# Computer Systems Organization
# CS2.201

**Lecture 1**

Based on chapter 1 from Computer Systems by
Randal E. Bryant and David R. O ' H a l l a r

# Introduction

Welcome to CSO Class

Text Book:

Computer Systems A Programmers Perspective (Third Edition) by Randal E. Bryant and David R. O'Hallaron

Please prepare notes during lectures –slides may not all be accessible

# A Rough Grading Structure (some changes)

Assignments: 20%

Quizzes: 10-15%

- Surprise quizzes are a possibility if need arises

Mid Exam: 20%

Final Exam: 30%

Lab exam: 10-12%

Notes, Attendance, Participation: 5-10% (TBD)

Cheating, Attendance and Ethics related issues can incur 100% penalty

# Overview of topics we may cover

Introduction (1)

Computer Arithmetic (2.1 –2.3)

Assembly language programming (3.4 –3.7)

Processor architecture and design (4.1 –4.5)

Memory Hierarchy (6.1 - 6.4)

Optional:

System calls Intro to process control (8.2 –8.4)

Virtual memory high level overview

# Basic Course Goal

Course Goal: To study the anatomy of a typical Computer System.

Well, what is a typical computer system?

- Desktops, Laptops, Notebooks

How about Server Machines? In what way they are different from Desktops/Laptops?

- Desktops run a user-friendly **OS** and desktop applications to facilitate desktop-oriented tasks.
- Server manages all network resources and are often dedicated (performs no other task besides server tasks). Servers are engineered to manage, store, send and process data 24-hours a day, has to be more reliable than a desktop computer and offers a variety of features/hardware.

# Basic Course Goal

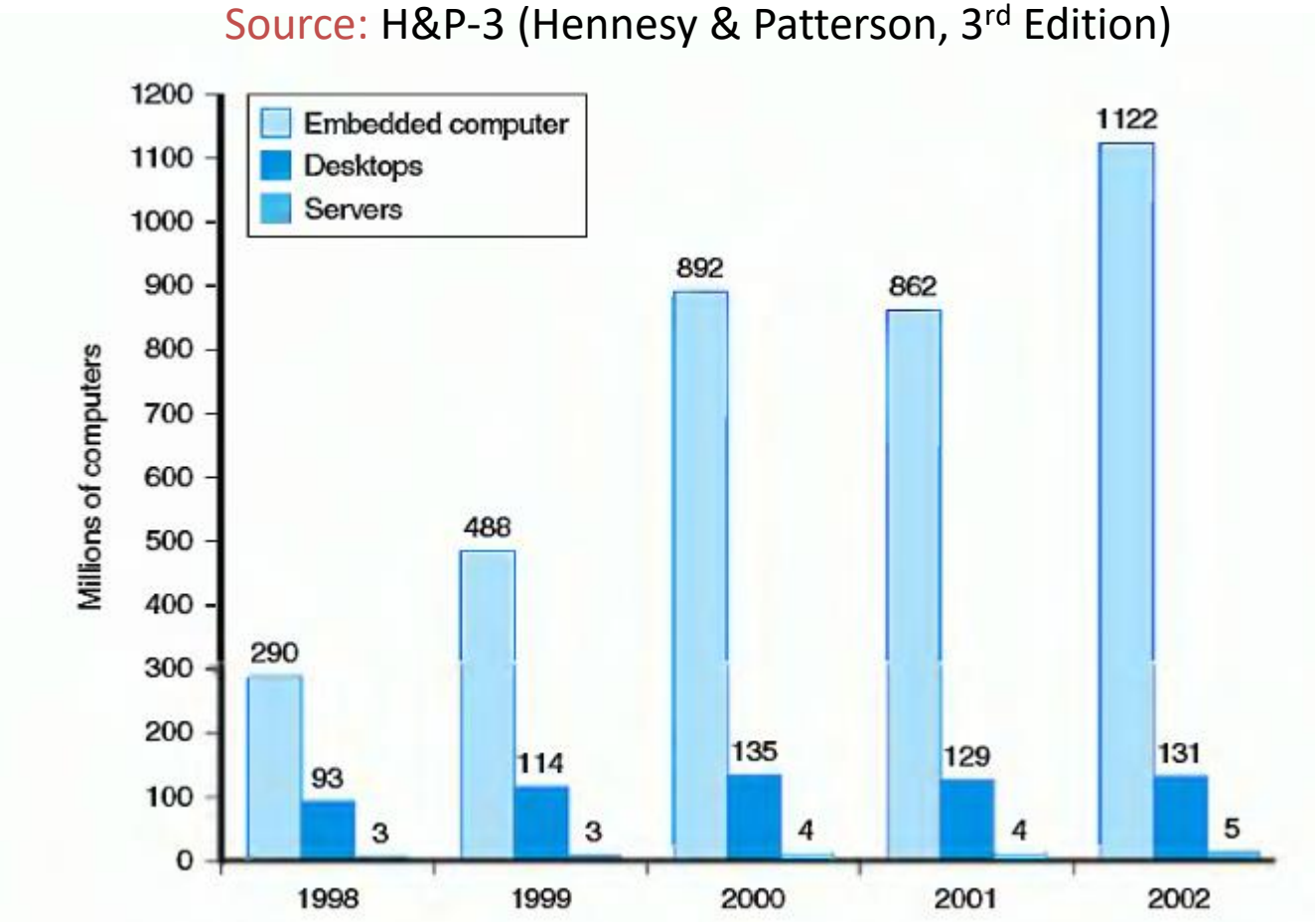
Operating System (OS): OS is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers. It is an interface between a computer user and computer hardware.

