Tema 4

Memoria

3º ETSI

Informática

Contenido

Tema 4. Gestión de memoria. 7h

4.1 La jerarquía de memoria y su organización

4.2 Etapas en la traducción de direcciones

- Espacios de direcciones
- Fases de traducción de direcciones

4.3 Modelos de gestión de memoria

4.4 Paginación y segmentación

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

4.5 Memoria Virtual

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

4.1 Jerarquía de mem.

4.1 Jerarquía de mem.

objetivos del sistema de gestión de memoria

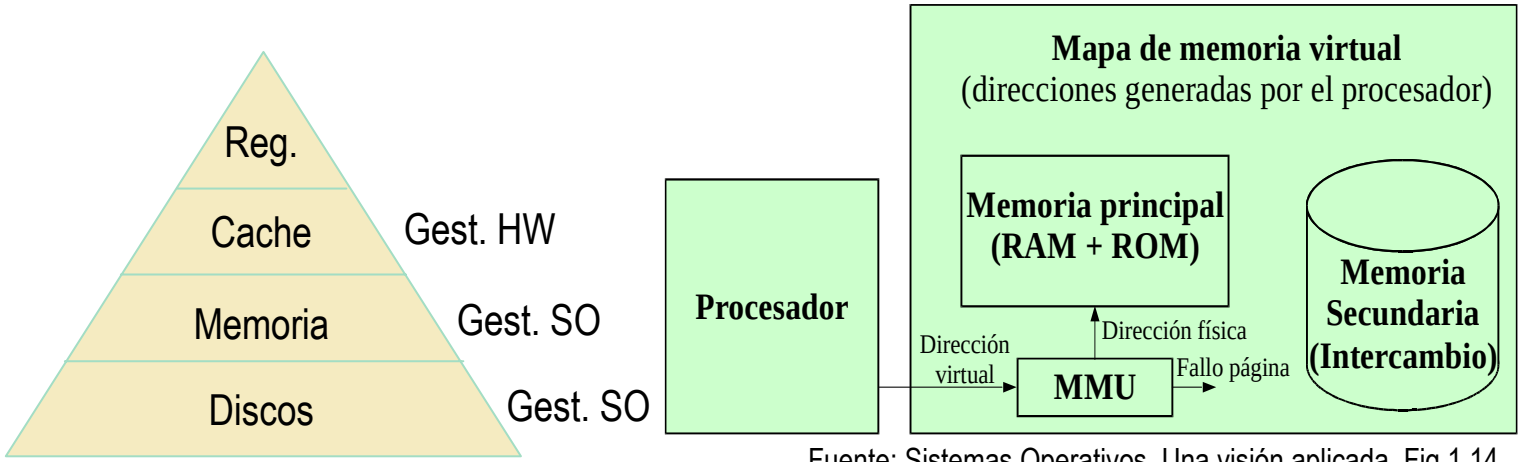
- S.O. multiplexa recursos entre procesos
  - Cada proceso cree que tiene una máquina para él solo
  - Gestión de procesos: Reparto de procesador
  - Gestión de memoria: Reparto de memoria

- 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

4.1 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.

4.2 Traducción de dir.

4.2 Traducción de dir.

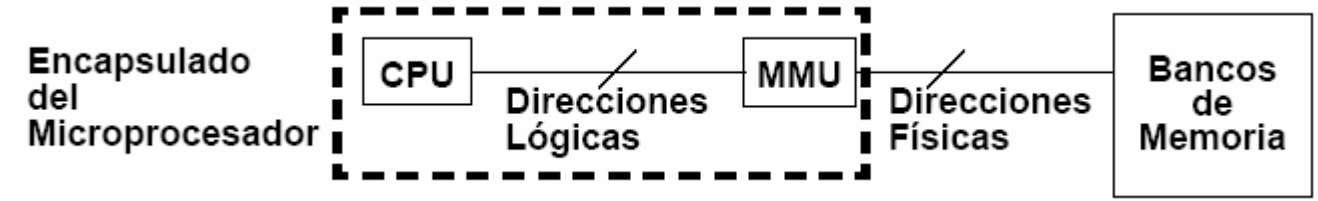
espacio de direccionamiento

- Antes de llegar a la mem. virtual:
  - Entender cómo un programa usa la mem.
  - Cómo vemos la memoria a alto nivel
- Espacio de direccionamiento
  - Rango de valores usados para acceder a mem
- 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 accdr.

4.2 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.



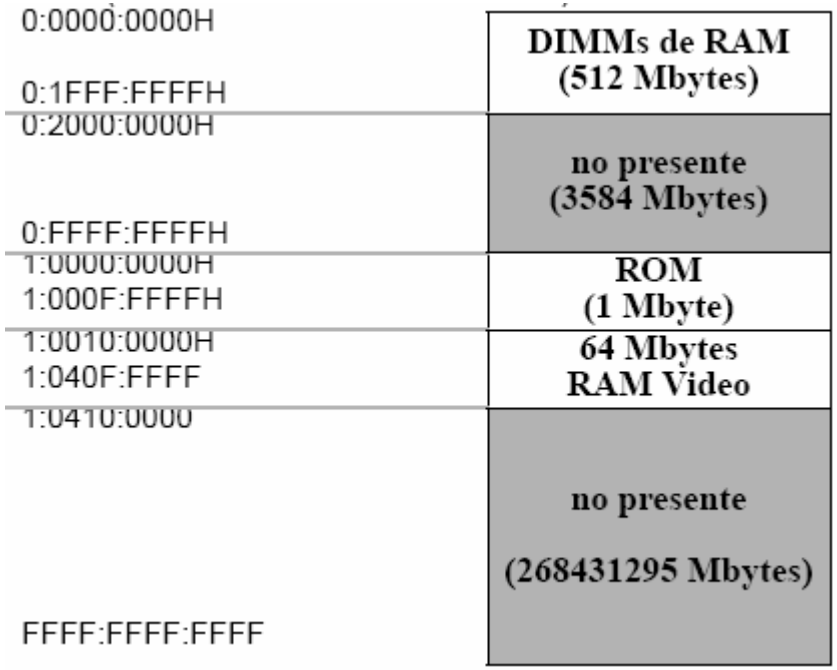
4.2 Traducción de dir.

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] (16ExaBytes = 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

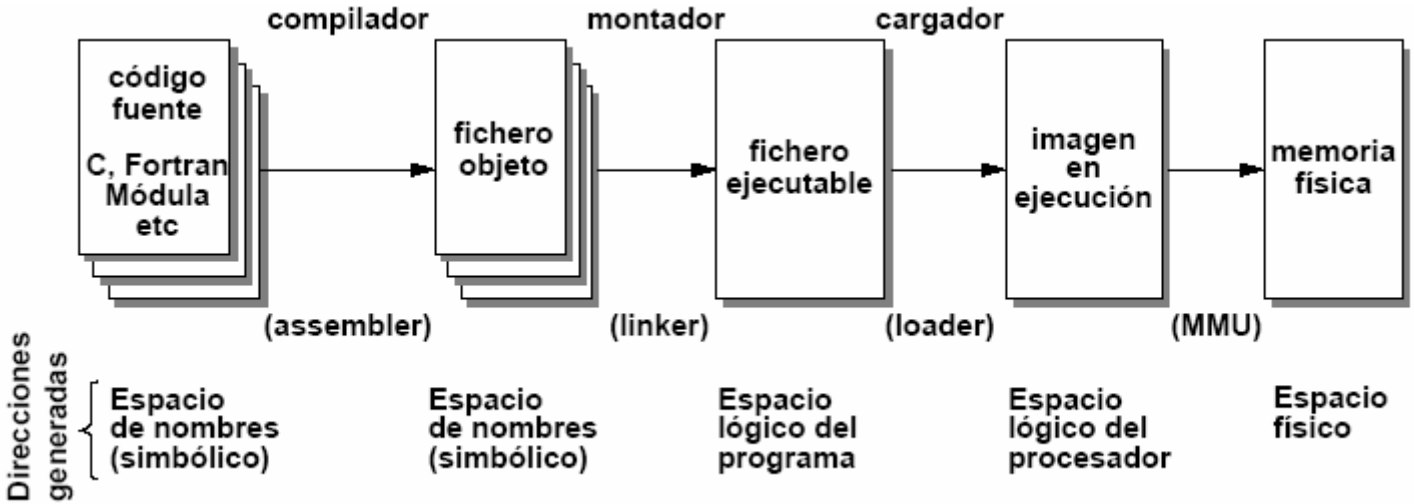
- Espacio físico
  - Posible: [0:2^{48}-1] (256 TB)
  - Válido: [0:2^{29}-1] (512 MB)



4.2 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 dir.
    - Distintas fases de traducción
  - Ventajas: abstracción y portabilidad



4.2 Traducción de dir.

ejemplo de traducción de direcciones

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;
}
```

Módulo Objeto

```
;; Bloque de código: instrucciones de 4 bytes
;; Longitud: 44 bytes
_main  0000  JSR _f ; llamada a f
       0004  MOVE.L R0,_s ; resultado en R0
       0008  RTS ; fin de main

_f     0012  MOVE.L #0,_b ; b=0
       0016  MOVE.L #0,_i ; i=0
_f+8   0020  MOVE.L _i,R0
       0024  ADD.L _tabla(R0),_b ; b+=tabla[i]
       0028  ADD.L #4,_i ; i=i+4
       0032  CMP.L _i,#40 ; ¿i=40?
       0036  JNE _f+8 ; sino ir a _f+8
       0040  MOVE.L _b,R0 ; retorna b en R0
       0044  RTS ; fin de _f

;; Bloque de datos: variables del programa
;; Longitud: 52 bytes
_i     0000  DC.L 0 ; reserva 4 bytes
_b     0004  DC.L 0 ; reserva 4 bytes
_s     0008  DC.L 0 ; reserva 4 bytes
_tabla 0012  DS.L 10 ; reserva 10*4 bytes

;; la pila la añadirá el linker, por lo que
;; no se reserva espacio en el módulo objeto
```

4.2 Traducción de dir.

ejemplo de traducción de direcciones

Fichero ejecutable

PIC:
Código independiente de la posición

