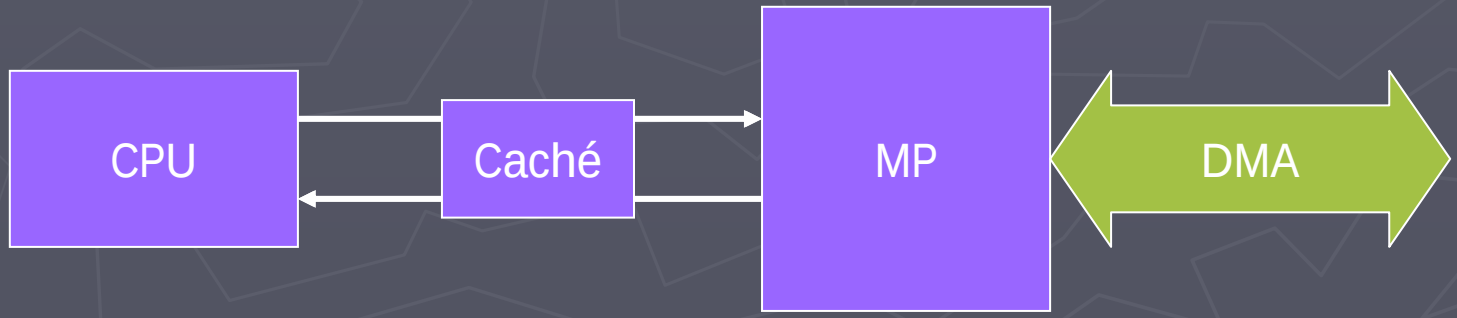
Sistemas operativos

Tema 7: Gestión de memoria

La memoria principal

- La memoria es el componente central en la arquitectura de un computador.
  - A menudo, el único dispositivo de almacenamiento al que la CPU puede acceder directamente.
  - Un vector de palabras, cada una con su propia dirección física.
- La CPU lee instrucciones y lee o modifica datos de la memoria durante cada ciclo de instrucción.
- Las operaciones de E/S con DMA (*acceso directo a memoria*) escriben/leen datos en memoria.



Gestión de memoria

- Asignar y liberar espacio en memoria según se necesite.
  - La política empleada repercute en la utilización de la CPU y los tiempos de respuesta del sistema.
  - Reubicación: vinculación del espacio de direcciones lógicas de cada proceso con direcciones físicas concretas.
- Seguir la pista de qué partes de la memoria están siendo usadas y por qué procesos.
  - Protección: impedir accesos (accidentales o malintencionados) a la memoria propia de otros procesos o del SO.
  - Compartición: permitir el acceso de varios procesos a zonas de memoria comunes.
    - Comunicación entre procesos, mismo código para varias instancias de un mismo programa, etc.

Sobre la reubicación

- Reubicación estática en tiempo de compilación.
  - Dirección lógica = dirección física.
  - El compilador genera código máquina con direcciones absolutas.
- Reubicación estática durante la carga.
  - Dirección lógica = dirección física.
  - El código máquina contiene direcciones relativas al comienzo del programa (código reubicable), que se vinculan a direcciones físicas al comenzar su ejecución.
- Reubicación dinámica.
  - Dirección lógica ≠ dirección física.
  - Código reubicable.
  - Las direcciones físicas se calculan en tiempo de ejecución, por medio de un soporte *hardware* especializado (unidad de gestión de memoria, MMU).

Sobre la protección

- En general, la ubicación de un proceso en memoria no se conoce de antemano, y no es posible anticipar todas sus referencias a memoria.
  - Muchas las referencias se calculan dinámicamente (e.g. a raíz de instrucciones *malloc*).
- La protección requiere mecanismos que actúen en tiempo de ejecución.
  - La solución prácticamente universal es proveerla por *hardware*, en la propia MMU.

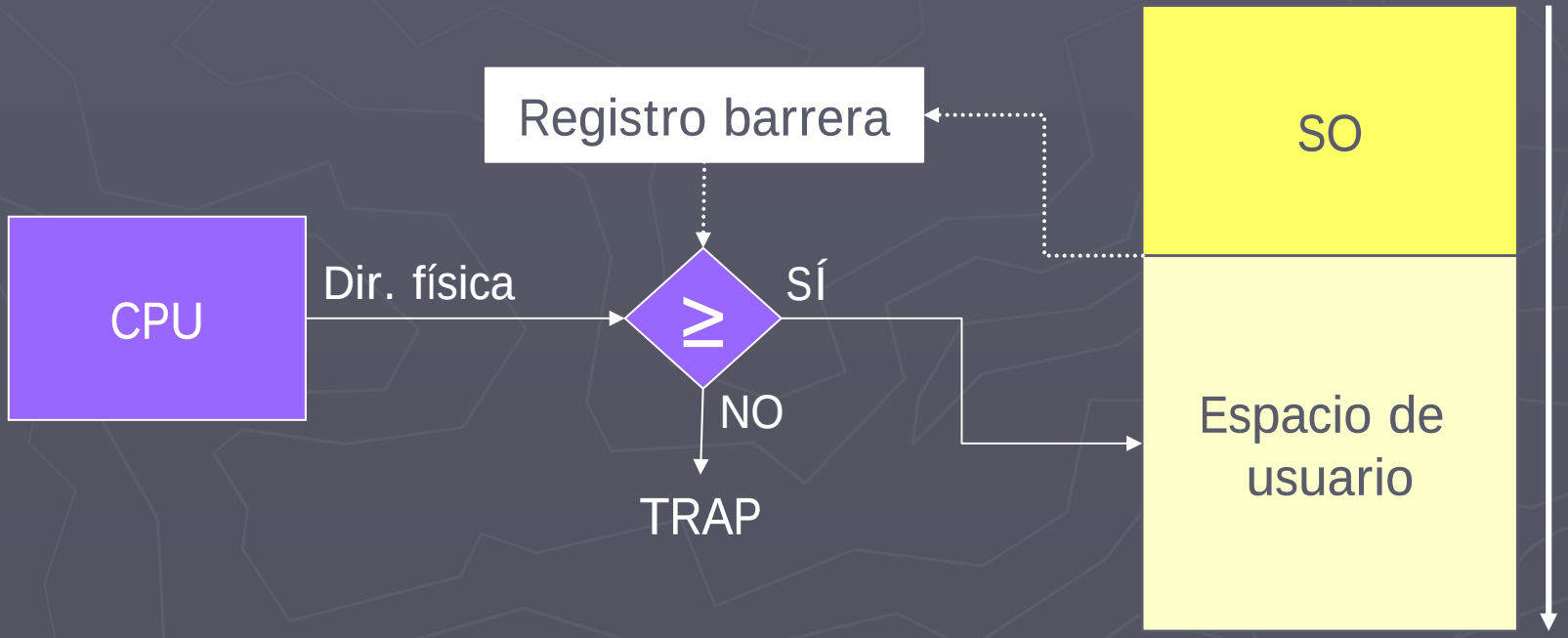
Técnicas de gestión de memoria

- Monitor residente.
- Asignación contigua con particiones múltiples.
  - Particiones estáticas.
  - Particiones dinámicas.
- Asignación no contigua.
  - Paginación.
  - Segmentación.
  - Segmentación paginada.

