Sistemas operativos avanzados

Gestión de memoria

Contenido

- **Aspectos generales de la gestión de memoria**
- Modelo de memoria de un proceso
- Esquemas de gestión de la memoria del sistema
- Memoria virtual

Aspectos generales de la gestión de memoria

- SO multiplexa recursos entre procesos
  - Gestión procesos: Reparto (espacial y temporal) de procesador
  - Gestión de memoria: Reparto (espacial y temporal) de memoria
- Gestión integral de la memoria: no solo SO
  - Compilador, montador y hardware de gestión de memoria
  - Presentación se centra en SO
- Gestor de memoria: elevada complejidad y dependencia del HW
- Ofrece un nº muy reducido de llamadas al sistema
  - Pero posee una gran complejidad
- El largo camino desde el acceso simbólico al real
  - Código de programa hace referencia a variables y funciones
  - En tiempo de ejecución programa genera accesos a memoria
  - Requiere varias etapas de transformación
    - compilador, montador, SO y HW se reparten el trabajo

Niveles de gestión de memoria

- **Nivel de sistema**
  - Reparto de memoria del sistema entre procesos
- **Nivel de proceso**
  - Reparto de memoria del proceso entre regiones
- **Nivel de región** (si aplicable)
  - Reparto de espacio de región entre sus zonas
    - No gestionado por sistema operativo
    - Por ejemplo, región de *heap*
- Cada nivel es caso del problema gral. de la asignación de espacio
  - En cada nivel se puede usar esquema diferente
  - Un nivel reparte espacio proporcionado por nivel superior

Objetivos del sistema de gestión de memoria

- Reparto seguro y eficiente de la memoria entre procesos activos
  - Segura: procesos no pueden afectarse entre sí ni al SO
    - Mitigar ataques debido a vulnerabilidades de programas y SO
  - Eficiente: rápida y con buen aprovechamiento de la memoria
- Satisfacer las necesidades de los programas y del SO
  - Espacios lógicos independientes para procesos y SO
  - Protección de los procesos y del SO
  - Compartir memoria para comunicarse y optimizar su uso interno
  - Soporte de las regiones de los procesos y del SO
  - Buen aprovechamiento del espacio de memoria

Un experimento

fperez@box1: ~/SII/experimentos_memoria/1_experimento

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int variable=10;

int main(int argc, char **argv) {
        printf("Proceso %d: variable (dir=%p; val=%d)\n", getpid(),
                &variable, variable++);
        printf("%d: Antes de fin; pulsa para acabar ", getpid());
        getchar();

        return 0;
}
~
~
~
~
~
~
"programa.c" 17L, 288C                                  6,0-1         Todo
```

Espacios lógicos independientes

Actividades Terminal lun 12:48

Papelera

fperez@box1: ~/libro/memoria

Archivo Editar Ver Buscar Terminal Ayuda

```
fperez@box1:~/libro/memoria$ ./prog
Proceso 7140: variable (dir=0x601040; val=10)
7140: Antes de fin; pulsa para acabar
```

fperez@box1: ~/libro/memoria

Archivo Editar Ver Buscar Terminal Ayuda

