Computer SW and HW

Information in Computer Systems

- Every Info. is a Sequence of bits.
  - Source Program: Sequence of bits (Text file)

```
#include <stdio.h>

int main()
{
        printf("hello, world₩n");
}
```

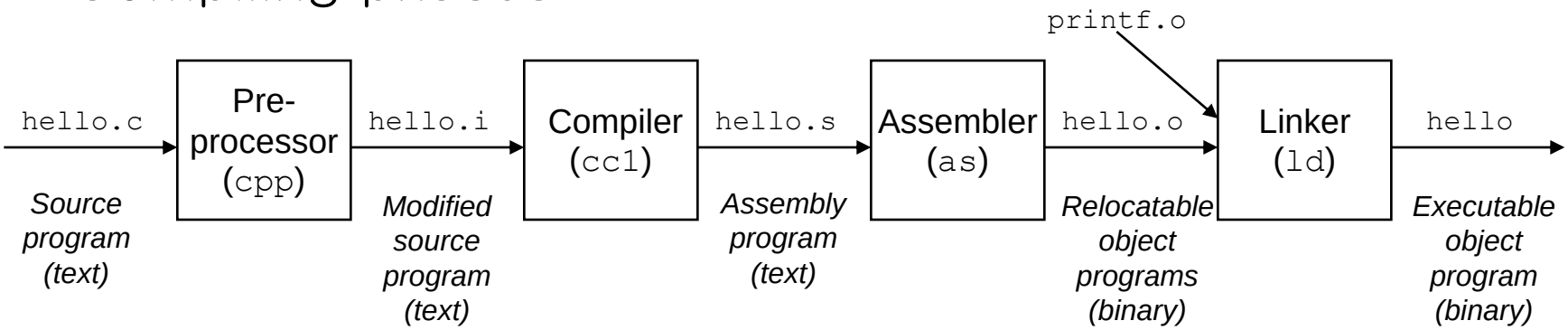
#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i
35	105	110	99	108	117	100	101	32	60	115	116	100	105

  - Executable Program: Sequence of bits (Binary file)

```
.&##A.+?@@^
```

Source Program vs. Executable Program

- unix> gcc -o hello hello.c
  - compiler (gcc) translates the source program to the executable program
- Compiling phases



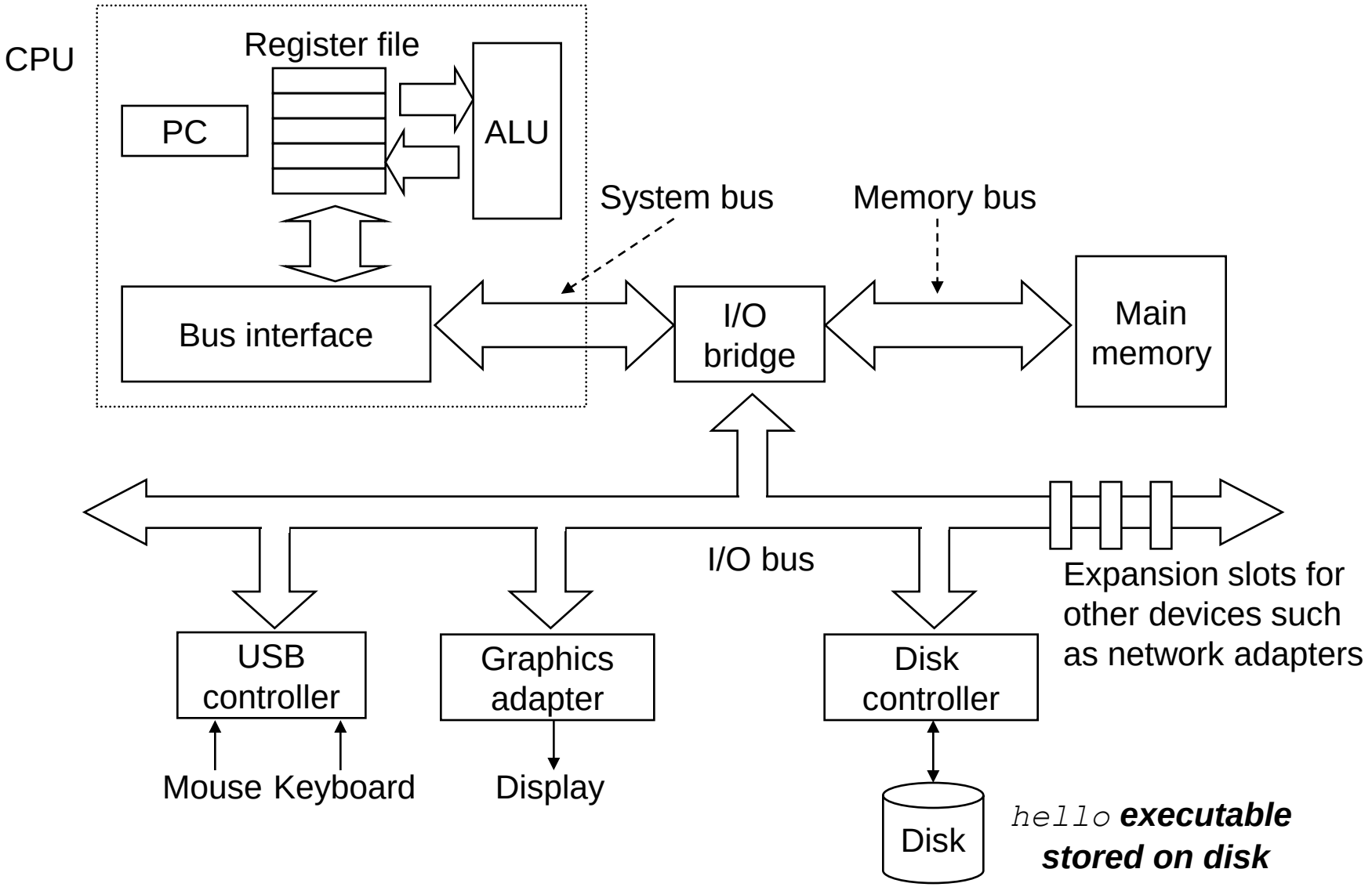
  - Preprocessing phase: Modify the original C program according to directives (e.g., #include)
  - Compilation phase: translate to assembly-language program
  - Assembly phase: translate to relocatable object program
  - Linking phase: merge with other pre-compile object file and result in the executable program

How the executable program runs on a computer?

