Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Part 1: Introduction

Unit 2

System Call Concept

fSO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Goals and bibliography

- Goals
  - To provide an **overview of computer operation**, highlighting the aspects that directly affect the operating system.
  - To introduce the **concept of system call**, as the mechanism for accessing operating system services.
  - To describe the **services** the operating system provides to users and processes.

- Bibliography

A. Silberschatz, P. B. Galvin. “Operating Systems Concepts” 8th ed. Chapters. 1 and 2

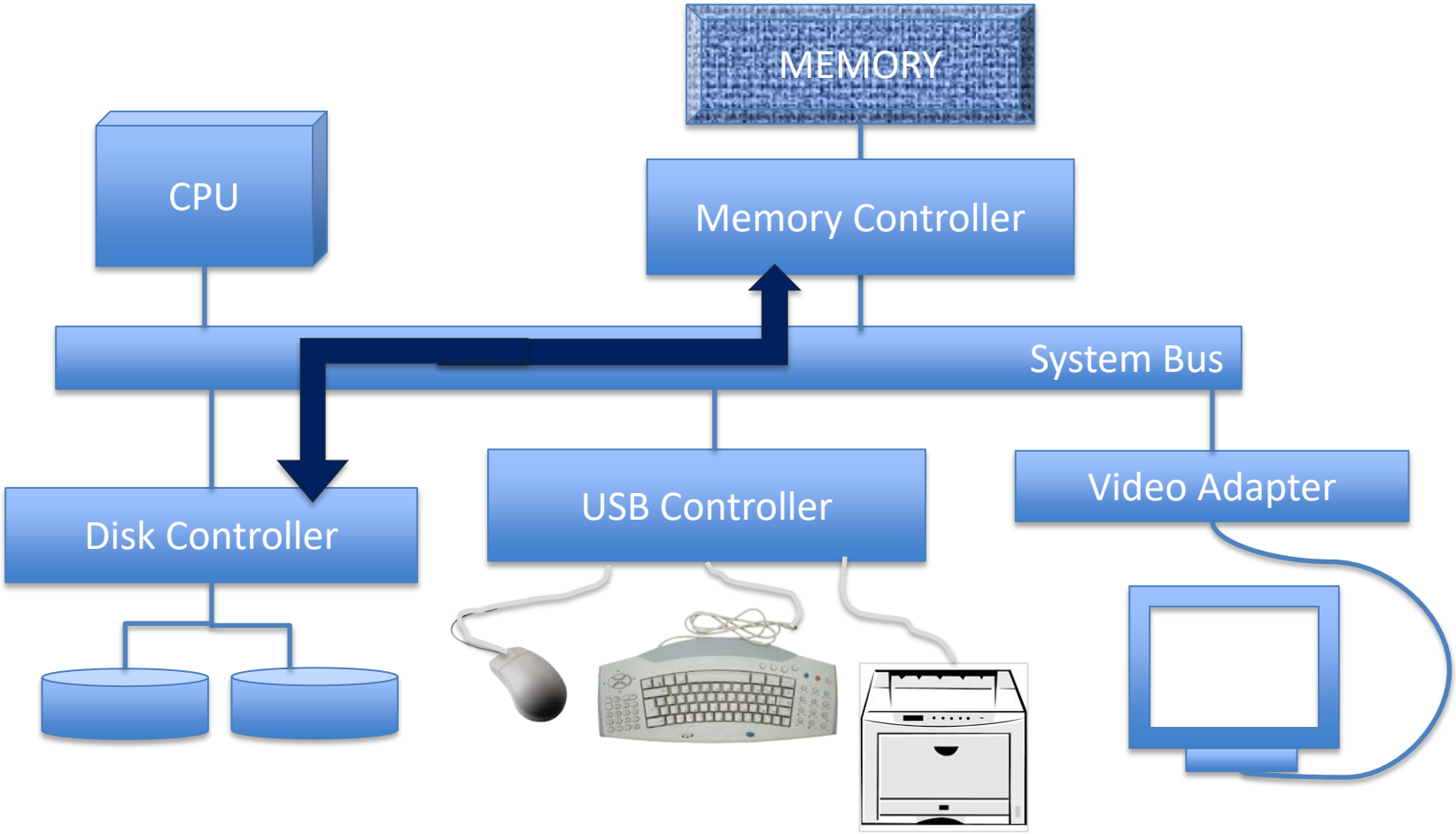
- **Computer hardware architecture**
- Interrupts
- Execution modes
- System calls
- System utilities

Terms:	
DMA	Direct Memory Access
SO	Operating System
USB	Universal Serial Bus
INT	Interrupt
POSIX	Portable Operting System Interface