```
fperez@box1:~/libro/memoria$ ./prog
Proceso 7142: variable (dir=0x601040; val=10)
7142: Antes de fin; pulsa para acabar
```

fperez@box1: ~/libro/memoria

Archivo Editar Ver Buscar Terminal Ayuda

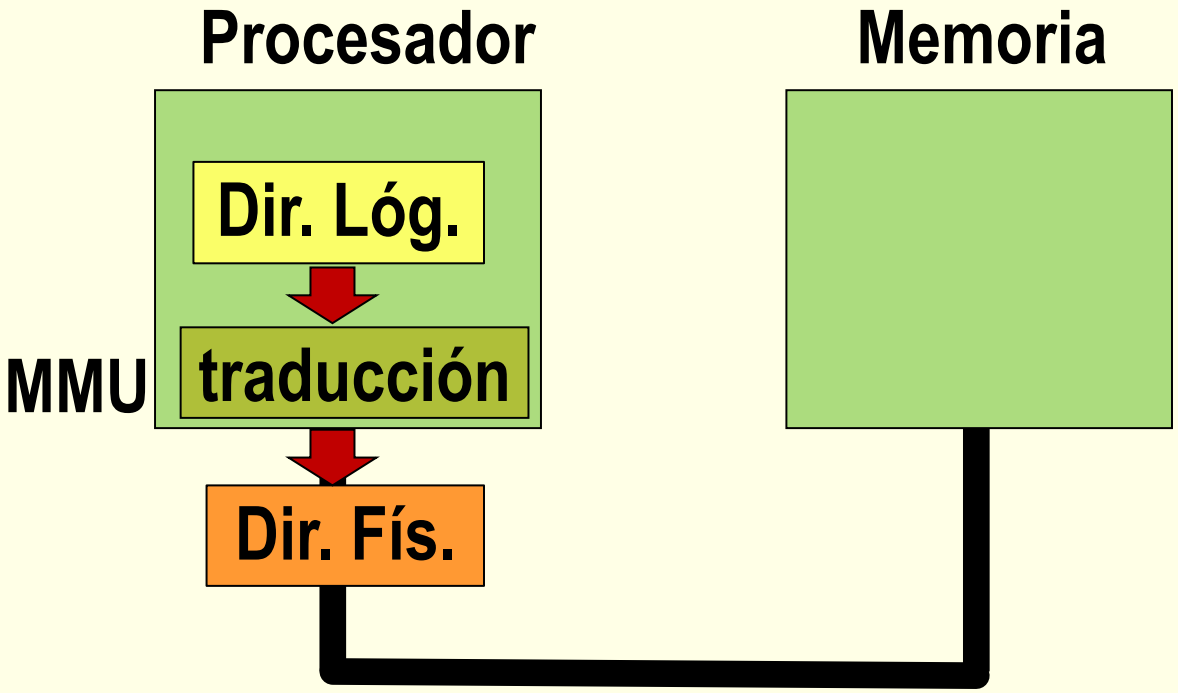
```
  400597:       8b 1d a3 0a 20 00       mov    0x200aa3(%rip),%ebx        # 601040 <variable>
  40059d:       8d 43 01                lea    0x1(%rbx),%eax
  4005a0:       89 05 9a 0a 20 00       mov    %eax,0x200a9a(%rip)        # 601040 <variable>
  4005a6:       e8 c5 fe ff ff          callq  400470 <getpid@plt>
  4005ab:       89 d9                   mov    %ebx,%ecx
  4005ad:       48 8d 15 8c 0a 20 00    lea    0x200a8c(%rip),%rdx        # 601040 <variable>
:
```

Compilado con -no-pie

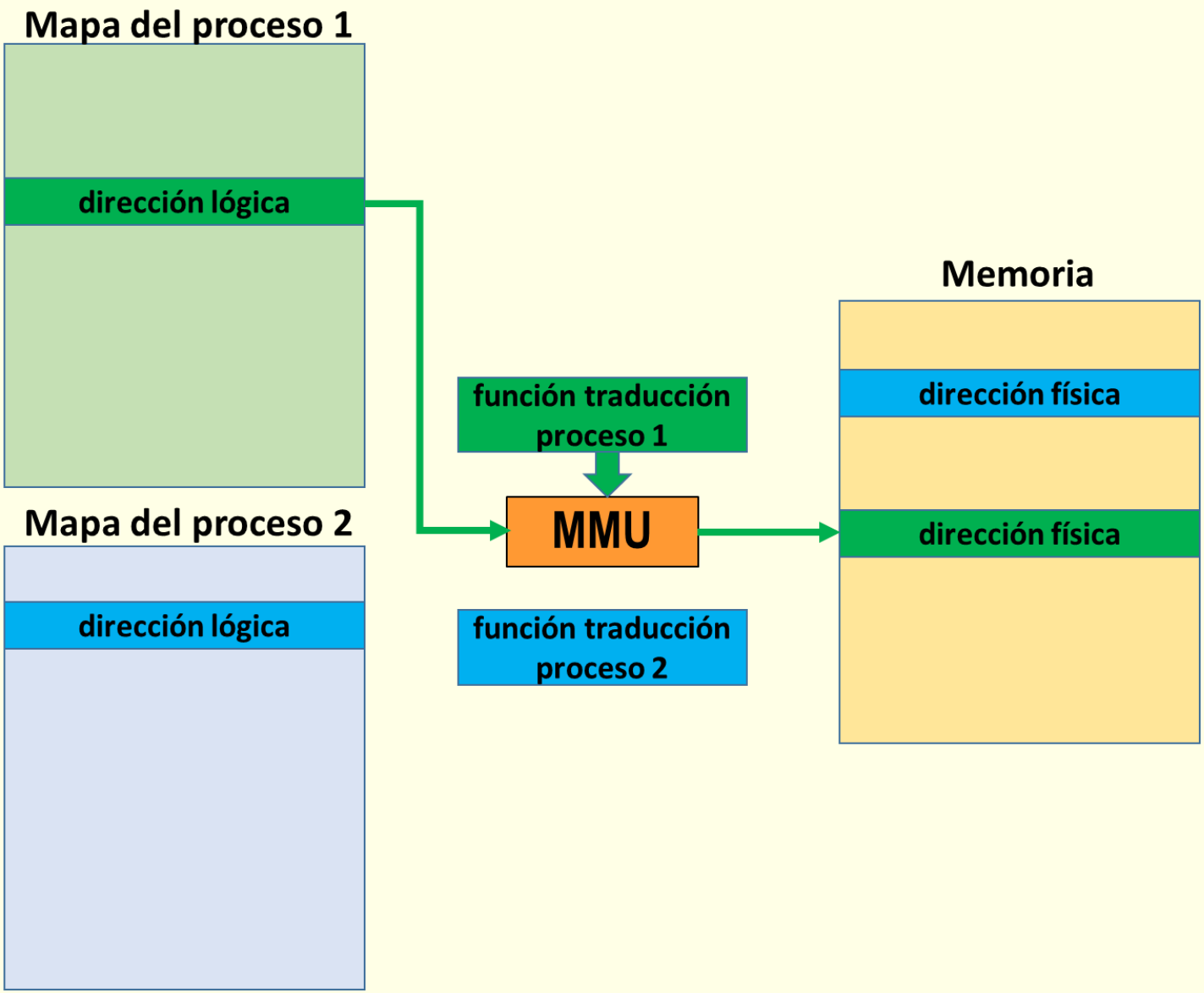
Espacios lógicos independientes para procesos

- Código en ejecutable incluye referencias entre 0 y N
  - En SO multiprogramado: necesidad de reubicar (traducir)
- Cada proceso cree tener su mapa/espacio lógico propio entre 0 y N
  - Función traducción hace corresponder a **d. físicas asignadas**
  - SO crea función de traducción → MMU la aplica
  - En cambio de proceso SO instala función de traducción de este
- Protección entre procesos: no pueden afectarse entre sí ni al SO
  - Función de traducción asegura *estanqueidad*
    - Hace corresponder direcciones físicas distintas a los procesos
- Pero permitiendo que compartan memoria de forma controlada
  - Dir. lógicas de 2 o más procesos asociadas a mismas d. físicas
  - Para optimizar uso: p.e. procesos comparten código *bash* o *libc*
  - Como medio de comunicación eficiente

Modo de operación de la MMU



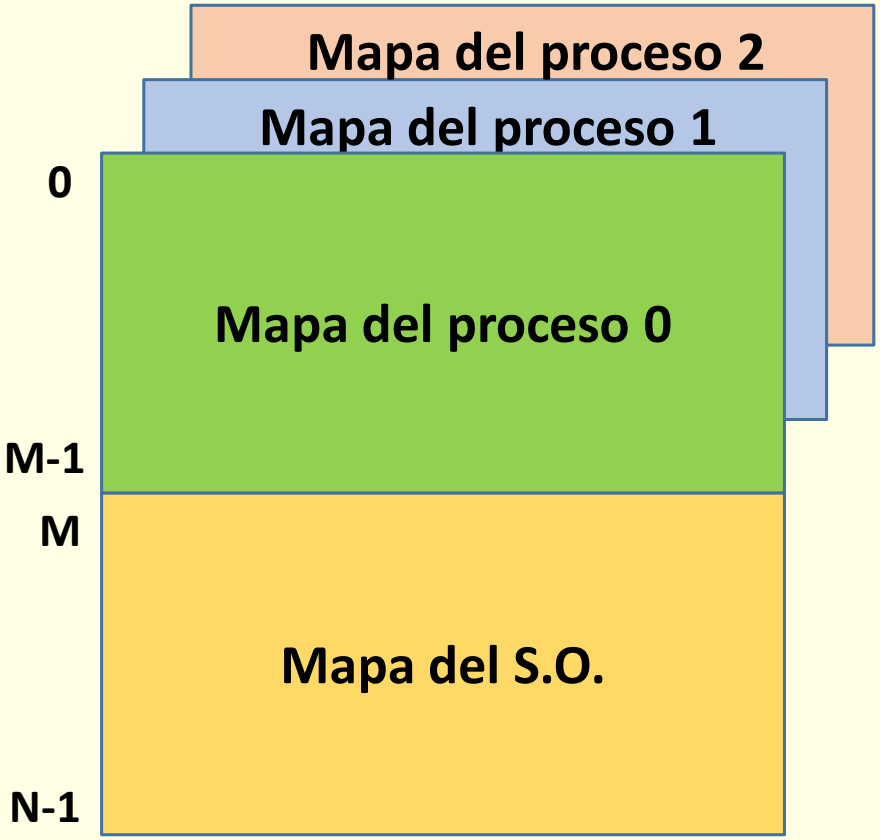
Mapa de memoria del proceso



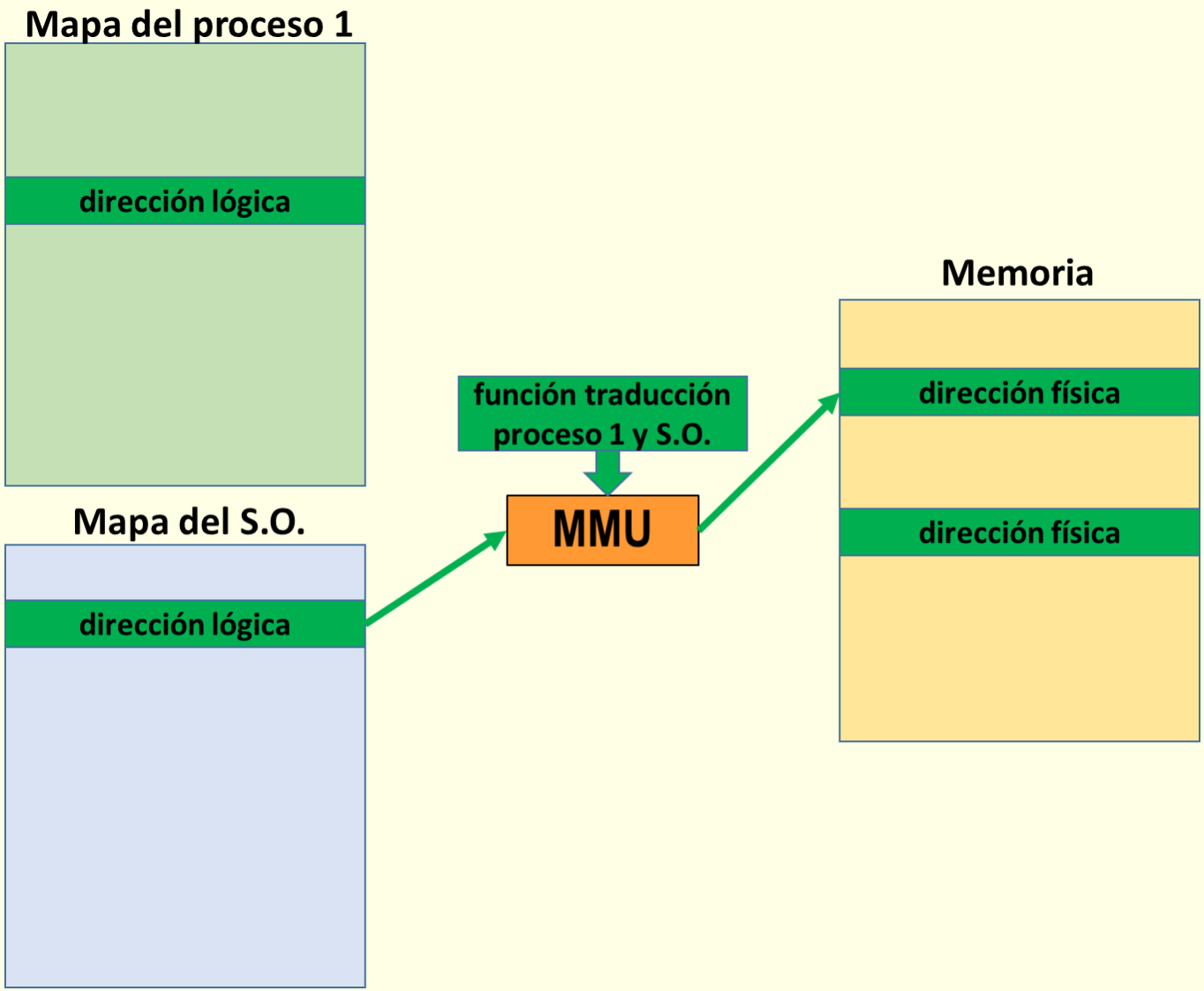
Espacio lógico para el Sistema Operativo

- Preferible que SO también tenga su propio espacio lógico (mapa)
  - Direcciones de SO también son traducidas
  - Flexibiliza ubicación de SO en memoria
- Esquema habitual: mapa común SO y proceso activo
  - Proceso mapa de 0 a M-1 y SO de M a N-1
    - Linux x86-32 bits → 0–0xBFFFFFFF (3GB; 1GB restante SO)
    - Linux x86-64 bits → 0–2^{47}-1 (128TB; 128TB restantes para SO)
  - Acceso directo de SO a dirección del proceso
    - SO usa directamente parámetro de llamada que es una dirección
  - Mapa SO activo pero no accesible mientras ejecuta proceso
  - Si proceso accede a dirección de SO se produce excepción
  - Problemas potenciales de seguridad (*Meltdown*)
    - Programa accede a dir. SO y con valor obtenido a un vector suyo
    - Acceso 1º produce excepción; 2º acceso no debería suceder
    - Por ej. especulativa sí ocurre: UCP elimina de caché valor leído 1º
    - Pero 2º queda en caché y puede deducirse valor 1º a partir de 2º

Espacio lógico común de proceso y del SO



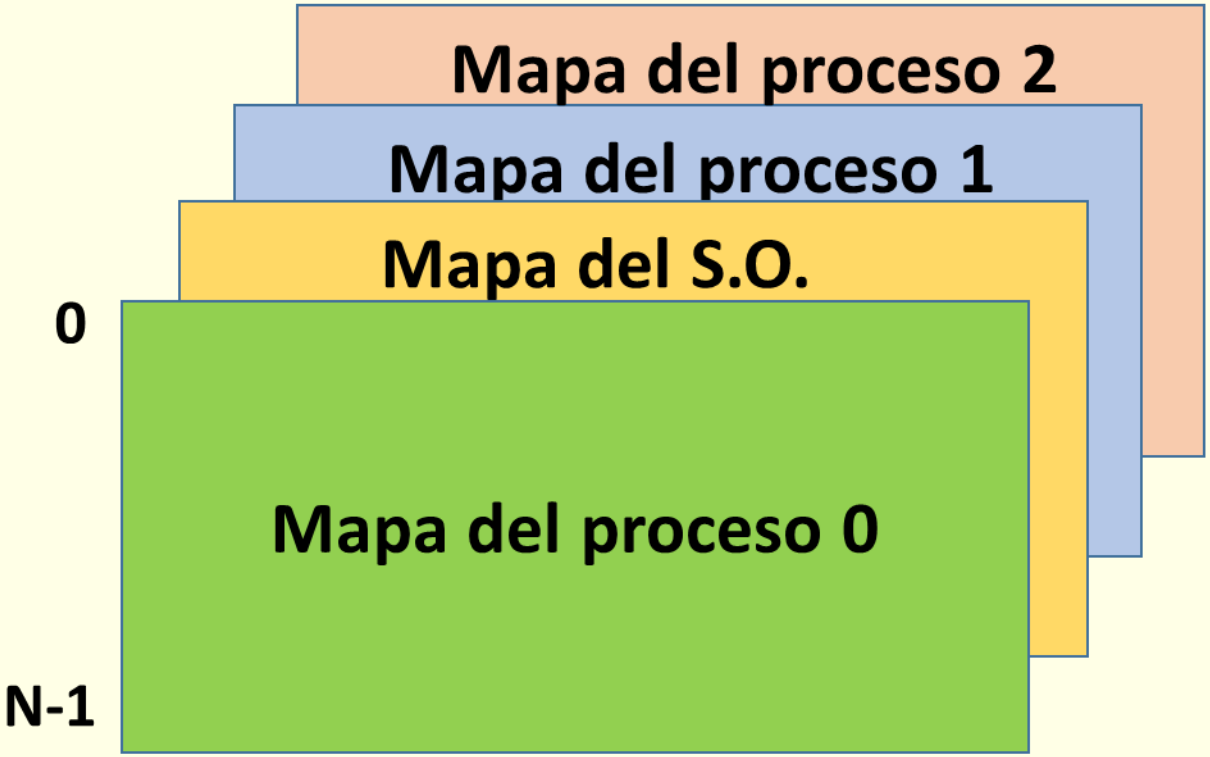
Espacio lógico común de proceso y del SO



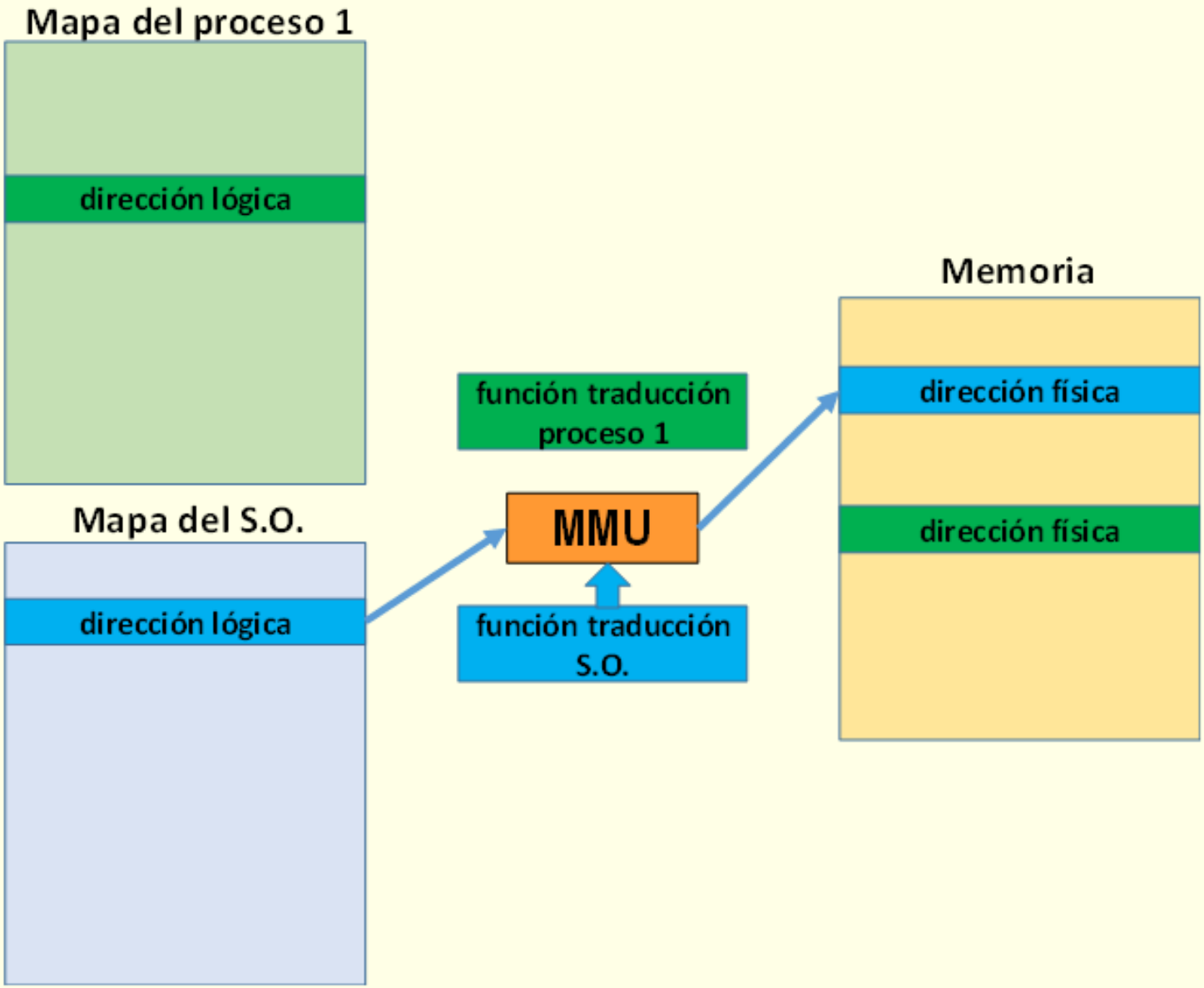
Espacio lógico independiente para el SO

- Alternativa: espacio lógico independiente para SO y proceso activo
  - Proceso mapa de 0 a N-1 y SO de 0 a N-1
    - Necesidad de convivencia mínima parte mapa de SO con procesos
  - No acceso directo de SO a direcciones de proceso
    - Necesita usar mecanismos que causan ineficiencia
  - Mapa SO no activo mientras ejecuta proceso
  - Elimina problemas potenciales de seguridad
- *Kernel page-table isolation* (KPTI)
  - Intenta eliminar problemas de seguridad pero con < ineficiencia
  - Como esquema habitual:
    - Reparto de mapa para proceso de 0 a M-1 y para SO de M a N-1
    - Cuando ejecuta SO (m. privilegiado): activos ambos mapas
    - Igual que con esquema habitual: acceso directo a mapa de usuario
  - Cuando ejecuta proceso (m. usuario): solo activo mapa proceso
    - Solo instalada en MMU la 1ª parte de la función de traducción

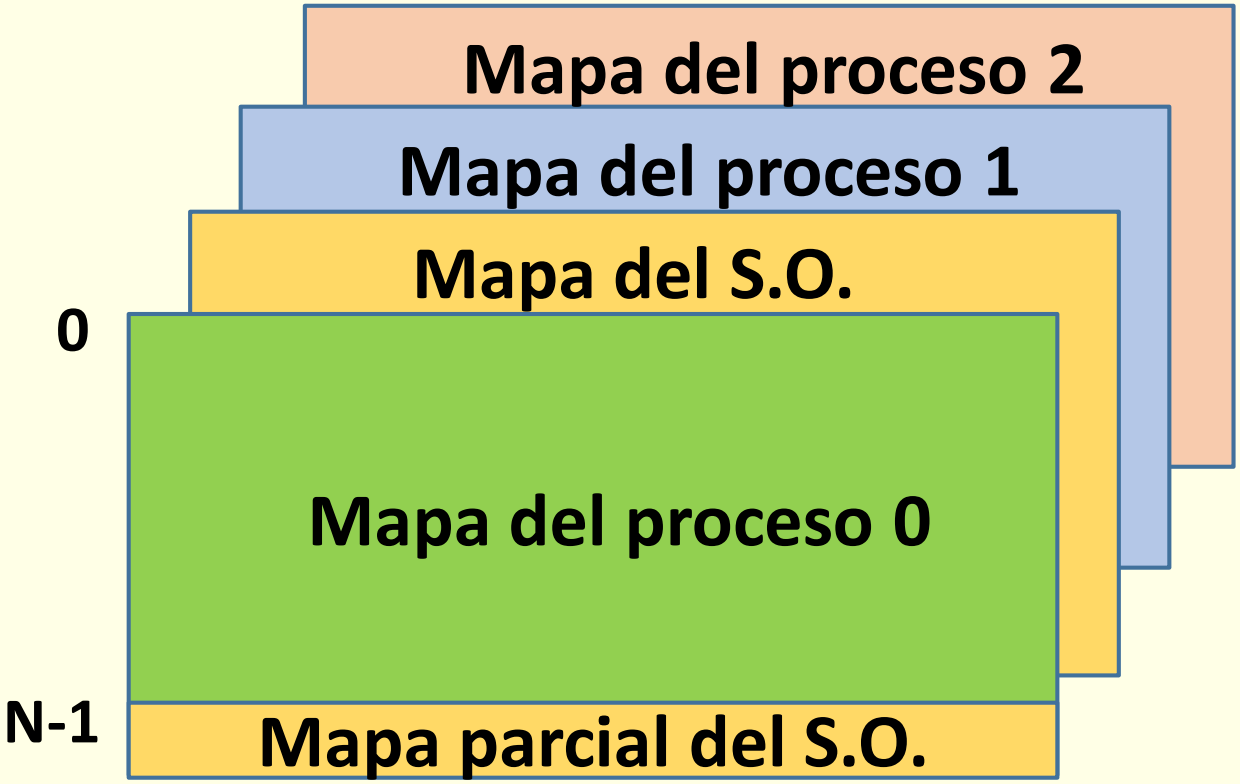
Espacio lógico independiente de proceso y SO



Espacio lógico independiente de proceso y SO



Espacio independiente proc/SO: mínima convivencia



Soporte de regiones de procesos y de SO

