Introduction to Operating Systems

Lecturer: Dr. T.Y. Wong

Chapter 1

Overview of an Operating System

Outline

- What is an Operating System?

- Components
  - Process management;
  - Memory management;
  - File system;

What is an operating system?

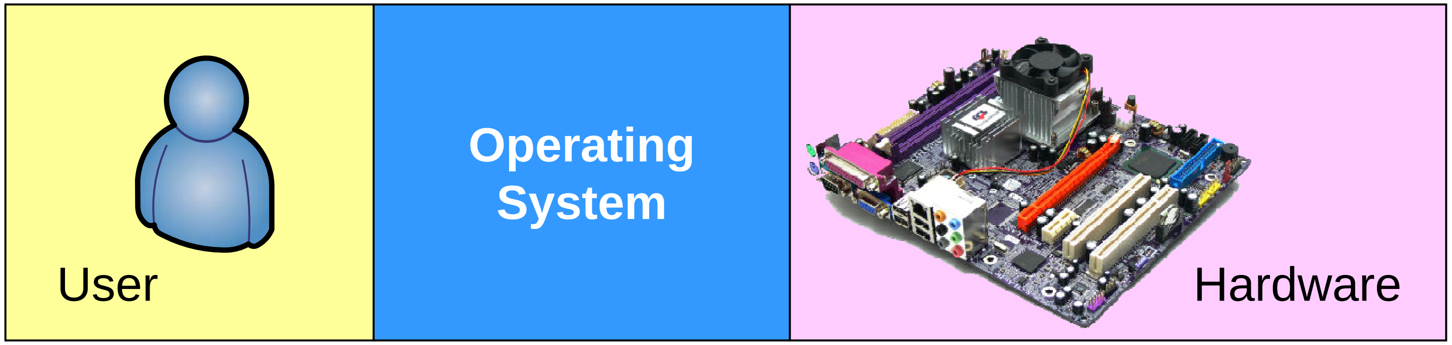
- A system that is operating?
- A system that operates something?

What is an operating system?

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

What is an operating system?

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

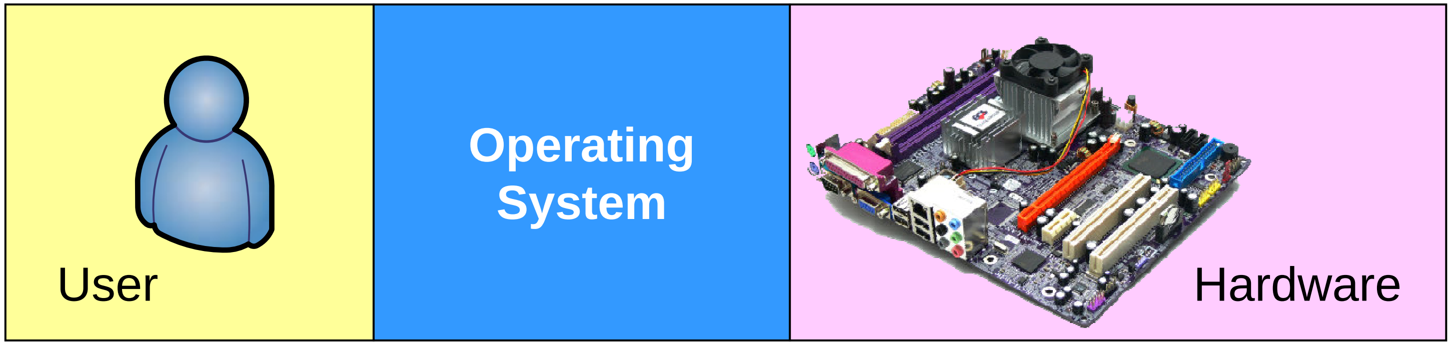


What is an operating system?

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

  - The OS provide an abstraction of the hardware.

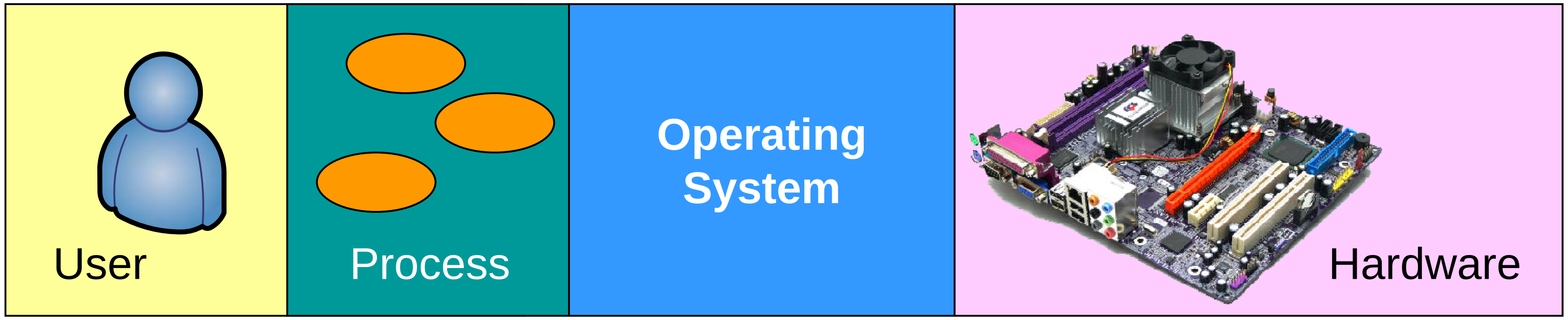
Example. The OS, on one hand, hides the physical system memory away from you. On the other hand, it tells you that there is system memory available when you run your applications.



What is an operating system?

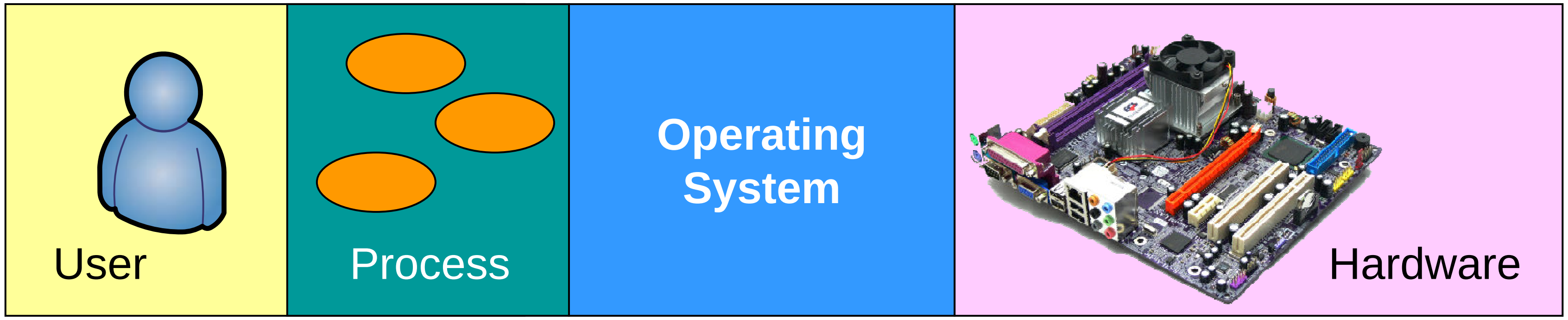
- Users are not working with the OS directly.
  - They always work with processes.
- What is a process?
  - It is a running image of a program.

Example. You have only one copy of “firefox.exe”. However, you can open different browser windows using the same program file.



What is an operating system?

