Unidad 2:
Gestión de Memoria

Tema 3, Gestión de Memoria:

3.1 Definiciones y técnicas básicas.

3.2 Gestión de memoria contigua:

Partición, fragmentación, algoritmos de ubicación...

3.3 Paginación:

Estructura de la tabla de paginas, tabla de pág. invertida,...

3.4 Segmentación y técnicas combinadas.

3.1 Definiciones y técnicas básicas.

- Introducción:
  - En la actualidad:
    - *El coste de memoria ha descendido mucho.*
    - *El tamaño de la memoria principal ha crecido mucho.*
  - Sin embargo **nunca hay suficiente memoria principal** para contener todos los programas y estructuras de datos.
  - Una de las tareas principales del SO es **gestionar la memoria** que supone cargar y descargar bloques desde y hacia el almacenamiento secundario.

3.1 Definiciones y técnicas básicas.

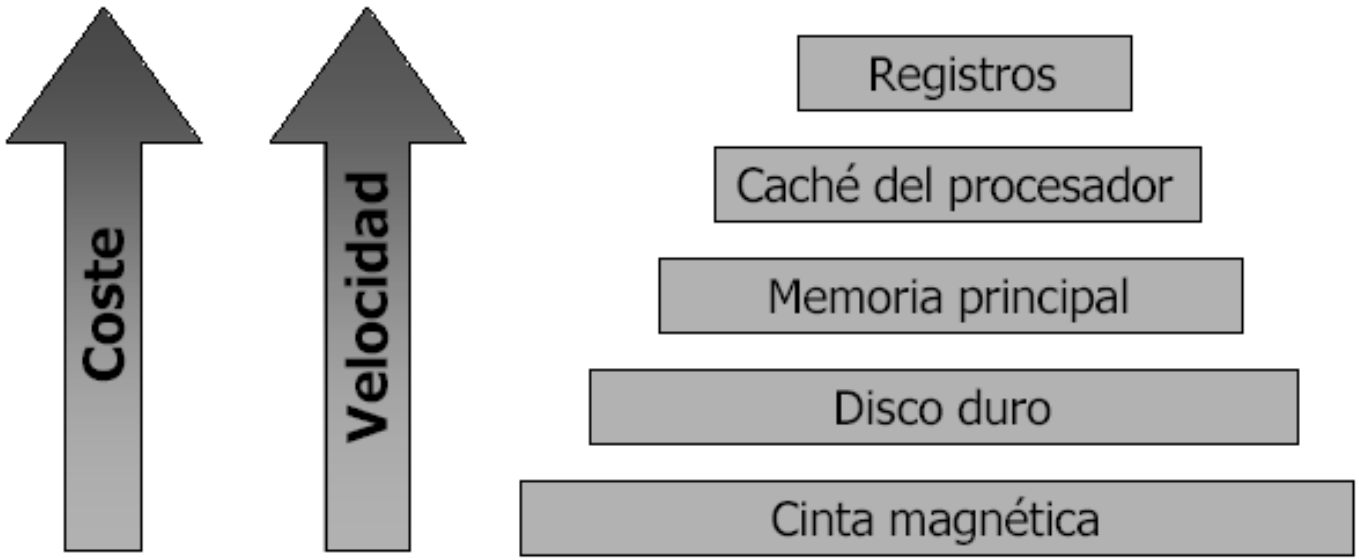
- Introducción: Definición de Memoria.
  - Memoria: amplia **tabla de datos**, cada uno de ellos con su propia dirección (conjunto de celdas referenciables por medio de una dirección lineal)
  - El **tamaño** de esa tabla **y los datos** incluidos en ella **dependen de cada arquitectura**.
  - Para que los programas puedan ser ejecutados, sus **códigos y** sus **datos** deben de estar **cargados en memoria principal.**
  - La información que es necesario almacenar se guarda en **dispositivos de almacenamiento secundario** (memoria secundaria).
  - El SO (sistemas multitarea) tratará de **repartir de forma eficiente la memoria** para introducir tantos procesos como sea posible.

3.1 Definiciones y técnicas básicas.

- Introducción: Localidad.
  - Los procesos se referencian y éstas referencias, en un intervalo de tiempo, se agrupan, en un ***subconjunto de espacios de direcciones***, llamado ***localidad***.
  - ***Localidad Espacial:*** Al referenciar una posición de memoria, **las posiciones (localidades) próximas también se ven referenciadas**. Este hecho lo observamos en:
    - Ejecución secuencial de código.
    - Colocación próxima de las variables relacionadas.
    - Acceso a estructuras de datos matriciales y pilas.
  - ***Localidad Temporal:*** Tras referenciar una posición de memoria en t, **probablemente vuelva a ser referenciada en t + Δt**:
    - Formación de ciclos.
    - Subrutinas.
    - Pilas.

3.1 Definiciones y técnicas básicas.

- Introducción: Jerarquía de Memoria.
  - Organización jerárquica según **coste, velocidad y tamaño**.
  - Nota: Son **volátiles** por encima de la memoria principal.



3.1 Definiciones y técnicas básicas.

- Introducción: Gestor de Memoria.
  - Parte del SO **encargada de asignar memoria a los procesos**, tratará de repartir de forma eficiente la memoria para introducir tantos procesos como sea posible.
  - Varios procesos podrán ejecutarse de forma concurrente teniendo en cuenta que:
    - **La memoria desaprovechada debe de ser la menor posible**.
      - Evitando 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.
    - **Debe de proporcionar protección y compartición**.
    - **No debe de perjudicar al rendimiento**, debiendo minimizar:
      - Complejidad de los procesos en el tiempo.
      - Procesos suplementarios (tiempos) de acceso a memoria.

3.1 Definiciones y técnicas básicas.

- Gestión de la memoria principal:
  - **Memoria principal: almacén de datos de acceso rápido**, que son **compartidos** por la CPU y los dispositivos de E/S.
  - Es el **único dispositivo de almacenamiento** grande que **la CPU puede direccionar y acceder directamente.**
  - Las **instrucciones** deben estar **en la MP** para que la CPU pueda ejecutarlas (es preciso cargar los programas en MP).
  - El SO se encarga de las siguientes actividades relacionadas con la gestión de memoria:
    - Saber **qué partes de la memoria se están usando**, cuáles están libres y quién las está usando.
    - Decidir **qué procesos cargar en la memoria**.
    - **Asignar y liberar espacio** de memoria.

3.1 Definiciones y técnicas básicas.

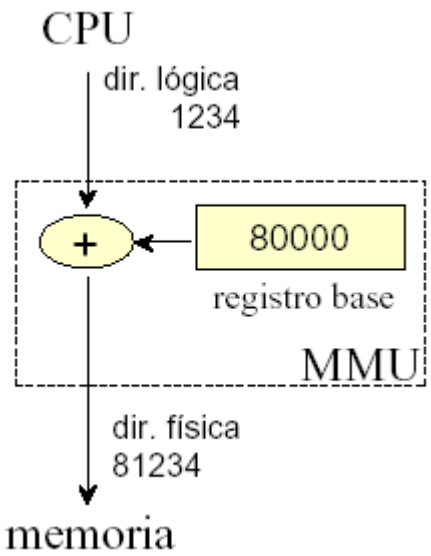
- Requisitos de la gestión de memoria (Reubicación):
  - Un problema asociado consiste en **saber si un proceso puede residir en cualquier parte de la memoria física**.
  - **Reubicación:**
    - El programador no conoce qué otros programas residirán en la memoria en el momento de la ejecución.
    - Mientras se está ejecutando el programa, puede que se descargue en el disco y que vuelva a la memoria principal, pero en una ubicación distinta a la anterior (reubicación).
    - Se deben traducir las referencias a la memoria encontradas en el código del programa a las direcciones físicas reales.