How about Embedded Computers lying inside Cell Phones, Automobiles, Airplanes, Set Top Boxes, Televisions etc. ?

A general purpose computer system can be programmed to perform a large number of tasks. Embedded systems are designed to perform a small number of tasks efficiently.

# Sales Distribution

Source: H&P-3 (Hennesy & Patterson, 3rd Edition)




Number of distinct processors sold between 1998 and 2002.

# The hello world program

- Source program or source file
- Organized as 8 bit chunks called bytes
- ASCII (American Standard Code for Information Interchange) representation below
  - Files that exclusively containing ASCII chars are called text files - others called binary files

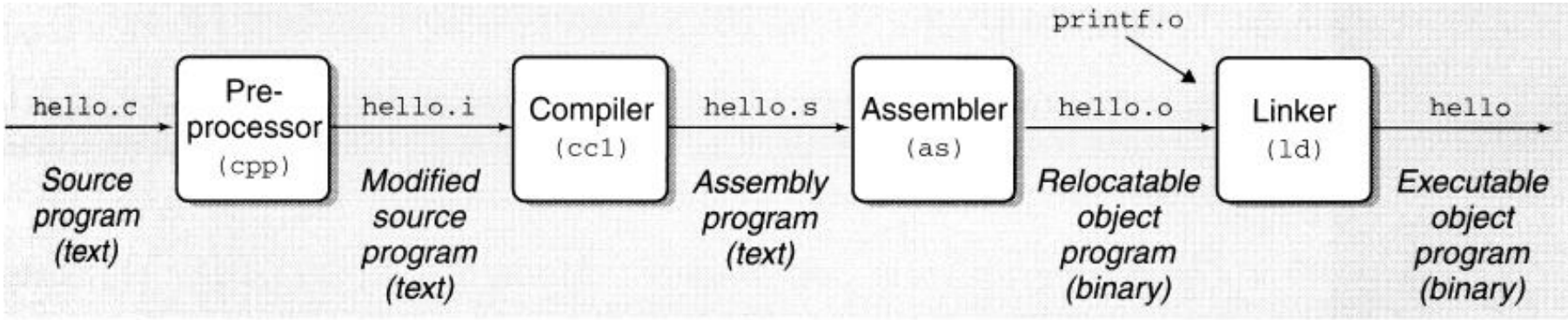
| # | i | n | c | l | u | d | e | <sp> | < | s | t | d | i | o | . |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 35 | 105 | 110 | 99 | 108 | 117 | 100 | 101 | 32 | 60 | 115 | 116 | 100 | 105 | 111 | 46 |

| 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 |

# Typical Compilation Sequence




Steps during compilation of the program

- Preprocessing phase
- Compilation phase
- Assembly phase
- Linking phase

# Typical Compilation Sequence

Preprocessing phase: Modifies original C program according to directives of # character

- Read contents of stdio.h and insert into program text. Results in another program with .i affix

Compilation phase: Translates .i into .s containing assembly language program

- Useful since it provides a common output language for different compilers for different high level languages e.g., C and Fortran compilers generate files in same assembly language

Assembly phase: Translates .s into machine language instructions –packages into relocatable object program and outputs .o.

# Typical Compilation Sequence

Linking Phase: printf function resides in a separate precompiled object file called print.o which needs to be merged into hello.o. Linker handles the merging and creates hello [an executable file ready to be loaded into memory and executed by the system]

Understanding compilation systems helps with

- Optimizing program performance e.g., switch vs. if-else, overhead incurred by function call etc.
- Understanding link-time errors which are hard to catch e.g., linker says cannot resolve a reference, difference between stati c a n d g l o b a
- Avoiding security holes e.g., buffer overflow v u l n e r a b i l i t i e s ...

