

# Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)

*Universitat Politècnica de València*

## Part 2: Process Management

### Unit 3

## Process concept and implementation

fSO

DISCA



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

- **Goals**

- To define **process** concept
- To analyze the differences between **sequential** and **concurrent** execution
- To define the **process states** as well as the causes of the transitions between them
- To study the **basic structures to support processes** in the OS
- To analyze the **context switch** mechanism

- **Bibliography**

- “Operating System Concepts” Silberschatz 8ª Ed
- “Sistemas operativos: una visión aplicada” Carretero 2º Ed

- **Previous concepts**
- Executable files
- Process concept
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- Exercises

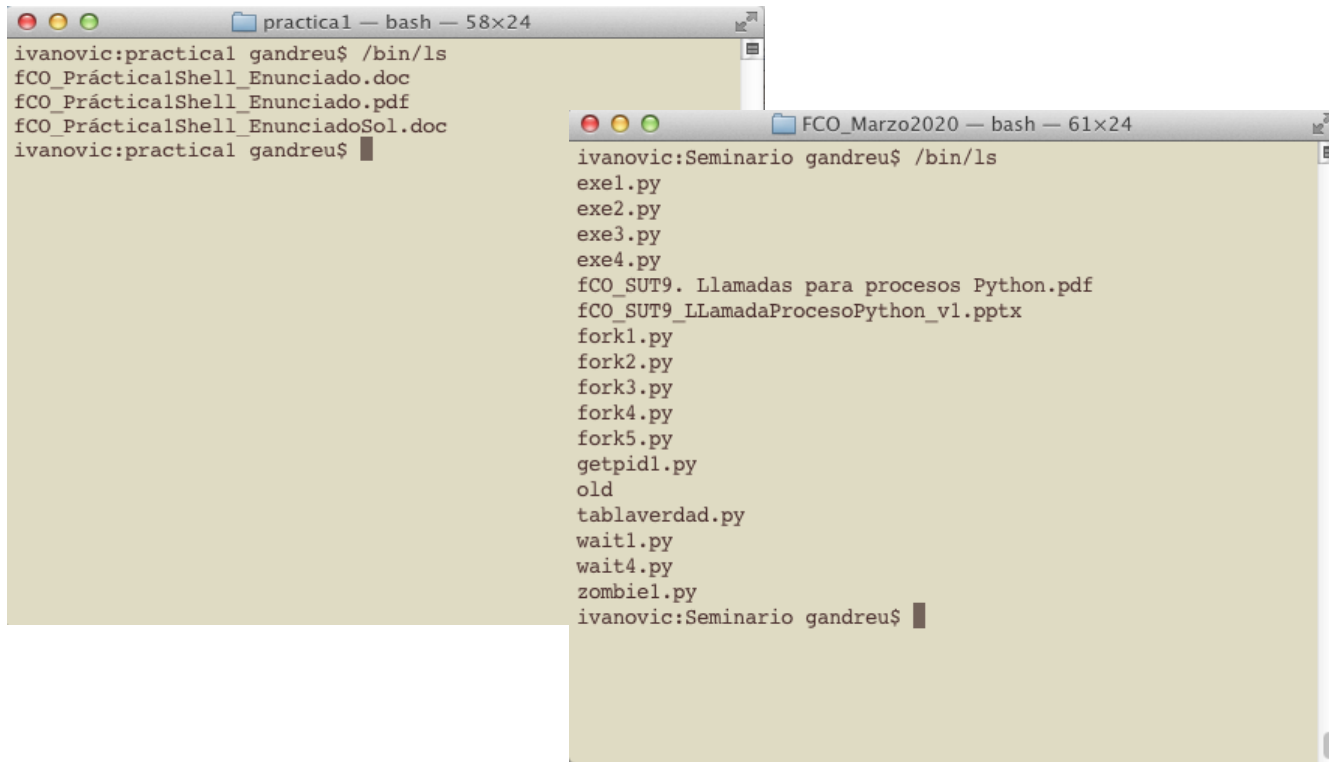
- In order to understand the process concept some terms have to be clarified, particularly the differences between the following pairs of concepts:
  - Program / Process
  - Sequential execution / Concurrent execution

## Program vs. Process

- **Program:**
  - **File** in executable format generated by a compilation of source code followed by a linking operation.
  - **Passive entity** it doesn't change along time.
- **Process:** a certain program running:
  - Working unit for the OS
  - Resource consumer
  - Every activity happening on a computer is related to a process
  - **Abstract entity belonging to the OS** that allows modeling computer activity

A process is an alive entity that experiments changes while it exists

- **Program versus Process**
  - The same program can be run multiple times concurrently



