Tema 3
Memoria

Contenido

Tema 3. Gestión de memoria. 7h

3.1 La jerarquía de memoria y su organización

3.2 Etapas en la traducción de direcciones

- Espacios de direcciones
- Fases de traducción de direcciones

3.3 Modelos de gestión de memoria

3.4 Paginación y segmentación

- Paginación en uno y dos niveles.
- Tabla de traducción invertida y TLB
- Segmentación y modelos mixtos

3.5 Memoria Virtual

- HW y SW para su implementación
- Algoritmos de asignación y remplazo
- Hiperpaginación y algoritmo PFF

Contenido

3.1 Jerarquía de memoria

3.1 Jerarquía de mem.

objetivos del sistema de gestión de memoria

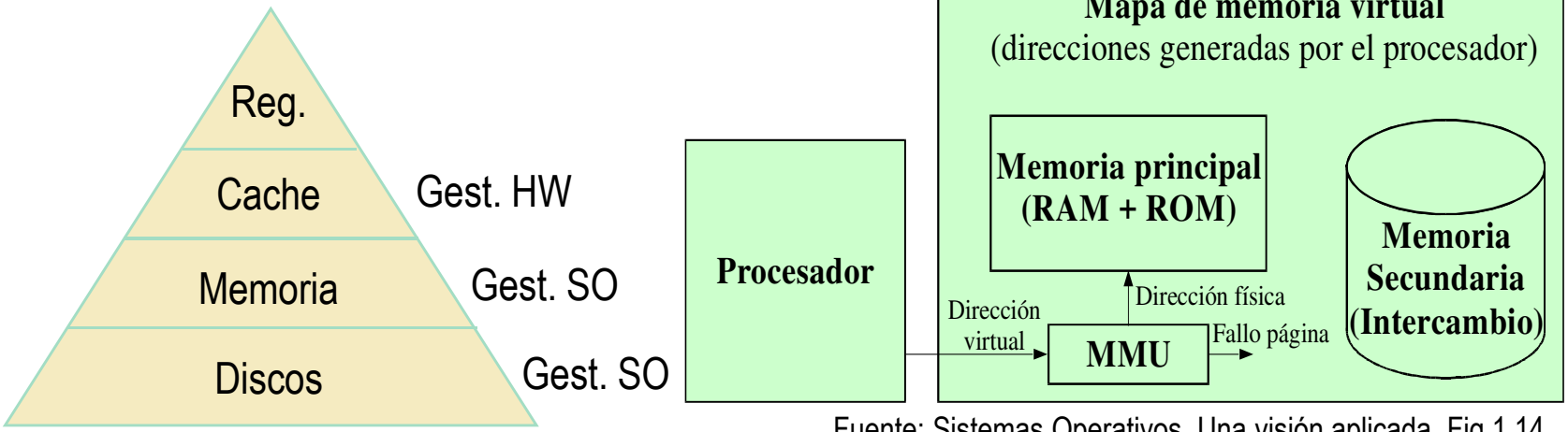
- S.O. multiplexa recursos entre procesos
  - Cada proceso cree que tiene una máquina para él solo
  - Gestión de procesos: Reparto de procesador
  - Gestión de memoria: Reparto de memoria

- Objetivos:
  - Ofrecer a cada proceso un espacio lógico propio
  - Proporcionar protección entre procesos
  - Permitir que procesos compartan memoria
  - Maximizar el grado de multiprogramación
  - Proporcionar a los procesos espacios de memoria muy grandes
  - Al mismo tiempo: velocidad y bajo coste

3.1 Jerarquía de mem.

solución: memoria virtual

- Memoria virtual
  - La CPU genera direcciones virtuales
  - Parte del mapa de memoria está en Mem. y parte en HD
  - La MMU traduce las dir. virtuales a dir. físicas
  - Fallo de página cuando la dir. no está en Mem. principal
  - El SO trae una página a Mem (posible reemplazo en Mem)
- Responde a los objetivos anteriores
- Completa la jerarquía de memoria



Fuente: Sistemas Operativos. Una visión aplicada. Fig 1.14.

3.2 Traducción de direcciones

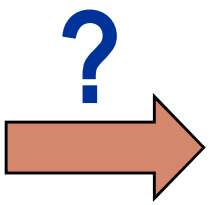
3.2 Traducción de dir.

¿Cuál es el problema?

Programa Fuente

```
int i,b;
int s;
int tabla[10];
void main ()
{
   s=f();
}

int f ()
{
   b=0;
   for (i=0; i<10; i++)
   {
      b=b+tabla[i];
   }
   return b;
}
```



Memoria del computador

```
;; Dirección lógica inicial del código: 1024
1024 JSR 1036
1028 MOVE.L R0,4104
1032 RTS
1036 MOVE.L #0,4100
1040 MOVE.L #0,4096
1044 MOVE.L 4096,R0
1048 ADD.L 4108(R0),4100
1052 ADD.L #4,4096
1056 CMP.L 4096,#40
1060 JNE 1044
1064 MOVE.L 4100,R0
1068 RTS
;; Dirección lógica inicial de los datos: 4096
4096 0 ; variable i
4100 0 ; variable b
4104 0 ; variable s
4108 ? ; tabla[0] no inicializado
4112 ? ; tabla[1] no inicializado
4116 ? ; tabla[2] no inicializado
...
4144 ? ; tabla[9] no inicializado
;; Dirección lógica inicial de la pila: 15360
15360 ? ; pila (valor inicial no
... ; definido)
16380 ? ; cima de la pila
```

3.2 Traducción de dir.

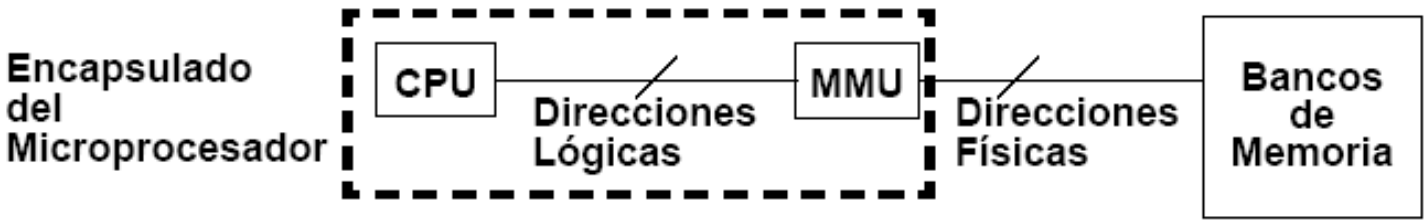
espacio de direccionamiento

- Antes de llegar a la memoria virtual:
  - Entender cómo un programa usa la memoria
  - Cómo vemos la memoria a alto nivel
- Espacio de direccionamiento
  - Rango de valores usados para acceder a memoria
- Los valores pueden ser
  - Etiquetas de texto
    - Usados a alto nivel (LAN, ASM). Variables y etiquetas
  - Números naturales
    - Rango $[0:2^n-1]$ cuando el bus de direcciones es de n bits
- Distinguir
  - Espacio posible: todas las direcciones posibles
  - Espacio válido: a las que está permitido acceder

3.2 Traducción de dir.

otra distinción: espacio lógico y físico

- Espacio lógico:
  - Las direcciones que emite el procesador y que contiene en los reg. de propósito general y de direcciones
  - Tamaño del espacio lógico válido para un proceso
    - Depende de la cantidad de memoria asignada por el SO
- Espacio físico:
  - El que aceptan los bancos de memoria del computador
  - Tamaño del espacio físico válido del sistema
    - Direcciones a las que responden los bancos de RAM, ROM y controladores de E/S mapeados en memoria
- Unidad de manejo de memoria (MMU)
  - Traduce direcciones lógicas a físicas.



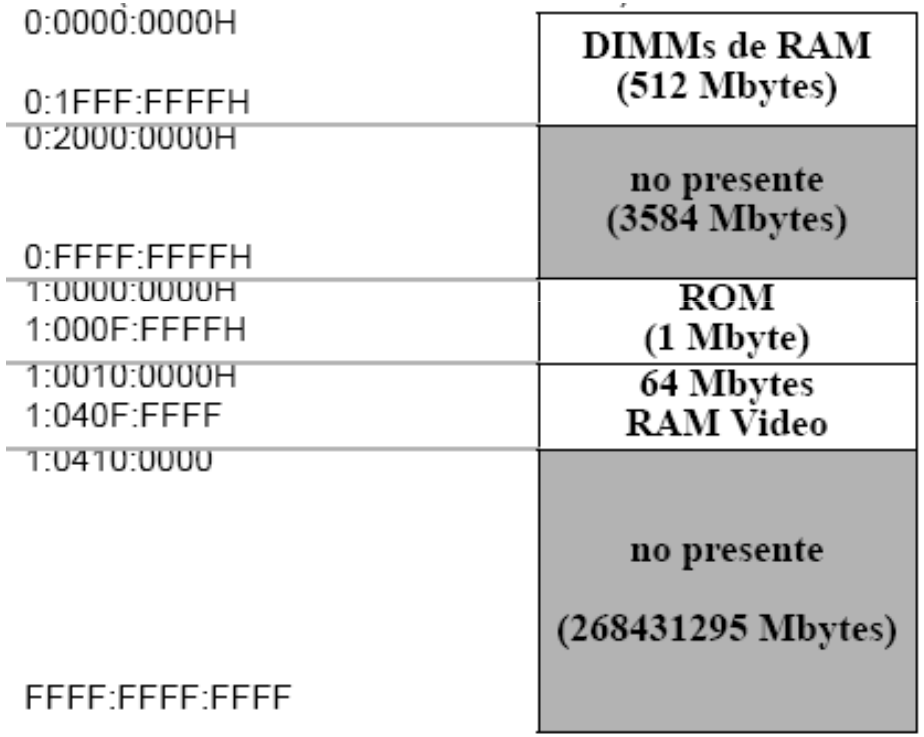
3.2 Traducción de dir.

ejemplo de espacio de direccionamiento

- Sistema SGI con MIPS R10000 (64bits)
  - 512MB RAM, 1MB ROM y 64MB RAM video
  - Espacio lógico
    - Posible: [0:2^{64}-1] (16 ExaBytes = 17.179.869.184 GB!!)
    - Válido: el que asigne el SO a cada proceso

- Bus de direcciones
  - De 48 bits
  - Ahorro de pines
  - 256TB es razonable

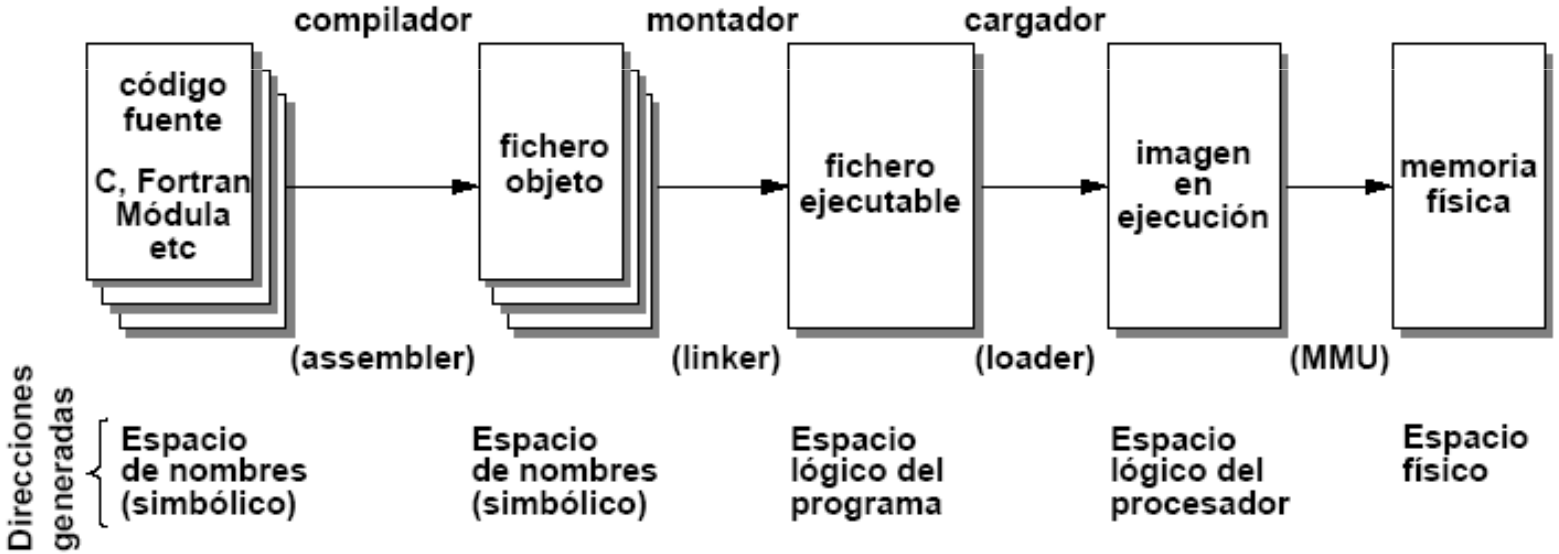
- Espacio físico
  - Posible: [0:2^{48}-1] (256 TB)
  - Válido: [0:2^{29}-1] (512 MB)



3.2 Traducción de dir.

espacios de direcciones

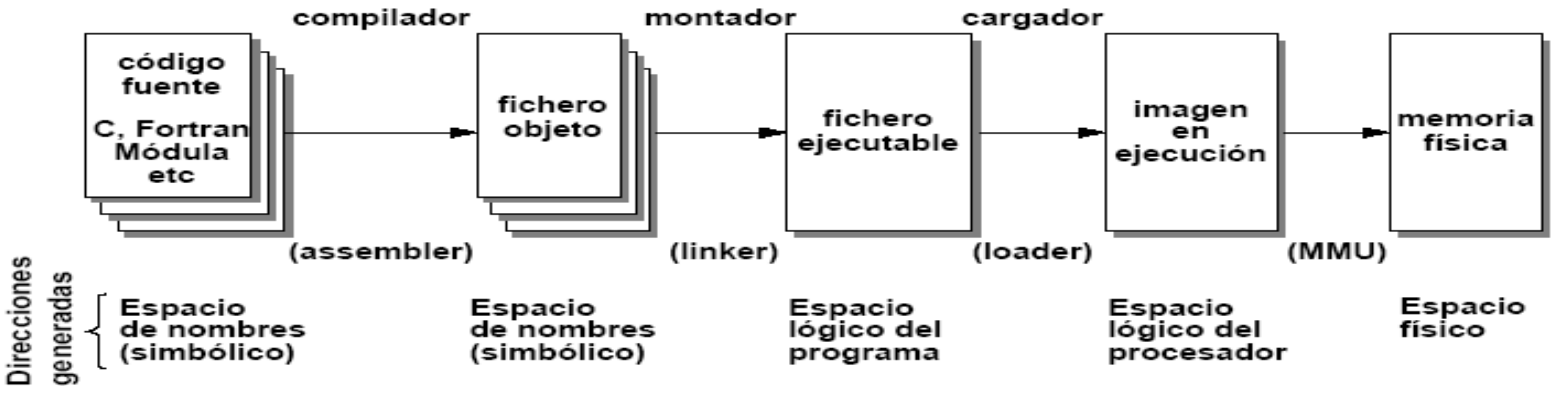
- A alto nivel (LAN)
  - El programador no usa direcciones numéricas
    - Variables para identificar posiciones de mem. con datos
    - Funciones para identificar posiciones de mem. con instruc.
  - Esos nombres simbólicos se traducen a direcciones
    - Distintas fases de traducción
  - Ventajas: abstracción y portabilidad



3.2 Traducción de dir.

etapas en la traducción de direcciones

- Etapas de traducción de direcciones
  - Compilación
    - Separa el código de los datos (crea zonas distintas)
    - Resuelve referencias internas (dentro de cada módulo)
  - Montaje
    - Resuelve referencias cruzadas entre módulos
    - Reubicación si fuera necesario
  - Carga
    - Asigna direcciones iniciales a los segmentos del programa
    - Reubicación si fuera necesario
  - Ejecución
    - Traduce direcciones lógicas a físicas (pueden coincidir).



3.2 Traducción de dir.

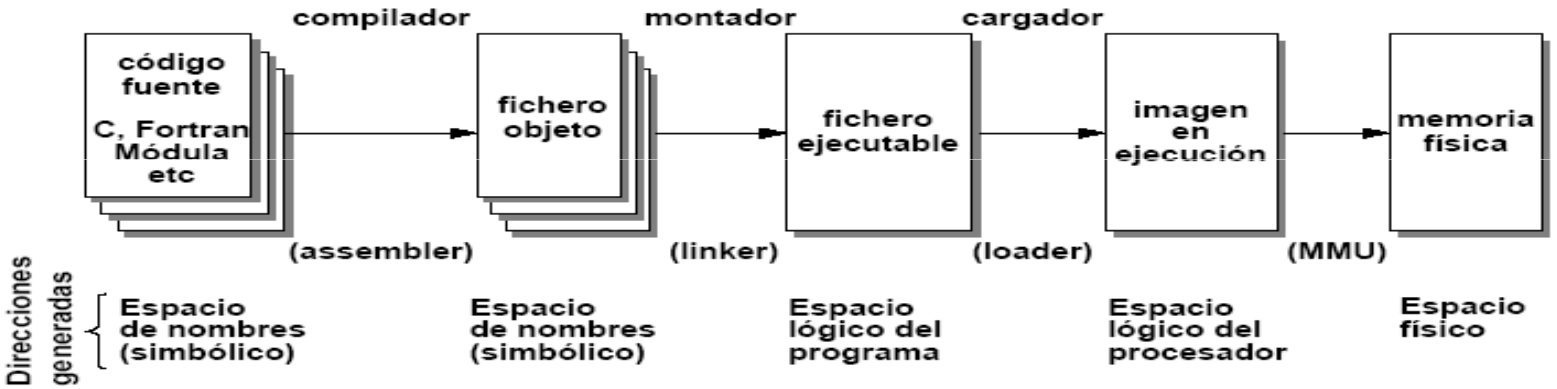
etapas en la traducción de direcciones

Programa Fuente → Memoria Fisica

```
int i;
void main
()
{
   i=32;
}
```

```
.....
MOVi ?,#32
.....
```

¿En que posición física se almacena el valor de i?



3.2 Traducción de dir.

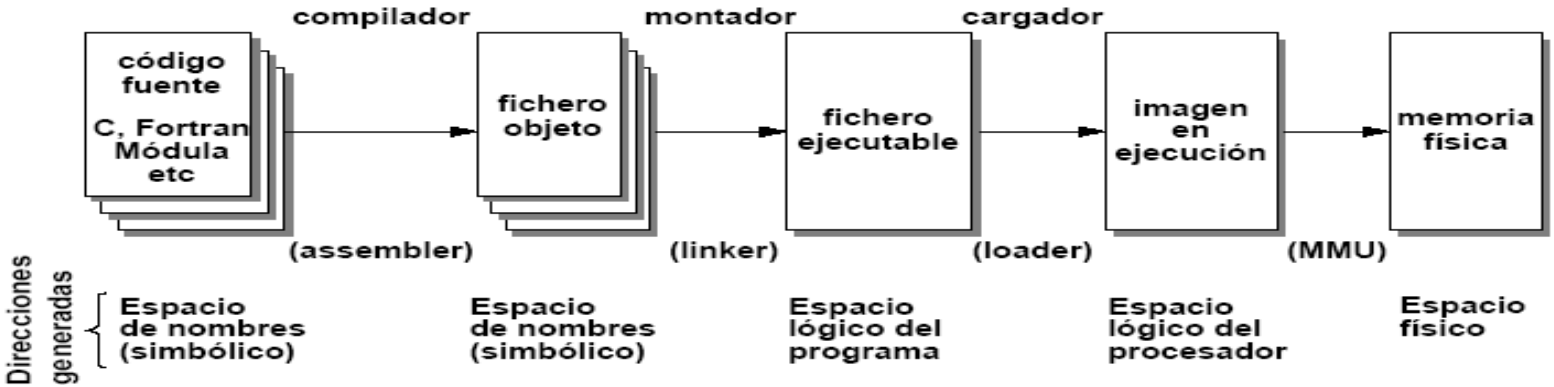
etapas en la traducción de direcciones

Programa Fuente → Memoria Fisica

```
int i;
void main
()
{
  i=32;
}
```

```
.....
MOVi ?,#32
.....
```

