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

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Goals and bibliography

- **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

Previous concepts

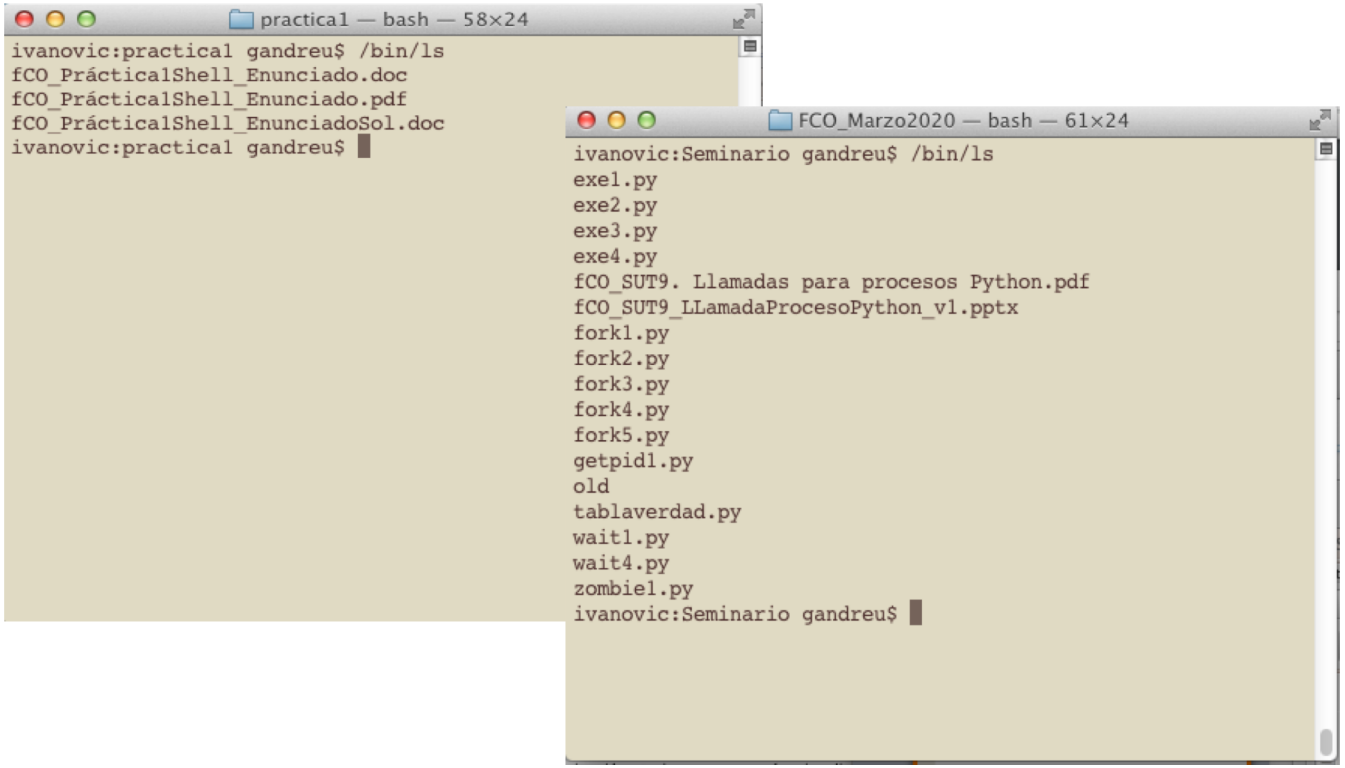
- In order to understand the process concept some terms have to be clarified, particularly the differences between the following pairs of concepts:
  - Program / Process
  - Sequential execution / Concurrent execucion

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

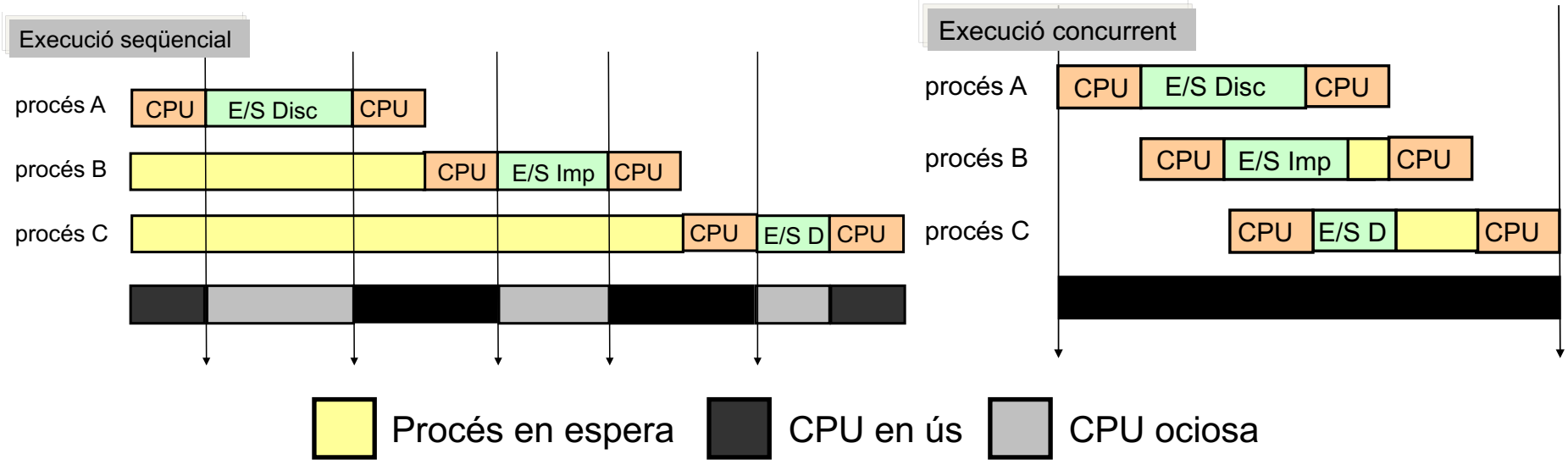
- **Program versus Process**
  - The same program can be run multiple times concurrently



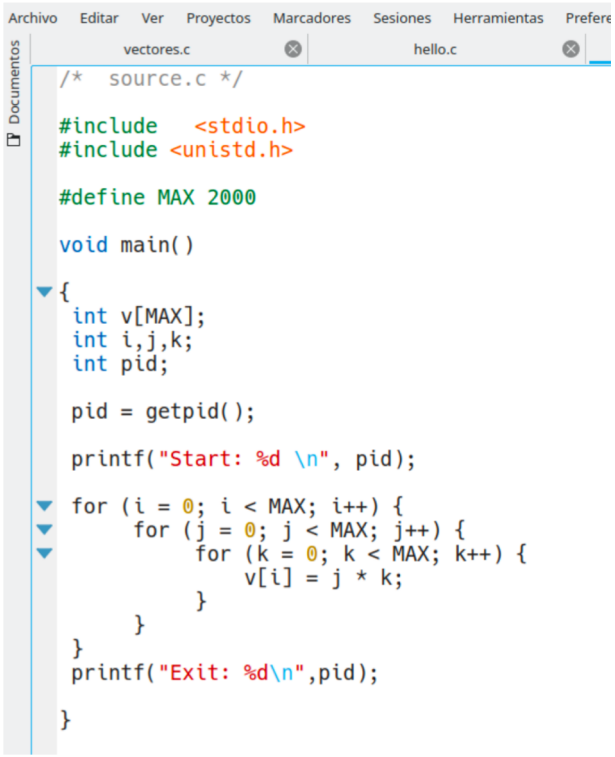
-The file with executable code /bin /ls is always the same

-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



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




```
usuario@lgiifso:~/examples$ gcc -o tc timeConsuming.c
usuario@lgiifso:~/examples$ time ./tc
Start: 5091
Exit: 5091

real    0m8,528s
user    0m8,457s
sys     0m0,000s
usuario@lgiifso:~/examples$
```

```
usuario@lgiifso:~/examples$ time ./tc ; time ./tc ; time ./tc
Start: 5117
Exit: 5117

real    0m8,267s
user    0m8,208s
sys     0m0,001s
Start: 5118
Exit: 5118

real    0m8,214s
user    0m8,170s
sys     0m0,006s
Start: 5119
Exit: 5119

real    0m8,216s
user    0m8,178s
sys     0m0,000s
usuario@lgiifso:~/examples$
```