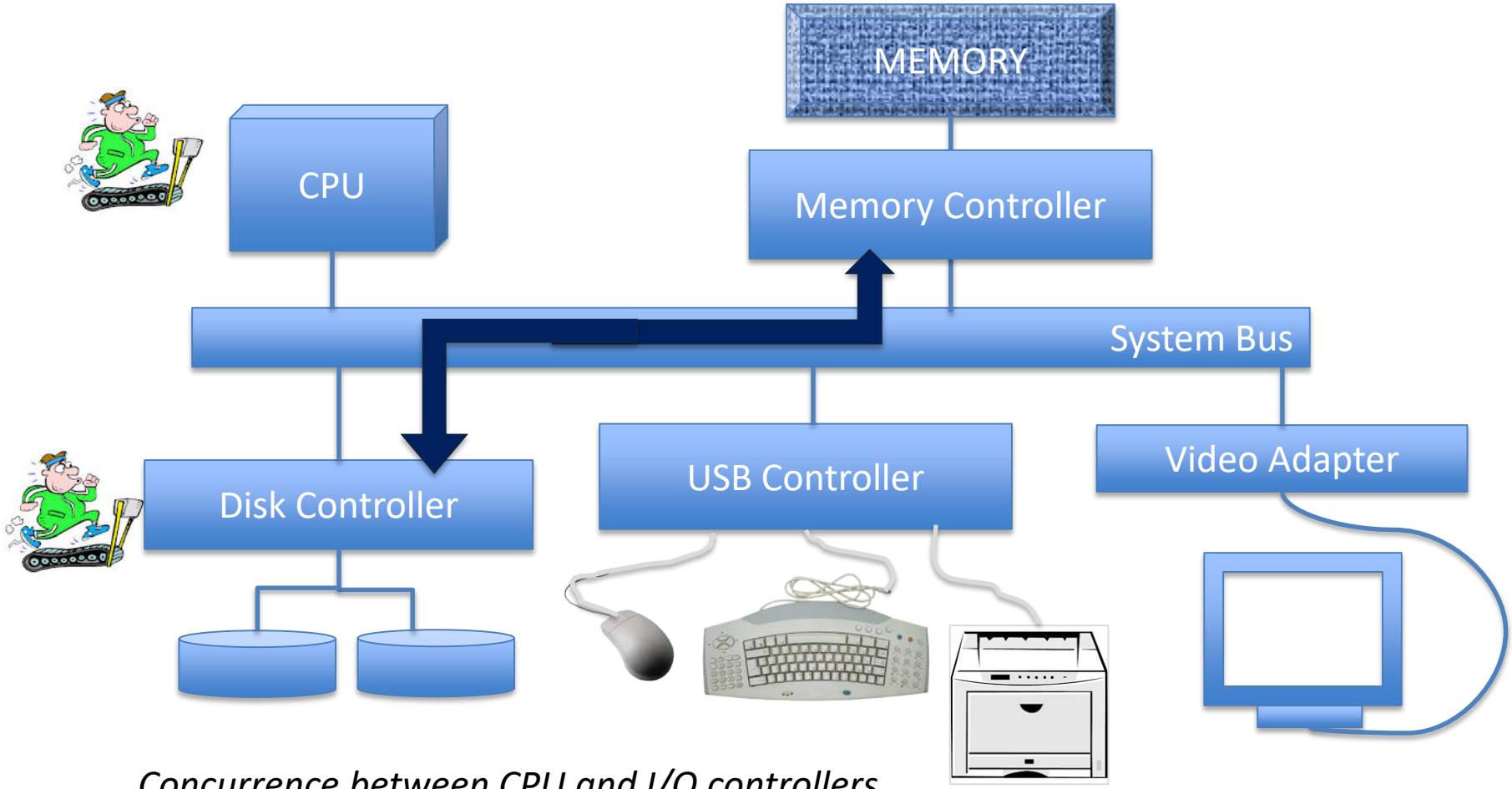
- Concurrency between I/O and CPU
  - **I/O devices are much slower than processors**
    - i.e. the time it takes to access information stored on disk
  - A modern processor can execute **billions of machine instructions** in the time it takes to access disk.
  - It is important that while performing I/O, the processor can execute useful instructions instead of waiting

