



计算机系统基础



北京邮电大学 计算机学院

周 锋

zfeng@bupt.edu.cn

课程目的

- 使学生能从程序员角度出发，理解高级语言中的变量、语句和函数调用等在计算机系统中的实现细节
- 建立C语言程序、汇编语言、编译器和链接器等之间的关联关系
- 初步理解基本输入输出控制方式、I/O函数的用法和设备驱动程序基本原理
- 建立起层次型计算机系统概念，增强在程序设计和程序调试等方面的动手能力，为后续课程打下坚实基础

课程计划

- 总 学 时：32学时
- 课堂教学：32学时
- 作 业：6 ~ 8次
- 考核方式：期中考试
期末考试（闭卷）
- 成绩评定：平时成绩50%（作业、期中考试、考勤等）
期末考试50%
- 答 疑：课前、课间、课后及微信群

课程计划 – 《计算机系统基础实践》

- 本课程**配套**实践课
- 总学时：12学时
- 课堂教学：8学时
- 实验内容：4次实验
- 考核方式：1、实验报告+人工验收（实验演示、回答问题）
2、机器自动评分+实验报告
- 开始时间：第7、9、10、12、14周

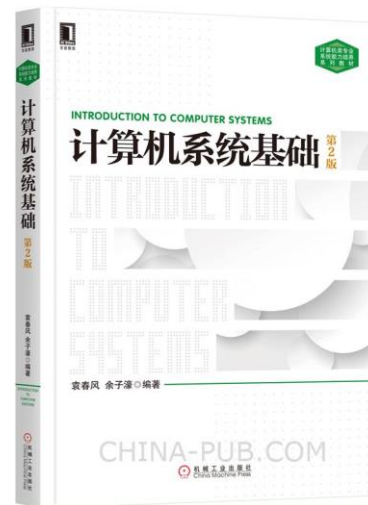
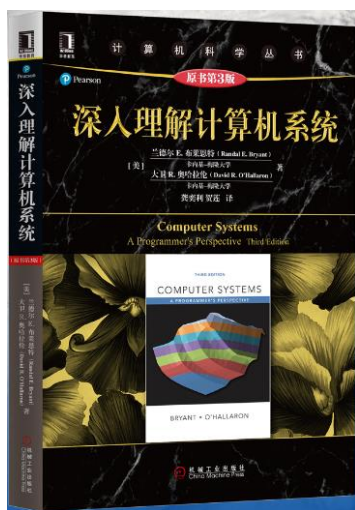
教材及参考书

教材

- 《深入理解计算机系统》 **第3版**；Randal E. Bryant 等；机械工业出版社
- 教材官网 <http://csapp.cs.cmu.edu>

参考书

- 《计算机系统基础》；袁春风；机械工业出版社



课件网址

- 教学云平台“计算机系统基础”课程

- CMU Class Web page

- <http://www.cs.cmu.edu/~213>

由于课时限制，本课程仅是CMU课程内容的一部分

课堂要求

- 不要迟到、不要随意进出教室
- 自己完成作业、按时提交作业、实验报告
- 禁止玩手机



手机静音



你可能遇到的问题（摘自同学反馈）

■ 英文PPT，理解有困难，课后没怎么看

建议 课后一定要先消化PPT内容，不懂之处去阅读教材章节，英文专业术语需要记住，后续课程将受益匪浅.....

■ 讲课内容顺序与教材对应不太一致

解释 由于“计算机系统基础实践”包含4个实验，按照实验开设的时间对课程内容进行了调整.....

■ 教材内容太多，认真阅读需要很多时间

建议 国外教材比较适合自学，但有时内容较长，完整阅读耗时长，最好按需阅读.....

■ 课程内容多、难度大

解释 本课程源自CMU的经典课程，由于本课程学分少，我们仅讲授了原课程的部分内容，但尽量做到“原汁原味”

从程序员视角全面剖析计算机系统，国内北大、清华、上交、南大等知名高校选用CMU教材开设类似课程.....

Overview

■ Big Picture

- Course theme
- Five realities
- How the course fits into the CS curriculum
- Anatomy of a Computer System

■ Lab Assignments Overview

■ Academic integrity (学术诚信)

The Big Picture

Course Theme: (Systems) Knowledge is Power!

■ Systems Knowledge

- How hardware (processors, memories, disk drives, network infrastructure)

处理器

存储器

磁盘驱动器

网络基础设施

plus software (operating systems, compilers, libraries, network protocols)

操作系统

编译器

库

网络协议

combine to support the execution of application programs

- How you as a programmer can best use these resources

■ Useful outcomes from taking this course

- Become more effective programmers
 - Able to find and eliminate bugs efficiently
 - Able to understand and tune for program performance
- Prepare for later “systems” classes in CS & ECE
 - Compilers (编译), Operating Systems (操作系统), Networks, Computer Architecture (计算机系统结构), Embedded Systems (嵌入式系统), Storage Systems, etc.

It's Important to Understand How Things Work

■ Why do I need to know this stuff?

- Abstraction is good, but don't forget reality

■ Most CS and CE courses emphasize abstraction

- Abstract data types
- Asymptotic analysis (渐进式分析)

■ But it's helpful to understand what abstractions build upon and their limits!

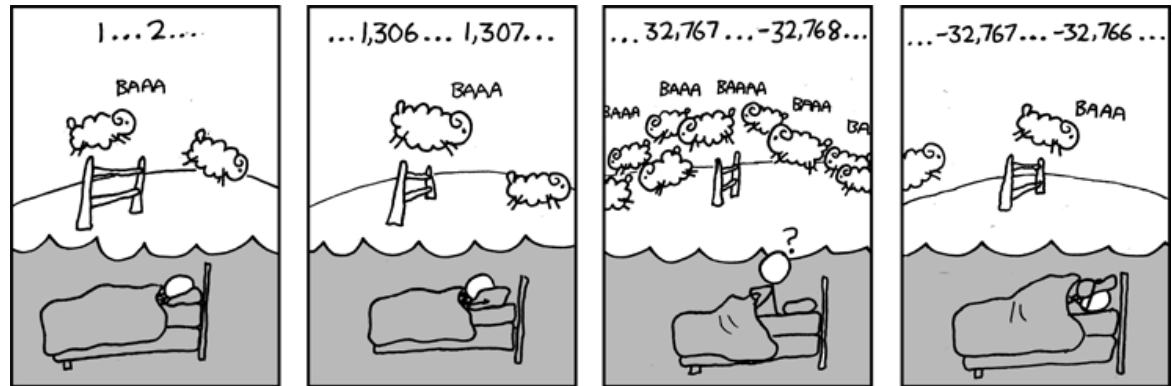
- Especially in the presence of bugs
- Need to understand details of underlying implementations
- Sometimes the abstract interfaces don't provide the level of control or performance you need

Great Reality #1:

Ints are not Integers, Floats are not Reals

■ Example 1: Is $x^2 \geq 0$?

- Float's: Yes!