Previous concepts: example

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

Archivo Editar Ver Proyectos Marcadores Sesiones Herramientas Prefere

Documentos

vectores.c hello.c

```c
/*  source.c */

#include   <stdio.h>
#include <unistd.h>

#define MAX 2000

void main()

{
 int v[MAX];
 int i,j,k;
 int pid;

 pid = getpid();

 printf("Start: %d \n", pid);

 for (i = 0; i < MAX; i++) {
      for (j = 0; j < MAX; j++) {
           for (k = 0; k < MAX; k++) {
               v[i] = j * k;
           }
      }
 }
 printf("Exit: %d\n",pid);

}
```

```
usuario@lgiifso:~/examples$ gcc -o tc timeConsuming.c
usuario@lgiifso:~/examples$ time ./tc
Start: 5091
Exit: 5091

real    0m8,528s
user    0m8,457s
sys     0m0,000s
usuario@lgiifso:~/examples$
```

```
usuario@lgiifso:~/examples$ time ./tc & time ./tc & time ./tc
[2] 5104
[3] 5105
Start: 5106
Start: 5107
Start: 5108
Exit: 5106

real    0m25,871s
user    0m8,465s
sys     0m0,000s
usuario@lgiifso:~/examples$ Exit: 5108

real    0m25,943s
user    0m8,497s
sys     0m0,000s
Exit: 5107

real    0m25,948s
user    0m8,503s
sys     0m0,000s
```

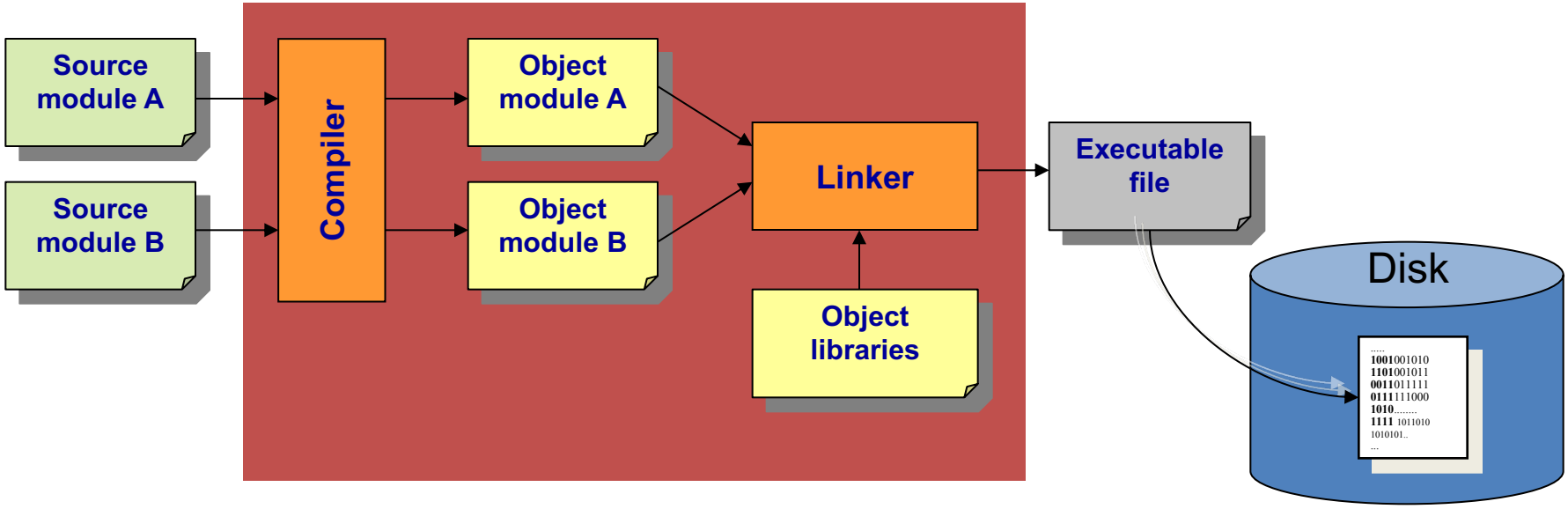
```
usuario@lgiifso:~/examples$ ps aux | grep ./tc
usuario   5106 29.6  0.0   4504   856 pts/1    R+   16:50   0:01 ./tc
usuario   5107 29.4  0.0   4504   748 pts/1    R    16:50   0:01 ./tc
usuario   5108 29.4  0.0   4504   736 pts/1    R    16:50   0:01 ./tc
usuario   5110  0.0  0.0  16944  1088 pts/2    S+   16:50   0:00 grep --color=aut
 ./tc
usuario@lgiifso:~/examples$
```

Content

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

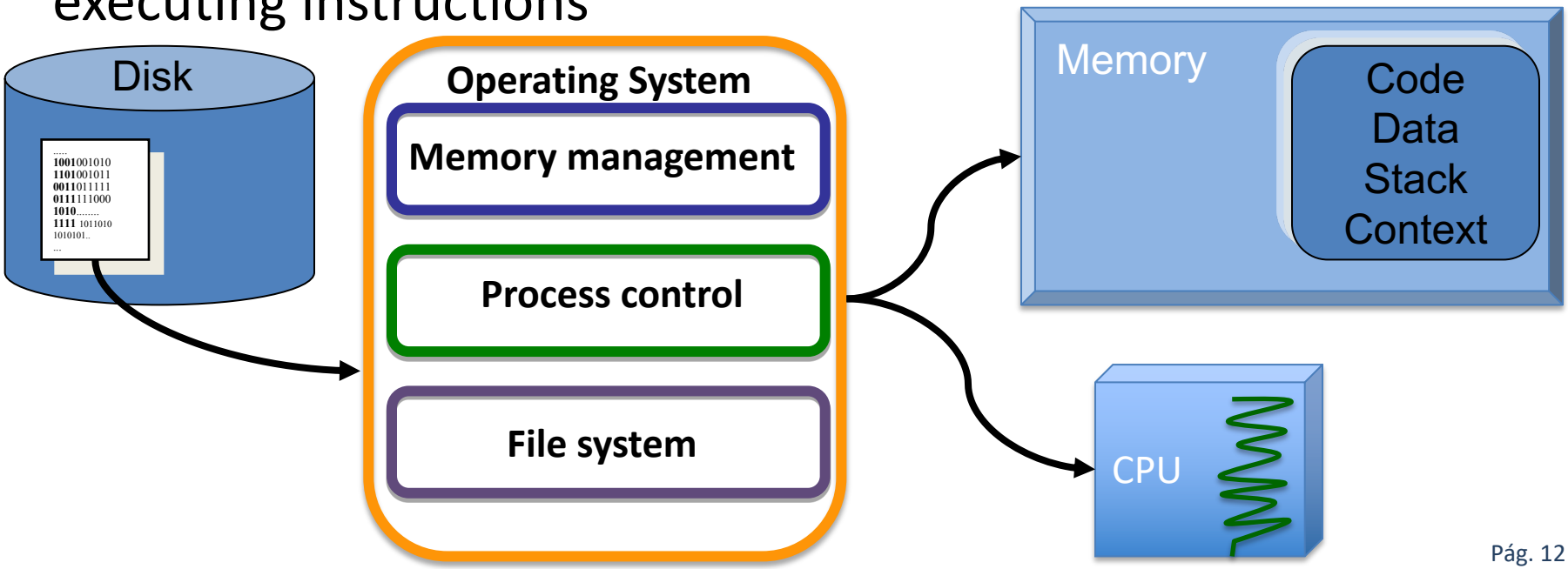
Executable files

- 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**



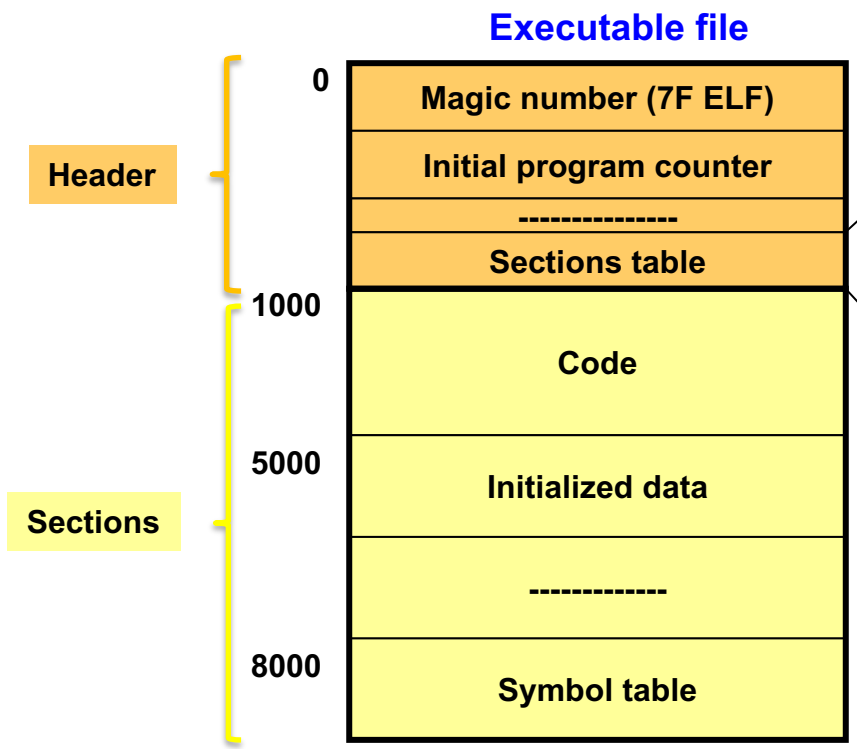
Executable files

- 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



The executable ELF

- **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



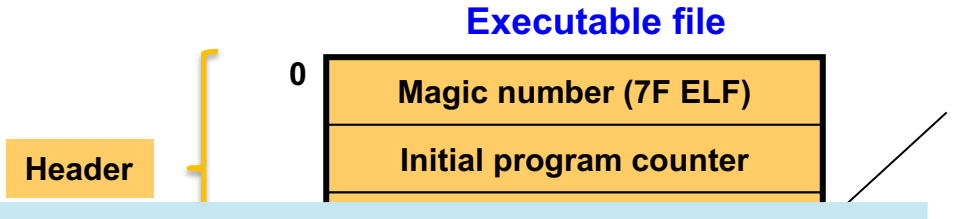
	Offset	Size
Code	1000	4000
I.V. Data	5000	1000
Not I.V. data	------	500
----------	------	-----
# symbols	8000	1000