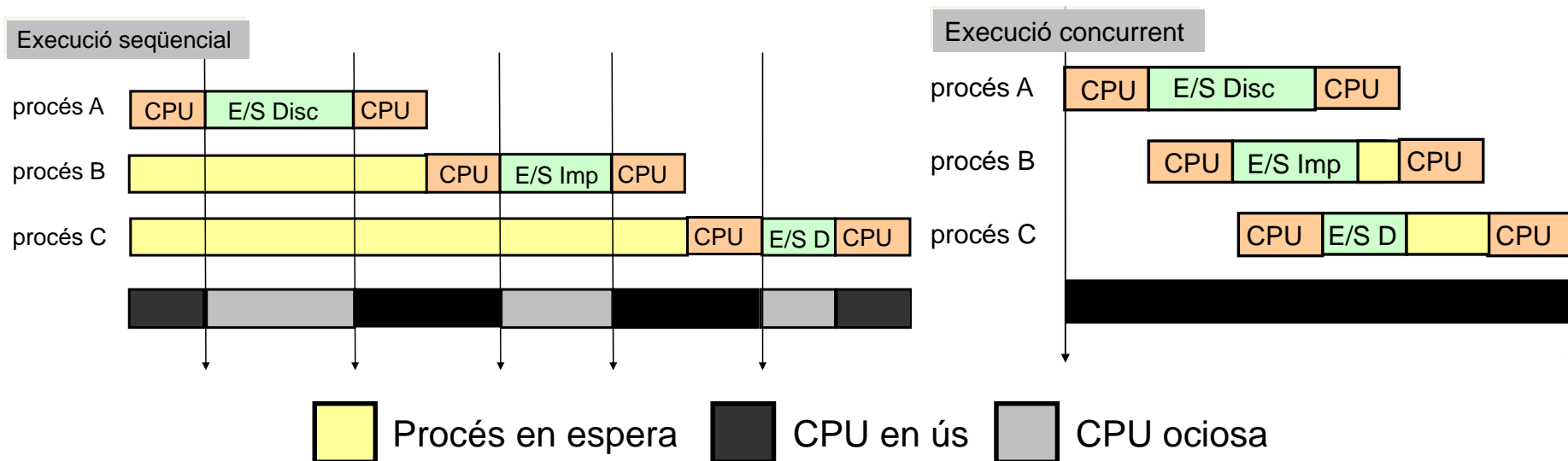
The image shows two terminal windows side-by-side. The left window is titled 'practical1 — bash — 58x24' and shows the command `/bin/ls` being executed in the directory `ivanovic:practical gandreu$`. The output lists files: `fCO_PracticalShell_Enunciado.doc`, `fCO_PracticalShell_Enunciado.pdf`, and `fCO_PracticalShell_EnunciadoSol.doc`. The right window is titled 'FCO\_Marzo2020 — bash — 61x24' and shows the same command `/bin/ls` being executed in the directory `ivanovic:Seminario gandreu$`. The output lists a different set of files, including `exe1.py` through `exe4.py`, `fCO_SUT9. Llamadas para procesos Python.pdf`, `fCO_SUT9_LlamadaProcesoPython_v1.pptx`, `fork1.py` through `fork5.py`, `getpid1.py`, `old`, `tablaverdad.py`, `wait1.py`, `wait4.py`, and `zombiel.py`.

```
ivanovic:practical gandreu$ /bin/ls
fCO_PracticalShell_Enunciado.doc
fCO_PracticalShell_Enunciado.pdf
fCO_PracticalShell_EnunciadoSol.doc
ivanovic:practical gandreu$

ivanovic:Seminario gandreu$ /bin/ls
exe1.py
exe2.py
exe3.py
exe4.py
fCO_SUT9. Llamadas para procesos Python.pdf
fCO_SUT9_LlamadaProcesoPython_v1.pptx
fork1.py
fork2.py
fork3.py
fork4.py
fork5.py
getpid1.py
old
tablaverdad.py
wait1.py
wait4.py
zombiel.py
ivanovic:Seminario gandreu$
```

- The file with executable code `/bin /ls` is always the same
- Every time it is run a different process is created

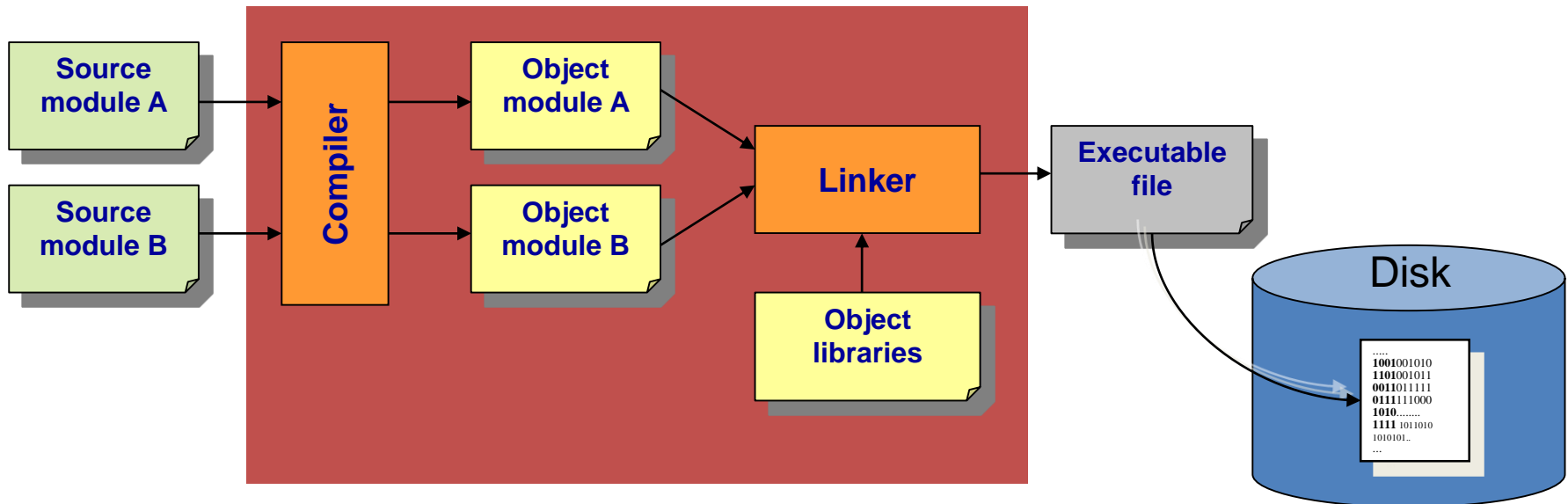
- **Sequential execution:** when the CPU is used by a single process from the beginning till the end of its execution
- **Concurrent execution:** when the CPU is used alternatively by several processes during their lifetime
  - The most direct benefit of concurrent execution is the increase in CPU utilization, since processes are not constantly demanding CPU because they alternate CPU with I/O bursts