- unix> ./hello
- hello, world
- unix>

- To understand what happens here, let's take a look at the hardware organization of a computer

Hardware Organization

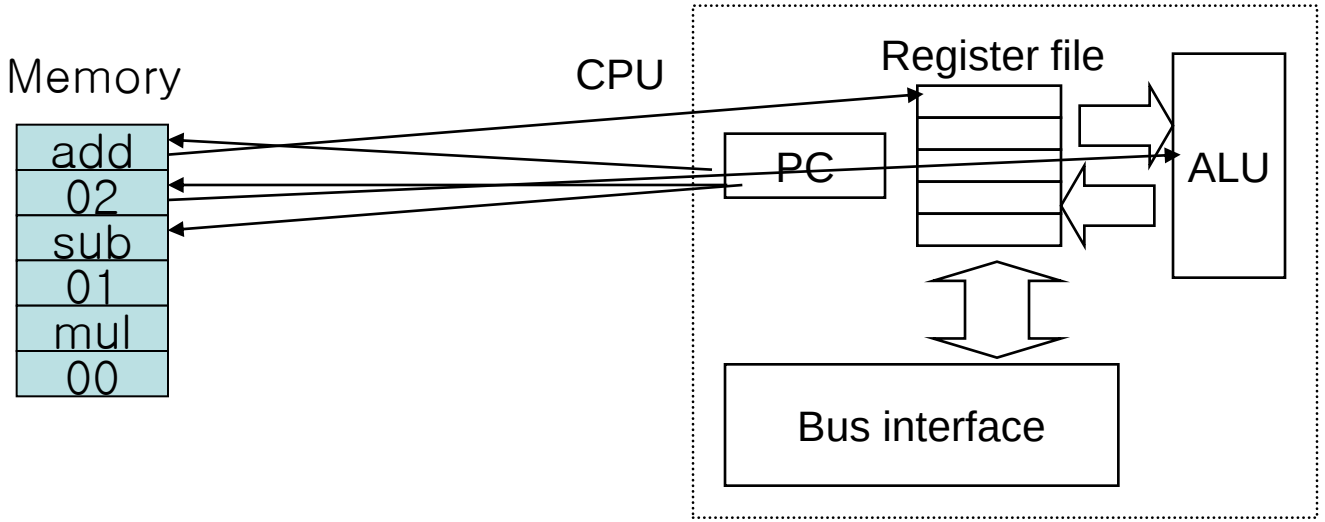


Hardware Organization

- Buses
  - a collection of electrical conduits
  - number of conduits = bus width = number of bits = word
  - Intel Pentium (32bits), Sun SPARCS (64bits), embedded controllers (8bits or 16bits)
- I/O Devices
  - system's connection to the external world
  - Input (Keyboard, Mouse), Output (display), Storage of data and program (disk)
  - Each I/O device is connected to the I/O bus by either a controller or an adapter
    - controller: chip set in the device itself or on the motherboard
    - adapter: card that plugs into a slot of the motherboard
- Main Memory
  - a temporary storage device that holds both a program and the data it manipulates while the program is executing the program
  - Physically, a collection of DRAM chips
  - Logically, a linear array of bytes

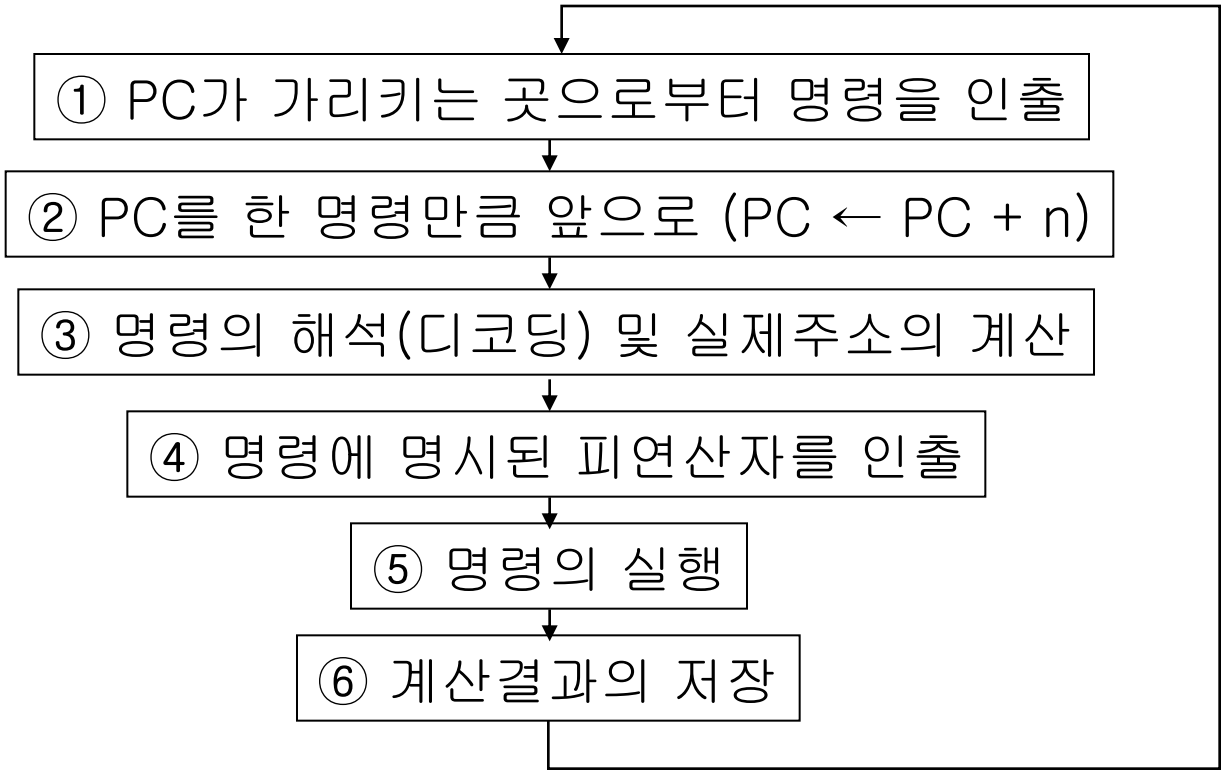
Hardware Organization

- Processor
  - the engine that interprets and executes instructions stored in main memory
  - A word size register called the Program Counter always points a machine instruction to be executed next
  - After power-on, the processor repeats
    - reads the instruction from the memory pointed by PC
    - interprets the bits in the instruction
    - perform some simple operation (load, store, calculate, I/O read, I/O write, Jump) dictated by the instruction
    - update the PC to point the next instruction



