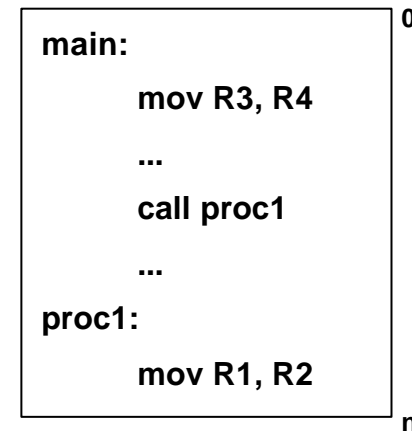
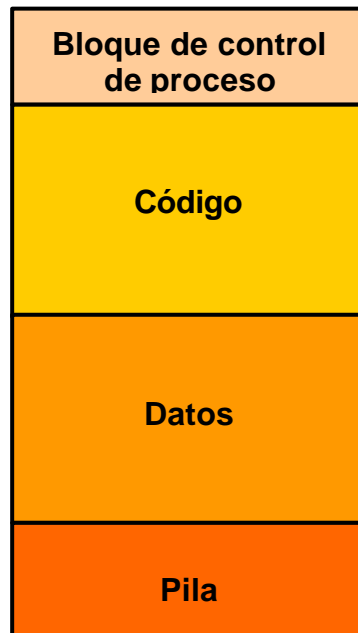
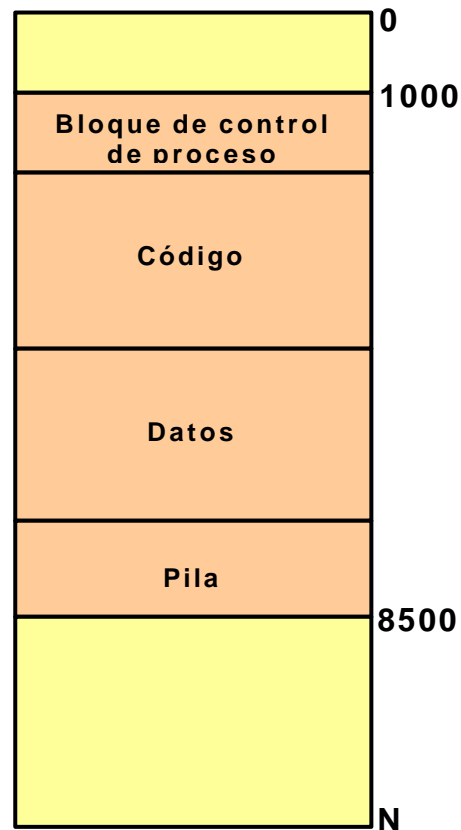


Conceptos básicos

- La gestión de la memoria se encarga de administrar el recurso “memoria” en un sistema donde se ejecutan varios procesos.
- El espacio asociado a un proceso se contempla como un conjunto de referencias a instrucciones o datos (espacio de direcciones lógicas)



- El espacio de direcciones físico es la zona de memoria física donde se ubica el proceso para su ejecución.

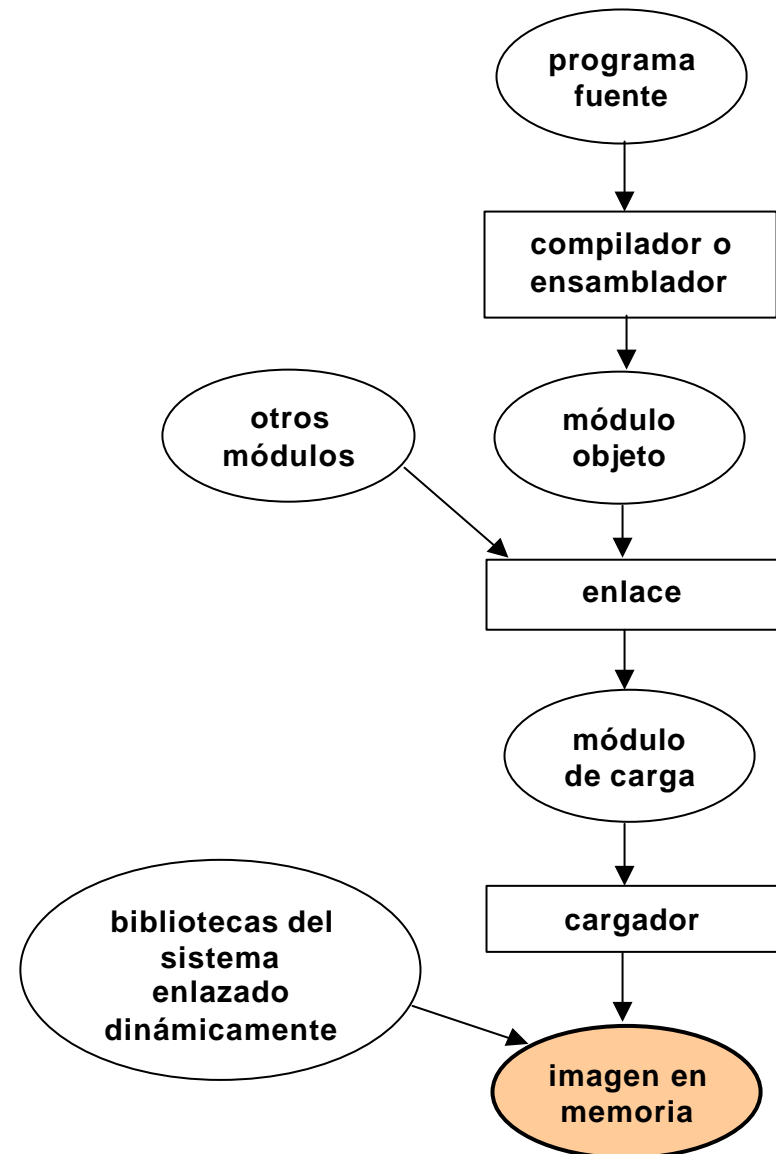


Problemas fundamentales

- **Reubicación:** las referencias de un proceso deben adaptarse a la ubicación de memoria donde éste se ejecuta.
- **Escasez:** la memoria requerida por los procesos puede ser mayor que la memoria física disponible.
- **Protección:** la zona de memoria asignada a cada proceso es privada y no debe ser “invadida” por otros procesos.
- **Compartición:** se reduce la ocupación de memoria si varios procesos comparten datos o código.
- **Organización:** la memoria física puede ser asignada a los procesos de forma contigua o dispersa.

Problema de reubicación

- Todas las referencias a instrucciones o datos (direcciones lógicas) realizadas en un proceso tienen que ser traducidas a direcciones absolutas o físicas.
- El problema de reubicación puede resolverse en cualquiera de las siguientes fases:
 - ✓ Programación
 - ✓ Compilación
 - ✓ Carga
 - ✓ Ejecución

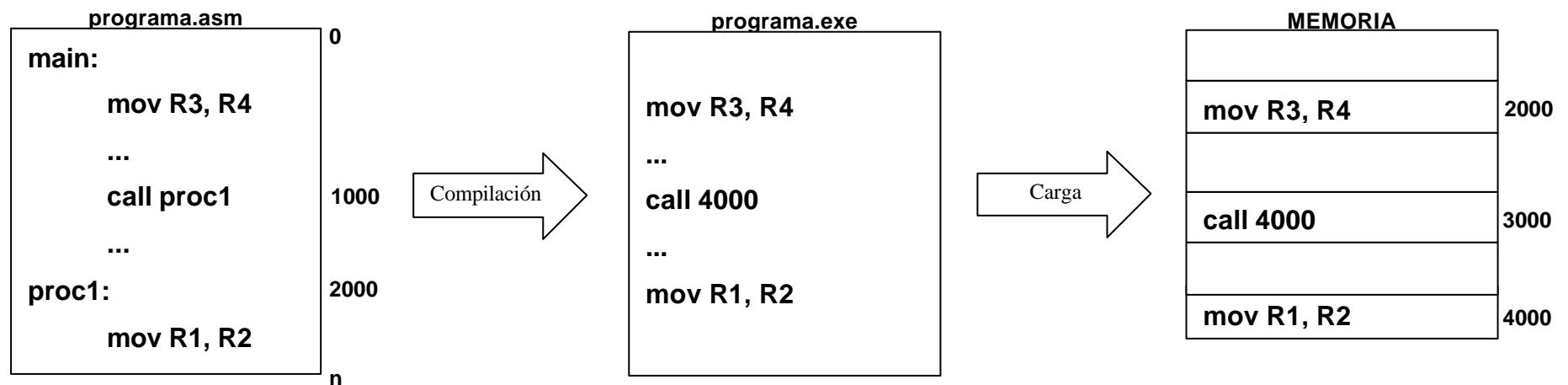


Reubicación en tiempo de programación

- El programador especifica directamente todas las direcciones físicas reales en el propio programa.

Reubicación en tiempo de compilación

- La asignación de direcciones físicas se produce durante la compilación del programa. Es necesario saber dónde va a estar ubicado el proceso.
- El proceso debe ser recompilado, si finalmente se asigna a una ubicación diferente de memoria.



