Sistemas Operativos
Tema 8. Gestión de memoria

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

Contenidos

- Antecedentes
- Técnicas básicas: recubrimientos, intercambio
- Gestión de memoria contigua
- Segmentación
- Paginación
- Técnicas mixtas

Gestión de la memoria
Antecedentes

- La memoria física es un conjunto de celdas referenciables por medio de una dirección lineal (p.ej. de la 00000h a la FFFFFh)
- Para que un programa se ejecute, su código y sus datos necesitan estar cargados en memoria (al menos en parte)
- En un sistema multitarea, la memoria ha de repartirse entre los diferentes procesos

Gestión de la memoria
Antecedentes (2)

- Las rutinas del sistema operativo también deberán residir en memoria, en todo o en parte
- Puede ser que la memoria principal no tenga capacidad suficiente para todos los procesos en ejecución

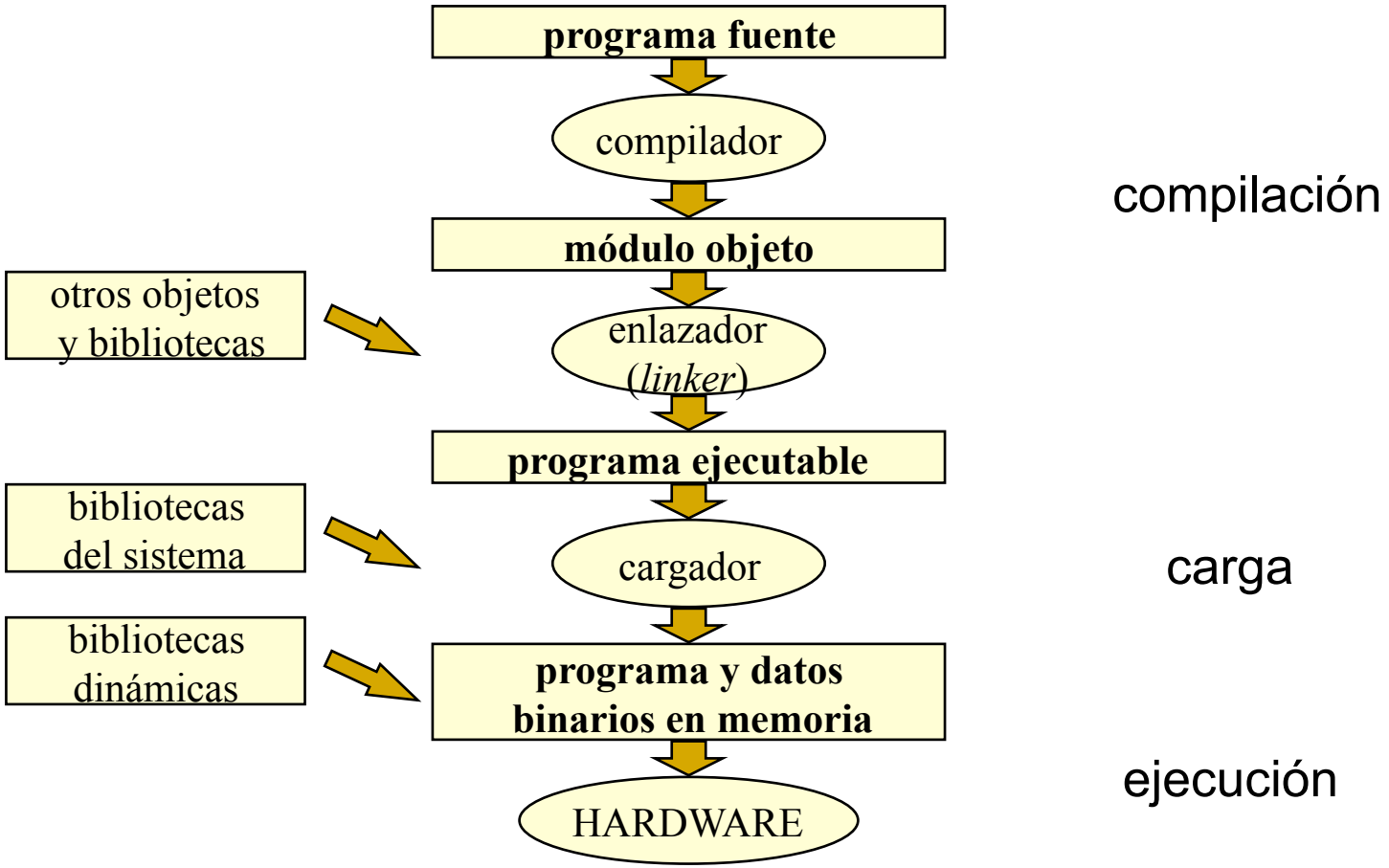
Gestión de la memoria
Objetivo principal

- Conseguir que varios procesos puedan ejecutarse de forma concurrente,
  - evitando los conflictos de uso
  - protegiendo al sistema operativo
  - aprovechando eficazmente el espacio disponible:
    - Minimizar la memoria desaprovechada
      - Evitar fragmentación
      - Memoria ocupada por varias copias de un mismo objeto
      - Memoria ocupada por las estructuras de datos necesarias para la operación del gestor de memoria
    - Carga parcial de programas
  - sin perjudicar el rendimiento:
    - Complejidad temporal
    - Tiempo de acceso a memoria

Gestión de la memoria
Objetivo principal (2)

- Un gestor de memoria ideal debería por tanto,
  - minimizar la memoria desaprovechada
  - tener una complejidad temporal mínima
  - y presentar un recargo por acceso a memoria mínimo
  - además de proporcionar una buena protección y una compartición flexible

Ciclo de vida de un programa



Conversión de direcciones: reubicación

- El compilador traduce direcciones de memoria *simbólicas* a direcciones binarias.
- Si las direcciones binarias son absolutas, el programa sólo se puede ejecutar en una zona fija de la memoria: **NO ES REUBICABLE.**
- Ej.: los programas con formato .COM de MS-DOS

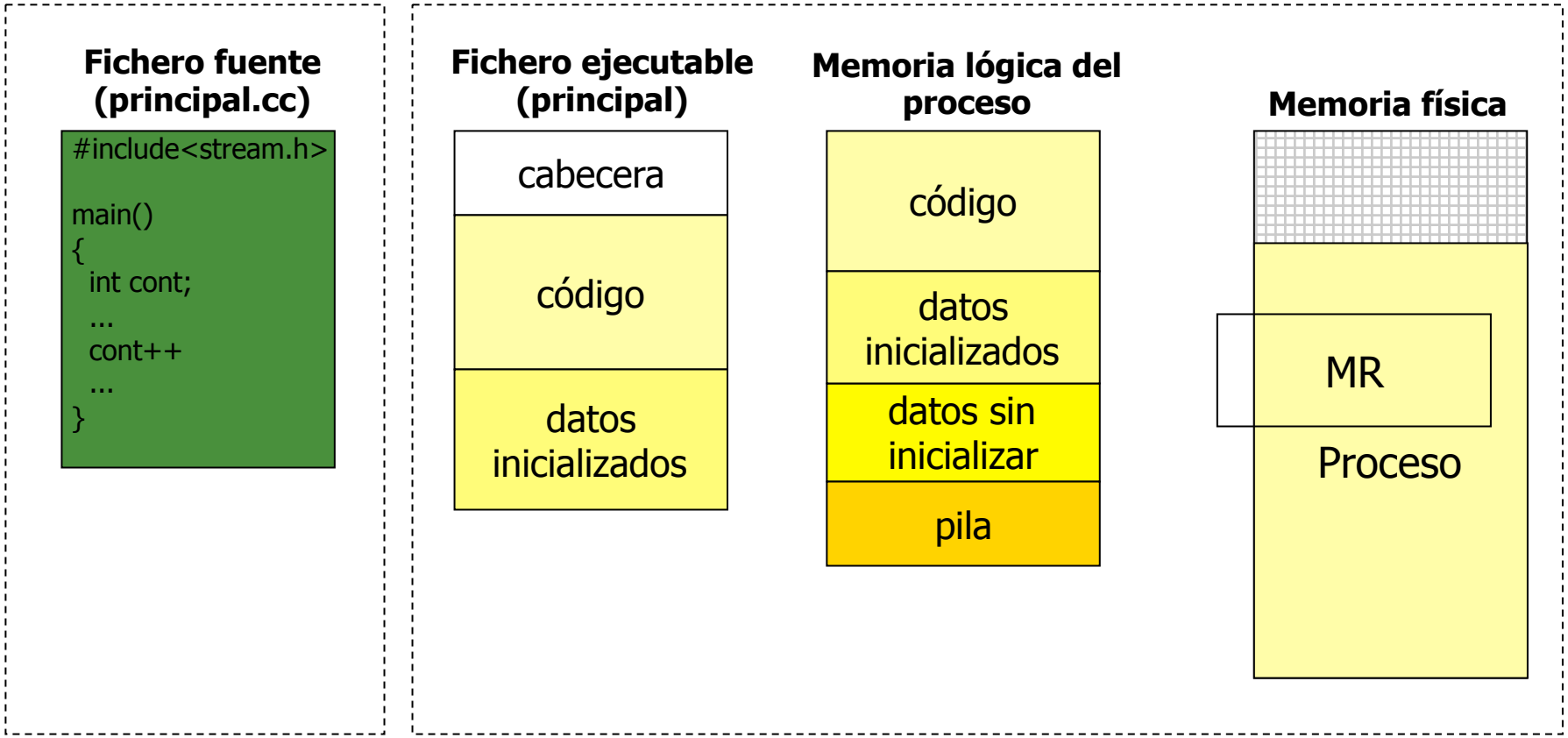
Conversión de direcciones: reubicación (2)

- Nos interesa que el compilador no genere direcciones definitivas, sino direcciones provisionales, **reubicables**.
  Cuando se sepa dónde van a residir el código y los datos, se convertirán a direcciones absolutas.
- ¿ En qué momento (etapa) se realiza esta reubicación ?
  - Carga (enlazador o cargador) → Reubicación estática
  - Ejecución (*hardware*) → Reubicación dinámica

