Tema 5:
Administración de Memoria

Grado en Ing. Informática

Curso 2024/25

Índice

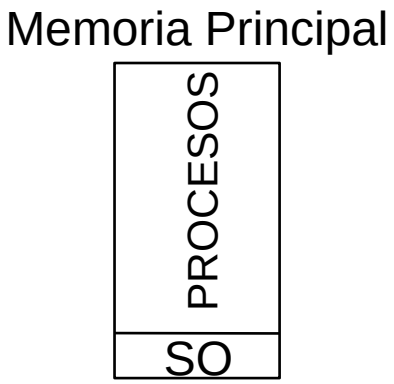
5.1 Introducción

- El programador quiere: memoria de uso exclusivo, tan grande y rápida como sea posible, no volátil → esto garantizaría gran rendimiento, lecturas y escrituras de muchos datos a gran velocidad, sin pérdida de datos ante caídas del sistema
  - Imposible de construir con tecnología actual a un coste razonable
- Solución: **Jerarquía de memoria**
  - Unos cuantos MB de memoria cache, unos cuantos GB de memoria principal, unos cuantos TB de disco duro
- Parte de esa jerarquía controlada por el hardware (gestión de caché) y otra por el SO (mem ppal y almacenam. secundario)
- El trabajo del SO es abstraer esa jerarquía y administrarla
  - A esa parte del SO se le llama **administrador de memoria**
- Su misión es administrar la memoria con eficiencia: controlar qué partes de la memoria RAM están en uso, asignar memoria RAM a los procesos cuando la necesiten y liberarla si no se necesita, mover información entre disco y RAM…

Índice

5.2 Administración de memoria sin memoria virtual

- Primeros esquemas de administración de memoria no usaban técnicas propias de la memoria virtual:
  - El SO ocupaba una posición fija en la memoria física y el resto de la memoria disponible para los procesos
  - A cada proceso se le asignaba una zona contigua de memoria principal



- Vamos a estudiar dos alternativas:
  - 5.2.1 Multiprogramación con particiones fijas
  - 5.2.2 Multiprogramación con particiones variables

5.2.1 Multiprogramación con particiones fijas

- Una primera solución es dividirla en N particiones de igual o distinto tamaño con límites fijos → **particiones fijas**

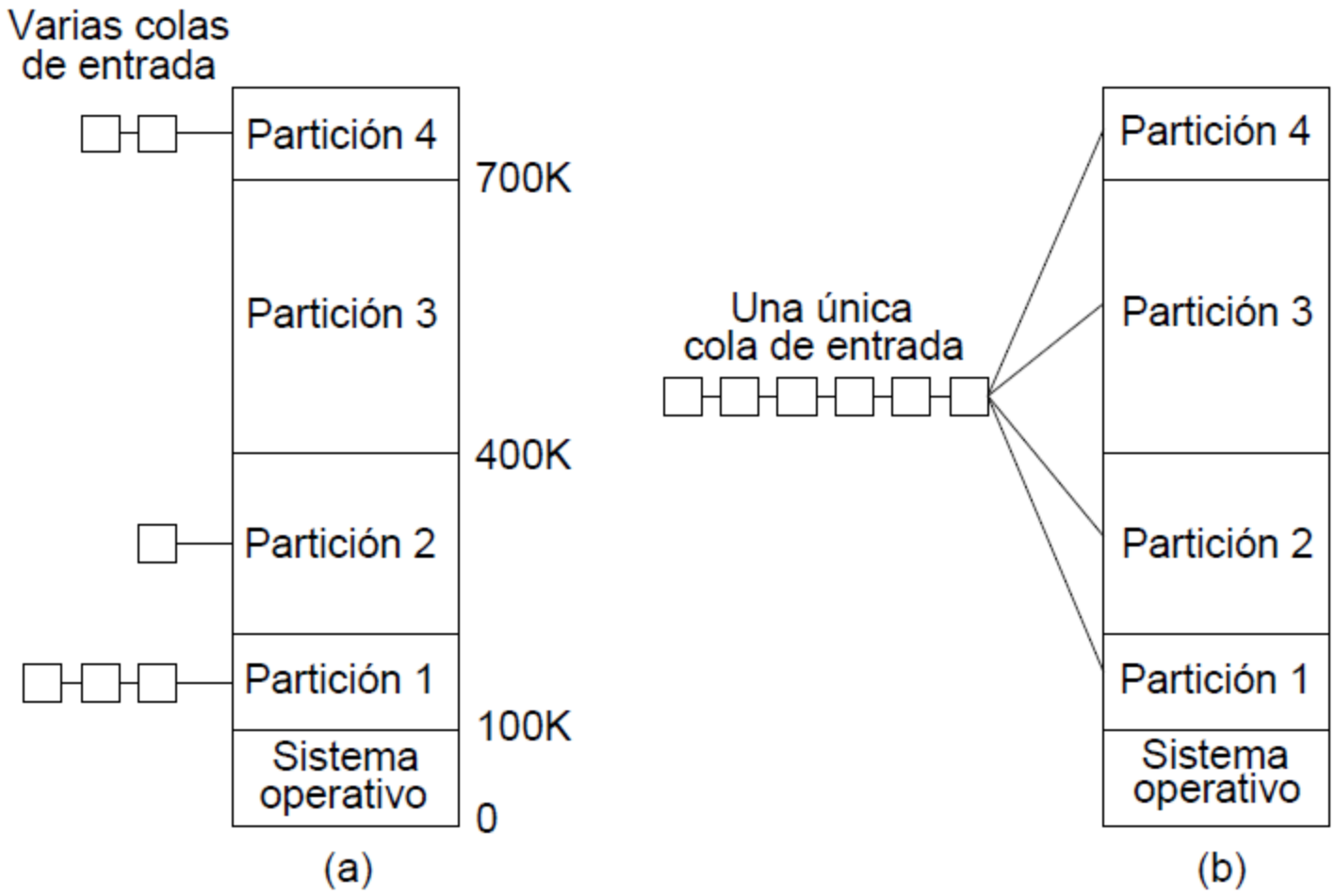
1. **Particiones de igual tamaño:**
   - Cualquier proceso de tamaño menor o igual que la partición puede cargarse en cualquier partición libre
   - Los procesos bloqueados pueden suspenderse para obtener nuevas particiones libres
   - El tamaño de las particiones condiciona el tamaño de los procesos
   - **Fragmentación interna**: parte de la memoria de la partición se desperdicia haciendo que el uso de la memoria sea ineficiente

5.2.1 Multiprogramación con particiones fijas

2. Particiones de distinto tamaño:

- Mejoran la eficiencia
- Necesitamos algoritmo de ubicación → ¿a qué partición va cada proceso?
  a) Una cola por partición y cada proceso asignado a la partición más pequeña en la que quepa:
     - Minimiza la fragmentación interna
     - No es óptimo, ya que puede haber particiones sin usar (las más grandes) y otras con grandes colas de espera
  b) Una cola para todos los procesos y al cargar un proceso se le asigna la partición más pequeña disponible:
     - Un proceso pequeño puede desperdiciar mucho espacio
     - Alternativamente, buscar en la cola el más grande que quepa, pero evitar excluir a un proceso más de *K* veces

5.2.1 Multiprogramación con particiones fijas

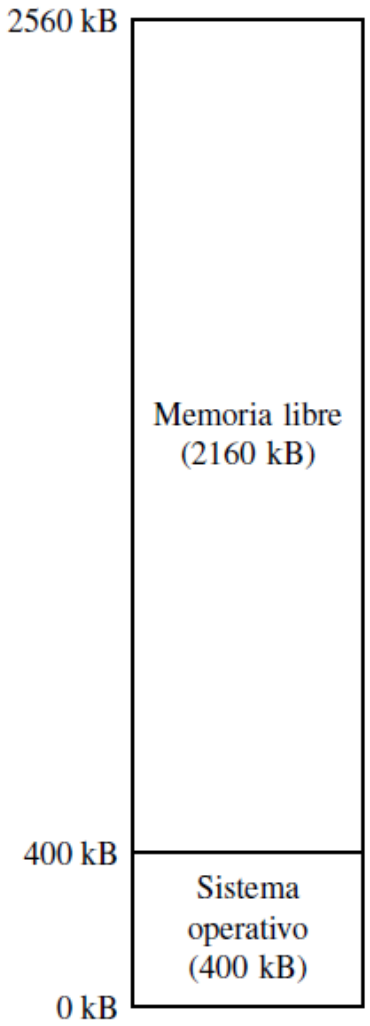


5.2.1 Multiprogramación con particiones fijas

- **Desventajas de las particiones fijas:**
  - Número de particiones limita el número de procesos activos **(grado de multiprogramación)**
    - En sistemas de tiempo compartido es un problema, dado que hay un gran número de procesos y se requerirían muchas particiones pero pequeñas (no pueden ser usadas por procesos más grandes)
  - Procesos pequeños causan mucha fragmentación interna
  - Fragmentación externa cuando hay espacio suficiente para trabajo nuevo, pero repartido en varias particiones

5.2.2 Multiprogramación con particiones variables

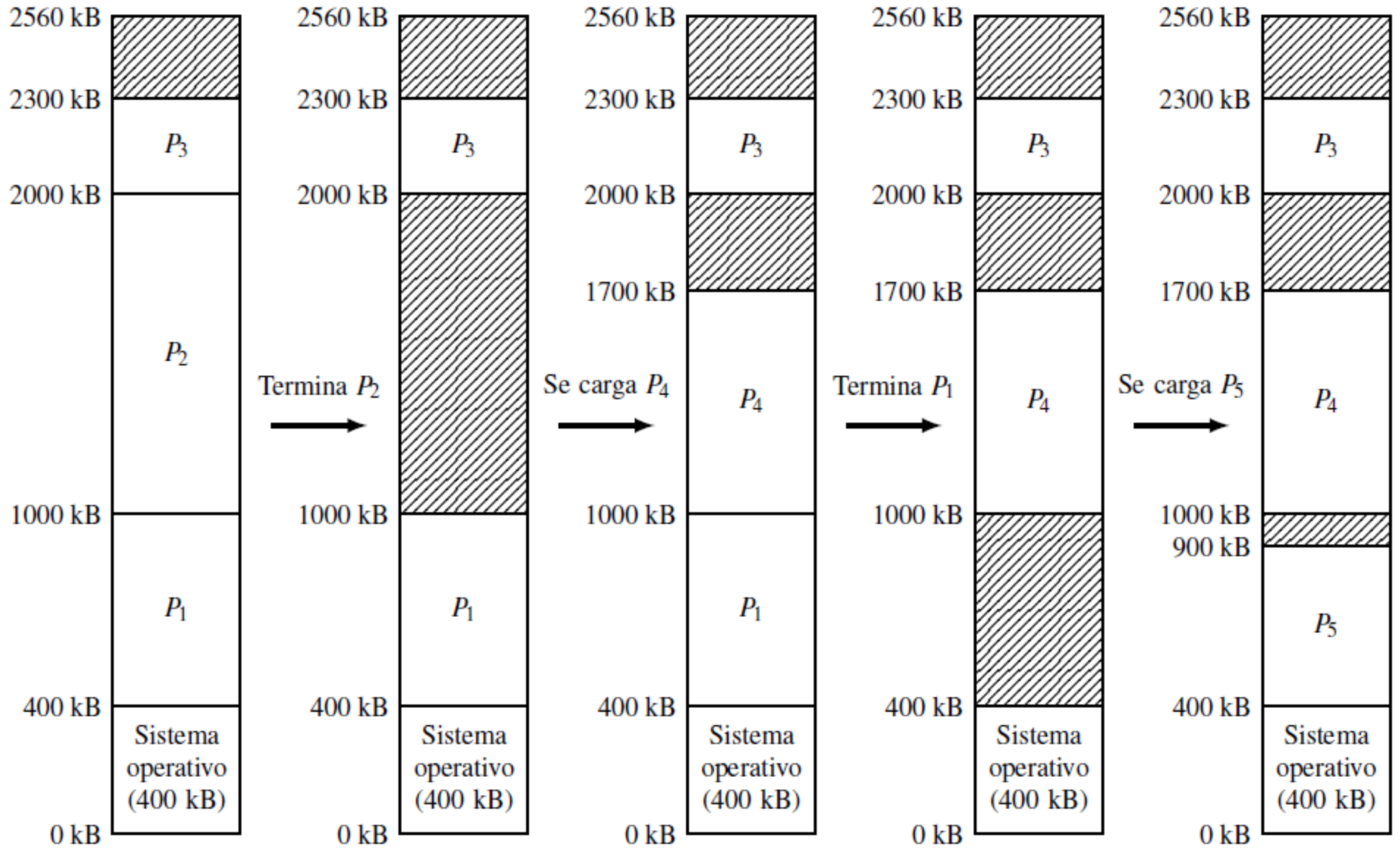
- Particiones variables en número, tamaño y posición
- Al cargar un proceso se le asigna exactamente la cantidad de memoria que necesita:
  - Las particiones se crean, desaparecen o se intercambian a disco según demanda
  - Mayor complejidad



Cola de trabajos

Proceso	Memoria	Tiempo de CPU
P_1	600 kB	10
P_2	1000 kB	5
P_3	300 kB	20
P_4	700 kB	8
P_5	500 kB	15

5.2.2 Multiprogramación con particiones variables



5.2.2 Multiprogramación con particiones variables

- **Políticas de asignación de huecos:**
  - **Primero en ajustarse (*first-fit*)** ⇒ Primer hueco disponible suficientemente grande (recorre desde el principio)
    - El más sencillo y rápido. Genera pequeñas particiones en las primeras posiciones de la memoria
  - **Siguiente en ajustarse (*next-fit*)** ⇒ Siguiente bloque disponible suficientemente grande (busca desde la última posición que se usó)
    - Peores resultados que el algoritmo del primer ajuste, necesita compactación más frecuentemente
  - **Mejor en ajustarse (*best-fit*)** ⇒ Tamaño más próximo al solicitado
    - Tiene los peores resultados (lenta) y tiende a generar huecos demasiado pequeños (más desperdicio)
  - **Peor en ajustarse (*worst-fit*)** ⇒ Hueco libre más grande
    - Lenta pero buen aprovechamiento de memoria

