

Grupo ARCOS

**uc3m** | Universidad **Carlos III** de Madrid

# Tema 5: Jerarquía de memoria (III)

## **Estructura de Computadores**

Grado en Ingeniería Informática  
Grado en Matemática aplicada y Computación  
Doble Grado en Ingeniería Informática y Administración de Empresas



# Contenidos

1. Tipos de memoria
2. Jerarquía de memoria
3. Memoria principal
4. Memoria caché
5. Memoria virtual

# Caché vs memoria virtual

adelanto

## Caché

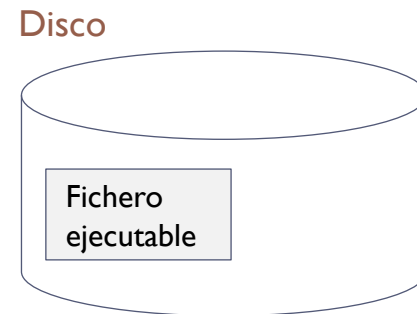
- ▶ Objetivo: acelerar el acceso (+ rápido)
- ▶ Transferencia por bloques o líneas.
- ▶ Bloques: 32-64 bytes.
- ▶ Traducción: algoritmo de correspondencia.
- ▶ Escritura inmediata o diferida.

## Memoria virtual

- ▶ Objetivo: incrementar el espacio direccionable
- ▶ Transferencia por páginas.
- ▶ Páginas: 4-8 KiB
- ▶ Traducción: totalmente asociativa.
- ▶ Escritura diferida.

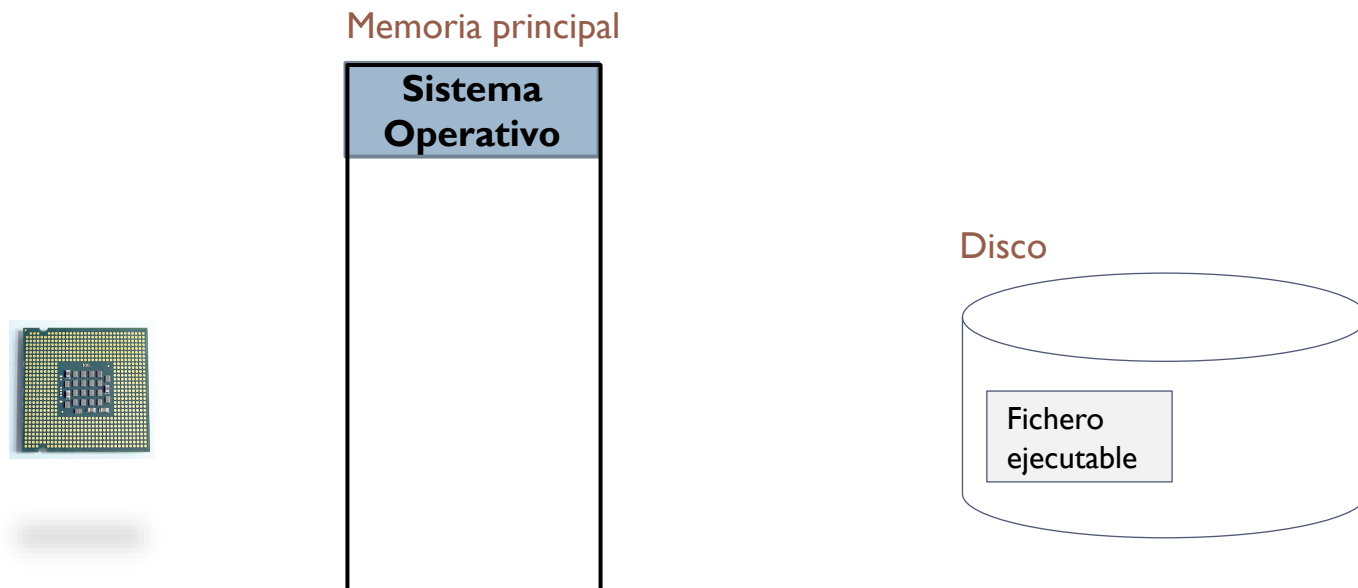
# Programa, proceso e imagen

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



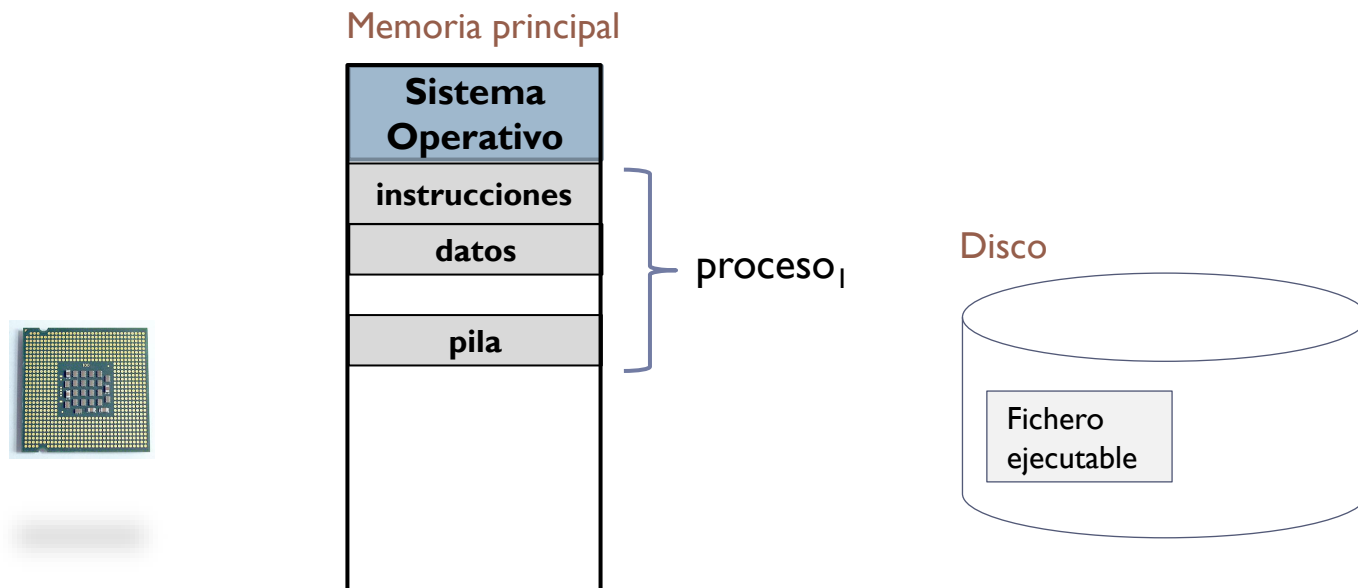
# Programa, proceso e imagen

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



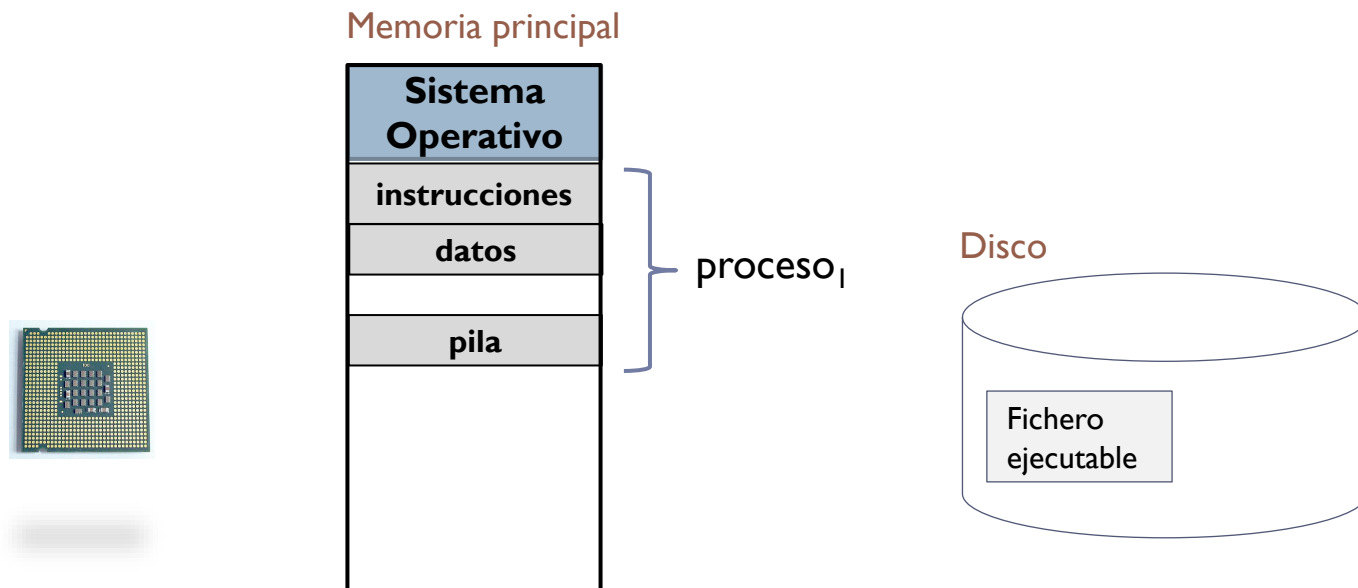
# Programa, proceso e imagen

- **Proceso:** programa en ejecución.



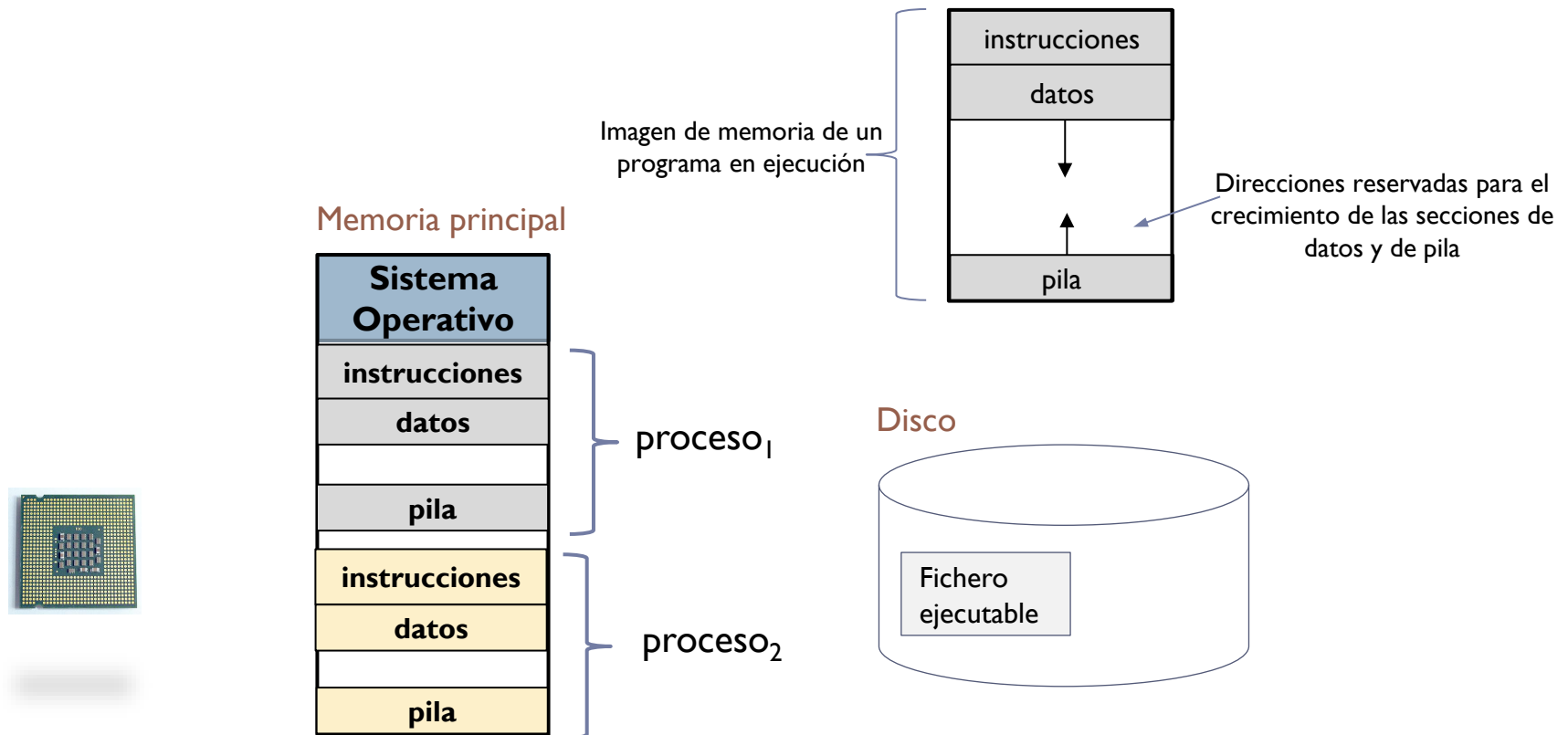
# Programa, proceso e imagen

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



# Imagen de un proceso

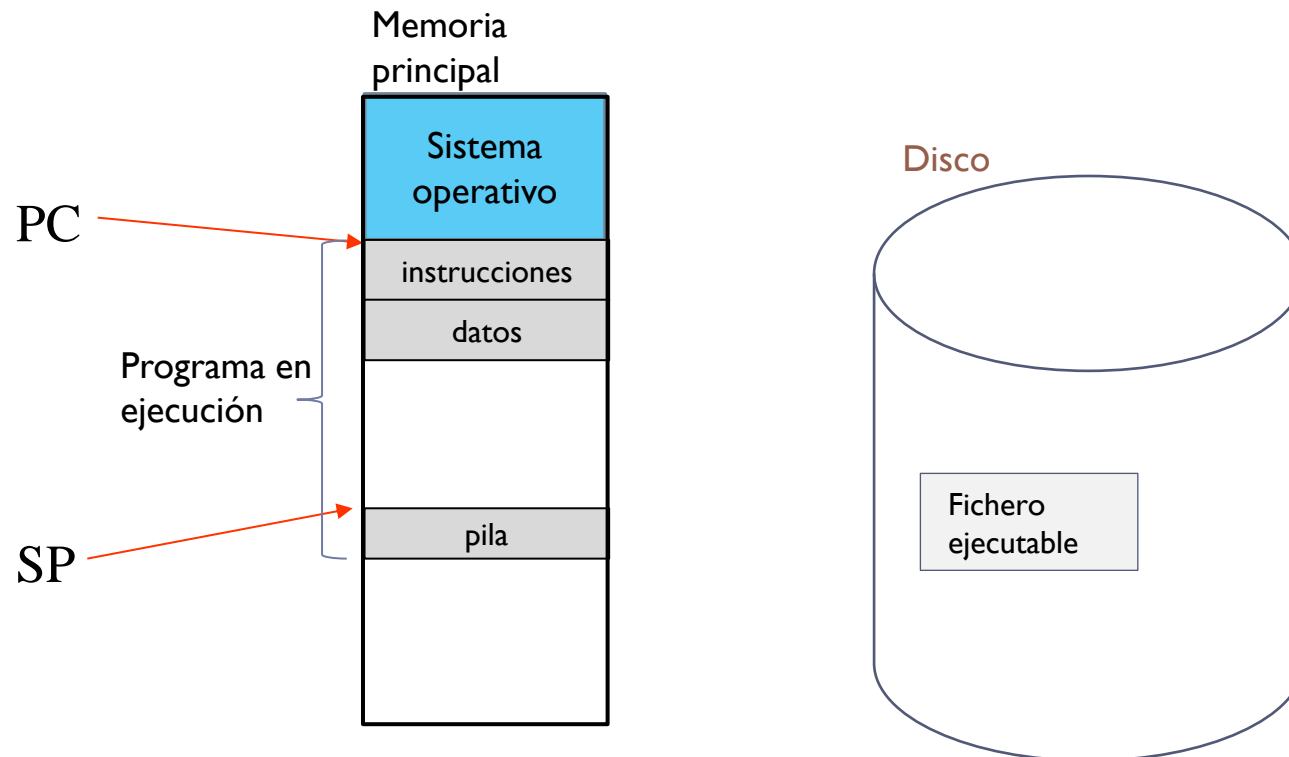
- **Imagen de memoria:** conjunto de direcciones de memoria asignadas al programa que se está ejecutando (y contenido)





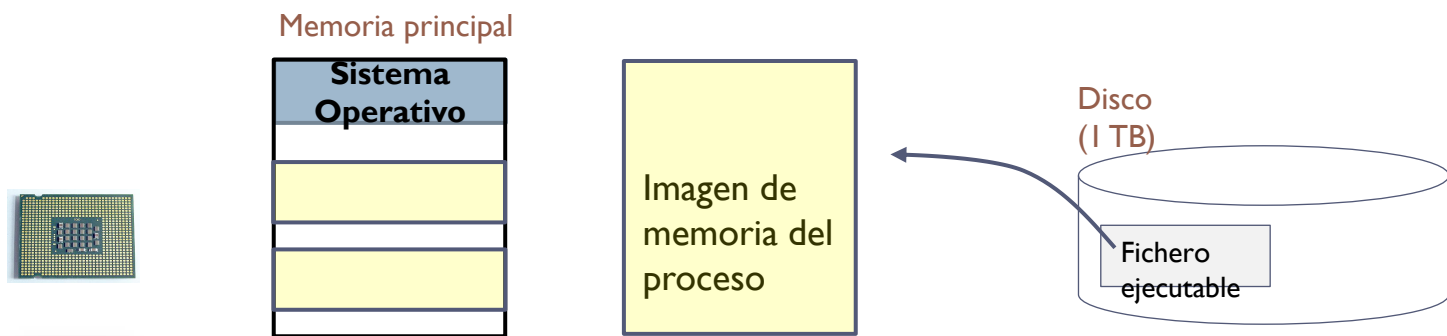
# Sistemas **sin** memoria virtual

- ▶ En los sistemas sin memoria virtual, el programa se carga completamente en memoria para su ejecución.



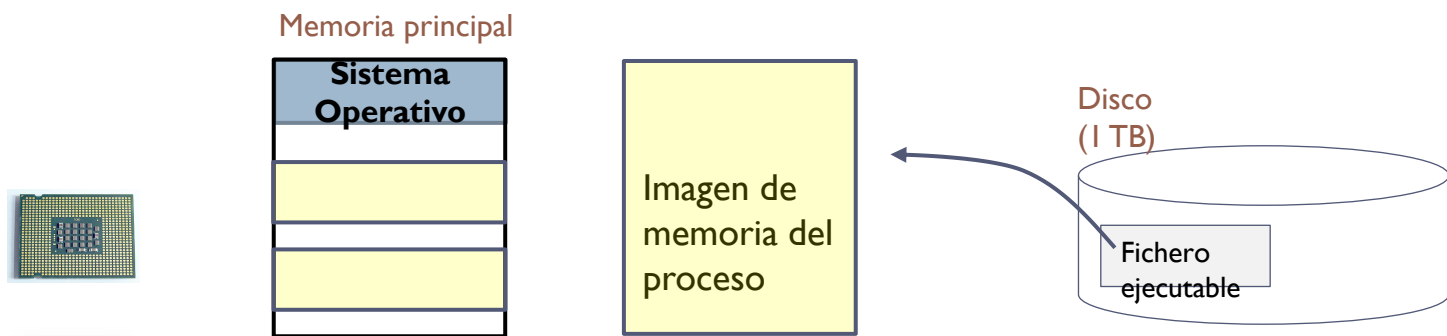
# Sistemas **sin** memoria virtual

- ▶ En los sistemas sin memoria virtual, el programa se carga completamente en memoria para su ejecución.
- ▶ Principales problemas (1/2):
  - ▶ **Reubicación:** imagen ha de poder cargarse en cualquier lugar asignado.
  - ▶ **Protección:** impedir acceso fuera del espacio asignado.



# Sistemas **sin** memoria virtual

- ▶ En los sistemas sin memoria virtual, el programa se carga completamente en memoria para su ejecución.
- ▶ Principales problemas (1/2):
  - ▶ **Reubicación:** imagen ha de poder cargarse en cualquier lugar asignado.
  - ▶ **Protección:** impedir acceso fuera del espacio asignado.




# Fichero ejecutable hipotético

```
int v[1000]; // global
int i;
for (i=0; i < 1000; i++)
    v[i] = 0;
```

# Fichero ejecutable hipotético

```
int v[1000]; // global
int i;
for (i=0; i < 1000; i++)
    v[i] = 0;
```



```
.data
    v: .space 4000
.text
main:  li    t0, 0
       li    t1, 0
       li    t2, 1000
bucle: bge    t0, t2, fin
       sw     x0, v(t1)
       addi   t0, t0, 1
       addi   t1, t1, 4
       j      bucle
fin:   ...
```

# Fichero ejecutable hipotético

```
int v[1000]; // global
int i;
for (i=0; i < 1000; i++)
    v[i] = 0;
```

ensamblador

```
.data
    v: .space 4000
.text
main:  li    t0, 0
      li    t1, 0
      li    t2, 1000
bucle: bge    t0, t2, fin
      sw     x0, v(t1)
      addi   t0, t0, 1
      addi   t1, t1, 4
      j      bucle
fin:   ...
```

ejecutable

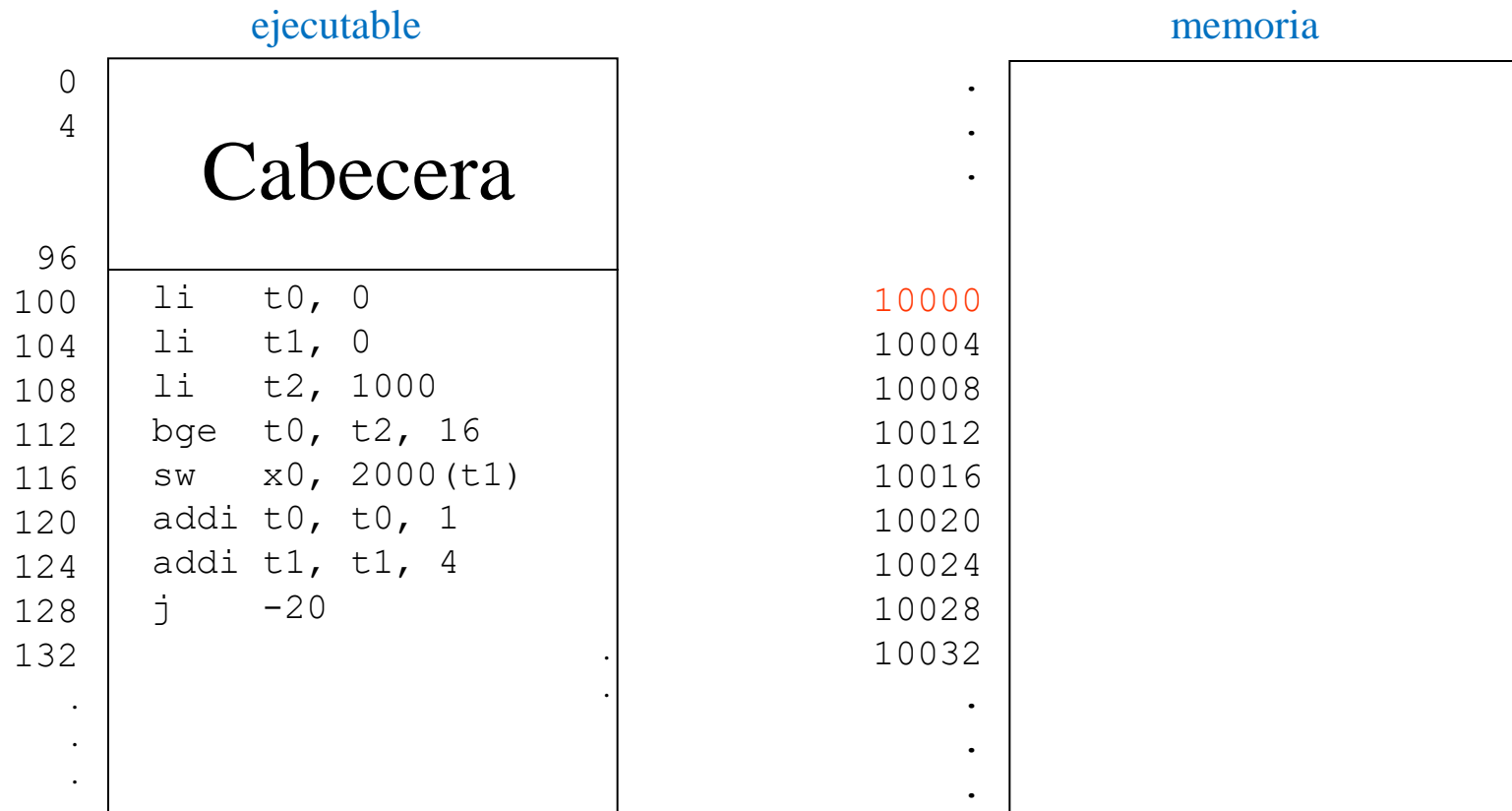
0	
4	
	<b>Cabecera</b>
96	
100	li    t0, 0
104	li    t1, 0
108	li    t2, 1000
112	bge    t0, t2, 16
116	sw     x0, <b>2000</b> (t1)
120	addi   t0, t0, 1
124	addi   t1, t1, 4
128	j      -20
132	
.	.
.	.
.	.

Se asigna a **v** la dirección **2000**

Se asume que el programa empieza en la 0

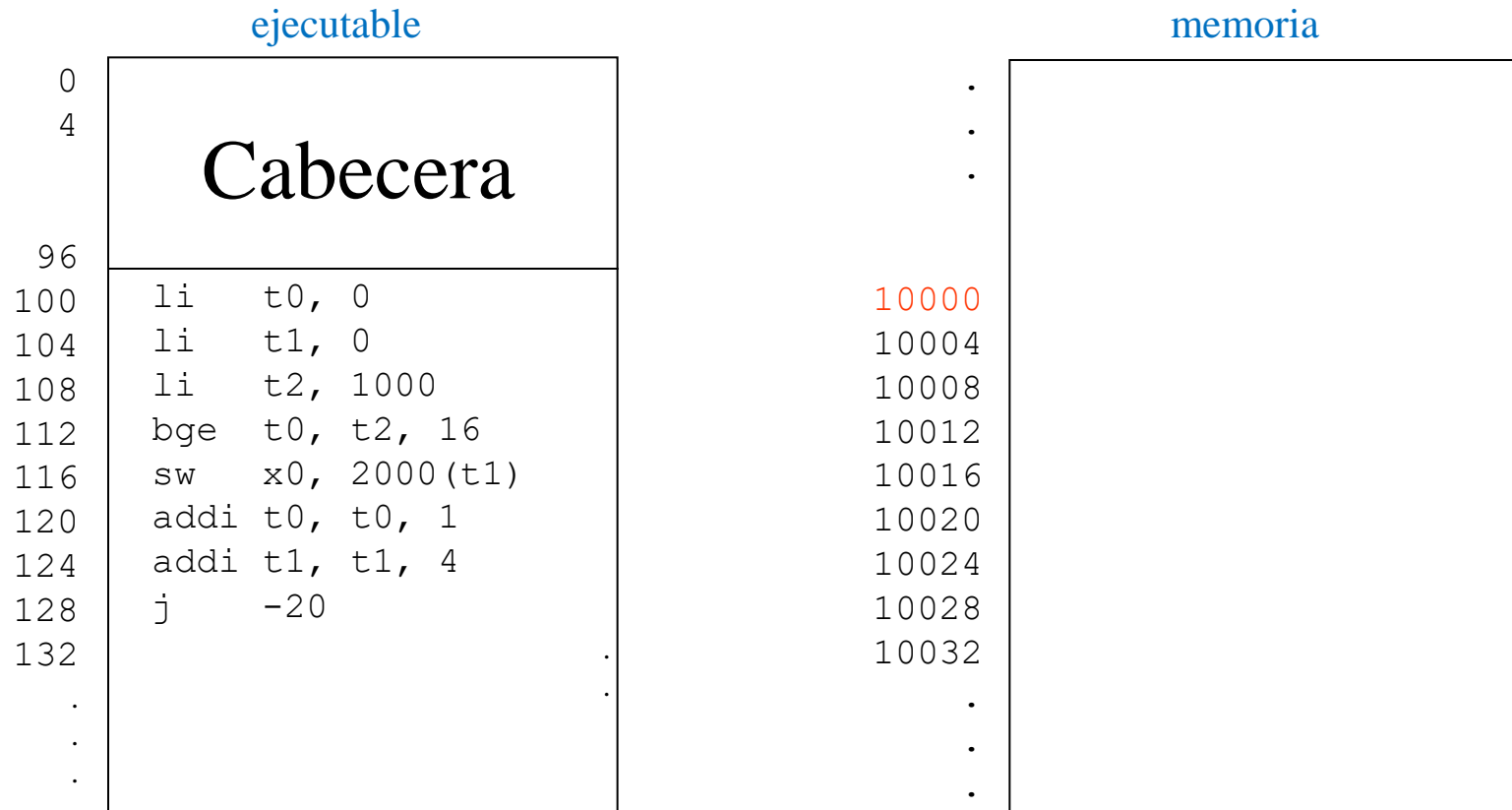
# Carga del programa en memoria

- El sistema operativo reserva un hueco libre en memoria para **toda** la imagen del proceso



# Carga del programa en memoria

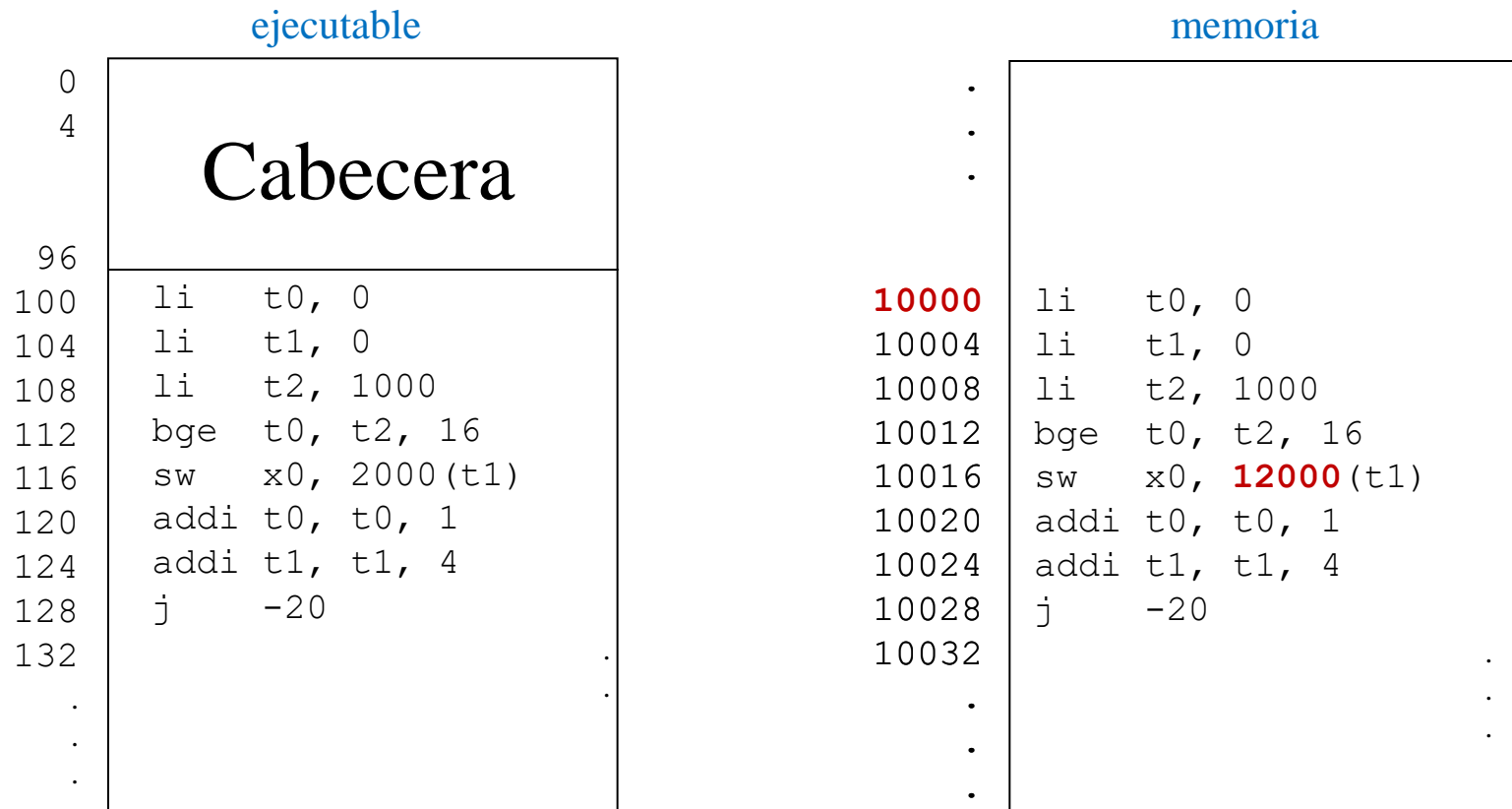
- **Direcciones lógicas:** en el fichero ejecutable se considera como dirección de inicio la 0
- **Direcciones físicas:** en memoria, la dirección inicial es la 10000





