Tema 4. Organización y Gestión de Memoria

① Conceptos generales
② Esquemas de memoria basados en la asignación contigua
③ Recubrimientos (*overlay*)
④ Intercambio (*swapping*)
⑤ Organización de la Memoria Virtual
⑥ Gestión de la Memoria Virtual

Objetivos

- Distinguir entre dirección relativa o lógica y dirección física o real y entre espacio de direcciones lógico y físico
- Entender qué es la reubicación y sus diferentes tipos (estática y dinámica)
- Conocer las distintas formas en las que el sistema operativo puede organizar y gestionar la memoria física
- Saber en qué consiste y para qué se utiliza el mecanismo de recubrimientos (*overlays*) y el intercambio (*swapping*)
- Entender qué es la Memoria Virtual y por qué se utiliza
- Conocer los mecanismos de paginación, segmentación y segmentación paginada y cómo se llevan a cabo en un sistema con memoria virtual
- Comprender qué es la propiedad de localidad y su relación con el comportamiento de un programa en ejecución
- Conocer la teoría del conjunto de trabajo y el problema de la hiperpaginación

1. Conceptos Generales

① Jerarquía de Memoria

② Objetivos generales

③ Espacio de direcciones lógico y espacio de direcciones físico

④ Carga absoluta y reubicación

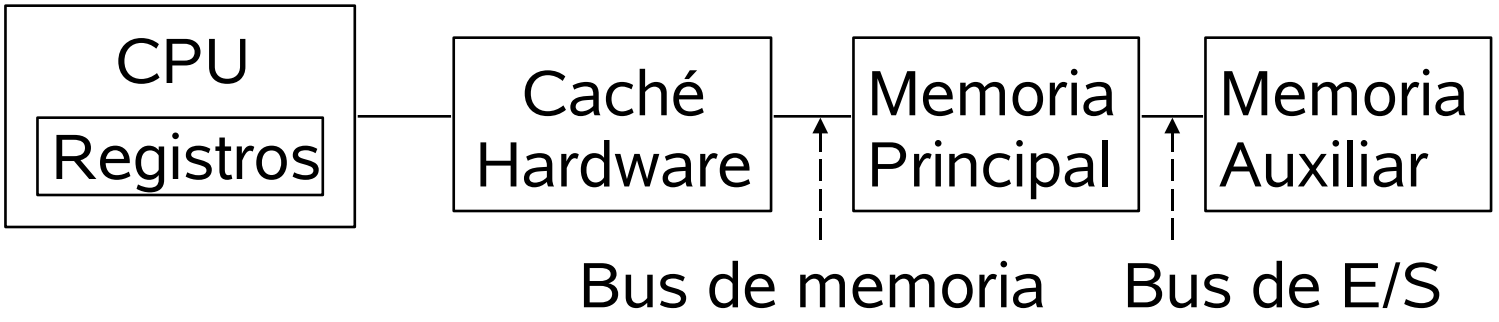
1.1 Jerarquía de Memoria

Dos principios sobre memoria:

① Menor cantidad, acceso más rápido

② Mayor cantidad, menor coste por byte

Así, los elementos frecuentemente accedidos se ponen en memoria rápida, cara y pequeña; el resto, en memoria lenta, grande y barata.



Conceptos sobre Cachés

- Caché - copia que puede ser accedida más rápidamente que el original
- Idea: hacer los casos frecuentes eficientes, los caminos infrecuentes no importan tanto
- *Acierto de caché*: item en la caché
- *Fallo de caché*: item no en caché; hay que realizar la operación completa
- **Tiempo de Acceso Efectivo** (TAE) =

  Probabilidad_acierto * coste_acierto +
  Probabilidad_fallo * coste_fallo
- Funciona porque los programas no son aleatorios, explotan la **localidad** → **principio de localidad**

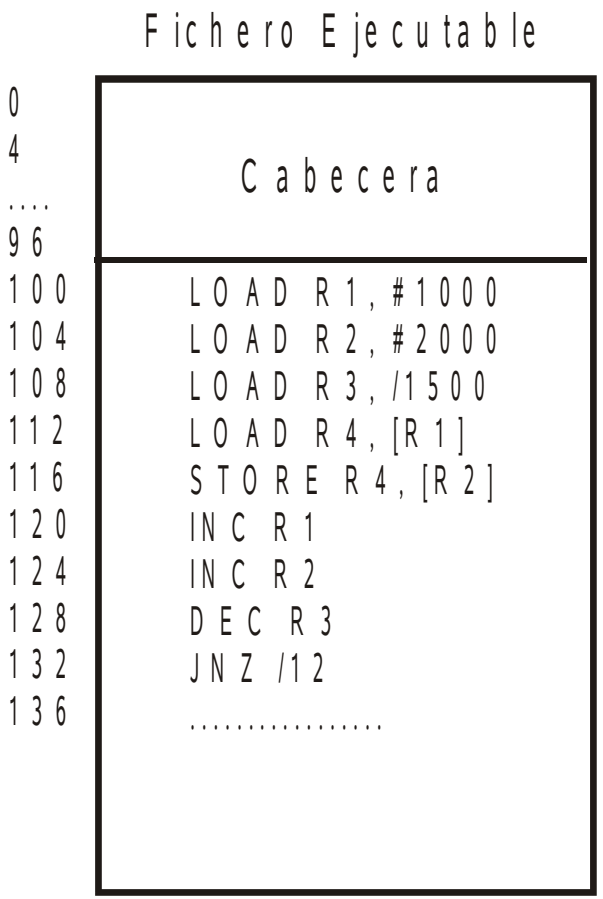
1.2 Objetivos generales

- **Organización**: ¿cómo está dividida la memoria?
- **Gestión**: Dado un esquema de organización, ¿qué estrategias se deben seguir para obtener un rendimiento óptimo?
  - » Estrategias de **asignación**
  - » Estrategias de **sustitución**
  - » Estrategias de **búsqueda**
- **Protección**
  - » El SO de los procesos de usuario
  - » Los procesos de usuario entre ellos

1.3 Espacio de direcciones lógico y espacio de direcciones físico

Espacio de direcciones lógico: conjunto de direcciones lógicas o virtuales generadas por un programa

Espacio de direcciones físico: conjunto de direcciones físicas correspondientes a las direcciones lógicas en un instante dado



Carretero, p.165

Mapa de memoria de un proceso

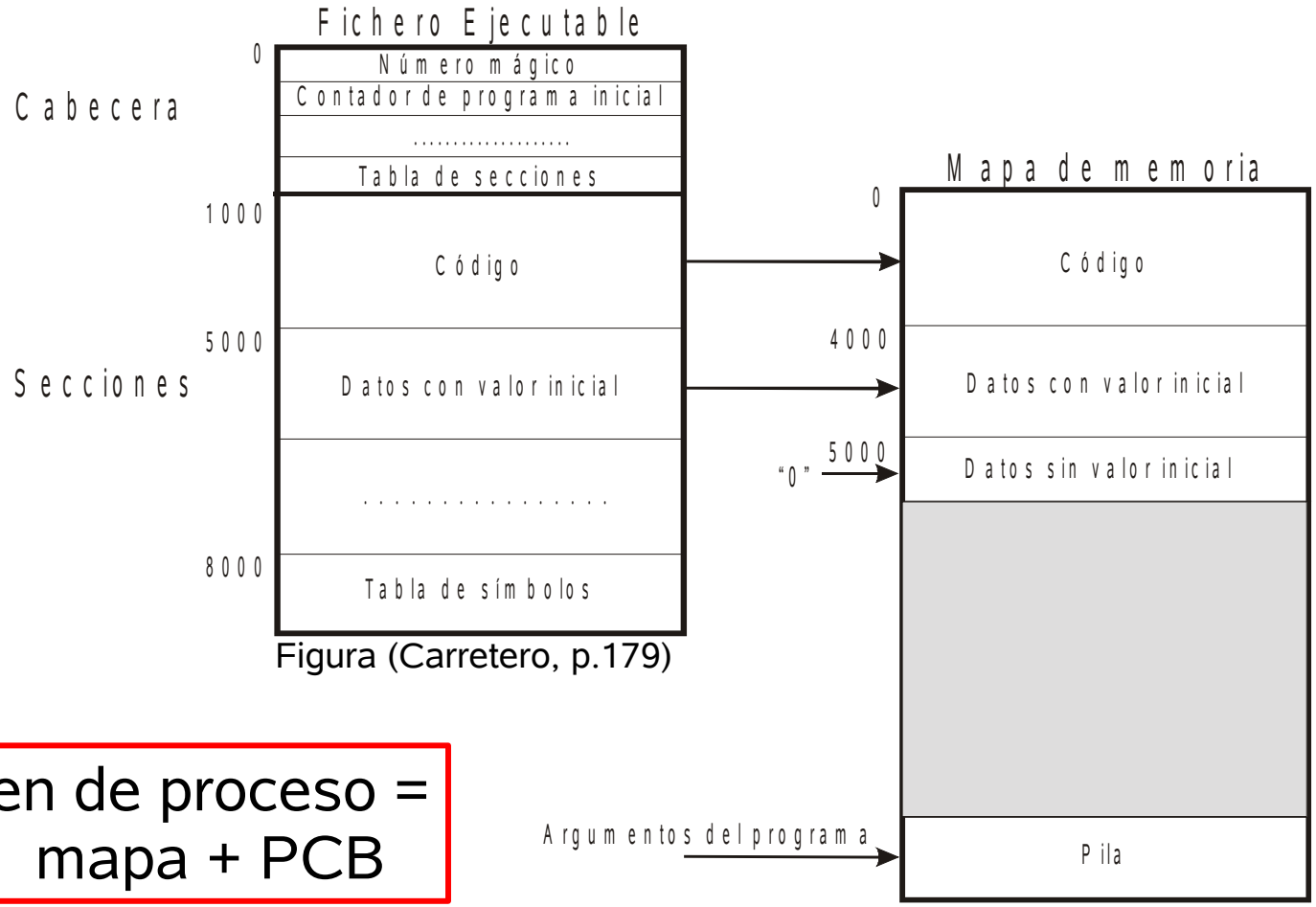


Figura (Carretero, p.179)

Imagen de proceso = mapa + PCB

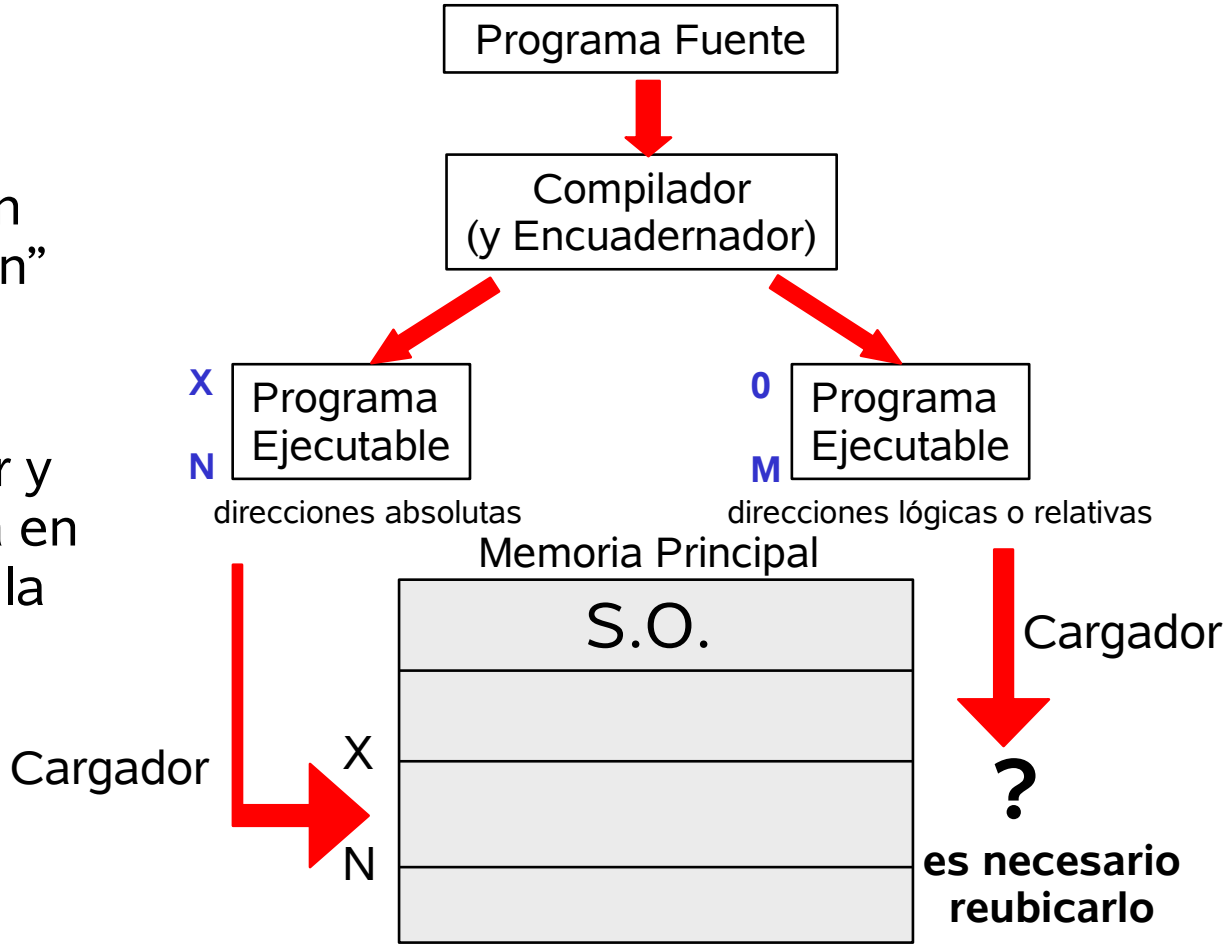
1.4 Carga absoluta y Reubicación

Carga absoluta

"Asignar direcciones físicas al programa en tiempo de compilación"