3.1 Definiciones y técnicas básicas.

- Reubicación, Vinculación de direcciones:
  - **Compilación:** Si en el momento de la compilación se sabe en qué parte de la memoria va a residir el proceso, es posible generar código absoluto.
  - **Carga:** Si al compilar el programa no se sabe en qué parte de la memoria va a residir el proceso, el compilador deberá generar código reubicable. En este caso la vinculación final se efectuará en el momento de la carga.
  - **Ejecución:** Si durante la ejecución los procesos cambian de segmento, la vinculación hay que realizarla durante la ejecución.

3.1 Definiciones y técnicas básicas.

- Reubicación, Direcciones lógicas/físicas:
  - ¿En qué momento (etapa) se realiza esta reubicación?
    - **Carga** (enlazador o cargador) => **Reubicación estática**.
    - **Ejecución** (hardware) => **Reubicación dinámica**.
  - **Reubicación dinámica:**



- ***Dirección física:*** la que llega a la memoria.
- ***Dirección lógica o virtual:*** la generada por la CPU.
- El ***MMU*** **(Unidad de Manejo de Memoria)** es el *dispositivo que traduce direcciones virtuales a físicas.*

3.1 Definiciones y técnicas básicas.

- Reubicación, Carga dinámica:
  - Para que un proceso se ejecute:
    - **Código + Datos => Memoria física**
  - Consecuencia:
    - **Tamaño de un proceso limitado al tamaño de la memoria física.**
  - Carga dinámica:
    - **Las rutinas no se cargan hasta que se invocan** (hasta que el programa llame a alguna rutina del mismo), mientras tanto permanecen en disco.

3.1 Definiciones y técnicas básicas.

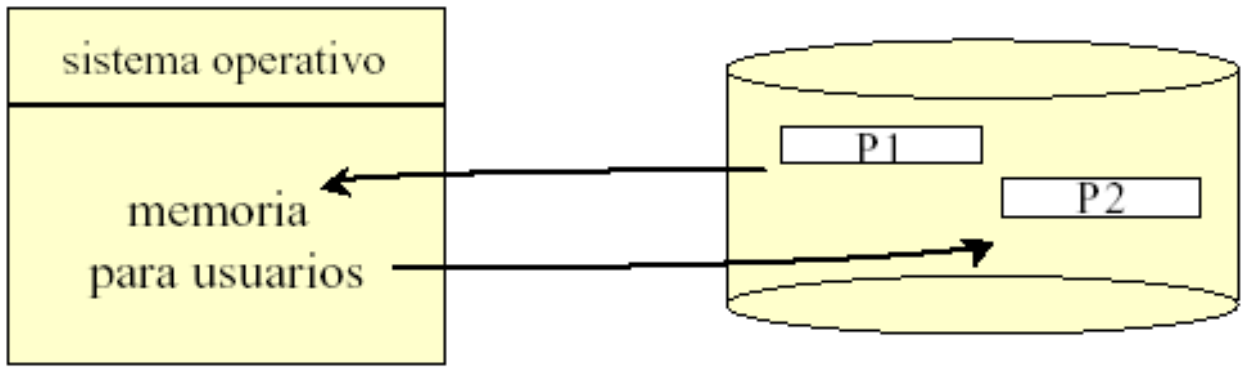
- Reubicación, Enlace dinámico:
  - Similar a la carga dinámica, pero efectuando el **enlace en tiempo de ejecución**: bibliotecas dinámicas (DLL).
    - Las bibliotecas del sistema no se enlazan de forma estática a los programas. Los programas incluyen un *fragmento* en la imagen por cada referencia a una rutina.
    - Este fragmento permite localizar y en su caso cargar la rutina necesaria en tiempo de ejecución.
    - Una vez localizada la rutina, el fragmento se sustituye en la imagen por la dirección de la rutina.
  - **Ejemplos de enlace dinámico:**
    - UNIX: shared libraries (shlib)
    - Windows: dynamic load libraries (dll).

3.1 Definiciones y técnicas básicas.

- Reubicación, superposiciones o recubrimientos (overlays):
  - Se utiliza **cuando un proceso es más grande que el tamaño de memoria que se le asigna.**
  - **Muchos programas** no necesitan todo el código al mismo tiempo, sino que **se ejecutan por fases**.
    - Se mantiene en memoria sólo lo que se necesita.
  - 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.

3.1 Definiciones y técnicas básicas.

- 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).**



3.1 Definiciones y técnicas básicas.

- Intercambio (swapping):
  - ***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.
  - ***¿Qué se necesita para llevarlo a cabo?***
    - Proceso intercambiador (tipo PMP).
    - Criterios para elegir víctima (política de swapping out).
    - Espacio en disco para almacenar la imagen de los procesos.
      - Área específica para el intercambio (área de swap).
      - Ficheros de intercambio.
    - Criterios para gestionar el espacio de intercambio (política de gestión del área de swap).

3.1 Definiciones y técnicas básicas.

- Requisitos de la gestión de memoria (Protección):
  - ***El código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos*** sin permiso.
  - ***Es imposible comprobar las direcciones absolutas de los programas***, puesto que se desconoce la ubicación de un programa en la memoria principal.
  - **Debe comprobarse durante la ejecución:**
    - El sistema operativo no puede anticiparse a todas las referencias a la memoria que hará un programa.

3.1 Definiciones y técnicas básicas.

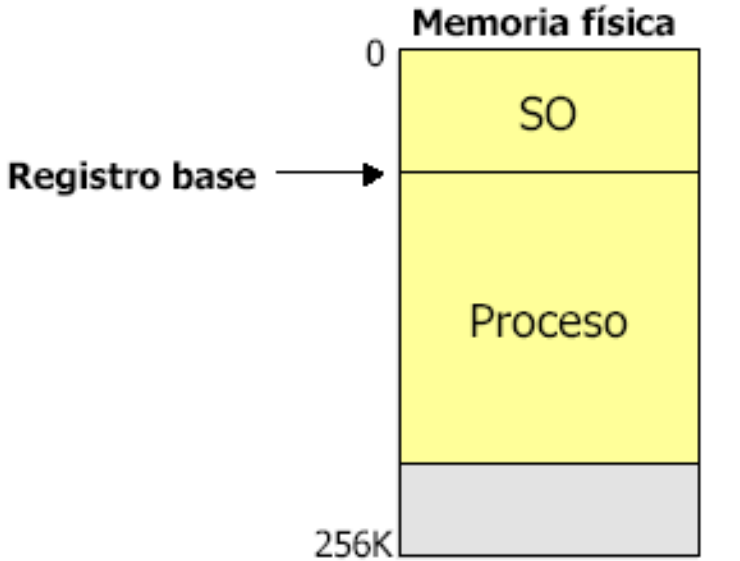
- Requisitos de la gestión de memoria (Compartimiento):
  - Permite el ***acceso de varios procesos a la misma zona de la memoria principal***.
  - Nota: Es mejor permitir a cada proceso que acceda a la misma copia del programa, en lugar de tener cada uno su propia copia aparte.

3.1 Definiciones y técnicas básicas.

- Requisitos de la gestión de memoria (Org. lógica/física):
  - **Organización lógica:**
    - Los programas se organizan en módulos.
    - Los módulos pueden escribirse y compilarse independientemente.
    - Pueden otorgarse distintos grados de protección (sólo lectura, sólo ejecución) a los módulos.
    - Compartir módulos.
  - **Organización física:**
    - La memoria disponible para un programa y sus datos puede ser insuficiente:
      - Mediante superposición varios módulos son asignados a la misma región de memoria.
    - El programador no conoce cuánto espacio habrá disponible.