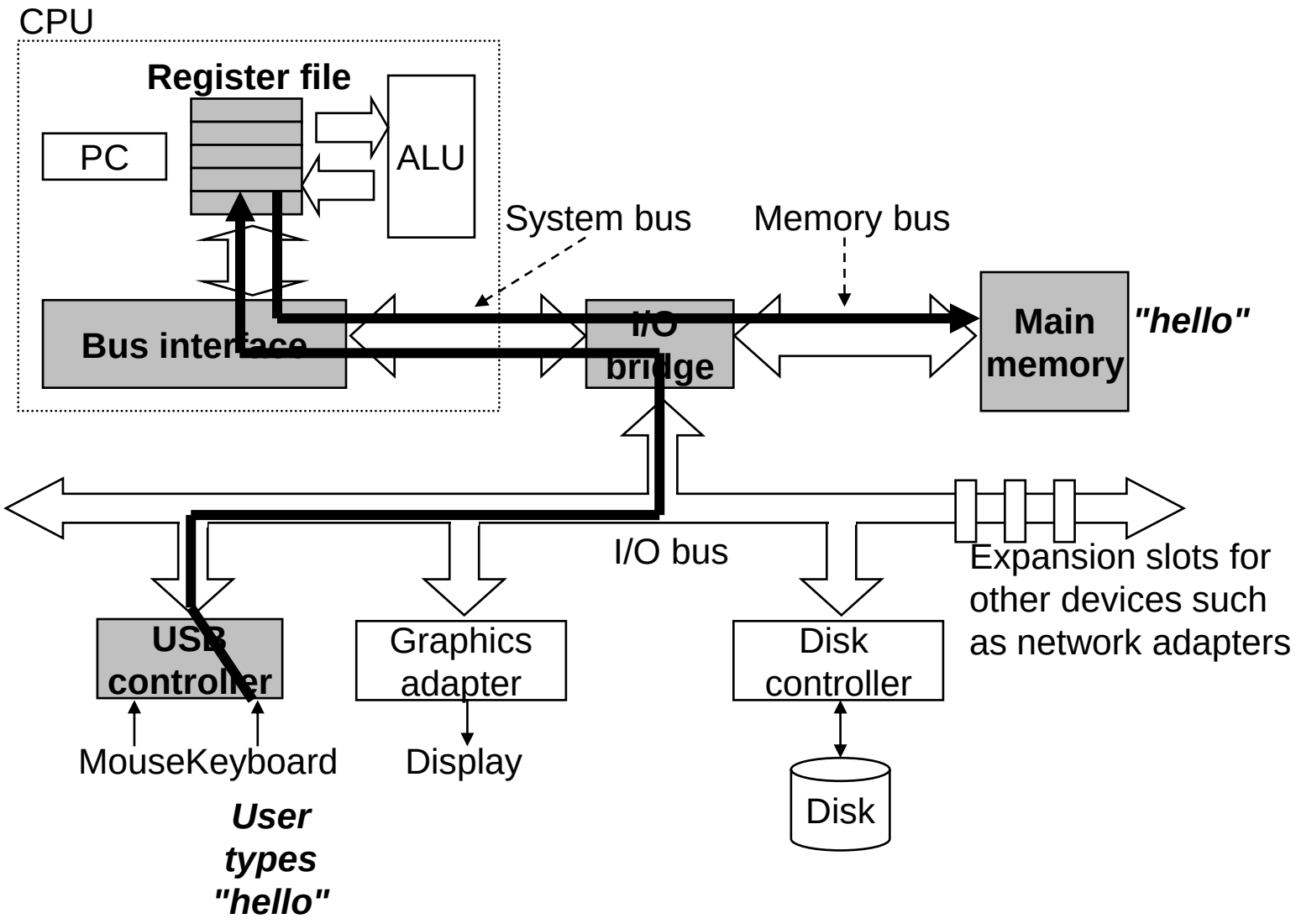
Executing Instructions

- Repeat fetch-execute cycles
  - Program Counter (PC) always points the next instruction addr.
  - 인출-실행 명령 주기 *fetch-execution instruction cycle*
    1. 인출 *fetch*: 주소(PC)에 의거하여 명령어 인출
    2. 실행 *execute*: 가져온 명령어가 요구하는 일을 수행