Monitor residente

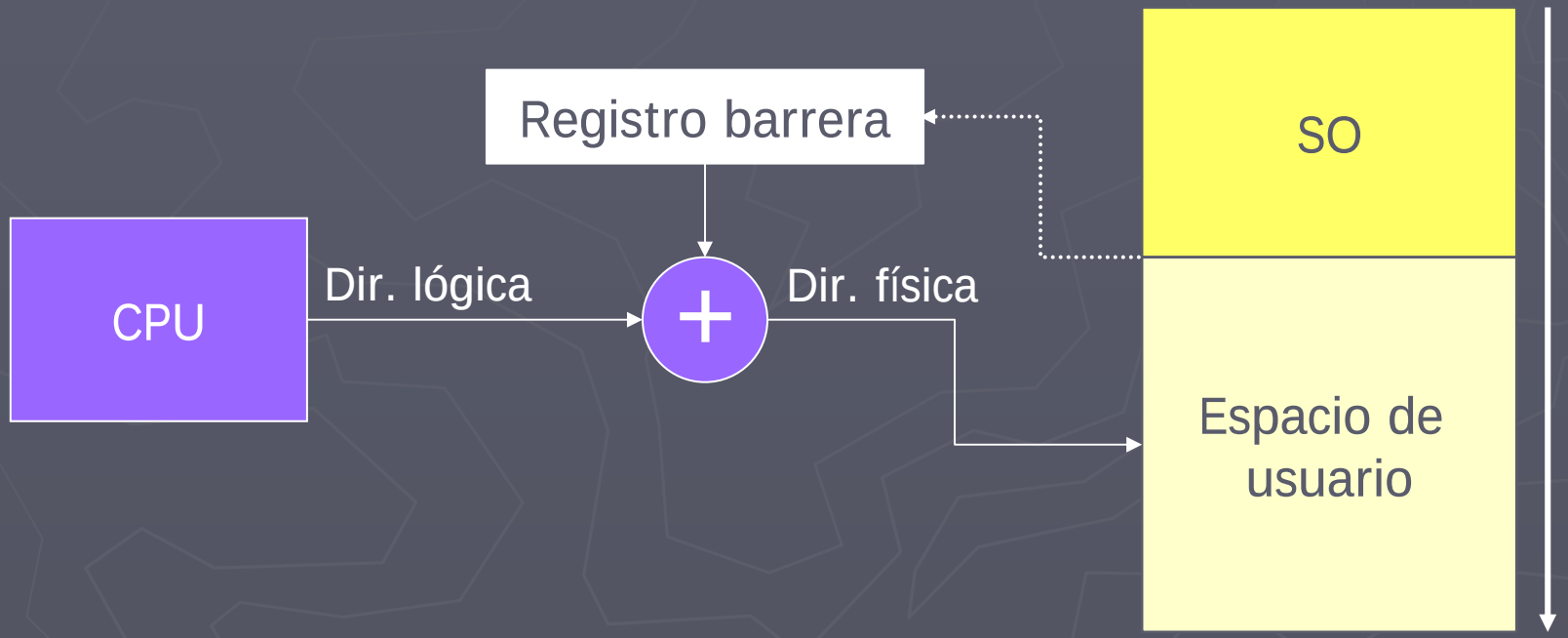
Monitor residente

- Se divide la memoria en dos zonas: monitor residente (SO) y espacio de usuario.
- Con reubicación estática:



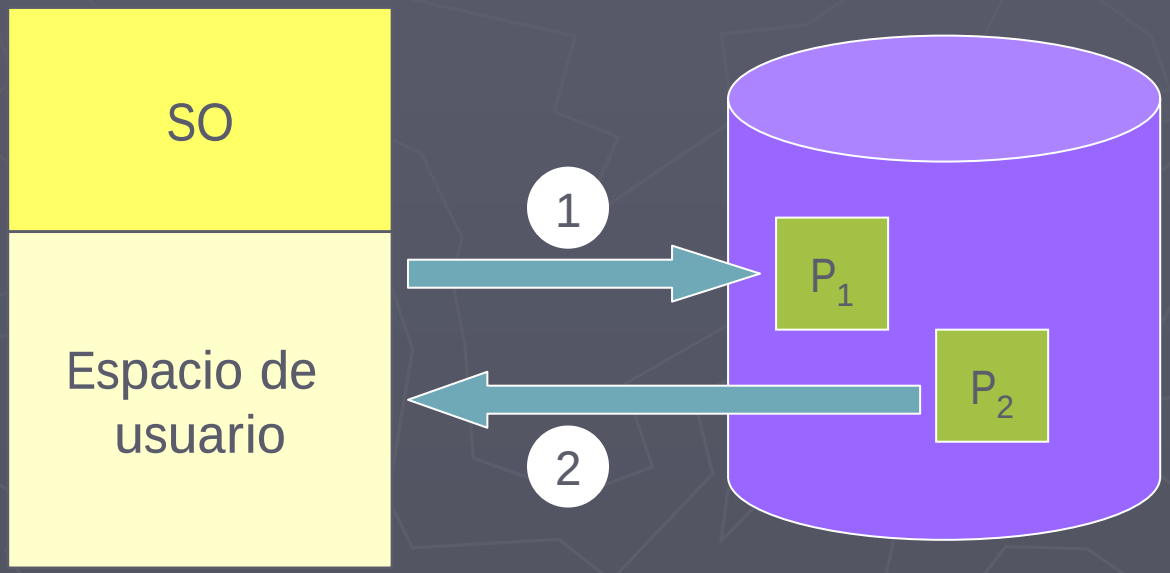
Monitor residente

- Con reubicación dinámica:



Intercambio *(swapping)*

- Forma de conseguir multiprogramación, utilizando almacenamiento secundario de apoyo.



