Computer Systems Principles

Wrapup Overview

UMASSCS

SCHOOL OF COMPUTER SCIENCE

Computer Systems

- **Hardware**
  - Central Processing Unit (CPU)
  - Memory
  - Input/Output (I/O) devices
- **Software**
  - Programming Languages and Tools
  - Operating System
  - System Software
  - User Applications

i>clicker Question

- **How are files represented by the Unix operating system at the lowest level?**

  a) Sequence of characters
  b) Sequence of bytes
  c) Sequence of 32-bit integers
  d) Sequence of base-10 digits
  e) None of these

i>clicker Question

- **How are files represented by the Unix operating system at the lowest level?**

a) Sequence of characters

b) Sequence of bytes

c) Sequence of 32-bit integers

d) Sequence of base-10 digits

e) None of these

i-clicker Question

- **How are characters represented by the Unix operating system?**

a) Each character is a 32-bit integer
b) Each character is a byte
c) Each character is contained in a string
d) Each character is a nibble
e) None of these

i-clicker Question

- **How are characters represented by the Unix operating system?**

a) Each character is a 32-bit integer

b) Each character is a byte

c) Each character is contained in a string

d) Each character is a nibble

e) None of these

The **hello** Program

```
#include <stdio.h>

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

Information Is Bits

- **Source File**
  - The hello program begins life as a source program
  - Most programs consist of multiple source files

- **Representation**
  - A sequence of *bits* with a value of 0 or 1
  - Organized into *bytes* of 8 bits each
  - Each byte *represents* a character in the program

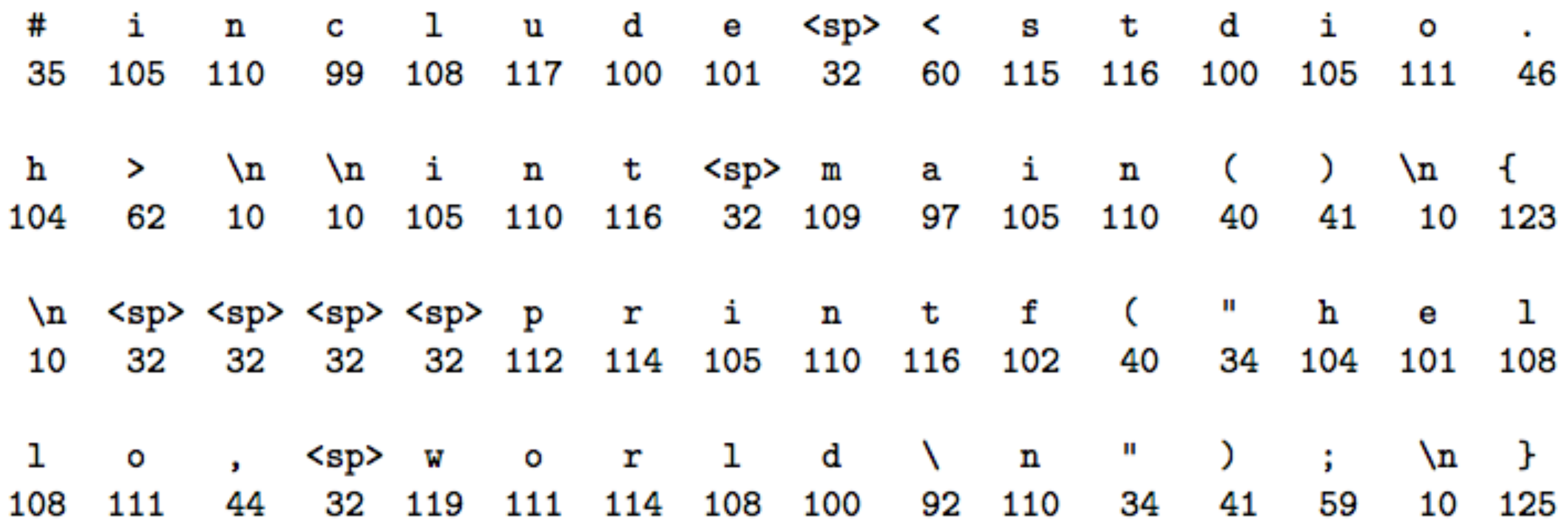
Character Interpretation

- **Bytes and Characters**
  - – Characters are encoded in bytes