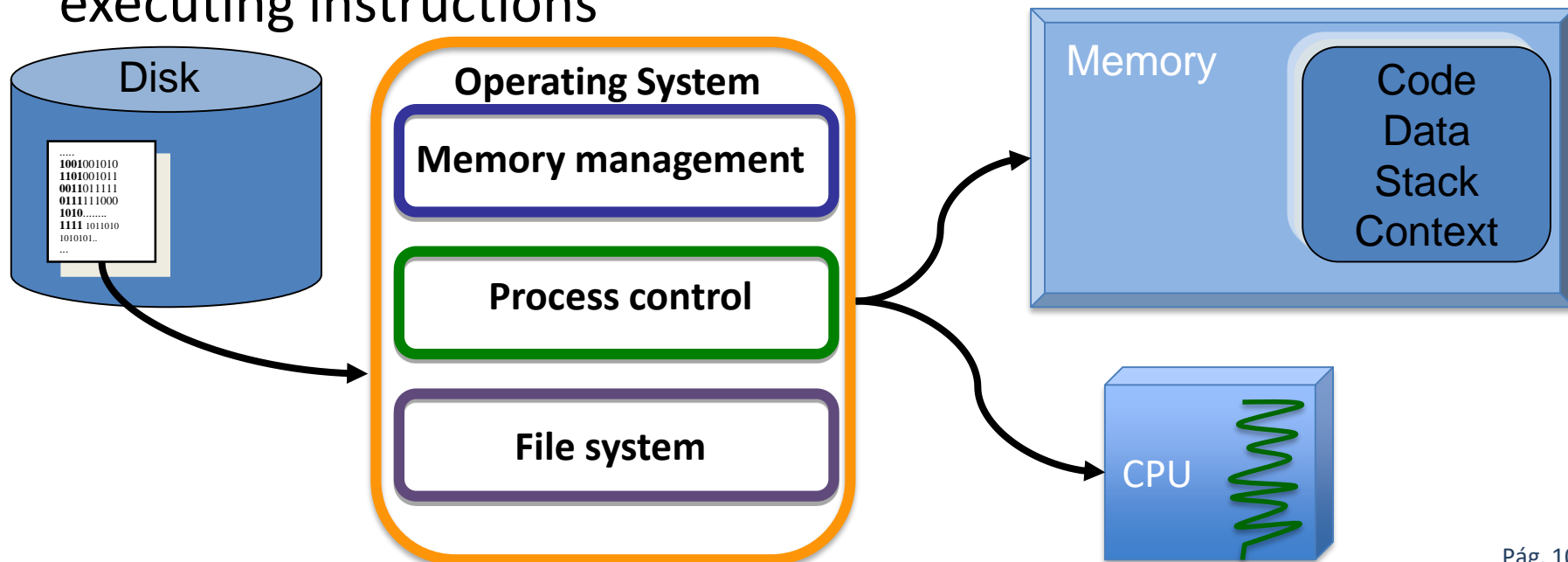
- Previous concepts
- **Executable files**
- Process concept
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- Exercises



- Steps to get an executable file:
  - The first step is **to compile** the **source code** to get **object code**
  - Later it is **linked** to system or other user **library code** -> this allows incorporating **external code**
  - The final result is an **executable file**

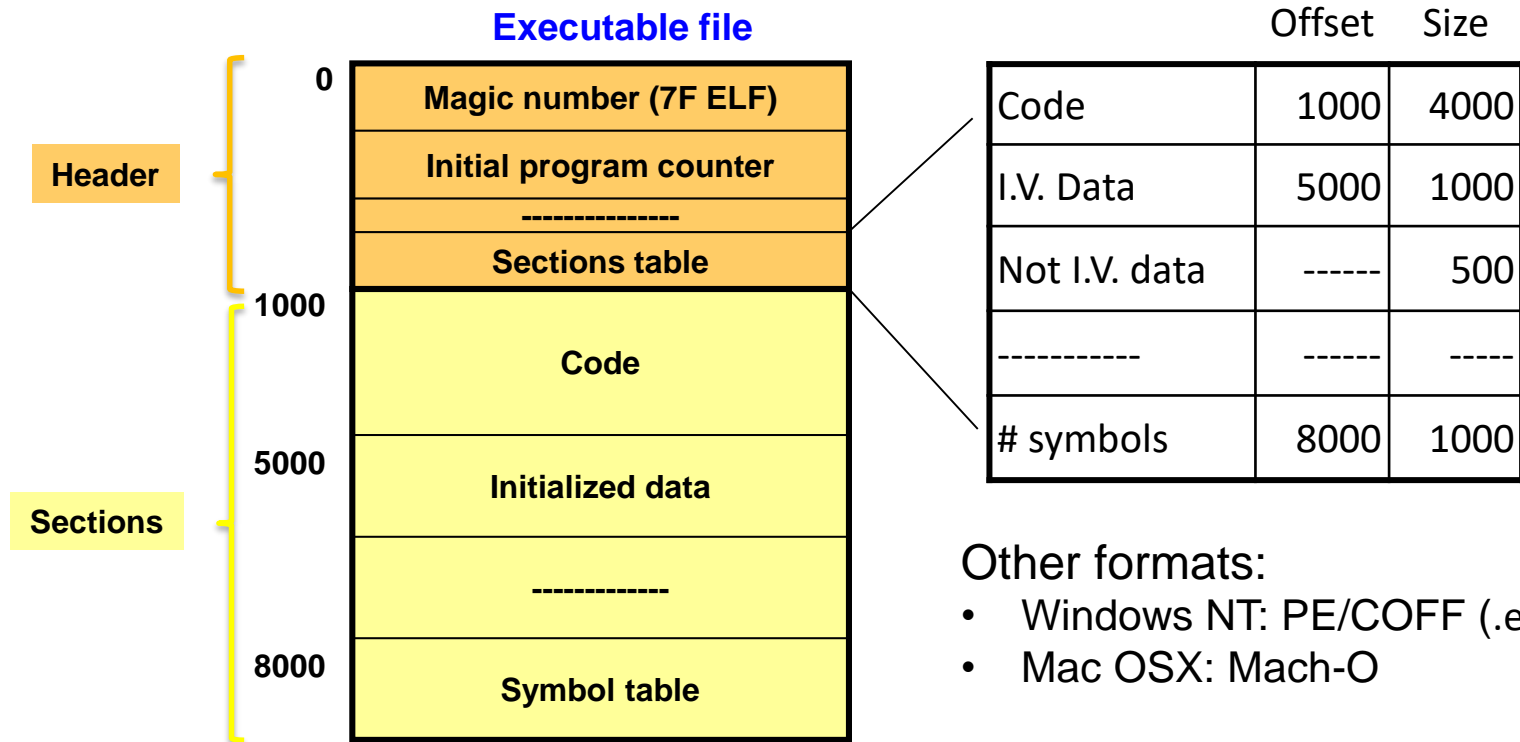


- An executable is a file whose structure is well known by the operating system because it contains:
  - The code to be executed
  - Initialized data
  - Library functions (static linking)
- With this information the OS can allocate the necessary resources for execution. Load data, code etc. and start executing instructions