Determinado en tiempo de compilación



3.2 Traducción de dir.

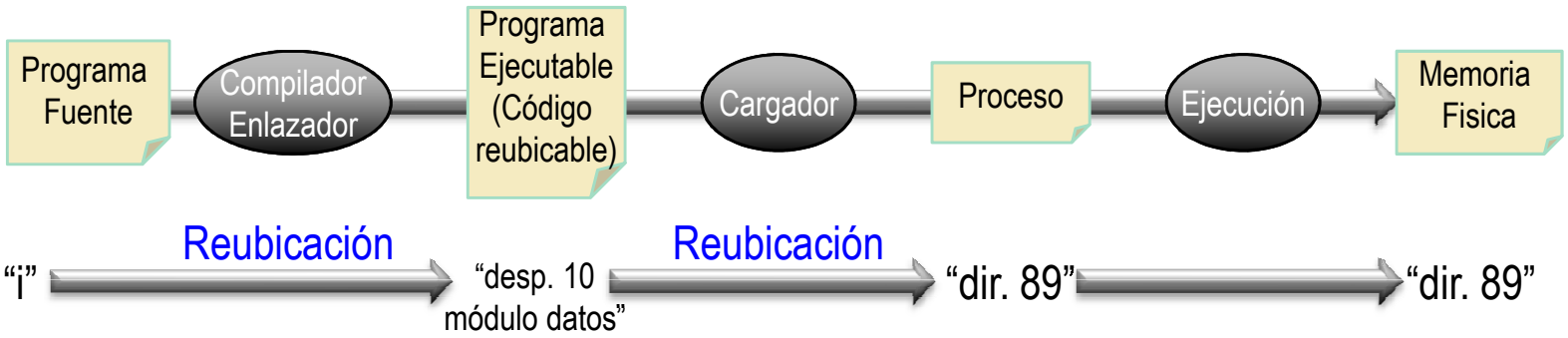
etapas en la traducción de direcciones

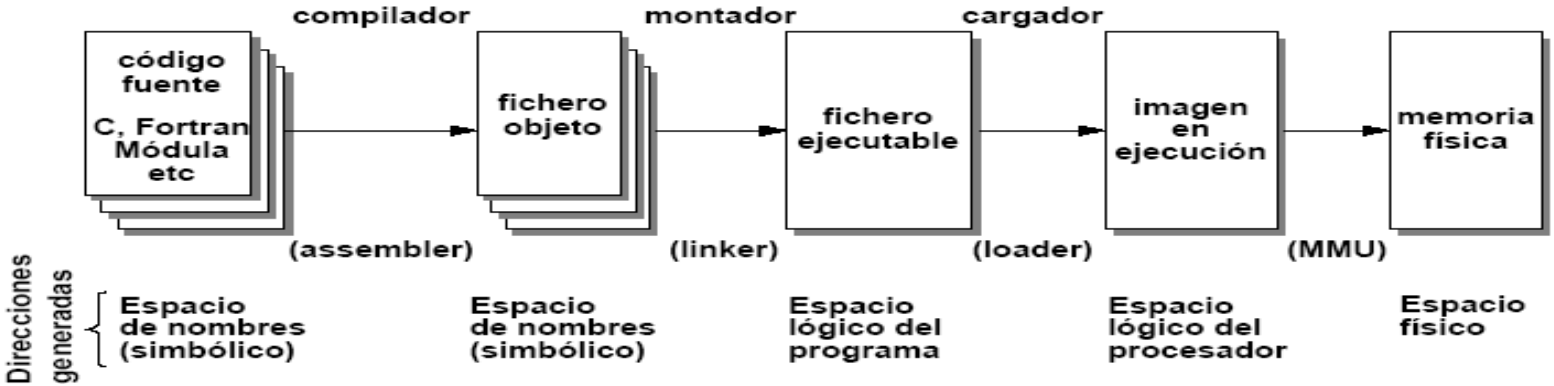
Programa Fuente → Memoria Fisica

```
int i;
void main
()
{
  i=32;
}
```

```
.....
MOVi ?,#32
.....
```

Reubicación en la carga





3.2 Traducción de dir.

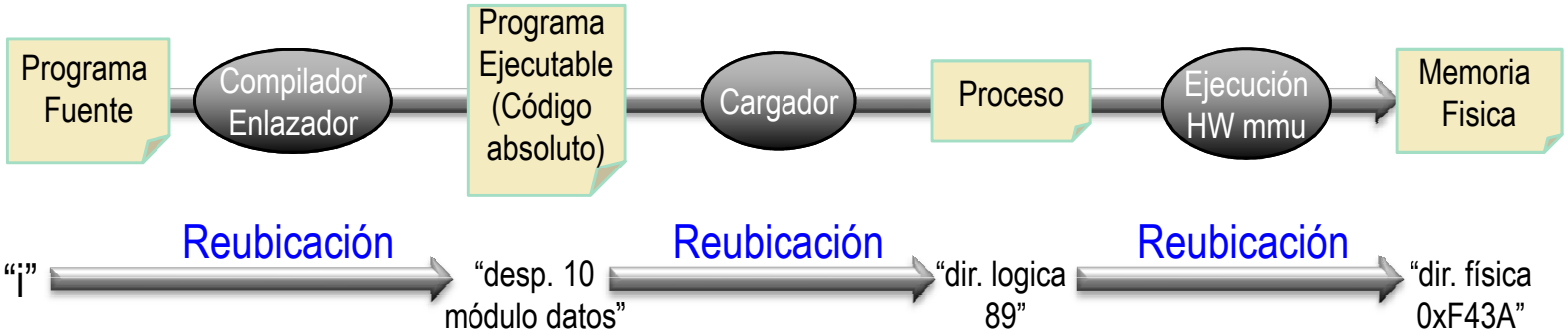
etapas en la traducción de direcciones

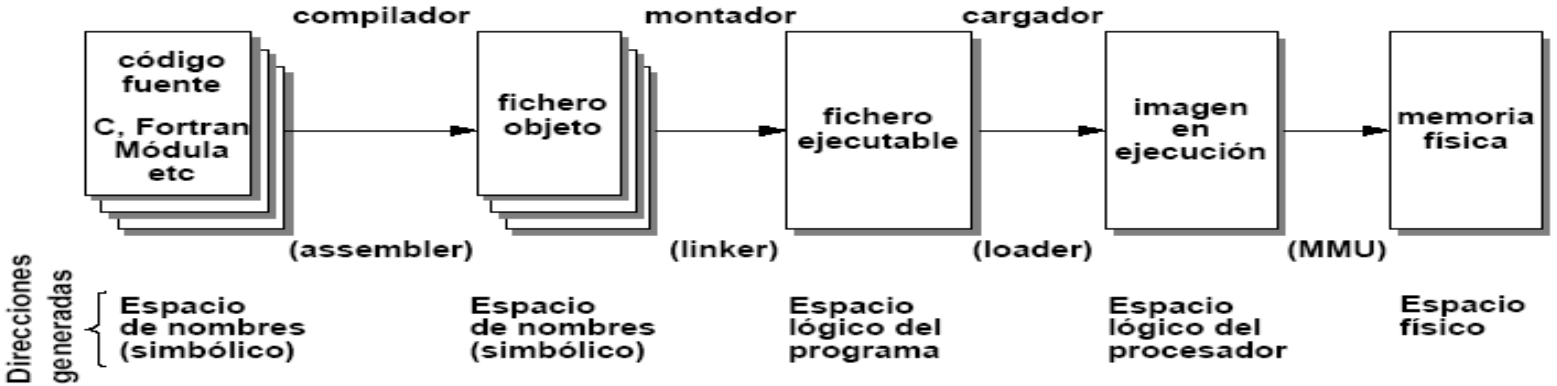
Programa Fuente → Memoria Fisica

```
int i;
void main
()
{
  i=32;
}
```

```
.....
MOVi ?,#32
.....
```

Reubicación en tiempo de ejecución





3.3 Modelos de memoria

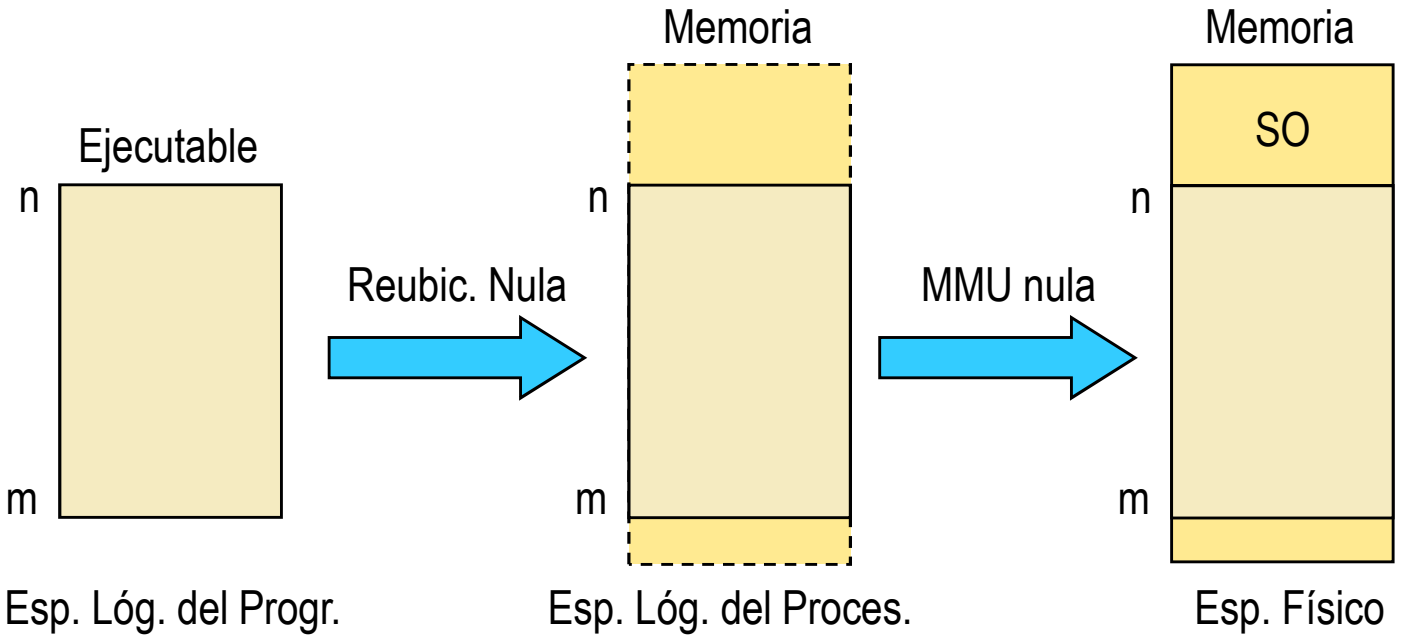
3.3 Modelos de mem.
modelos básicos de traducción de dir.

- ¿Cómo se lleva a cabo la traducción de direcciones?
  - Depende del modelo de sistema
- Características de los modelos:
  - Uniprogramado/Multiprogramado
    - Uno o varios procesos en memoria
  - Residente/No residente
    - El proceso permanece o no en memoria al ejecutarse
  - Inmóvil/Móvil
    - Ocupa las mismas posiciones o se puede mover
  - Contiguo/No contiguo
    - Posiciones de memoria consecutivas o distribuidas
  - Entero/No entero
    - Posibilidad o no de que no esté cargado todo el proceso en memoria

3.3 Modelos de mem.

uniprogr.|residente|inmóvil|contiguo|entero

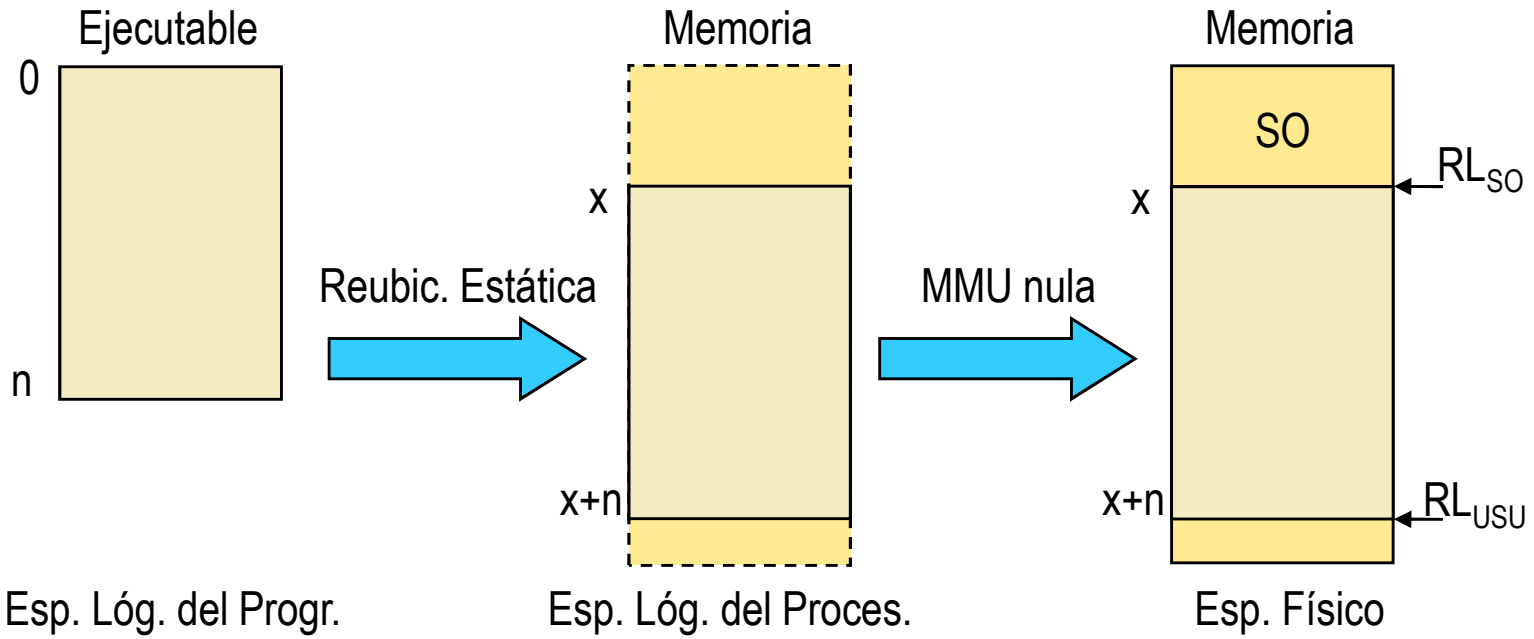
- Modelo primitivo
  - Sin reubicación
    - Esp. Lóg. Prog = Esp. Lóg. Proc. = Esp. Físico
    - Al enlazar ya se genera el código con las direcciones físicas definitivas (reubicación en montaje)
    - No hay ninguna traducción después de enlazar
    - El enlazador tiene que saber cuanto ocupa el SO en memoria



3.3 Modelos de mem.

uniprogr.|residente|inmóvil|contiguo|entero

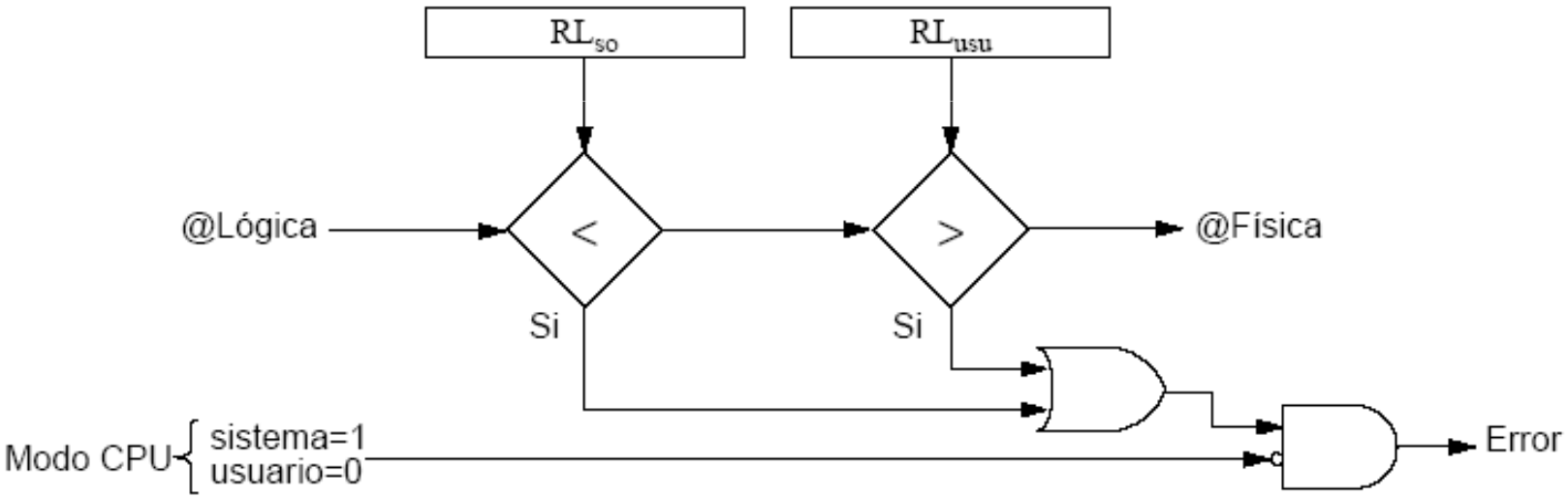
- Modelo primitivo
  - Con reubicación
    - Esp. Lóg. Prog ≠ Esp. Lóg. Proc. = Esp. Físico
    - El enlazador genera el ejecutable entre 0 y n
    - El cargador reubica el código al llevarlo a memoria (reubicación en la carga)



3.3 Modelos de mem.

uniprogr.|residente|inmóvil|contiguo|entero

- Modelo primitivo: Función de la MMU
  - La MMU no traduce direcciones
    - (Esp. Lóg. Proc. = Esp. Físico)
  - Pero vigila los accesos a regiones
  - En modo usuario:
    - La @Lógica tiene que estar en el rango [RL_{SO}, RL_{USU}]
    - En caso contrario se lanza una excepción
  - En modo supervisor: no hay restricciones

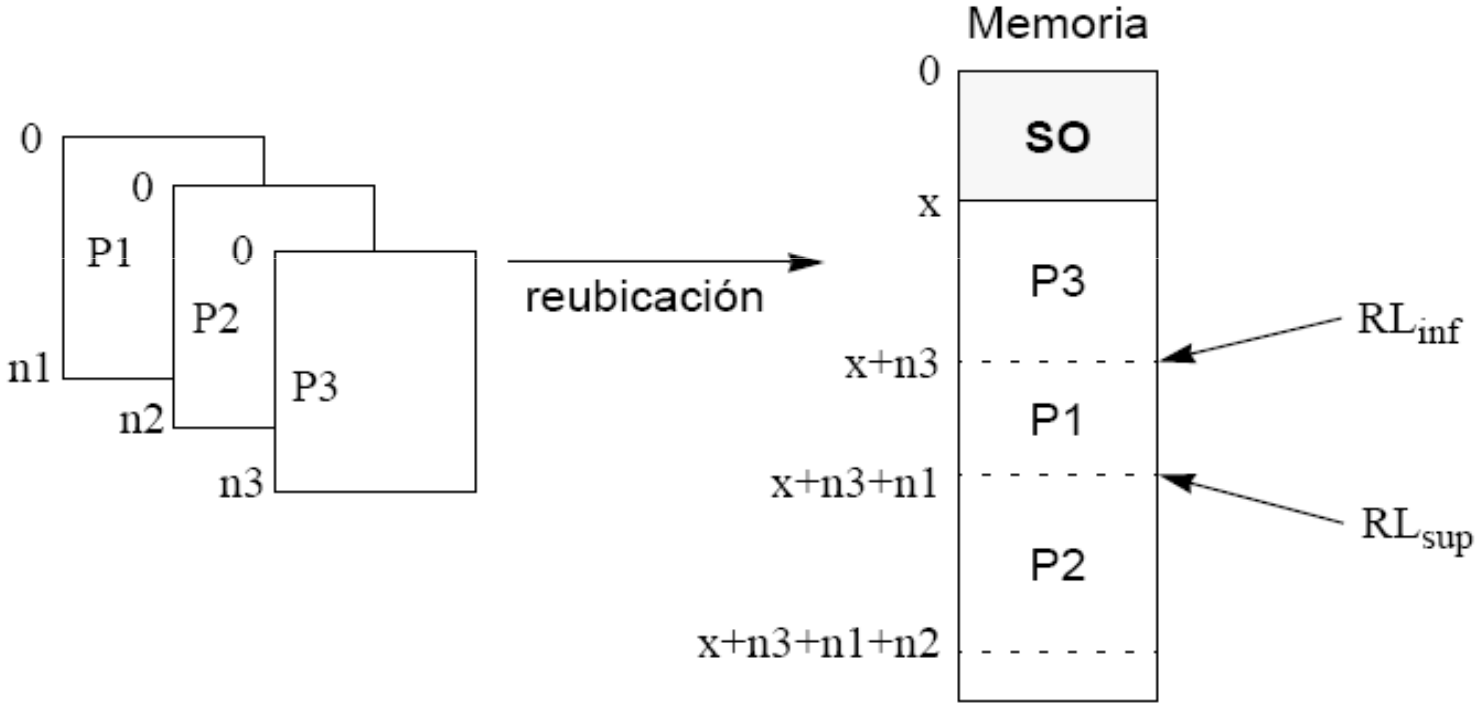


3.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

Modelo Multiprogramado

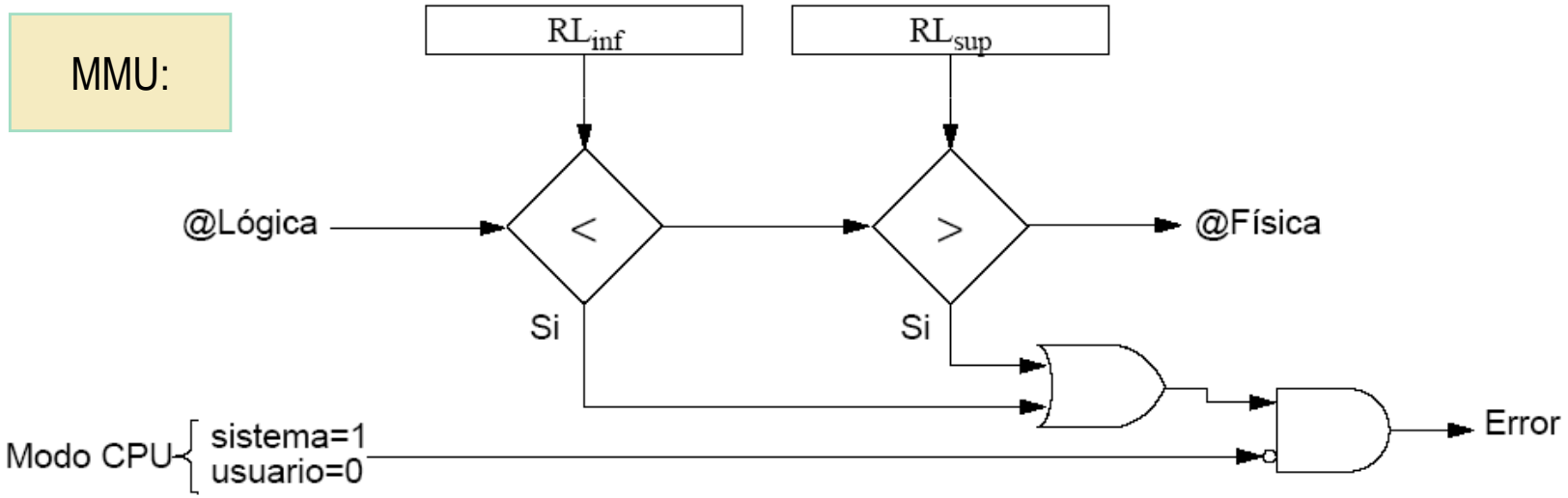
- La multiprogramación mejora el uso de CPU
  - Repartir la memoria entre más de un proceso
- Dos problemas:
  - Proteger los procesos entre si
  - Ubicar los procesos en memoria (reubicación en la carga)