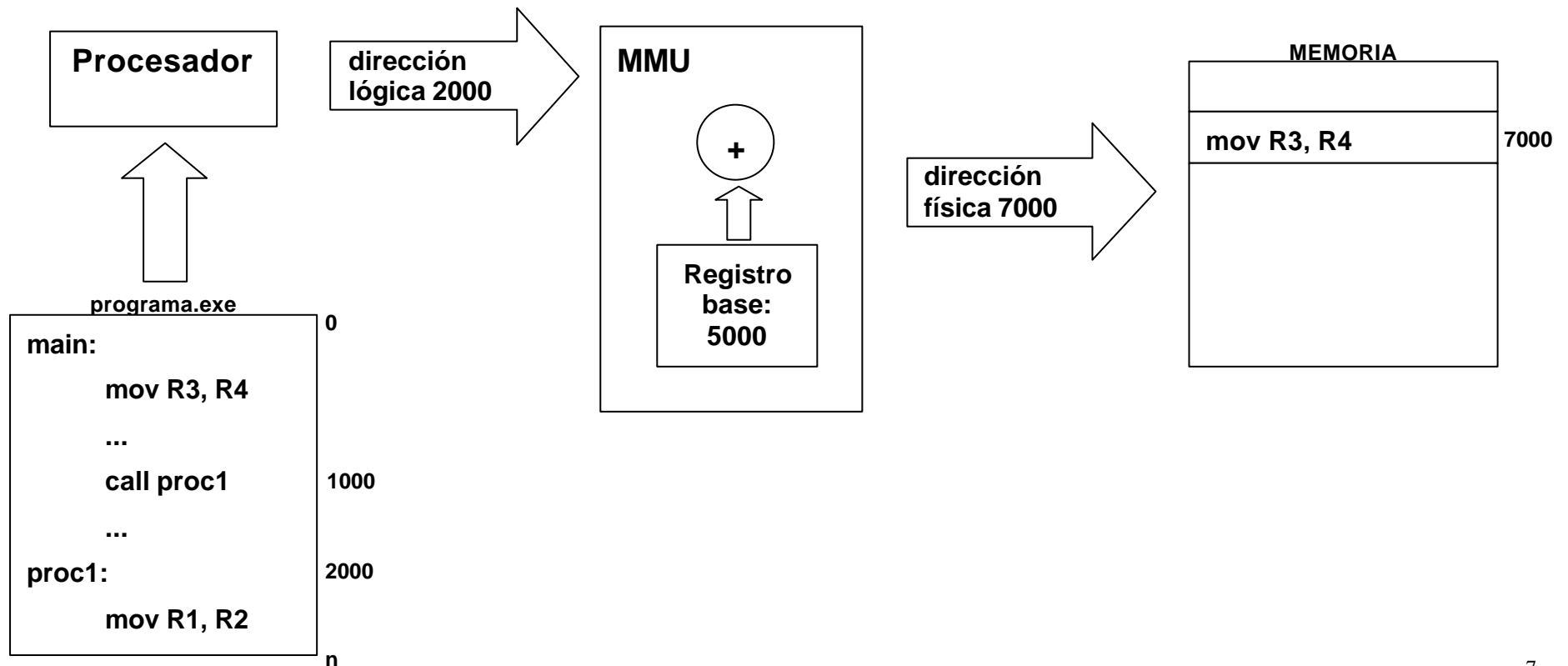
Reubicación en tiempo de carga

- El cargador lee un fichero ejecutable y lo carga en memoria
- El cargador puede detectar las referencias no resueltas en tiempo de compilación y resolverlas en función de la ubicación donde se cargue el fichero ejecutable



Reubicación en tiempo de ejecución

- El programa cargado conserva direcciones relativas.
- Necesita el soporte de hardware específico (MMU: Unidad de Manejo de Memoria) que permita traducir direcciones lógicas a direcciones físicas de forma dinámica.



Problema de escasez de memoria

- Los requerimientos de memoria de los procesos tienden a superar la cantidad de memoria física disponible.
- Técnicas para solucionarlo:
 - ✓ Superposiciones (overlays)
 - ✓ Intercambios (swapping)
 - ✓ Memoria virtual
 - ✓ Carga y enlace dinámicos

Método de superposiciones

- Un proceso puede fragmentarse en partes denominadas superposiciones (overlays).
- Sólo se mantienen en memoria las partes necesarias en cada momento.
- Inconveniente: debe ser manejado por el diseñador del programa.
- Ejemplo: compilador de dos pasadas.

Paso 1: 70K

Memoria disponible: 150K

Paso 2: 80K

Memoria necesaria: 200K

Tabla de símbolos: 20K

Paso 1: 120K

Rutinas comunes: 30K

Paso 2: 130K

Método de intercambios (swapping)

- Técnica para poder ejecutar concurrentemente más procesos que los que caben en memoria física.
- Inconveniente: hace más complejos y costosos los cambios de contexto, sobre todo si el tamaño de los procesos es grande.

Memoria virtual

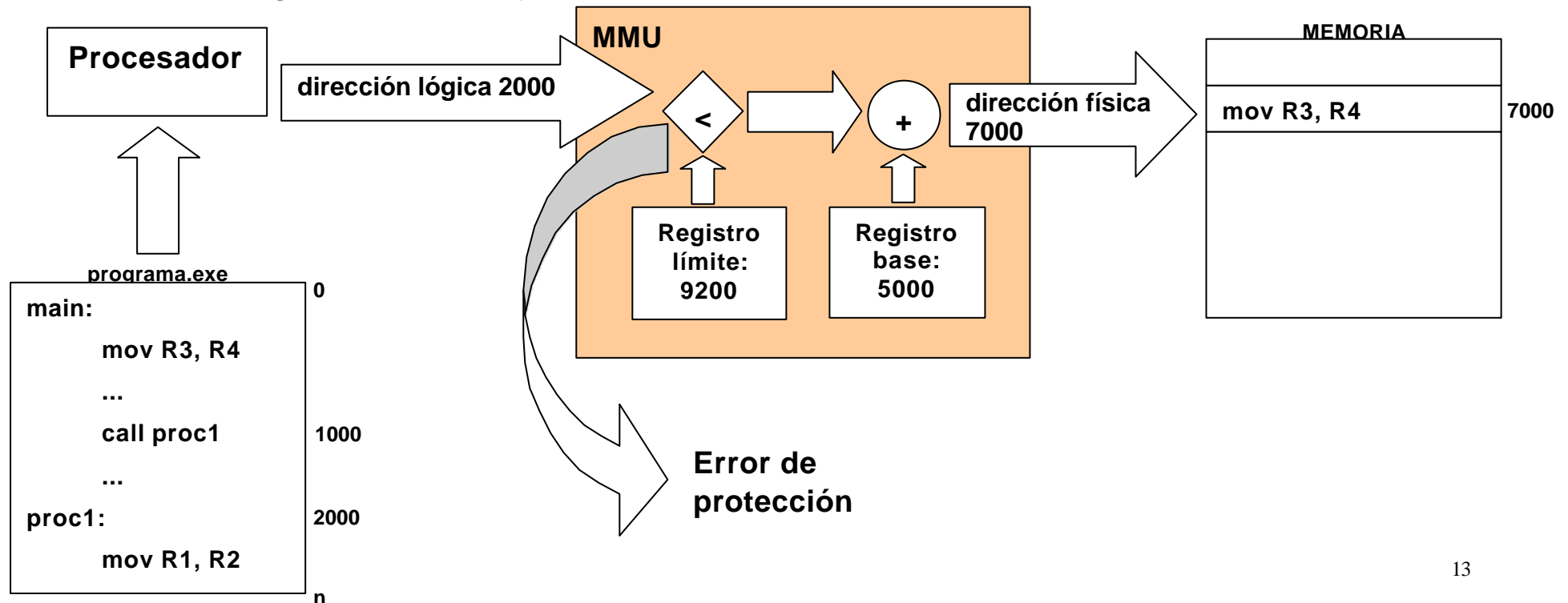
- La alternativa consiste en intercambiar “fragmentos” de los procesos.
- Para ello se requieren técnicas de gestión basadas en el uso de memoria virtual:
 - ✓ que permitan la división del espacio del proceso y su asignación a zonas de memoria física no continuas
 - ✓ pueden utilizarse zonas de disco (memoria secundaria) como complemento del espacio de memoria primaria disponible

Método de carga y enlace dinámico

- Se puede retrasar tanto el enlace como la carga de una biblioteca hasta el momento de ejecución. Por tanto, no es necesario que las bibliotecas permanezcan en memoria.
- También permite la compartición de las bibliotecas cargadas.
- Ejemplo: DLL en entorno Windows.

Problema de protección de memoria

- Cada proceso debe tener un espacio de direcciones (o zona de memoria) propio.
- Los procesos sólo pueden acceder a su propio espacio de direcciones salvo en aquellos casos que el sistema permita compartir zonas de memoria.
- Se utilizan registros base y límite.



Compartición

- Procesos cooperativos pueden compartir código o datos.
- El gestor de memoria debe permitir el acceso a áreas compartidas de memoria sin comprometer la protección básica.

- Ejemplo:

Código: 50K

Usuarios: 20

Datos: 10K

Memoria necesaria: $20 * (50K + 10K)$

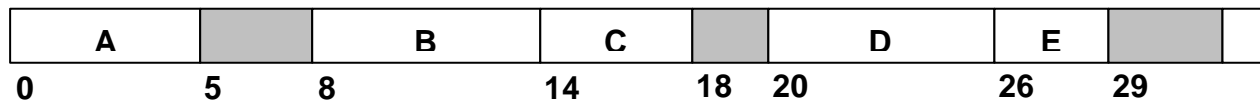
Con compartición: $50K + 20 * 10K$

Organización de la memoria física

- **Asignación contigua:** las direcciones físicas de un mismo proceso son contiguas:
 - ✓ Particiones fijas
 - ✓ Particiones variables
- **Asignación dispersa:** las direcciones físicas de un mismo proceso no tienen por qué ser contiguas:
 - ✓ Paginación
 - ✓ Segmentación
 - ✓ Segmentación paginada

