Sistemas Operativos
2º Curso – Grado en Ingeniería Informática

Tema 3:

Gestión de memoria

José Antonio Gómez Hernández, 2016.

Gestión de memoria: índice

- Repaso (Tema 2 de FS):
  - Conceptos generales de gestión de memoria: hardware y software
  - Mecanismos de gestión: Paginación y segmentación
- Paginación multinivel
- Paginación bajo demanda
- Gestión de memoria en Linux:
  - Gestión de memoria del kernel
  - Gestión de memoria para procesos
  - Cachés del sistema

0.

Repaso

Conceptos básicos de gestión de memoria y paginación

Repaso: recursos

▷ Tema 2 de Fundamentos del Software.

▷ Libro de W. Stallings:

- Elementos hardware: Apartados 1.5 y 1.6.
- Gestión de memoria: Apartados 7.1 a 7.3
- Memoria virtual: Apartado 8.1, epigrafes de “Proximidad y memoria virtual” y “Paginación” (sin ver paginación a dos niveles, ni la tabla invertida de páginas, si el “Búfer de traducción anticipada”). El restp de epígrafes no es necesario.

Elementos hardware

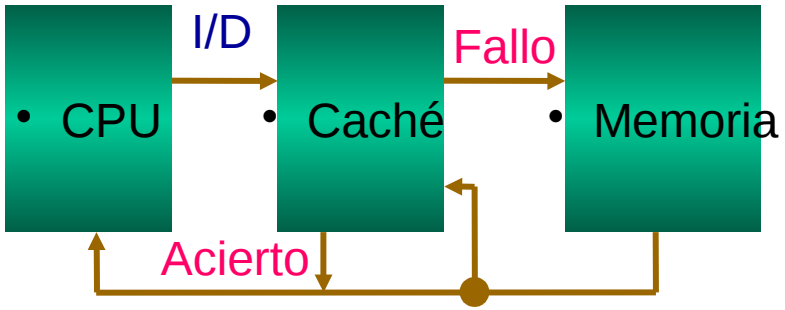
▷ Los computadores mantienen una jerarquía de memoria para mejorar el acceso de la CPU a memoria.

▷ Elementos importantes de la misma son las *cachés*: memorias intermedias que mantienen instrucciones/datos previamente accedidos y que son más rápidas que la RAM.

▷ Para cachés, definimos:

- *Acierto de cache*: la instrucción/dato buscado está en ella.
- *Fallo de caché*: la instrucción/dato no esta en la misma.

▷Esquema:



Cachés

▷ Las cachés “funcionan” porque explotan las **localidad** de las referencias del código, datos, y pila de los programas.

▷Tipos de localidad:

- *Espacial*: si un item es referenciado, las direcciones próximas a él tienden también a ser referenciadas.
- *Temporal*: si un item referenciado, tiende de nuevo a ser referenciado en breve.

▷Cuando se usan cachés, el coste de acceder a memoria viene dado por el Tiempo de Acceso Efectivo (TAE):

▷ `TAE = p*ta + (1-p)*tf`

▷donde: `p` =probabilidad de acierto; `ta` =tiempo de acceso si hay acierto; `1-p` = probabilidad de fallo, y `tf` =tiempo de acceso si hay fallo.

Requisitos

▷El SO asigna memoria a los procesos para su ejecución, garantizando:

- *Protección*: Un proceso no accede a memoria de otro. Diferentes módulos del programa deben tener diferentes permisos de acceso.
- *Compartición*: De datos/código entre procesos. Permite el ahorro de memoria.
- *Reubicación*: En sistemas multiprogramados, un programa debe poder cargarse en diferentes zonas de memoria.

Diseño

- El SO debe esconder la **Organización física** (jerarquía de niveles, estructura no lineal) de la memoria física.
- para que el usuario tenga una **visión lógica** simple de la memoria como una matriz lineal. Además permitirá la estructuración de un programa en módulos.

Espacios lógicos y físicos

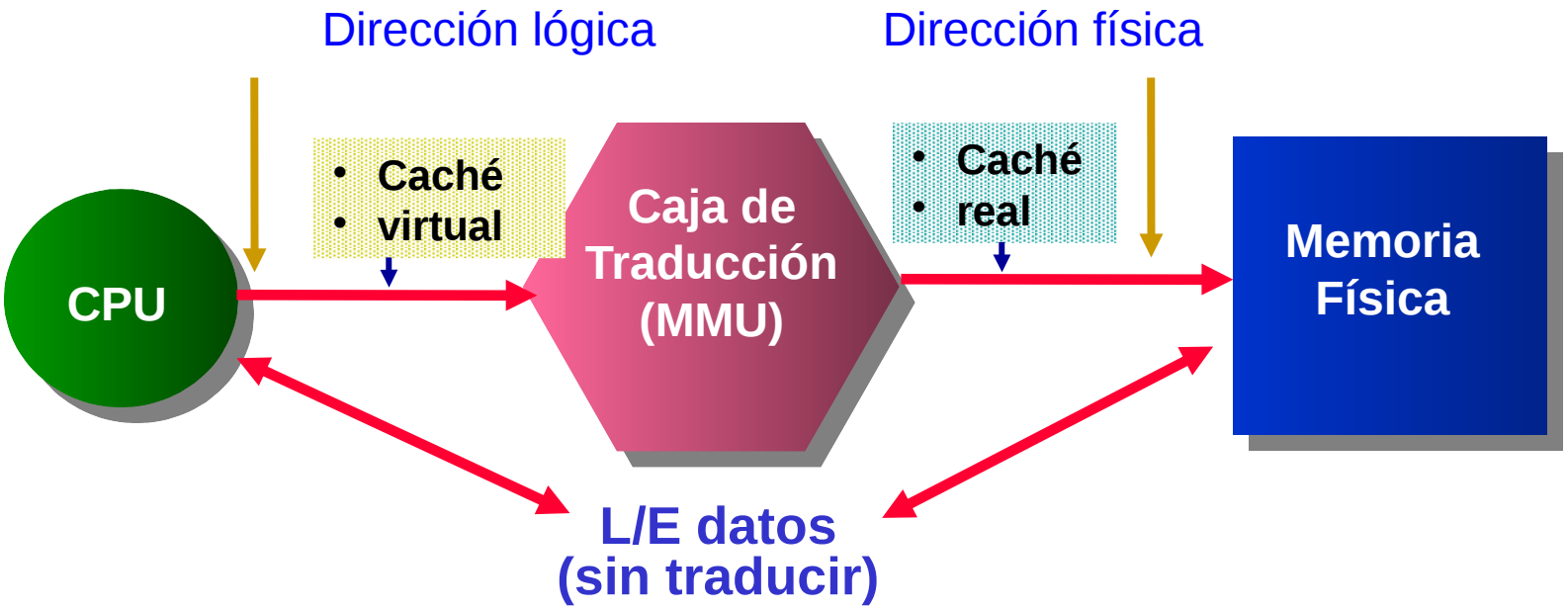
▷ La necesidad de poder reubicar un programa en memoria hace necesario separar el espacio de direcciones generadas por el compilador, **espacio lógico o virtual**, del **espacio físico** en el que se carga, el espacio de direcciones físicas (en RAM).

▷ Denominamos:

- **Dirección lógica** - la generada por la CPU; también conocida como virtual.
- **Dirección física** - dirección que se pasa al controlador de memoria.

Traducción de direcciones

▷ La separación de espacios me obliga a realizar una traducción de direcciones:



Unidad de Gestión de Memoria

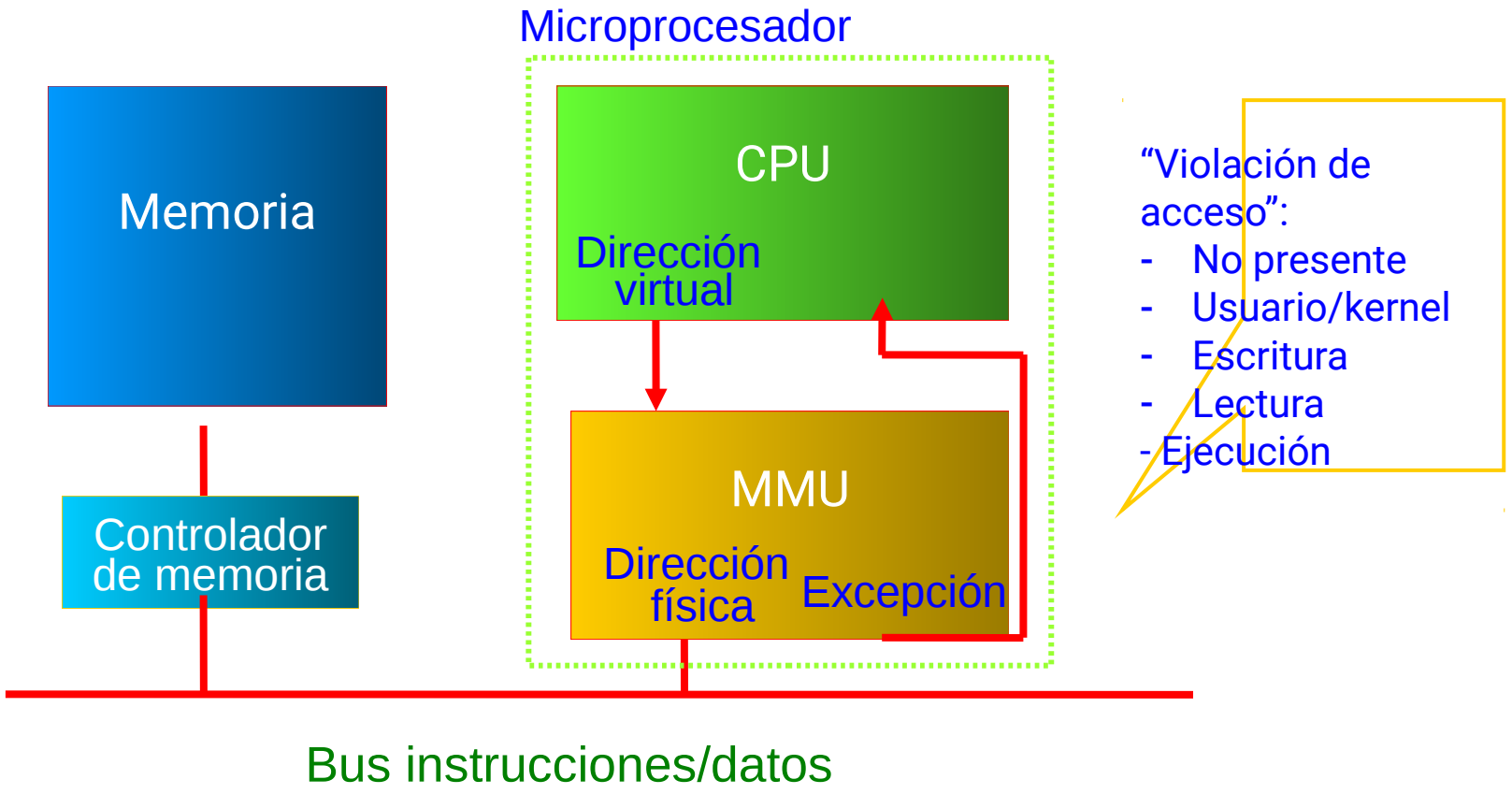
▷ La MMU (Memory Management Unit) es el dispositivo hardware que:

- Traduce direcciones virtuales en direcciones físicas.
- Implementa la protección.

▷ El hardware determina la forma en la que el SO gestiona la MMU.

▷ La forma de la MMU dependerá del esquema de gestión de memoria implementado en hardware. En el esquema más simple, contendrá un registro de reubicación que almacena el valor a sumar a cada dirección generada por el proceso de usuario al mismo tiempo que es enviado a memoria.

