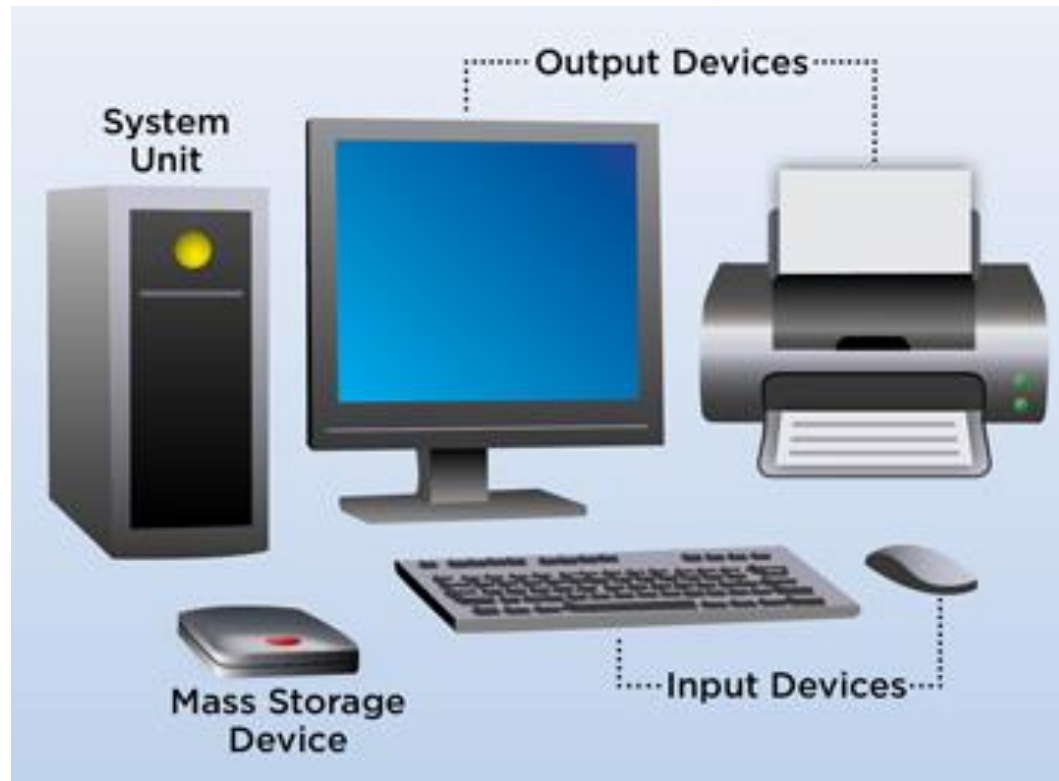


Lecture 1

OS Introduction

Bo Tang @ 2020, Spring

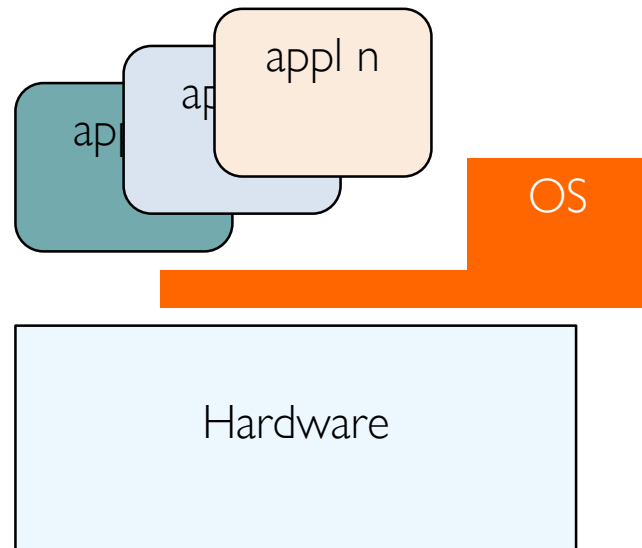
What is a Computer



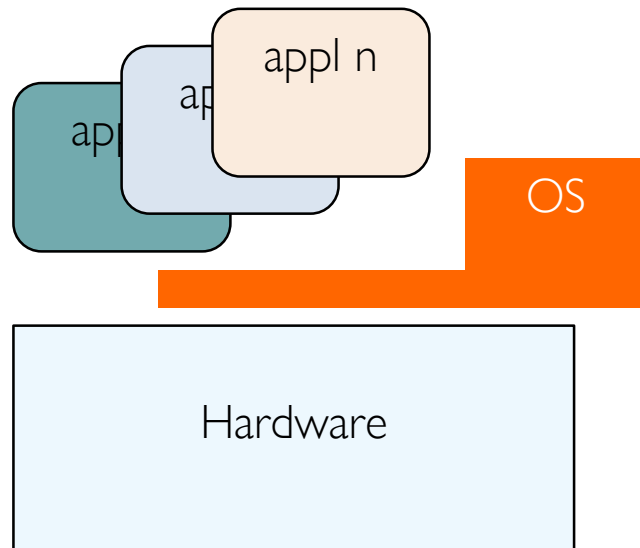
Print “Hello CS302” in a Computer?

How do we do?