3.2 Gestión de memoria contigua.

- Asignación contigua:
  - La memoria principal *debe dar cabida al SO y a los procesos de usuario.*
  - Generalmente *se divide la memoria en dos particiones*, una para el SO residente y otra para los procesos de usuario (se suele utilizar un registro base para proteger al SO).



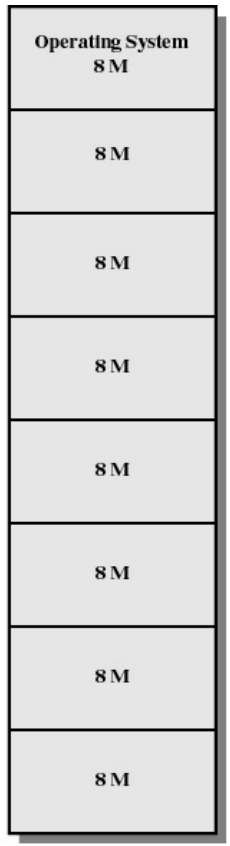
3.2 Gestión de memoria contigua.

- Particiones de la memoria:
  - **Particiones estáticas, particiones de igual tamaño:**
    - ***Cualquier proceso cuyo tamaño sea menor o igual que el tamaño de la partición puede cargarse en cualquier partición libre*.**
    - ***Si todas las particiones están ocupadas, el sistema operativo puede sacar un proceso de una partición.***
    - Un programa puede que no se ajuste a una partición. El programador debe **diseñar el programa mediante superposiciones.**
    - No requieren el uso de memoria virtual.
    - Es una política de gestión de memoria que prácticamente ha quedado obsoleta. Ej: SO IBM OS/360 - 1964.

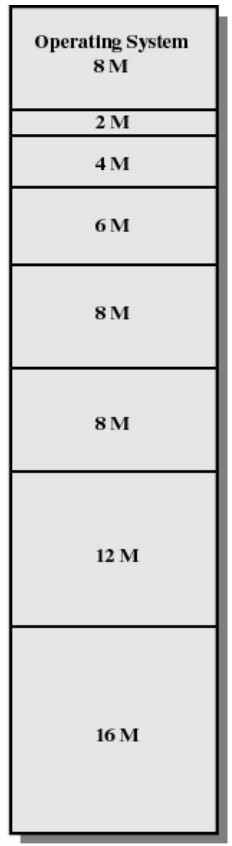
3.2 Gestión de memoria contigua.

- Particiones estáticas:
  - El *uso de la memoria principal es ineficiente*. Cualquier programa, sin importar lo pequeño que sea, ocupará una partición completa.

    Este fenómeno se denomina **fragmentación interna**.



(a) Equal-size partitions



(b) Unequal-size partitions

3.2 Gestión de memoria contigua.

- Particiones estáticas:
  - **Particiones de igual tamaño:**
    - Puesto que todas las particiones son de igual tamaño, no importa la partición que se use.
  - **Particiones de distintos tamaños:**
    - Pueden asignar cada proceso a la partición más pequeña en la que quepa.
    - Hace falta una cola para cada partición.
    - Los procesos están asignados de forma que se minimiza la memoria desaprovechada dentro de cada partición.

3.2 Gestión de memoria contigua.

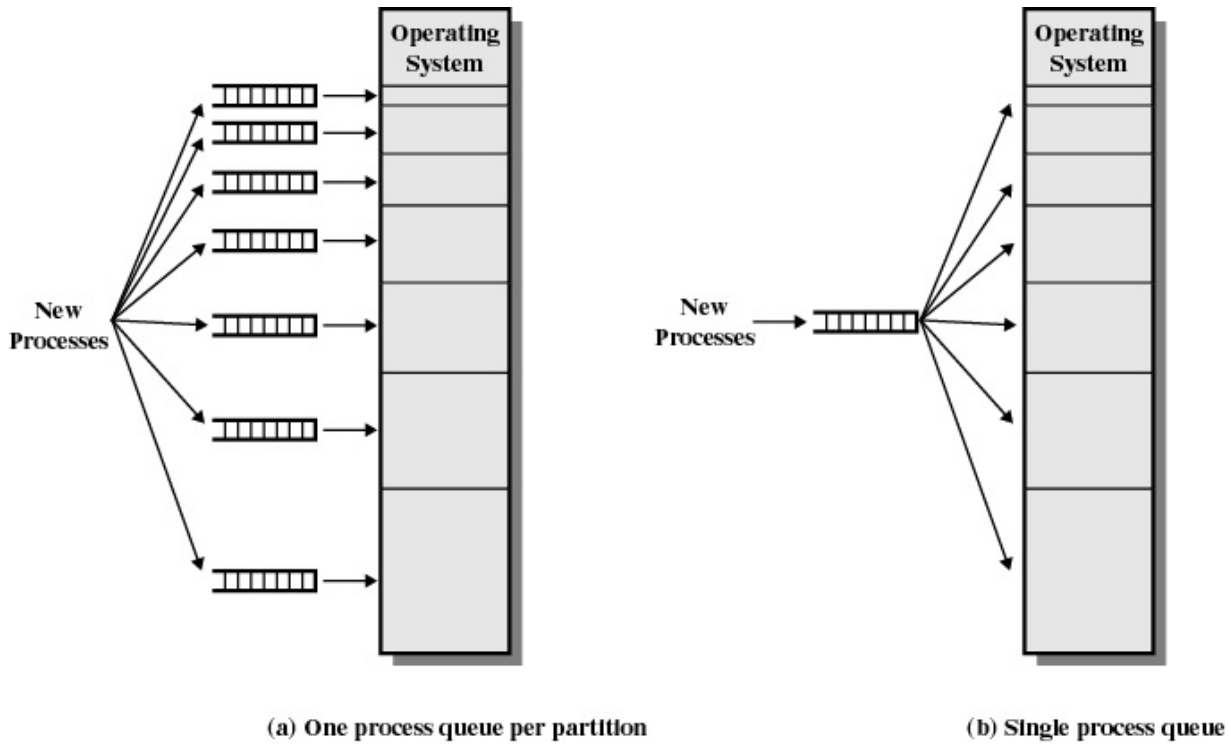
- Particiones estáticas:
  - **Presentan dos problemas:**
    - El programa puede ser mayor que los espacios de la partición => diseño del programa para que sólo una parte del programa esté en la memoria principal en cada instante => **Superposición**.
    - Uso de la memoria principal ineficiente, cualquier programa aunque sea pequeño ocupará toda una partición => desaprovechamiento de memoria => **Fragmentación**.

3.2 Gestión de memoria contigua.

- Fragmentación:
  - Desaprovechamiento de memoria por haber realizado una mala partición.
  - Puede ser de dos tipos:
    - ***Fragmentación interna:*** Debida a la diferencia de tamaños entre la partición de memoria y el objeto residente en ella.
    - ***Fragmentación externa:*** Desaprovechamiento de memoria entre particiones.

3.2 Gestión de memoria contigua.

- Fragmentación:
  - **Algoritmo de ubicación:**



(a) One process queue per partition

(b) Single process queue

3.2 Gestión de memoria contigua.

- Fragmentación (Algoritmo de ubicación):
  - Si cada partición tuviera asociada una cola de planificación de procesos (agrupando los procesos por tamaños) se minimiza la fragmentación interna. Pero si hay muchos procesos de un mismo tamaño pueden quedar particiones sin usar.
  - Una única cola de planificación para todos los procesos: Cuando el proceso se va a cargar se selecciona la partición más pequeña disponible que pueda albergar el proceso.

