CS330 - Computer Organization and Assembly Language Programming

Lecture 2

-Life Cycle of Hello World-

Professor : Mahmut Unan – UAB CS

Agenda

- We will learn the major ideas and themes in computer systems by tracing the **life cycle** of a simple “hello world” program.
- Amdahl’s Law

Announcement

- If you are having issue with the Vulcan Server connection;
  – E-mail to CS IT Admin, BJ Wilson

C Programming Tutorials

- https://www.learn-c.org/
- https://www.cprogramming.com/tutorial/c-tutorial.html
- Udemy
  - C Programming Tutorial – Complete Tutorial for beginners
  - The Complete C Programming Tutorial
- Lynda
  - Learning C
  - Advanced C Programming
- https://www.tutorialspoint.com/cprogramming/
- https://www.geeksforgeeks.org/c-programming-language/
- http://www.zentut.com/c-tutorial/
- Youtube
  - https://www.youtube.com/watch?v=2NWeucMKrLI&list=PL6gx4Cwl9DGAKIXv8Yr6nhGJ9Vlcjyymq

Hello World !

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

```
#include <stdio.h>
int main()
{
 /* the first program in CS330 */
 printf("hello, world\n");
 return 0;
}
```

Hello World !

A typical *C program* basically consists of the following parts;

– Preprocessor Commands
– Functions
– Variables
– Statements & Expressions
– Comment

```
#include <stdio.h>
int main()
{
/* the first kfnakjnsdfkprogram in
CS330 */
 printf("hello, world\n");
 return 0;
}
```

Information is Bits + Context

- "hello.c" is a source code
  - Sequence of bits (0 or 1)
  - 8-bit data chunks are called bytes
  - Each byte represents some text character in the program

#	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

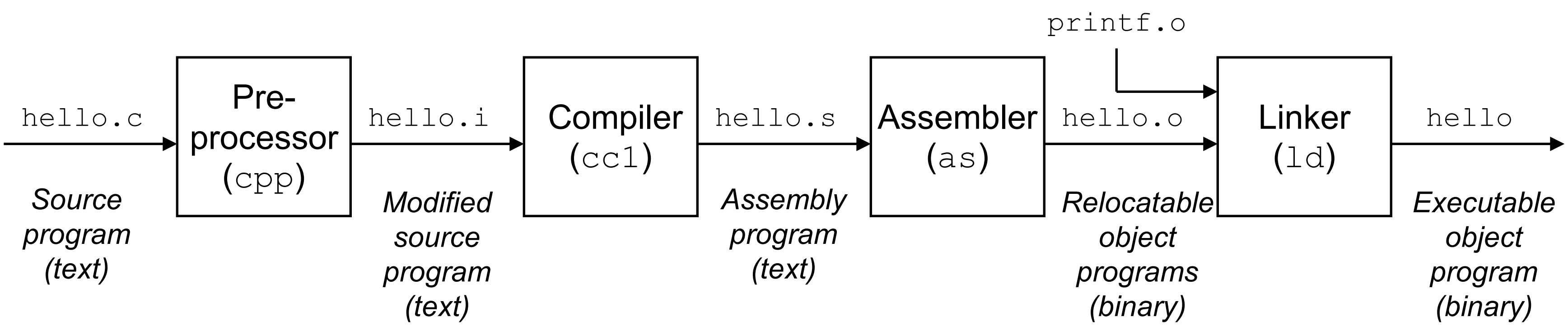
Figure 1.2 **The ASCII text representation of `hello.c`.**

Information is Bits + Context

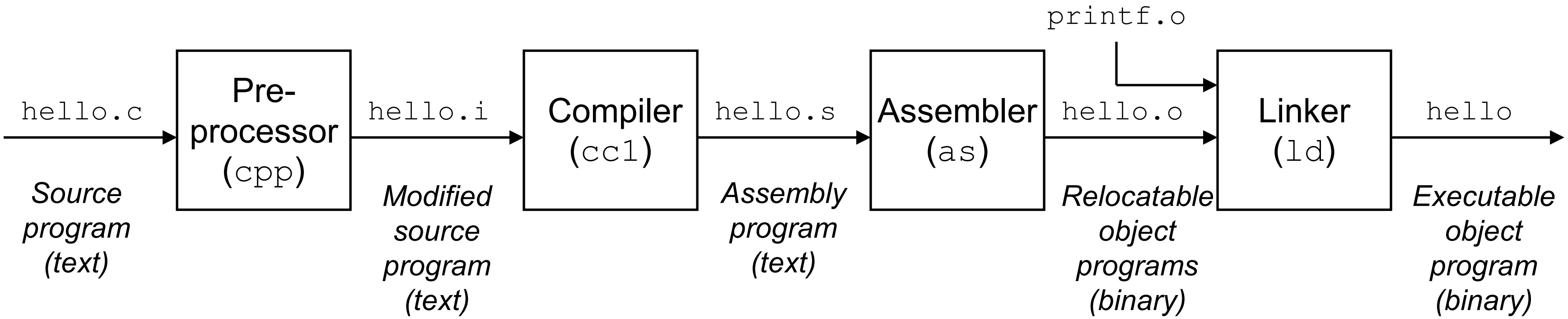
- Each byte has an integer value, corresponding to some character (ASCII, e.g. '#' → 35)
- Files made up of ASCII char. → text files
- All other files → binary files (e.g., 35 is a part of a machine command)
- Context is key
  - Same byte sequence might represent a character string or machine instruction

