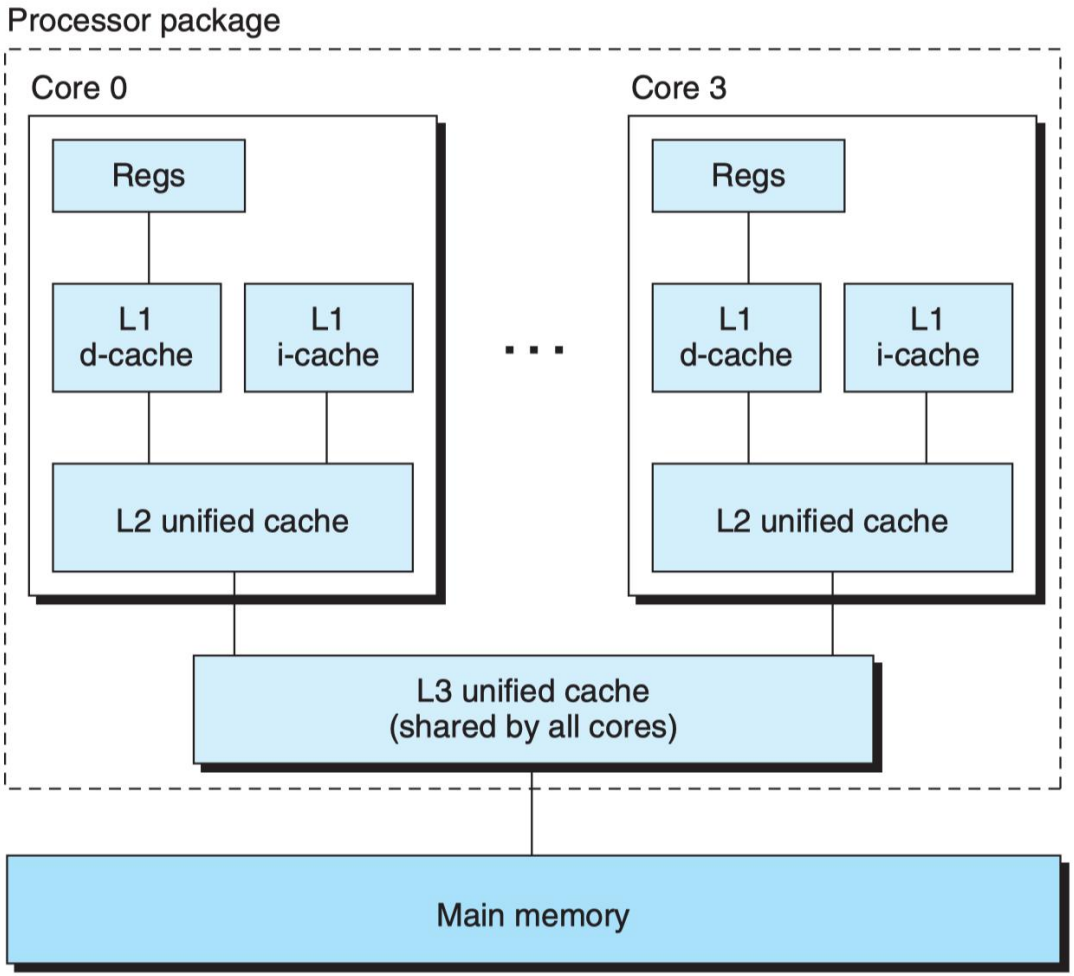
# Memory Architecture

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



# Memory Architecture

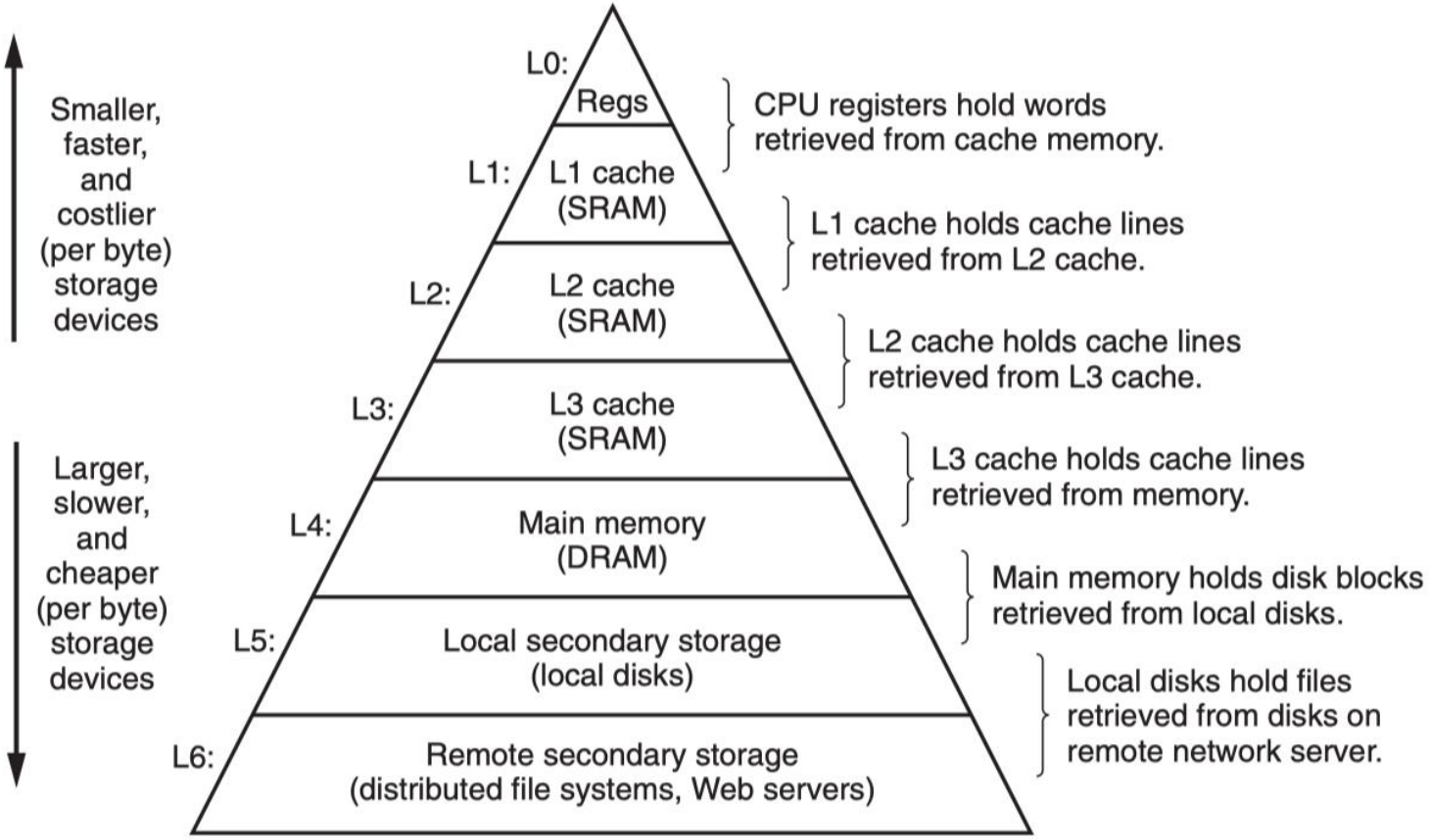
- Cache




Intel Core i7

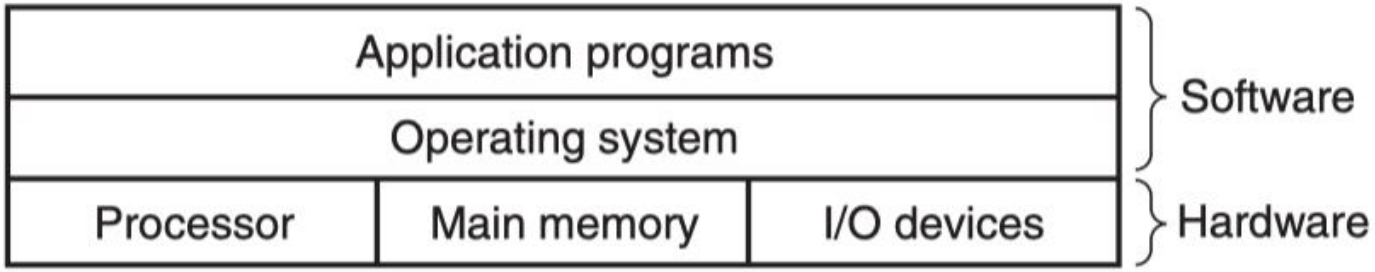
# Memory Architecture

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



# Operating System

- A layer of software between application and 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



# 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