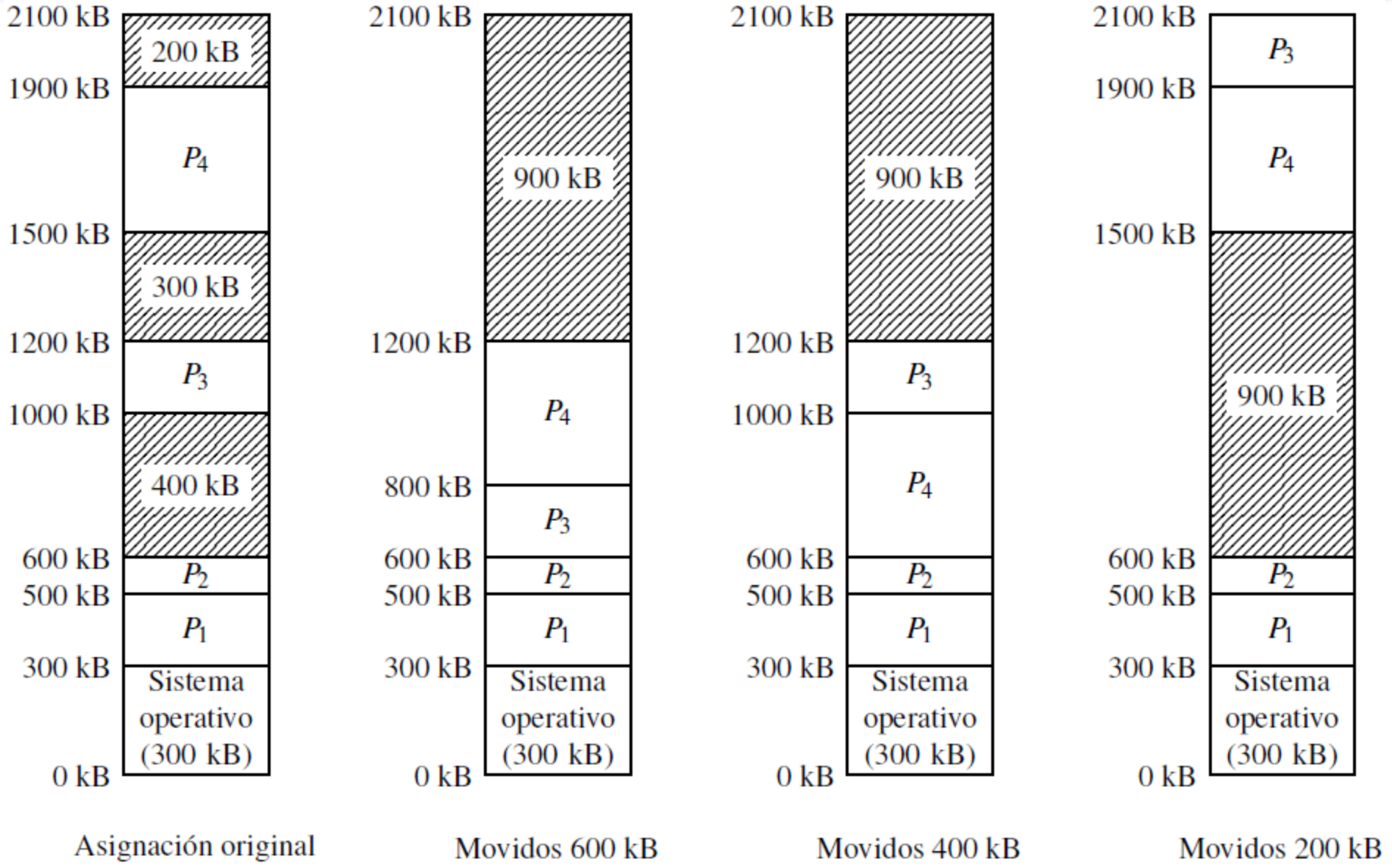
5.2.2 Multiprogramación con particiones variables

- Conforme pasa el tiempo, los procesos se crean y se destruyen, hay intercambios entre disco y memoria, dejando huecos en la memoria que se reutilizarán con otros procesos

- Esto provoca **fragmentación externa** de la memoria : huecos te pequeño tamaño que raramente pueden aprovecharse
  - Hay suficiente memoria libre pero ninguna zona contigua suficientemente grande

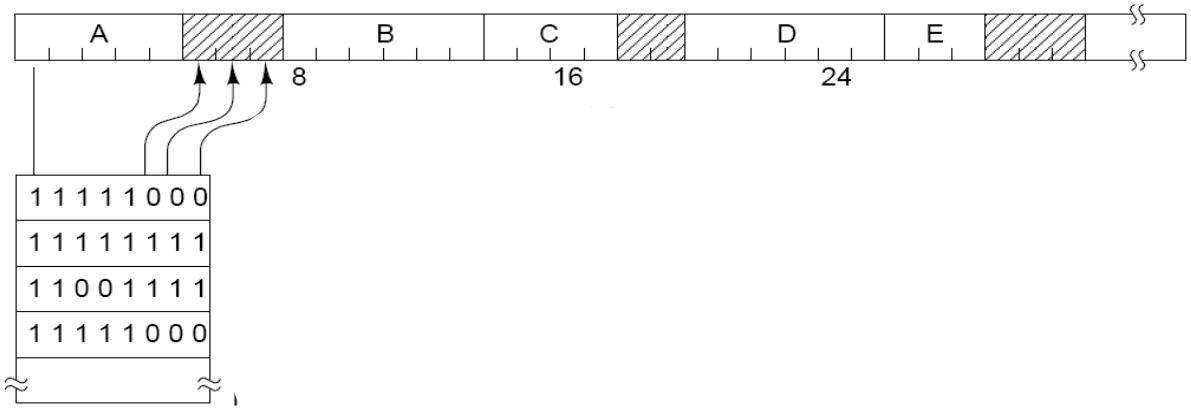
- Solución: **Compactación**: mover algunos procesos en memoria para combinar huecos entre ellos
  - Muy costosa en tiempo por lo que no se usa

5.2.2 Multiprogramación con particiones variables



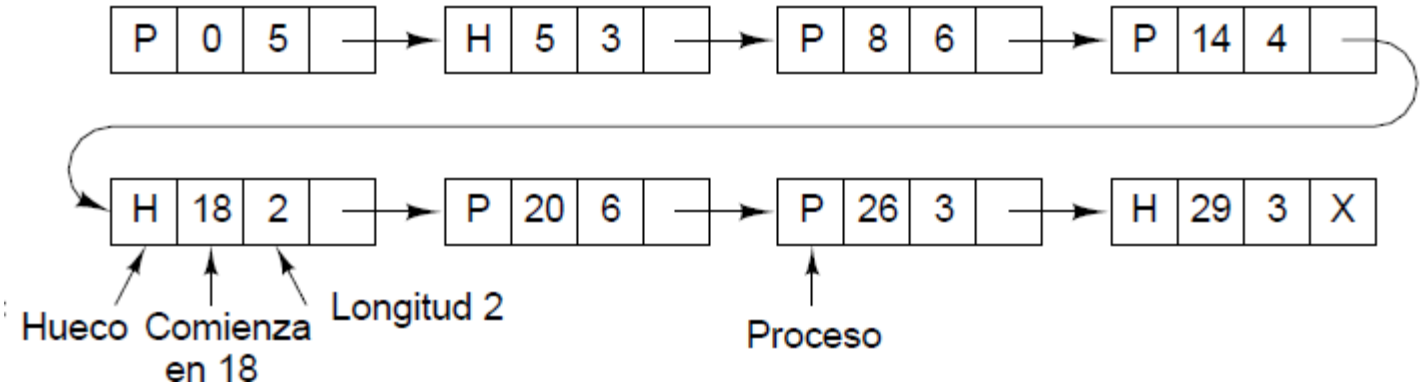
5.2.2 Multiprogramación con particiones variables

- **¿Cómo se controlan los huecos de memoria?**
  - **Mapas de bits**
    - Memoria dividida en unidades de asignación de tamaño fijo: a cada unidad le corresponde un bit del mapa de bits
    - El bit es: **0** si la unidad está libre; **1** si está ocupada
    - Problema 1: ¿Tamaño de la unidad de asignación?
      - Pequeño ⇒ mayor será el mapa de bits
      - Grande ⇒ fragmentación interna
    - Problema 2: puede ser lento (buscar largas cadenas de 0s)



5.2.2 Multiprogramación con particiones variables

- **¿Cómo se controlan los huecos de memoria?**
  - **Listas enlazadas**
    - Lista enlazada de segmentos de memoria asignados y libres, donde un segmento es un proceso (P) o un hueco (H) entre dos procesos:
      - Si la lista se ordena por dirección es más fácil fusionar el hueco que resulta cuando un proceso acaba
      - Si hay dos listas, una para memoria usada y otra para huecos, la asignación es más rápida (acelera la búsqueda de hueco), pero la liberación es más lenta



5.2.2 Multiprogramación con particiones variables

- **¿Cómo sabemos qué zona ocupada corresponde a cada proceso?**
  - La información sobre la memoria asignada a un proceso se obtiene a partir del Bloque de Control de Proceso (BCP)
- **Asignación de memoria de intercambio**
  - Cuando un proceso ha de ir a disco hay que asignarle un hueco en la zona de intercambio del disco:
    1. Cuando un proceso se crea, se crea también un hueco en la zona de intercambio para él
    2. Cuando un proceso está en memoria, no hay asignado un hueco en la zona de intercambio. Hay que buscarlo cuando sale de memoria
  - Los algoritmos para la administración de la zona de intercambio son similares a los usados para la gestión de la memoria principal

5.2.3 Reubicación y protección

- Los dos esquemas anteriores permiten tener varios procesos en memoria al mismo tiempo, cada uno en una partición, así que será necesario resolver **dos problemas**:
  - **Protección**: Un proceso no puede acceder a la partición de memoria de otros procesos ni a la del SO
  - **Reubicación**: A priori no se puede saber la partición a la que irá el proceso (*código relocalizable*: puede ejecutarse correctamente con independencia de las posiciones de memoria que ocupe)
- **Solución**: Una nueva abstracción para la memoria denominada **espacio de direcciones lógicas**
  - Conjunto de direcciones que usa un proceso para direccionar la memoria. Cada proceso tiene su propio espacio de direcciones que comienza por la posición 0.
- **Problema:**
  - ¿Cómo le damos a cada proceso su propio espacio de direcciones de forma que la dirección 28 en un proceso sea una dirección física diferente de la dirección 28 en otro?

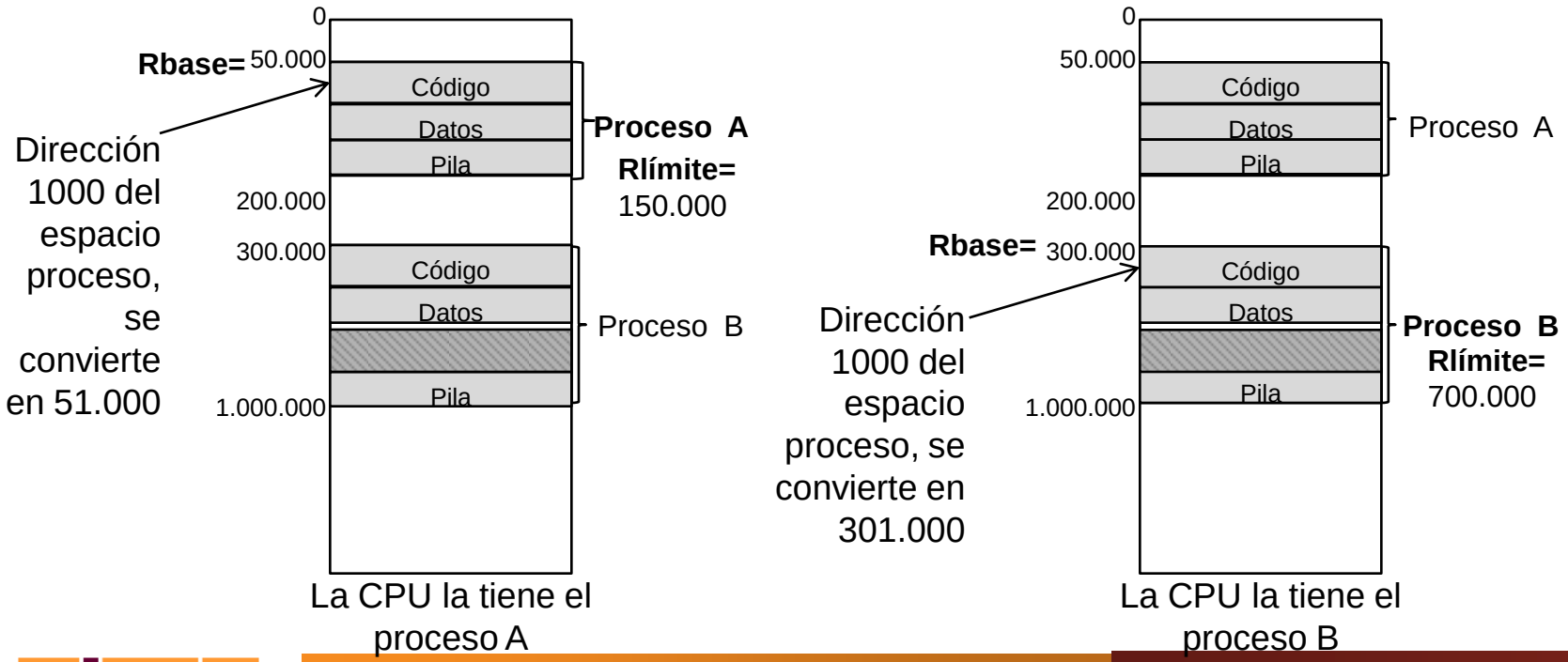
5.2.3 Reubicación y protección

- **Solución**: La CPU incorpora registros base y límite
  - Se utilizan para generar las direcciones de memoria físicas a partir de las direcciones lógicas
  - Cuando se asigna la CPU el registro base se carga con la dirección física donde comienza el proceso, y el límite con el tamaño de la partición. Estos datos se cogen del BCP del proceso y estamos suponiendo que el proceso se encuentra contiguo en memoria
  - Ahora cada proceso tiene su propio espacio de direcciones que comienza por la posición 0. Cuando se accede a una dirección, la CPU comprueba si supera el valor del registro límite (en cuyo caso se genera una excepción y aborta el proceso) y si no es así, le suma el valor del registro base

- Ejemplo:
  - Proceso situado en la posición 16000 de memoria física
  - Accede a la posición 100 de su espacio de direcciones de 4096 bytes: realmente está accediendo a la posición 16100
  - No podría acceder a la posición 20096 y siguientes

5.2.3 Reubicación y protección

- Los registros base y límite son una forma sencilla de asegurar protección (no puedes salirte del espacio asignado) y reubicación (puedes cambiar la posición del proceso en memoria, simplemente cambiando el valor del registro base)
- La zona sombreada indica que se ha dejado un espacio para que la pila crezca hacia direcciones inferiores y la zona de datos hacia direcciones crecientes.