3.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

- Modelo Multiprogramado: Protección
  - Esp. Lóg. Prog ≠ Esp. Lóg. Proc. = Esp. Físico
  - Cada proceso tiene un límite inferior y sup.
  - Esos limites están en el PCB de cada proceso
  - En cada cambio de contexto:
    - Se actualizan los registros límite de la MMU
  - En modo supervisor no hay restricciones



3.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

- Modelo Multiprogramado: Asignación
  - El SO debe controlar las zonas libres y ocupadas
- Dos posibilidades
  - Particiones fijas (estáticas)
    - El SO divide la memoria en particiones (tamaño prefijado)
    - Proceso nuevo → asignarle partición ≥ tamaño del proc.
    - Gestión: tabla con tantas entradas como particiones
  - Particiones variables (dinámicas)
    - Inicialmente toda la memoria es una única partición
    - Proceso nuevo →
      - Buscar partición mayor o igual al tamaño del proceso
      - Dividir la part. en dos: un trozo para el proceso y el otro libre
    - Gestión de bloques libres y asignados
      - Listas enlazadas o mapa de bits

3.3 Modelos de mem.

Problemas de la gestión de memoria

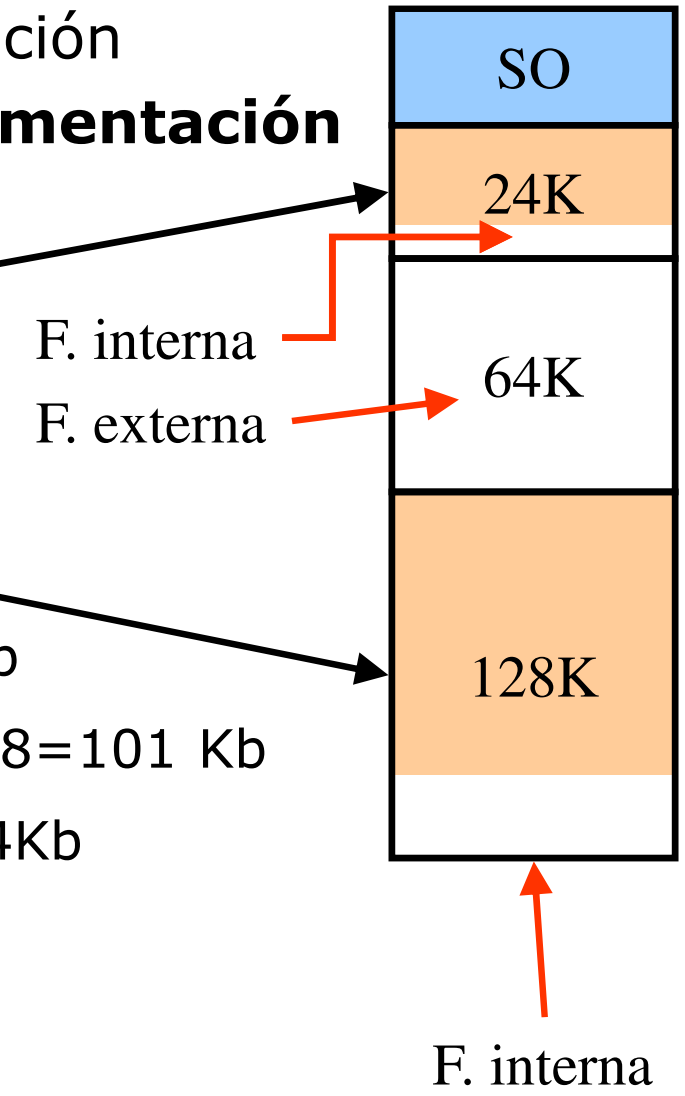
- Fragmentación interna
  - Memoria asignada a un proceso pero no usada
  - Aparece cuando
    - Asignas a un proceso un bloque mayor del necesario
    - Típico en particiones de tamaño fijo
  - Solución: reducir tamaño de las particiones
- Fragmentación externa
  - Bloques de memoria libre no usados
    - Son demasiado pequeños para ningún proceso
  - Aparece cuando
    - Los procesos necesitan bloques contiguos de memoria
  - Solución: desfragmentar

3.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

- Particiones fijas
  - Difícil escoger la mejor partición
  - Perdida de espacio por **fragmentación**
- Ejemplo:

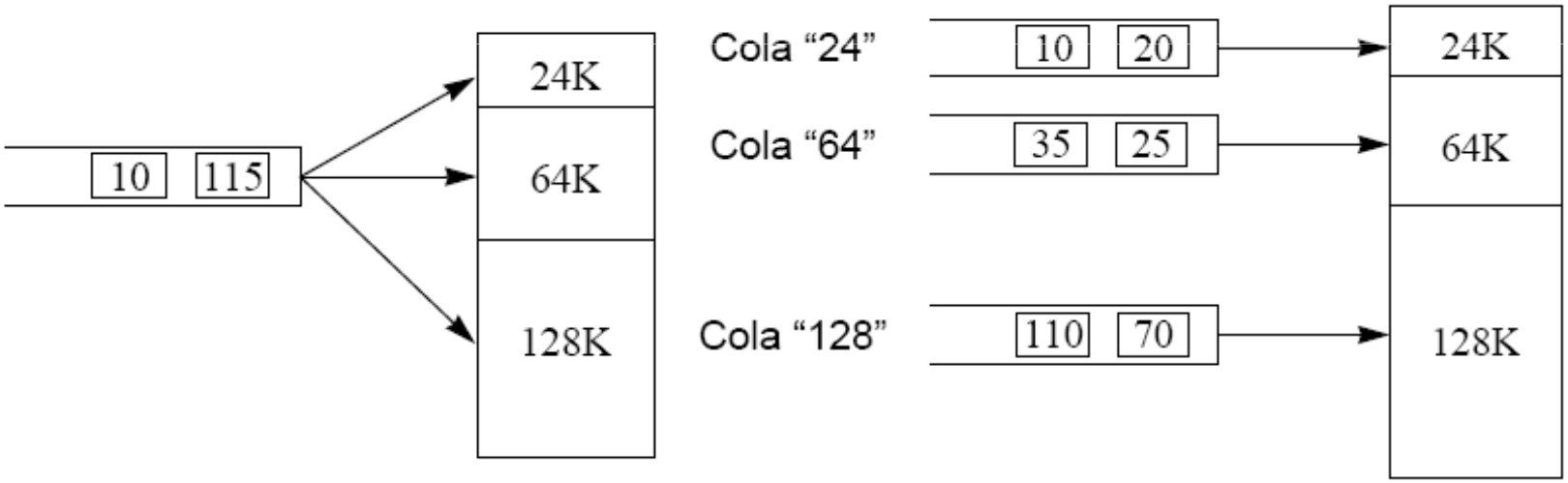
P1 (15Kb)
P2 (70Kb) – sin espacio
P3 (100Kb)



- Particiones: 24Kb, 64Kb, 128Kb
- Memoria libre (real) = 9+64+28=101 Kb
- Memoria libre (disponible) = 64Kb

3.3 Modelos de mem.

- Particiones fijas: Políticas de asignación
- Una sola cola:
  - El primero de la cola entra en la primera partición de tamaño suficiente
- Una cola para cada partición
  - Cada proceso entra en la primera cola válida
  - Aunque otra partición esté libre tiene que esperar en su cola a la partición correspondiente



3.3 Modelos de mem.

Particiones variables

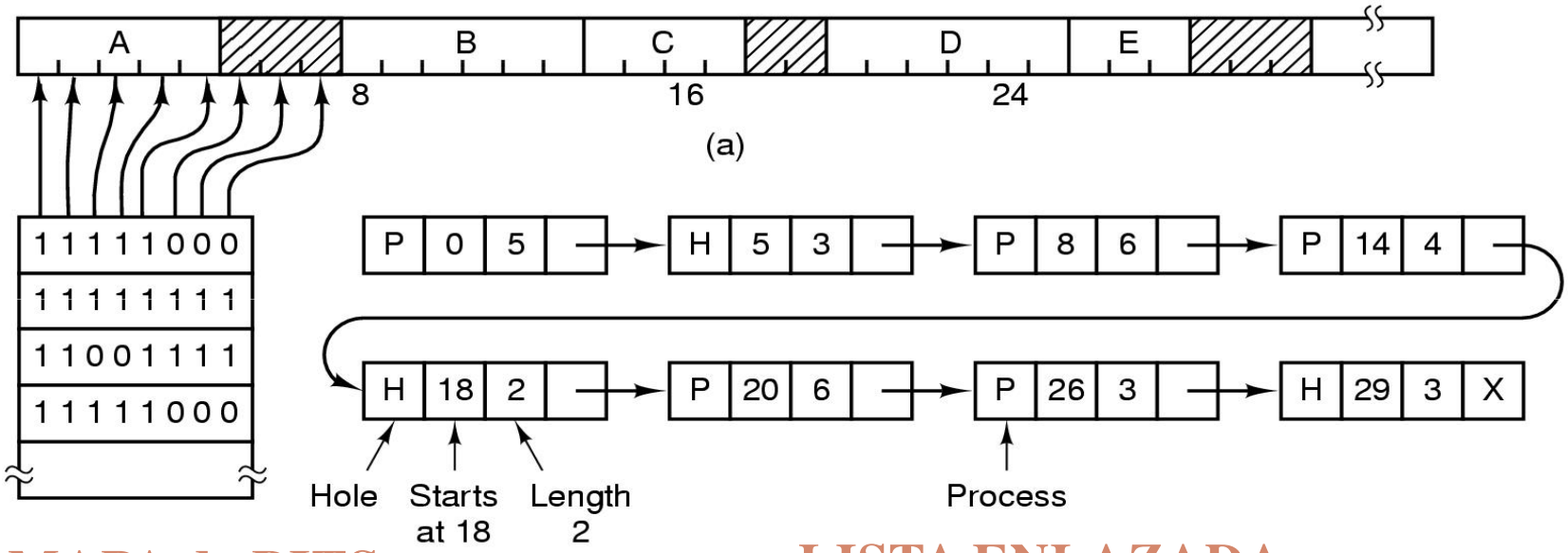
- Gestión del espacio
  - Mapa de bits: Memoria dividida en bloques pequeños (clics)
    - –Un mapa de bits indica los clics libres con ceros
    - –Se asignan clics consecutivos al proceso
    - –Puede haber fragmentación interna
  - Listas enlazadas: cada elemento identifica una zona de mem.
    - –Dos listas: una de zonas libres y otra de zonas ocupadas
    - –Puede haber varias listas de zonas libres según tamaños

3.3 Modelos de mem.

Particiones variables

- Ejemplos de gestión del espacio
  - Con mapa de bits
  - Con listas enlazadas

ZONA DE MEM. CON 5 PROCESOS y 3 HUECOS LIBRES



MAPA de BITS

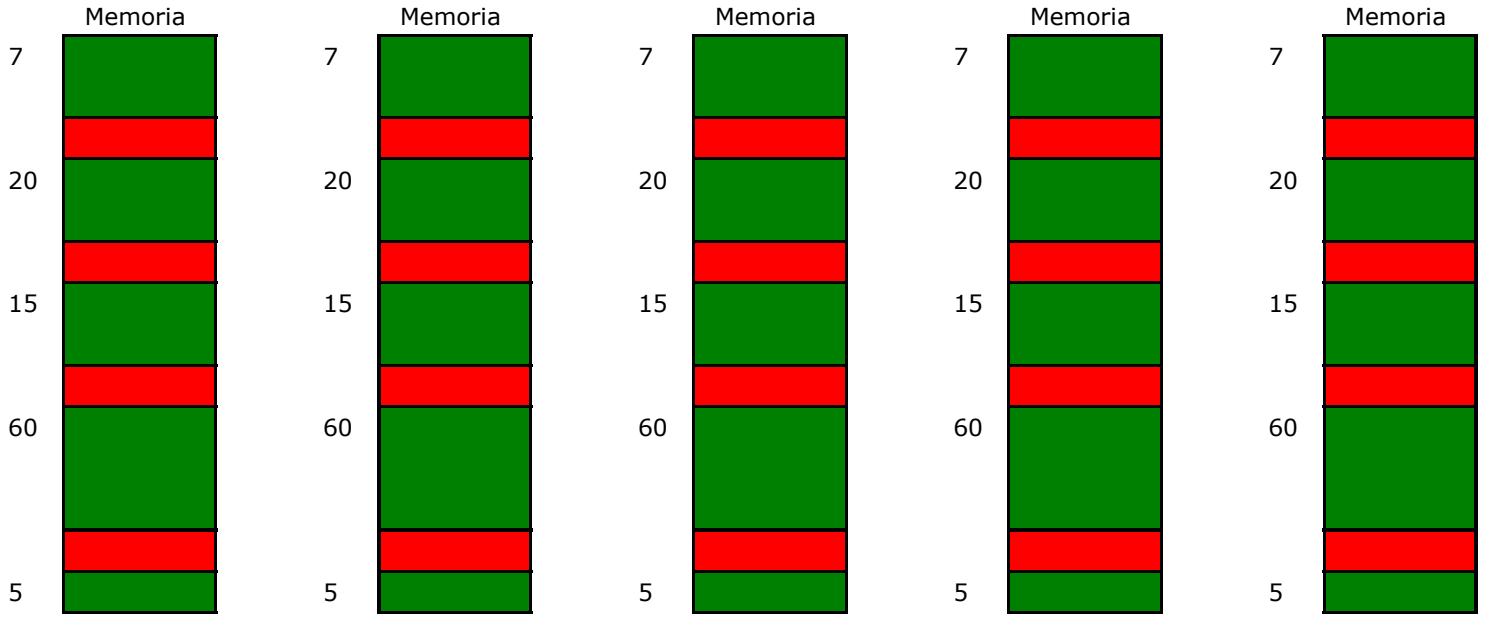
LISTA ENLAZADA

3.3 Modelos de mem.

Particiones variables

- Políticas de asignación
  - Best-Fit: Busca el bloque al que mejor se ajuste el proceso
    - –Genera un bloque libre más pequeño posible → frag. ext.
  - Worst-Fit: Busca el bloque más grande de todos
    - –Genera el bloque libre más grande posible → inanición pr. grandes
  - First-Fit: El primero que encuentra. Comportamt. Intermedio
  - Next-Fit: El primero que encuentra a partir del último asignado

Peticiones de memoria:
P1: 15, P2: 5

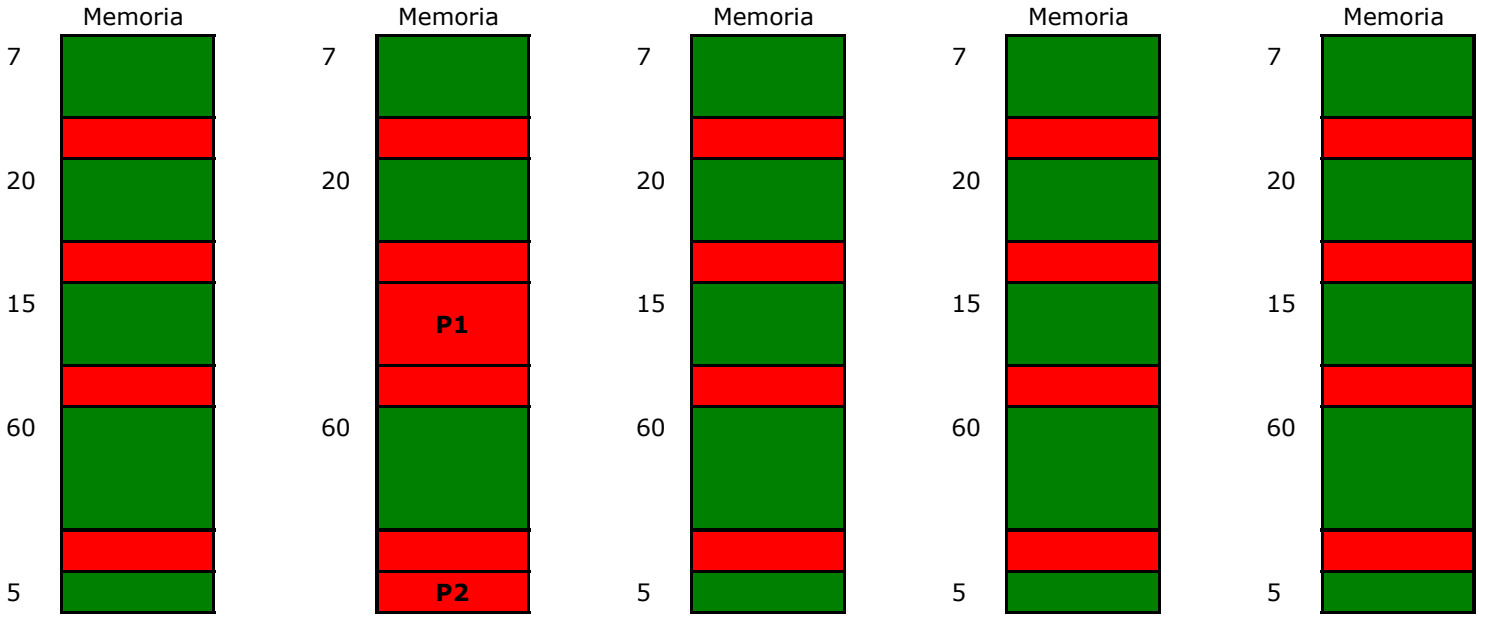


3.3 Modelos de mem.

Particiones variables

- Políticas de asignación
  - Best-Fit: Busca el bloque al que mejor se ajuste el proceso
    - –Genera un bloque libre más pequeño posible → frag. ext.
  - Worst-Fit: Busca el bloque más grande de todos
    - –Genera el bloque libre más grande posible → inanición pr. grandes
  - First-Fit: El primero que encuentra. Comportamt. Intermedio
  - Next-Fit: El primero que encuentra a partir del último asignado

Peticiones de memoria:
P1: 15, P2: 5

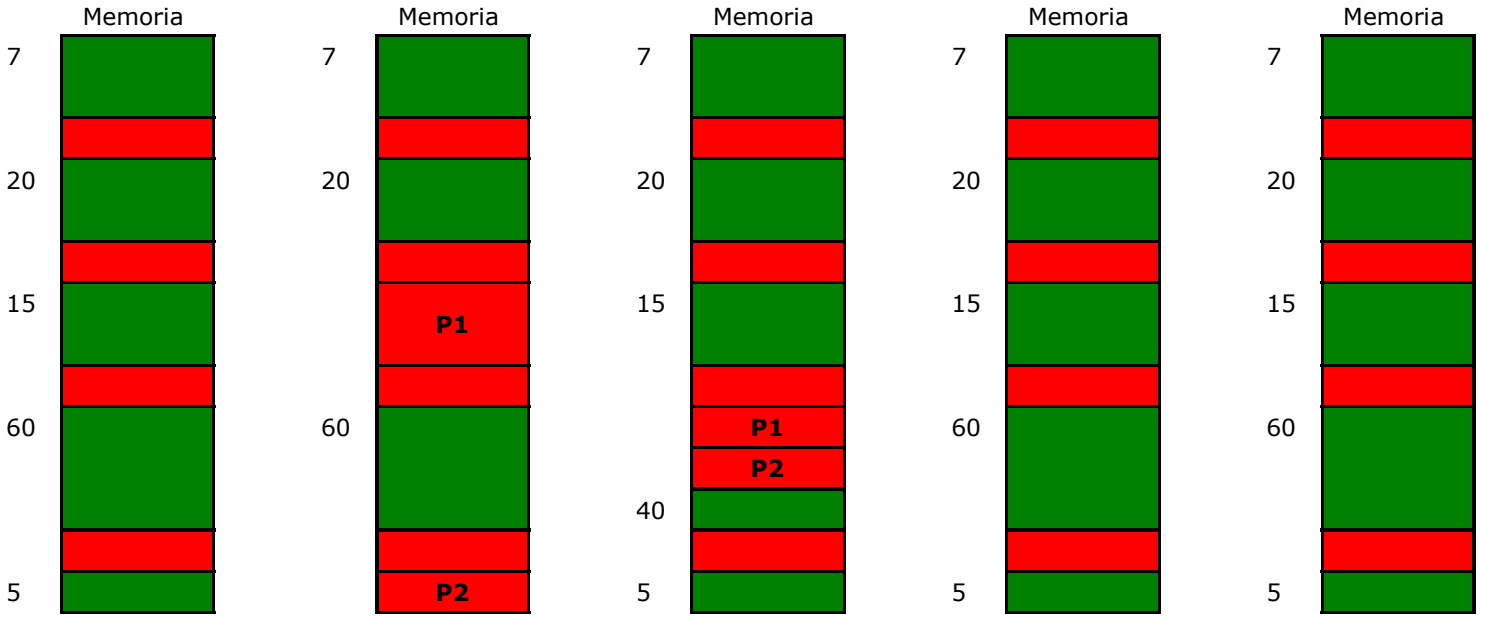


3.3 Modelos de mem.

Particiones variables

- Políticas de asignación
  - Best-Fit: Busca el bloque al que mejor se ajuste el proceso
    - –Genera un bloque libre más pequeño posible → frag. ext.
  - Worst-Fit: Busca el bloque más grande de todos
    - –Genera el bloque libre más grande posible → inanición pr. grandes
  - First-Fit: El primero que encuentra. Comportamt. Intermedio
  - Next-Fit: El primero que encuentra a partir del último asignado

