Sistemas Operativos
Tema 4. Memoria

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

Contenidos

- Conceptos básicos
- Gestión de memoria contigua
- Segmentación
- Paginación

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 que quieren ejecutarse

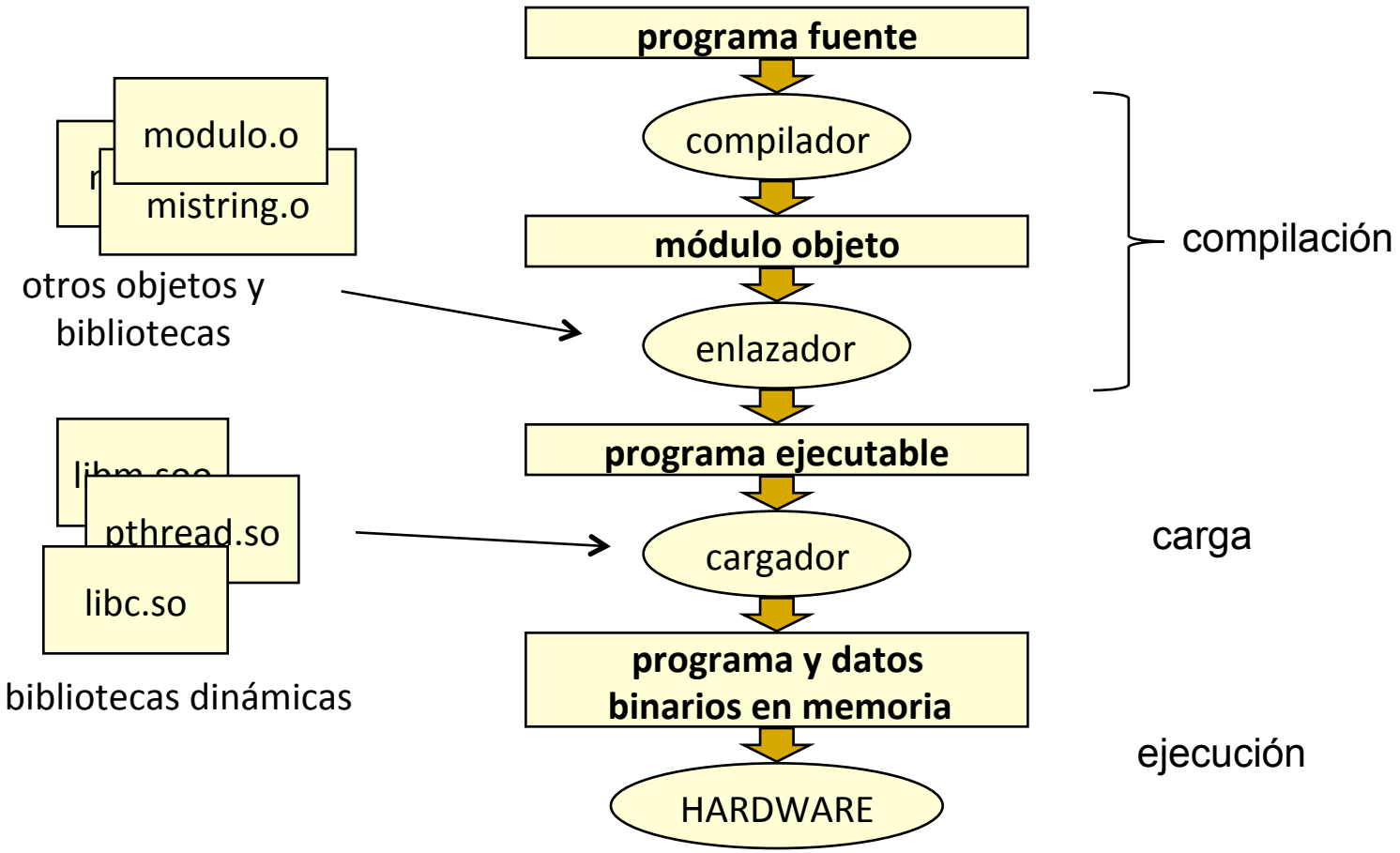
Gestión de la memoria: objetivos

- **Rendimiento:**
  - Aprovechar al máximo la memoria disponible
  - Conseguir tener cargados en memoria varios procesos… y que el sistema no se venga abajo
  - Que un programa no necesite estar totalmente cargado en memoria para poder ejecutarse → apoyarse en la memoria secundaria, si es preciso
- **Protección y compartición:**
  - Proteger al SO de accesos indebidos
  - Proteger el espacio de memoria de cada proceso
  - Permitir que varios procesos puedan compartir zonas de memoria (para comunicación, o para reutilizar código o datos)

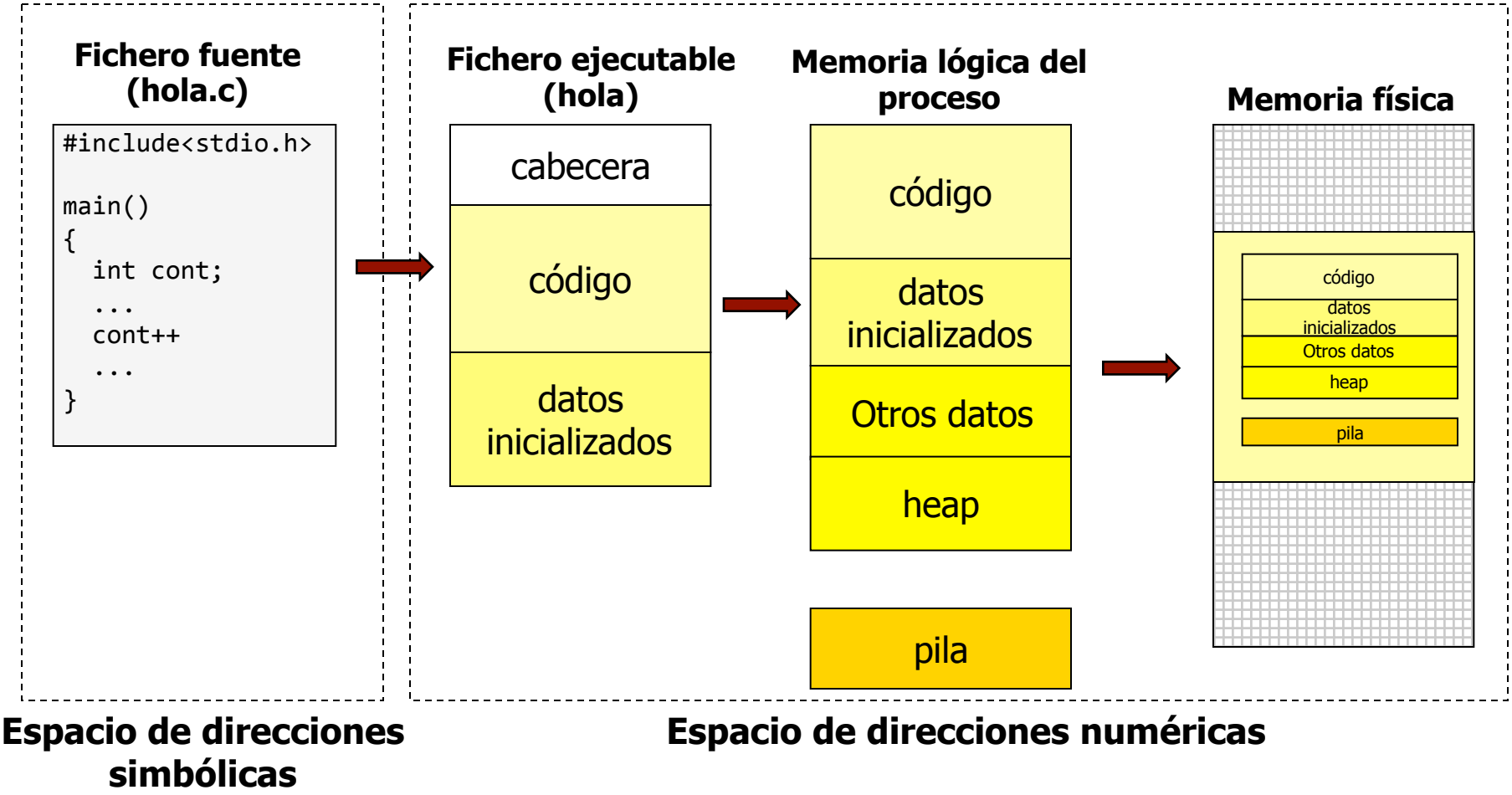
Gestión de la memoria: cuestiones críticas

- **Reubicación:** facilitar que un programa pueda residir en cualquier zona de la memoria física.
- **Fragmentación:** ojo a los huecos que van quedando a medida que se asignan y liberan zonas de memoria.
- **Tiempo de acceso:** ojo al impacto de los mecanismos de gestión en el tiempo neto de acceso a memoria.

