Computer System

ARQCP Course

Arquitetura de Computadores
Licenciatura em Engenharia Informática

2023/24

Paulo Baltarejo Sousa
pbs@isep.ipp.pt

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Disclaimer

Material and Slides

Some of the material/slides are adapted from various:

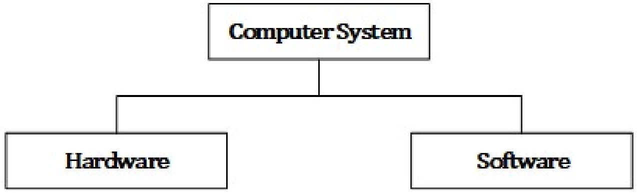
- Presentations found on the internet;
- Books;
- Web sites;
- ...

Outline

Introduction

What is?

- A **computer system** consists of hardware and system software that work together to run application programs.



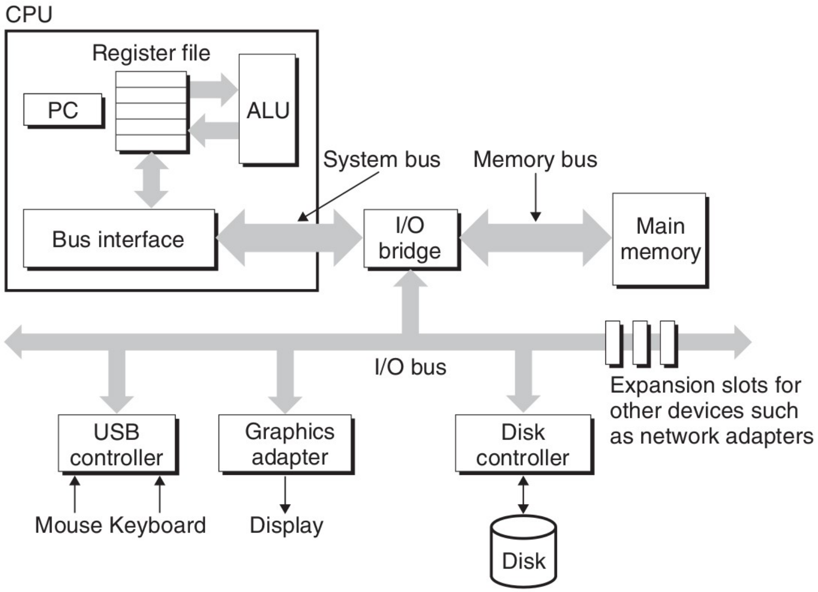
  - All physical components that forms computer system are known as computer **hardware**.
  - **Software** is basically collection of different programs that tells computer's hardware what to do.
- How these components work and interact **affects the correctness and performance** of application programs

How is it structured?

- Typically, it follows a **layer architecture**.

- Operating System, System Utilities, and Device Drivers are **out of scope of this course**.

Hardware Organization



Buses & I/O Devices

- Buses
  - Running throughout the system is a collection of electrical conduits that **carry bytes of information back and forth between the components**.
  - Buses are typically designed to transfer fixed-sized chunks of bytes known as **words**.
    - The number of bytes in a word (the **word size**) is a fundamental system parameter that varies across systems.
    - Most machines today have word sizes of either 4 bytes (32 bits) or 8 bytes (64 bits).
- I/O Devices
  - Input/output (I/O) devices are the system's connection to the external world.
  - Each I/O device is connected to the I/O bus by either a **controller** or an **adapter**
    - The purpose of each is to transfer information back and forth between the I/O bus and an I/O device.

Main Memory

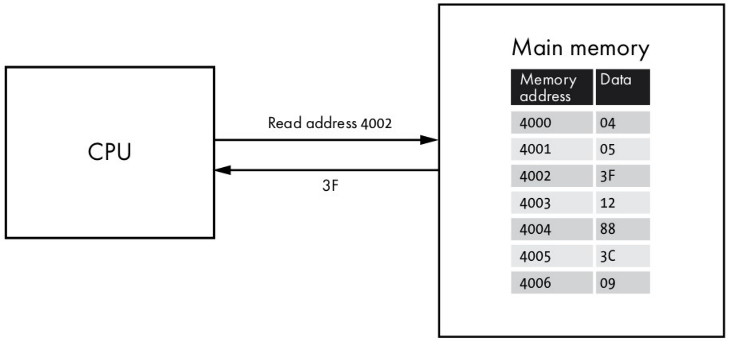
- The main memory is a **temporary storage device that holds both a program (instructions) and the data**.
- Logically, **memory is organized as a linear array of bytes**, each with **its own unique address** (array index) starting at zero.