- Mapa de memoria de un proceso es heterogéneo
  - Formado por conjunto de regiones con distintas características
  - Código solo ejecutar; constantes solo leer; datos leer/escribir…
  - Función traducción especifica permisos acceso a una dirección
  - Si se produce tipo acceso no permitido, MMU genera excepción
- Mapa de memoria de un proceso es dinámico
  - Regiones cambian de tamaño, se crean y destruyen
    - La pila y la región de memoria dinámica (*malloc*) crecen
    - Cuando se crea un *thread* hay que habilitarle una pila…
  - "Huecos" en el espacio lógico no deben consumir memoria
  - Función traducción especifica que una dirección es inválida
  - Si se produce acceso a dir. inválida, MMU genera excepción
- SO debe detectar inmediatamente operaciones no permitidas
  - Facilita depuración del programa
- Aplicable también a regiones del SO

Soporte de regiones y seguridad

- Ataque habitual: desbordamiento de pila o *heap* (p.e. infame gets)
  - Consigue inyectar código malicioso (*shellcode*) y ejecutarlo
- Defensa: quitar permiso de ejecución de pila y *heap*
- Nuevas técnicas de ataque: *Return-Oriented Programming* (ROP)
  - Se inyecta en pila retornos a func. del programa o de bibliotecas
    - Ejecutando trocitos de funciones benignas realiza labor maligna
    - No requiere permiso de ejecución en pila o *heap*
- Ataque ROP facilitado por conocimiento de ubicación de regiones
- Defensa: *address space layout randomization* (ASLR)
  - Aleatoriedad en ubicación de regiones
  - PIE: *Position Independent Executables*
    - Más seguridad pero cierta pérdida de eficiencia
    - NOTA: por eso el experimento se compila con *-no-pie*
- En SO: *Kernel address space layout randomization* (KASLR)

Compartimiento de memoria

- Dir. lógicas de procesos corresponden con misma dir. física
- Mecanismo de comunicación entre procesos
- Permite mejor aprovechamiento
  - Compartir código de programas o de bibliotecas
- Datos de programa y de biblioteca no deben compartirse
  - Pero inicialmente idénticos
  - Diferir copia de cada dato hasta que se modifique (COW)
  - También aplicable a `fork`

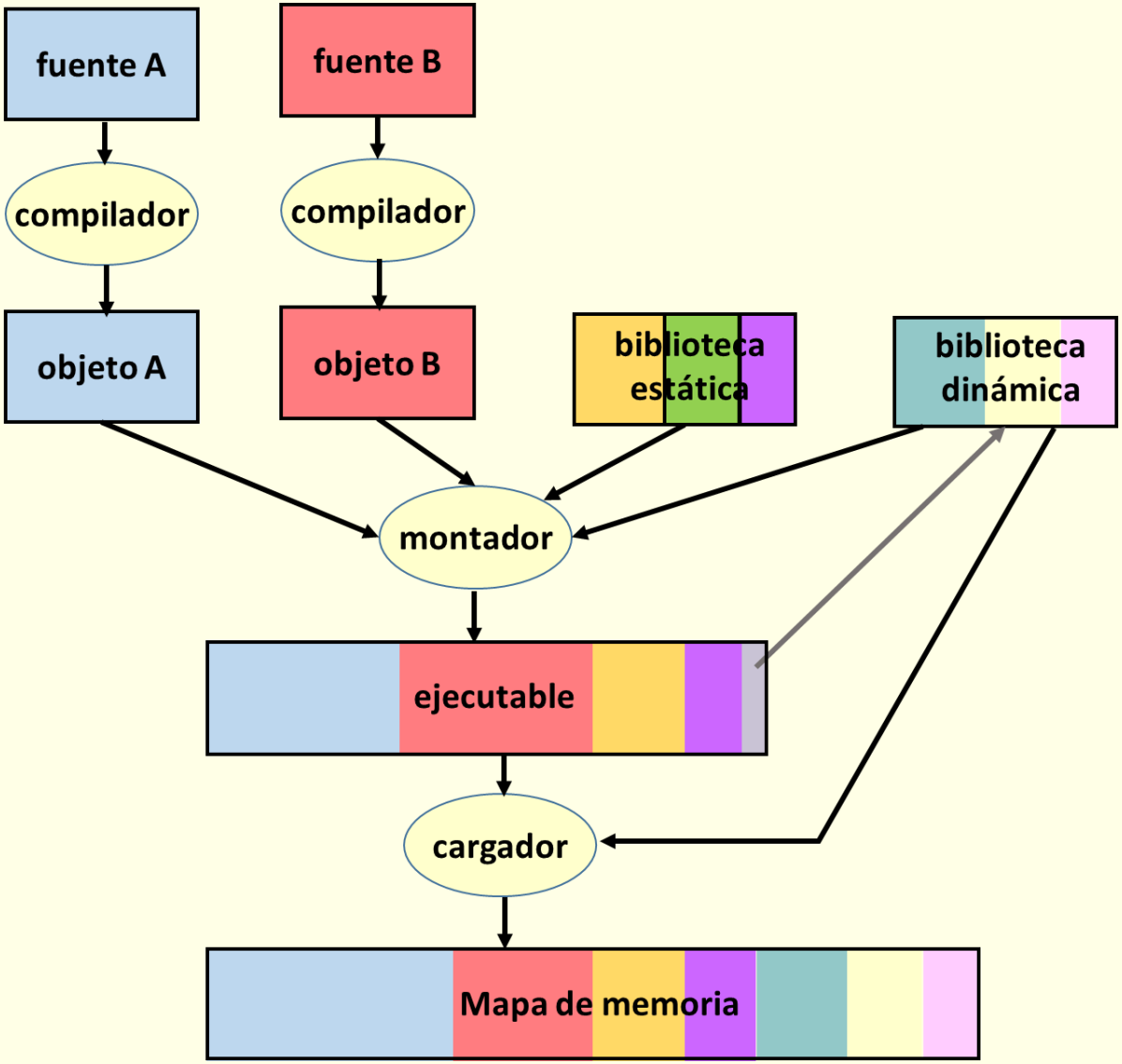
Buen aprovechamiento de la memoria

- Todo byte debería almacenar información de utilidad pero...
  - Desperdicio inherente a la propia gestión (**fragmentación**)
    - Externa o interna
  - Gasto de propia gestión de memoria (estructuras de datos)
- Mejor aprovechamiento → mayor grado de multiprogramación
  → mejor rendimiento
- Para mejorar rendimiento, uso de **memoria virtual**
  - Requisito de todo SO de propósito general

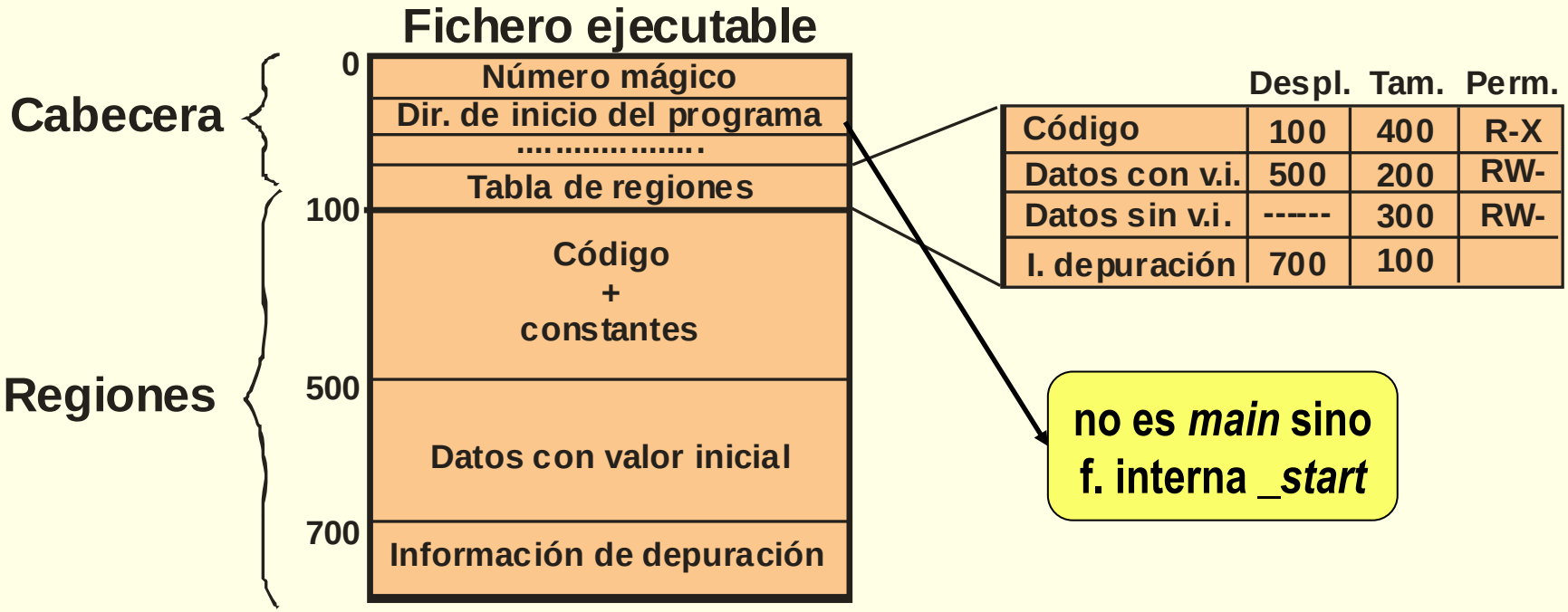
Contenido

- Aspectos generales de la gestión de memoria
- **Modelo de memoria de un proceso**
- Gestión de la memoria del sistema
- Memoria virtual

Ciclo de vida de un programa



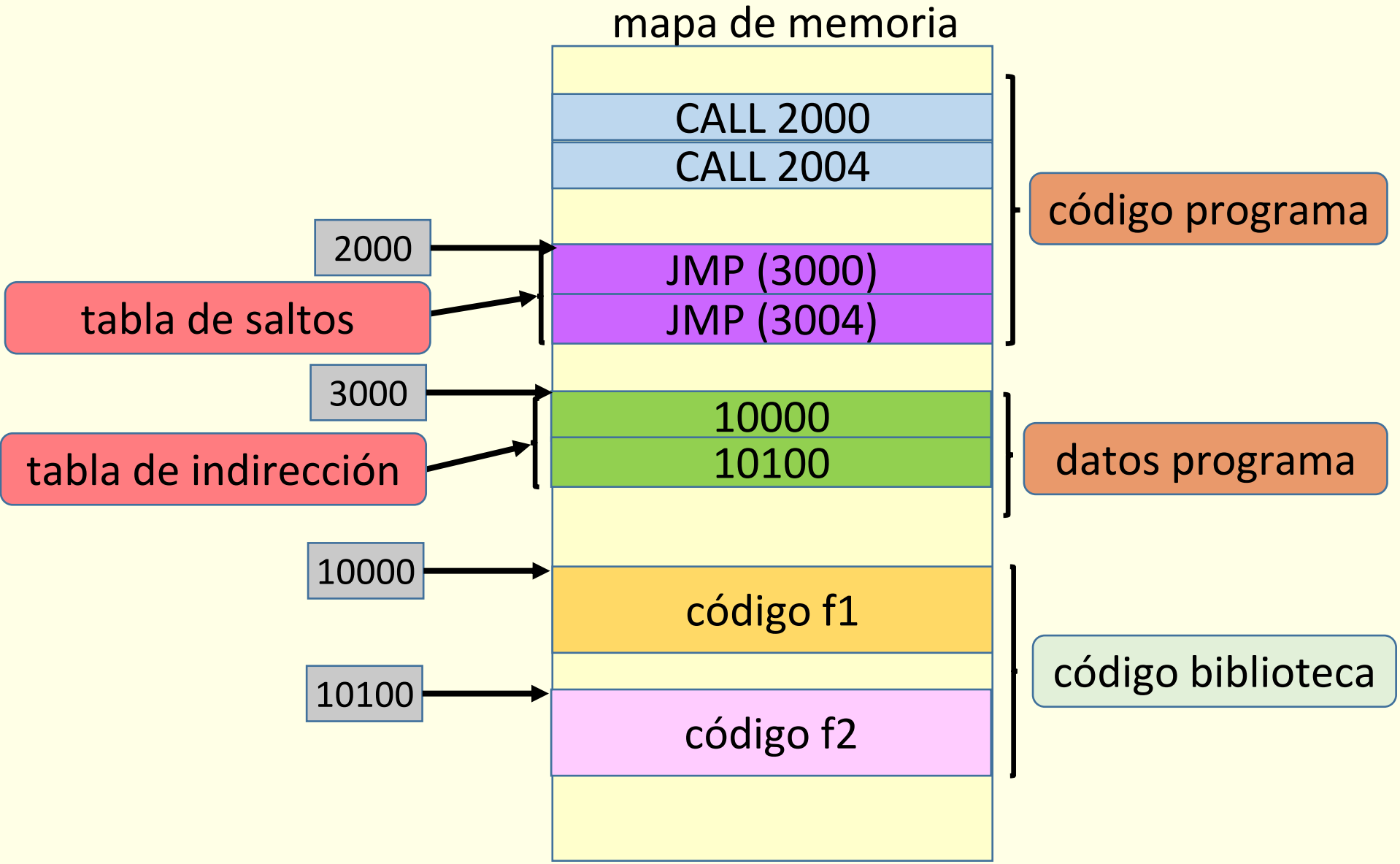
Formato del ejecutable



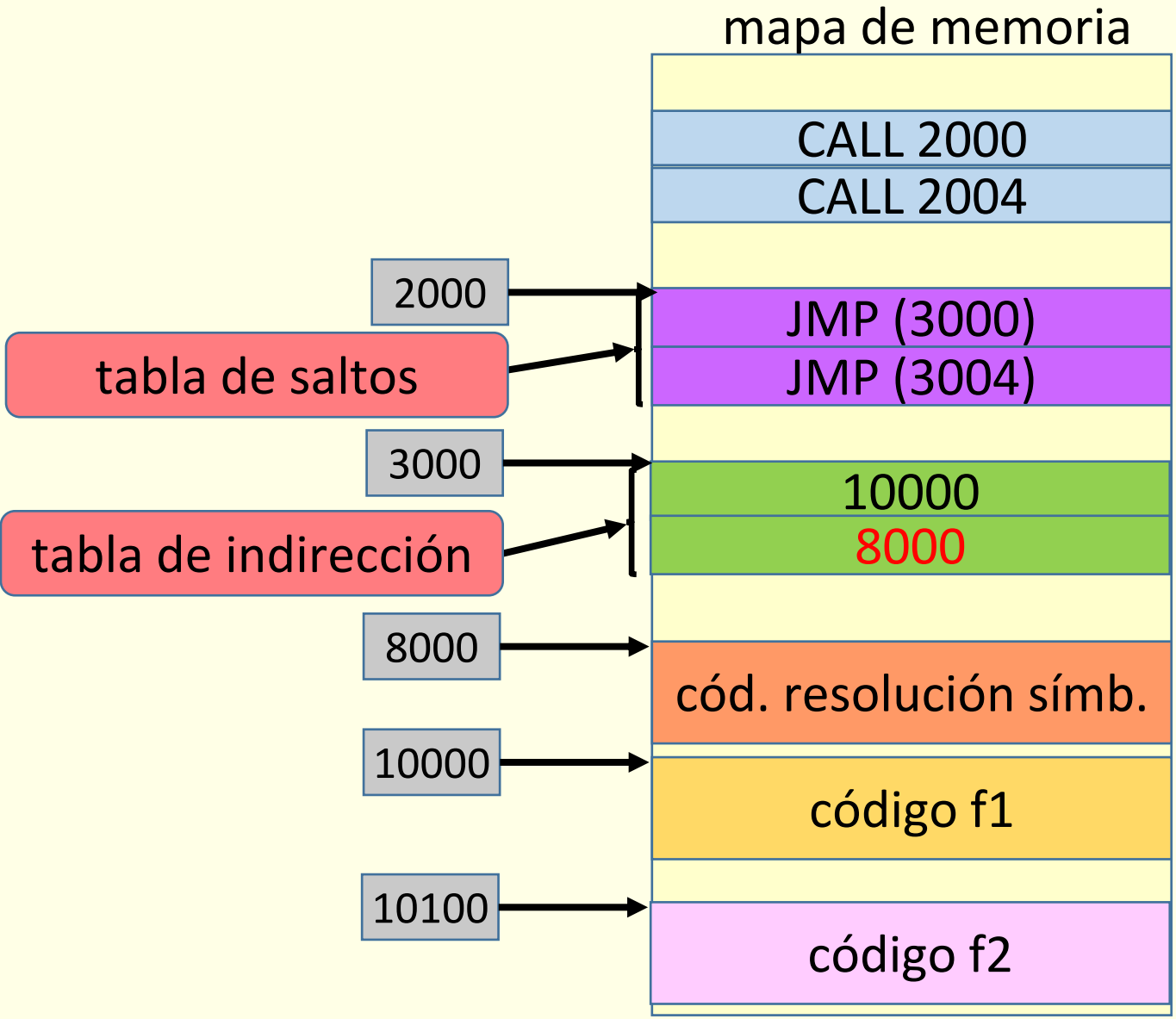
Linux: *Executable and Linkable Format (ELF)*

mandatos útiles Linux: *size, nm, readelf, objdump*

Resolución de referencias símbolos biblioteca



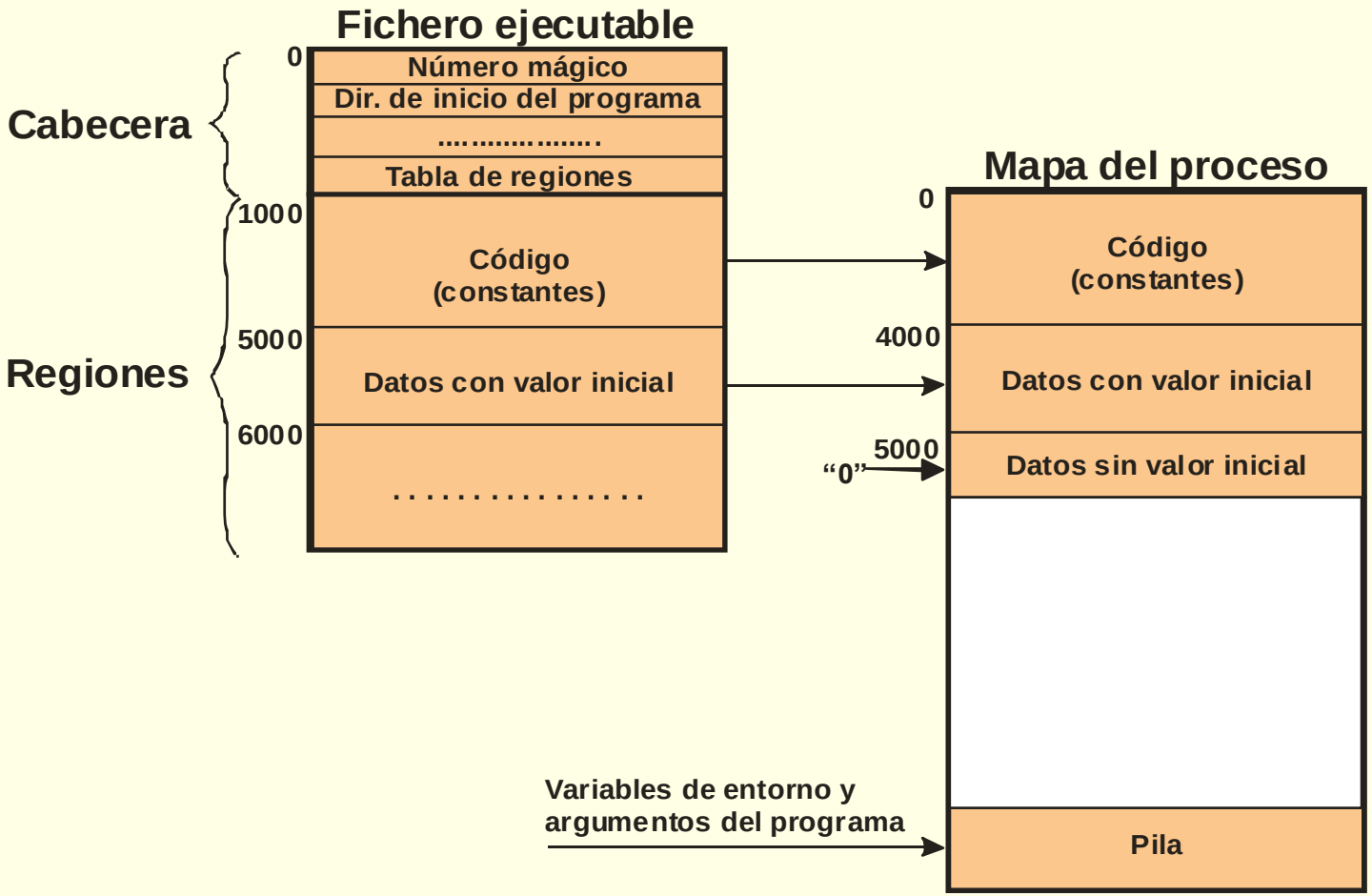
Resolución perezosa de referencias de símbolos



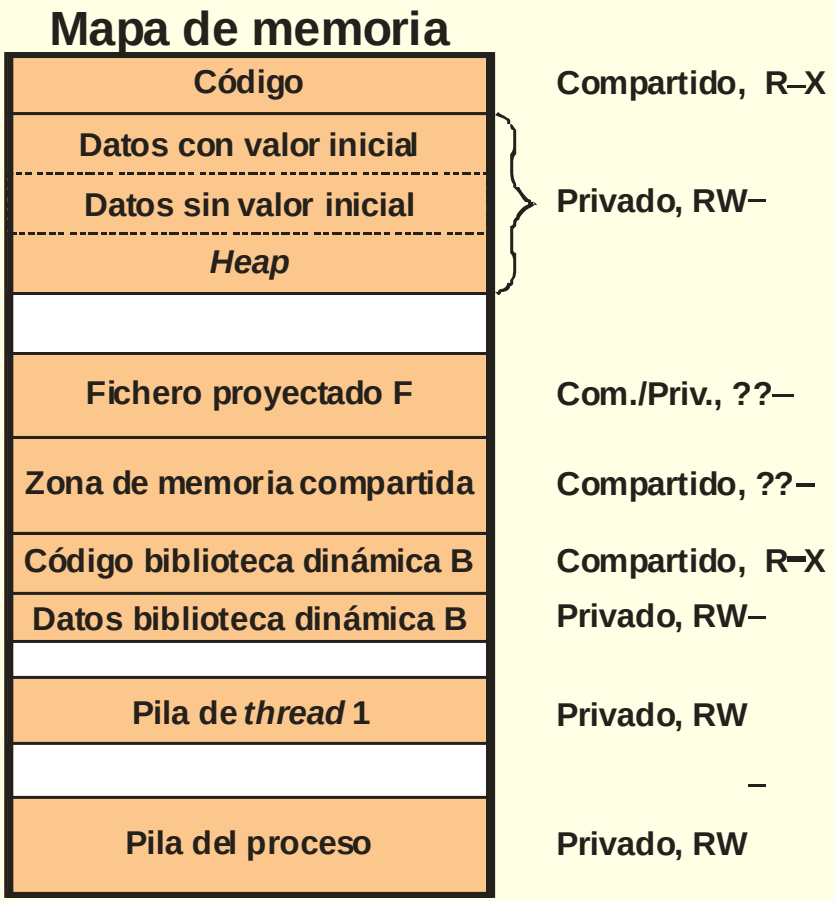
Mapa de memoria de un proceso

- Mapa de memoria o imagen del proceso: conjunto de regiones
- Región: área contigua en espacio lógico con mismas características
  - dirección de comienzo y tamaño inicial
  - protección: RWX
  - origen (soporte) de la región: En fichero o sin soporte (anónima)
    - Si no tiene soporte hay que rellenarla con ceros por seguridad
  - compartida o privada
    - Privada: Cada proceso tiene su propia copia de la región y
    - modificaciones no afectan al soporte original
  - tamaño fijo o variable (crecimiento ascendente o descendente)
  - ubicación en el mapa lógico prefijada o libre
    - Tradicionalmente, regiones provenientes de ejecutable posición fija
    - Uso de ASLR → regiones en ubicaciones aleatorias
      - PIE: *Position Independent Executables*

Crear mapa desde ejecutable



Mapa de memoria de un proceso hipotético




