Overview

Introduction to Computer Systems

https://xjtu-ics.github.io/ or http://ics.xjtu-ants.net/

课程主页

2025. Spring

Xi’an Jiaotong University

Danfeng Shan

QQ群

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	16
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
网络编程	4
并行编程	4
处理器体系结构	8

教学规划——实践

内容	参考学时
datalab	2
bomblab	2
attacklab	2
cachelab	4
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://xjtu-ics.github.io/ 或 http://ics.xjtu-ants.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/88273
  - 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 • Linux命令行 • ssh • gcc/gdb	• vim • Make/Cmake • Git • Google

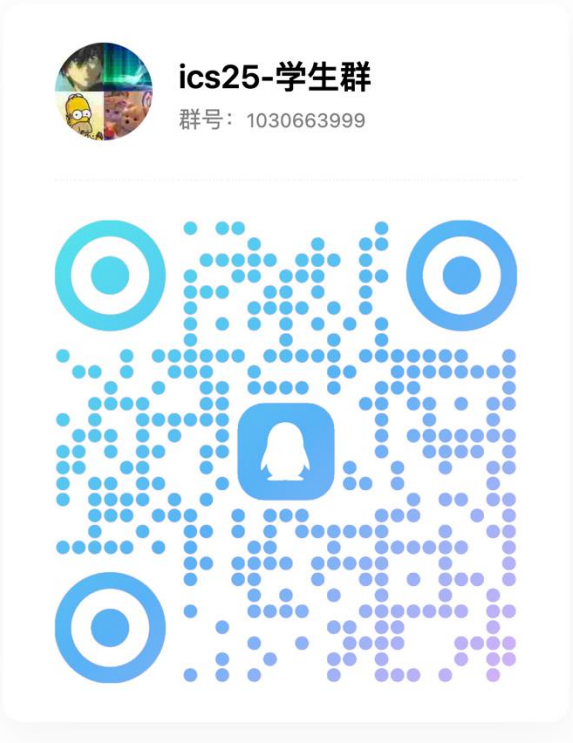
严禁作弊

- 什么是作弊
  - 参考/借鉴/复制别人（同学/学长）代码
  - 参考/借鉴/复制网上代码（GitHub、博客）
  - 使用AI工具（ChatGPT/Copilot）生成/美化代码
  - 合作完成Lab
- 什么不是作弊
  - 帮助别人使用工具
  - 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

答疑

- Piazza：一款专业的国际性课程答疑论坛
  - 注册链接：https://piazza.com/stu.xjtu.edu.cn/spring2025/xjtuics
- QQ群：不保证每个问题都能被解答
- 一对一：Office Hour
- 《提问的智慧》

- 遇到问题不要不解决！



这门课
不一样！

- 来自学生的评价
  - 非常好的课程，使我有一种不是在西交读cs的感觉
  - lab 才是这门课的精华
  - 虽然对我来说实验有一定的难度，但做起来真的很爽，希望课程越来越好
  - 实验不要抄袭，真给0分

- 来自学生的误解
  - 简单的导论课程
  - 可以刷分课程

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

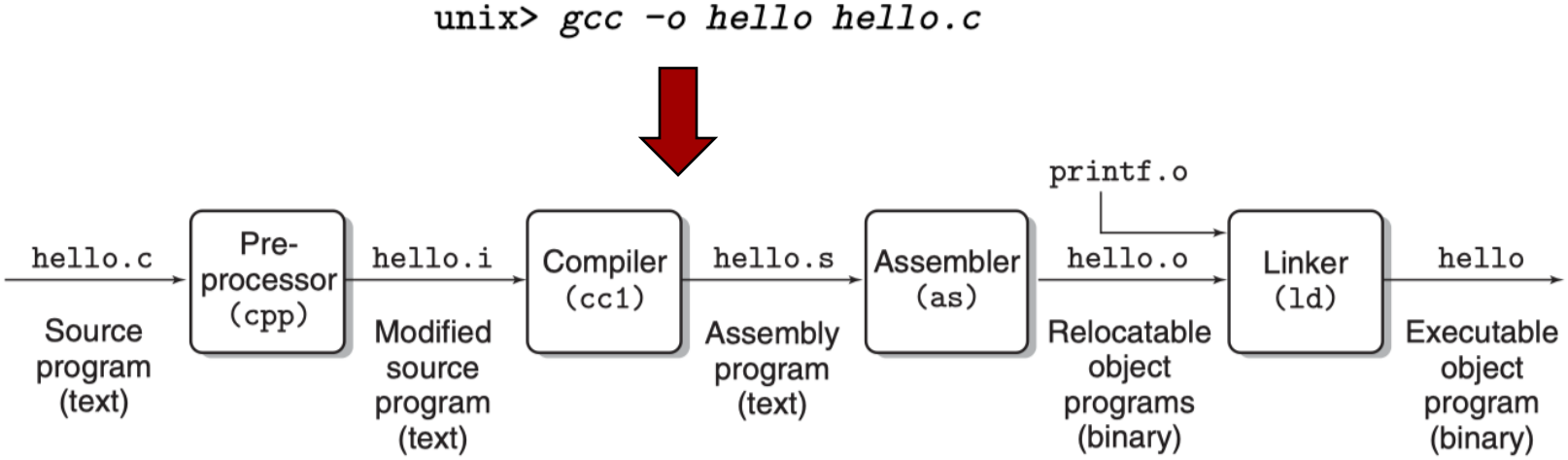
What about Chinese Character?