  - – Bytes are 8 bits
  - – A bit is a 1 or a 0
- **Bytes and Numbers**
  - – A byte can represent a number in base-2: $00000110_2 = 6_{10}$, $00100000_2 = 64_{10}$, $10000111_2 = 135_{10}$, $11111111_2 = 255_{10}$
  - – Numbers can be used to *represent* characters

Representing Characters

- **ASCII Standard** (ASCII = American Standard Code for Information Interchange)
  - Represents each character with a unique byte-sized integer value

The ASCII text representation of **hello** source file



#	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

ASCII Chart

- **A link to it from this week’s material**
  - Go check it out!
  - Find how the digits ‘0’ through ‘9’ are arranged
  - Find how the upper case letters ‘A’ through ‘Z’ are arranged
  - How about the lower case letters ‘a’ through ‘z’?

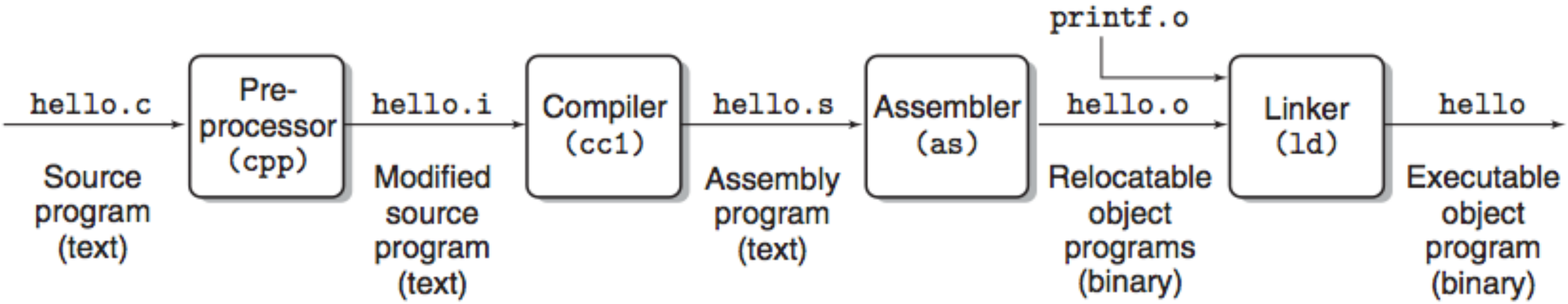
Unicode

- **ASCII has limitations (only 128 characters).**
