Course Overview

OS

RAM

CPU Architecture.

Bit Level.

Prof. Hyuk-Yoon Kwon

Most parts are based on slides written by Brayant and O'Hallaon, CMU
(http://csapp.cs.cmu.edu/3e/instructors.html)

Contents

Part1: Course Overview

- Course Policies
- Course Theme
- Why This Course Important? Five realities!

Part2: The Overview of Computer Systems

Course Objective

- In this course, we focus on **what really happens when your programs run**. The aim of this course is to explain the enduring concepts **underlying all computer systems**, and to show you the concrete ways that these ideas affect the **correctness, performance, and utility** of your application programs.

Course Description

- This course deals with the basic concepts underlying all computer systems **from a programmer’s perspective**. It describes how application programmers can use their knowledge of a system **to write better programs**. Specifically, this course will cover **how to represent and manipulate data in computers**, machine-level representation of programs, processor architecture, memory hierarchy, and virtual memory.

  Lecture notes and textbooks provide many useful program samples **written in C programming language**, which is still the representative language to implement system-layer applications. During the entire course, it is recommended to be familiar with C language. For the students who are not familiar with C, we will cover the basics of C in the beginning of the course.

Course Policy: Lectures

Flipped learning

- (Online lecture contents) The contents will be uploaded **by the previous Tuesday**
- (Offline lecture) **Every Monday 11:00 AM ~** 1시간 정도 대면수업 예정!
  - Summary (will be given in English)
  - Q&A (Korean questions are also allowed)

Lecture place

- Laboratory: Frontier 511

Course Policy: Grading

- **Exams (70%)**
  - Mid-term (30%)
  - Final (40%)
- **Assignments (30%)**
  - Homework (10%) → 3개
  - Projects (20%) 2 Team Projects (2-3 members)

    ↳ Making Program.

Course Policy: Homework

- **Penalty for cheating**
  - The lowest grade
- **What is cheating?**
  - **Sharing code**: by copying, retyping, looking at, or supplying a file
  - **Searching the Web for solutions**
  - Helping your friend to write a lab, line by line
- **What is NOT cheating?**
  - Helping others with high-level design issues
  - Explaining how to use systems or tools

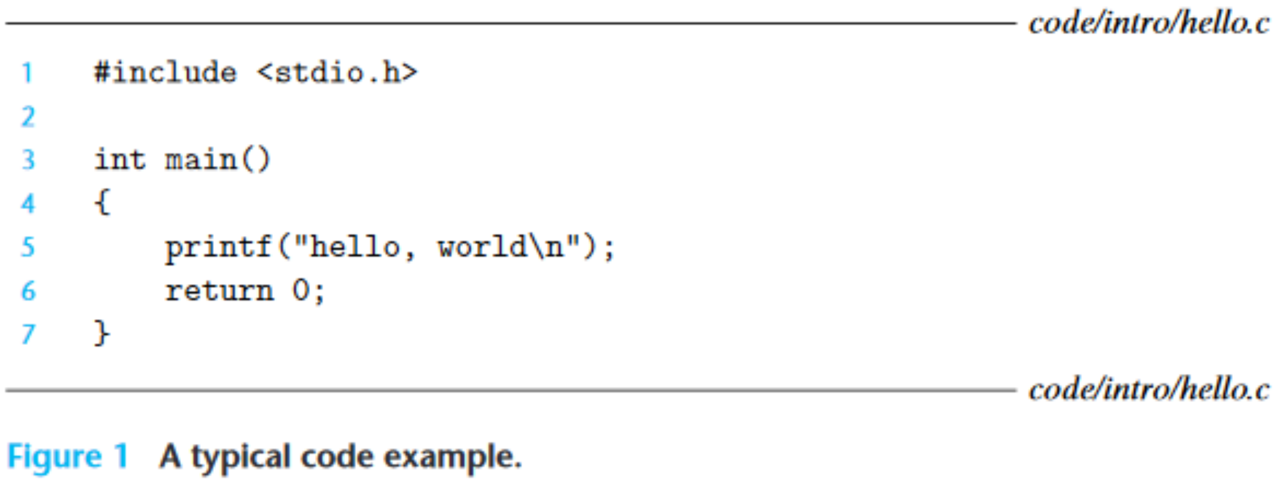
Course Assumption

Required: Basic programming knowledge

- Any kind of programming languages - Java, C/C++, or Python

C Programming Language

- Lecture notes and textbooks provide many useful program samples written in C
- C is the representative language to implement system-layer applications
- For the students who are not familiar with C, we will cover the basics of C in the next lecture