Ciclo de vida de un programa



Espacios de direcciones



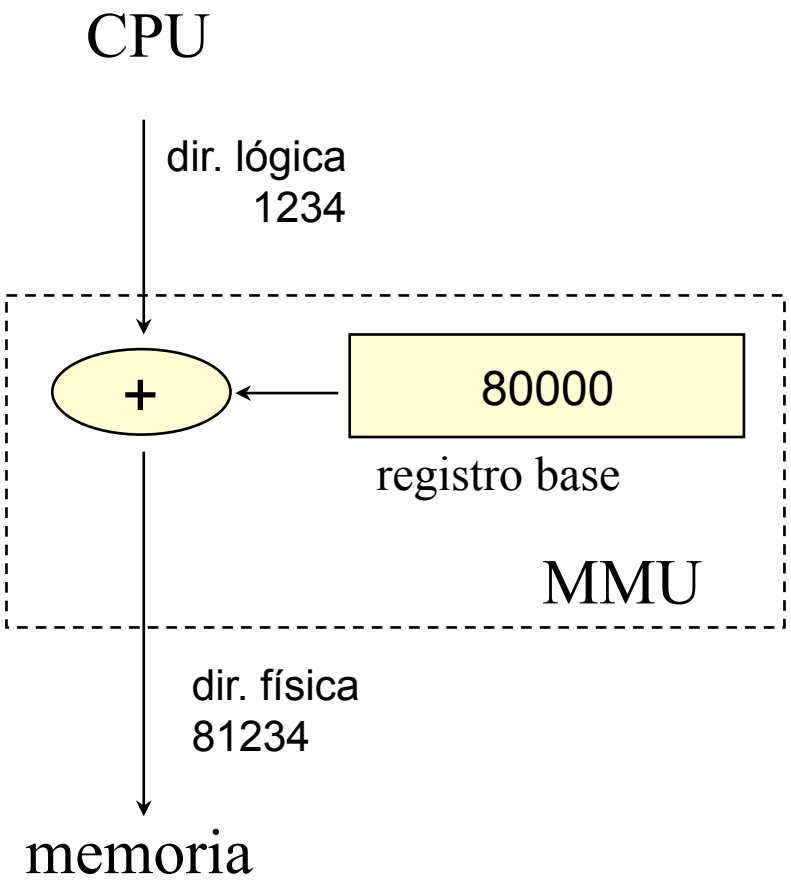
Conversión de direcciones: reubicación

- 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.
- Esto es una grave limitación.

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: ejemplo mínimo (registro base)



Reubicación dinámica: direcciones lógicas/direcciones físicas

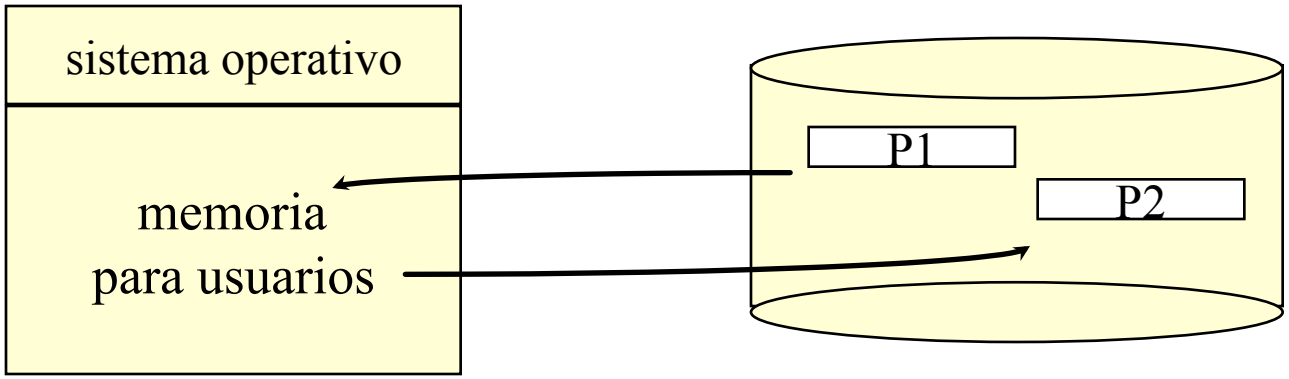
- **Dirección lógica o virtual:** la generada por la CPU.
- **Dirección física:** la que llega al chip de memoria.
- **Unidad de manejo de memoria (MMU):** el dispositivo que traduce direcciones lógicas a físicas.

Enlace dinámico / carga dinámica

- Postergar la carga en memoria de un módulo de código hasta que se ejecute por primera vez.
  - Windows: DLL (dynamic link libraries)
  - Unix: shared libraries
- 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.
- Código compartido → si varios procesos emplean la biblioteca dinámica, sólo se mantiene una copia de ella en memoria.
- El SO revisa las DLL que están en memoria pero que nadie referencia.

Intercambio (swapping)

- Si un proceso lleva mucho tiempo bloqueado, su espacio de memoria está desperdiciado.
- Idea: se vuelca su imagen de la memoria al disco (swap out). Ese espacio queda disponible para otros.
- Cuando se decide reanudar el proceso, se recupera su imagen del disco (swap in).

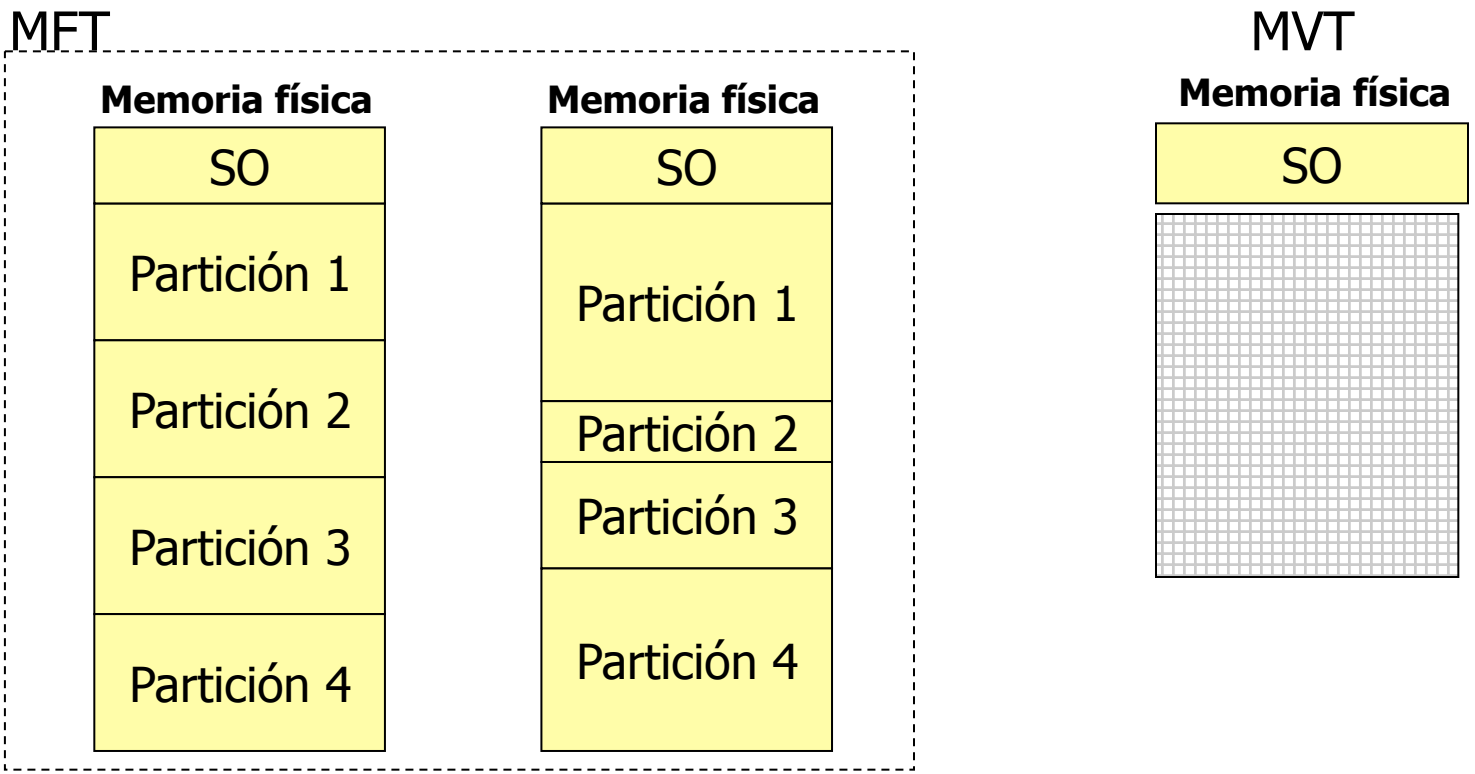


Intercambio (swapping) (2)

- Historia: fue una forma de conseguir multiprogramación cuando había muy poca RAM
  - Caso extremo: sólo un proceso en memoria; en cada cambio de contexto se realiza un intercambio con disco.
- Problema: tiempo invertido en el intercambio
- Soluciones:
  - Mientras se intercambia un proceso, otros procesos pueden seguir ejecutándose en otras zonas de memoria.
  - Usar varias áreas de intercambio en diferentes dispositivos físicos (se pueden hacer varias transferencias simultáneas).