    → **Concurrency between CPU and I/O**

- Diagram of computer operation

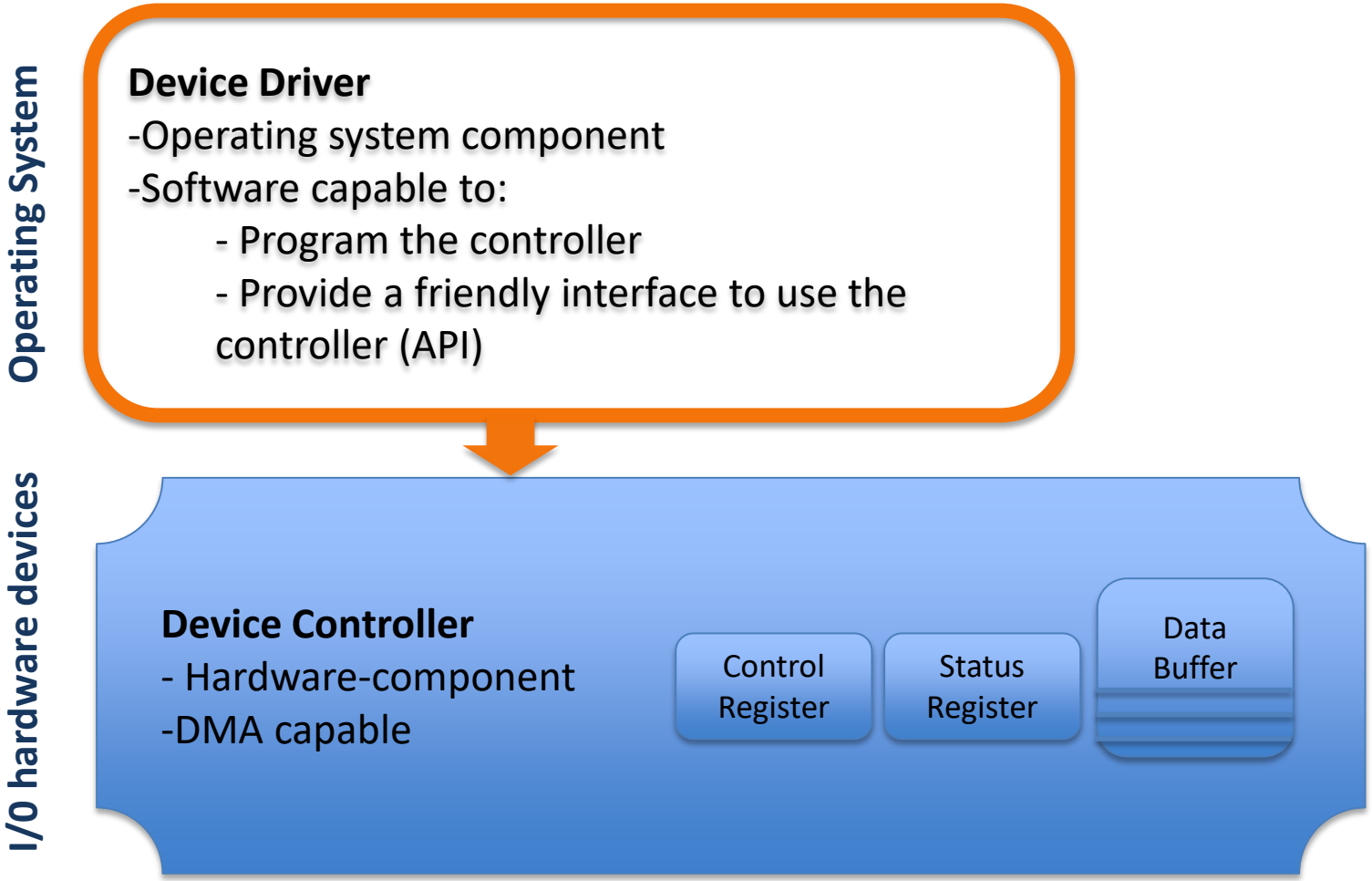


- Diagram of computer operation



Concurrence between CPU and I/O controllers
CPU and I/O devices work simultaneously along time

- Device driver vs device controller



Content

- Computer hardware architecture
- **Interrupts**
- Execution modes
- System calls
- System utilities

Interrupts

- An OS is an **event-driven program**
- Events are **hardware interrupts**, **software interrupts** and **exceptions**
- The operating system **acts as a server** program waiting for work that is commissioned by interrupts
- OS processes and I/O devices perform OS service requests

- What generates an interrupt request?
- When an interrupt request occurs?