Programs translated by other programs

- `unix> gcc -o hello hello.c`



Compilation System



- 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 mach. language instructions
- Linking
  - Get printf() which resides in a separate precompiled object file

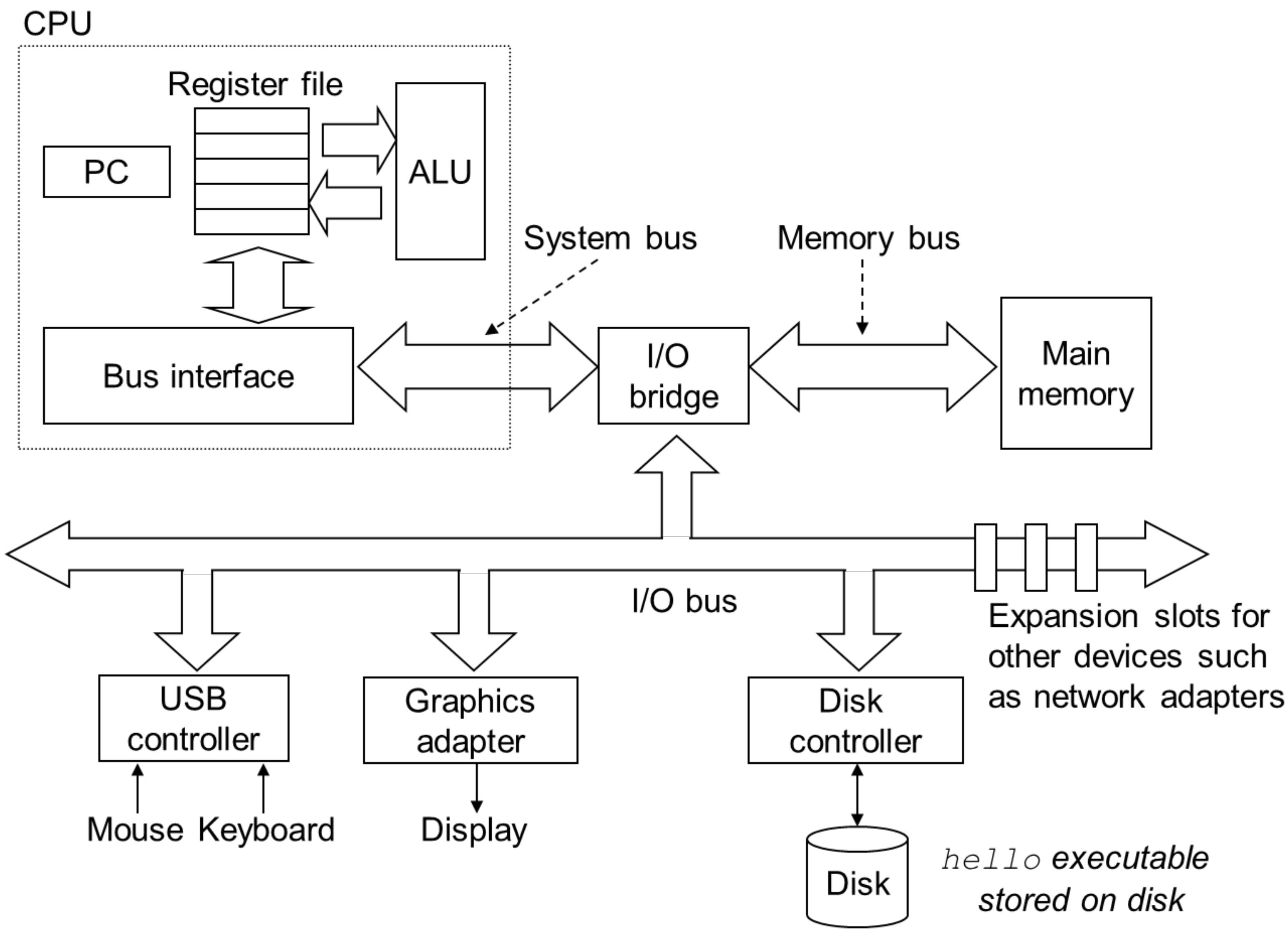
Running Hello Object File

- Running hello object file on the shell

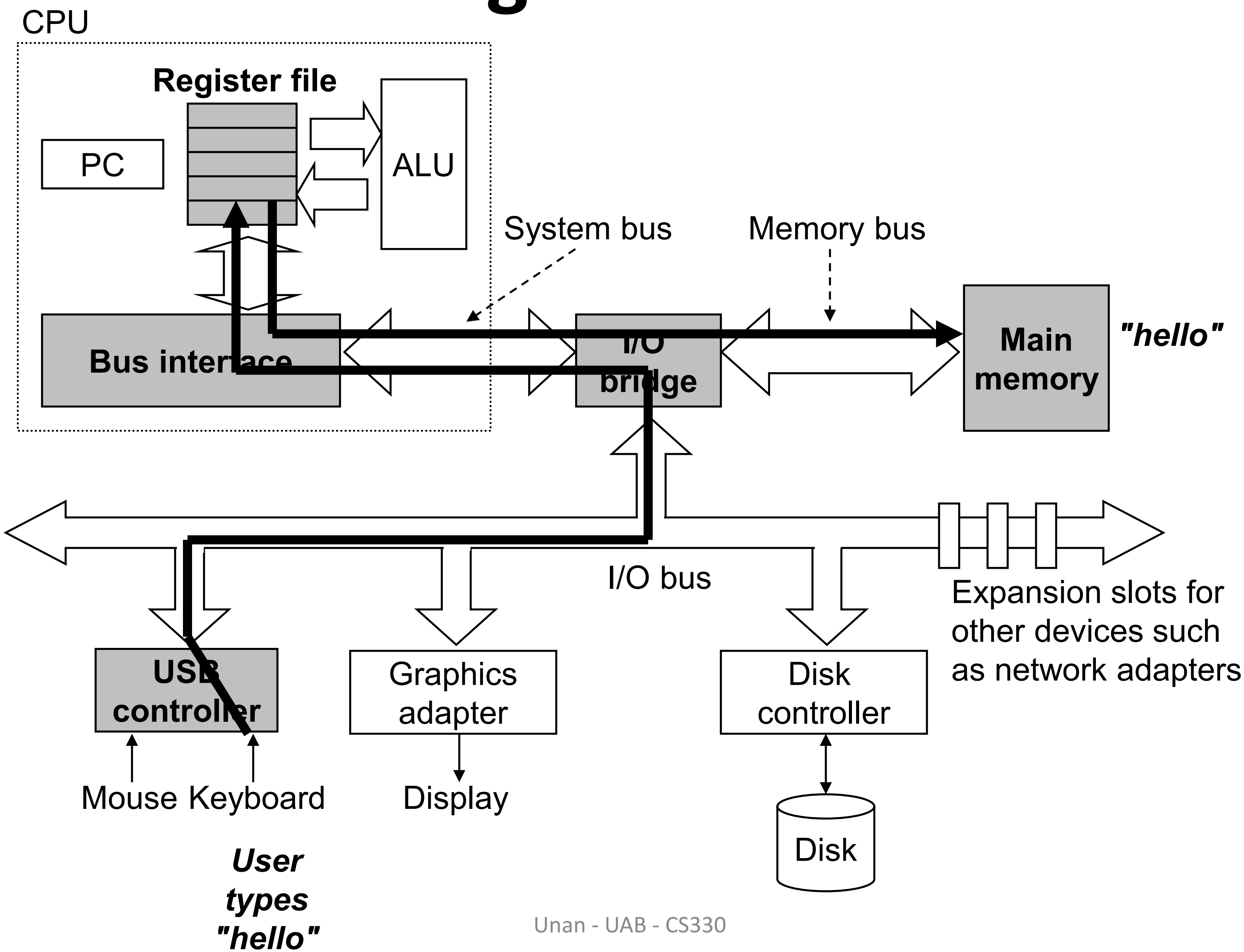
```
unix> ./hello
hello, world
unix>
```