- **Executable and Linking Format**

- File format for executables, object code, libraries and memory dumps
  - Widely used in Unix, Linux, Solaris and BSD
  - Extensions: .o, .so, .elf



Other formats:

- Windows NT: PE/COFF (.exe, .dll)
- Mac OSX: Mach-O

- Previous concepts
- Executable files
- **Process concept**
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- Exercises

- In order to define a process there are three aspects to be considered:
  - **Attributes** or features that define a process and allow to manage them
  - **Behaviour**, as being an active entity it will move between states so these states and their transitions have to be defined
  - **Operations** that can be done with processes

- **Process attributes**

Resources and features owned by a process and kept by the OS

- **Identity**

- Process identifier (PID)

- **Runtime environment**

- Working directory
- Opened file descriptors

- **State**

- Process state
- Machine context (program counter, stack pointer, general purpose register values)
- Etc.

- **Memory**

- Memory addresses for code, data and stack

- **Scheduling**

- CPU consumption time
- Priority

- **Monitoring**

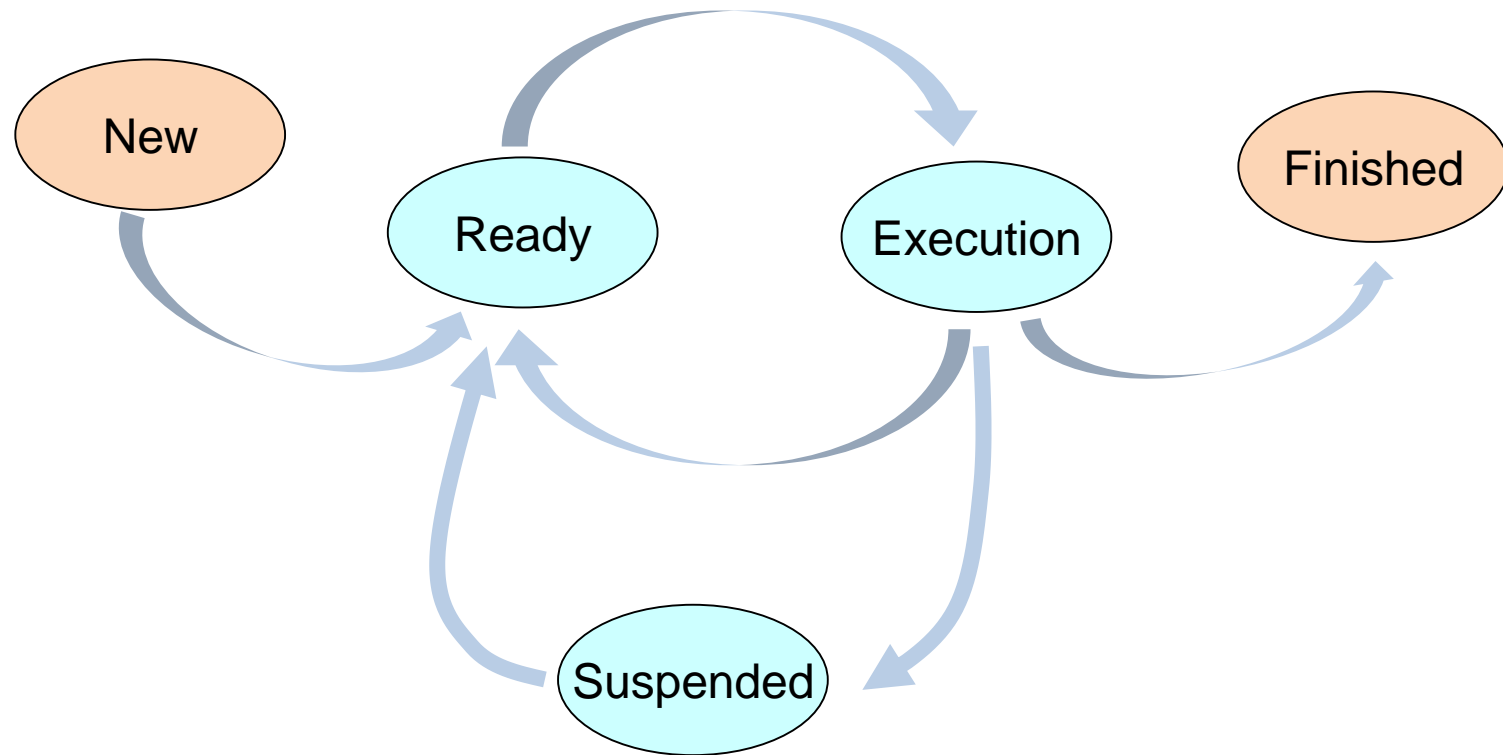
- **Operations on processes**

- As happens with the attributes, the number and the type of operations that can be done on processes **depend on the OS considered**
- Anyway, the following operations are always supported:
  - Creation
  - Communication
  - Waiting
  - Resource access
  - Ending

