

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

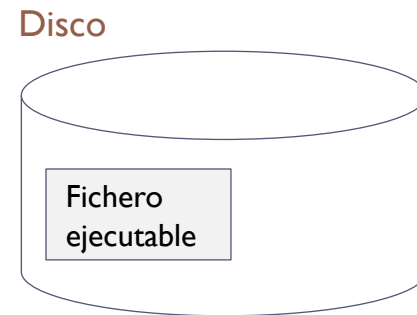
- ▶ Objetivo: acelerar el acceso (+ rápido)
- ▶ Transferencia por bloques o líneas.
- ▶ Bloques: 32-64 bytes.
- ▶ Traducción: algoritmo de correspondencia.
- ▶ Escritura inmediata o diferida.

Memoria virtual

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

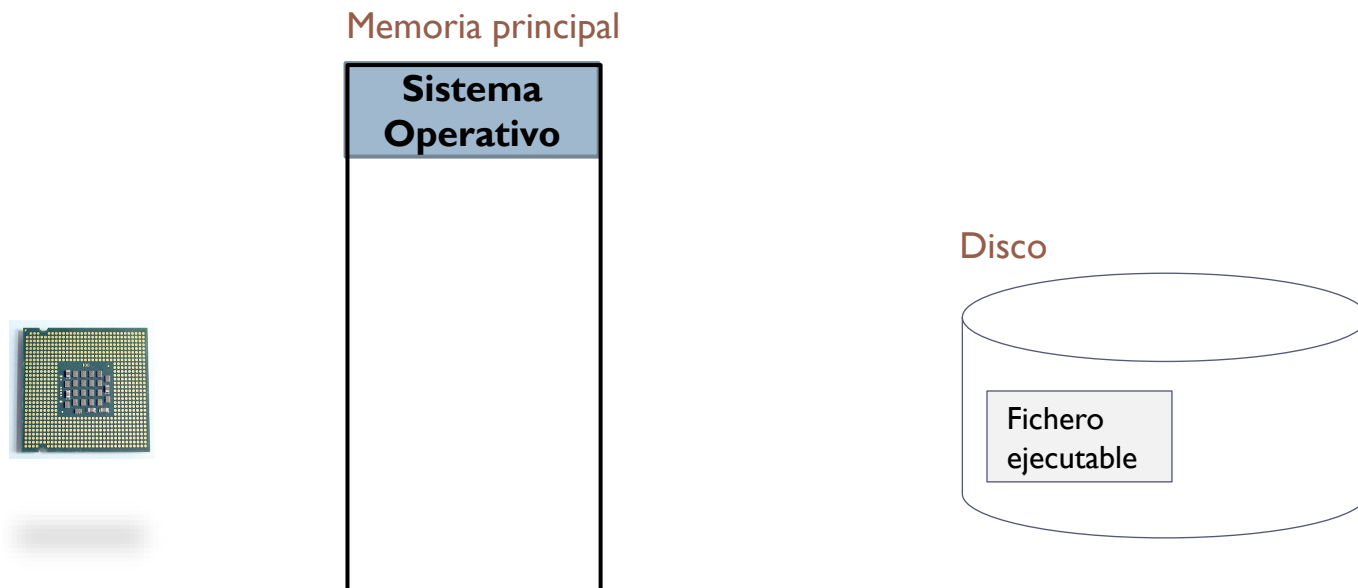
Programa y proceso

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



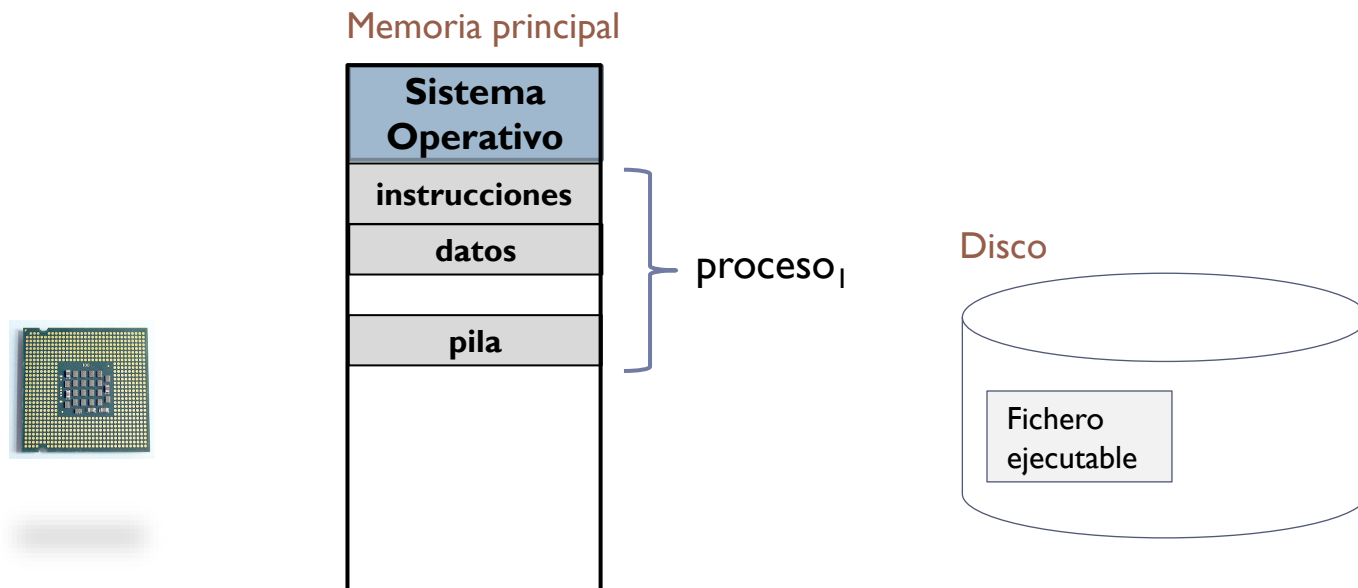
Programa y proceso

- ▶ **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 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)

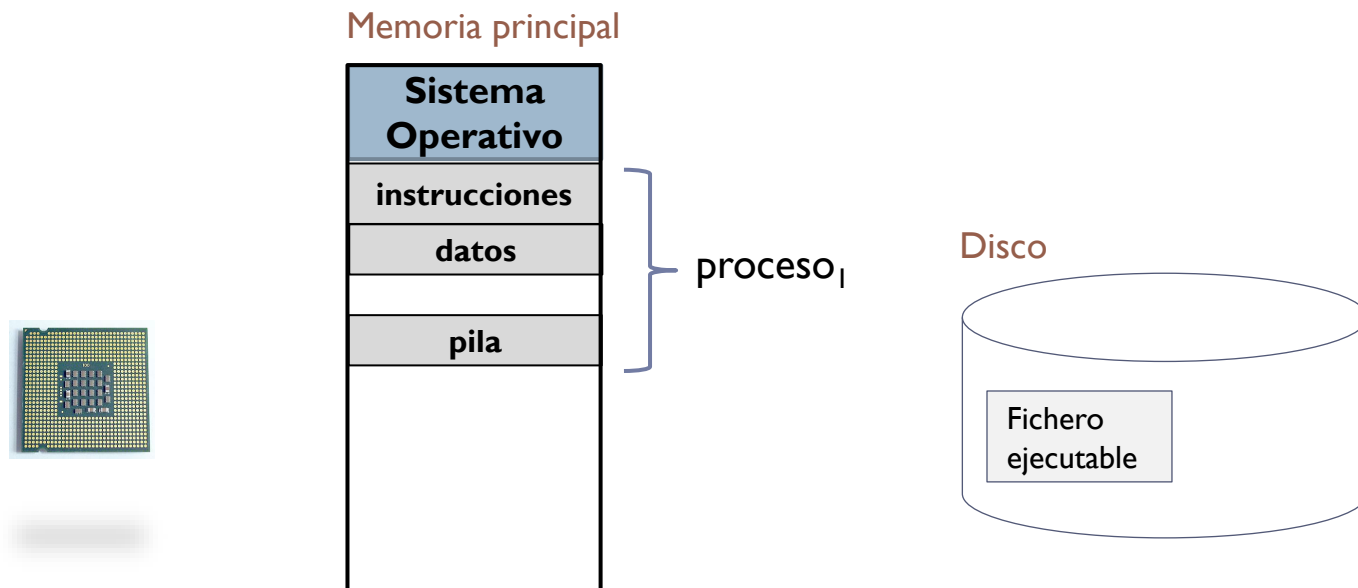
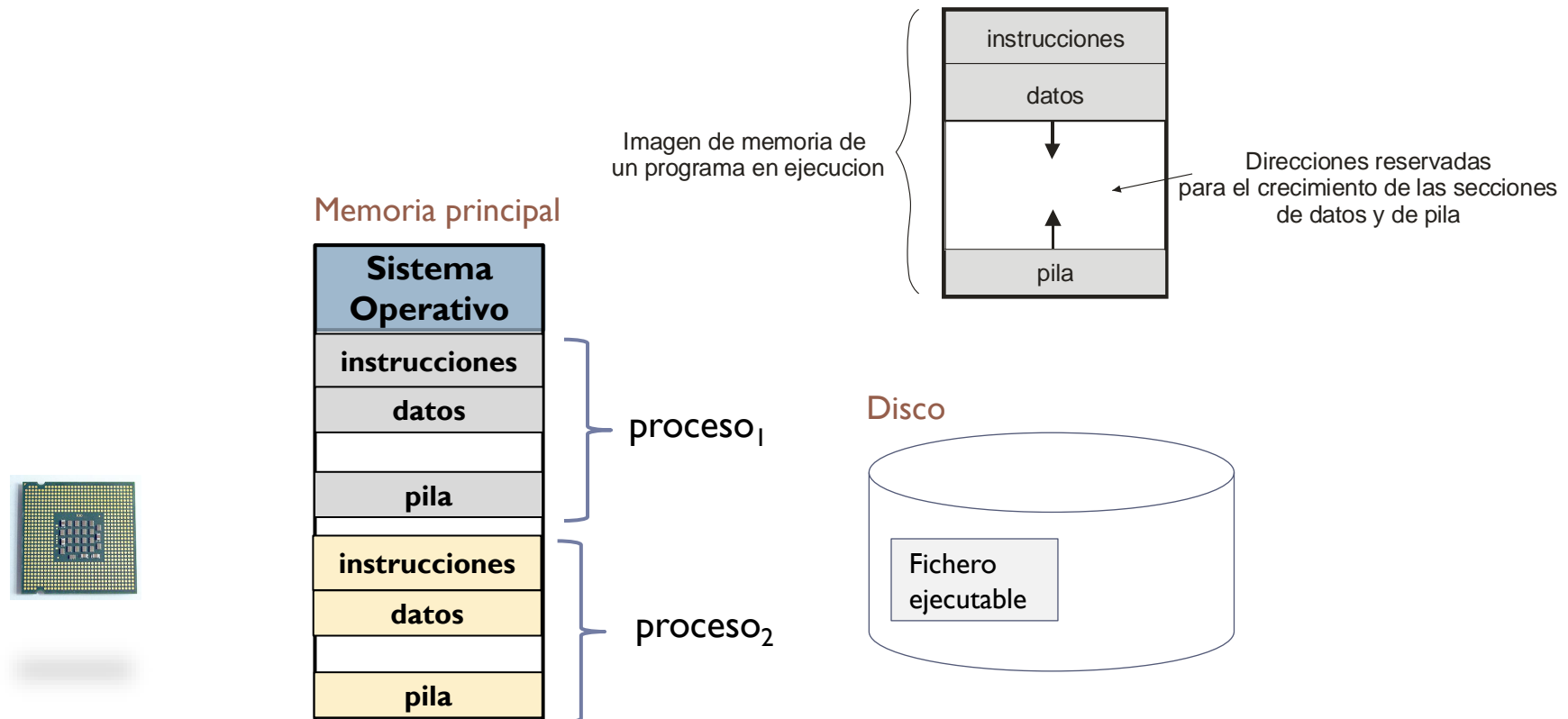


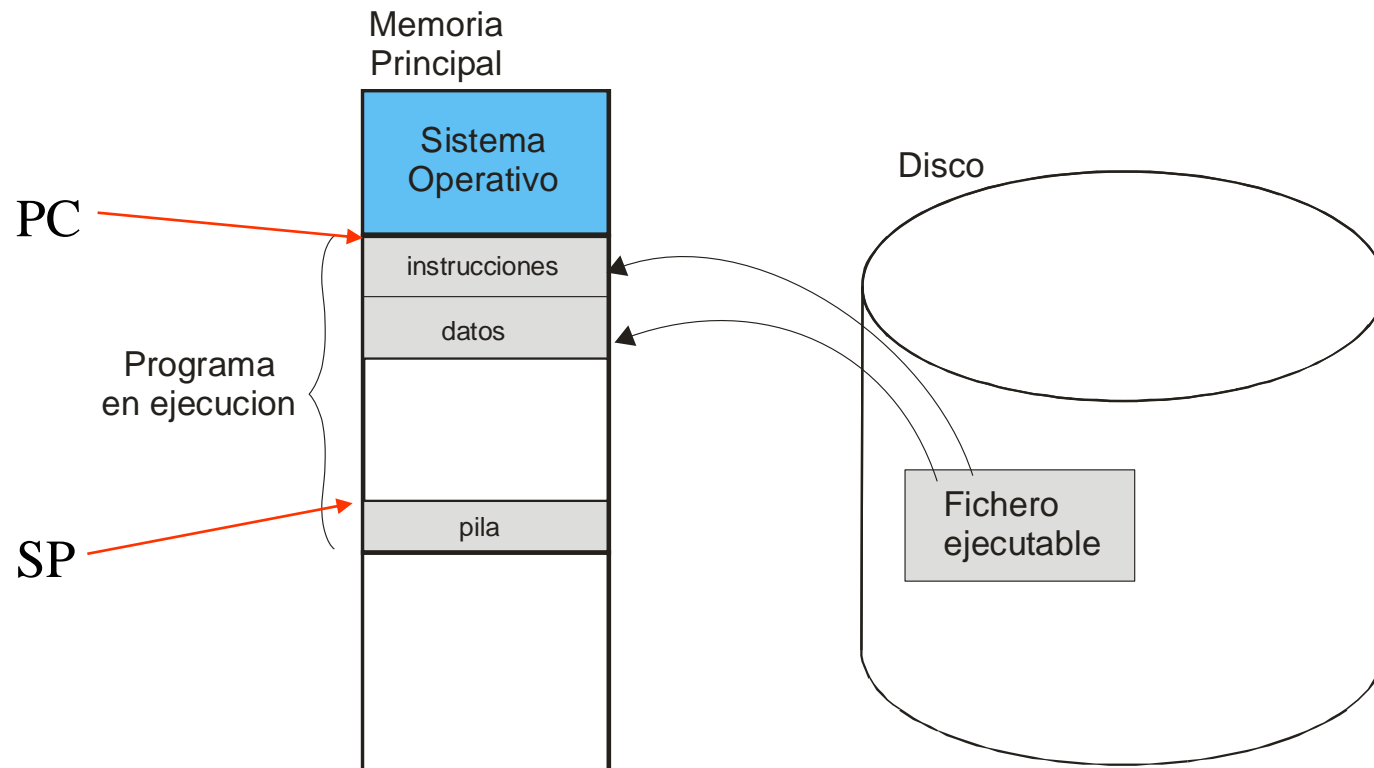
Imagen de un proceso

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



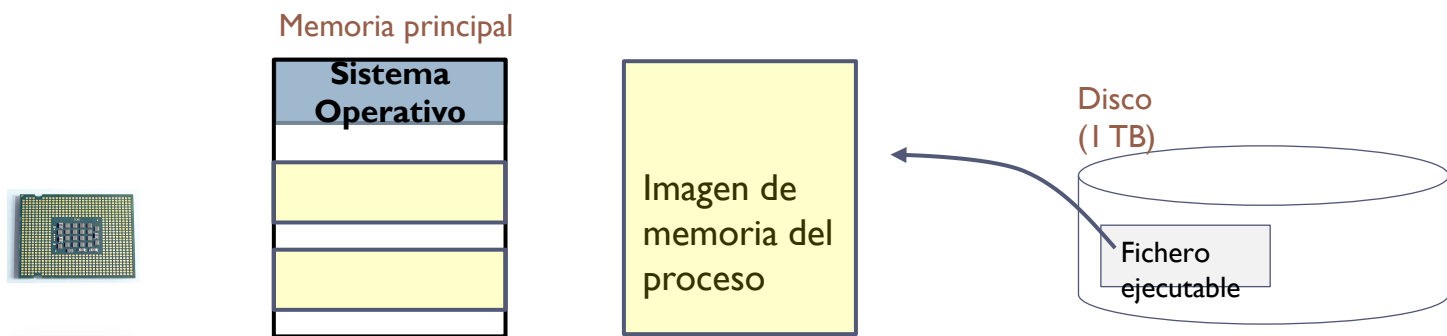
Sistemas **sin** memoria virtual

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



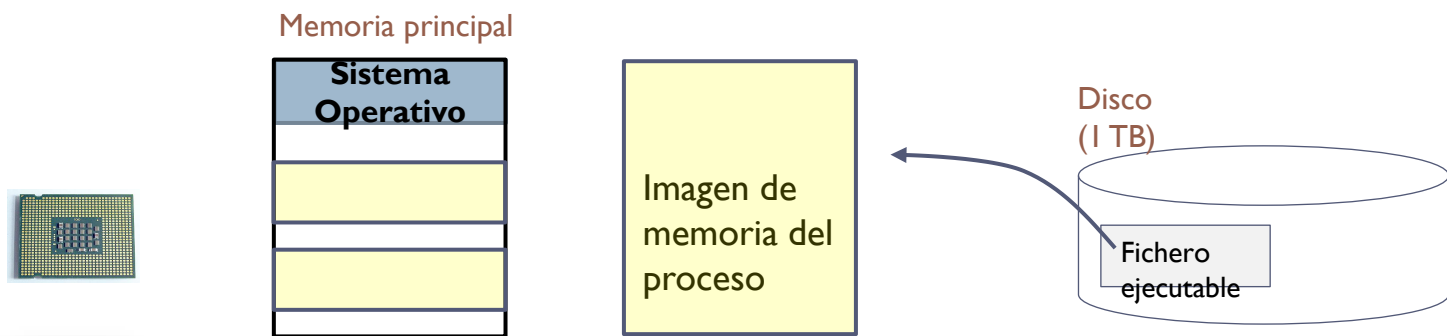
Sistemas **sin** memoria virtual

- ▶ 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 (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**.




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

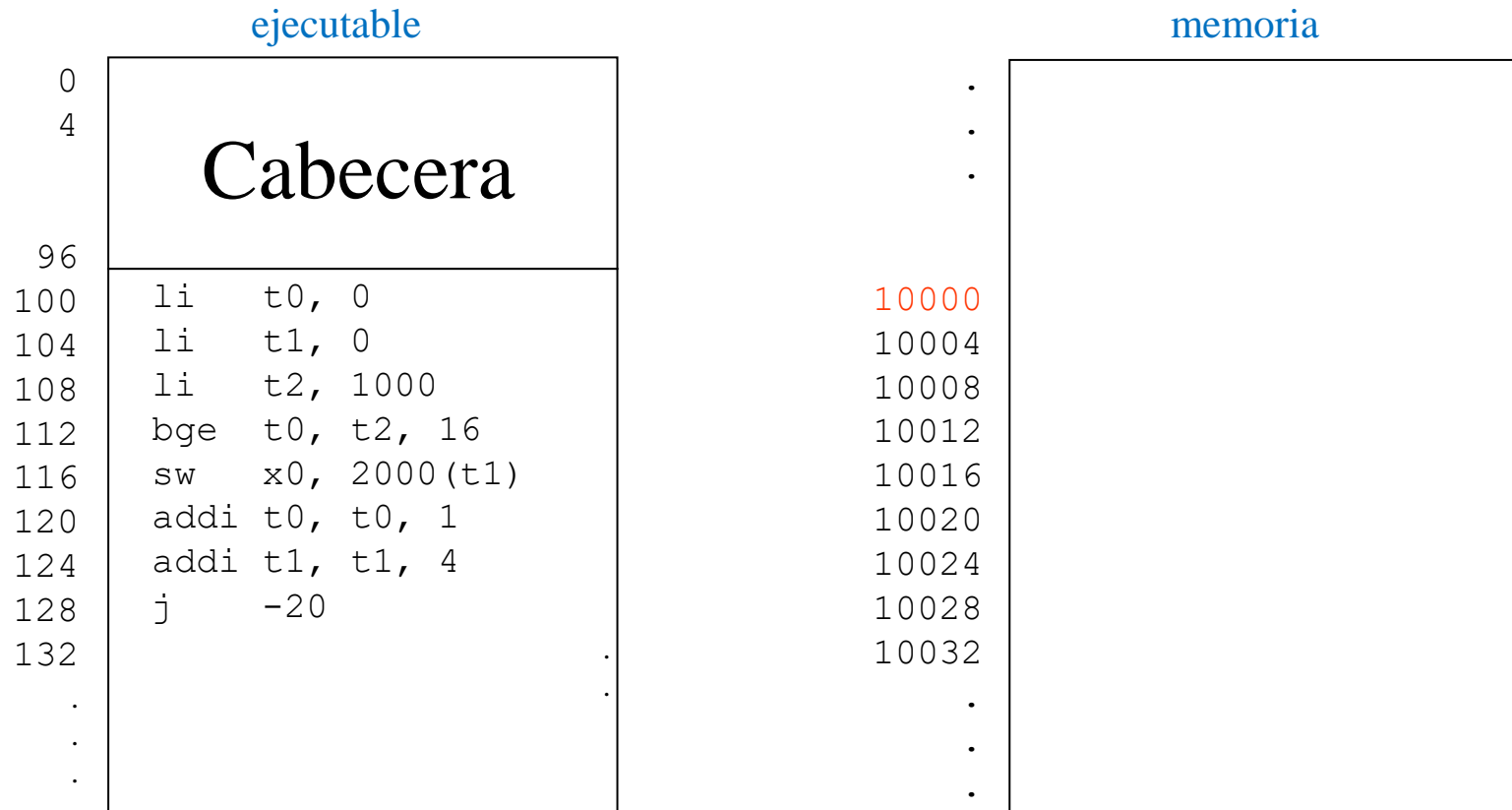
0	
4	
	Cabecera
96	
100	li t0, 0
104	li t1, 0
108	li t2, 1000
112	bge t0, t2, 16
116	sw x0, 2000(t1)
120	addi t0, t0, 1
124	addi t1, t1, 4
128	j -20
132	
.	.
.	.
.	.

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

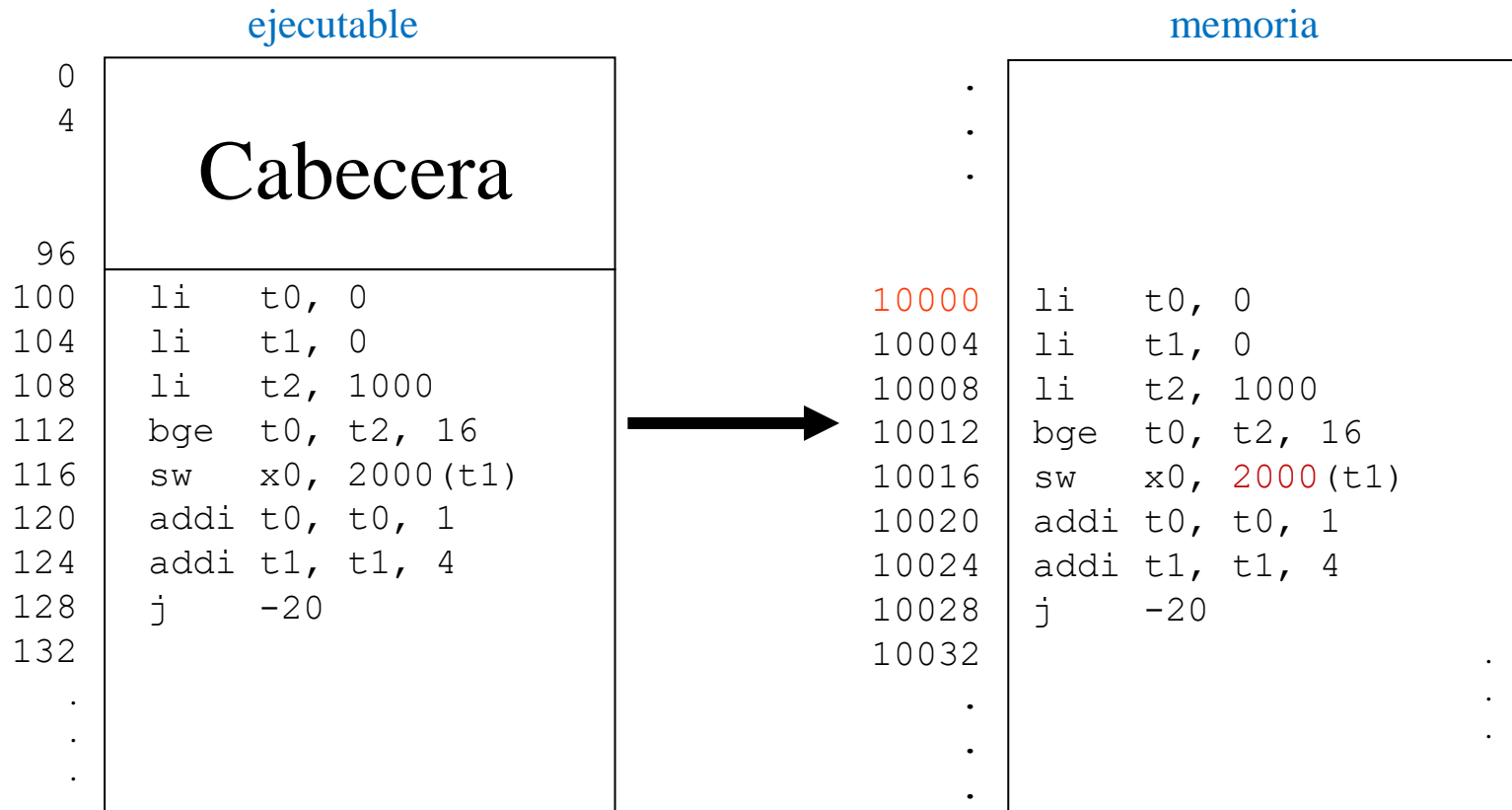


Carga del programa en memoria

- ▶ En el fichero ejecutable se considera como dirección de inicio la 0
 - ▶ Direcciones lógicas
- ▶ En memoria, la dirección de inicio es la 10000
 - ▶ Direcciones físicas
- ▶ Hay que realizar una **traducción de direcciones**
 - ▶ De direcciones lógicas a físicas
- ▶ El array en memoria está
 - ▶ En la dirección lógica 2000
 - ▶ En la dirección física $2000 + 10000$
- ▶ A este proceso se le denomina **reubicación**
 - ▶ Reubicación software
 - ▶ Reubicación hardware

Reubicación software

- Se realiza la traducción en el momento de la carga



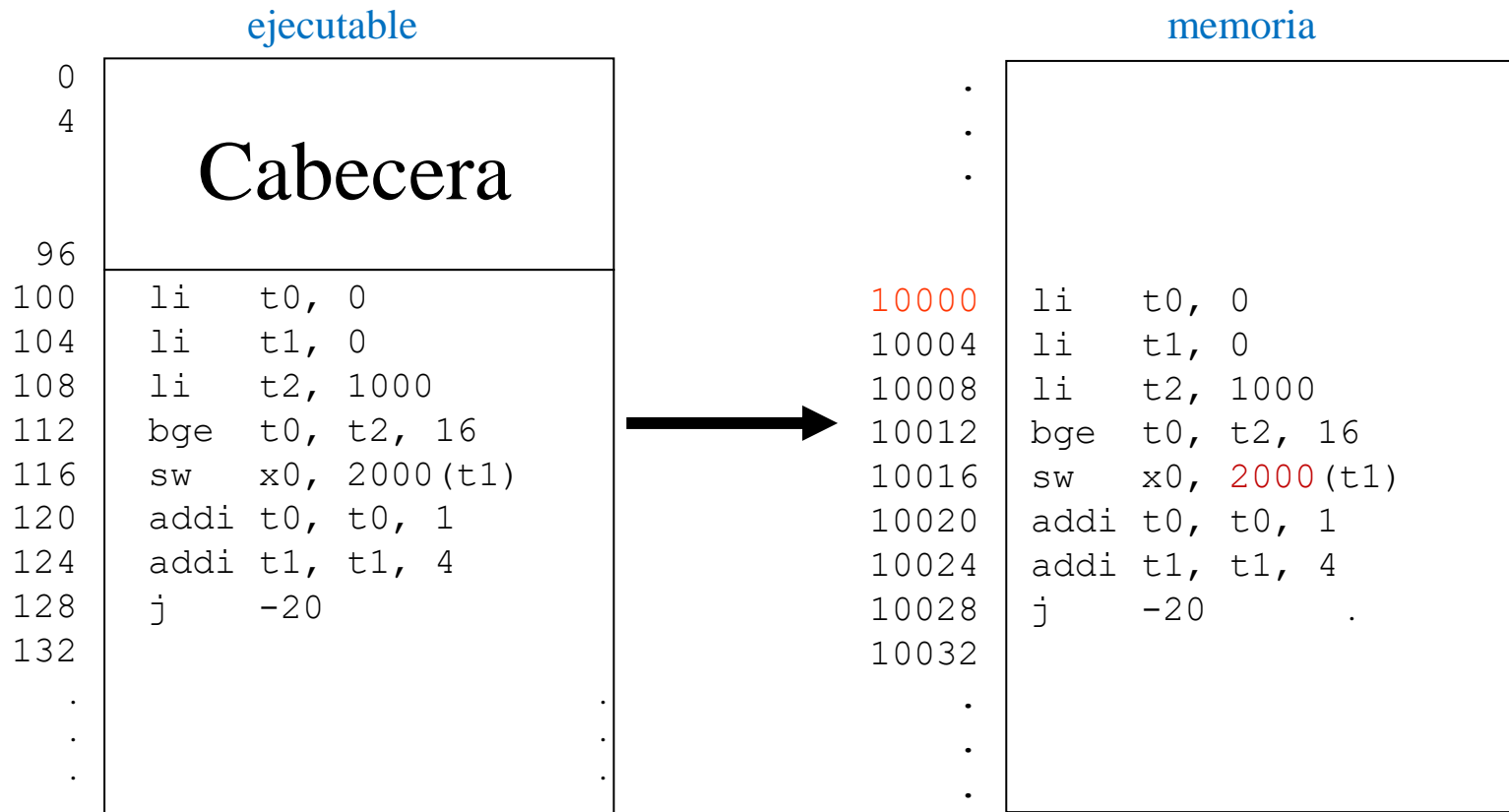
Reubicación software

- ¿Qué ocurre con estas instrucciones cargadas en las posiciones 10012 y 10028?