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

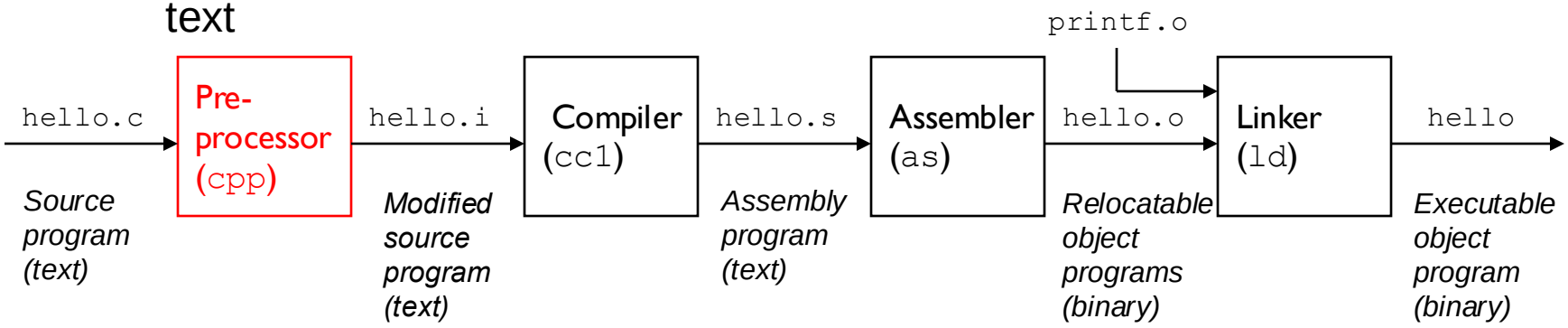
Translating Program

- 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



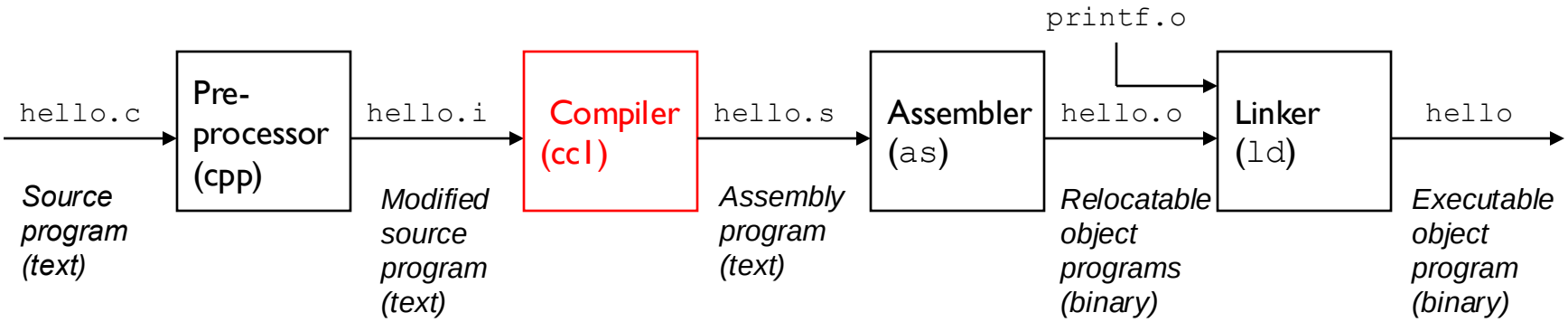
code/intro/hello.c

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

code/intro/hello.c

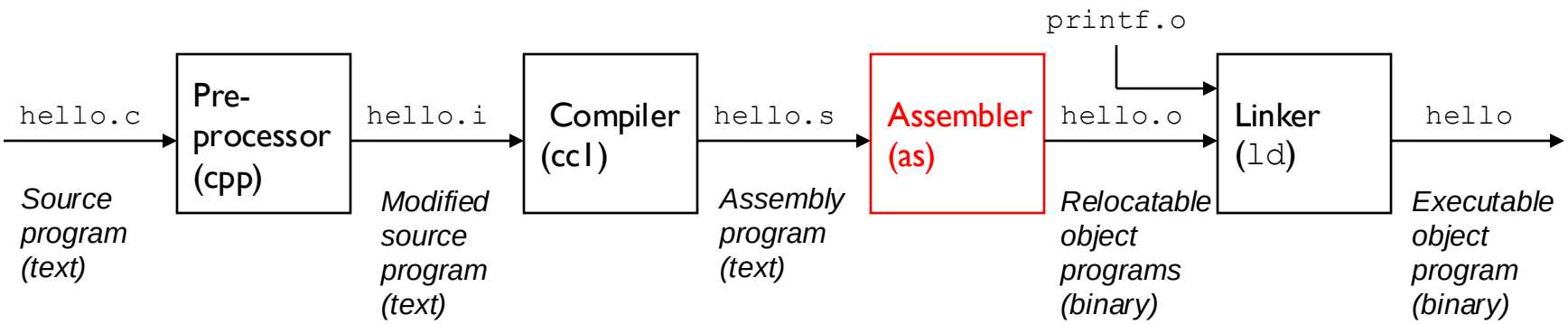
Translating Program

- 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