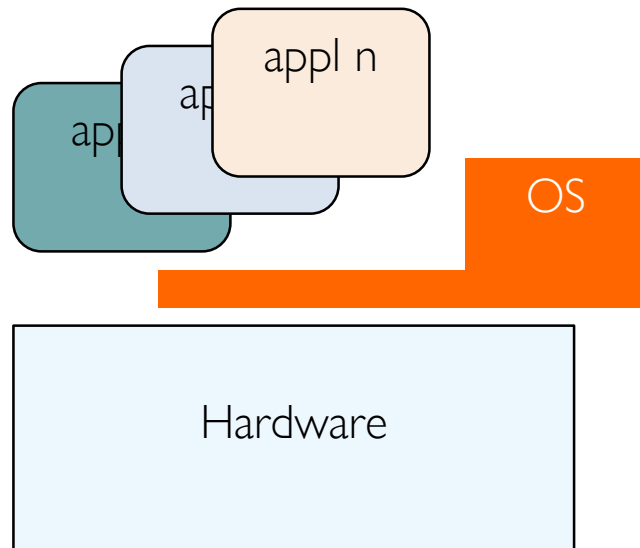
Different Levels of OS course: use it



Different Levels of OS course: play it



Different Levels of OS course: design it



Stanford / CMU OS Course

Learning OS concepts by Coding them

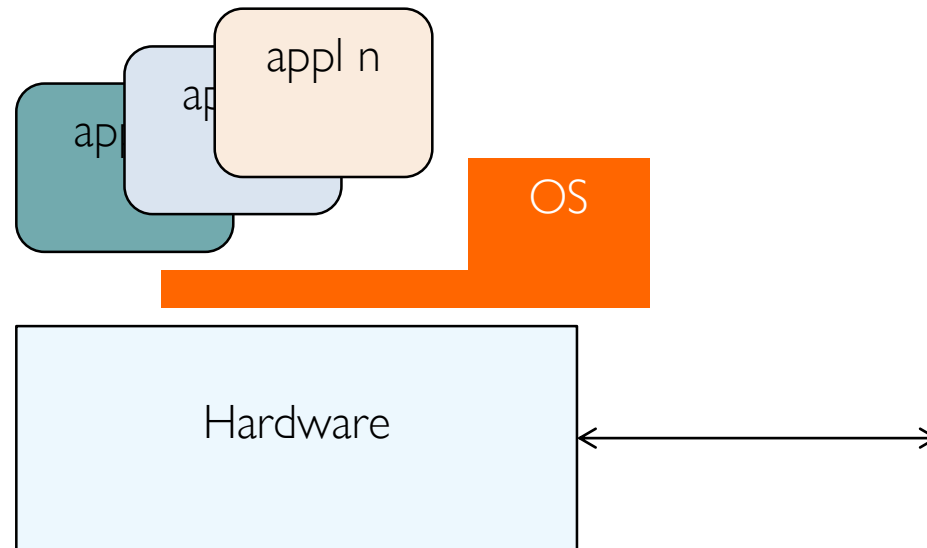
绝知此事要躬行

Our Roadmap

- ◆ What is an OS?
- ◆ What does an OS do?
- ◆ OS basics
- ◆ What is a process?
- ◆ What is a shell?
- ◆ What is a system call ?
- ◆ OS components

What is an OS

- ◆ **Special layer** of software that provides application software access to hardware resources:
 - ◆ Convenient abstraction of complex hardware device
 - ◆ Protected access to shared sources
 - ◆ Security and authentication
 - ◆ Communication among logical entities



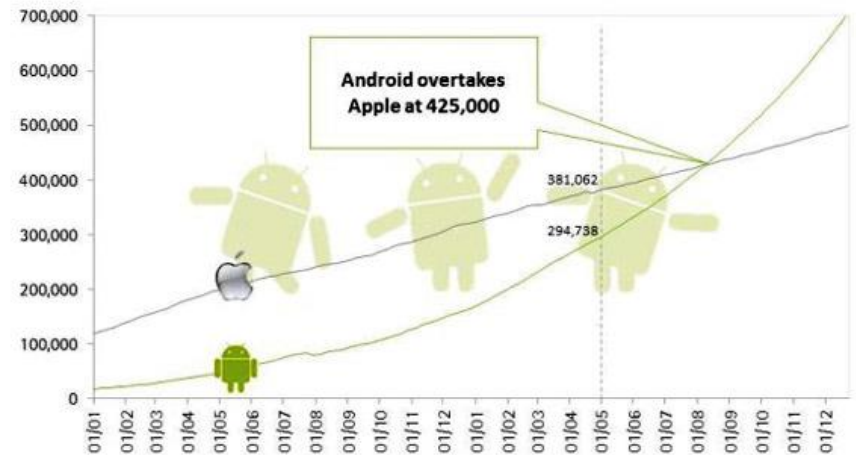
An OS

- ◆ Includes a **program**
 - ◆ called “**kernel**” (e.g., kernel.exe), which manages all the physical devices (e.g., CPU, RAM and hard disk)
 - ◆ exposes some functions as **system calls** for others to configure the kernel or build things (e.g., C library) on top
- ◆ Includes some more **programs**
 - ◆ called “**drivers**”, which handles the interaction between the kernel and the external devices (e.g., keyboard)
 - ◆ called a “**shell**”, which renders a simple command-line user interface with a full set of commands
 - ◆ ...
- ◆ Includes some “optional” **programs**
 - ◆ GUI, Browser, Paintbrush, ...