Other formats:

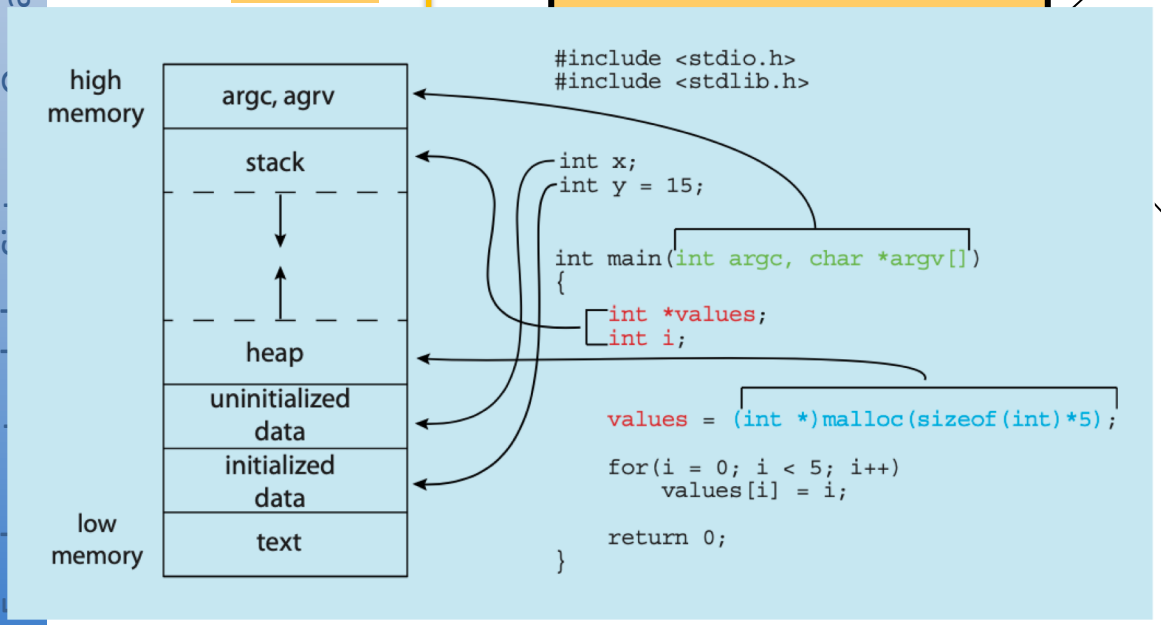
- Windows NT: PE/COFF (.exe, .dll)
- Mac OSX: Mach-O

The executable ELF

- **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



	Offset	Size
Code	1000	4000
I.V. Data	5000	1000
Not I.V. data	------	500
----------	------	-----
# symbols	8000	1000



Other formats:

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

Content

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

Process concept

- In order to define a process there are three aspects to be considered:
  - **Atributtes** 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**

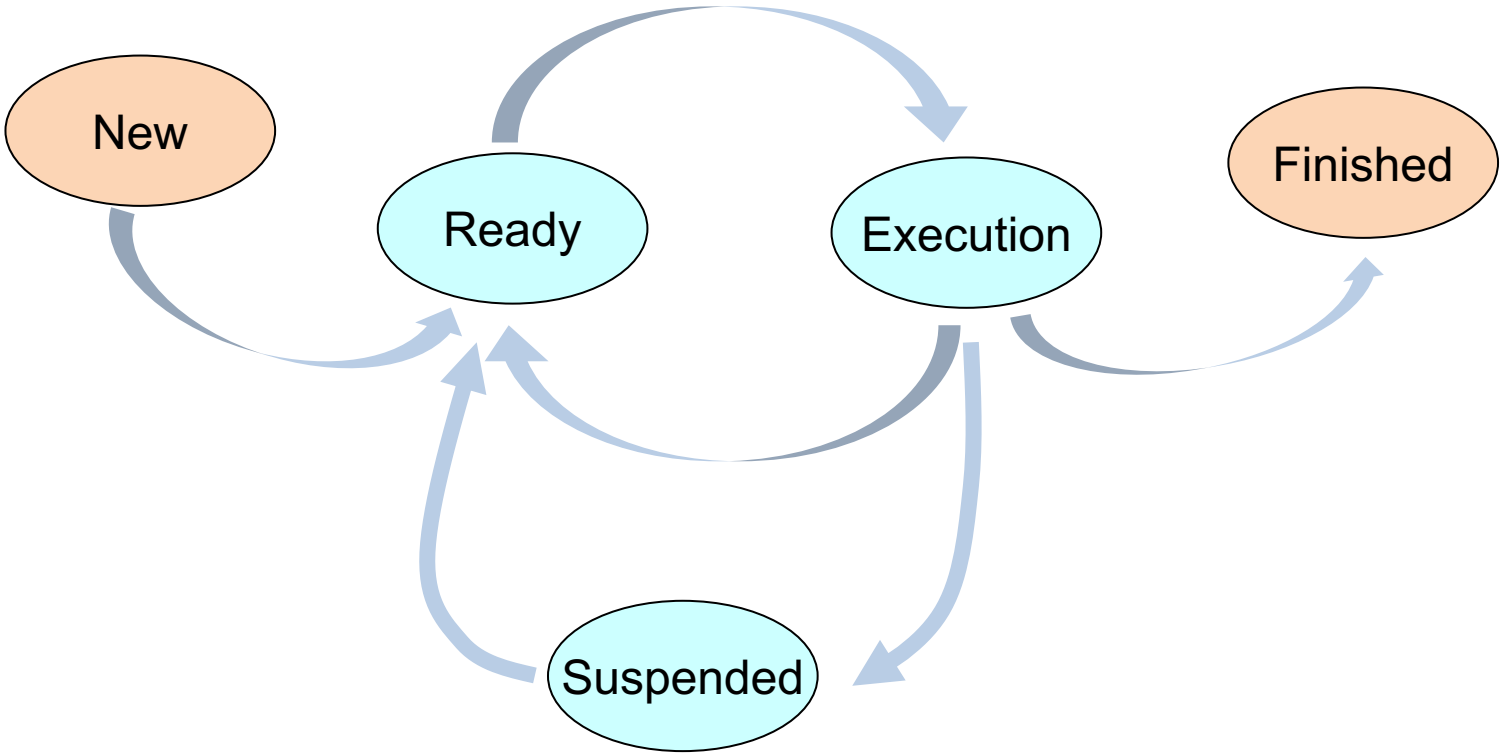
- **Operations on processes**
  - As happens with the attributes, the number and the type of operations that can be done on processes **depend on the OS considered**
  - Anyway, the following operations are always supported:
    - Creation
    - Communication
    - Waiting
    - Resource access
    -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

