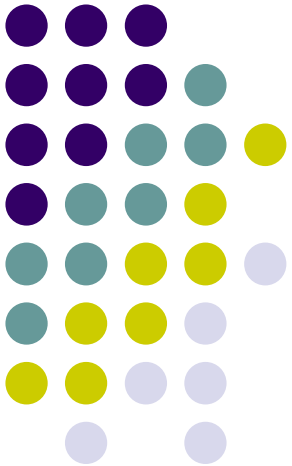
# Tema 3

Administración de memoria



# Índice

1. Introducción
2. Fases de la carga de un programa
3. Características de los gestores de memoria
4. Modelos básicos de traducción de direcciones
5. Modelos avanzados de traducción de direcciones
6. Memoria virtual
7. Casos de estudio

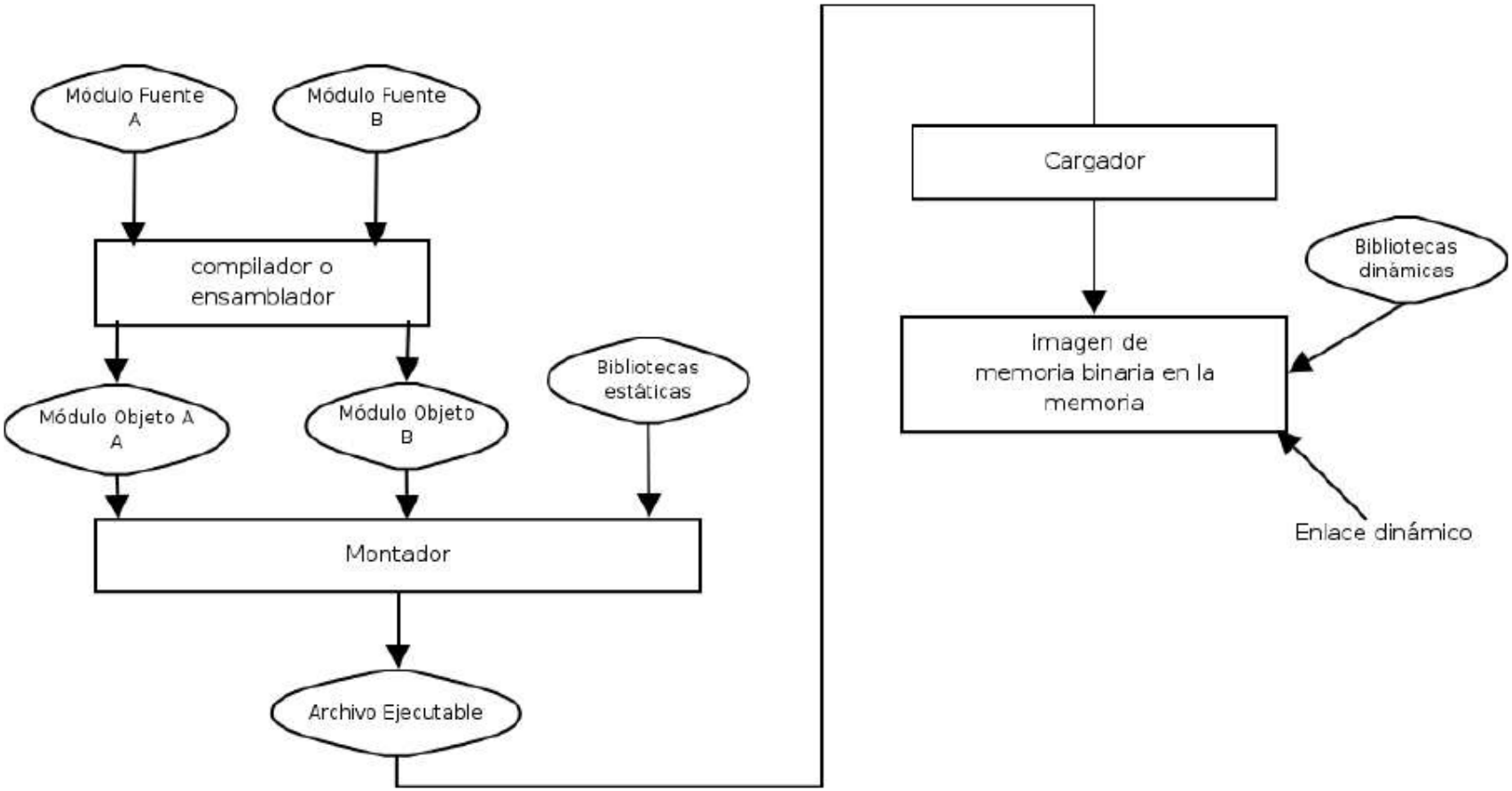
# Índice

# 1. Introducción

- La memoria es uno de los recursos más importantes en un sistema de multiprogramación.
- Ningún programa puede ejecutarse en un ordenador sin que, previamente, haya sido cargado en la memoria principal (MP).
- Para poder ejecutar más de un proceso a la vez necesitaremos mantener dichos procesos simultáneamente en memoria → deberán compartirla.
- La tarea de subdividir la memoria la lleva a cabo dinámicamente el SO y se conoce como gestión de memoria.

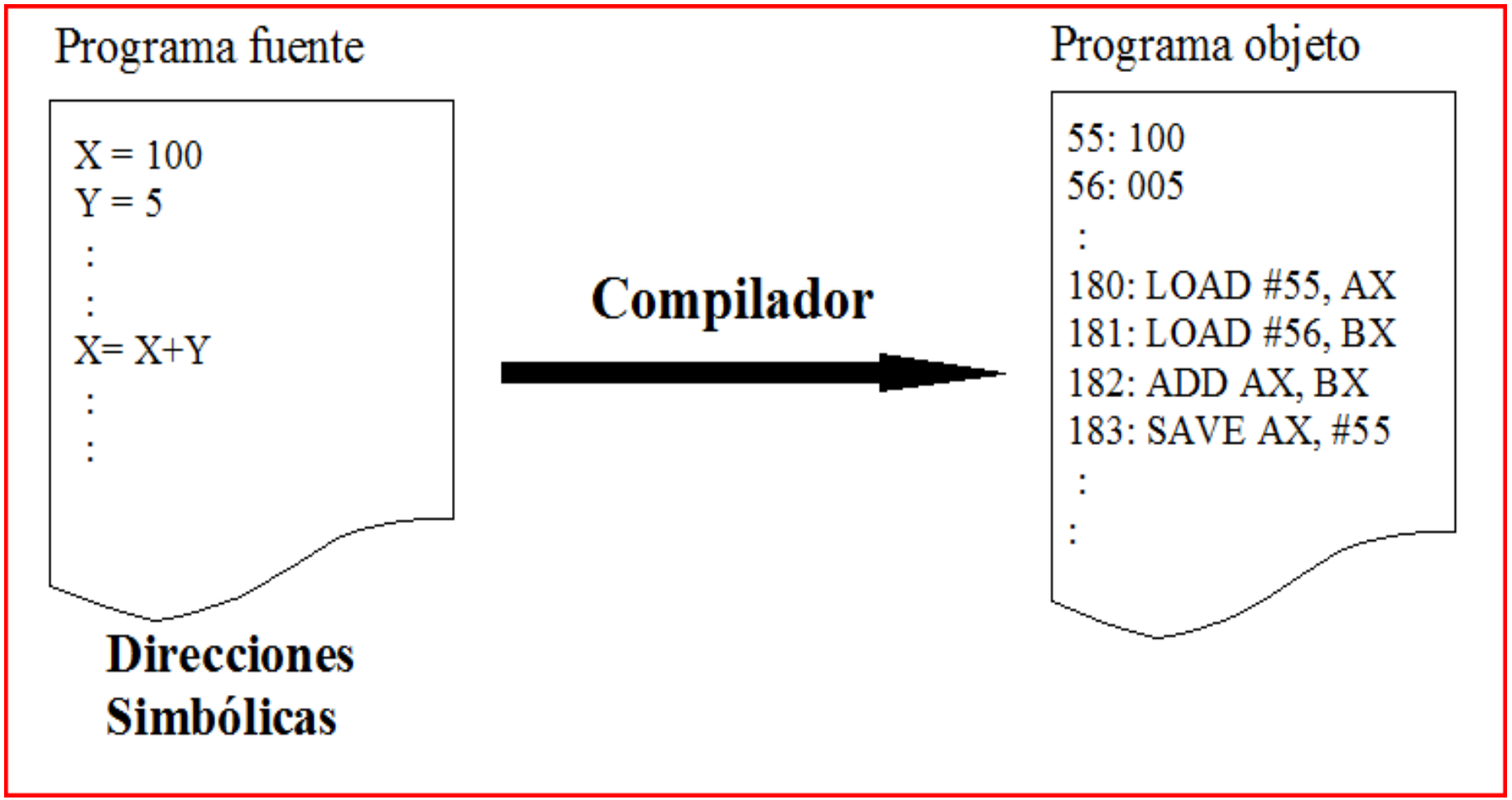
# Índice

# 2. Fases de la carga de un programa



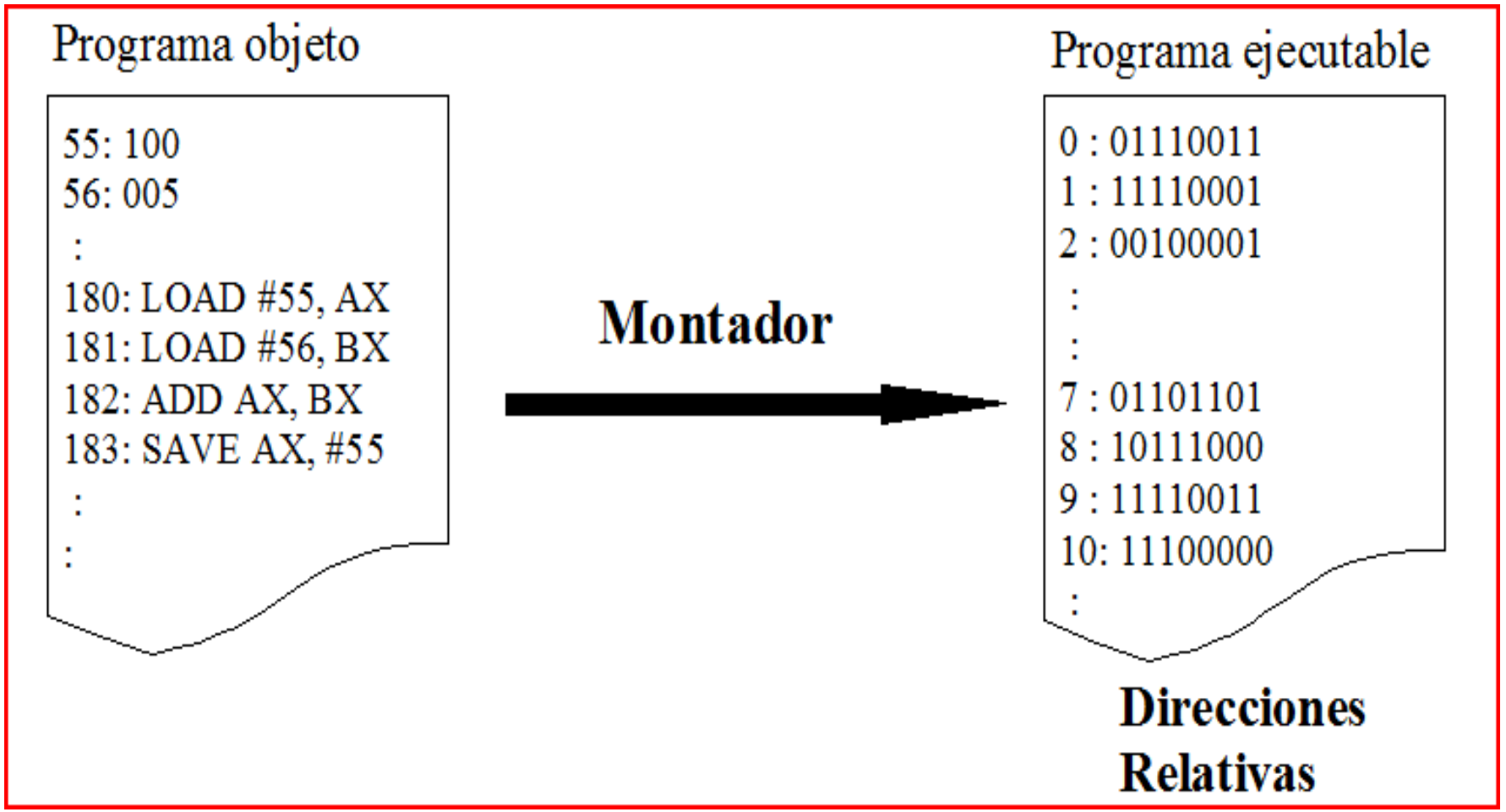
# 2. Fases de la carga de un programa

## Compilación



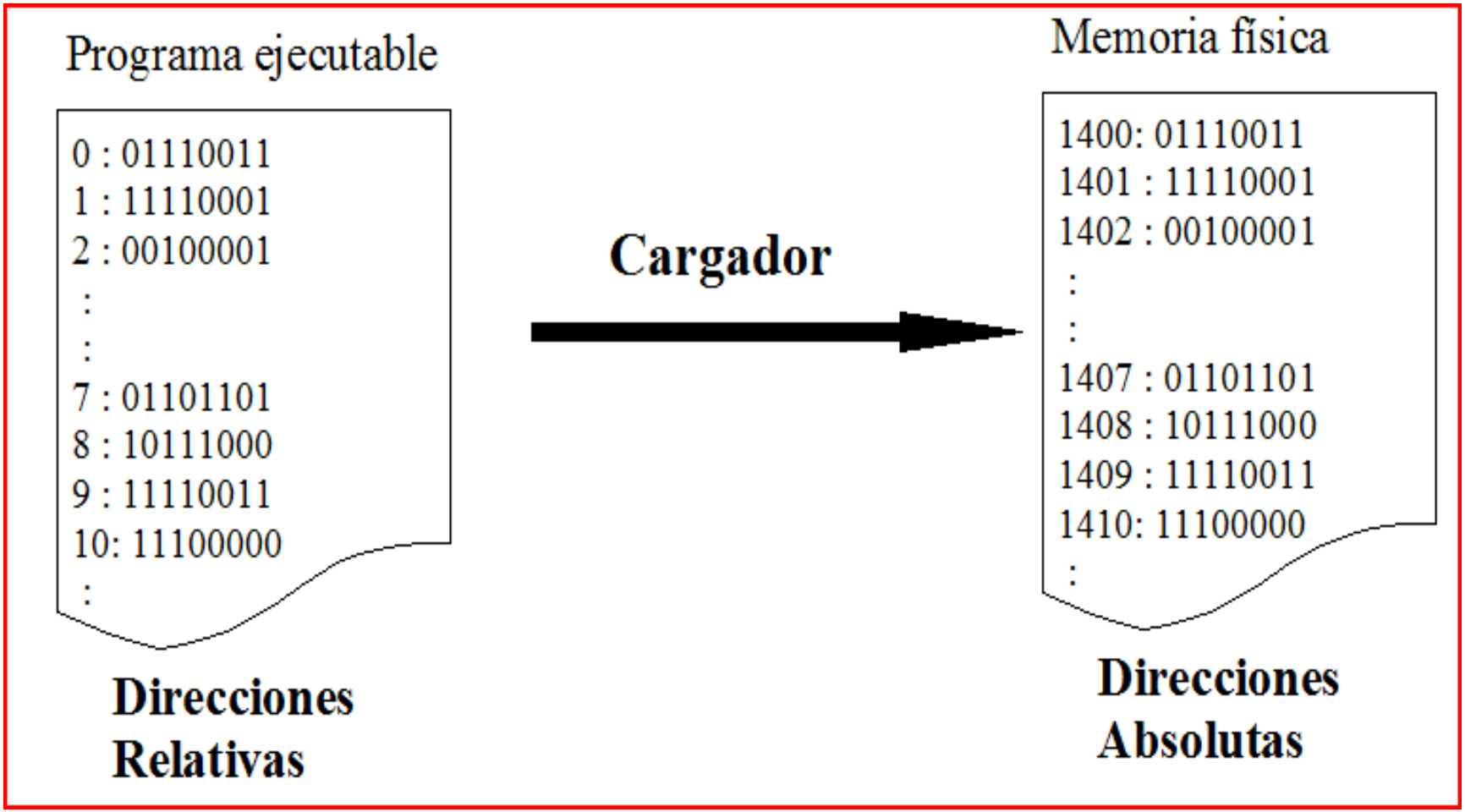
# 2. Fases de la carga de un programa

## Montaje o enlace



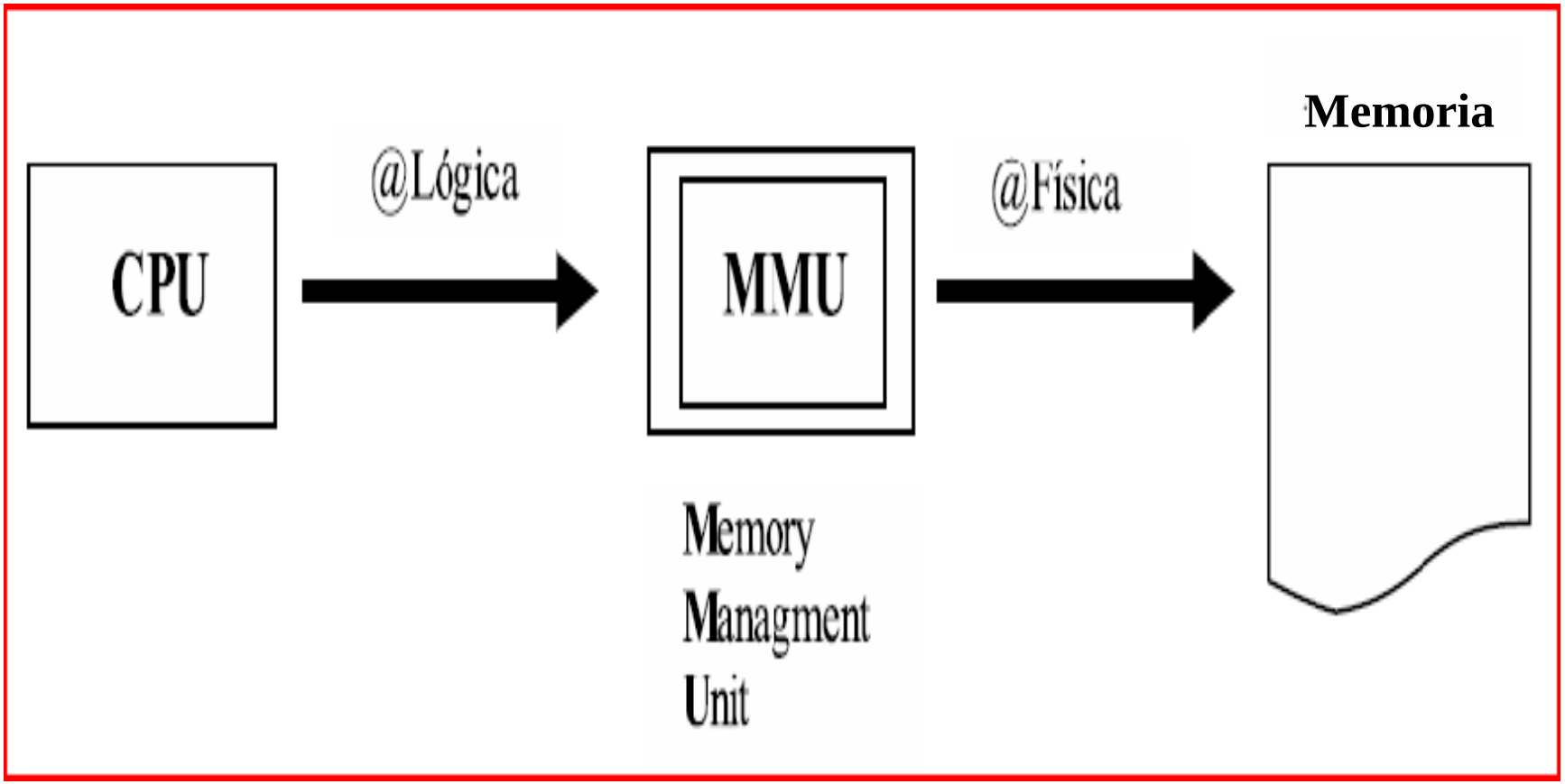
# 2. Fases de la carga de un programa

## Carga



# 2. Fases de la carga de un programa

## Ejecución



# 2. Fases de la carga de un programa

- Para un programa se suele hablar de:
  - **Espacio lógico del programa**: conjunto de direcciones lógicas válidas para un determinado programa ejecutable.
  - **Espacio lógico del procesador**: conjunto de direcciones lógicas válidas que puede lanzar un procesador.
  - **Espacio físico del programa**: conjunto de direcciones físicas válidas para el programa.

# Índice

# 3. Características de los gestores de memoria

- **Monoprogramados/Multiprogramados**: se permite sólo un proceso a la vez cargado en memoria o más de uno.
- **Residentes/No residentes**: el proceso debe estar en la memoria durante toda su ejecución o se permite al proceso salir y volver a cargarse.
- **Inmóvil/Móvil**: el proceso no puede cambiar de lugar en la memoria durante su ejecución o puede cargarse en otra zona de memoria y continuar ejecutándose.
- **Contiguo/No contiguo**: el proceso ha de estar cargado en posiciones consecutivas de memoria o no es necesario.
- **Entero/No entero**: el proceso ha de estar cargado completo en la memoria o no es imprescindible.

# 3. Características de los gestores de memoria

- Con PARTICIONES FIJAS y PARTICIONES VARIABLES se añadió la multiprogramación.
- Con el SWAPPING (Intercambios), se incorporaba la característica de NO residencia en memoria a lo largo de la ejecución.
- Con la REUBICACIÓN DINÁMICA, se evitaba la inmovilidad de los mismos.
- Los modelos de PAGINACIÓN y SEGMENTACIÓN conseguían que el código de los procesos no tuvieran que situarse en zonas contiguas de la memoria.
- Los modelos más modernos han incorporado una nueva característica, el proceso ya no tiene que estar entero en la memoria para poder ejecutarse. Este último modelo, que incorpora, por tanto, las características de: multiprogramado, no residente, móvil, no contiguo y no entero es al que se le conoce con el nombre de

  **MEMORIA VIRTUAL**

# Índice

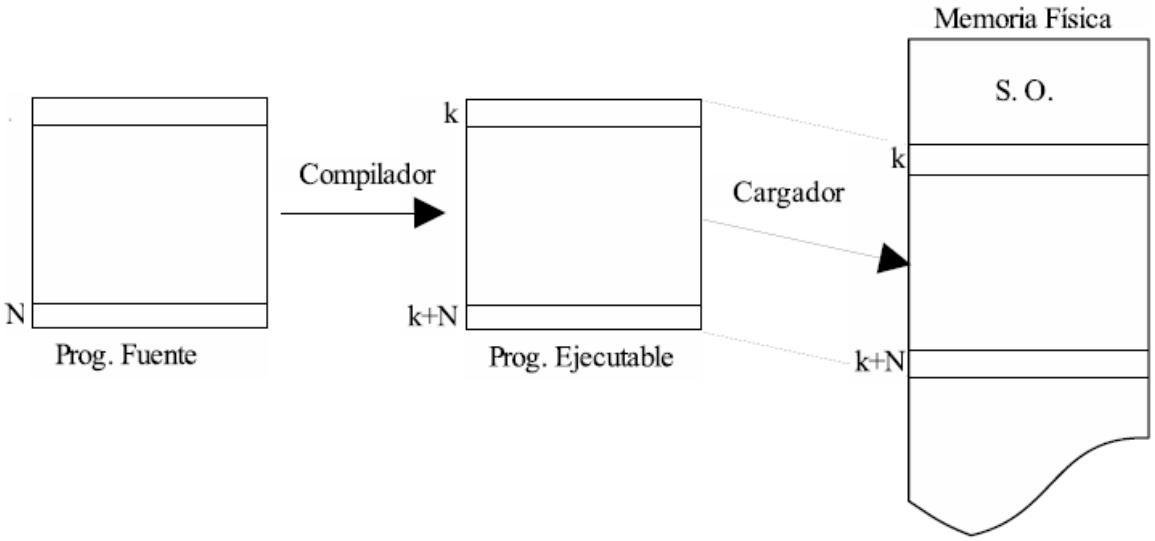
# 4. Modelos básicos de traducción de direcciones


- Los modelos que veremos son:
  - Modelo 0.
  - Modelo 1 (Reubicación estática).
  - Modelo 2 (Particiones).
  - Modelo 3 (Swapping).
  - Modelo 4 (Reubicación dinámica).

# 4. Modelos básicos de traducción de direcciones

## 4.1. Modelo 0

- Características: monoprogramado, residente, inmóvil, contiguo y entero.
- La **traducción de direcciones** lógicas a físicas se realiza **durante la generación del ejecutable** → el ejecutable contiene las direcciones físicas.



# 4. Modelos básicos de traducción de direcciones


## 4.1. Modelo 0

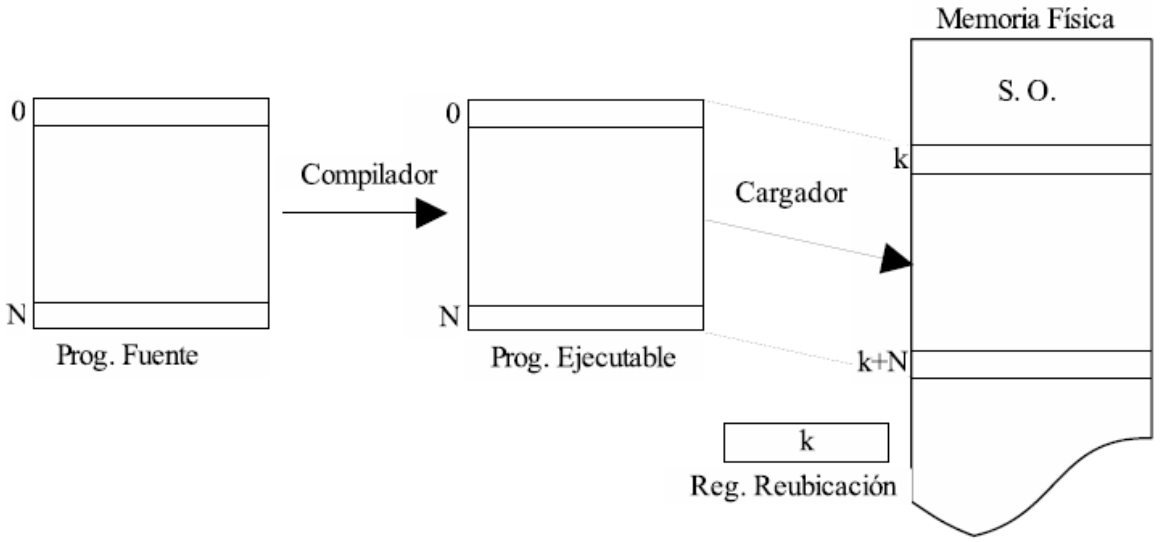
- Problemas:
  - Si crece el SO hay que volver a compilar todos los programas que existen en la máquina.

