

# Lección 4

## Gestión de memoria

Sistemas Operativos  
Ingeniería Informática

# A recordar...

---

Antes de clase

Clase

Después de clase

Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:  
las transparencias solo no son suficiente.  
Preguntar dudas (especialmente tras estudio).

Ejercitar las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar los **cuadernos de prácticas** y las **prácticas** de forma progresiva.

# Lecturas recomendadas

---

## Base



1. Carretero 2020:
  1. Cap. 5
2. Carretero 2007:
  1. Cap. 4

## Recomendada



1. Tanenbaum 2006:
  1. (es) Cap. 4
  2. (en) Cap. 4
2. Stallings 2005:
  1. Parte tres
3. Silberschatz 2006:
  1. 4

# Contenidos

---

1. **Introducción:**
  1. Modelo abstracto y definiciones básicas.
  2. Mapa de memoria de un proceso: regiones de memoria.
2. **Funciones del gestor de memoria.**
  1. Particionamiento de memoria.
  2. Algoritmos de gestión de memoria.
3. **Memoria virtual.**
4. **Servicios del gestor de memoria.**

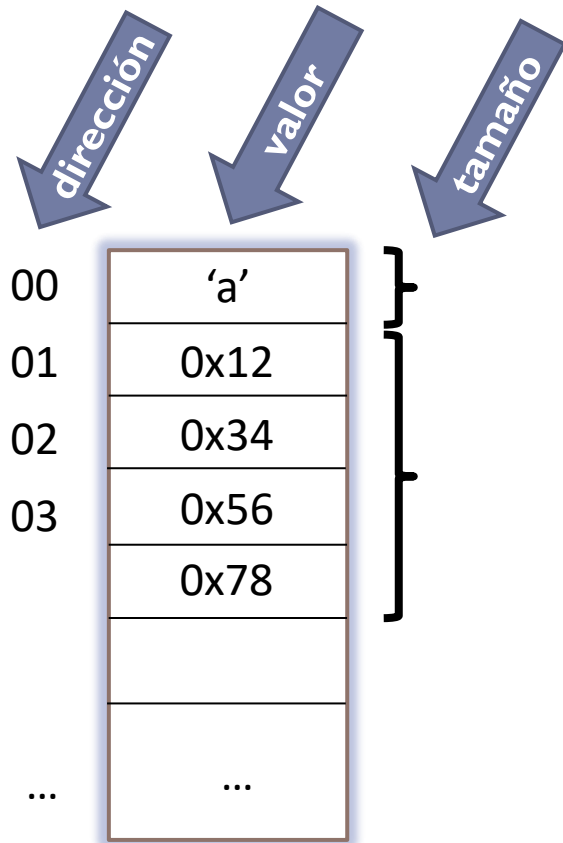
# Contenidos

---

1. **Introducción:**
  1. **Modelo abstracto y definiciones básicas.**
  2. Mapa de memoria de un proceso: regiones de memoria.
2. **Funciones del gestor de memoria.**
  1. Particionamiento de memoria.
  2. Algoritmos de gestión de memoria.
3. **Memoria virtual.**
4. **Servicios del gestor de memoria.**



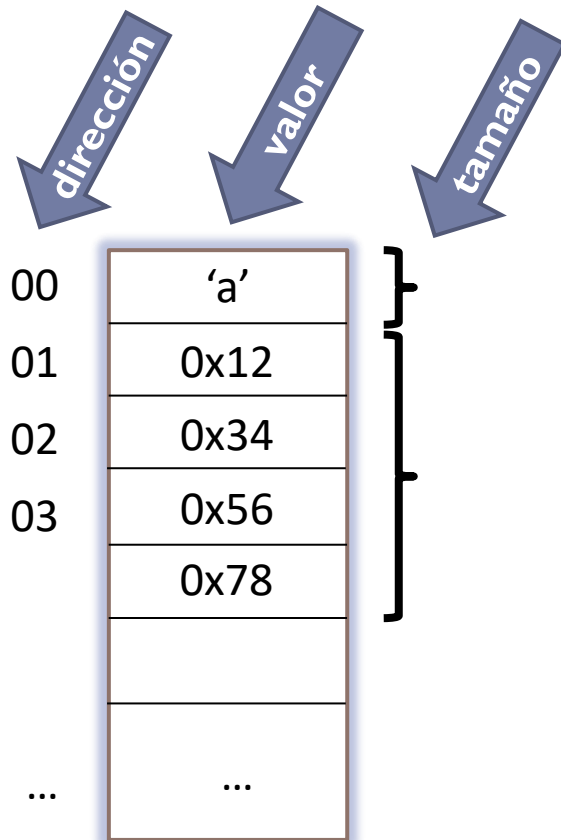
## Modelo abstracto dirección, valor y tamaño



- ▶ **Valor**
  - ▶ Elemento guardado en memoria a partir de una dirección, y que ocupa un cierto tamaño para ser almacenada.
- ▶ **Dirección**
  - ▶ Número que identifica la posición de memoria (celda) a partir de la cual se almacena el valor de un cierto tamaño.
- ▶ **Tamaño**
  - ▶ Número de bytes necesarios a partir de la dirección de comienzo para almacenar el valor.

# Modelo abstracto interfaz funcional

REPASO



- ▶ **valor = read (dirección, tamaño)**
- ▶ **write (dirección, valor, tamaño)**

Antes de acceder a una dirección, tiene que apuntar a una zona de memoria previamente reservada (tener autorización/permiso).

# Definiciones básicas

---

- ▶ Elementos

Programa

Imagen de proceso

Proceso

- ▶ Entornos

monoprogramados

multiprogramados



# Definiciones básicas

---

- ▶ Elementos

Programa

Imagen de proceso

Proceso

- ▶ Entornos

monoprogramados

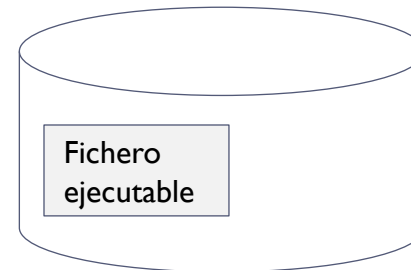
multiprogramados

# Programa y proceso

---

- ▶ **Programa:** conjunto de datos e instrucciones ordenadas que permiten realizar una tarea o trabajo específico.

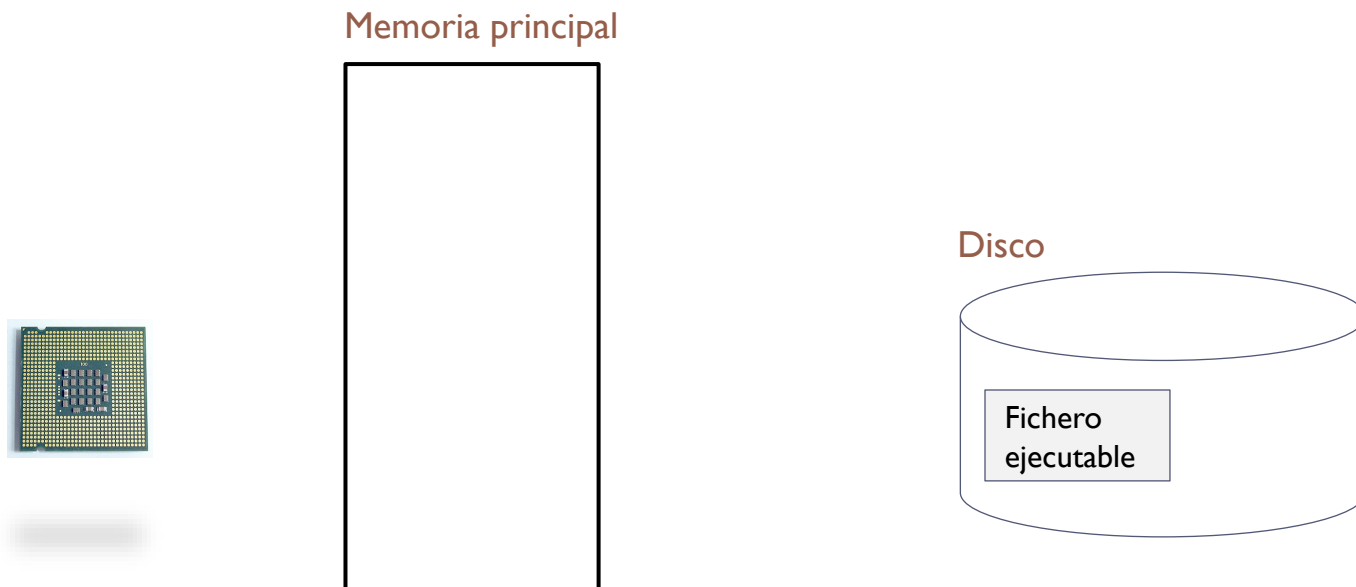
Disco



# Programa y proceso

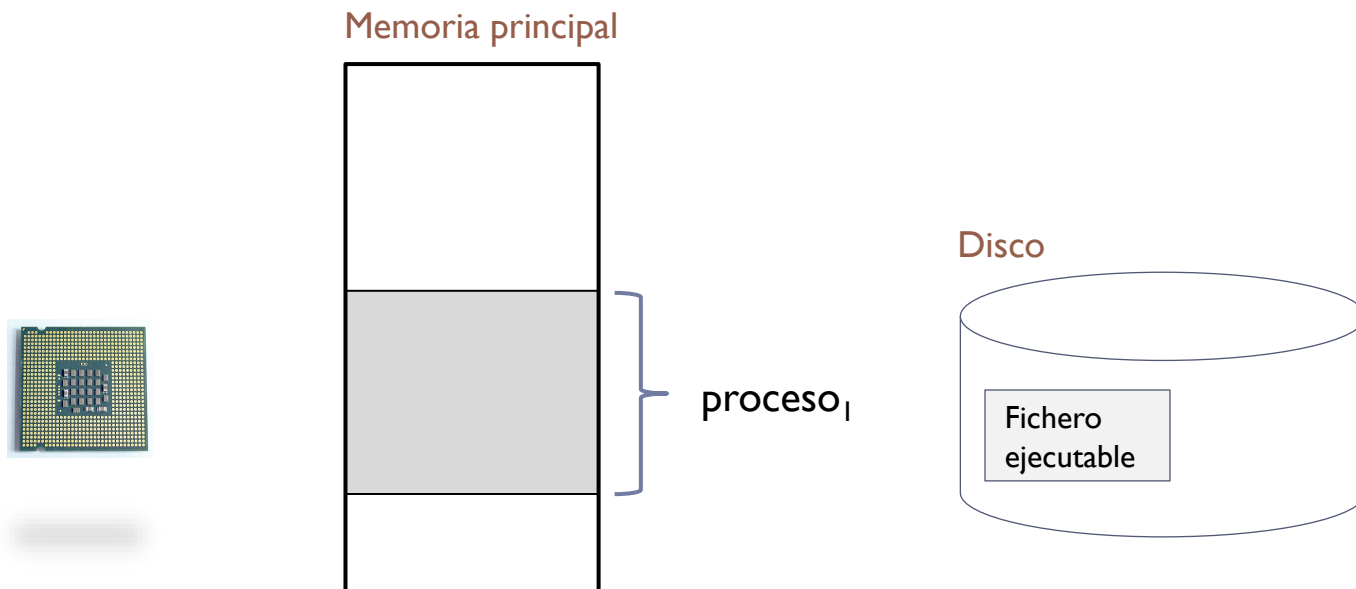
---

- ▶ **Programa:** conjunto de datos e instrucciones ordenadas que permiten realizar una tarea o trabajo específico.
  - ▶ Para su ejecución, ha de estar en memoria.



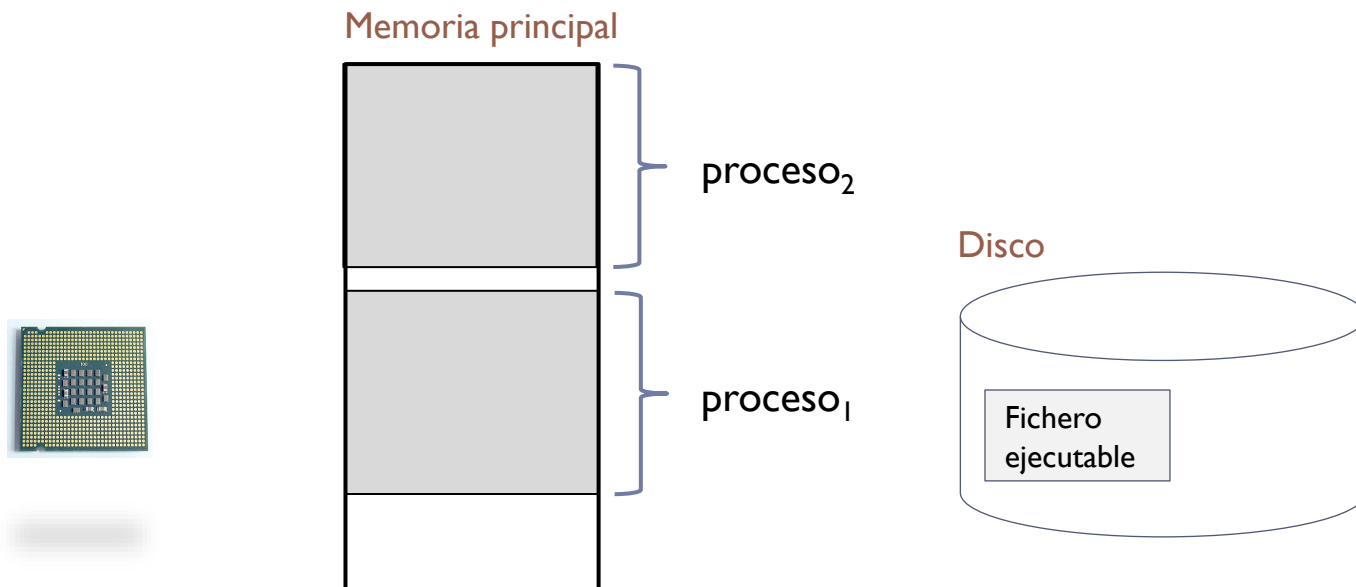
# Programa y proceso

- ▶ **Proceso:** programa en ejecución.



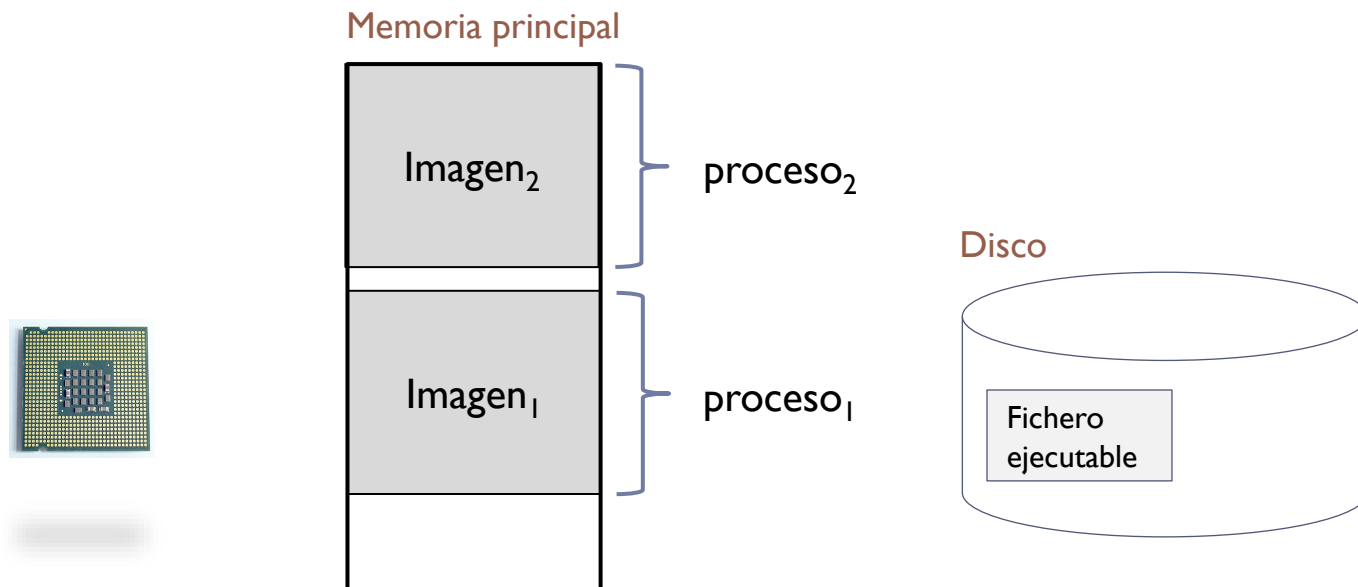
# Programa y proceso

- ▶ **Proceso:** programa en ejecución.
  - ▶ Es posible un mismo programa ejecutarlo varias veces (lo que da lugar a varios procesos).



## Imagen de un proceso

- ▶ **Imagen de memoria:** conjunto de direcciones de memoria asignadas al programa que se está ejecutando (y contenido).



# Definiciones básicas

---

- ▶ Elementos

Programa

Imagen de proceso

Proceso

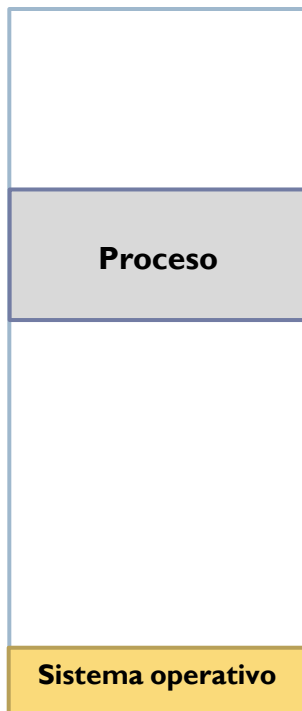
- ▶ Entornos

monoprogramados

multiprogramados

# Sistemas monoprogramados

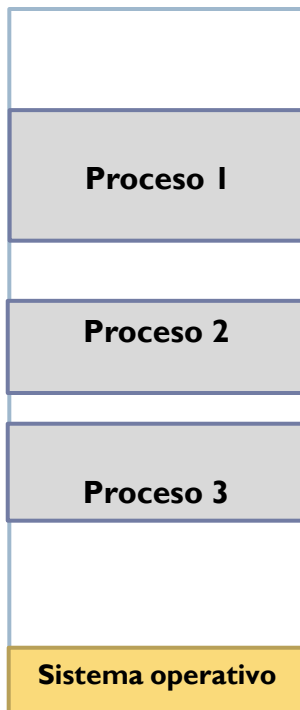
---



- ▶ Ejecución un proceso como máximo
- ▶ Memoria compartida entre el sistema operativo y el proceso
- ▶ Ej.: MS-DOS, DR-DOS, etc.



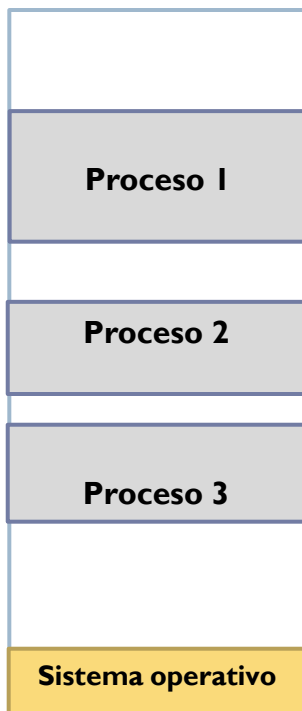
# Sistemas multiprogramados



- ▶ Se mantiene más de un proceso en memoria
- ▶ Se mejorar ocupación de CPU:
  - ▶ proceso bloqueado -> ejecuta otro
- ▶ La gestión de memoria es una tarea de optimización bajo restricciones
- ▶ Ej.: **Unix, Windows NT, etc.**

# Sistemas multiprogramados

---

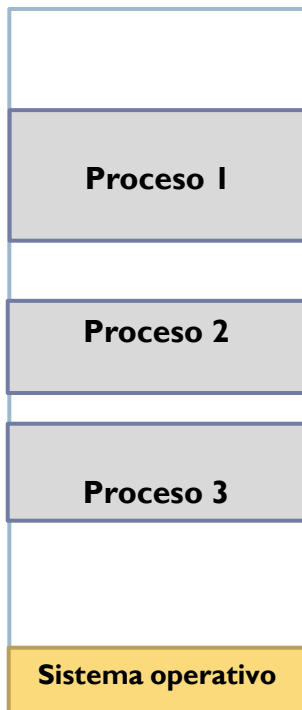


- ▶ Se mantiene más de un proceso en memoria
- ▶ Se mejorar ocupación de CPU:
  - ▶ proceso bloqueado -> ejecuta otro
- ▶ La gestión de memoria es una tarea de optimización bajo restricciones
- ▶ Ej.: **Unix, Windows NT, etc.**

# Sistemas multiprogramados

## ejemplo de cálculo de utilización

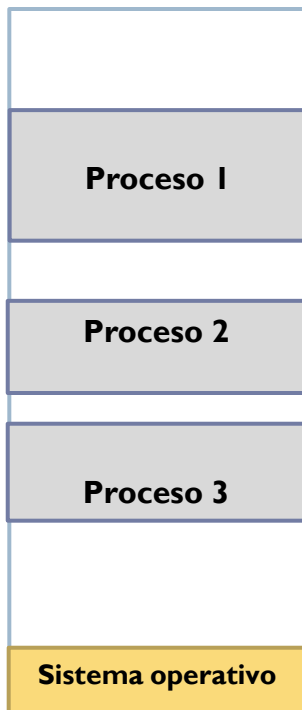
---



- ▶  $n = 5$  procesos independientes
- ▶  $p = 0,8$  del tiempo bloqueado (20% en CPU)
- ▶ **utilización =  $5 * 20\% \rightarrow 100\%$**

# Sistemas multiprogramados

## ejemplo de cálculo de utilización



- ▶  $p$  = % tiempo un proceso está bloqueado
- ▶  $p^n$  = probabilidad de que  $n$  procesos independientes estén todos bloqueados
- ▶  $1 - p^n$  = probabilidad de que la CPU **no** esté ociosa (no *idle*)
- ▶  $n = 5$  procesos independientes
- ▶  $p = 0,8$  del tiempo bloqueado (20% en CPU)
- ▶ **utilización =  $5 * 20\% \rightarrow 100\%$**
- ▶ **utilización =  $1 - 0,8^5 \rightarrow 67\%$**
- ▶  $1 - 0,8^{10} \rightarrow 89\%$

# Contenidos

---

## 1. Introducción:

1. Modelo abstracto y definiciones básicas.
2. **Mapa de memoria de un proceso: regiones de memoria.**

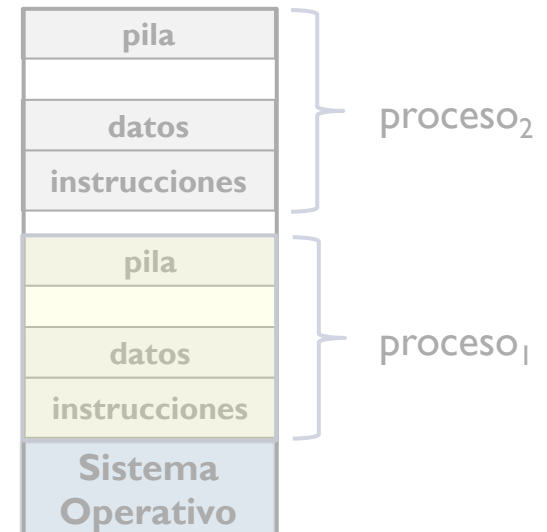
## 2. Funciones del gestor de memoria.

1. Particionamiento de memoria.
2. Algoritmos de gestión de memoria.

## 3. Memoria virtual.

## 4. Servicios del gestor de memoria.

Memoria principal

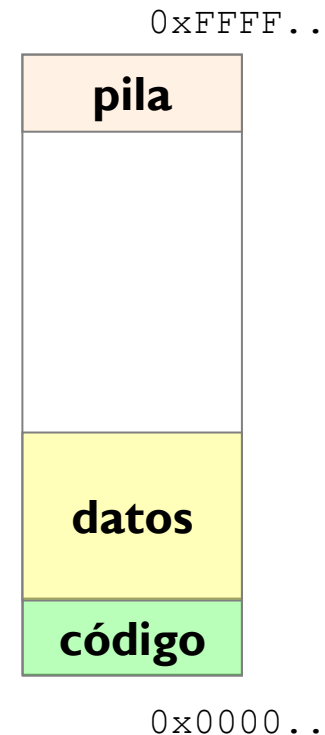


# Organización lógica (de los programas)

## modelo de memoria de un proceso

---

- ▶ Un proceso está formado por una serie de regiones.
  - ▶ Unidad de trabajo al proteger, compartir, etc.
- ▶ Una **región** es una **zona contigua del espacio de direcciones de un proceso con las mismas propiedades.**
- ▶ Principales propiedades:
  - ▶ Permisos: lectura, escritura y ejecución.
  - ▶ Compartición entre hilos: *private* o *shared*
  - ▶ Tamaño (fijo/variable)
  - ▶ Valor inicial a usar (con/sin soporte)
  - ▶ Creación estática o dinámica
  - ▶ Sentido de crecimiento
  - ▶ Dirección de comienzo y tamaño inicial



# Principales regiones de un proceso

## código (*text*)

### Código

- Estático
- Se conoce en tiempo de compilación
- Secuencia de instrucciones a ser ejecutadas

```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    .....  
    free (p)  
    ....  
    exit (0)  
}
```

0xFFFF..

**pila**

**datos estáticos  
sin valor inicial**

**datos estáticos  
con valor inicial**

**código**

0x0000..

# Principales regiones de un proceso datos (*data*)

## Variable globales

- Estáticas
- Se crean al iniciar el programa
- Existen durante ejecución
- Dirección fija en memoria y ejecutable

```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    .....  
    free (p)  
    ....  
    exit (0)  
}
```

0xFFFF..

**pila**

**datos estáticos  
sin valor inicial**

**datos estáticos  
con valor inicial**

**código**

0x0000..





# Principales regiones de un proceso

## pila (*stack*)

### Variable locales y parámetros

- Dinámicas
- Se crean al invocar la función
- Se destruyen al retornar
- Recursividad: varias instancias de una variable

```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    .....  
    free (p)  
    ....  
    exit (0)  
}
```

0xFFFF..

**pila**

**datos** estáticos  
sin valor inicial

**datos** estáticos  
con valor inicial

**código**

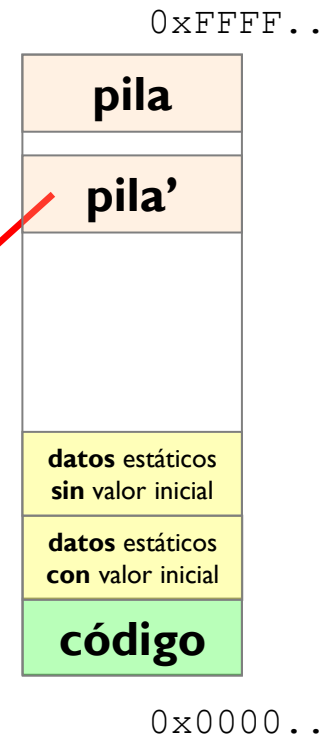
0x0000..



# Principales regiones de un proceso

## pila (*stack*)

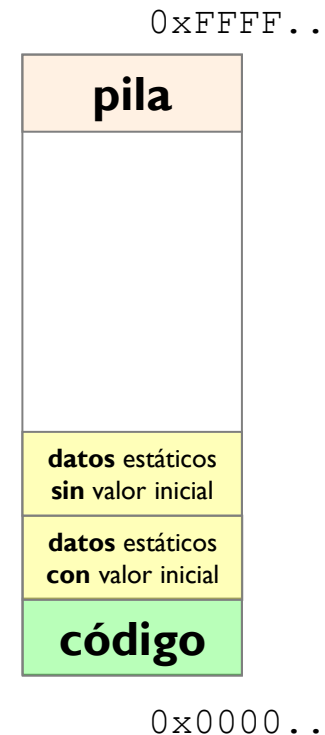
```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    ..... pthread_create(f...)  
    free (p)  
    ....  
    exit (0)  
}
```



# Organización lógica (de los programas) principales regiones de un proceso

---

- ▶ **Código** ◦ **Texto**
  - ▶ Compartida, RX, T. Fijo, soporte en ejecutable
- ▶ **Datos**
  - ▶ Con valor inicial
    - ▶ Compartido, RW, T. Fijo, soporte en ejecutable
  - ▶ Sin valor inicial
    - ▶ Compartido, RW, T. Fijo, sin soporte (rellenar 0)
- ▶ **Pila**
  - ▶ Privada, RW, T. Variable, sin soporte (rellenar 0)
  - ▶ Crece hacia direcciones más bajas
  - ▶ Pila inicial: argumentos del programa
  - ▶ Cada hilo/thread tiene su pila (similares características que la pila del proceso)

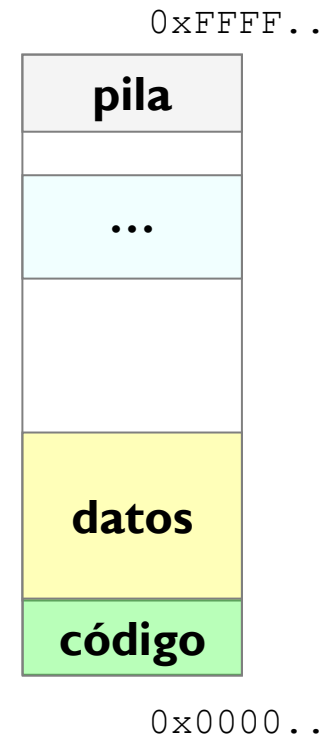


# Organización lógica (de los programas)

## modelo de memoria de un proceso

---

- ▶ **Región de Heap**
  - ▶ Soporte de memoria dinámica (malloc en C)
  - ▶ Compartido, RW, T.Variable, sin soporte (rellenar 0)
  - ▶ Crece hacia direcciones más altas
- ▶ **Ficheros proyectados**
  - ▶ Región asociada al fichero proyectado
  - ▶ Privado/Compartido, T.Variable, soporte en fichero
  - ▶ Protección especificada en proyección
- ▶ **Bibliotecas dinámicas**
  - ▶ Regiones con código y datos **proyectados**
- ▶ **Memoria compartida**
  - ▶ Entre **distintos procesos** (proyección)
  - ▶ Compartida, T.variable, Sin soporte (o swap)
  - ▶ Protección especificada en proyección



# Organización lógica (de los programas)

## modelo de memoria de un proceso

### ▶ Región de *Heap*

- ▶ Soporte de memoria dinámica (malloc en C)
- ▶ Compartido, RW, T.Variable, sin soporte (rellenar 0)
- ▶ Crece hacia direcciones más altas

### ▶ **Ficheros proyectados**

- ▶ Región asociada al fichero proyectado
- ▶ Privado/Compartido, T.Variable, soporte en fichero
- ▶ Protección especificada en proyección

### ▶ **Bibliotecas dinámicas**

- ▶ Regiones con código y datos **proyectados**

### ▶ **Memoria compartida**

- ▶ Entre **distintos procesos** (proyección)
- ▶ Compartida, T.variable, Sin soporte (o swap)
- ▶ Protección especificada en proyección



# Principales regiones de un proceso

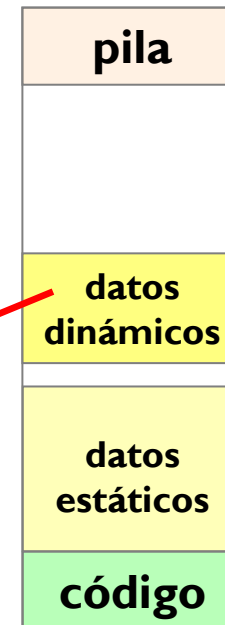
## datos dinámicos (*heap*)

### Variable dinámicas

- Variables locales o globales sin espacio asignado en tiempo de compilación
- Se reserva (y libera) espacio en tiempo de ejecución

```
int a;  
int b = 5;  
  
void f(int c) {  
    int d;  
    static e = 2;  
  
    b = d + 5;  
    .....  
    return;  
}  
  
main (int argc, char **argv) {  
    char *p;  
    p = (char *) malloc (1024)  
    f(b)  
    .....  
    free (p)  
    ....  
    exit (0)  
}
```

0xFFFF..



0x0000..

# Organización lógica (de los programas)

## modelo de memoria de un proceso

### ▶ Región de *Heap*

- ▶ Soporte de memoria dinámica (malloc en C)
- ▶ Compartido, RW, T.Variable, sin soporte (rellenar 0)
- ▶ Crece hacia direcciones más altas

### ▶ **Ficheros proyectados**

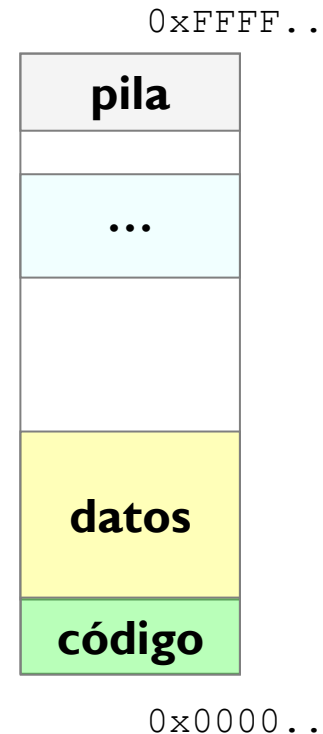
- ▶ Región asociada al fichero proyectado
- ▶ Privado/Compartido, T.Variable, soporte en fichero
- ▶ Protección especificada en proyección

### ▶ **Bibliotecas dinámicas**

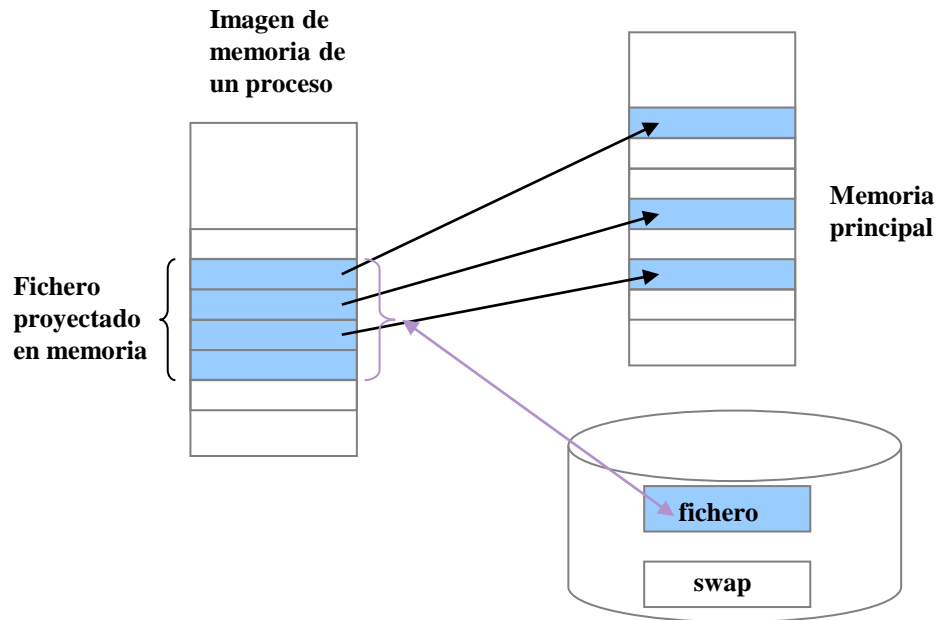
- ▶ Regiones con código y datos **proyectados**

### ▶ **Memoria compartida**

- ▶ Entre **distintos procesos** (proyección)
- ▶ Compartida, T.variable, Sin soporte (o swap)
- ▶ Protección especificada en proyección



# Ficheros proyectados en memoria



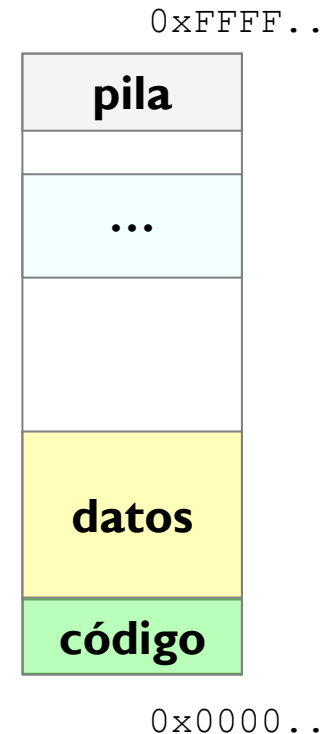
- ▶ Una región de un proceso se asocia a un fichero
- ▶ Habrá páginas del fichero en memoria principal
- ▶ El proceso direcciona dentro del fichero con instrucciones de acceso a memoria (en lugar de read/write)



# Organización lógica (de los programas)

## modelo de memoria de un proceso

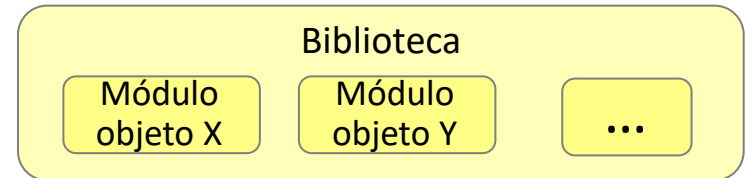
- ▶ **Región de Heap**
  - ▶ Soporte de memoria dinámica (malloc en C)
  - ▶ Compartido, RW, T.Variable, sin soporte (rellenar 0)
  - ▶ Crece hacia direcciones más altas
- ▶ **Ficheros proyectados**
  - ▶ Región asociada al fichero proyectado
  - ▶ Privado/Compartido, T.Variable, soporte en fichero
  - ▶ Protección especificada en proyección
- ▶ **Bibliotecas dinámicas**
  - ▶ Regiones con código y datos **proyectados**
- ▶ **Memoria compartida**
  - ▶ Entre **distintos procesos** (proyección)
  - ▶ Compartida, T.variable, Sin soporte (o swap)
  - ▶ Protección especificada en proyección



# Biblioteca estática vs dinámica

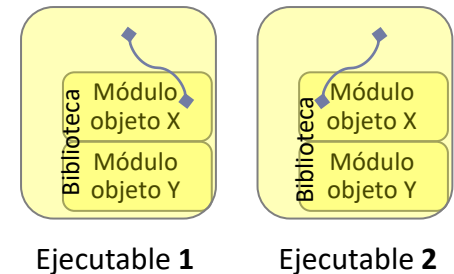
## ▶ Biblioteca

- ▶ Colección de módulos objetos relacionados



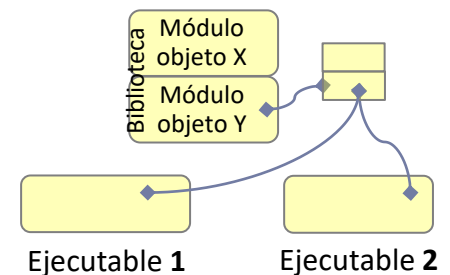
## ▶ Biblioteca **estática**

- ▶ Carga y montaje en tiempo de compilación.
- ▶ [V] Ejecutable autocontenido
- ▶ [I] Desperdicio de memoria por múltiples copias, Actualizar precisa recompilar los ejecutables



## ▶ Biblioteca **dinámica**

- ▶ Carga y montaje en tiempo de ejecución.
  - ▶ Ejecutable indica bibliotecas dinámicas a cargar.
  - ▶ Uso de referencias indirectas mediante una tabla.
- ▶ [V] Ejecutable menor + evita duplicación + facilita actualizaciones (y ofrece versionado)
- ▶ [I] Algo más tiempo de ejecución + dependencia



# Ejemplo: biblioteca estática

2

```
#include <stdio.h>

extern void hola ( void );

int main()
{
    hola ();
    return 0;
}
```

```
gcc -Wall -g -o main main.c -lhola -L./main
```

1

```
#include <stdio.h>
#include <stdlib.h>

void hola ( void )
{
    printf("hola") ;
}
```

```
gcc -Wall -g -o libhola.o -c libhola.c  
ar rcs libhola.a libhola.o
```

# Ejemplo: biblioteca dinámica (carga implícita)

2

```
#include <stdio.h>

extern void hola ( void );

int main()
{
    hola ();
    return 0;
}
```

```
gcc -Wall -g -o main main.c -lhola -L./
env LD_LIBRARY_PATH=$LD_LIBRARY_PATH:. ./main
```

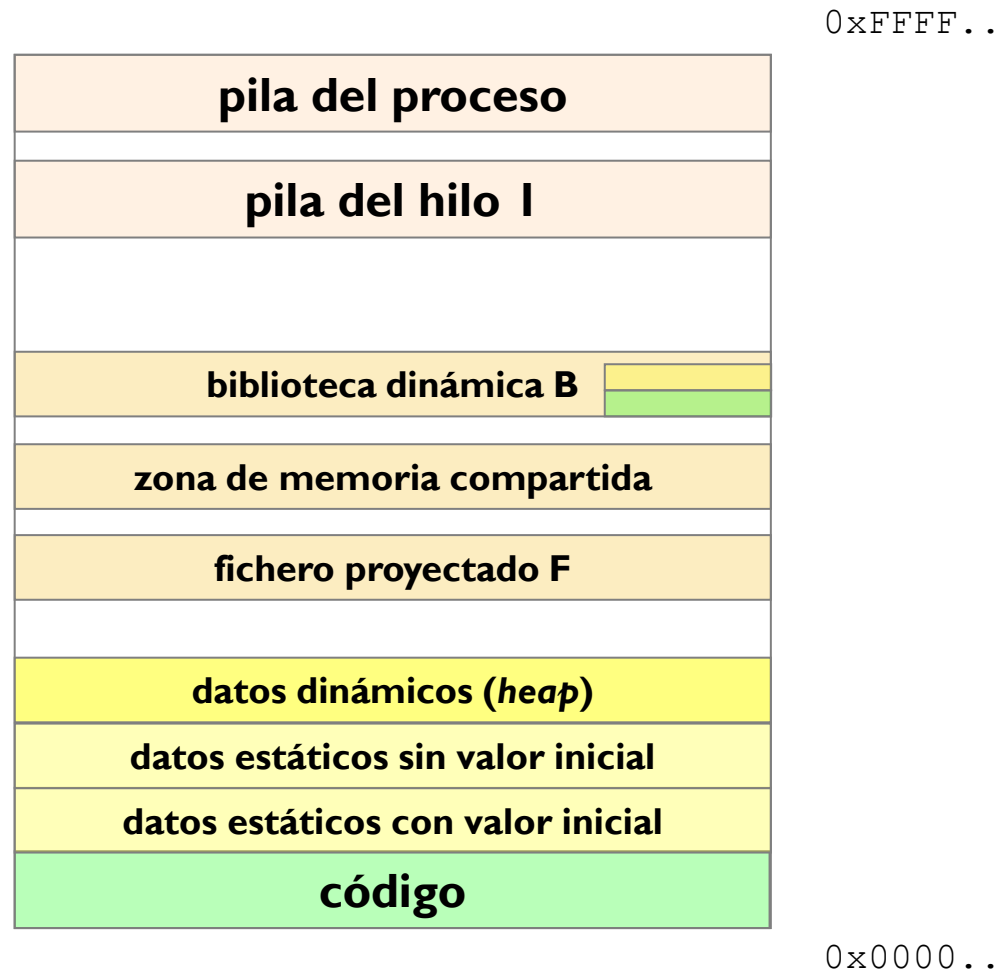
1

```
#include <stdio.h>
#include <stdlib.h>

void hola ( void )
{
    printf("hola") ;
}
```

```
gcc -Wall -g -fPIC -o libhola.o -c libhola.c
gcc -shared -Wl,-soname,libhola.so \
    -o libhola.so.1.0 libhola.o
ln -s libhola.so.1.0 libhola.so
```

# Ejemplo de mapa de memoria

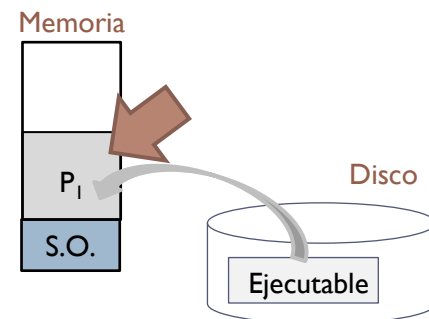


# Características de regiones

## resumen

Región	Soporte	Protección	Comp./Priv.	Tamaño	Valor inicial
Código	Fichero ejec.	RX	Compartida	Fijo	<desde soporte>
Datos con v.i.	Fichero ejec.	RW	Privada	Fijo	<desde soporte>
Datos sin v.i.	Sin soporte	RW	Privada	Fijo	Rellenar con 0
Pila	Sin soporte	RW	Privada	Variable	Args. del programa, dirs. decrecientes
Heap	Sin soporte	RW	Comp./priv.	Variable	
Fich. Proyect.	Fichero	Por usuario	Comp./priv.	Variable	<desde soporte>
Mem. Comp.	Sin soporte	Por usuario	Compartida	Variable	

# Inspeccionar un proceso

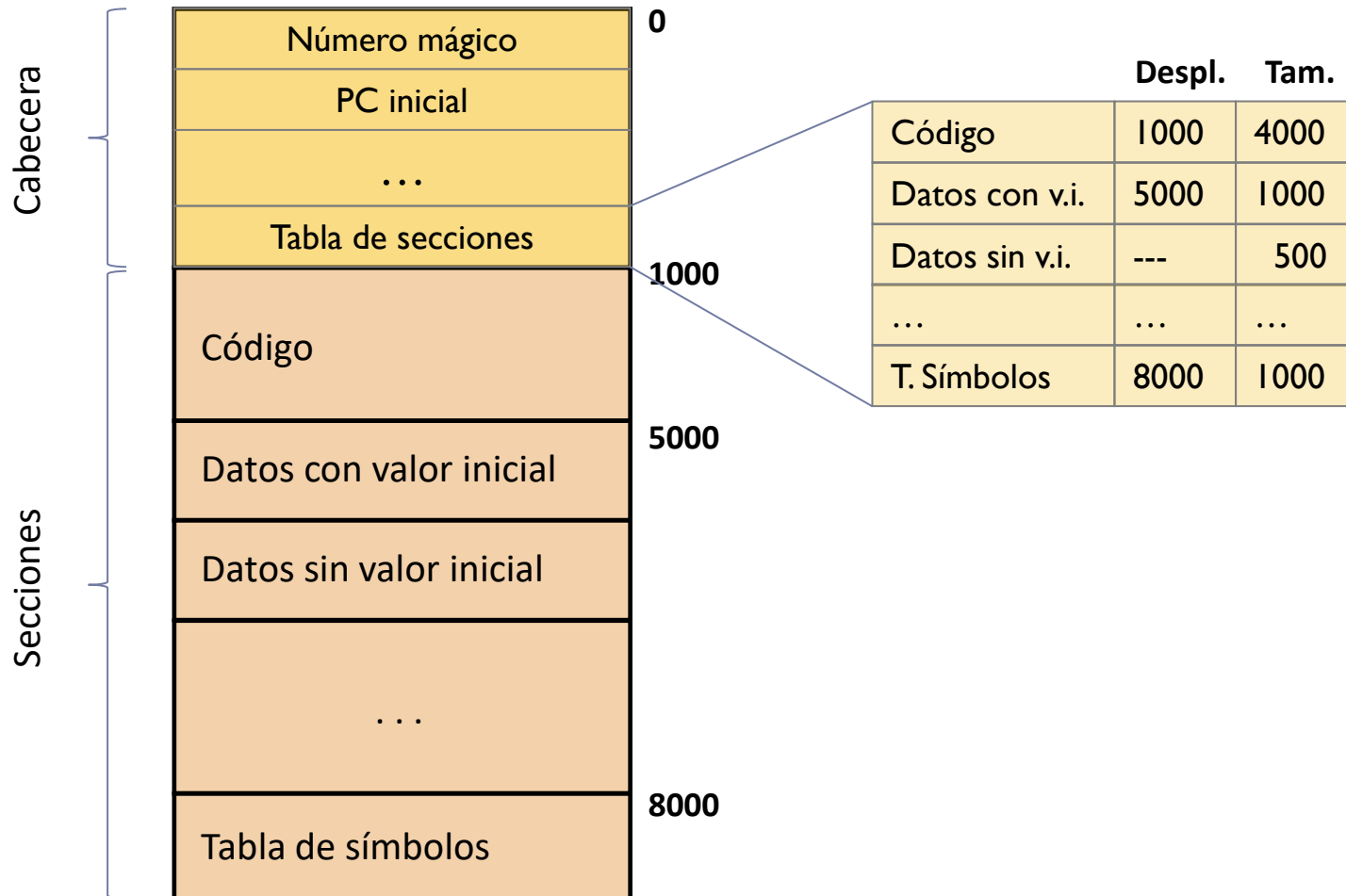


## ► Detalles de las secciones de un proceso:

```
acaldero@patata:~/infodso/$ cat /proc/1/maps
b7688000-b7692000 r-xp 00000000 08:02 1491      /lib/libnss_files-2.12.1.so
b7692000-b7693000 r--p 00009000 08:02 1491      /lib/libnss_files-2.12.1.so
b7693000-b7694000 rw-p 0000a000 08:02 1491      /lib/libnss_files-2.12.1.so
b7694000-b769d000 r-xp 00000000 08:02 3380      /lib/libnss_nis-2.12.1.so
b769d000-b769e000 r--p 00008000 08:02 3380      /lib/libnss_nis-2.12.1.so
b769e000-b769f000 rw-p 00009000 08:02 3380      /lib/libnss_nis-2.12.1.so
b769f000-b76b2000 r-xp 00000000 08:02 1414      /lib/libnsl-2.12.1.so
b76b2000-b76b3000 r--p 00012000 08:02 1414      /lib/libnsl-2.12.1.so
b76b3000-b76b4000 rw-p 00013000 08:02 1414      /lib/libnsl-2.12.1.so
b76b4000-b76b6000 rw-p 00000000 00:00 0
..
b78b7000-b78b8000 r-xp 00000000 00:00 0          [vdso]
b78b8000-b78d4000 r-xp 00000000 08:02 811      /lib/ld-2.12.1.so
b78d4000-b78d5000 r--p 0001b000 08:02 811      /lib/ld-2.12.1.so
b78d5000-b78d6000 rw-p 0001c000 08:02 811      /lib/ld-2.12.1.so
b78d6000-b78ef000 r-xp 00000000 08:02 1699      /sbin/init
b78ef000-b78f0000 r--p 00019000 08:02 1699      /sbin/init
b78f0000-b78f1000 rw-p 0001a000 08:02 1699      /sbin/init
b81e5000-b8247000 rw-p 00000000 00:00 0          [heap]
bf851000-bf872000 rw-p 00000000 00:00 0          [stack]
```

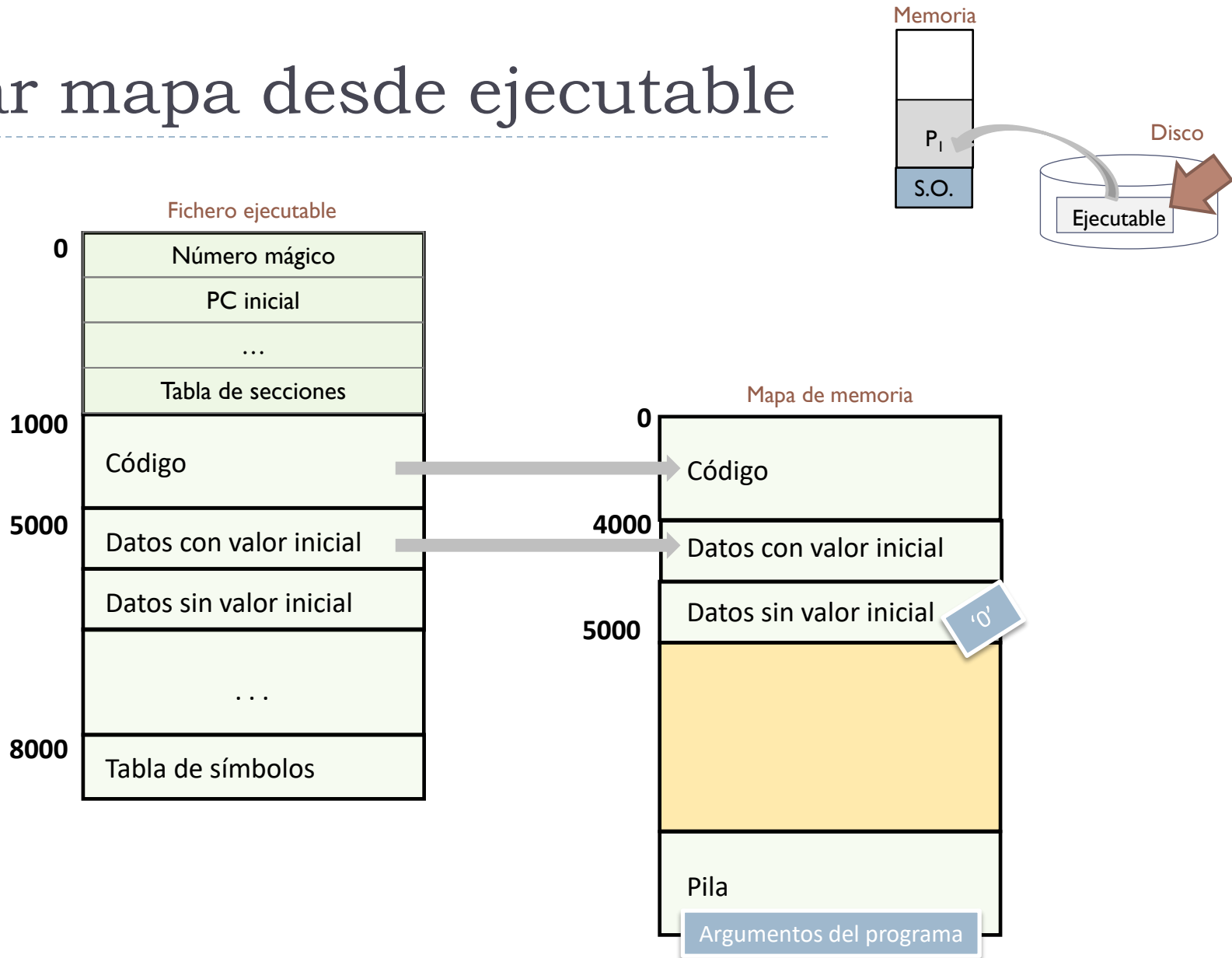
direccion      perm.    offset    dev    nodo-i    nombre

# Ejemplo de formato de ejecutable

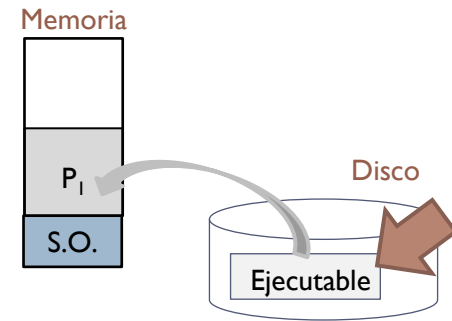




# Crear mapa desde ejecutable



# Inspeccionar un ejecutable



## ► Detalles de las secciones de un ejecutable:

```
acaldero@patata:~/infodso/$ objdump -x main.exe
...

Program Header:
...
DYNAMIC off      0x00000f20 vaddr 0x08049f20 paddr 0x08049f20 align 2**2
          filesz 0x000000d0 memsz 0x000000d0 flags rw-
...
STACK off       0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**2
          filesz 0x00000000 memsz 0x00000000 flags rw-
...

Dynamic Section:
NEEDED          libdinamica.so
NEEDED          libc.so.6
INIT           0x08048368
...
```

# Fases de generación de un ejecutable

## ► Procesado en dos fases:

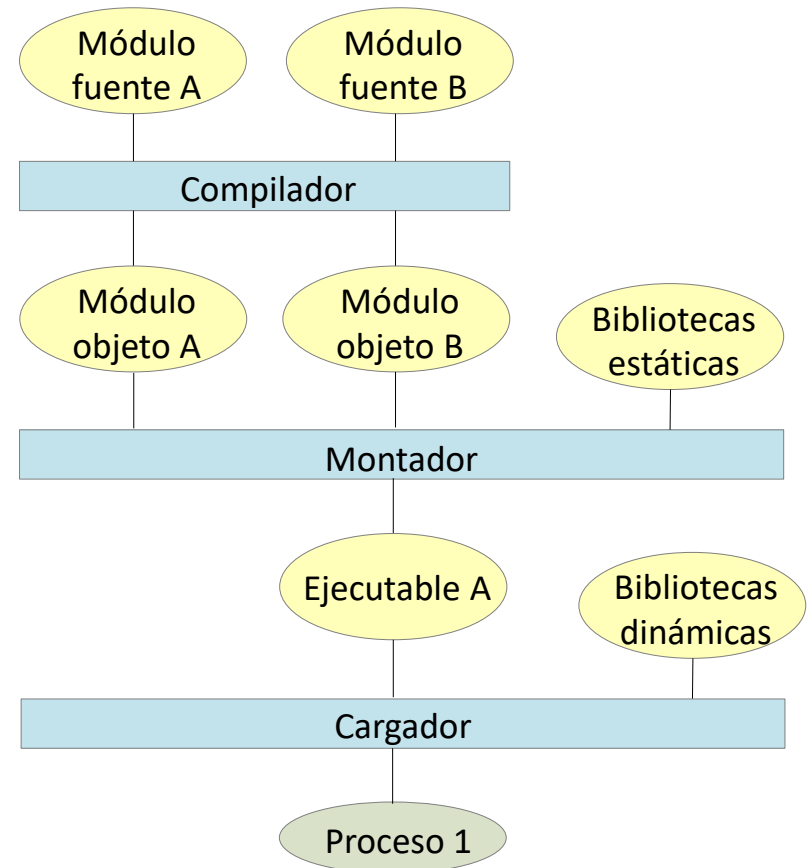
### ► Compilación:

- Generar binario y referencias
- Resolver referencias dentro cada módulo
- Genera módulo objeto

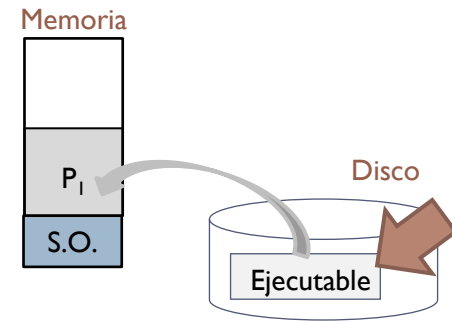
### ► Montaje (o enlace):

- Resolver referencias entre módulos objeto
- Resolver referencias a símbolos de bibliotecas
- Generar ejecutable incluyendo bibliotecas estáticas

- Durante la carga del ejecutable se usa la información para generar la imagen del proceso.



# Inspeccionar un ejecutable



## ► Dependencias de un ejecutable (lib. dinámicas):

```
acaldero@patata:~/infodso/$ ldd main.exe
linux-gate.so.1 => (0xb7797000)
libdinamica.so.1 => not found
libc.so.6 => /lib/libc.so.6 (0xb761c000)
/lib/ld-linux.so.2 (0xb7798000)
```

## ► Símbolos de un ejecutable:

```
acaldero@patata:~/infodso/$ nm main.exe
08049f20 d __DYNAMIC
08049ff4 d __GLOBAL_OFFSET_TABLE__
0804856c R __IO_stdin_used
          w __Jv_RegisterClasses
08049f10 d __CTOR_END__
08049f0c d __CTOR_LIST__
...
```

# Contenidos

---

## 1. Introducción:

1. Modelo abstracto y definiciones básicas.
2. Mapa de memoria de un proceso: regiones de memoria.

## 2. **Funciones del gestor de memoria.**

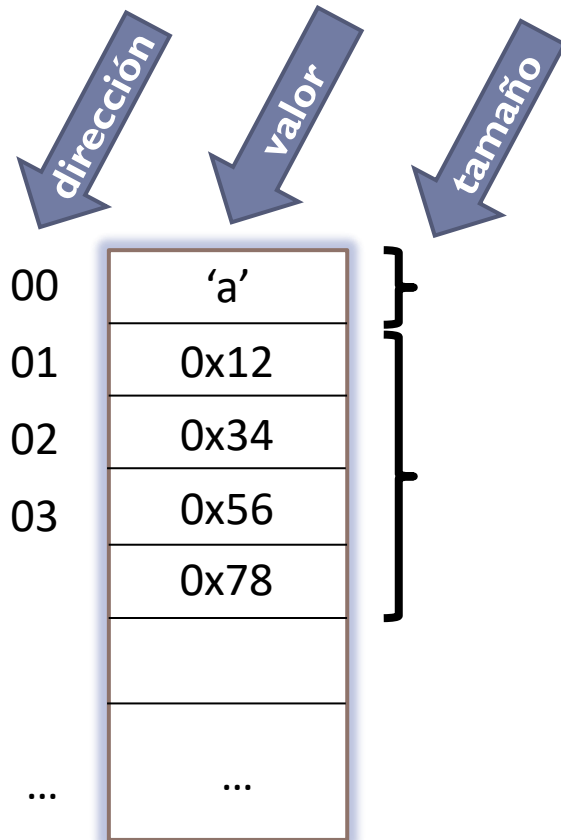
1. Particionamiento de memoria.
2. Algoritmos de gestión de memoria.

## 3. Memoria virtual.

## 4. Servicios del gestor de memoria.

# Modelo abstracto interfaz funcional

---



- ▶ **valor = read (dirección, tamaño)**
- ▶ **write (dirección, valor, tamaño)**

Antes de acceder a una dirección, tiene que apuntar a una zona de memoria previamente reservada (tener autorización/permiso).

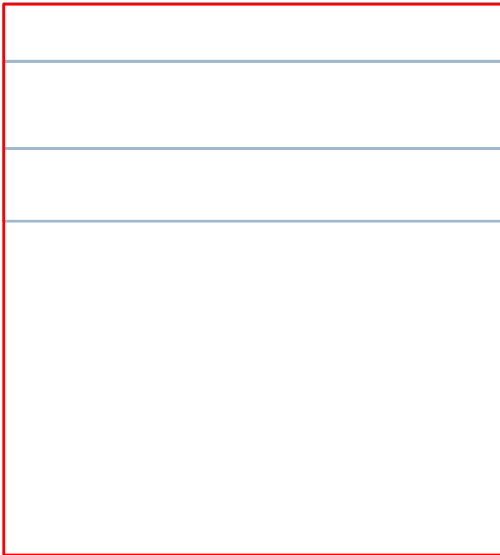


**Gestor de memoria**

# Gestor de memoria (*memory allocator*)

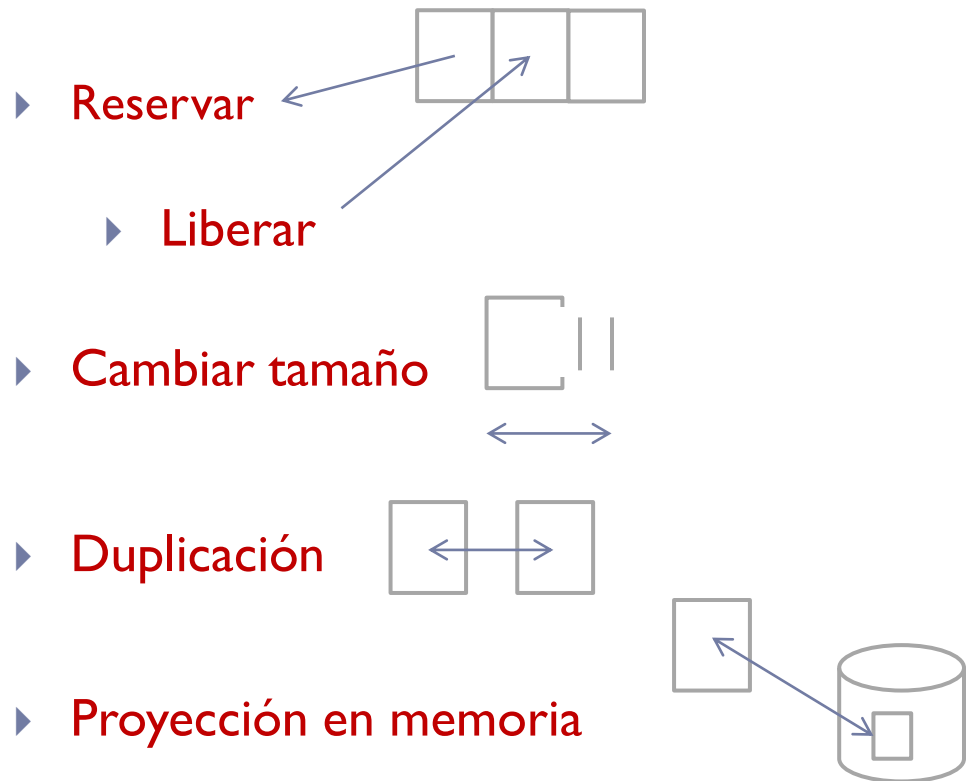
---

*memory allocator* = Bloque



# Gestor de memoria (*memory allocator*)

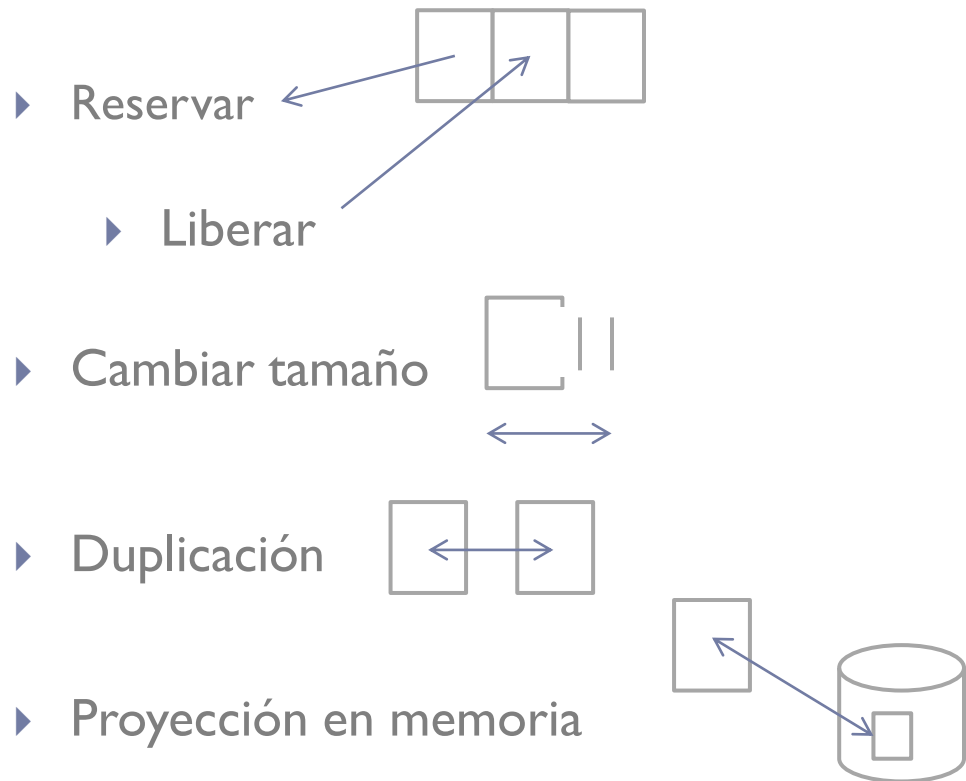
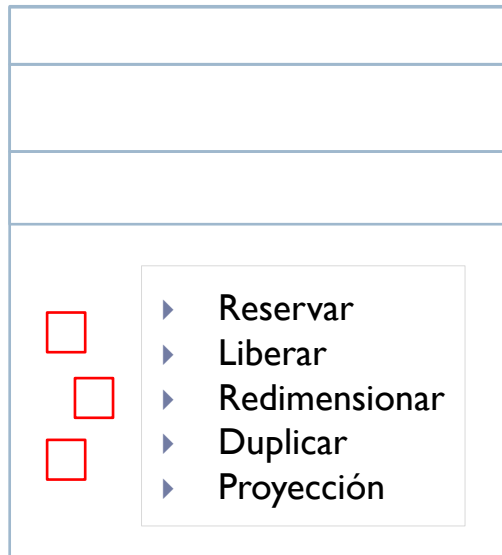
*memory allocator* = Bloque + Interfaz



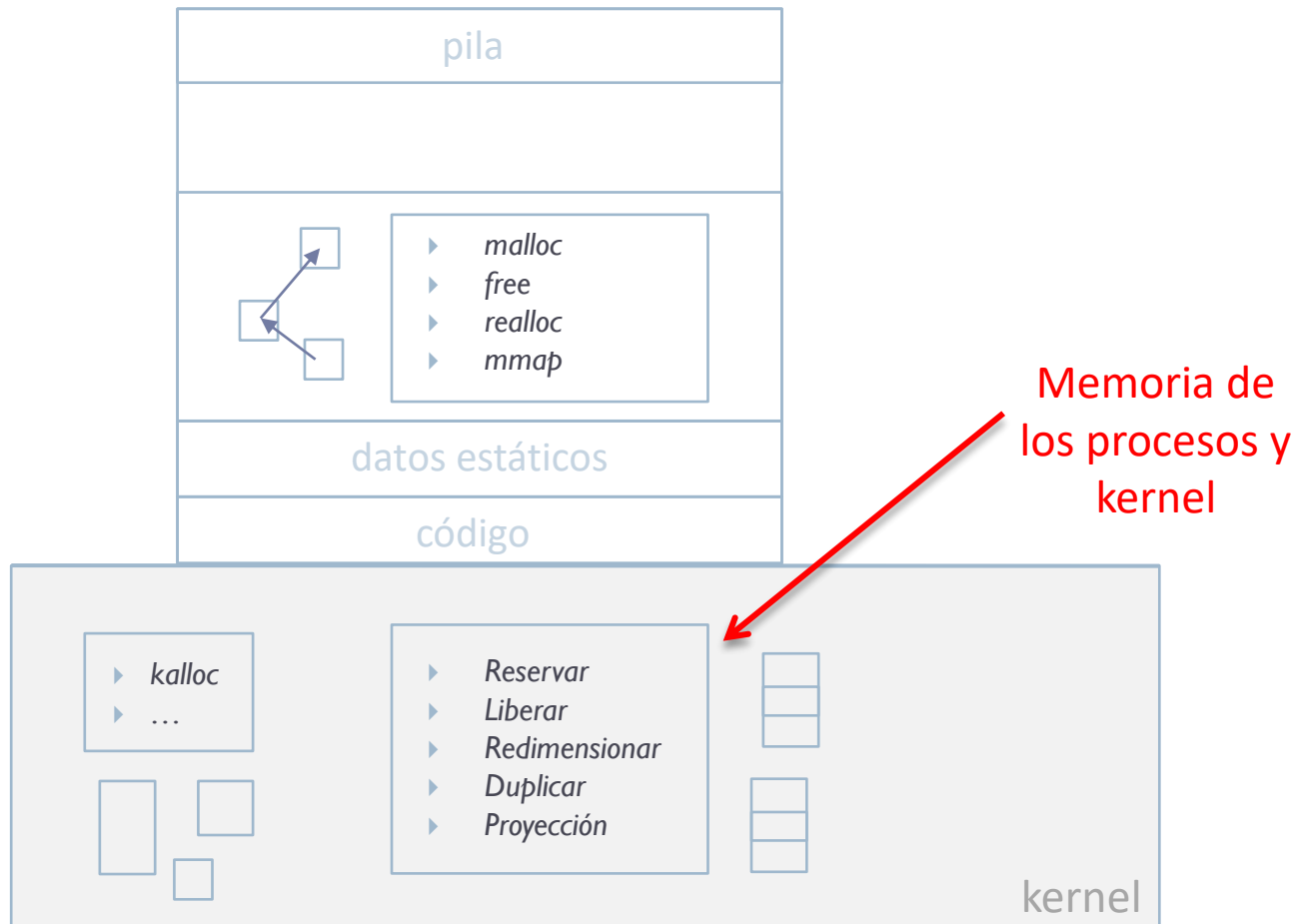


# Gestor de memoria (*memory allocator*)

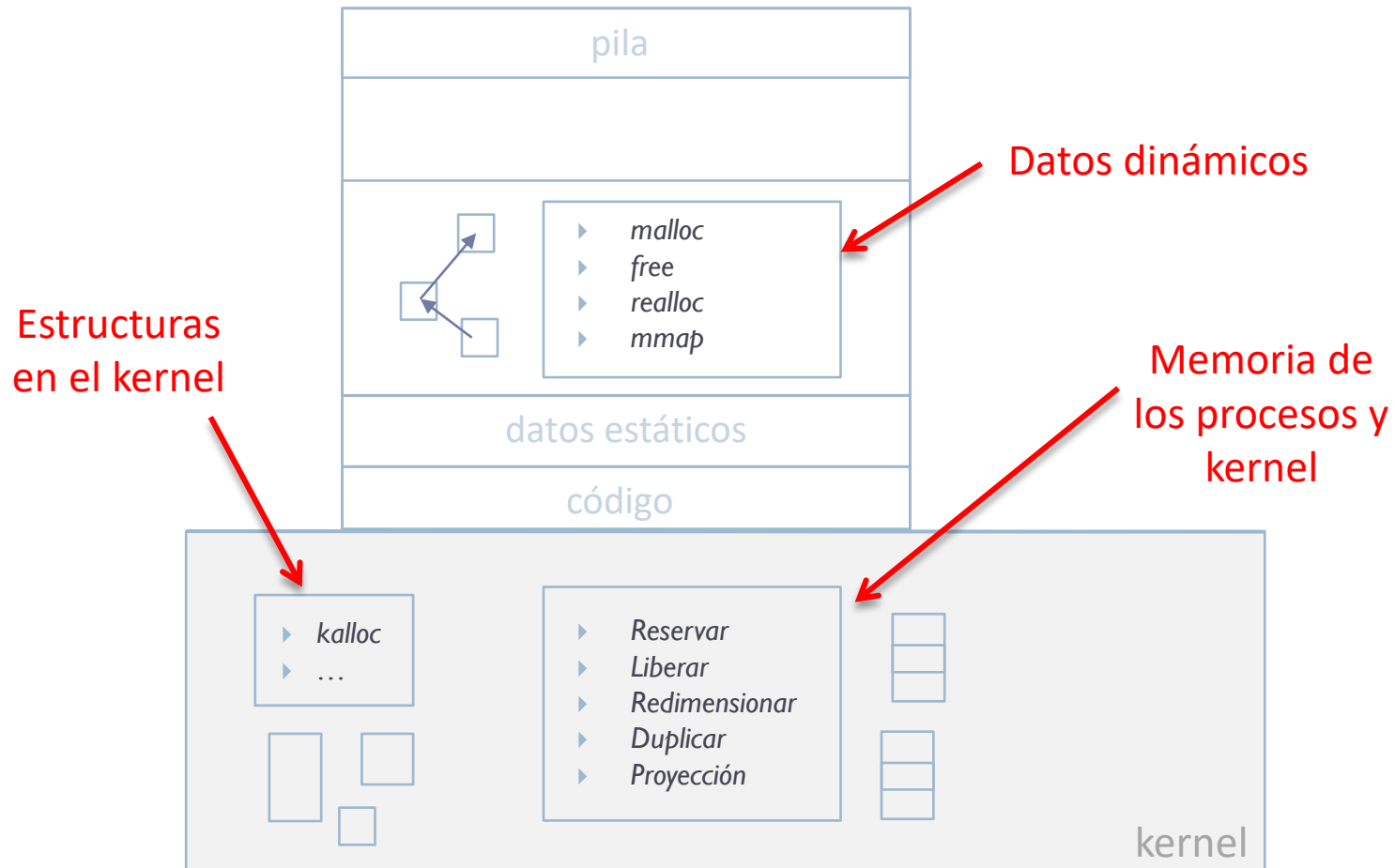
*memory allocator* = Bloque + Interfaz + Metadatos



# Gestores a varios niveles: N1

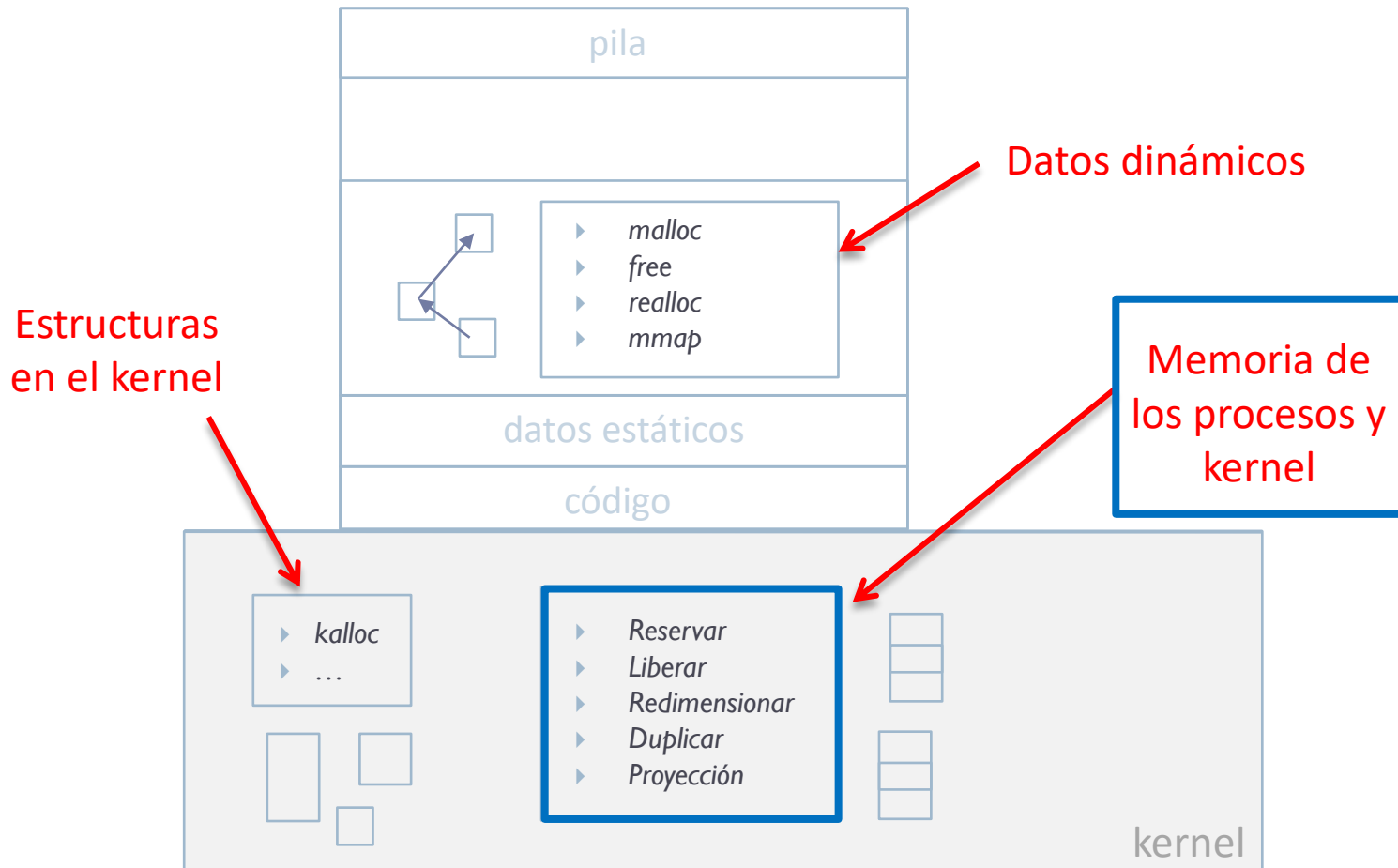


# Gestores a varios niveles: N2

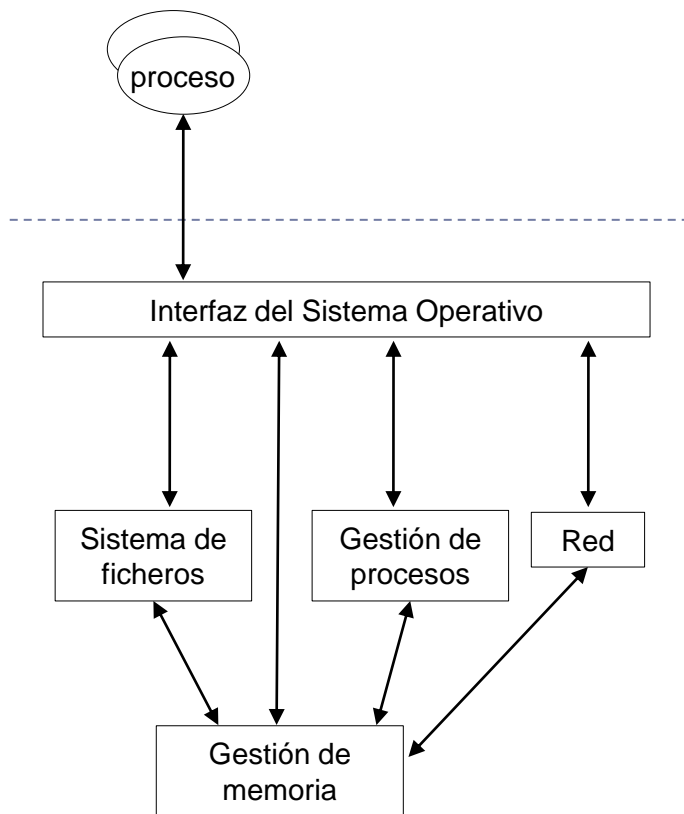


# Gestores a varios niveles

## Estudio del gestor N1 y su ámbito de trabajo



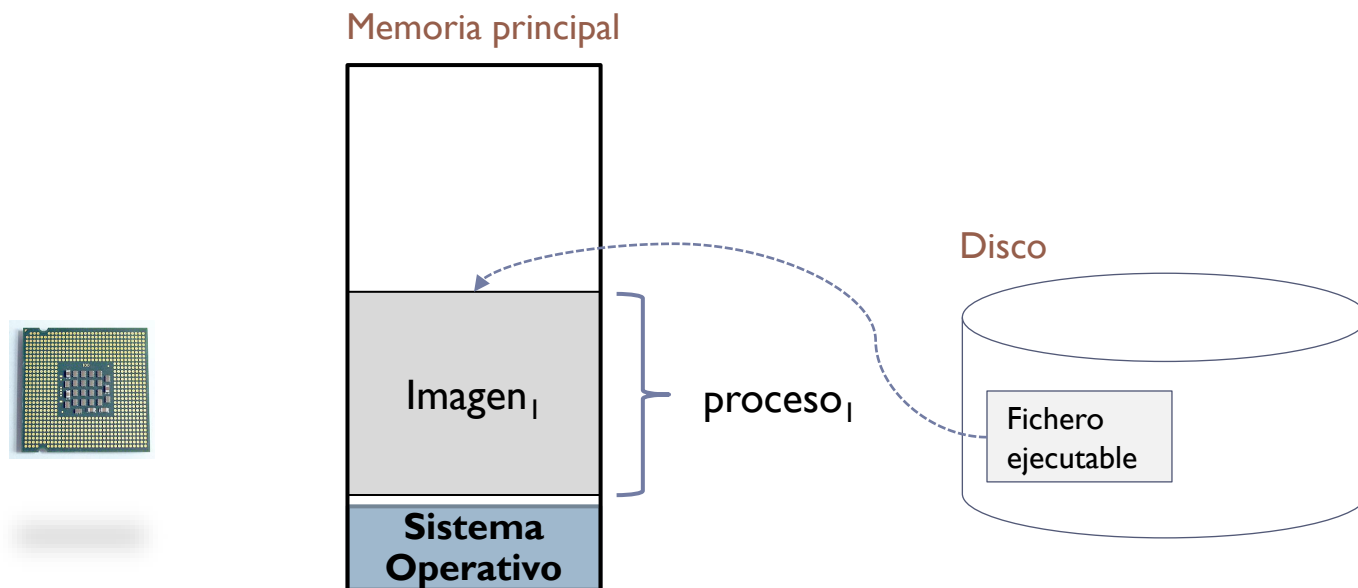
# Ámbito de la gestión de memoria (N1)



- ▶ Encargado de la gestión de la memoria entre procesos y el propio s.o. (*kernel*)
- ▶ El resto del sistema operativo es su mejor cliente...:
  - ▶ Gestión de procesos
  - ▶ Gestión de ficheros
- ▶ ...Pero es un reflejo de las necesidades de los procesos

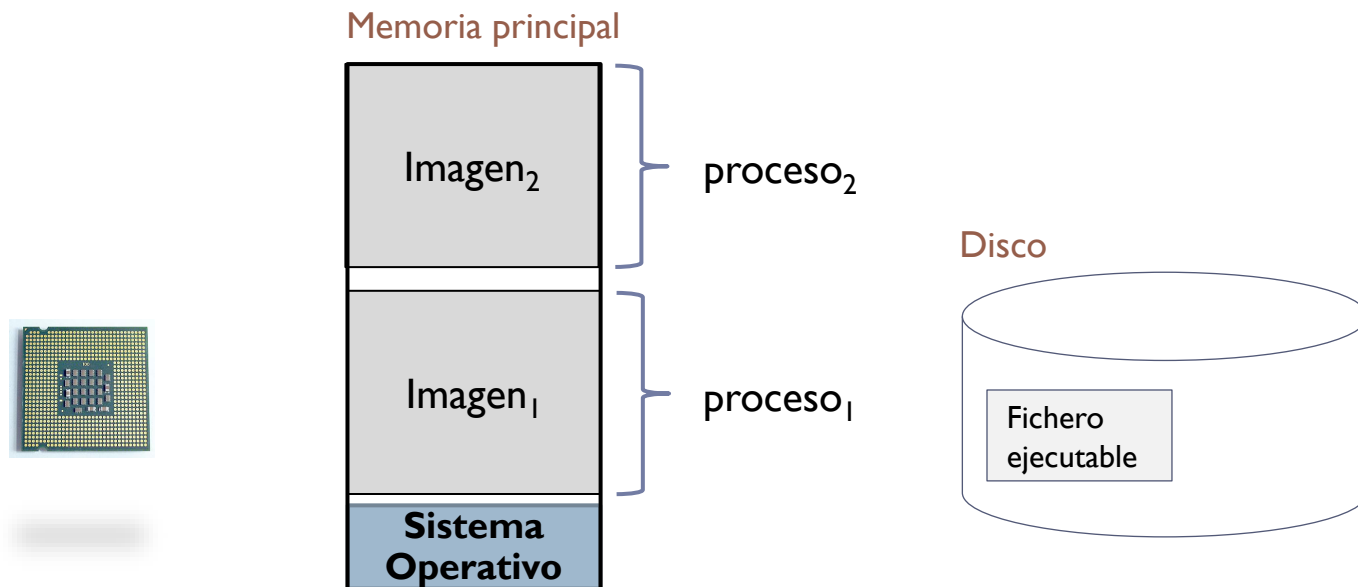
# Gestión de memoria N1: MP+**Interfaz**+Metadatos

- ▶ **Crear** un proceso a partir del ejecutable o **clonar** un proceso existente.
- ▶ **Repartir** su uso entre todos los procesos (similar a repartir la CPU).
- ▶ Poder **modificar** la imagen a petición de los procesos o **liberarla**.
- ▶ **Cargar/Descargar** parte de la imagen (tener lo necesario de **↑** procesos).
  - ▶ Trata de mejorar el uso de recursos: *Out-of-core* + ocupación de la CPU



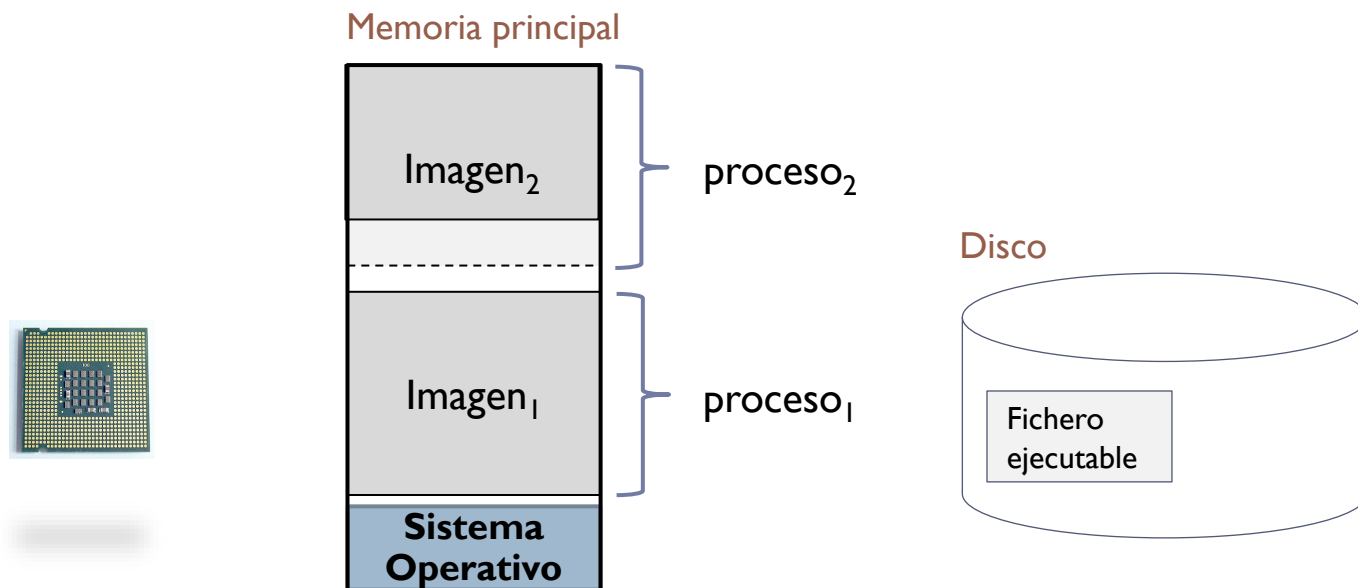
# Gestión de memoria N1: MP+**Interfaz**+Metadatos

- ▶ Crear un proceso a partir del ejecutable o **clonar** un proceso existente.
- ▶ **Repartir** su uso entre todos los procesos (similar a repartir la CPU).
- ▶ Poder **modificar** la imagen a petición de los procesos o **liberarla**.
- ▶ **Cargar/Descargar** parte de la imagen (tener lo necesario de **↑** procesos).
  - ▶ Trata de mejorar el uso de recursos: *Out-of-core* + ocupación de la CPU



# Gestión de memoria N1: MP+**Interfaz**+Metadatos

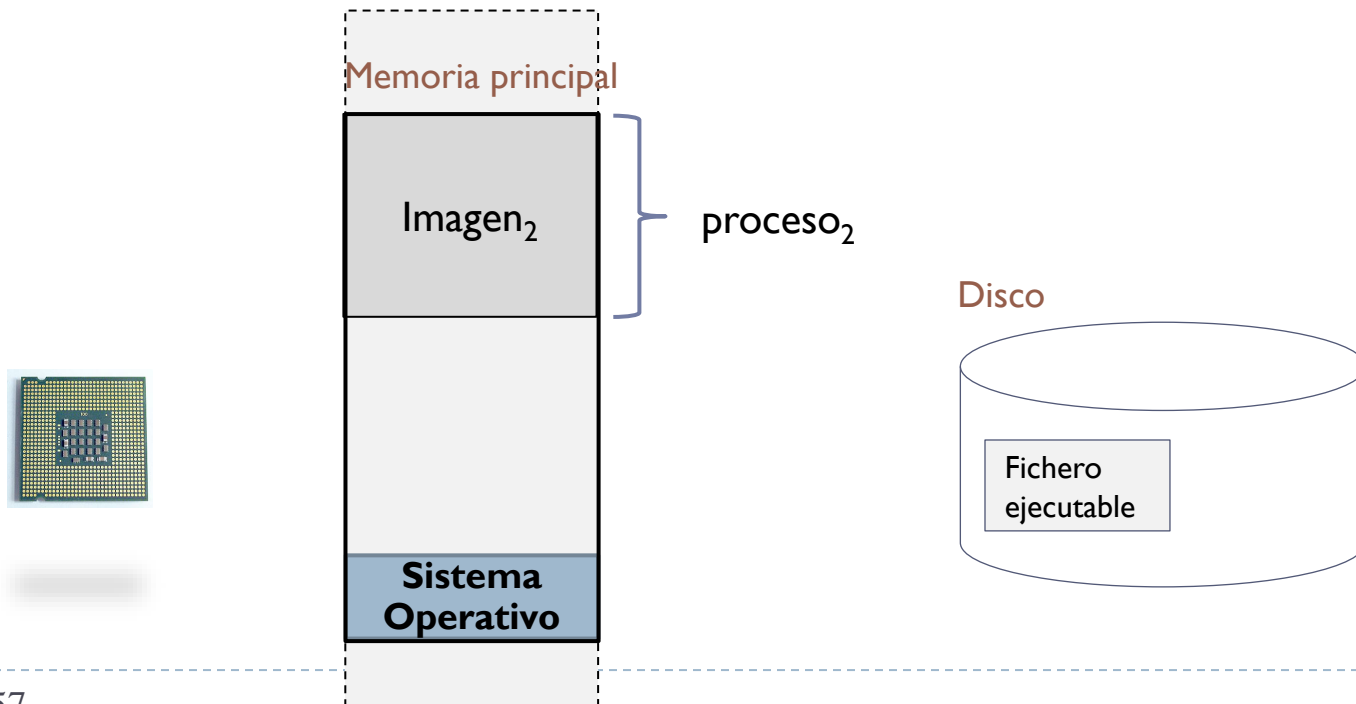
- ▶ **Crear** un proceso a partir del ejecutable o **clonar** un proceso existente.
- ▶ **Repartir** su uso entre todos los procesos (similar a repartir la CPU).
- ▶ Poder **modificar** la imagen a petición de los procesos o **liberarla**.
- ▶ **Cargar/Descargar** parte de la imagen (tener lo necesario de **↑** procesos).
  - ▶ Trata de mejorar el uso de recursos: *Out-of-core* + ocupación de la CPU



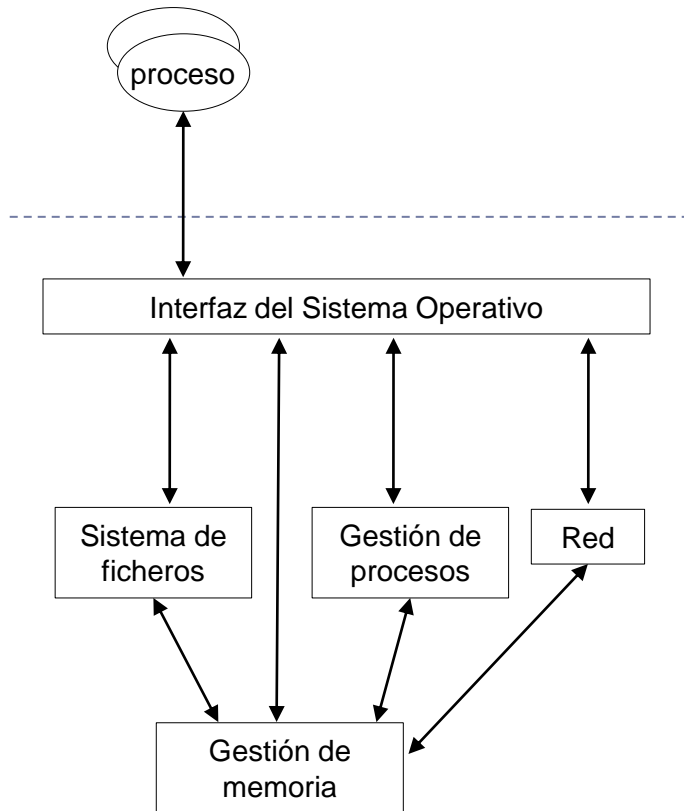


# Gestión de memoria N1: MP+**Interfaz**+Metadatos

- ▶ **Crear** un proceso a partir del ejecutable o **clonar** un proceso existente.
- ▶ **Repartir** su uso entre todos los procesos (similar a repartir la CPU).
- ▶ Poder **modificar** la imagen a petición de los procesos o **liberarla**.
- ▶ **Cargar/Descargar** parte de la imagen (tener lo necesario de **↑** procesos).
  - ▶ Trata de mejorar el uso de recursos: *Out-of-core* + ocupación de la CPU



# Ámbito de la gestión de memoria (N1)



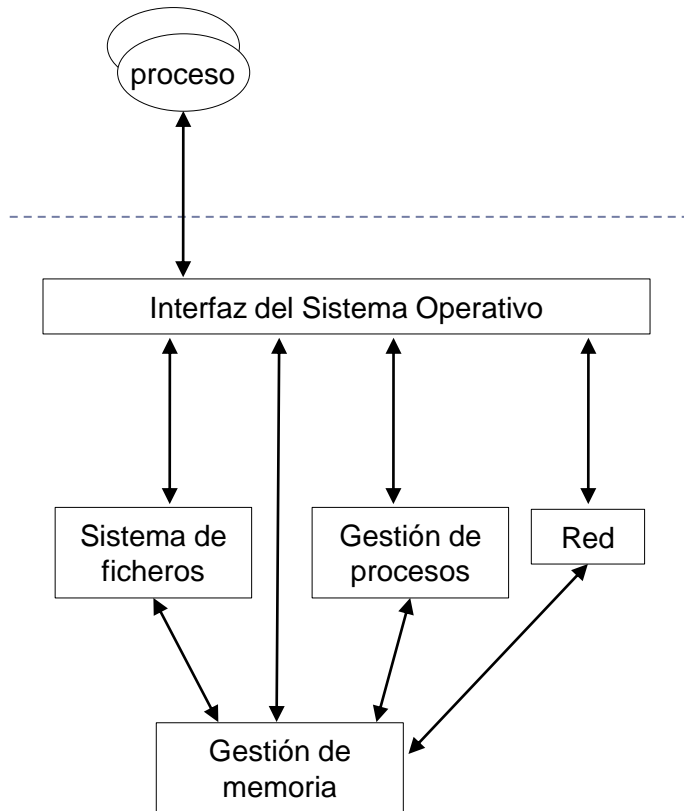
- ▶ S.O. multiplexa recursos entre procesos
  - ▶ Cada proceso cree que tiene una máquina para él solo.
  - ▶ Gestión de procesos: Reparto de procesador.
  - ▶ Gestión de memoria: Reparto de memoria.

## Objetivos:

Objetivos generales

- ▶ Ofrecer a cada proceso un espacio lógico propio.
- ▶ Proporcionar protección entre procesos.
- ▶ Permitir que los procesos compartan memoria.
- ▶ Dar soporte a las regiones del proceso.
- ▶ Maximizar el grado de multiprogramación.
- ▶ Proporcionar a los procesos mapas de memoria muy grandes.

# Objetivos generales de la memoria

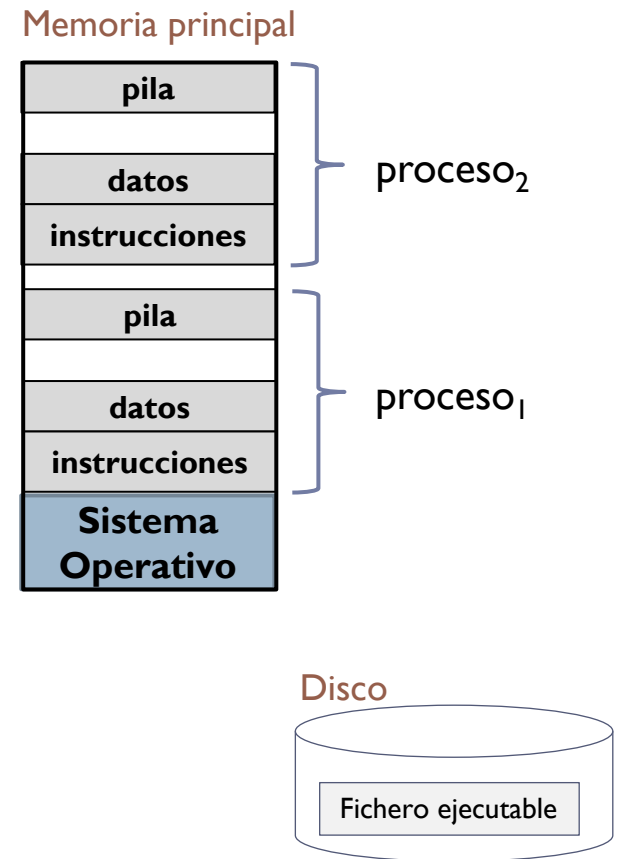


1. **Localización de referencias a memoria**
  - ▶ ha de traducir las referencias a memoria a direcciones físicas
2. **Protección de espacios de memoria**
  - ▶ prohibir referencias entre procesos distintos
3. **Compartición de espacios de memoria**
  - ▶ permitir que varios procesos accedan a un espacio de memoria común
4. **Organización lógica (de programas)**
  - ▶ los programas se dividen en módulos independientes
5. **Organización física (de la memoria)**
  - ▶ rellenar la memoria con múltiples programas y módulos

# Objetivos generales de la memoria

## 1.- Localización de referencias a memoria

- ▶ El programador no tiene que saber dónde se colocará el programa en memoria cuando se ejecute:
  - ▶ Si se ejecuta varias veces, cada una de ellas irá a una parte de memoria diferente.
  - ▶ Mientras es ejecutado también puede suspenderse en m. secundaria y volver a memoria en una posición diferente.
- ▶ Ejecutable contiene direcciones lógicas (ej.: vector en la posición 1004) y al cargar hay que asignar las correspondientes direcciones físicas.
- ▶ Reubicación: traducción de las referencias lógicas (relativas) de memoria a direcciones físicas (absolutas)



# Objetivos generales de la memoria

## 1.- Localización de referencias a memoria

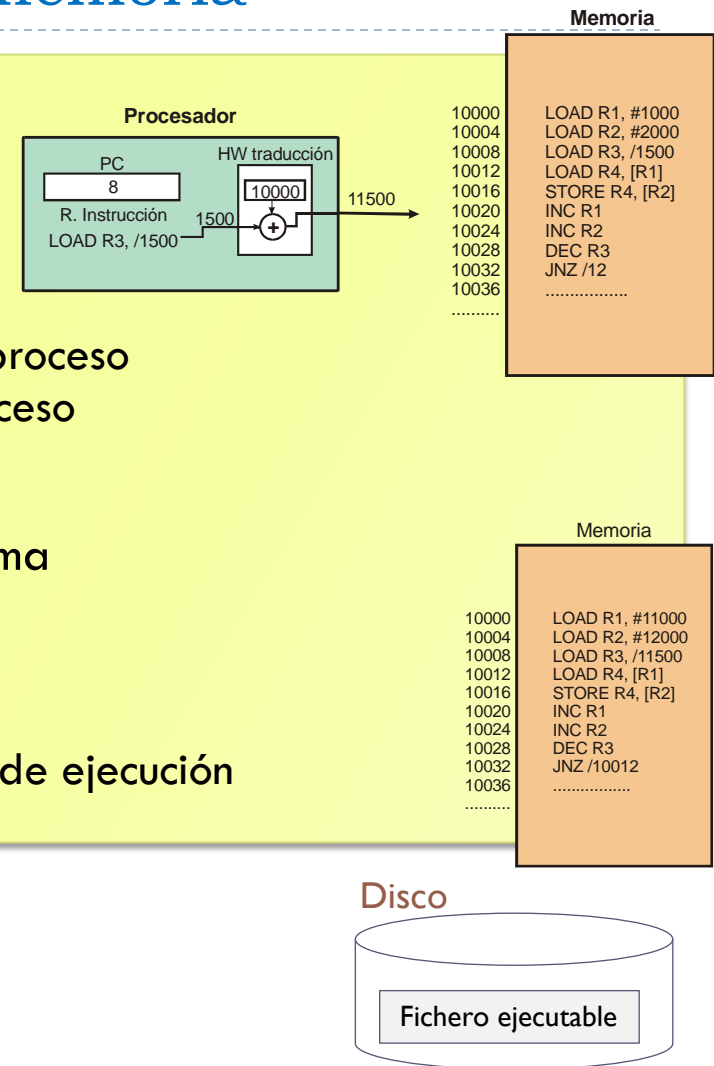
### ■ Reubicación hardware

- Hardware (MMU) encargado de traducción
- Programa se carga en memoria sin modificar
- S.O. se encarga de:
  - Almacenar la función de traducción de cada proceso
  - Especificar al hw. la función a aplicar por proceso

### ■ Reubicación software

- Traducción de direcciones al cargar el programa
- Programa en memoria distinto del ejecutable
- Desventajas:
  - No asegura protección
  - No permite mover imagen proceso en tiempo de ejecución

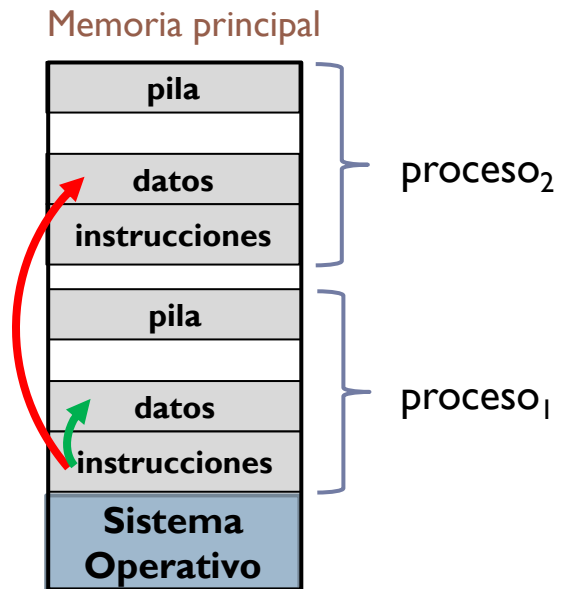
- ▶ Reubicación: traducción de las referencias lógicas (relativas) de memoria a direcciones físicas (absolutas)



# Objetivos generales de la memoria

## 2.- Protección de espacios de memoria

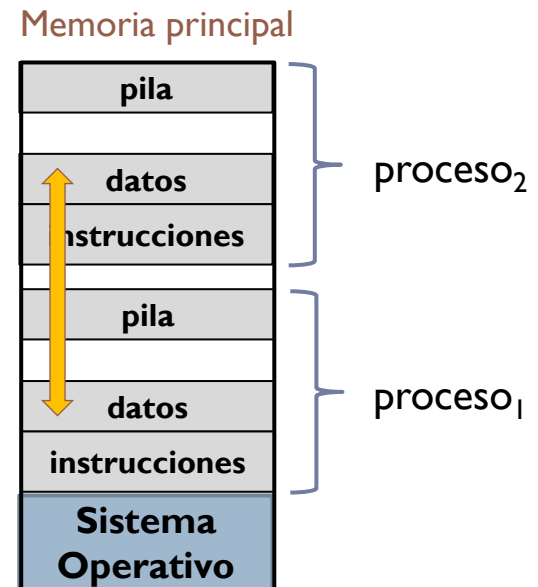
- ▶ Los procesos no han de usar posiciones de memoria de otros procesos por seguridad.
  - ▶ Excepción: depurador.
  - ▶ Reubicación previene, pero no evita.
- ▶ Las **posiciones de memoria** tendrían que ser **comprobadas en tiempo de ejecución**
  - ▶ No es posible comprobar los accesos a memoria física en tiempo de compilación
- ▶ Las posiciones de memoria han de comprobarse **por hardware**
  - ▶ El sistema operativo trata excepciones pero no anticipa las referencias de memoria que un proceso va a realizar.



# Objetivos generales de la memoria

## 3.- Compartición de espacios de memoria

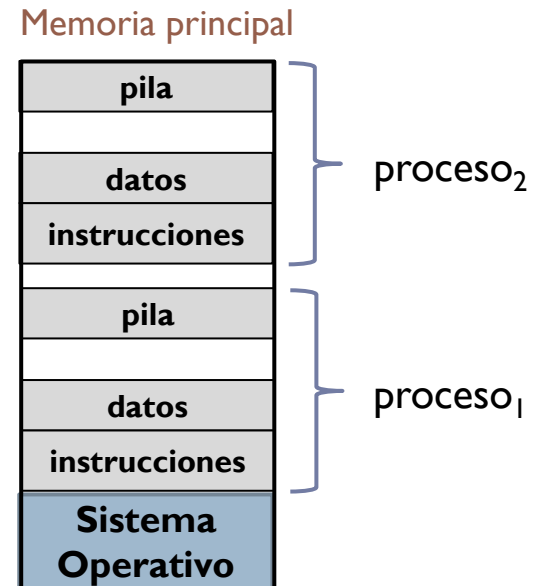
- ▶ Opuesto al punto anterior (aparentemente) debe ser **posible** que **varios procesos** puedan **acceder** a la **misma porción de memoria**:
  - ▶ Procesos ejecutando el mismo código podrían compartir la misma copia de código en memoria
  - ▶ Procesos que cooperan en la misma tarea pueden necesitar acceder a las mismas estructuras de datos
- ▶ Debe ser solicitado y concedido explícitamente
  - ▶ Depurador, etc.



# Objetivos generales de la memoria

## 4.- Organización lógica (de los programas)

- ▶ Imagen de un proceso no es homogénea
  - ▶ Ej.: código, variables locales, etc.
  - ▶ Cada tipo de información tiene distintas necesidades
    - ▶ Lectura, escritura, ejecución, etc.
    - ▶ Creación estática o dinámica
- ▶ La información de un proceso (su imagen) se divide en diferentes regiones
  - ▶ Cada región se adapta a un tipo de datos concreto (código, variable dinámica, etc.)
    - ▶ Hay que gestionar las zonas sin asignar (huecos)
- ▶ Gestionar la memoria de un proceso es gestionar cada una de sus regiones



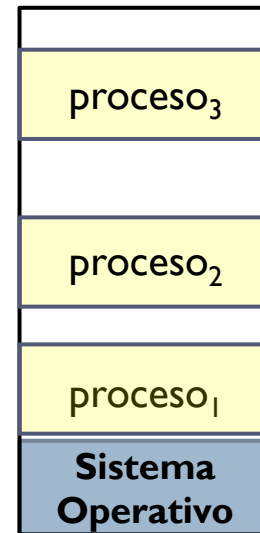


# Objetivos generales de la memoria

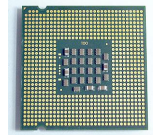
## 5.- Organización física (de la memoria)

- ▶ Poder ejecutar un proceso cuando su imagen de memoria es más grande que la memoria principal:
  - ▶ Se guardan en disco las partes del proceso que no se usen en el momento
- ▶ Poder ejecutar un conjunto de procesos cuya ocupación de memoria es mayor que la memoria principal
- ▶ Evitar pérdida de memoria por fragmentación:
  - ▶ Hay memoria física libre pero está fragmentada en espacios no contiguos que el sistema de gestión no puede aprovechar

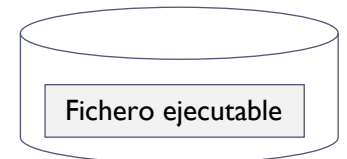
Memoria principal  
(512 MB)



32 bits  
(4 GB)



Disco  
(1 TB)

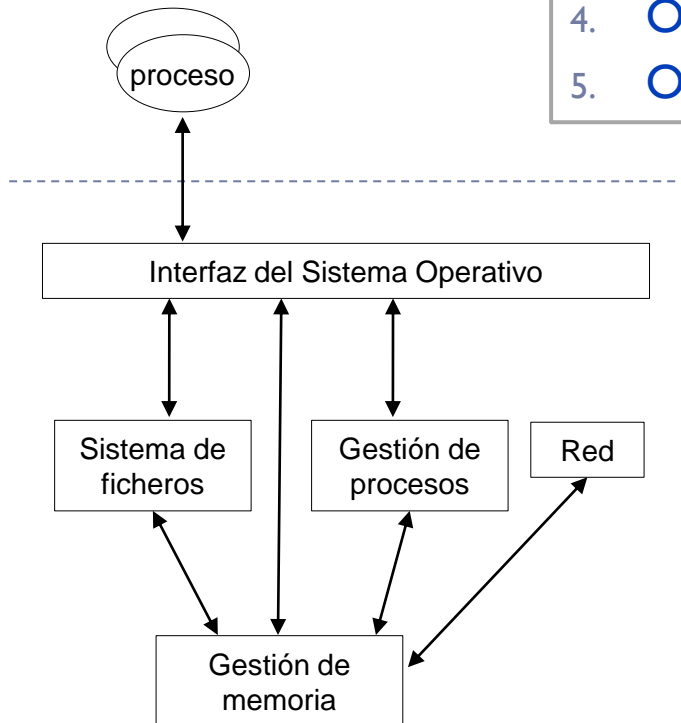


# Ámbito, arquitectura y objetivos

## resumen

---

1. Localización de referencias a memoria
2. Protección de espacios de memoria
3. Compartición de espacios de memoria
4. Organización lógica (de programas)
5. Organización física (de la memoria)



# Contenidos

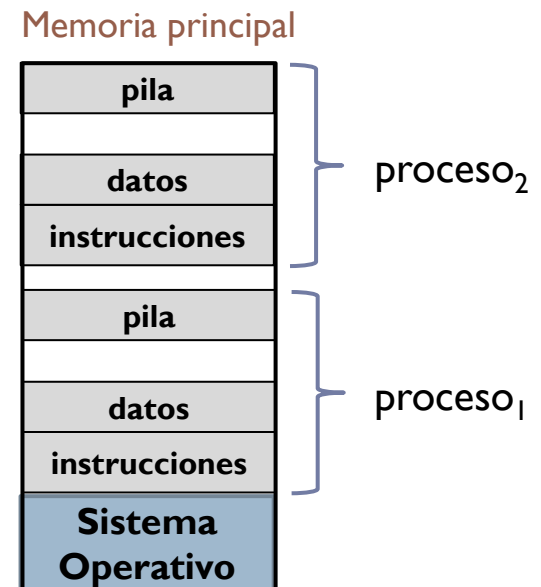
---

1. **Introducción:**
  1. Modelo abstracto y definiciones básicas.
  2. Mapa de memoria de un proceso: regiones de memoria.
2. **Funciones del gestor de memoria.**
  1. **Particionamiento de memoria.**
  2. **Algoritmos de gestión de memoria.**
3. Memoria virtual.
4. Servicios del gestor de memoria.

# Gestión de memoria N1

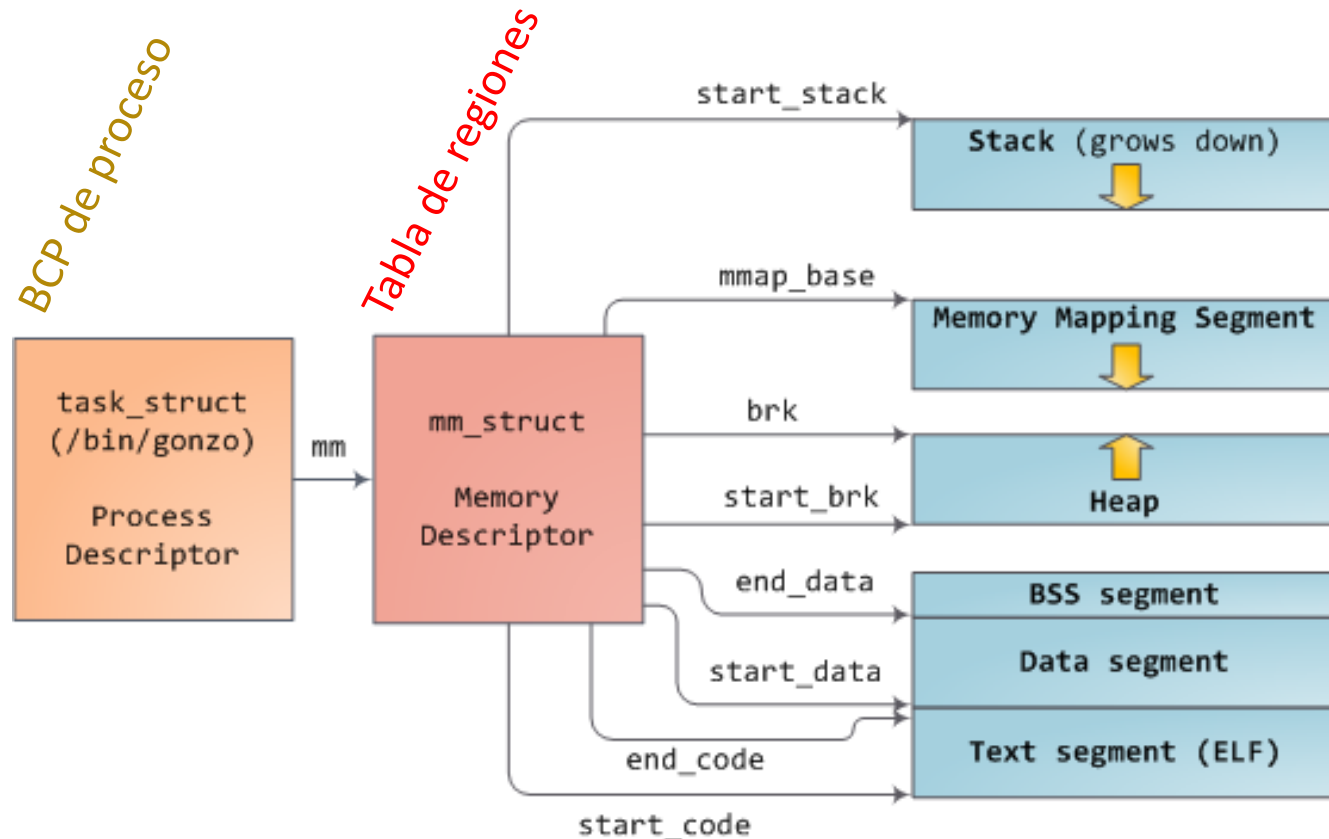
## Imágenes de proceso: regiones de memoria

- ▶ **Imagen de un proceso no es homogénea**
  - ▶ Dividida en diferentes regiones
    - ▶ Cada región tiene mismas características.
    - ▶ Cada región se adapta a un tipo de datos concreto (código, variable dinámica, etc.)
- ▶ **Mapa de proceso variable en el tiempo**
  - ▶ Se crean y destruyen regiones.
  - ▶ Las regiones pueden cambiar de tamaño.
  - ▶ Existen zonas sin asignar (huecos)
- ▶ **Gestor de memoria administra cada una de las regiones de un proceso**
  - ▶ S.O. usa “tabla” de regiones por proceso
  - ▶ C.R.U.D. de regiones
  - ▶ Controla el adecuado uso, detectando:
    - ▶ accesos no permitidos, accesos a huecos, reserva de huecos, etc.



# Gestor de memoria: MP+Interfaz+Metadatos

## tabla de regiones en Linux



# Gestor de memoria: Gestión de regiones

## particionado de memoria

---

### ▶ Típicas estructuras de datos:

- ▶ Lista única
- ▶ Múltiples listas
- ▶ Sistema *buddy* binario (huecos:  $2^n$ )
- ▶ Mapa de bits

### ▶ Tipos generales de técnicas

#### ▶ Memoria continua

- ▶ Particionamiento **estático** o particiones de tamaño fijo
- ▶ Particionamiento **dinámico** o particiones de tamaño variable

#### ▶ Memoria no continua

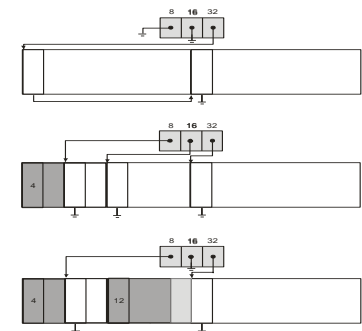
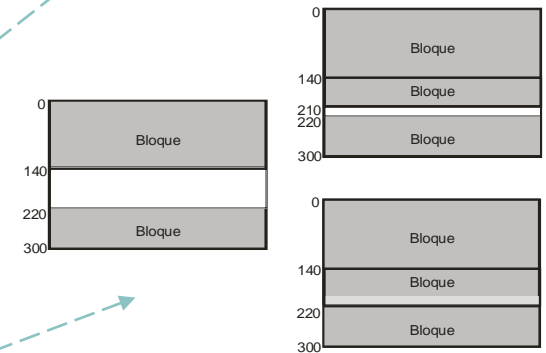
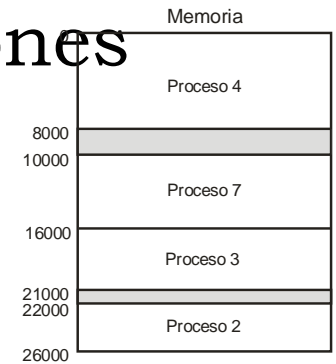
- ▶ **Paginación**
- ▶ **Segmentación**
- ▶ Segmentación paginada

### Algoritmos de reserva



# Gestor de memoria: Gestión de regiones particionado de memoria


- ▶ Se precisa llevar la pista de bloques y huecos
  - ▶ Internamente a la unidad gestionada
  - ▶ Externamente (tabla de regiones)
- ▶ Distintas soluciones:
  - ▶ Lista única
  - ▶ Múltiples listas con huecos de tamaño variable
    - ▶ *Problema*: se genera huecos de distintos tamaños no continuos no aprovechables (fragmentación externa)
    - ▶ *Solución*: compactar memoria (lento en m. grande)
  - ▶ Múltiples listas con particiones estáticas
    - ▶ *Problema*: huecos no aprovechables dentro del bloque asignado (fragmentación interna) + límite tamaño
    - ▶ *Solución*: huecos fijos de varios tamaños
  - ▶ Sistema *buddy* binario (huecos:  $2^n$ )
  - ▶ Mapa de bits



# Algoritmo de asignación de espacio

---

+ **eficiente**: menos fragmentación

- 
- El hueco que mejor ajuste (**best fit**)
    - ▣ Selección: comprobar todos u ordenados por tamaño.
  - El hueco que peor ajuste (**worst fit**)
    - ▣ Selección: comprobar todos u ordenados por tamaño.
  - El primer hueco que ajuste (**first fit**)
    - ▣ Suele ser la mejor política en muchas situaciones.
  - El próximo hueco que ajuste (**next fit**)
    - ▣ Variación del primero que ajuste.
    - ▣ Busca a partir del último asignado.

+ **rápido**: menos tiempo de búsqueda



# Contenidos

---

## 1. Introducción:

1. Modelo abstracto y definiciones básicas.
2. Mapa de memoria de un proceso: regiones de memoria.

## 2. Funciones del gestor de memoria.

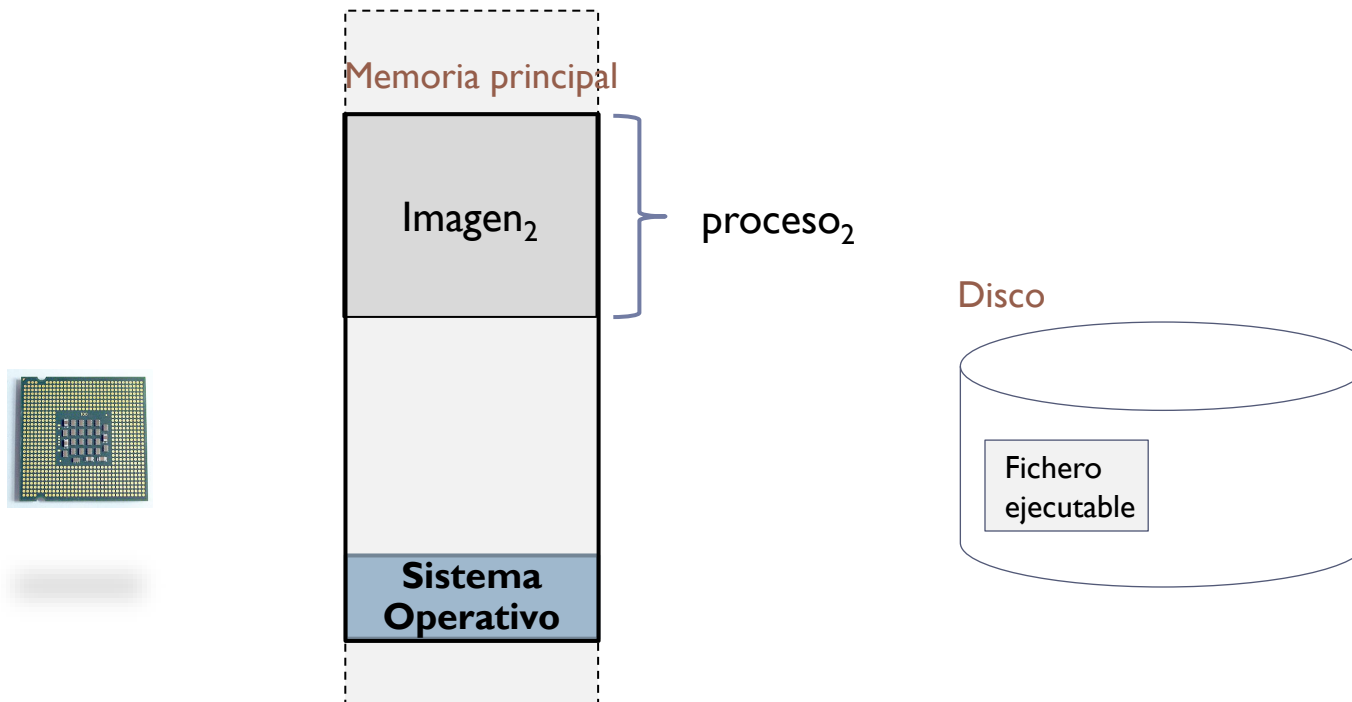
1. Particionamiento de memoria.
2. Algoritmos de gestión de memoria.

## 3. **Memoria virtual.**


## 4. Servicios del gestor de memoria.

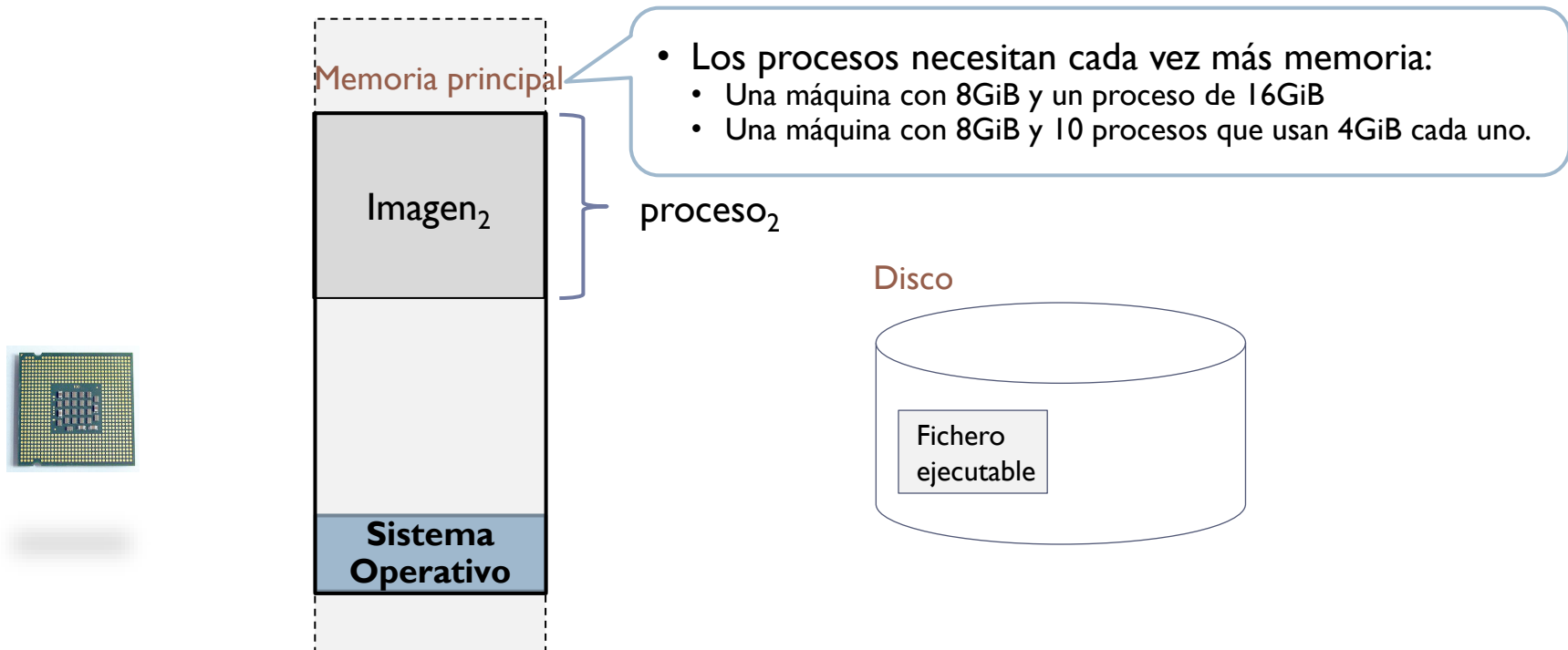
## El sistema operativo...

- ▶ El sistema operativo se encargará en la gestión de la memoria de:
  - ▶ Cargar/Descargar parte de la imagen (solo tener lo necesario en memoria).
  - ▶ Trata de mejorar el uso de recursos:
    - ▶ Poder ejecutar **Out-of-core**
    - ▶ Mejor ocupación de la CPU (tener lo necesario de ↑ procesos).



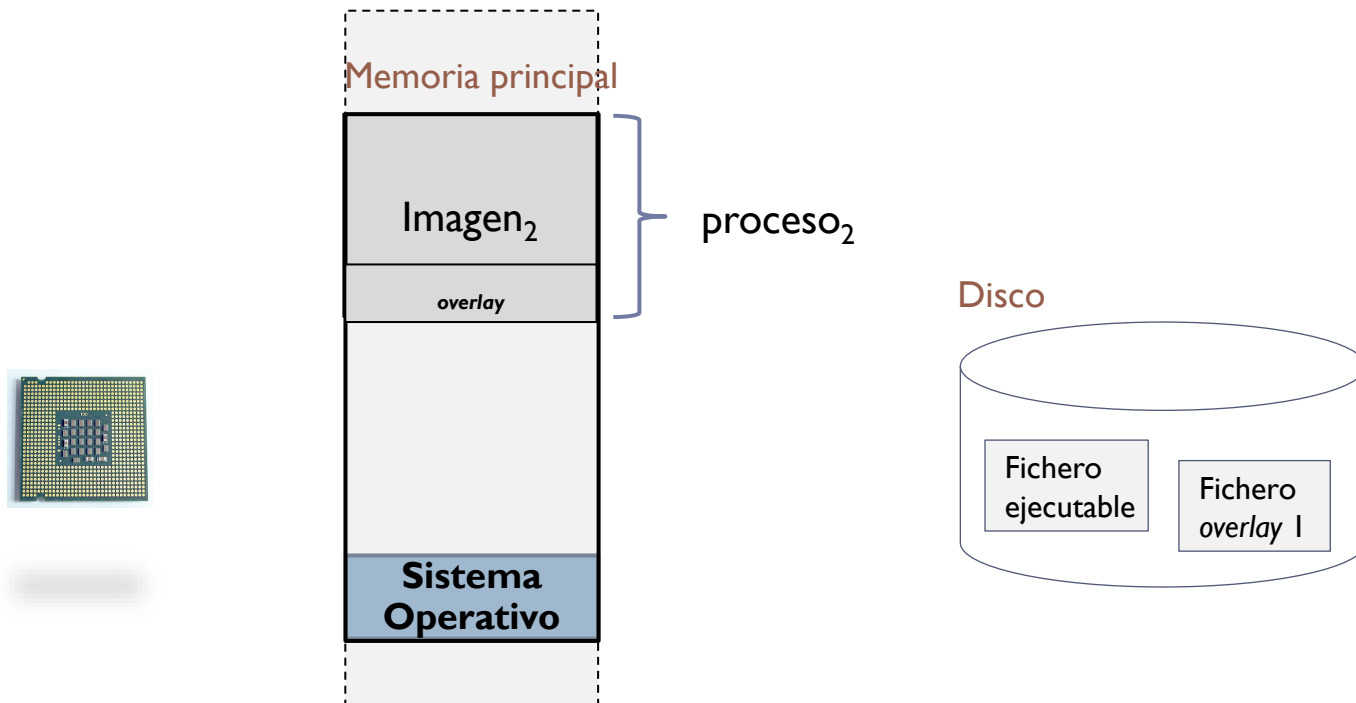
## El sistema operativo...

- ▶ El sistema operativo se encargará en la gestión de la memoria de:
  - ▶ Cargar/Descargar parte de la imagen (solo tener lo necesario en memoria).
  - ▶ Trata de mejorar el uso de recursos:
    - ▶ Poder ejecutar **Out-of-core**
    - ▶ Mejor ocupación de la CPU (tener lo necesario de  procesos).



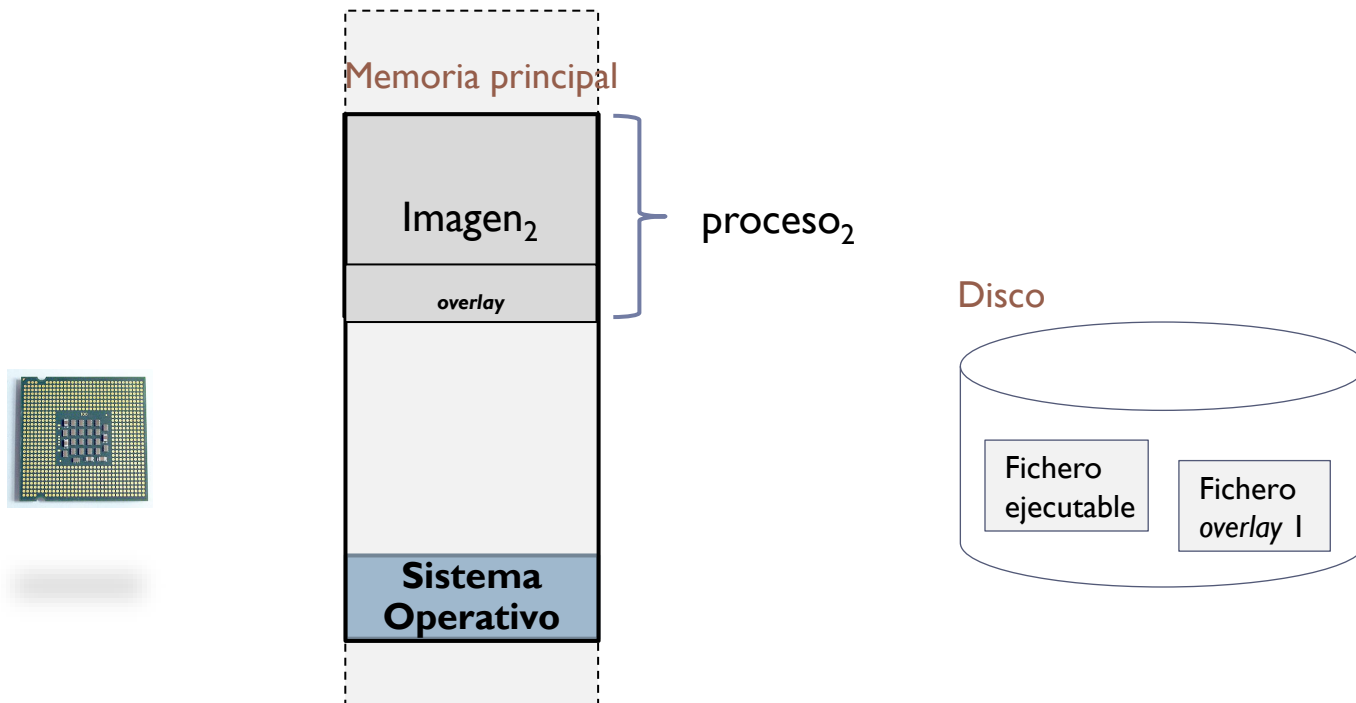
## El sistema operativo...

- ▶ Inicialmente se usó el mecanismo de *overlays* de:
  - ▶ *Cargar/Descargar* parte de la imagen (solo tener lo necesario en memoria).
  - ▶ Cada programador se ocupa explícitamente de los *overlays* de su programa.

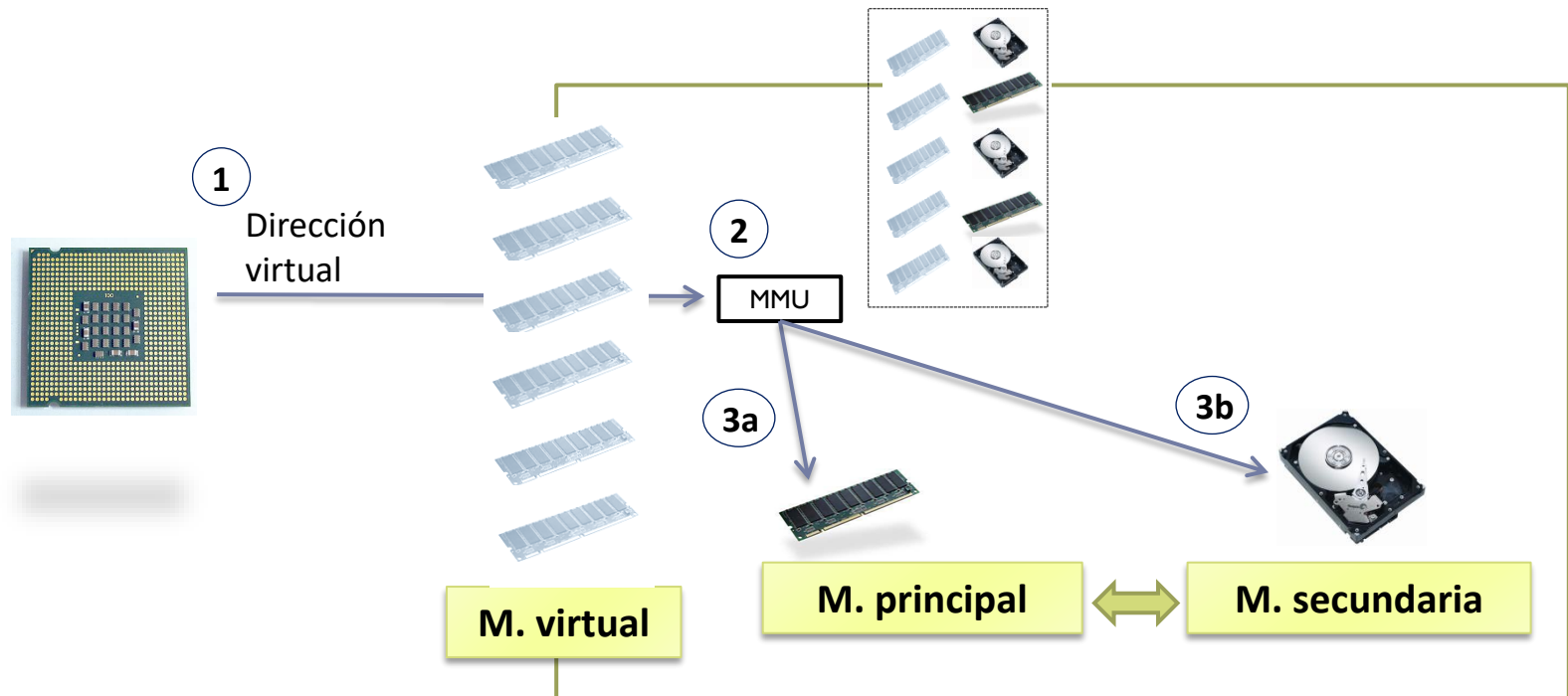


## El sistema operativo...

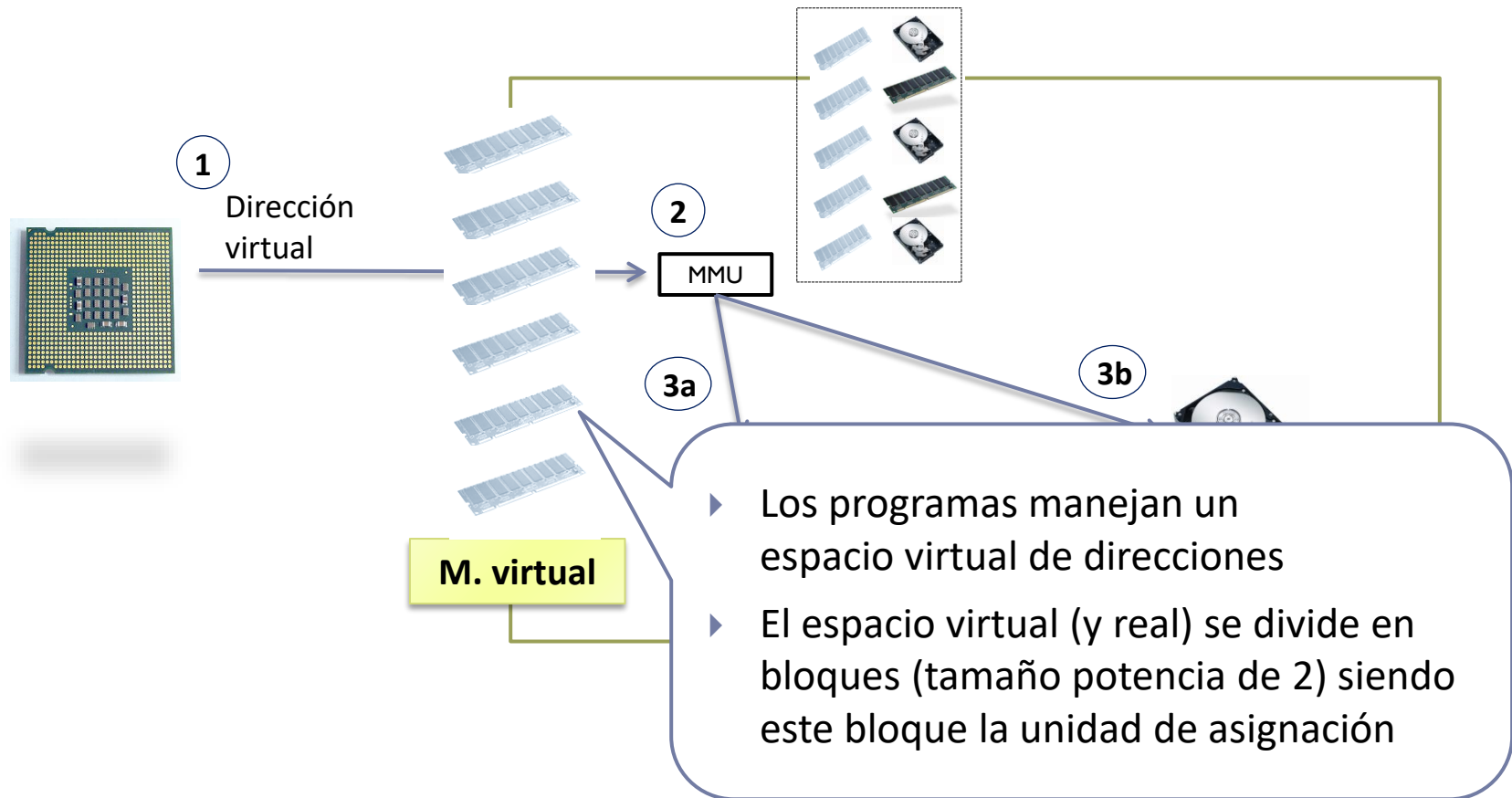
- ▶ La CPU + S.O. ofrecen un mecanismo alternativo a *overlays*:
  - ▶ **Memoria virtual** (automatiza la parte de solo tener lo necesario en memoria).
  - ▶ Transparente a los programadores.



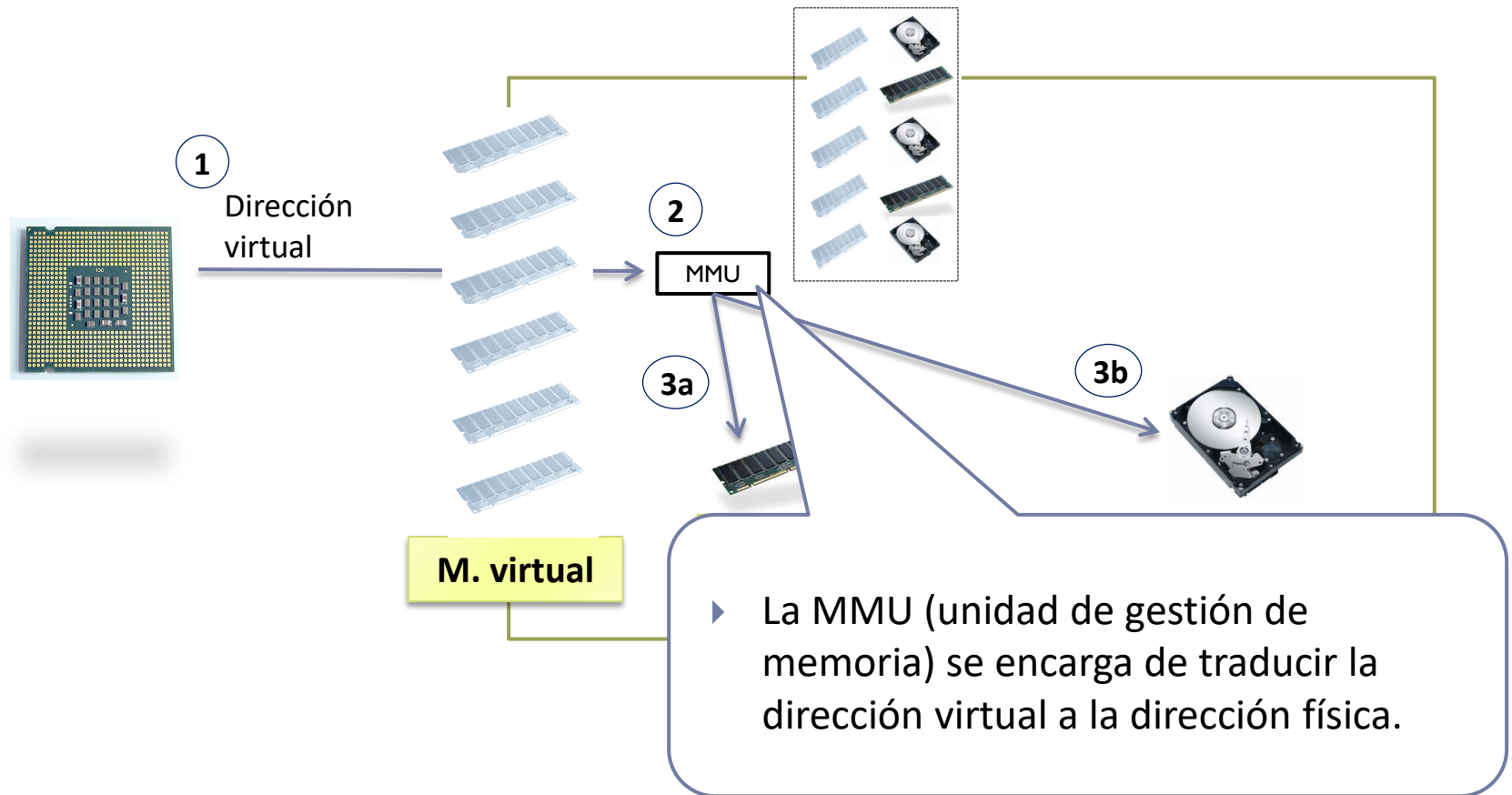
# Sistemas con memoria virtual



# Sistemas con memoria virtual

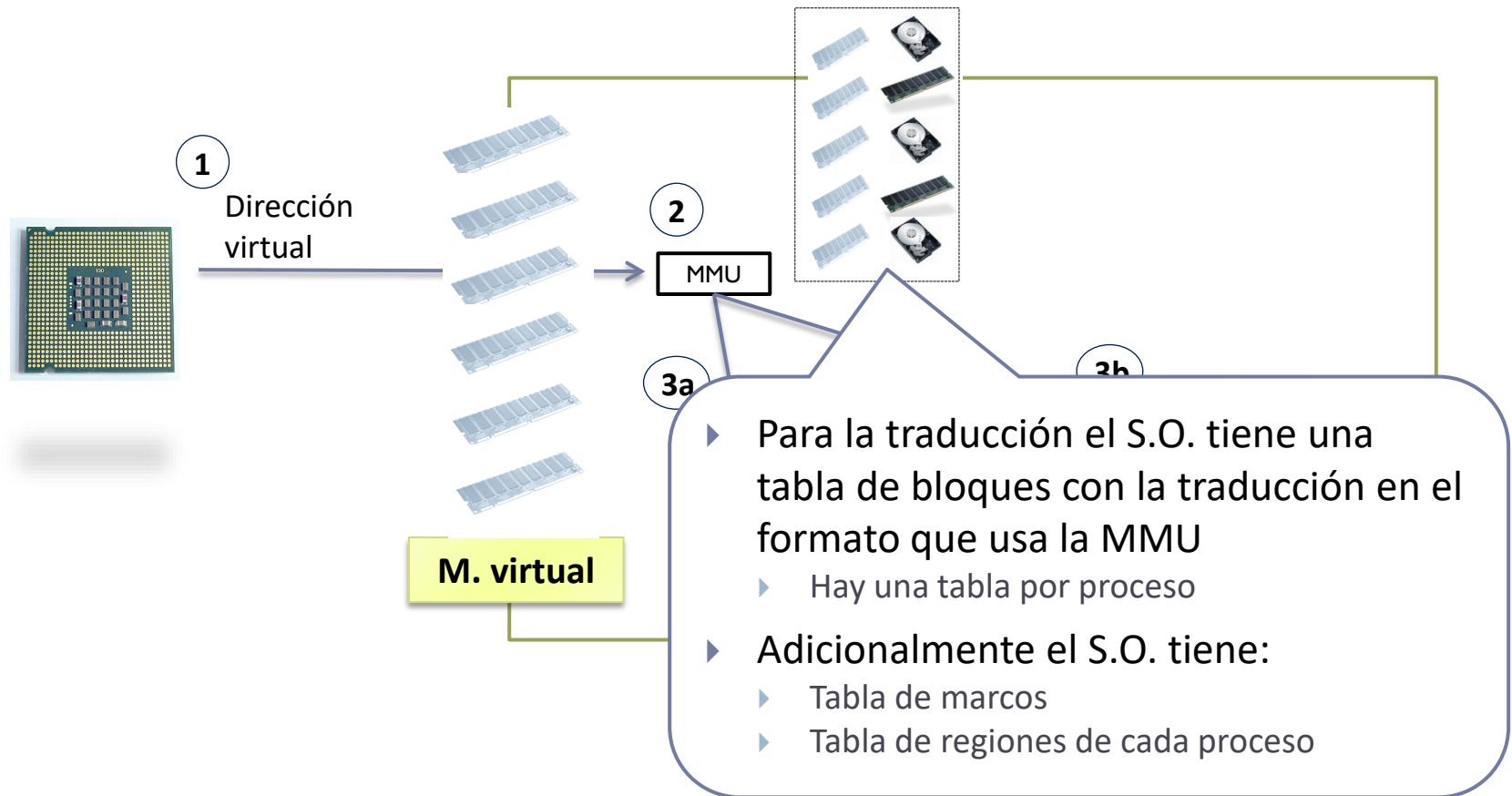


# Sistemas con memoria virtual

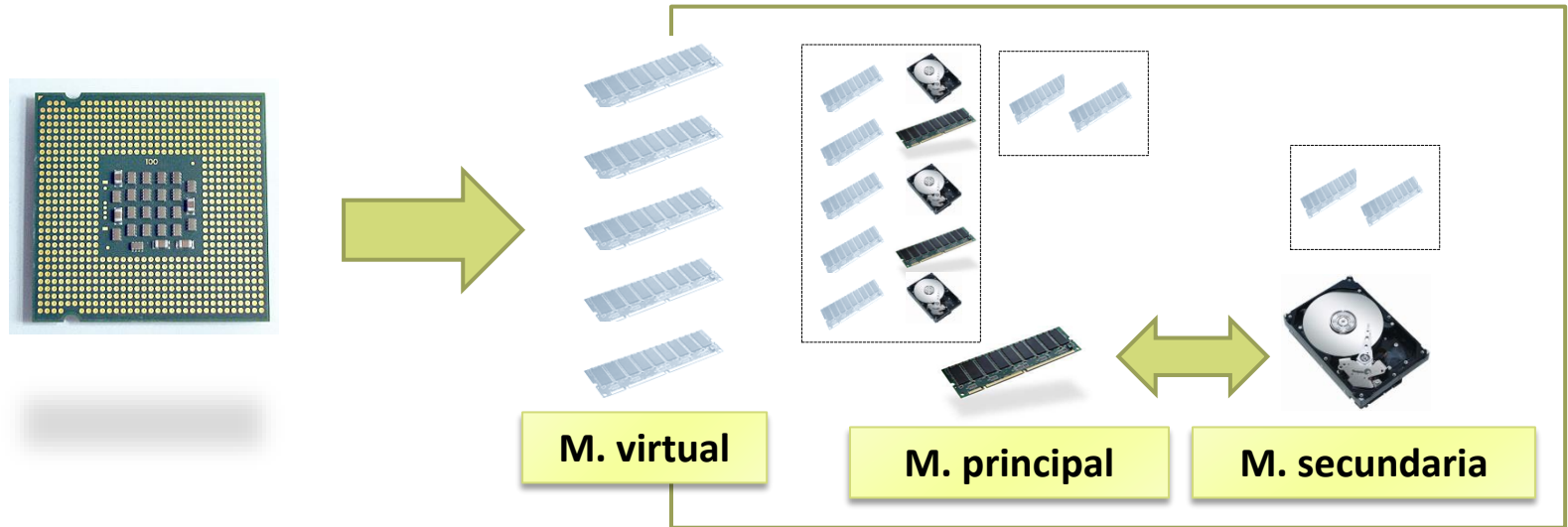




# Sistemas con memoria virtual

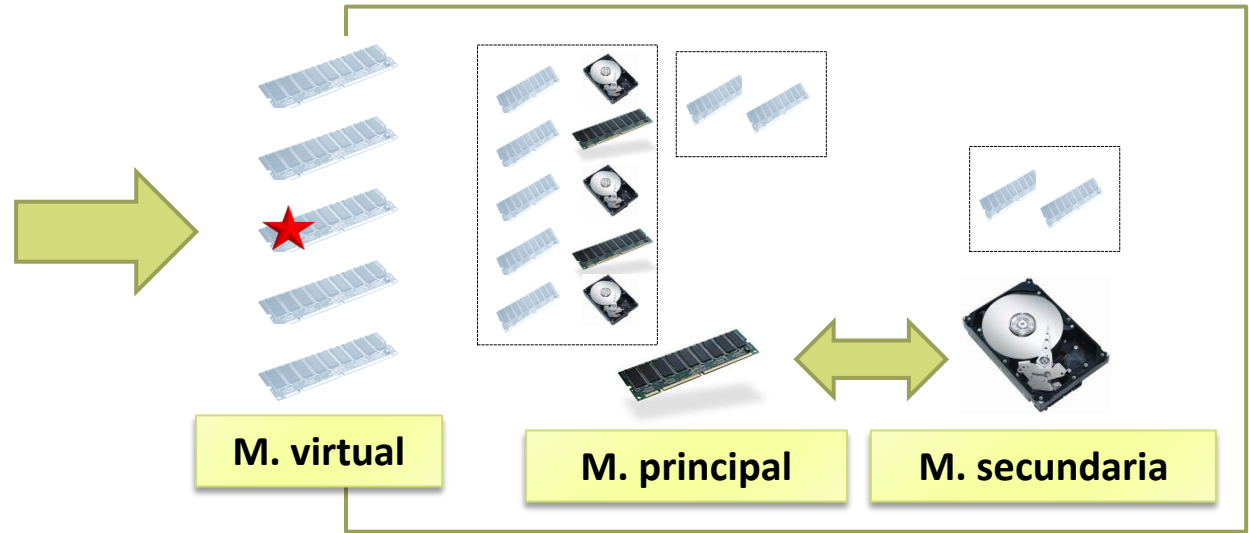
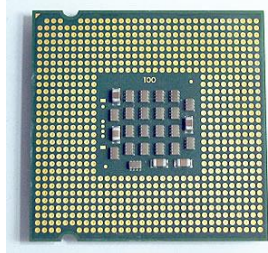


# Introducción a memoria virtual

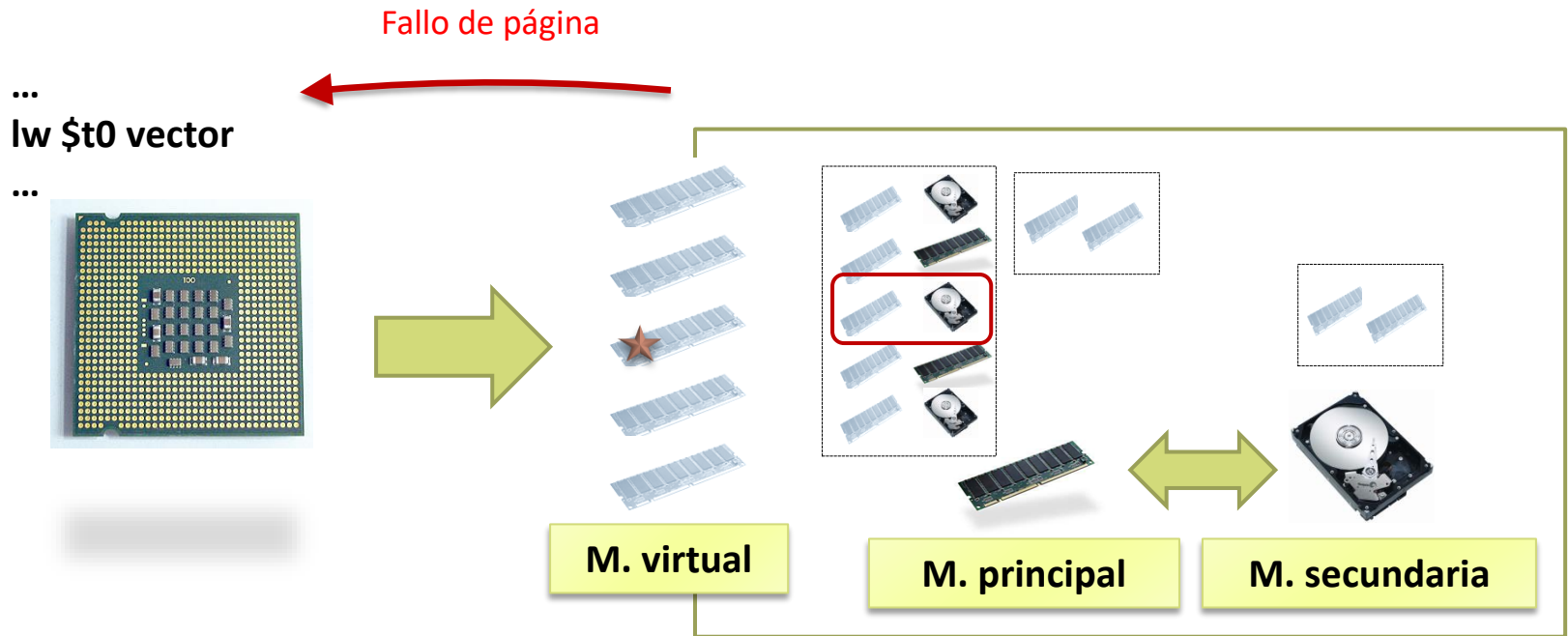


# Introducción a memoria virtual

...  
lw \$t0 vector  
...

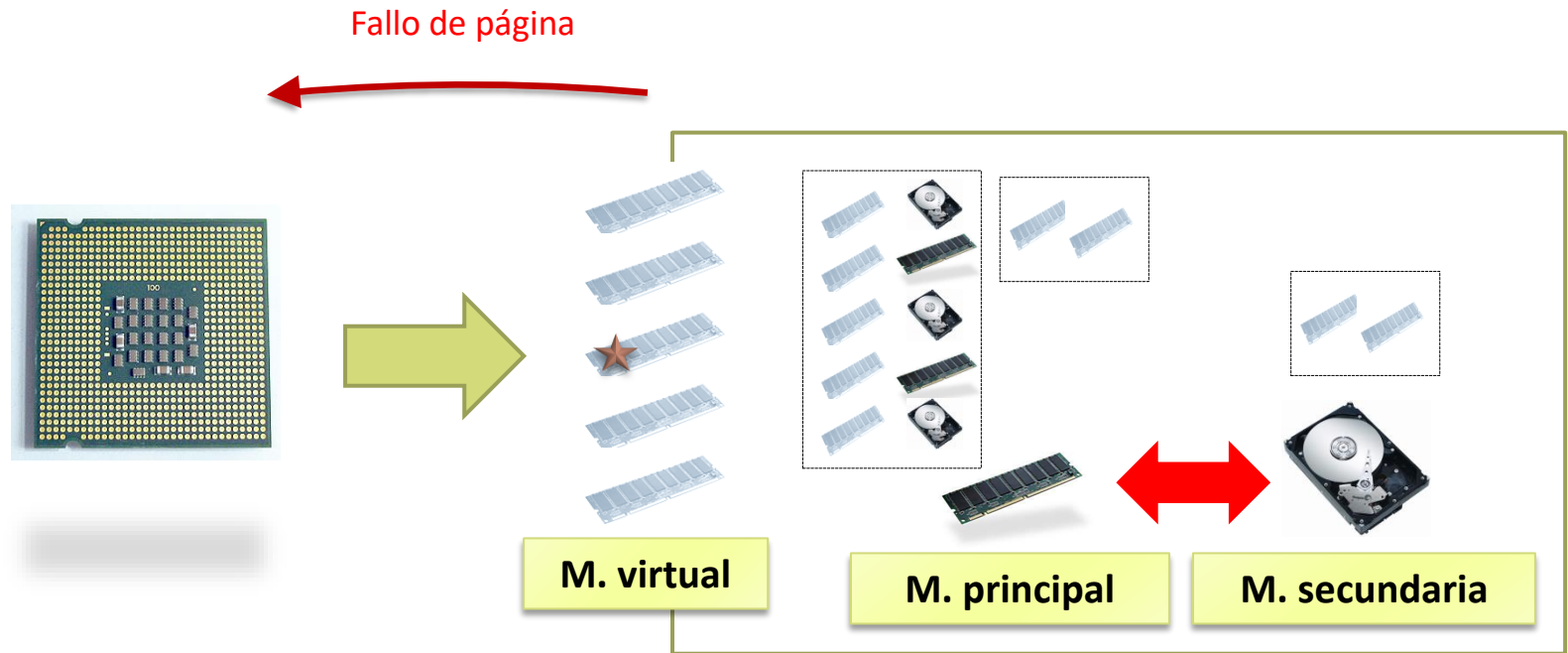


# Introducción a memoria virtual



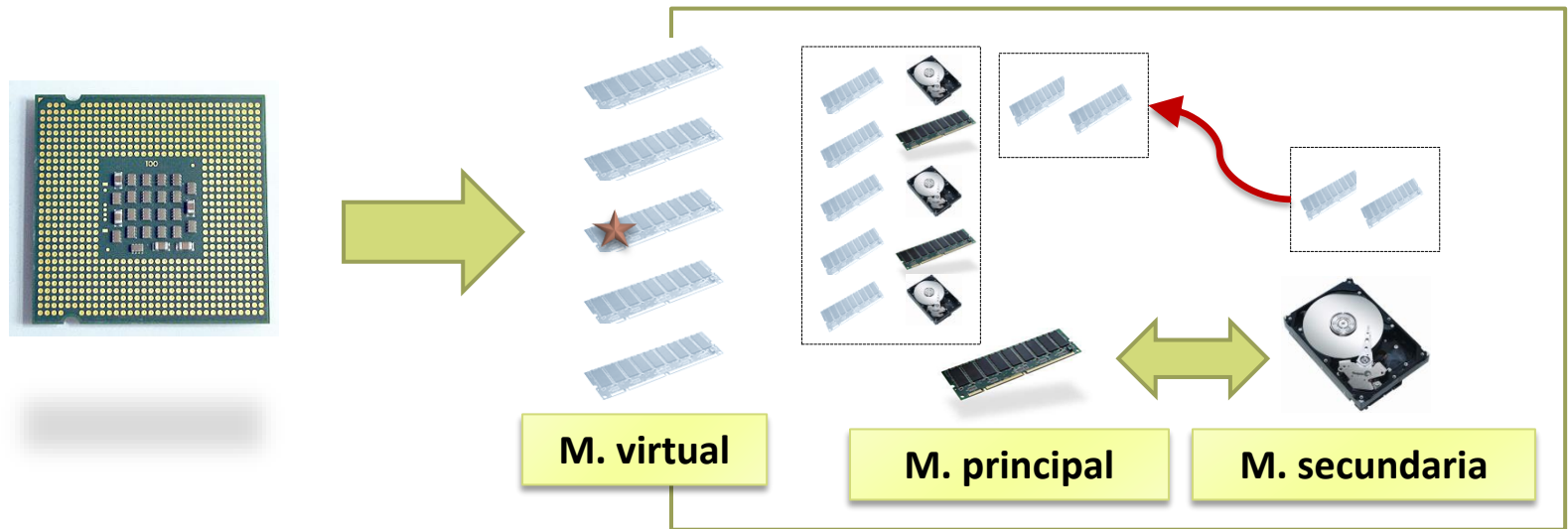
- ▶ El fallo de página es una excepción que provoca que el procesador ejecute la rutina de tratamiento asociada (kernel del s.o.).
- ▶ Fallo de pág. no es siempre error: acceso a pág. no residente.

# Introducción a memoria virtual



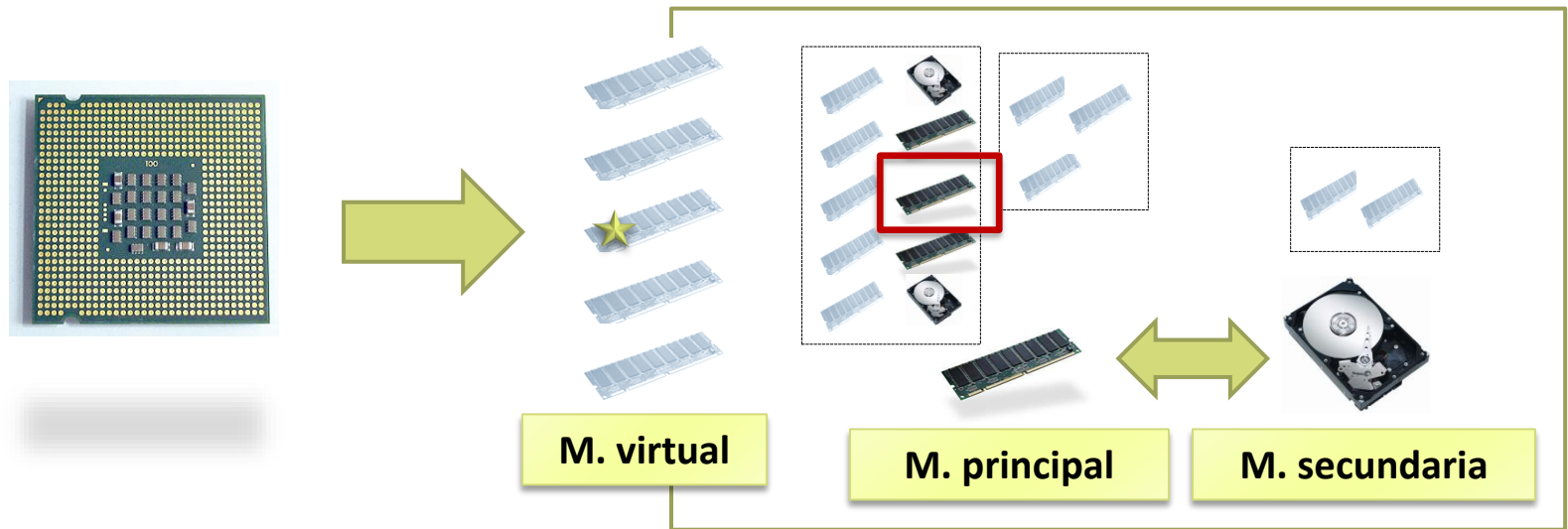
- ▶ La rutina de tratamiento de fallo de página pide los bloques de disco asociados y bloquea el proceso.

# Introducción a memoria virtual



- ▶ En la interrupción hardware de disco se transfiere el 'bloque' solicitado a memoria principal, ...

# Introducción a memoria virtual

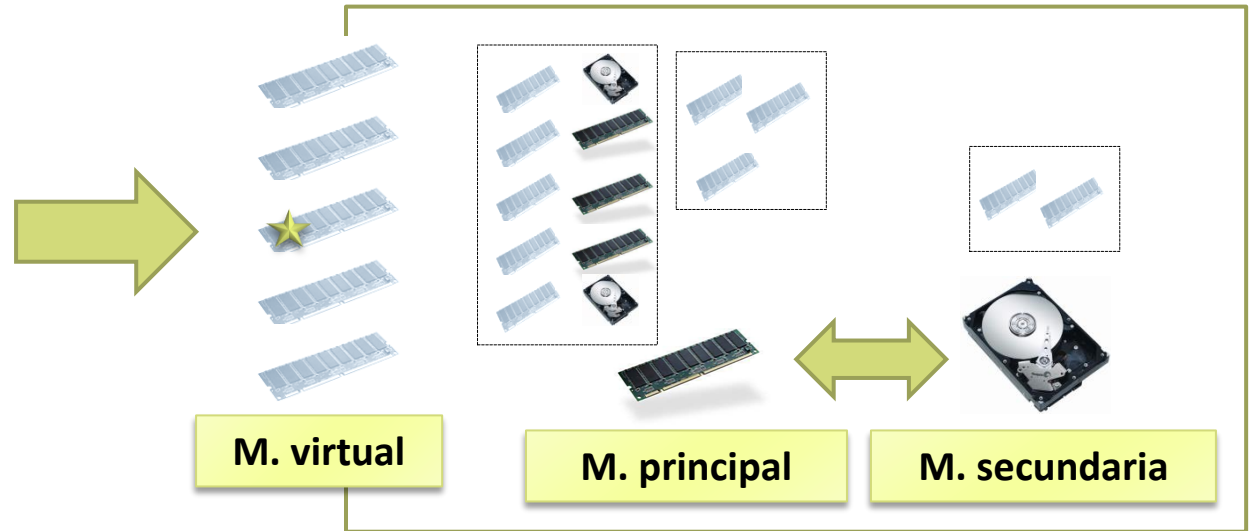
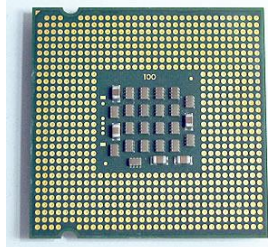


- ▶ ... se actualiza la tabla de 'bloques' y se pone el proceso listo para ejecutar.

# Introducción a memoria virtual

Acierto

...  
lw \$t0 vector  
...



- ▶ Se reanuda la ejecución de la instrucción que provocó el fallo.
- ▶ Prepaginación: trae páginas por anticipado (no por demanda)

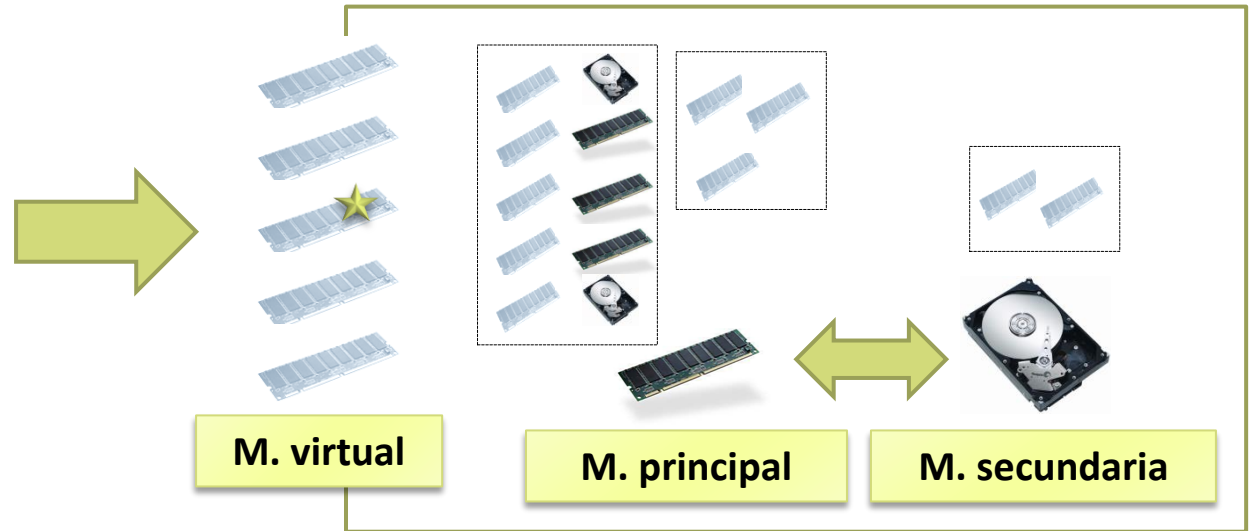
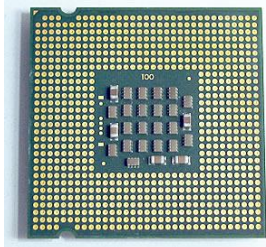


# Introducción a memoria virtual

Acierto

...  
lw \$t0 vector+4

...



- ▶ La siguiente instrucción del mismo bloque no provoca fallo.

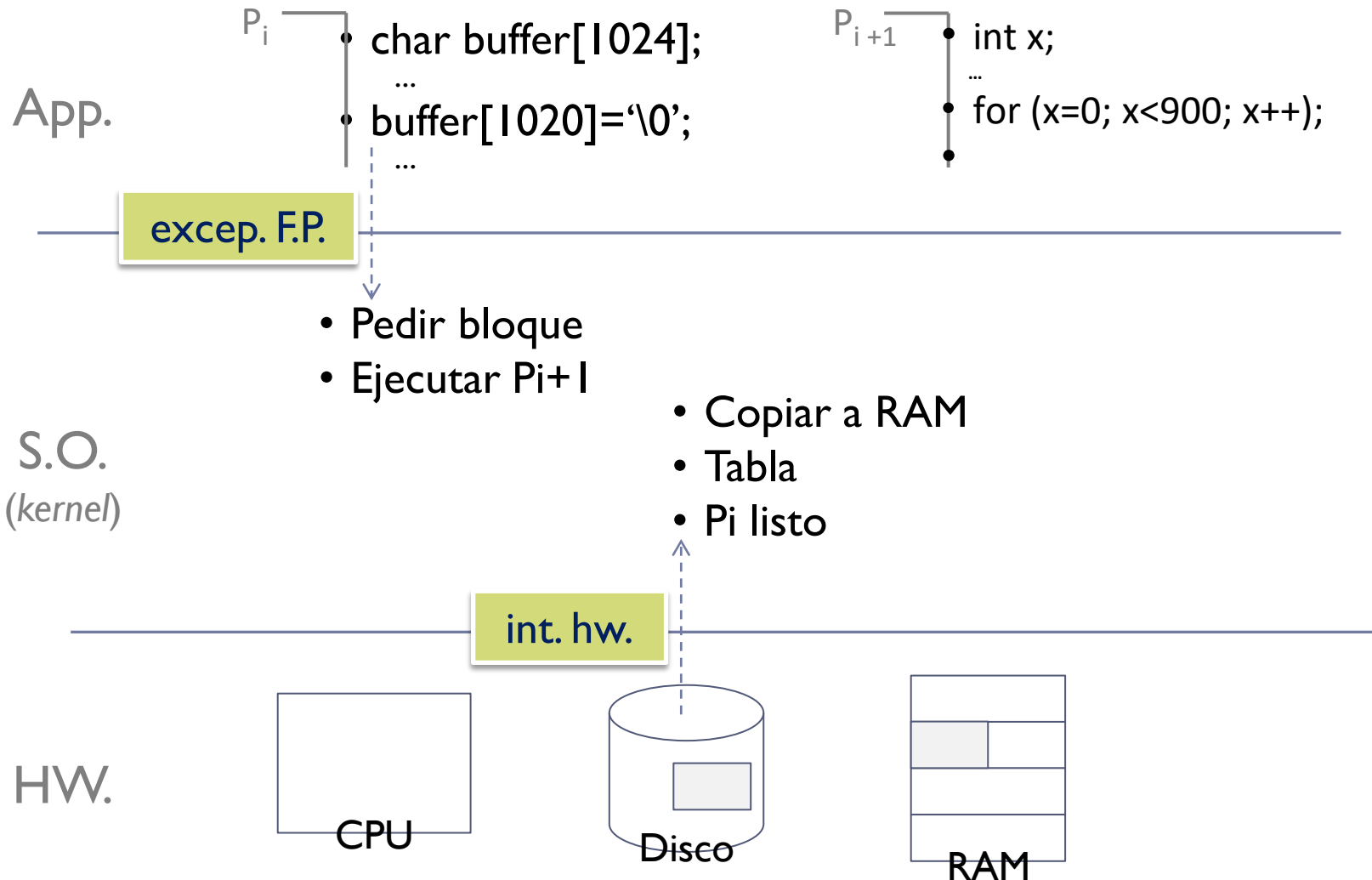
# Fundamento de la memoria virtual

## resumen

---

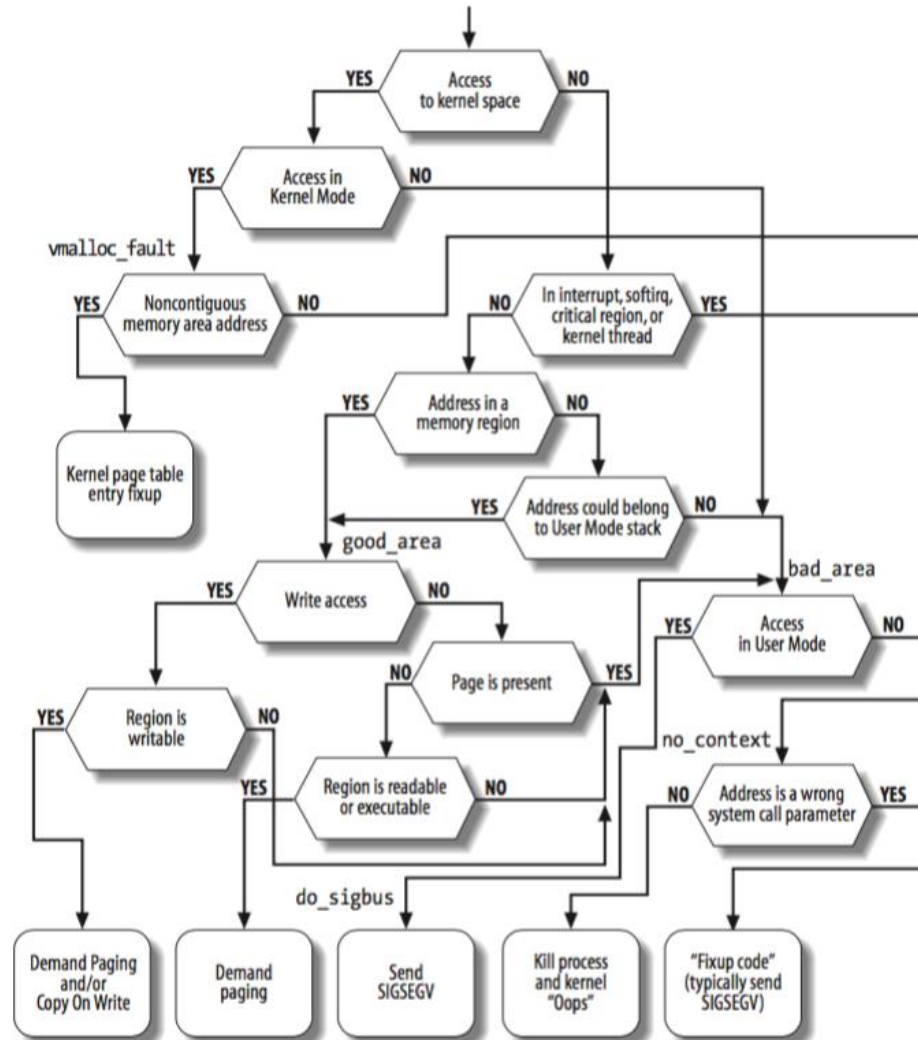
- ▶ **M. virtual:**
  - ▶ SO gestiona niveles de m. principal y m. secundaria.
  - ▶ Dividido en bloques (ej.: páginas) y tabla de páginas para gestión.
  - ▶ Página no residente se marca con bit de validez a cero (no válida).
  - ▶ Sube (ms->mp) por demanda => En acceso: excepción de fallo de página.
  - ▶ Baja (mp->ms) por expulsión => Algoritmo de reemplazo.
- ▶ **Aplicable por proximidad de referencias:**
  - ▶ Procesos sólo usan parte de su mapa en intervalo de tiempo.
  - ▶ Parte usada (*conjunto de trabajo*) en m. principal (*conjunto residente*)
- ▶ **Beneficios:**
  - ▶ Aumenta el grado de multiprogramación.
  - ▶ Permite ejecución de programas que no quepan en mem. principal.

# Ejemplo de excepción de fallo de página (sin remplazo)



# Tratamiento general del fallo de página

## caso de estudio: Linux



# Tratamiento general del fallo de página (con remplazo)

---

- ▶ Si **dirección inválida** → aborta proceso o le manda señal
- ▶ Si **no hay ningún marco libre** (consulta T. marcos)
  - ▶ Selección de víctima (**alg. de reemplazo**): página P marco M
    - ▶ Marca P como inválida
    - ▶ Si P modificada (bit *Mod* de P activo)
      - ▶ Inicia escritura P en memoria secundaria
- ▶ Si **hay marco libre** (se ha liberado o lo había previamente):
  - ▶ Inicia lectura de página en marco M
  - ▶ Marca entrada de página válida referenciando a M
  - ▶ Pone M como ocupado en T. marcos (si no lo estaba)

# Algoritmos de reemplazo

---

## ▶ **Política de reemplazo:**

- ▶ Reemplazo **local**: dentro del proceso.
- ▶ Reemplazo **global**: de cualquier proceso del sistema.

## ▶ **Algoritmo de reemplazo:**

- ▶ Qué página se va a reemplazar:
  - ▶ La página tiene que ser la que tenga menor probabilidad de ser referenciada en un futuro lejano (alg. óptimo, difícil de determinar en avance)
  - ▶ Se intenta “predecir” el comportamiento futuro en base al pasado: tiempo residencia, frecuencia de uso, “frescura” de la página (- reciente usada), etc.
- ▶ Algoritmos de reemplazo (validos para local y global):
  - ▶ **Óptimo**, **FIFO**, **Reloj** (o segunda oportunidad), **LRU** (evitar belady con alg. pila)

## ▶ **Política de asignación de marcos a los procesos:**

- ▶ Asignación **fija** (siempre con reemplazo local):
  - ▶ **Conjunto residente** del proceso es **constante**
- ▶ Asignación **dinámica** (reemplazo local<sub>↑</sub> o global<sub>↓</sub>):
  - ▶ **Conjunto residente** del proceso es **variable** (regulado por **tasa de fallos**: minimizar)

# Algoritmos de **no** reemplazo

---

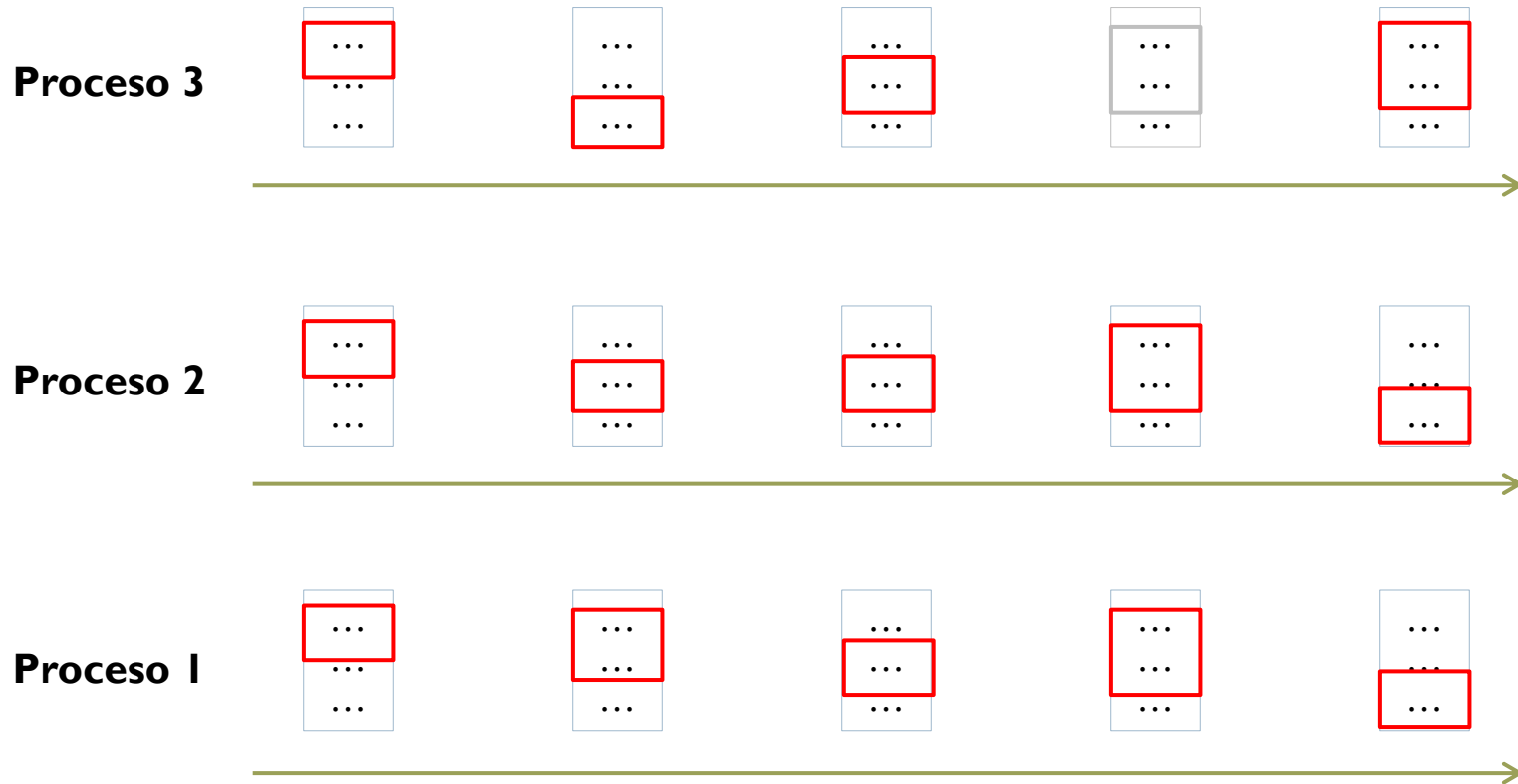
- ▶ Bloqueo o retención de marcos en memoria:
  - ▶ Cuando un marco está bloqueado, la página cargada en ese marco no puede ser reemplazada.
- ▶ Ejemplos de cuándo se bloquea un marco:
  - ▶ La mayoría del núcleo del sistema operativo.
  - ▶ Estructuras de control.
  - ▶ Buffers de E/S (Ej.: los usados con DMA).
- ▶ **Existe un bit de bloqueo en cada marco.**
  - ▶ El SO lo usa (y los procesos con servicio POSIX `mlock()`)



# M.virtual: principales parámetros

grado multiprogramación + conjunto residente + tamaño de página

---





# M.virtual: principales parámetros

grado multiprogramación + conjunto residente + tamaño de página

Proceso 3



Proceso 2



Proceso 1



## Sistemas multiprogramados

ejemplo de cálculo de utilización



- ▶  $p$  = % tiempo un proceso está bloqueado
- ▶  $p^n$  = probabilidad de que  $n$  procesos independientes estén todos bloqueados
- ▶  $1 - p^n$  = probabilidad de que la CPU **no** esté ociosa (no idle)
- ▶  $n = 5$  procesos independientes
- ▶  $p = 0,8$  del tiempo bloqueado (20% en CPU)
- ▶ **utilización =  $5 * 20\% \rightarrow 100\%$**
- ▶ **utilización =  $1 - 0,8^5 \rightarrow 67\%$**   
 $1 - 0,8^{10} \rightarrow 89\%$

▶ 20

[http://www.cs.rutgers.edu/~pdx416/notes/content09-memory\\_management-slides-6.pdf](http://www.cs.rutgers.edu/~pdx416/notes/content09-memory_management-slides-6.pdf)

ARCOS @ UC3M  
Alejandro Calderón Mateos



# M.virtual: principales parámetros

grado multiprogramación + conjunto residente + tamaño de página

- **Hiperpaginación/Thrashing:** tasa excesiva de fallos de página.
  - Con asignación fija: Hiperpag. en proceso
  - Con asignación variable: Hiperpag. en el sistema
- Necesario suspender a m. secundaria algunos procesos.

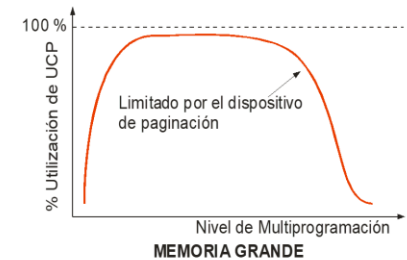
Proceso I



## Principales parámetros (4/4)

- ▶ Hay que equilibrar:
  - ▶ El número de procesos en memoria (**grado de multiprogramación**)
    - ▶ **Swapping:**
      - Trasiego de información entre M. Principal y M. secundaria.
    - ▶ **Hiperpaginación:**
      - Se produce cuando el número de fallos es muy elevado
      - El sistema está más tiempo intercambiando fragmentos que ejecutando instrucciones de usuario.

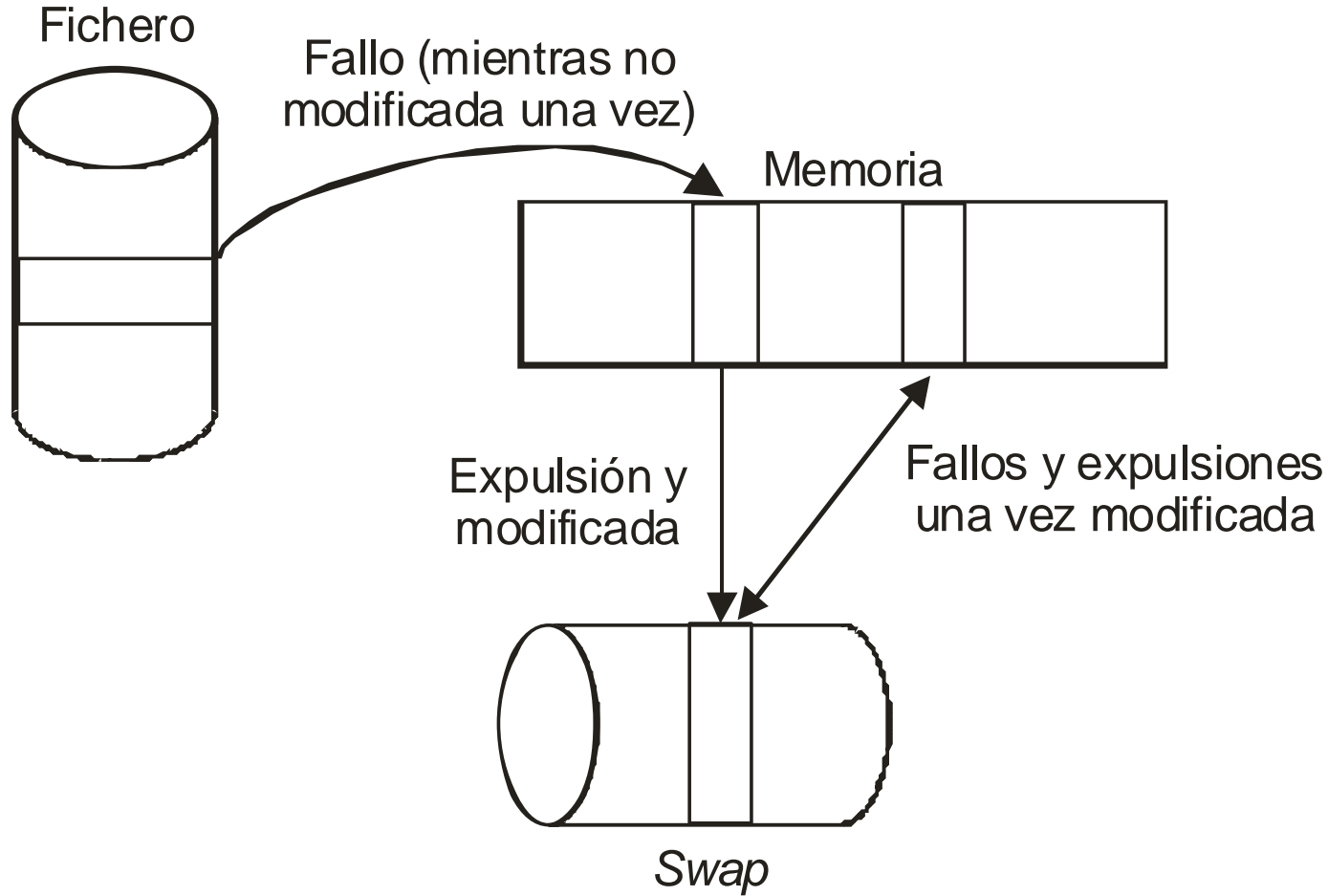
Comportamiento típico de la paginación con varios programas.



▶ 77

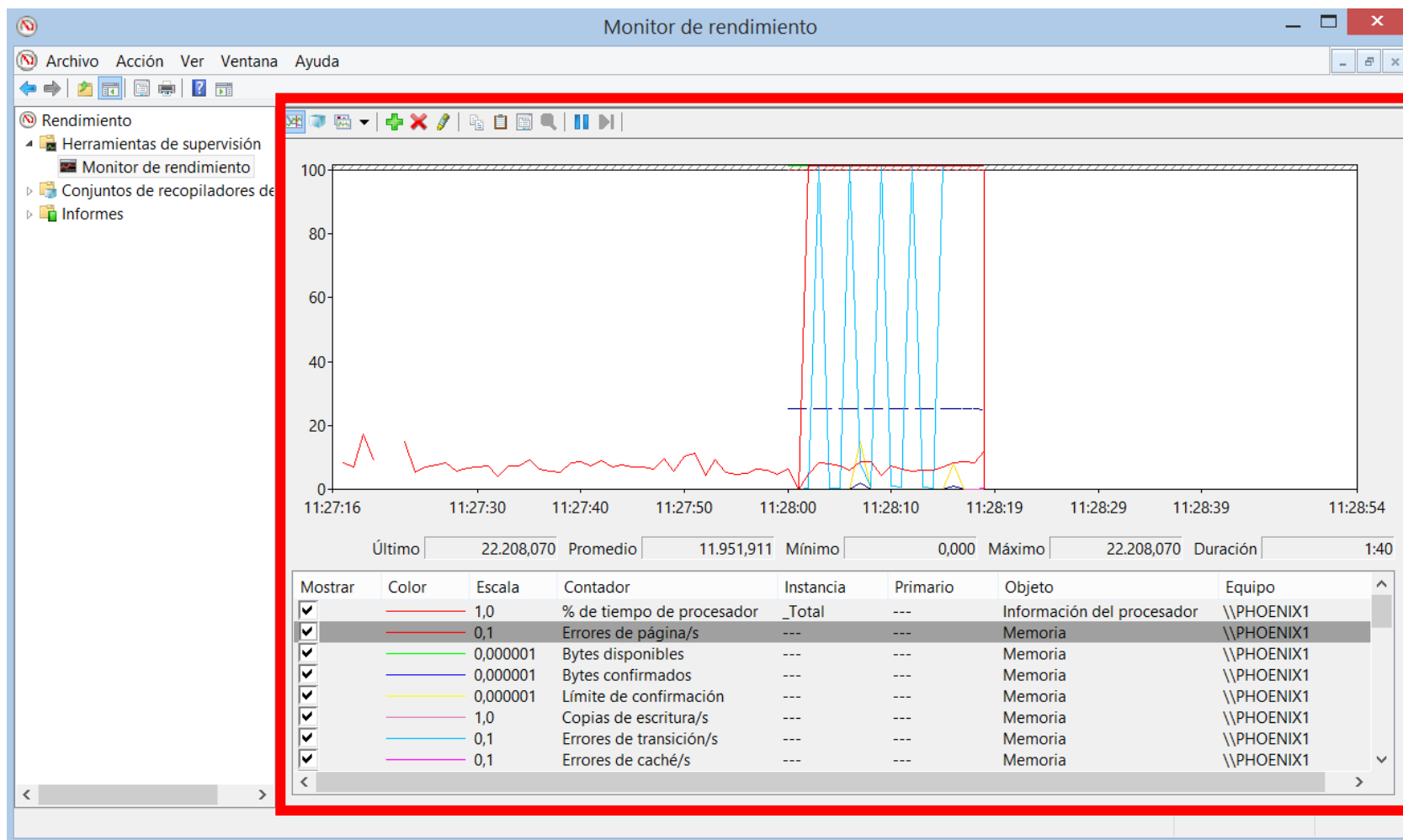
ARCOS @ UC3M  
Alejandro Calderón Mateos

# Ciclo de vida de página privada y en fichero





# Windows: perfmon





# Linux: ps, top, ...

---

```
arcos:~$ ps -o min_flt,maj_flt 1
MINFL MAJFL
18333  25
```

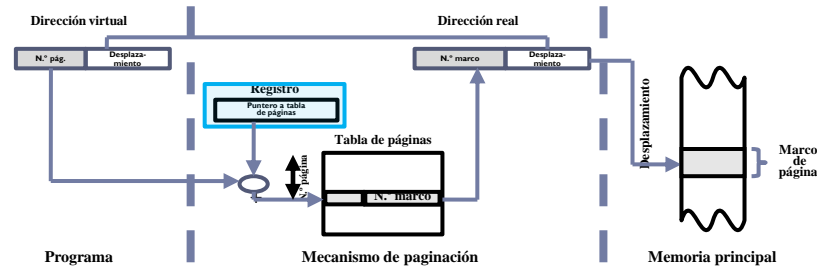
**Minor fault:** petición de reserva de página

**Major fault:** se precisa acceso a disco

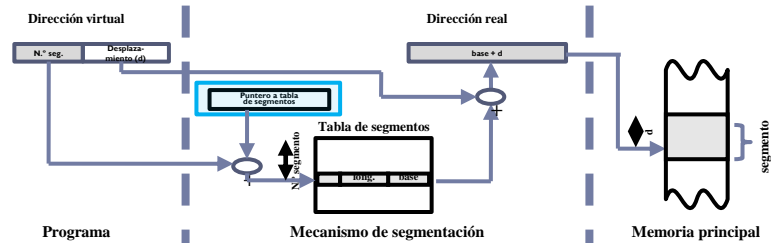
# Memoria virtual

## mecanismos de implementación

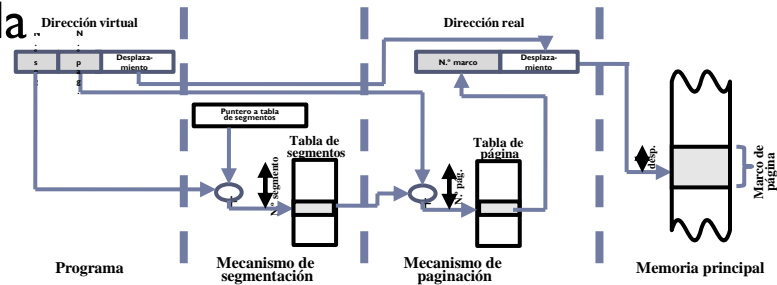
### ▶ Paginación



### ▶ Segmentación



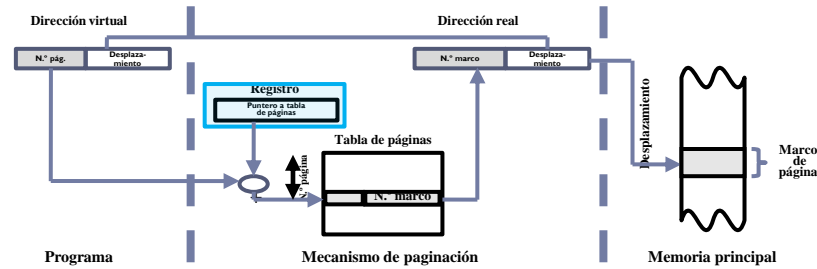
### ▶ Segmentación paginada



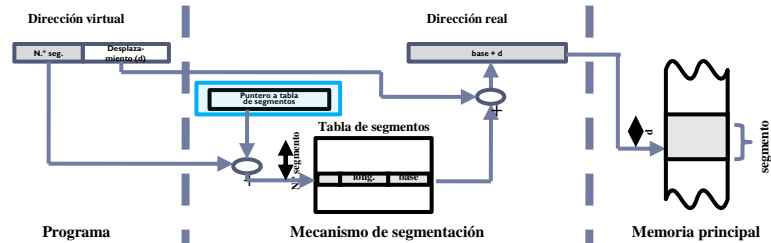
# Memoria virtual

## mecanismos de implementación

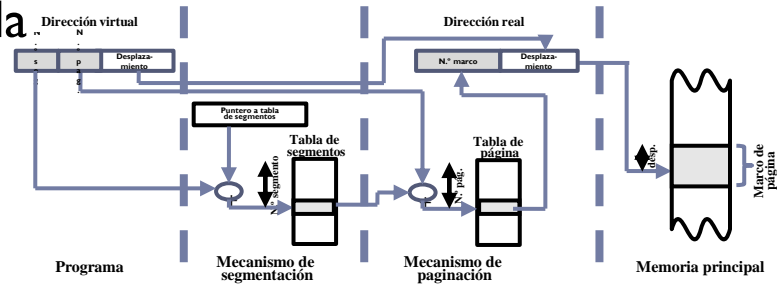
### ▶ Paginación

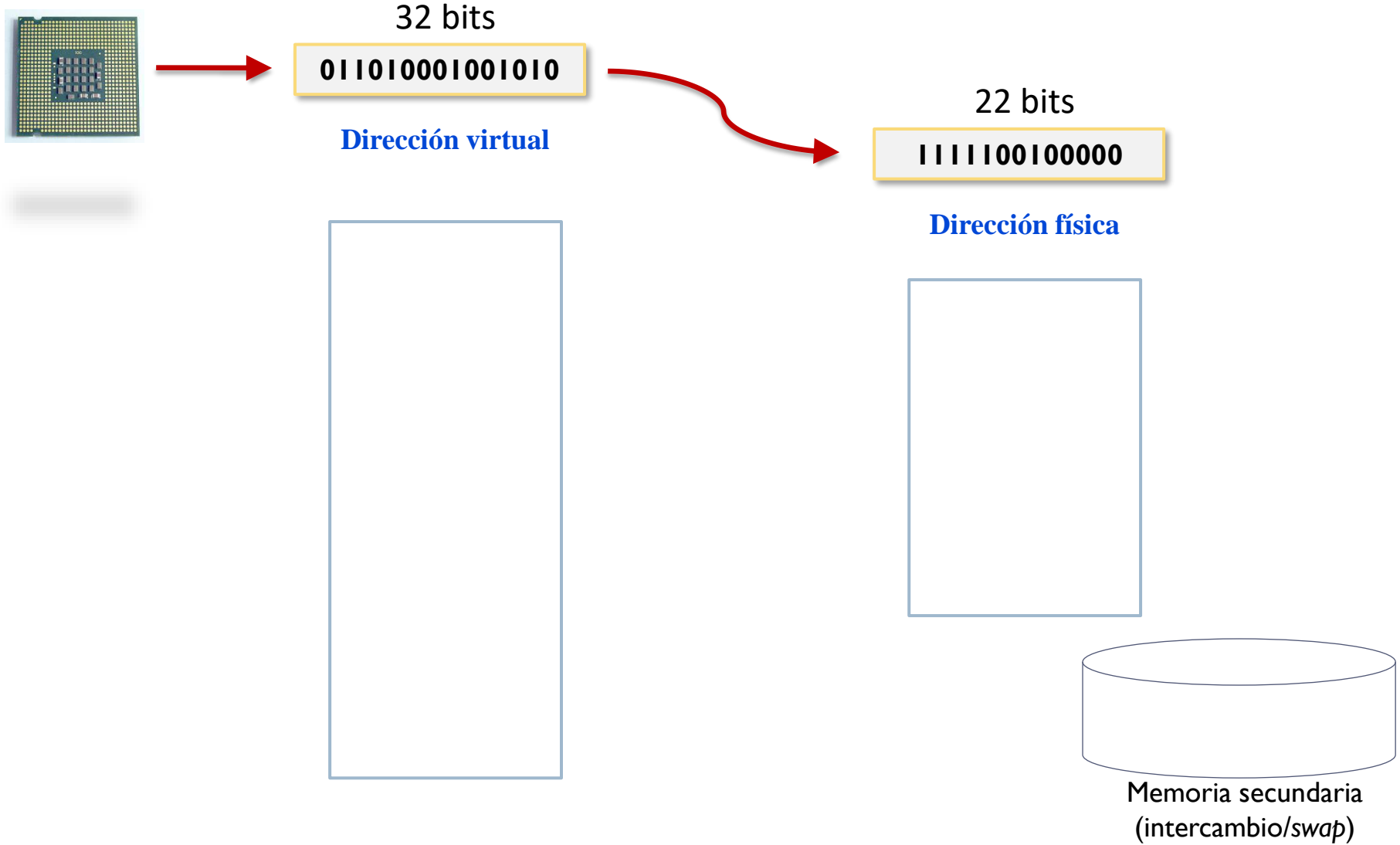


### ▶ Segmentación

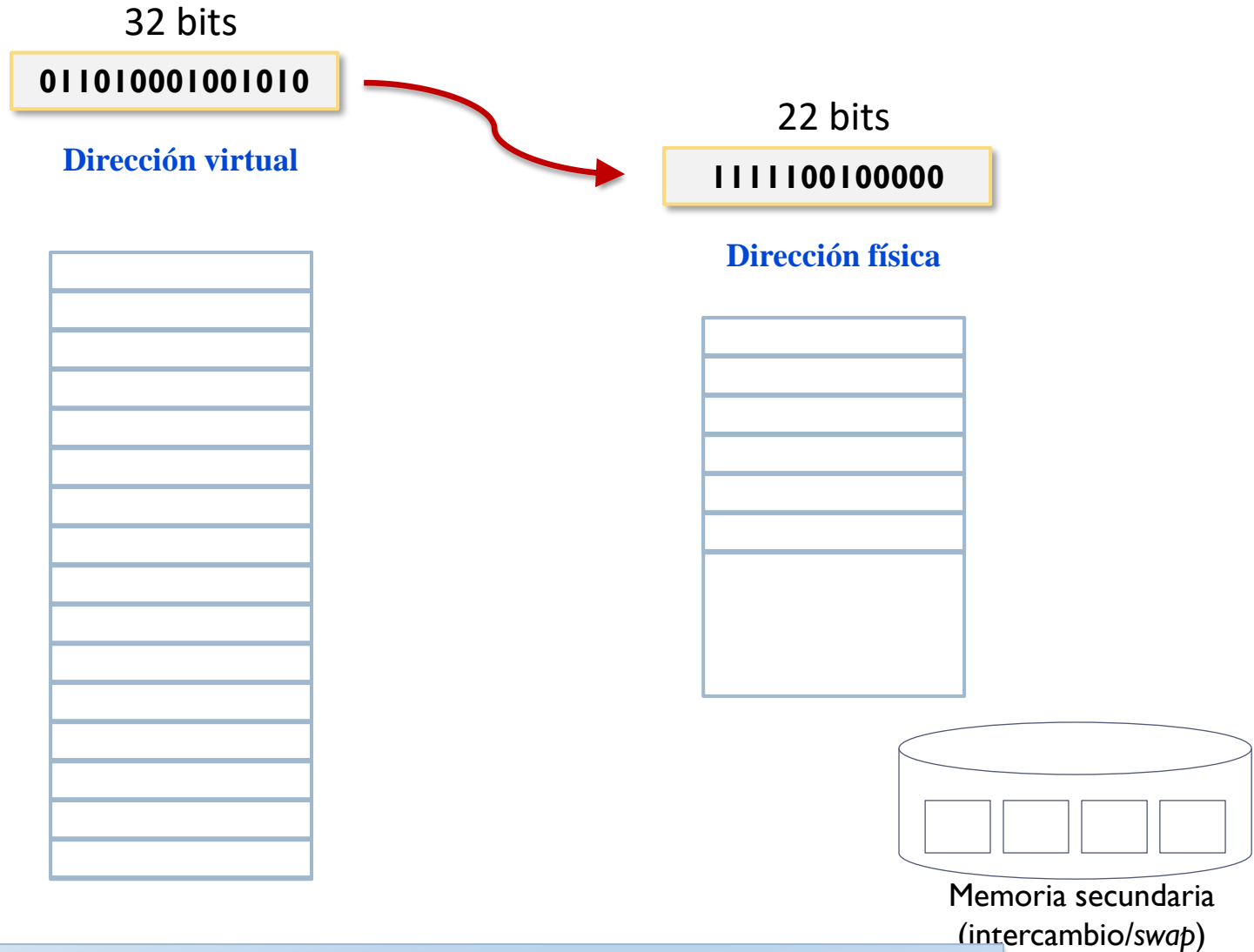


### ▶ Segmentación paginada

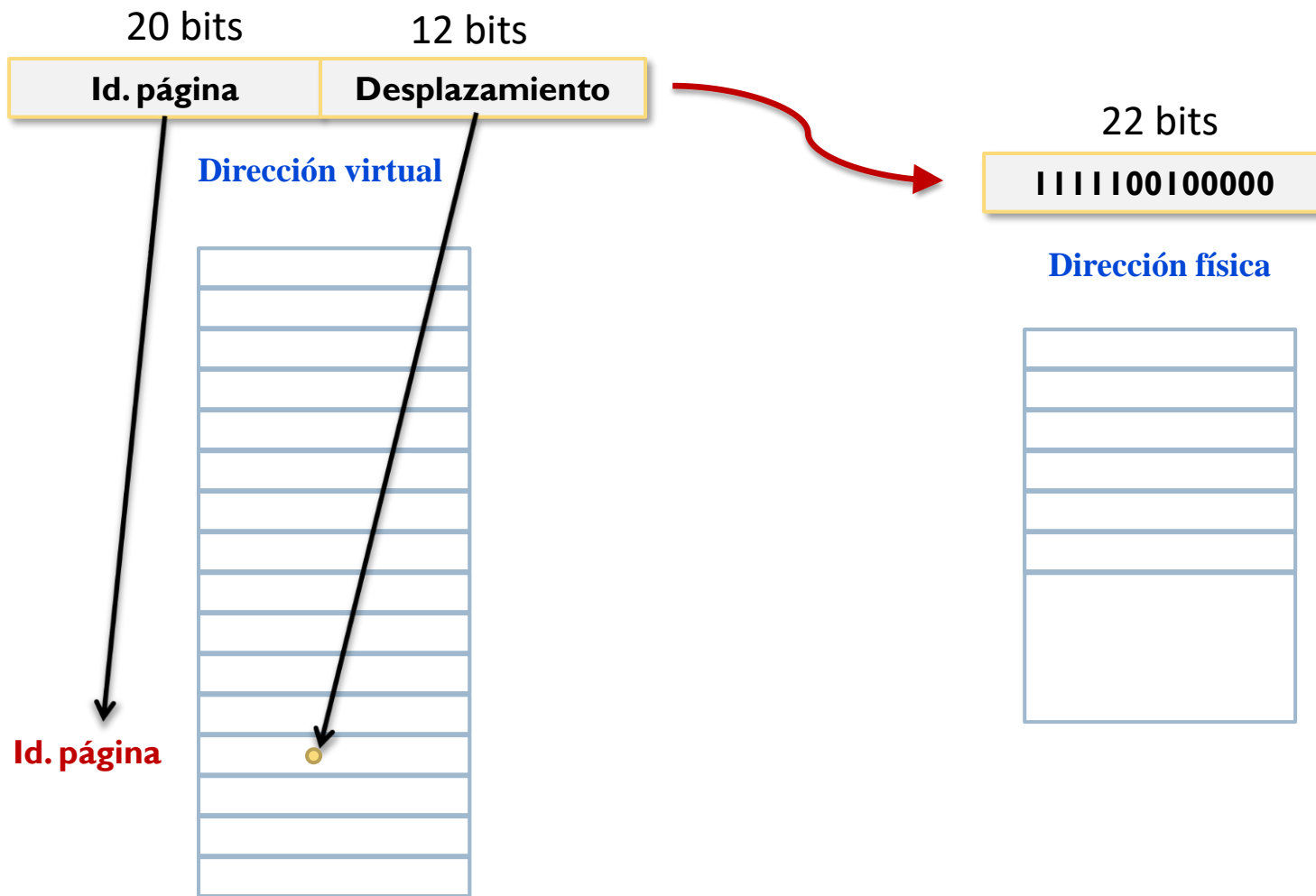




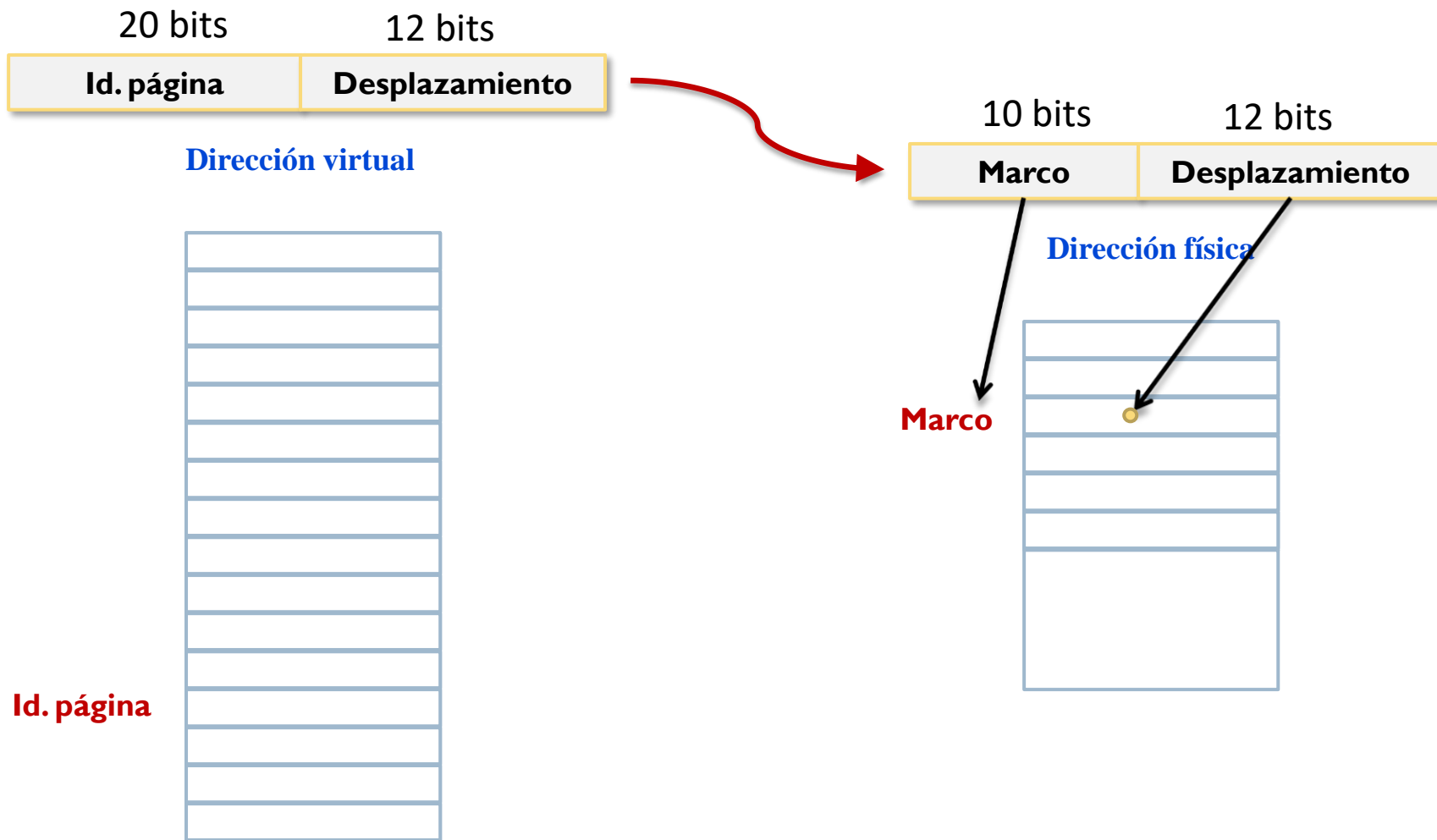




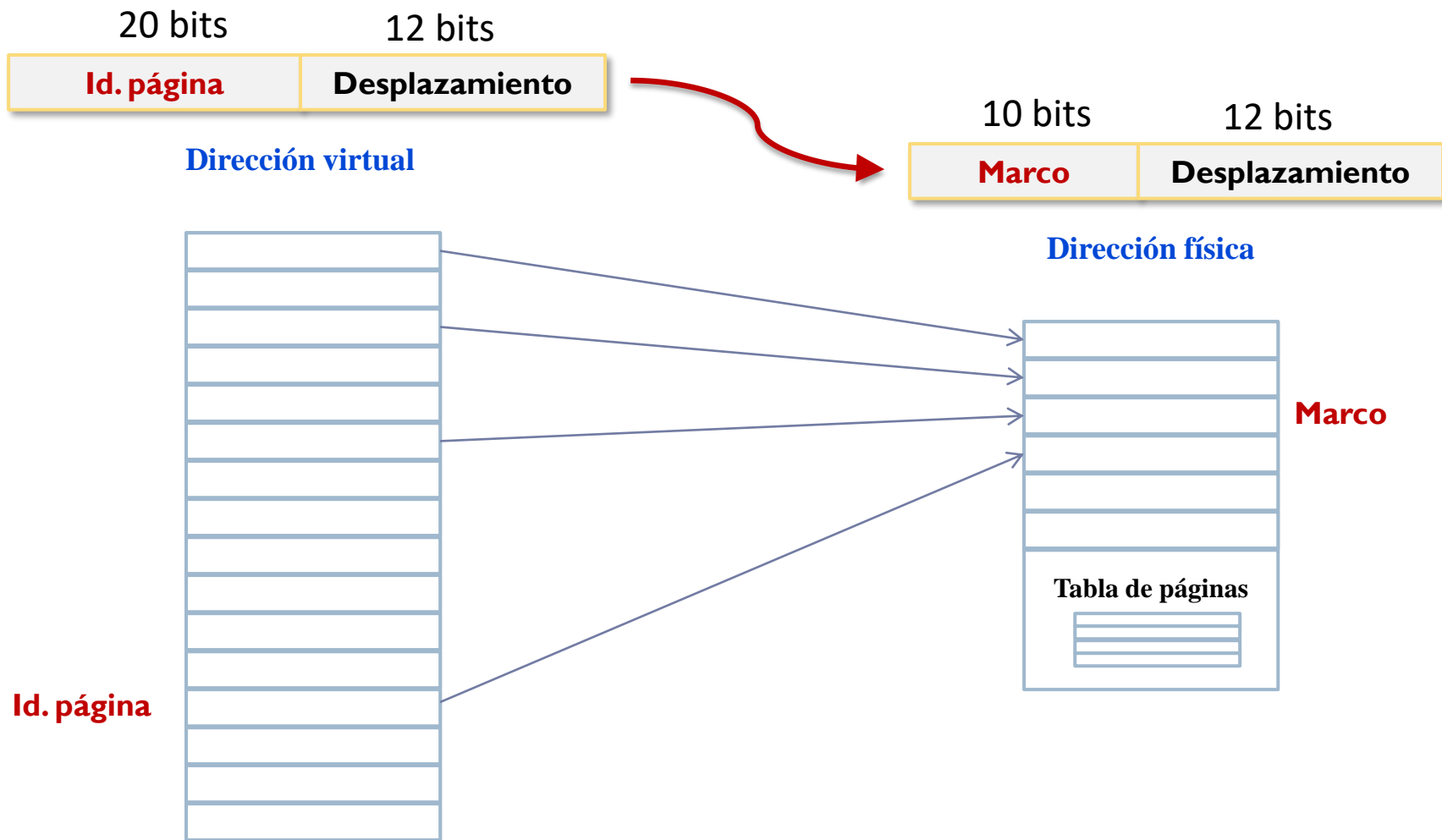
División en bloques del mismo tamaño -> páginas



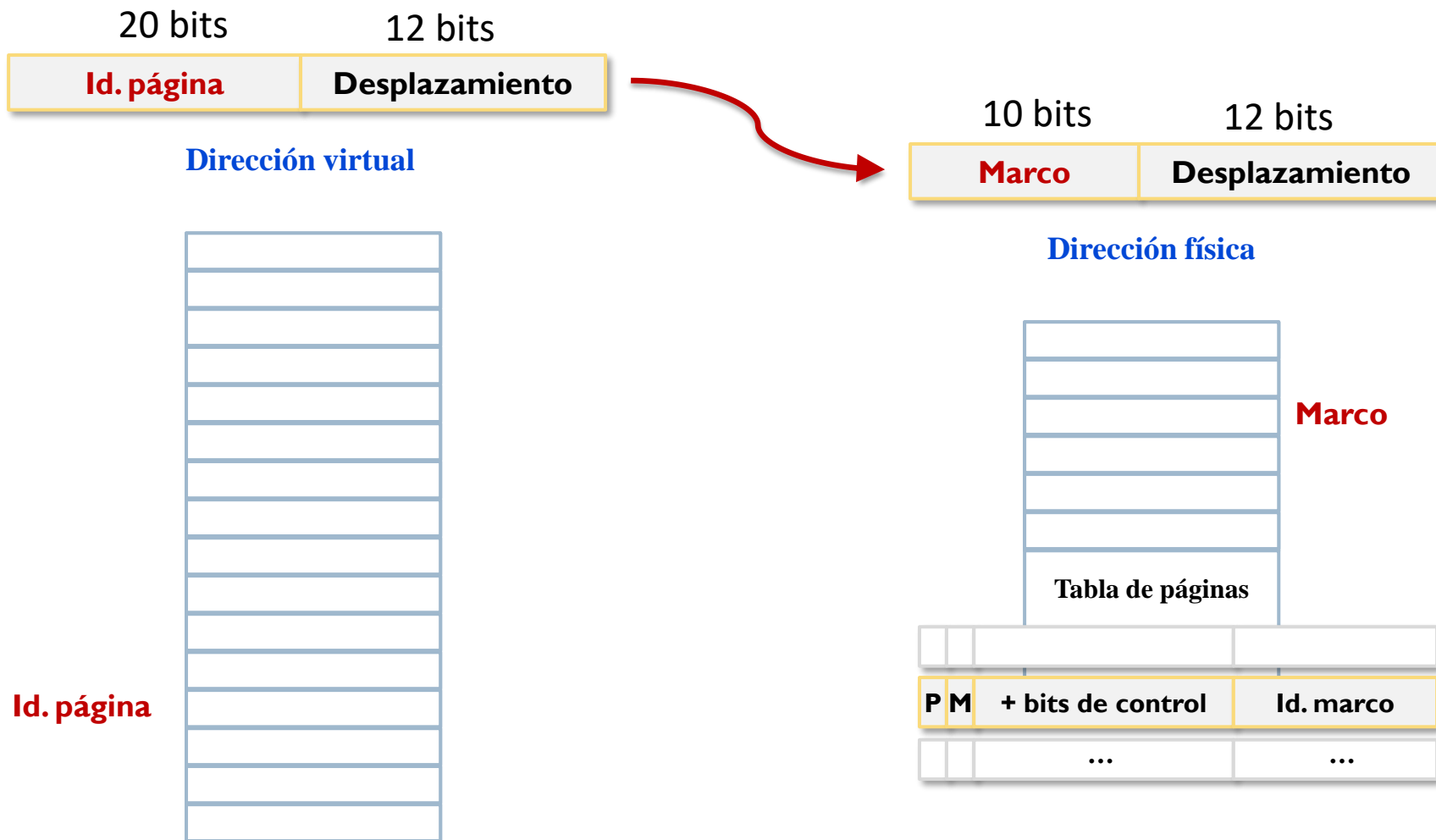
División en bloques del mismo tamaño -> páginas



División en bloques del mismo tamaño -> páginas

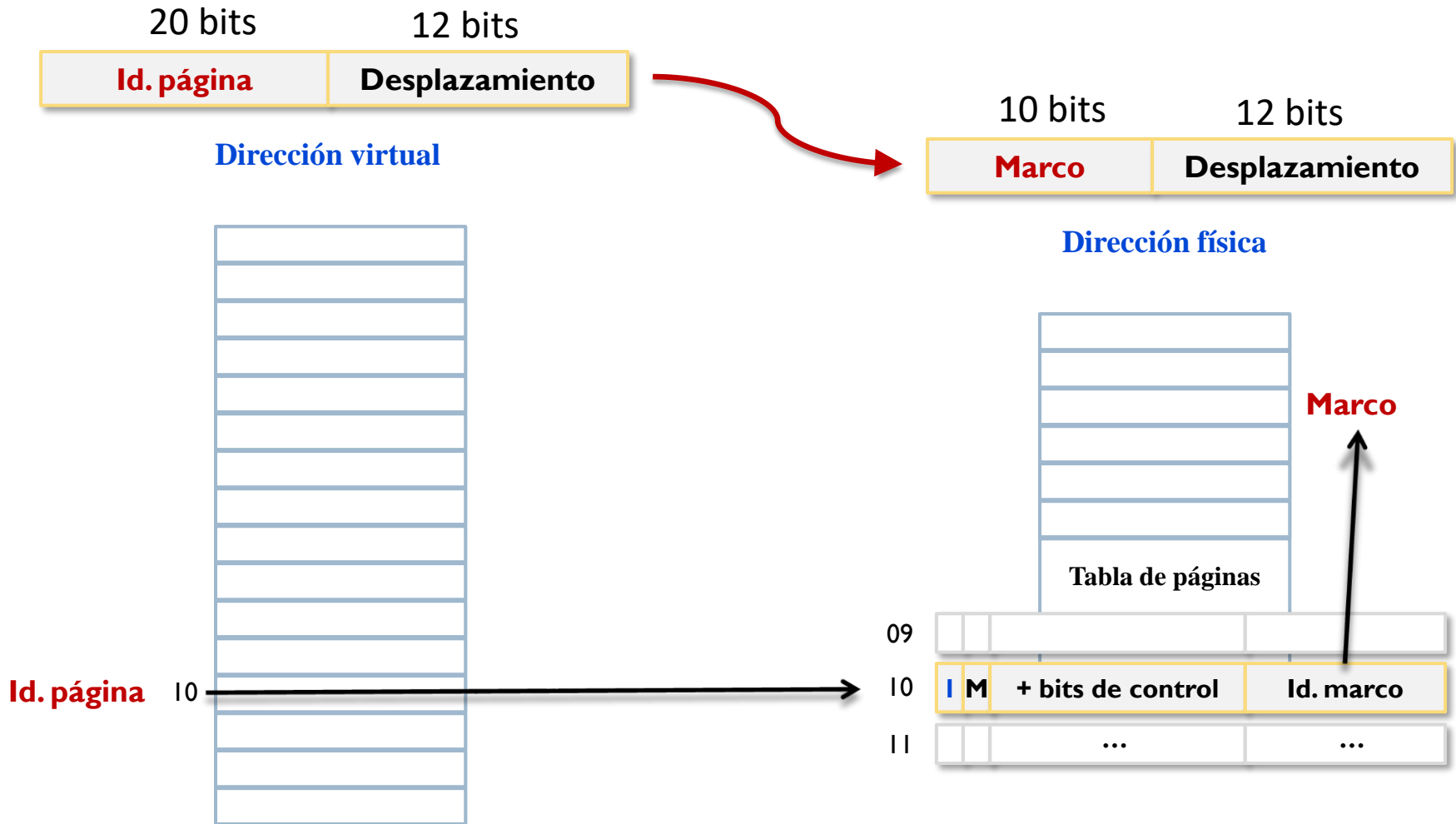


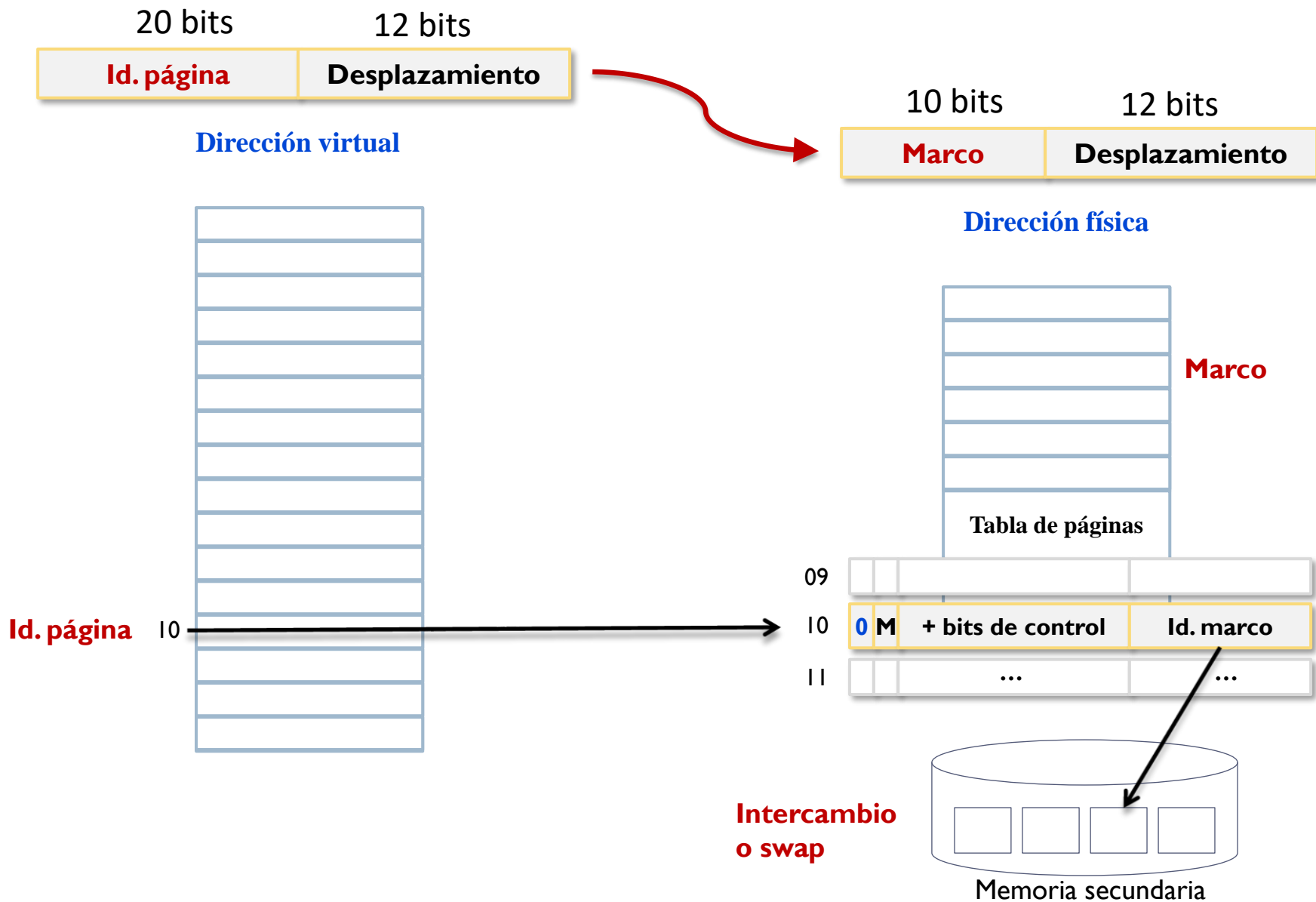
Correspondencia entre Id. página y marco -> T. páginas

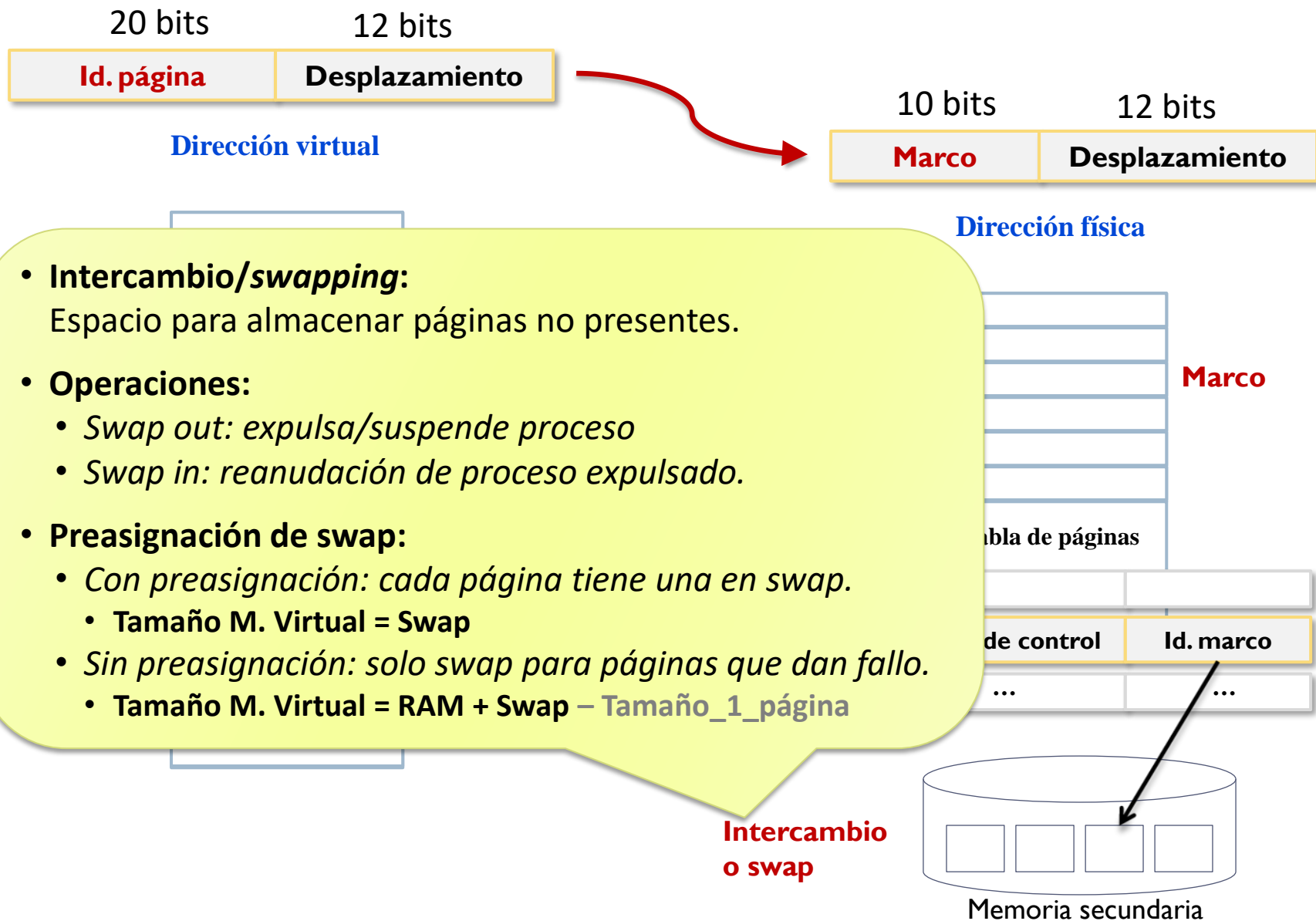


Correspondencia entre Id. página y marco -> T. páginas

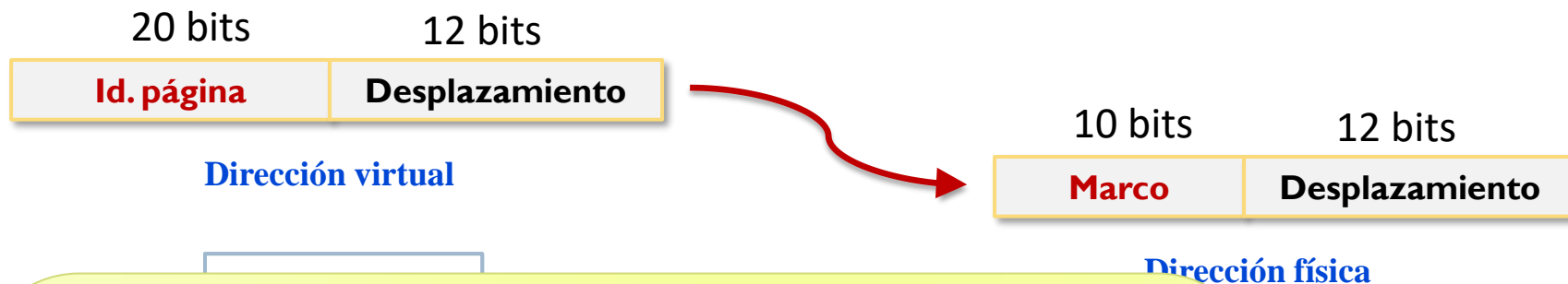
# Ejemplo



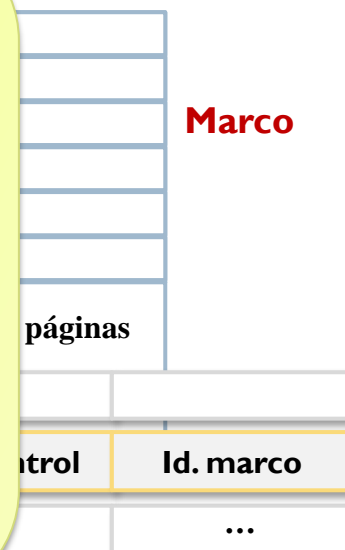








- **Tamaño de la página:**
  - Potencia de 2 y múltiplo de sector de disco.
  - Compromiso (ni grande ni pequeño, entre 1KiB y 16KiB)
- **Condicionado por factores contrapuestos:**
  - *Pequeño:*
    - Mejor ajuste al conjunto de trabajo, - frag. Interna.
  - *Grande:*
    - Tablas más pequeñas, mejor rendimiento de disco, + frag. interna.
- **La fija el procesador:**
  - Algunos permiten configurar distintos tamaños.



División en bloques del mismo tamaño -> páginas

# Ejemplo: memoria virtual con paginación

Sea un computador con direcciones virtuales de 32 bits y páginas de 4 KB.  
En este computador se ejecuta un programa cuya tabla de páginas es:

P	M	Perm.	Marco/Bloque
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

► Se pide:

- Tamaño que ocupa la imagen de memoria del programa
- Si la primera dirección virtual del programa es `0x00000000`, indique la última
- Dadas las siguientes direcciones virtuales, indique si generan fallo de página o no:
  - `0x00001000`
  - `0x0000101C`
  - `0x00004000`

# Ejercicio (solución)

---

P	M	Perm.	Marco/Bloque
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- ▶ El tamaño que ocupa la imagen de memoria del programa dependerá del número de páginas total que tenga asignado y el tamaño de la página:
  - ▶  $7 * 4 \text{ KB} = 28 \text{ KB}$

# Ejercicio (solución)

---

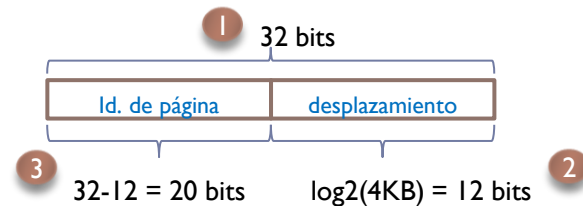
P	M	Perm.	Marco/Bloque
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- ▶ Si el tamaño total del programa es de 28 KB y la primera dirección virtual es la 0x00000000, la última dirección será:
  - ▶  $28 * 1024 - 1$

# Ejercicio (solución)

P	M	Perm.	Marco/Bloque
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- ▶ Lo primero es conocer el formato de la dirección virtual

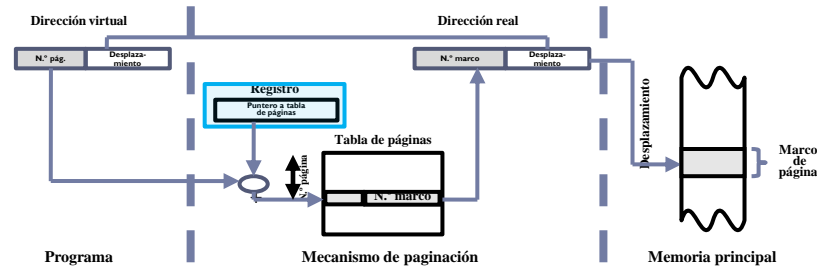


- ▶ Para cada dirección virtual, se extrae el identificador de página, se busca en la Tabla de páginas su entrada, y se ve si el bit de presente (P) está a 1:
  - 0x**00001**000 -> no
  - 0x**00001**01C -> no
  - 0x**00004**000 -> si

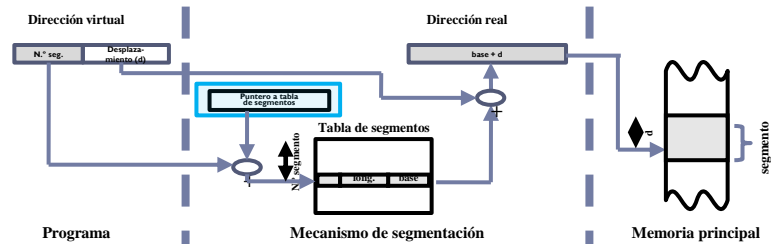
# Memoria virtual

## mecanismos de implementación

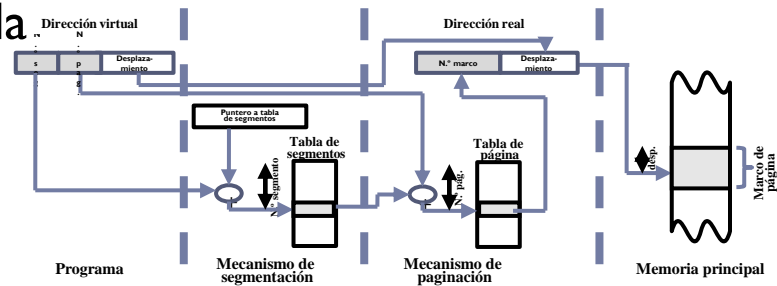
### ▶ Paginación

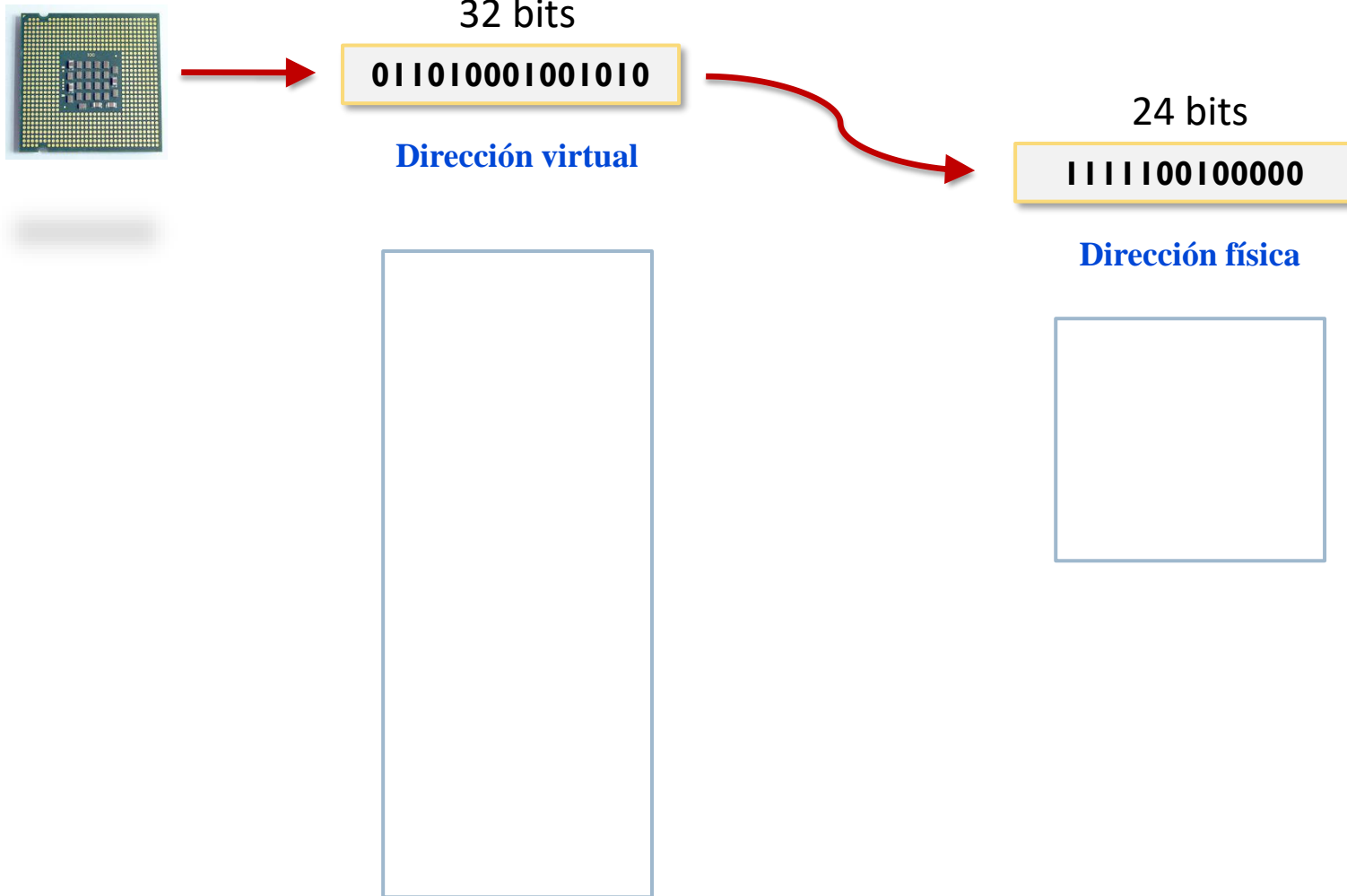


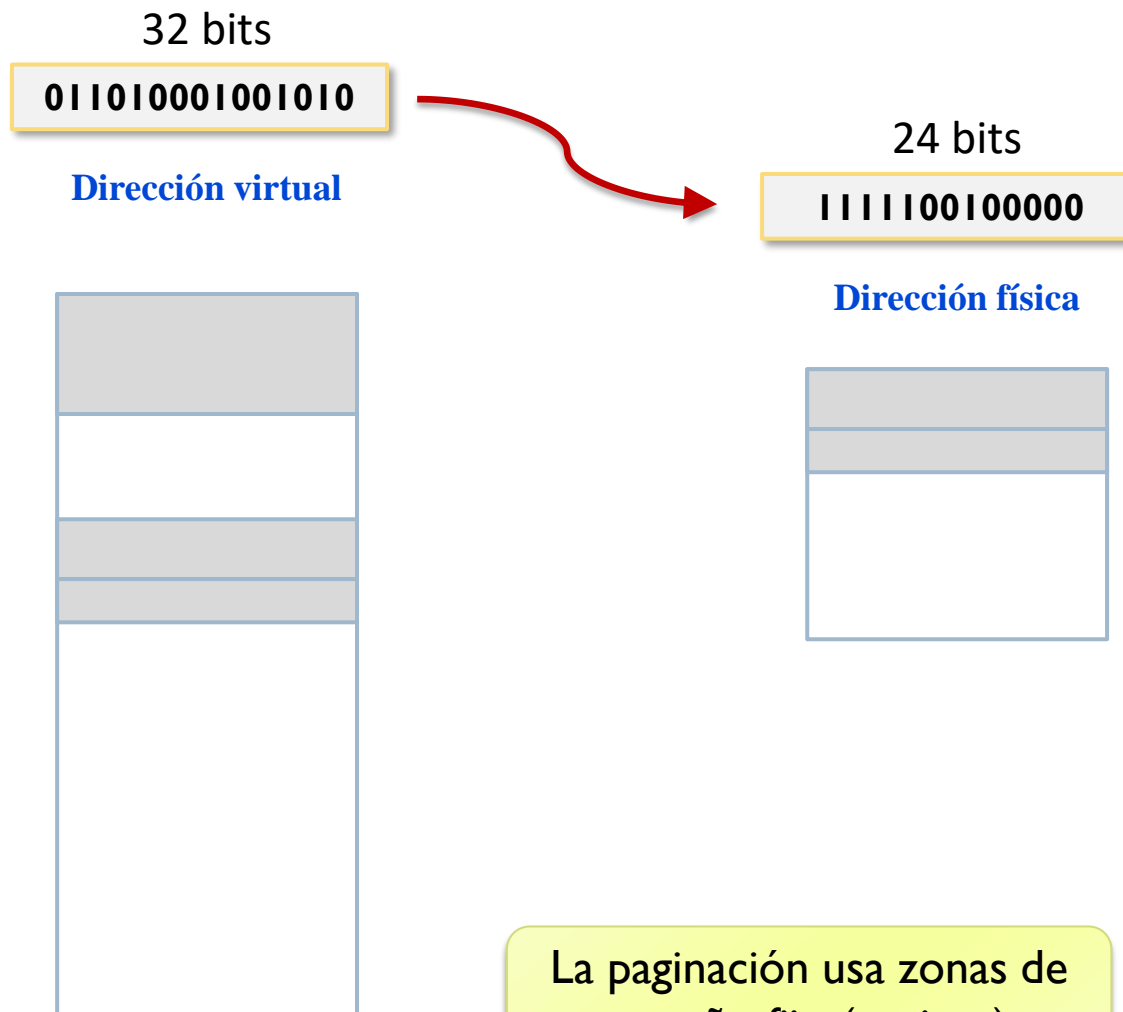
### ▶ Segmentación



### ▶ Segmentación paginada

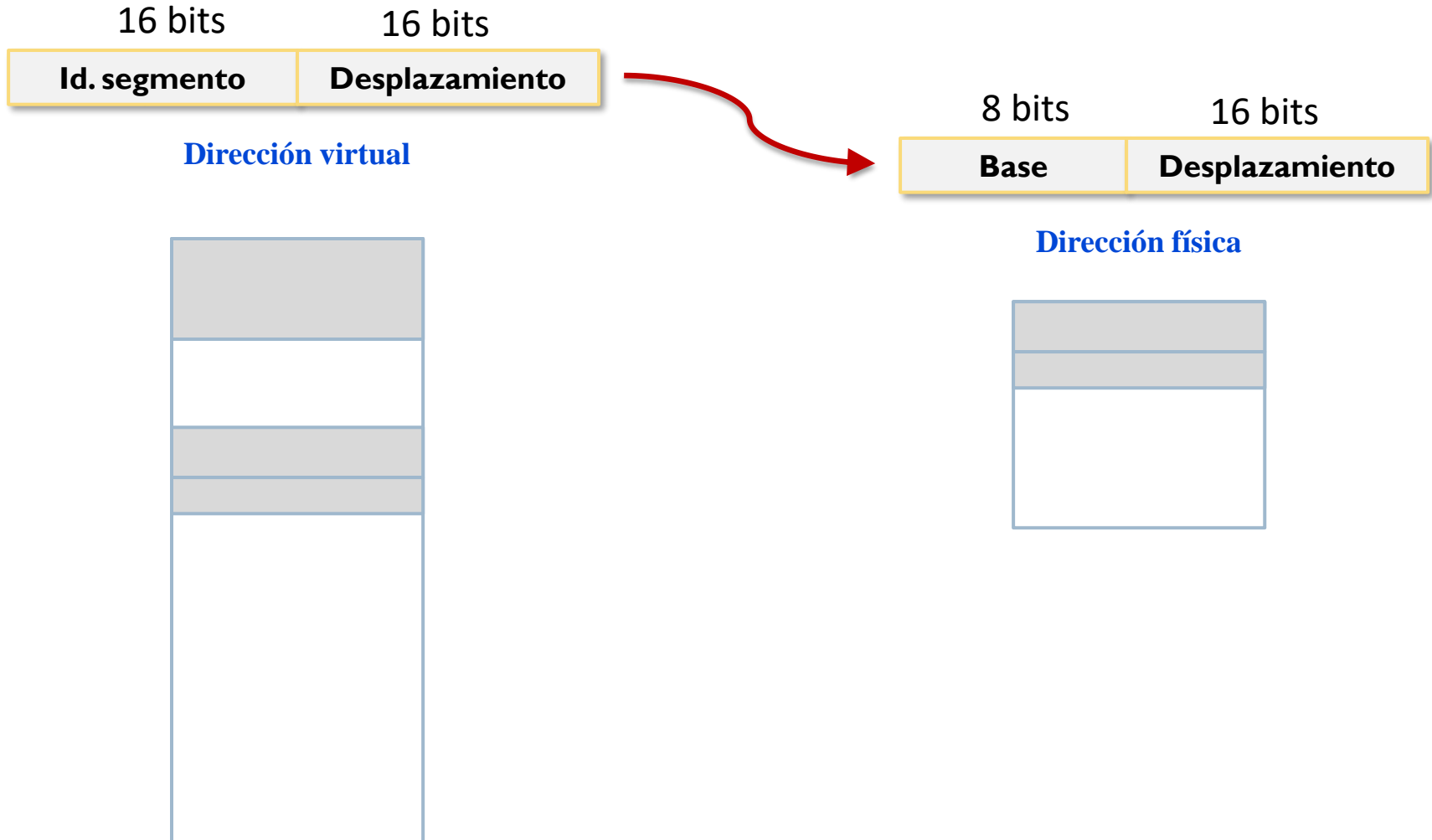




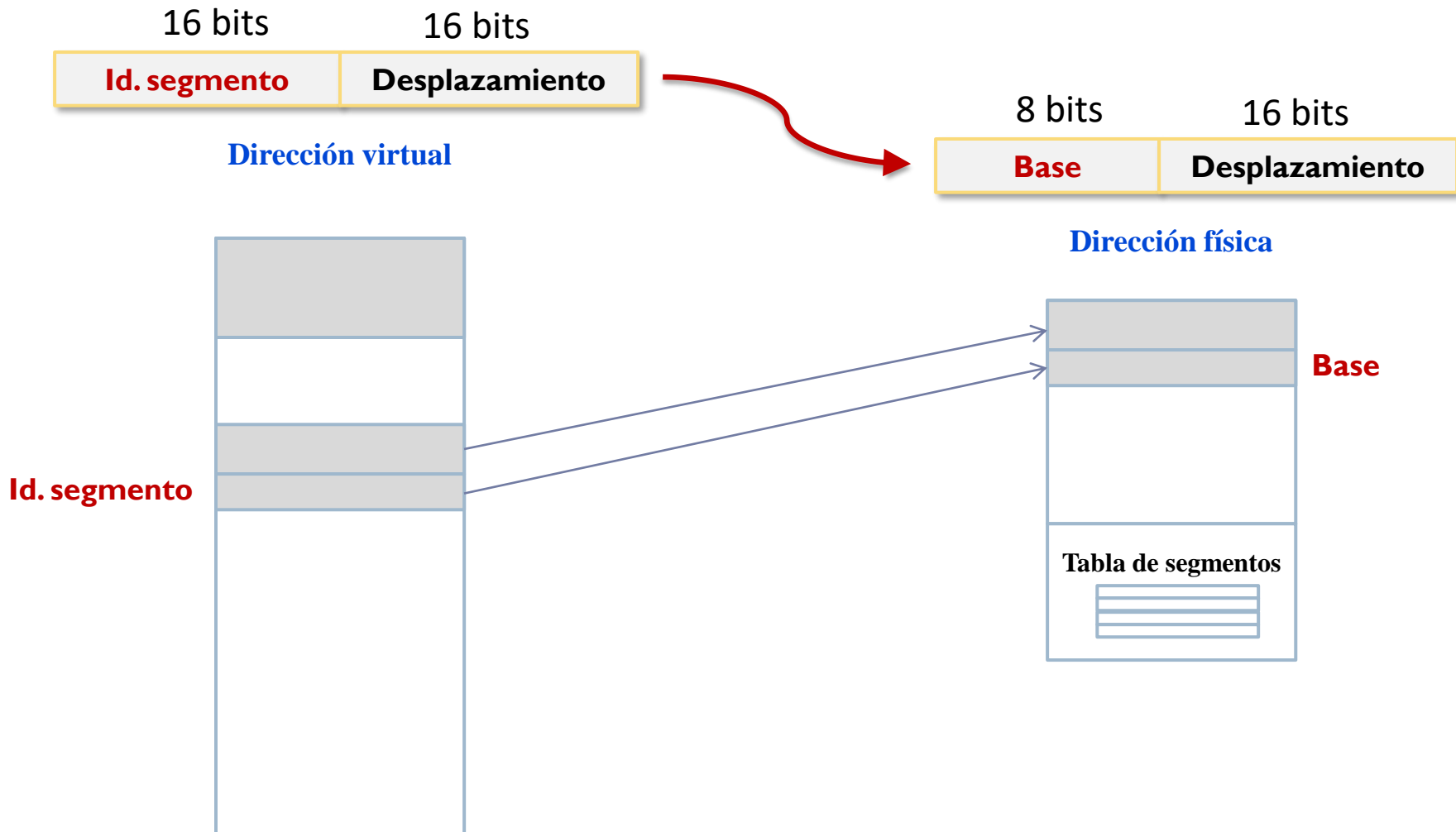


División en zonas de tamaño variable -> segmentos

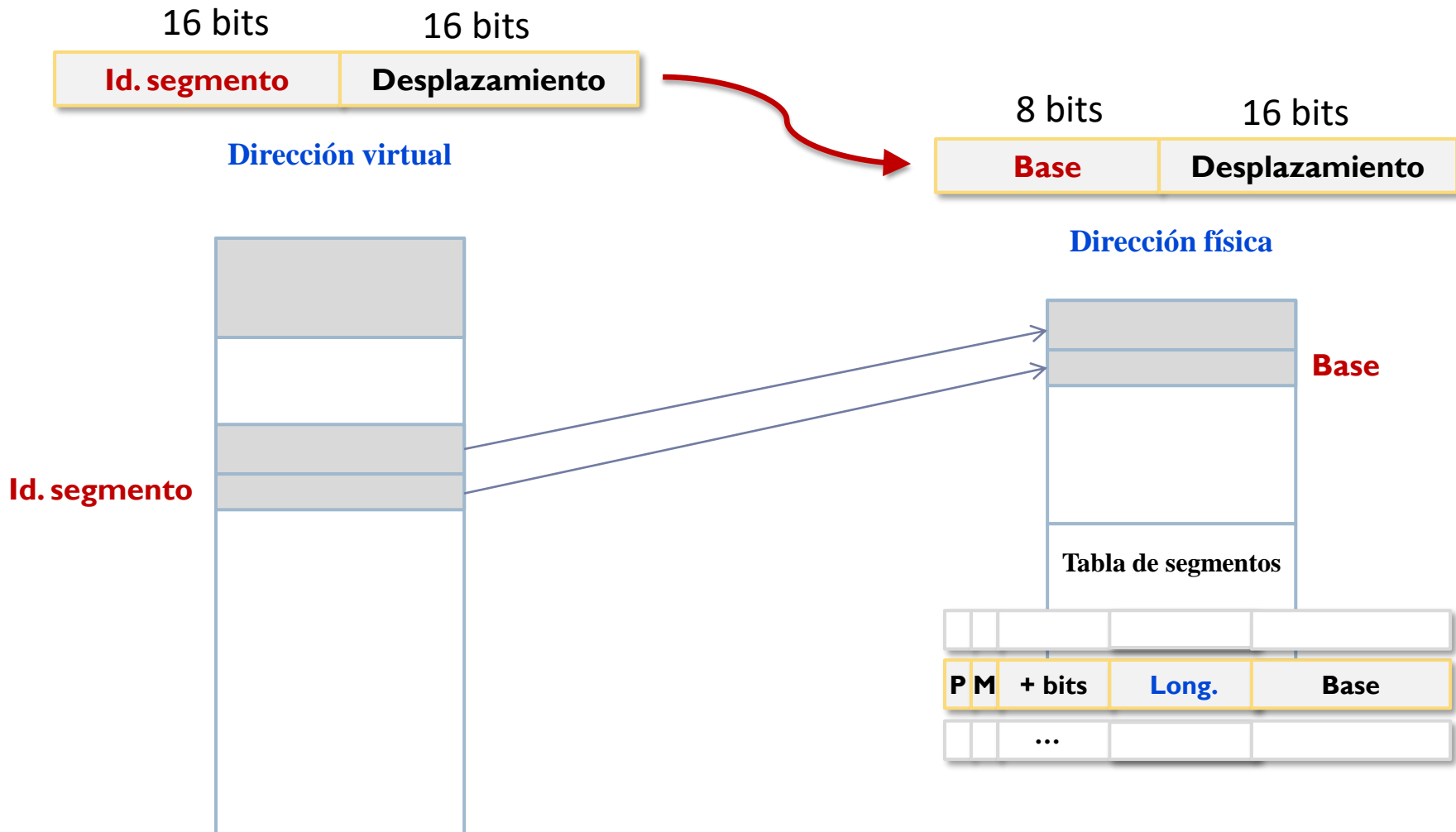




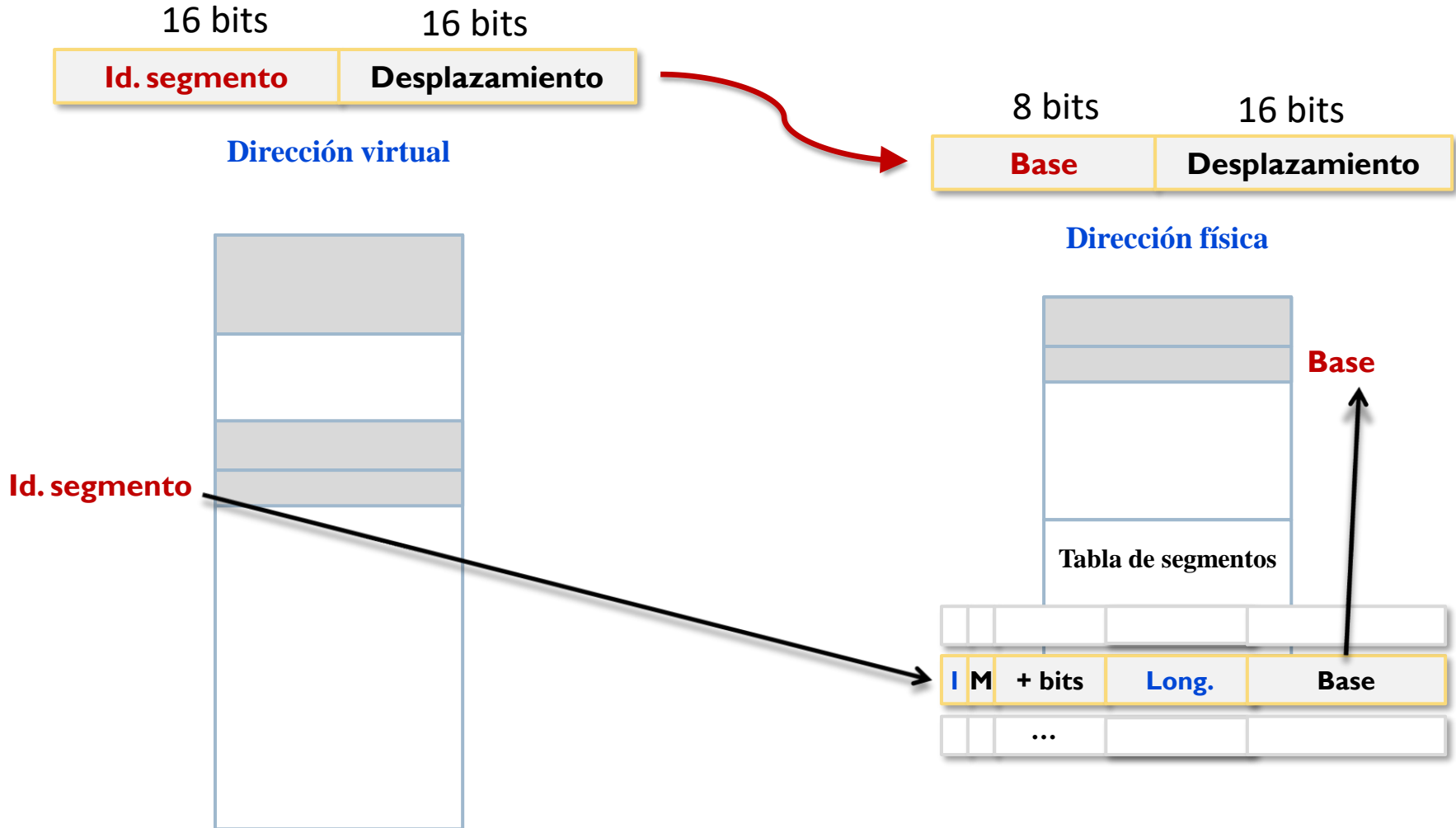
División en zonas de tamaño variable -> segmentos

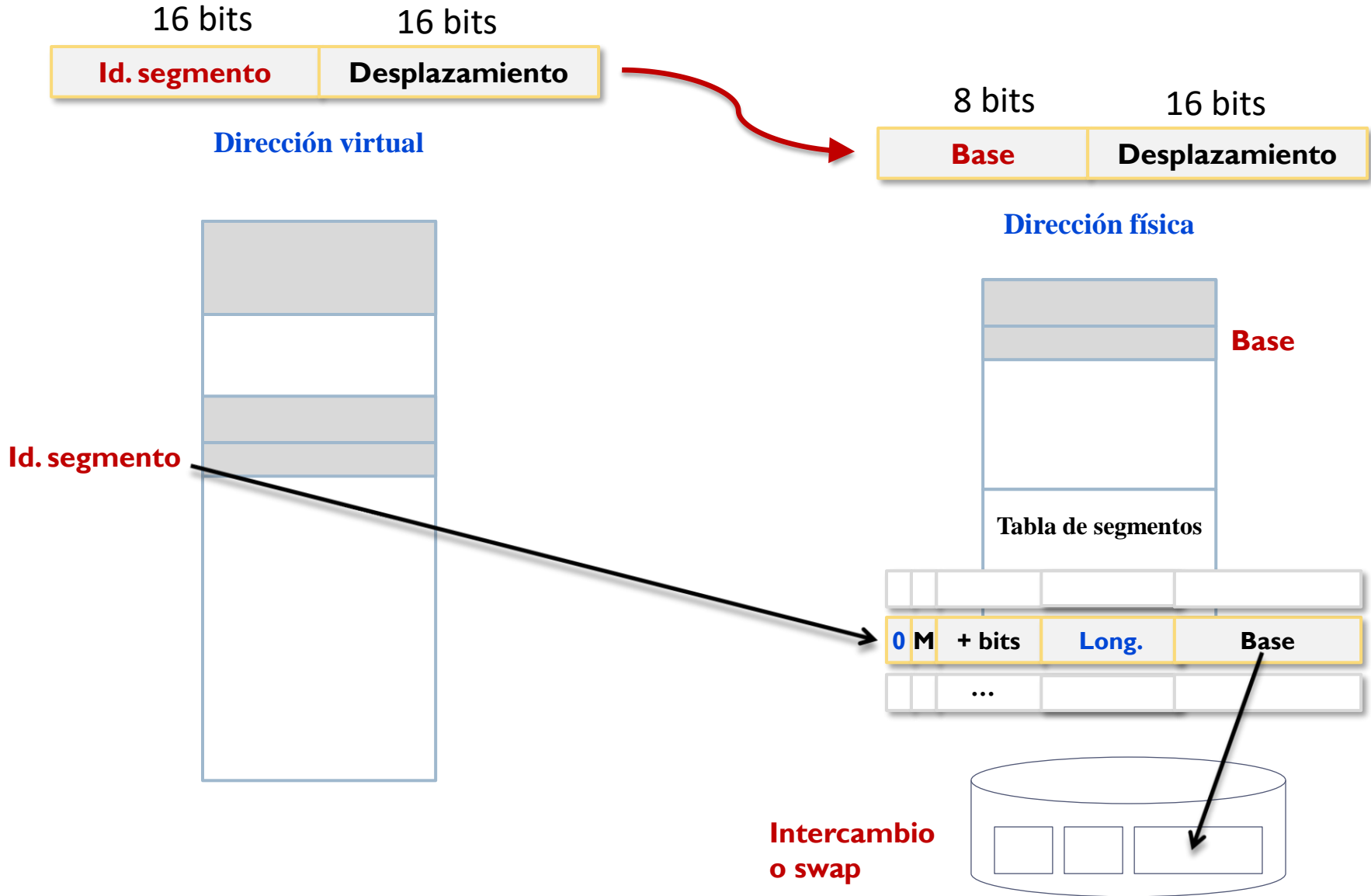


Correspondencia M.V. y M.F. -> tabla de segmentos



Correspondencia M.V. y M.F. -> tabla de segmentos

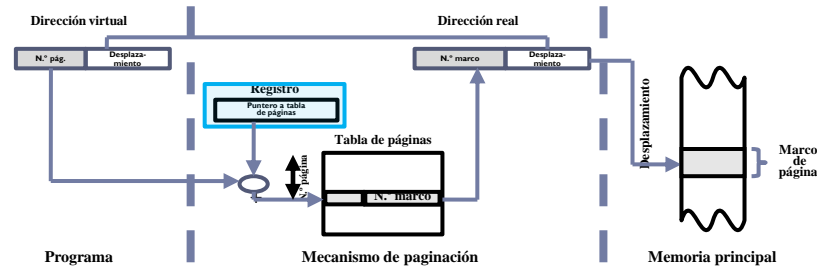




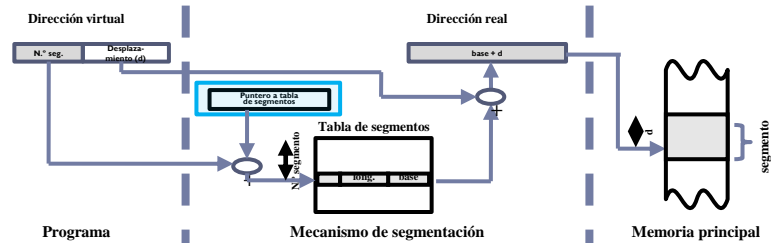
# Memoria virtual

## mecanismos de implementación

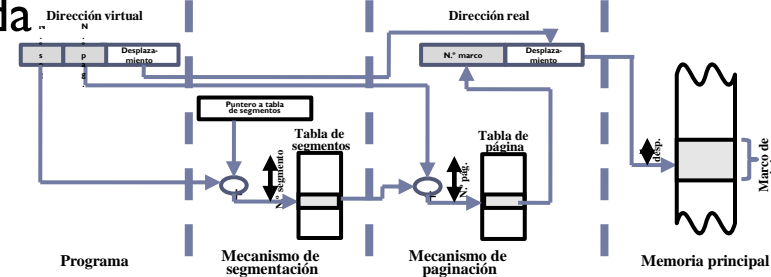
### ▶ Paginación



### ▶ Segmentación



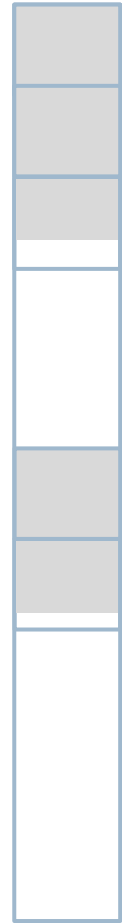
### ▶ Segmentación paginada



# Memoria virtual: segmentación y paginación

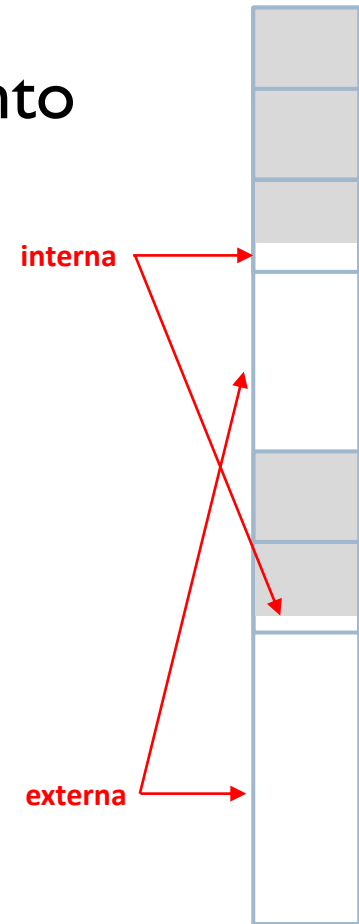
---

- ▶ Entrada en la tabla de segmentos ‘apunta’ a una tabla de páginas asociada al segmento
  - ▶ Los segmentos de tamaño variable se fragmentan en páginas de tamaño fijo



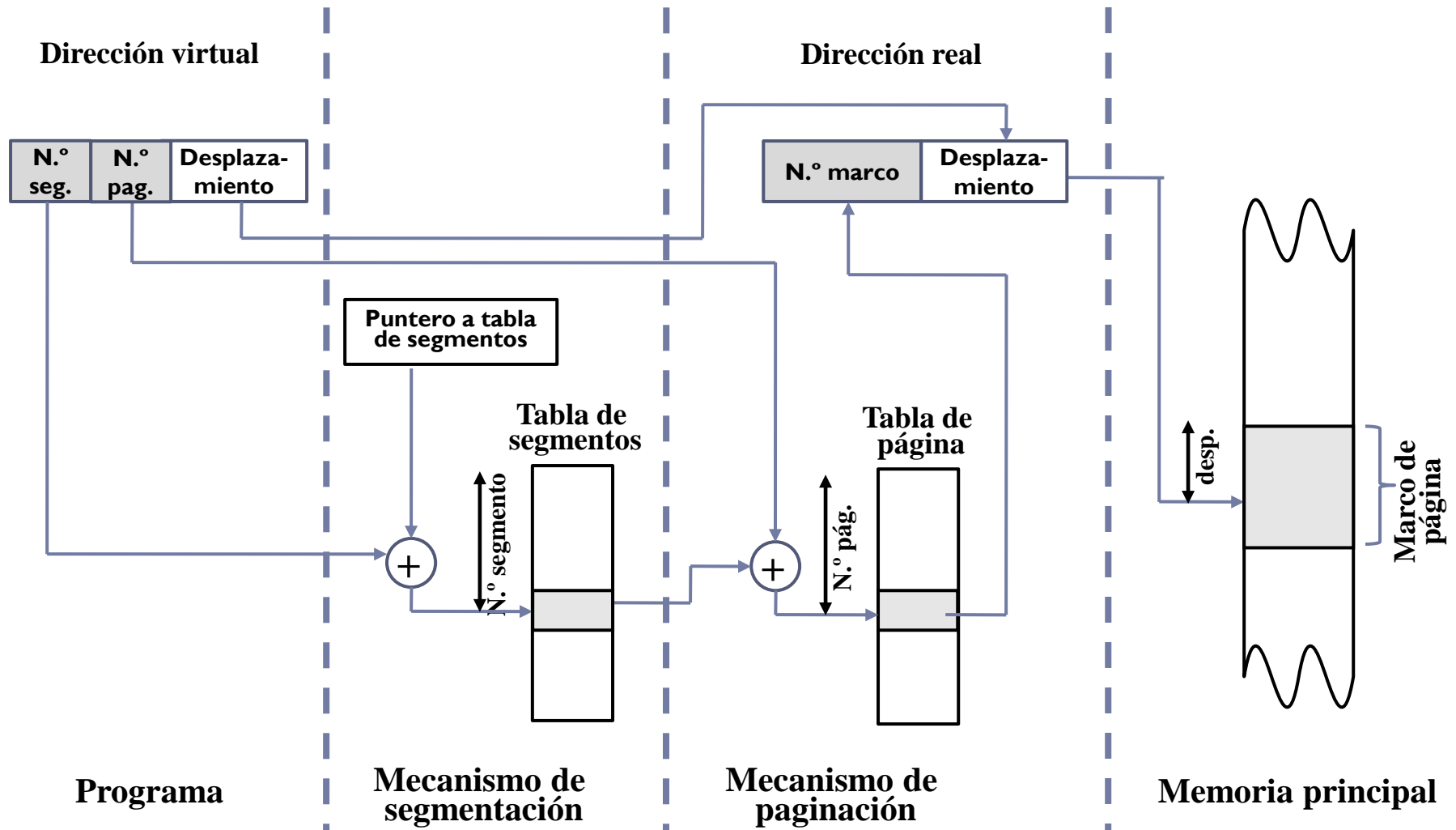
# Memoria virtual: segmentación y paginación