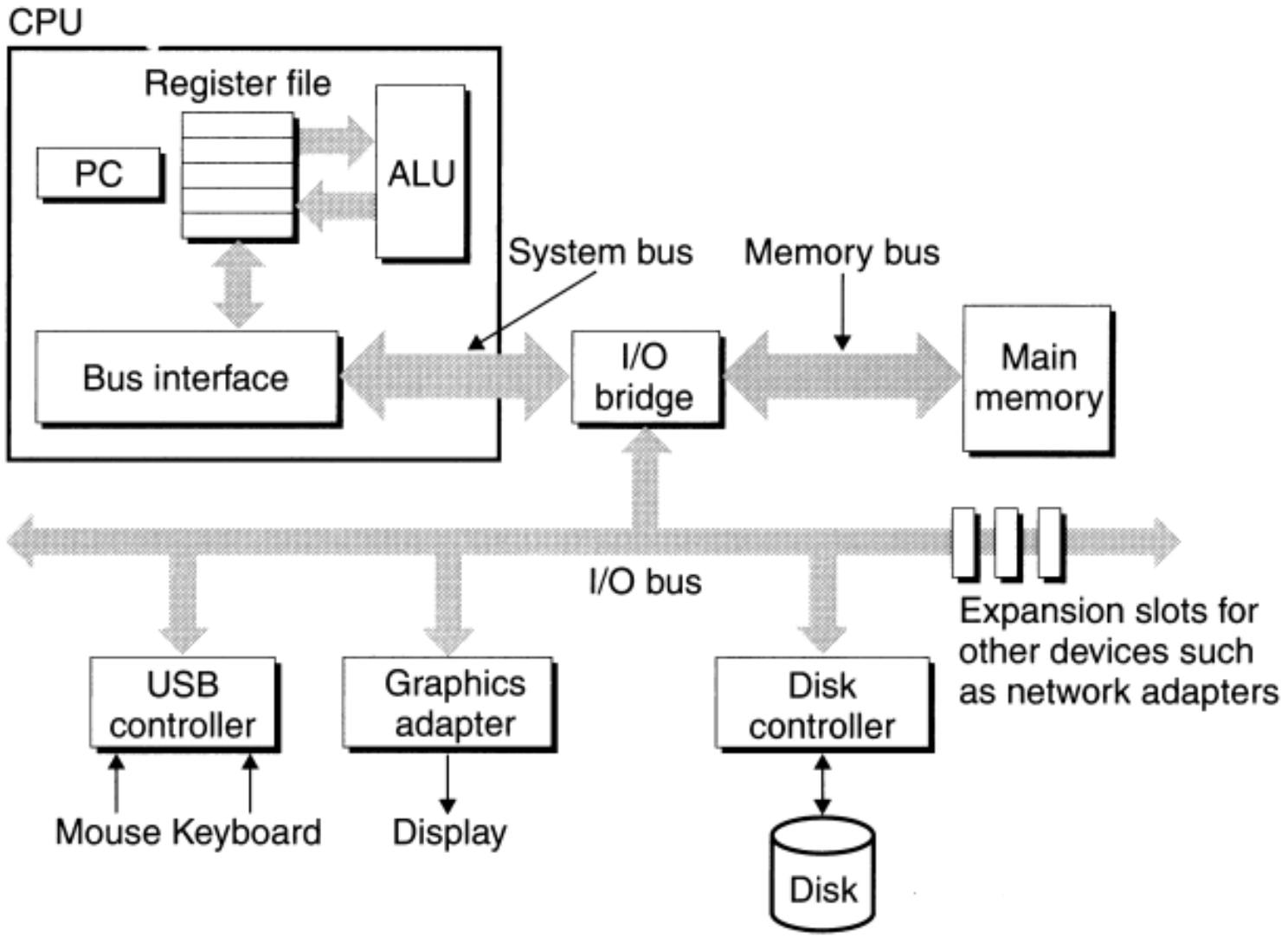
# Buffer Overflow Vulnerability

Better understanding helps to avoid issues

Code below uses gets() function to read an arbitrary amount of data into a stack buffer. Because there is no way to limit the amount of data read by this function, safety of the code depends on the user to always enter fewer than BUFSIZE characters.

```
...
char buf[BUFSIZE];
gets(buf);
...
```

# Typical Hardware Organization of a System




Source: RB&DO -1 (Randal E. Bryant & David O'Hallaron, 1st Ed)

# Hardware Organization of System

Buses: Collection of electric conduits that carry bytes of information between components. Carry fixed size chunks of bytes called words (called word size) e.g., 4 bytes or 8 bytes word

I/O devices: Systems connection to the external world e.g., keyboard, mouse, display and disk drive/disk

- I/O devices connected to bus by either a controller (chipset in the device itself or on motherboard) or adapter (card that plugs into a slot on motherboard)

# Hardware Organization of System

Processor repeatedly executes instruction pointed by the program counter and updates the PC to point to next instruction (which may not be in next memory address)

Processor operates according to a very simple instruction execution model, defined by its instruction set architecture

Few simple operations that revolve around main memory, the register file and the arithmetic/logic unit (ALU).

Register file is a small storage device that consist of a collection of word size registers each with its own unique name (Accumulator, Memory Address Register, Memory Data Register ... for fast

ALU computes new data and address values

# Hardware Organization of System

Main memory: Temporary storage device that holds both a program and data it manipulates

- Collection of DRAM (Dynamic Random Access Memory) chips
- Memory organized as linear array of bytes, each with its own unique address starting at 0

Processor: CPU is the engine that interprets (or executes) instructions stored in main memory. At its core is a word size storage device (or register) called the program counter (PC).