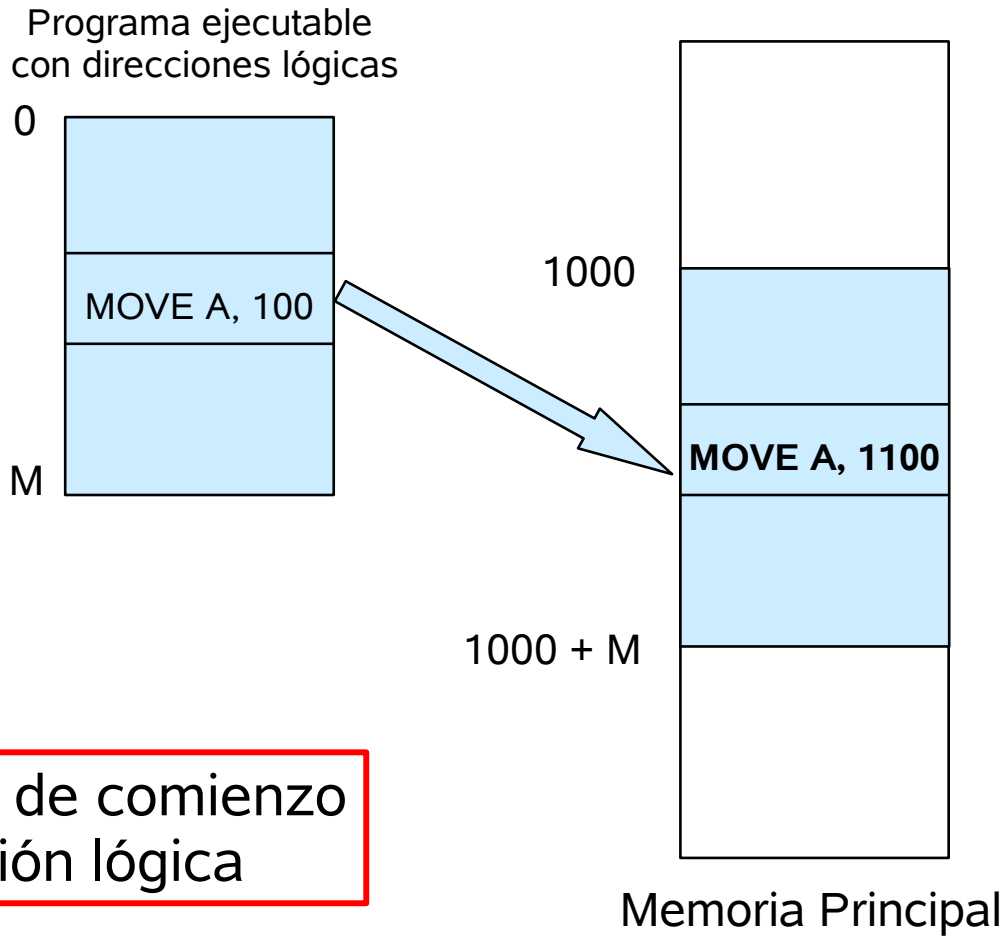
Reubicación

"Capacidad de cargar y ejecutar un programa en un lugar arbitrario de la memoria"



Reubicación estática

- Decisión de dónde ubicar el programa en **tiempo de carga**
- El compilador genera **direcciones lógicas** (relativas)

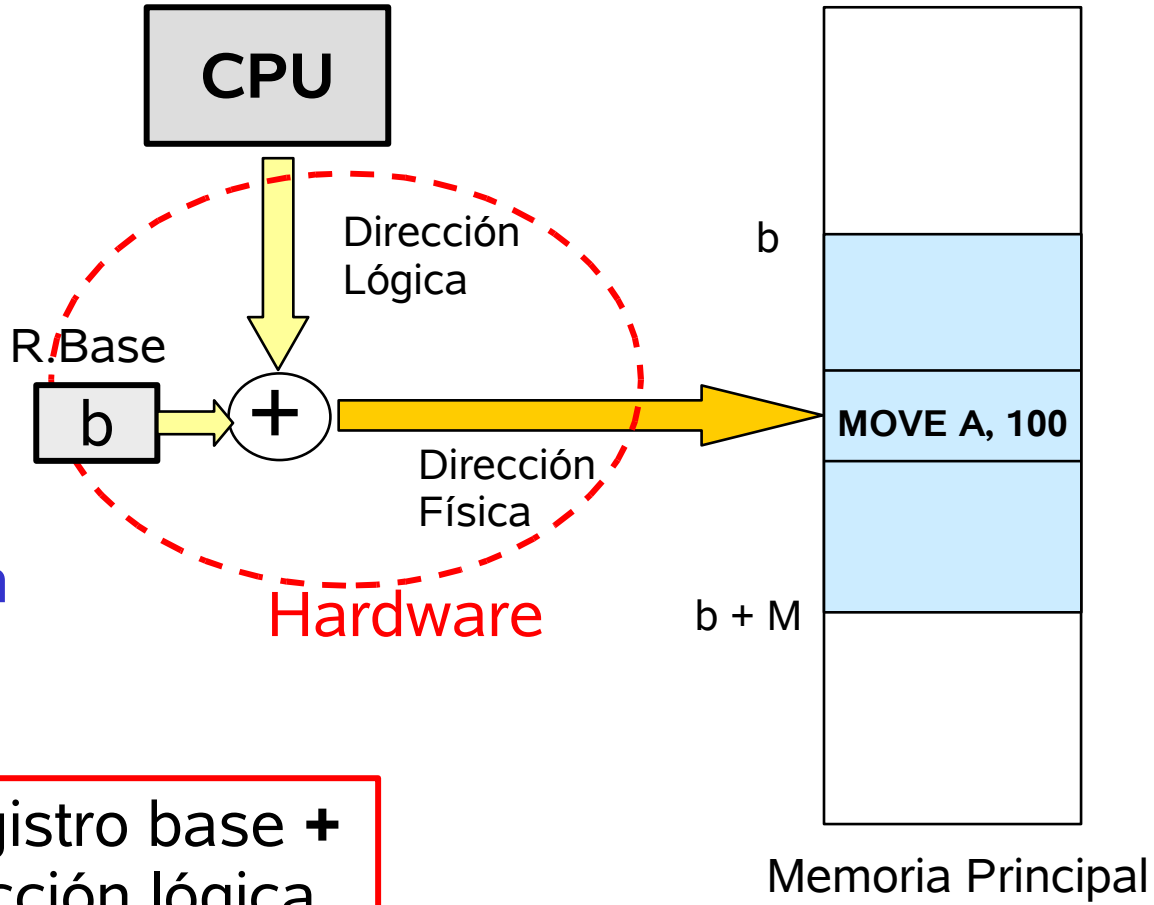


dirección física = dirección de comienzo
+ dirección lógica

Reubicación dinámica

- El compilador genera **direcciones lógicas** (relativas)

- La traducción de direcciones lógicas a físicas se hace en **tiempo de ejecución**



dirección física = registro base + dirección lógica

2. Esquemas de memoria basados en la asignación contigua

① Asignación contigua y no contigua

② Sistemas monoprogramados

③ Sistemas multiprogramados con particiones:

- Particiones fijas
- Particiones variables
- Protección en sistemas multiprogramados

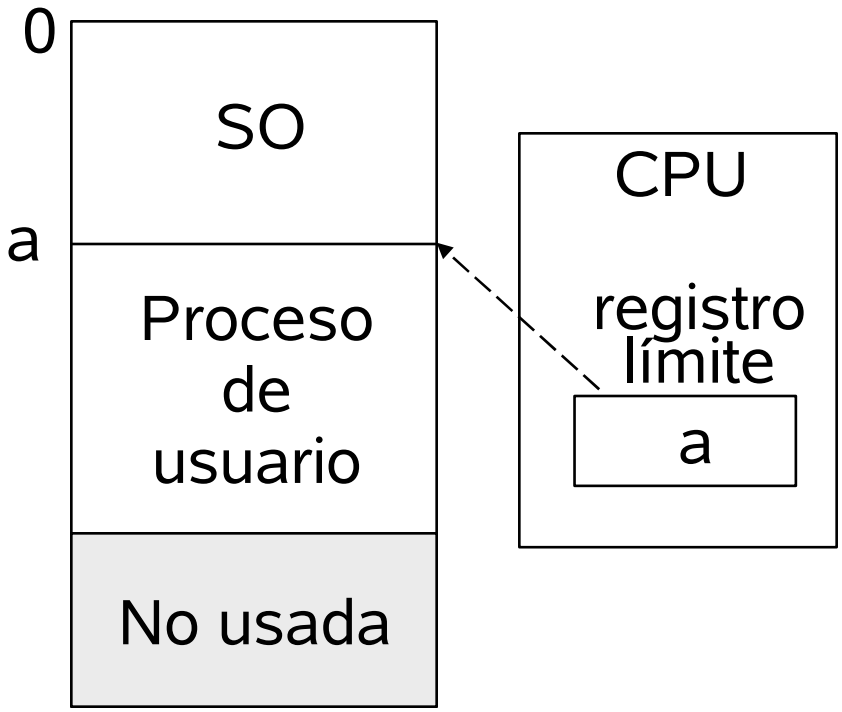
2.1 Asignación contigua y no contigua

Podemos clasificar las organizaciones de memoria como:

- *Contiguas* - La asignación de almacenamiento para un programa se hace en un único bloque de posiciones continuas de memoria → Particiones fijas y Particiones variales
- *No contiguas* - Permiten dividir el programa en bloques o segmentos que se pueden colocar en zonas no necesariamente continuas de memoria principal → Paginación, Segmentación y Segmentación paginada

2.2 Sistemas monoprogramados

- Memoria principal dividida en dos partes:
  - » S.O. residente
  - » Proceso de usuario
- La asignación contigua en sistemas monousuario implementa protección a través del registro de relocalización (o límite)



2.3 Sistemas multiprogramados con particiones

- Existen varios procesos en memoria.
- El tamaño máximo de cada proceso es el tamaño de la memoria física disponible.
- El esquema más sencillo es dividir la memoria en regiones o particiones cada una de las cuales puede ser ocupada por un proceso. Existen dos formas de dividir la memoria en particiones
  - » **Particiones fijas o estáticas**
  - » **Particiones variables o dinámicas**
- La protección se obtiene a través de los *registros base* y *límite* (sus valores para un proceso se guardan en su PCB)

Particiones fijas

- El número y tamaño de cada partición viene establecido por el sistema y es constante
- Tamaño de las particiones:
  - » Particiones de igual tamaño
  - » Particiones de distinto tamaño
- Estrategias de asignación: proceso – partición
  - » Cola única
  - » Varias colas (particiones de distinto tamaño)

Problema de las particiones fijas

- Uso de la memoria principal ineficiente. Cualquier proceso, sin importar lo pequeño que sea, ocupará una partición completa
- **Fragmentación**: incapacidad del sistema operativo para asignar posiciones de memoria principal no utilizadas. Dos tipos: interna y externa
- Problema de fragmentación en particiones fijas: **Fragmentación interna**

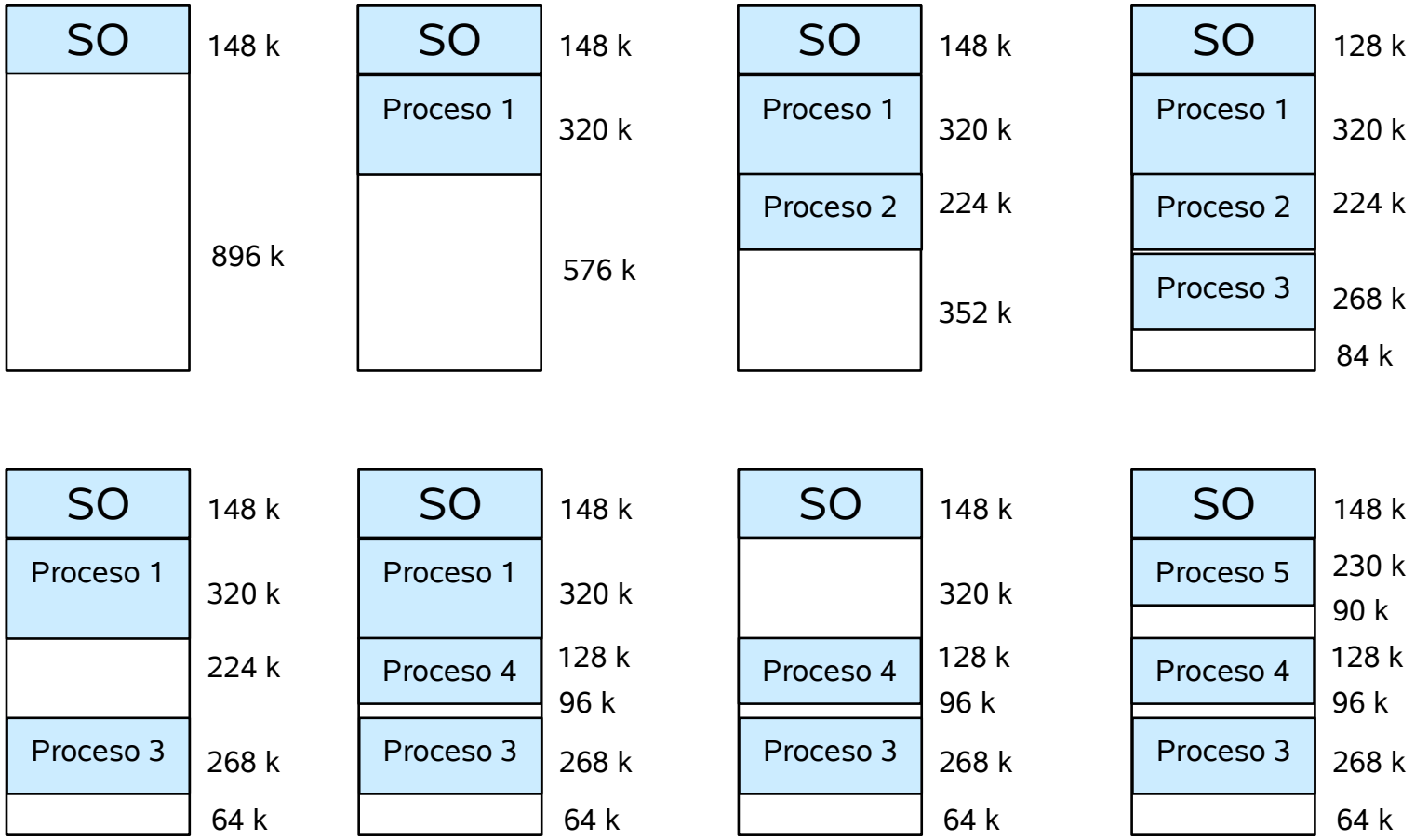
Particiones variables

- Las particiones son variables en número y tamaño
- Cuando llega un proceso a memoria se le asigna la memoria que necesita: se crea una partición
- El SO mantiene información sobre las zonas de memoria asignadas (tabla de particiones) y las libres (huecos)
- Es necesario tener una estrategia de asignación de espacio y gestión de espacio libre