# Carga del programa en memoria

- ▶ Hay que realizar una **traducción de direcciones**
  - ▶ De direcciones lógicas a físicas
- ▶ El array v está en:
  - ▶ Dirección lógica 2000 y dirección física 2000 + 10000



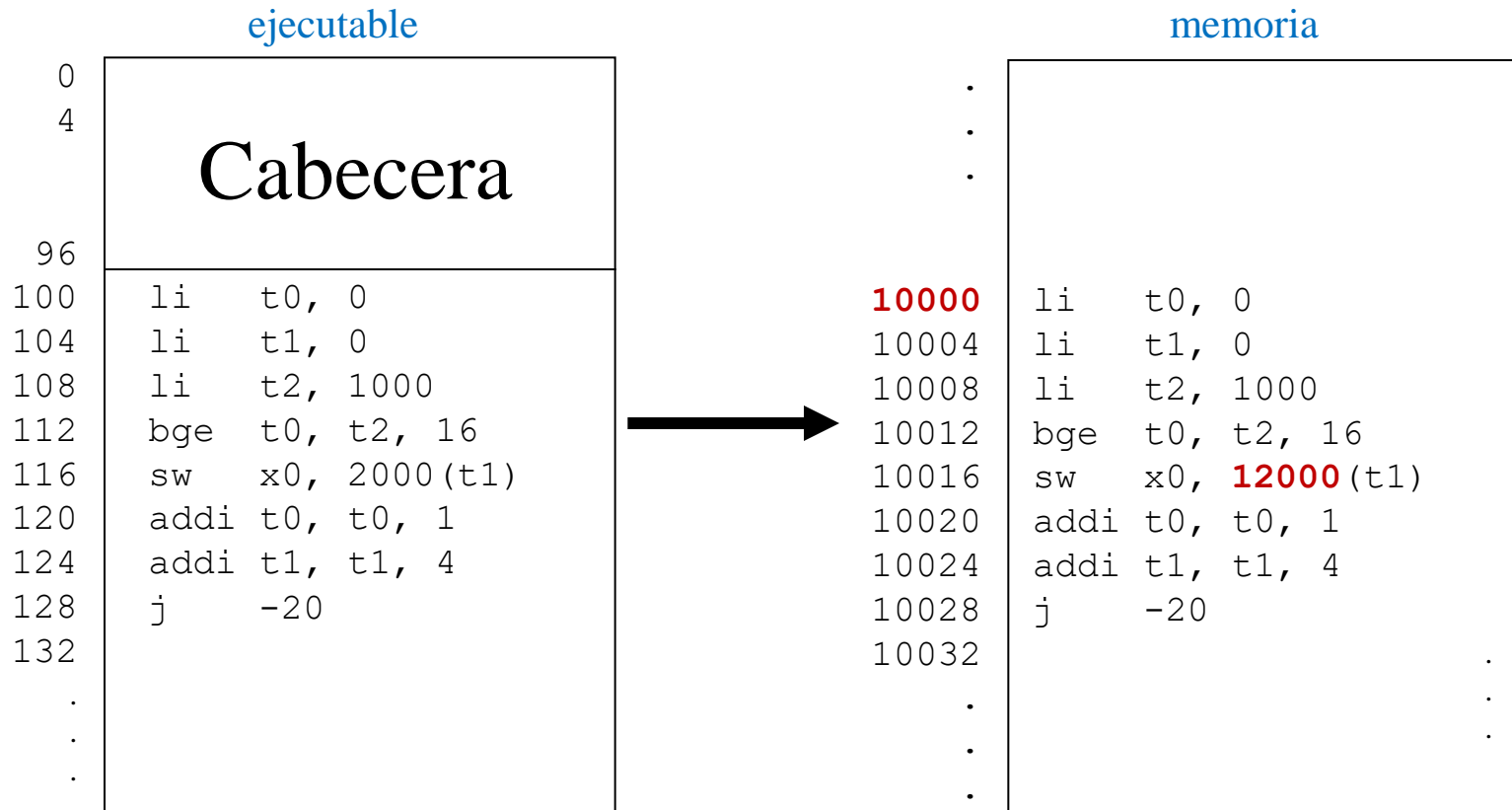
# Carga del programa en memoria

- ▶ Hay que realizar una **traducción de direcciones**
  - ▶ De direcciones lógicas a físicas
- ▶ **A este proceso se le denomina reubicación**
  - ▶ Reubicación software y reubicación hardware

ejecutable		memoria	
0	Cabecera	.	
4		.	
		.	
96			
100		10000	li t0, 0
104	li t1, 0	10004	li t1, 0
108	li t2, 1000	10008	li t2, 1000
112	bge t0, t2, 16	10012	bge t0, t2, 16
116	sw x0, 2000(t1)	10016	sw x0, 12000(t1)
120	addi t0, t0, 1	10020	addi t0, t0, 1
124	addi t1, t1, 4	10024	addi t1, t1, 4
128	j -20	10028	j -20
132	.	10032	.
.	.	.	.
.	.	.	.
.	.	.	.

# Reubicación software

- Se realiza la traducción en el momento de la carga



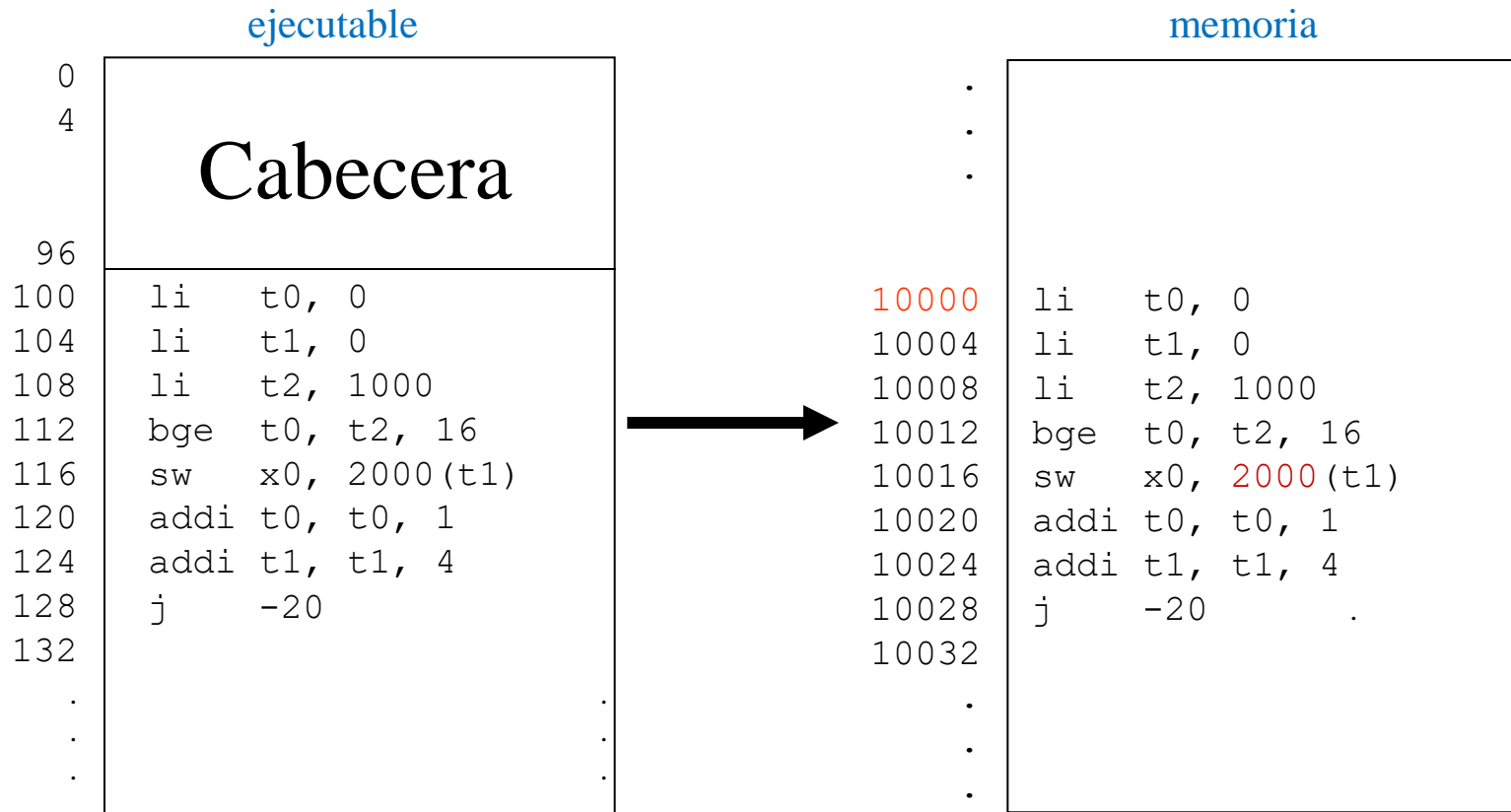
# Reubicación software

- ▶ ¿Qué ocurre con estas instrucciones cargadas en las posiciones 10012 y 10028?
  - ▶ Direcciones absolutas vs direcciones relativas

ejecutable		memoria	
0	Cabecera	.	
4		.	
		.	
96			
100	li t0, 0	10000	li t0, 0
104	li t1, 0	10004	li t1, 0
108	li t2, 1000	10008	li t2, 1000
112	bge t0, t2, 16	10012	bge t0, t2, <b>16</b>
116	sw x0, 2000(t1)	10016	sw x0, 2000(t1)
120	addi t0, t0, 1	10020	addi t0, t0, 1
124	addi t1, t1, 4	10024	addi t1, t1, 4
128	j -20	10028	j <b>-20</b>
132		10032	
.	.	.	
.	.	.	
.	.	.	

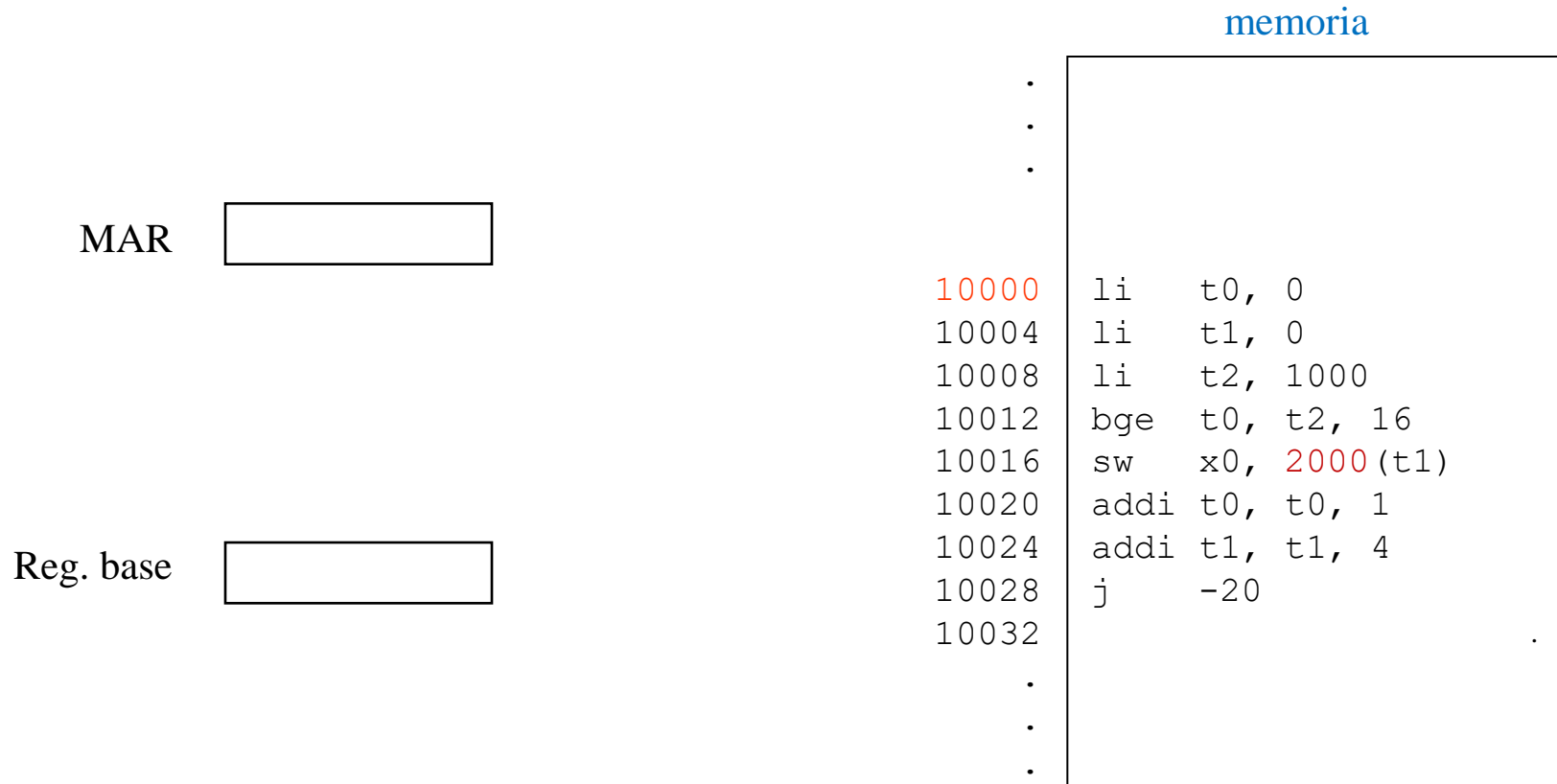
# Reubicación hardware

- ▶ Se realiza la traducción durante la ejecución
- ▶ Necesita un hardware especial.



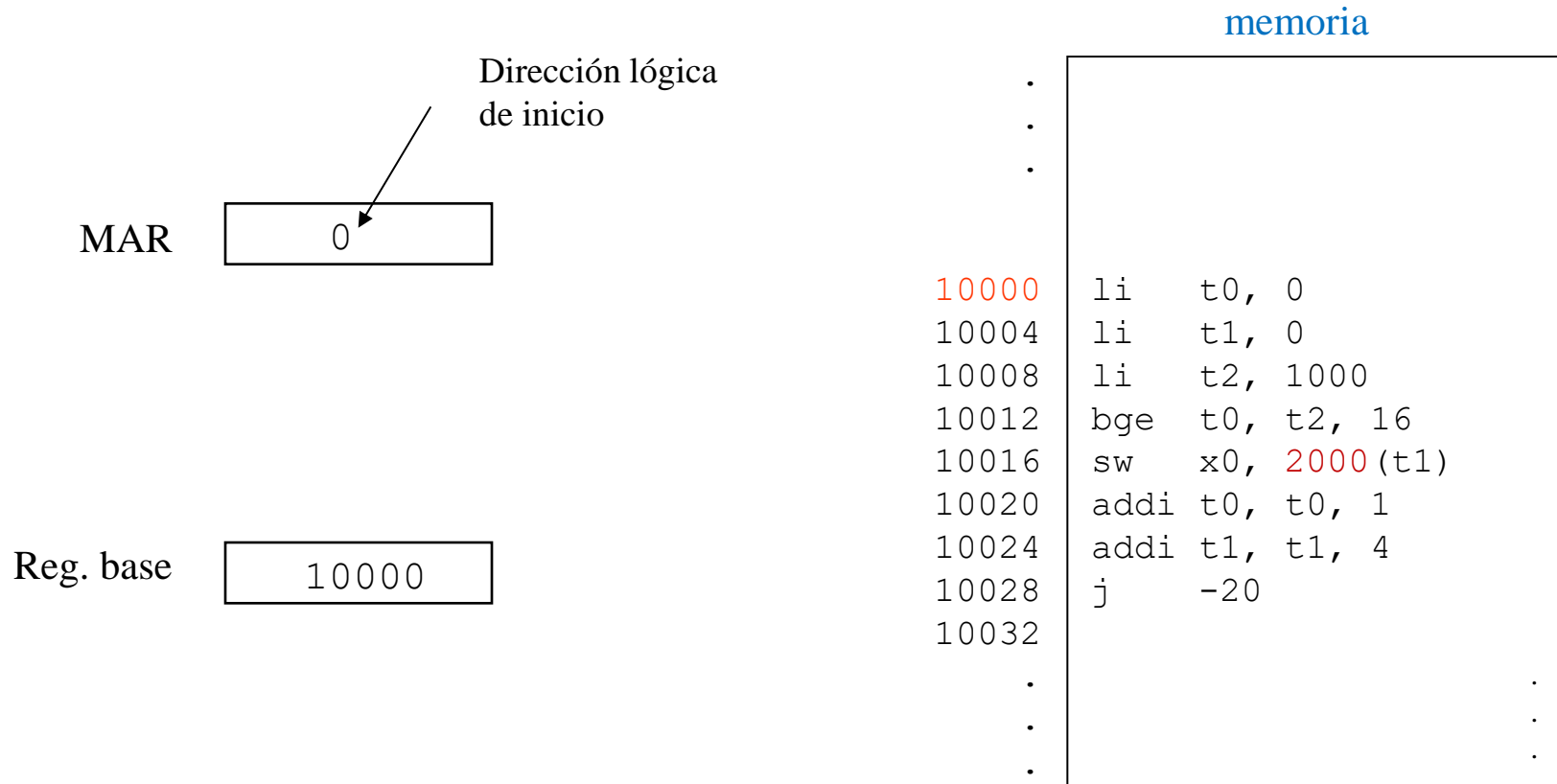
# Ejemplo de soporte hardware

- **Registro base:** dirección de inicio del programa en memoria



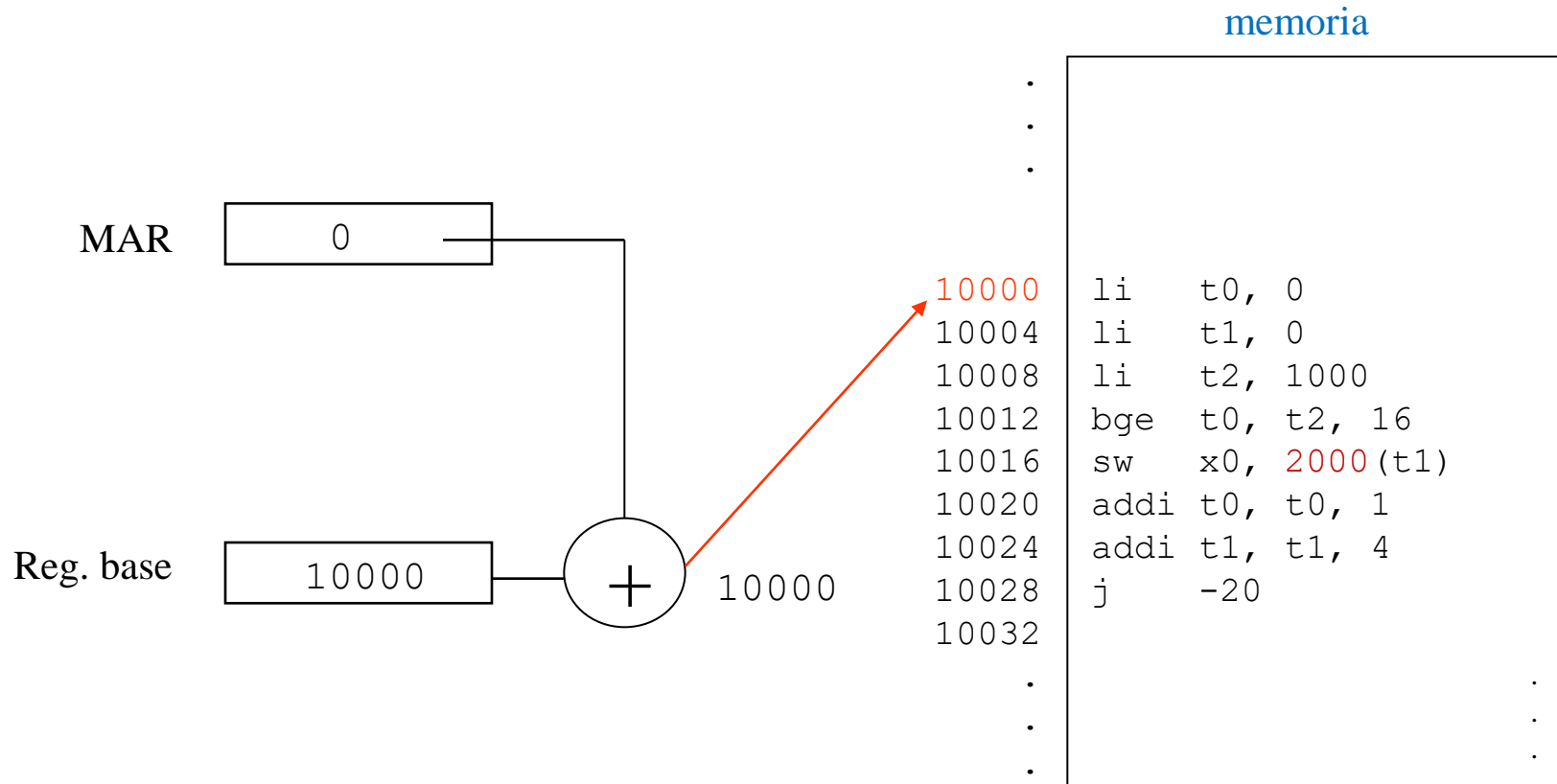
# Ejemplo de soporte hardware

- **Registro base:** dirección de inicio del programa en memoria



# Ejemplo de soporte hardware

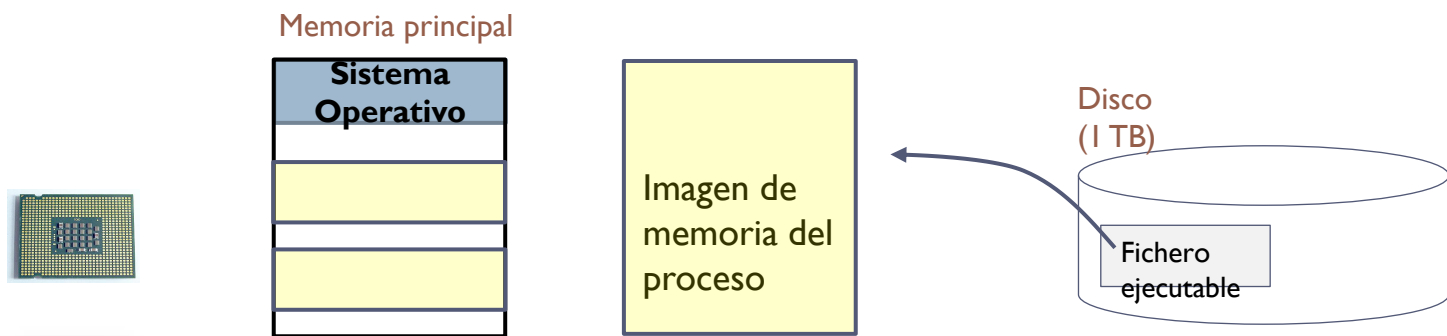
- **Registro base:** dirección de inicio del programa en memoria





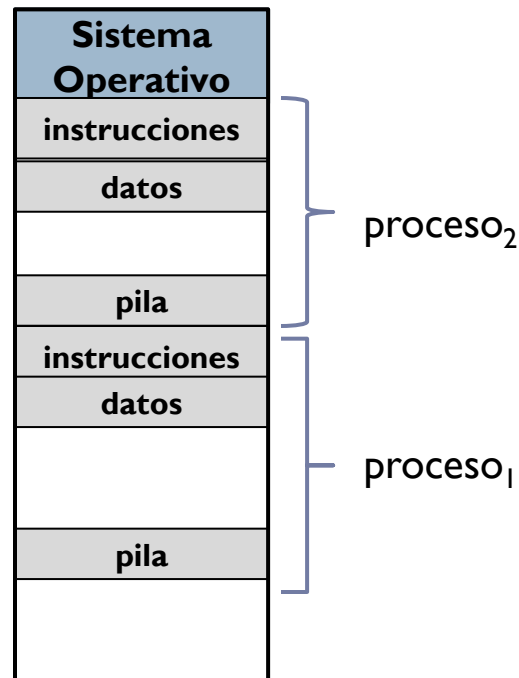
# Sistemas **sin** memoria virtual

- ▶ En los sistemas sin memoria virtual, el programa se carga completamente en memoria para su ejecución.
- ▶ Principales problemas (1/2):
  - ▶ **Reubicación:** imagen ha de poder cargarse en cualquier lugar asignado.
  - ▶ **Protección:** impedir acceso fuera del espacio asignado.

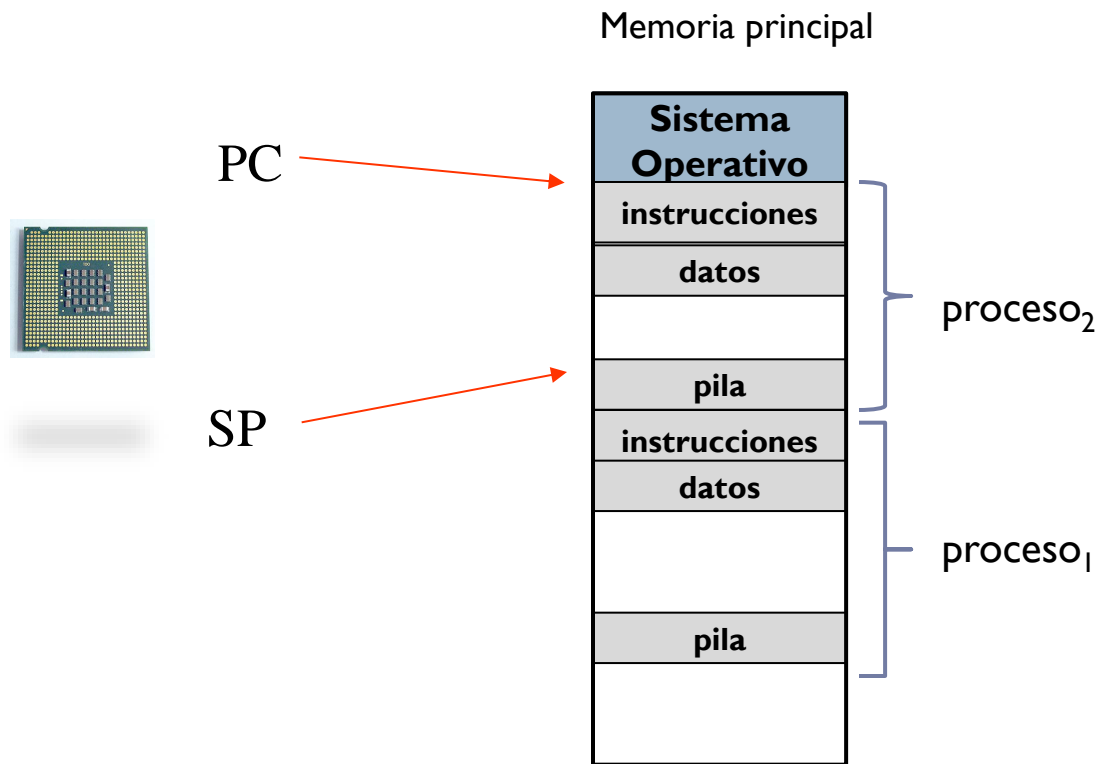


# Múltiples programas cargados en memoria

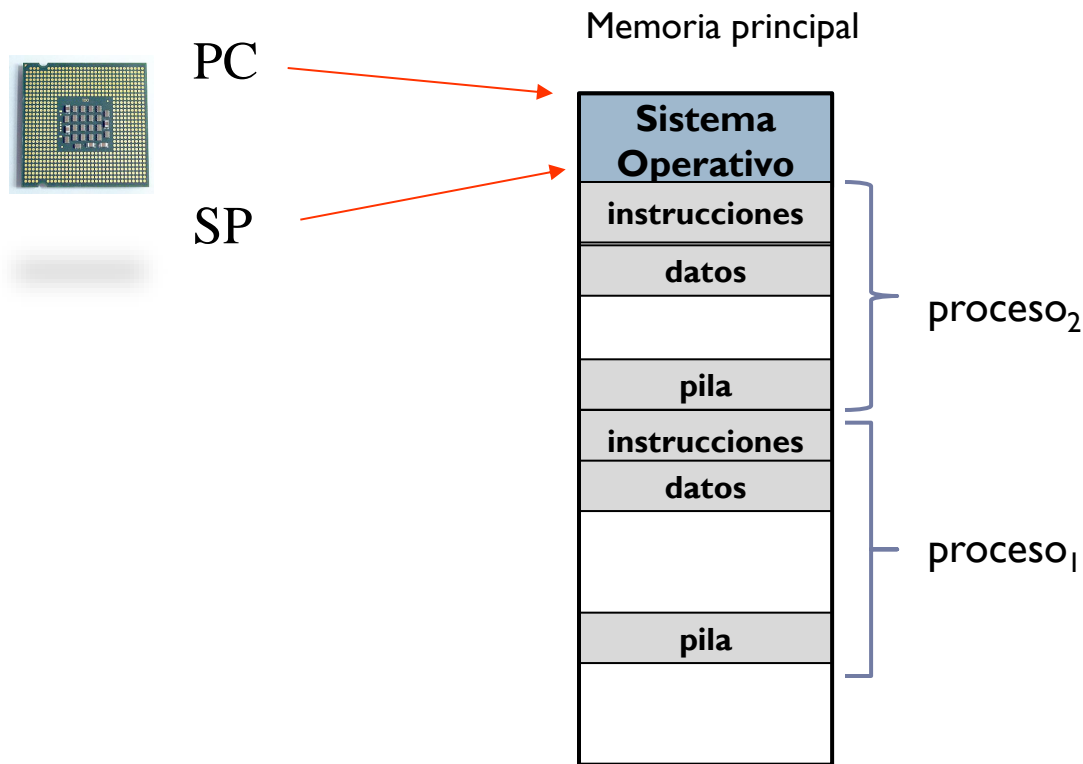
Memoria principal



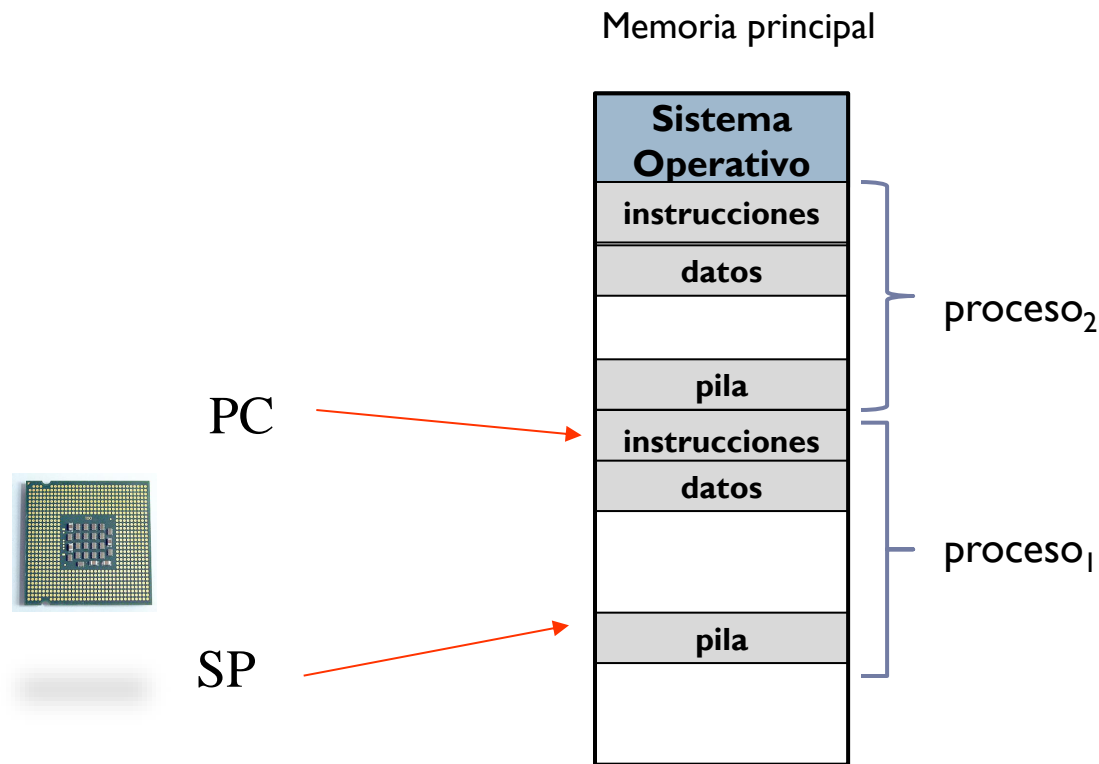
# Múltiples programas cargados en memoria



# Múltiples programas cargados en memoria

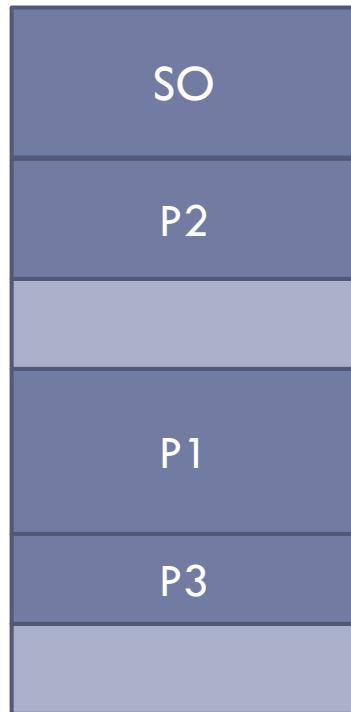


# Múltiples programas cargados en memoria



# Multiprogramación: protección de memoria

- ▶ Un computador puede tener varios programas en memoria.
- ▶ Hay que asignar un espacio de memoria a cada programa en ejecución (proceso).