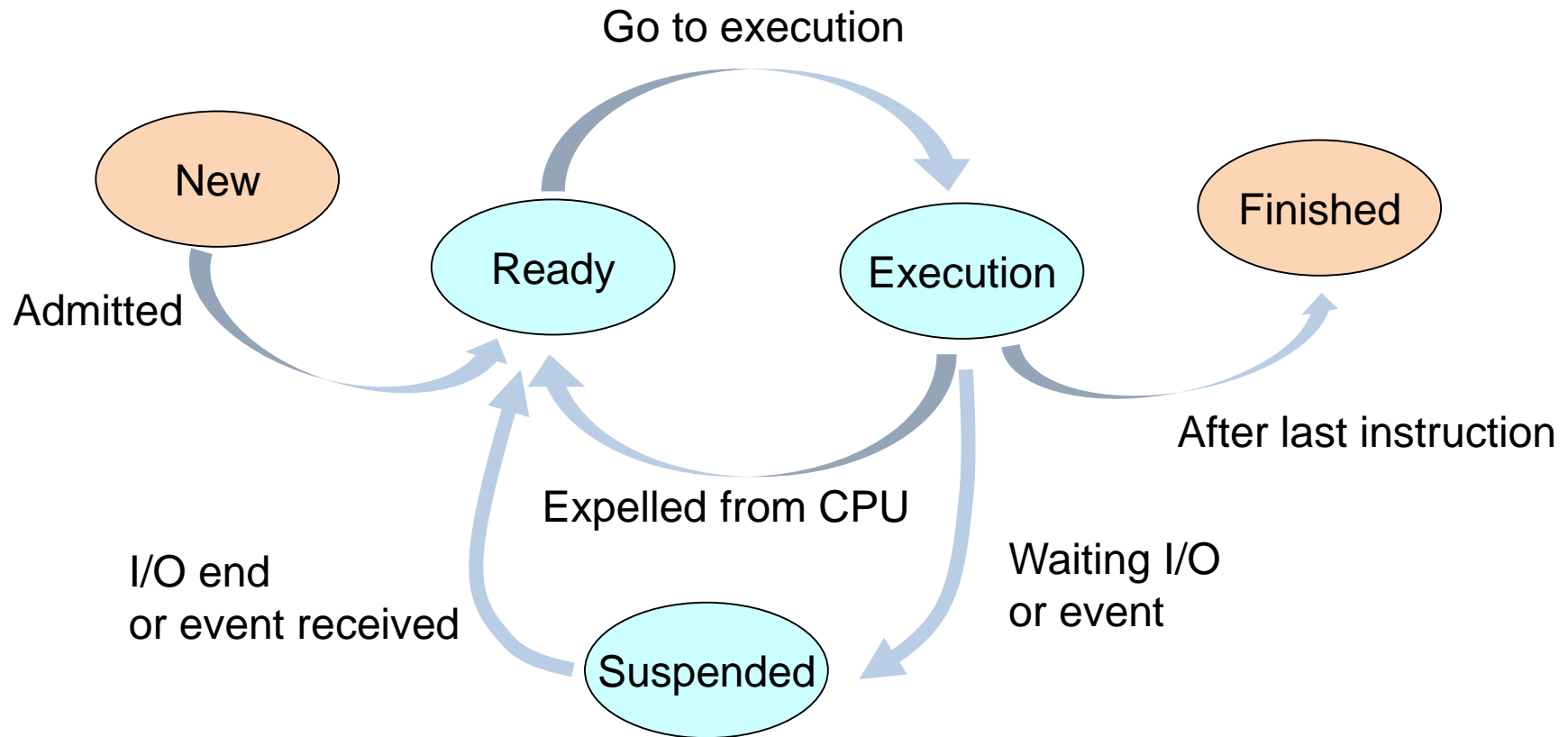
- Previous concepts
- Executable files
- Process concept
- **Process states**
- Process implementation: PCB
- System calls table for processes and signals
- Exercises



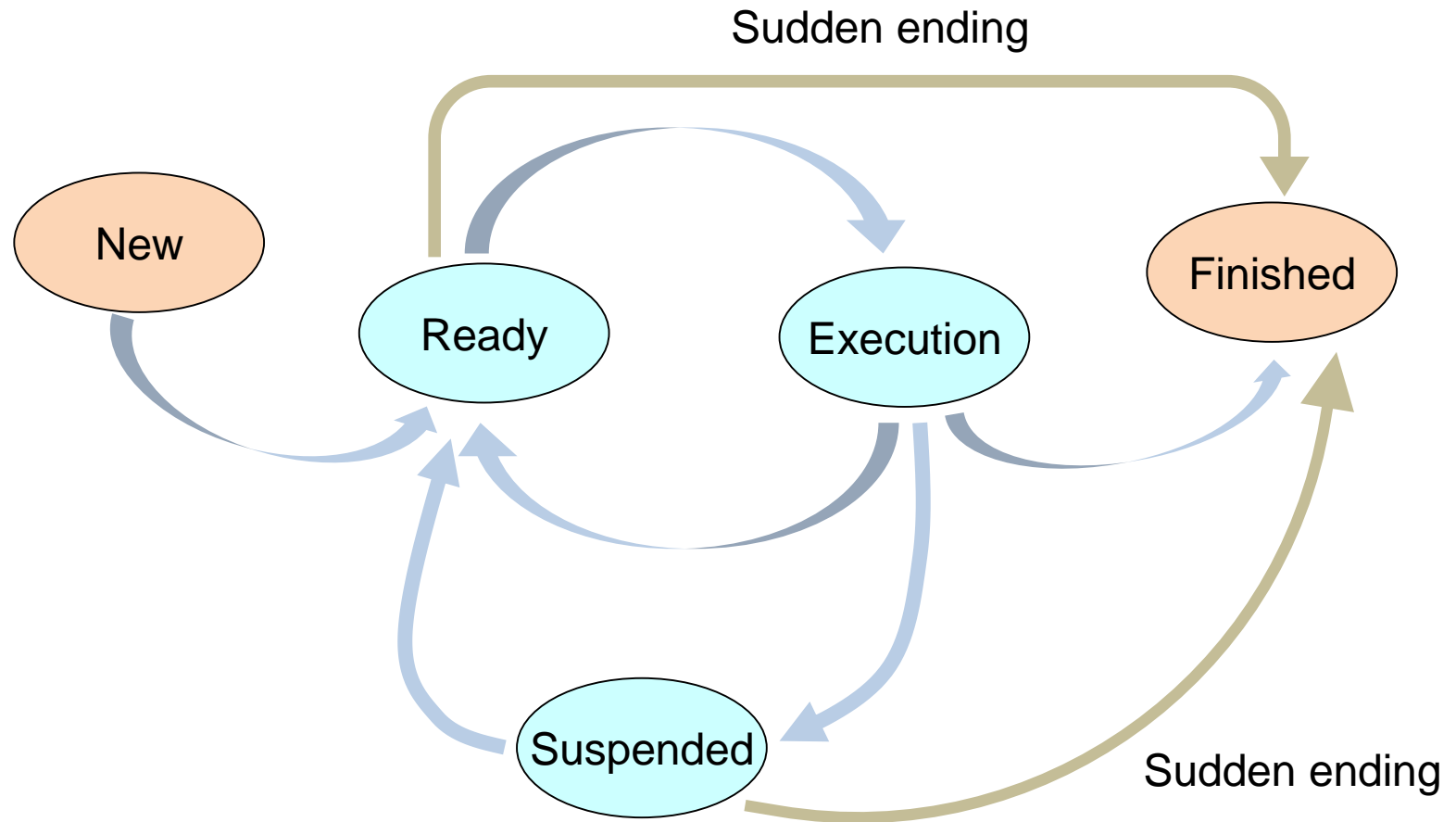
- **Processes** are **active entities** so they evolve during his lifetime through a **set of states**