- Soluciones:
  - Cargar el SO en la parte más alta de la memoria y los procesos en la parte baja.
  - Usar el modelo siguiente.

# 4. Modelos básicos de traducción de direcciones

## 4.2. Modelo 1 (Reubicación estática)

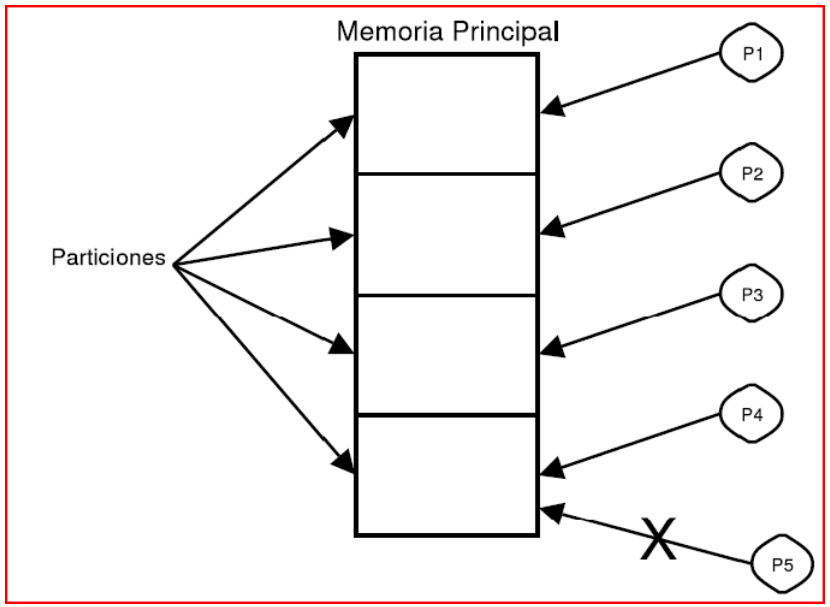
- Las mismas características que el Modelo 0 pero ahora la **traducción de direcciones** lógicas a físicas se realiza **durante la carga del proceso en memoria**.
- Disponemos de un registro (Registro de reubicación) que nos dice la posición de memoria donde tenemos que cargar el proceso:



# 4. Modelos básicos de traducción de direcciones

## 4.3. Modelo 2 (Particiones)

- Las mismas características que en Modelo 1 pero con multiprogramación.
- La **traducción de direcciones durante la carga** del proceso en memoria.
- Al tener más de un proceso en memoria:
  - Debe cuidar la protección entre procesos.
  - Se debe organizar el espacio lógico de cada proceso, **divide la memoria disponible para los procesos en particiones en cada una de las cuales sólo puede residir un proceso** (influye en el grado de multiprogramación):




- Particiones Fijas (Multiprogramming with Fixed Tasks).
- Particiones Variables (Multiprogramming with Variable Tasks).

# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

Con particiones de **igual tamaño:**

- El SO necesita almacenar la siguiente información:

| Nº Part | @Base | Estado |
|---|---|---|
| 1 | | |
| 2 | | |
| : | | |
| N | | |

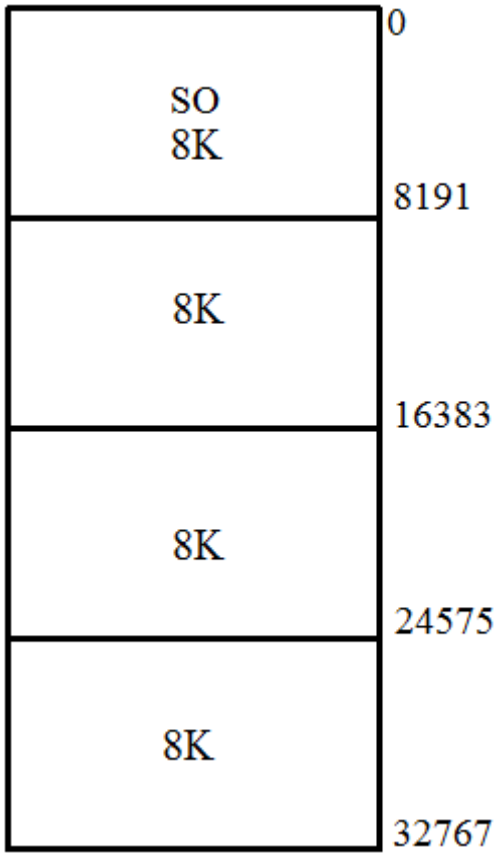
**Nº Part** es el número de partición.
**@Base** es la dirección física donde empieza la partición.
**Estado** de la partición (libre u ocupado).

# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

Ejemplo

| | @Base | Estado |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 8192 | 1 |
| 2 | 16384 | 1 |
| 3 | 24576 | 1 |



# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

Con particiones de **tamaño diferente:**

- El SO necesita almacenar la siguiente información:

| Nº Part | @Base | Estado | **Tamaño** |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| : | | | |
| N | | | |

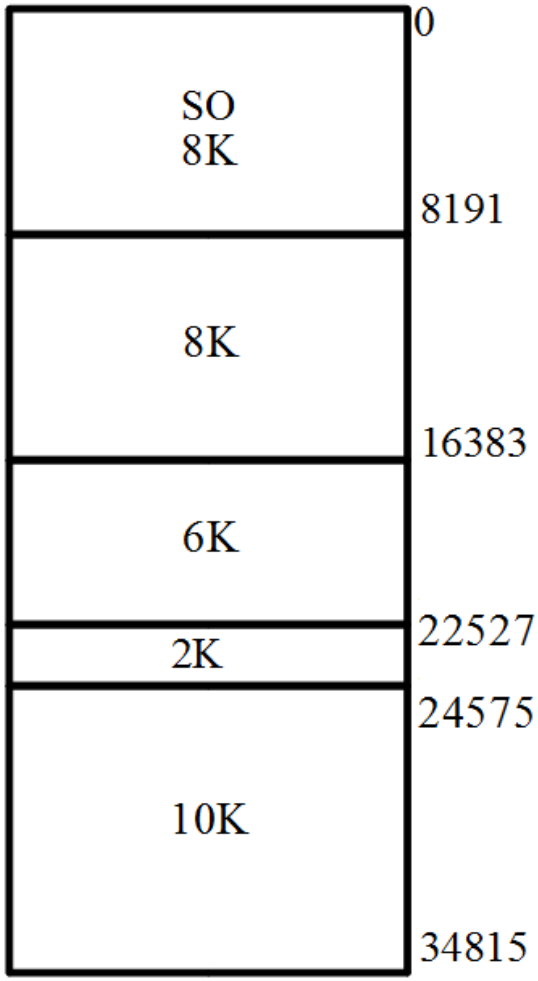
**Nº Part** es el número de partición.
**@Base** es la dirección física donde empieza la partición.
**Estado** de la partición (libre u ocupado).
**Tamaño** es el tamaño de cada partición

# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

- Ejemplo

| | @Base | Estado | Tamaño |
|---|---|---|---|
| 0 | 0 | 1 | 8192 |
| 1 | 8192 | 1 | 8192 |
| 2 | 16384 | 1 | 6144 |
| 3 | 22528 | 1 | 2048 |
| 4 | 24576 | 1 | 10240 |



# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

Algoritmo de asignación de particiones:

Con particiones de **igual tamaño:**

- Buscamos cualquier partición que esté libre y la asignamos al proceso.
- Si todas están ocupadas hay que buscarle sitio sacando uno de los procesos que actualmente están en memoria.

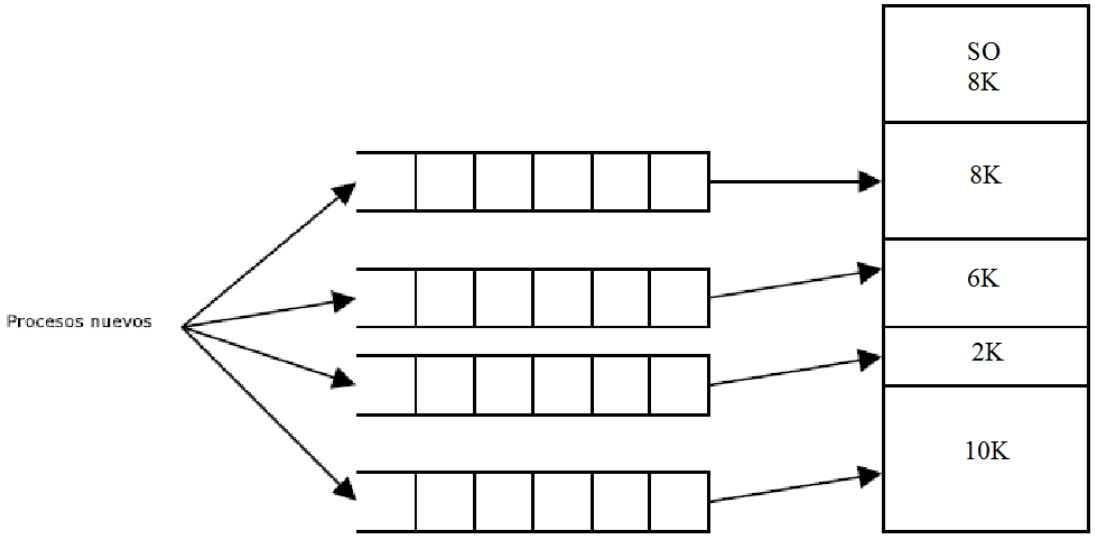
# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

Algoritmo de asignación de particiones:

Con particiones de **tamaño diferente: existen 2 métodos**

- **Una cola por partición:** Asignarle la **partición más pequeña en la que quepa** (suponiendo que conocemos su tamaño máximo):

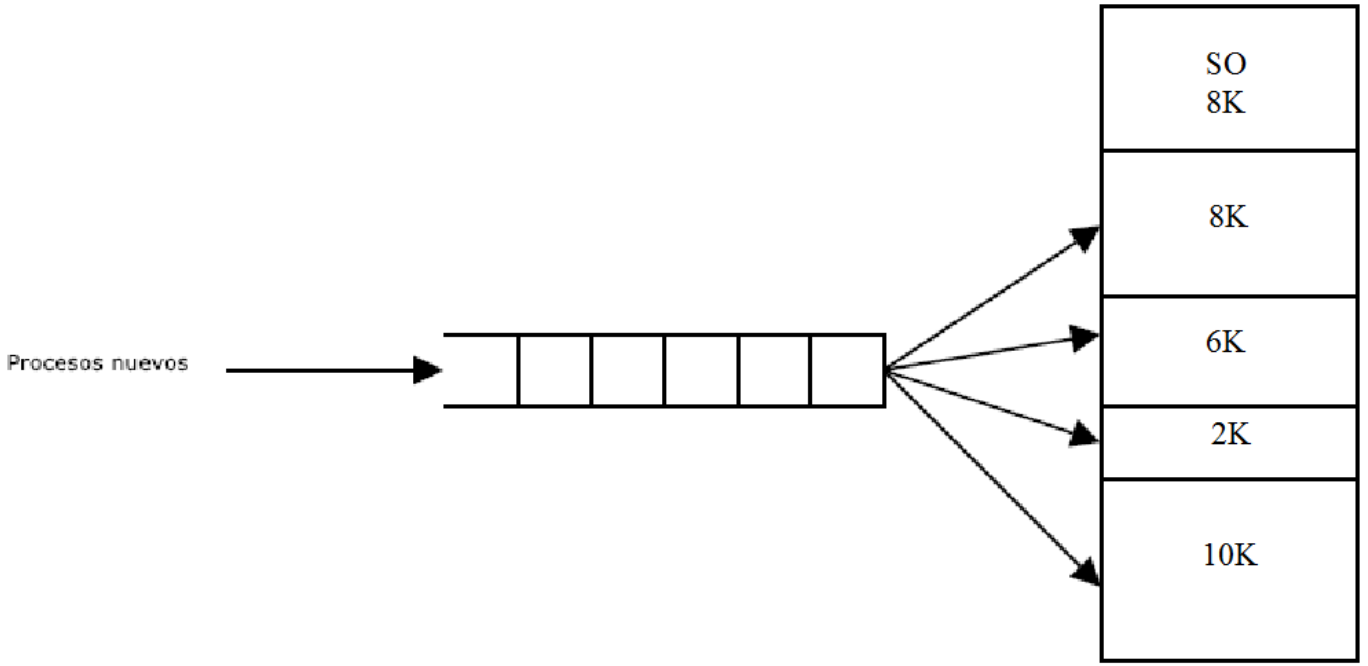



- Puede suceder que tengamos particiones muy grandes sin usar.

# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

- **Una sóla cola para todas las particiones:** Asignarle la partición más pequeña disponible en la que quepa:



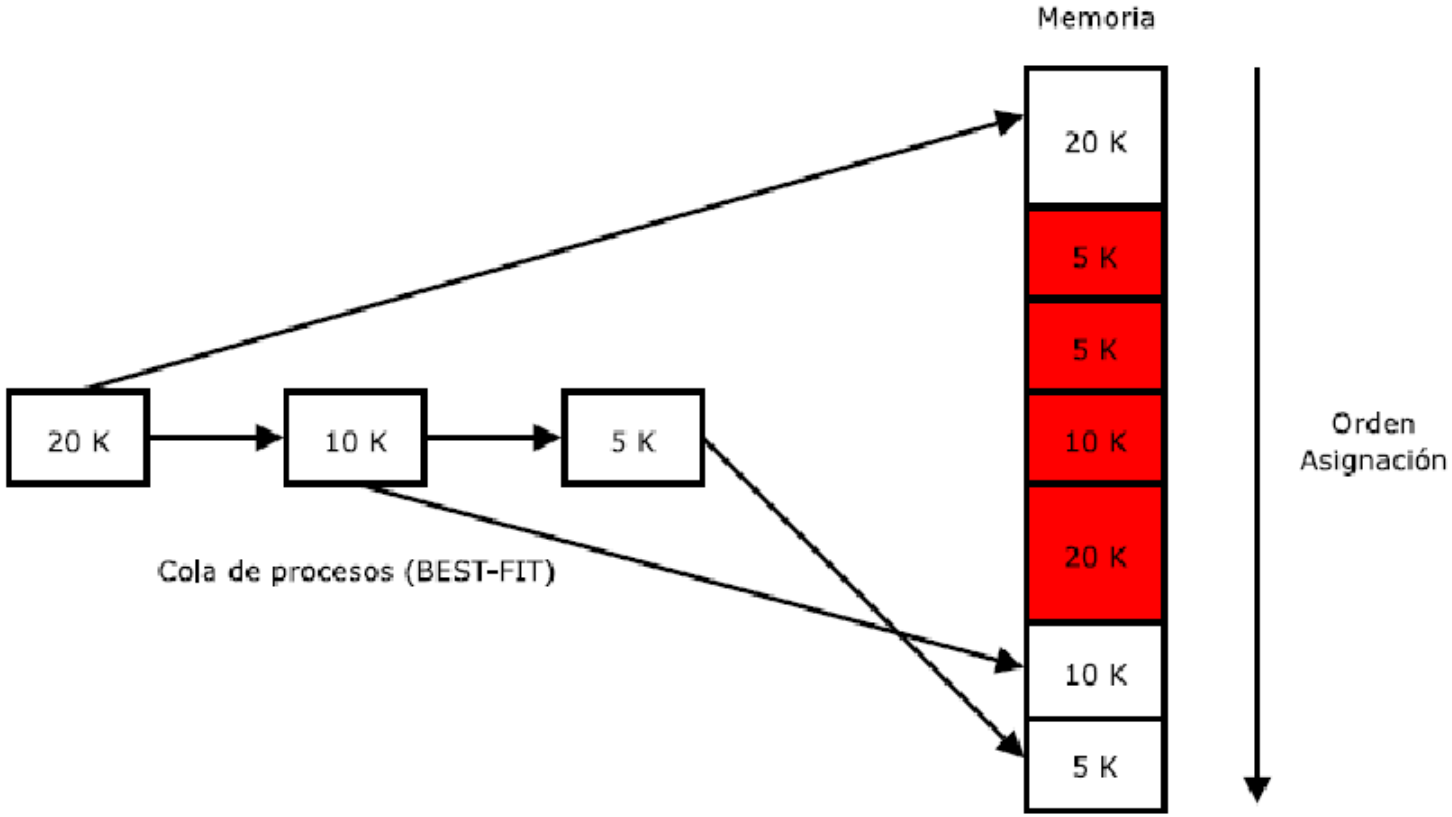

¿Qué proceso pasa a MP?

- **First Fit:** el primer proceso que quepa en la partición que quede libre.
- **Best Fit:** el proceso que mejor se adapte al tamaño de la partición que quede libre.

# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

- **En particiones fijas las particiones se asocian a procesos y no al revés, para cada partición buscamos entre los procesos en cola:**



# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

**Ventajas**

- El uso de particiones de distinto tamaño proporciona cierto grado de flexibilidad a las particiones estáticas.
- Ambas son fáciles de implementar y exigen un software del SO y una sobrecarga mínima.

**Inconvenientes**

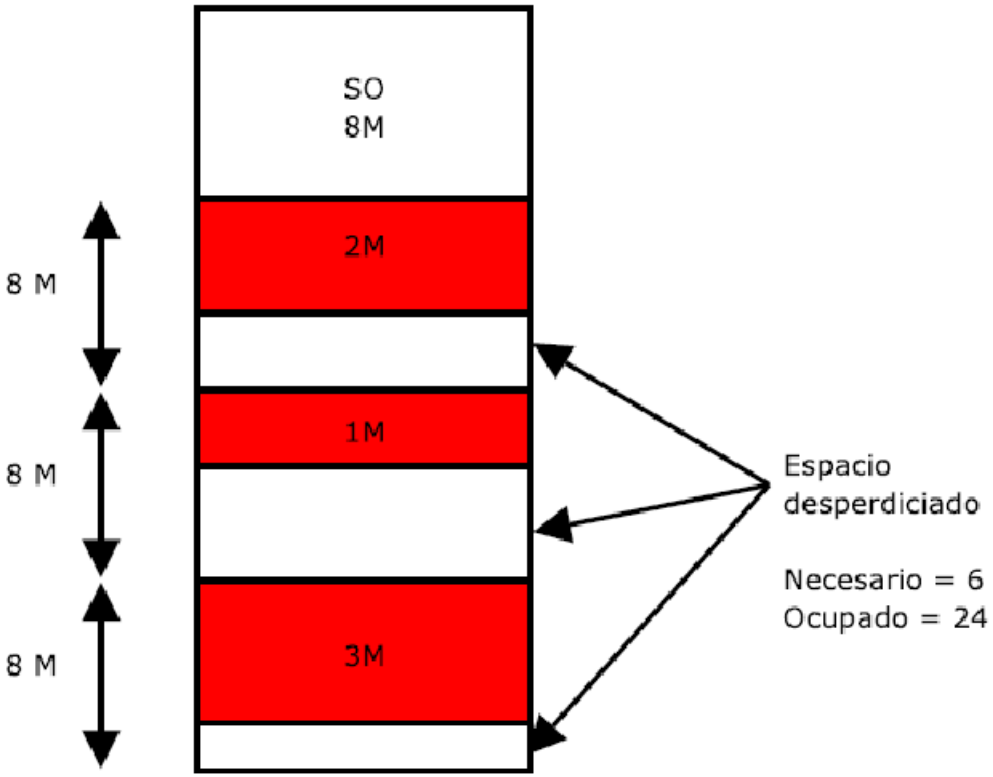
- El número de particiones y su tamaño se especifica en el momento de la generación del sistema. Limita el número de procesos activos (grado de multiprogramación).
- Los trabajos pequeños no hacen uso eficiente del espacio de las particiones (no sabemos de antemano el tamaño de los procesos). Cualquier programa por pequeño que sea, siempre ocupa una partición completa.

# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

### Fragmentación interna

- Al problema de malgastar espacio interno en cada partición se le denomina fragmentación interna.



# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

**Fragmentación externa**

- Cuando hay memoria suficiente para atender la solicitud de un proceso pero no se le puede conceder porque no es contiguo o porque el administrador de memoria utilizado no lo permite.

¿Cómo calculamos esta fragmentación?

- Suponemos que los distintos huecos libres en memoria forman una partición (**no** contamos los huecos dentro de una partición, es decir, la fragmentación interna no se contabiliza nunca como externa).
- Empezamos a asignar los **procesos pendientes** a dicha partición hasta que ya no nos quede más espacio.
- La fragmentación externa será igual a la suma de los tamaños de los procesos contenidos en esta partición.

# 4. Modelos básicos de traducción de direcciones

## 4.3.1 Particiones Fijas (MFT)

### Ejemplo:

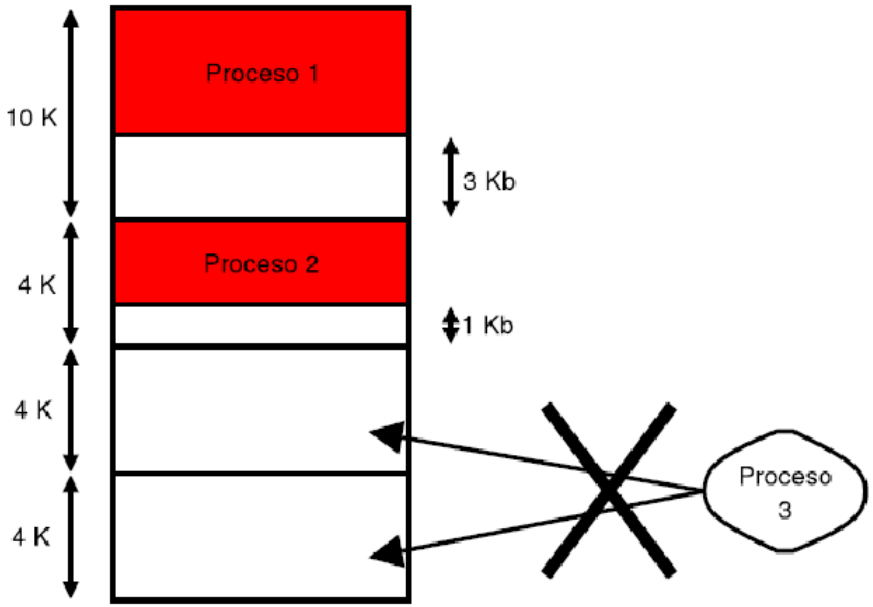
- Disponemos de un sistema con MP de 22 K que está dividida según la siguiente tabla de particiones:

| Partición | Tamaño | Partición | Tamaño |
|---|---|---|---|
| 1 | 10 Kb | 3 | 4 Kb |
| 2 | 4 Kb | 4 | 4 Kb |

Si llegan los siguientes procesos:

| Proceso | Tamaño | Proceso | Tamaño |
|---|---|---|---|
| 1 | 7 Kb | 3 | 6 Kb |
| 2 | 3 Kb | | |

- **¿Cuánto vale la fragmentación interna? ¿Y la externa?**




- **Fragmentación interna = 3 + 1 = 4 Kb;**
- **Fragmentación externa = 6 Kb**

# 4. Modelos básicos de traducción de direcciones

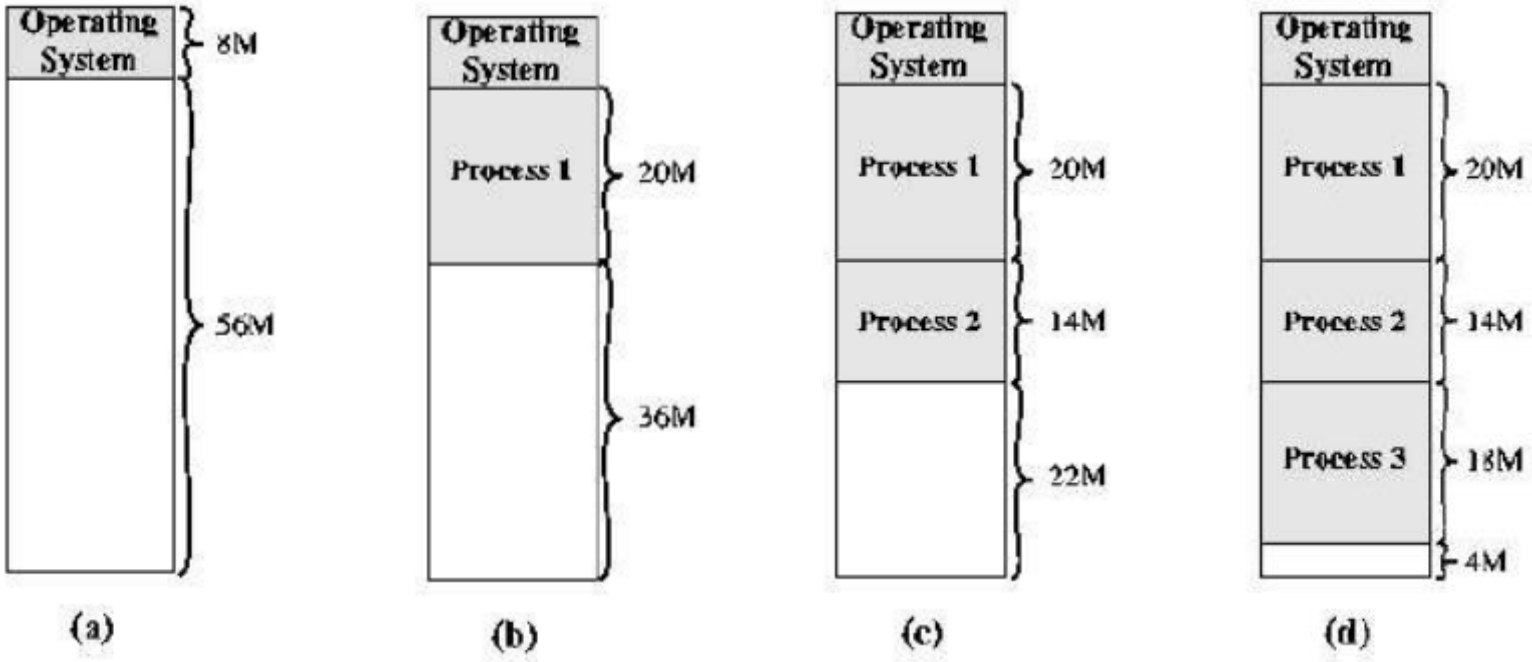
## 4.3.2 Particiones Variables (MVT)

- Cada vez que se crea un proceso, el SO busca un hueco en memoria de tamaño suficientemente grande para alojar al proceso.
- Trata de eliminar la fragmentación interna permitiendo que los tamaños de las particiones varían dinámicamente.
- El SO mantiene información en la que se indica qué partes de la memoria están disponibles y cuáles están ocupadas.
- La asignación de espacio consistirá en buscar aquel hueco lo suficientemente grande para albergar al proceso.

# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

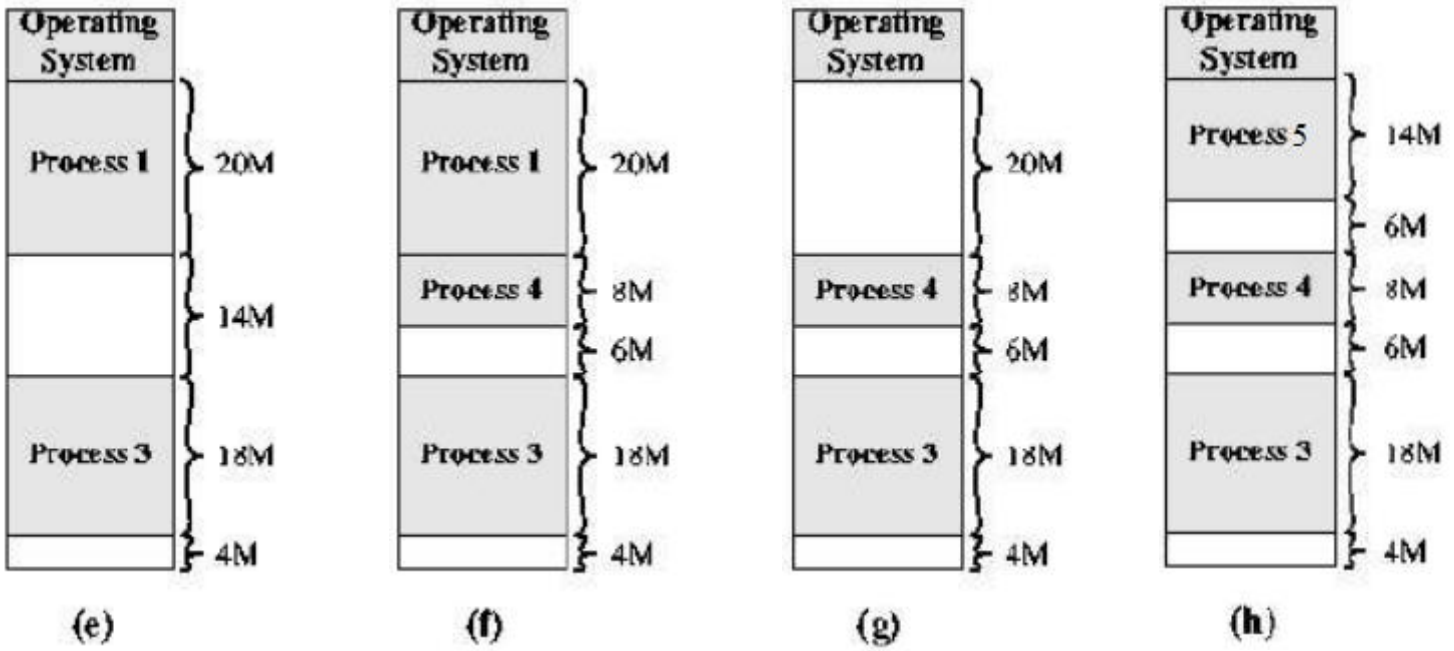
- **Secuencia de sucesos:** (a) Situación inicial (b) Llega proceso de 20 M (c) Llega proceso de 14 M (d) Llega proceso de 18 M (e) Termina proceso de 14 M (f) Llega proceso de 8 M (g) Termina proceso de 20 M (h) Llega proceso de 14 M.



# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

- **Secuencia de sucesos:** (a) Situación inicial (b) Llega proceso de 20 M (c) Llega proceso de 14 M (d) Llega proceso de 18 M (e) Termina proceso de 14 M (f) Llega proceso de 8 M (g) Termina proceso de 20 M (h) Llega proceso de 14 M.

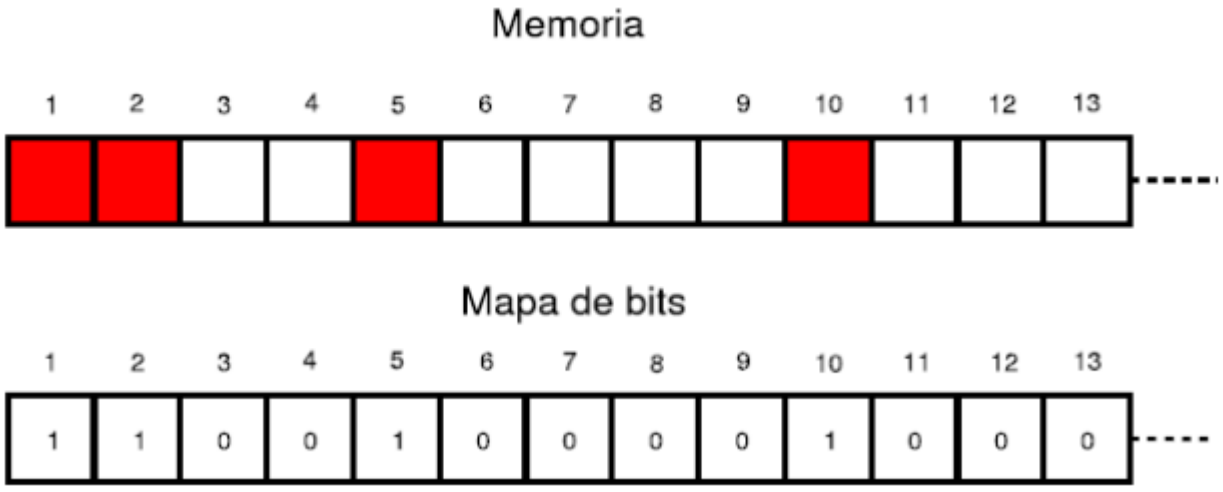


# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

Para llevar la cuenta de los huecos libres podemos usar:

- **Mapas de bits**: La memoria se divide en *clicks* del mismo tamaño. Tendremos una ristra de *bits* donde pondremos un 1 si el click de memoria está ocupado y 0 si está libre:



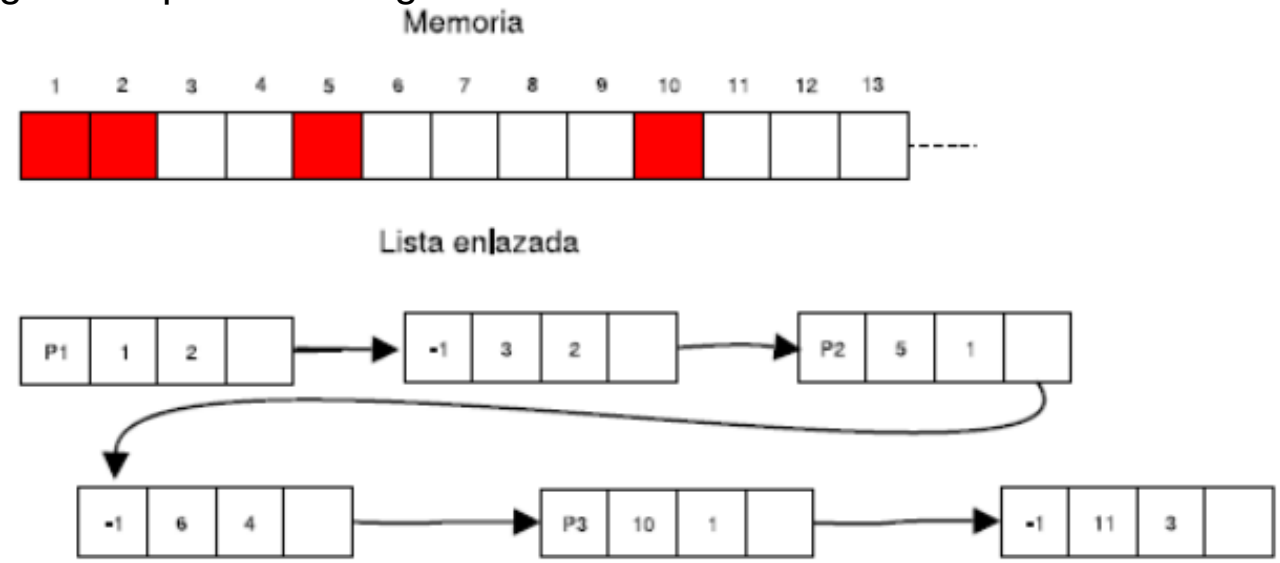

Mientras más pequeño es el click menos fragmentación interna tendremos pero la cantidad de memoria que necesitaremos para almacenarlo será mayor. El mapa de bits no indica qué proceso está ocupando los clicks.

# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

**Listas encadenadas o enlazadas**: lista en donde cada celda tiene la siguiente información:

- Id. proceso: identificador del proceso.
- @inicio: dirección donde comienza la partición.
- Tamaño: tamaño de la partición.
- Siguiente: puntero al siguiente elemento de la lista.




Los elementos de la lista de huecos suelen estar ordenados por direcciones. Esto tiene la ventaja de que la actualización de la lista cuando un proceso termina o se manda a disco es relativamente sencilla.

# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

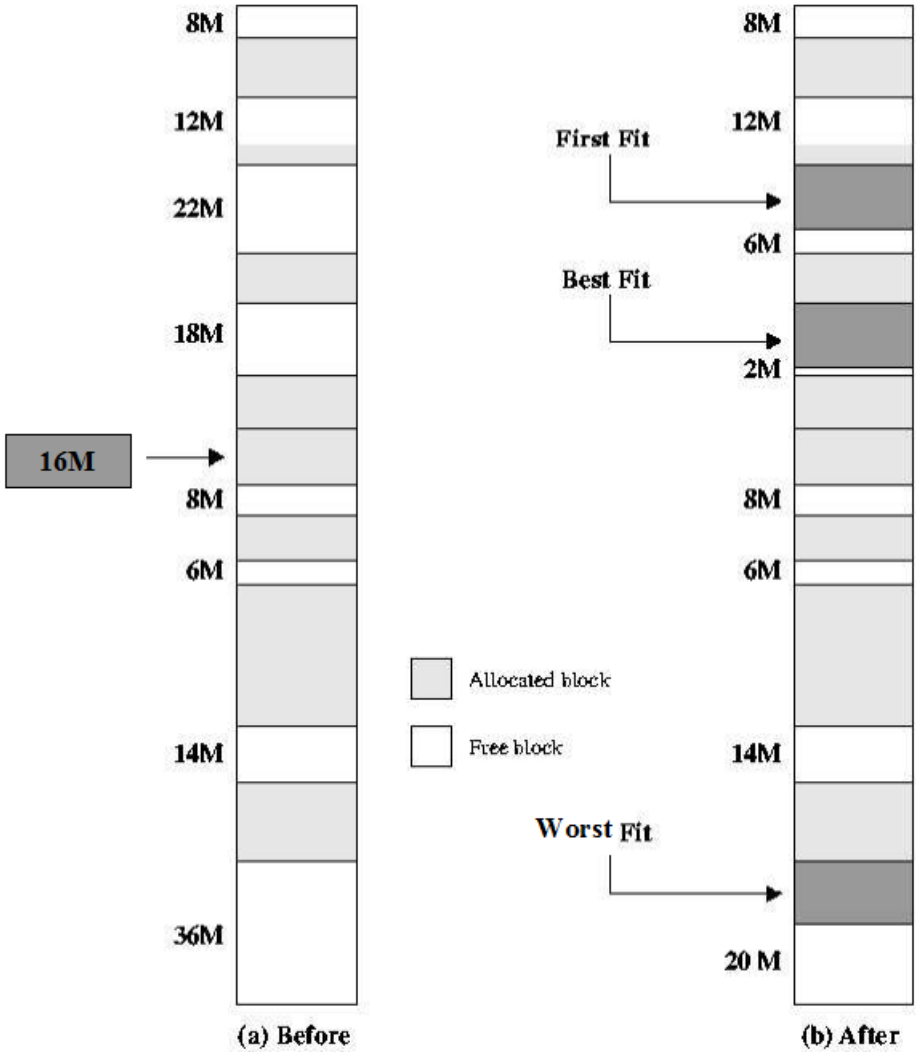
**Planificación de procesos**

- Partimos de una lista enlazada de huecos libres y ocupados.
- Disponemos de una cola en donde tenemos los procesos que han entrado al sistema por orden de llegada.
- Planificamos esta cola de manera ordenada en base a varios algoritmos:
  - **First Fit**: buscamos el primer hueco donde quepa el proceso.
  - **Next Fit**: buscamos el próximo hueco donde quepa el proceso, pero sin empezar a buscar por el principio como en first fit sino desde donde nos quedamos en la última asignación.
  - **Best Fit**: buscamos el hueco donde mejor quepa.
  - **Worst Fit**: buscamos el hueco donde peor quepa.
- **En particiones variables los procesos se asocian a particiones y no al revés: para cada proceso en cola buscamos entre las particiones disponibles**

# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

- **Ejemplo**: asignación de un bloque de 16 M



# 4. Modelos básicos de traducción de direcciones

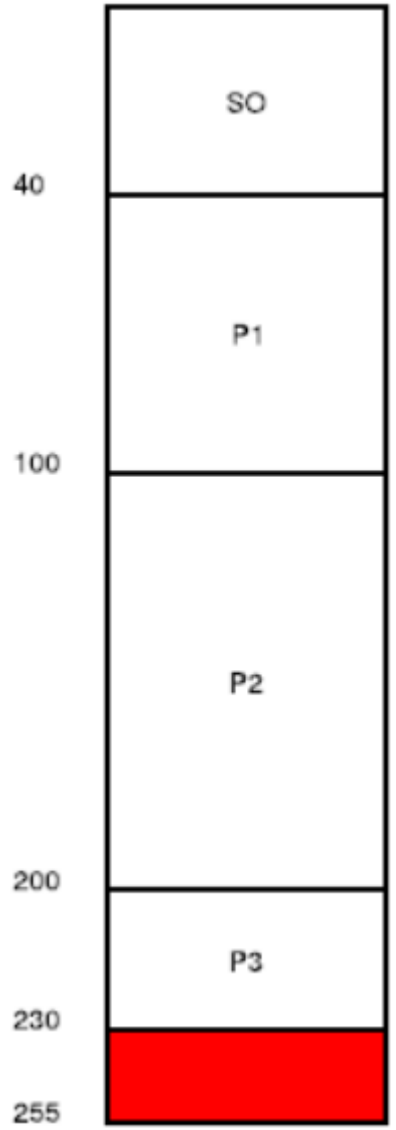
## 4.3.2 Particiones Variables (MVT)

- **Ejemplo**: tenemos una memoria de 256 Kb. 40 Kb de ellos están ocupados por el SO y llegan los siguientes procesos:

| Proceso | Tamaño | Tiempo en memoria |
|---|---|---|
| P1 | 60K | 10 seg. |
| P2 | 100K | 5 seg. |
| P3 | 30K | 20 seg. |
| P4 | 70K | 8 seg. |
| P5 | 50K | 15 seg. |

- Representar las siguientes situaciones suponiendo que usamos First Fit:

1) Llegan los tres primeros procesos.



# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

| Proceso | Tamaño | Tiempo en memoria |
|---|---|---|
| P1 | 60K | 10 seg. |
| P2 | 100K | 5 seg. |
| P3 | 30K | 20 seg. |
| P4 | 70K | 8 seg. |
| P5 | 50K | 15 seg. |

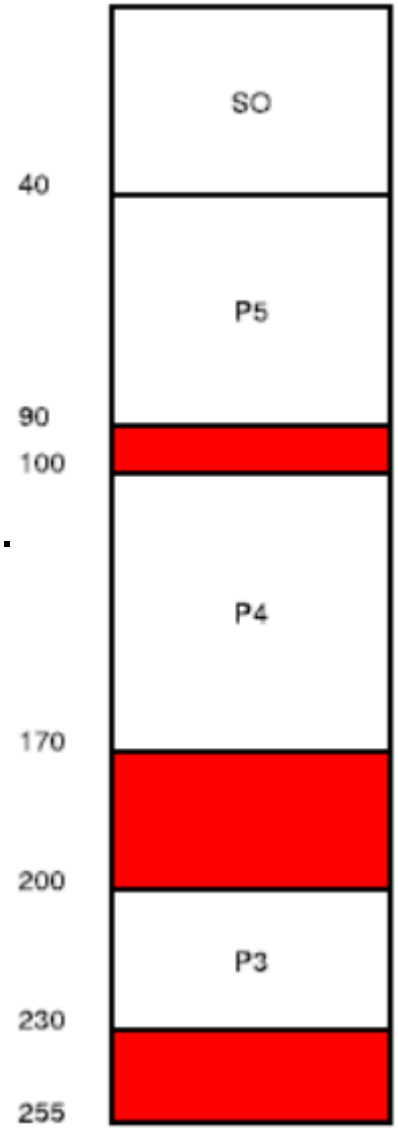
2) Sale el proceso P2 y entra el proceso P4.

# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

| Proceso | Tamaño | Tiempo en memoria |
|---|---|---|
| P1 | 60K | 10 seg. |
| P2 | 100K | 5 seg. |
| P3 | 30K | 20 seg. |
| P4 | 70K | 8 seg. |
| P5 | 50K | 15 seg. |

3) Sale el proceso P1 y entra el proceso P5.



# 4. Modelos básicos de traducción de direcciones

## 4.3.2 Particiones Variables (MVT)

**Ventajas**

- Soluciona el problema de la fragmentación interna.

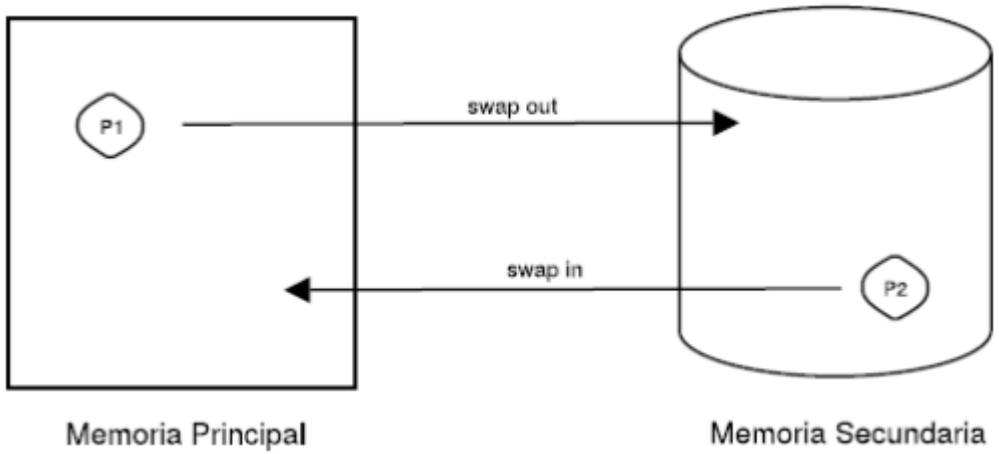
**Inconvenientes**

- Fragmentación externa, se puede solucionar mediante la **compactación.**
- En la compactación se fusionan los huecos no contiguos en un único hueco.
- Se hace una reubicación de los procesos.
- No hay algoritmo eficiente para la reubicación de los procesos en memoria.

# 4. Modelos básicos de traducción de direcciones

## 4.4. Modelo 3 (Swapping)

- Mismas características que Modelo 2 salvo la **no residencia** de los procesos.
- Se basa en usar un disco como respaldo de la memoria principal, cuando no caben en MP todos los procesos activos, se elige un proceso residente y se almacena en MS (Memoria secundaria).



# 4. Modelos básicos de traducción de direcciones

## 4.4. Modelo 3 (Swapping)

- Hay dos alternativas en la asignación de espacio en el dispositivo swap:
  - Preasignación: al crear el proceso se reserva espacio de swap.
  - Sin preasignación: sólo se reserva espacio de swap cuando se expulsa el proceso de MP.

# 4. Modelos básicos de traducción de direcciones

## 4.4. Modelo 3 (Swapping)

**Desventajas**

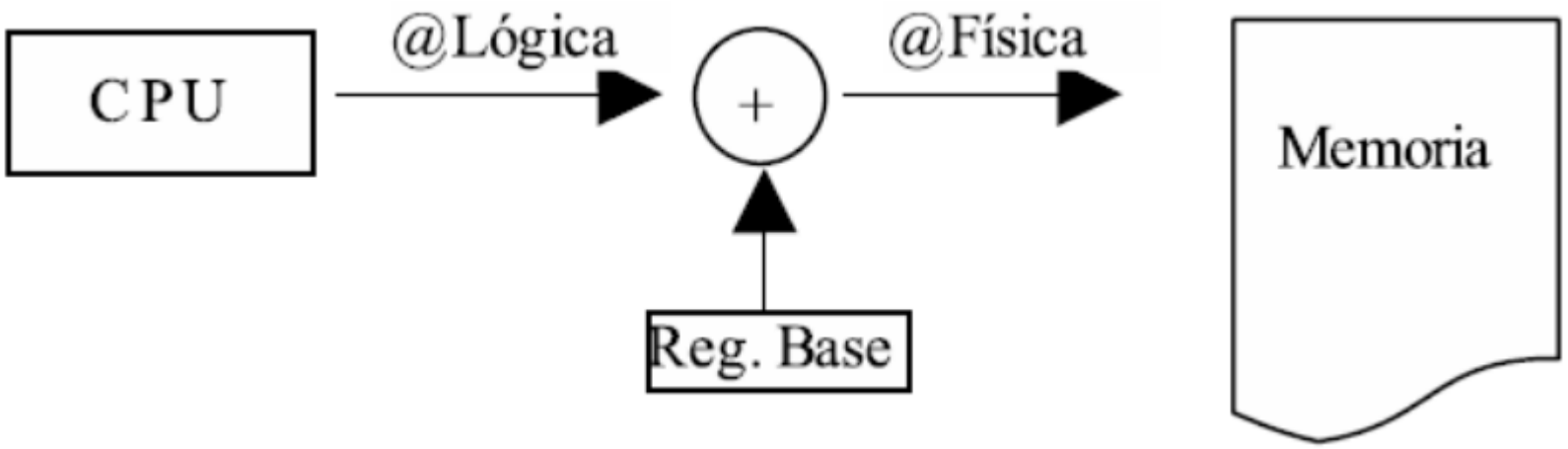
- Inmovilidad → colocar al proceso en la misma zona de memoria donde se encontraba.
- El dispositivo de almacenamiento secundario consume mucho tiempo.
- Tiempo de intercambio alto. Se mejora:
  - Aumentando la velocidad de transferencia entre Memoria Secundaria y Memoria Principal.
  - Intercambiando exactamente la cantidad de memoria necesaria.
  - Solapando la ejecución de un proceso con el intercambio del proceso de usuario previamente ejecutado y el que se ejecutará.
- Planificación: la cola de preparados consiste en aquellos procesos cuya imagen está en MS y están preparados para ejecutarse, hay que crear un nuevo estado.
- Los procesos que se encuentran en espera de realizar una operación de E/S no pueden salir de memoria.
- Puede producirse interbloqueo.

# 4. Modelos básicos de traducción de direcciones

## 4.5. Modelo 4 (Reubicación Dinámica)

Mismas características Modelo 3 salvo la **movilidad** de los procesos.

- **Se retrasa la traducción de direcciones lógicas en físicas hasta el momento de la ejecución**.
- El registro de reubicación puede cambiar su valor a lo largo de la ejecución de un mismo proceso, ahora hablamos de registro base



# Índice

# 5. Modelos avanzados de traducción de direcciones


- Se caracterizan por: multiprogramación, no residente, móvil, **no contiguo** y entero.

- La traducción se realiza en tiempo de ejecución.

- Existen 3 modelos que cumplen estas características: Paginación, Segmentación y Modelos que combinan la paginación y la segmentación.

# 5. Modelos avanzados de traducción de direcciones

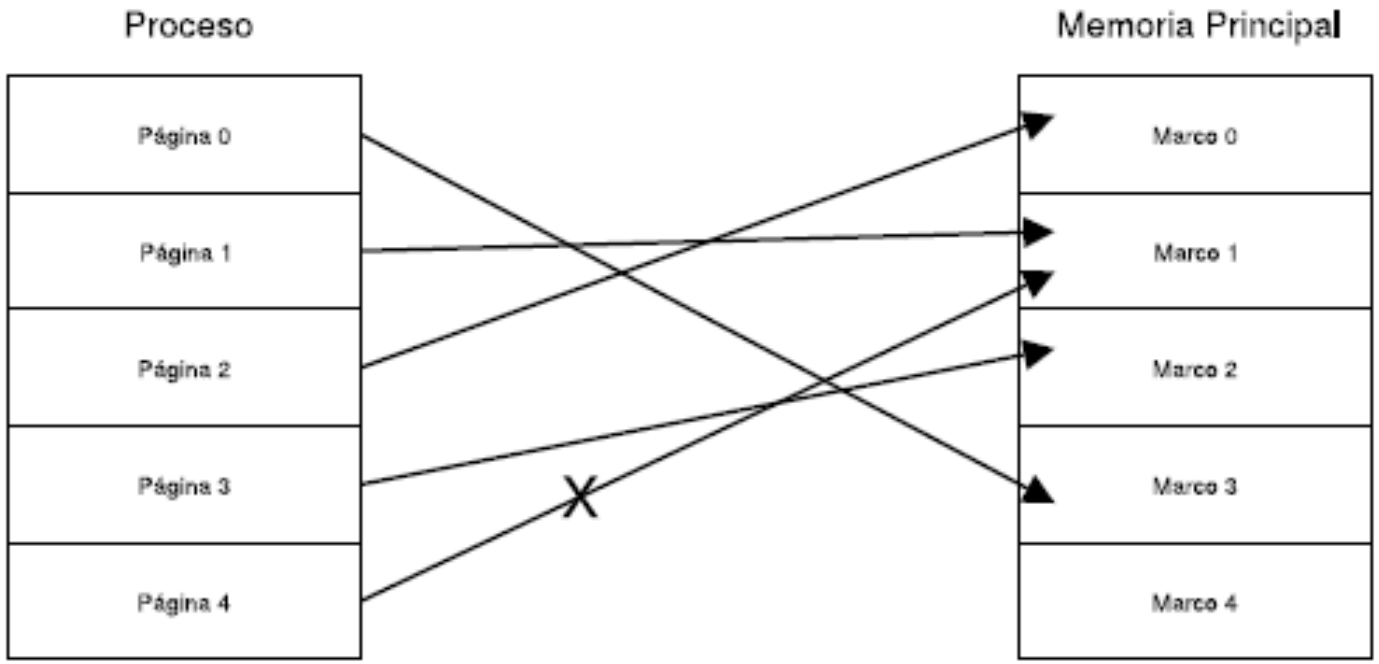
## 5.1. Paginación

- La fragmentación externa se debe a la imposibilidad de asignar espacios no contiguos de memoria.
- La paginación intenta solucionar este problema sin usar compactación.
- La memoria se divide en trozos denominados **marcos o tramas** de un tamaño fijo.
- El proceso se divide en bloques denominados **páginas** del mismo tamaño que los marcos.