```
pmap $(pgrep firefox) | wc -l
1330
```

Regiones del mapa de memoria

Región	Soporte	Protección	Comp/Priv	Tam	Ubicación
Código	Fichero	RX	Compartida	Fijo	Prefijada
Dat. con v.i.	Fichero	RW	Privada	Fijo	Prefijada
Dat. sin v.i.	Sin soporte	RW	Privada	Fijo	Prefijada
Pilas	Sin soporte	RW	Privada	Var	Cualquiera
Heap	Sin soporte	RW	Privada	Var	Cualquiera
F. Proyect.	Fichero	por usuario	Comp./Priv.	Fijo	Cualquiera
M. Comp.	Sin soporte	por usuario	Compartida	Fijo	Cualquiera
Bib.dinám.	Regiones para código y datos (con y sin valor inicial)				

Si PIE: ubicación de todas → cualquiera

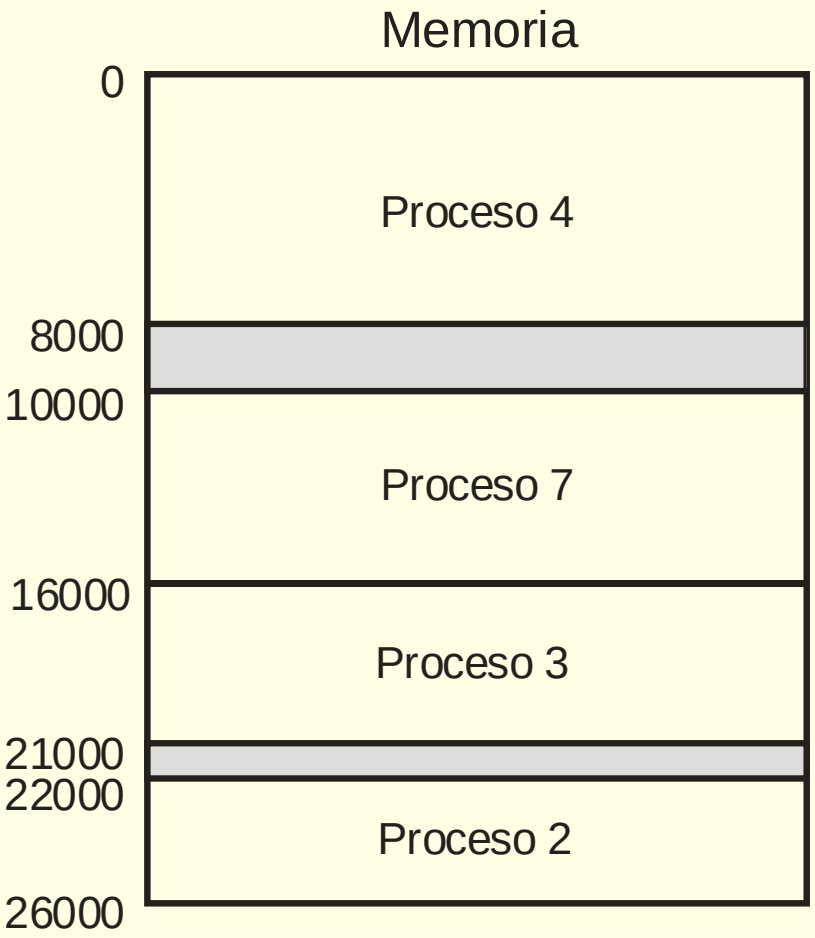
Contenido

- Aspectos generales de la gestión de memoria
- Modelo de memoria de un proceso
- **Gestión de la memoria del sistema**
- Memoria virtual

Esquemas de gestión de la memoria del sistema

- Nivel sistema: ¿Cómo se reparte memoria entre mapas procesos?
- Se requiere función de traducción para cada proceso que:
  - Cree un espacio lógico de 0 hasta un cierto valor muy grande
  - Proporcione aislamiento entre los procesos
  - Pero permitiendo compartir memoria de forma controlada
  - Gestione los permisos de acceso a cada posición de memoria
    - Generando excepción si se producen accesos no permitidos
  - Evite que huecos en espacio lógico consuman espacio
    - Generando excepción si se producen accesos a direc. en huecos
  - Con buen aprovechamiento de la memoria (soporte m.virtual)
- Evolución de esquemas de gestión de memoria:
  - Asignación contigua (obsoleto en SO propósito general)
  - Segmentación (obsoleto en SO propósito general)
  - Paginación: soporte mínimo esperado por SO propósito general
  - Segmentación paginada: Más sofisticado que paginación
    - Por portabilidad mayoría de SSOO desactivan la segmentación

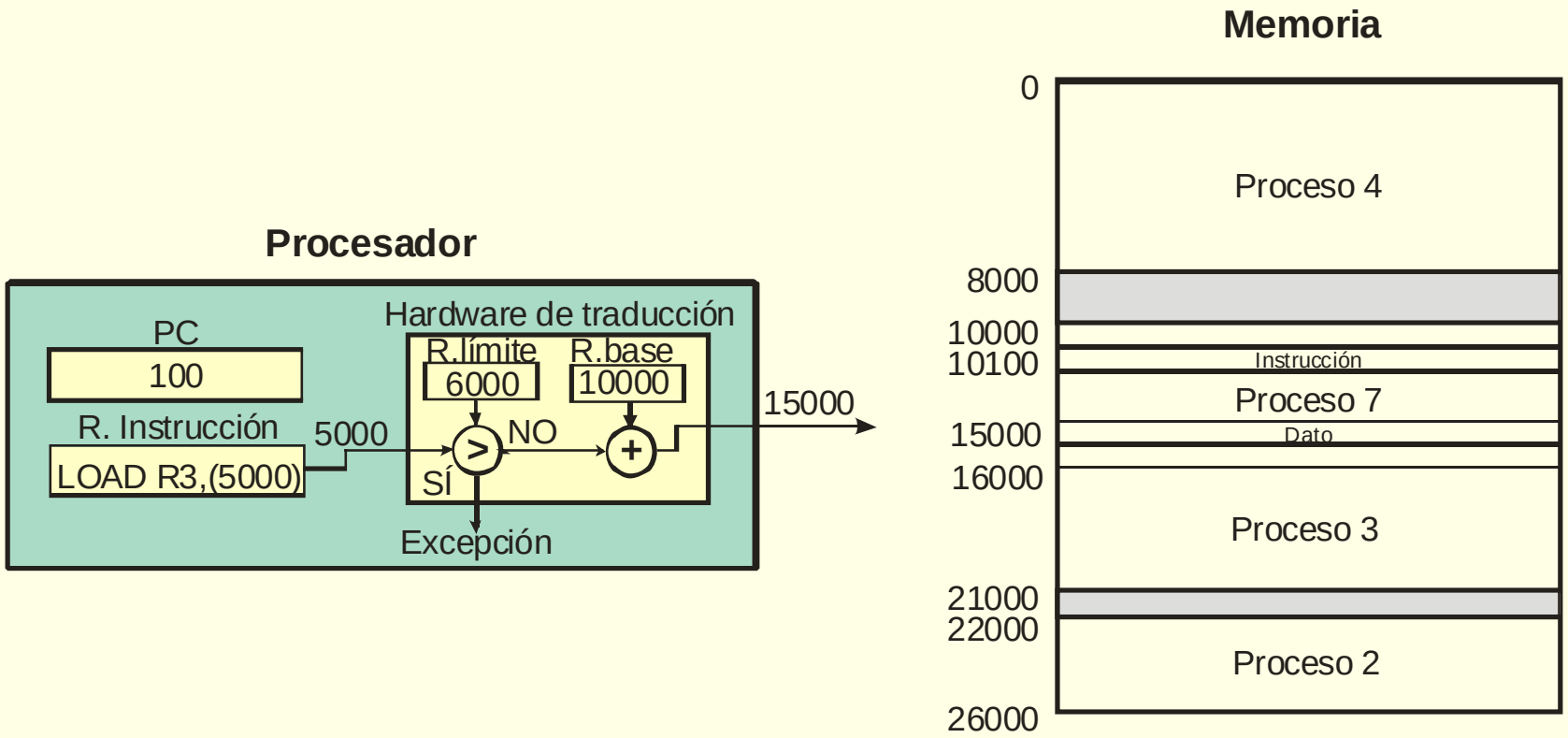
Asignación contigua



- Deficiencias de asignación contigua
  - No soporte regiones, no compartir, no base de m. virtual

Registros base y límite

- Protección + reubicación de procesos por hardware



Creación de espacio lógico independiente

BCP de P4

base	límite
0	8000

BCP de P7

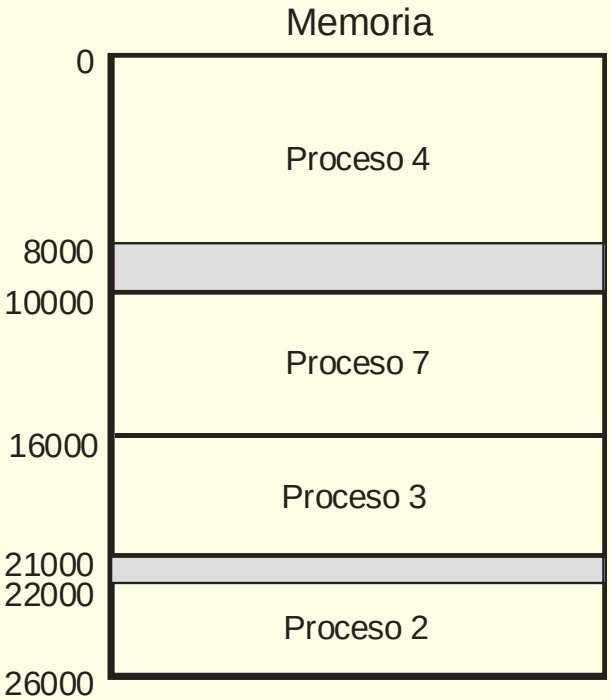
base	límite
10000	6000

BCP de P3

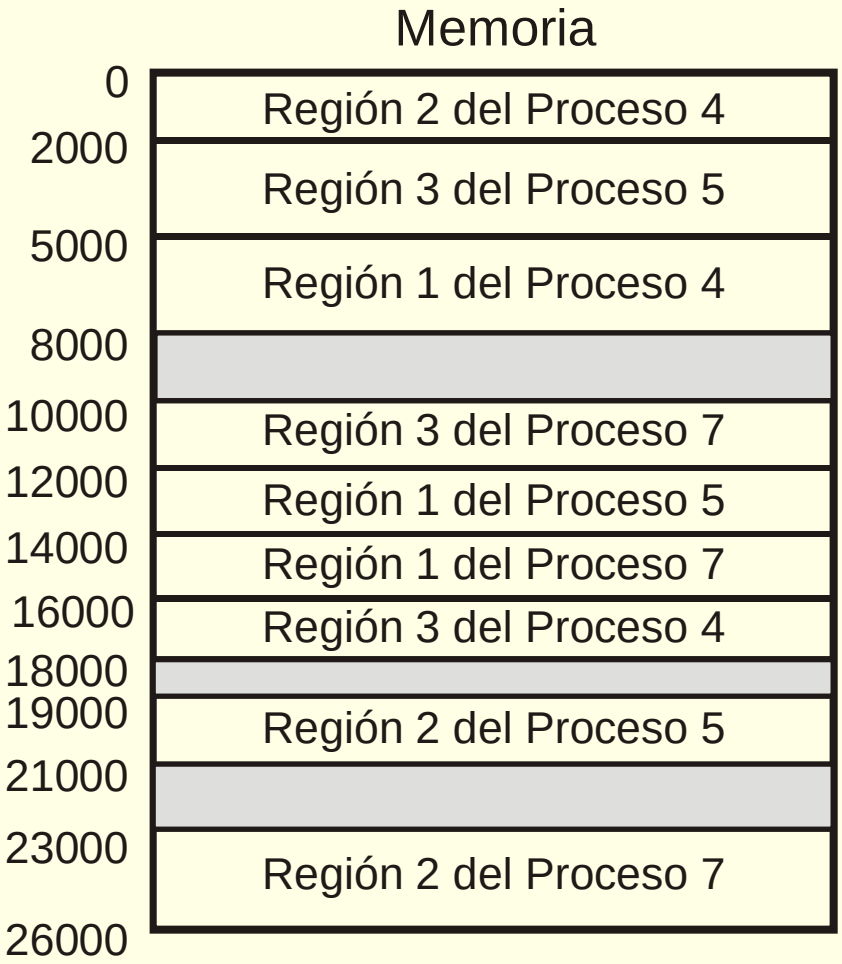
base	límite
16000	5000

BCP de P2

base	límite
22000	4000



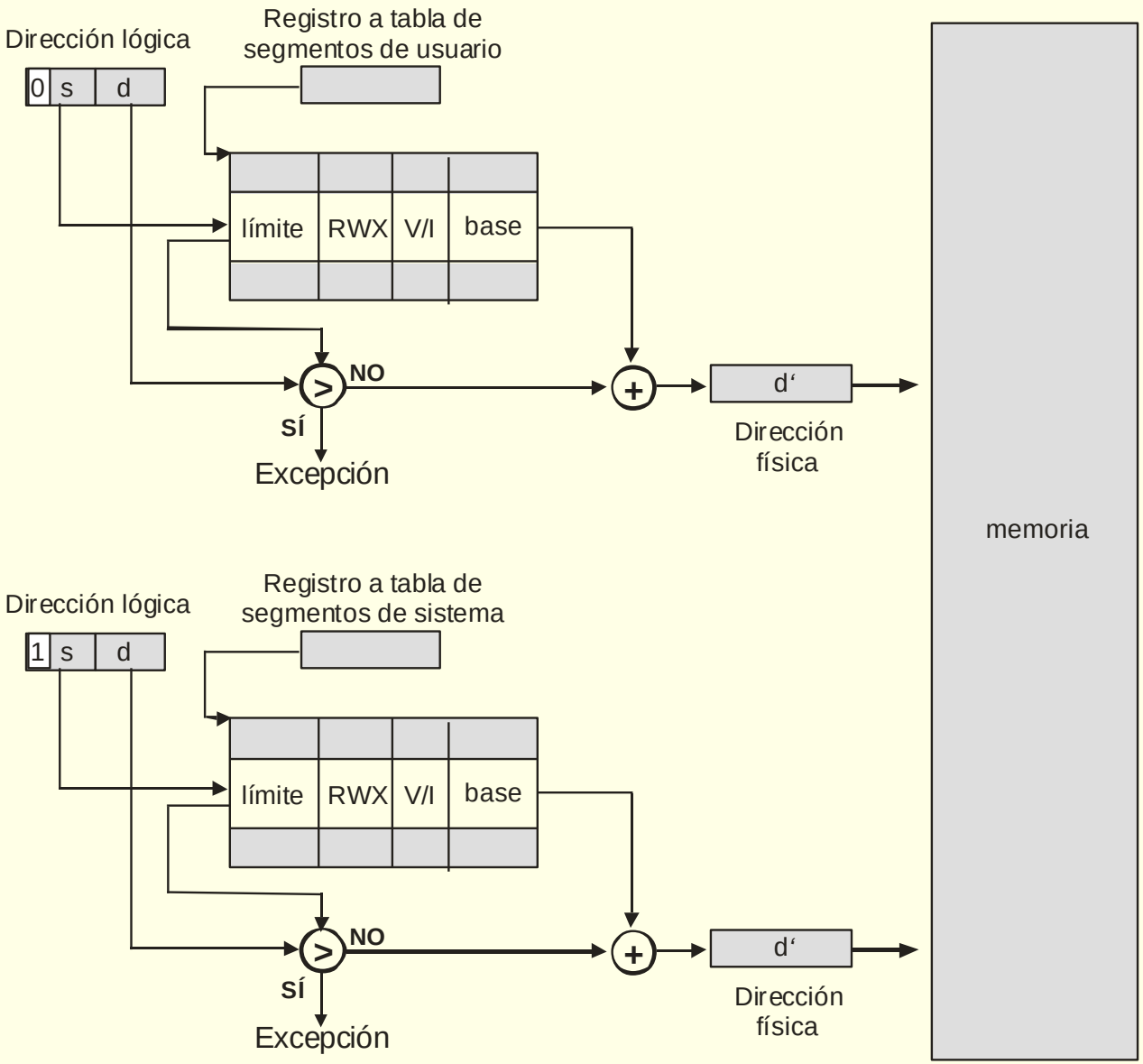
Segmentación



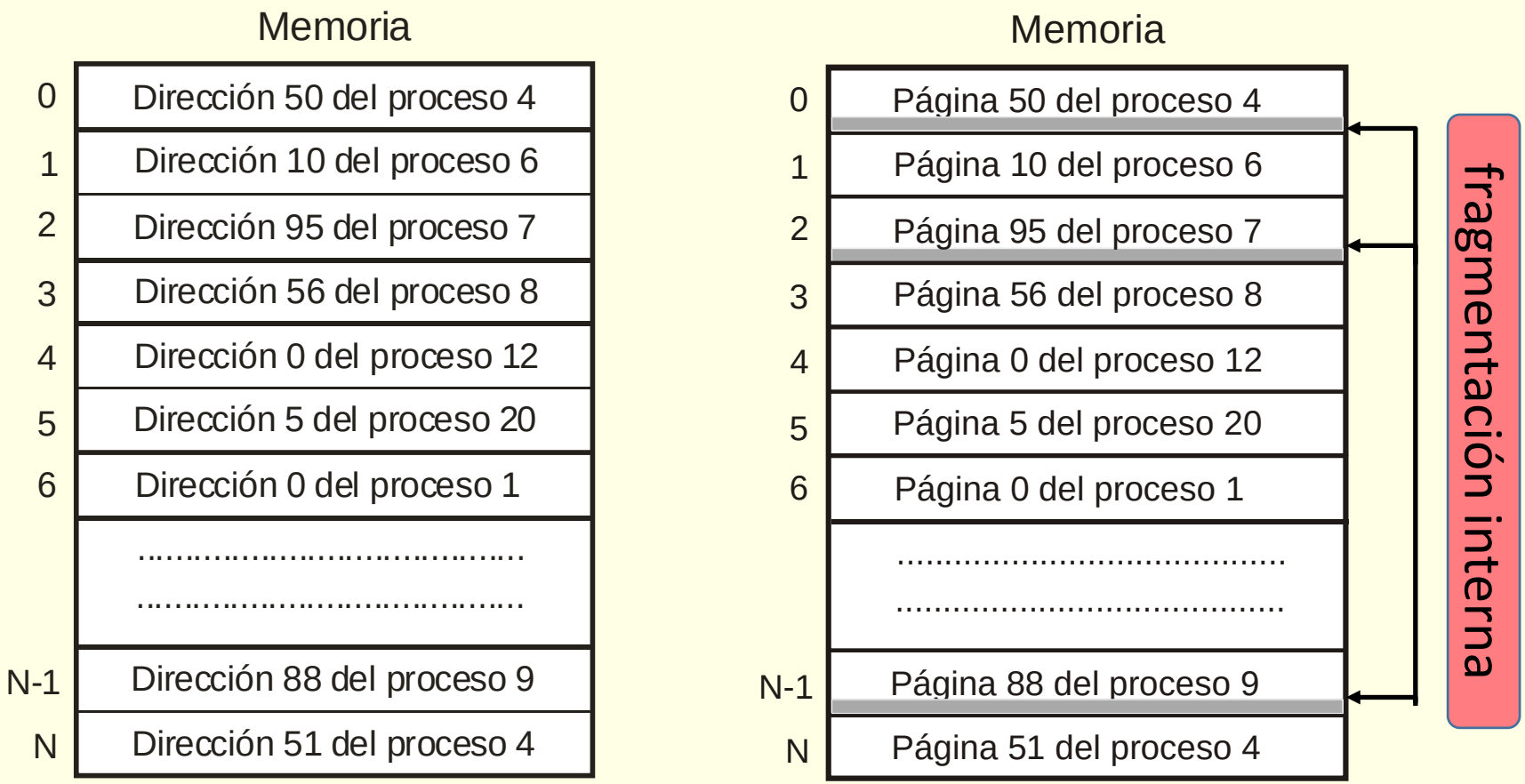
Segmentación

- Generalización de registro base y límite: 1 pareja/segmento
- Dirección lógica: núm. de segmento + dirección en segmento
- MMU usa una tabla de segmentos (TS)
- Entrada de TS contiene (entre otros):
  - r. base y límite del segmento; protección: RWX + bit de validez
- Dir. lógicas de usuario y de sistema (p.e. empiezan por 0 o 1)
- SO mantiene TSU por proceso (t. de regiones)
  - En c. contexto notifica a MMU qué TSU debe usar
- SO mantiene una única TSS que no cambia
  - Procesos comparten mapa del sistema operativo
- Soporte directo de regiones
- Compartir: 2 entradas iguales en 2 TSU
- Soporte memoria virtual:
  - No adecuado por tamaño variable de segmentos

Esquema de traducción con segmentación



Paginación: Fundamento

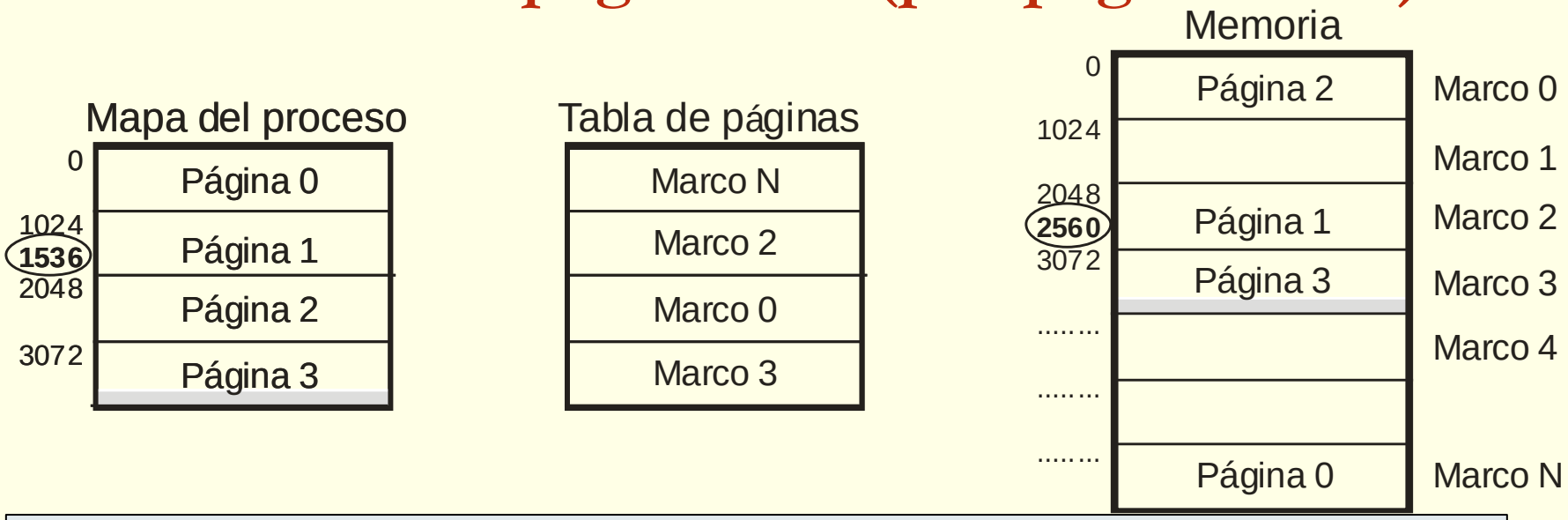


- Solución no factible: Tabla por proceso que haga corresponder
  - Cada dirección lógica con dirección física asociada
  - ¡Ocuparía toda la memoria!
- Cambio de escala: de 1 dirección a un bloque → **página**

Paginación. Aspectos hardware

- Mapa de memoria del proceso dividido en páginas
- Memoria principal dividida en marcos (tam. marco=tam. página)