- Int's:

- $40000 * 40000 \rightarrow 1600000000$
- $50000 * 50000 \rightarrow ?$

■ Example 2: Is $(x + y) + z = x + (y + z)$?

- Unsigned & Signed Int's: Yes!
- Float's:
 - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
 - $1e20 + (-1e20 + 3.14) \rightarrow ??$

Computer Arithmetic

■ Does not generate random values

- Arithmetic operations have important mathematical properties

■ Cannot assume all “usual” mathematical properties

- Due to finiteness (有限性) of representations
- Integer operations satisfy “ring” (环) properties
 - Commutativity (交换性), associativity (结合性), distributivity (分配性)
- Floating point operations satisfy “ordering” properties
 - Monotonicity (单调性), values of signs

■ Observation

- Need to understand which abstractions apply in which contexts
- Important issues for compiler writers and serious application programmers

Great Reality #2:

You've Got to Know Assembly (汇编语言)

- Chances are, you'll never write programs in assembly
- Compilers are much better & more patient than you are
- But: Understanding assembly is key to machine-level execution model
 - Behavior of programs in presence of bugs
 - High-level language models break down
 - Tuning program performance
 - Understand optimizations done / not done by the compiler
 - Understanding sources of program inefficiency
 - Implementing system software
 - Compiler has machine code as target
 - Operating systems must manage process state
 - Creating / fighting malware (恶意软件) ^{进程}
 - x86 assembly is the language of choice!

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

(随机存取存储器)

■ Memory is not unbounded

- It must be allocated and managed
- Many applications are memory dominated

■ Memory referencing (引用) bugs especially pernicious

- Effects are distant in both time and space

■ Memory performance is not uniform

- Cache and virtual memory effects can greatly affect program performance

高速缓存 虚拟存储器

- Adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

```
typedef struct {
    int a[2];
    double d;
} struct_t;

double fun(int i) {
    volatile struct_t s;
    s.d = 3.14;
    s.a[i] = 1073741824; /* Possibly out of bounds */
    return s.d;
}
```

fun(0)	-->	3.14
fun(1)	-->	3.14
fun(2)	-->	3.1399998664856
fun(3)	-->	2.00000061035156
fun(4)	-->	3.14
fun(6)	-->	Segmentation fault (段错误)

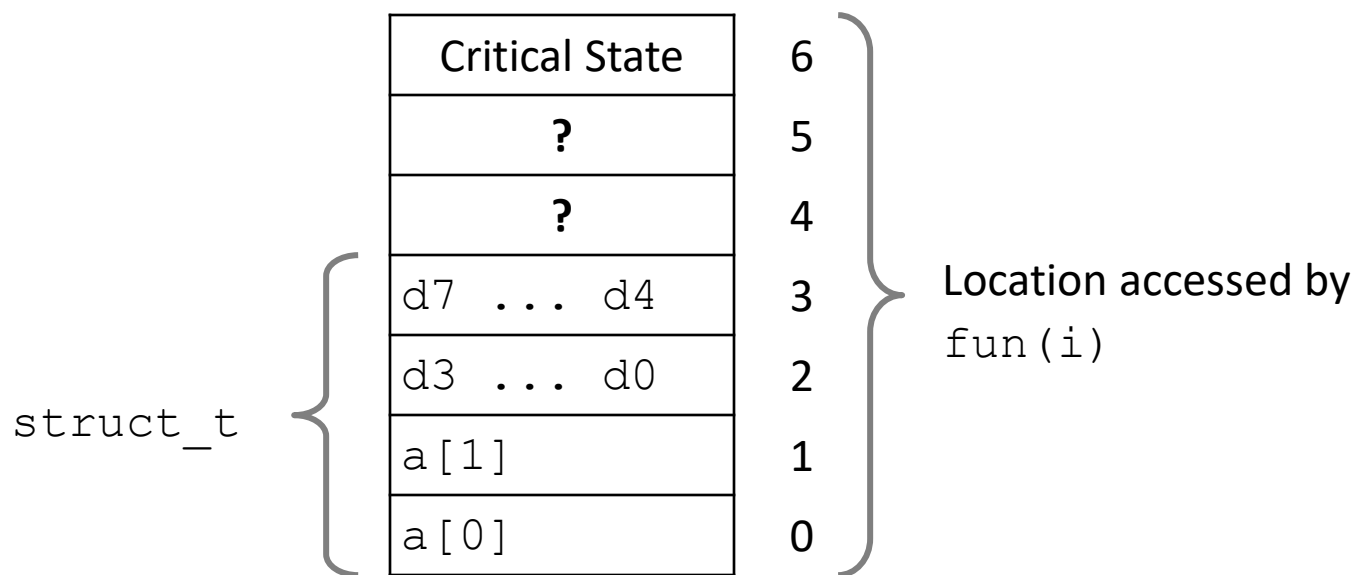
- Result is system specific

Memory Referencing Bug Example

```
typedef struct {
    int a[2];
    double d;
} struct_t;
```

```
fun(0) --> 3.14
fun(1) --> 3.14
fun(2) --> 3.1399998664856
fun(3) --> 2.00000061035156
fun(4) --> 3.14
fun(6) --> Segmentation fault
```

Explanation:



Memory Referencing Errors

■ C and C++ do not provide any memory protection

- Out of bounds array references
- Invalid pointer values
- Abuses of malloc/free

■ Can lead to nasty bugs

- Whether or not bug has any effect depends on system and compiler
- Action at a distance
 - Corrupted object logically unrelated to one being accessed
 - Effect of bug may be first observed long after it is generated

■ How can I deal with this?

- Program in Java, Ruby, Python, ML, ...
- Understand what possible interactions may occur
- Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
 - Easily see 10:1 performance range depending on how code written
 - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
 - How programs compiled and executed
 - How to measure program performance and identify bottlenecks
 - How to improve performance without destroying code modularity and generality

Memory System Performance Example

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

4.3ms

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```

81.8ms

2.0 GHz Intel Core i7 Haswell

- Hierarchical memory organization
- Performance depends on access patterns
 - Including how step through multi-dimensional array

Great Reality #5:

Computers do more than execute programs

- **They need to get data in and out**
 - I/O system critical to program reliability and performance

- **They communicate with each other over networks**
 - Many system-level issues arise in presence of network
 - Concurrent (并发) operations by autonomous processes (进程)
 - Coping with unreliable media
 - Cross platform compatibility (兼容性)
 - Complex performance issues

Course Perspective

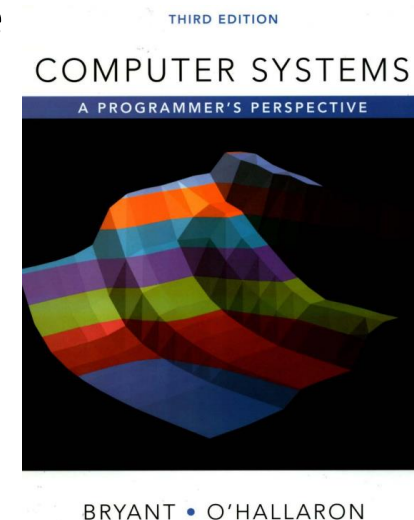
■ Most Systems Courses are Builder-Centric