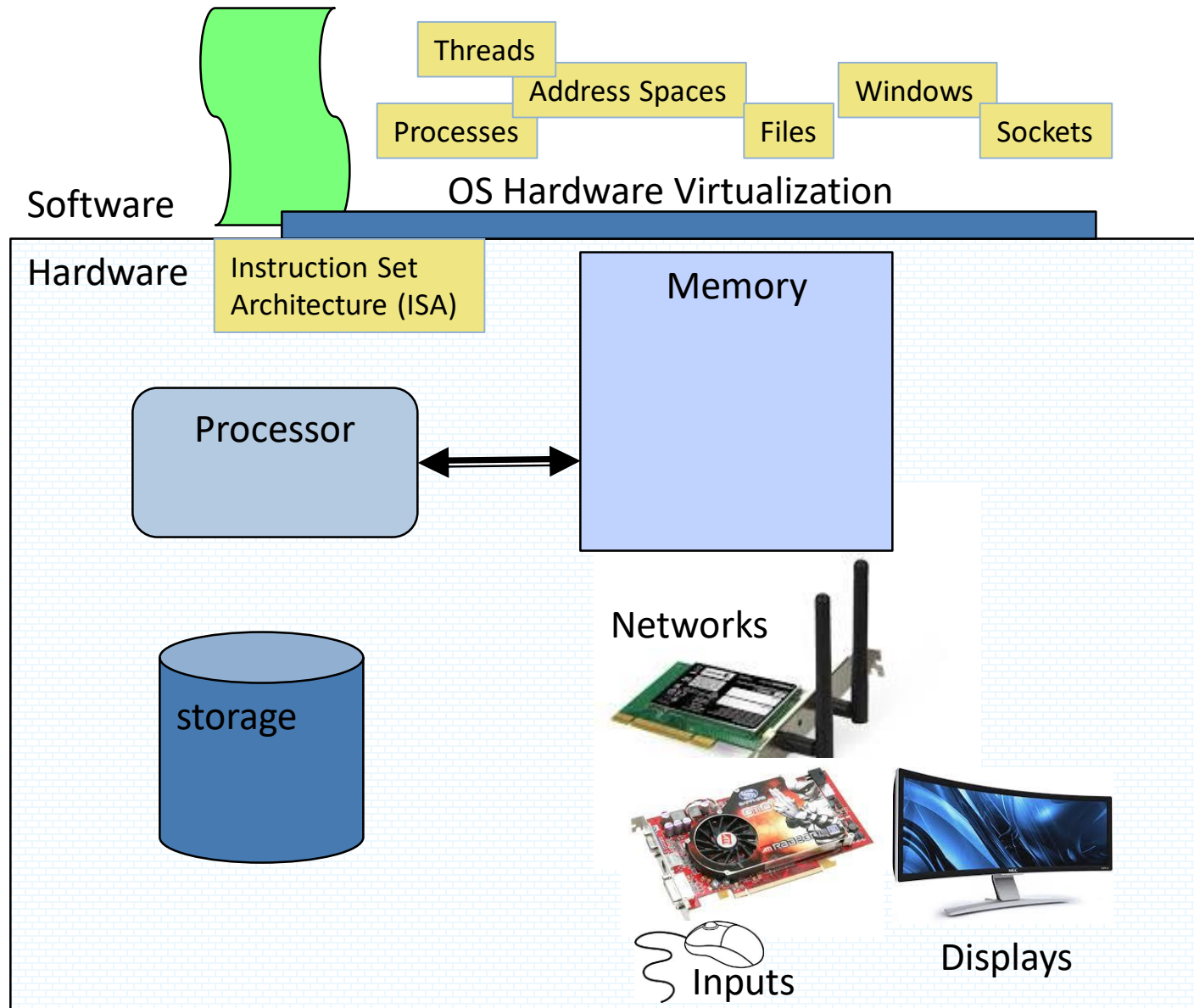
What does an OS do

- ◆ Provide abstractions to apps
 - ◆ File systems
 - ◆ Processes, threads
 - ◆ VM, containers,
 - ◆ ...
- ◆ Manage resources:
 - ◆ Memory, CPU, Storage,
 - ◆ ...
- ◆ Achieves the above by implementing specific algorithms and techniques
 - ◆ Scheduling
 - ◆ Concurrency
 - ◆ ...

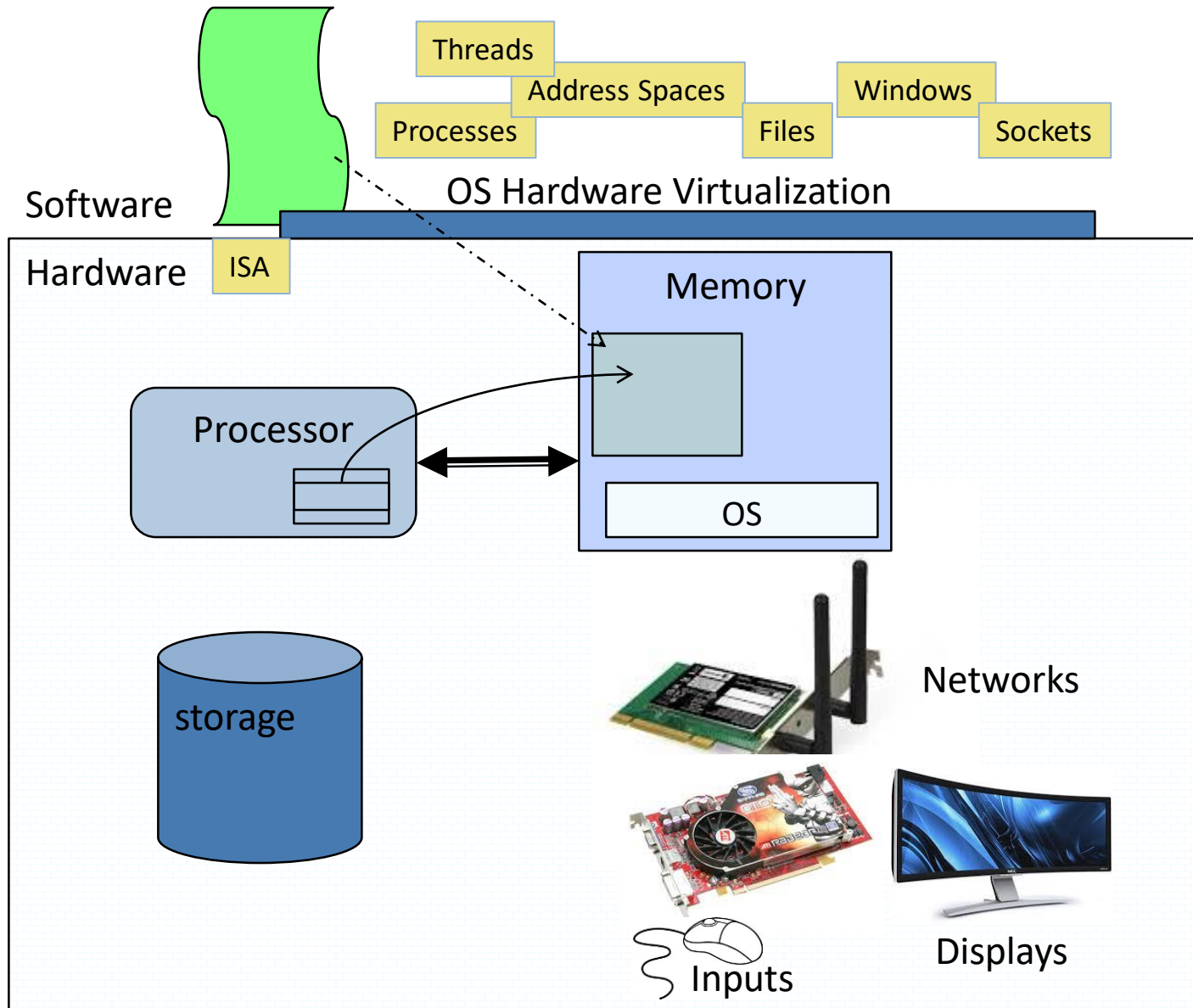
Number of apps in Apple App Store and Android Market (01/2010 – 12/2011E)



