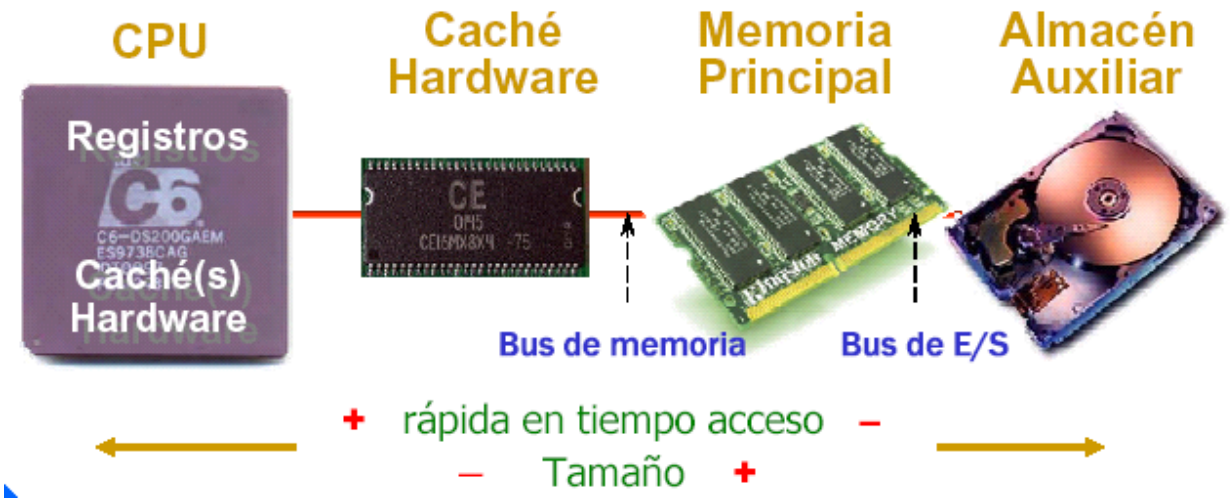
GESTIÓN DE MEMORIA

Karim Guevara Puente de la Vega

2017

Introducción

- La ley de Parkinson dice que *"los programas se expanden con el fin de llenar la memoria disponible para contenerlos"*.
- Usuarios: memoria privada, de tamaño y rapidez infinitas, no volátil y barata.
- Los computadores usan una **jerarquía de memoria**.

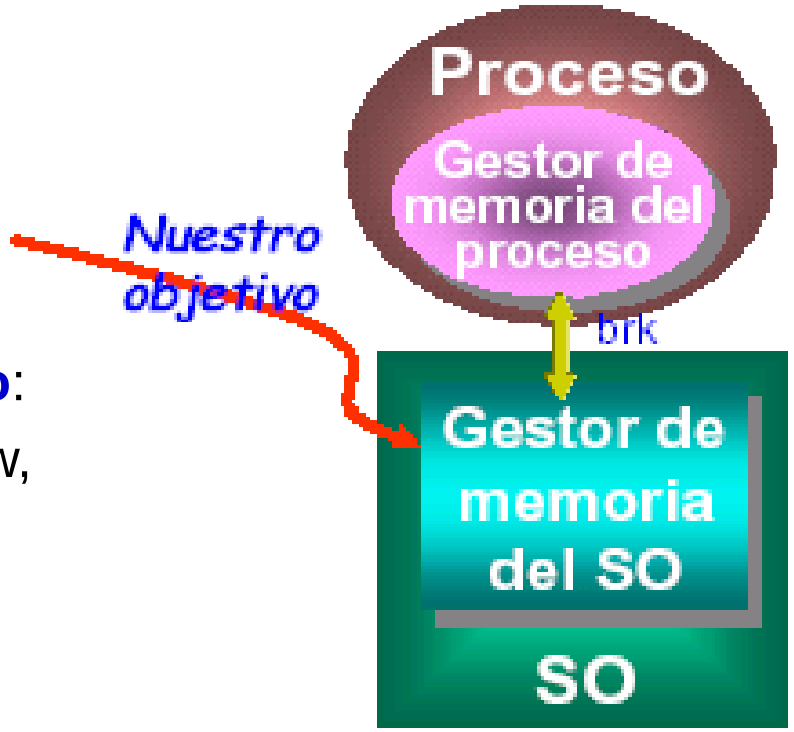


Introducción

- El trabajo del sistema operativo es abstraer esta jerarquía en un modelo útil y después administrarla: **Administrador de memoria**.
- Su labor:
  - Llevar el control de qué partes de la memoria están en uso y cuáles no lo están
  - Asignar memoria a los procesos cuando la necesiten y retirárselas cuando terminen
  - Administrar el intercambio entre la memoria central y el disco
  - Aislar el uso exclusivo de las celdas de memoria asignadas a un proceso
  - Permitir la compartición

Niveles de gestión de memoria

- Existen dos niveles:
  - **Gestor de memoria del SO**: asigna porciones de memoria al proceso.
  - **Gestor de memoria del proceso**: gestiona esta porciones (p.ej. new, delete).