code/intro/hello.c

```c
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

code/intro/hello.c

Figure 1 A typical code example.

Environments for Practices

Recommended environment: Linux

- Sample codes in lecture notes and textbooks are written in Linux

Possible environments

- Install Linux OS: any kind of Linux OS is allowed
  - Most Linux systems are open sourced
  - Example: https://www.ubuntu.com/ (you can download here)
- Install Virtual Machines on Windows
  - **Vmware (recommended)** or Virtual box for Windows
  - You can install Linux on virtual machines in Windows
- Or
  - Cloud service provided by Amazon or Google
  - Any kind of environments to use Linux systems are possible

The easiest recommended way will be provided by manual

Textbooks

- **Randal E. Bryant and David R. O'Hallaron,**
  - *Computer Systems: A Programmer's Perspective*, Third Edition (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

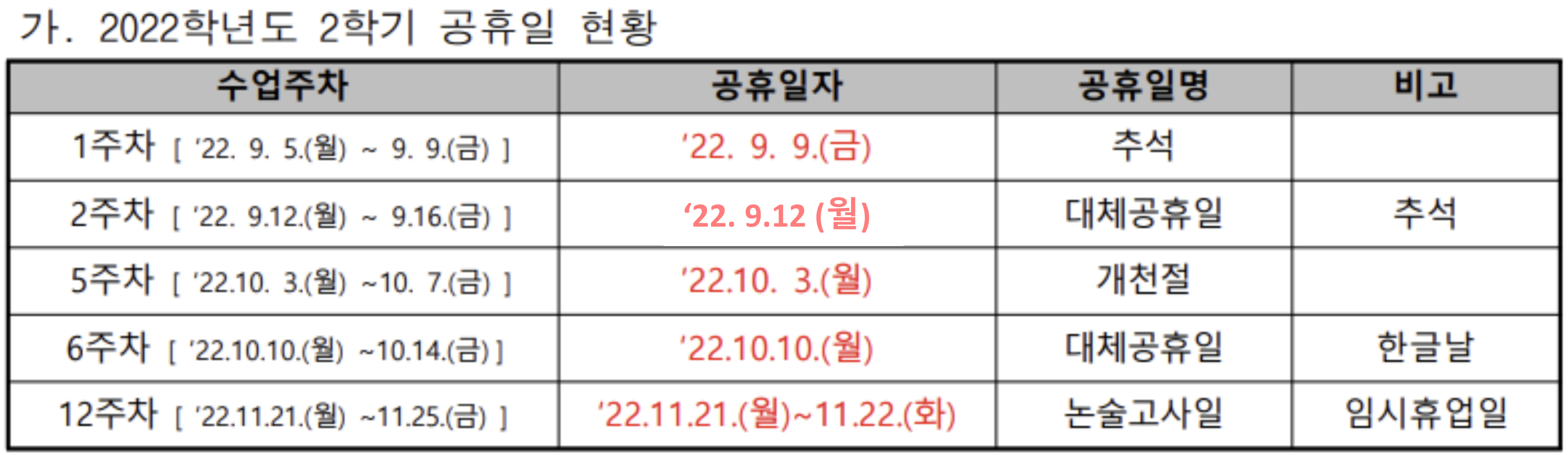
- **Brian Kernighan and Dennis Ritchie,**
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators

Getting Help

- **E-class: http://eclass.seoultech.ac.kr**
  - Complete schedule of lectures, exams, and assignments
  - Lecture slides, assignments, exams, solutions
  - Clarifications to assignments
  - Q&A

- **Office hours**
  - Monday, 2:00-4:00pm, Frontier 614
  - You can schedule 1:1 appointments in advance

Course Schedule



가. 2022학년도 2학기 공휴일 현황

수업주차	공휴일자	공휴일명	비고
1주차 ['22. 9. 5.(월) ~ 9. 9.(금)]	'22. 9. 9.(금)	추석	
2주차 ['22. 9.12.(월) ~ 9.16.(금)]	'22. 9.12 (월)	대체공휴일	추석
5주차 ['22.10. 3.(월) ~10. 7.(금)]	'22.10. 3.(월)	개천절	
6주차 ['22.10.10.(월) ~10.14.(금)]	'22.10.10.(월)	대체공휴일	한글날
12주차 ['22.11.21.(월) ~11.25.(금)]	'22.11.21.(월)~11.22.(화)	논술고사일	임시휴업일

9/12 (추석)

10/3 (개천절)

10/10 (한글날)

Week	Contents	Offline (Flipped Learning)
1	Overview	Introduction of course
2	Introduction of C Programming Languages	No offline lecture
3	Information Storage	Summary for 1st and 2nd lecture
4	Integer/Floating Representation	Summary for 3rd and 4th lecture
5	Machine-Level Programming: Basics	No offline lecture
6	Machine-Level Programming: Controls	No offline lecture
7	Machine-Level Programming: Procedures	Summary for 5th and 6th lecture
8	Mid-Term Exam 1-6 week.	
9	1st Project Presentation	

11/21 (논술)

Week	Contents	Offline (Flipped Learning)
10	Machine-Level Programming: Data	
11	Machine-Level Programming: Advanced	No offline lecture
12	Memory Hierarchy	
13	Cache Memories and Virtual Memories	
14	2nd Project Presentation	
15	Final Exam	

Contents

- **Part1: Course Overview**
  - Course Policies
  - Course Theme
  - Why This Course Important? Five realities!
- **Part2: The Overview of Computer Systems**

Course Theme: Abstraction Is Good But Don’t Forget Reality

Abstraction is important

- Asymptotic analysis
  - If $f(n) = n^2 + 3n$, then as n becomes very large, $3n$ becomes insignificant compared to n^2

The abstraction has limits

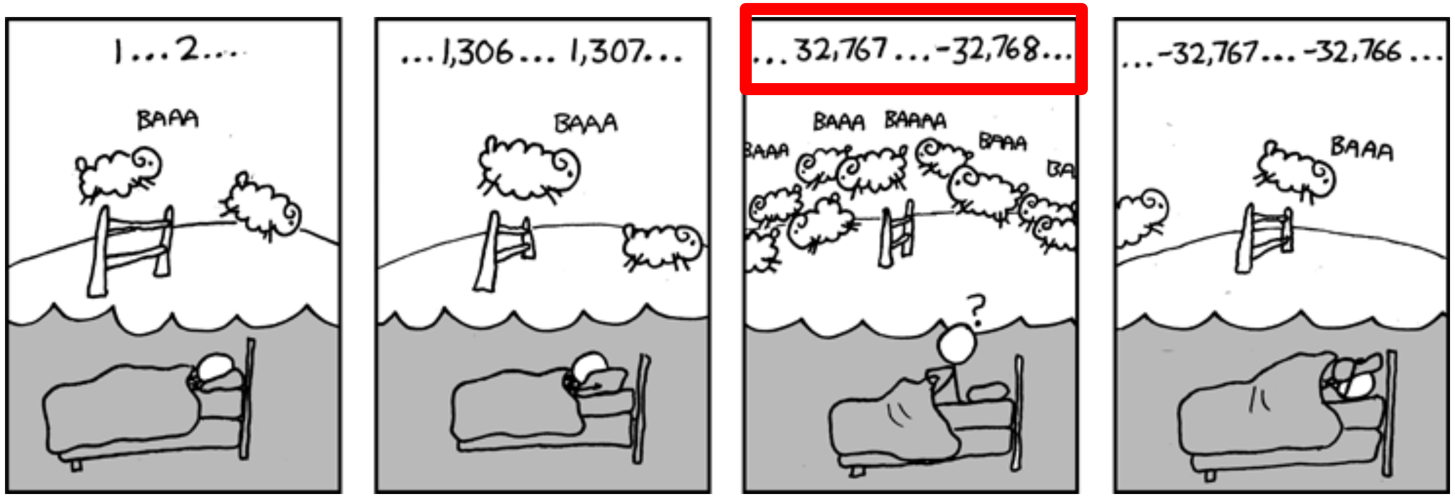
- Especially in the presence of bugs
- Need to understand details of underlying implementations

Useful outcomes from this course

- Become more effective programmers
  - Able to find and eliminate bugs efficiently
  - Able to understand for program performance
- Become more effective IT consultant

Great Reality #1: Ints are not Integers?

- **Example : Is $x^2 \geq 0$?**



Source: xkcd.com/571

