3150 - Operating Systems

Dr. WONG Tsz Yeung

Chapter 1
Overview of an Operating System

About this course

- Programming-oriented
  - **C & English** are the languages used in lectures;
  - **C & C++** are the languages used in assignments;

- Topics covered:
  - Process Management: 6 weeks;
  - File System: 4 weeks;
  - Memory Management: 3 weeks;

About this set of handout

- **Extra** : means materials would be excluded in final examination.
- **Challenge** : means a programming challenge for you; not included in final examination.
- "**`Consolas font`**" is used for representing program codes and commands.
- The **"light blue color"** is used for representing kernel computations or kernel-related activities.

What is an Operating System?

- According to your experience…
  - Networking;
  - Storage;
  - Multimedia;
  - Gaming;
  - What else?

What will you learn?

- None of the above were about the operating system!

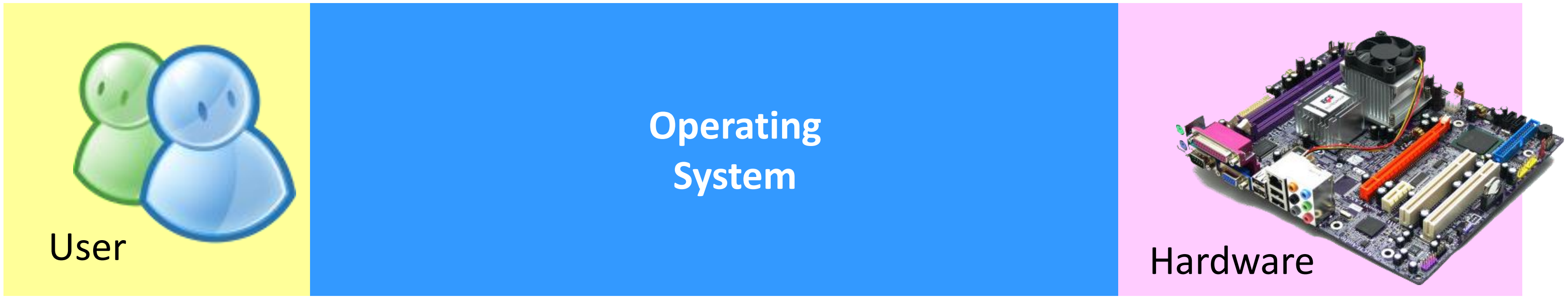
- You're going to learn the core of an operating system.
  - Through the course, you'll, at least, explore the answer of the following 2 questions:

Why can't we save an 4.7GB DVD ISO file onto a 8GB USB drive?

Why does the system not performing faster at a factor of 2 when I upgrade by computer from a Dual-Core CPU to an Quad-Core CPU at the same internal clock speed?

What is an Operating System?

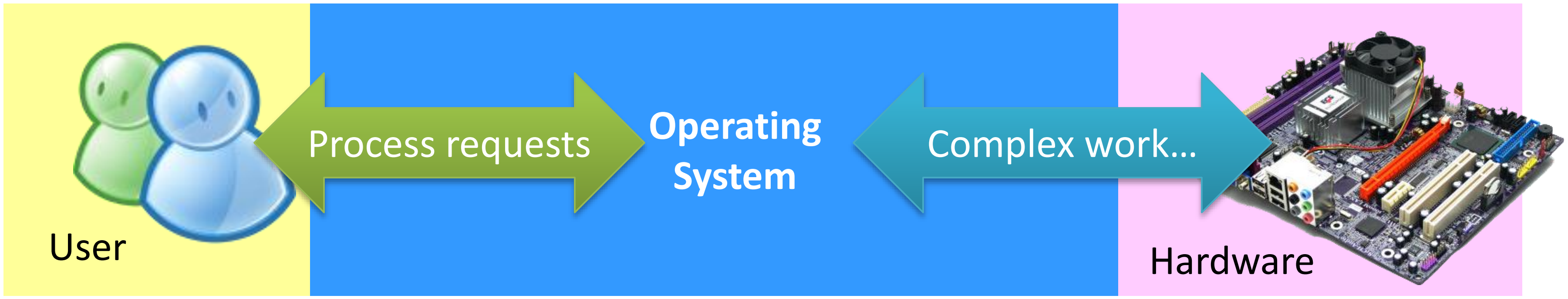
- Let’s start understanding an OS from this question: **Where is it?**
  - It stands **between** the hardware and the user.



What is an Operating System?

- How good is this design?
  - The user does not have to program the hardware directly.
    - It hides *<u>all the troublesome operations</u>* of the hardware.

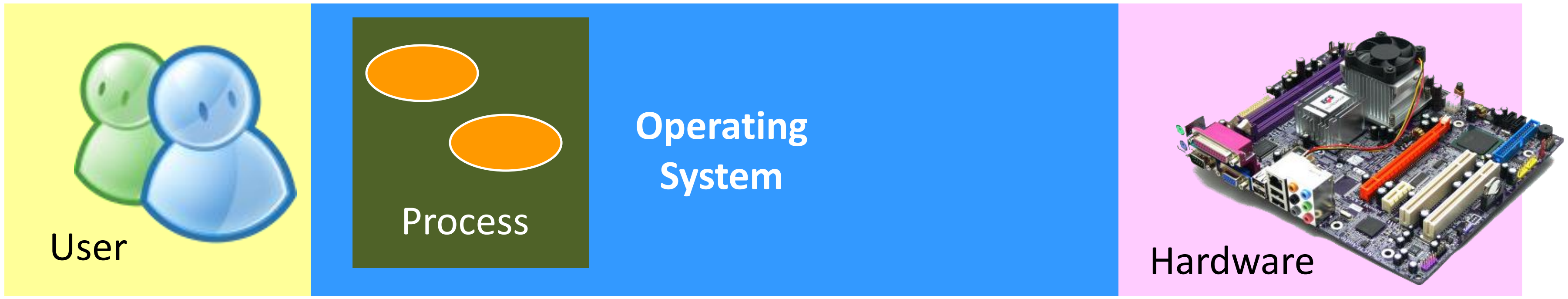
<u>Example</u>. The OS, on one hand, hides the physical system memory away from you. On the other hand, it tells you that there is system memory available when you run your applications.



What is an Operating System?

- **Processes** as the starting point!
  - Whatever programs you run, you create processes.
    - i.e., you need processes to open files, utilize system memory, listen to music, etc.

  - So, process lifecycle, process management, and other related issues are essential topics of this course.