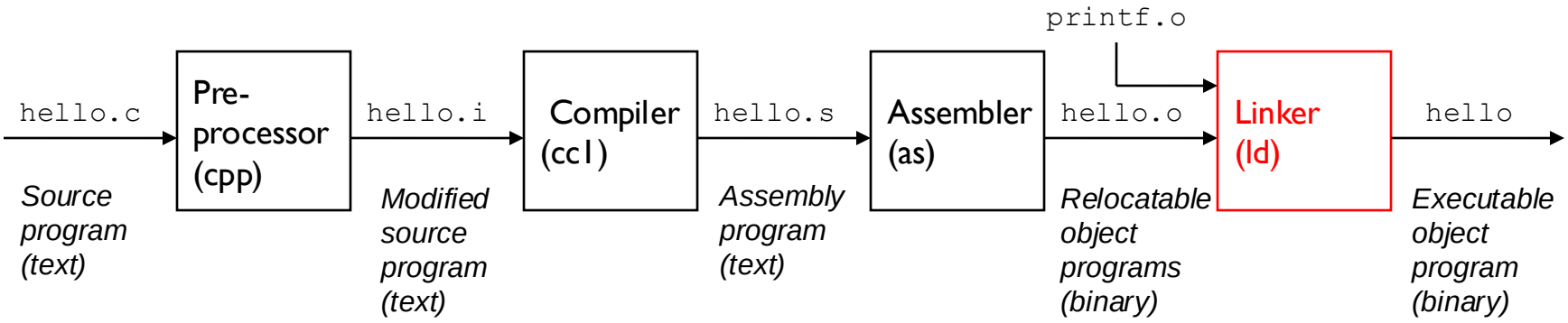
Translating Program

- 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

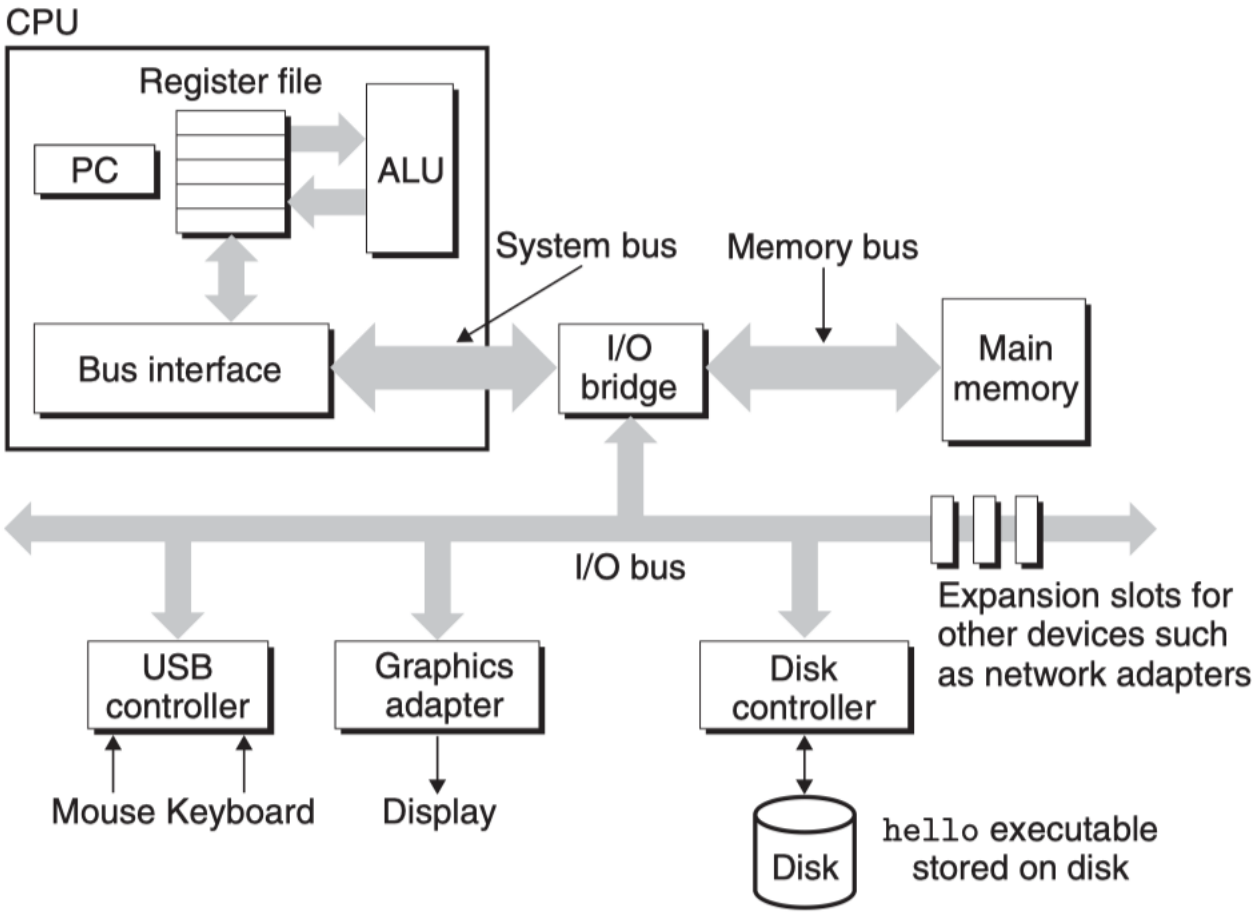
- Eliminating bugs
  - #define min(x, y) x < y ? x : y [example01.c]
- Optimizing program performance
  - If-else vs. switch-case
  - foo * 1024 → foo << 10
- Understanding link-time errors
  - undefined reference to….
- Avoiding security holes
  - Buffer overflow