Problema particiones variables

- **Fragmentación externa** - existe el espacio necesario para satisfacer una petición pero no es contiguo
- **Compactación** - técnica utilizada para reducir la fragmentación externa. Consiste en arrastrar los contenidos de memoria a un lugar para reunir toda la memoria libre en un bloque. Problemas:
  - » Requiere reubicación dinámica
  - » Consume recursos del sistema
  - » El sistema se detiene mientras se realiza → no se puede hacer demasiado frecuentemente

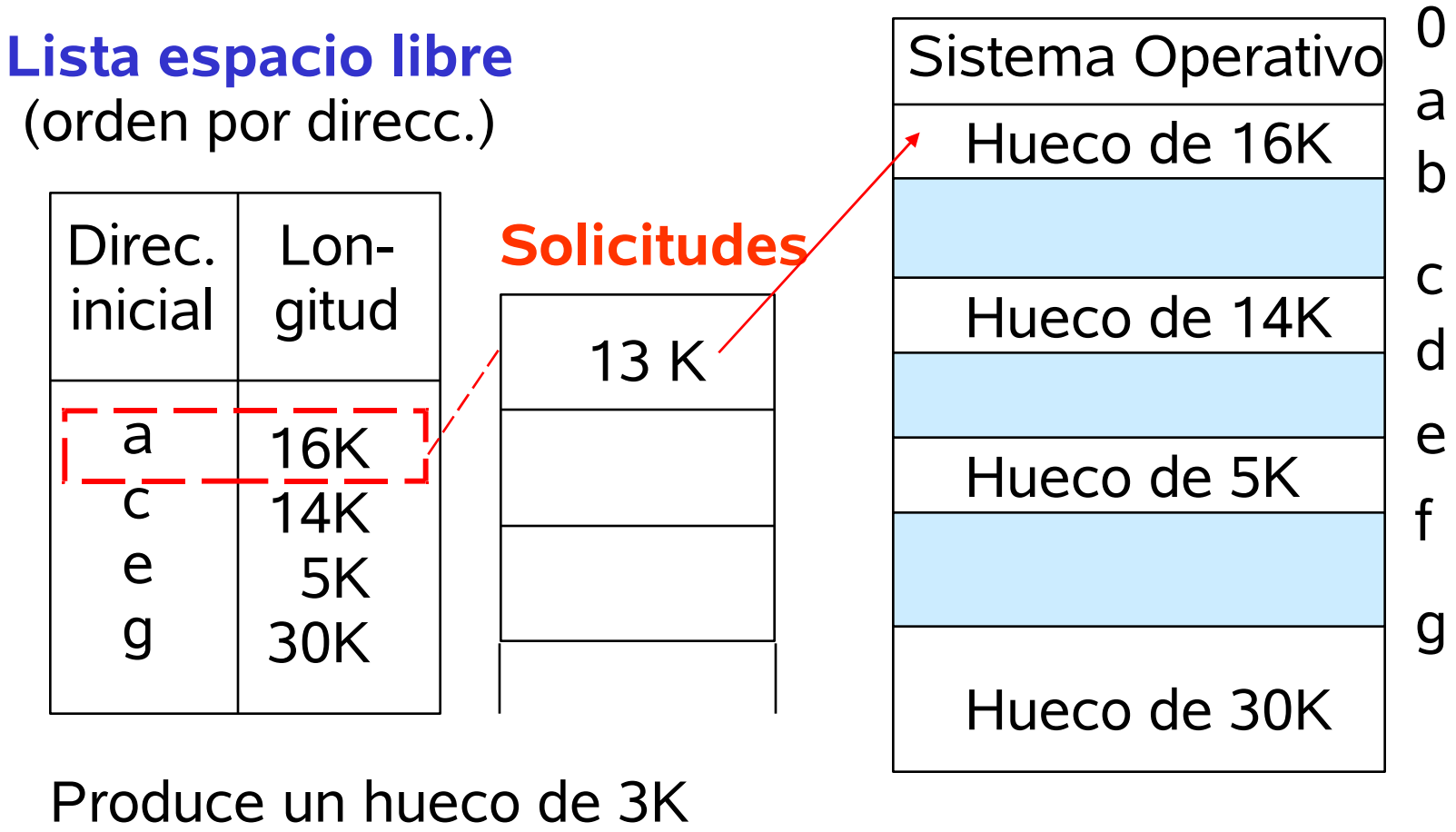
Ejemplo fragmentación externa



Particiones variables: Estrategias de asignación de espacio

- ¿Cómo satisfacer una petición de tamaño *n* desde una lista de huecos? Estrategias:
  - » **Primer ajuste** – asigna el primer hueco lo suficientemente grande para satisfacer la petición
  - » **Mejor ajuste** – asigna el hueco más pequeño que mejor se ajuste al espacio necesitado. Se debe buscar en la lista entera, si no esta ordenada por tamaños. Produce el hueco sobrante menor.
  - » **Peor ajuste** - asigna el hueco mayor. Debemos buscar en toda la lista. Produce el hueco sobrante mayor

Ejemplo: Primer Ajuste



Produce un hueco de 3K

Ejemplo: El Mejor Ajuste

Lista espacio libre
(orden ascendente
tamaño de hueco)

Direc. inicial	Lon-gitud
e	5k
c	14k
a	16k
g	30k

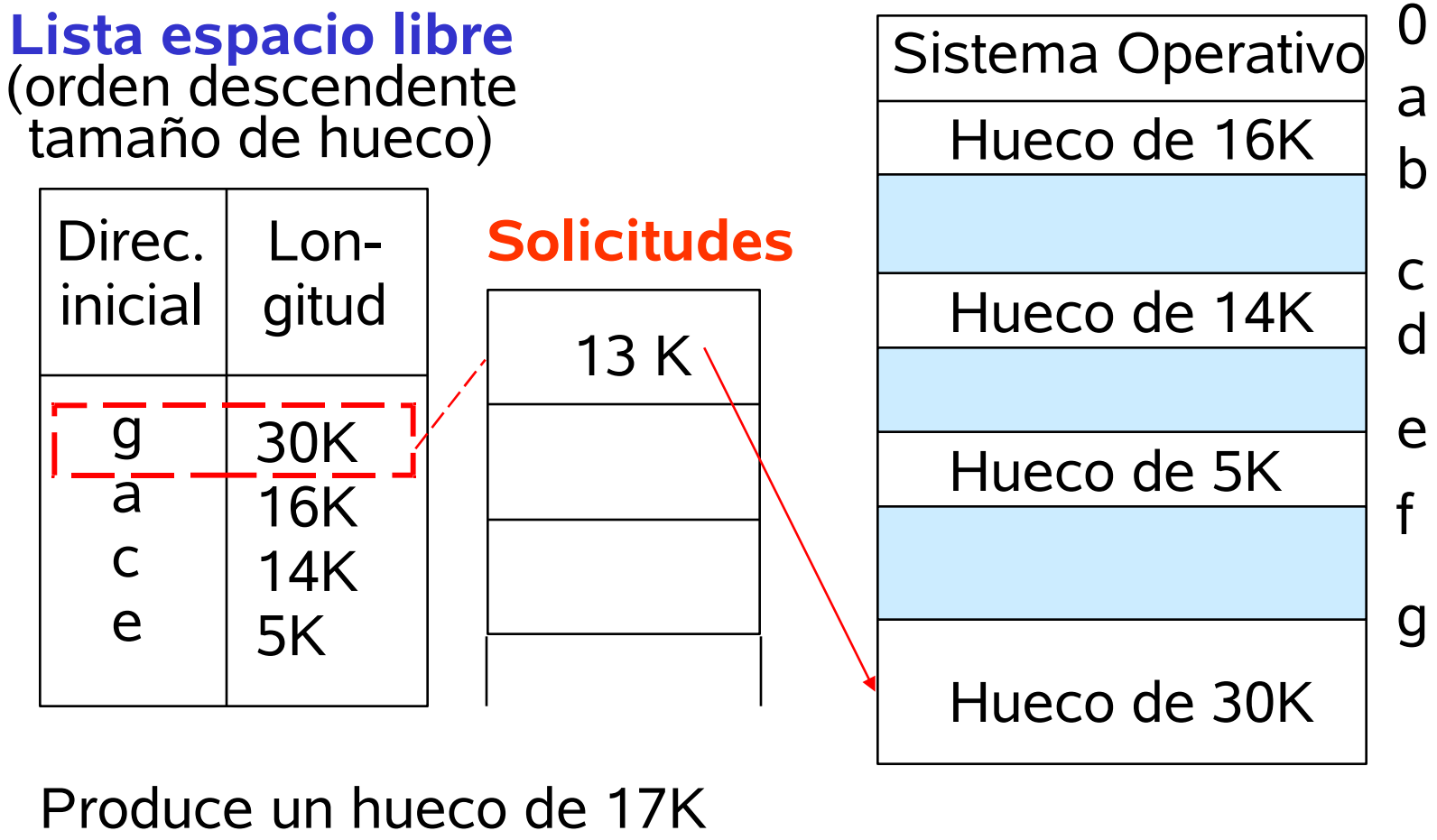
Solicitudes

13 K

Sistema Operativo	0
Hueco de 16K	a
	b
Hueco de 14K	c
	d
Hueco de 5K	e
	f
	g
Hueco de 30K	

Produce un hueco de 1K

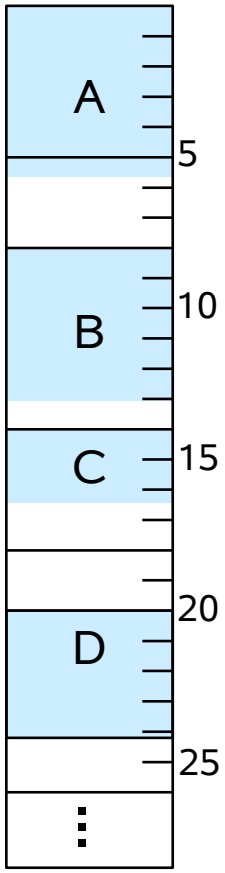
Ejemplo: El Peor Ajuste



Produce un hueco de 17K

Gestión de espacio libre

Estado actual de la M.P.

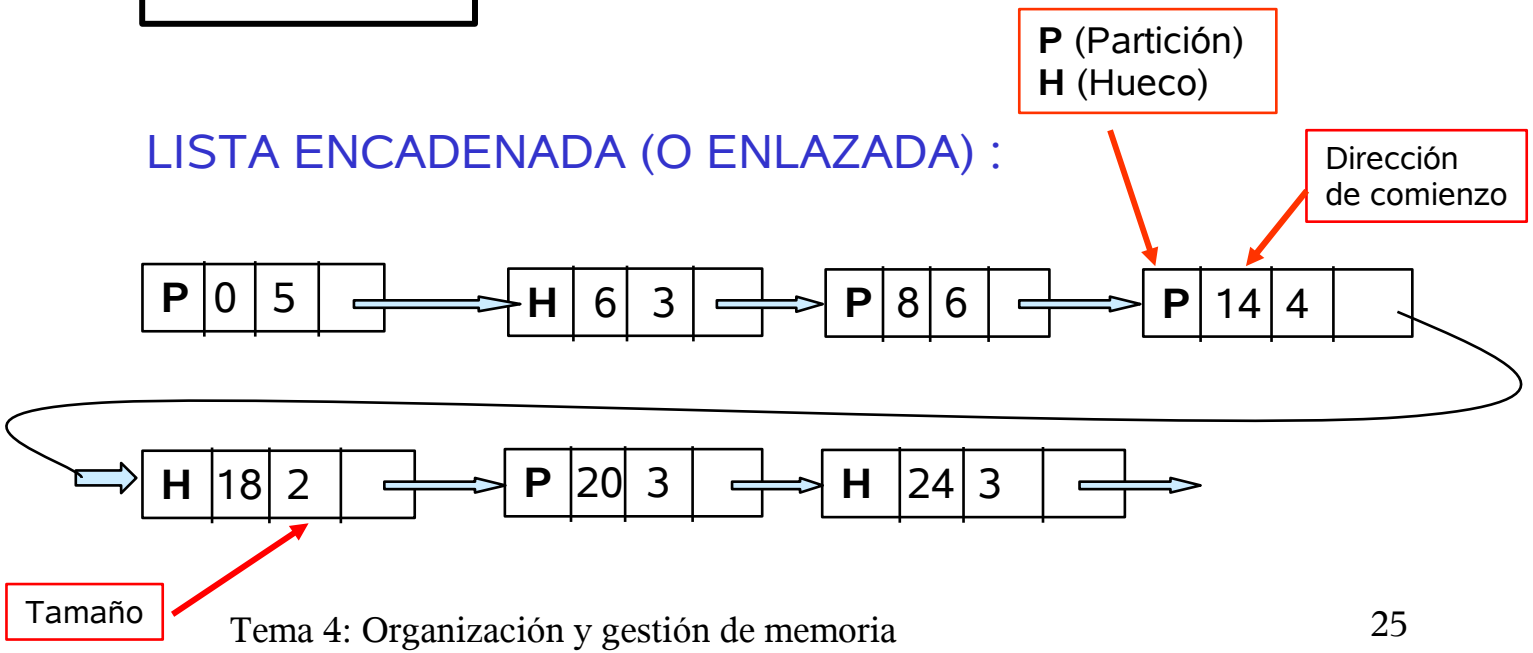


MAPA DE BITS :

1 1 1 1 1 0 0 0 1
1 1 1 1 1 1 1 1 1
0 0 1 1 1 0 0 ....