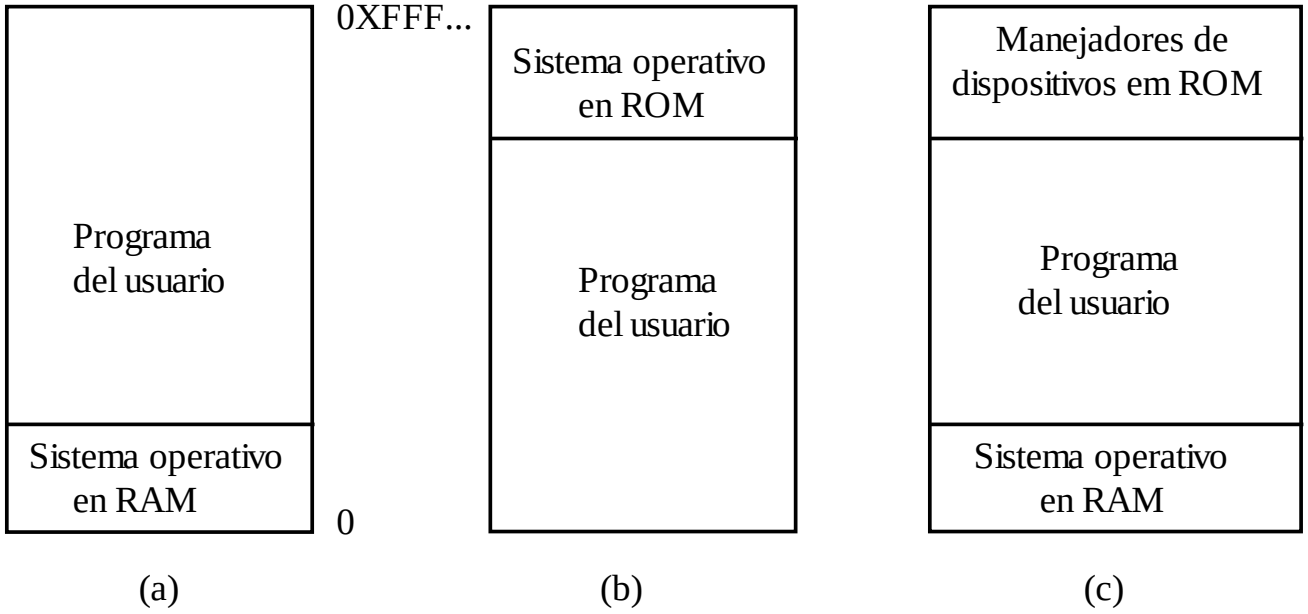


Gestión de memoria más simple

- Monoprogramación:
  - Un único proceso en memoria principal que puede acceder a toda la memoria física
    - No estaría el S.O
    - Obliga a que cada proceso gestione cada dispositivo de E/S
    - P.e. MOV REGISTRO1, 1000
  - Dejar sitio para el S.O.
    - Se ejecutan los procesos de 1 en 1.
    - El S.O carga un proceso, cuando termina carga otro encima.

Gestión de memoria más simple

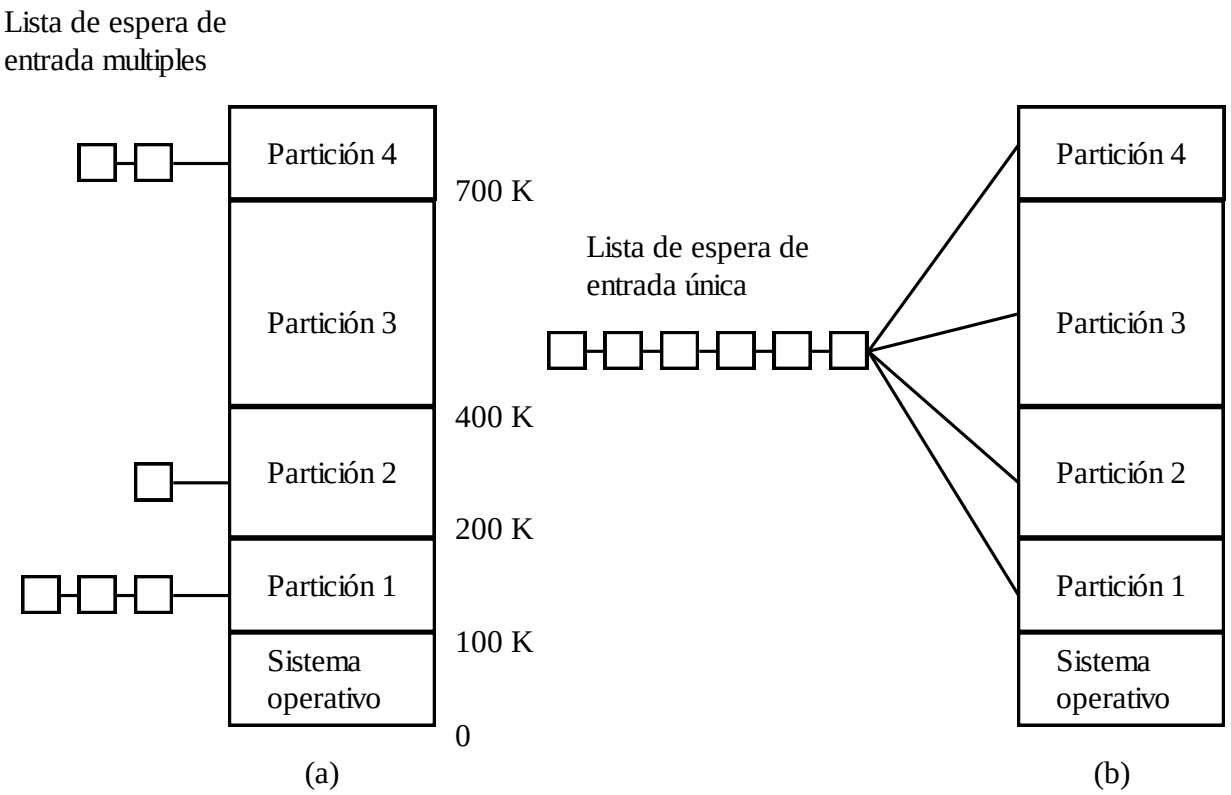
- Monoprogramación:



  - Utilización de la multiprogramación para aprovechar tiempos muertos de E/S

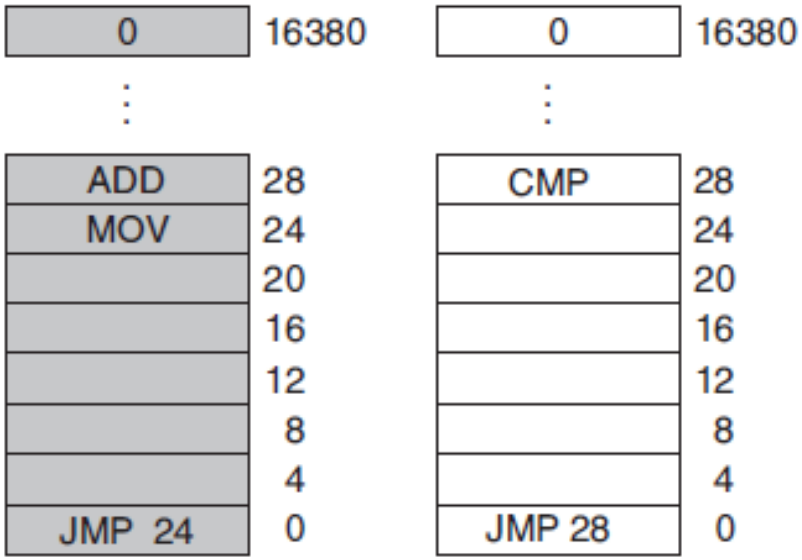
Multiprogramación con particiones fijas

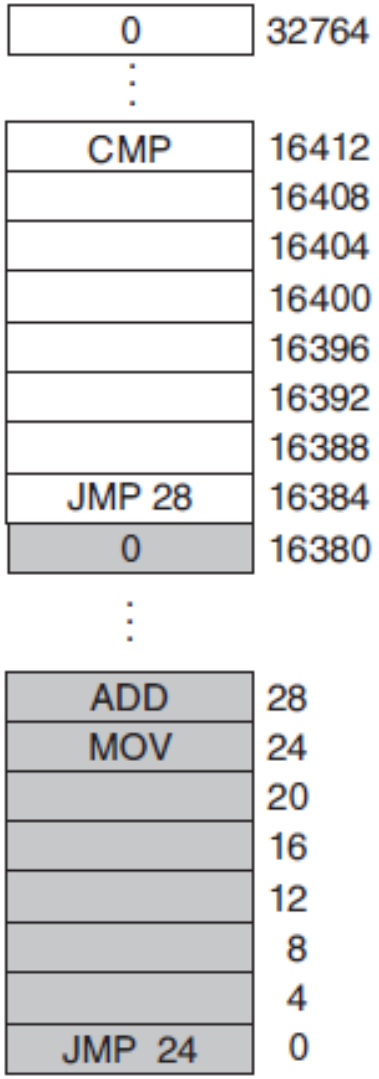
- Se divide la memoria en n particiones
  - El espacio que no se usa se desperdicia: ***fragmentación interna***