- Tabla de páginas (TP): Asocia página y marco que la contiene
- Tabla de páginas única (bit *S*) vs. 2 tablas de páginas separadas
- Contenido entrada TP
  - Número de marco asociado
  - Información de protección: RWX
  - Bit de página válida/inválida
  - Bit de página accedida (*Ref*) y de página modificada (*Mod*)
  - Bit de desactivación de caché (para direcciones de E/S)
  - Entrada de sistema (*S*): Presente en MMU con TP única
  - Indicador de página global (*G*): accesible por todos
  - Indicador de superpágina

Traducción con paginación (p.e. páginas 1KB)



Dfís = TablaPág[DLóg/TamPág] * TamPág + DLóg%TamPág

Dfís = TablaPág[1536/1024] * 1024 + 1536%1024

Dfís = TablaPág[1] * 1024 + 512 → Dfís = 2*1024+512 → Dfís = 2560

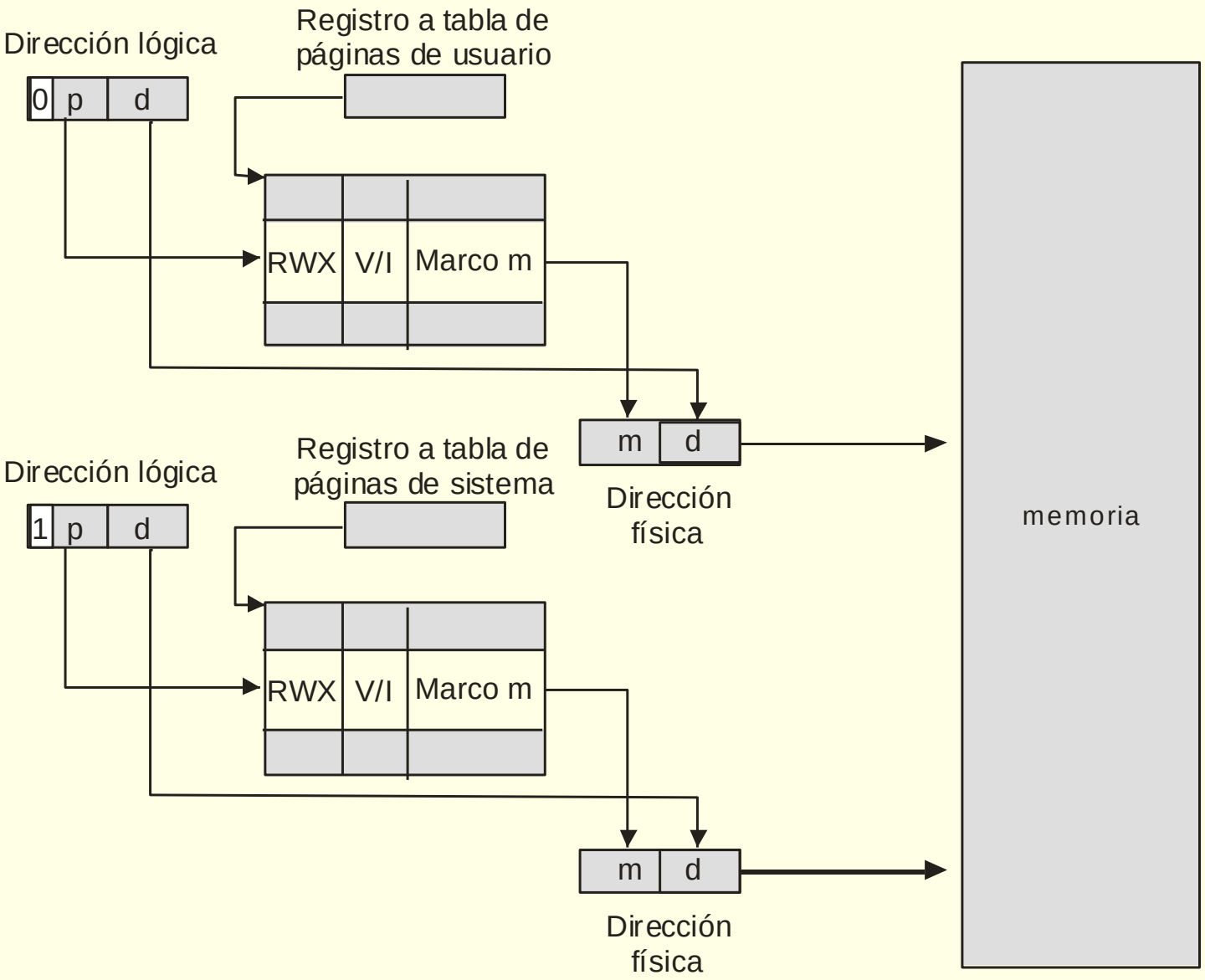
¡Por cada acceso a memoria multiplicaciones y divisiones! ¿Eficiencia?

Tamaño de tabla de páginas es potencia de 2: * / % son inmediatas

1536/1024 1536%1024

1536 → 00000000000000000000011000000000

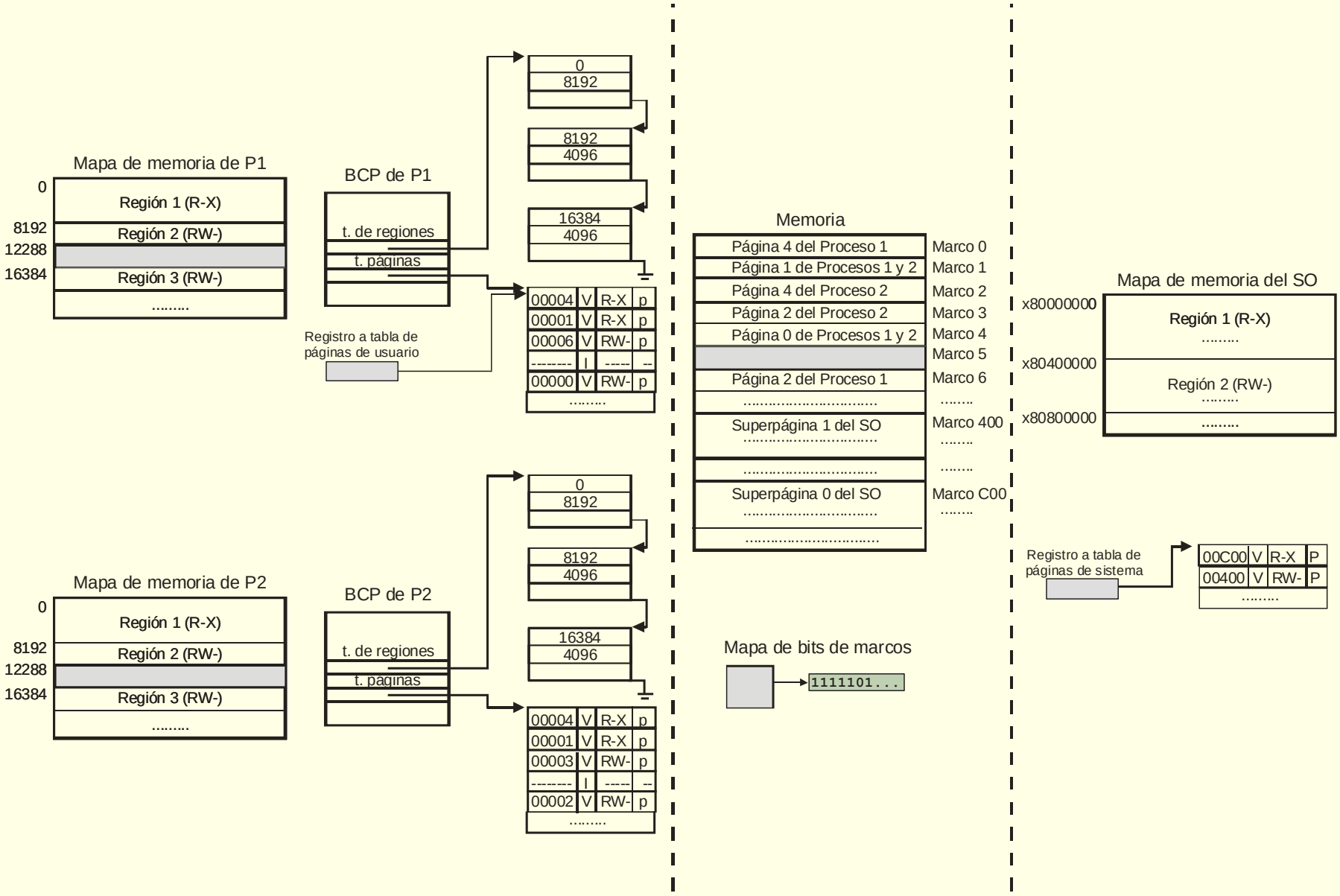
Esquema de traducción con TP usuario y sistema



Creación de espacio lógico independiente

- SO mantiene una TP por cada proceso
  - En c. contexto notifica a MMU cuál debe usar
- Con TP separadas, SO mantiene una única TP para propio SO
- Proceso modo sistema acceso directo a su mapa y al de SO
  - Procesos comparten mapa del sistema operativo
  - SO interpreta directamente direcciones de usuario de p. actual
- No soporte directo de regiones pero fácil conseguirlo
  - Región ocupa número entero de páginas
  - SO mantiene tabla de regiones por cada proceso
  - Protección de región: uso de bits de protección de sus páginas
  - No reserva espacio para huecos: bit de validez desactivado
- Compartir región: Entradas corresponden a mismo marco
- Soporte memoria virtual: Unidad de transferencia de tamaño fijo

Visión global de la paginación



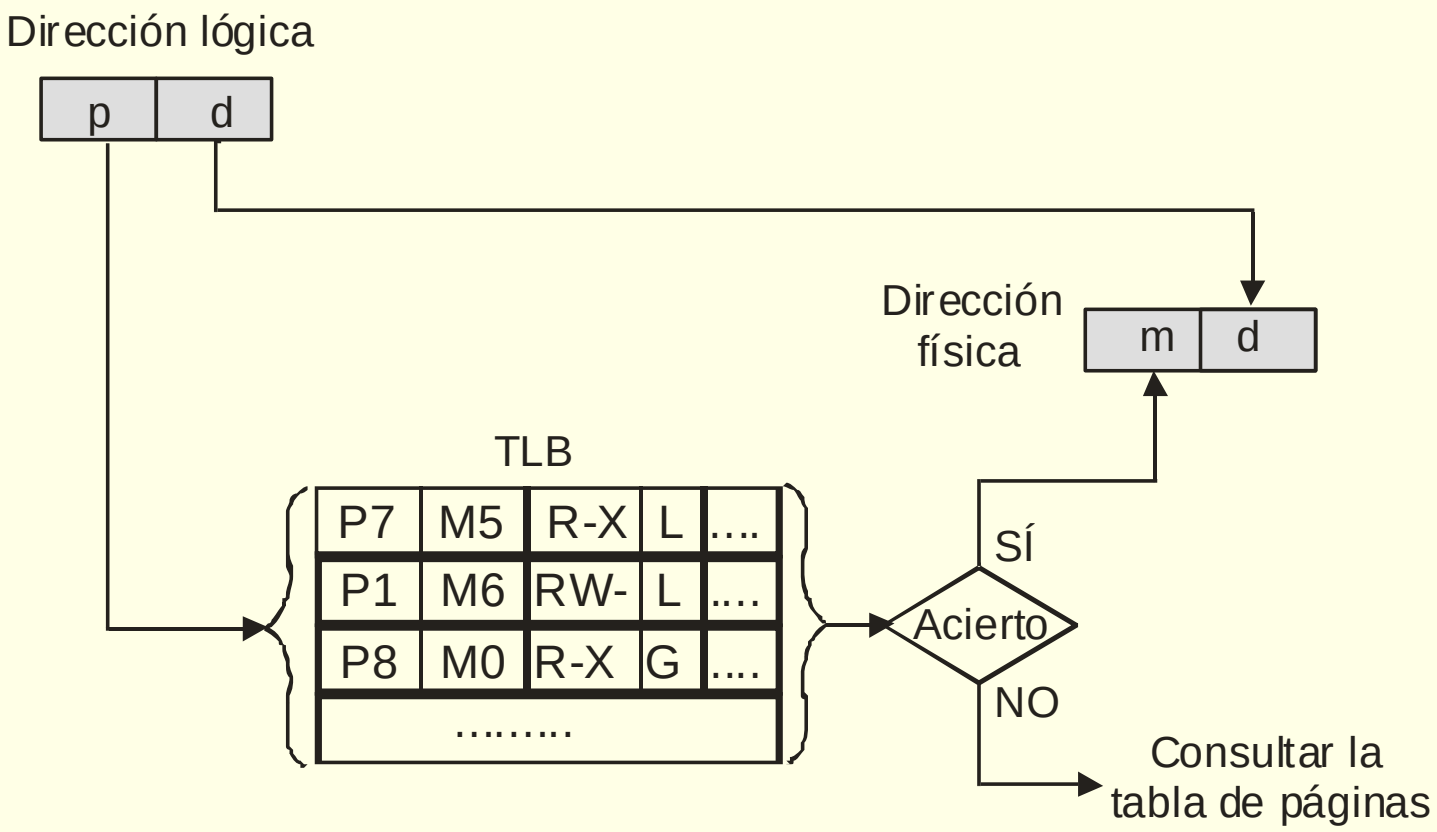
Implementación de TP

- TP se mantiene normalmente en memoria principal
- 2 problemas: eficiencia y gasto de almacenamiento
- Eficiencia:
  - Cada acceso lógico requiere 2 accesos a memoria principal
  - Solución: *caché* de traducciones → TLB
- Gasto de almacenamiento: Tablas muy grandes
  - Ejemplo: páginas 4K, dir. lógica 32 bits y 4 bytes/entrada
    - Tamaño TP: 2^{20} *4 = 4MB/proceso
  - Solución: tablas multinivel y tablas invertidas

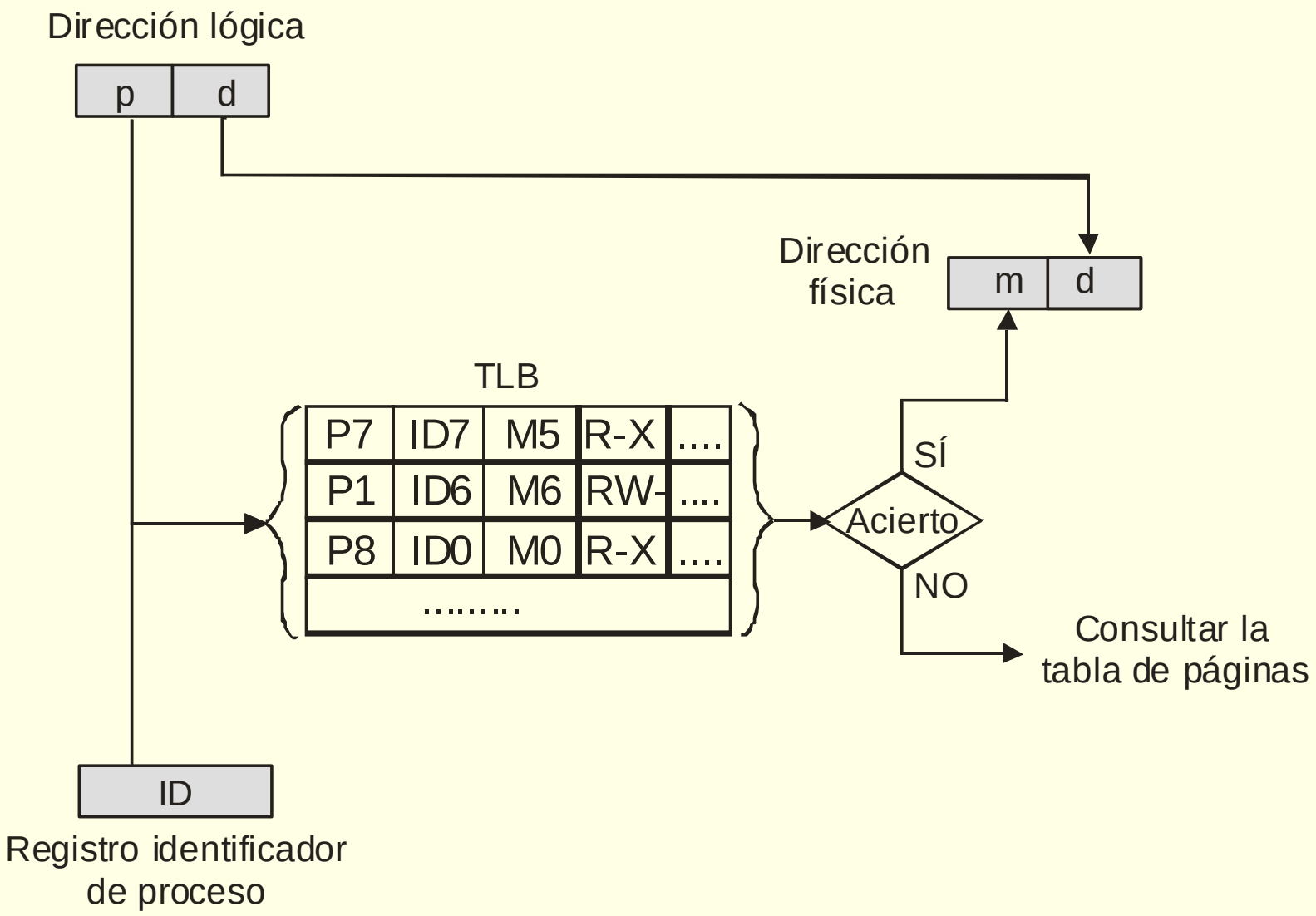
Translation Look-aside Buffer (TLB)

- Memoria asociativa con info. sobre últimas páginas accedidas
- Entradas en TLB no incluyen información sobre proceso
  - **Invalida** TLB en c. contexto (no entradas de sistema)
- Entradas en TLB incluyen información sobre proceso
  - Incluyen *Address Space Identifiers* (ASID)
  - Registro UCP mantiene ASID de p. actual:
    - No invalidar excepto si se reutilizan
- Gestionada por MMU: Si fallo usa la TP en memoria
  - MMU activa *Mod* y *Ref* en TP
- Pero no totalmente transparente al SO
  - Además de invalidar contenido en c. contexto si no ASID
  - También invalidar entrada TLB si cambio en la entrada T. pág.
    - Si se marca como inválida, se modifica bit ref o mod o permisos…

TLB sin información de proceso (sin ASID)



TLB con información de proceso (con ASID)



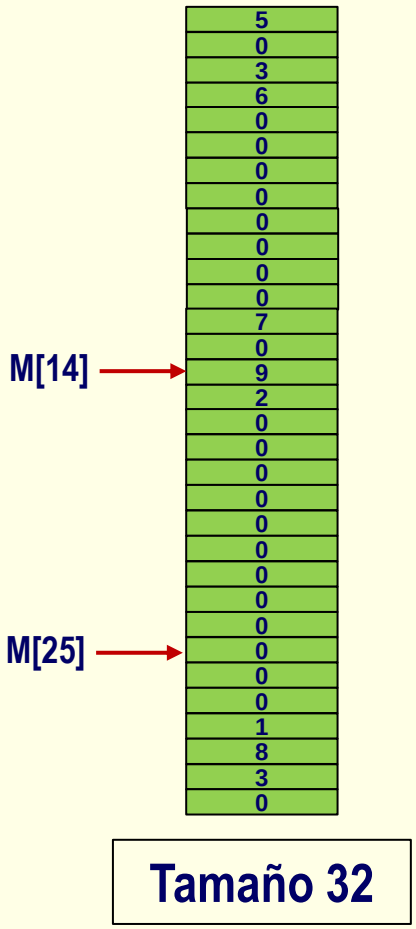
TLB gestionada por software

- Alternativa: traspasar al SO parte del trabajo de traducción
- MMU no usa tablas de páginas, solo consulta TLB
- SO mantiene TPs que son independientes del HW
- Fallo en TLB → Activa SO
- SO se encarga de:
  - Buscar "a mano" en TP la traducción
  - Rellenar TLB con la traducción