```
;; Bloque de código
codigo::
_main   0000 JSR codigo+12
        0004 MOVE.L R0,datos+8
        0008 RTS
_f      0012 MOVE.L #0,datos+4
        0016 MOVE.L #0,datos+0
_f+8    0020 MOVE.L datos+0,R0
        0024 ADD.L datos+12(R0),datos+4
        0028 ADD.L #4,datos+0
        0032 CMP.L datos+0,#40
        0036 JNE codigo+20
        0040 MOVE.L datos+4,R0
        0044 RTS
;; Bloque de datos
datos::
_i      0000 DC.L 0 ; reserva 4 bytes
_b      0004 DC.L 0 ; reserva 4 bytes
_s      0008 DC.L 0 ; reserva 4 bytes
_tabla  0012 DS.L 10 ; reserva 10*4 bytes
;; Bloque de pila
pila::
pila 0000 DS.L 256 ; reserva 256*4 bytes
```

4.2 Traducción de dir.

ejemplo de traducción de direcciones

Imagen en memoria

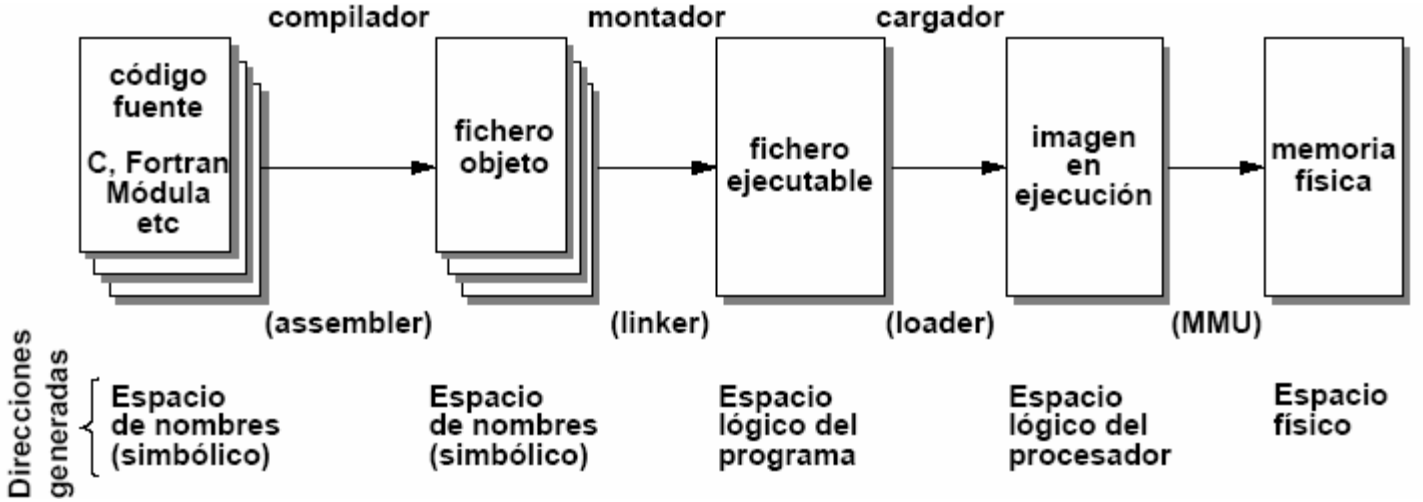
```
;; Dirección lógica inicial del código: 1024
1024 JSR 1036
1028 MOVE.L R0,4104
1032 RTS
1036 MOVE.L #0,4100
1040 MOVE.L #0,4096
1044 MOVE.L 4096,R0
1048 ADD.L 4108(R0),4100
1052 ADD.L #4,4096
1056 CMP.L 4096,#40
1060 JNE 1044
1064 MOVE.L 4100,R0
1068 RTS
;; Dirección lógica inicial de los datos: 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
```

Aún son direcciones lógicas que serán traducidas en la MMU por direcciones físicas

4.2 Traducción de dir.

etapas en la traducción de direcciones

- Etapas de traducción de direcciones
  - Compilación
    - Resuelve referencias dentro de cada módulo fuente y genera el módulo objeto
    - Separa el código de los datos
  - Montaje
    - Resuelve referencias cruzadas entre módulos objeto
    - Resuelve referencias a símbolos de bibliotecas
    - Genera ejecutable, incluyendo bibliotecas
  - Carga
    - Asigna direcciones iniciales a los segmentos del programa
  - Ejecución
    - Traduce direcciones lógicas a físicas.



4.3 Modelos de mem.

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

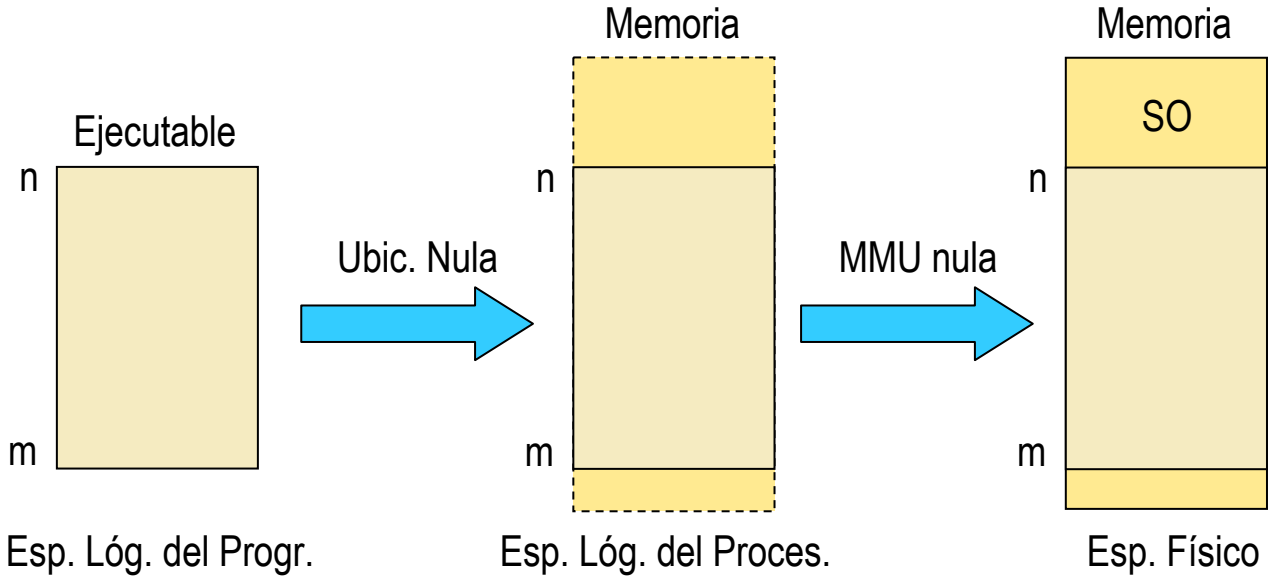
- Modelos básicos de traducción de dir.
- Combinan las siguientes características
  - Uniprogramado/Multiprogramado
    - Uno o varios procesos en memoria
  - Residente/No residente
    - El proceso permanece o no en memoria al ejecutarse: intercambio (swapping)
  - Inmóvil/Móvil
    - Ocupa las mismas posiciones o se puede mover: reubicación para compactación
  - Contiguo/No contiguo
    - Posiciones de memoria consecutivas o distribuidas: aprovechar memoria libre fragmentada
  - Entero/No entero
    - Posibilidad o no de que no esté cargado todo el proceso en mem.

4.3 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 dir. físicas definit.
  - No hay ninguna traducción después de enlazar
  - El enlazador tiene que saber cuanto ocupa el SO en mem.

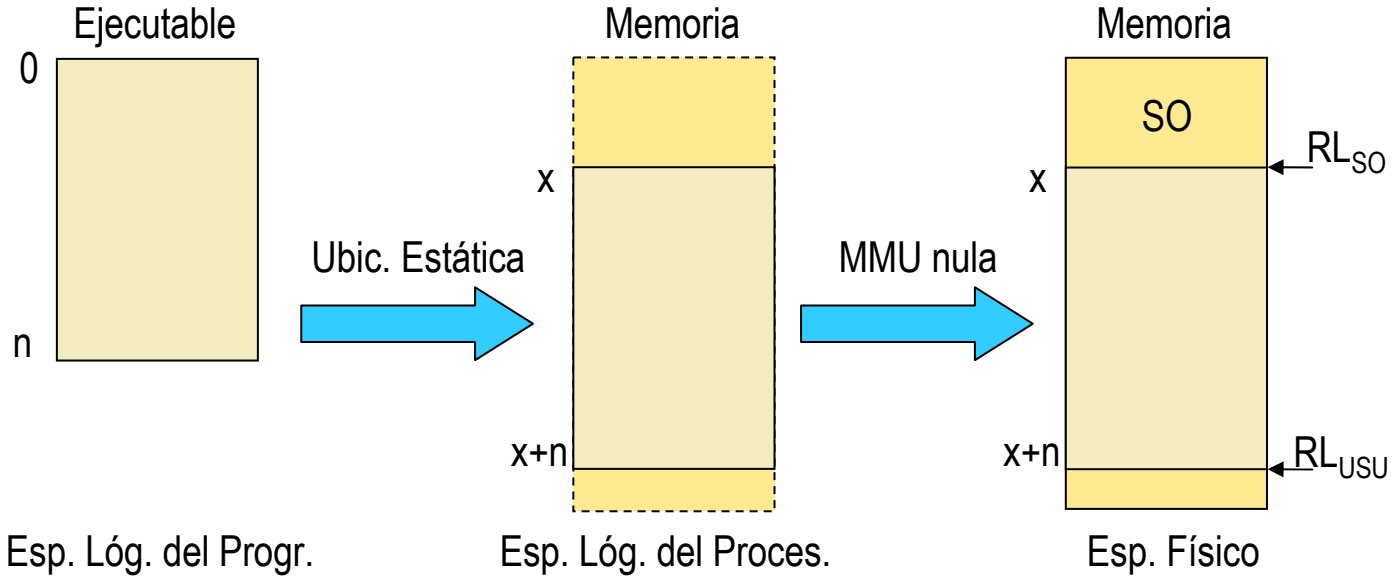


4.3 Modelos de mem.

uniprogr.|residente|inmóvil|contiguo|entero

Modelo primitivo

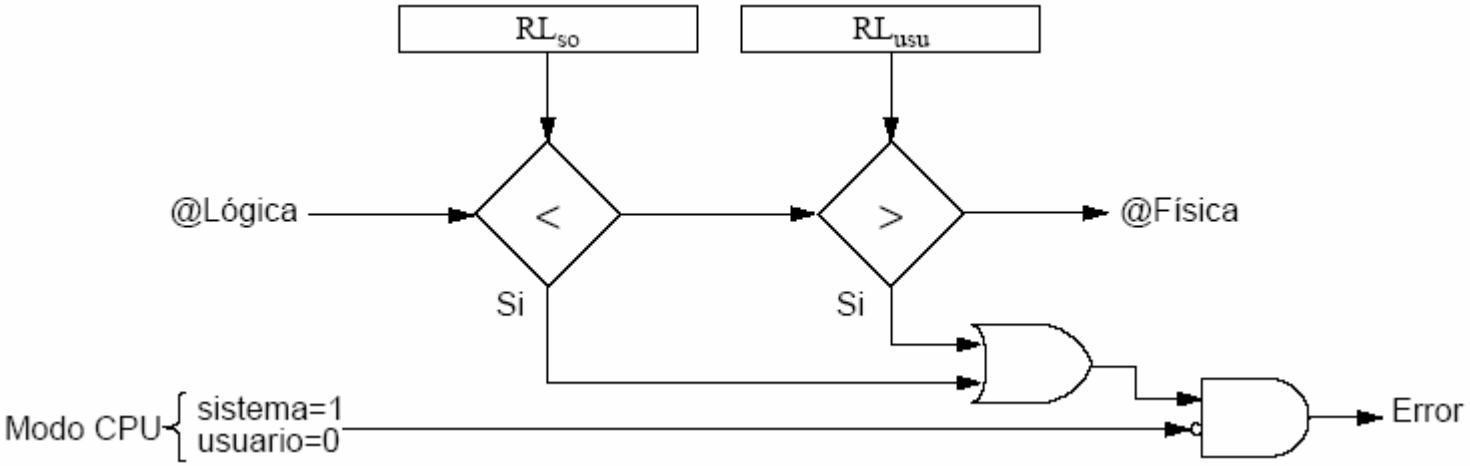
- Con reubicación
  - Esp. Lóg. Prog ≠ Esp. Lóg. Proc. = Esp. Físico
  - Cuando no se conoce cuanto ocupa el SO en mem.
  - El enlazador genera el ejecutable entre 0 y n
  - El cargador ubica el código



4.3 Modelos de mem.

uniprogr.|residente|inmóvil|contiguo|entero

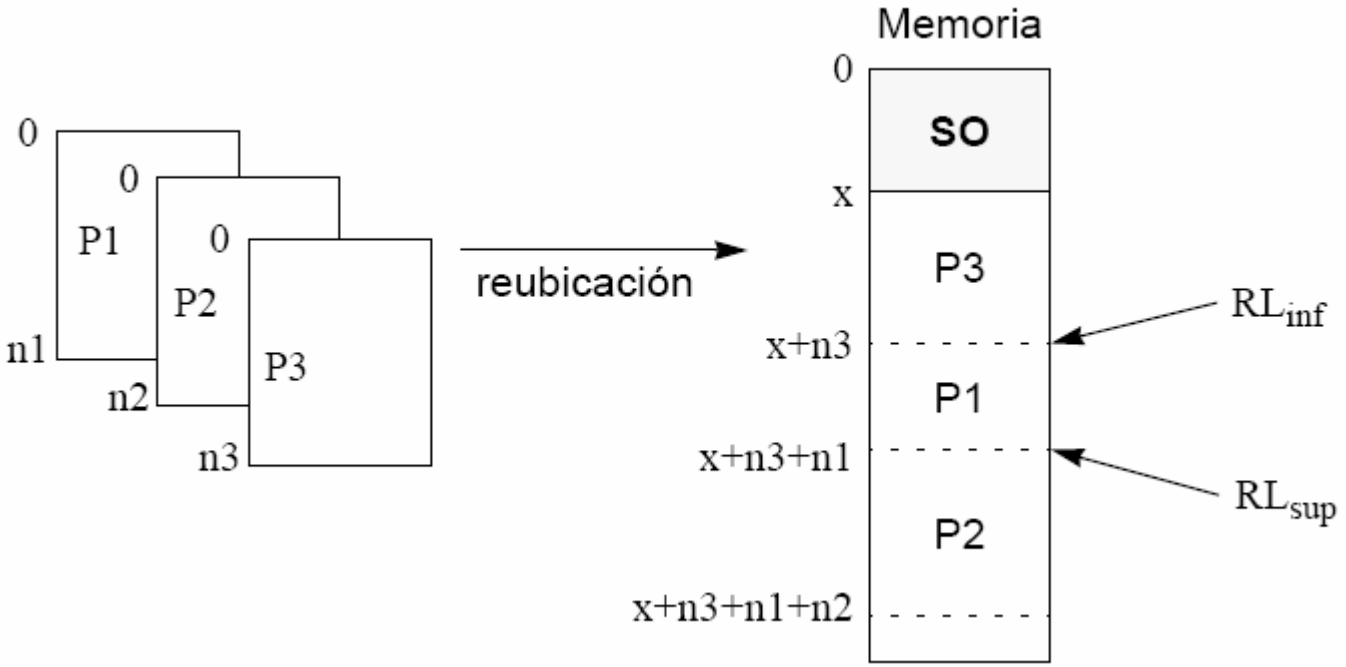
- Modelo primitivo: Función de la MMU
  - La MMU no traduce direcciones
  - Pero vigila los accesos a regiones
  - En modo usuario:
    - La @Lógica tiene que estar en el rango [RL_{SO}, RL_{USU}]
    - En caso contrario se lanza una excepción
  - En modo supervisor: no hay restricciones



4.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

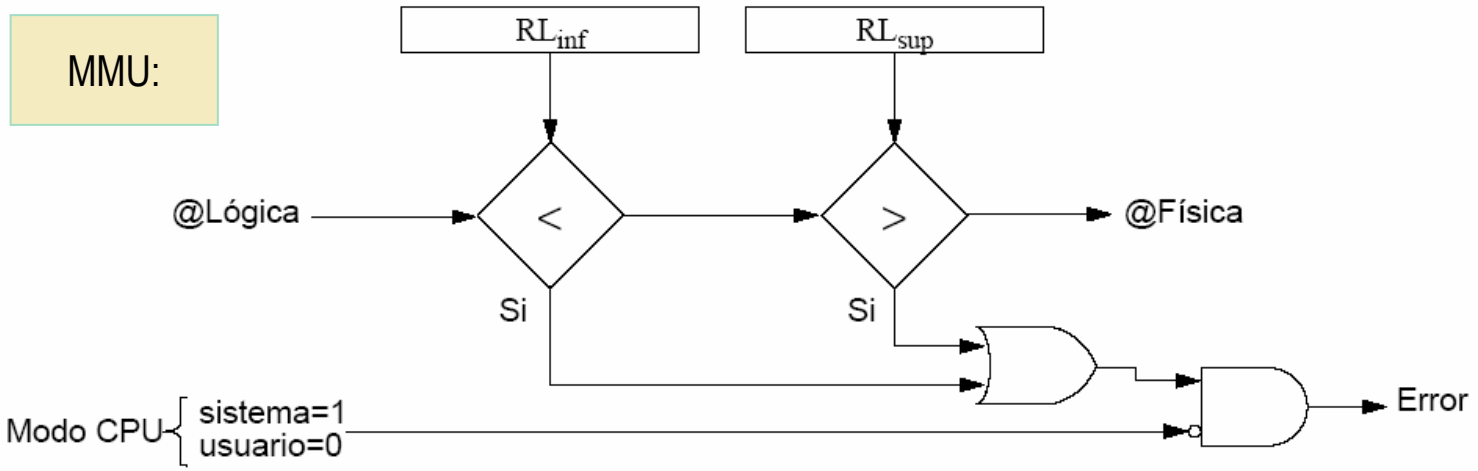
- Modelo Multiprogramado
  - La multiprogramación mejora el uso de CPU
  - Dos problemas:
    - Proteger los procesos entre si
    - Ubicar los procesos en memoria



4.3 Modelos de mem.

Modelo Multiprogramado: Protección

- 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



4.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

- Modelo Multiprogramado: Asignación