- ▶ Entrada en la tabla de segmentos ‘apunta’ a una tabla de páginas asociada al segmento
  - ▶ Los segmentos de tamaño variable se fragmentan en páginas de tamaño fijo
- ▶ Lo mejor de las dos soluciones:
  - ▶ Segmentación:
    - ▶ Facilita operaciones con regiones de memoria
    - ▶ Evita la fragmentación interna (tiene externa)
  - ▶ Paginación:
    - ▶ Optimiza el acceso a la memoria secundaria
    - ▶ Evita la fragmentación externa (tiene interna)





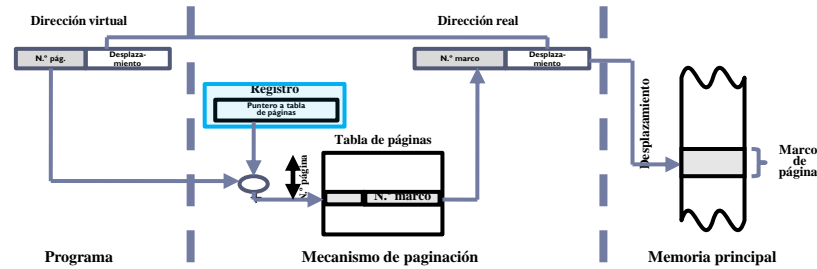
# Traducción de direcciones segmentación paginada



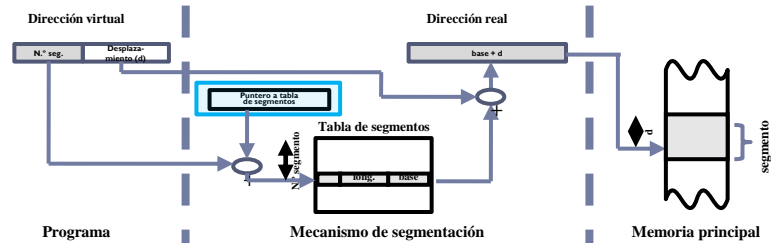
# Memoria virtual

## resumen

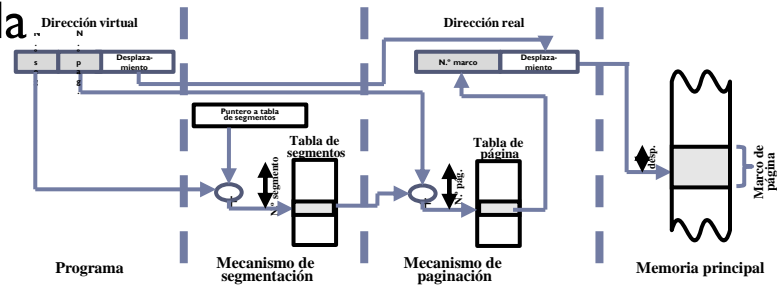
### ▶ Paginación



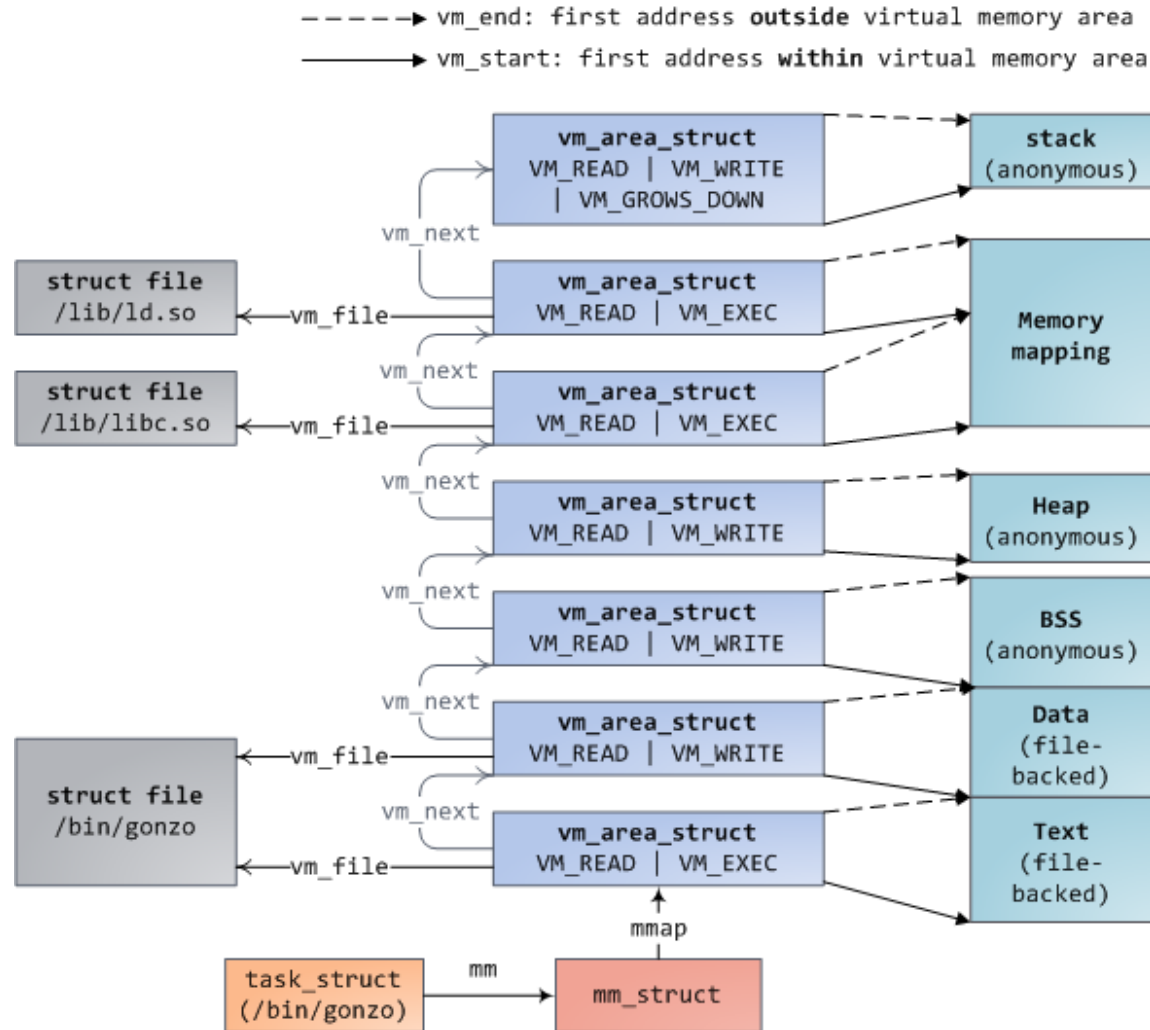
### ▶ Segmentación



### ▶ Segmentación paginada

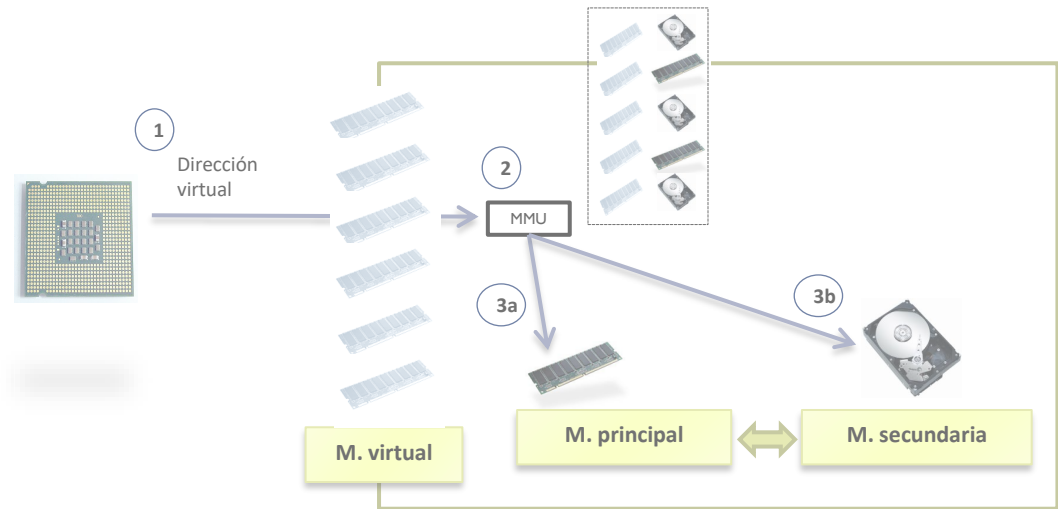


# Estructuras principales de gestión gestión de memoria de un proceso: Linux



# Gestión de memoria

## aspectos avanzados



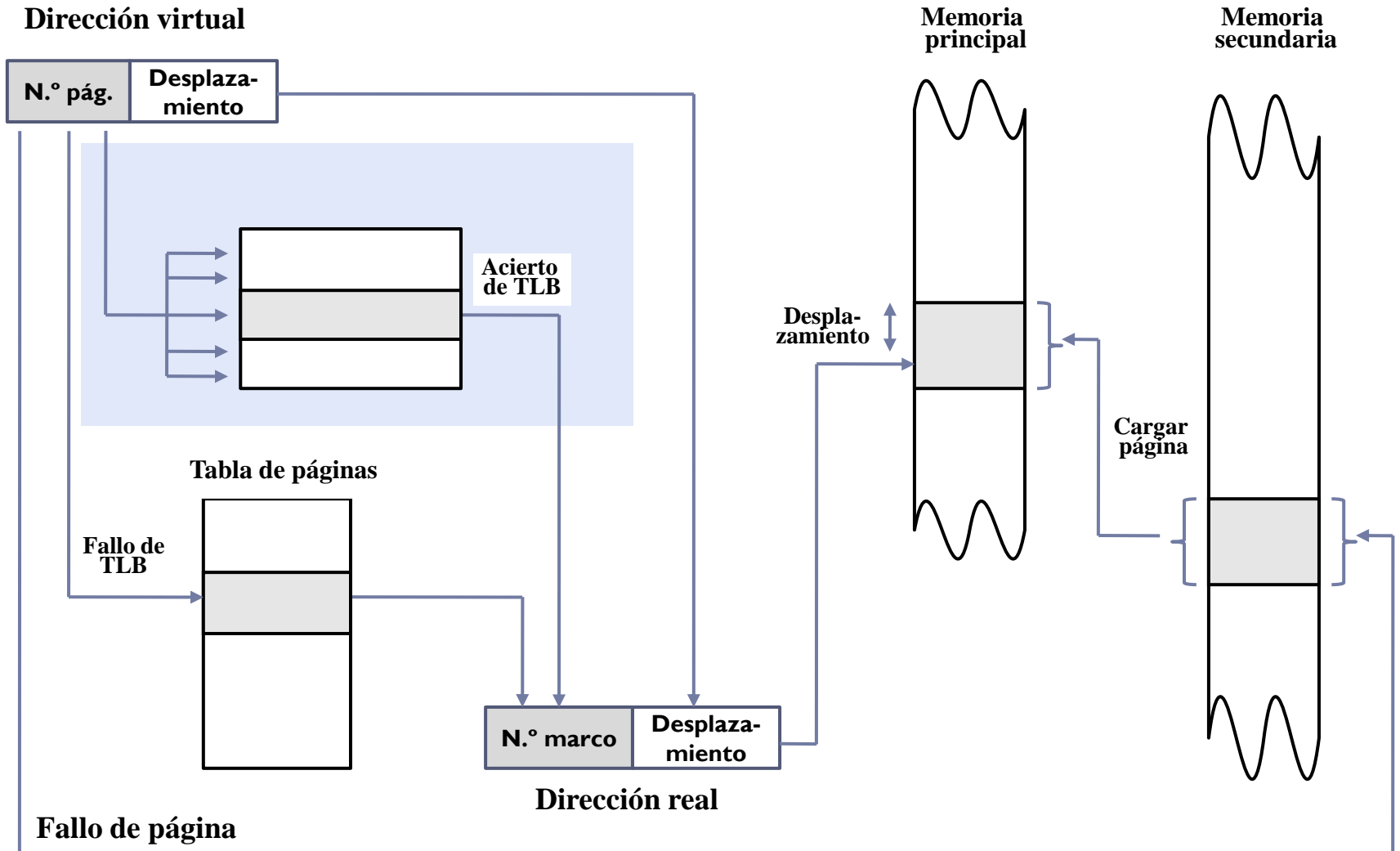
- ▶ TLB
- ▶ Tablas multinivel
- ▶ Kernel/Procesos

# Cache de traducciones

---

- ▶ Memoria virtual basado en tablas de páginas:
  - ▶ Problema: sobrecarga de acceso a memoria (2 accesos)
    - ▶ A la tabla de páginas/segmentos + al propio dato o instrucción
  - ▶ Solución: **TLB**
    - ▶ **Caché de traducciones**
- ▶ **TLB** (buffer de traducción adelantada)
  - ▶ Memoria caché asociativa que almacena las entradas de la tabla de página usadas más recientemente
  - ▶ Permite acelerar el proceso de búsqueda del marco

# Traducción de direcciones (con TLB)

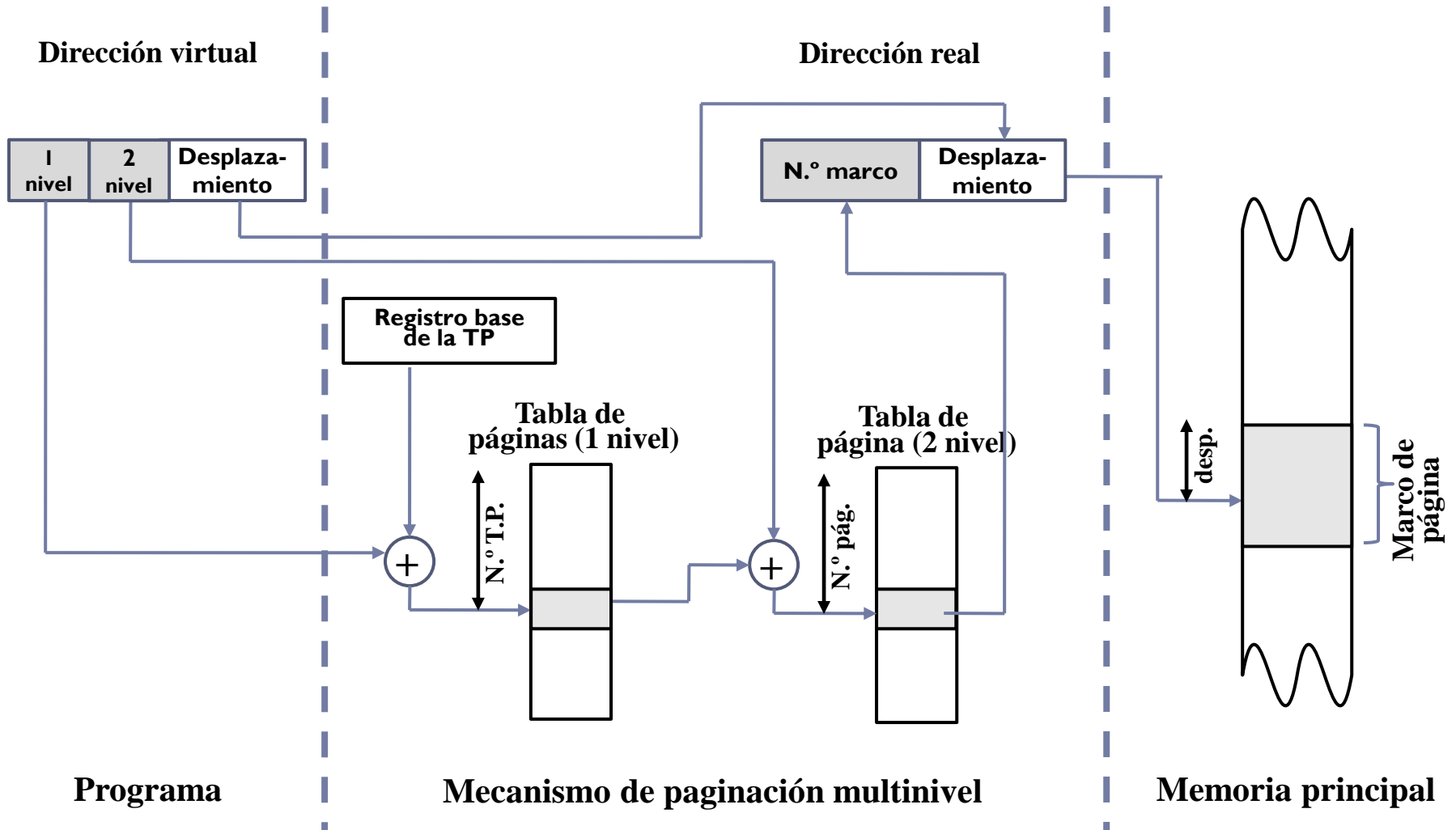


# Tablas multinivel

---

- ▶ Memoria virtual basado en tablas de páginas:
  - ▶ Problema: consumo de memoria para las tablas
    - ▶ Ej.: páginas 4KB, dir. lógica 32 bits y 4 bytes por entrada:  
 $2^{20} * 4 = 4\text{MB/proceso}$
  - ▶ Solución: **tablas multinivel**
  
- ▶ **Tabla multinivel**
  - ▶ Esquema de traducción en dos niveles:
    - ▶ En memoria la tabla de primer nivel
    - ▶ Solo en memoria las tablas de segundo nivel que se necesiten
  - ▶ Tablas de cada nivel mucho más compactas:  $2^{10} * 4 = 4\text{KB/tabla}$

# Tablas multinivel

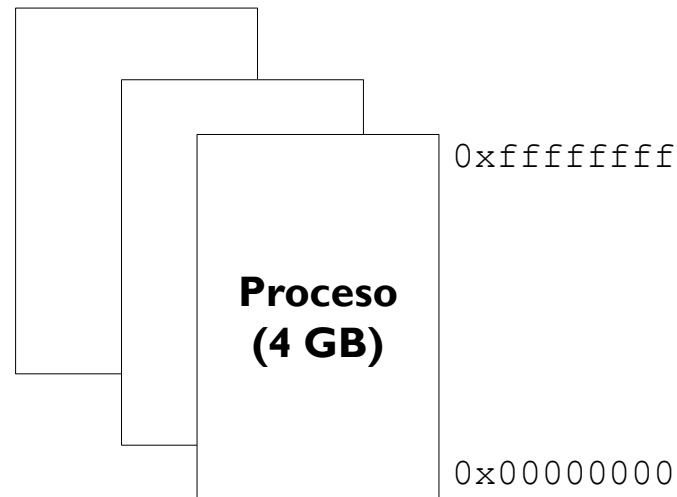




# Espacio de memoria: proceso + kernel

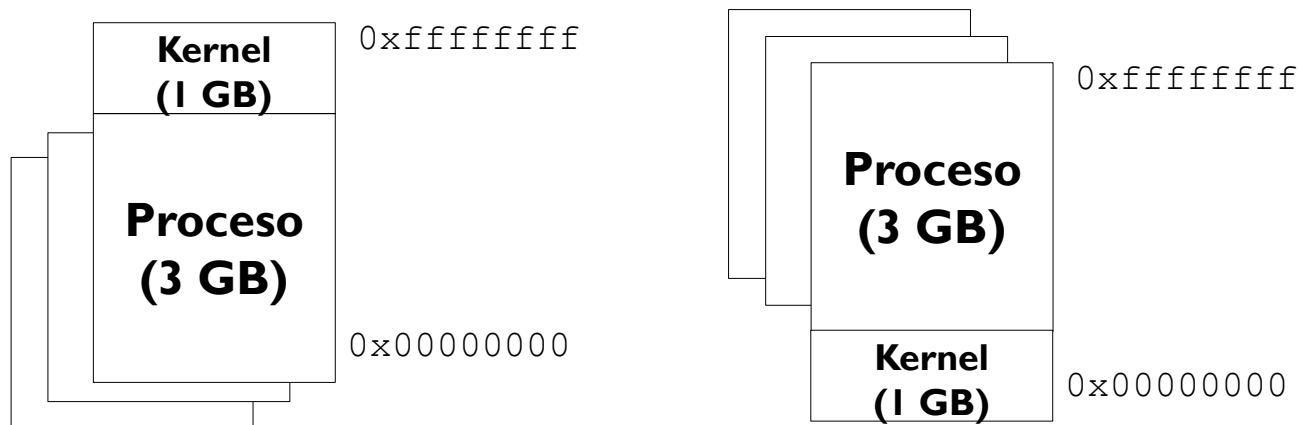
---

- ▶ Cada proceso ve un espacio de direcciones lineal y plano
  - ▶ Cada proceso podría acceder a todo el espacio de direcciones posible



# Espacio de memoria: proceso + kernel

- ▶ El espacio usado por el kernel es compartido por todos los procesos
  - ▶ No cambia en los cambios de contexto
- ▶ El espacio del kernel está protegido (lectura, escritura y ejecución)
  - ▶ La mayoría de llamadas al sistema más rápidas (evita cambio de modo  $u \rightarrow k$  y  $k \rightarrow u$ )



# Contenidos

---

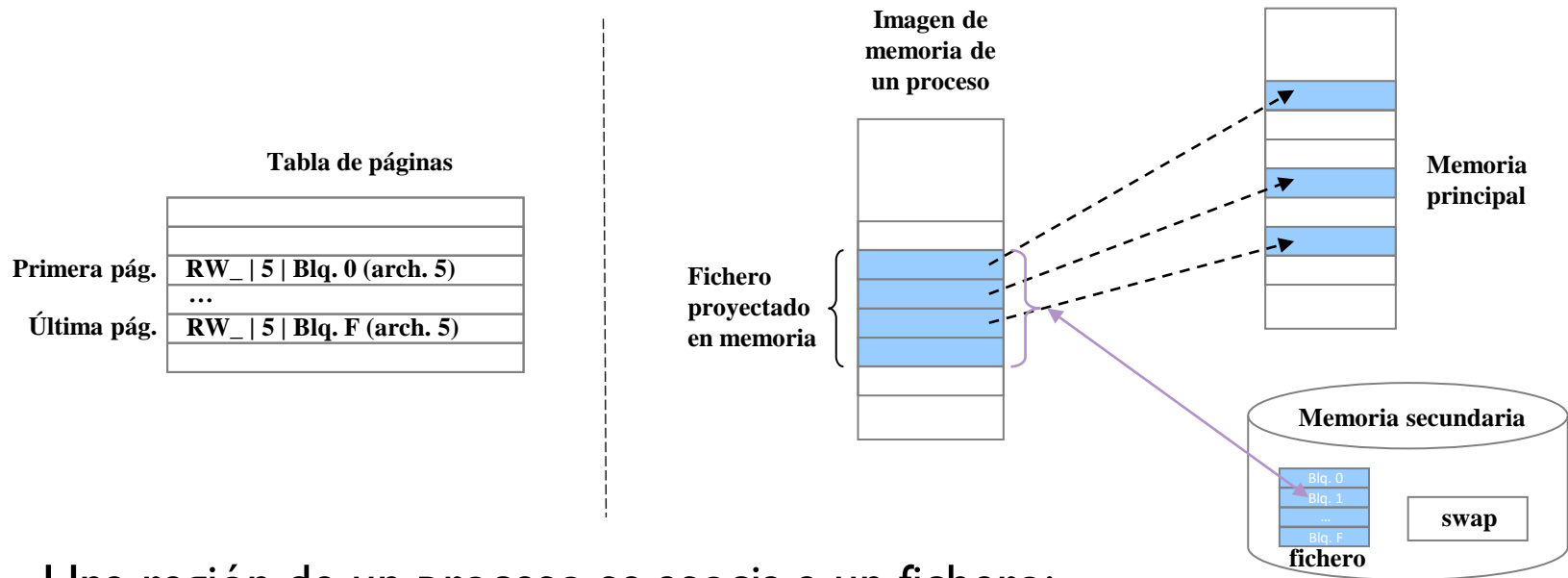
1. **Introducción:**
  1. Modelo abstracto y definiciones básicas.
  2. Mapa de memoria de un proceso: regiones de memoria.
2. **Funciones del gestor de memoria.**
  1. Particionamiento de memoria.
  2. Algoritmos de gestión de memoria.
3. **Memoria virtual.**
4. **Servicios del gestor de memoria.**

# Servicios de gestión de memoria

---

- ▶ Gestor de memoria realiza funciones internas.
- ▶ Ofrece pocos servicios directos a aplicaciones.
  
- ▶ Principales servicios POSIX:
  - ▶ Gestión de la proyección de archivos
    - ▶ `mmap` y `munmap`
  - ▶ Gestión de bibliotecas dinámicas
    - ▶ `dlopen`, `dlsym` y `dlclose`

# Ficheros proyectados en memoria



- ▶ Una región de un proceso se asocia a un fichero:
  - ▶ Habrá páginas del fichero en memoria principal (bajo demanda)
  - ▶ Acceso al contenido del fichero con instrucciones de memoria (en lugar de read/write)
- ▶ Simplifica y mejora el rendimiento:
  - ▶ Facilita programación (uso de memoria vs fichero), menos llamadas al sistema (carga bajo demanda), evita copias en caché del sistema de ficheros (uso caché páginas).
  - ▶ Usado en bibliotecas dinámicas, carga ejecutable (código proyectado), etc.

# Ficheros proyectados en memoria

## mmap

Servicio	<pre>void *mmap( void *addr, size_t len,             int prot, int flags,             int fildes, off_t off );</pre>
Argumentos	<ul style="list-style-type: none"><li>▪ <code>addr</code> dirección donde proyectar. Si <code>NULL</code> el SO elige una.</li><li>▪ <code>len</code> especifica el número de bytes a proyectar.</li><li>▪ <code>prot</code> el tipo de acceso (lectura, escritura o ejecución).</li><li>▪ <code>flags</code> especifica información sobre el manejo de los datos proyectados (compartidos, privado, etc.).</li><li>▪ <code>fildes</code> representa el descriptor de fichero del fichero o descriptor del objeto de memoria a proyectar.</li><li>▪ <code>off</code> desplazamiento dentro del fichero a partir del cual se realiza la proyección.</li></ul>
Devuelve	Devuelve la dirección de memoria donde se ha proyectado el fichero.
Descripción	Establece una proyección entre el espacio de direcciones de un proceso y un descriptor de fichero u objeto de memoria compartida.

# Ficheros proyectados en memoria

## mmap

Servi

- **PROT\_READ**: Se puede leer.
- **PROT\_WRITE**: Se puede escribir.
- **PROT\_EXEC**: Se puede ejecutar.
- **PROT\_NONE**: No se puede acceder a los datos.

Argumentos

- **len** especifica el número de bytes a proyectar.
- **prot** el tipo de acceso (lectura, escritura o ejecución).
- **flags** especifica información sobre el manejo de los datos proyectados (compartidos, privado, etc.).

- **MAP\_SHARED**: La región es compartida. Fichero puede modificarse. Los procesos hijos comparten la región.
- **MAP\_PRIVATE**: La región es privada. El fichero no se modifica. Los procesos hijos obtienen duplicados no compartidos.
- **MAP\_FIXED**: El fichero debe proyectarse en la dirección especificada por la llamada.

# Ficheros proyectados en memoria

## **munmap**

---

Servicio	<code>void munmap(void *addr, size_t len);</code>
Argumentos	<ul style="list-style-type: none"><li>▫ <code>addr</code> dirección donde está proyectado.</li><li>▫ <code>len</code> especifica el número de bytes proyectados.</li></ul>
Devuelve	Nada.
Descripción	Desproyecta parte del espacio de direcciones de un proceso comenzando en la dirección <code>addr</code> .





# Ficheros proyectados en memoria

---

- ▶ Cuántas veces aparece carácter en fichero proyectando en memoria.

```
/* 1) Abrir el fichero */
fd=open(argv[2], O_RDONLY); /* Abre fichero */
fstat(fd, &fs); /* Averigua long. fichero */

/* 2) Proyectar el fichero */
org=mmap((caddr_t)0, fs.st_size, PROT_READ, MAP_SHARED, fd, 0);
close(fd); /* Se cierra el fichero */

/* 3) Bucle de acceso */
p=org;
for (i=0; i<fs.st_size; i++)
    if (*p++==caracter) contador++;

/* 4) Eliminar la proyección */
munmap(org, fs.st_size);
printf("%d\n", contador);
```

# Ejemplo: copia de un fichero (1/2)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    int i, fd1, fd2;
    struct stat dstat;
    char * vec1, *vec2, *p, *q;

    fd1 = open("f1", O_RDONLY);
    fd2 = open("f2", O_CREAT|O_TRUNC|O_RDWR,0640);
    fstat(fd1,&dstat);
    ftruncate(fd2, dstat.st_size);

    vec1=mmap(0, bstat.st_size,
        PROT_READ, MAP_SHARED, fd1,0);
    vec2=mmap(0, bstat.st_size,
        PROT_READ, MAP_SHARED, fd2,0);

    close(fd1); close(fd2);

    p=vec1; q=vec2;
    for (i=0;i<dstat.st_size;i++) {
        *q++ = *p++;
    }

    munmap(fd1, bstat.st_size);
    munmap(fd2, bstat.st_size);

    return 0;
}
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    int i, fd1, fd2;
    struct stat dstat;
    char * vec1, *vec2, *p, *q;

    fd1 = open("f1", O_RDONLY);
    fd2 = open("f2", O_CREAT|O_TRUNC|O_RDWR,0640);
    fstat(fd1,&dstat);
    ftruncate(fd2, dstat.st_size);
```

# Ejemplo: copia de un fichero (2/2)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    int i, fd1, fd2;
    struct stat dstat;
    char * vec1, *vec2, *p, *q;

    fd1 = open("f1", O_RDONLY);
    fd2 = open("f2", O_CREAT|O_TRUNC|O_RDWR, 0640);
    fstat(fd1, &dstat);
    ftruncate(fd2, dstat.st_size);

    vec1=mmap(0, bstat.st_size,
        PROT_READ, MAP_SHARED, fd1, 0);
    vec2=mmap(0, bstat.st_size,
        PROT_READ, MAP_SHARED, fd2, 0);

    close(fd1); close(fd2);

    p=vec1; q=vec2;
    for (i=0; i<dstat.st_size; i++) {
        *q++ = *p++;
    }

    munmap(fd1, bstat.st_size);
    munmap(fd2, bstat.st_size);

    return 0;
}
```

```
vec1=mmap(0, bstat.st_size,
           PROT_READ, MAP_SHARED, fd1, 0);
vec2=mmap(0, bstat.st_size,
           PROT_READ, MAP_SHARED, fd2, 0);

close(fd1); close(fd2);

p=vec1; q=vec2;
for (i=0; i<dstat.st_size; i++) {
    *q++ = *p++;
}

munmap(fd1, bstat.st_size);
munmap(fd2, bstat.st_size);

return 0;
}
```

# Ejemplo: contar el # de espacios (1/2)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    int fd;
    struct stat dstat;
    int i, n;
    char c,
    char * vec;

    fd = open("datos.txt",O_RDONLY);
    fstat(fd, &dstat);
    vec = mmap(NULL, dstat.st_size,
        PROT_READ, MAP_SHARED, fd, 0);
    close(fd);
    c = vec;
    for (i=0; i<dstat.st_size; i++) {
        if (*c==' ') {
            n++;
        }
        c++;
    }
    munmap(vec, dstat.st_size);
    printf("n=%d,\n",n);
    return 0;
}
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
```

```
int main() {
    int fd;
    struct stat dstat;
    int i, n;
    char c,
    char * vec;
```

# Ejemplo: contar el # de espacios (2/2)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    int fd;
    struct stat dstat;
    int i, n;
    char c,
    char * vec;

    fd = open("datos.txt", O_RDONLY);
    fstat(fd, &dstat);
    vec = mmap(NULL, dstat.st_size,
               PROT_READ, MAP_SHARED, fd, 0);
    close(fd);
    c = vec;
    for (i=0; i<dstat.st_size; i++) {
        if (*c==' ') {
            n++;
        }
        c++;
    }
    munmap(vec, dstat.st_size);
    printf("n=%d, \n", n);
    return 0;
}
```

```
fd = open("datos.txt", O_RDONLY);
    fstat(fd, &dstat);

    vec = mmap(NULL, dstat.st_size,
                PROT_READ, MAP_SHARED, fd, 0);

    close(fd);

    c = vec;

    for (i=0; i<dstat.st_size; i++) {
        if (*c==' ') { n++; }
        c++;
    }

    munmap(vec, dstat.st_size);

    printf("n=%d, \n", n);

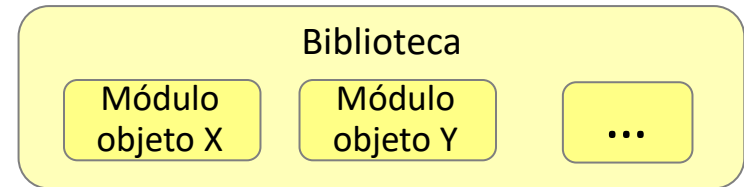
    return 0;
}
```

# Biblioteca estática vs dinámica

---

## ▶ Biblioteca

- ▶ Colección de módulos objetos relacionados



## ▶ Biblioteca **estática**

- ▶ Montaje en tiempo de compilación.
- ▶ Ejecutable incluye librería (autocontenido)

## ▶ Biblioteca **dinámica**

- ▶ Carga y montaje en tiempo de ejecución
- ▶ Normalmente enlace **implícito**
  - ▶ Carga y montaje bajo demanda transparente (generación similar a estáticas)
- ▶ Es posible enlace **explícito**
  - ▶ [I] Hay que añadir código para cargar y enlazar los símbolos
  - ▶ [V] En tiempo de ejecución permite indicar biblioteca a usar (entre varias)

# Ejemplo: biblioteca estática

2

```
#include <stdio.h>

extern void hola ( void );

int main()
{
    hola ();
    return 0;
}
```

```
gcc -Wall -g -o main main.c -lhola -L./main
```

1

```
#include <stdio.h>
#include <stdlib.h>

void hola ( void )
{
    printf("hola") ;
}
```

```
gcc -Wall -g -o libhola.o -c libhola.c
ar rcs libhola.a libhola.o
```

# Ejemplo: biblioteca dinámica (carga implícita)

2

```
#include <stdio.h>

extern void hola ( void );

int main()
{
    hola ();
    return 0;
}
```

```
gcc -Wall -g -o main main.c -lhola -L./
env LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./main
```

1

```
#include <stdio.h>
#include <stdlib.h>

void hola ( void )
{
    printf("hola") ;
}
```

```
gcc -Wall -g -fPIC -o libhola.o -c libhola.c
gcc -shared -Wl,-soname,libhola.so \
    -o libhola.so.1.0 libhola.o
ln -s libhola.so.1.0 libhola.so
```



# Ejemplo: biblioteca dinámica (carga **explícita**)

2

```
#include <stdio.h>
#include <dlfcn.h>

void (*hola) (void);

int main()
{
    void * dbd ;

    dbd = dlopen("libhola.so",
                RTLD_LAZY);
    hola = dlsym(dbd, "hola");
    hola ();
    dlclose(dbd);
    return 0;
}
```

```
gcc -Wall -g -o main main.c -ldl
env LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./main
```

1

```
#include <stdio.h>
#include <stdlib.h>

void hola ( void )
{
    printf("hola") ;
}
```

```
gcc -Wall -g -fPIC -c libhola.c
gcc -shared -o libhola.so libhola.o
```

# Bibliotecas dinámicas

## **dlopen**

---

Servicio	<pre>#include &lt;dlfcn.h&gt;  void * <b>dlopen</b> ( const char * bib, int flags );</pre>
Argumentos	<ul style="list-style-type: none"><li>▫ <code>bib</code> es el nombre del fichero con la biblioteca dinámica.</li><li>▫ <code>flags</code> opciones de trabajo:<ul style="list-style-type: none"><li>▫ Una forma de resolución de referencias:<ul style="list-style-type: none"><li>▫ <code>RTLD_LAZY</code>: resolución diferida de referencias.</li><li>▫ <code>RTLD_NOW</code>: resolución inmediata de referencias.</li></ul></li><li>▫ Cero o más opciones usando OR:<ul style="list-style-type: none"><li>▫ <code>RTLD_GLOBAL</code>, <code>RTLD_LOCAL</code>, <code>RTLD_NODELETE</code>, <code>RTLD_NOLOAD</code>, <code>RTLD_DEEPBIND</code></li></ul></li></ul></li></ul>
Devuelve	Devuelve un descriptor o NULL en caso de error.
Descripción	Carga una biblioteca dinámica y la enlaza con el proceso actual.

# Bibliotecas dinámicas

## **dl**sym

---

<b>Servicio</b>	<pre>#include &lt;dlfcn.h&gt;  void * <b>dl</b>sym ( void *descr, char *simbolo );</pre>
<b>Argumentos</b>	<ul style="list-style-type: none"><li>▫ <code>descr</code> es el descriptor previamente devuelto por <code>dlopen</code>.</li><li>▫ <code>simbolo</code> es la cadena de caracteres con el nombre del elemento (variable, función, etc.) del que se desea su dirección.</li></ul>
<b>Devuelve</b>	Devuelve un puntero a un símbolo de la biblioteca dinámica.
<b>Descripción</b>	Permite obtener una referencia a un símbolo de la biblioteca cargada.

# Bibliotecas dinámicas

## **dlclose**

---

Servicio	<pre>#include &lt;dlfcn.h&gt;  void <b>dlclose</b> ( void * descriptor );</pre>
Argumentos	<ul style="list-style-type: none"><li>▫ <code>descriptor</code> es el descriptor devuelto previamente por <code>dlopen</code>.</li></ul>
Devuelve	No devuelve nada.
Descripción	Descarga una biblioteca dinámica del proceso que llama a <code>dlclose</code> .

# Lección 4

## Gestión de memoria

Sistemas Operativos  
Ingeniería Informática