Gestión de la ocupación de memoria

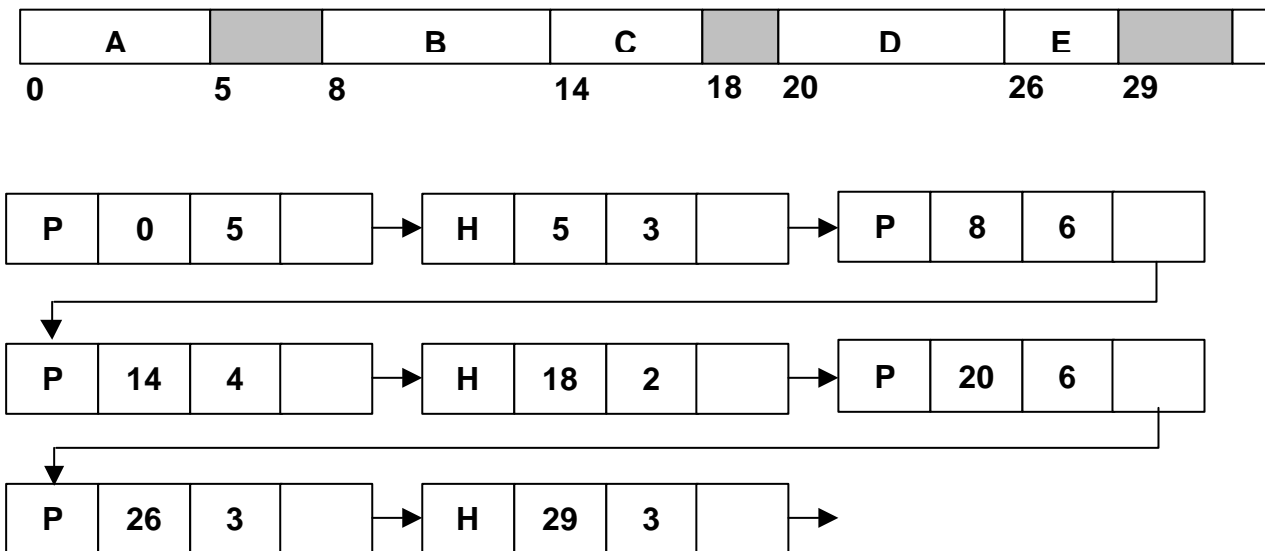
- Mediante mapa de bits:
 - ✓ Se divide la memoria en unidades de asignación.
 - ✓ El estado de cada asignación se representa con un bit (1=asignada, 0=no asignada)
 - ✓ Es importante la elección del tamaño de la unidad de asignación



```
11111000
11111111
11001111
11111000
...
```

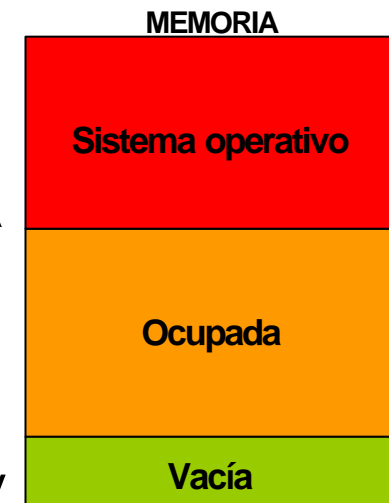

- Mediante lista enlazada:

- ✓ Se tiene una lista de segmentos ocupados (proceso) y libres (hueco)
- ✓ Se suelen tener ordenadas por direcciones.
- ✓ Se pueden tener listas separadas para procesos y huecos (ordenadas por tamaño)



Asignación contigua simple

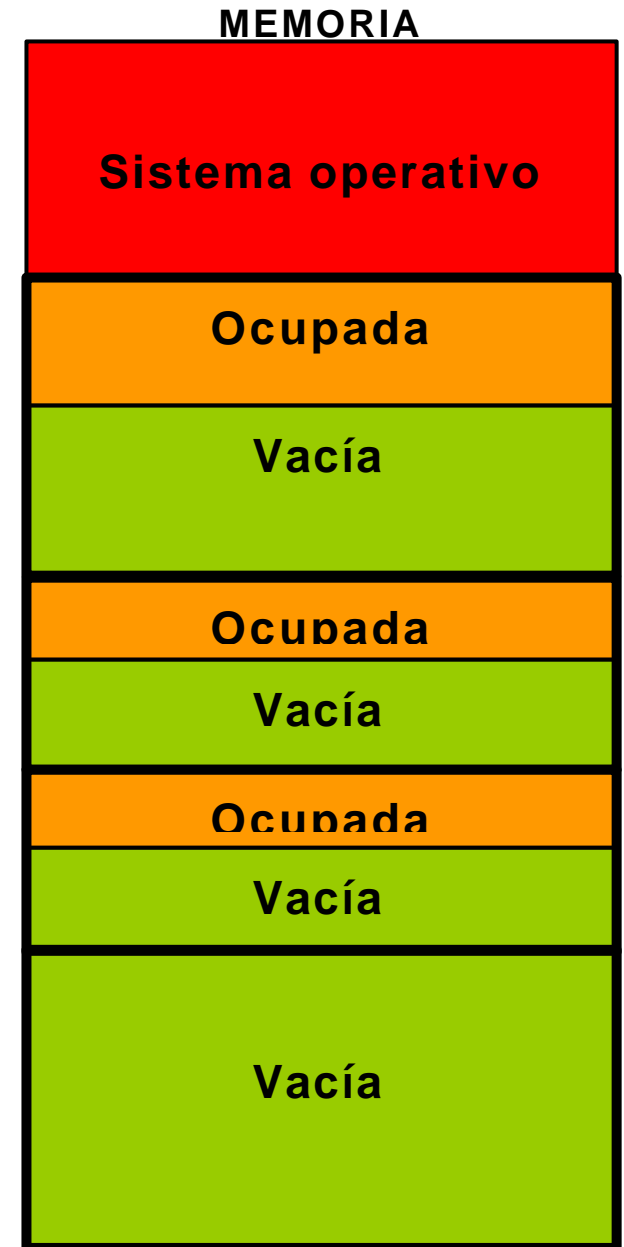
- Reserva una zona de memoria consecutiva al sistema operativo y el resto lo asigna al programa objeto de procesamiento.
- Utilizada en sistemas monousuario.
- Consideraciones:
 - ✓ es espacio de direcciones se encuentra todo en el mismo estado (asignado o no)
 - ✓ la asignación de memoria se produce durante la planificación del proceso
 - ✓ la liberación ocurre al terminar el procesamiento
 - ✓ la protección se establece mediante un registro límite y un indicador del modo de procesamiento (usuario o superusuario)



- Inconvenientes:
 - ✓ No se permite multiprogramación
 - ✓ Parte de la zona asignada puede estar vacía (**pobre utilización de la memoria**)
 - ✓ Limitación del tamaño máximo de programa que puede procesarse
 - ✓ Incapacidad para compartir recursos lógicos comunes
 - ✓ Todo el programa tiene que estar cargado en memoria para poderlo ejecutar

Método de particiones

- Se reserva una zona de memoria al sistema operativo y el resto se divide en particiones, cada una de las cuales se asigna a un programa.
- Es el mecanismo más simple para la gestión de la multiprogramación.
- El grado de multiprogramación (número máximo de programas en memoria) viene dado por el número máximo de particiones. ■
- Consideraciones:
 - ✓ Es necesario saber el estado, asignado o no, de cada partición y su tamaño.



- ✓ La asignación se produce, por particiones completas, durante la planificación.
- ✓ La liberación, por particiones completas, se realiza al terminar.
- ✓ La protección se consigue mediante registros límite. En cada acceso a memoria se obtendrá la dirección física y se comparará con los registros límite.

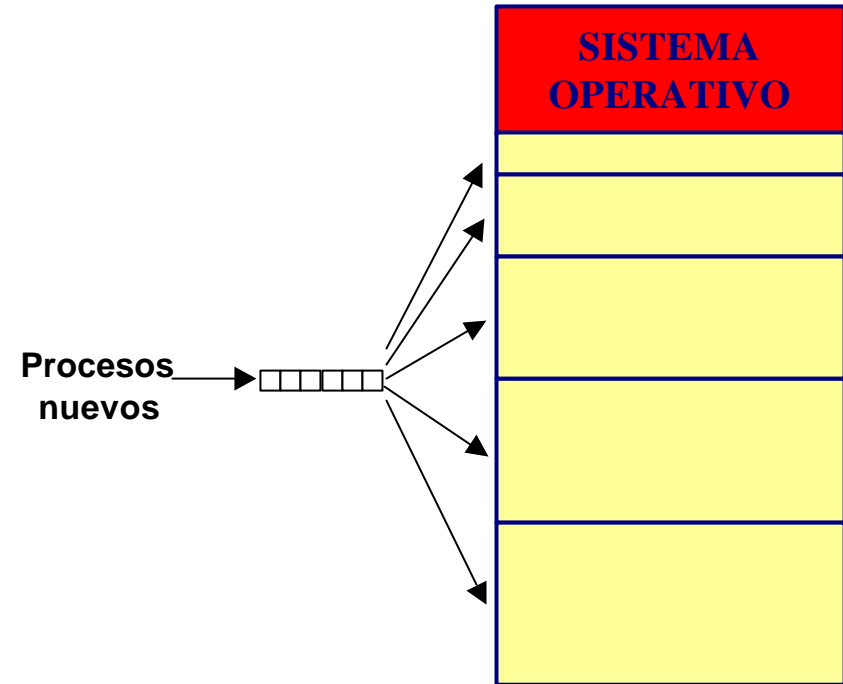
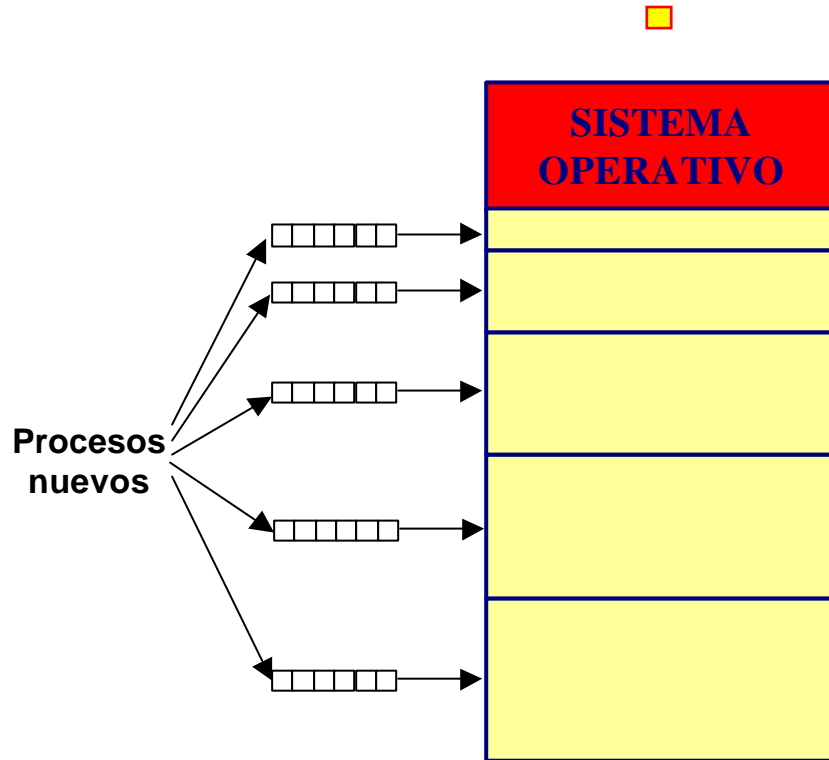
- **Tabla de estado de descripción de las particiones:**

- ✓ Estado de la partición (asignada o no)
- ✓ Identificador del programa al que está asignada
- ✓ Registro base de la partición
- ✓ Registro límite o tamaño de la partición



Particiones fijas

- Las particiones tienen un tamaño fijo.
- Estas pueden tener o no el mismo tamaño.
- El administrador del sistema elige los tamaños de particiones dependiendo del entorno donde se ubique el sistema.
- Cada partición contiene un proceso.
- El número de particiones limita el nivel de multiprogramación.
- Alternativas:
 - ✓ Cola única
 - ✓ Múltiples colas: se asigna cada proceso a la menor partición en la que quepa.