Hace falta asegurar que un programa no accede a la zona de memoria asignada a otro programa

# Problema de protección de memoria

- ¿Qué ocurre si en el programa ejecuta las siguientes instrucciones?

```
li t0, 8  
sw t0, 0(x0)
```



# Problema de protección de memoria

- ¿Qué ocurre si en el programa ejecuta las siguientes instrucciones?

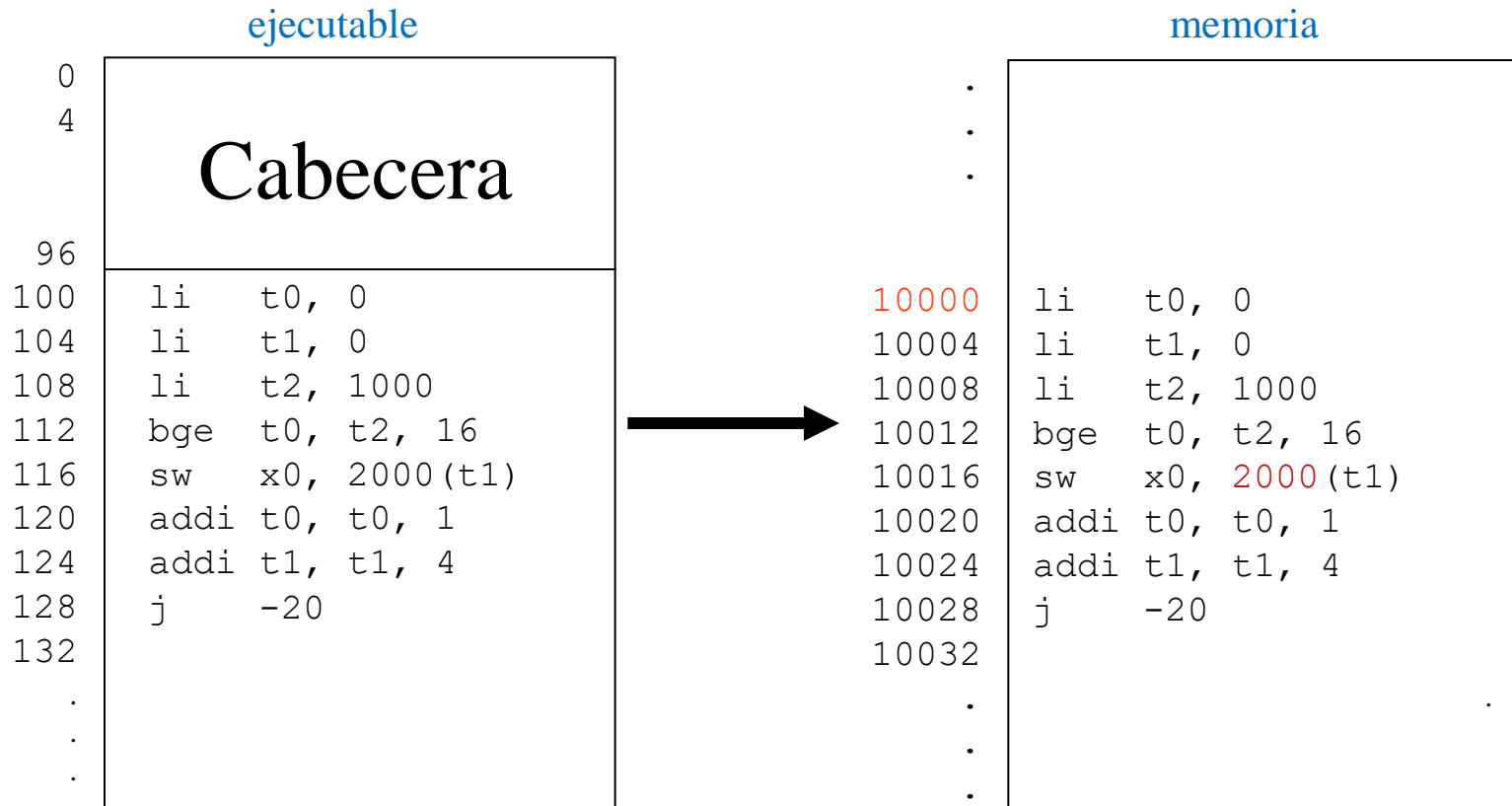
```
li t0, 8  
sw t0, 0(x0)
```

Acceso ilegal a la dirección física 0  
que no está asignada al programa



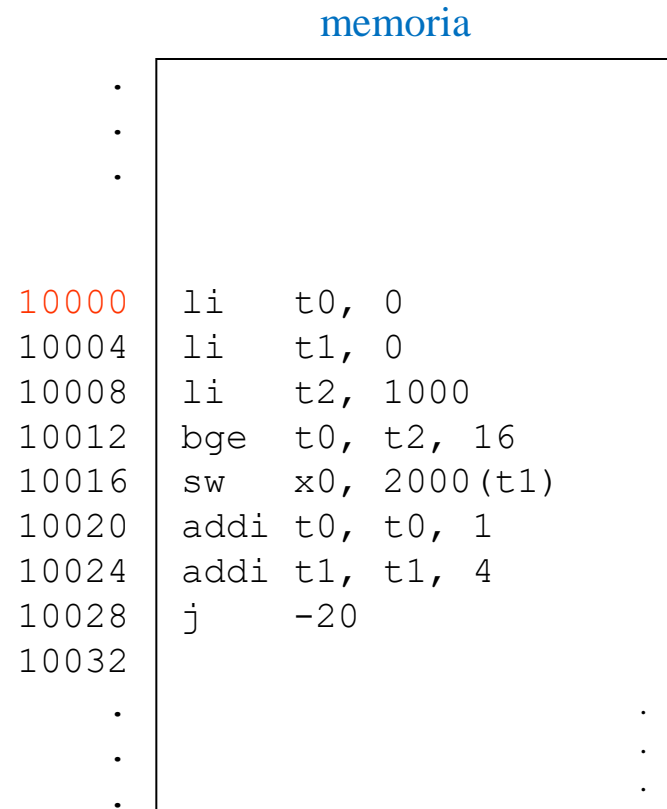
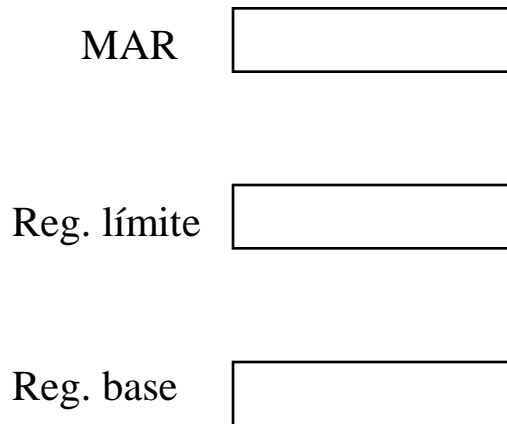
# Reubicación hardware (con registro límite)

- ▶ Se realiza la **traducción y comprobación** durante la ejecución
- ▶ Necesita un hardware especial. Asegura protección.



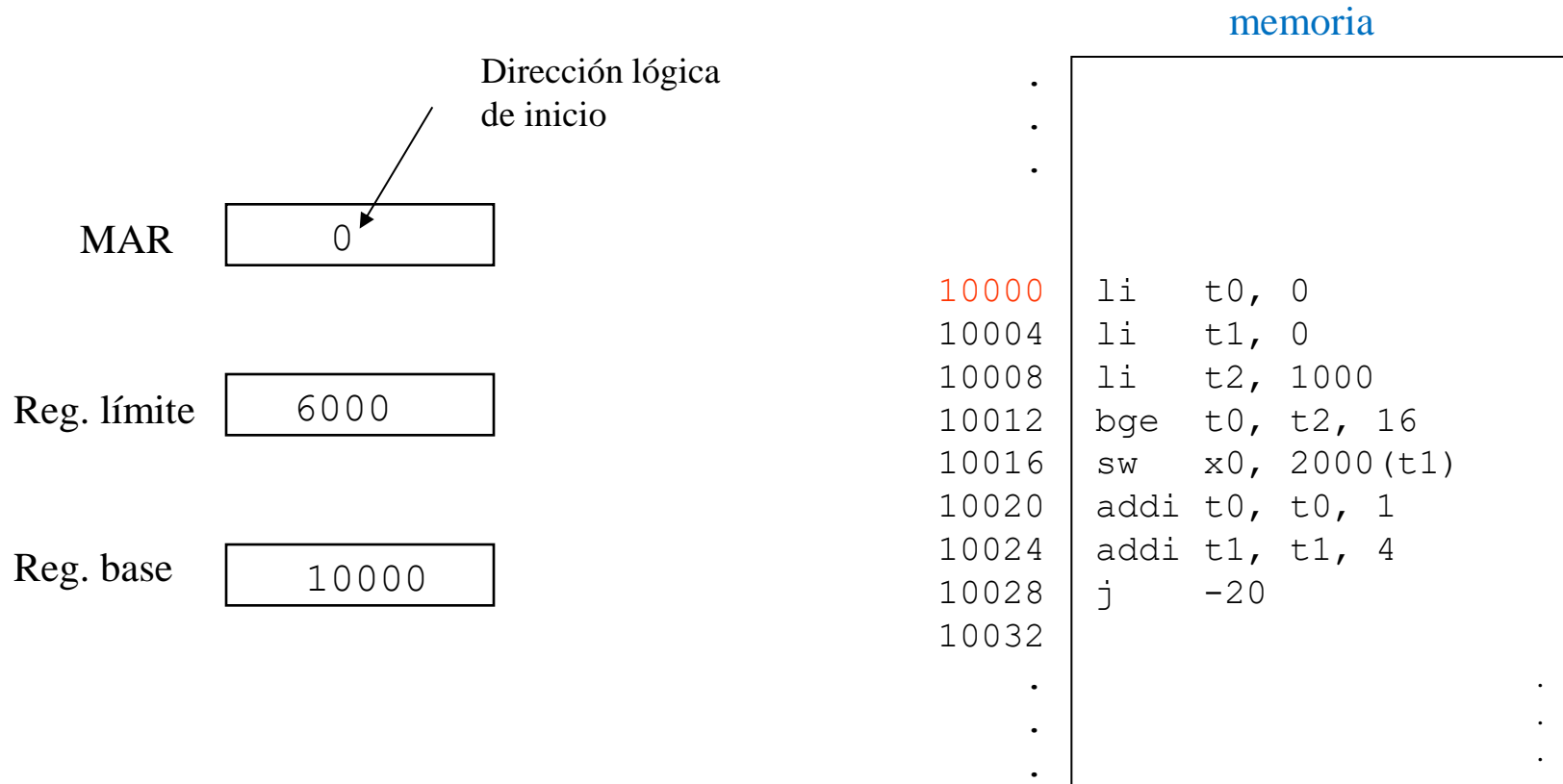
# Ejemplo de soporte hardware

- ▶ **Registro límite:** dirección lógica máxima asignada al programa
- ▶ **Registro base:** dirección de inicio del programa en memoria



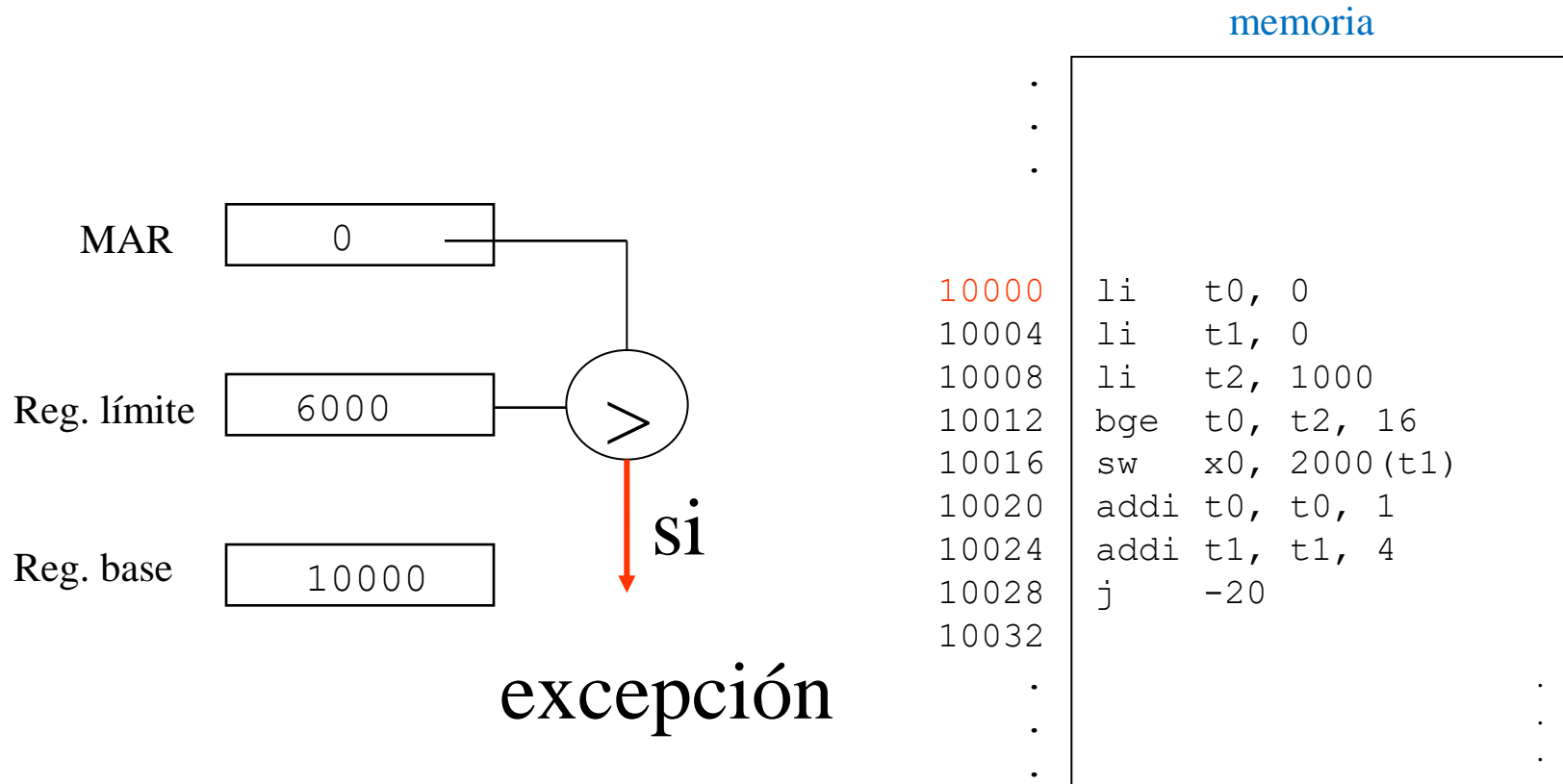
# Ejemplo de soporte hardware

- ▶ **Registro límite:** dirección lógica máxima asignada al programa
- ▶ **Registro base:** dirección de inicio del programa en memoria



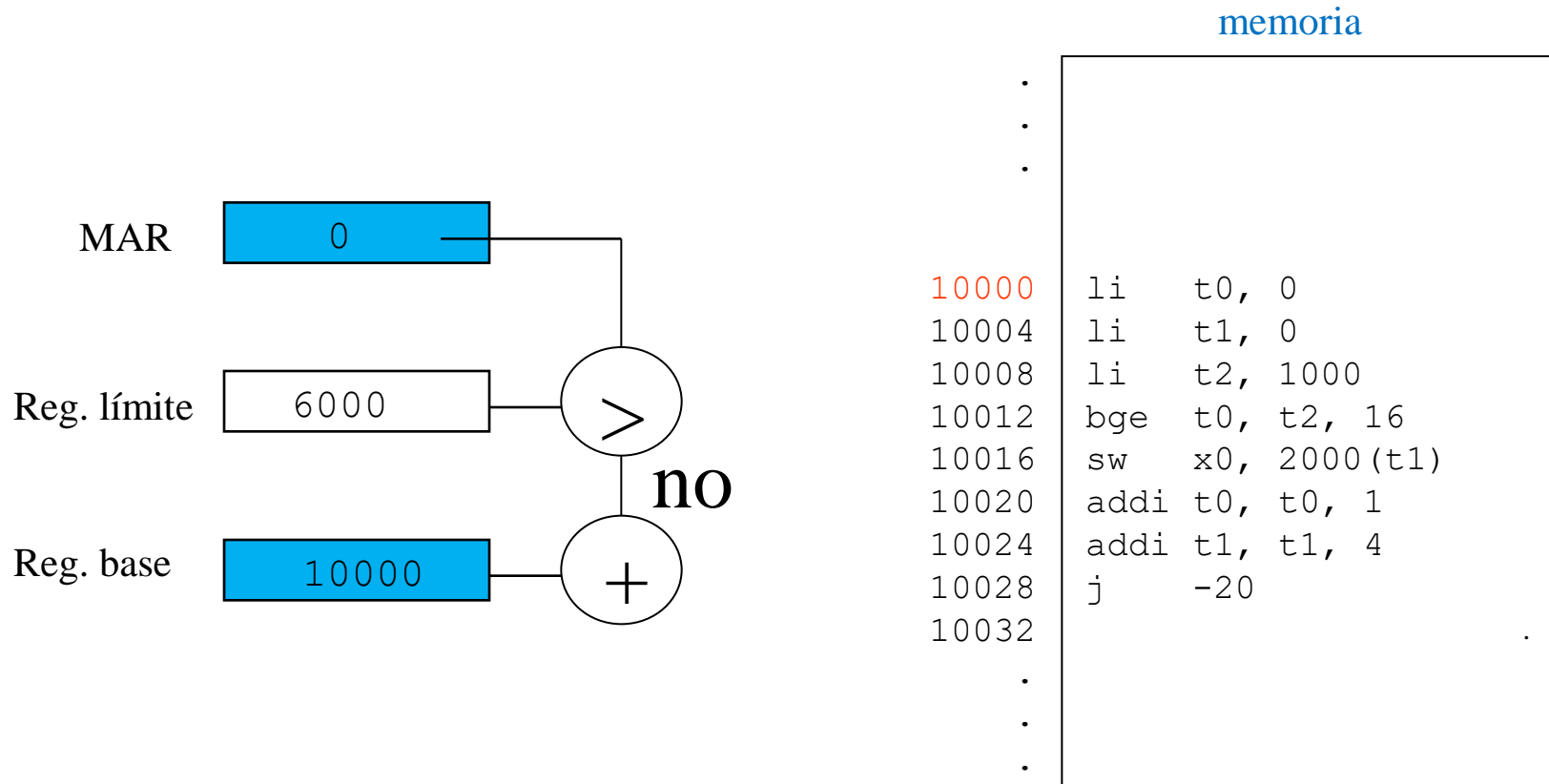
# Ejemplo de soporte hardware

- ▶ **Registro límite:** dirección lógica máxima asignada al programa
- ▶ **Registro base:** dirección de inicio del programa en memoria



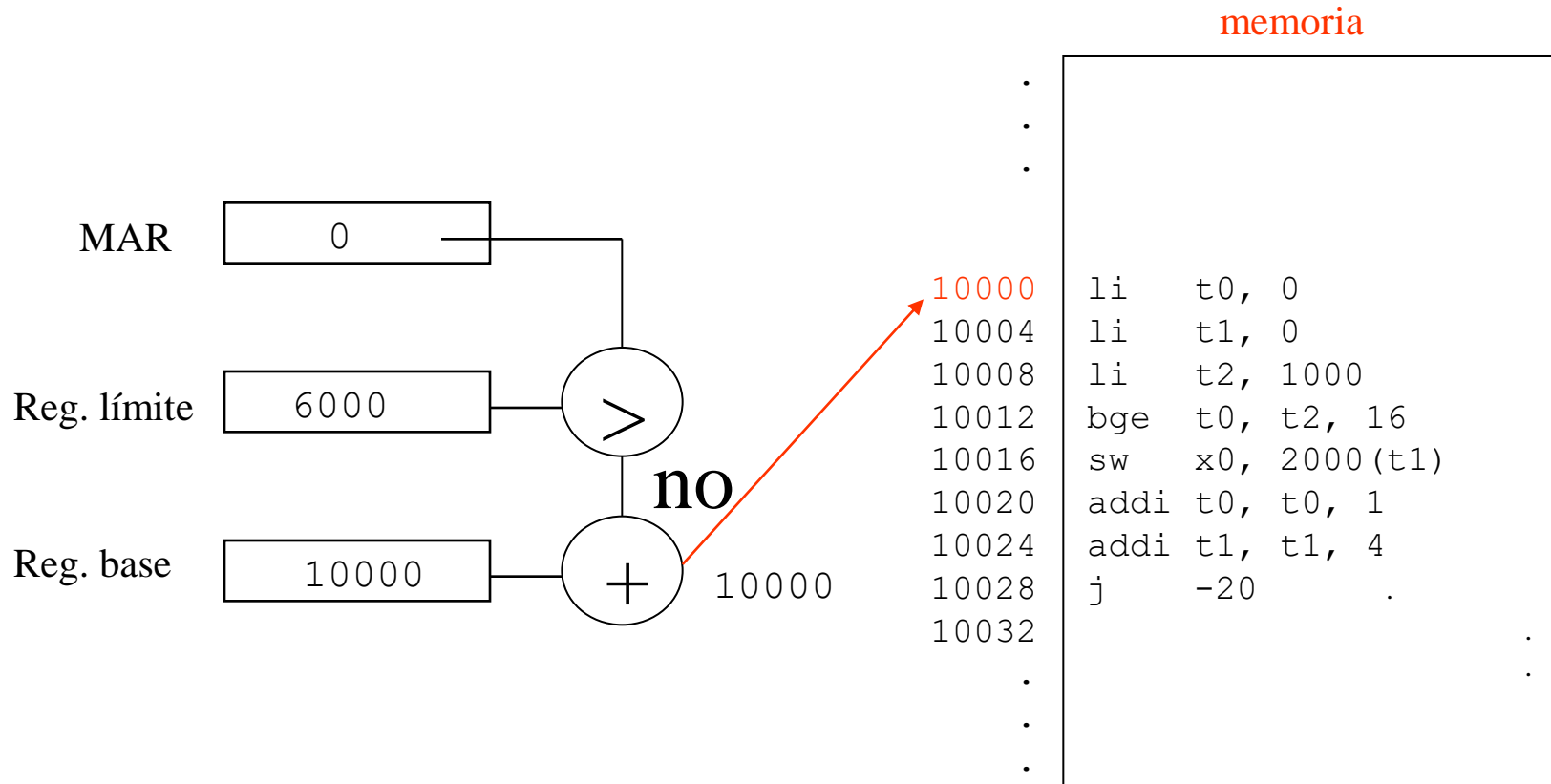
# Ejemplo de soporte hardware

- ▶ **Registro límite:** dirección lógica máxima asignada al programa
- ▶ **Registro base:** dirección de inicio del programa en memoria



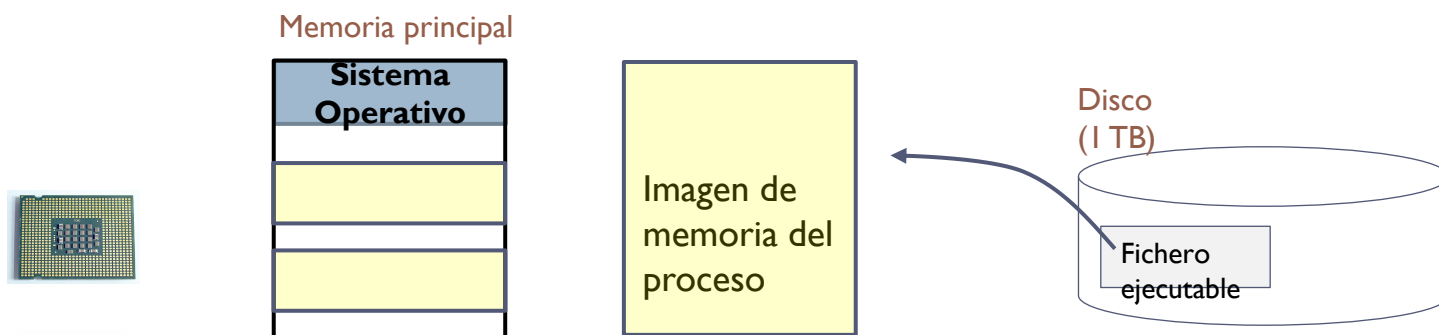
# Ejemplo de soporte hardware

- ▶ **Registro límite:** dirección lógica máxima asignada al programa
- ▶ **Registro base:** dirección de inicio del programa en memoria

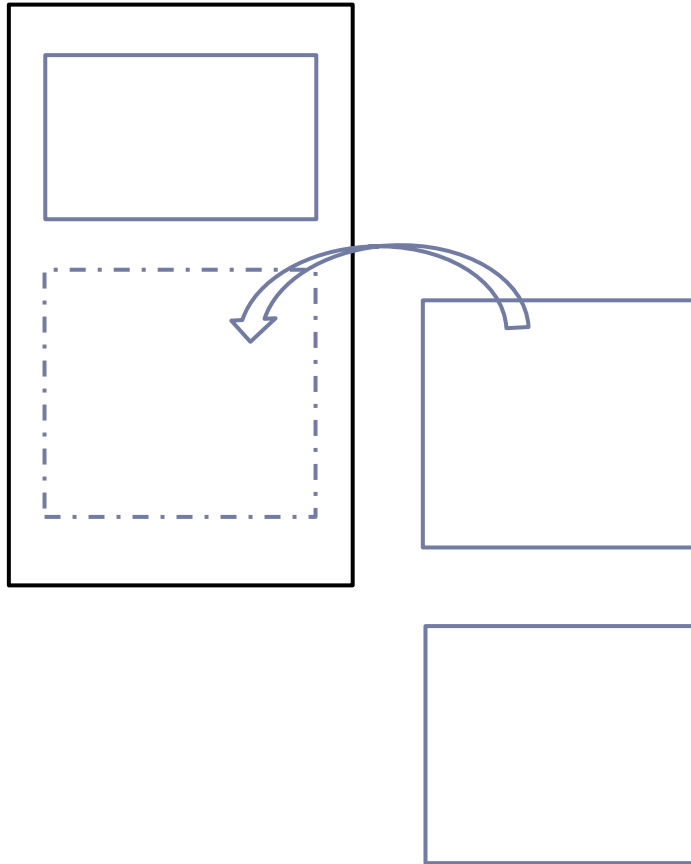


# Sistemas **sin** memoria virtual

- ▶ En los sistemas sin memoria virtual, el programa se carga completamente en memoria para su ejecución.
- ▶ Principales problemas (2/2):
  - ▶ Si la **imagen de memoria** de un proceso es **más grande que la memoria principal**, no es posible su ejecución.
  - ▶ El gran **tamaño de la imagen** en memoria de un proceso **puede impedir ejecutar otros procesos**.



# Overlays



- ▶ En los años 1950-85 el IBM Mainframe-PC tenía poca memoria y sin memoria virtual.
- ▶ Usar *overlays* era una técnica popular para cargar parte del programa cuando se usaba, y descargar para hacer hueco cuando no era necesario.



# Sistemas sin memoria virtual

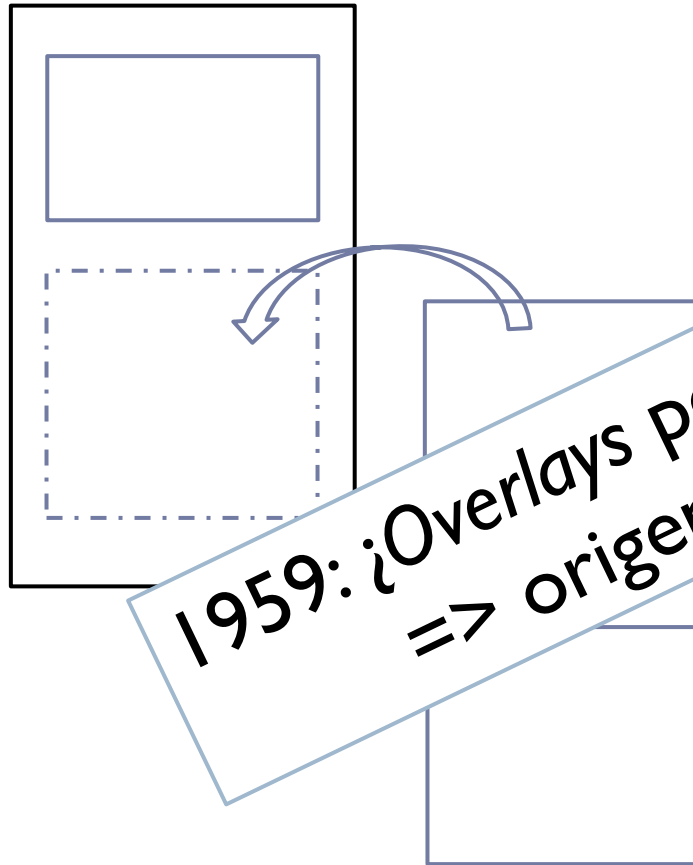
## Principales problemas (resumen)

- ▶ **Reubicación y protección.**
- ▶ **Si la imagen en memoria del proceso es mayor que la memoria disponible, no se puede ejecutar.**
- ▶ **El número de programas activos en memoria limitado.**

En un computador de 32 bits:

- ▶ ¿Cuál es el tamaño máximo teórico del programa que se puede ejecutar?
- ▶ ¿Y si solo se dispone de una memoria de 512 MiB?
- ▶ Si cada programa ocupa 100 MiB, ¿cuántos puedo ejecutar?