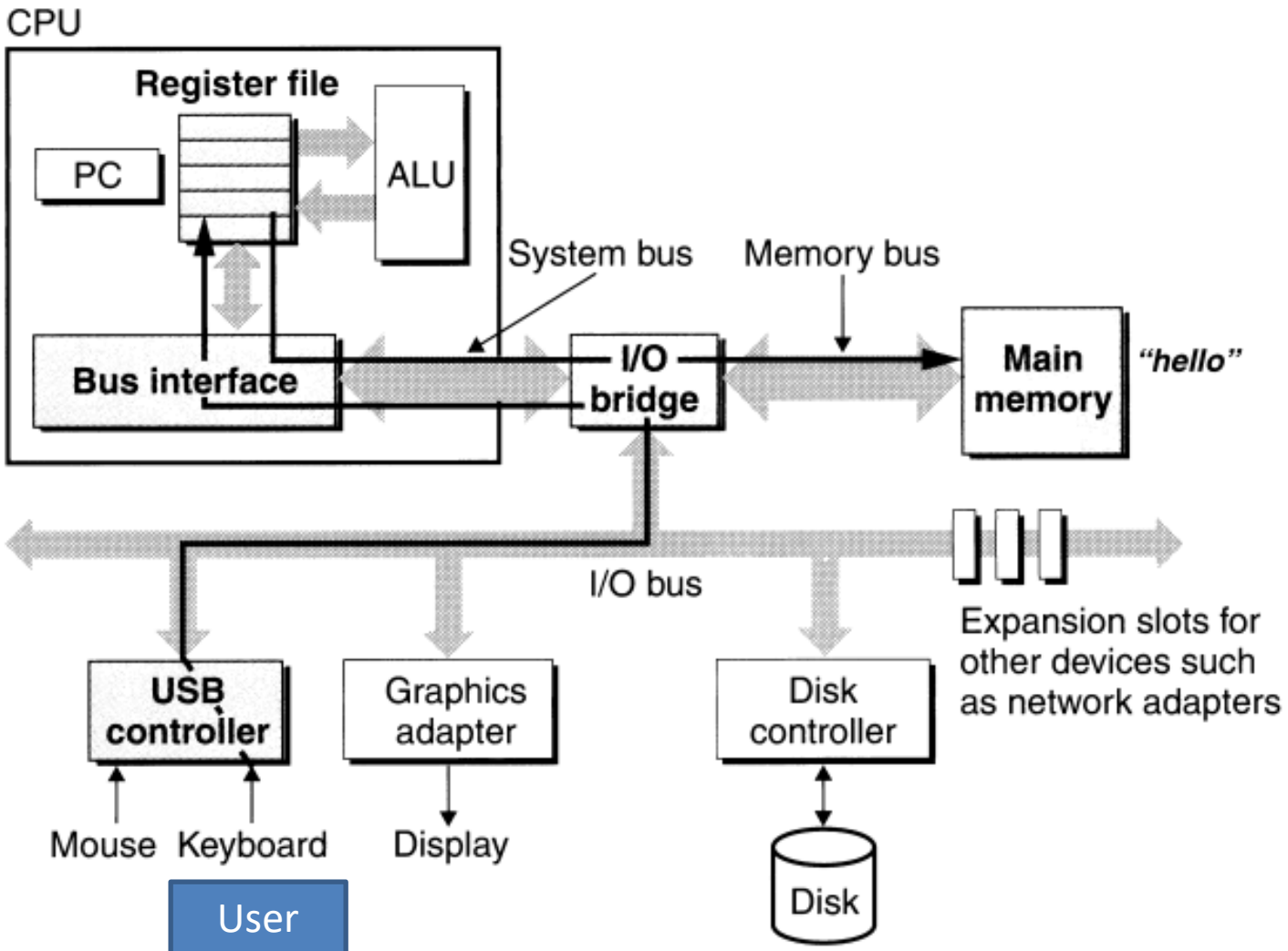
- At any point, PC points to or contains address of some machine language instruction in main memory

# Hardware Organization of System

Some operations carried out by CPU:

- Load: Copy a byte or a word from main memory into a register, overwriting previous contents of the register
- Store: Copy a byte or word from a register to a location in main memory, overwriting the previous contents of that location
- Operate: Copy the contents of two registers to the ALU, perform an arithmetic operation on the two words and store the result in a register, overwriting its previous contents
- Jump: Extract a word from the instruction set itself and copy the word into the PC, overwriting the previous value of the PC