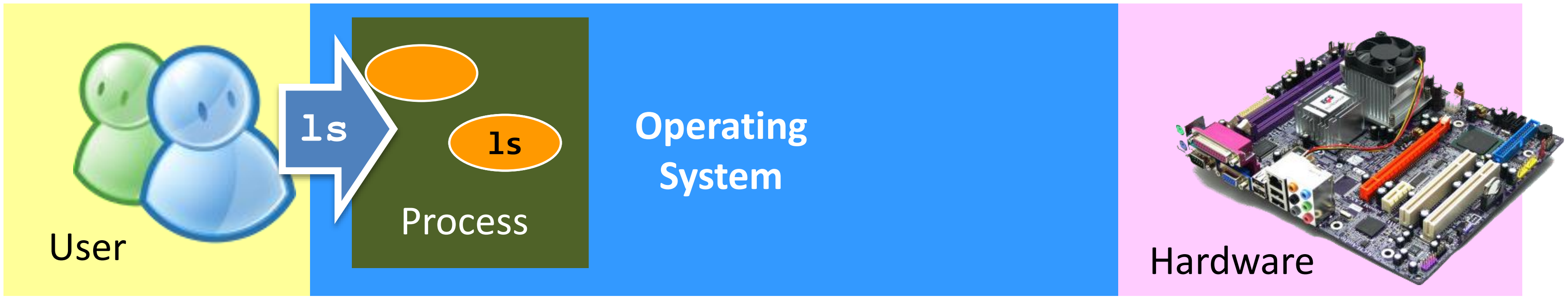


What is an Operating System?

- Example (step 1)

```
$ ls
```

Most commands you type in the **shell** are the same as starting a new process.



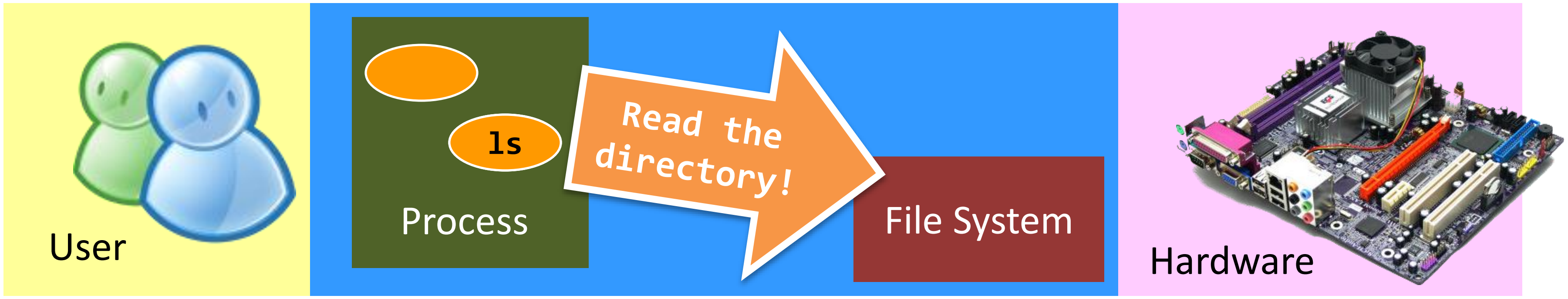
What is an Operating System?

- Example (step 2)

```
$ ls
```

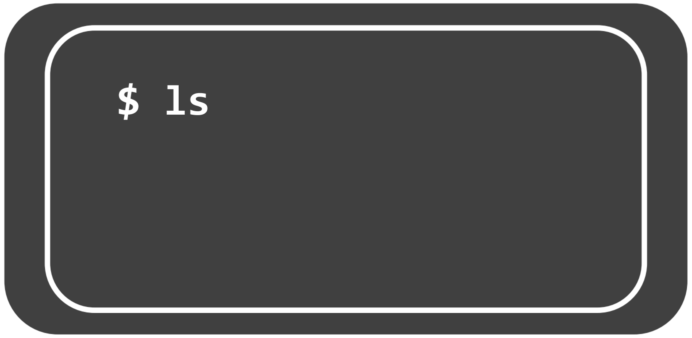
The operating system contains the codes that are needed to work with the file system.

The codes are called the **kernel**.



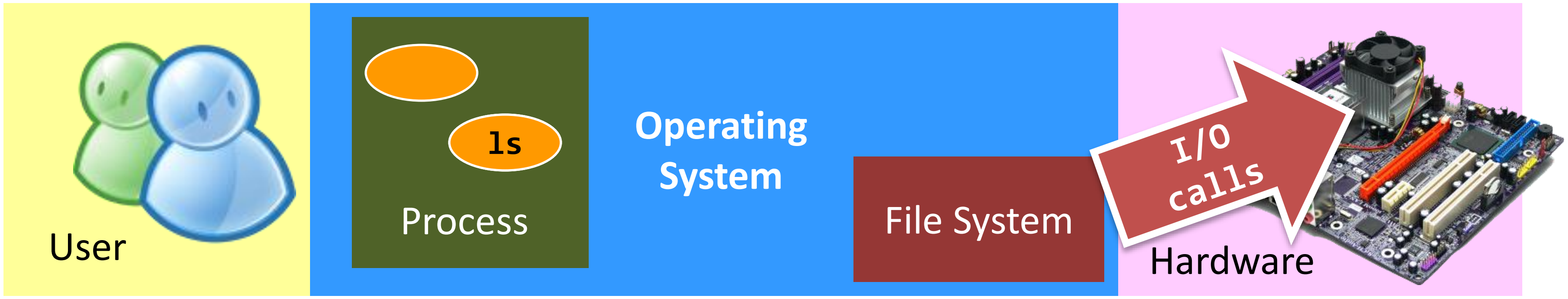
What is an Operating System?

- Example (step 3)



```
$ ls
```

The file system module inside the operating system knows how to work with devices, using device drivers.



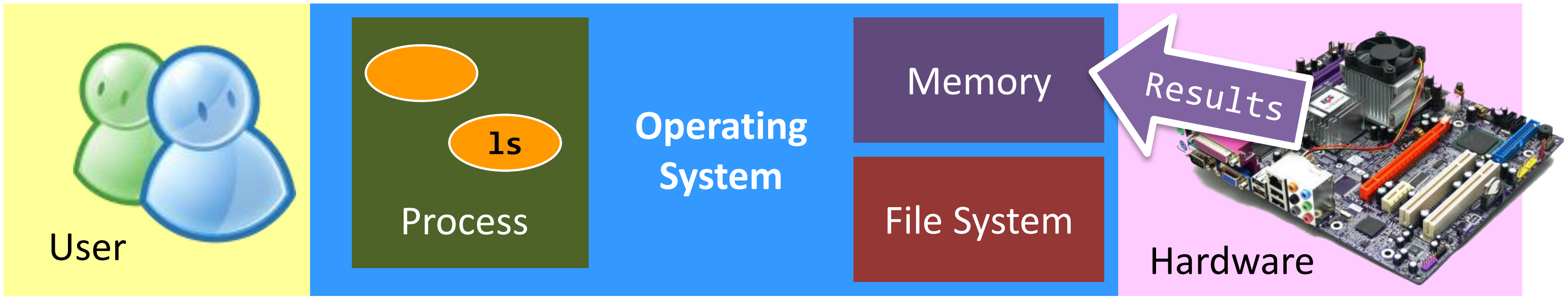
What is an Operating System?

- Example (step 4)

```
$ ls
```

Of course, the operating system will allocate memory for the results.

CE students: do you know what is *DMA – direct memory access*?