Intercambio

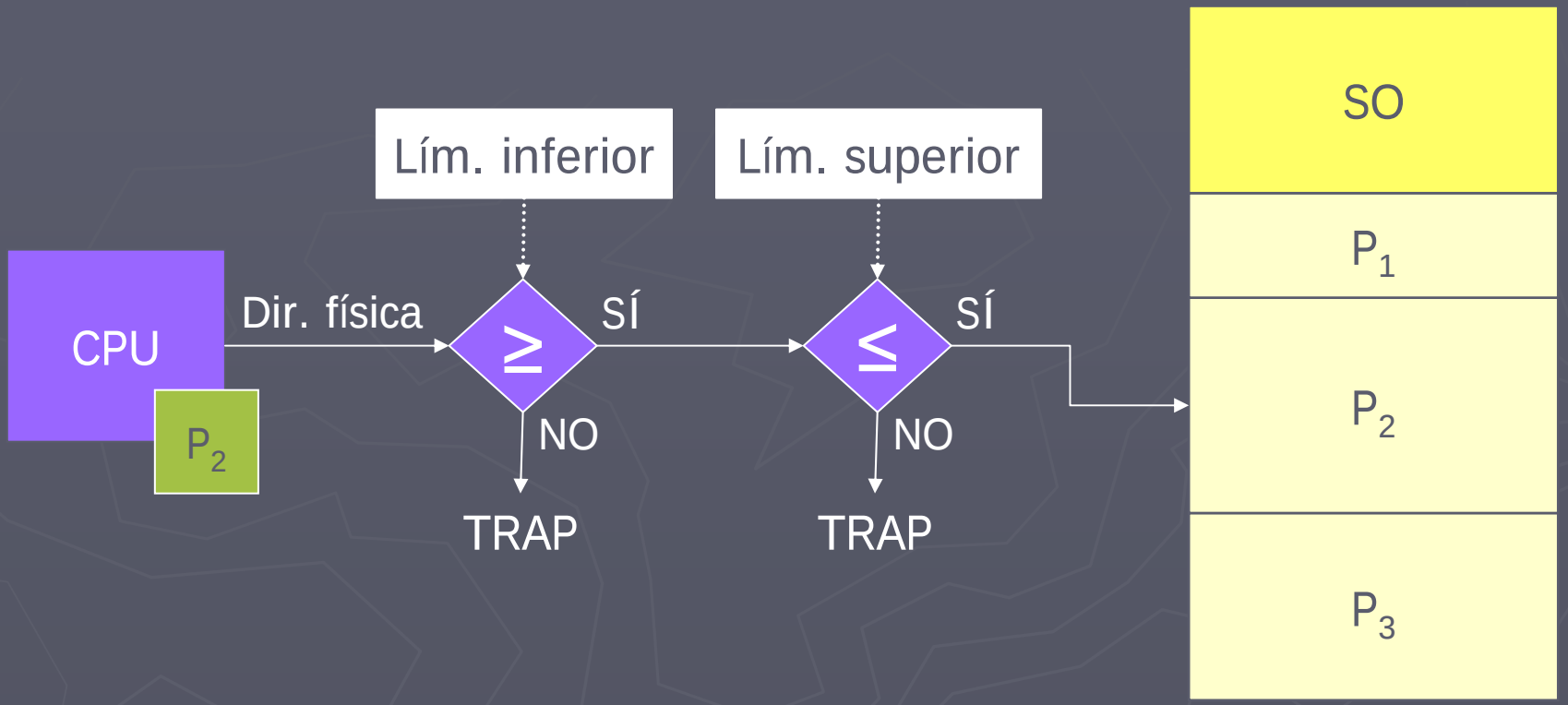
- Tiempos de cambio de contexto muy elevados.
  - Mejoran solapando la carga o descarga de un proceso con la ejecución de otro.
    - *Buffers* del SO, e intercambio dentro de la memoria principal.
- Cuidado con las operaciones de E/S por DMA.
  - Podrían iniciarse sobre la memoria de un proceso y terminar sobre la de otro.
  - Soluciones:
    - No descargar procesos con E/S pendiente.
    - Realizar la E/S sobre *buffers* del SO.

Asignación contigua con particiones múltiples

Sistemas de particiones múltiples

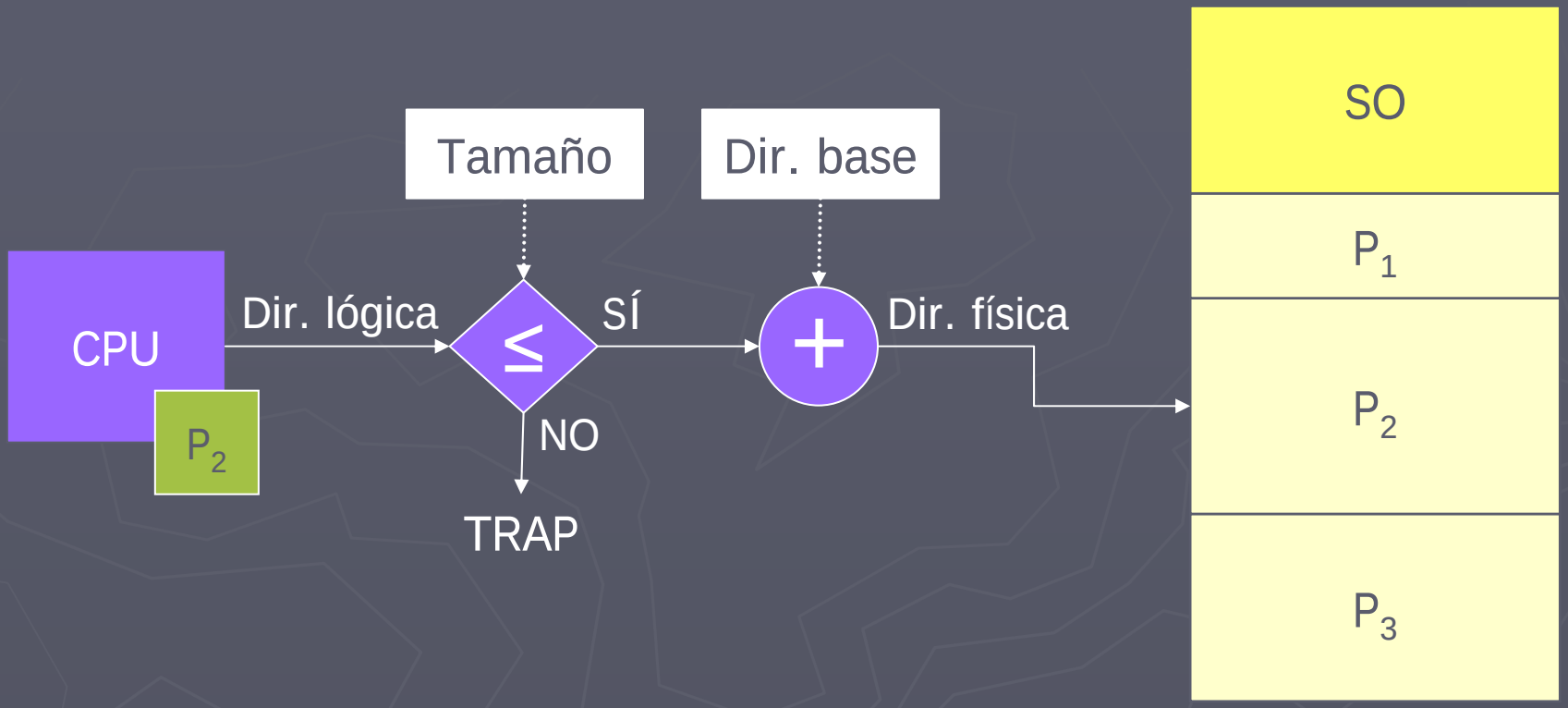
- Objetivo: soporte más eficiente a la multiprogramación.
- Se divide el espacio de usuario en un conjunto de regiones o particiones.
  - Cada región alberga un proceso.
- Dos variantes:
  - *Multiprogramación con número fijo de tareas* (MFT): particiones estáticas.
  - *Multiprogramación con número variable de tareas* (MVT): particiones dinámicas.