# Overlays

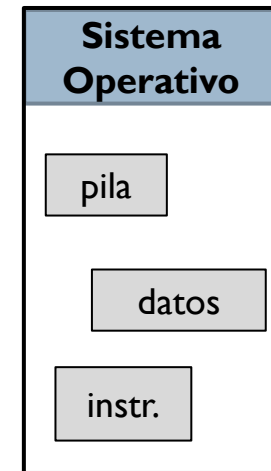


1959: ¿Overlays por hardware?  
=> origen de la memoria virtual

- ▶ En los años 1950-85 el IBM Mainframe y el PC tenía poca memoria física. El uso de overlays era una técnica popular para cargar parte del programa cuando se usaba, y descargar para hacer hueco cuando no era necesario.

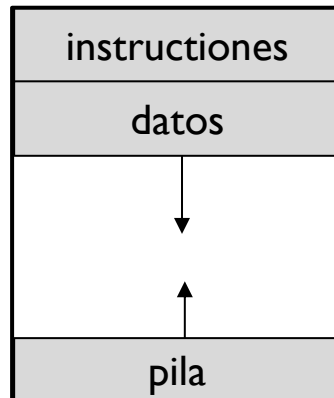
# Sistemas **con** memoria virtual

- ▶ Los programas se cargan parcialmente en memoria principal para su ejecución:
  - ▶ Cuando se necesite una parte del mismo, se carga en memoria principal dicha parte
  - ▶ Cuando no se necesite, se mueve a memoria secundaria (ej.: ssd, disco duro, etc.)
- ▶ Principales ventajas:
  - ▶ Se puede ejecutar programas cuya imagen es mayor que la memoria principal disponible.
  - ▶ Se pueden ejecutar más programas a la vez.
  - ▶ Cada programa tiene su propio espacio.

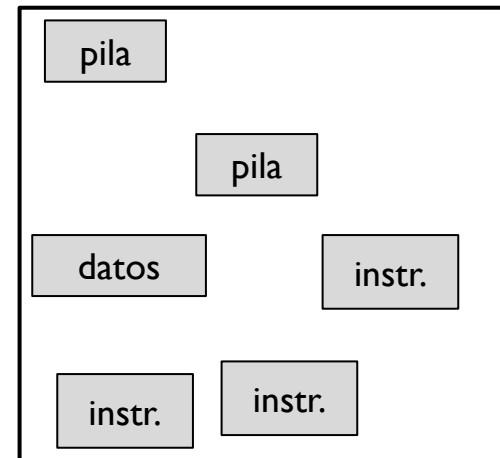


# Sistemas **con** memoria virtual

Imagen de memoria



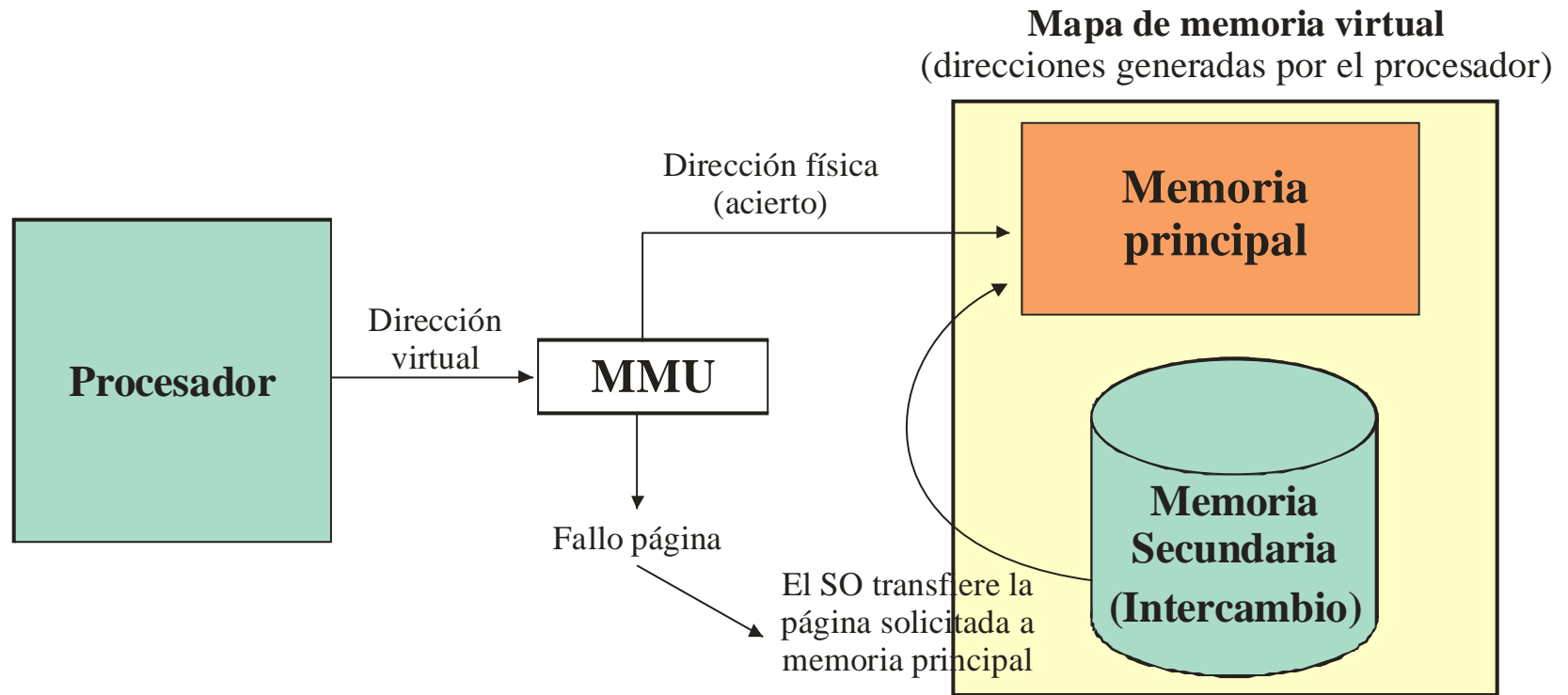
Memoria principal



# Fundamentos de la memoria virtual

La M.V. utiliza dos niveles:

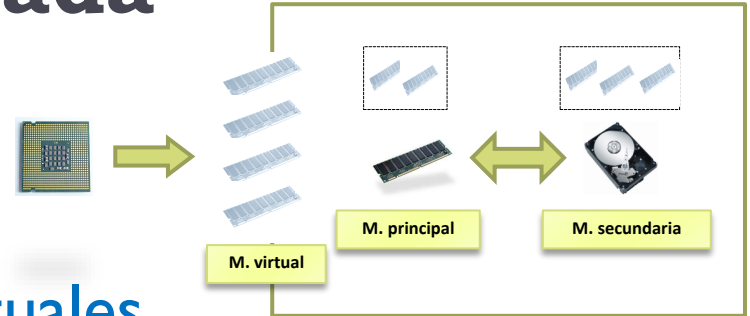
- ❑ Memoria principal: RAM
- ❑ Memoria secundaria: SSD, disco, ...



# Diferentes modelos de memoria virtual

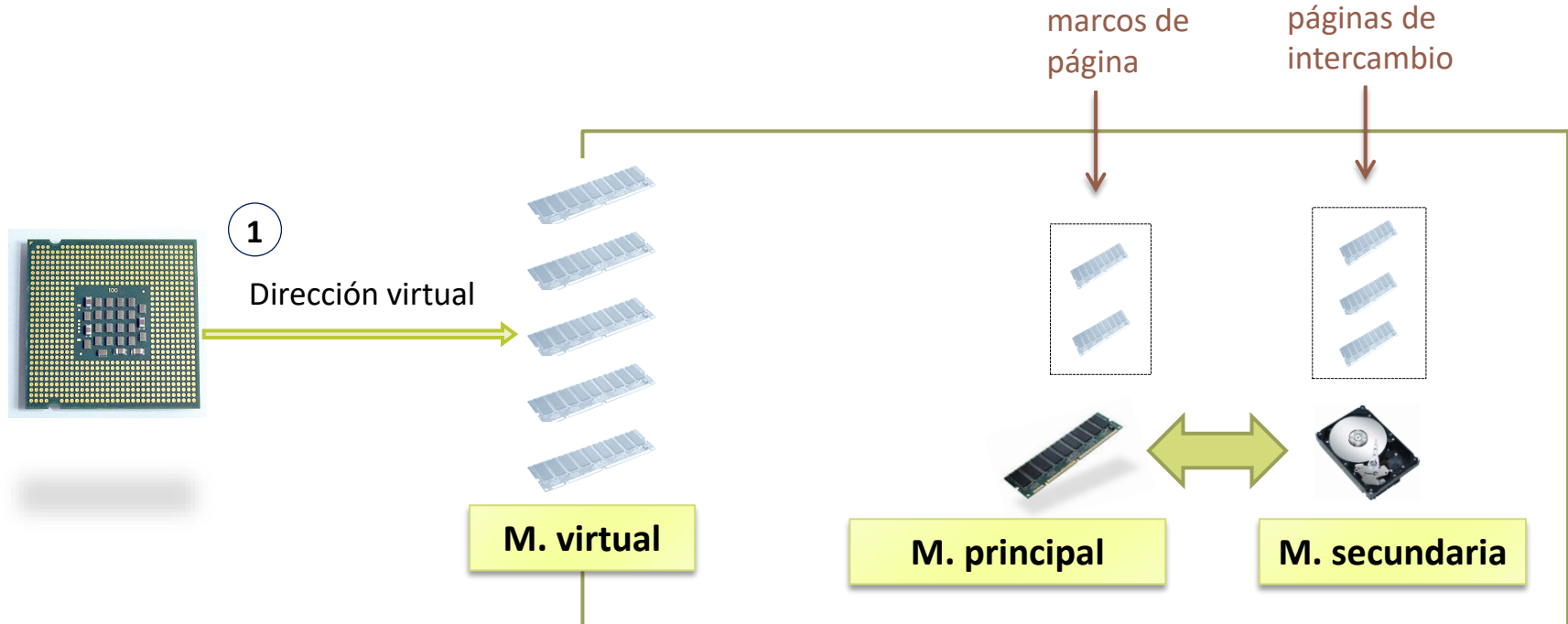
- ▶ Memoria virtual **paginada**
- ▶ Memoria virtual **segmentada**
- ▶ Memoria virtual con **segmentación paginada**

# Memoria virtual **paginada**



- ▶ La CPU genera **direcciones virtuales**
- ▶ El espacio de direcciones virtuales se divide en trozos de igual tamaño denominado **páginas**
- ▶ Se utilizan la memoria principal y el almacenamiento secundario para soporte a la memoria virtual:
  - ▶ La memoria principal se divide en trozos de igual tamaño a las páginas denominados **marcos de página**
  - ▶ La zona del disco que sirve de soporte a la memoria virtual se divide en trozos de igual tamaño denominados **páginas de intercambio** o **páginas de swap**

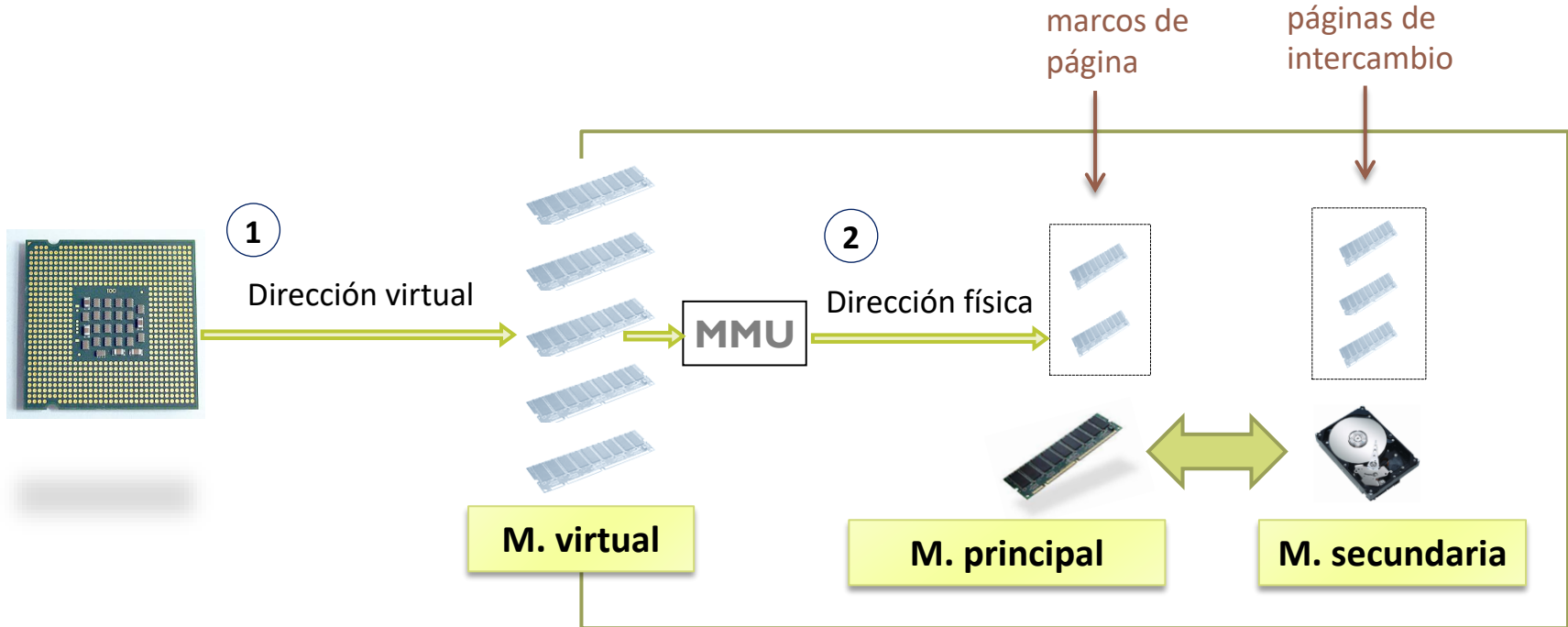
# Espacio de direcciones virtual



- ▶ **Espacio virtual de direcciones (M. virtual)**
  - ▶ Los programas manejan un espacio virtual que reside en M.P.+M.S.

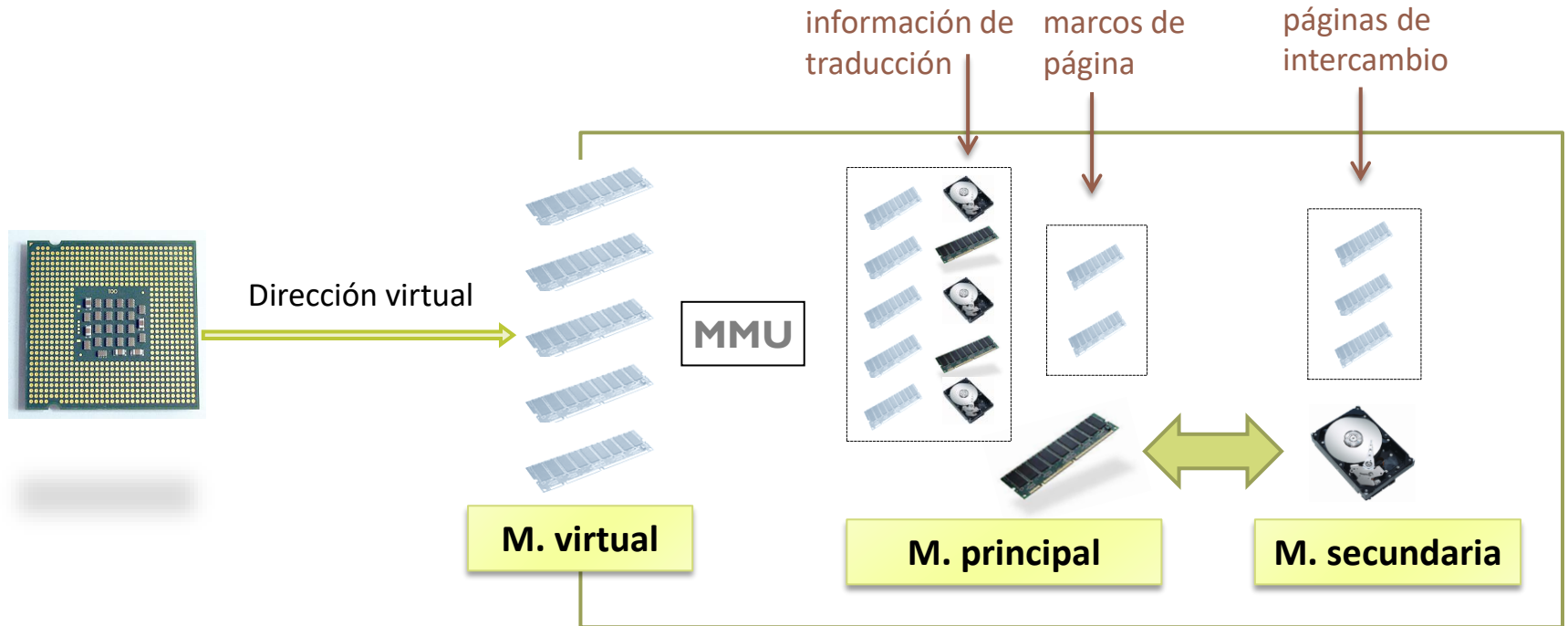


# Espacio de direcciones virtual

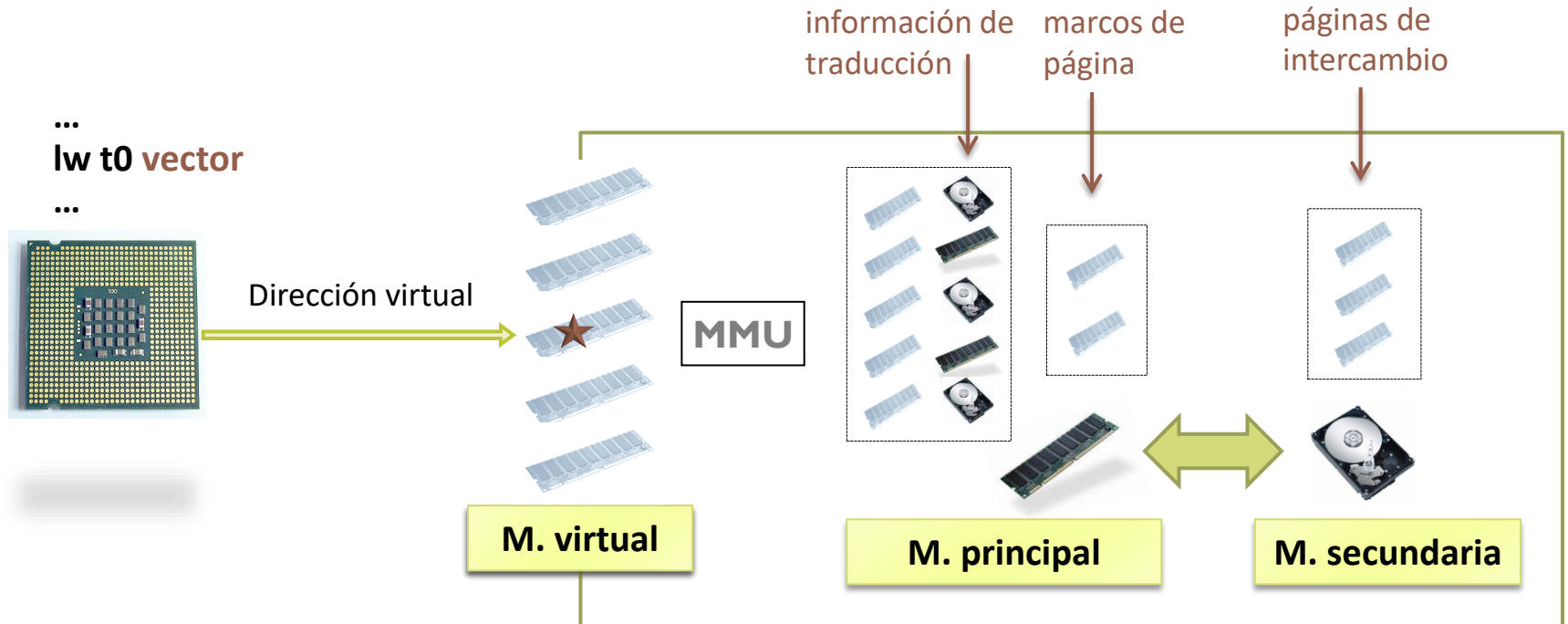


- ▶ **Espacio virtual de direcciones (M. virtual)**
  - ▶ Los programas manejan un espacio virtual que reside en M.P.+M.S.
  - ▶ MMU: Unidad de gestión de memoria

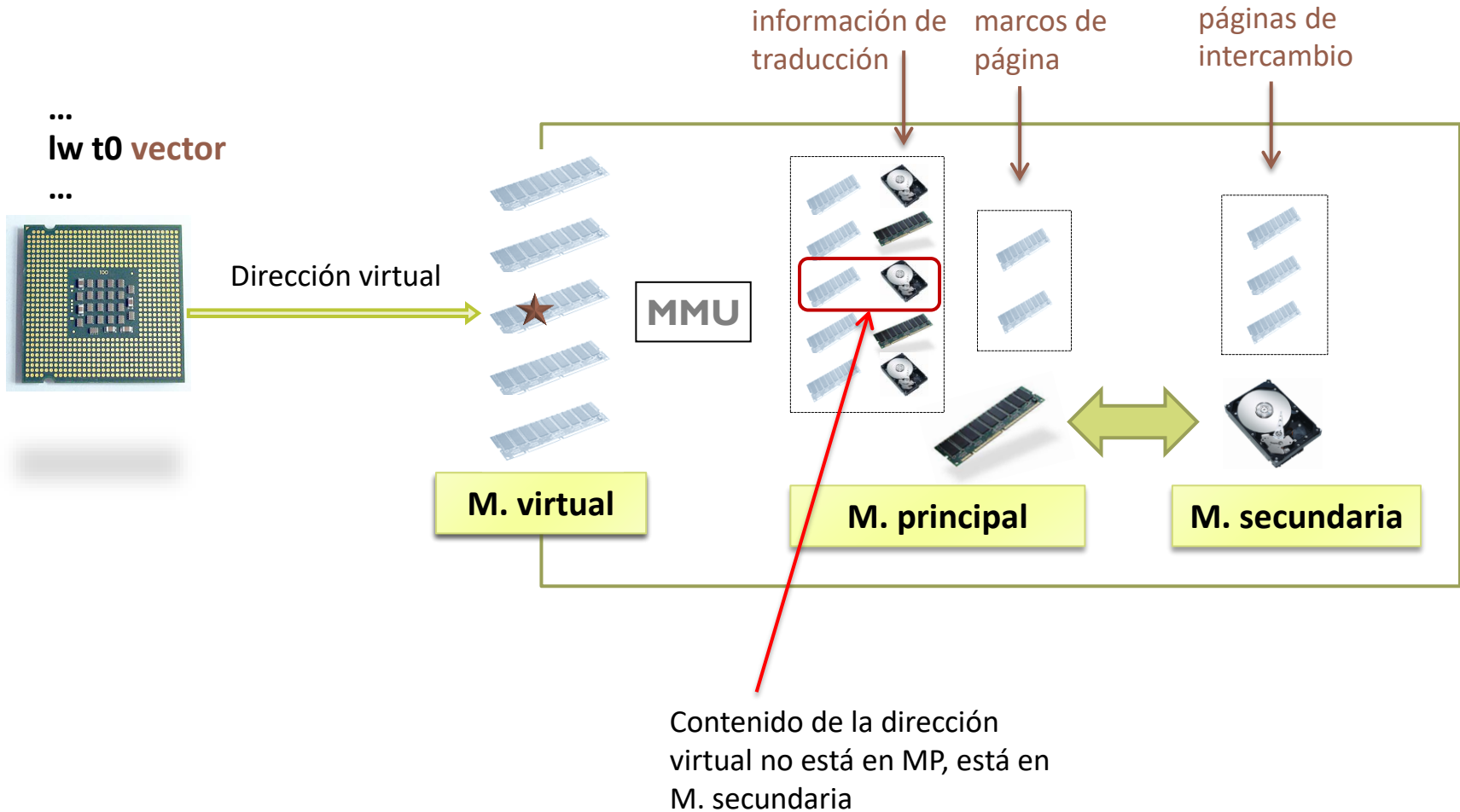
# Fundamentos de la memoria virtual



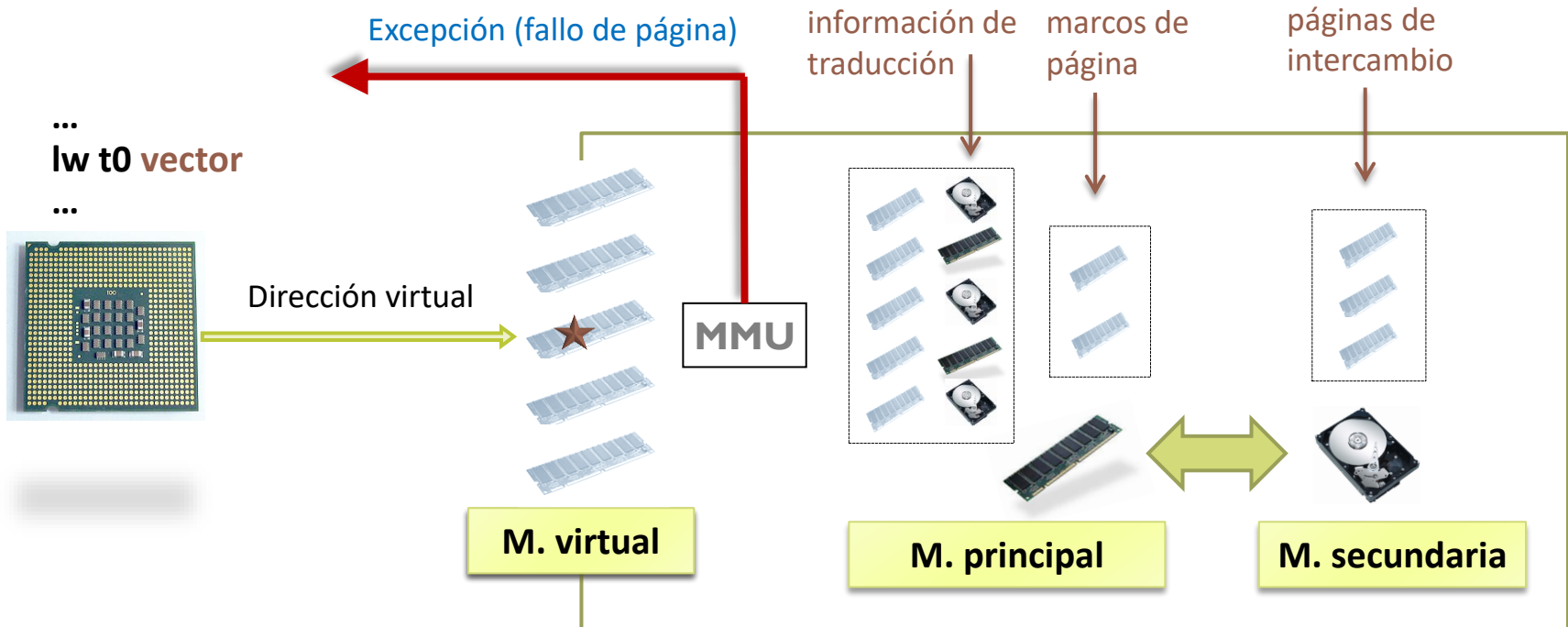
# Fundamentos de la memoria virtual



# Fundamentos de la memoria virtual

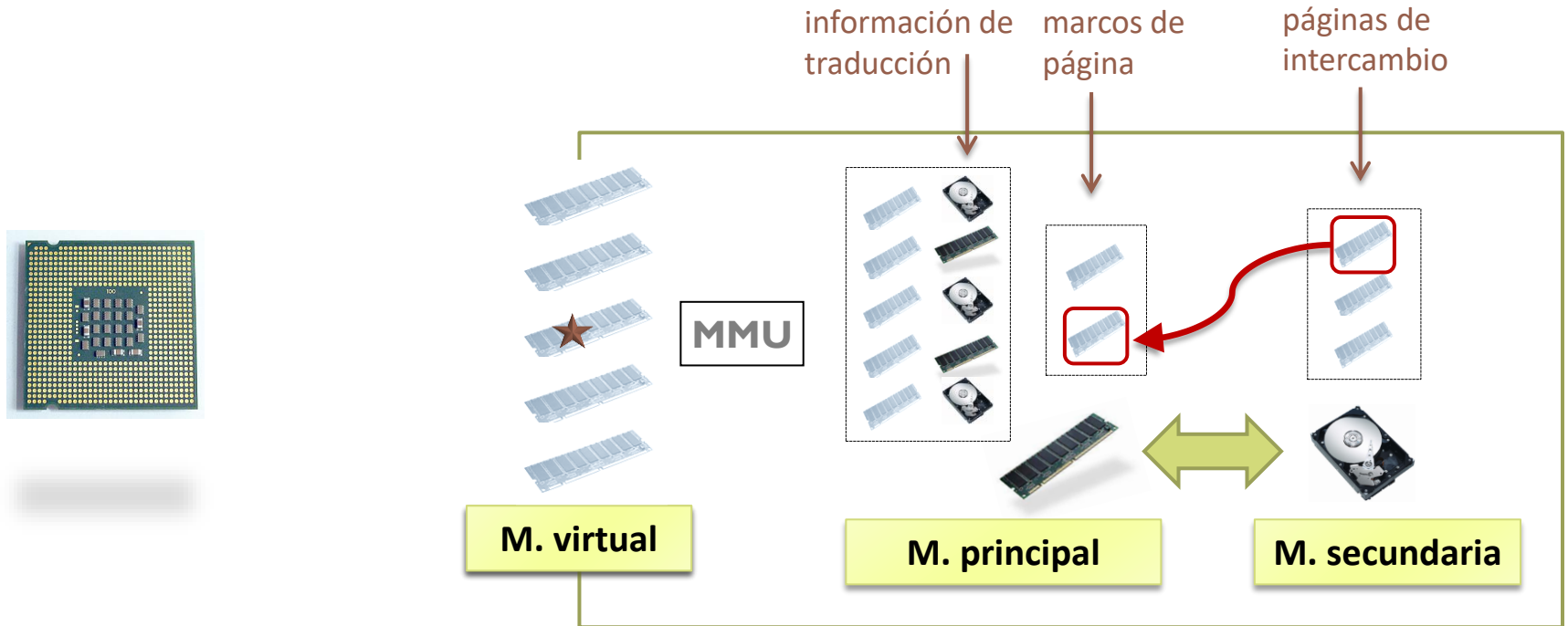


# Fundamentos de la memoria virtual



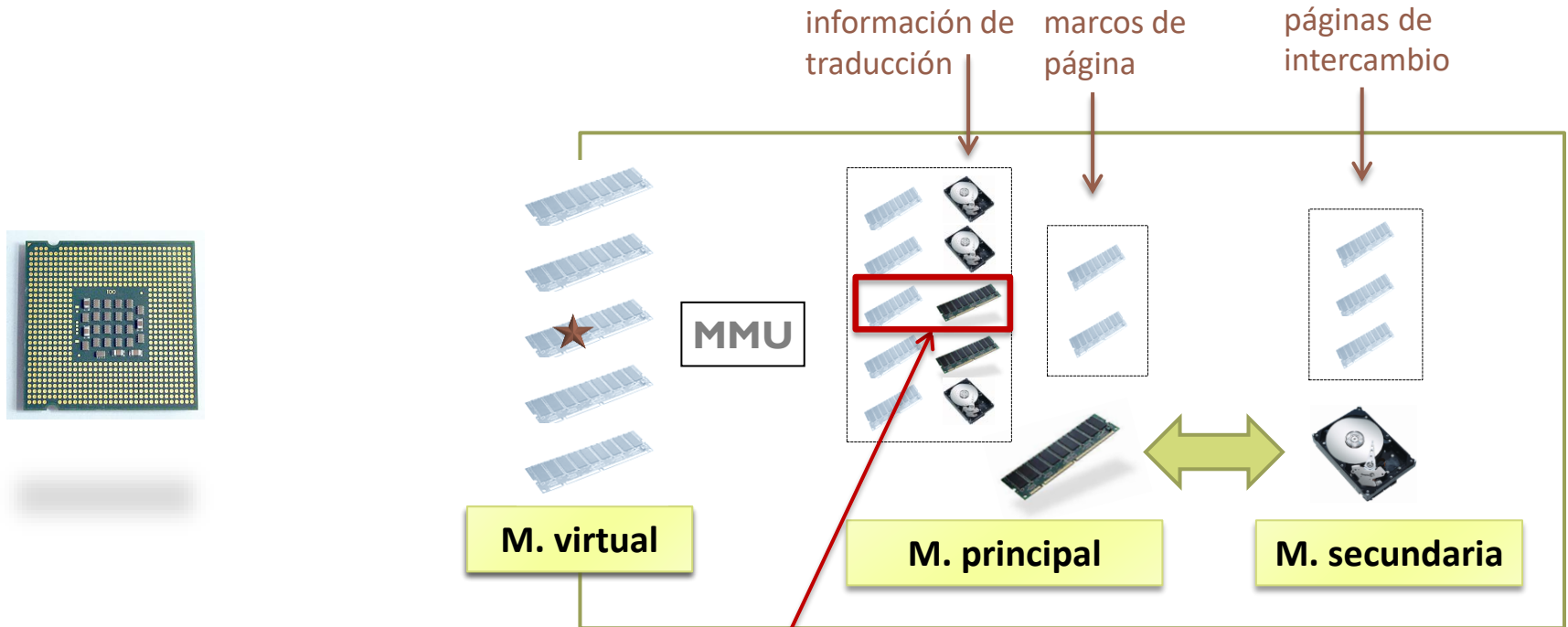
- ▶ El fallo de página es una excepción que provoca que el procesador ejecute la rutina de tratamiento asociada que está implementada en el sistema operativo.
- ▶ Se pide la página a disco y el proceso que generó el fallo de página se suspende su ejecución (no puede continuar) y se pone a otro proceso a ejecutar.