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, 16
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 -20
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

MAR

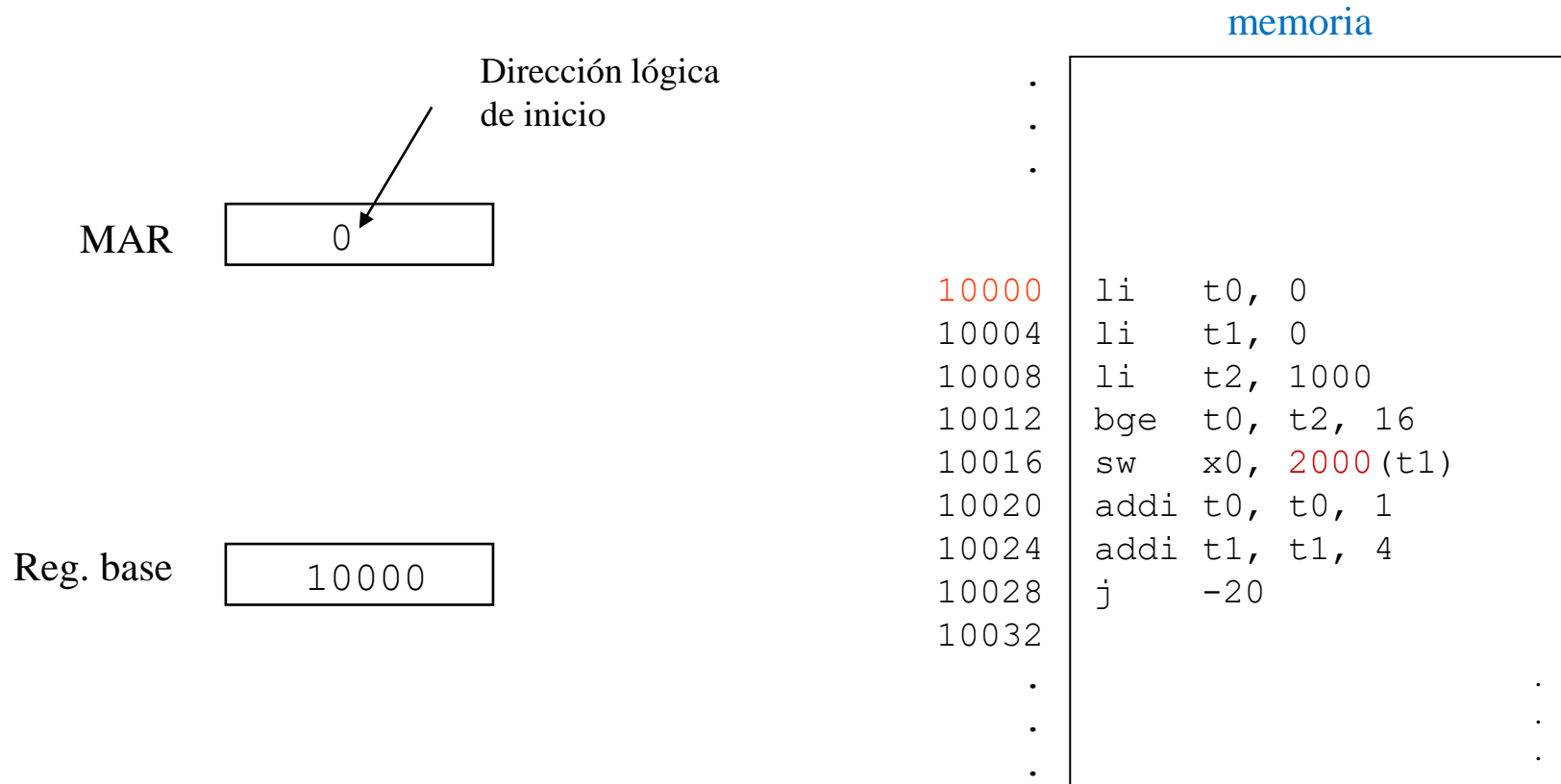
Reg. base

memoria

```
.  
. .  
10000 li    t0, 0  
10004 li    t1, 0  
10008 li    t2, 1000  
10012 bge   t0, t2, 16  
10016 sw    x0, 2000(t1)  
10020 addi  t0, t0, 1  
10024 addi  t1, t1, 4  
10028 j     -20  
10032 .
```

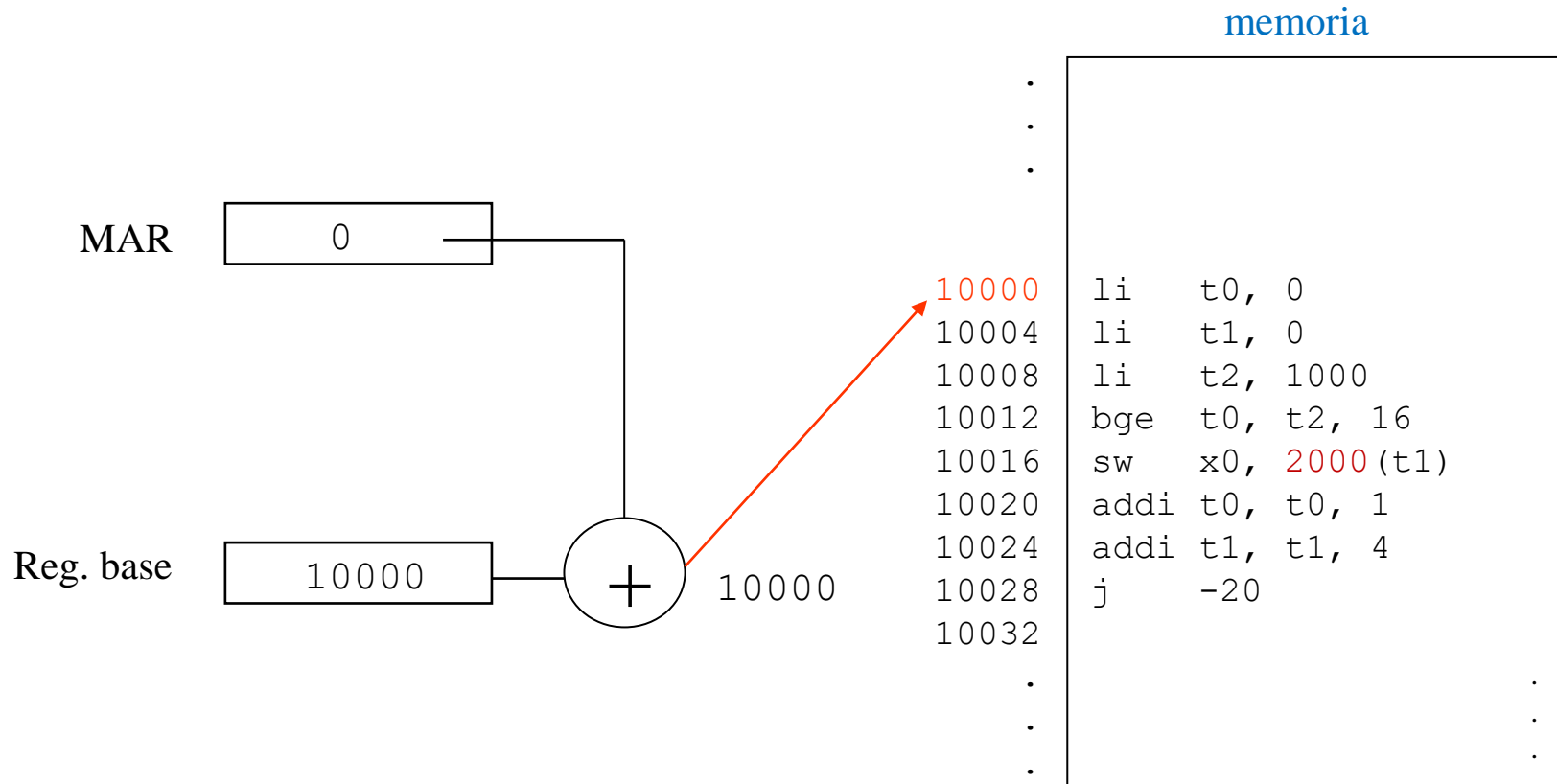
Ejemplo de soporte hardware

- Registro base: dirección de inicio del programa en memoria



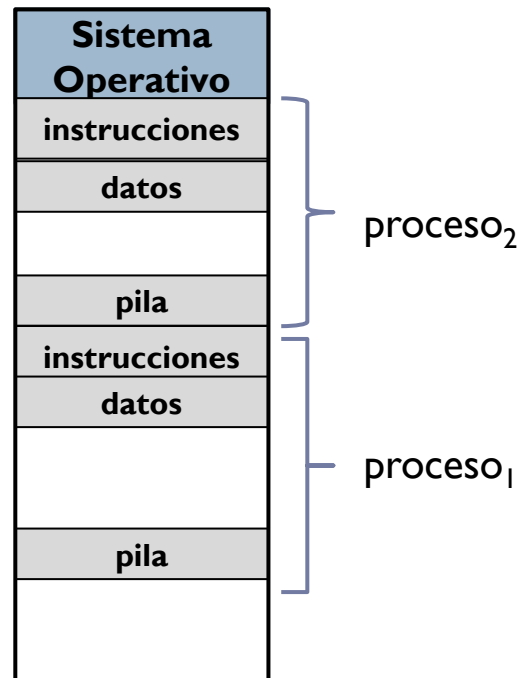
Ejemplo de soporte hardware

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

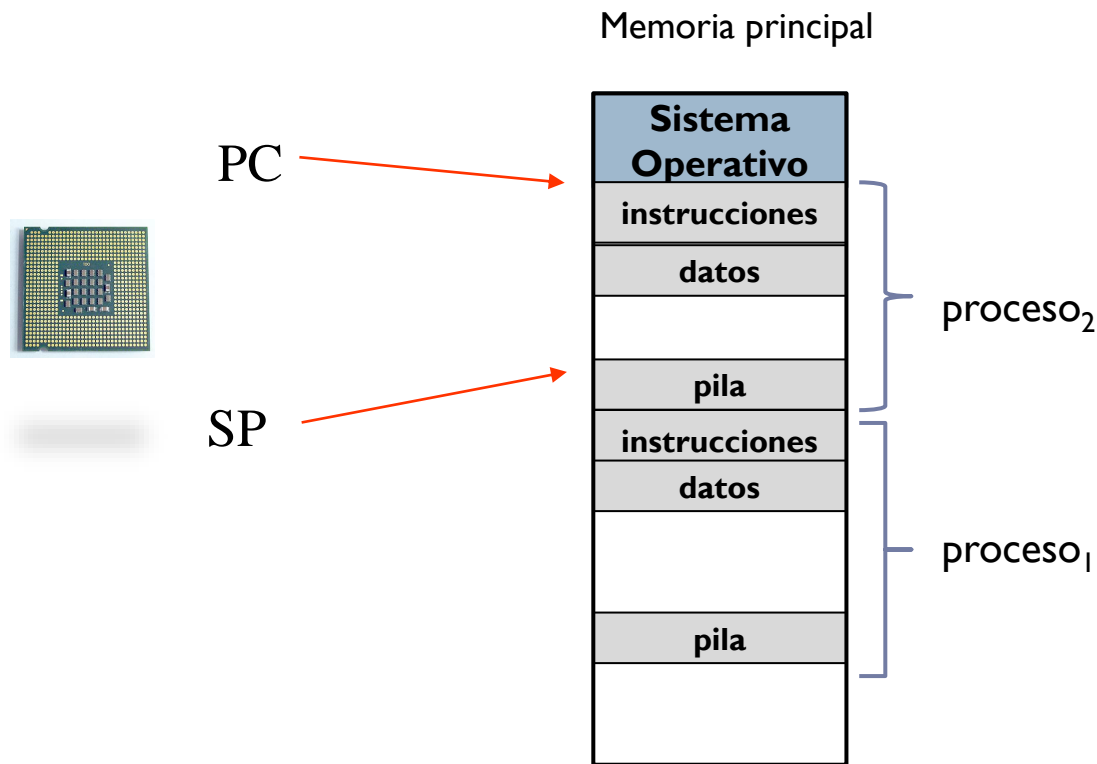


Múltiples programas cargados en memoria

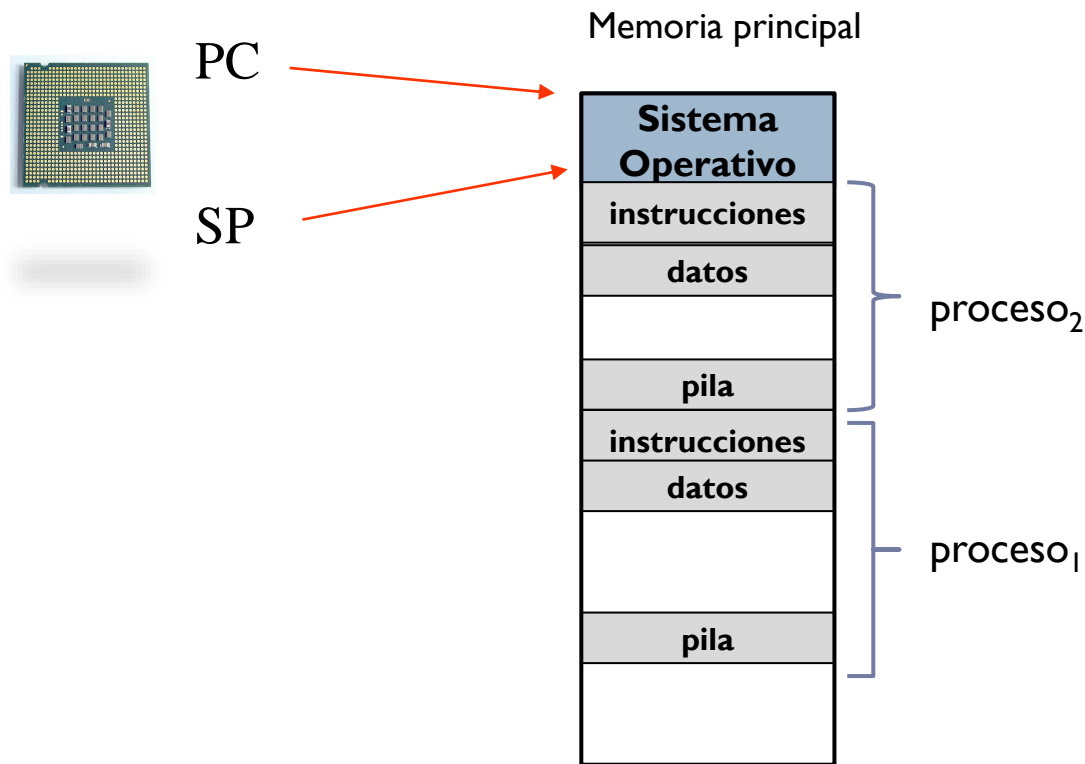
Memoria principal



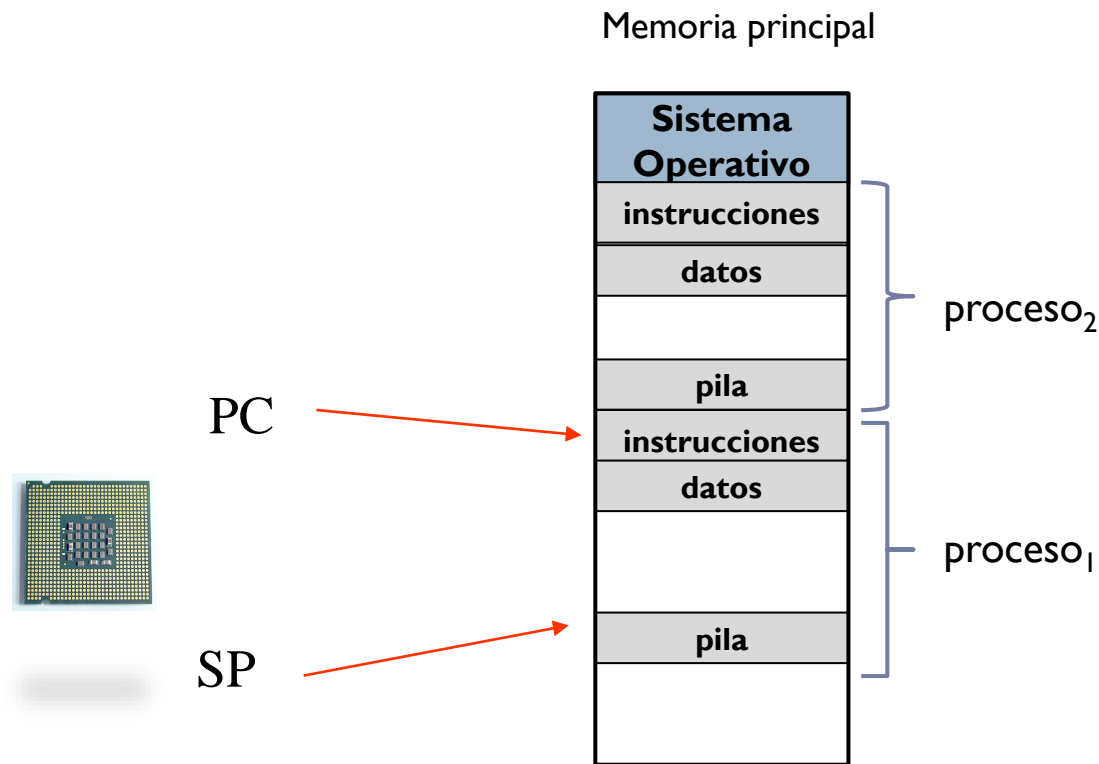
Múltiples programas cargados en memoria



Múltiples programas cargados en memoria

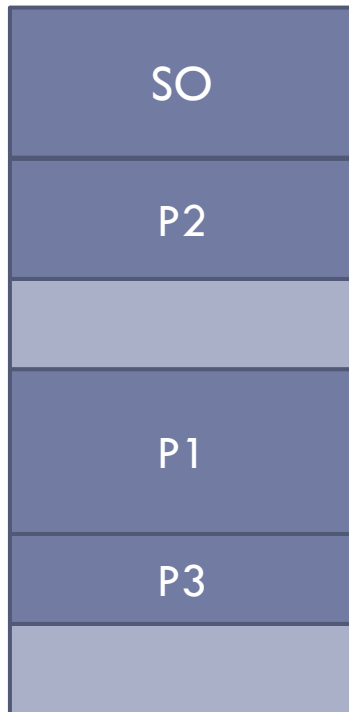


Múltiples programas cargados en memoria



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

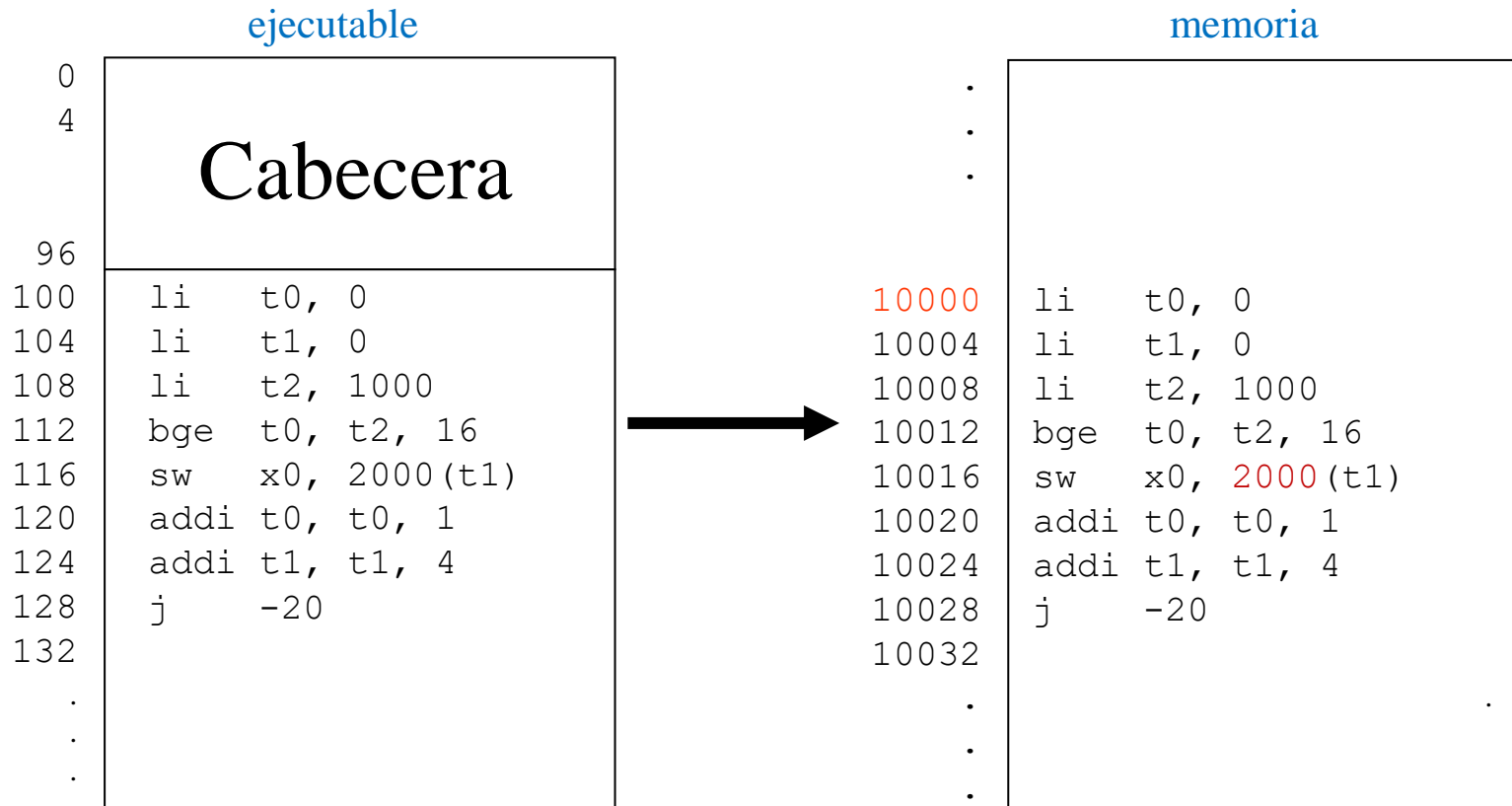
- ¿Qué ocurre si en el programa ejecuta las siguientes instrucciones?

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

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

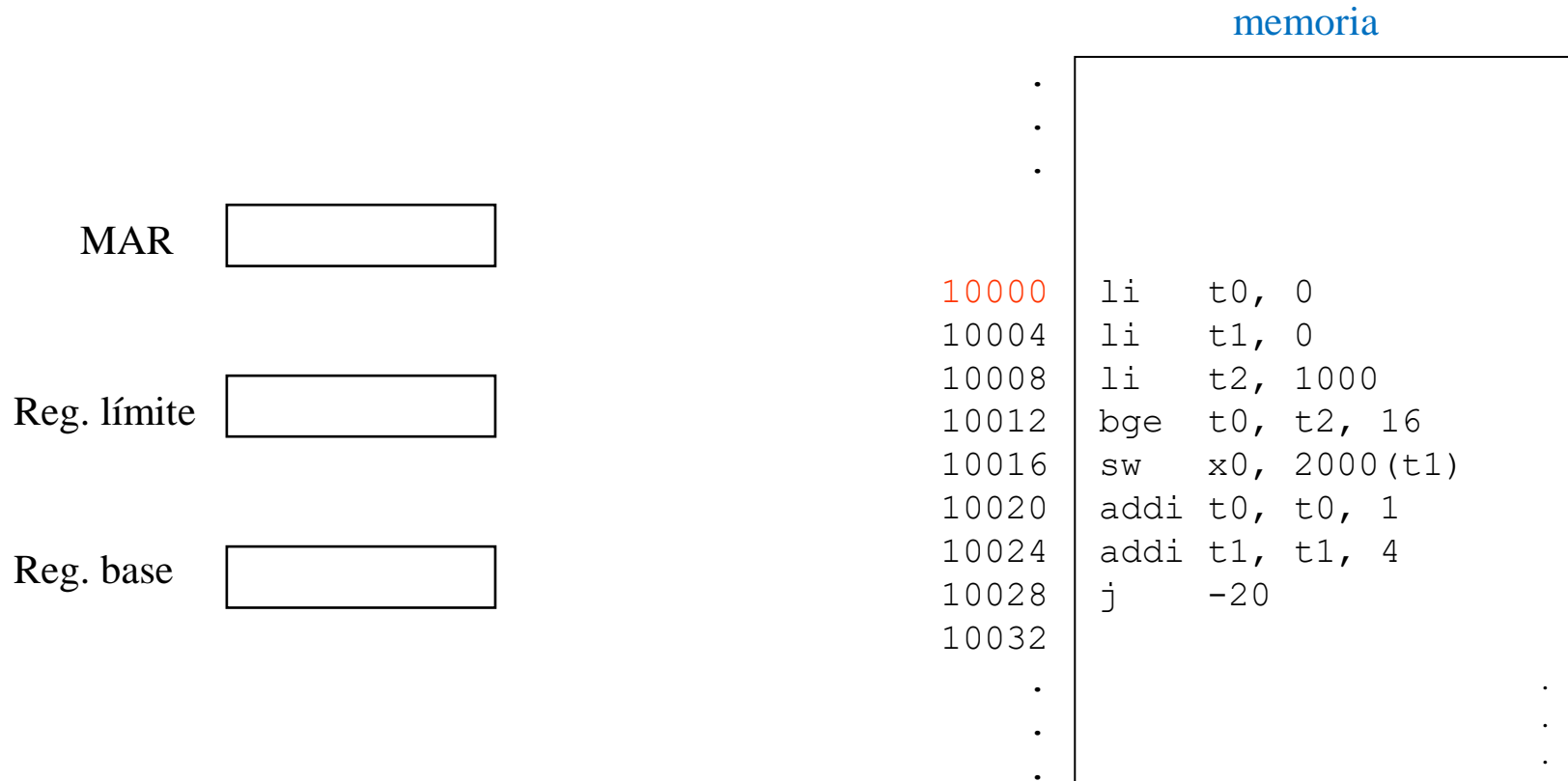
Reubicación hardware (con registro límite)