Con reubicación estática



- Los límites se actualizan en los cambios de contexto, con la información del bloque de control del proceso en cuestión.

Con reubicación dinámica

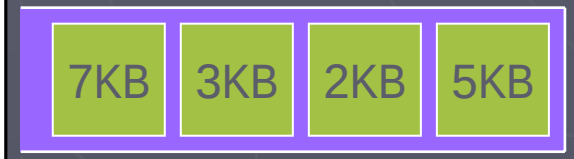


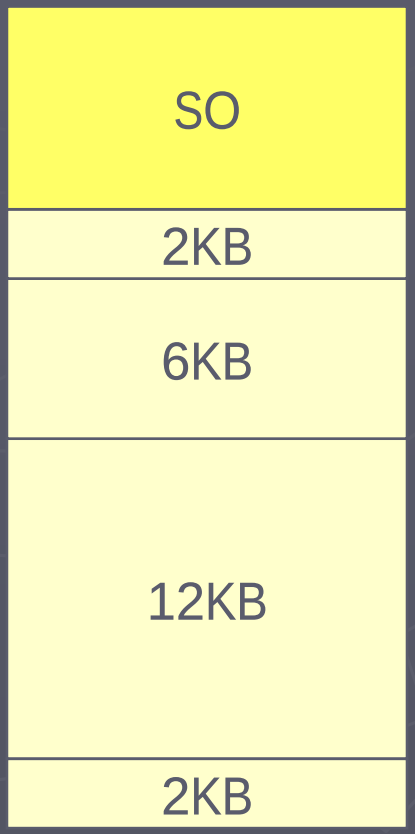
- El tamaño y la dirección base se actualizan en los cambios de contexto.

MFT

- Las particiones de la memoria son fijas, definidas *a priori*.
- Asignación de procesos a particiones:
  - Una cola de procesos por partición.
    - Estrategia *best-fit:* cada proceso se asigna a la partición más pequeña que puede albergarlo.
    - Posibilidad de particiones vacías aún habiendo procesos preparados que cabrían en ellas.
  - Una única cola.
    - Estrategia *best-fit-only*.
    - Estrategia *best-fit-available:* los procesos entran en la partición más pequeña en que caben de entre las disponibles.

Ejemplo





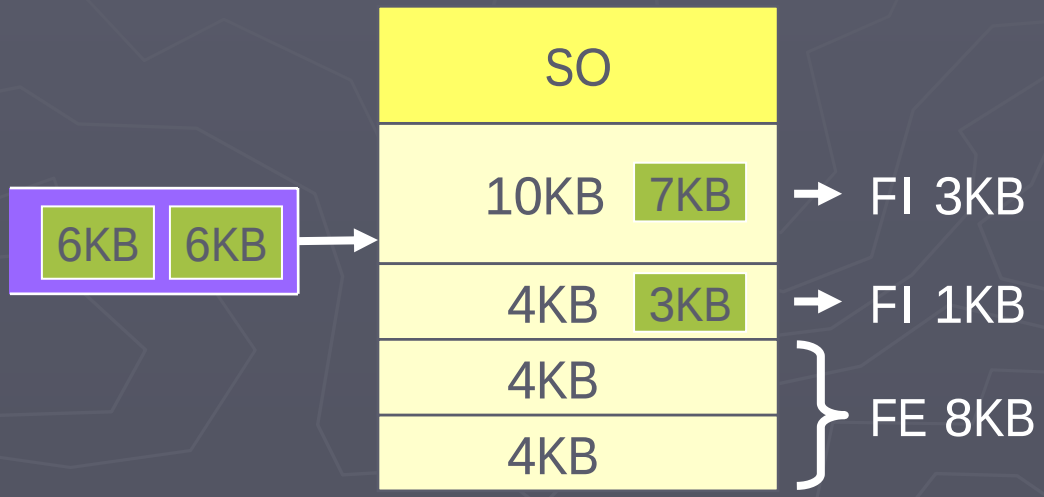
- El proceso de 5KB entra en la partición de 6KB.
- El proceso de 2KB entra en una partición de 2KB.
- *Best-fit-only:*
  - El proceso de 3KB espera.
  - Con exploración de la cola, el proceso de 7KB entra en la partición de 12KB.
- *Best-fit-available:*
  - El proceso de 3KB entra en la partición de 12KB.

Más sobre MFT

- Utilizando intercambio, se puede aumentar el grado de multiprogramación más allá del número de particiones.
  - Los procesos pueden residir en particiones distintas durante su ejecución sólo con reubicación dinámica.
- ¿Qué hacer si un proceso solicita más memoria, excediendo el tamaño de la partición asignada?
  - No conceder más memoria y abortar la ejecución.
  - Encolar el proceso en espera de una partición suficientemente grande.
    - Exige reubicación dinámica.

