GESTION DE MEMORIA

SISTEMAS OPERATIVOS

Universidad Autónoma de Entre Ríos
FCyT FACULTAD DE CIENCIA Y TECNOLOGÍA

Gestión de memoria

- Sistemas monoprocesos
  - *una parte para el sistema operativo (monitor residente, núcleo) y una parte para el programa actualmente en ejecución*
- Sistemas multiprocesos
  - *la parte de «usuario» de la memoria se debe subdividir posteriormente para acomodar múltiples procesos.*

Requisitos para el manejo de la memoria

- **Relocalización**
  - el programador no sabe en que parte de la memoria se cargará el programa cuando se ejecute
  - mientras se ejecuta el programa, puede salir a disco y regresar a memoria principal en una posición diferente
  - las referencias a memoria deben traducirse en el código a la dirección de memoria física

Requisitos para el manejo de la memoria

- **Protección**
  - los procesos no deben poder referenciar posiciones de memoria en otro proceso sin permiso
  - es imposible checar direcciones en programas ya que el programa podría moverse en memoria
  - deben checarse durante la ejecución

Requisitos para el manejo de la memoria

- **Compartición**
  - permitir que varios procesos accesen la misma porción de memoria
  - mejor permitir a cada proceso (persona) accesar la misma copia del programa en vez de que tengan su propia copia separada

Requisitos para el manejo de la memoria

- **Organización lógica**
  - los programas están escritos en módulos
  - diferentes grados de protección dados a diferentes módulos (solo lectura, solo ejecución)
  - compartir módulos

Requisitos para el manejo de la memoria

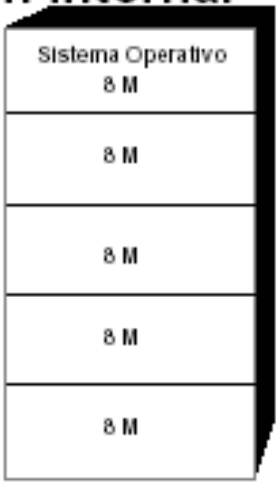
- **Organización física**
  - la memoria disponible para un programa mas sus datos puede ser insuficiente
    - overlaying permite que varios módulos sean asignados a la misma región de memoria
  - la memoria secundaria es más barata, de mayor capacidad, y permanente

Particiones fijas

- **Particiona la memoria disponible en regiones con límites fijos**
- **Particiones de tamaño igual**
  - cualquier proceso cuyo tamaño sea menor o igual al de la partición puede cargarse en una partición disponible
  - si todas las particiones están llenas, el SO puede sacar un proceso fuera de una partición
  - un programa puede no caber en una partición. El programador debe diseñar el programa con overlays

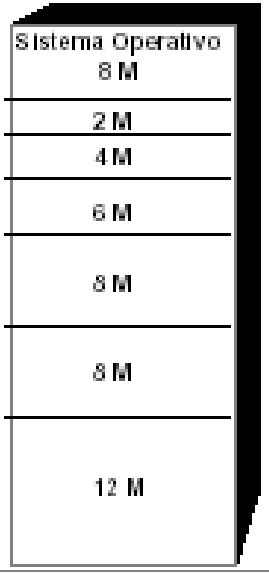
Particiones fijas

- El uso de la memoria principal es ineficiente. Cualquier programa, no importa que tan pequeño sea, ocupa una partición completa. A esto se le llama fragmentación interna.



Particiones fijas

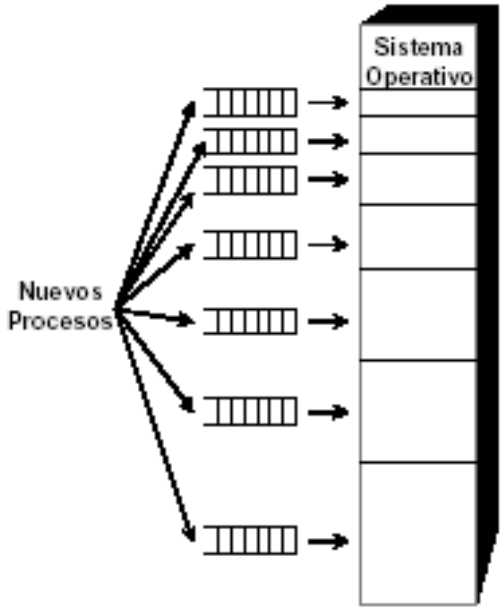
- **Particiones de tamaño desigual**
  - reduce el problema de las particiones de tamaño igual