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

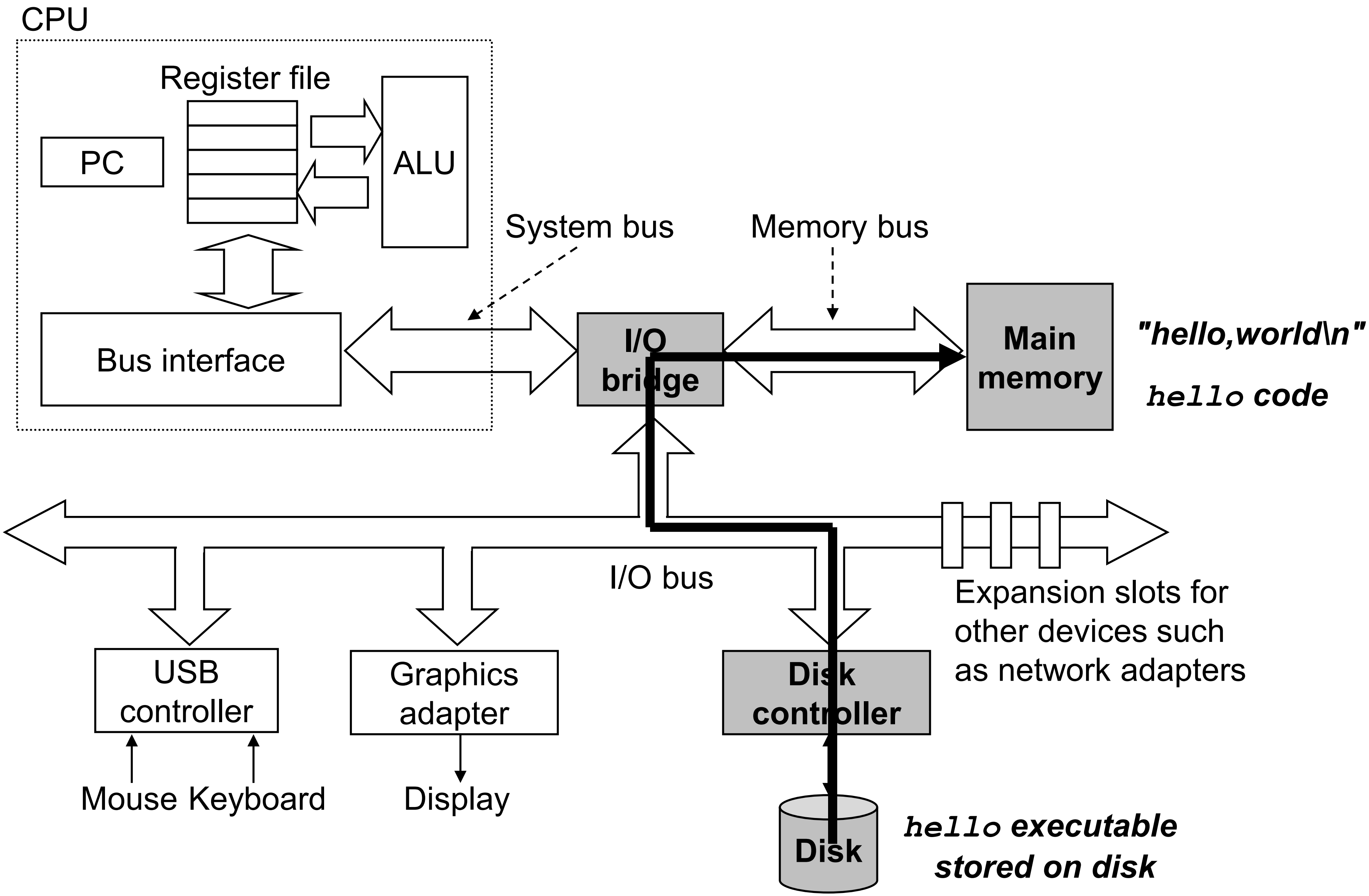
Hardware organization



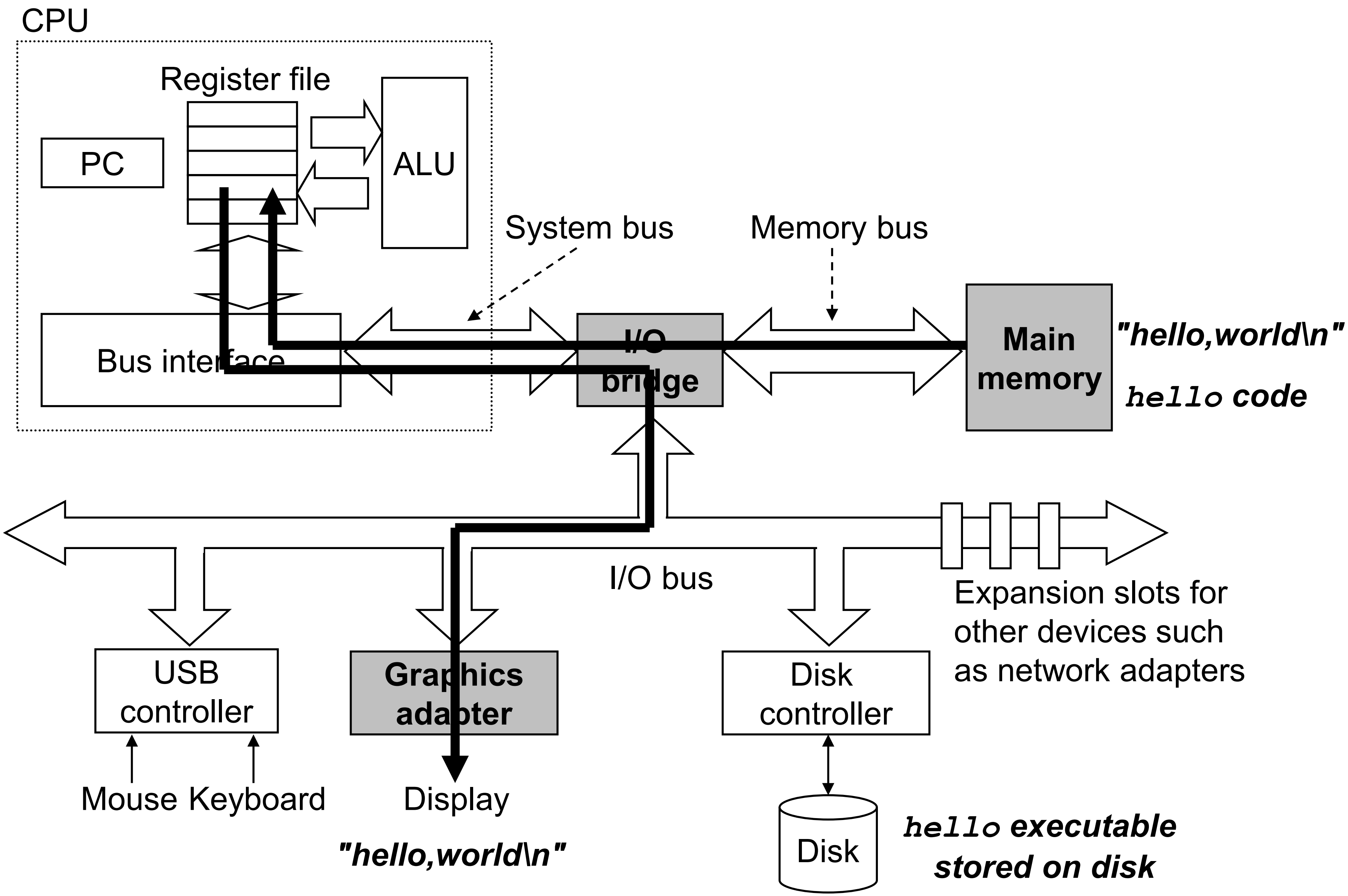
Running Hello World !