Executing Program

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

Shell loads and runs the program

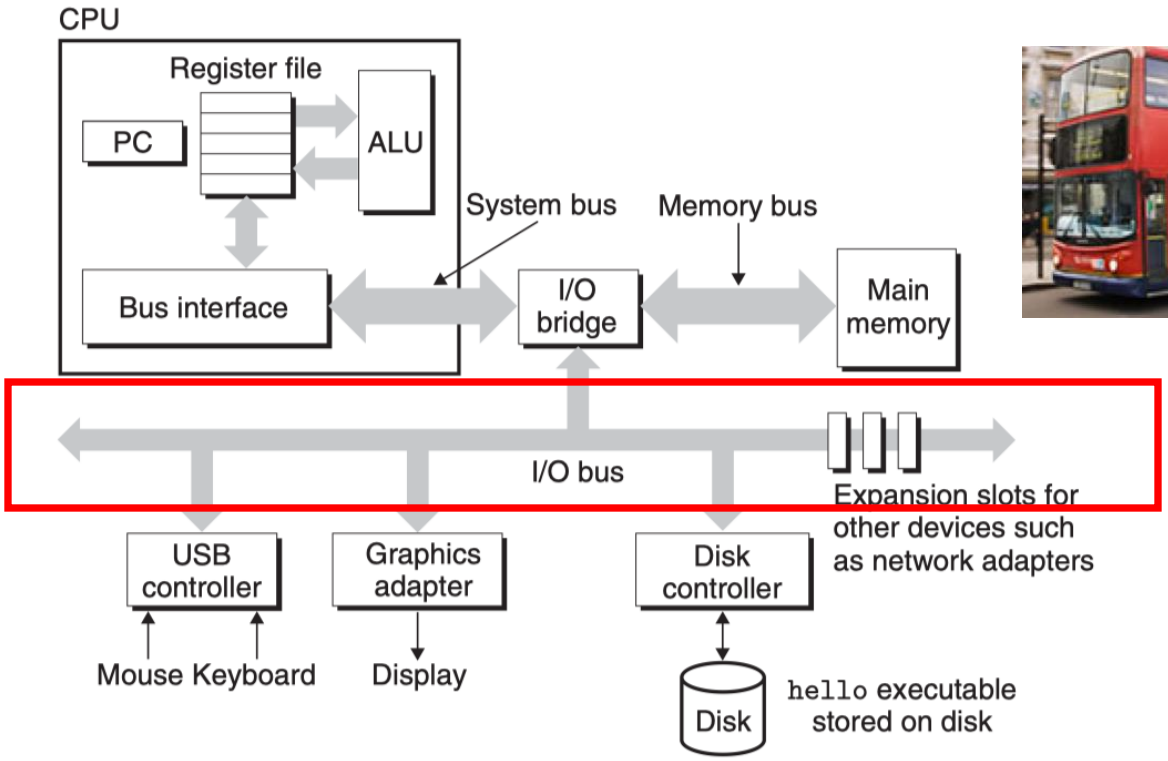
Hardware Organization



Hardware organization of a typical system

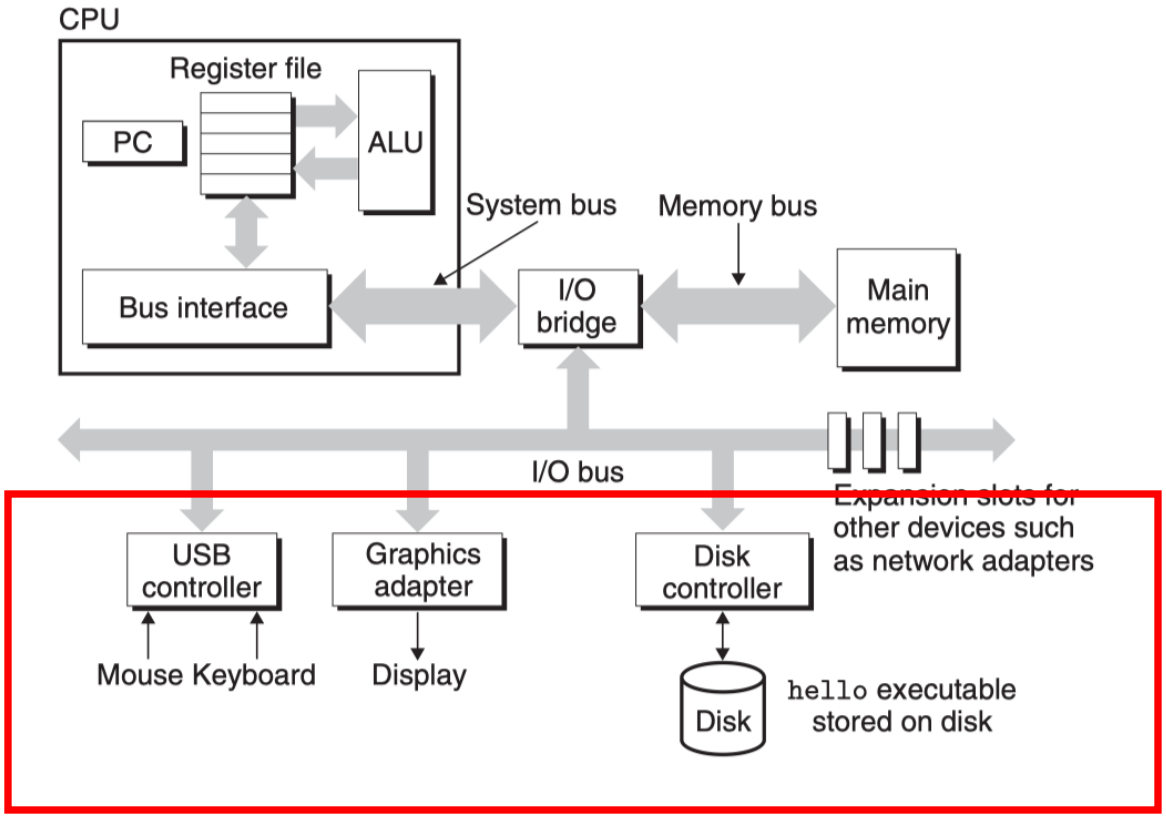
Hardware Organization

- Buses