```
int a = 40000 * 40000; // 1600000000
int b = 50000 * 50000; // 2500000000 ??
```

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all “usual” mathematical properties**
  - Due to finiteness of representations
- **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

- **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

Great Reality #3: Memory Matters

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing 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
  - 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) {
  struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

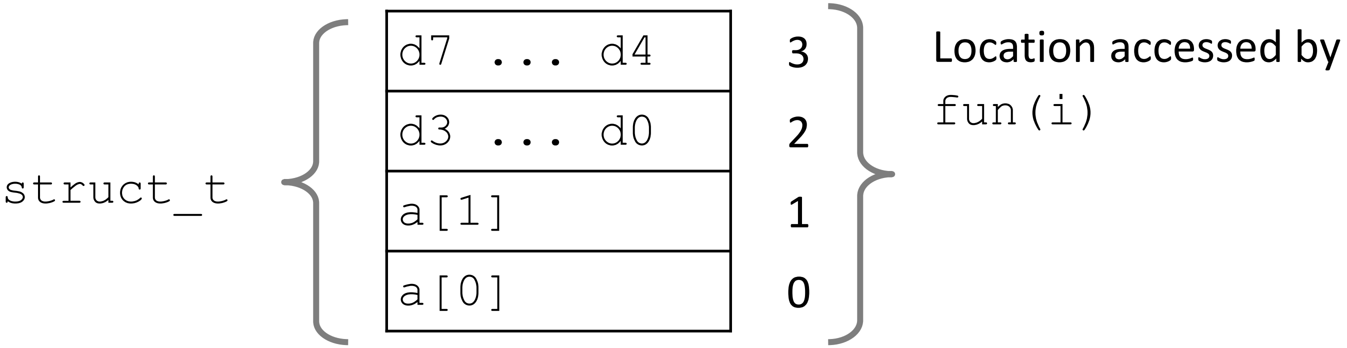
```
fun(0) -->     3.14
fun(1) -->     3.14
fun(2) -->     3.1399998664856
fun(3) -->     2.00000061035156
```

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
```

Explanation:



struct_t

d7 ... d4	3
d3 ... d0	2
a[1]	1
a[0]	0

Location accessed by `fun(i)`

Memory Referencing Errors

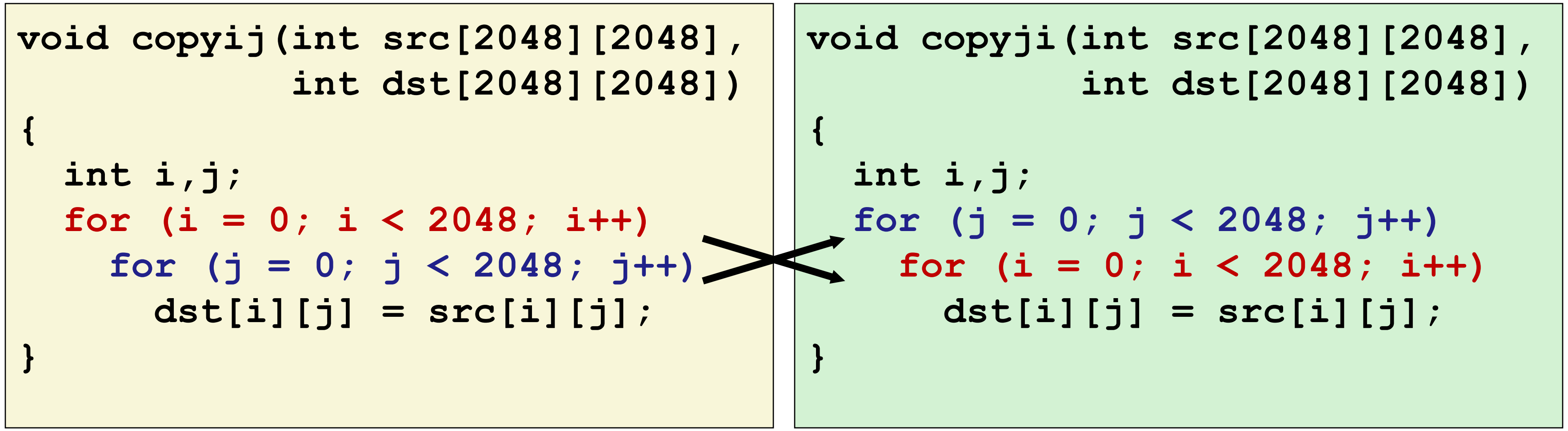
- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated 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!**

- **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



```c
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];
}
```

```c
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];
}
```

