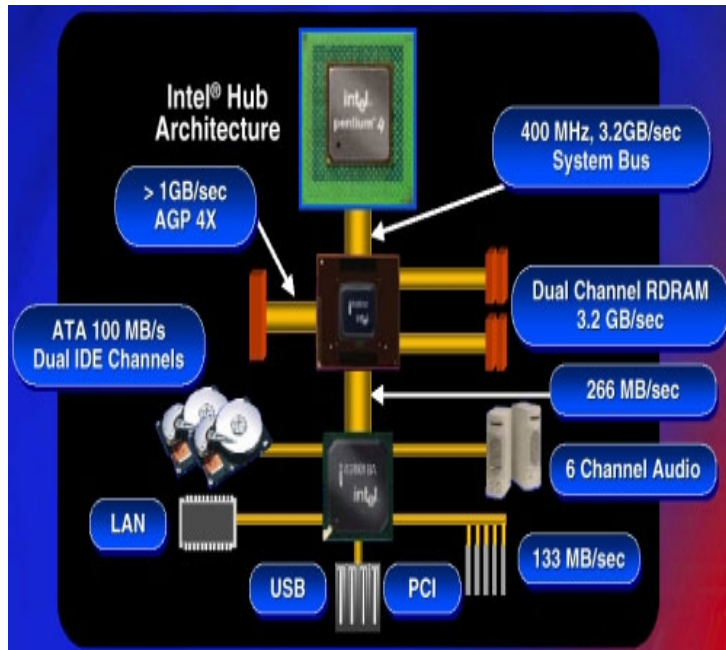


- Conceptos generales: hard / soft
- Esquemas contiguos de asignación:
 - Fragmentación
 - Compactación
- Esquemas no contiguos:
 - Paginación
 - Segmentación
 - Segmentación paginada
- Memoria Virtual: paginación por demanda



Conceptos generales hardware



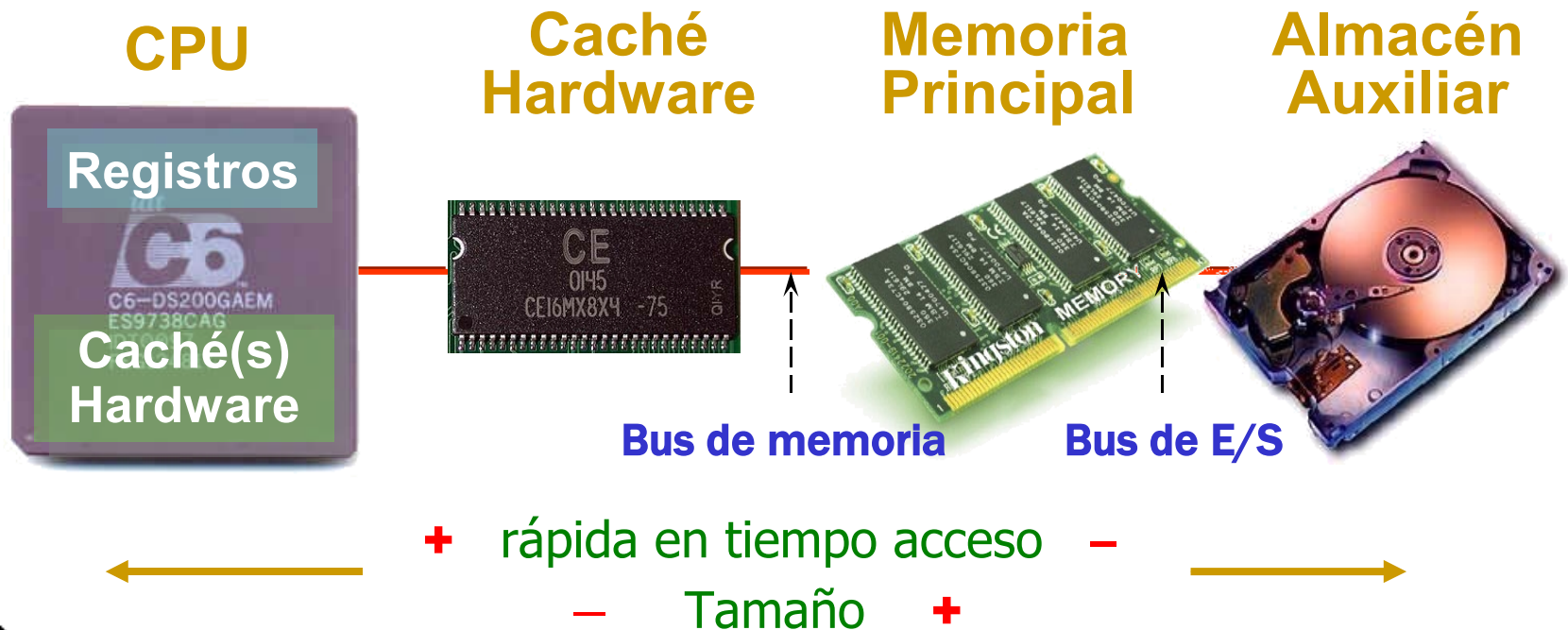
■ Jerarquía de memoria:

- Las cachés.
- Tiempo efectivo de acceso.



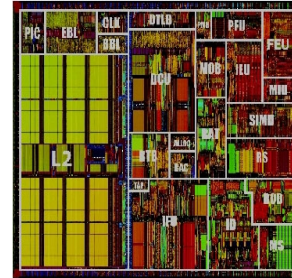
Jerarquía de Memoria

- Los computadores usan una jerarquía de memoria similar a la que muestra la figura.





Cachés



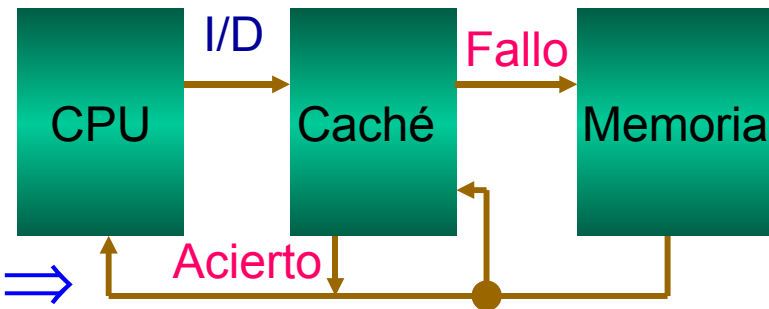
- **Caché** -contiene copia de instrucción/dato que son accedidos +rápido que el original.



Hacer los casos frecuentes eficientes, los caminos infrecuentes no importan tanto.

- **Denominamos:**

- *Acierto de caché:* item en caché.
- *Fallo de caché:* no está en caché ⇒ operación completa.





Localidad

- Las cachés “funcionan” porque explotan las **localidad** de las referencias del código, datos, y pila de los programas.
- Tipos de localidad:
 - **Espacial**: si un item es referenciado, las direcciones próximas a él tienden también a ser referenciadas.
 - **Temporal**: si un item referenciado, tiende de nuevo a ser referenciado en breve.



Tiempo de Acceso Efectivo

- Cuando usamos cachés ¿Cuánto no cuesta acceder a memoria? Denominamos ...
- **Tiempo efectivo de acceso** (TAE) al tiempo medio de acceso a una celda de memoria:

$$\text{TAE} = p * t_a + (1-p) * t_f$$

Donde:

- p = probabilidad de acierto.
- t_a = tiempo de acceso si hay acierto.
- $1-p$ = probabilidad de fallo.
- t_f = tiempo de acceso si hay fallo.



Conceptos generales software



- Requisitos de la gestión de memoria.
- Niveles de gestión de memoria.
- Compilación de programas:
 - Enlace y carga.
- Espacios de direcciones: lógico y físico:
 - Traducción de direcciones
 - Apoyo hardware a la traducción: la MMU



Requisitos de la gestión de memoria (i)

- El SO **asigna memoria** a los procesos para su ejecución, garantizando:
 - **Protección:**
 - ◆ Un proceso no accede a memoria de otro.
 - ◆ Diferentes módulos del programa deben tener diferentes permisos de acceso.
 - **Compartición:**
 - ◆ De datos/código entre procesos.
 - ◆ Permite el ahorro de memoria.
 - **Reubicación:** En sists. multiprogramados, un programa debe poder cargarse en diferentes zonas de memoria.

Contrapuestos



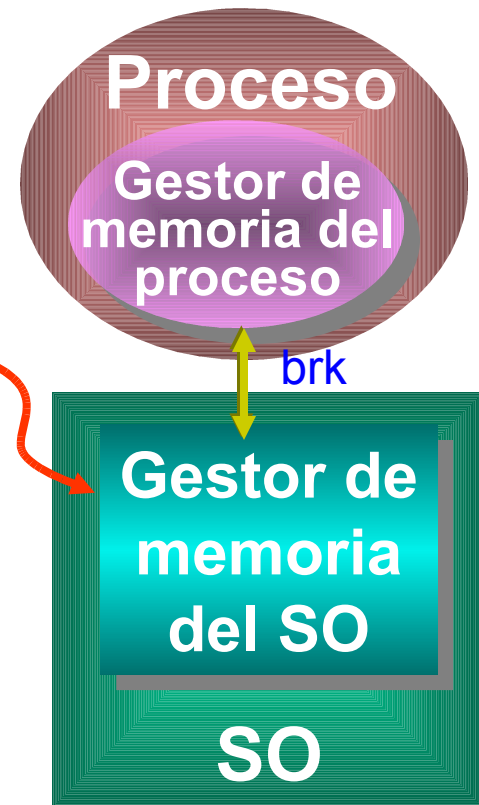
Requisitos de la gestión de memoria (y ii)

- El SO debe esconder la
 - **organización física** (jerarquía de niveles, estructura no lineal) de la memoria física.
 - para que el usuario tenga una **visión lógica** de la memoria como una matriz lineal. Además permita la estructuración de un programa en módulos.

Niveles de gestión de memoria

- Existen dos niveles:
 - **Gestor de memoria del SO** –asigna porciones de memoria al proceso
 - **Gestor de memoria del proceso** –gestiona estas porciones (p. ej. a través de *malloc* y *free*).

Nuestro objetivo





Procesamiento de un programa

- Etapas por las que pasa un programa antes de cargarse en memoria.





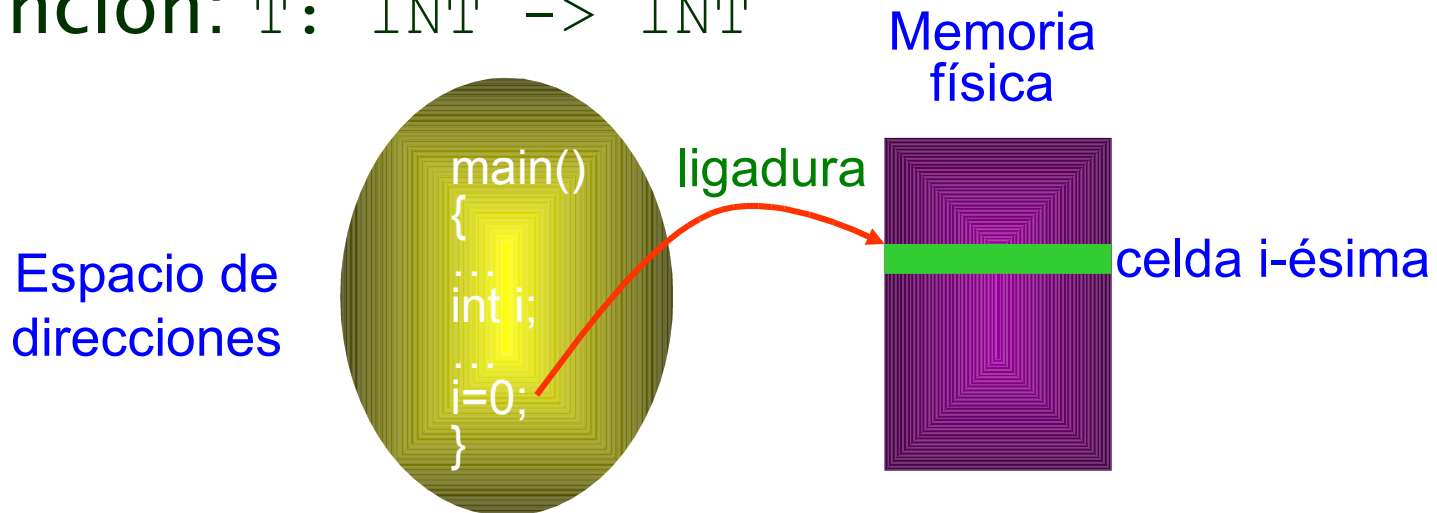
Compiladores, enlazadores y cargadores

- **Compiladores y ensambladores:** Sus salidas contienen direcciones reubicables y referencias externas.
- **Enlazadores:** resuelven las referencias externas de las subrutinas compiladas o ensambladas por separado.
- **Cargadores:** ligan direcciones reubicables a direcciones absolutas.



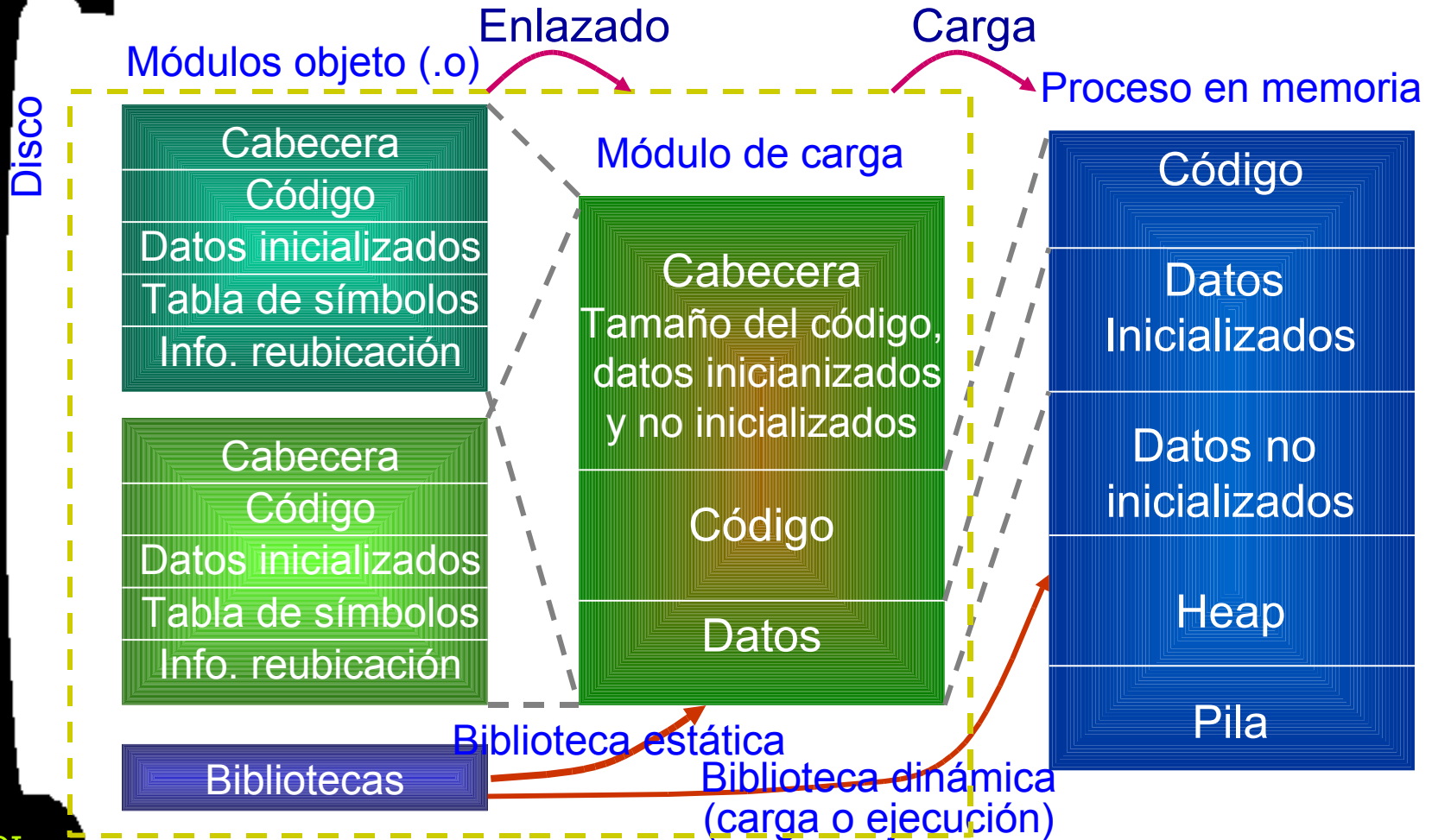
Ligadura de direcciones

- **Ligadura de direcciones** – correspondencia entre direcciones, de instrucciones y datos, de nuestro programa con las direcciones de memoria.
- Es un esquema de designación; una función: $T: \text{INT} \rightarrow \text{INT}$





Enlazado y carga





¿Cuándo realizar la ligadura? (i)

■ Tiempo de compilación:

- El compilador genera código absoluto. Para ello debemos conocer las direcciones de memoria donde se va a cargar el programa.
- ➔ Hay que recompilar el programa si se cambia en dirección de inicio.