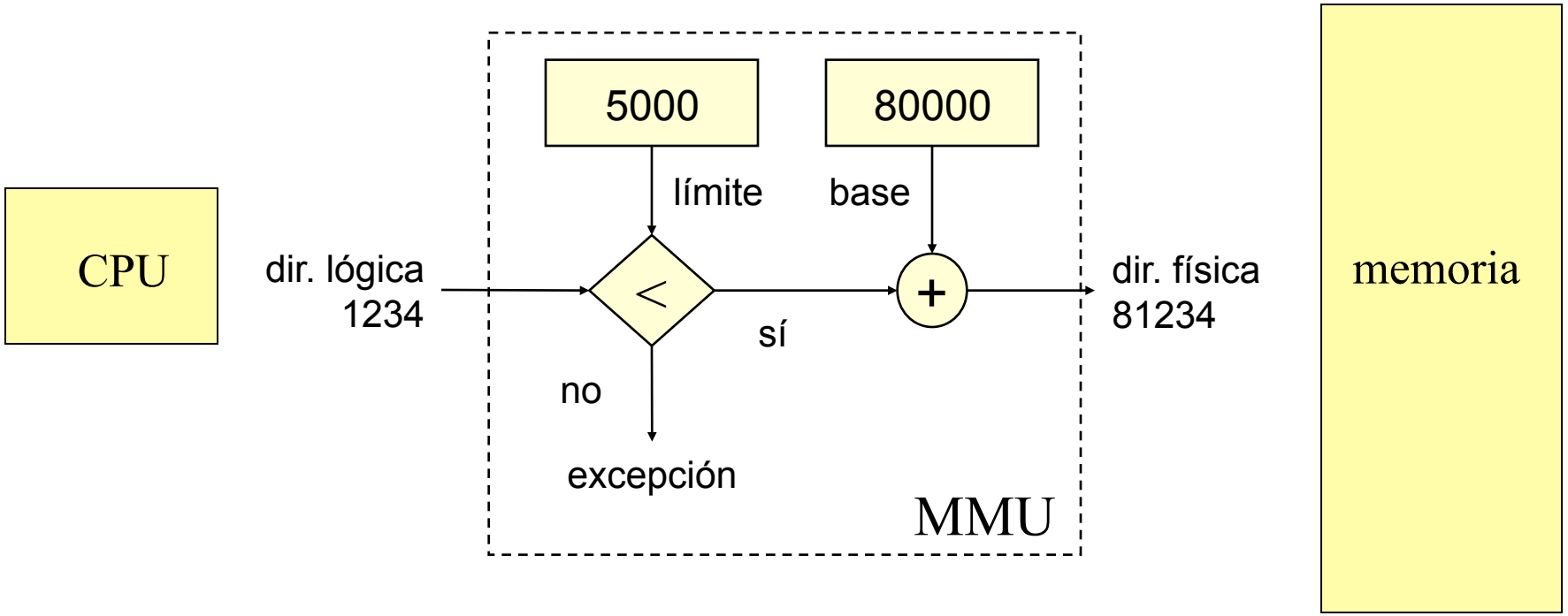
Particiones múltiples (años 50/60)

- **particiones de tamaño fijo (MFT)** – preestablecidas en el arranque del SO
- **particiones de tamaño variable (MVT)** – evolución del MFT - lista dinámica de huecos libres



Memoria contigua: protección

- Pareja de registros base y límite



Memoria contigua: estructuras de datos

- Tabla de descripción de particiones (TDP) – indica qué proceso posee cada partición
- 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

Políticas de asignación de memoria

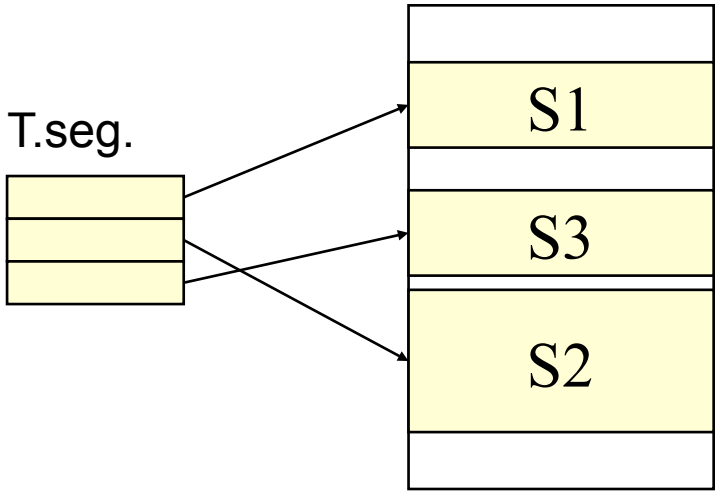
- Disponemos de un conjunto de huecos libres
- Cada hueco tendrá un tamaño variable
- ¿Qué hueco damos ante una petición?
  - Primer hueco (first-fit) – recorremos la lista y damos el primer hueco mayor o igual que lo solicitado
  - Siguiente hueco (next-fit) – igual que first-fit, pero se busca a partir de donde se encontró el último hueco libre
  - Mejor hueco (best-fit) – buscamos el hueco que deje menor resto libre
  - Peor hueco (worst-fit) – siempre usamos el hueco más grande
- las políticas de “primer hueco” y “mejor hueco” son similares en rendimiento; y mejores que la de “peor hueco” y “siguiente hueco”

Fragmentación externa / compactación

- A medida que se van asignando y liberando huecos, van quedando zonas pequeñas que no sirven para nada.
- Posible solución → **compactación**
- Sólo es viable si se usa reubicación dinámica (cambian los espacios físicos de los procesos)
- La compactación puede consumir mucho tiempo
- Otra estrategia: permitir que el programa se divida en "trozos" → **memoria no contigua**
  - **Segmentación / paginación**