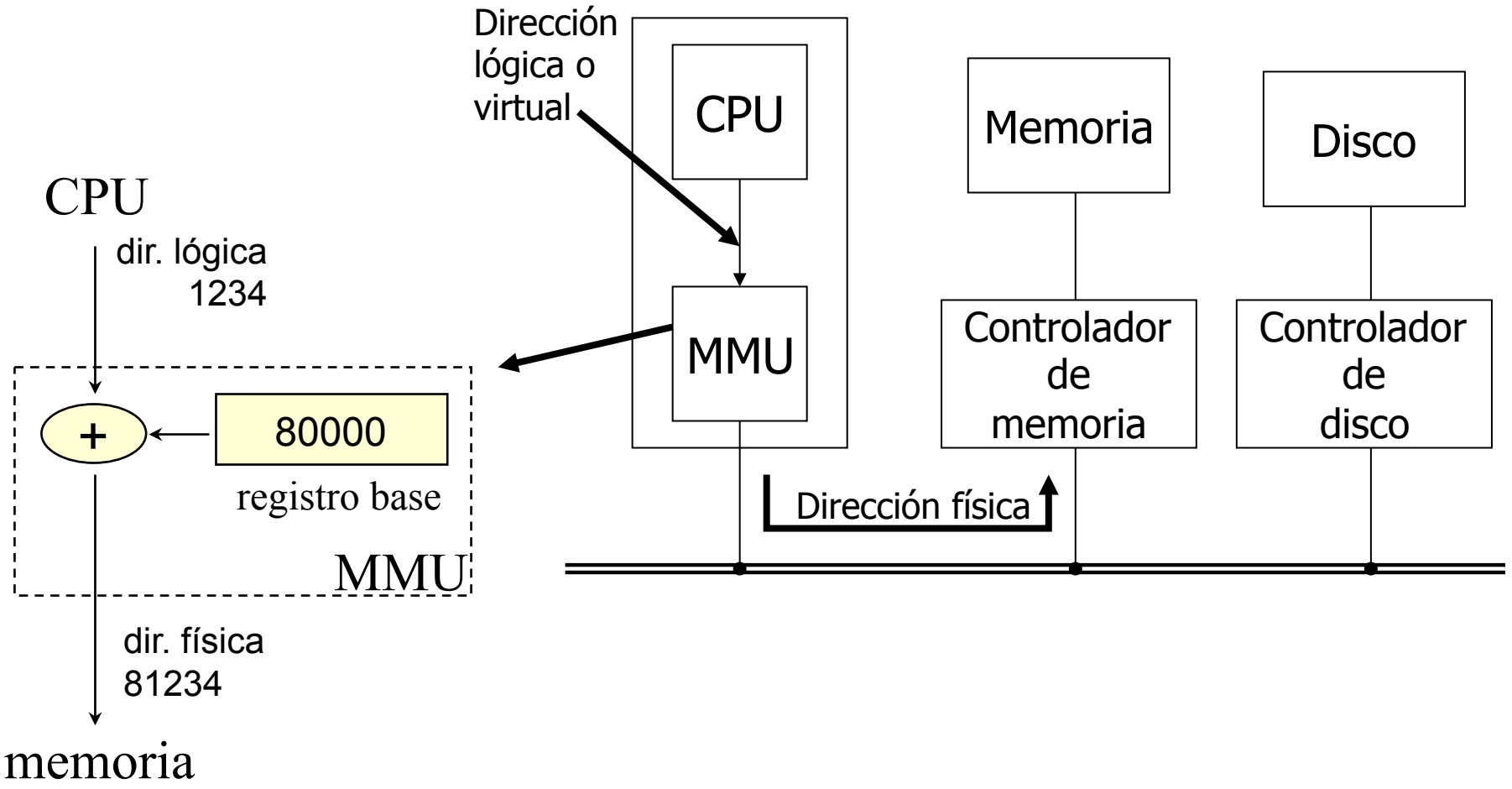
Reubicación dinámica: direcciones lógicas/direcciones físicas

- **Dirección física:** la que llega al *chip* de memoria
- **Dirección lógica o virtual:** la generada por la CPU
- El dispositivo que traduce direcciones virtuales a físicas se llama **unidad de manejo de memoria** (MMU, en inglés)
- El espacio de direcciones lógicas y el espacio de direcciones físicas no tienen por qué coincidir
- Ejemplo: registro base

Espacios de direcciones



Direcciones lógicas/direcciones físicas



Carga dinámica

- Proceso se ejecute
  - Código + Datos → Memoria física
- Consecuencia:
  - Tamaño de un proceso limitado al tamaño de la memoria física
- Carga dinámica
  - postergar la carga en memoria de un módulo hasta que el programa llame a alguna rutina del mismo

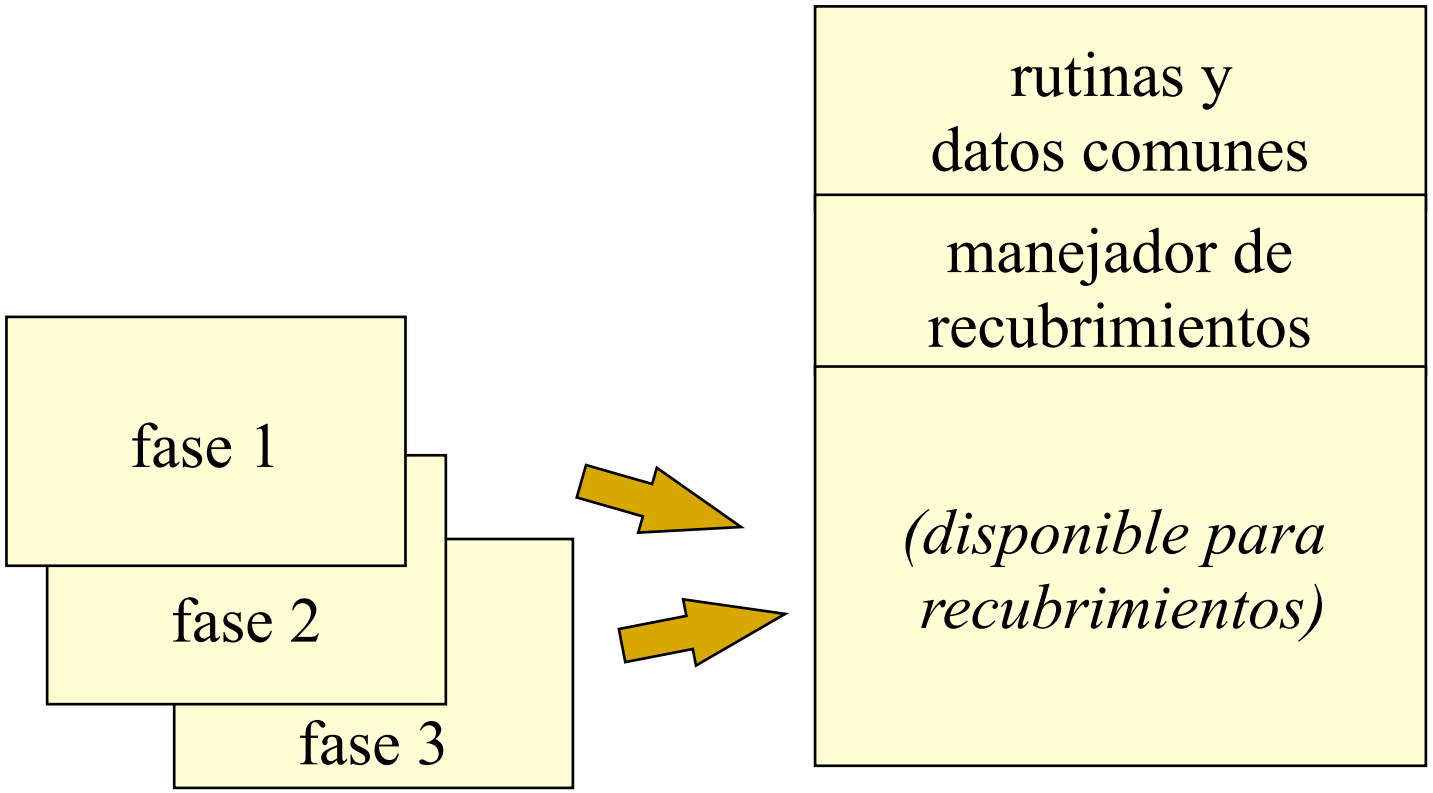
Enlace dinámico

- Similar a la carga dinámica, pero efectuando el *enlace* en tiempo de ejecución: bibliotecas dinámicas (DLL)
- La DLL se carga en memoria cuando algún proceso llama a una de sus rutinas. Las llamadas a sus funciones se efectúan a través de una tabla de punteros.
- Si varios procesos emplean la biblioteca dinámica, sólo se mantiene una copia de ella en memoria.
- Ejemplos de enlace dinámico:
  - UNIX: *shared libraries (shlib)*
  - Windows: *dynamic load libraries (dll*)

Recubrimientos (*overlays*)

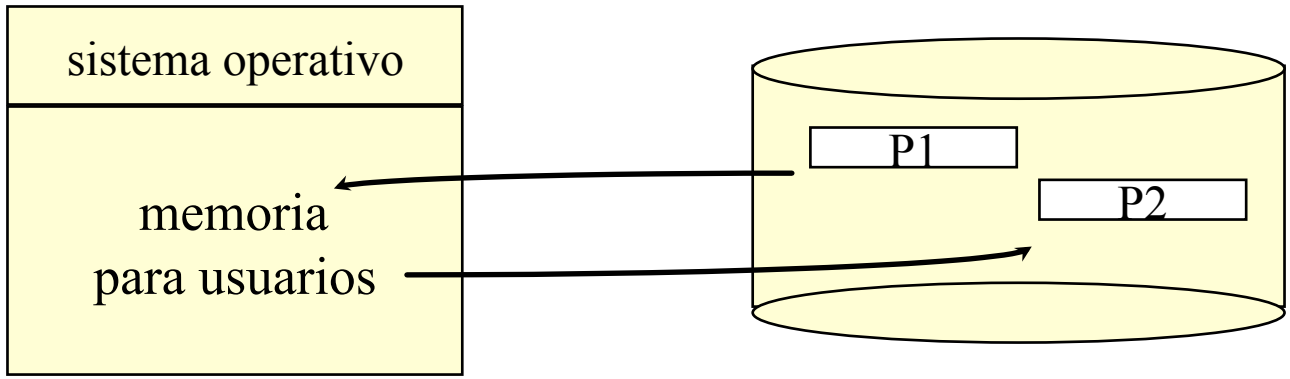
- Muchos programas no necesitan todo el código al mismo tiempo, sino que se ejecutan por fases (ej. un compilador)
- El programa se descompone en módulos separados (recubrimientos), que se cargan en un área de memoria al efecto
- Si se carga un recubrimiento, borra al que se encontraba ya cargado
- El programa de usuario es responsable de cargar recubrimientos según se necesiten

Recubrimientos (*overlays*) (2)



Intercambio (*swapping*)

- **Objetivo:** cuando un proceso queda bloqueado o en espera, la memoria que ocupa podría desasignársele.
- **Intercambio:** Cuando un proceso pierde la CPU, se vuelca su imagen de la memoria al disco (*swap out*). Cuando se decide reanudar el proceso, se recupera su imagen del disco (*swap in*)



Intercambio (*swapping*) (2)

- **Problemas:**
  - aumenta el tiempo de cambio de contexto
  - E/S que accede por DMA
- **Mejoras:**
  - varios procesos en memoria
  - intercambio un proceso mientras se ejecuta otro

Intercambio (*swapping*) (3)