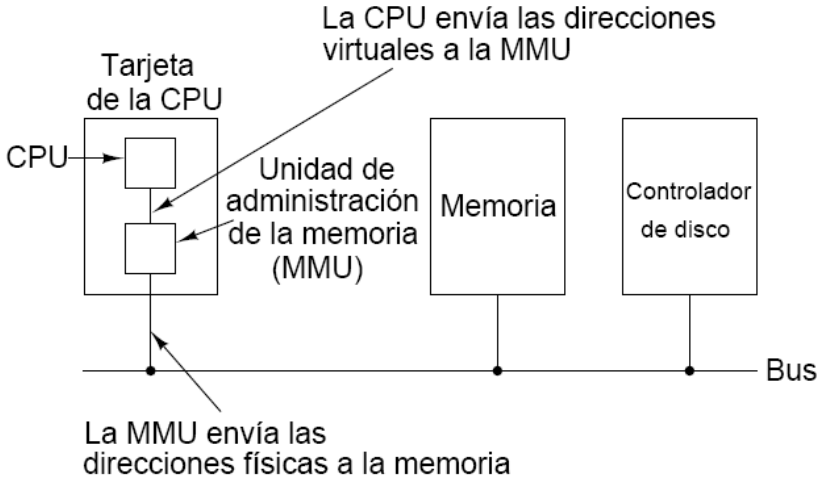
Índice

5.3 Paginación

- **Memoria virtual:** el tamaño combinado del programa, sus datos y su pila podría exceder la cantidad de memoria física que se le puede asignar. El sistema mantiene:
  - Las partes del proceso que se están usando en memoria principal
  - El resto del proceso en disco
- El SO decide en cada instante qué está en memoria y qué en disco:
  - Transfiere información entre disco y memoria de forma automática y transparente para los procesos
- Vamos a estudiar dos técnicas de memoria virtual:
  - **Paginación**: es la más utilizada
  - Segmentación
  - En ambos casos, se supone un proceso dividido en trozos y se mantiene en memoria principal solo las partes activas

5.3 Paginación

- **Paginación:** Una de las técnicas de memoria virtual más usadas que permite ejecutar programas más grandes que la memoria física disponible de forma transparente al programador
- **Direcciones virtuales vs. Direcciones físicas**
  - A las direcciones generadas por el proceso se les llamará **direcciones virtuales**, y a su espacio de direcciones, **espacio de direcciones virtual**, y habitualmente supera el tamaño de la memoria física
- La **unidad de administración de memoria** (**MMU**) traduce las direcciones virtuales en direcciones físicas



5.3.1 Funcionamiento de la paginación

- **El espacio de direcciones virtuales** del proceso se divide en trozos de igual tamaño llamados **páginas virtuales**. Cada página es un rango contiguo de direcciones (normalmente de 4 KiB)
- La memoria física también se divide en páginas denominadas **marcos de página** (en ETC se las denominaba páginas físicas)
- La **página** es la unidad de intercambio de memoria: las transferencias entre la memoria principal y el disco siempre se realizan en páginas
- Cuando el proceso hace referencia a una dirección que **no está en memoria física**, la MMU lo detecta y hace que la CPU genere una excepción (**fallo de página**). El SO buscará la página que falta y la pondrá en memoria física. Después, se retoma la ejecución por la instrucción que falló
- La forma que tenemos de llevar el control de en qué marco físico se encuentra cada página virtual es la **Tabla de Páginas**

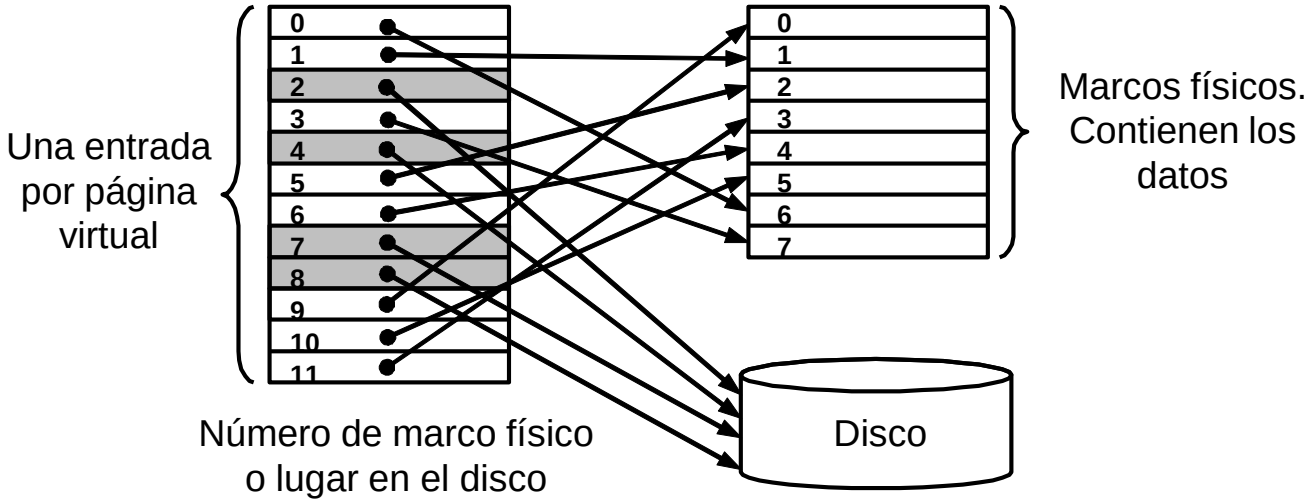
5.3.2 Tabla de páginas

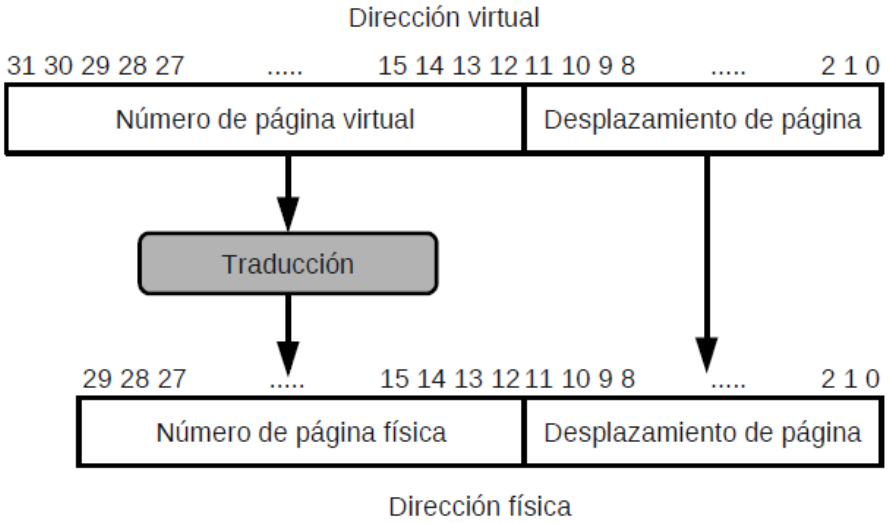
- **Establece la correspondencia entre direcciones virtuales y físicas** (páginas virtuales→marcos de página):
  - Para un funcionamiento óptimo el tamaño de la página y el marco debe ser potencia de 2
  - La dirección virtual se divide en **número de página virtual** y **desplazamiento**

Nº página virtual	Desplazamiento

  - El número de página virtual se usa como **índice** para consultar la tabla de páginas
  - La entrada correspondiente de la tabla de páginas indica el **número de marco de página** (si está en memoria). Si no está en memoria, puede indicar dónde se encuentra en disco
  - El número de marco de página se concatena delante del desplazamiento para formar una **dirección física**

5.3.2 Tabla de páginas

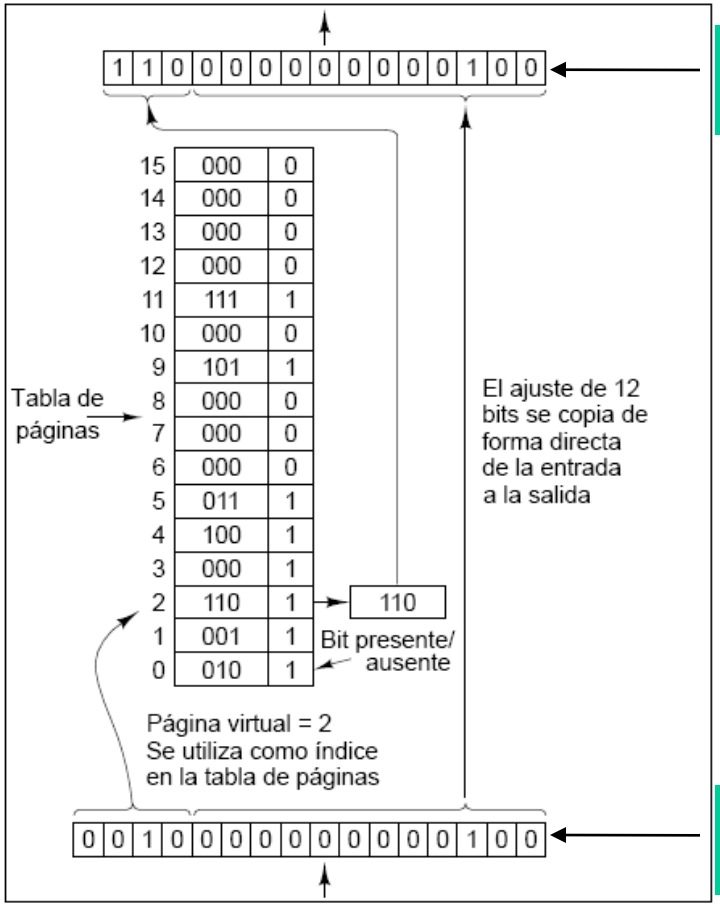


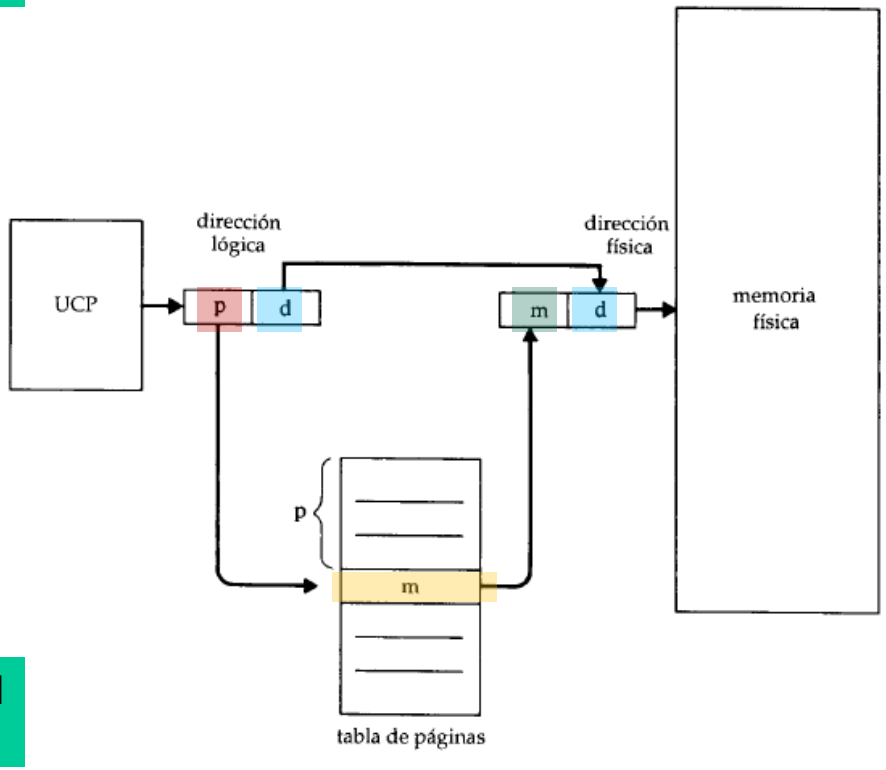


- Página de 4KiB = 2^{12} bytes → 12 bits de desplazamiento
- Número de página virtual = dirección virtual / 4096
- Desplazamiento dentro de la página = dirección virtual % 4096

5.3.2 Tabla de páginas

- Ejemplo con direcciones virtuales de 16 bits, tamaño de página de 4096 bytes y direcciones físicas de 15 bits:
- 4 bits de página virtual y 12 bits de desplazamiento. El número de marco físico tiene 3 bits



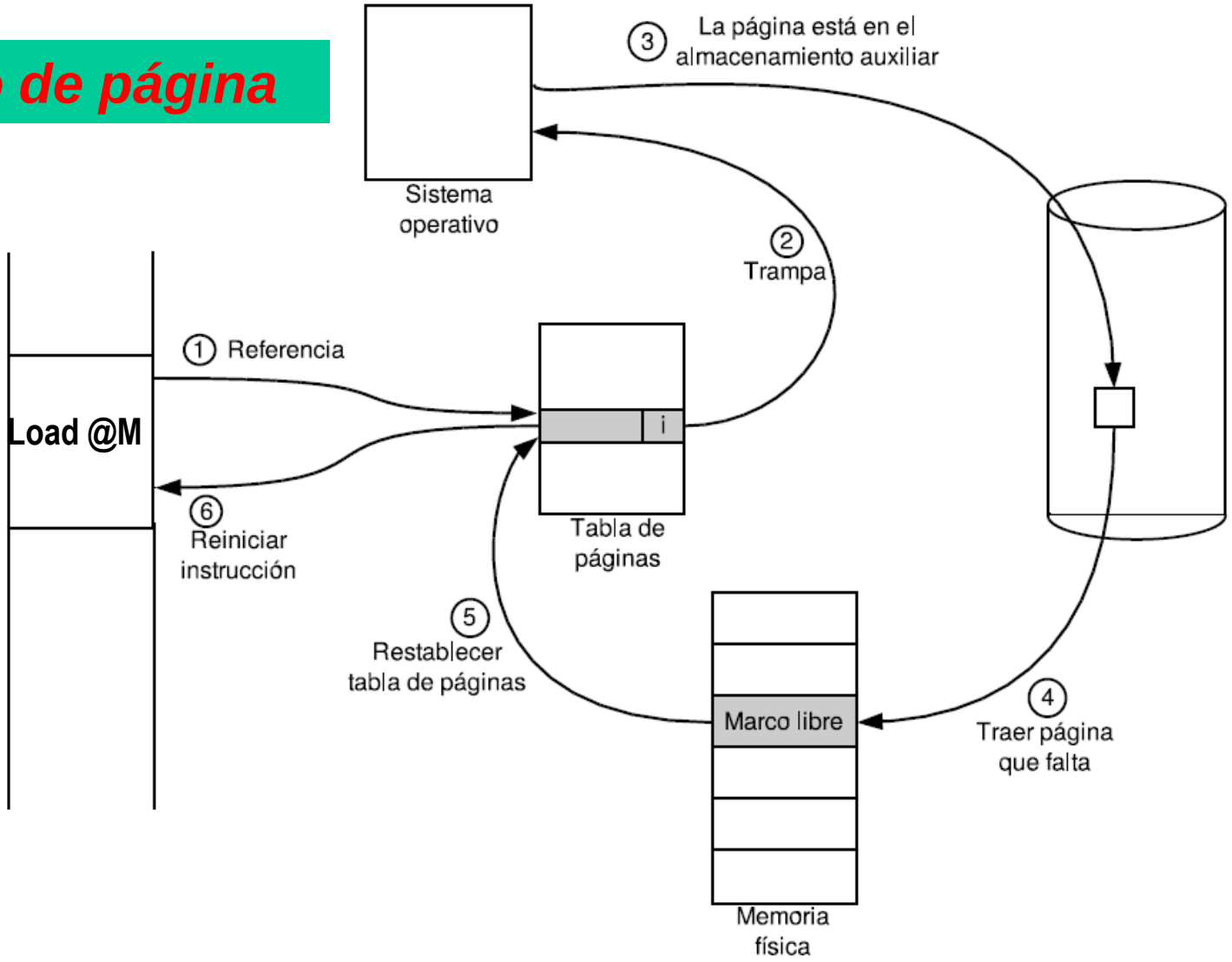


5.3.2 Tabla de páginas

- En general, espacio de direcciones virtuales más grande que la memoria física (por ejemplo, en x86-64 las direcciones virtuales tienen 48 bits)
- ¿Cómo sabemos qué páginas virtuales están en memoria y cuáles no?
  - **Bit presente/ausente** (en ETC, bit de validez) en cada entrada de la TP
- ¿Qué sucede si no hay asociación página/marco? ⇒ **Fallo de página**
  1. La MMU detecta que la página no tiene correspondencia
  2. La MMU provoca un fallo de página (excepción)
  3. S.O. escoge un marco que no se esté usando mucho (si es necesario vuelve a escribir su contenido en disco)
  4. Trae la nueva página a ese marco
  5. Modifica la tabla de páginas
  6. Reinicia la instrucción