  - Propagar bits *Ref* y *Mod* a TP
- Flexibilidad en diseño de SO pero menor eficiencia

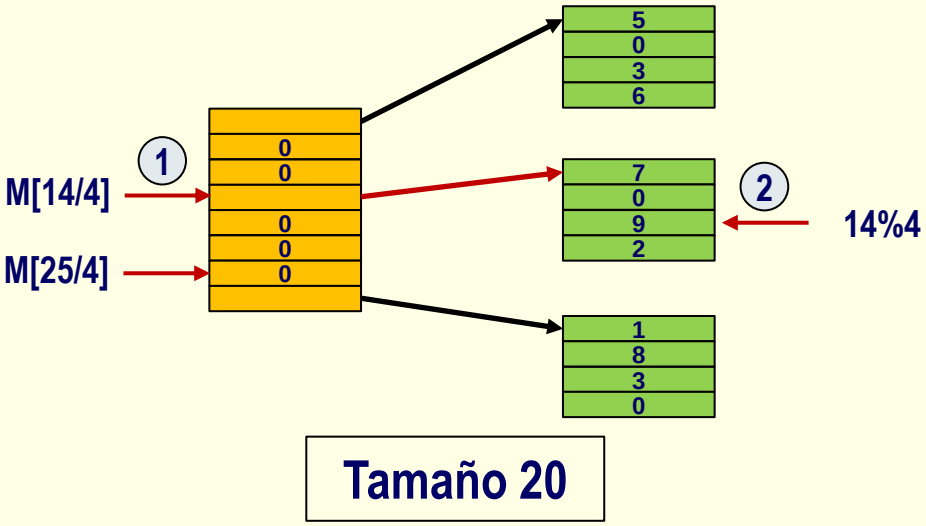
Tablas de páginas multinivel

- Fundamento
  - Tablas de páginas muy grandes con muchas entradas nulas
  - Fragmentar tabla y acceder mediante tabla maestra
- Tablas de páginas organizadas en 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
- Si todas las entradas de una TP son inválidas
  - No se almacena esa TP e inválida entrada de TP superior
- Procesadores actuales usan típicamente 4 niveles

Compactando una matriz dispersa



Pro: Menor ocupación
Con: 2 accesos

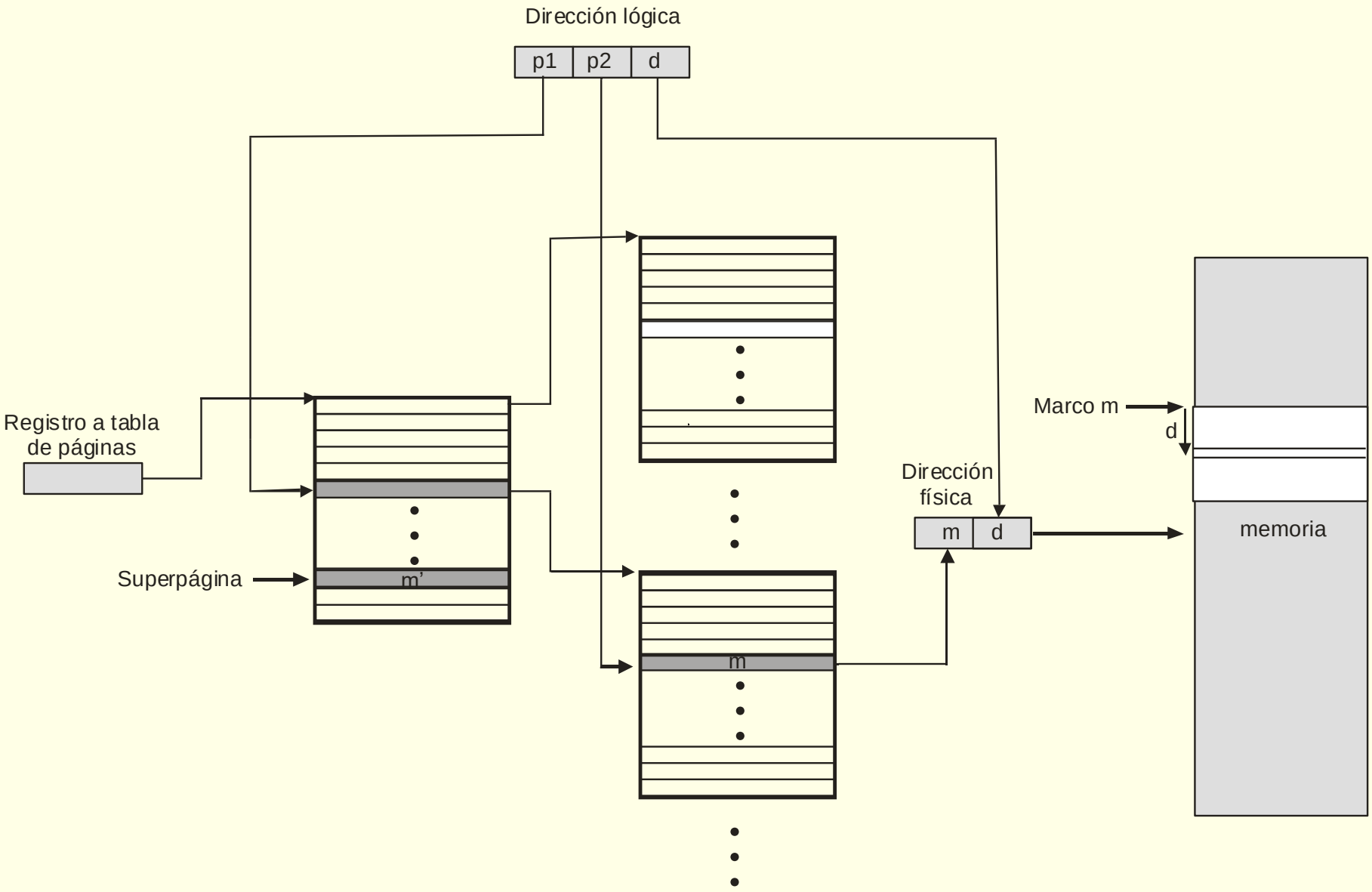


/ y % inmediatas si potencia de 2

14/4 14%4

14 → 0 1 1 1 0

Esquema de traducción con 2 niveles



Ventajas de tablas de páginas multinivel

- Si proceso usa una parte pequeña de su espacio lógico
  - Ahorro en espacio para almacenar TPs
- Ejemplo: Proceso que usa 12MB superiores y 4MB inferiores
  - 2 niveles, pág. 4K, dir. lógica 32 bits (10 bits/nivel), 4B/entrada
    - Tamaño: 1 TP N_1 + 4 TP N_2= 5 * 4KB = 20KB (frente a 4MB)
- Ventajas adicionales:
  - Permite compartir TPs intermedias
  - solo se requiere que esté en memoria la TP de nivel superior
- Facilita implementación de superpáginas
  - Entrada de primer nivel de superpágina apunta a m. física

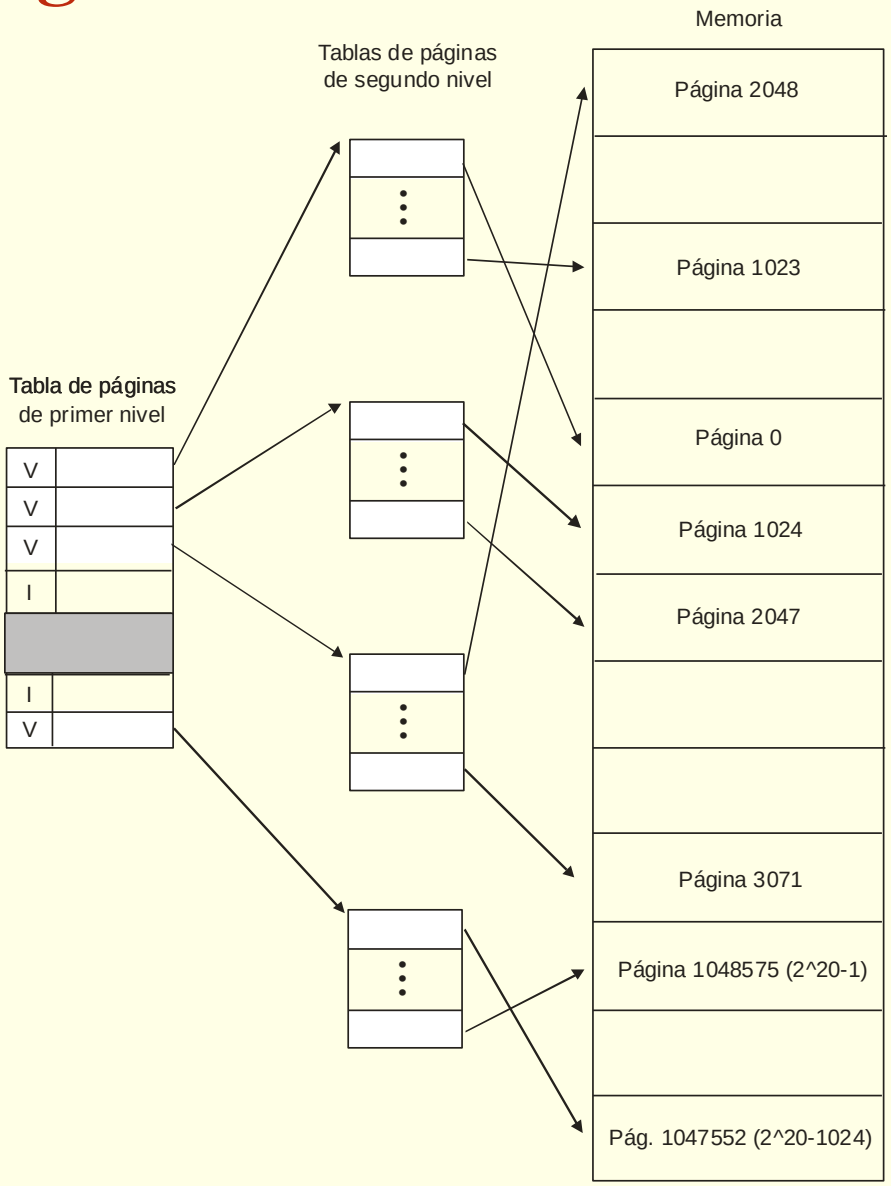
Ventajas de tablas de páginas multinivel

Si en ejemplo se usan
2 niveles con 10 bits/nivel

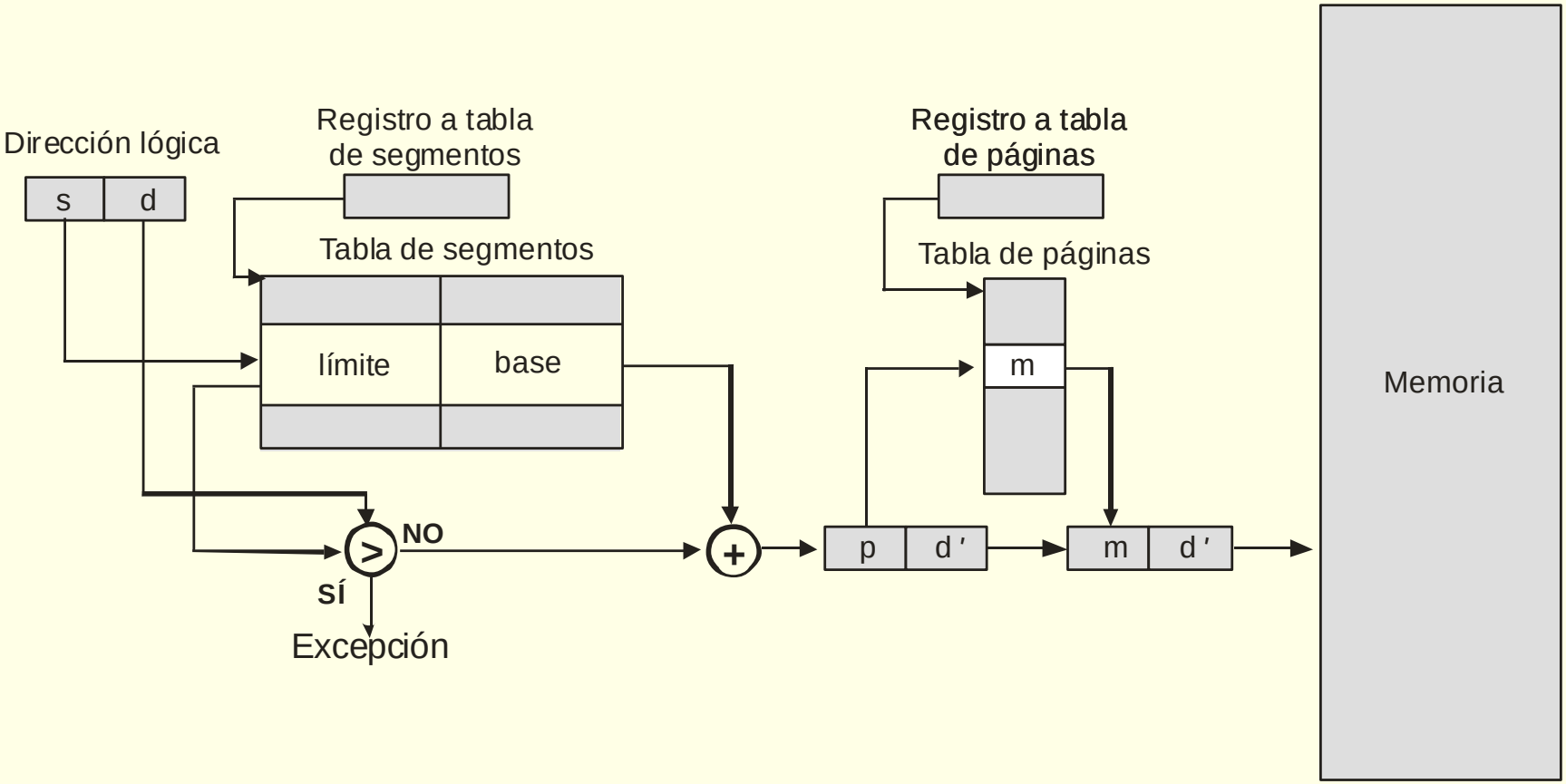
Tamaño total en t. de pág.:

1 TP N1 + 4 TP N2= 5 * 4KB
20KB (frente a 4MB)

si 1K procesos iguales→ 20MB
(frente a 4G)



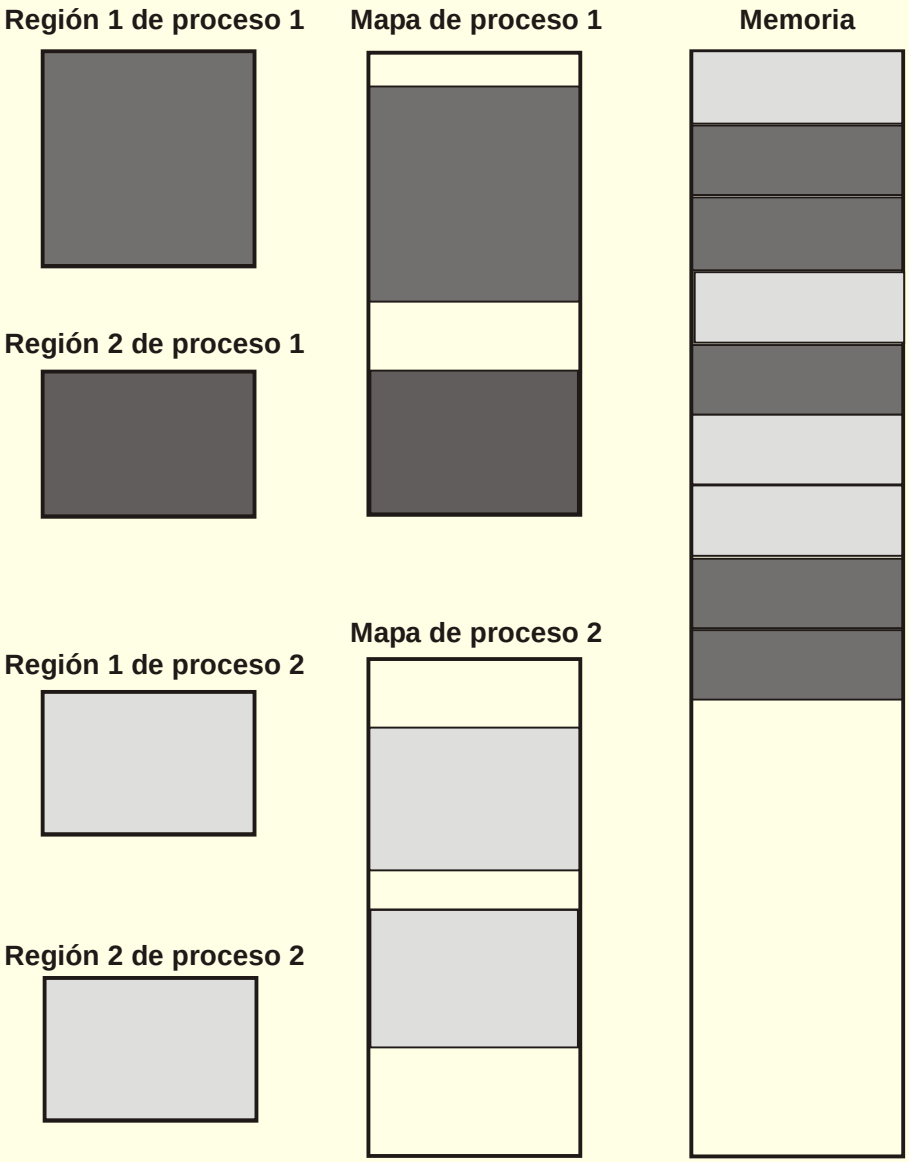
Segmentación paginada: esquema de traducción



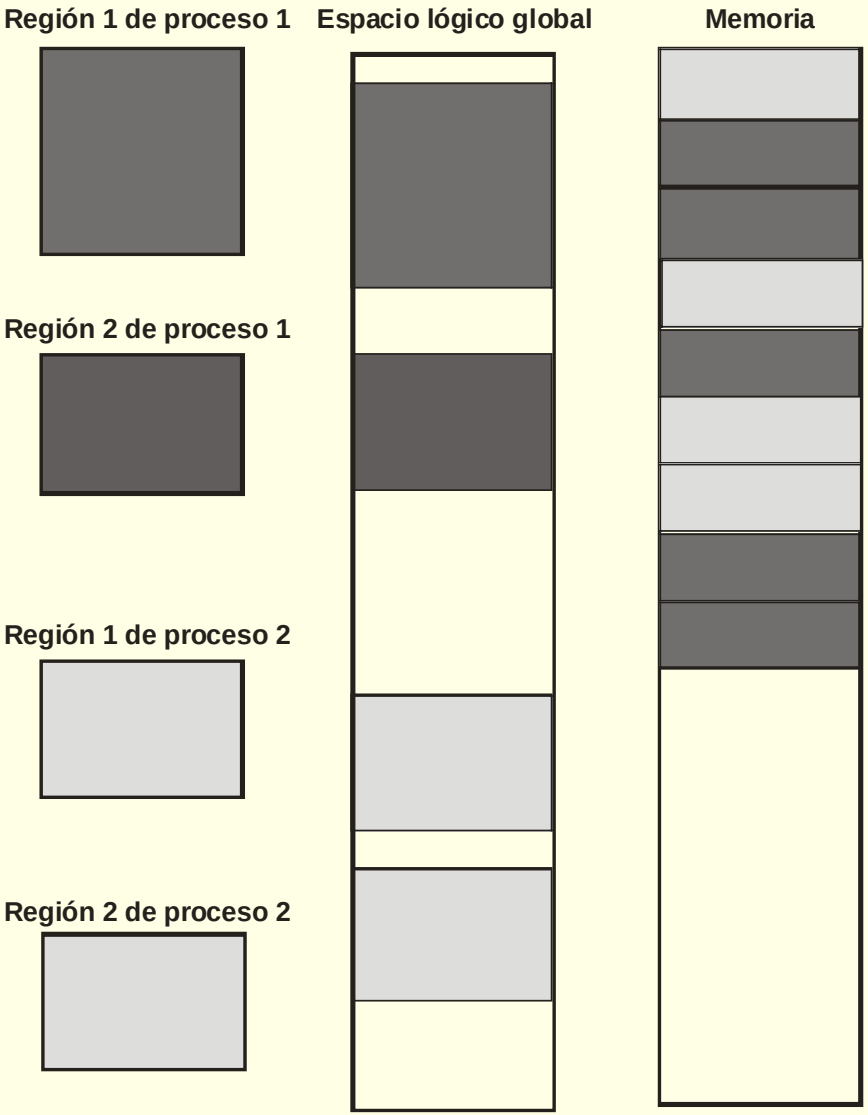
"Lo mejor de los dos mundos"

Pero la mayoría de los SSOO desactivan la paginación

Segmentación paginada local



Segmentación paginada global



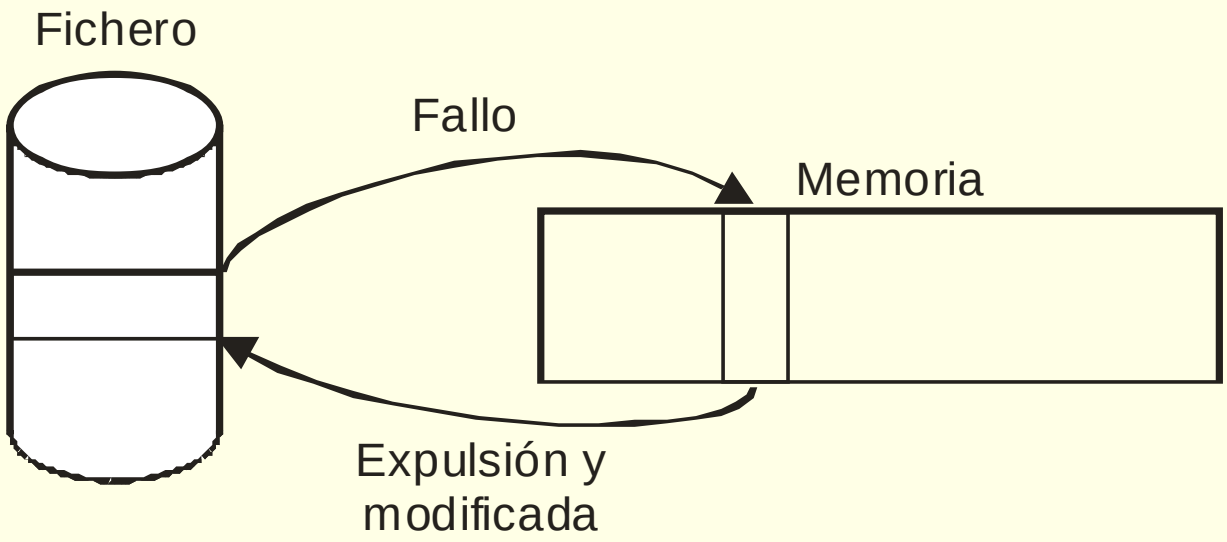
Contenido

- Aspectos generales de la gestión de memoria
- Modelo de memoria de un proceso
- Esquemas de gestión de la memoria del sistema
- **Memoria virtual**

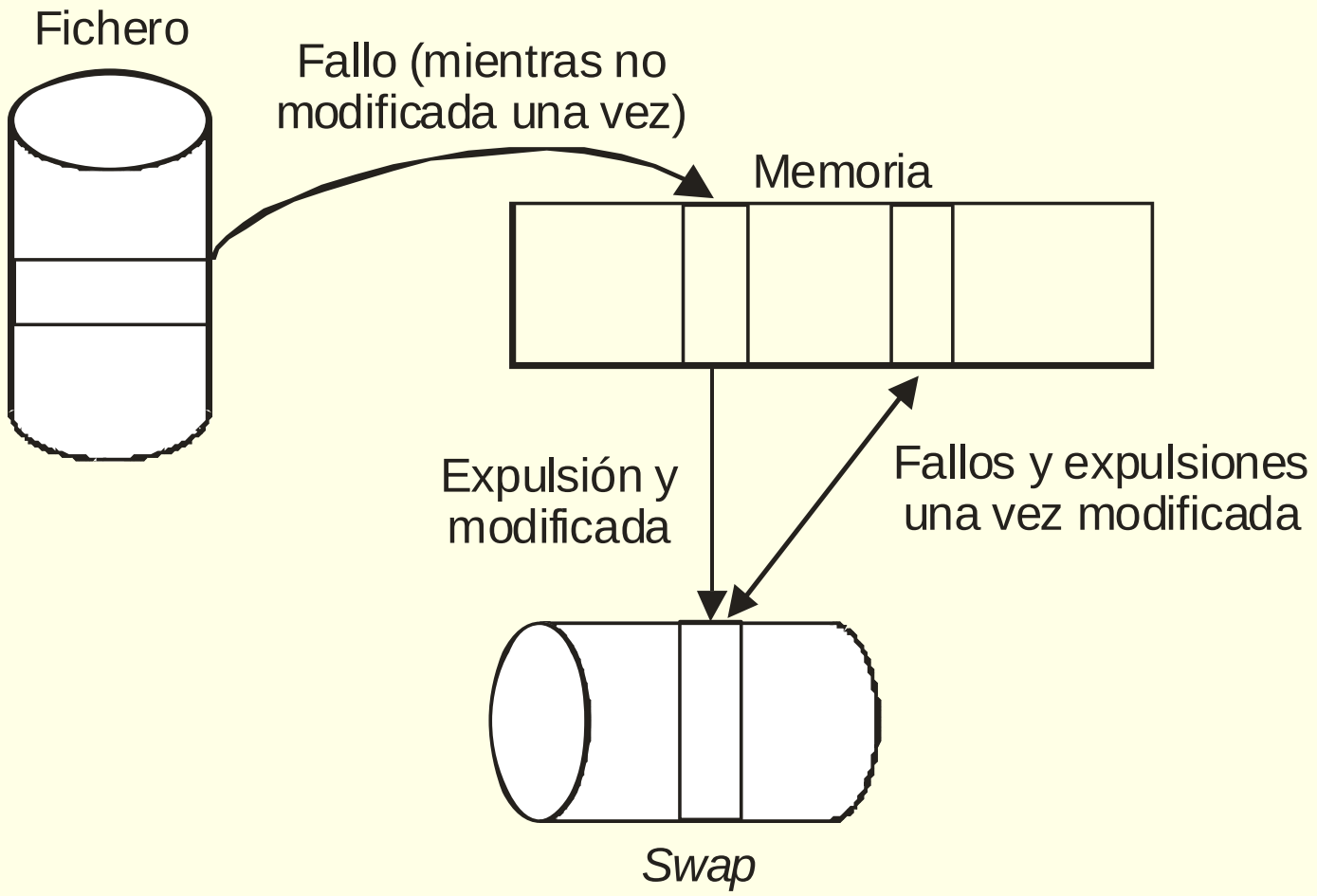
Fundamento de la memoria virtual

- M. virtual: SO gestiona niveles de m. principal y m. secundaria
  - Sube por demanda; Baja por expulsión
- Aplicable por proximidad de referencias
  - Procesos solo usan parte de su mapa en intervalo de tiempo
  - Parte usada (*cjto de trabajo*) en m. principal (*cjto residente*)
- Beneficios:
  - Aumenta el grado de multiprogramación
  - Permite ejecución de programas que no quepan en mem. ppal
- No adecuada para sistemas de tiempo real
- Basada en paginación: Uso del bit de validez
  - Página no residente se marca como no válida
    - ¿Cómo distinguir si página no residente o realmente inválida?
  - En acceso: Excepción de fallo de página

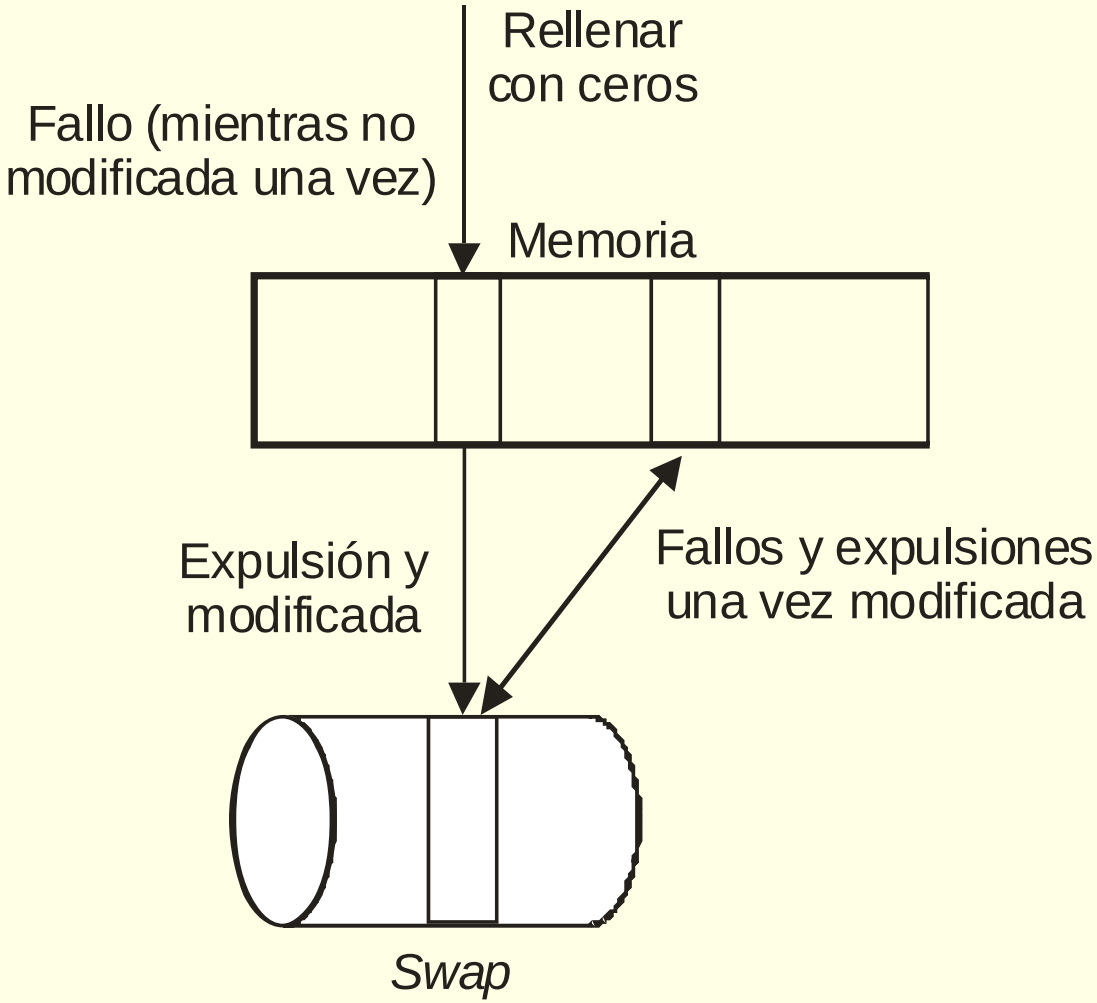
Ciclo de vida de página compartida y en fichero



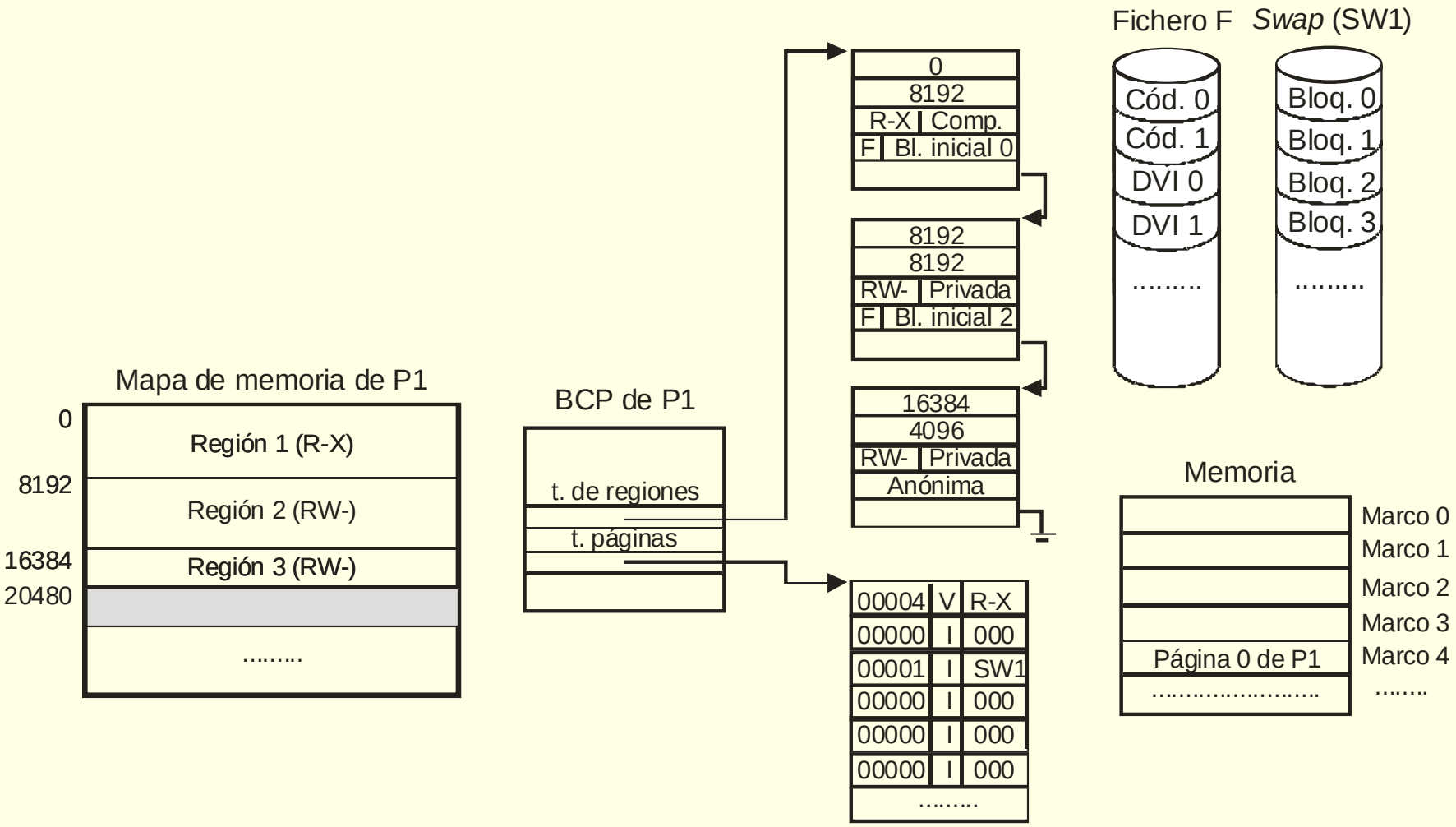
Ciclo de vida de página privada y en fichero



Ciclo de vida de página anónima



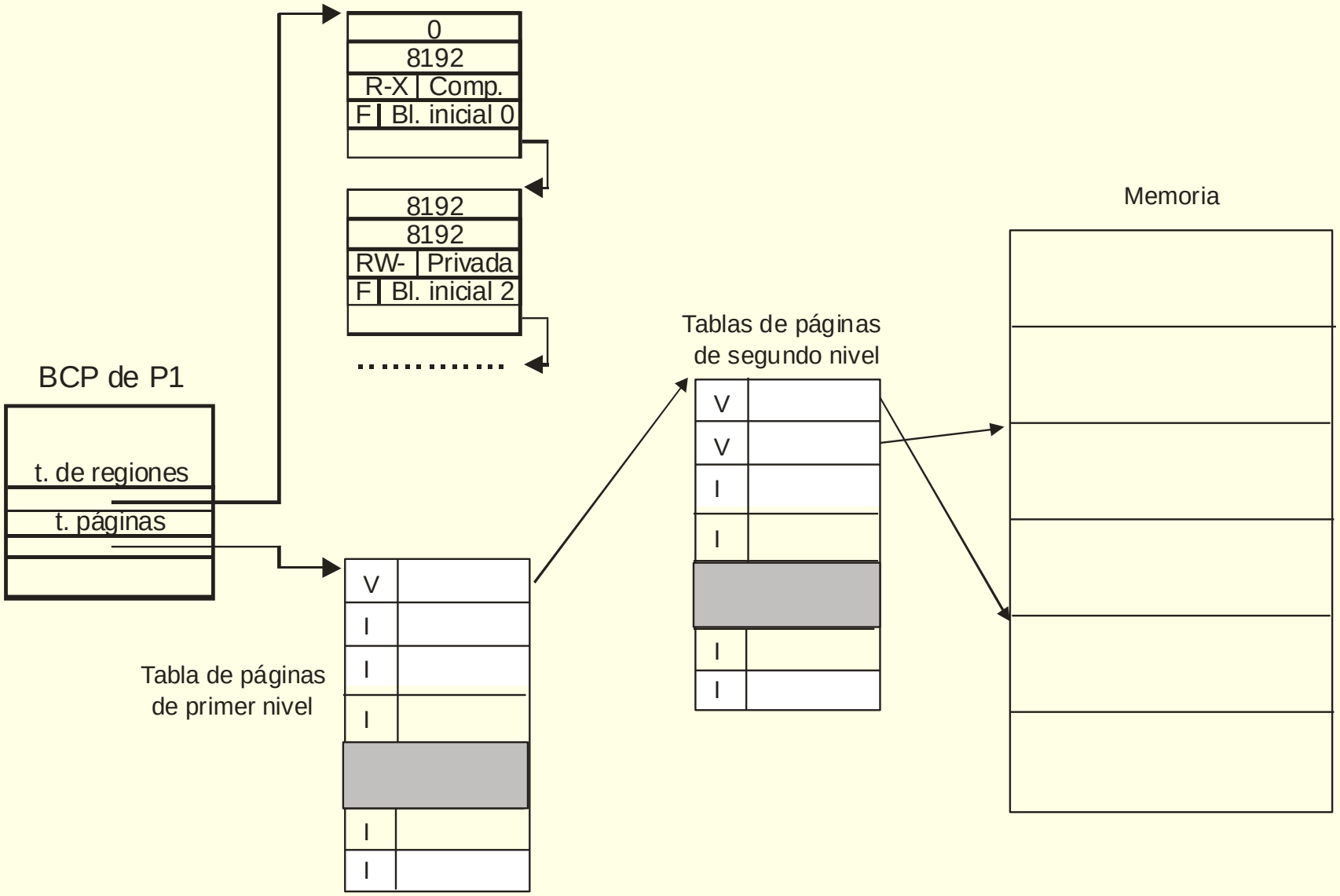
Política de localización: posibles ubicaciones



Política de extracción: Fallo de página

- Si dirección inválida → Aborta proceso o le manda señal
- Si no hay ningún marco libre (consulta T. marcos)
  - Reemplazo: pág P marco M → P inválida
    - Si *mod* → **escribir** fichero (comp) o *swap* (priv)
- Hay marco libre (se ha liberado o lo había previamente):
  - Inicia **lectura** de página en marco M
    - De fichero, swap o rellenar a ceros
  - Conecta entrada de TP a M