  - El SO debe controlar las zonas libres y ocup.
- 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

4.3 Modelos de mem.

Problemas de la gestión de memoria

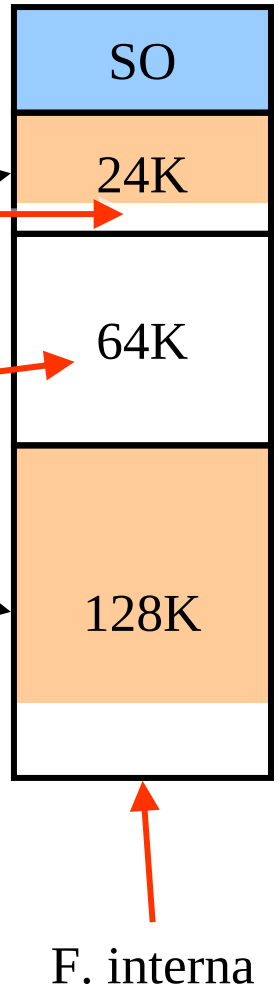
- Fragmentación interna
  - Memoria asignada a un proceso pero no usada
  - Aparece cuando
    - Asignas a un proceso un bloque mayor del necesario
    - Típico en particiones de tamaño fijo
  - Solución: reducir tamaño de las particiones
- Fragmentación externa
  - Bloques de mem. libre no usados
 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

4.3 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)

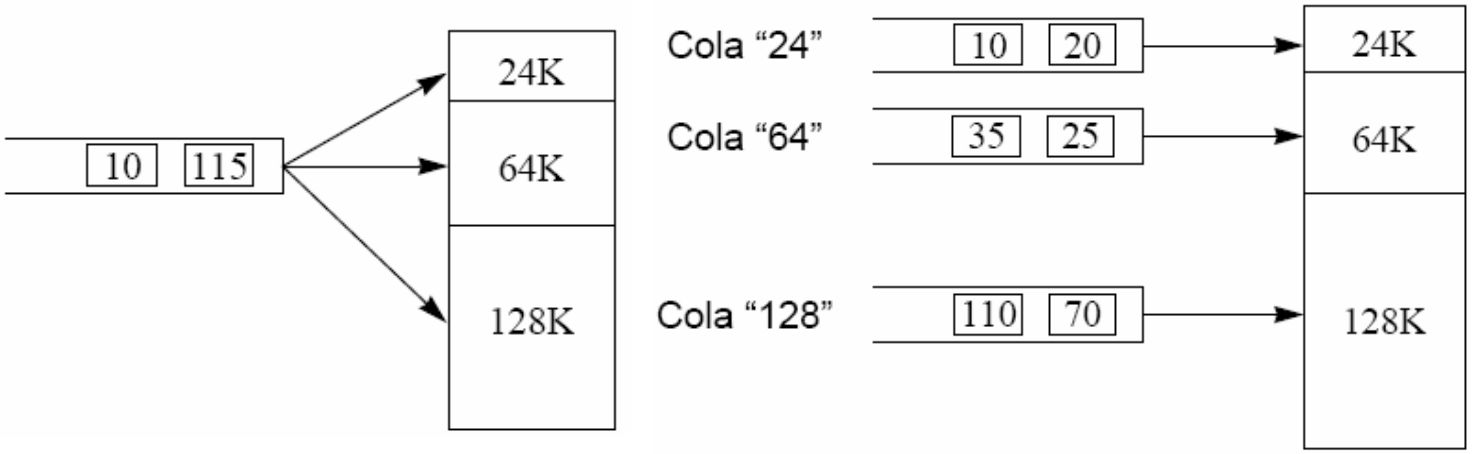


- Particiones: 24Kb, 64Kb, 128Kb
- Memoria libre (real) = 9+64+28=101 Kb
- Memoria libre (disponible) = 64Kb

4.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

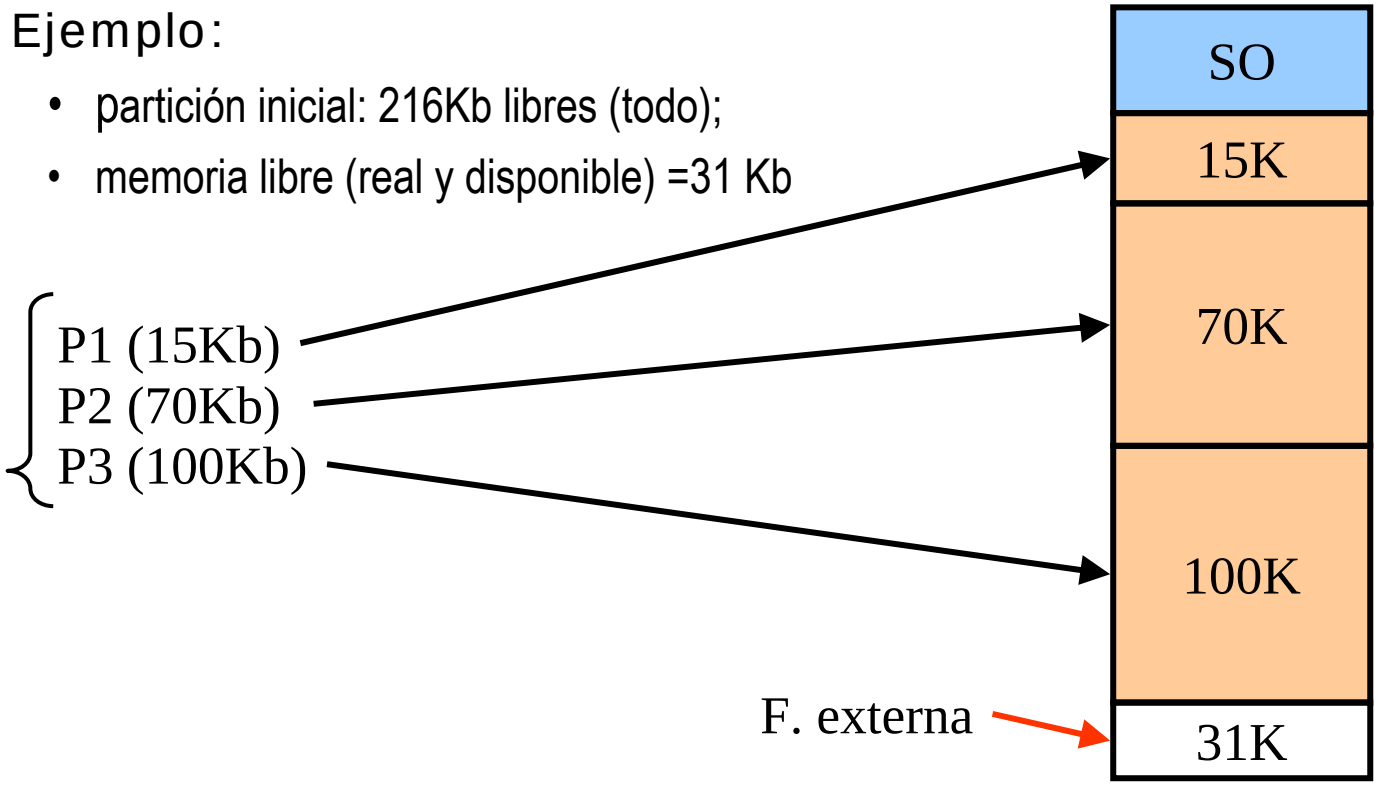
- Particiones fijas: Políticas de asignación
- Colas múltiples: una cola para cada partición
  - Cada proceso asignado a un tamaño de bloque (el que mejor ajuste)
  - Cada proceso espera en colas exclusivas para cada tamaño de bloque
  - Aunque otra partición esté libre tiene que esperar en su cola a la partición correspondiente
- Cola única:
  - Asignación de memoria por turno de llegada
  - Primer ajuste: asignar el primer hueco con tamaño suficiente
  - Mejor ajuste disponible: asignar el menor hueco con tamaño suficiente que esté libre (la tabla de particiones se ordena por tamaños)



4.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

- Particiones variables: no existen particiones predefinidas
  - S.O. asigna a cada proceso justo lo que necesita: **Aumenta nivel de multiprogramación**
  - No existe fragmentación interna
  - Se complica la gestión de trozos libres/ocupados
  - Ejemplo:
    - partición inicial: 216Kb libres (todo);
    - memoria libre (real y disponible) =31 Kb



4.3 Modelos de mem.

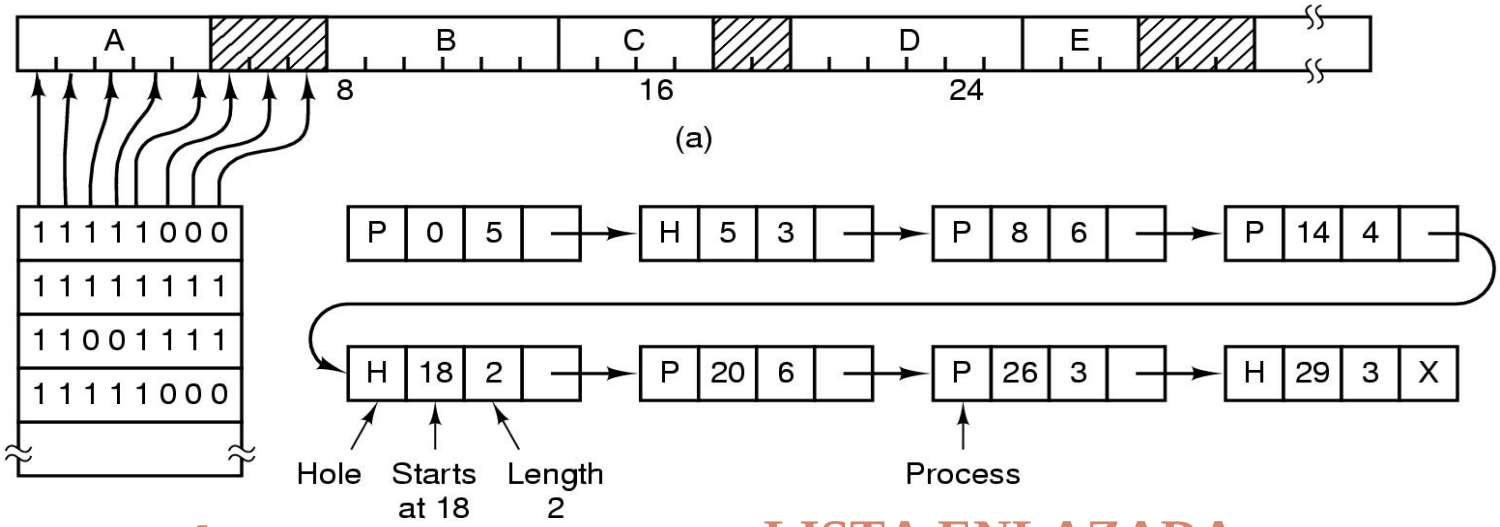
multipr.|residente|inmóvil|contiguo|entero

- 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

4.3 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

4.3 Modelos de mem.

multipr.|residente|inmóvil|contiguo|entero

- Particiones variables. Políticas de asignación:
  - Best-Fit: Asignar el menor hueco con tamaño suficiente para el proceso
    - –Lista ordenada por tamaño o buscar en toda la lista
    - –Genera un bloque libre más pequeño posible → frag. ext.
  - Worst-Fit: Asigna el bloque más grande de todos
    - –Lista ordenada por tamaño o buscar en toda la lista
    - –Genera el bloque libre más grande posible → inanición pr. grandes
  - First-Fit: Asignar el primer hueco con tamaño suficiente. Comportamt. Intermedio
- Primer ajuste es más eficiente y proporciona un buen aprovechamiento de la memoria
- Este esquema presenta fragmentación externa:
  - Se generan pequeños fragmentos libres entre zonas asignadas

4.3 Modelos de mem.

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

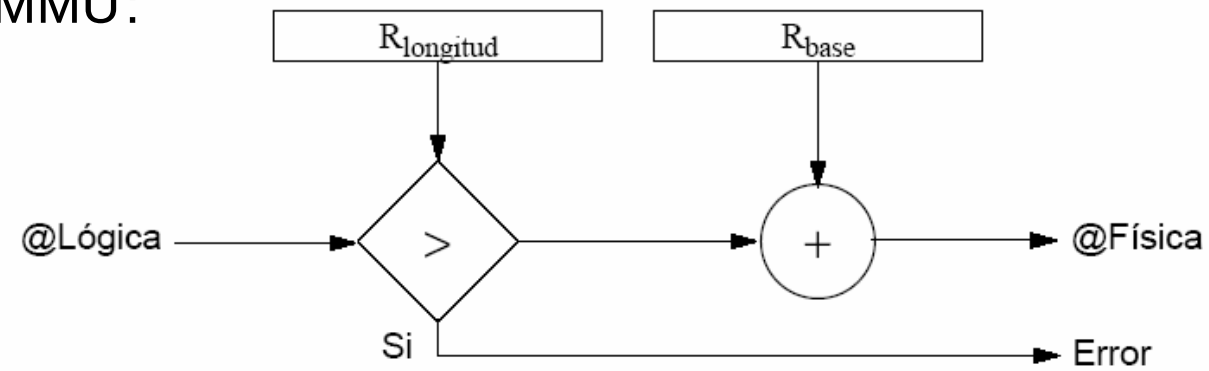
- Modelo no residente
  - Modelo residente: limitado por la mem. física
  - Alternativa: descargar procesos al HD
    - Operaciones swap-out (Mem→HD) y swap-in (HD→Mem)
- Ventajas:
  - Aumenta el grado de multiprogramación
  - Memoria dinámica (malloc)
    - Un proceso puede crecer descargando a algún 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

4.3 Modelos de mem.

multipr.|no resid.|**móvil**|contiguo|entero

Modelo móvil

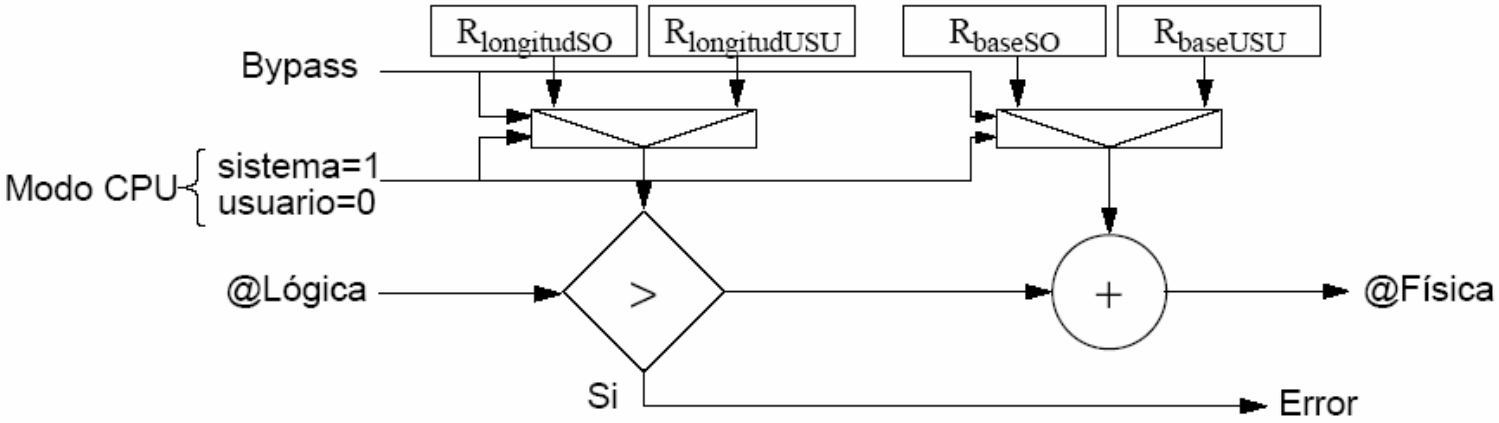
- Podemos cambiar un proceso de sitio (reubicar)
  - Esp. Lóg. Prog = Esp. Lóg. Proc. ≠ Esp. Físico
- Ventajas
  - Se puede hacer "garbage collection" (desfragmentar)
  - Mejor aprovechamiento de la memoria
  - Aumenta el grado de multiprogramación máximo
  - Memoria dinámica más fácil de implementar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



4.3 Modelos de mem.

multipr.|no resid.|**móvil**|contiguo|entero