# 5. Modelos avanzados de traducción de direcciones

## 5.1. Paginación

- El SO realiza esta división y asigna páginas a marcos:




- Para establecer la relación entre páginas y marcos se usa la tabla de páginas.

# 5. Modelos avanzados de traducción de direcciones

## 5.1. Paginación

**Estructuras**

- El SO necesita nuevas estructuras para implementar la paginación.
- Una es la **tabla de marcos** de página que indicará los marcos que están libres y ocupados
- Otra es la **tabla de páginas** que servirá para indicar en qué marco está situada una página. Es una tabla indexada por paginas cuya estructura (aunque depende de la computadora) es la siguiente:

| | t | BV | R | W | X |
|---|---|---|---|---|---|
| 0 | | | | | |
| 1 | | | | | |
| | | | | | |
| N | | | | | |

- **BV** es un bit de validez (si vale 1 indica que la entrada es válida)
- **R,W,X** son bits de protección (Opcional)
- **t** indica el marco que contiene la página

- Hay una tabla de páginas por proceso.

# 5. Modelos avanzados de traducción de direcciones

## 5.1. Paginación

### Planificación de procesos

- Estructuras de datos: cola de trabajos (necesidades de memoria), lista de marcos no asignados, tablas de páginas (en cada proceso).

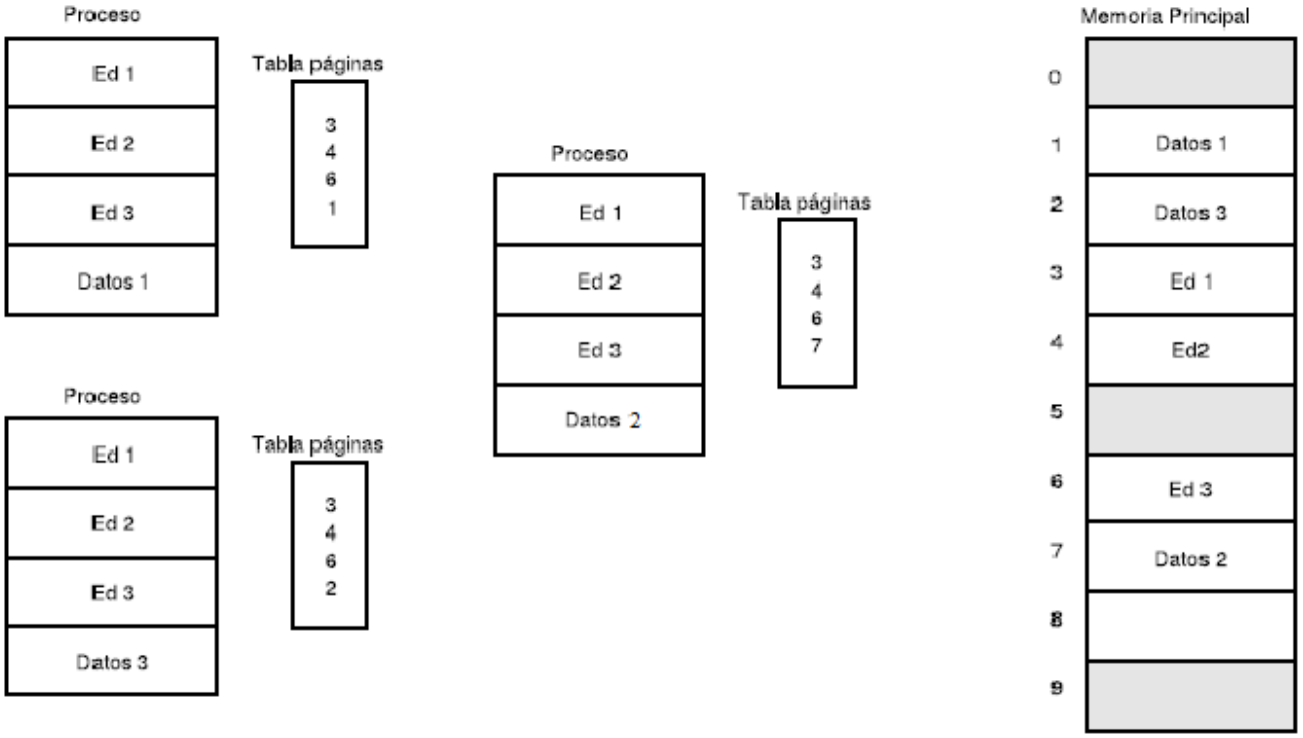
- Algoritmo
  - Si un trabajo requiere **n** páginas, buscamos **n** marcos disponibles y se los asignamos.
  - Cargamos cada página en el marco que le corresponde.
  - Actualizamos los valores de la tabla de páginas.

# 5. Modelos avanzados de traducción de direcciones

## 5.1. Paginación

**Ventajas**

- No se produce fragmentación externa.
- Protección de páginas y control de direcciones ilegales.
- Compartición de páginas.



# 5. Modelos avanzados de traducción de direcciones


## 5.1. Paginación

**Inconvenientes**

- **Fragmentación interna**, se puede reducir disminuyendo el tamaño de página → tabla de páginas mayor.

# 5. Modelos avanzados de traducción de direcciones

## 5.1. Paginación

**¿Cómo se almacena esta tabla?**

- Tabla de paginas con un único nivel.
- Tablas de páginas multinivel.
- Tablas de páginas invertidas.
- Buffer de traducción adelantada.

# 5. Modelos avanzados de traducción de direcciones

## 5.1.1. Tabla de páginas con un único nivel

- Cuando un proceso se ejecuta se carga completamente su tabla de páginas en memoria en posiciones consecutivas.

- ¿Dónde se almacena esta tabla?

  - Usando **registros** de propósito general, la tabla de paginas almacenada en algún dispositivo se carga en registros. Gran rapidez de acceso pero las tablas no suelen caber.

  - Almacenamos la tabla en **memoria principal**, en el registro PTBR guardaremos la dirección de la tabla en memoria. Este valor se actualiza cuando hay cambio de proceso.

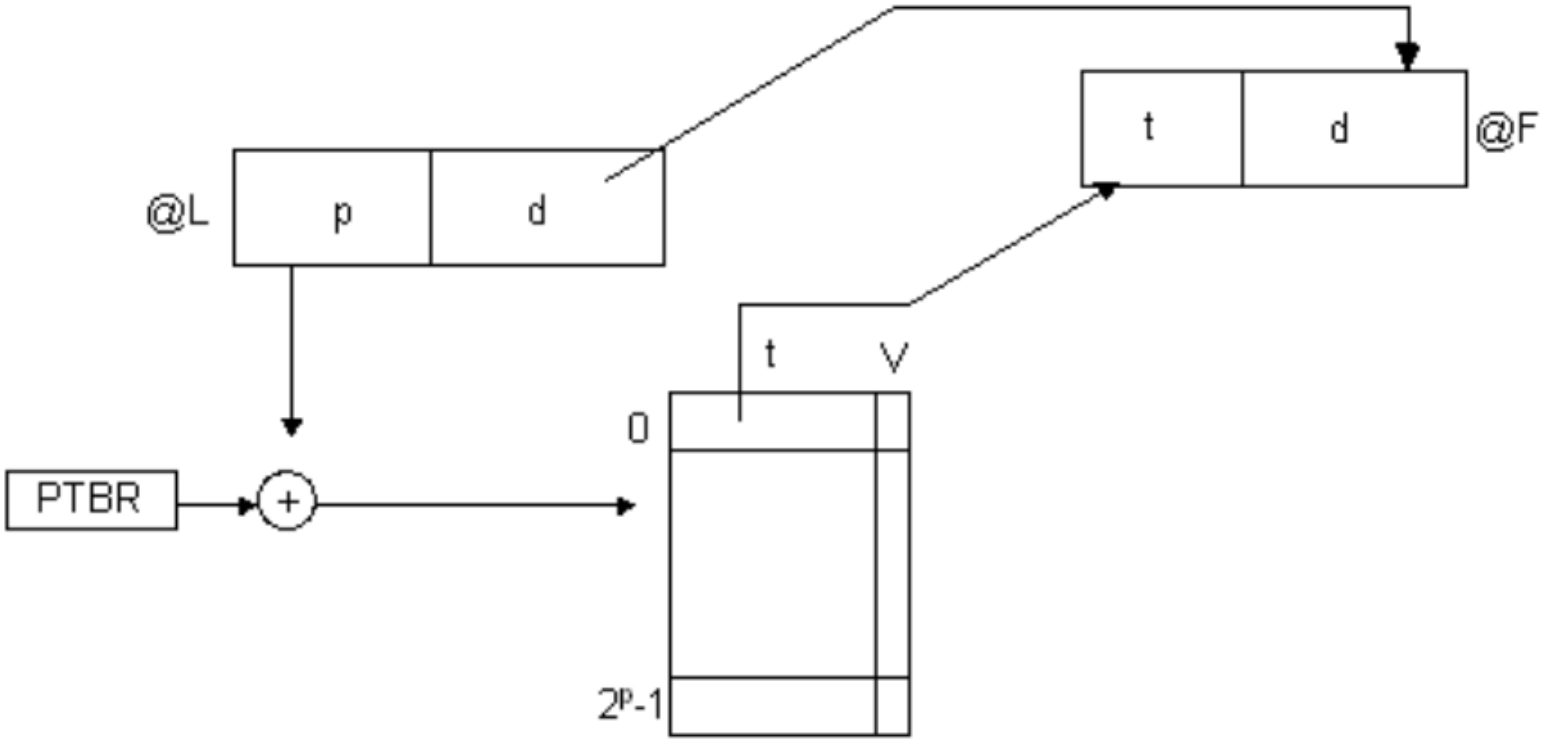
  - El esquema anterior presenta problemas de velocidad. Se opta por una solución basada en **registros asociativos** (memoria caché) denominado TLB (ver punto siguiente).

# 5. Modelos avanzados de traducción de direcciones

## 5.1.1. Tabla de páginas con un único nivel

**Traducción de direcciones, hardware de paginación**

- El esquema de ejecución de una instrucción es:



# 5. Modelos avanzados de traducción de direcciones


## 5.1.1. Tabla de páginas con un único nivel

- La CPU genera direcciones lógicas **L** que tienen dos componentes:
  - Un número de página (**p**).
  - Un desplazamiento en la página (**d**).
- En general, **d** se obtiene del tamaño de la página y **p** de la diferencia entre **L** y **d**.
- Una vez que tenemos calculado **p**, usamos la entrada **p** de la tabla de páginas para calcular el **número del marco** que la contiene.
- Este valor (**t**) lo concatenamos (||) con **d**, y el resultado final es la dirección física.

# 5. Modelos avanzados de traducción de direcciones

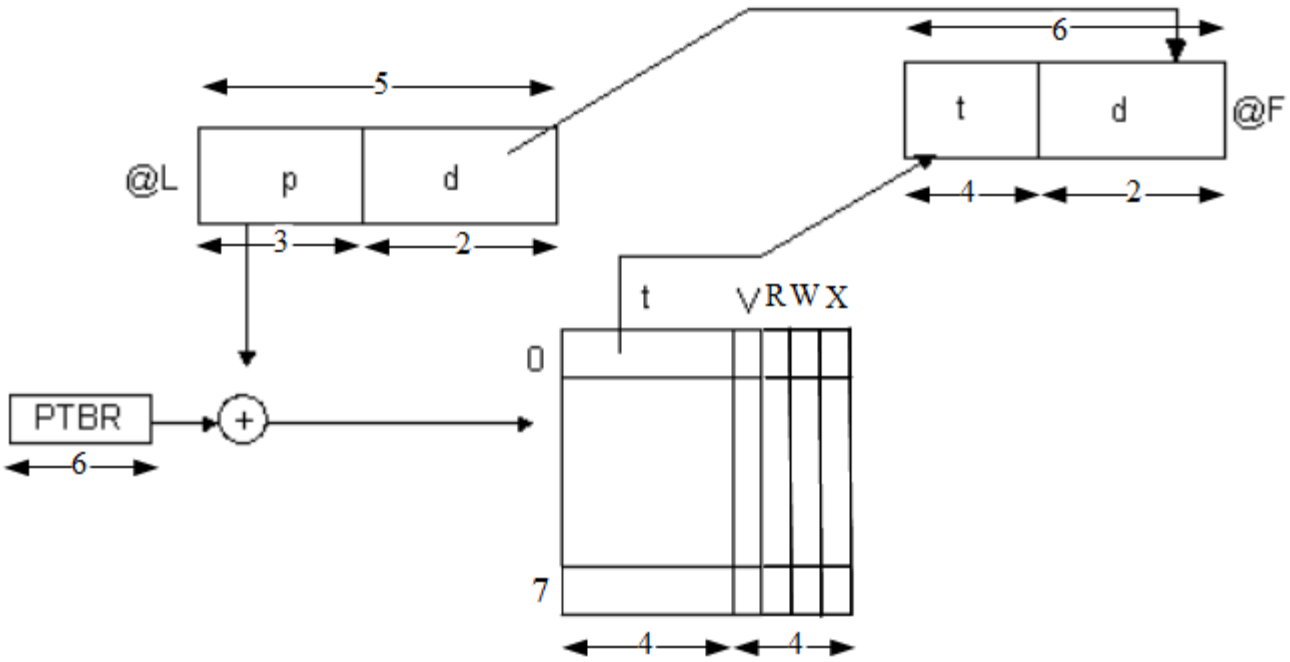

## 5.1.1. Tabla de páginas con un único nivel

- **Ejemplo 1**: Tenemos una mini memoria de 64 bytes. Las páginas son de 4 bytes y la @Lógica tiene 5 bits. Las tablas de páginas tienen bits de protección de R,W,X. Dibuja el esquema de traducción de direcciones e indica el tamaño de cada campo.

- Necesitamos 6 bits para direccionar 64 bytes ⇒ @Fisica= 6
- Si las páginas son de 4 bytes ⇒ d=2
- p=L−d=5−2=3 bits más significativos ⇒ un proceso puede tener 8 páginas

# 5. Modelos avanzados de traducción de direcciones

## 5.1.1. Tabla de páginas con un único nivel



# 5. Modelos avanzados de traducción de direcciones

## 5.1.1. Tabla de páginas con un único nivel

- **Ejemplo 1**: Si un proceso A de 16 bytes se almacena en las tramas 5, 11, 9 y 2 respectivamente, ¿cómo queda la tabla de páginas?

Proceso A

| |
|---|
| P0 |
| P1 |
| P2 |
| P3 |

| | t | V | R | W | X |
|---|---|---|---|---|---|
| 0 | 5 | 1 | | | |
| 1 | 11 | 1 | | | |
| 2 | 9 | 1 | | | |
| 3 | 2 | 1 | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |

| | |
|---|---|
| | T0 |
| | T1 |
| P3 | T2 |
| | T3 |
| | T4 |
| P0 | T5 |
| | T6 |
| | T7 |
| | T8 |
| P2 | T9 |
| | T10 |
| P1 | T11 |
| | T12 |
| | T13 |
| | T14 |
| | T15 |

# 5. Modelos avanzados de traducción de direcciones

## 5.1.1. Tabla de páginas con un único nivel

- **Ejemplo 1**: ¿Si el proceso A lanza la @Lógica 9 que @Física se genera?

- 9 = 1001 = 010 || 01 = @Lógica (5 bits)
- p= 010 d= 01 (Segunda instrucción de la página 2)
- La @Física es de 6 bits, 4 bits para la t y 2 bits para la d. Accedemos por lo tanto a la segunda instrucción almacenada en la trama 9, cuya dirección física exacta es:
- @Física = 1001 || 01 = 37

# 5. Modelos avanzados de traducción de direcciones

## 5.1.1. Tabla de páginas con un único nivel

- **Ejemplo 2:** Si tenemos una máquina con páginas de 256 bytes y la dirección lógica ocupa 10 bits, y la tabla de páginas siguiente:

| | t |
|---|---|
| 0 | 0 |
| 1 | 3 |
| 2 | 2 |
| 3 | 1 |

La siguiente dirección lógica 1101111101, ¿a qué marco y dirección física accede?

- 256= $2^8$ → d=8 *bits* menos significativos = 01111101 = 125
- p=10-8=2 *bits* más significativos = 11 → accedo a la página 3 que está en la trama 1.
- La dirección física se calcula concatenando a la dirección del marco el desplazamiento:

@física = t || d = 1 || 125 = 01 || 01111101 = 101111101 = 381

# 5. Modelos avanzados de traducción de direcciones

## 5.1.2. Buffer de traducción adelantada

- En general cada instrucción suele generar varios accesos a memoria → accesos a la tabla de páginas → accesos a memoria.
- Para reducir el tiempo de búsqueda del marco de página se hace uso de una cache especial, incluida en la MMU, para las entradas de la tabla de páginas llamada **buffer de traducción adelantada** (TLB).
- Esta caché almacena las **entradas completas de la tabla de páginas más usadas** junto con el identificador de la página**:**

| Número página | Entrada Tabla página |
|---|---|
| 0 | |
| : | |
| : | |
| n | |

- El identificador de páginas sirve para hacer búsquedas asociativas.

# 5. Modelos avanzados de traducción de direcciones

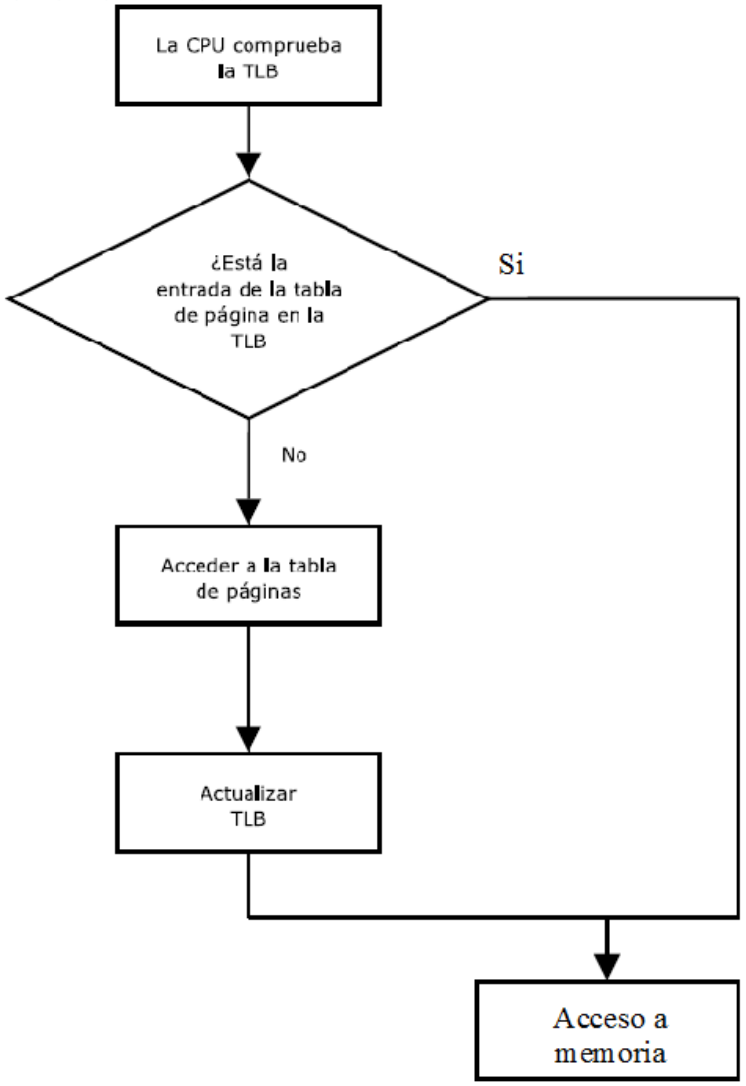
## 5.1.2. Buffer de traducción adelantada

- Dos alternativas para el diseño TLB:
  - TLB sin identificador de proceso: cuando hay un cambio de proceso se invalida la TLB completa
  - TLB con identificador de proceso: se añade un campo más para identificar al proceso, no es necesario invalidar la TLB cuando hay cambio de contexto.

# 5. Modelos avanzados de traducción de direcciones

## 5.1.2. Buffer de traducción adelantada

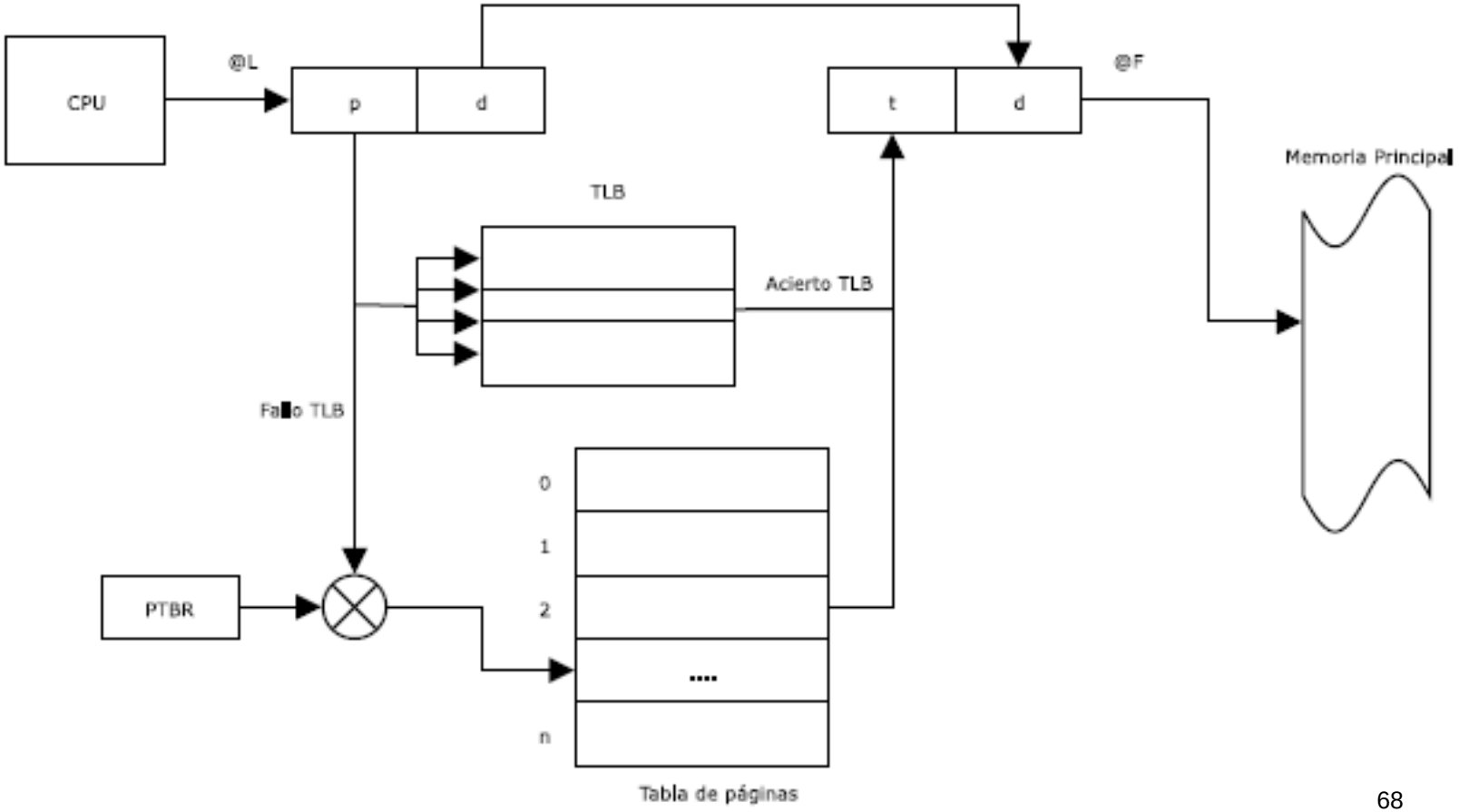
- Una TLB sin identificador de proceso se actualiza siguiendo el siguiente diagrama de flujo:



# 5. Modelos avanzados de traducción de direcciones

## 5.1.2. Buffer de traducción adelantada

- Si usamos tablas de un único nivel la MMU quedaría de la siguiente manera:



# 5. Modelos avanzados de traducción de direcciones

## 5.1.3. Tabla de páginas multinivel

- Se divide la tabla de paginas en secciones.
- Cada proceso tiene asociado una **jerarquía de tablas de páginas**.
- Si esta jerarquía es de 2 niveles:
  - El **primer nivel** es un directorio **raíz de páginas**, cada entrada apunta a tablas de páginas (secciones):

| | tabla_paginas | BV |
|---|---|---|
| 0 | | |
| 1 | | |
| : | | |
| N | | |

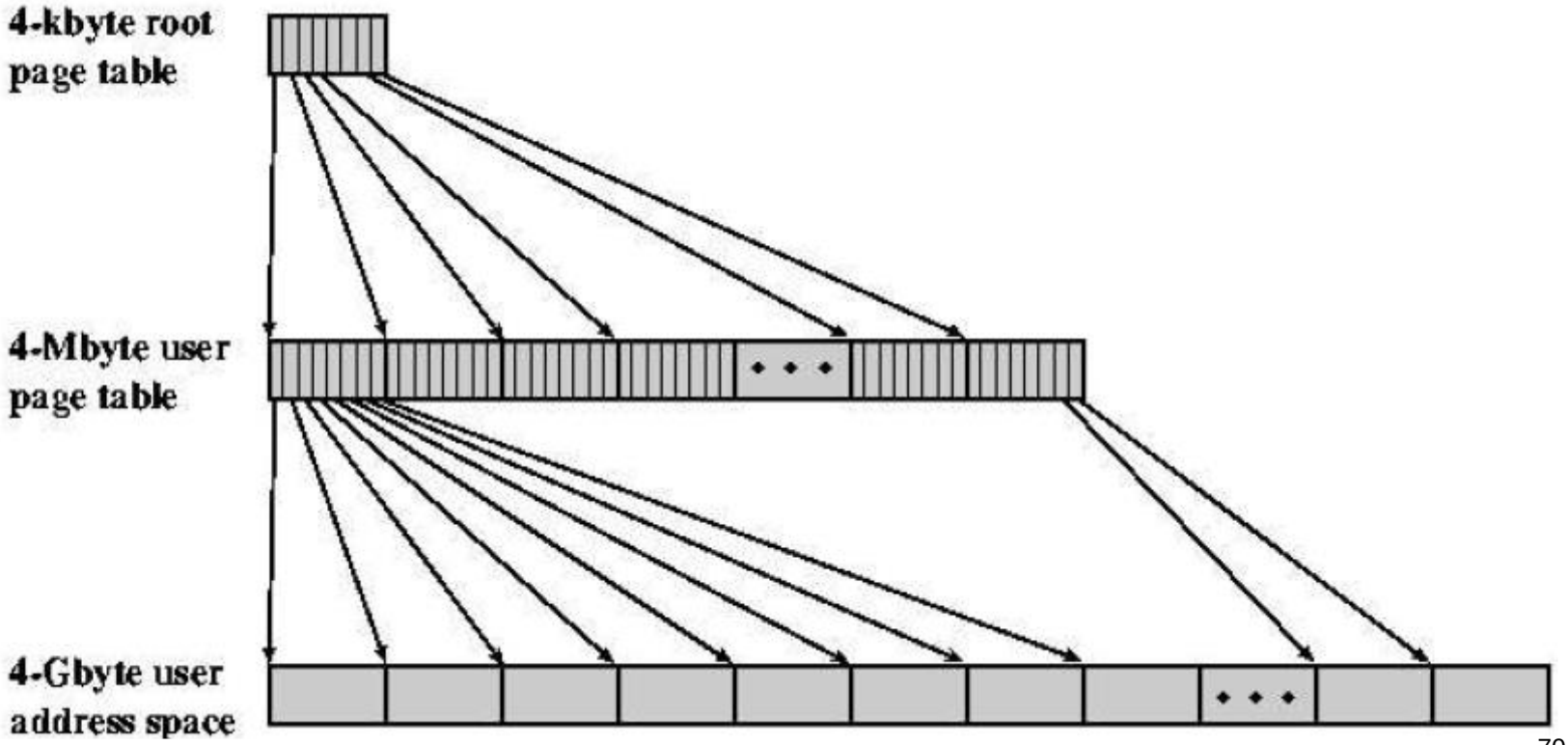
    - **tabla_paginas** es la **dirección en memoria** de la tabla de páginas.
    - **BV** es un bit de validez (si vale 1 indica que la entrada es válida).