Cuando hacer ligadura (ii)

■ Tiempo de carga:

- Se genera **código reubicable** si no se conoce la ubicación en memoria en tiempo de compilación.
- El procesador debe disponer al menos de un **registro de reubicación**, que contendrá la dirección de inicio del programa. El SO carga el programa en memoria y ajusta el registro de reubicación.



Cuando hacer ligadura (iii)

- **Tiempo de ejecución:**
 - Si la ligadura se retrasa hasta la ejecución del proceso, este puede moverse durante su ejecución de una zona de memoria a otra.
 - Necesitamos soporte hardware para la correspondencia de direcciones. Lo veremos para el caso de segmentación y paginación.



Ligaduras dinámicas

- Para programas con mucho código, el módulo de carga presenta dos problemas:
 - Su tamaño en disco es grande.
 - Su tiempo de carga es elevado.
- El SO mitiga el problema mediante las ligaduras dinámicas. Para ello:
 - Retrasa la carga de un módulo hasta que este es necesario – **carga dinámica** (o enlace dinámico en tiempo de carga).
 - Retrasar el propio enlace hasta la ejecución – **enlace dinámico** (o enlace dinámico en tiempo de ejecución).



Ligaduras dinámica (y ii)

■ Carga dinámica:

- ◆ Mejor uso del espacio de memoria. Las rutinas no usadas no se cargan.
- ◆ No necesita soporte especial del SO; implementado en el diseño del programa o biblioteca.
- ◆ No solventa el problema del tiempo de carga.

■ Enlace dinámico:

- El enlazador inserta *tocones (stubs)* para localizar la biblioteca residente adecuada.
- El SO comprueba que la rutina está en el espacio de direcciones del proceso.
- Más costosa que la carga estática pero ahorra espacio y el arranque más rápido.



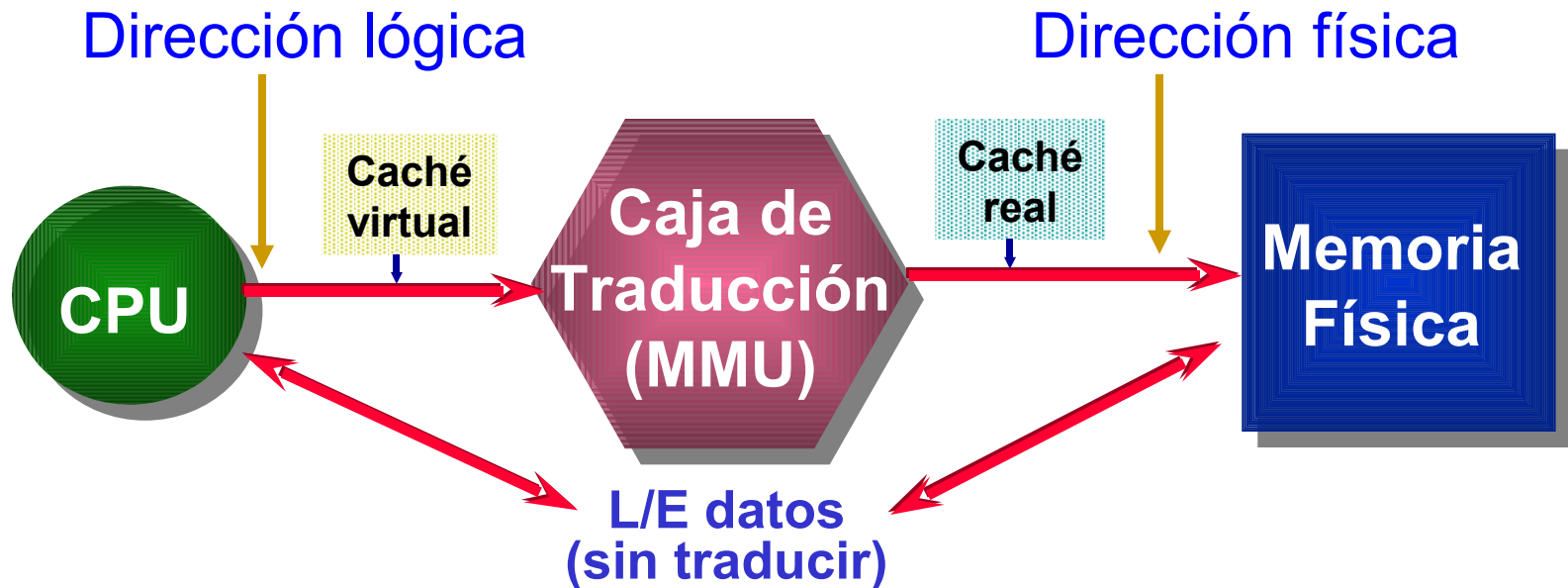
Espacio lógico y espacio físico

- La necesidad de poder reubicar un programa en memoria, hace necesario separar el espacio de direcciones generadas por el compilador, *espacio lógico o virtual*, del espacio físico en el que se carga, el *espacio de direcciones físicas*.
- Denominamos:
 - **Dirección lógica** – la generada por la CPU; también conocida como virtual.
 - **Dirección física** – dirección que se pasa al controlador de memoria.



Traducción de direcciones

- La separación de espacios me obliga a realizar una traducción de direcciones:



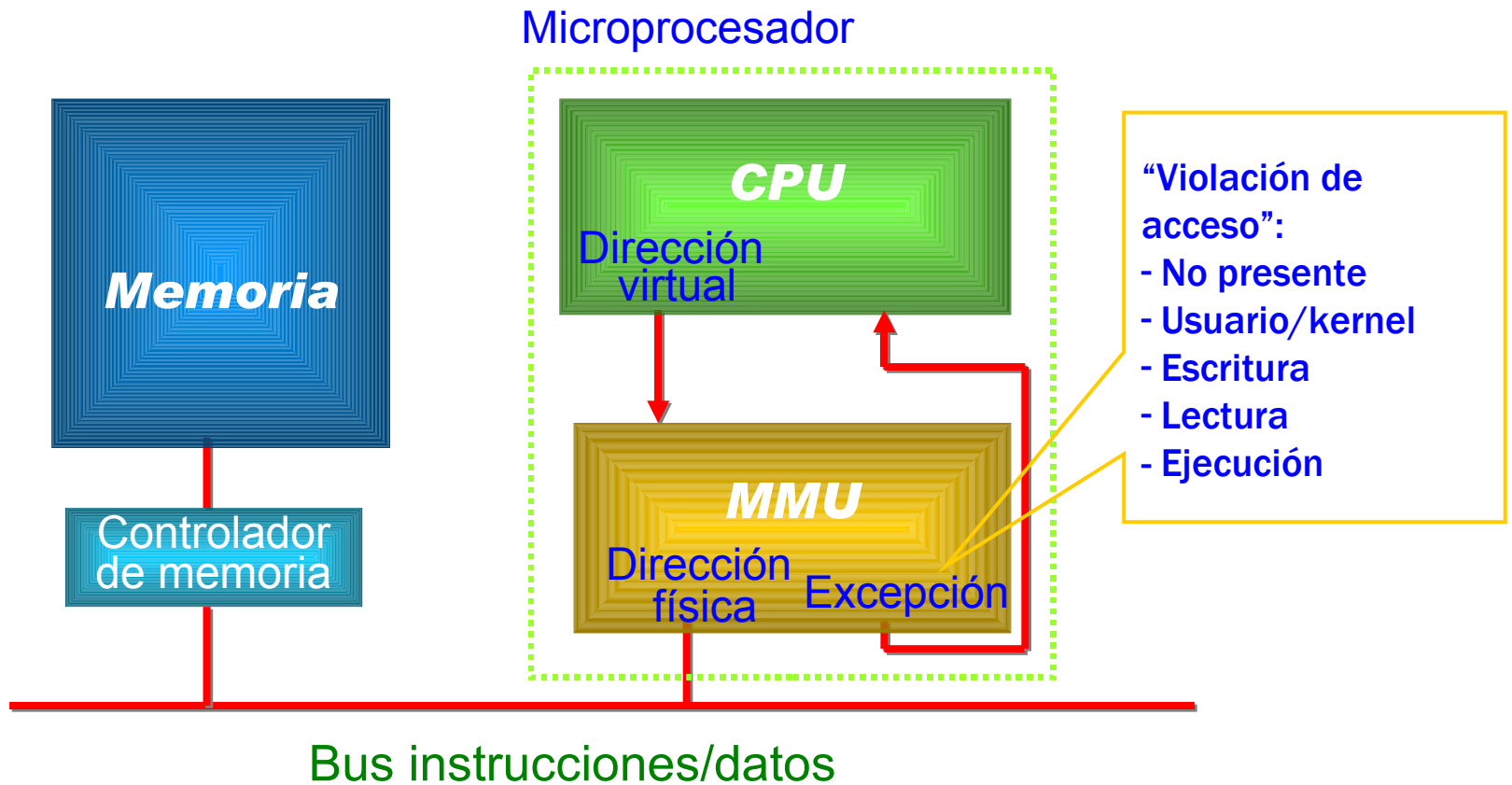



Unidad de Gestión de Memoria

- **MMU** (*Memory Management Unit*) – dispositivo hardware que traduce direcciones virtuales en direcciones físicas. También implementa protección.
- El hardware determina la forma en la que el SO gestiona la MMU.
- En el esquema MMU más simple, el valor del registro de reubicación se añade a cada dirección generada por el proceso de usuario al mismo tiempo que es enviado a memoria.

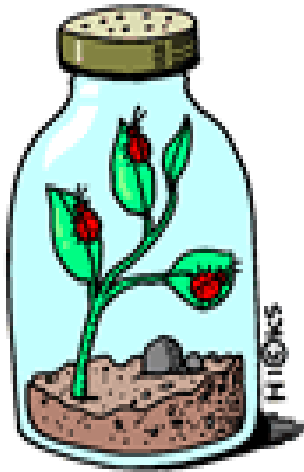


MMU





Esquemas simples de asignación



- Gestión de memoria en sists. multiprogramados:
 - Asignación estática.
 - Asignación dinámica.
- Intercambio
- Fragmentación
- Compactación



Organización contigua del almacenamiento

- En este apartado veremos esquemas de **asignación contigua**, es decir, la memoria principal asignada a un proceso es un único bloque de memoria contigua.
- En el tema siguiente, veremos esquemas de **asignación no contigua**, es decir, permitimos que el programa este dividido en bloques, o segmentos, que se pueden colocar en zonas no necesariamente contiguas de memoria principal.



Particiones

- Dividimos la memoria en particiones:
 - El SO ocupa permanentemente una región de memoria.
 - El resto de la memoria se particiona entre los procesos de forma:
 - Estática – número fijo de particiones; pueden ser de igual o diferente tamaño.
 - Dinámica – particiones de diferente tamaño y número.
- El SO mantiene información sobre las particiones asignadas y las libres (huecos).



Asignación en sist. de particiones variables

- Para cargar proceso en memoria se le asigna un hueco suficientemente grande como para albergarlo.
- ¿Cómo satisfacer una petición de tamaño n ? Asignar según:
 - **Primer encaje** – Asignar el primer hueco lo bastante grande.
 - **Mejor encaje** – Asignar el hueco más pequeño lo bastante grande; produce el hueco sobrante menor.

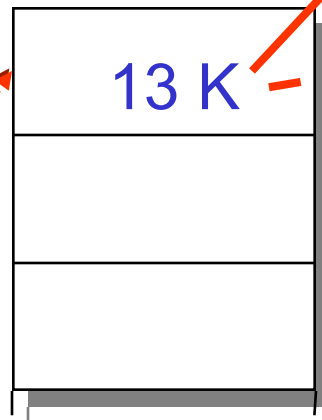


El Primer encaje y el Mejor encaje

Lista espacio libre
(orden por direcc.)

Direc. inicial	Longitud
a	16K
c	14K
e	5K
g	30K

Solicitudes



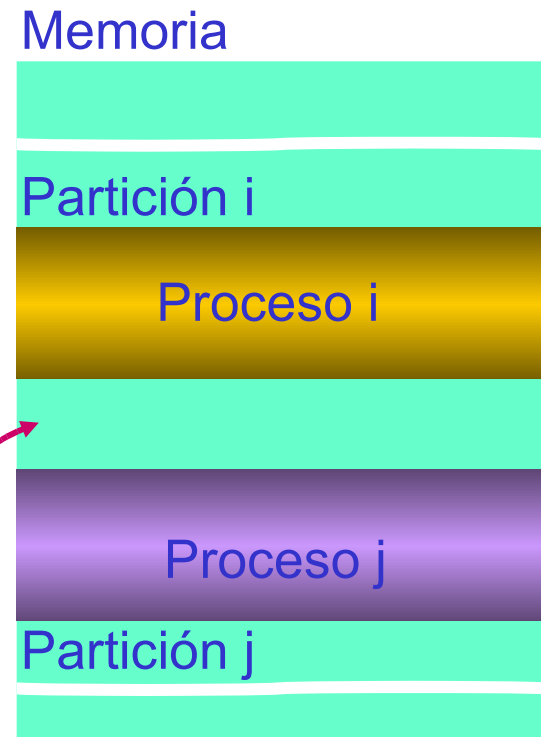
Mejor encaje:
Se ordenaría por tamaño de hueco





Fragmentación externa

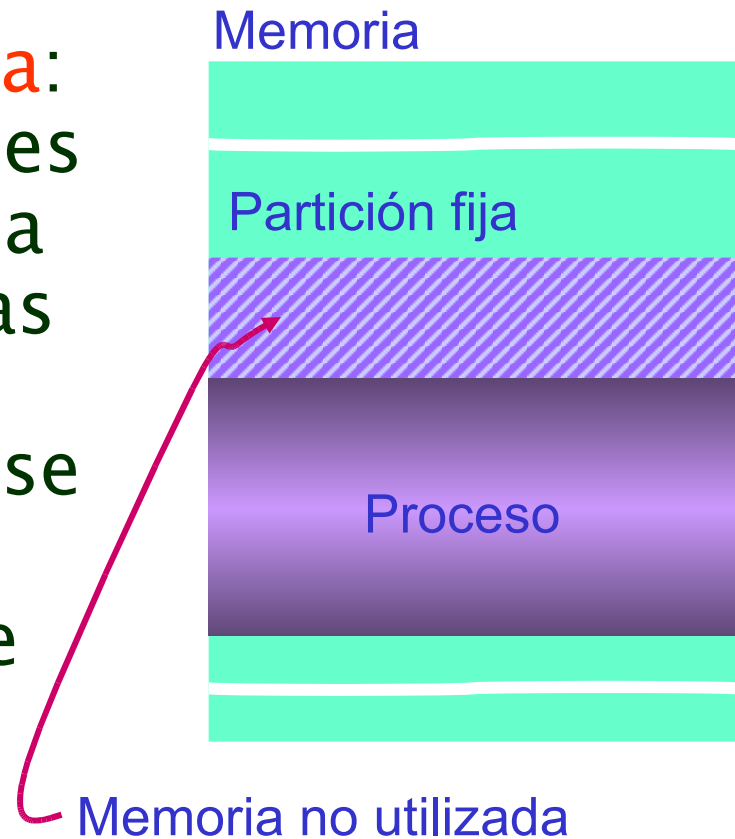
- **Fragmentación externa:** existe espacio para satisfacer una petición pero no es contiguo. Ej. No podemos satisfacer una petición de 40 KB, habiendo 62 KB libres.