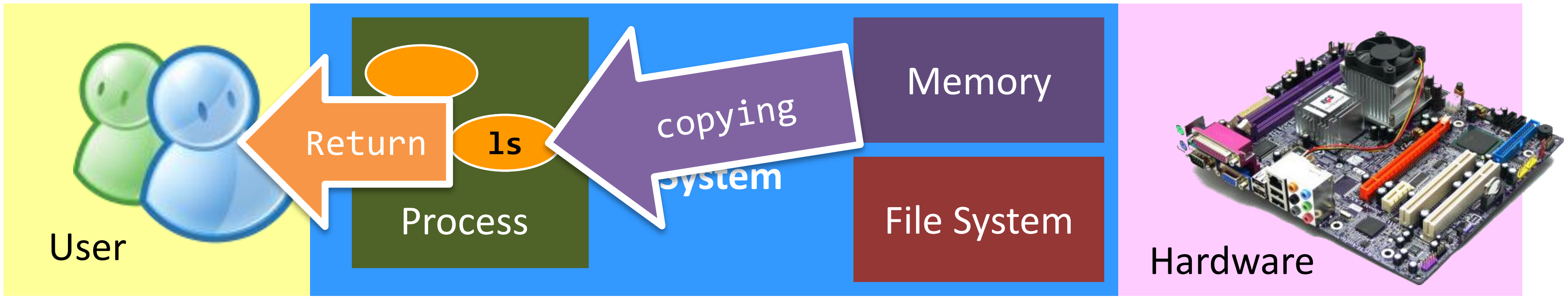
What is an Operating System?

- Example (final step)

```
$ ls
.   ..   index.html
$ _
```

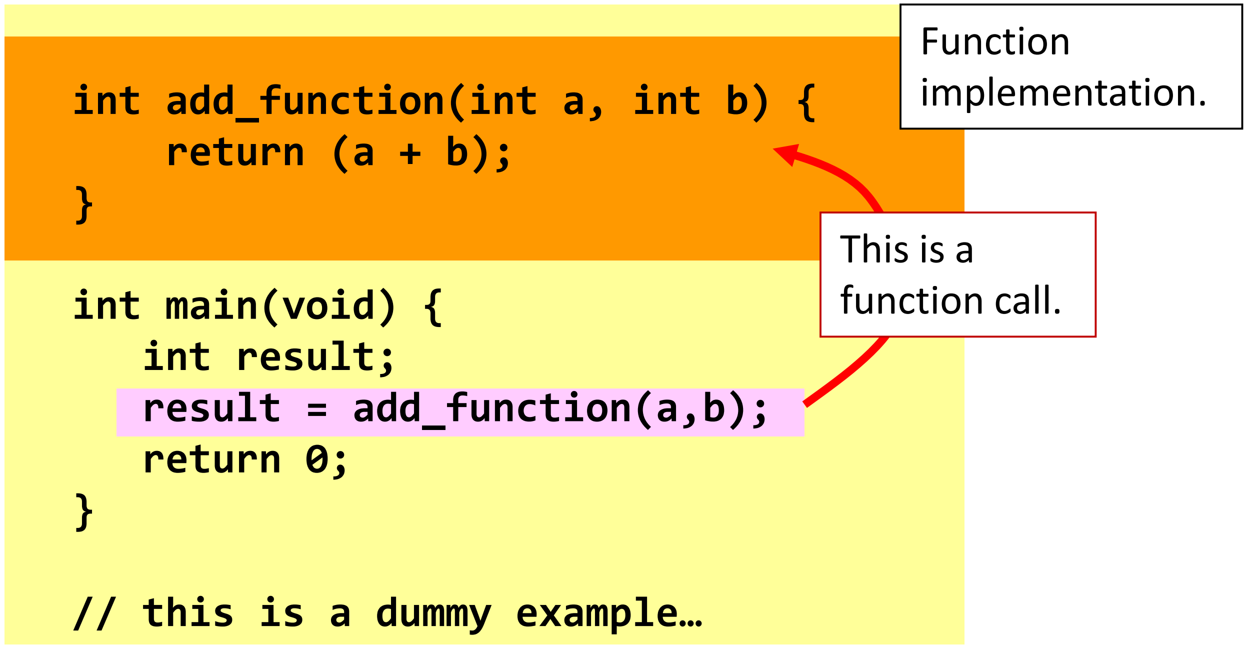
The memory management sub-system will copy the result to the memory of the process.

At last, the result returns.



Interacting with the OS

- System call
  - Informally, a system call is similar to a function call, but…
  - The function implementation is inside the OS.
  - We name it the **OS kernel**.

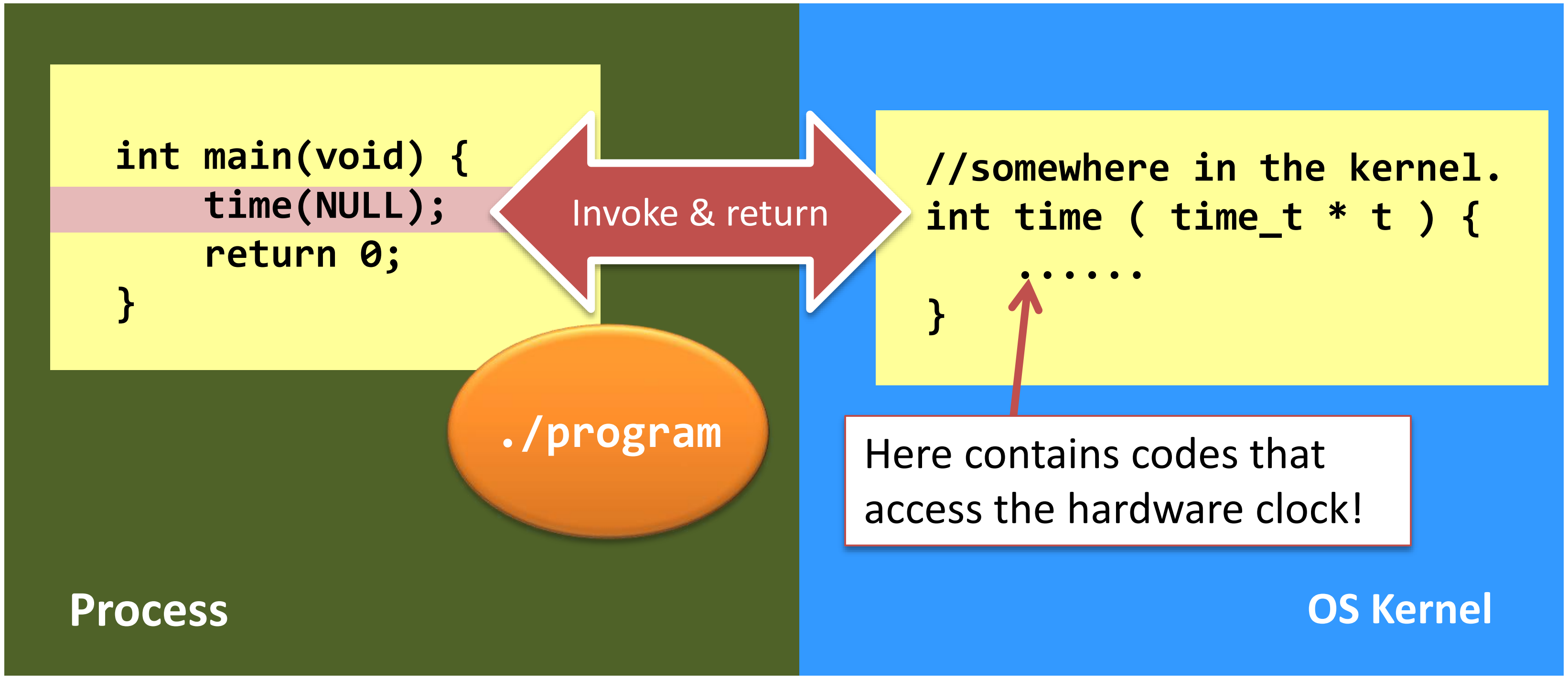


```
int add_function(int a, int b) {
    return (a + b);
}

int main(void) {
   int result;
   result = add_function(a,b);
   return 0;
}

// this is a dummy example…
```