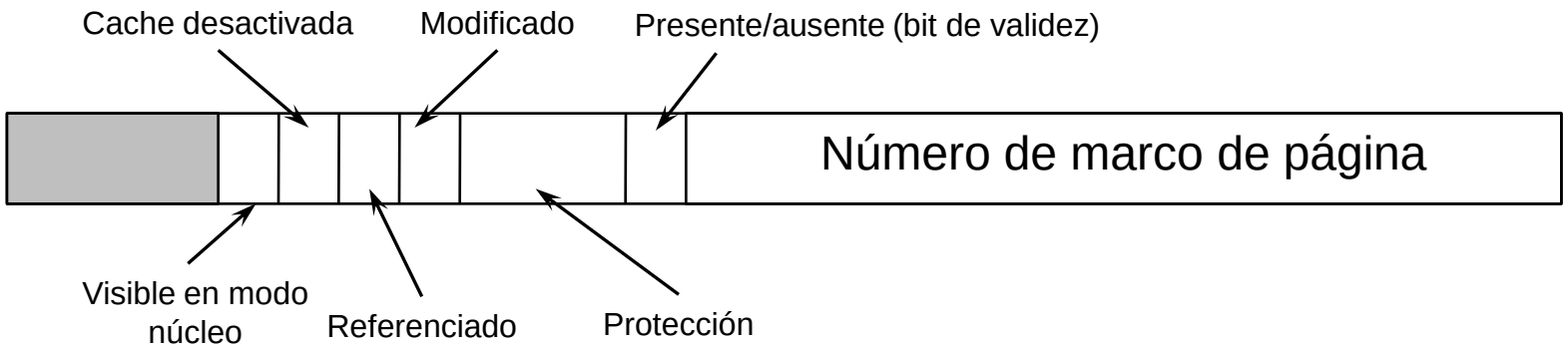
5.3.2 Tabla de páginas

Fallo de página



5.3.2 Tabla de páginas

- Estructura de una entrada de tabla de páginas (posibles campos)
  - Número de marco físico
  - Bits presente/ausente (en ETC era el bit de validez) ⇒ 1 entrada válida, 0 no está en memoria
  - Bits de protección ⇒ tipos de acceso permitido (Lect/Esc/Ejec…)
  - Bit **modificado**
  - Bit **referenciado** o **solicitado** ⇒ se ha referenciado a la página (en ETC se denominaba bit de uso)
  - *Caching* desactivado para *mapeo* en memoria ⇒ evitar el uso de la caché para los contenidos de esa página
  - Página visible en modo núcleo



5.3.2 Tabla de páginas

A la hora de diseñar la TP hay que tener en cuenta que:

- La traducción de direcciones virtuales a físicas debe ser **rápida**:
  - **Cada acceso a memoria** ⇒ asociación dirección virtual-física
  - Cada instrucción ⇒ 1, 2 ó más accesos a la tabla de páginas
  - Difícil implementación en registros hw (1 reg por entrada):
    - Es costoso si la tabla de páginas es grande (muchos regs)
    - Cada cambio de proceso hay que inicializar los registros
  - Se puede utilizar memoria principal para almacenarla, más un registro con la dirección de memoria de inicio
  - Si se implementa completamente en memoria es más lento ⇒ varios accesos a memoria por cada acceso efectivo
- El **tamaño** de la tabla no puede ser muy grande:
  - Direcciones virtuales de 48 bits y 4KiB/página ⇒ espacio de direcciones virtual de 256TiB y 64×2^{30} páginas
  - Con entradas de 8 bytes: ¡ 512 GiB la tabla de páginas !
  - Además, una por proceso (¿Por qué?)

5.3.2 Tabla de páginas. TLB

Para conseguir traducciones rápidas: TLB (*translation lookaside buffer*)

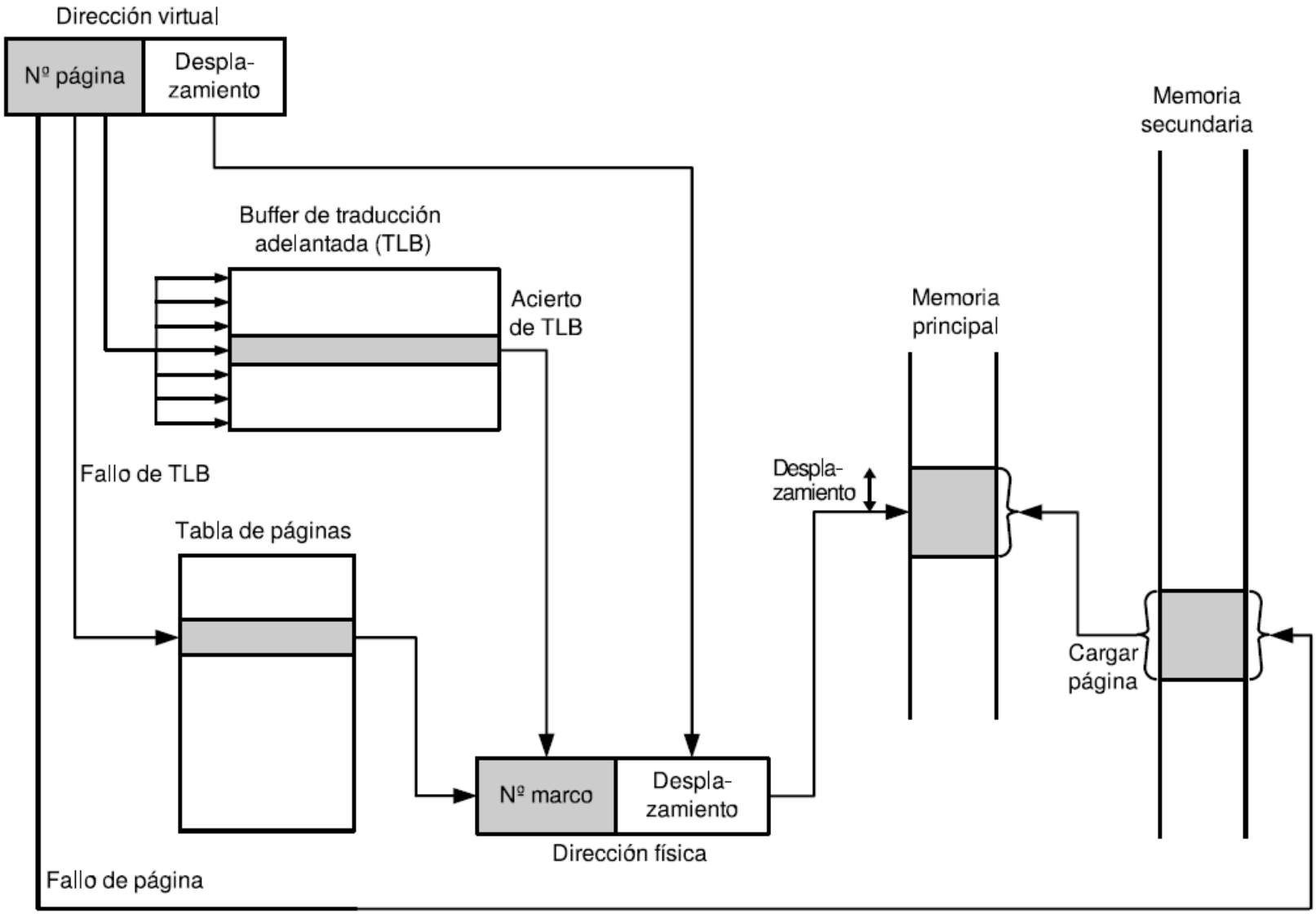
- Hardware que traduce direcciones virtuales a físicas sin acceder a la tabla de páginas
  - Suele implementarse como caché totalmente asociativa y está en el interior de la MMU
  - Dispone de un número pequeño de entradas
  - Cada entrada contiene información sobre una página:
    - Bit de validez
    - Nº de página Virtual
    - Bit de modificado
    - Bits de protección
    - Nº de marco de página

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

5.3.2 Tabla de páginas. TLB

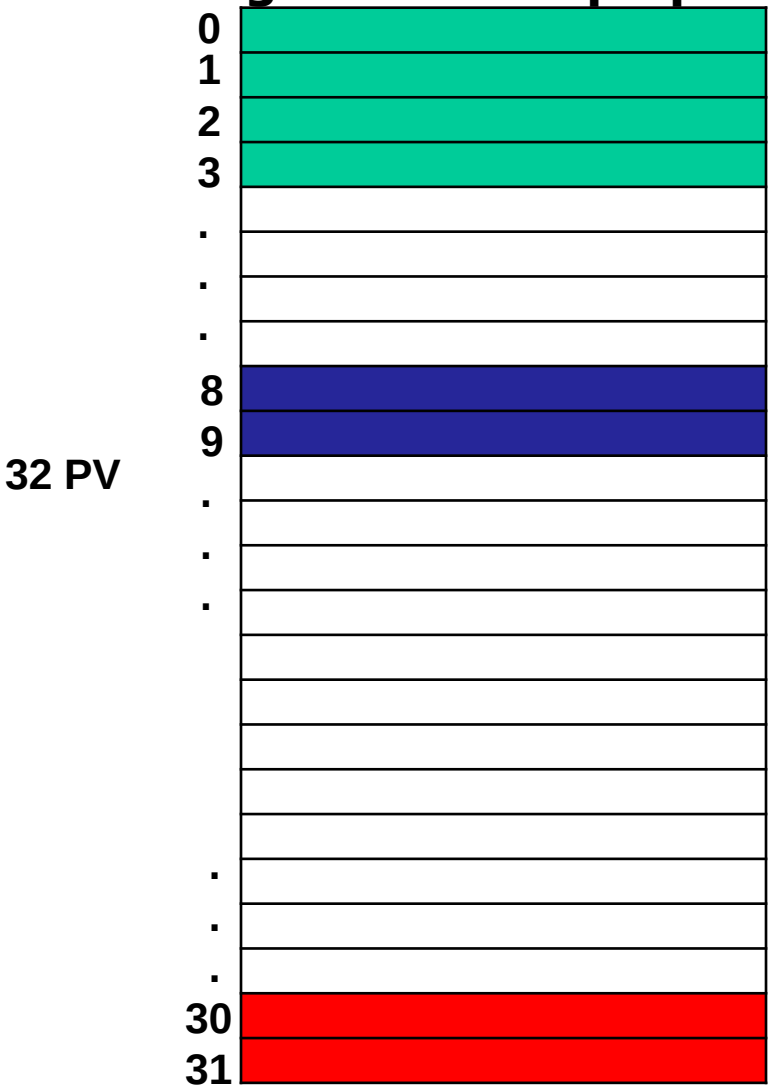
- **Funcionamiento del TLB**:
  - Dirección virtual ⇒ se comprueba, en paralelo en todas las entradas, si su número de página está presente en el TLB
  - *Si está:*
    - Si no viola los bits de protección ⇒ el número de marco se toma del TLB
    - Si viola los bits de protección ⇒ **Fallo de protección**
  - *Si no está ⇒ Fallo de TLB*
    - La MMU consulta la tabla de páginas
    - Se almacena la traducción en el TLB (desaloja una entrada del TLB y la sustituye por la nueva entrada)
  - ¿Qué pasa con el TLB en los cambio de proceso dado que todos los procesos manejan las mismas direcciones virtuales? Sol:
    - Sol1: Usar una instrucción especial para hacer un flush
    - Sol2: Añadir al TLB el PID del proceso dueño de cada entrada y un registro hw con el PID del proceso activo

5.3.2 Tabla de páginas. TLB



5.3.2 Tabla de páginas. Tablas multinivel

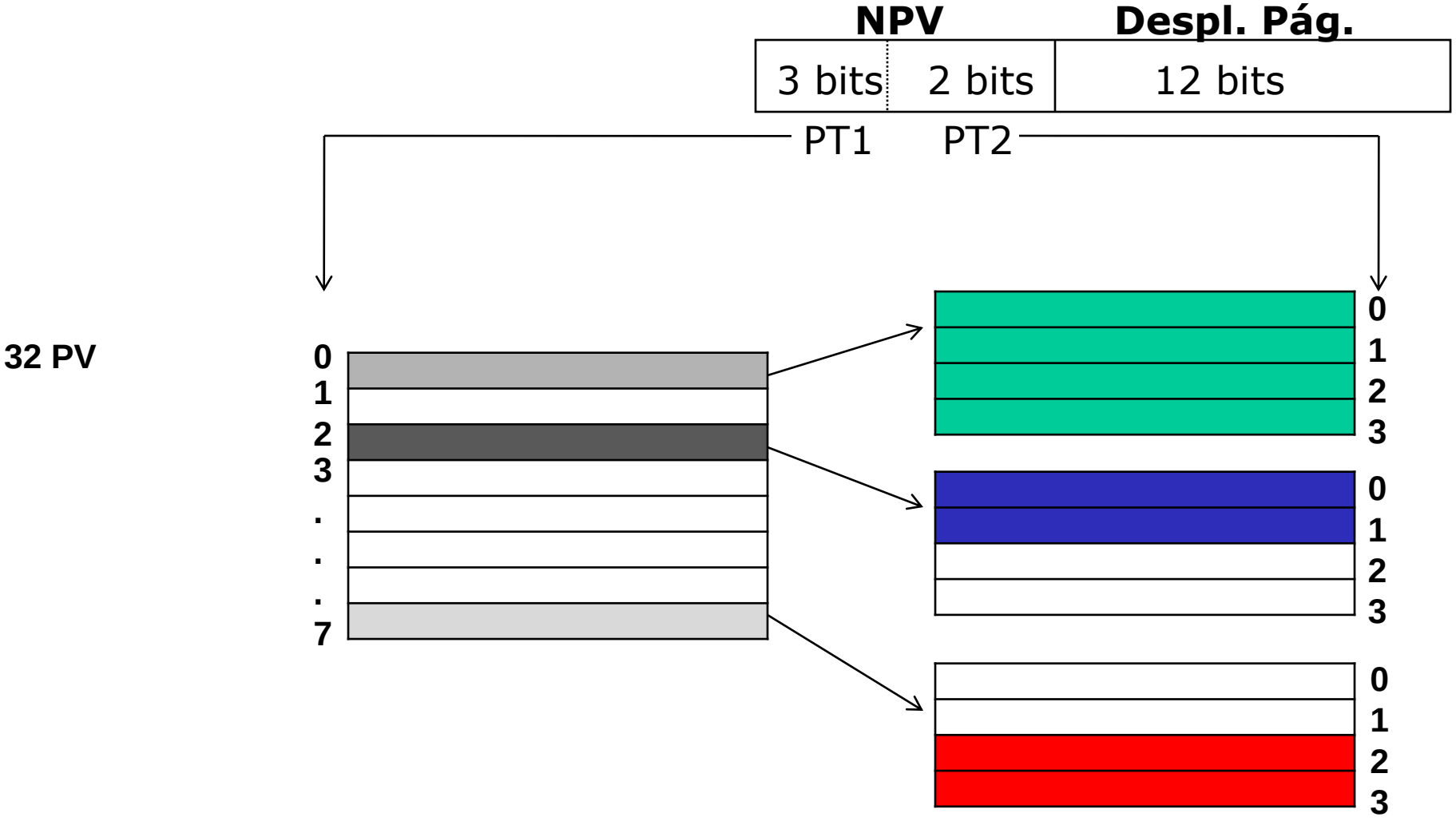
Cómo lograr TP más pequeñas: TP multinivel



NPV	Despl. Pág.
5 bits	12 bits

5.3.2 Tabla de páginas. Tablas multinivel

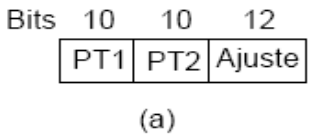
Cómo lograr TP más pequeñas: TP multinivel