Una abstracción de memoria

- P.e. dos programas, cada uno con un tamaño de 16 KB.





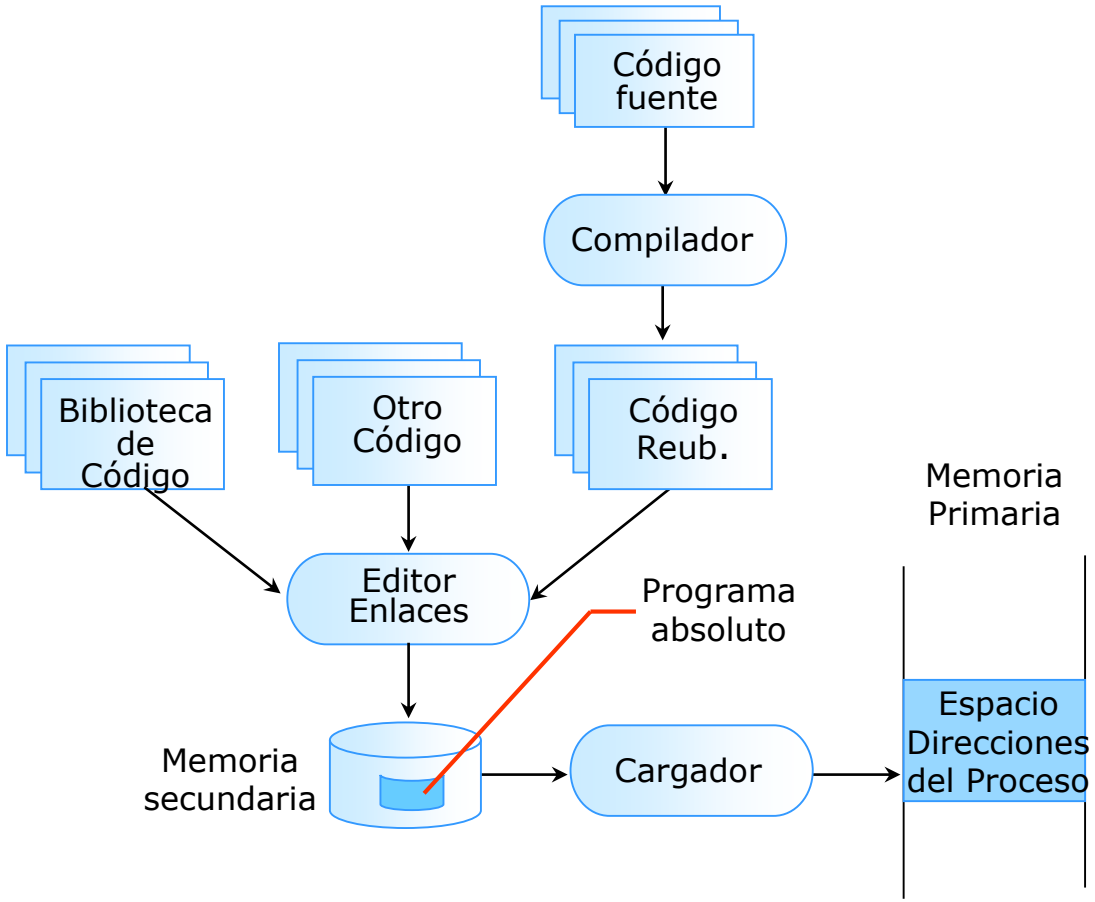
Una abstracción de memoria

- Exponer la memoria física a los procesos tiene varias desventajas.
  - si los programas de usuario pueden direccionar cada byte de memoria, pueden estropear el sistema operativo
  - con este modelo es difícil tener varios programas en ejecución a la vez

- **Espacio de direcciones**
  - Permitir que haya varias aplicaciones en memoria al mismo tiempo sin que interfieran entre sí:
    - **protección y reubicación**.
  - Cada proceso tiene su propio espacio de direcciones, independiente de los que pertenecen a otros procesos
    - Direcciones relativas (lógicas)
    - Direcciones físicas

Espacio de direcciones de un programa

- Conjunto de direcciones lógicas mapeadas sobre direcciones físicas de memoria primaria



Enlace estático

- **Tiempo de compilación** – el compilador:
  - Por cada programa genera código reubicable (modulo objeto reubicable)
    - Segmentos: texto (código), datos y pila
    - Si procedimiento invocado esta en otro módulo reubicable, se pospone hasta el tiempo de enlace.
      - Se referencia con dirección externa
- **Tiempo de enlace** – editor de enlaces:
  - Los segmento de código y de datos de los modulo objetos reubicables son combinados, forman programa absoluto (almacenado en disco) que empieza en la posición 0.
  - Las referencias externas se resuelven
- **Tiempo de carga** – el cargador:
  - Direcciones de los segmentos son ajustados de forma que hagan referencia a direcciones de memoria primaria asignada.