Interacting with the OS



```
int main(void) {
    time(NULL);
    return 0;
}
```

```
//somewhere in the kernel.
int time ( time_t * t ) {
    ......
}
```

Extra information:

The kernel is *not an executing entity*. Rather, it is just a bunch of compiled codes and allocated memory.

System calls

- System calls are the **programming interface** between processes and the OS kernel.

- The system calls are usually
  - **primitive**,
  - **important**, and
  - **fundamental**.
  - e.g., the **`time()`** system call.

- Roughly speaking, we can categorize system calls as follows:

Process	**File System**	**Memory**
Security	**Device**	

In this course, we focus on the first three areas!

System calls

- How can we know if a "function" is a system call?
  - Read the man page "**syscalls**" under Linux.

- Without reading the manpages, guess which of the following is/are system call(s)?

Name	Yes/No?
`printf() & scanf()`	No
`malloc() & free()`	No
`fopen() & fclose()`	No
`mkdir() & rmdir()`	Yes
`chown() & chmod()`	Yes

Who are they?

System calls VS Library function calls

- If a call is not system calls, then they are **library calls** (or function calls)!

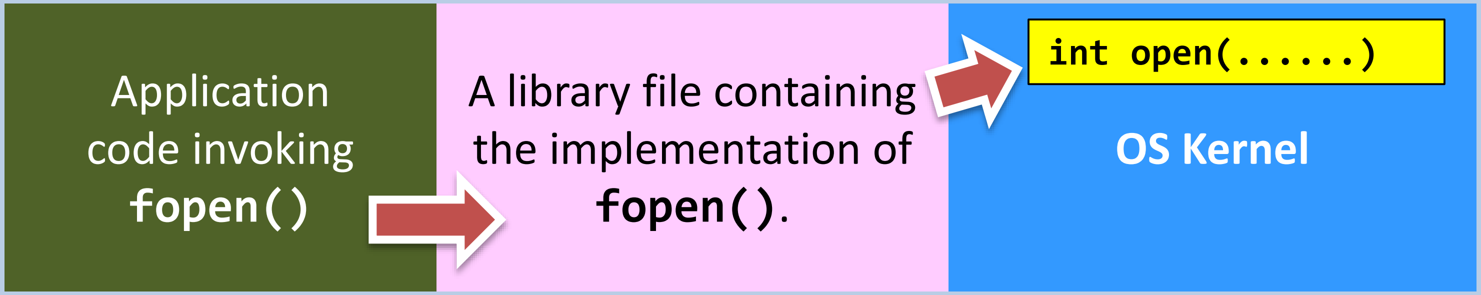
- Take **fopen()** as an example.
  - **fopen()** invokes the system call **open()**.
  - So, why people invented **fopen()**?
  - Because **open()** is too primitive and is not programmer-friendly!

Library call	`fopen(“hello.txt”, “w”);`
System call	`open(“hello.txt”, O_WRONLY | O_CREAT | O_TRUNC, 0666);`

System calls VS Library function calls

- Library functions are usually compiled and packed inside an object called the **library file**.
  - In windows: DLL – dynamically linked library.
  - In Linux: SO – shared objects.

Big picture



Application code invoking **fopen()**
A library file containing the implementation of **fopen()**.
int open(......)
OS Kernel

OS standards

- Who defines the system calls? Functionalities? Arguments? Return values?

- There are standards!

Standards	Full Name	Example OS
POSIX	Portable Operating System Interface	Linux
BSD	Berkeley Software Distribution	Mac OS Darwin
SVR4	System V (five) Release 4	Solaris Unix

So, what I asked is "*what is an OS*"?

- Now, you know that an OS is …
  - **a piece of software**;
  - **a resource manager**, which manages all the physical devices, and
  - **a service provider**, which provides a set of programming interfaces for processes to access to the resources.
- Although, the OS is controlling everything…
  - **It does not control you!**
  - Through this course, you will learn:
    - what are the capabilities of an OS, and
    - what are the limits.
- At the end, you will have a better control over the OS and the programs you develop on it.

Introduction to Operating System Components

Process

Process OR Program?

- A process is not a running program!

Let's consider the following two commands

Command A	**`ls -R /`**	Recursively print the directory entries, starting from the directory '/'
Command B	**`ls -R /home`**	Recursively print the directory entries, starting from the directory '/home'

Similarity	Difference
Both use the program file "**`/bin/ls`**".	The program arguments are different.
---	The processes' internal status are different, such as running time.

Program != Process

- A process is an execution instance of a program.
  - More than one process can execute the same program code
  - Later, you'll find that a process is not bounded to execute just one program!

- A process is active.
  - A process has its local states concerning the execution. E.g.,
    - which line of codes it is running;
    - which CPU core (if there are many) it is running on.
  - The local states change over time.

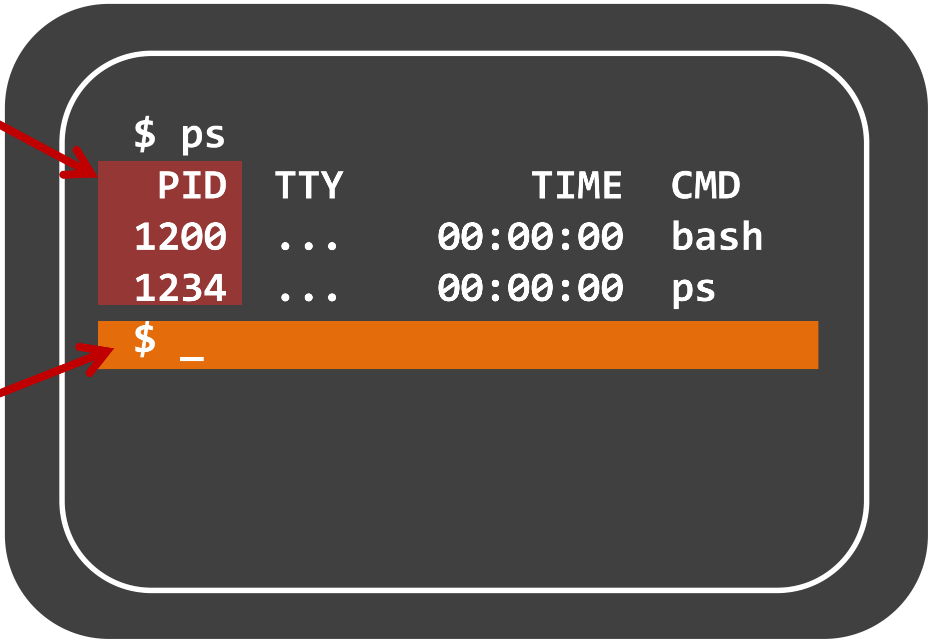
- Commands about processes (and hopefully you've tried them before) – e.g., `ps` & `top`.