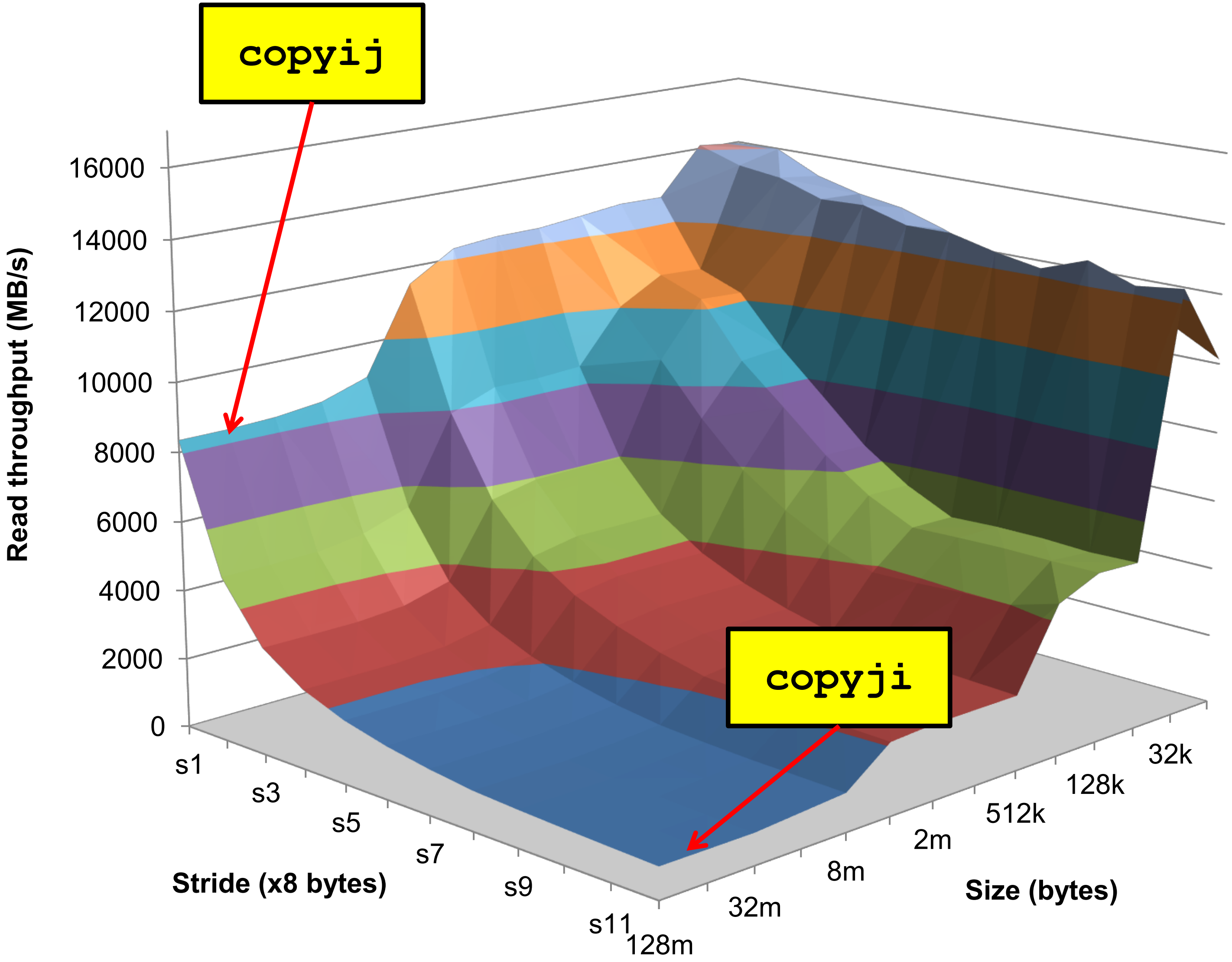
4.3ms

81.8ms

2.0 GHz Intel Core i7 Haswell

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

Why The Performance Differs



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 operations by multiple processes
    - Cross platform compatibility
    - Complex performance issues

Course Perspective

- **Most Systems Courses are Builder-Centric**

- **Our Course is Programmer-Centric**
  - Purpose is to show that by knowing more about the underlying system
  - Enable you to write programs that are more reliable and efficient

Contents

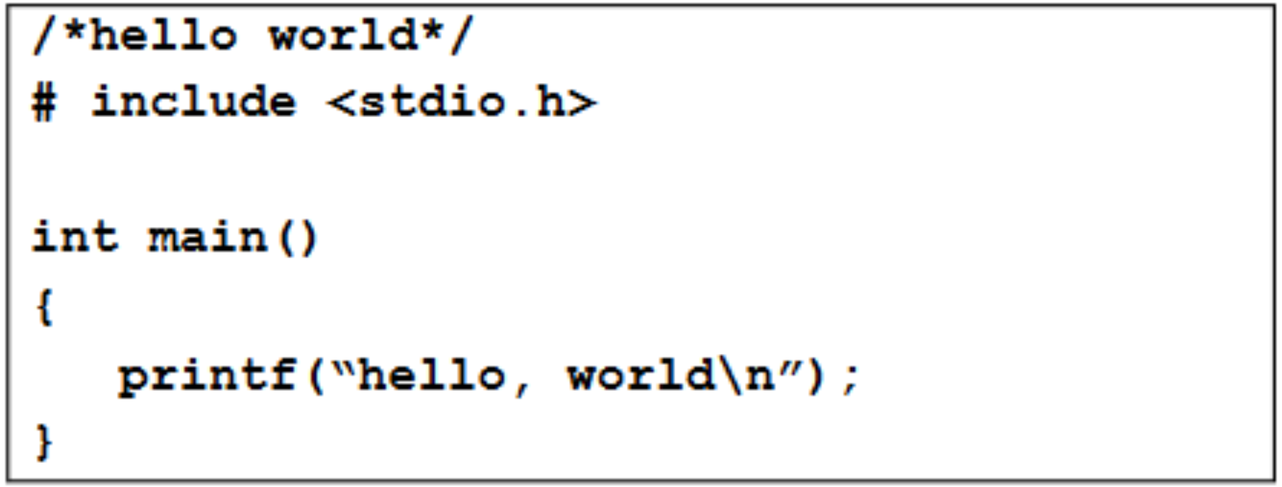
■ Part1: Course Overview

- Course Theme
- Why This Course Important? Five realities!
- Course Perspective and Policies

■ Part2: The Overview of Computer Systems

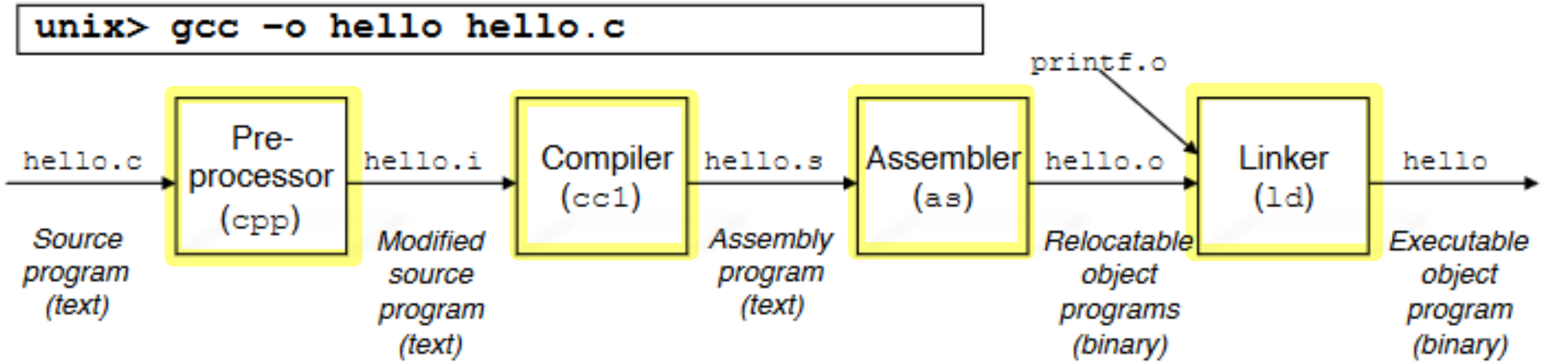
Hello World

- What happens and why when you run “hello” on your system?