MMU: funcionamiento



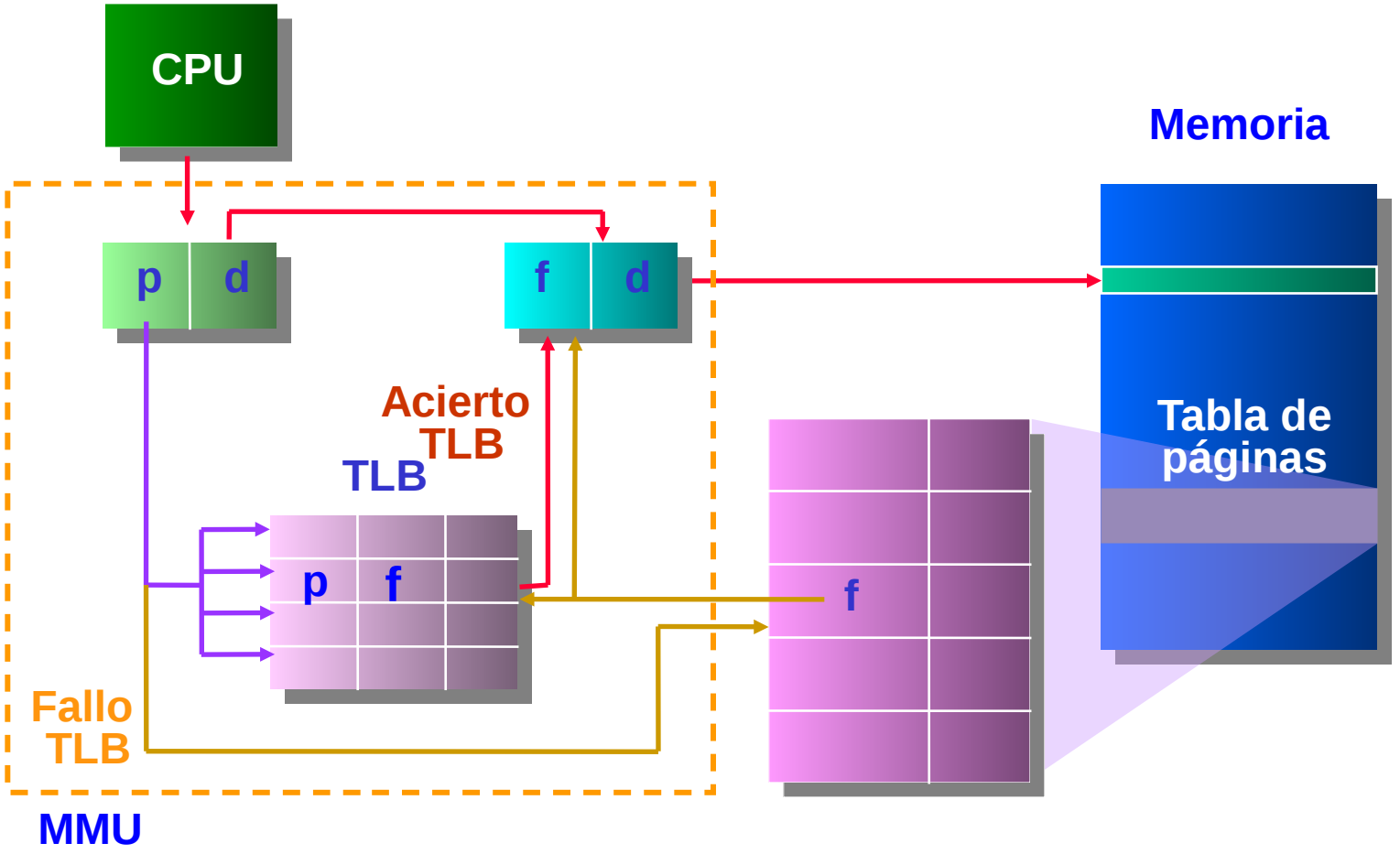
Fragmentación

- Denominamos **fragmentación de memoria** ha aquella fracción de la misma que no es asignable debido al propio mecanismo de gestión de memoria.
- Los sistemas de gestión de memoria han evolucionado con el objetivo principal de reducir la fragmentación de memoria.
- Al desacoplar los espacios lógicos de los físicos, podemos hacer que el espacio de direcciones de un proceso no sea continuo, podemos trocearlo, reduciendo así la demanda de memoria contigua.
- Los SOs actuales suelen utilizar paginación como esquema básico de gestión de memoria, si bien, dependiendo del procesador, deben también utilizar segmentación. Por ejemplo, los procesadores Intel implementan segmentación como esquema básico de gestión de memoria (protección:modos de funcionamiento del procesador) y opcionalmente se puede activar o no la paginación.

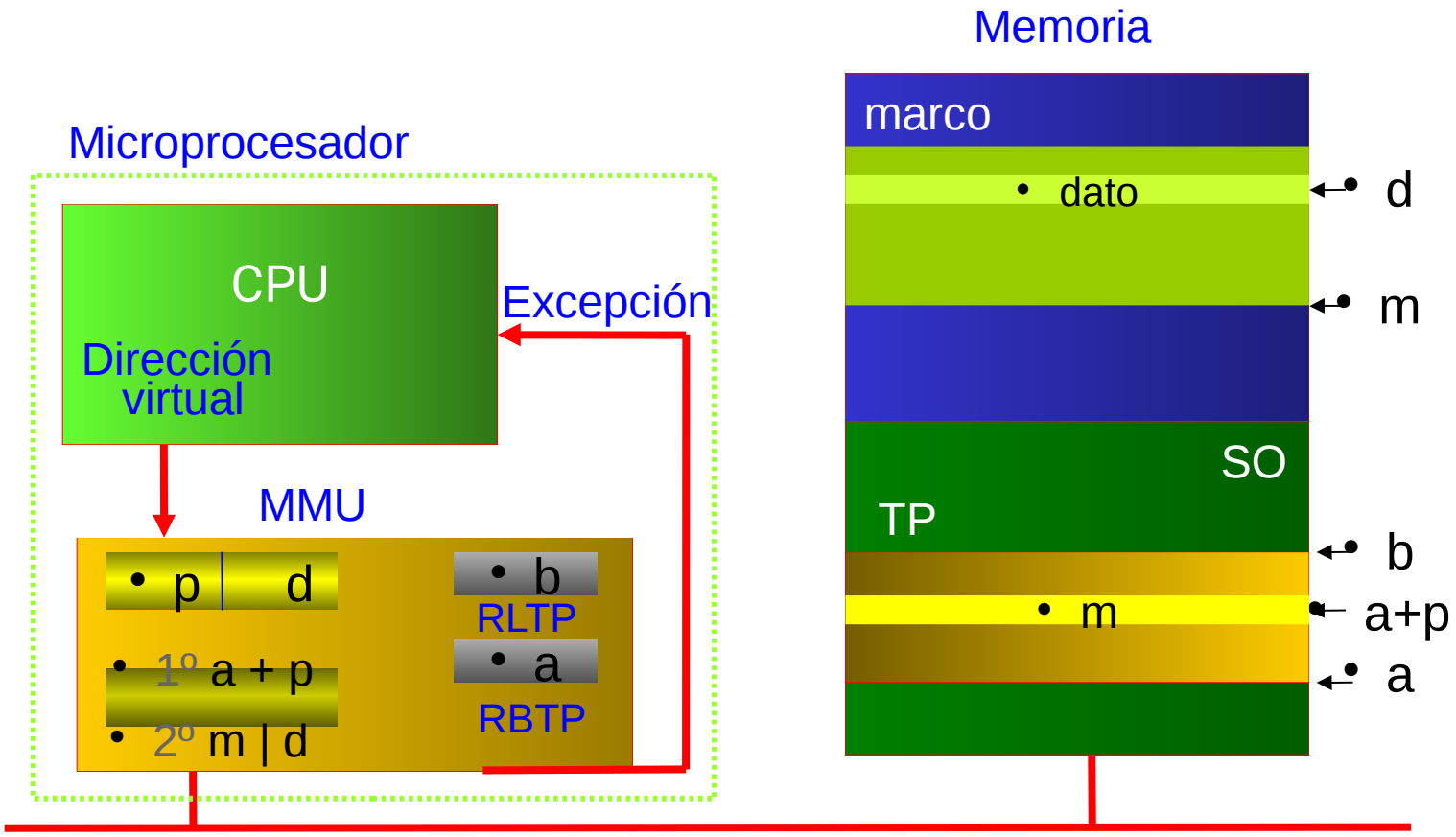
Paginación

- La MMU “divide” el programa en bloques del mismo tamaño, denominados **páginas**, para cargarlos en bloques de memoria principal del mismo tamaño, denominados **marcos**.
- Esto permite evitar que la búsqueda de un hueco de RAM para cargar una página sea una asignación dinámica.
- El SO mantiene la pista de cuales son los marcos que contienen las páginas de un programa mediante una estructura de datos por proceso denominada **tabla de páginas** (TP). Esta estructura tiene una **entrada de TP** (PTE) por cada página del proceso, donde cada entrada indica cual es la dirección base de memoria principal del marco que la contiene. También contiene información de protección de la página. Si una entrada no es válida en el espacio de direcciones se desactiva el *bit de validez* de la PTE correspondiente.

Paginación: traducción



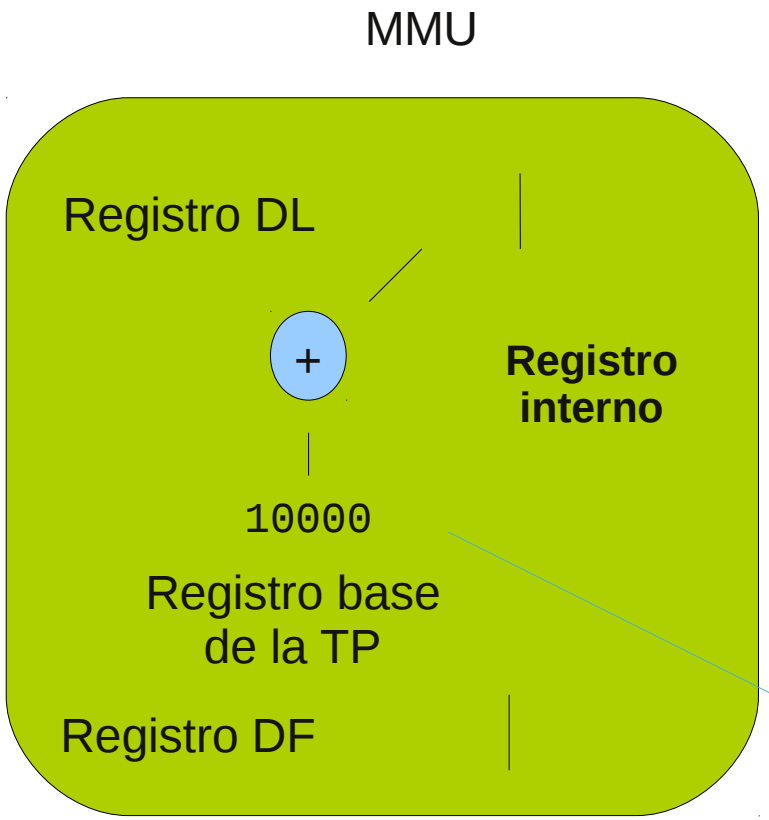
Paginación: ejemplo de traducción



Paginación: ejemplo

Espacio lógico

00000	I1
00001	I2
00010	I3
00011	I4
00100	D1
00101	D2
00110	D3
00111	No ini
01000	No ini
01001	No ini
01010	No ini
01011	-



Memoria principal

00000	I1
00001	I2
00010	I3
00011	I4
00100	
00101	
00110	
00111	
01000	D1
01001	D2
01010	D3
01011	-
. . .	

10000	000	1
10001	010	1
10010	000	0

Tabla de páginas

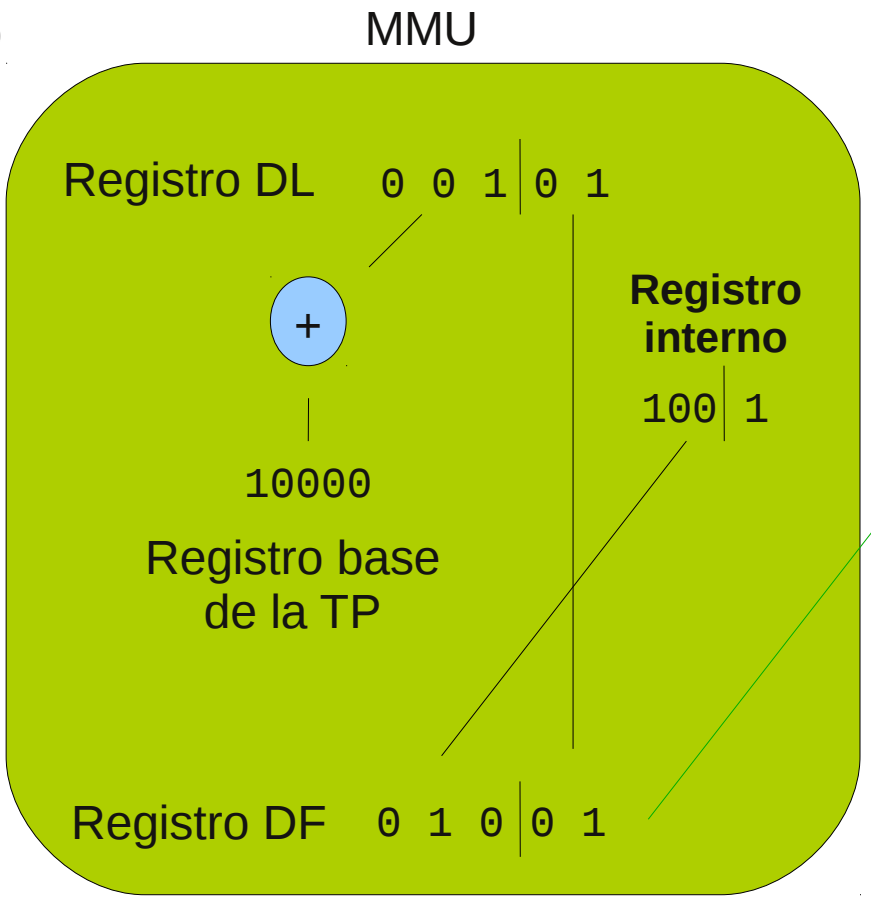
Dirección base del marco

Bit de validez

Inicialización:

- Cargamos la página 0 en el marco 0
- Cargamos la página 1 en el marco 2
- La página 2 no necesita de momento memoria (bit de validez 0)
- Ajustamos las PTEs: - la la base del marco 0 (000) y bit validez a 1
  - la base del marco 2 (010) y bit validez a 1
- El RBTP se ajusta a la dirección de inicio de la TP

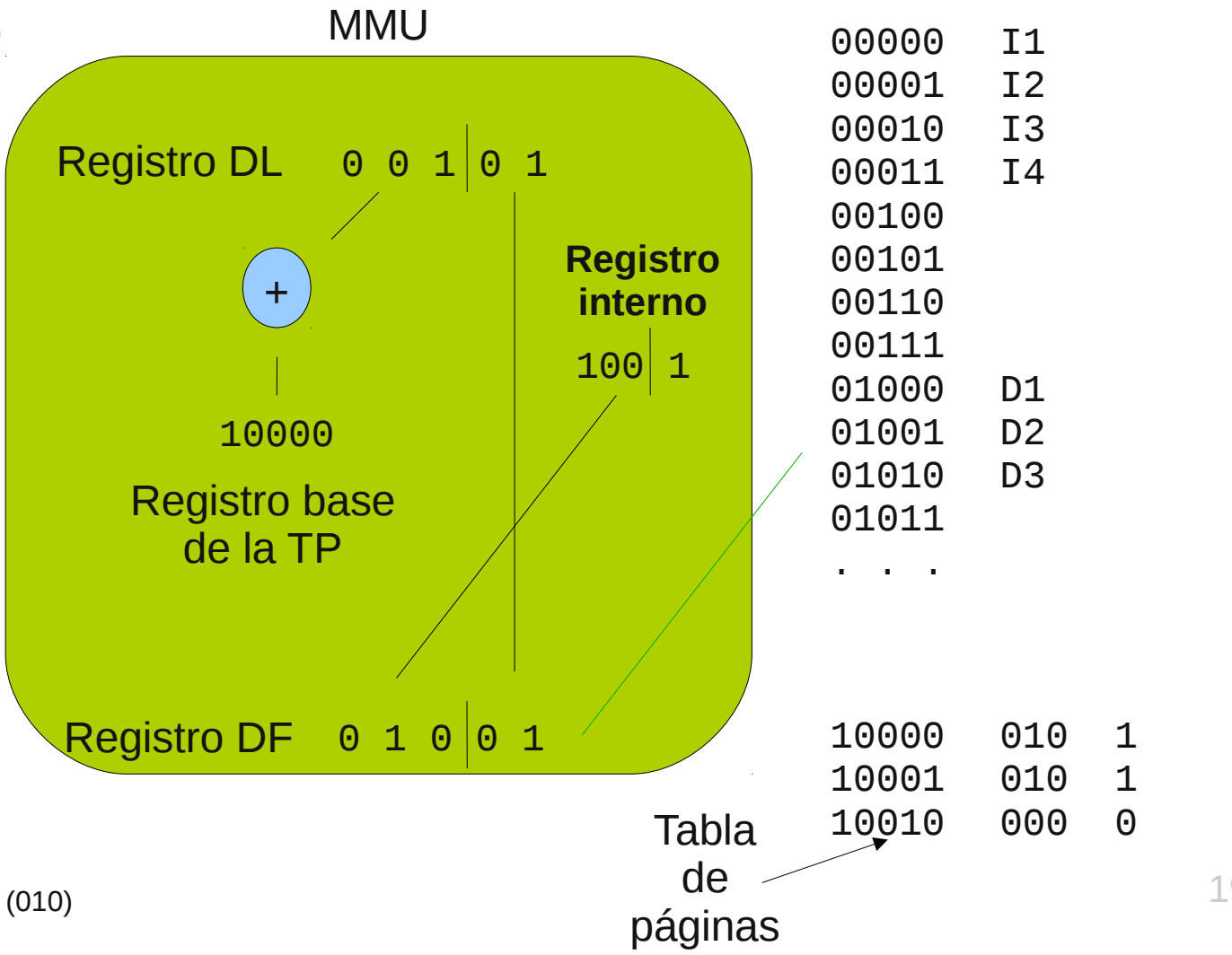
Paginación: traducción (paso 1)



Traducción: paso 2

- Concatenamos:
  - la dirección base del marco (010)
  - el offset de la dirección (01)

Paginación: traducción (paso 2)



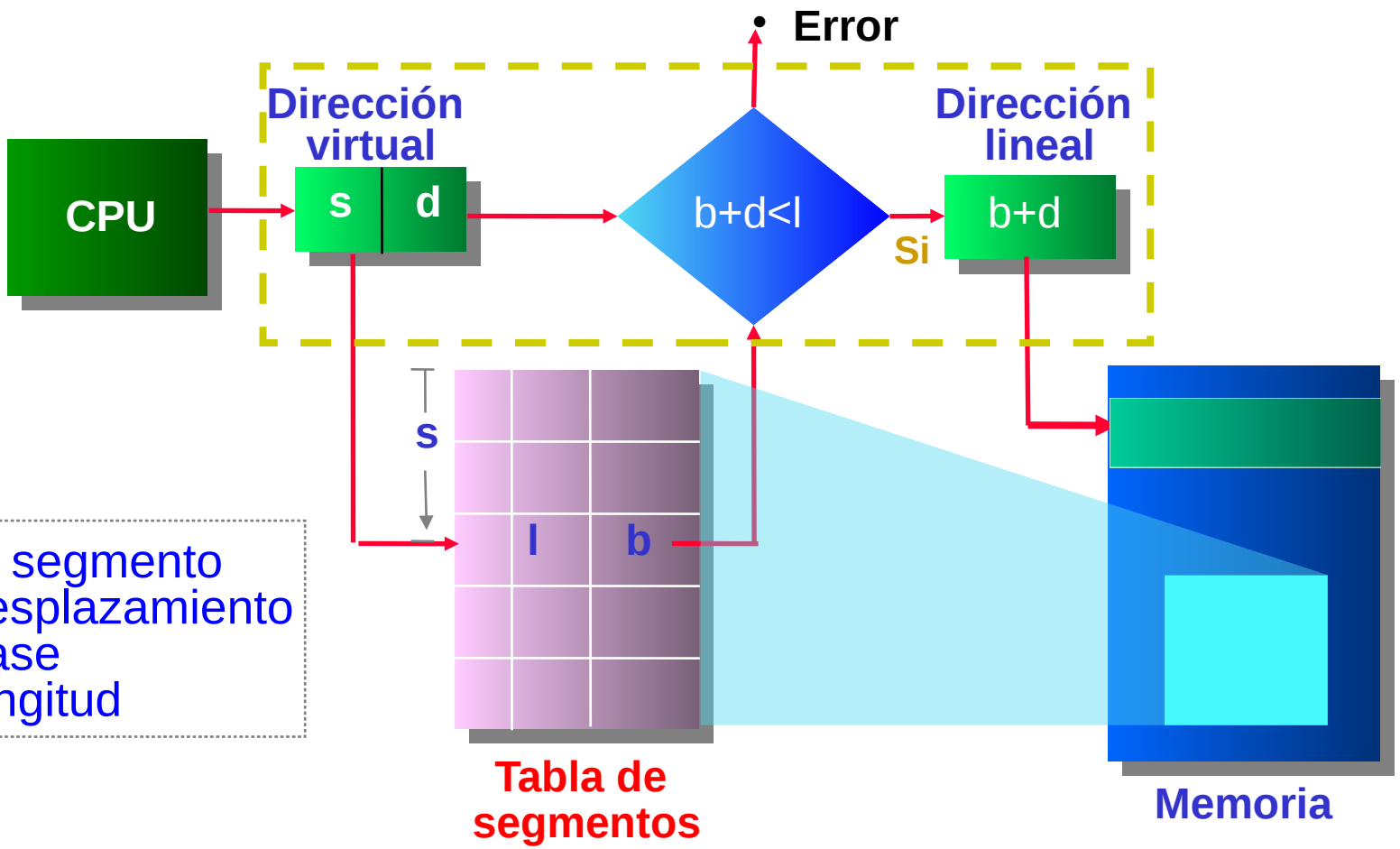
Traducción: paso 2

- Concatenamos:
  - el offset de la dirección (01)
  - la dirección base del marco (010)

Segmentación

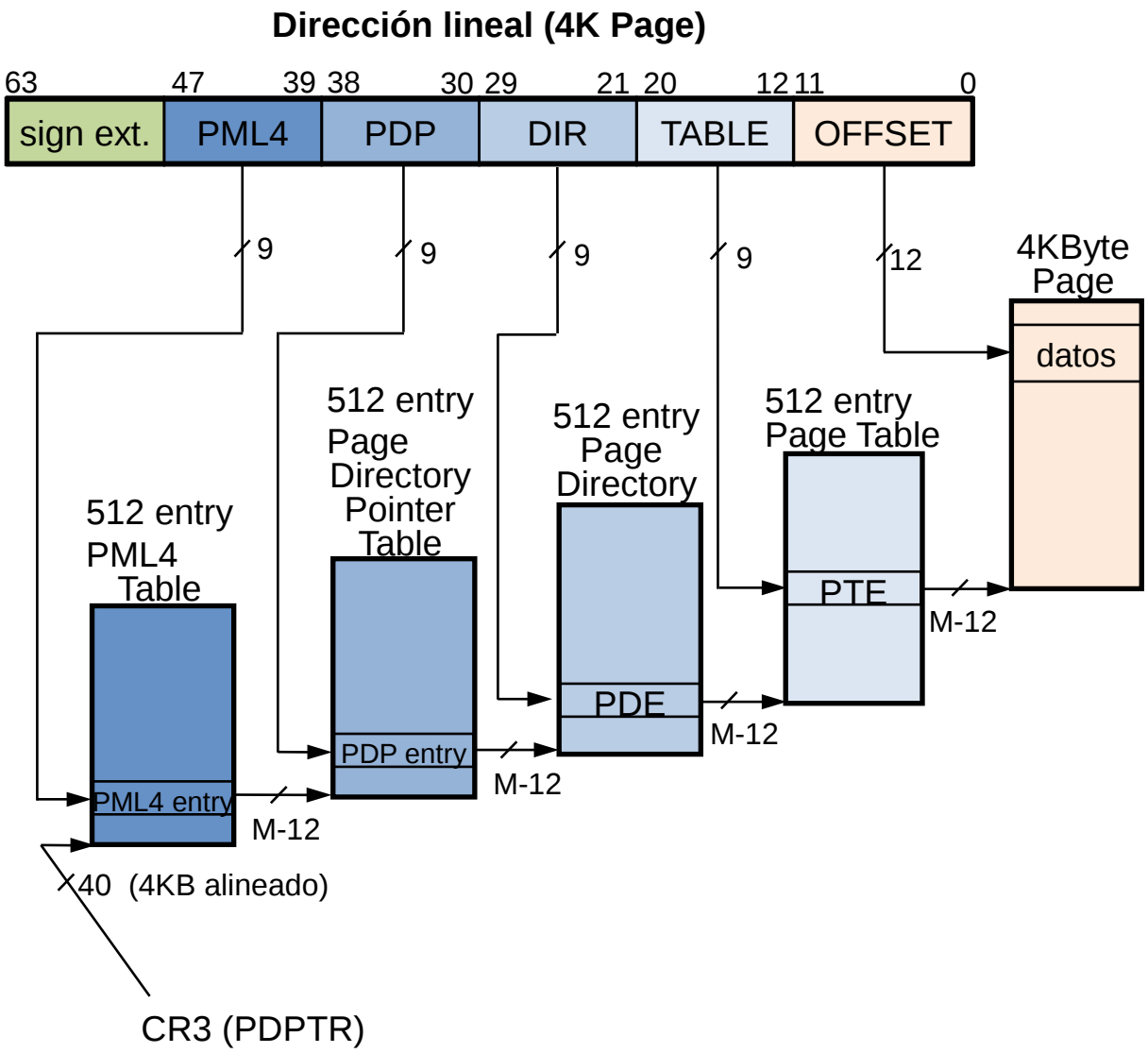
- Troceamos el programa en unidades lógicas de programación (procedimientos, pilas, código, datos, tabla de símbolos, etc.) denominadas segmentos.
- Cada segmento suele tener una tamaño diferente del resto.
- Ahora, una dirección lógica es una tupla:
- <número_de_segmento, desplazamiento>
- La **Tabla de Segmentos** aplica direcciones bidimensionales definidas por el usuario en direcciones físicas de una dimensión (lineales). Cada entrada de la tabla de segmentos tiene los siguientes elementos:
  - *base* - dirección física donde reside el inicio del segmento en memoria.
  - *límite* - longitud del segmento.