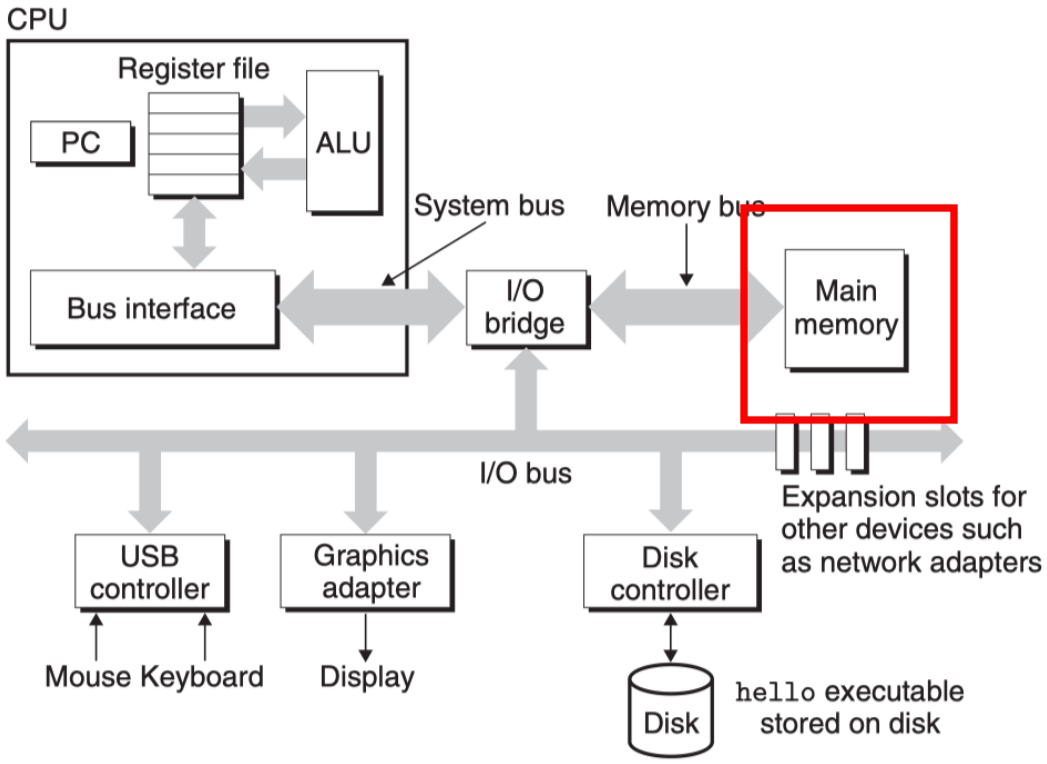
Hardware Organization

- I/O Devices (Chapter 6&10)



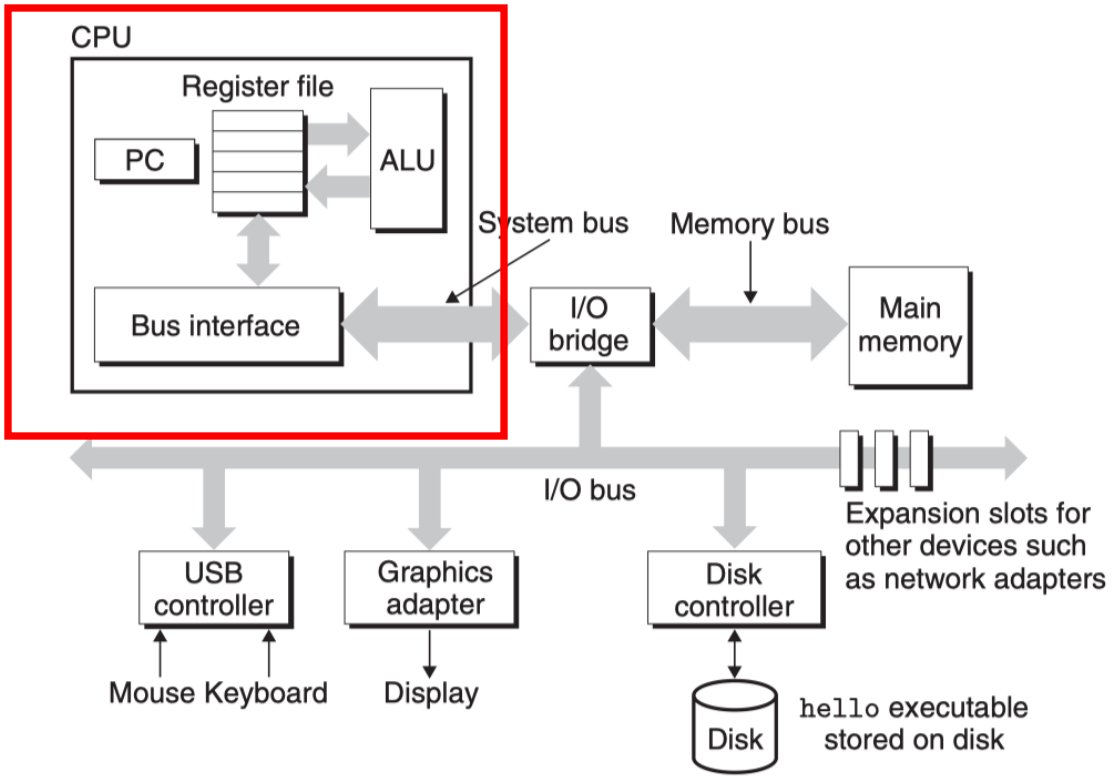
Hardware Organization

- Main Memory (Chapter 6)
  - Temporary storage



Hardware Organization

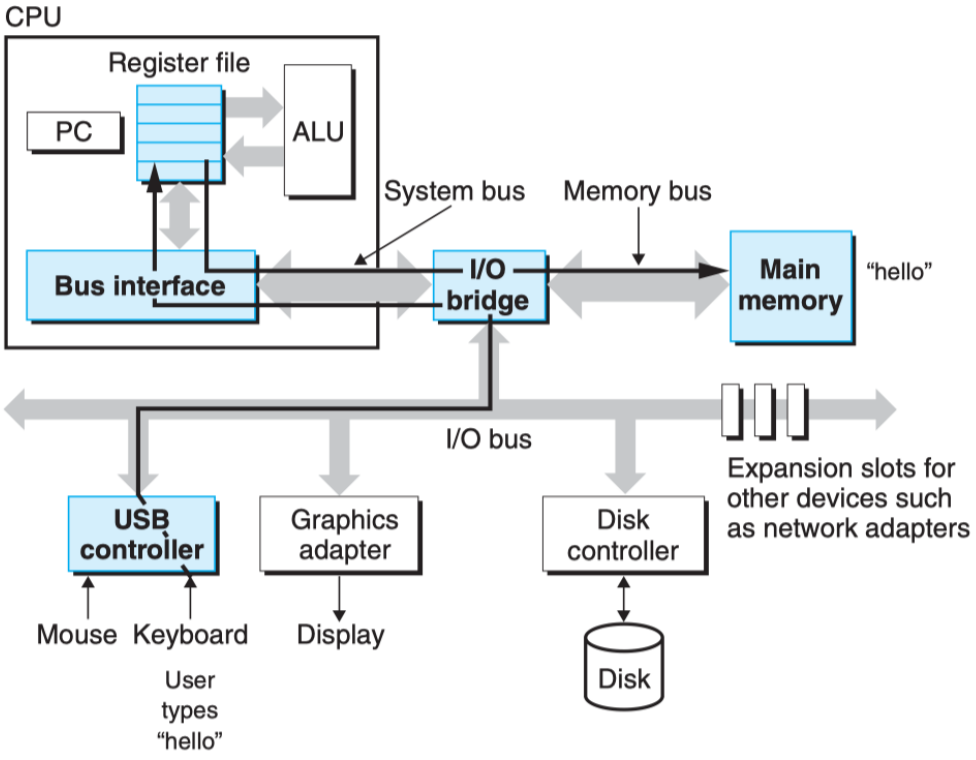
- Processor (Chapter 4)