# Fundamentos de la memoria virtual



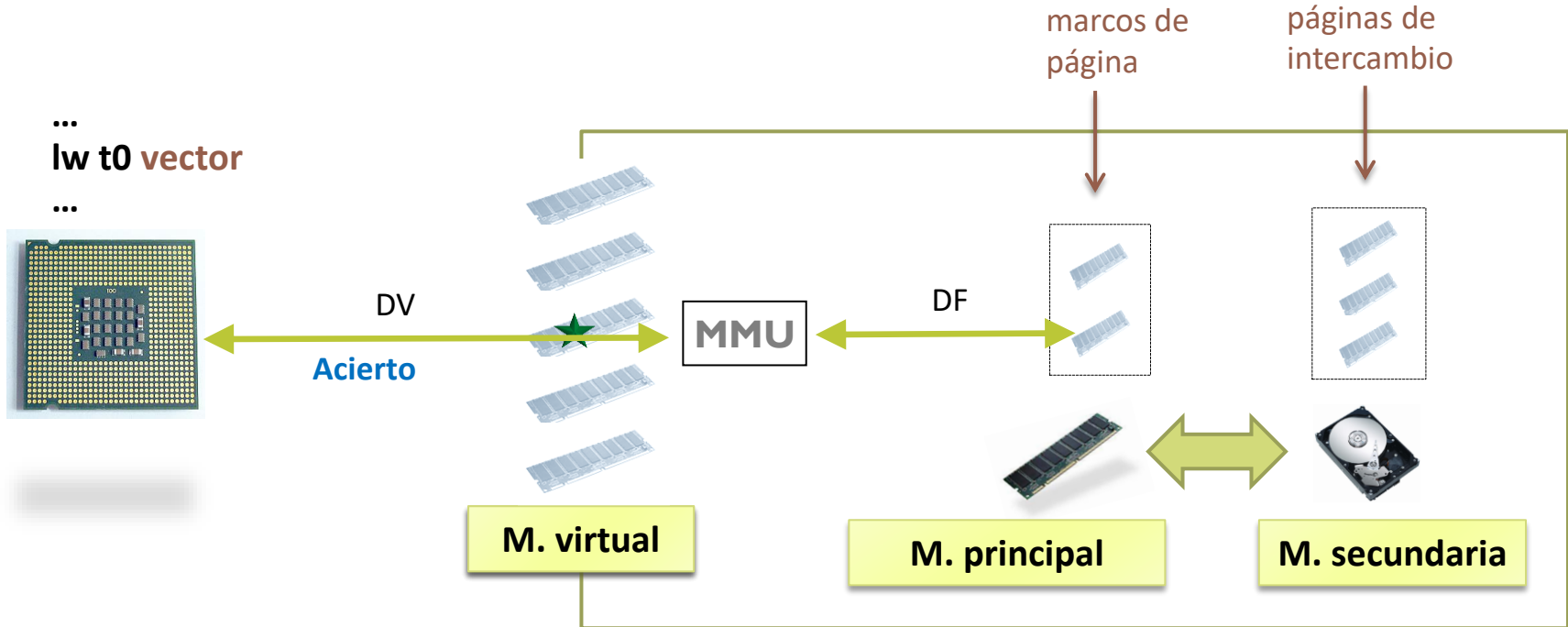
- ▶ Cuando ya está el bloque de disco con la página, el disco pide transferir el 'bloque' solicitado a memoria principal por acceso directo a memoria (DMA)

# Fundamentos de la memoria virtual



- El sistema operativo es interrumpido cuando el 'bloque' solicitado ya en memoria principal y actualiza la información de traducción

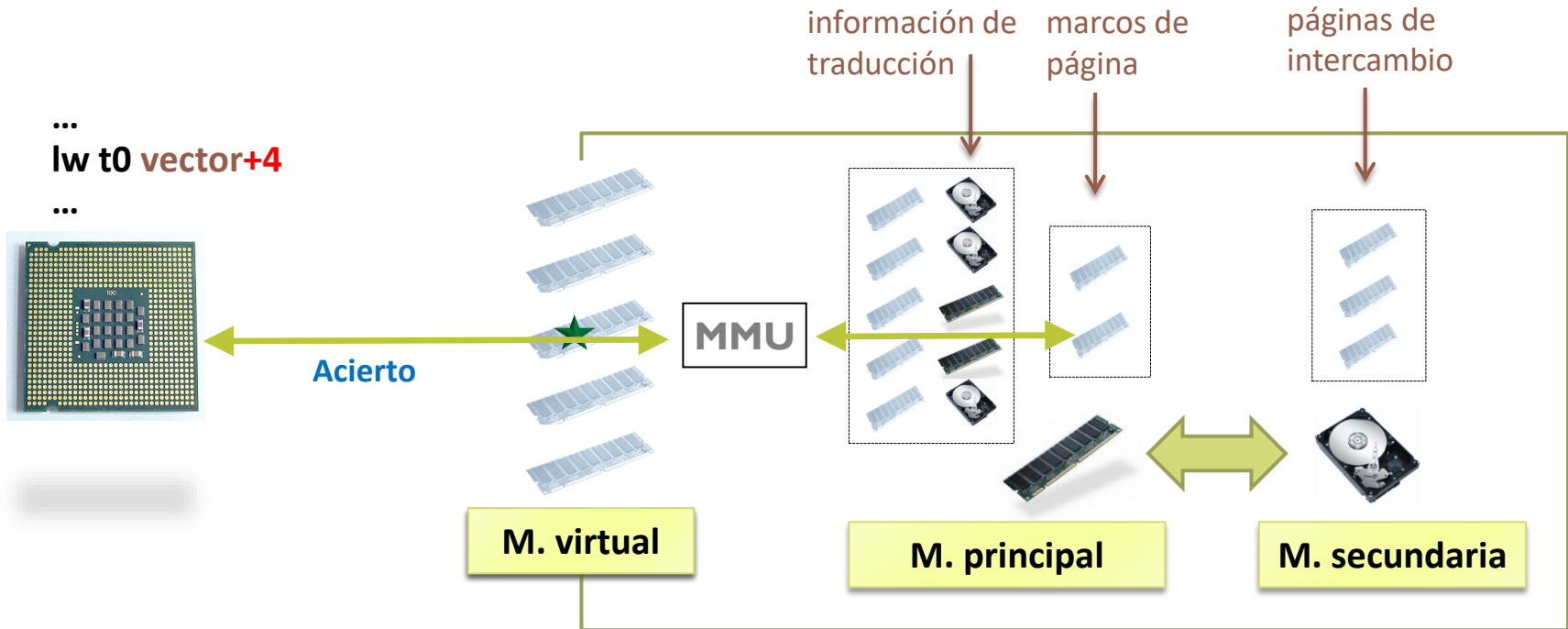
# Fundamentos de la memoria virtual



- ▶ Se reanuda la ejecución del proceso que provocó el fallo y se reanuda la ejecución de la instrucción que provocó el fallo.
- ▶ Se envía la palabra/byte solicitada al procesador.



# Fundamentos de la memoria virtual



- ▶ Sigüientes accesos a la misma página virtual ya están en el mismo marco de página.

# Memoria virtual: windows

Administrador de tareas

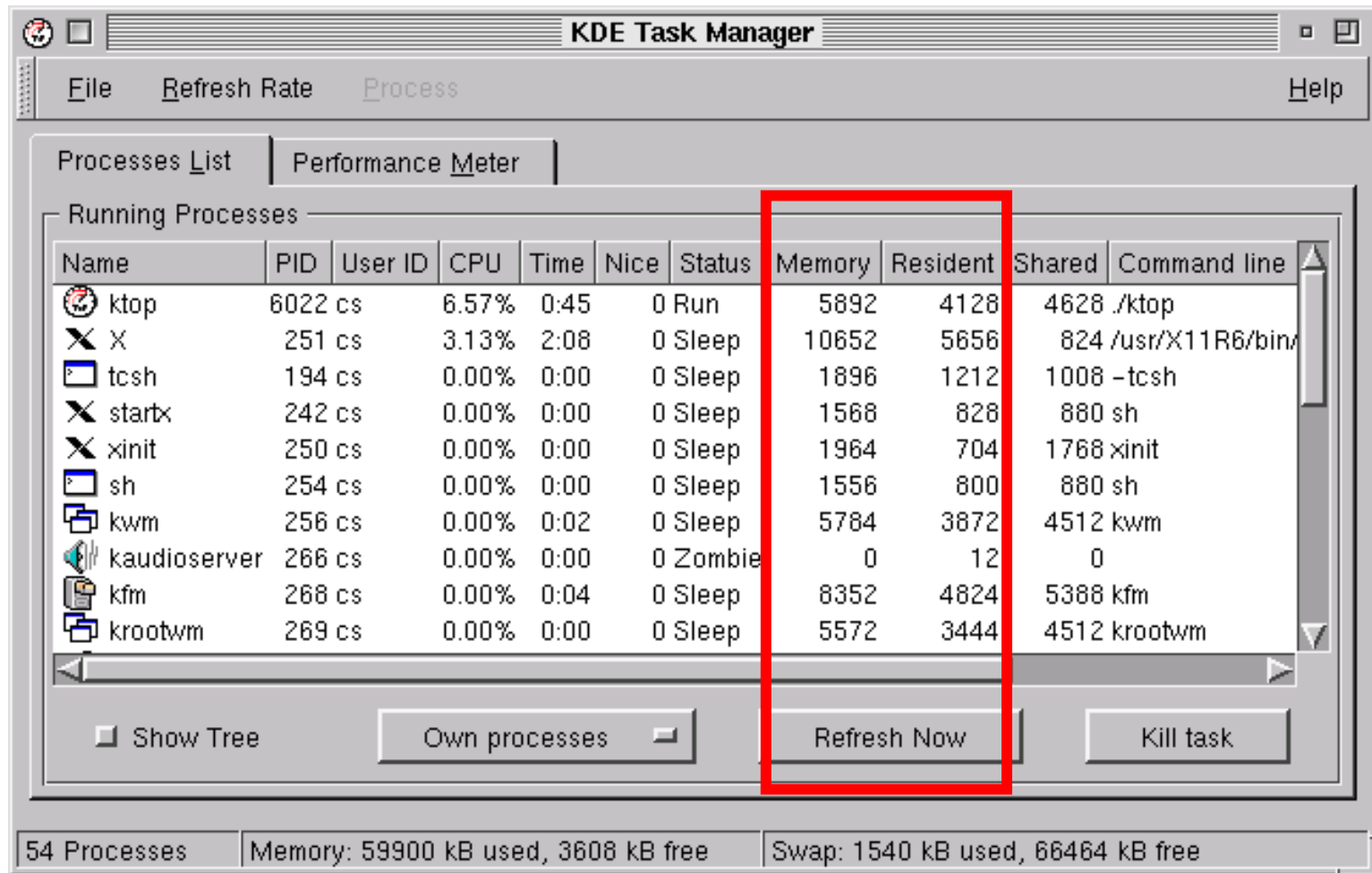
Busque un nombre, publicador o PID

Detalles

Ejecutar nueva tarea Finalizar tarea

Nombre	PID	Estado	Nombre de ...	CPU	Memoria (espaci...	Bloque pagina...	Bloque no paginado	Errores de pági...	arquitect...
AdobeCollabSync.exe	21060	En ejecución	acald	00	472 K	277 K	17 K	14.206	64
AdobeCollabSync.exe	15172	En ejecución	acald	00	2.316 K	277 K	23 K	184.464	64
AggregatorHost.exe	11908	En ejecución	SYSTEM	00	4.232 K	91 K	11 K	1.657.399	64
ai.exe	30404	En ejecución	acald	00	23.004 K	239 K	17 K	18.750	64
AMSPTelemetryServic...	12816	En ejecución	SYSTEM	00	9.192 K	136 K	14 K	55.388	64
AppHelperCap.exe	4660	En ejecución	SYSTEM	00	7.612 K	199 K	20 K	51.137	64
ApplicationFrameHos...	8076	En ejecución	acald	00	20.116 K	703 K	31 K	130.259	64
armsvc.exe	7320	En ejecución	SYSTEM	00	2.436 K	138 K	17 K	5.937	86
audiodg.exe	32816	En ejecución	SERVICIO L...	00	14.092 K	164 K	29 K	270.408	64
backgroundTaskHost...	24424	Suspendido	acald	00	0 K	320 K	25 K	11.449	64
backgroundTaskHost...	20916	Suspendido	acald	00	0 K	425 K	32 K	16.529	64
backgroundTaskHost...	22796	Suspendido	acald	00	0 K	424 K	32 K	15.970	64
backgroundTaskHost...	37252	Suspendido	acald	00	0 K	424 K	32 K	16.439	64
backgroundTaskHost...	8732	Suspendido	acald	00	0 K	252 K	28 K	7.518	64
backgroundTaskHost...	13456	Suspendido	acald	00	0 K	424 K	32 K	16.396	64
backgroundTaskHost...	36004	Suspendido	acald	00	0 K	399 K	38 K	20.945	64
backgroundTaskHost...	22564	Suspendido	acald	00	0 K	425 K	32 K	16.305	64
backgroundTaskHost...	40448	Suspendido	acald	00	0 K	204 K	14 K	8.363	64
BridgeCommunicatio...	46268	En ejecución	acald	00	5.192 K	270 K	21 K	15.036	64
chrome.exe	46064	En ejecución	acald	00	21.420 K	635 K	25 K	28.499	64
chrome.exe	33856	En ejecución	acald	00	7.008 K	635 K	20 K	8.691	64
chrome.exe	45536	En ejecución	acald	00	41.448 K	1.060 K	53 K	124.396	64
chrome.exe	45616	En ejecución	acald	00	1.756 K	144 K	10 K	2.949	64
chrome.exe	45788	En ejecución	acald	00	35.208 K	1.097 K	36 K	72.130	64
chrome.exe	45800	En ejecución	acald	00	7.156 K	689 K	23 K	14.417	64
chrome.exe	45832	En ejecución	acald	00	3.856 K	613 K	13 K	6.536	64
com.docker.backend...	25528	En ejecución	acald	00	23.656 K	256 K	16 K	14.869	64
com.docker.backend...	33860	En ejecución	acald	00	68.160 K	470 K	50 K	132.969	64
com.docker.build.exe	35352	En ejecución	acald	00	20.980 K	220 K	18 K	29.442	64
com.docker.dev-envs...	23548	En ejecución	acald	00	3.628 K	80 K	11 K	3.726	64

# Memoria virtual: linux



KDE Task Manager

File Refresh Rate Process Help

Processes List Performance Meter

Running Processes

Name	PID	User ID	CPU	Time	Nice	Status	Memory	Resident	Shared	Command line
ktop	6022	cs	6.57%	0:45	0	Run	5892	4128	4628	./ktop
X	251	cs	3.13%	2:08	0	Sleep	10652	5656	824	/usr/X11R6/bin/
tcsh	194	cs	0.00%	0:00	0	Sleep	1896	1212	1008	-tcsh
startx	242	cs	0.00%	0:00	0	Sleep	1568	828	880	sh
xinit	250	cs	0.00%	0:00	0	Sleep	1964	704	1768	xinit
sh	254	cs	0.00%	0:00	0	Sleep	1556	800	880	sh
kwm	256	cs	0.00%	0:02	0	Sleep	5784	3872	4512	kwm
kaudioserver	266	cs	0.00%	0:00	0	Zombie	0	12	0	
kfm	268	cs	0.00%	0:04	0	Sleep	8352	4824	5388	kfm
krootwm	269	cs	0.00%	0:00	0	Sleep	5572	3444	4512	krootwm

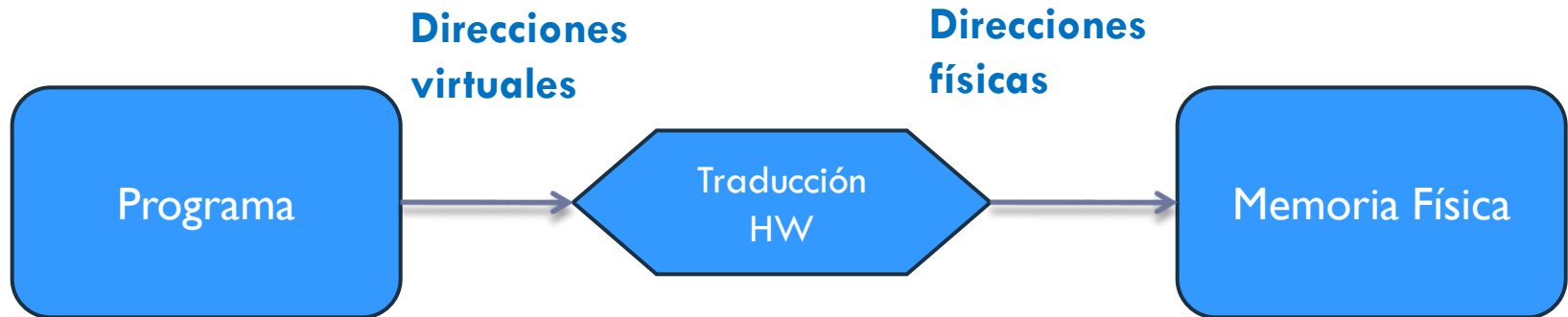
Show Tree Own processes Refresh Now Kill task

54 Processes Memory: 59900 kB used, 3608 kB free Swap: 1540 kB used, 66464 kB free

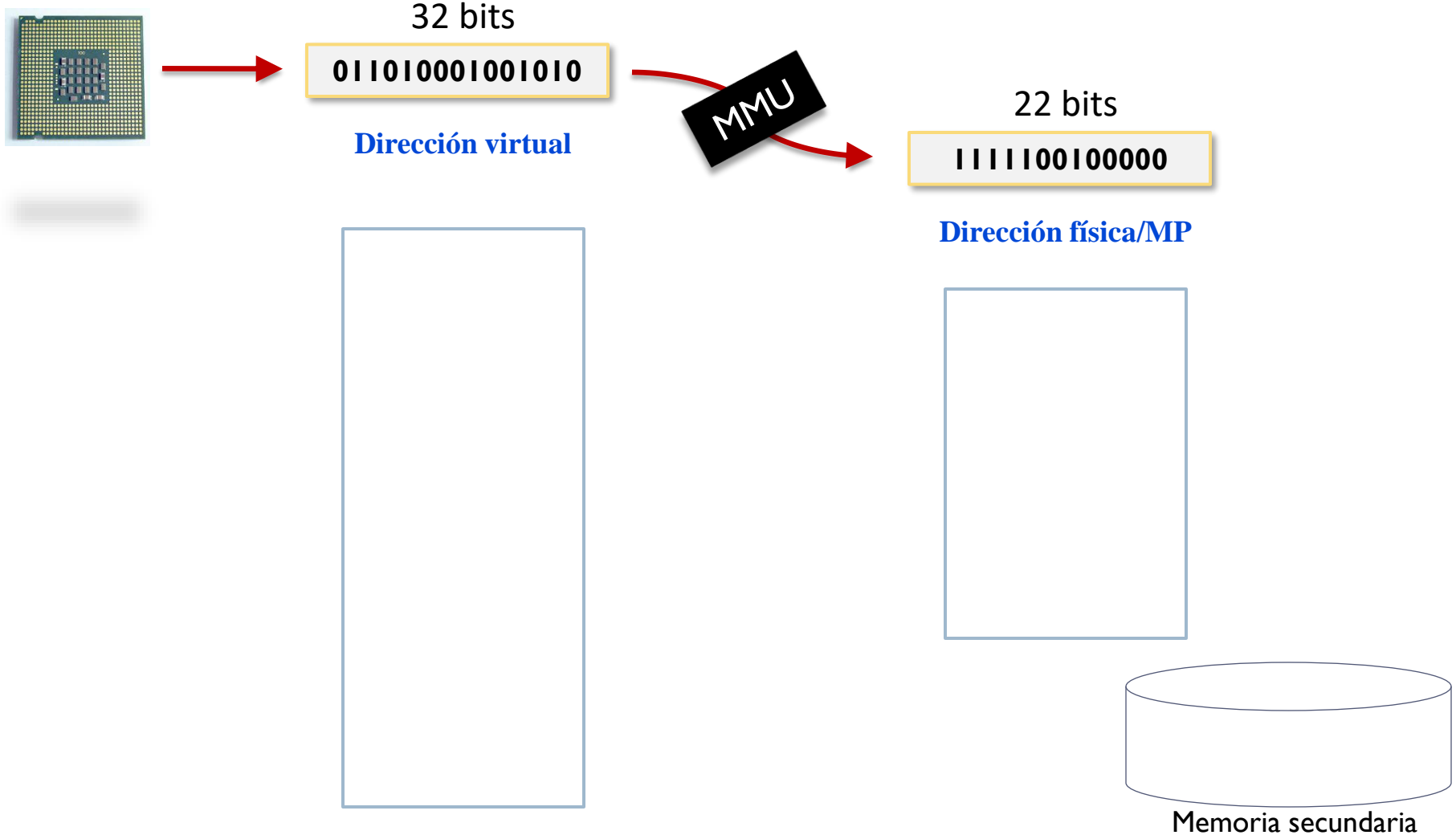
# Memoria virtual paginada

## Traducción

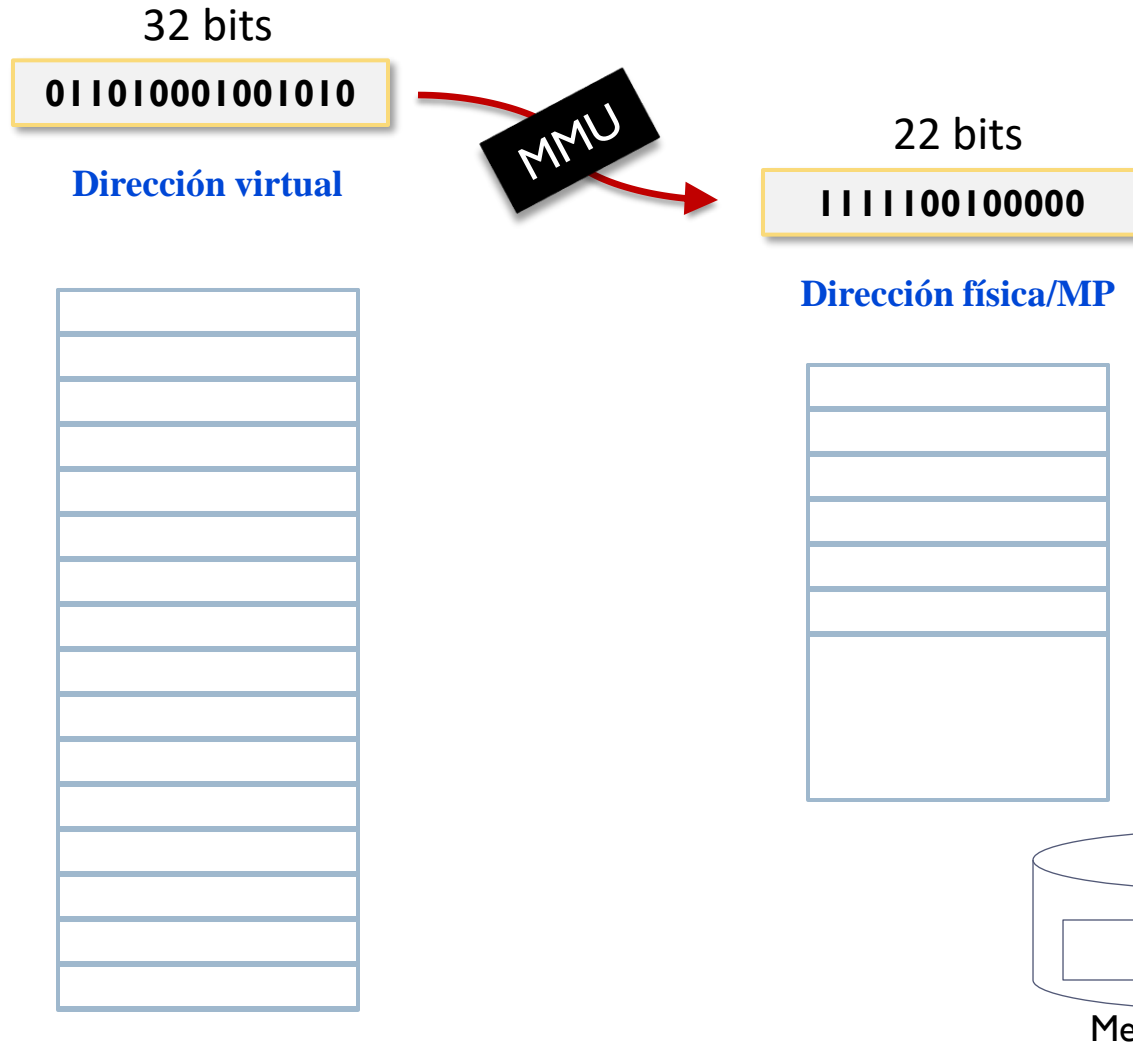
- ▶ Espacio de **direcciones virtuales**:
  - ▶ Direcciones de memoria con las que trabaja cada proceso.
- ▶ Espacio de **direcciones físicas**:
  - ▶ Direcciones de memoria principal en las que residen los datos.