- Computer Architecture
 - Design pipelined (流水线) processor in Verilog
- Operating Systems
 - Implement sample portions of operating system
- Compilers
 - Write compiler for simple language
- Networking
 - Implement and simulate network protocols (协议)

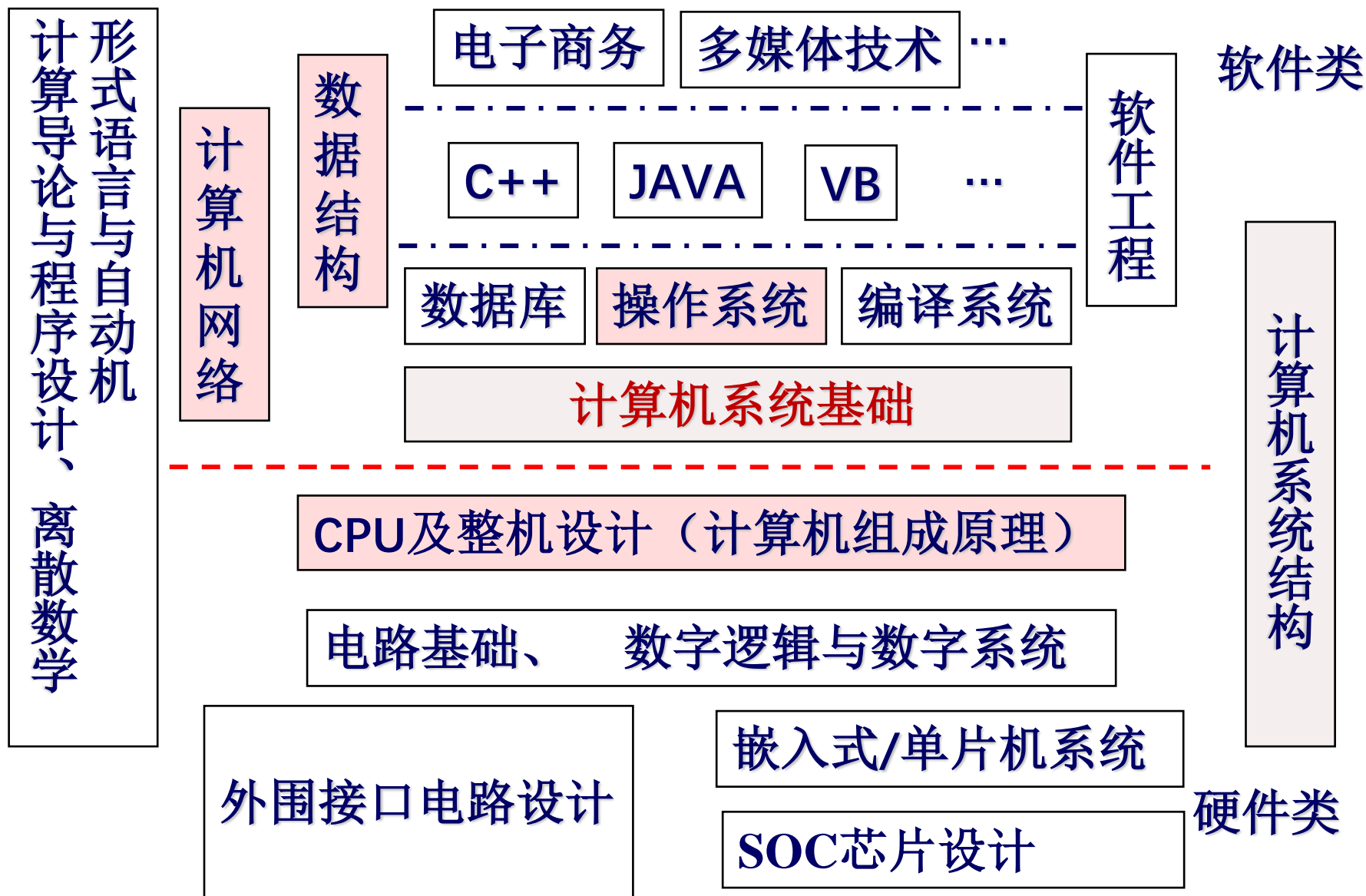
Course Perspective (Cont.)

■ Our Course is Programmer-Centric

- By knowing more about the underlying system, you can be more effective as a programmer
- Enable you to
 - Write programs that are more reliable and efficient
 - Incorporate features that require hooks into OS
 - E.g., concurrency, signal handlers (处理程序)
- Cover material in this course that you won't see elsewhere
- Not just a course for dedicated hackers
 - **We bring out the hidden hacker in everyone!**



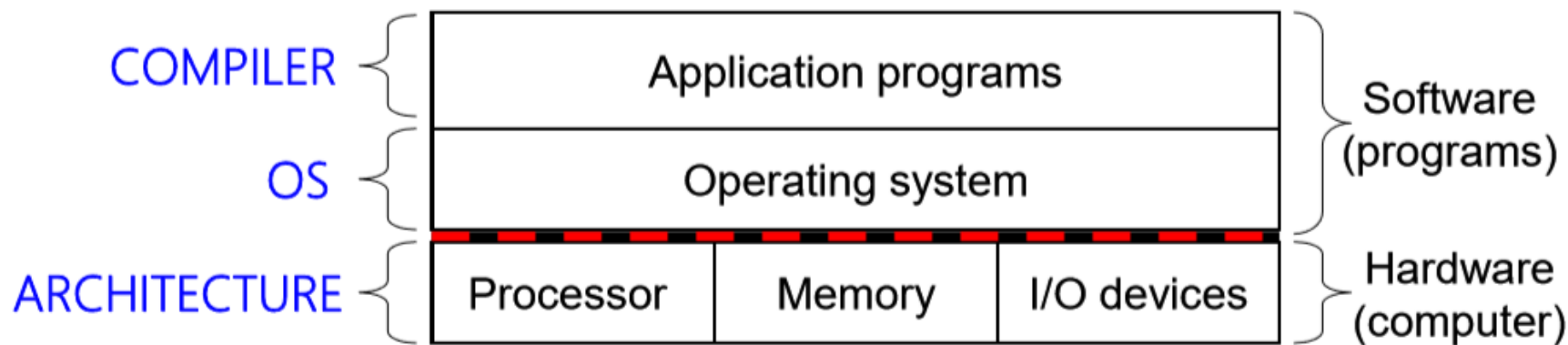
Role within CS Curriculum (课程设置)