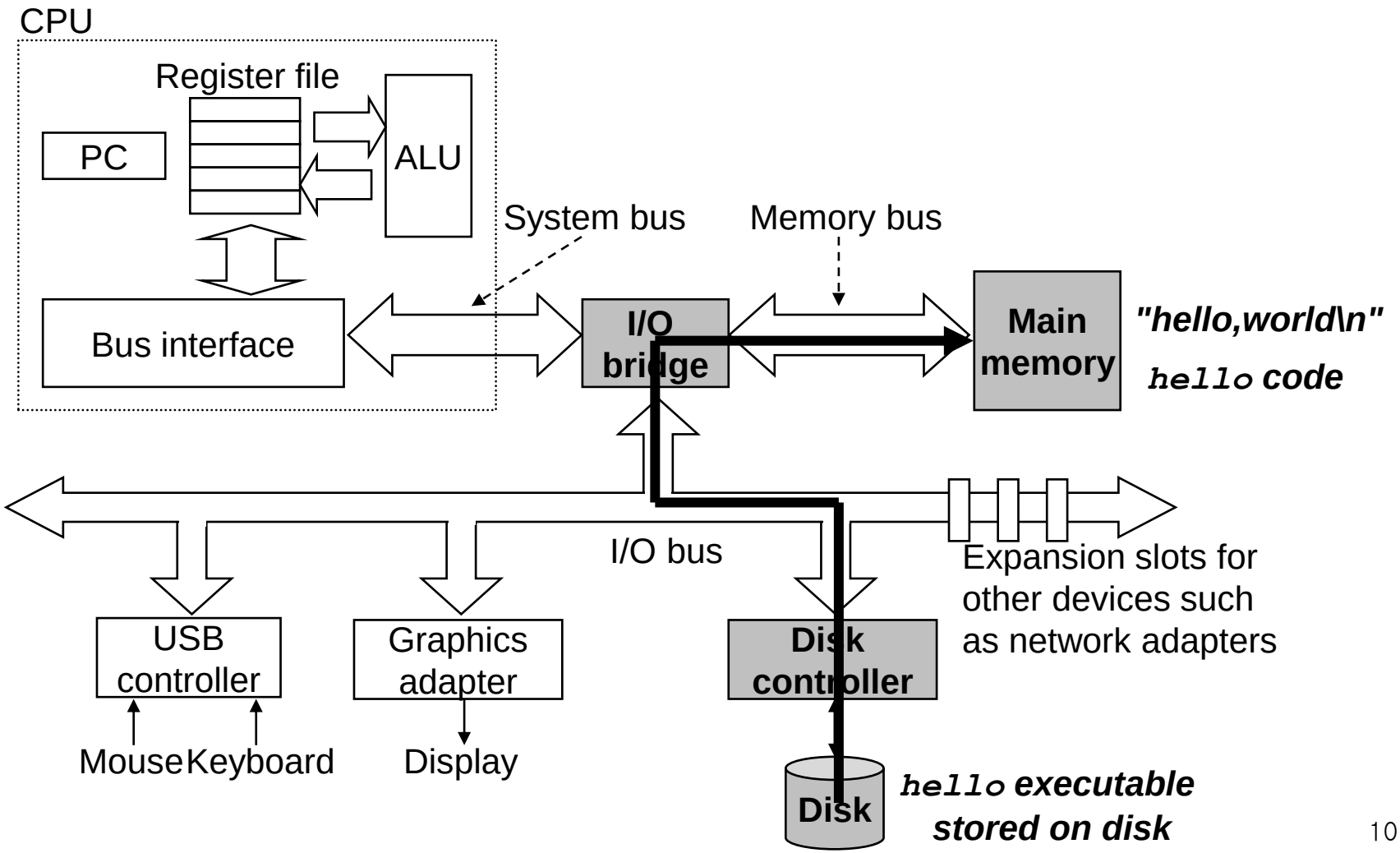
Running the hello program

Shell program is executing its instructions, reading characters (“./hello”) from keyboard into a register and storing them in memory



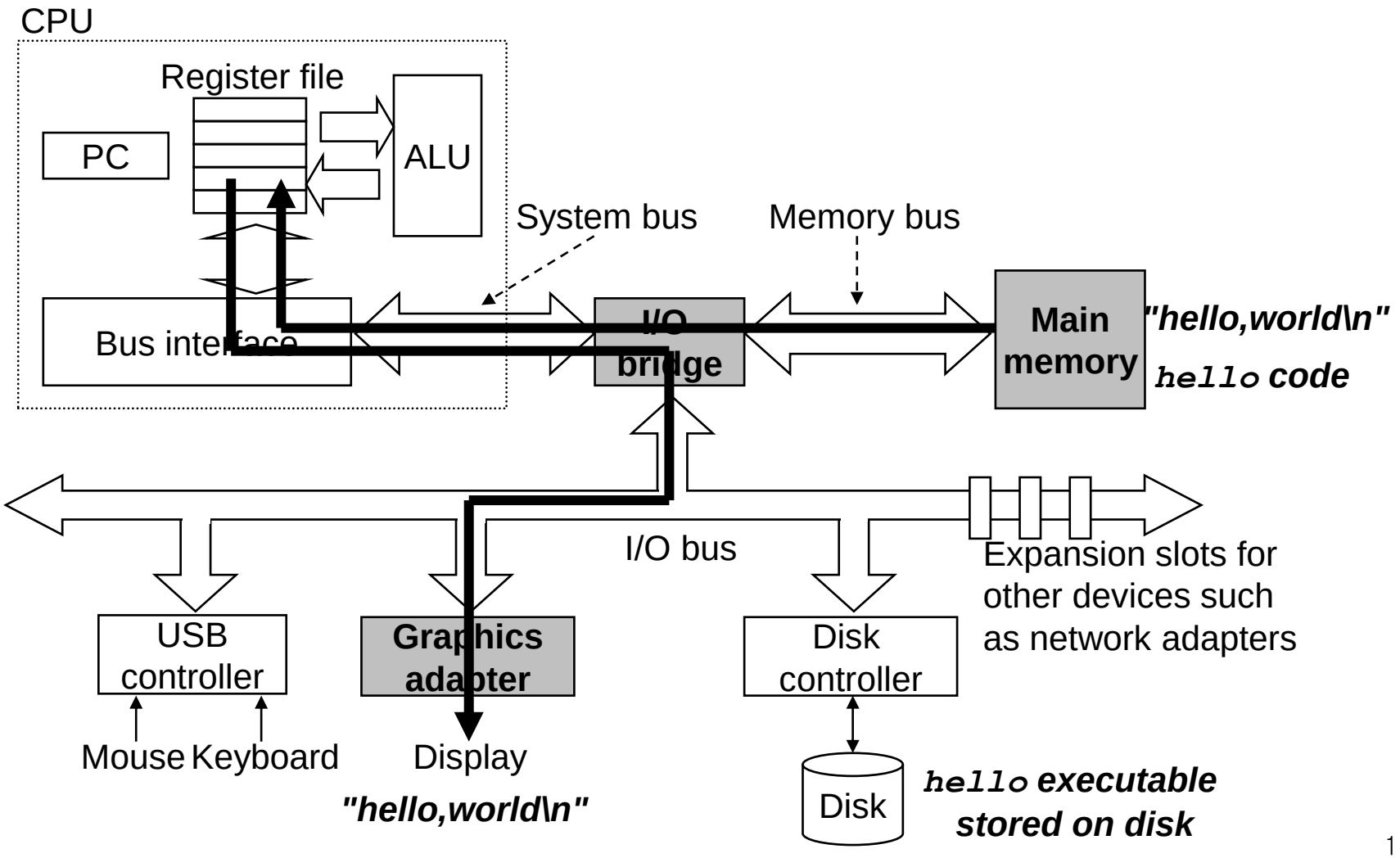
Running the hello program

After enter, the shell program load the executable “hello” file by executing a sequence of instructions that copies the code and data in the hello object file from disk to memory (using DMA)