Peticiones de memoria:
P1: 15, P2: 5

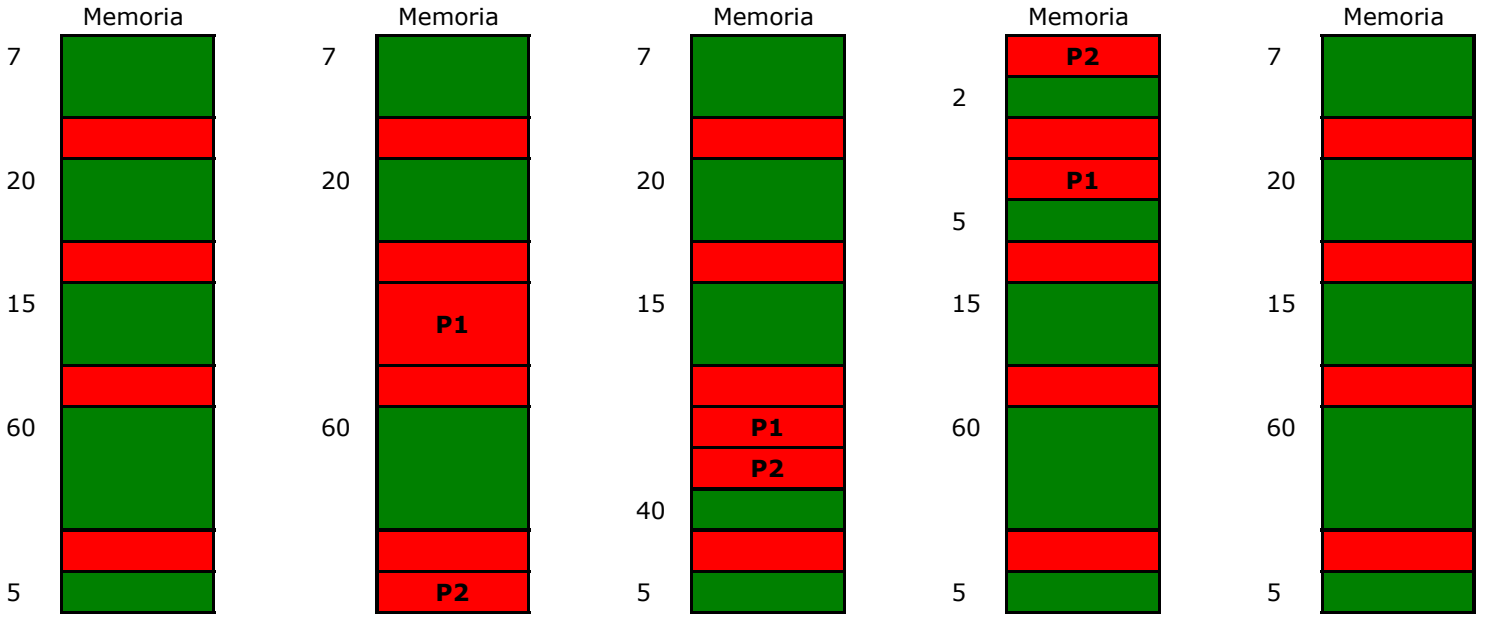


3.3 Modelos de mem.

Particiones variables

- Políticas de asignación
  - Best-Fit: Busca el bloque al que mejor se ajuste el proceso
    - –Genera un bloque libre más pequeño posible → frag. ext.
  - Worst-Fit: Busca el bloque más grande de todos
    - –Genera el bloque libre más grande posible → inanición pr. grandes
  - First-Fit: El primero que encuentra. Comportamt. Intermedio
  - Next-Fit: El primero que encuentra a partir del último asignado

Peticiones de memoria:
P1: 15, P2: 5

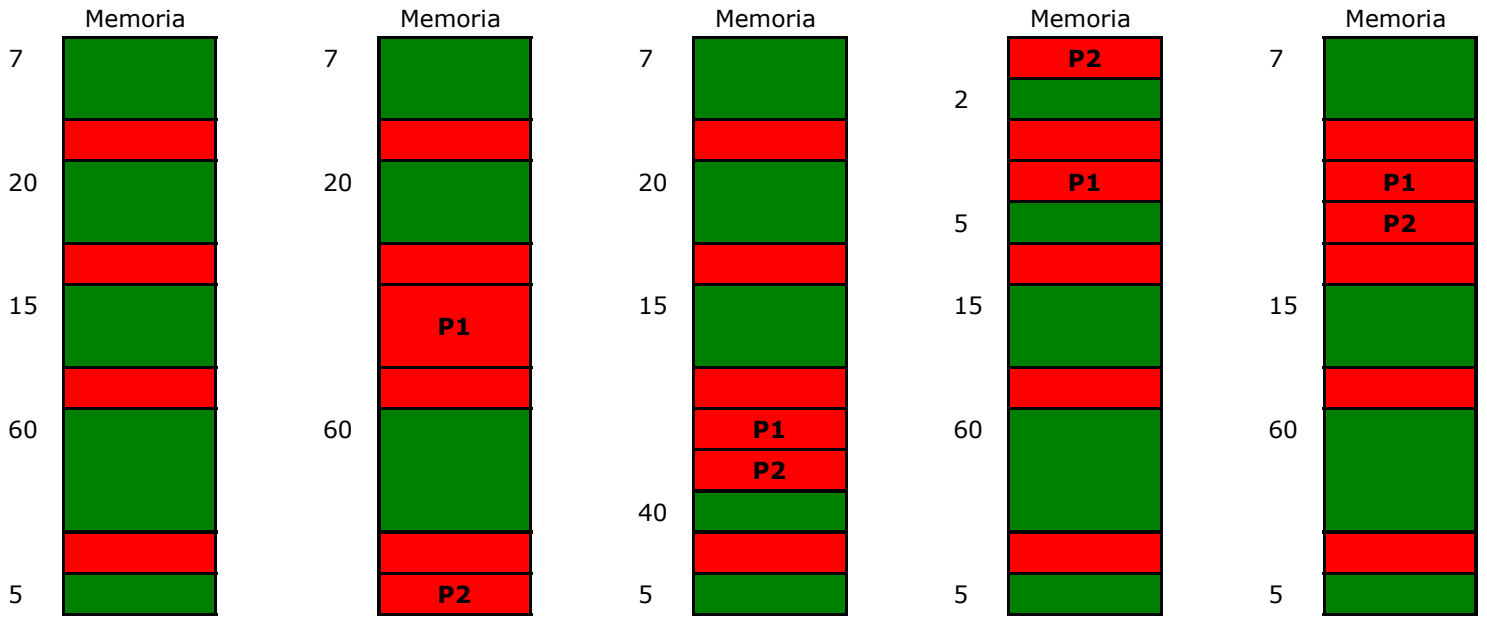


3.3 Modelos de mem.

Particiones variables

- Políticas de asignación
  - Best-Fit: Busca el bloque al que mejor se ajuste el proceso
    - –Genera un bloque libre más pequeño posible → frag. ext.
  - Worst-Fit: Busca el bloque más grande de todos
    - –Genera el bloque libre más grande posible → inanición pr. grandes
  - First-Fit: El primero que encuentra. Comportamt. Intermedio
  - Next-Fit: El primero que encuentra a partir del último asignado

Peticiones de memoria:
P1: 15, P2: 5



3.3 Modelos de mem.

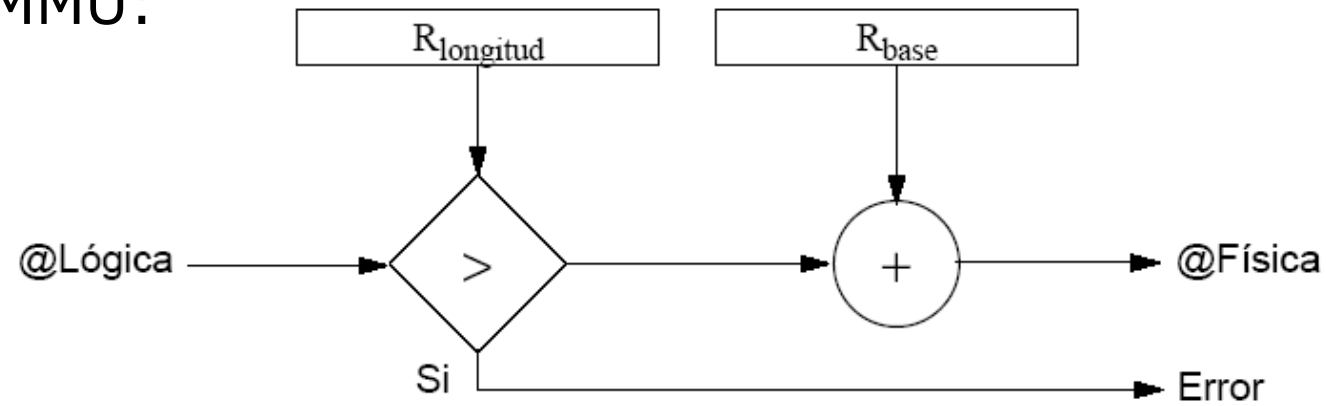
multipr.|no resid.|inmóvil|contiguo|entero

- Modelo no residente
  - Modelo residente: limitado por la mem. física
  - Alternativa: descargar procesos al HD
    - Operaciones swap-out (Mem→HD) y swap-in (HD→Mem)
- Ventajas:
  - Aumenta el grado de multiprogramación
  - Memoria dinámica (malloc)
    - Un proceso puede crecer descargando a algún vecino
- Inconvenientes
  - Sigue siendo inmóvil:
    - El proceso descargado luego se carga en el mismo sitio
  - Coste de espacio en HD y tiempo de swap
  - Procesos no descargables
    - Ej: si el DMA necesita acceso a los datos para terminar E/S

3.3 Modelos de mem.

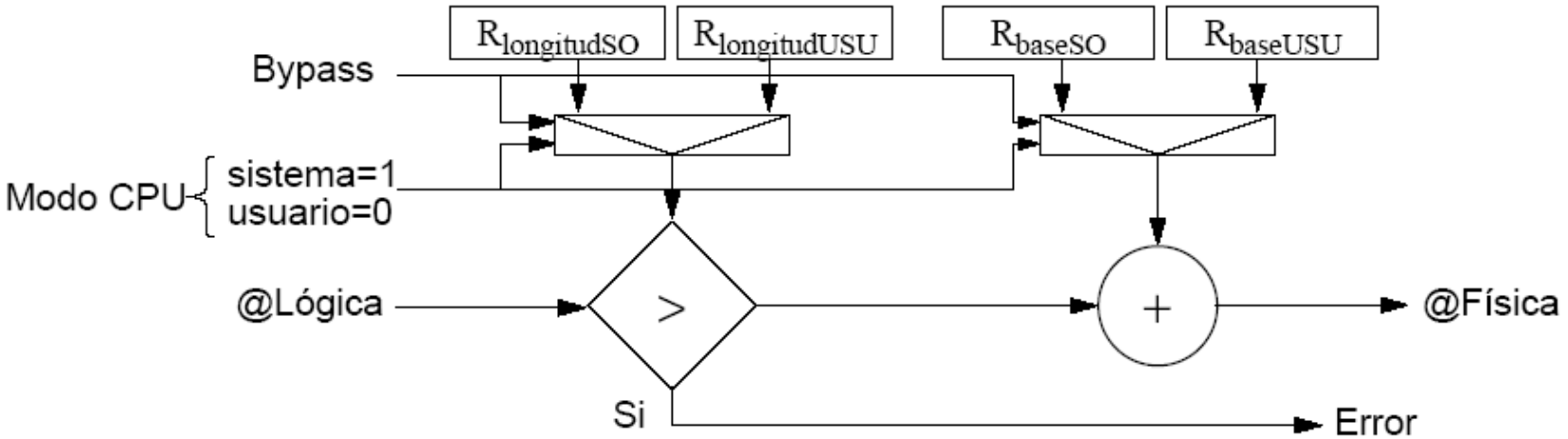
Modelo móvil

- Podemos cambiar un proceso de sitio (reubicar)
  - Esp. Lóg. Prog = Esp. Lóg. Proc. ≠ Esp. Físico
- Ventajas
  - Se puede hacer "garbage collection" (desfragmentar)
  - Mejor aprovechamiento de la memoria
  - Aumenta el grado de multiprogramación máximo
  - Memoria dinámica más fácil de implementar
    - Si un proceso quiere crecer → lo movemos a una zona mayor
- MMU:



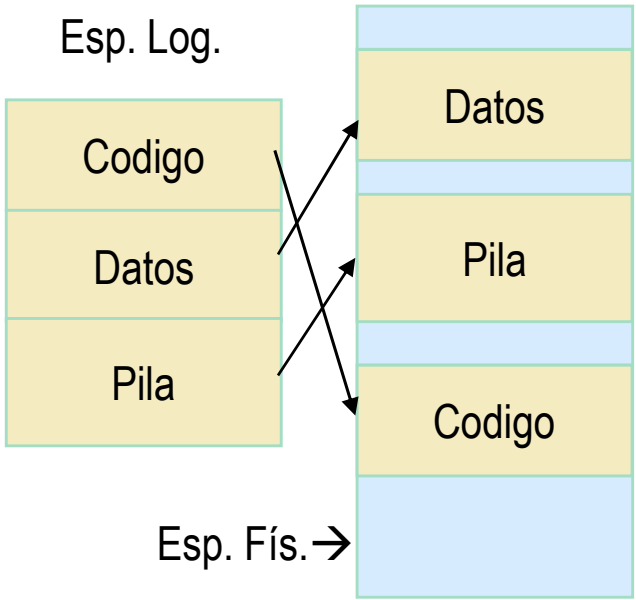
3.3 Modelos de mem.

- Modelo móvil: detalles de la MMU
  - Dir. base y longitud almacenadas en el PCB
  - En los cambios de contexto → actualizar regs
  - El SO tiene un espacio distinto (regs propios)
    - En un syscall se pasa a modo sist. → acceso a mem del SO
  - El SO debe poder acceder al espacio del proceso
    - La syscall debe poder acceder a los datos del proceso
    - Señal Bypass: en modo sistema puedes usar los registros base y longitud del proceso

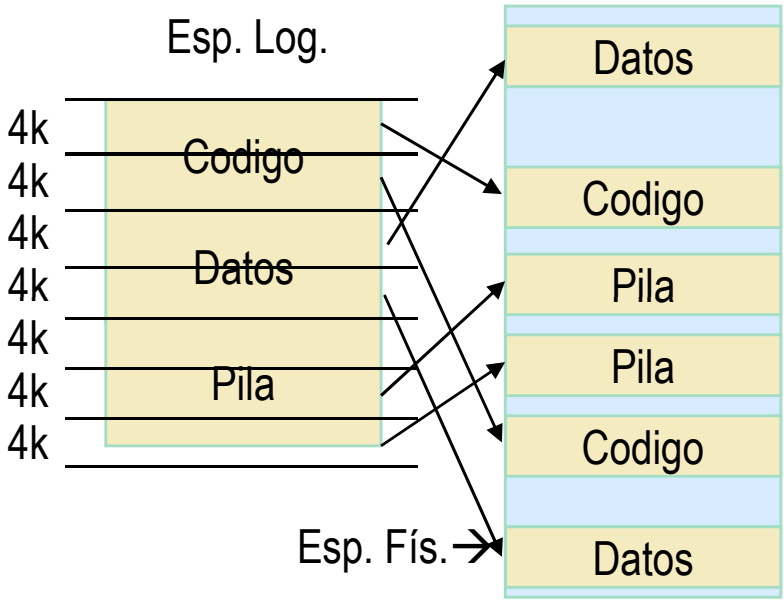


3.3 Modelos de mem.

- Modelo no contiguo: apartado 3.4
  - El proceso no tiene por que ocupar posiciones consecutivas en el espacio físico
    - Puede estar repartido en bloques disjuntos por la memoria
    - El espacio lógico sigue siendo contiguo. El físico no.



Segmentación



Paginación

3.3 Modelos de mem.

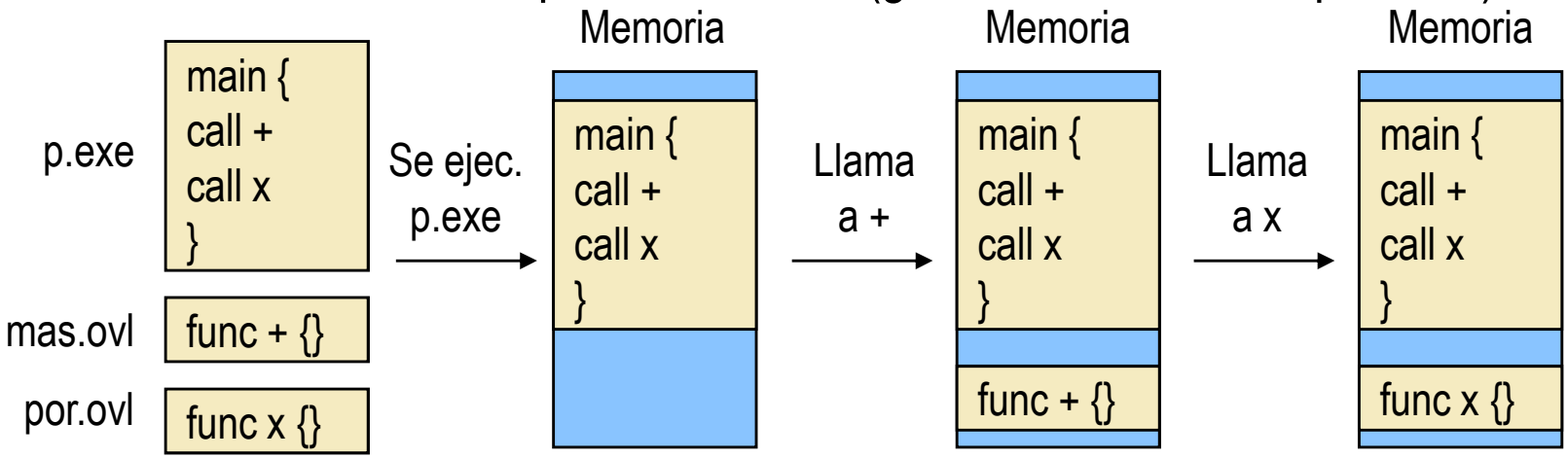
multipr.|no resid.|móvil|no cont.|no entero

- Modelo no entero
  - La imagen del proceso no tiene por que estar cargada en memoria completamente
  - Basta con mantener las zonas en uso
  - Se basa en los principios de localidad
  - Ventajas
    - Se pueden ejecutar procesos más grandes que la mem
    - Caben más procesos en memoria (más grado de multipr.)
    - Swapping de trozos (no de procesos completos)
  - Posibilidades
    - Overlays
    - Librerías con enlace dinámico (DLL)
    - Memoria Virtual (apartado 3.5)

3.3 Modelos de mem.

multipr.|no resid.|móvil|no cont.|no entero

- Overlays (recubrimientos)
  - Se divide el progr. en bloques (un .exe y .ovl's)
    - En el fichero .exe está el programa principal
    - En los .ovl's puede haber conjuntos de rutinas
  - Cuando se ejecuta el programa se carga el .exe
    - Antes de llamar a una rutina → cargar el .ovl correspondiente
    - El linker debe incluir el código de carga antes de la 1ª llamada
    - Un .ovl puede reemplazar a otro en memoria (lo recubre)
    - Método transparente al S.O. (gestionado desde el proceso)



3.3 Modelos de mem.

multipr.|no resid.|móvil|no cont.|no entero

- Dynamic Link Libraries (DLL)
  - Con enlace estático:
    - El ejecutable incluye el código de todas las rutinas
  - Con dinámico, el ejecutable no incluye las rutinas
    - A cambio se inserta un código que carga la rutina en mem. (desde un fichero .DLL) cuando ésta rutina es llamada por 1ª vez
  - Ventajas
    - Los ejecutables ocupan menos y se elimina la redundancia
      - El código de las rutinas está sólo 1 vez tanto en HD como en mem.
    - Si modificas las rutinas los ejecutables no se ven afectados
      - Siempre que mantengan el mismo interfaz (versiones de DLLs)
  - Inconveniente
    - Se pierde la portabilidad del ejecutable
      - Ahora el ejecutable necesita las DLLs correspondientes

3.4 Paginación y segmentación

3.4 Paginación y seg.

motivación

- Motivación del modelo **no contiguo**
  - Inconvenientes del modelo contiguo:
    - Encontrar una partición donde quepa el proceso contiguo en mem
    - Si no hay un hueco lo suficientemente grande:
      - Swap-out de otros proc. en el modelo no residente: Poco práctico
      - Desfragmentar la memoria en el modelo móvil: Alto coste temporal
      - En otro caso, no se puede crear el proceso: Poca flexibilidad
  - Solución: fragmentar el E.L. del proceso
    - Mapear el E.L. del proceso en varios fragmentos del E.F.
  - Esquema:
    - Paginación
      - Implementación. Criterios de diseño
      - Optimizaciones: Paginación multinivel. Tabla invertida. TLB
    - Segmentación
    - Segmentación paginada