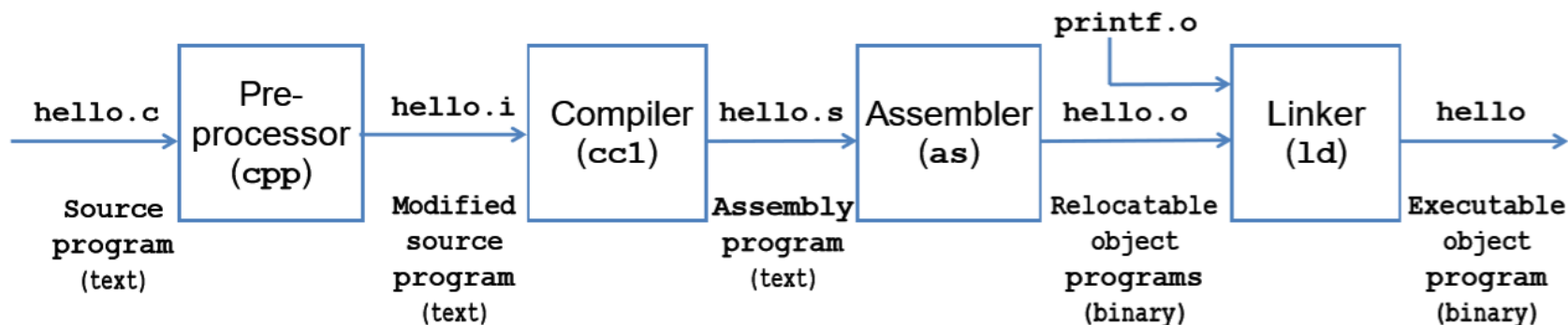
Anatomy of a Computer System

■ What is a Computer System?

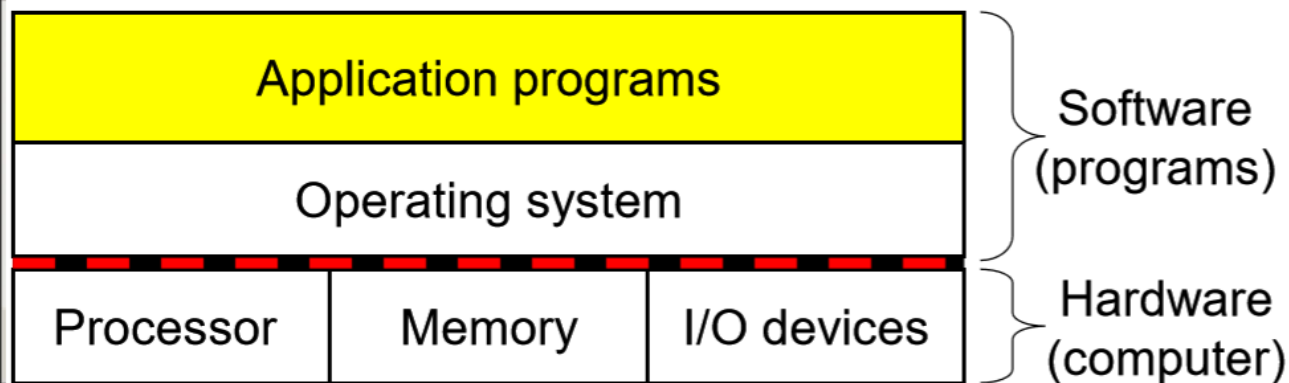
- Software + Hardware
- Programs + Computer → [Application program + OS] + Computer
- Programming Language + Operating Systems + Computer Architecture



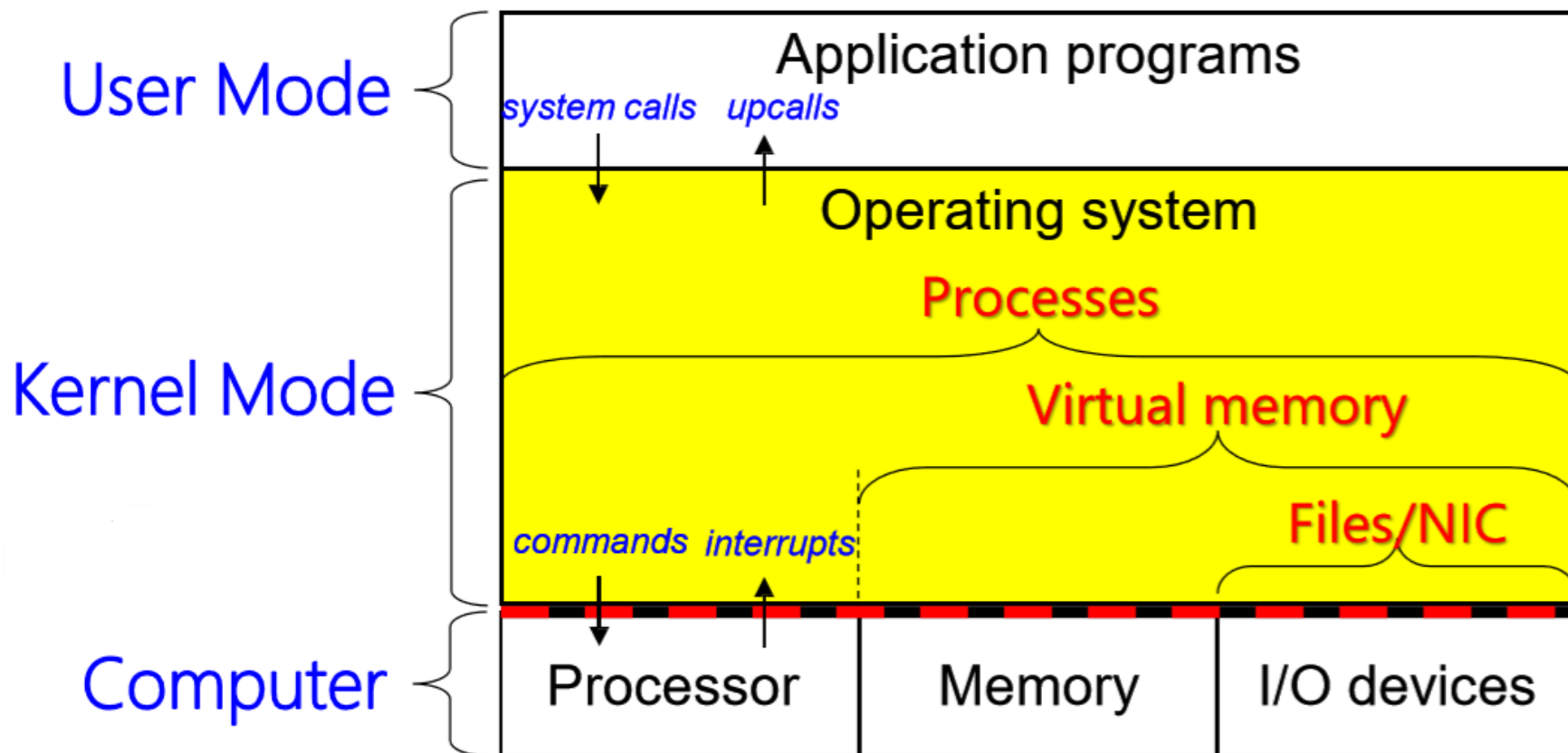
Anatomy of a Computer System: Compiler



The image shows three screenshots of code. The leftmost screenshot is labeled **SOURCE CODE** and shows C code for a program that prints "hello, world!". The middle screenshot is labeled **ASSEMBLY CODE** and shows the corresponding assembly code for the same program. The rightmost screenshot is labeled **OBJECT CODE** and shows the binary representation of the program.