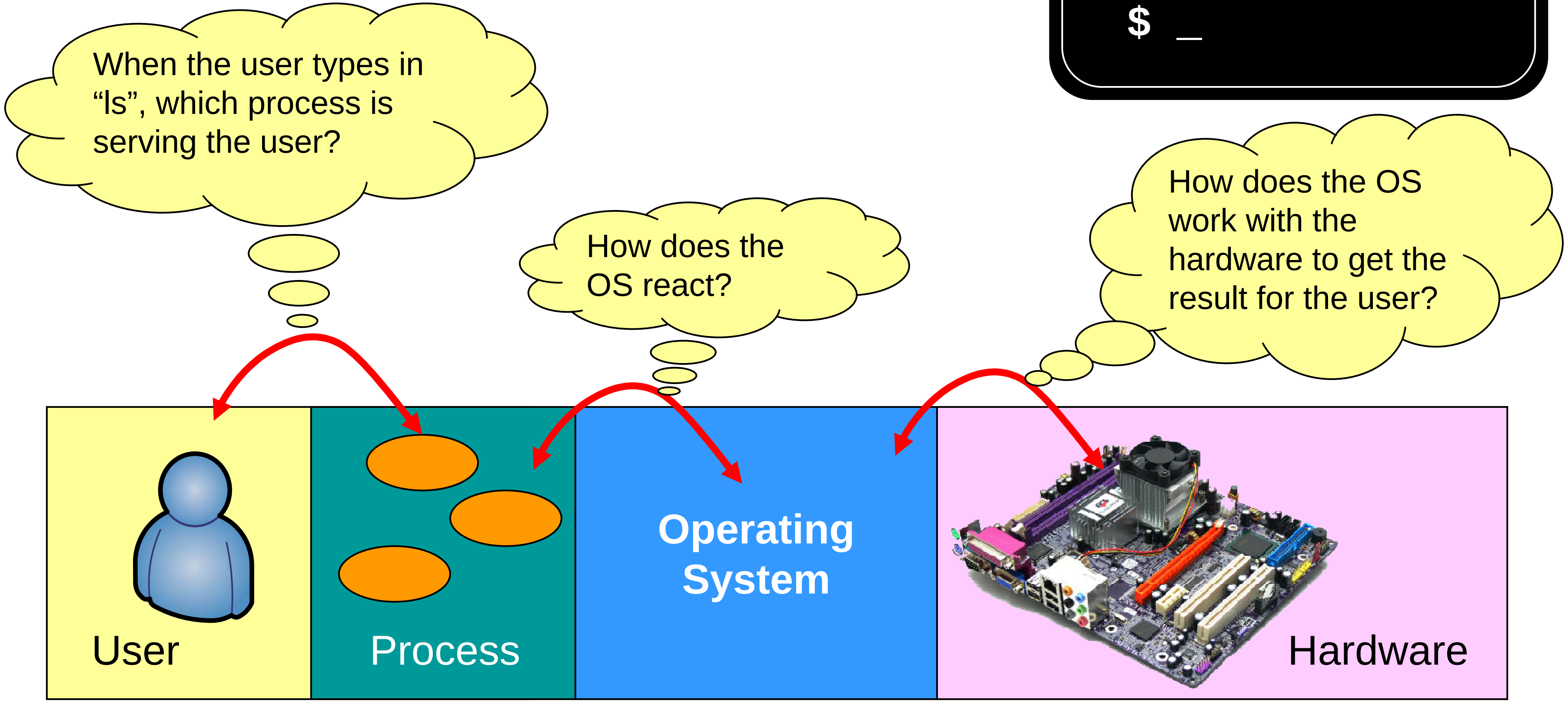
- Processes are important!
  - Whatever programs you run, it behaves as processes.
    - i.e., you need processes to open files, utilize system memory, listen to music, etc.
  - So, process lifecycle, process management, and other process related issues are essential topics of this course.



What is an operating system?

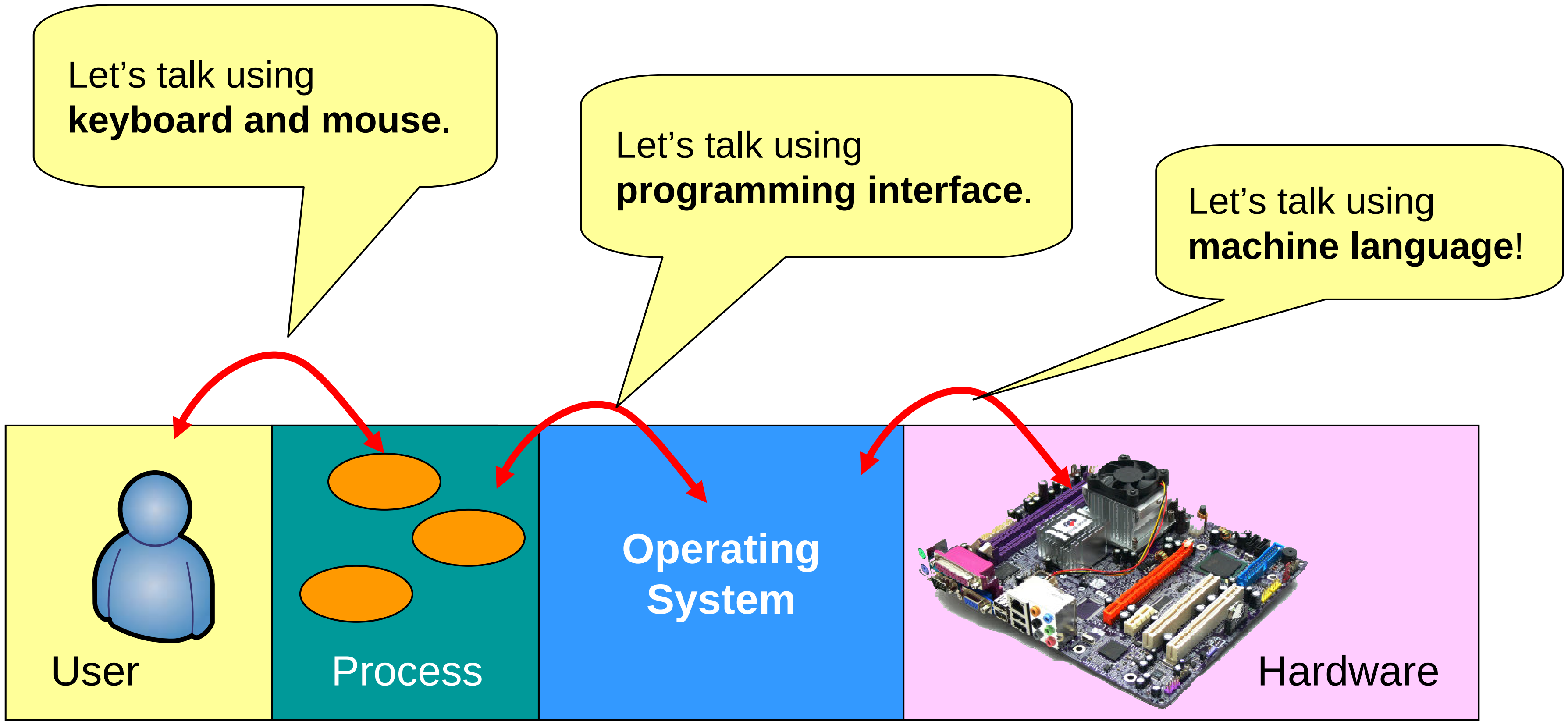
- Interactions?

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


When the user types in "ls", which process is serving the user?
How does the OS react?
How does the OS work with the hardware to get the result for the user?
User
Process
Operating System
Hardware

How to interact with the OS?

- The interactions.

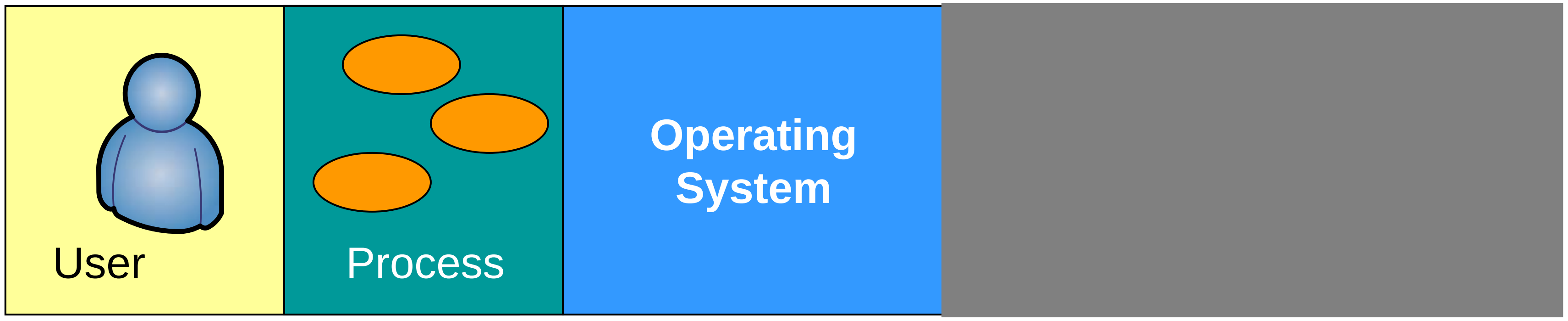

Let's talk using **keyboard and mouse.**
Let's talk using **programming interface.**
Let's talk using **machine language**!
User
Process
Operating System
Hardware

How to interact with the OS?

- For an ordinary users:
  - OS is a “thing” that is able for them to store programs and run programs.
  - They don’t even need to know what a process is.
    - As long as they don’t want to kill the word processor when it is frozen.

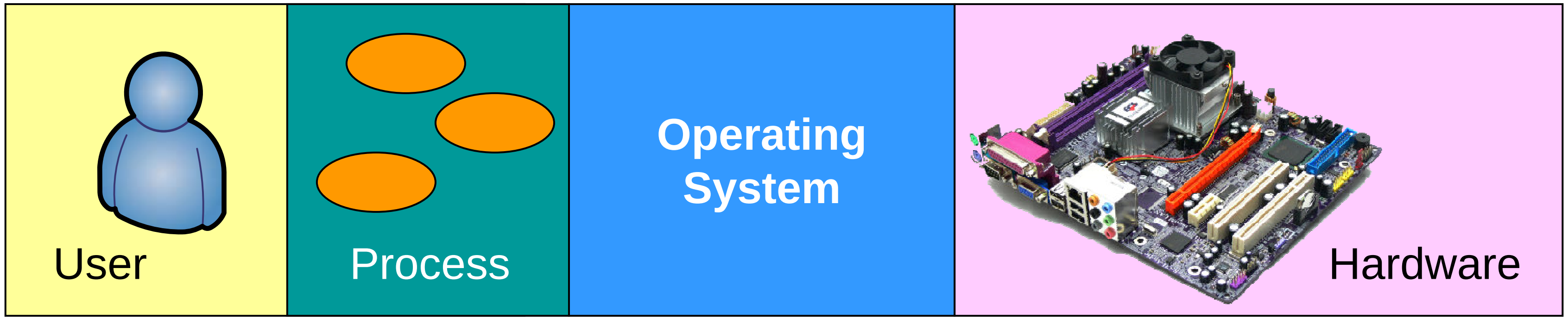
How to interact with the OS?

