ARCOS Group
Universidad Carlos III de Madrid

Lesson 3
Process and threads

Operating Systems
Computer Science and Engineering

To remember…

Before classes	Class	After class

Prepare the prerequisites.

Study the material associated with the **bibliography**: slides alone are not enough.
Please ask questions (especially after study).

Exercising skills:

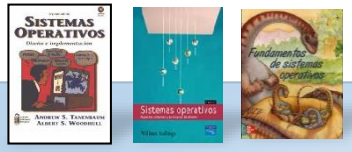
- Perform all **exercises**.
- Carrying out the **practice notebooks** and **the practical exercises** progressively.

Recommended reading

Base

1. Carretero 2020:
   1. Cap. 5
2. Carretero 2007:
   1. Cap. 3 and 7

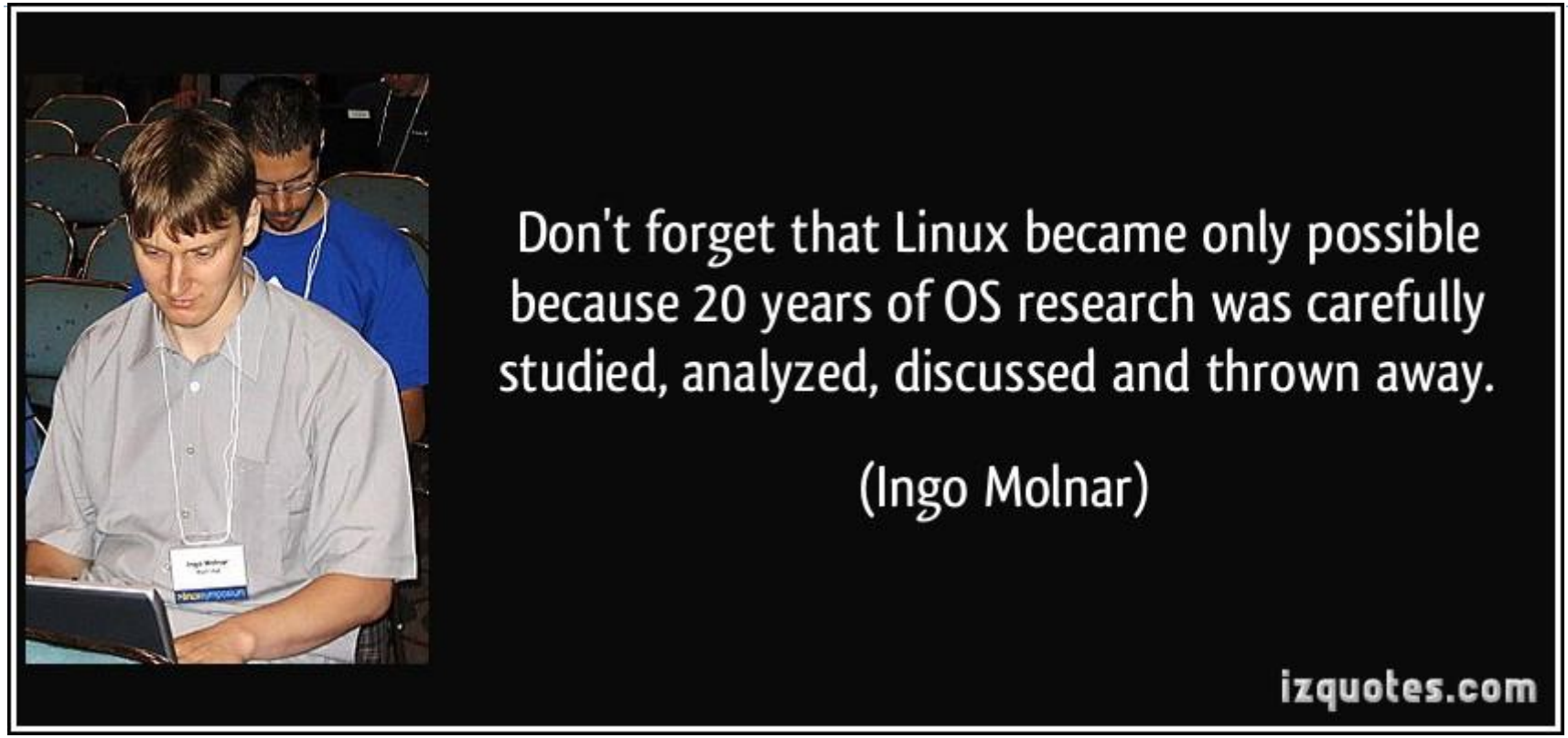
Suggested



1. Tanenbaum 2006(en):
   1. Cap.3
2. Stallings 2005:
   1. 3.2, 3.3 and 3.5
3. Silberschatz 2006:
   1. 3.1 and 3.3

WARNING!

- This material is a script for the class but it is not the notes of the full course.
- The books given in the bibliography together with what is explained in class represent the study material for the course syllabus.


Don't forget that Linux became only possible because 20 years of OS research was carefully studied, analyzed, discussed and thrown away.
(Ingo Molnar)
izquotes.com

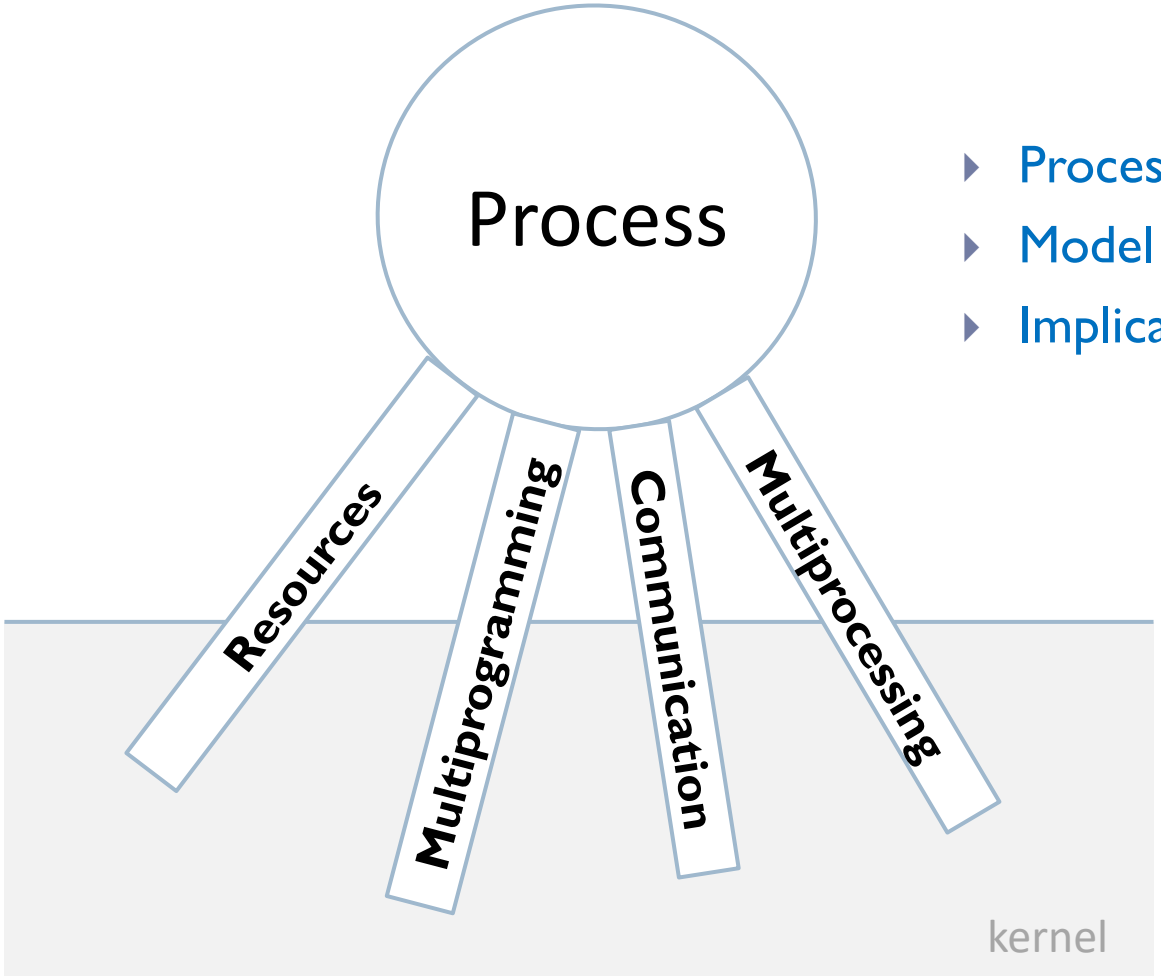
Contents

1. Introduction
   - Process definition.
   - Model offered: resources, multiprogramming, multitasking and multiprocessing
2. Process life cycle: process status.
3. Services to manage processes provided by the operating system.
4. Definition of thread
5. Kernel and library threads.
6. Services for threads in the operating system.

Contents

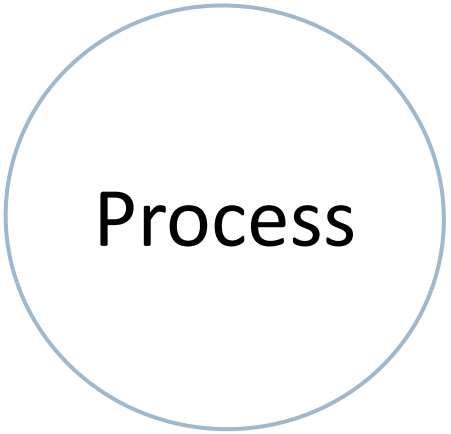
1. **Introduction**
   - **Process definition.**
   - **Model offered: resources, multiprogramming, multitasking and multiprocessing**
2. Process life cycle: process status.
3. Services to manage processes provided by the operating system.
4. Definition of thread
5. Kernel and library threads.
6. Services for threads in the operating system.

Introduction



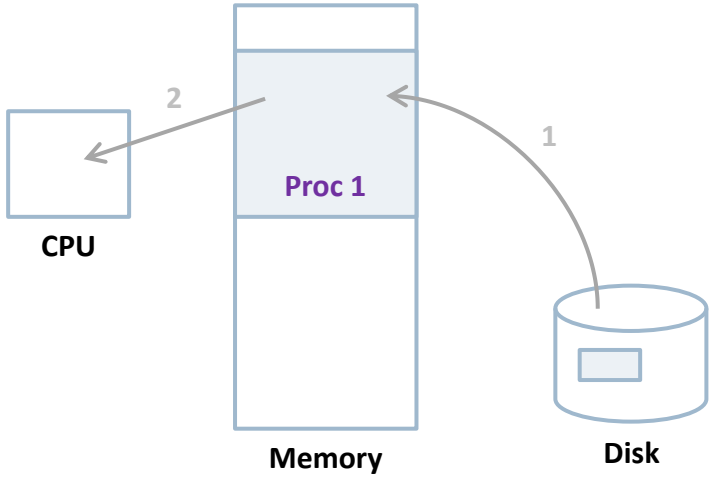
- Process Concept
- Model offered
- Implications in O.S.

Introduction



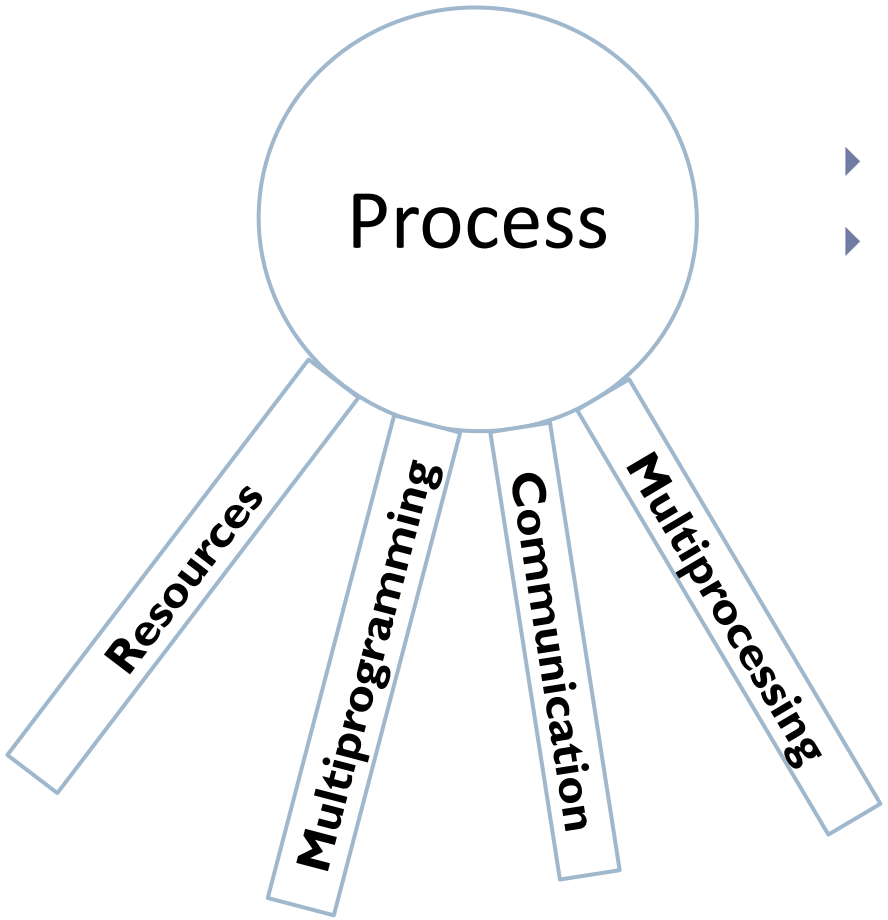
- **Process Concept**

Process Concept



- Process
  - Running program
  - Processing unit managed by the O.S.

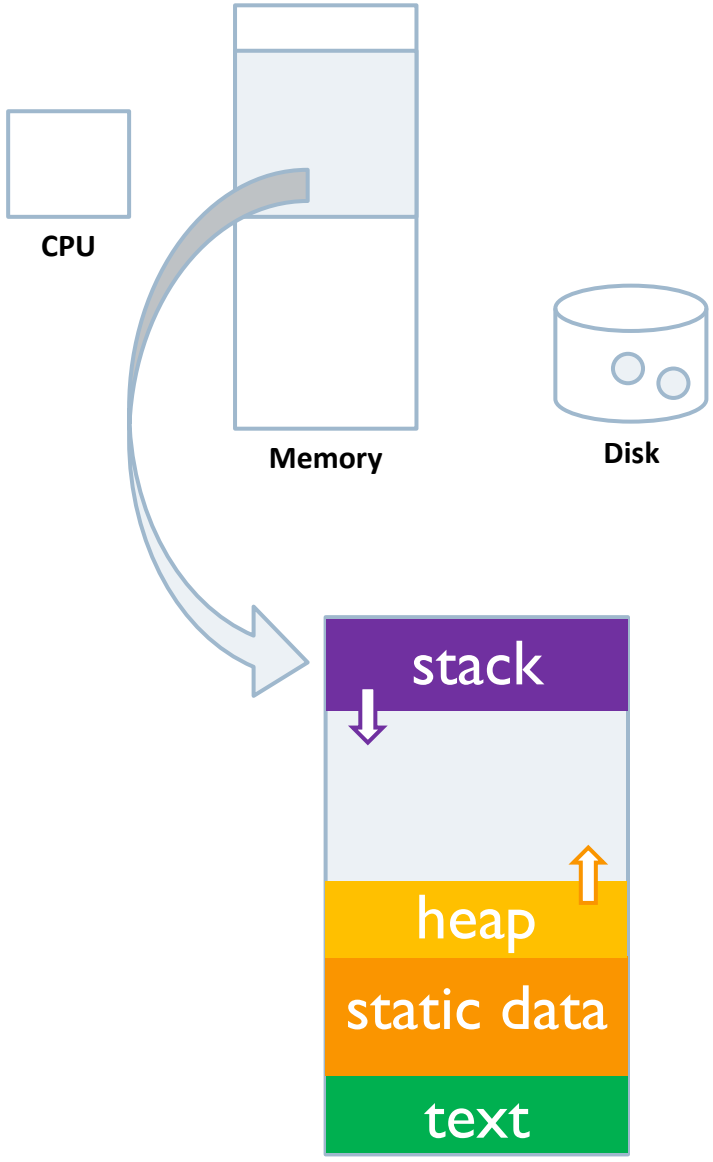
Introduction



- Process Concept
- **Model offered**

Model offered

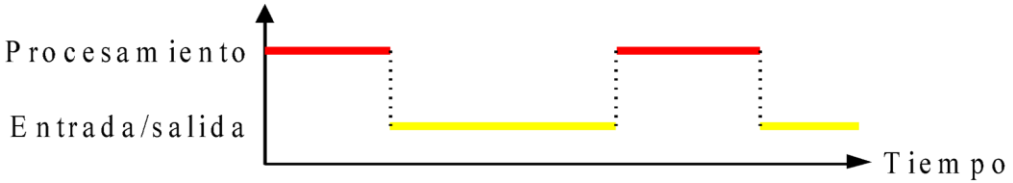
- resources
- multiprogramming
  - protection/sharing
  - process hierarchy
- multitasking
- multiprocessing



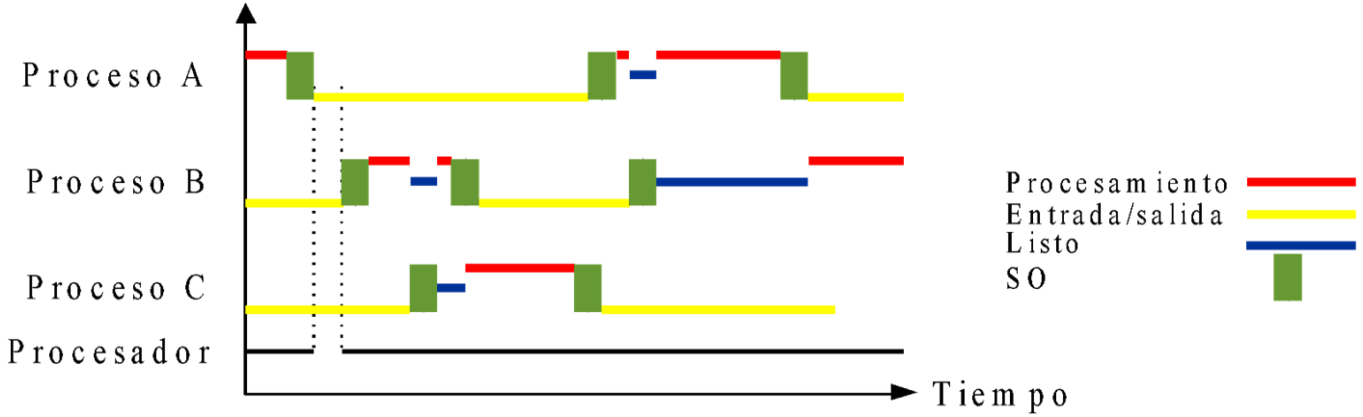
- Associated resources
  - Memory areas
    - At least: code, data and stack
  - Open files
  - Signals

Principles of multiprogramming…

- Alternation of I/O and processing phases in the processes:

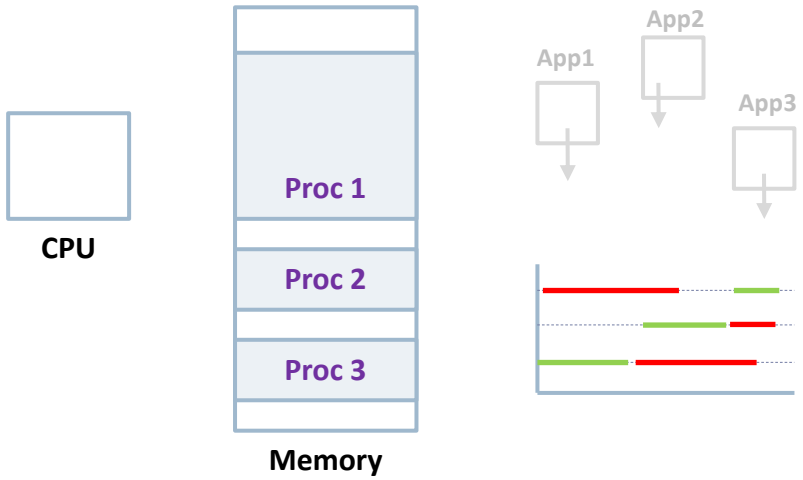


- Memory stores several processes.
- Real parallelism between I/O and CPU/UCP (DMA).



Model offered

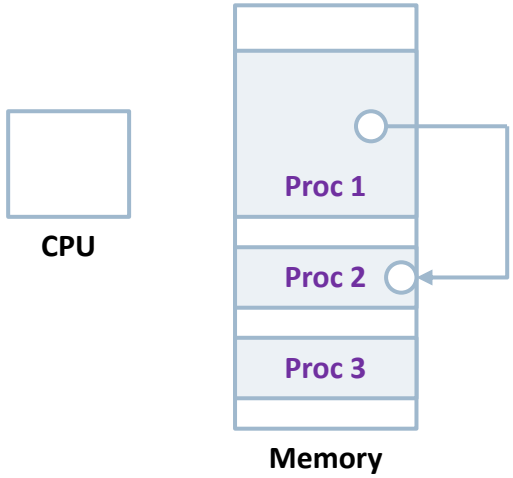
- resources
- **multiprogramming**
  - protection/sharing
  - process hierarchy
- multitasking
- multiprocessing



- Multiprogramming
  - Having several applications in memory
  - If an application is blocked by I/O, then another one is executed until it is blocked too
    - Voluntary Context Switching (V.C.S.)
  - Efficient use of the processor
  - Degree of multiprogramming = number of applications in RAM

Model offered

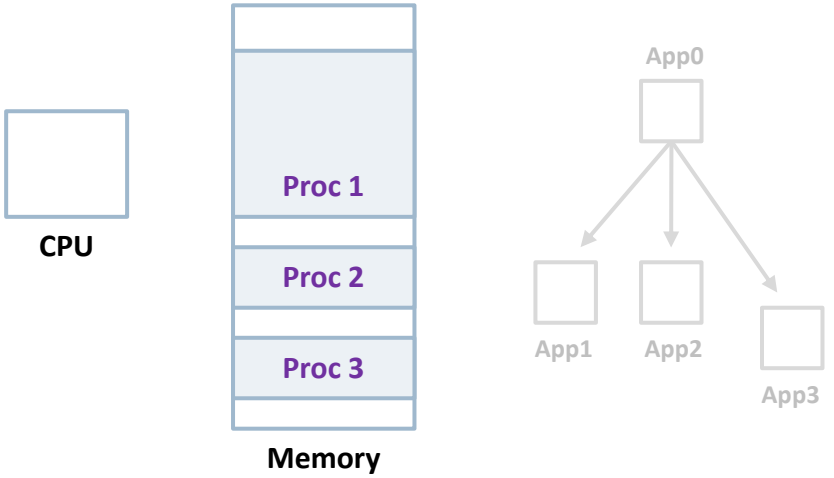
- resources
- multiprogramming
  - **protection/sharing**
  - process hierarchy
- multitasking
- multiprocessing



- Protection / Sharing
  - Private address space per application, but
  - Possibility to communicate data between two applications
    - Message passing
    - Memory sharing

Model offered

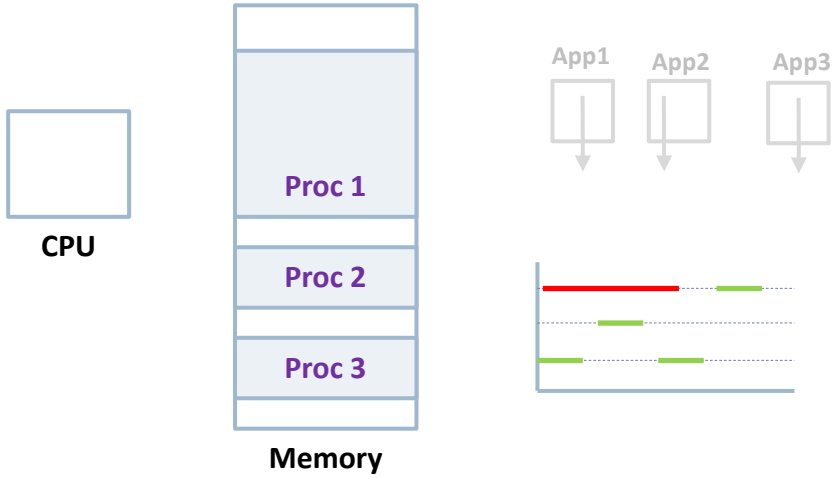
- resources
- multiprogramming
  - protection/sharing
  - **process hierarchy**
- multitasking
- multiprocessing



- Process hierarchy
  - Process creation
    - As a copy of another existing process
    - From a program on disk
    - As a system process at boot
  - Group of processes that can share the same treatment
    - E.g.: in VMS at the end of a process all its children are terminated (cascade).