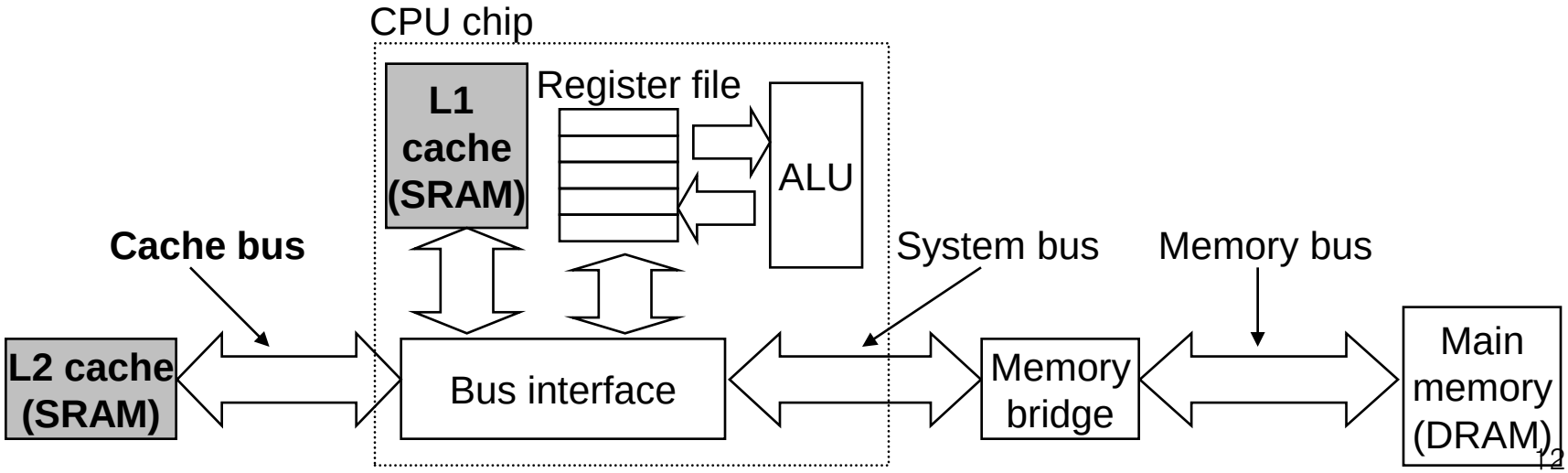
Running the hello program

Then, the processor begins executing the instructions in the hello program that copy the bytes in the “hello, world₩n” string from memory to the register file, and from there to the display device