Running Hello World ! /2

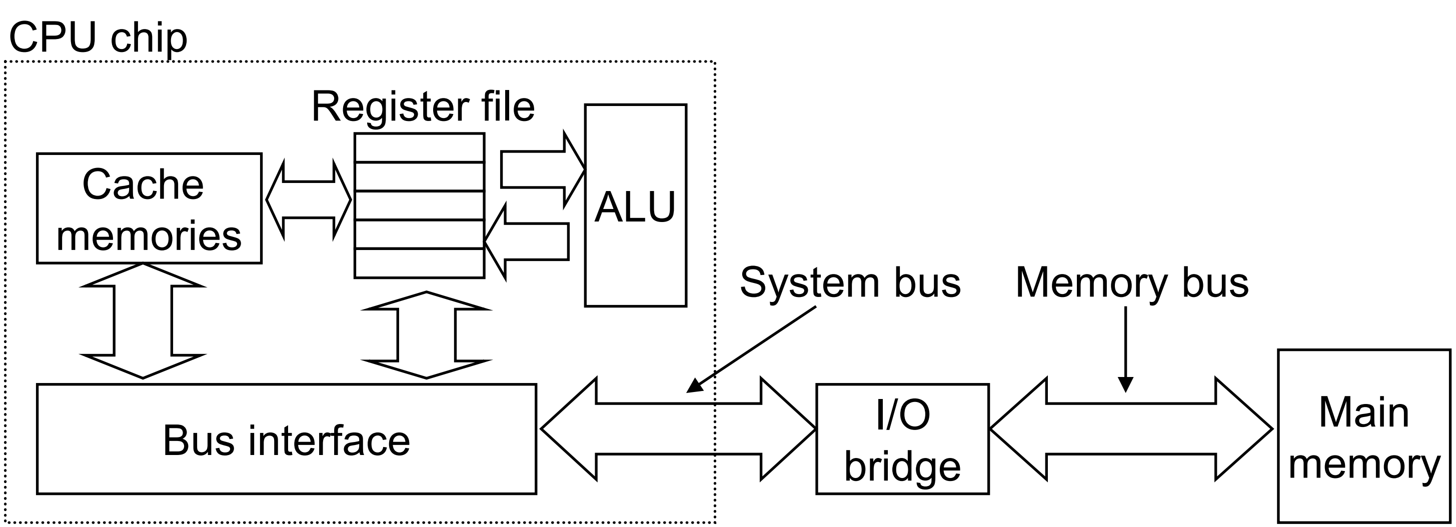


Running Hello World ! /3

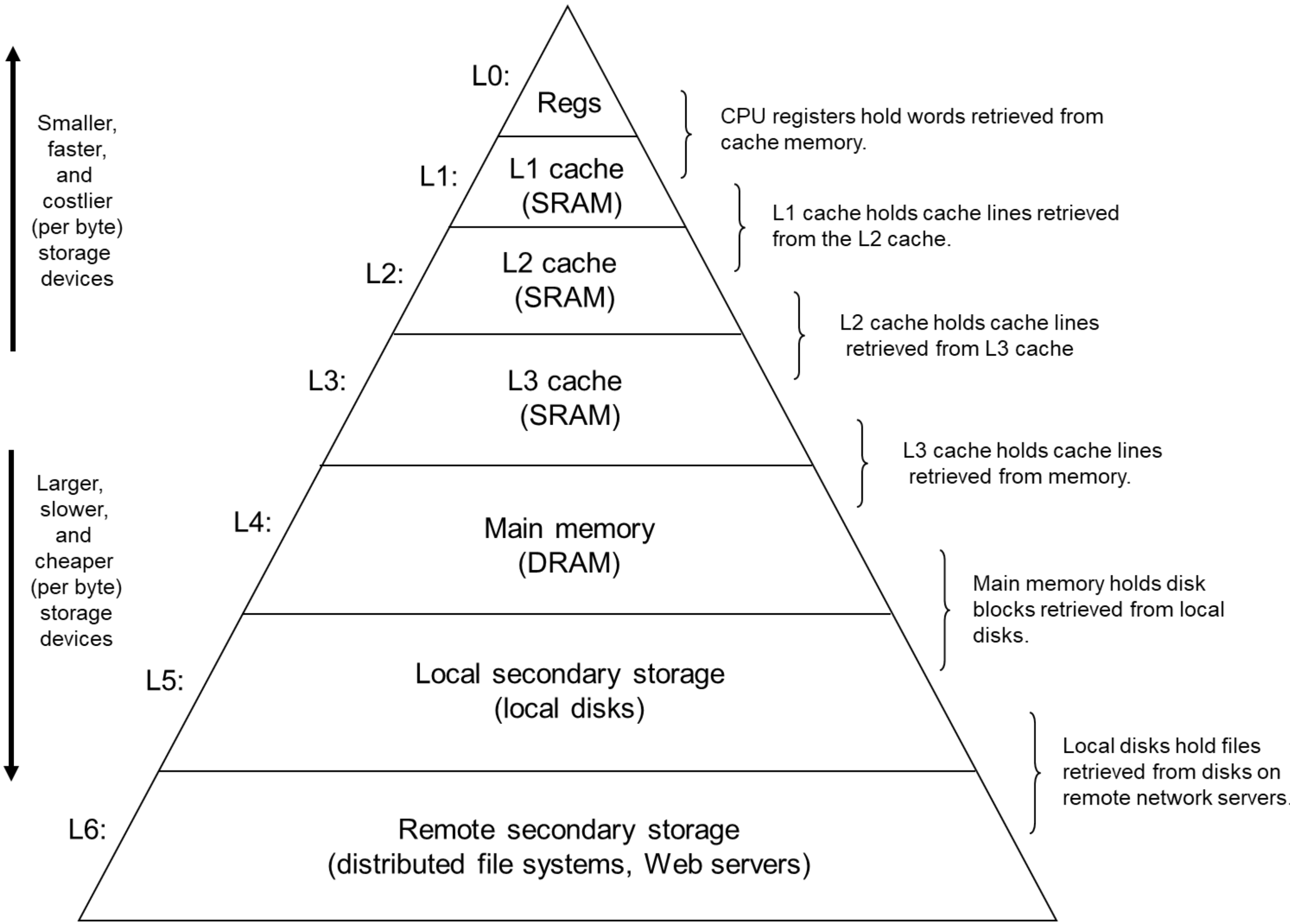


Caches matter

- System spends a lot of time moving info. around
- Larger storage devices are slower than smaller 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