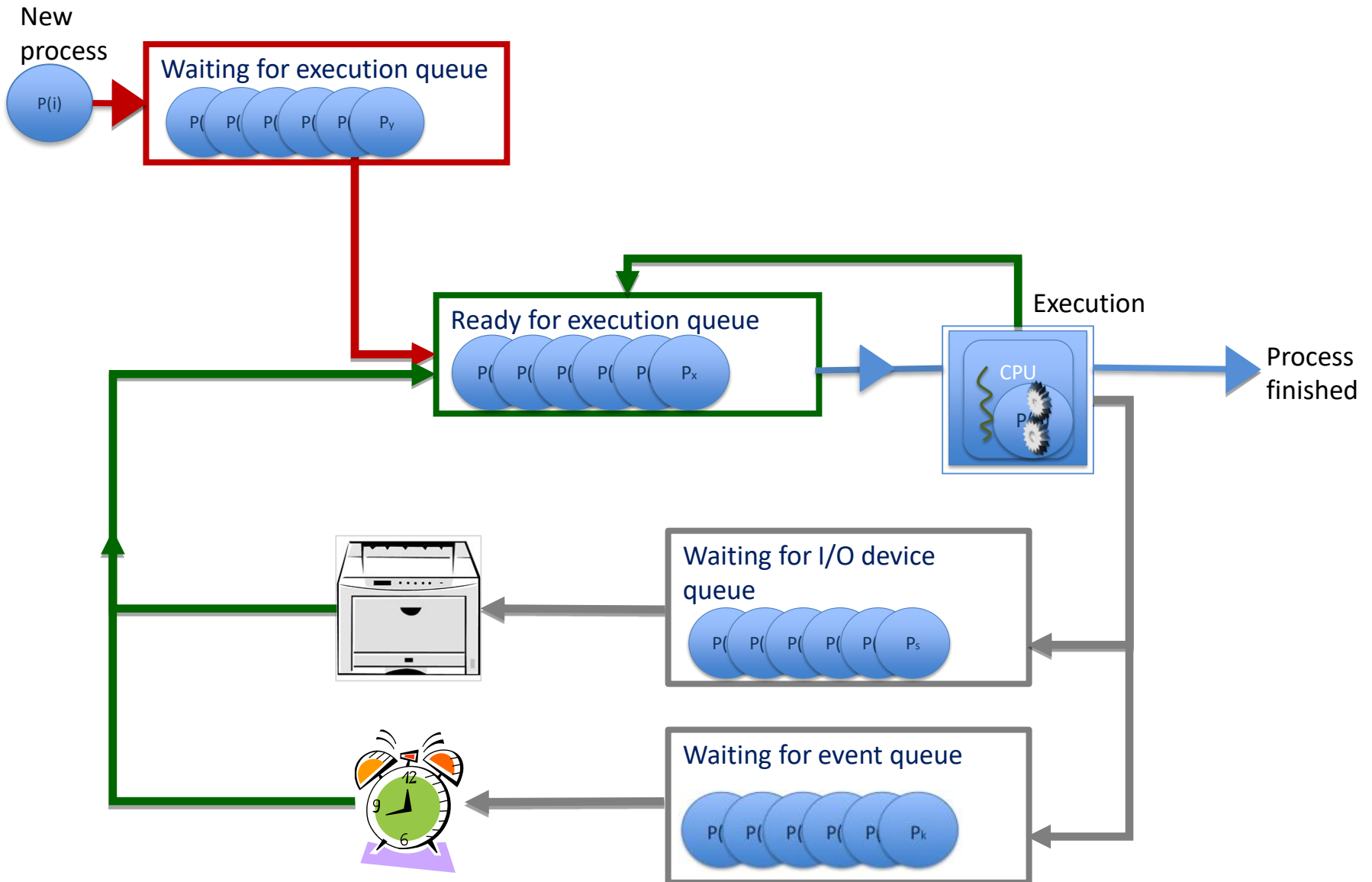


- **Common process state transitions**



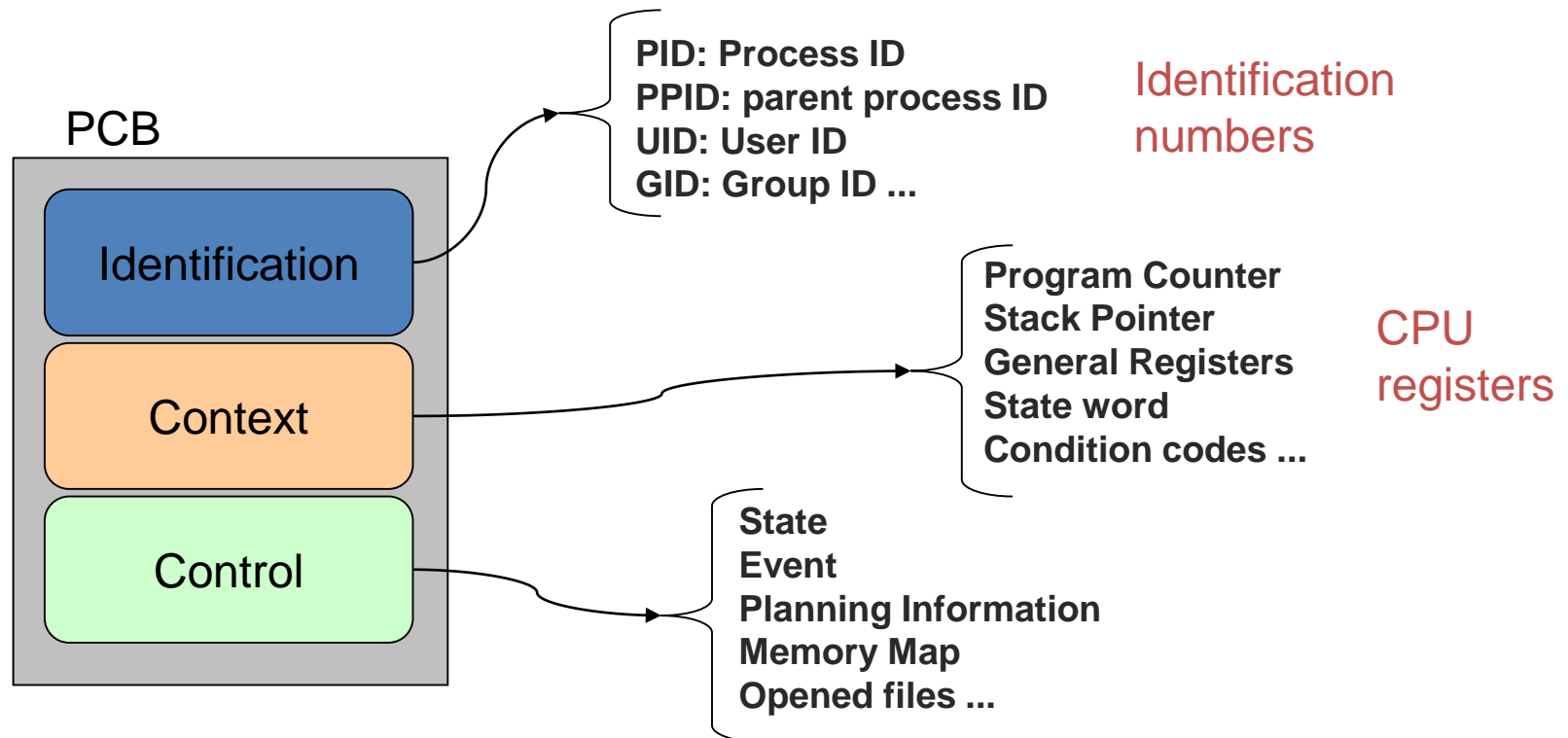
- Transitions due to anomalous behavior

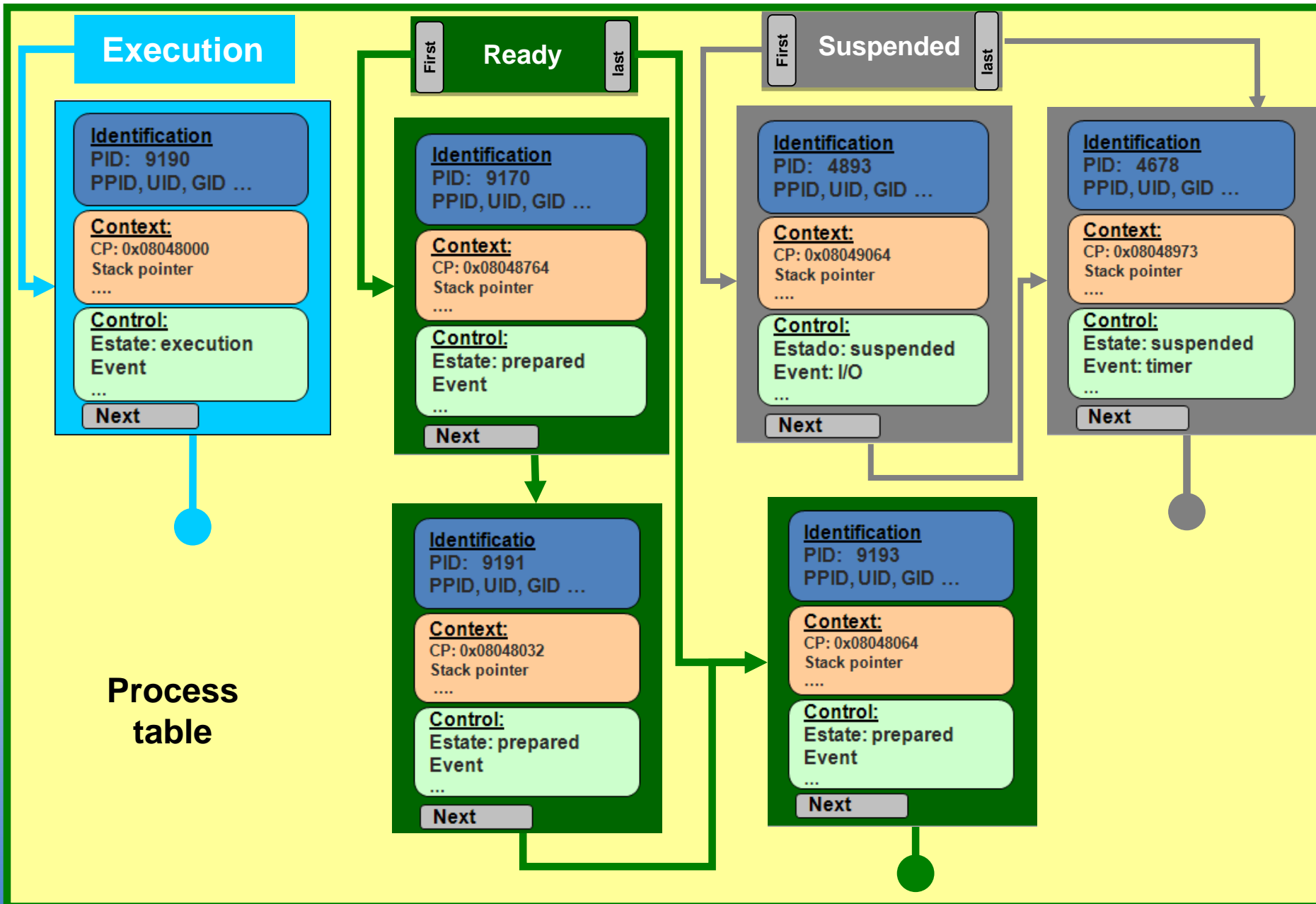




- Previous concepts
- Executable files
- Process concept
- Process states
- **Process implementation: PCB**
- System calls table for processes and signals
- Exercises

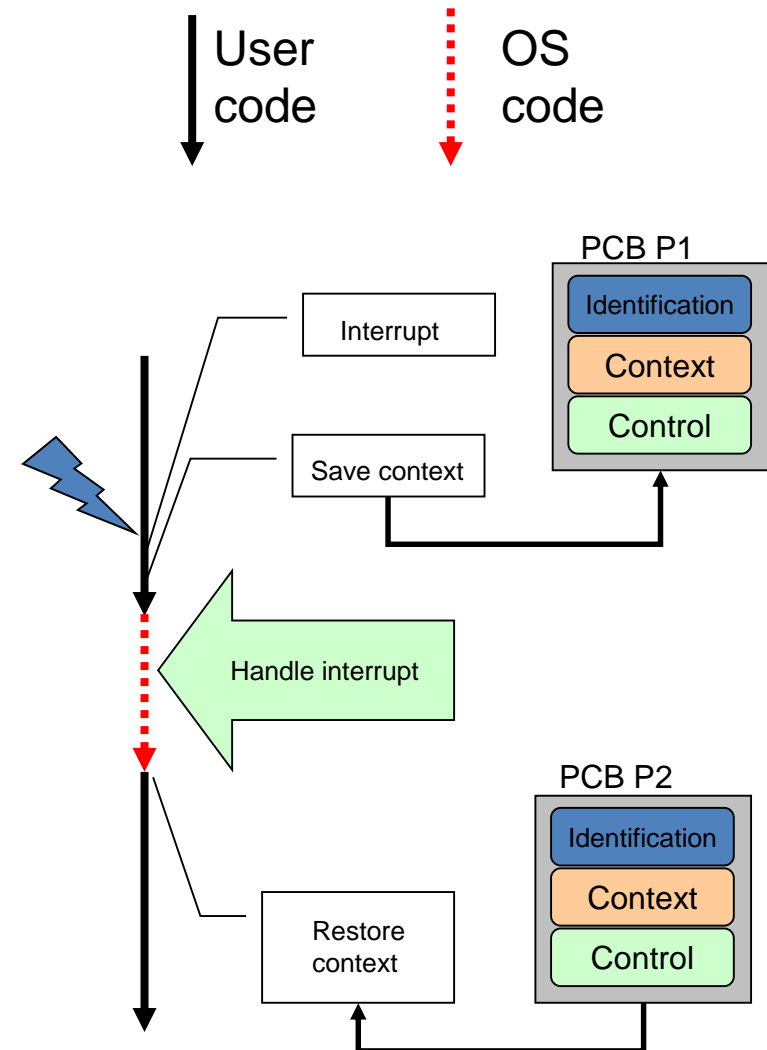
- A PCB (Process Context Block) is the **data structure** which supports the concept (abstraction) process
  - An OS is also a program, based on using **algorithms** and **data structures**
- PCB keeps process relevant information which changes during process lifetime -> each operating system has its own structure





- **Context switch**

- Mechanism that allows an OS to suspend the execution of the current process in order to start or resume the execution of another process.
  - This process is activated by an interrupt (i.e. clock interrupt)
- What is done?
  1. The state information relevant to the running process (context) is saved