Enlace estático – tiempo de compilación

```
. . .
static int gVar;
. . .
int proc_a( int arg ) {
      . . .
     gVar = 7;
     pon_registro(gVar);
     . . .
}
```

Dirección relativa del segmento de código	Código generado	
0000	. . .	
. . .		
0008	entrada	proc_a
. . .		
0220	load	=7, R1
0224	store	R1, 0036
0228	push	0036
0232	call	‘pon_registro’
. . .		
0400	Tabla de referencias externas	
. . .		
0404	‘pon_registro’	0232
. . .		
0500	Tabla de definición externa	
. . .		
0600	(tabla de símbolos)	
. . .		
0799	(última posición del segmento de código)	

Dirección relativa del Segmento de Datos	Espacio de variables generado
. . .	
0036	[espacio para la variable gVar
. . .	
0049	(última posición del seg. datos)]

Enlace estático – tiempo de enlace

Dirección relativa del segmento de código	Código generado	
0000	. . .	
. . .		
1008	entrada	proc_a
. . .		
1220	load	=7, R1
1224	store	R1, 0136
1228	push	0136
1232	call	2334
. . .		
1399	(fin de proc_a)	
. . .	(otros módulos)	
2334	entrada	pon_registro
. . .		
2670	(tabla de símbolos)	
. . .		
2999	(última posición del segmento de código)	

Dirección relativa del Segmento de Datos	Espacio de variables generado
. . .	
0136	[espacio para la variable gVar
. . .	
1000	(última posición del seg. datod)]

Enlace estático – tiempo de carga

Dirección física	Código generado
0000	(Programas de otros procesos)
4000	(otros módulos)
. . .	
5008	entrada proc_a
. . .	
5220	load =7, R1
5224	store R1, 7136
5228	push 7136
5232	call 6334
. . .	
5399	(fin de proc_a)
. . .	(otros módulos)
6334	entrada pon_registro
. . .	
6670	(tabla de opcional de símbolos)
. . .	
6999	(última posición del segmento de código)
7000	(primera posición del segmento de datos)
. . .	
7136	[espacio para la variable gVar]
. . .	
8000	(Programas de otros procesos)

Enlace Dinámico

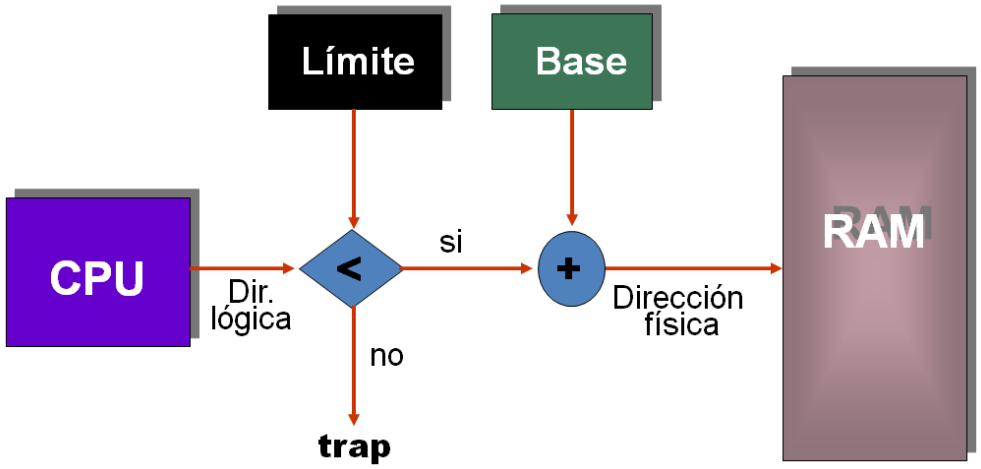
- ¿Al moverse los procesos en la compactación o en el intercambio?
  - Problema… el cargador debería relanzar el programa sobre las nuevas ubicaciones de memoria primaria
  - Solución… enlazar las direcciones absolutas del programa en tiempo de ejecución
    - La fase de carga de pospone hasta cuando se ejecute el proceso, en donde la reubicación se realiza cada vez que la CPU haga una referencia a la memoria.
    - Hardware de reubicación, intercepta cada una de las direcciones relativas y añade un *valor de reubicación* antes de enviarlo a la memoria primaria.

Reubicación y Protección

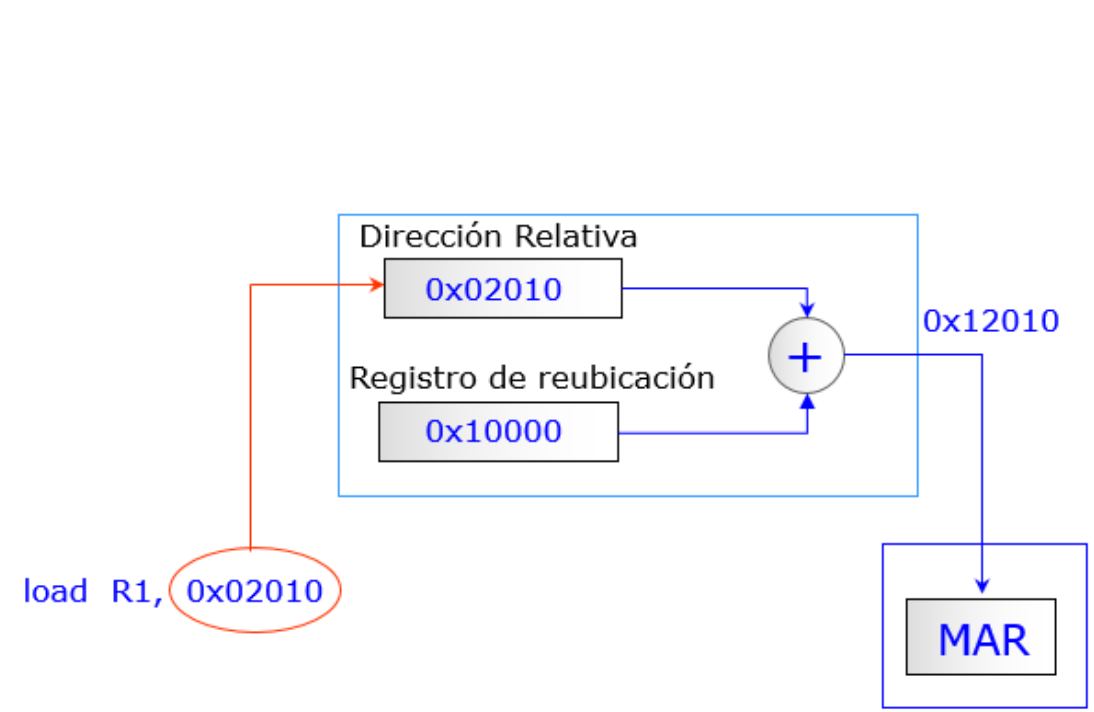
- Se usan direcciones relativas
- **Reubicación**: convertir direcciones relativas a absolutas una vez conocida la partición en la que se ha cargado
  - Se puede hacer en el momento de la carga del programa en memoria
    - El montador de enlace incluye en el fichero ejecutable información referente a las palabras dentro del programa que se han de reubicar
- **Protección**: evitar que cualquier proceso acceda a una partición que no es la suya