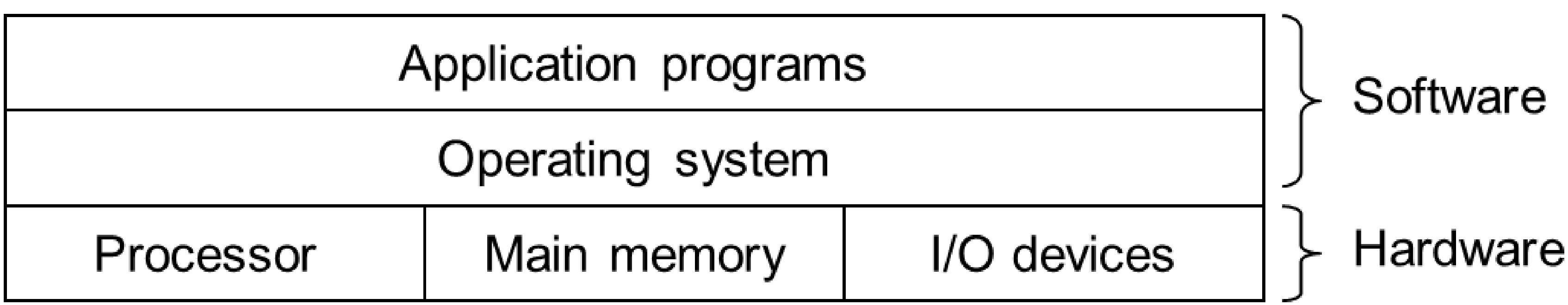


Storage devices form a hierarchy



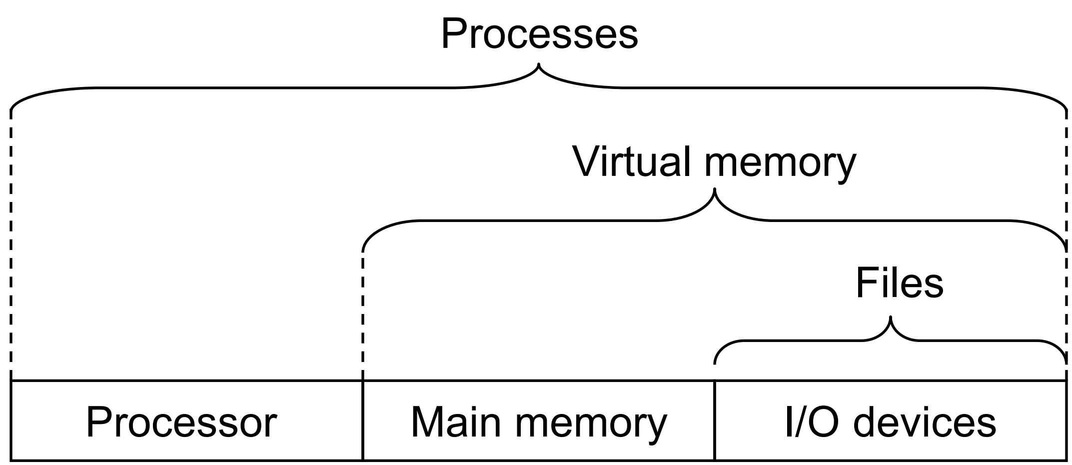
Operating system

- Hello program;
  - Doesn’t access the I/O, disk, memory..etc directly
  - It is relied o the services provided by OS
- OS has two primary goals
  - Protect resources from misuse by applications
  - Provide simple and uniform mechanisms for manipulating low-level hardware devices