Executing Program

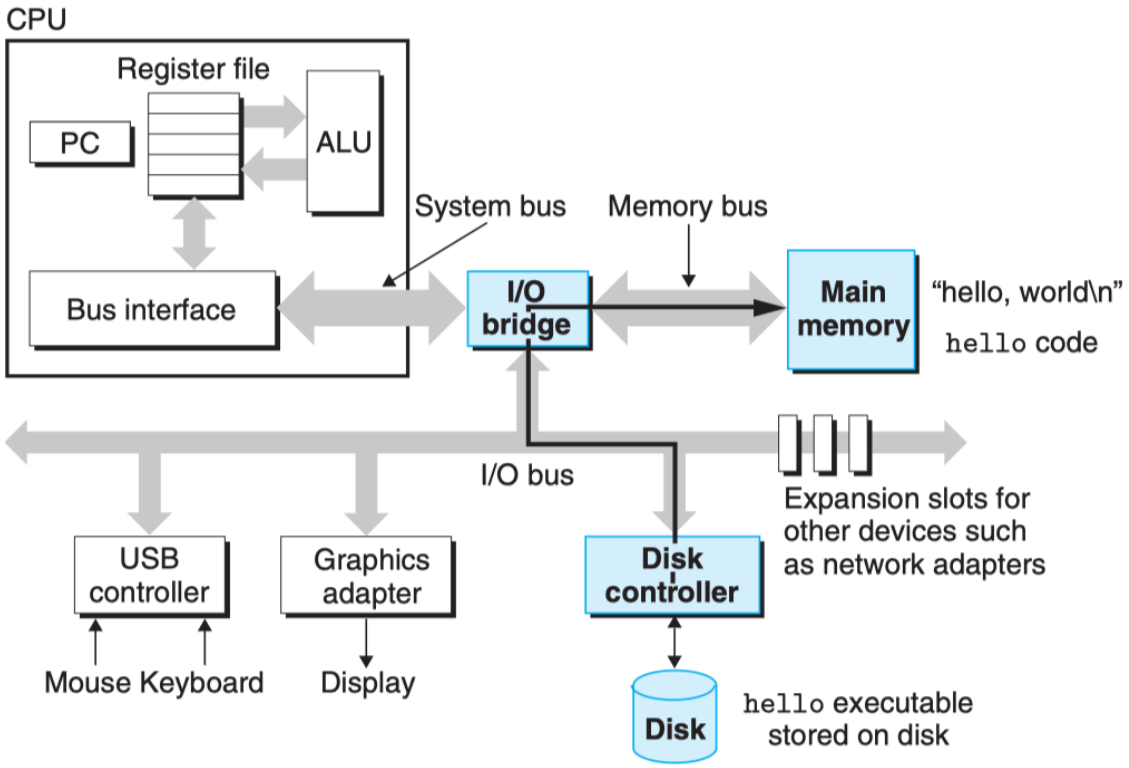
- Shell read "./hello" from keyboard into a register
- Store it into memory