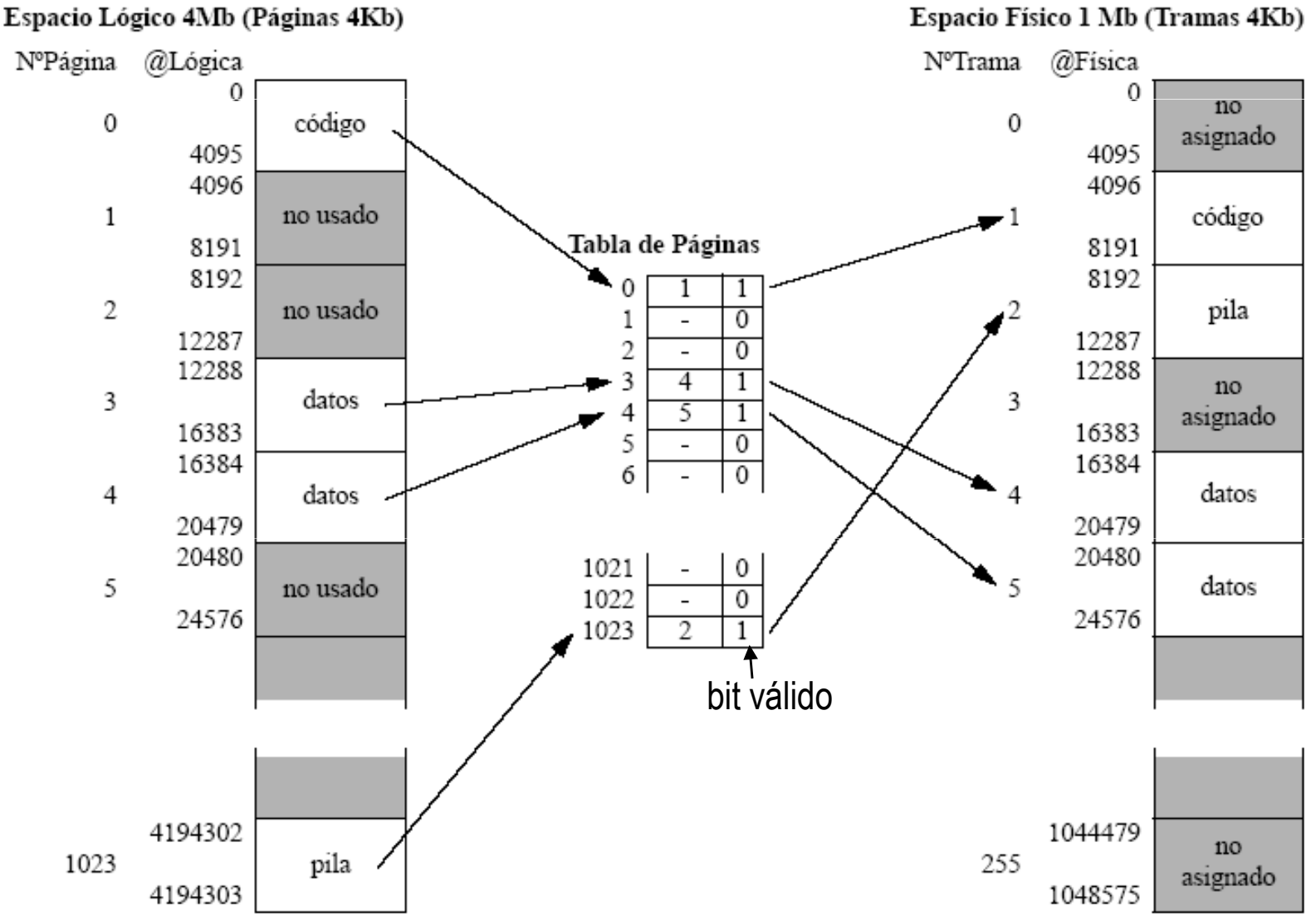
3.4.1 Paginación

Comentarios sobre la implementación

- Sólo se traducen las zonas del E.L. posible en uso
  - En la tabla de páginas, el bit válido a 0 indica una página sin uso
- Máxima flexibilidad particionando E.L en bytes
  - No habría fragment. interna en las páginas parcialmente usadas
  - Pero eso es inviable: Tabla de páginas prohibitiva
  - Solución:
    - Unidad asignación no contigua: **página** (tamaño potencia de 2)
    - Espacio Lógico del proceso dividido en páginas
    - Espacio Físico dividida en marcos o tramas (tam. tram.=tam. pág.)
- Traducción de direcciones
  - Dirección lógica: nº página + desplazamiento
  - Una tabla de páginas (TP) para cada proceso:
    - Relaciona cada página con el marco que la contiene
    - La MMU usa la TP para traducir direcciones lógicas a físicas.

3.4.1 Paginación
implementación

Implementación de la paginación



3.4.1 Paginación

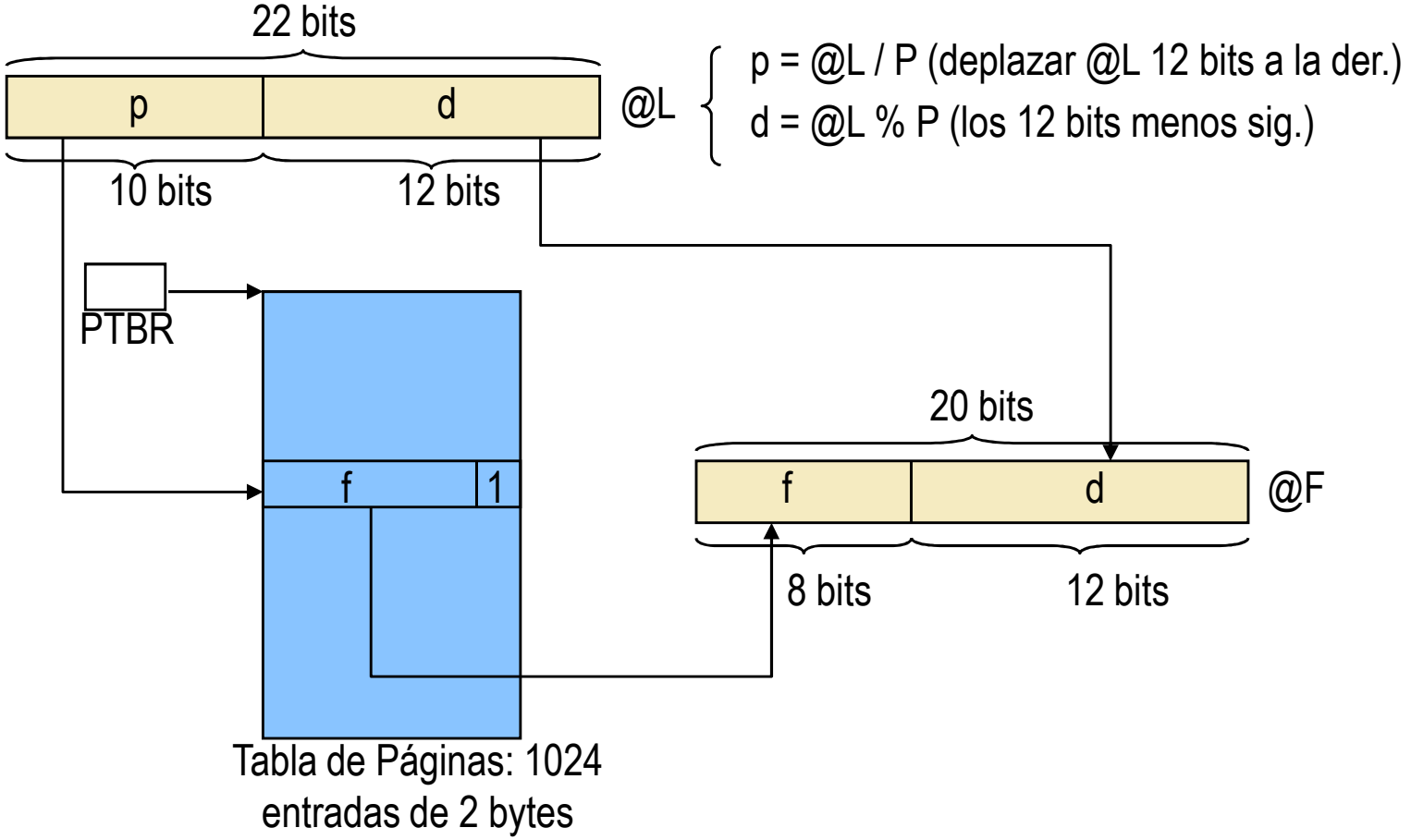
notación

- Notación necesaria
  - @L: Dirección lógica
  - @F: Dirección física
  - d: desplazamiento (byte dentro de la página)
  - p: número de página
  - f: número de trama (marco o página física)
  - P: tamaño de página
  - PTBR: Page Table Base Register
    - Registro base de la tabla de páginas
    - Apunta al comienzo de la TP del proceso (almacenada en mem)
  - Rangos de bits en las direcciones:
    - [M..N] especifica los bits del M al N inclusive
    - Ejemplo: @F[19..0] identifica los bits 19 al 0 de la dir. física

3.4.1 Paginación

ejemplo

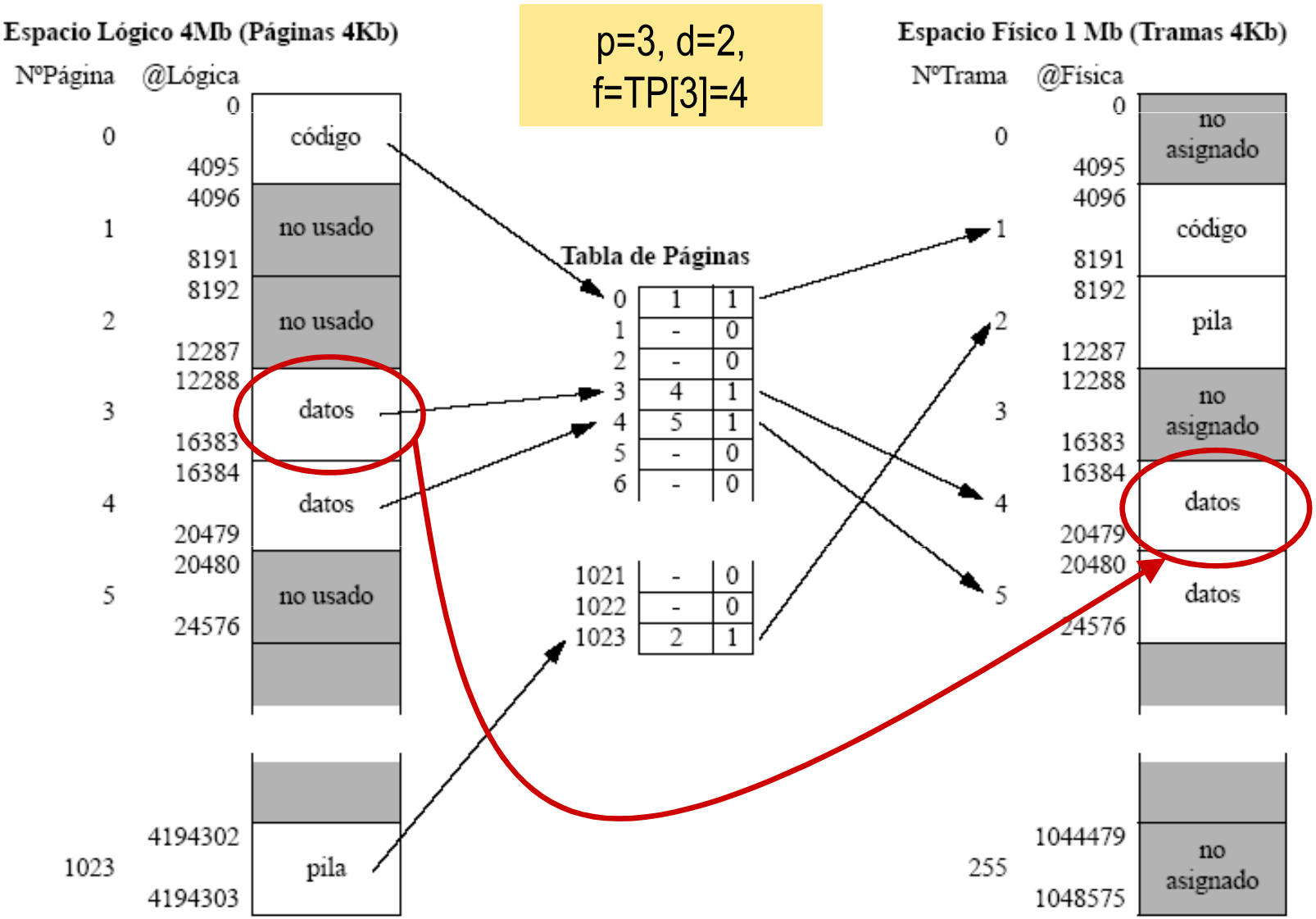
- Ejemplo de traducción:
  - E.L. de 4Mb. → @L de 22 bits (@L[21..0])
  - E.F. de 1Mb. → @F de 20 bits (@F[19..0])
  - P = 4Kb = 4096bytes = 2^{12}



3.4.1 Paginación

ejemplo

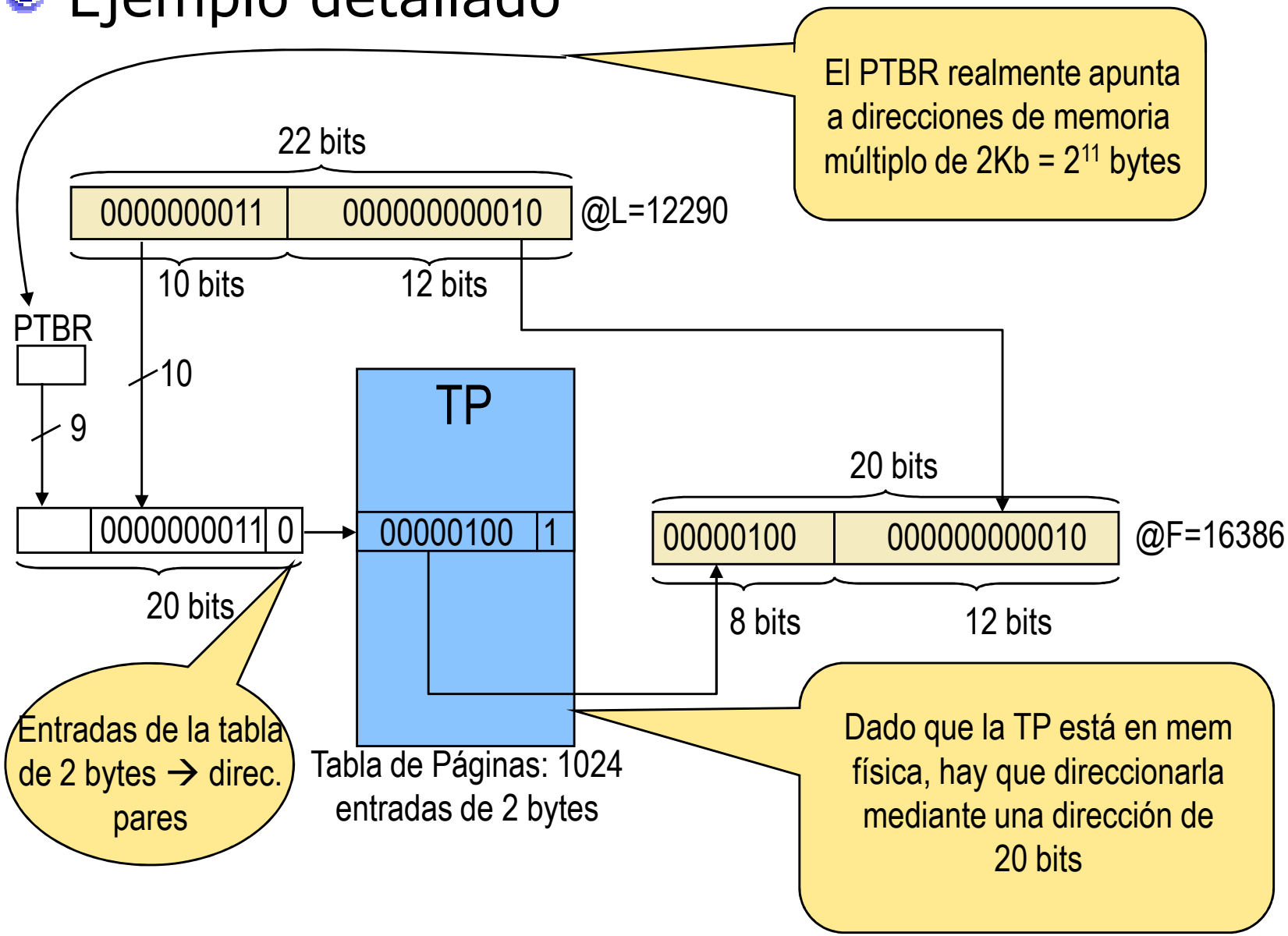
- Ejemplo: @L = 12290 → @F = 16386



3.4.1 Paginación

ejemplo

Ejemplo detallado

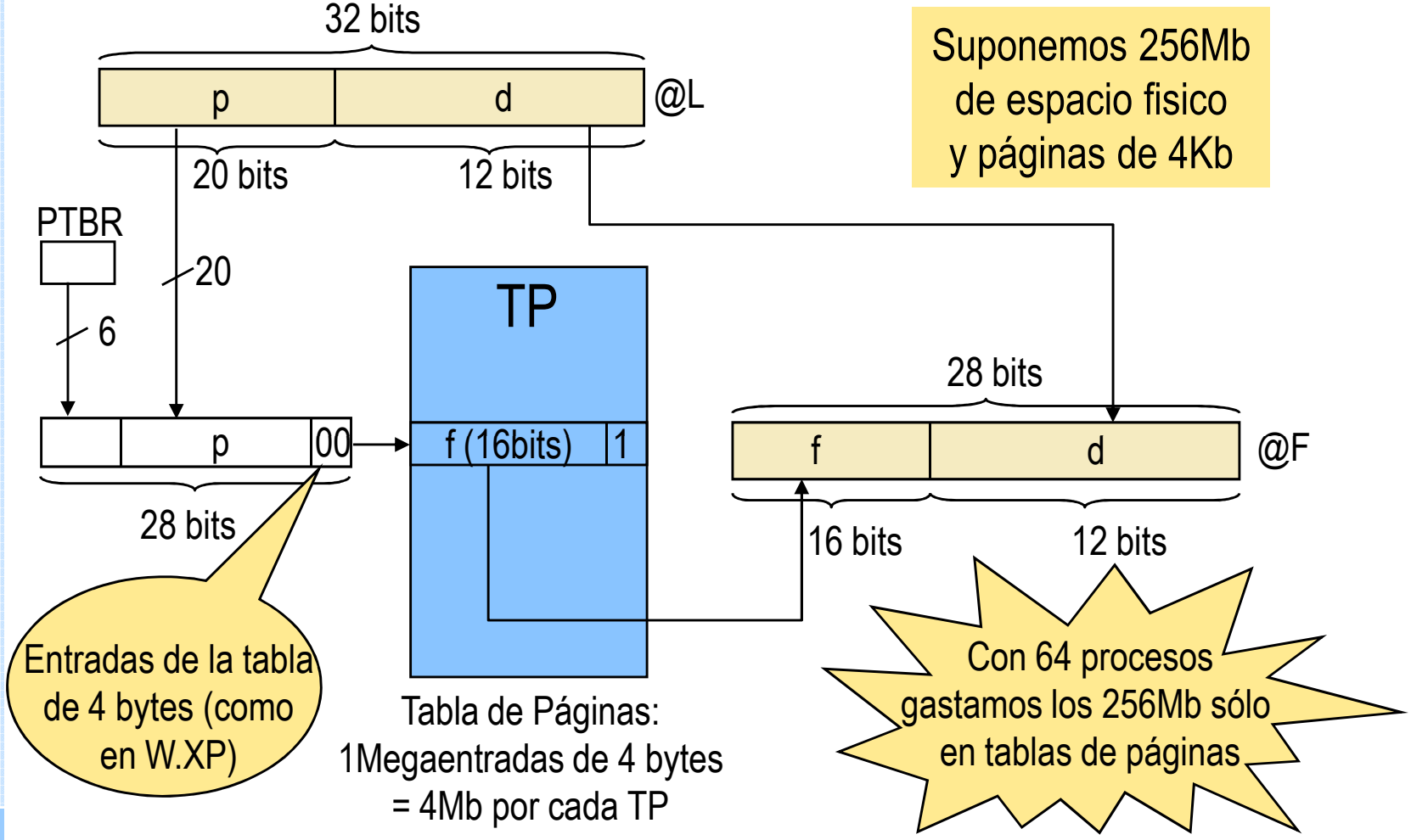


3.4.1 Paginación
tamaño de la TP

- Tamaño de la tabla de páginas (TP)
  - Nº de entradas = nº de páginas del espacio lóg.
    - nº entradas = Tamaño E.L. / Tamaño de página
    - Páginas mas grandes:
      - menos entradas, más fragmentación interna en los procesos.
    - Páginas más pequeñas:
      - más entradas, menos fragmentación interna.
  - Bytes de cada entrada (potencia de dos)
    - Campo dirección de trama (f) de ancho = $\log_2$(Tam E.F./P)
    - Bit validez (V) : a 1 si la pág. tiene trama asociada
    - Permisos de acceso: R (Read), W (Write), X (eXecution)
    - Bit de acceso (A): a 1 si se accede la página
    - Bit de modificación (M/D): a 1 si se escribe en la página
    - Bit de presencia (P): usado en memoria virtual (ap. 3.5)