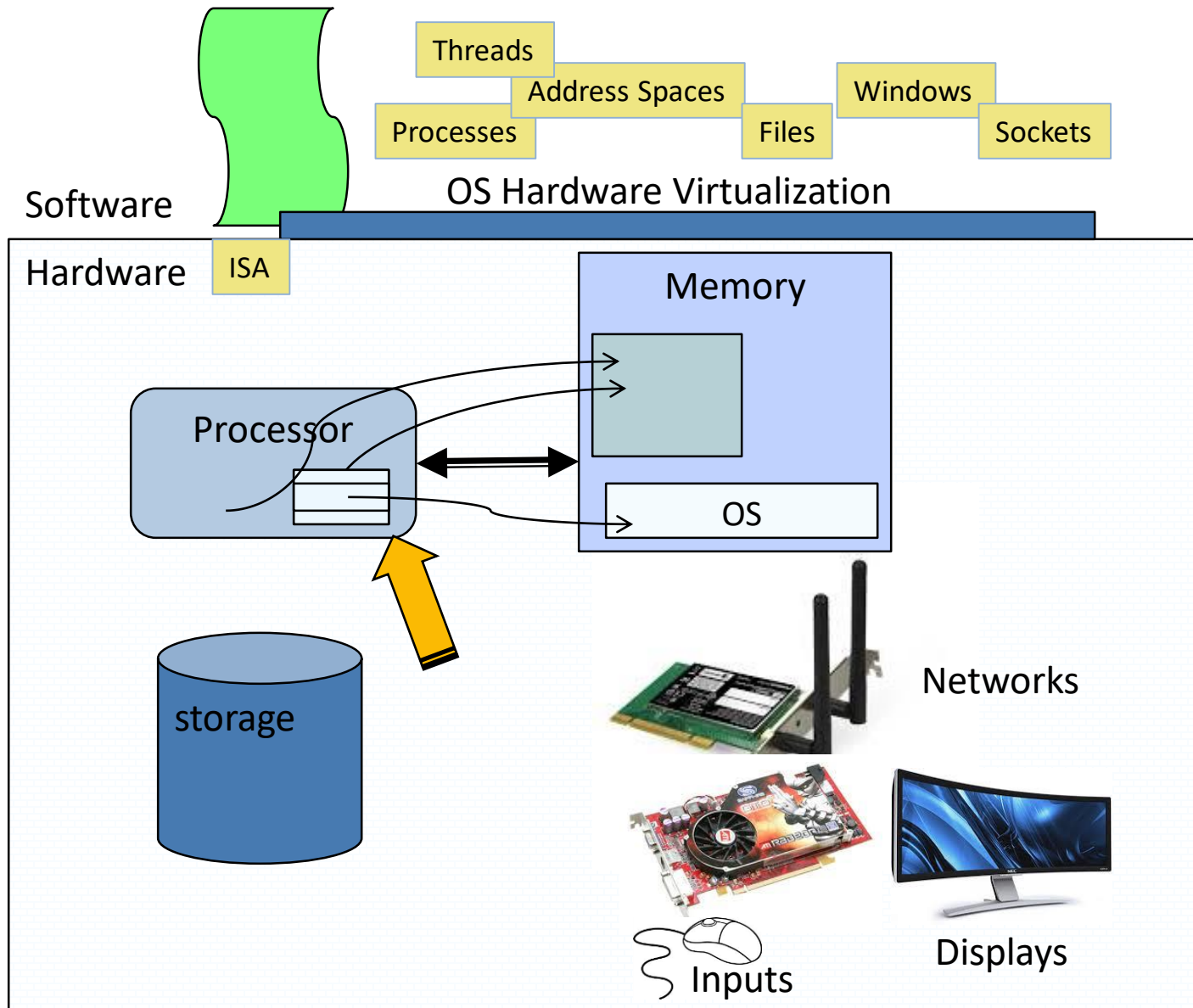
OS basics: “Virtual Machine” Boundary



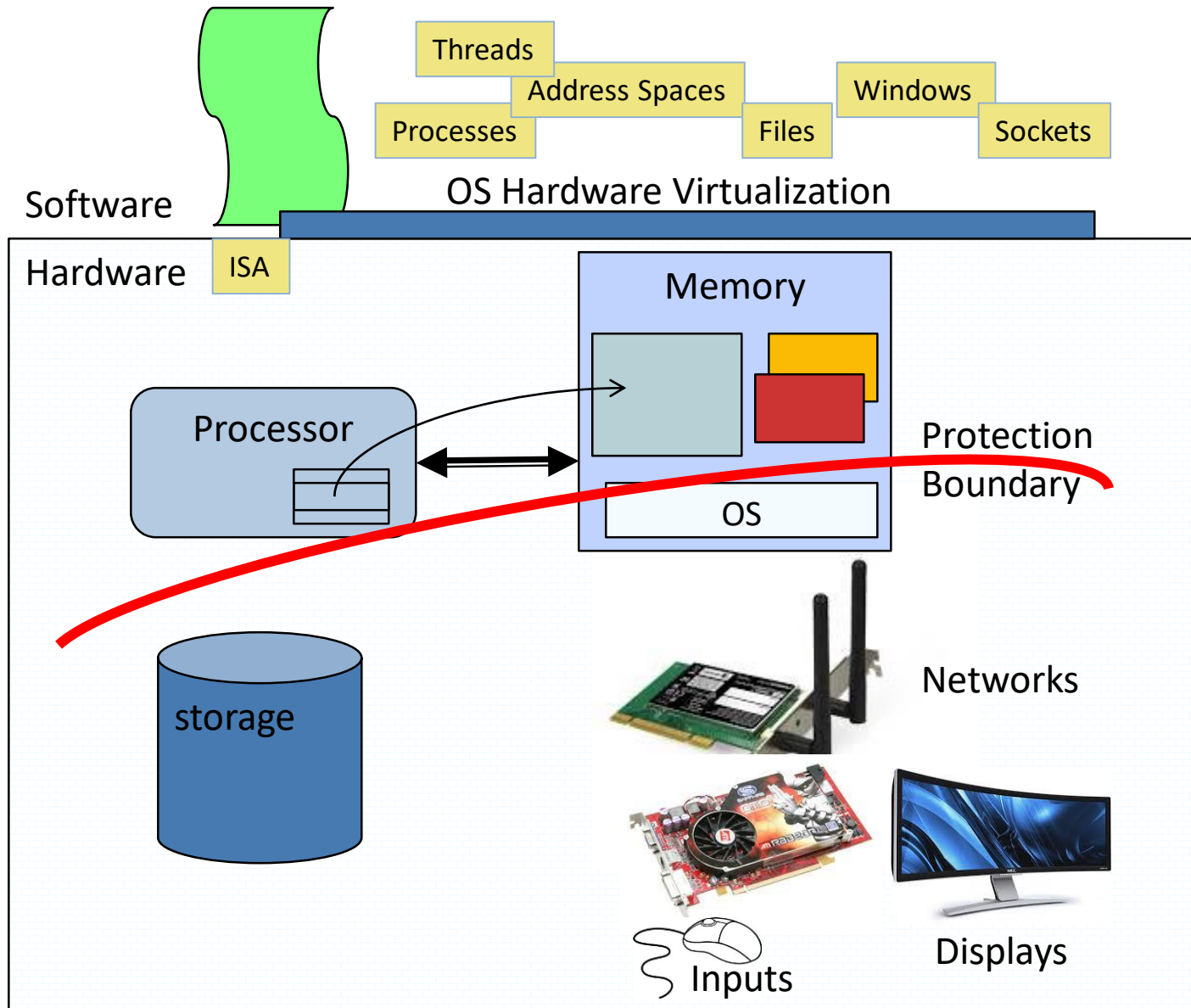
OS basics: Program and Process



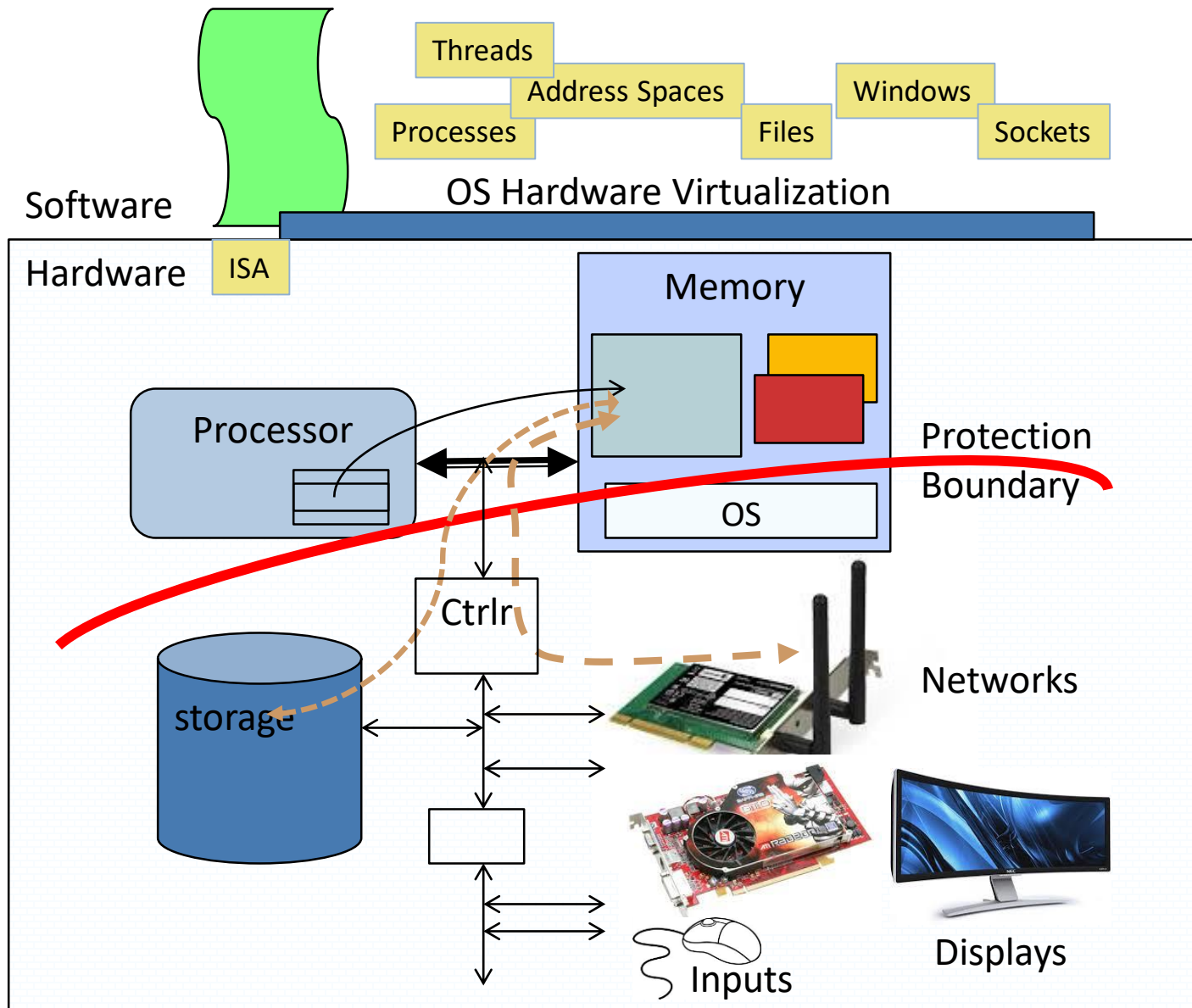
OS basics: Context Switch



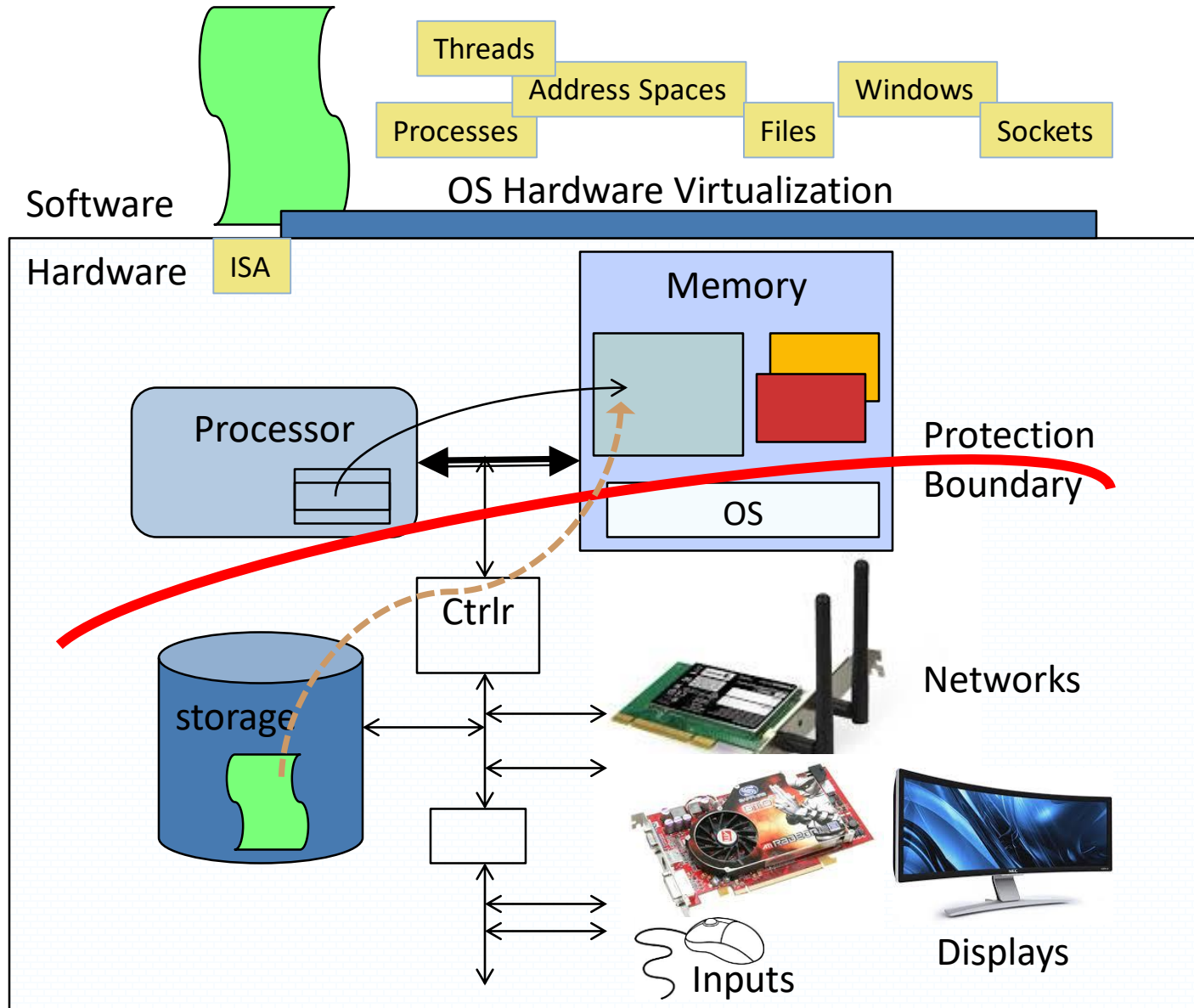
OS basics: Scheduling, Protection



OS basics: IO



OS basics: loading



What is a process

- ◆ A process is an **execution instance** of a program.
 - ◆ More than one process can execute the same program code
- ◆ Consider the following two commands:

Command A	<code>ls -R /</code>	Recursively print the directory entries, starting from the directory '/'
Command B	<code>ls -R /home</code>	Recursively print the directory entries, starting from the directory '/home'

They are **2** different processes

Process vs. Program

- ◆ A process has **states** concerning the execution. E.g.,
 - ◆ Which line of codes it is running
 - ◆ How much time left before returning the CPU to others
- ◆ Linux commands about processes
 - ◆ **ps**: “process status”, it 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.