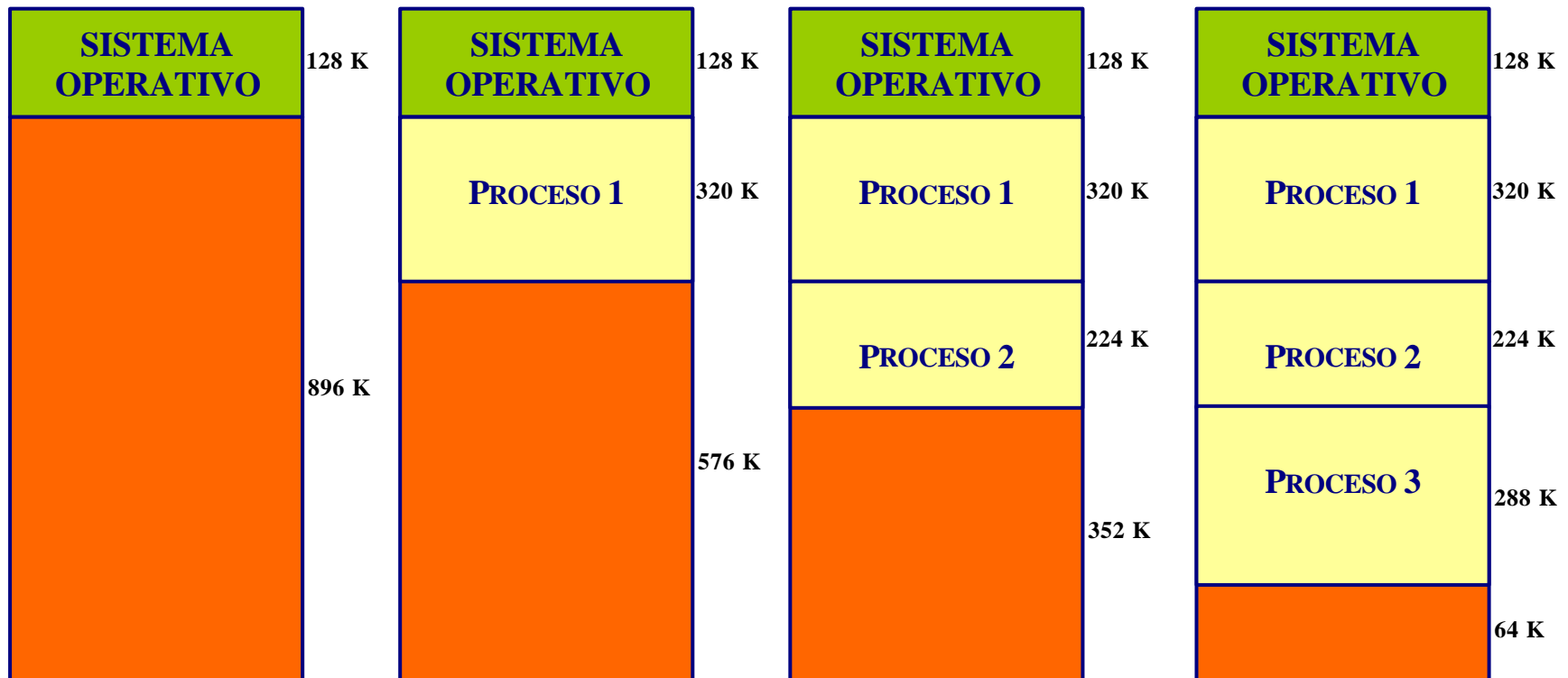


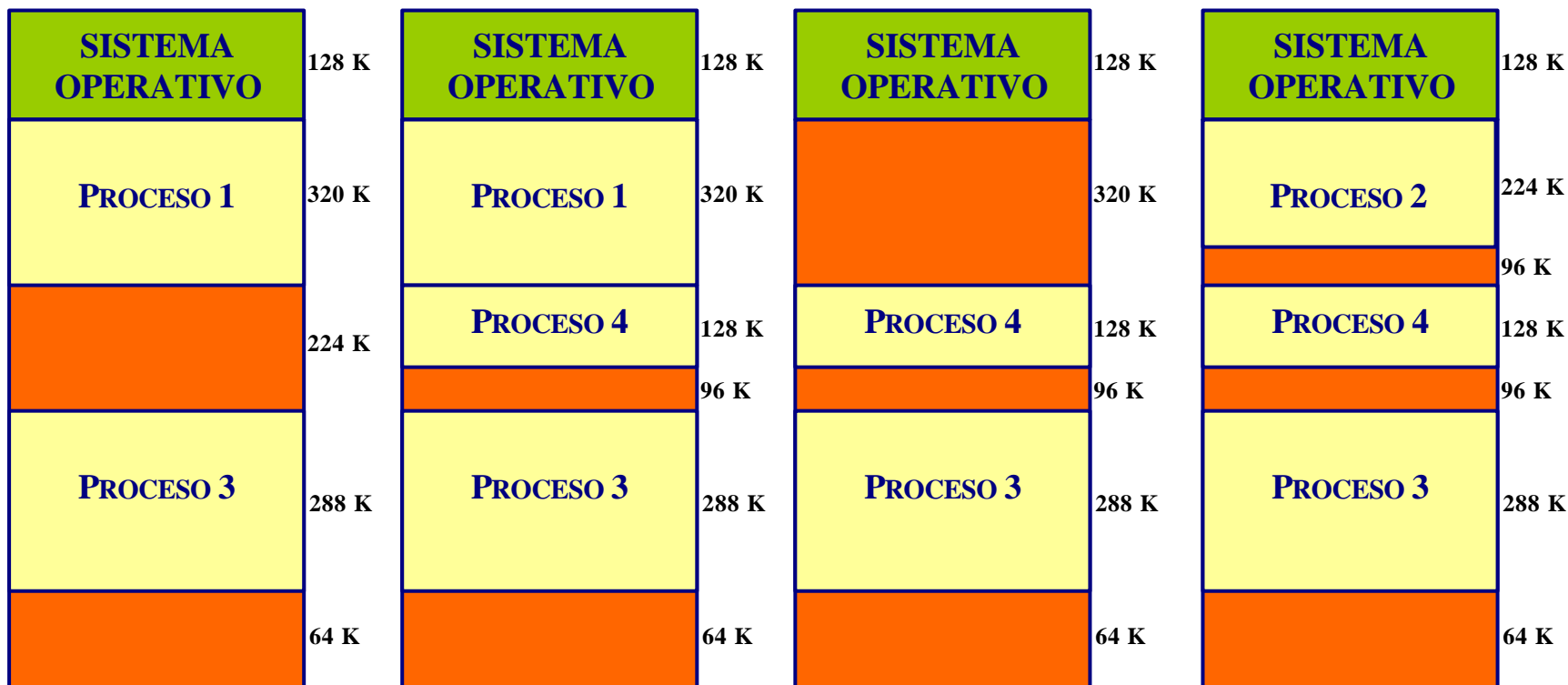
- Inconvenientes:

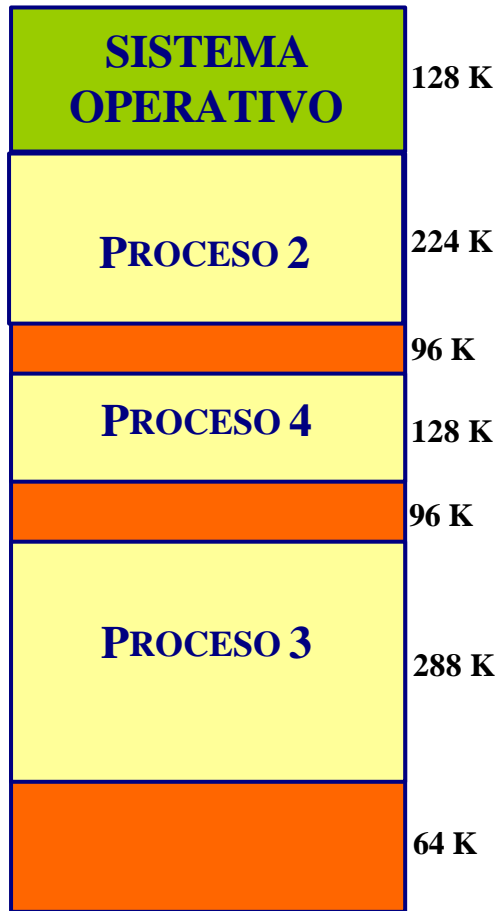
- ✓ fragmentación interna
- ✓ limitación del tamaño máximo de programa a la partición más grande
- ✓ bajo índice de ocupación de memoria

Particiones variables

- Los procesos son asignados a particiones que se ajustan a su tamaño.
- De este modo, no se produce fragmentación interna.
- Cuando se va a procesar un programa se le crea una partición y se libera cuando termine.