Algoritmos de ubicación con particiones

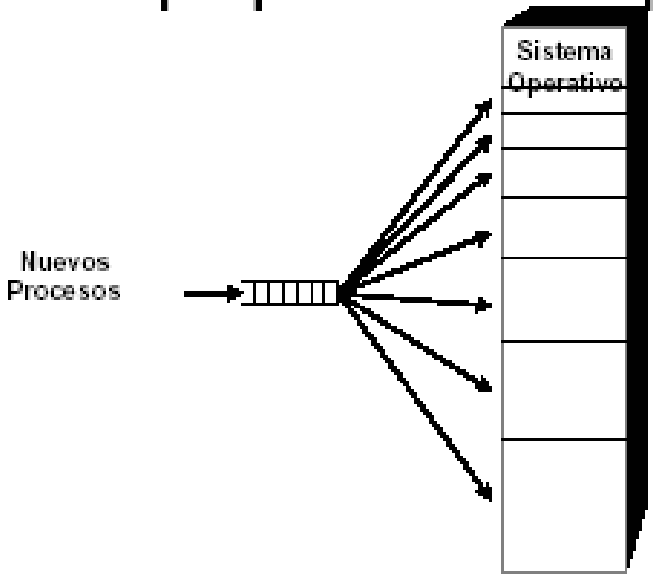
- **Particiones de tamaño igual**
  - ya que todas las particiones son del mismo tamaño, no importa que partición se use
- **Particiones de tamaño desigual**
  - puede asignarse cada proceso a la partición más pequeña en la que pueda caber
  - una cola por cada partición
  - los procesos son asignados de tal forma como para minimizar memoria desperdiciada dentro de una partición

Una cola de procesos para cada partición



Una cola de procesos para todas las particiones

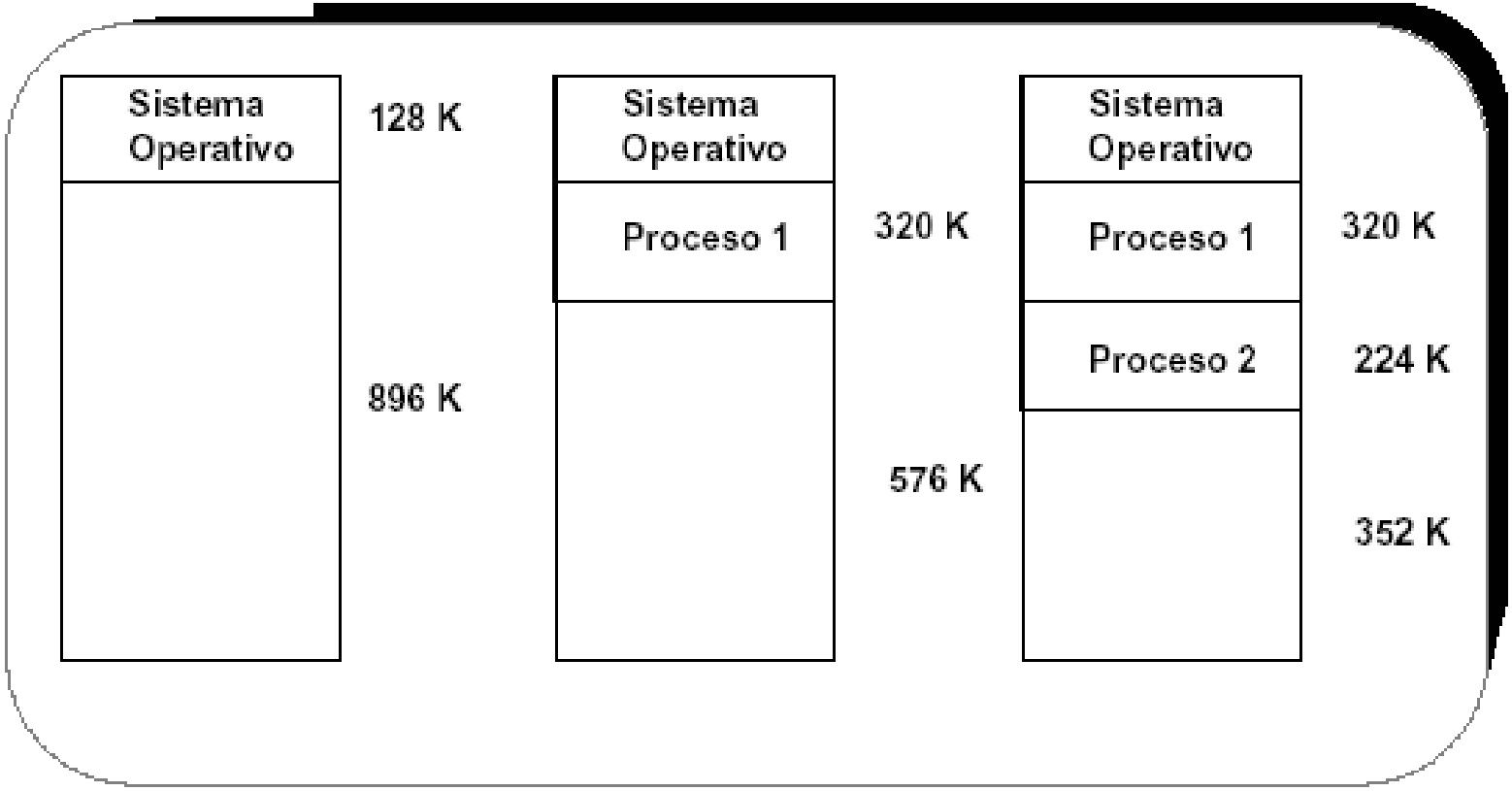
- Cuando es hora de cargar un proceso a memoria principal se selecciona la partición disponible más pequeña que pueda tener el proceso