Content

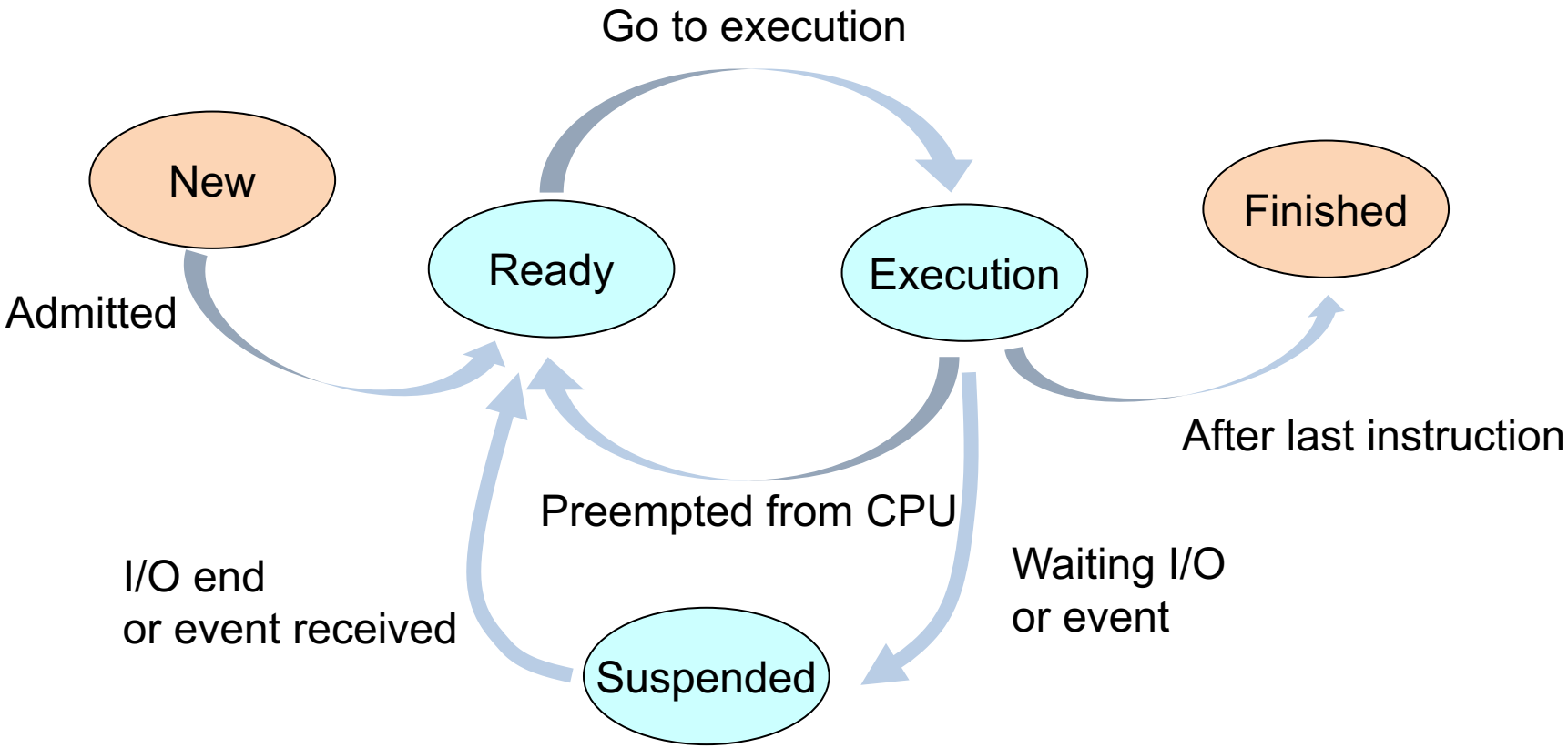
- Previous concepts
- Executable files
- Process concept
- **Process states**
- Process implementation: PCB
- System calls table for processes and signals
- Exercises

Process states

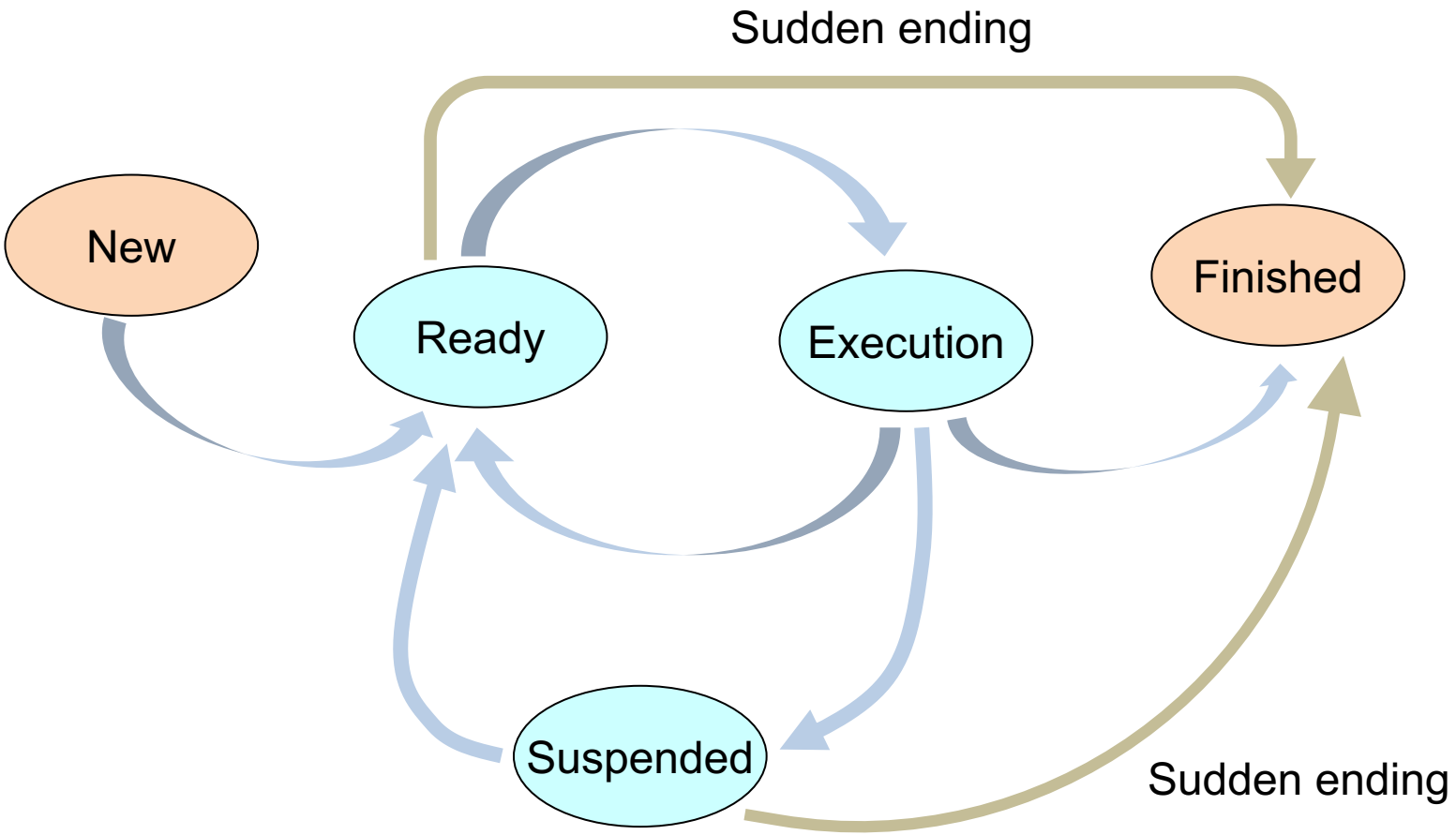
- **Processes** are **active entities** so they evolve during his lifetime through a **set of states**