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



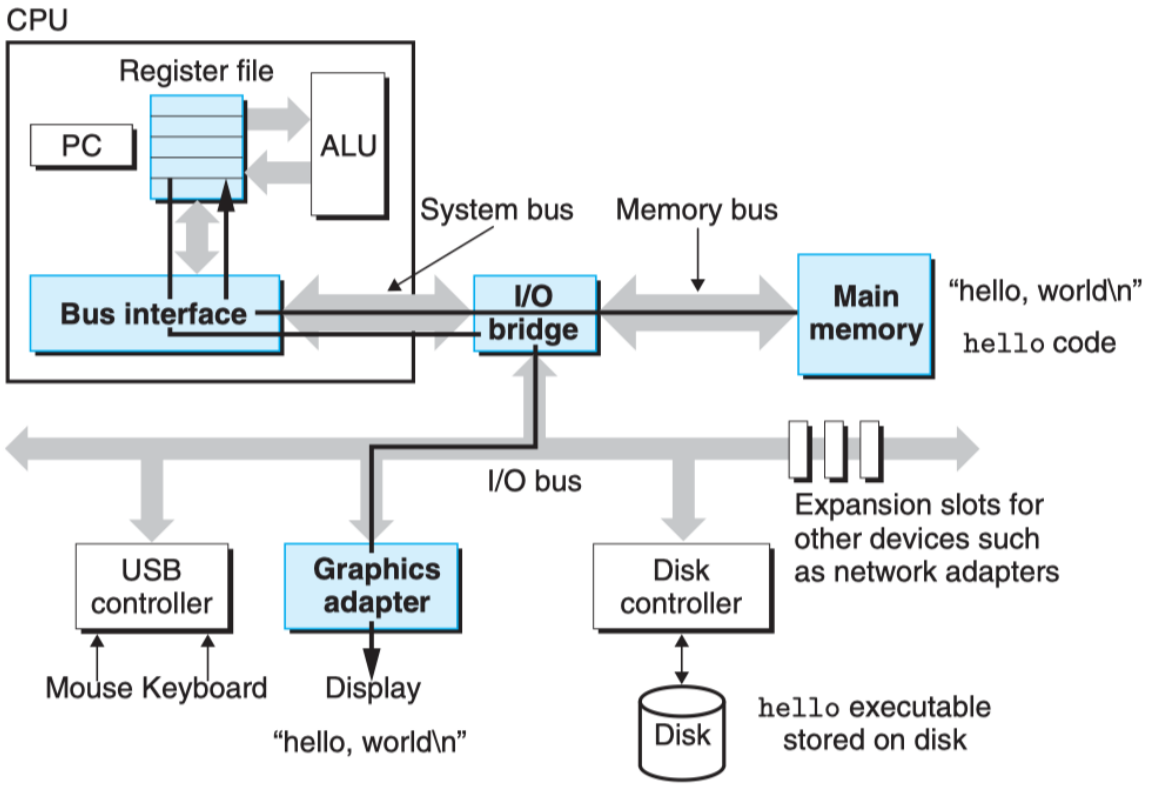
Executing Program

- 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

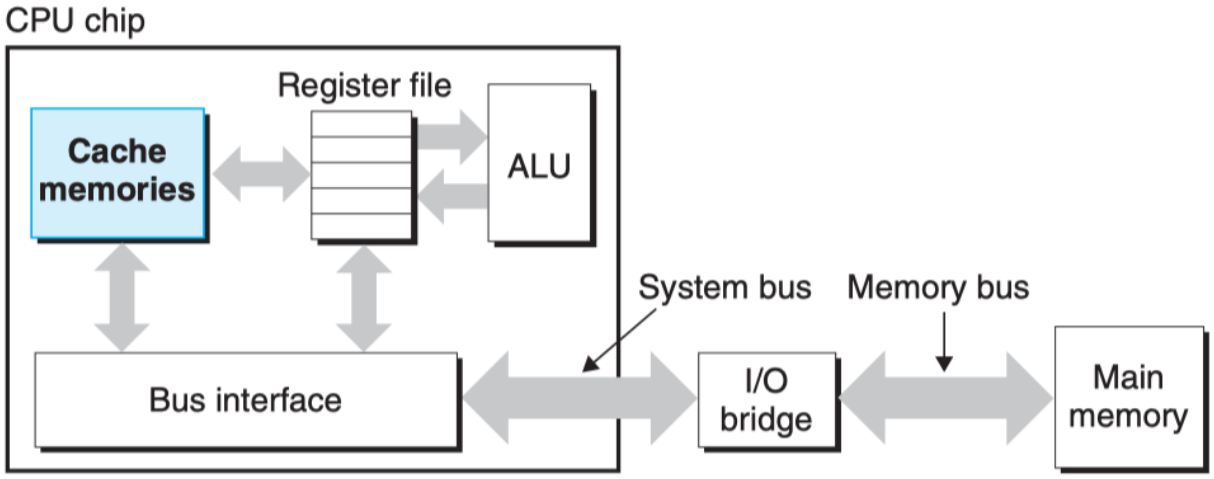


Memory Architecture

- 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