Main memory

Memory address	Data
4000	04
4001	05
4002	3F
4003	12
4004	88
4005	3C
4006	09

Processor (I)

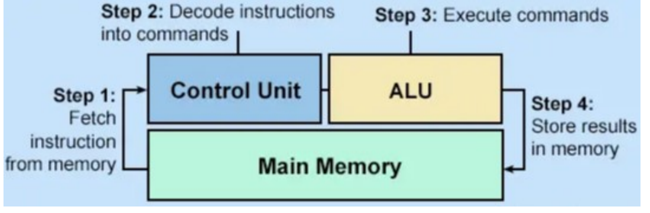
- The **central processing unit** (CPU), or simply processor, is the **engine that interprets (or executes) instructions stored in main memory**.



- At its core is a word-sized storage device (or register) called the **program counter (PC)**.
  - At any point in time, the **PC points at** (contains the address of) some **machine-language instruction in main memory**.

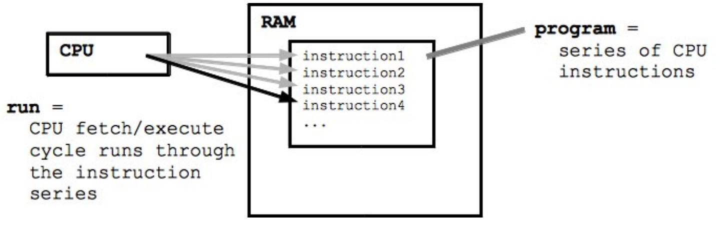
Processor (II)

- Instruction cycle
  - The **instruction cycle is the time required by the CPU to execute one single instruction**.
  - The **instruction cycle is the basic operation** of the CPU which consists on four steps:
    - **Fetch** the next instruction from memory
    - **Decode** the instruction just fetched
    - **Execute** this instruction as decoded
    - **Store** the result

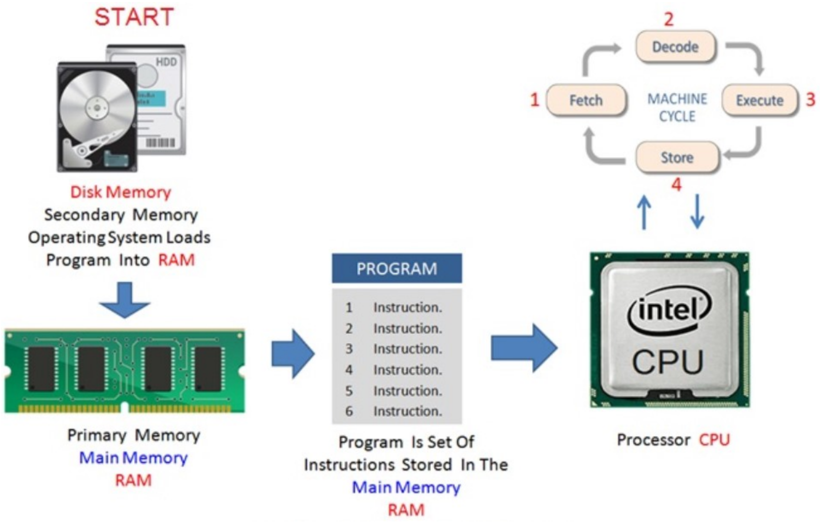


  - At the end of each instruction cycle **CPU advances PC register**.