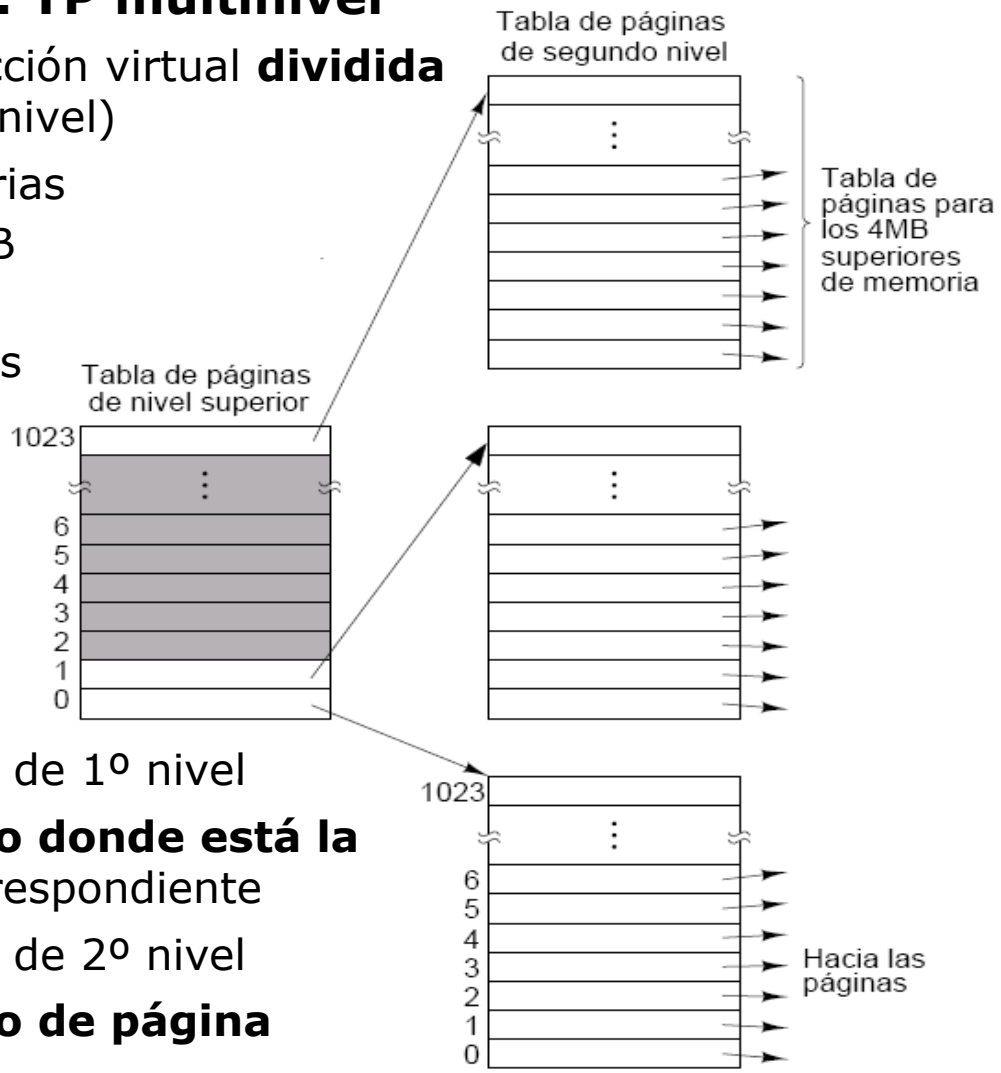


5.3.2 Tabla de páginas. Tablas multinivel

Cómo lograr TP más pequeñas: TP multinivel

- El campo **nº de página** de la dirección virtual **dividida en varios campos** (uno por cada nivel)
- En memoria sólo las tablas necesarias
- Proceso de 12MB: 4MB código, 4MB datos, 4MB pila
  - Total: 4 tablas × 1024 entradas
  - Antes: 1 tabla × 1 048 576 entradas





- PT1: entrada de la tabla de páginas de 1º nivel
- Tabla de páginas de 1º nivel: **marco donde está la tabla de páginas de 2º nivel** correspondiente
- PT2: entrada de la tabla de páginas de 2º nivel
- Tabla de páginas de 2º nivel: **marco de página donde está la página**

5.3.2 Tabla de páginas: TP invertida

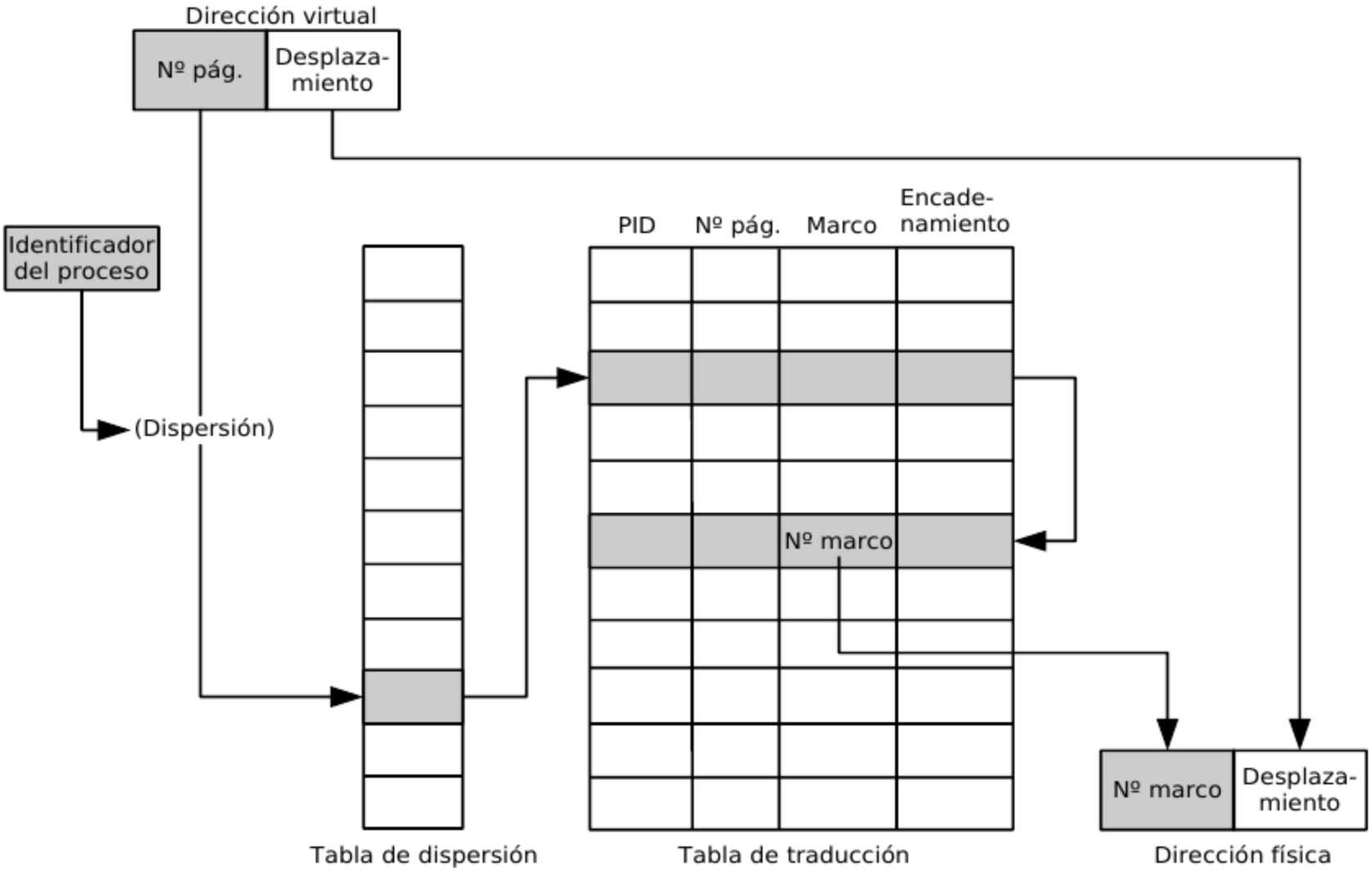
Cómo lograr TP más pequeñas: TP invertida

- En las arquitecturas de 64 bits actuales, las tablas de páginas pueden ser enormes:
  - Tablas de páginas multinivel demasiado complejas por la gran cantidad de niveles necesarios
  - Si páginas muy grandes ⇒ menos páginas
    - Aumenta fragmentación interna
  - Solución: **tabla de páginas invertida (usadas en arquitecturas de 64 bits PowerPC y UltraSPARC)**

5.3.2 Tabla de páginas: TTP invertida

- Características de la TP invertida:
  - Una entrada **por cada marco de página** de la memoria física
    - Tamaño de la TP depende de la cantidad de memoria física
    - Con direcciones virtuales de 48 bits, 4KiB/página y 4 GiB de RAM, sólo se necesitan 1 048 576 entradas
      - 2^{32} Bytes/RAM / 2^{12} Bytes/Marco = 2^{20} Marcos/RAM
    - Una tabla de páginas para TODOS los procesos (¿por qué?)
  - Cada entrada indica qué página virtual (NPV) de qué proceso (PID) está en el marco correspondiente (NMP)
  - Problema: la traducción de direcciones virtuales a físicas es mucho más difícil:
    - Cada vez que un proceso referencia a una página, hay que buscar una entrada (PID, NPV) en toda la tabla invertida para averiguar el marco (NMP)
  - Sol: **Tabla de dispersión + Tabla de traducción (+ TLB)**

5.3.2 Tabla de páginas: TTP invertida



5.3.3 Mapa de memoria de un proceso

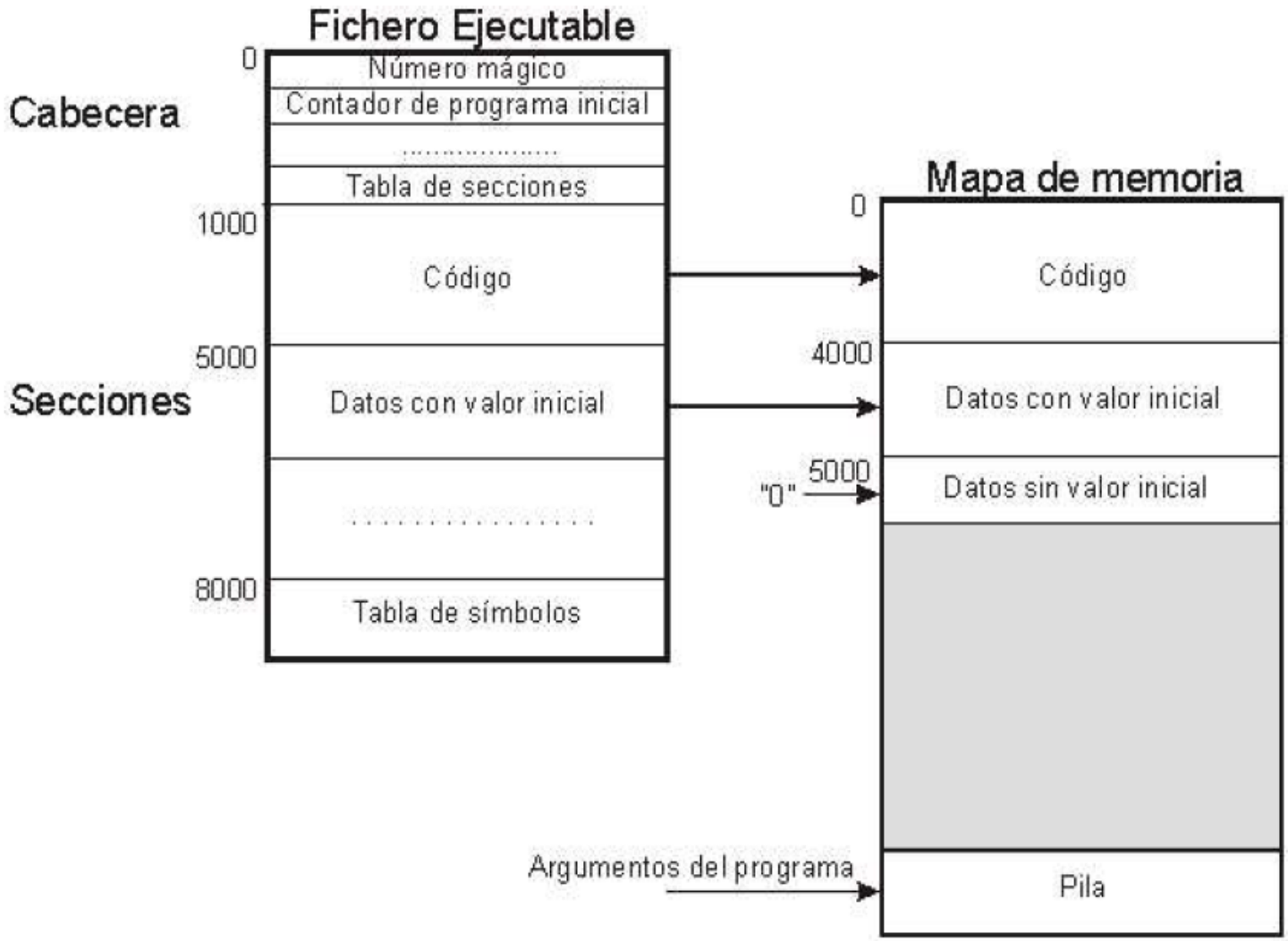
- Para poder ejecutarse, un proceso necesita tener en memoria: código, zona de datos y pila
  - Código y datos con valor inicial se toman directamente del ejecutable
  - Pila y zona de datos sin valor inicial se crean al crear el proceso
- A la forma en que se estructura la memoria virtual del proceso y lo que se mapea en cada rango de direcciones de memoria le llamamos **mapa de memoria del proceso**
  - No hay que confundirlo con lo que hay almacenado en la memoria física
- El mapa de memoria comienza construyéndose a partir del ejecutable cuando se crea el proceso:
  - Se asignan rangos de direcciones virtuales contiguas dedicadas al mismo propósito, que denominamos **regiones de memoria**
- El mapa de memoria debe controlar qué zonas del espacio de direcciones virtual están ocupadas y cuáles no

5.3.3 Mapa de memoria de un proceso

- Inicialmente se definen 2 regiones cuyos contenidos se toman directamente del ejecutable:
  - una región dedicada al **código** (suele ubicarse al principio de la memoria)
  - otra a los **datos con valor inicial**

- También se crean otras regiones que no tienen contenido inicial:
  - los **datos sin inicializar** del programa
  - la **pila** (suele ubicarse al final de la memoria asignada)