Memoria no utilizada

Fragmentación interna

- **Fragmentación interna:** la memoria asignada es mayor que la usada; la diferencia entre ambas es memoria interna a una partición que no se usa.
- Se produce cuando se asigna memoria en particiones fijas.

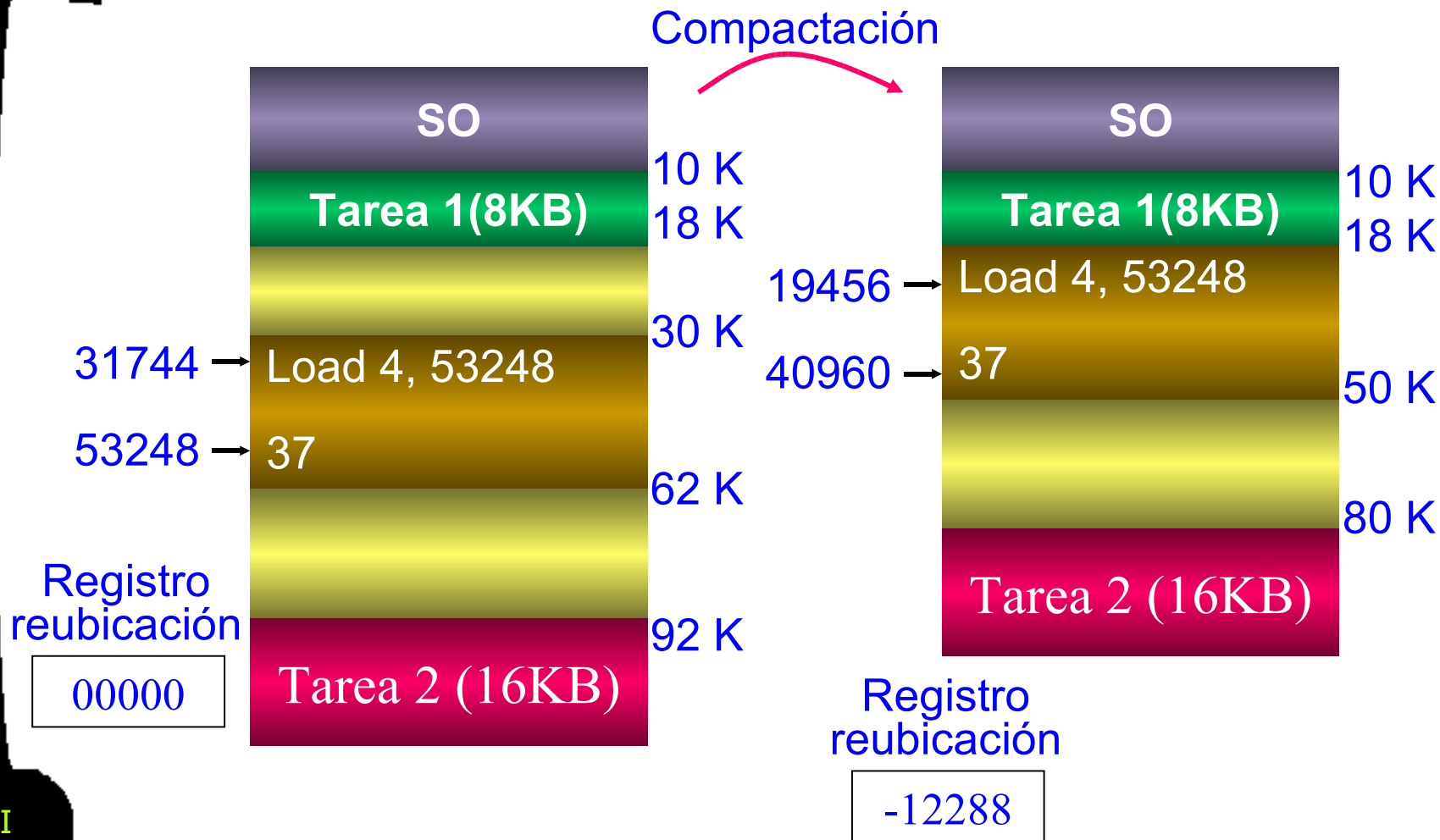




Compactación

- **Compactación** –técnica para reducir la fragmentación externa que consiste en arrastrar los contenido de memoria a un lugar para reunir la memoria libre en un bloque.
- Posible en sistemas con reubicación dinámica (se realiza en tiempo de ejecución).
- Problema: ¿procesos que realizan E/S ?:
 - Anclarlo en memoria durante la E/S
 - Realizar E/S dentro de los búferes del SO.

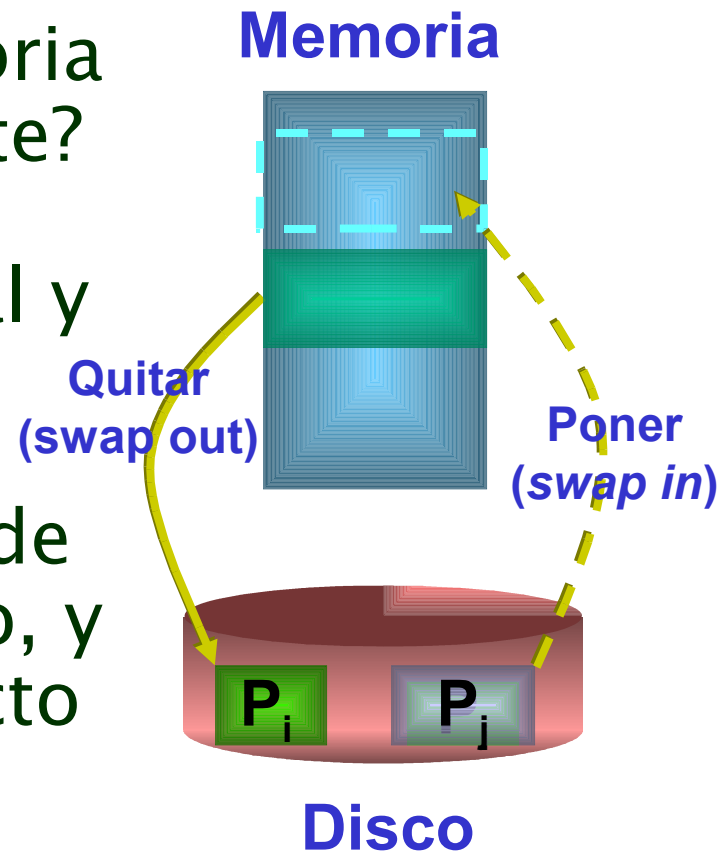
Ejemplo de compactación





Intercambio (*swapping*)

- ¿Qué ocurre si la memoria esta ocupada totalmente?
- Intercambiar procesos entre memoria principal y secundaria (un disco rápido que alberga las imágenes de memoria de los procesos de usuario, y suministra acceso directo a esas imágenes).





Intecambio (y ii)

- Aconsejable en sists. de tiempo compartido: usuarios conectados al sistema alternan periodos de trabajo e inactividad.
- Un proceso se sacará de memoria:
 - Si va a estar cierto tiempo bloqueado
 - Necesitamos compartir CPU y memoria
- El factor principal en el tiempo de intercam- bio es el tiempo de transferencia, que es proporcional a la memoria intercambiada.
- Se usa en UNIX, Windows, etc.



Gestión de memoria

- Los SOs multitarea asignan una zona de memoria a cada proceso en ejecución y deben garantizar que un proceso de usuario no lea/escriba de/en la memoria asignada a otro proceso
- Al menos, debemos de tener protección de memoria para la tabla de vectores de interrupción/excepción y las rutinas de servicio de interrupción/excepción.



Protección y reubicación

- ¿Cómo aseguramos la protección? y ¿Cómo reubicamos los procesos?
- Necesitamos dos registros: el registro base y límite. En cada *load* y *store*:

- Reubicación:

Direc. Física = Direc. Virtual + Reg. Base

- Protección – comprobar que la dirección cae en el rango [base, límite).



Registros base y límite

CPU

Registro Base: 3000

Registro Límite: 2000

⊕ Comentario:

- Proceso 1 en ejecución.
- El SO carga los registros base y límite con los valores adecuados al planificar un proceso

Memoria

SO

PID	RB	RL
1	3000	2000
2	5600	1400

3000

Proceso 1

5000

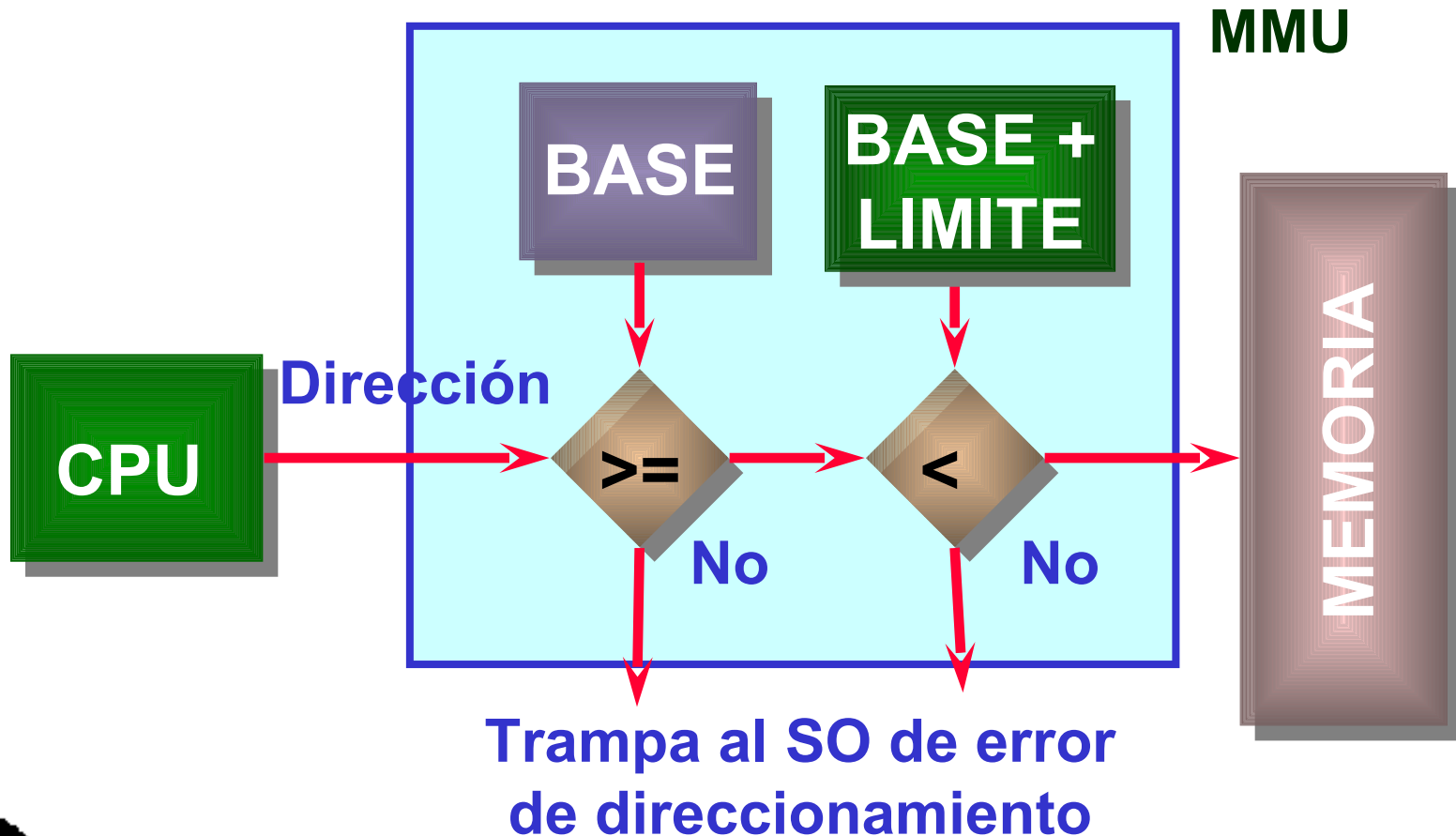
5600

Proceso 2

7000



Protección hardware del direccionamiento

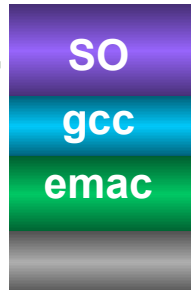




Pros y contras de los registros base y límite

- Pros: Económico en términos de:
 - Registros: sólo dos.
 - Ciclos – suma y comparación.

- Contras: ¿ cómo ...
 - aumentar la memoria de un proceso, ó
 - compartir módulos y protegerlos adecuadamente?



■ Solución: asignación no contigua ... a continuación



Organizaciones no contiguas



- Paginación
- Segmentación
- Segmentación paginada
- Ejemplo: Intel x86



Idea básica

- ¿Cómo evitar la fragmentación de memoria?...

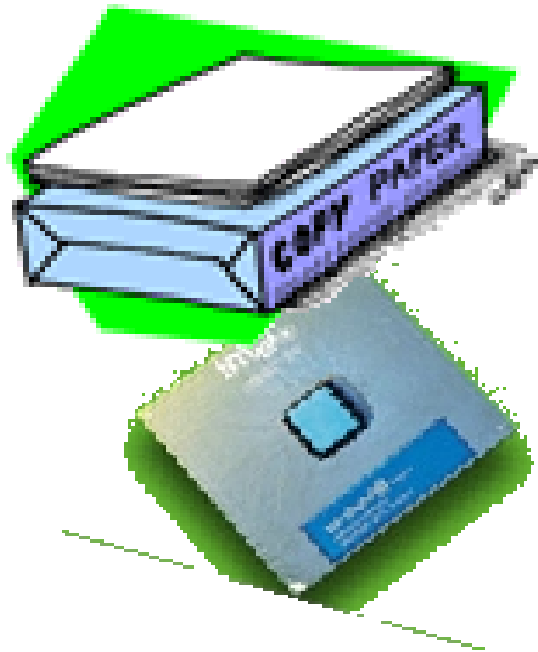


Al desacoplar los espacios lógico y físico, el espacio de direcciones de un proceso no tiene que ser contiguo.

- Segmentación – partimos el programa en trozos de diferentes tamaños.
- Paginación – hacemos que todos los bloques tengan el mismo tamaño.



Paginación



- Ideas básicas
- Traducción de direcciones
- Búfer de reconocimiento de traducciones
- Protección de memoria
- Reducción de la TP en memoria:
 - Paginación multinivel



Paginación: ideas básicas

- En segmentación la asignación de memoria es dinámica. La paginación lo evita haciendo que todos los fragmentos sean iguales.
- La MMU divide el programa en bloques de igual tamaño (en general una potencia de dos):
 - La memoria física se “divide” en bloques denominados **marcos de página**.
 - El espacio lógico se “divide” en bloques denominados **páginas**.



Paginación

- El SO mantiene el rastro de los marcos que están libres.
- Para ejecutar un programa con n páginas, necesitamos m marcos libres. Como veremos en el tema siguiente m puede ser menor que n , esto es, podemos ejecutar un programam mayor que la memoria principal disponible.