  - Un **segundo nivel** con las tablas de páginas.

# 5. Modelos avanzados de traducción de direcciones

## 5.1.3. Tabla de páginas multinivel

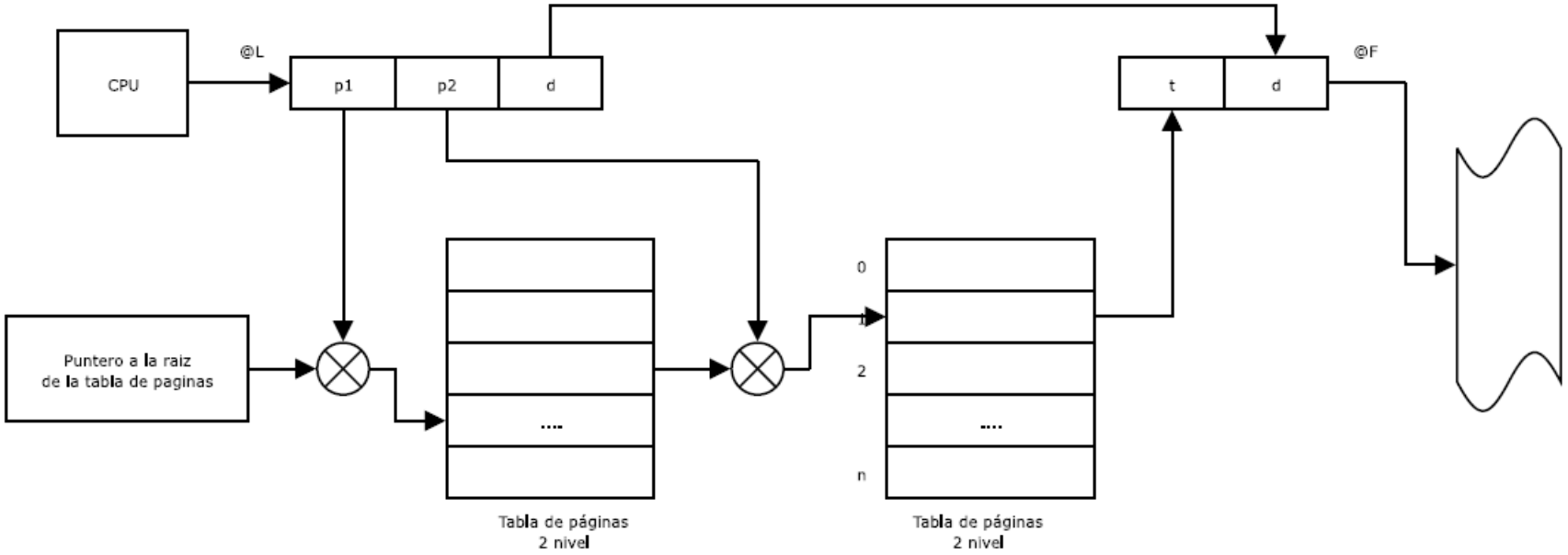
- Pueden existir más de dos niveles.
- El esquema general es:



# 5. Modelos avanzados de traducción de direcciones

## 5.1.3. Tabla de páginas multinivel

- El esquema de ejecución de una instrucción es:




- La CPU genera direcciones lógicas **L** que tienen tres componentes:
  - La tabla de páginas a la que queremos acceder (**p1**).
  - El número de página (**p2**).
  - Un desplazamiento en la página (**d**).

# 5. Modelos avanzados de traducción de direcciones

## 5.1.3. Tabla de páginas multinivel

- **Ejemplo 1**: Si tenemos direcciones lógicas de 32 bits, un tamaño de pagina de 4 Kb, cada entrada de la tabla de paginas ocupa 4 bytes y se dedican 10 bits de la dirección a cada nivel ¿Qué espacio de direcciones cubre cada tabla de segundo nivel?¿Qué espacio total se puede direccionar?

- Las tablas de páginas de 2º nivel tienen $2^{10}$ entradas.
- Como cada marco ocupa 4Kb, cada tabla de 2º nivel direcciona $\Rightarrow 2^{10}$ x 4 Kb $\Rightarrow$ 4 Mb.
- El espacio total que se puede direccionar con este esquema es de $2^{10}$ x 4 *Mb*= 4Gb.

# 5. Modelos avanzados de traducción de direcciones

## 5.1.3. Tabla de páginas multinivel

- **Ejemplo 2**: Teniendo en cuenta los datos del ejemplo anterior, si un proceso utiliza sólo los 12 Mb de la parte superior de su mapa de memoria y 4 Mb de la parte inferior ¿cuánto espacio necesitaríamos para almacenar toda su tabla de páginas? ¿y si usáramos un esquema con un único nivel?

- Gastamos el espacio de la tabla de 1º nivel ($2^{10}$ entradas x 4 bytes = 4 Kb), como tenemos que direccionar 16 Mb necesitamos 4 tablas secundarias (cada una ocupa 4 Kb y cada una direcciona 4 Mb). En total necesitamos almacenar 4 Kb de la tabla de 1º nivel + 16 Kb de tablas secundarias = 20Kb.
- Si usáramos un único nivel es necesario tener almacenada completamente la tabla de páginas ⇒ $2^{20}$ x 4 bytes = 4 Mb.

# 5. Modelos avanzados de traducción de direcciones

## 5.1.3. Tabla de páginas multinivel

### Ventajas

- Requieren menos espacio en memoria principal.
- Permite compartir páginas intermedias.
- Sólo es preciso que esté residente en memoria la tabla del primer nivel.

# 5. Modelos avanzados de traducción de direcciones

## 5.1.4. Tabla de páginas invertida

- Para reducir el tamaño se usa una **tabla de páginas invertida**.
- Esta tabla contiene **una entrada por cada marco de página** y está indexada por marco:

| | Página | PID |
|---|---|---|
| 0 | 7 | |
| 1 | 4 | |
| .. | .. | |
| N | 3 | |

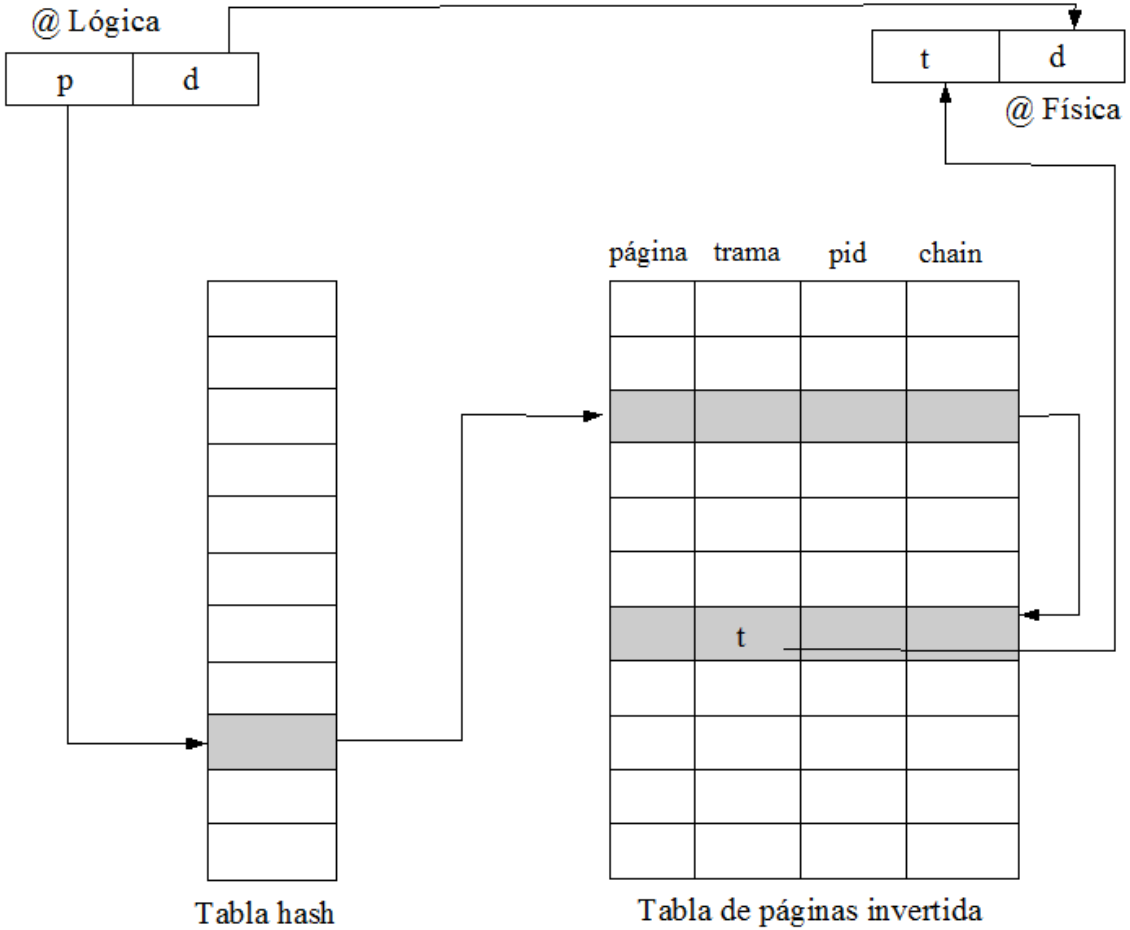
  - **PID** es el identificador de proceso que ocupa el marco.
  - **Página** es la página que ocupa el marco.

- Como la tabla está indexada por marcos no se puede hacer una búsqueda directa por páginas.

# 5. Modelos avanzados de traducción de direcciones

## 5.1.4. Tabla de páginas invertida

- Para mejorar la velocidad de búsqueda la parte del número de página de una dirección virtual se traduce a una **tabla hash** por medio de una función hash:



# 5. Modelos avanzados de traducción de direcciones

## 5.1.4. Tabla de páginas invertida

- Es necesario incluir el campo **encadenamiento** (chain) que es un apuntador a la siguiente entrada de la tabla de páginas invertida que comparte el mismo código hash.

- A partir de la página **p** se usa la función hash para acceder a la primera posición de la tabla de paginas invertidas.

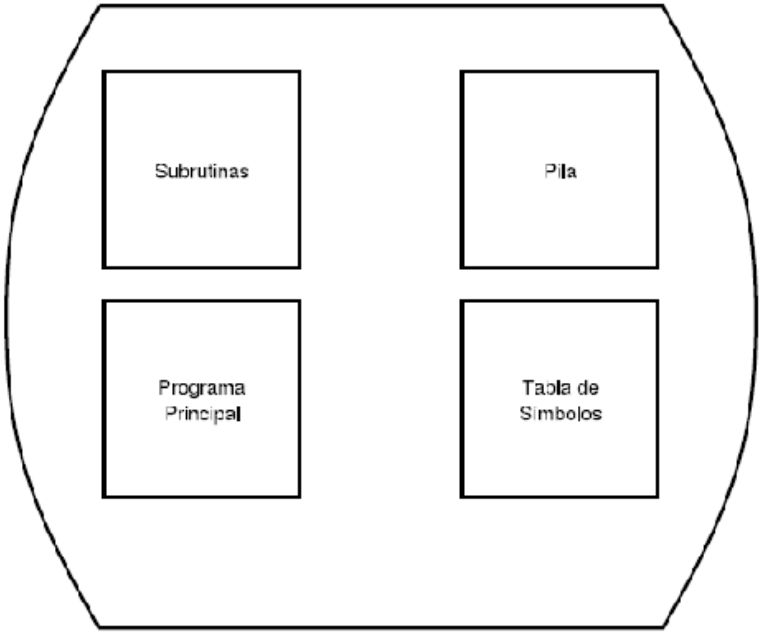
- Si el identificador de la página contenida en la tabla de paginas invertidas coincide con p, leemos la entrada de la tabla de páginas para obtener el número del marco.

- Si el identificador de la pagina no coincide, buscamos en la siguiente entrada de la tabla de páginas invertida indicada en el campo **encadenamiento**.

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

- La segmentación apareció en las arquitecturas Intel para permitir a los programadores dividir sus programas en “entidades lógicamente relacionadas entre sí”:




- Estas entidades se denominan segmentos y el SO les irá asignando el espacio en memoria que necesiten (similar al particionamiento variable) ⟶ no hay fragmentación interna.

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

- El SO mantiene **una tabla de segmentos por proceso** cuyo objetivo es almacenar la información necesaria para localizar los segmentos en memoria, **se almacena en posiciones consecutivas de memoria**.
- La estructura de esta tabla es:

| | @Base | límite |
|---|---|---|
| 0 | | |
| 1 | | |
| .. | | |
| N | | |

  - **@Base** es la dirección física que ocupa el segmento.
  - **Límite** es el número de posiciones del segmento (realmente se pone el número de posiciones del segmento -1 como veremos más adelante).

# 5. Modelos avanzados de traducción de direcciones

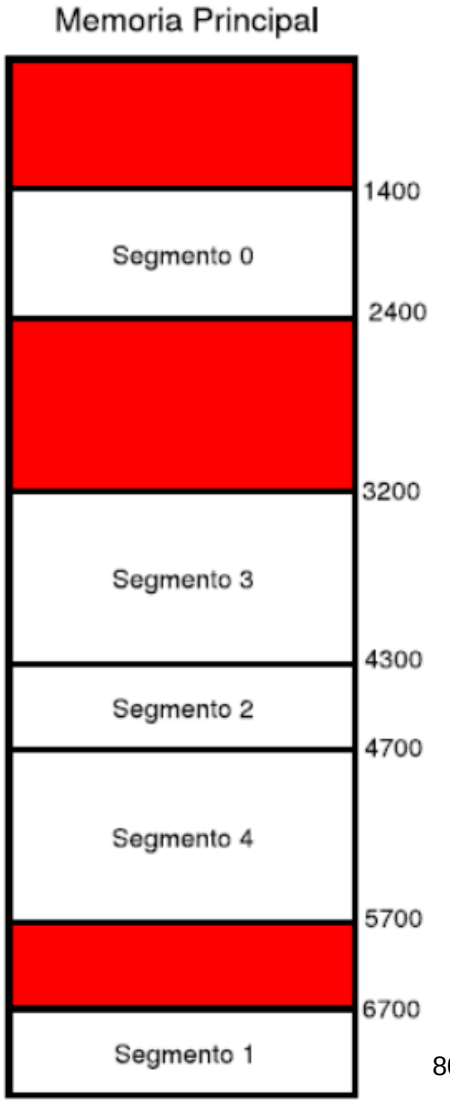
## 5.2. Segmentación

**Ejemplo**:

- Memoria de 7 Kb (7168 bytes). 5 Segmentos de 1000, 468, 400, 1100 y 1000 bytes respectivamente.

Tabla de segmentos

| | base | límite |
|---|---|---|
| 0 | 1400 | 999 |
| 1 | 6700 | 467 |
| 2 | 4300 | 399 |
| 3 | 3200 | 1099 |
| 4 | 4700 | 999 |



# 5. Modelos avanzados de traducción de direcciones

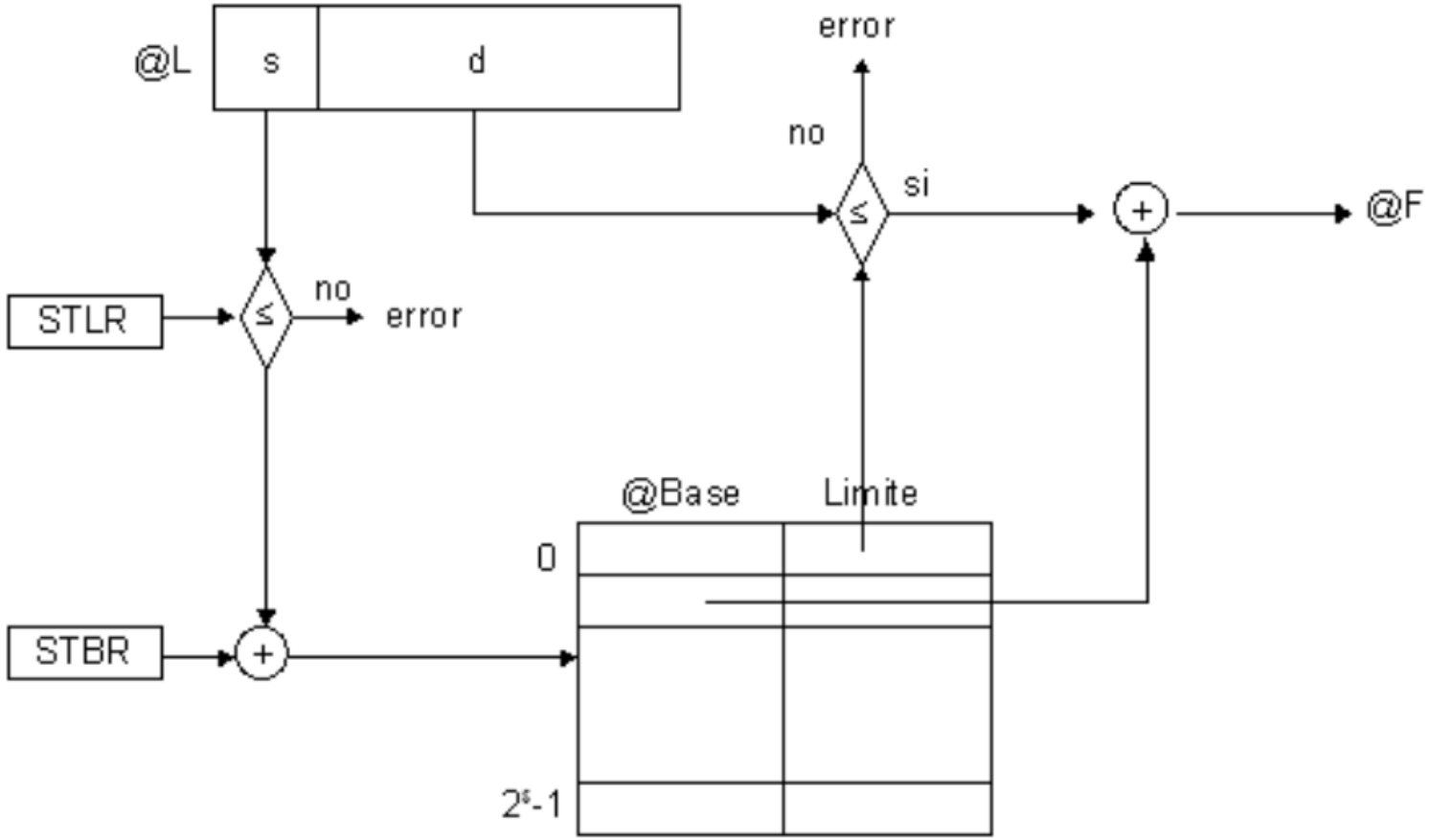
## 5.2. Segmentación

- ¿Dónde se carga esta tabla?
  - En **registros** de propósito general (CS, SS, DS). Gran rapidez de acceso pero las tablas no suelen caber.
  - En **memoria principal**, el registro STBR guarda la dirección de la tabla en memoria y el STLR el número de registros máximo en la tabla (**entonces no es necesario el BV**).
  - **Registros asociativos** (memoria caché), mejoran la velocidad.

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

- El esquema de ejecución de una instrucción es:



# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

- ¿Por qué el límite se debe de poner el número de posiciones del segmento -1?
  - Si el tamaño del segmento es de 1024 posiciones → necesitamos 10 bits para poder direccionarlos
  - Tanto d como Limite tienen un tamaño de 10 bits pero es que en 10 bits el número máximo que podemos representar es 1111111111 = 1023 y no 1024

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

- **Ejemplo 1:** Si el máximo número de segmentos que puede tener un programa es 4 y la dirección lógica ocupa 10 bits, ¿cuántos bits necesitamos para codificar el segmento y el desplazamiento?:

  - Necesitamos 2 bits para direccionar los 4 segmentos ⇒ s= 2 bits más significativos

  - d=L−s=8 bits menos significativos ⇒ el número de posiciones máximo de un segmento puede ser $2^8 = 256$ posiciones.

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

- **Ejemplo 2:** Usando la arquitectura anterior, y la tabla de segmentos siguiente:

| Segmento | @base | límite |
|---|---|---|
| 0 | 0 | 99 |
| 1 | 300 | 499 |
| 2 | 1000 | 199 |
| 3 | 1500 | 99 |

La siguiente dirección lógica 1101111101, ¿a qué segmento y dirección física accede?

- *s* = 2 *bits* más significativos = 11 accedo al segmento 3 de límite 99.
- $d=L-s=10-2=8$ *bits* menos significativos = 01111101 = 125
- El desplazamiento supera al tamaño del segmento (125>99)⇒ infracción de memoria.

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

- **Ejemplo 2:** Usando la arquitectura anterior, y la tabla de segmentos siguiente:

| | @base | Límite |
|---|---|---|
| 0 | 0 | 99 |
| 1 | 300 | 255 |
| 2 | 1000 | 199 |
| 3 | 1500 | 99 |

¿y si la dirección fuera 1100110010?

- s = 2 bits más significativos = 11 accedo al segmento 3 de límite 99.
- d=L−s=10−2=8 bits menos significativos = 00110010 = 50.
- La dirección lógica se calcula sumando a la dirección del segmento (1500) el desplazamiento (50):

  @física = @Base + d = 1500 + 50 = 1550 = 11000001110 (11 bits).

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

**Planificación de procesos**

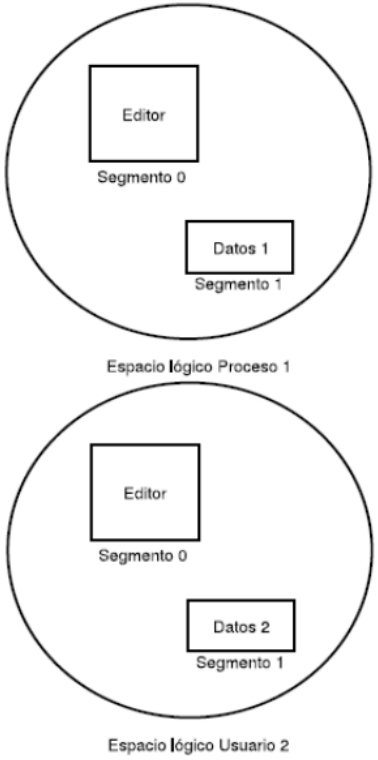
- Se realiza de forma similar a Particiones Variables (MVT).
- Estructuras de datos: cola de trabajos (necesidades de memoria), lista de huecos no asignados.
- Algoritmo
  - Si un trabajo tiene **n** segmentos, buscamos **n** huecos disponibles y se los asignamos:
    - First fit.
    - Best fit.
    - Otros esquemas.
- Cargamos cada segmento en el hueco que le corresponde.
- Actualizamos los valores de la tabla de segmentos.

# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

**Ventajas**

- Protección de segmentos.
- Compartir segmentos.