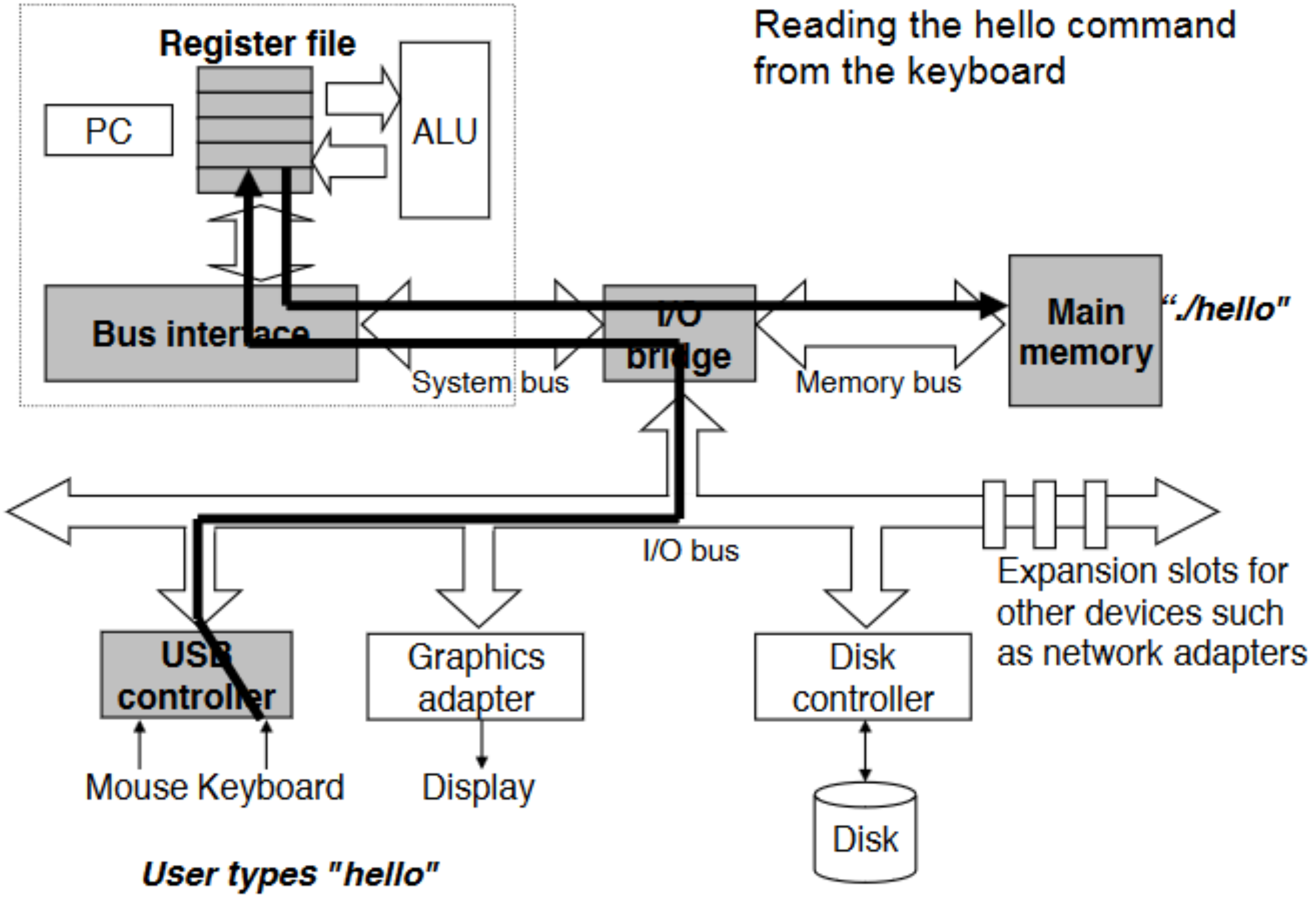
```
/*hello world*/
# include <stdio.h>

int main()
{
   printf(“hello, world\n”);
}
```

Programs translated by other programs



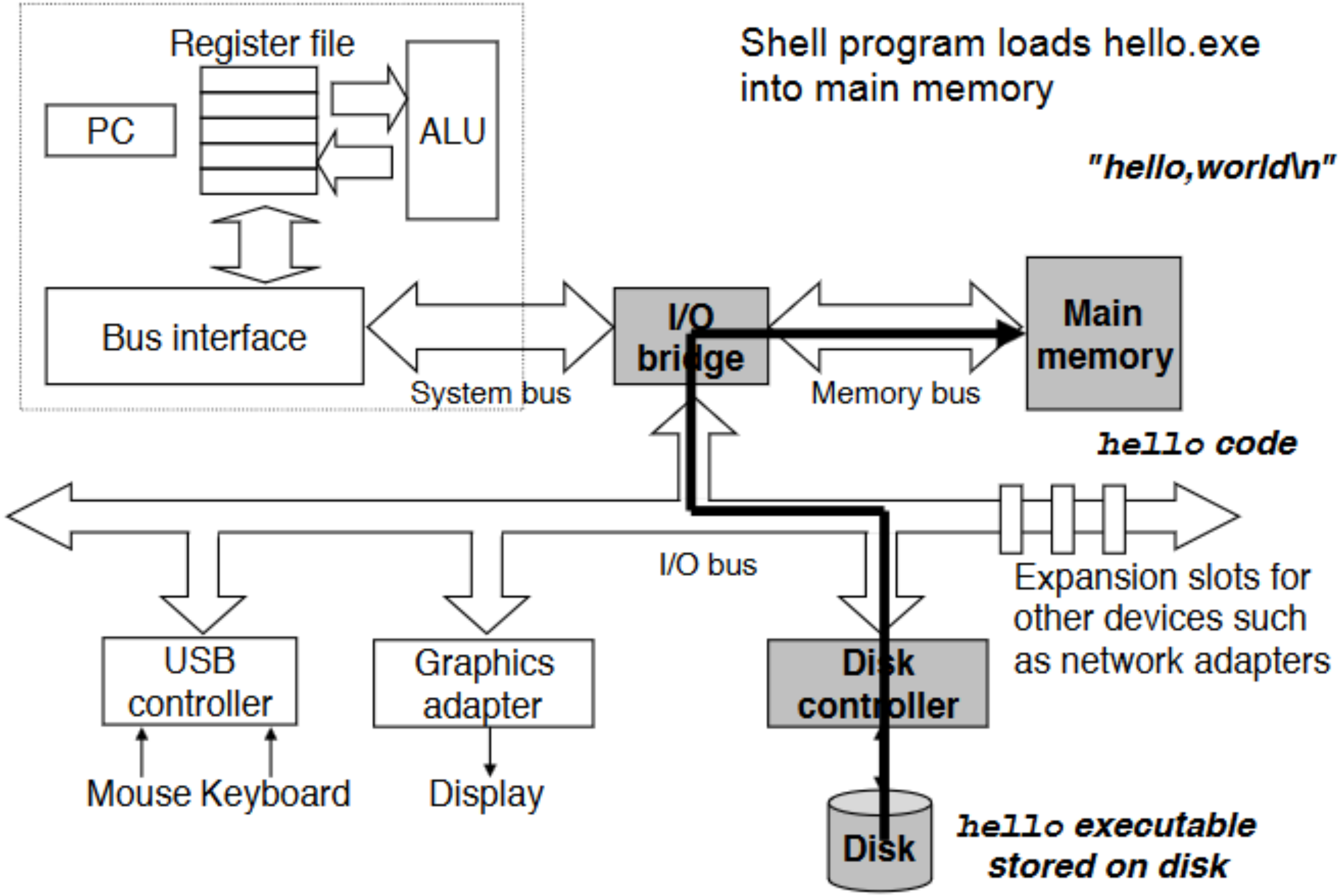
- **Pre-processing**
  - E.g., #include <stdio.h> is inserted into hello.i
- **Compilation (.s)**
  - Each statement is an assembly language program
- **Assembly (.o)**
  - A binary file whose bytes encode machine language instructions
- **Linking**
  - Get printf() which resides in a separate precompiled object file