Traducción: Tabla de páginas

- La traducción de direcciones virtuales a direcciones físicas consiste básicamente en conocer donde está el marco en el que se encuentra cargada una página dada.
- La traducción se realiza mediante una estructura de datos denominada **tabla de páginas** (TP), que tiene un elemento, **entrada** (*PTE*), por cada página de proceso. El contenido de esta entrada es la dirección del marco donde está cargada.

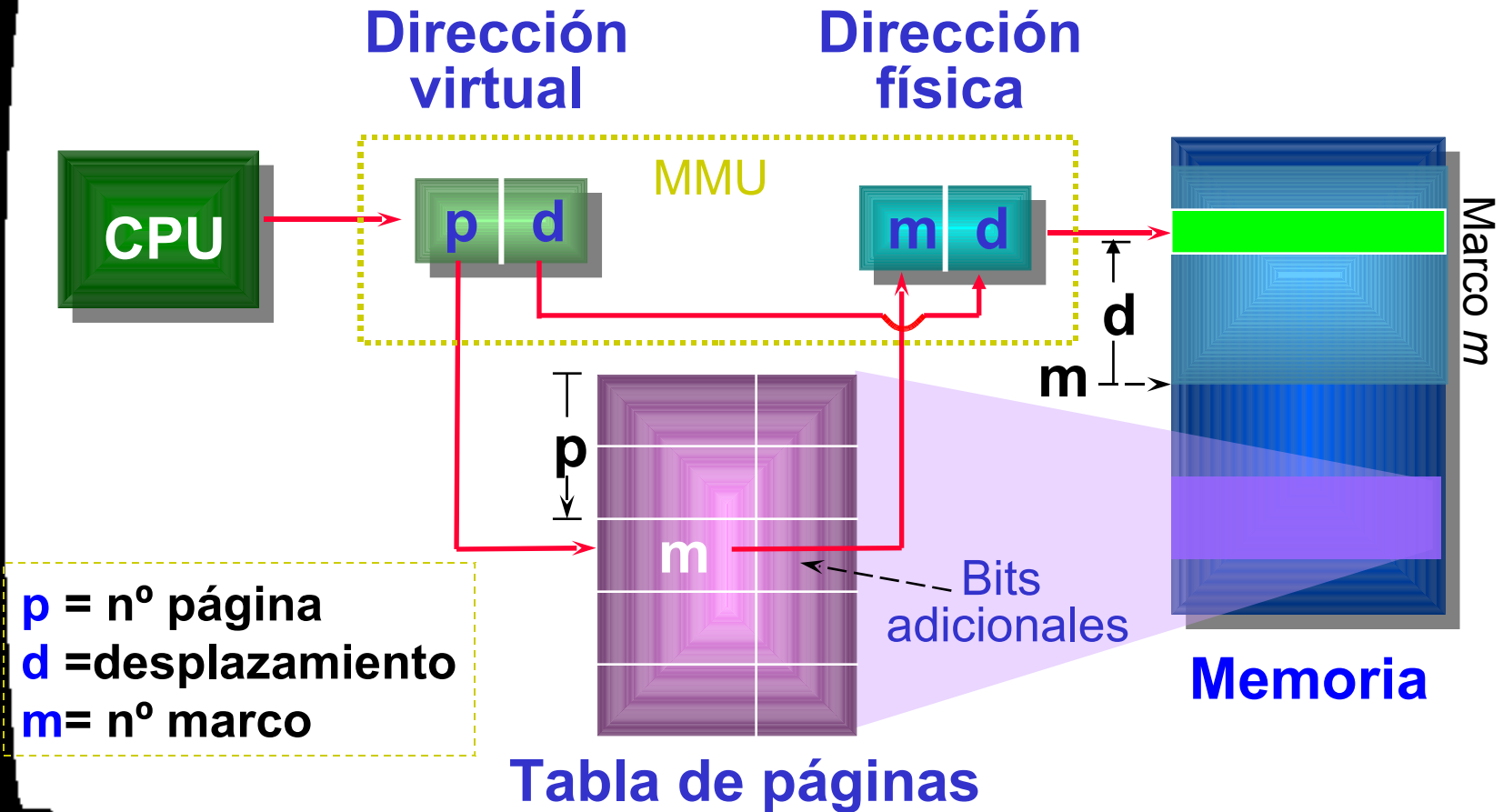


Dirección lógica

- La MMU descompone las direcciones lógicas que le pasa la CPU en dos campos:
 - Número de página (p) –se usa como índice para acceder a la p -ésima TPE, que nos dará la dirección de la base del marco donde se encuentra la página en memoria.
 - Desplazamiento de página (d) –sumado con la base de página define la dirección física que debe utilizar la MMU para acceder a la instrucción o datos.

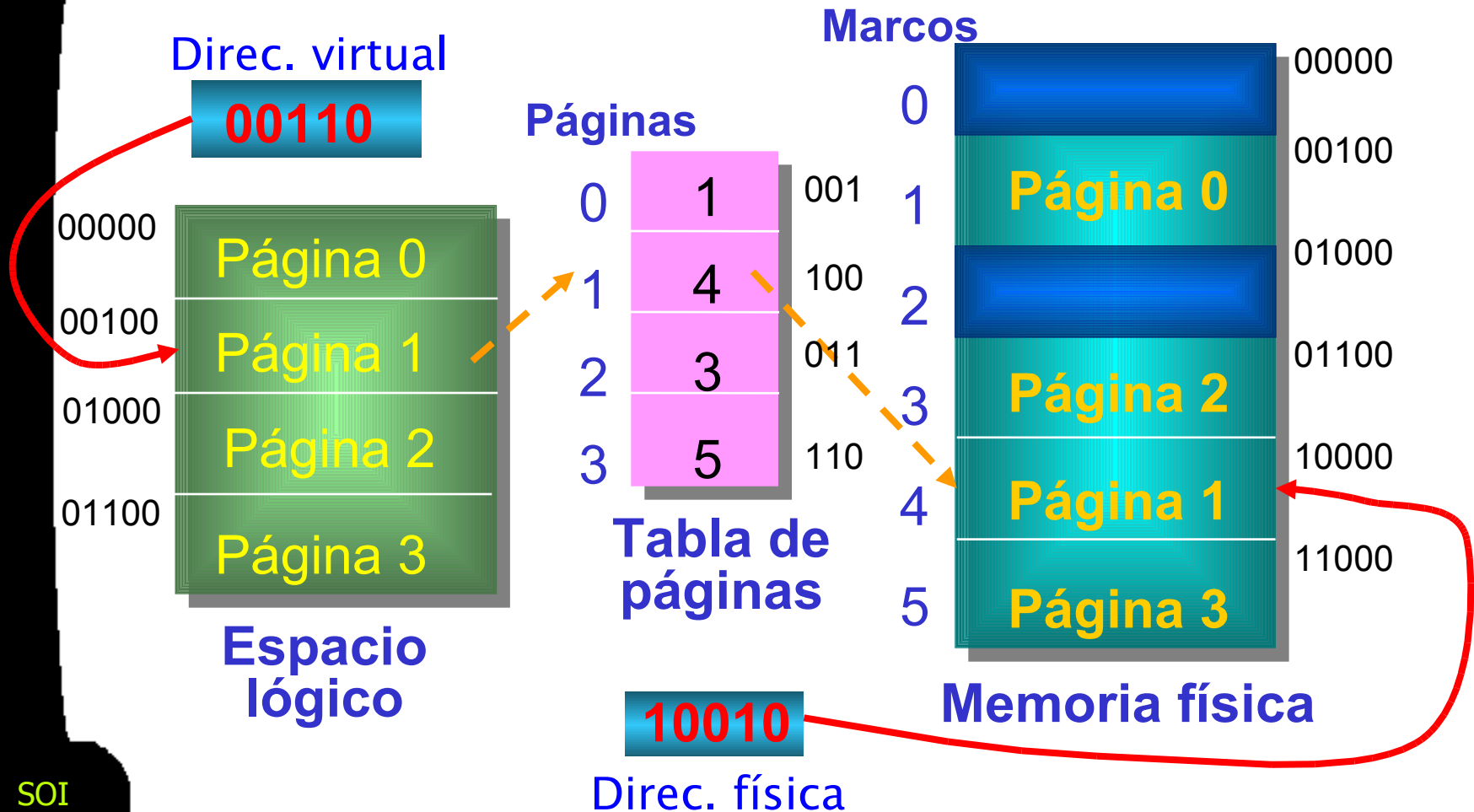


Esquema de traducción





Un ejemplo sencillo

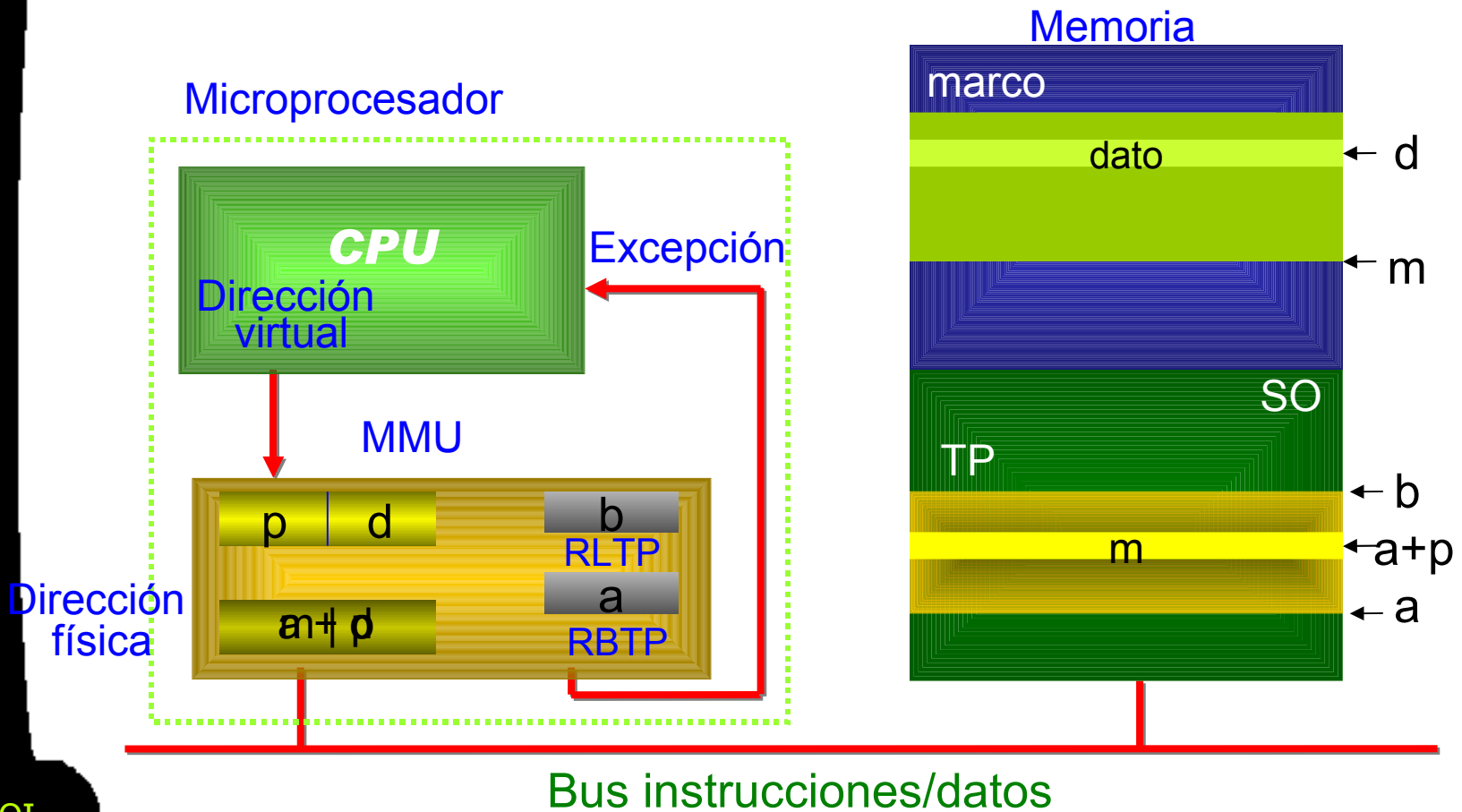




Implementación de la TP

- Por eficiencia, todas las TPs se mantienen siempre en memoria principal.
- Para localizar la TP de un proceso es necesario que la MMU disponga de:
 - **Registro base de la tabla de páginas (PTBR)** apunta a la base de la TP.
 - **Registro longitud de la tabla de páginas (PTLR)** indica el tamaño de la TP.
- Al asignar la CPU a P_i , cargamos en estos registros los valores $PTBR_i$ y $PTLR_i$ de su TP.

Acceso a memoria





Búfer de reconocimiento de traducciones

- Cada acceso a una instrucción/dato requiere dos accesos a memoria: uno a la TP y otro a la instrucción/dato.
- Para reducir el tiempo de acceso se introduce una caché hardware rápida denominada **Búfer de reconocimiento de traducciones** (*TLB* – *Translation Look-aside Buffer*).
- La traducción ahora se realiza:
 - ① Si existe ya en el TLB, tenemos el marco.
 - ② Si no, la buscamos en la TP y ajustamos el TLB.

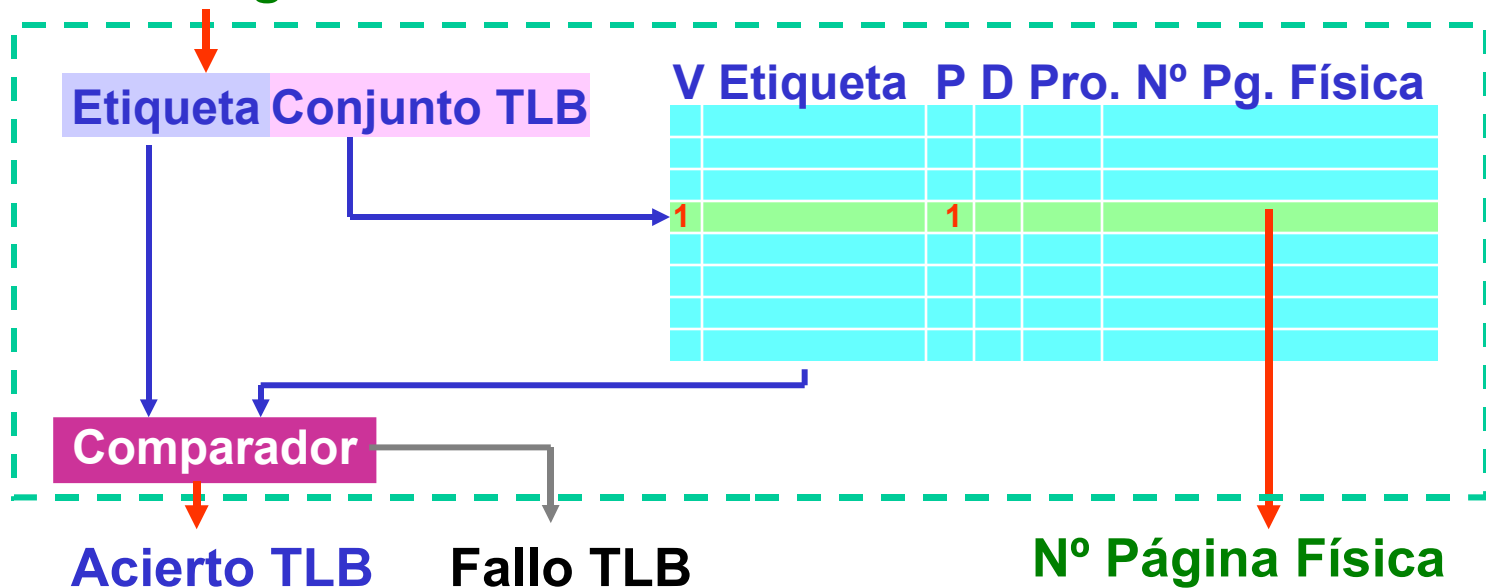


TLB: implementación

- Se implementa como un conjunto de *registros asociativos*, que permiten búsquedas en paralelo de una dirección (+ en EC II).