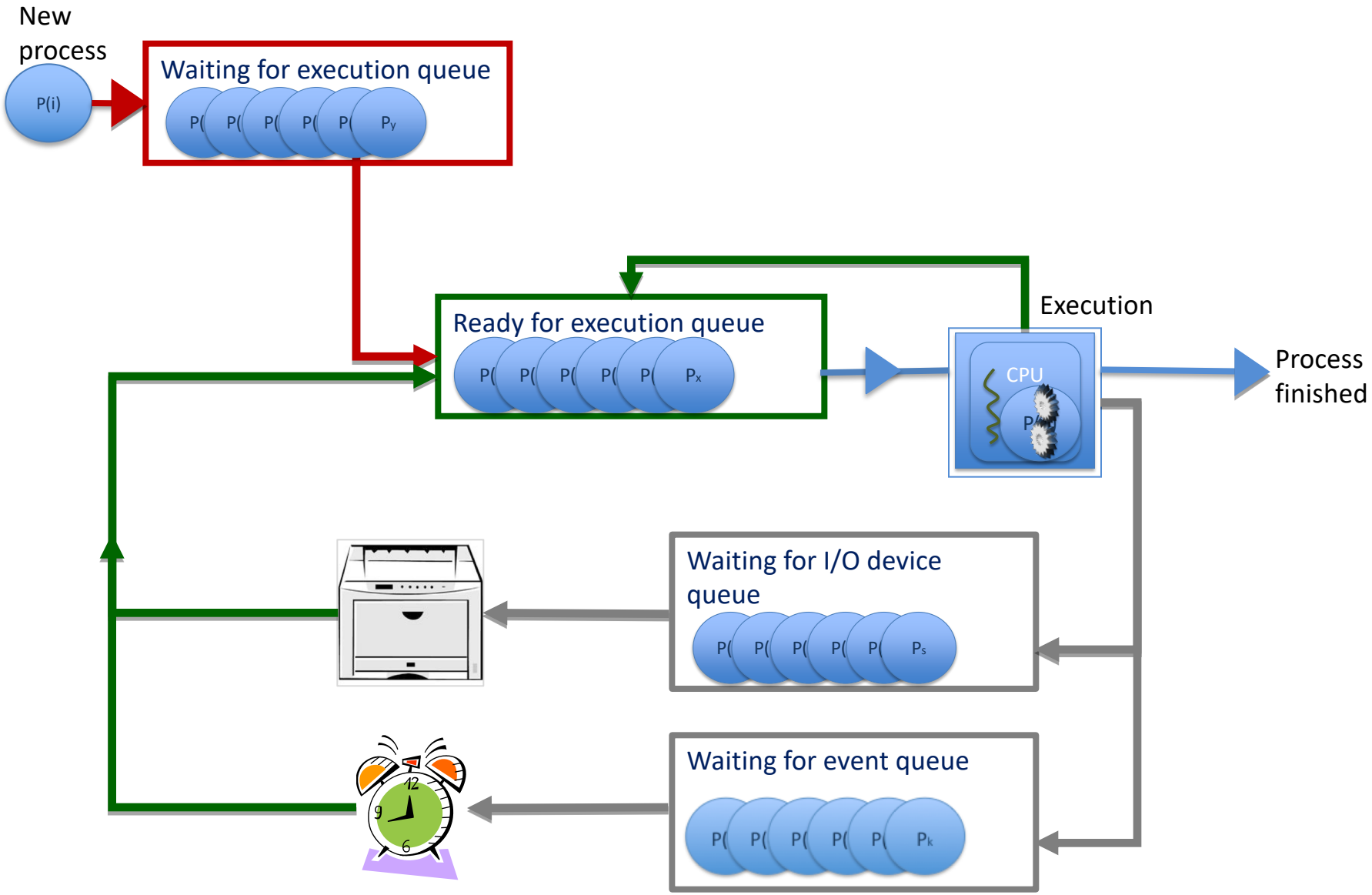
- **Common process state transitions**



- **Transitions due to anomalous behavior**



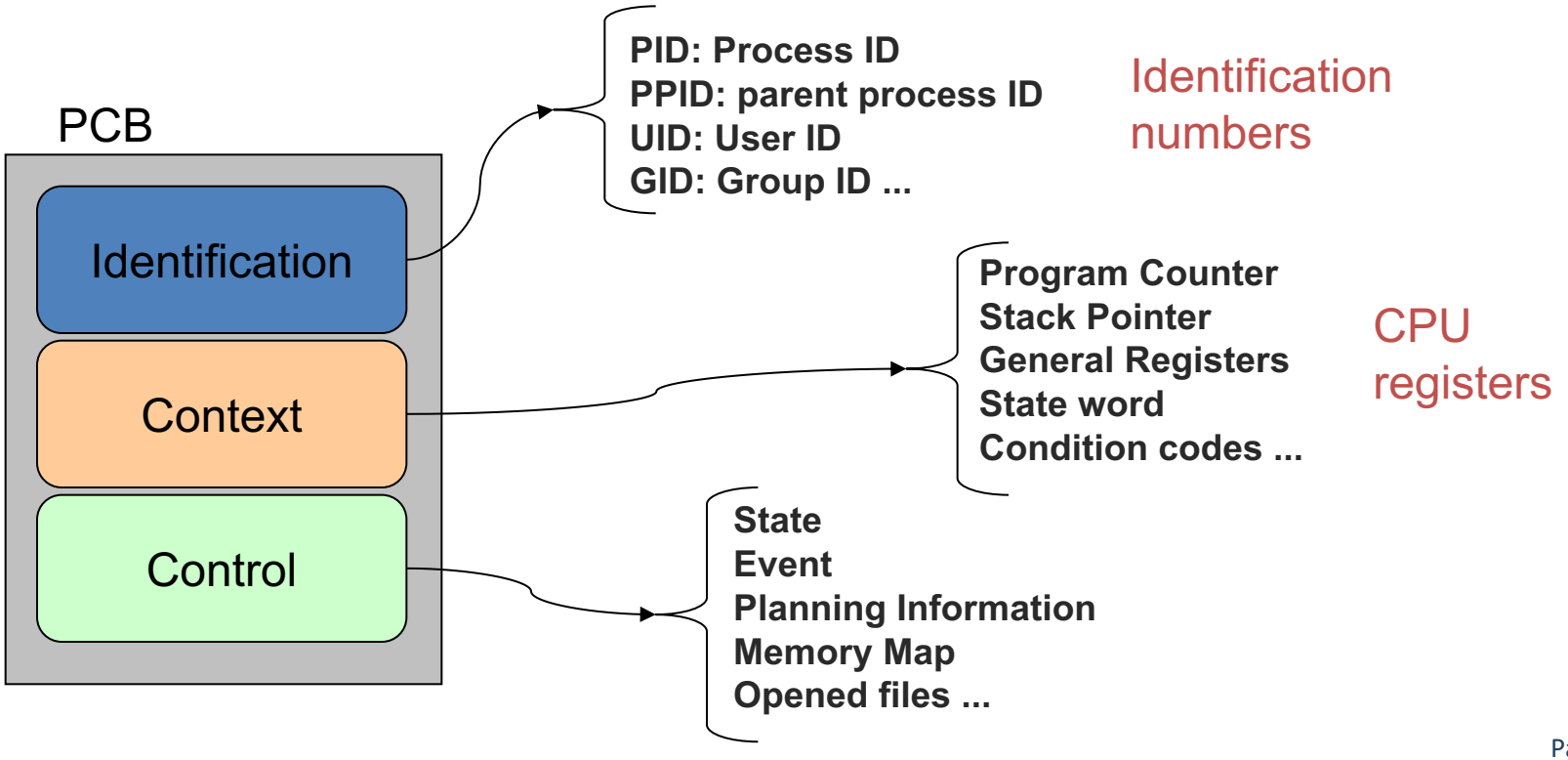
Process states



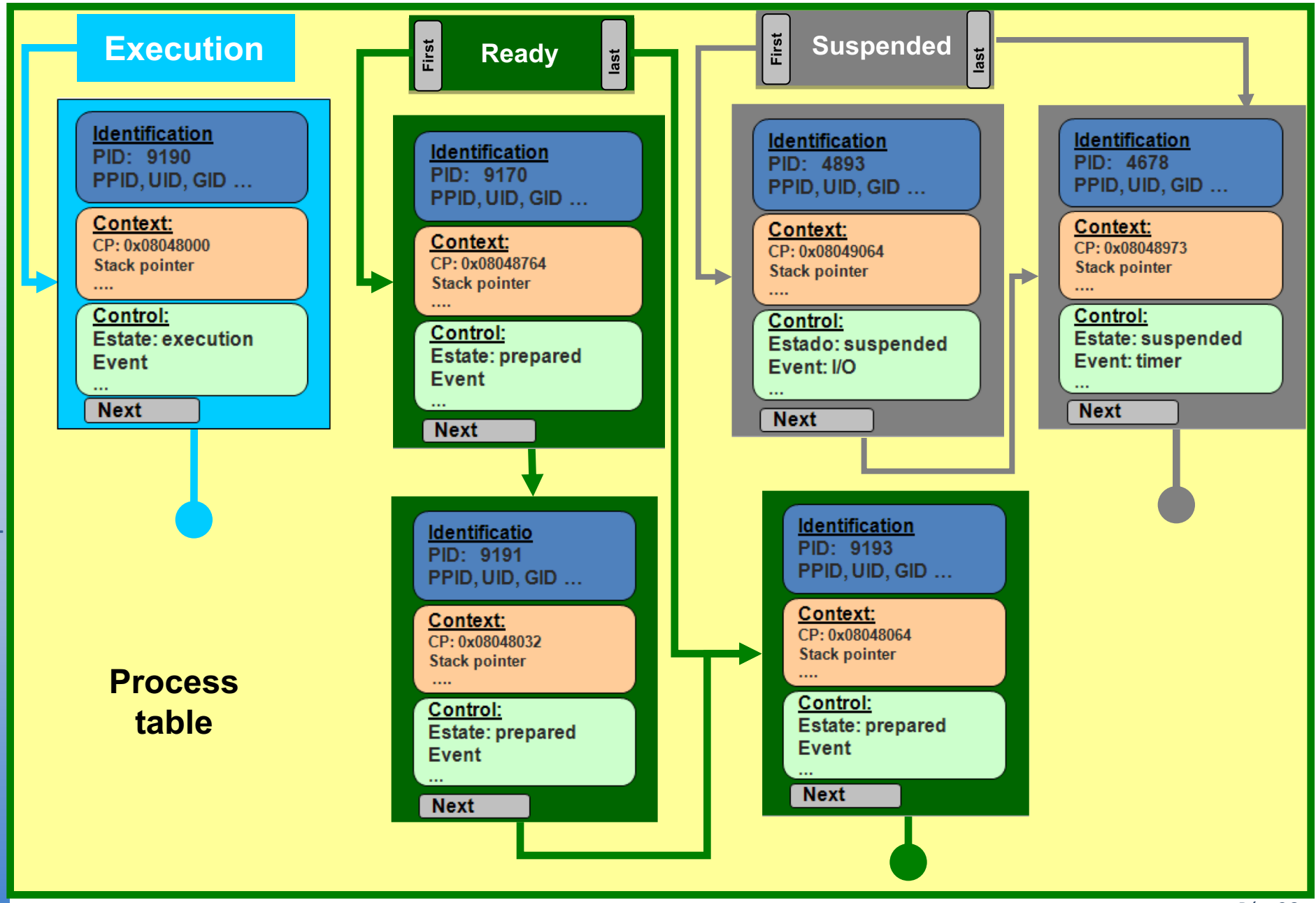
- Previous concepts
- Executable files
- Process concept
- Process states
- **Process implementation: PCB**
- System calls table for processes and signals
- Exercises