# Running the "Hell



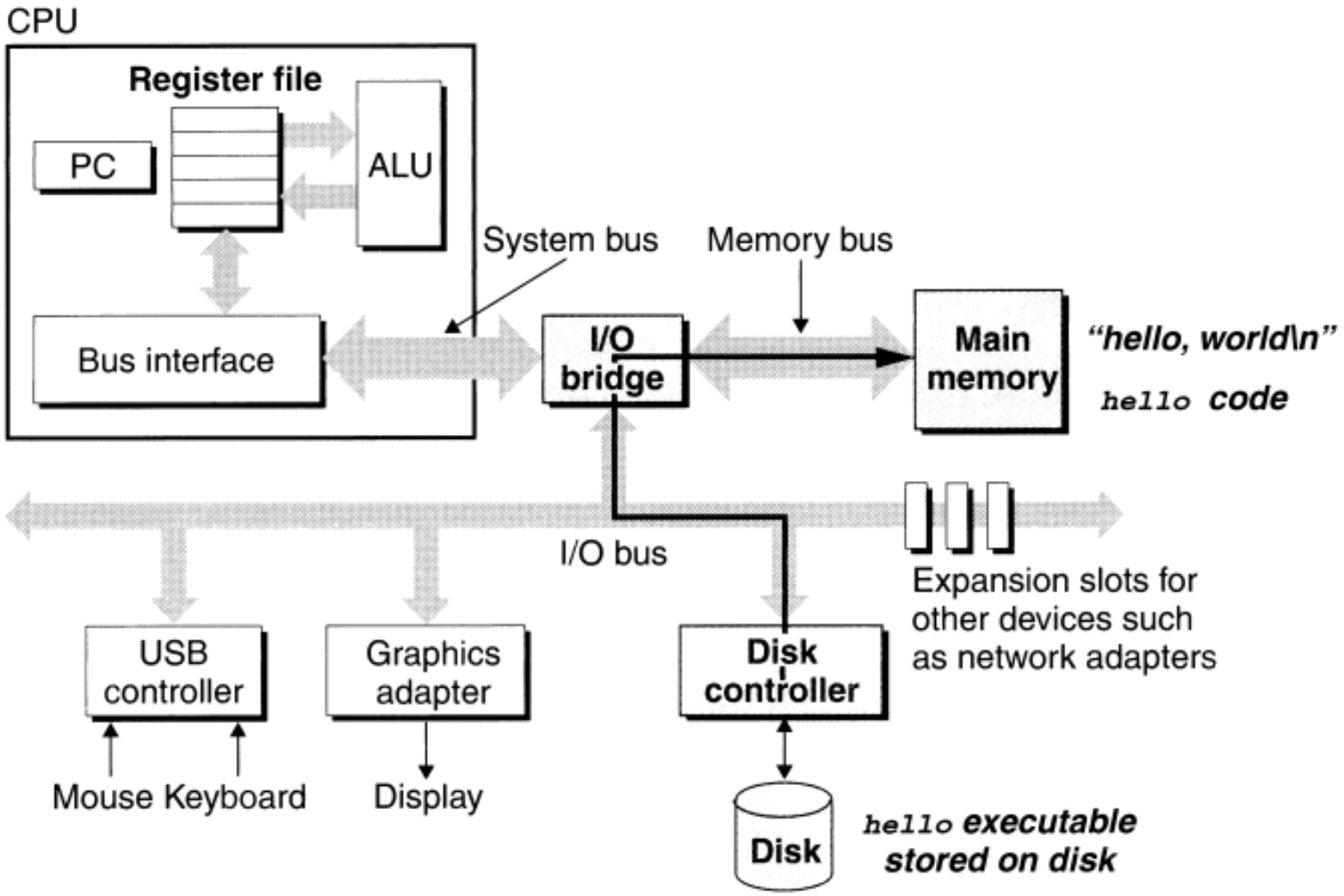

Source: RB&DO-1

# Sequence of steps

Shell program waiting for inputs

As we type the characters ./hello, shell program reads each one into a register and then stores it in memory. Once enter key is hit, shell knows we have finished typing.

# Running the "Hell

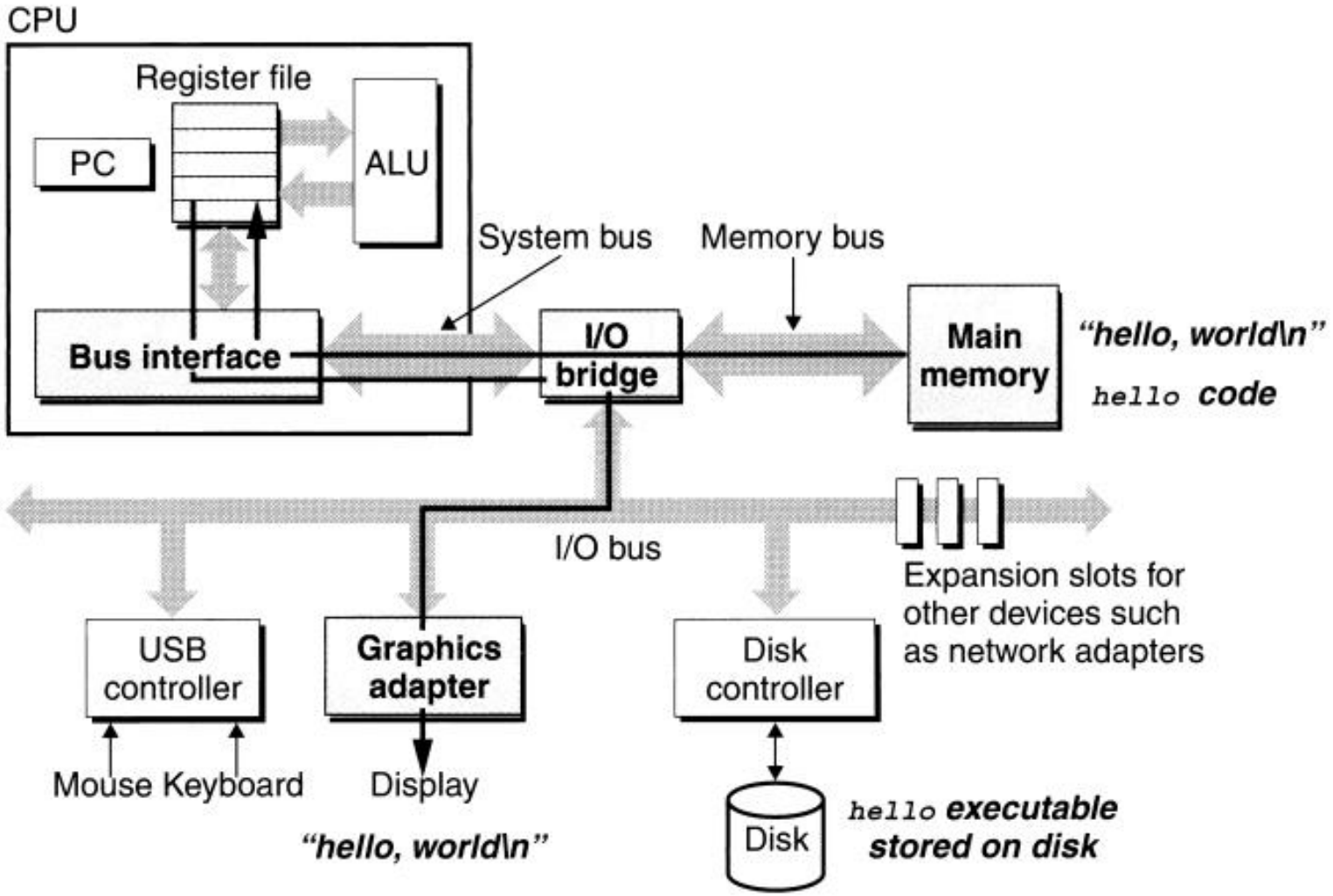



Source: RB&DO-1

# Sequence of steps

Shell then loads the executable hello file by executing a sequence of instructions that copies the code and data in hello object file from disk to main memory