- **Unicode is an extension of ASCII.**
- **Unicode characters can be stored in 32 bits, but there are representations of them that use fewer bits.**
- **Java uses Unicode, though Linux does not.**

Program Translation

- **Program Source Files**
  - Beginning of life for a C program
  - Represented as ASCII character text
  - “Easy” for humans to understand
  - Not understood by machines
- **Executable Object File**
  - Low-level primitive machine operations
  - Understood by the machine
- **Program Translation**
  - Translates source file into object (machine code) file!
  - Also known as compilation

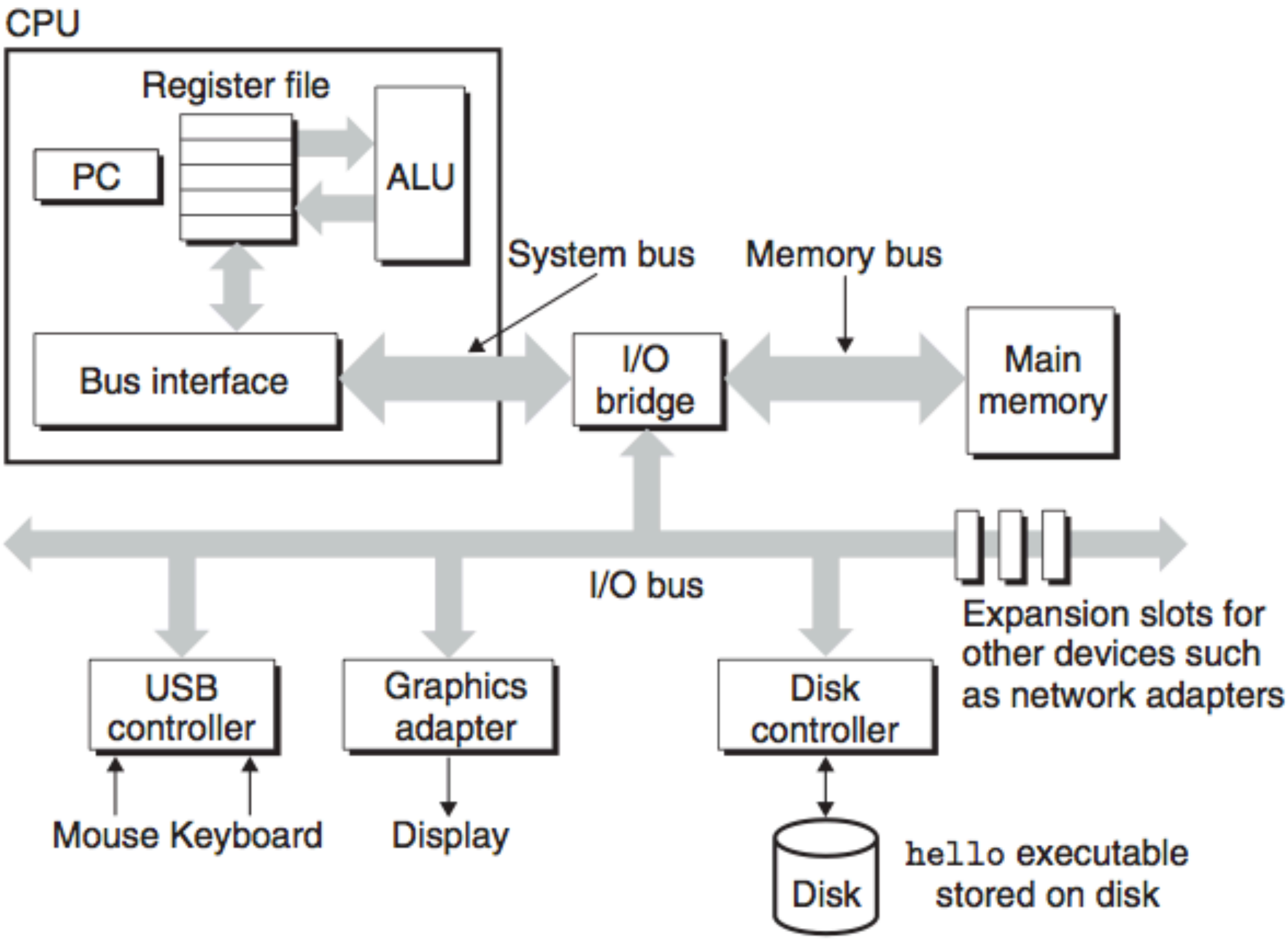
Compilation System



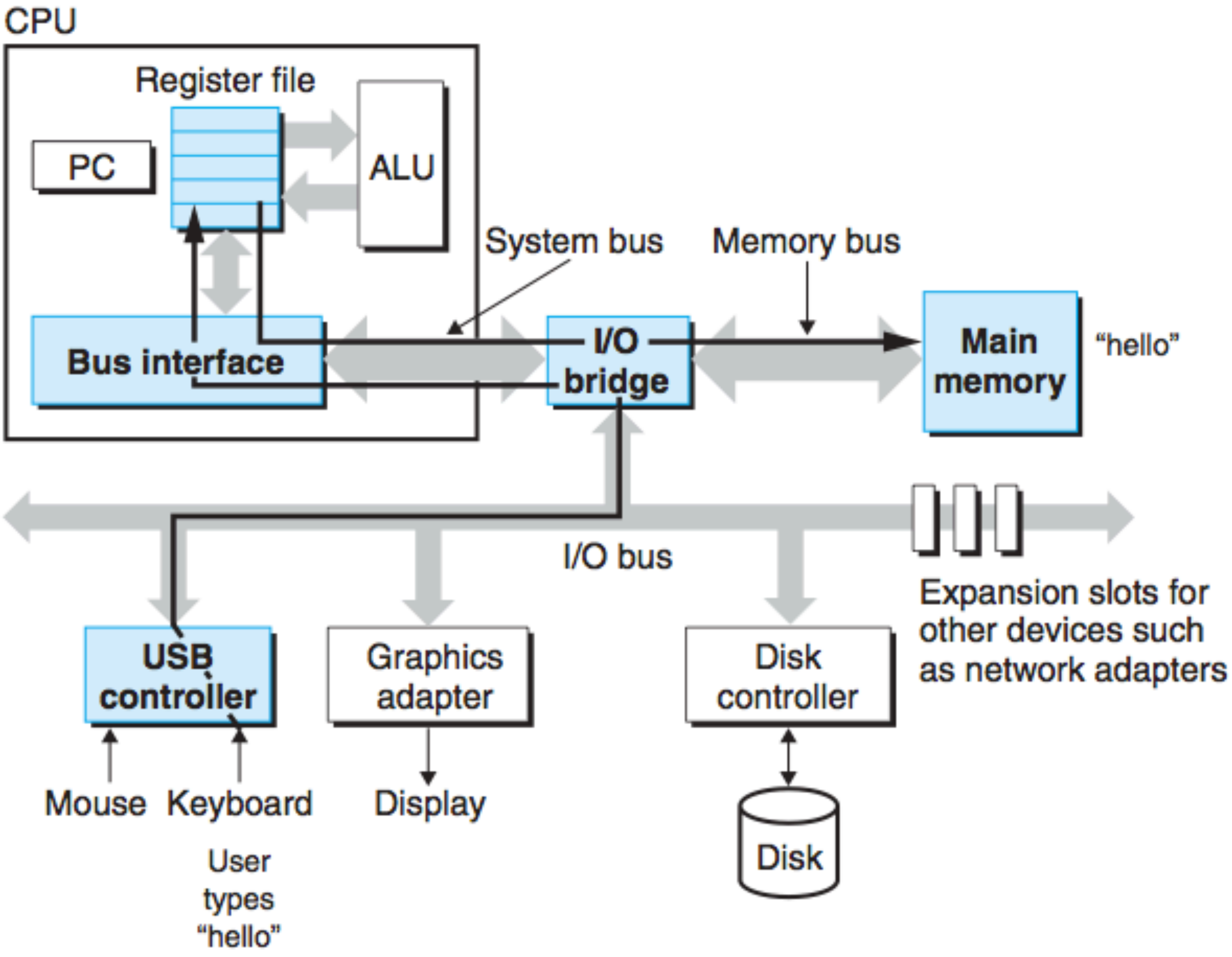
Processors

- **Machine Code Instructions**
  - Programs at the only level a machine can understand
  - Stored in memory
- **Processors**
  - Read instructions from memory