- For programmers:
  - OS is a piece of **software** that allows them to:
    - to interact with the user, thanks the OS for the I/O interface!
    - to interact with the OS itself, thanks the OS for the **system calls**!



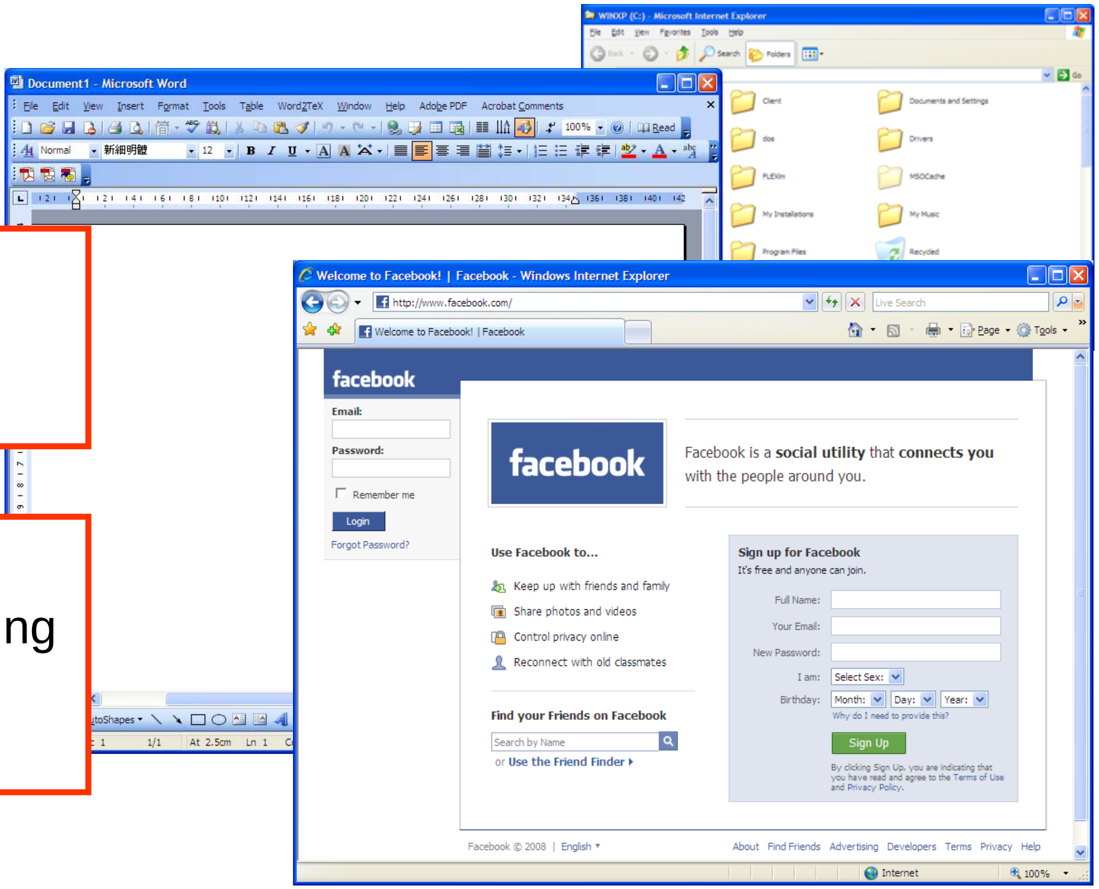
How to interact with the OS?

- For hardware driver programmers:
  - OS is a piece of **software** that allows them to:
    - to host the driver programs.



How to interact with the OS?

- Therefore…

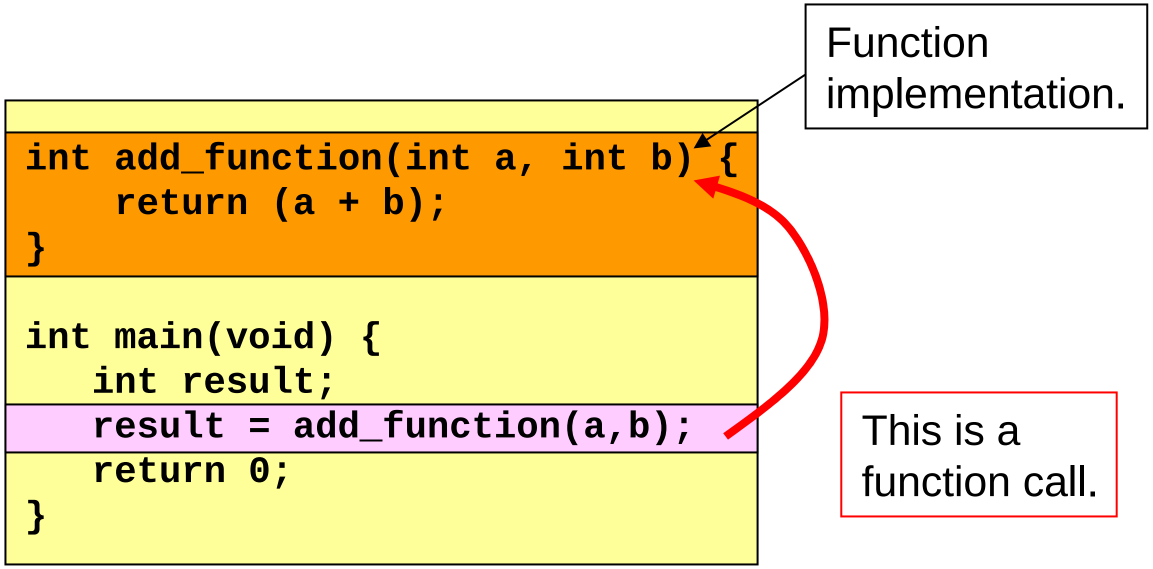

Document1 - Microsoft Word
WINXP (C:) - Microsoft Internet Explorer
Welcome to Facebook! | Facebook - Windows Internet Explorer
http://www.facebook.com/
facebook
Email:
Password:
Remember me
Login
Forgot Password?
Facebook is a social utility that connects you with the people around you.
Use Facebook to...
Keep up with friends and family
Share photos and videos
Control privacy online
Reconnect with old classmates
Find your Friends on Facebook
Sign up for Facebook
It's free and anyone can join.
Sign Up

These are just application programs that run on top of the operating system!

Inside these application programs, they are interacting with the operating system using system calls.

How to interact with the OS?