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



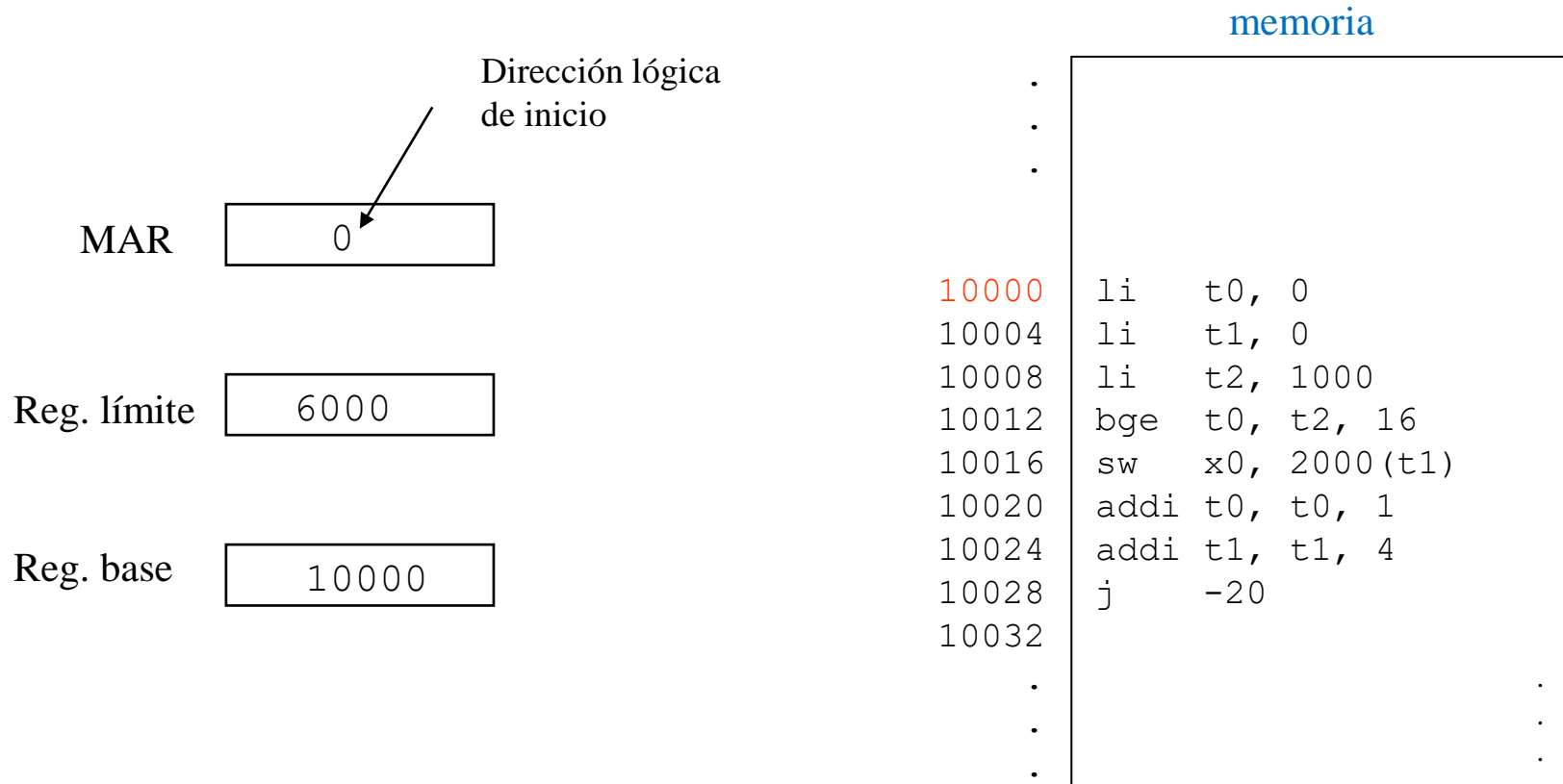
Ejemplo de soporte hardware

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



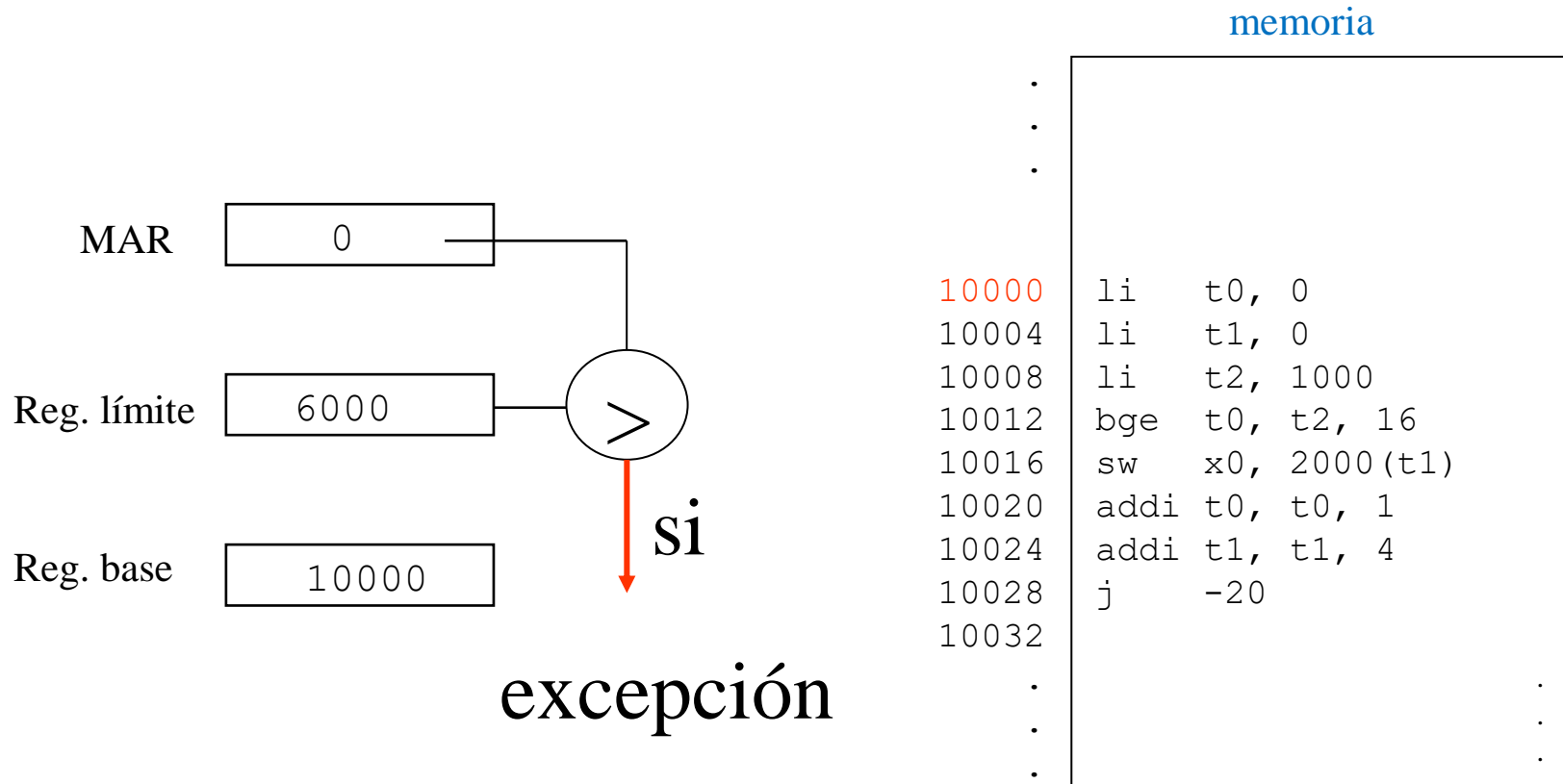
Ejemplo de soporte hardware

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



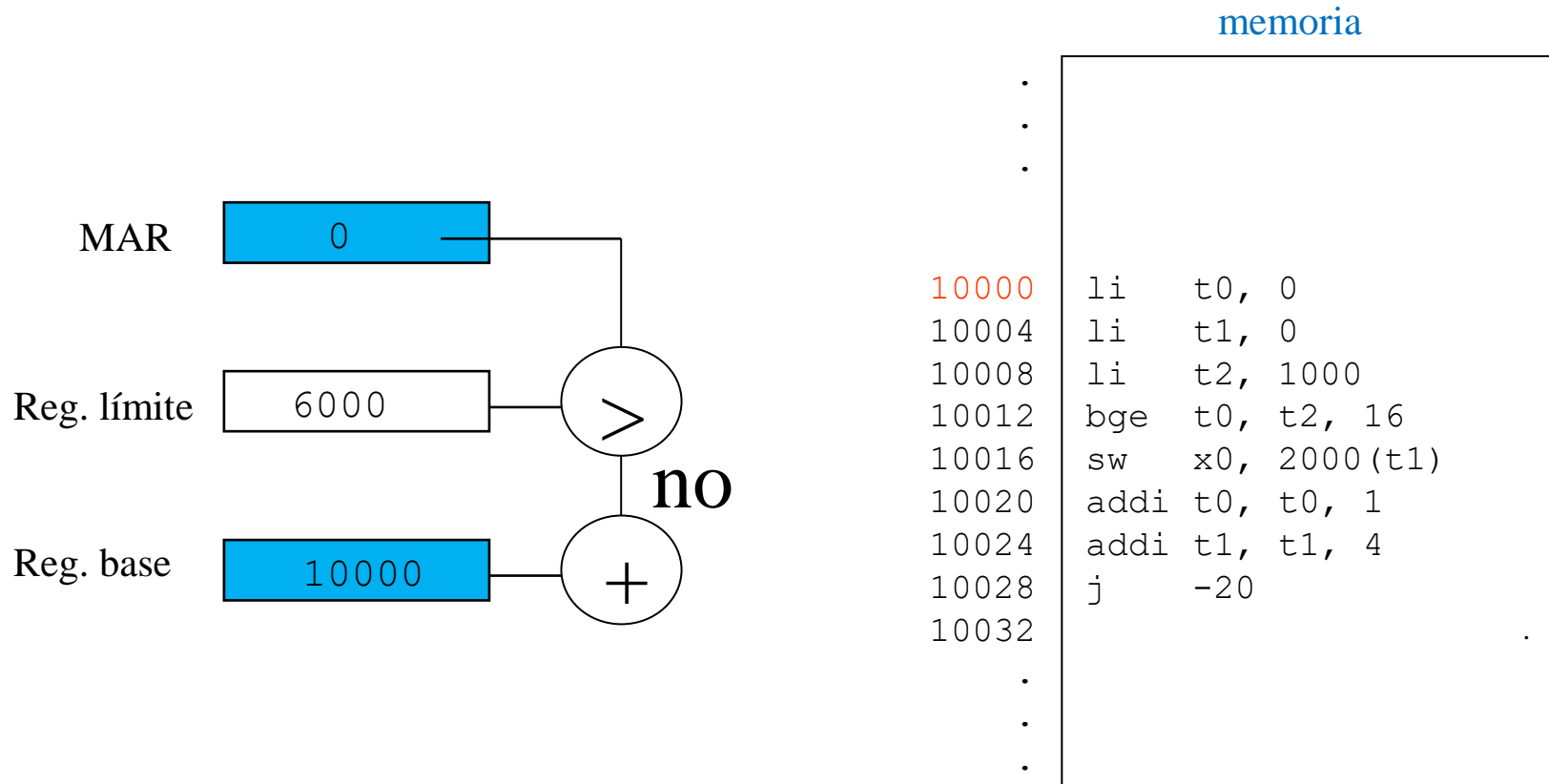
Ejemplo de soporte hardware

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



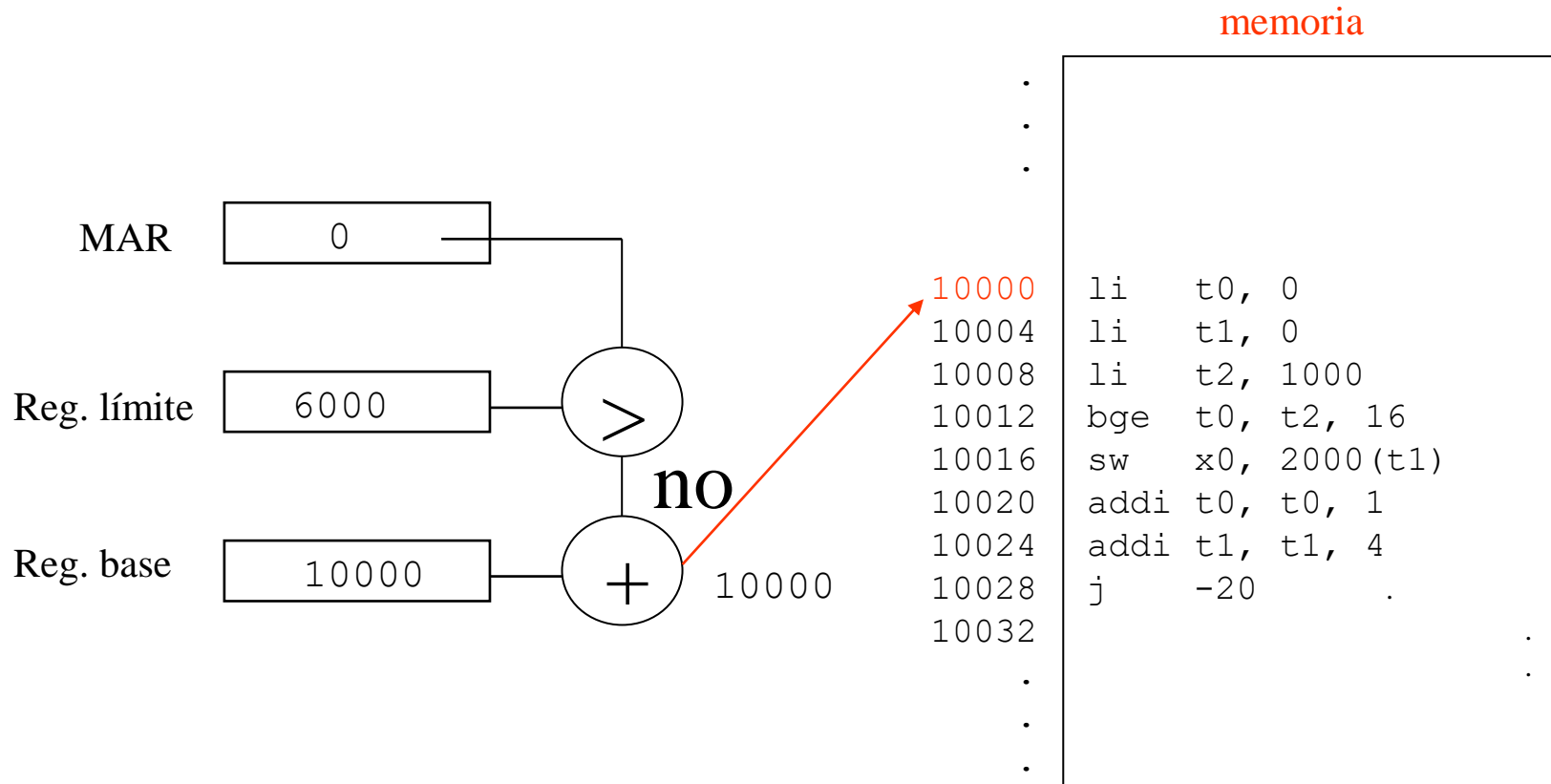
Ejemplo de soporte hardware

- ▶ 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

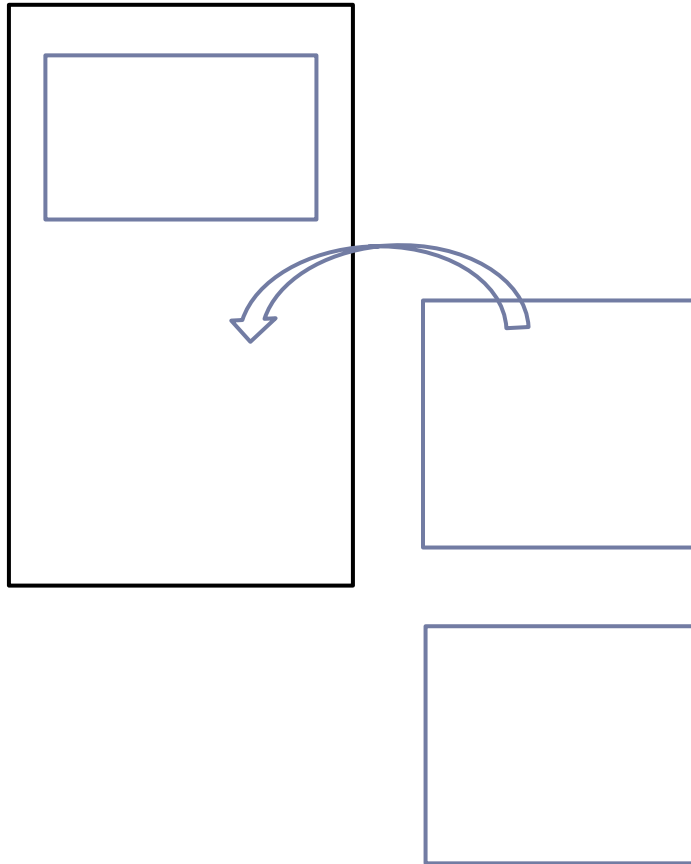


Sistemas sin memoria virtual

Principales problemas (resumen)

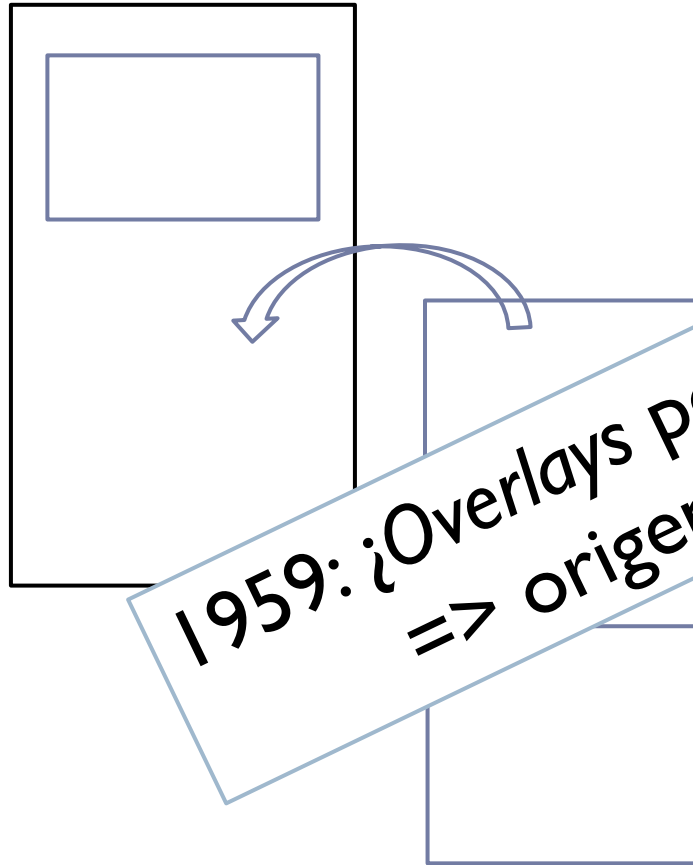
- ▶ **Reubicación y protección.**
- ▶ **Si la imagen en memoria del proceso es mayor que la memoria disponible, no se puede ejecutar.**
- ▶ **El número de programas activos en memoria limitado.**
- ▶ En un computador de 32 bits:
 - ▶ ¿Cuál es el tamaño máximo teórico del programa que se puede ejecutar?
 - ▶ ¿Y si solo se dispone de una memoria de 512 MiB?
 - ▶ Si cada programa ocupa 100 MiB, ¿cuántos puedo ejecutar?

Overlays



- ▶ 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.

Overlays

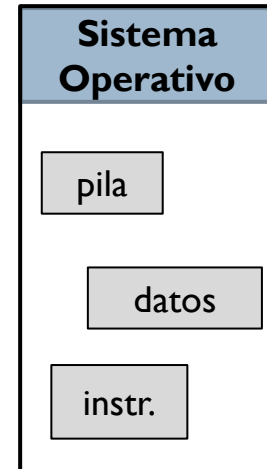


1959: ¿Overlays por hardware?
=> origen de la memoria virtual

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

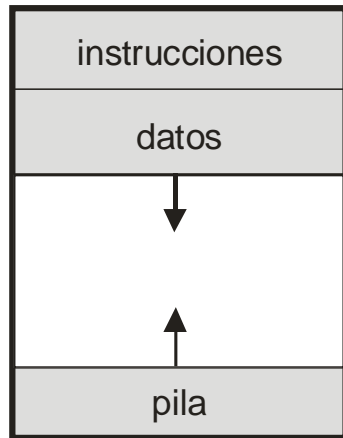
Sistemas **con** memoria virtual

- ▶ Los programas se carga 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.

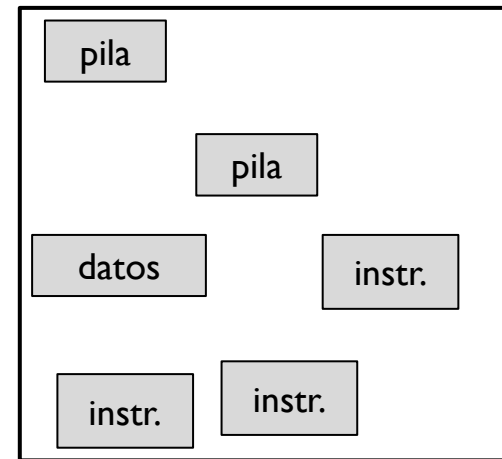


Sistemas **con** memoria virtual

Imagen de memoria



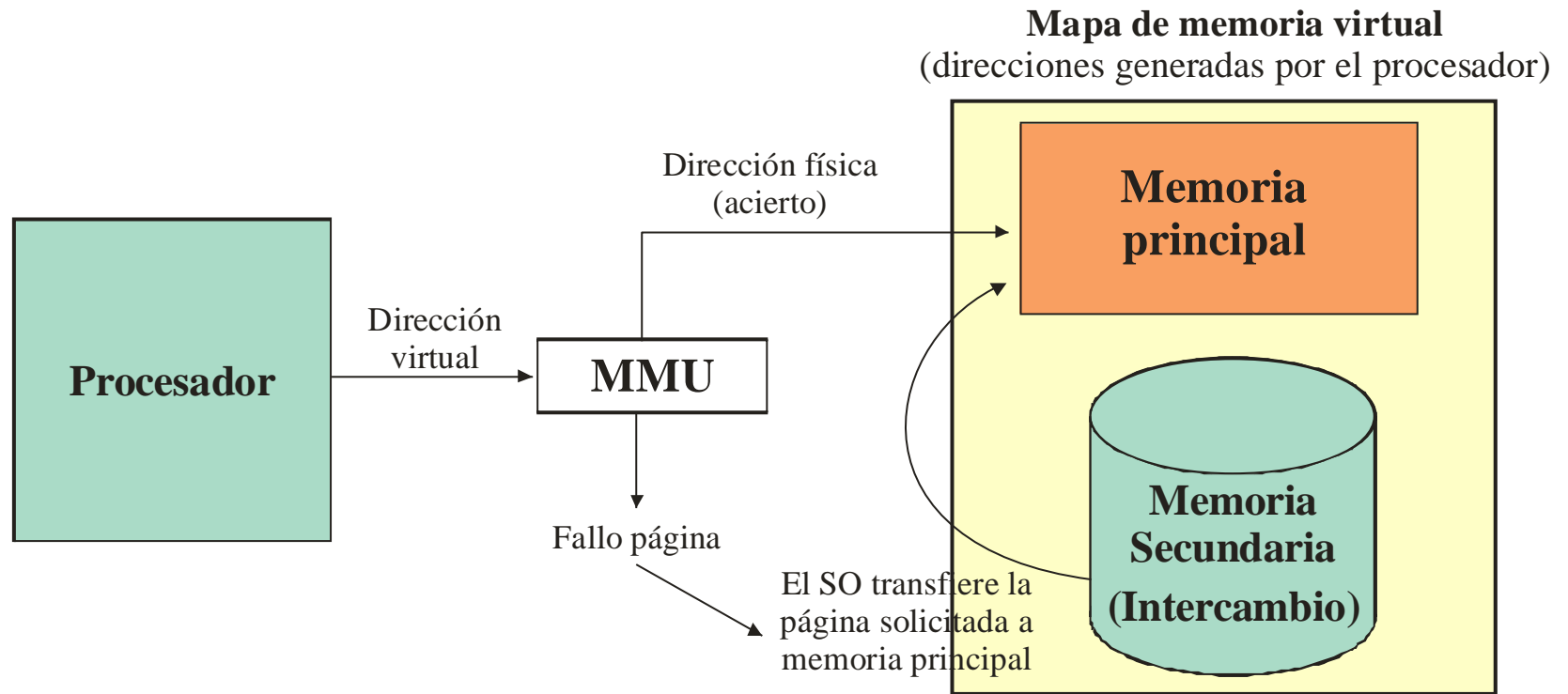
Memoria principal



Fundamentos de la memoria virtual

La M.V. utiliza dos niveles:

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



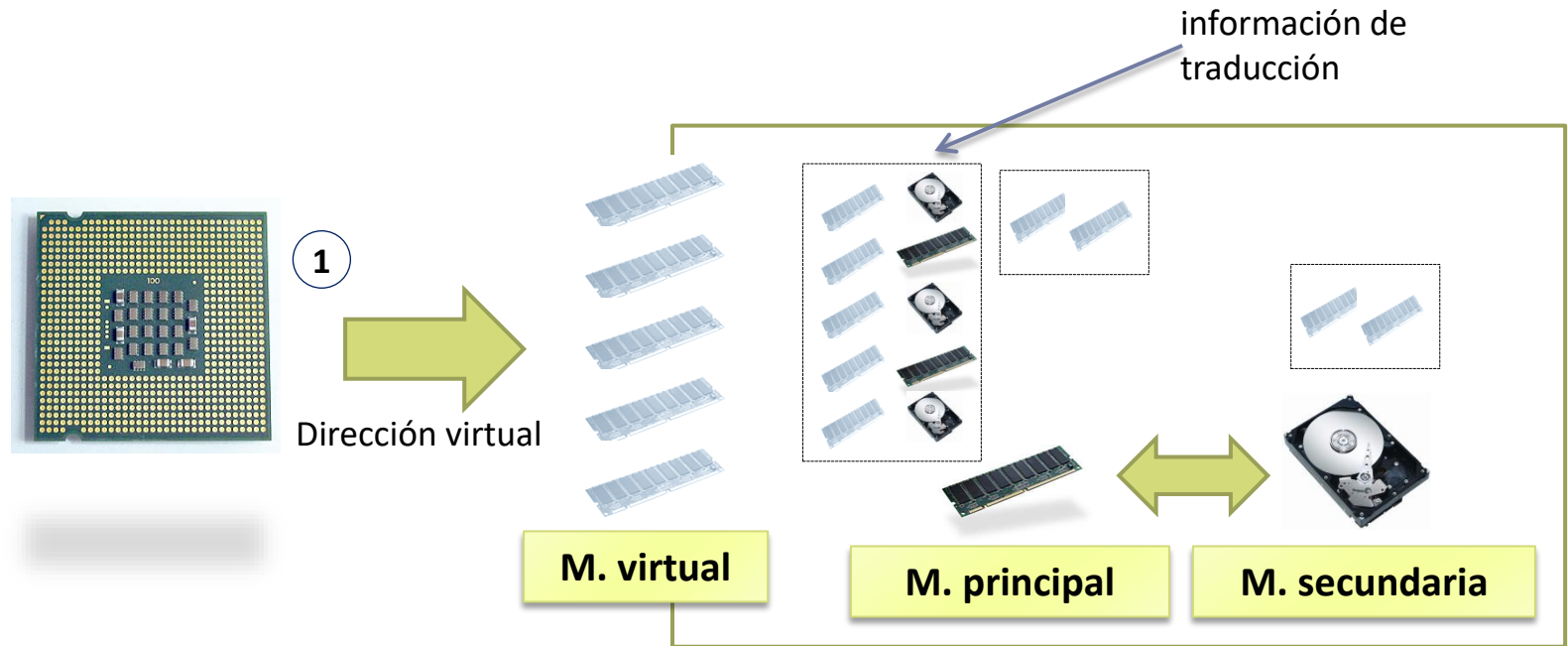
Diferentes modelos de memoria virtual

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

Memoria virtual **paginada**

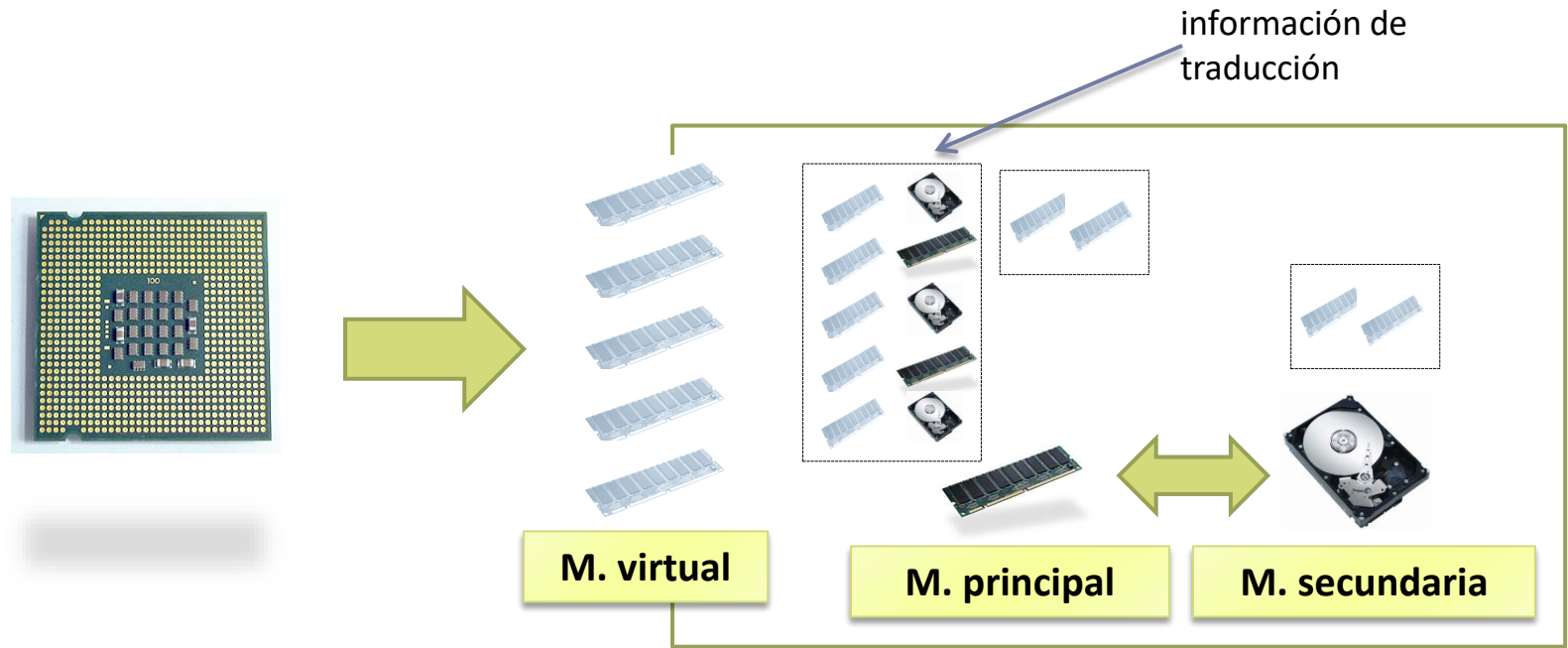
- ▶ Las direcciones que genera el procesador son **direcciones virtuales**
- ▶ El espacio de direcciones virtuales se divide en trozos de igual tamaño denominado **páginas**
- ▶ La memoria principal se divide en trozos de igual tamaño a las páginas denominados **marcos de página**
- ▶ La zona del disco 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