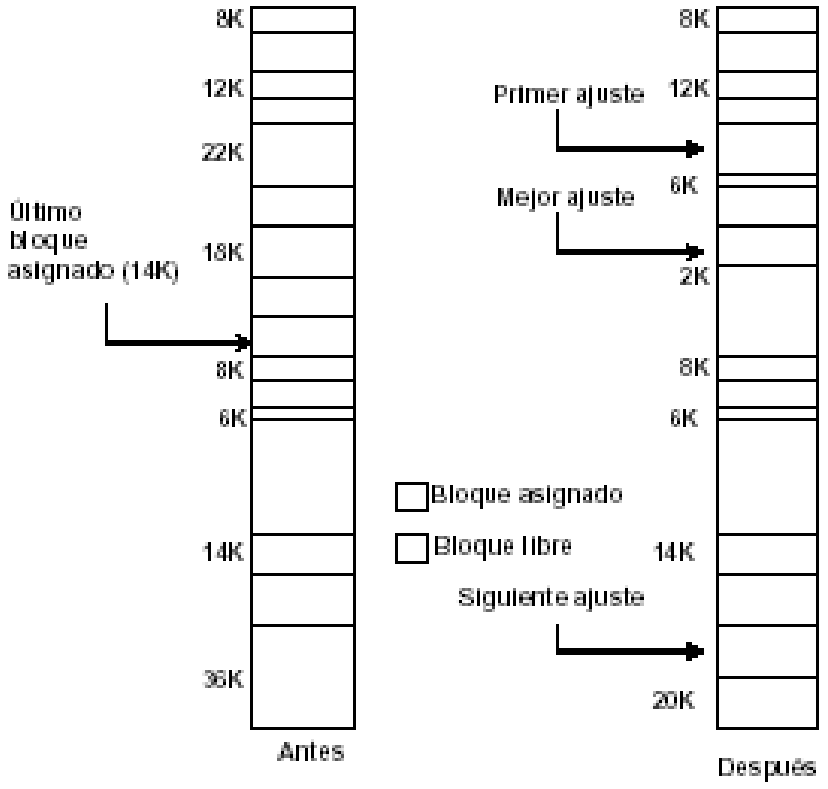
Particiones dinámicas

- Las particiones son variables en tamaño y número
- Un proceso se le asigna exactamente la memoria que requiere
- Eventualmente se hacen huecos en la memoria. A esto se le llama fragmentación externa
- Se debe usar compactación para desplazar procesos tal que queden contiguos y toda la memoria libre en un bloque

Ejemplo de particiones dinámicas



Partición dinámica, Algoritmos de ubicación



Partición dinámica, Algoritmos de ubicación

El SO debe decidir que bloque libre asignar a un proceso

El mejor ajuste

- escoje el bloque que está lo más cerca en tamaño al solicitado
- en conjunto el peor rendimiento
- ya que se encuentra el bloque más pequeño por proceso, queda la menor cantidad de fragmentación, se debe compactar más seguido

Algoritmo del primer ajuste

- comienza recorriendo la memoria desde el inicio y selecciona el primer bloque disponible que sea lo suficientemente grande
- el más rápido
- puede tener muchos procesos cargados en el principio de la memoria que deben buscarse cuando se intente encontrar un bloque libre

Siguiente ajuste

- inicia recorriendo la memoria desde la posición de la última asignación y selecciona el siguiente bloque disponible que sea lo suficientemente grande
- es más común que se asigne un bloque al final de la memoria donde se encuentra el bloque más grande
- el bloque más grande de memoria se parte en bloques más pequeños
- se requiere compactación para obtener un bloque grande al final de la memoria

Algoritmos de reemplazo

- **Cuando todos los procesos en memoria principal están bloqueados, el SO debe escoger que proceso debe reemplazar**
  - Un proceso debe ser expulsado (A un estado de bloqueado-suspendido) y será remplazado por un nuevo proceso o un proceso de la cola Listo-suspendido
  - ver memoria virtual

- Un sistema monotarea con gestión de memoria particionada variable mantiene, en un instante dado, un número N de fragmentos en memoria real. Existen huecos disponibles no contiguos de tamaño 100K, 500K, 200K, 300K y 600K en orden de posiciones de memoria. Se produce el arribo de 4 nuevas solicitudes con demandas de memoria de 212K, 417K, 112K y 426K respectivamente. a) Determine cómo ubicará los fragmentos en memoria cada uno de los algoritmos de asignación de memoria de primer ajuste, de mejor ajuste y de peor ajuste. b) ¿Cuál algoritmo hace uso más eficiente de la memoria?