5.3.3 Mapa de memoria de un proceso



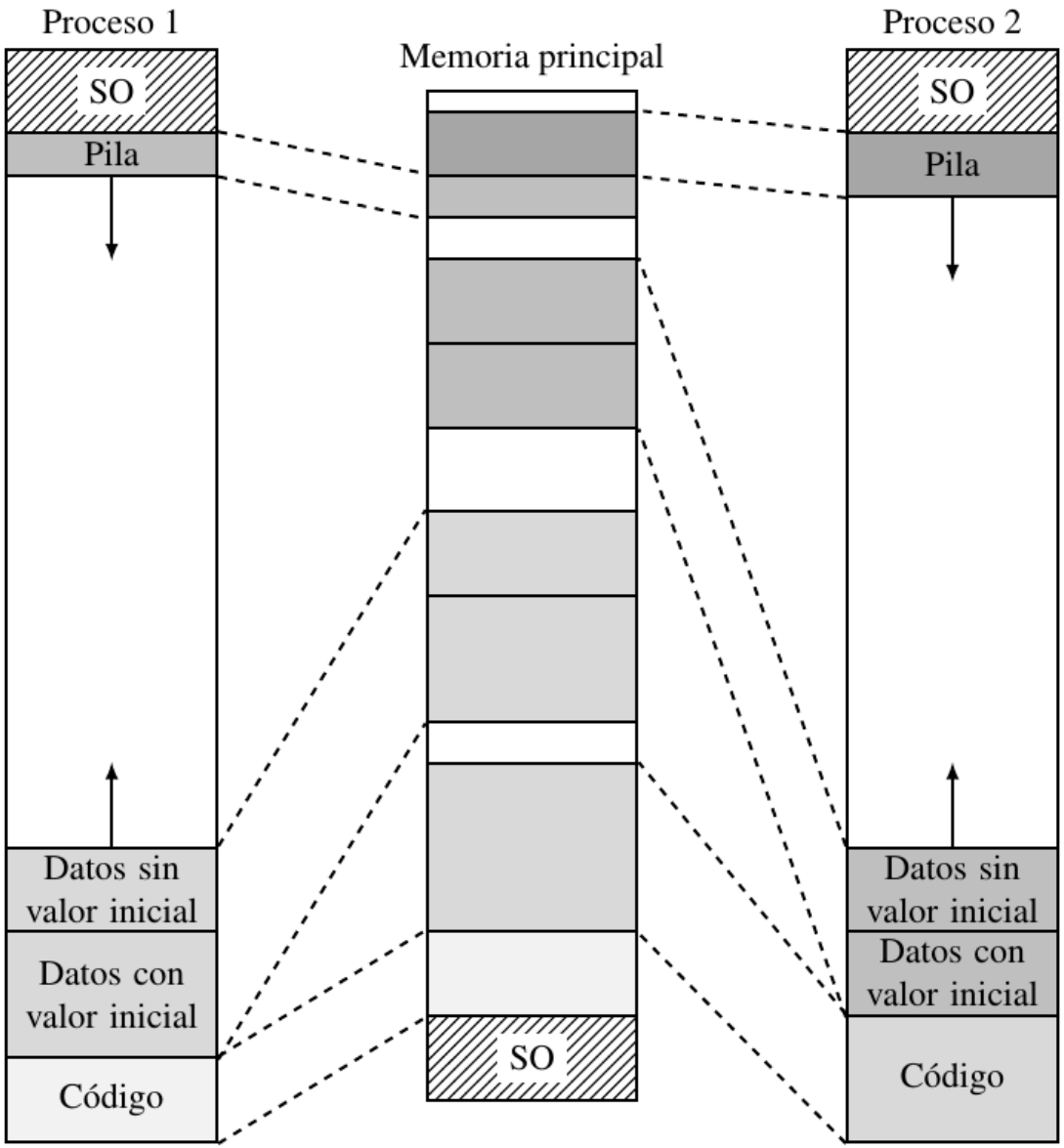
5.3.3 Mapa de memoria de un proceso

- Cada región tiene asociadas una serie de **propiedades y características:**
  - Un **soporte** que describe de dónde se obtienen los contenidos asociados a esa región:
    - **Soporte en fichero**: Se encuentran en un fichero o parte del mismo. P.ej. las páginas del código se encuentran en el propio ejecutable
    - **Sin soporte**: No tienen contenido inicial. P. ej. las páginas de pila. Cuando se modifiquen y tengan que ser expulsadas de memoria irán a la zona de intercambio en disco
  - **Tipo** de compartición:
    - Privada: Contenido de la región solo accesible al proceso
    - Compartida: Contenido puede compartirse entre procesos
  - **Protección**: Lectura, escritura, ejecución sobre páginas
  - **Tamaño** fijo o variable:
    - En el caso variable, se indica si crece hacia direcciones de memoria mayores (el montón, o zona de datos dinámicos del programa) o menores (la pila)

5.3.3 Mapa de memoria de un proceso

- Además de las 3 regiones vistas, son comunes otras para:
  - **Heap (o montón):** Para soportar la memoria dinámica de los lenguajes de programación. Comienza típicamente tras la región de datos sin valor inicial y crece hacia direcciones superiores. No tiene soporte asociado (está inicialmente a cero) y va creciendo conforme el proceso necesita memoria
  - **Ficheros proyectados:** Se puede mapear el contenido de un fichero a un rango de direcciones virtuales (por ejemplo para tratarlo como un array), simplemente haciendo que el soporte asociado sea el fichero. Cuando se produzca un fallo de página, se buscarán los datos en el fichero mapeado, dando la sensación de que estamos accediendo a un array cuando estamos realmente usando el fichero
  - **Pila de los hilos:** Cada hilo necesita una pila así que cuando se crean hilos se crean también una región asociada a cada una de sus pilas

5.3.3 Mapa de memoria de un proceso



5.3.3 Mapa de memoria de un proceso

- Organización del mapa de memoria en regiones + paginación = Implementación eficiente de llamada **fork**()
  - Esta llamada crea una copia exacta del proceso llamante, salvo que tiene un BCP y un PID diferentes
  - El nuevo proceso hereda todo lo del padre: ficheros abiertos, variables de entorno, **mapa de memoria (copia de la TP del padre)**, etc.
  - Eficiente pues no se require duplicar en memoria física la memoria del proceso padre para asignársela al hijo
  - **Problema**: regiones con permiso de escritura (datos, pila, etc)
    - Cuando un proceso modifica un dato, el resto de procesos que comparten la región no deben ver la modificación
    - **Solución 1**: Las regiones modificables no se comparten y se replican cuando se crea el proceso

5.3.3 Mapa de memoria de un proceso

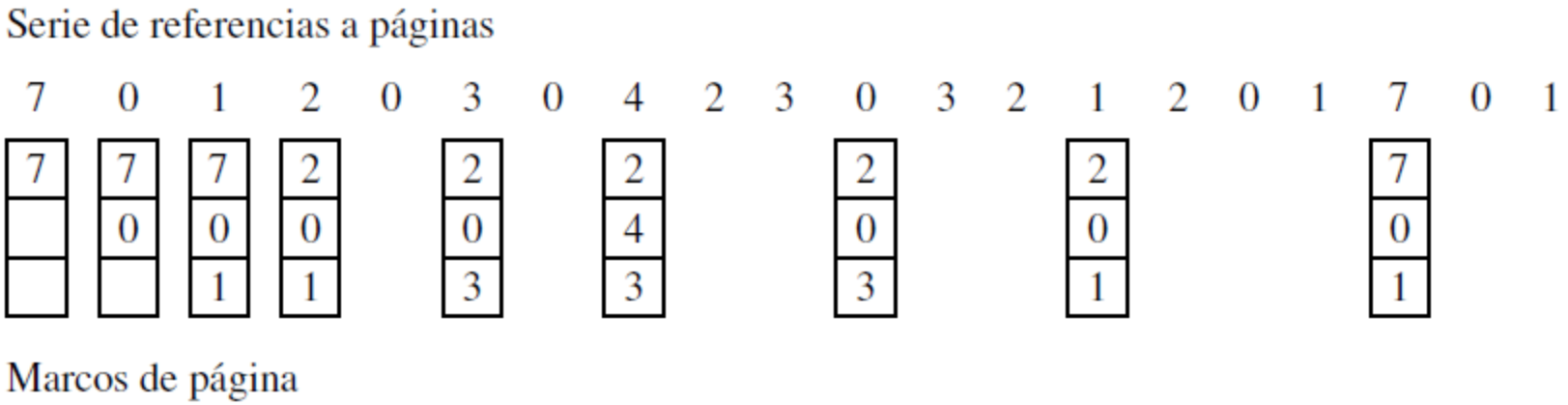
- **Problema**: regiones con permiso de escritura (datos, pila, etc)
  - **Solución 2**: Copia en escritura (CoW)
    - **Al crear el proceso, se comparten todas las regiones entre padre e hijo**, haciendo que en los dos procesos cada región mapee a la misma memoria física
    - Se **desactiva permiso de escritura** de todas las páginas de las regiones modificables
      - » Ahora si **uno de los dos procesos intenta escribir**, saltará una excepción, el SO comprobará que la región está compartida y es modificable y hará una **copia de la página que falló**, asignándola a la TP del proceso escritor y quitando la protección contra escritura en ambas TTP
- El **SO** forma parte del mapa de memoria de un proceso: transferencia de información rápida entre proceso y SO
  - Las páginas virtuales de la región del SO tienen activado el bit "visible en modo núcleo" en la TP.
  - KPTI separa las TPs para modo usuario y modo núcleo

5.3.4 Algoritmos de reemplazo de páginas

Algoritmo de reemplazo de páginas:

- Fallo de página y no hay marcos libres, ¿qué marco seleccionar para la nueva página?

- **Algoritmo óptimo**
  - Reemplaza la página que más tiempo va a tardar en necesitarse
  - Irreal: **el orden de las referencias a memoria no se puede saber de antemano**

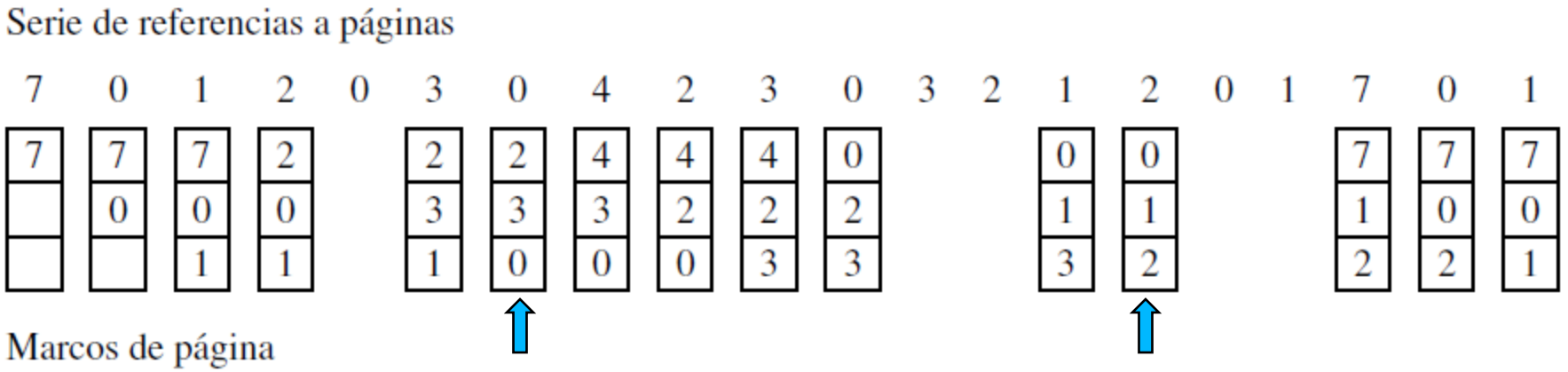


5.3.4 Algoritmos de reemplazo de páginas

- **Algoritmo NRU (No Usada Recientemente)**
  - Basado en la utilización de los bits de R (*referencia*) y M (*modificado*) de la página
  - De forma periódica el bit R se establece a 0 ⇒ distinguir las páginas que se han solicitado recientemente
  - El bit M se pone a 1 cuando se modifica la página ⇒ reemplazo de una página con bit M a 1 es más costoso
  - Se establecen 4 categorías en base a los bits R y M:
    - CLASE 0: R = 0, M = 0
    - CLASE 1: R = 0, M = 1
    - CLASE 2: R = 1, M = 0
    - CLASE 3: R = 1, M = 1
  - Desaloja aleatoriamente una página de la clase de número más bajo que no esté vacía
  - Funciona bien, es rápido y es sencillo de implementar

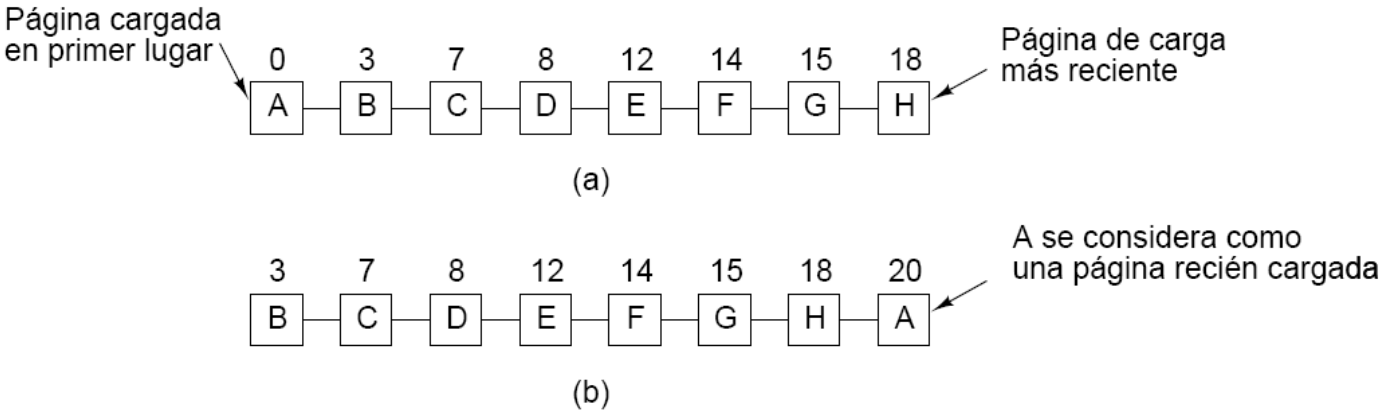
5.3.4 Algoritmos de reemplazo de páginas

- **Algoritmo FIFO: Primero en entrar, primero en salir**
  - El SO mantiene una lista de las páginas que hay en memoria
  - Reemplaza la página que entró hace más tiempo en memoria (la primera que entró)
  - Fácil de implementar pero no muy bueno, al no tener en cuenta el uso de las páginas



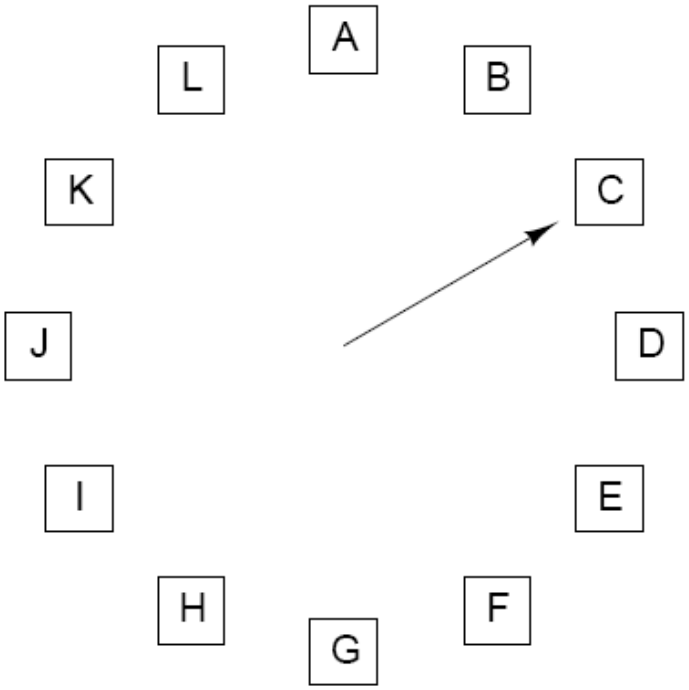
5.3.4 Algoritmos de reemplazo de páginas

- **Algoritmo de la segunda oportunidad**
  - Modificación del FIFO para tratar de evitar desalojar una página que se use mucho: se tiene en cuenta **bit R**
  - **Funcionamiento**:
    - Si el **bit R** de la 1ª página es 0, se selecciona esa página
    - Si su **bit R es 1**, se mueve al final de la lista y se anula el valor del bit R (como si fuera nueva)
    - Si todos los bit R son 1, su comportamiento es el de un FIFO
  - Busca una página antigua a la que no se le haya hecho referencia desde el último reemplazo



5.3.4 Algoritmos de reemplazo de páginas

- **Algoritmo del reloj**
  - Difiere del anterior sólo en la **implementación**
  - Utiliza una lista circular y un puntero a la página a considerar
  - Evita tener que mover las páginas en la lista