Segmentación

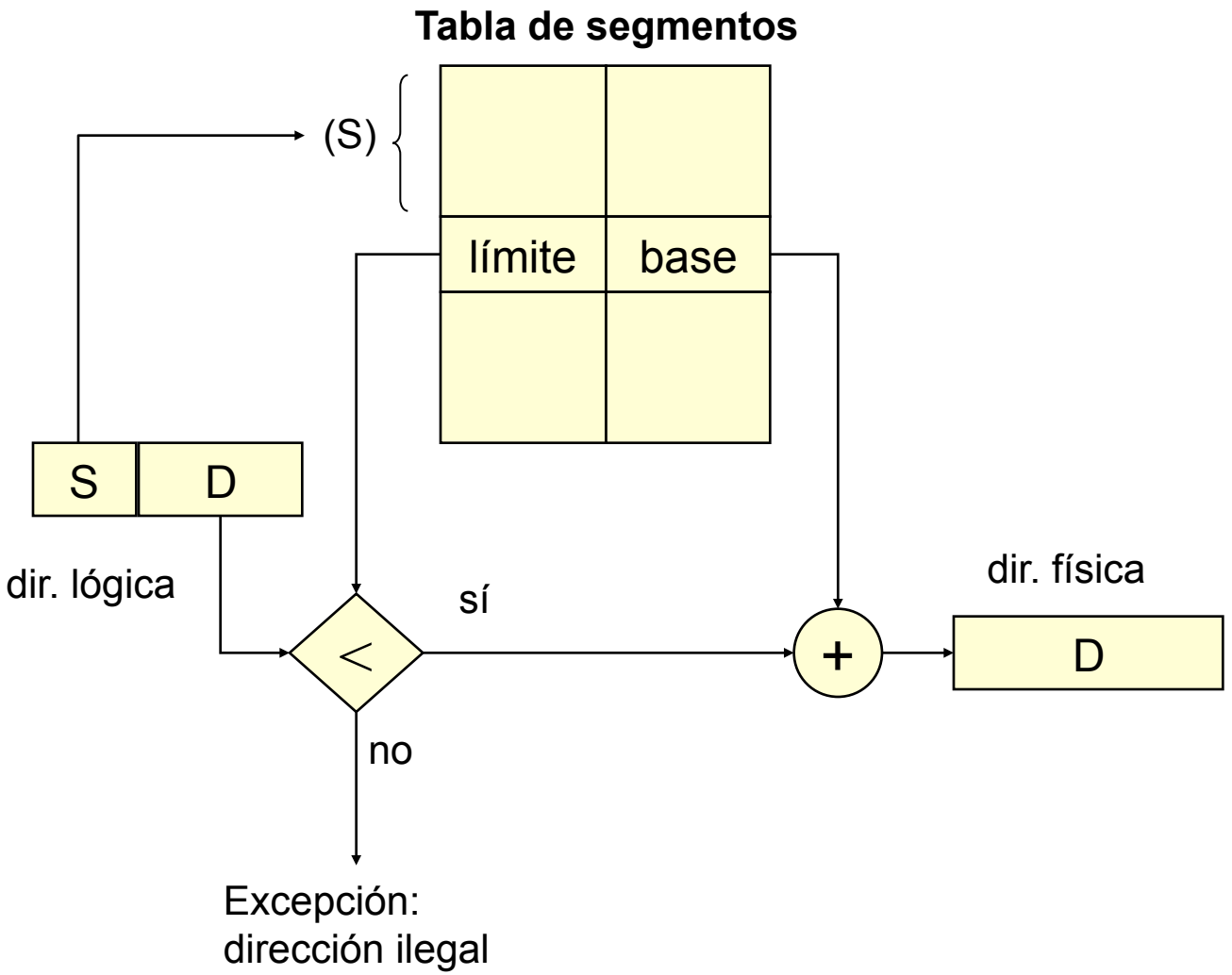
- Idea: descomponer el proceso en varios segmentos de memoria (código, datos, pila...)
- Con el hardware adecuado, podemos ubicar esos segmentos en zonas de memoria no contiguas.
- Podemos reducir bastante la fragmentación



Segmentación

- La **CPU** trabaja con direcciones de memoria de dos piezas: *<segmento,desplazamiento>*
- El **compilador** identifica segmentos del programa y genera direcciones segmentadas
- El **cargador** del SO ubica cada segmento en una zona de memoria diferente
- La **MMU** traduce las direcciones <seg,desplz> a direcciones lineales que entiende la RAM
  - **Tabla de segmentos** (una tabla de registros base +límite)

Hardware de segmentación



Segmentación: beneficios

- Atenúa el problema de la fragmentación
- Permite compartir zonas de memoria
  - ej. una DLL compartida como un segmento en las tablas de varios procesos
- Los segmentos pueden tener protección por hardware:
  - ej. segmentos de sólo lectura
  - ej. sólo permitir ejecución de instrucciones en segmentos que estén marcados como “de código”

Segmentación: inconvenientes

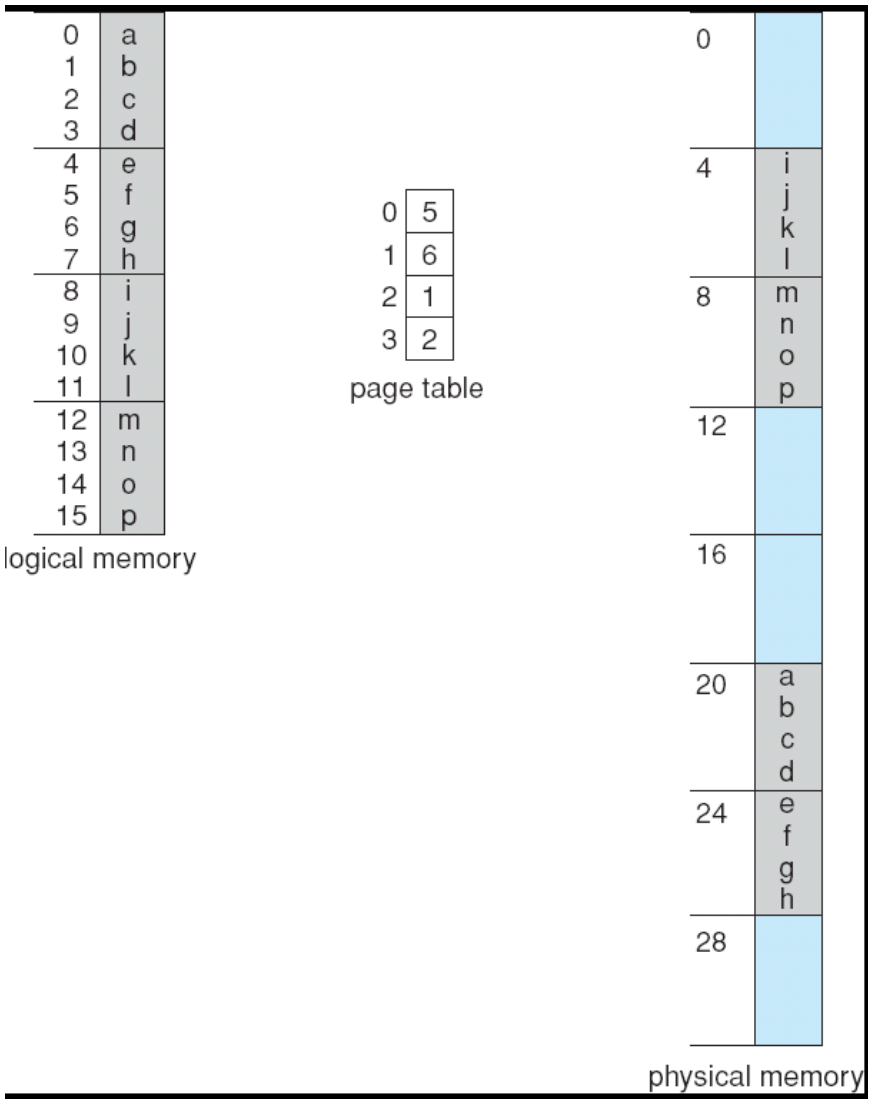
- El compilador/enlazador debe trabajar con un espacio lógico de dos dimensiones (segmentos + desplazamientos)
- Necesita soporte del hardware
- El acceso a memoria se hace mucho más lento (hay que acceder a la tabla de segmentos)
- No soluciona del todo la fragmentación
- Su eficacia depende mucho de cómo el compilador haya troceado el proceso en segmentos

PAGINACIÓN

Paginación

- Técnica que erradica definitivamente la fragmentación externa.
- La idea: Trocear el programa en **páginas** de tamaño fijo (ej. 4KiB). Un programa puede residir en varias páginas no contiguas en la memoria.
- 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 cada número de página lógico con el marco de página físico. Para ello emplea una **tabla de páginas**.