Número Página virtual

TLB



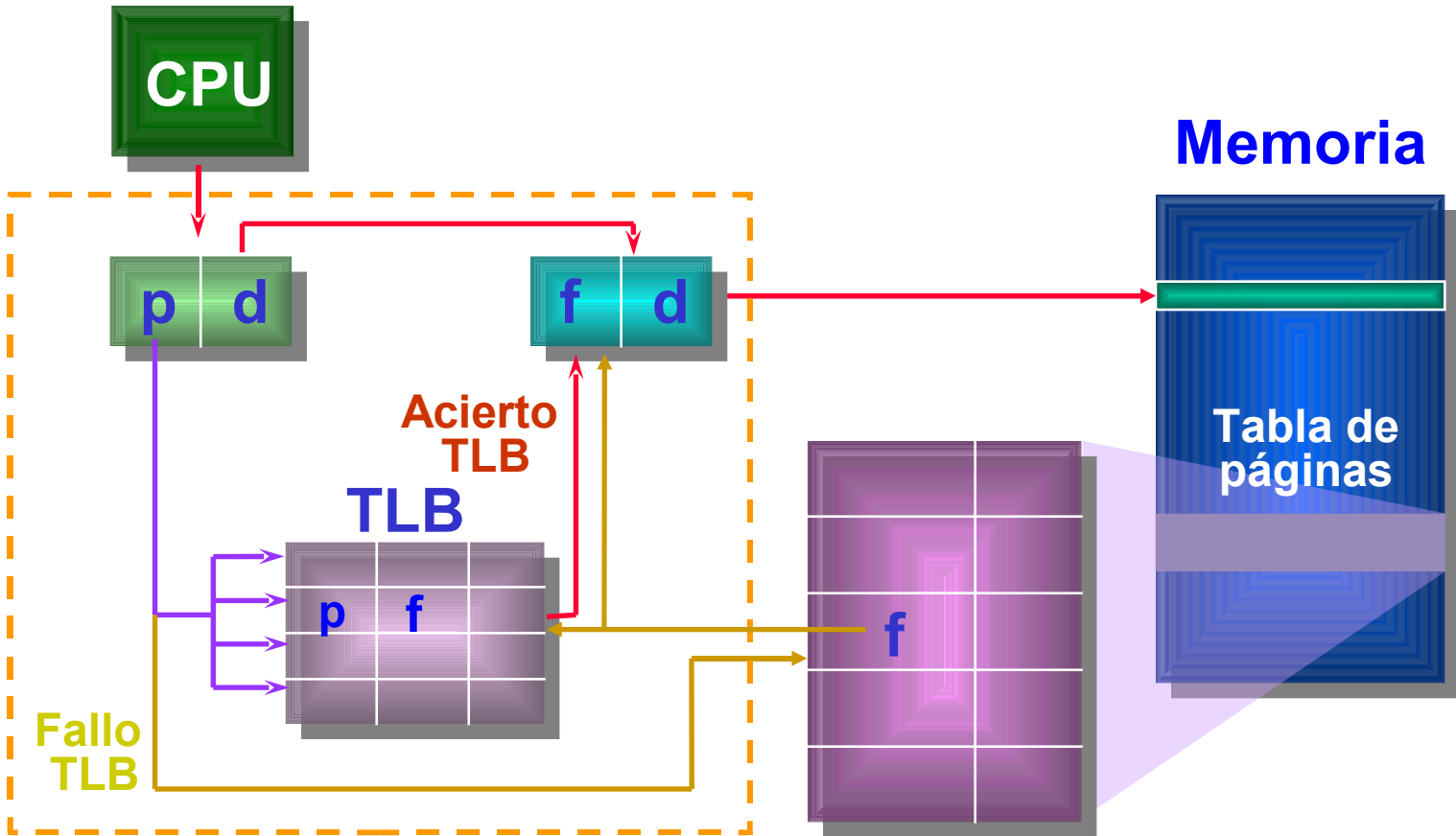


Características del TLB

- El TLB es una cache organizada típicamente como un conjunto asociativo total de n -vías.
- Tiene una Entrada de la TP (PTE) por línea.
- Tiene entre 32 y 1K de entradas.
- El tiempo de acierto esta diseñado para estar dentro de un ciclo de reloj.
- Tiene una tasa de fallos típica de entre 0.01 y 1%.
- El TLB es una tabla de sistema \Rightarrow cuando se planifica un proceso se invalidan todas las entradas de la tabla.



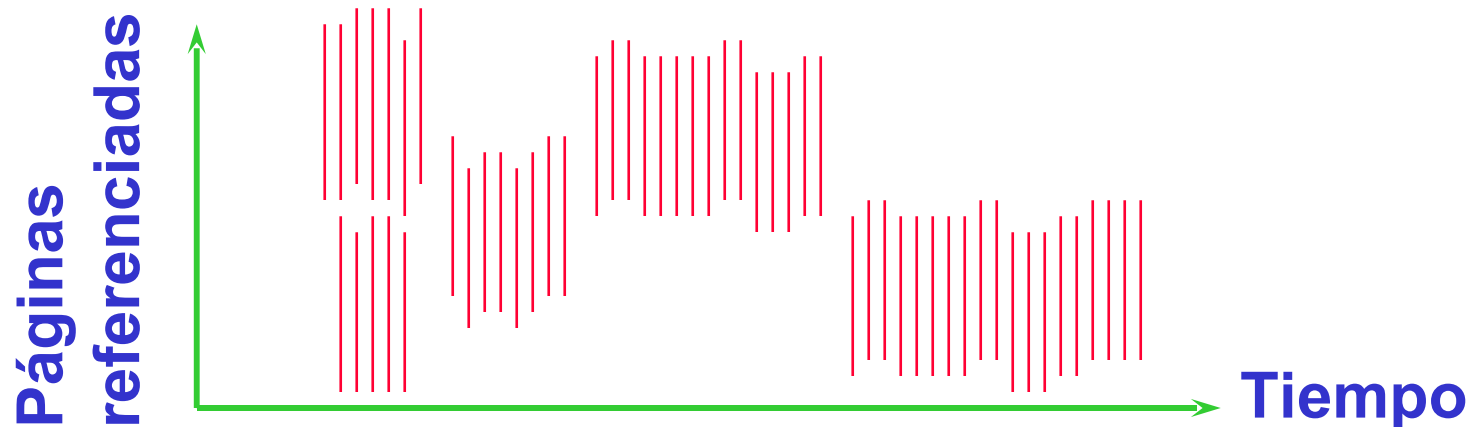
Esquema de traducción





Localidad de referencia

- Las cachés funcionan con un alto % de aciertos ya que los programas suelen exhibir **localidad de referencia** = un programa hace referencias locales, espaciales o temporales, a posiciones de su espacio de direcciones.





Tiempo de Acceso Efectivo

- Sea τ el tiempo de ciclo de memoria, ε el de una consulta asociativa de y α la tasa de aciertos del TLB, entonces:

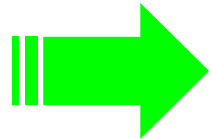
$$TAE = \alpha(\tau + \varepsilon) + (1 - \alpha)(2\tau + \varepsilon)$$

*Despreciable
Para valores altos de α*

$$\tau = 100\text{ns}$$

$$\varepsilon = 20\text{ns}$$

$$\alpha = 98\%$$



$$TAE = 122\text{ns}$$

¡Sólo 22% penalización!

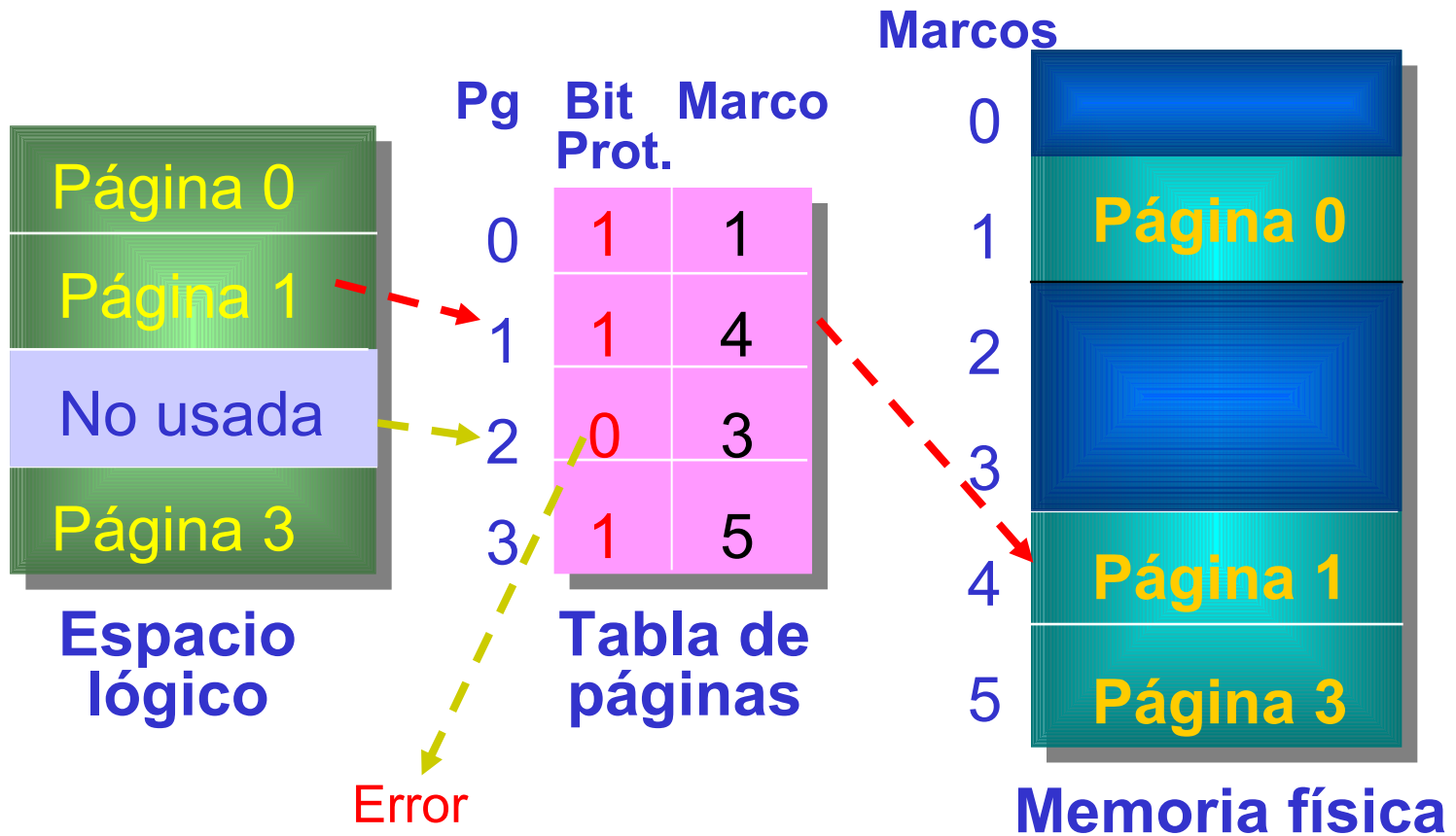


Protección de memoria

- Implementamos protección asociando un bit de protección con cada página.
- Cada entrada de la TP contiene un **bit de validez o de presencia**:
 - **Bit_válido=1**, la página asociada está en el espacio de direcciones lógicas del proceso, y por tanto es una página legal.
 - **Bit_válido=0**, la página no pertenece al espacio de direcciones del proceso.



Ejemplo





Tamaño de la TP

- Sea un sistema con:
 - Dirección virtual: 32 bits.
 - Tamaño de página = 4 Kbytes (2^{12} bytes).
 - Para el desplazamiento necesitamos 12 bits.
 - Para el n° página $(32-12) = 20$ bits.
 - N° posible de páginas virtuales = $2^{20} = \mathbf{j1,048,576 !}$
- Solución para reducir el tamaño en memoria de la TP: **paginación multinivel.**



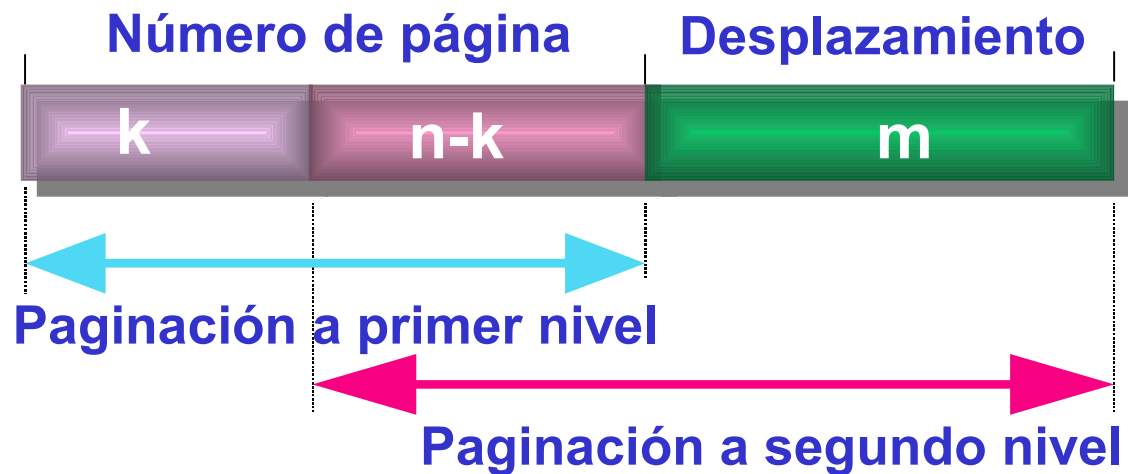
Paginación multinivel

- “Paginamos las tablas de páginas”.
- La partición de las TPs permite al SO:
 - Dejar particiones no usadas sin cargar hasta que el proceso las necesita.
 - Las porciones del espacio de direcciones que no se usan no necesitan tener TPs.
- Las arquitecturas de 32/64 bits utilizan varios niveles de paginación, p. ej. 2 niveles en Intel, 3 niveles en SPARC y 4 niveles en Motorola 68030.