3.4.1 Paginación
tamaño de la TP

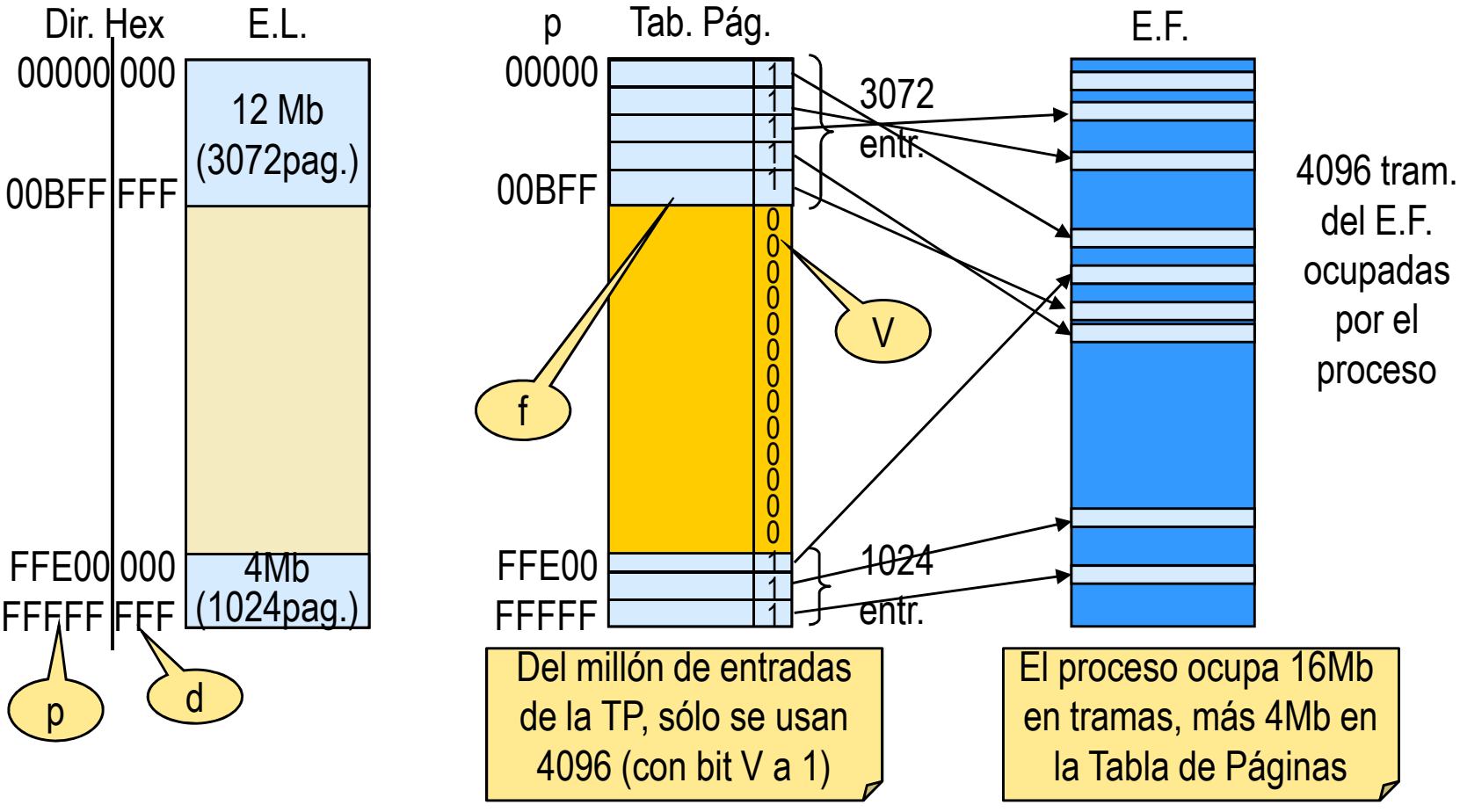
- Desperdicio de mem. en tablas de páginas
  - Sobre todo cuando el E.L. es grande
  - Ejemplo: E.L. de 32 bits (4Gb, como en Pent.4)



3.4.1 Paginación

tamaño de la TP

- Pero además: la TP está desperdiciada
  - Los procesos no usan todo el E.L. (bits V = 0)
  - Ejemplo: Un proceso usa 16Mb del E.L.
    - 12Mb para el código y los datos y 4Mb de pila al final del E.L.



3.4.1 Paginación
problemas de la paginación

- Por tanto: problemas de la paginación:
  - La TP es proporcional al espacio lógico
    - Tiene tantas entradas como páginas tiene el E.L.
    - Además hay una tabla para cada proceso
    - Solución:

Tabla de páginas invertida

  - La TP no se aprovecha
    - En el ejemplo anterior, se usan 16Kb de los 4Mb de TP (el 0.4%)
    - Solución:

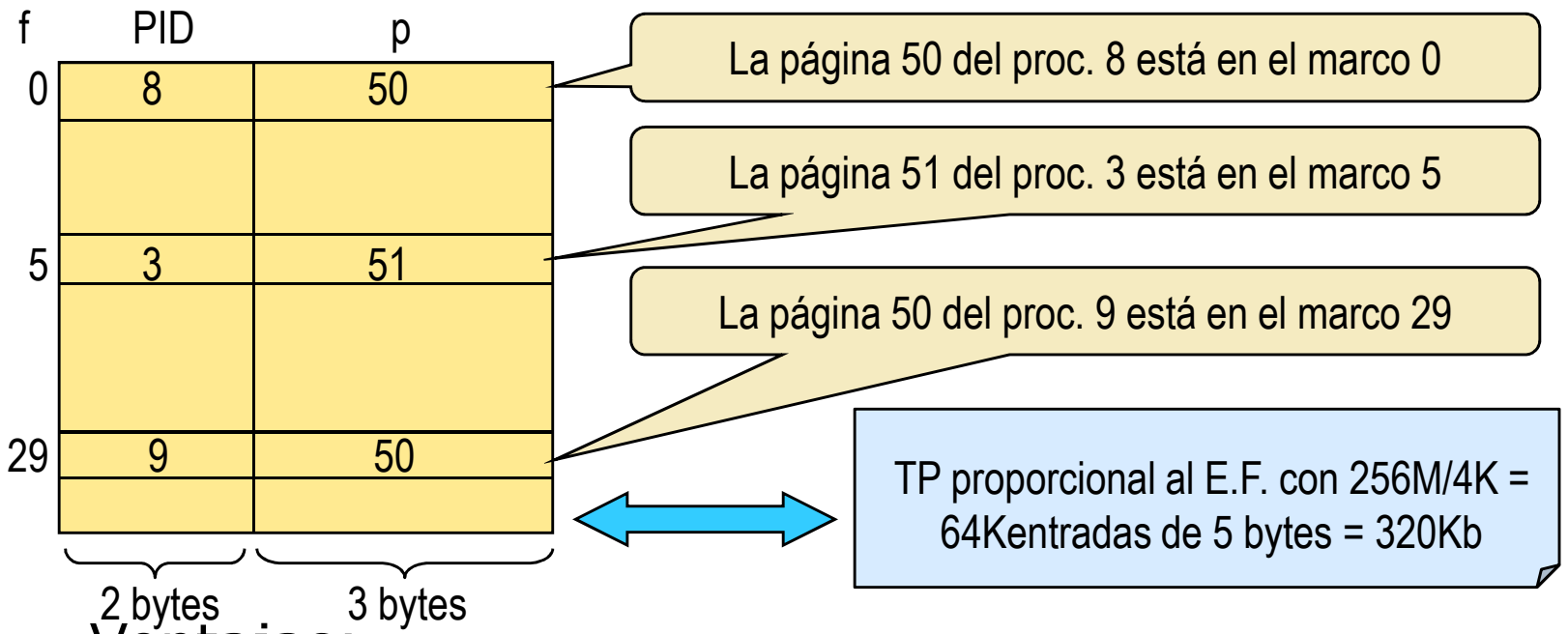
Paginación multinivel

  - Se ralentizan los accesos a memoria
    - Un acceso a memoria (fetch, load, store) se traduce en 2 acces.
    - El 1º para leer en la TP el nº de trama (f) y el 2º accede a mem.
    - Solución:

Translation Lookaside Buffer (TLB)

3.4.1 Paginación
tabla invertida

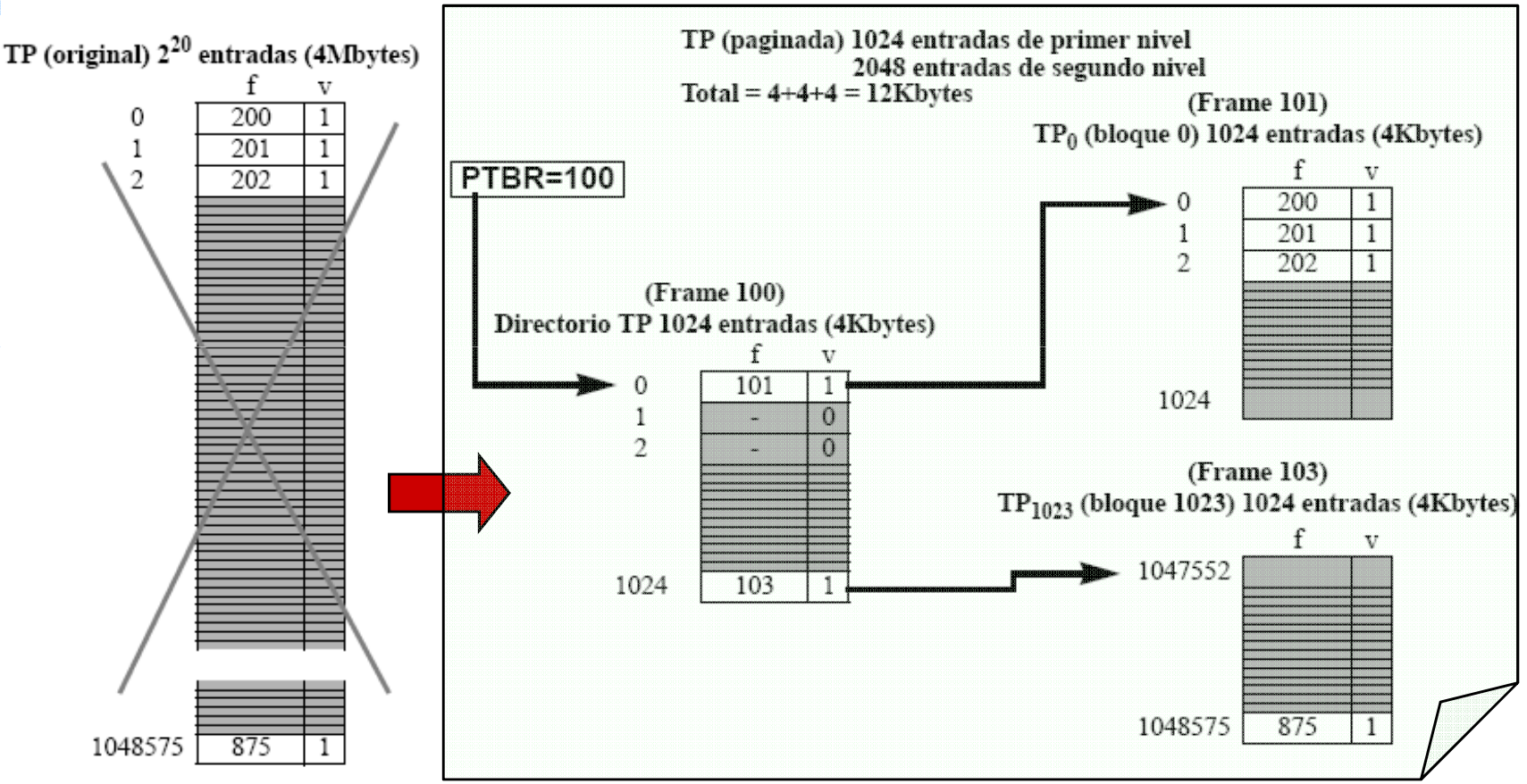
- Tabla de páginas invertida
  - En lugar de buscar qué trama corresponde a cada página → qué pág. corresponde a cada trama



- Ventajas:
  - Una sola tabla de 320Kb (proporcional al E.F. en lugar de al E.L.)
  - Se usa en procesadores de 64bits (con E.L. de 2^{64}bytes)
- Inconvenientes:
  - Tiempo para buscar en la tabla (aunque se implemente hash)
  - Dificultad para compartir páginas

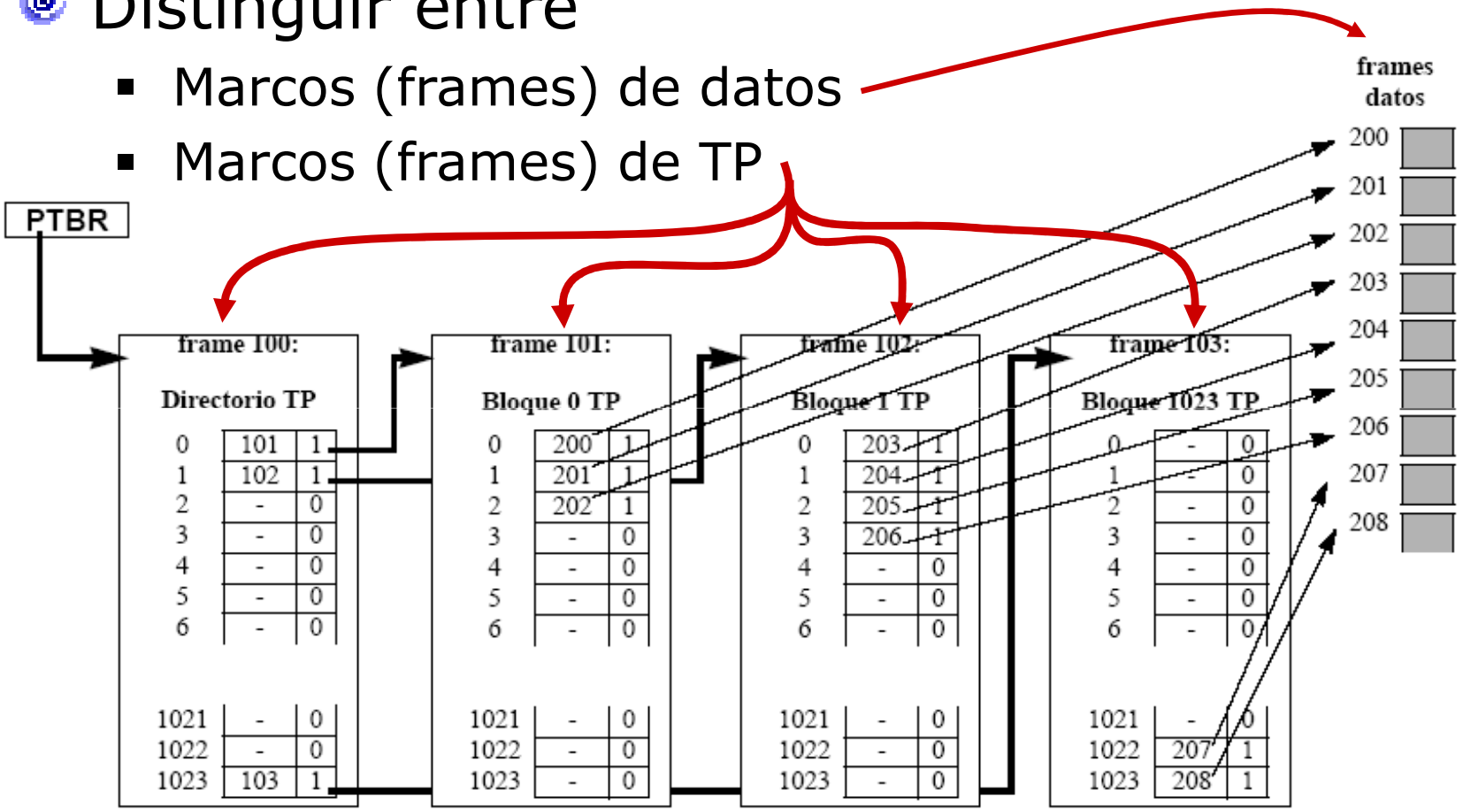
3.4.1 Paginación
paginación multinivel

- Paginación en dos niveles:
  - Idea: paginar la tabla de páginas
    - Dividimos la TP original en bloques de una página de tamaño
    - Si no hay ninguna entrada válida en el bloque→ bloque no val.



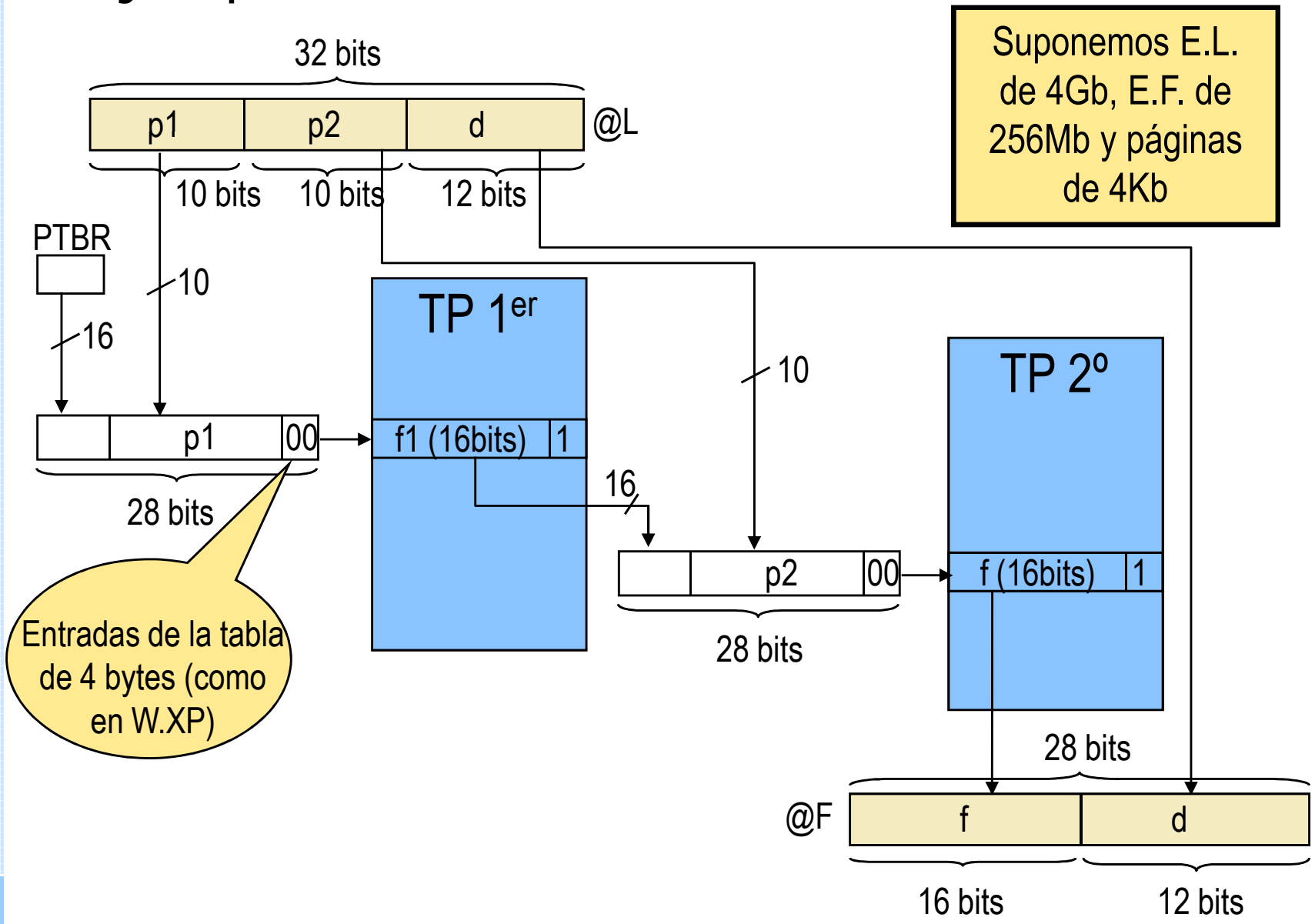
3.4.1 Paginación
paginación multinivel

- Ahorro de espacio en TP de 2º nivel
  - La TP de 1er nivel tiene que estar en mem
  - Sólo las TP de 2º nivel válidas ocupan mem
- Distinguir entre
  - Marcos (frames) de datos
  - Marcos (frames) de TP