- Fallo de página en modo sistema no siempre es error:
  - Acceso a página de usuario no residente

Creación de tablas de páginas por demanda



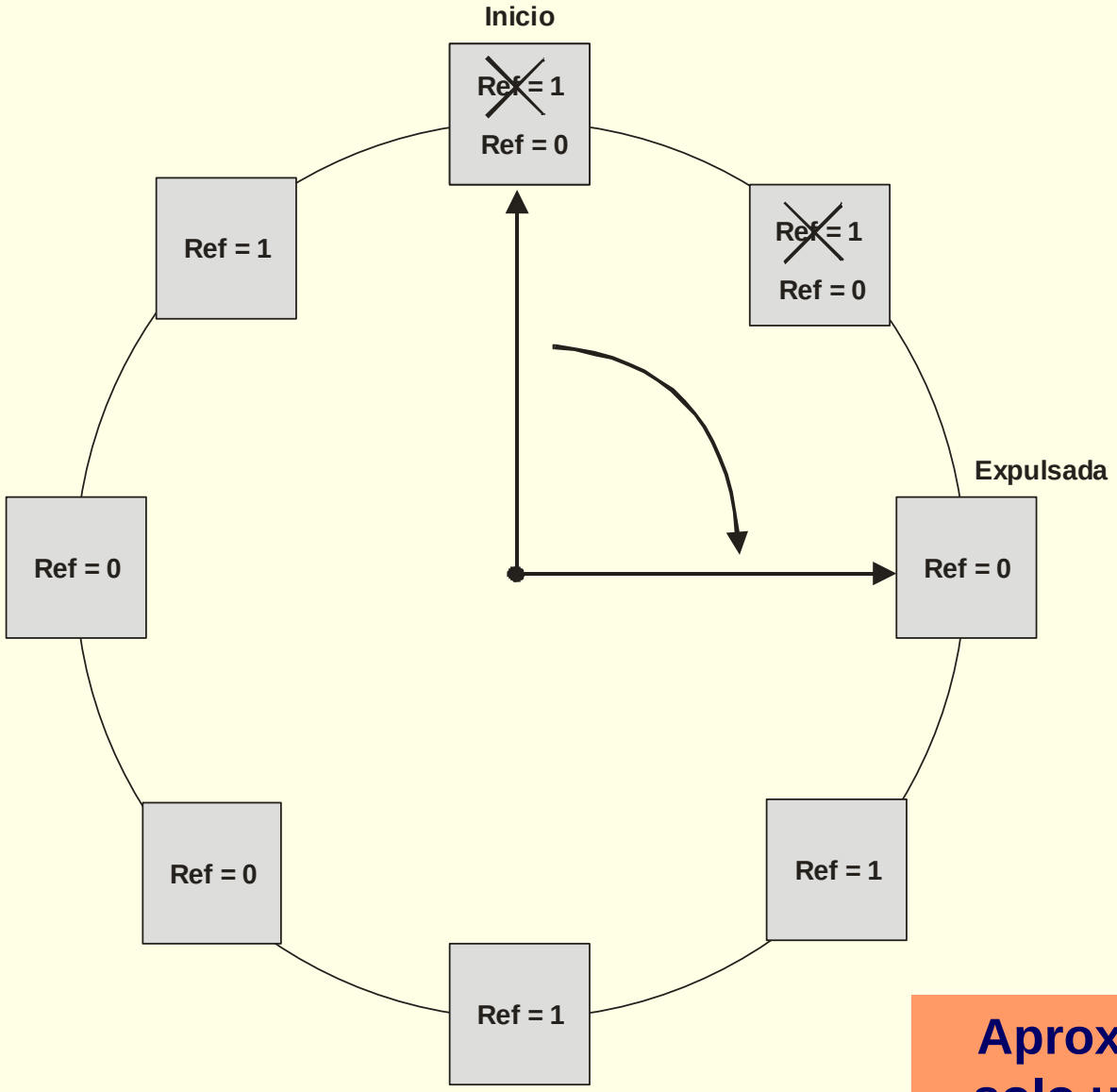
Política de reemplazo

- Tipo de reemplazo: local o global
- Factores a tener en cuenta:
  - Tiempo de residencia, Frecuencia uso y *Frescura* de la página
- Objetivo: Minimizar la tasa de fallos de página.
  - Poca sobrecarga en tiempo y espacio y MMU estándar
- También se usan en caché de sistemas de ficheros
- Algoritmo FIFO: Página que lleva más tiempo residente
  - Fácil implementación:
    - Páginas residentes en orden FIFO
    - No requiere el bit de página accedida (*Ref*)
  - Sobrecarga constante (independiente del tamaño de la memoria)
  - No es una buena estrategia: basado solo en tiempo de residencia
  - Anomalía de Belady: ejemplos donde ↑ nº marcos ⇒ ↑ nº fallos

Algoritmo LFU, LRU y más allá

- Algoritmo LFU (*Least Frequently Used*)
  - Página residente menos frecuentemente usada
  - Problema: muchos accesos en una fase y ninguno después
- Mejor: Algoritmo LRU (*Least Recently Used*)
  - Página residente menos recientemente usada (frescura)
  - Proximidad de referencias: pasado reciente → futuro próximo
- En ambos: No anomalía de Belady → algoritmo de pila
- Difícil implementación estricta: Precisarían una MMU específica
- Problema del LRU: No *scan-resistant*
  - Proceso accede a muchas páginas solo una vez
  - Quedan residentes solo esas páginas que no vuelven a usarse
- ¿Cómo lograr ser *scan-resistant*?
  - Añadir frecuencia a frescura
  - LRU/2 (en gral., LRU/K) tiene en cuenta el penúltimo acceso
  - ARC, CAR, LIRS, Clock-Pro…

Algoritmo del reloj (o 2ª oportunidad)



Aproximación a LRU que solo usa bit de referencia

Buffering de páginas y caché de páginas

- Reemplazo bajo demanda: Mejor por anticipado
- Reserva de marcos libres
- Fallo de página: siempre usa marco libre (no reemplazo)
- Si nº marcos libres < umbral
  - "demonio de paginación" aplica algoritmo de reemplazo
    - páginas no modificadas → lista de marcos libres
    - páginas modificadas → lista de marcos modificados
      - cuando se escriban a disco pasan a lista de libres
- Si se referencia una página mientras está en estas listas:
  - fallo de página la recupera directamente de la lista (no E/S)
- Caché de páginas: Encontrar eficientemente si página residente:
  - Necesario en *buffering* y al compartir
    - [Fichero|Disp. *swap*, nº bloque] → {nº marco | !residente}
- Sistema de ficheros también incluye caché de bloques
  - Tendencia: fusión de caché de páginas y de bloques

Retención de páginas en memoria

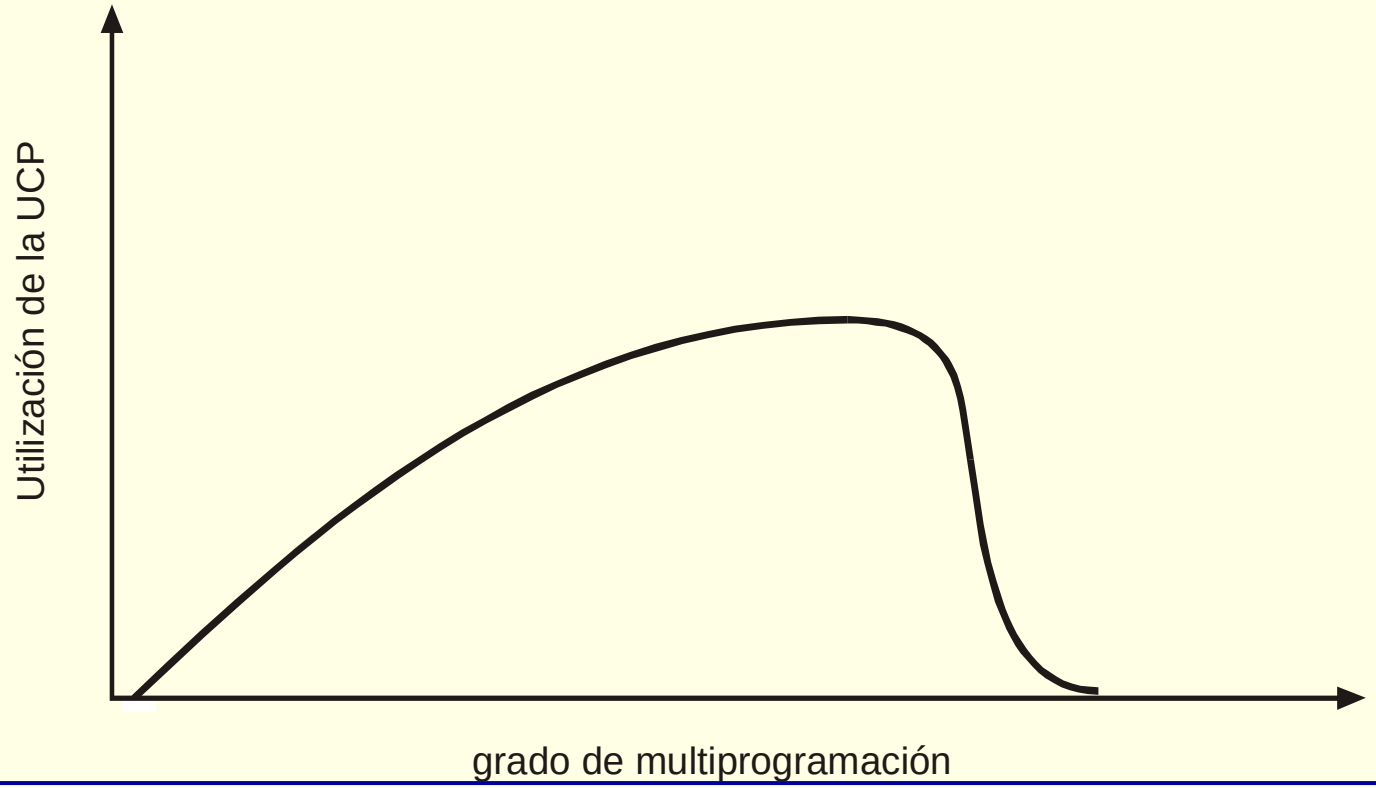
- Páginas marcadas como no reemplazables
- Se aplica a páginas del propio SO
  - SO con páginas fijas en memoria es más sencillo
- También se aplica mientras se hace DMA sobre una página
- Servicio para fijar en memoria una o más páginas de su mapa
  - Adecuado para procesos de tiempo real
  - Puede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

Hiperpaginación (*Thrashing*)

Tasa excesiva de fallos de página en el sistema

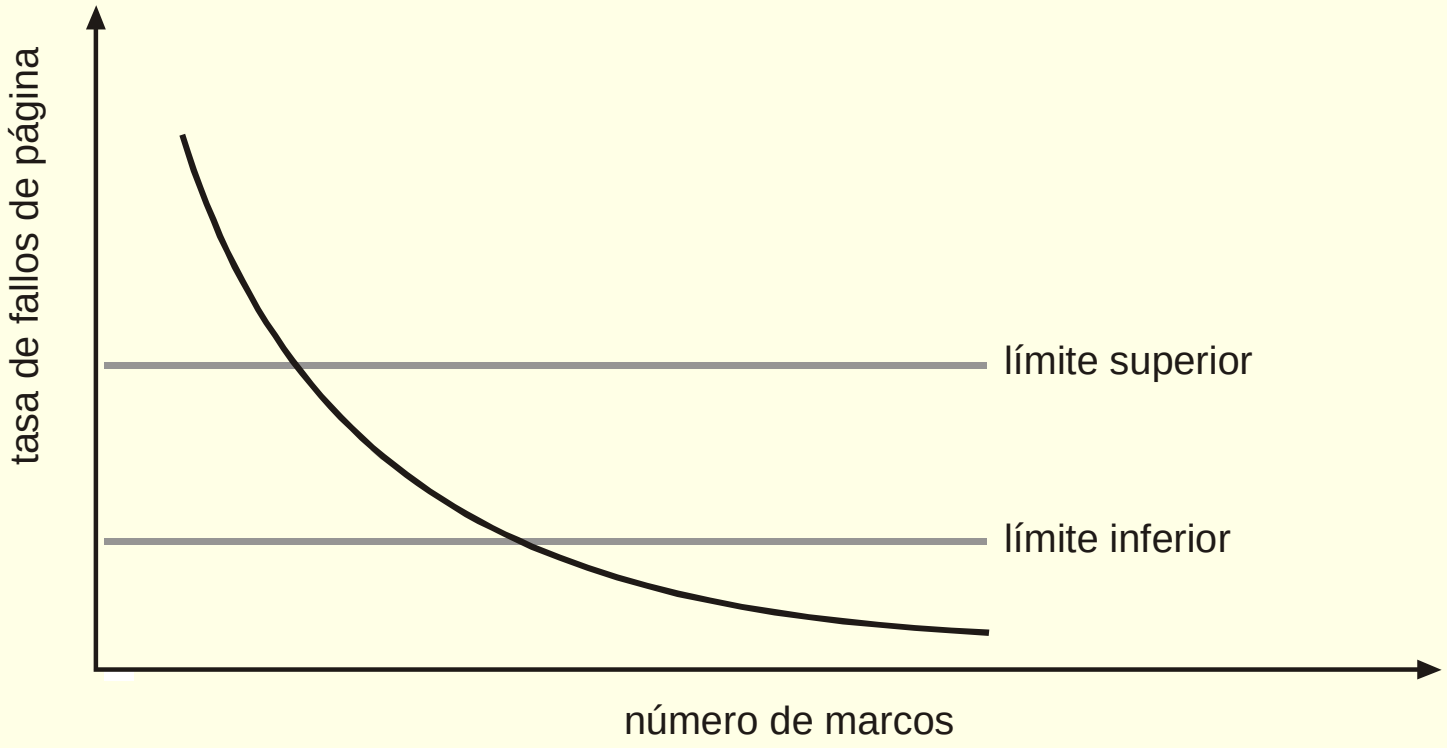
Si demasiados procesos, no caben en memoria sus conjuntos de trabajo

- Rendimiento del sistema cae en picado
- Control de carga: Hay que suspender la ejecución de procesos



Control de carga basado en frecuencia de fallos

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



Dispositivo de paginación (*swap*)

- Disco, partición o fichero (más lento)
  - Incorporación dinámica y configurable de espacio de *swap*