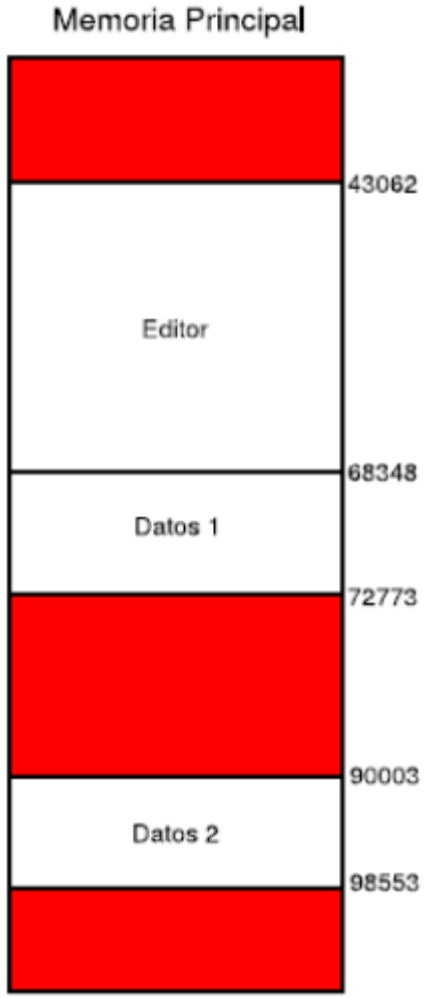



| | limite | base |
|---|---|---|
| 0 | 25285 | 43062 |
| 1 | 4424 | 68348 |

Tabla de segmentos Proceso 1

| | limite | base |
|---|---|---|
| 0 | 25285 | 43062 |
| 1 | 8549 | 90003 |

Tabla de segmentos Proceso 2



# 5. Modelos avanzados de traducción de direcciones

## 5.2. Segmentación

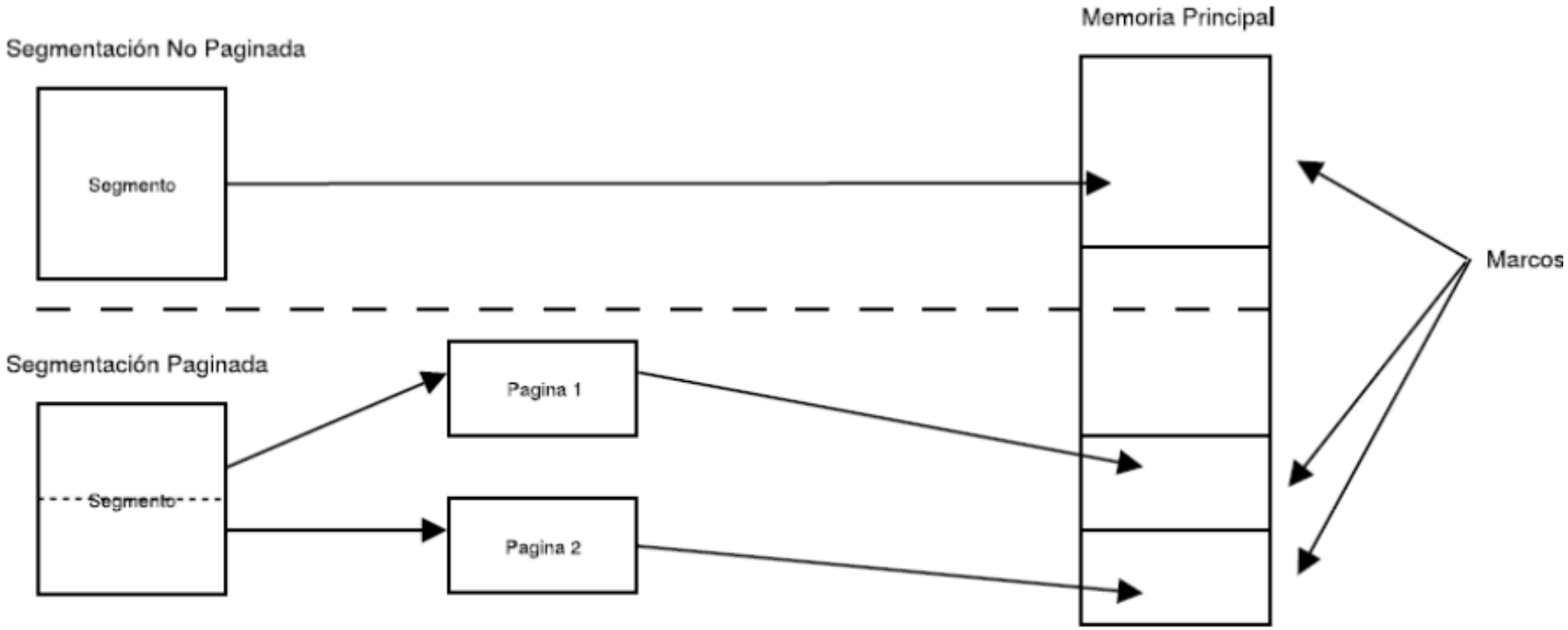
**Inconvenientes**

- Fragmentación externa.
- Gasto de tiempo en la búsqueda de un número de huecos adecuado.

# 5. Modelos avanzados de traducción de direcciones

## 5.3. Métodos combinados

- Se beneficia de las ventajas que aporta tanto la paginación como la segmentación.
- Reduce la fragmentación provocada por la segmentación mediante la paginación de cada segmento.




- Tendremos **una tabla de páginas por cada segmento**, **cada entrada de la tabla de segmentos apunta a su tabla de páginas**.

# 5. Modelos avanzados de traducción de direcciones

## 5.3. Métodos combinados

**Tabla de segmentos**

- El SO mantiene **una tabla de segmentos por proceso** cuyo objetivo es almacenar la información necesaria para localizar las tablas de páginas.
- La estructura de esta tabla es:

| | @Base | límite |
|---|---|---|
| 0 | | |
| 1 | | |
| .. | | |
| N | | |

- **@Base** es la **dirección física donde está la tabla de páginas del segmento.**
- **Límite** es el número de posiciones del segmento (realmente se pone el número de posiciones del segmento -1 como veremos más adelante).

# 5. Modelos avanzados de traducción de direcciones

## 5.3. Métodos combinados

**Tabla de páginas**

- El SO mantiene **una tabla de páginas por cada segmento** que forme el proceso, su objetivo es almacenar a qué marcos están asignadas las páginas.
- Esta tabla está indexada por páginas.
- La estructura de esta tabla es la siguiente:

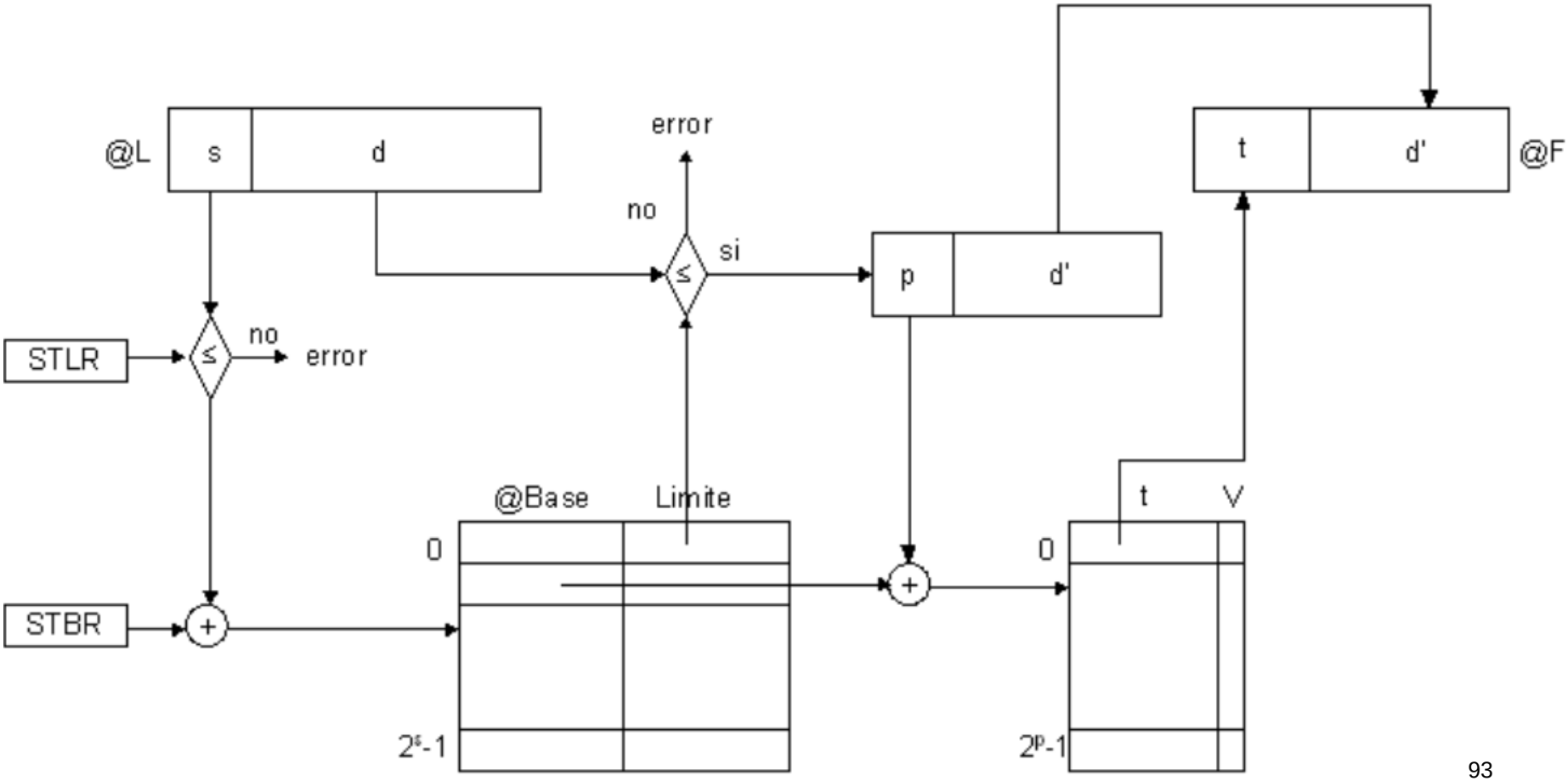
| | t | BV |
|---|---|---|
| 0 | | |
| 1 | | |
| .. | | |
| N | | |

- **t** es el número marco que contiene la página.
- **BV** es un Bit de validez

# 5. Modelos avanzados de traducción de direcciones

## 5.3. Métodos combinados

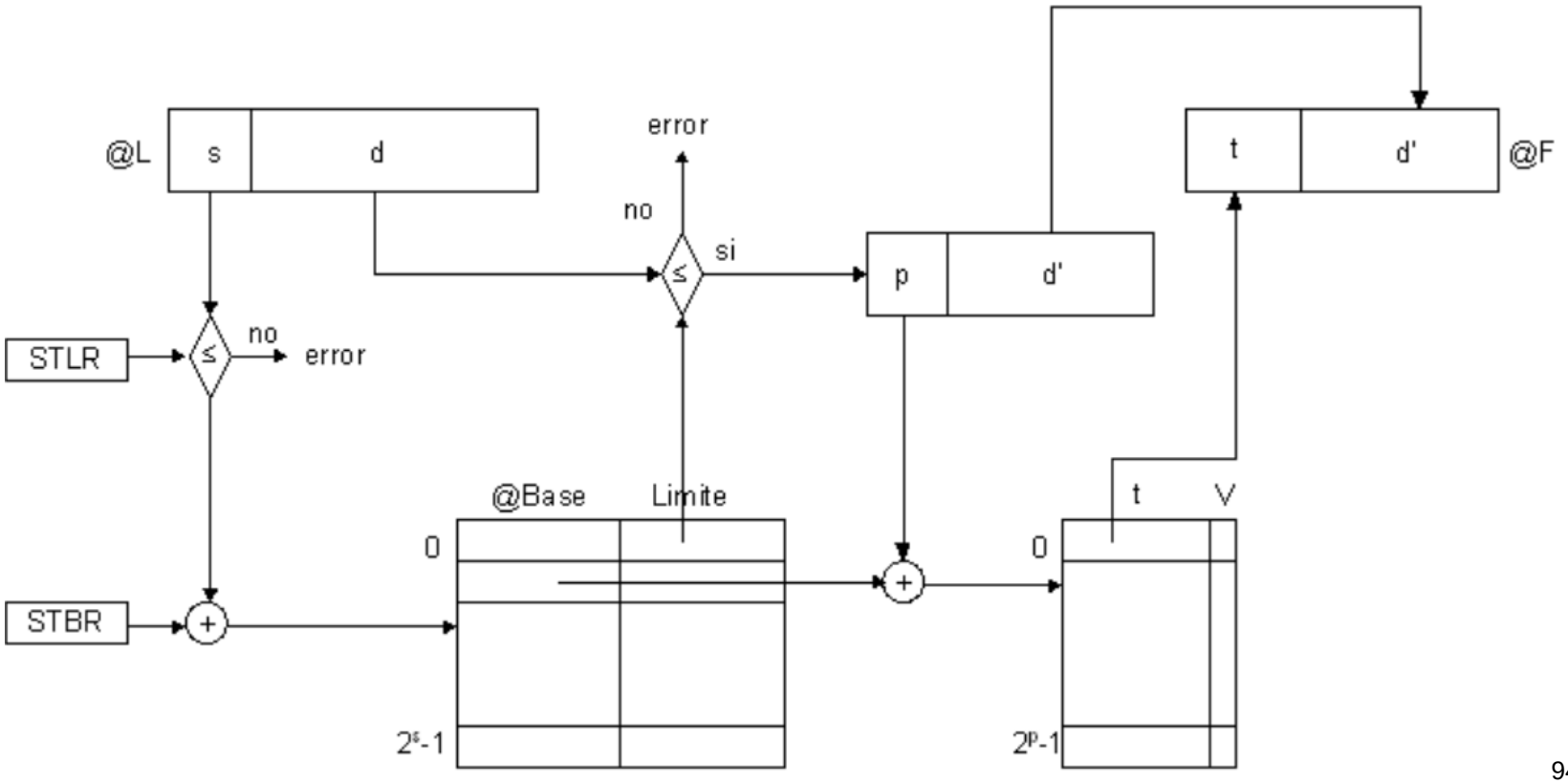
- El esquema de traducción de una instrucción es:



# 5. Modelos avanzados de traducción de direcciones

## 5.3. Métodos combinados

- **Ejemplo 1:** Si tenemos una MMU que trabaja con segmentación paginada, direcciones lógicas de 28 bits, la memoria principal es de 16 Mb, el tamaño de la página es de 1 Kb y el tamaño máximo de un segmento es 1 Mb. Indicar la longitud en bits que se necesitan para los distintos elementos de la MMU:



# 5. Modelos avanzados de traducción de direcciones

## 5.3. Métodos combinados

- Si MP es de 16 Megas ⇒ el tamaño de las direcciones físicas y del campo @Base de la tabla de segmentos es:

  $@Base = t + d' = 24$.
- Si el tamaño máximo de una página es 1Kbyte el número de bits que necesitamos para referenciar a cualquier posición de la página es:

  $d' = 10$
- Si el tamaño máximo de un segmento es 1 Mega el número de bits que necesitamos para referenciar a cualquier posición del segmento es:

  $limite = d = 20$
- A partir de estos valores obtenemos el resto:

  $s = 28 - d = 28 - 20 = 8$  $t = @base - d' = 24 - 10 = 14$  $p = d - d' = 10$
- Como el STBR apunta a cualquier dirección de memoria y nuestra memoria es de 16 Mb, este registro mide 24 bits.

# 5. Modelos avanzados de traducción de direcciones

## 5.3. Métodos combinados

***Ventajas***

- *Protección de segmentos sin fragmentación externa.*

***Inconvenientes***

- *Necesitamos tres accesos a memoria.*
- *Puede producir fragmentación interna.*

# Índice

# 6. Memoria Virtual

- Todos los esquemas vistos hasta ahora exigen que el programa esté entero en memoria.
- El principio de cercanía indica que probablemente cuando se ejecute un programa nunca se vaya a ejecutar todo su código.
- Si tenemos esquemas con:
  - Reubicación dinámica.
  - División del espacio lógico del proceso.

# 6. Memoria Virtual

- No es necesario tener todos los fragmentos del programa en memoria principal para ejecutarse.

**Ventajas**

- El tamaño del proceso no está limitado por la memoria.
- Aumenta el grado de *multiprogramación.*
- Es más rápido el inicio de las aplicaciones.
- El programador se despreocupa del tamaño de memoria.

**Desventajas**

- Se producen retrasos en la ejecución de los procesos.
- Requiere un hardware especial.
- Requiere apoyo software.

# 6. Memoria Virtual

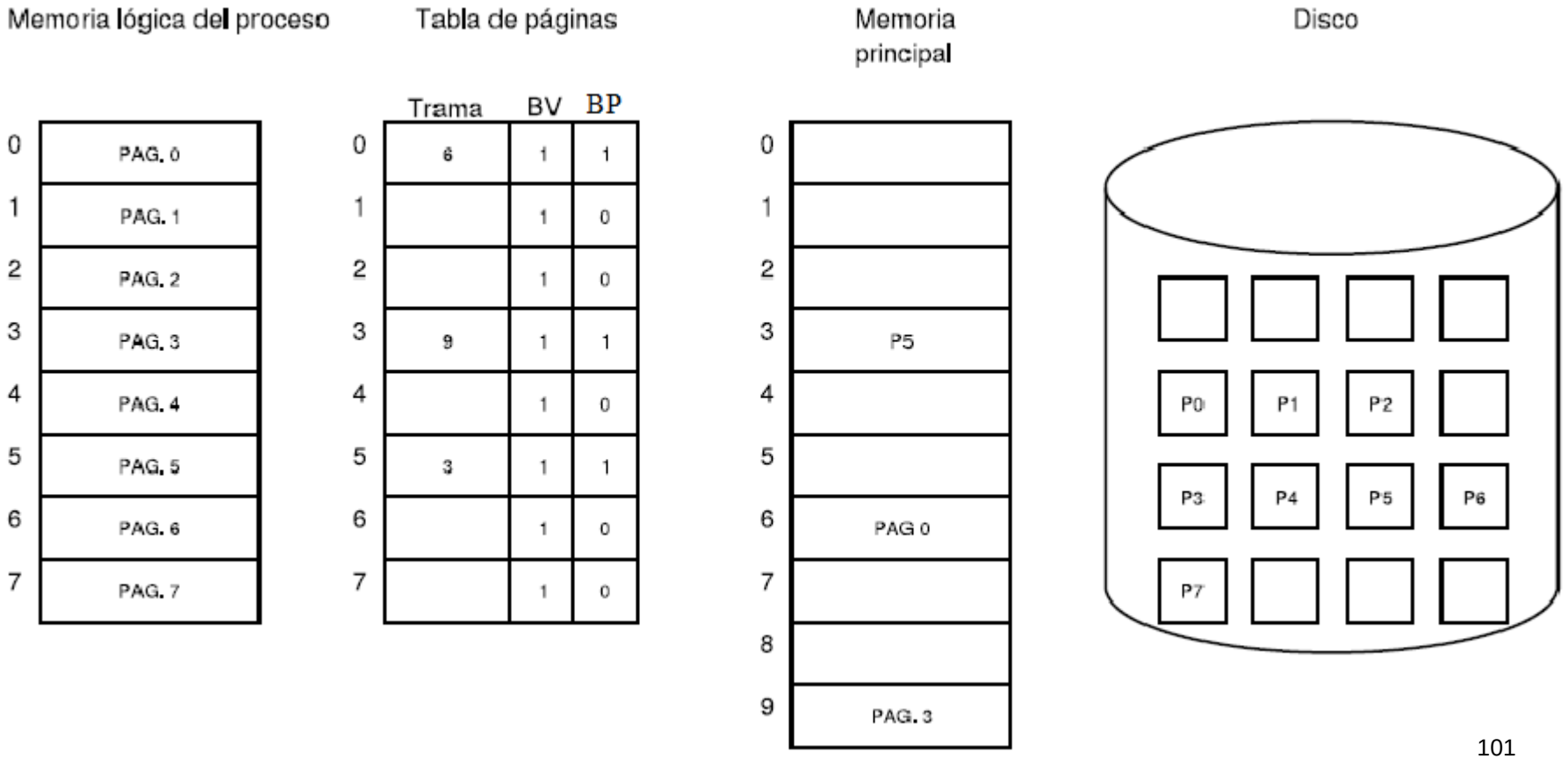
## 6.1. Paginación por demanda

- ¿Cuándo se debe cargar una página en la memoria principal?:
  - **Paginación por demanda pura**, se trae una página sólo cuando se hace referencia a una posición en dicha página.
  - **Paginación previa**, se carga la página que se necesita y algunas más.

# 6. Memoria Virtual

## 6.1. Paginación por demanda

- **Ejemplo**: Estado típico de la memoria en un gestor de paginación por demanda.



# 6. Memoria Virtual

## 6.1. Paginación por demanda

- Realmente en los SO actuales se usa el mismo bit de validez para indicar presencia:
  - Si el BV vale 1 sabemos que la página está presente.
  - Si el BV vale 0 la página no está en memoria principal pero no sabemos si la página es válida. En el PCB del proceso se guardará información sobre las páginas que son válidas para el proceso.
  - Cuando se intente acceder a alguna página no cargada se produce un fallo de página.

# 6. Memoria Virtual

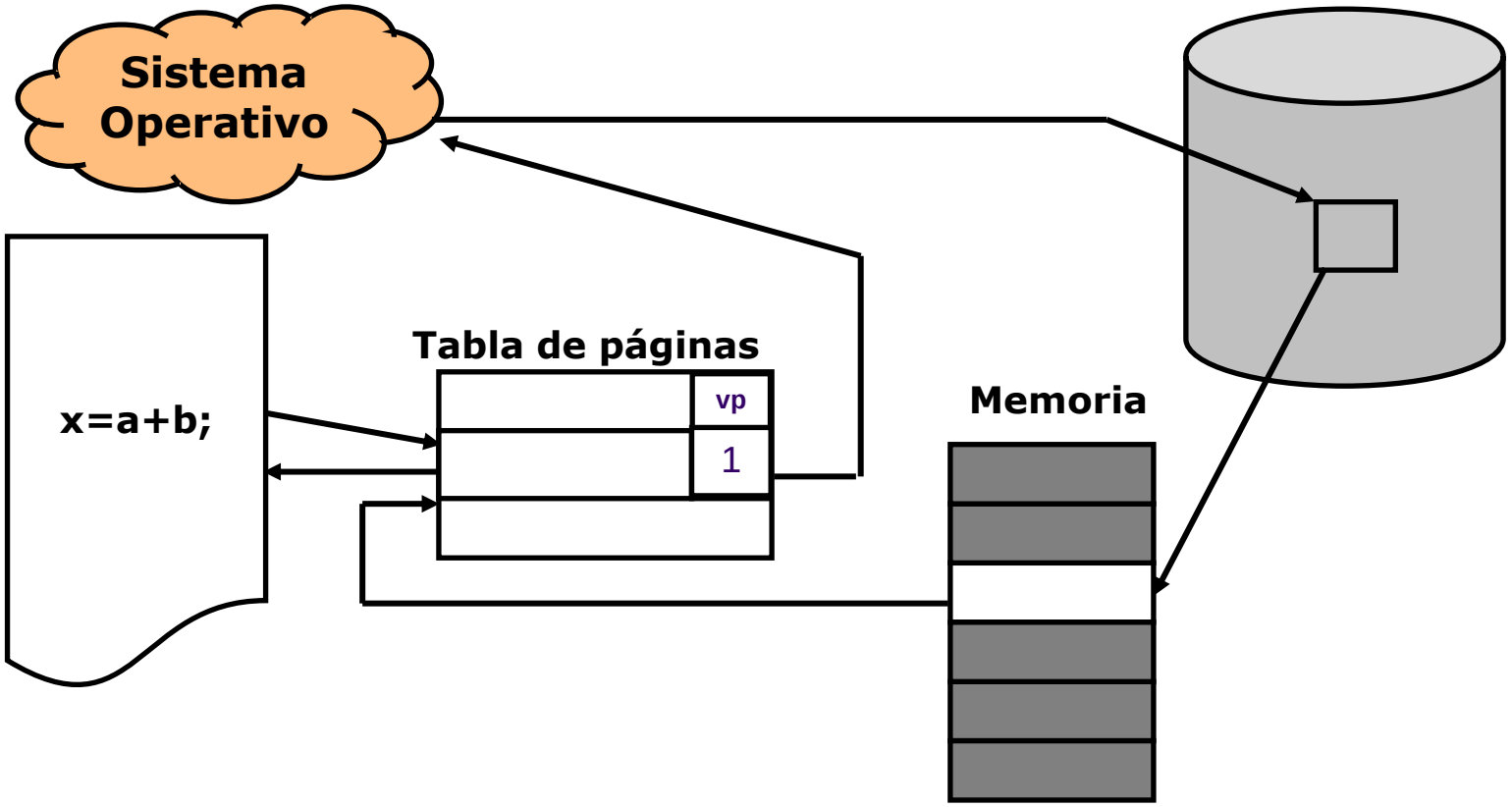
## 6.1. Paginación por demanda

¿Qué pasos se realizan ante un fallo de página?

- Se detecta si el error es realmente un fallo de página o una dirección inválida.
- Si es una dirección inválida finalizo.
- Si es por fallo de página, la buscamos en memoria secundaria.
- Le asignamos un marco de memoria principal.
- Cargamos el contenido de la página en el marco.
- Ponemos el bit de presencia de la tabla a 1 para indicar que la página está en memoria.
- Reiniciamos la ejecución de la instrucción.

# 6. Memoria Virtual

## 6.1. Paginación por demanda



# 6. Memoria Virtual

## 6.2. Hardware necesario

- Para la traducción necesitamos añadir a la tabla de páginas un bit de presencia **BP** para detectar las páginas cargadas en memoria.

| | t | BV | **BP** |
|---|---|---|---|
| 0 | | | |
| 1 | | | |
| | | | |
| N | | | |

- Un dispositivo auxiliar de alta velocidad. Además las operaciones de E/S se hacen mediante controladores DMA (Acceso Directo a Memoria).
- Hardware necesario para distinguir entre fallo de página y la interrupción por direccionamiento incorrecto.
- Hardware para poder reiniciar la instrucción interrumpida.