Model offered

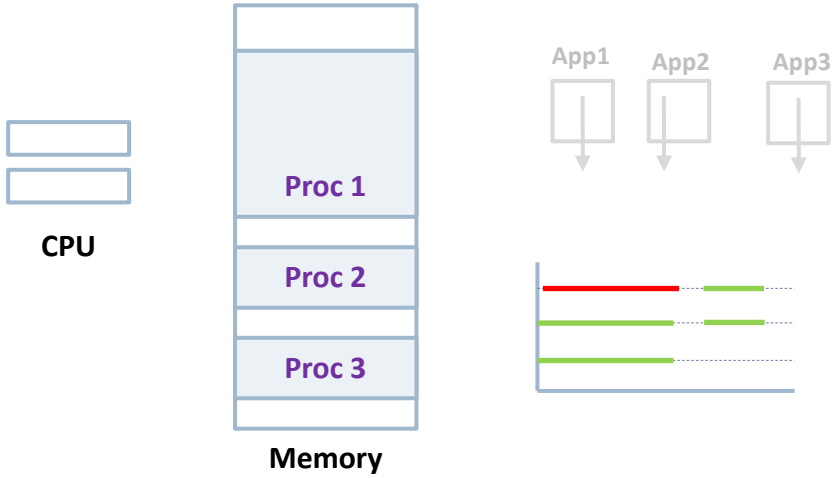
- resources
- multiprogramming
  - protection/sharing
  - process hierarchy
- **multitasking**
- multiprocessing



- Multitasking
  - Each process is executed for a quantum of time (e.g., 5 ms) and the turn is rotated to execute non-blocked processes
    - Involuntary context switching (I.C.S.)
  - Sharing of processor usage
    - Everything seems to run at the same time

Model offered

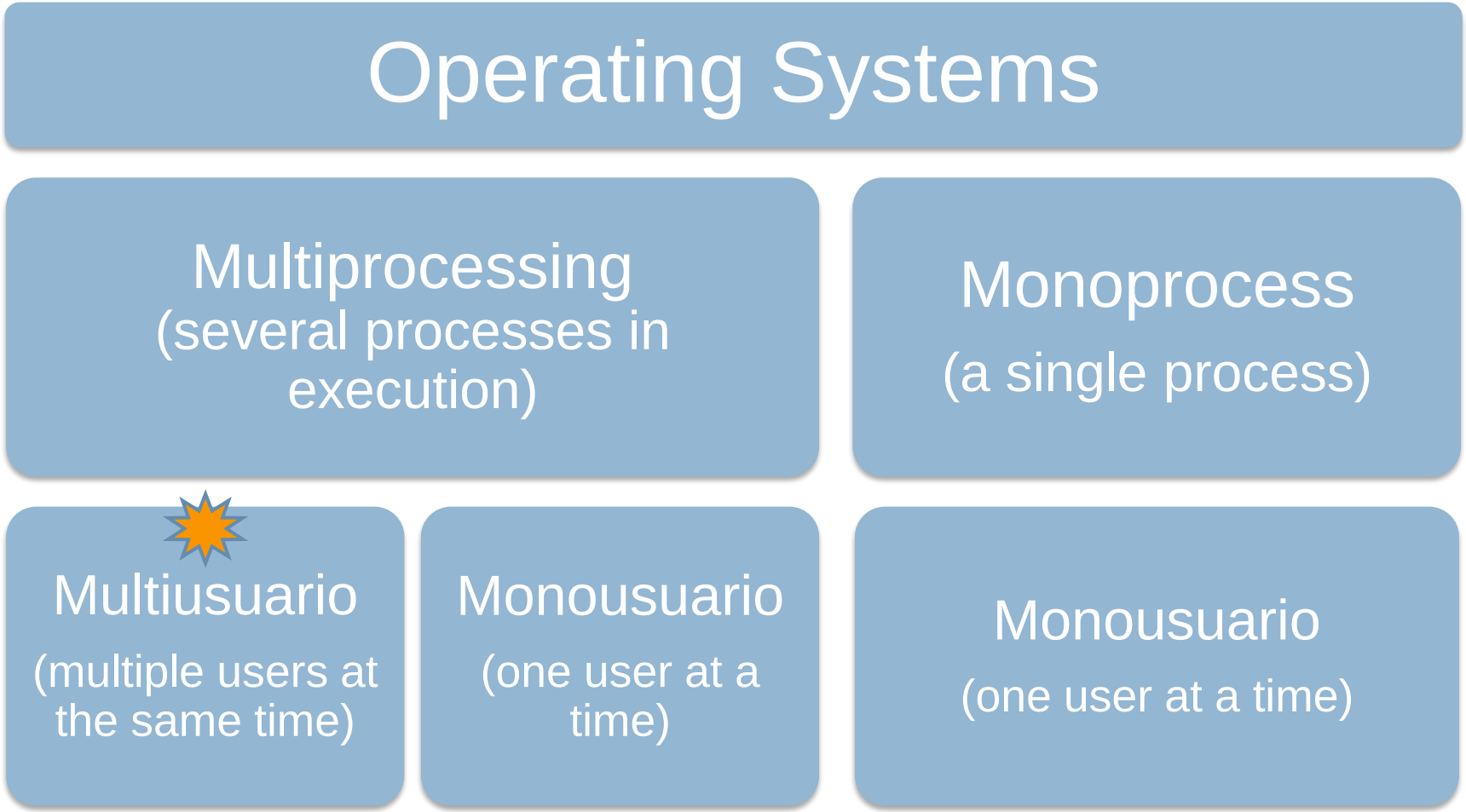
- resources
- multiprogramming
  - protection/sharing
  - process hierarchy
- multitasking
- **multiprocessing**



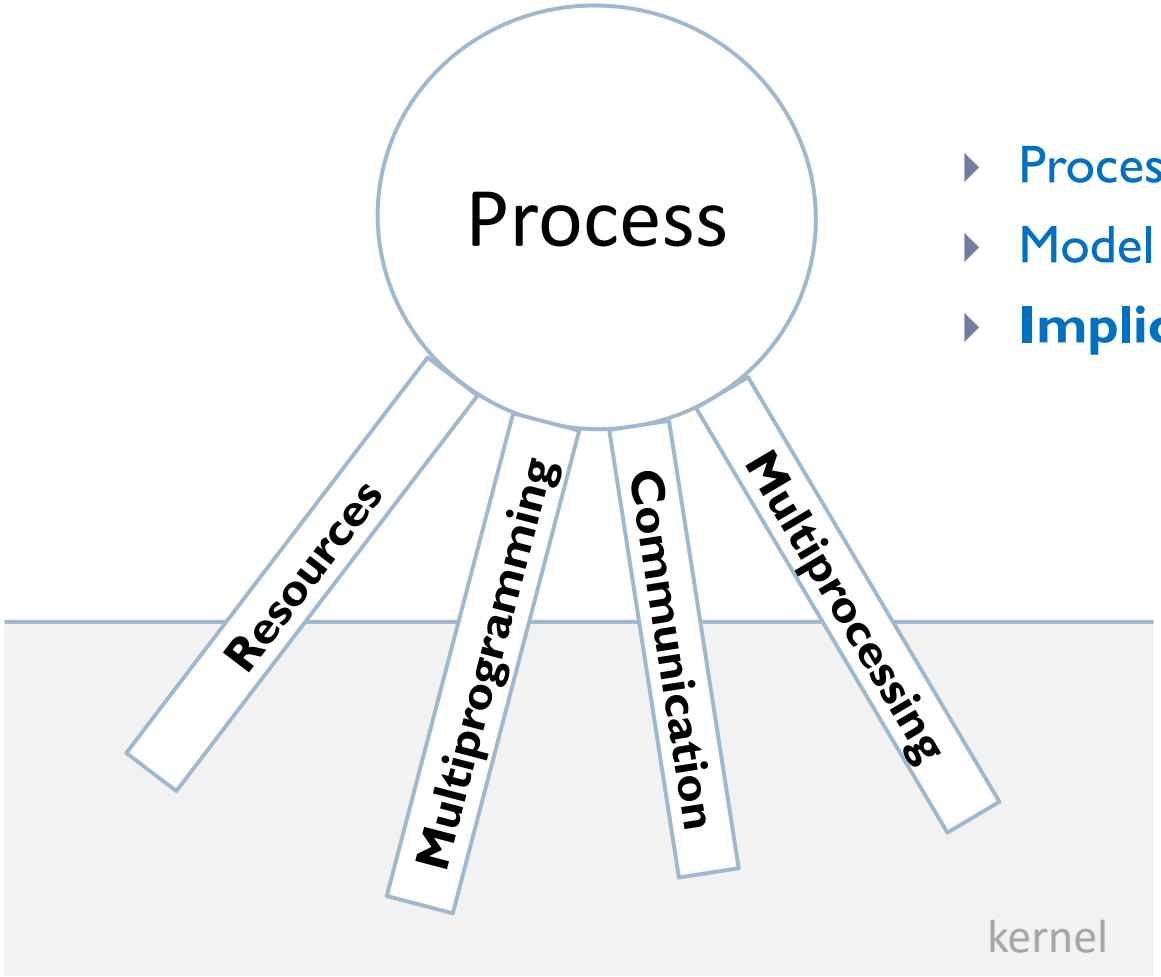
- Multiprocessing
  - Several processors are available (multicore/multiprocessor)
  - In addition to the distribution of each CPU (multitasking) there is real parallelism between several tasks (as many as processors)
    - Scheduler and separate data structures per processor are usually used with some load balancing mechanism

Types of operating systems

tailor-made model



Introduction



- Process Concept
- Model offered
- **Implications in O.S.**

Implications on the operating system

1. Data structures

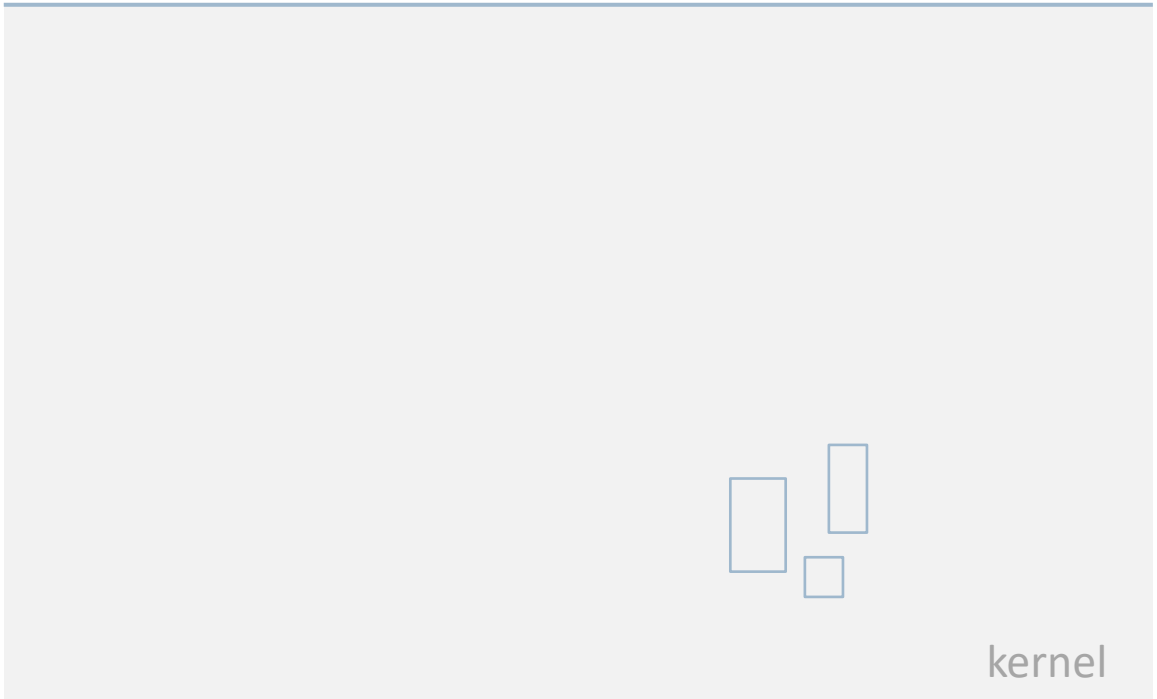
Requirements	Information (in data structures)	Functions (internal, service and API)
Multiprogramming	• State of execution • Context: CPU registers… • List of processes	• Hw/Sw devices Int. • Scheduler • Create/Destroy/Schedule process
Multiprogramming	• State of execution • Context: registros de CPU… • List of processes	• Hw/Sw devices Int. • Scheduler • Create/Destroy/Schedule process

○ Protection / Sharing	• Message passing
	  • Receive message queue
	• Shared memory
	  • Zones, locks and conditions
		• Message sending/receiving and message tail management
		• Concurrency API and data structure management

○ Process hierarchy	• Family relationship • Sets of related processes • Processes of the same session	• Clone/Change process image • Associate processes and indicate representative process
Multitasking	• Remaining quantum • Priority	• Clock hw/sw int. • Scheduler • Create/Destroy/Schedule process
Multiprocessing	• Afinity	• Clock hw/sw int. • Scheduler • Create/Destroy/Schedule process

Implications on the operating system

1. Data structures



Implications on the operating system

2. Functions: internal management

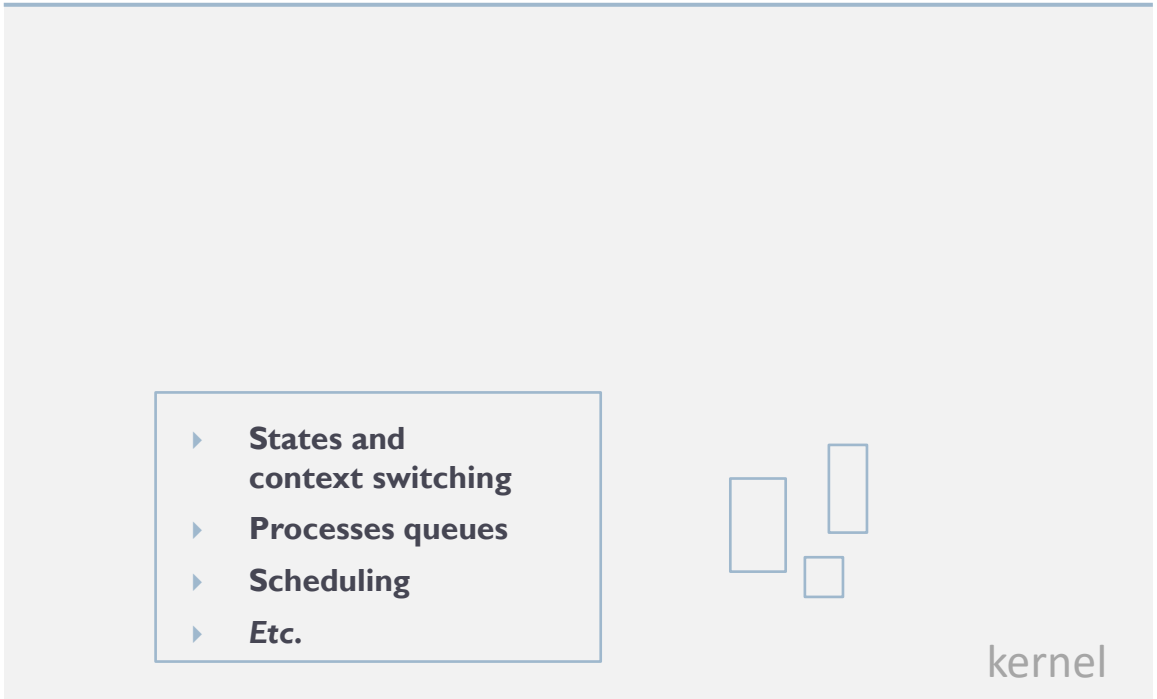
Requirements	Information (in data structures)	Functions (internal, service and API)
Multiprogramming	• State of execution • Context: CPU registers… • List of processes	• Hw/Sw devices Int. • Scheduler • Create/Destroy/Schedule process
Multiprogramming	• State of execution • Context: registros de CPU… • List of processes	• Hw/Sw devices Int. • Scheduler • Create/Destroy/Schedule process

○ Protection / Sharing	• Message passing
	  • Receive message queue
	• Shared memory
	  • Zones, locks and conditions
		• Message sending/receiving and message tail management
		• Concurrency API and data structure management

○ Process hierarchy	• Family relationship • Sets of related processes • Processes of the same session	• Clone/Change process image • Associate processes and indicate representative process
Multitasking	• Remaining quantum • Priority	• Clock hw/sw int. • Scheduler • Create/Destroy/Schedule process
Multiprocessing	• Afinity	• Clock hw/sw int. • Scheduler • Create/Destroy/Schedule process

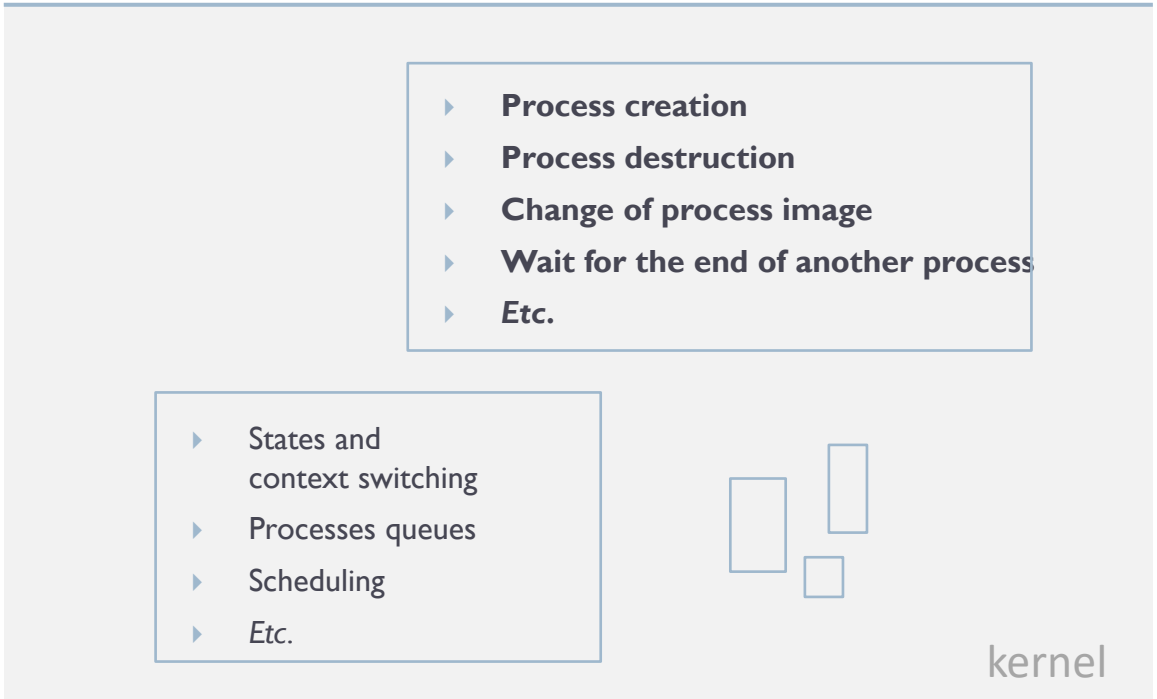
Implications on the operating system

2. Functions: internal management



Implications on the operating system

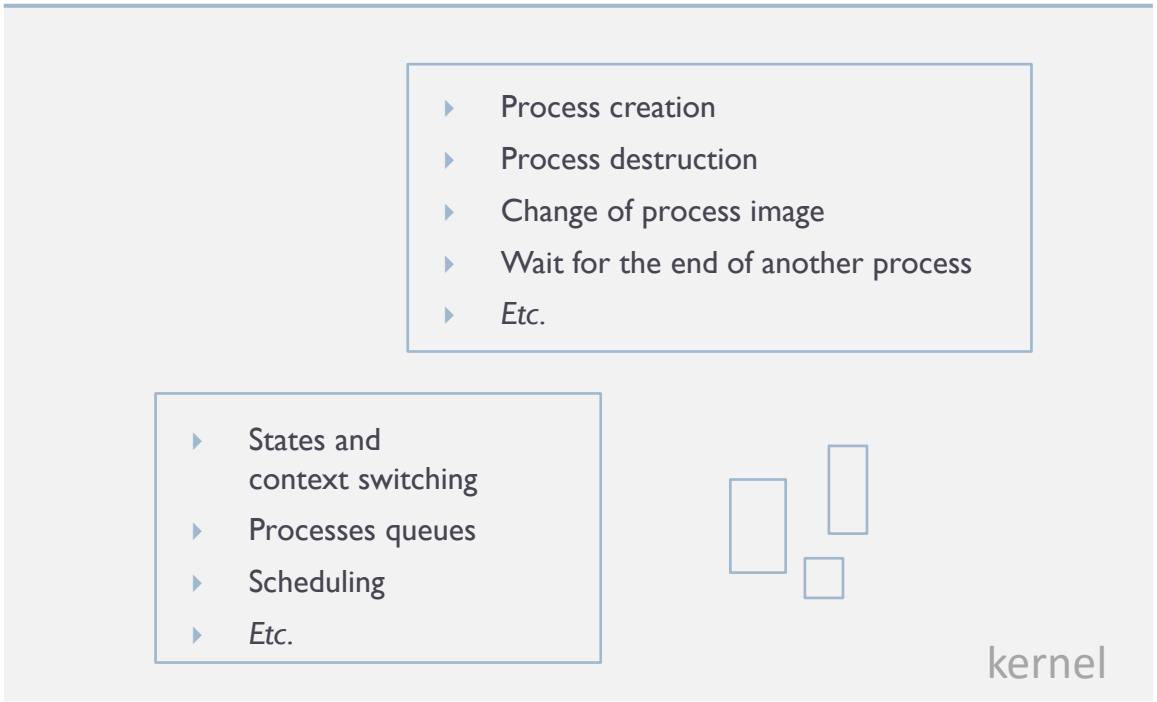
3. Functions: service



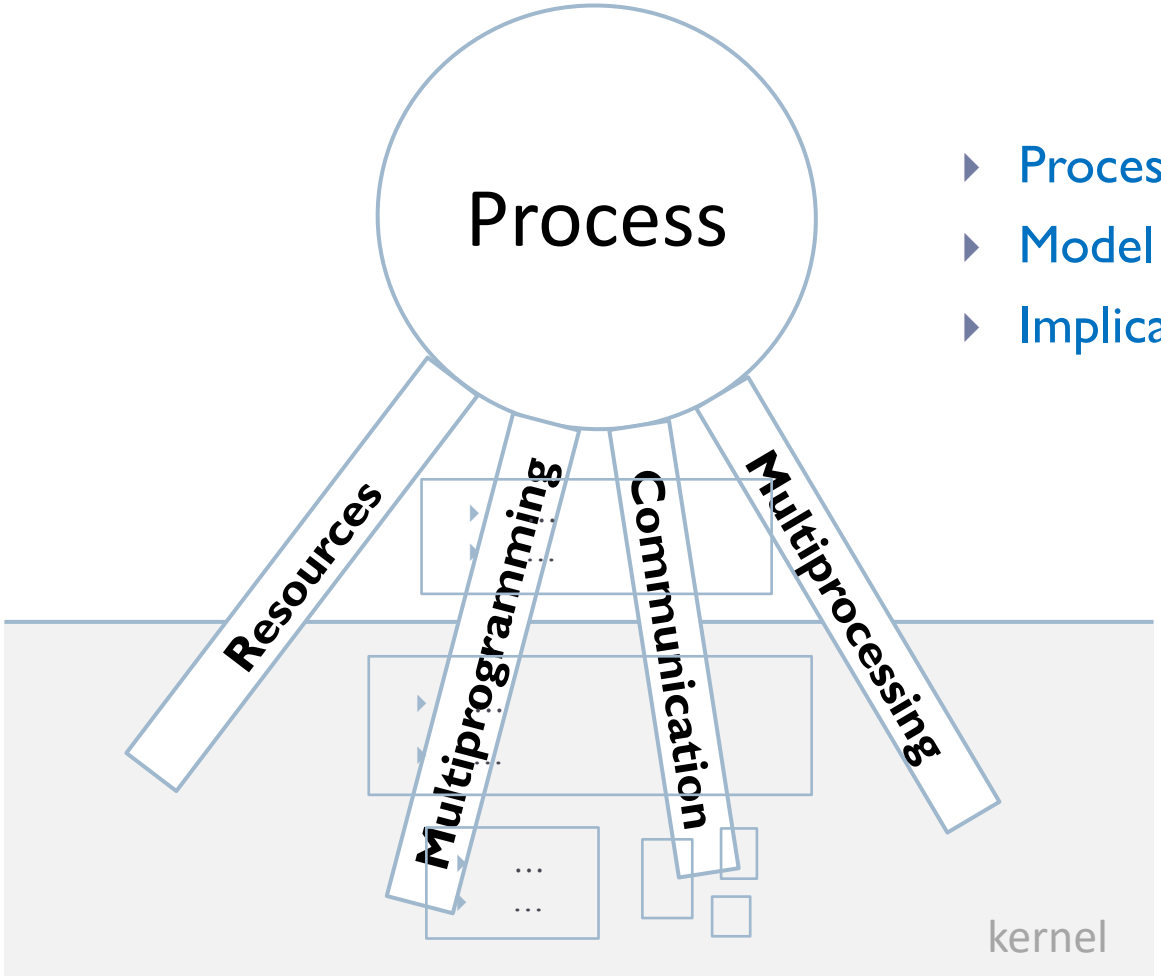
Implications on the operating system

3. Functions: Service API

- fork, exit, exec, wait, …
- *pthread_create, pthread…*

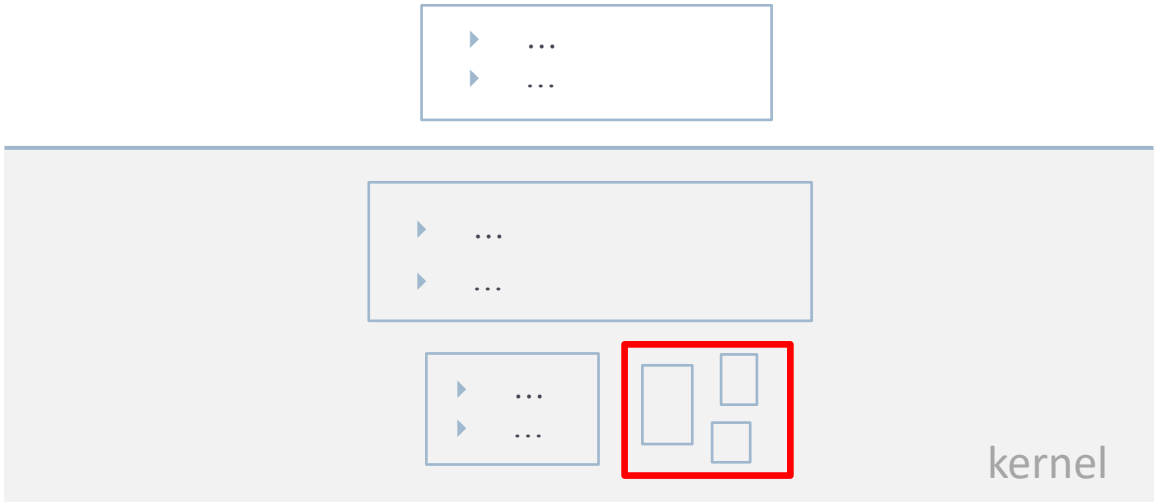


Introduction
summary

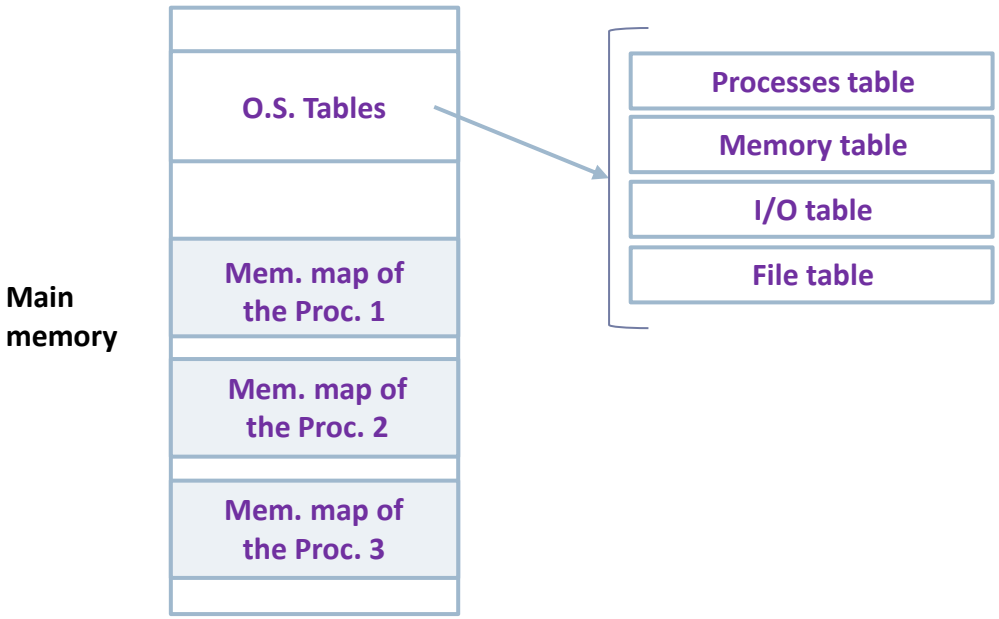


- Process Concept
- Model offered
- Implications in O.S.