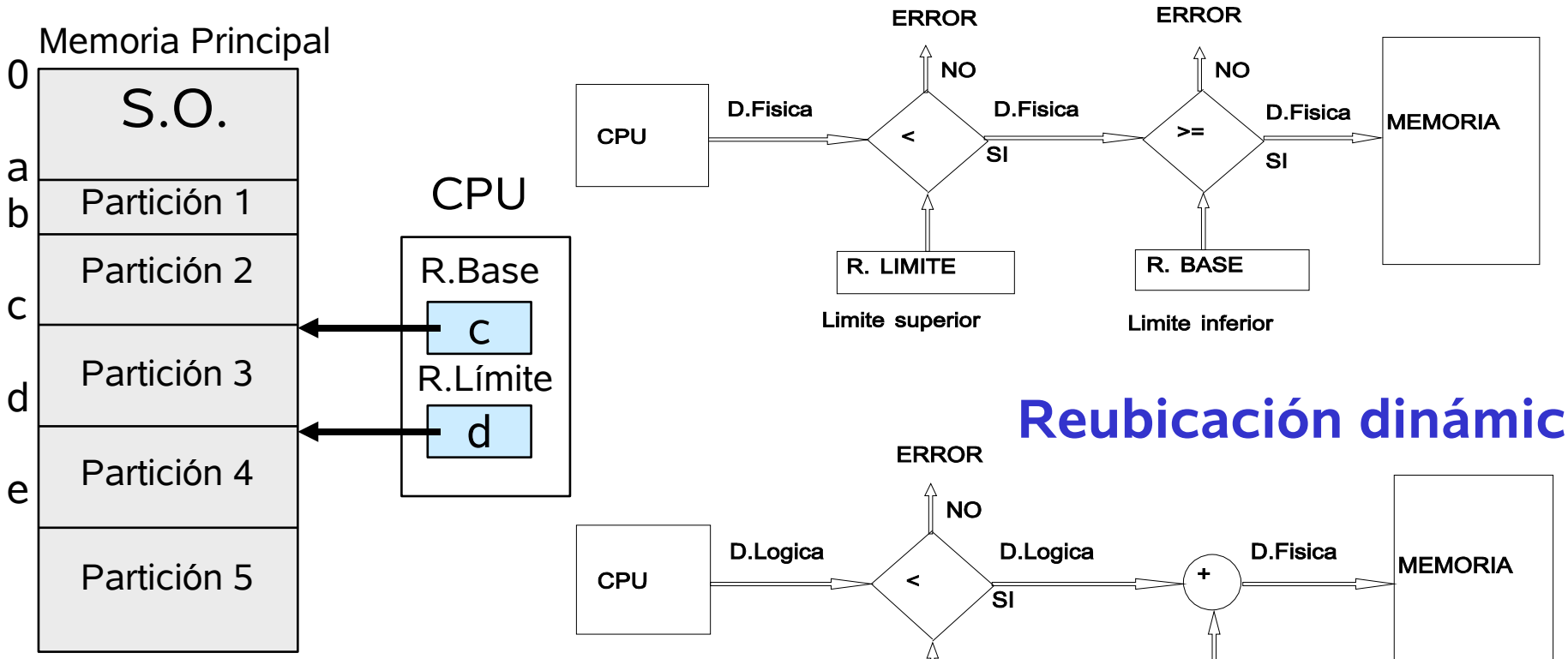
Divide la memoria principal en unidades de asignación y se utiliza un bit por cada una de ellas que indica si está libre (0) u ocupada (1)

LISTA ENCADENADA (O ENLAZADA) :



Protección en sistemas multiprogramados

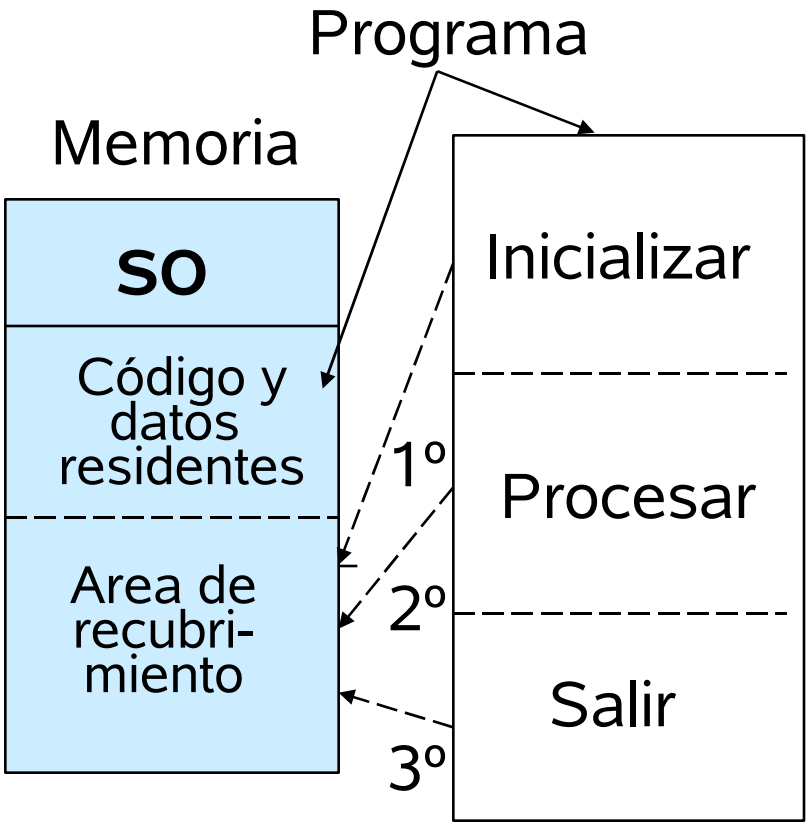
Reubicación estática



Reubicación dinámica

3. Recubrimientos (*overlays*)

- Era un mecanismo para ejecutar programas más grandes que la memoria. En memoria sólo se tenían las instrucciones y datos que se necesitan en un momento dado
- El SO no da soporte. Es responsabilidad del programador diseñar y programar la estructura de overlays



4. Intercambio (*Swapping*)

- Intercambiar procesos entre memoria y un almacenamiento auxiliar
- El almacenamiento auxiliar debe ser un disco rápido con espacio para albergar las imágenes de memoria de los procesos de usuario
- El factor principal en el tiempo de intercambio es el tiempo de transferencia
- El ***intercambiador*** tiene las siguientes responsabilidades:
  - » Seleccionar procesos para retirarlos de MP
  - » Seleccionar procesos para incorporarlos a MP
  - » Gestionar y asignar el espacio de intercambio

Localización del espacio de intercambio

- Intercambio Dinámico:
  - » Un archivo de intercambio global del sistema cubre las necesidades de intercambio de todos los procesos
  - » Problema: elección de su tamaño
- Intercambio Estático:
  - » Existe un archivo de intercambio dedicado por cada proceso intercambiable del sistema
  - » Elimina el problema del tamaño
  - » No impone límites al número de procesos intercambiados
  - » Problema: Necesita más espacio en disco, los accesos son más lentos y los direccionamientos son más complicados

5. Organización de la Memoria Virtual

① Concepto de memoria virtual

② Paginación

③ Segmentación

④ Segmentación paginada

5.1 Concepto de memoria virtual

- Memoria Virtual
  - » El tamaño del programa, los datos y la pila puede exceder la cantidad de memoria física disponible para él.
  - » Se usa un almacenamiento a dos niveles:
    - – *Memoria Principal* ➜ partes del proceso necesarias en un momento dado
    - – *Memoria Secundaria* ➜ espacio de direcciones completo del proceso

Concepto de memoria virtual (y II)

» Es necesario:

– saber qué se encuentra en memoria principal

– una política de movimiento entre MP y MS

- Además, la memoria virtual

» resuelve el problema del crecimiento dinámico de los procesos

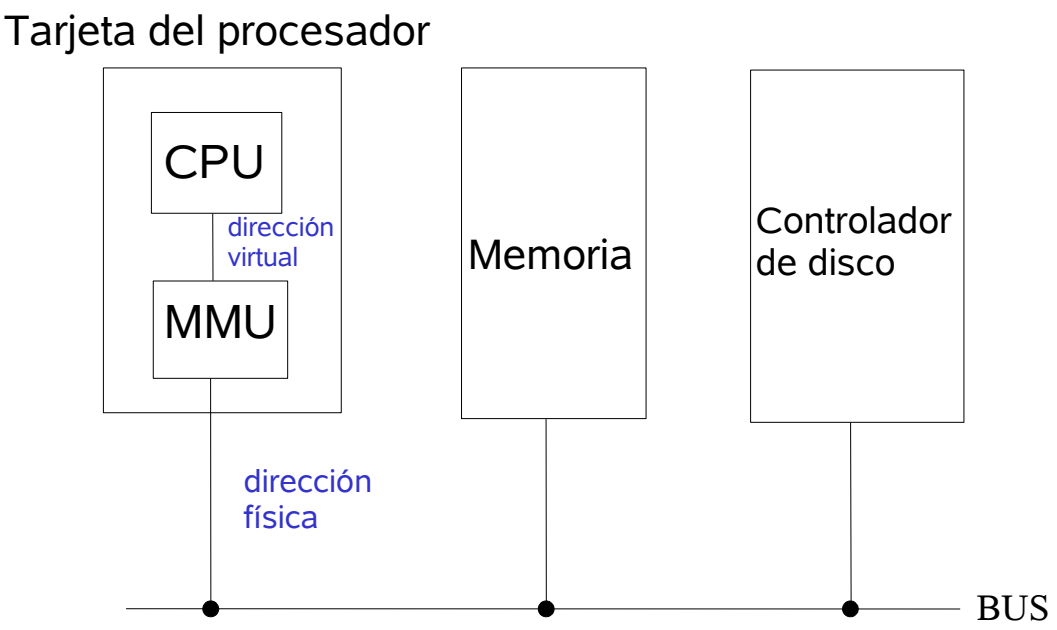
» puede aumentar el grado de multiprogramación

Unidad de Gestión de Memoria

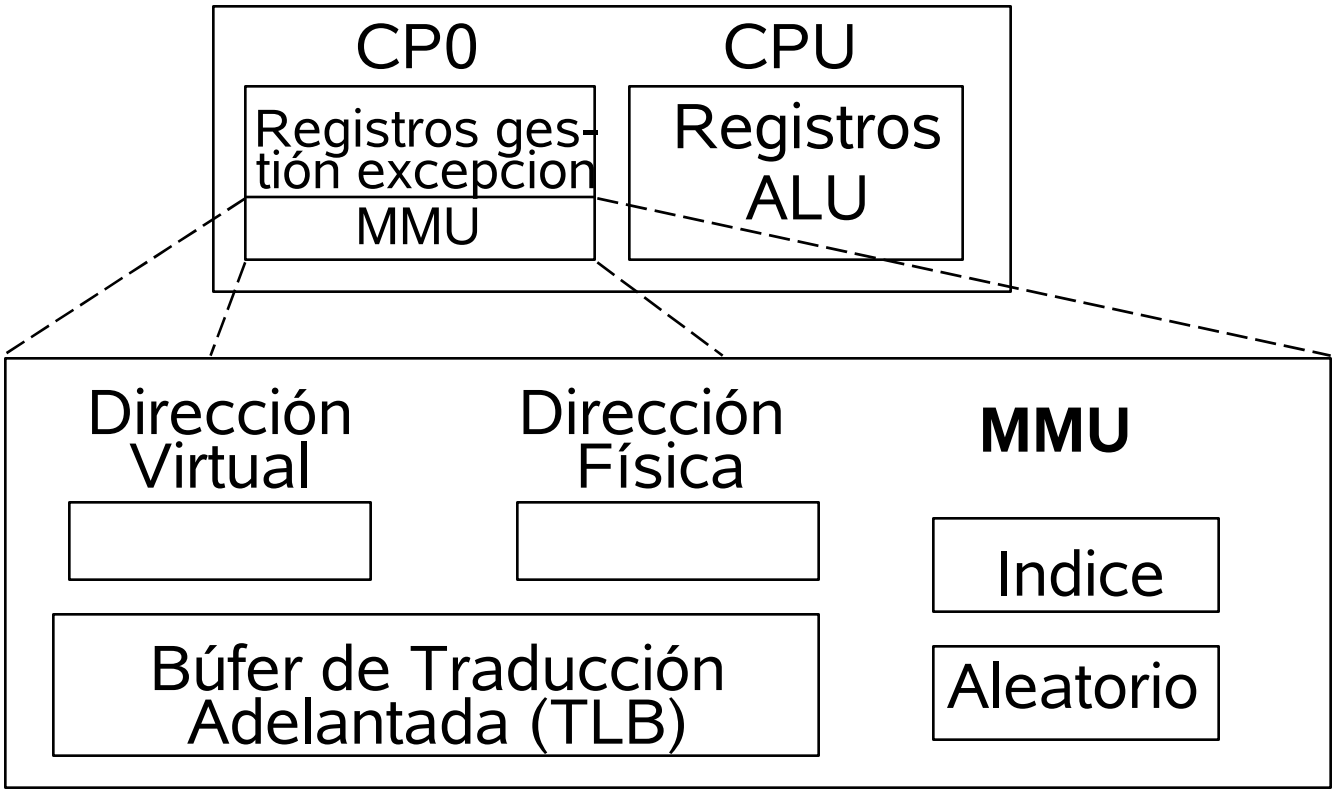
- La **MMU** (*Memory Management Unit*) es un dispositivo hardware que traduce direcciones virtuales a direcciones físicas ¡Este dispositivo está gestionado por el SO!
- En el esquema MMU más simple, el valor del registro base se añade a cada dirección generada por el proceso de usuario al mismo tiempo que es enviado a memoria
- El programa de usuario trata sólo con direcciones lógicas; éste nunca ve direcciones reales

Unidad de Gestión de Memoria (y II)

- Además de la traducción, el MMU deberá:
  - detectar si la dirección aludida se encuentra o no en MP
  - generar una interrupción si se encuentra en MS



Ejemplo de MMU: El MIPS R2000/3000



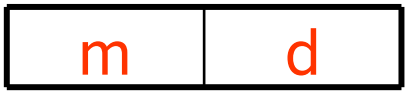
5.2 Paginación

- El espacio de direcciones físicas de un proceso puede ser no contiguo
- La memoria física se divide en bloques de tamaño fijo, denominados *marcos de página*. El tamaño es potencia de dos, de 0.5 a 8 Kb
- El espacio lógico de un proceso se divide en bloques del mismo tamaño, denominados *páginas*
- Los marcos de páginas contendrán páginas de los procesos

Paginación (y II)

- Las **direcciones lógicas**, que son las que genera la CPU se dividen en ***número de página*** (p) y ***desplazamiento*** dentro de la página (d)

- Las **direcciones físicas** se dividen en ***número de marco*** (m, *dirección base del* marco donde está almacenada la página) y ***desplazamiento*** (d)



Paginación (y III)

- Cuando la CPU genere una dirección lógica será necesario traducirla a la dirección física correspondiente, la *tabla de páginas* mantiene información necesaria para realizar dicha traducción. *Existe una tabla de páginas por proceso*
- *Tabla de ubicación en disco* (una por proceso) ubicación de cada página en el almacenamiento secundario
- *Tabla de marcos de página*, usada por el S.O. y contiene información sobre cada marco de página