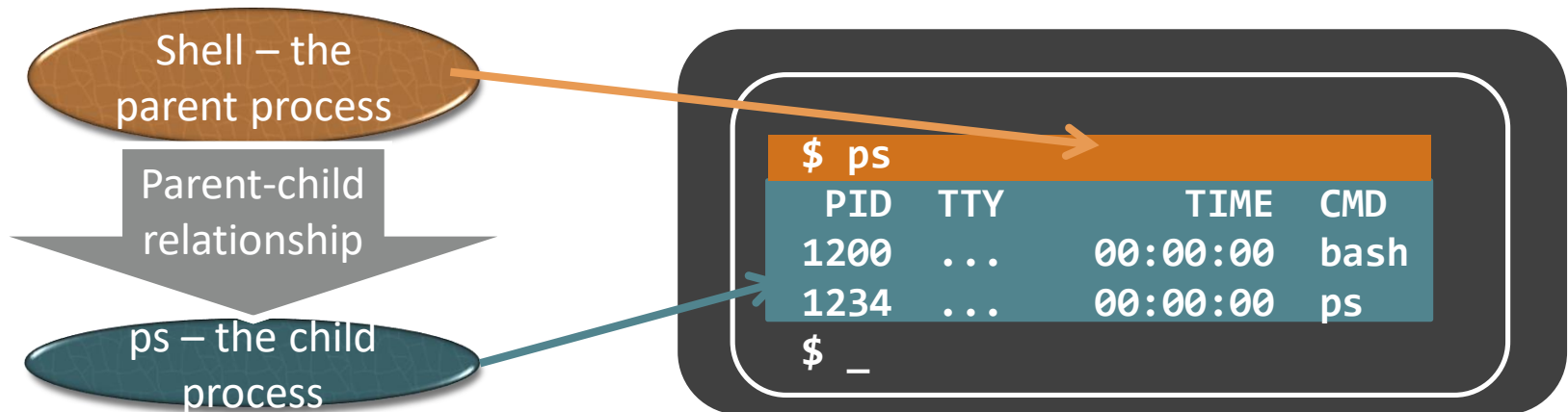
By the way, this is called **shell**.

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

- ◆ **top**: it allows users to monitor processes and system resource usage on Linux. It is interactive!

What is a Shell?

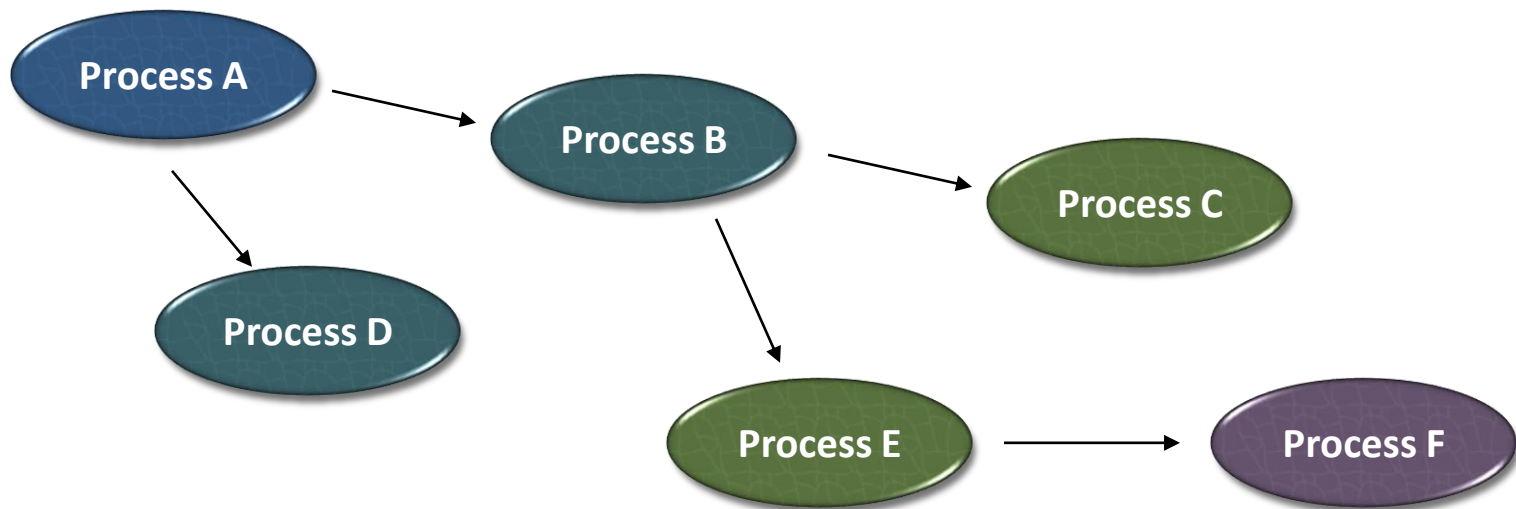
- ◆ A shell is a program, you open a “terminal”, which actually launches a “shell” process
 - ◆ Bash in linux
- ◆ Written in C
 - ◆ use `getchar()` (to get your command “**ps**”)
 - ◆ syntax checking
 - ◆ invoke a function `fork()` (a **system call**) to create a new process
 - ◆ i.e., becoming a **child process** of the shell.
 - ◆ Ask the the child process to `exec()` the program “**ps**”.



Process hierarchy

❖ Process relationship

- ❖ A parent process will have its child processes.
- ❖ Also, a child process will have its child processes.
- ❖ This forms a **tree hierarchy**.



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

What is a system call?

- ◆ System call
 - ◆ is a function call.
 - ◆ exposed by the **kernel**.
 - ◆ abstract away most low-level details.
 - ◆ Do you know how to read an input from keyboard?

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

Function
implementation.