Main data structures

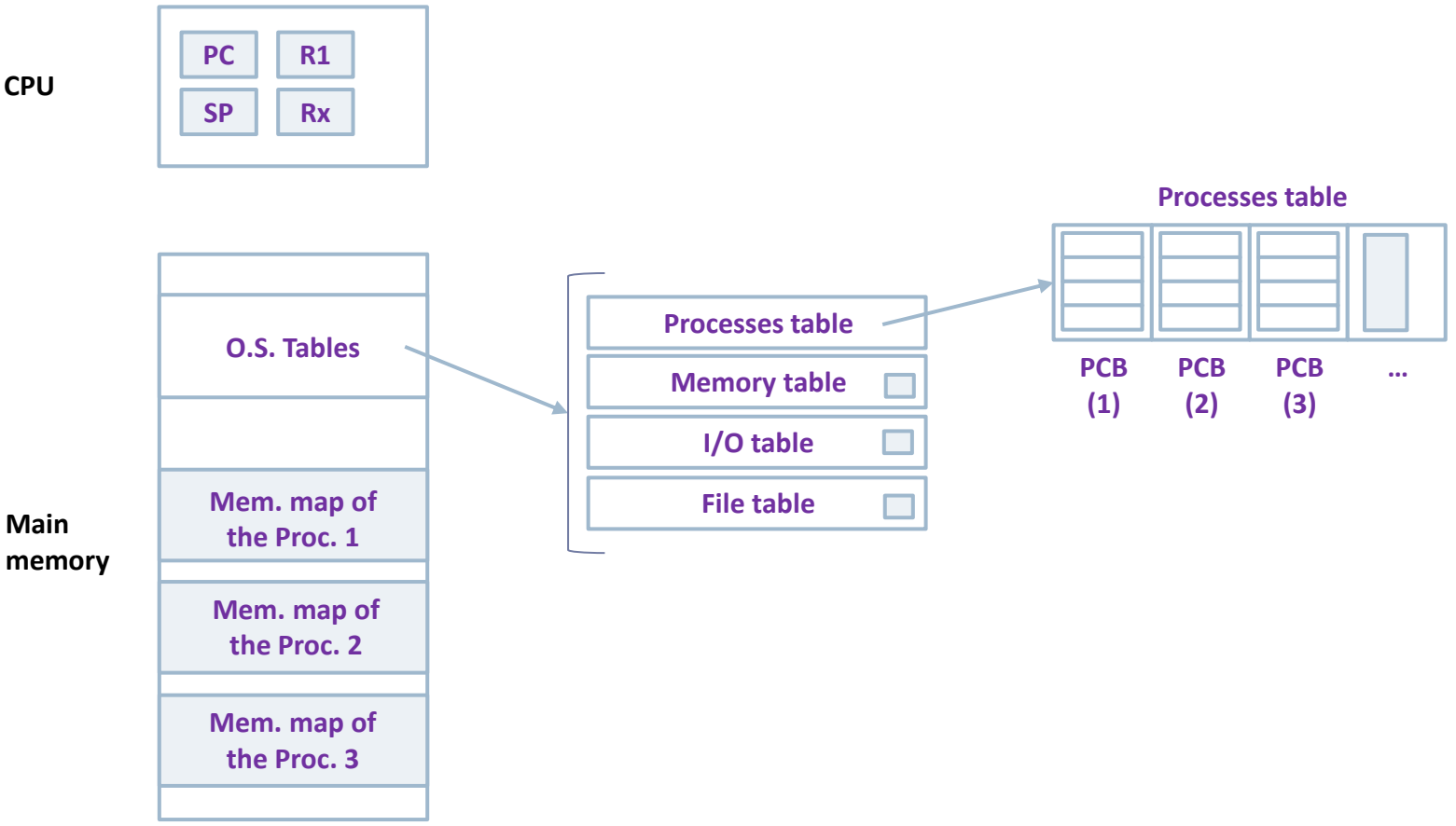


Information in the operating system



Information for a process is in:

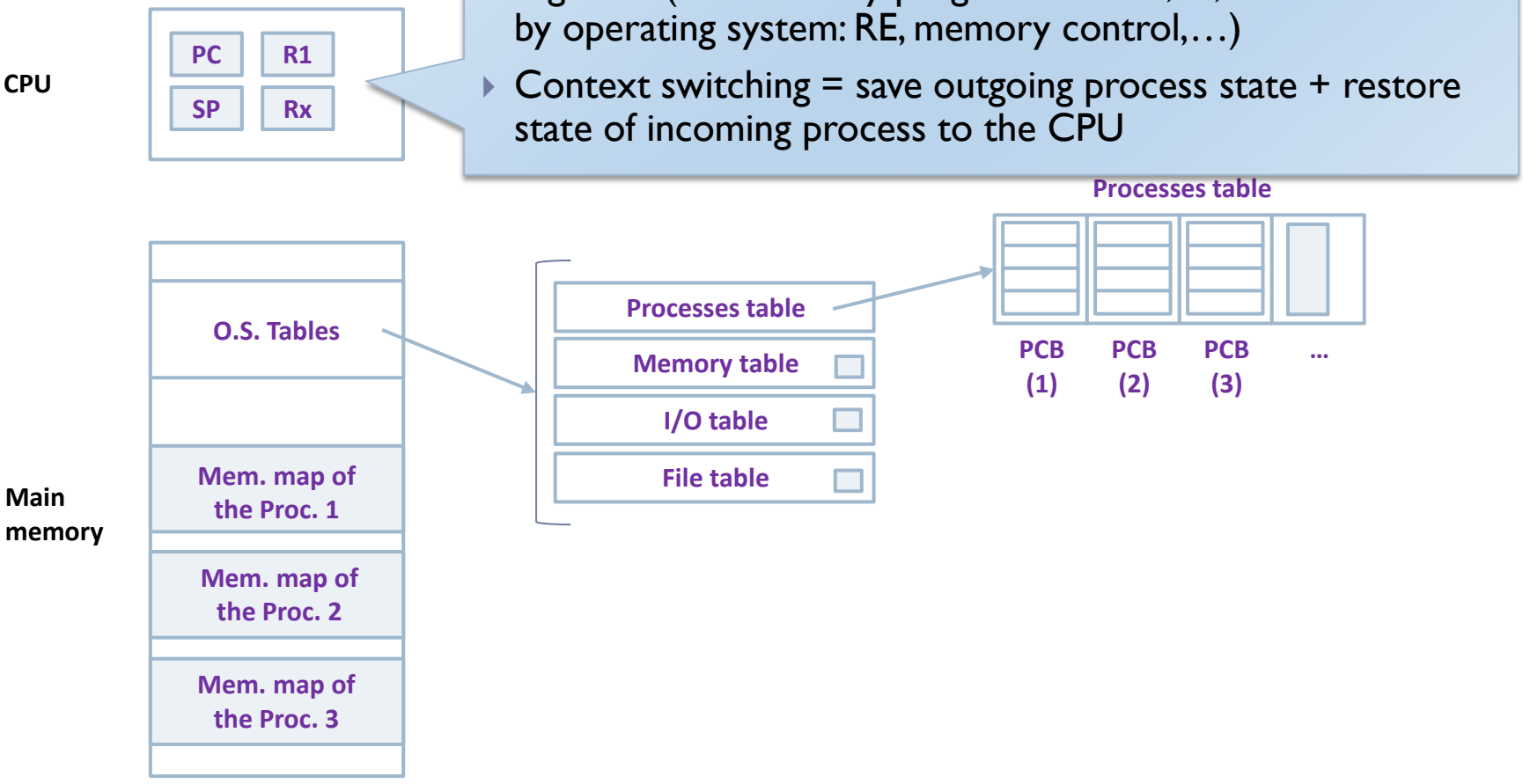
processor + main memory + additional data of the O.S.



Information for a process

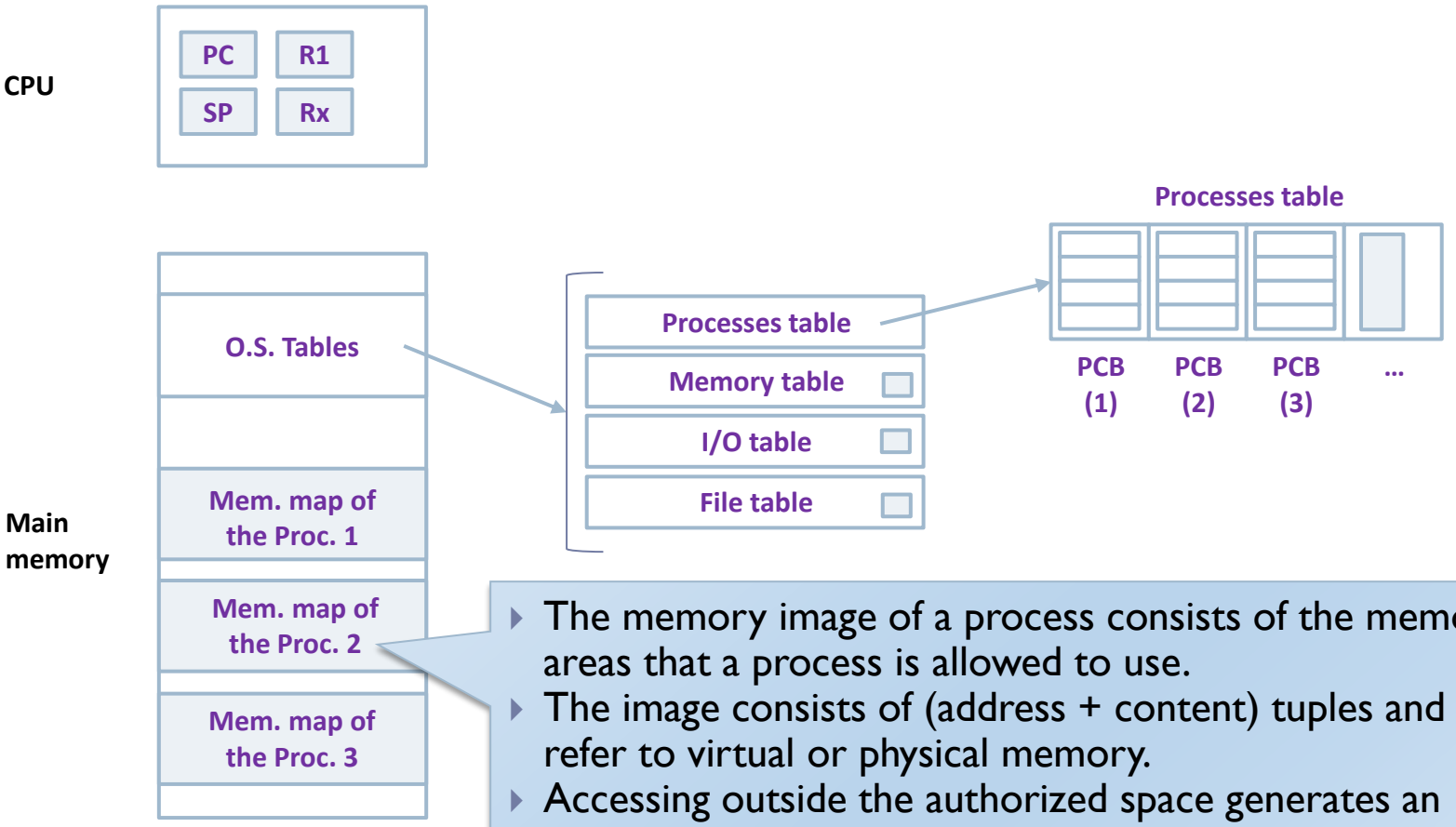
processor status

- **The processor status** includes all values of processor registers (accessible by programmer: PC, SP, ... and accessible by operating system: RE, memory control,…)
- Context switching = save outgoing process state + restore state of incoming process to the CPU



Information for a process

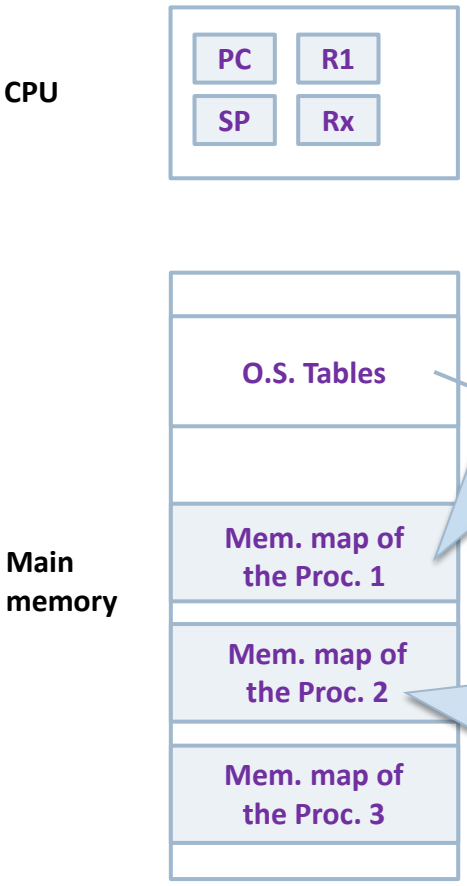
memory image



- The memory image of a process consists of the memory areas that a process is allowed to use.
- The image consists of (address + content) tuples and can refer to virtual or physical memory.
- Accessing outside the authorized space generates an exception.

Information for a process

memory image



	… fixed size	… variable size

Process with single region…	Used in systems without virtual memory	▸ Systems without Virtual Memory:
		  ▸ Needs reserve space ➔ Memory wastage.
		▸ Systems with Virtual Memory:
		  ▸ Virtual reserve space ➔ Feasible, less flexible than multiple regions.

Process with fixed number of regions…		▸ Prefixed regions (text, data, stack). ▸ Each region can grow. ▸ With V.M. the gap between stack and data does not consume physical resources

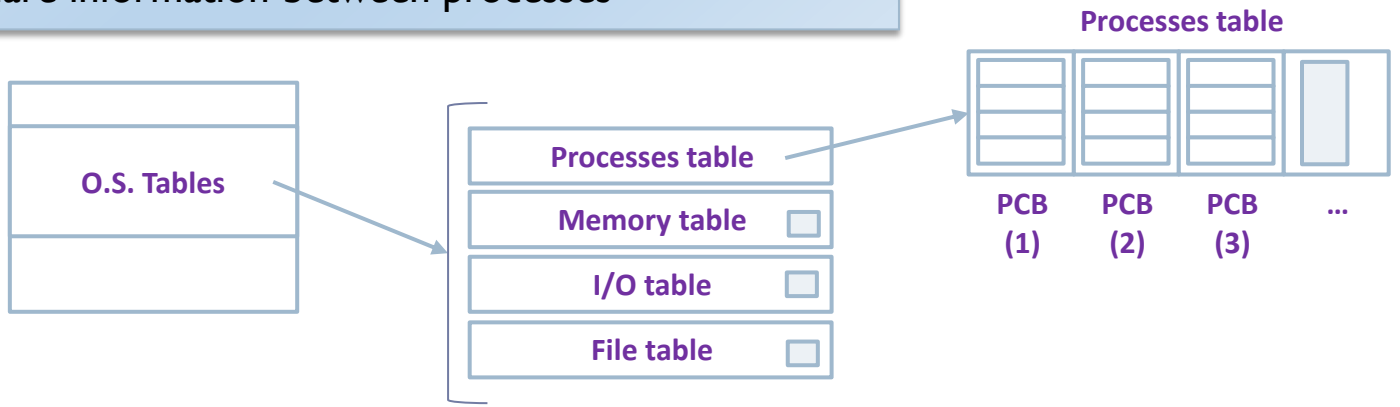
Process with variable number of regions…		▸ A process is structured in an arbitrary number of regions (more recent).
		▸ More advanced and very flexible:
		  ▸ Shared regions.
		  ▸ Regions with different permissions.

- The memory image of a process consists of the memory areas that a process is allowed to use.
- The image consists of (address + content) tuples and can refer to virtual or physical memory.
- Accessing outside the authorized space generates an exception.

Information for a process

data managed by the O.S.

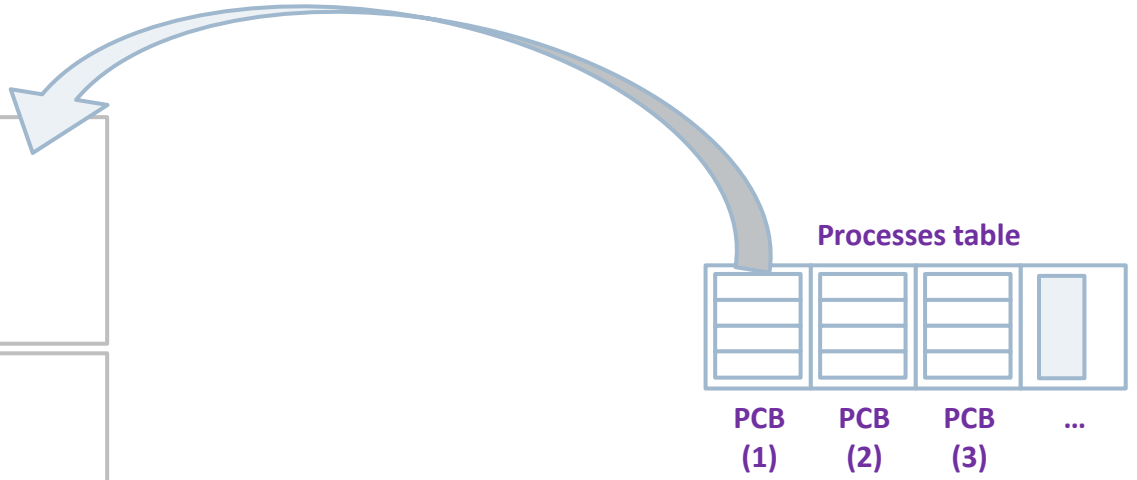
- The information of each process is in the PCB…
- …Information outside the PCB:
  - Because of efficiency reasons
  - To share information between processes



- Examples:
  - Table of memory segments and pages
  - Table of file position pointers
  - List of requests to device

PCB: entry of the processes table

- Process management
  - state
    - General registers
    - Program counter
    - State register
    - Stack pointer
  - Id.
    - Process identifier
    - Parent process
    - Process group
  - control
    - Priority
    - Scheduler parameters
    - Signals
    - Execution start time
    - CPU usage time
    - Time to next alarm



- *Process Control Block* (PCB)
  - Data structure with the information necessary to manage a particular process
  - Manifestation of a process in the kernel
- *Thread Control Block* (TCB)
  - Similar to the PCB for each thread of a process

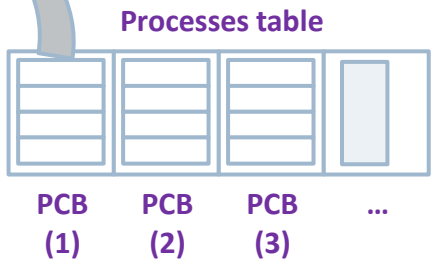
PCB: entry of the processes table

- Process management

state
- General registers
- Program counter
- State register
- Stack pointer

Id.
- Process identifier
- Parent process
- Process group

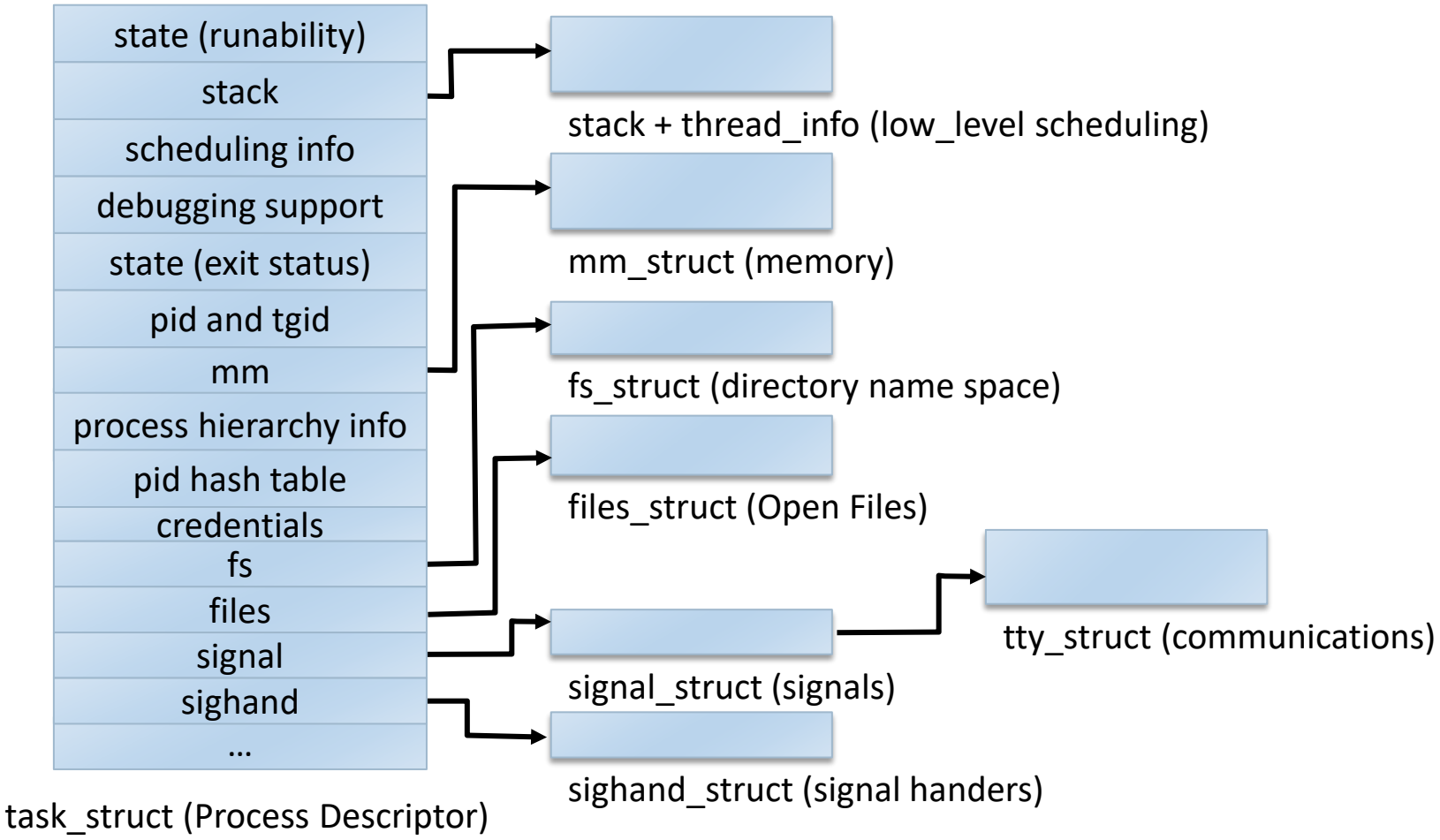
control
- Priority
- Scheduler parameters
- Signals
- Execution start time
- CPU usage time
- Time to next alarm



- *Process Identification* (PID)
  - Identifier for users
  - Dynamically assigned positive 16-bit number (32767), not immediately reused
- *Address of process descriptor* (APD)
  - Identification within the kernel
  - There is a PID -> APD mechanism (e.g.: hash)

Process information

Linux

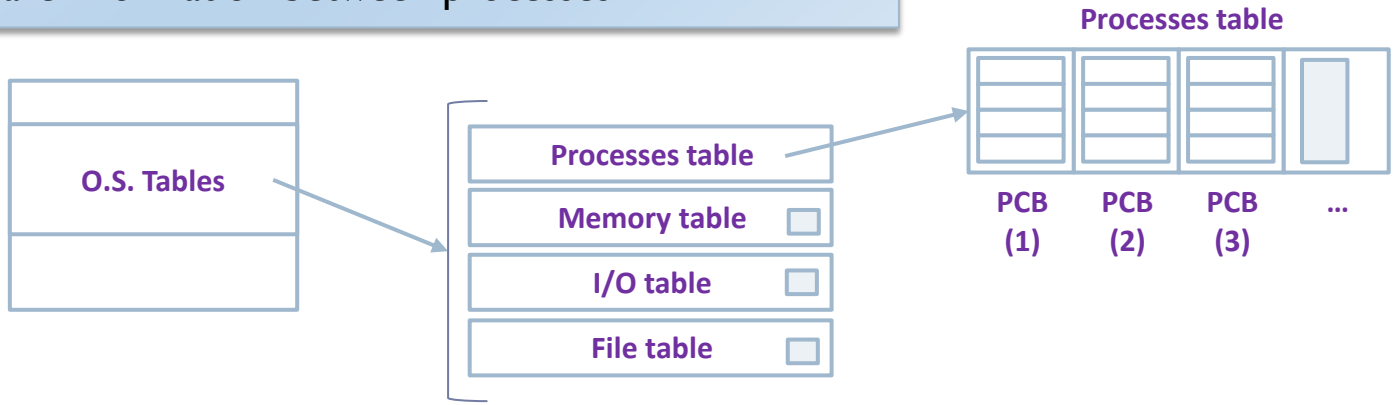


Understanding the Linux Kernel (O'Really)
http://www.eecs.harvard.edu/~cs161/assignments/sched.h.html

Process information

data managed by the O.S.

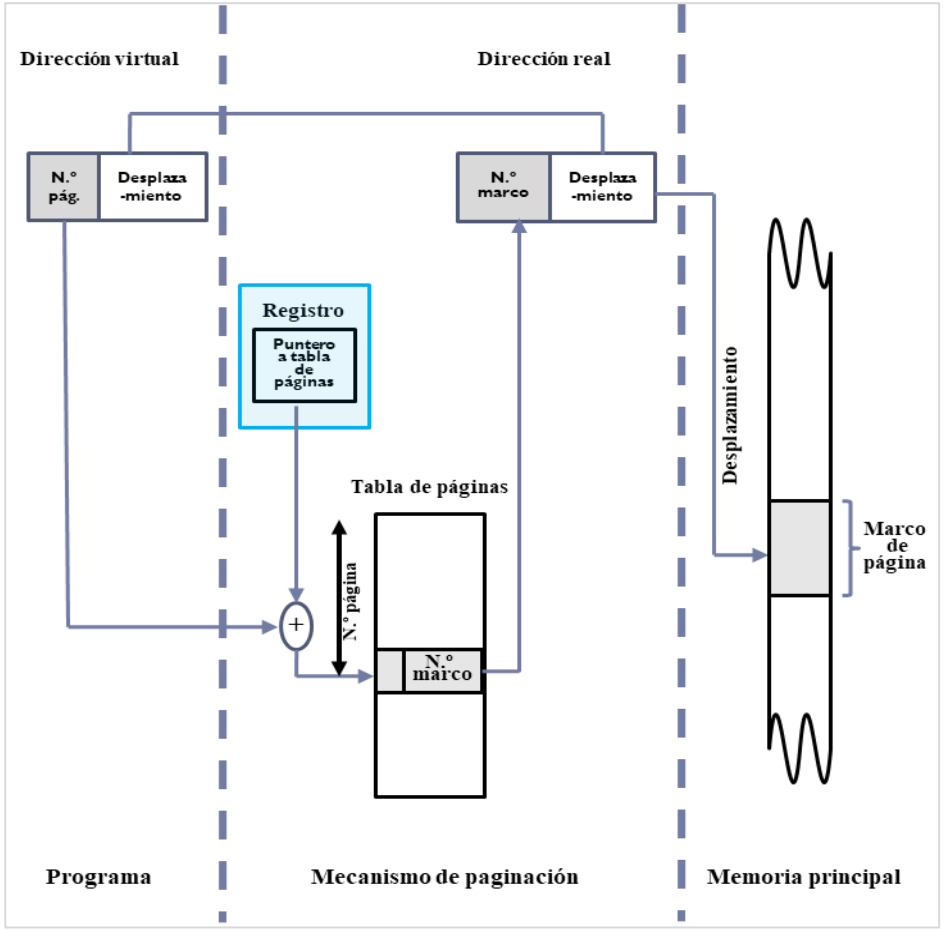
- The information of each process is in the PCB…
- …Information outside the PCB:
  - Because of efficiency reasons
  - To share information between processes



- Examples:
  - **Table of memory segments and pages**
  - **Table of file position pointers**
  - List of requests to device

Information for a process

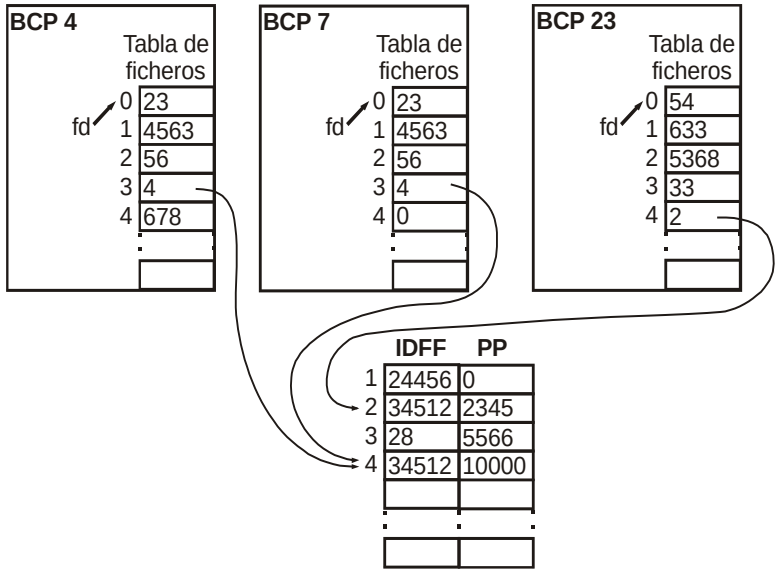
data managed by the O.S. => outside the PCB



- Page table:
  - Describes the memory image of the process.
  - Reasons:
    - It has variable size.
    - Memory sharing between processes requires it to be external to the PCB.
  - The PCB contains the pointer to the page table.

Information for a process

data managed by the O.S. => outside the PCB



- Table of file position pointers:
  - Describes the read/write position of open files.
  - The file state sharing between processes forces it to be external to the PCB.
  - The PCB contains the index of the table element containing the open file information: the i-node and the read/write position.

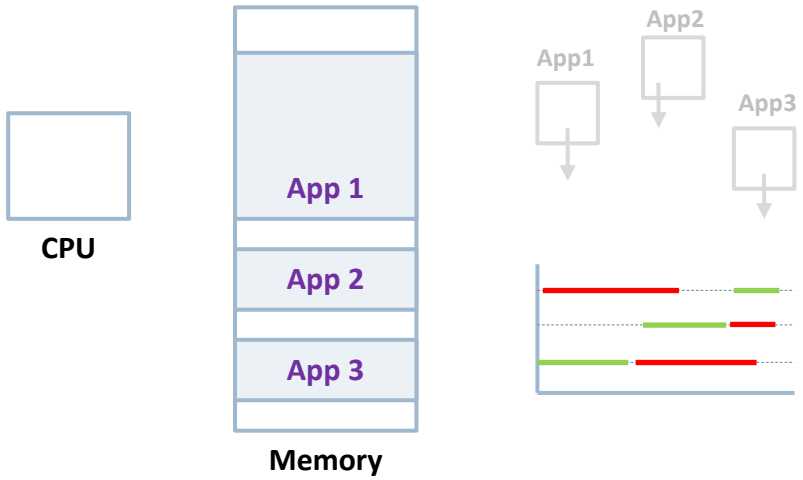
Contents

1. Introduction
   - Process definition.
   - Model offered: resources, multiprogramming, multitasking and multiprocessing
2. **Process life cycle: process status.**
3. Services to manage processes provided by the operating system.
4. Definition of thread
5. Kernel and library threads.
6. Services for threads in the operating system.