<i>Número</i>	<i>Proceso</i>	<i>Tamaño</i>	<i>Dirección</i>	<i>Estado</i>
1	2	224 K	128 K	asignada
2		96 K	352 K	disponible
3	4	128 K	448 K	asignada
4		96 K	576 K	disponible
5	3	288 K	672 K	asignada
6		64 K	960 K	disponible

Algoritmo general de asignación de partición

```
procedimiento asignación(demanda: entero)
{
    recorrer tabla hasta encontrar estado_p=disponible
    si existe
    {
        caso tamaño_p>demanda
            tamaño_p=tamaño_p-demanda
            crear nueva entrada en tabla
        caso tamaño_p=demanda
            estado_p=no_disponible
        caso tamaño_p<demanda
            repetir procedimiento
    }
    sino
        esperar
}
```

Algoritmo general de liberación de partición

```
procedimiento liberación(partición)
{
    caso partición adyacente a zona libre
        tamaño_z_l=tamaño_z_l+tamaño_p
        direccion_z_l=min(direccion_z_l,direccion_p)
        borrar entrada_p
    caso partición adyacente a zonas libres
        tamaño_z_l_1=tamaño_z_l_1+tamaño_p+tamaño_z_l_2
        borrar entrada_p
        borrar entrada_z_l_2
    caso partición no adyacente a zona libre
        estado_p=disponible
}
```

Algoritmos de asignación de particiones

- **Algoritmo de asignación del primer hueco:** se asigna la primera partición en la que quepa el proceso.
 - ✓ Tiende a concentrar grandes zonas libres al final de la memoria
 - ✓ Rápido
- **Algoritmo de asignación del siguiente hueco:** se asigna la primera partición en la que quepa el proceso, a partir de la posición de memoria de la última partición asignada.
 - ✓ Tiene las ventajas del primer hueco, distribuyendo, además, los procesos por toda la memoria
 - ✓ Rápido

- **Algoritmo de asignación del mejor hueco:** se asigna el hueco más pequeño que tenga el tamaño suficiente.
 - ✓ Tiende a preservar las zonas libres más grandes
 - ✓ Puede fragmentar la memoria en múltiples zonas muy pequeñas
 - ✓ Lento
- **Algoritmo de asignación del peor hueco:** se asigna el proceso al hueco más grande.
 - ✓ Tiende a homogeneizar el tamaño de las zonas libres
 - ✓ Se adapta mal para atender programas grandes
 - ✓ Proporciona buenos resultados para gestionar sistemas donde los programas son sensiblemente parecidos
 - ✓ Lento

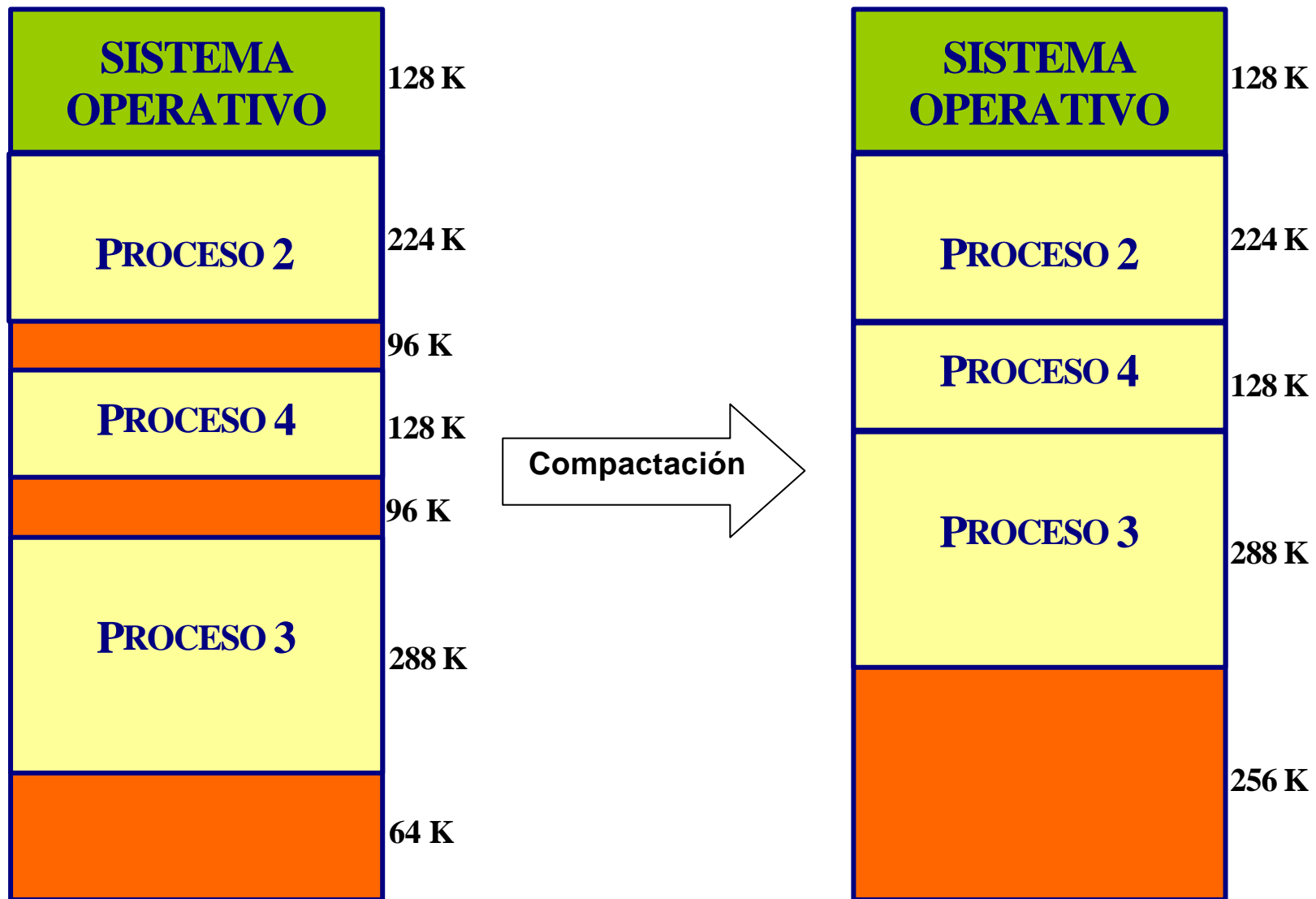
- Estructuras de datos:
 - ✓ Lista enlazada de huecos ordenada por direcciones:
 - Adecuada para los algoritmos de primer y siguiente hueco
 - Facilita la fusión de huecos
 - ✓ Lista enlazada de huecos ordenada por tamaño:
 - Adecuada para los algoritmos de mejor y peor hueco
 - Dificulta la fusión de huecos
- Inconvenientes:
 - ✓ Gestión de particiones compleja
 - ✓ Fragmentación externa
 - ✓ Limitación del tamaño de los programas
 - ✓ Incapacidad para compartir recursos lógicos comunes

Ejercicio 1.

PARTICIONES FIJAS

Compactación

- Técnica para mejorar la utilización de memoria en la estrategia de particiones dinámicas.
- Desplazar las particiones asignadas para colocar toda la memoria libre en una única partición de gran tamaño.
- Se resuelve el problema de la fragmentación externa.
- Los algoritmos de compactación se caracterizan, básicamente, por el instante de efectuarla:
 - ✓ compactar tras cada liberación de partición
 - ✓ compactar cuando se solicita una partición mayor que cualquier zona libre y menor que la suma de éstas
- El tiempo de compactación puede llegar a ser alto.



Ejercicio 2.
COMPACTACIÓN

Intercambio (swapping)


- Técnica para aumentar la multiprogramación en la gestión de memoria por asignación contigua simple o por particiones.
- Se realiza cuando se decide admitir un nuevo proceso para el cual no se encuentra una partición libre adecuada.
- Se selecciona alguno de los procesos suspendidos que ocupan particiones en las que cabe el nuevo proceso.
- Consiste en transferir bloques de información entre memoria central y memoria auxiliar de modo que los programas se van ejecutando a base de intervalos de tiempo.
- La memoria de swap debe ser rápida.

- Para reducir el tiempo de transferencia entre memoria central y auxiliar:
 - ✓ transferir sólo aquellas direcciones conteniendo información
 - ✓ mejorar las prestaciones de los dispositivos sobre lo que se efectúa el swap

Técnicas de asignación dispersa

- El espacio de direcciones virtuales no ha de estar, necesariamente, mapeado en memoria de forma contigua.
 - ✓ Paginación
 - ✓ Segmentación
 - ✓ Segmentación paginada

Paginación

- Las técnicas de asignación contigua usan de forma ineficiente la memoria, debido a la fragmentación externa o interna.
- La paginación es una estrategia de asignación no contigua de la memoria que consiste en dividir los espacios de direcciones lógicas y físicas en zonas de igual tamaño.
- La memoria lógica está dividida en bloques de tamaño fijo denominados **páginas**. 
- La memoria física se encuentra dividida en bloques, de igual tamaño que las páginas, denominados **marcos**.