INTERRUPT

I/O interrupt:
Generated by I/O device controllers

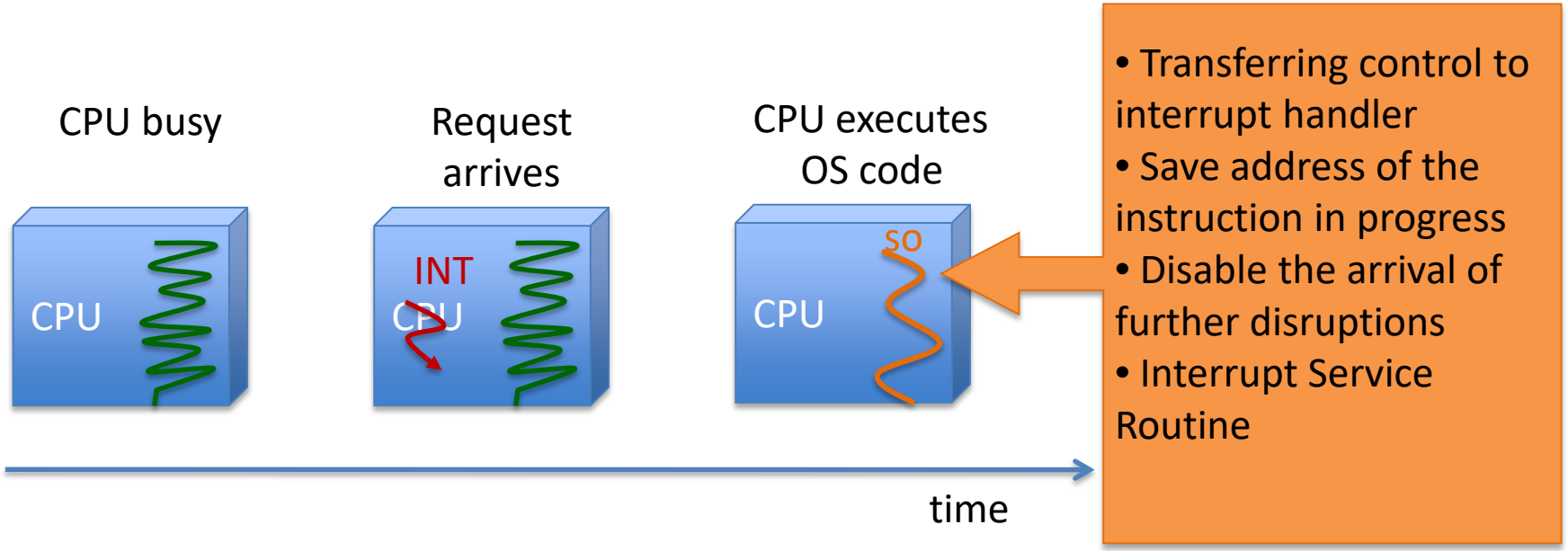
Clock interrupt:
The OS enters execution at certain time intervals

Hardware exception:
Memory parity error, power outage ...

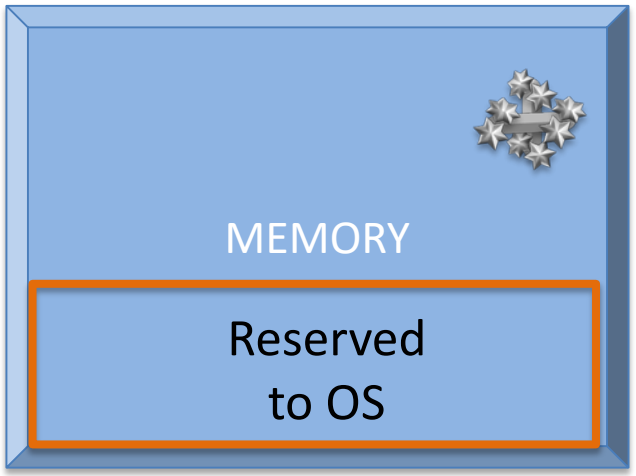
Traps:
Used by programs to request OS services

Software exceptions:
Are generated when happen some events like a division by zero, an arithmetic overflow, an addressing to a memory location prohibited, etc

- Interrupt mechanism



Interrupt handler
Interrupt vector
Interrupt service routines



Content

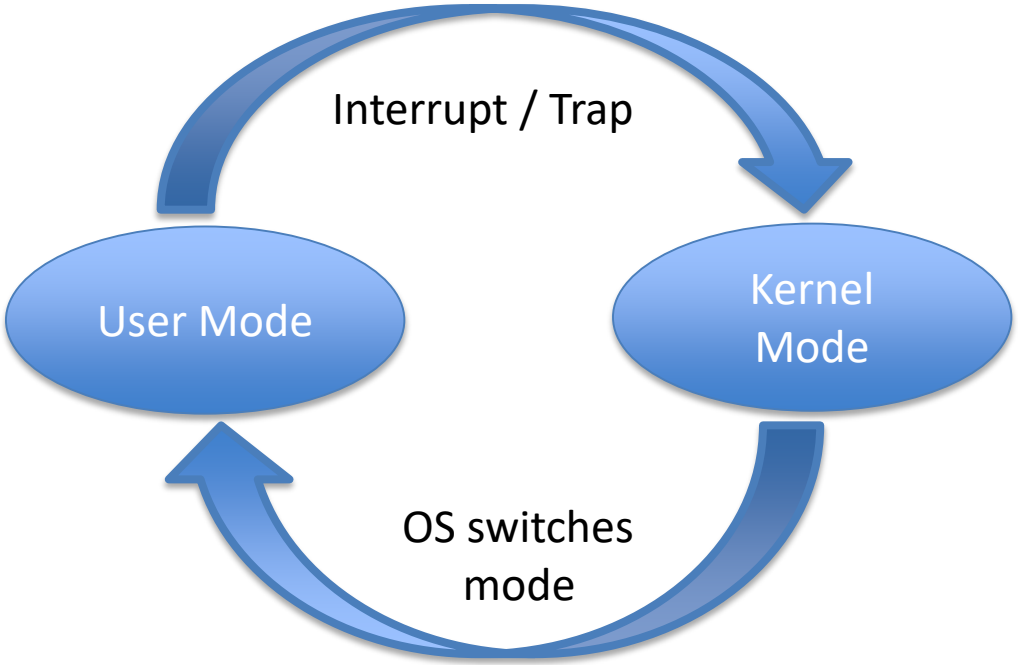
- Computer hardware architecture
- Interrupts
- **Execution modes**
- System calls
- System utilities