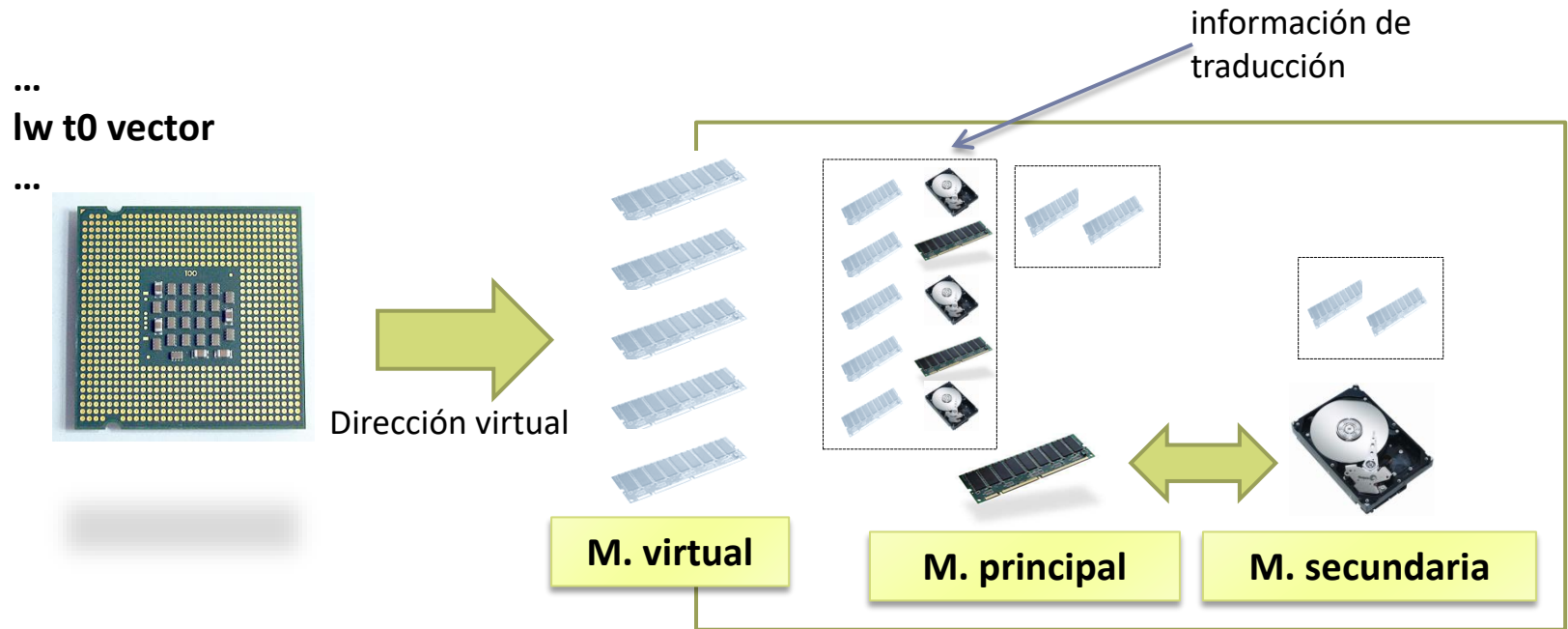


- ▶ **Espacio virtual de direcciones (M. virtual)**
 - ▶ Los programas manejan un espacio virtual que reside en MP+disco
 - ▶ MMU: Unidad de gestión de memoria

Fundamentos de la memoria virtual



Fundamentos de la memoria virtual

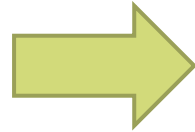
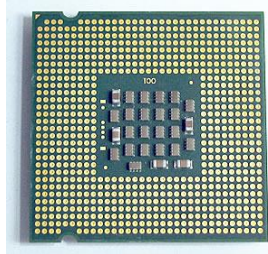


Fundamentos de la memoria virtual

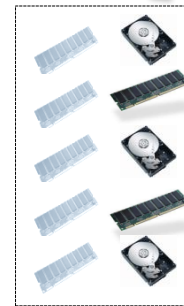
...

lw t0 vector

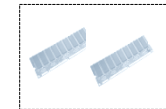
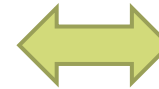
...



M. virtual



M. principal



M. secundaria

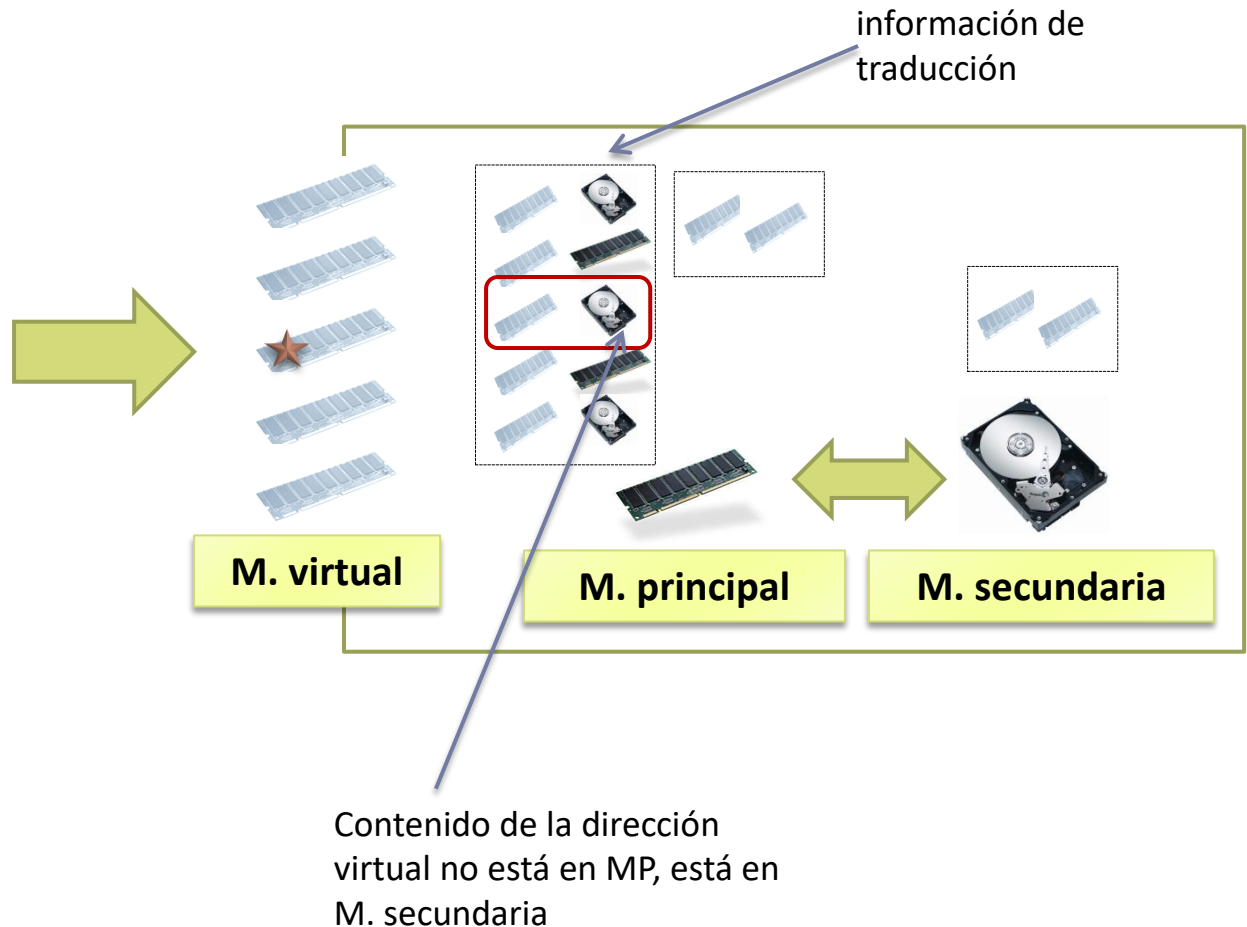
información de traducción

Fundamentos de la memoria virtual

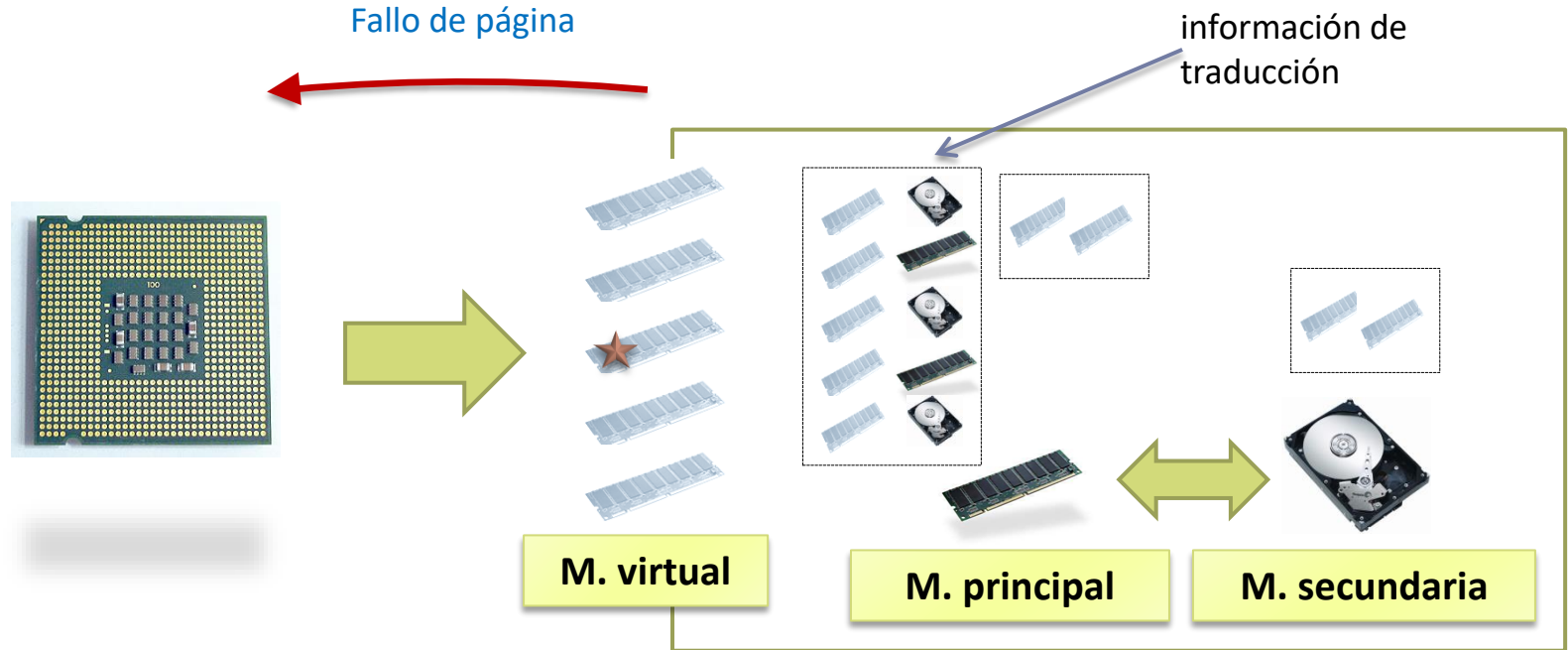
...

lw t0 vector

...

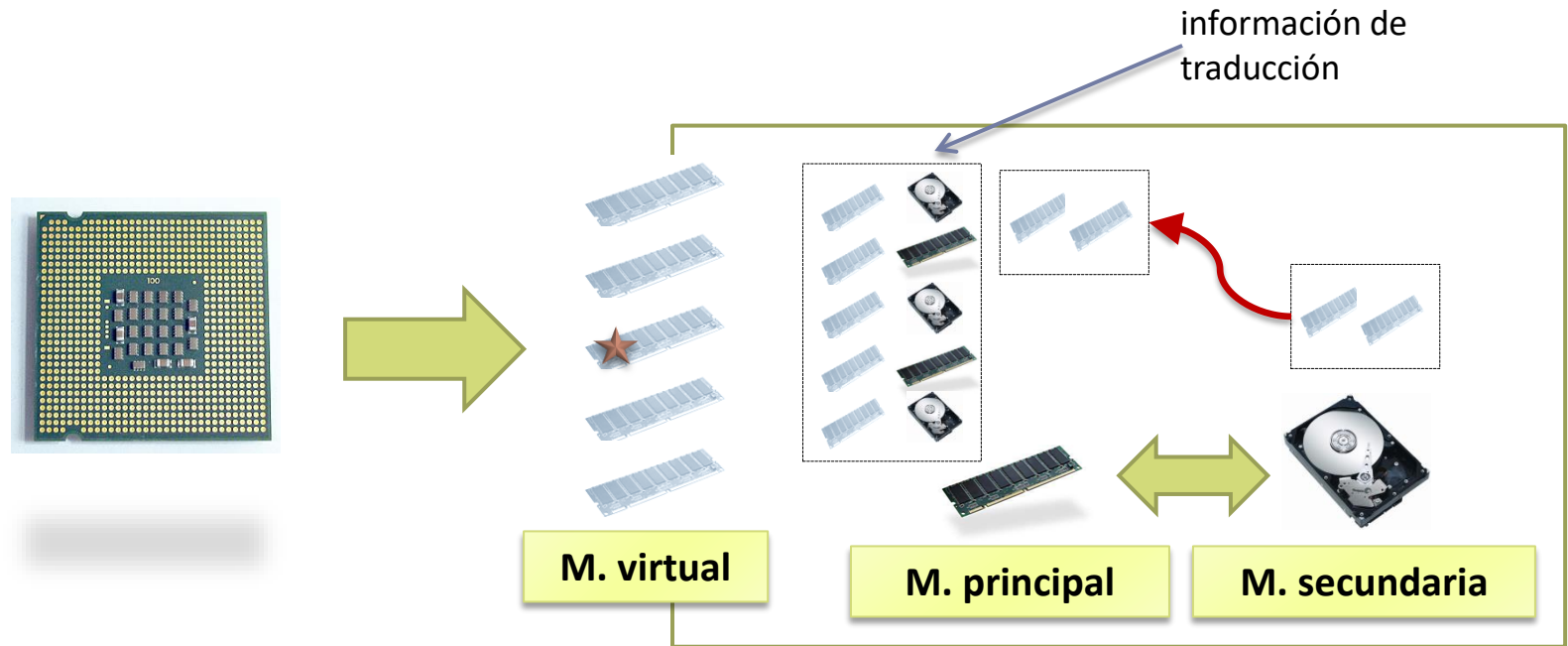


Fundamentos de la memoria virtual



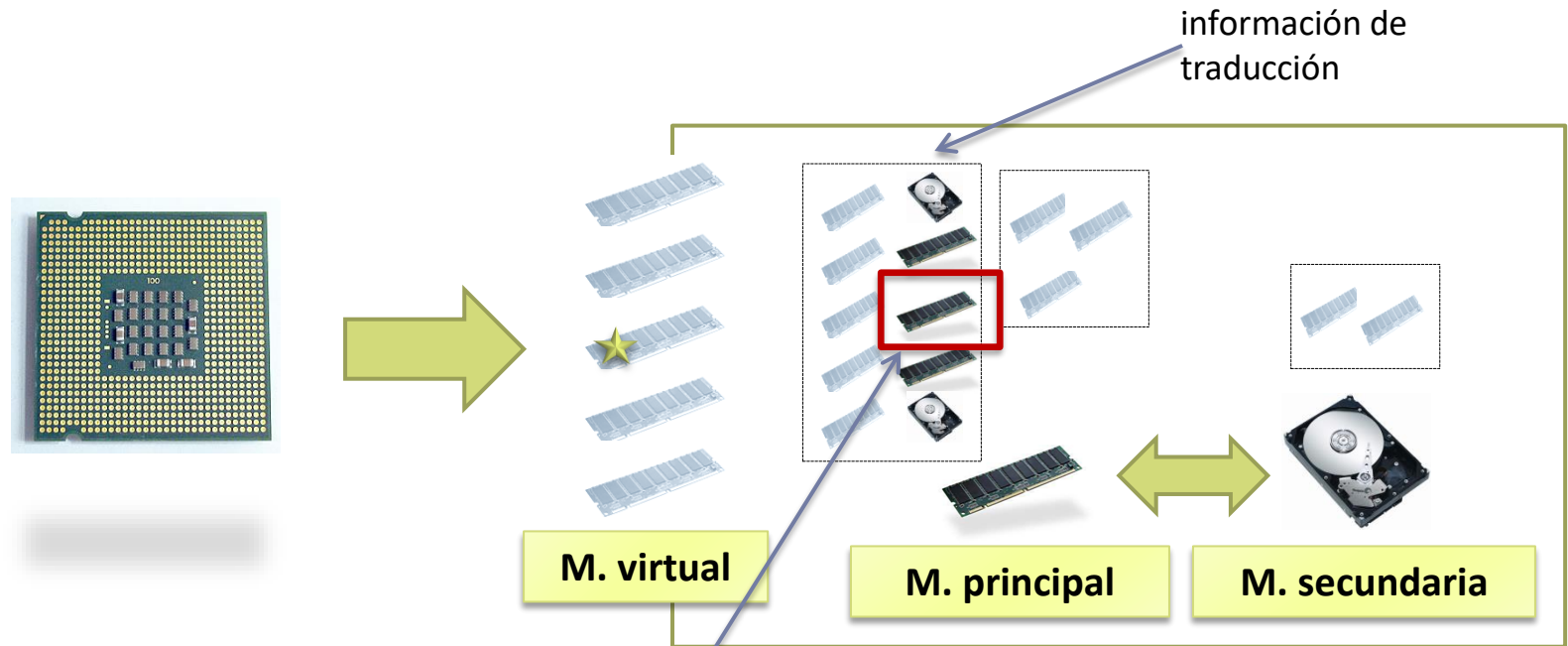
- ▶ El fallo de página es una excepción que provoca que el procesador ejecute la rutina de tratamiento asociada (el proceso que generó el fallo de pagina se suspende, no puede continuar su ejecución)
- ▶ Está implementada en el sistema operativo.

Introducción a memoria virtual



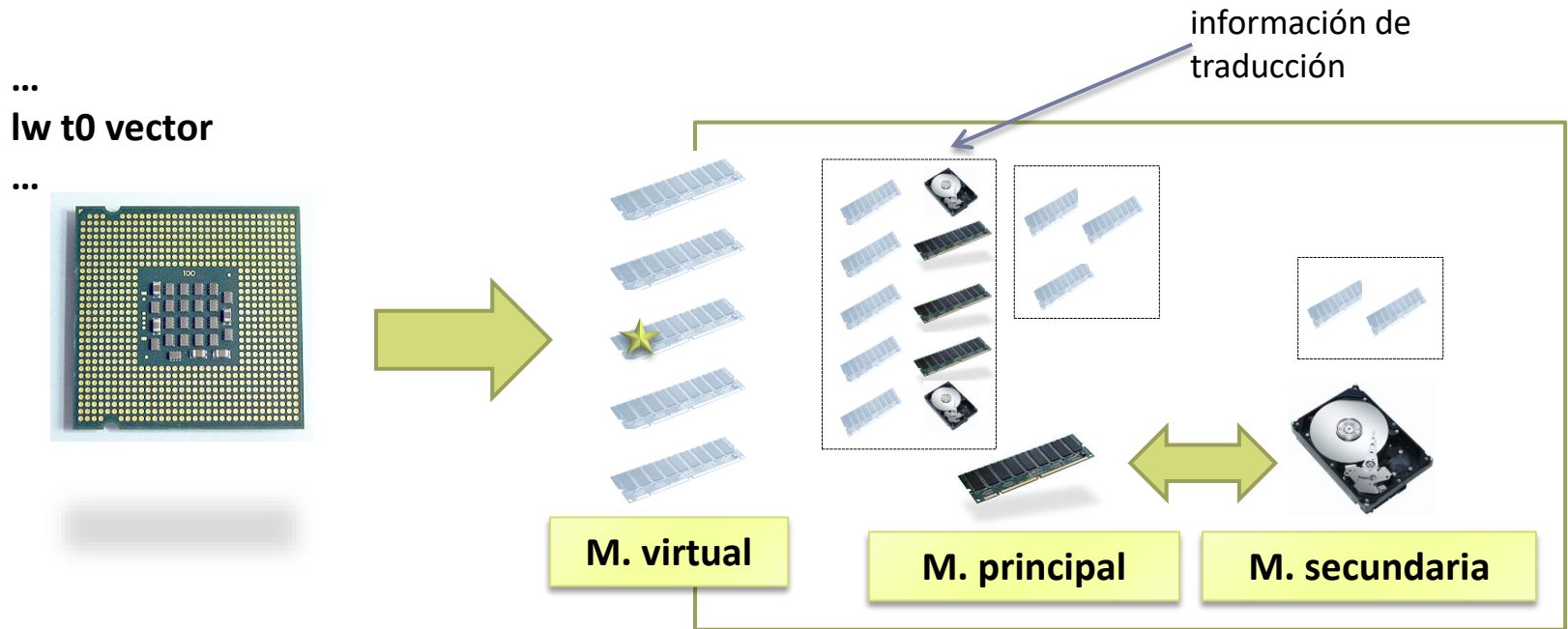
- ▶ El sistema operativo pide transferir el 'bloque' solicitado a memoria principal (el sistema operativo pone a otro proceso a ejecutar)

Introducción a memoria virtual



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

Introducción a memoria virtual



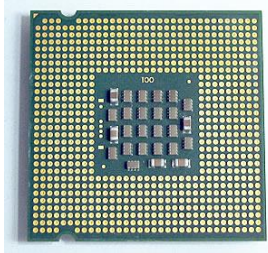
- ▶ Se reanuda la ejecución del proceso que provocó el fallo y se reanuda la ejecución de la instrucción que provocó el fallo.

Introducción a memoria virtual

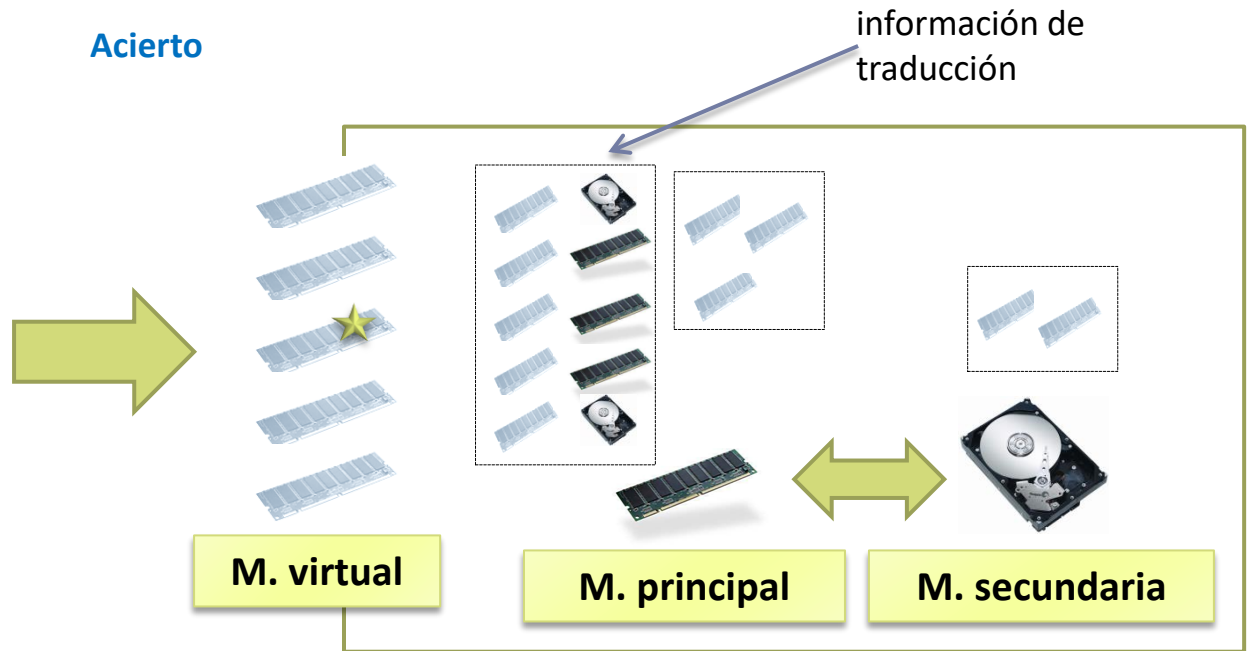
...

lw t0 vector+4

...



Acierto



Memoria virtual: windows

Administrador de tareas

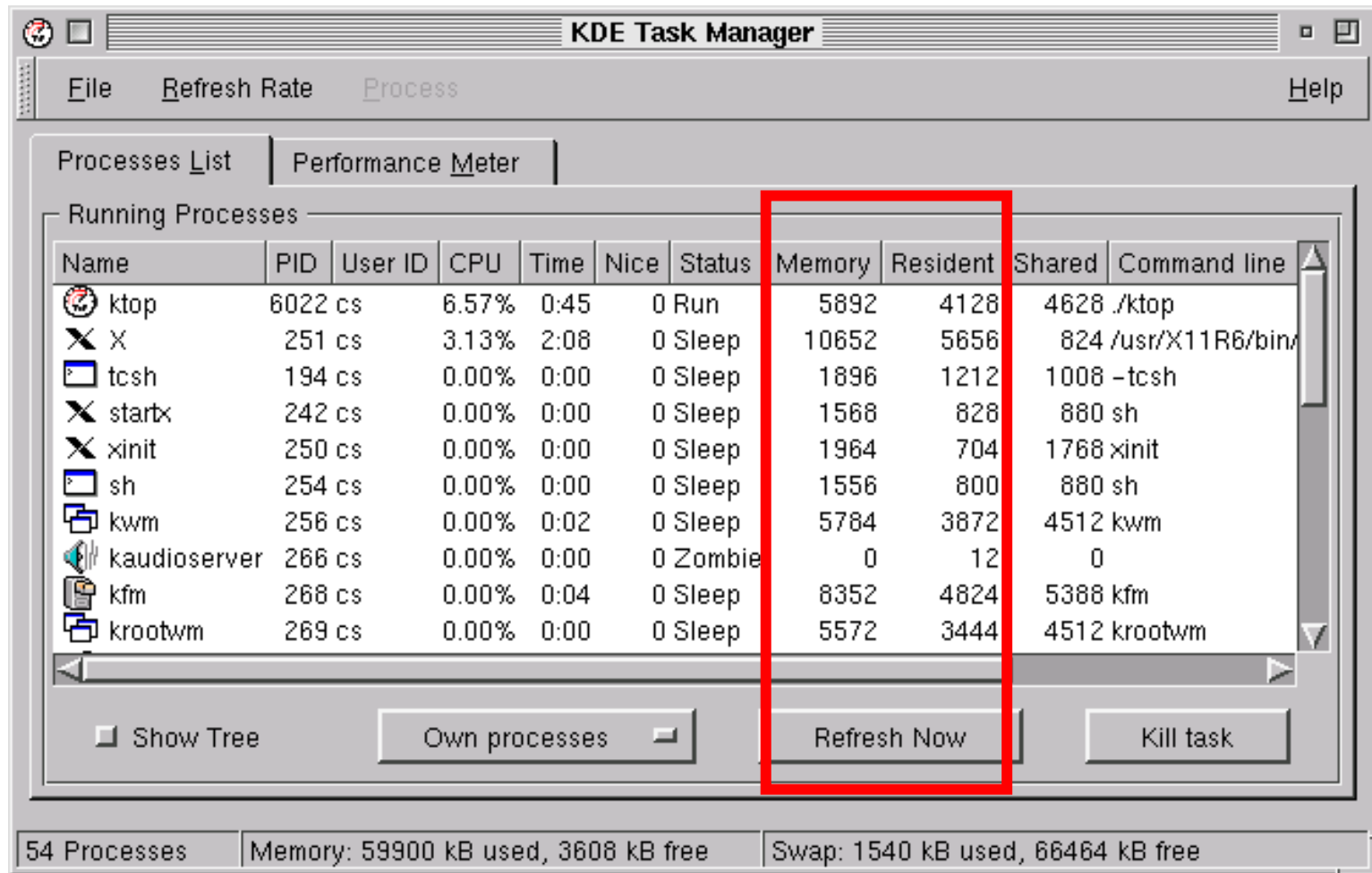
Archivo Opciones Vista

Procesos Rendimiento Historial de aplicaciones Inicio Usuarios Detalles Servicios

Nombre	PID	Estado	CPU	Espacio máxi...	Tamaño de asig...	Bloque paginado	Bloque no pagin...	Errores de página	Bytes ...	Bytes ...	Bytes ...
vmmem	12344	En ejecución	00	1.441.916 K	1.440.732 K	4 K	0 K	361.205	0	0	0
firefox.exe	20192	En ejecución	00	1.453.808 K	952.152 K	1.470 K	287 K	5.985.252	19.026...	7.767....	65.210...
firefox.exe	2284	En ejecución	00	619.744 K	548.144 K	584 K	105 K	650.839	50.095...	34.927...	29.014
firefox.exe	20324	En ejecución	00	723.144 K	1.049.092 K	1.624 K	204 K	5.364.079	3.751....	120.55...	174.572
explorer.exe	10248	En ejecución	00	657.404 K	258.764 K	1.828 K	181 K	3.020.867	1.475....	599.46...	157.72...
firefox.exe	15728	En ejecución	00	625.320 K	388.416 K	765 K	96 K	2.613.652	114.66...	208.82...	36.550
firefox.exe	19876	En ejecución	00	507.448 K	333.428 K	845 K	91 K	3.542.574	169.70...	1.005....	34.022
firefox.exe	18772	En ejecución	00	417.088 K	266.080 K	634 K	69 K	1.174.722	267.42...	1.014....	29.824
firefox.exe	20032	En ejecución	00	404.616 K	266.908 K	595 K	69 K	1.303.844	24.727...	66.542...	27.760
OmenCommandCent...	14124	En ejecución	00	310.596 K	235.660 K	1.119 K	120 K	3.394.989	174.98...	10.743...	88.235...
firefox.exe	20692	En ejecución	00	504.408 K	238.300 K	648 K	70 K	1.202.506	42.110...	126.64...	29.770
POWERPNT.EXE	13268	En ejecución	00	310.764 K	241.548 K	4.016 K	91 K	162.819	16.575...	119.046	830.478
SearchApp.exe	11348	Suspendido	00	278.592 K	193.240 K	1.143 K	120 K	287.989	40.963...	23.637...	1.916....
MsMpEng.exe	5952	En ejecución	00	893.472 K	302.164 K	588 K	102 K	3.688.020	18.314...	613.47...	1.207....
firefox.exe	14576	En ejecución	00	262.540 K	184.328 K	597 K	51 K	292.382	45.066...	23.459...	27.512
NVIDIA Broadcast.exe	4268	En ejecución	00	224.456 K	681.952 K	1.529 K	36 K	147.585	5.628....	2.329...	344.48...
firefox.exe	11516	En ejecución	00	257.252 K	173.380 K	582 K	48 K	284.367	31.678...	92.144...	27.664
firefox.exe	8736	En ejecución	00	262.104 K	169.856 K	683 K	51 K	288.369	41.391...	128.82...	28.728
firefox.exe	20020	En ejecución	00	245.512 K	158.676 K	637 K	47 K	1.133.089	40.375...	47.792...	28.448
firefox.exe	23004	En ejecución	00	267.684 K	146.736 K	597 K	51 K	157.884	14.799...	11.489...	27.646
dwm.exe	1832	En ejecución	01	362.268 K	502.292 K	872 K	91 K	1.134.978	72.124	207	1.226....