  - Interpret those instructions (do what the instructions say to do)
  - Implemented in hardware

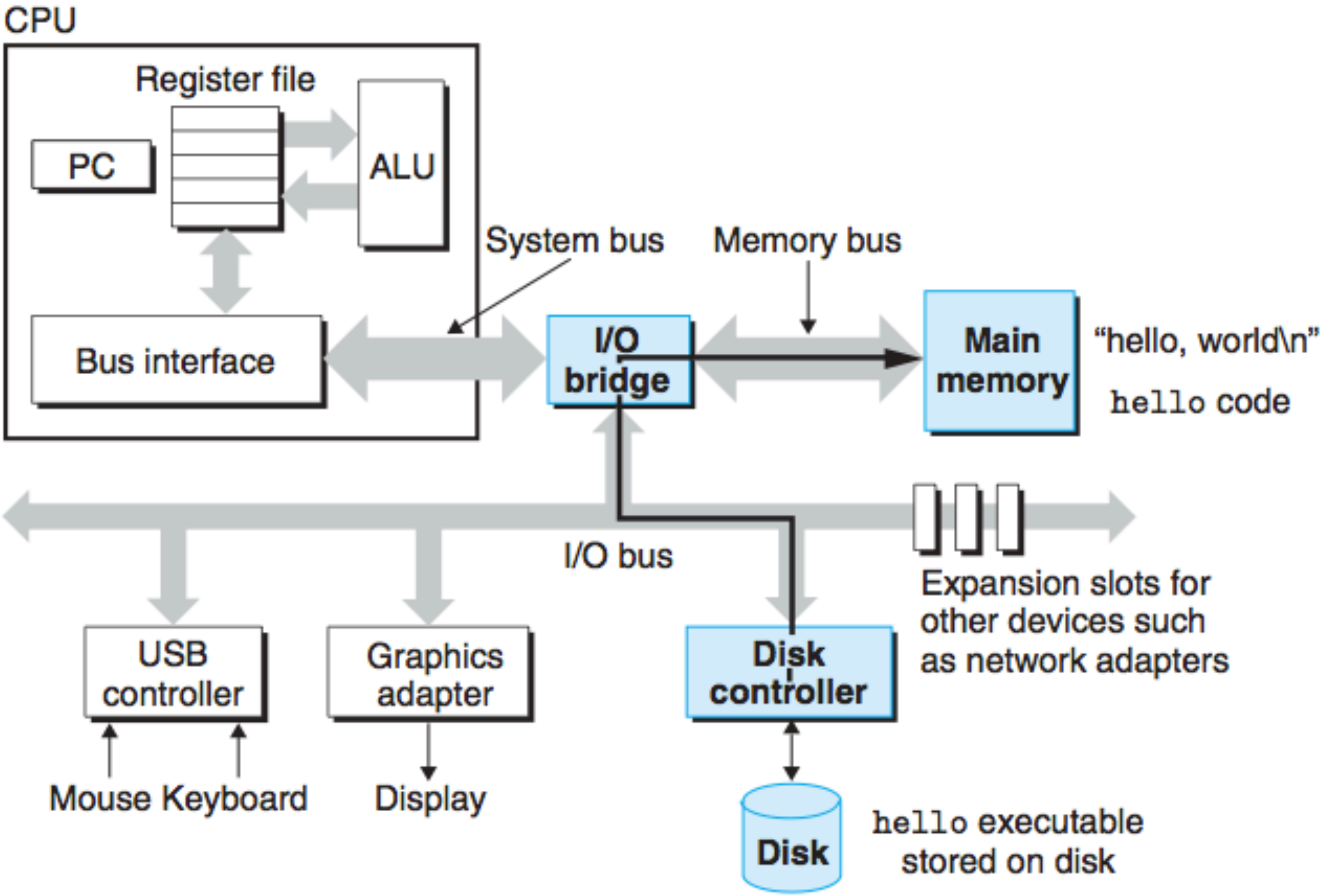
Hardware Organization



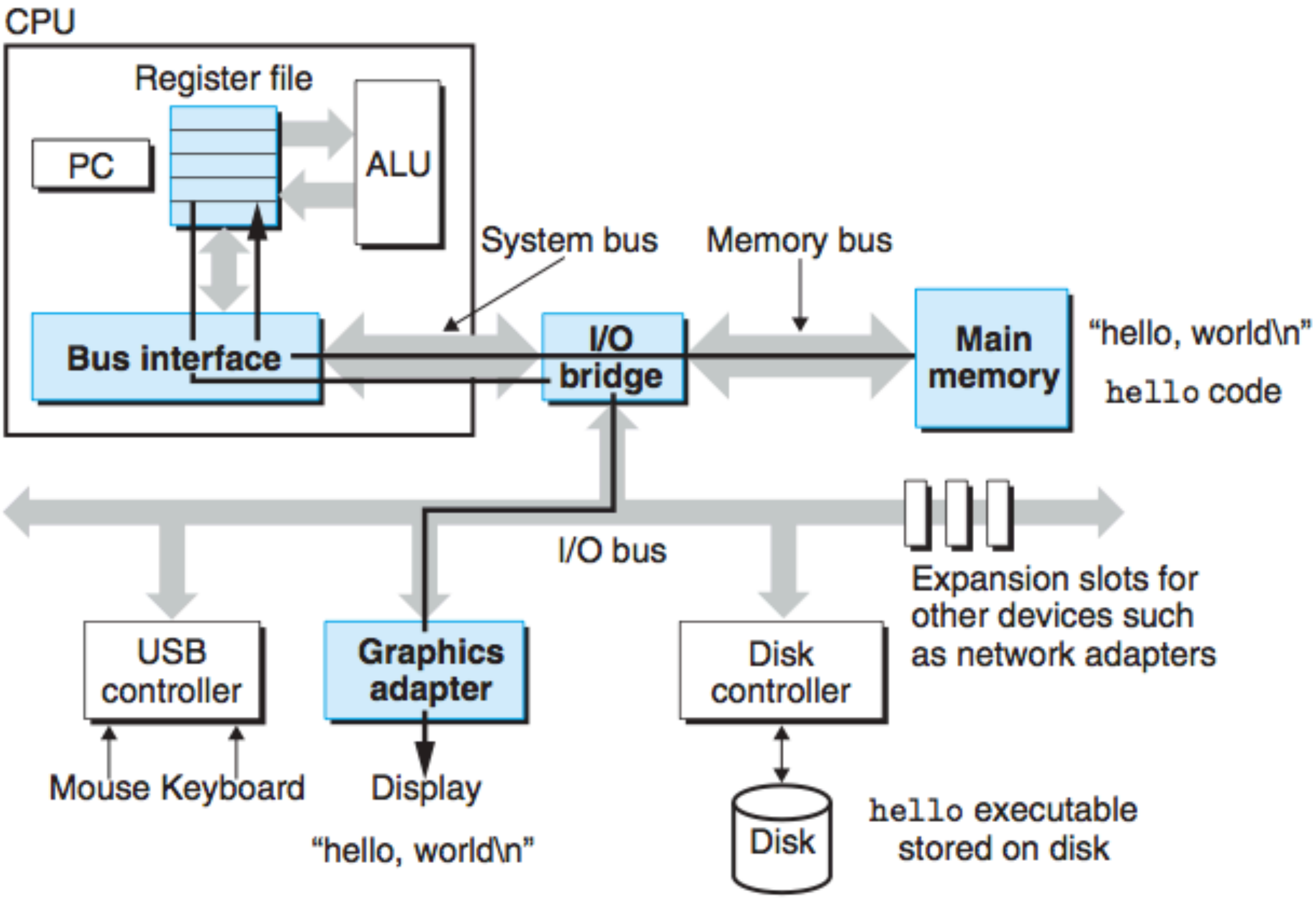
Reading hello command from keyboard



Loading the executable from disk into main memory



Writing the output string from memory to the display



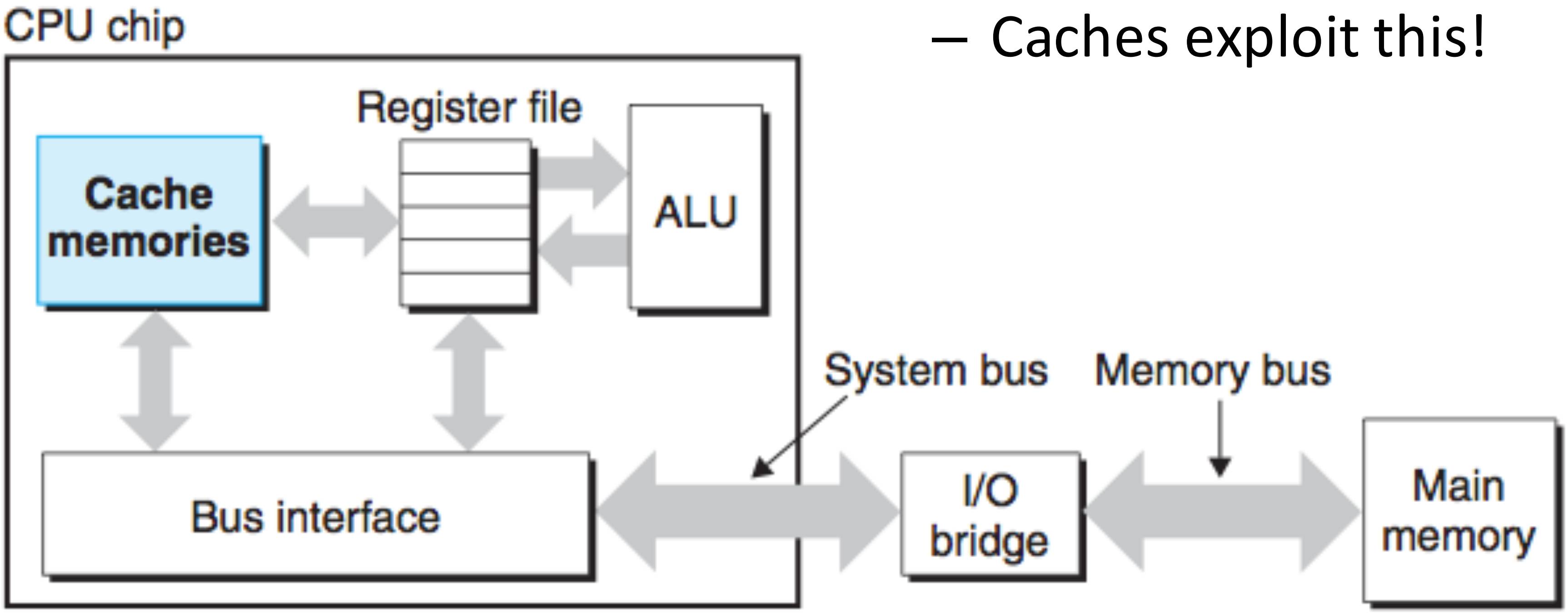
Memory this, Memory that

- **Memory is Important**
  - Stores program code
  - Stores program data
  - Accesses required for execution
- **Memory is Slow**
  - Yep, it takes a long time to access memory
  - Need a mechanism to reduce memory latency

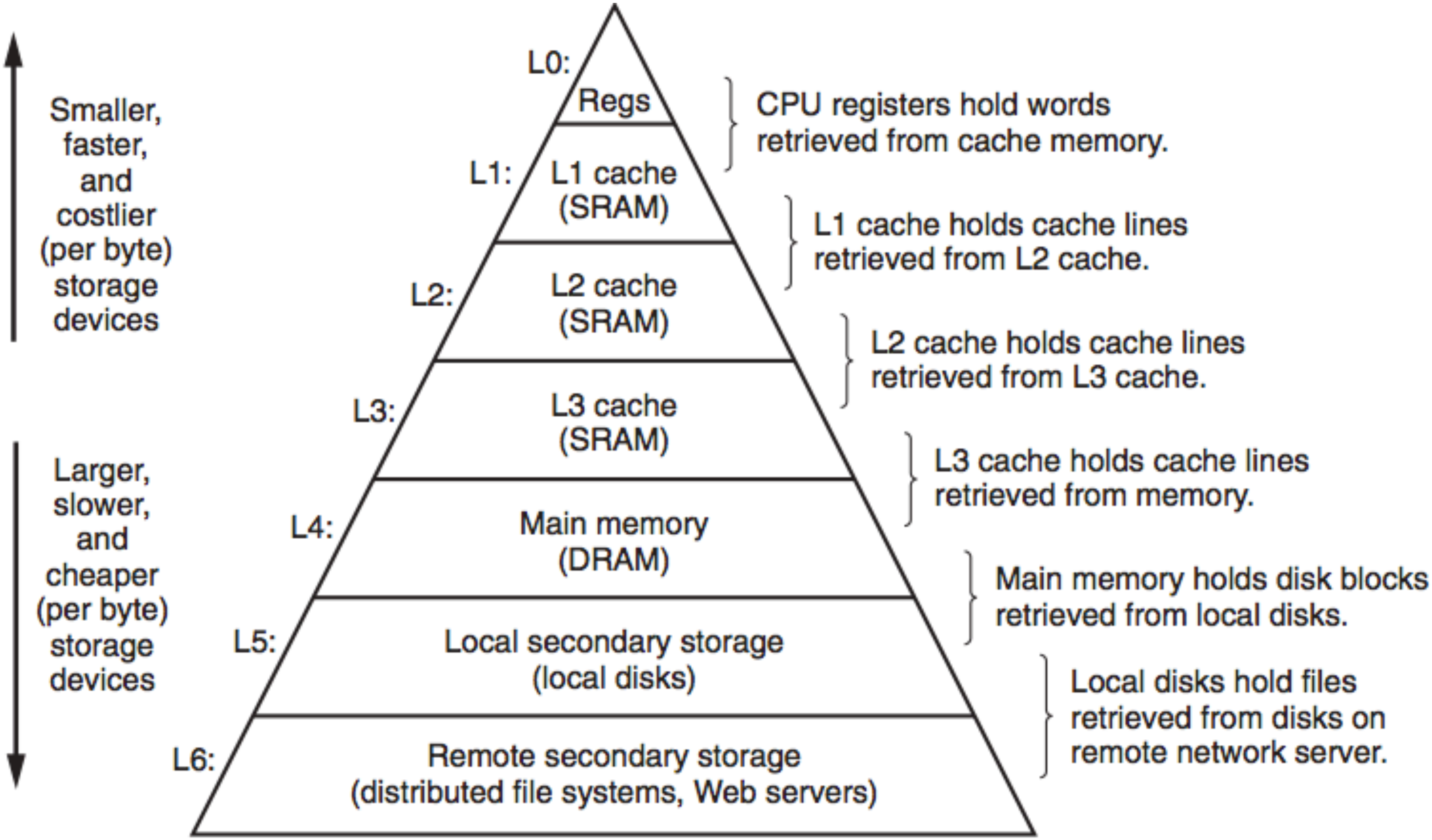
Cache

- **Smaller Memories**
  - Resides on CPU chip
  - Larger than register file
  - Smaller than RAM

- **Locality**
  - Access to program code and data tends to exhibit a high degree of locality, on both space and time