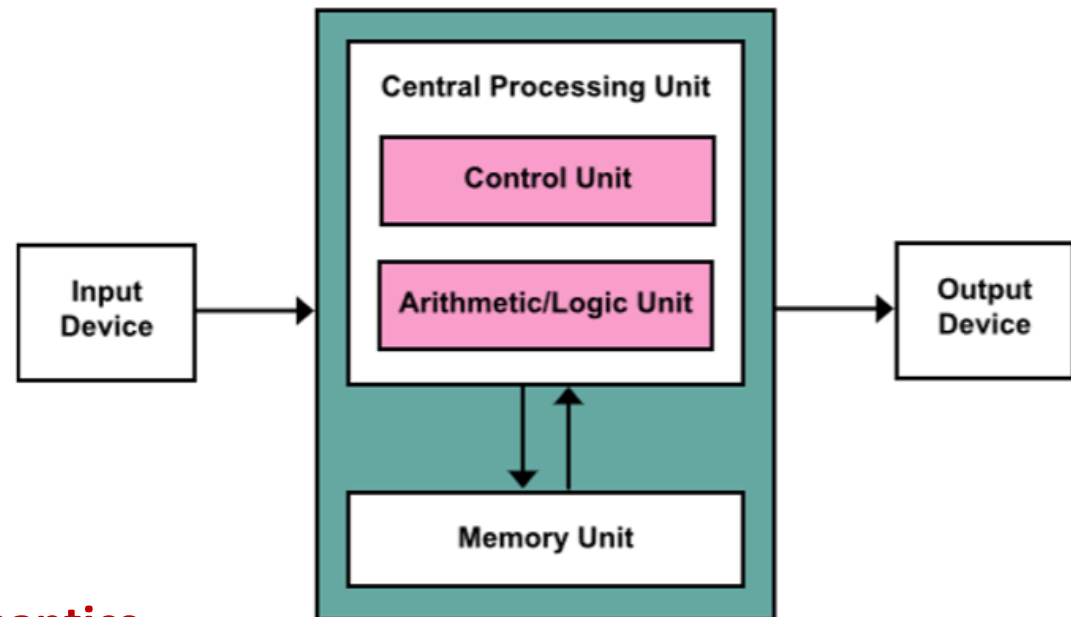


Anatomy of a Computer System: OS

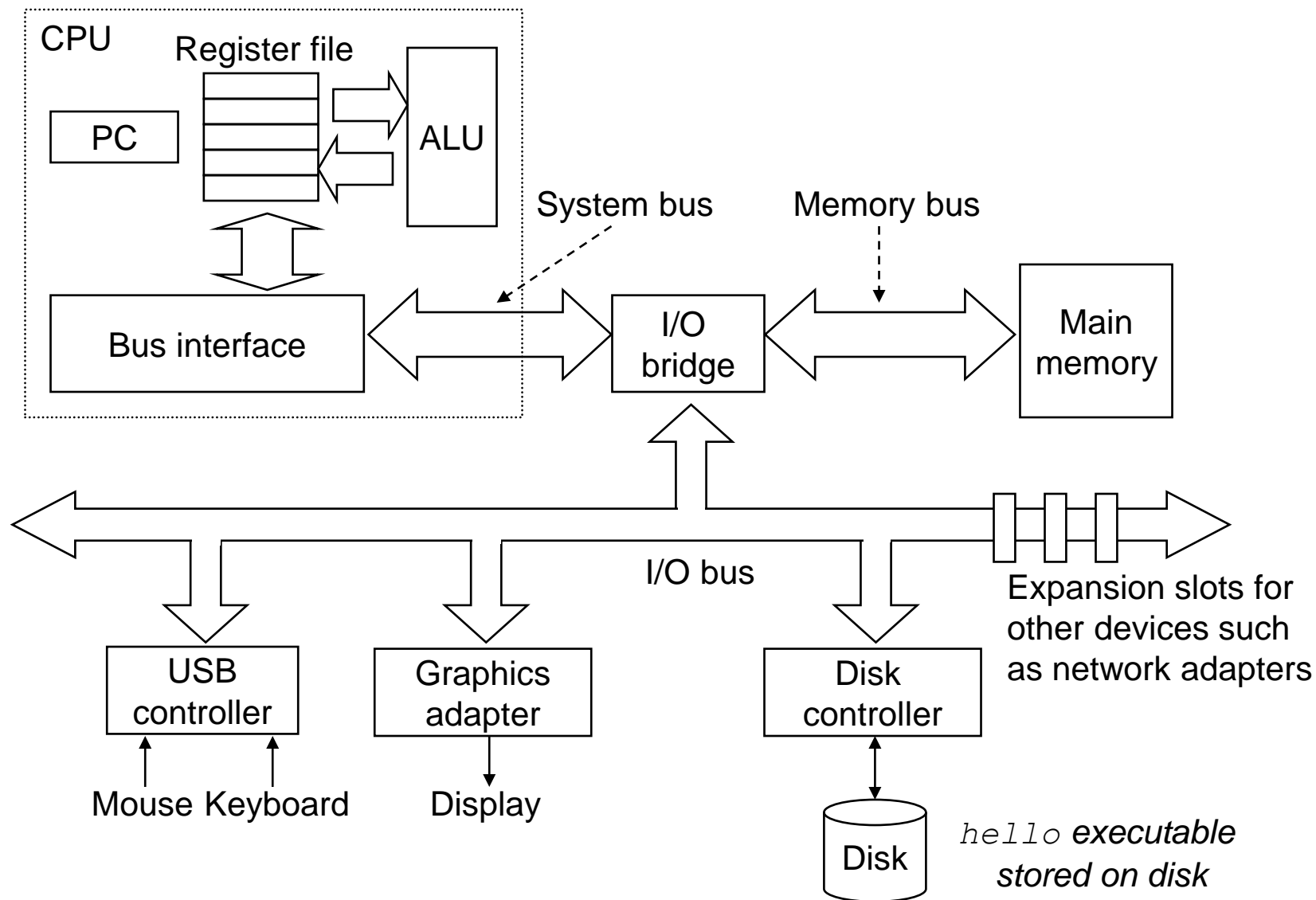


Von Neumann Computer Model

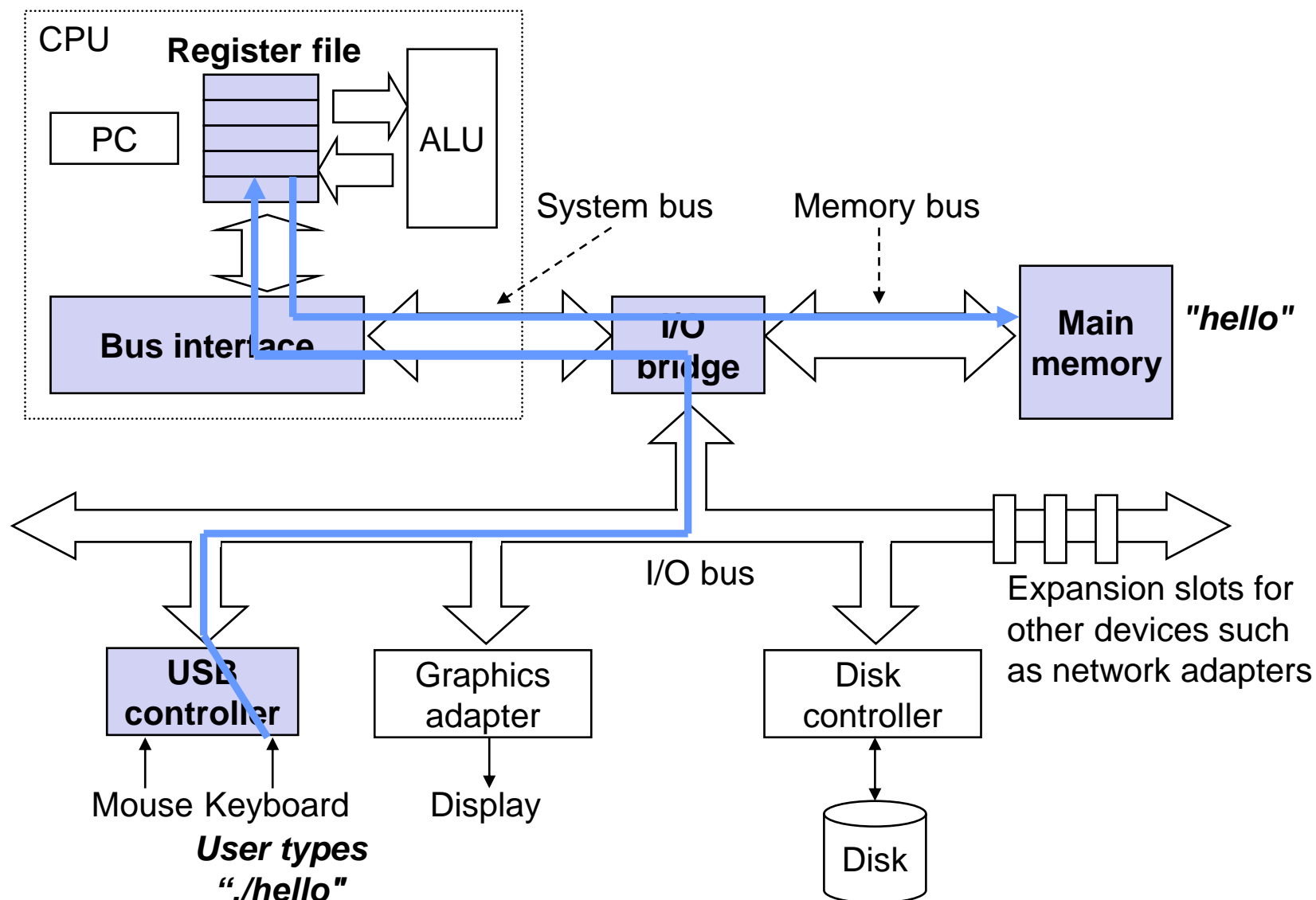
- Proposed in 1945 by Von Neumann and others (Alan Turing, J. Presper Eckert and John Mauchly).
- A “Stored Program Computer”
 1. **One CPU**