Execution modes

- Processors have **two or more execution modes**
  - Execution modes are included to **support the operating system**
  - Processes running simultaneously share machine resources → **they need protection**
- **Protecting access to the hardware** to prevent user programs to:
  - Access system **memory** freely
  - Monopolize the **CPU** and system registers
  - Direct access to **I/O devices**
- Processors have a **mode bit** that indicates the actual operation mode**: *kernel (0) or user (1)***

Execution modes

- Two modes of execution
  - **User Mode or Regular:** has restricted instruction set
  - **Kernel mode or Privileged:** Run any type of hardware operations, memory access and I/O device
- Dual operation mode



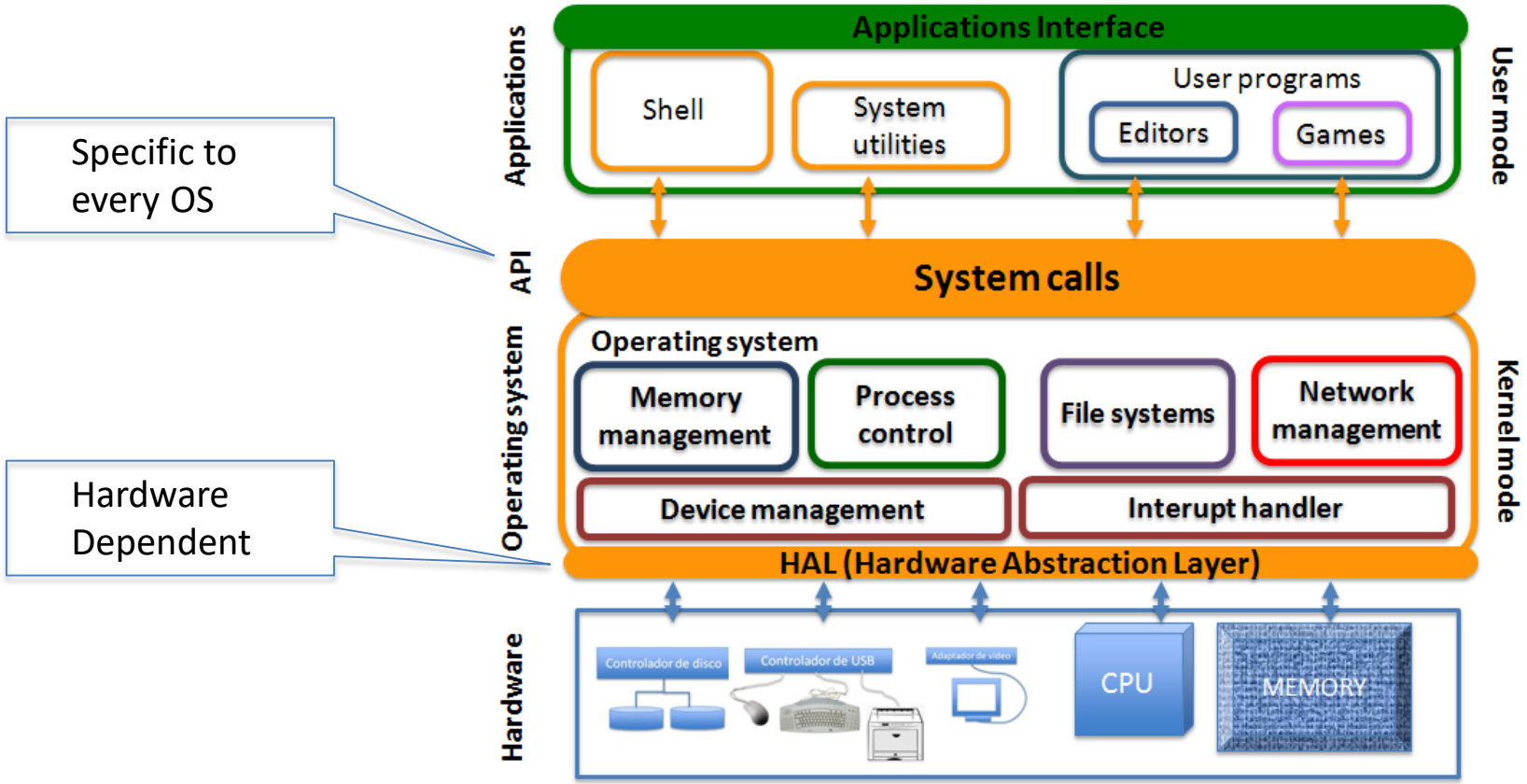
Execution modes

- Privileged instructions
  - **Instructions available in kernel mode** and which are not in user mode are called privileged instructions.
  - Privileged instructions are mainly associated with three types of protection
    - I/O protection
    - Memory protection
    - Process protection