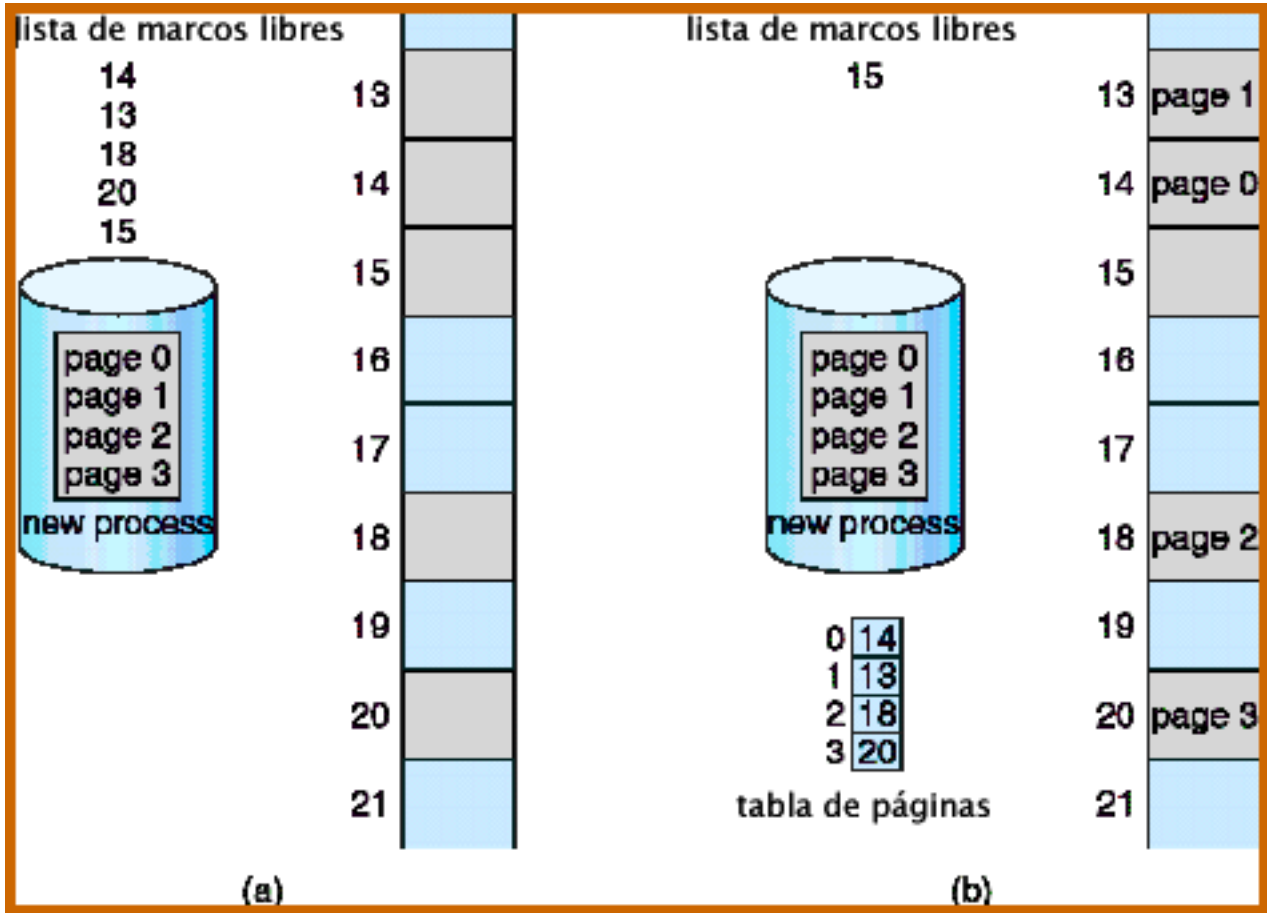
Paginación: ejemplo



Paginación: gestión del espacio libre

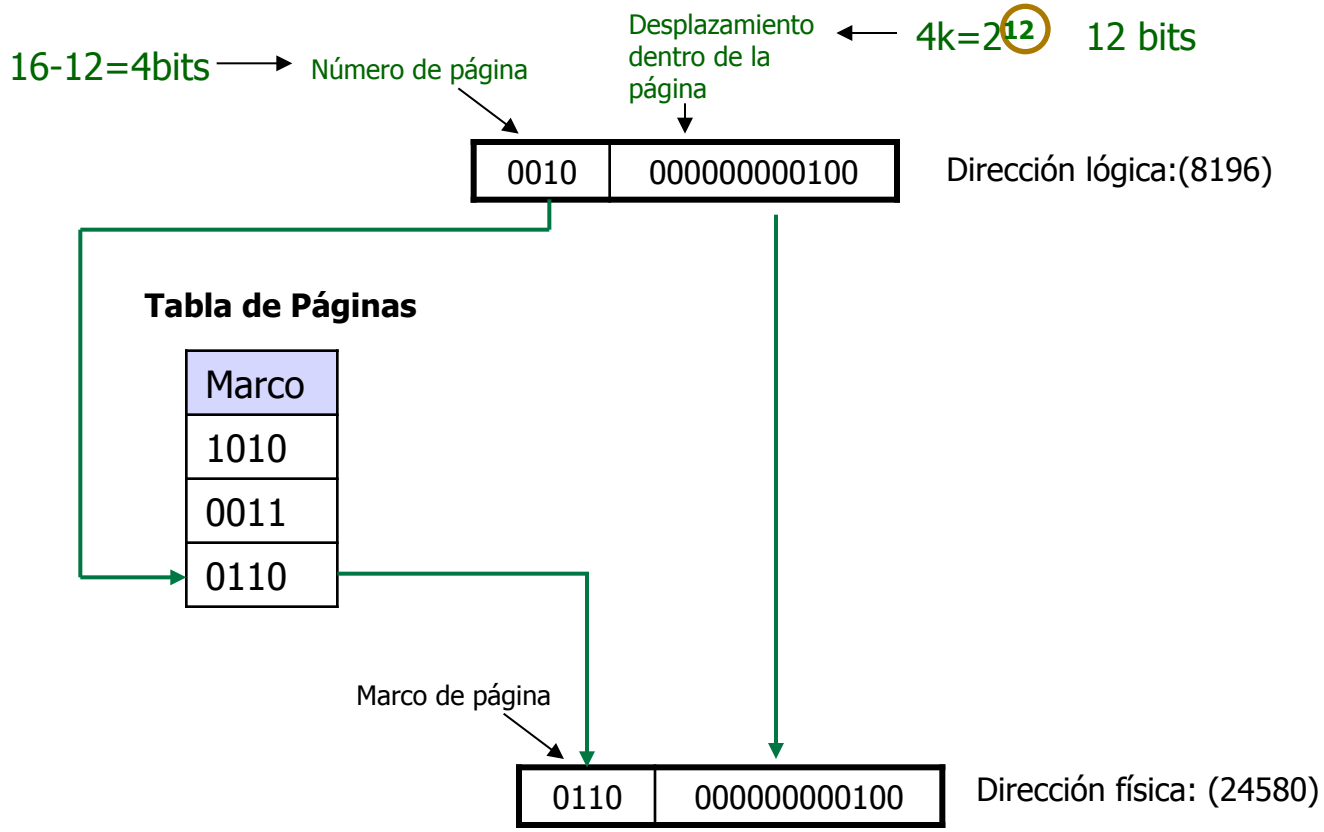
- La gestión del espacio libre consiste simplemente en saber qué marcos están libres
- El SO posee una **tabla de marcos de páginas** (TMP) → la podemos implementar con un mapa de bits