 - One Control Unit
 - Arithmetic/Logic Unit
 - Program Counter
 - Instruction Register
 2. **Monolithic Memory**
 - Data Store
 - Instruction Store
 3. **Sequential Execution Semantics**
 - Instructions from an Instruction Set



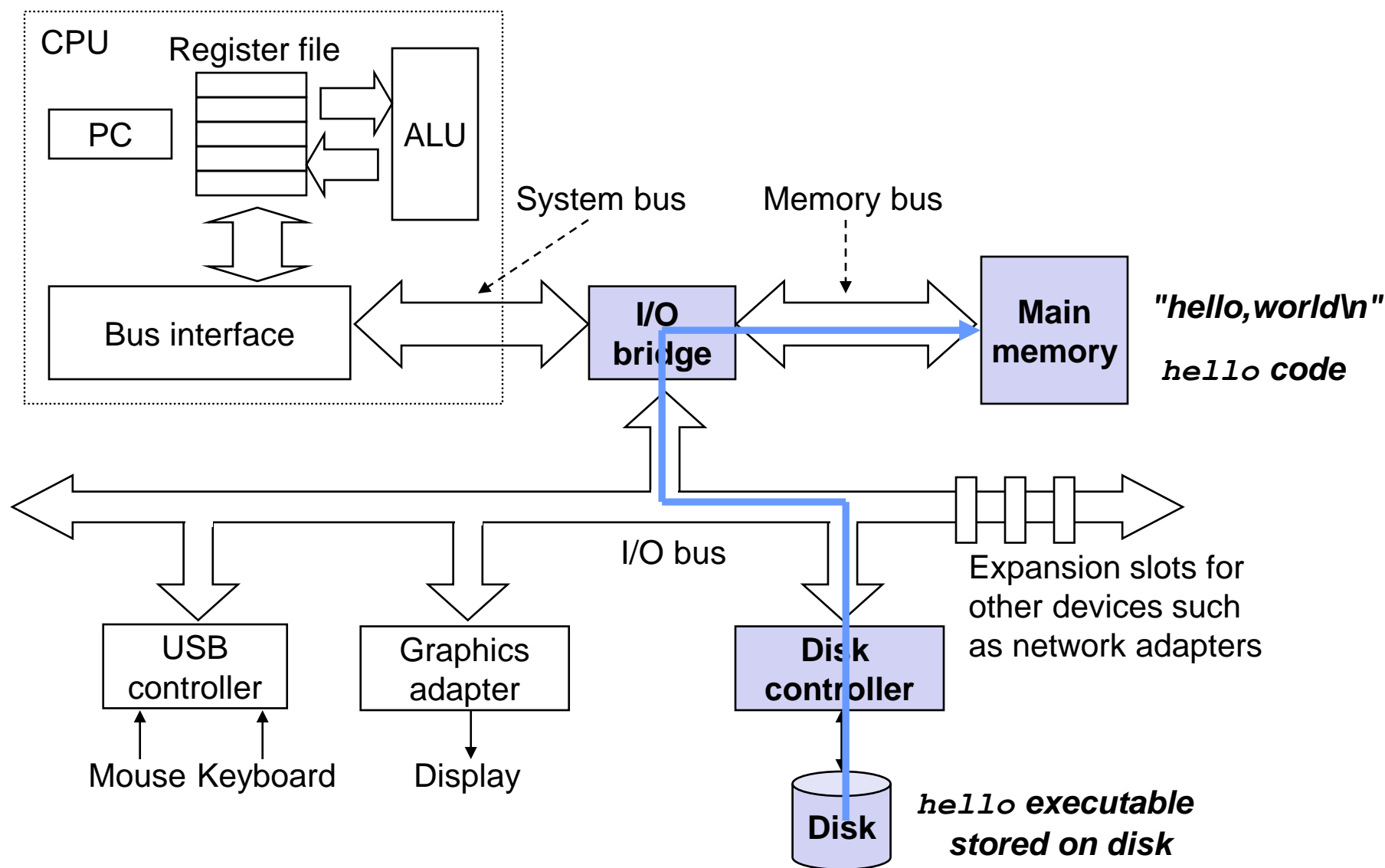
Typical Computer (PC) Today: HW Organization