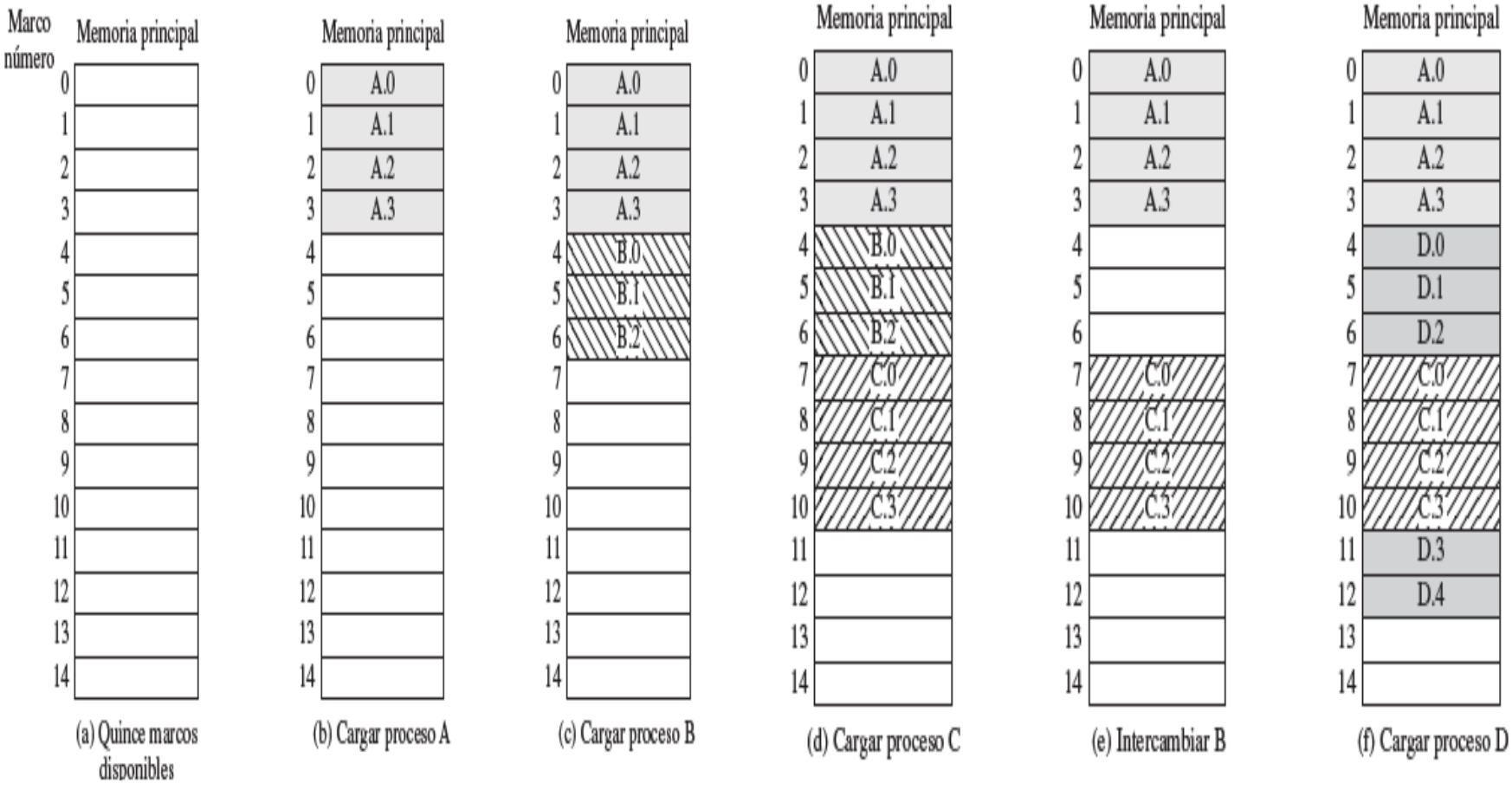
Direcciones

- **Lógica**
  - referencía a una posición de memoria independiente de la actual asignación de datos a memoria
  - se debe hacer traducción a la dirección física
- **Relativa**
  - la dirección se expresa como una posición relativa a un punto conocido
- **Física**
  - la dirección absoluta o posición actual