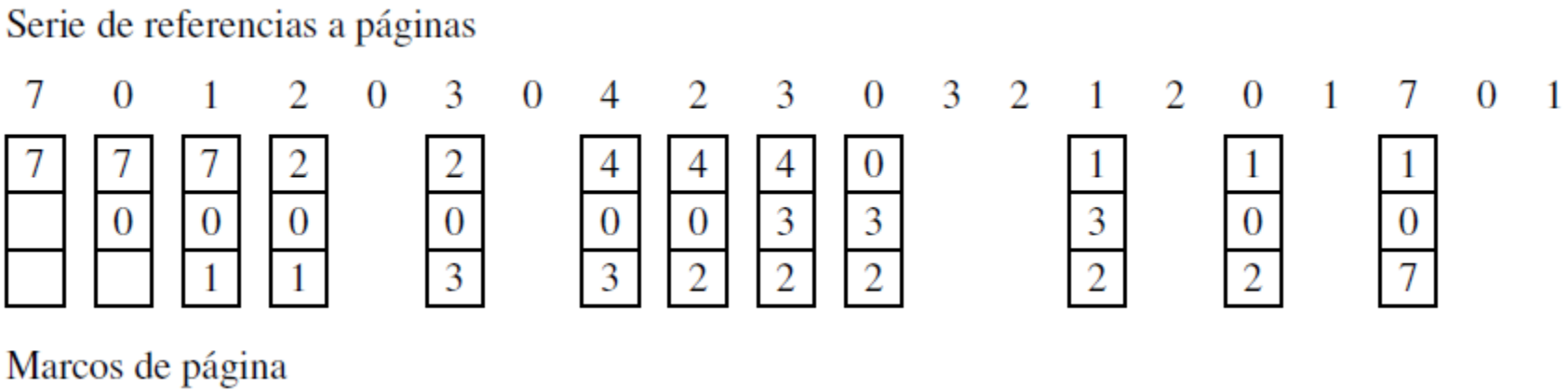


Cuando ocurre un fallo de página, se inspecciona la página a la que apunta la manecilla. La acción a realizar depende del bit R:

R = 0: Retira la página de la memoria
R = 1: Limpia R y avanza la manecilla

5.3.4 Algoritmos de reemplazo de páginas

- **Algoritmo LRU** (*Least Recently Used*)
  - Buena aproximación al algoritmo óptimo
  - Selecciona la página que no ha sido utilizada hace más tiempo



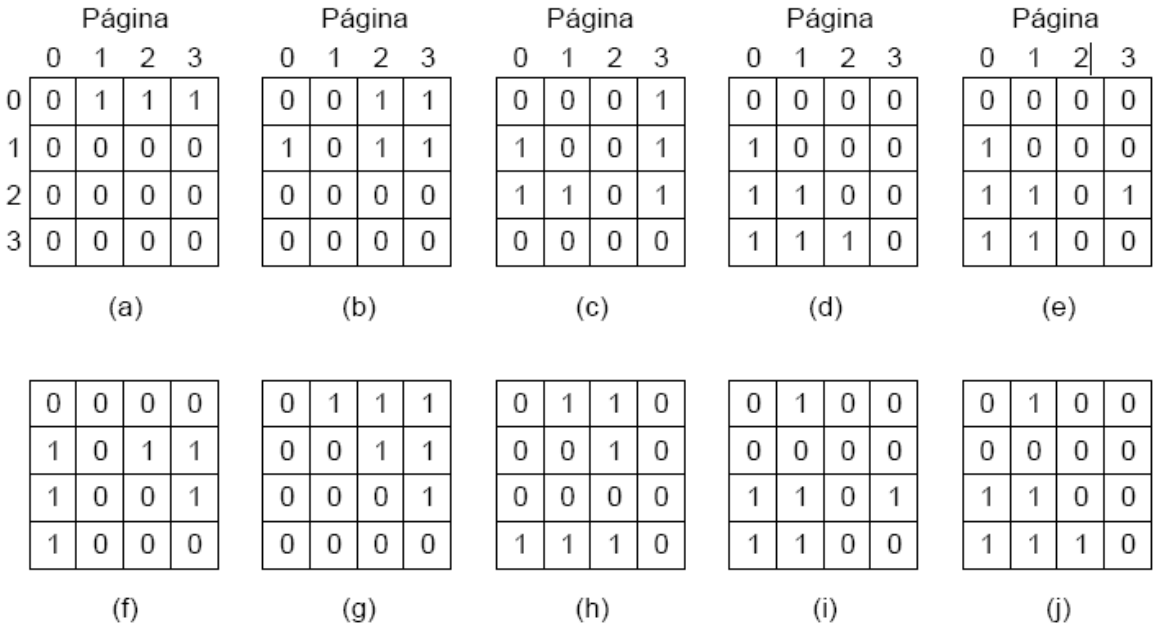
  - Problema: Es ineficiente si se implementa con una lista enlazada ⇒ actualización de la lista en cada referencia a memoria (accesos a memoria más lentos)

5.3.4 Algoritmos de reemplazo de páginas

- **LRU: Implementaciones hardware**
  - **Contador especial en *hardware* de 64 bits**
    - Después de cada referencia a memoria, el valor del contador:
      - Se incrementa de automáticamente
      - Se copia en la entrada de la página referenciada
    - Fallo de página: se sustituye la página con menor contador

5.3.4 Algoritmos de reemplazo de páginas

- **LRU: Implementaciones hardware**
  - **Matriz de *N x N* bits, con *N* igual al número de marcos**
    - Referencia un marco ⇒ su fila a 1 y su columna a 0
    - Se reemplaza el marco con menor fila (como número binario)
    - Implementación inviable en sistemas con varios GiB de RAM
    - Ejemplo con serie de referencias 0, 1, 2, 3, 2, 1, 0, 3, 2, 3

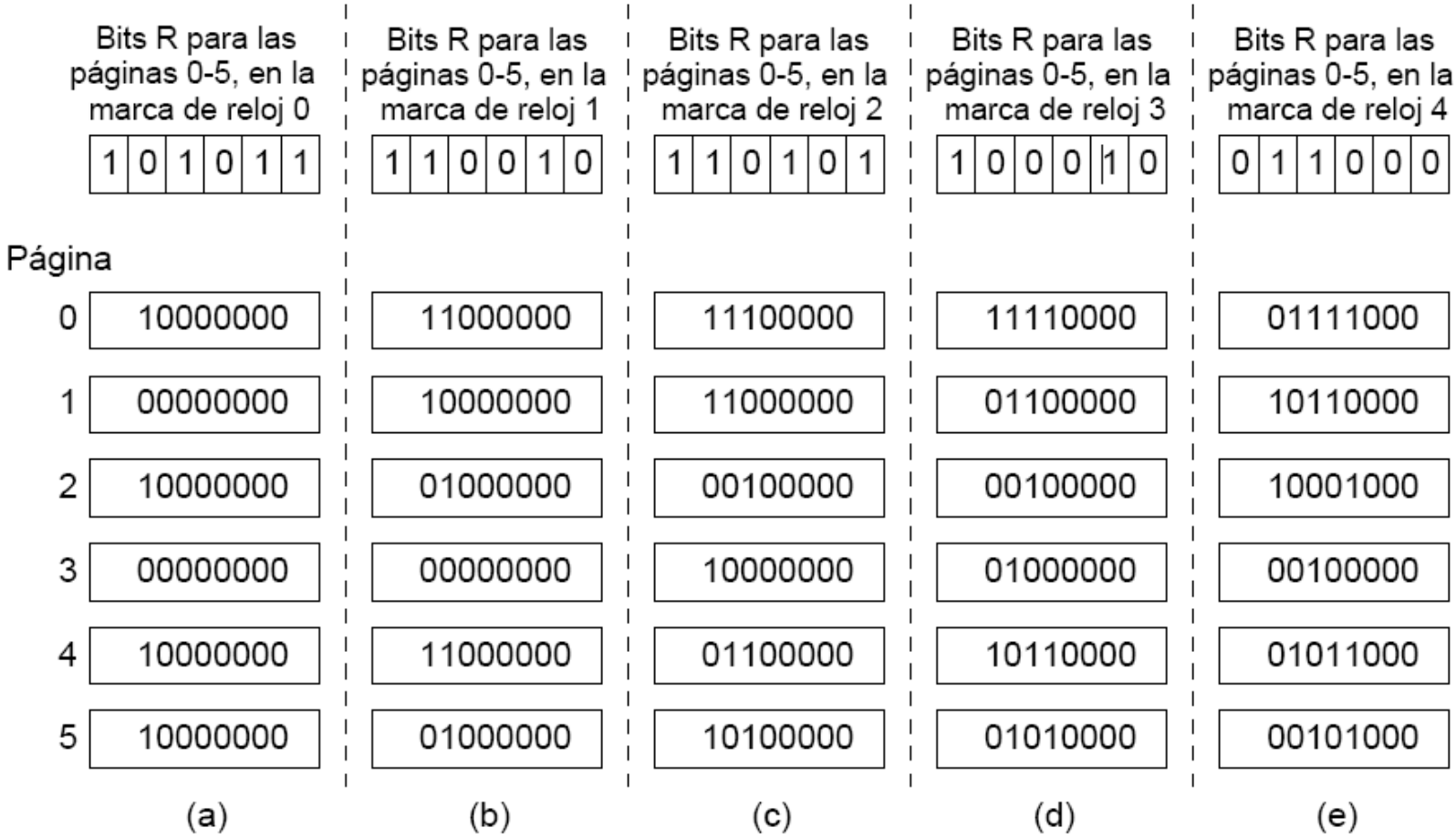


5.3.4 Algoritmos de reemplazo de páginas

- **Algoritmo de maduración o envejecimiento**
  - Simulación por software del LRU
  - Contadores software de N bits para cada página
  - Cada marca de reloj y para cada página:
    - El contador se desplaza un bit hacia la derecha
    - Se suma el valor del bit R al bit del extremo izquierdo
    - Se pone a 0 el bit R de la página
  - Fallo de página ⇒ desaloja la página con el contador más bajo
  - No tan bueno como LRU:
    - Registra sólo un bit por intervalo de tiempo ⇒ no sabe en qué instante de un intervalo de tiempo se accedió por última vez a una página (¿al principio o al final?)
    - Contadores de tamaño finito

5.3.4 Algoritmos de reemplazo de páginas

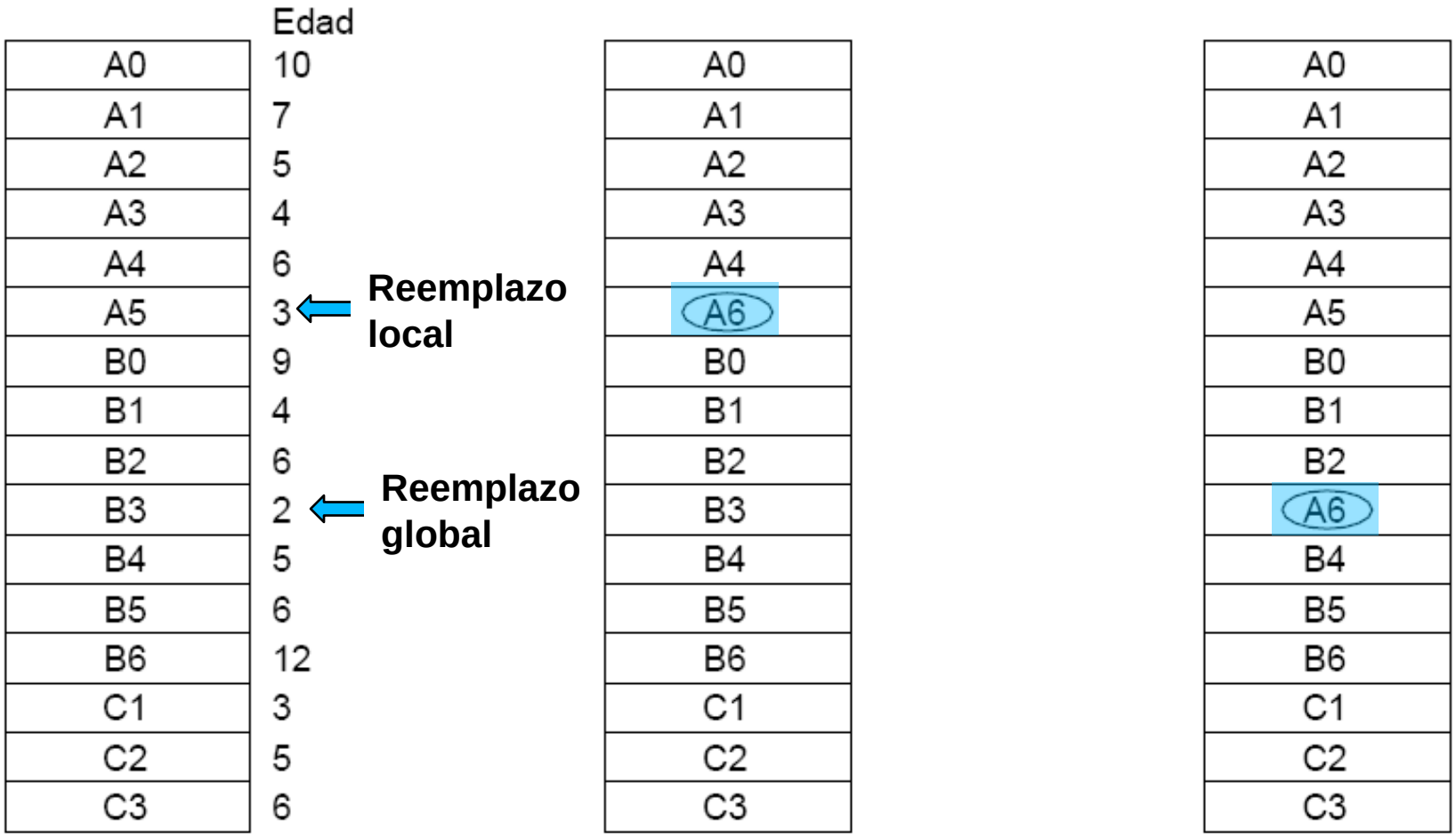
- **Algoritmo de maduración**
  - Ejemplo:



5.3.5 Cuestiones de diseño en paginación

- **Políticas de asignación y reemplazo**
  - Todos los procesos comparten los marcos de página
  - ¿Cómo se reparten los marcos de página entre los procesos?
  - **Asignación**:
    - **Estática** :nº de marcos página fijo
    - **Dinámica** : se asignan dependiendo de cómo se comporte el proceso
  - **Reemplazo:**
    - **Local** : se escoge entre los marcos asignados a ese proceso
    - **Global** : se elige un marco de entre todos

5.3.5 Cuestiones de diseño en paginación



Fallo Pag. 6 Proc. A

Reemplazo local

Reemplazo global

5.3.5 Cuestiones de diseño en paginación

	Reemplazo local	Reemplazo global
Asignación fija	El nº de marcos asignados a un proceso es fijo. La página a reemplazar se elige de entre los marcos asignados al proceso.	No es posible.
Asignación dinámica	El nº de marcos asignados a un proceso puede cambiar de un momento a otro. La página a reemplazar se elige de entre los marcos asignados al proceso.	La página a reemplazar se elige de entre todos los marcos disponibles en la memoria principal; esto hace que cambie el número de marcos asignados a cada proceso.

Tabla 5.1: Combinación de las políticas de reemplazo con las políticas de asignación.