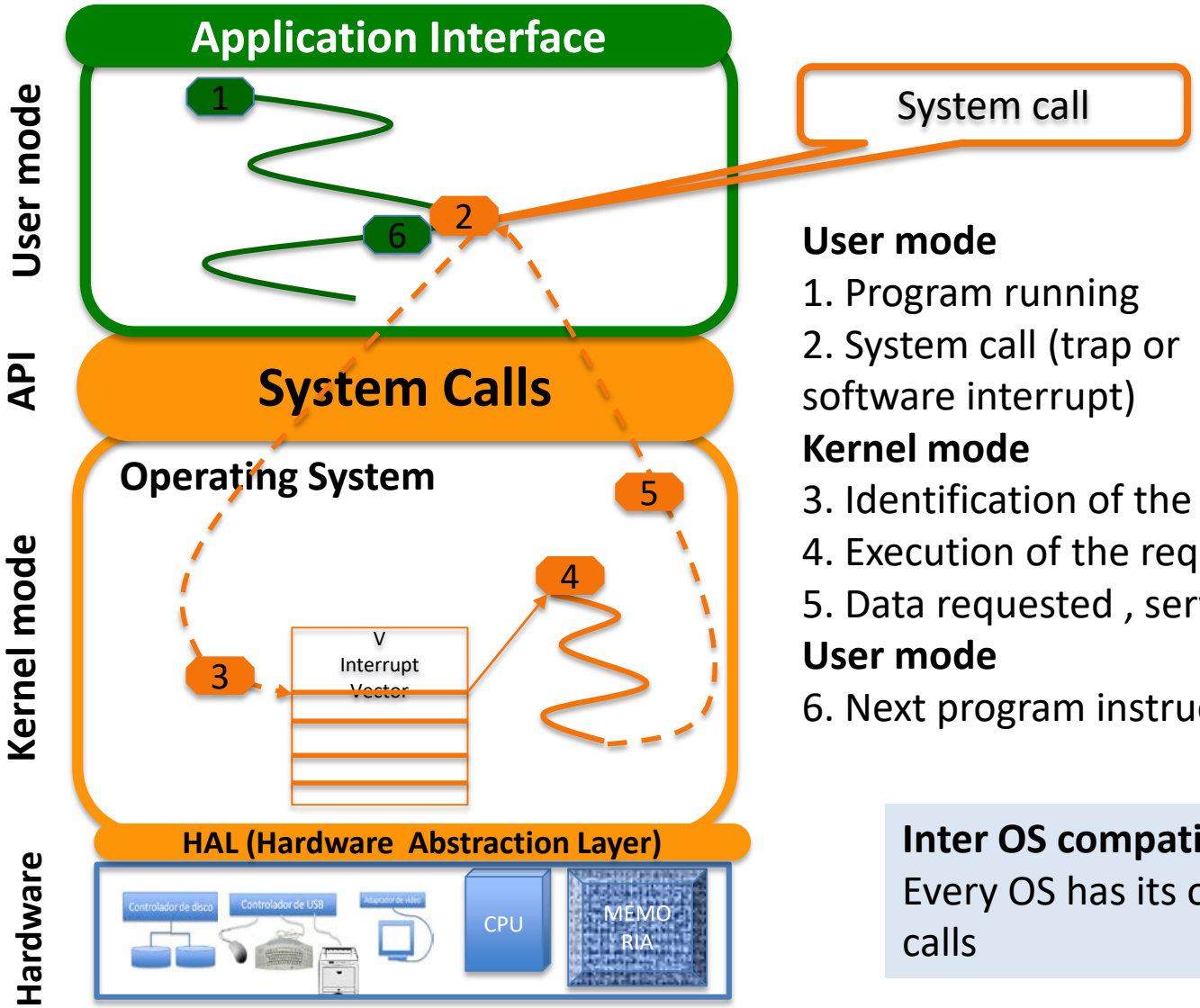
- Computer hardware architecture
- Interruptions
- Execution modes
- **System calls**
- System utilities

System calls

- OS mechanism to **provide services on-demand**.
- **Interface** that defines the operation set provided by the OS to allow accessing the machine resources.
- It is implemented as **C library functions**.