Contenido de la tabla de páginas

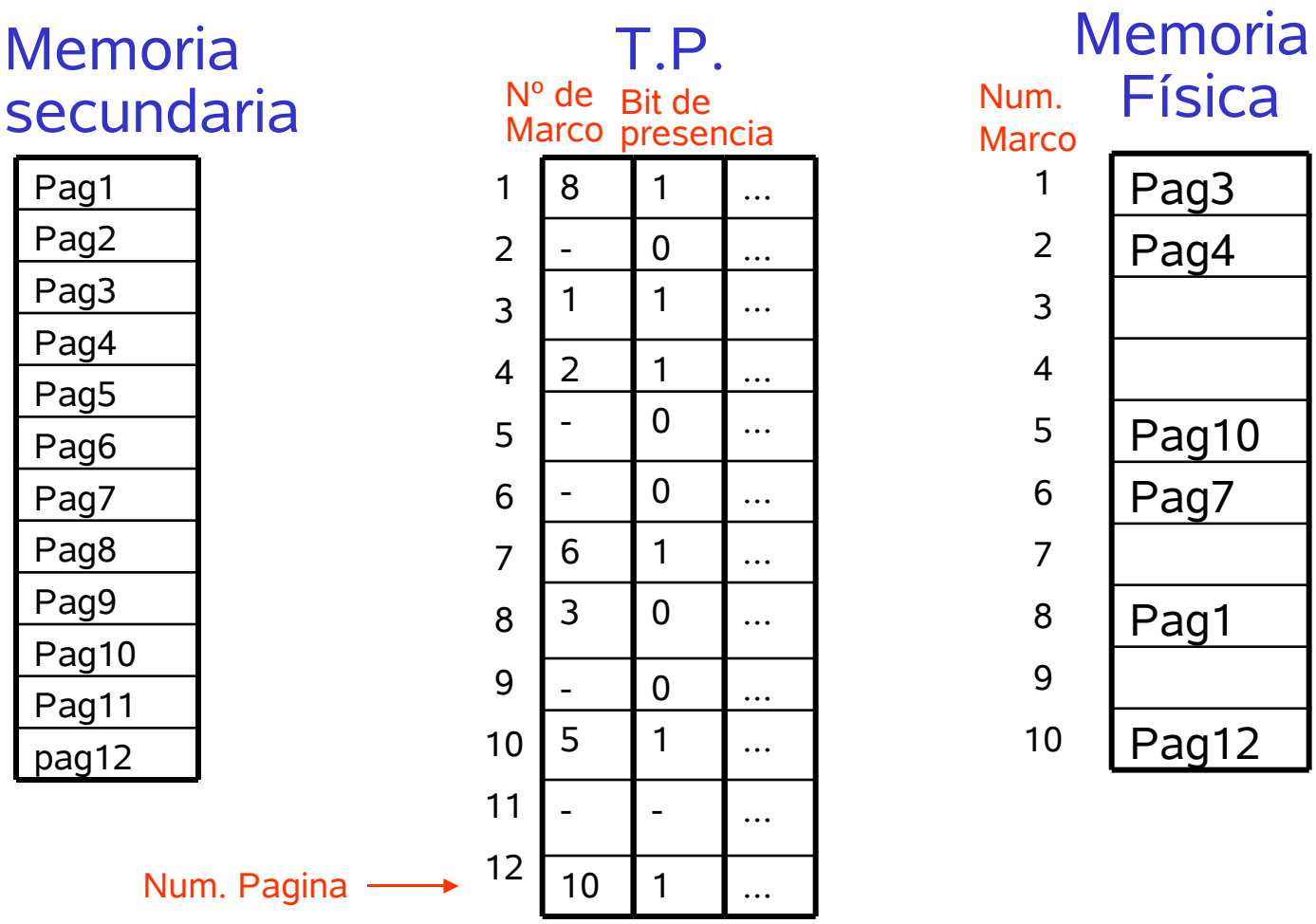
Una entrada por cada página del proceso:

- Número de marco (dirección base del marco) en el que está almacenada la página si está en MP
- Bit de presencia o bit válido
- Bit de modificación
- Modo de acceso autorizado a la página (bits de protección)

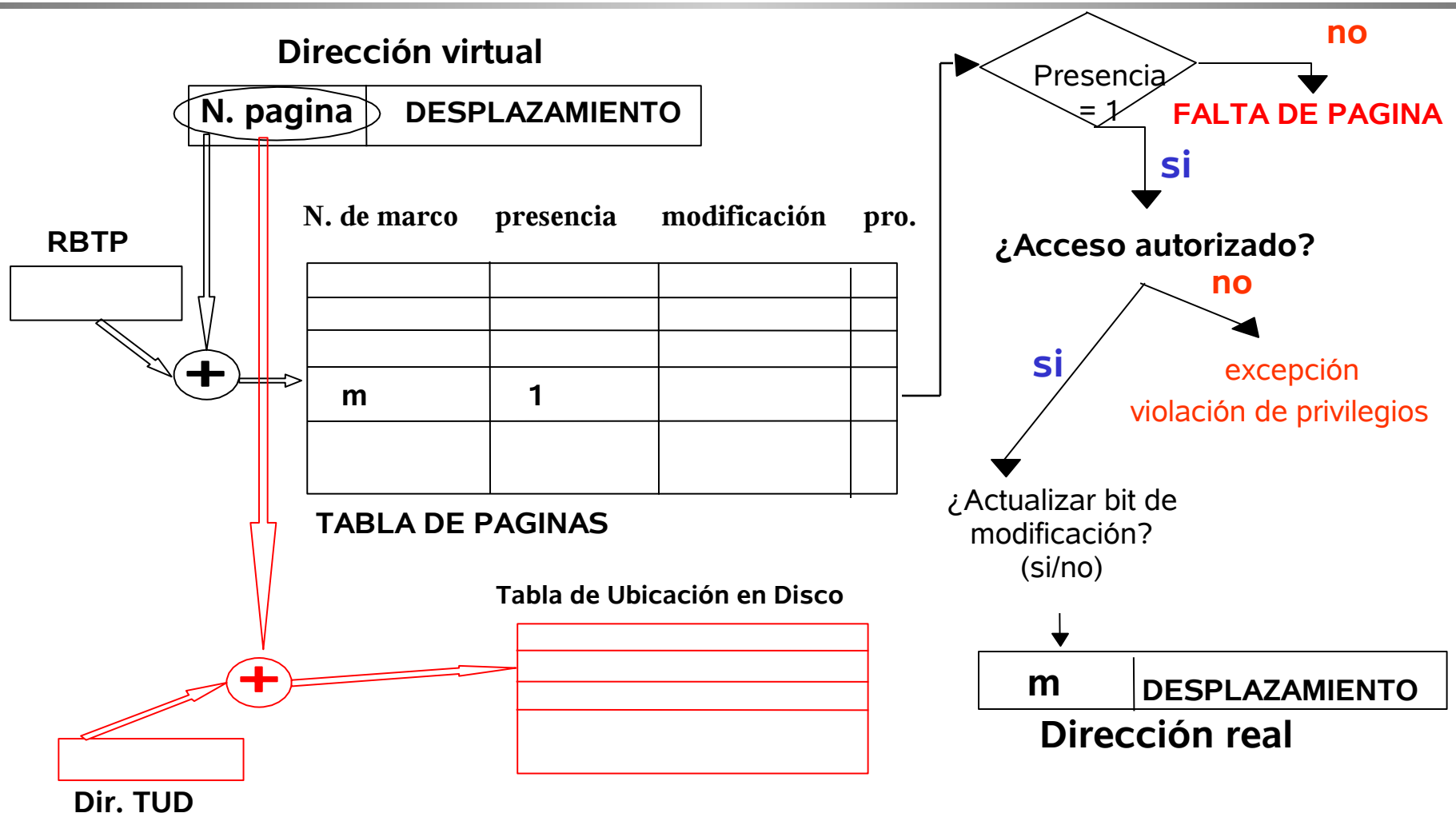
Nº de página

nº marco	presencia	modificación	protección
46	1	0	01

Ejemplo: contenido de la tabla de páginas



Esquema de traducción



Falta de página

1. Bloquear proceso
2. Encontrar la ubicación en disco de la página solicitada (*tabla de ubicación en disco*)
3. Encontrar un marco libre. Si no hubiera, se puede optar por desplazar una página de MP
4. Cargar la página desde disco al marco de MP
5. Actualizar tablas (bit presencia=1, nº marco, ...)
6. Desbloquear proceso
7. Reiniciar la instrucción que originó la falta de página

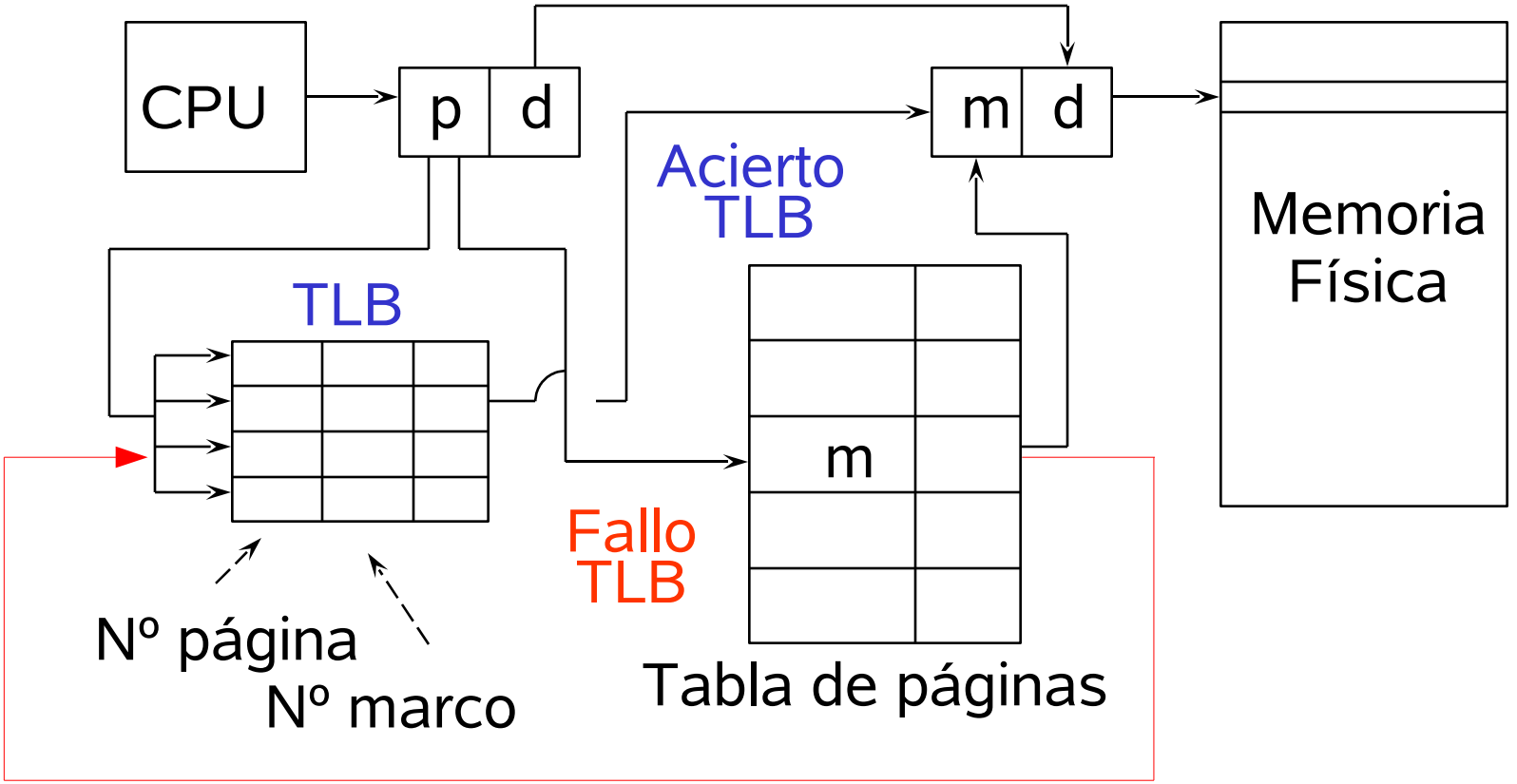
Implementación de la Tabla de Páginas

- La tabla de páginas se mantiene en memoria principal
- El *registro base de la tabla de páginas* (*RBTP*) apunta a la tabla de páginas (suele almacenarse en el PCB del proceso)
- En este este esquema:
  - » cada acceso a una instrucción o dato requiere dos accesos a memoria: uno a la tabla de páginas y otro a memoria
  - » un problema adicional viene determinado por el tamaño de la tabla de páginas

Búfer de Traducción Adelantada (TLB)

- El problema de los dos accesos a memoria se resuelve con una caché hardware de consulta rápida denominada *búfer de traducción adelantada* o *TLB* (*Translation Look-aside Buffer*)
- El TLB se implementa como un conjunto de *registros asociativos* que permiten una búsqueda en paralelo
- De esta forma, para traducir una dirección:
  1. Si existe ya en el registro asociativo, obtenemos el marco
  2. Si no, la buscamos en la tabla de páginas y se actualiza el TLB con esta nueva entrada

Traducción con TLB



Tamaño de la Tabla de Páginas

- Ejemplo:
  - » Dirección virtual: 32 bits.
  - » Tamaño de página = 4 Kbytes (2^{12} bytes).