Process-Related Tools

- The tool "**ps**" can report a vast amount of information about every process in the system
  - Try "`ps -ef`".

This column shows the unique identification number of a process, called **Process ID**, or PID for short.

Hint: you can treat **ps** as the short-form of "**process status**"

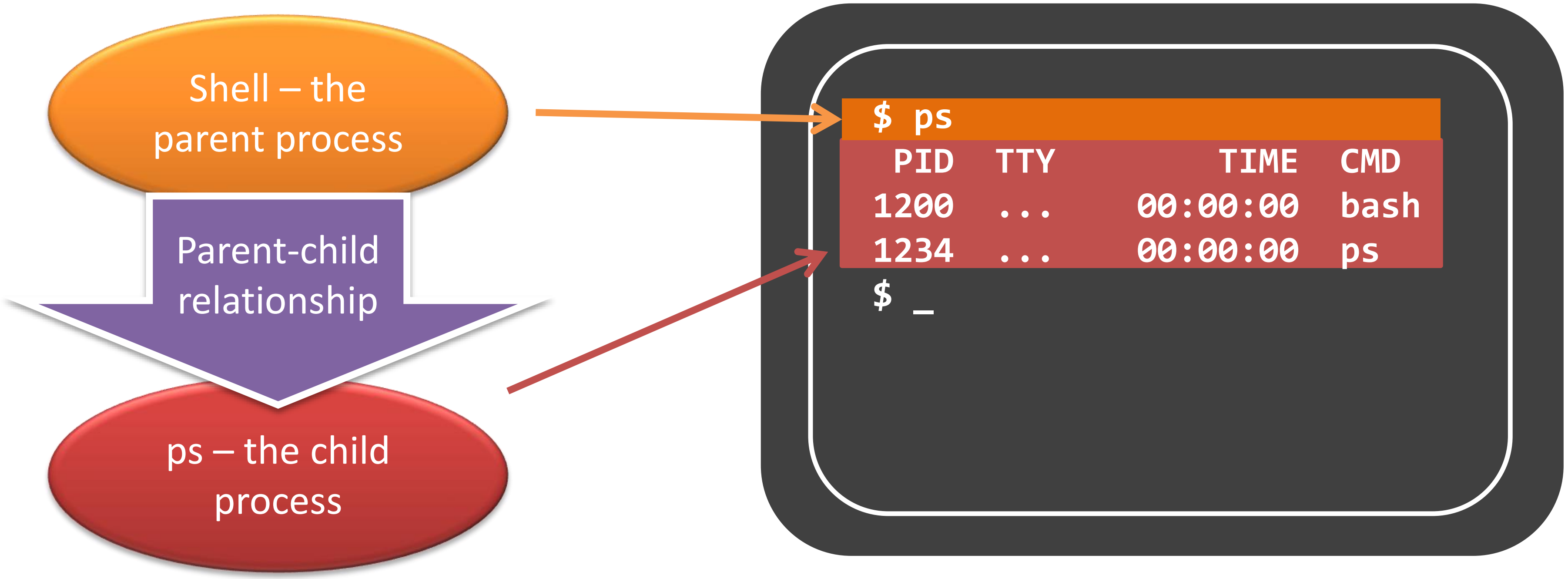


```
$ ps
  PID  TTY          TIME  CMD
 1200  ...      00:00:00  bash
 1234  ...      00:00:00  ps
$ _
```

By the way, this is called **shell**.

Shell – a process launching pad

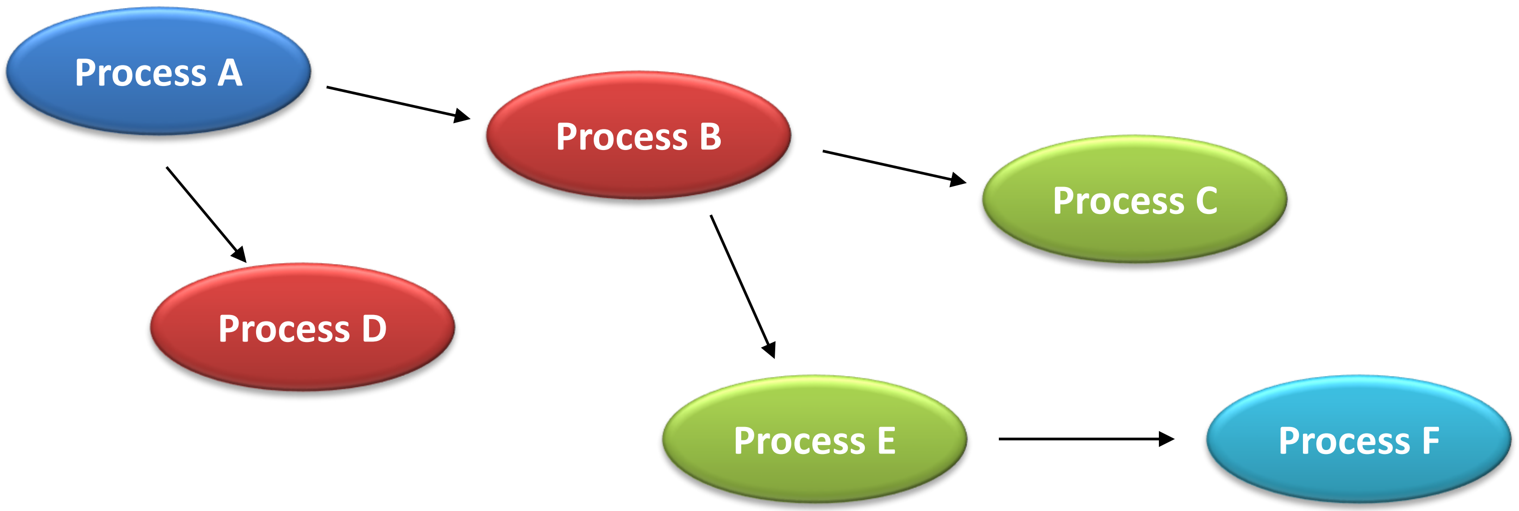
- So, what is going on inside that shell?
  - The shell creates a new process, and is called a **child process** of the shell.
- The child process then executes the command "**ps**".



```
$ ps
  PID  TTY          TIME   CMD
 1200  ...      00:00:00   bash
 1234  ...      00:00:00   ps
$ _
```

Process hierarchy

- Process relationship:
  - A parent process will have its child process.
  - Also, a child process will have its child processes.
  - This form a **tree hierarchy**.