- Estructura: cabecera y bloques; mapa de bits solo en memoria
- Preasignación de espacio
  - Al crear región privada o sin soporte
  - Permite detectar síncronamente la falta de espacio de *swap*
- Sin preasignación de espacio
  - En primera expulsión se le asigna espacio en *swap*
  - Mejor aprovechamiento de espacio de almacenamiento
- Regiones privadas o sin soporte usan el *swap*
  - Compartidas con soporte usan directamente fichero
- Bloque de *swap* puede compartirse: contador de referencias

Compartimiento de páginas

- Escenarios:
  - Zona de memoria compartida
  - Compartir código de programa o biblioteca
  - Fichero proyectado en modo compartido
  - Regiones compartidas después de `fork`
- Compartir igual que con paginación convencional pero:
  - Caché de páginas para localizar pág compartida ya residente
  - Traducción inversa al expulsar página compartida
    - De marco de página a entradas de TP que lo referencian

Duplicado perezoso de páginas

- Escenarios de uso duplicado:
  - Datos con valor inicial de programas y bibliotecas
  - Regiones privadas después de `fork`
  - Fichero proyectado en modo privado
- Optimización: Duplicado por demanda (*copy-on-write*, COW)
  - Se comparte una página mientras no se modifique
  - Si un proceso la modifica se crea una copia para él
- Implementación de COW
  - Se comparten páginas de regiones duplicadas pero:
    - se marcan de solo lectura en TP (no en t. regiones)

Tratamiento del fallo de COW

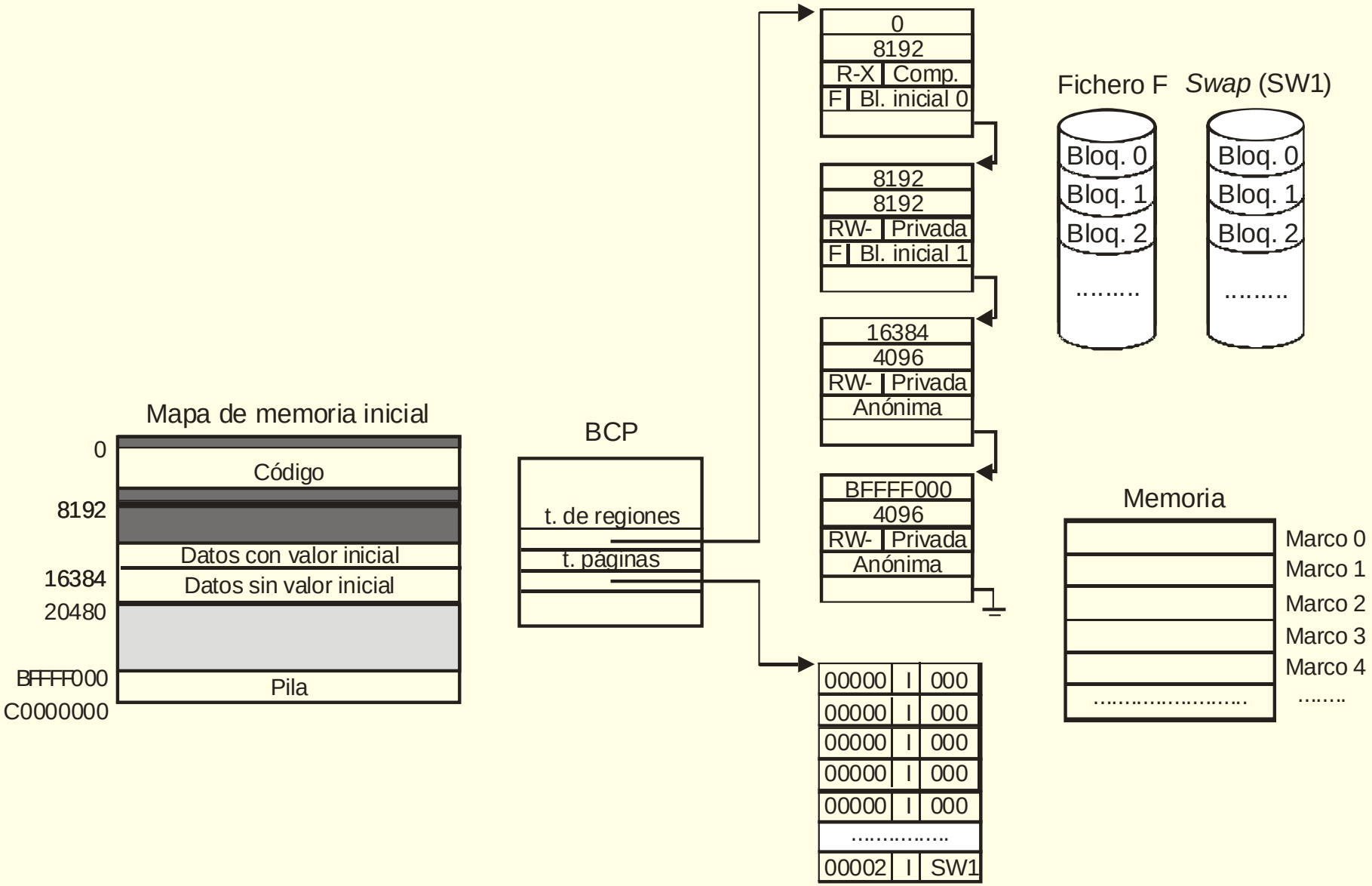
- Si dirección $\notin$ región WR $\rightarrow$ Aborta (o señal) proceso
- Si ref > 1
  - Reserva marco libre, copia contenido y conecta a TP
  - Devuelve permiso WR a entrada, ref--, no inserta caché págs
- Si ref == 1
  - Devuelve permiso WR a entrada, elimina de caché págs

- **Recordatorio importante**:
  - solo se asigna espacio (marcos) en fallo de página y de COW
    - **Nunca** al crear el mapa o una región del mismo

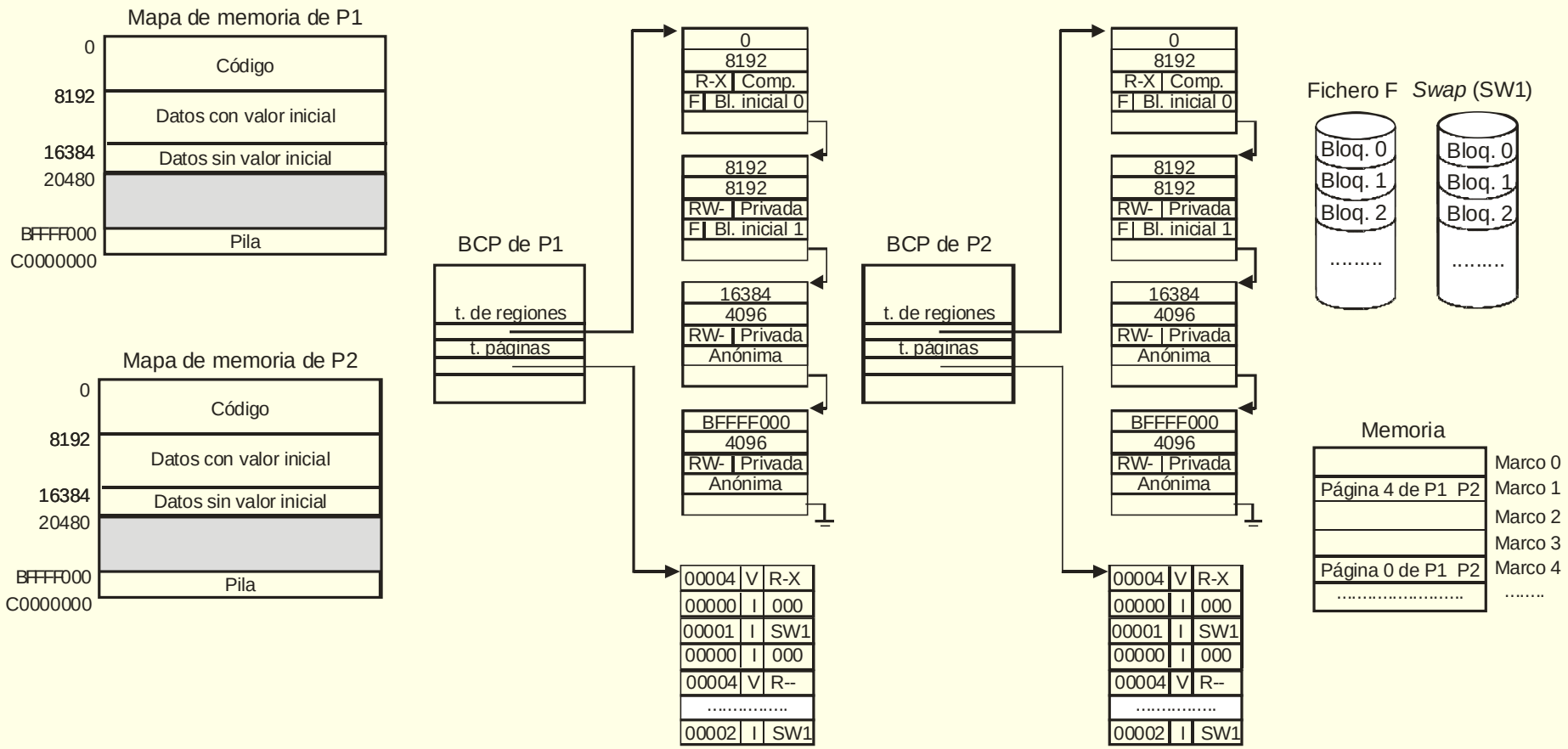
Operaciones en el nivel de proceso

- Crear una región: No asigna m. principal ni entradas TP
  - Busca zona libre en mapa de proceso usando t. regiones
    - Excepto si ubicación de la región está prefijada
  - Reserva y rellena entrada t. regiones
    - Si preasignación, reserva *swap*
- Eliminar una región:
  - Libera entrada t. regiones e invalida entradas TP
  - Marcos y bloques de *swap*: ref--
- Redimensionar una región (*heap* o pila)
  - Ajusta entrada t. regiones
    - Decrece: Invalida entradas TP + marcos y *swap*: ref--
    - Crece: Nada más, excepto reserva *swap* si preasignación
  - *heap* crece por llamada al SO pero pila automáticamente
    - Extensión tratamiento de fallo página para la expansión de la pila:
      - Si dir fallo > SP y < dir inicio región de pila → expansión

Creación del mapa inicial

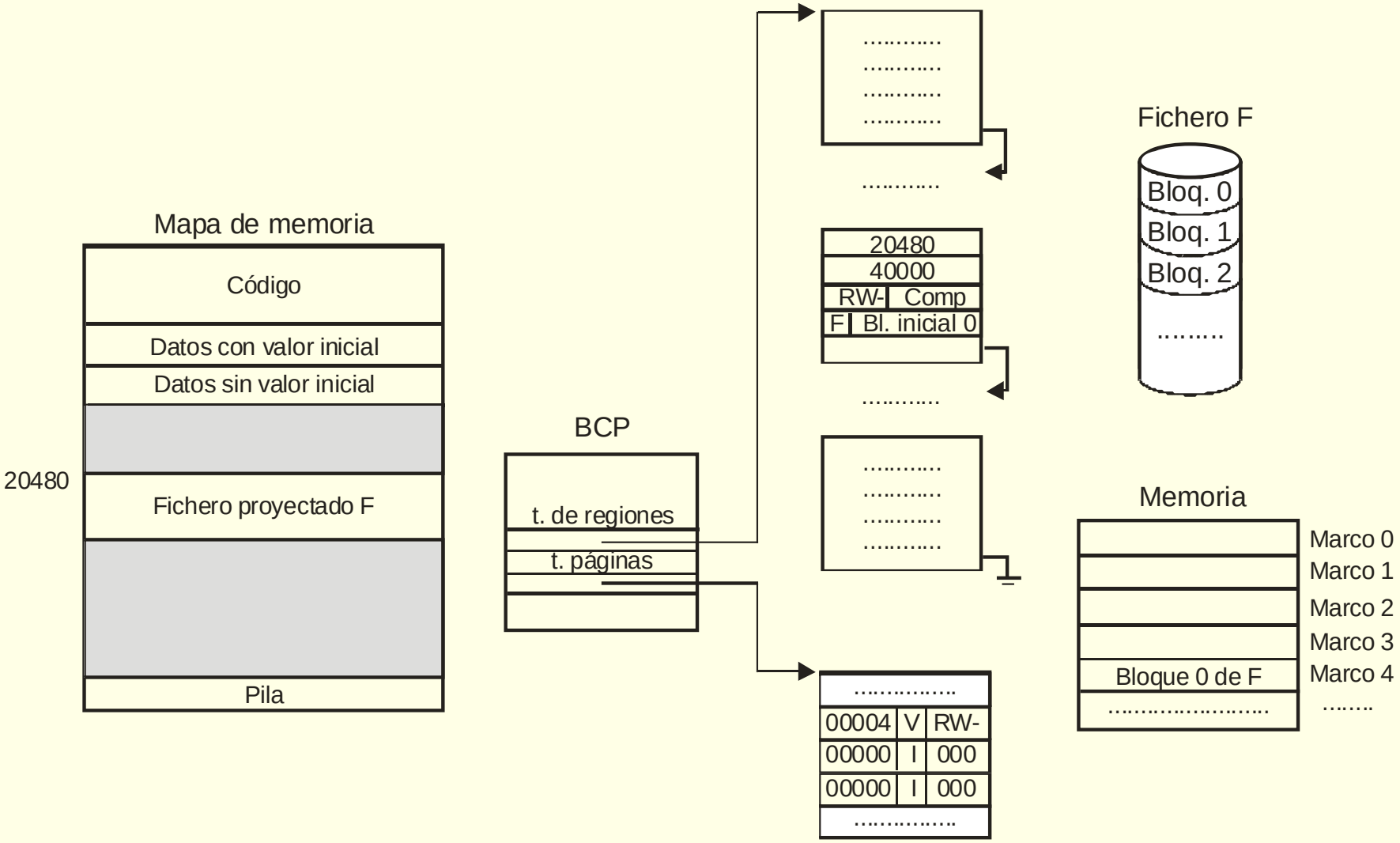


Resultado del `fork`



Duplica tablas de páginas eliminando permiso de escritura de páginas de regiones privadas

Ficheros proyectados en memoria

- Generalización de memoria virtual
  - Entradas de TP referencian a un fichero de usuario
- Programa solicita proyección de fichero (o parte) en su mapa
  - Puede especificar protección y si privada o compartida
- Programa accede a posición de memoria asociada a fichero
  - Está accediendo al fichero
- Forma alternativa de acceso a ficheros frente a *read*/*write*
  - Menos llamadas al sistema
  - Se evitan copias intermedias en caché de sistema de ficheros
  - Se facilita programación: acceso a estructuras de datos

Proyección de un fichero