Paginación: marcos libres

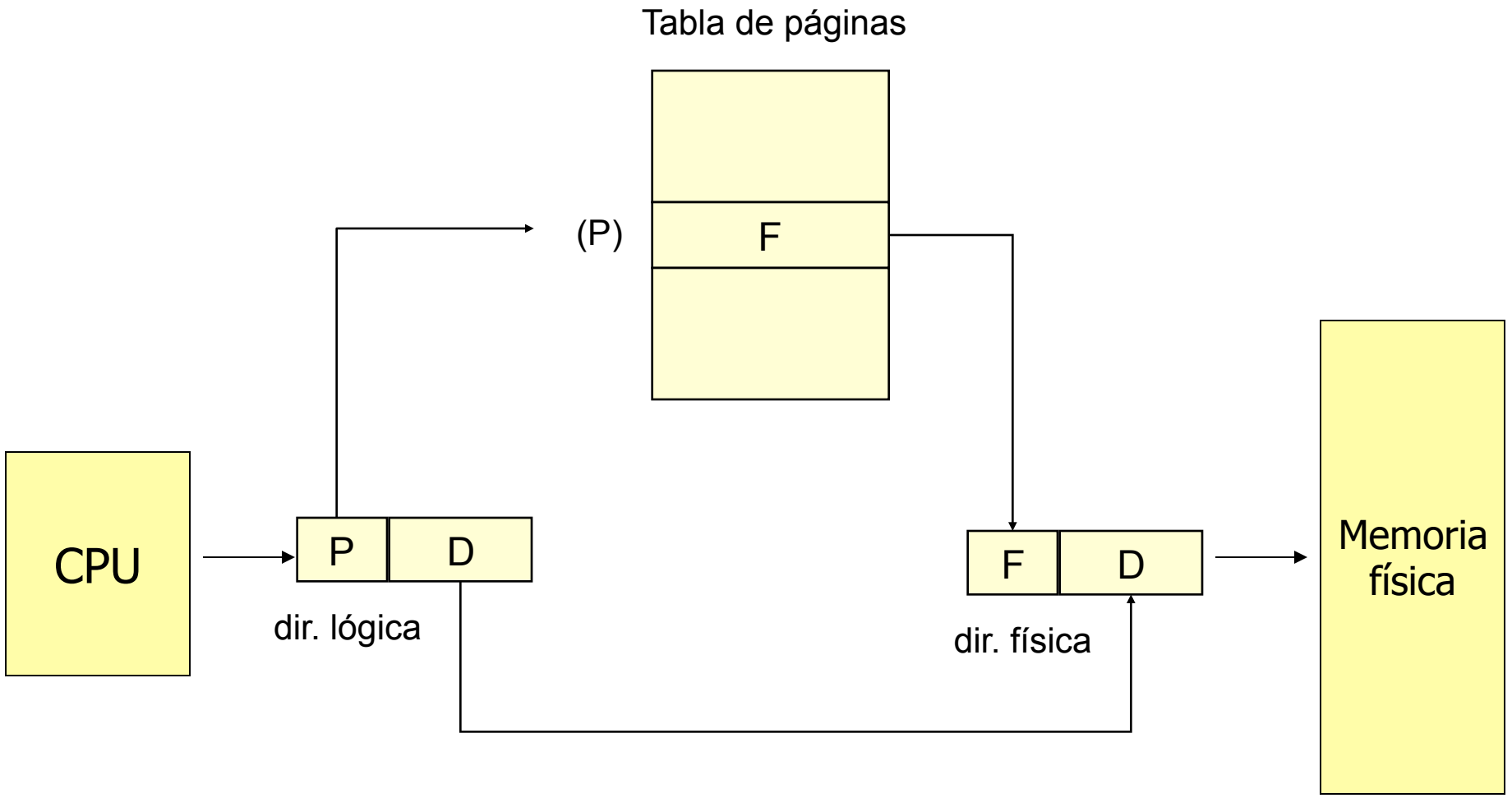


Paginación: ejemplo de traducción

- Direcciones lógicas de 16 bits, páginas de 4KiB

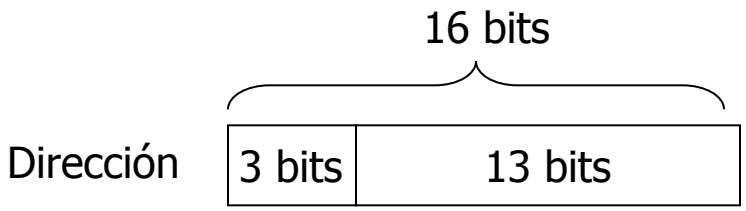


Hardware de paginación



Hardware de paginación primitivo

- 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

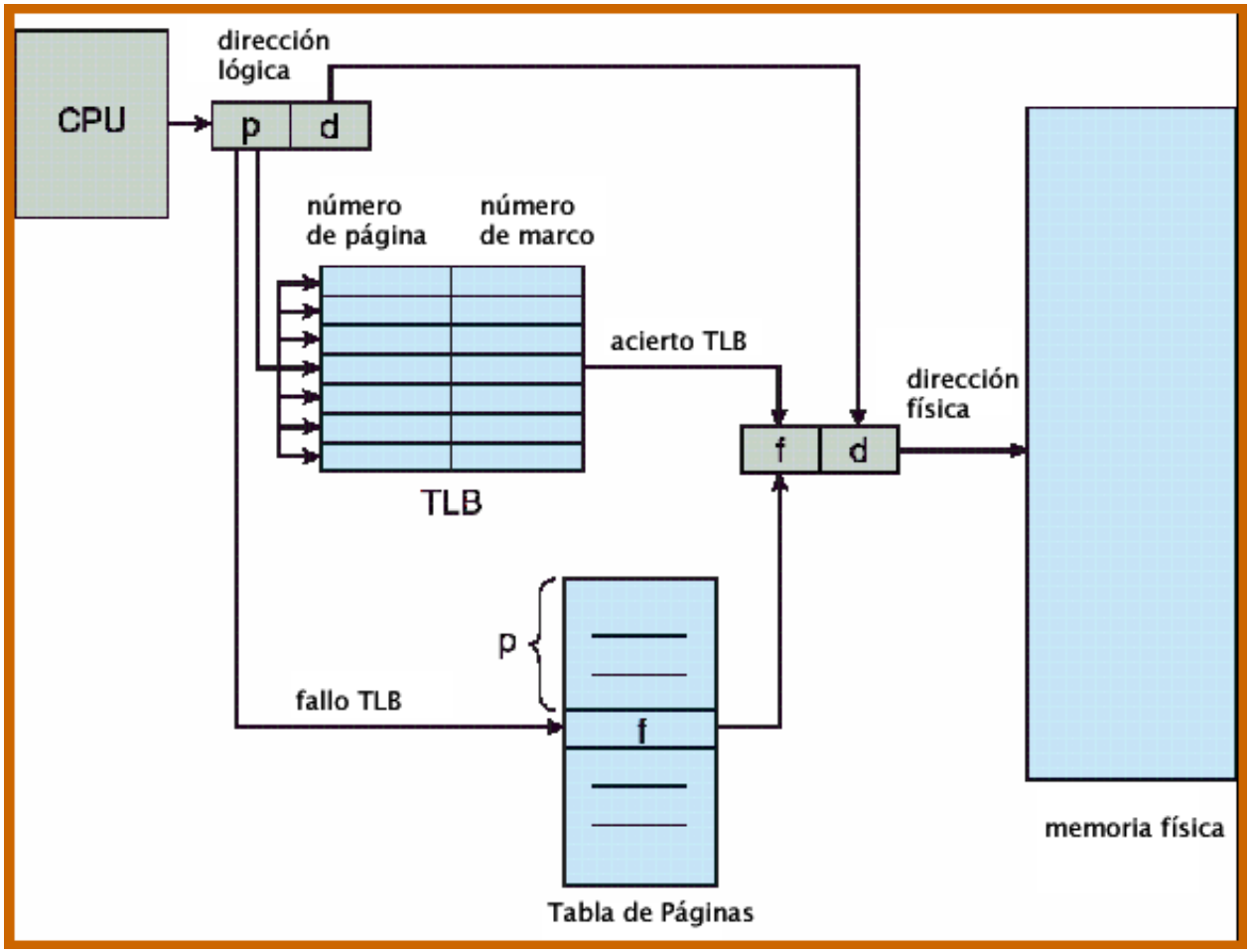
Tabla de páginas en memoria

- En los sistemas actuales, la TP se guarda en memoria principal.
  - Registro base de la tabla de páginas (RBTP)
  - Registro de longitud de la tabla de páginas (RLTP)
- Esto permite TP grandes (ej. un proceso de 20MB con páginas de 1K tiene una TP de 20K entradas).
- Un registro de la CPU apunta a la TP.
- Problema: ¡resolver un acceso de memoria lógica necesita **DOS accesos** a la memoria principal! (uno a la TP, otro a la dirección solicitada).

TLB (Translation Lookaside Buffer)

- Para qué:
  - Acelera el tiempo de traducción
  - Evita accesos a la TP
- Qué es:
  - Pequeña caché dentro del procesador
  - Contiene unas pocas entradas de la TP
  - Tiempo de acceso muy rápido (igual que acceder a un registro de la CPU)

TLB: esquema general



TLB: cómo funciona

- Contiene un conjunto de parejas <*página lógica, marco físico*>
- Cuando se quiere resolver una dirección y la página lógica es P, se busca P en todas las entradas de la TLB, simultáneamente.
- Si P se encuentra en la TLB, se obtiene el marco M de la página P y no hace falta acceder a la TP **(acierto de TLB)**.
- Si no se encuentra **(fallo de TLB)**, se accede a la TP y se actualiza la TLB con la info de P+marco.

TLB: situaciones especiales

- Fallo de TLB + TLB llena → hay que reemplazar una de las actuales entradas… ¿cuál de ellas? → la más vieja, la menos usada, al azar…
- Cambio de contexto → hay que vaciar la TLB, o cargarla con el contenido que tenía el proceso que entra en CPU.

TLB: tasa de aciertos

- Tasa de aciertos
  - porcentaje de las veces que un número de página se encuentra en los registros asociativos
  - pueden obtenerse tasas de aciertos entre 80 y 98%.
  - **Tiempo efectivo de acceso a memoria.** Si la tasa de aciertos es alta, el impacto en el tiempo de acceso es mínimo *(ver discusión y cálculos en el libro)*.
- Ejemplos
  - Motorola 68030 → TLB de 22 entradas
  - Intel 80486 → TLB de 32 entradas