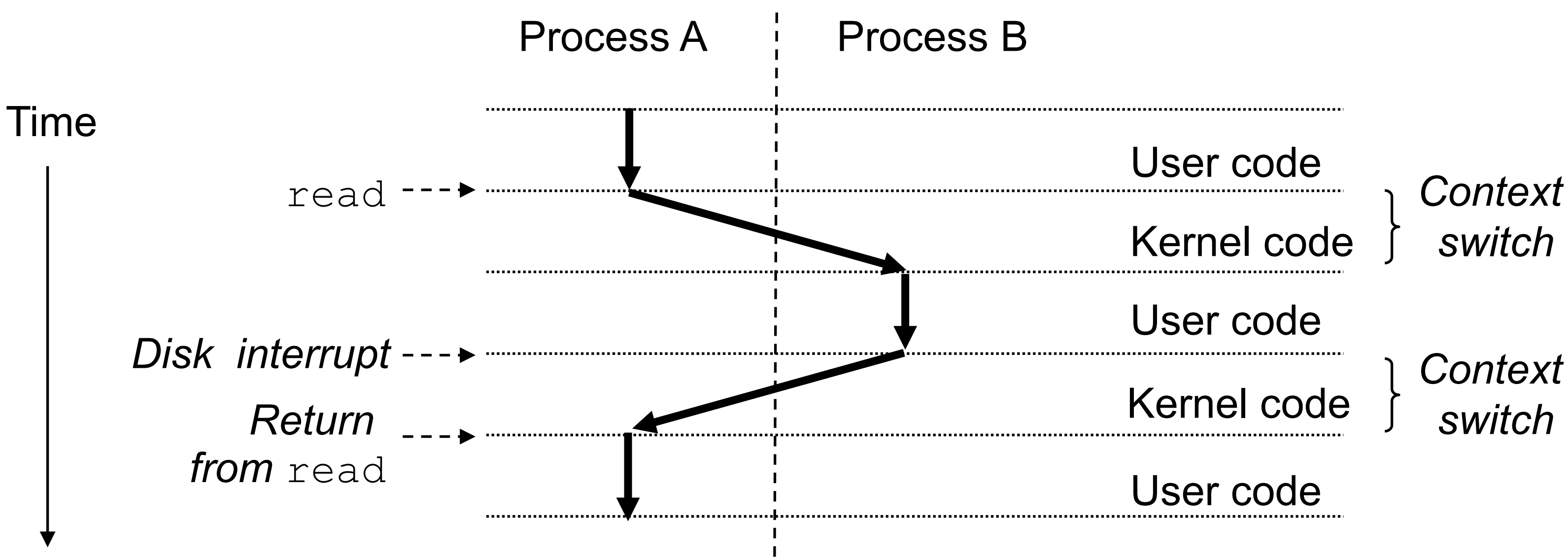
OS Abstractions

- The OS achieves these goals via fundamental abstraction;
  - 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

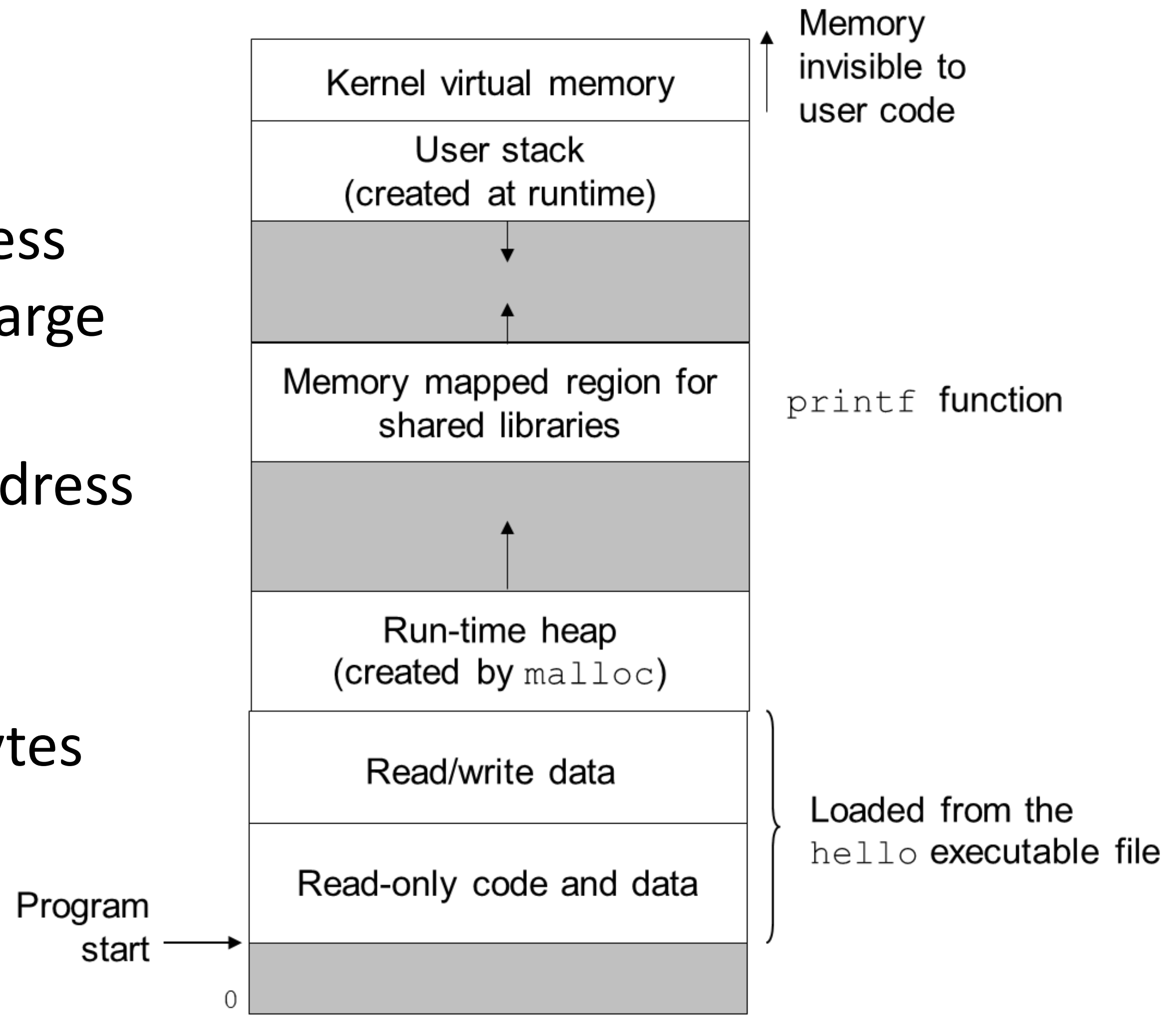


Threads

- Is a process has only one single control flow?
- Modern systems?
  - Threads
- Sharing data between multiple threads…vs… sharing between multiple processes?