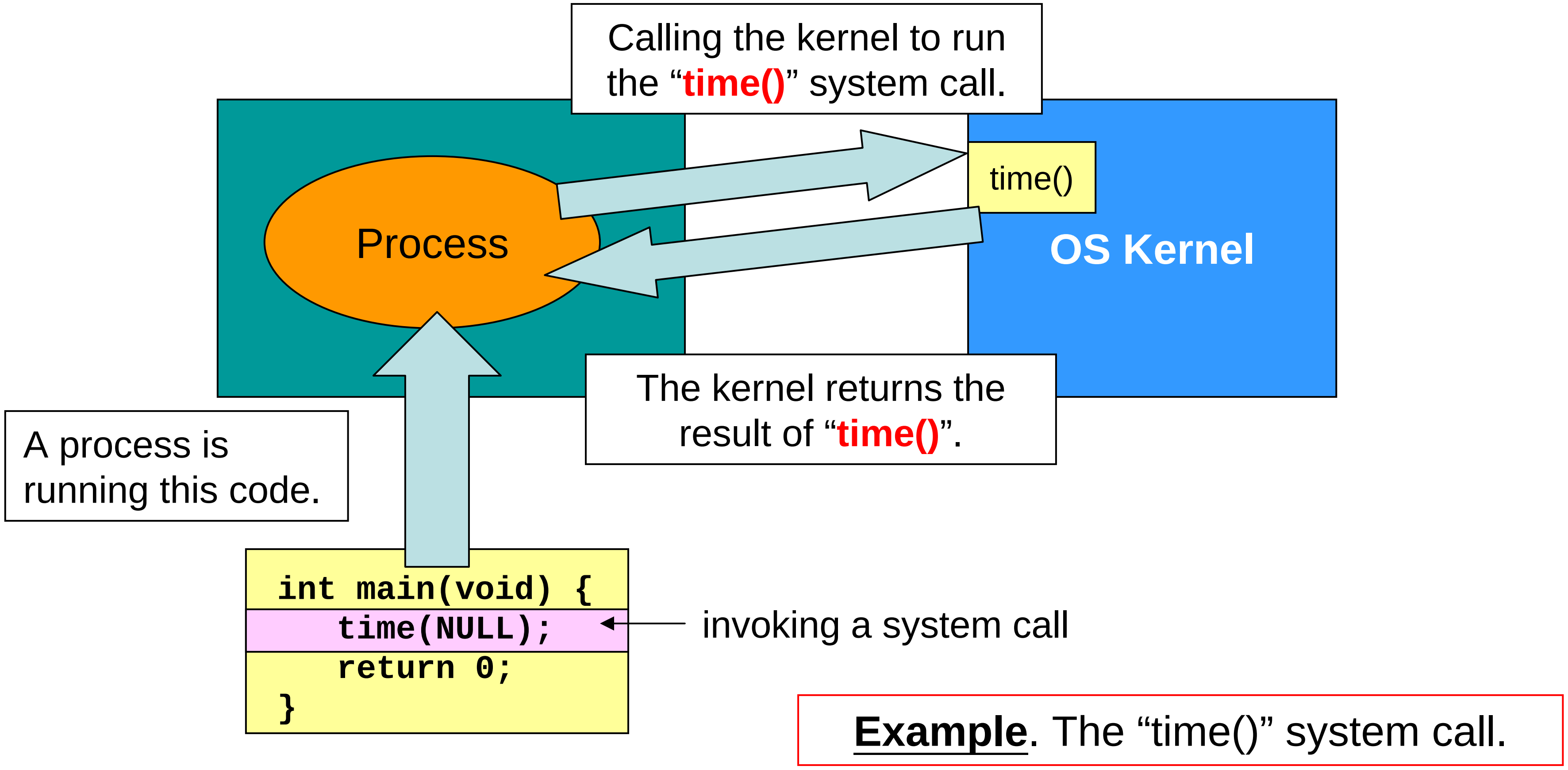
- What is a system call?
  - Informally, a system call is similar to a function call, but…
  - The function implementation is inside the OS, or we name it the **OS kernel**.



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

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

How to interact with the OS?

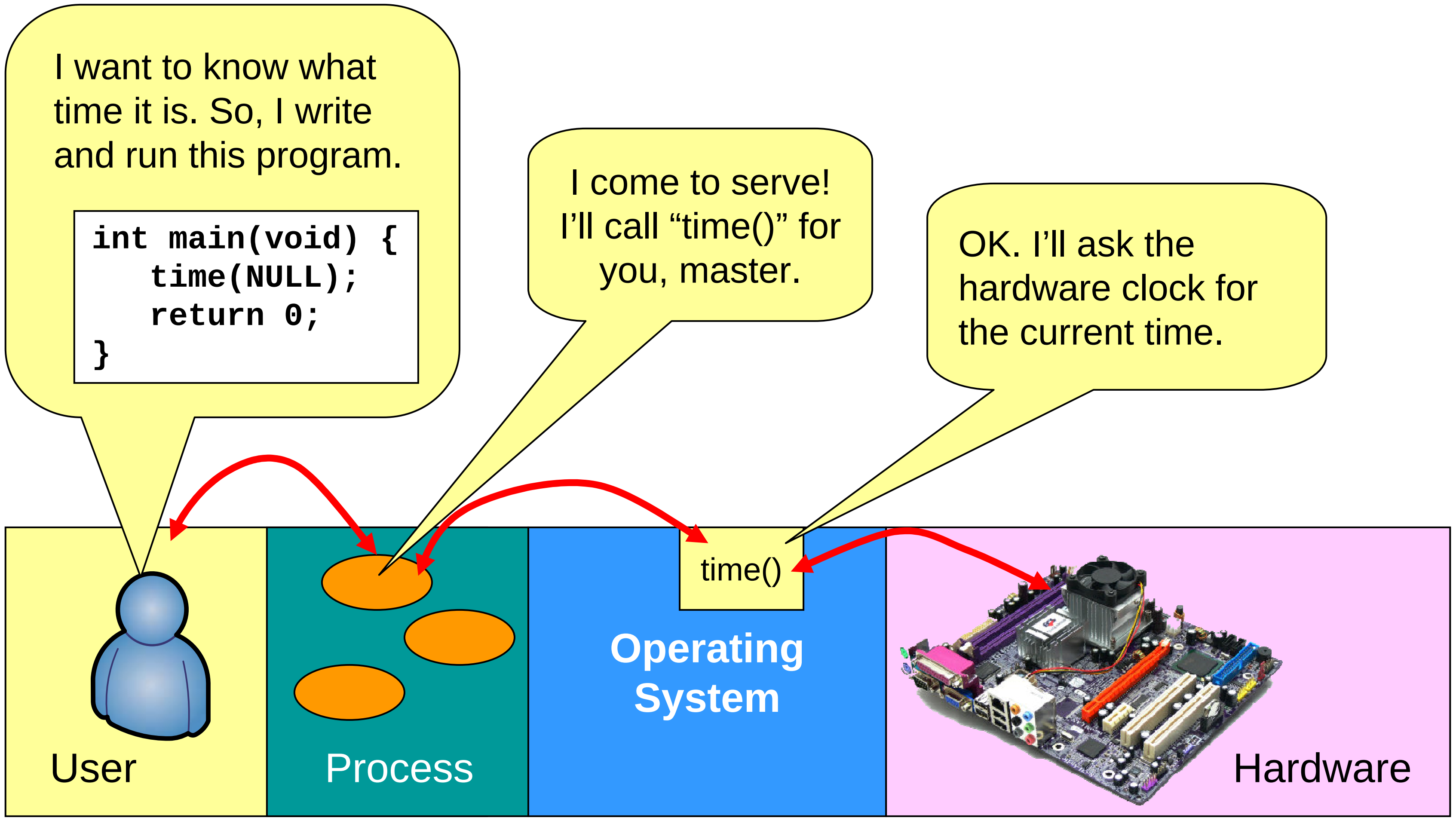
- What is a system call?


Calling the kernel to run the "time()" system call.
time()
Process
OS Kernel
The kernel returns the result of "time()".
A process is running this code.
invoking a system call

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

Example. The "time()" system call.

System Calls


I want to know what time it is. So, I write and run this program.
I come to serve! I'll call "time()" for you, master.
OK. I'll ask the hardware clock for the current time.
time()
User
Process
Operating System
Hardware

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

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 tells you what time it is.

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

Process	File system	Memory
Networking	Security	Device

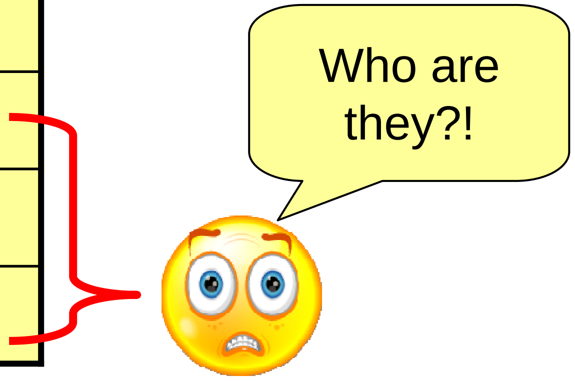
In this course, we focus on these three areas.

System Calls

- When do we know if a "function" is a system call?
  - Read the man page "syscalls" under Linux.

- Let's guess: who are system calls?

Name	System Call?
exit()	Yes
malloc()	No
fopen()	No
fclose()	No



System Call vs Library Function Call

- If they are not system calls, then they are 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!

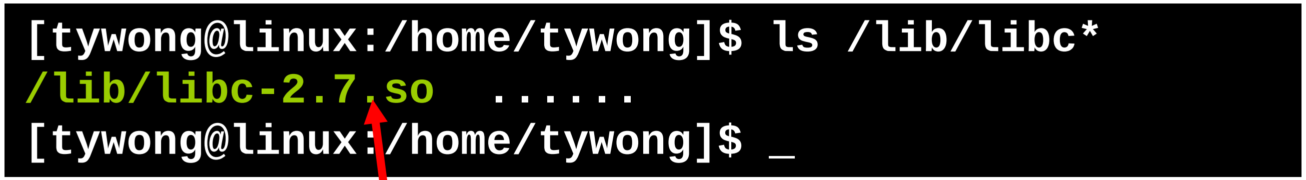
The following two calls have the same effect, but…

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

as a programmer, which one do you prefer?