  2. The scheduler updates PCBs and queues
  3. The new process takes the CPU





- Previous concepts
- Executable files
- Process concept
- Process state
- Process implementation: PCB
- **System calls table for processes and signals**
- Exercises

# System calls table for processes and signals fso

	Processes
<b>fork</b>	Create a child process
<b>exit</b>	End process
<b>wait</b>	Wait for a process ending
<b>exec</b>	Execute a program
<b>getpid</b>	Get process attributes

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

- Previous concepts
- Executable files
- Process concept
- Process states
- Process implementation: PCB
- System calls table for processes and signals
- **Exercises**

- In the following actions list, indicate from each one if it is shell code or OS code the one that performs it. Mark one, none or both options with a cross:

OS	Shell	
		Reading the command line and parse it
		Programming a device controller
		Providing a system calls interface
		Selecccting a process to assign it the CPU
		Performing a system call
		Providing a confortable user interface

- Which state (new, ready, execution, suspended, finished) corresponds to every one of the following processes:

Process		State
P1	The CPU is executing instructions belonging to P1	
P2	P2 has asked for a disk access, but the disk is busy serving to process P3	
P3	P3 is doing disk access	
P4	P4 belongs to a user that has finished all his/her jobs in a terminal and is logging out the session	
P5	P5 has a process identifier assigned and only its control tables are already created	
P6	P6 has all its SO control structures and its memory image stored in main memory	

- In the following command line:

```
$ cat f1 f2 f3 | grep start | wc -l >tracd
```

Indicate:

- How many processes will be created during its execution in a UNIX system
- What files has associated every command

- Given the following code:

```
#include <stdio.h>
#include <sys/types.h>

int main(void)
{
    pid_t pid;
    int i;

    for (i=0; i<2; i++)
        pid=fork();
    return 0;
}
```

How many processes will be created along its execution?