3.2 Gestión de memoria contigua.

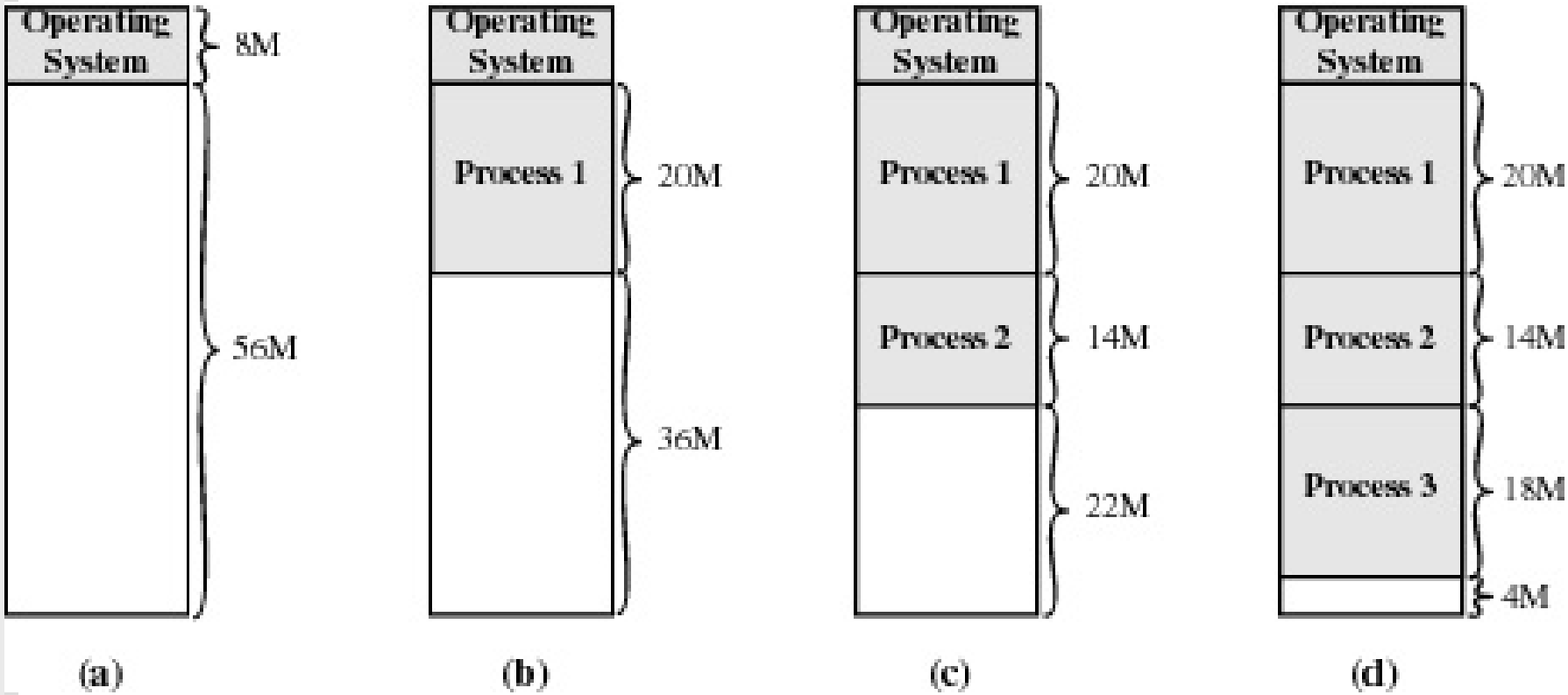
- Desventaja de las particiones estáticas:
  - 1. El número de particiones en el momento de la generación del sistema limita el número de procesos activos en el sistema.
  - 2. Puesto que los tamaños de las particiones se han programado previamente => los procesos pequeños hacen un uso muy ineficiente del espacio de las particiones.

3.2 Gestión de memoria contigua.

- Particiones Dinámicas:
  - Las **particiones son variables en número y longitud**.
  - **Cuando se carga un proceso en la memoria principal se le asigna exactamente tanta memoria como necesite**.
  - Finalmente, hay **varios huecos en la memoria**. Este fenómeno se denomina **fragmentación externa**.
  - Se debe usar la compactación para desplazar los procesos que estén contiguos, de forma que toda la memoria libre quede junta en un bloque.

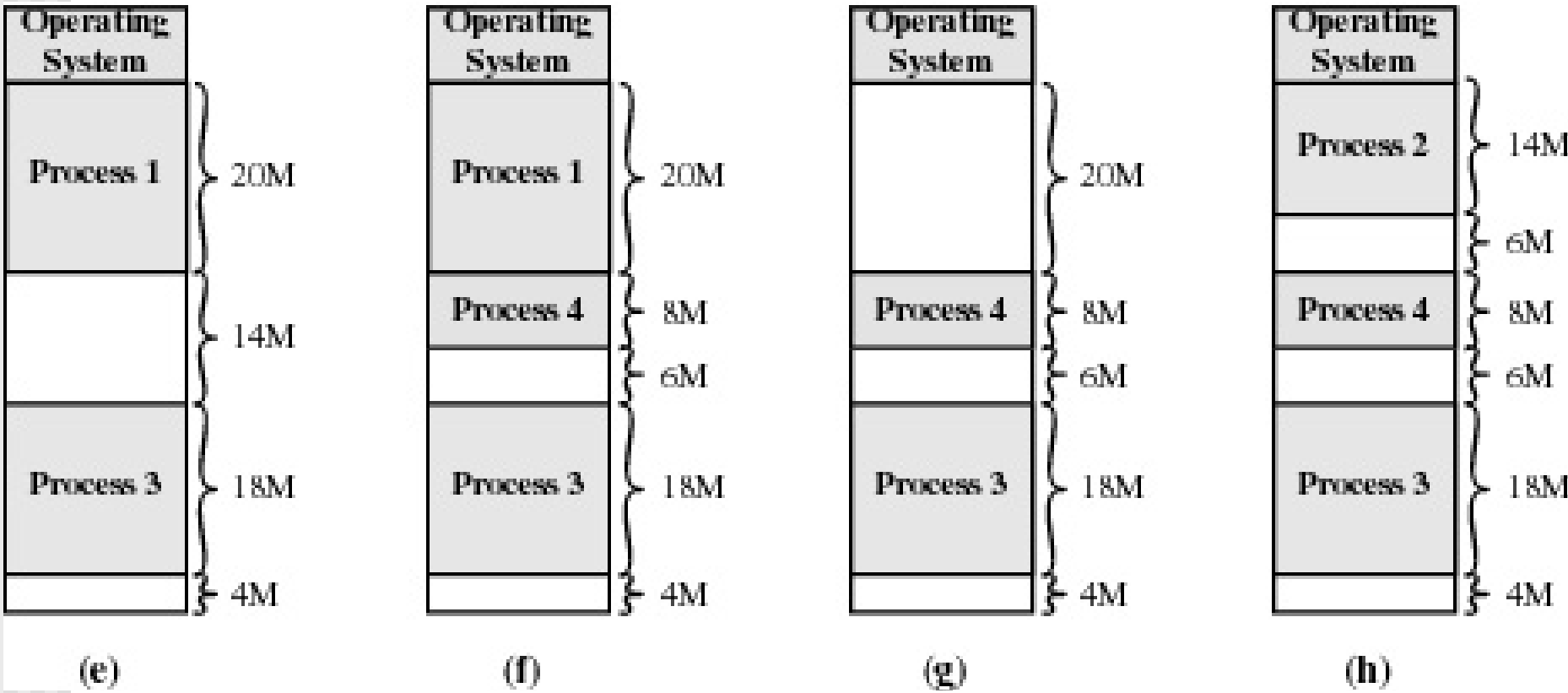
3.2 Gestión de memoria contigua.