- Service request to the OS



User mode
1. Program running
2. System call (trap or software interrupt)

Kernel mode
3. Identification of the requested service
4. Execution of the requested service
5. Data requested , service result

User mode
6. Next program instruction

Inter OS compatibility? NO
Every OS has its own system calls

- POSIX system calls sample set
  - POSIX (Portable Operting System Interface)

	Processes
fork	Creates a child process
exit	Ends process execution
wait	Waits for a process end
exec	Executes an specicfic program from another program
getpid	Get process attributes
setsid	Changes process attributes

	Protection
chmod	Sets file permission bits (rwx, suid, sgid, etc)
chown	Sets file group and owner
umask	Set the protection mask of a process

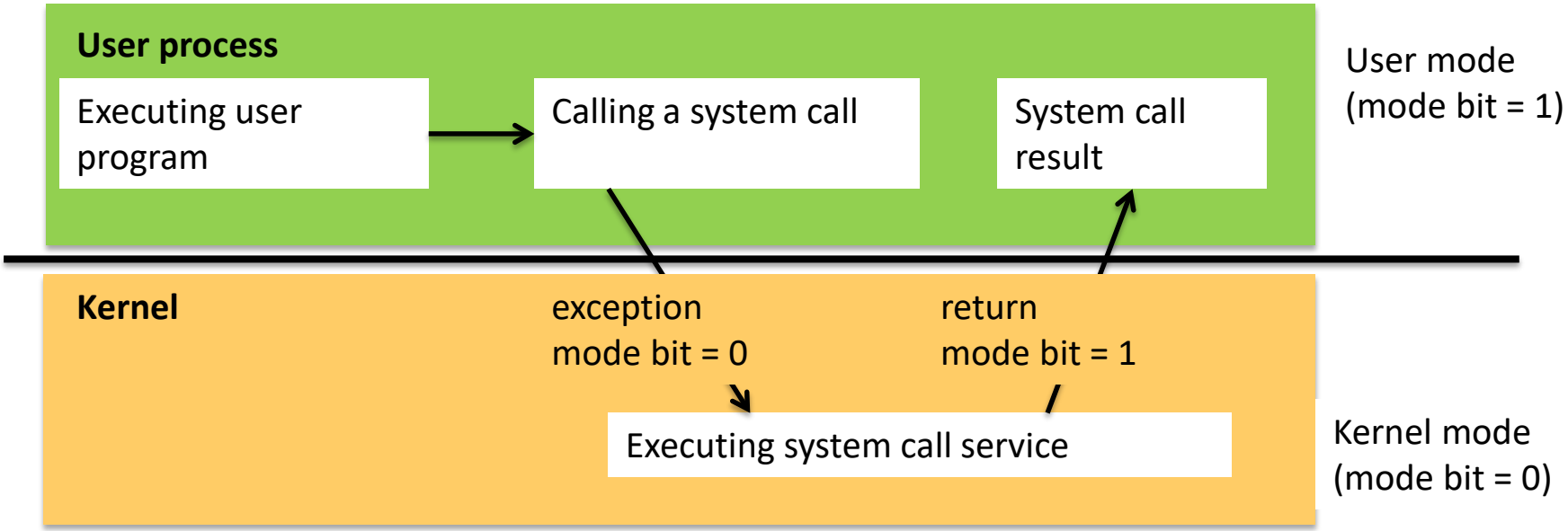
	Directories
mkdir	Create directory
rmdir	Remove an empty directory
opendir	Open directory
readdir	Get the next directory entry
closedir	Close directory
link	Get information about file i-node
unlink	Remove a directory entry

	Signals
kill	Send signals
alarm	Generate an alarm (clock signal)
sigemptyset	Init a mask with no signals set
sigfillset	Init a mask with signals set
sigaddset	Append a signal to a signal set
sigdelset	Delete a signals in a signal set
sigismember	Check if a signals belongs to a signal set
sigprocmask	Check/Modify/Set a signal mask
sigaction	Capture/Manage a signal
sigsuspend	Wait signals capture