Segmentación: esquema

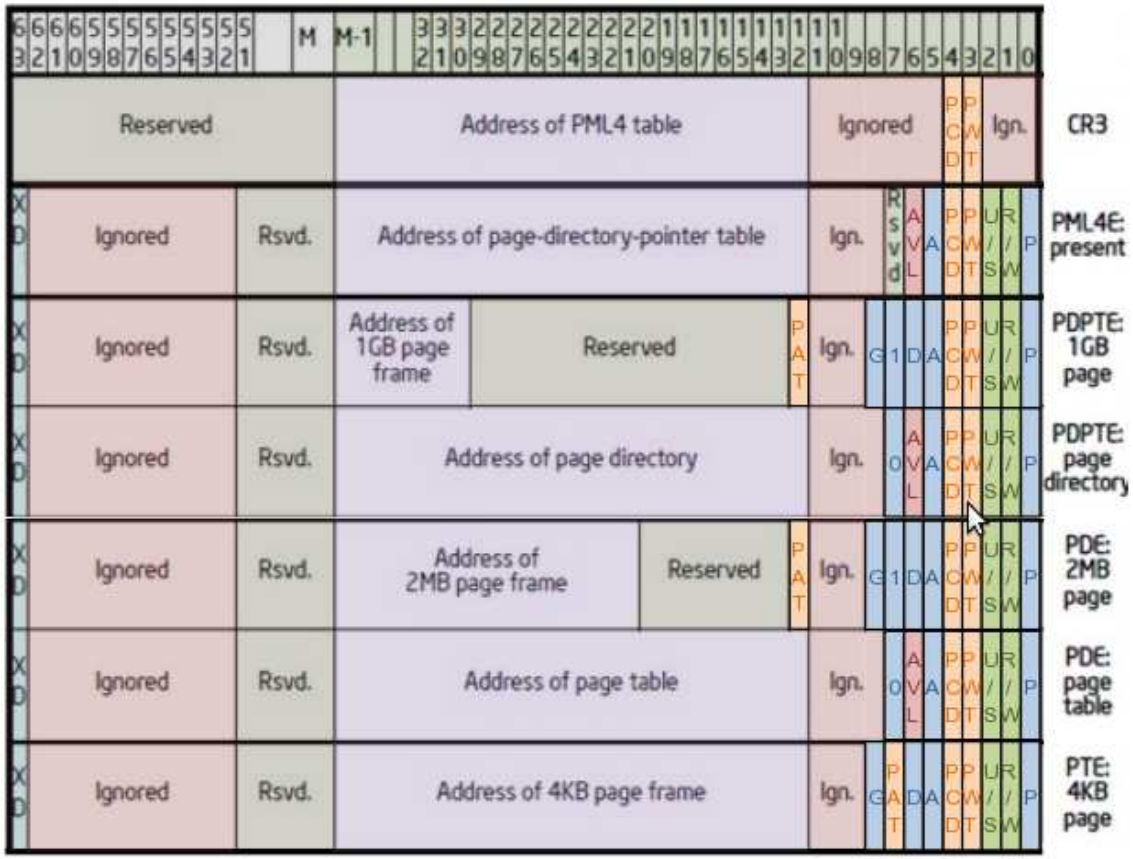


- s = nº segmento
- d = desplazamiento
- b = base
- l = longitud

Paginación Intel x64: páginas 4KB



Entradas de Tablas de Pg en x64



- Memoria Virtual:
  - Presente (P)
  - Accedida (A)
  - Sucia (D)
  - Gobal (G)
  - Tamaño pg. (bit 7– 0:4KB;1:4MB)
- Protección:
  - Escritura (RW)
  - Usuario/supervisor (U/S)
  - Ejecución (XD)
- Caché:
  - Página Write-Though (PWT)
  - Cache pg. Deshabilitada (PCD)
  - PAT (atributo indice PT)(PAT)

1.

Paginación multinivel

Justificación e implementación

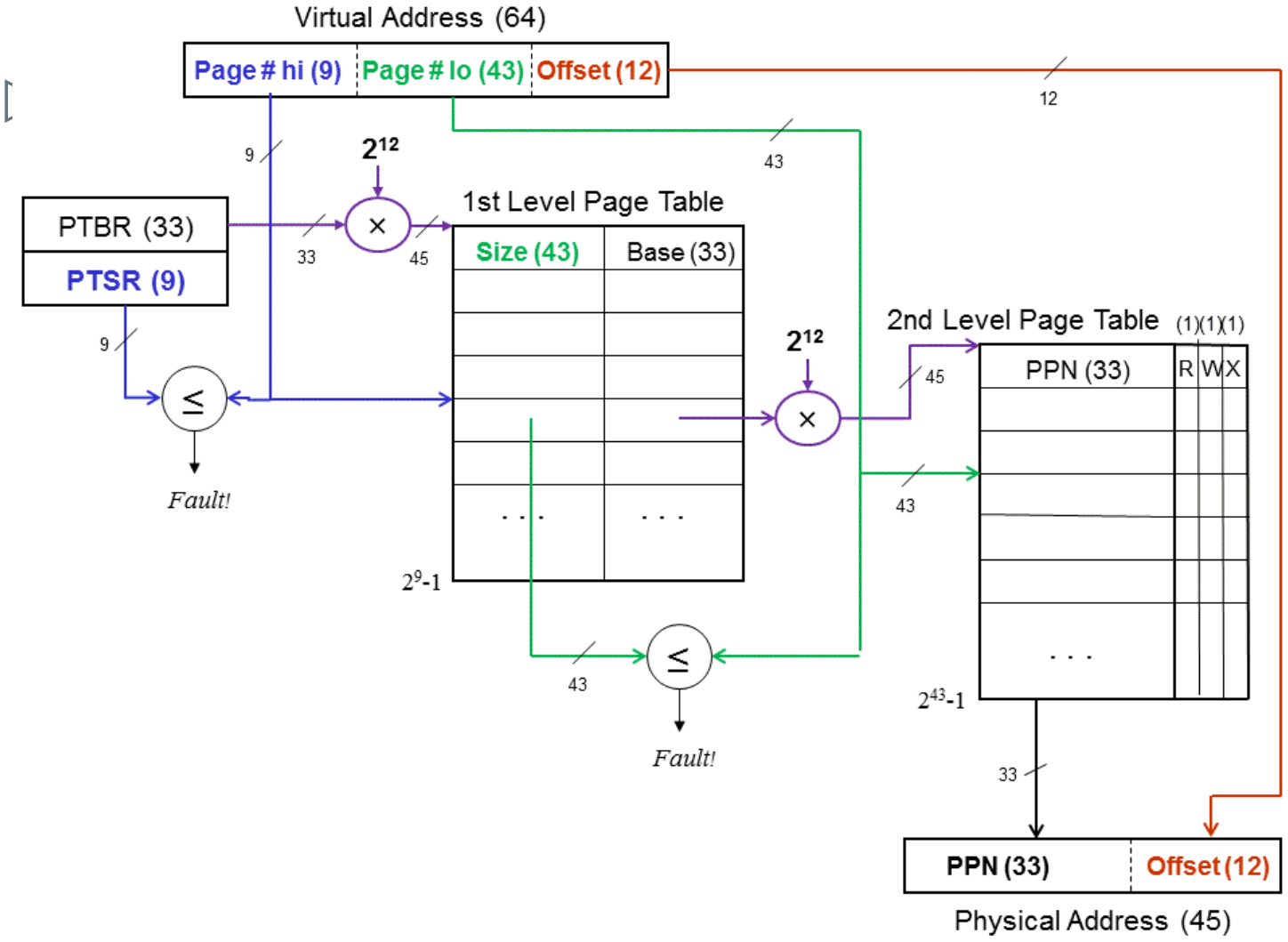
Paginación multinivel: justificación

- Supongamos una arquitectura de 32 bits, que permite un espacio de direcciones de 2^32, es decir, 4GB. Para describir este espacio necesitamos una TP con 2^20 entradas, cada una de 4 B de tamaño.
- Es decir, la TP debería tener 4 MB de tamaño.
- La paginación simple requiere TP de un tamaño muy elevado lo que no permitiría de forma eficiente la existencia de muchos procesos.
- Solución: “Paginar la tabla de páginas”, es decir, aplicar paginación a la propia TP.

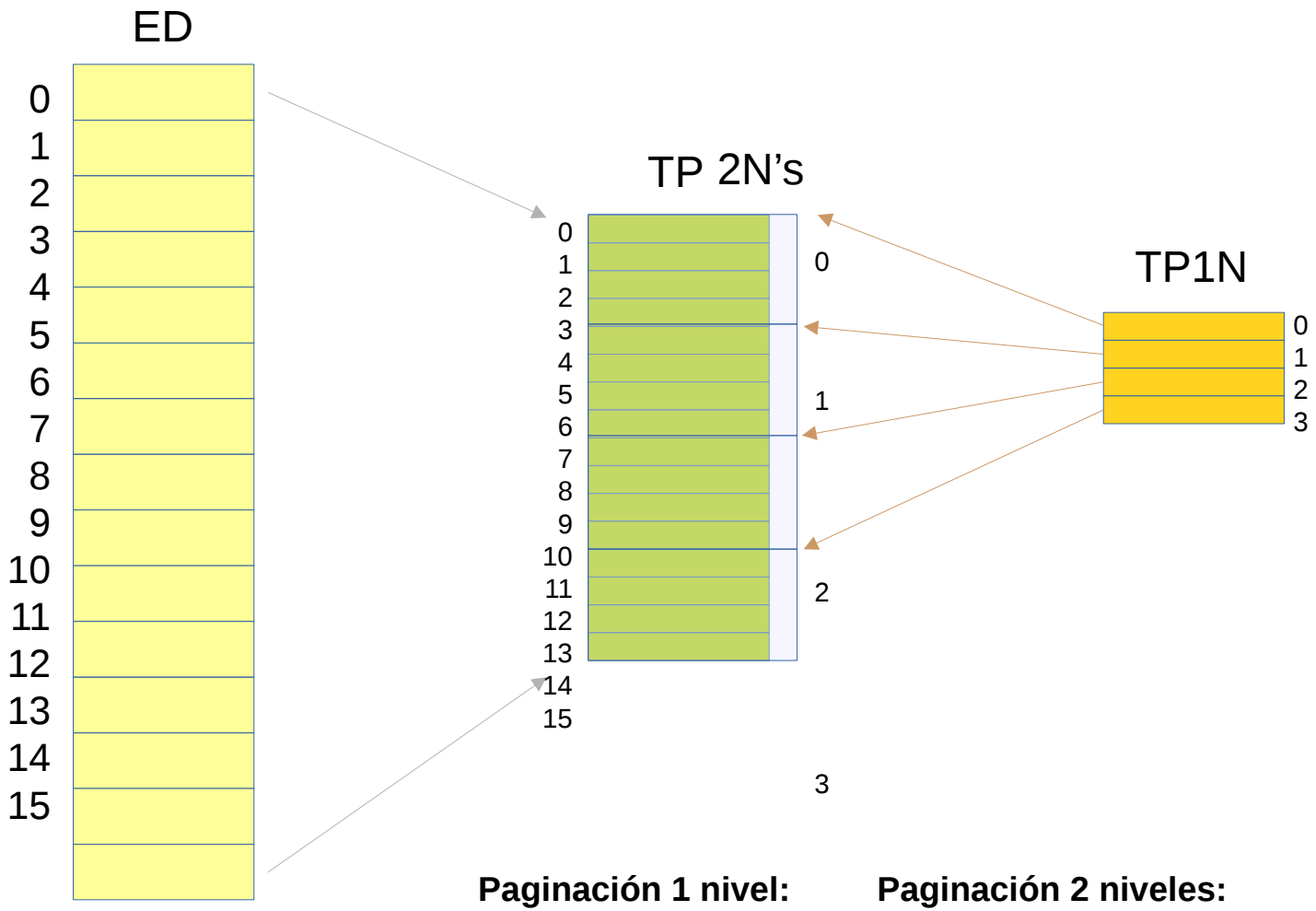
Paginación a 2 niveles: DL

▷ La MMU dividirá ahora las direcciones virtuales en 3 campos:

Paginación a 2 niveles: esquema



Paginación: 1 y 2 niveles



Paginación 1 nivel:

DL: | 4 | X |

Tamaño de página: 2^X

N.º de PTEs: $2^4 = 16$

Paginación 2 niveles:

2^X

$2^2 = 4$

DL: | 2 | 2 | X |

Paginación multinivel

▷ ¿Qué ganamos? Ahora podemos ahorrar:

- Tablas de 2 nivel (o mayores) que no son necesarias porque mapean regiones no válidas del ED del proceso.
- Si bien la TP de primer nivel debe estar en memoria principal, no ocurre así con las de segundo nivel y superiores.

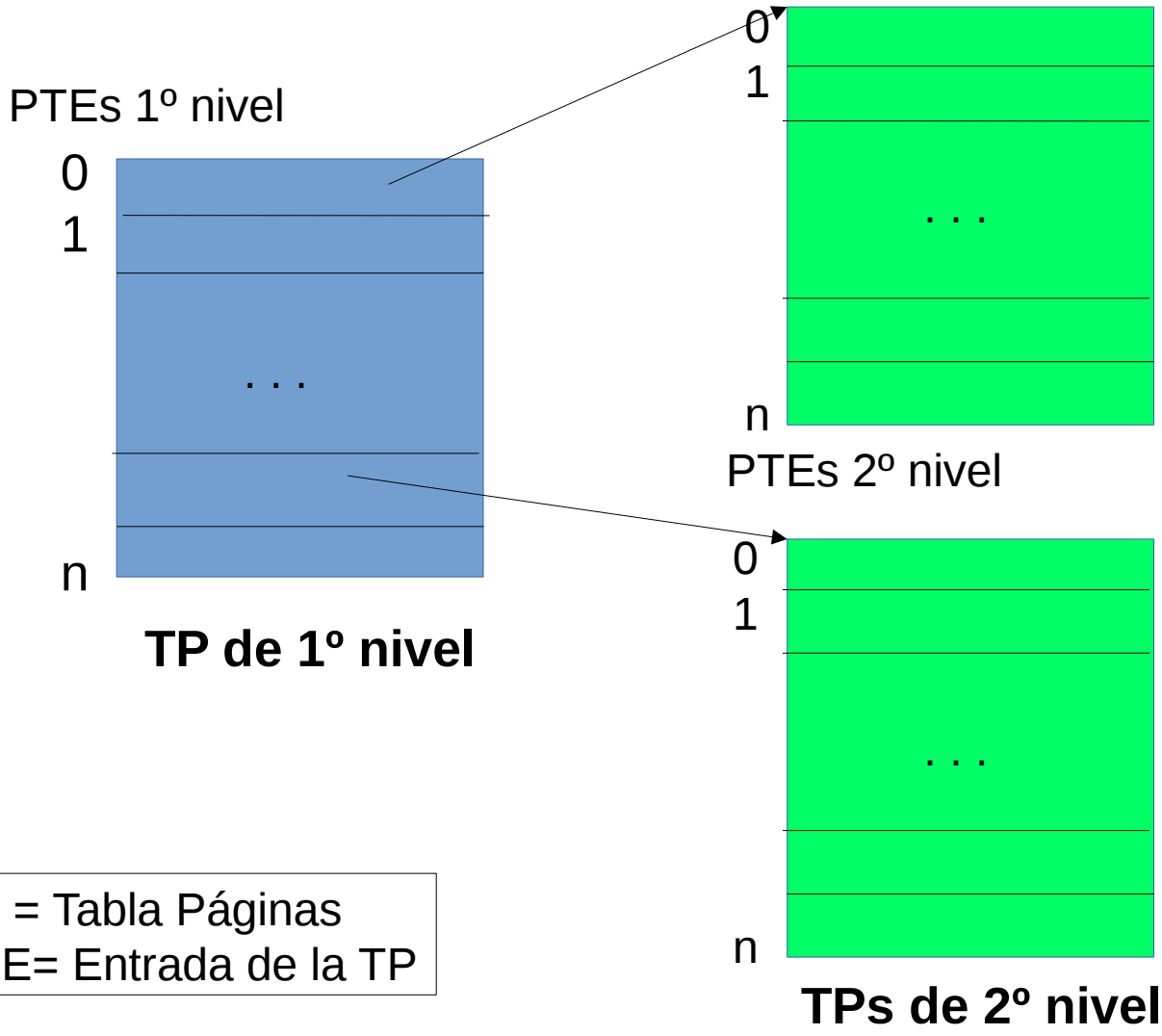
▷ Los procesadores actuales utilizan hasta cuatro niveles de paginación, por ejemplo, x64.

▷ Coste: Los TLBs mantienen un TAE razonable.

Paginación 2 niveles: ejemplo

- Sea una arquitectura que direcciona octetos (bytes) y con sistema de paginación a dos niveles. Una dirección lógica tiene la estructura siguiente: 10 bits para paginación a primer nivel, 10 bits para paginación a segundo nivel, y 12 bits para el desplazamiento de página (offset). Tenemos un proceso cuyo espacio de direcciones lógicas esta estructurado como sigue: 8 KB de código que comienza en la dirección lógica 0; una región de datos de 4 KB cuya dirección lógica de inicio es la siguiente a la última dirección del código; y una pila de 2 KB cuya dirección de base esta en la dirección 3GB.
- Se pide construir las tablas de páginas de 1º y 2º nivel para el citado proceso, suponiendo que el primer marco asignado al proceso es el marco 3 que contiene la primera página de código y el resto de páginas están asignadas en marcos contiguos

Ejemplo: estructura de las TPs



TP = Tabla Páginas
PTE= Entrada de la TP

Ejemplo: dirección virtual

▷ La distribución de una Dirección virtual lógica (DL):

▷

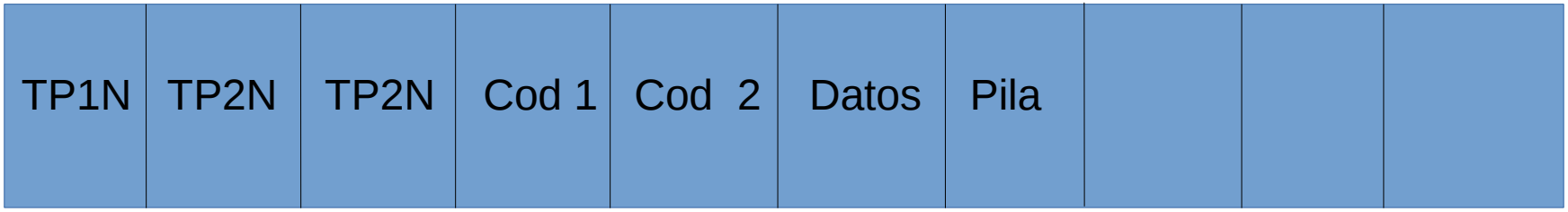
▷ De aquí, calculamos:

- Tamaño de las páginas a partir del desplazamiento de página: 2^14 = 4 K direcciones. Como cada dirección es a un octeto → tamaño página = 4 KB
- Nº de entradas de las TPs de primer y segundo nivel: 2^10= 1024 entradas cada TP
- En el dibujo anterior n = 1023.

Ejemplo: páginas y marcos

▷ De la descripción del espacio de direcciones para el proceso del enunciado:

- Código = 8 KB → 2 páginas a partir de la DL=0
- Datos = 4 Kb → 1 página después del código.
- Pila = 0,5 KB → 1 página con base en la DL=3 GB.
- El proceso esta cargado en memoria a partir del marco 3, por tanto la RAM queda:



Ejemplo: rellenando las TPs

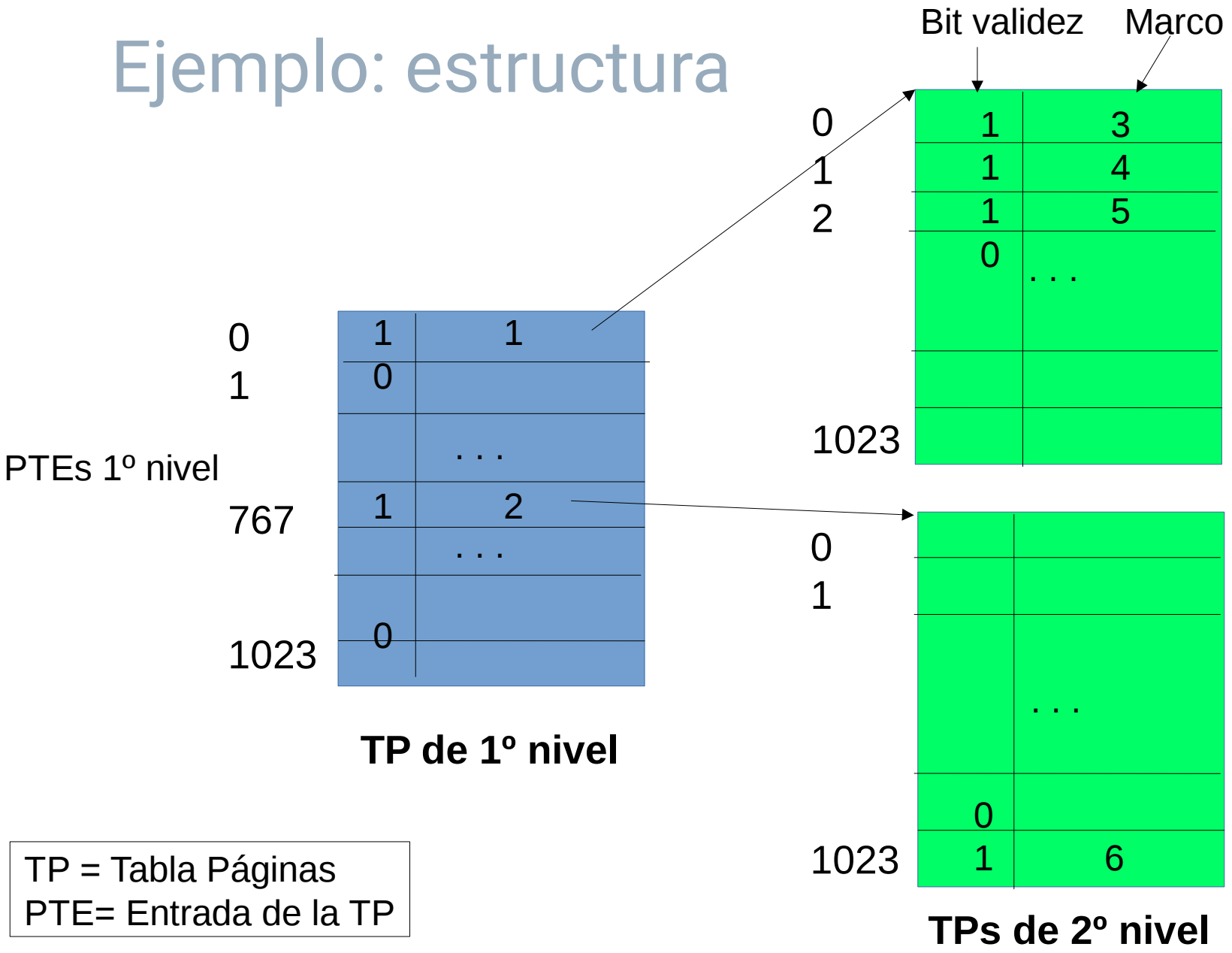
▷ Para rellenar las TPs, vamos calcular las PTEs en uso:

- *Código y datos*: van desde la dirección 0 a 12 KB -1, dado que con una PTE de 1º nivel puedo direcciónar 4MB (direcciona una TP de 2º nivel que permite direccionar 1024x4KB = 4 MB), la PTE de 1º nivel usada en la 0, que apunta al marco de la TP de 2º nivel, y las PTEs de 2º nivel 0,1 y 2 se rellenan con los marcos 3, 4 y 5 respectivamente.
- Calculemoslo:
  - 12k-1/4M → cociente=0 (entrada 0 de laPTE 1º nivel) → resto: 12K-1
  - Resto/4K-> cociente=2 (PTE 2) y resto = 4095 (offset)

Ejemplo: rellenando las Tps (ii)

- *Pila*: La base de la pila esta en la dirección 3GB-> necesito 3GB/4MB entradas de primer nivel = 768 entradas de 1º nivel → La PTE 767 apunta al marco que contiene la TP de segundo nivel. Para calcular que entrada uso para la página de pila en la TP de 2ºnivel, como 3GB%4MB=0 quiere decir que la TP de 2 nivel se utiliza completa, por tanto la página de pila esta en la TPE 1023 de la correspondiente tabla de 2ºNivel.

Ejemplo: estructura



TP = Tabla Páginas
PTE= Entrada de la TP

Ejemplo: traducción de direcciones

- Dadas la TPs podemos traducir direcciones como lo haría es sistema. Supongamos que deseamos traducir las direcciones:
- a) 5678
- b) 13631488
- Acceso a la TP de primer nivel:
- `Direccion_Lógica/tamaño_direccionado_con_una_PTE`
  → cociente= nº entrada usada
- → resto = desplazamiento en TP de 2º nivel.

Ejemplo: traducción de (a)

- 5678 / 4M → cociente=0
  - → resto 5678
  - Uso la entrada 0 de la TP 1º nivel, que como tiene el bit de validez a 1, podemos acceder a la TP de 2º nivel, de la cual debo usar la entrada dada por:
  - 5678 / 4x1024 = Cociente =1 y Resto = 1582 → Entrada 1 de la TP2ºnivel, que como tiene el bit de validez a 1, se puede utilizar y apunta al marco 4 y desplazamiento 1582
  - La dirección física será = 4x4096+1582 = 17966.

Ejemplo: traducción de (b)

- Al igual que el primer caso, vemos la entrada de la TP1N que debemos usar:
  - 13631488/4x1024x1024 → cociente=3
- Ahora la entrada 3, que no está en uso, tiene el bit de validez a 0, por lo que se produciría un excepción de página inválida.

2.

Paginación por demanda

Conceptos y operatoria de la paginación por demanda

Paginación por demanda

> Para reducir la contención de memoria podemos cargar en memoria solo las páginas de un proceso que sean necesarias en un momento dado, no todas.
> Tiene como ventajas:
    Reducir las E/S
    Menos memoria en uso
    Respuesta más rápida
    Más procesos en ejecución
> ¿Cómo sabemos cuando se necesita una página? Cuando el proceso referencia una instrucción o dato de los que contiene:
    - Si la referencia es inválida, se aborta la ejecución
    - Si es válida, se trae a memoria desde disco

Falta de página

▷ Vamos a sobrecargar la semántica del *bit de validez* presente en cada PTE:

- Bit a 1: la página esta en memoria.
- Bit a 0: la página “no esta en memoria”.

▷ Al iniciar un proceso, todos los bits de validez de las PTEs estan a cero. Durante la traducción de direcciones, cuando se referencia una página, la MMU comprueba el bit de validez de la PTE correspondiente, y, si está a cero, va a producir una excepción denominada *falta de página*. Si no, es decir, esta a 1, se realiza la traducción.

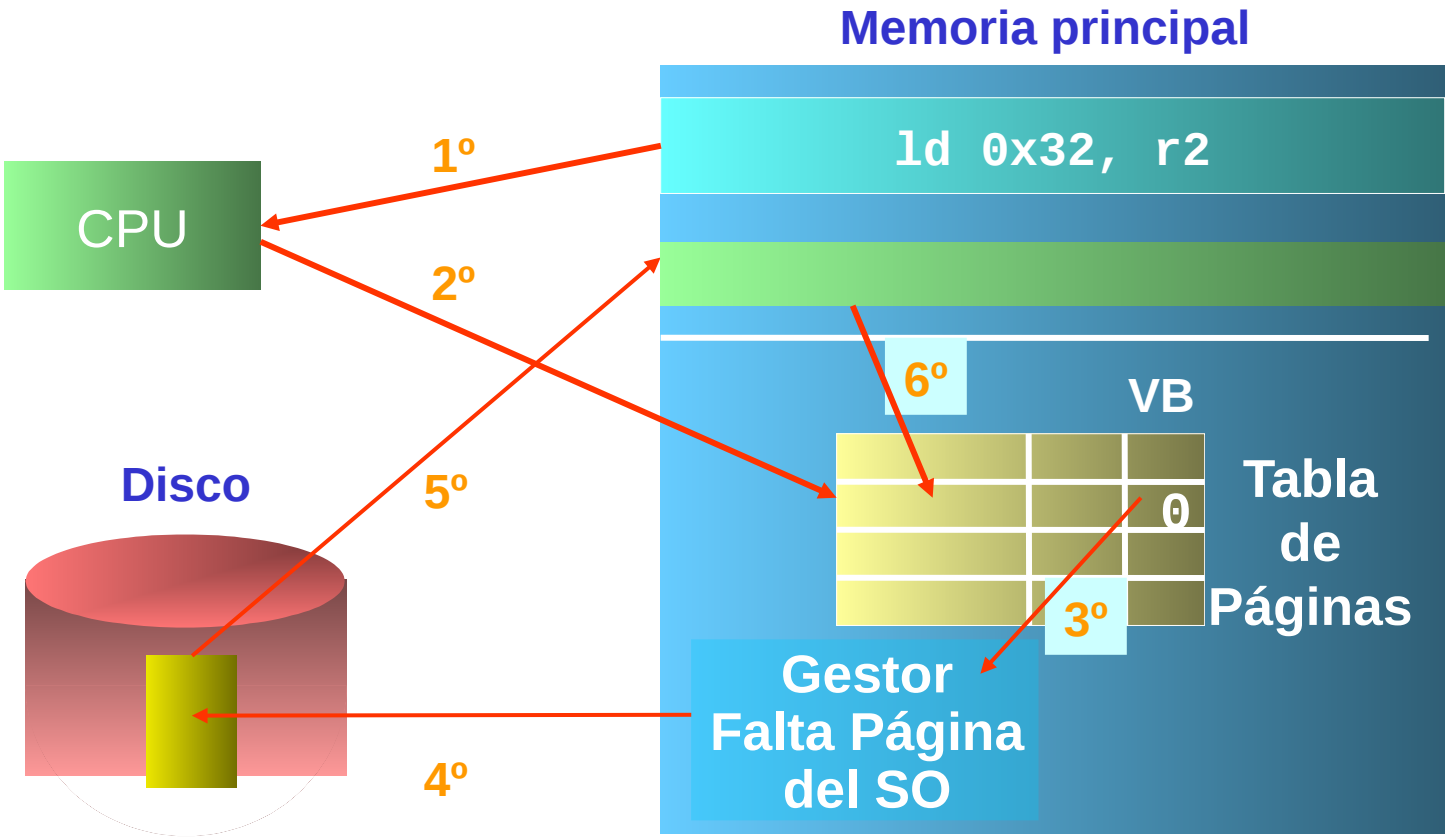
Rutina excepción “falta de página”

▷ La rutina de la excepción falta de página:

1) Mira si la referencia es válida, si no lo es aborta al proceso.
2) Si es válida pero no esta en memoria:
3) Obtiene un marco vacío.
4) Carga la página faltante en el marco libre.
5) Activa el bit de validez.
6) Rearranca la instrucción que produjo la falta.

▷Si la página esta en memoria, pero la traducción no es válida, entonces se reasigna la página.

Gestión Falta de página: esquema



Sustitución de páginas

> Podemos reducir la sobre-asignación de memoria si la RSE de falta de página incluye la *sustitución de páginas:*

"encontrar una página asignada pero que no esta actualmente en uso para sacarla y poder usar el marco."

> La sustitución de páginas completa la separación entre memoria lógica y física.

> Rendimiento: el algoritmo de sustitución que se utilice debería producir el mínimo número de faltas de páginas, es decir, debemos retirar páginas que tengan la menor probabilidad de ser usadas.

Algoritmos de sustitución de páginas

> Existen numerosos algoritmos de sustitución de páginas, pero los sistemas en producción utilizan:

- Linux – LRU (Least Rencently Used).
- Windows - monoprocesadores, Reloj (LRU)
  - multiprocesadores, aleatorio.
- OS X - Segunda oportunidad.

Bit sucio

> El uso de un bit adicional en las PTEs, el *bit sucio* (o de modificación), reduce la sobrecarga de transferencia de algunas páginas:

- Aquellas que no se han modificado no hay que salvarlas ya que tenemos una copia en el archivo ejecutable.
- Solo escribimos en disco las páginas que han sido modificadas.

3.
Gestión de memoria en Linux

Cómo gestiona Linux la memoria

Niveles de gestión de memoria

> Existen dos niveles con requisitos diferentes:

- *Gestor de memoria de SO*:
  - Asigna porciones de memoria a los procesos (es no crítica).
  - Asigna memoria a los subsistemas del SO (crítica).
- *Gestor de memoria de procesos*: gestiona dinámica de los procesos (malloc, free, ...)

Gestor de memoria en Linux

> Tiene dos interfaces:

Interfaz de llamadas al sistema: interfaz de usuario

- `malloc/free/..`
- `mmap/...`
- `mlock/munlock/...`
- `swapon/swappoff`

Interfaz intra-kernel: interfaz de mm al resto del kernel.

- `kmalloc/kfree`
- `verify_area`
- `get_free_page/ free_page`
- ...

Elementos de gestión

> La implementación Linux de gestión de memoria cubre:

- Memoria kernel:
  - *Distribuidor sistema amigo* para asignar grandes bloques contiguos de memoria.
  - *Distribuidores tableta* (slab, slob y slub) para asignar memoria inferior a una página.
  - *Mecanismo vmalloc()* para asignar bloques no contiguos de memoria.
- Memoria de usuario:
  - Mecanismo de construcción y gestión de los *espacios de direcciones* de los procesos.

Gestión de memoria para procesos

> La asignación de memoria dinámica a los procesos de usuario tiene requisitos diferentes a la propia del kernel:

- La peticiones de procesos no se consideran urgentes: el kernel intenta diferir su asignación, ya que la solicitud no indica utilización inmediata.
- Los procesos de usuario no son confiables: el kernel debe estar preparado para atrapar errores de direccionamiento.
- Cada proceso tiene su propio espacio de direcciones separado del resto de procesos.
- El kernel puede añadir/suprimir rangos de direcciones lineales.

Regiones de memoria

> El kernel representa un intervalo de direcciones lineales contiguas con el mismo tipo de protección mediante un recurso denominado *región de memoria*.
> Estas se caracterizan por su dirección de inicio, su longitud y los derechos de acceso. Por eficiencia, estas tienen un tamaño múltiplo del tamaño de página (4KiB en I32).
> Por ejemplo:

- región de código: permisos lectura-ejecución
- región de datos: permisos de lectura-escritura
- región de pila: lectura-escritura, crecimiento
- ...

Regiones de memoria

> Las tablas de páginas NO son adecuadas para representar espacios de direcciones grandes, especialmente si son dispersos → vamos a superponer otra gestión de memoria sobre la paginación.
> Linux representa cada región de memoria con una estructura denominada *vm-área* (*virtual memory area*).
> Las vm-áreas contienen la información necesaria para poder establecer la traducción de una dirección que la TP no pueda realizar.
> El ED de un proceso se representa como una lista de vm-áreas. Si el número de vm-áreas el muy elevado, las vm-áreas se organizan en un árbol rojo-negro.
> Las vm-áreas no tienen contador de referencias, por lo que solo pueden pertenecer a un proceso.

Elementos de un vm-área

> Los principales componentes son:

- *Rango de direcciones* – direcciones de inicio/fin de la región.
- *Indicadores VM* – Palabra que contiene, entre otras cosas, los indicadores `VM_READ`, `VM_WRITE`, `VM_EXEC`, que controlan si un proceso puede leer, escribir, ejecutar la memoria virtual proyectada por el vm-área.
- *Información de enlace* – Punteros a la lista de vm-áreas, subárboles derecho e izquierdo.
- *Operaciones VM* y *datos privados* – contiene el puntero de operaciones VM que apunta a funciones, y sus datos privados, a invocar cuando se producen ciertos eventos relacionados por memoria virtual, p. ej. una falta de página.
- *Información de archivo proyectado* – Si el vm-área proyecta un archivo, este componente almacena el puntero a archivo y el desplazamiento necesario para localizar los datos.

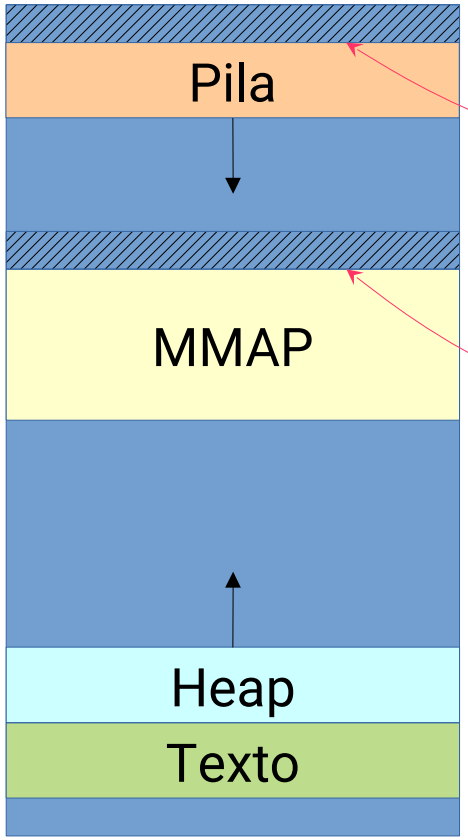
Operaciones MV

> El puntero de operaciones VM proporciona a las vm-áreas características de orientación objetos: cada tipo de vm-área puede tener diferentes manejadores.
> Cualquier objeto (sistema de archivos, dispositivo de caracteres, ..) proyectado en memoria de usuario (`mmap`) puede suministrar sus propias operaciones. Algunas de ellas son:

- `open()` / `close()` – crear/destruir un vm-área.
- `fault()` – manejador de la falta de página que no esta en TP.

Espacio virtual en x86

TASK_SIZE
STACK_TOP – randomized_variable



Ramdom offset
mm->mmap_base

El agujero aleatorio dificulta que programas maliciosos exploten agujeros de seguridad provocados por “buffer overflow”.

Nos asegura que la pila no colisiona con la región proyectada

0x0804 80000
0

Llamadas relacionadas con regiones

> Las llamadas al sistema relacionadas con la creación, destrucción o modificación de regiones de memoria son:

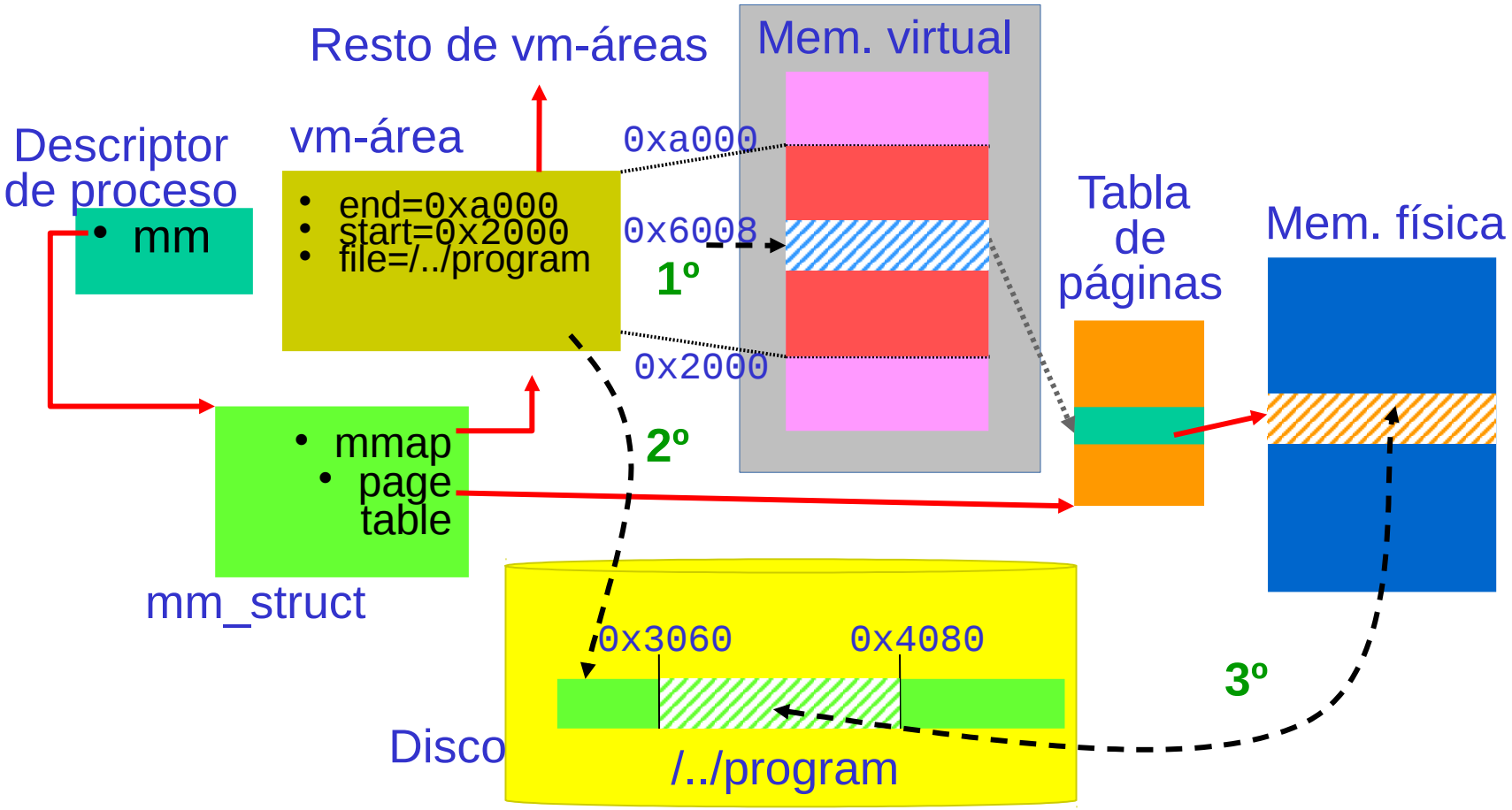
Llamada al sistema	Descripción
`brk()`	Cambia el tamaño del *heap* de un proceso
`exec()`	Carga un nuevo programa ejecutable
`exit()`	Termina el proceso actual
`fork()`	Crea un nuevo proceso
`mmap()`	Crea una proyección de memoria para un archivo
`munmap()`	Destruye una proyección de memoria ara un archivo
`shmat()`	Crea una región de memoria compartida
`shmdt()`	Destruye una región de memoria compartida

Descripción del ED de un proceso

> Viene descripto por la mm_struct:

```
struct mm_struct {
    struct vm_area_struct * mmap, . . /* list of VMAs y ... */
    . . .
    unsigned long task_size;          /* size of task vm space */
       . . .
    pgd_t * pgd;
    atomic_t mm_count;                /* How many references to ... */
    int map_count;                    /* number of VMAs */
    . . .
    unsigned long hiwater_rss;      /*High-watermark of RSS usage */
    unsigned long hiwater_vm;       /*High-water virtual memory usage */
    unsigned long total_vm;         /*Total pages mapped */
    unsigned long locked_vm;        /*Pages that have PG_mlocked set */
    unsigned long pinned_vm;        /*Refcount permanently increased */
    unsigned long shared_vm;        /*Shared pages (files) */
    . . .
    unsigned long nr_ptes;          /* Page table pages */
    unsigned long start_code, end_code, start_data, end_data;
    unsigned long start_brk, brk, start_stack;
    unsigned long arg_start, arg_end, env_start, env_end;
       . . .
}
```

Ejemplo de vm-área

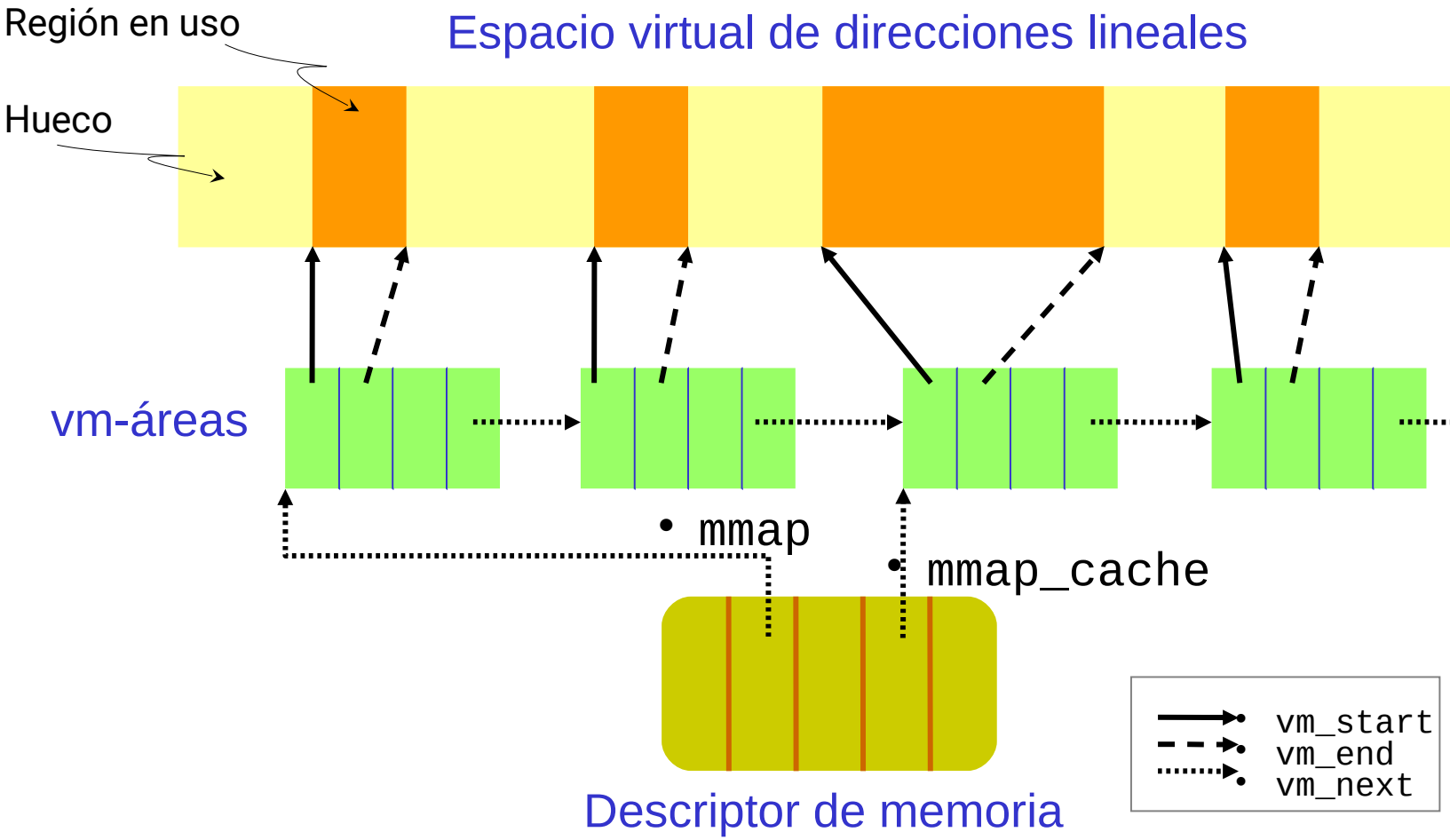


Ejemplo: explicación

> Suponemos que un proceso proyecta los primeros 32KB (8 páginas) de un archivo en la dirección virtual 0x2000.
> Los pasos del dibujo anterior:

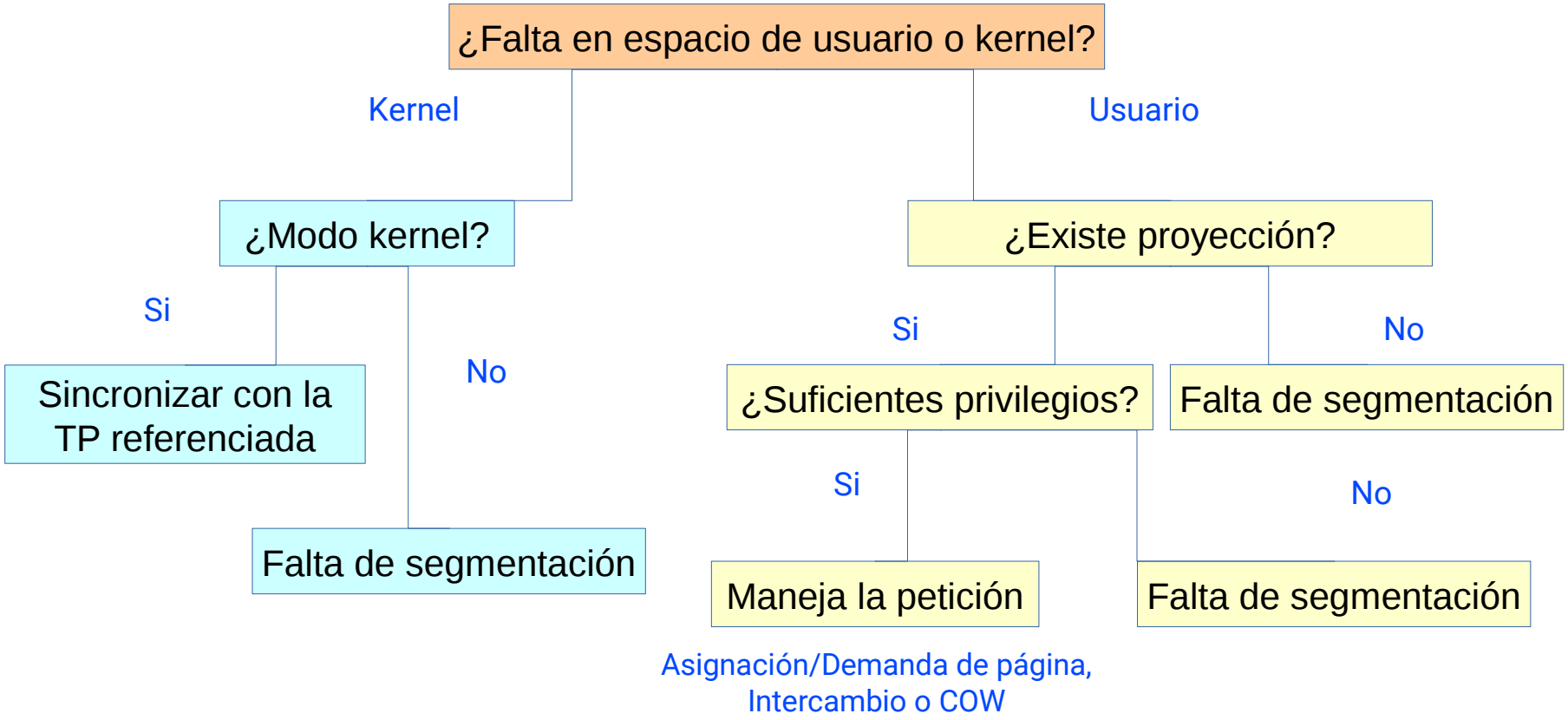
- Intenta leer la dirección 0x6008. Linux localiza la vm-área que cubre la dirección que provoca la falta (página 3)
- Linux inicia la transferencia de la página 3
- Linux copia el contenido del archivo en el marco adecuado y actualiza la tabla de páginas. El proceso reanuda su ejecución.

Descripción de un ED



Gestión de la falta de página

> Esquema simplificado del gestor de falta de página:



Mecanismo COW

> El mecanismo *COW* (*copy-on-write*) permite reducir:

- Las operaciones de copia en memoria.
- El número de copias de marcos.

> El mecanismo permite que varios procesos compartan un marco de memoria. Si alguno de los procesos intenta escribir en la página compartida COW, el kernel asigna una nueva copia de la página al proceso escritor y realiza la escritura.

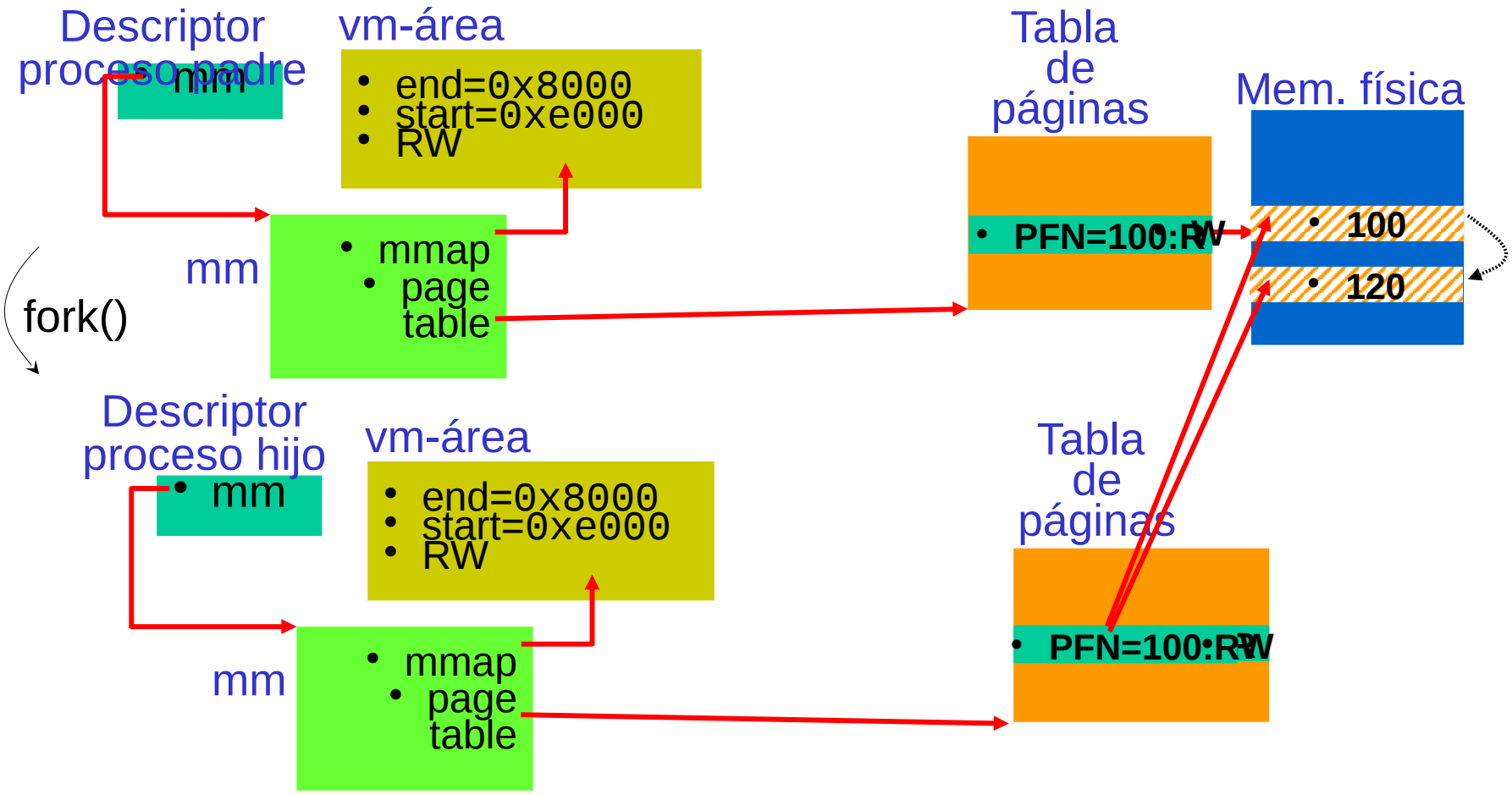
> Implementación: La página compartida COW se marca como de solo lectura pero la vm-área correspondiente permite la escritura.

Ejemplo de COW

(a) Antes de fork()

(b) Después de fork()

(c) Hijo escribe



Gestión de pila

> La rutina de gestión de la falta de página permite que el sistema operativo gestione la asignación de memoria para la pila de usuario.
> Supongamos que el usuario intenta empujar en la pila dato que requiere nueva asignación de memoria. La rutina de servicio de falta de página detecta que se esta intentando escribir en una dirección inicialmente no válida pero muy próxima a una región que tiene permiso de escritura y es de tipo `VM_GROWSDOWN`, entonces asigna una página adyacente (virtualmente) a las páginas de la pila.

Demanda de página

> Como hemos visto, la demanda de página es la técnica mediante la cual la asignación dinámica de memoria se pospone hasta que el proceso direcciona una página y genera la excepción “falta de página”.

> Las razones por la que una página no esta presente en memoria principal:

1) La página nunca ha sido accedida por el proceso: su PTE esta rellena a ceros.
2) La página ha sido accedida pero su contenido se ha sacado a disco: su PTE no esta rellena con ceros y el indicador *Present* esta limpio.

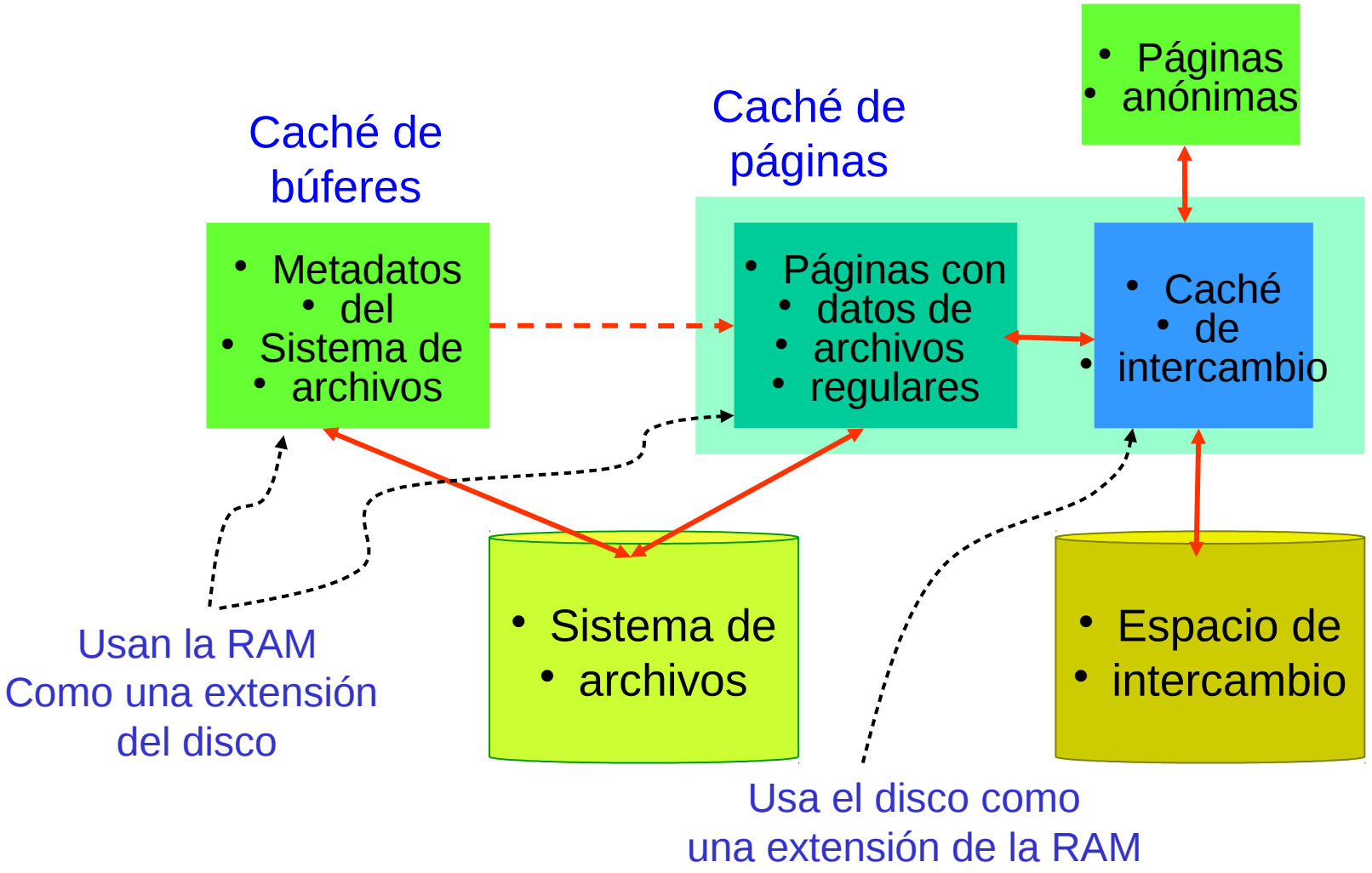
Demanda de página (cont.)

> En el caso 1, hay dos formas de cargar la página dependiendo de si la página esta proyectada en un archivo de disco.

Esto se determina consultando `vma->vm_ops->nopage`:

- Si es no nulo: apunta a la función que carga la página.
- Si es nulo: se trata de una proyección anónima (no proyectada a ningún archivo).
  - Para la proyección anónima y de cara a alcanzar el nivel C2 de seguridad, se utiliza la página cero (página asignada estáticamente en la inicialización y almacenada en el sexto marcho de página). Esta página no se puede escribir, de cara a disparar el mecanismo COW cuando se intenta.

Cachés del sistema



Cachés (cont.)

> **Caché de páginas** – Todos los accesos a archivos a través de read(), write() o mmap() se realizan a través de ésta caché.
> **Caché de búferes** – Almacena los metadados de los archivos para hacer más rápida su recuperación.
> **Caché de intercambio** – Evita condiciones de carrera entre procesos que intentar acceder a páginas que han sido sacadas de RAM: una página compartida puede estar presente para un proceso y sacada de memoria para otro.