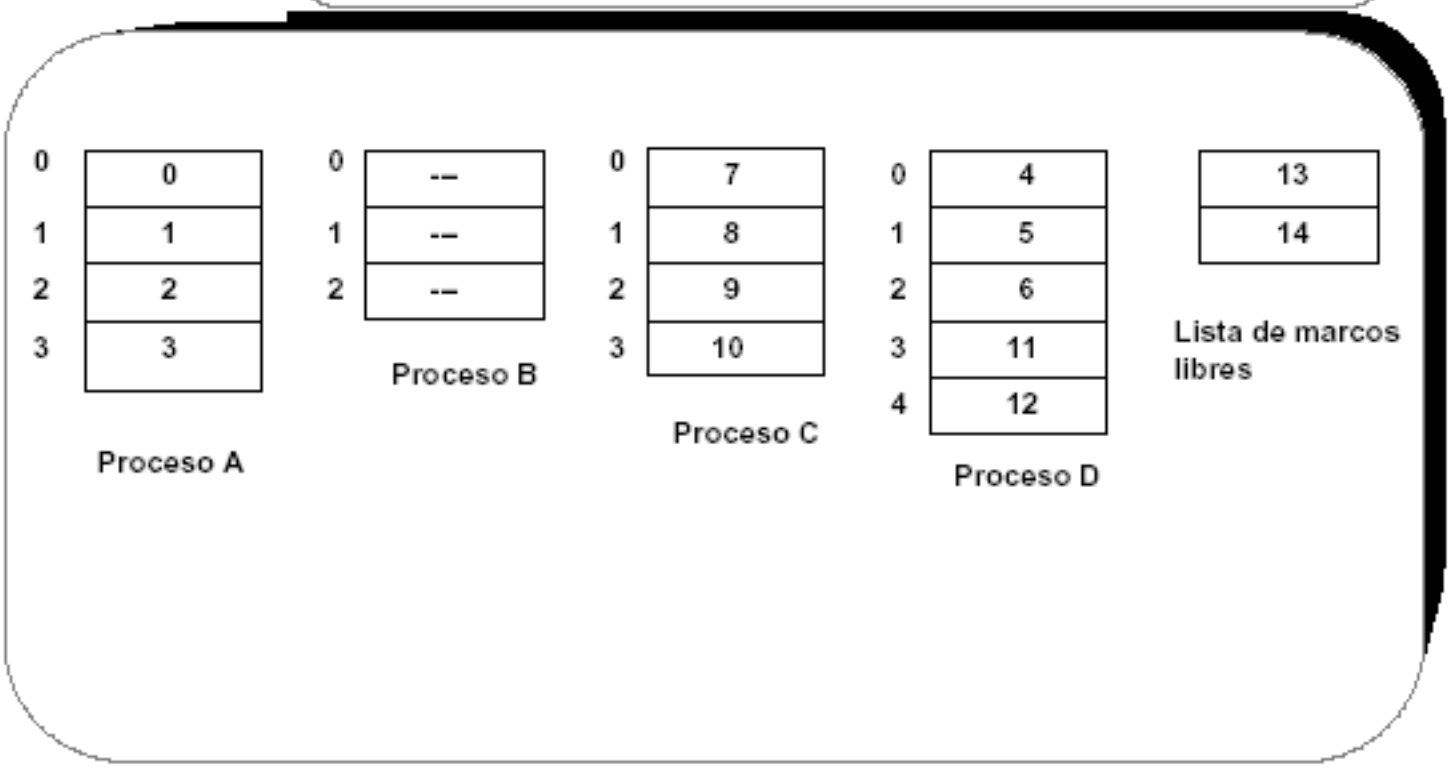
Paginación

- **Particionar la memoria en fragmentos pequeños del mismo tamaño y dividir cada proceso en fragmentos del mismo tamaño**
- **Los fragmentos de un proceso se llaman páginas y los fragmentos de memoria se llaman marcos.**
- **El SO mantiene una tabla de páginas por cada proceso**
  - contiene la posición del marco por cada página en el proceso
  - las direcciones de memoria consisten en un número de página y un desplazamiento dentro de la página

Paginación


Marco número
Memoria principal
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
A.0
A.1
A.2
A.3
B.0
B.1
B.2
C.0
C.1
C.2
C.3
D.0
D.1
D.2
D.3
D.4
(a) Quince marcos disponibles
(b) Cargar proceso A
(c) Cargar proceso B
(d) Cargar proceso C
(e) Intercambiar B
(f) Cargar proceso D

Ejemplo de tablas de páginas

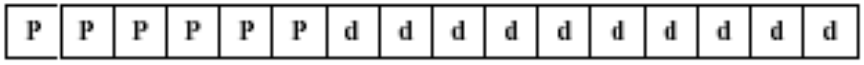


Direccionamiento lógico usado en la paginación

- Dentro de cada programa, cada dirección lógica debe consistir de un numero de página y un desplazamiento dentro de la página
- Un registro del CPU siempre tiene la dirección física inicial de la tabla de páginas del proceso que está en ejecución
- Dada una dirección lógica (número de página, desplazamiento) el procesador emplea la tabla de páginas para obtener la dirección física (número de marco, desplazamiento)