Traducción de direcciones



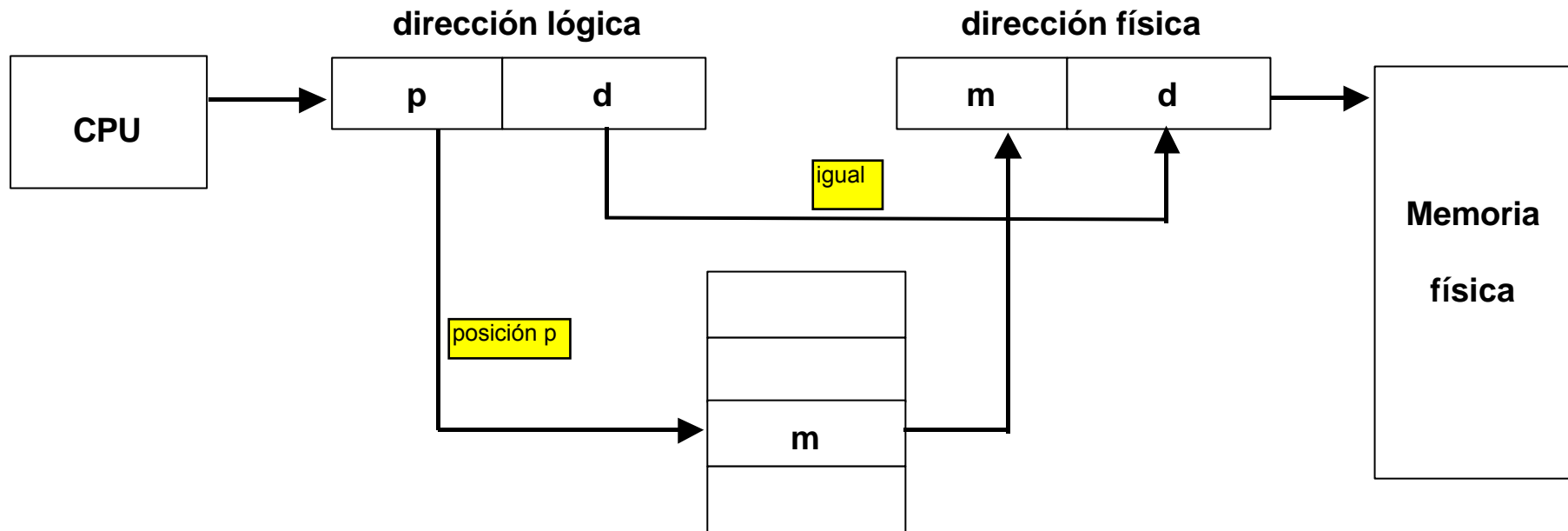
- Las direcciones lógicas tienen dos campos: página y desplazamiento



- Las direcciones físicas tienen dos campos: marco y desplazamiento.

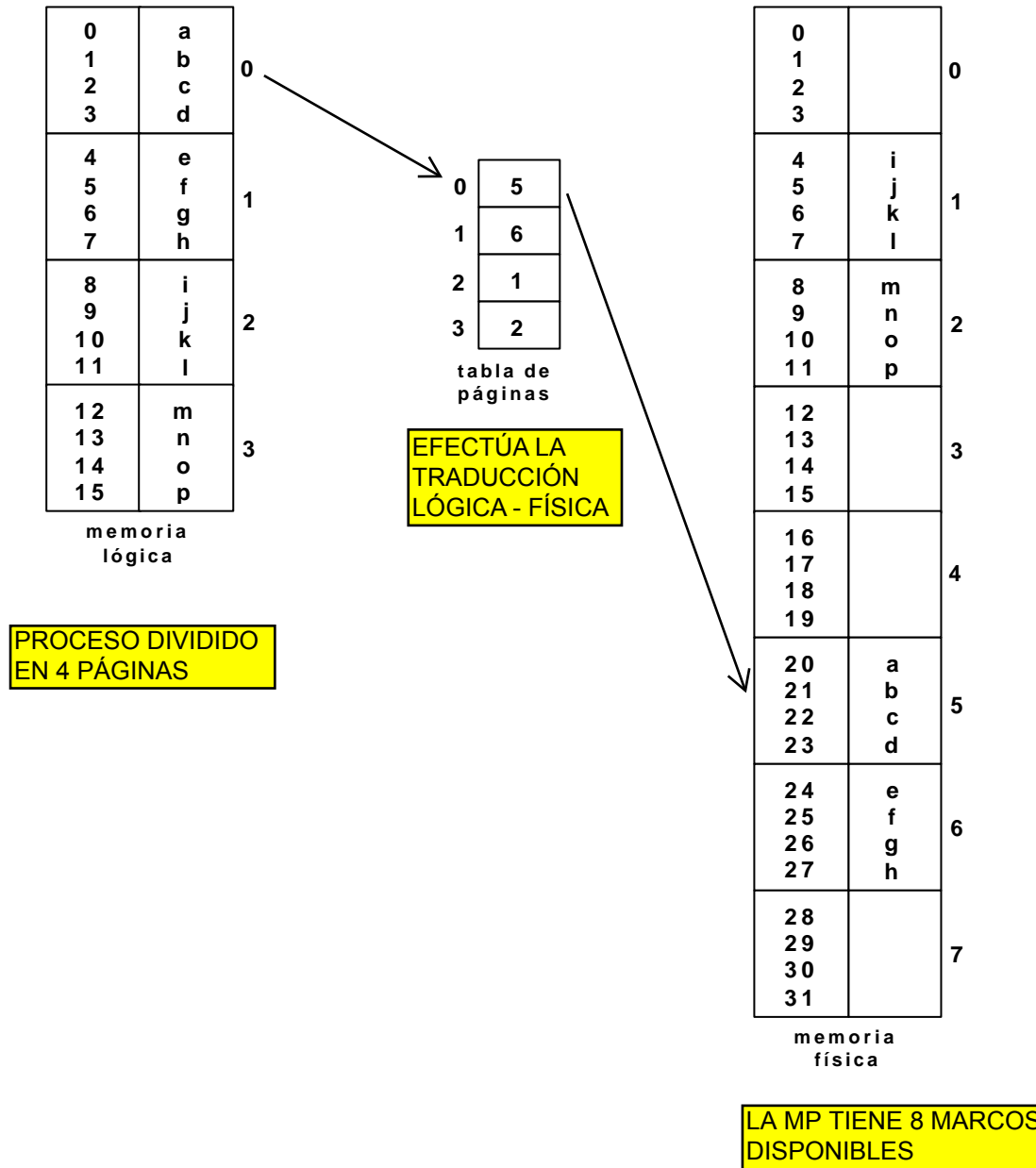


- Cada proceso tiene asignada una tabla de páginas (en su Bloque de Control de Proceso) que establece la correspondencia entre las páginas y los marcos donde están cargadas, además de información adicional.



- El tamaño de página suele estar entre 512 bytes y 8K. RELATIVAMENTE PEQUEÑO
- La selección de una potencia de 2 como tamaño hace más fácil la traslación de una dirección lógica a un número de página y un desplazamiento.
- Si el tamaño de página es 2^k , los k bits menos significativos representan un desplazamiento y los $(m-k)$ restantes el número de página.

EJEMPLO





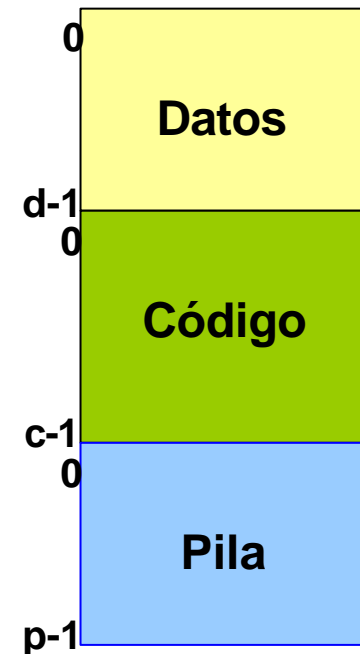
- Se puede producir **fragmentación interna**.
 - ✓ debida a espacio no utilizado en el último marco asignado a cada proceso.
 - ✓ Los tamaños de página grandes aumentan la fragmentación interna.
 - ✓ Los tamaños de pagina pequeños requieren tablas de páginas grandes.
- La reubicación se facilita, ya que el espacio de direcciones lógicas comienza en la dirección 0.
- Varios procesos pueden compartir un marco de página referenciándolo desde diferentes tablas, con lo que se duplica página pero no marco.
- Las páginas pueden estar protegidas de diferentes modos: lectura, escritura o lectura/escritura.



Ejercicio 3.
PAGINACIÓN

Segmentación

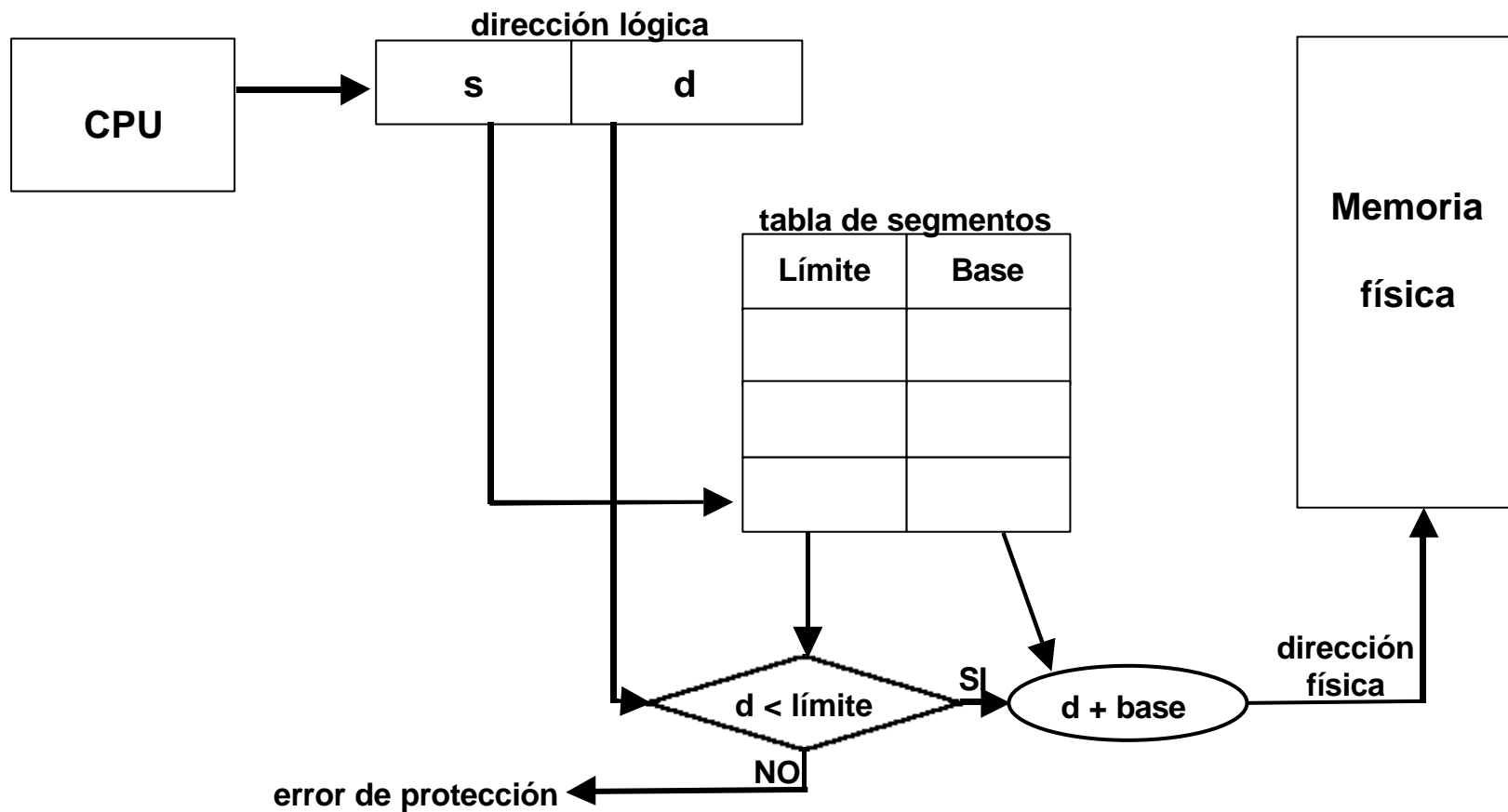
- Se divide el espacio lógico de direcciones del proceso en agrupaciones lógicas de información denominados **segmentos**.
- Los segmentos son entidades lógicas relacionadas por el programador:
 - ✓ Conjuntos de subrutinas
 - ✓ Conjuntos de variables o datos
 - ✓ La pila
- Cada segmento consiste en un espacio lineal de direcciones, desde 0 hasta algún máximo.