- Efectos de la partición dinámica:



3.2 Gestión de memoria contigua.

- Efectos de la partición dinámica:



3.2 Gestión de memoria contigua.

- Solución: Compactación.
  - Para evitar esta fragmentación el SO desplaza los procesos para que estén contiguos de forma que todos los espacios de memoria libre se agrupen en un bloque.
  - Consume tiempo de procesado.
  - Necesita la capacidad de reubicación dinámica, es decir, poder mover un programa de una región a otra de la memoria principal, sin invalidar las referencias a la memoria del programa.

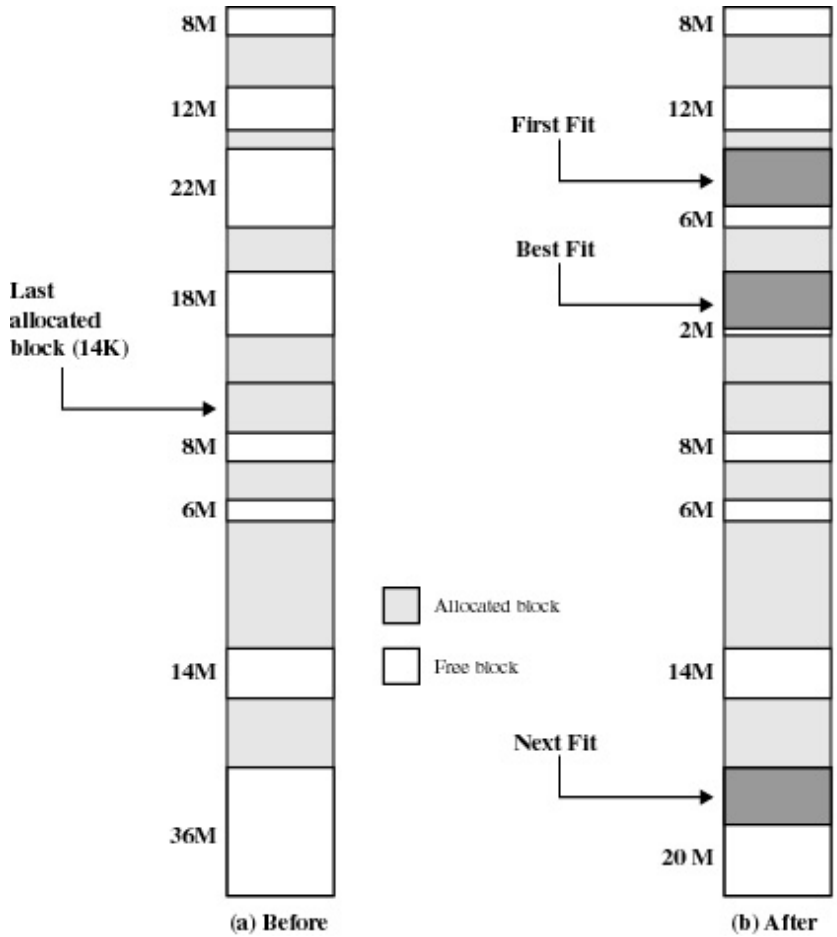
3.2 Gestión de memoria contigua.

- Algoritmos de ubicación con particiones dinámicas:
  - El SO debe decidir qué bloque libre se tiene que asignar al proceso.
  - Ejemplos de algoritmos de ubicación:
    - **First Fit (Primer ajuste):** Selecciona el primer bloque disponible de tamaño suficientemente grande.
      - Nota: Suele ser el más eficiente.
    - **Best Fit (Mejor ajuste):** Selecciona el bloque disponible de tamaño más próximo al solicitado.
    - **Next Fit (Siguiente ajuste):** Desde la última ubicación y elige el bloque disponible suficientemente grande.
      - Nota: Necesidad de compactación frecuente.

3.2 Gestión de memoria contigua.

- Algoritmos de ubicación con particiones dinámicas:

Ejemplo: Asignación en memoria de un bloque de 16M, mediante algoritmos de ubicación.

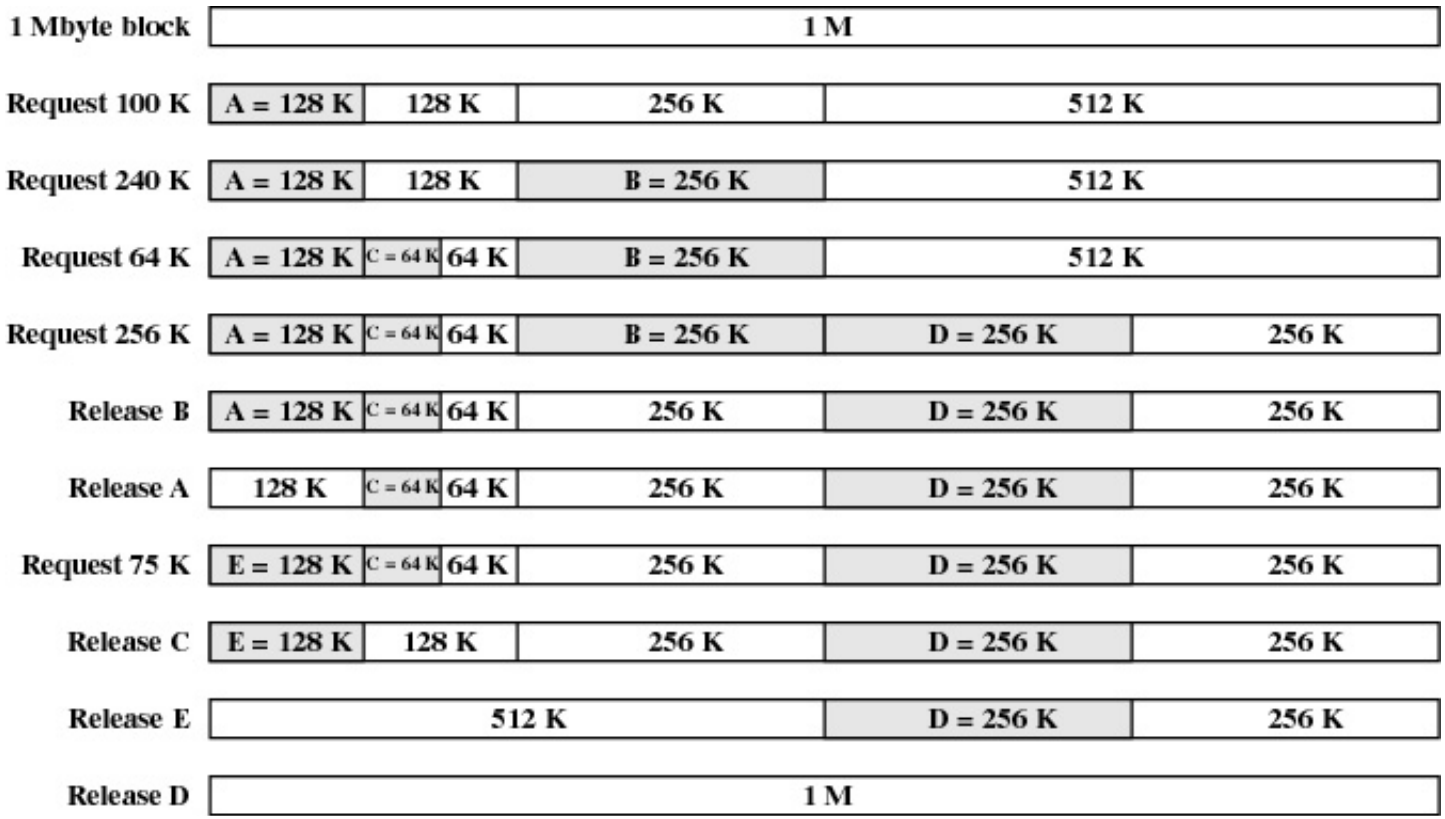


3.2 Gestión de memoria contigua.

- Sistema de los colegas:
  - Sistema de los socios: Busca el equilibrio entre la ineficiencia de las particiones estáticas y la complejidad de mantener las particiones dinámicas.
  - El espacio entero disponible para la asignación se trata como un solo bloque de tamaño 2^U.
  - Si se hace una solicitud de tamaño s tal que $2^{U-1} < s <= 2^U$, entonces el bloque entero se asigna:
    - En otro caso, el bloque se divide en dos colegas de igual tamaño.
    - Este proceso continúa hasta que el bloque más pequeño sea mayor o igual que s generándose.