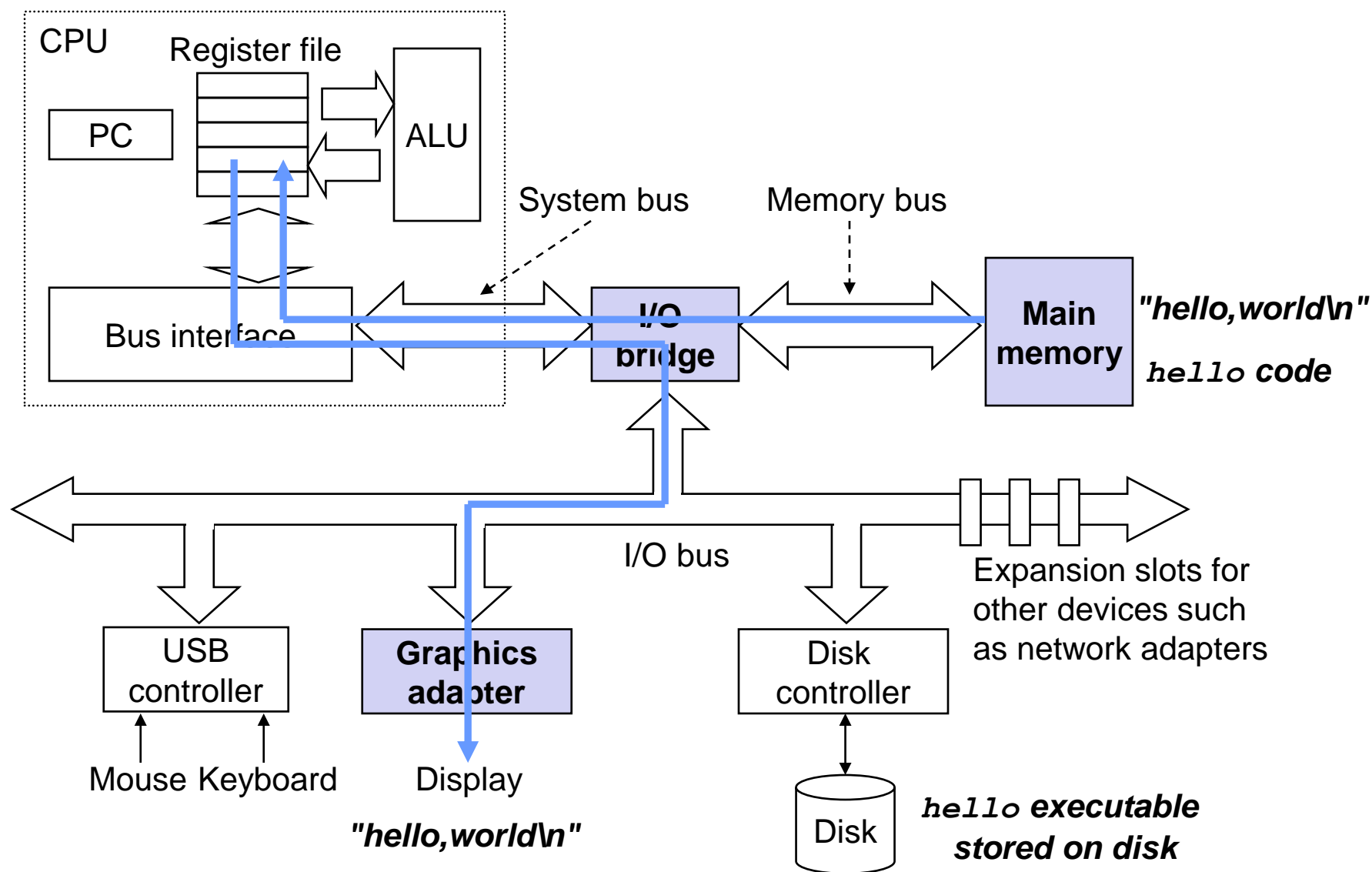
Reading “hello” command from the keyboard



Loading executable from disk to main memory



Writing output string from memory to display



Lab Assignments Overview

Lab Assignments Supporting The Course

Labs

- **Lab1 (Linux&GNU Toolchain Lab):** Basic usage of linux commands, vi commands and GNU toolchain
- **Lab2 (bomblab):** Defusing a binary bomb
- **Lab3 (attacklab):** The basics of code injection attacks
- **Lab4 (I/O lab):** The basics of I/O control program

Facilities

- **Lab1-3 will use bup1 machine** (Dell server) with dual Xeon Processor (total 16 Intel Cores) and 32GB. Login bupt1 using your Student ID and password
- **Lab4 will use Ubuntu virtual machine on your own computer**

Labs Policies

■ Lab schedule

- Starts in week 7, two or three weeks for each lab assignment

■ Work groups

- You must work alone on all lab assignments

■ Hand-ins

- Labs are due at 11:59pm usually on Monday/Wednesday
- Electronic handins using Ucloud Platform

■ Penalty for late submission

- loss of 5 marks per day

■ Autograding and Scoreboards for Lab2 and Lab3

- Autograding: Providing you with instant feedback
- Scoreboards: Real-time, rank-ordered, and summary

Lab Rationale

- **Each lab has a well-defined goal such as solving a puzzle or winning a contest**
- **Doing the lab should result in new skills and concepts**
- **We try to use competition in a fun and healthy way**
 - Set a reasonable threshold for full credit
 - Post intermediate results on Autolab scoreboard for glory!

Academic Integrity

**Please pay close attention, especially
if this is your first semester at CMU**

了解国外大学对学术诚信的要求

Cheating/Plagiarism: Description

■ Unauthorized use of information

- **Borrowing code:** by copying, retyping, *looking at* a file
 - **Describing:** verbal description of code from one person to another
 - *Even if you just describe/discuss how to put together CS:APP code snippets to solve the problem*
 - **Searching the Web** for solutions, discussions, tutorials, blogs, other universities' 213 instances,... in English or any other language
 - **Copying code** from a previous course or online solution
 - **Reusing *your* code** from a previous semester (here or elsewhere)
 - *If you take the course this semester, all work has to be done this semester (unless you have special permission from the instructors)*
- *Using AI to generate your code (e.g., GitHub CoPilot, ChatGPT)*

Cheating/Plagiarism: Description (cont.)