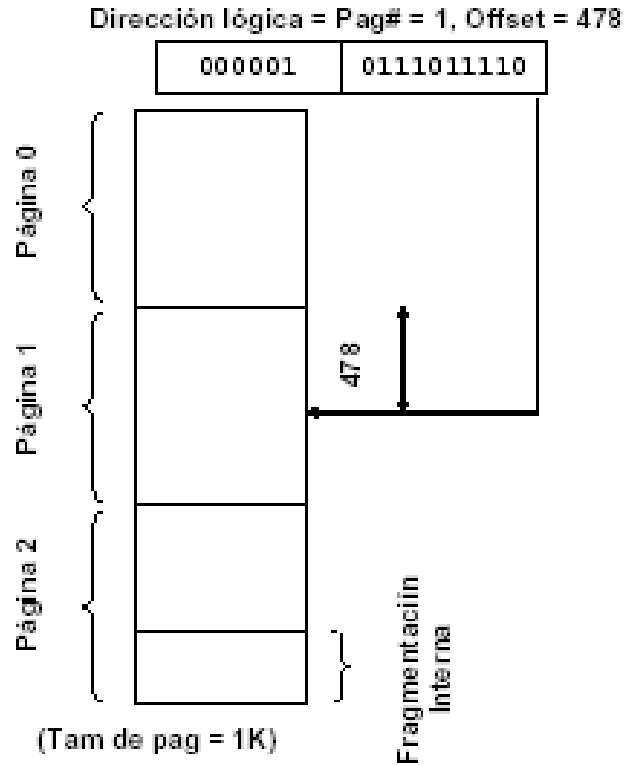
Direccionamiento lógico en la paginación

- **La dirección lógica es la misma que la dirección relativa cuando el tamaño de la página es una potencia de 2**
- **Ejemplo:**
  - Si se usan direcciones de 16 bits y el tamaño de página de = 1K
  - 10 bits para el desplazamiento y 6 bits para el número de página

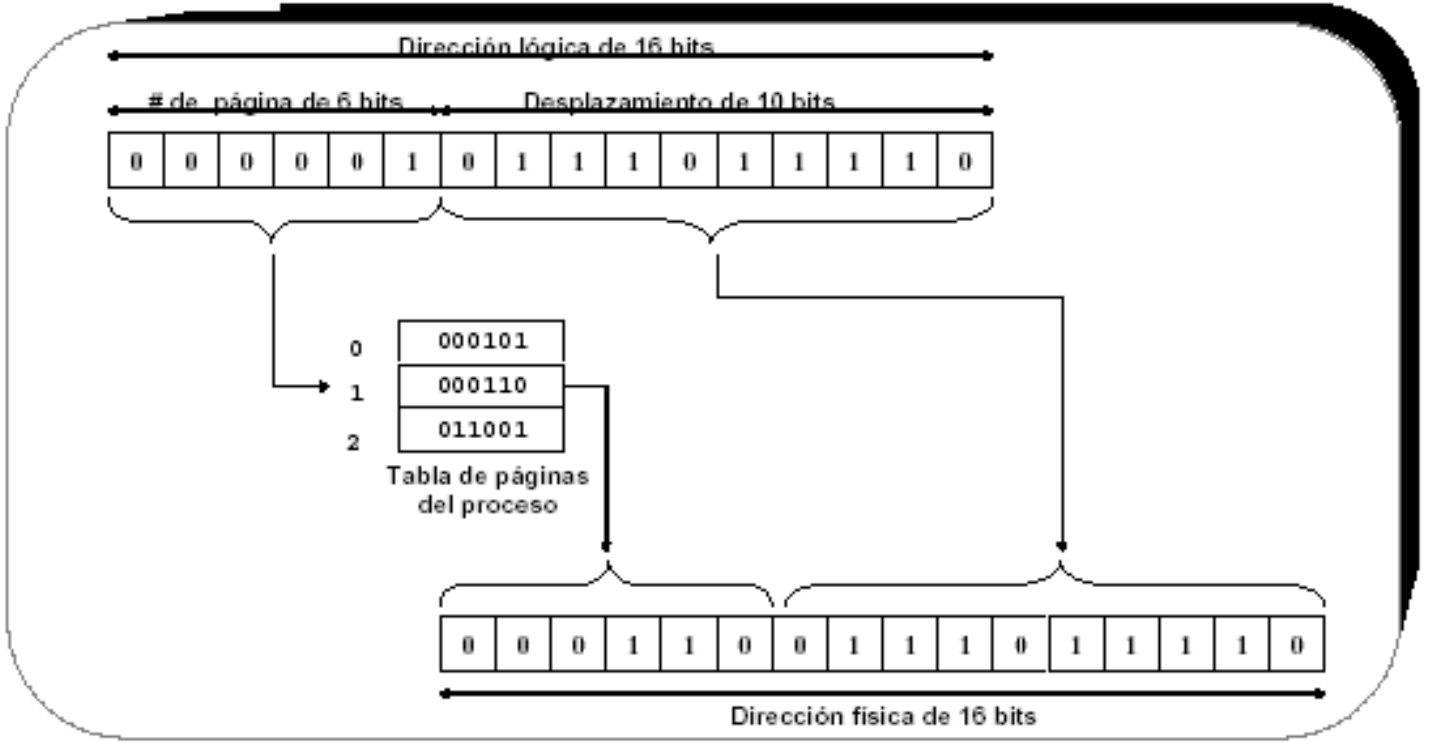


- **La dirección de 16 bits es una posición relativa al inicio del proceso**

Direccionamiento lógico en la paginación



Traducción de direcciones lógicas a físicas en paginación



Traducción de direcciones lógicas a físicas en paginación

- Usando un tamaño de página de una potencia de 2, las páginas son invisibles al programador, compilador/ensamblador, y encadenador