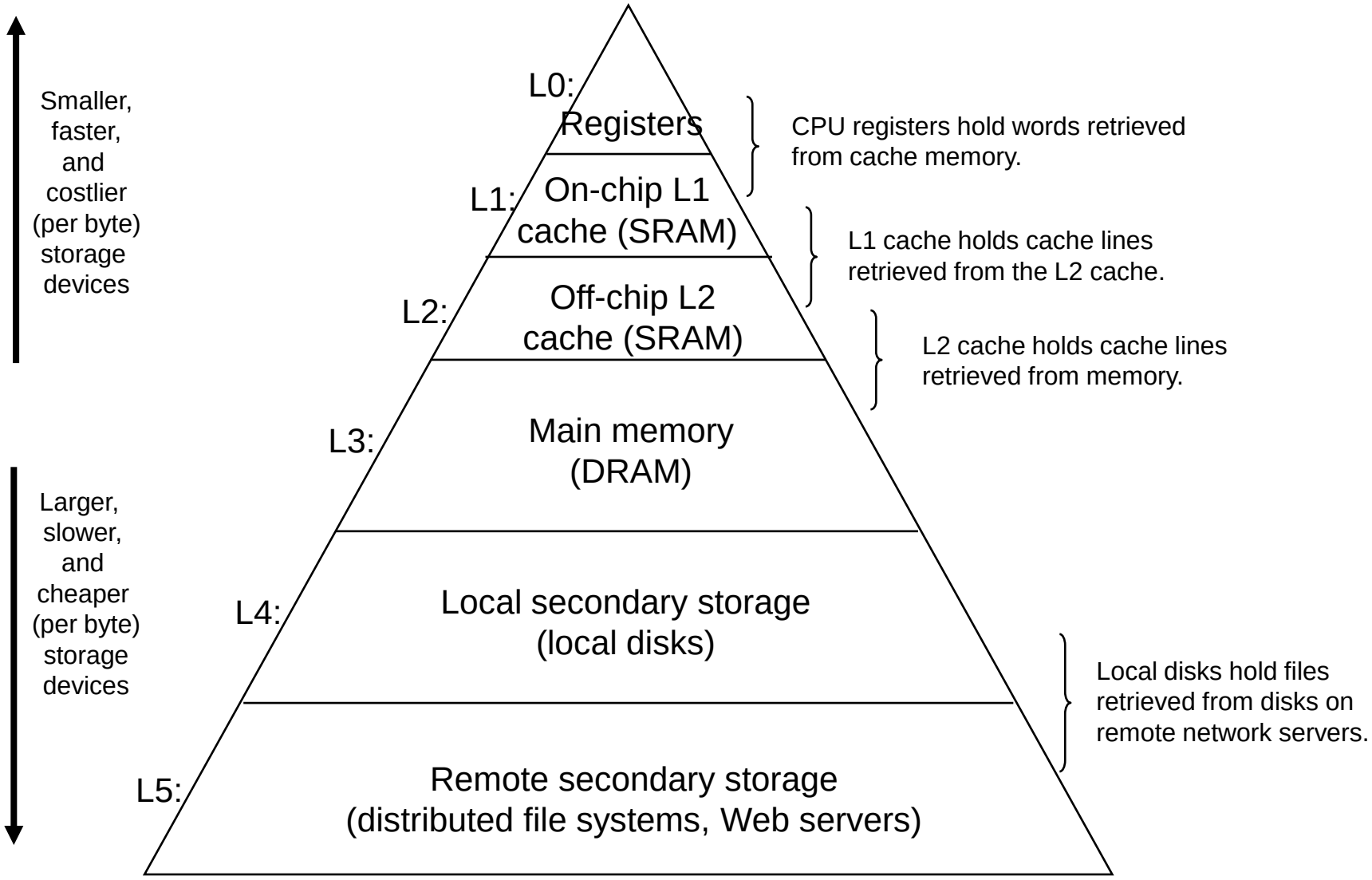


Caches Matter

- Program execution
  - moving information (code and data) from one place to another
  - Much of this copying is overhead that slow down the “real work” of the program
  - How to make these copy operations run as fast as possible?
- Physical laws
  - larger storage devices are slower than smaller storage devices
  - faster devices are more expensive to build than their slower counter part
  - e.g., disk drive 100 times larger than main memory (10,000,000 times longer read time than memory)
  - smaller faster storage devices (caches)



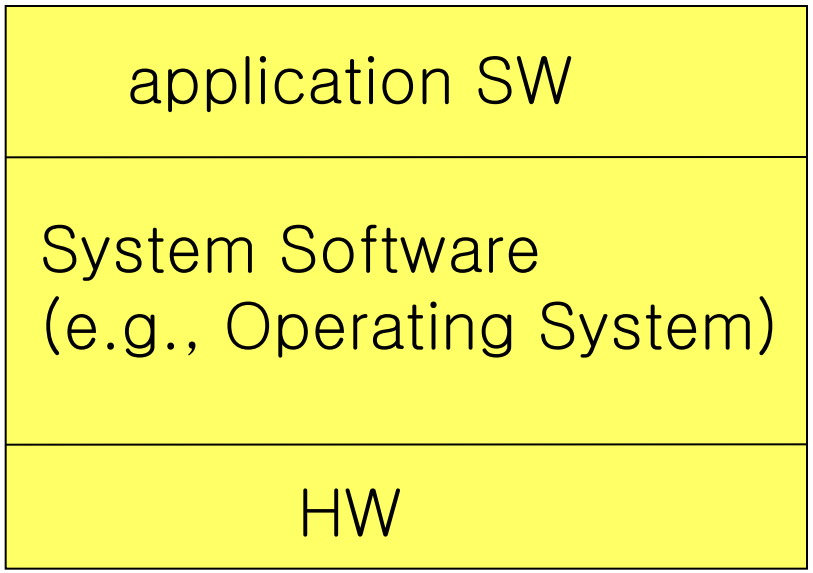
Generalizing the Cache Idea: Memory Hierarchy



- storage at one level serves as a cache for storage at the next lower level.

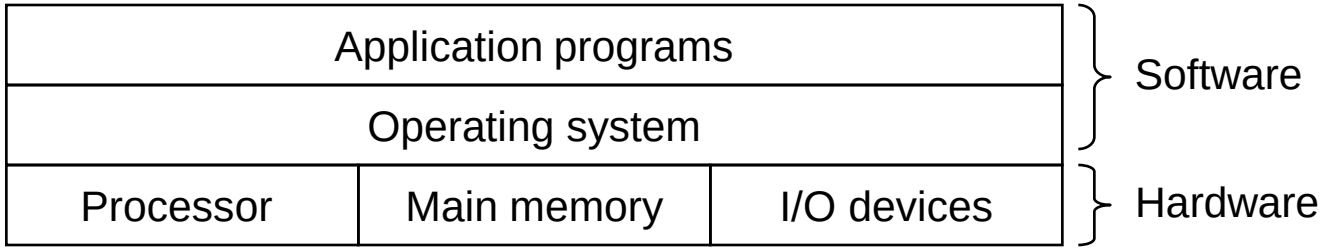
Application Program vs. HW

- Think about the scenario where application program directly control all HW resources
- Why we need pre-build system software layer (like operating system)?

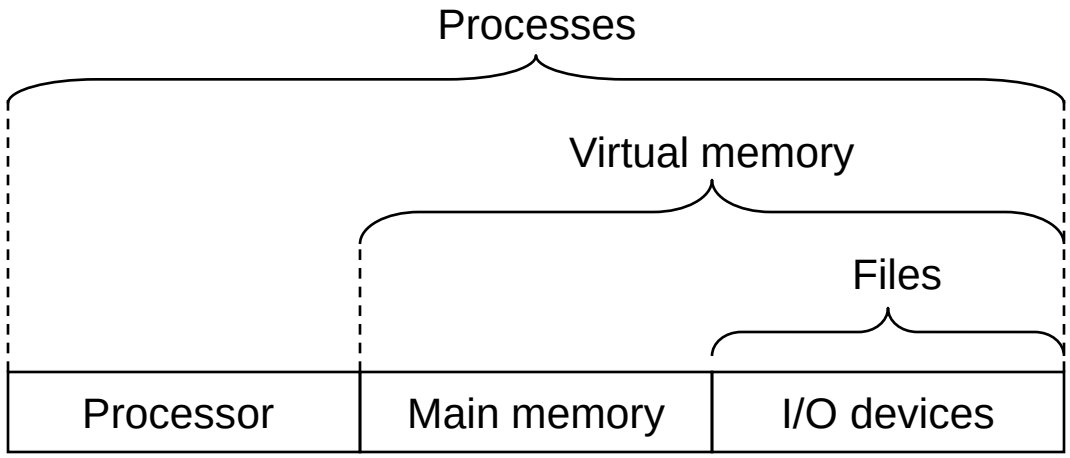


Operating System

- Note! our “hello” example does not directly access display or disk directly.
- Rather, it relies on the services provided by the operating system.
- OS’s two primary purposes
  - to protect the hardware from misuse by runaway applications
  - to provide applications with simple and uniform mechanisms for manipulating complicated and often widely different low-level hardware devices