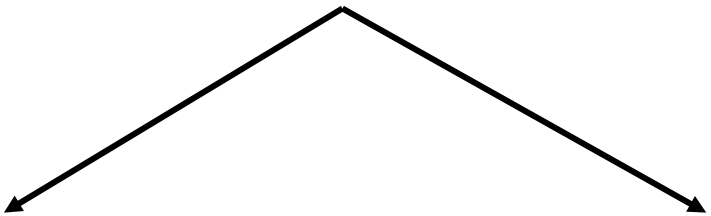
# 6. Memoria Virtual

## 6.3. Reemplazo de páginas

**Problema de la Memoria Virtual:** El rendimiento decae cuando hay que cargar páginas.

**Problema adicional:** Si no hay marcos libres hay que desalojar alguna página.

**¿Qué página desalojar?**



Suspender un proceso

(Se desalojan todos sus marcos. Reduce el grado de multiprogramación)

Reemplazar una página

(Hay que seleccionar la página a desalojar)

# 6. Memoria Virtual

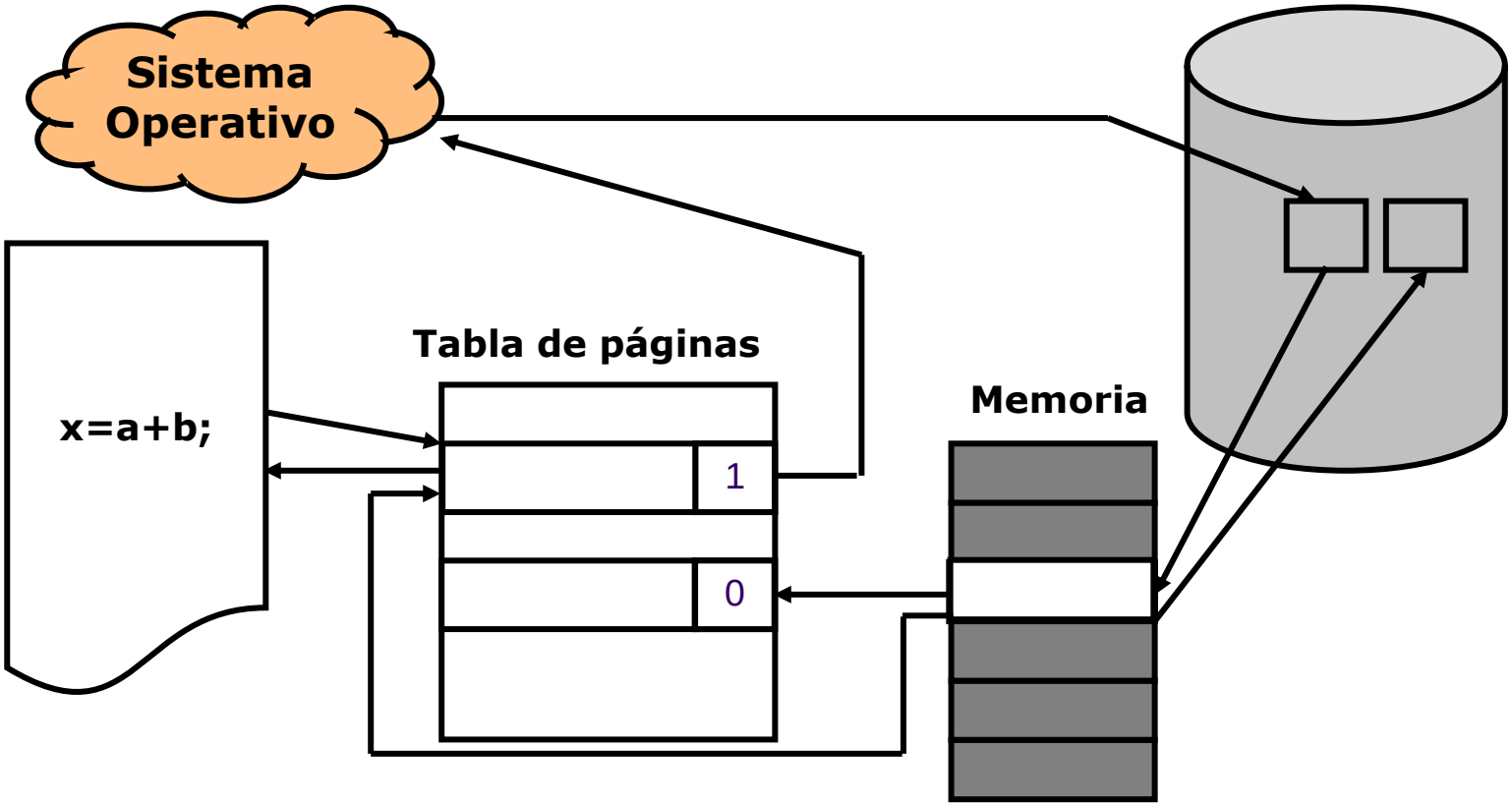
## 6.3. Reemplazo de páginas

**La rutina de atención a la interrupción de Fallo de Página queda:**

1. Encontrar la página a cargar en el disco.
2. Buscar un marco libre.
   - Si hay marco libre, usarlo.
   - Si no:
     - Usar un algoritmo de reemplazo para seleccionar la página a desalojar.
     - Guardar la página en el disco y modificar la tabla de páginas y la tabla de marcos libres.
3. Cargar la página en el nuevo marco libre.
4. Modificar la tabla de páginas y la tabla de marcos libres.
5. Reiniciar la instrucción interrumpida.

# 6. Memoria Virtual

## 6.3. Reemplazo de páginas



# 6. Memoria Virtual

## 6.3. Reemplazo de páginas

**Problema:** Hay que realizar dos transferencias a disco, una de descarga y otra de carga.

**Solución parcial:** Utilizar el *bit de modificada* de la tabla de páginas.

| | t | BV | BP | **BM** |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| | | | | |
| N | | | | |

**Hay que resolver**

**¿Qué criterios utilizar para elegir la página que debe ser reemplazada?**

**¿Cuántos marcos asignar a cada proceso para su ejecución?**

# 6. Memoria Virtual

## 6.4. Algoritmos de reemplazo de páginas

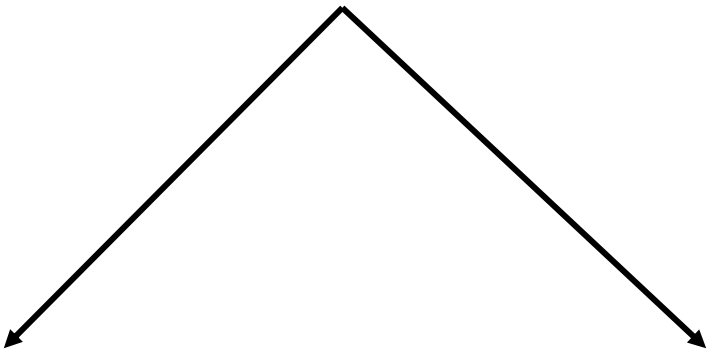
- Algoritmo Óptimo.
- Algoritmo FIFO (First In - First Out).
- Algoritmo LRU (Least Recently Used).
- Algoritmos de aproximación a LRU.
- Algoritmos de conteo.
- Algoritmos de almacenamiento intermedio de páginas.

# 6. Memoria Virtual

## 6.4. Algoritmos de reemplazo de páginas

**Criterio:** Seleccionar aquel algoritmo que minimice la frecuencia de fallos de página.

**Evaluación:** Se ejecuta cada algoritmo para una *serie de referencias* a memoria y se cuentan los fallos de página que produce.



**Generada artificialmente**

**Rastrear el sistema anotando las direcciones de memoria accedidas**

# 6. Memoria Virtual

## 6.4. Algoritmos de reemplazo de páginas

*100, 135, 180, 432, 101, 612, 524, 420, 003, 650, 108, 120*

*1, 1, 1, 4, 1, 6, 5, 4, 0, 6, 1, 1*

*1, 4, 1, 6, 5, 4, 0, 6, 1*

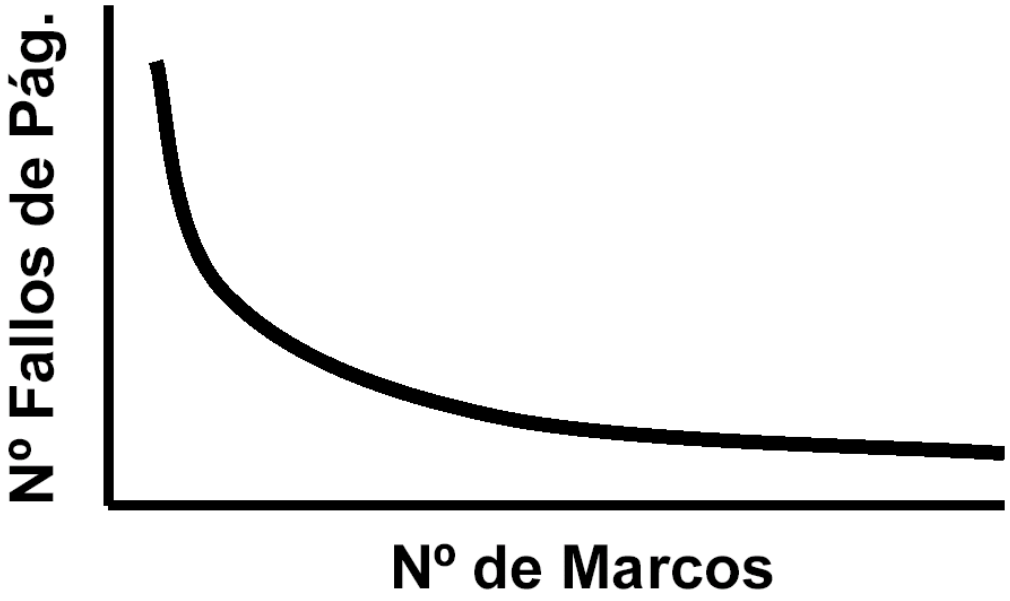
***Serie de referencias:***

***7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1***

# 6. Memoria Virtual

## 6.4. Algoritmos de reemplazo de páginas

**Número de marcos de página:** **3**



# 6. Memoria Virtual

## 6.4.1. Algoritmo óptimo

La página a reemplazar será aquella que tardará más tiempo en volver a referenciarse.

**Ventajas**

- Provoca el mínimo número de fallos de página.
- Se usa como patrón para comparar con el resto.

**Desventaja**

- Imposible de llevar a la práctica.

# 6. Memoria Virtual

## 6.4.1. Algoritmo óptimo

| | * | * | * | * | | * | | * | | * | | * | | | | * | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 7 | 7 | 7 | **2** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | **7** | 7 | 7 |
| | | 0 | 0 | 0 | 0 | 0 | 0 | **4** | 4 | 4 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | 1 | 1 | 1 | **3** | 3 | 3 | 3 | 3 | 3 | 3 | 3 | **1** | 1 | 1 | 1 | 1 | 1 |
| | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **1** | **7** | **0** | **1** |

Número de Fallos de Página: 9

Número de Reemplazos: 6

# 6. Memoria Virtual

## 6.4.2. Algoritmo FIFO

La página a reemplazar será aquella que hace más tiempo que ha sido **cargada** en memoria.

**Ventajas**

- Fácil de programar.
- Provoca poca sobrecarga.

**Desventajas**

- Es poco eficiente.
- Presenta la *anomalía de Belady.*

# 6. Memoria Virtual

## 6.4.2. Algoritmo FIFO

| | * | * | * | * | | * | * | * | * | * | * | | | * | * | | | | * | * | * |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 7 | 7 | 7 | **2** | 2 | 2 | 2 | **4** | 4 | 4 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | **7** | 7 | 7 |
| | | 0 | 0 | 0 | 0 | **3** | 3 | 3 | **2** | 2 | 2 | 2 | 2 | **1** | 1 | 1 | 1 | 1 | **0** | 0 |
| | | | 1 | 1 | 1 | 1 | **0** | 0 | 0 | **3** | 3 | 3 | 3 | 3 | **2** | 2 | 2 | 2 | 2 | **1** |
| | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **1** | **7** | **0** | **1** |

Número de Fallos de Página: 15

Número de Reemplazos: 12

# 6. Memoria Virtual

## 6.4.2. Algoritmo FIFO

**Anomalía de Belady:** El aumento de marcos en la memoria no siempre lleva consigo una disminución en el número de fallos de página que producen algunos algoritmos.

*1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5*

**Para tres marcos de página: 9 Fallos de página**

| | * | * | * | * | * | * | * | | | * | * |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 3 |
| | | | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 4 | 4 |
| | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |

# 6. Memoria Virtual

## 6.4.2. Algoritmo FIFO

**Para cuatro marcos de página: 10 Fallos de página**

| * | * | * | * | | | * | * | * | * | * | * |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 4 | 4 |
| | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 5 |
| | | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 |
| | | | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 |
| 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |

# 6. Memoria Virtual

## 6.4.3. Algoritmo LRU (Least Recently Used)

Se basa en la consideración de que aquellas páginas muy usadas en el pasado reciente lo serán también en el futuro. Por ello, en este algoritmo se propone reemplazar la página que hace más tiempo que no se ha usado, la ***menos usada recientemente*****.**

**Ventajas**

- ❑ Presenta buen rendimiento.
- ❑ No sufre la *anomalía de Belady.*

**Desventaja**

- ❑ Necesita apoyo del hardware.

# 6. Memoria Virtual

## 6.4.3. Algoritmo LRU (Least Recently Used)

| | * | * | * | * | | * | | * | * | * | * | | | * | | * | | * | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 7 | 7 | 7 | **2** | 2 | 2 | 2 | **4** | 4 | 4 | **0** | 0 | 0 | **1** | 1 | 1 | 1 | 1 | 1 | 1 |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **3** | 3 | 3 | 3 | 3 | 3 | **0** | 0 | 0 | 0 | 0 |
| | | | 1 | 1 | 1 | **3** | 3 | 3 | **2** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | **7** | 7 | 7 |
| | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **1** | **7** | **0** | **1** |

Número de Fallos de Página: 12

Número de Reemplazos: 9

# 6. Memoria Virtual

## 6.4.3. Algoritmo LRU (Least Recently Used)

Algunas formas de implementarlo pueden ser:

- **Contadores Hardware**. El sistema operativo mantiene un contador, y anota en la Tabla de Páginas el valor del contador al usarla. Se reemplaza la de contador más bajo.
- **Pilas**. Se mantiene una pila de los números de las páginas utilizadas. Cada vez que se usa una página, su número se apila y se saca de donde estuviese. La página menos recientemente utilizada será la del fondo de la pila.
- **Matrices Hardware**. Para N marcos, se mantiene una matriz de NxN bits inicializados a cero. Al acceder al marco K, el hardware pone a 1 todos los bits de la fila K y a 0 todos los de la columna K. La página menos recientemente usada es aquella cuya fila tiene un valor binario menor.

# 6. Memoria Virtual

## 6.4.3. Algoritmo LRU (Least Recently Used)

Ejemplo de implementación LRU con matrices Hardware para las referencias: ***0, 1 ,2 ,3 ,2 ,1 ,0 ,3 ,2 ,3*** y 4 marcos.

Página

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |

(a)

Página

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

(b)

Página

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 |

(c)

Página

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

(d)

Página

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |

(e)

| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |

(f)

| 0 | 1 | 1 | 1 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |

(g)

| 0 | 1 | 1 | 0 |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |

(h)

| 0 | 1 | 0 | 0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |

(i)

| 0 | 1 | 0 | 0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

(j)

# 6. Memoria Virtual

## 6.4.4. Otros algoritmos (Aproximaciones al LRU)

La tabla de páginas suele tener dos bits muy útiles para los algoritmos de reemplazo.

- Bit de referenciada. Se pone a 1 cada vez que se accede a dicha página. Periódicamente el sistema operativo la pone a 0.
- Bit de modificada. Se pone a 1 cada vez que se escribe en dicha página.

# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. NRU (Not Recently Used)

Tomando en cuenta el bit de referenciada y el bit de modificada, cada una de las páginas que están en memoria se pueden asignar a una de estas 4 categorías:

| Grupo | R | M | |
|---|---|---|---|
| 0 | 0 | 0 | Ni referenciada ni modificada recientemente. |
| 1 | 0 | 1 | Modificada hace tiempo. |
| 2 | 1 | 0 | Se usó recientemente pero no se modificó. |
| 3 | 1 | 1 | Se modificó recientemente. |

Se selecciona para el reemplazo alguna de las páginas del grupo 0, de no existir se busca entre las del grupo 1, y así sucesivamente.

# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. NRU (Not Recently Used)

**Ejemplo:** W=escritura, en caso de igualdad eliminamos la que lleve más tiempo sin referenciarse

| * | * | * | ** | | ** | | ** | ** | ** |
|---|---|---|---|---|---|---|---|---|---|
| **7(1,0)** | 7 (1,0) | 7 (1,0) | **2(1,0)** | 2 (1,0) | 2 (1,0) | 2 (1,0) | **4(1,1)** | 4 (1,1) | 4 (1,1) |
| | **0 (1,0)** | 0 (1,0) | 0 (1,0) | **0 (1,1)** | 0 (1,1) | **0 (1,1)** | 0 (1,1) | 0 (1,1) | 0 (1,1) |
| | | **1 (1,0)** | 1 (1,0) | 1 (1,0) | **3 (1,0)** | 3 (1,0) | 3 (1,0) | **2 (1,0)** | **3 (1,0)** |
| 7 | 0 | 1 | 2 | W 0 | 3 | 0 | W 4 | 2 | 3 |

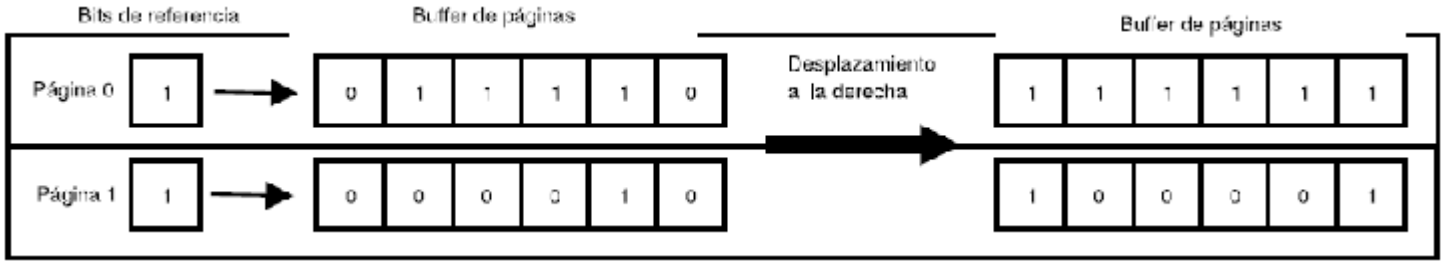
| | | ** | ** | ** | | ** | ** | | ** |
|---|---|---|---|---|---|---|---|---|---|
| 4 (1,1) | 4(1,1) | 4 (1,1) | 4 (1,1) | 4 (1,1) | 4 (1,1) | 4 (1,1) | 4 (1,1) | 4 (1,1) | 4 (1,1) |
| **0 (1,1)** | 0 (1,1) | 0 (1,1) | 0 (1,1) | 0 (1,1) | **0 (1,1)** | 0 (1,1) | 0 (1,1) | **0 (1,1)** | 0 (1,1) |
| 3 (1,0) | **3 (1,0)** | **2 (1,0)** | **1 (1,0)** | **2 (1,0)** | 2 (1,0) | **1 (1,0)** | **7 (1,0)** | 7 (1,0) | **1 (1,0)** |
| W 0 | 3 | 2 | 1 | 2 | W 0 | 1 | 7 | W 0 | 1 |

**14 Fallos, 11 Reemplazos**

# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. Envejecimiento (AGING)

- Se tienen tantos buffer como páginas.
- En estos se irán almacenando el histórico de bits de referencia de las páginas.
- ¿Cuándo se actualiza el contenido? Cada tic de reloj.
- ¿Cómo se actualiza? En cada uno de estos buffer se almacena el bit de referencia de la página correspondiente. El bit que entra ocupa la posición más significativa y se desplaza hacia la derecha a los bits ya existente.

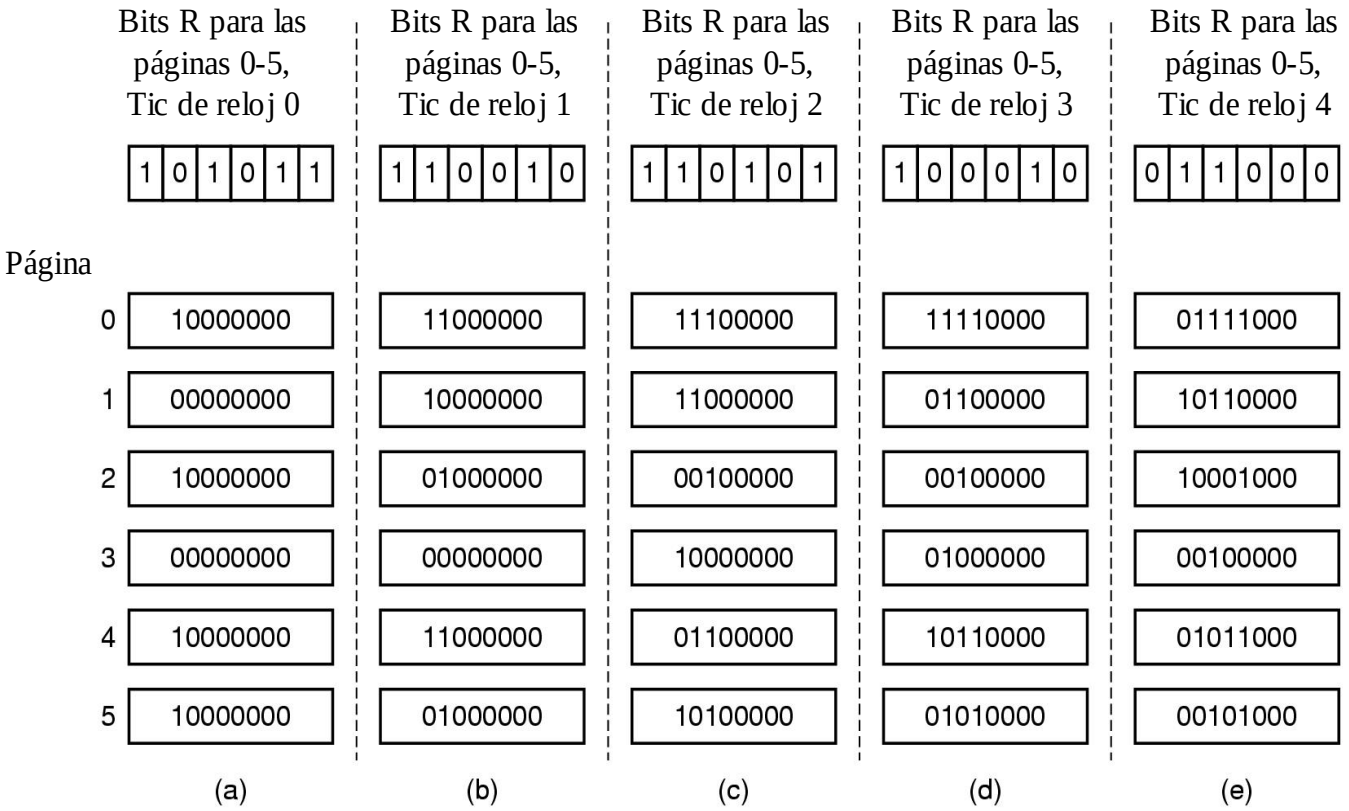



- Cuando sea necesario reemplazar una página se seleccionará aquella cuyo buffer tenga un número en binario menor.

# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. Envejecimiento (AGING)

Ejemplo 1: funcionamiento del algoritmo de envejecimiento.



# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. Envejecimiento (AGING)

**Ejemplo 2:** AGING con 3 bits

| * | * | * | ** | | ** | | ** | ** | ** | ** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **7** | 7 | 7 | **2** | 2 | 2 | 2 | **4** | 4 | 4 | **0** | 0 |
| | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **3** | 3 | 3 |
| | | **1** | 1 | 1 | **3** | 3 | 3 | **2** | 2 | 2 | 2 |
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 |

| | ** | | ** | | ** | | |
|---|---|---|---|---|---|---|---|
| 0 | **1** | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 3 | 3 | **0** | 0 | 0 | 0 | 0 |
| 2 | 2 | 2 | 2 | 2 | **7** | 7 | 7 |
| 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |

Estado de los buffers (se rellena de derecha a izquierda):

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | | | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 2 | | | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. Algoritmo del reloj

Es también llamado de **segunda oportunidad**, y básicamente es un algoritmo FIFO, donde una vez seleccionada la página se examina su bit de referenciada.

Si el bit de referenciada de la página es 0, entonces se reemplaza la página, pero si es 1 se le da a esa página una segunda oportunidad y se selecciona la siguiente página FIFO.

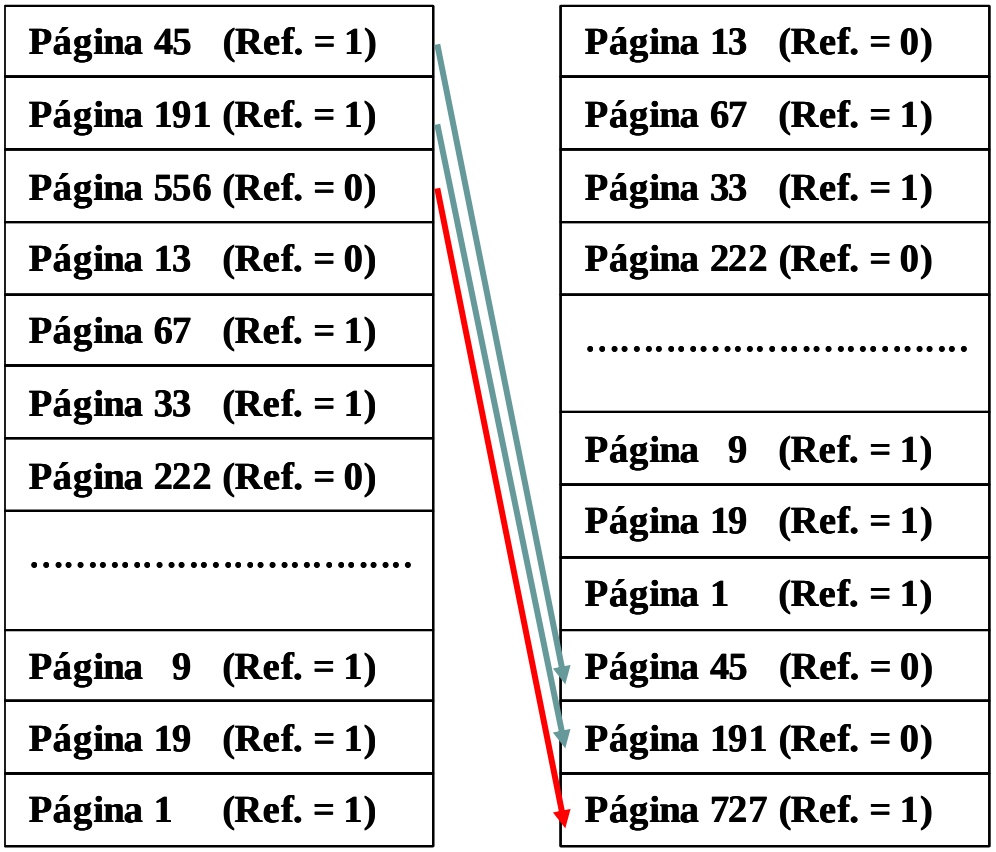
Cuando se le da a la página una segunda oportunidad, su bit de referenciada se pone a 0, y se coloca dicha página como la última de la cola FIFO.