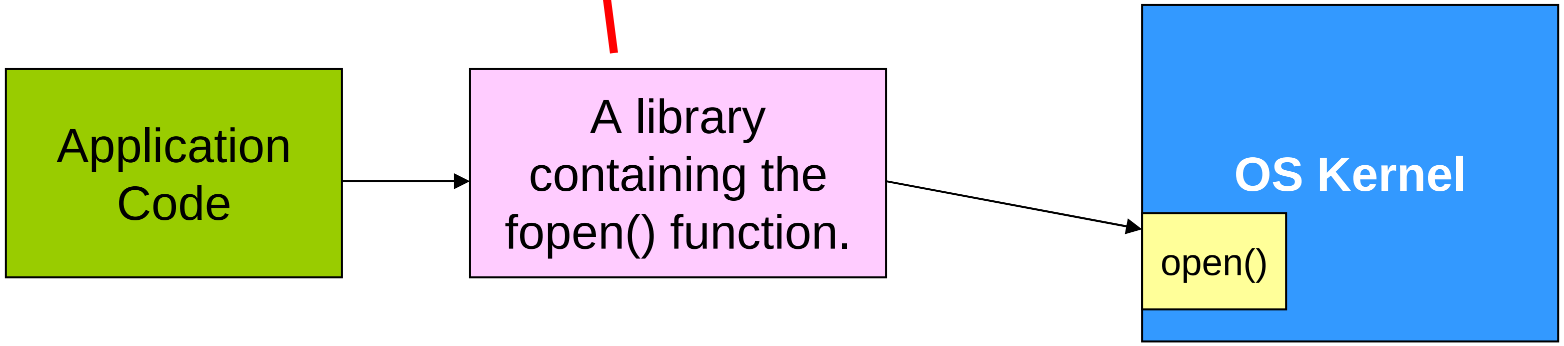
System Call vs Library Function Call

- Those functions are usually packed inside an object called the **library**.



```
[tywong@linux:/home/tywong]$ ls /lib/libc*
/lib/libc-2.7.so  ......
[tywong@linux:/home/tywong]$ _
```

The C standard library file.
Delete it at your own risk!



So, what I ask 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!**
  - Throughout this course, you will learn the details of the OS design:
    - what are its capabilities, and
    - what are its limits.
  - So, you will have a better control over the OS.

Operating System Components

Process	Memory	File system

Introduction to Process

- A process is not just a running program!

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'

Assume that Commands A and B creates Process A and Process B, respectively.

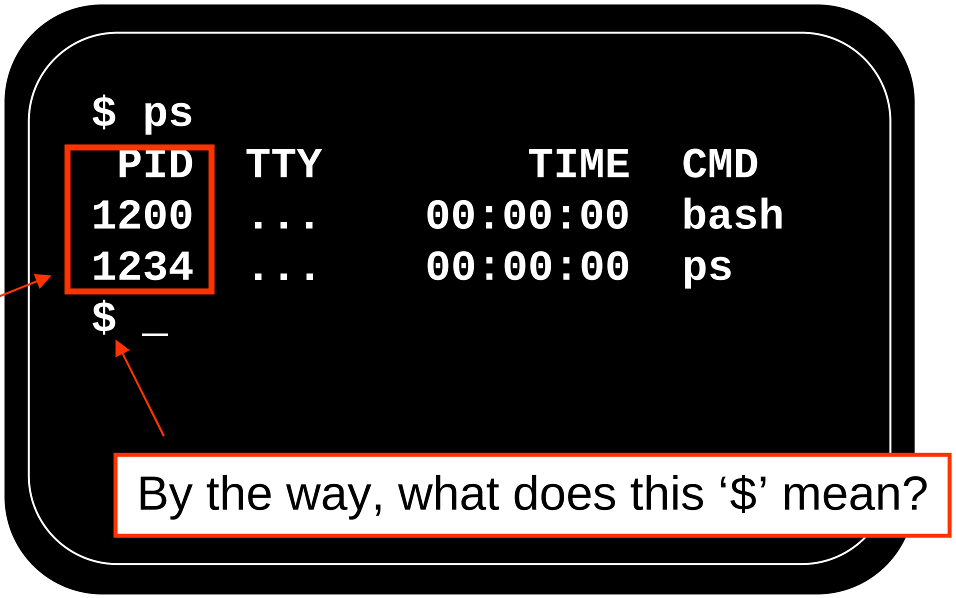
Similarity	Difference
Both use the program file "/bin/ls".	The program arguments are different.
---	The processes' internal status are different, such as running time.

Introduction to 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 <u>not bounded to execute just one program</u>!
- A process is an active entity.
  - A process has its local states concerning the execution. E.g.,
    - which line of codes it is running;
    - which CPU (if there are many) it is running on.
  - The local states change while running.
- 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”.



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

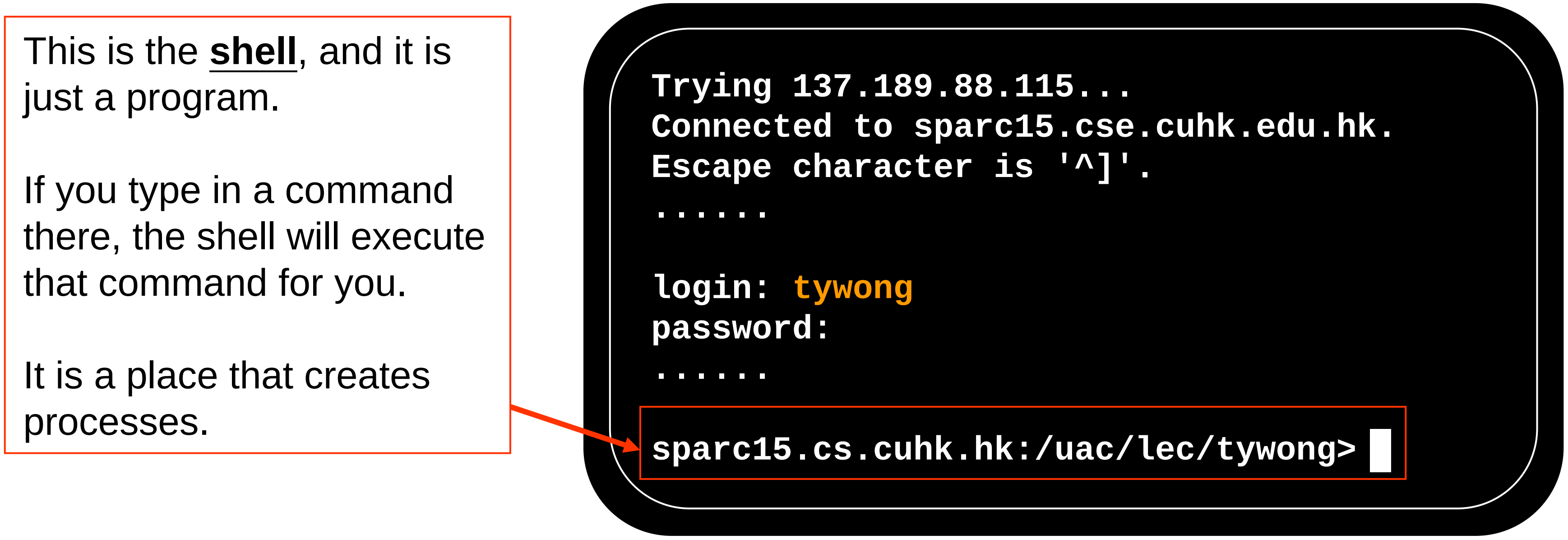
This is the unique identification number of a process, called **Process ID**.

By the way, what does this ‘$’ mean?

Process-Related Tools

- The '$' sign in the previous example represents a **shell**.

  (I'm lazy, so I just place a '$' sign there.)



This is the **shell**, and it is just a program.

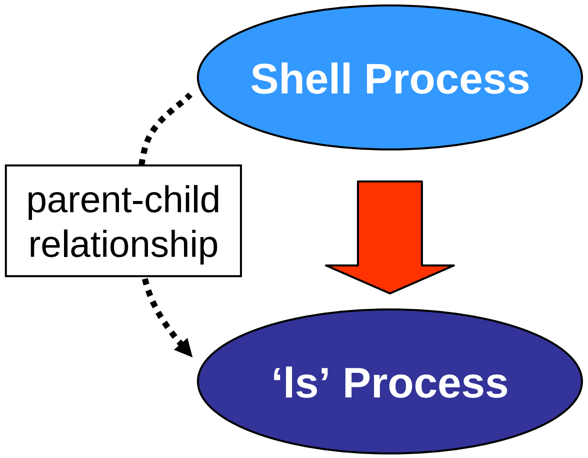
If you type in a command there, the shell will execute that command for you.