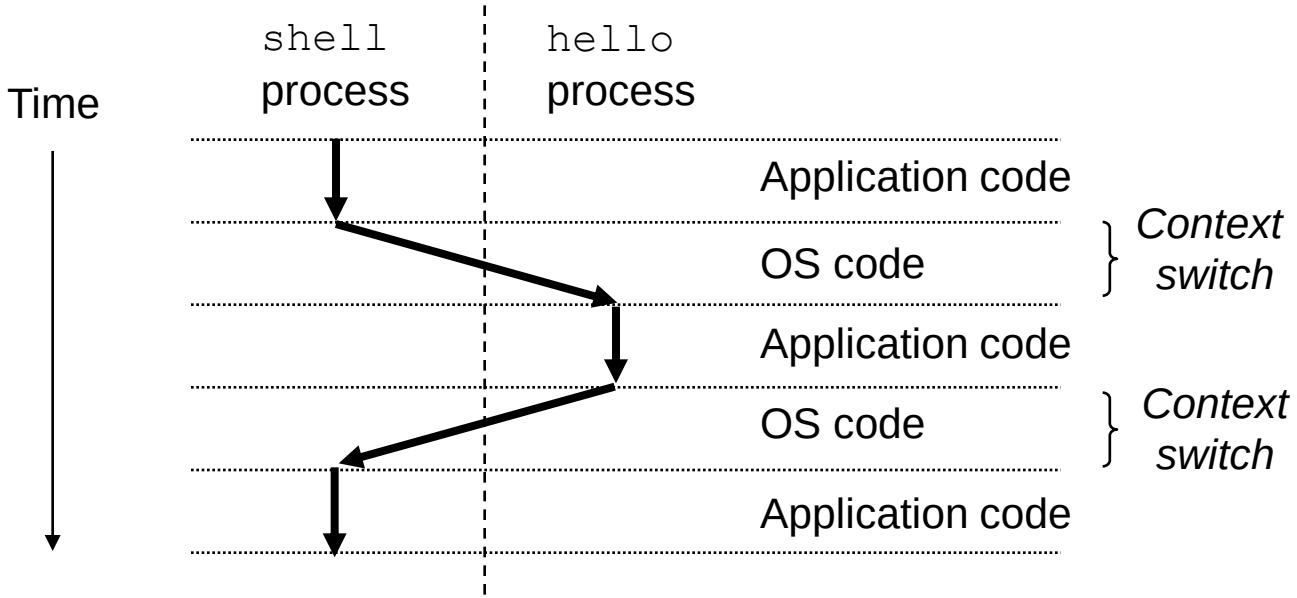
Abstractions provided by an Operating System



- Files: abstractions for I/O devices
- Virtual Memory: abstraction for both the main memory and disk I/O devices
- Processes: abstractions for the processor, main memory and I/O devices