3.4.1 Paginación
paginación multinivel

- Ejemplo de traducción detallado



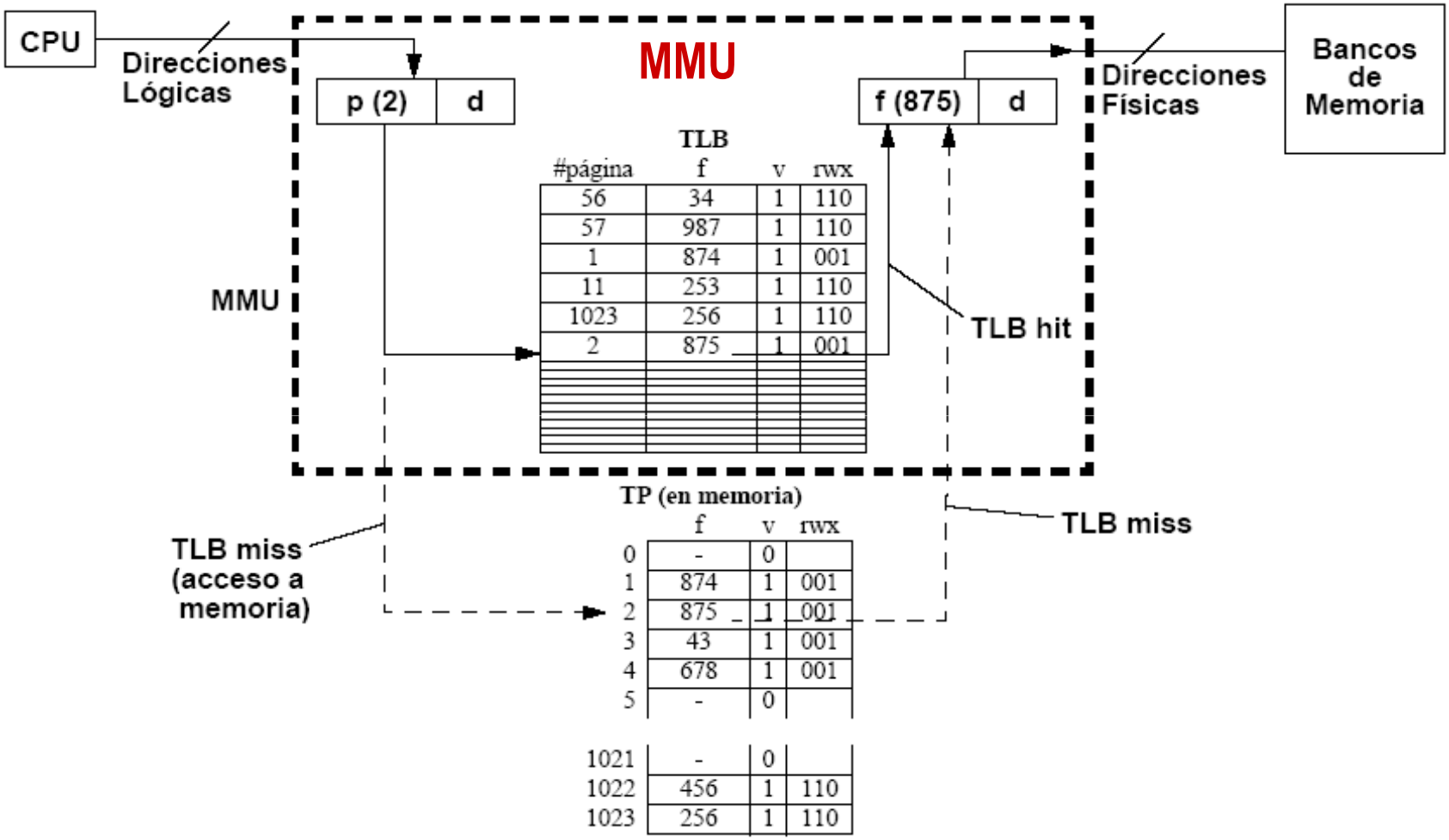
3.4.1 Paginación
paginación multinivel

- Ventajas
  - Soluciona el problema del desperdicio de TP
    - Las tablas ocupan menos memoria
    - Las tablas están mejor aprovechadas
  - En el ejemplo del proceso de 16M (12+4)
    - En la TP 1er nivel se usan 4 entradas (3 primeras y la última)
    - Usamos 4 TP de 2º nivel con todos los bit V a 1 (aprovechadas)
    - Total: 5 TPs x 4Kb cada una = 20 Kb por proceso (antes 4M !!)
- Inconvenientes: perdida de rendimiento!
  - Un acceso a memoria se traduce en 3
    - 1º, para acceder a la TP de primer nivel
    - 2º, para acceder a la TP de segundo nivel
    - y 3º, para realizar el acceso al marco y leer/escribir el dato
- Generalizable a 3 o más niveles

3.4.1 Paginación

TLB

- Translation Lookaside Buffer (TLB)
  - Soluciona el problema del tiempo de traducción
  - TLB: caché con los pares (p,f) más frec. usados



3.4.1 Paginación

TLB

- Implementación de la TLB
  - Es una caché y por tanto se implementa como tal
- Cambio de contexto:
  - TLB sin identificación de proceso → invalidar las entradas de la TLB
  - TLB con identificación de proceso → pueden coexistir entradas de varios procesos
- TLB gestionada por Software:
  - El SO introduce los valores en la TLB (no la MMU)
  - Mayor flexibilidad para definir tablas de páginas (no impuesta por hardware). Más lento.
- Ejemplo: Pentium 4
  - TLB de código: 128 entradas (4Kb)
  - TLB de datos de 4Kb en dos niveles: TLB datos L1 → 8 entradas de 4 vías (dos conjuntos), TLB datos L2 → 64 entradas, tot. asociativa

3.4.2 Segmentación

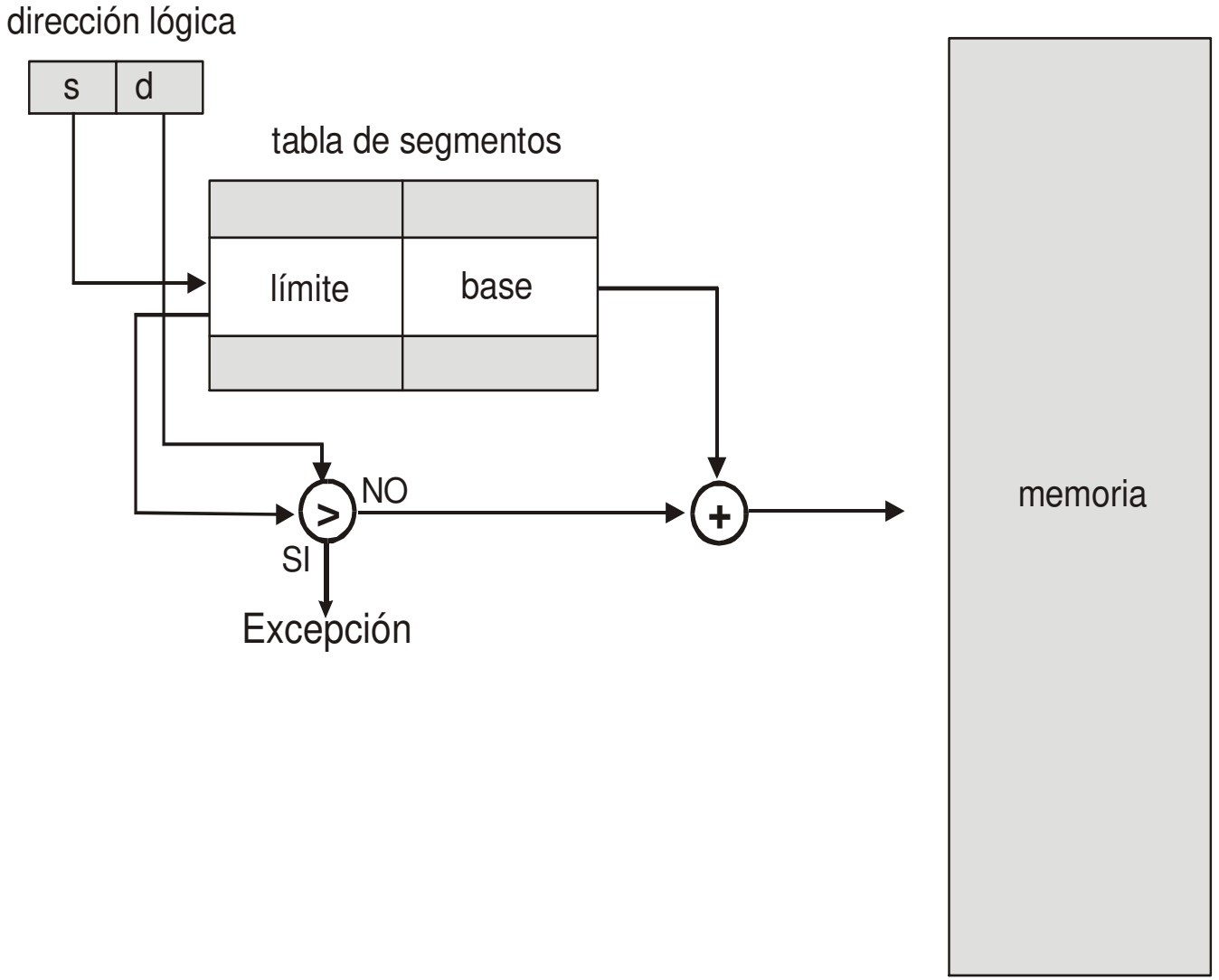
segmentación

- Particionar E.L. en seg. de tamaño variable
  - Generalización de reg. base y límite
  - @L: nº de seg + desplazamiento
- MMU usa una tabla de segmentos (TS)
  - El S.O. mantiene una TS por proceso
    - En cambio de contexto se notifica a MMU cuál debe usar
- Cada entrada de TS contiene (entre otros):
  - Registro base y límite del segmento
  - Protección: RWX
- S.O. mantiene inf. sobre estado de la mem:
  - Estructuras de datos que identifiquen huecos y zonas asignadas
  - Fragmentación externa

3.4.2 Segmentación

segmentación

Traducción de direcciones con seg.



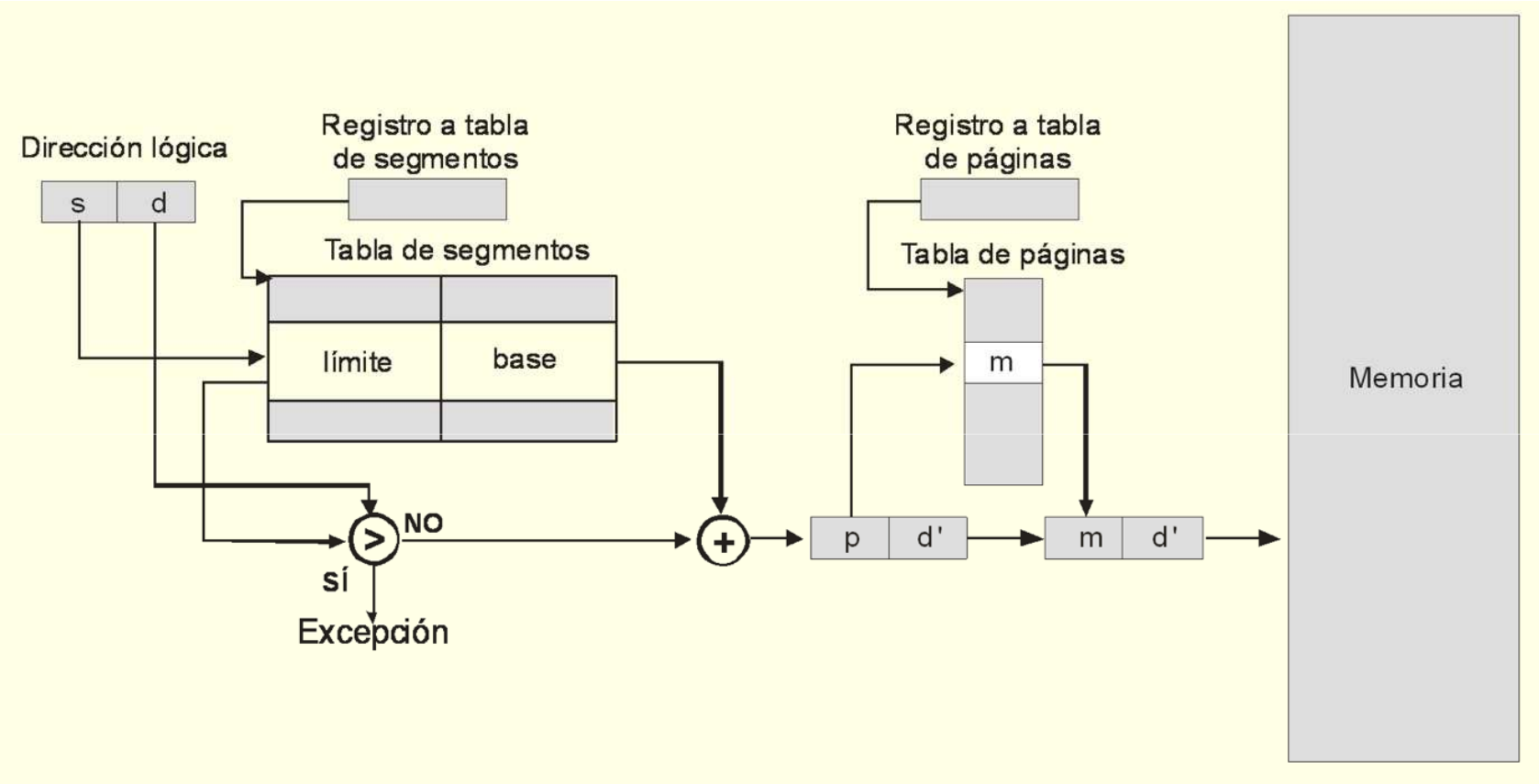
3.4.2 Segmentación
paginación vs. segmentación

- Comparación entre segment. y paginación
  - Tamaño de la dirección base
    - Pag. : sólo núm. de pag. (f). Concatenar con despl (d)
    - Seg.: dir. física completa que hay que SUMAR con d
  - Solapamiento (compartir memoria)
    - Pag.: Se pueden solapar páginas completas
    - Seg.: Se pueden solapar parcialmente segmentos físicos
  - Asignación de memoria
    - Pag.: Muy sencillo:
      - Un proceso necesita n páginas → buscar n huecos (tramas libres)
      - No hay frag. ext. pero si frag. interna (media de P/2 por proceso)
    - Seg: Muy complicado:
      - Un proceso necesita n segmentos, cada uno de un tamaño → Buscar n huecos del tamaño adecuado lo cual puede implicar hacer garbage collection (desfragmentar la memoria)
      - No tiene frag. interna pero sí fragmentación externa

3.4.3. Segmentación pag.

segmentación paginada

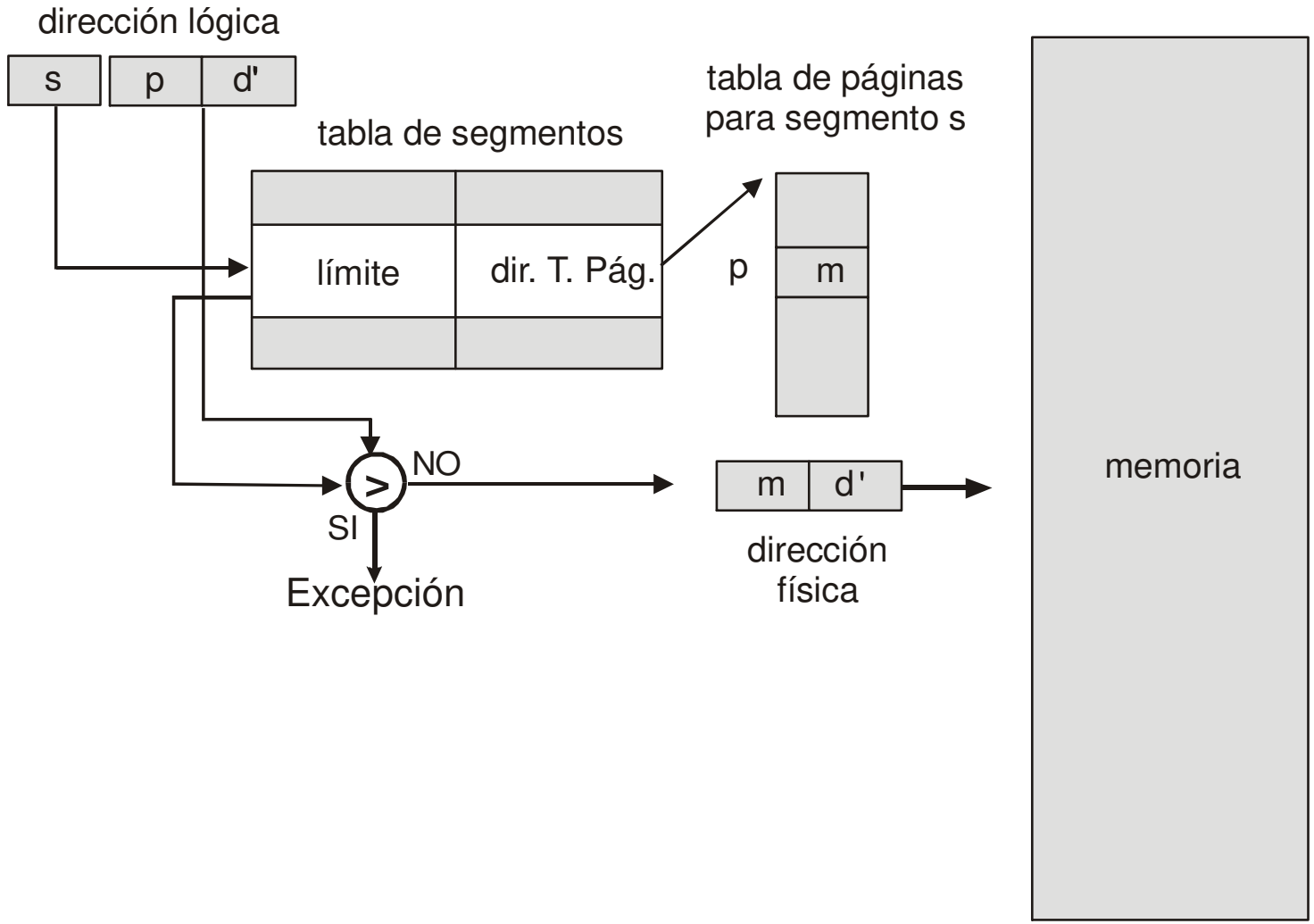
- Modelo híbrido
  - El E.L. del proceso se divide en segmentos
  - Cada segmento se divide en páginas



3.4.3. Segmentación pag.

segmentación paginada

- Modelo híbrido
  - Una tabla de páginas por cada segmento



3.5 Memoria virtual

3.5 Mem. virtual

motivación

- Motivación
  - Modelo no entero
  - El proceso no necesita estar cargado completamt.
    - Mantener en mem. la zona actualmente en uso
      - Pueden ser páginas (lo habitual) o segmentos.
    - El resto de las zonas del proceso reside en el swap
      - Partición (unix) o fichero (windows) del disco duro
  - Funciona gracias a los principios de localidad
    - Espacial y temporal
    - Resumidos en la regla 90/10 de localidad
  - Ventajas
    - Aumenta el grado máximo de multiprogramación
    - Los procesos no están limitados por la memoria física
    - Disminuye el tiempo de swapping respecto del modelo no reside.
      - Ahora descargas/cargas un trozo del programa (no todo)

3.5 Mem. virtual
implementación

- Implementación (mem. virtual paginada)
  - Cuando nace un proceso se crea su E.L.
  - Cuando el proceso emite una @L
    - Si está presente en E.F. (ocupa una trama) → HIT
    - Si no está presente → MISS (excepción de fallo de página)
      - El S.O. toma el control. Se ejecuta la RTI para resolver el fallo
      - La RTI trae la página (swap-in) del swap y la carga en el E.F.
      - Posible reemplazo en el E.F. si éste está lleno (swap-out).
  - Nota: las páginas de código no ocupan swap
    - Ya están en la zona "normal" del HD en el fichero ejecutable
  - Posibilidades de implementación
    - Por demanda: Sólo cargas las pag. que se piden
      - La 1ª instrucción de un programa ya genera un fallo de página
    - Prepaginación: Al crear proceso cargas algunas páginas
    - Prefetch: En cada fallo de página, cargas páginas adicionales

3.5 Mem. virtual
requerimientos HW

- Requerimientos HW
  - Tabla de páginas con mas bits
    - Bit P (presencia) indica si la página está o no cargada
    - Bit D (dirty o modificado) a 1 si la pág. se ha cambiado en mem.
      - Al reemplazar una página modificada hay que hacer swap-out
  - Zona de swap en el HD y DMA
    - La zona de swap puede ser
      - Un fichero (flexibilidad de tamaño) como en windows
      - Una partición (sistema de ficheros específico = rendimiento) Linux
  - Mecanismo de interrupciones
    - La MMU tiene que lanzar una excepción cuando P=0
    - Recomenzar instrucciones. Ejemplo:
      - Una instrucción lw r5,0(r4) genera un miss en la búsq. de opernd.
      - La instrucción no se ha completado. Se ejecuta la RTI (swap-in)
      - Recomenzar la instrucción después al volver de la RTI

3.5 Mem. virtual
requerimientos SW

- Requerimientos SW
  - RTI de tratamiento de fallos de página:

```
SI V=0 o permisos insuficientes ENTONCES
        notificar error o terminar proceso
SI V=1 y P=0 ENTONCES
  buscar una trama libre en memoria
  SI NO HAY trama libre ENTONCES
        Ejecutar algoritmo de reemplazo (sel. pág victima)
        SI pág. VICTIMA tiene D=1 (modificado) ENTONCES
            SWAP-OUT de la víctima (varios milisegundos)
            (durante el swap-out → El proc. se bloquea)
  SWAP-IN de la página que falló (proceso bloqueado)
  Actualizar tabla de página
  Restaurar el proceso bloqueado y recomenzar instr.
```

3.5 Mem. virtual
algoritmos de asignación y reemplazo

- En caso de memoria llena → sel. víctima
  - Asignación: selec. de varias páginas candidatas
  - Reemplazo: selec. víctima de entre las candidatas
  - Working-set: Asignación+reemplazo (W.XP)
- Algoritmos de asignación
  - Local: las tramas se reparten estáticamente
    - Según criterios a cada proceso se le asignan n tramas
    - En caso de F.P. → reemplazar de sus propias tramas
    - Inconvenientes:
      - Cuando se crea un proceso → recalcular reparto de tramas
      - Poco adaptativo a la localidad de cada proceso
  - Global: las tramas se asignan al prc. que las pide
    - Todas las tramas son candidatas a ser víctima
    - Se adapta mejor a la localidad de cada proceso
    - Inconveniente: un proc. con + F.P. se apodera de + tramas

3.5 Mem. virtual
algoritmos de reemplazo

- Algoritmos de reemplazo
  - Selecciona la víctima de entre los candidatos
  - Minimizar el nº de F.P. en el futuro
  - Algoritmos
    - Optimo: necesita conocer el futuro. No implementable
    - Aleatorio: selecciona cualquier víctima
    - FIFO: Primero en entrar, primero en salir. Anomalía de Belady
    - LRU: Least Recently Used
    - Aproximación LRU
    - NFU: Not Frecuently Used
    - NRU: Not Recently Used
    - LFU: Least Frecuently Used
    - Clock o segunda oportunidad
    - MFU: Most Frecuently Used