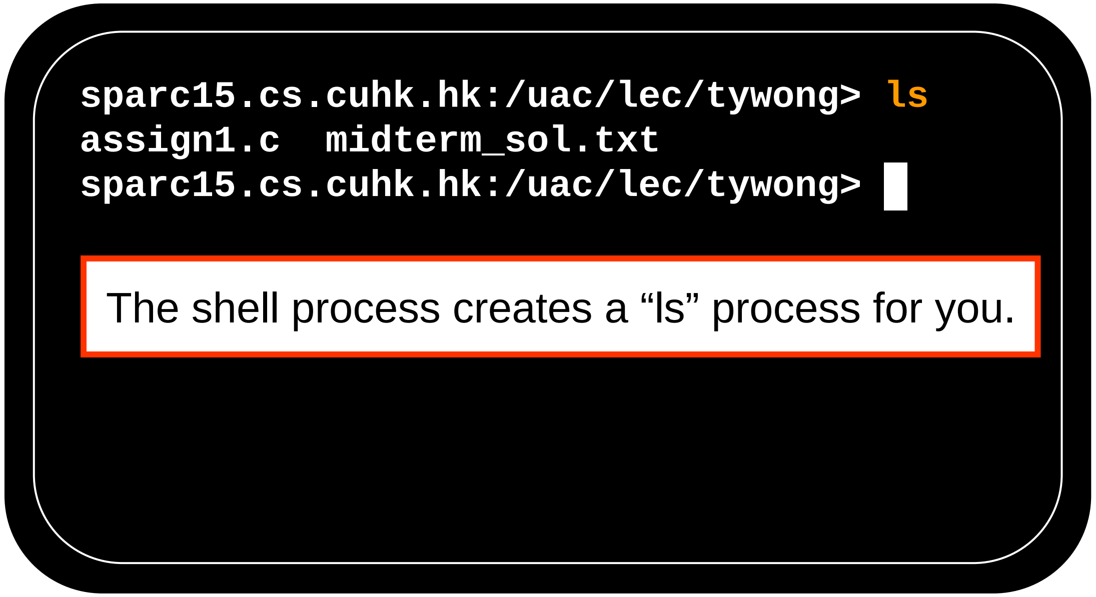
It is a place that creates processes.

```
Trying 137.189.88.115...
Connected to sparc15.cse.cuhk.edu.hk.
Escape character is '^]'.
......

login: tywong
password:
......

sparc15.cs.cuhk.hk:/uac/lec/tywong> █
```

Process-Related Tools

- So, what is going on inside that shell?
  - The shell is called a **parent process**.
  - The shell creates the “ls” process, and is called a **child process**.
  - After its creation, the child process executes the command “ls”.



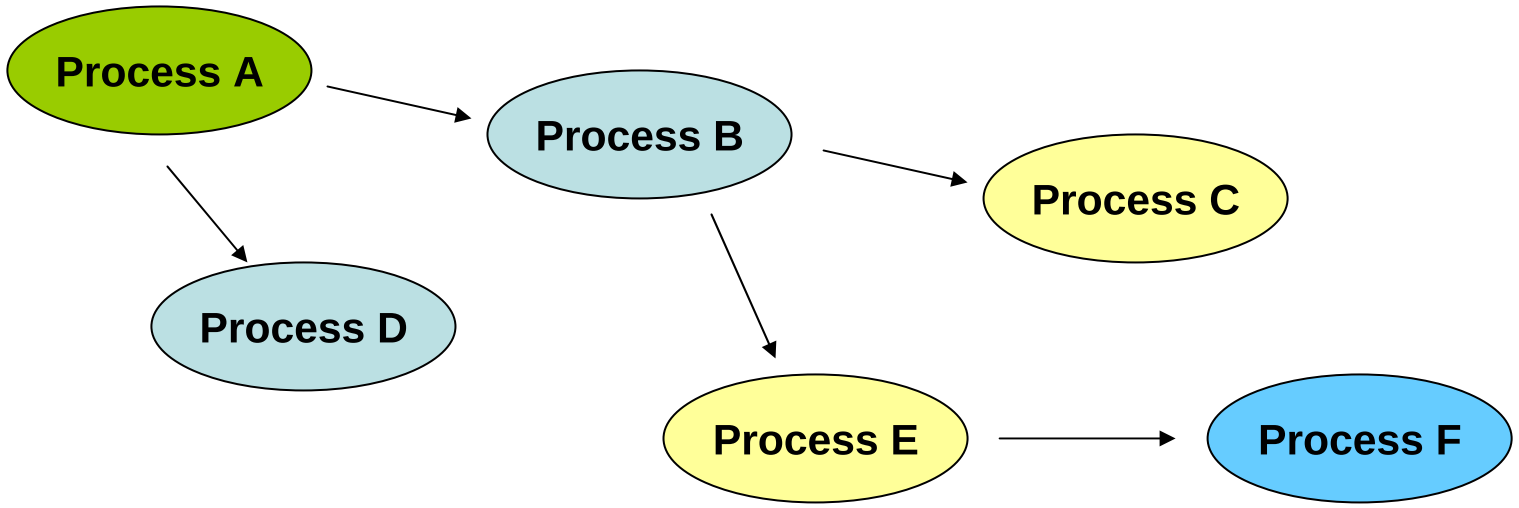


```
sparc15.cs.cuhk.hk:/uac/lec/tywong> ls
assign1.c  midterm_sol.txt
sparc15.cs.cuhk.hk:/uac/lec/tywong> █
```

The shell process creates a “ls” process for you.

Process-Related Tools

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

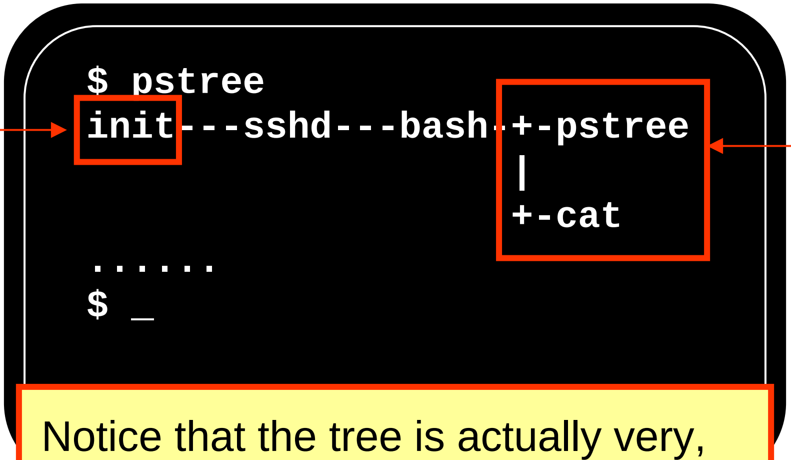


E.g., "Process E" is the shell and "Process F" is "/bin/ls".

Process-Related Tools

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

- Meet the program "**`pstree`**".



```
$ pstree
init---sshd---bash-+-pstree
                   |
                   +-cat

......
$ _
```

"init" is the mother of all processes.

The "**`bash`**" is the shell process.

It has its parent, called the "**`sshd`**" – the SSH server program.

It has two children:
- "**`pstree`**" and "**`cat`**".

Note: this is just an example. "**`pstree`**" does not always show this output.

Notice that the tree is actually very, very long, here only shows the concept of the tree hierarchy.

What will we learn about process?

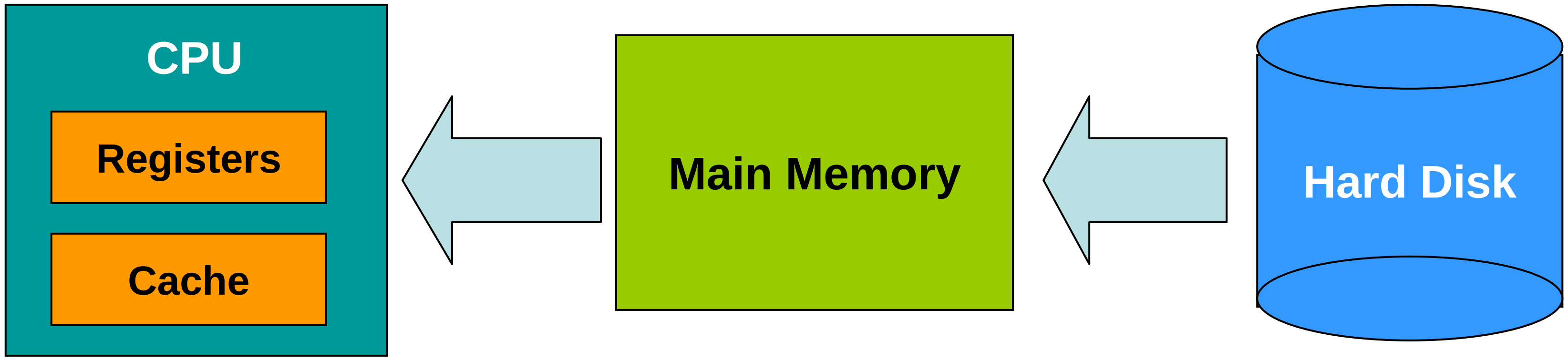
- Process-related system calls
  - How to program a simple, bare-bone shell?
- Process Lifecycle
  - How to create processes?
  - How to handle the death of the processes?
- Process Synchronization
  - How processes can cooperate to do useful work together?