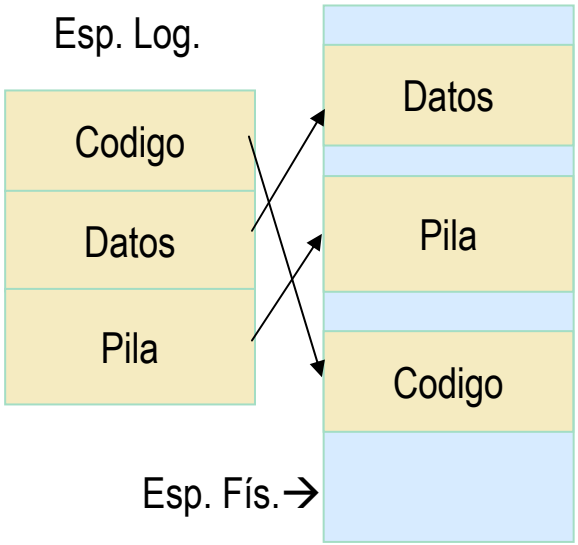
- Modelo móvil: detalles de la MMU
  - Dir. base y longitud almacenadas en el PCB
  - En los cambios de contexto → actualizar regs MMU
  - El SO tiene un espacio distinto (regs propios)
    - En un syscall se pasa a modo sist. → acceso a mem del SO
  - El SO debe poder acceder al espacio del proceso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



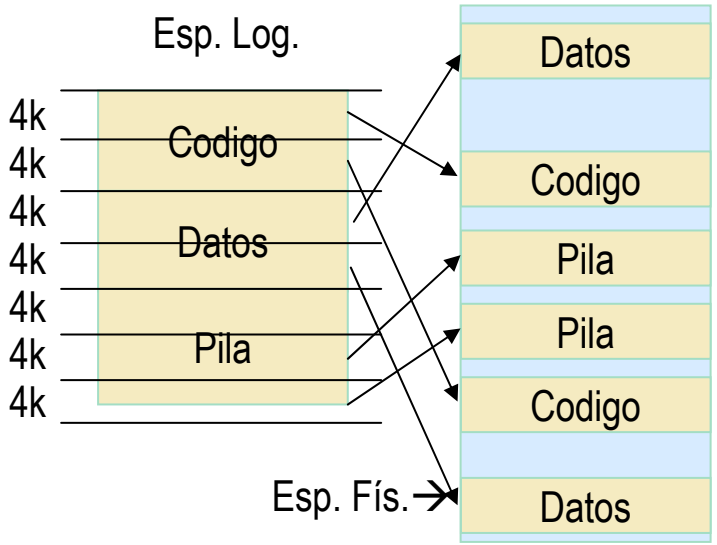
4.3 Modelos de mem.

multip.|no resid.|móvil|**no contiguo**|entero

- Modelo no contiguo: apartado 4.4
  - 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

4.3 Modelos de mem.

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

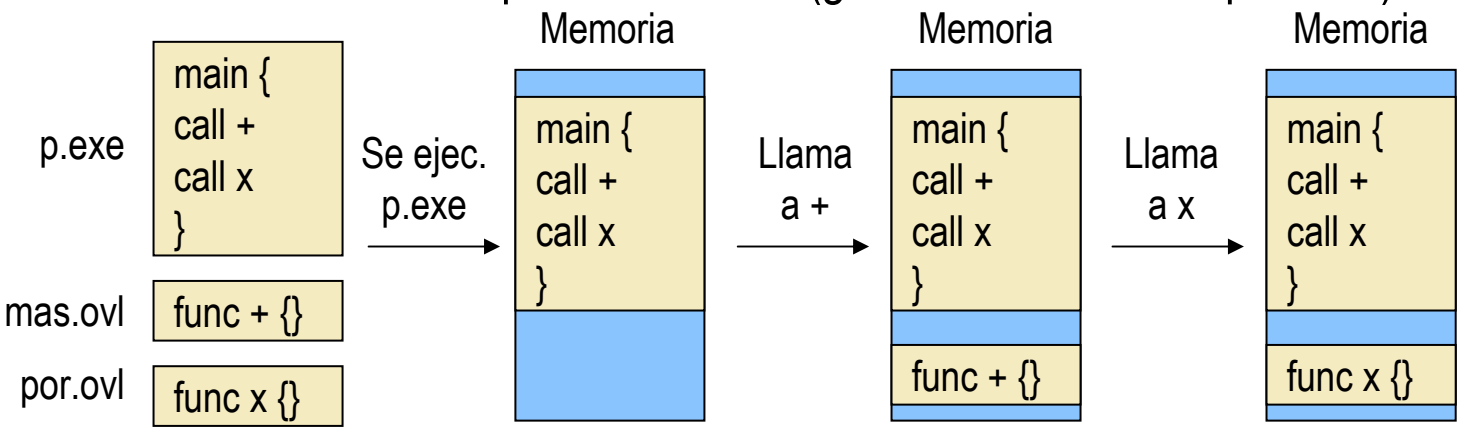
- Modelo no entero
  - La imagen del proceso no tiene por que estar cargada en memoria completamente
  - Basta con mantener las zonas en uso
  - Se basa en los principios de localidad
  - Ventajas
    - Se pueden ejecutar procesos más grandes que la mem
    - Caben más procesos en memoria (más grado de multipr.)
    - Swapping de trozos (no de procesos completos)
  - Posibilidades
    - Overlays
    - Librerías con enlace dinámico (DLL)
    - Memoria Virtual (apartado 4.5)

4.3 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)



4.3 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

4.4 Paginación y seg.

4.4 Paginación y seg.

motivación

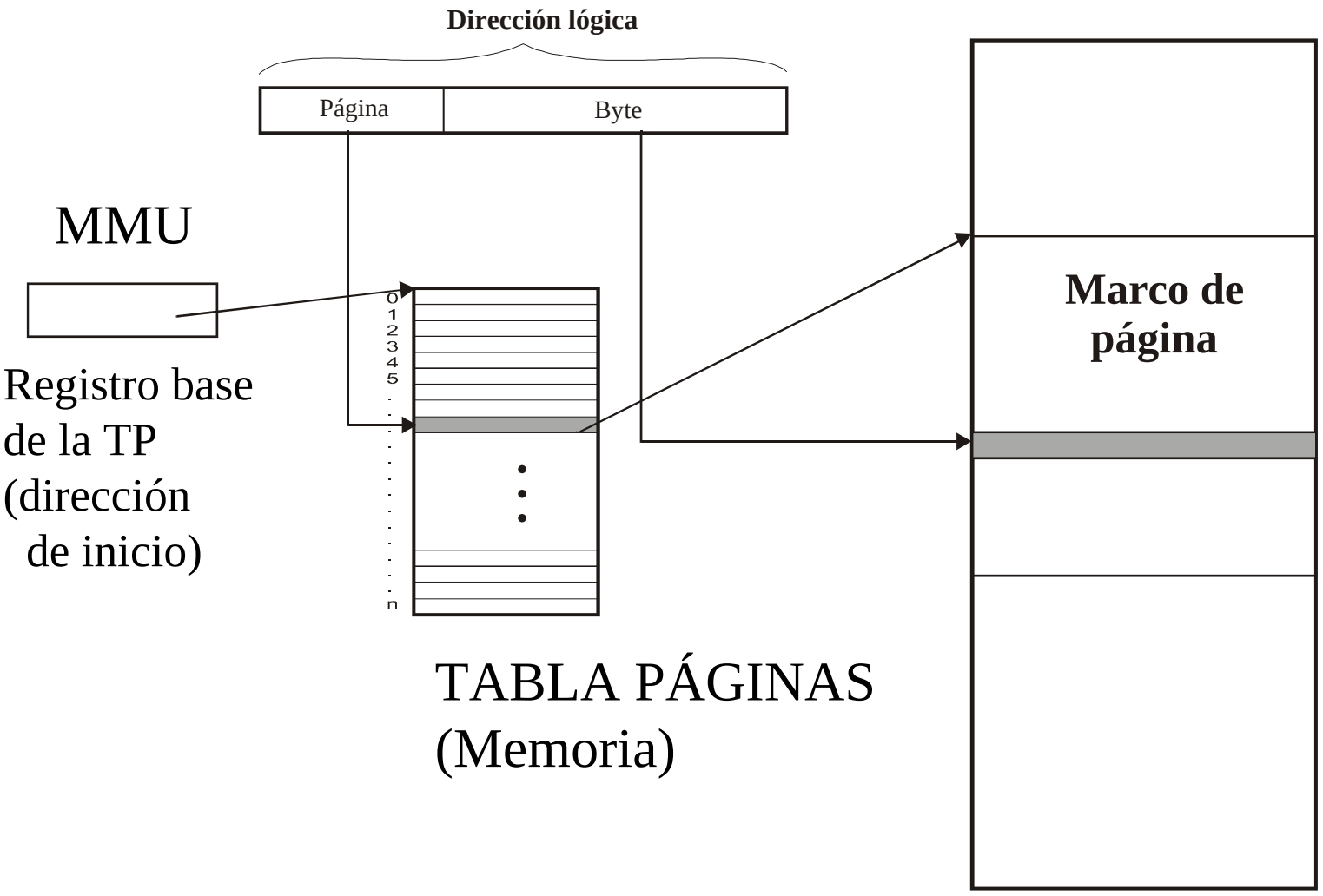
- Motivación del modelo **no contiguo**
  - Inconvenientes del modelo contiguo:
    - Encontrar una partición donde quepa el proceso contiguo en mem
    - Si no hay un hueco lo suficientemente grande:
      - Swap-out de otros proc. en el modelo no residente: Poco práctico
      - Desfragmentar la memoria 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

4.4.1 Paginación

Comentarios sobre la implementación

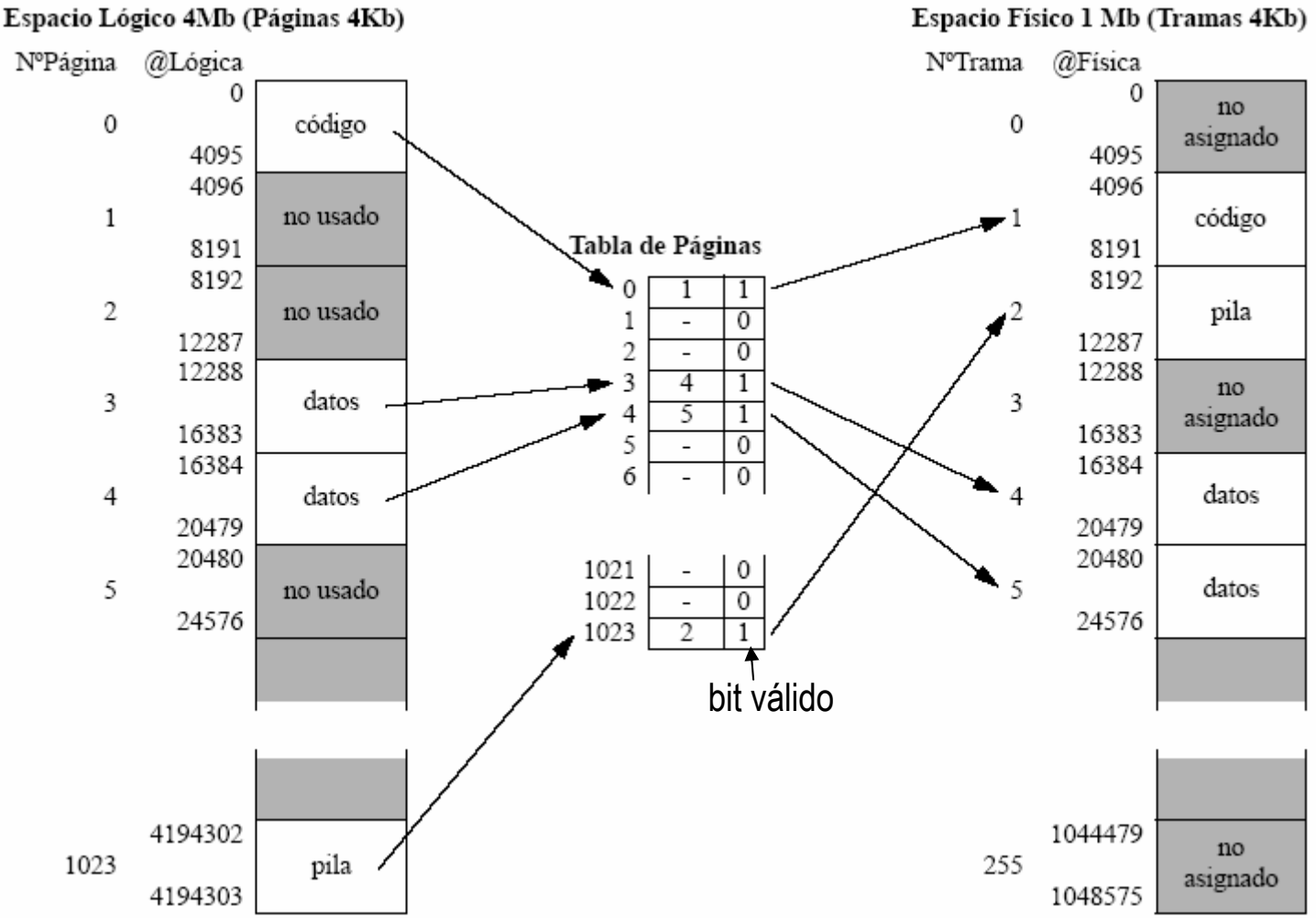
- Máxima flexibilidad particionando E.L en bytes
  - No habría fragment. interna en las páginas parcialmente usadas
  - Pero eso es inviable: Tabla de páginas prohibitiva
  - Solución:
    - Unidad asignación no contigua: **página** (tamaño potencia de 2)
    - Espacio Lógico del proceso dividido en páginas
    - Espacio Físico dividido 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.
- 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

4.4.1 Paginación
esquema de traducción



4.4.1 Paginación
implementación

Implementación de la paginación



4.4.1 Paginación

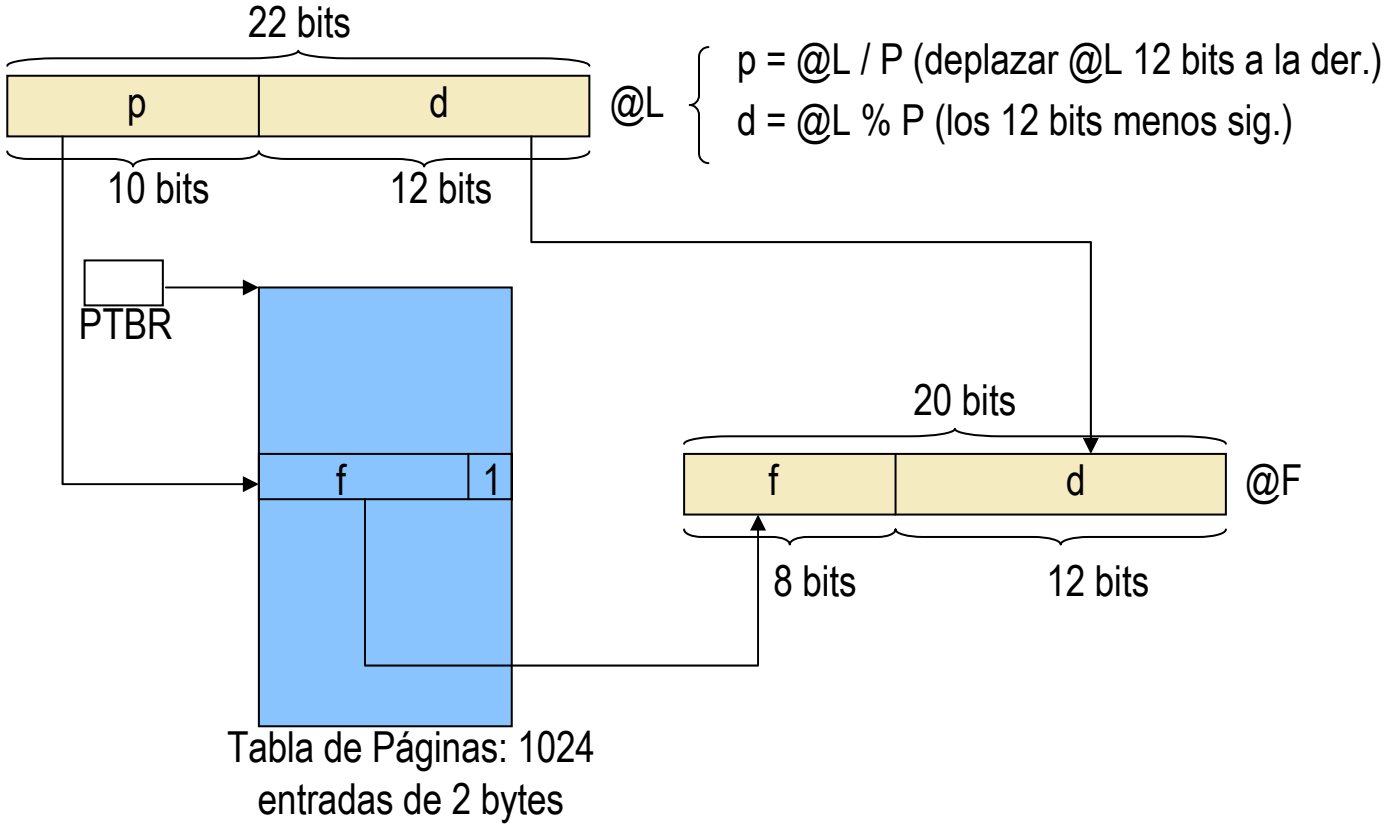
notación

- 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

4.4.1 Paginación

ejemplo

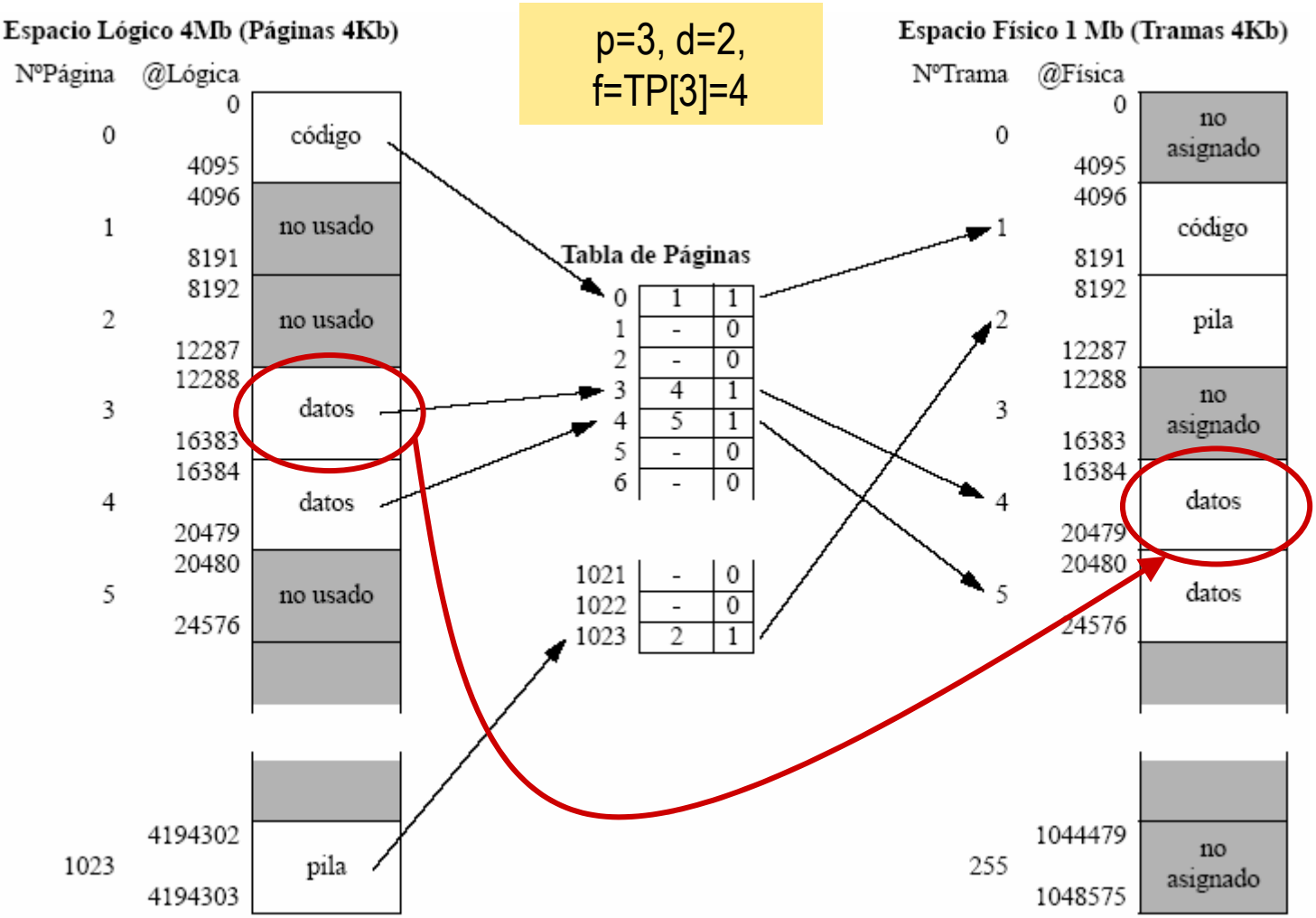
- Ejemplo de traducción:
  - E.L. de 4Mb. → @L de 22 bits (@L[21..0])
  - E.F. de 1Mb. → @F de 20 bits (@F[19..0])
  - P = 4Kb = 4096bytes = 2^{12}



4.4.1 Paginación

ejemplo

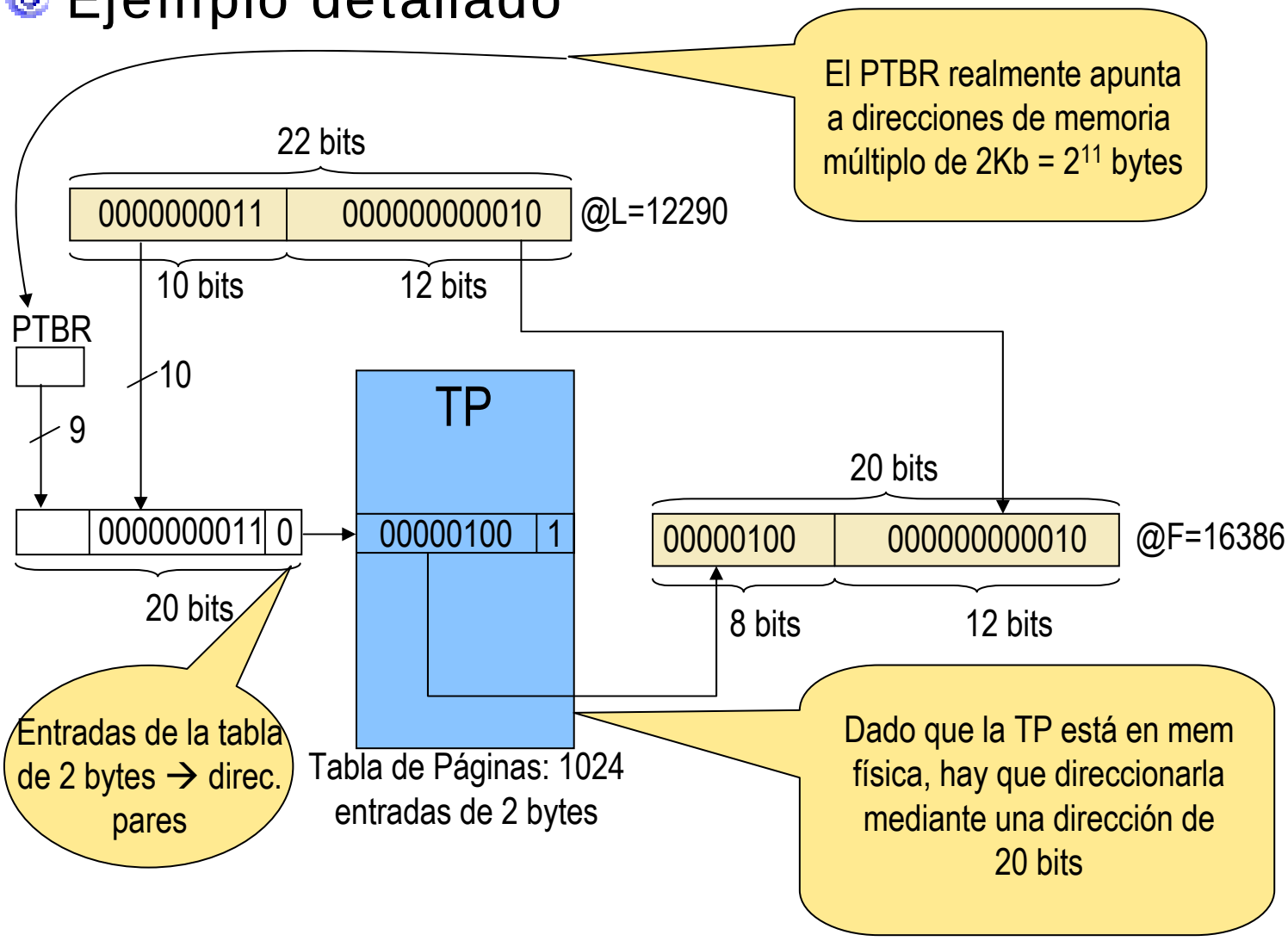
- Ejemplo: @L = 12290 → @F = 16386



4.4.1 Paginación

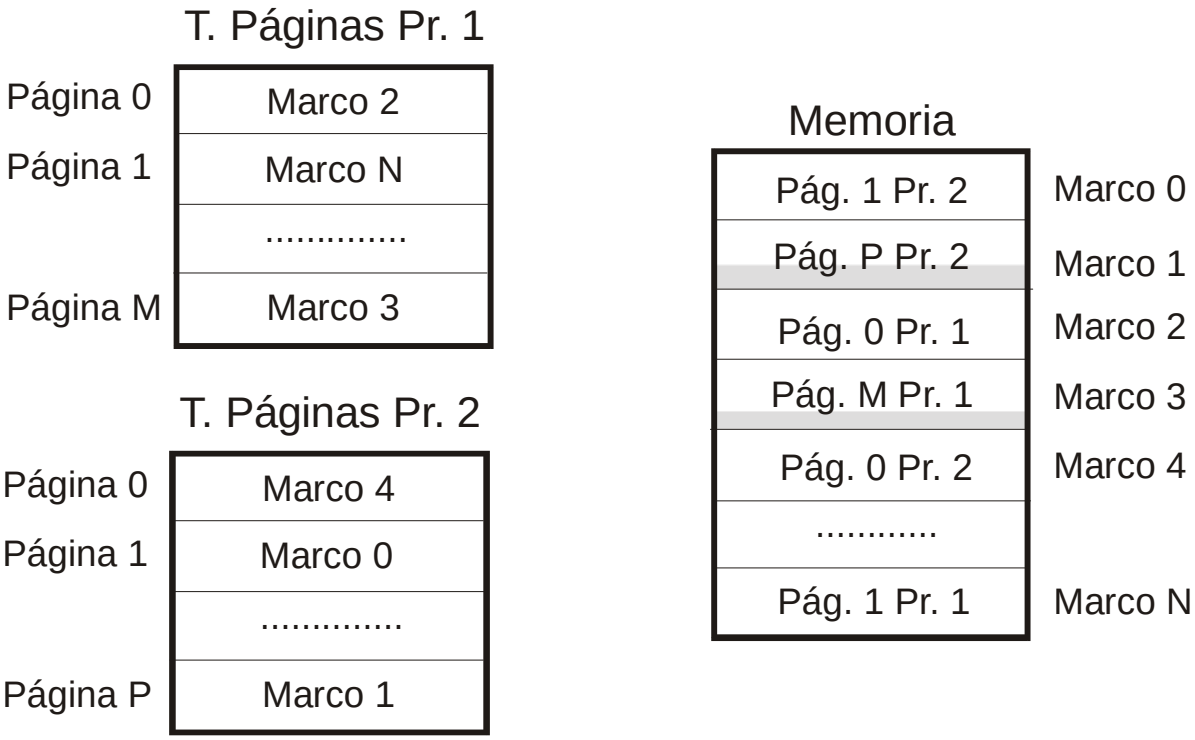
ejemplo

Ejemplo detallado



4.4.1 Paginación

fragmentación interna en paginación



- mem. asignada > mem. requerida
  - puede desperdiciarse parte de un marco asignado

4.4.1 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): a 1 si se escribe en la página
    - Bit de presencia (P): usado en memoria virtual (ap. 4.5)

4.4.1 Paginación
tamaño de la página

- Condicionado por diversos factores contrapuestos:
  - Potencia de 2 y múltiplo del tamaño del bloque de disco
  - Mejor pequeño por:
    - Menor fragmentación
    - Se ajusta mejor al conjunto de trabajo
  - Mejor grande por:
    - Tablas más pequeñas
    - Mejor rendimiento del dispositivo de E/S
  - Compromiso (entre 2K y 16K)

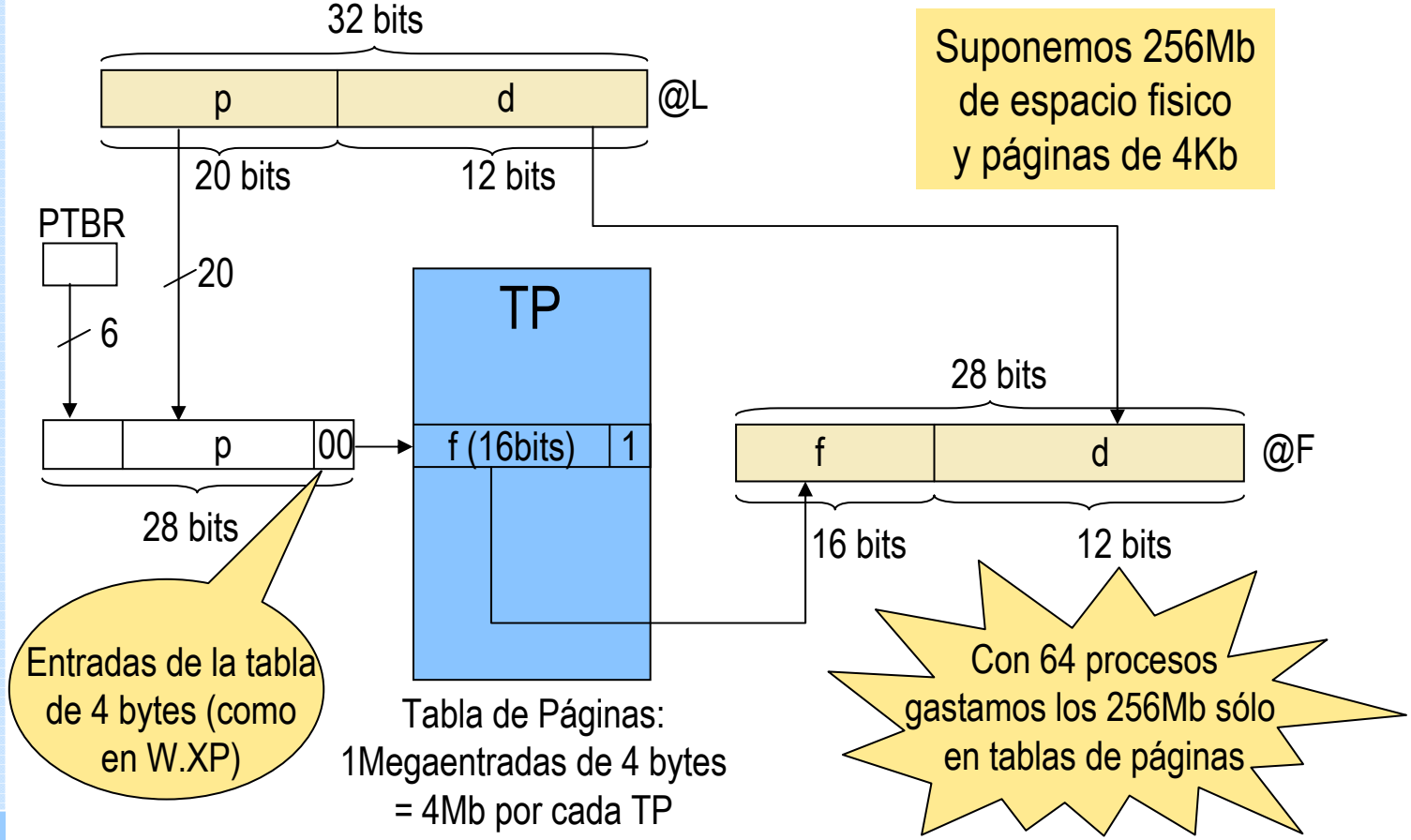
4.4.1 Paginación

gestión del S.O.

- S.O. mantiene una TP por cada proceso
  - En cambio de contexto notifica a MMU cuál debe usar
- S.O. mantiene una única TP para el propio S.O.