3.2 Gestión de memoria contigua.

- Sistema de los colegas:



3.2 Gestión de memoria contigua.

- Reubicación:
  - Cuando el proceso se carga en la memoria, se determina la ubicación real (absoluta) de la memoria.
  - Un proceso puede ocupar diferentes particiones, lo que significa diferentes posiciones absolutas de la memoria durante su ejecución (a partir de la carga).
  - La compactación también hará que un programa ocupe una partición distinta, lo que significa que las ubicaciones absolutas de la memoria cambien.

3.2 Gestión de memoria contigua.

- Direcciones:
  - **Dirección lógica:**
    - Es una referencia a una posición de memoria independiente de la asignación actual de datos a la memoria.
    - Se debe hacer una traducción a una dirección física.
  - **Dirección relativa:**
    - La dirección se expresa como una posición relativa a algún punto conocido.
  - **Dirección física:**
    - La dirección absoluta o la posición real en la memoria principal.

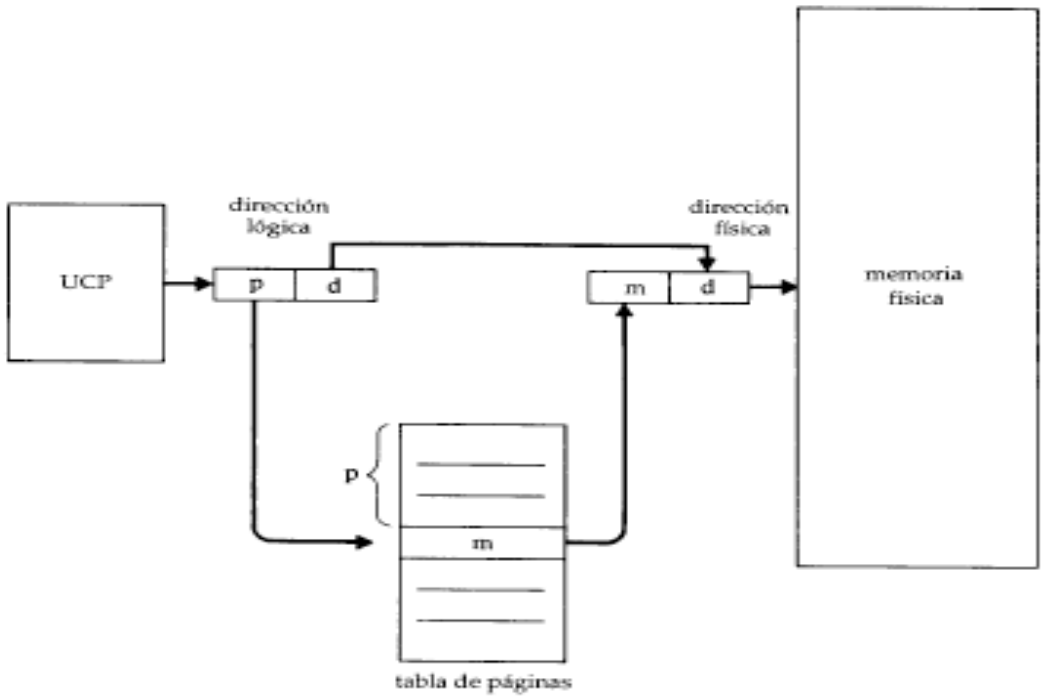
3.3 Paginación.

- Paginación:
  - ***Solución al problema de la fragmentación externa.***
  - Idea fundamental:
    - Dividir la memoria física (principal) en bloques iguales de tamaño fijo relativamente pequeños llamados **marcos**.
    - La memoria lógica (procesos) se divide en bloques del mismo tamaño llamados **páginas**.
    - **Ejecución:** Las páginas se cargan desde el almacenamiento auxiliar a un marco de memoria que esté disponible.
  - El sistema operativo mantiene una tabla de páginas para cada proceso:
    - Muestra la posición del marco de cada página del proceso.
    - **La dirección de la memoria consta de un número de página y de un desplazamiento dentro de la página**.

3.3 Paginación.

- Soporte hardware para la paginación:
  - Cada dirección generada por la CPU se divide en:

    - **Número de página** "p": índice de la tabla de páginas.
    - **Desplazamiento** en la página "d".



3.3 Paginación.

- Soporte hardware para la paginación:
  - ***La dirección base de cada página (número de página) se combina con el desplazamiento para definir la dirección de memoria física.***
  - El SO controla la utilización de memoria mediante una tabla de marcos de página.
  - El tamaño de la página y del marco está definido por el hardware y suele ser una potencia de dos que varía entre los 512 B y los 16 MB.
  - Cuando se utiliza paginación las direcciones relativas y lógicas y coinciden.
  - ***La paginación puede producir fragmentación interna.***

3.3 Paginación.

- Inconveniente: Fragmentación interna.
  - Sucede cuando las necesidades de memoria de un proceso no coinciden con los tamaños de las páginas.
  - **¿Tamaño de las páginas?**
    - **Pequeño:**
      - Mejora la fragmentación interna.
      - Aumenta el tamaño de la tabla de páginas.
    - **Grande:**
      - Peor desde el punto de vista de la fragmentación interna.
      - Tamaño de las tablas de páginas menor.
      - La E/S de disco es más eficiente cuando la cantidad de datos transferidos es mayor.
    - **Tendencia en los últimos años:**
      - Aumentar el tamaño a medida que los procesos, los conjuntos de datos y la memoria principal se han vuelto más grandes.
      - 2-4 KB.

3.3 Paginación.

- Estructura de la tabla de páginas:
  - Cada SO tiene sus propios métodos para almacenar la TDP.
  - Denominador común:
    - **Una tabla de páginas para cada proceso.**
  - **¿Cómo localiza el SO la TDP de un proceso?**
    - **BCP:**
      - Contador de instrucciones, registros, información de E/S, etc, ... Y
      - Puntero a la TDP.
  - **¿Qué ocurre en un cambio de contexto?**
    - Despachador cargará los registros con los valores del nuevo proceso.
    - A partir de la TDP almacenada, cargará los valores correctos de la TDP en hardware.