Process implementation : PCB

- 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

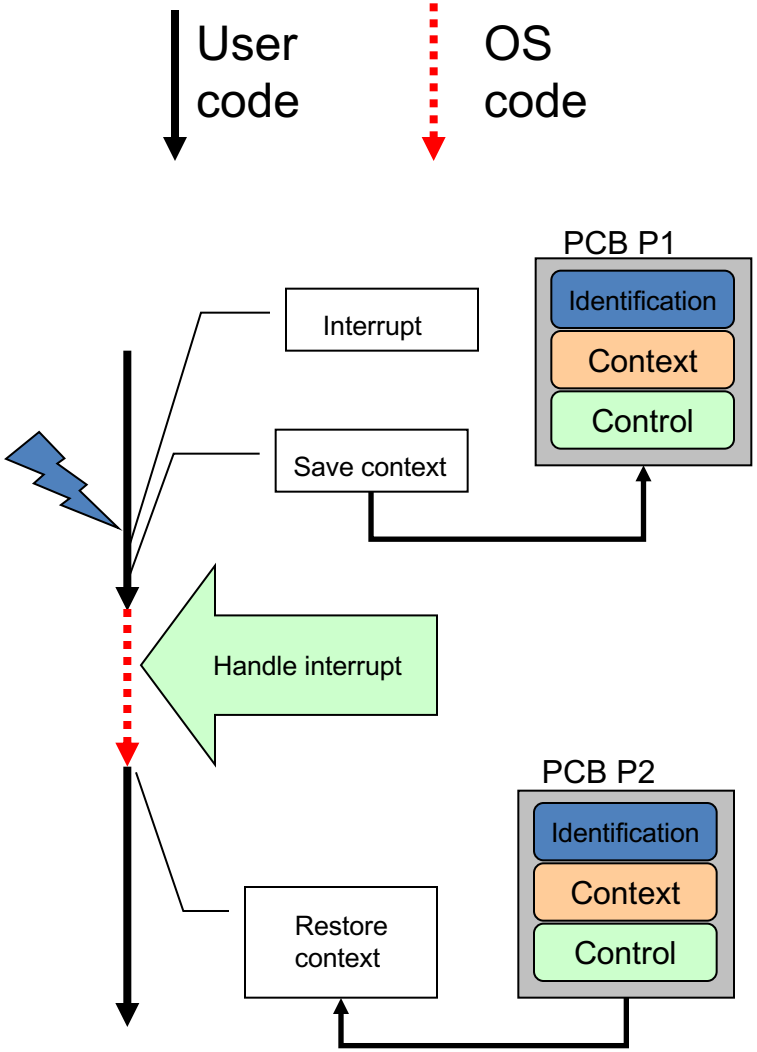


Process implementation : PCB

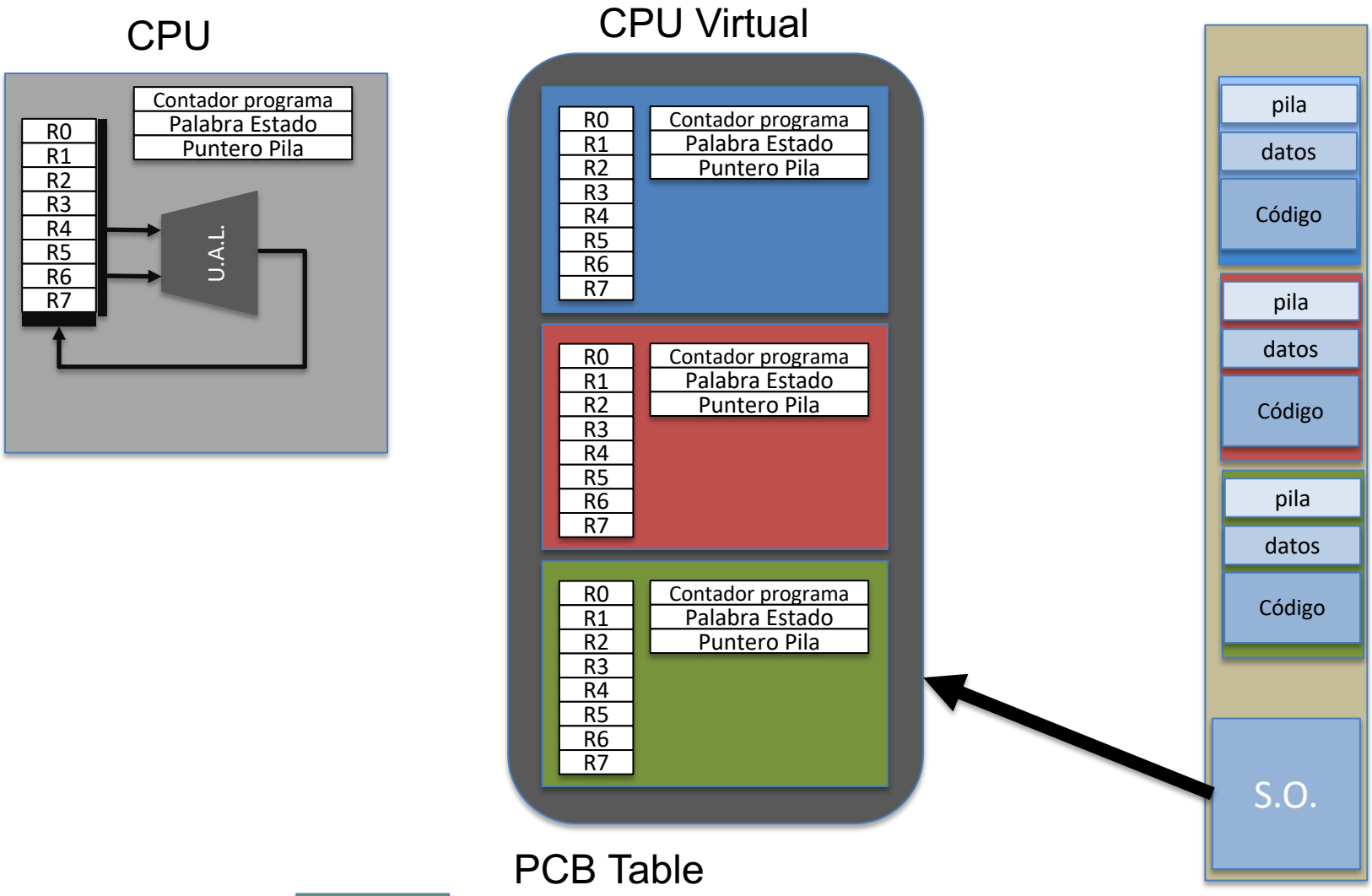


Process implementation: PCB

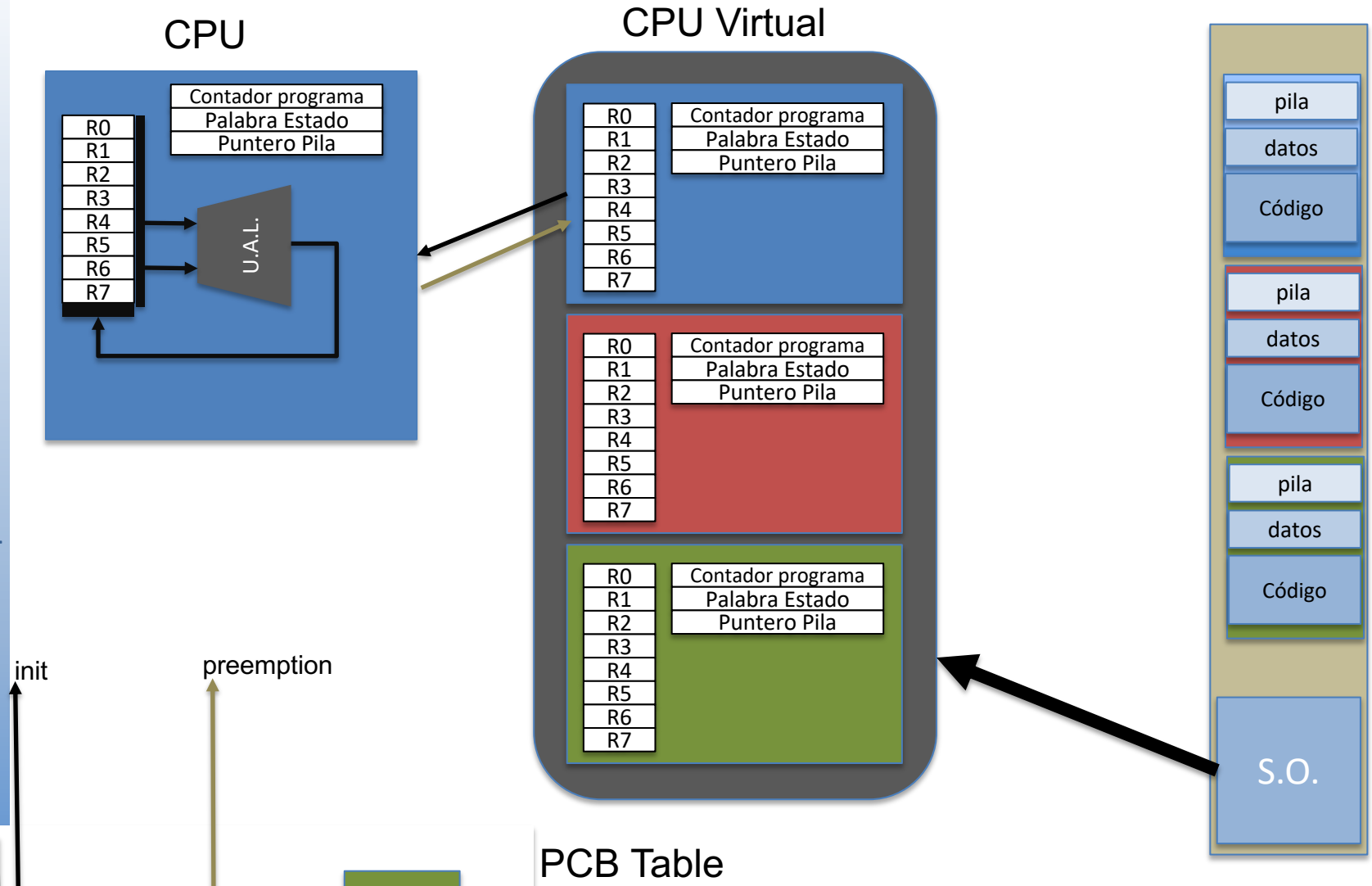
- **Context switch**
  - Mechanism that allows an OS to suspend the execution of the current process in order to start or resume the execution of another process.
    - This process is activated by an interrupt (i.e. clock interrupt)
  - What is done?
    1. The state information relevant to the running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