Virtual memory

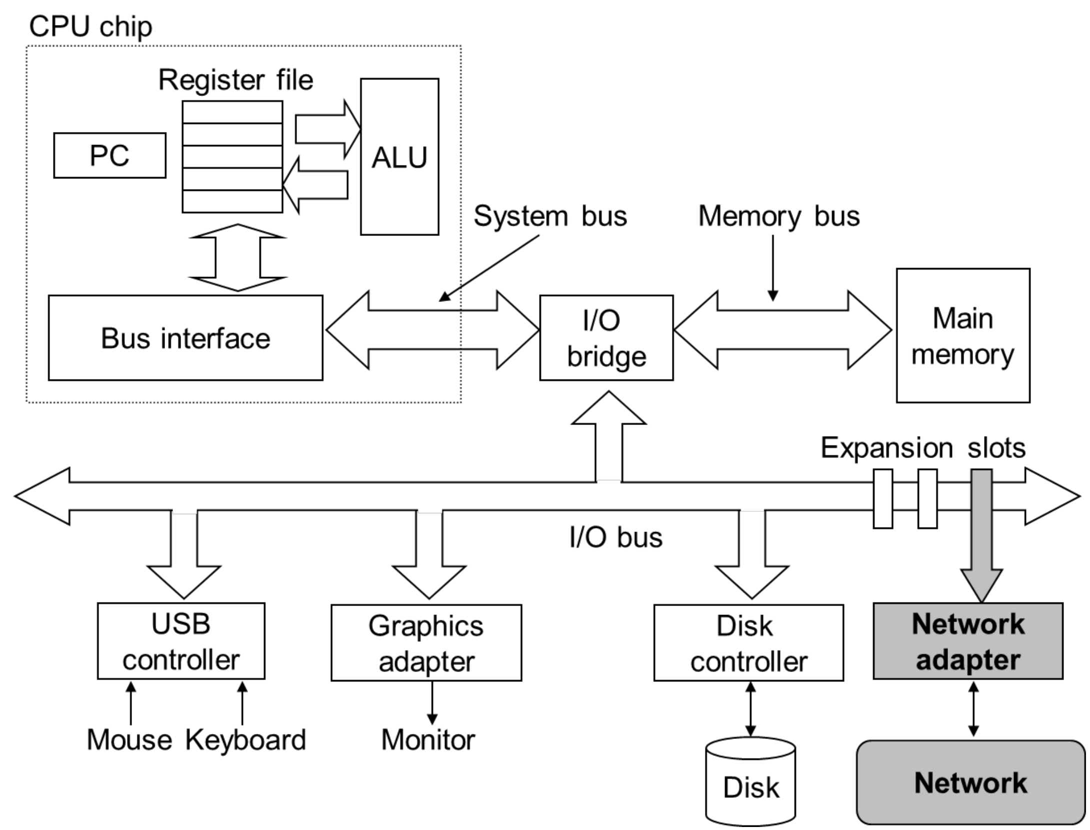
- Illusion that each process has exclusive use of a large main memory
  - Example: Virtual address space for Linux



- **Files:** A sequence of bytes

Networking

- Talking to other systems
- Network seen as another I/O device
- Many system-level issues arise in presence of network
  - Coping with unreliable media
  - Cross platform compatibility
  - Complex performance issues



Amdahl's Law

- Effectiveness of improving the performance of one part of system
- Speed up one part → Effect on the overall system performance?
- $T_{new} = (1-\alpha)T_{old} + \frac{\alpha T_{old}}{k}$
- $= T_{old}[(1-\alpha) + \frac{\alpha}{k}]$
- $S = \frac{T_{old}}{T_{new}}$
- $S = \frac{1}{((1-\alpha)+(\alpha/k))}$

Amdahl's Law / Example

- It is named after computer scientist Gene Amdahl(a computer architect from IBM and Amdahl corporation).

- Consider a system;
  - A part of the system initially consumed 60% of the time (α = 0.6)
  - It is sped up by a factor of **3** (k=3)

- Overall improvement ?

Amdahl's Law / Example

- Consider a system;
  - A part of the system initially consumed 60% of the time (α = 0.6)
  - It is sped up by a factor of **3** (k=3)

- Overall improvement ?
- $S = \frac{1}{((1-\alpha)+(\alpha/k))}$

- = 1 / [0.4 + 0.6/3] = **1.67 times**

Amdahl's Law / Example 2

- Calculate the following improvements on a current system and decide which one is better

- **1)** if we make 90% of a program run 10 times faster.

- **2)** if we make 80% of a program run 20% faster

Amdahl's Law / Example 2

- **1)** if we make 90% of a program run 10 times faster.

$$S = \frac{1}{((1-\alpha)+(\alpha/k))} = \frac{1}{((1-0.9)+(0.9/10))} = \mathbf{5.26}$$

- **2)** if we make 80% of a program run 20% faster

$$S = \frac{1}{((1-\alpha)+(\alpha/k))} = \frac{1}{((1-0.8)+(0.8/1.2))} = \mathbf{1.153}$$

Conclusion

- We have seen the big picture
- 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

Next Class

- We will start **Chapter 2**
- **Representing and manipulation Information**
- Please read **2.1.1 – 2.1.5**