➪ tamaño del campo desplazamiento = **12 bits**

➪ tamaño número de página virtual = **20 bits**

➪ Nº de páginas virtuales = 2^{20} = **¡1,048,576!**

- Solución para reducir el tamaño de la TP:
  - » Paginación multinivel

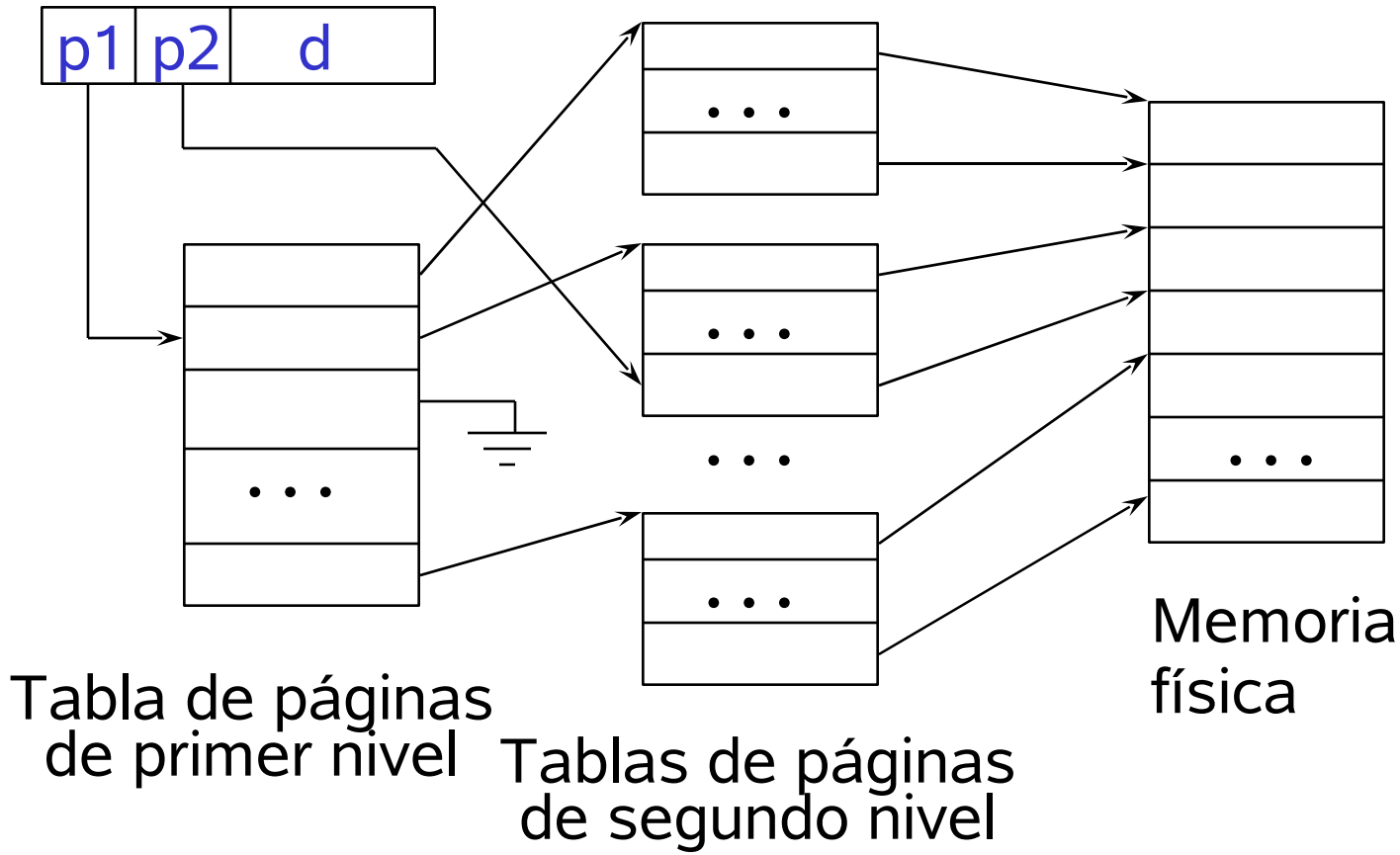
Paginación multinivel

- Esta solución opta por “paginar las tablas de páginas”
- La partición de la tabla de páginas permite al SO dejar particiones no usadas sin cargar hasta que el proceso las necesita. Aquellas porciones del espacio de direcciones que no se usan no necesitan tener tabla de página

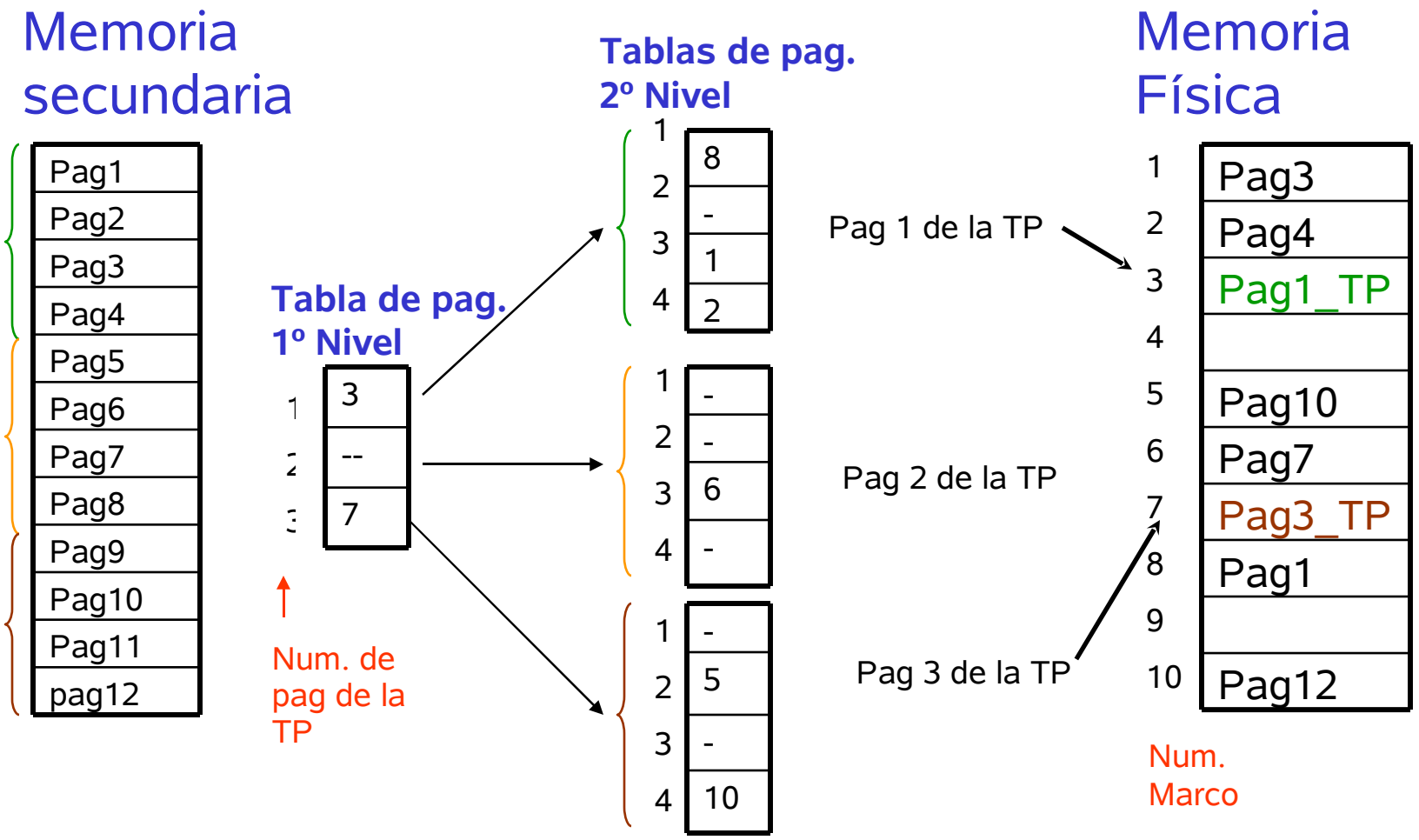
Paginación a dos niveles

- Lo que hacemos es paginar la tabla de páginas.
- La dirección lógica se divide en:
  - » número de página (n bits):
    - – un número de página p1 (=k)
    - – desplazamiento de página p2 (=n-k)
  - » desplazamiento de página d (m bits)
- Así una dirección lógica es de la forma:

Esquema de paginación a dos niveles

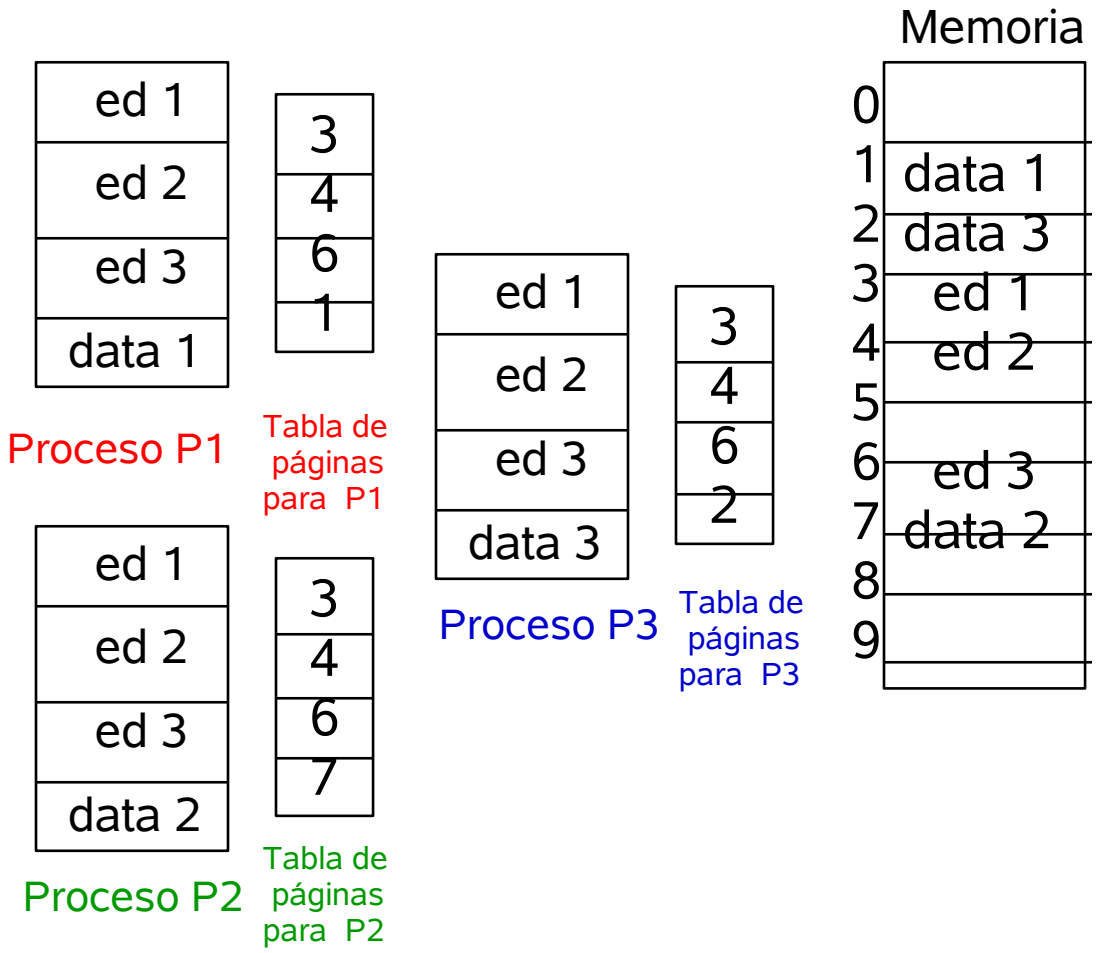


Ejemplo: Esquema de paginación a dos niveles



Páginas compartidas

- Una copia de código de solo lectura (*reentrante*) compartido entre varios procesos. Ej. editores, compiladores, sistemas de ventanas



5.3 Segmentación

- Esquema de organización de memoria que soporta mejor la visión de memoria del usuario: un programa es una colección de unidades lógicas -segmentos-, p. ej. procedimientos, funciones, pila, tabla de símbolos, matrices, etc.