Fragmentación

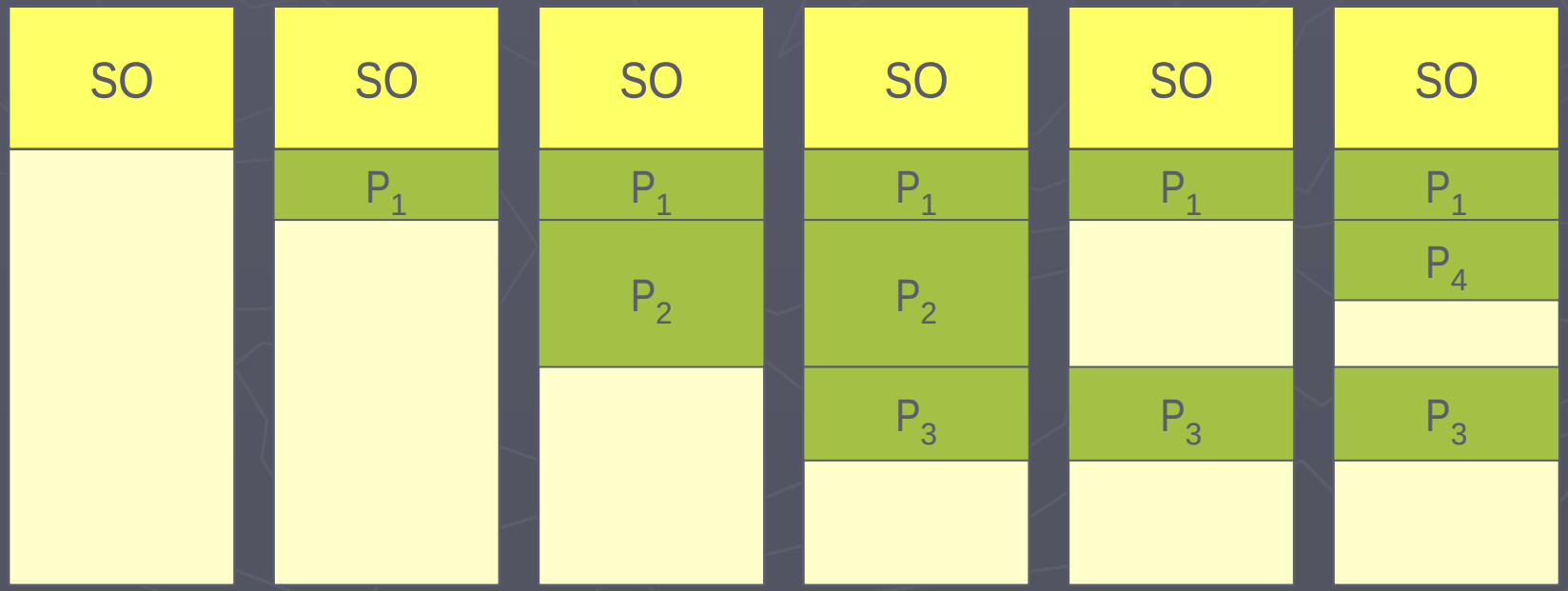
- Aprovechamiento subóptimo de la memoria.
- Fragmentación interna: memoria asignada que no se utiliza.
- Fragmentación externa: memoria desocupada que no puede aprovecharse para dar cabida a nuevos procesos.



- En MFT no habría fragmentación si las particiones coincidieran con el tamaño de los procesos.

MVT

- El tamaño y número de las particiones de la memoria varía dinámicamente.
  - Mayor flexibilidad que MFT.
  - Elimina la fragmentación interna: a cada proceso se le asigna únicamente la memoria que va a utilizar.



Hardware y software para MVT

- Mismo *hardware* que en MFT.
  - Predominantemente, reubicación dinámica.
- El SO mantiene una lista de huecos, y asigna memoria según una determinada estrategia:
  - *First-fit:* selecciona el primer hueco suficientemente grande.
  - *Best-fit:* selecciona el hueco más pequeño de tamaño suficiente.
    - Hay que explorar toda la lista, salvo que esté ordenada por tamaño.
    - Tiende a generar huecos pequeños, cuyo rastreo es comparativamente costoso.
  - *Worst-fit:* selecciona el hueco más grande.
    - Hay que explorar toda la lista, salvo que esté ordenada por tamaño.
    - Pretende crear huecos grandes.

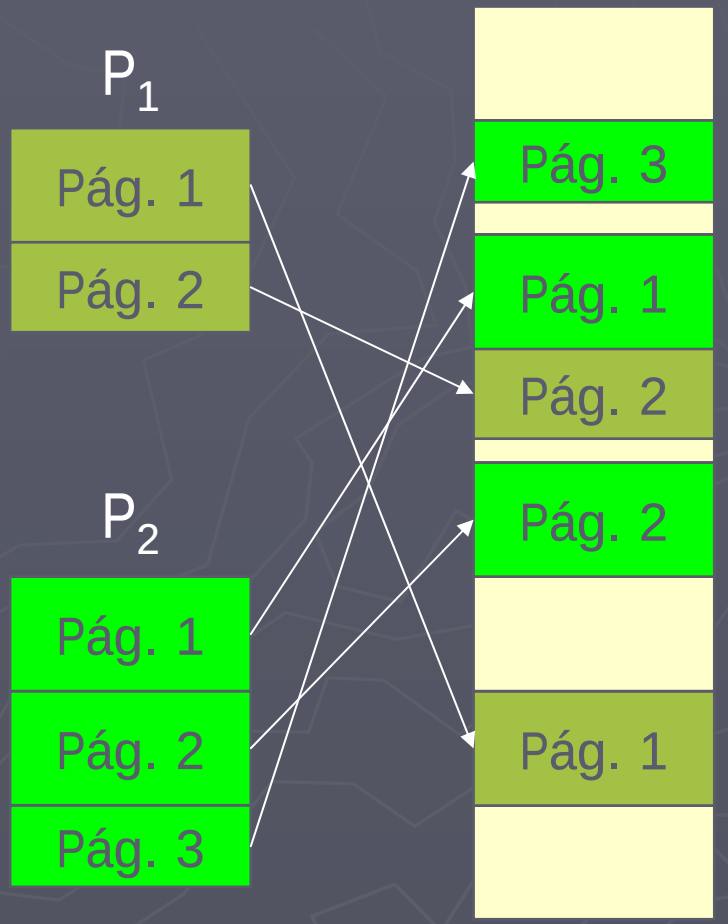
Compactación

- La fragmentación externa en MVT puede ser grande, y en general aumenta con el tiempo.
  - Caso extremo: un hueco desperdiciado entre cada dos procesos.
- El problema se alivia con compactación, desplazando procesos en un sentido y huecos en otro.
  - El desplazamiento obliga a detener la computación.
  - Se necesita reubicación dinámica.
  - ¿Cuándo compactar?
    - Cuando un proceso no pueda alojarse en memoria.
    - De modo preventivo, cuando se exceda un determinado nivel de fragmentación.
    - Cuando se realice algún intercambio.