- A computer **program is a file**, in which its content is a **set of CPU instructions**.
- The program **instructions are loaded into the main memory** (RAM).
- The CPU **initiates the program execution by fetching the instructions one by one from memory to registers**.
- The CPU executes these instructions **by repetitively performing an instruction cycle**.
- **At the end of each instruction cycle it increments the PC register**



How Computer Executes a Program (II)



Instruction Set Architecture (ISA)

- An ISA is part of the abstract model of a computer that defines how the CPU is controlled by the software.
- The ISA acts as an interface between the hardware and the software, specifying both **what the CPU is capable of doing** as well as **how it gets done**.
  - The ISA defines the **set of commands that the CPU can perform to execute the program instructions**.
    - Instructions are **bit-patterns.**
- Different “families” of processors, such as Intel x86-64, IBM/Freescale PowerPC, and the ARM processor family have **different ISAs**.
  - A program **compiled for one type of machine will not run on another**.

Creating the `hello` Program

Source code file (I)

- When you create a program, you tell the **computer what to do**.
- Using a text editor, you create what is called a **source code** file.
  - The only special thing about this file is that it has to contain **statements according to the programming language rules**.
    - In this case, the statements are written in the C programming language, so the source code file must be saved with ".c" extension (in this case `hello.c`)

```c
#include <stdio.h>

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

Source code file (II)

- The source code file is a **text file**, which **consist exclusively of American Standard Code for Information Interchange (ASCII) codes**.
- The ASCII code **associates an integer value for each symbol in the character set**, such as letters, digits, punctuation marks, special characters, and control characters.
  - The `hello.c` program is stored in a file as a sequence of bytes and each byte has an integer value that corresponds to some character.