■ Unauthorized supplying of information

- Providing copy: Giving a copy of a file to someone
- Providing access:
 - Putting material in unprotected directory
 - Putting material in unprotected code repository (e.g., Github)
 - Or, letting protections expire
- Applies to this term and the future
 - There is **no statute of limitations** for academic integrity violations

■ Collaborations beyond high-level, strategic advice

- Anything more than block diagram or a few words
- Code / pseudo-code is NOT high level
- Coaching, arranging blocks of allowed code is NOT high level
- Code-level debugging is NOT high level

Cheating/Plagiarism: Description

■ What is NOT cheating?

- Explaining how to use systems or tools
- Helping others with *high-level* design issues
High means VERY high
- Using code supplied by us
- Using code from the CS:APP web site
- Using books from the library, Unix manpages, other published material
 - Except the “Solutions Manual for CS:APP”
- Using *general* online references
 - OK: The [GNU C Library Manual](#), [Beej's Guide to C](#), [cplusplus.com](#)
 - Not OK: searching for “213 malloc solution”

Cheating: Consequences

■ Penalty for cheating:

- Best case: -100% for assignment
 - *You would be better off to turn in nothing*
- Worst case: Removal from course with failing grade
 - This is the default
- Permanent mark on your record
- Loss of respect by you, the instructors and your colleagues
- *If you do cheat – come clean asap!*

■ Detection of cheating:

- We have sophisticated tools for detecting code plagiarism
- In Fall 2015, 20 students were caught cheating and failed the course.
 - Some were **expelled** from the University
- In January 2016, 11 students were penalized for cheating violations that occurred as far back as Spring 2014.
- In May 2019, we gave an AIV to a student who took the course in Fall 2018 for unauthorized coaching of a Spring 2019 student. His grade was changed retroactively.

■ Don't do it!

- Manage your time carefully
- Ask the staff for help when you get stuck

Cheating Notes

- **We have written over 100 letters for cheating cases**
 - Don't add to this total
 - Some have been for years earlier

- **Your work is sophisticated enough that there are many different solutions**
 - Things that look the same are very suspicious
 - If you do your own work and commit regularly, your work is unique

- **We use PhD-level research to detect similarities**
 - Inputs include: multiple tools, online searches, past semester submissions

Some Concrete Examples:

■ This is Cheating:

- Searching the internet with the phrase 15-213, 15213, 213, 18213, malloclab, etc.
 - That's right, just entering it in a search engine
- Looking at someone's code on the computer next to yours
- Giving your code to someone else, now or in the future
- Posting your code in a publicly accessible place on the Internet, now or in the future
- Hacking the course infrastructure

■ This is OK (and encouraged):

- Googling a man page for fputs
- Asking a friend for help with gdb
- Asking a TA or course instructor for help, showing them your code, ...
- Looking in the textbook for a code example
- Talking about a (high-level) approach to the lab with a classmate

How it Feels: Student and Instructor

- Fred is desperate. He can't get his code to work and the deadline is drawing near. In panic and frustration, he searches the web and finds a solution posted by a student. He carefully strips out the comments and inserts his own. He changes the names of the variables and functions. Phew! Got it done!
- The course staff run checking tools that compare all submitted solutions to the solutions from this and other semesters, along with ones that are on the Web.
 - Remember: We are as good at web searching as you are
- Meanwhile, Fred has had an uneasy feeling: Will I get away with it? Why does my conscience bother me?
- Fred gets email from an instructor: "Please see me tomorrow at 9:30 am."
 - Fred does not sleep well that night

How it Feels: Student and Instructor

- The instructor feels frustrated. His job is to help students learn, not to be police. Every hour he spends looking at code for cheating is time that he cannot spend providing help to students. But, these cases can't be overlooked
- At the meeting:
 - Instructor: "Explain why your code looks so much like the code on Github."
 - Fred: "Gee, I don't know. I guess all solutions look pretty much alike."
 - Instructor: "I don't believe you. I am going to file an academic integrity violation."
 - Fred will have the right to appeal, but the instructor does not need him to admit his guilt in order to penalize him.
- Consequences
 - Fred may (most likely) will be given a failing grade for the course
 - Fred will be reported to the university
 - A second AIV will lead to a disciplinary hearing
 - Fred will go through the rest of his life carrying a burden of shame
 - The instructor will experience a combination of betrayal and distress

Why It's a Big Deal

■ This material is best learned by doing

- Even though that can, at times, be difficult and frustrating
- Starting with a copy of a program and then tweaking it is very different from writing from scratch
 - Planning, designing, organizing a program are important skills

■ We are the gateway to other system courses

- Want to make sure everyone completing the course has mastered the material

■ Industry appreciates the value of this course

- We want to make sure anyone claiming to have taken the course is prepared for the real world

■ Working in teams and collaboration is an important skill

- But only if team members have solid foundations
- This course is about foundations, not teamwork

A Scenario: Cheating or Not?

Alice is working on malloc lab and is just plain stuck. Her code is seg faulting and she doesn't know why. It is only 2 days until malloc lab is due and she has 3 other assignments due this same week. She is in the cluster.

Bob is sitting next to her. He is pretty much done.

Sitting next to Bob is Charlie. He is also stuck.

- 1. Charlie gets up for a break and Bob makes a printout of his own code and leaves it on Charlie's chair.
 - Who cheated: Charlie? Bob?
- 2. Charlie finds the copy of Bob's malloc code, looks it over, and then copies one function, but changes the names of all the variables.
 - Who cheated: Charlie? Bob?

Another Scenario

Alice is working on malloc lab and is just plain stuck. Her code is seg faulting and she doesn't know why. It is only 2 days until malloc lab is due and she has 3 other assignments due this same week. She is in the cluster.

Bob is sitting next to her. He is pretty much done.

Sitting next to Bob is Charlie. He is also stuck.

- **1. Bob offers to help Alice and they go over her code together.**
 - Who cheated: Bob? Alice?
- **2. Bob gets up to go to the bathroom and Charlie looks over at his screen to see how Bob implemented his free list.**
 - Who cheated: Charlie? Bob?

Another Scenario (cont.)

- **3. Alice is having trouble with GDB. She asks Bob how to set a breakpoint, and he shows her.**
 - Who cheated: Bob? Alice?
- **4. Charlie goes to a TA and asks for help**
 - Who cheated: Charlie?
- **If you are uncertain which of these constitutes cheating, and which do not, please read the syllabus carefully. If you're still uncertain, ask one of the staff**

*Welcome
and Enjoy!*

教材阅读

■ 第1章 1.1-1.4