E.g., “Process E” is the shell and “Process F” is “**ps**”.

What will we learn about processes?

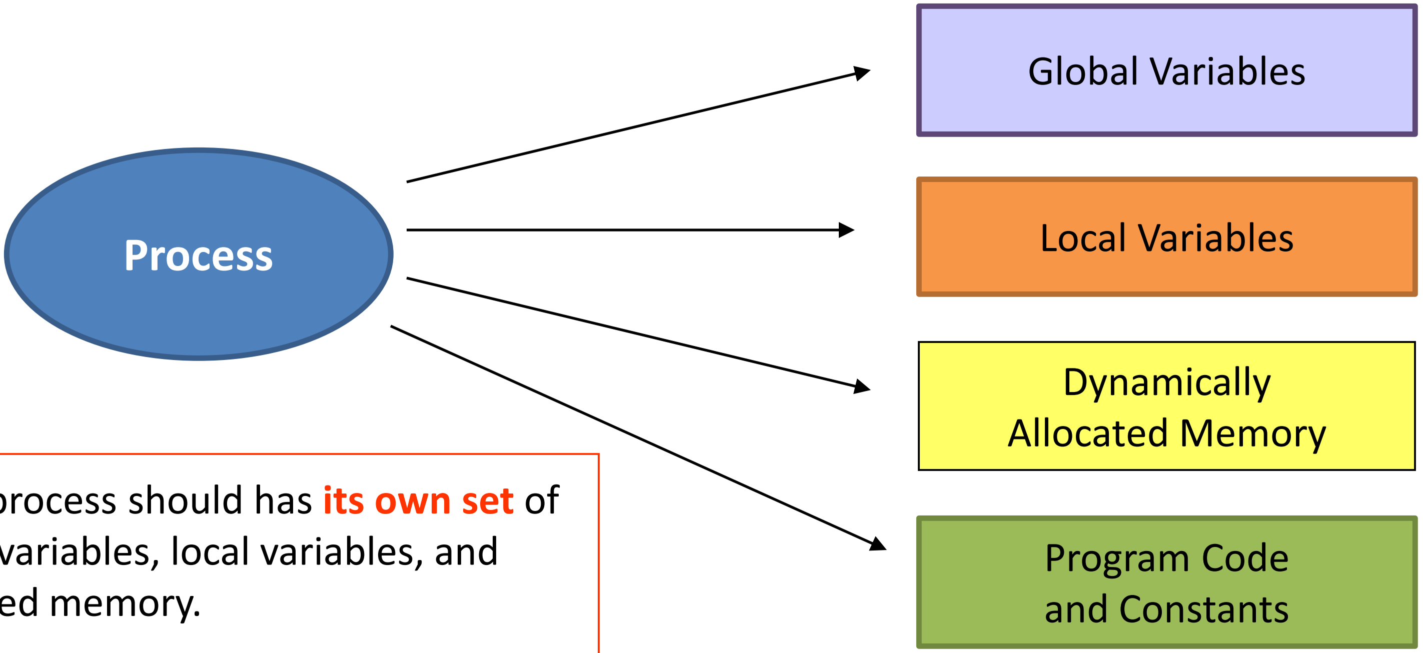
- System calls
  - How to program a simple, bare-bone shell?
- Lifecycle and Scheduling
  - How to create processes?
  - How to handle the death of the processes?
- Signals
  - How to suspend a process?
  - How to change a process behavior when it faces a **Ctrl+C**?
- Synchronization
  - How processes can cooperate to do useful work together?

Introduction to Operating System Components

Memory

Process' Memory

- What are the things that a process has to stored?
  - Do you know that the process memory is arranged **in C-style**?

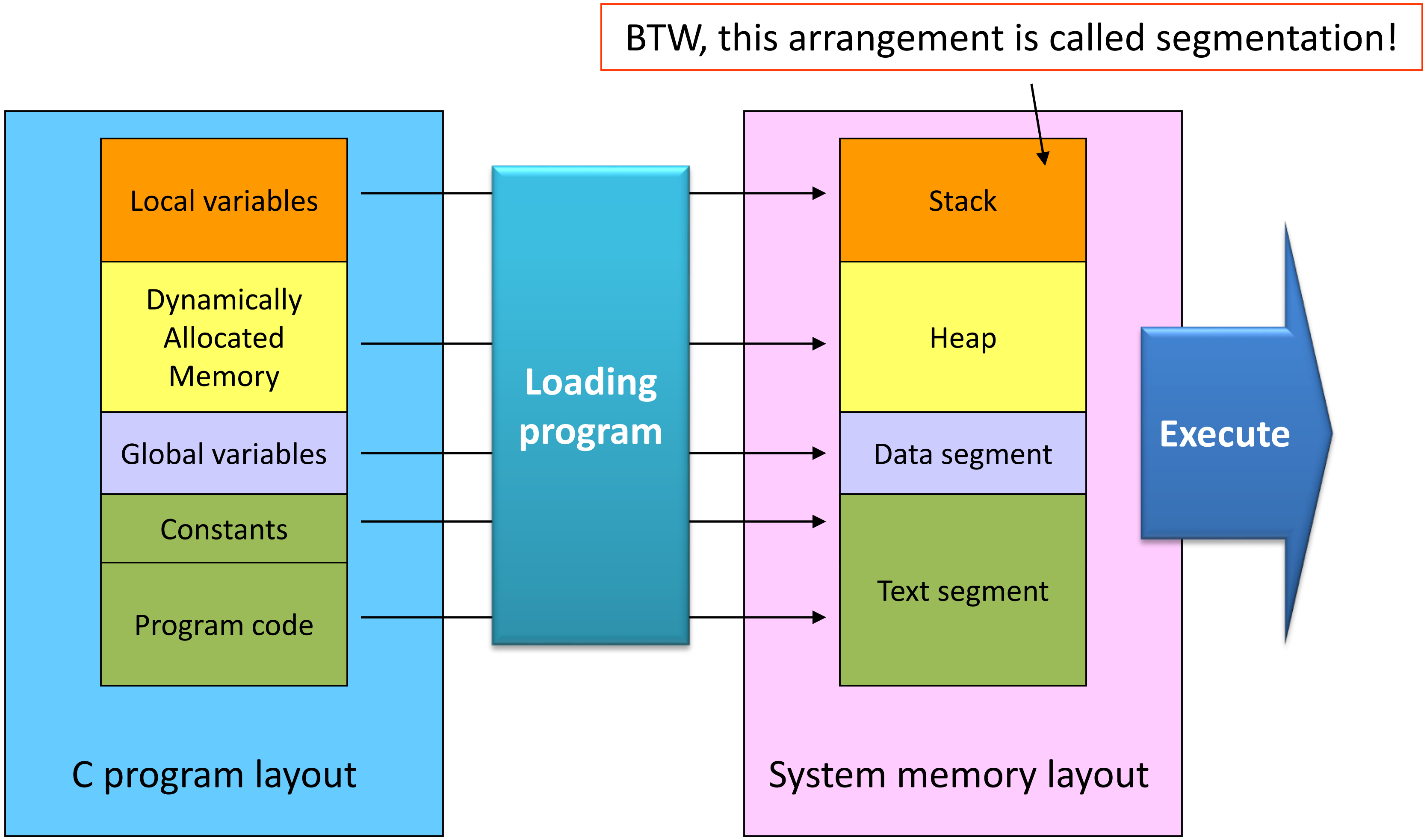


Every process should has **its own set** of global variables, local variables, and allocated memory.