Paginación

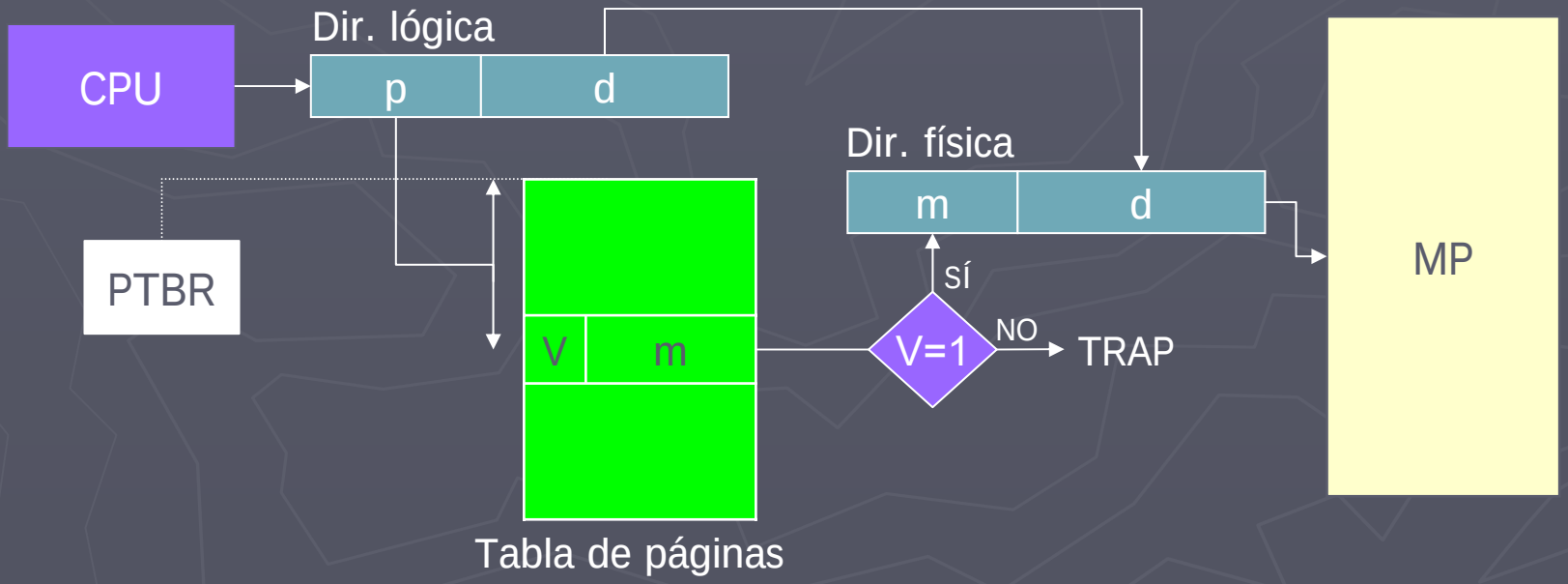
Paginación

- El espacio lógico de los procesos se divide en páginas de tamaño fijo.
- La memoria se divide en marcos, del mismo tamaño que las páginas.
- Las páginas se asignan a marcos, posiblemente de manera no contigua.



Hardware de paginación

- Las direcciones lógicas se descomponen en número de página (p) y desplazamiento dentro de la misma (d).
- La MMU vincula números de página a marcos concretos de la memoria física.



Observaciones

- El tamaño de páginas y marcos es potencia de 2.
- Igual número de bits para p y para m: los procesos no pueden direccionar más memoria que la que hay.
- Hay una tabla de páginas para cada proceso.
- Cuando hay pocas páginas, las tablas pueden alojarse en registros de la CPU. Lo habitual, sin embargo, es almacenarlas en la propia memoria.
  - En memoria, cada entrada de la tabla ha de ocupar un número entero de palabras.

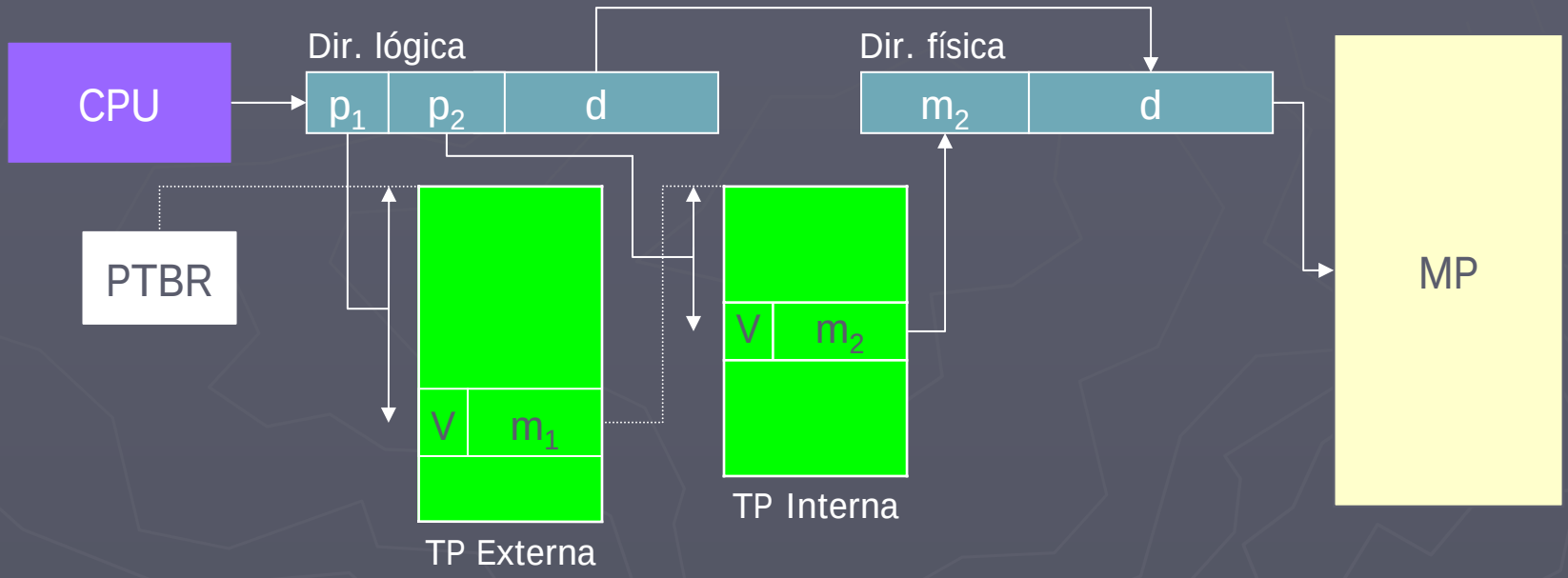
Puntos fuertes

- No hay fragmentación externa, y la asignación de memoria es trivial.
  - Cualquier marco libre es igualmente bueno.
- La fragmentación interna se reduce a la última página de cada proceso.
  - Mejor aprovechamiento de la memoria cuanto más pequeñas sean las páginas.
    - Caso peor: (1 página - 1 palabra) × número de procesos.
    - En media: (½ página) × número de procesos.
- Se facilita la compartición: varios procesos pueden acceder a un mismo marco.
  - El código compartido debe ser reentrante (i.e. no debe automodificarse).