Paginación a 2 niveles

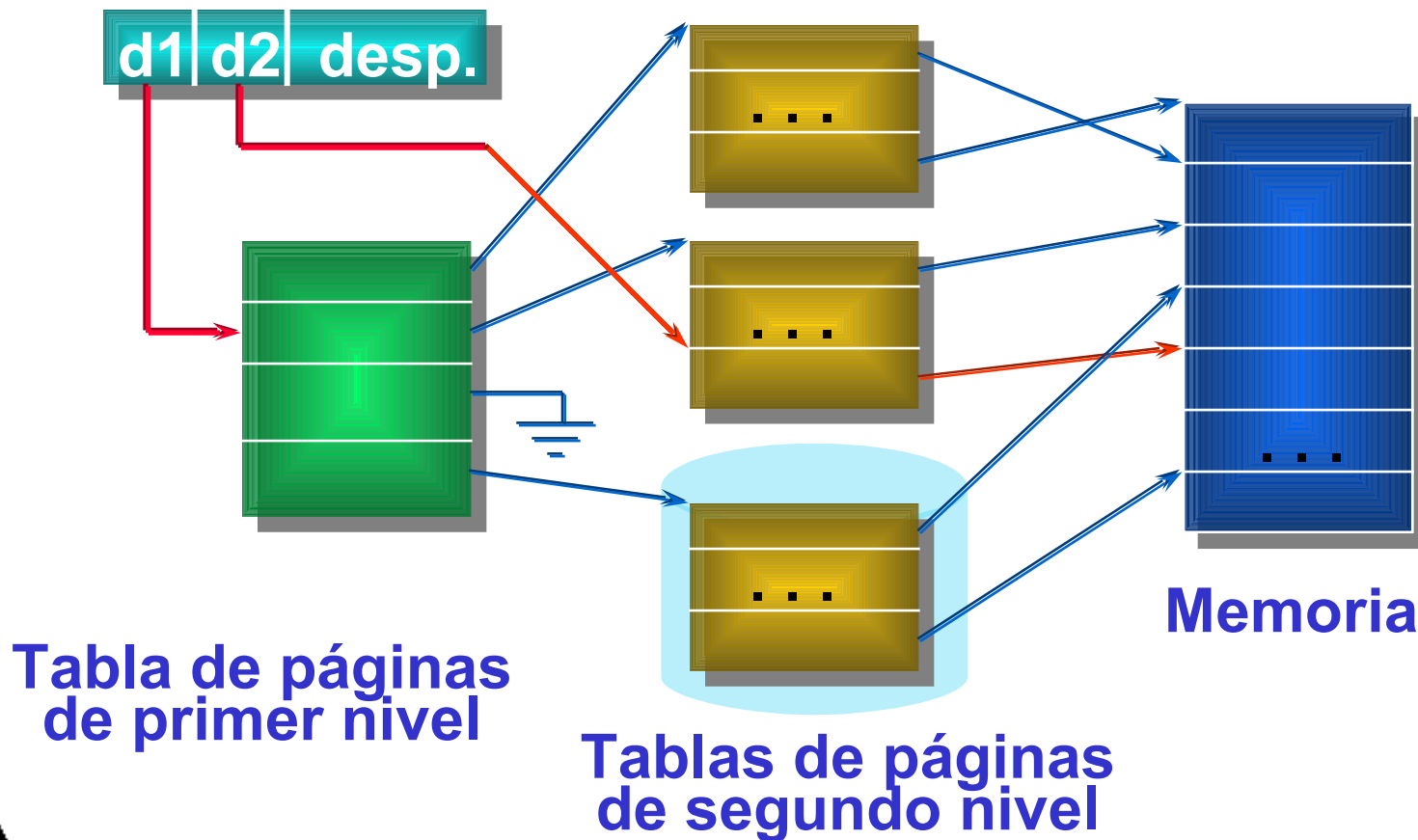
- La dirección lógica se divide de la forma:

Dirección virtual de n bits





Paginación a dos niveles (cont.)





Rendimiento

- Cada nivel se almacena en una tabla separada en memoria \Rightarrow convertir una dirección puede llevar hasta **tres accesos a memoria** en paginación a dos niveles.
- Aunque los accesos se triplican, las cachés permiten que el TAE sea razonable.
- Con los datos del ejemplo anterior:
$$\text{TAE} = 0.98 * 120 + 0.02 * 320 = 124 \text{ nsg.}$$
esto es, sólo un 24% de penalización.

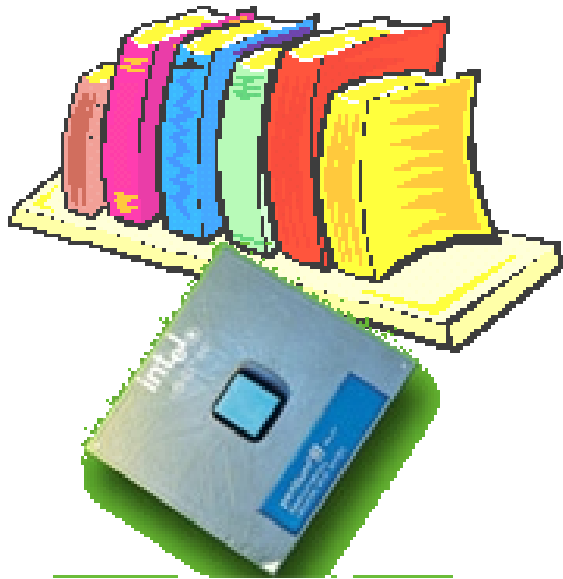


Compartir código

- La paginación facilita la compartición de código reentrante entre procesos.
- Si varios procesos están ejecutando el mismo programa sólo necesitamos tener una copia en memoria de las páginas de código. La compartición se implementa ajustando las correspondientes PTEs de la TPs de los procesos para que apunte al mismo marco.
- Problema: páginas con código y datos.



Segmentación

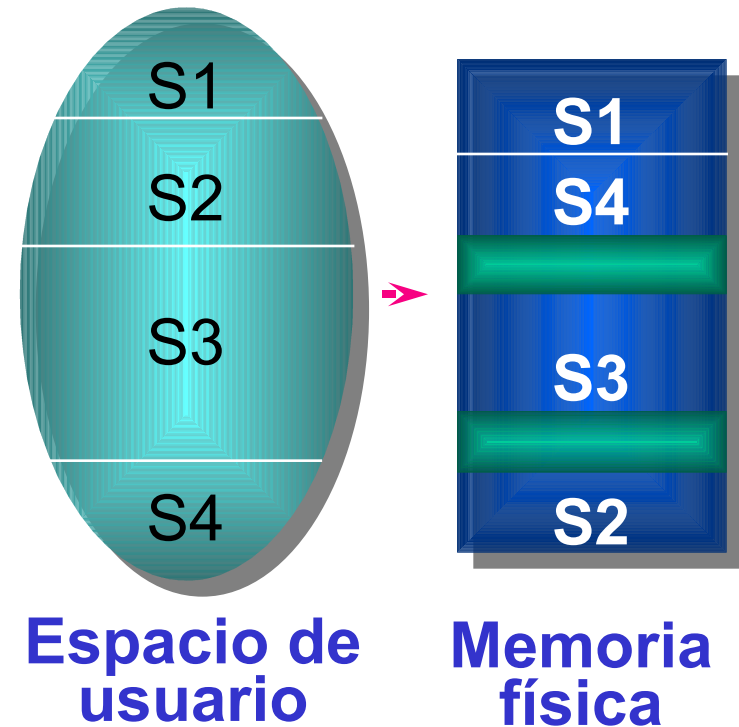


- Definición
- Traducción de direcciones
- Características
- Segmentación paginada



Segmentación

- Soporta mejor la visión de memoria del usuario: un programa es una colección de unidades lógicas – *segmentos*: procedimientos, funciones, pila, tabla de símbolos, matrices, etc.



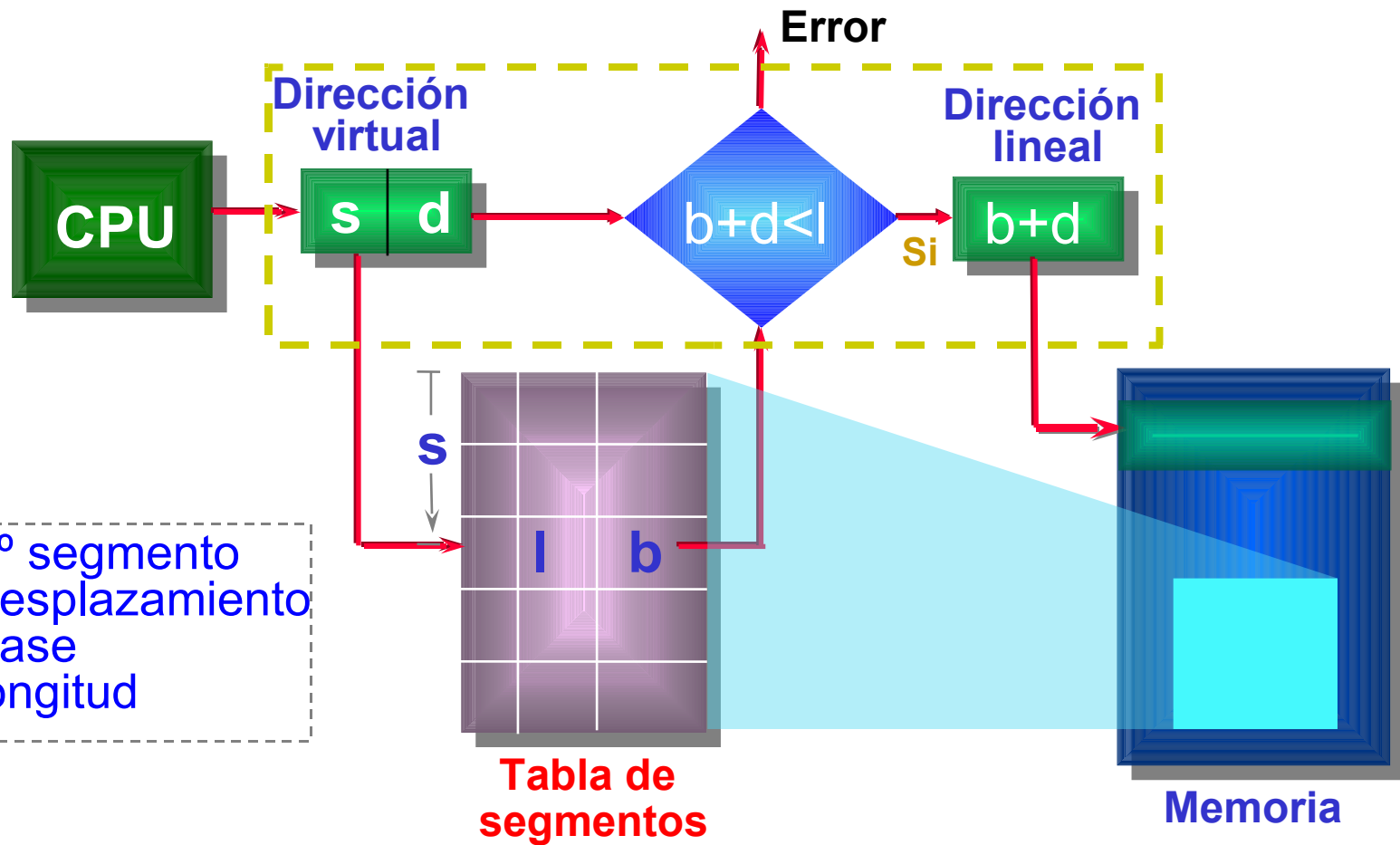


Segmentación (cont.)

- 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:
 - *base* – dirección física donde reside el inicio del segmento en memoria.
 - *límite* – longitud del segmento.



Traducción en segmentación





Implementación de la tabla de segmentos

- Para acceder a la TS del sistema o de un proceso necesitamos los registros:
 - **Registro Base de la Tabla de Segmentos** (*STBR*) apunta a la posición en memoria de la tabla de segmentos.
 - **Registro Longitud de la Tabla de Segmentos** (*STLR*) indica el número de segmentos usados por el programa; el número de segmento s es legal si $s < STLR$.



Características

- Es un esquema eficiente para espacios de direcciones dispersos.
- Reubicación: como los segmentos varían en longitud, la asignación de memoria es un problema de asignación dinámica de almacenamiento: primero/mejor encaje.
- Presenta fragmentación externa. La simulación da un 10% de memoria no usable
- Compartición: segmentos compartidos, con el mismo número de segmento.



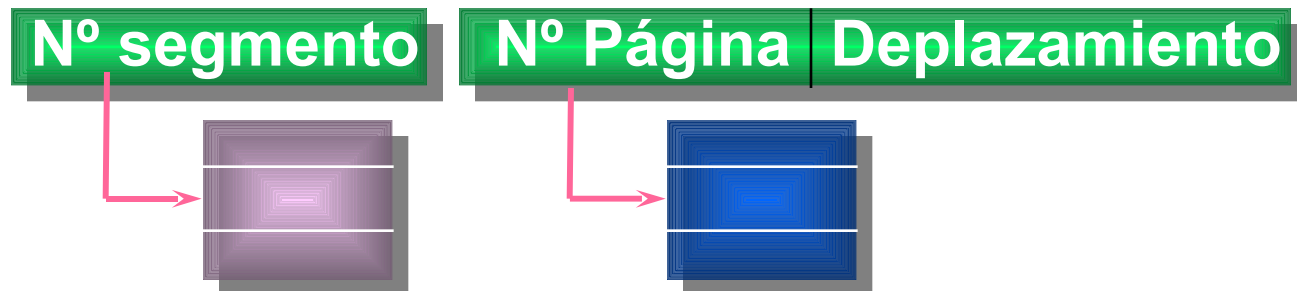
Características (cont.)

- Protección: cada entrada de la TS tiene asociado un bit de validez (0 si es ilegal; 1 si es segmento legal).
- Dado que los bits de protección se asocian con el segmento; la compartición de código ocurre a nivel de segmento.
- A diferencia de la paginación, la segmentación no es transparente al programador, debe realizarse por el compilador o el usuario.



Segmentación paginada

- Trata de resolver los problemas de fragmentación externa e interna, y los elevados tiempos de búsqueda mediante la paginación de los segmentos.
- Un segmento es un espacio lineal de direcciones que puede ser paginado.



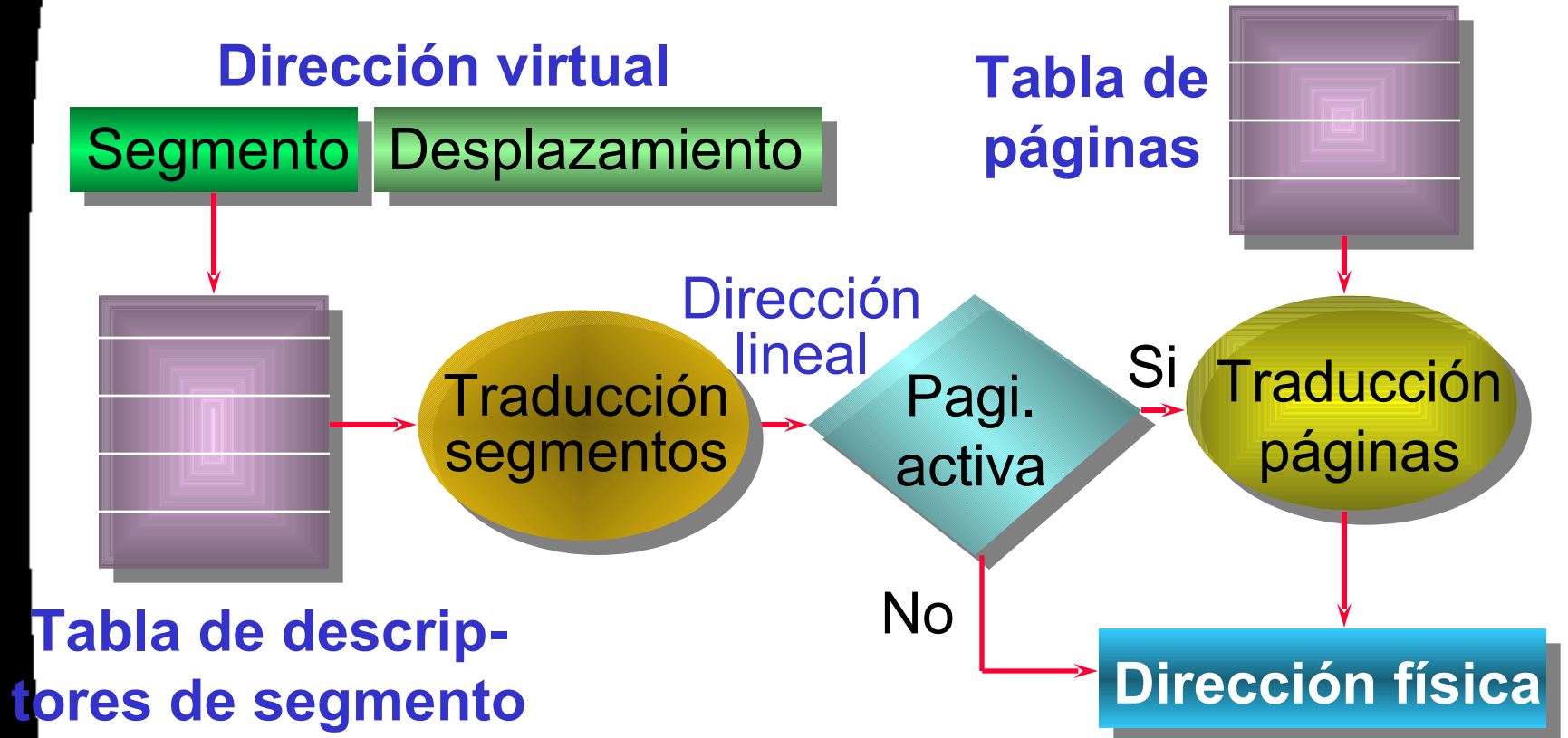


Ejemplo: Intel 80x86

- Utiliza segmentación con paginación a dos niveles. Tamaño de página de 4 kB ó 4 MB.
- Un proceso puede contener hasta 8192 segmentos. Cada proceso tiene su propia **tabla local de descriptores (LDT)**, con una entrada por cada segmento.
- Existe una **tabla global de descriptores (GDT)** que tiene entradas para el código, datos y pila del kernel y algunos objetos especiales como las LDTs por proceso.