  - Proceso en modo sistema accede directamente a su mapa y al del S.O.
- S.O. mantiene tabla de marcos:
  - estado de cada marco (libre o ocupado, ...)
- Mucho mayor gasto en tablas que con asignación contigua
  - Es el precio de mucha mayor funcionalidad

4.4.1 Paginación
tamaño de la TP

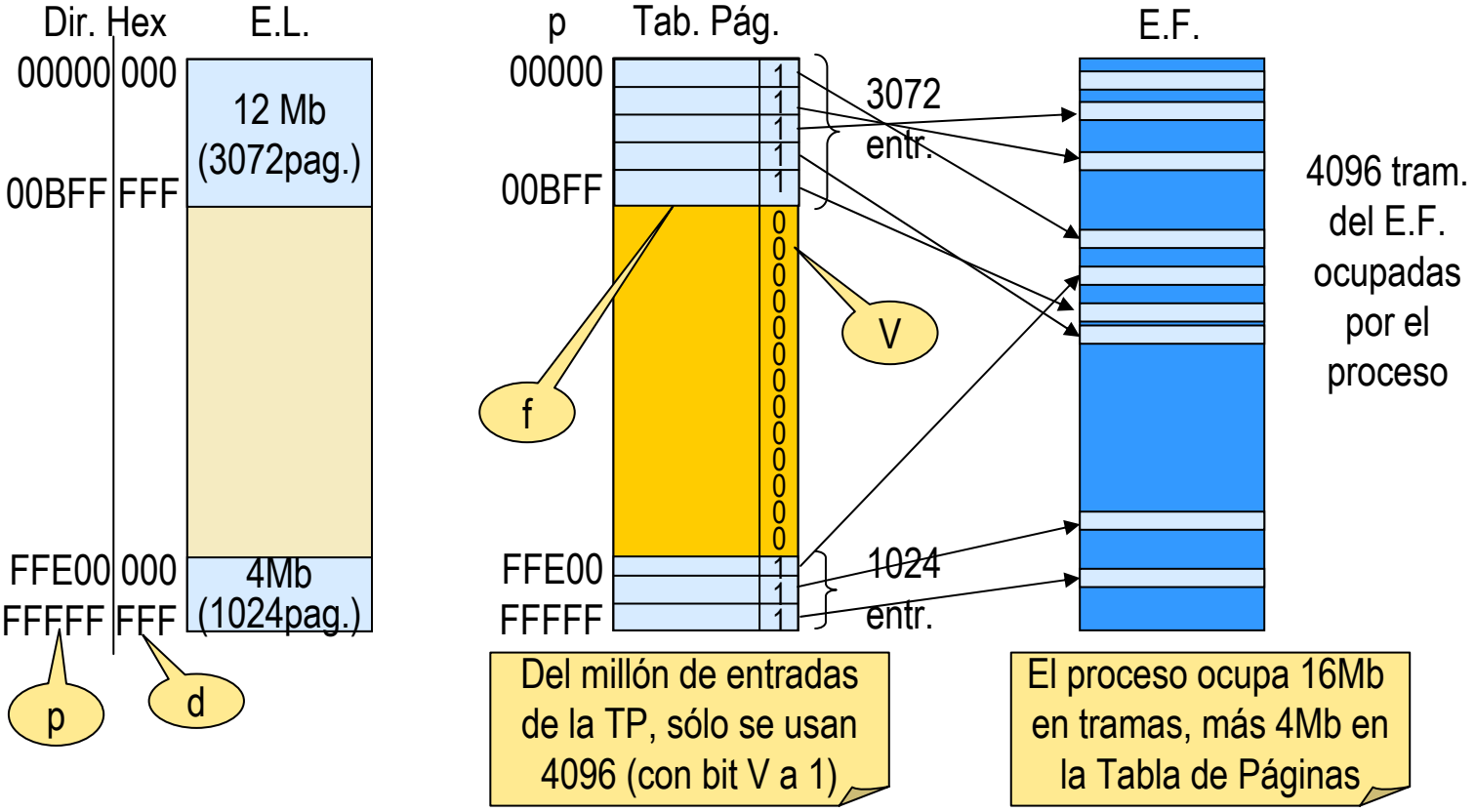
- Desperdicio de mem. en tablas de páginas
  - Sobre todo cuando el E.L. es grande
  - Ejemplo: E.L. de 32 bits (4Gb, como en Pent.4)



4.4.1 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.



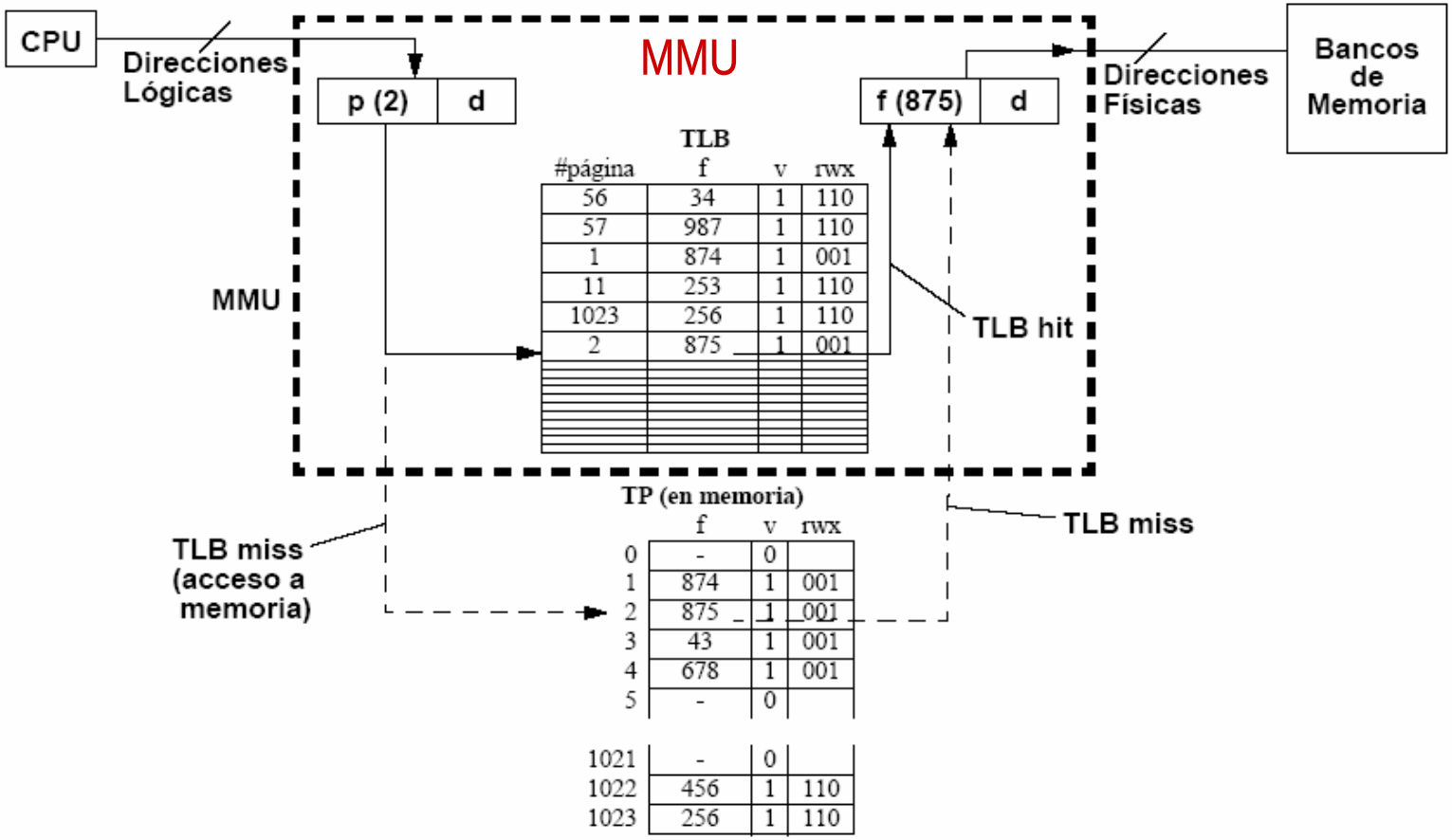
4.4.1 Paginación
problemas de la paginación

- TPs normalmente en memoria principal.
- 2 problemas: eficiencia y gasto de almacenamiento
- Eficiencia:
  - 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)
- Gasto de almacenamiento:
  - La TP no se aprovecha
    - En el ejemplo anterior, se usan 16Kb de los 4Mb de TP (el 0.4%)
    - Solución: Paginación multinivel
  - 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

4.4.1 Paginación

TLB

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



4.4.1 Paginación

TLB

- Memoria asociativa con info. sobre últimas páginas accedidas
  - *cache* de entradas de TP correspondientes a estos accesos
- 2 alternativas:
  - Entradas en TLB no incluyen información sobre proceso
    - Invalidar TLB en cambios de contexto
  - Entradas en TLB incluyen información sobre proceso
    - Registro de CPU debe mantener un ident. de proceso actual
- Gestionada por HW:
  - MMU consulta TLB: Si fallo usa la TP en memoria
  - "Casi" trasparente al S.O.
    - Volcar a TP en c.contexto para actualizar *Ref* y *Mod*
    - Invalidar, si entradas no incluyen información del proceso
- Diseño alternativo: TLB gestionada por SW

4.4.1 Paginación
TLB gestionada por SW

- TLB gestionada por SW: organización alternativa bastante usada actualmente
  - Traspasar al S.O. parte del trabajo de traducción
- MMU no usa tablas de páginas, sólo consulta TLB
- S.O. mantiene TPs que son independientes del HW
- Fallo en TLB –› Activa S.O.
- S.O. se encarga de:
  - Buscar “a mano” en TP la traducción
  - Rellenar (con posible reemplazo) TLB con la traducción
- Proporciona flexibilidad en diseño de S.O. pero menor eficiencia

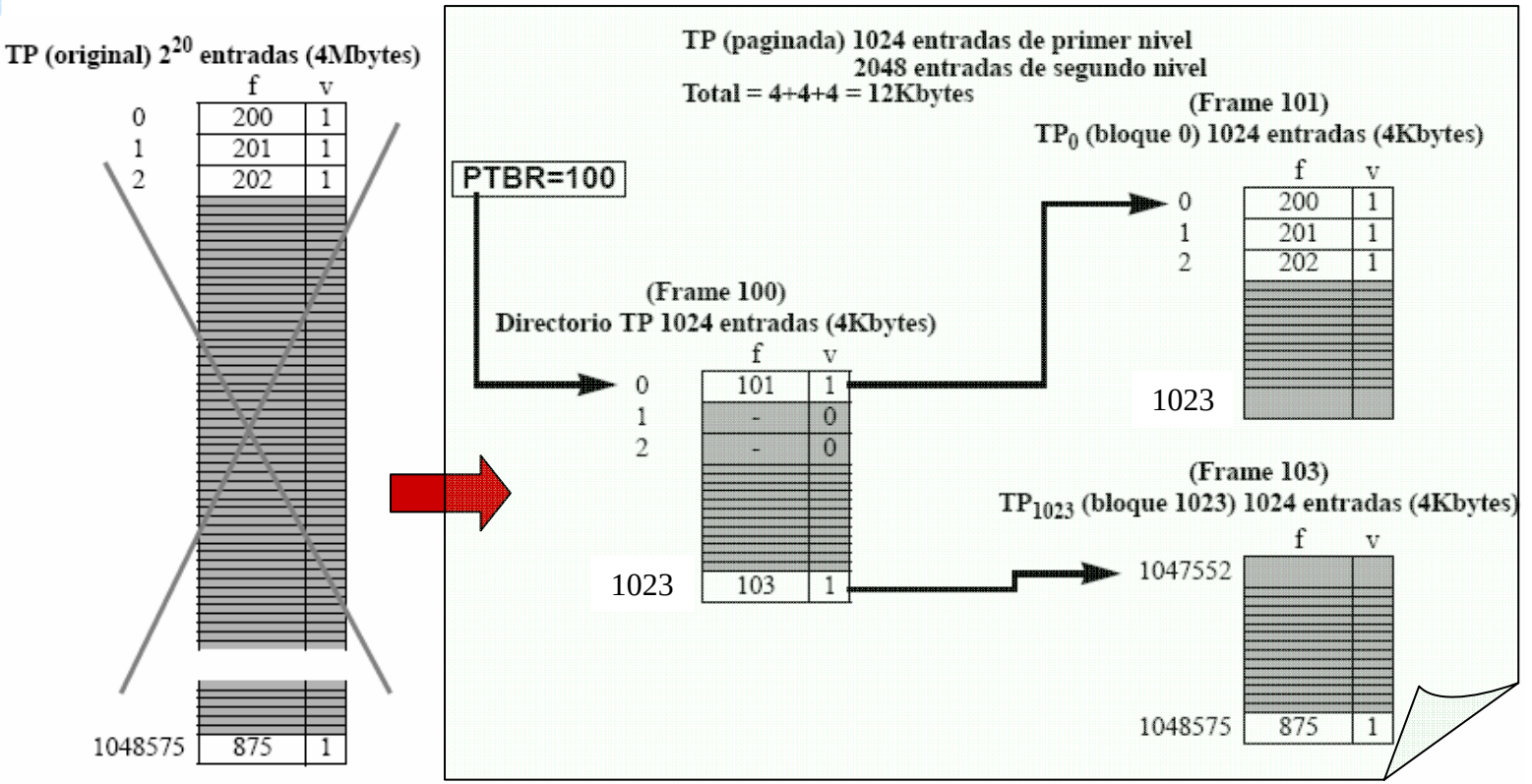
4.4.1 Paginación

TLB

- Implementación de la TLB
  - Es una caché y por tanto se implementa como tal
- Ejemplos comerciales:
  - DEC Alpha 21264: 128 entradas. 2 vías. 64Kb
  - 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, totalmente asociativa
  - AMD XP y Opteron
    - TLB L1 de 32 entradas
    - TLB L2 de 256 entradas en XP y 512 entradas en Opteron

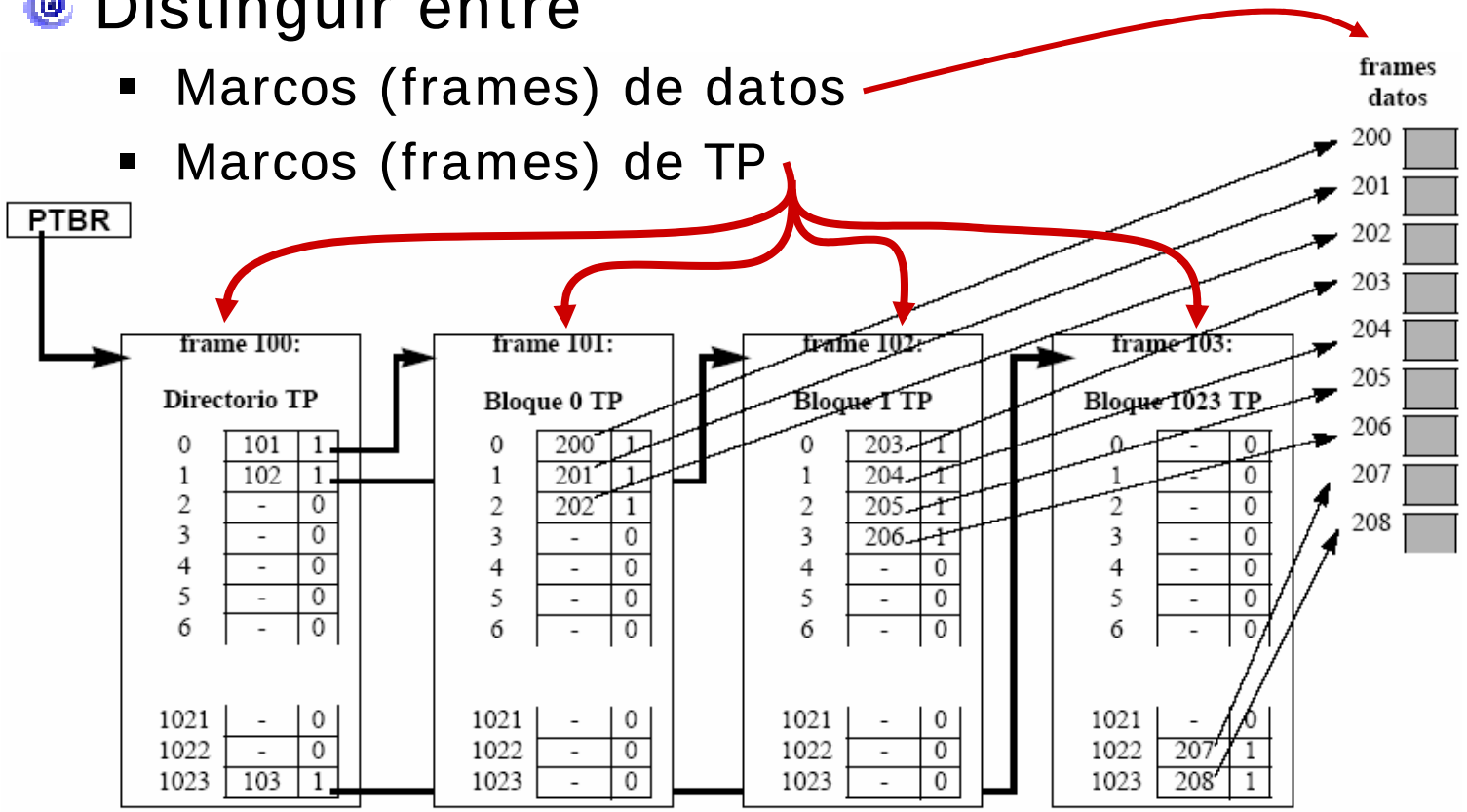
4.4.1 Paginación
paginación multinivel

- 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.



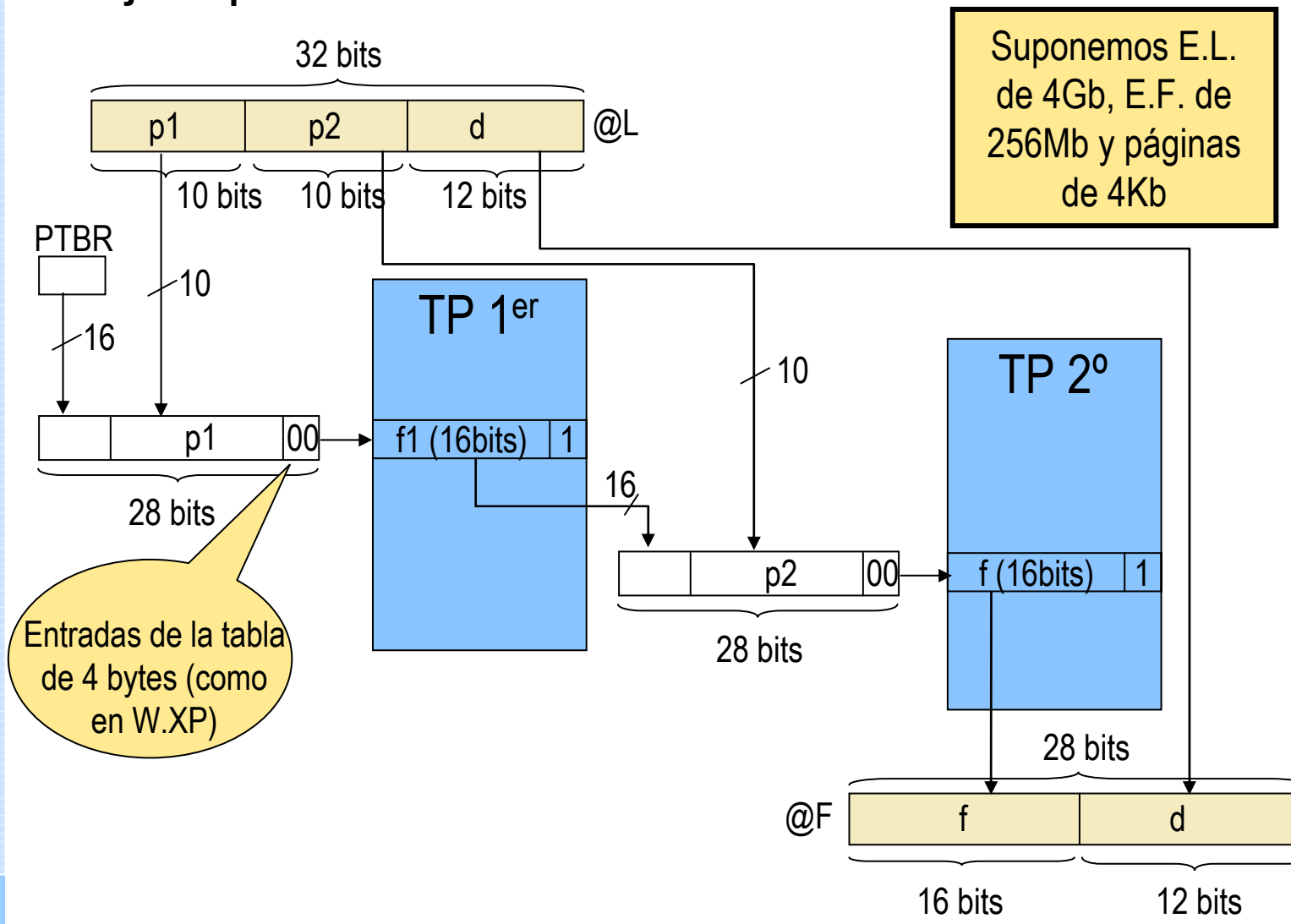
4.4.1 Paginación
paginación multinivel

- 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



4.4.1 Paginación
paginación multinivel

- Ejemplo de traducción detallado



4.4.1 Paginación
paginación multinivel

- Ventajas
  - Soluciona el problema del desperdicio de TP
    - Las tablas ocupan menos memoria y están mejor aprovechadas: si un proceso usa una parte pequeña de su espacio lógico se ahorra en espacio para almacenar TPs
  - En el ejemplo del proceso de 16M (12+4)
    - 2 niveles, páginas de 4K, dir. lógica 32 bits (10 bits por nivel) y 4 bytes por entrada
    - 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 !!)
  - Sólo se requiere que esté en memoria la TP de nivel superior. TPs restantes pueden estar en disco y traerse por demanda

4.4.1 Paginación
paginación multinivel

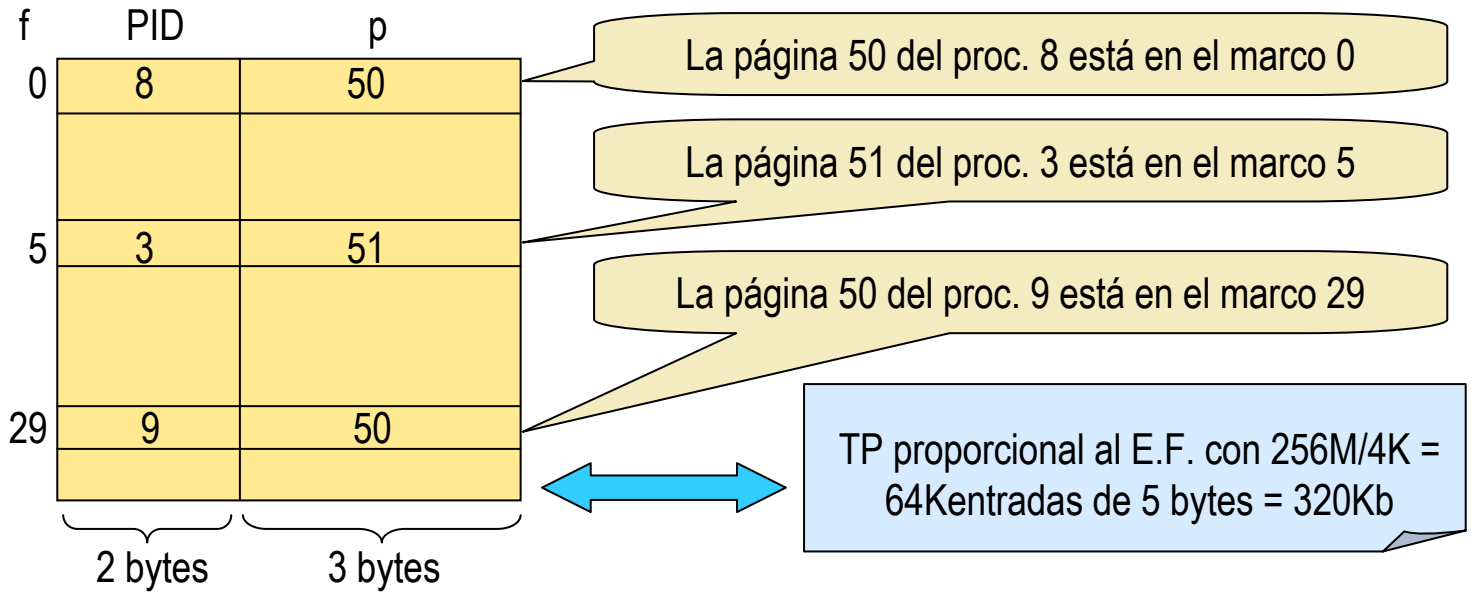
- Inconvenientes: perdida de rendimiento!
  - Un acceso a memoria se traduce en 3
    - 1º, para acceder a la TP de primer nivel
    - 2º, para acceder a la TP de segundo nivel