# R u n n i n g  t h e  " H e l l




Source: RB&DO-1

# Sequence of steps

Once loaded in memory, CPU begins executing the machine language instructions in hello program's main routin the bytes in the hello, world\n string from memory to the register file and from there to the display device.

# Cache Memory

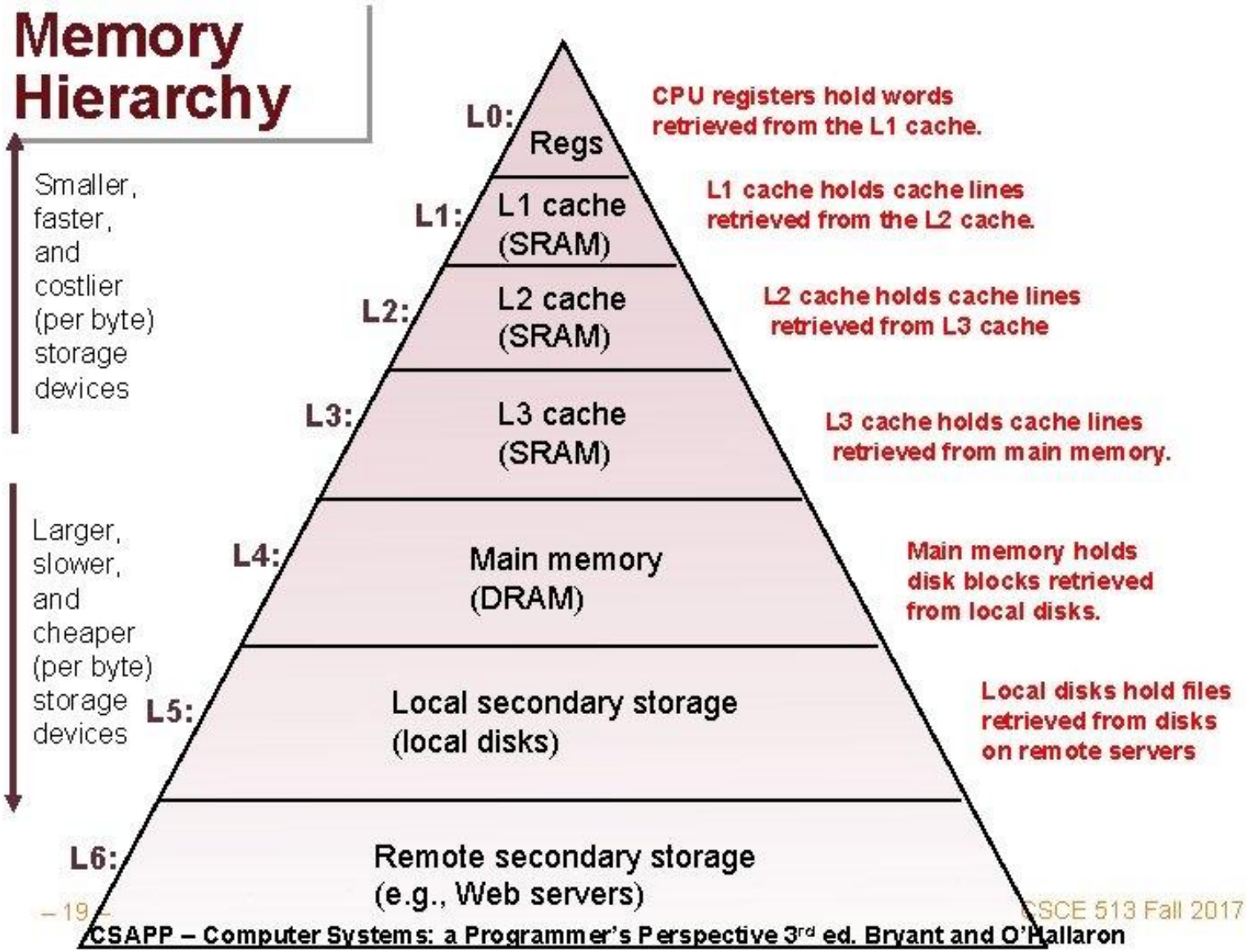
Disk drive is typically 1000 times larger than main memory but it takes processor 10,000,000 times longer to read a word from disk than from memory

Similarly, register file stores only a few 100 bytes of information - however processor can read data 100 times faster than from memory