Running Hello



Reading the hello command from the keyboard

Running Hello



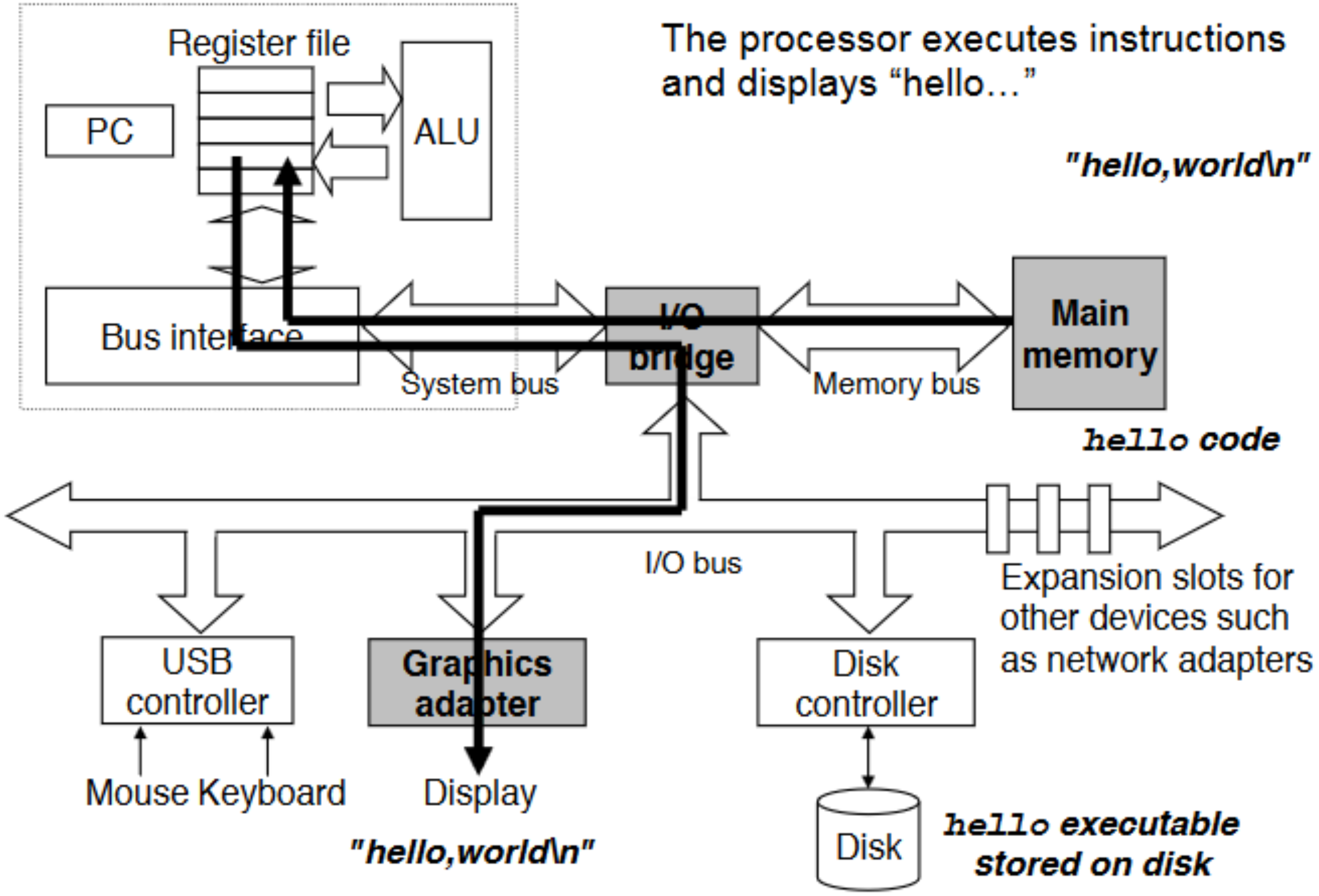
Shell program loads hello.exe into main memory