# Ejemplo

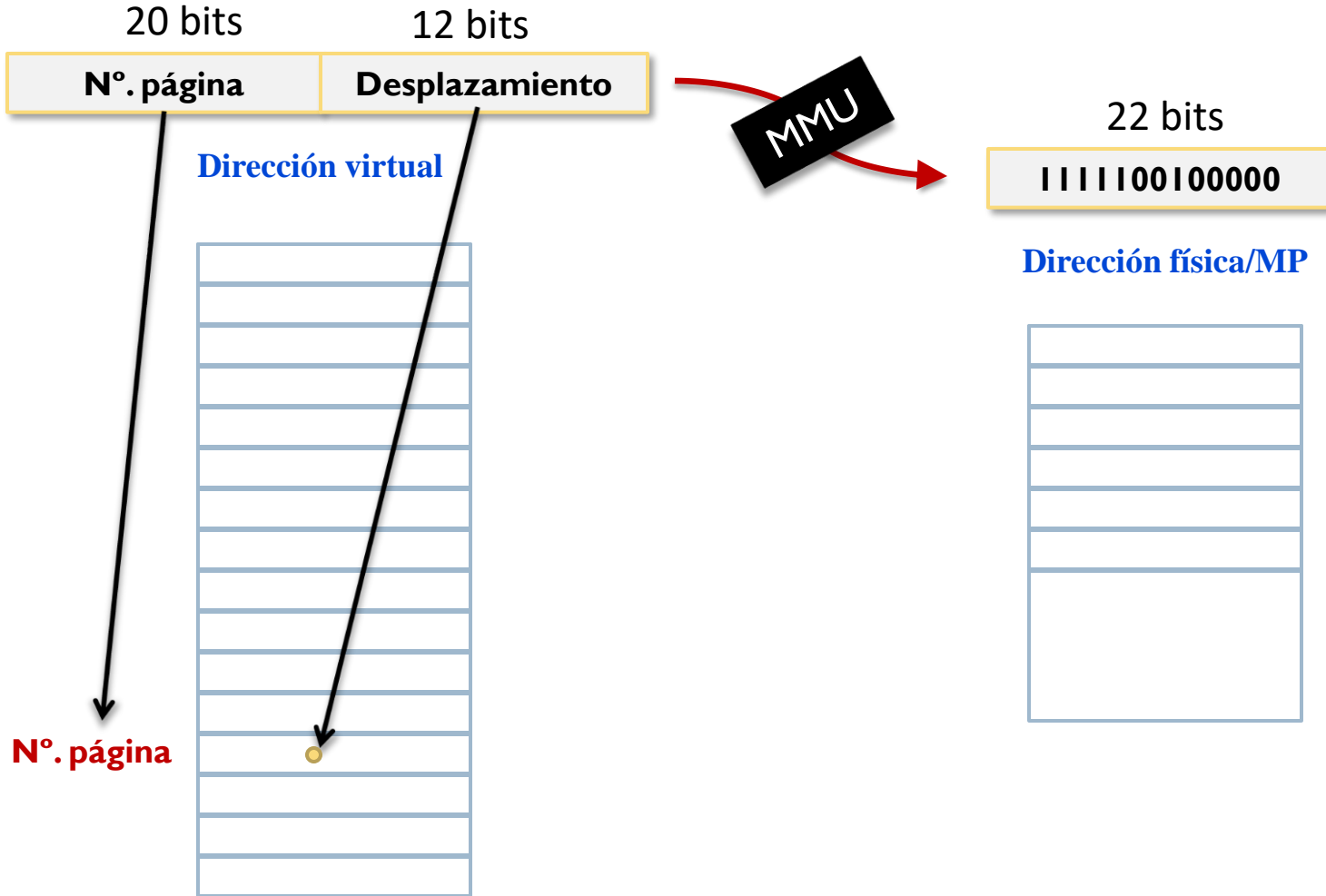


# Ejemplo



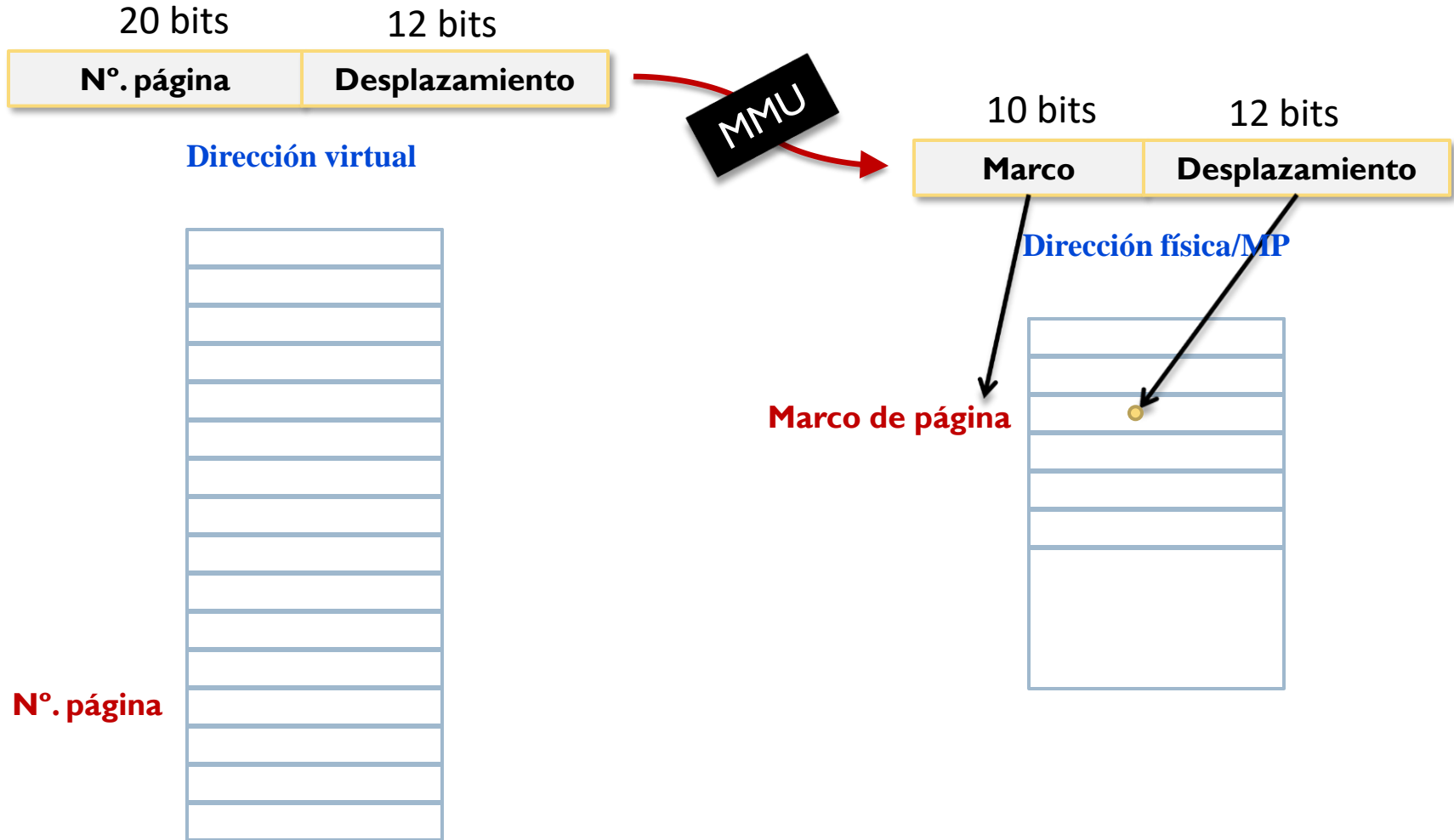
División en bloques del mismo tamaño -> páginas

# Ejemplo



División en bloques del mismo tamaño -> páginas

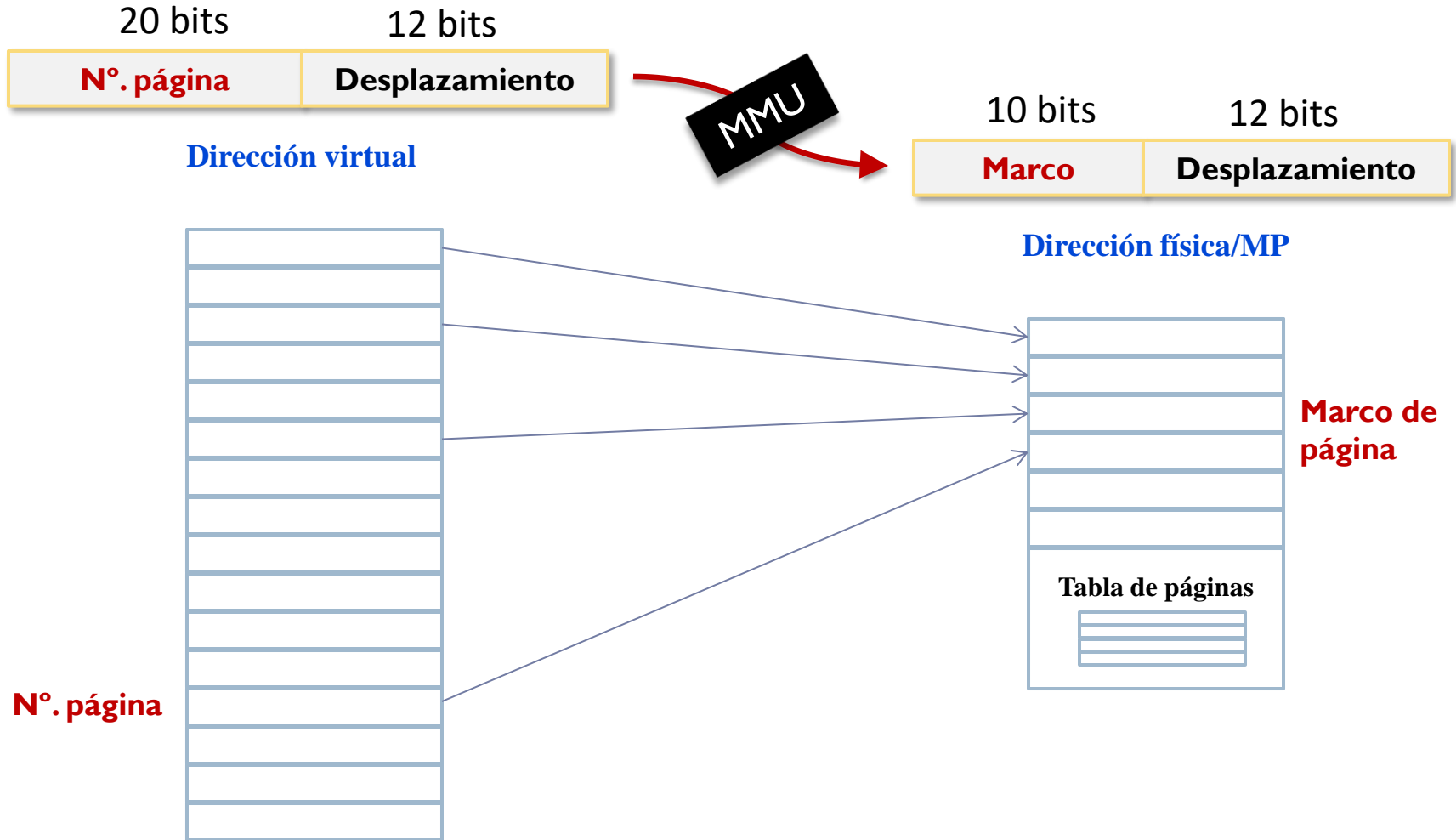
# Ejemplo



División en bloques del mismo tamaño -> páginas

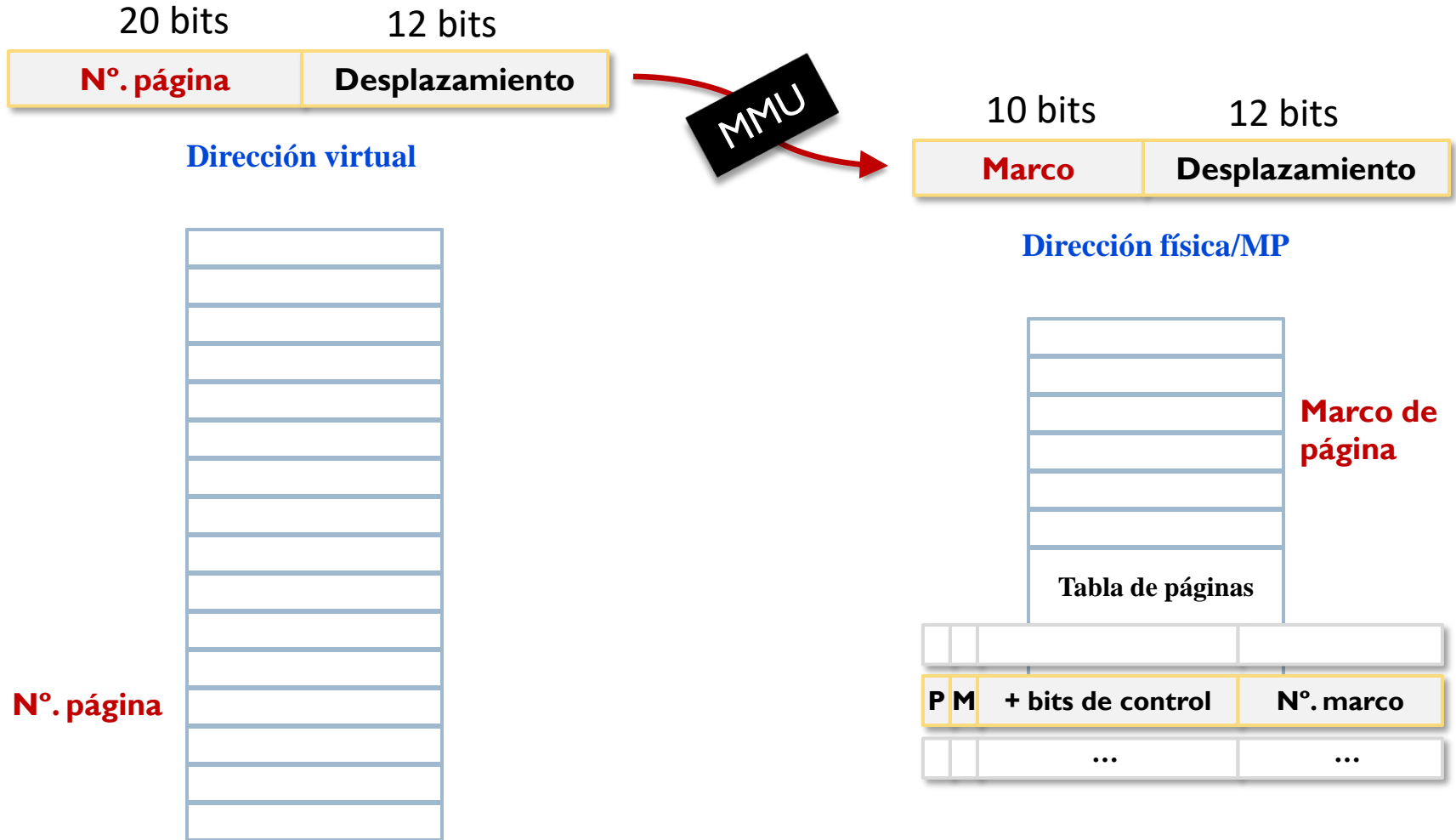


# Ejemplo



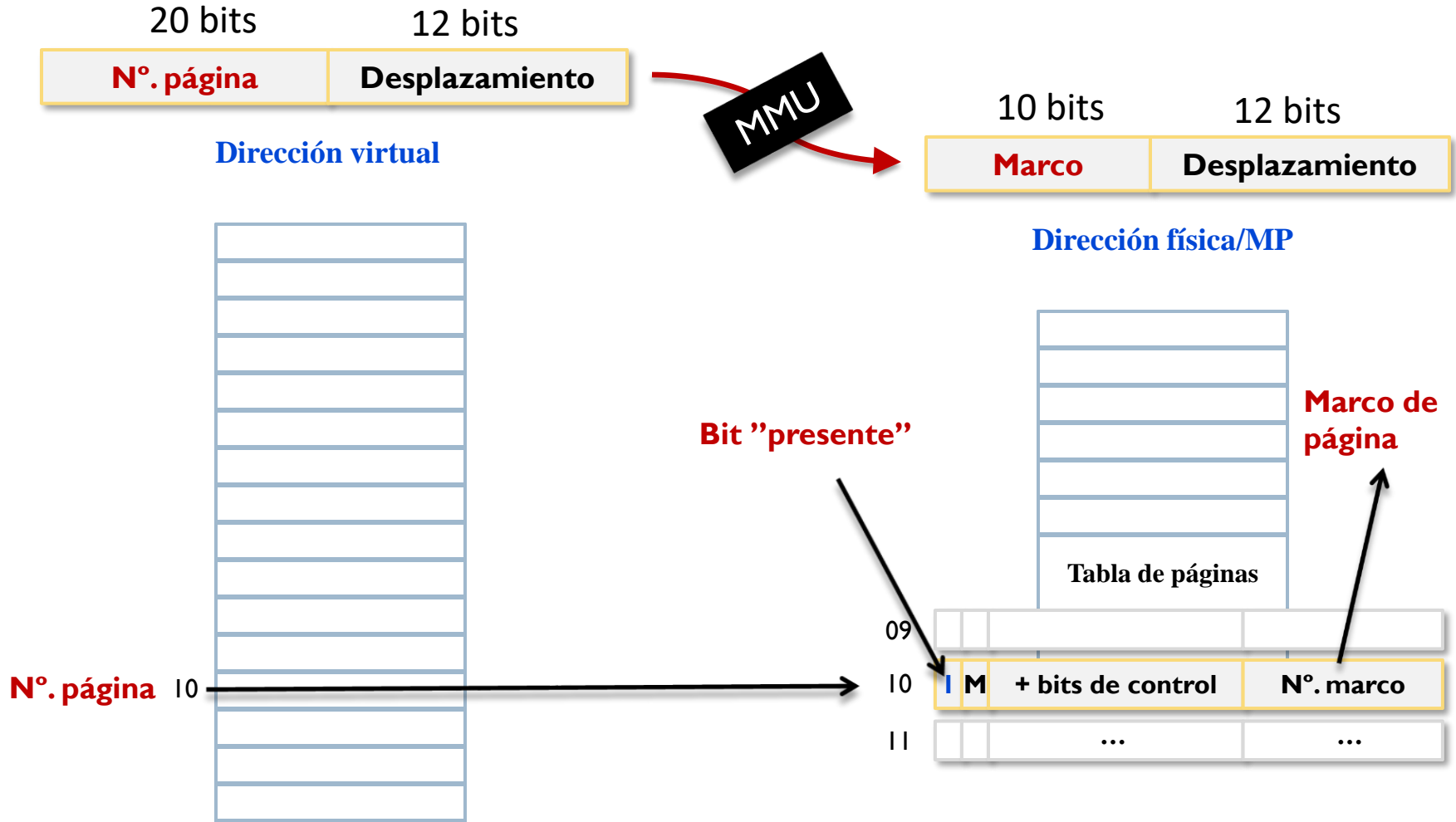
Correspondencia entre Id. página y marco -> T. páginas

# Ejemplo

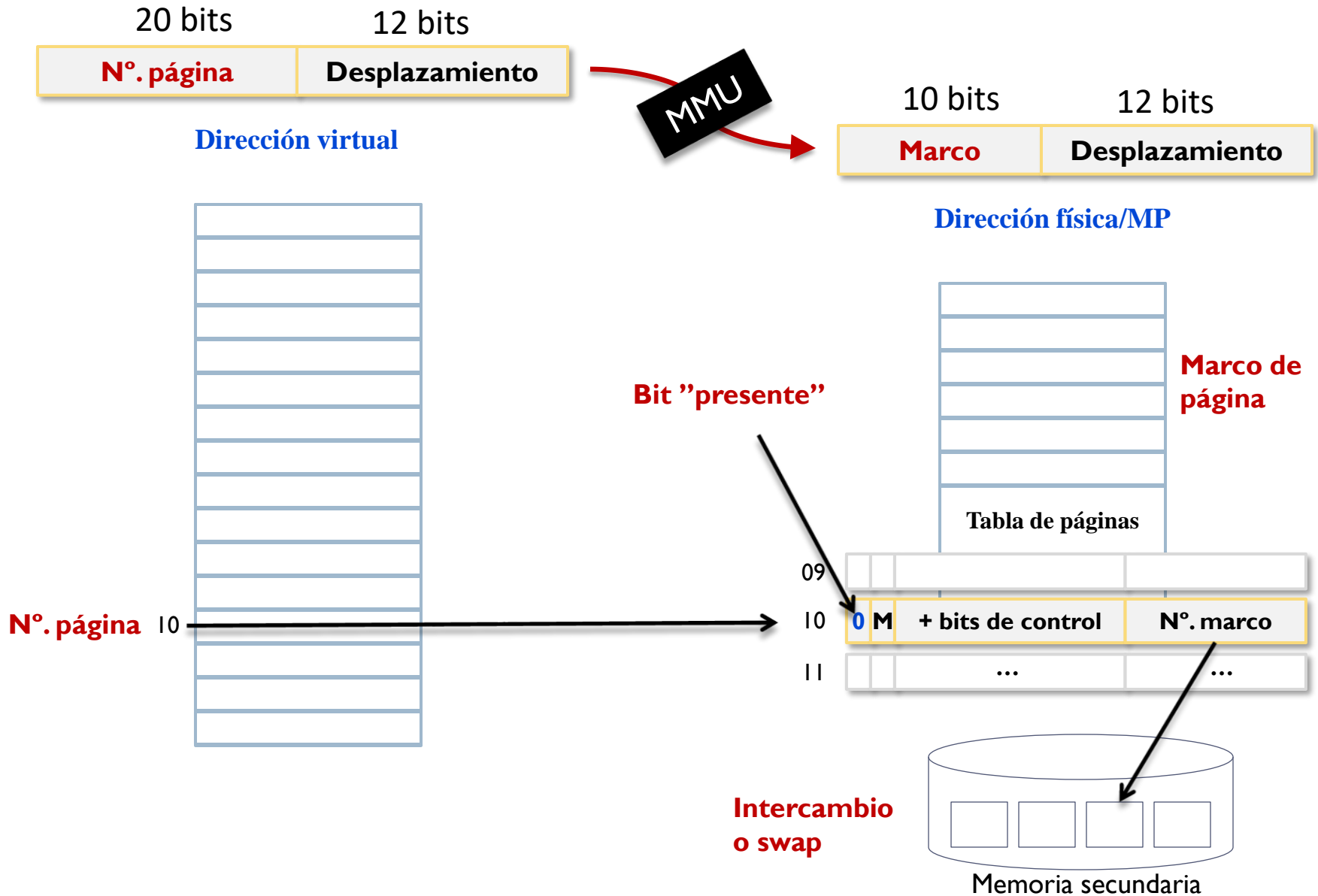


Correspondencia entre Nº. página y marco de página-> Tabla de páginas

# Ejemplo



## Ejemplo



# Estructura de una dirección virtual

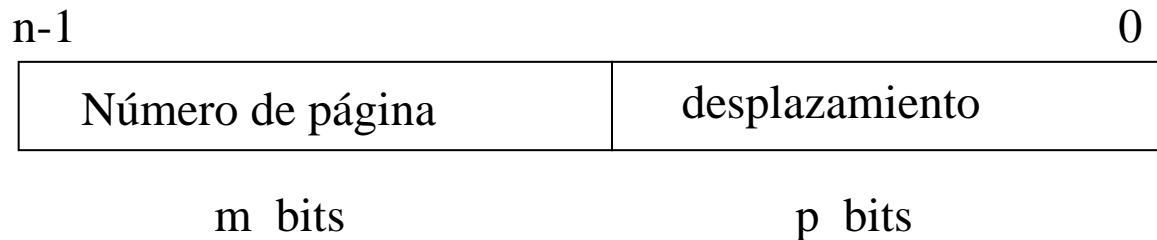
- ▶ Un computador de  $n$  bits tiene:
  - ▶ Direcciones de  $n$  bits



- ▶ Puede direccionar  $2^n$  bytes

# Estructura de una dirección virtual

- ▶ La imagen de memoria está compuesta por páginas de igual tamaño ( $2^p$  bytes)



- ▶  $n = m + p$
- ▶ Memoria direccionable:  $2^n$  bytes
- ▶ Tamaño de la página:  $2^p$  bytes
- ▶ Máximo número de páginas:  $2^m$  páginas

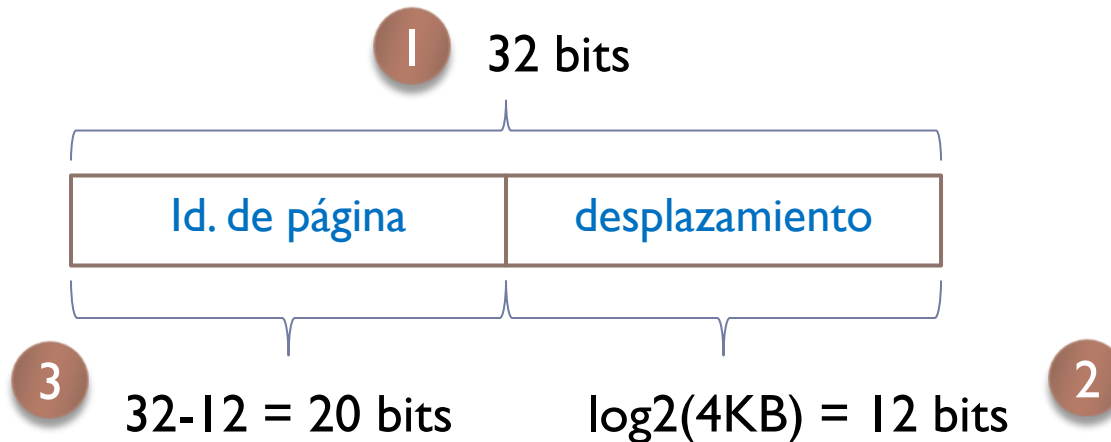
# Ejercicio

Sea un computador con direcciones virtuales de 32 bits y una memoria principal de 512 MB, que emplea páginas de 4 KB.

- ▶ Se pide:
  - a) Indique el formato de la dirección virtual y el número de marcos de página.

# Ejercicio (solución)

- Formato de la dirección virtual:



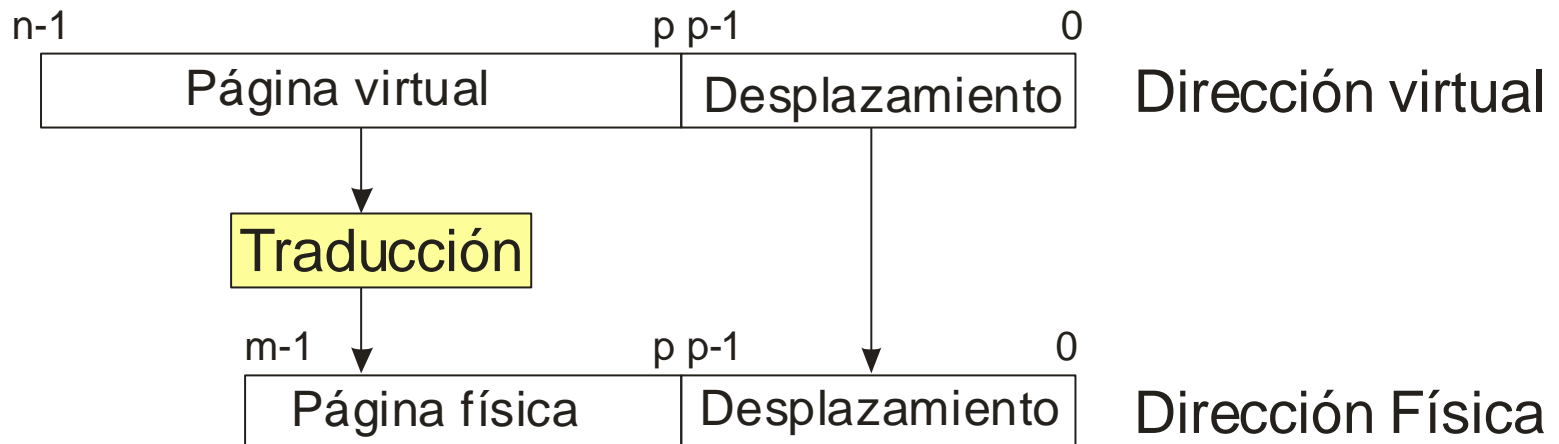
- Número de marcos de página:

$$\frac{\text{Tamaño de M.P.}}{\text{Tamaño de página}} = \frac{512 \text{ MB}}{4 \text{ KB}} = \frac{512 * 2^{20}}{4 * 2^{10}} = 128 * 2^{10}$$

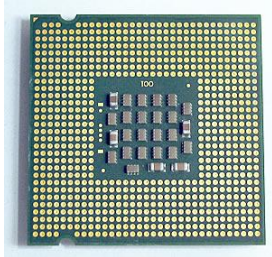


# Memoria virtual paginada

## Tablas de páginas



# Entradas de la tabla de páginas (formato típico)



**Dirección virtual**

20 bits

12 bits

**Número de página**

**Desplazamiento**

**Entrada de la tabla de páginas**

<b>P</b>	<b>M</b>	<b>Otros bits de control</b>	<b>Número de marco</b>

- Bit P: indica si está presente la página en M.P.
- Bit M: indica si ha sido modificada la página en M.P.
- Otros bits: protección (lectura, escritura, ejecución, etc.), gestión (cow, etc.)

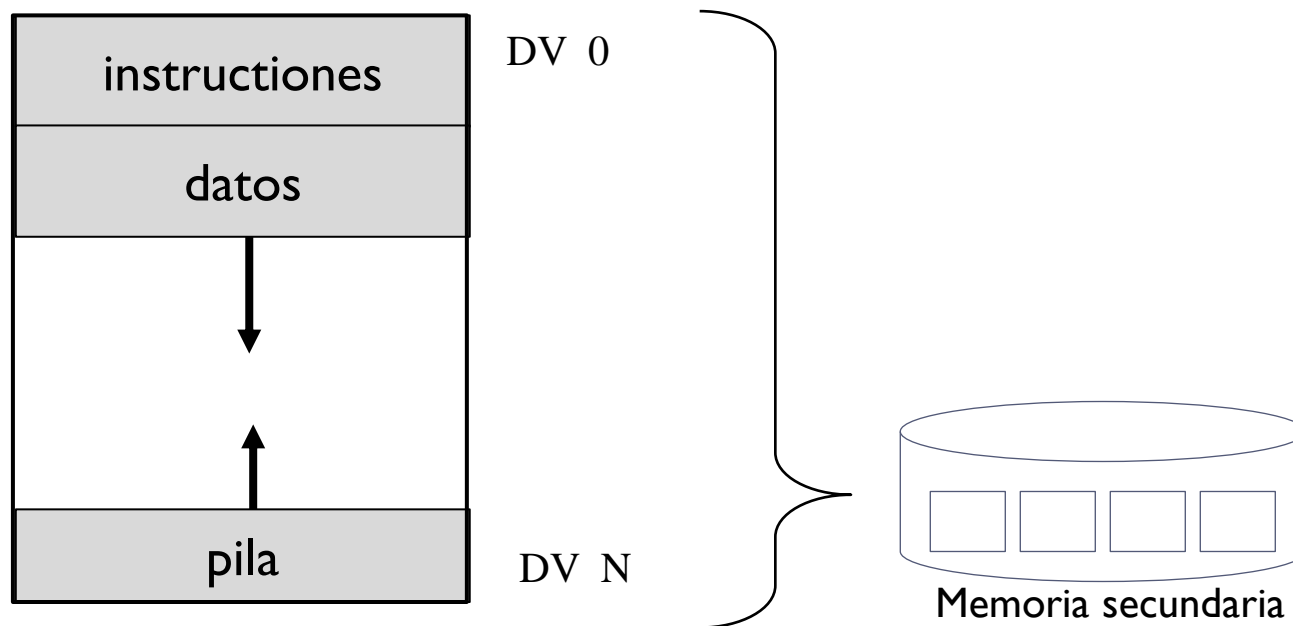
# Gestión de la tabla de páginas

CRUD (create, read, update, delete)

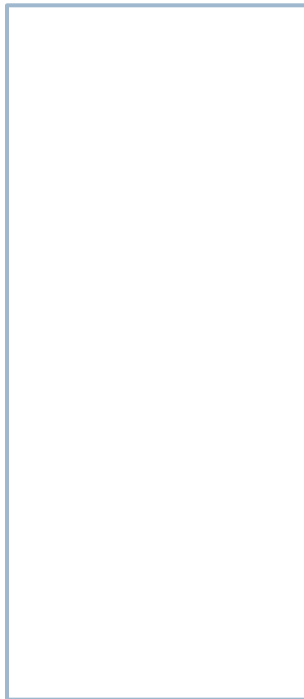
- ▶ Inicialmente:
  - ▶ La **crea** el sistema operativo cuando se va a ejecutar el programa.
- ▶ Uso:
  - ▶ La **consulta** la MMU en la traducción.
- ▶ Actualización:
  - ▶ La **modifica** el sistema operativo en los fallos de página.