Operating System Components

Process	**Memory**	File system

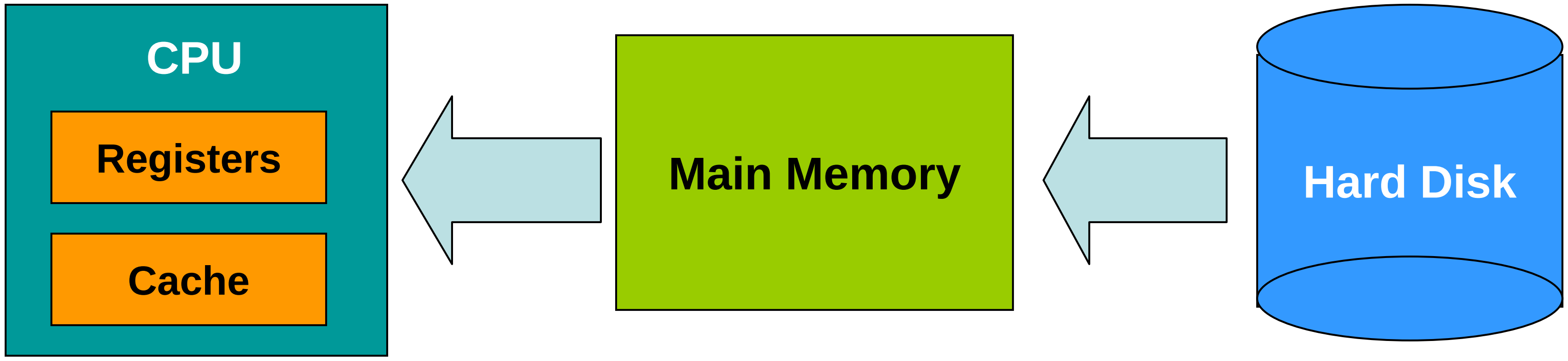
Memory Hierarchy

- 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.



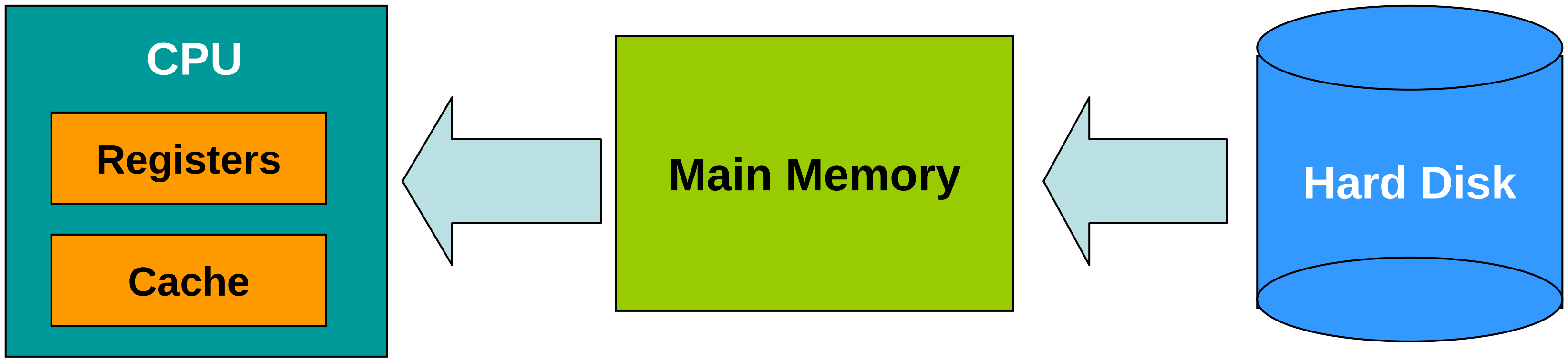
Memory Hierarchy

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



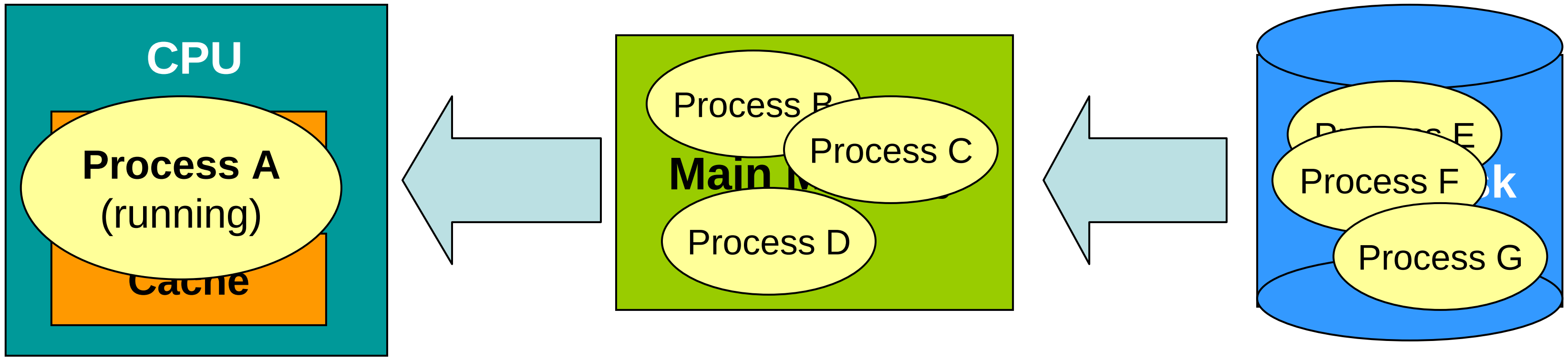
Memory Hierarchy

- The following hierarchy does not just fit the program execution flow, but also for storing processes!
  - You already know that a process is more than a running program.
  - So, the states of the process needs storage.



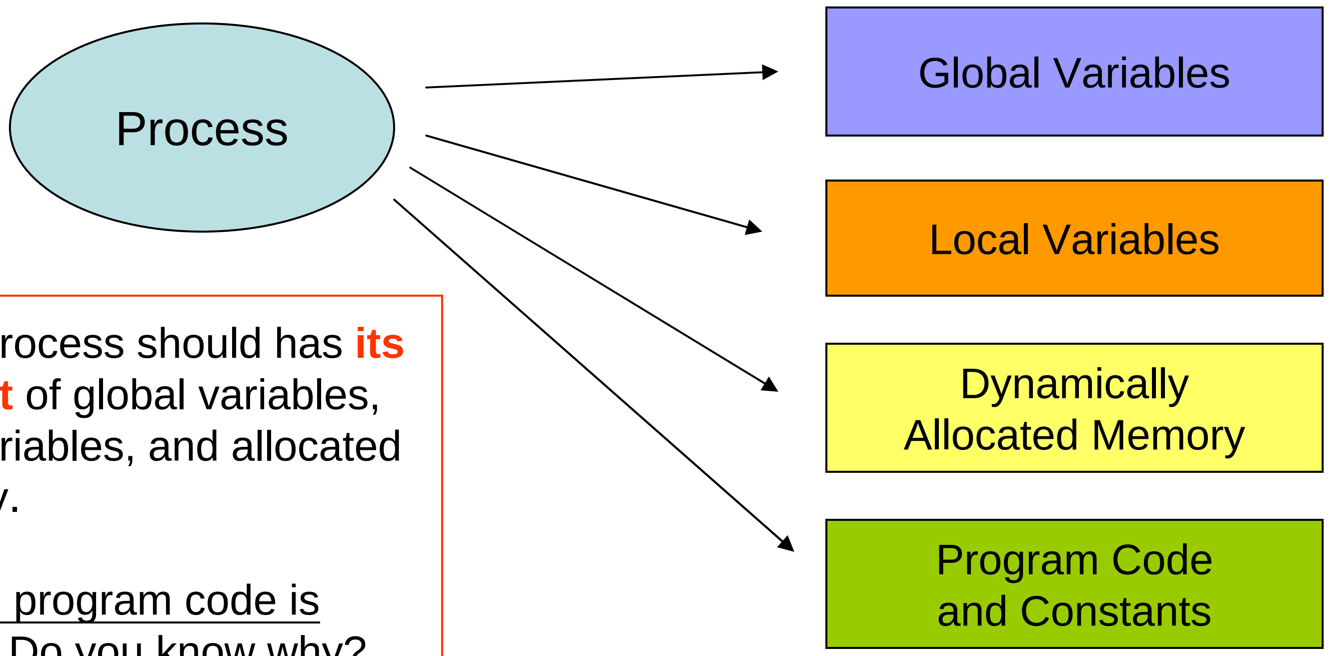
Memory Hierarchy

- Typically, there are more than 50 processes running "*at the same time*".
  - There is only a finite number of CPUs (1-2), depending on how much money you spent.
  - Then, only a finite number of processes can be executed "*really at the same time*".
  - So, they are stored at different devices controlled by the OS before they get a chance to run.