Puntos débiles

- Si las tablas de páginas se alojan en memoria, se duplica el tiempo de acceso efectivo a la misma.
  - Solución: una caché especial para entradas de las tablas de páginas (*Translation Lookaside Buffer*, TLB).
- Las tablas de páginas pueden ocupar un espacio considerable.
  - Ejemplo: memoria de 4GB, páginas de 4KB y 4 *bytes* por entrada → tablas de 4MB por proceso.
  - Parece recomendable un tamaño de páginas grande, para que las tablas tengan pocas entradas.
    - Compromiso con la fragmentación interna.
  - Soluciones:
    - Paginación multinivel.
    - Tabla de páginas invertida.

Paginación en 2 niveles



- A menudo, los procesos no utilizan todas las páginas del espacio lógico.
  - Las páginas no utilizadas se pueden condensar en entradas de la TPE con V=0.
  - Ejemplo (continúa):
    - Memoria de 4GB y páginas de 4KB
    - 4 *bytes* por entrada en TPE y TPI
    - 10 bits para p_1 y 10 para p_2
    - Procesos de 12 MB

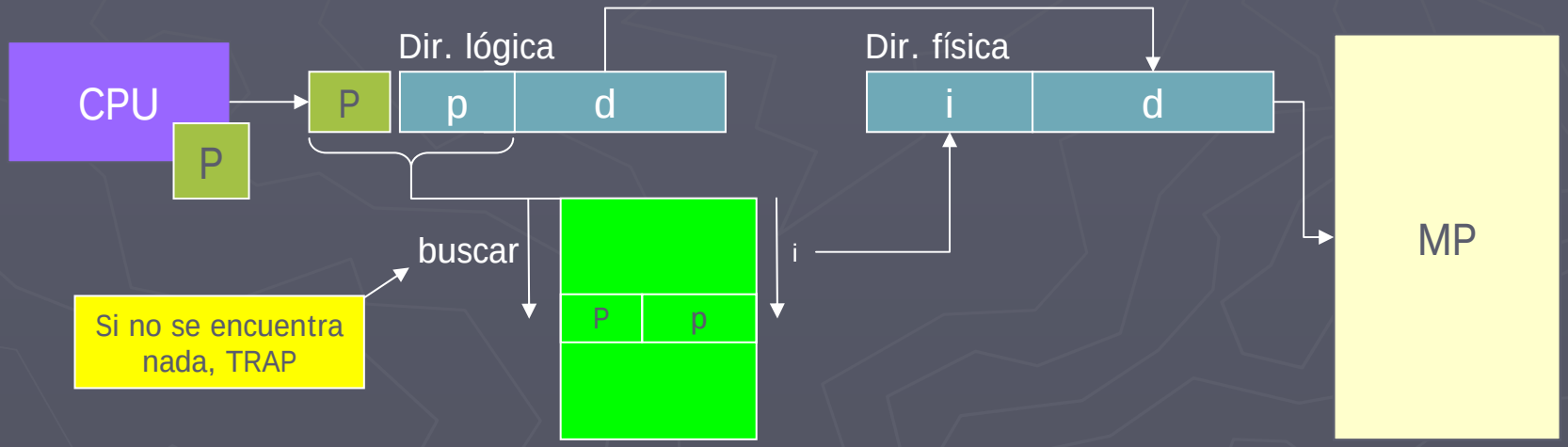
} 16KB en tablas por proceso

Observaciones

- El número de bits de p_2 se ajusta para aprovechar al máximo el marco de cada TPI.
- Si las tablas de páginas se alojan en memoria, se triplica el tiempo de acceso efectivo a la misma.
- Se pueden compartir TPIs.
- La idea es generalizable a más de 2 niveles.

Tabla de páginas invertida

- Una única tabla, en memoria, con tantas entradas como marcos.
  - Cada entrada indica qué página de qué proceso se aloja en el marco correspondiente.



- Tantos accesos a memoria como entradas se recorren en la búsqueda.
  - Una tabla *hash* limita la búsqueda a un número reducido de entradas.
- Se dificulta la compartición.