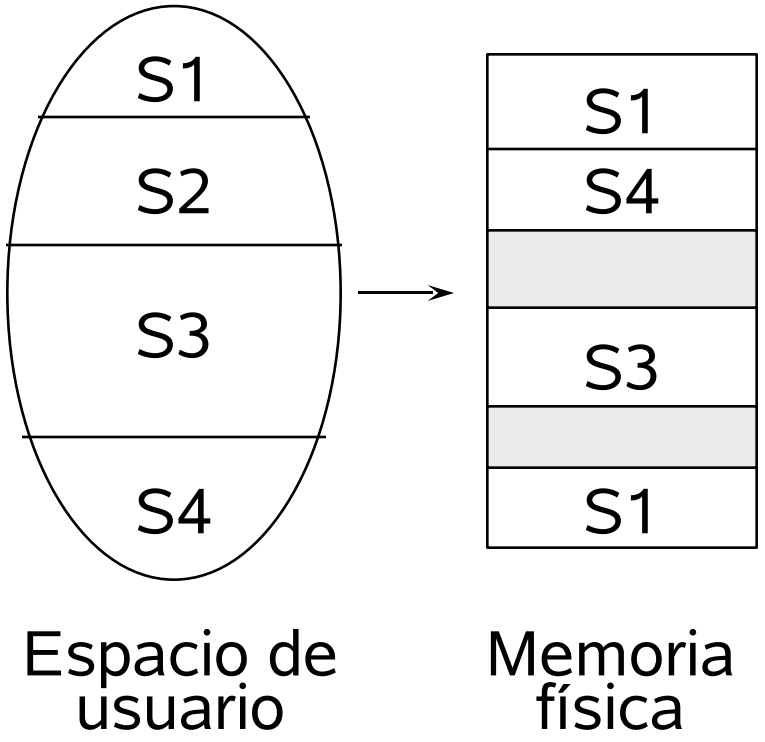


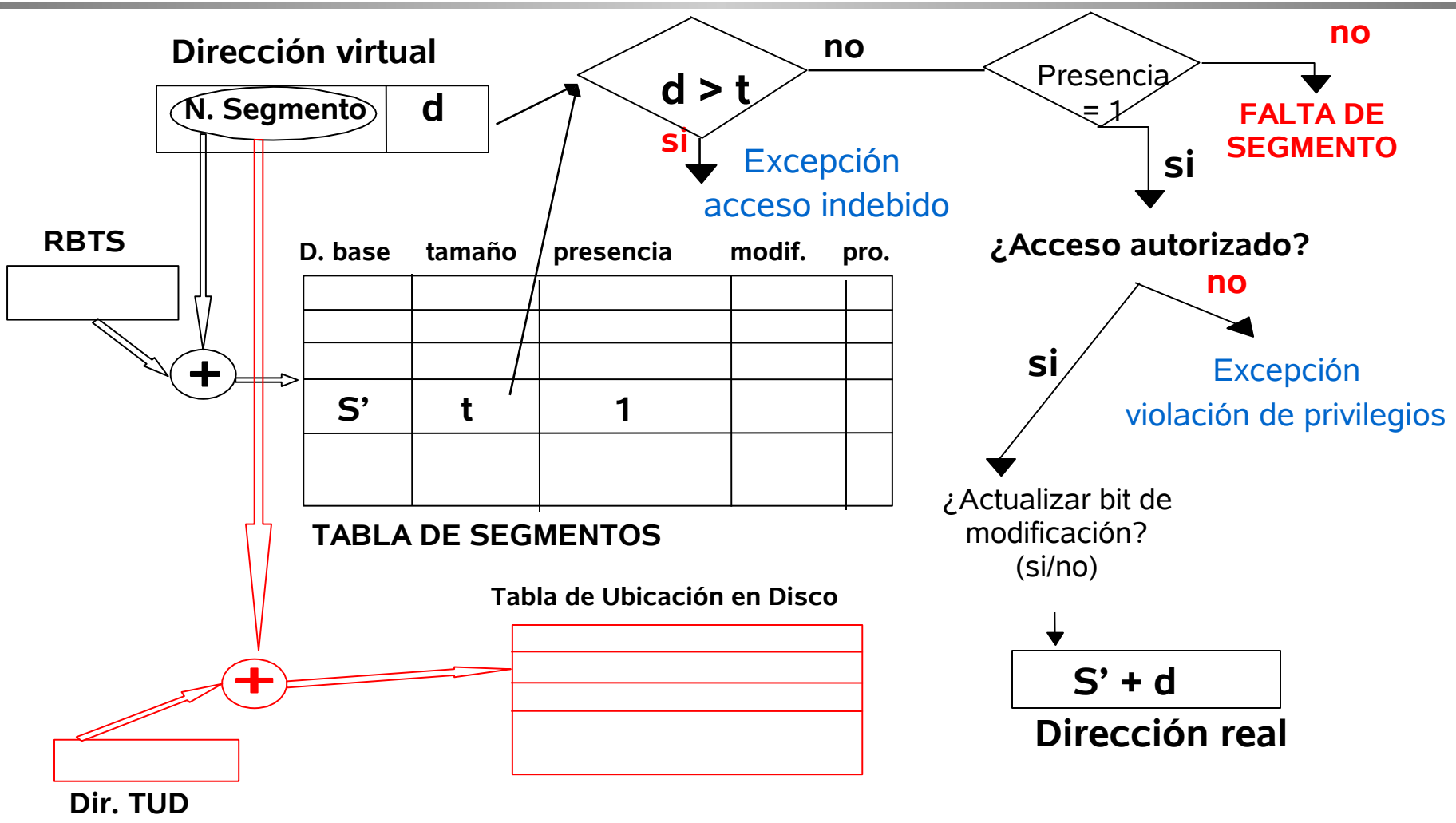
Tabla de Segmentos

- Una dirección lógica es una tupla:
  <número_de_segmento, desplazamiento>
- La *Tabla de Segmentos* aplica direcciones bidimensionales definidas por el usuario en direcciones físicas de una dimensión. Cada entrada de la tabla tiene los siguientes elementos (aparte de presencia, modificación y protección):
  - » *base* - dirección física donde reside el inicio del segmento en memoria.
  - » *tamaño* - longitud del segmento.

Implementación de la Tabla de Segmentos

- La tabla de segmentos se mantiene en memoria principal
- El *Registro Base de la Tabla de Segmentos* (*RBTS*) apunta a la tabla de segmentos (suele almacenarse en el PCB del proceso)
- El *Registro Longitud de la Tabla de Segmentos* (*STLR*) indica el número de segmentos del proceso; el nº de segmento *s*, generado en una dirección lógica, es legal si $s < STLR$ (suele almacenarse en el PCB del proceso)

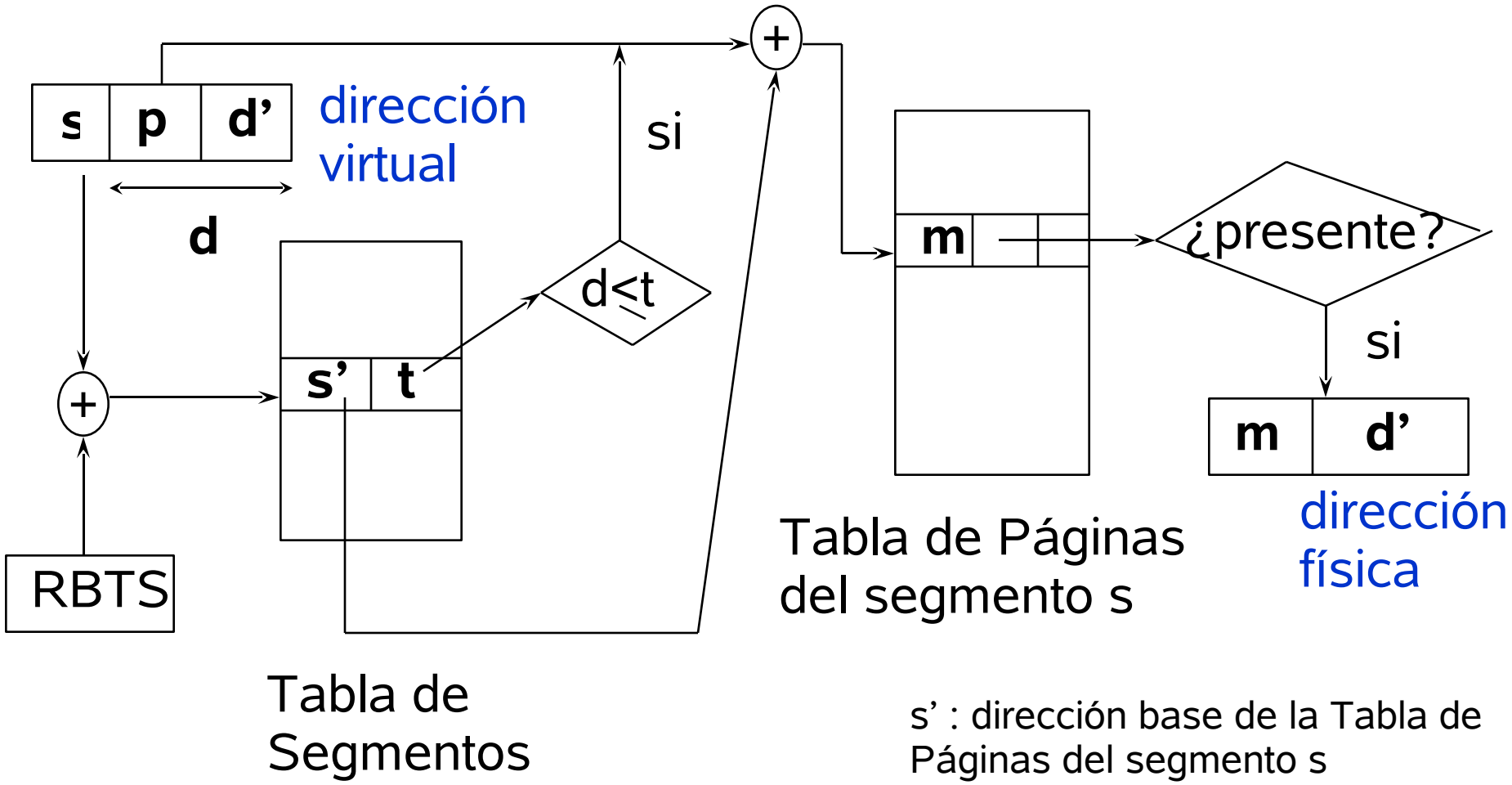
Esquema de traducción



5.4 Segmentación Paginada

- La variabilidad del tamaño de los segmentos y el requisito de memoria contigua dentro de un segmento, complica la gestión de MP y MS
- Por otro lado, la paginación simplifica la gestión pero complica más los temas de compartición y protección
- Algunos sistemas combinan ambos enfoques, obteniendo la mayoría de las ventajas de la segmentación y eliminando los problemas de una gestión de memoria compleja

Esquema de traducción



s' : dirección base de la Tabla de Páginas del segmento s

6. Gestión de la Memoria Virtual

① Introducción

② Algoritmos de sustitución

③ Comportamiento de los programas

④ Hiperpaginación (*Thrashing*)

⑤ Algoritmos de asignación y sustitución

6.1 Introducción

- Gestión de Memoria Virtual con paginación
- Criterios de clasificación respecto a:
  - » *Políticas de asignación*:
    - –Fija
    - –Variable
  - » *Políticas de búsqueda*:
    - –Paginación por demanda
    - –Paginación anticipada (≠ prepaginación)
  - » *Políticas de sustitución*:
    - –Sustitución local
    - –Sustitución global

Introducción (y II)

- Independientemente de la **política de sustitución** utilizada, existen ciertos criterios que siempre deben cumplirse:
  - » Páginas "limpias" frente a "sucias"
    - – se pretende minimizar el coste de transferencia
  - » Páginas compartidas
    - – se pretende reducir el nº de faltas de página
  - » Páginas especiales
    - – algunos marcos pueden estar bloqueados (ejemplo: buferes de E/S mientras se realiza una transferencia)

Asignación fija

- Asignación por igual
  - » Se asignan el mismo número de marcos a todos los procesos
  - » Si hay m marcos, y n procesos. A cada proceso se se asignan *m/n* marcos
- Asignación proporcional por tamaño
  - » Asigna según tamaño del proceso
  - » s_i=tamaño de p_i
  - » $S = \sum s_i$
  - » m = número total de marcos
  - » la asignación, a_i, para p_i es:

    $a_i = (s_i / S)*m$

Paginación por demanda frente a anticipada

- Las ventajas de la paginación **por demanda** son:
  - » Se garantiza que en MP solo están las páginas necesarias en cada momento
  - » La sobrecarga de decidir qué páginas llevar a MP es mínima
- Las ventajas de la paginación **anticipada** son:
  - » Se puede optimizar el tiempo de respuesta para un proceso pero los algoritmos son más complejos y se consumen más recursos

Rendimiento de la paginación por demanda

- Sea *p* la tasa de falta de página; p=0 no hay faltas de páginas, ó p=1, toda referencia es una falta. Por tanto, $\mathbf{0 \leq p \leq 1}$
- **TAE** = (1 - p) * acceso_a_memoria
  + p * (sobrecarga_falta_de_página
  + [sacar_fuera_la_página]
  + traer_la_página
  + sobrecarga_de_rearranque)

Influencia del tamaño de página

- Cuanto más pequeñas
  - » Aumento del tamaño de las tablas de páginas
  - » Aumento del nº de transferencia MP↔Disco
  - » Reducen la fragmentación interna
- Cuanto más grandes
  - » Grandes cantidades de información que no serán usadas están ocupando MP
  - » Aumenta la fragmentación interna
- Búsqueda de un equilibrio

6.2 Algoritmos de sustitución

- Podemos tener las siguientes combinaciones
  - » asignación fija y sustitución local
  - » asignación variable y sustitución local
  - » asignación variable y sustitución global
- Veremos distintos algoritmos de sustitución y nos basaremos (por simplicidad) en que se utiliza una política de asignación fija y sustitución local
- *Cadena de referencia*, $\omega = r_1, r_2, r_3, \ldots, r_i, \ldots$: secuencia de números de páginas referenciadas por un proceso durante su ejecución

Algoritmo Optimo

- Se sustituye la página que no será objeto de ninguna referencia posterior o que se referencie más tarde

» 4 marcos de página
» Problema: debemos tener un "conocimiento perfecto" de la cadena de referencia
» Se utiliza para medir cómo de bien se comportan otros algoritmos
» Faltas de páginas: 6

1,2,3,4,1,2,5,1,2,3,4,5

1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5
*	*	*	*			*				*	

Algoritmo FIFO

- Se sustituye la página por orden cronológico de llegada a MP (la página más antigua)

» 4 marcos de página
» Faltas de página: 10
» Sufre de la *Anomalía de Belady*: "más marcos no implican menos faltas de páginas"

1,2,3,4,1,2,5,1,2,3,4,5

1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	1	1	1	1	5	
		3	3	3	3	3	2	2	2	2	
			4	4	4	4	4	3	3	3	
*	*	*	*			*	*	*	*	*	

Algoritmo LRU

- Se sustituye la página que fue objeto de la referencia más antigua (*Least Recently Used*)

» 4 marcos de página
» Faltas de página: 8
» Implementación del algoritmo:
  – con contadores
  – con pila
» Mayor coste

1,2,3,4,1,2,5,1,2,3,4,5

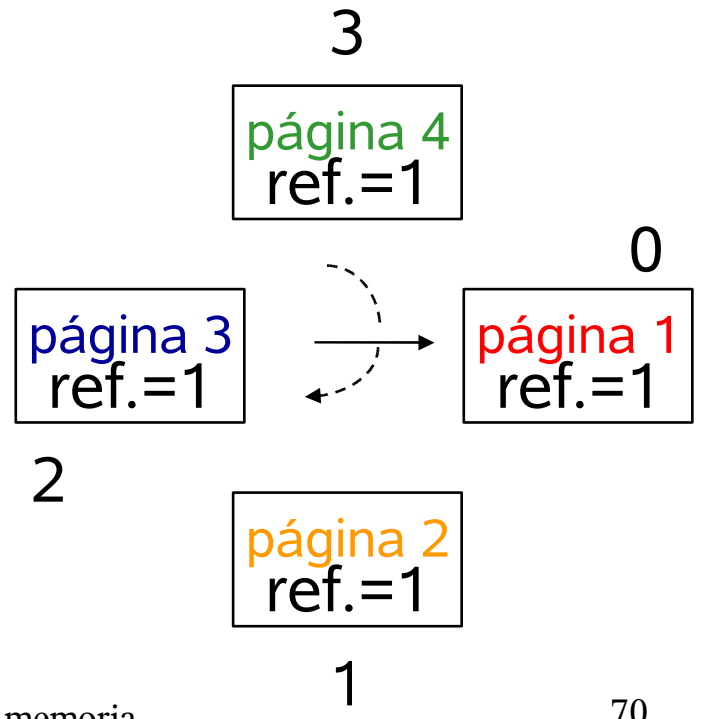
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3
*	*	*	*			*			*	*	*