En este algoritmo, si una página es usada con la suficiente frecuencia como para mantener su bit de referenciada a 1, nunca será reemplazada.

# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. Algoritmo del reloj

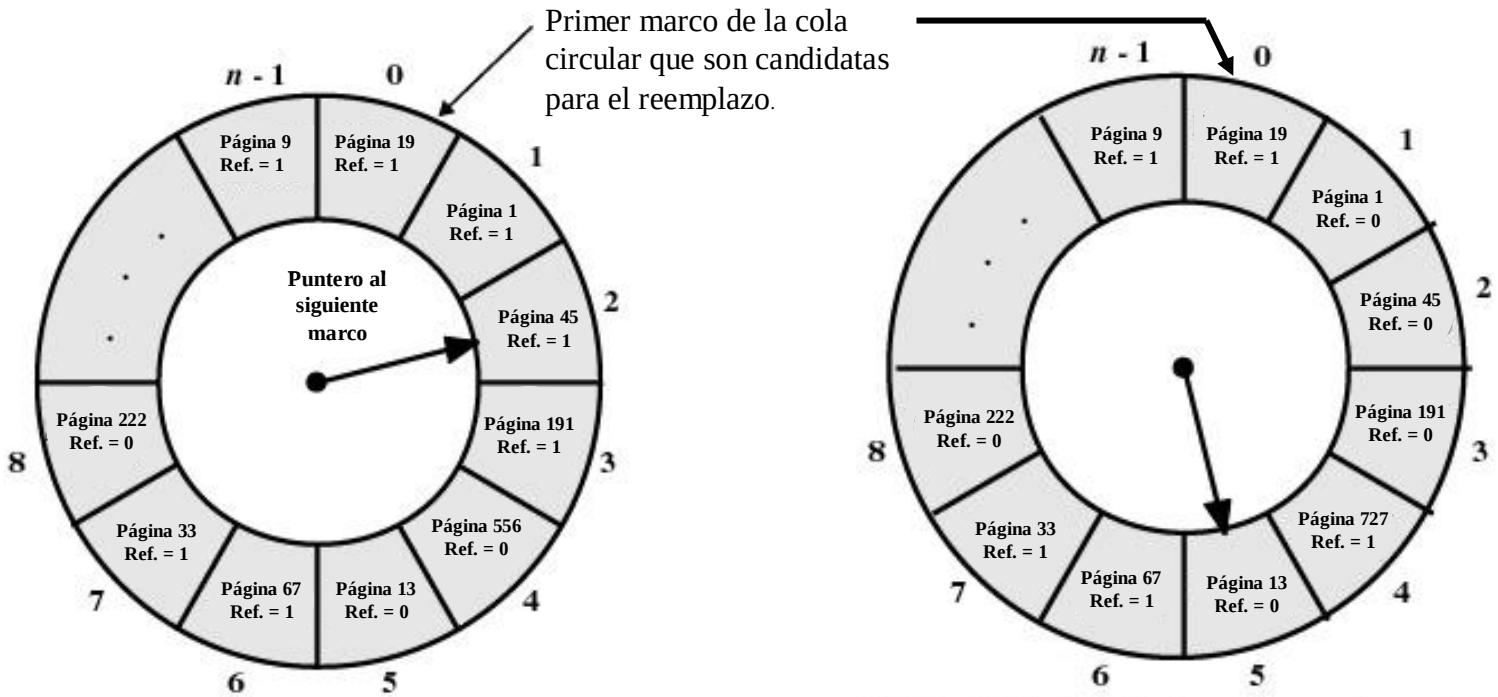
Ejemplo de algoritmo del reloj: debe cargarse la página 727.



# 6. Memoria Virtual

## 6.4.4. Aproximaciones al LRU. Algoritmo del reloj

Ejemplo: Uso de un cola circular. Debe cargarse la página 727.



# 6. Memoria Virtual

## 6.4.5. Algoritmos de conteo

Se basan en el **número de veces** que se referencia una página

- **LFU (Least Frequently used):** En el algoritmo de reemplazo de páginas **menos frecuentemente usada** abandonará la memoria aquella página que tenga el contador más bajo.

- **MFU (Most Frequently used):** En el algoritmo de reemplazo de páginas **más frecuentemente usada** abandonará la memoria aquella página que tenga el contador más alto. (Asume que las de contador bajo se acaban de cargar y se van a necesitar en el futuro).

# 6. Memoria Virtual

## 6.4.5. Algoritmos de conteo

**LFU** **Ejemplo:** Suponemos que a igualdad de referencias reemplazamos la que lleva más tiempo:

| * | * | * | ** | | ** | | ** | ** | ** |
|---|---|---|---|---|---|---|---|---|---|
| **7 (1)** | 7 (1) | 7 (1) | **2 (1)** | 2 (1) | 2 (1) | 2 (1) | **4 (1)** | 4 (1) | **3 (1)** |
| | **0 (1)** | 0 (1) | 0 (1) | 0 (2) | 0 (2) | **0 (3)** | 0 (3) | 0 (3) | 0 (3) |
| | | **1 (1)** | 1 (1) | 1 (1) | **3 (1)** | 3 (1) | 3 (1) | **2 (1)** | 2 (1) |
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 |

| | | | ** | | | | ** | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 (1) | **3 (2)** | 3 (2) | **1 (1)** | 1 (1) | 1 (1) | **1 (2)** | **7 (1)** | 7 (1) | **1 (1)** |
| **0 (4)** | 0 (4) | 0 (4) | 0 (4) | 0 (4) | **0 (5)** | 0 (5) | 0 (5) | **0 (6)** | 0 (6) |
| 2 (1) | 2 (1) | **2 (2)** | 2 (2) | **2 (3)** | 2 (3) | 2 (3) | 2 (3) | 2 (3) | 2 (3) |
| 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |

**10 Fallos, 7 Reemplazos**

# 6. Memoria Virtual

## 6.5. Políticas de asignación de tramas

- La política por demanda de páginas no se aplica tal cual porque puede dar lugar a problemas de **hiperpaginación** (thrashing).

- La hiperpaginación es el aumento incontrolado de faltas de página que se produce cuando un proceso no tiene asignadas el número de marcos suficiente para su correcta ejecución. Lleva a graves problemas de rendimiento.

- En la práctica un proceso que se vaya a ejecutar se quedará con un mínimo de tramas.

- Las políticas de asignación de tramas calculan cuál debe de ser ese mínimo.

# 6. Memoria Virtual

## 6.5. Políticas de asignación de tramas

- Las políticas de asignación de tramas se dividen en dos grandes grupos:
  - **Fijas:** Asignación de un número fijo de tramas a cada proceso
  - **Variables:** Asignan un número variables de tramas a cada proceso
    - Locales
    - Globales

# 6. Memoria Virtual

## 6.5.1. Políticas de asignación de tramas fijas

### Equipartición

- Se reparte el número de tramas de la memoria por proceso (NTP) de forma equitativa entre todos los procesos en ejecución:

$$NTP = Tramas\ totales / Num\ procesos$$

### Proporcional

- Mayor número de tramas mientras mayor sea el proceso.

### Prioritaria

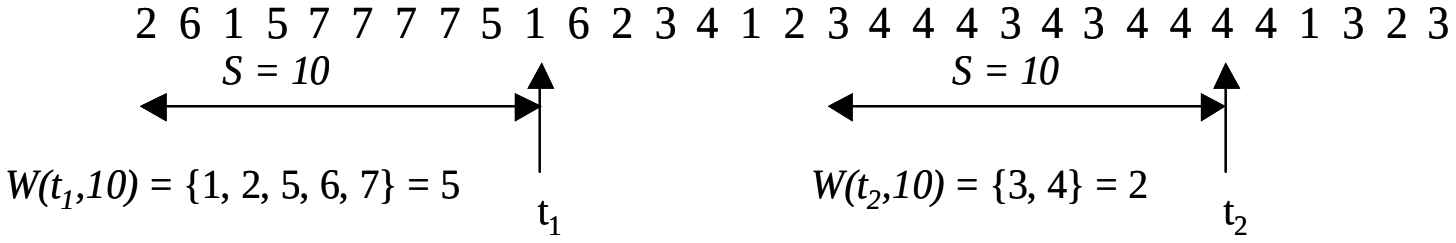
- Cuanto más prioritario es el proceso mayor número de tramas se asignan.

# 6. Memoria Virtual

## 6.5.2. Políticas de asignación de tramas variable

**Working Set** (Conjunto de trabajo):El **conjunto de trabajo** de un proceso en el instante virtual t, y con un parámetro *S*, denotado por *W(t,S)*, es el conjunto de páginas, a las que el proceso ha hecho referencia en las *S* unidades de tiempo virtual.

**Tiempo virtual**, indica el tiempo que transcurre mientras el proceso está realmente en ejecución (lo podemos considerar medido por ciclos de instrucción). El tiempo virtual se considera medido en ciclos de instrucción.



# 6. Memoria Virtual

## 6.5.2. Políticas de asignación de tramas variable

Con este concepto podemos usar la siguiente **estrategia** para el control del conjunto residente:

1) Supervisar el conjunto de trabajo de cada proceso.
2) Eliminar periódicamente del conjunto residente de un proceso, aquellas páginas que no pertenezcan a su conjunto de trabajo.
3) Un proceso se puede ejecutar sólo si su conjunto de trabajo está en memoria principal, es decir, si su conjunto residente incluye su conjunto de trabajo.

# 6. Memoria Virtual

## 6.5.2. Políticas de asignación de tramas variable

Esta estrategia se basa en controlar la **frecuencia de fallos de página** de cada proceso. Busca aproximarse al conjunto de trabajo.

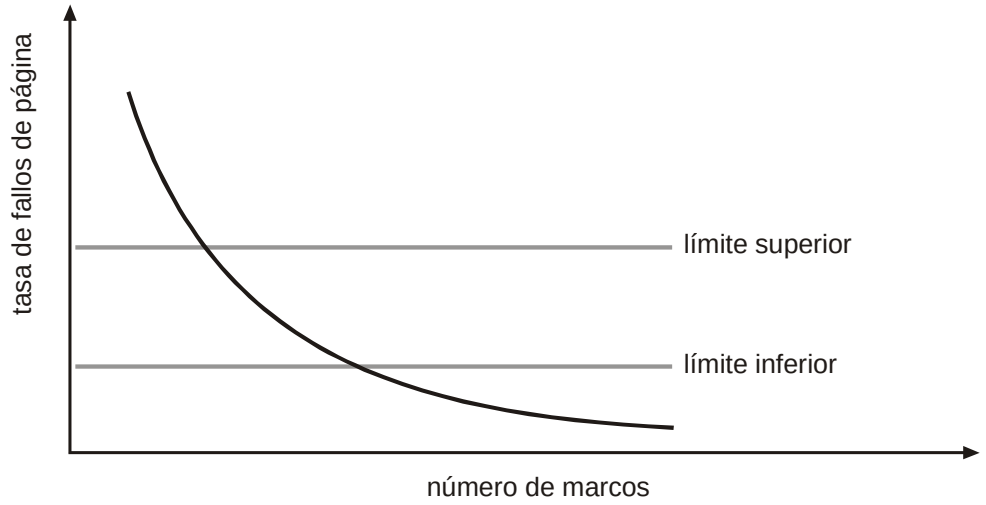
Se define un umbral F, de número de fallos de página por unidad de tiempo:

- Si el número de fallos de página es mayor que F, entonces la página se añade al conjunto residente del proceso. (Si no hay marcos libres se suspende algún proceso, liberando sus marcos de página).
- En otro caso, se descartan todas las páginas con bit de referenciada a 0, reduciendo por tanto el conjunto residente. Al mismo tiempo se ponen a 0 todos los bits de referenciada de todas las páginas cargadas.

# 6. Memoria Virtual

## 6.5.2. Políticas de asignación de tramas variable

La estrategia puede mejorarse usando dos umbrales, uno superior, para activar el crecimiento del conjunto residente, y otro inferior para hacerlo decrecer.




**Problema**: No rinde adecuadamente durante los periodos transitorios, cuando el proceso está cambiando de conjunto de referencias.

# 6. Memoria Virtual

## 6.6 Hiperpaginación

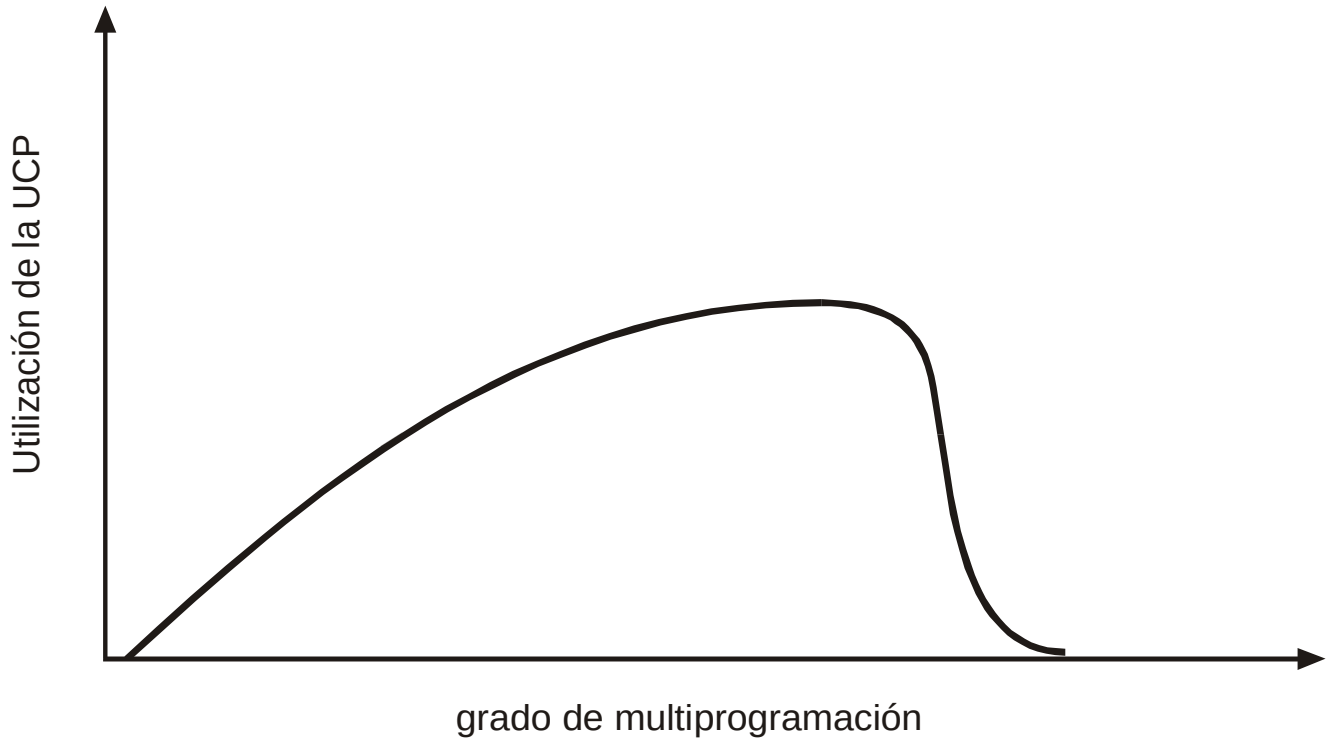
La **hiperpaginación** (thrashing) ocurre cuando el número de marcos asignados a un proceso no es suficiente para almacenar las páginas referenciadas activamente por el mismo, y se producirá un número elevado de fallos de página.

Dependiendo del tipo de asignación afecta a:

- Sólo un proceso, si la asignación es fija.
- Todo el sistema, si la asignación es variable y el grado de multiprogramación provoca que el *conjunto residente* de los procesos sea menor que su *conjunto de trabajo*.

# 6. Memoria Virtual

## 6.6 Hiperpaginación



# Índice

# 7. Casos de estudio

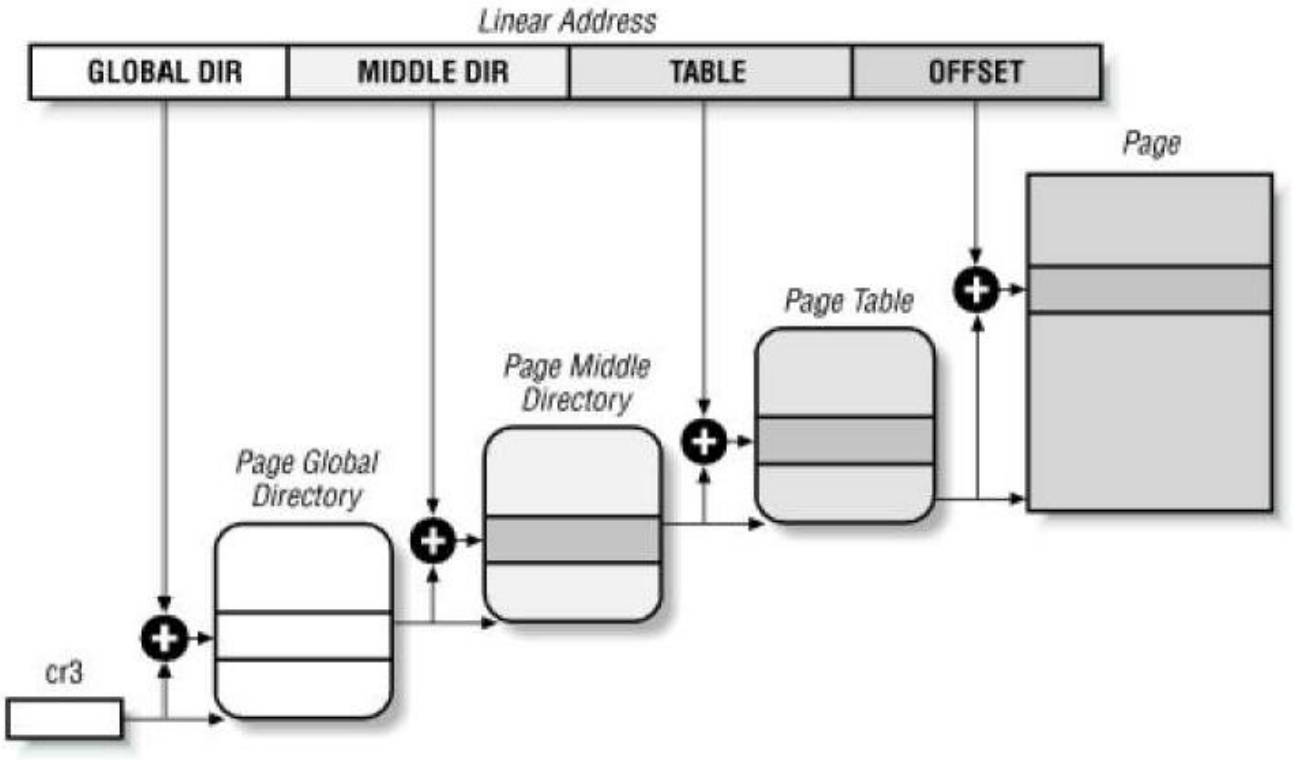
## 7.1. Linux

- Se utiliza Memoria Virtual Paginada.
- Linux hace un uso limitado de la segmentación (sólo las usa en arquitecturas i386).
- Prefiere usar paginación a la segmentación por:
  - La gestión de memoria es más simple.
  - Linux busca la portabilidad y no todas las arquitecturas dan soporte para segmentos.

# 7. Casos de estudio

## 7.1. Linux

- Adopta un modelo de tres niveles lo que lo hace muy flexible en arquitecturas de 64 bits:



# 7. Casos de estudio

## 7.1. Linux

- Con direcciones de 64 bits se usa el esquema de 3 niveles explicado anteriormente, sin embargo, en plataformas de 32 bits, arquitecturas Intel, se usa un sistema de paginación de 2 niveles. Linux se acomoda al esquema de 2 niveles definiendo el tamaño del directorio intermedio de páginas como 1.

- Se considera que el Directorio de Páginas intermedio contienen una única página para mantener compatibilidad en 32 y 64 bits.

- La páginas suelen ser de 4 Kb.

# 7. Casos de estudio

## 7.1. Linux

### Algoritmo de reemplazo

- Se basa en el algoritmo de reloj pero sustituye el bit de referenciada por un contador de 8 bits (bits de edad).
- Cada vez que se accede a una página se incrementa el contador.
- Durante el proceso de búsqueda vamos decrementando en 1 la edad de las páginas que no tengan una edad de 0.
- Cuando se necesita reemplazar una página se busca aquella que tenga una edad=0.

# 7. Casos de estudio

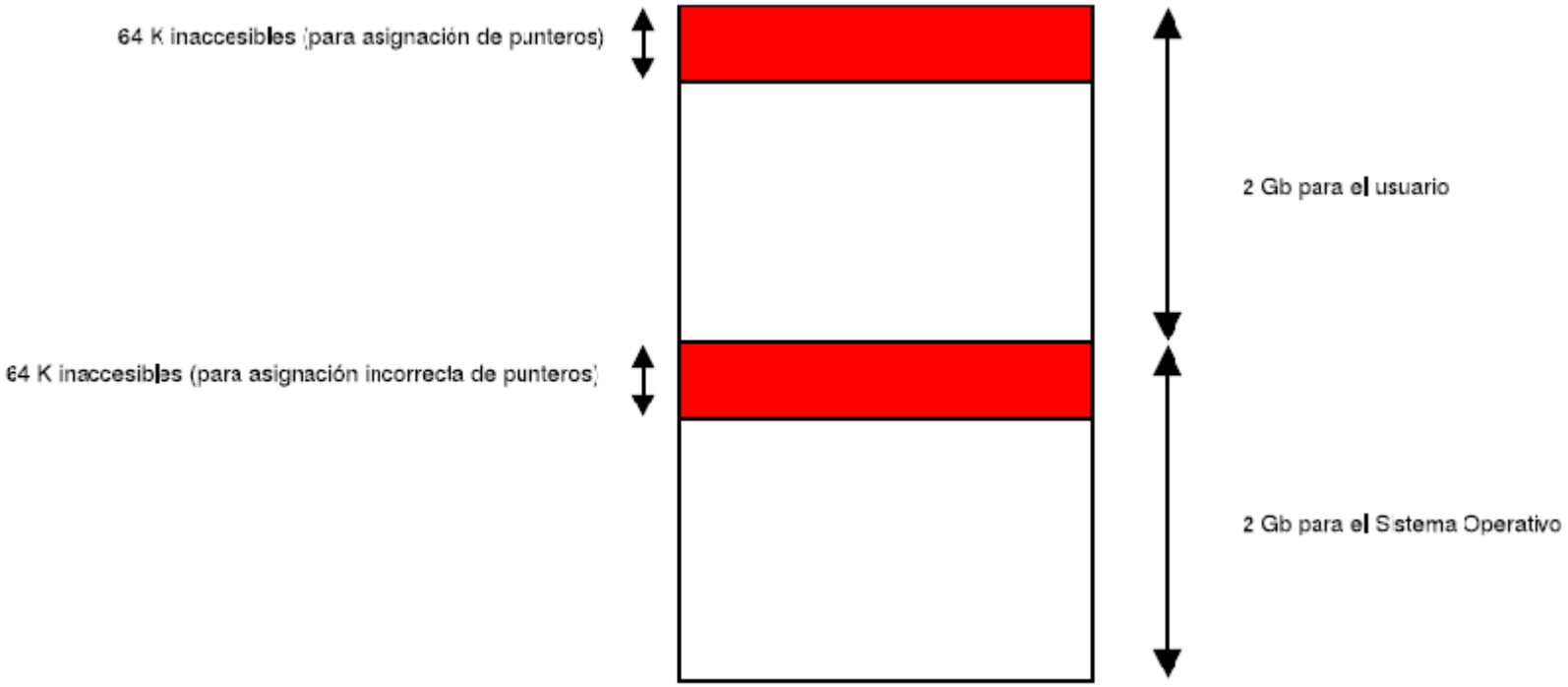
## 7.2. Windows

- Usa memoria virtual con paginación por demanda con preasignación de un determinado número de tramas.
- Usa tamaños de páginas entre 4 y 64 Kb.
- En plataformas Intel el tamaño es de 4 Kb.

# 7. Casos de estudio

## 7.2. Windows

- El espacio lógico por proceso tiene direcciones de 32 bits = 4 Gb:



# 7. Casos de estudio

## 7.2. Windows

- Usa una estructura a dos niveles de directorios de página:

# 7. Casos de estudio

## 7.2. Windows

- **Asignación de páginas a procesos y algoritmo de Reemplazo**

  Se usa una política de asignación variable. Inicialmente se asigna un determinado número de tramas, aplicando posteriormente Working Set. Las páginas que se van a reemplazar se eligen de entre los marcos asignados al proceso (ámbito de reemplazo local).

# 7. Casos de estudio

## 7.2. Windows

¿Cómo se calcula el conjunto de trabajo?

- Cuando hay memoria disponible, el Conjunto de Trabajo de los procesos activos crece. Cuando se produce un fallo de página, se trae la nueva página a memoria sin expulsar una página antigua, incrementándose el conjunto residente del proceso en una página.
- Si la memoria escasea, el sistema operativo mueve las páginas que se han utilizado hace más tiempo, de cada uno de los procesos activos, a la zona de swap, reduciendo el tamaño de los conjuntos residentes de los procesos.

Si no se puede aumentar el conjunto residente de un proceso se usa reemplazo mediante LRU.

# Dudas

**CONSEJO**:

No dejar para el final la realización del test del tema publicado en Moodle