- La traducción de direcciones durante el tiempo de ejecución es entonces facil de implementar por hardware
  - dirección lógica (n,m) se traduce a la dirección física (k,m) indexando la tabla de páginas y añadiendo el mismo desplazamiento m al número de marco k

Segmentación

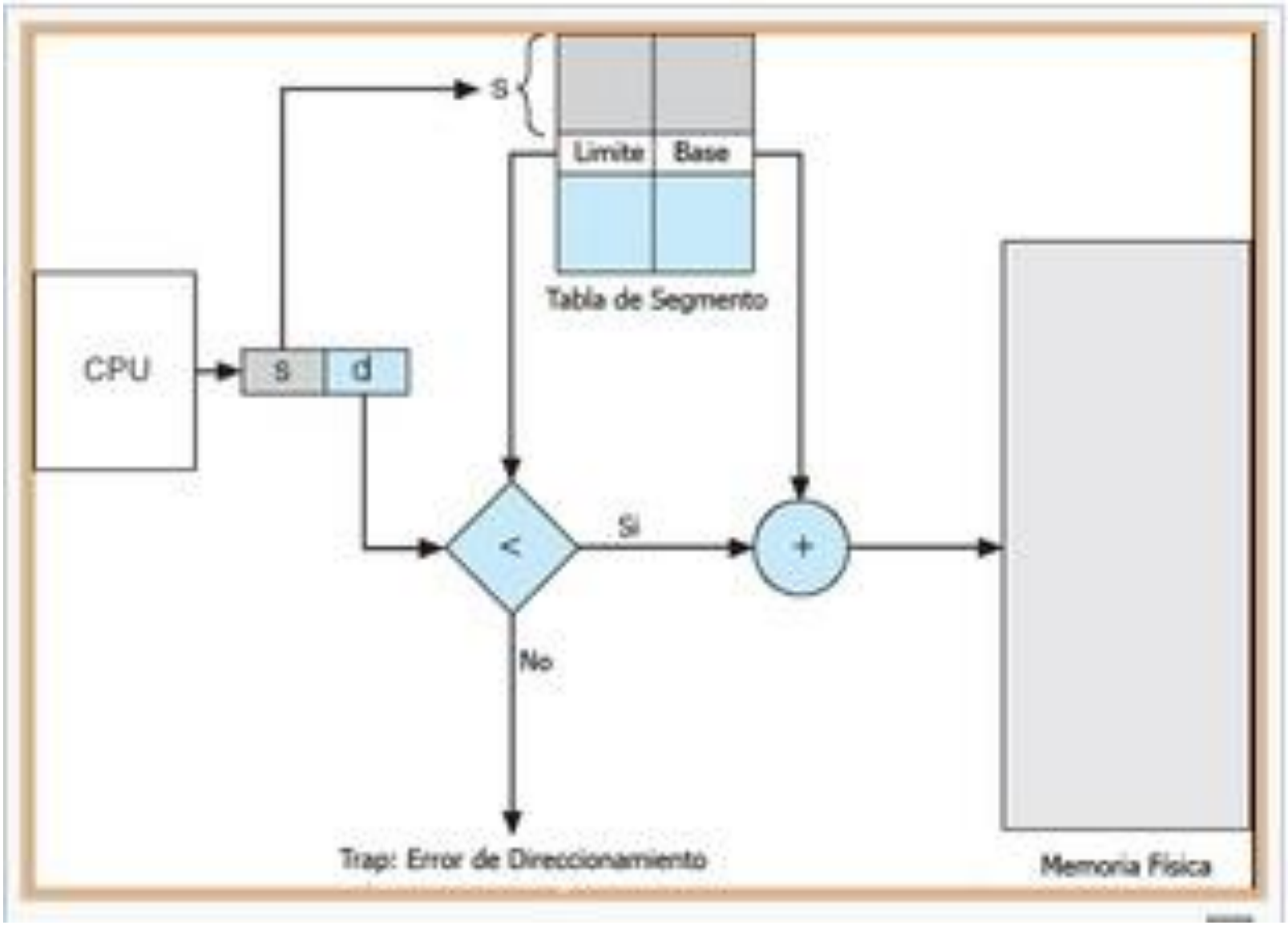
- Todos los segmentos de todos los programas no tienen que ser de la misma longitud
- Hay un máximo en la longitud de segmento
- El direccionamiento consiste de dos partes, un número de segmento y un desplazamiento
- Ya que los segmentos no son iguales, la segmentación es similar al particionamiento dinámico