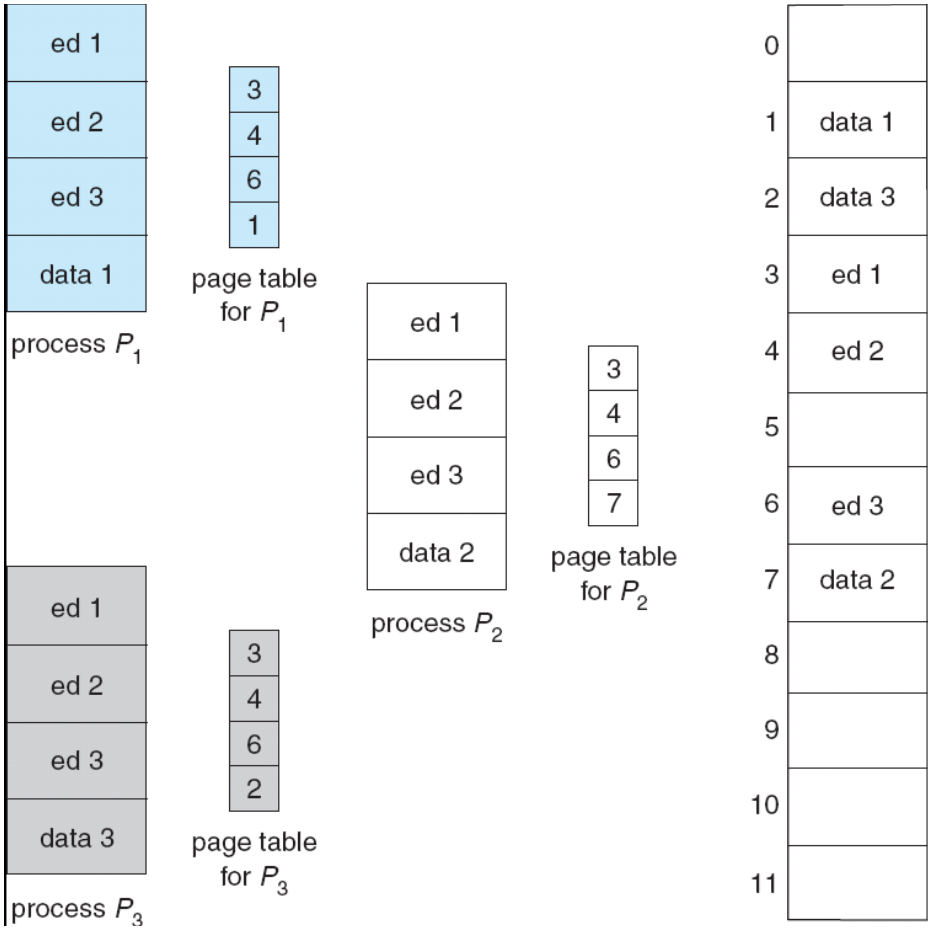
Tamaño de la página: ¿pequeñas o grandes?

- 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

Paginación: protección

- Las páginas pueden tener asignados bits de protección (ej. lectura, escritura, ejecución).
- También hay que protegerse de accesos fuera de límites del espacio lógico. Posibles medidas:
  - Registro de longitud de la tabla de páginas (RLTP)
  - Tabla de páginas invertida (sólo figuran las páginas válidas, ver más adelante)
  - Bit de validez (entradas marcadas como no válidas)

Compartición de páginas

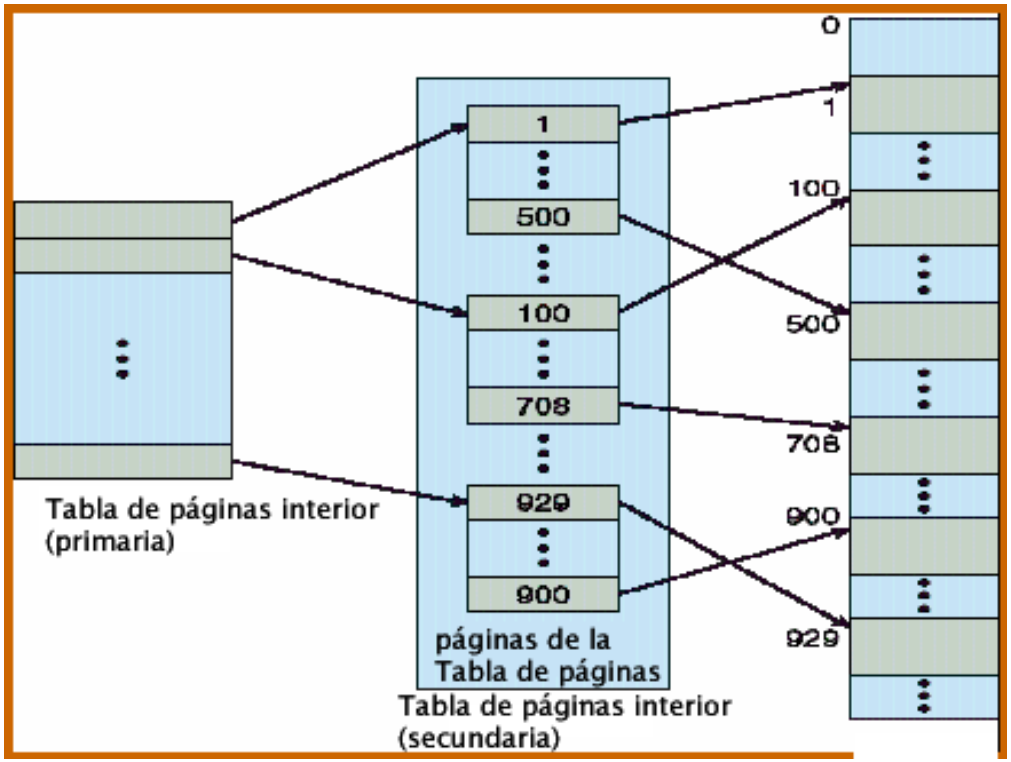


Tamaño de la tabla de páginas

- En una arquitectura de direcciones de 32 bits, con páginas de 4KiB, la tabla de páginas de un proceso podría llegar a ocupar 4 megas. ¡Más grave con 48 o 64 bits!
- Solución "mala": aumentar el tamaño de página → aumenta la fragmentación interna
- Soluciones "buenas":
  - Paginación jerárquica
  - Tabla de páginas invertida
  - Tabla hash de páginas

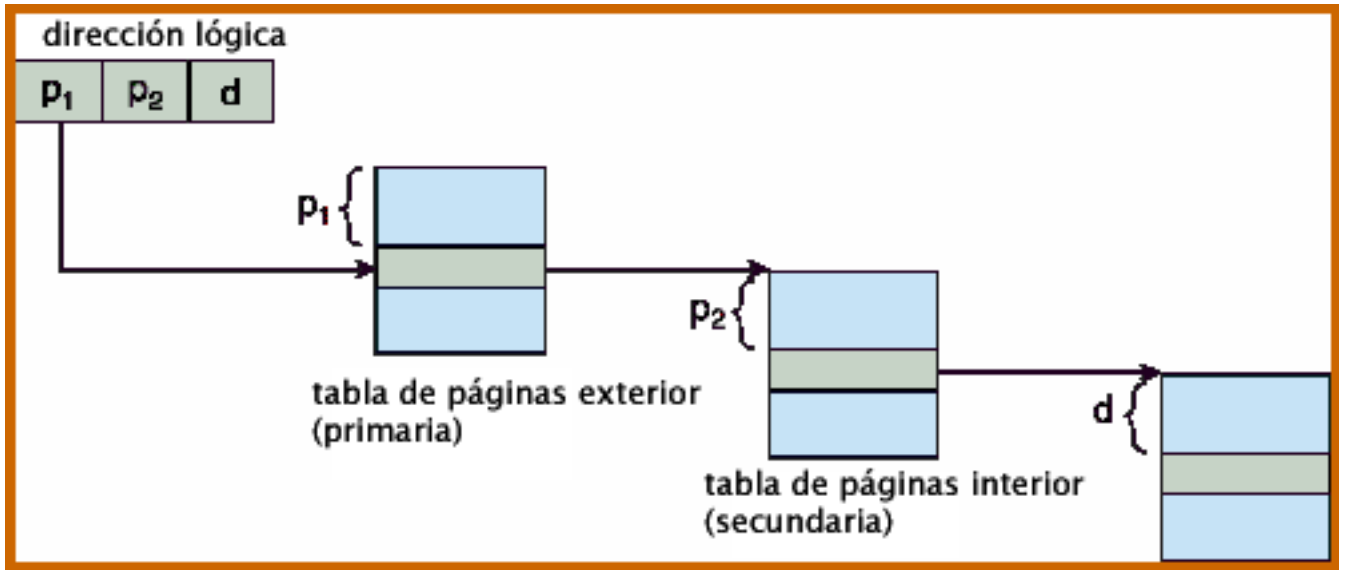
Paginación jerárquica

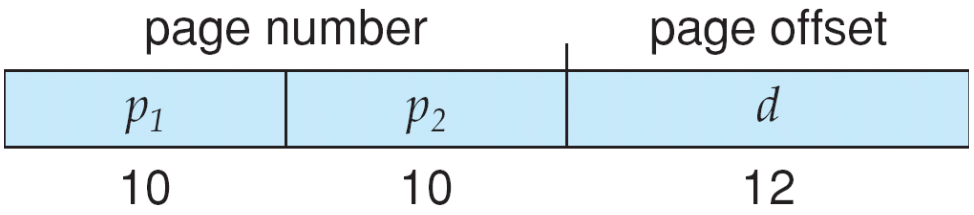
- Paginar la tabla de páginas, con varios niveles jerárquicos (ej. 80386).