  - Caches exploit this!

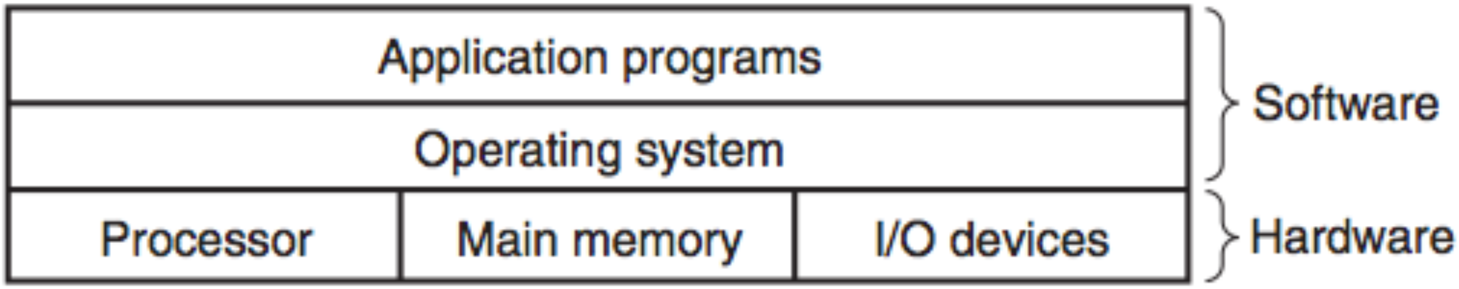


Memory Hierarchy


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (distributed file systems, Web servers)
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from L2 cache.
L2 cache holds cache lines retrieved from L3 cache.
L3 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network server.

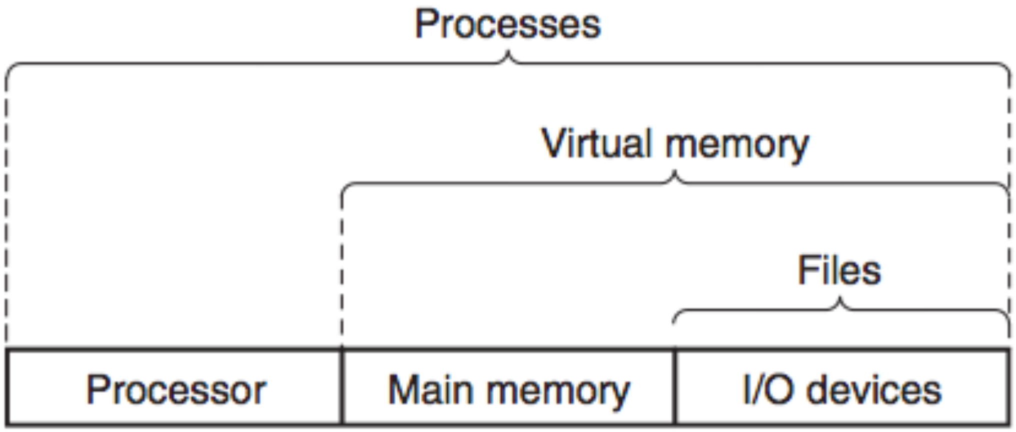
Operating System

- **Two Primary Purposes**
  - to protect the hardware (and other programs and files) from misuse by runaway applications
  - provide applications with simple and uniform mechanisms for manipulating complicated and often wildly different low-level hardware devices



OS Abstractions

- **How does the OS do this?**
  - Three fundamental abstractions



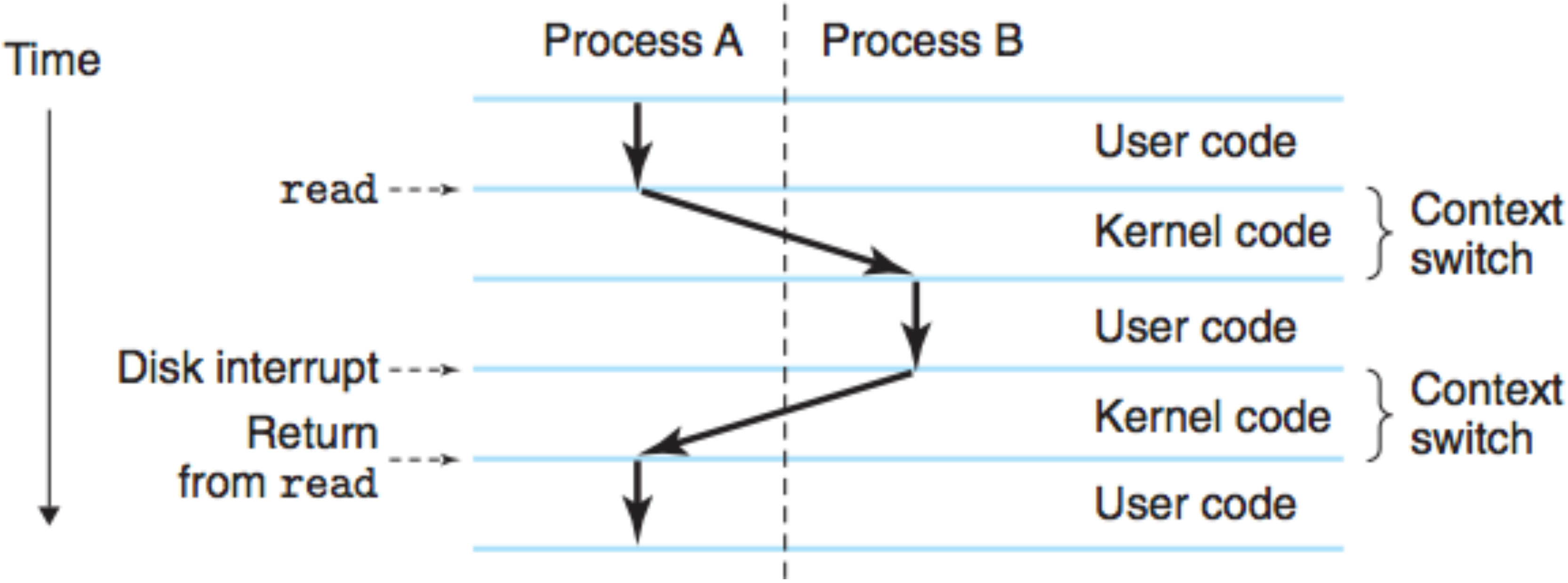
- **1: Files**
  - Abstraction for I/O devices
- **2: Virtual Memory**