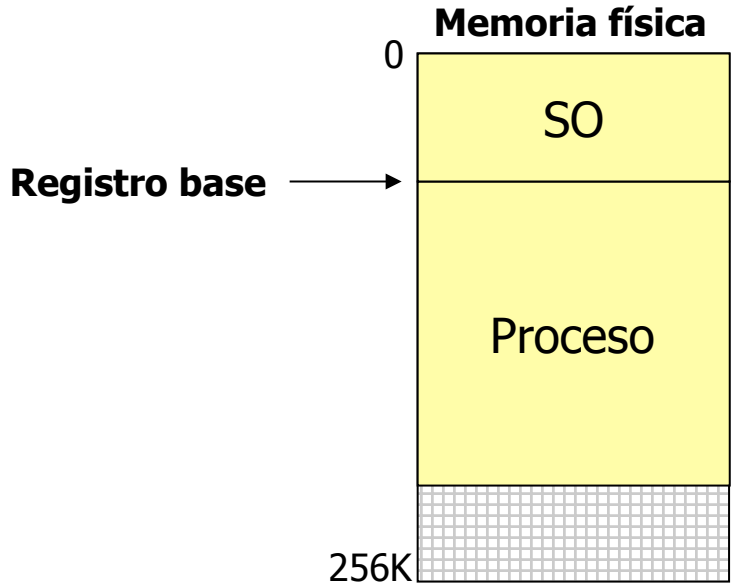
- ¿Qué se necesita para llevarlo a cabo?
  - Proceso intercambiador (efectúa la mayoría de las funciones del PMP)
  - Criterios para la elección de la víctima (política de *swapping out*)
  - Criterios para elegir los procesos que entran (política de *swapping in*)
  - Espacio en disco donde almacenar las imágenes de los procesos
    - Area específica para el intercambio (área de *swap*)
    - Ficheros de intercambio
  - Criterios para la gestión del espacio de intercambio (política de gestión del área de *swap*)

Intercambio (*swapping*) (4)

- Otras consideraciones:
  - ¿ Operaciones de E/S pendientes ?
  - Un proceso intercambiado, ¿ regresa al mismo espacio de memoria que ocupó previamente ?
  - Versiones modificadas del intercambio estándar:
    - Unix
    - Microsoft Windows 3.1

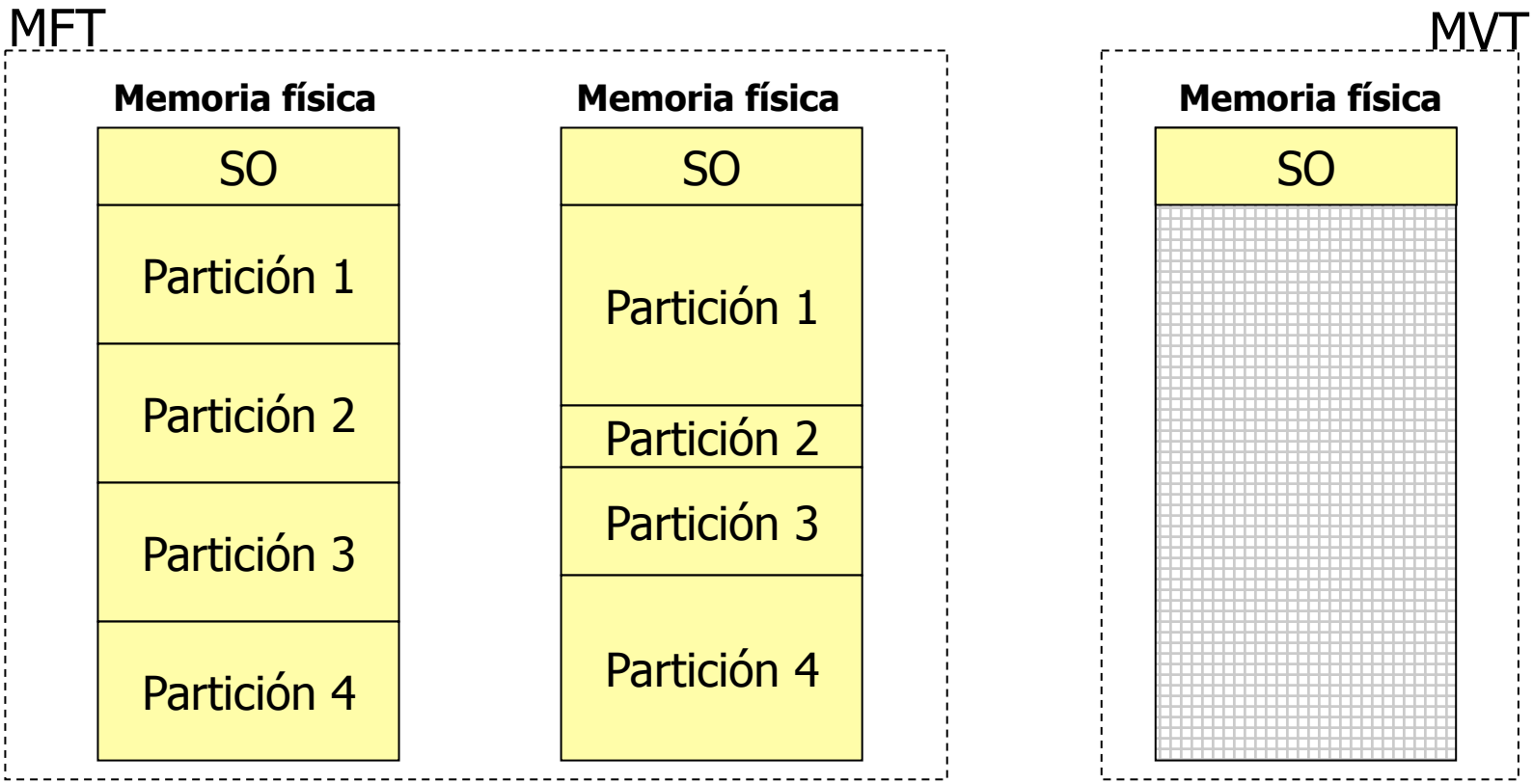
Gestión de memoria contigua

- Caso trivial: una sola partición (sistema operativo+área de usuario)
  - como mucho, utilizar un registro base para proteger al S.O.



Gestión de memoria contigua

- Evolución: **particiones múltiples**
  - particiones de tamaño fijo (MFT)
  - particiones de tamaño variable (MVT)



Memoria contigua: estructuras de datos

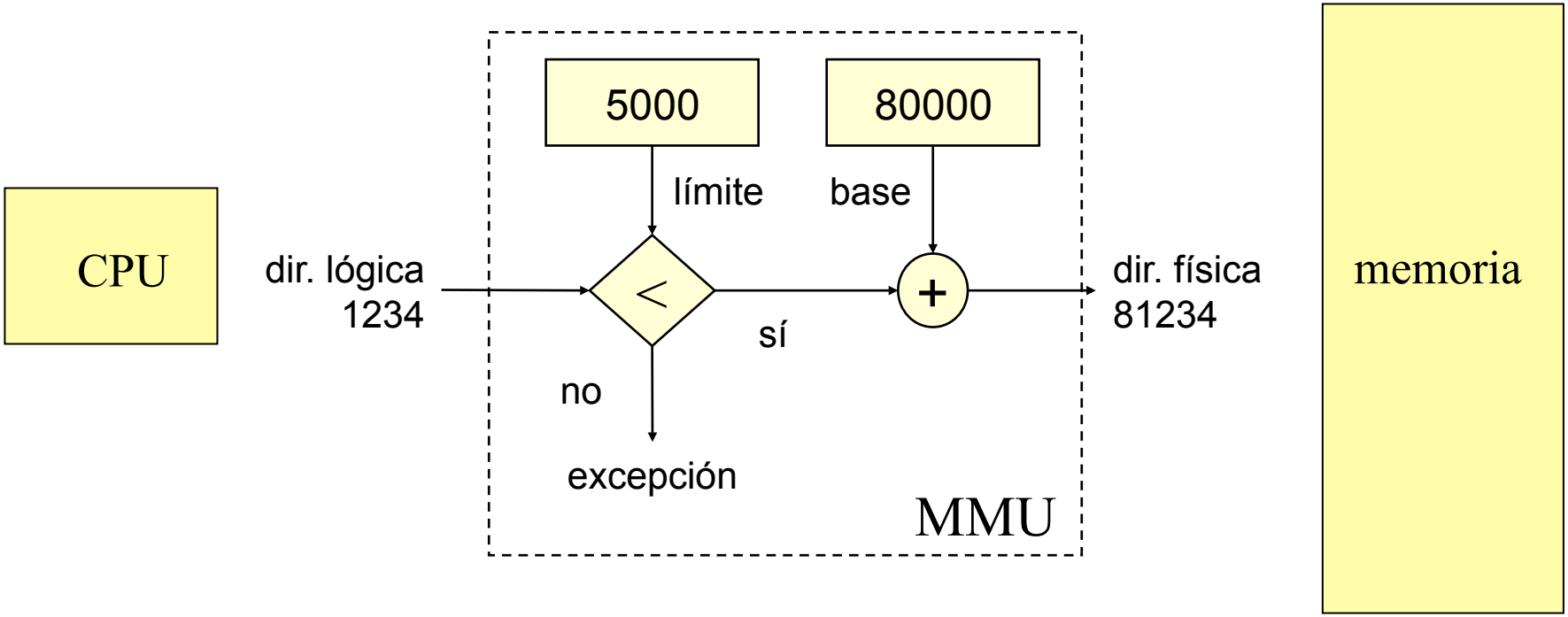
- Mecanismos de gestión de la memoria contigua
  - **Tabla de descripción de particiones** (TDP)
  - El S.O. gestionará una **lista de huecos libres** en memoria y seleccionará qué procesos pueden cargarse en memoria para ejecutarse
  - Primitivas internas de pedir y liberar memoria

Memoria contigua: cómo reservar espacio a un proceso

- **POLÍTICAS DE UBICACIÓN**
  - El problema general se conoce como la **gestión de memoria dinámica**
    - Primer hueco (*first-fit*)
    - Mejor hueco (*best-fit*)
    - Peor hueco (*worst-fit*)
  - *las políticas de "primer hueco" y "mejor hueco" son similares en rendimiento; y mejores que la de "peor hueco"*
  - Otras políticas: siguiente ajuste, *buddies*

Memoria contigua: protección

- Pareja de registros base y límite



Memoria contigua: protección

- Registrar los derechos de acceso en la propia memoria
  - A cada dirección se le añade un número de bits para identificar al propietario
    - Problema: costoso
    - Mejora: asociar estos bits a bloques de memoria física
  - Comprobación: tiempo de ejecución
  - SO → Clave "maestra" única que le da acceso sin restricciones a todos los bloques de memoria

Compartir zonas de memoria

- Confiar los objetos compartidos al SO
  - Forma de acceder a estos objetos: llamadas al sistema
  - Inconvenientes:
    - SO grande y monolítico, lo que hace más difícil su desarrollo, mantenimiento y verificación
    - No se podrían incorporar dinámicamente nuevos objetos (sólo permite la inclusión de nuevos objetos durante la generación del sistema)

Compartir zonas de memoria (2)

- Mantener múltiples copias de los objetos compartidos
  - Redundancia → las modificaciones deben ser propagadas a todas las copias restantes (SO sería el encargado cada vez que se realice un cambio de contexto)
  - Si se soporta intercambio, OJO, podrían existir copias de los objetos compartidos en disco