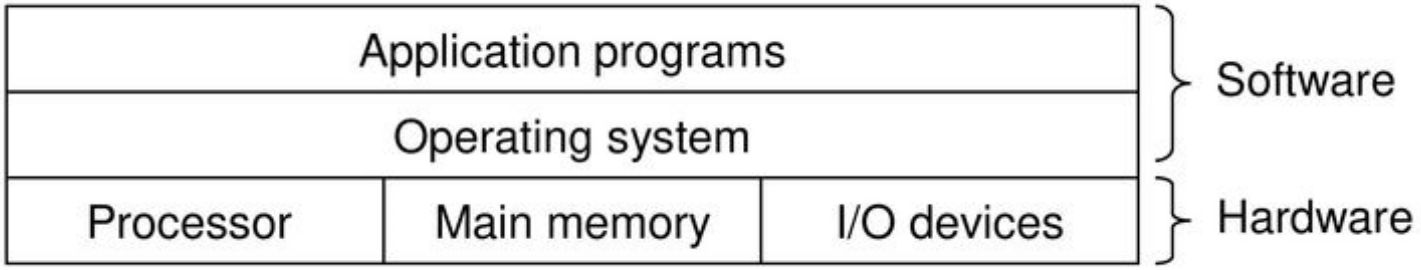
As semiconductor technology advances, the processor-memory gap continues to increase, hence cache memories become important

Application programmers can exploit cache memories to improve performance by an order of magnitude

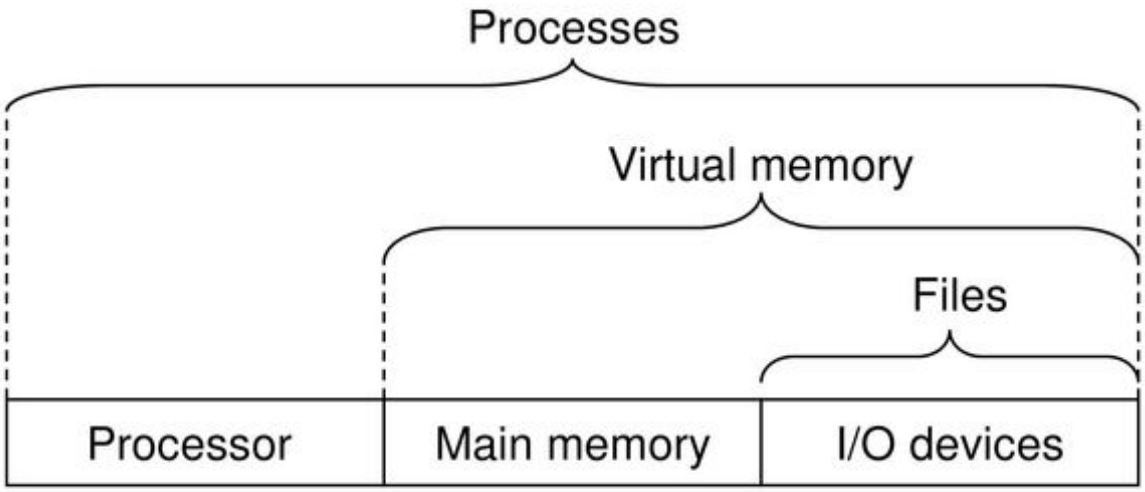
# Memory hierarchy



# OS




**Layered view of a computer system**




**Abstractions provided by an OS**

# OS

Layer interposed between application program and the h/w. 2 primary purposes:

- Protect h/w from misuse by any application
- Provide applications with simple and uniform mechanisms for manipulating complicated and wildly different low level h/w devices

Achieves the goals via fundamental abstractions namely processes, virtual memory and files. Each are abstractions for:

- Files for I/O devices
- Virtual memory for main memory and disk I/O devices
- Processes for the processor, main memory, I/O devices

# Processes

Process is OS's abstrac

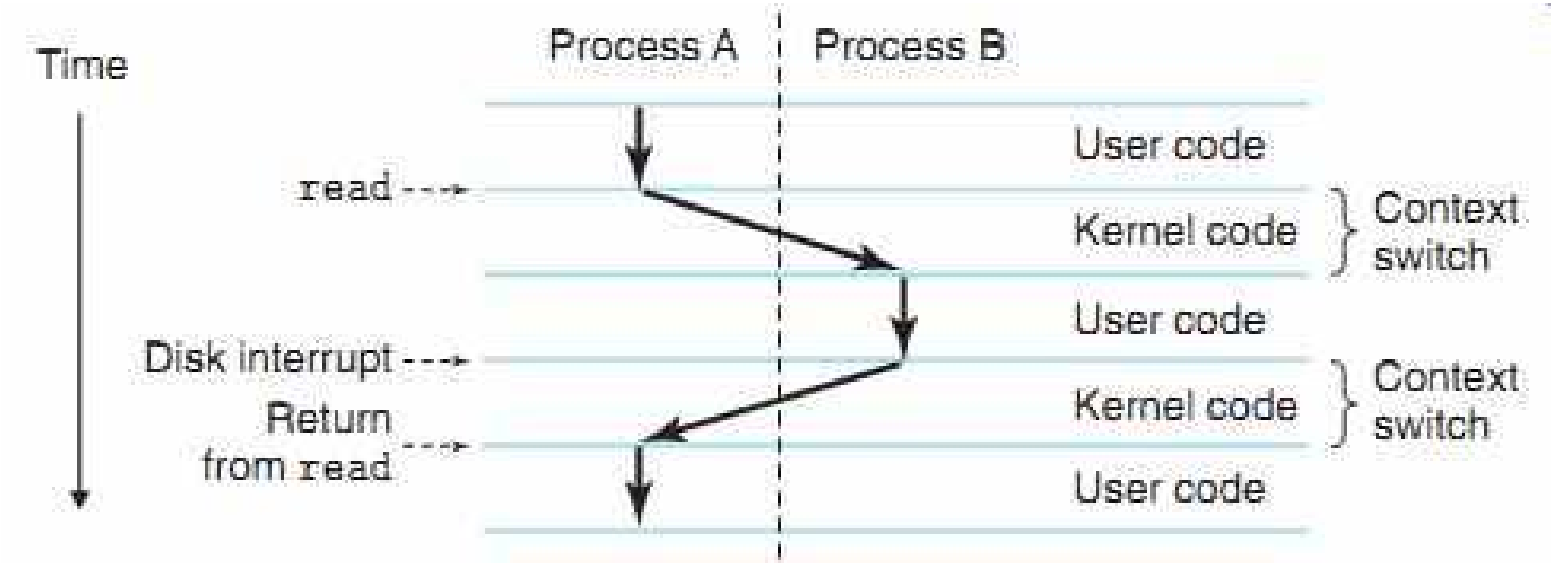
Provides illusion that it is the only program running on the system