  - Abstraction for main memory
  - Abstraction for I/O devices
- **3: Processes**
  - Abstraction for the processor
  - Abstraction for main memory
  - Abstraction for I/O devices

Processes

- **What are they?**
  - An abstraction for a running program
- **How many?**
  - Lots of them
  - Multiple processes can run concurrently
- **What do they give us?**
  - Illusion that each program has exclusive access to the processor and memory

What does “concurrently” mean?

The machine code instructions of one process are interleaved with the machine code instructions of another process.



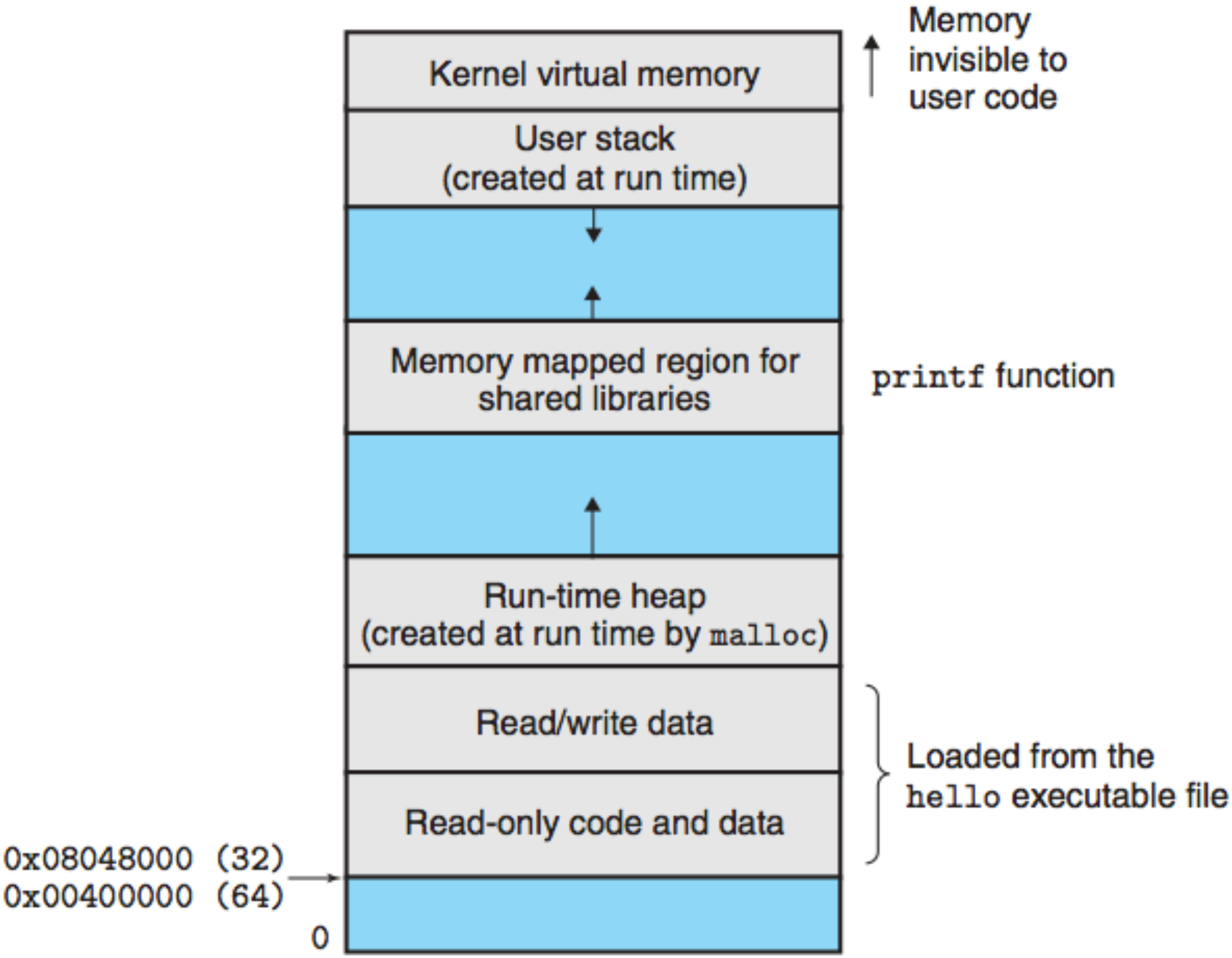
Aside: What about multiple “cores”?

- Each “core” in a multi-core CPU is effectively a *separate CPU*
- Each core can context-switch independently
- If enough processes are ready to run, two or more cores can be running programs *at the same time*