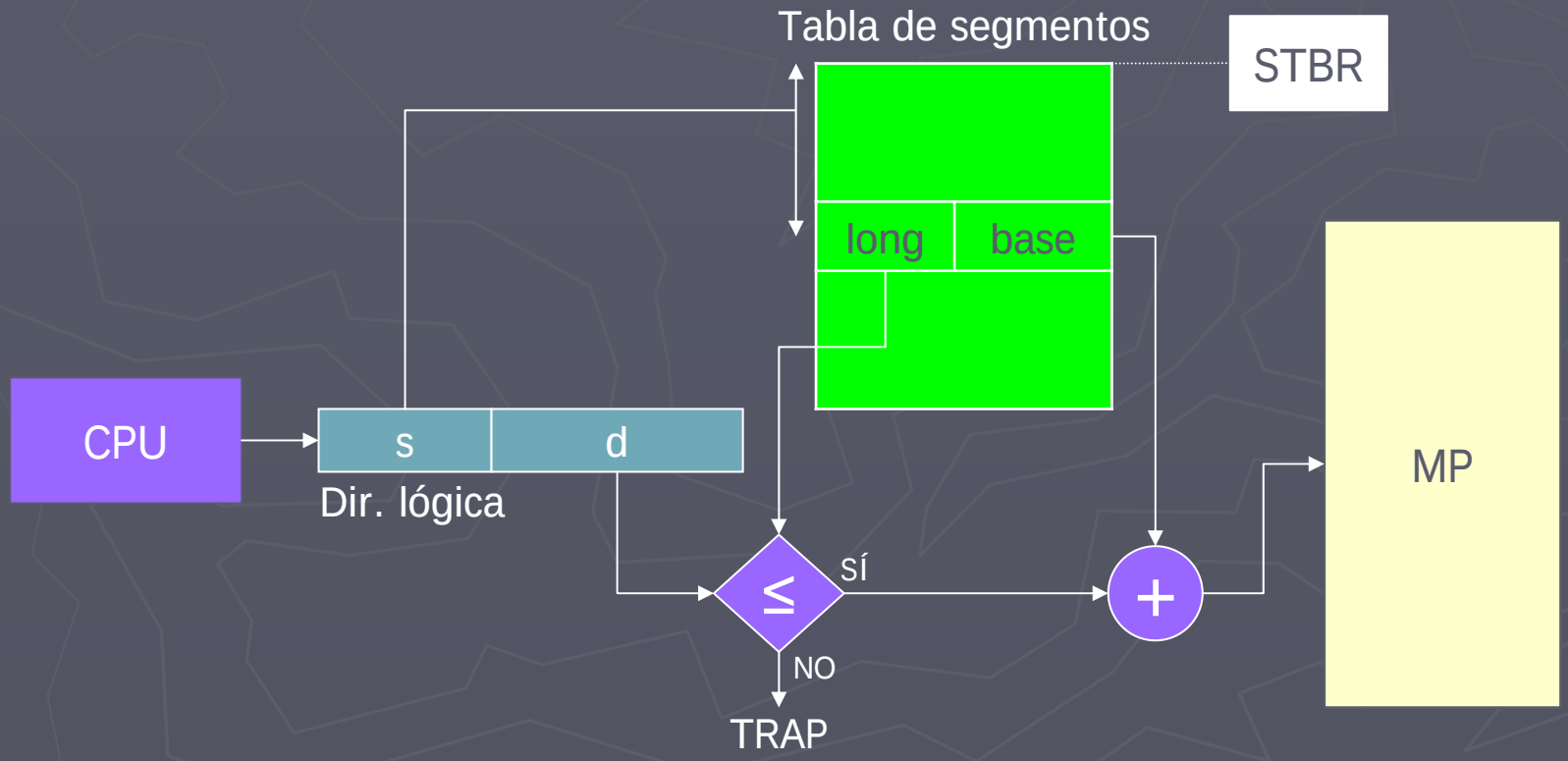
Segmentación

Segmentación

- El espacio lógico de un proceso se concibe como un conjunto de segmentos de tamaño variable.
  - Funciones, variables, pila, etc.
- Similar a la MVT, sólo que los distintos segmentos de un proceso no tienen por qué estar contiguos en memoria.

Hardware de segmentación

- Las direcciones lógicas se descomponen en número de segmento (s) y desplazamiento dentro del mismo (d).



Observaciones

- Los campos long y d tienen igual número de bits.
- El campo base tiene tantos bits como las direcciones de memoria física.
- No se necesita bit de validez.
  - La protección la proporciona el campo long.
- Hay una tabla de segmentos para cada proceso.
- Nuevamente, las tablas suelen almacenarse en memoria, con cada entrada ocupando un número entero de palabras.

Puntos fuertes

- Se divide el espacio lógico en partes semánticamente definidas.
  - Optimiza la protección y la compartición.
- No hay fragmentación interna.

Puntos débiles

- Se duplica el tiempo de acceso efectivo a la memoria.
  - Solución: una caché especial para entradas de las tablas de segmentos.
- Las tablas de segmentos pueden ocupar un espacio considerable.
- Hay fragmentación externa.
  - Política de asignación *first-fit*, *best-fit* o *worst-fit*.
  - Compactación.
  - Reducir el tamaño medio de los segmentos aumentaría el espacio consumido en tablas.

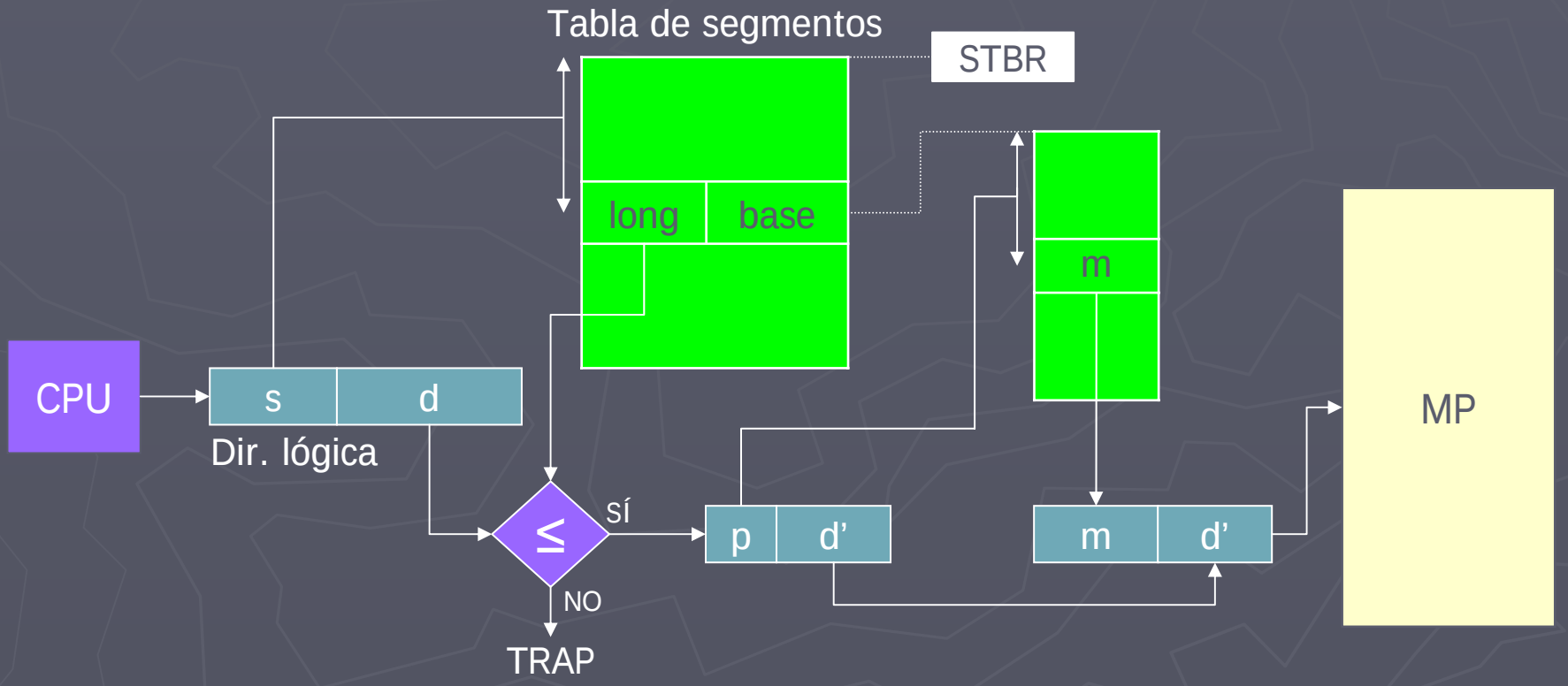
Segmentación paginada

Segmentación paginada

- Combinación de segmentación y paginación.
  - Los segmentos se dividen en páginas y se meten en marcos.
- Se mantiene la visión natural del espacio lógico de los procesos.
- Se trivializa la asignación (cualquier marco es igualmente bueno), y se elimina la fragmentación externa.

Hardware de segmentación paginada

- El desplazamiento dentro del segmento se descompone en un número de página (p) y un desplazamiento dentro de ésta (d').



Observaciones

- Una tabla de segmentos por proceso, y una tabla de páginas por segmento.
- Se introduce fragmentación interna, en la última página de cada segmento.
- Con todas las tablas en memoria, se triplica el tiempo efectivo de acceso a la misma.
- En comparación con la segmentación, aumenta el espacio consumido en tablas.

- La paginación segmentada (paginación segmentando la tabla de páginas) tiene poco sentido práctico, y apenas se usa.

Fin