# Memoria virtual paginada

- ▶ La imagen de memoria de los procesos reside inicialmente en disco



# Ejemplo



- ▶ Páginas de 1 KB
- ▶ Proceso de 8 KB
  - ▶ Número de páginas que ocupa: 8
- ▶ Tamaño de las secciones:
  - ▶ Instrucciones: 1.5 KB
  - ▶ Datos: 1 KB
  - ▶ Pila 0.2 KB

# Ejemplo

Instr.	Pag. 0
Instr.	Pag. 1
Datos	Pag. 2
	Pag. 3
	Pag. 4
	Pag. 5
	Pag. 6
Pila	Pag. 7

- ▶ Páginas de 1 KB
- ▶ Proceso de 8 KB
  - ▶ Número de páginas que ocupa: 8
- ▶ Tamaño de las secciones:
  - ▶ Instrucciones: 1.5 KB -> 2 páginas
  - ▶ Datos: 1 KB -> 1 página
  - ▶ Pila 0.2 KB -> 1 página

# Ejemplo

Instr.	Pag. 0
Instr.	Pag. 1
Datos	Pag. 2
	Pag. 3
	Pag. 4
	Pag. 5
	Pag. 6
Pila	Pag. 7

- ▶ DV de inicio: 0
- ▶ DV final: 8191
- ▶ Pags. 3, 4, 5 y 6 no asignadas inicialmente al programa

# Ejemplo

## Imagen inicialmente en disco

Instr.	Pag. 0
Instr.	Pag. 1
Datos	Pag. 2
	Pag. 3
	Pag. 4
	Pag. 5
	Pag. 6
Pila	Pag. 7

0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

# Swap

Páginas del proceso



# Ejemplo

## El SO crea la tabla de páginas

Instr.	Pag. 0
Instr.	Pag. 1
Datos	Pag. 2
	Pag. 3
	Pag. 4
	Pag. 5
	Pag. 6
Pila	Pag. 7

	P	M	marco/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Todas las páginas  
Inicialmente en swap

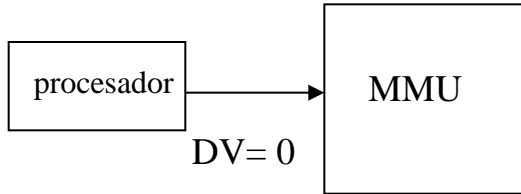
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

# Swap

Páginas del proceso

# Ejemplo

## Acceso a la DV 0



	P	M	marco/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Instr.	Pag. 0
Instr.	Pag. 1
Datos	Pag. 2
	Pag. 3
	Pag. 4
	Pag. 5
	Pag. 6
Pila	Pag. 7

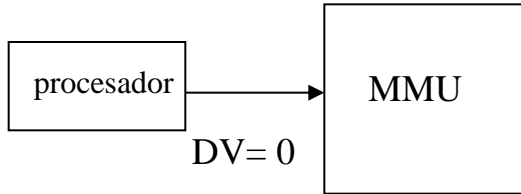
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

# Swap

Páginas del proceso

# Ejemplo

## Acceso a la DV 0



	P	M	marco/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

DV=0	0	0
	NP	D

Instr.	Pag. 0
Instr.	Pag. 1
Datos	Pag. 2
	Pag. 3
	Pag. 4
	Pag. 5
	Pag. 6
Pila	Pag. 7

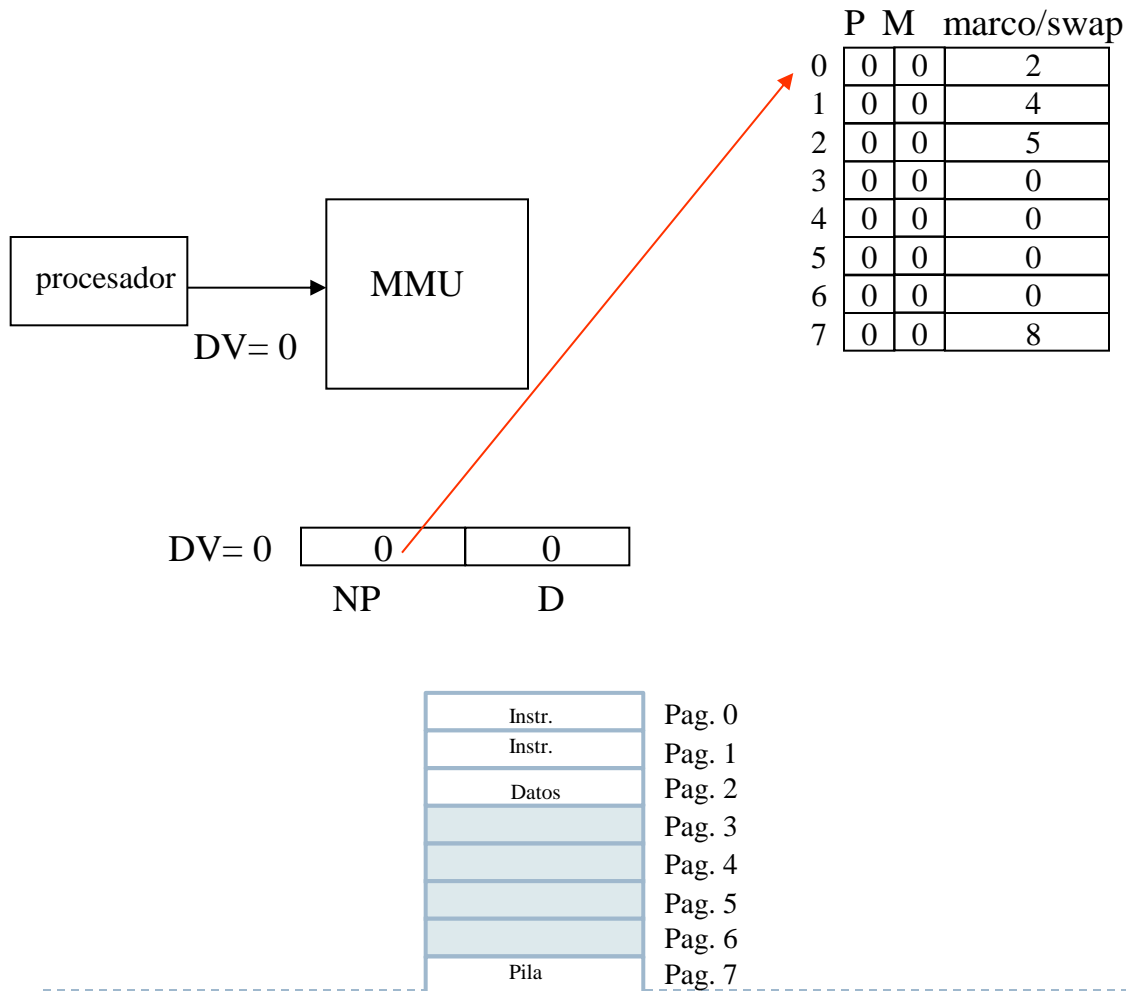
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

# Swap

Páginas del proceso

# Ejemplo

## Acceso a la DV 0



Fallo de página  
La página 0 no está en memoria

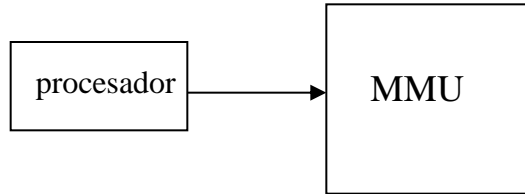
# Swap

Páginas del proceso

0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

# Ejemplo

## Tratamiento del fallo de página

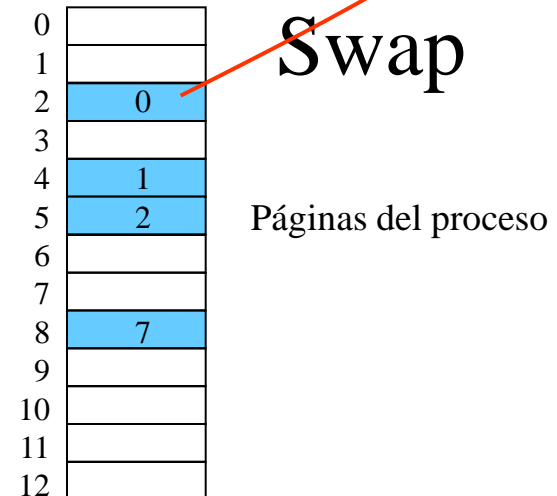
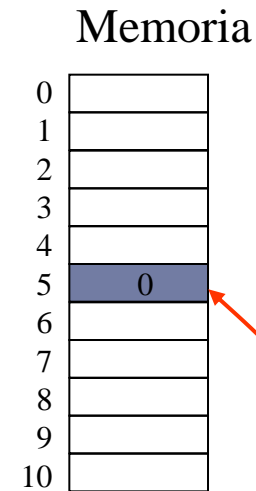


DV= 0    

0	0
---	---

NP                  D

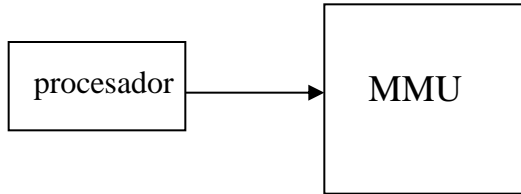
	P	M	marco/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8



El SO reserva un marco de página libre en memoria ( el 5) y copia el bloque 2 al marco 5

# Ejemplo

## Tratamiento del fallo de página



DV= 0    

0	0
---	---

  
              NP                  D

El SO actualiza la tabla de páginas

	P	M	marco/swap
0	1	0	5
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Memoria

0	
1	
2	
3	
4	
5	0
6	
7	
8	
9	
10	

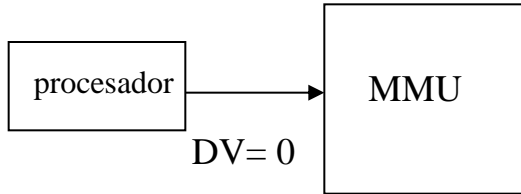
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

## Swap

Páginas del proceso

# Ejemplo

## Reanudación del proceso



	P	M	marco/swap
0	1	0	5
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Memoria

0	
1	
2	
3	
4	
5	0
6	
7	
8	
9	
10	

DV=0

0	0
NP	D

Se vuelve a genera la DV 0

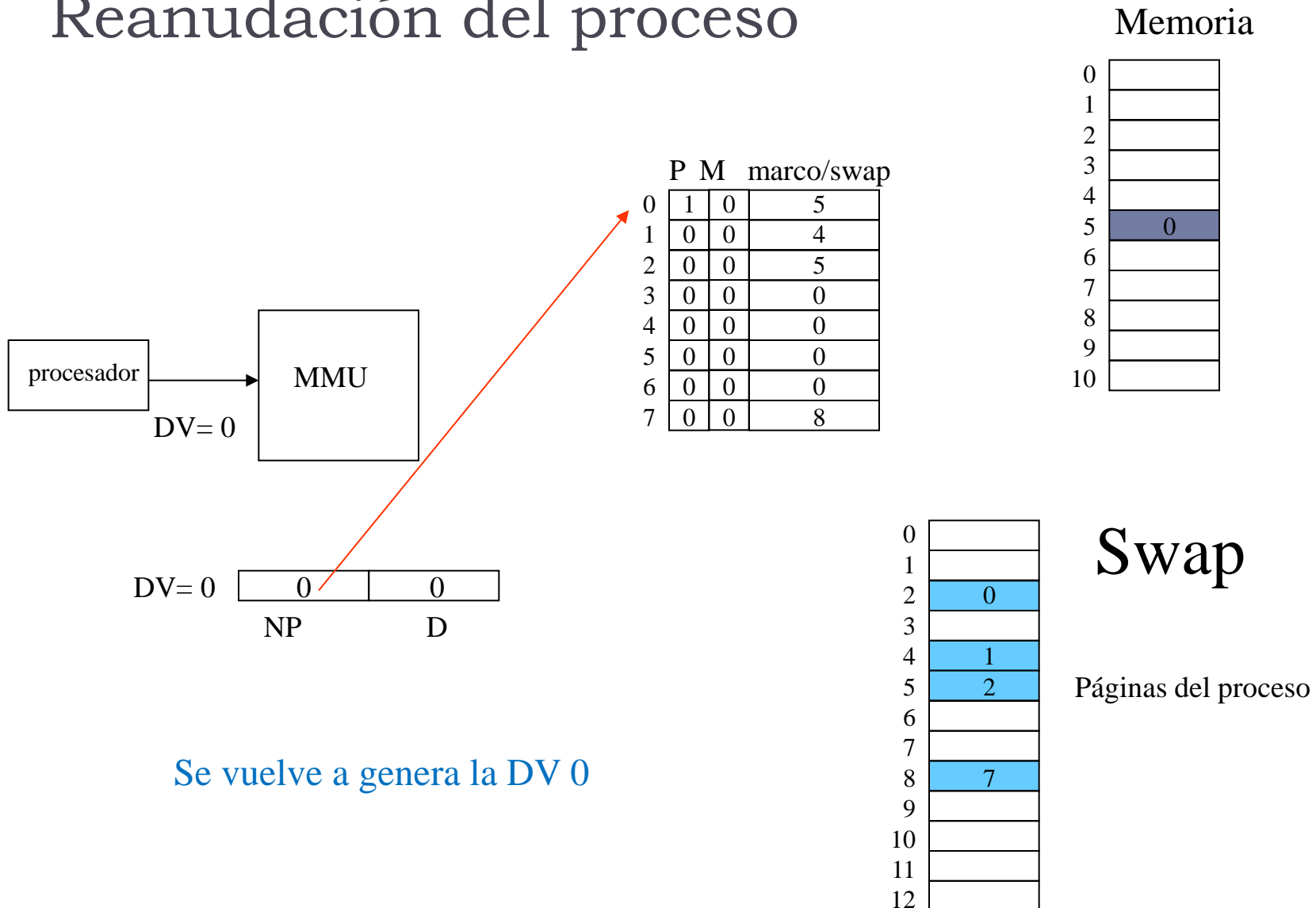
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

Swap

Páginas del proceso

# Ejemplo

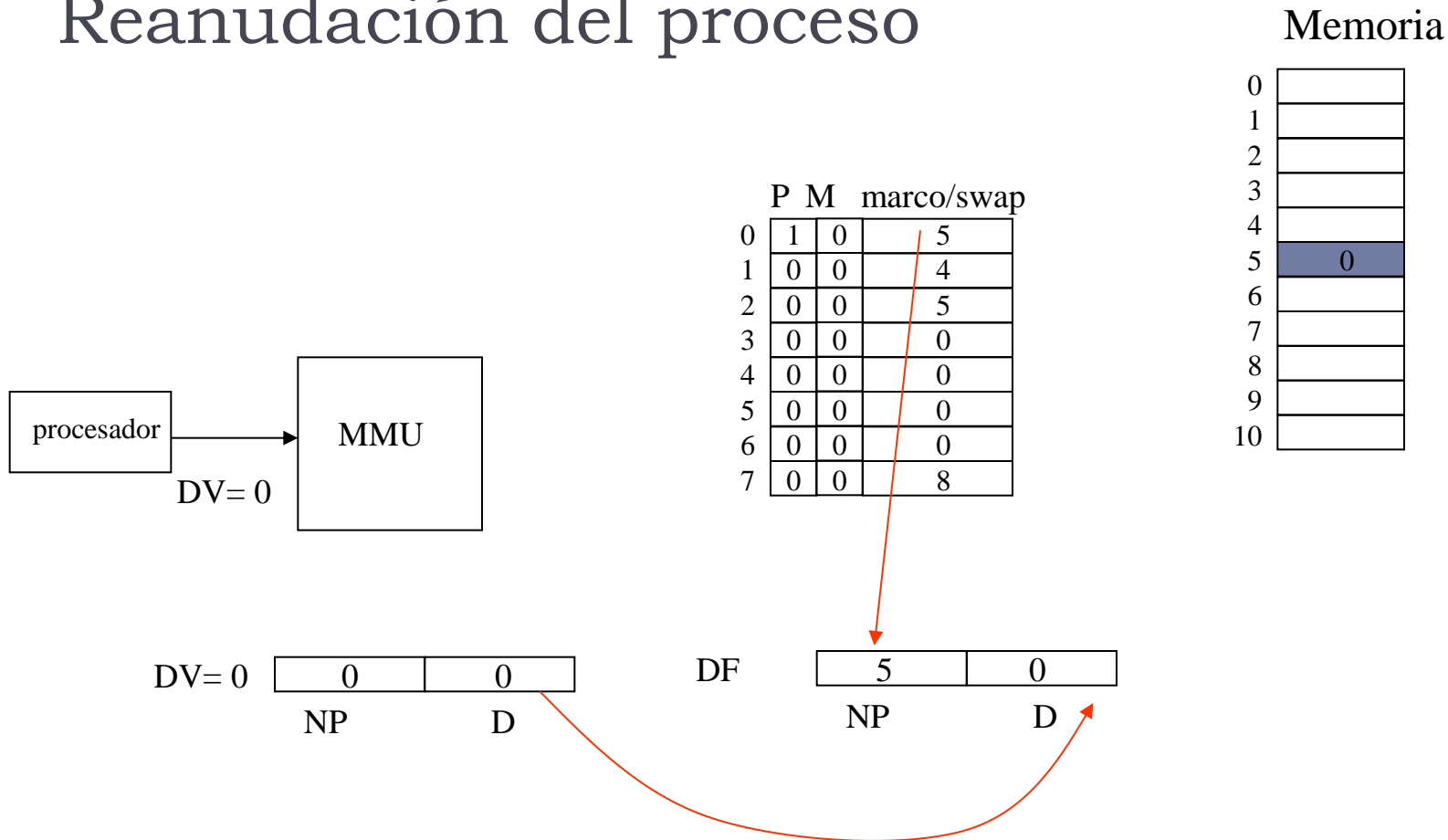
## Reanudación del proceso





# Ejemplo

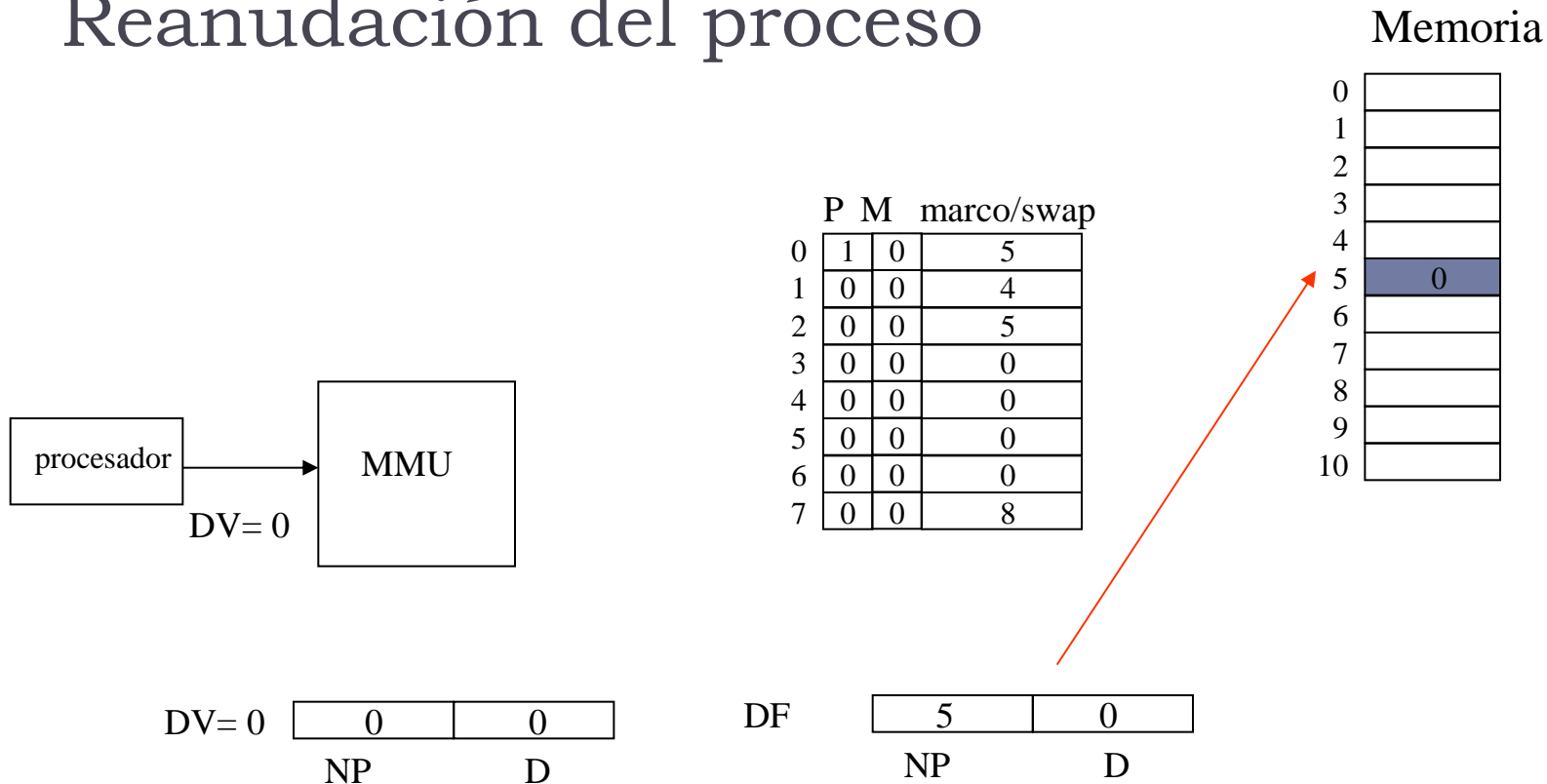
## Reanudación del proceso



Página presente  
Se genera la DF

# Ejemplo

## Reanudación del proceso



Se accede a memoria

# Ejercicio

Un computador que direcciona la memoria por byte emplea direcciones virtuales de 32 bits.

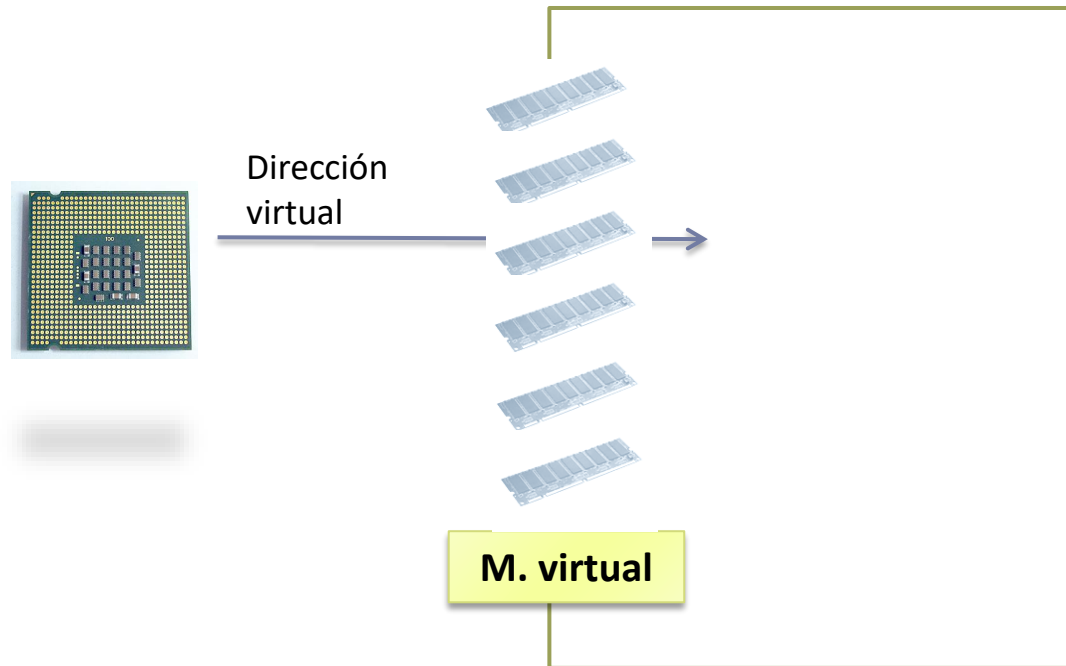
Cada entrada de la tabla de páginas requiere de 32 bits.

El sistema emplea páginas de 4 KB.

- ▶ Se pide:
  - a) ¿Cuál es el espacio de memoria direccionable por un programa en ejecución?
  - b) ¿Cuál es el máximo tamaño de la tabla de páginas en este computador?

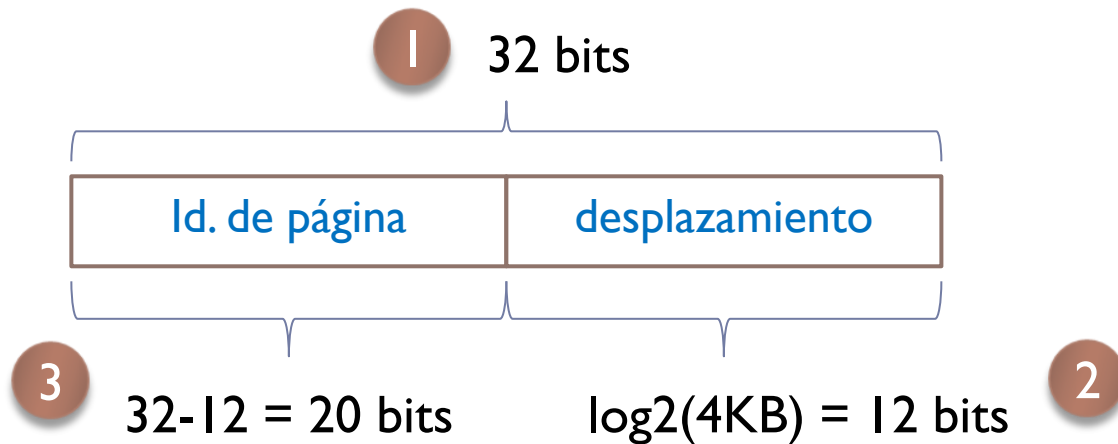
# Ejercicio (solución)

- ▶ El espacio de memoria direccionable por un programa en ejecución está determinado por el número de bits de la dirección virtual:
  - ▶  $2^{32} = 4 \text{ GB}$



# Ejercicio (solución)

- ▶ El tamaño de la tabla de páginas dependerá del máximo número de marcos de páginas y del tamaño de cada entrada de la tabla:
  - ▶  $2^{20} * 4 \text{ bytes (32 bits)} = 4 \text{ MB}$



- 4 Si hay tanta memoria principal como memoria virtual, los identificadores de marco de página tendrán también 20 bits

# Ejercicio

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	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

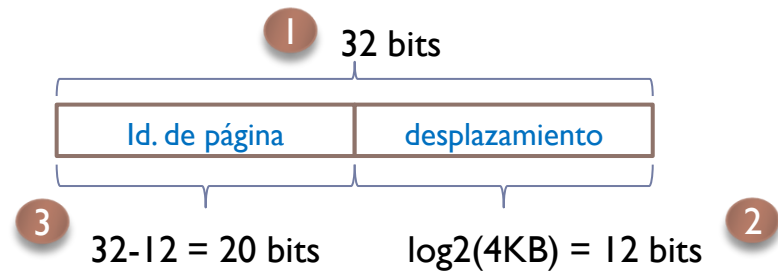
- ▶ Si el tamaño total del programa es de 28 KB y la primera dirección virtual es la 0x00000000, la última dirección será:
  - ▶  $28 * 1024 - 1$



# Ejercicio (solución)

P	M	Perm.	Marco/ Bloque
0	0	R	1036
<b>1</b>	<b>0</b>	<b>R</b>	<b>4097</b>
0	0	W	3000
0	0	W	7190
<b>0</b>	<b>0</b>	<b>W</b>	<b>3200</b>
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**0000****1**000 -> no
  - 0x**0000****1**01C -> no
  - 0x**0000****4**000 -> si

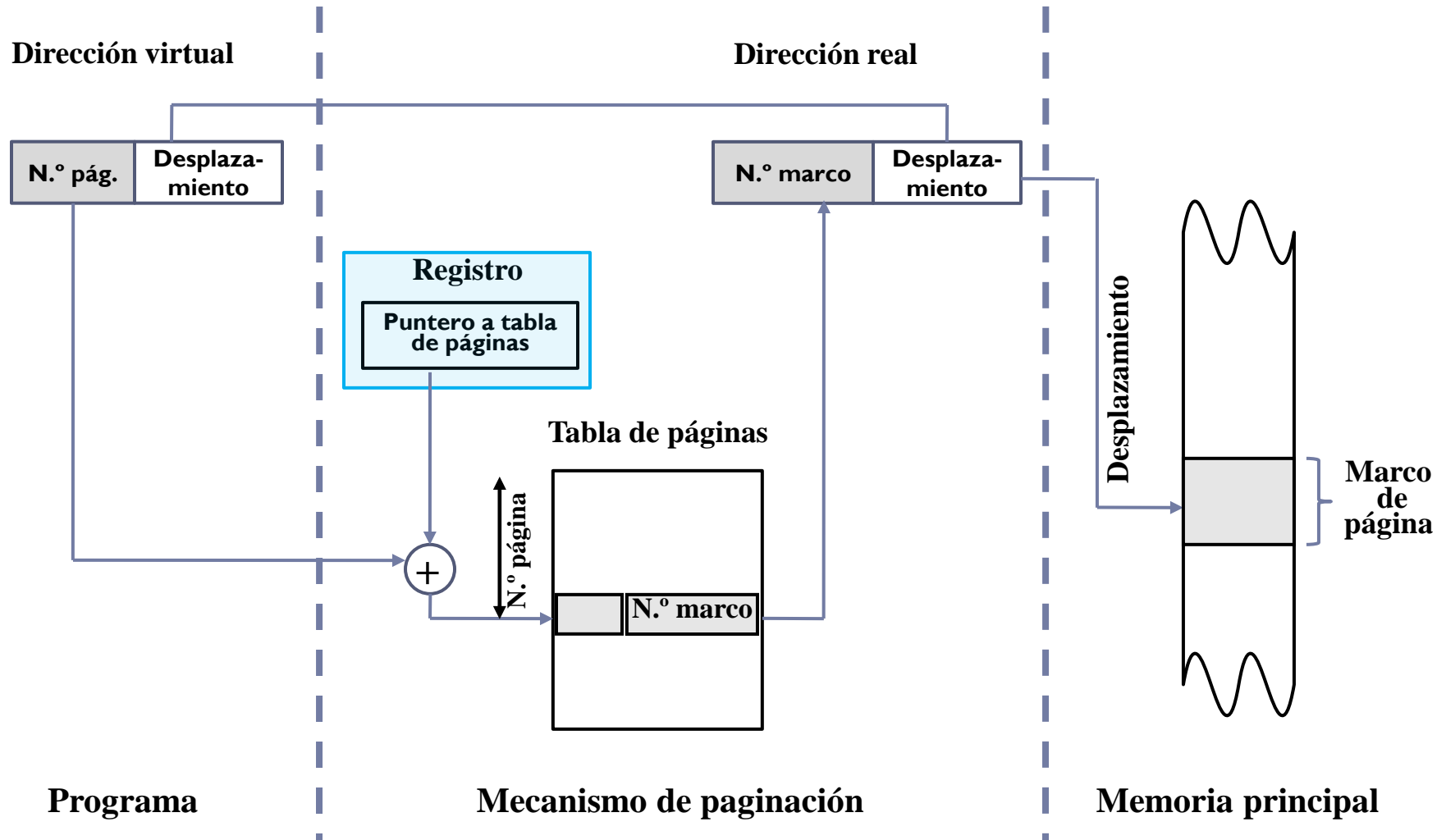
# Gestión de la tabla de páginas

CRUD (create, read, update, delete)

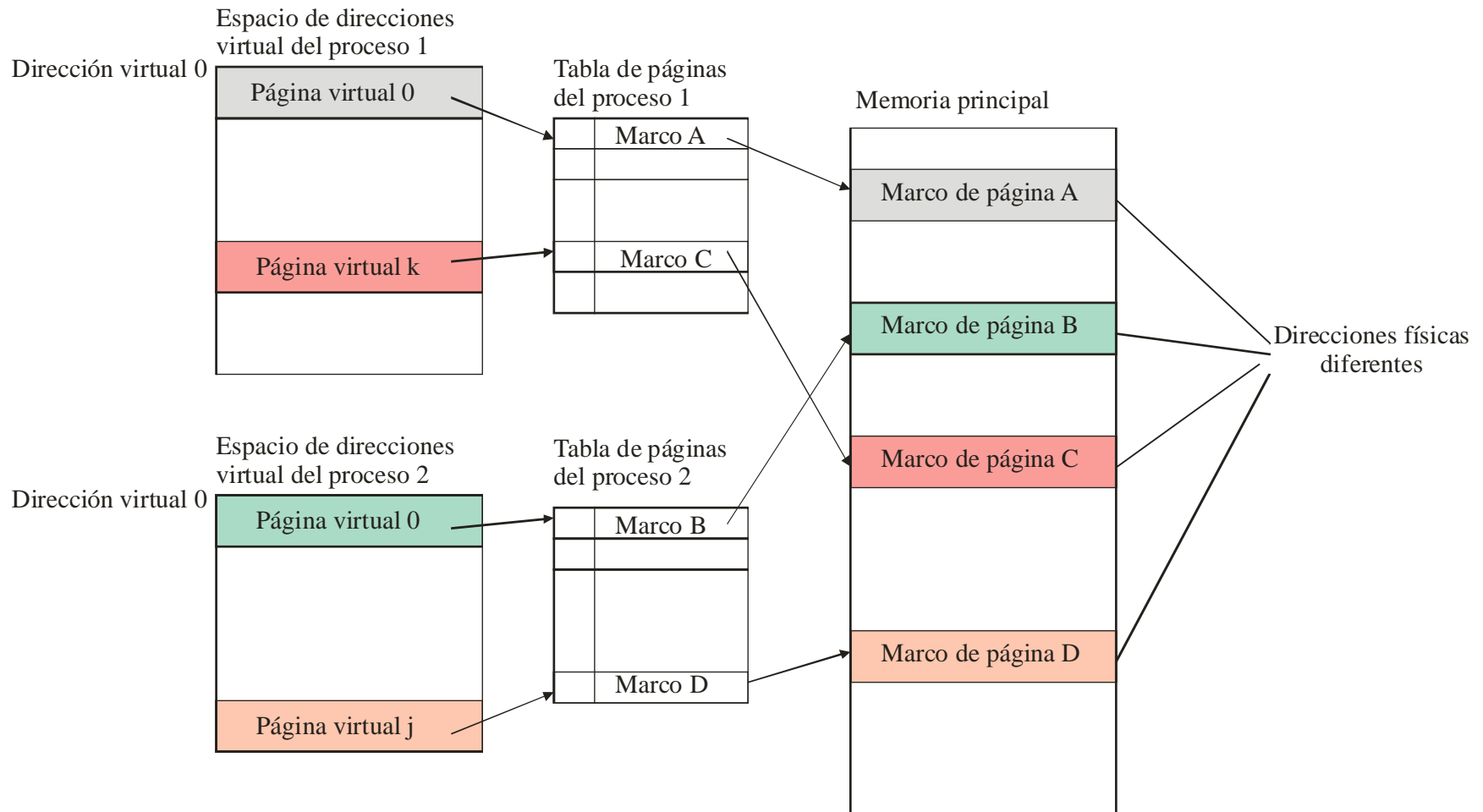


- ▶ Inicialmente:
  - ▶ La **crea** el sistema operativo cuando se va a ejecutar el programa.
- ▶ Uso:
  - ▶ La **consulta** la MMU en la traducción.
- ▶ Actualización:
  - ▶ La **modifica** el sistema operativo en los fallos de página.