Menos detalles Finalizar tarea

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

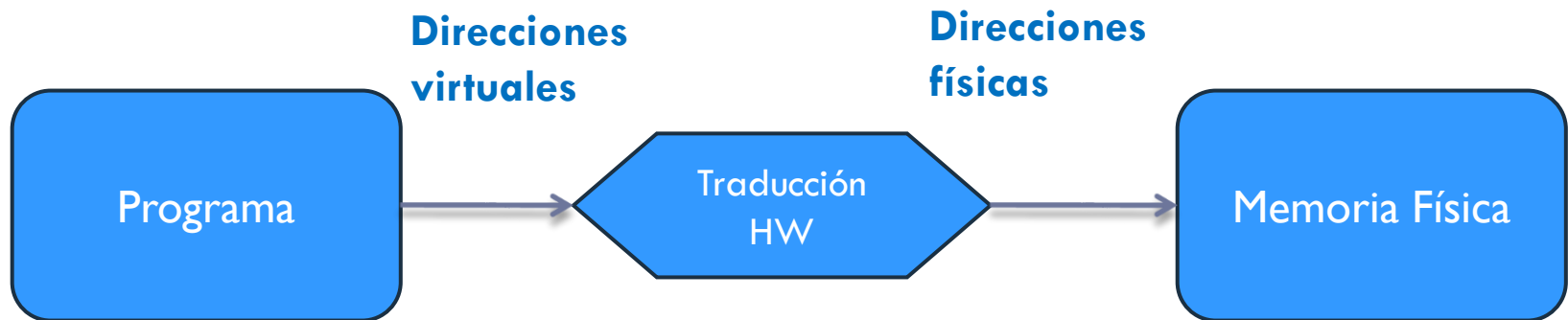
Resumen

- ▶ Las direcciones que genera el procesador son **direcciones virtuales**
- ▶ El espacio de direcciones virtuales se divide en trozos de igual tamaño denominado **páginas**
- ▶ 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**

Memoria virtual paginada

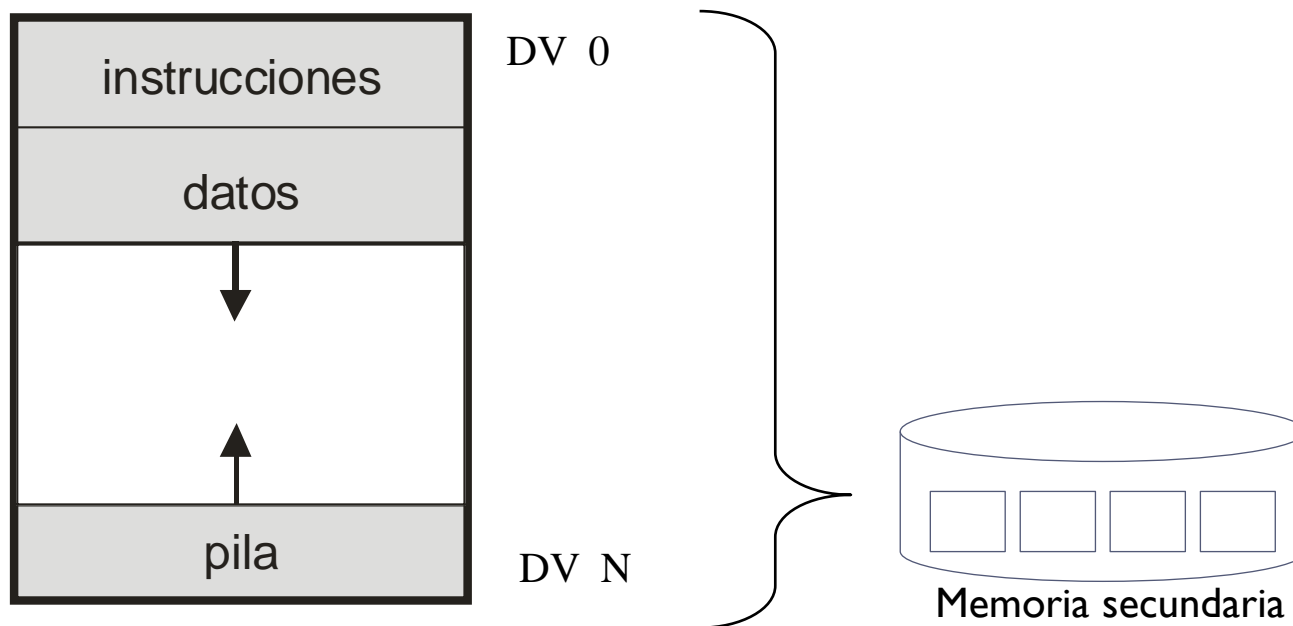
Traducción

- ▶ 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.

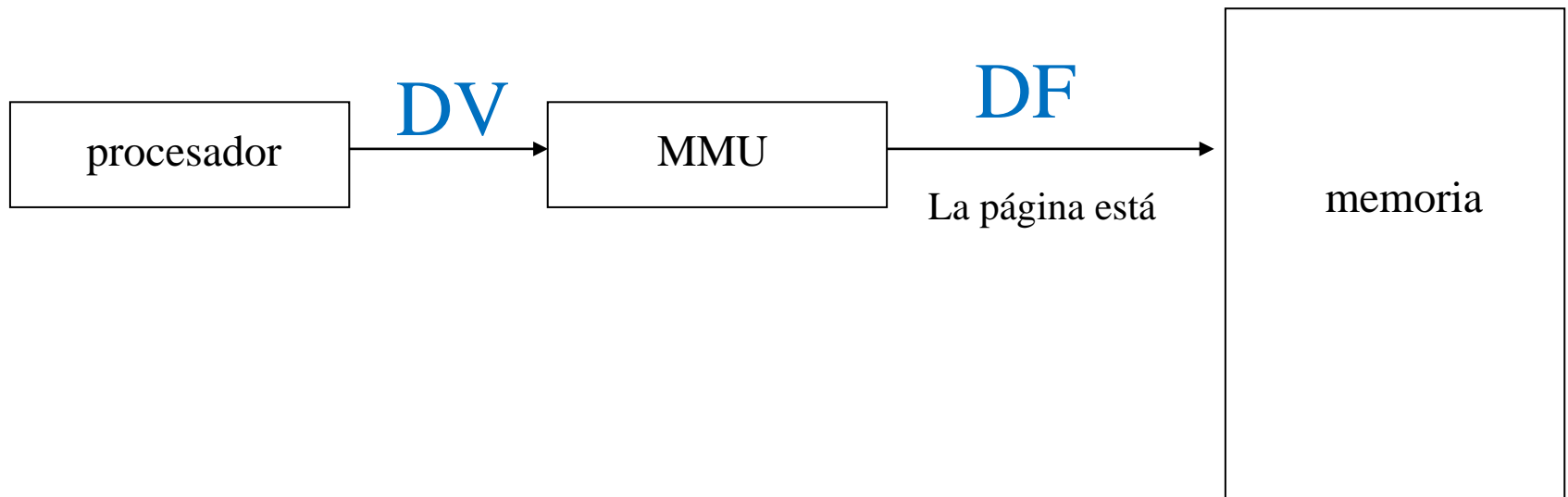


Memoria virtual paginada

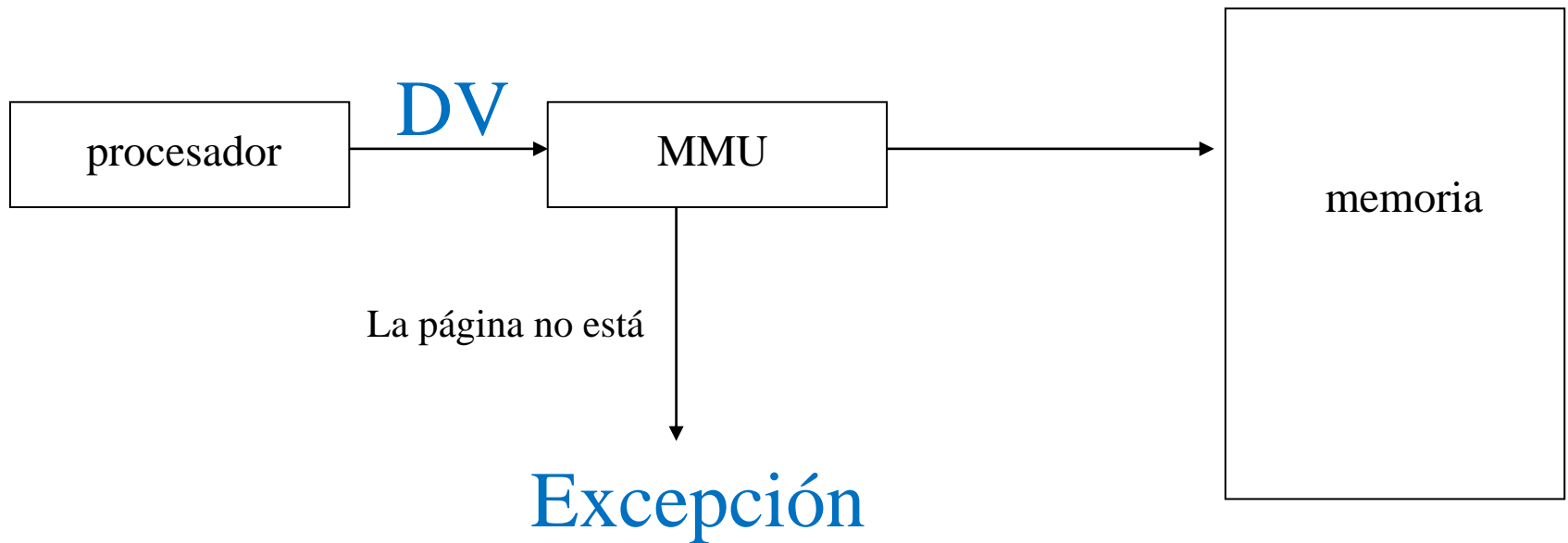
- ▶ La imagen de memoria de los procesos reside inicialmente en disco