5.3.5 Cuestiones de diseño en paginación

- **Rendimiento de las políticas de asignación y reemplazo**
  - **Interesante: reemplazo local con asignación dinámica**
    - Evita que un proceso, durante una fase con muchos fallos, quite demasiadas páginas a otros
    - Más flexible que la asignación fija: periódicamente se puede comprobar si se necesitan asignar más marcos al proceso
  - **Algoritmo de frecuencia de fallos de página** ⇒ cuándo hay que reducir o aumentar el nº de marcos asignados a un proceso
    - Calcula la tasa de fallos de página de cada proceso
    - Trata de mantener la tasa de fallos de página dentro de unos límites razonables
  - Otros aspectos:
    - Decidir el número mínimo de marcos a asignar a un proceso (viene dado por el HW)
    - Cómo realizar el reparto de marcos entre los procesos (equitativo, proporcional al tamaño…)

5.3.5 Cuestiones de diseño en paginación

- **Tamaño de página**
  - Páginas pequeñas
    - Menos fragmentación interna
    - Tablas de páginas más grandes
    - En general, más fallos de página
  - Páginas grandes
    - Más fragmentación interna
    - Tablas de páginas más pequeñas
    - En general, menos fallos de página
  - Otro factor a tener en cuenta es que las transferencias entre memoria y disco suelen ser de 1 página:
    - Velocidad de lectura/escritura de disco: no suele haber mucha diferencia entre páginas grandes y pequeñas
    - Tiempo 4 trans. 1KiB >> Tiempo 1 trans. de 4KiB
  - Tamaños típicos: 4KiB u 8KiB

5.3.5 Cuestiones de diseño en paginación

- **Hiperpaginación**
  - Un proceso da lugar a muchos fallos de página y pasa más tiempo paginando (esperando a que se resuelvan los fallos de página) que ejecutando
  - Con política de reemplazo local :
    - El proceso necesita muchos más marcos de los que tiene asignados
  - Con política de reemplazo global :
    - Un proceso que tiene muchos fallos le quita muchas páginas a otro, que sufrirá muchos fallos y empezará a quitar marcos a otro, …
  - Soluciones:
    - Ampliar memoria
    - El SO detecta situaciones de hiperpaginación y suspende temporalmente procesos para liberar memoria

5.3.5 Cuestiones de diseño en paginación

- **Políticas de lectura y escritura de páginas**
  - **Lectura de página de disco**: ¿Ante un fallo de página leemos solo esa página o algunas más?
    - Paginación por demanda: ante un fallo de página leemos solo la página que lo produce
    - Prepaginación o Paginación por adelantado: en un fallo de página se lee la página que produjo el fallo y algunas más (generalmente las que están a continuación en el espacio de direcciones virtuales)

5.3.5 Cuestiones de diseño en paginación

- **Políticas de lectura y escritura de páginas**
  - **Escritura de página a disco**: ¿las escrituras de las páginas modificadas las hacemos cuando se expulsan o periódicamente vamos limpiando?
    - Escritura por demanda: cuando se expulsa una página que está modificada (incrementa el tiempo necesario para resolver muchos fallos de página: lectura + escritura disco)
    - Escritura por anticipado: por ejemplo, cada X segundos (**demonio de paginación**) y va apagando el bit M (se aceleran muchos fallos de página y se escriben varias páginas a la vez)
      - Si además se liberan páginas ⇒ **Caché de páginas**

      (tenemos un conjunto de marcos a utilizar, pero además un proceso puede "repescar" un página que esté en caché)

Memoria virtual con segmentación

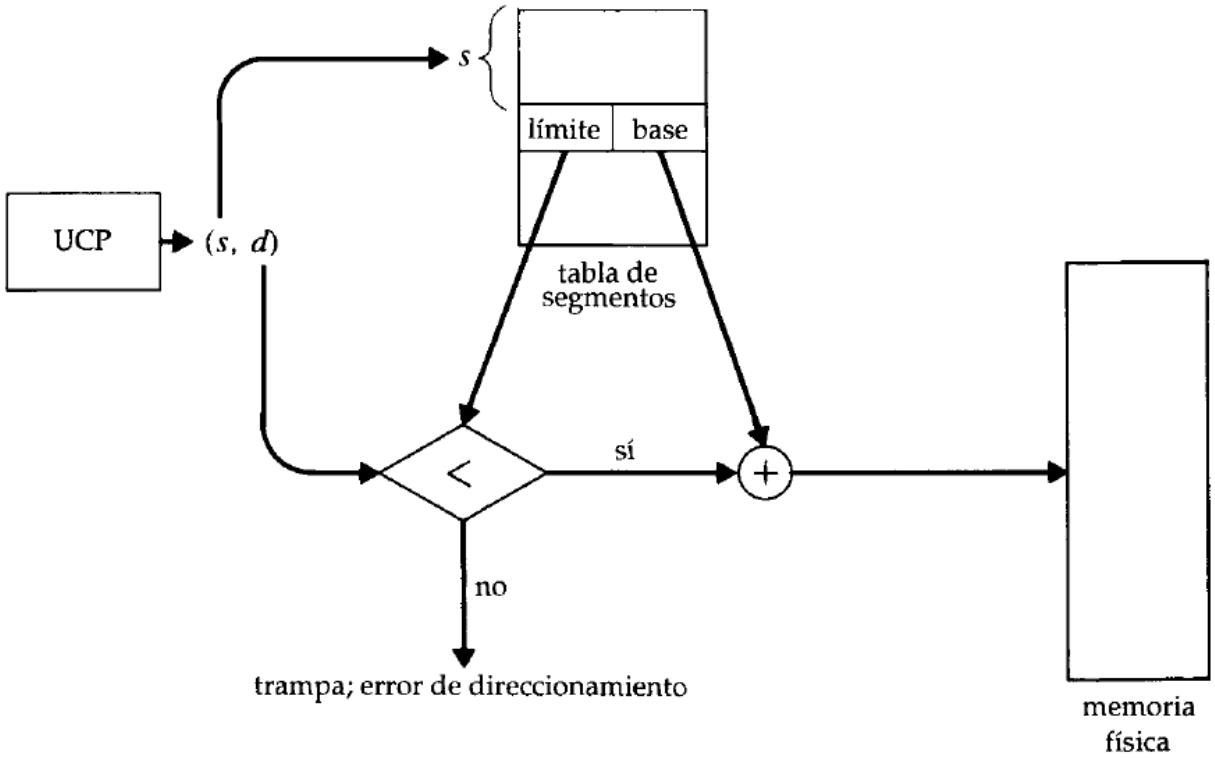
- 5.4 Segmentación pura
- 5.5 Segmentación paginada

5.4 Segmentación pura

Segmentación

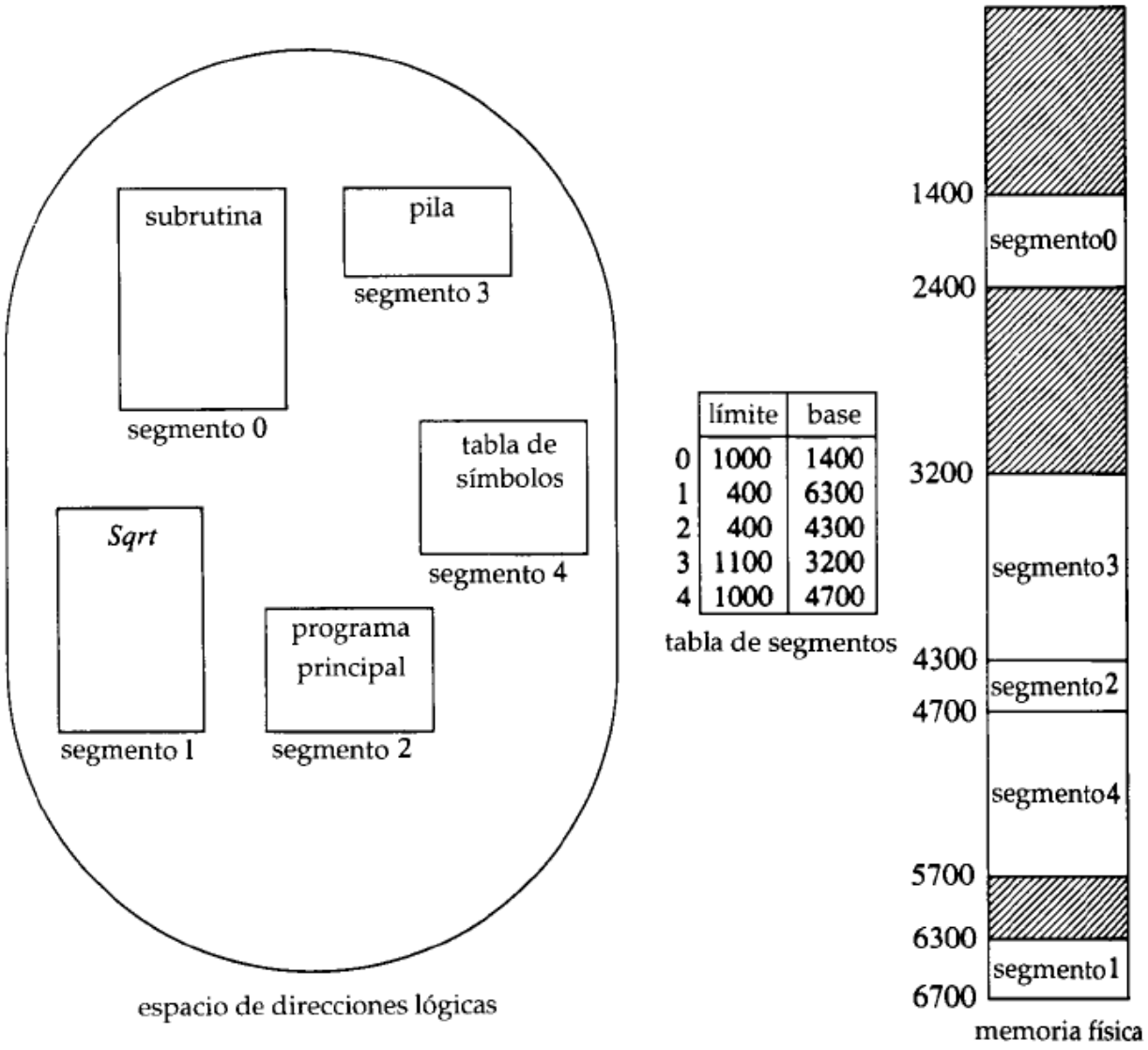
- Trata de aproximarse a la visión del usuario de la memoria
  - Conjunto de segmentos de tamaño variable y sin orden especial (arrays, tablas, pilas, funciones, …)
- Espacio de direcciones lógicas de un proceso formado por un conjunto de segmentos, cada uno tiene una posición de memoria principal y un tamaño
- Cada segmento:
  - Una sucesión lineal de direcciones, desde 0 hasta un máximo
  - Tiene una longitud distinta y variable
  - Puede crecer o disminuir independientemente
  - Puede tener una protección diferente (lectura, ejecución,. . .)
- Direcciones generadas por los procesos de dos dimensiones ⇒ (Nº de Segmento, Desplazamiento dentro del segmento)
- Tabla de segmentos: hace corresponder direcciones bidimensionales con direcciones físicas

5.4 Segmentación pura



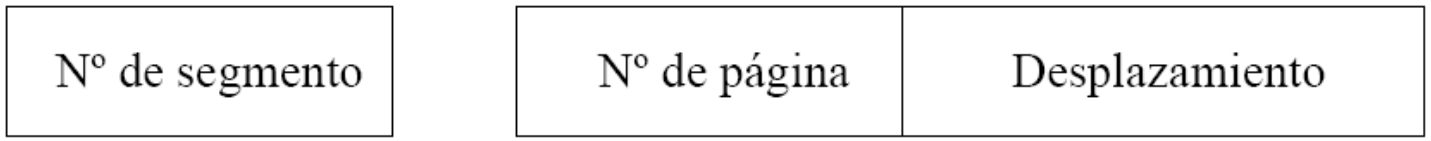
- Facilita la protección y la compartición de objetos (procedimientos, funciones, estructuras de datos, etc.) de forma individual
- Problema: **Fragmentación externa**

5.4 Segmentación pura

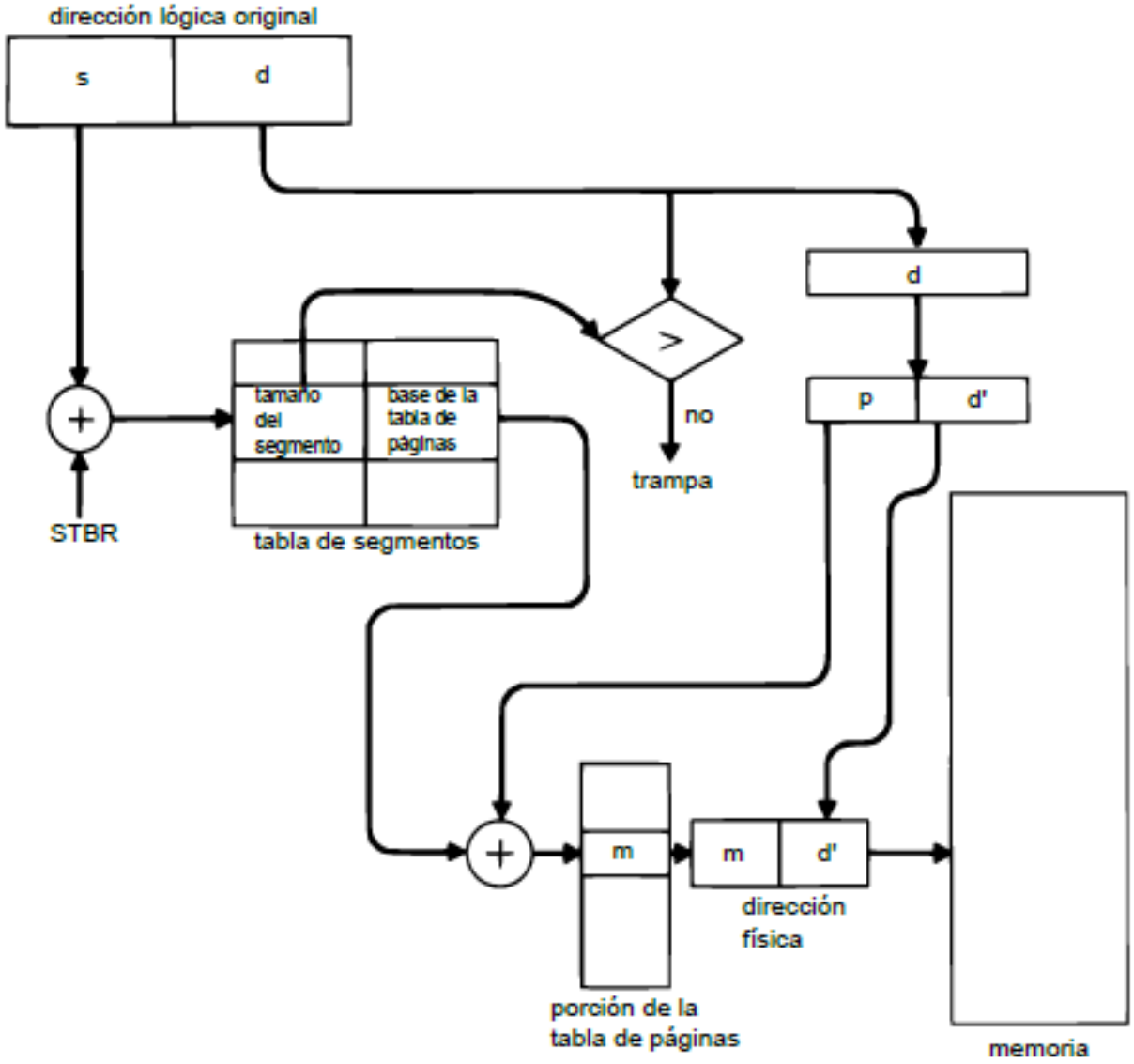


5.5 Segmentación paginada

- Cada segmento se pagina internamente
  - ⇒ ventajas de la paginación y de la segmentación
- Dirección formada por dos partes:
  - **Segmento** ⇒ entrada en la tabla de segmentos que nos da la dirección de la tabla de páginas
  - **Dirección dentro del segmento** ⇒ dividida en 2 partes:
    - **Número de página** ⇒ entrada de la tabla de páginas que da el marco de página correspondiente
    - **Desplazamiento** dentro de la página
- **Cada segmento** tiene asociada **una tabla de páginas**
  - El tamaño (nº de entradas) de la TP de un segmento depende del tamaño del segmento



5.5 Segmentación paginada



5.5 Segmentación paginada