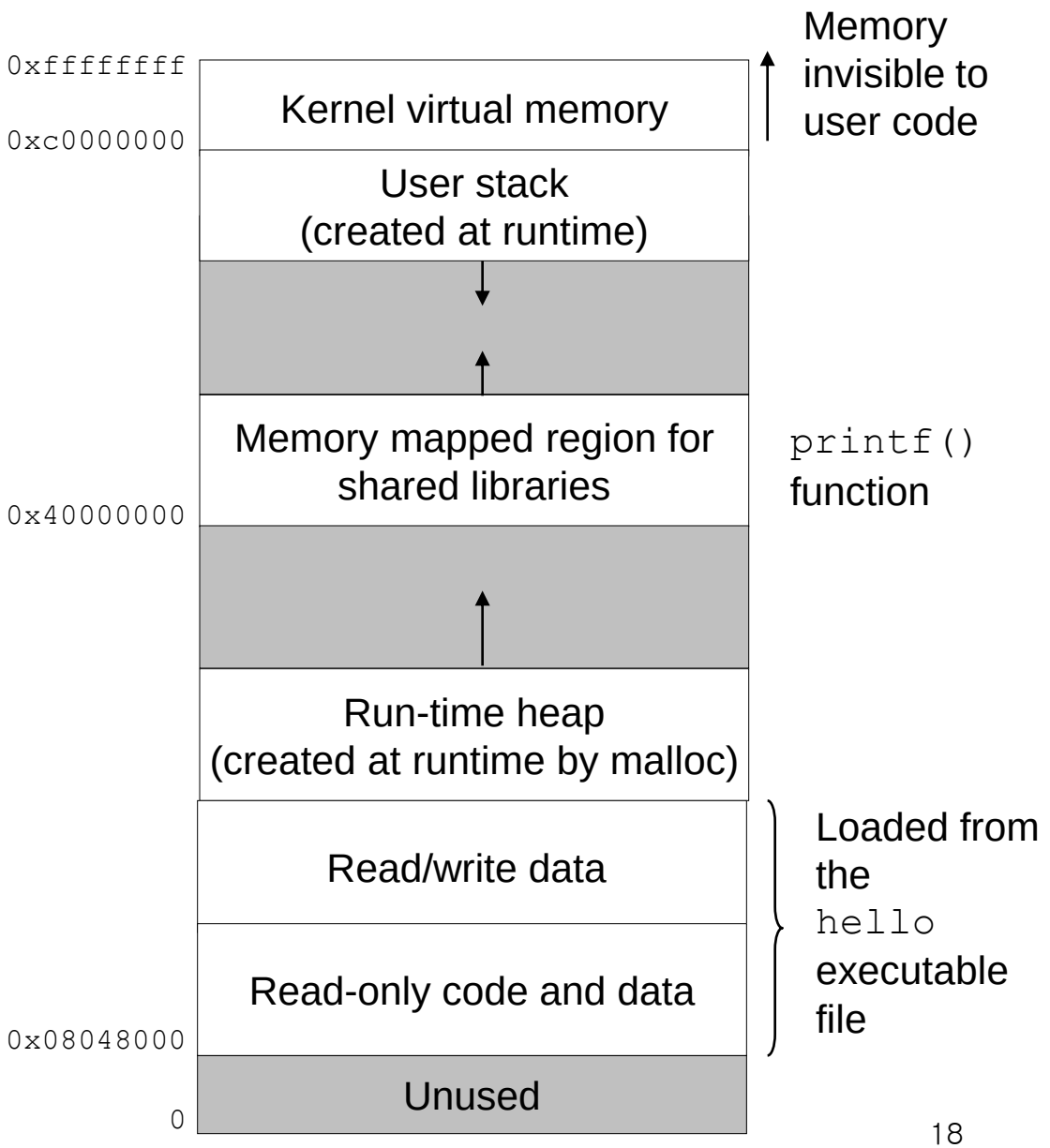
Processes

- A process is the operating system's abstraction for a running program
- The OS provides the illusion that the program is the only one running on the system
- Multiple processes can run concurrently on the same system, but each process appears to have exclusive use of the hardware
- To give this illusion, the OS saves the context (PC, register files, memory contents) and switches among contexts of different processes – context switches



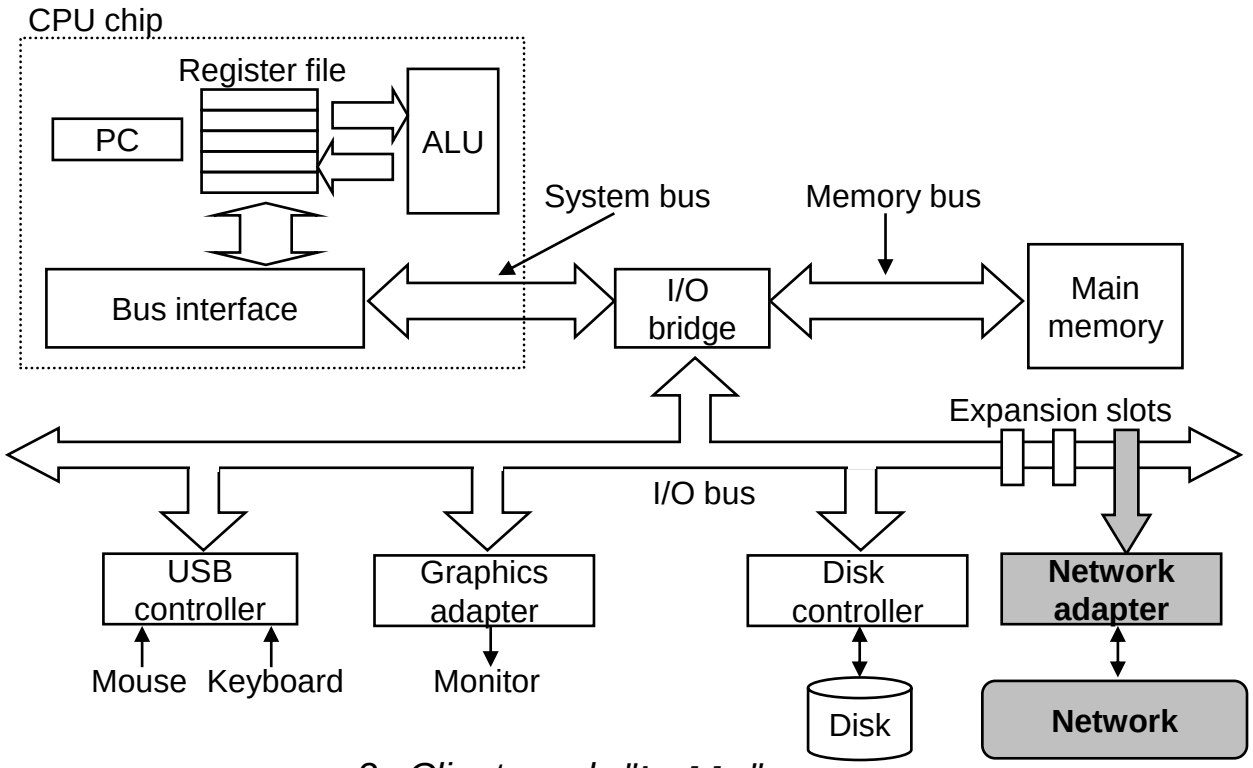
Virtual Memory

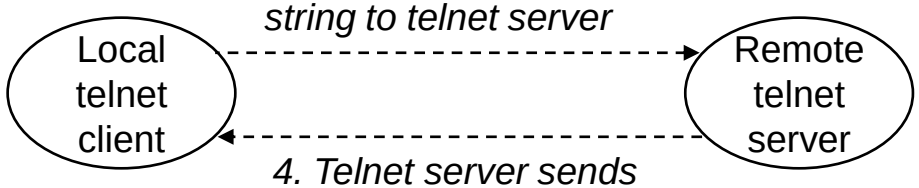
- Virtual memory is an abstraction that provides each process with the illusion that it has exclusive use of main memory.
- Each process has the same uniform view of memory, virtual address space.
- The basic idea is to store the contents of a process's virtual memory on disk and then load subset of the contents into the main memory as needed using the main memory as a cache for the disk



Networked Computer System

- The network can be viewed as just another I/O device (from the individual system's viewpoint) by the support of OS





1. User types "`hello`" at the keyboard

2. Client sends "`hello`" string to telnet server

3. Server sends "`hello`" string to the shell, which runs the `hello` program, and sends the output to the telnet server

4. Telnet server sends "`hello, world\n`" string to client

5. Client prints "`hello, world\n`" string on display

Summary

- Information in Computer Systems (bits)
- Transform the source code to the executable. What does it mean?
- What happens when we run a program?
  - Understanding the hardware organization
  - Understanding the information flow along the HW resources
- Performance consideration – memory hierarchy
- Marriage of application program and hardware – intermediate layer (system software)
- Operating system as a peace maker
  - processes
  - virtual memory
  - files
  - network