But, the program code is shared. Do you know why?

Process' Memory

- OMG…C is too low-level…

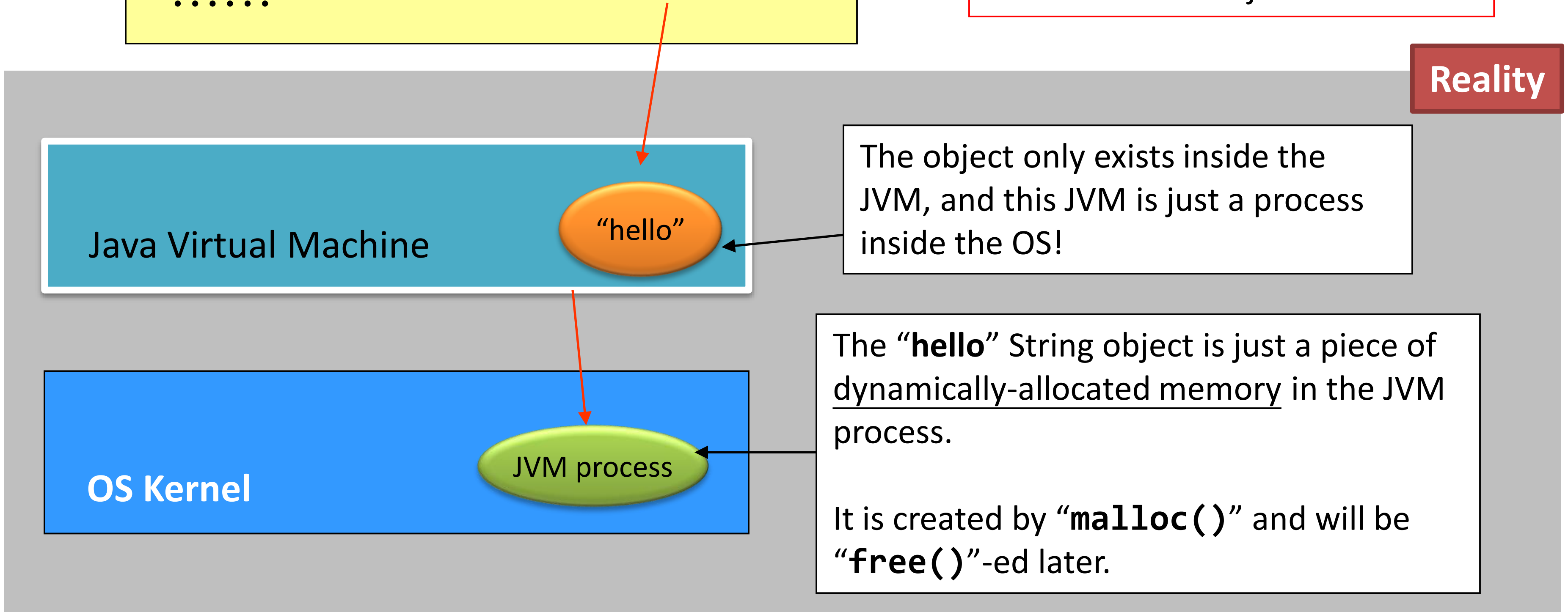


Process' Memory

- *"Hey, you're wrong! Java does not have the above layout…"*, you cried.

```
......
String str = new String("hello");
......
```

This statement creates an object! C doesn't have objects!



The object only exists inside the JVM, and this JVM is just a process inside the OS!

The "**hello**" String object is just a piece of dynamically-allocated memory in the JVM process.

It is created by "**`malloc()`**" and will be "**`free()`**"-ed later.

Sidetrack: Pros and Cons in using C

- Cons:
  - Some people argued that C is a **bad beginner's programming language**. Now, you can understand why…

Because C requires a programmer to take care of the process-level memory management. Every programmer needs to know about the low-level memory layout in order for him/her to understand what <u>segmentation fault</u> means!
Every aspect on memory management can be manipulated using C. Learning **`malloc()`** exposes you to the heap manipulation. This makes a high-level programming language becoming low-level. Plus, **this exposes you to unpredictable dangers!**

* Disclaimer: choosing which programming language is really a personal choice.

Sidetrack: Pros and Cons in using C

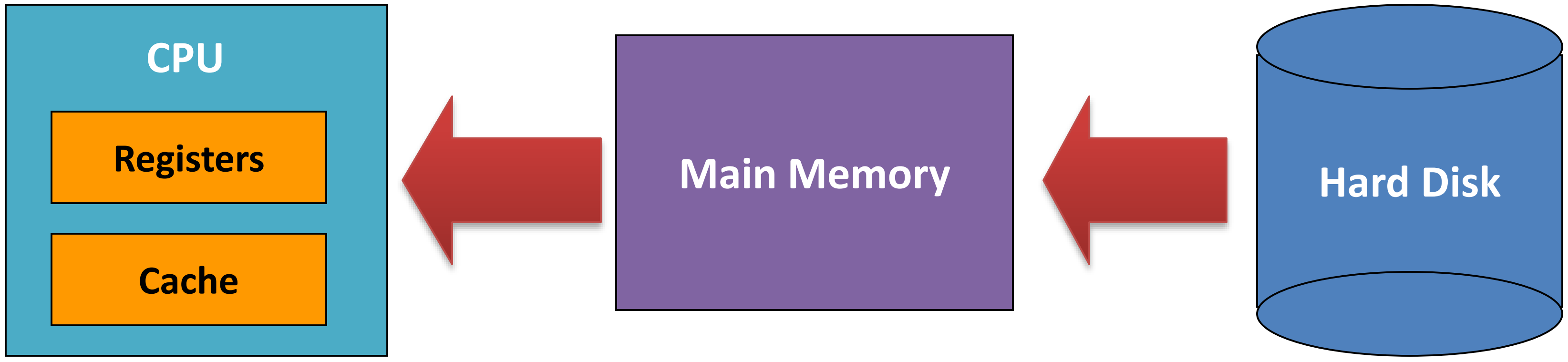
- Pros:
  - Some people argued that C is an **efficient programming language**. Now, you can understand why…

Because C allows a programmer to **manipulate the process-level memory management "directly"**.
That's why many user libraries are implemented using C because of efficiency consideration. E.g., the Java Virtual Machine is implemented using C!
Most importantly, **C is the only language to interact with the OS directly**! In other words, the system call interface is written in C.