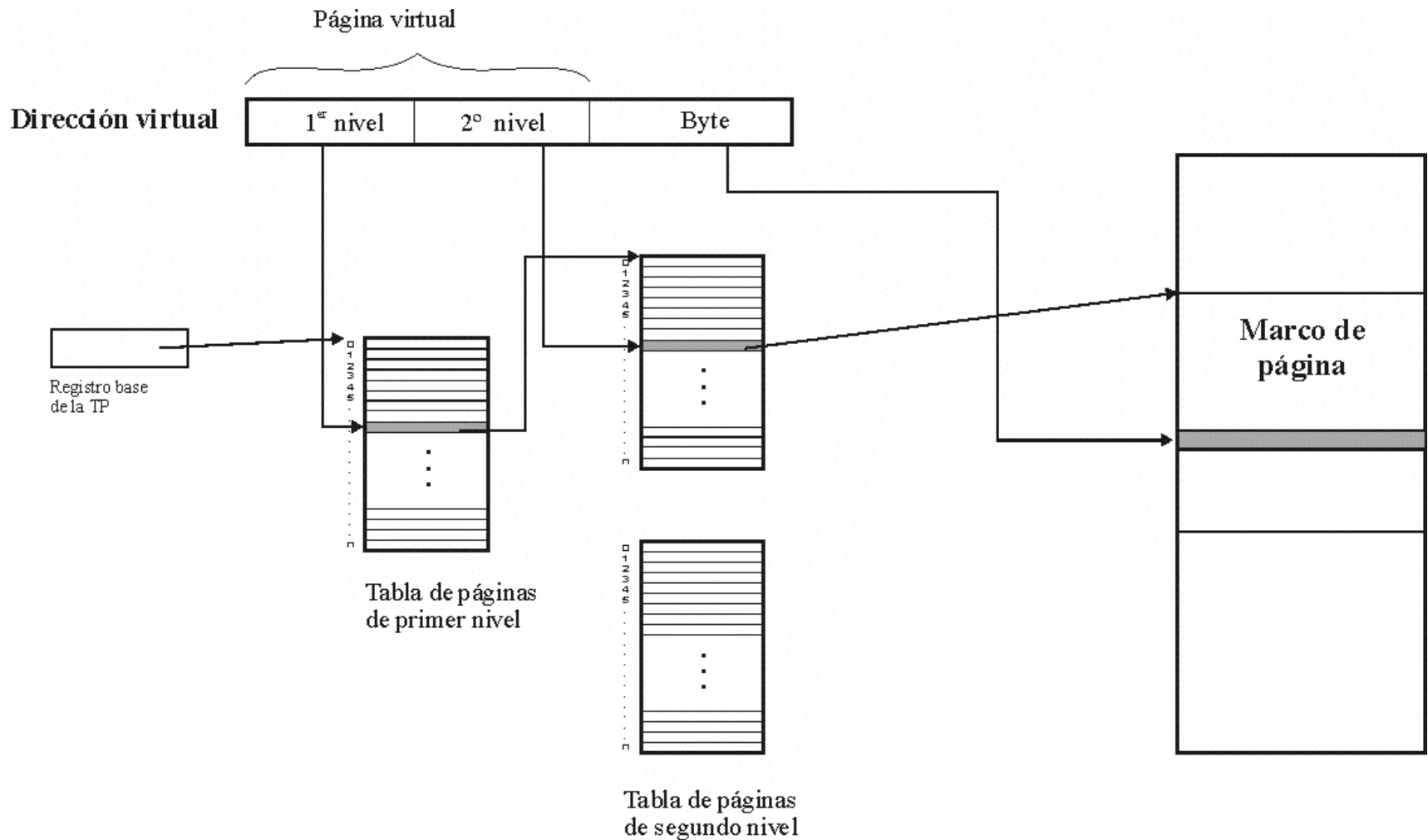
# Traducción de direcciones (paginación)



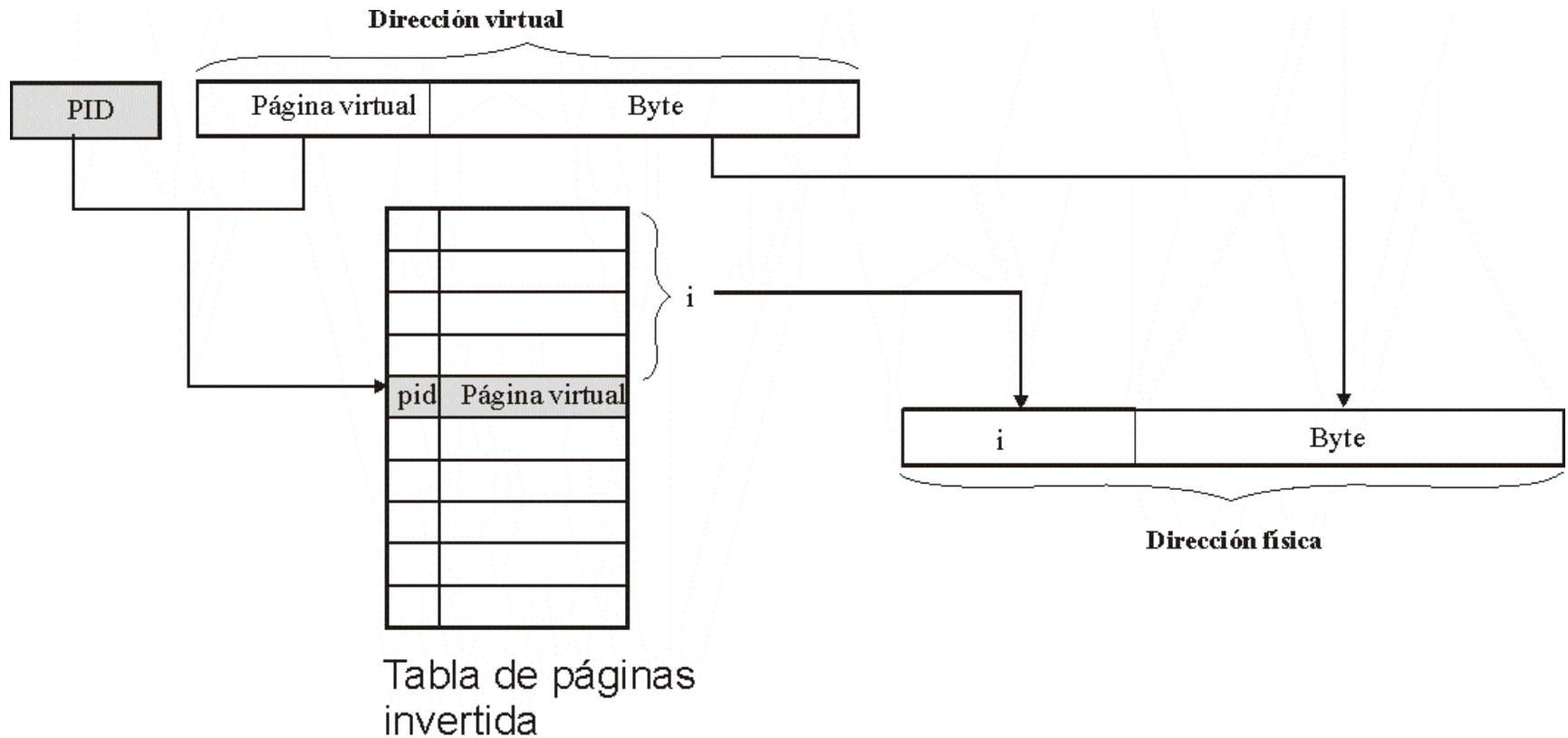
# Protección de memoria



# Tabla de páginas de dos niveles



# Tabla de páginas invertida



# Movimiento de las páginas

- ▶ **Inicialmente:**
  - ▶ Página no residente se marca ausente
  - ▶ Se guarda dirección del bloque de *swap* que la contiene
- ▶ **De M. secundaria a M. principal (por demanda):**
  - ▶ Acceso a pág. no residente: Fallo de página
  - ▶ S.O. lee página de M. secundaria y la lleva a M. principal
- ▶ **De M. principal a M. secundaria (por expulsión):**
  - ▶ No hay espacio en M. principal para traer página
  - ▶ Se expulsa (reemplaza) una página residente
  - ▶ S.O. escribe página expulsada a M. secundaria (si bit  $M=1$ )

# Movimiento de las páginas

## ► Inicialmente:

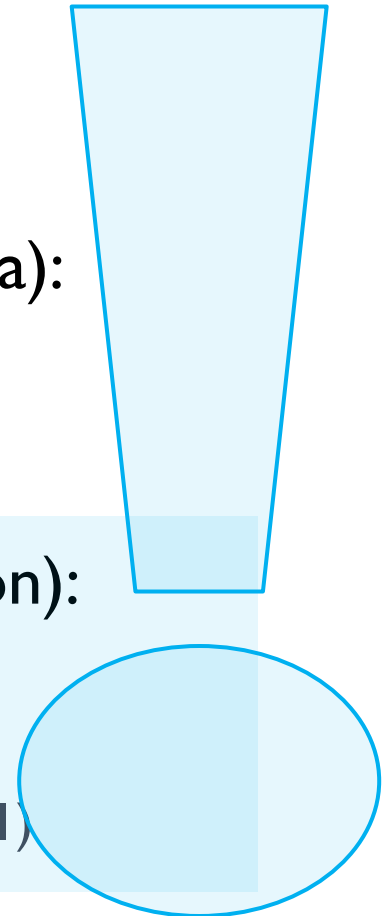
- Página no residente se marca ausente
- Se guarda dirección del bloque de *swap* que la contiene

## ► De M. secundaria a M. principal (por demanda):

- Acceso a pág. no residente: Fallo de página
- S.O. lee página de M. secundaria y la lleva a M. principal

## ► De M. principal a M. secundaria (por expulsión):

- No hay espacio en M. principal para traer página
- Se expulsa (reemplaza) una página residente
- S.O. escribe página expulsada a M. secundaria (si bit  $M=1$ )





# Políticas de reemplazo

- ▶ Qué página se va a reemplazar (sistema operativo)
- ▶ La página que se va a reemplazar tiene que ser la que tenga una menor posibilidad de ser referenciada en un futuro cercano.
- ▶ La mayoría de las políticas intentan predecir el comportamiento futuro en función del comportamiento pasado.
- ▶ Ejemplo de políticas: **LRU, FIFO, etc.**

# Políticas de **no** reemplazo

- ▶ Bloqueo de marcos:
  - ▶ 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.
- ▶ **El bloqueo se consigue asociando un bit de bloqueo a cada marco.**



# Cache de traducciones

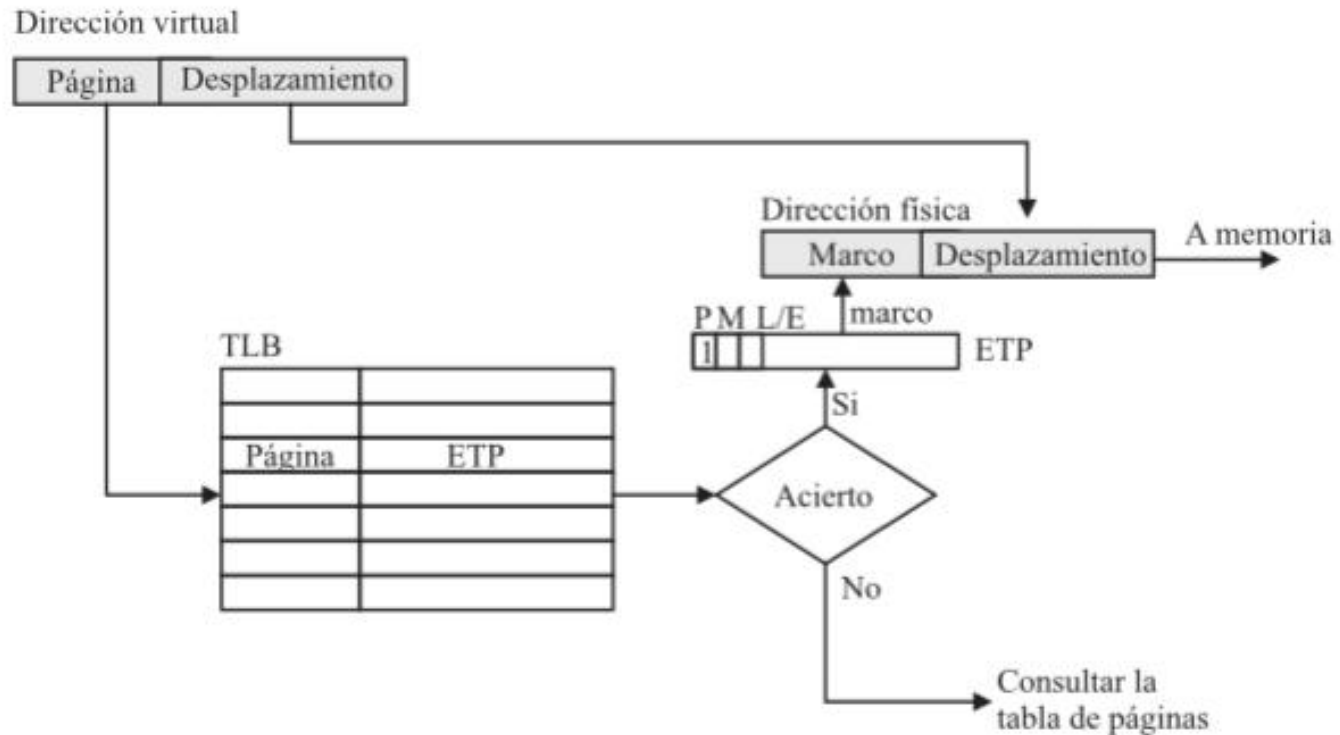
## TLB (*Translation Lookaside Buffer*)

- ▶ Memoria virtual basado en tablas de páginas:
  - ▶ Problema: sobrecarga de acceso a memoria (2 accesos)
    - ▶ Uno a la tabla de páginas que reside en MP
    - ▶ Otro a la página que contiene el dato
  - ▶ Solución: **TLB**.
- ▶ **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.

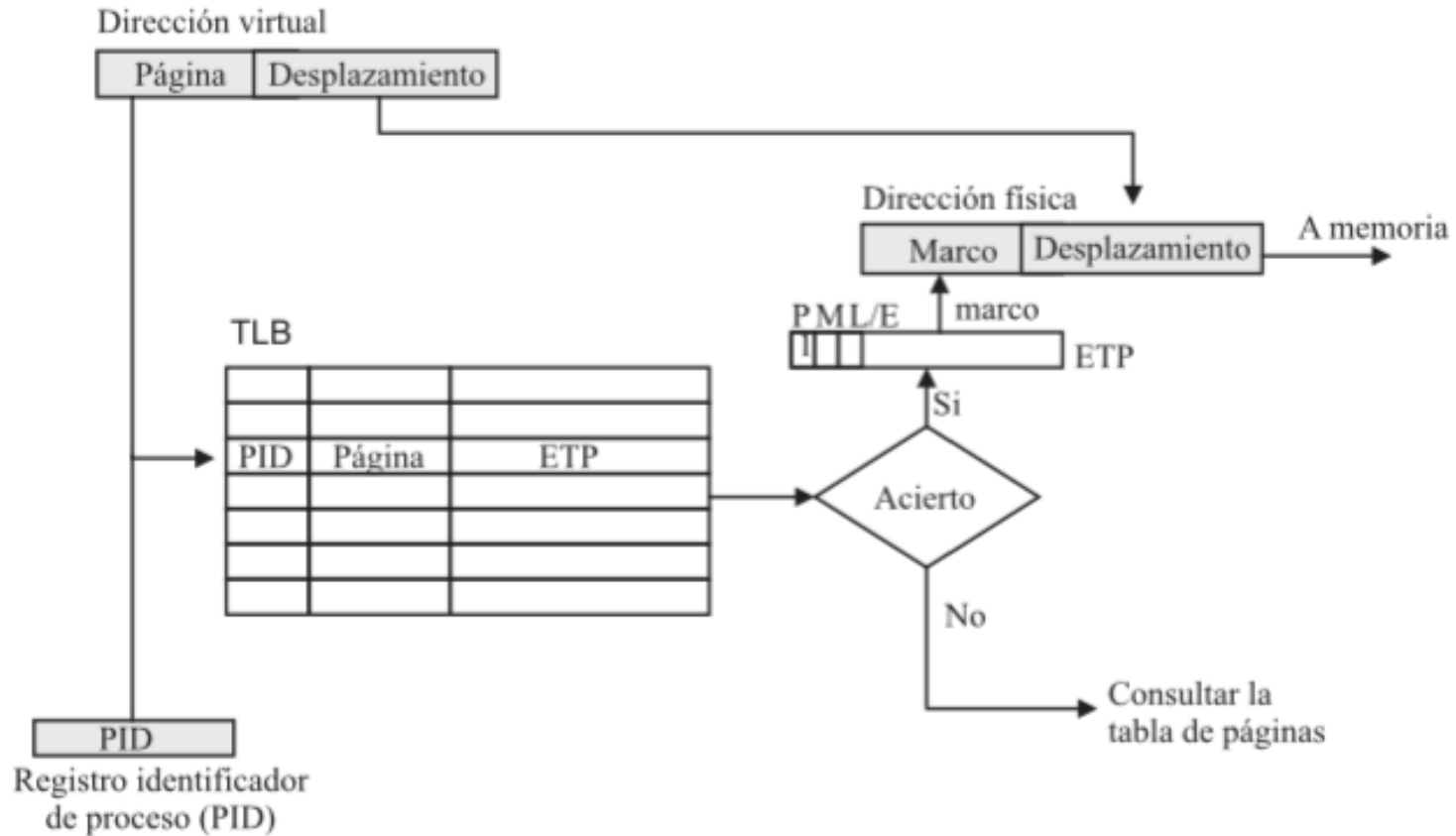
# TLB (Translation Lookaside Buffer)

- ▶ **La TLB optimiza los accesos a memoria**
  - ▶ Tabla con tiempo de accesos pequeño situado en la MMU
  - ▶ Cada entrada contiene un número de página y la entrada de la TB correspondiente
    - ▶ En caso de acierto no hace falta acceder a la TP en memoria
- ▶ **Dos tipos:**
  - ▶ TLB sin identificación de proceso
  - ▶ TLB con identificación de proceso

# TLB sin identificación de proceso



# TLB con identificación de proceso



# Caché y memoria virtual

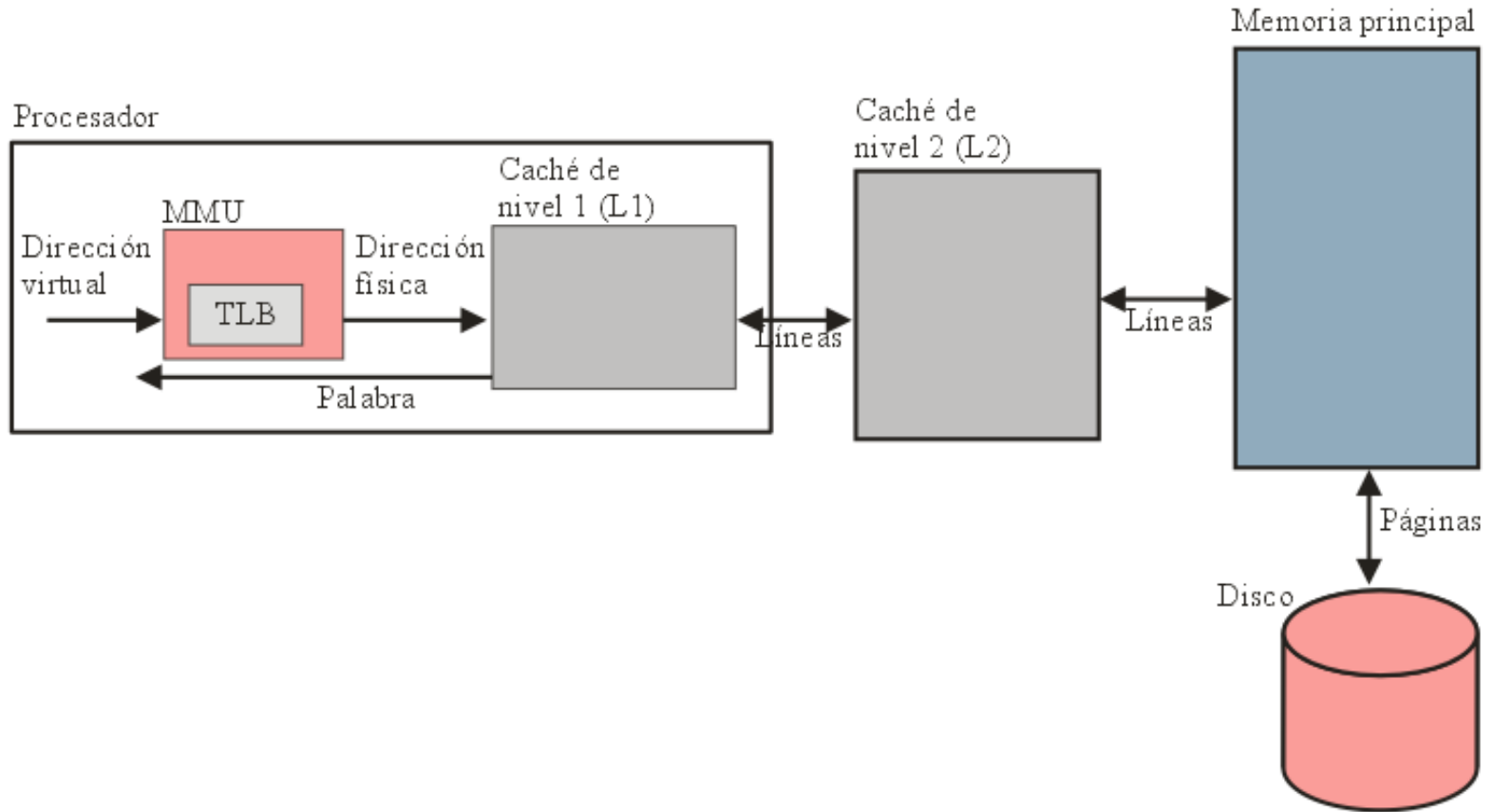
## Caché

- ▶ Acelerar el acceso
- ▶ Transferencia por bloques o líneas.
- ▶ Bloques: 32-64B.
- ▶ Traducción: Algoritmo de correspondencia.
- ▶ Escritura inmediata o diferida.

## Memoria virtual

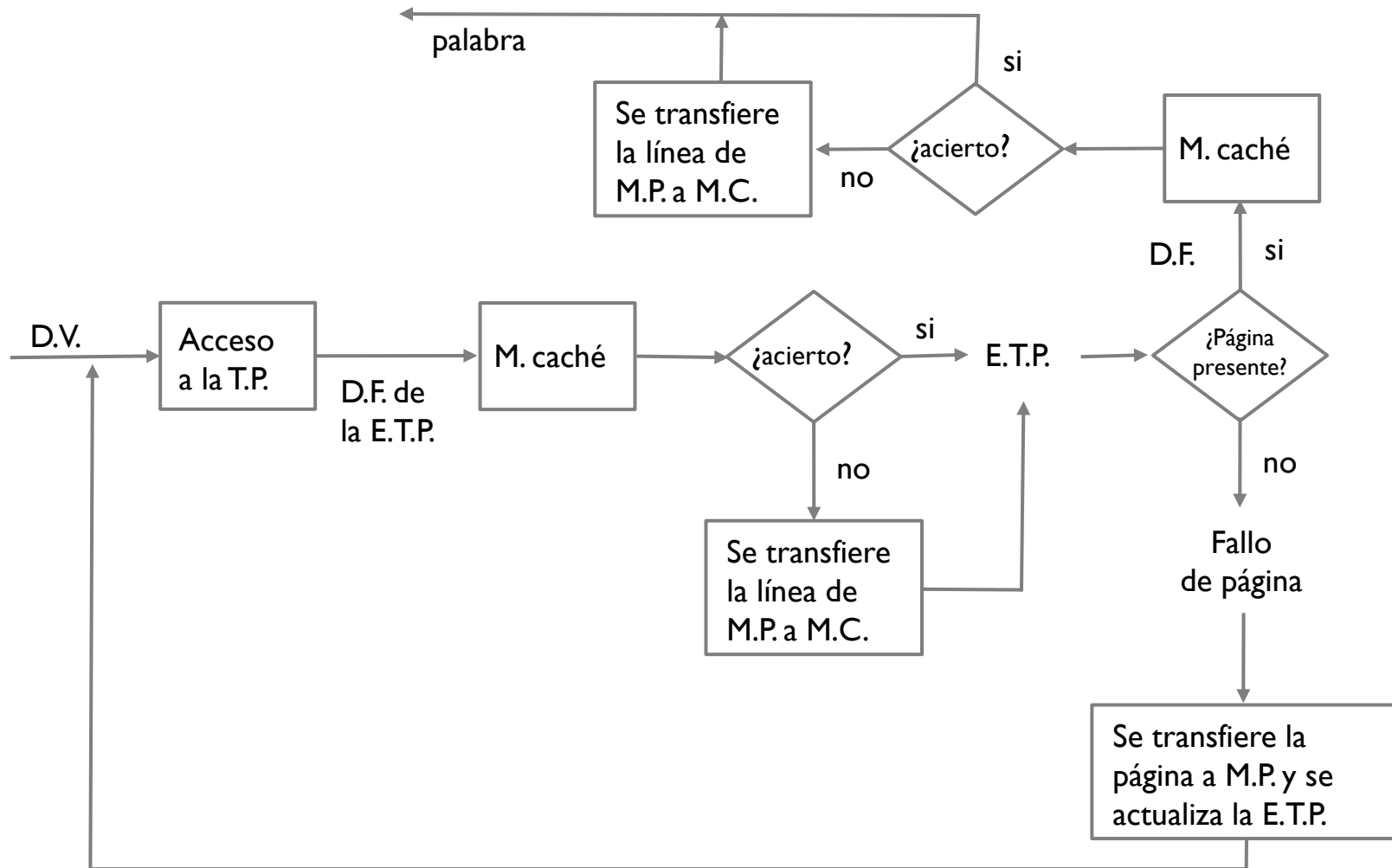
- ▶ Incrementar el espacio direccionable
- ▶ Transferencia por páginas.
- ▶ Páginas: 4-8 KB.
- ▶ Traducción: Totalmente asociativa.
- ▶ Escritura diferida.

# Memoria virtual y memoria caché





# Proceso de lectura en un sistema con memoria virtual y caché



Grupo ARCOS

**uc3m** | Universidad **Carlos III** de Madrid

# Tema 5: Jerarquía de memoria (III)

## **Estructura de Computadores**

Grado en Ingeniería Informática  
Grado en Matemática aplicada y Computación  
Doble Grado en Ingeniería Informática y Administración de Empresas



# Ejercicio 1

- ▶ Sea un computador que utiliza páginas de 8 KB y que direcciona la memoria por bytes. Dada la dirección virtual (en hexadecimal) 0x20018004. Indique:
  - ▶ El tamaño de la dirección virtual.
  - ▶ El número máximo de páginas.
  - ▶ El número de página en el que se encuentra el dato referenciado por la dirección anterior.
  - ▶ El desplazamiento dentro de la página en el que se encuentra el dato referenciado por la dirección anterior.

## Ejercicio 2

- ▶ Un computador que direcciona la memoria por bytes emplea direcciones virtuales de 32 bits. Cada entrada de la tabla de páginas requiere 32 bits. El sistema emplea páginas de 4 KB.
  - ▶ ¿Cuál es el espacio de memoria direccionable por un programa en ejecución?
  - ▶ ¿Cuál es el máximo tamaño de la tabla de páginas en este computador?

# Ejercicio 3

- ▶ Sea un sistema con un espacio de direcciones virtual de 256 Kpáginas de 8 KB cada una y una memoria física de 128 MB.
  - ▶ ¿Cuántos bits hay en la dirección virtual?

# Ejercicio 4

- ▶ Si un computador trabaja con direcciones de 16 bits, y posee páginas de tamaño 2 KB. Se pide:
  - ▶ ¿Qué tamaño de memoria virtual se puede direccionar?
  - ▶ ¿Cuántas páginas tiene la memoria virtual?
  - ▶ ¿Cuál será el tamaño del marco de página?
  - ▶ ¿Suponiendo que la memoria física es de 32 KB, cuántos marcos hay?
  - ▶ ¿Cuántos bits de la dirección de memoria virtual se utilizan para seleccionar entradas en la tabla de páginas?
  - ▶ ¿Para que se emplean los bits restantes de la dirección de memoria virtual?
  - ▶ ¿Cuántas entradas tendrá la tabla de páginas?

# Ejercicio 5

- Dado un hipotético computador con **memoria virtual paginada** con un **espacio de direcciones virtuales de 64 KB**, una **memoria física de 16 KB**. En este computador, que **direcciona la memoria por bytes**, el **número de páginas por proceso** es como **máximo de 512**. En un instante de tiempo dado, la tabla de páginas del proceso en ejecución contiene la siguiente información:

# Ejercicio 5 (cont.)

- ▶ Se pide:
  - ▶ Calcule el tamaño de cada página y el número de marcos de página.
  - ▶ ¿Cuántas páginas tiene asignadas el proceso en ejecución?
  - ▶ ¿Para qué se utiliza el bit M?
  - ▶ Indique el formato de las direcciones virtuales especificando el tamaño de los campos y el significado de cada uno.
  - ▶ ¿Cuántos marcos de página tiene la memoria?
  - ▶ Indique las direcciones físicas, en binario y hexadecimal, correspondientes a las direcciones virtuales 258 y 1224 expresadas ambas en decimal.
  - ▶ ¿Dada una dirección virtual cuántos accesos a memoria física se requieren para obtener el dato?

## P M marco/bloque

1	0	000010
1	0	000001
1	0	000110
1	1	000000
1	0	000100
1	0	000011
0	0	000100
0	0	000010
0	0	000110
1	0	000101
0	0	000000
1	0	000111
0	0	000011
0	0	000101
0	0	000001
1	1	000111