3.3 Paginación.

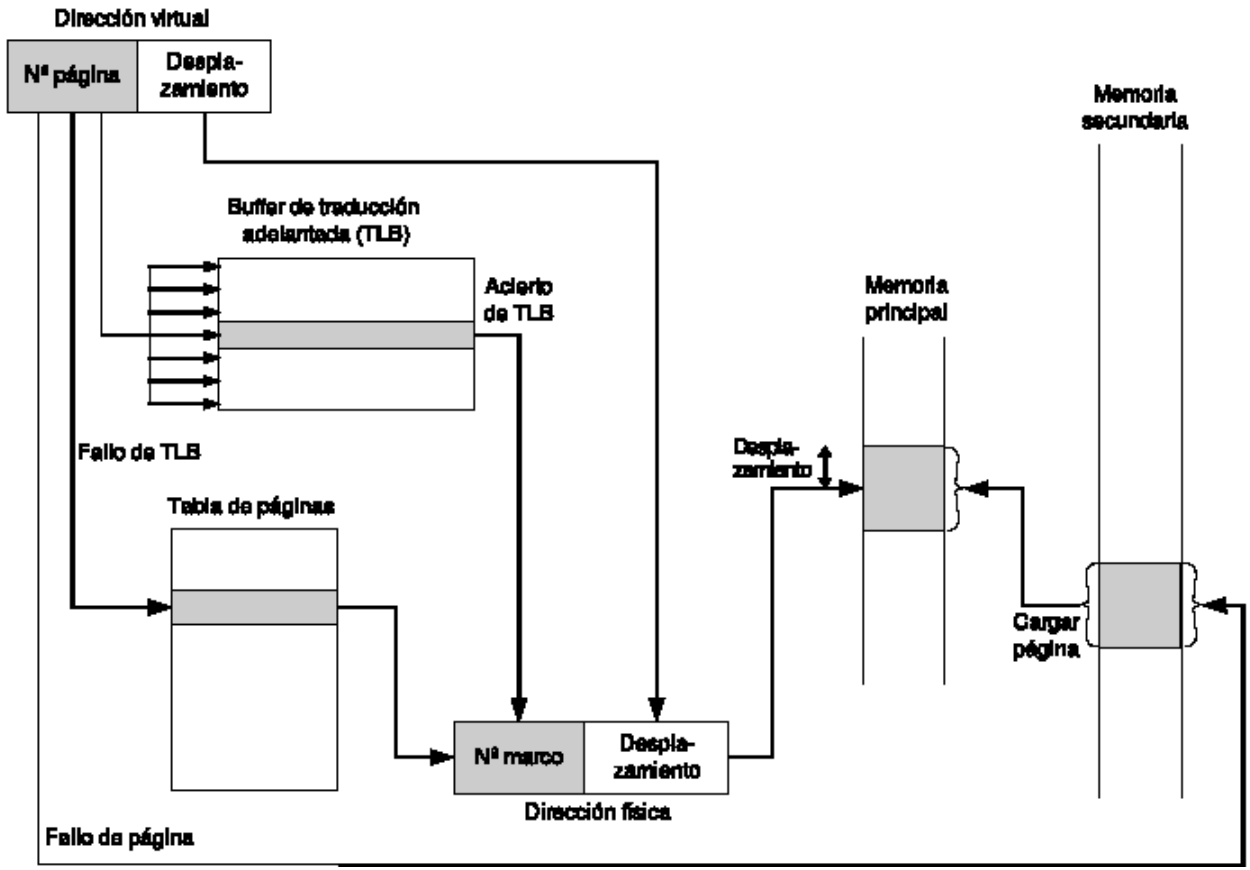
- Implementación en hardware de la tabla de páginas:
  - **Una TDP se implementa en el hardware como un conjunto de registros.**
  - **Esquema de registros:**
    - **Problema:** El empleo de registros para la TDP es satisfactorio si la tabla es razonablemente pequeña.
    - **Solución:**
      - Mantener la TDP en memoria.
      - Registro base de la tabla de páginas que apunta a la TDP:
        - Cambio de contexto: más rápido (sólo cambiar el valor de este registro).
        - Gran inconveniente: tiempo de traducción.

3.3 Paginación.

- Implementación en hardware de la tabla de páginas:
  - **Solución: usar un TLB (Translation Lookahead Buffer, o tabla de registros asociativos).**
    - Pequeño caché 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.

3.3 Paginación.

- TLB:



3.3 Paginación.

- TLB:
  - **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 TDP y actualizar TLB.
      - Si TLB llena => Sustitución de una de las existentes.
  - Ojo, **cambio de contexto**:
    - Desalojar (borrar) el TLB.
  - **Tasa de aciertos:**
    - Porcentaje de las veces que un número de página se encuentra en los registros asociativos.
    - Buenas tasas de aciertos: 80% - 98%
  - Ejemplo: Intel 80486 => TLB de 32 entradas.
    - Sus fabricantes dicen que tiene una tasa de aciertos del 98%.

3.3 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.
  - Sin embargo, un proceso casi nunca utiliza todo su intervalo de direcciones.
    - En este caso sería un desperdicio crear una TDP con entradas para todas las páginas del intervalo de direcciones.
  - Algunos sistemas: **registro de longitud de la TDP**
    - Indica el tamaño de la TDP y se coteja con cada dirección lógica para asegurar que la dirección esté en el intervalo válido para el proceso.

3.3 Paginación.

- Compartición:
  - Varios procesos podrían tener la misma memoria física apuntada en sus respectivas TDP´s.
  - La compartición de código exige que el código sea reentrante, es decir, no puede modificarse a sí mismo.

3.3 Paginación.

- Paginación multinivel:
  - Sistemas modernos => espacio de direcciones lógico muy grande (2^{32} a 2^{64}) => **la TDP crece demasiado.**
  - **Problema: tamaño de la TDP.**
    - Por ejemplo, si el tamaño de página es de 4k, un proceso podría requerir hasta 4Mb de espacio físico para la TDP.
    - Solución: paginar la TDP teniendo varios niveles de páginas (ej: 80336).

3.3 Paginación.

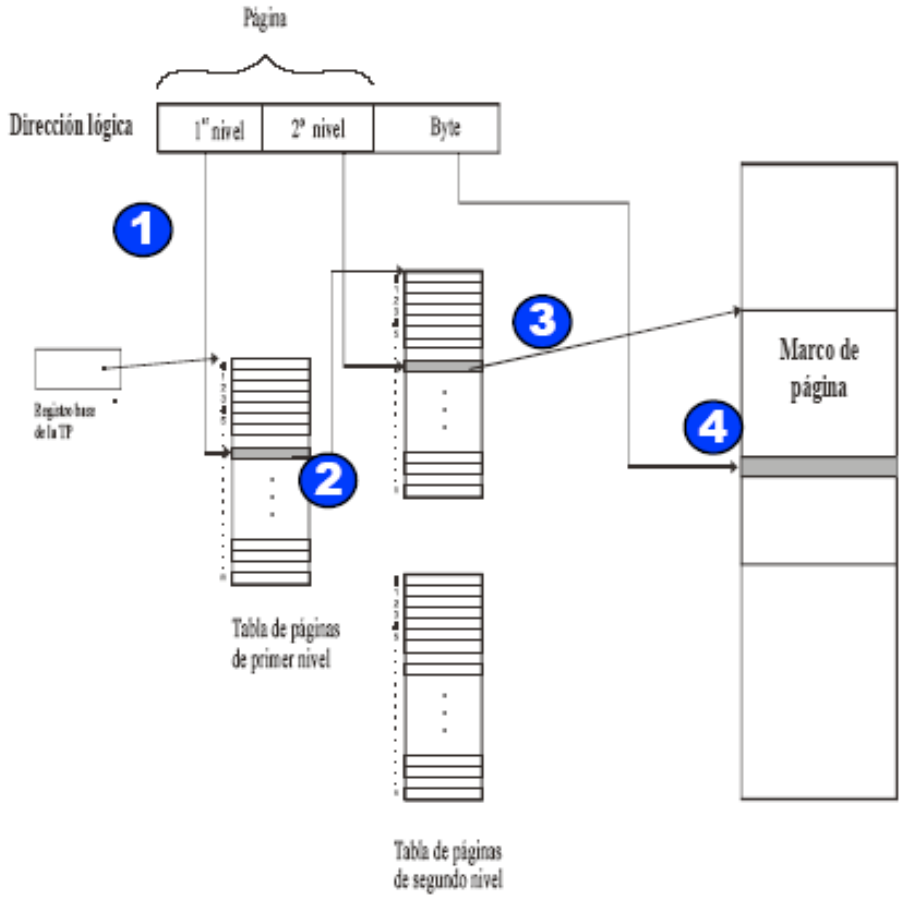
Paginación multinivel:

1. La MMU toma la parte asociada al nº de página del primer nivel.

2. Busca de entrada de la TDP: Obteniendo el 2º nivel.

3. Busca la entrada de 2º nivel: Obtiene el marco de página.

4. Indexa el byte dentro del MP de memoria física.



3.3 Paginación.

- Tabla de páginas invertida:
  - **Problema:** tamaño que puede llegar a ocupar la TDP, ya que la TDP puede contener millones de entradas que podrían consumir grandes cantidades de memoria.
  - **Idea: usar una tabla de páginas invertida.**
    - Tiene una entrada por cada marco real de la memoria.
    - Cada entrada consiste en la dirección virtual de la página almacenada en esa posición de memoria real => sólo hay una tabla de páginas en el sistema y sólo tiene una entrada por cada página de memoria física.
  - **Ventaja:** Reduce la cantidad de memoria necesaria.
  - **Desventaja:**
    - Tiempo de búsqueda en la tabla de páginas invertida.
    - Soluciones:
      - Tabla de dispersión y Registros asociativos (caché).

3.4 Segmentación.

- Segmentación:
  - ***Memoria como una colección de segmentos de tamaño variable.***
  - Un espacio de direcciones lógico es una colección de segmentos, cada uno de ellos con nombre y longitud.
  - ***Una dirección contiene el nombre del segmento y un desplazamiento.***
  - No es necesario que todos los segmentos de todos los programas tengan la misma longitud.
  - Existe una longitud máxima de segmento.
  - Como consecuencia del empleo de segmentos de distinto tamaño, la segmentación resulta similar a la partición dinámica.

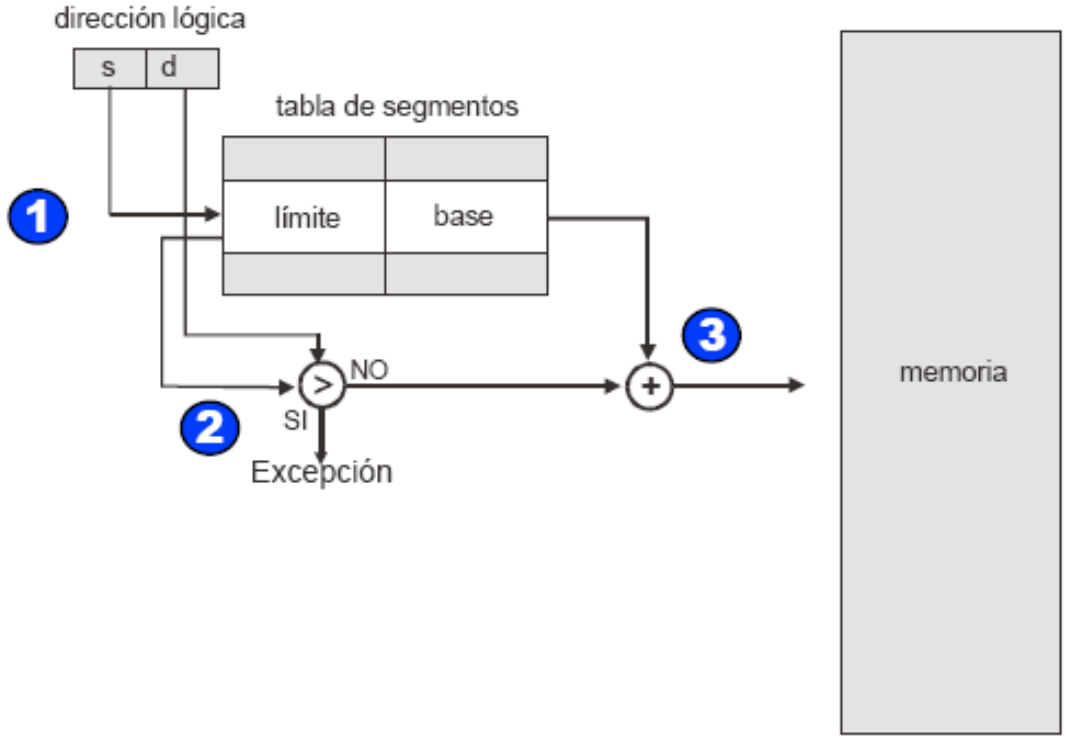
3.4 Segmentación.

- Hardware de segmentación:

1. La MMU toma la parte asociada al nº de segmento se busca en la TDS.

2. Se verifica si el desplazamiento está dentro de los límites.

3. Indexa el byte dentro de la memoria física. Sumando el desplazamiento base.



3.4 Segmentación.

- Hardware de segmentación:
  - Una dirección lógica tiene dos partes:
    - **Número de segmento: "s"** => índice de la tabla de segmentos.
    - **Desplazamiento dentro del segmento: "d"** => entre 0 y el límite (longitud) del segmento.
  - **Proceso de traducción:** dirección lógica (s,d).
    - Se comprueba que s<RLTDS (Registro de longitud de la TDS).
    - Se calcula la dirección de la entrada de la tabla de segmentos (Registro Base de la TDS + 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.

3.4 Segmentación.

- Implementación de la TDS:
  - La tabla de segmentos se puede colocar en registros rápidos o en memoria. (Una TDS que se mantiene en registros se puede consultar rápidamente).
  - Si los programas manejan muchos segmentos, podemos tener un registro base de la tabla de segmentos, que apunta a la tabla de segmentos.
  - Se puede usar un conjunto de registros como caché de entradas de la TDS.

3.4 Segmentación.

- Ventajas de la segmentación:
  - **Protección y compartimiento:**
    - Se puede establecer protección a nivel de segmentos: los segmentos de código no se modifican.
    - Un segmento puede compartirse haciendo que alguna entrada de la TDS de dos procesos distintos apunten al mismo segmento de memoria.
    - La segmentación produce fragmentación externa.
    - Puede utilizarse paginación para encontrar memoria suficiente, o intentar ejecutar otros procesos más pequeños, aunque tengan menor prioridad.

3.4 Segmentación.

- Técnica combinada: Segmentación paginada.

  - **Paginación y segmentación pueden combinarse** (ej. 80386) con el fin de aprovechar las ventajas que ofrecen ambas políticas por separado.
    - **Segmentación:** *Flexibilidad y facilidad para la organización lógica.*
    - **Paginación:** *Mejorar el problema de la fragmentación* (importante para segmentos muy grandes).

  - **Solución => paginar los segmentos:**
    - Las páginas evitan la fragmentación.
    - Simplificación de la asignación de memoria: Cualquier hueco libre ahora es válido.

3.4 Segmentación.

- Segmentación paginada: Esquema de traducción.

1. La MMU toma la parte asociada al nº de Segmento y lo busca en la TDS.

2. Se verifica que el desplazamiento esté dentro de los límites.

3. Se obtiene la TDP de la TDS y se determina el MP.

4. Indexa el byte dentro del MP de la memoria física.