- Toda la información de un segmento se ubica en un área contigua de memoria.
- Distintos segmentos pueden ir ubicados en zonas no contiguas de la memoria física (se pueden ocupar diferentes particiones no contiguas).
- La segmentación es similar al método de particiones dinámicas, salvo en que el espacio de direcciones está dividido en bloques (“segmentos”).
 - ✓ No se produce fragmentación interna.
 - ✓ Se produce fragmentación externa (menor que con particiones variables)
- Las direcciones lógicas tienen dos campos: segmento y desplazamiento.



- Cada programa posee una **tabla de segmentos** que relaciona las direcciones lógicas y físicas.



- Ventajas:
 - ✓ Elimina la fragmentación interna (si es preciso, se compacta)
 - ✓ Los segmentos pueden crecer de forma independiente.
 - ✓ Facilidad para compartición de memoria.
 - ✓ Protección a nivel de unidades lógicas.
 - ✓ El programador debe desarrollar de forma modular los programas.
- Desventajas:
 - ✓ Dificultades de gestión de la memoria debido a que los segmentos tienen tamaño variable.
 - ✓ Todo el programa tiene que estar en memoria para poderlo ejecutar.
 - ✓ Sobrecarga debida al uso de tablas y a la compactación.

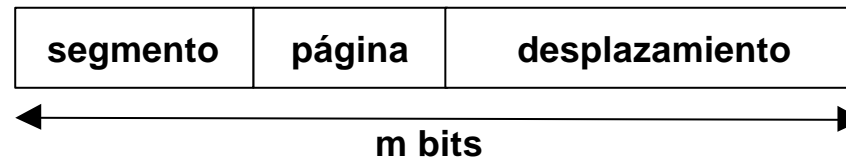


Paginación frente a segmentación

Paginación	Segmentación
La división de los programas en páginas es transparente al programador	El programador especifica al compilador los segmentos del programa
Las páginas tienen tamaño fijo	Los segmentos tienen tamaño variable
Dirección virtual = $n^{\circ} \text{ de página} + \text{desplamiento}$	Dirección virtual = $n^{\circ} \text{ de segmento} + \text{desplamiento}$
No hay fragmentación externa	Hay fragmentación externa
Hay fragmentación interna	No hay fragmentación interna
Se puede compartir y proteger a nivel de página	Se puede compartir y proteger a nivel de segmentos

Segmentación paginada

- Cuando los segmentos son excesivamente grandes:
 - ✓ Aumenta la fragmentación externa
 - ✓ Aumenta el problema de encontrar un hueco libre en memoria
- Solución: paginar los segmentos.
- A cada segmento se le asocia su propia tabla de páginas.
- Las direcciones lógicas están formadas por un número de segmento, un número de página y el desplazamiento.



Ejercicio 4.

SEGMENTACIÓN Y PAGINACIÓN

- Es necesaria la consulta de dos tablas para obtener la dirección física, lo cual puede repercutir en el rendimiento del sistema.
- Aumenta el espacio consumido por las tablas.
- Se reduce la fragmentación interna a la última página del segmento.

Memoria virtual

- Técnica que permite la ejecución de un proceso aunque no esté cargado totalmente en memoria. El resto del proceso se encontrará en memoria secundaria.
- Ventajas:
 - ✓ Un programa ya no está restringido por la cantidad de memoria física disponible.
 - ✓ Se aumenta la utilización de la CPU, al poder ejecutar más programas al mismo tiempo.
 - ✓ Se requiere menos E/S para cargar o intercambiar los procesos de usuario, por lo que se ejecutarán más rápido.
- El mecanismo más elemental de memoria virtual es el de **recubrimientos**.

- Se encuentran dos implementaciones de memoria virtual:
 - ✓ Paginación por demanda
 - ✓ Segmentación por demanda
- Se mantienen en memoria principal aquellas páginas o segmentos que sean necesarios en cada momento.
- El conjunto de páginas o segmentos que se mantienen para cada proceso constituyen su **conjunto residente**.

Paginación por demanda

- Se combinan las técnicas de:
 - ✓ paginación
 - ✓ intercambios (swapping)
- Un intercambiador manipula procesos enteros entre memoria principal y secundaria.
- Un paginador hace intercambios de páginas entre memoria principal y secundaria según haga falta.

- Cada entrada de la tabla de páginas posee un **bit de validez** que indica si la página está cargada actualmente en memoria o no, y un **bit de modificación** que indica si la página se ha modificado o no en memoria (**página sucia y página limpia**).

bit de validez	bit de modificación	marco
----------------	---------------------	-------

- La conversión de direcciones virtuales (página y desplazamiento) a direcciones físicas se realiza de una forma parecida a la paginación simple.
- Cuando se accede a una página que no está en memoria central, se genera una **interrupción por falta de página**, cuyo objeto es cargarla en memoria.
- Se produce un **reemplazo de páginas** si la memoria está totalmente ocupada y se produce un **fallo de página**.

- Algoritmo general:

```
inicio procesamiento instrucción
si bit_de_validez = 1
    acceder a memoria
sino
    si no existe bloque libre
        seleccionar página a sustituir
        actualizar tablas
        si página modificada (bit_de_modificación = 1)
            escribir página en memoria auxiliar
        fin si
    fin si
    transferir nueva página a memoria principal
    actualizar tablas
fin si
```

Rendimiento de la paginación por demanda

- La **tasa de fallos de página** es la probabilidad de que se produzca un fallo de página:

$$p = n^{\circ} \text{ de fallos} / n^{\circ} \text{ total de referencias}$$

$$0 \leq p \leq 1$$

- Si $p=0$ no se produce ningún fallo.
- Si $p=1$ cada referencia es un fallo.
- El tiempo de acceso efectivo es:

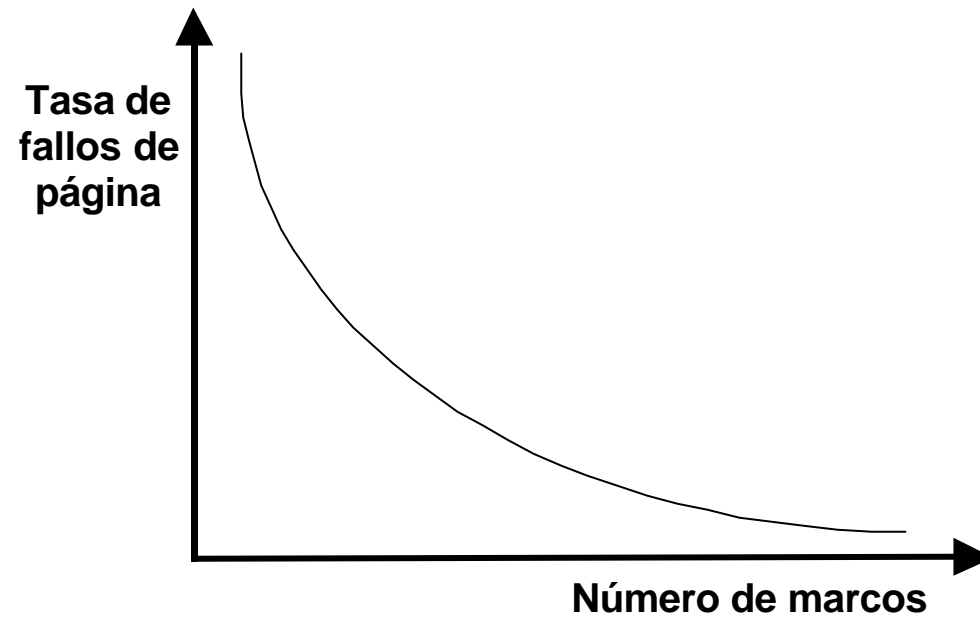
$$\text{TAE} = (1-p) * \text{TAM} + [p * (1+p_m) * \text{TTP}]$$

✓ TAM: tiempo de acceso a memoria

✓ TTP: tiempo de transferencia página/disco

✓ p_m : probabilidad de que la página a reemplazar haya sido modificada

- La tasa de fallos de página disminuye cuando aumenta el número de marcos.