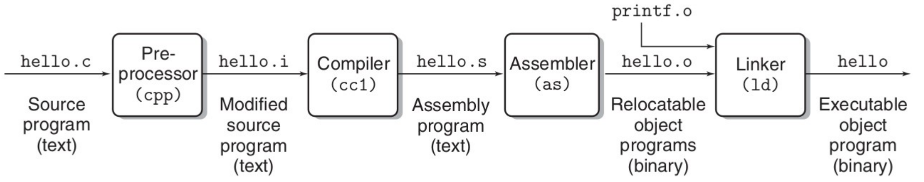
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
#	i	n	c	l	u	d	e	SP	<	s	t	d	i	o	.
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
h	>	\n	\n	i	n	t	SP	m	a	i	n	(	)	\n	{
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
\n	SP	SP	SP	SP	p	r	i	n	t	f	(	“	h	e	l
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	32
l	o	,	SP	w	o	r	l	d	\	n	“	)	;	\n	SP
32	32	32	114	101	116	117	114	110	32	48	59	10	125	10	
SP	SP	SP	r	e	t	u	r	n	SP	0	;	\n	}	\n	

Source code file (III)

- The `hello.c` program is a high-level C program because it can be read and understood by human beings, but not by processor.
- **Machine language**, or **machine code**, is the most basic set of instructions that a computer can execute.
- **Each type of processor** has its own set of **machine language instructions** (defined by ISA).
- To run `hello` program on the computer, the **C statements must be translated into a sequence of low-level machine-language instructions**.
- These instructions are then packaged in a form called **an executable object program and stored as a binary disk file**.
- This translation process is designated by **compilation**.
  - To perform compilation, it is required a **compiler**
    - A compiler is **a special computer program that translates a programming language's source code into machine code** (accordinga to ISA).

Compilation

- GNU Compiler Collection (GCC) is a **free and open source set of compilers and development tools** for C, C++ and other programming languages.
  - It is available for Linux, Windows and other operating systems.
- For compiling `hello` program
  > `gcc -o hello hello.c`
  - The gcc compiler **reads the source file `hello.c`** and **translates it into an executable object file `hello`**.
  - The translation is carried out **in a sequence of four stages**



Compilation Phases (I)

1. **Preprocessing phase.**
   - The preprocessor (`cpp`) modifies the original C program according to directives that begin with the # character.
     - For example, the `#include <stdio.h>` command in line 1 of `hello.c` tells the preprocessor to read the contents of the system header file `stdio.h` and insert it directly into the program text.
   - The result is another **text file**, source file C program, typically with the `.i` suffix.
2. **Compilation phase**.
   - The compiler (`cc1`) **translates the text file `hello.i` into the text file `hello.s`**, which contains an **assembly-language program**.
     - Each statement in an assembly-language program exactly describes one low-level machine-language instruction (according to the ISA) in a standard text form.

Compilation Phases (II)

3. **Assembly phase.**
   - The assembler (`as`) **translates `hello.s` into machine- language instructions**, packages them in a form known as a relocatable object program, and stores the result in the object file `hello.o`.
   - The `hello.o` file is **a binary file whose bytes encode machine language instructions (according to the ISA)** rather than characters.
4. **Linking phase.**
   - The linker (`ld`) links/**merges the object code with the library code to produce an executable file**.
     - Notice that our hello program calls the `printf` function, which is part of the standard C library provided by every C compiler.
     - The `printf` function resides in a separate precompiled object file called `printf.o`, which must somehow be merged with our `hello.o` program.
     - The result is the `hello` file, which is an executable object file (or simply executable) that is ready to be loaded into memory and executed by the system.

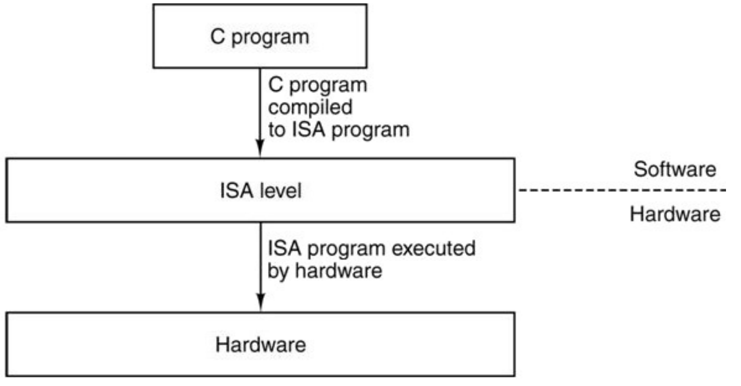
GCC compiler

- When you invoke `gcc`, it normally does preprocessing, compilation, assembly and linking.
- The `gcc` program accepts options that could change its default behavior.
- Using the **`-save-temps`** options you could save the compilation intermediate files (`*.i`, `*.s`, and, `*.o`)
  > `gcc -o hello hello.c -save-temps`

Executing the `hello` Program

An executable object file

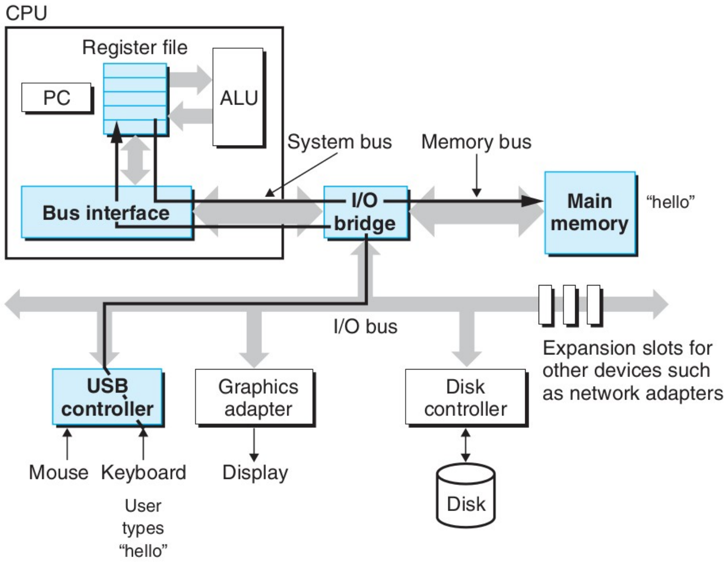
- The content of an executable object is a **set of CPU instructions** (according to ISA).



Executing (I)

- In order to run the executable file on a Unix derived system, we type its name to an application program known as a **shell**:
  `> ./hello`
  - **A shell is a program that takes commands from the keyboard and gives them to the operating system to perform**.
- As we type the characters `./hello` at the keyboard, the shell program reads each one into a register, and then stores it in memory.
- When we hit the `enter` key on the keyboard, the shell knows that we have finished typing the command.

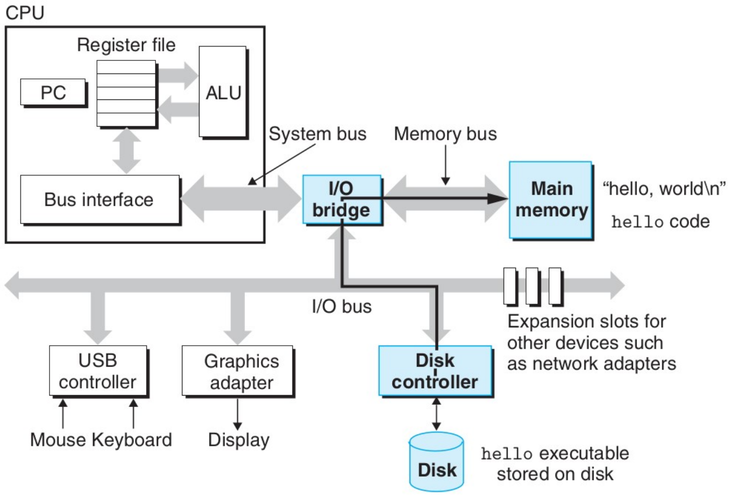
Executing (II)



Executing (III)

- A set of operations are executed by operating systems that copies the contents of the `hello` object file from disk to main memory.
  - Using a technique known as direct memory access (DMA), the data travels directly from disk to main memory, without passing through the processor.

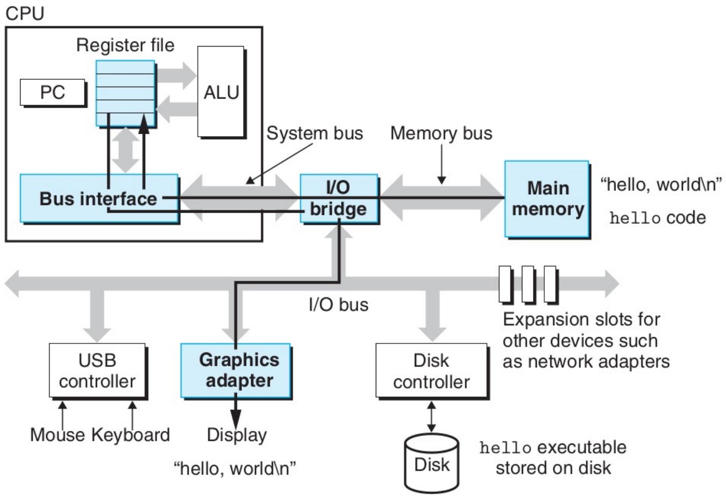
Executing (IV)



Executing (V)

- Once the contents of the `hello` **object file are loaded into memory**, the **processor begins executing the machine-language instructions** in the `hello` program's `main` routine.
  - These instructions copy the bytes in the `hello, world\n` string from memory to the register file, and from there to the display device, where they are displayed on the screen.

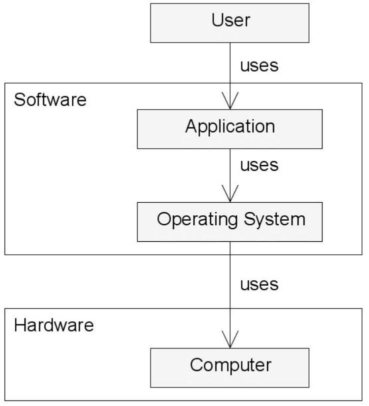
Executing (VI)



Application Execution

Operating System (OS) [1] (I)

- When the shell loaded and ran the `hello` program, and when the `hello` program printed its message, neither program accessed the keyboard, display, disk, or main memory directly.
  - Rather, they relied on the services provided by the OS.
- The OS has 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 wildly different low-level hardware devices.



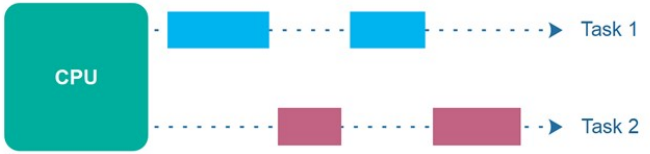
[1] OS issues are out of course scope

OS (II)

- A process is the OS's **abstraction for a running program**.
  - Whenever a program file (a file containing machine code) is loaded to be executed the OS creates a process.
- A process **is an active program and related resources**.
  - From the OS's point of view, the purpose of a **process is to act as an entity to which system resources (CPU time, memory, etc.) are allocated**.
- It provides two virtualisations, **giving the illusion that it alone monopolizes the system.**
  - **Virtualised processor**.
  - **Virtualised memory**.

Virtualised processor

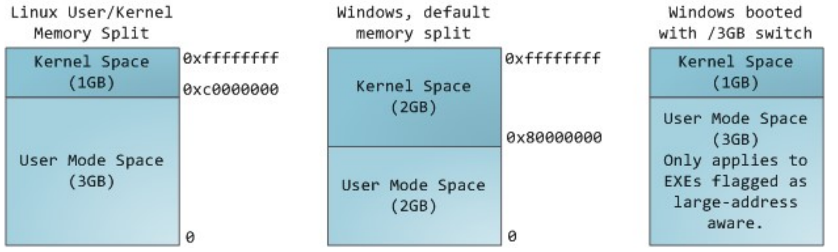
- Multiple processes **can run concurrently on the same system, and each process appears to have exclusive use of the hardware**, such as CPU.



  - The instructions of **one process are interleaved with the instructions of another process**.
  - Generally, **there are more processes to run than there are CPUs to run them**.
    - The OS performs this interleaving with a mechanism known as **context switching**.

Virtualised Memory

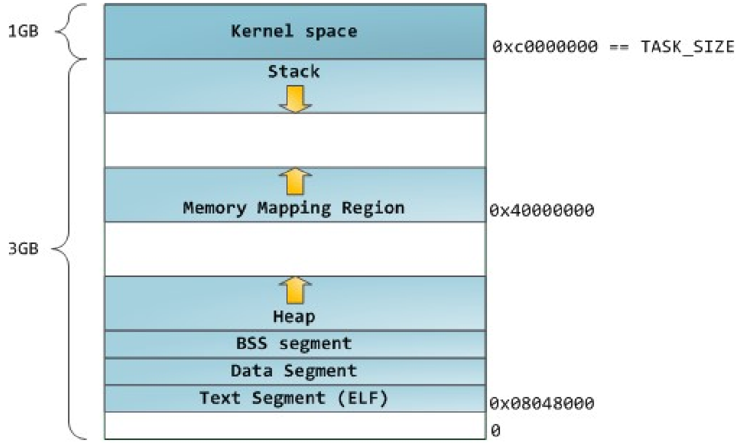
- Each process **runs in its own memory sandbox (virtual memory)**.
- Virtual memory is **an abstraction that provides each process with the illusion that it has exclusive use of the main memory**.



- **OS Kernel code and data are always addressable**, ready to handle interrupts or system calls at any time.

Virtual Memory Layout

- Each process has the same uniform view of memory, which is known as its **virtual address space**.



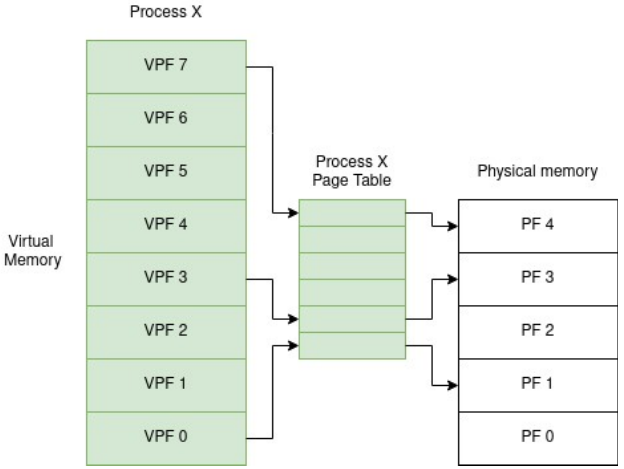
Memory Paging

- Paging is a memory management technique [2] in which the memory is divided into fixed size chunk of bytes, called **pages**.
- **Paging** is a memory management scheme:
  - That eliminates the need for a contiguous allocation of physical memory.
  - That improves the efficiency by moving pages in and out of memory as needed.

[2] There are other techniques such as segmentation.

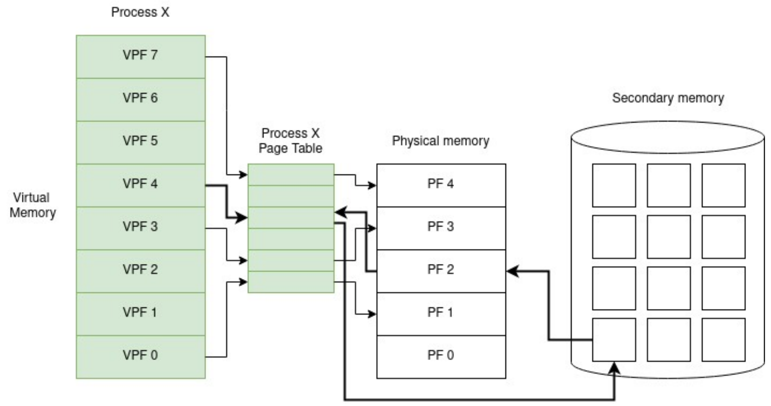
Memory Mapping (I)

- These **virtual addresses are mapped to physical memory by page tables**, which are maintained by the OS and consulted by the CPU.



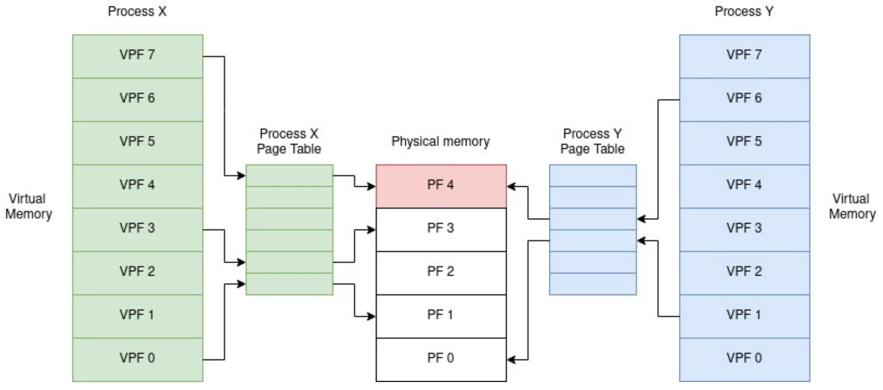
Memory Mapping (II)

- When a process needs to access a memory location that is not in physical memory, the **page table entry** for that location indicates that a **page fault** has occurred.
- The OS loads the required page into physical memory from disk, updating the page table accordingly.

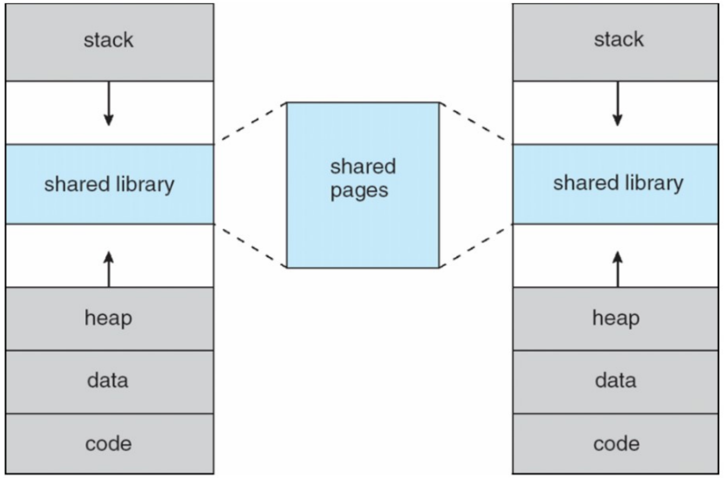


Memory Mapping (III)

- Each process has its page table.
- This mechanism allows memory to be **shared across several processes**.



Shared Memory (I)



Shared Memory (II)