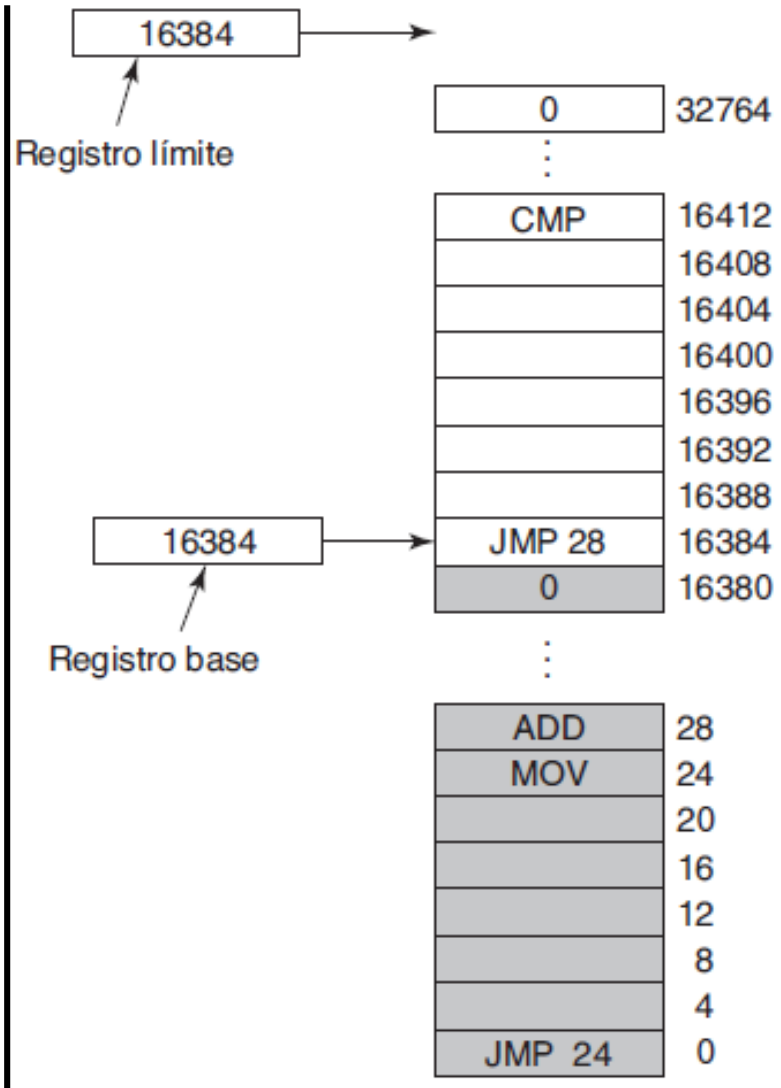
Una solución a la reubicación y protección

- 2 registros especiales
  - Registro de Base
  - Registro de Límite
- Al asignar el procesador a un proceso
  - Base: dirección de comienzo de la partición
  - Límite: tamaño de la partición
  - Dir. Absoluta= Dir. rel + Bas
  - Dir. rel < Límite
- Se puede mover un programa en memoria incluso después de empezar a ejecutarse



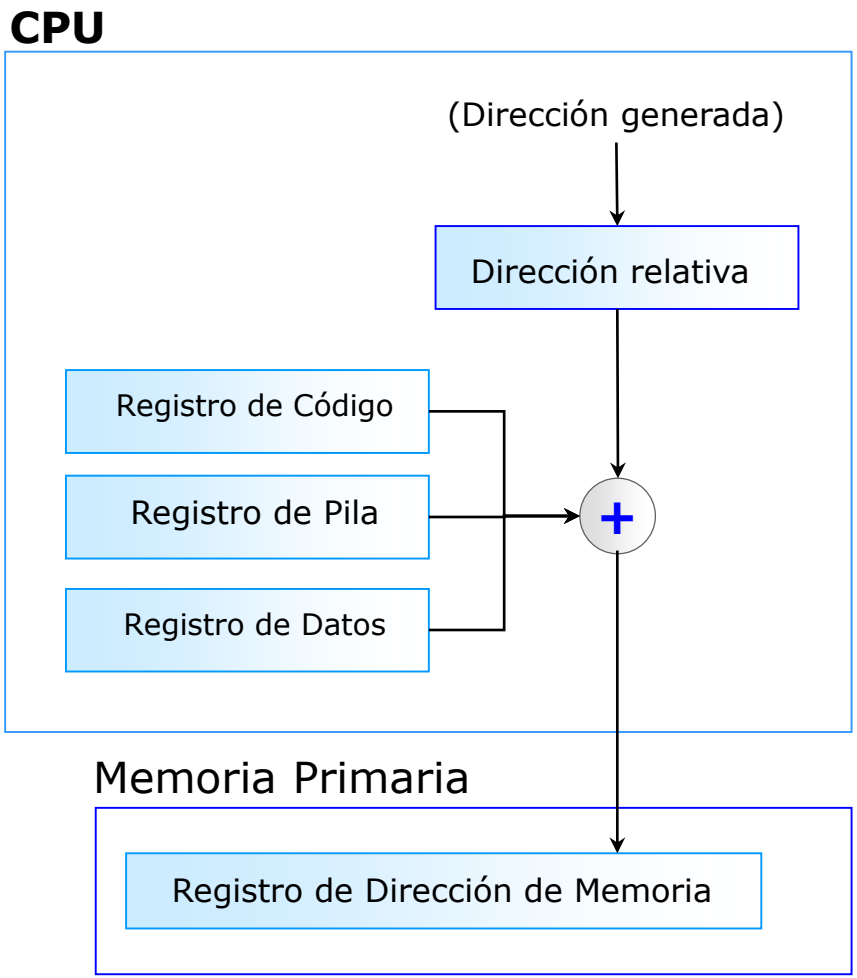
Una solución a la reubicación y protección