Multiple processes can run concurrently i.e., instruction of one process can be interleaved with another process –achieved via context switching

State information of process is called context (includes values of the PC, register file and contents in main memory).

OS performs context switch by saving the context of current process, restoring the context of new process and then passing control to the new process.

# Processes




When hello is run, invokes special function called system call to pass control to OS

OS saves shell's contex

Transition between processes is handled by OS Kernel (portion of OS code always resident in memory)

# Threads

A process can consist of multiple execution units called threads (run in context of the process and share the same code and global data)

Can allow concurrency in processes, easier to share data and more efficient than multiple processes

Threads provide a way to improve application performance (e.g., network or web servers) through parallelism and represent a software approach to improving performance

# Virtual Memory

Virtual memory provides each process with illusion that it has exclusive use of main memory

Each process has same uniform view of memory called virtual address space

Virtual address space seen by each process consists of a number of well defined areas each with specific purposes

- Program code and data, Heap, Shared libraries, Stack etc.

# Files

Sequence of bytes

Every I/O device incl. disks, keyboards, displays and networks modeled as a file

All I/O in system performed by reading/writing files

Provides a uniform view of the varied devices

- Will therefore run on different systems

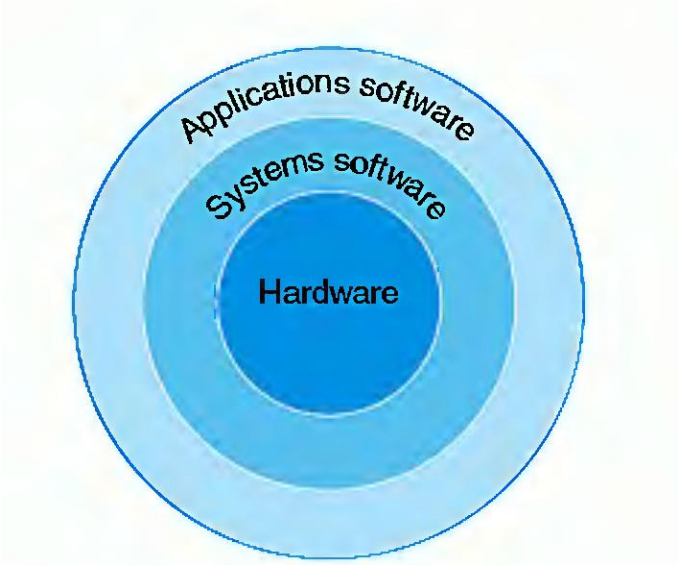
# Networking related

Modern systems linked to other systems by networks

Network can be viewed as another I/O device from/to which data is copied/sent e.g., email, messaging, World Wide

# Computer System = Hardware + System Software + Application Software

Source: H&P-3 (Hennesy & Patterson, 3rd Edition)




System Software: Operating System, Device Drivers, Loaders, Linkers, Compilers, Assemblers, … .

Application Software: Web browsers, user-specific ap

# Amdahl's law

Captures effectiveness of improving one part of the system –when we speedup one part of the system effect on overall system depends on how significant this part was

Suppose an application needs time T_old. One part needs $\alpha$ of this time (i.e., $\alpha$*T_old) and we improve performance by factor k.

T_new = (1- $\alpha$)*T_old + ( $\alpha$*T_old)/k

Speedup s = T_old/T_new = 1/((1- $\alpha$)+ $\alpha$/k)

If $\alpha$= .6 and we sped up the component 3 times, s = 1/((1-.6) + .6/3) = 1.67

Insight: Need to speed up large fraction of system

# Other details

- Concurrency is concept of system with multiple, simultaneous activities and parallelism refers to use of concurrency to make a system run faster
  - Used/implemented in different ways for significant performance improvements  e.g., multithreading, multiprocessing etc.
- Abstraction is one of the important concept in computer science
  - Different programming languages provide different levels of abstraction and support
  - Instruction set architecture provides abstraction of actual processor hardware [machine code behaves as if it were executed on a processor that performs just one instruction at a time while underlying h/w is far more elaborate, executing multiple instructions in parallel]

# Other details

On OS side, files are an abstraction for I/O devices, virtual memory for program memory and processes as abstraction for running program.

A virtual machine provides abstraction of an entire computer including the OS, processor and the programs.