    - y 3º, para realizar el acceso al marco y leer/escribir el dato
- Generalizable a 3 o más niveles (M niveles):
    - Entrada de TP de nivel K apunta a TP de nivel K+1
    - Entrada de último nivel apunta a marco de página
  - Dirección lógica especifica la entrada a usar en cada nivel: 1 campo por nivel + desplazamiento
  - 1 acceso lógico –› M + 1 accesos a memoria
    - Uso de TLB
  - Si todas las entradas de una TP son inválidas
    - No se almacena esa TP
    - Se pone inválida la entrada correspondiente de la TP superior

4.4.1 Paginación
tabla invertida

- Procesadores actuales espacio lógico enorme (dirs. de 64 bits)
  - TPs muy grandes incluso usando multinivel
- Posible solución alternativa: Uso de TPs invertidas
  - Una entrada por cada marco indica página almacenada en él
    - Tamaño de TP proporcional a memoria principal
  - Es necesario guardar núm. de página y PID de proceso
- Procedimiento de traducción:
  - MMU usa TLB convencional
  - Si fallo en TLB –› MMU busca traducción en TP invertida
- Para evitar búsqueda secuencial en TP invertida:
  - Se organiza como una tabla *hash*
- Hay que tener en cuenta que aunque TP pequeña, S.O. debe guardar info. de páginas no residentes
- Tabla de páginas invertida
  - En lugar de buscar qué trama corresponde a cada página → qué pág. corresponde a cada trama

4.4.1 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 sóla 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)

4.4.2 Segmentación

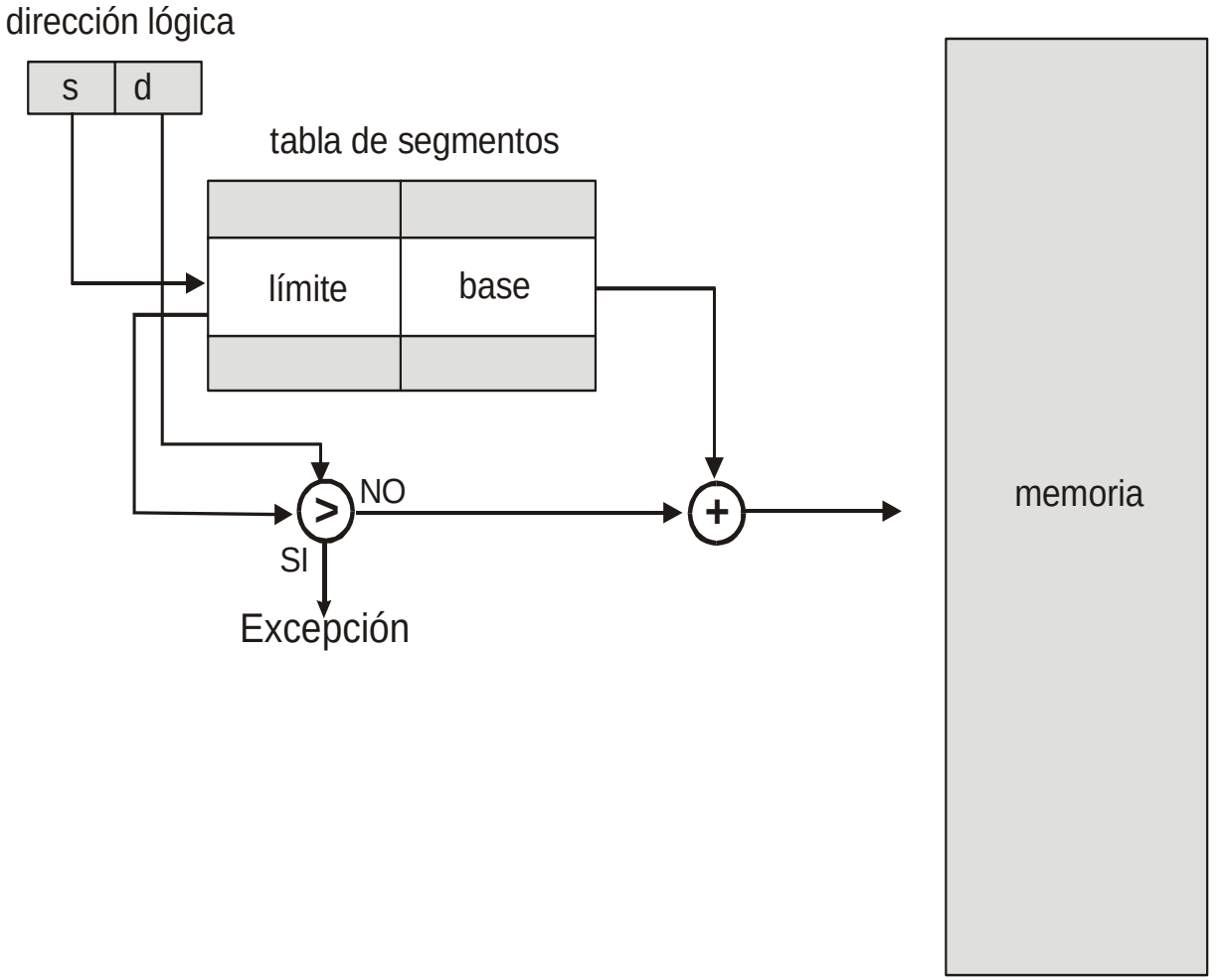
segmentación

- Particionar E.L. en seg. de tamaño variable
  - Generalización de reg. base y límite: 1 pareja por cada segmento
  - @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

4.4.2 Segmentación

segmentación

- Traducción de direcciones con seg.



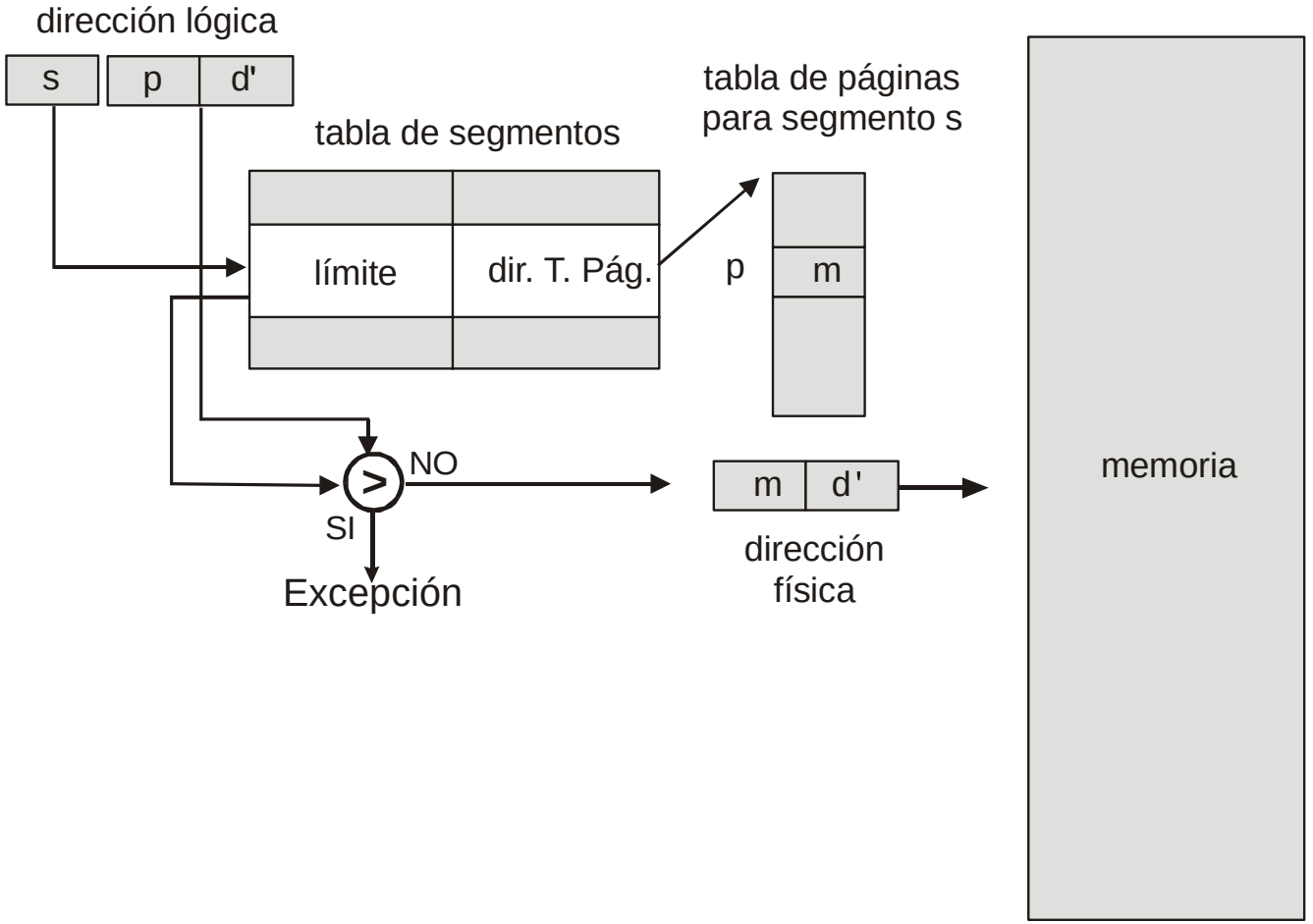
4.4.2 Segmentación
paginación vs. segmentación

- 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

4.4.3. Segmentación pag.

segmentación paginada

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



4.4.3. Segmentación pag.

segmentación paginada

- Entrada en TS apunta a una TP para el segmento
- "Lo mejor de los dos mundos"
- Segmentación:
  - Soporte directo de segmentos
  - Facilita operaciones sobre regiones:
    - Establecer protección –› Modificar sólo una entrada de TS
    - Definir compartición de segmento –› entradas de TS apuntando a la misma TP de segmento
- Paginación:
  - Asignación no contigua de segmento
  - Fragmentación interna

4.5 Mem. virtual

4.5 Mem. virtual

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 porque procesos exhiben localidad
  - Procesos sólo usan parte de su mapa en intervalo de tiempo
  - M. virtual: intentar que parte usada (*conjunto de trabajo*) resida en m. principal (*conjunto residente*)
- Dirección lógica ‹–› Dirección virtual
- Ventajas
  - Aumenta el grado máximo de multiprogramación
  - Permite ejecución de programas que no quepan en mem. ppal
  - Disminuye el tiempo de swapping respecto del modelo no residente.
    - Ahora descargas/cargas un trozo del programa (no todo)

4.5 Mem. virtual
implementación

- M. virtual: S.O. gestiona niveles de M. principal y M. secundaria
  - Transferencia de *bloques* entre ambos niveles
  - De M. secundaria a principal: por demanda
  - De M. principal a secundaria: por expulsión
- No adecuada para sistemas de tiempo real
- Normalmente **basada en paginación**
  - Segmentación pura no adecuada para memoria virtual
    - tamaño de segmentos variable
  - Paginación y segmentación paginada sí lo son:
    - Bloque transferido –› Página
    - M. virtual + Paginación –› Paginación por demanda

4.5 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

4.5 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

4.5 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 víctima)
        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.
```