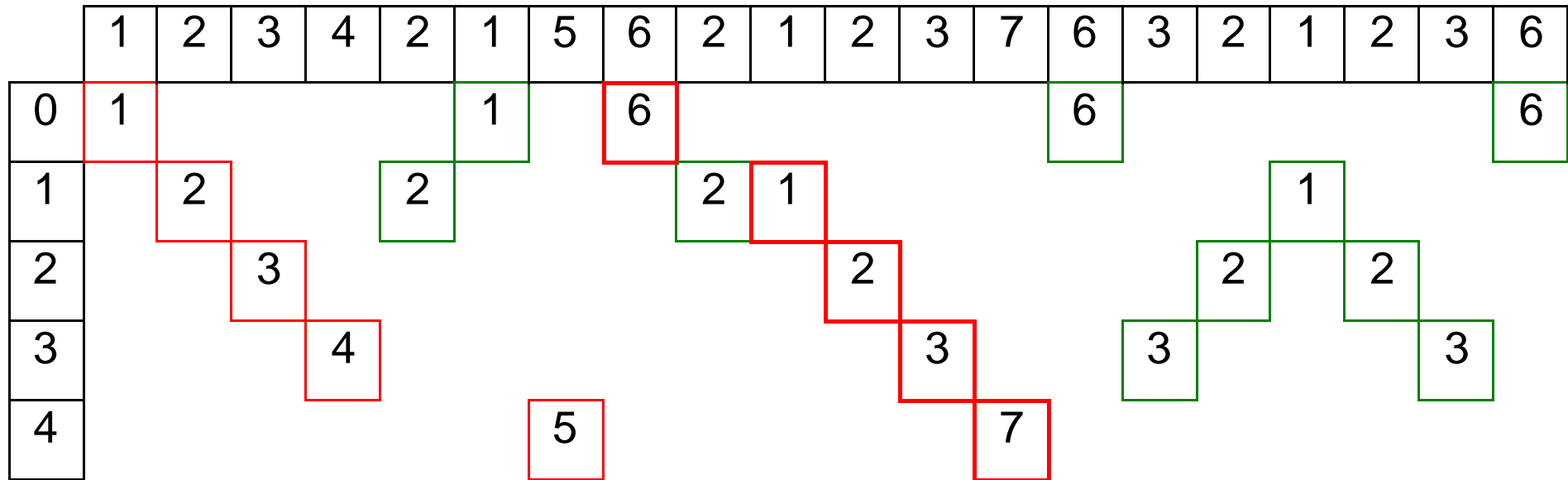


Algoritmos de reemplazo de páginas

- Algoritmo FIFO.
- Algoritmo óptimo.
- Algoritmo LRU.
- Algoritmo de segunda oportunidad.
- ~~Algoritmo LFU.~~
- ~~Algoritmo NUR.~~

Algoritmo FIFO

- Se elimina la página que hace más tiempo que fue cargada en memoria.



Se producen 10 fallos de página (5 reemplazamientos)

- Una página que se utilice mucho y sea antigua se sustituirá.
- Se puede producir la anomalía FIFO o **anomalía de Belady**: se producen más fallos de páginas al aumentar el número de marcos.

	1	2	3	4	1	2	5	1	2	3	4	5
0	1			4			5					5
1		2			1			1		3		
2			3			2			2		4	

Se producen 9 fallos de página (5 reemplazamientos)

	1	2	3	4	1	2	5	1	2	3	4	5
0	1				1		5				4	
1		2				2		1				5
2			3						2			
3				4						3		

Se producen 10 fallos de página (6 reemplazamientos)

Algoritmo óptimo

- Se reemplaza la página que tardará más tiempo en ser referenciada.

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1					1				1							1			
1		2			2				2		2					2		2		
2			3							3					3				3	
3				4				6						6						6
4						5					7									

Se producen 7 fallos de página (2 reemplazamientos)

- Algoritmo no realizable. Tiene importancia como comparativa.
- No presenta la anomalía de Belady.

Algoritmo LRU (Least Recently Used)

- Se elimina la página que hace más tiempo que no se ha utilizado.
- Aproximación al algoritmo óptimo que utiliza el pasado reciente como aproximación de lo que sucederá en el futuro cercano.

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1					1				1							1			
1		2			2				2		2					2		2		
2			3					6						6						6
3				4							3				3				3	
4						5						7								

Se producen 8 fallos de página (3 reemplazamientos)

Algoritmo de segunda oportunidad

- Se asocia a cada página un bit de referencia.
- Se elige una página con criterio FIFO y se inspecciona el bit de referencia:
 - ✓ Si el bit está a 1, se le da una segunda oportunidad:
 - poner el bit de referencia a 0
 - dejar la página en memoria
 - ✓ Si el bit está a 0, esa página será la víctima.
- Se elige la siguiente página con criterio FIFO y se aplican las mismas reglas.

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1					1		6						6						6
1		2			2				2		2					2		2		
2			3							1							1			
3				4							3			3					3	
4						5						3	7							

Se producen 9 fallos de página (4 reemplazamientos)

Ejercicio 5

ejerciciosmemoriavirtual.pdf

Asignación de páginas

- **Asignación equitativa:** si hay m marcos y n procesos, se le asignan m/n marcos a cada proceso.
- **Asignación proporcional:** se asigna memoria a cada proceso según su tamaño.

$$V = \sum v_i$$

$$m_i = \frac{v_i}{V} \times m$$

- ✓ v_i : tamaño del proceso i
- ✓ m : n° total de marcos
- ✓ m_i : marcos asignados al proceso i

Formas de reemplazo de páginas

- **Reemplazo global:**

- ✓ Un proceso elige para reemplazar marcos del conjunto de marcos total.
- ✓ Puede elegir marcos de otros procesos.
- ✓ El número de marcos asignados a un proceso es variable. Crece o decrece según lo necesite.

- **Reemplazo local:**

- ✓ Un proceso sólo selecciona páginas de su conjunto de marcos asignados.
- ✓ El número de páginas asignadas al proceso es fijo.
- ✓ Se puede producir **hiperpaginación**.

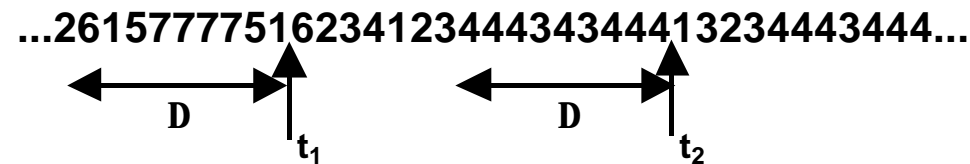
Hiperpaginación (thrashing)

- Alto índice de agitación o actividad en el trasiego de información entre la memoria central y la memoria auxiliar.
- Se produce cuando un proceso usa activamente un número determinado de páginas y tiene asignados un número menor de marcos. Por tanto, se producen **fallos de página frecuentes**.
- La hiperpaginación reduce el rendimiento del sistema.
- Si se emplea **estrategia de asignación global de marcos**:
 - ✓ No existe restricción en el número de marcos asignados a los procesos
 - ✓ Un programa puede tomar muchos marcos de otros procesos
 - ✓ Estos procesos producirán muchos fallos de página
 - ✓ Solución: reducir el grado de multiprogramación

- Si se emplea **estrategia de asignación local de bloques**:
 - ✓ Los procesos tienen asignados un número máximo de marcos
 - ✓ Un proceso puede necesitar más marcos de los asignados provocando fallos de página frecuentes
 - ✓ Puede llegar a ralentizar el paginador
 - ✓ Otros procesos que no sufren de hiperpaginación tardarán más al intentar cargar una página
 - ✓ Solución: asignar a cada proceso los marcos que necesite
- No se sabe a priori cuántos marcos necesitará un proceso, por lo que se intenta modelizar la utilización de la memoria.

- **Controlar la frecuencia de fallos de página:**
 - ✓ Se establecen límites superior e inferior a la proporción de fallos
 - ✓ Si se excede el límite superior, se asignan más marcos al proceso
 - ✓ Si se cae por debajo del límite inferior, se libera algún marco
 - ✓ Un proceso puede llegar a suspenderse si se queda con muy pocos marcos, por haberlos cedido a otros procesos
- **Modelo del working set** (conjunto de trabajo o conjunto residente):
 - ✓ Se basa en el modelo de la localidad: a medida que progresa la ejecución de un programa, las referencias a memoria evolucionan de una localidad a otra
 - ✓ Una localidad es un conjunto de páginas que se utilizan juntas activamente

- ✓ El **working set** es el conjunto de páginas de las n últimas referencias (ventana del WS) a memoria.



$$WS(t_1) = \{1, 2, 5, 6, 7\}$$

$$WS(t_2) = \{3, 4\}$$

- ✓ Si TWS_i es el tamaño de cada working set, el número total de demandas es $D = \sum TWS_i$
- ✓ Si M es el número de marcos de memoria física, si $D > M$ entonces se produce hiperpaginación

- ✓ Se asigna a cada proceso los marcos necesarios para tener su working set en memoria
- ✓ Si quedan marcos suficientes, se asigna memoria a otro proceso más

Prepaginación

- Cuando se inicia o se despierta un proceso se producen un gran número de fallos de página
- La prepaginación es un intento de reducir este elevado nivel inicial de paginación.
- Consiste en cargar en memoria al mismo tiempo todas las páginas que se necesitarán.
- Cuando el proceso está en cola de espera a ser ejecutado, se pueden cargar las páginas que había antes de su suspensión.

Segmentación por demanda

- En memoria central se encuentran los segmentos del programa en uso. El resto se encuentran en memoria secundaria.
- La tabla de segmentos está formada por los registros base y límite, un bit de validez, un bit de acceso (referencia), un bit de modificación y varios bits de control.

bit de validez	bit de acceso	bit de modificación	límite	base
----------------	---------------	---------------------	--------	------

- Si se accede a un dirección cuyo segmento está fuera de memoria se produce una interrupción por **fallo de segmento**.
- La asignación se produce sobre zonas libres de tamaño suficiente:
 - ✓ si no hubiera espacio libre, se compacta
 - ✓ Si sigue sin haber, se utilizan técnicas similares a paginación (FIFO, LRU,...)