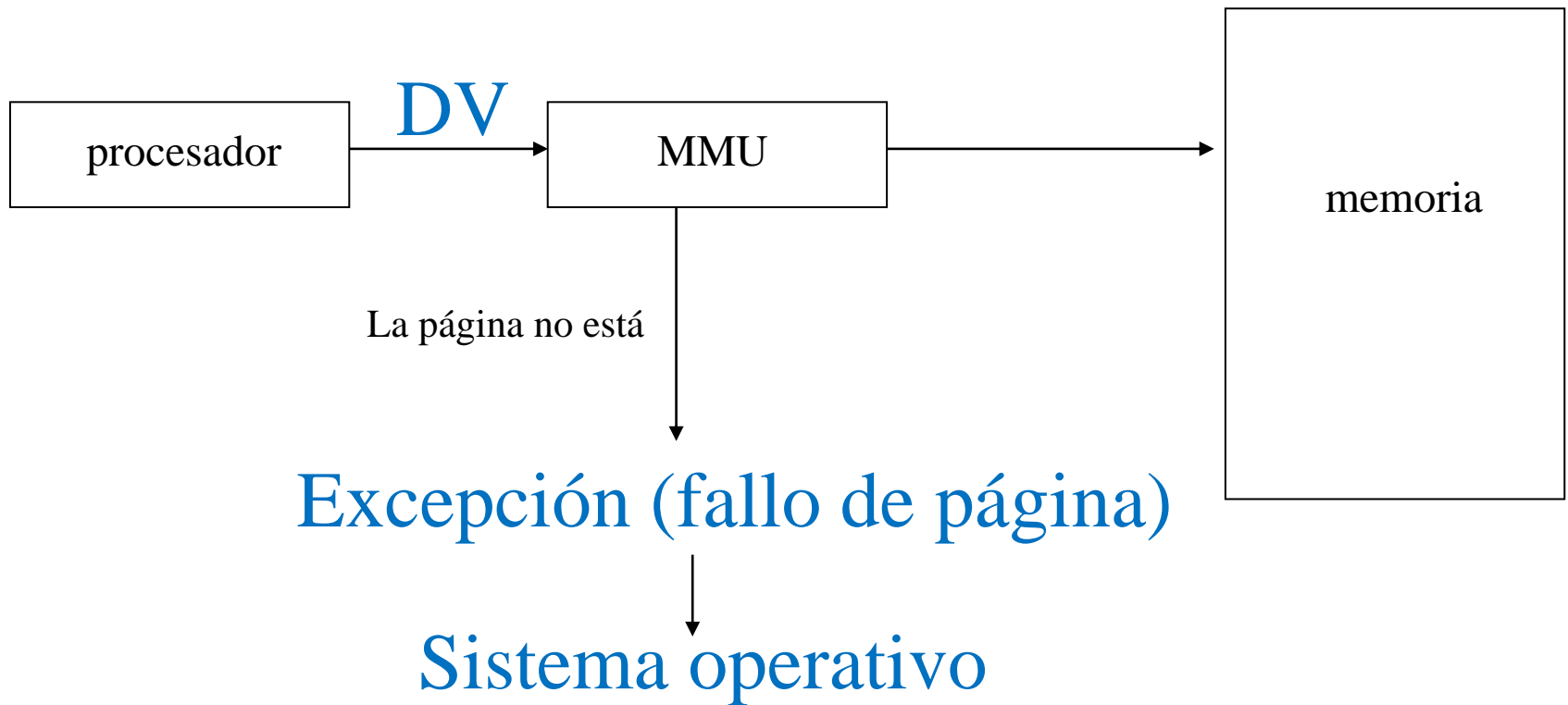
Traducción de direcciones



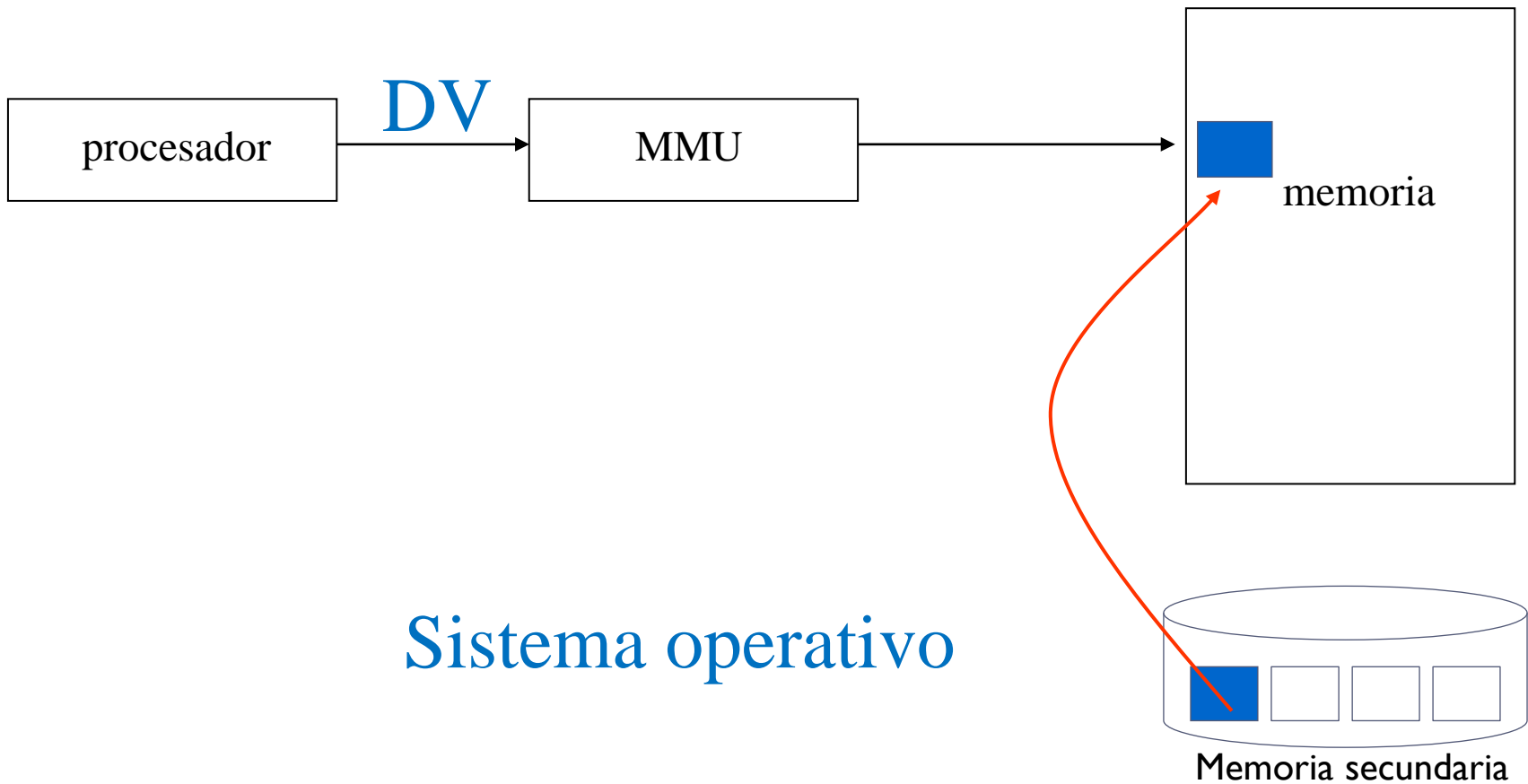
Traducción de direcciones



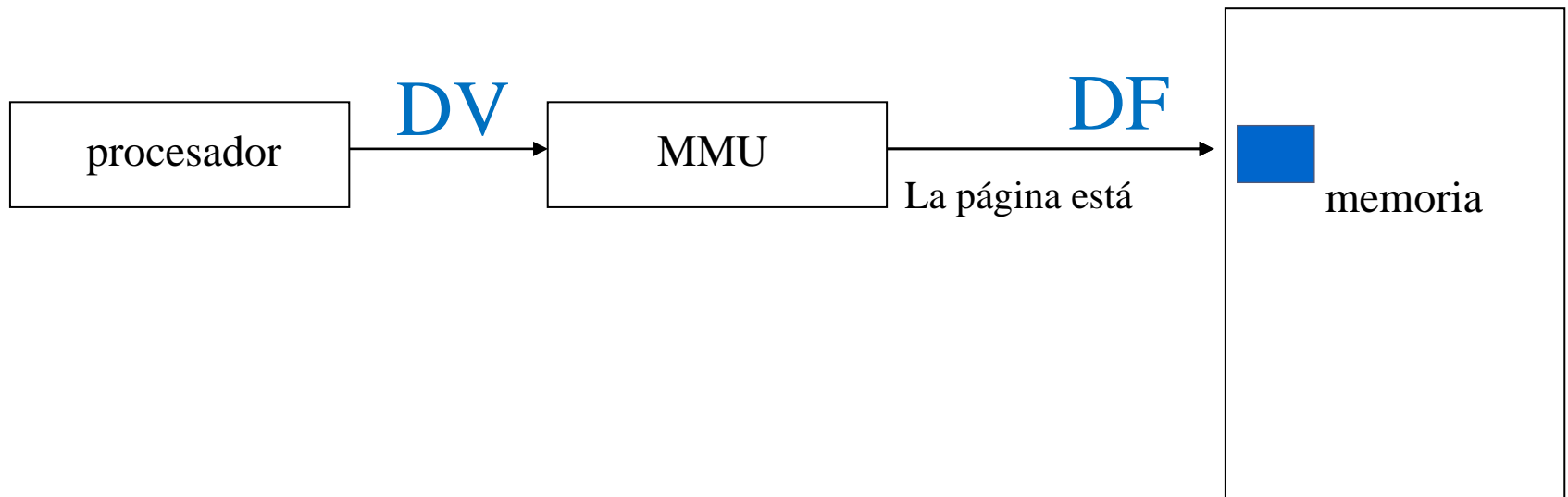
Traducción de direcciones



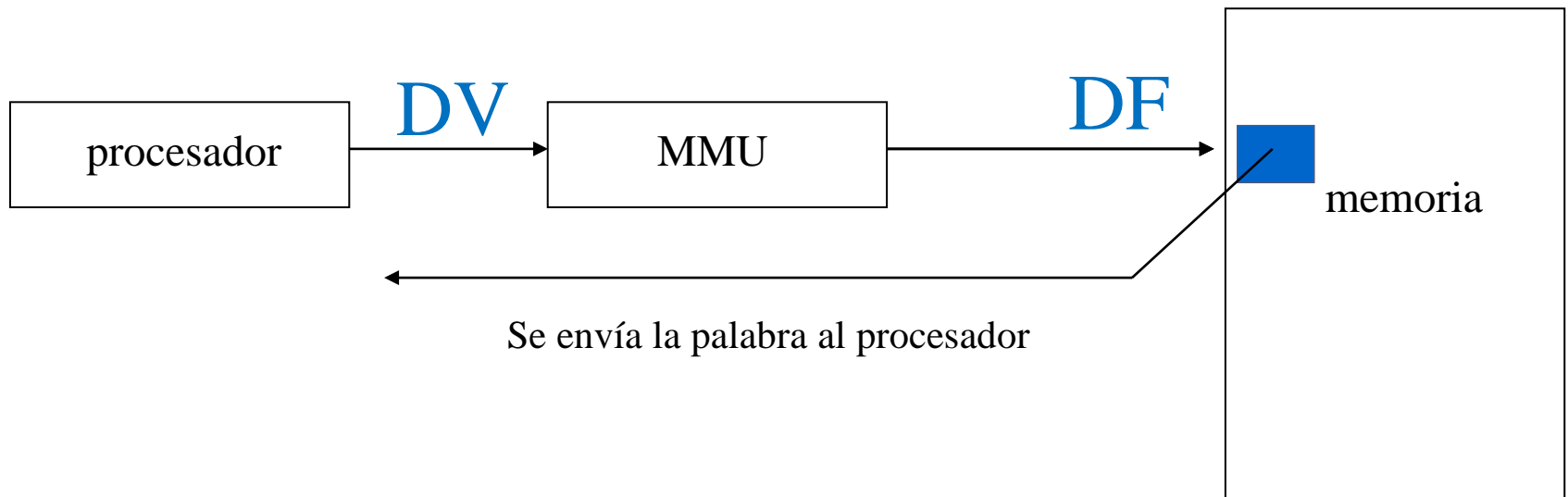
Traducción de direcciones



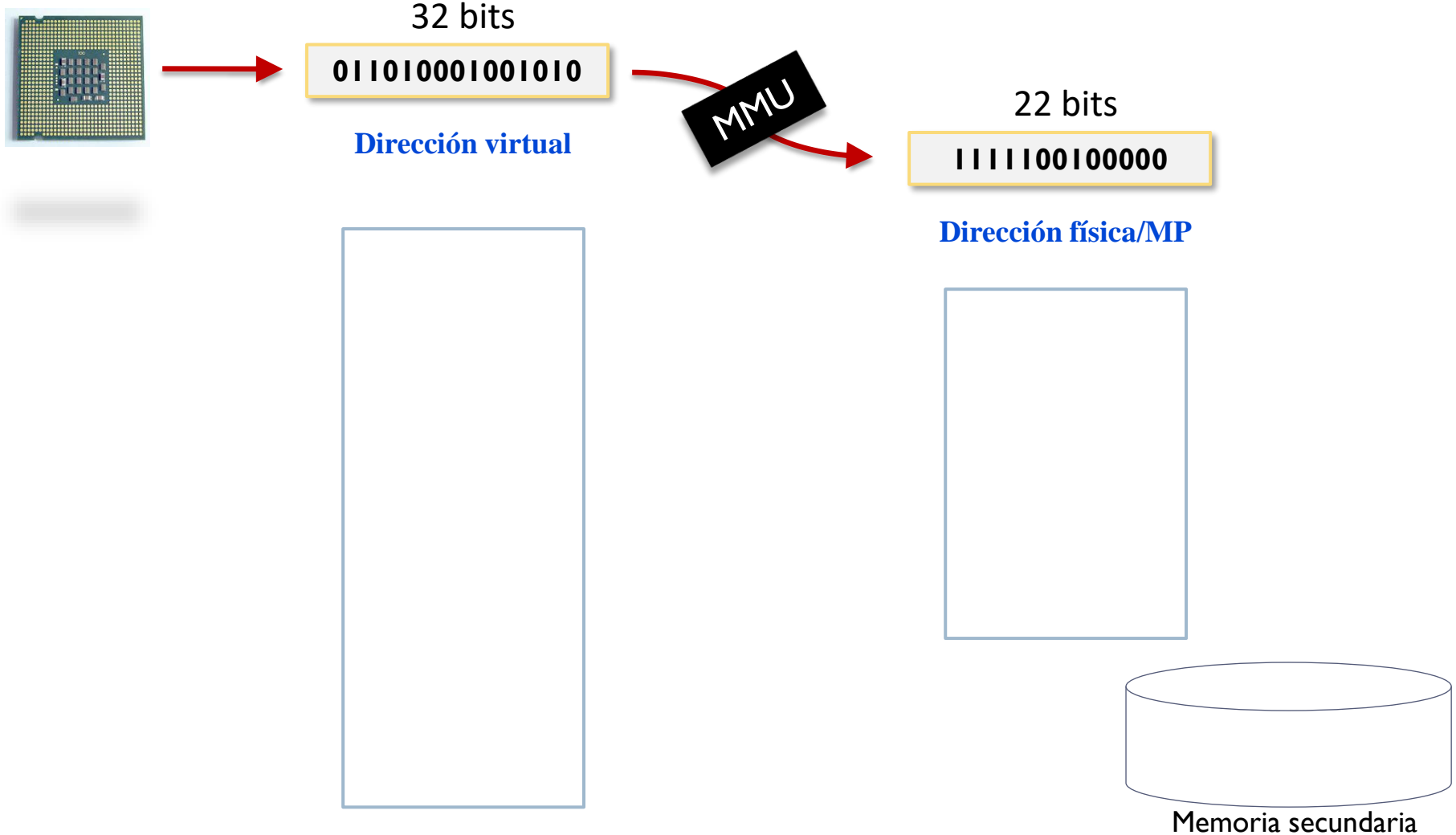
Traducción de direcciones



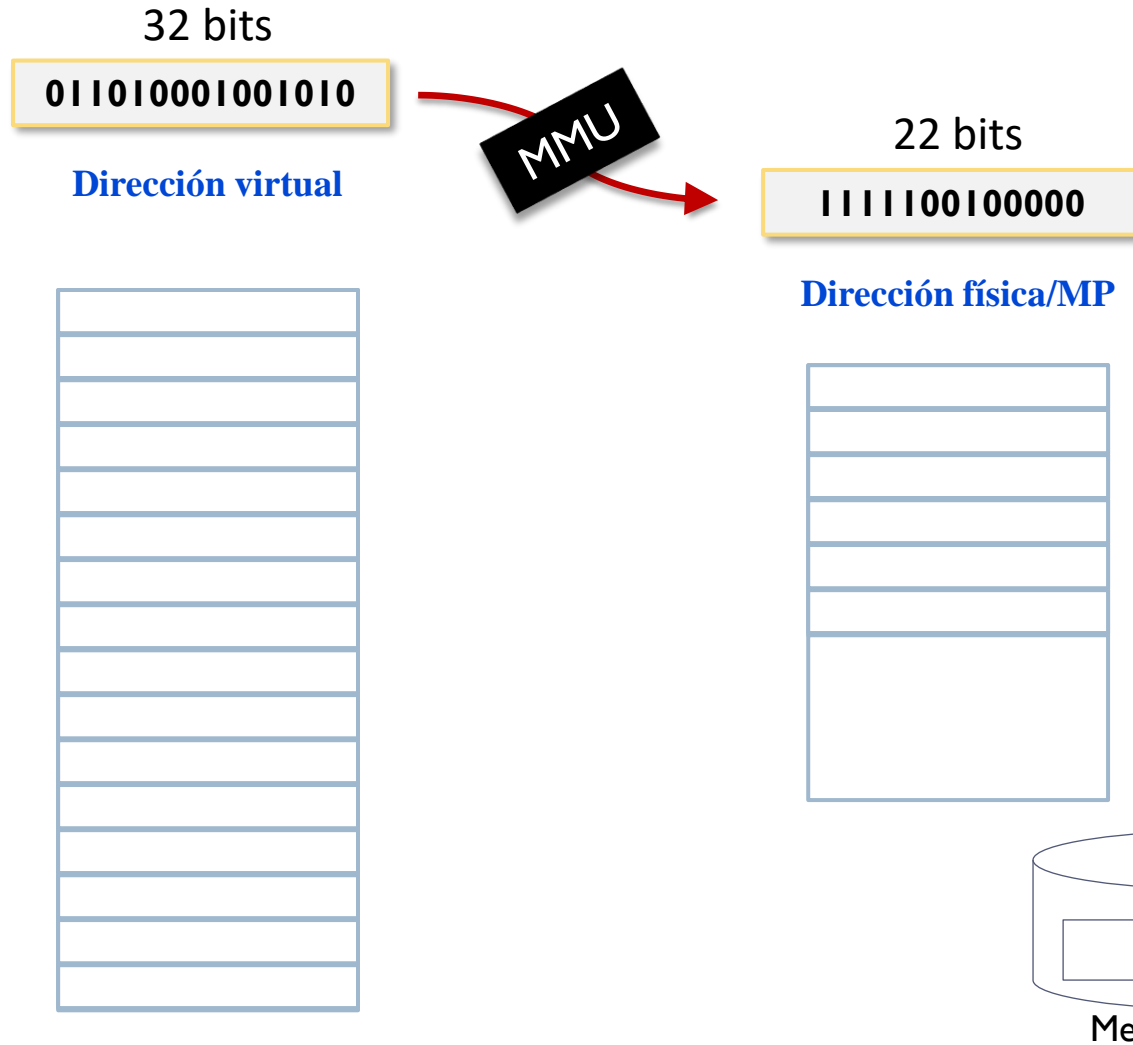
Traducción de direcciones



Ejemplo

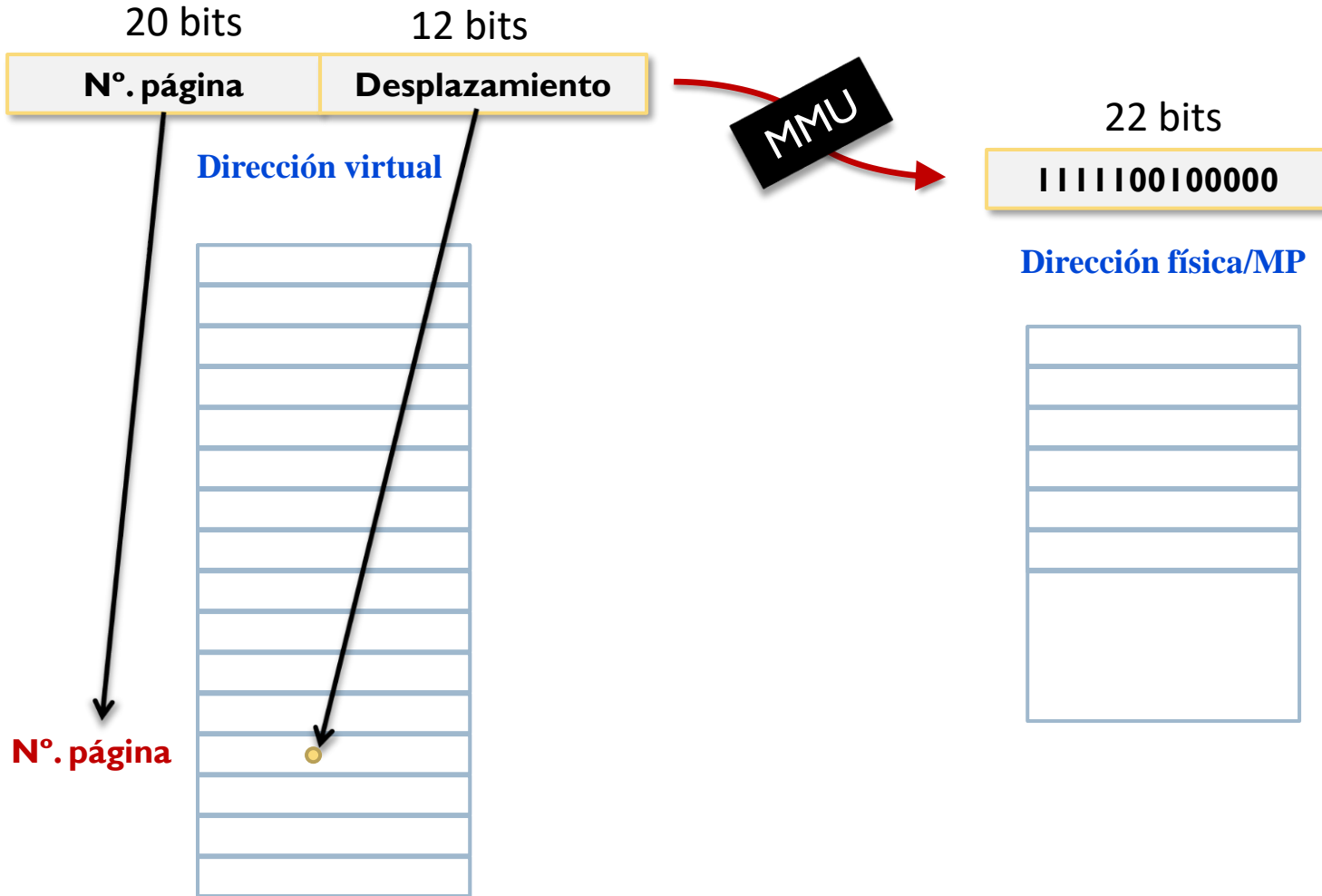


Ejemplo



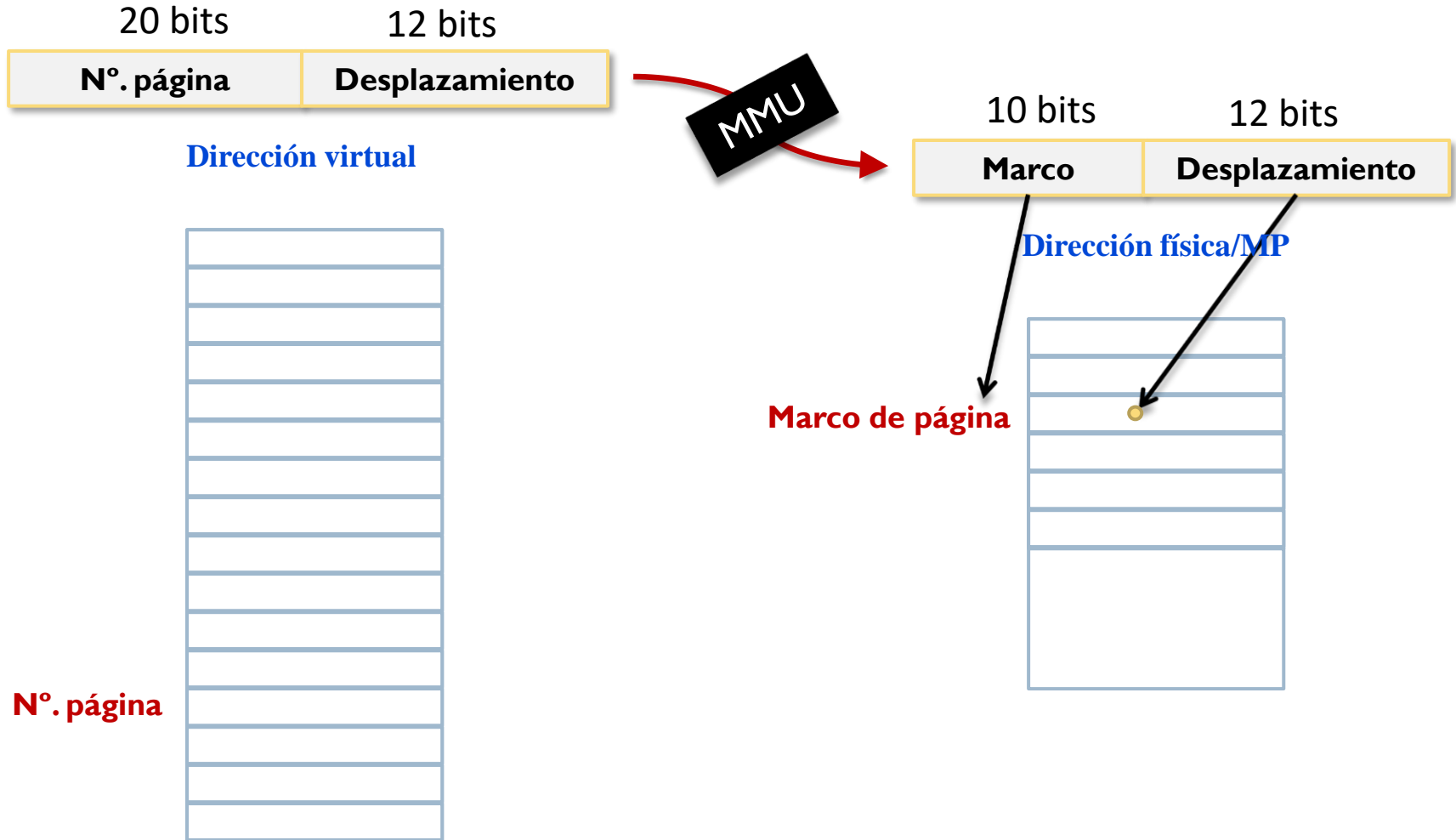
División en bloques del mismo tamaño -> páginas

Ejemplo



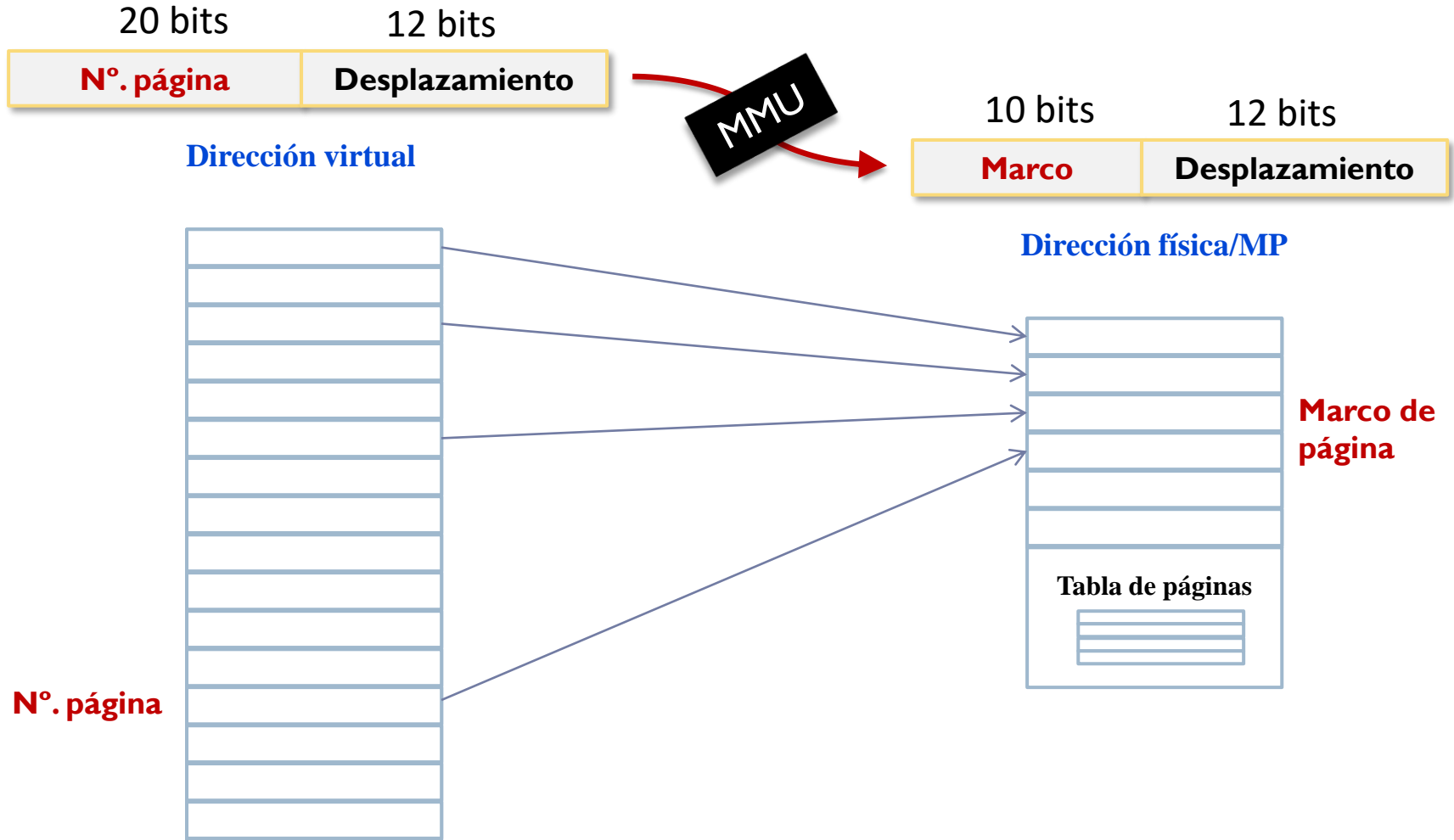
División en bloques del mismo tamaño -> páginas

Ejemplo

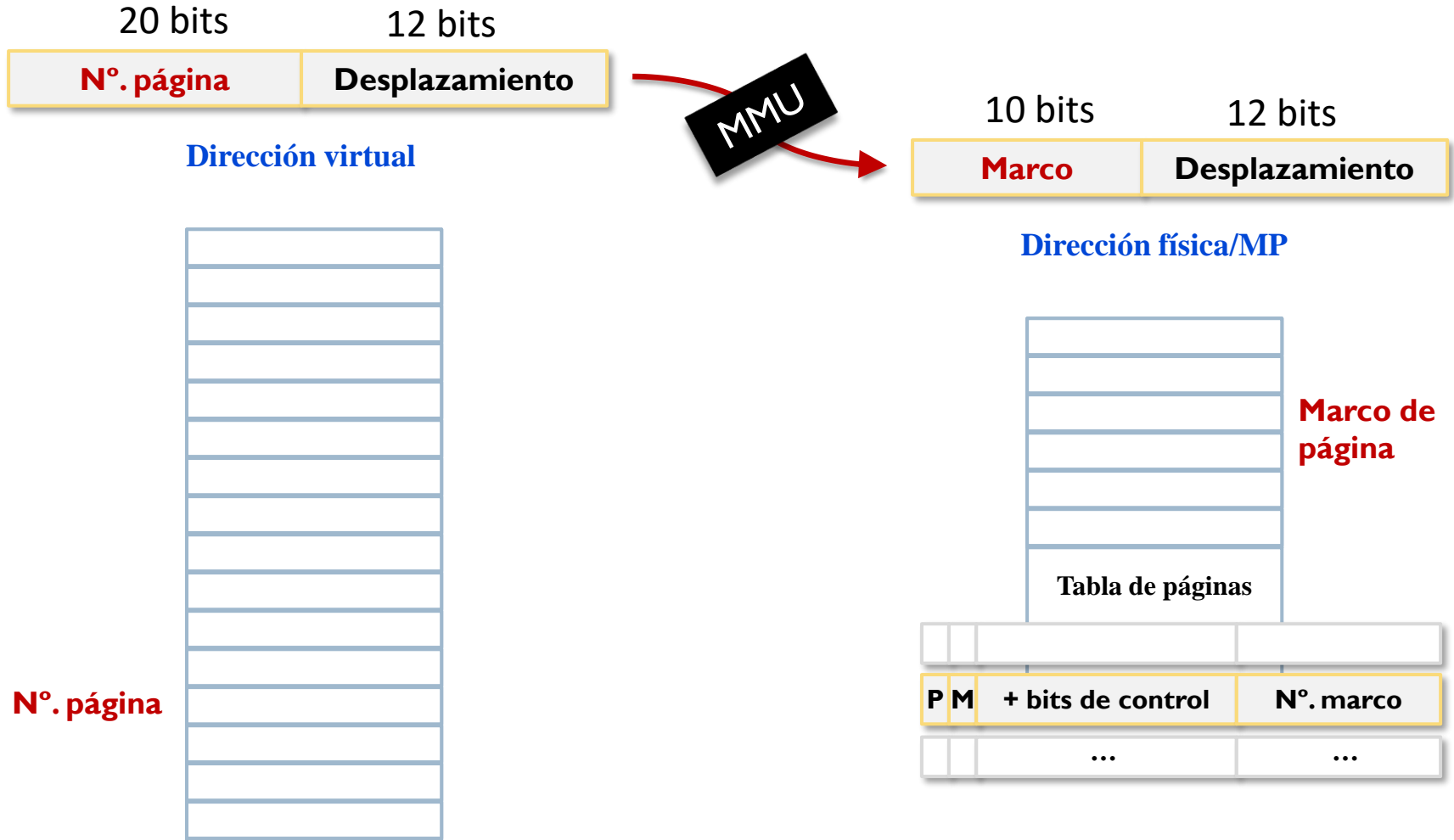


División en bloques del mismo tamaño -> páginas

Ejemplo



Ejemplo



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

Ejemplo

20 bits

12 bits

Nº. página

Desplazamiento

Dirección virtual



10 bits

12 bits

Marco

Desplazamiento

Dirección física/MP

Nº. página 10

10

Bit "presente"

Marco de página

Tabla de páginas

09

10

11

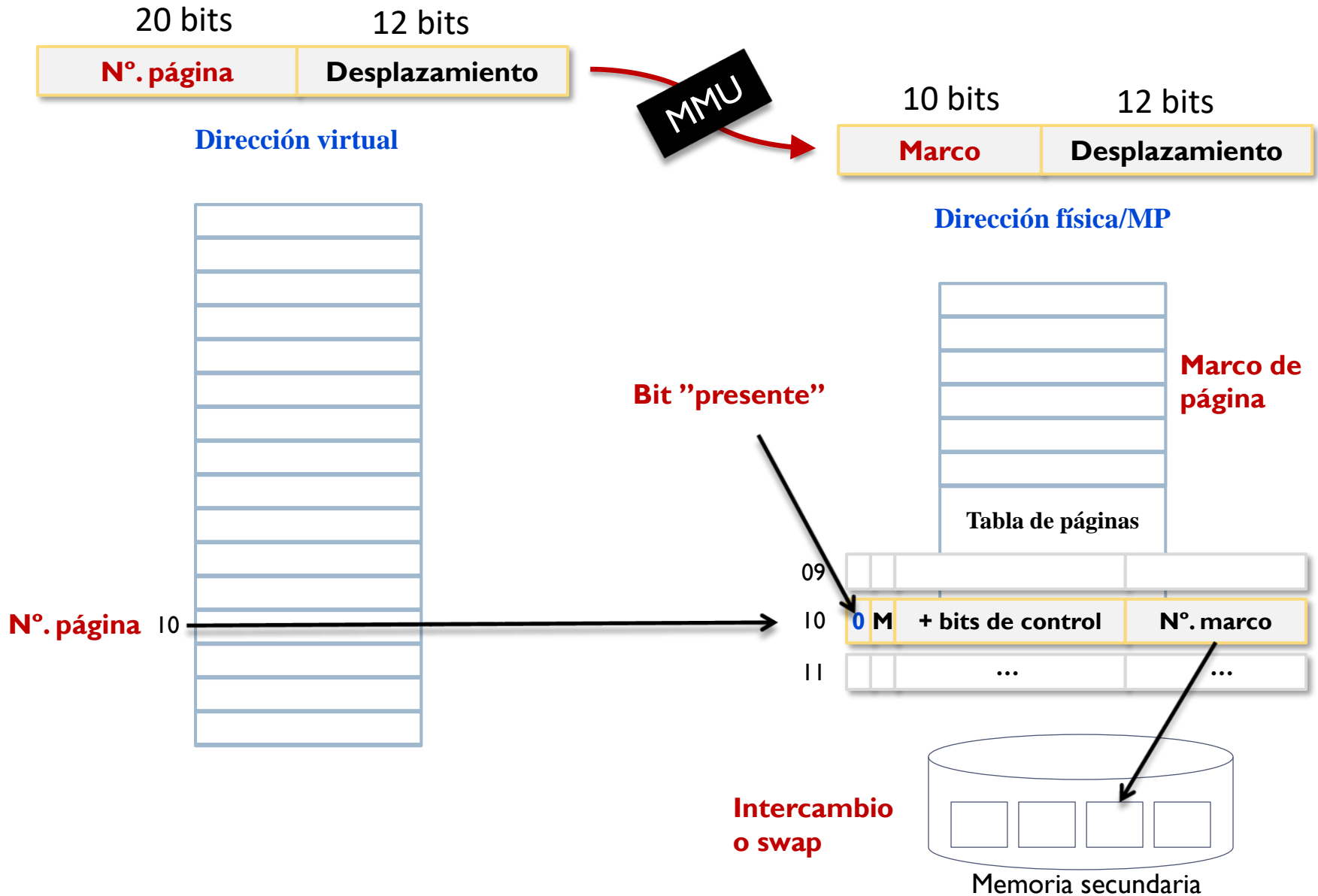
+ bits de control

N°. marco

...

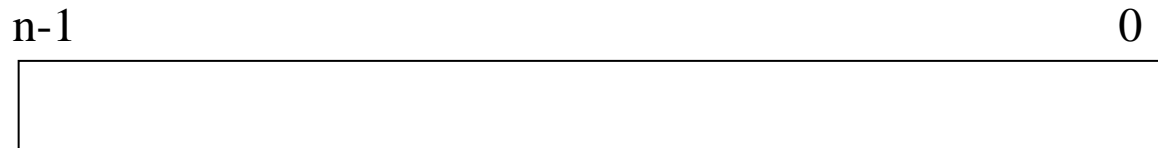
...

Ejemplo



Estructura de una dirección virtual

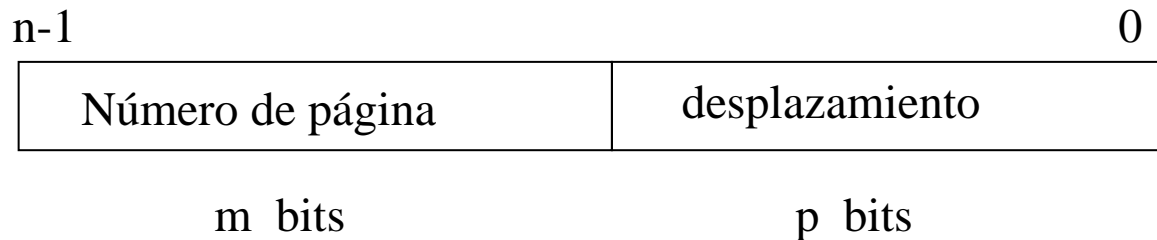
- ▶ Un computador de n bits tiene:
 - ▶ Direcciones de n bits



- ▶ 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

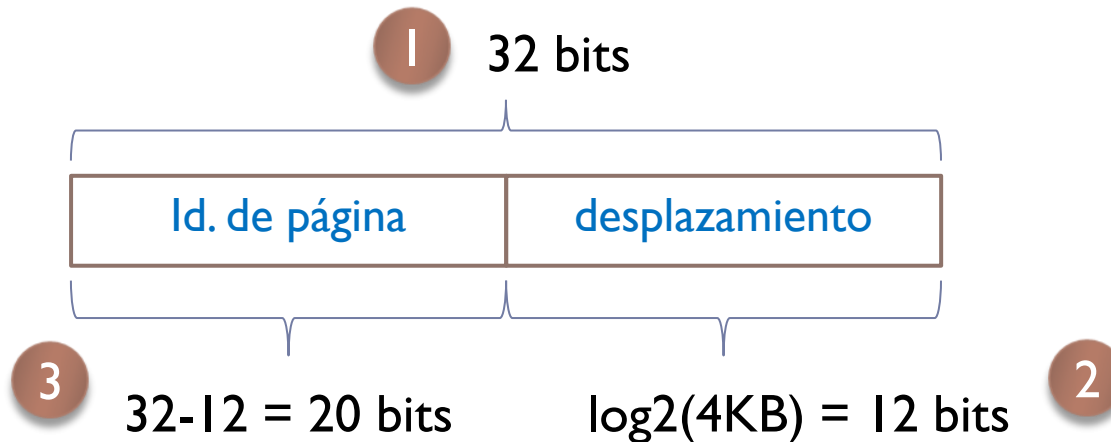
Ejercicio

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:

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

Memoria virtual paginada

Tablas de páginas

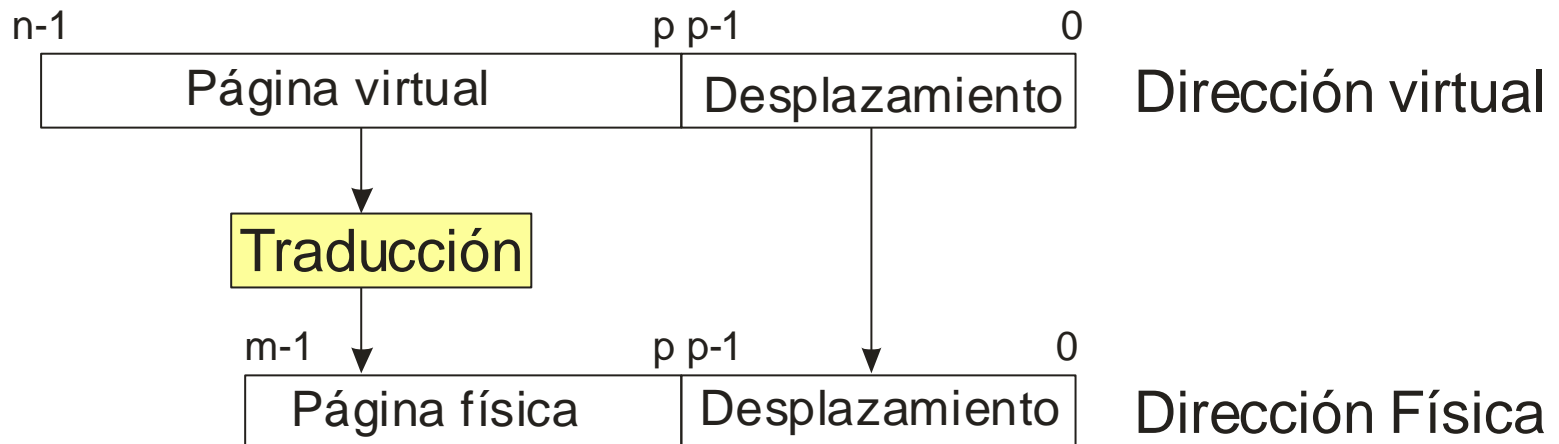
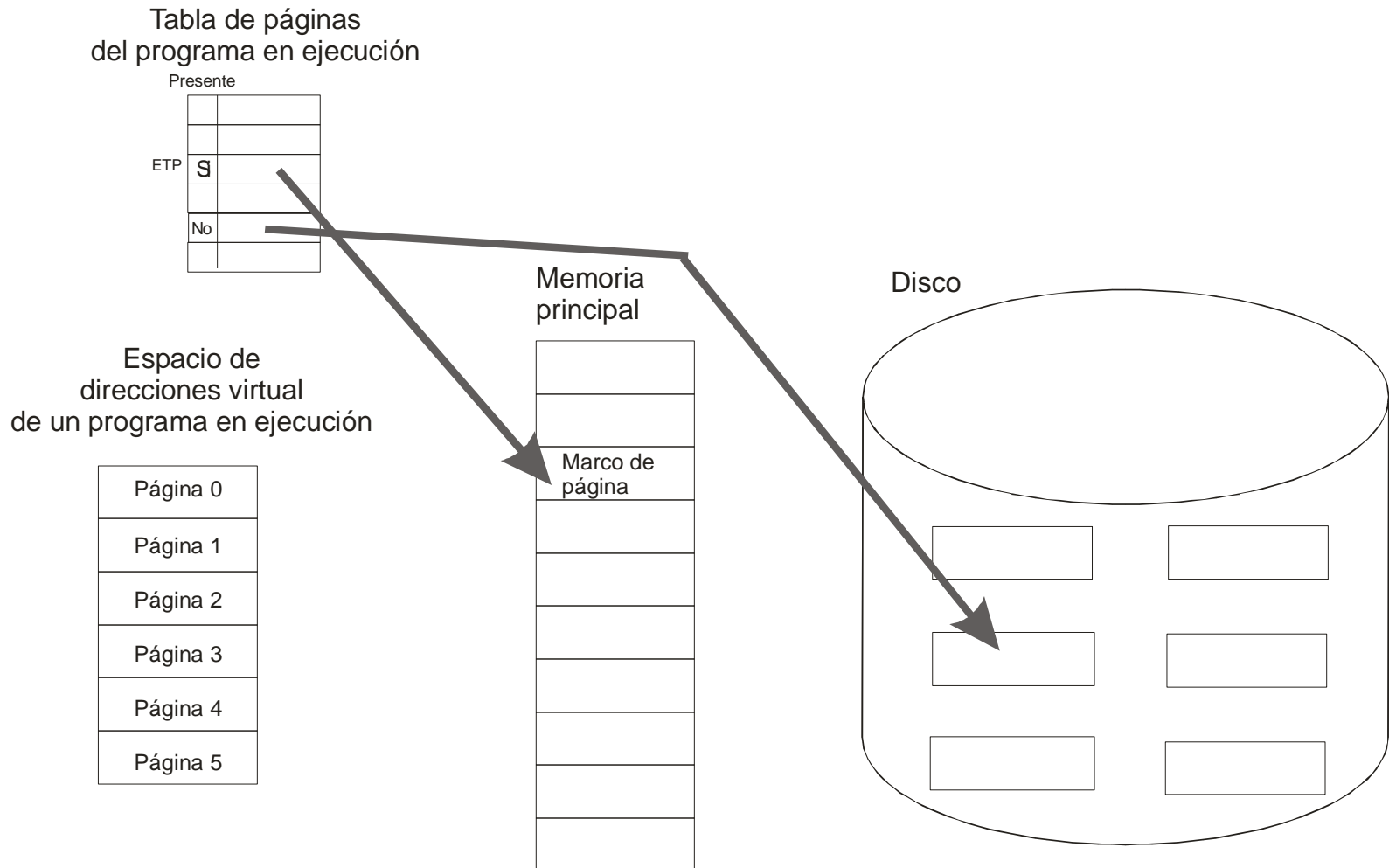
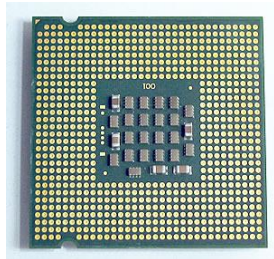


Tabla de páginas



Entradas de la tabla de páginas (formato típico)



Dirección virtual

20 bits

12 bits

Número de página

Desplazamiento

Entrada de la tabla de páginas

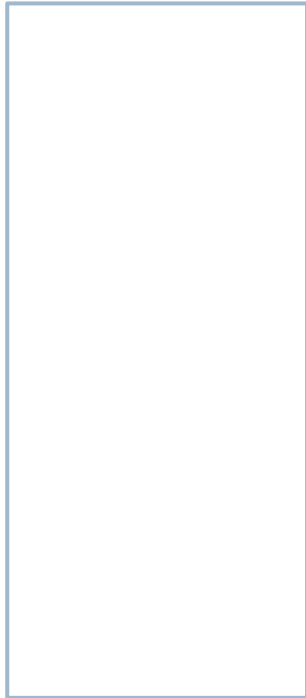
P	M	Otros bits de control	Número de marco

- 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.)