4.5 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

4.5 Mem. virtual
algoritmos de asignación y reemplazo

- Política de asignación de espacio a los procesos:
  - ¿Cómo se reparten los marcos entre los procesos?: asignación fija o dinámica
- Estrategia de asignación fija:
  - Número de marcos asignados al proceso (conjunto residente) es constante
  - Puede depender de características del proceso:
    - tamaño, prioridad,...
  - No se adapta a las distintas fases del programa
  - Comportamiento relativamente predecible
  - Sólo tiene sentido que sea local
  - Arquitectura impone nº mínimo:
    - Por ejemplo: instrucción MOVE /DIR1, /DIR2 requiere un mínimo de 3 marcos: instrucción y dos operandos deben estar residentes para ejecutar la instrucción

4.5 Mem. virtual
algoritmos de asignación y reemplazo

- Estrategia de asignación dinámica:
  - Número de marcos varía dependiendo de comportamiento del proceso (y posiblemente de los otros procesos)
  - Asignación dinámica y local
    - proceso va aumentando o disminuyendo su conjunto residente dependiendo de su comportamiento
    - comportamiento relativamente predecible
  - Asignación dinámica y global
    - procesos se quitan las páginas entre ellos
    - comportamiento difícilmente predecible

4.5 Mem. virtual
algoritmos de reemplazo

- Algoritmos de reemplazo
  - Selecciona la víctima de entre los candidatos
  - Objetivo: minimizar el nº de F.P. en el futuro
  - Cada algoritmo descrito tiene versión local (criterio se aplica a las páginas residentes del proceso) y global (criterio se aplica a todas las páginas residentes)
  - Algoritmos
    - Optimo: necesita conocer el futuro. No implementable
    - Aleatorio: selecciona cualquier víctima
    - : 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

4.5 Mem. virtual

algoritmos de reemplazo

- Óptimo:
  - Selecciona como víctima a la pág. residente 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
  - Interés para estudios analíticos comparativos
  - 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	

4.5 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	

4.5 Mem. virtual

algoritmos de reemplazo

- No es una buena estrategia: la página que lleva mucho tiempo residente puede seguir accediéndose frecuentemente -> su criterio no se basa en el uso de la página.
- 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

4.5 Mem. virtual

algoritmos de reemplazo

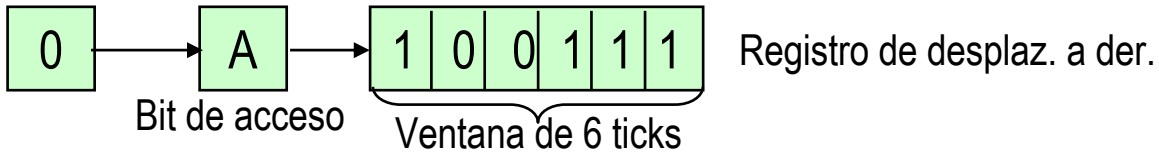
- LRU (Least Recently Used):
  - Aprende del algoritmo Óptimo
  - Pero en vez de mirar al futuro mira al pasado
    - Eliminar la página menos recientemente usada
  - Implementaciones:
    - Apuntar la hora de uso de cada página
    - Cola que se reordena en cada acceso. Ejemplo:

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		

4.5 Mem. virtual

algoritmos de reemplazo

- Aproximación LRU:
  - Más barata de implementar:
    - Añadir un bit A (de Acceso, o *Ref,* de referencia) 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

4.5 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

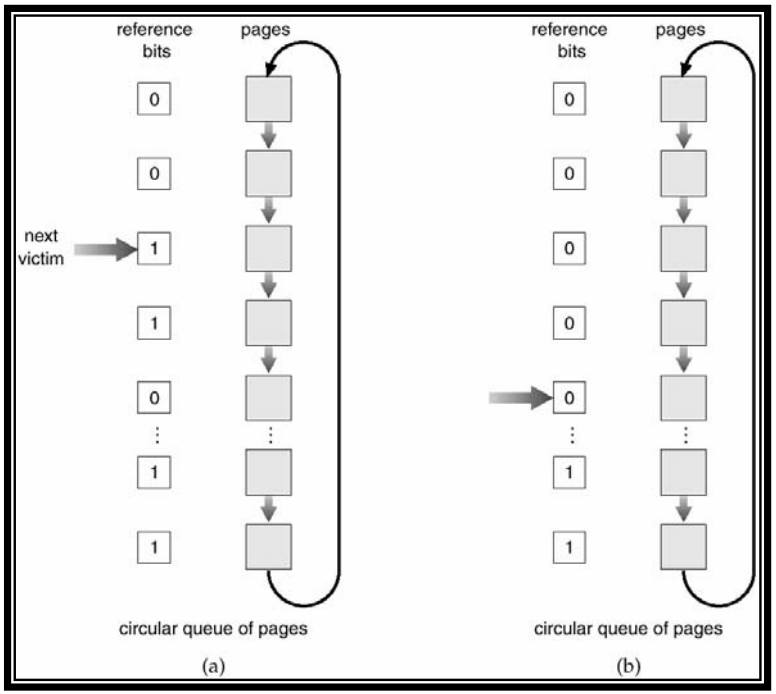
- LFU (Least Frecuently Used)
  - Contador para cada trama. Se incr. en cada acc.
  - Reemplazar la trama con menor valor
  - Problema:
    - Si una página se ref. mucho gana muchos puntos
    - Cuando deja de usarse sigue con todos los puntos
  - Solución: de vez en cuando div. por dos el cont.