Registro de reubicación de múltiples segmentos

- Intel 8088



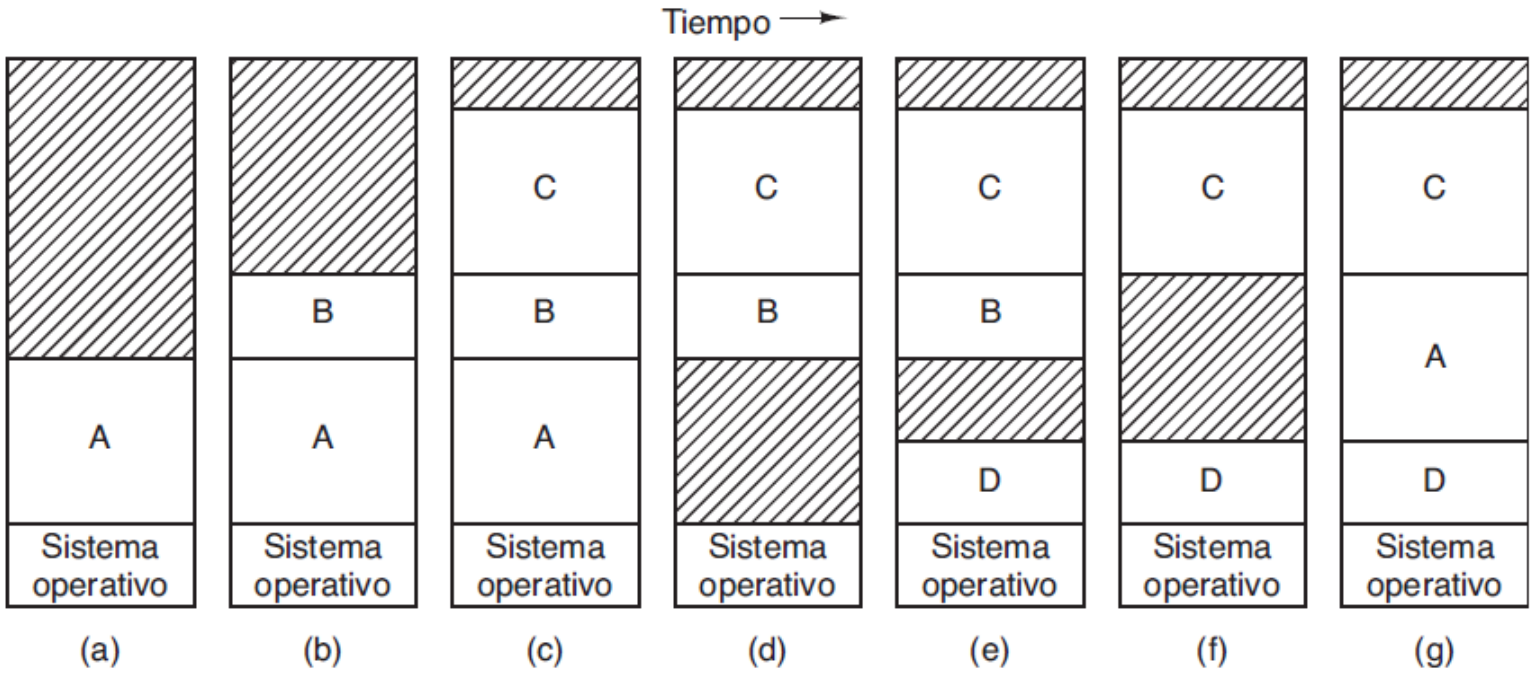
Intercambio

- La RAM que requieren todos los procesos es a menudo mucho mayor de lo que se tiene.

- Dos esquemas para resolverlo la sobrecarga de memoria
  - **Intercambio.** consiste en llevar cada proceso completo a memoria, ejecutarlo durante cierto tiempo y después regresarlo al disco
  - **Memoria virtual.** permite que los programas se ejecuten incluso cuando sólo se encuentran en forma parcial en la memoria

Intercambio: Multiprogramación con particiones variables

- Mejora la utilización de memoria
- El nº, la posición y el tamaño de las particiones varía dinámicamente al entrar y salir procesos de la memoria
- Complica la asignación y desocupación de la memoria
- Fragmentación externa



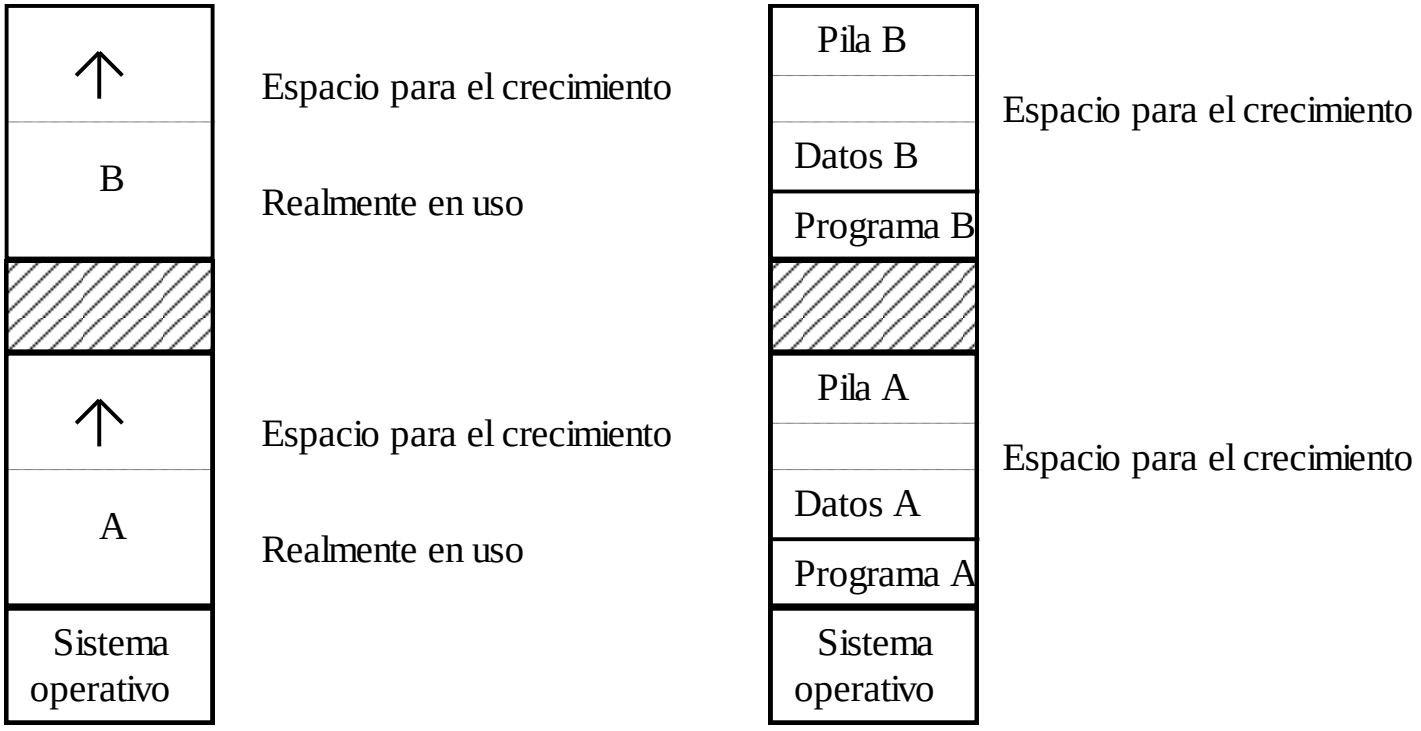
¿Qué cantidad de memoria asignar?