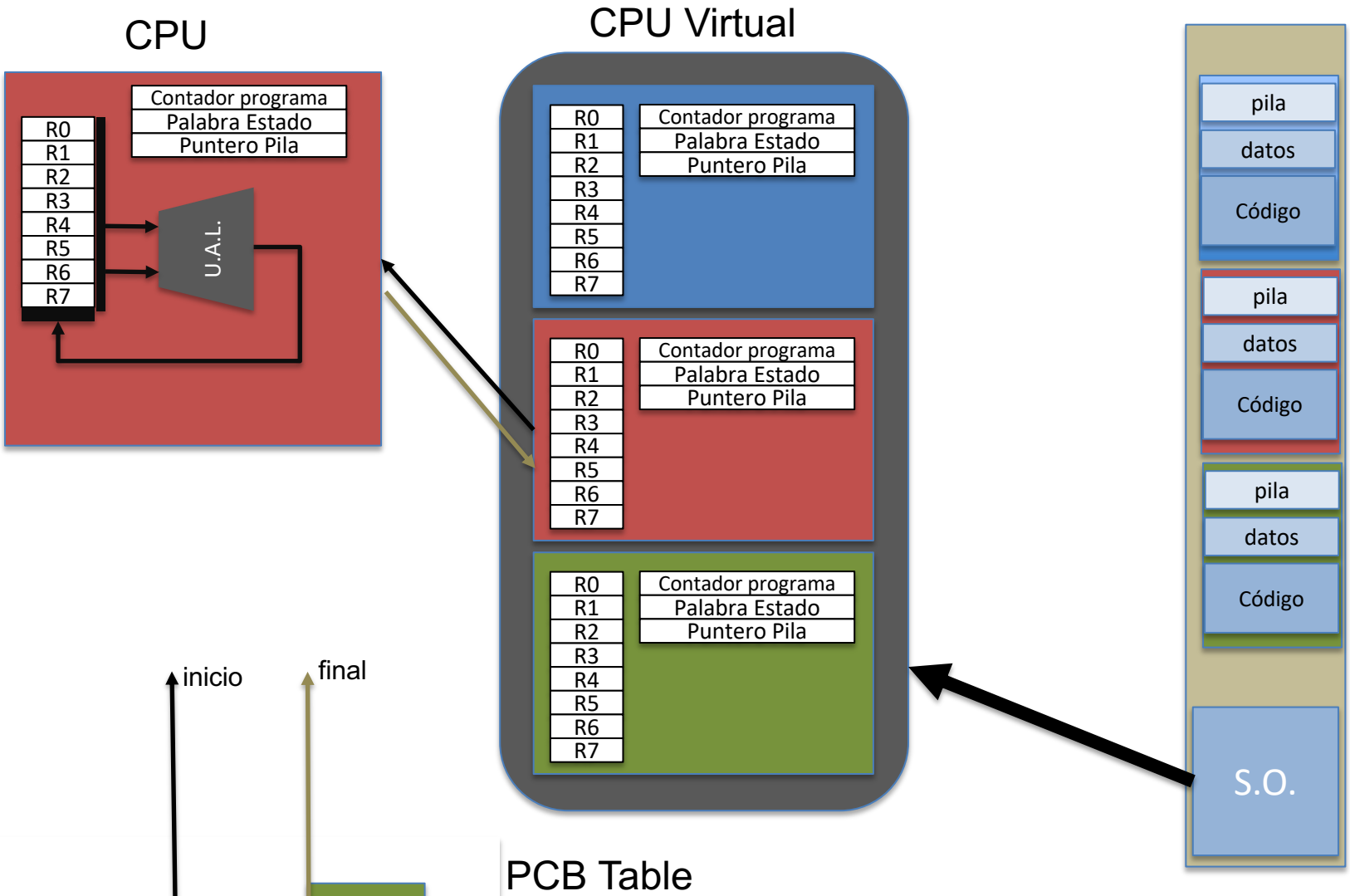
Concurrent execution



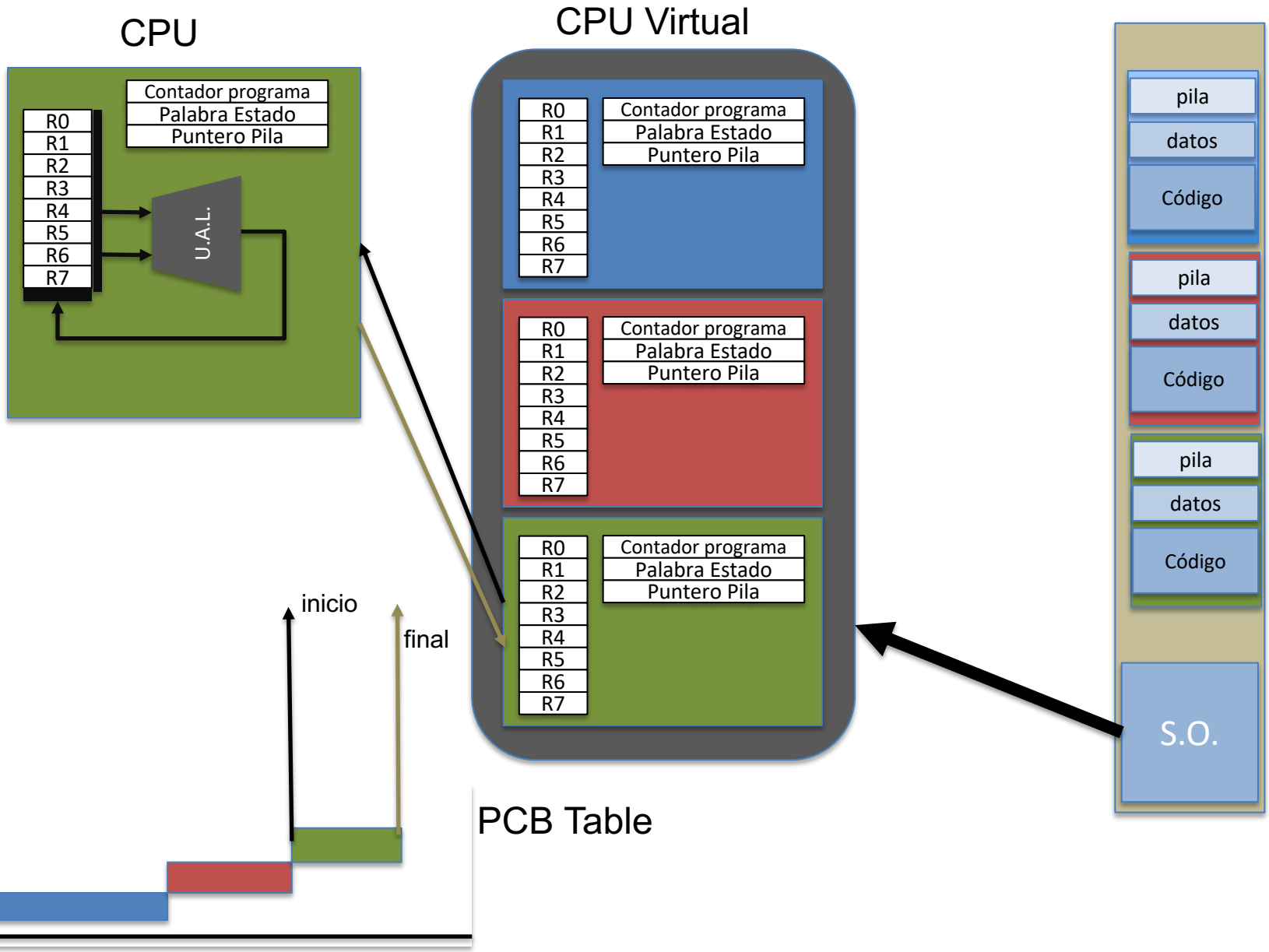
Concurrent execution



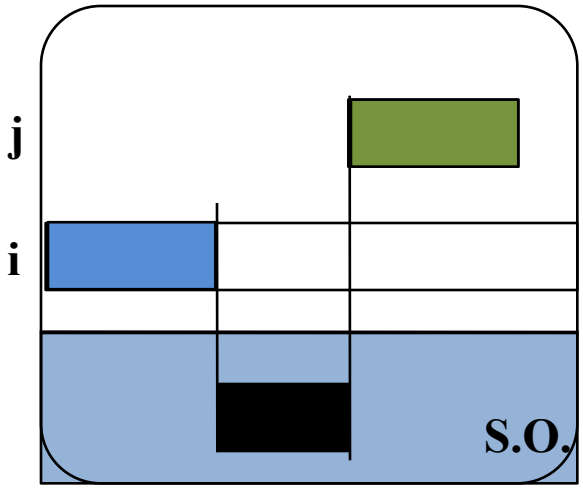
Concurrent execution

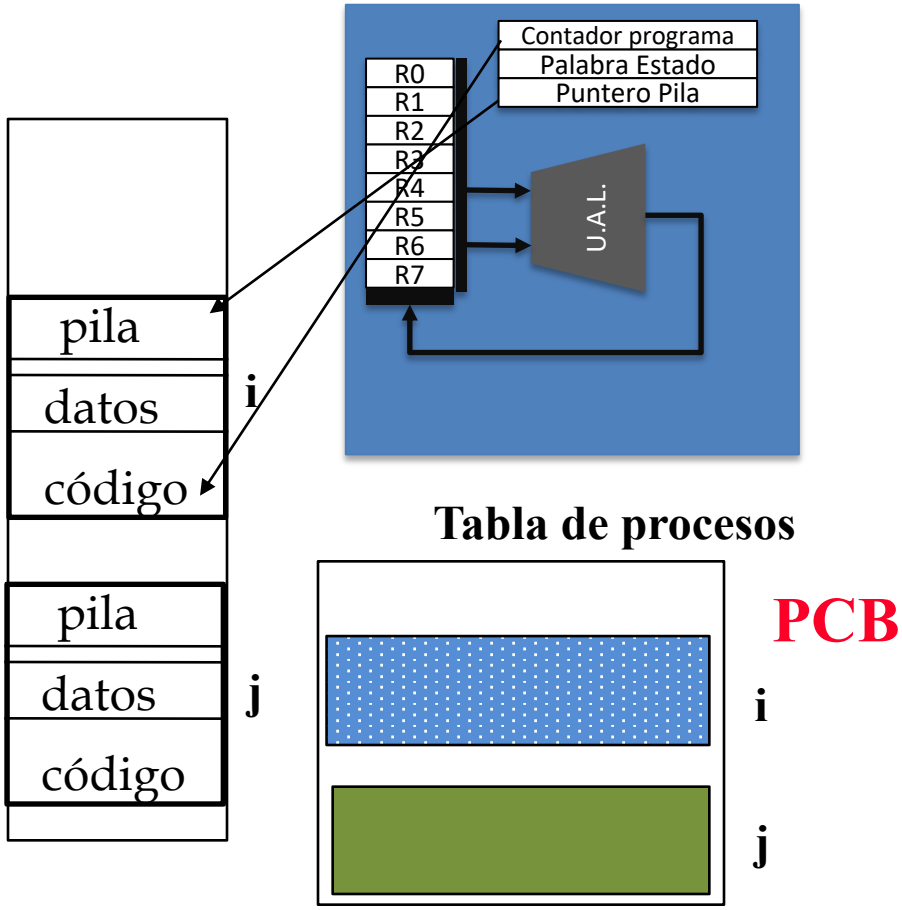


Concurrent execution

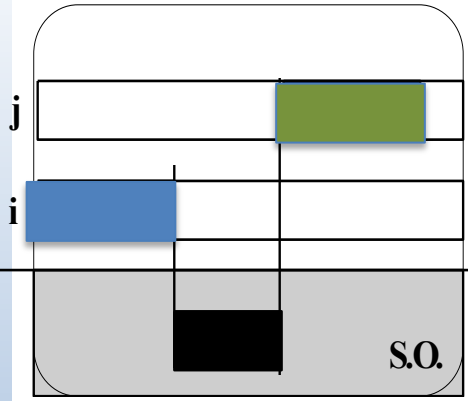


Context Switch

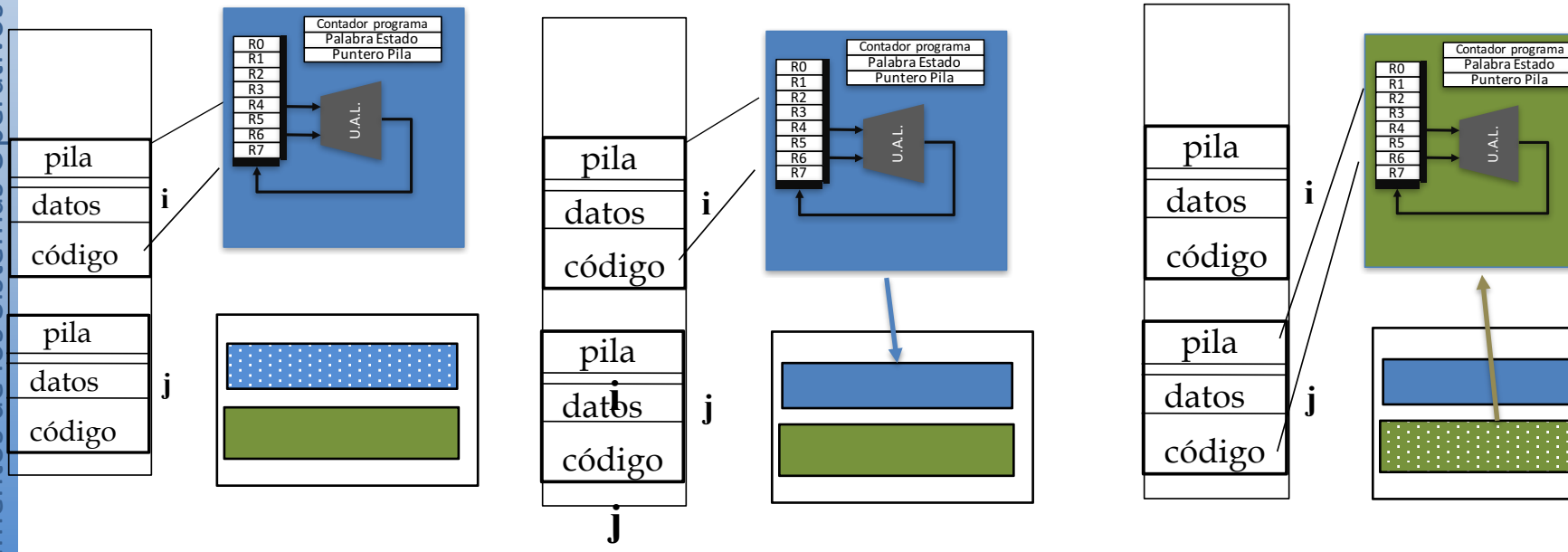




Context Switch



1. Process I is in execution
2. Interrupt arrives
3. Save context process i
4. Interrupt execution
5. OS selects the next process to b executed (j)
6. Load process j context



- 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

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

	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

Content

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

Exercise 1:

- 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
		Selecting a process to schedule in the CPU
		Performing a system call
		Providing a confortable user interface

Exercise 2

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

Process		State
P1	The CPU is executing intructions belowing to P1	
P2	P2 has ask 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 table are already created	
P6	P6 has all its SO control structures and its memory image stored in main memory	

Exercise 3

- 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

Exercise 4

- 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?