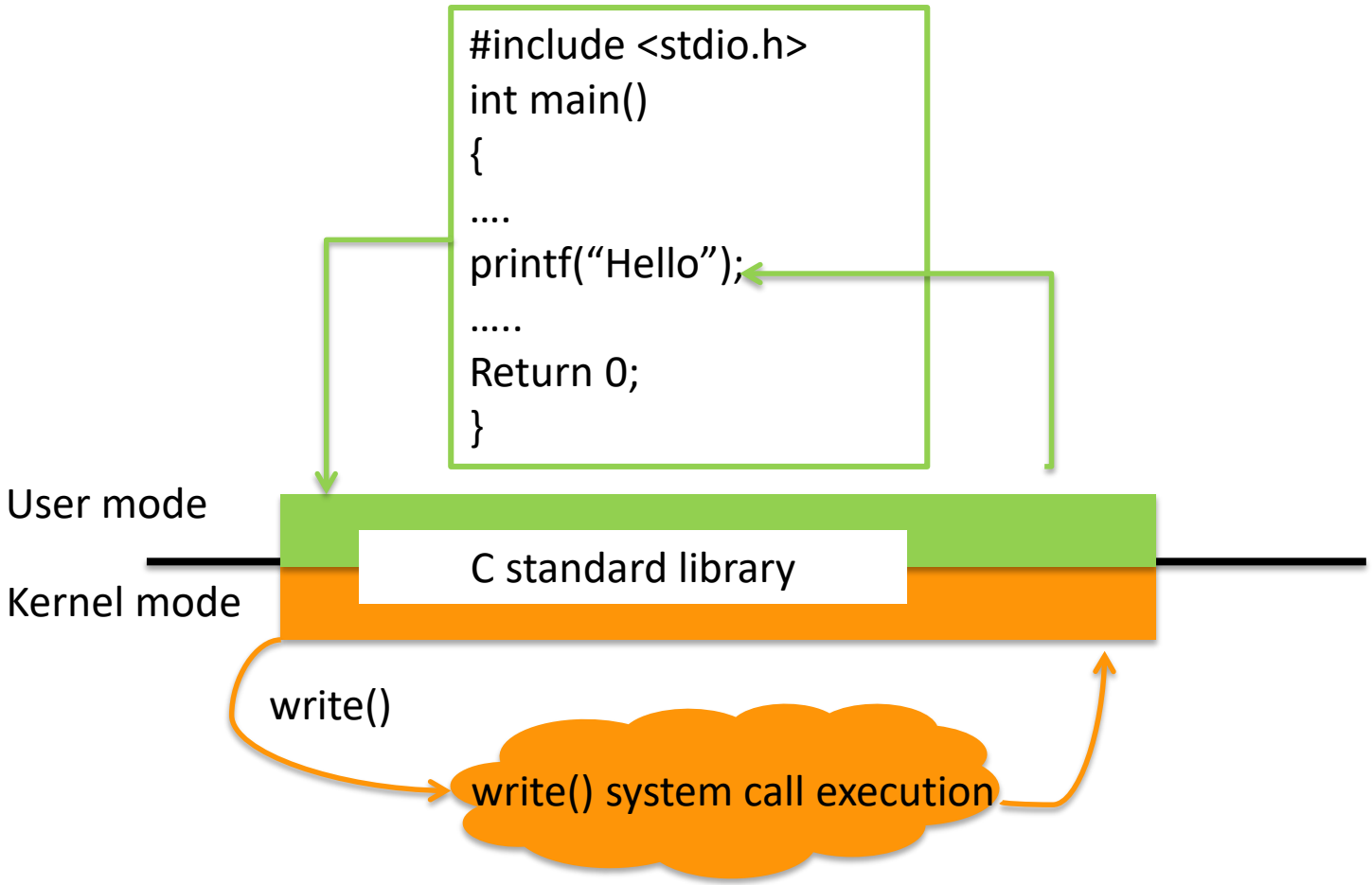
	Files
Open	Open/Create files
read	Read files
write	Write files
close	Close files
lseek	Locate read/write in a file
stat	Get information about file i-node
dup2	Duplicate a file descriptor
pipe	Create a pipe
mkfifo	Create a named pipe (fifo)

System calls

- The **mode bit** diferentiates OS executing tasks from user executing tasks
- Whe the system starts up, the hardware is initialized in kernel mode (mode bit = 0), then the OS is loaded in main memory and the user applications can be launched (mode bit = 1)



- **C language standard libraries**
  - They provide a portable interface to many system calls



Content

- Computer hardware architecture
- Interruptions
- Execution modes
- System calls
- **System utilities**

System utilities

- Programs that run as user processes and provide a **more comfortable environment**.
- They are provided as part of the OS, but are not essential for machine operation.
- UNIX Examples
  - File management: mkdir, cp, mv, ls ……
  - Filters: grep, sort, head, tail ….
  - Editors, compilers, assemblers, linkers …
  - Window systems : X11
  - Communication : mail, ftp, rlogin ….
  - Shells : sh, ksh, bash

- Code sample "*my_copy*"

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
  if (argc!=3) {
  fprintf(stderr,"Use: %s forig"
    " fdest\n", argv[0]);
  exit(1); }
  copy(argv[1],argv[2]);
  return 0;
}
```

```
#define BUFSIZE 1024

void copy(char *from, char *to)
{
  int fromfd, tofd, nread;
  char buf[BUFSIZE];

  if ((fromfd = open(from, O_RDONLY)) == -1)
    { perror(from); exit(1) }
  if ((tofd = creat(to,0666)) == -1)
    { perror(to); exit(1); }
  while ((nread = read(fromfd, buf, sizeof (buf))) > 0)
      if (write(tofd, buf, nread) != nread)
        { perror("write"); exit(1); }
  if (nread == -1) perror("read");
  if (close(fromfd) == -1 || close(tofd) == -1)perror("close");
}
```