Model offered

recap

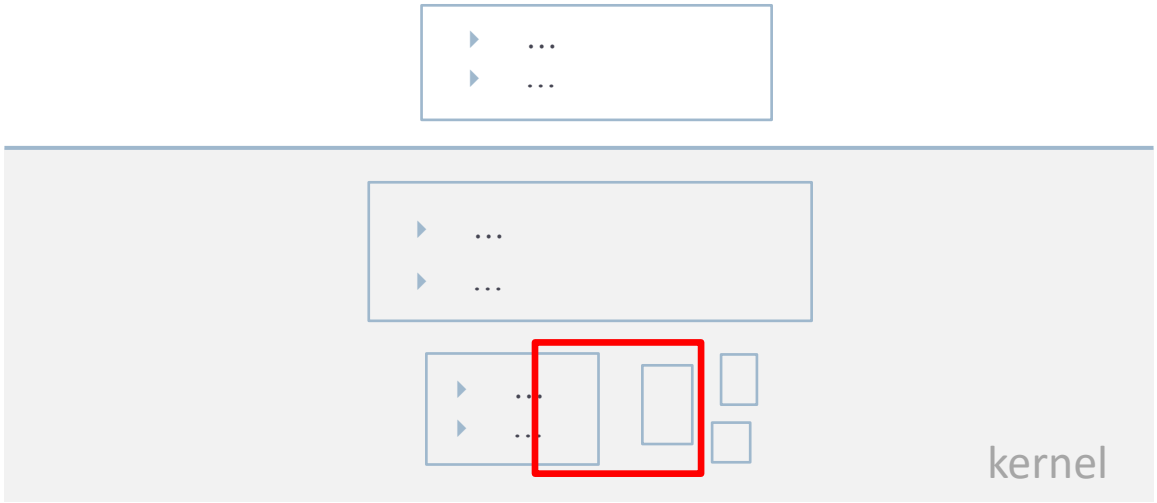
- resources
- **multiprogramming**
  - protection/sharing
  - process hierarchy
- multitasking
- multiprocessing



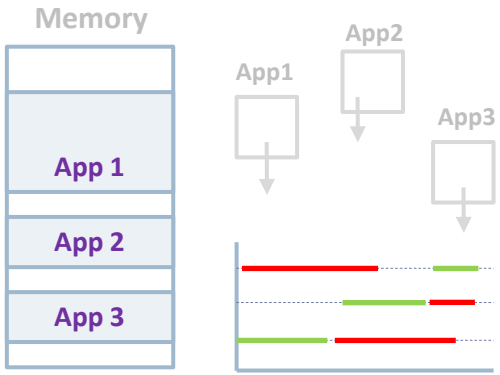
- Multiprogramming
  - Having several applications in memory
  - If an application is blocked by I/O, then another one is executed until it is blocked too
    - Voluntary Context Switching (V.C.S.)
  - Efficient use of the processor
  - Degree of multiprogramming = number of applications in RAM

Multiprogramming (data and functions)

Requirements	Information (in data structures)	Functions (internal, service and API)
Multiprogramming	• State of execution • Context: CPU registers… • List of processes	• Hw/Sw devices Int. • Scheduler • Create/Destroy/Schedule process



Multiprogramming

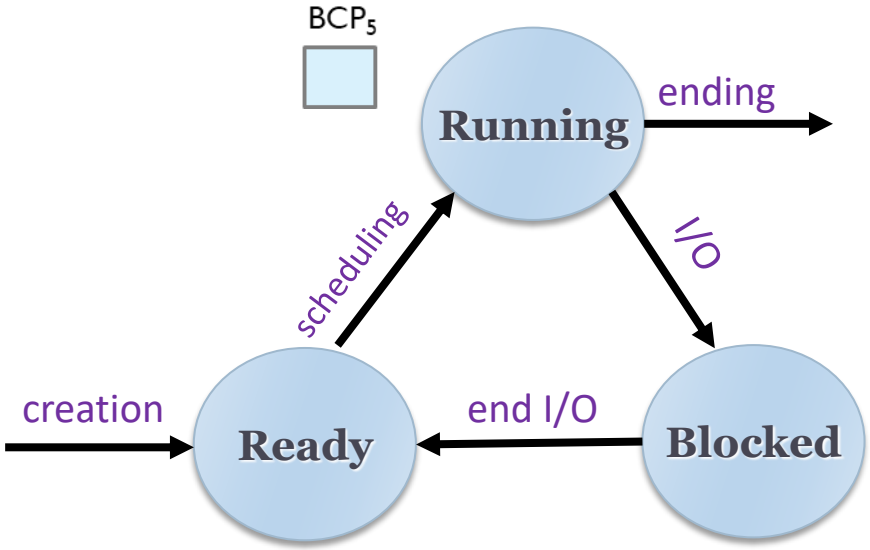


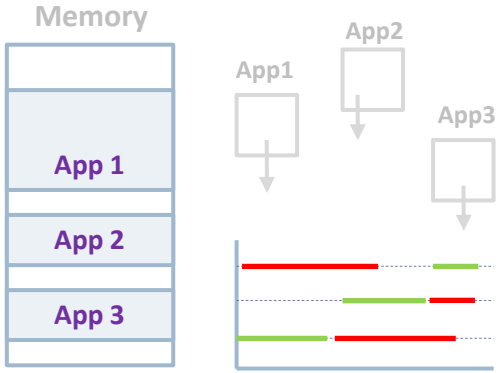
- Having several applications in memory
- If an application is blocked by I/O, then another one is executed (until it is blocked too)
  - Voluntary Context Switching (V.C.S.)

Multiprogramming (data)

States of a process (v.c.s.)

- State
- List/Queue
- Context



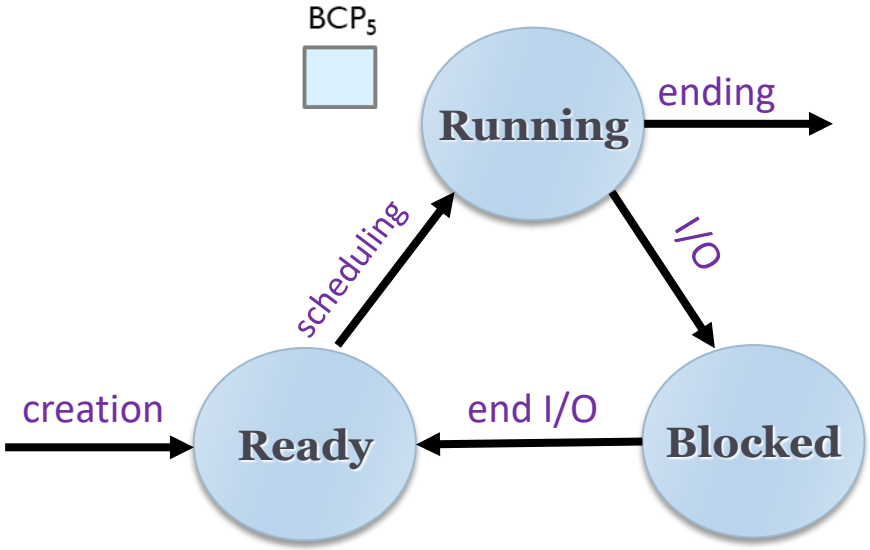


- Having several applications in memory
- If an application is blocked by I/O, then another one is executed (until it is blocked too)
  - Voluntary Context Switching (V.C.S.)

Multiprogramming (data)

States of a process (v.c.s.)

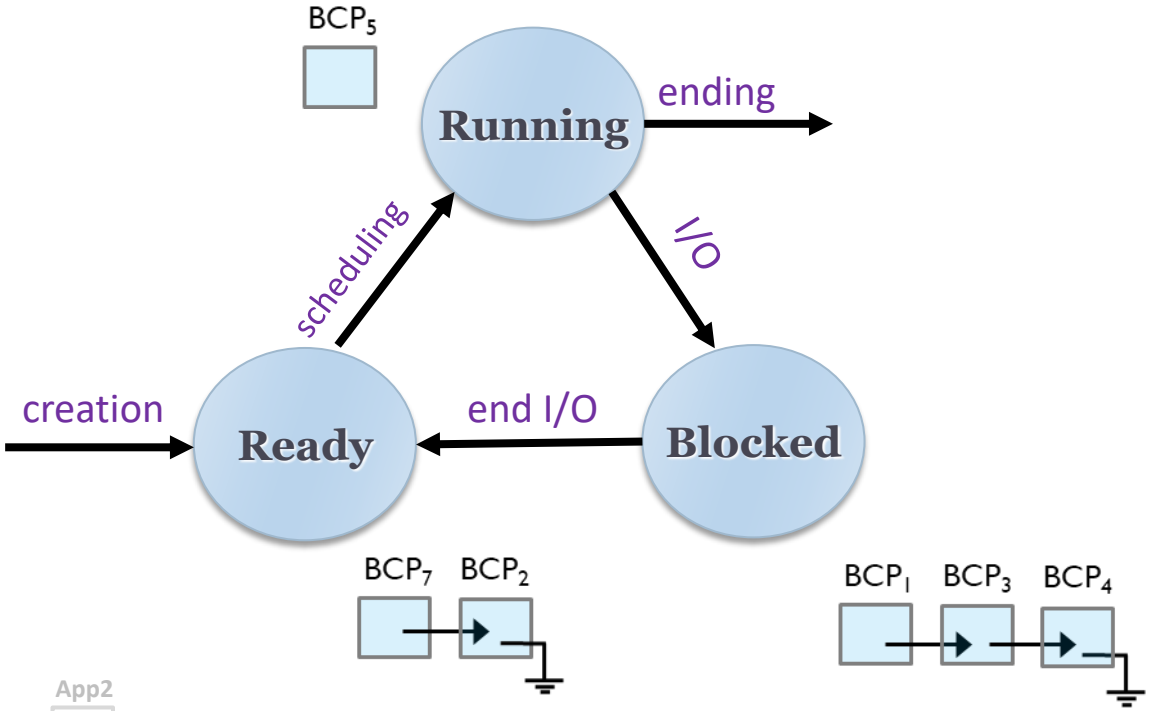
- State
- List/Queue
- Context

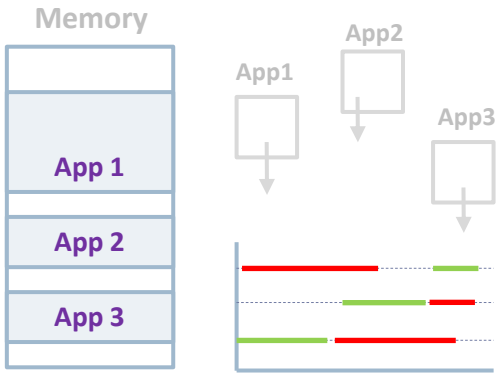


- Running: with CPU assigned
- Ready to execute: no processor available for it
- Blocked: waiting for an event
- Suspended and ready: evicted but ready to run
- Suspended and blocked: evicted and waiting for event

Multiprogramming (data)
List/Queue of processes (v.c.s.)

- State
- **List/Queue**
- Context

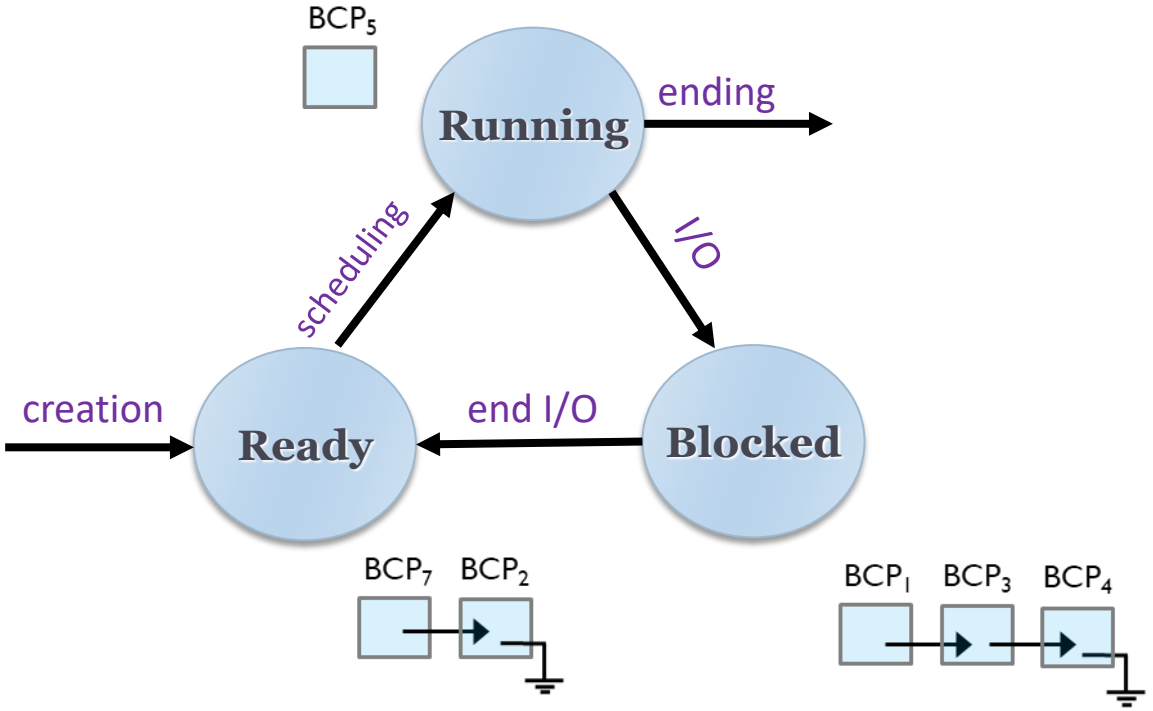




- Having several applications in memory
- If an application is blocked by I/O, then another one is executed (until it is blocked too)
  - Voluntary Context Switching (V.C.S.)

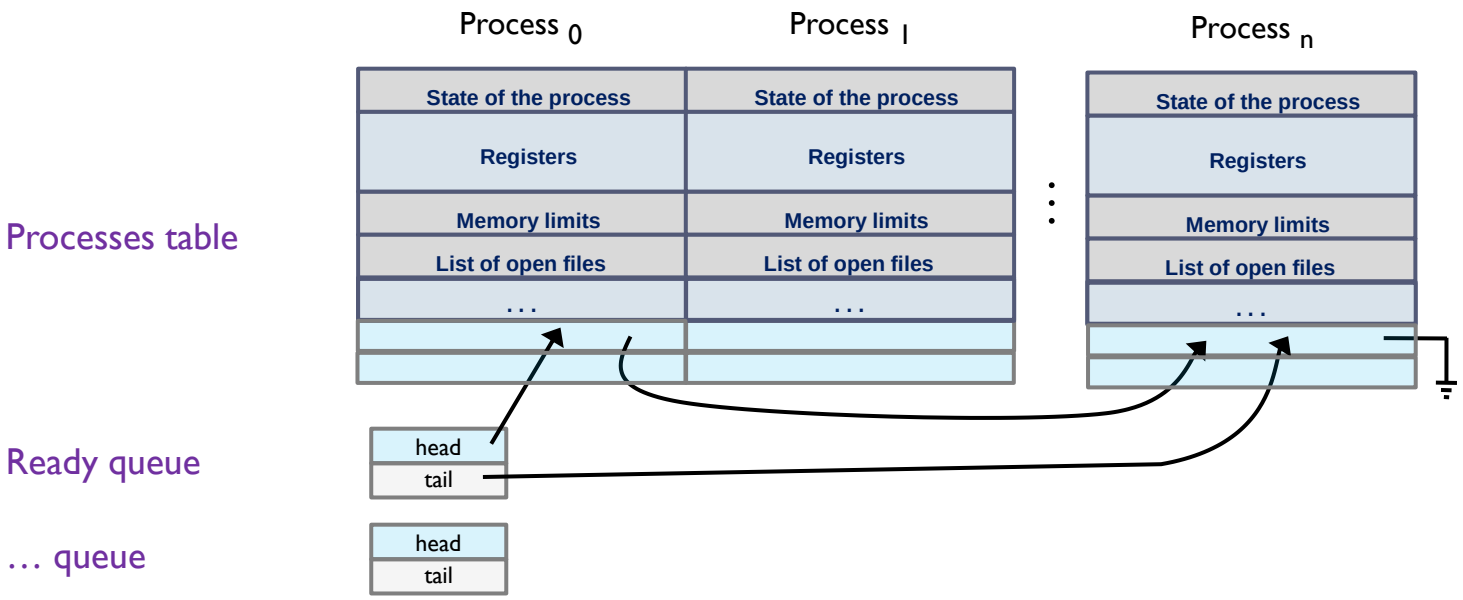
Multiprogramming (data)
List/Queue of processes (v.c.s.)

- State
- **List/Queue**
- Context



- Ready queue: processes waiting to run on CPU
- Queue blocked by resource: processes waiting to finish a blocking request to the associated resource
  - A process can only be in one queue (at most)

Implementation of the process queues

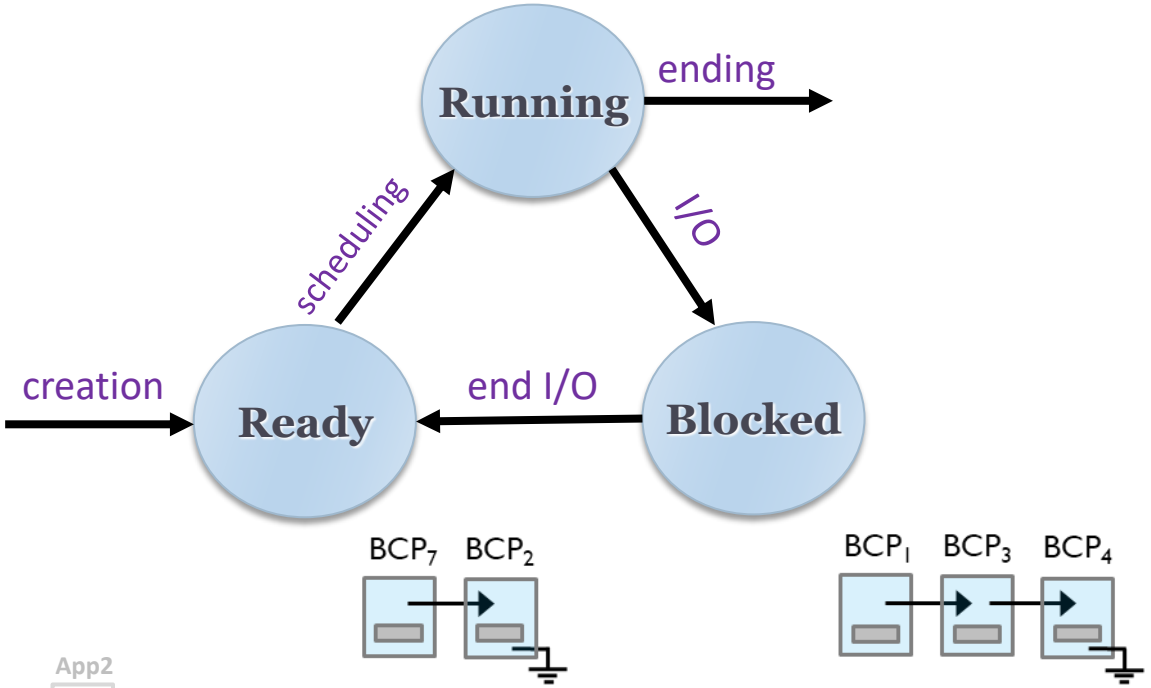


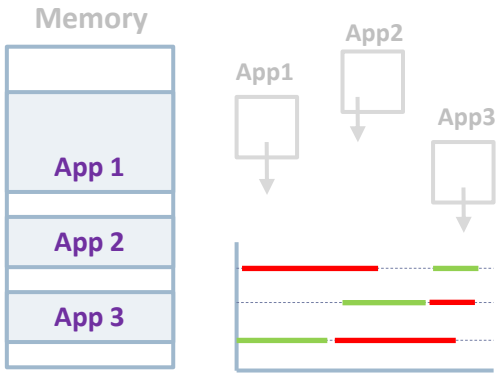
- Ready queue: processes waiting to run on CPU
- Queue blocked by resource: processes waiting to finish a blocking request to the associated resource
  - A process can only be in one queue (at most)

Multiprogramming (data)

Context of a process

- State
- List/Queue
- **Context**



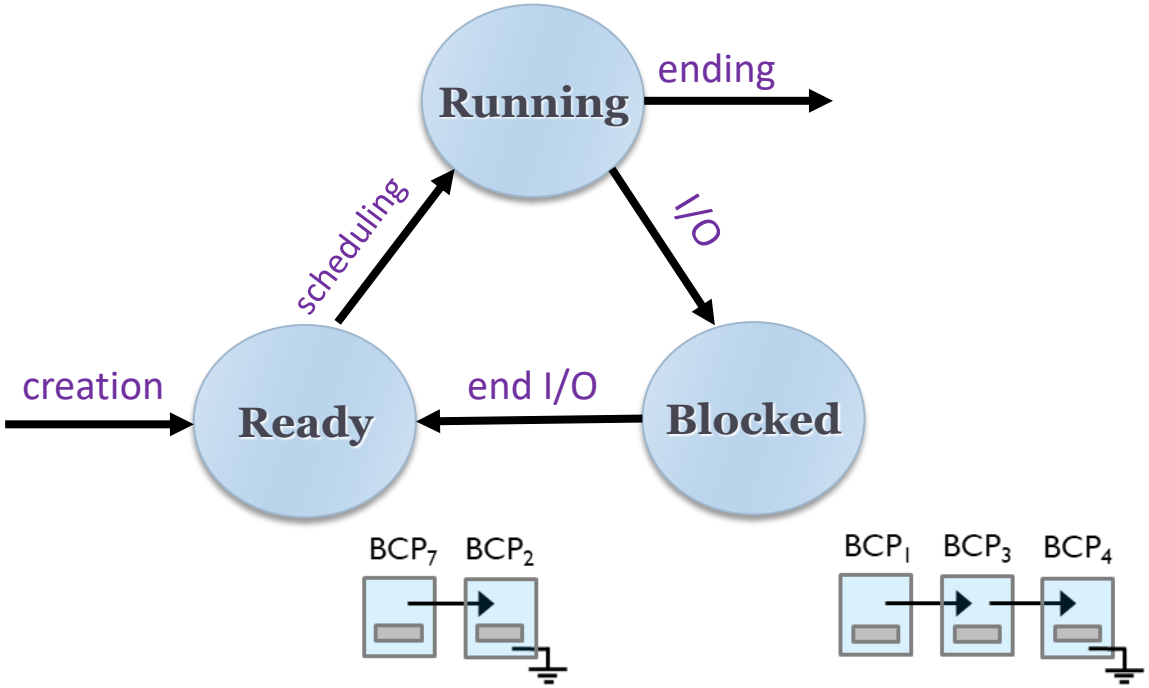


- Having several applications in memory
- **Context switching**:
  - If process is blocked then O.S. assigns CPU to another process (until blocked/terminated)

Multiprogramming (data)

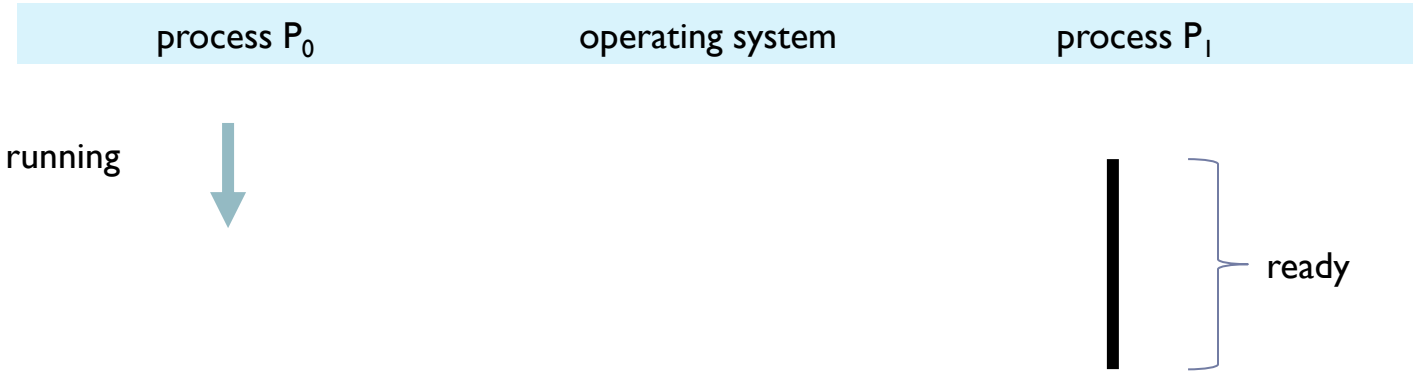
Context of a process

- State
- List/Queue
- **Context**

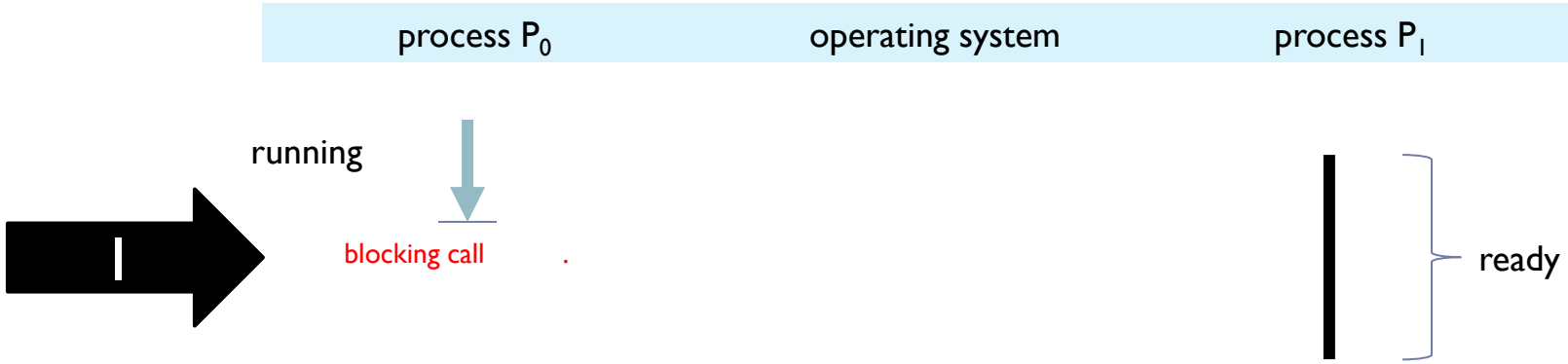


- General registers: PC, SR, etc.
- Specific registers: floating point registers, etc.
- References to resources: stack pointer, data pointer, etc.

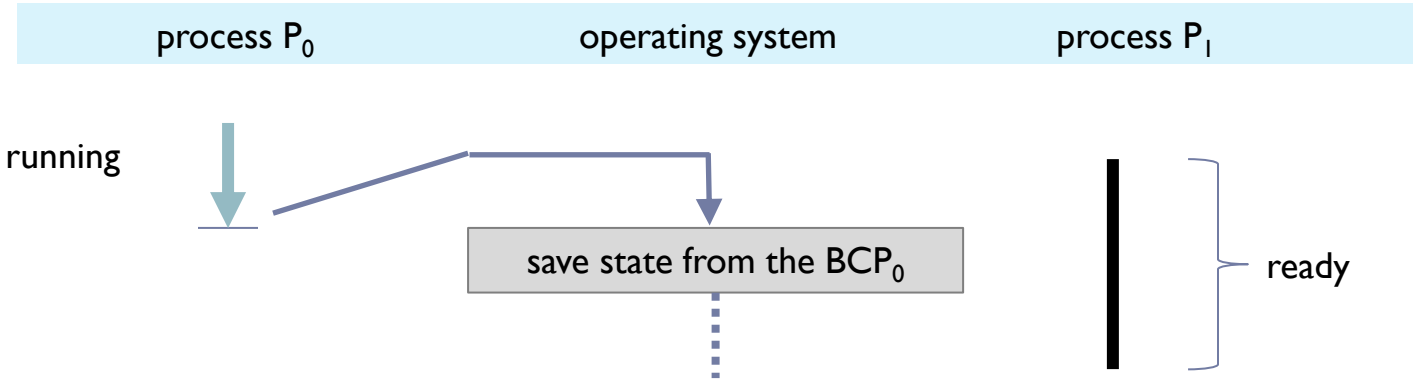
Multiprogramming: context switching



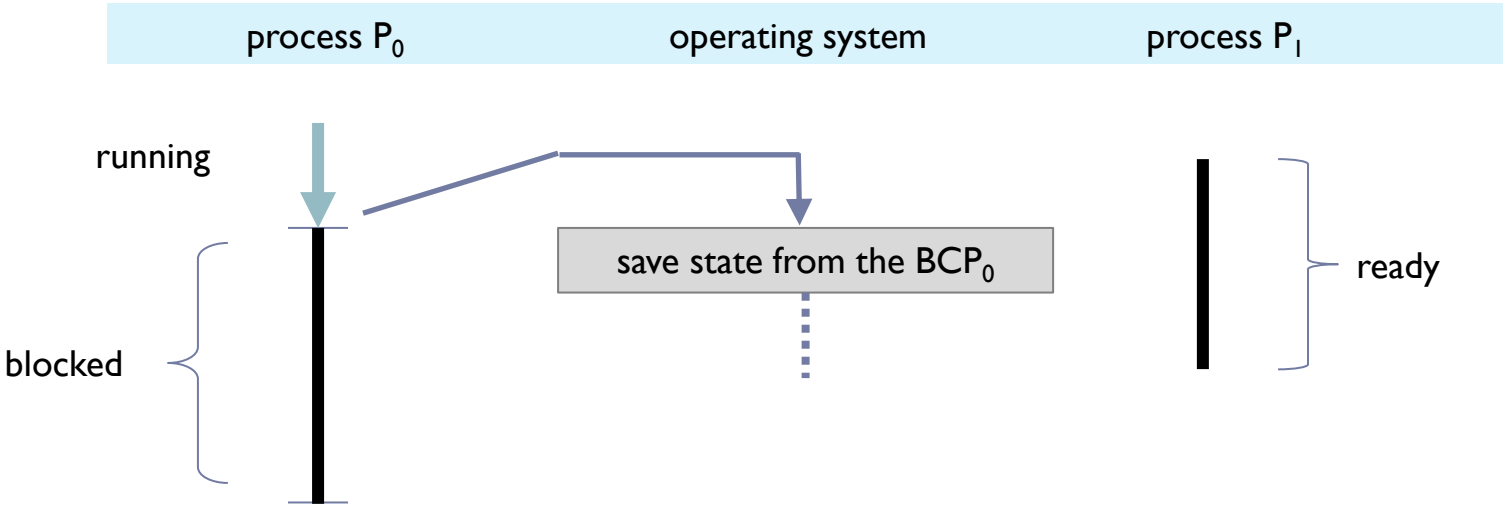
Multiprogramming: context switching



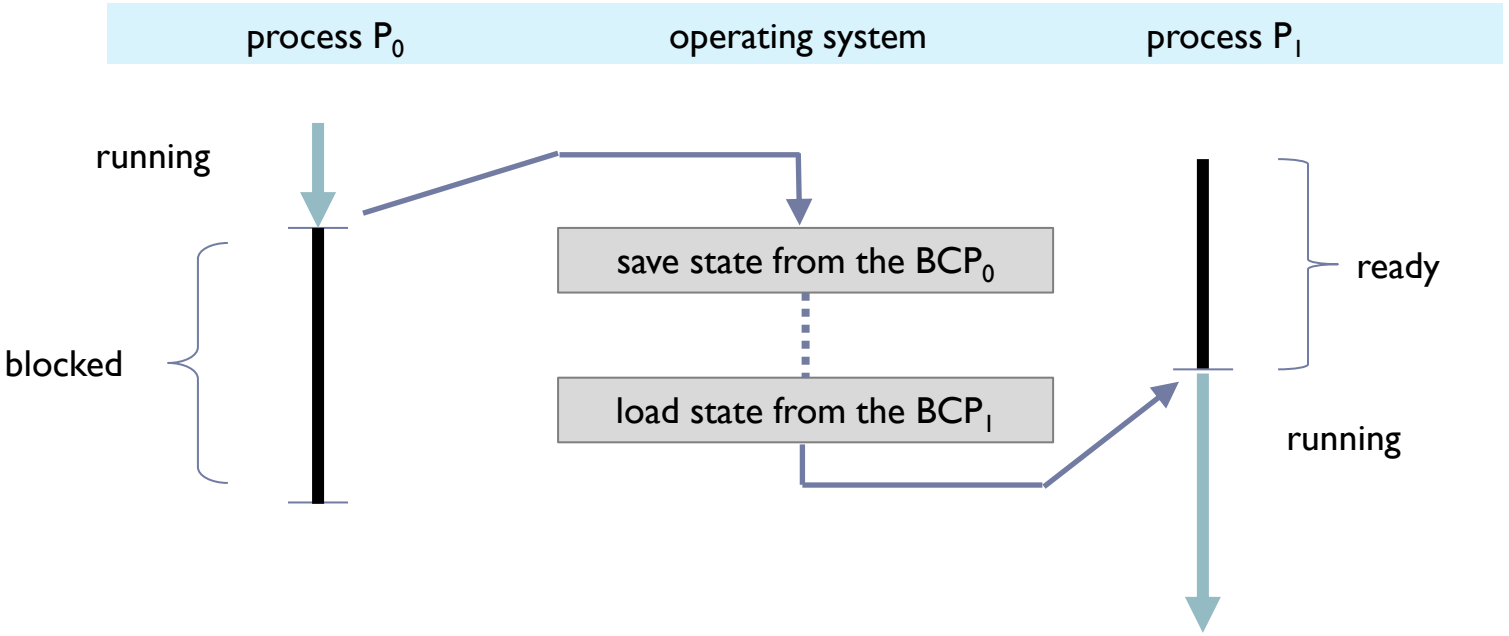
Multiprogramming: context switching



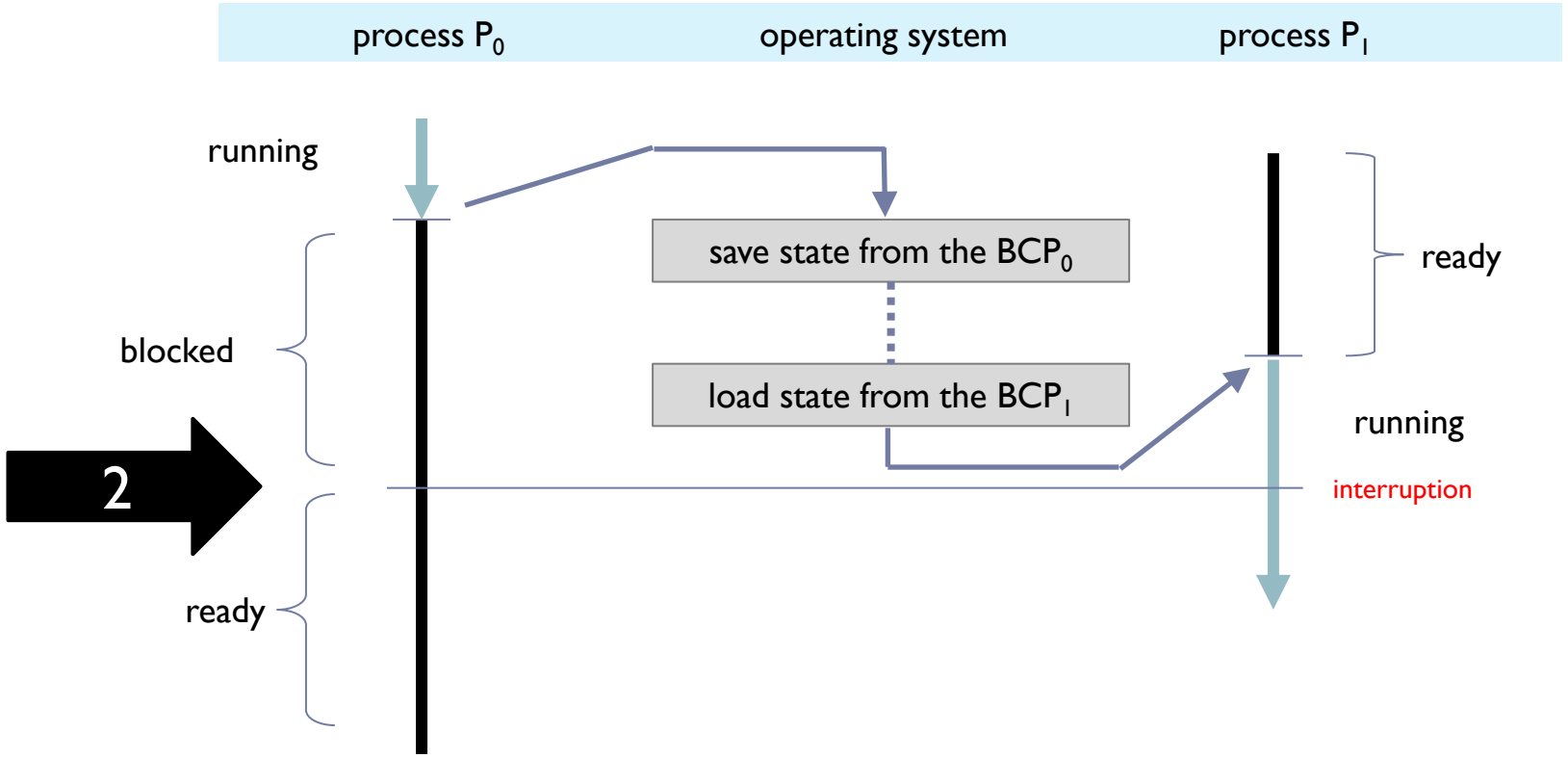
Multiprogramming: context switching



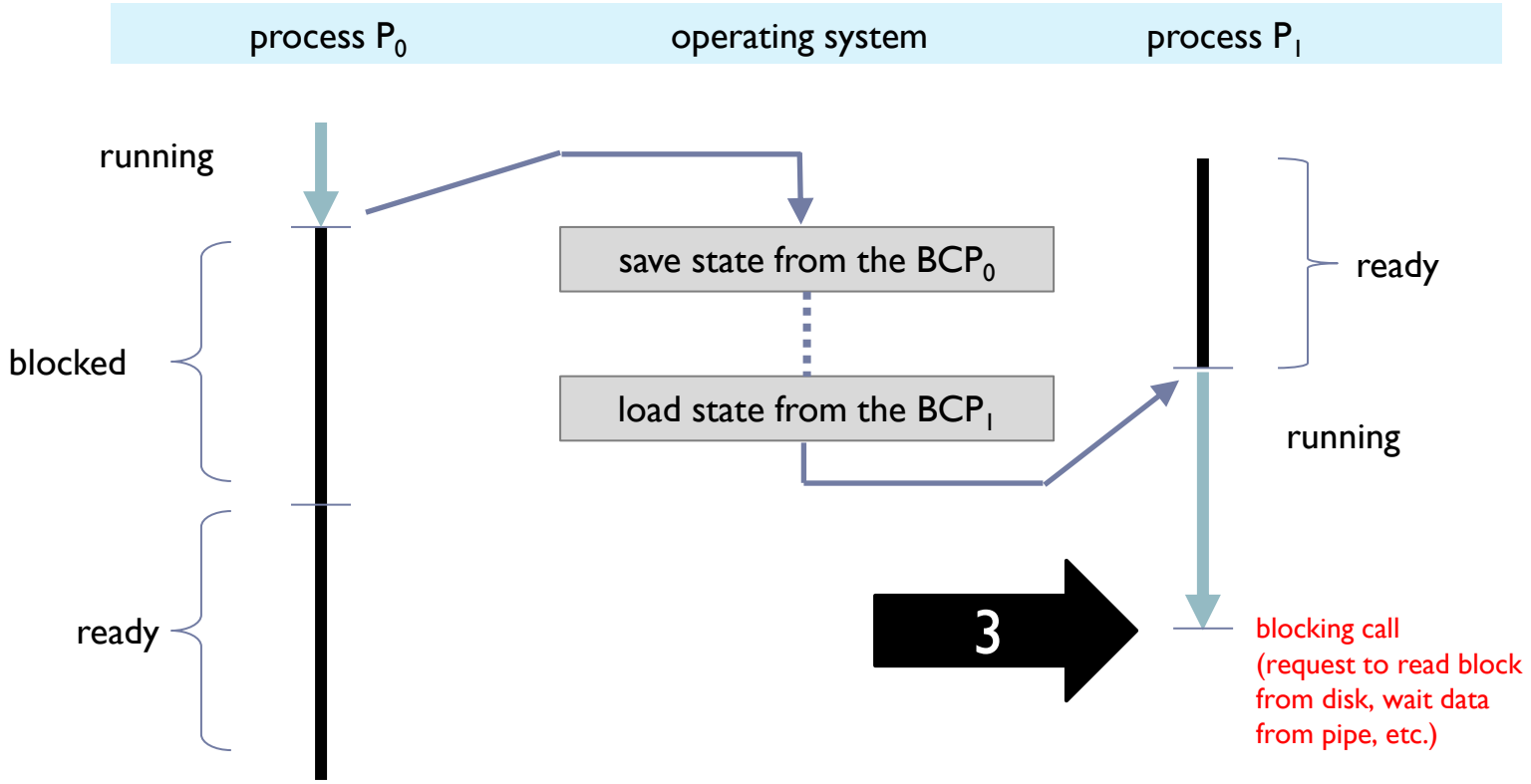
Multiprogramming: context switching



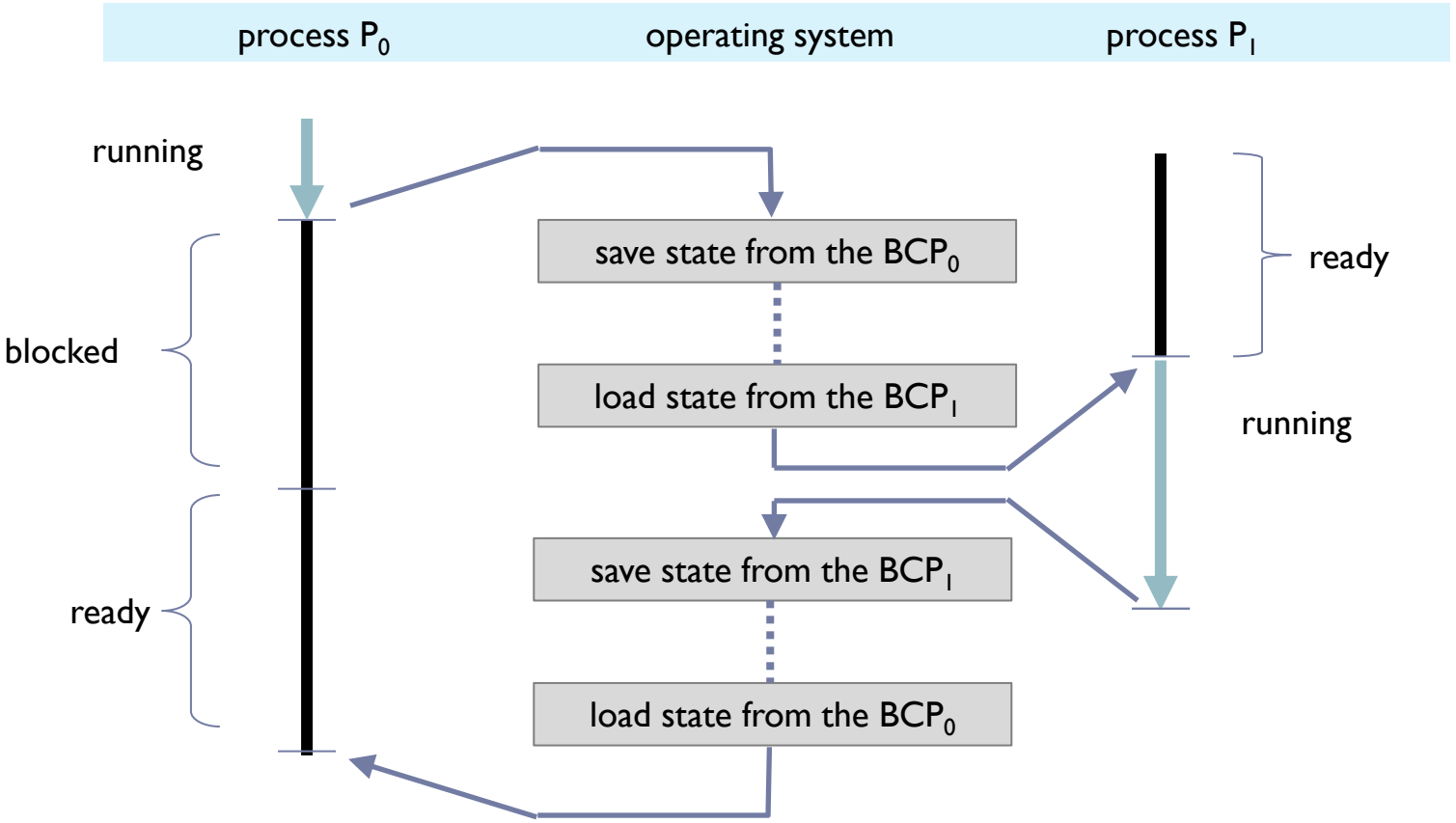
Multiprogramming: context switching



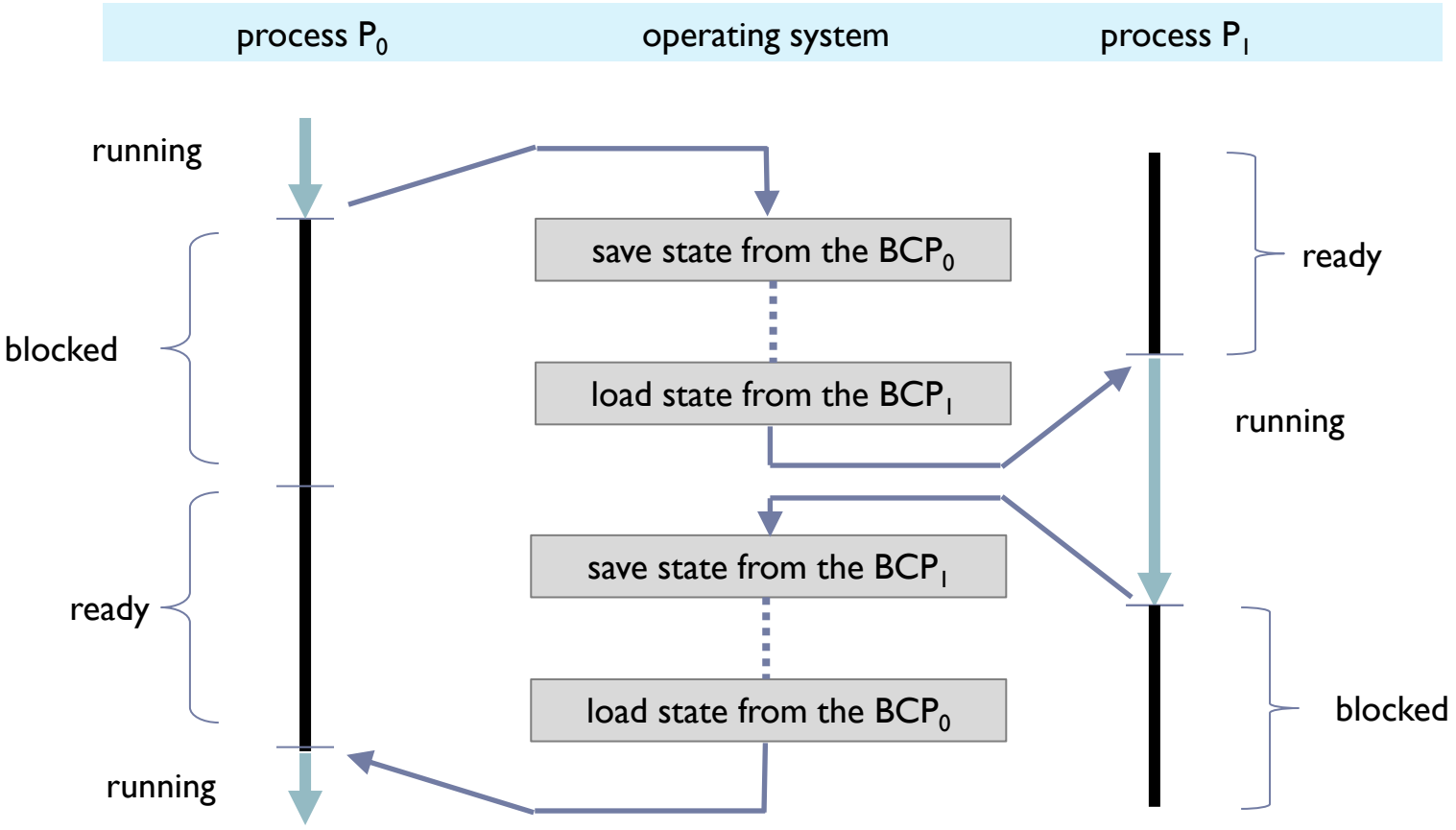
Multiprogramming: context switching



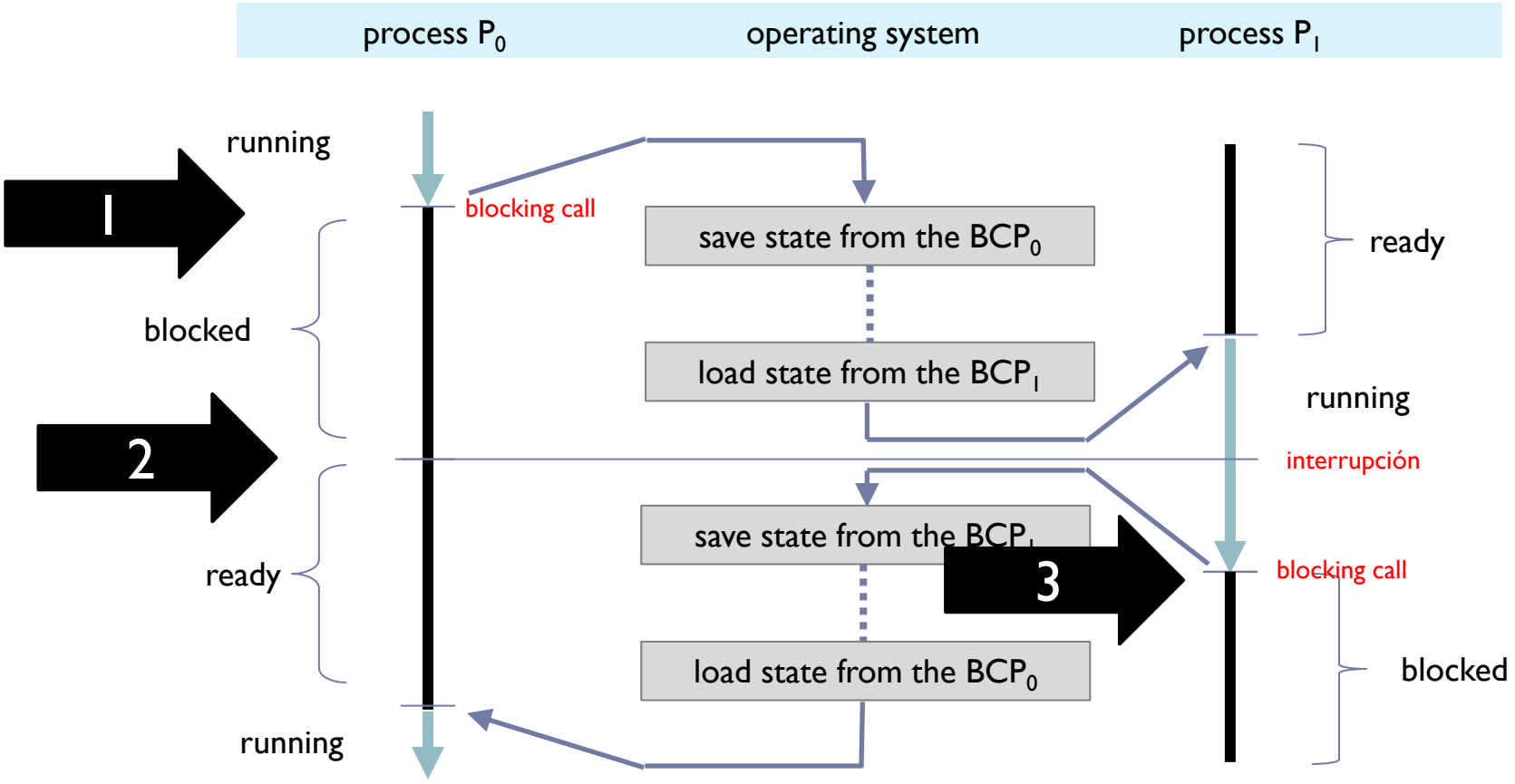
Multiprogramming: context switching



Multiprogramming: context switching



Multiprogramming: context switching



Example pseudocode (P0)

scheduler()

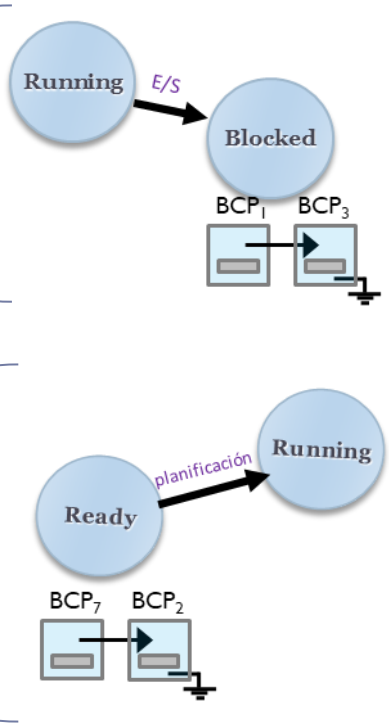
- return extract(CPU_Ready);

Keyboard_ReadKey()

- IF (isEmpty(Keyboard_Keys))
  - processActual->state = BLOCKED;
  - Insert(Keyboard_Processes, processActual);
  - process = processActual;
- processActual = scheduler();
- processActual->state = RUNNING;
- context_switching(&(process->context), &(processActual->context));
- return extract(Keyboard_Keys) ;

save state from BCP_0

load state to BCP_1



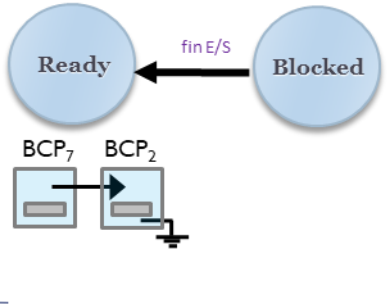
Example pseudocode (P1)

2

Keyboard_Hardware_Interrupt ()

- T = in (TECLADO_HW_ID);
- insert (T, Keyboard_Keys);

- process = first (Keyboard_Processes);
- IF (process != NULL)
  - remove (Keyboard_Processes);
  - process->state = READY;
  - insert (CPU_Ready, process);
- return ok;

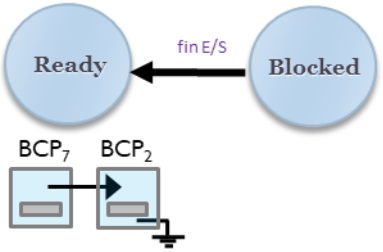


Example pseudocode (P1)

Keyboard_Hardware_Interrupt ()

- T = in (TECLADO_HW_ID);
- insert (T, Keyboard_Keys);

- process = first (Keyboard_Processes);
- IF (process != NULL)
  - remove (Keyboard_Processes);
  - process->state = READY;
  - insert (CPU_Ready, process);
- return ok;



- A process can only be in one queue (at the most):
  - [correct] remove + insert
  - [incorrect] insert + remove

Example pseudocode (P1)

3

scheduler()

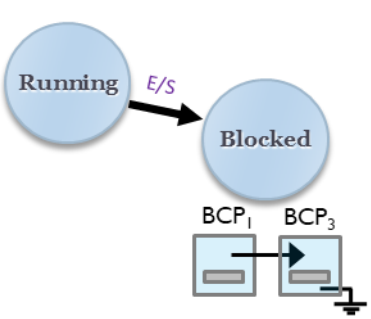
- return extract(CPU_Ready);

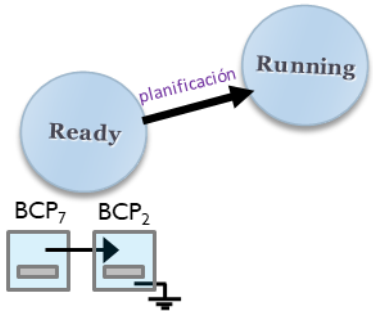
Disk_ReadBlockDisk()

- IF (no block in cache)
  - processActual->state = BLOCKED;
  - Insertar(Disk_Procesos, processActual);
  - process = processActual;
  - processActual = scheduler();
  - processActual->state = RUNNING;
  - context_switching(&(process->context), &(processActual->context));
- return extract(Disk_caché, block) ;

save state from BCP_1

load state to BCP_0





Example pseudocode (P0)

3

Disk_ReadBlockDisk()

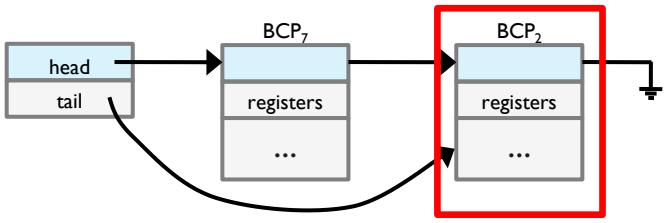
- IF (no block in cache)
  - processActual->state = BLOCKED;
  - Insertar(Disk_Procesos, processActual);
  - process = processActual;
  - processActual = scheduler();
  - processActual->state = RUNNING;
  - context_switching(&(process->context), &(processActual->context));
- return extract(Disk_caché, block) ;

Keyboard_ReadKey()

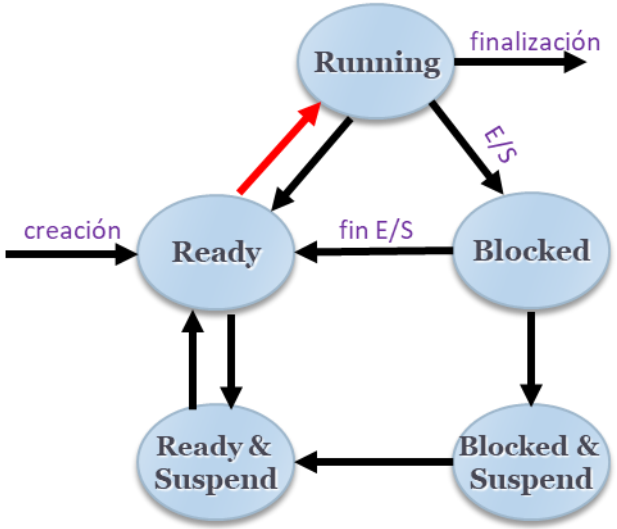
- IF (there is no key)
  - processActual->state = BLOCKED;
  - Insertar(Keyboard_Processes, processActual);
  - process = processActual;
  - processActual = scheduler();
  - processActual->state = RUNNING;
  - context_switching(&(process->context), &(processActual->context));
- return extract(Keyboard_Keys) ;

Scheduler and activator

- Scheduler:
  Selects the process to be executed from those ready to be executed



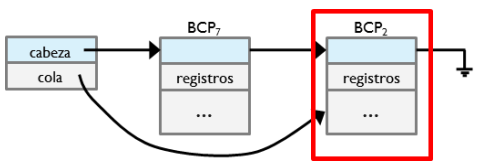
- Activator:
  Gives control to the process that the scheduler has selected (context switch - restore)

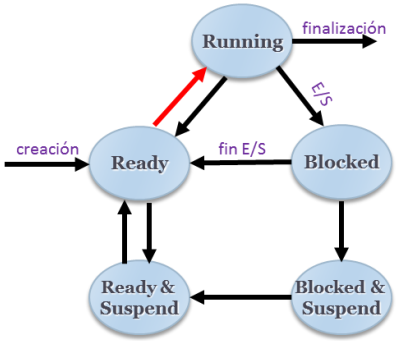


Scheduler and activator

scheduler()

- return extract(CPU_Ready);





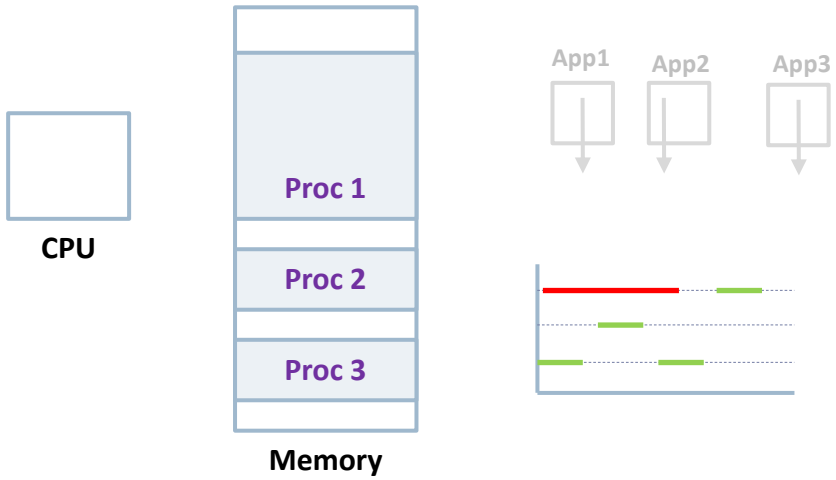
Keyboard_ReadKey()

- IF (there is no key)
  - processActual->state = BLOCKED;
  - Insert (Keyboard_Processes, processActual);
  - process = processActual;
  - processActual = **scheduler**();
  - processActual->state = RUNNING;
  - **activator** (&(process->context), &(processActual->context));
- return extract(Keyboard_Keys) ;

Model offered

recap

- resources
- multiprogramming
  - protection/sharing
  - process hierarchy
- **multitasking**
- multiprocessing



- Multitasking
  - Each process is executed for a quantum of time (e.g., 5 ms) and the turn is rotated to execute non-blocked processes
    - Involuntary context switching (I.C.S.)
  - Sharing of processor usage
    - Everything seems to run at the same time

Multitarea (datos y funciones)

Requisitos	Information (in data structures)	Functions (internal, service and API)
Multiprogramming	• State of execution • Context: CPU registers… • List of processes	• Hw/Sw devices Int. • Scheduler • Create/Destroy/Schedule process
Multiprogramming	• State of execution • Context: registros de CPU… • List of processes	• Hw/Sw devices Int. • Scheduler • Create/Destroy/Schedule process

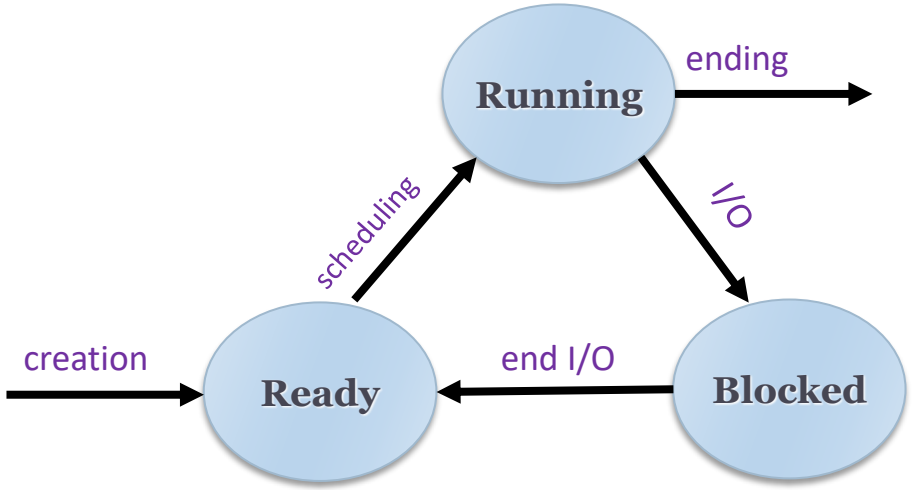
○ Protection / Sharing	• Message passing
	  • Receive message queue
	• Shared memory
	  • Zones, locks and conditions
		• Message sending/receiving and message tail management
		• Concurrency API and data structure management

○ Process hierarchy	• Family relationship • Sets of related processes • Processes of the same session	• Clone/Change process image • Associate processes and indicate representative process
Multitasking	• Remaining quantum • Priority	• Clock hw/sw int. • Scheduler • Create/Destroy/Schedule process
Multiprocessing	• Afinity	• Clock hw/sw int. • Scheduler • Create/Destroy/Schedule process

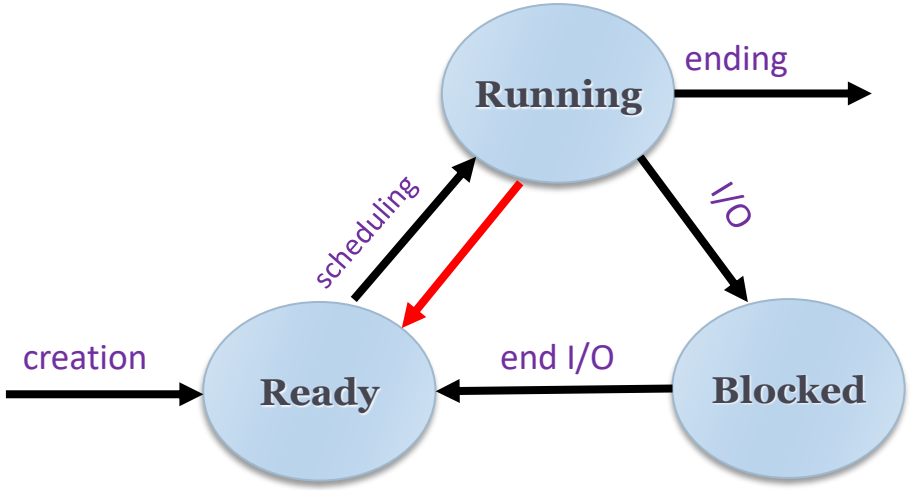
States of a process

- State
- List/Queue
- Context

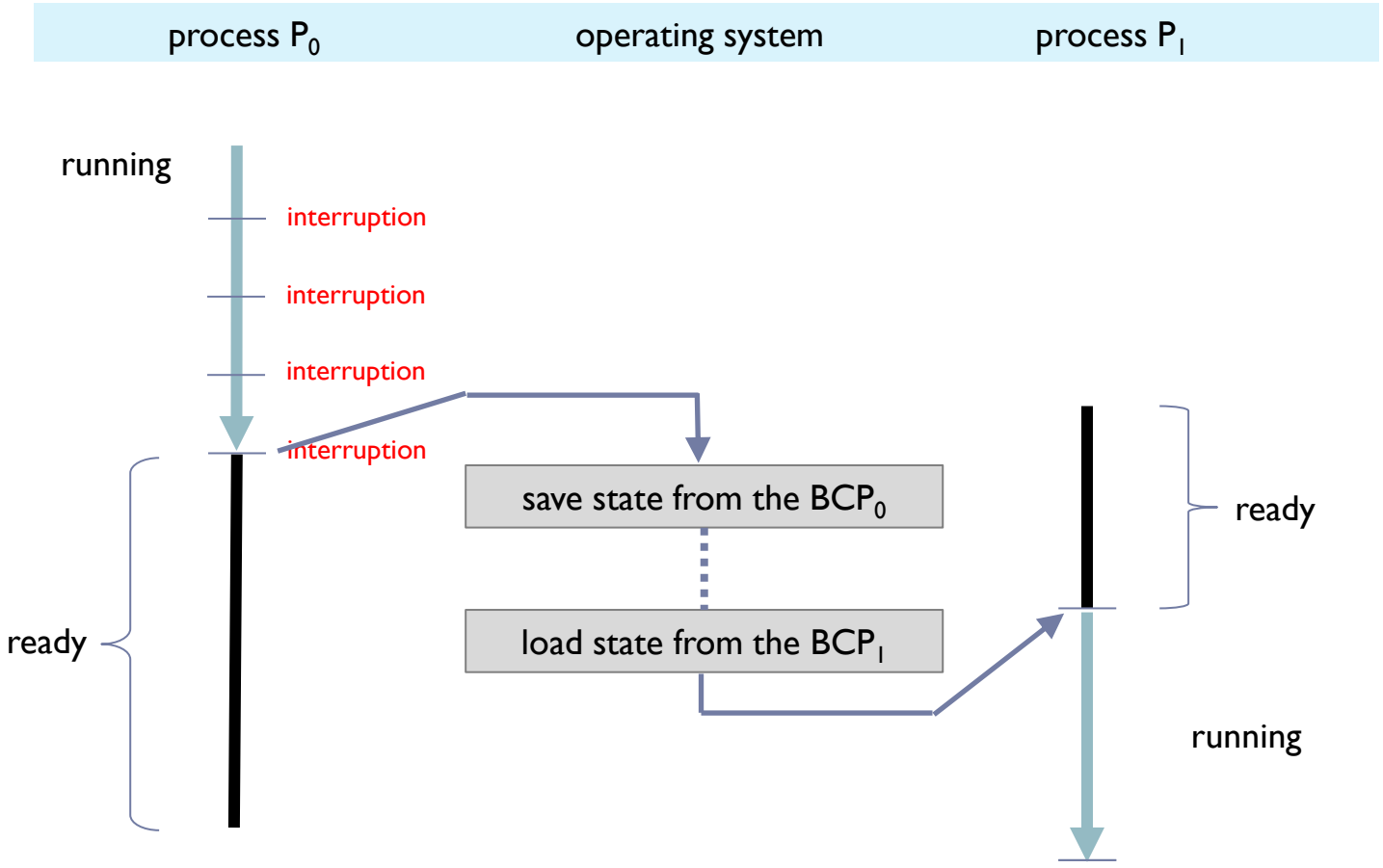
V.C.S.



V.C.S. + I.C.S.



Clock: handling with V.C.S. + I.C.S.

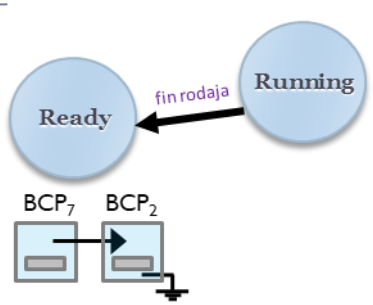


Example pseudocode (P0)

Clock_Hardware_Interrupt()

- Ticks++;

- pActual->slice = pActual->slice - 1;
- IF (pActual->slice == 0)
  - pActual->state = READY;
  - pActual->slice = SLICE;
  - insert (CPU_Ready, pActual);
  - process = pActual;



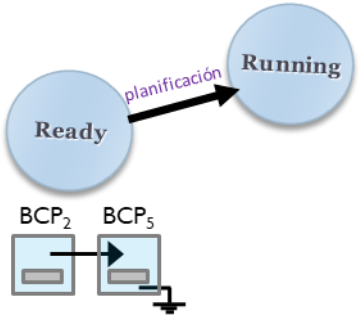
scheduler()

- return extract(CPU_Ready);

- pActual = scheduler();
- pActual->state = RUNNING;
- context_switching (
  &(process->context),
  &(pActual->context));

save state from BCP_0

load state to BCP_I



- return ok;

Types of context switching
summary

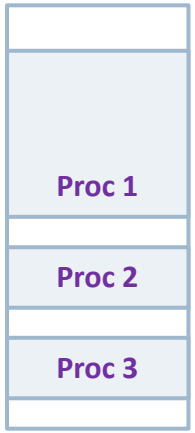
- ***Voluntary* context switching (V.C.S.):**
  - Process makes a system call (or produces an exception such as a page fault) that involves waiting for an event.
  - **Transition:** *Running* → B*locked.*
  - **Scenarios:** read from keyboard, page-fault, etc.
  - **Reason:** *Efficiency in processor usage*
- ***Involuntary* context switching (I.C.S.):**
  - OS removes the process from CPU
  - **Transition:** *Running* → R*eady*
  - **Scenarios:** end of slice/quantum or other process of higher priority goes to Ready state
  - **Reason:** *distribution of the process utilization*

Model offered

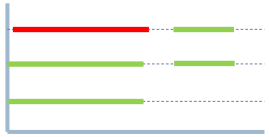
recap

- resources
- multiprogramming
  - protection/sharing
  - process hierarchy
- multitasking
- **multiprocessing**

CPU



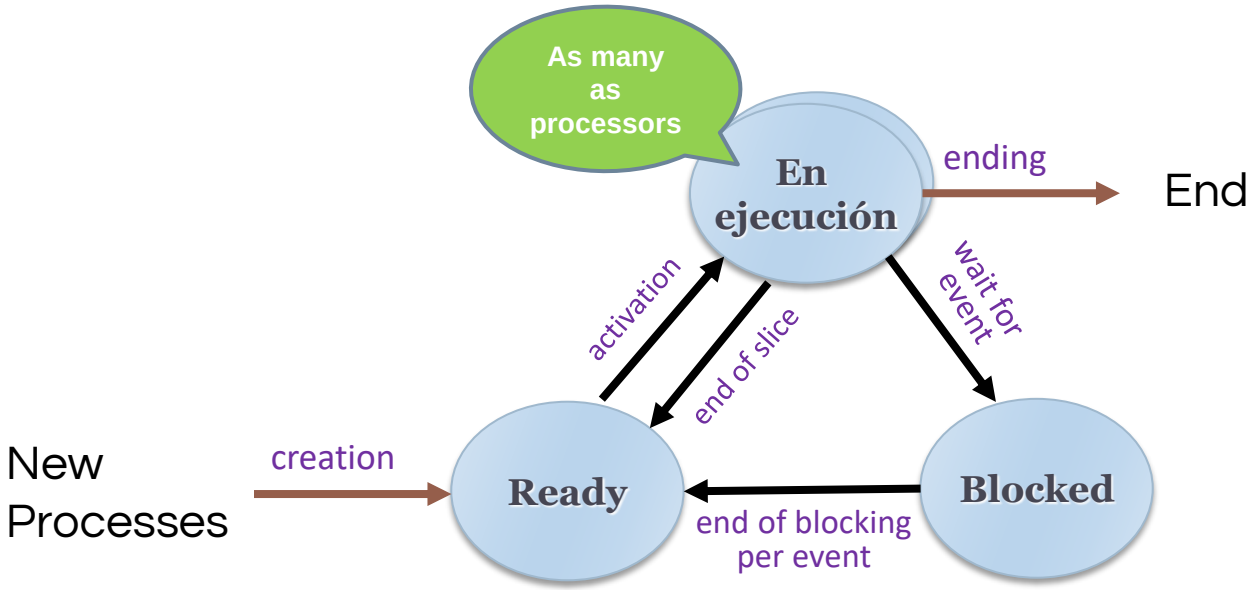
Memory



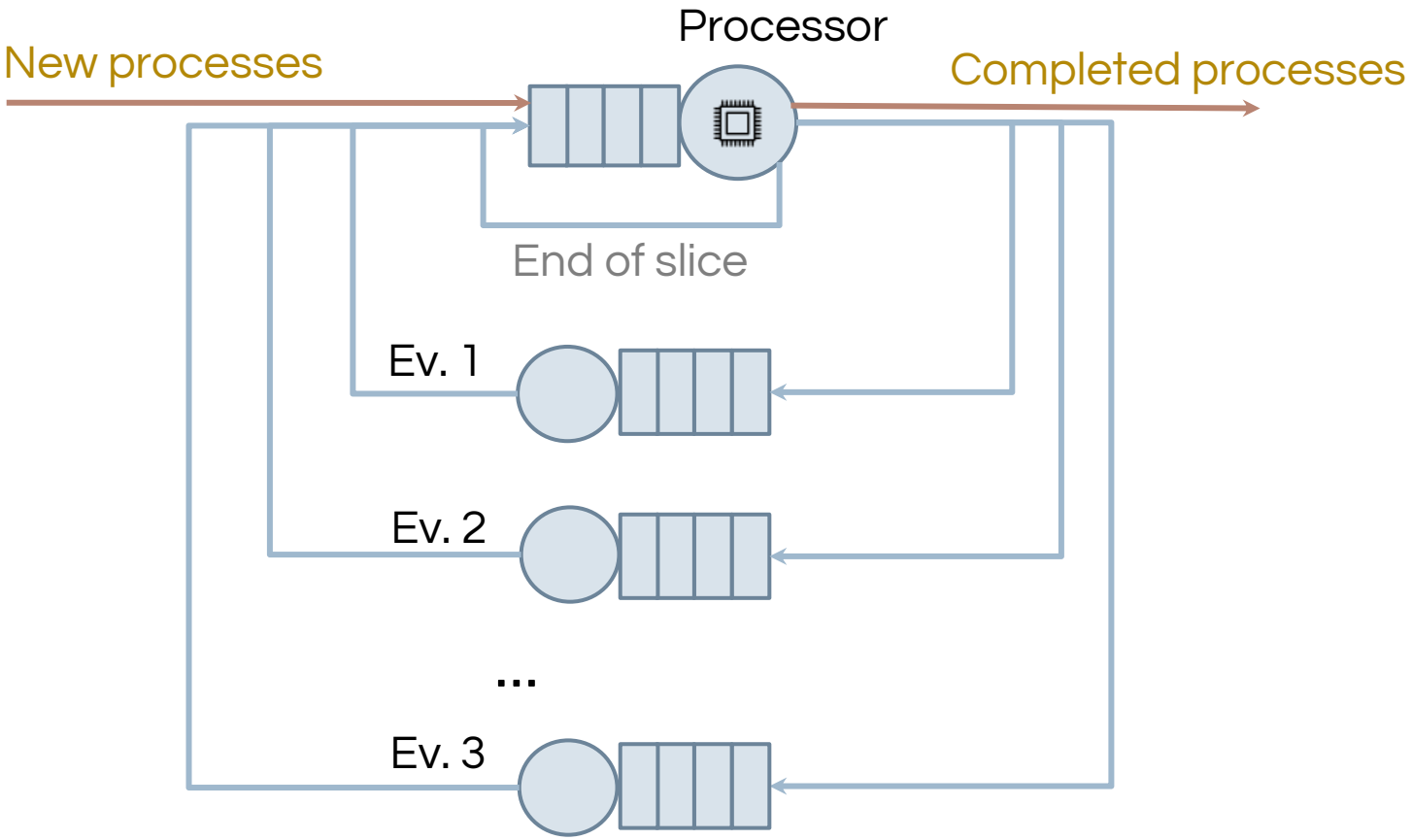
- Multiprocessing
  - Several processors are available (multicore/multiprocessor)
  - In addition to the distribution of each CPU (multitasking) there is real parallelism between several tasks (as many as processors)
    - Scheduler and separate data structures per processor are usually used with some load balancing mechanism

- **State**
- List/Queue
- Context

Basic lifecycle of a process

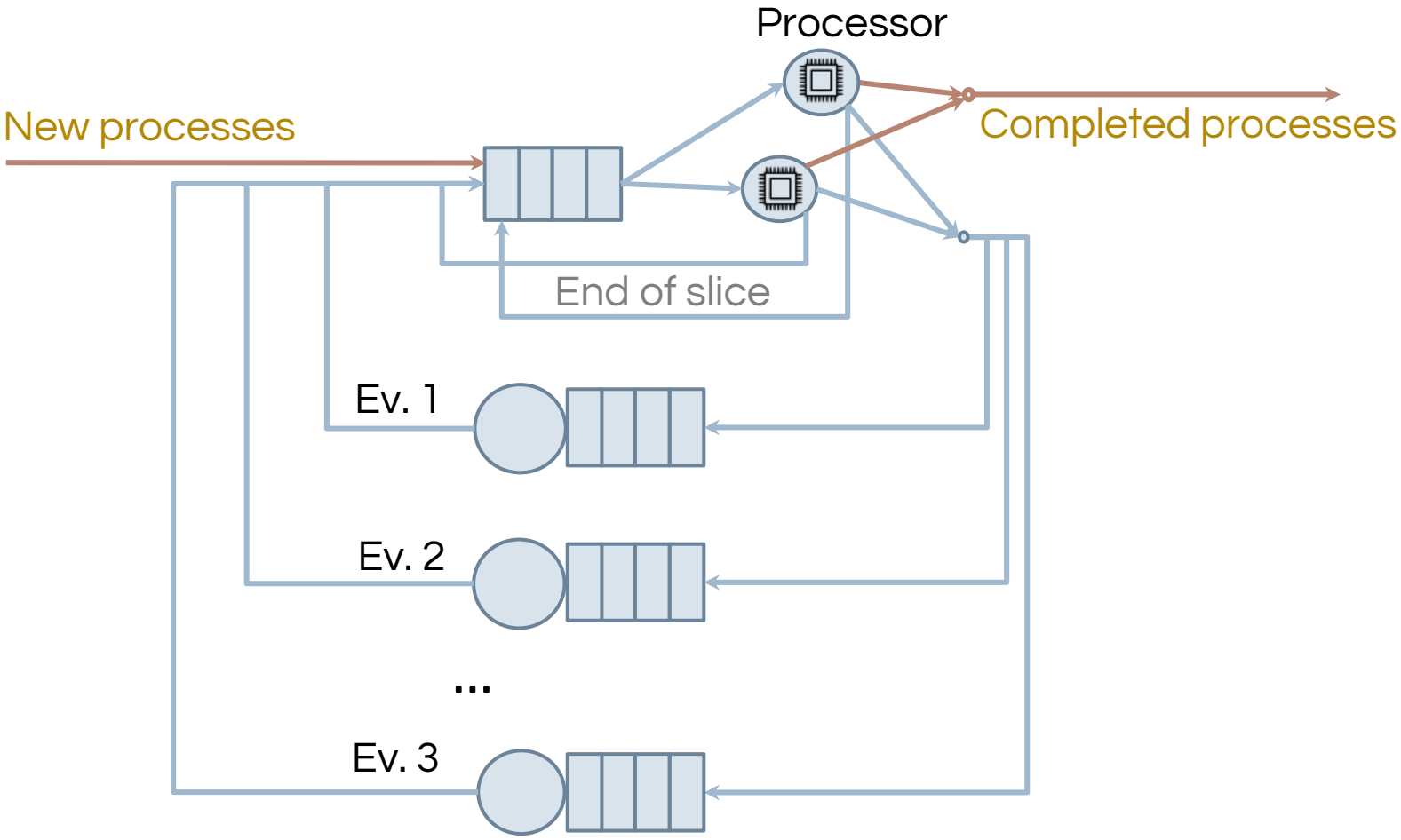


Simplified queuing model: 1 processor



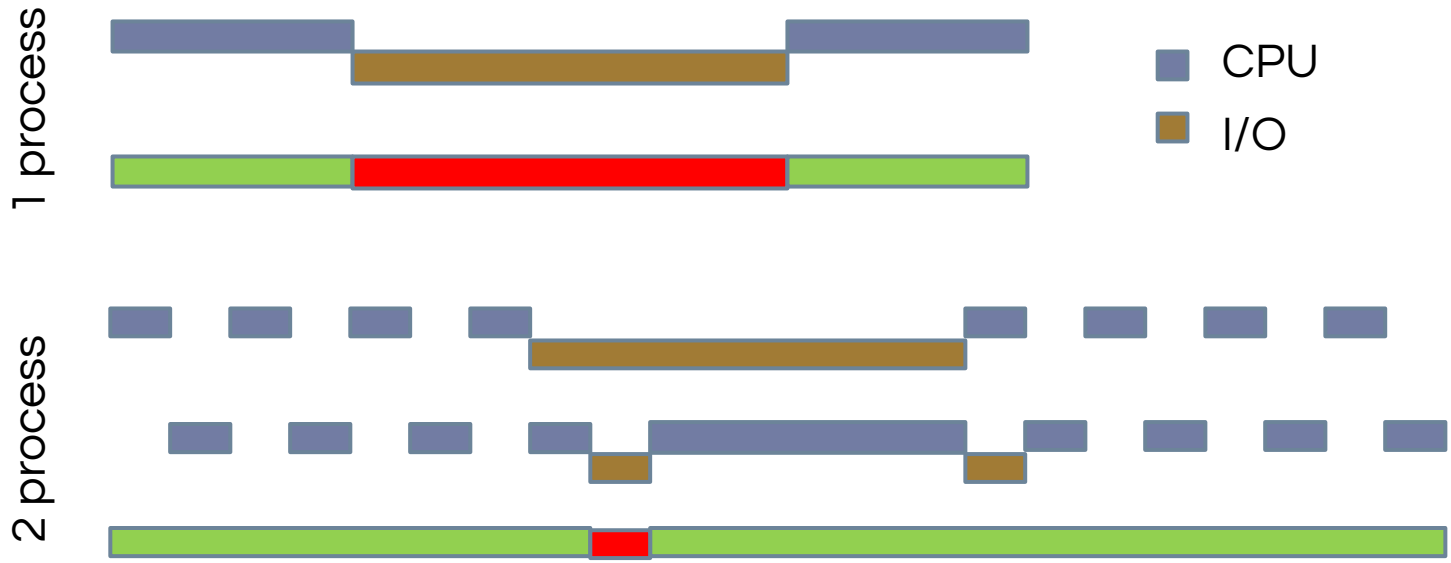
Sistemas operativos: una visión aplicada (© J. Carrete et al.)
https://iconarchive.com/show/outline-icons-by-iconsmind/CPU-icon.html

Simplified queuing model: N processors

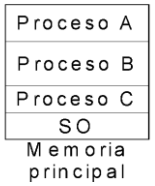


Multitasking advantages

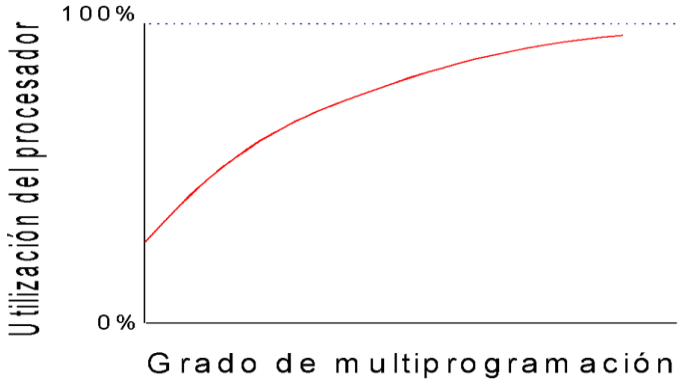
- Modularity: facilitates programming by dividing programs into processes.
- Allows simultaneous interactive service of N users in an efficient way.
- Exploits the time that processes spend waiting for their I/O operations to complete.
- Increases CPU utilization

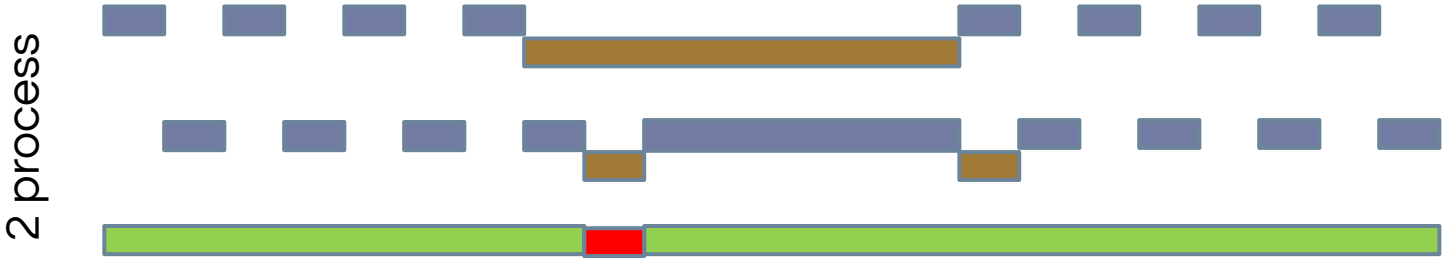


Multitasking advantages

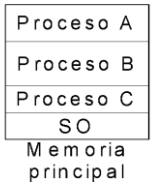


- CPU utilization... depends on degree of multiprogramming
- Degree of multiprogramming: nº of active processes.
- Does more processes always improve the % of CPU utilization?

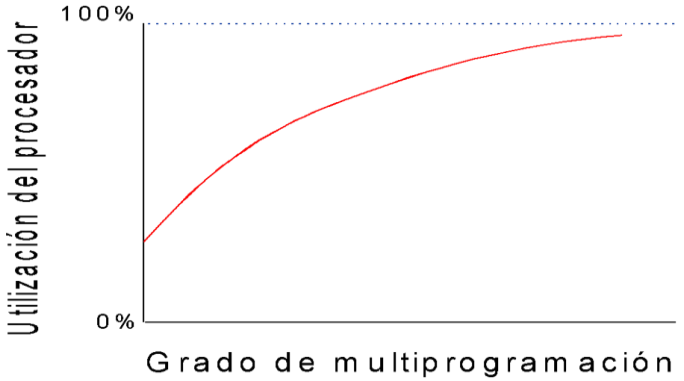




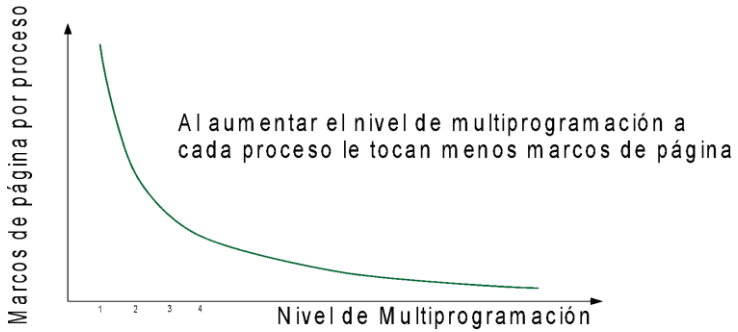
Multiprogramming and memory



- Systems **without** virtual memory:
  - Each process resides entirely in M.M.

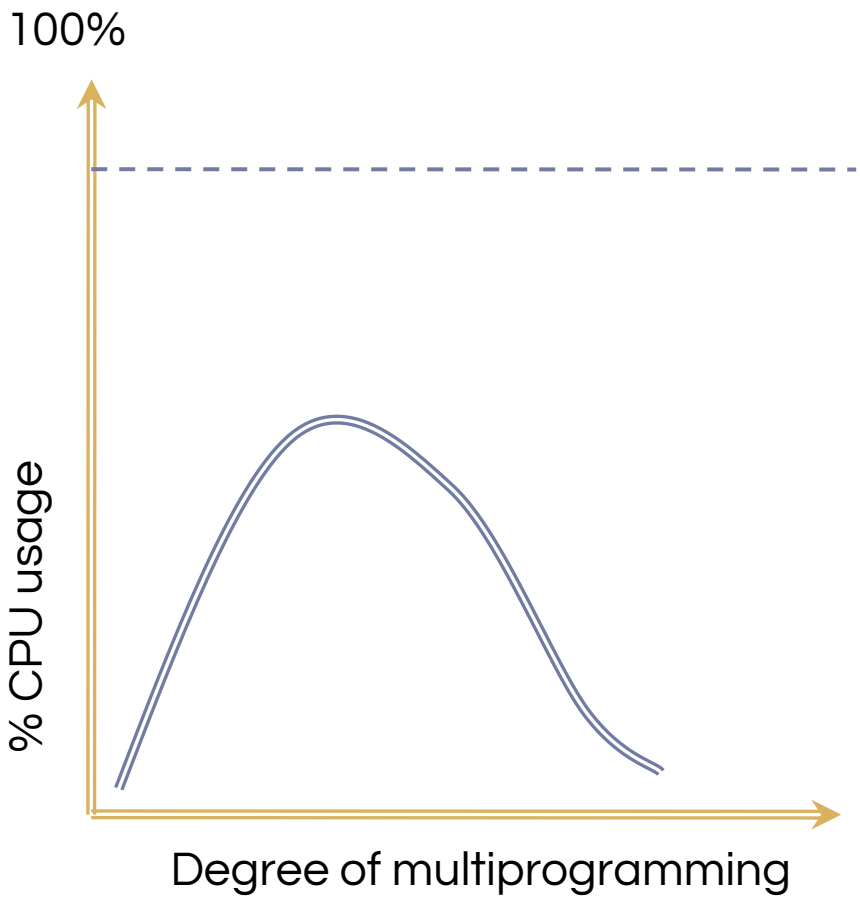


- Systems **with** virtual memory:
  - They divide the addressable space of the processes into pages.
  - They divide the main physical memory into page frames.
  - At any given time each process has a certain number of its pages in main memory (**resident set**).



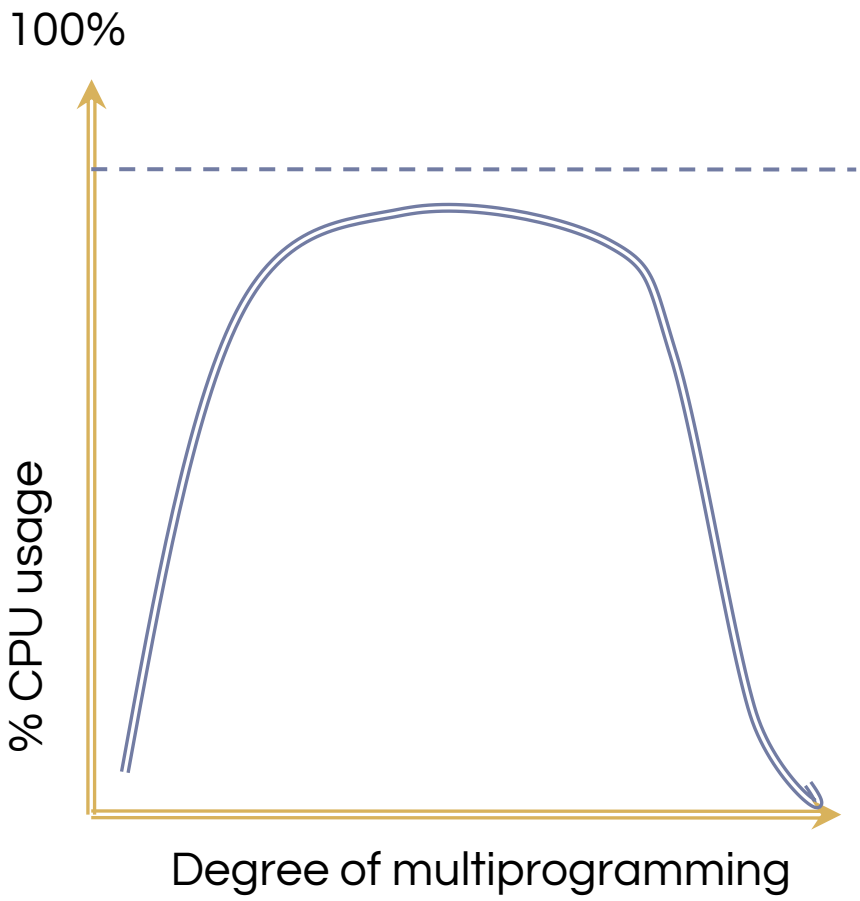
Performance: Small amount of physical memory

- As the degree of multiprogramming increases:
  - The size of the resident set of each process decreases.
- Low memory: hyper paging occurs before a high percentage of CPU usage can be reached.
- **Problem**: Lack of memory
  Solution: Main memory expansion.



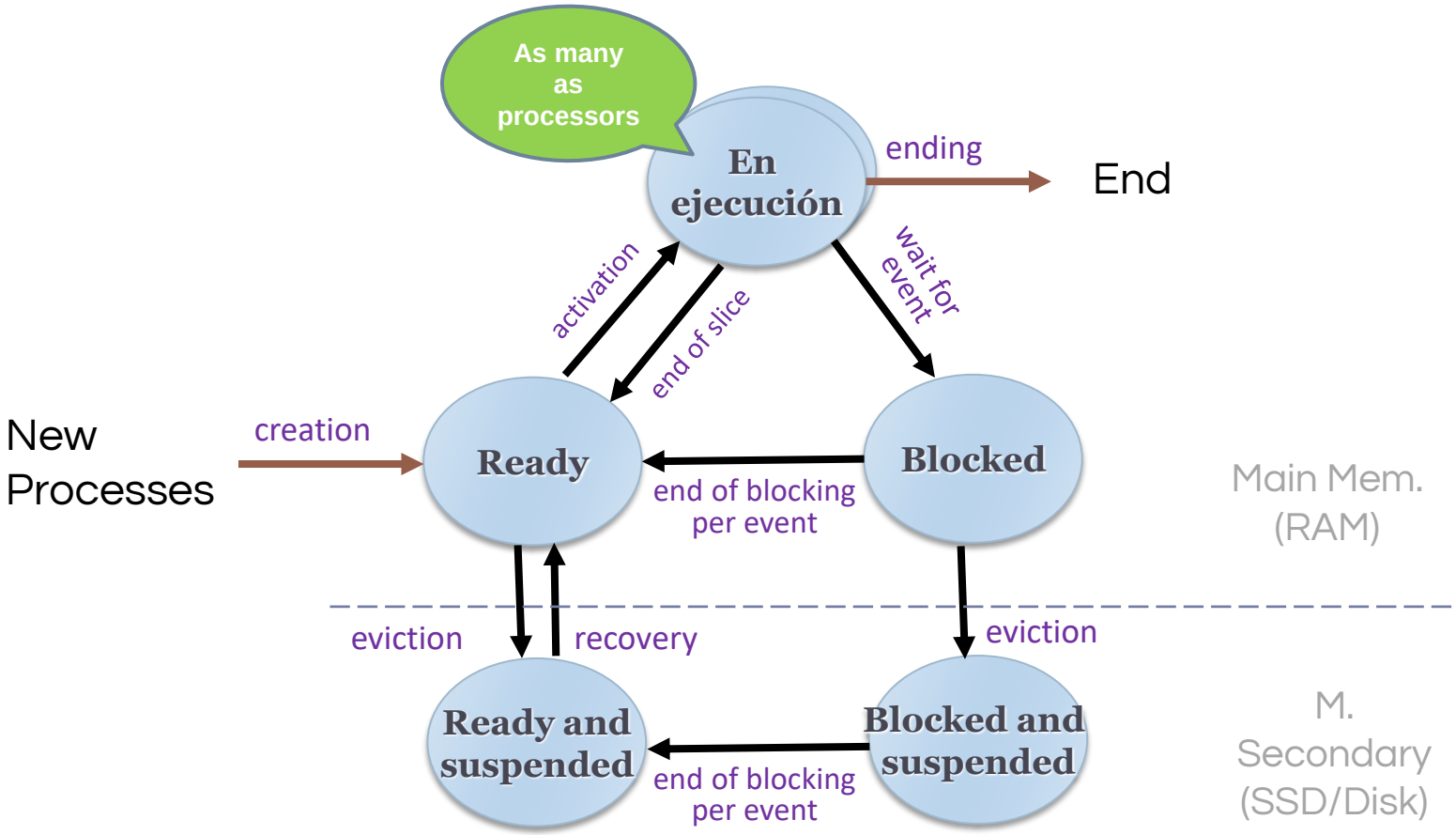
Performance: Large amount of physical memory

- As the degree of multiprogramming increases:
  - The size of the resident set of each process decreases.
- Too much memory: a high % of CPU usage is achieved with fewer processes than can fit in memory.
- **Problem**: "too much" memory. **Solution**: Processor upgrade or addition of more processors.



Basic lifecycle of a process

- State
- List/Queue
- Context

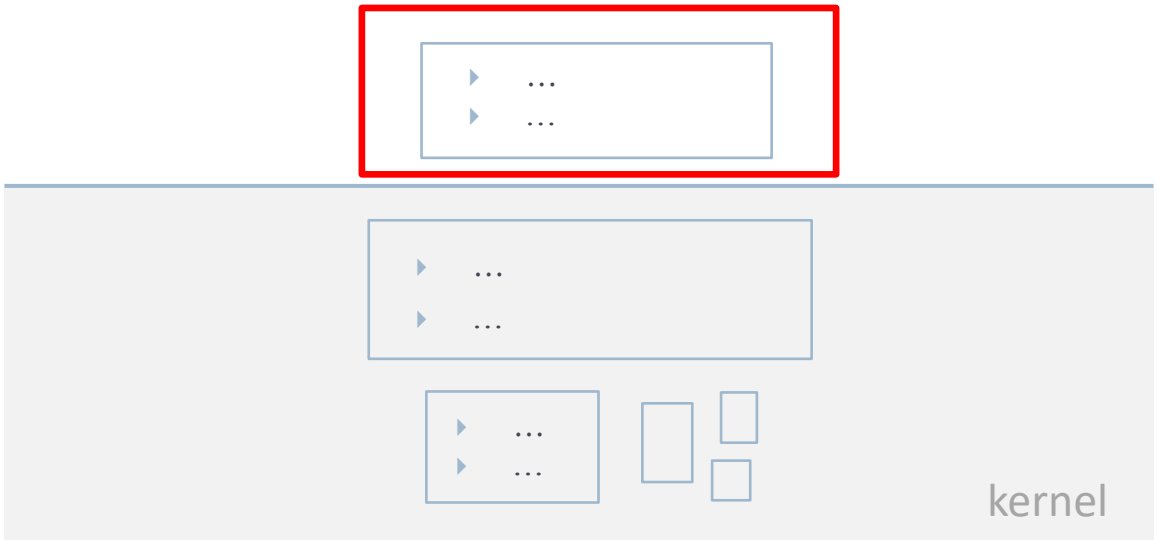


- The OS can totally eject processes to the swap if % CPU usage drops due to hyperpagination.
- Requires new states: blocked and suspended + ready and suspended.

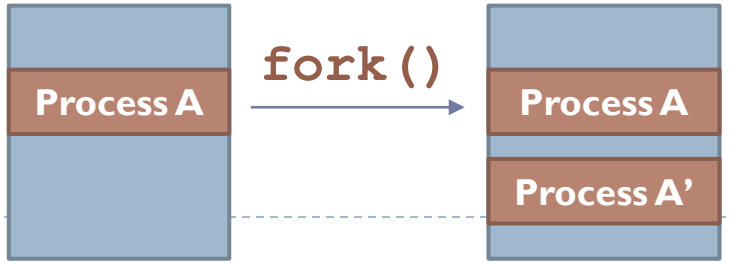
Contents

1. Introduction
   - Process definition.
   - Model offered: resources, multiprogramming, multitasking and multiprocessing
2. Process life cycle: process status.
3. **Services to manage processes provided by the operating system.**
4. Definition of thread
5. Kernel and library threads.
6. Services for threads in the operating system.

Operating system services
POSIX process management services



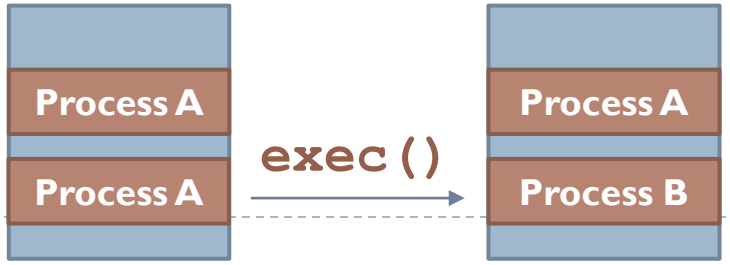
Service: fork



Service	`#include <unistd.h>`
	
	`pid_t fork(void);`

Arguments	
Returns	▫ -1 in case of error. ▫ In the parent process: the identifier of the child process. ▫ In the child process: 0
Description	▫ Duplicates the process that invokes the call. ▫ The parent and child processes keep running the same program. ▫ The child process inherits the open files from the parent process. ▫ Open file descriptors are copied. ▫ Pending alarms are deactivated.

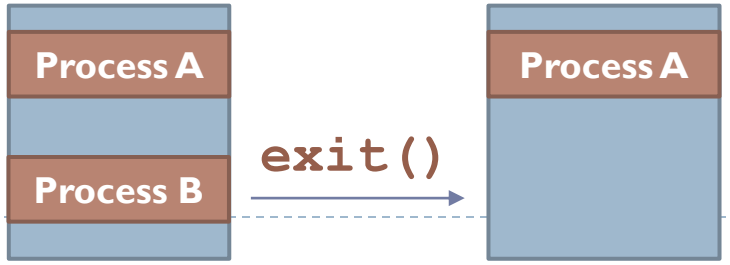
Service: exec



Service	`#include <unistd.h>`
	`int execl(const char *path, const char *arg, ...);`
	`int execv(const char* path, char* const argv[]);`
	`int execve(const char* path, char* const argv[], char* const envp[]);`
	`int execvp(const char *file, char *const argv[]);`

Arguments	▫ path: Path to executable file. ▫ file: Searches for the executable file in all directories specified by PATH
Returns	▫ Returns -1 in case of error, **otherwise it does not return.**
Description	▫ Changes the image of the current process. ▫ The same process executes another program. ▫ Open files remain open. ▫ Signals with the default action will continue by default, signals with handler will take the default action.

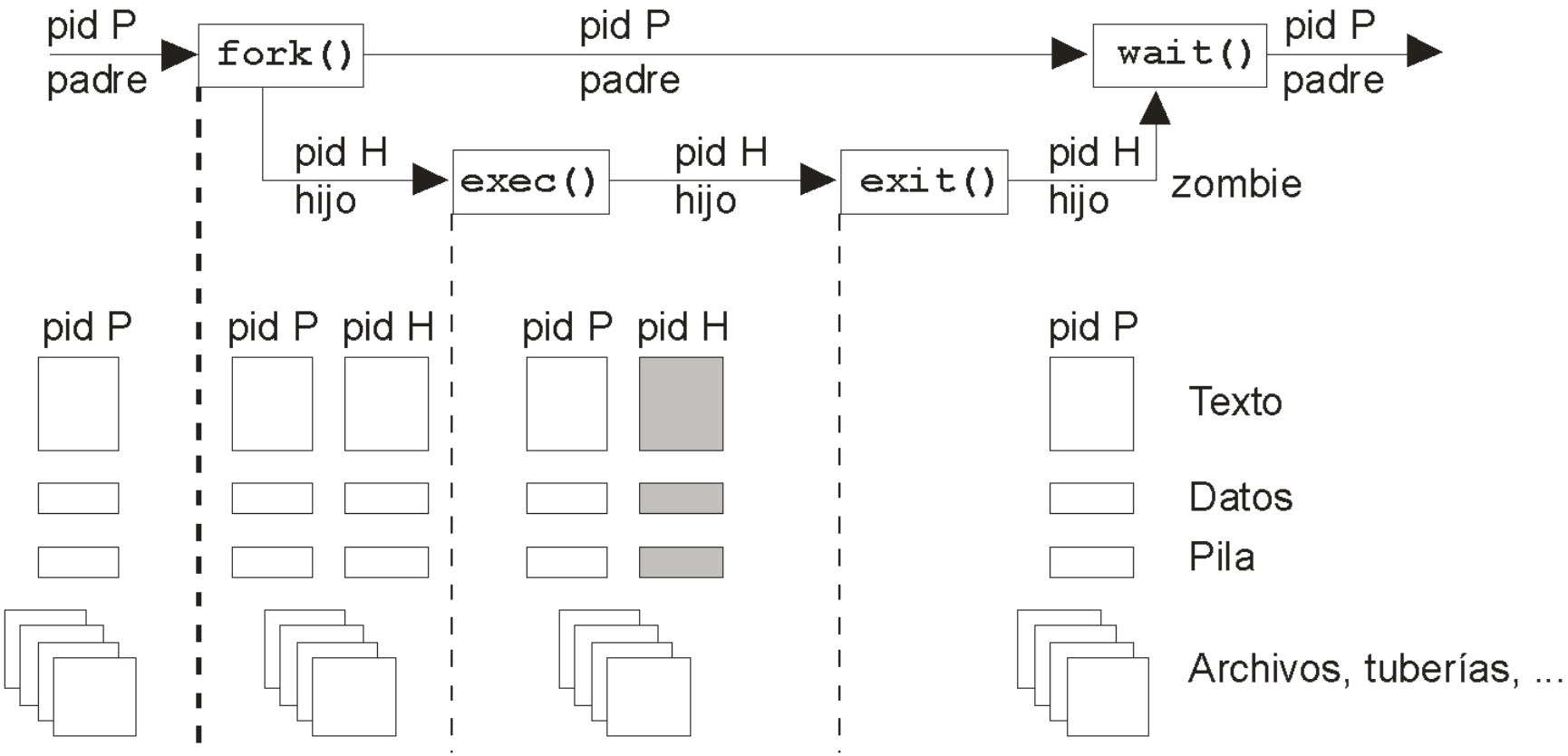
Service: exit



Service	`#include <unistd.h>`
	
	`void exit(status);`

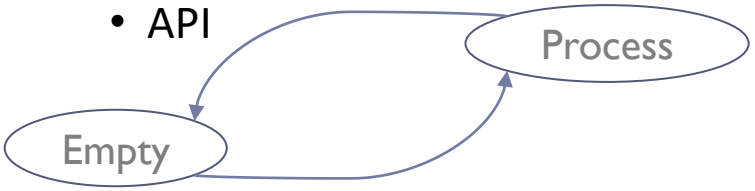
Arguments	▫ status: value retrieved by the parent in the wait() call
Returns	
Description	▫ The execution of the process ends. ▫ All open file descriptors are closed. ▫ All the resources of the process are released. ▫ The PCB of the process is released.

Using fork, exec, wait and exit services

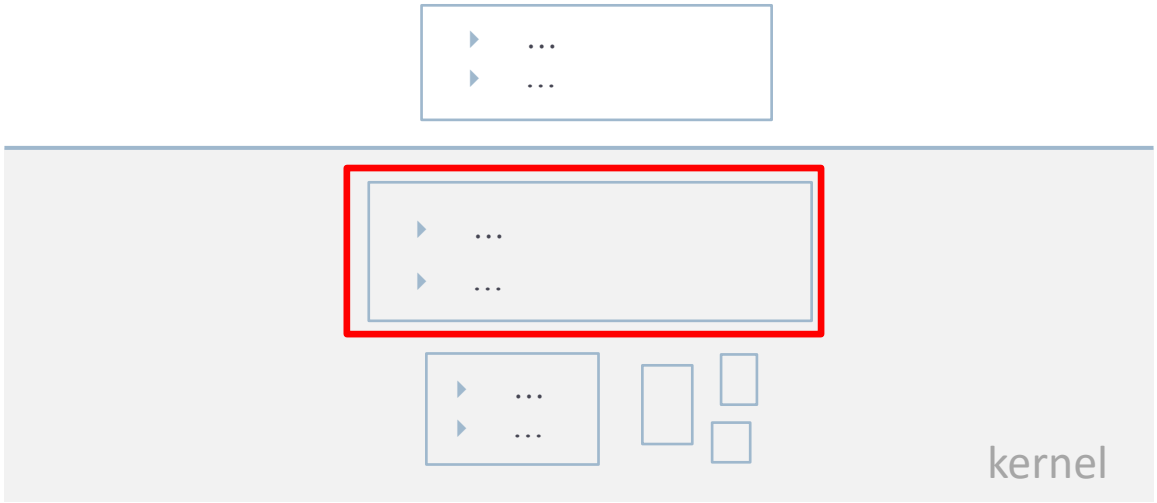


Operating system services
process initialization and termination

Finalize
- Reasons
- API



Create
- Reasons
- API



Process creation

- A process is created:
  - During system startup:
    - Kernel threads + first process (e.g., init, swapper, etc.)
  - When a process exists, it makes a call to the system to create another one:
    - When the operating system starts a new job
    - When a user starts a new program
    - When during the execution of a program a new job is needed
- Resources:
  - Obtains them from the OS, parent distributes resources (part or all to avoid denial of service by multiplication). E.g.: **Copy-on-Write**
  - Parent can run in parallel or wait for child termination.

Process completion

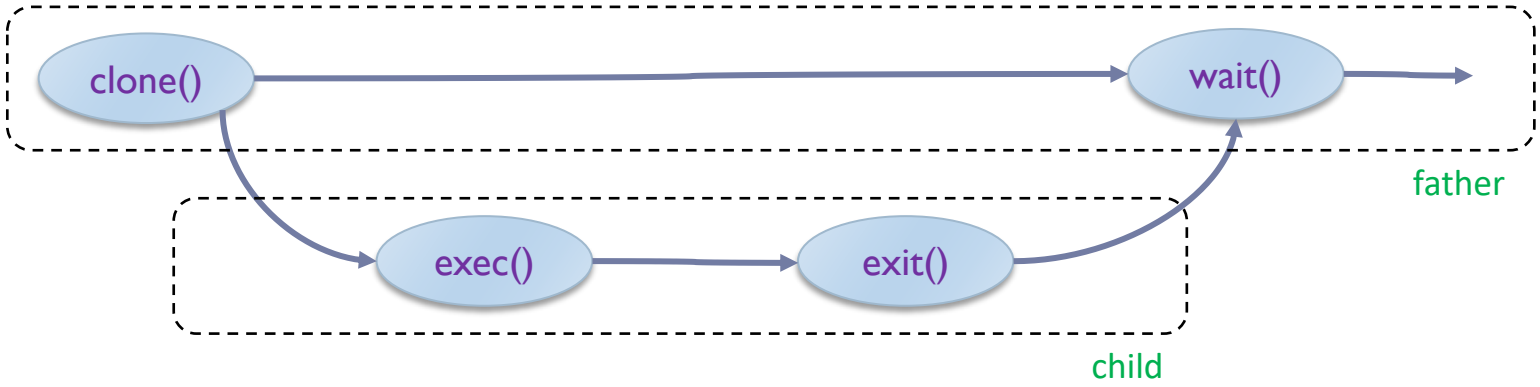
- A process ends:
  - Voluntarily (e.g., through exit call):
    - Normal termination
    - Termination with error
  - Involuntary:
    - Terminated by the system (e.g. exception, no resources needed)
    - Terminated by another process (e.g.: through system call kill)
    - Terminated by the user (e.g.: control-c by keyboard)

    - In Unix/Linux signals are used as a mechanism
    - They can be captured and processed (except SIGKILL) to avoid unintentional termination.

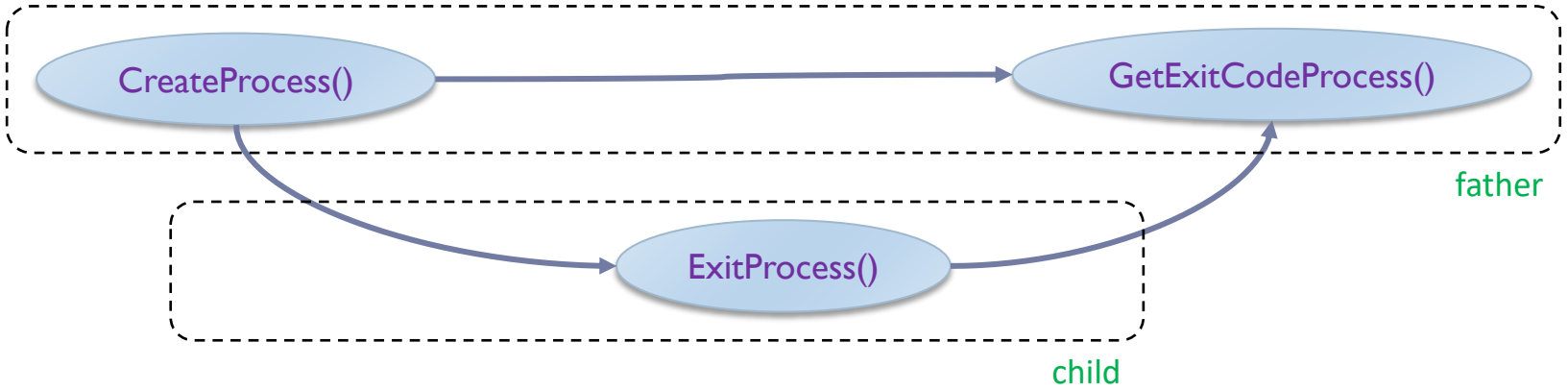
- Resources:
  - All resources are released (files, memory, etc.), >UNIX> except the PCB.
  - The parent is notified, >UNIX> and if it does a wait() then the PCB is released.
  - >UNIX> If parent dies without doing a wait it goes to zombie until init adopts it.

Creation and termination of processes
System calls

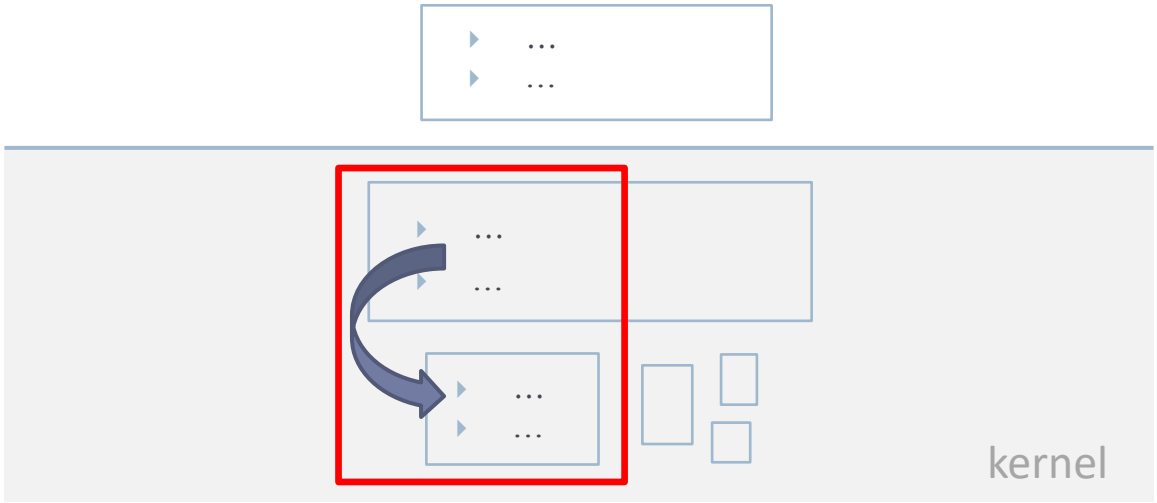
- Linux



- Windows



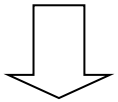
Operating system services
process initialization and termination



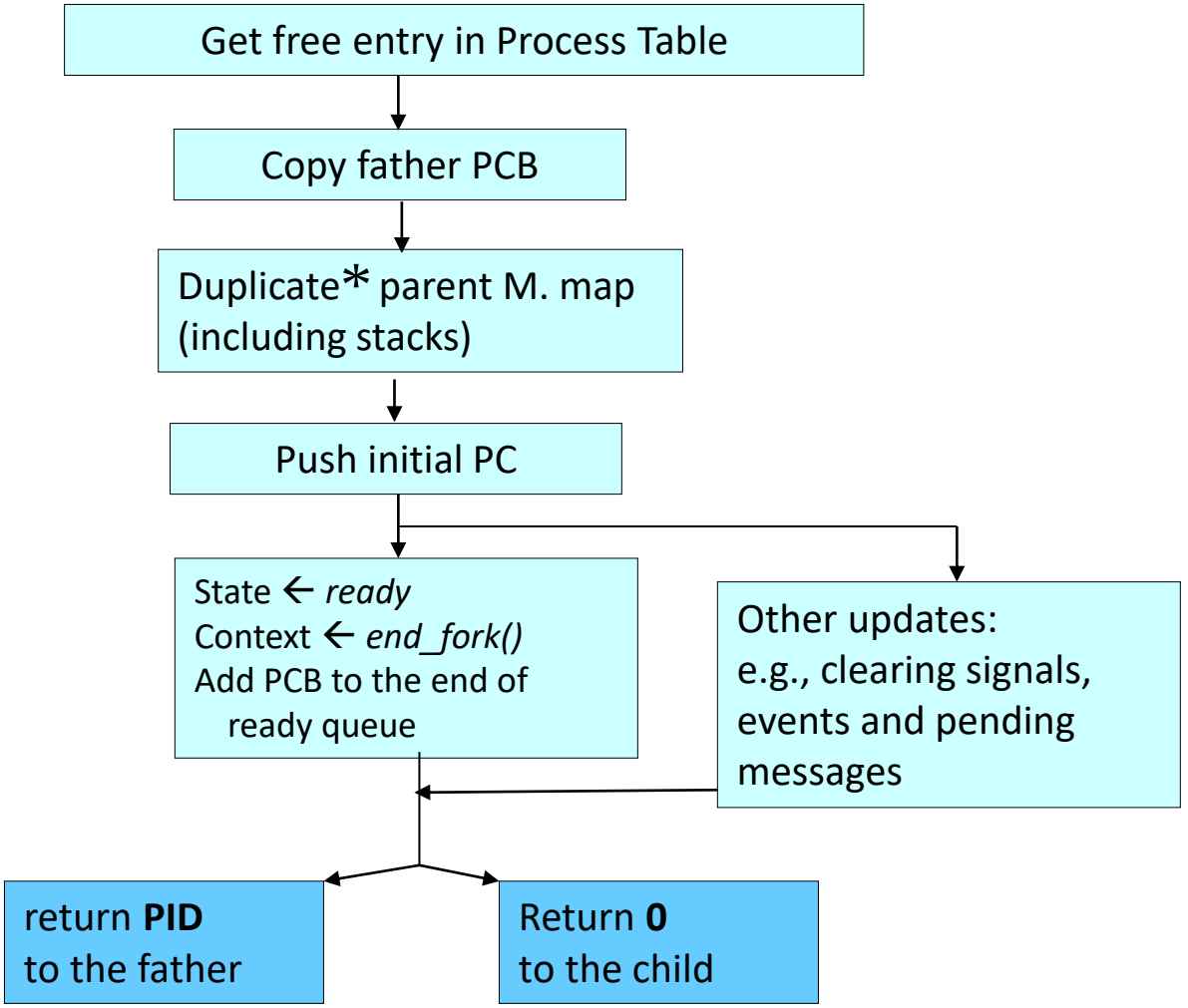
Process creation

Linux: clone

clone:



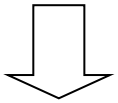
"Clones the father process and gives a new identity to the child"



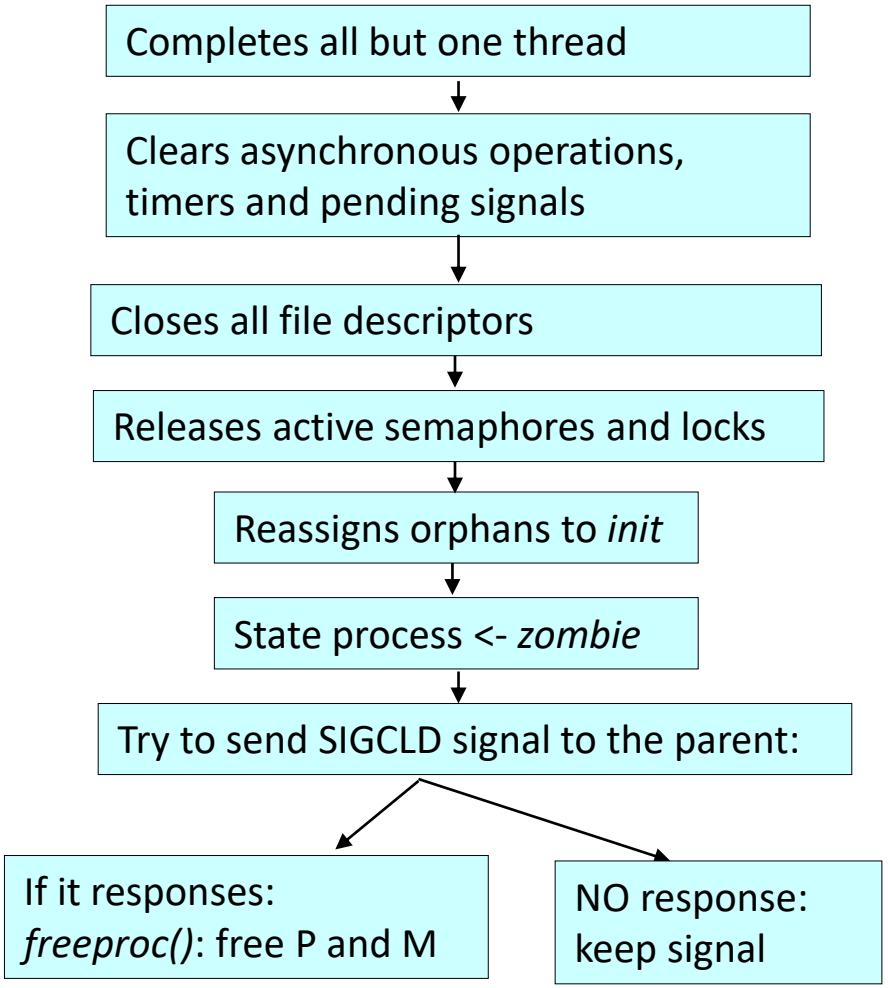
Completion of processes

Linux: exit

exit:



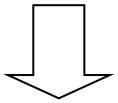
"Ends the execution of a process and releases resources"



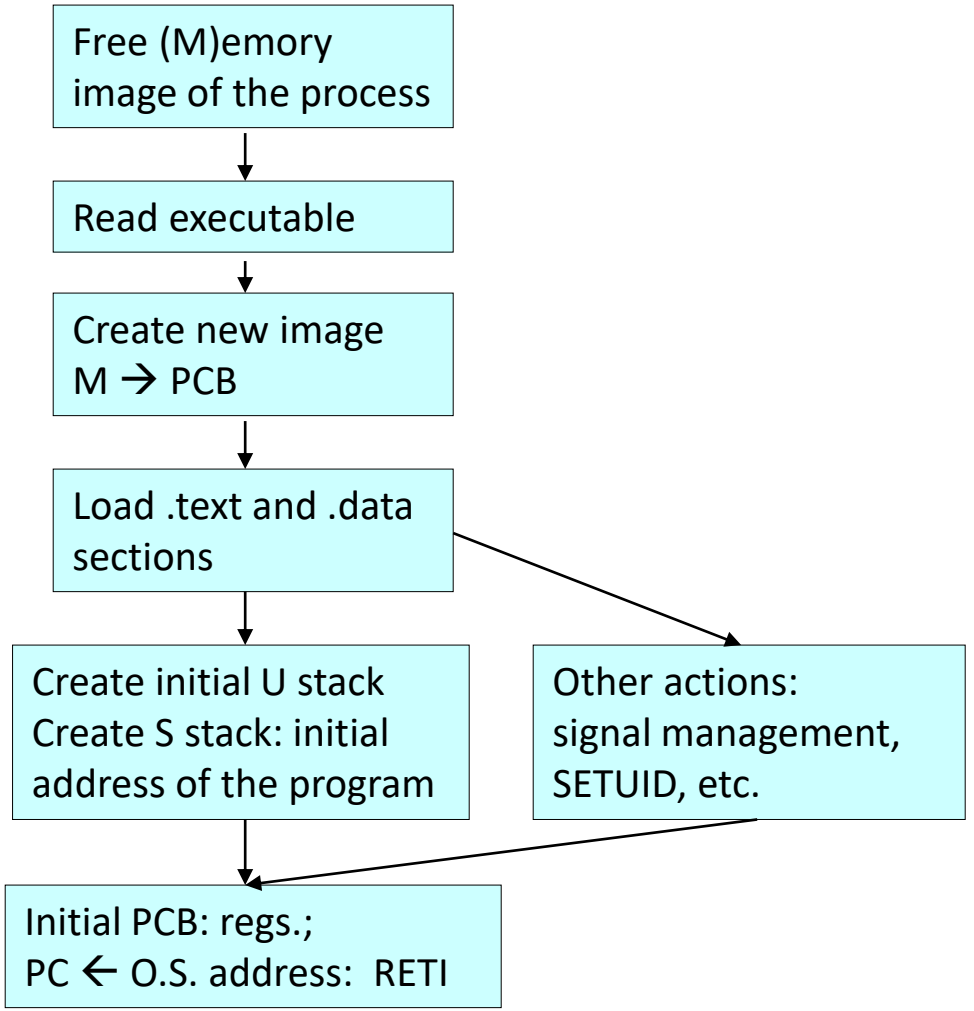
Change the image of a process

Linux: exec

exec:

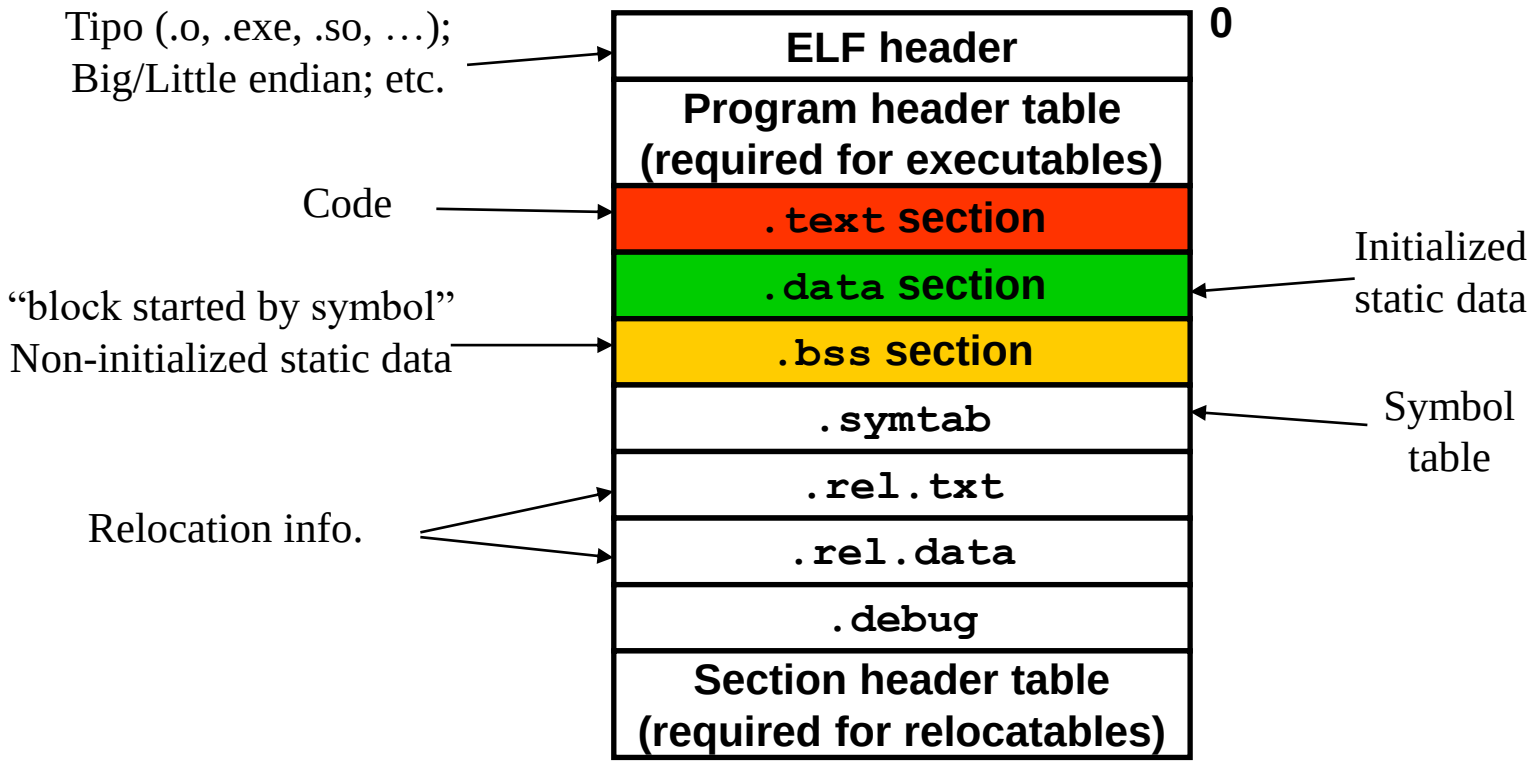


"Changes the memory image of a process using a previous one as a 'container'."

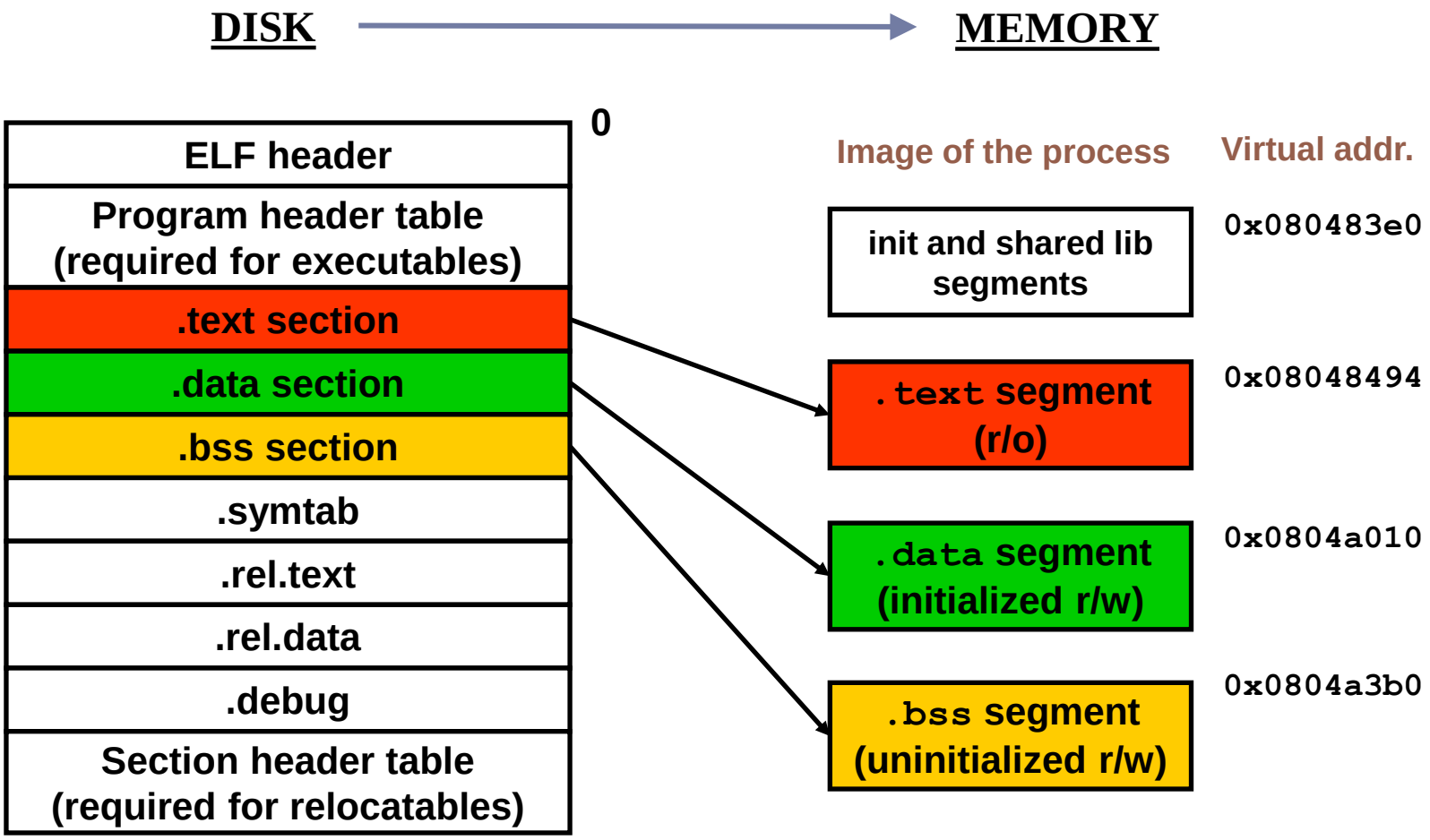


Change the image of a process
ELF format of executable in UNIX

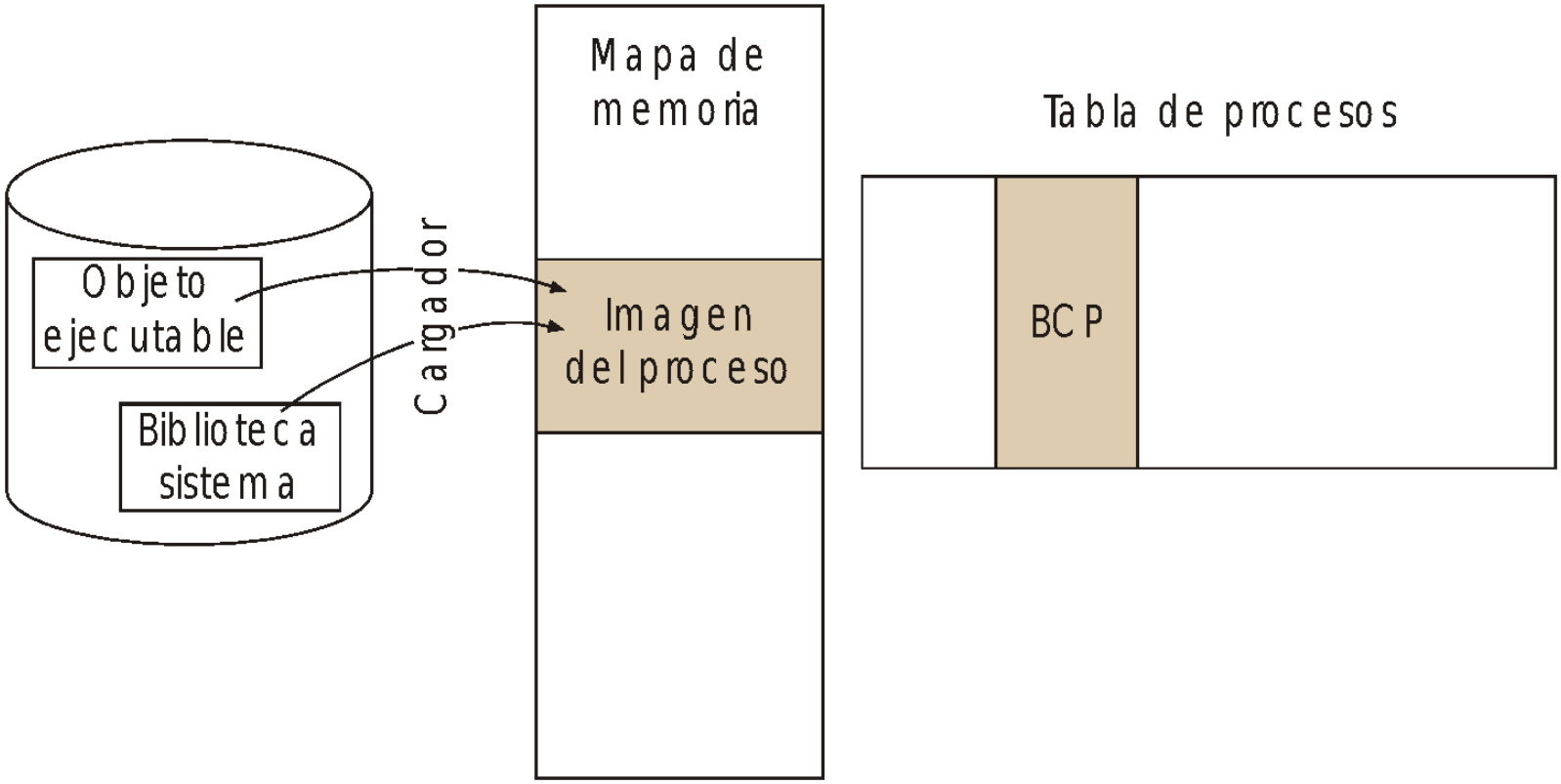
- ELF: Executable and Linkable Format



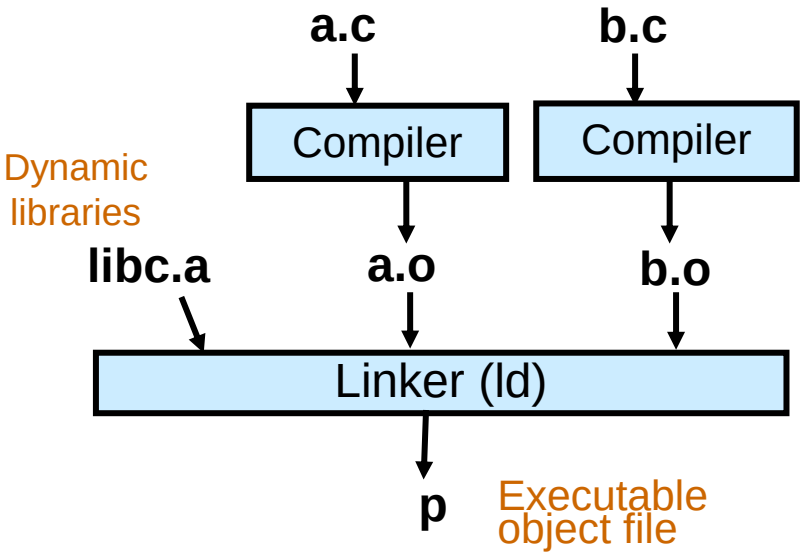
Change the image of a process
loading executable into memory

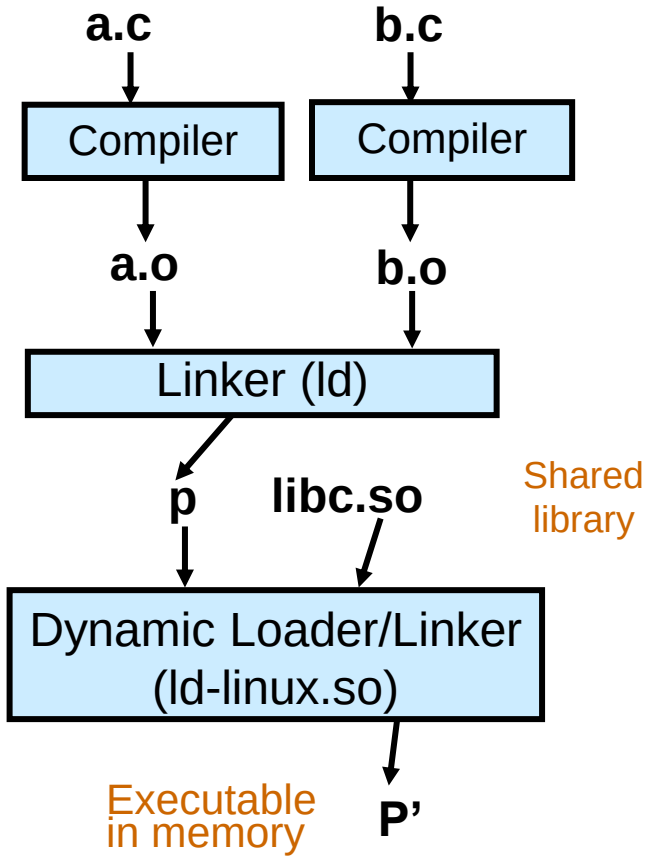


Change the image of a process
loading executable into memory



Change the image of a process
generation of executable





Static libraries

Dynamic libraries

ARCOS Group
Universidad Carlos III de Madrid

Lesson 3
Process and threads

Operating Systems
Computer Science and Engineering