3.5 Mem. virtual

algoritmos de reemplazo

- Óptimo:
  - Selecciona como víctima a la página que
    - Nunca más vas a necesitar en el futuro
    - O la que vas a necesitar dentro de más tiempo
  - Necesitas conocer el futuro!! → no implementable
  - Ejemplo
    - Tenemos 3 tramas en memoria física
    - La siguiente secuencia de accesos a estos nº de páginas:

p:	1	2	3	4	1	2	5	1	2	3	4	5
Tramas	1	1	1	1	1	1	1	1	1	3	4	4
		2	2	2	2	2	2	2	2	2	2	2
			3	4	4	4	5	5	5	5	5	5
Miss	x	x	x	x			x			x	x	

3.5 Mem. virtual

algoritmos de reemplazo

- FIFO:
  - Selecciona como víctima a la página que
    - lleva más tiempo en memoria (la 1ª que entra, 1ª que sale)
  - Dos posibles implementaciones:
    - Apuntar a cada trama la hora de llegada
    - Implementar una cola por SW
      - Las páginas entran pon un extremo y salen por el otro
  - Ejemplo:3 tramas en memoria física

p:	1	2	3	4	1	2	5	1	2	3	4	5
Tramas	1	1	1	2	3	4	1	1	1	2	5	5
		2	2	3	4	1	2	2	2	5	3	3
			3	4	1	2	5	5	5	3	4	4
Miss	x	x	x	x	x	x	x			x	x	

3.5 Mem. virtual

algoritmos de reemplazo

- FIFO: Anomalía de Belady
  - En principio: a + nº de tramas → - nº de fallos
  - Sin embargo, FIFO sufre de la anomalía Belady
    - Puede que incluso con + tramas → tengas más fallos de pág.
  - Ejemplo:
    - 4 tramas en memoria física (10 miss). Antes con 3 tr. (9 miss)

p:	1	2	3	4	1	2	5	1	2	3	4	5
Tramas	1	1	1	1	1	1	2	3	4	5	1	2
		2	2	2	2	2	3	4	5	1	2	3
			3	3	3	3	4	5	1	2	3	4
				4	4	4	5	1	2	3	4	5
Miss	x	x	x	x			x	x	x	x	x	x

3.5 Mem. virtual

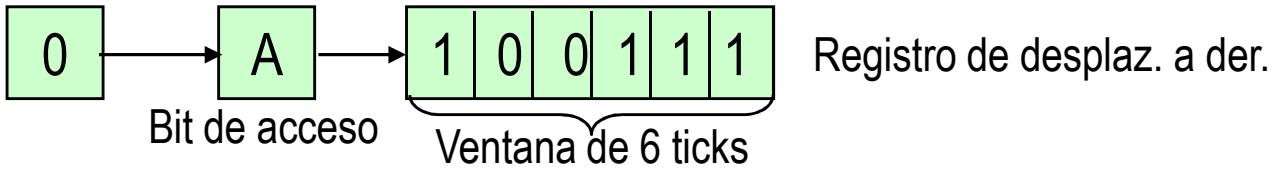
algoritmos de reemplazo

- LRU (Least Recently Used):
  - Aprende del algoritmo Óptimo
  - Pero en vez de mirar al futuro mira al pasado
    - Eliminar la página menos recientemente usada
  - Implementaciones:
    - Apuntar la hora de uso de cada página
    - Cola que se reordena en cada acceso. Ejemplo:

p:	7	0	1	2	0	1	0	4	0	3	0	3
Tramas	7	7	7	0	1	2	2	1	1	4	4	4
		0	0	1	2	0	1	0	4	0	3	0
			1	2	0	1	0	4	0	3	0	3
Miss	x	x	x	x				x		x		

3.5 Mem. virtual
algoritmos de reemplazo

- Aproximación LRU:
  - Más barata de implementar:
    - Añadir un bit A (Acceso) a cada entrada de la Tabla de Pág.
    - El bit A está inicialmente a 0.
    - Cuando la página se referencia se cambia el bit a 1
    - Cada "n" ticks de reloj: copiar A en un reg. de desp.



    - Cada página tiene su reg. de desplazamiento
    - Una pág. con 010011 es menos rec. usada que otra con 100000
- NFU (Not Frecuently Used):
  - Acumula el bit A en un registro contador
    - **Cada n ticks**, si A=1 incrementa reg. contador
    - Víctima: la página con valor más pequeño en reg. contador

3.5 Mem. virtual

algoritmos de reemplazo

- NRU (Not Recently Used):
  - En función de los bits A y D (dirty) se definen:
  - Para selec. víctima:
    - Primero mirar si hay de clase 0
    - Si no, seleccionar una de clase 1
    - Si no, seleccionar una de clase 2
    - Y si no seleccionar una de clase 3

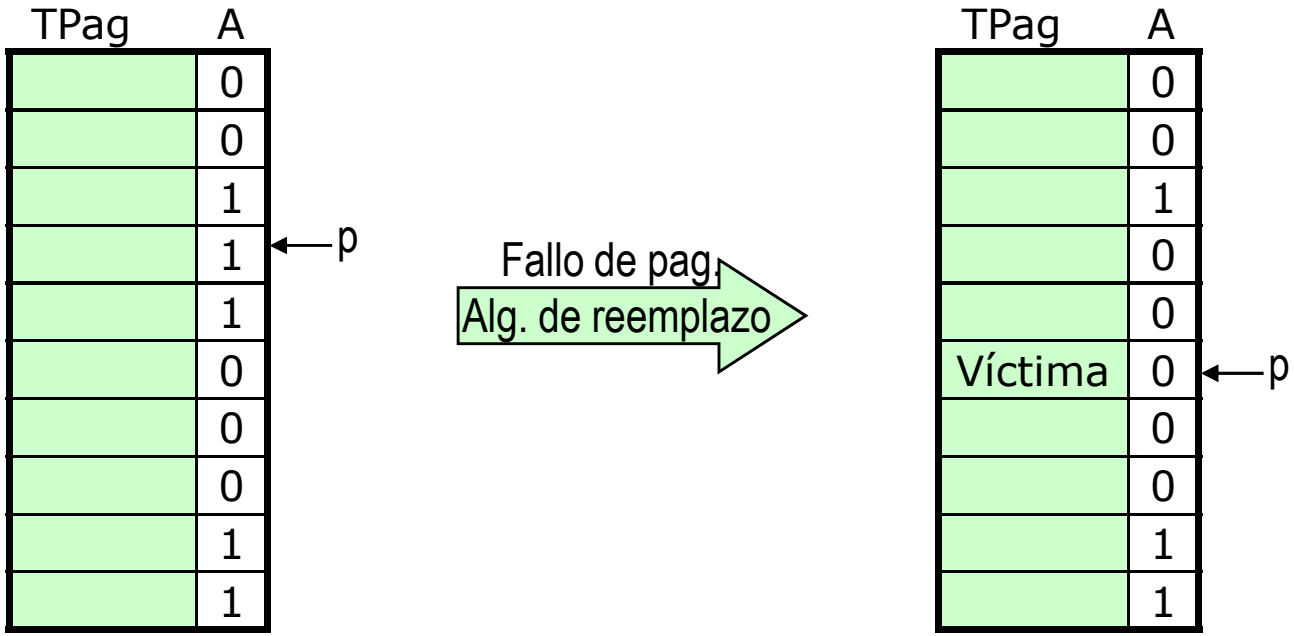
A	D	Clase
0	0	0
0	1	1
1	0	2
1	1	3

- LFU (Least Frecuently Used)
  - Contador para cada trama. **Se incr. en cada acc**.
  - Reemplazar la trama con menor valor
  - Problema:
    - Si una página se ref. mucho gana muchos puntos
    - Cuando deja de usarse sigue con todos los puntos
  - Solución: de vez en cuando div. por dos el cont.

3.5 Mem. virtual

algoritmos de reemplazo

- Clock (Segunda oportunidad)
  - Aplicar FIFO sólo a las pág. con A=0
  - Un puntero circular escanea los bits de la TP
    - Si encuentra A=1 lo pone a 0 y sigue (segunda oportunidad)
    - Si encuentra A=0 → esa pág. es la víctima
    - Caso peor: Todas las pag. con A=1 → dar toda la vuelta
    - Ejemplo:

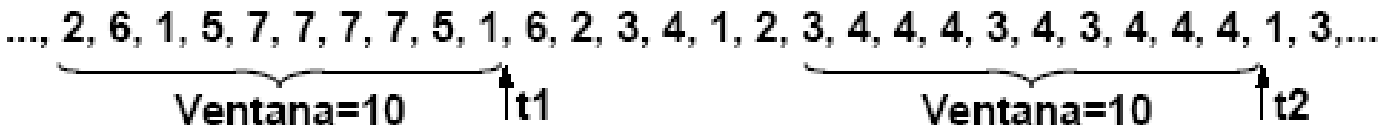


3.5 Mem. virtual

working set

- Working Set (WS)
  - Aproximación a LRU que calcula además el nº de tramas óptimo que necesita un proceso.
  - Ventana: intervalo de tiempo para calcular el WS
  - El WS es el conjunto de tramas necesario para satisfacer las necesidades de localidad del proceso dentro de la ventana de tiempo.
    - Una ventana muy grande preserva en memoria páginas que no son necesarias para la ejecución.
    - Una ventana muy pequeña puede abarcar sólo parte de las páginas necesarias.

Traza con números de las páginas accedidas a lo largo del tiempo

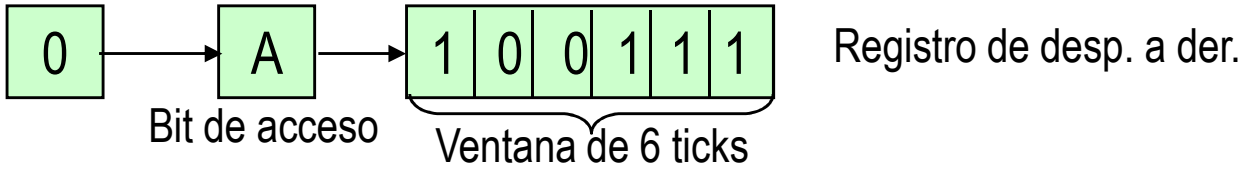


$WS(t_1) = \{1, 2, 5, 6, 7\}$ $WS(t_2) = \{3, 4\}$

3.5 Mem. virtual

working set

- Implementación de Working Set
  - Usa un reg. de desplaz. por cada entrada de TP
  - El nº de bits del reg. → tamaño de la ventana



  - Cada cierto número de ticks (int. del timer):
    - Se desplaza el reg. de cada página del proc. actual a la derecha
    - Se inserta el bit de acceso por la izq. del reg. de cada página.
    - Se pone a cero el bit de acceso de cada página.
  - El registro es una historia del uso de las páginas en cada intervalo temporal:
    - Las páginas no usadas tendrán el registro completo a cero (están fuera del WS).
    - Las páginas usadas tienen valores distintos de cero.
    - Las más recientemente usadas tienen valores mayores.

3.5 Mem. virtual

thrashing

- Thrashing
  - Cuando la suma de las páginas usadas por todos los procesos es mayor que el número de tramas físicas disponibles en el sistema:

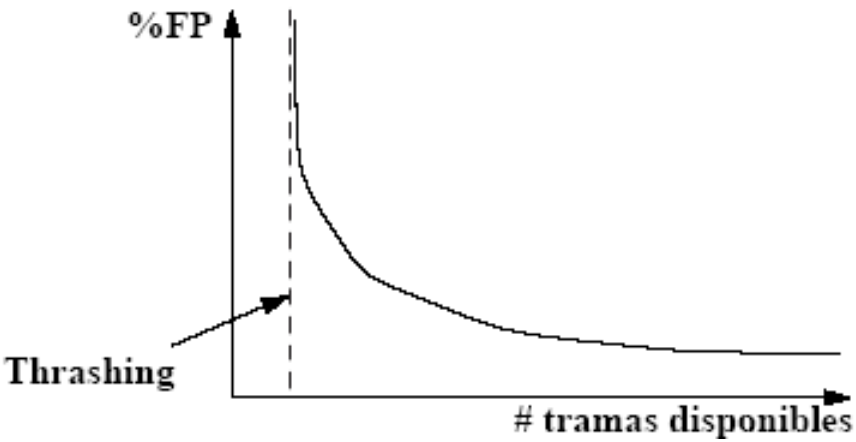
$$\sum_{i=1}^{nº\ proc.} WS_i(t) > MemFísica$$

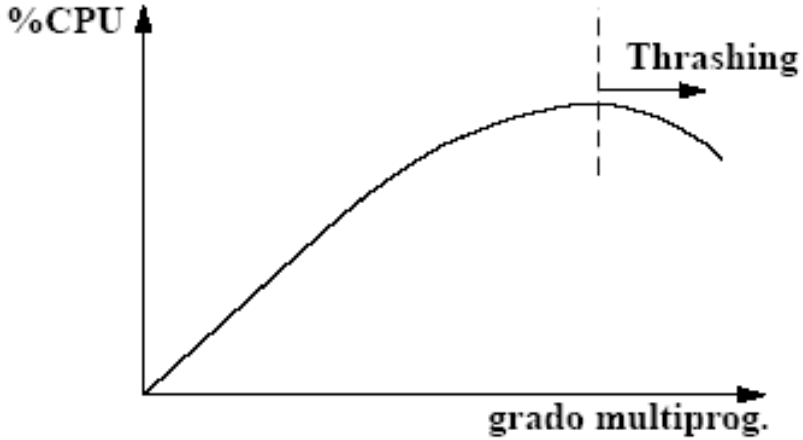
    - Cuando esto ocurre:
      - Hay páginas del WS de varios procesos que residen en el swap.
      - La probabil. de que dichas pág. se accedan en breve es casi 1.
    - Cuando en un proc. A se produce un fallo de página:
      - La página reemplazada también forma parte del WS de un proceso B.
      - Mientras se hace el swap-in de la página de A puede que se haga un cambio de contexto al proceso B, el cual tiene que reclamar de nuevo la trama que le ha quitado el proceso A.

3.5 Mem. virtual

thrashing

- Thrashing
  - *Los procesos del sistema pasan todo el tiempo bloqueados esperando el "swap-in" ya que cada varias instrucciones se produce un fallo de página.*
    - *El uso de la CPU puede llegar al 0% mientras que los programas avanzan tan solo varias instrucciones por segundo.*
    - *El disco de swap mueve continuamente páginas entre memoria y disco y viceversa.*



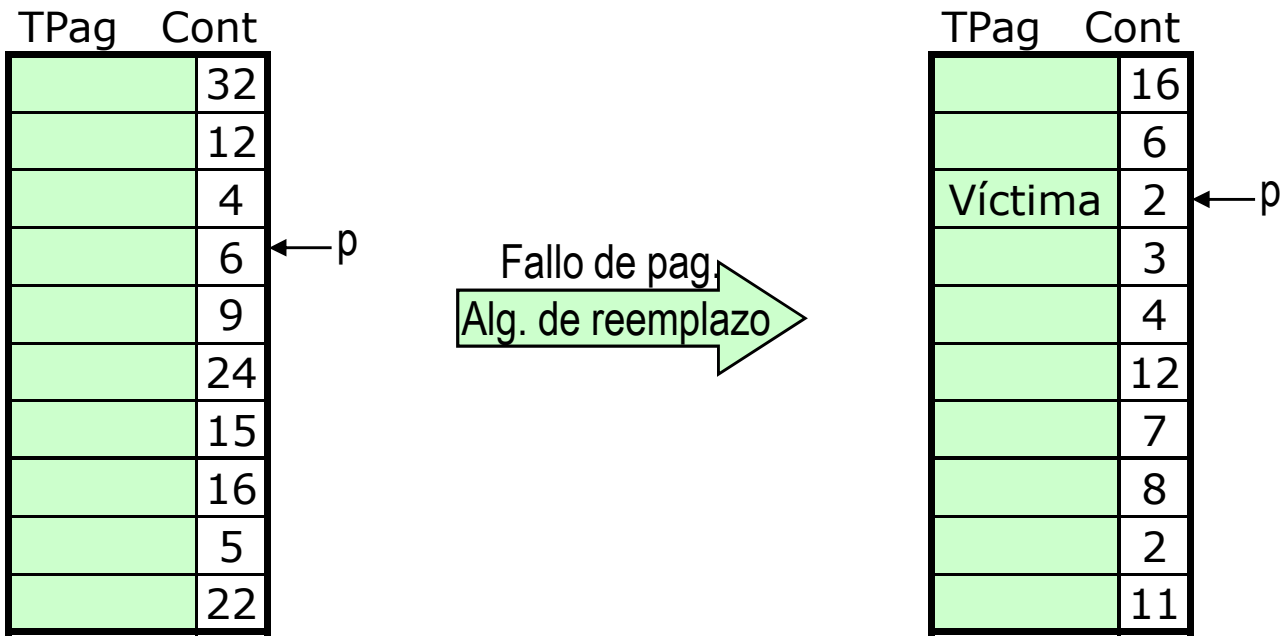


3.5.1 Ejemplos

linux

Memoria Virtual en Linux

- Algoritmo de reemplazo: Clock + LFU
  - En lugar del bit A, cada entrada tiene un contador
  - El contador mide la juventud (frecuencia de acceso) de la pág
  - En cada fallo de página
    - Dividir todos los contadores y encontrar el mínimo
    - Seleccionar la página de menos juventud como víctima



3.5.1 Ejemplos

linux

Memoria Virtual en Linux

- Gestión del swap
  - Soporte de partición específica y de archivo de swap
    - La partición de swap es más rápida (sin overhead de sis. ficheros)
  - Gestión de huecos en el swap → mapa de bits
    - Permanentemente cargado en memoria
    - Cuando hay que escribir varias páginas→buscar 0's consecutivos
      - » Más rendimiento al escribir en sectores consecutivos

- Doble uso de las entradas de la TP
  - En función del bit P (presencia)
    - P=1, almacenar nº de trama
    - P=0, almacena puntero en la zona del swap en que está la página

TPag	P
swap	0
f	1
f	1
swap	0
f	1

3.5.1 Ejemplos

windows xp

Memoria Virtual en Windows XP

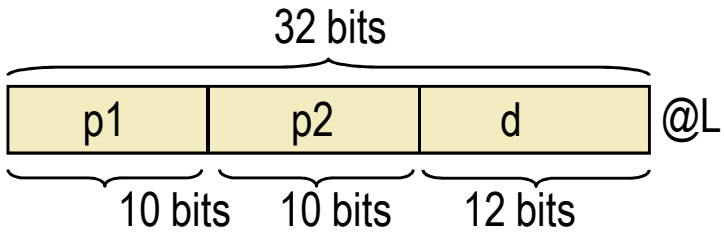
- Algoritmo de reemplazo: tipo working-set
  - A cada proceso se le asignan inicialmente 50 tramas (WS)
  - Periódicamente a un proc. se le quita una trama
    - Esa trama queda en “standby”: está ahí por si el proceso la necesita en el futuro, pero es una clara candidata a víctima
  - Si el proceso no genera fallo se decrementa el WS
  - También → si una pág. se usa poco se pasa a standby
  - Si un proc. supera una tasa de fallos umbral → incremt. WS
- En caso de fallo
  - Cada proceso tiene que reusar su propias tramas del WS
  - Buscar trama libre en la Page Frame Database
    - Da info sobre las tramas libres, tramas dirty y tramas standby
  - Si no hay tramas libres se aplica FIFO entre las tramas de WS

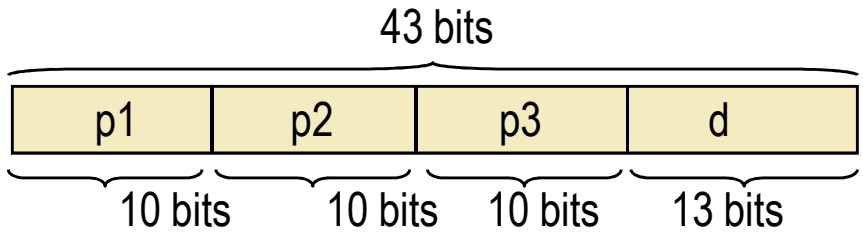
3.5.1 Ejemplos

windows xp

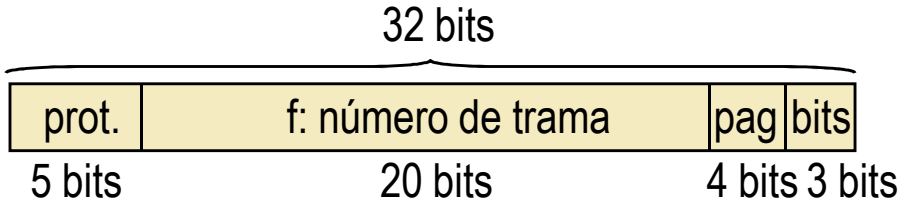
Memoria Virtual en Windows XP

- Paginación en 2 niv. en IA32 y en 3 en IA64





- Entradas de la TP de 2º nivel:



  - Prot: bits de protección (read-only, read-write, etc)
  - Pag: indica el archivo de swap que respalda la página
  - Bits: T (en transición), D (dirty) y P (presencia)
  - Cuando P=0 los 20 bits del campo f apuntan a la pag en el swap
- Precarga: en caso de fallo → trae pag. adicionales

bibliografía

bibliografía

- A. SILBERSCHATZ, P. GALVIN, **Sistemas Operativos. 5 Edición, Pearson, 1999.**
  - **Capítulos 8 (Paginación) y 9 (Mem. virtual)**
- J. CARRETERO, P. DE MIGUEL, F. GARCÍA, F. PÉREZ, **Sistemas Operativos. Una visión aplicada. Mc Graw-Hill, 2001**
  - **Capítulo 4.**
- W. STALLINGS, **Operating Systems: Internals and Design Principles. Cuarta Edición, Prentice-Hall, 2000.**
  - **Capítulos 7 (Paginación) y 8 (Mem. virtual)**