"hello,world\n"

hello code

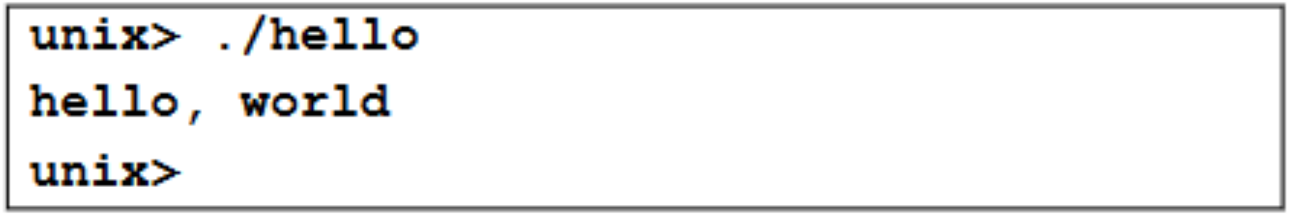
hello executable stored on disk

Running Hello



The processor executes instructions and displays "hello…"

Running Hello

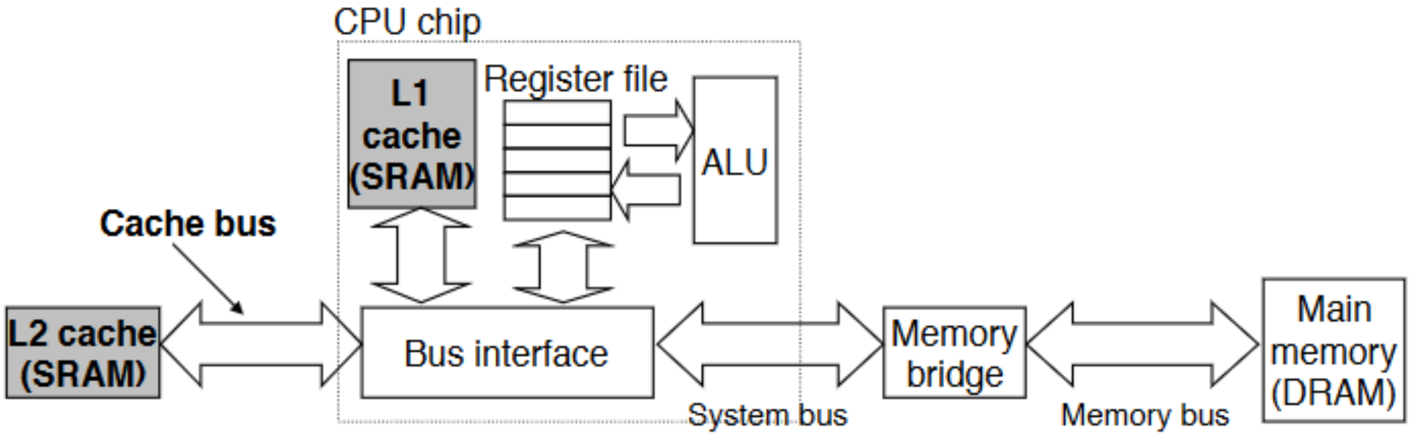


```
unix> ./hello
hello, world
unix>
```

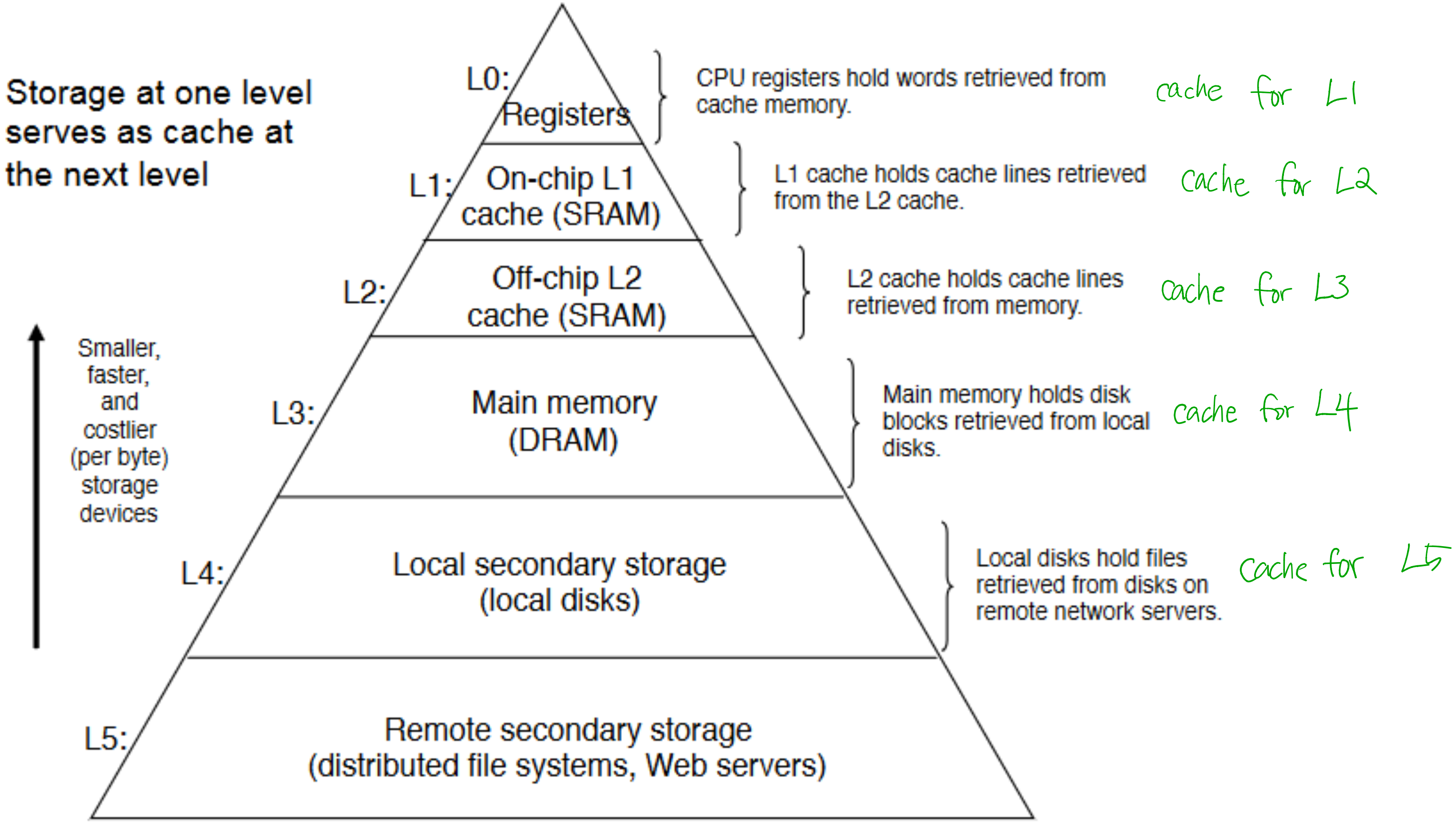
- **What's the shell?**
- **What does it do?**
  - prints a prompt
  - waits for you to type command line
  - loads and runs hello program
  - ...