Estructura de la tabla de páginas

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

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
	Pág. 5
	Pag. 6
	Pag. 7
Pila	

- ▶ 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
	Pág. 5
	Pag. 6
	Pag. 7
Pila	

- ▶ 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
	Pág. 5
	Pag. 6
	Pag. 7
Pila	

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
	Pag. 7
Pila	

	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

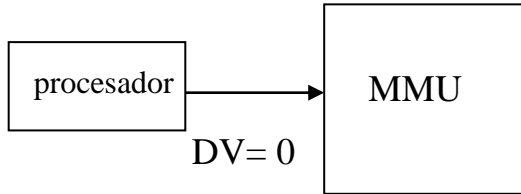
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

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
	Pag. 7
Pila	

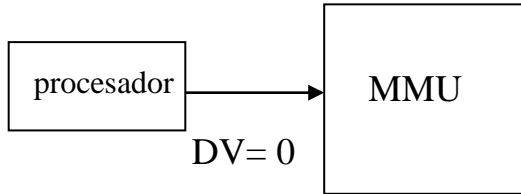
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

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
	Pág. 5
	Pag. 6
	Pag. 7
Pila	

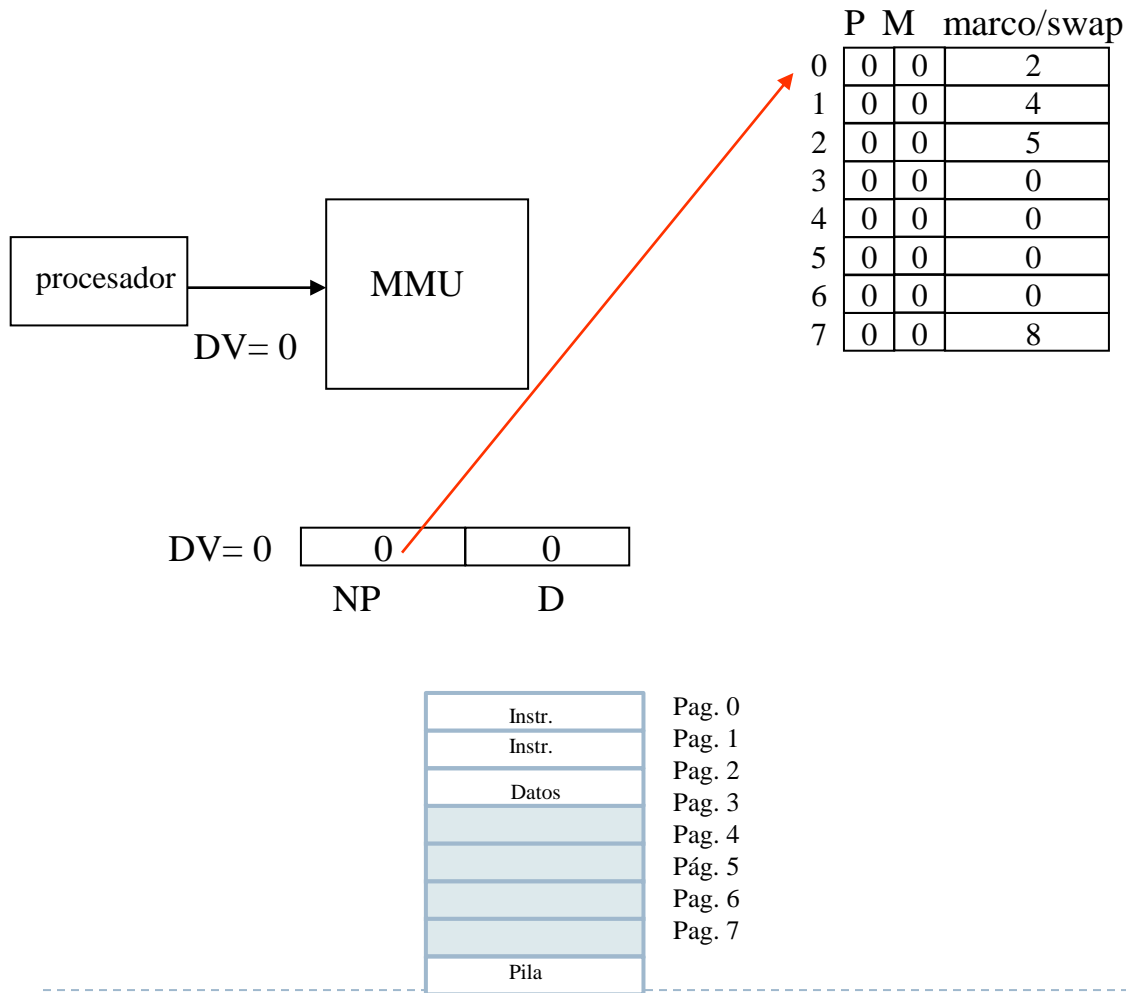
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

Swap

Páginas del proceso

Ejemplo

Acceso a la DV 0



Fallo de página
La página 0 no está en memoria

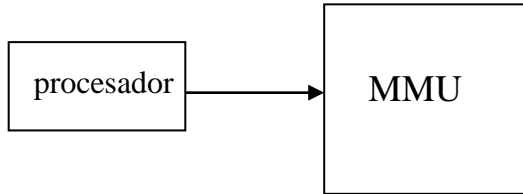
Swap

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

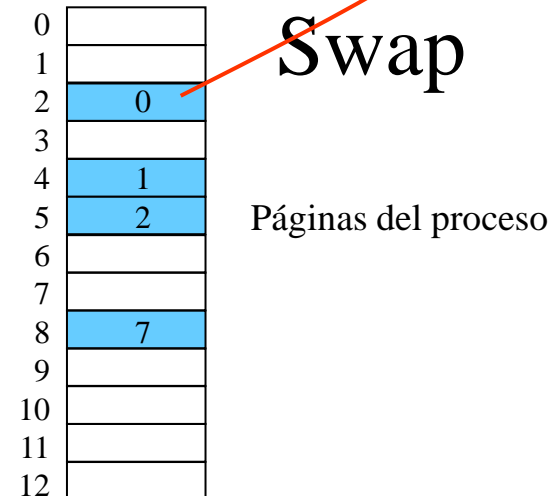
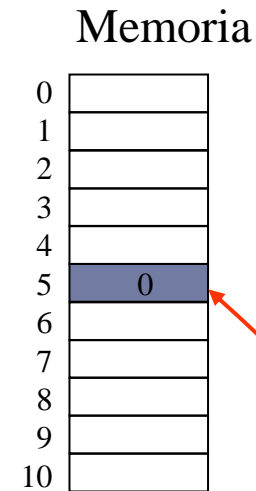


DV= 0

0	0
---	---

NP D

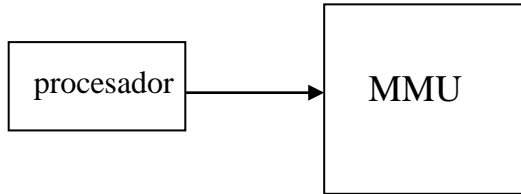
	P	M	marco/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8



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



	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

El SO actualiza la tabla de páginas

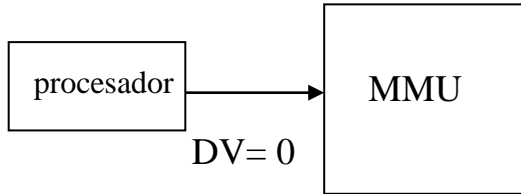
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

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

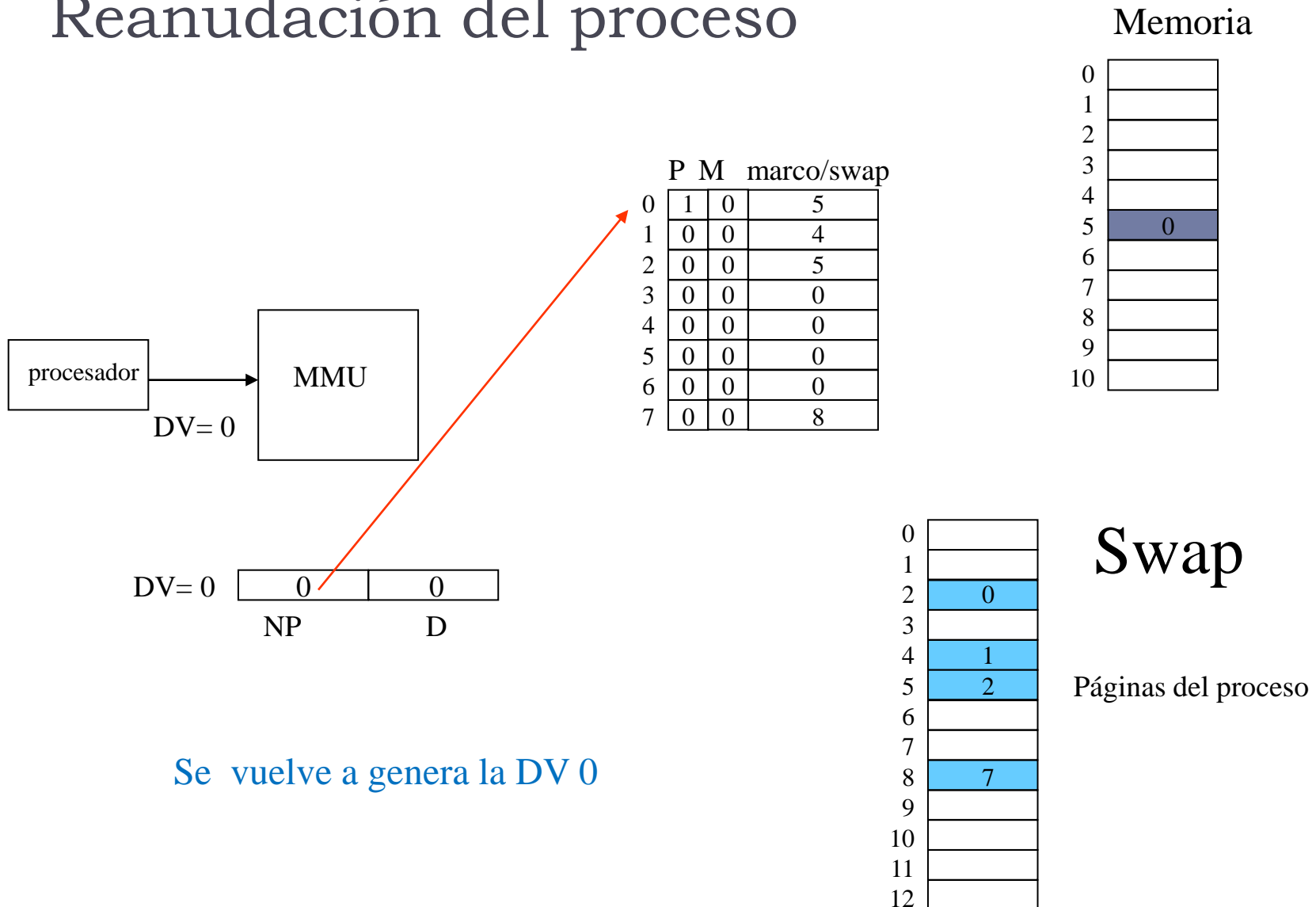
0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

Swap

Páginas del proceso

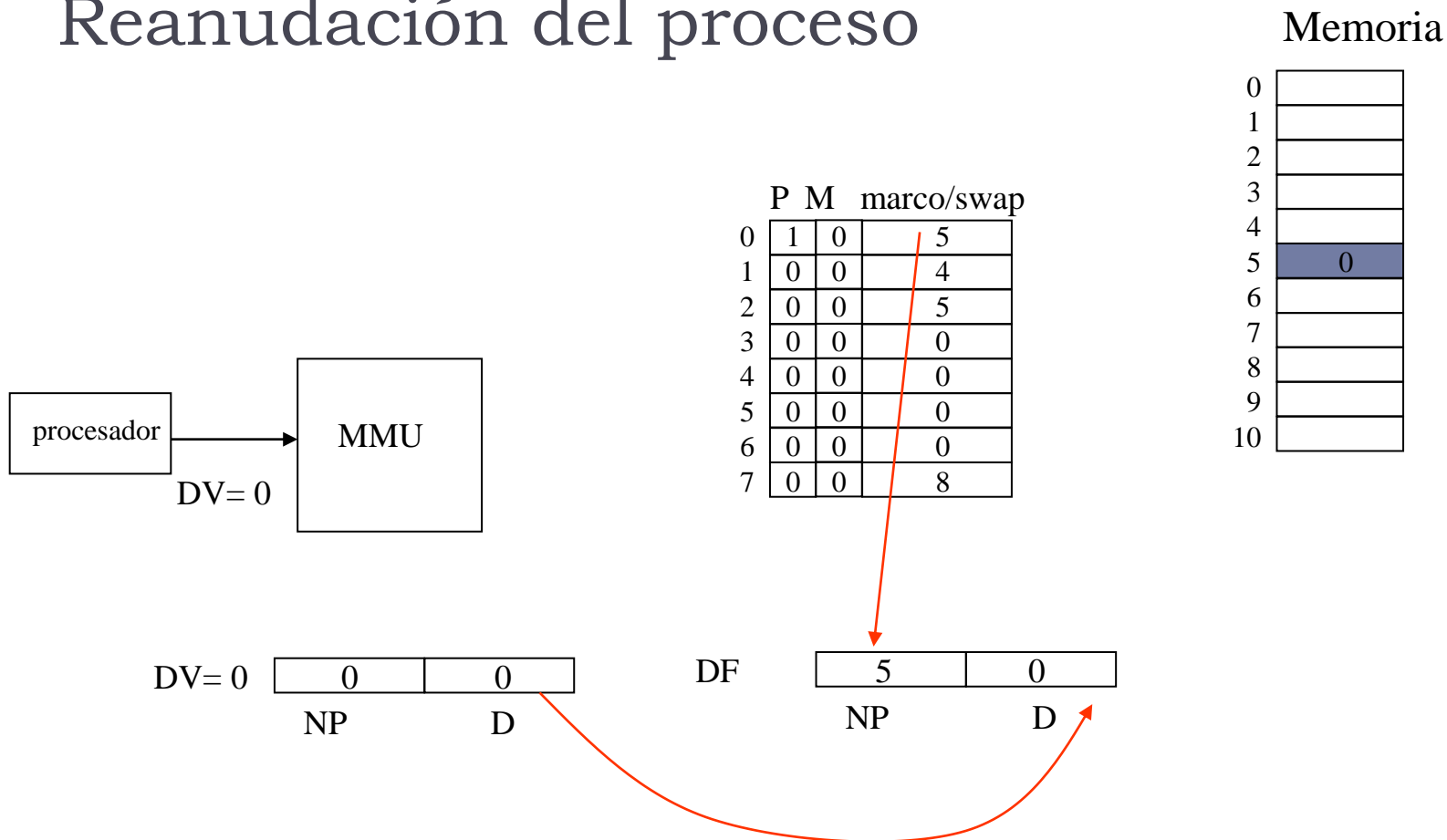
Ejemplo

Reanudación del proceso



Ejemplo

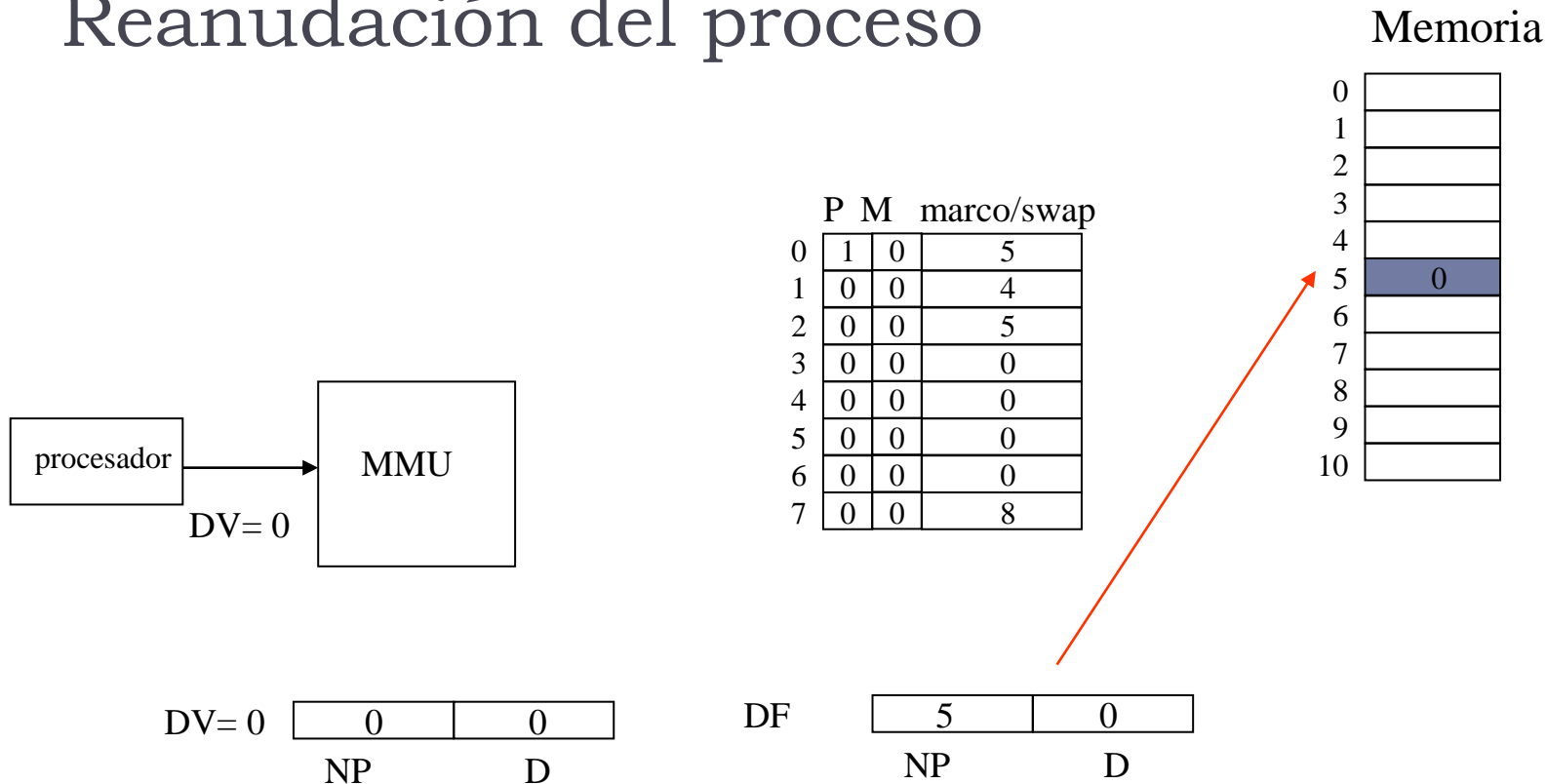
Reanudación del proceso



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.

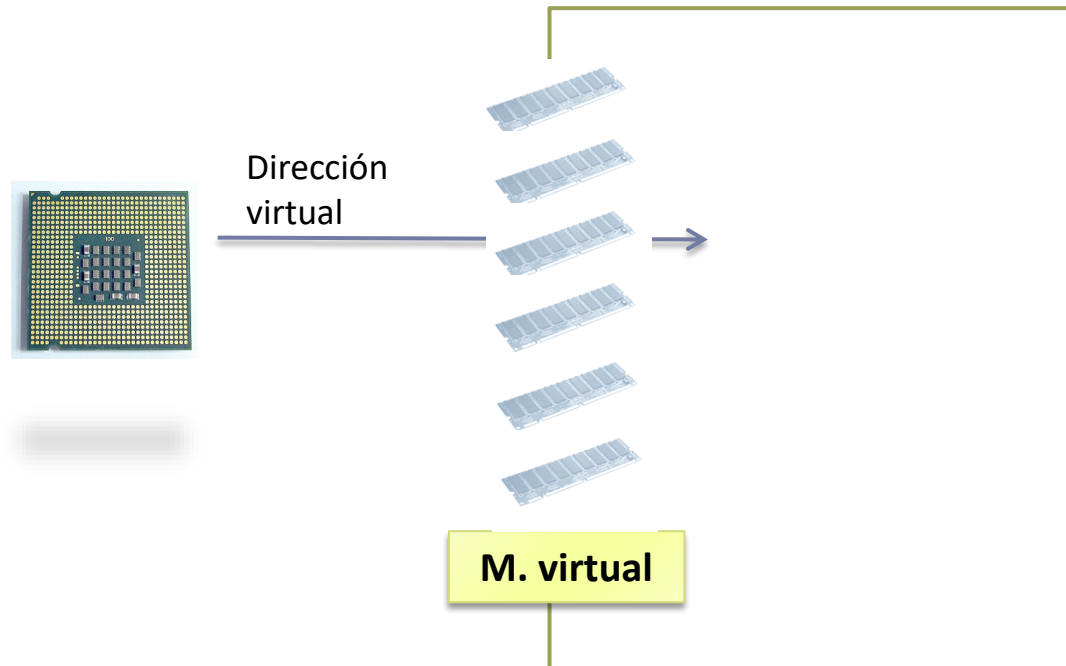
Cada entrada de la tabla de páginas requiere de 32 bits.

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?

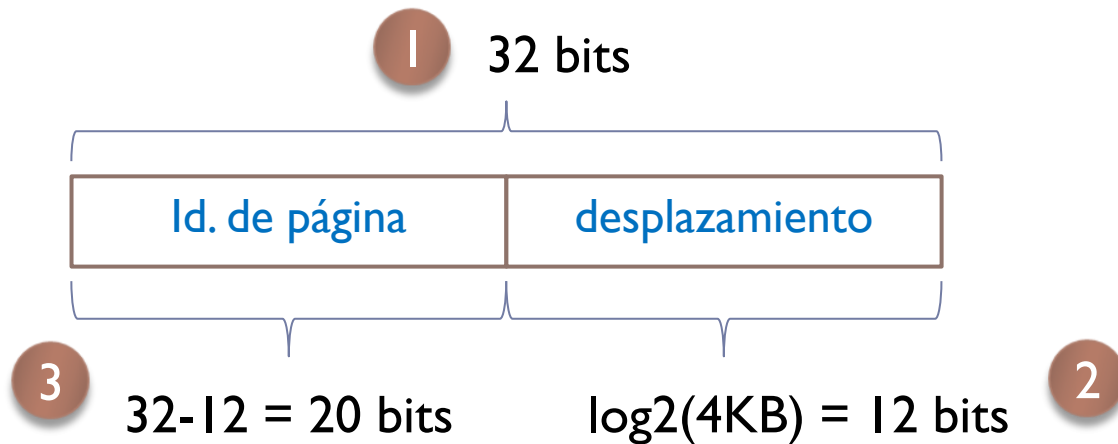
Ejercicio (solución)

- ▶ 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)

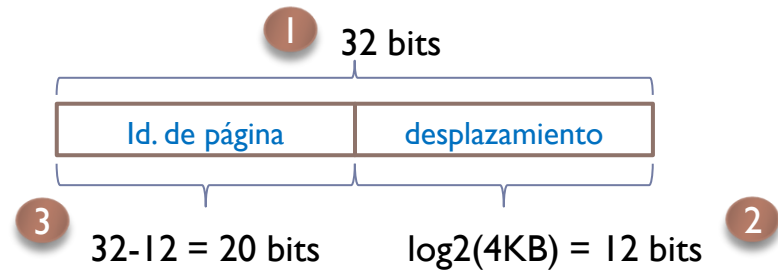
P	M	Perm.	Marco/ Bloque
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- ▶ 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
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	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

► Inicialmente:

- La **crea** el sistema operativo cuando se va a ejecutar el programa.