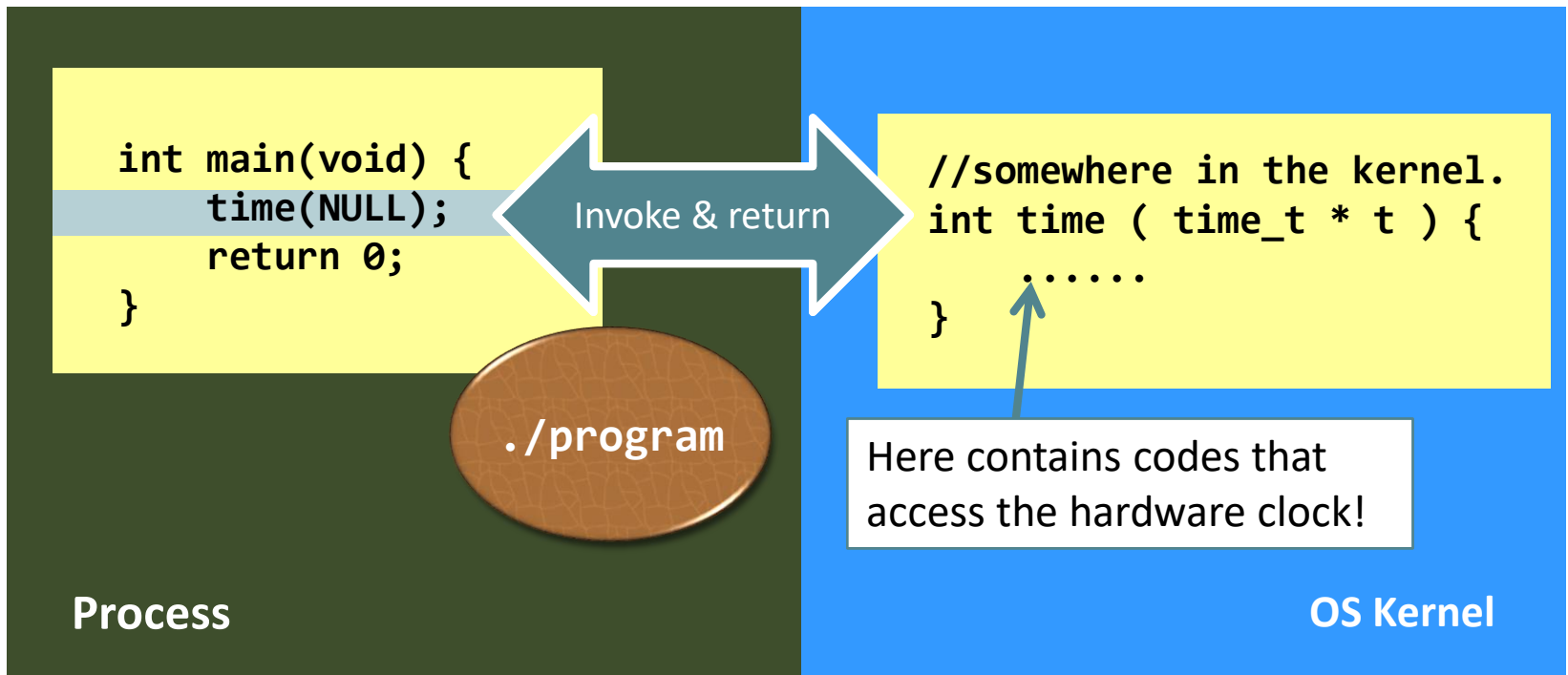
```
int main(void) {  
    int result;  
    result = add_function(a,b);  
    return 0;  
}
```

This is a
function call.

```
// this is a dummy example...
```


Interacting with the OS

How to measure the time cost of your program?



System calls

- ◆ Categorizing system calls:
 - ◆ Process, File system, Memory, Security, Device
- ◆ How can we know if a “function” is a system call
 - ◆ Read the man page “syscalls” under linux
- ◆ Pop quiz
 - ◆ Which of the following is/ are system call(s)?

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



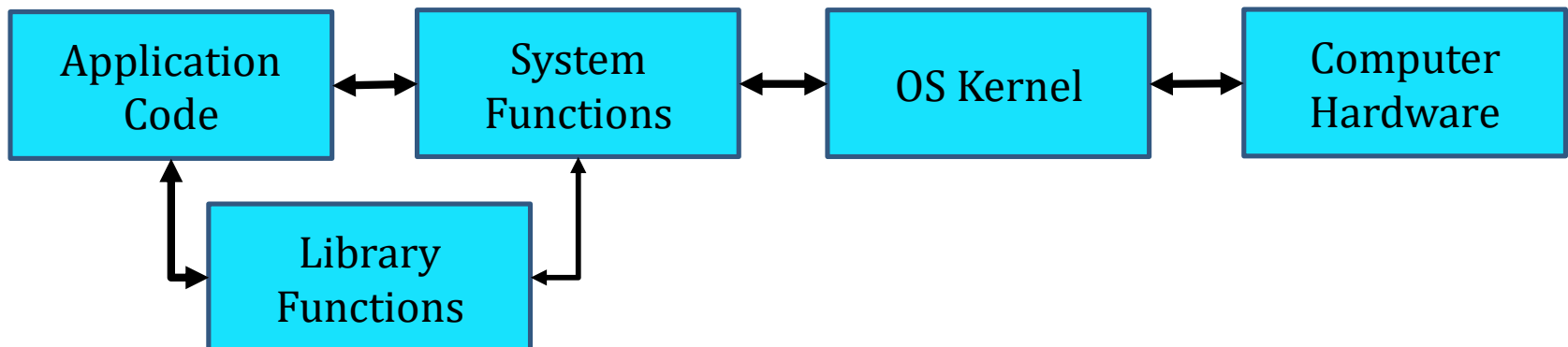
Who are they?

System calls VS Library 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!

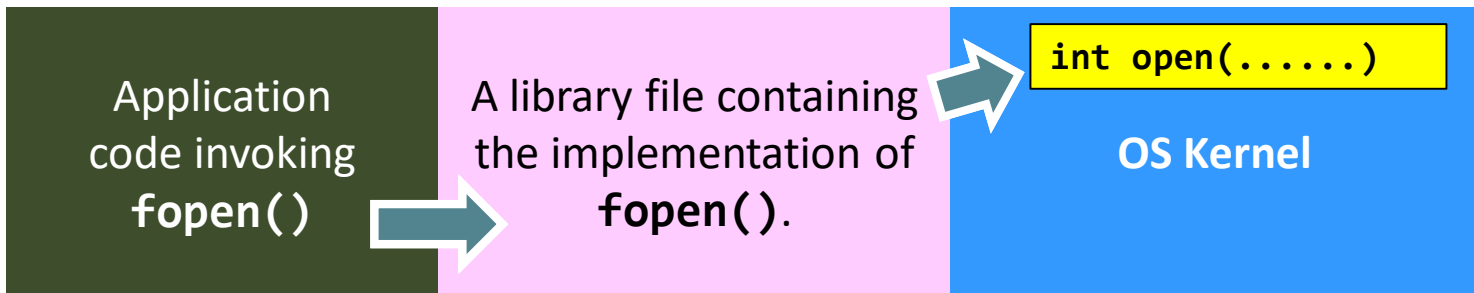
Library call	<code>fopen("hello.txt", "w");</code>
System call	<code>open("hello.txt", O_WRONLY O_CREAT O_TRUNC, 0666);</code>

- ◆ Function calls:



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:

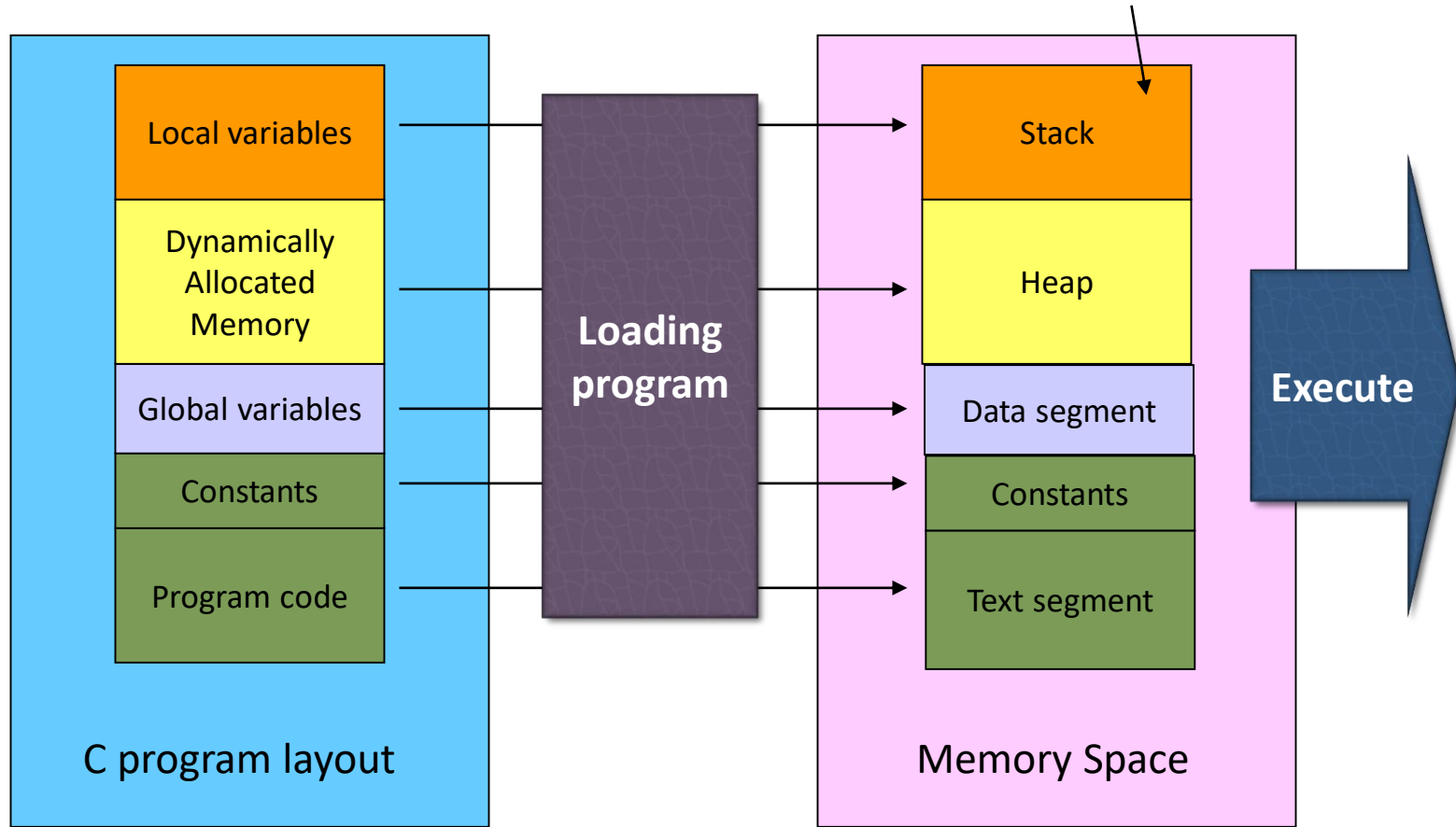


What will we learn about Process

- ◆ 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?
 - ◆ Which process shall get the core next?
- ◆ Signals
 - ◆ How to suspend a process?
 - ◆ A virus? We can make a program to play a song whenever you type **Ctrl+C**?
- ◆ Synchronization
 - ◆ How processes can cooperate to do useful work together?

The Memory of a Process

BTW, this arrangement is called s_____!

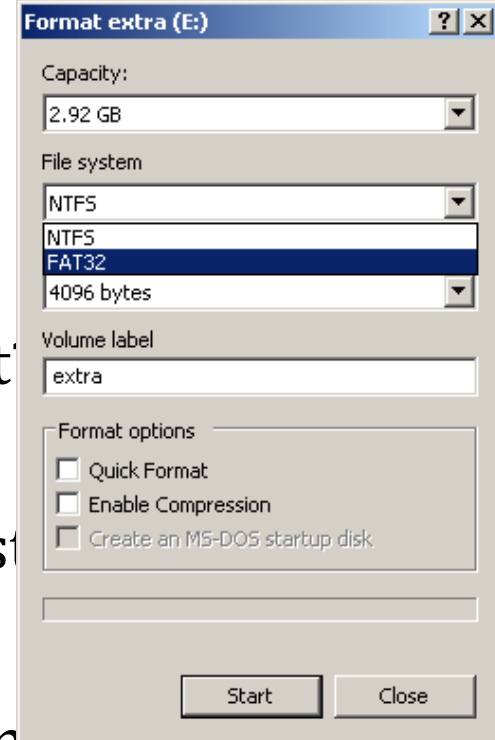


What will we learn about Memory

- ◆ Virtual memory
 - ◆ Your process virtually owns all your machine's RAM
- ◆ Memory-related functions
 - ◆ E.g., how to write “**malloc()**”?
- ◆ Stack overflow
 - ◆ Why & when?
- ◆ RAM = 256MB
 - ◆ **malloc(16MB)**
 - ◆ How much free memory left?

File System

- ◆ Have you heard of...
 - ◆ FAT16, FAT32, NTFS, Ext3, Ext4, BtrFS, Juliet
 - ◆ They are all file systems.
 - ◆ It is about how to organize your files in the storage
- ◆ If a FS just lays your files one-by-one, consecutively, tightly, in your hard disk, is it good?
 - ◆ What if you increase the size of your file?
 - ◆ What's the performance of searching for a file? $O(?)$
 - ◆ BTW, how to deal with directories?



Index

Metadata

Files / Data

FS vs OS

- ◆ Each disk can have multiple FSs
- ◆ An OS may understand different FSs

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

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

What will we learn about File System

- ◆ How to deal with directories?
- ◆ Implementation of some famous FS-es.
- ◆ Why does a file system perform badly?
- ◆ How to undelete a file?

More...

- ◆ Form programmer to a system programmer
- ◆ From system programming to programming a operating system
 - ◆ Multi-threading
 - ◆ Booting
 - ◆ Architectural Conscious OS programming
 - ◆ Lock-free programming
 - ◆ I/O
 - ◆ Virtualization

Thank You!