* Disclaimer: choosing which programming language is really a personal choice.

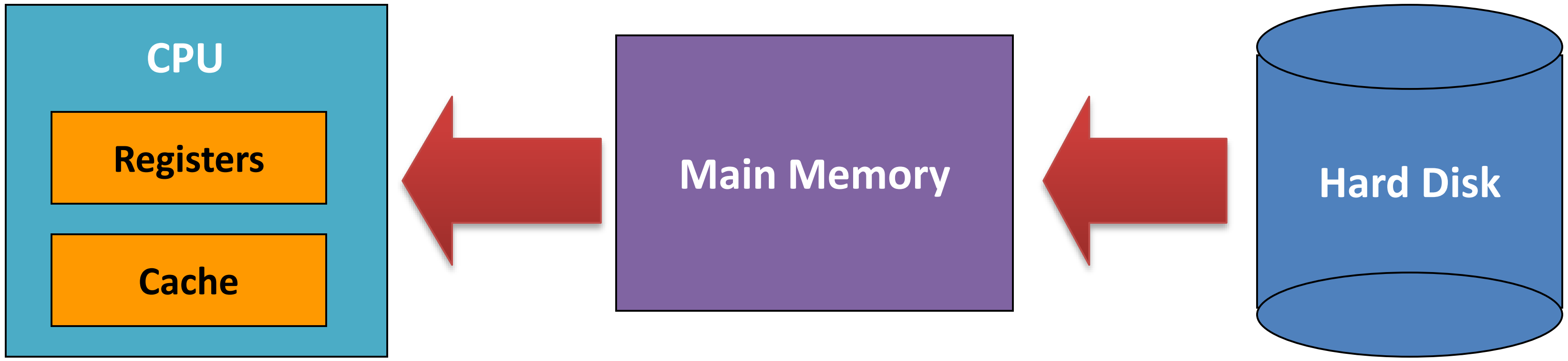
Memory Hierarchy

- In case that someone doesn’t know about the hierarchy below…
  - A program is fetched from hard disk to main memory.
  - When executed, instructions in the program are fetched from the main memory to CPU.



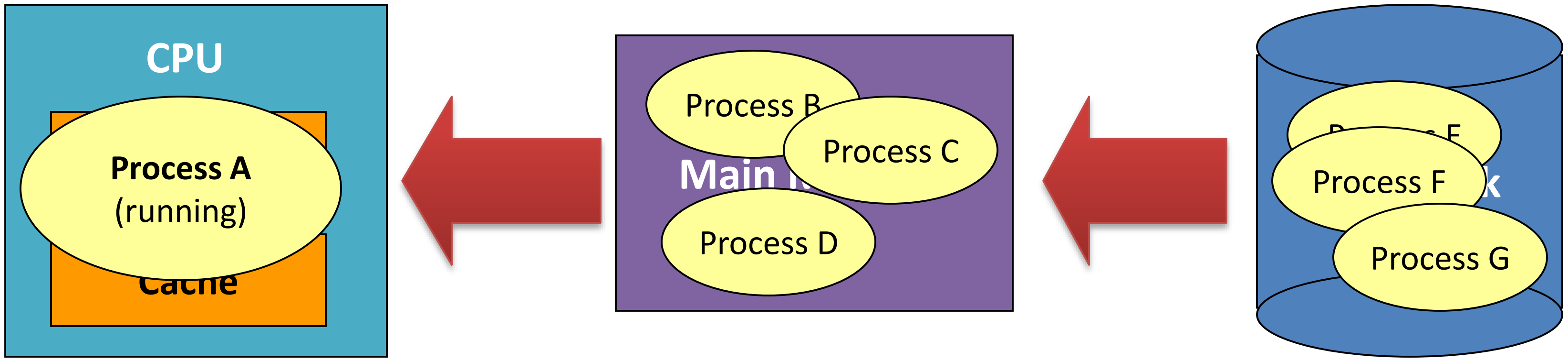
Memory Hierarchy

- However, did you ever need to program those three things when you want to run the program "ls"?
  - Never! Then, who have the jobs done?
  - Of course, **OS**!



Memory Hierarchy

- Typically, there are more than 100 processes running "*at the same time*".
  - There is only a finite number of CPU cores (1-4), depending on how much money you spent.
  - Then, only a finite number of processes can be executed "*really at the same time*".
  - So, other (non-running) processes are stored at different devices controlled by the OS before they get a chance to run.



What will we learn about memory?

- Memory-related functions
  - E.g., you'll learn how to write the "**`malloc()`**" function call using system calls.
- How is the memory of every process aligned in a piece of RAM?
- How can I take as much memory as I wanted?
  - E.g., can you use 16GB of memory for a process after you've bought such an amount of RAM?

Introduction to Operating System Components

<u>File System</u>

What is a File System?

- A file system, FS, means the way that a storage device is used.

- Have you heard of…
  - FAT16, FAT32, NTFS, Ext3, Juliet?
  - They are all file systems.
  - They mean the way that a storage device is utilized.

What is a File System?

- A file system must record the following things:
  - directories;
  - files;
  - allocated space;
  - free space.

- Think about the consequences if any one of the above is missing…

Two faces of a file system

- The **storage design** of the file system.
  - A file spends most of its time on the disk.
  - So, a file system is about how they are stored.
  - Apart from files, many others things are stored in the disk.
- The **operations** of the file system.
  - A file can be manipulated by processes.
  - So, a file system is also about the operations which manipulate the content stored.

FS VS OS

- **A FS is independent of an OS!**
  - If an OS supports a FS, then the OS can do whatever operations over that storage device.
  - Else, the OS doesn't know how to read or update the device's content.

Windows XP supports	Linux supports
NTFS, FAT32, FAT16, ISO9660, Juliet, CIFS	NTFS, FAT32, FAT16, ISO9660, Juliet, CIFS, Ext2, Ext3, etc…

Linux supports far more FS-es than any versions of Windows

File operations?

- Pop quiz!
  - Guess, what are the fundamental file (not dir) operations?

Open	**Read**	**Write**	**Close**	**Rename**	**Delete**

- Well…creating is not...
  - It is just a special case of opening a file.
- Sorry…copying is not…
  - Do you know how it is implemented through the above operations?
- Sorry…moving is the same as renaming…
  - Except that a file is moving from one disk to another.

What we will learn about FS?

- More types of files and operations.
  - Including the library functions and system calls.
  - E.g., directory operations.
- Implementation of some famous FS-es.
  - You'll have <u>an assignment</u> about it.
- Why does a FS fail me?
  - Why does a file system perform badly?
  - Will a file system lose files without bad sectors?
  - Why does a file recovery tool not always work?

Let’s Rock!