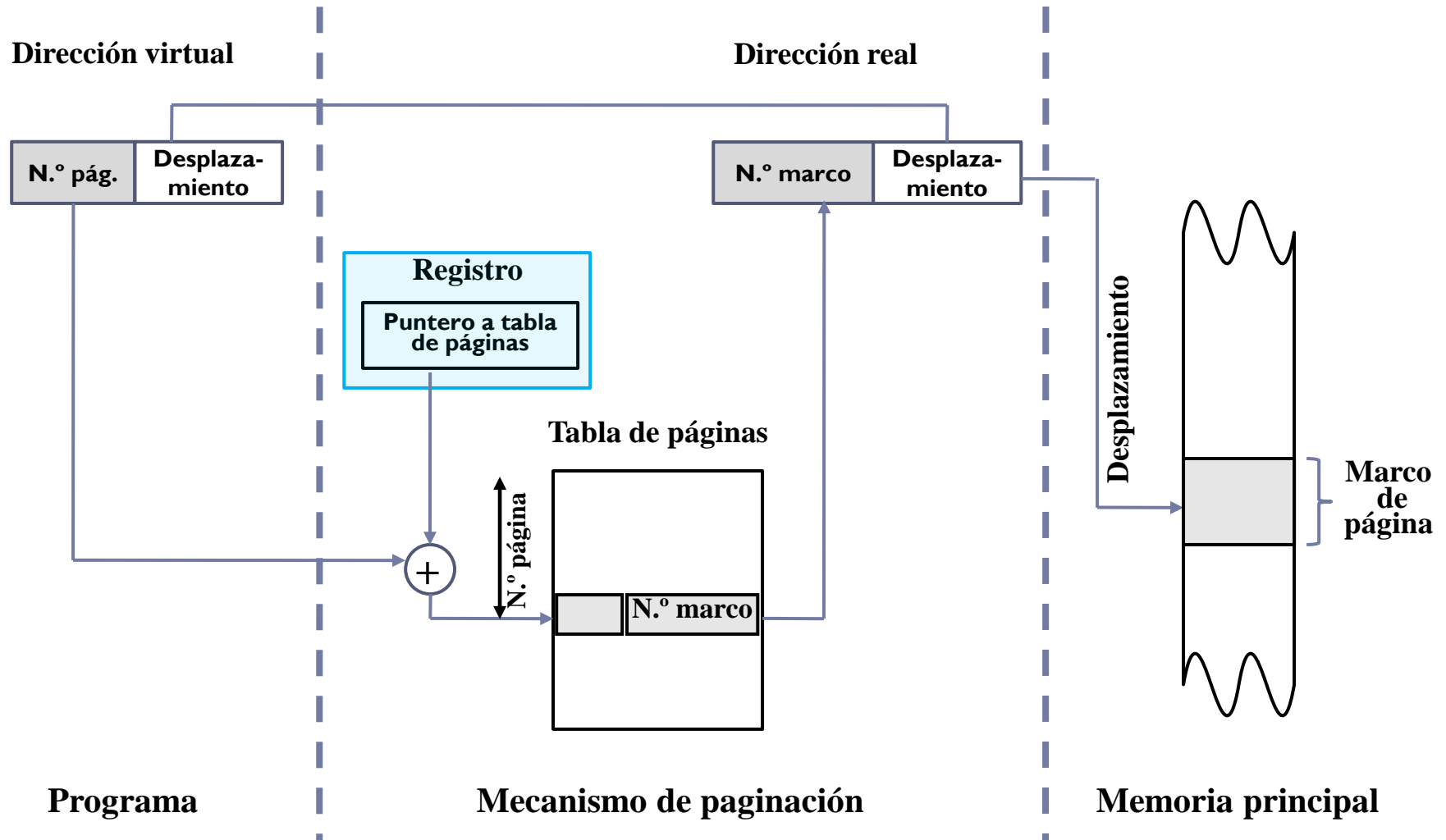
► Uso:

- 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

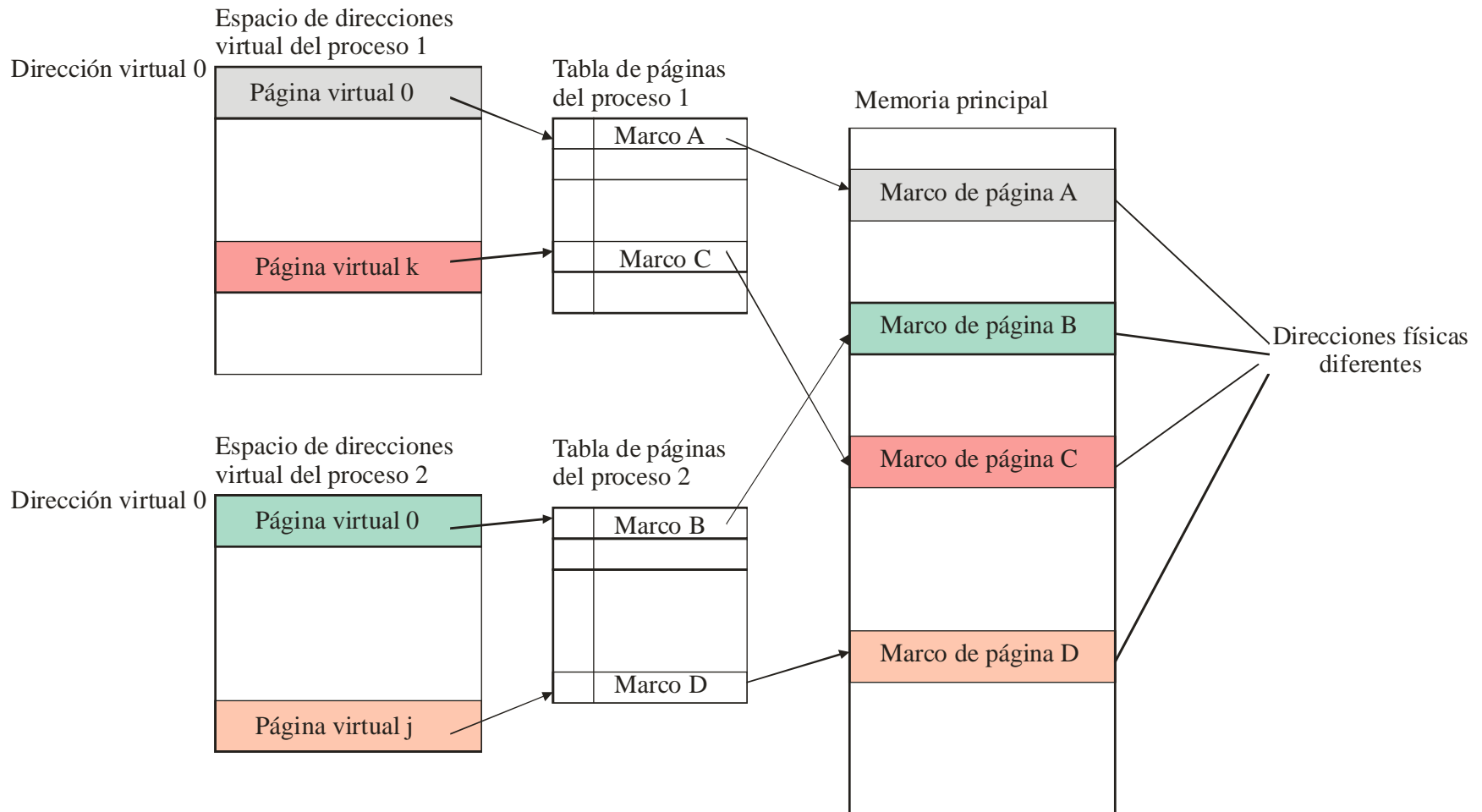


Tabla de páginas de dos niveles

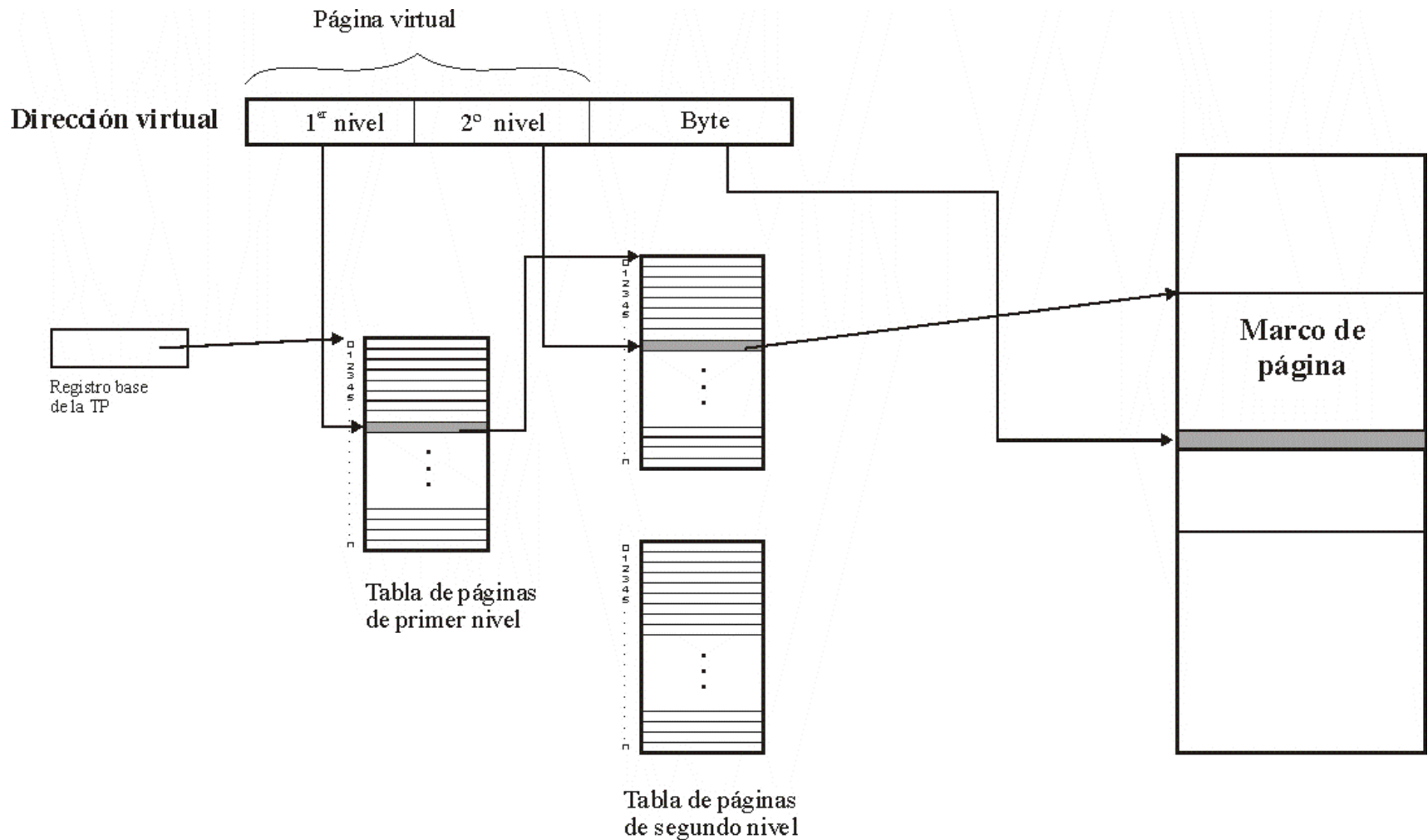
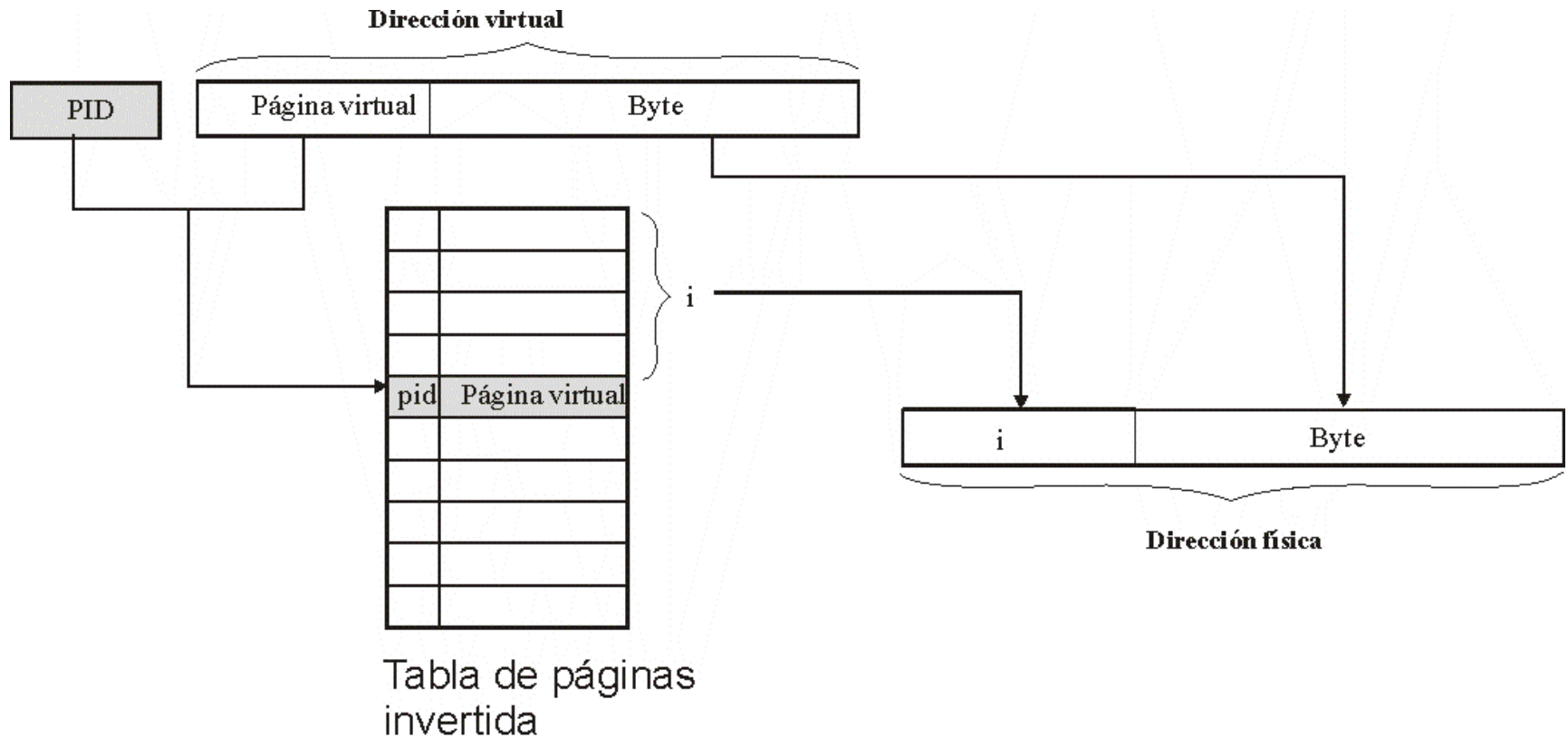


Tabla de páginas invertida



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:

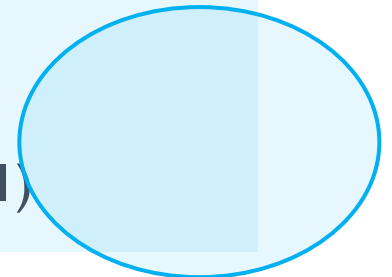
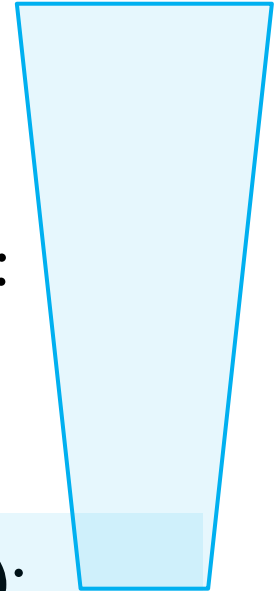
- Página no residente se marca ausente
- Se guarda dirección del bloque de *swap* que la contiene

► 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

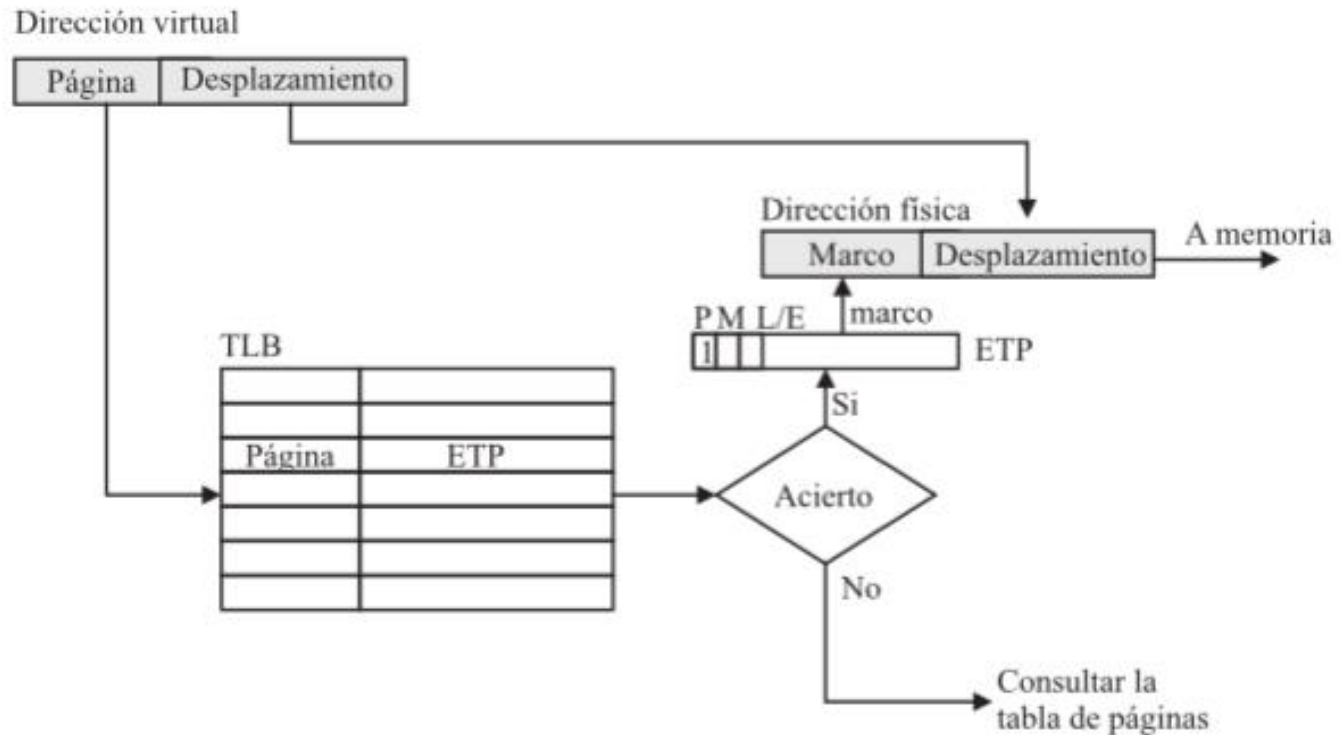
TLB (*Translation Lookaside Buffer*)

- ▶ 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.

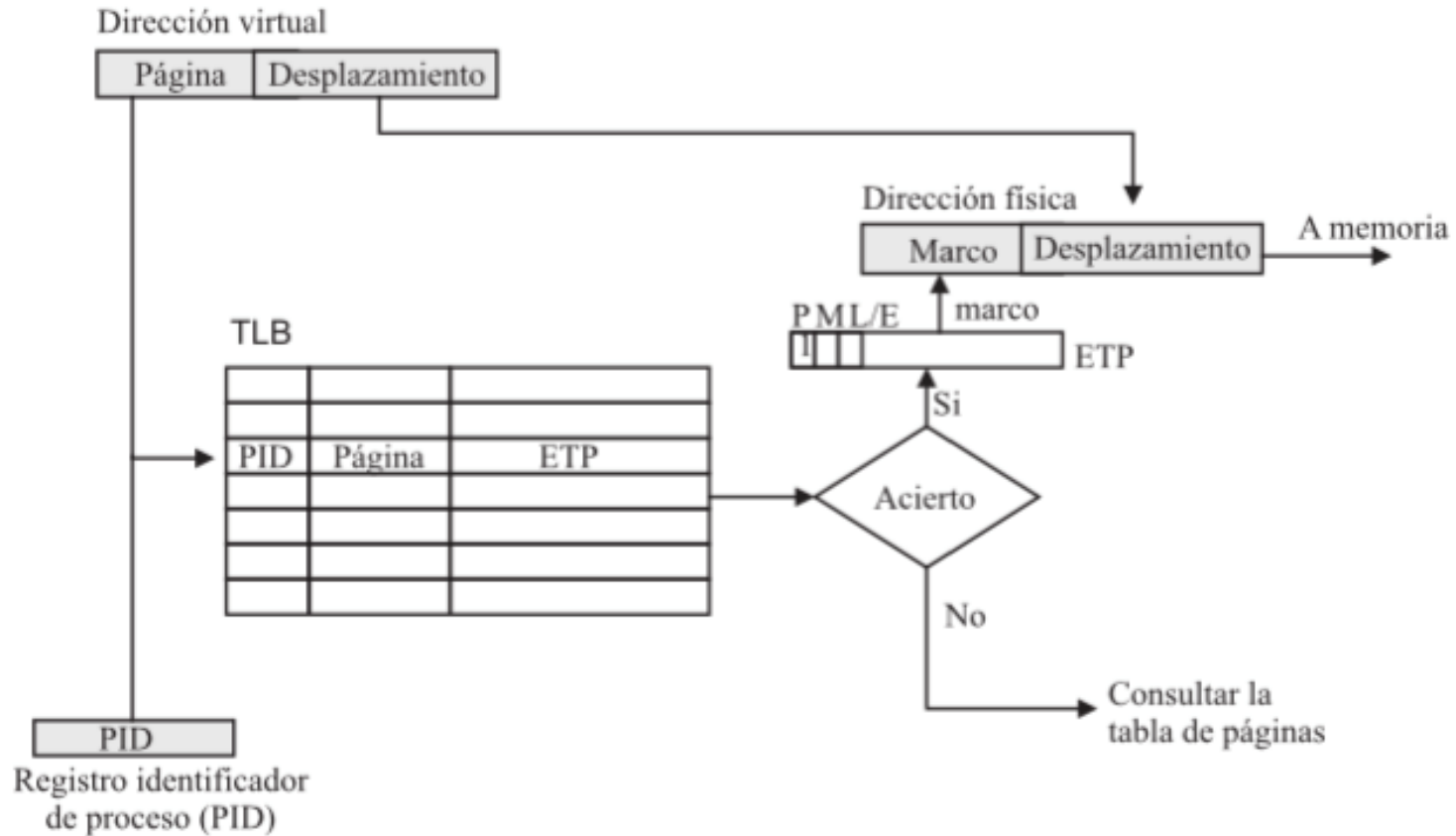
TLB (Translation Lookaside Buffer)

- ▶ **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

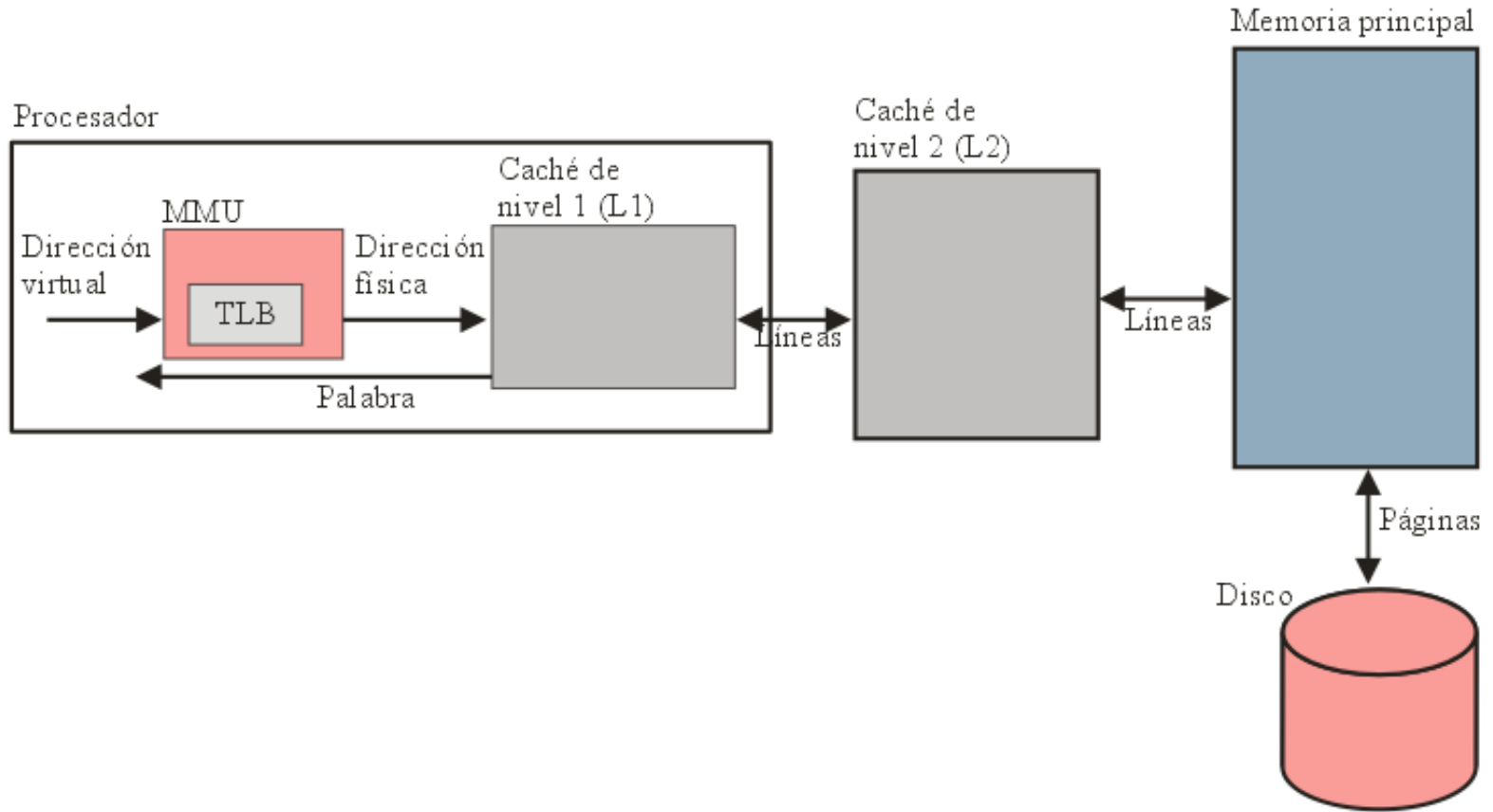
Caché

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

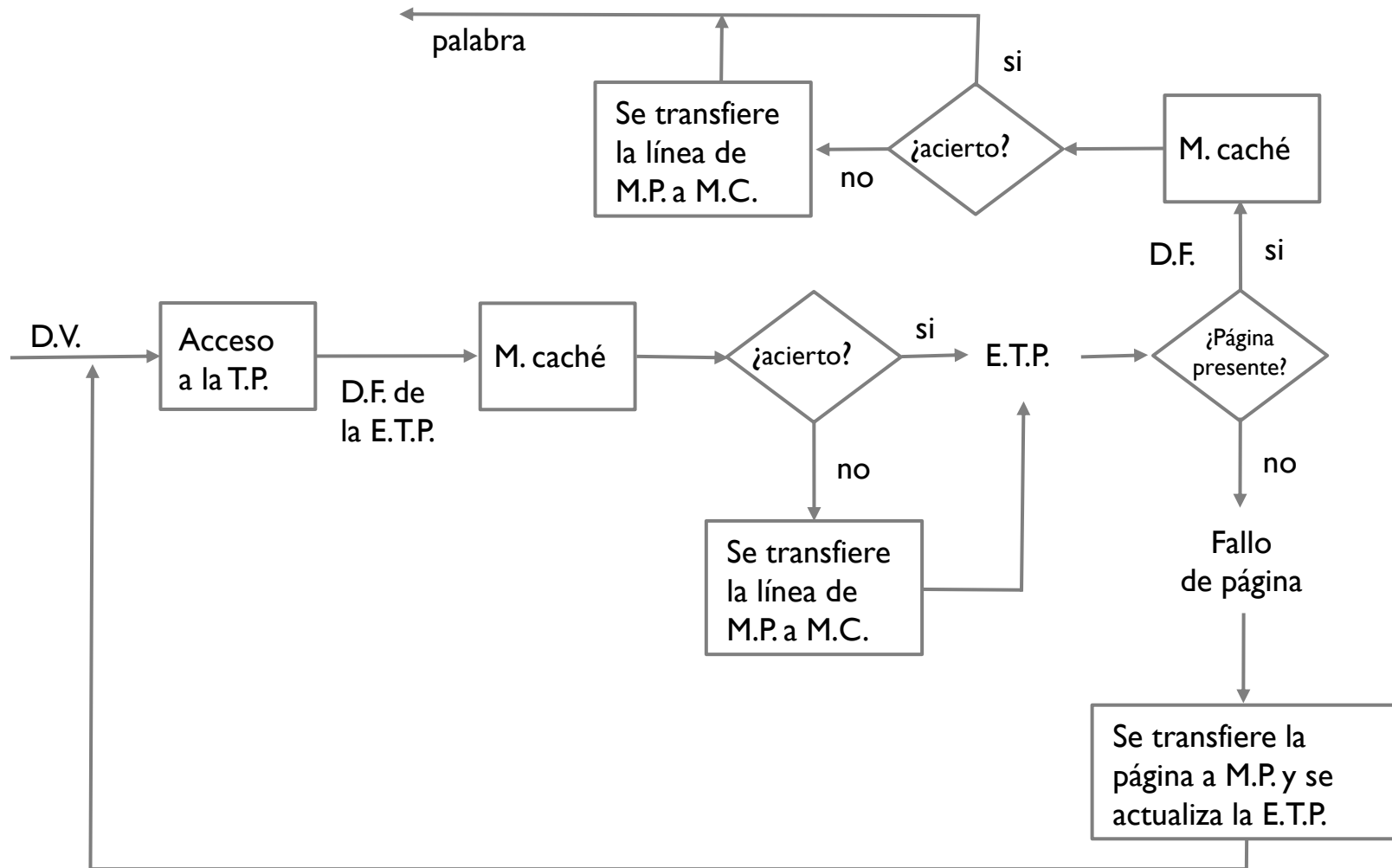
Memoria virtual

- ▶ 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