Process’ Memory

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

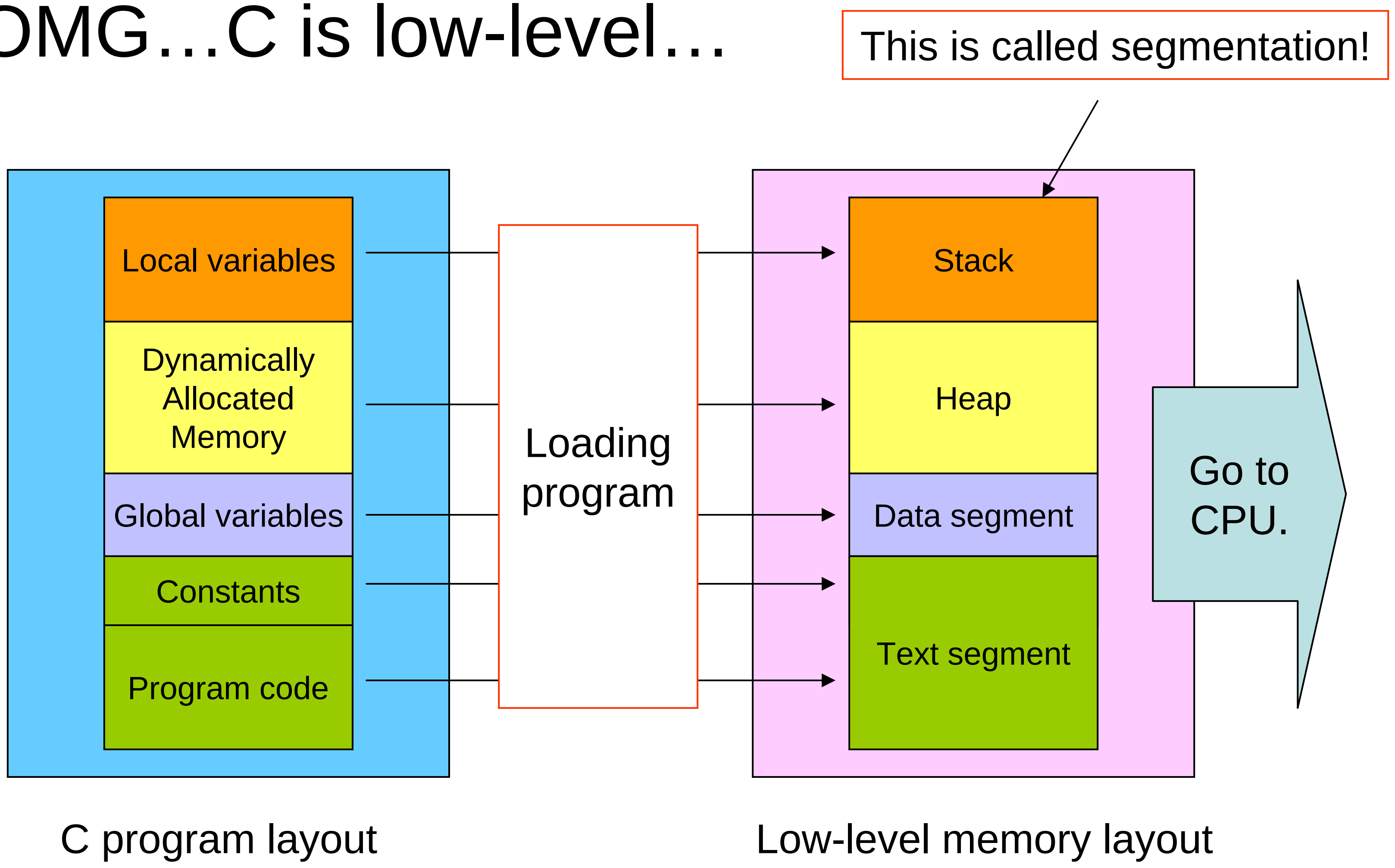


Every process should has **its own set** of global variables, local variables, and allocated memory.

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

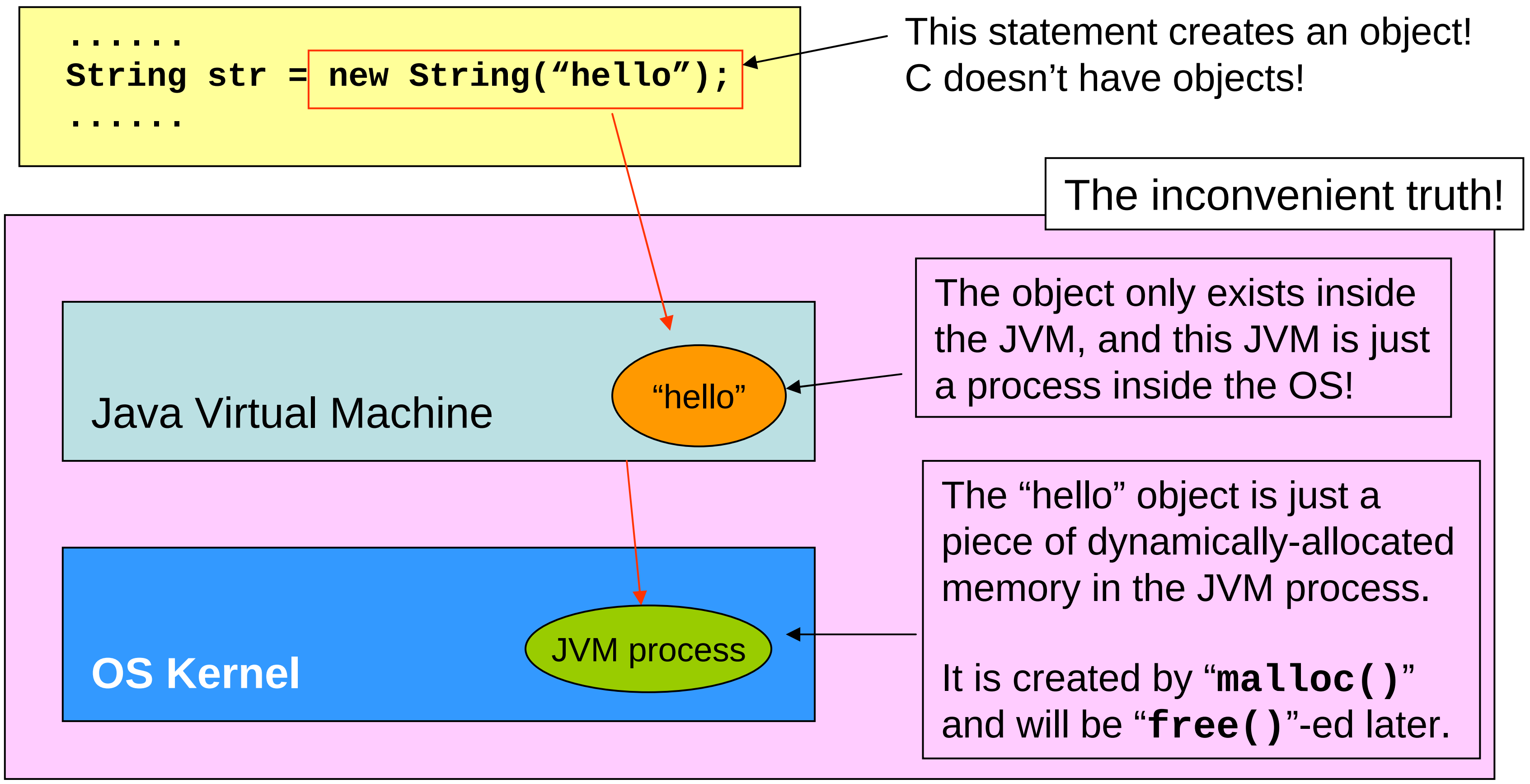
Process' Memory

- OMG…C is low-level…



Process' Memory

- But, Java does not have the above layout…



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

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

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

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

Pros and Cons in using C

- 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.
    - A program needs to know about the low-level memory layout in order for him/her to understand what is *segmentation fault* !
  - Every aspect on memory management can be manipulated using C.
    - Learning malloc() exposes you to the heap manipulation.
    - This makes a high-level prog. lang. becoming low-level.

Pros and Cons in using C

- 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*".
    - Also, it allows C programs to have embedded assembly-language statements.
  - That's why many user libraries are implemented using C because of efficiency consideration.
    - E.g., the Java Virtual Machine is implemented using C!

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

What will we learn about memory?

- Memory-related functions
  - E.g., you'll have an assignment on how to write the "**`malloc()`**" function call using system calls.

- How the memory of every process is aligned in a piece of RAM?

- How can I run a program of size 700MB with only 512MB RAM?

  (well…this may be an out-of-date question)

Operating System Components

Process	Memory	**File system**

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, etc.
  - They are all file systems.
  - They mean the way that a storage device is utilized.

What is a File System?

- A file system must contain the following components:
  - directories;
  - files;
  - allocated space;
  - free space.

Who don't know about these two?

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

Two Faces of a File System

- The **stable side** 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 **dynamic side** of the file system.
  - A file is always being manipulated.
  - So, a file system is also about the operations which update the content of the file system.

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!
  - According to your knowledge, what are the basic file operations?

open	read	write	close	rename	delete

  - Sorry…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 type of files and operations.
  - Including the library functions and system calls.
  - E.g., directory operations.

- Implementation of some famous FS-es.
  - You’ll have an assignment about it.

- Why does a FS fail me?
  - Why does a file system perform bad?
  - Why does a file system lose files without bad sectors?
  - Why does a file recovery tool not always work?

Extra Topics

- If we’ve time during the last 2 weeks, we may cover:
  - System security:
    - The root account is the supreme account in Linux/Unix system.
    - How can I become the root?
  - Device driver:
    - A driver contains all the operations specified by a particular device.
    - How is a driver related to the kernel.