Segmentación simple

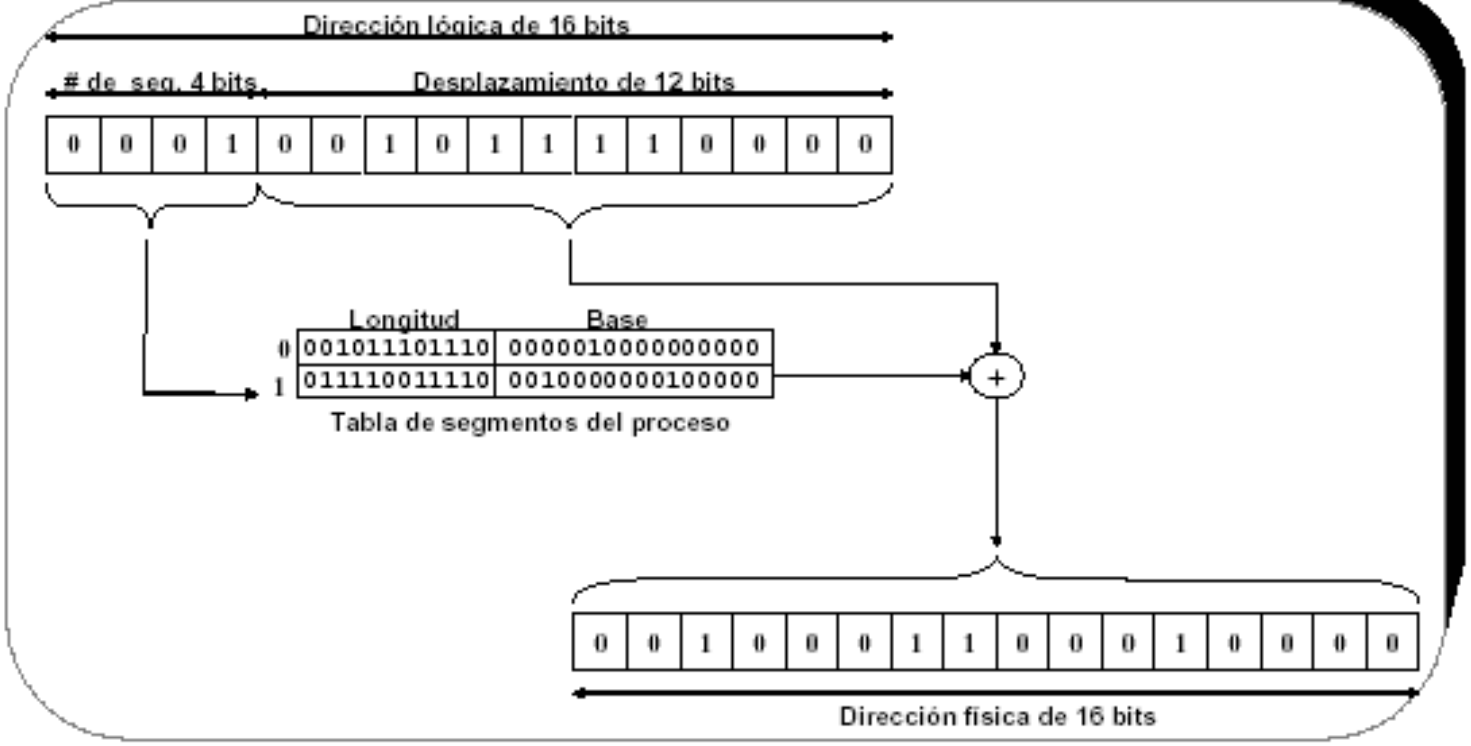
- **En contraste con la paginación, la segmentación es visible al programador**
  - Una comodidad para organizar lógicamente los programas
    - ejemplo: datos en un segmento, en otro código
  - Se debe estar al pendiente del tamaño máximo de los segmentos

Segmentación simple

- **El SO mantiene una tabla de segmentos para cada proceso. Cada entrada contiene:**
  - La dirección física inicial de este segmento.
  - La longitud del segmento por protección


s
Limite
Base
Tabla de Segmento
CPU
s
d
<
Si
+
No
Trap: Error de Direccionamiento
Memoria Física

Traducción de direcciones lógicas a físicas en la segmentación



Traducción de direcciones lógicas a físicas en la segmentación

- Cuando un proceso pasa al estado "ejecutando", se carga la dirección de su tabla de segmentos en un registro del CPU.
- Dada con una dirección lógica (número de segmento, desplazamiento) = (n,m), el CPU indexa (con n) la tabla de segmentos para obtener la dirección física inicial k y la lomgitud l de este segmento
- La dirección física se obtiene añadiendo m a k (en contraste con la paginación)
    - el hardware también compara el desplazamiento m con la longitud l de este segmento para determinar si la dirección es válida

Comparación entre segmentación y paginación

- **La segmentación**
  - requiere hardware más complejo para la traducción de direcciones
  - sufre de fragmentación externa
  - es visible al programador cuando la paginación es transparente
  - puede verse como comodidad al programador organizar lógicamente un programa en segmentos
    - para usar diferentes tipos de protección
      - ej. solo ejecución para el código, lectura y escritura para datos
      - necesitamos usar bits de protección en las entradas de tabla de segmentos

Comparación entre segmentación y paginación

- **La paginación**
  - presume de tener poca fragmentación interna
  - la paginación es transparente al programador