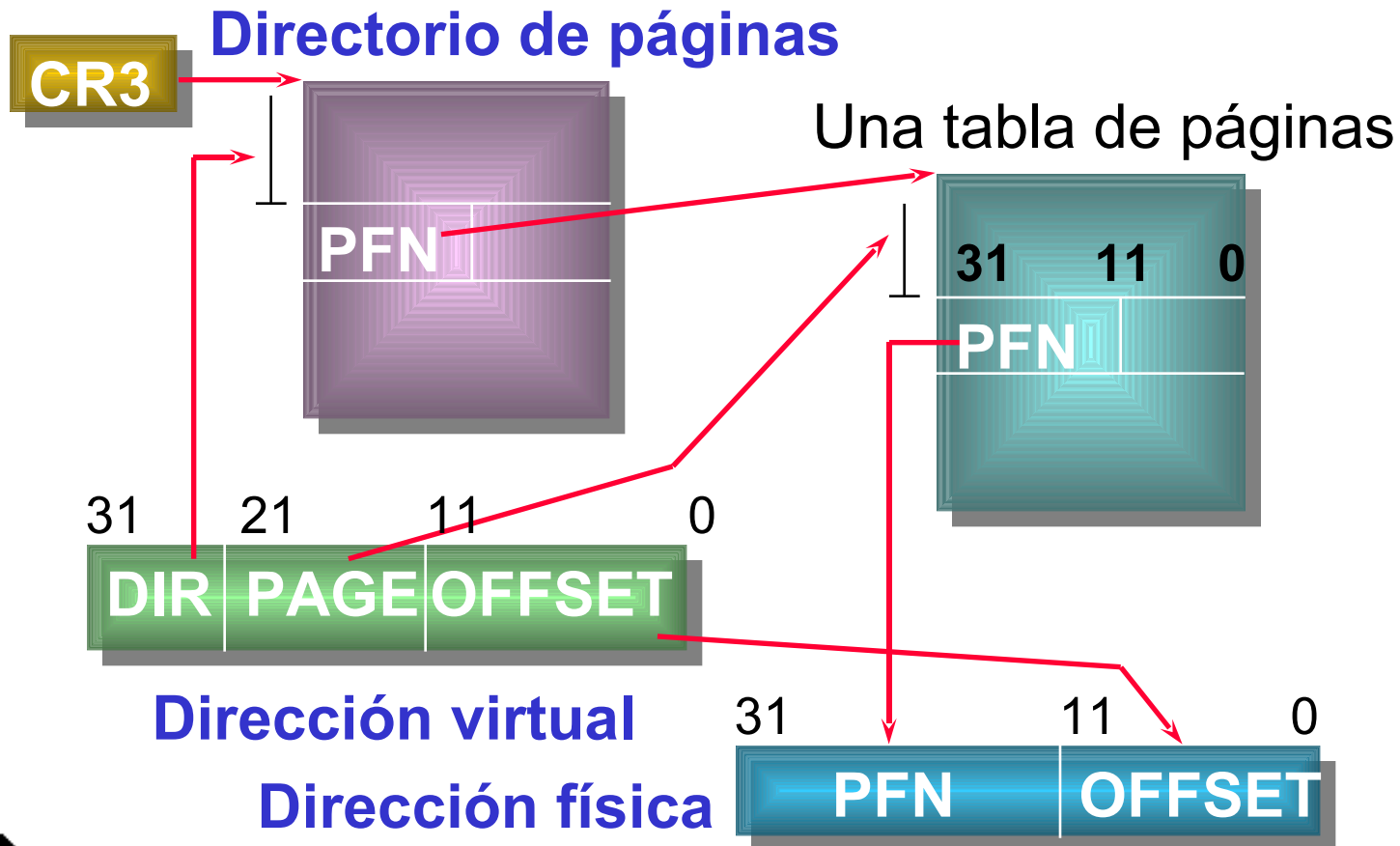


Intel 80x86



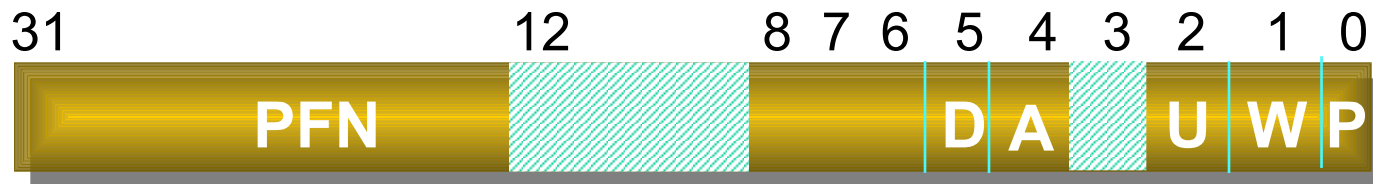



Paginación





PTEs en el Intel x86



PFN	Número de marco de página
D	Sucio
A	Accedida (referenciada)
U	Usuario (0) /supervisor (1)
W	Lectura (0) /escritura (1)
P	Presente (válido)
	Sin usar



Memoria Virtual



- Paginación por demanda
- Falta de página
- Sustitución de páginas
- Conjunto activo (de trabajo)



Memoria virtual es ...

- Gestión automática del almacenamiento, que determina: qué almacenar en memoria principal, dónde almacenarlo y cuando cargarlo/descargarlo.
- MV simplifica la programación:
 - Elimina la necesidad de gestión manual.
 - Maneja las ligaduras.
 - Gestiona la compartición de memoria.
 - Abstrae los detalles de memoria física.
- MV es el gestor de la memoria principal



MV como una abstracción

- La MV abstrae los detalles de memoria:
 - Memoria como una matriz infinita.
 - Continuidad lógica de los datos
 - Aislamiento de datos.
- MV como un tipo abstracto de datos:
 - Crea/destruye espacio de direcciones.
 - Copia/saca datos en/del espacio direc.
- Estas operaciones no son típicas de la API, sino internas al SO.




Memoria Virtual

- La separación de espacios permite:
 - Ejecutar un programa sin estar cargado completamente en memoria, y/o
 - Espacios de direc. lógicas mayores que el espacio de direcciones físicas.
- Necesitamos asignar páginas para intercambiarlas de memoria ↔ disco.
- Podemos implementar la paginación por demanda o anticipada.
- MV suele suministrar la noción de “región” para trabajar con MMUs segmentadas.



Paginación por demanda



```
pushl %ebp
movl %esp,%ebp
pushl $0
pushl $.LC0
call open
addl $8,%esp
movl %eax,%eax
movl %eax,fd
```

```
.L1:
leave
ret
```

- Características
- Faltas de páginas
- Sustitución de páginas
- Rendimiento



Características

- Sólo traeremos una página a memoria cuando se necesite. Las ventajas son:
 - menos E/S.
 - menos memoria.
 - respuesta más rápida.
 - más usuarios.
- Problema: el rendimiento
- ¿cuándo se necesita? cuando se referencia
 - Referencia es inválida \Rightarrow se aborta.
 - Si es válida \Rightarrow se trae pág. a memoria.



Bit de validez

- Sobrecargamos la semántica del **bit de validez** de cada PTE para implementar la paginación por demanda. Si el bit vale:
 - 1 \Rightarrow la página está en memoria.
 - 0 \Rightarrow "no está en memoria"(no es válida).
- Al inicio, el bit está a cero en todas las PTE's. Durante la traducción de direcciones, si se referencia una página con el bit de validez a cero se produce una trampa **falta de página**.

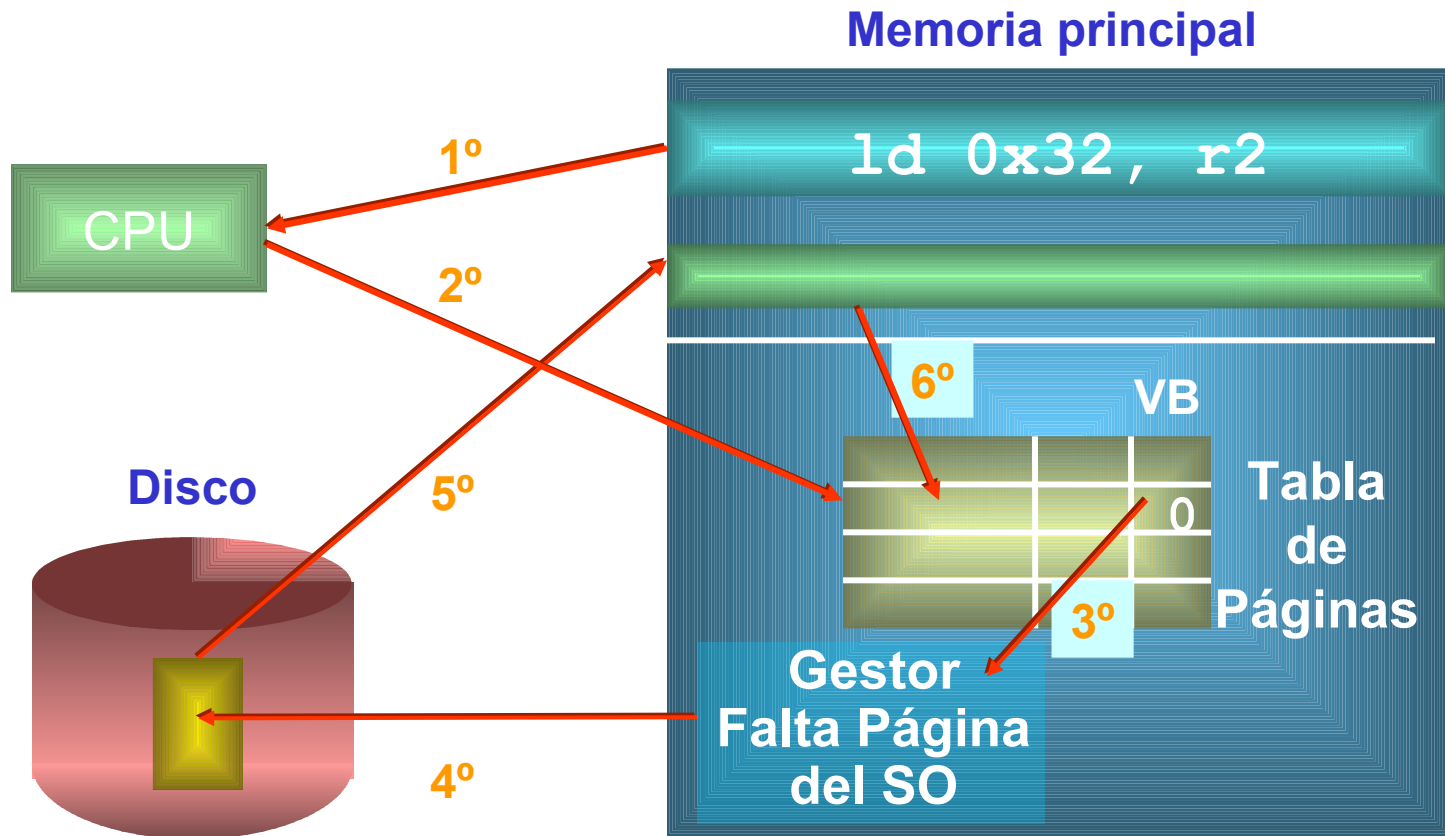


Falta de página

- En una falta de página, el SO mira en la tabla que mantiene el espacio de direc. completo del proceso para decidir si:
 - La referencia es inválida → aborta proceso
 - La página no está en memoria, entonces:
 - ① Obtiene un marco vacío.
 - ② Carga la página dentro del marco.
 - ③ Pone el bit de validez a 1.
 - ④ Rearranca la instrucción.
 - Si la página esta en memoria pero la traducción no es válida ⇒ reasigna la pág.



Esquema de la gestión de falta de página





Otros usos del bits de validez

- En ocasiones puede que una página este en memoria pero el SO haya desactivado el bit de validez.
- Esto le permite al SO:
 - Simular en soft el **bit de referencia**.
 - Implementar la **copia-sobre-escritura** (*copy-on-write*) – mecanismo que para ahorrar memoria permite compartir páginas y duplicarlas cuando se intenta escribir en en ellas.



¿si no hay marcos libres? ... Sustitución de páginas

- Reducimos la sobreasignación de memoria si la rutina de servicio de falta de página incluye la **sustitución de páginas**, es decir, encontrar página que no este realmente en uso para sacarla, y poder utilizar su marco.
- La sustitución de páginas completa la separación entre la memoria lógica y física.
- Rendimiento: el algoritmo de sustitución debería producir el mínimo número de faltas de páginas.



Rendimiento de la paginación por demanda

- Sea p la probabilidad de faltas de página. $p=0$, no hay faltas de páginas; $p=1$, toda referencia es una falta. Por tanto, $0 \leq p \leq 1$.

$$\begin{aligned} \text{TAE} = & (1 - p) * \text{acceso_a_memoria} \\ & + p * (\text{sobrecarga_falta_de_página} \\ & \quad + \text{sacar_fuera_la_página} \\ & \quad + \text{traer_la_página} \\ & \quad + \text{sobrecarga_de_rearranque}) \end{aligned}$$

- $t_{\text{servicio-falta-pág.}} = 25 \text{ msg}$ y $\text{TAE} = 100\text{ns}$, un 10% de degradación sólo si 1 / 2,5 millones de accesos a memoria producen falta de página.



Bit sucio

- El uso de un bit adicional en las PTEs, el **bit sucio** (o *de modificación*) reduce la sobrecarga de transferencia de algunas páginas.
- Sólo se escriben en disco las páginas que han sido modificadas; las no modificadas pueden descartarse ya que tenemos una copia de ellas en el archivo ejecutable en disco.

Algoritmos de sustitución



- Definiciones
- Algoritmo óptimo
- “ LRU
 - Aproximación: el algoritmo del reloj
- Asignación de marcos



Definiciones

- Buscamos un algoritmo de sustitución que de la menor tasa de faltas de páginas.
- Denominamos **cadena de referencia**, $\omega = r_1, r_2, r_3, \dots, r_i, \dots$, la secuencia de números de las páginas referenciadas por un proceso durante su ejecución.
- Denominamos **tiempo virtual** al índice de la cadena de referencia de páginas. En entornos multiprogramados, éste nos da idea del progreso del proceso a lo largo de su ejecución.



Estado de la memoria

- El conjunto de páginas cargadas en m marcos en tiempo virtual t , $S_t(m)$, viene dado por

$$S_t(m) = S_{t-1}(m) \cup X_t - Y_t$$

donde X_t es la página traída en t , e Y_t la página sustituida en t .

- Así, dando Y_t , identificamos unívocamente la estrategia de sustitución ($S_{t-1}(m)$ y X_t están definidos cuando ocurre la falta de página).



Algoritmo óptimo

- Sea $FDW_t(r)$, la *distancia adelante* de la página r en t , la distancia desde el punto actual de la cadena de referencia a la siguiente posición en ω donde la misma página es referenciada de nuevo.
- El **algoritmo óptimo** sustituye la página Y_t con máxima distancia adelante:

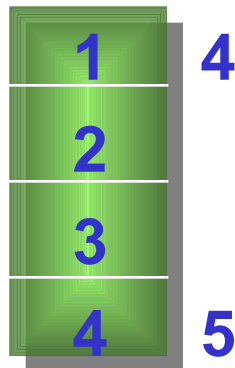
$$Y_t = \max_{x \in St-1(m)} (FWD_t(r))$$

- Si se pueden cargar más de una página que no aparecen en ω , entonces se elige una arbitraria.