Paginación jerárquica: Intel x86

OJO: ¡tres accesos a memoria! La TLB se hace más necesaria





Intel: arquitectura de 64 bits

- Tamaños de páginas de 4K, 2M, 1G (configurable dinámicamente)
- Hasta cuatro niveles de jerarquía
- Actualmente, los chips sólo soportan 48 bits

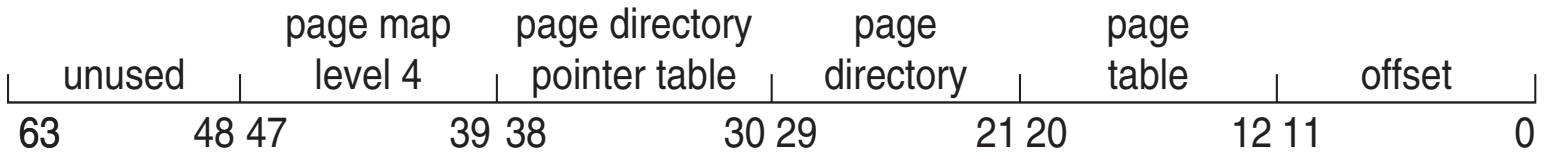


Tabla de páginas invertida

- 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.
- Grandísimo ahorro en el espacio consumido por las TP.

Tabla de páginas invertida

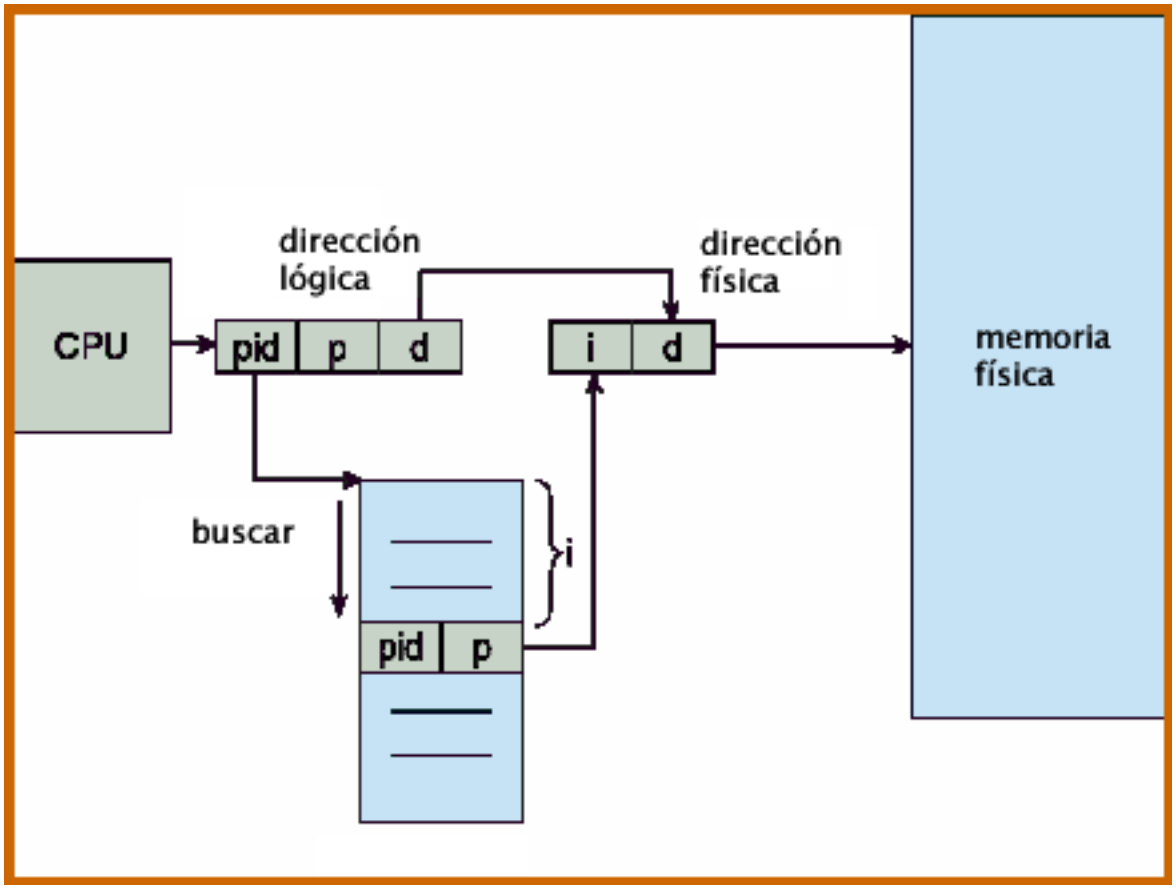


Tabla de páginas invertida

- Ventaja: reduce la cantidad de memoria necesaria para la TP
- Desventaja: aumenta muchísimo el tiempo de acceso a la TP
  - Confiamos en los aciertos de la TLB
  - O bien implementamos la tabla inversa como una tabla de índices hash

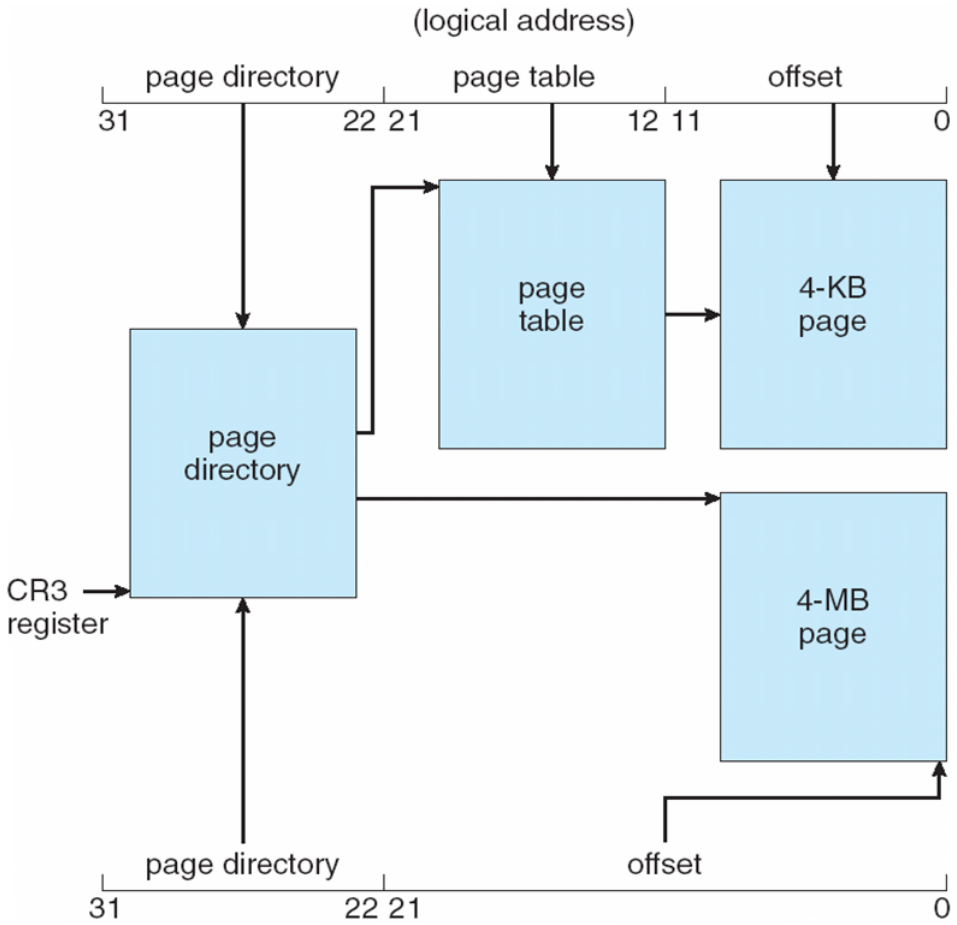
Híbrido segmentado/paginado

- La paginación y la segmentación pueden combinarse (ej. MULTICS, x86).
- Motivación: aprovecharse de las ventajas que ofrecen los esquemas por separado
  - Segmentación: flexibilidad y facilidad para la organización lógica, compartición, protección…
  - Paginación: solución eficiente para la fragmentación

Híbrido segmentado/paginado: Intel x86

Tamaños de páginas variables

- Ejemplo: x86



Sistemas Operativos
Tema 4. Memoria

Fin del primer bloque

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA