OPERATING SYSTEMS: PROCESSES

Process Management

Contents

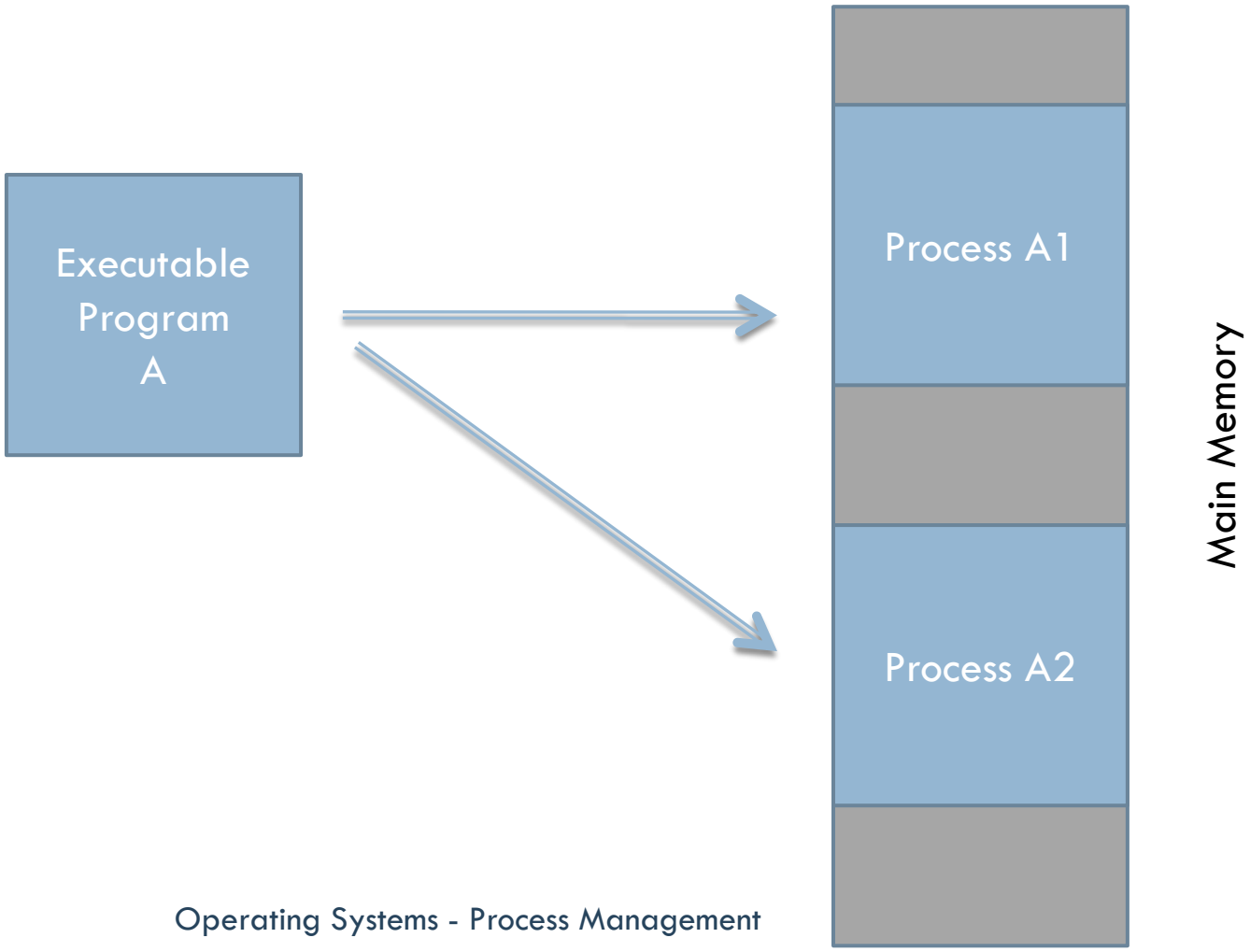
- **Process concept.**
- Basic lifecycle of a process.
- Process information
- Multitasking.
- Context switch.
- Generating an executable.

Process

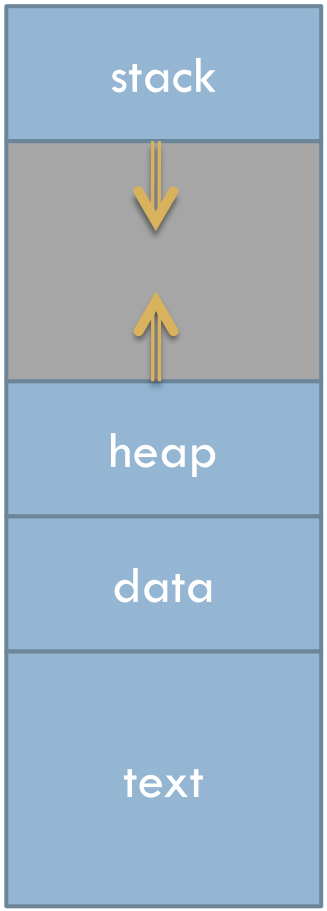
- Process: Program in execution.
  - Each execution of a program leads to a process.
  - Process is the unit of management for operating system

- A process consists of:
  - Program text: Instructions.
  - Set of data associated to program execution.

Program execution



Memory representation

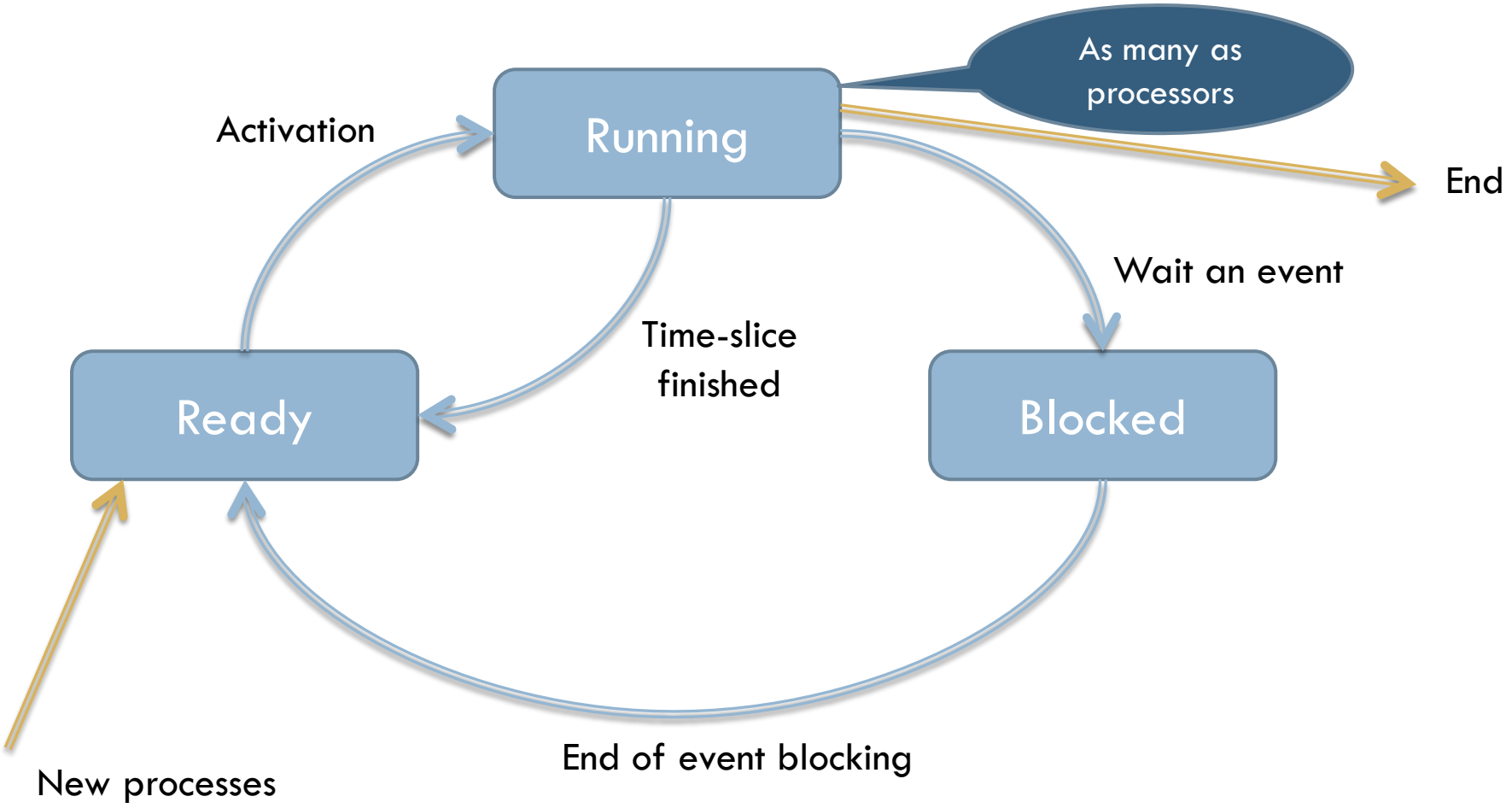


- A process needs memory for instructions and data.
- Different instances of a program need independent areas for data.

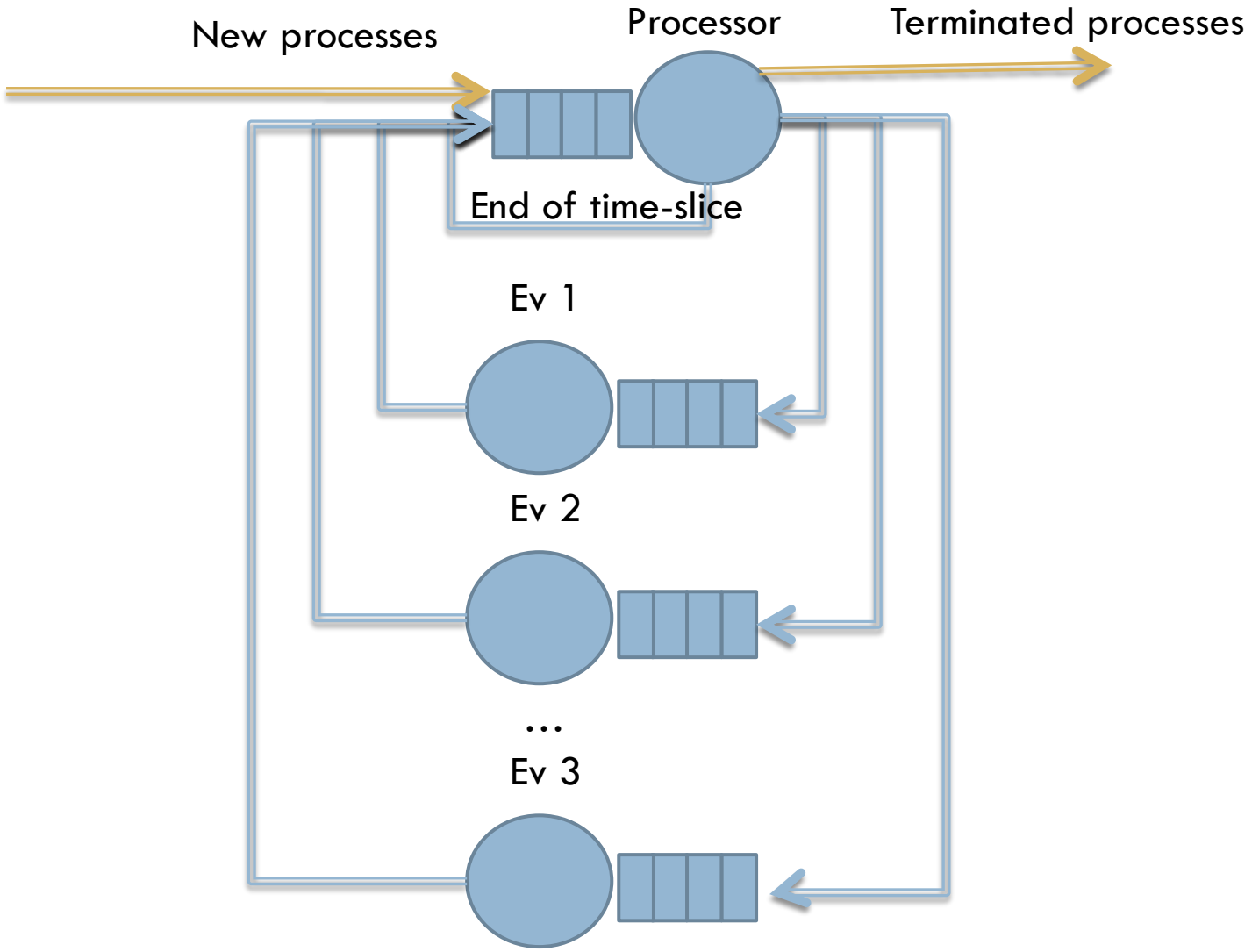
Contents

- Process concept.
- **Basic lifecycle of a process.**
- Process information
- Multitasking.
- Context switch.
- Generating an executable.

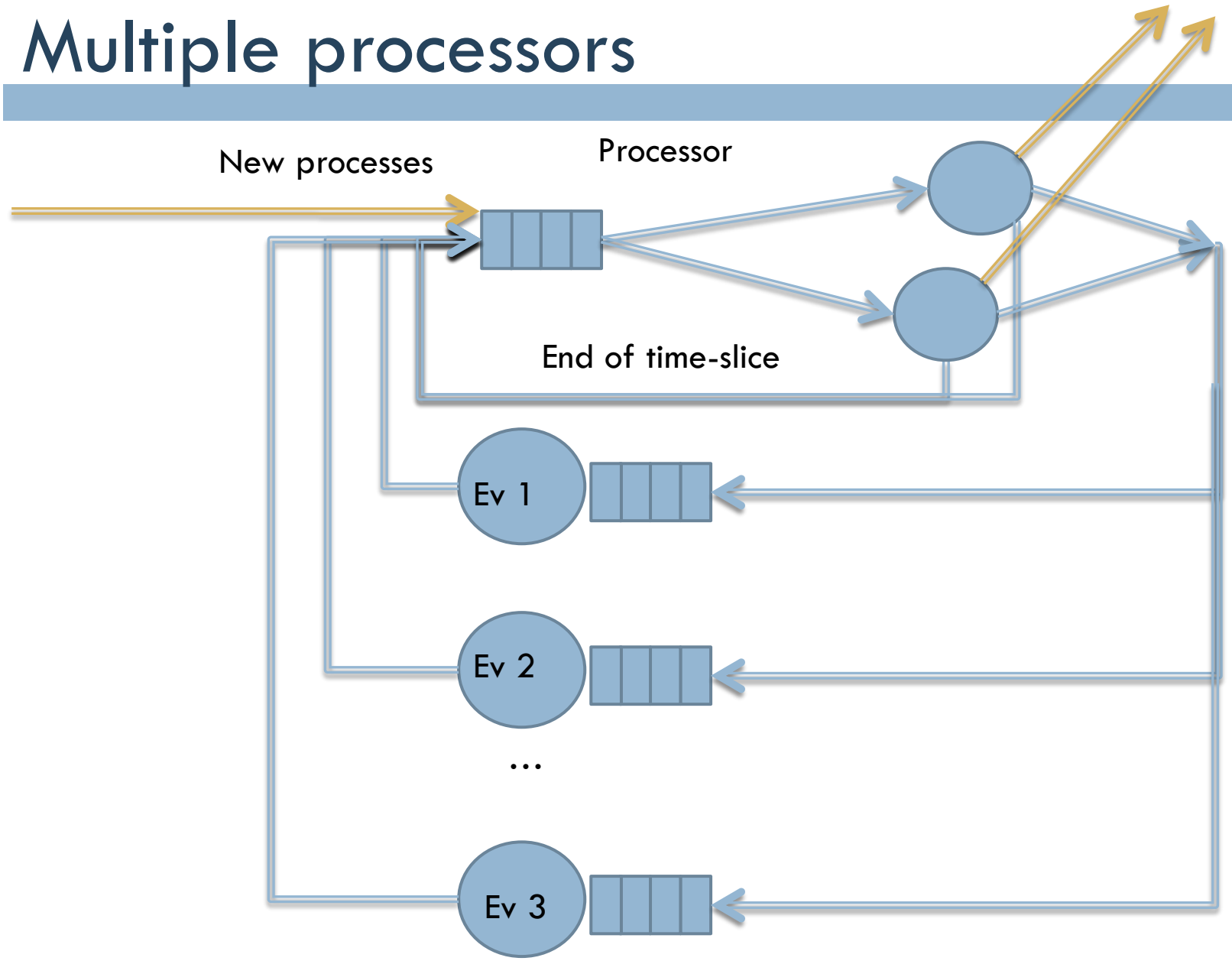
Basic lifecycle of process



Simplified queuing model: Single processor



Simplified queuing model: Multiple processors



Contents

- Process concept.
- Basic lifecycle of a process.
- **Process information**
- Multitasking.
- Context switch.
- Generating an executable.

Process information

- All the information allowing the process correct execution.

- Three categories:
  - Information stored in the processor.
  - Information stored in memory.
  - Additional information managed by operating system.

Processor state

- Processor state includes values of processor registers.
  - Registers accessible in user mode.
    - General registers: Register file.
    - Program counter.
    - Stack pointer.
    - User part in status register.
  - Registers accessible in privileged mode:
    - Privileged part from status register.
    - Memory management registers (e.g. PTBR).

- Context switch:
  - Save processor state for outgoing process.
  - Restore processor state for incoming process.

Memory image of a process

- Memory image consists of the **memory spaces** that a process is authorized to use.
- If a process generates an address out of the address space, hardware generates a **trap.**
- Depending on specific computer, memory image may be referred to *virtual memory* or *physical memory*.

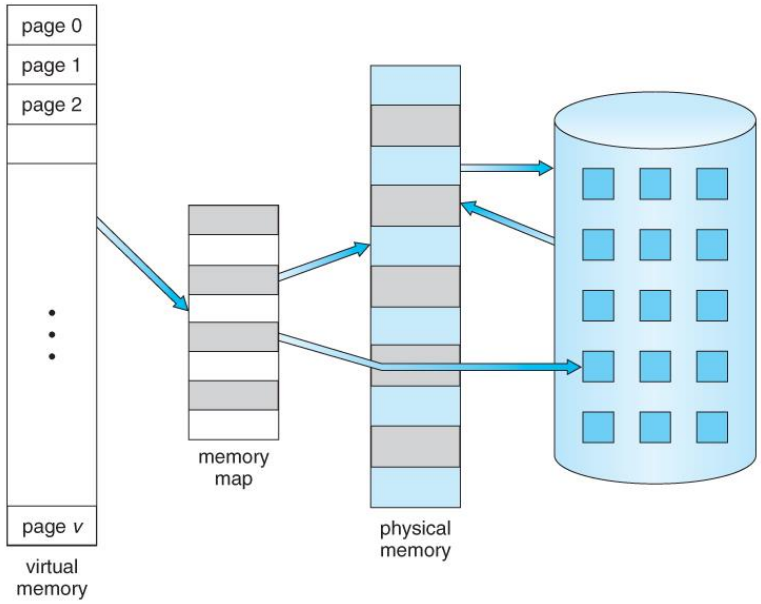
Memory image models: single region

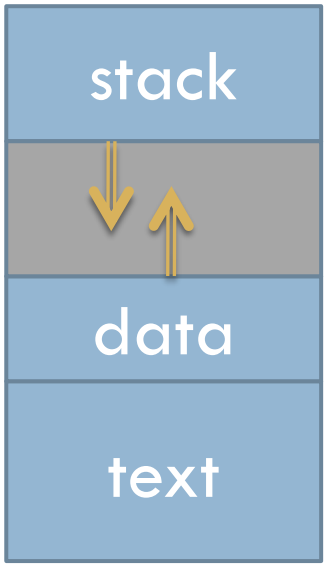
- Process with a single fixed size region.
  - Used in systems without virtual memory.

- Process with a single variable sized region.
  - Systems without virtual memory:
    - Need to allocate space → Memory waste.
  - Systems with virtual memory:
    - Virtual reserve space → Feasible but less flexible than multiple region.
    - Not used.

Memory image models: multiple regions

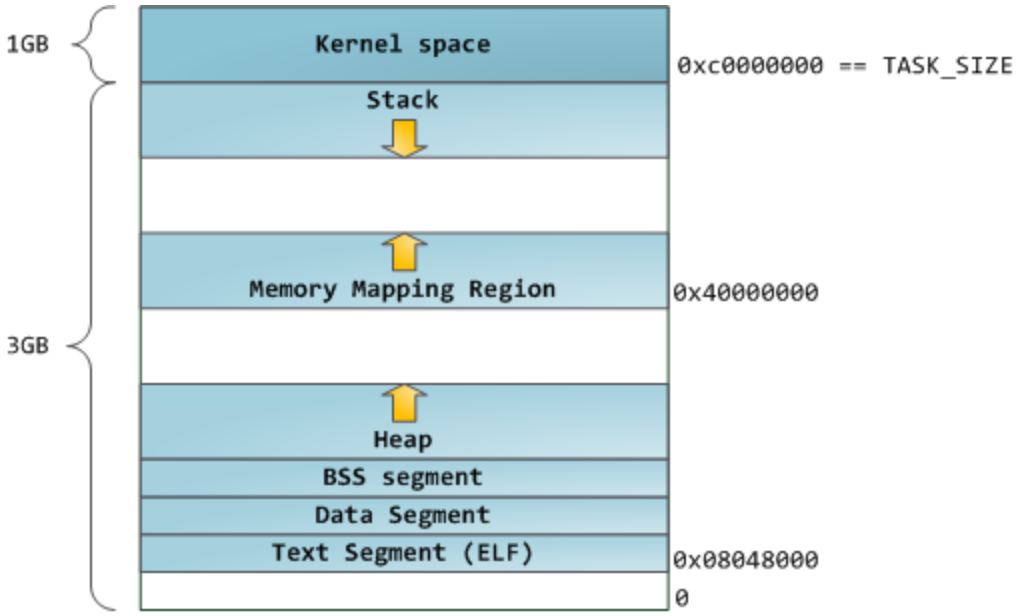
- Process with fixed number of regions of variable size.
  - Predefined regions (text, data, stack).
  - Each region may grow.
  - With virtual memory, the *gap* between stack and heap does not consume resources.





Memory image models: multiple regions

- Process with variable number of regions of variable size.
  - More advanced option (used in current versions in Windows and UNIX).
  - Process structured as a number of regions.
  - Very flexible:
    - Shared regions.
    - Regions may differ in permissions.



Operating system information

- Operating system keeps additional information on processes.
  - Operating system keeps information in a table: **Process Table**.
  - **Process Control Block** (PCB): Each entry in table keeps information about one process.
  - Almost all information about process stored in PCB.
    - Some information elements kept outside due to implementation reasons.

Contenidos del BCP

- Identification information.
- Processor state.
- Process control information.

Scheduling and state information:
- Process state.
- Waited event (if blocked).
- Process priority.
- Scheduling information.

Allocated regions description.
- Per region information.

Allocated resources:
- Open files.
- Used communication ports.
- Timers.

Pointers for structuring process queues (or rings).

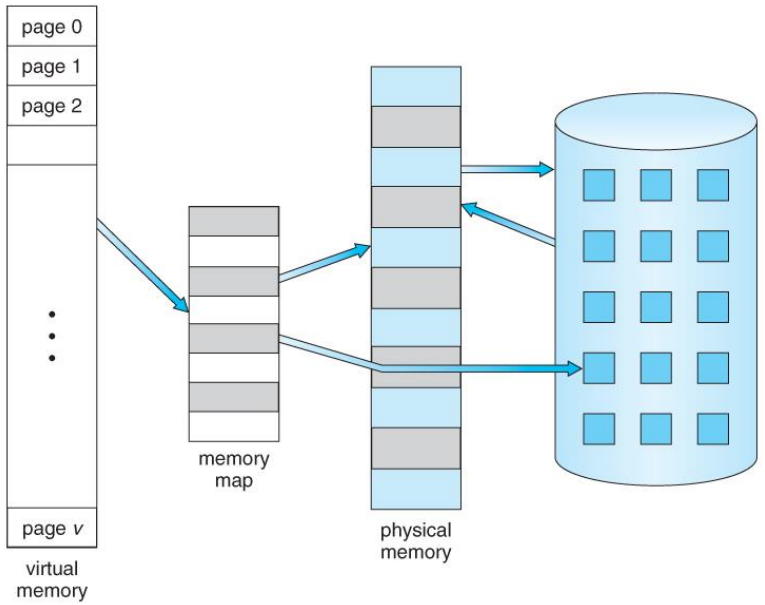
Information for inter process communication.

Information out of PCB

- Not all the information referred to a process is stored in its PCB.
- Decision taken in function of:
  - **Efficiency.**
    - Tables should have a prefixed size and always be in memory.
    - Size needs to be optimized.
  - **Information Sharing.**
    - If data needs to be shared it cannot be in the PCB.
    - Pointers are used to point to other structures (tables) allowing for information sharing.
      - Open files.
      - Memory pages.

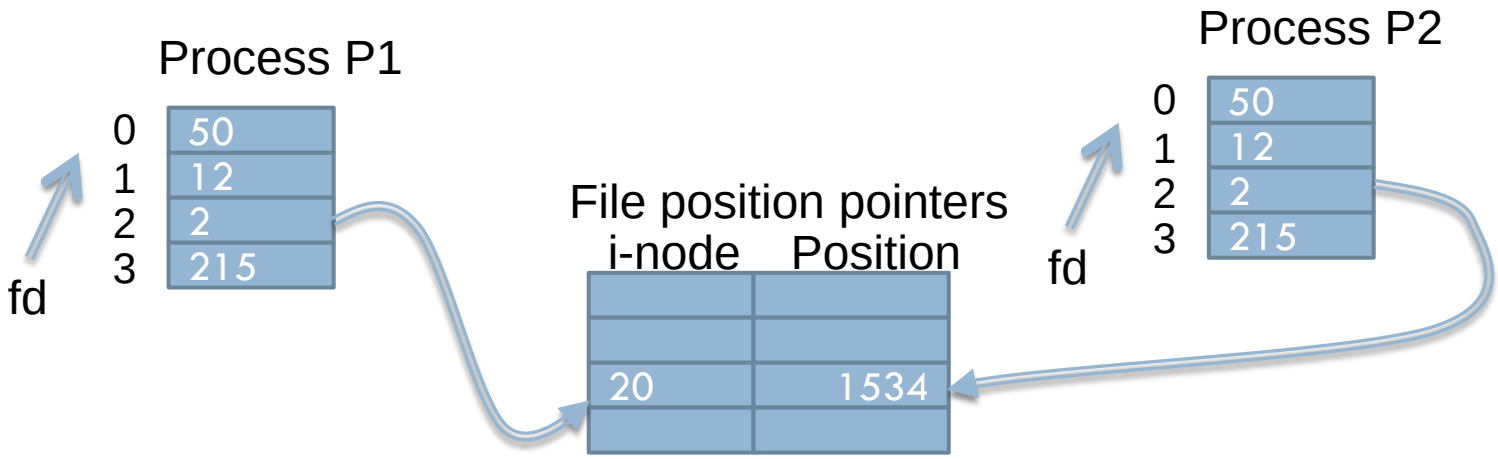
Page Table

- Placed outside PCB.
- Describes process memory image.
- PCB contains pointer to page table.
- Reasons:
  - Variable size.
  - Memory sharing among processes requires it to be external to PCB.



File position pointers

- Placed outside PCB.
- If open files table (in PCB) was in PCB I would not be be shared.
- If associated to i-node is always shared.
- Stored in a common structure for multiple processes and a new one allocated with OPEN service.



Example: Running a command


```
#include <sys/types.h>
#include <stdio.h>
int main(int argc, char** argv) {
  pid_t pid;
  pid = fork();
  switch (pid) {
    case -1: /* error */
      exit(-1);
    case 0: /* proceso hijo */
      if (execvp(argv[1], &argv[1])<0) { perror("error"); }
        break;
    default:
      printf("Proceso padre");
  }
  return 0;
}
```

prog cat f1

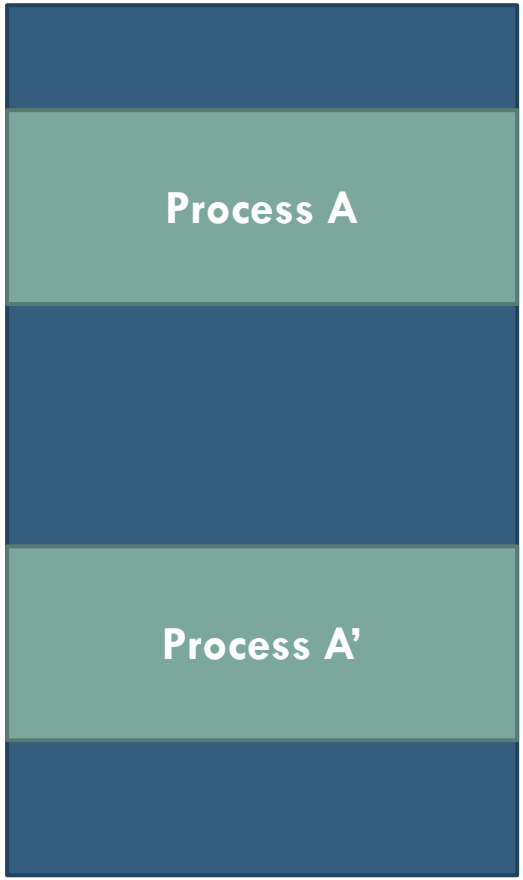
Fork service

- `pid_t fork(void);`

- Duplicates process invoking the call.
- Parent process and child process go on running the same program.
- Child process inherits open files from parent process.
  - Open file descriptors are copied.
- Pending alarms are deactivated.

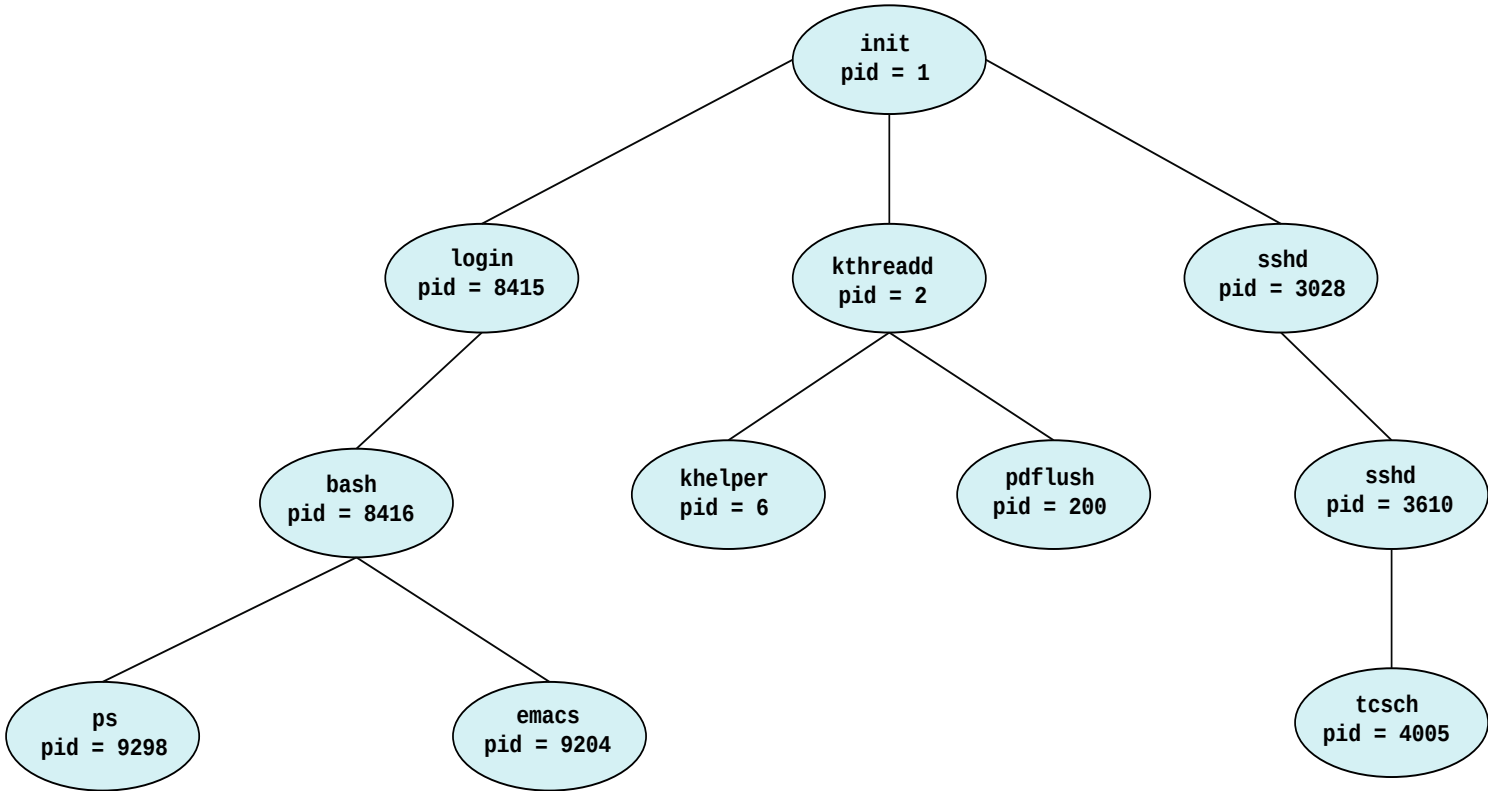
- Returns:
  - -1 on error.
  - In parent process: child process descriptor.
  - In child process: 0.

Fork service



FORK

Linux process tree



Exec service

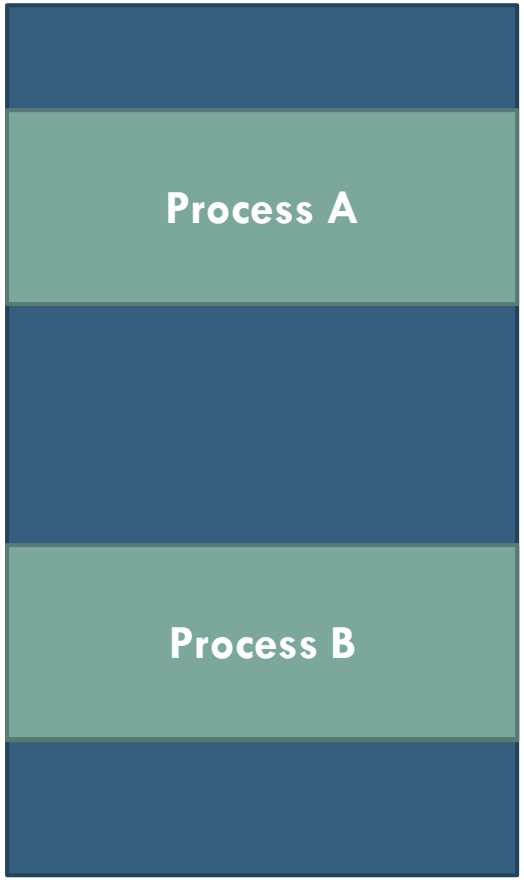
- Single service with multiple library functions.

```
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[])
```

- Changes current process image.
  - **path**: path to executable file.
  - **file**: Looks for the executable file in all directories specified by PATH.

- Description:
  - Returns -1 on error, otherwise it does not return.
  - The same process runs another program.
  - Open files remain open.
  - Signals with default action remain defaulted, signals with handler take default action.

Exec service



EXEC

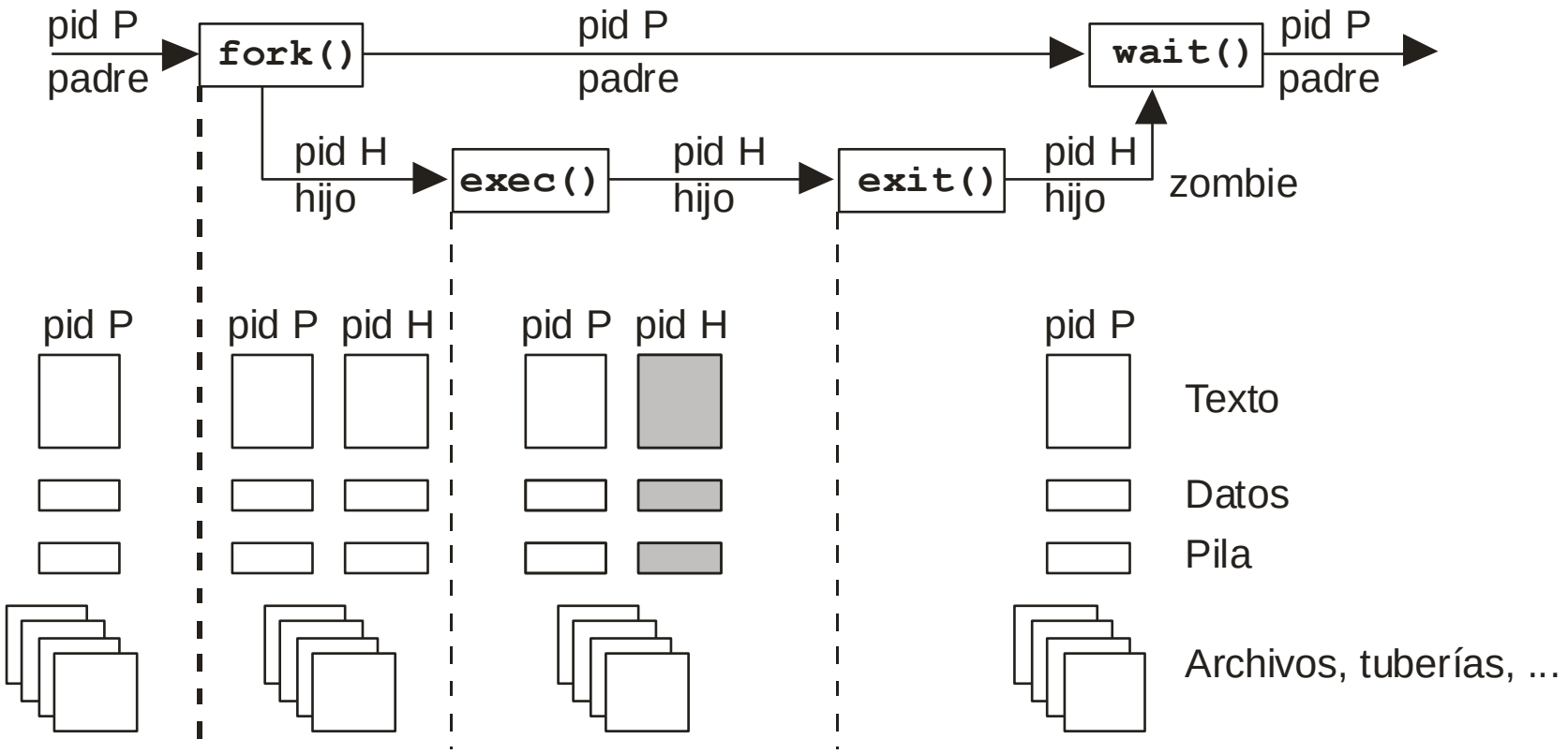
Exit service

- Finalizes process execution.

```
void exit(status);
```

- All open files descriptors are closed.
- All process resources are released.
- **PCB** (Process Control Block) is released.

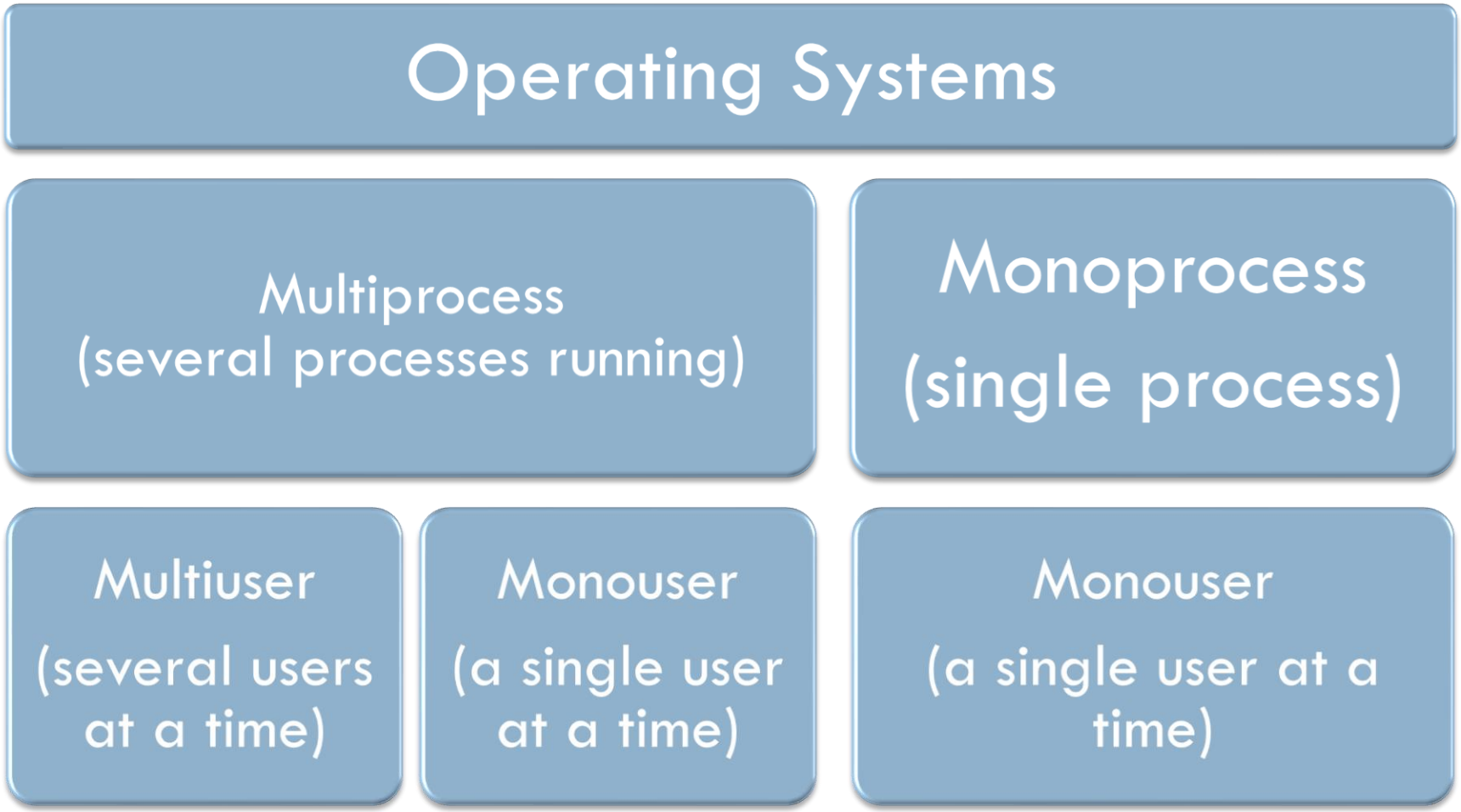
Use of fork, exec, wait y exit



Contents

- Process concept.
- Basic lifecycle of a process.
- Process information
- **Multitasking.**
- Context switch.
- Generating an executable.

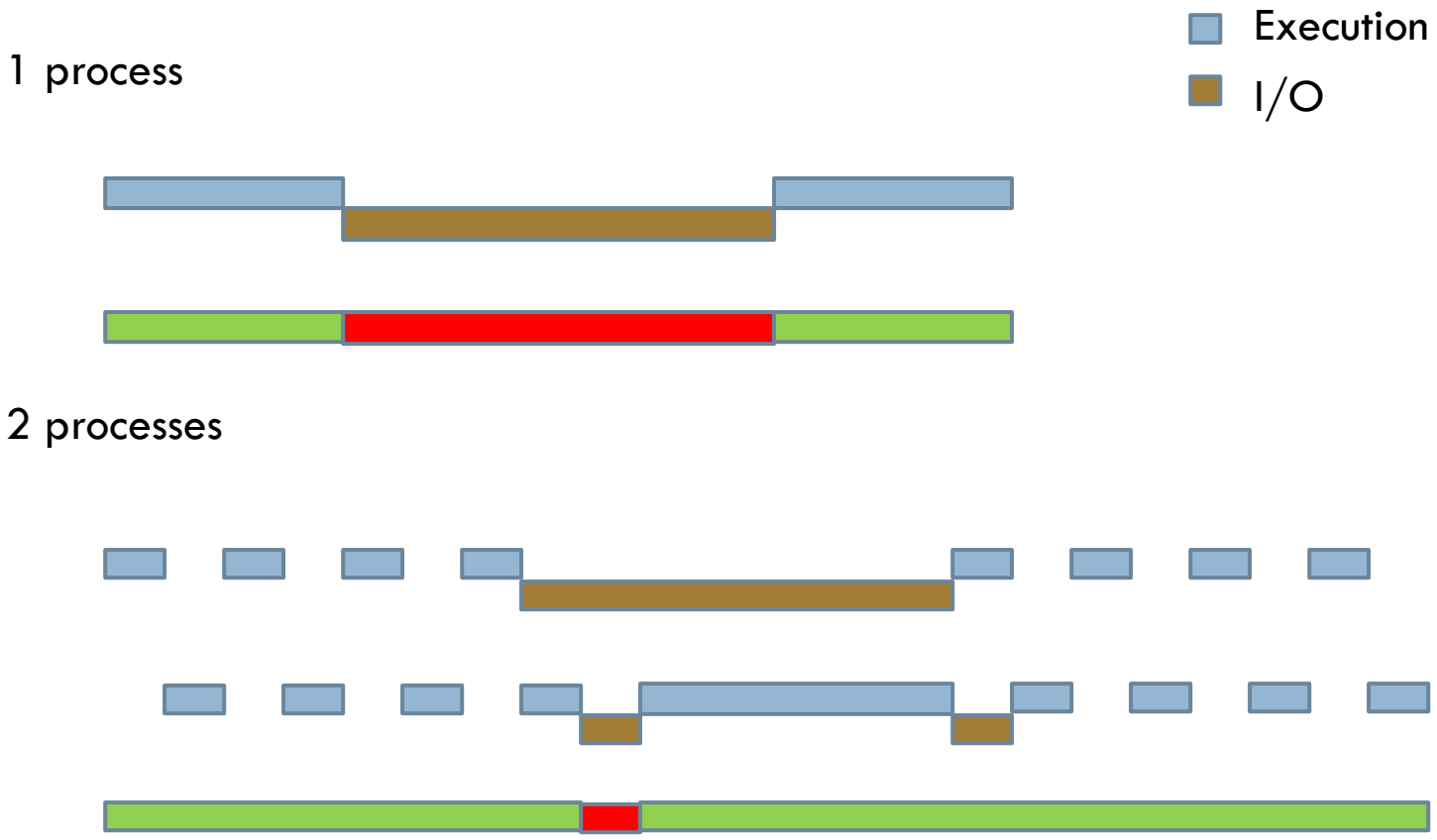
Operating system types



Principles of multitasking

- Real parallelism between I/O and CPU (DMA).
- Process alternate between I/O and processing phases.
- Several processes stored in memory.

Multiprogramming: CPU use

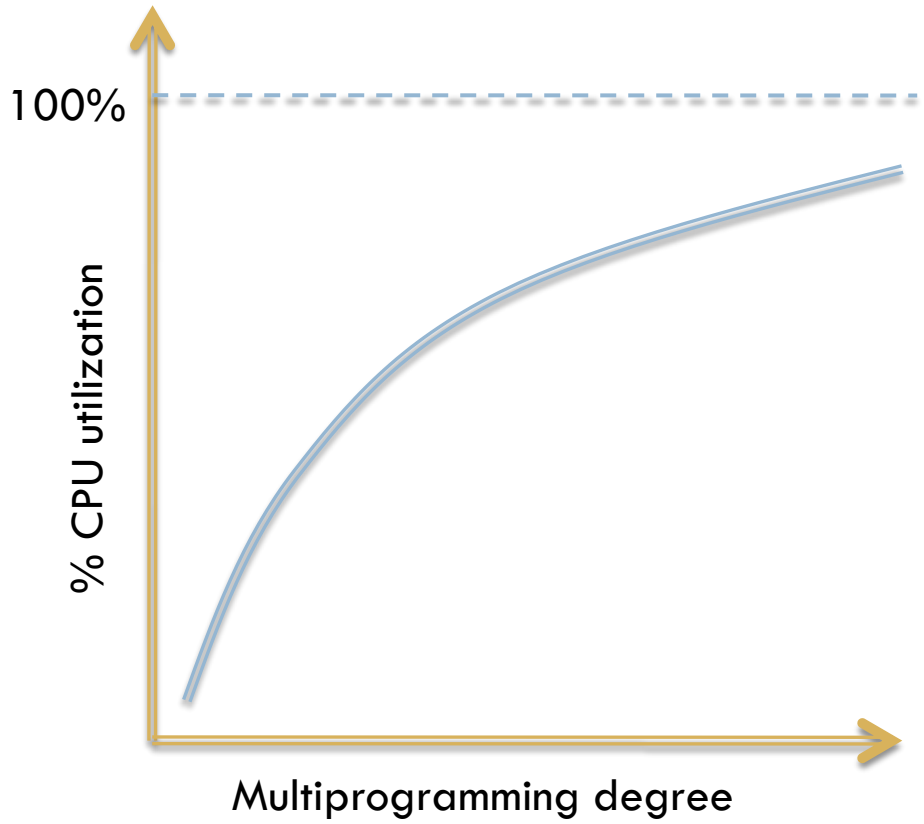


Advantages of multitasking

- Eases programming, dividing a program in multiple processes (modularity).
- Allows simultaneous interactive service of multiple users in an efficient way.
- Takes advantage of the times a process spends waiting for an I/O operation to be completed.
- Increases utilization of CPU.

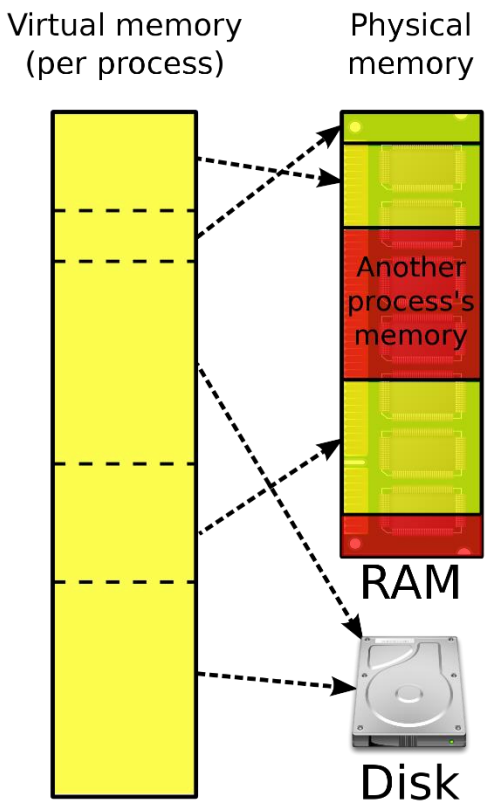
Multiprogramming degree

- **Multiprogramming degree**: Number of active processes.
- **Main memory needs**: System without virtual memory.

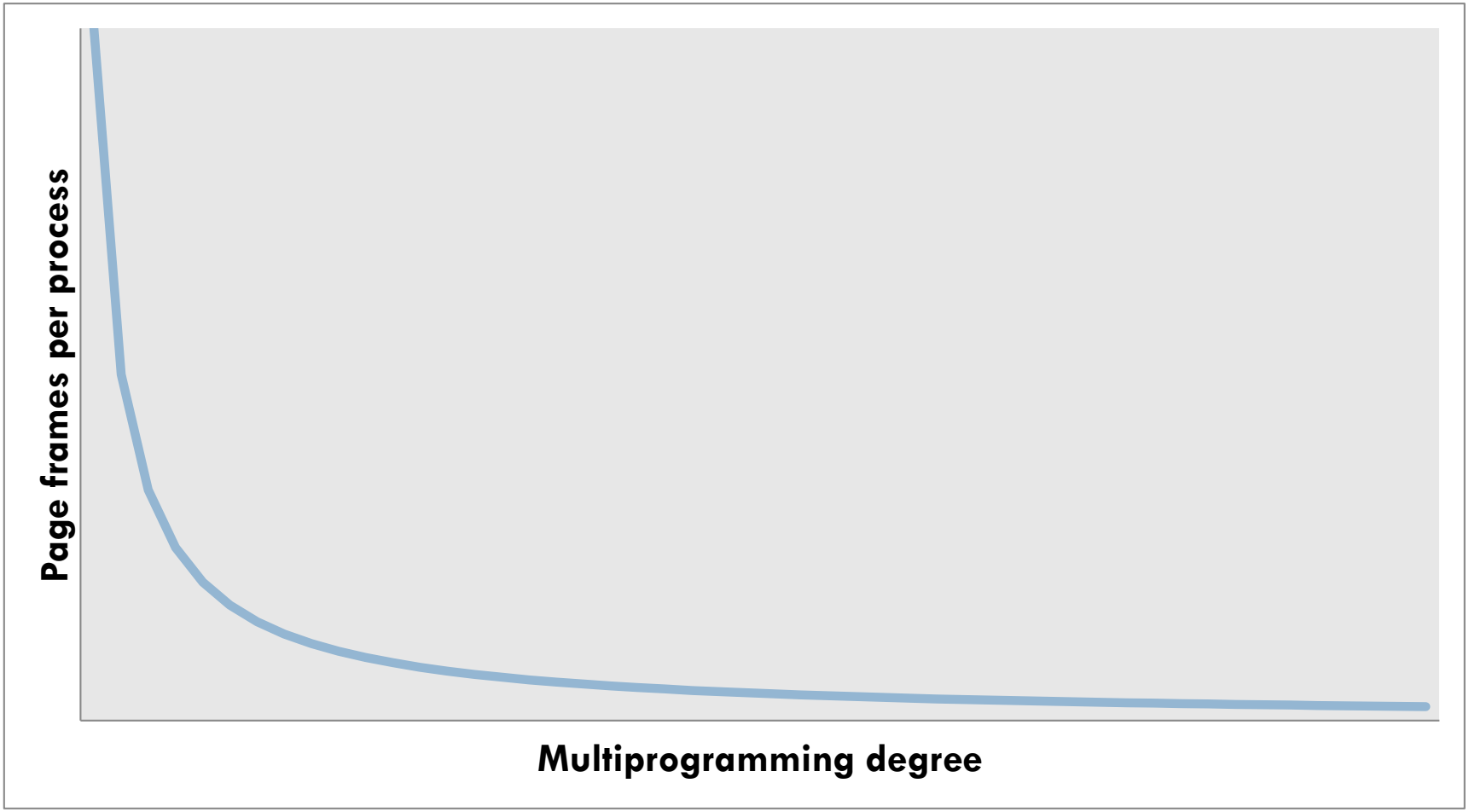


Multiprogramming and virtual memory

- Systems with virtual memory:
  - Divide addressing space of processes in pages.
  - Divide physical memory addressing space in main memory in page frames.
- At a given time, each process has a certain number of its pages in main memory (resident set).

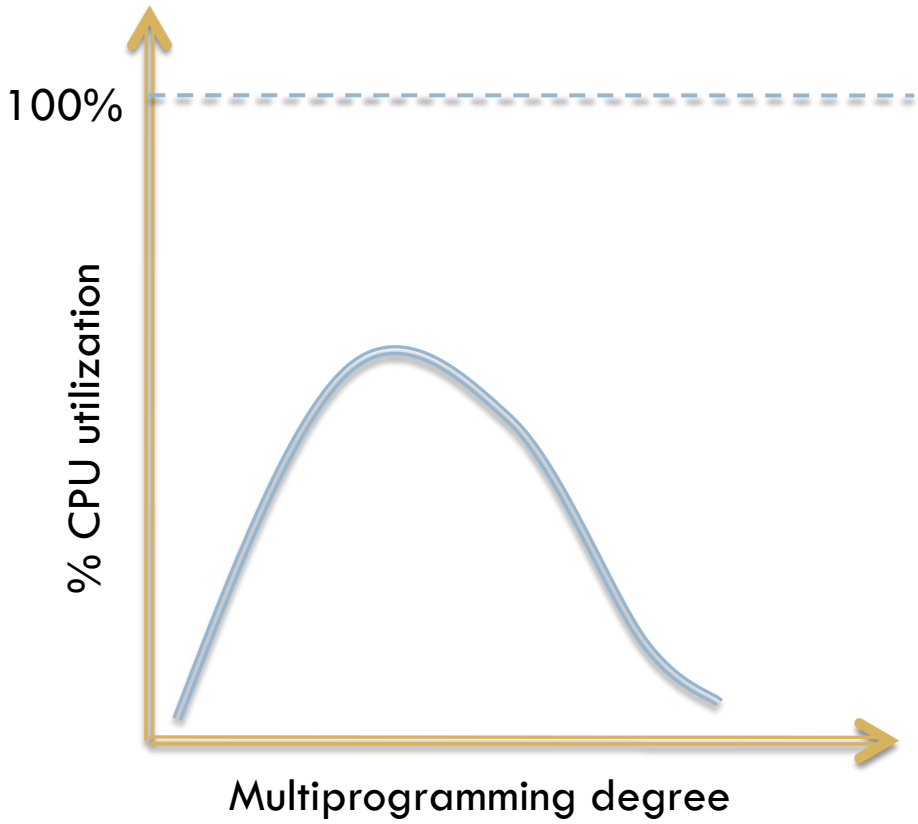


Memory needs: Virtual memory system



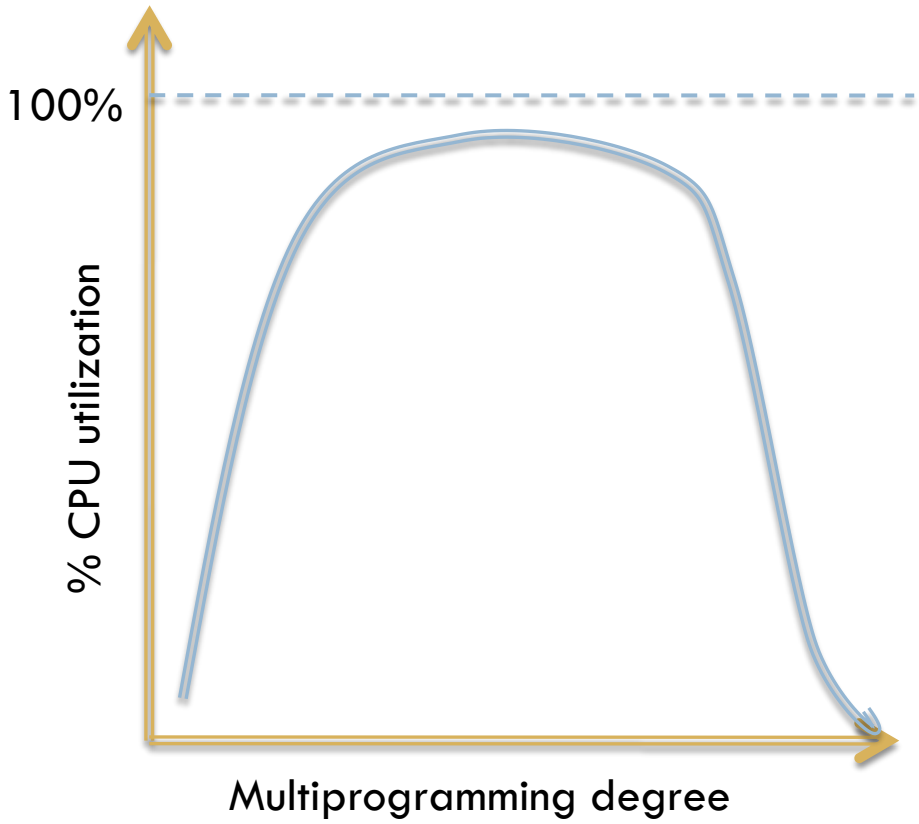
Performance: Small physical memory

- When multiprogramming degree increases:
  - Resident set size decreases for each process.
- Trashing happens before achieving a high CPU utilization percentage.
- **Solution**: Add more main memory.



Performance: Large physical memory

- When multiprogramming degree increases:
  - Resident set size decreases for each process.
- High CPU utilization percentage is achieved with less processes that fit in memory.
- **Solution:** Improve processor or add more processors.



Contents

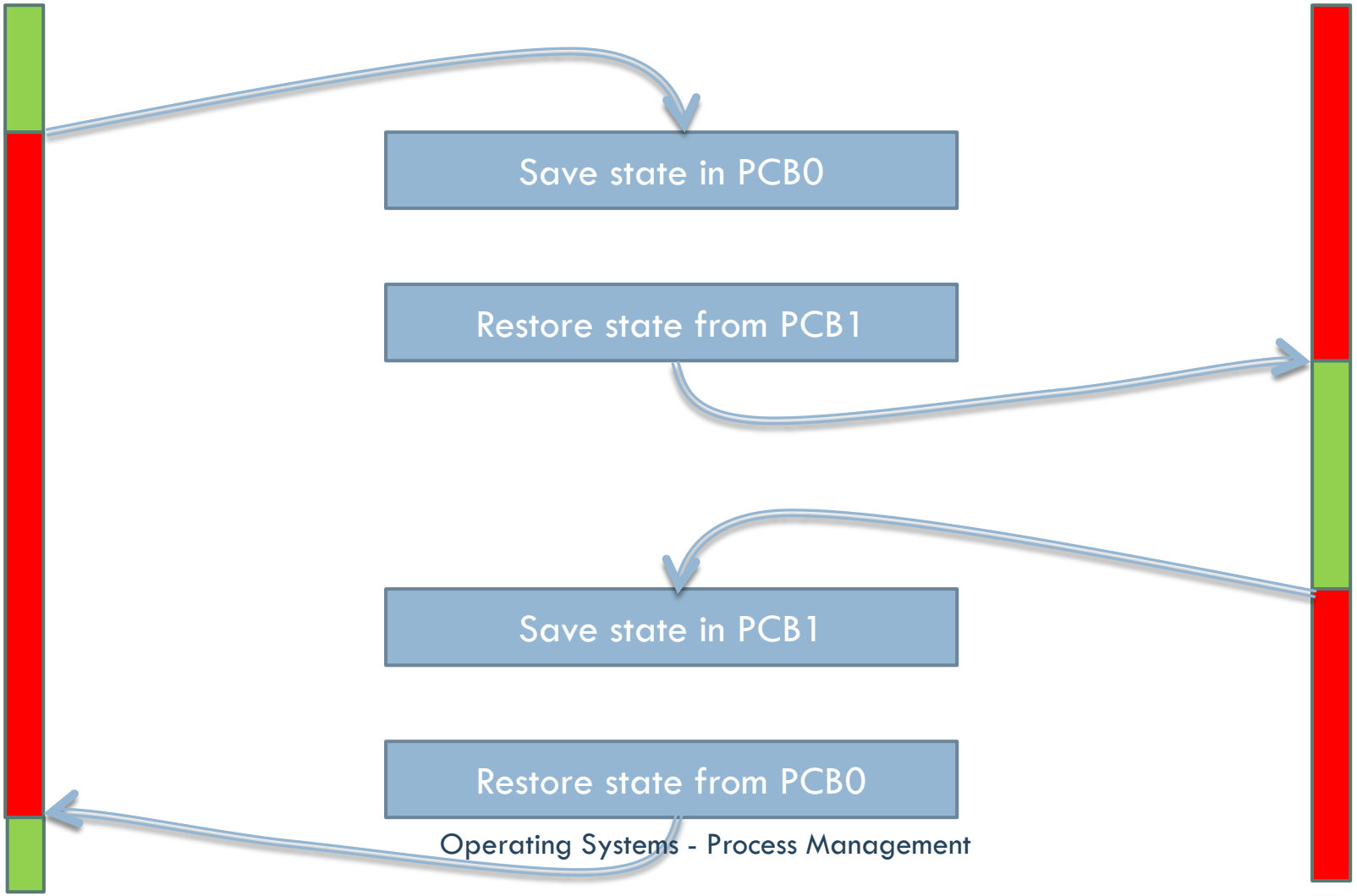
- Process concept.
- Basic lifecycle of a process.
- Process information
- Multitasking.
- **Context switch.**
- Generating an executable.

Context switching

- When operating system assigns processor to a new process.

- **Actions**:
  - Save process state in PCB for process in execution.
  - Restore state of new process in processor.

Context switch



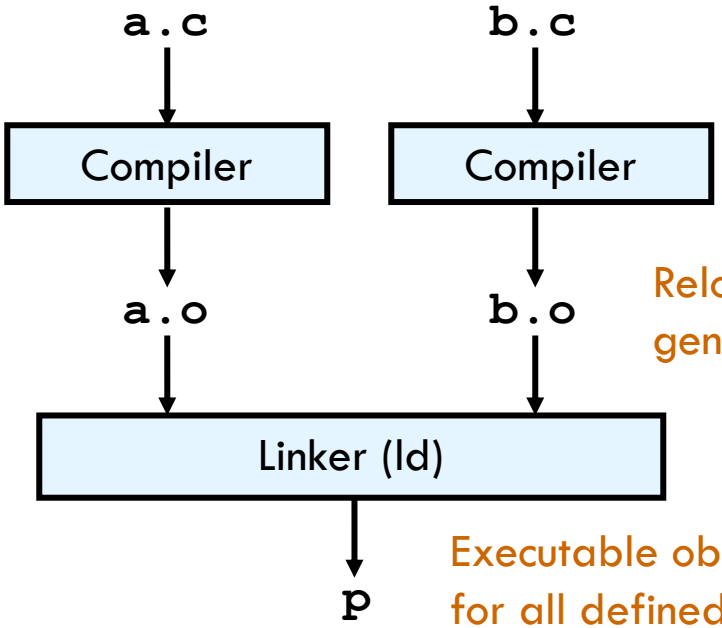
Context switching types

- **Voluntary context switch:**
  - Process performs call to operating system (or generates exception like page fault) implying waiting for an event.
  - *Running* → *Blocked*.
  - Examples: reading from terminal, page fault.
  - Reason ⇒ *efficiency*.
- **Involuntary context switch:**
  - OS appropriates CPU.
  - *Running* → *Ready*.
  - Examples: time slice elapsed or process moves from blocked to ready and has higher priority.
  - Reason ⇒ *use sharing*

Contents

- Process concept.
- Basic lifecycle of a process.
- Process information
- Multitasking.
- Context switch.
- **Generating an executable.**

Executable generation



Relocatable object files separately generated

Executable object file (contains code and data for all defined functions in a.c and b.c).

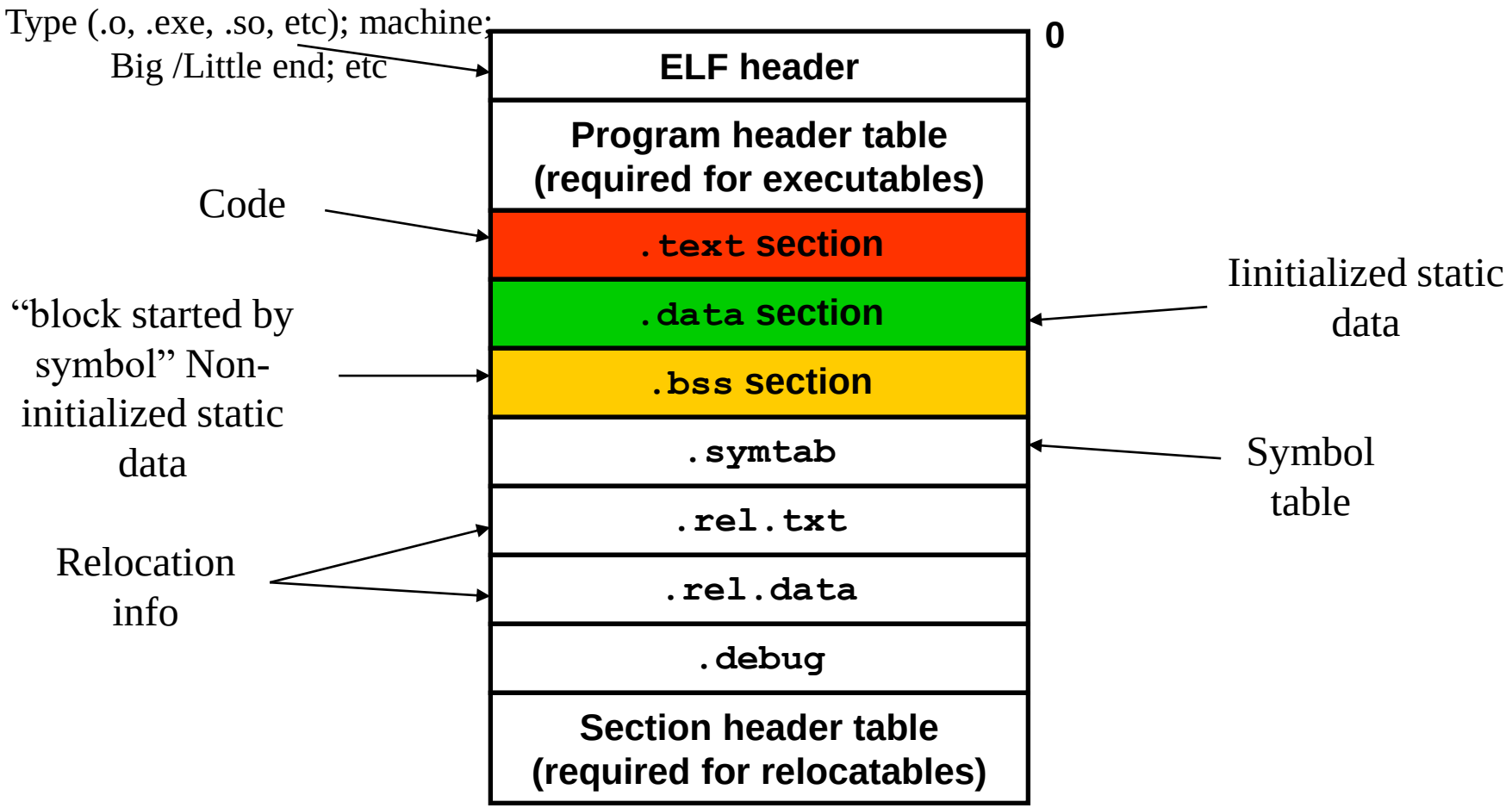
Link Editor (*linker*)

- Combines object files:
  - Merges several relocatable object files (.o) in a single executable object file: input for loader.
- Resolves external references:
  - References to symbols defined in another object file.
- Relocates symbols:
  - From relative positions in .o to absolute positions in executable: adjust refs to these new positions.
  - Symbols: refs to functions and data.

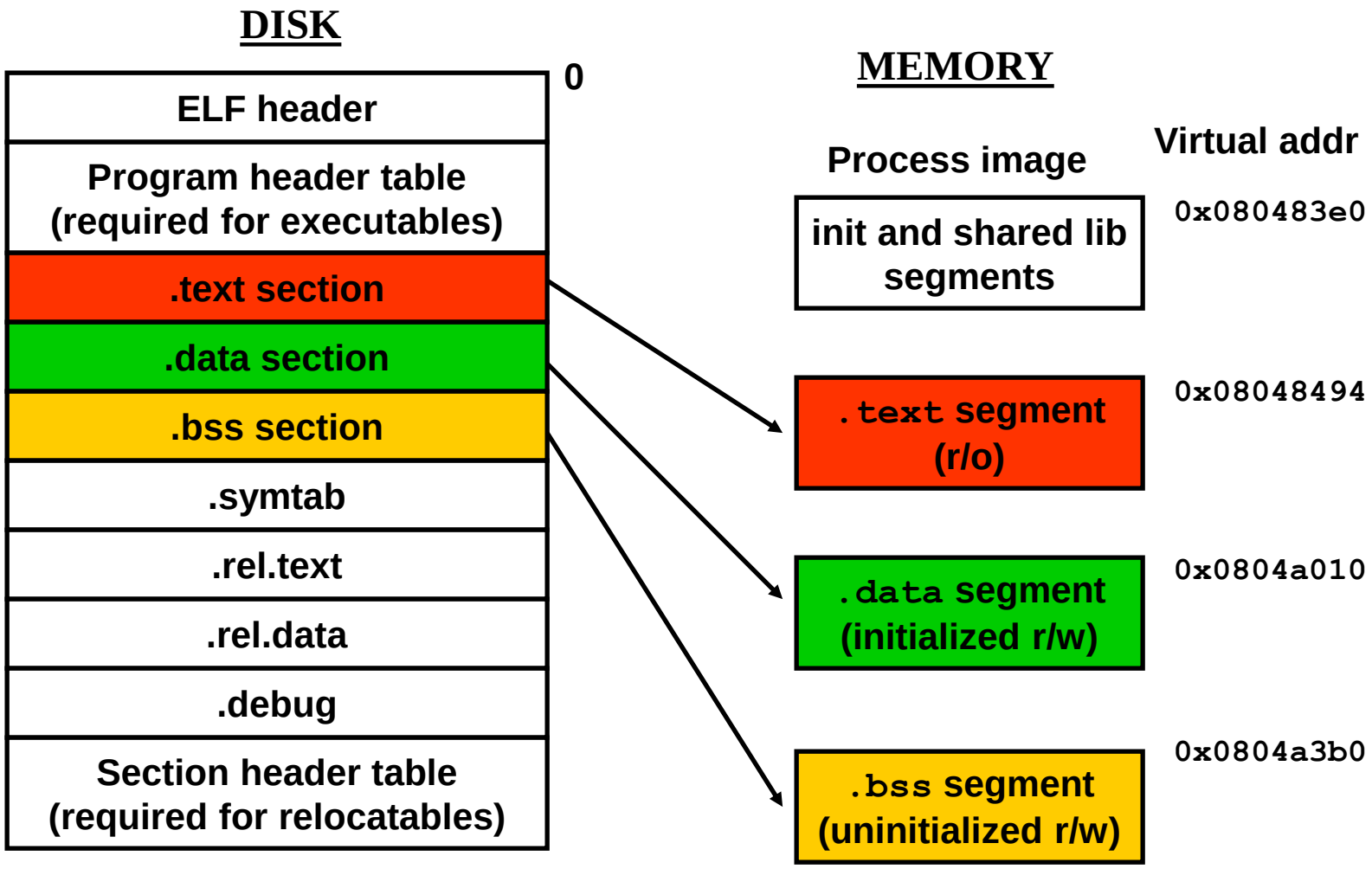
Ejemplo: Formato ELF

- ELF: Executable and Linkable Format
  - formato binario estándar para ficheros objeto
  - original de System V → BSD, Linux, Solaris
  - formato unificado para:
    - ficheros objeto reubicables
    - ficheros objeto ejecutables
    - ficheros objeto compartidos

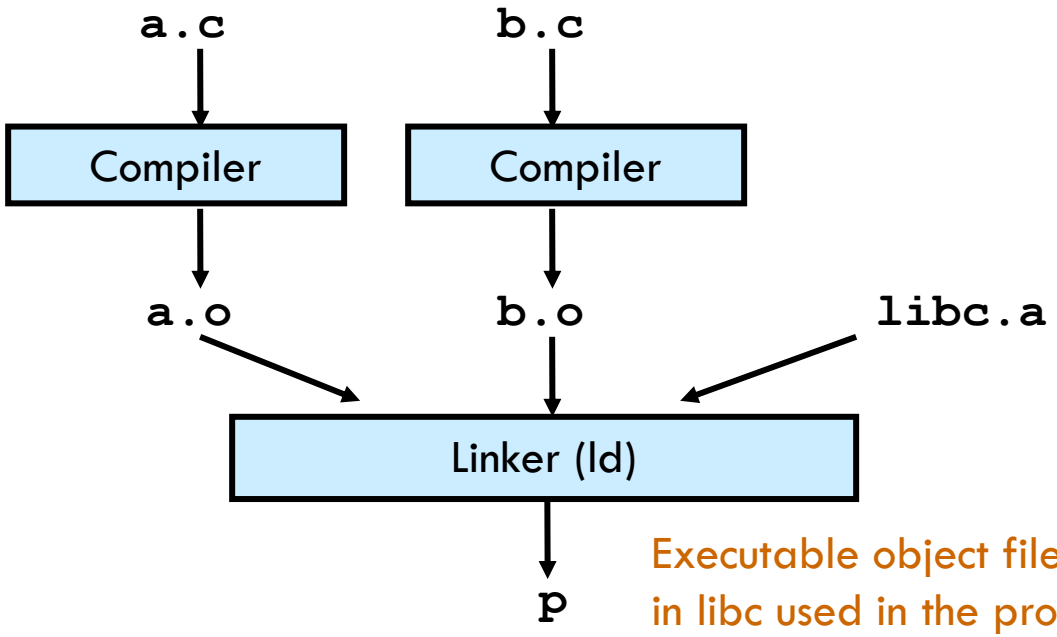
ELF format


Type (.o, .exe, .so, etc); machine; Big /Little end; etc
0
ELF header
Program header table (required for executables)
Code
.text section
Iinitialized static data
.data section
"block started by symbol" Non-initialized static data
.bss section
.symtab
Symbol table
.rel.txt
Relocation info
.rel.data
.debug
Section header table (required for relocatables)

Executable load



Static library



Static library: file with contatenation of relocatable object files.

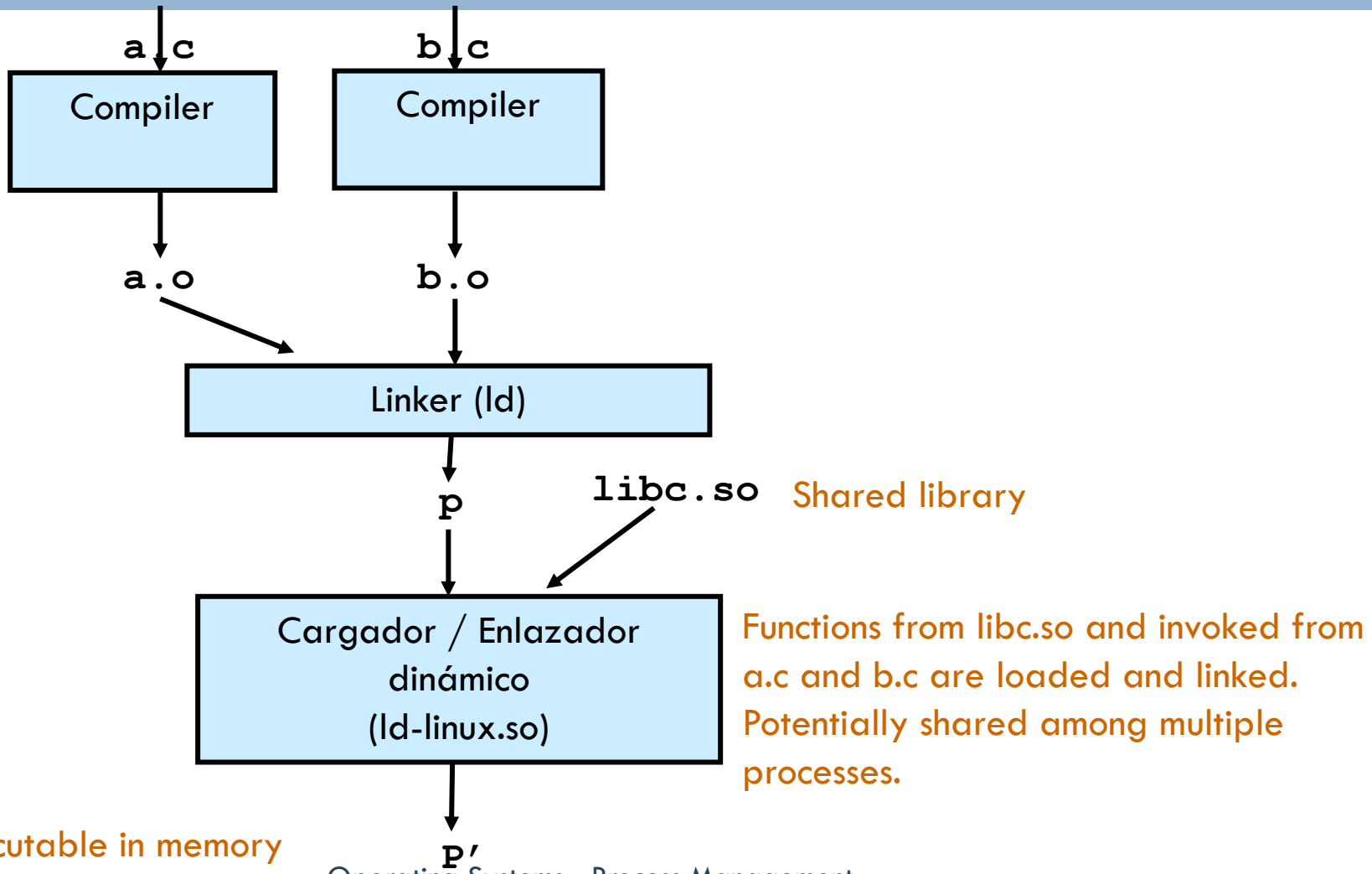
Executable object file: includes code from functions in libc used in the program.

Static and dynamic libraries

- Drawbacks of static libraries:
  - Code potentially duplicated in executables:
    - Disk (file system).
    - Virtual memory space in processes.
  - Bugs in libraries → new version → need to relink

- Solution: **dynamic libraries (*.so)** (dynamic link libraries, DLLs):
  - Components loaded in memory and executed at runtime.
  - Functions from libraries may be shared among multiple processes.

Bibliotecas dinámicas



Reminder

- Difference between program and process.
  - A process is a program in execution.
- Operating system manages running processes (process lifecycle).
- Process information consisting of process state, memory image and PCB.
- Multitasking allows a better use of computer resources.
- Context switching introduces a small overhead.
- Static libraries are linked at compile time while dynamic libraries are linked at process creation time.
- Process creation implies creation of its memory image and the allocation of a PCB.