Ejemplo de Alg. Optimo

□ Sean 4 marcos y
 $\omega = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$



⇒ 6 faltas de página

- Problema: debemos tener un “conocimiento perfecto” de la cadena de referencia.
- Aunque no es implementable, se utiliza para medir cómo de bien se comportan otros algoritmos.




Algoritmo LRU

- Sea $BKWD_t(r)$, **distancia atrás** de la página r en t , la distancia de r a la última ocurrencia de la página en la parte precedente de ω .
- **LRU** (*Least Recently Used*) selecciona la página con máxima distancia atrás:

$$Y_t = \max_{x \in St-1(m)} (BKWD_t(r))$$

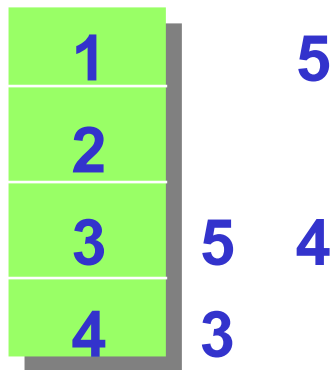
si hay más de una, se elige una arbitraria.



Debido a la localidad, la distancia atrás es un buen estimador de la distancia adelante.

Ejemplo: Algoritmo LRU

- 4 marcos y la ω anterior:



⇒ 8 faltas de página

- Implementar LRU requiere contabilizar el uso de cada página.
- Dos formas:
 - Con contador.
 - Con pila.
- Ambas son costosas, por lo que se utilizan aproximaciones como el algoritmo del reloj.



LRU : implementaciones

LRU con contador

- Añadimos a cada PTE un contador.
- Al referenciar una página, se copia el tiempo del reloj en el contador.
- Si necesitamos cambiar una página, se elige aquella cuyo contador sea menor menor.

LRU con pila

- Los números de páginas se mantienen en una pila (lista doble enlazada).
- Al referenciar una pág., esta se mueve a la cima (ajustar 6 punteros).
- Sustituimos la página que hay en el fondo de la pila; no hay que hacer búsquedas.

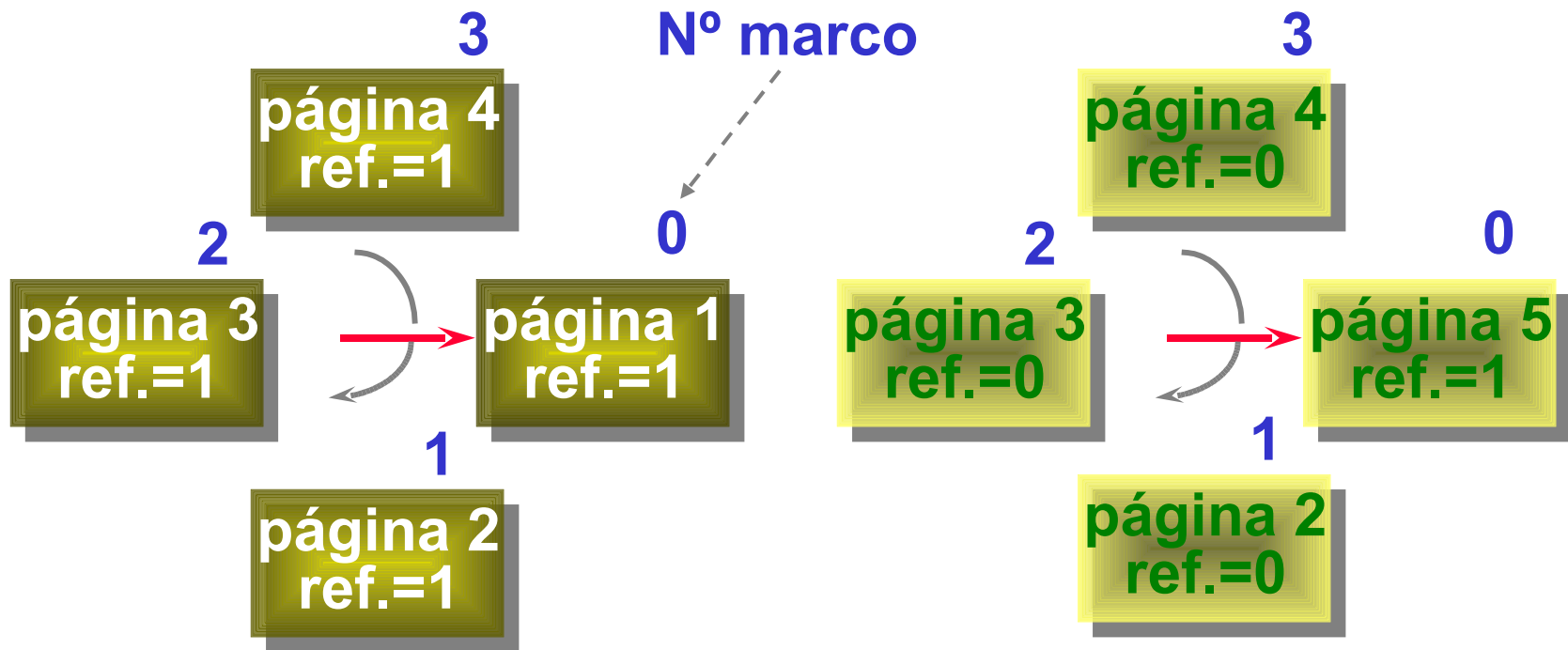


Algoritmo del reloj

- Necesitamos:
 - 1 bit de referencia (inicialmente a 0).
 - Marcos candidatos en lista circular
 - Un puntero (la manecilla del reloj).
- A partir de la posición actual de la manecilla:
 - Si la pg. tiene bit=1, se pone a 0 (se le da una segunda oportunidad).
 - Se avanza la manecilla, y seguimos en 1.
 - Se sustituye la primera que encontramos con el bit=0. Se avanza la manecilla.



Algoritmo del reloj: ejemplo





Asignación de marcos

- Cada proceso necesita para su ejecución un número mínimo de páginas.
- Criterios de asignación de marcos:
 - Según el número de marcos asignados:
 - Fija – todos los procesos igual.
 - Por prioridad – n° marcos proporcional a la prioridad–
 - Según a quién pertenecen los marcos:
 - Local – de entre los marcos asignados al proceso.
 - Global –de entre toda la memoria.



Modelo del Conjunto Activo

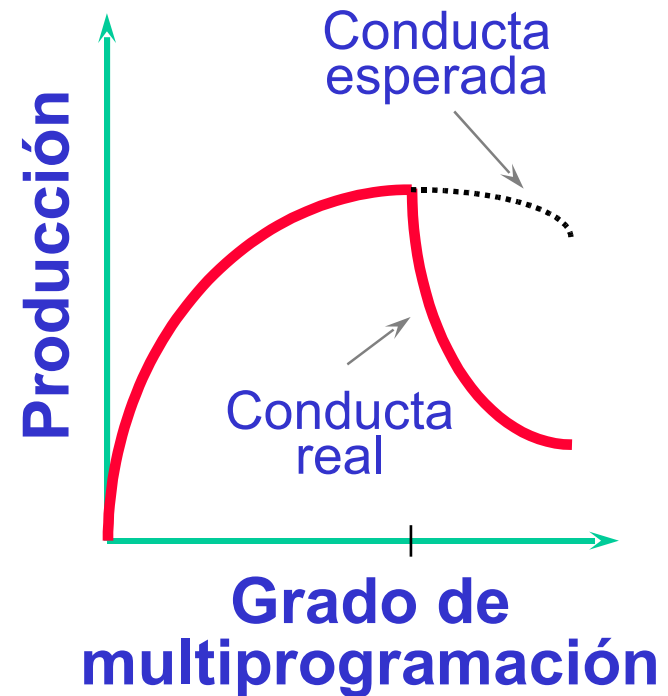


- Hiperpaginación
- Modelo del conjunto activo
- Aproximaciones al algoritmo del conjunto activo:
 - Algoritmo del reloj del conjunto activo
 - Frecuencia de faltas de páginas



Hiperpaginación

- Si un proceso no tiene “suficientes” páginas, la tasa de faltas es alta → baja uso de la CPU → el SO incrementa el grado de multiprogramación → más faltas de páginas → ...
- **Hiperpaginación** = sistema ocupado en traer/sacar páginas.





¿ A qué se debe ?

- Según el **modelo de localidad**:
 - Los procesos migran de una localidad a otra.
 - Las localidades pueden solaparse.
- ⚙ La hiperpaginación se produce cuando la suma de los tamaños de todas las localidades es mayor que el tamaño total de memoria.
- Se puede limitar con algoritmos de sustitución locales (o por prioridad).



Modelo del Conjunto Activo


- El **Conjunto Activo** (*Working Set*) con parámetro Δ , $WS(t, \Delta)$, es el conjunto de páginas que han sido referenciadas en las últimas Δ unidades de tiempo.
- $|WS(t, \Delta)|$ varía con el tiempo.
- Δ es el **tamaño de la ventana** y es un parámetro ajustable. Δ debe medirse en tiempo virtual.
 - Si $\Delta \downarrow$ no abarca la localidad completa.
 - $\Delta \uparrow$ abarca varias localidades.
 - $\Delta \rightarrow \infty$ abarca el programa entero.



Ejemplo

- Sea la cadena de referencias

$\omega = \dots 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 6 \ 6 \ 5 \ 2 \ 4 \ 4 \ 7 \ 7 \ 5 \ 5 \ \dots$



Δ t_1 Δ t_2

- Si el tamaño de la ventana es $\Delta=3$:
 - En t_1 , $WS(t_1, 3) = \{4, 5, 6\}$
 - En t_2 , $WS(t_2, 3) = \{4, 7\}$
- Si $\Delta=5$, los conjuntos activos serían:
 - En t_1 , $WS(t_1, 5) = \{2, 3, 4, 5, 6\}$
 - En t_2 , $WS(t_2, 5) = \{2, 4, 7\}$



Principio del WS

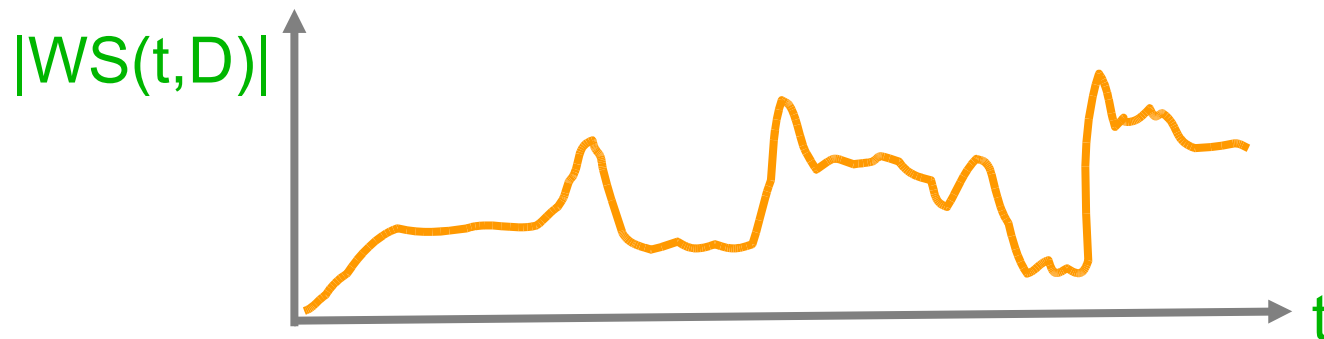
- La importancia del modelo del conjunto activo se debe a que liga la gestión de memoria y la planificación través del **Principio del Conjunto Activo**:

“Un proceso sólo puede ejecutarse si su conjunto activo está en memoria principal. Una página no puede retirarse de memoria principal si está dentro del conjunto activo del proceso en ejecución”.



Propiedades del WS

- $|WS(t, \Delta)|$ es variable, en concreto,
 $1 \leq |WS(t, \Delta)| \leq \min(\Delta, n)$
(n = número total de páginas). Esto implica que podemos evitar las particiones fijas.
- ¿Cuál es la forma de la gráfica de $|WS(t, \Delta)|$ frente a t ?





WS e hiperpaginación

- Sea $D = \sum |W s_i|$ = Total marcos demandados.
- Si $D > m$ (marcos totales) \Rightarrow hiperpaginación.
- Política: si $D > m$, entonces suspender un proceso (o sea liberar memoria).
- Posibles políticas de control de carga:
 - Proceso de menor prioridad.
 - El que genera más faltas de página.
 - El último proceso activado.
 - El proceso más pequeño, o el mayor.
 - Proceso con el resto de quantum más largo.



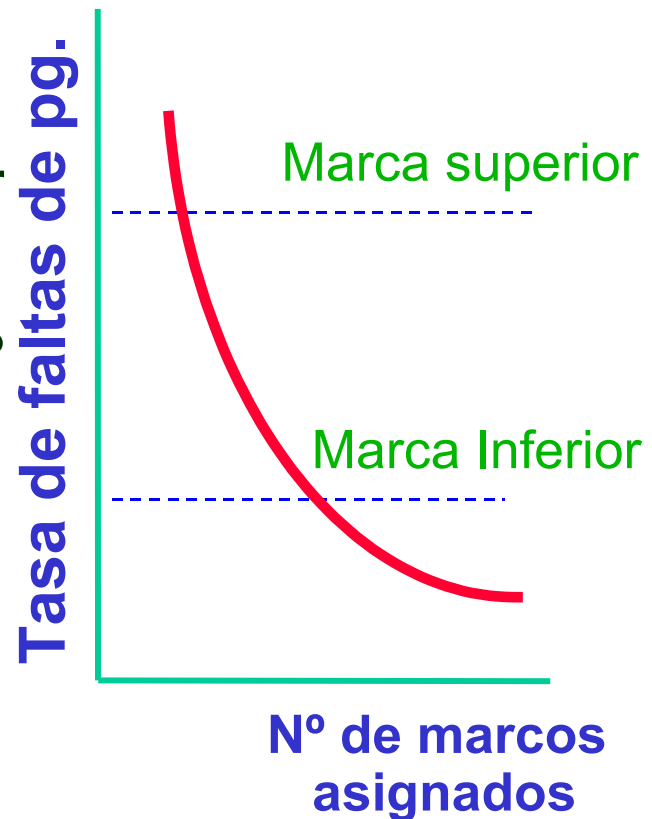
Algoritmo WSReloj

- Es más difícil de implementar que el LRU. WSReloj es una aproximación a WS que produce la misma tasa de faltas de págs.
- Igual al del reloj + 1 variable, **UltRef**, para cada marco. Al activar el bit de referencia, **UltRef[marco] := $T_{\text{proceso-}i}$** (tiempo virtual del proceso que usa el marco). Al producirse una falta actúa como el reloj, pero cuando encuentra un marco con bit de referencia a cero elimina la página del WS si $T_{\text{proceso-}i} - \text{UltRef}[m] > \Delta$.



Algoritmo Frecuencia de Faltas de Página

- PFF es una estimación del $|WS|$.
- Asignar/desasignar marcos para mantener la tasa de faltas de página dentro de unos límites (marcas superior e inferior).
- Tasa falta páginas = $1 / \text{tiempo entre faltas}$.





PFF en detalle

- El SO mantiene un bit de referencia y el tiempo virtual de la última referencia de la página. $Tasa\ falta\ pag. = 1 / t_{actual} - t_{ultref}$.
- Cuando se produce una falta de página, si:
 - $tfp < marca_inferior \Rightarrow$ añadir una página al conjunto activo.
 - $tfp > marca_superior \Rightarrow$ eliminar páginas con bit de referencia a cero, y poner a cero los bits de las restantes.
- ★ Problema: el algoritmo no se comporta bien en las transiciones entre localidades.