Compartición

- Utilizar particiones de memoria compartidas (comunes)
- ¿Protección?
  - Registros base/límite: requeriría conjuntos separados de pares de registros base/limite dedicados para acceder a los espacio de memoria privado y compartido
    - Implica la existencia de algún mecanismo que indique en cada acceso que conjunto de registros emplear

Compartición

- Claves de protección: requeriría controlar los bloques compartidos para que en los conmutaciones de contexto se cambien las claves de protección
  - Necesidad de llevar la cuenta de que bloques están siendo compartidos y por quién
- MVT => Permite que particiones adyacentes en memoria física puedan solaparse
  - Compartición de datos y código (dos procesos)

La fragmentación de la memoria

- Es el gran problema de la memoria contigua
  - Fragmentación interna (MFT)
  - Fragmentación externa (MVT)

Fragmentación: soluciones

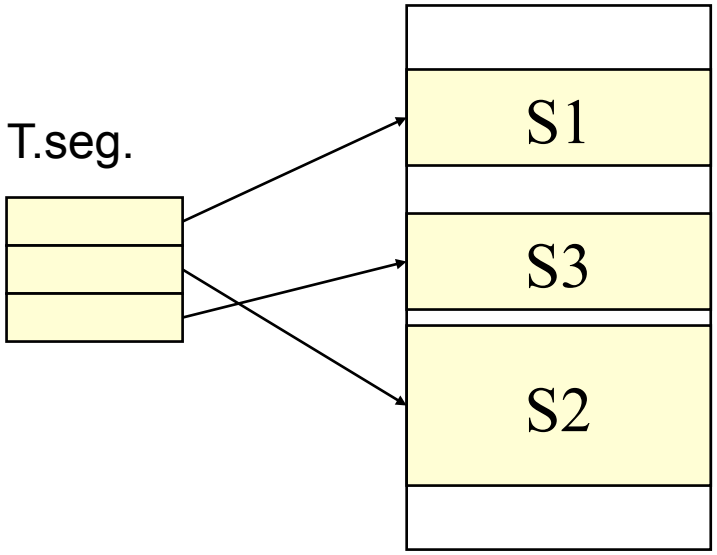
- Permitir que un proceso puede ubicarse en zonas de memoria no necesariamente contiguas
- Y si no se puede lo anterior → **compactación**
- ¿Es posible siempre?
  - Si la reubicación es estática y se efectúa durante el ensamblado o la carga, no puede haber compactación
  - Esta sólo es posible si la reubicación es dinámica y se efectúa en tiempo de ejecución
    - Sólo requiere mover el programa y los datos y modificar el registro base para que refleje la nueva dirección base

Compactación

- Estrategia a seguir
  - Seleccionar una estrategia óptima de traslados
    - Raras veces es implementado debido al gasto en que se incurre al evaluar las opciones
  - Reubicar todas las particiones en un extremo de la memoria
    - El recargo por copia es generalmente más alto que el de un traslado más selectivo
  - Compactación + intercambio
    - Copiar en disco los procesos que tienen que cambiar de lugar

Segmentación

- Un programa se puede descomponer en varios **segmentos** de memoria (código, datos, pila...)
- Con el *hardware* adecuado, podemos ubicar esos segmentos en zonas de memoria no contiguas.



Segmentación (2)

- El compilador tiene que generar código que haga referencias a direcciones de memoria con dos componentes: <segmento,desplazamiento>
- El S.O. ubica cada segmento en un hueco contiguo de memoria
- El *hardware* se encarga de la reubicación dinámica mediante una **tabla de segmentos**

Hardware de segmentación

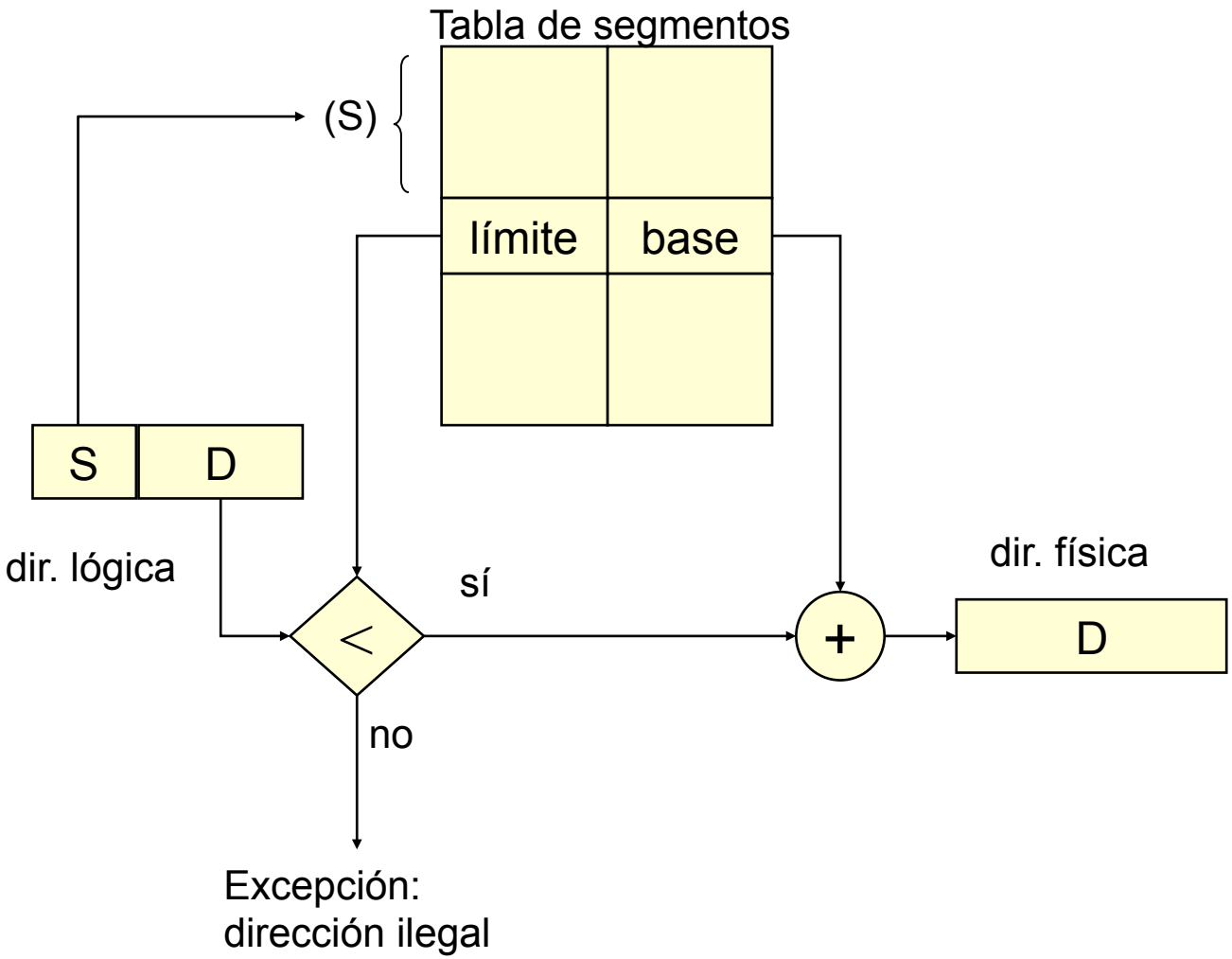


Tabla de segmentos

- Registros
  - Problema: muchos segmentos
- Memoria
  - Registro base de la tabla de segmentos (*RBTS*)
  - Registro de longitud de la tabla de segmentos (*RLTS*)

Tabla de segmentos

- Proceso de traducción: dirección lógica (s,d)
  - Se comprueba que s<RLTS
  - Se calcula la dirección de la entrada de la tabla de segmentos (RBTS+s) y se lee dicha entrada
  - Se coteja el desplazamiento con la longitud del segmento
  - Se calcula la dirección física del byte deseado como la suma de la base del segmento y el desplazamiento
- Proceso de traducción: requiere dos referencias a memoria por cada dirección lógica
- Solución: usar un conjunto de registros asociativos para guardar las entradas de la tabla de segmentos que se usaron mas recientemente

Segmentación: ventajas

- Atenúa el problema de la fragmentación
- Permite definir protecciones selectivamente
- Permite compartición de zonas de memoria de forma eficaz
- Todo ello sin añadir complejidad a los algoritmos de gestión de espacio

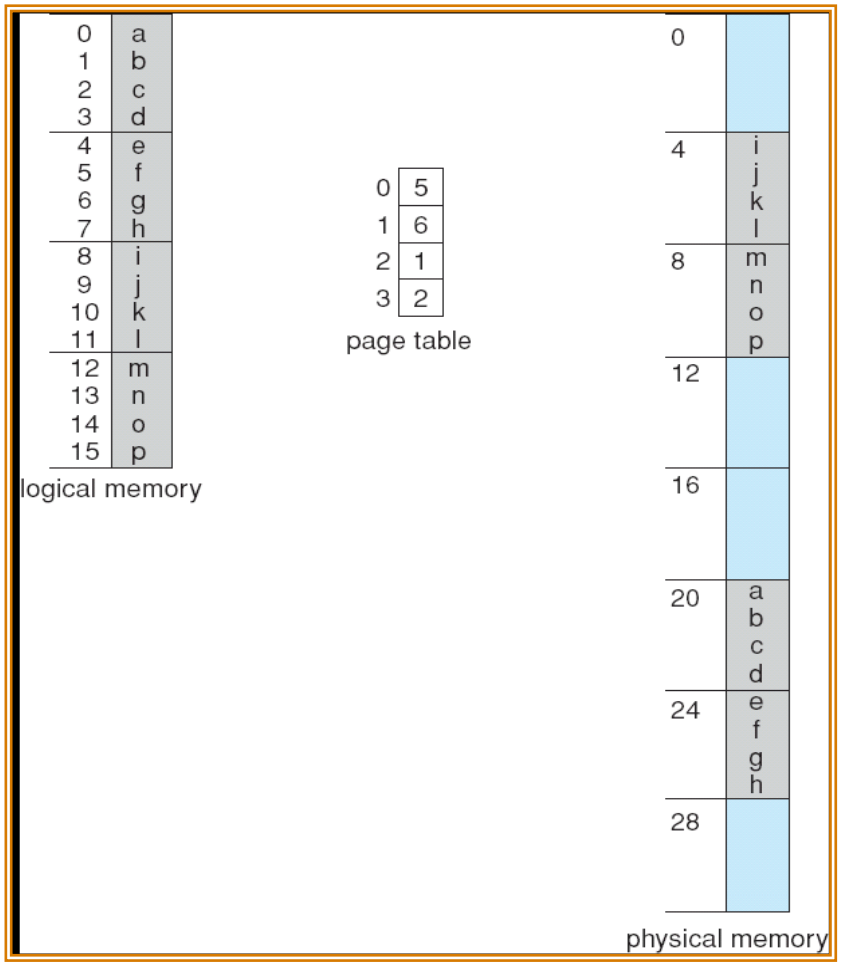
Segmentación: inconvenientes

- El compilador/enlazador debe reconocer un espacio segmentado (desventaja leve)
- Necesita soporte del *hardware*
- Incurre en un acceso adicional a memoria (para la tabla de segmentos)
- No soluciona del todo los problemas de las técnicas de ubicación contigua (fragmentación)

Paginación

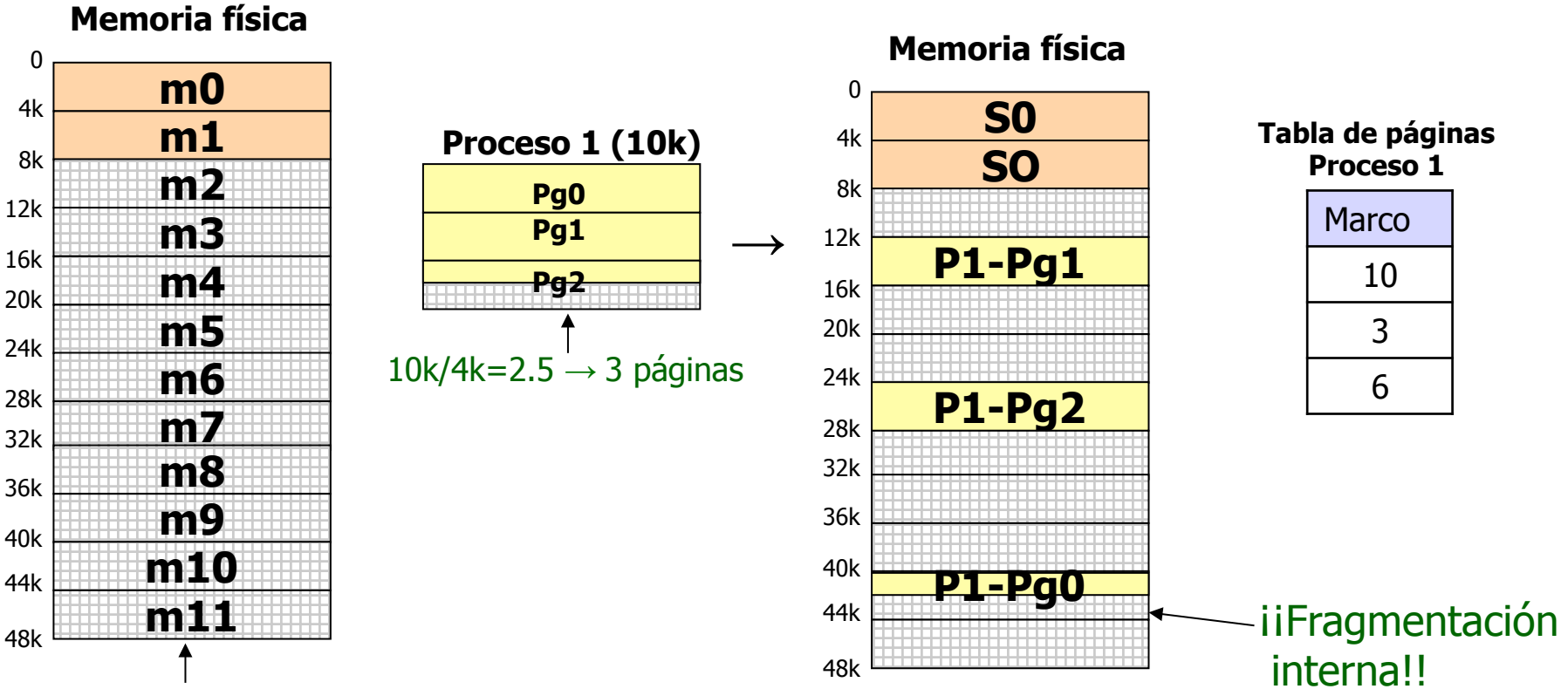
- Técnica que soluciona la fragmentación externa
- **La idea:** Trocear la memoria disponible en **páginas** de tamaño fijo (ej. 4Kb). Un programa puede residir en varias páginas no contiguas
- Las páginas disponibles en memoria se llaman **marcos de página** (*page frames*).
- Toda dirección lógica se descompone en dos partes: número de página y desplazamiento.
- La MMU se encarga de asociar el número de página lógico con el marco de página asignado. Para ello emplea una **tabla de páginas.**

Paginación/Ejemplo



Ejemplo de paginación

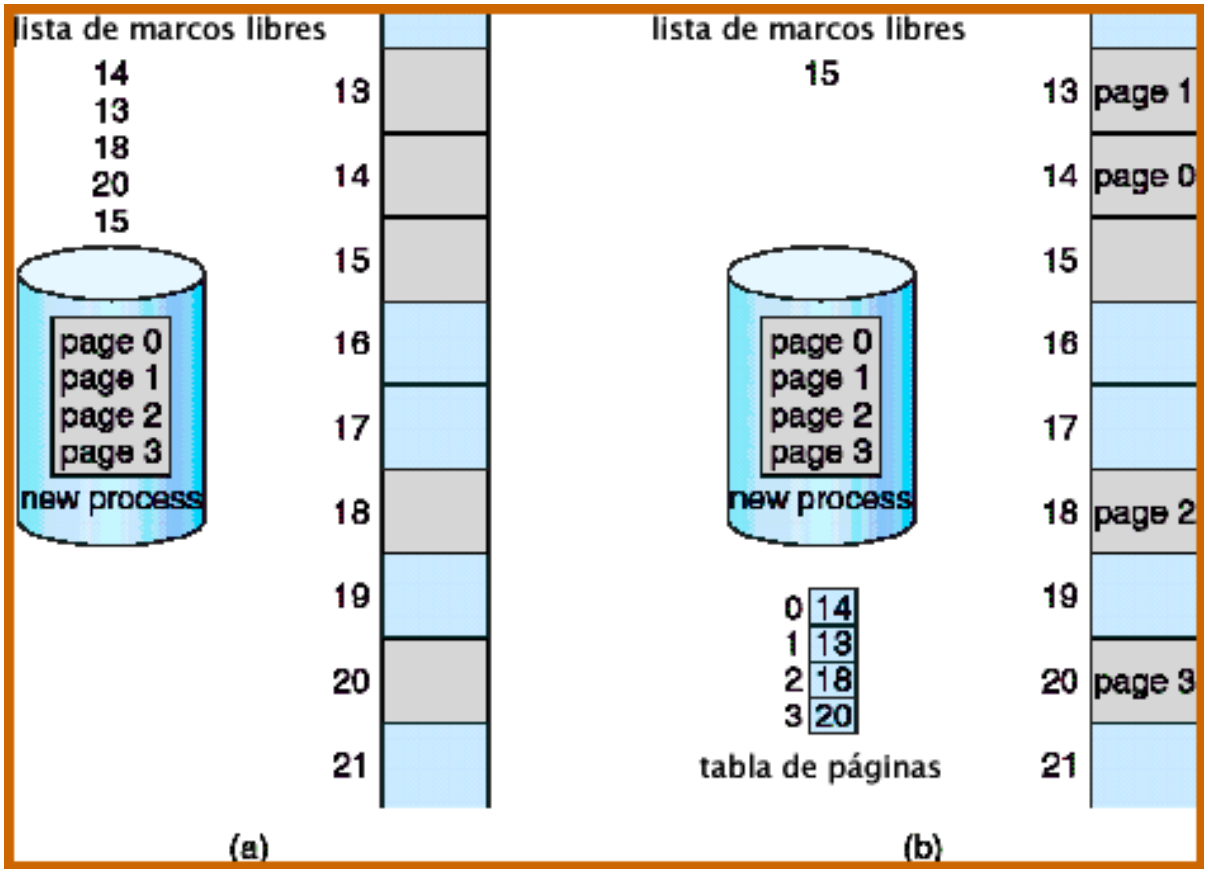
Ubicación de un proceso en memoria y creación de la tabla de páginas: tamaño de página 4kbytes, tamaño de la memoria 48kbytes



Paginación: gestión del espacio libre

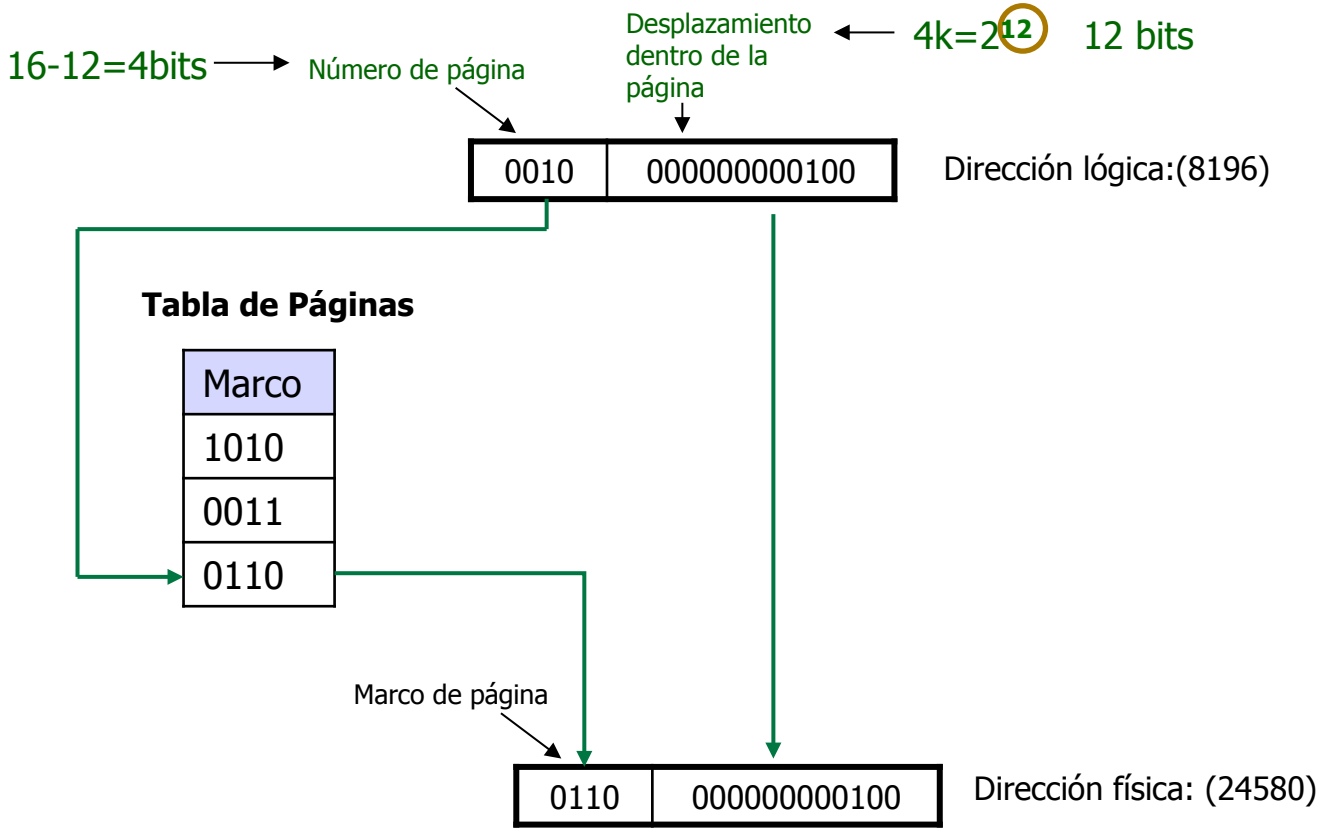
- La gestión del espacio libre consiste simplemente en saber qué marcos están libres
- El SO posee una **tabla de marcos de páginas** (TMP)
  - ¿Implementación?

Paginación/Marcos Libres

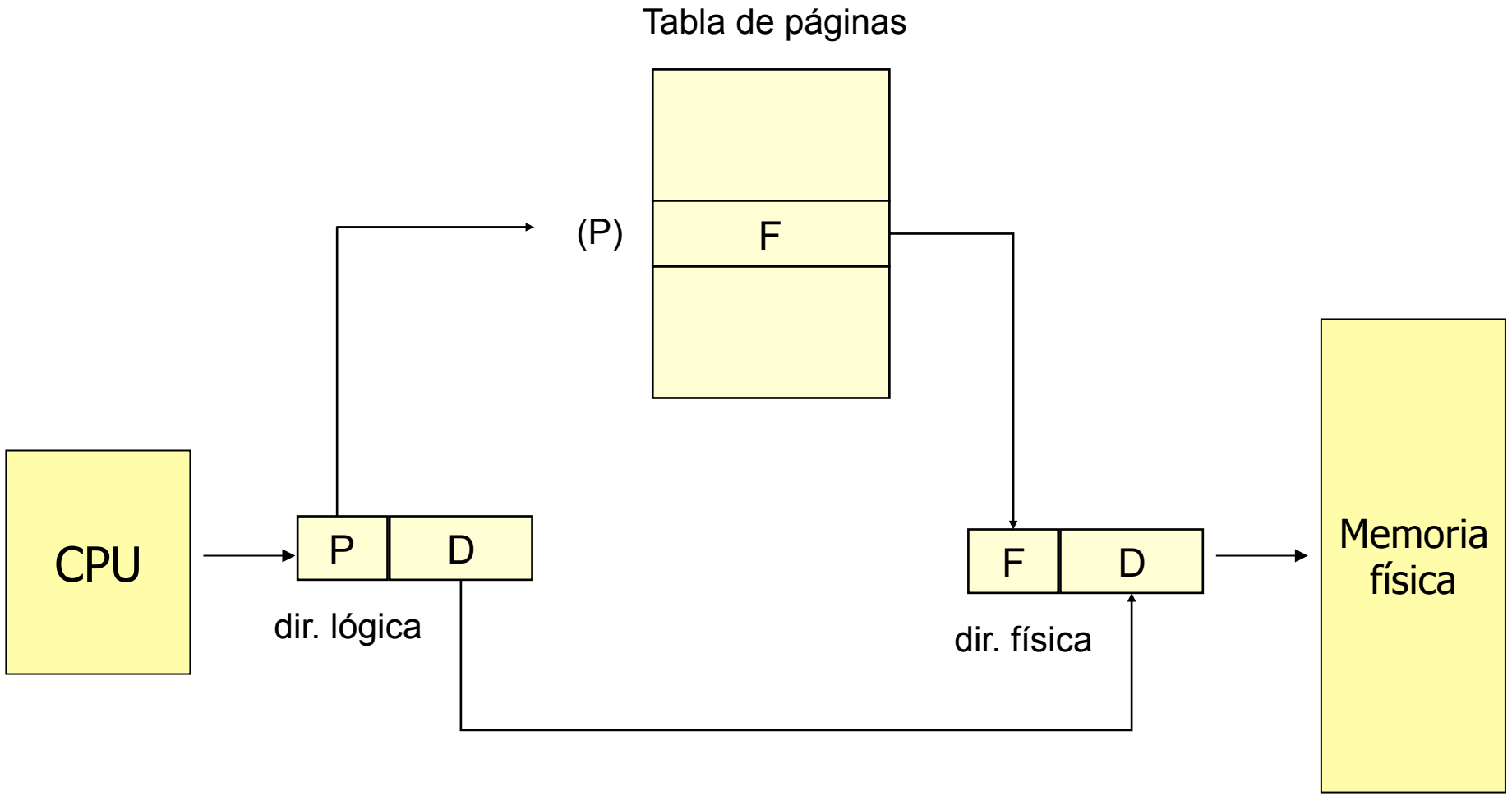


Paginación: traducción de direcciones

Traducción de direcciones: direcciones de 16 bits y tamaño de página 4kbytes



Hardware de paginación

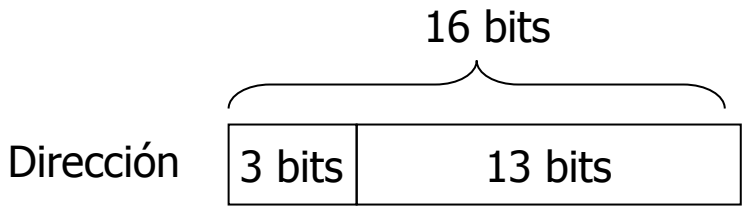


Estructura de la tabla de páginas

- Depende del SO
- Denominador común
  - Una tabla de páginas para cada proceso
- ¿cómo localiza el SO la tabla de páginas de un proceso?
  - BCP
    - Contador de instrucciones, registros, info. de E/S, etc... Y
    - Puntero a la tabla de páginas
- ¿Qué ocurre en un cambio de contexto?
  - Despachador cargará los registros con los valores del nuevo proceso y
  - A partir de la tabla de páginas almacenada, cargará los valores correctos de la tabla de páginas en “hardware” !!

Implementación en hardware de la tabla de páginas

- Conjunto de registros dedicados
  - Ejemplo: Computador DEC PDP-11
    - la dirección consiste en 16 bits y el tamaño de página es de 8k)

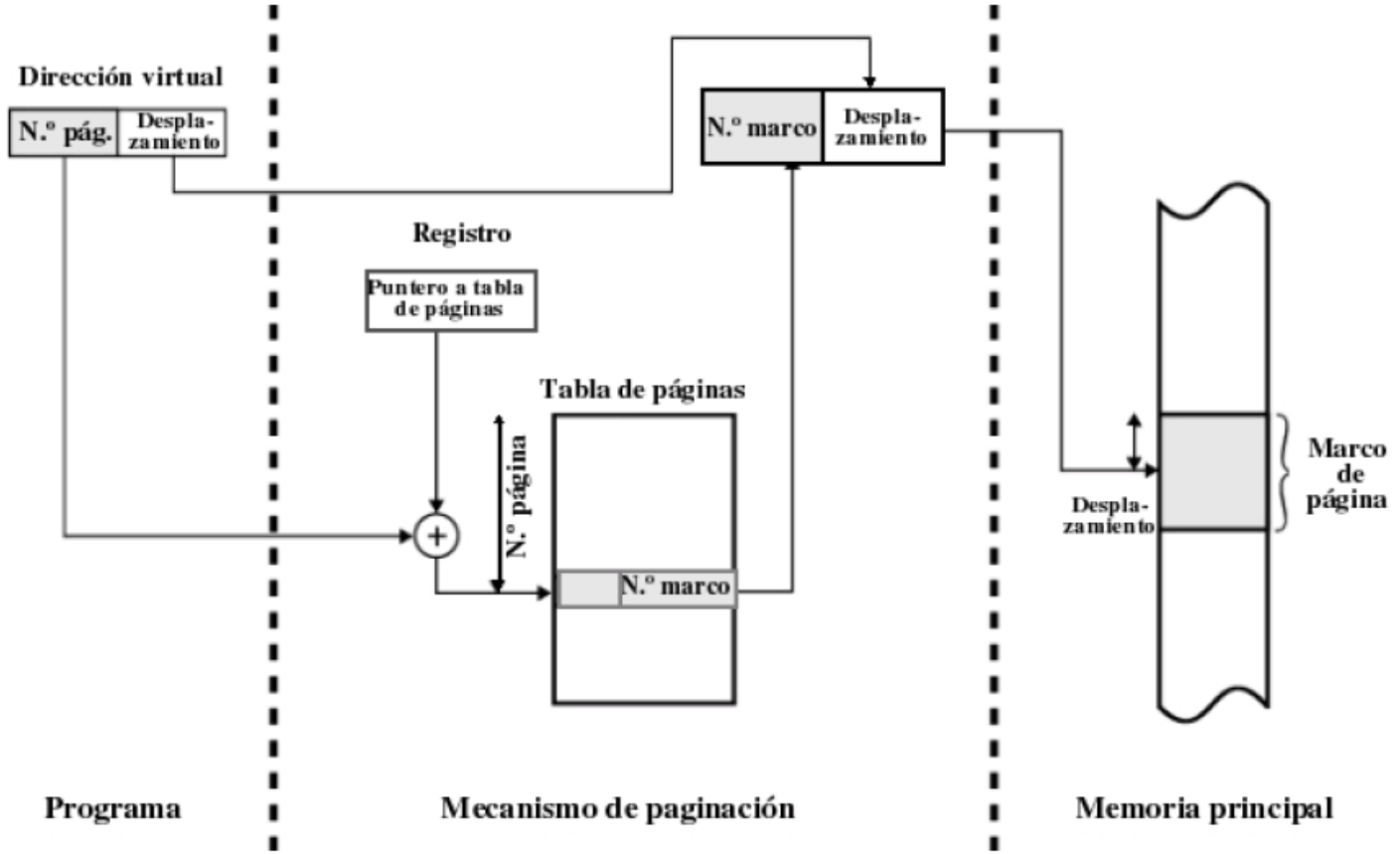


    - La tabla de páginas consta por tanto de ocho entradas que se mantienen en registros rápidos

Implementación en hardware de la tabla de páginas

- Esquema de registros
  - Problema: tablas de páginas grandes
- Solución
  - Mantener la tabla de páginas en memoria
  - Registro base de la tabla de páginas (RBTP) que apunta a la TP
    - Cambio de contexto: más rápido (sólo cambiar el valor de este registro)
    - Gran inconveniente: tiempo de traducción

Paginación
Traducción de direcciones



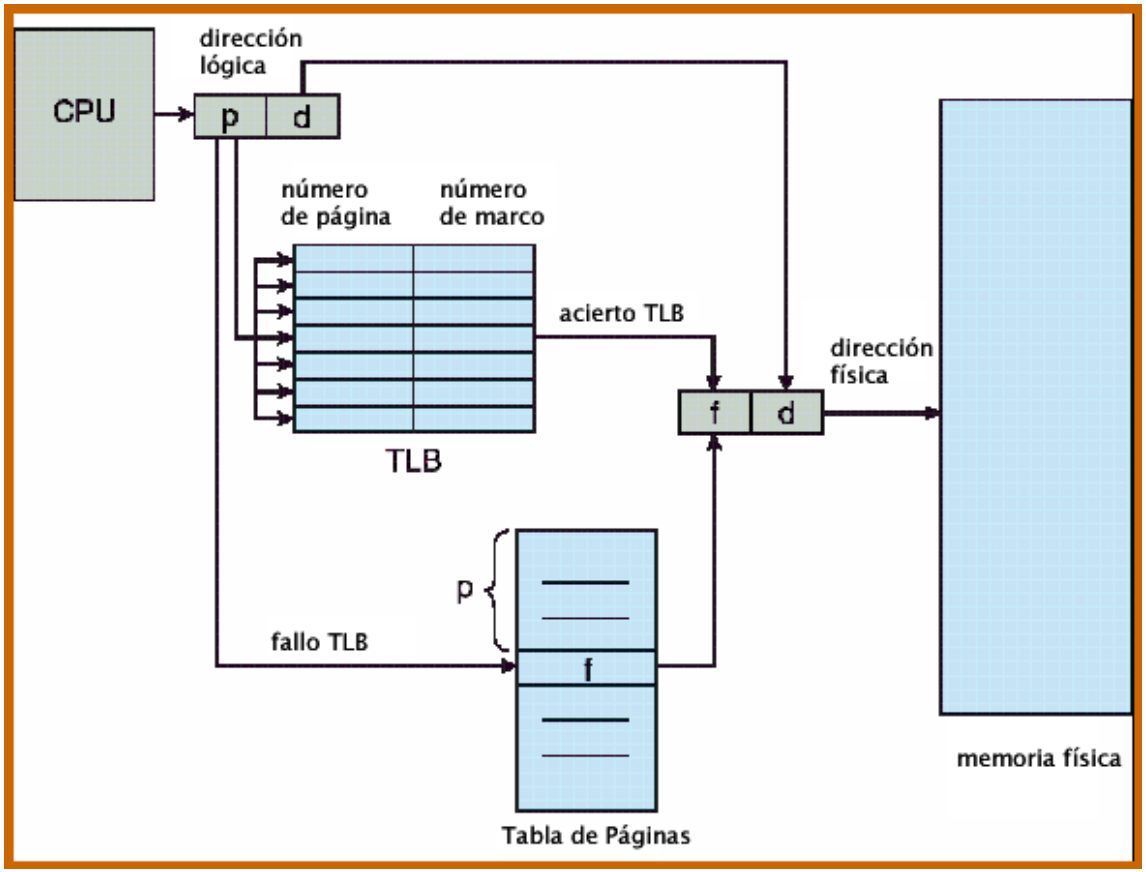
Implementación en hardware de la tabla de páginas

- Solución: usar un TLB (*translation lookahead buffer*)
  - Pequeño *cache* especial en hardware
  - Cada registro consta de dos partes: clave y valor
  - Funcionamiento:
    - Se presenta una clave y, si encuentra alguna coincidencia, devuelve el valor correspondiente
  - Permite búsquedas rápidas pero el hardware es costoso

TLB: cómo funciona

- Funcionamiento: acceso posición i
  - Obtiene el número de página donde se encuentra i
  - Si está en TLB → Obtenemos el marco de página donde se encuentra
  - sino, acceso a la tabla de páginas y actualizar TLB
    - Si TLB llena → Sustitución de una de las existentes
- OJO, cambio de contexto
  - Desalojar (borrar) el TLB

TLB: esquema general



TLB: tasa de aciertos

- Tasa de aciertos
  - porcentaje de las veces que un número de página se encuentra en los registros asociativos
  - relacionada con el número de registros asociativos
  - 16-512 => pueden obtenerse tasas de aciertos entre 80 y 98%.
- Ejemplos
  - Motorola 68030 => TLB de 22 entradas
  - Intel 80486 => TLB de 32 entradas
    - Sus fabricantes dicen que tiene una tasa de aciertos del 98%

Tabla de páginas invertidas

- Problema: tamaño que puede llegar a ocupar la tabla de páginas
- Idea: usar una **tabla de páginas invertida**
  - Tiene una entrada por cada marco real de la memoria
  - Cada entrada consiste en la página virtual almacenada en dicho marco y el proceso al que pertenece
  - Por tanto, sólo hay una tabla de páginas en el sistema que contiene una entrada por cada marco de página

Tabla de páginas invertida

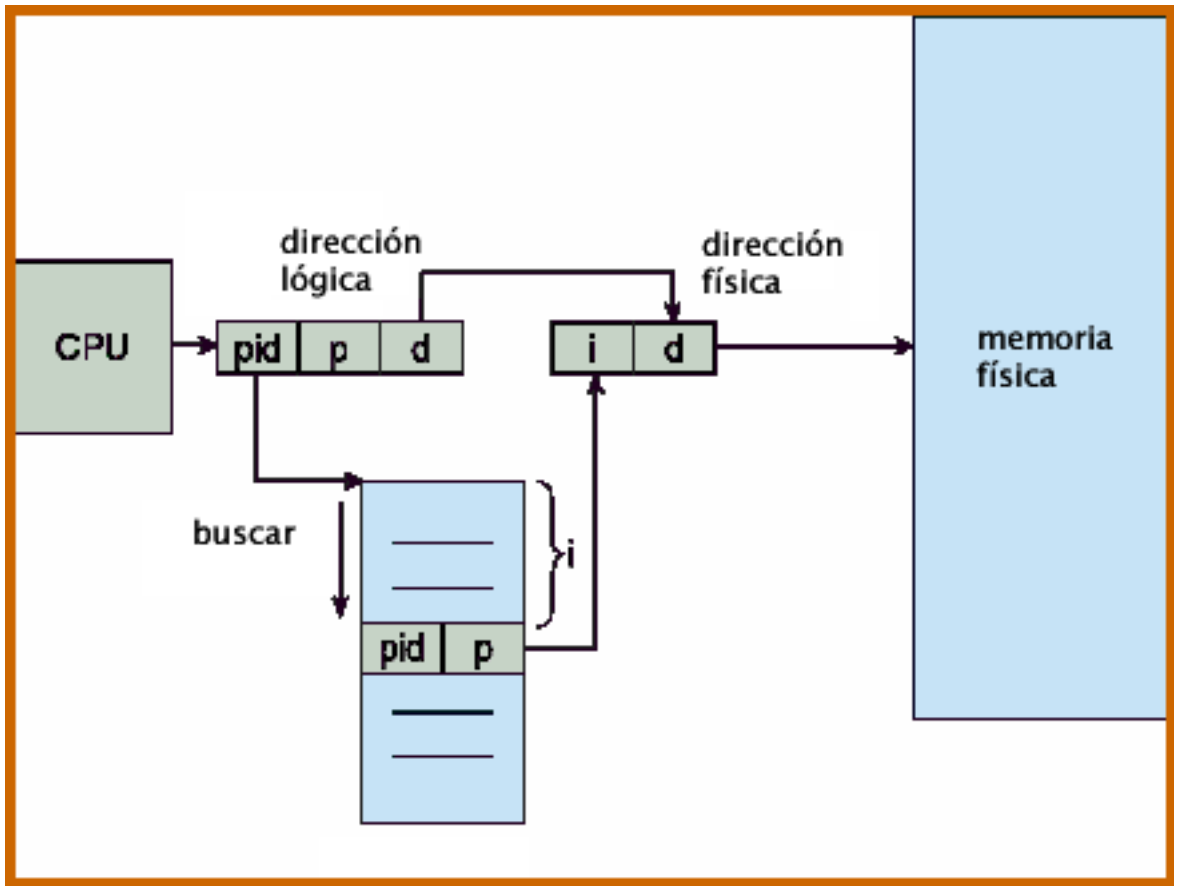


Tabla de páginas invertida

- Ventajas
  - Reduce la cantidad de memoria necesaria
- Desventaja
  - Tiempo de búsqueda en la tabla de páginas invertida
  - Soluciones:
    - Tabla de dispersión (hash)
    - Registros asociativos (caché)

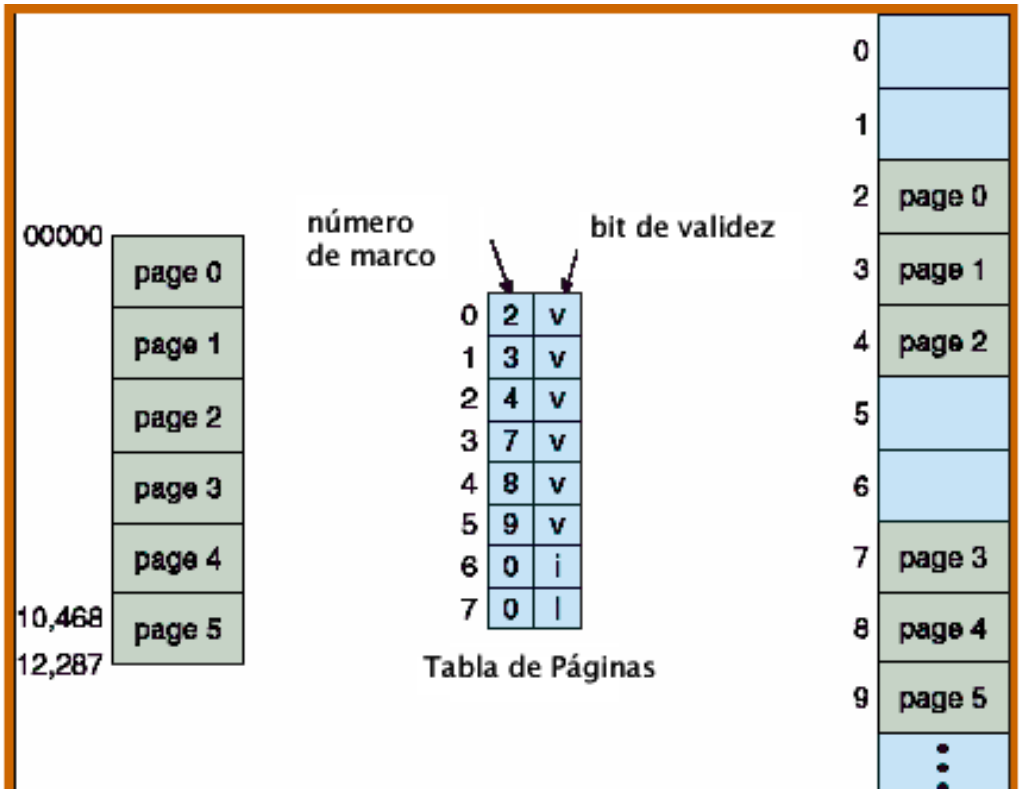
Paginación: inconvenientes

- Pequeño inconveniente: fragmentación interna
  - ¿Tamaño de las páginas?
    - Pequeño
      - mejora fragmentación interna
      - Aumenta el tamaño de la tabla de página
    - Grande
      - Peor desde el punto de vista de la fragmentación interna
      - Tamaño de las tablas de páginas menor
    - Tendencia en los últimos años
      - Aumentar a medida que los procesos, los conjuntos de datos y la memoria principal se han vuelto más grandes
      - 2-4 Kbytes

Paginación: protección

- Las páginas pueden tener asignados bits de protección (ej. lectura, escritura, ejecución)
- Bit de validez/no validez
  - Indica si la página correspondiente está en el espacio de direcciones lógico del proceso y por tanto es válida

Paginación: bit de validez



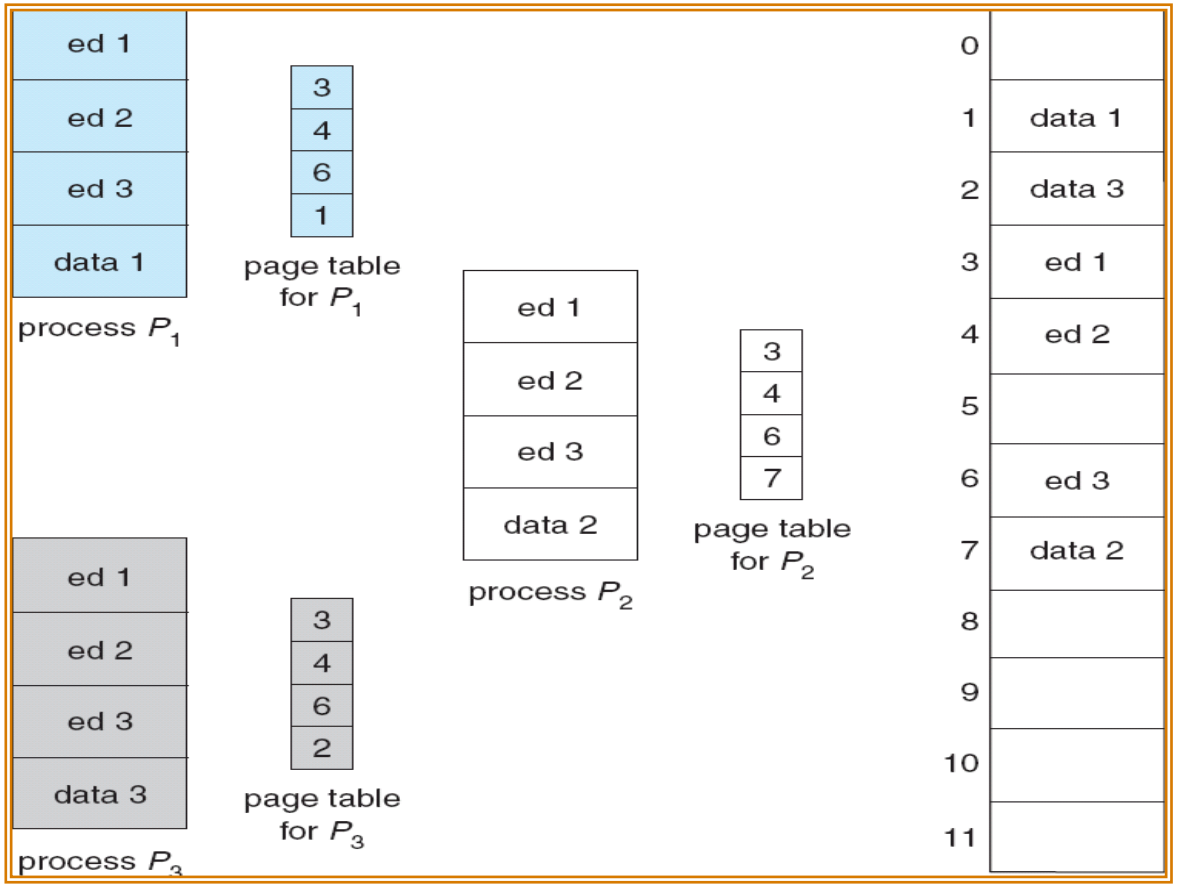
Paginación: protección contra accesos fuera de límites

- Un proceso casi nunca utiliza todo su intervalo de direcciones
  - En estos casos sería un desperdicio crear una tabla de páginas con entradas para todas las páginas del intervalo de direcciones
- Algunos sistemas: registro de longitud de la tabla de páginas (RLTP)
  - Indica el tamaño de la tabla de páginas y se coteja con cada dirección lógica para asegurar que la dirección esté en el intervalo válido para el proceso

Compartición de páginas

- Varios procesos podrían tener la misma memoria física apuntada en sus respectivas tablas de páginas
- La compartición de código exige que el código sea *reentrante,* es decir, no puede modificarse a sí mismo

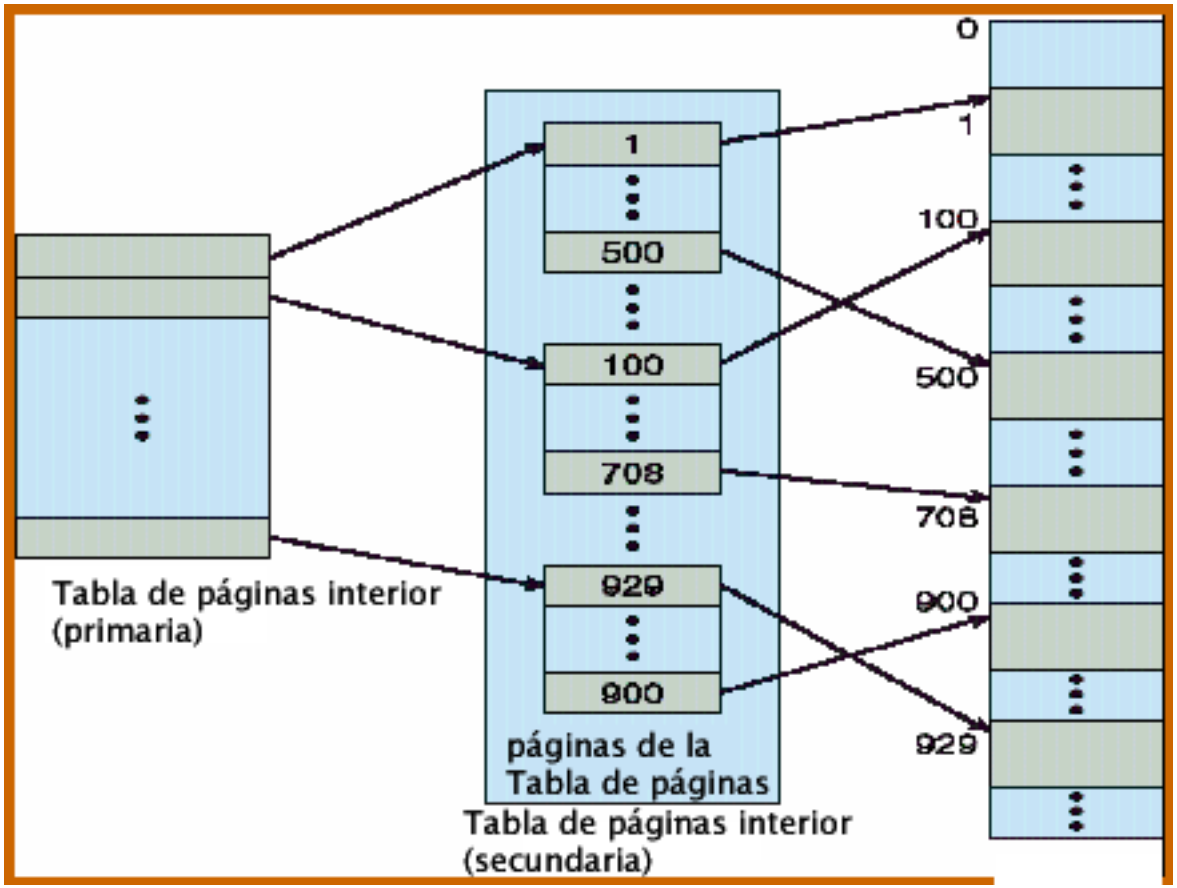
Compartición de páginas



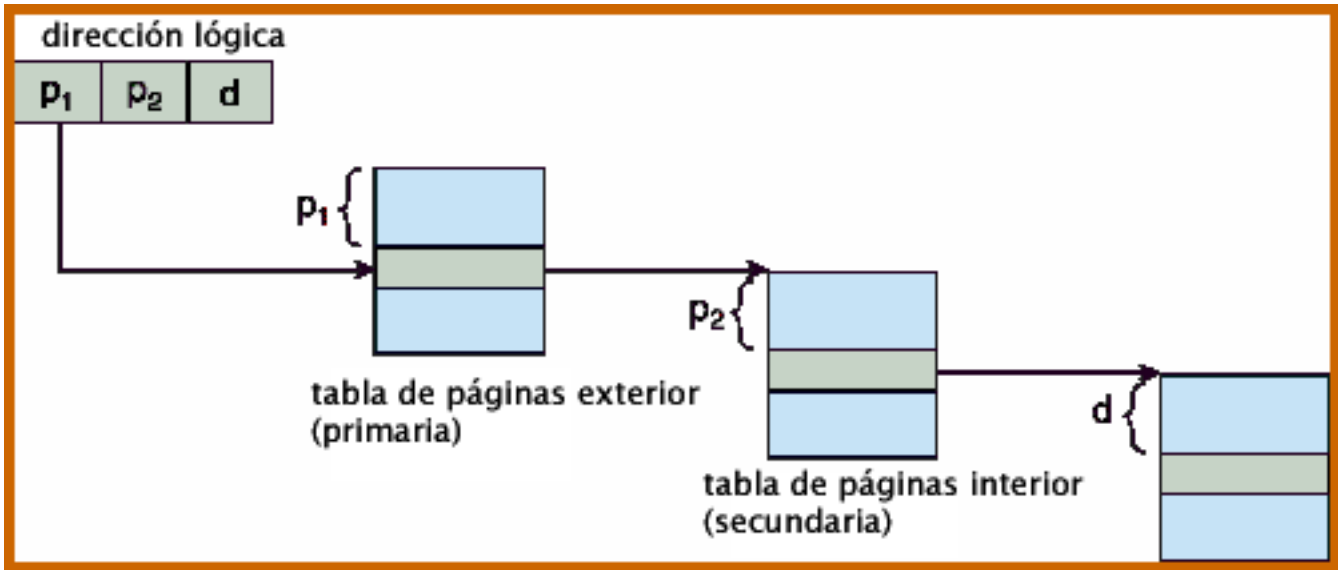
Paginación de varios niveles

- Sistemas modernos => espacio de direcciones lógico muy grande (2^{32} a 2^{64})
- Problema: tamaño de la tabla de páginas
  - Por ejemplo, si tamaño de página es de 4k, un proceso podría requerir hasta **4 Mb de espacio** físico para la tabla de páginas
  - Solución: paginar la tabla de páginas, teniendo varios niveles de páginas (ej. 80386)

Paginación de varios niveles



Paginación de varios niveles: 80386



Como máximo hay un total de :

$2^{(n1+n2)}$ páginas por proceso

2^{n2} entradas en cada tabla de páginas interior

2^{n1} entradas en la tabla de páginas exterior

2^{n1} tablas de páginas interiores

Esquema combinado segmentado/paginado

- La paginación y la segmentación pueden combinarse (ej. MULTICS, 80386).
- Motivación: aprovecharse de las ventajas que ofrecen los esquemas por separado
  - Segmentación: flexibilidad y facilidad para la organización lógica
  - Paginación: mejorar el problema de la fragmentación

Esquema combinado segmentado/ paginado: 80386