- Proceso de tamaño fijo: asignar tamaño

- Problema: cuando los procesos pueden crecer pidiendo reserva dinámica
  - Si hay espacio (hueco) junto al proceso utilizar este espacio
  - En caso contrario:
    - mover el proceso a un hueco mayor
    - llevar algunos procesos a disco

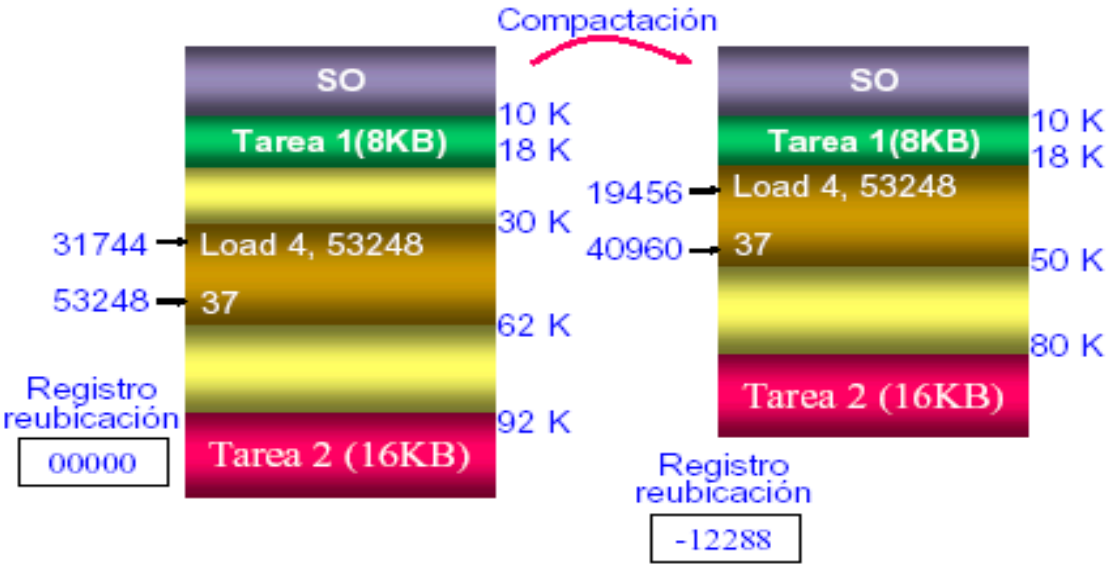
Asignación de memoria para poder crecer

- Si se espera que la mayoría crezca:
  - Se les asigna algo más para reducir la probabilidad de que no quepan en la partición asignada



Compactación

- Técnica para reducir la fragmentación externa que consiste en arrastrar los contenido de memoria a un lugar para reunir la memoria libre en un bloque.
- Posible en sistemas con reubicación dinámica (se realiza en tiempo de ejecución).
- Costoso



Administración de memoria libre

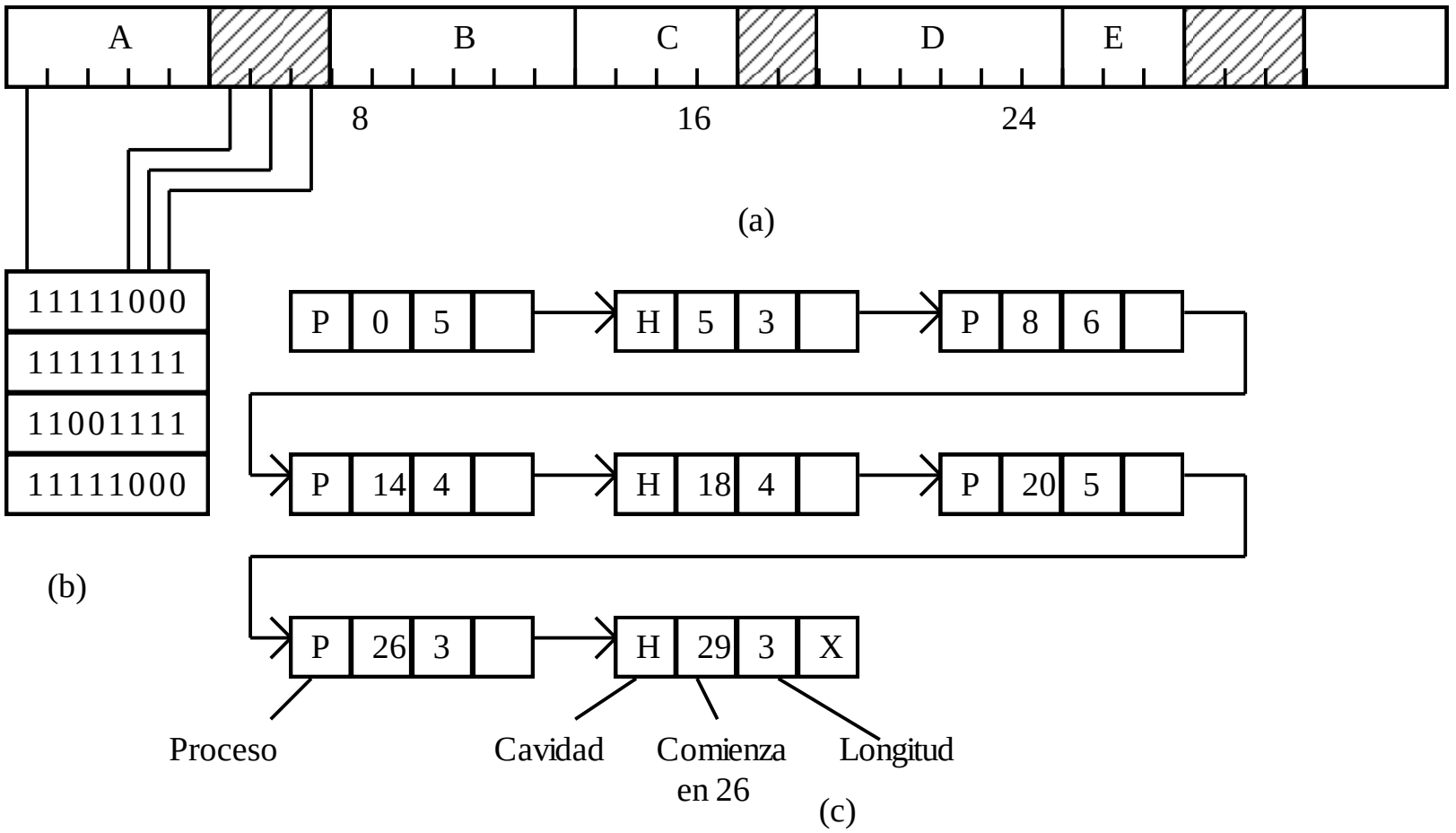
Mapa de bits

- La memoria se divide en unidades de asignación (cada una está representada por un bit en el mapa)
  - 0-libre, 1-asignada
- Cuanto más pequeña sea la unidad, mayor será el mapa de bits