- Can be thought of as multiple computers on the same chip, but managed by the same OS and sharing the same memory and I/O devices

Processes and Threads

- **Processes**
  - The illusion is great, but what if I want to share my memory with another process?
  - You can’t!
- **Threads**
  - Associated with each process
  - Can be lots of them
  - Can share memory between them

Virtual Memory

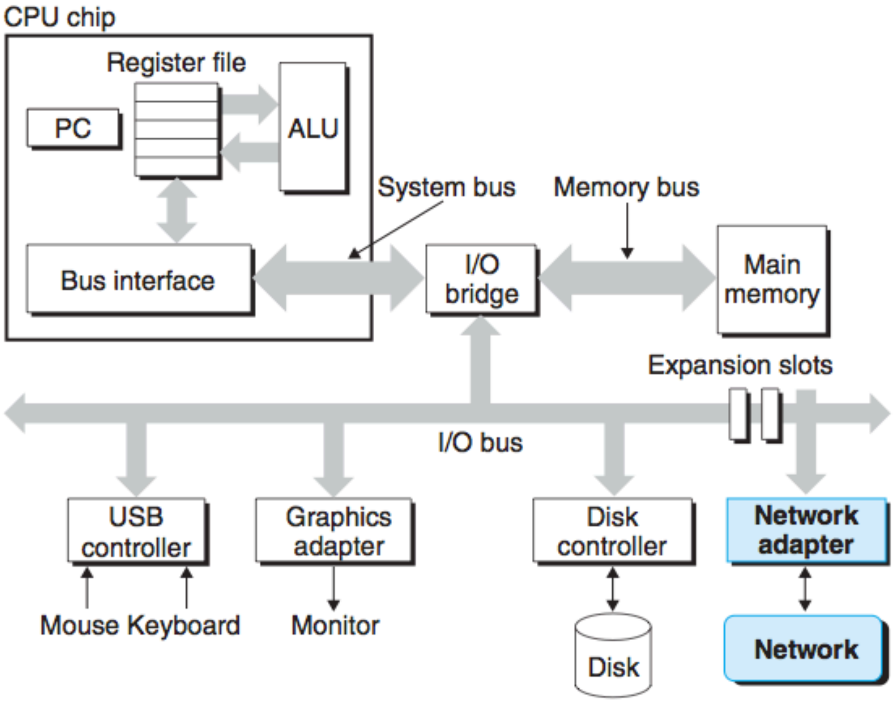

Memory invisible to user code
Kernel virtual memory
User stack (created at run time)
Memory mapped region for shared libraries
printf function
Run-time heap (created at run time by malloc)
Read/write data
Read-only code and data
Loaded from the hello executable file
0x08048000 (32)
0x00400000 (64)
0

Files

- **Sequence of bytes…**
  nothing more, nothing less

Network Communication

Processes like to talk to other processes



Anything Else?

- **Are there other questions that people have?**