4.5 Mem. virtual

algoritmos de reemplazo

- Clock (Segunda oportunidad, FINUFO)
  - Aplicar FIFO sólo a las pág. con A=0
  - Un puntero circular escanea los bits de una cola
    - 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:

Se usa una cola con los marcos de página en el orden en que se han ido llenando. Cuando todos los marcos están llenos, el último enlaza con el primero.



Fuente: Sistemas Operativos. Silberschatz, Galvin. Fig. 10-14

4.5 Mem. virtual

algoritmos de reemplazo

- FINUFO. Ejemplo con conjunto residente=4 y referencias a b a d g a f d g a f c b g

Conjunto					0g	0g	0g	0g	1g	1g	1g	·0g	0b	0b
				0d	0d	0d	·0d	·1d	·1d	·1d	·1d	0c	0c	0c
residente		0b	0b	0b	0b	0b	0f	0f	0f	0f	1f	0f	0f	·0f
	·0a	·0a	·1a	·1a	·1a	·1a	0a	0a	0a	1a	1a	0a	·0a	·0g
Tiempo	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Fallos	*	*		*	*		*					*	*	*

Candidato FIFO

* Fallo de página

4.5 Mem. virtual
estrategia del conjunto de trabajo

- Working Set (WS)
  - Intentar conocer el conjunto de trabajo de cada proceso
    - páginas usadas por el proceso en las últimas N referencias
  - Si conjunto de trabajo decrece se liberan marcos
  - Si conjunto de trabajo crece se asignan nuevos marcos
    - si no hay disponibles: suspender proceso(s)
    - se reactivan cuando hay marcos suficientes para c. de trabajo
  - Asignación dinámica con reemplazo local
  - Difícil implementación estricta
    - precisaría una MMU específica
  - Se pueden implementar aproximaciones:
    - Estrategia basada en frecuencia de fallos de página (PFF):
      - Controlar tasa de fallos de página de cada proceso

4.5 Mem. virtual

ejemplo de working set

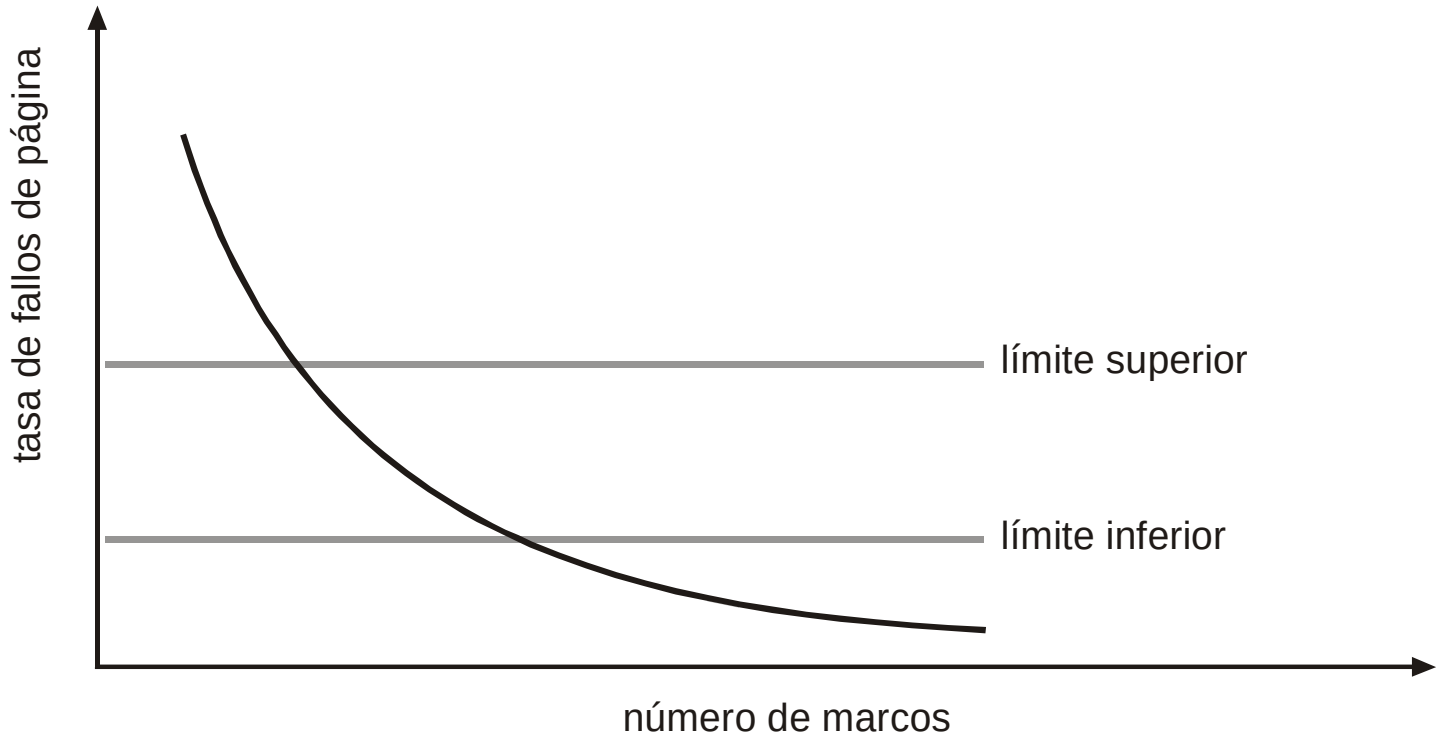
- Se define el conjunto working set de un proceso en una ventana de tiempo T como el conjunto de páginas distintas referenciadas entre el instante actual t y t-T+1.
- Ejemplo con ventana=4 y referencias a b a d g a f d g a f c b g

Conjunto residente					g		g	g	g	g	g	c	b	g
				d	d	g	f	f	f	f	f	g	c	b
		b	b	b	b	d	d	d	d	d	d	f	f	c
	a	a	a	a	a	a	a	a	a	a	a	a	a	f
Tiempo	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Fallos	*	*		*	*		*					*	*	*

4.5 Mem. virtual

estrategia basada en el número de fallos

- Si tasa < límite inferior se liberan marcos aplicando un algoritmo de reemplazo
- Si tasa > límite superior se asignan nuevos marcos
  - Si no marcos libres se suspende algún proceso



4.5 Mem. virtual

buffering de páginas

- Peor caso en tratamiento de fallo de página:
  - 2 accesos a dispositivo (salvar la reemplazada y leer la cargada)
- Alternativa: mantener una reserva de marcos libres
- Fallo de página: siempre usa marco libre (no reemplazo)
- Si número de marcos libres < umbral
  - "demonio de paginación" aplica repetidamente el algoritmo de reemplazo:
    - páginas no modificadas pasan a lista de marcos libres
    - páginas modificadas pasan a lista de marcos modificados
      - cuando se escriban a disco pasan a lista de libres
      - pueden escribirse en tandas (mejor rendimiento)
- Si se referencia una página mientras está en estas listas:
  - fallo de página la recupera directamente de la lista (no E/S)
  - puede arreglar el comportamiento de algoritmos "malos"

4.5 Mem. virtual
retención de páginas en memoria

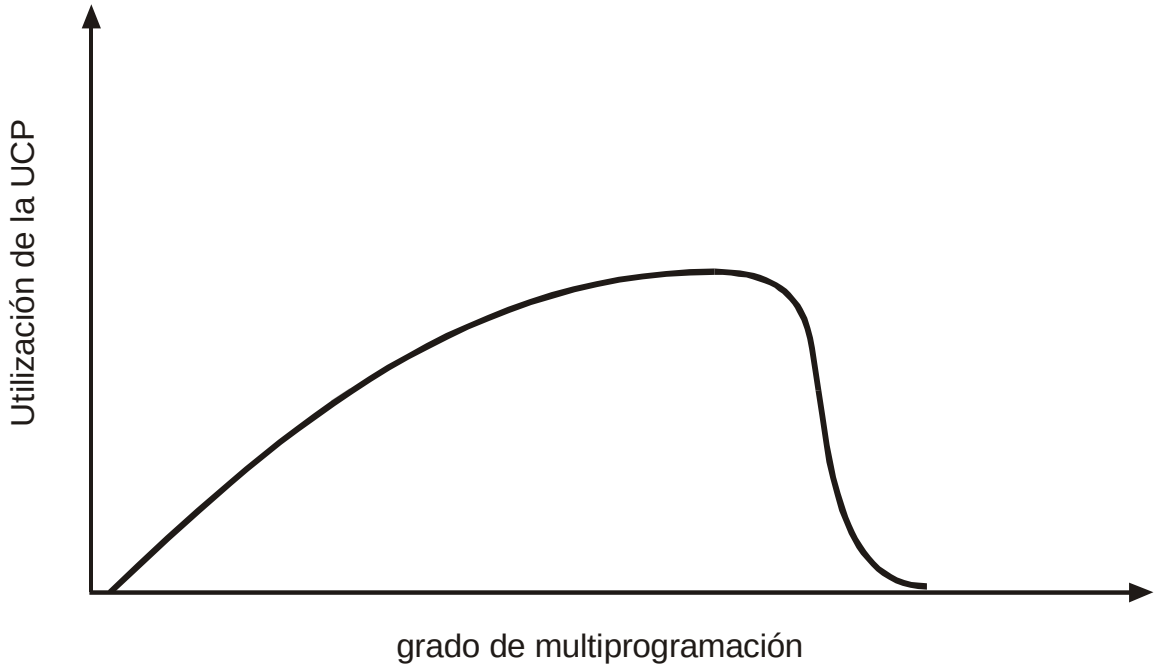
- Páginas marcadas como no reemplazables
- Se aplica a páginas del propio S.O.
  - S.O. con páginas fijas en memoria es más sencillo
- También se aplica mientras se hace DMA sobre una página
- Algunos sistemas ofrecen a aplicaciones un servicio para fijar en memoria una o más páginas de su mapa
  - Adecuado para procesos de tiempo real
  - Puede afectar al rendimiento del sistema
  - En POSIX syscall `mlock`

4.5 Mem. virtual
hiperpaginación (*thrashing*)

- Thrashing
  - Tasa excesiva de fallos de página de un proceso o en el sistema
  - Con asignación fija: Hiperpaginación en P_i
    - Si conjunto residente de P_i < conjunto de trabajo P_i
  - Con asignación variable: Hiperpaginación en el sistema
    - Si nº marcos disponibles < Σ conjuntos de trabajo de todos
    - Grado de utilización de CPU cae drásticamente
    - Procesos están casi siempre en colas de dispositivo de paginación
    - Solución: Control de carga
      - disminuir el grado de multiprogramación
      - suspender 1 o más procesos liberando sus páginas residentes
    - Problema: ¿Cómo detectar esta situación?

4.5 Mem. virtual

thrashing



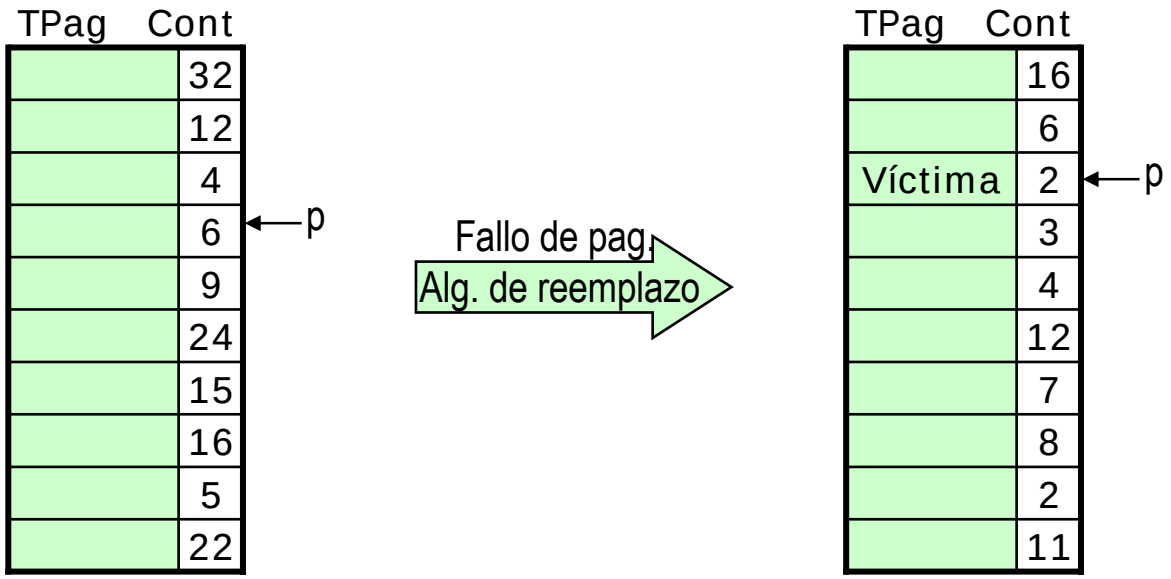
4.5 Mem. virtual
control de carga y reemplazo global

- Algoritmos de reemplazo global no controlan hiperpaginación
  - ¡Incluso el óptimo!
  - Necesitan cooperar con un algoritmo de control de carga
- Ejemplo: UNIX 4.3 BSD
  - Reemplazo global con algoritmo del reloj
    - Variante con dos “manecillas”
  - Uso de *buffering* de páginas
    - “demonio de paginación” controla nº de marcos libres
  - Si número de marcos libres < umbral
    - “demonio de paginación” aplica reemplazo
  - Si se repite con frecuencia la falta de marcos libres:
    - Proceso “swapper” suspende procesos

4.5.1 Ejemplos

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



4.5.1 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

4.5.1 Ejemplos

windows xp

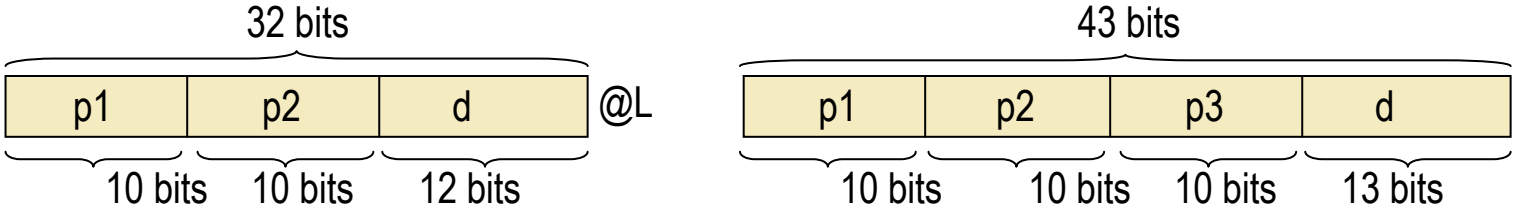
Memoria Virtual en Windows XP

- 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

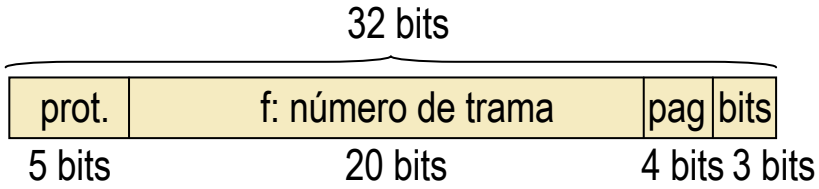
4.5.1 Ejemplos

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