Implementaciones de LRU

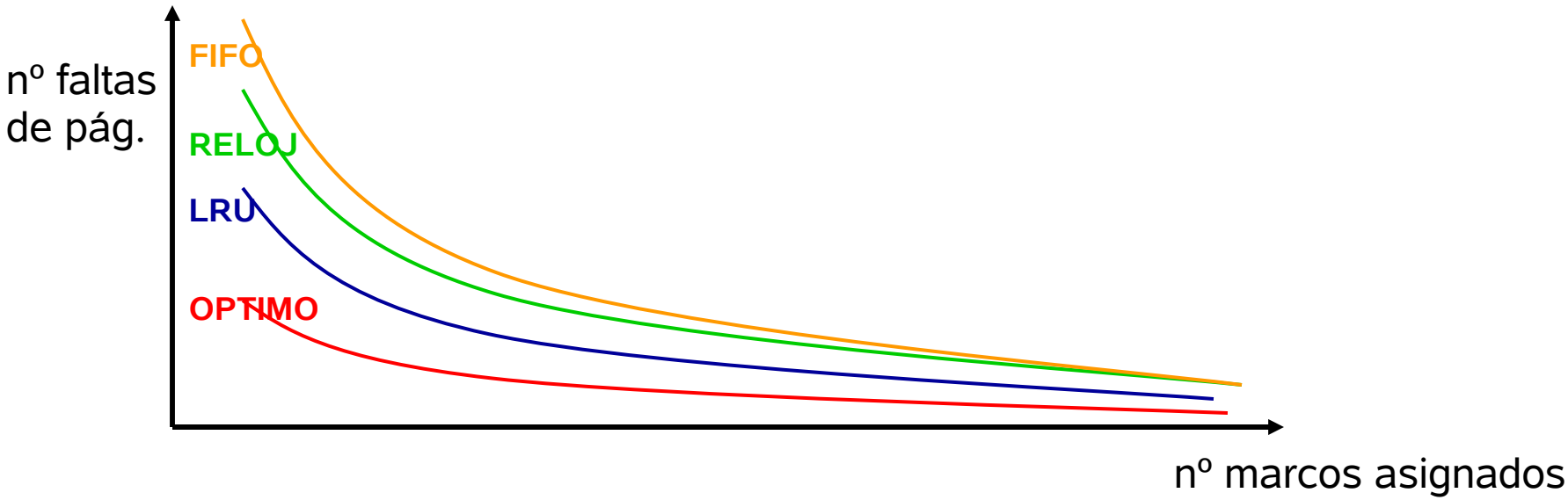
- LRU con contador
  - » Cada entrada de la tabla de páginas tiene un contador. Cada vez que se referencia la página, se copia el tiempo del reloj en el contador
  - » Cuando necesitamos cambiar una página, se miran todos los contadores y se elige la que tiene el menor tiempo
- LRU con pila
  - » Los números de páginas se mantienen en una pila (lista doblemente enlazada). Cuando se referencia una página se mueve a la cima de la pila (cambio de seis punteros como máximo)
  - » No hay que hacer búsqueda para la sustitución de una página, se sustituye la del fondo de la pila

Algoritmo del reloj

- Cada página tiene asociado un bit de referencia R (lo pone a 1 el hardware)
- Los marcos de página se representan por una lista circular y un puntero a la página visitada hace más tiempo
- Selección de una página:
  1. Consultar marco actual
  2. ¿Es R=0?
     - No: R=0; ir al siguiente marco y volver al paso 1
     - Si: seleccionar para sustituir e incrementar posición



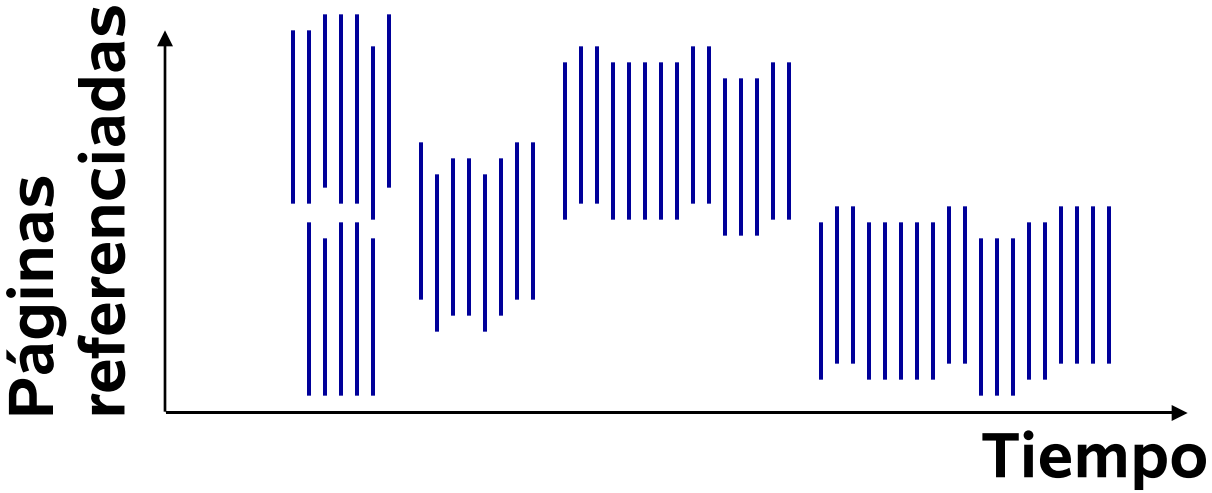
Comparación



- Conclusión:
  - » Influye más la cantidad de MP disponible que el algoritmo de sustitución usado

6.3 Comportamiento de los programas

- Viene definido por la secuencia de referencias a página que realiza el proceso
- Importante para maximizar el rendimiento del sistema de memoria virtual (TLB, alg. sustitución, ...)



Propiedad de localidad

- Distintos tipos
  - » Temporal: Una posición de memoria referenciada recientemente tiene una probabilidad alta de ser referenciada en un futuro próximo (ciclos, rutinas, variables globales, ...)
  - » Espacial: Si cierta posición de memoria ha sido referenciada es altamente probable que las adyacentes también lo sean (array, ejecución secuencial, ...)

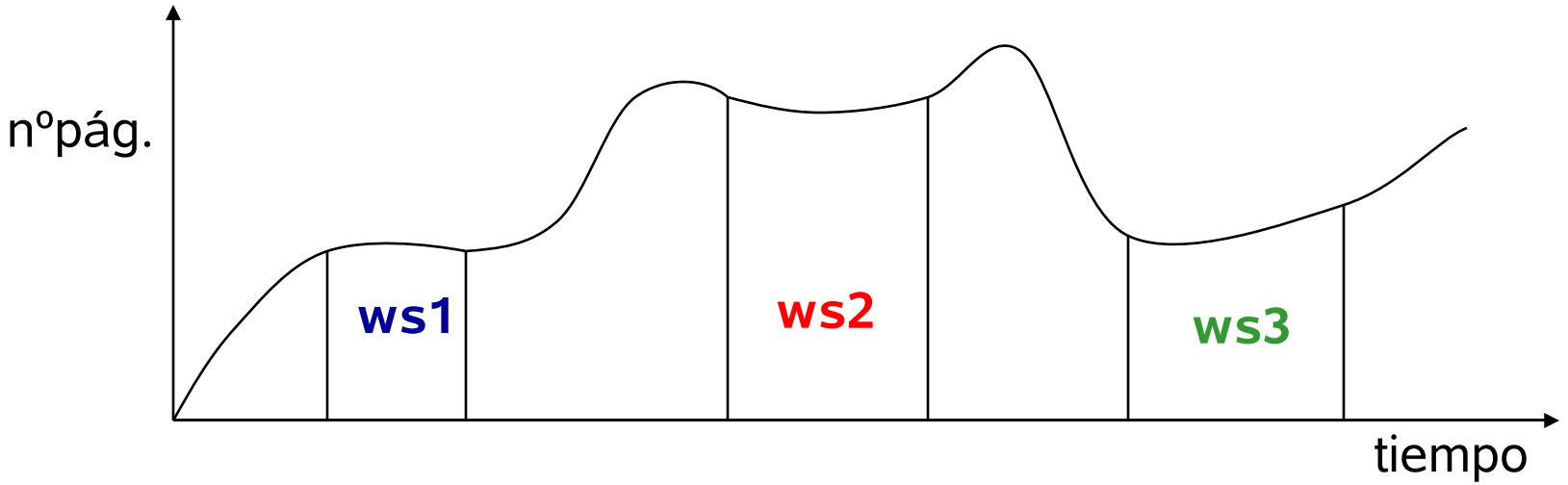
Conjunto de Trabajo

- Observaciones:
  - » Mientras el conjunto de páginas necesarias puedan residir en MP, el nº de faltas de página no crece mucho
  - » Si eliminamos de MP páginas de ese conjunto, la activación de paginación crece mucho
- Conjunto de trabajo *(Working Set)* de un proceso es el conjunto de páginas que son referenciadas frecuentemente en un determinado intervalo de tiempo

WS(t,Δ) = páginas referenciadas en el intervalo de tiempo t - Δ y t

Conjunto de Trabajo: propiedades

- Propiedades
  - » Los conjuntos de trabajo son transitorios
  - » No se puede predecir el tamaño futuro de un conjunto de trabajo
  - » Difieren unos de otros sustancialmente

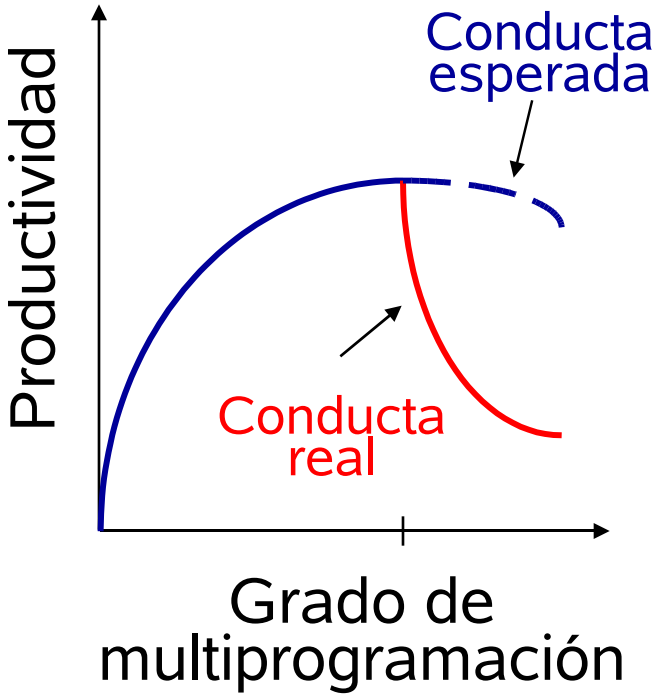


Teoría del Conjunto de Trabajo

- Un proceso sólo puede ejecutarse si su conjunto de trabajo está en memoria principal
- Una página no puede retirarse de memoria principal si está dentro del conjunto de trabajo del proceso en ejecución

6.4 Hiperpaginación

- Si un proceso no tiene "suficientes" páginas, la tasa de faltas es alta
  - » bajo uso de la CPU
  - » el SO incrementa el grado de multiprogramación
  - » más faltas de páginas
- *Hiperpaginación* = el sistema operativo está ocupado en resolver faltas de página



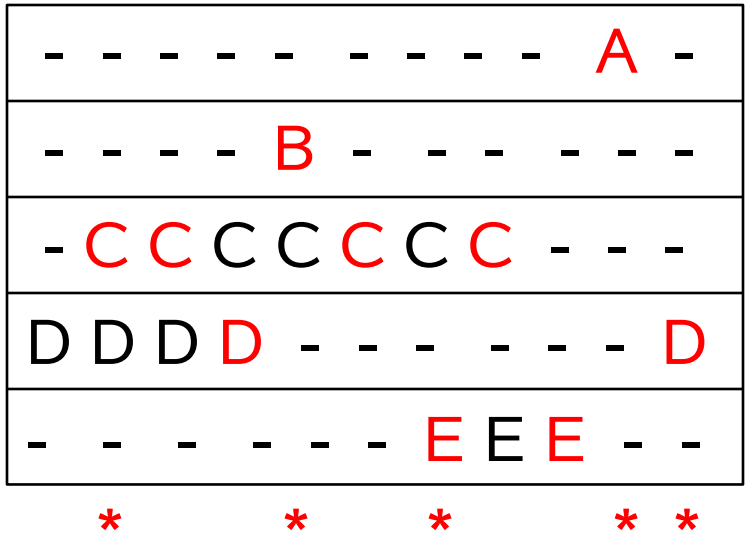
Hiperpaginación (y II)

- Formas de evitar la hiperpaginación:
  - » Asegurar que cada proceso existente tenga asignado un espacio en relación a su comportamiento ➔ Algoritmos de asignación variables
  - » Actuar directamente sobre el grado de multiprogramación ➔ Algoritmos de regulación de carga

6.5 Algoritmos de asignación y sustitución: Algoritmo Optimo

- En cada referencia, busca qué página (en memoria real) no es referenciada en el intervalo de tiempo: **[t, t + Δ]** y ésta es la desplazada. **Δ:** Tamaño de la ventana para cada proceso
- Requiere un conocimiento "a priori" de la cadena de páginas referenciadas (irrealizable)

C, C, D, B, C,E,C, E,A,D



– En la figura se representan las páginas del proceso que están en MP
– Proceso de 5 páginas
– Δ = 4
– En t=0 sólo la página D reside en MP

Algoritmo basado en el modelo del WS

- En cada referencia, determina el conjunto de trabajo: páginas referenciadas en el intervalo **(t - Δ, t]** y **sólo** esas páginas son mantenidas en MP

  – En la figura se representan las páginas del proceso que están en MP
  – Proceso de 5 páginas
  – Δ = 4
  – En t=0 WS = {A, D, E}, A se referenció en t=0, D en t=-1 y E en t=-2

Algoritmo FFP (Frecuencia de Falta de Página)

- **Idea**: para ajustar el conjunto de páginas de un proceso, usa los intervalos de tiempo entre dos faltas de página consecutivas:
  - » Si intervalo de tiempo grande, mayor que λ, todas las páginas no referenciadas en dicho intervalo son retiradas de MP
  - » En otro caso, la nueva página es simplemente incluida en el conjunto de páginas residentes
- Formalmente:

  t_c = instante de tº de la actual falta de página
  t_{c-1}= instante de tº de la anterior falta de página
  Z = conjunto de páginas referenciadas en un intervalo de tº
  R = conjunto de páginas residentes en MP

$$R(t_c,\lambda) = \begin{cases} Z(t_{c-1}, t_c) & \text{si } t_c - t_{c-1} > \lambda \\ R(t_{c-1}, \lambda) + Z(t_c) & \text{en otro caso} \end{cases}$$

Ejemplo del algoritmo FFP

- Se garantiza que el conjunto de páginas residentes crece cuando las faltas de página son frecuentes y decrece cuando no lo son

– En la figura se representan las páginas del proceso que están en MP
– Proceso de 5 páginas
– $\lambda = 2$
– La página A se referenció en t=-1 y E en t=-2
– D se referencia en t=0

D,C, C,D,B,C,E, C,E, A,D