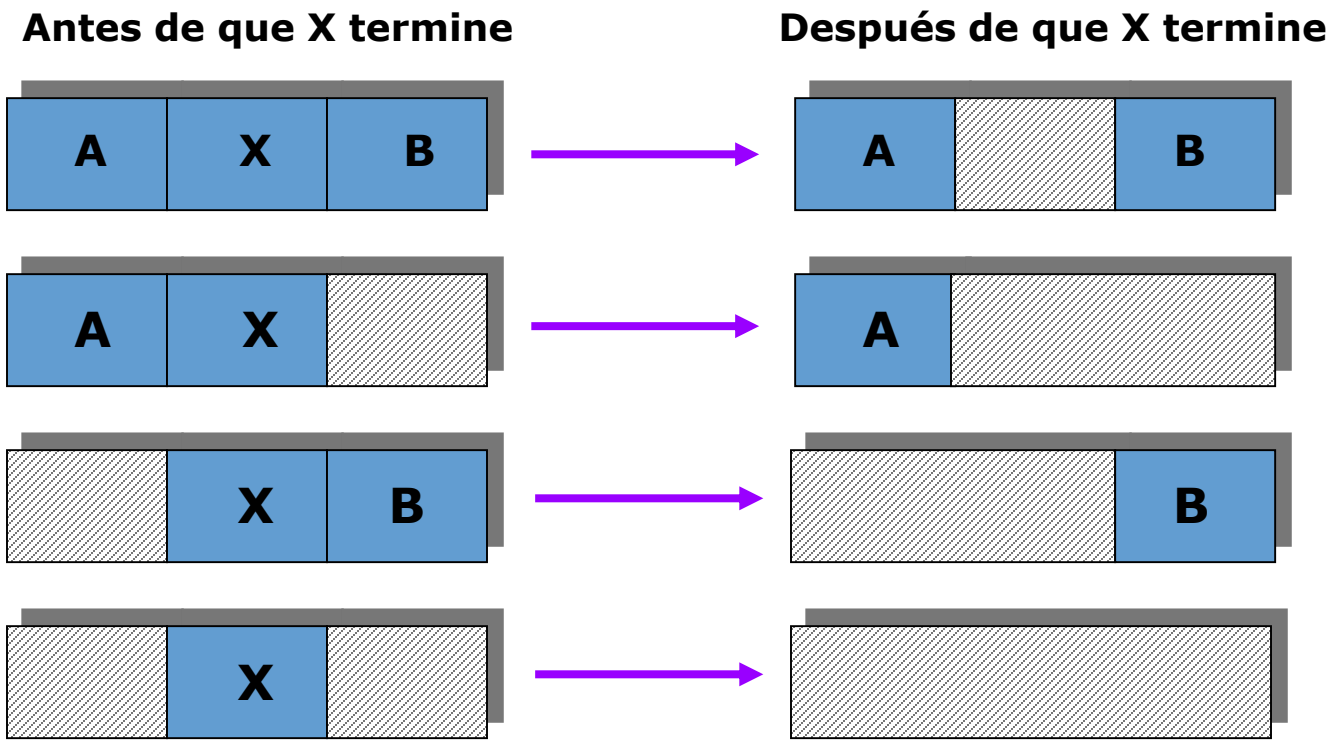
Listas enlazadas

- Mantiene una lista ligada de segmentos de memoria asignados y libres:
  - Hueco (H) o Proceso (P)
  - Dirección inicial
  - Longitud
  - Apuntador a la siguiente entrada
- Suele mantenerse ordenada por la dirección
  - Fácil actualizar cuando un proceso termina o es intercambiado

Administración de memoria libre



Administración con listas enlazadas



Administración con listas enlazadas

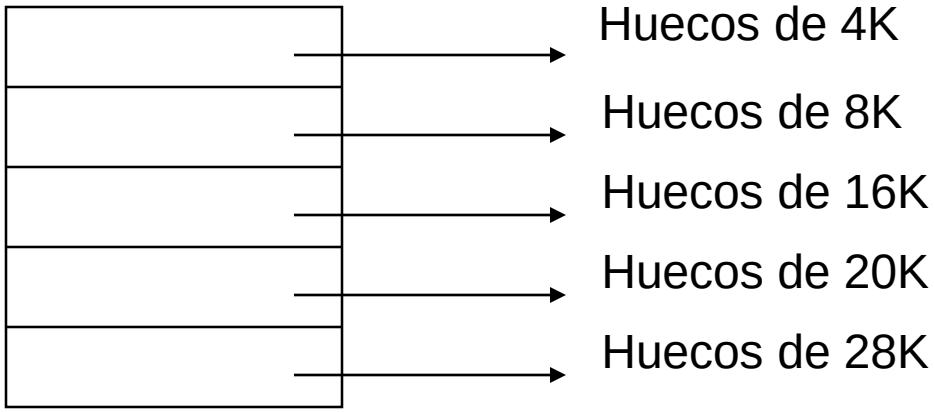
- **Algoritmos de asignación**
  - **Primer ajuste**: el administrador rastrea la lista hasta hallar una cavidad que sea lo suficientemente grande.
  - **Siguiente ajuste**: igual que el primer ajuste pero empezando en donde se quedó la vez anterior.
  - **El que mejor ajusta**: busca en toda la lista hasta encontrar la cavidad que ajusta mejor.
  - **El que peor ajusta**: toma siempre la cavidad más grande disponible.

Administración con listas enlazadas

- Para optimizar:
  - Listas separadas de huecos y procesos
    - La desocupación se vuelve más lenta
  - Ordenar por tamaños de huecos
    - Mejora el primer ajuste y el mejor ajuste

Administración con listas enlazadas

- **Quick Fit (ajuste rápido)**
  - Mantiene listas separadas de bloques libres para los tamaños más usuales:
    - Listas para agujeros de: 4K, 8K, 12K, etc., más una lista para agujeros con tamaños poco comunes.
    - Se busca en una de las listas.
    - Hay sobrecarga en la liberación de un bloque (buscar a los vecinos para compactar).



Administración por el método de los colegas

- El gestor de memoria mantiene una serie de listas de bloques libres, una para los de tamaño 1 byte, otra para 2, otra para 4, 8, 16, etc. hasta llegar a n, siendo n el tamaño total del área de memoria libre inicial.
  - Fragmentación interna

- Optimiza la fusión de huecos
  - En la liberación de k bytes, el gestor de memoria solo busca en la lista de huecos de longitud k.

Administración por el método de los colegas