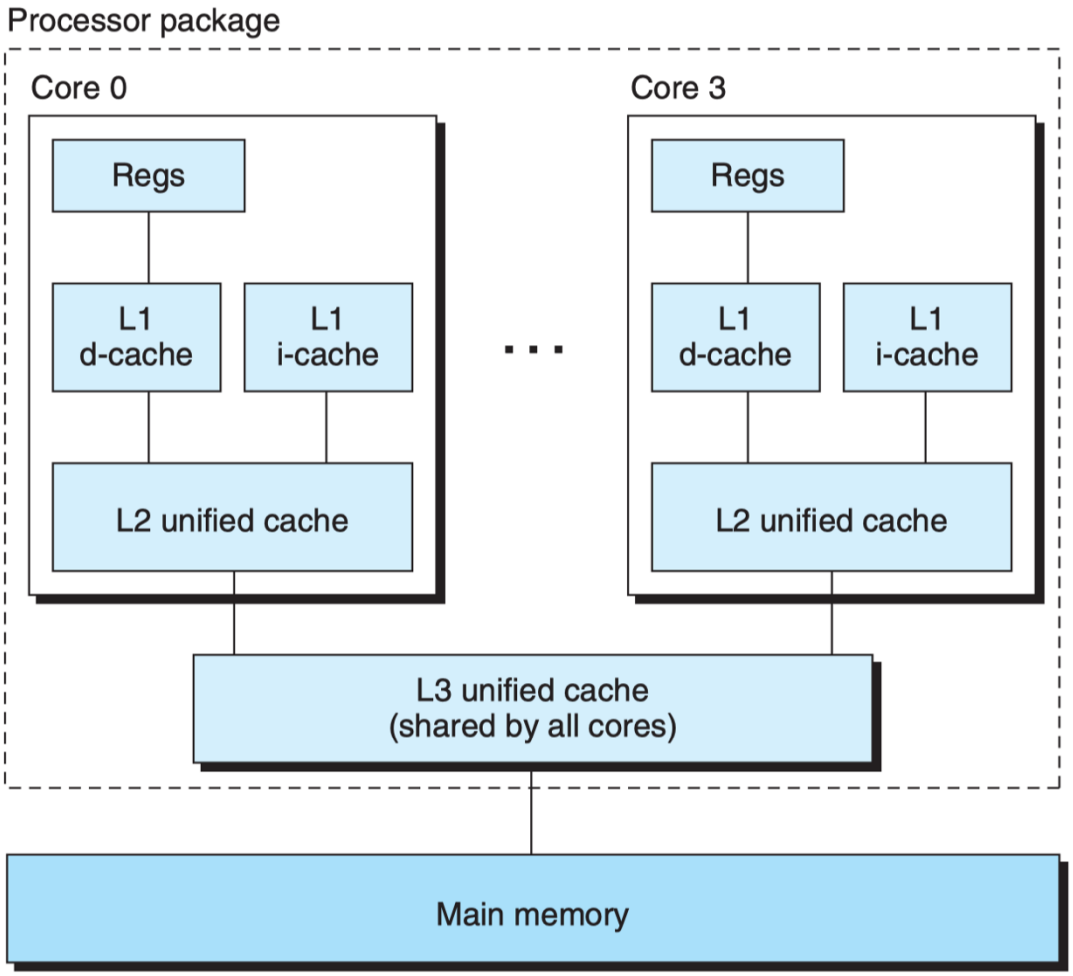
Memory Architecture

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



Memory Architecture

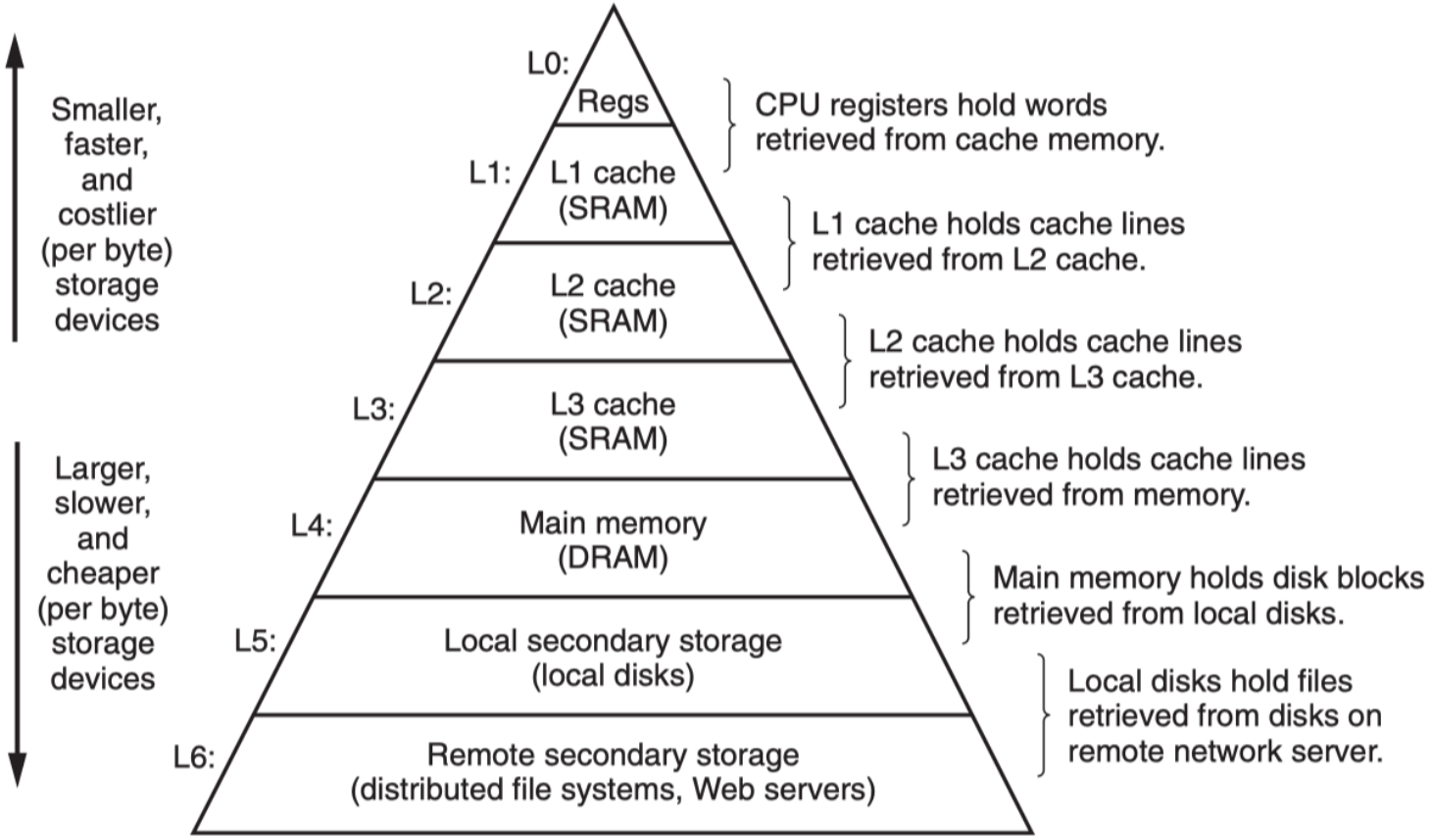
- Cache



Intel Core i7

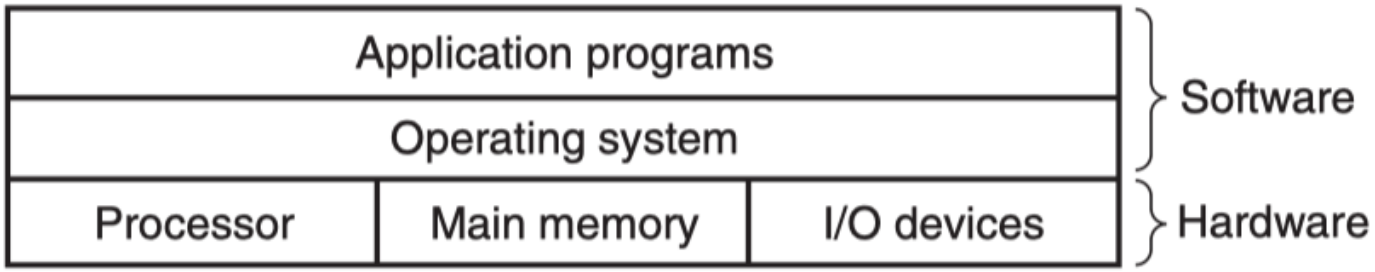
Memory Architecture

- 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