Cache Matter

- **System spends a lot of time moving info. around**
- **Smaller storage devices are faster than larger ones**
  - Register file ~ 100 Bytes & Main memory ~ millions of Bytes
- **Easier and cheaper to make processors run faster than to make main memory run faster**
  - Standard answer – cache



Storage Devices Form a Hierarchy



Storage at one level serves as cache at the next level

L0:
Registers
L1:
On-chip L1 cache (SRAM)
L2:
Off-chip L2 cache (SRAM)
L3:
Main memory (DRAM)
L4:
Local secondary storage (local disks)
L5:
Remote secondary storage (distributed file systems, Web servers)
Smaller, faster, and costlier (per byte) storage devices

CPU registers hold words retrieved from cache memory. cache for L1

L1 cache holds cache lines retrieved from the L2 cache. cache for L2

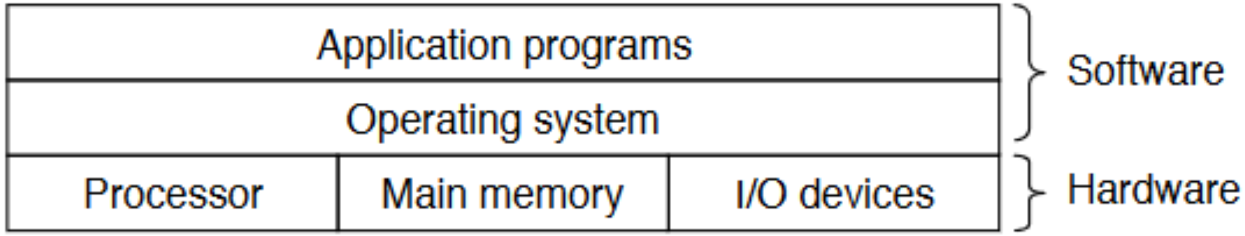
L2 cache holds cache lines retrieved from memory. cache for L3

Main memory holds disk blocks retrieved from local disks. cache for L4

Local disks hold files retrieved from disks on remote network servers. cache for L5

Operating System

- **OS – a layer of software interposed between the application program and the hardware**



Application programs

Operating system

Processor | Main memory | I/O devices

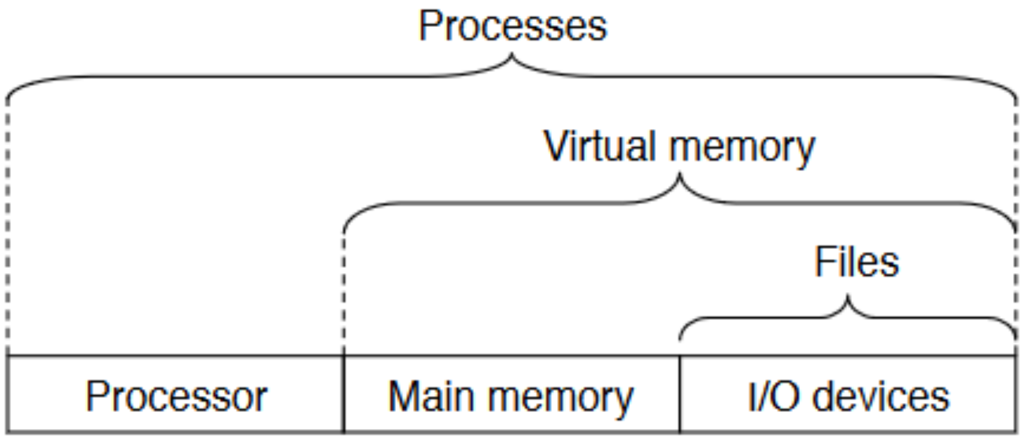
Software

Hardware

- **Two primary goal**
  - Provide simple and uniform mechanisms for manipulating low-level hardware devices
  - Protect resources from misuse by applications

OS Abstractions

- **Files** – abstractions of I/O devices
- **Virtual memory** – abstraction for main memory and I/O devices
- **Processes** – abstractions for processor, main memory, and I/O devices

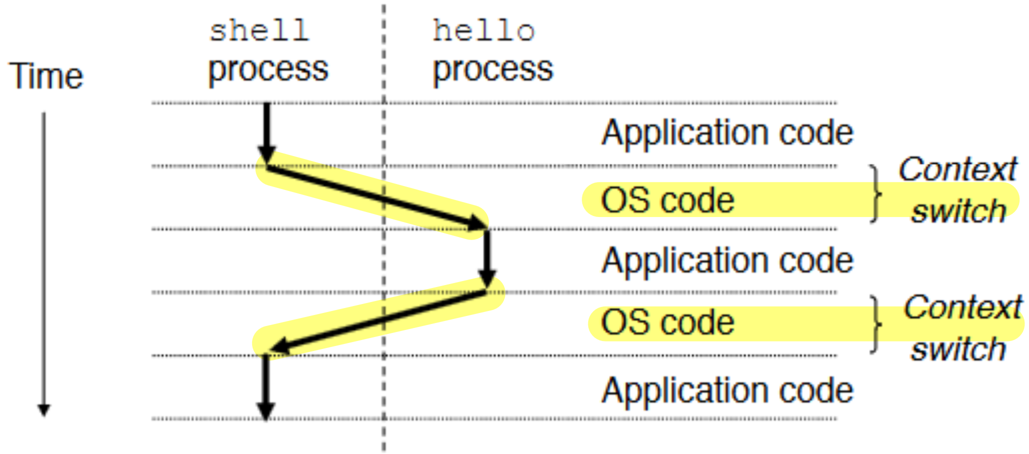


Processes

- **OS provides the illusion of a dedicated machine per process**

- **Process**
  - OS's abstraction of a running program

- **Context switch**
  - Saving context of one process, restoring that of another one
  - Distorted notion of time



Summary of Today’s Lecture

- **A computer system is more than just hardware**
  - A collection of intertwined HW & SF that must cooperate to achieve the end goal – running applications
- **The rest of the course will expand on this**

Welcome and Enjoy!