

1.

Paginación multinivel

Justificación e implementación

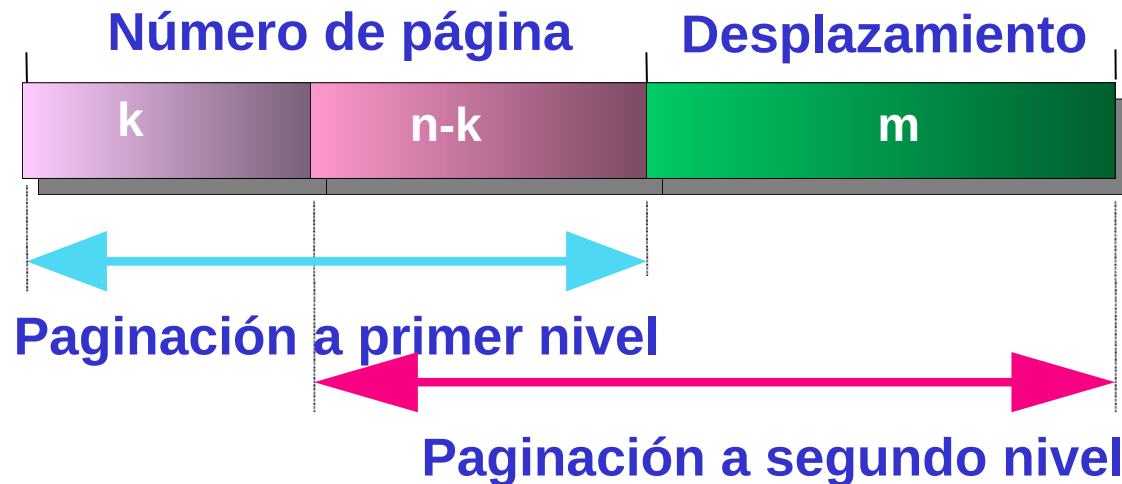
Paginación multinivel: justificación

- ▷ 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: “Pagar la tabla de páginas”, es decir, aplicar paginación a la propia TP.

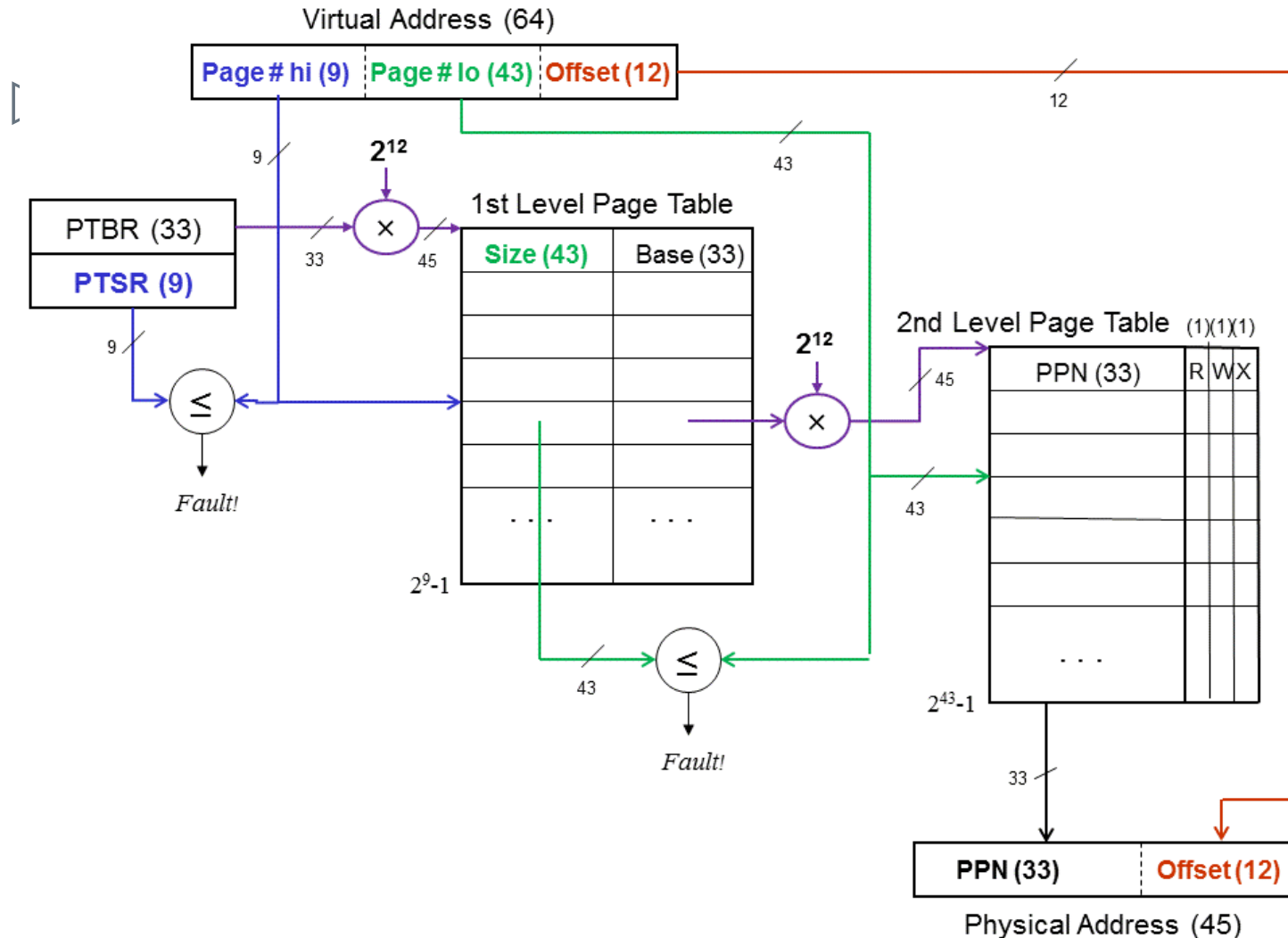
Paginación a 2 niveles: DL

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

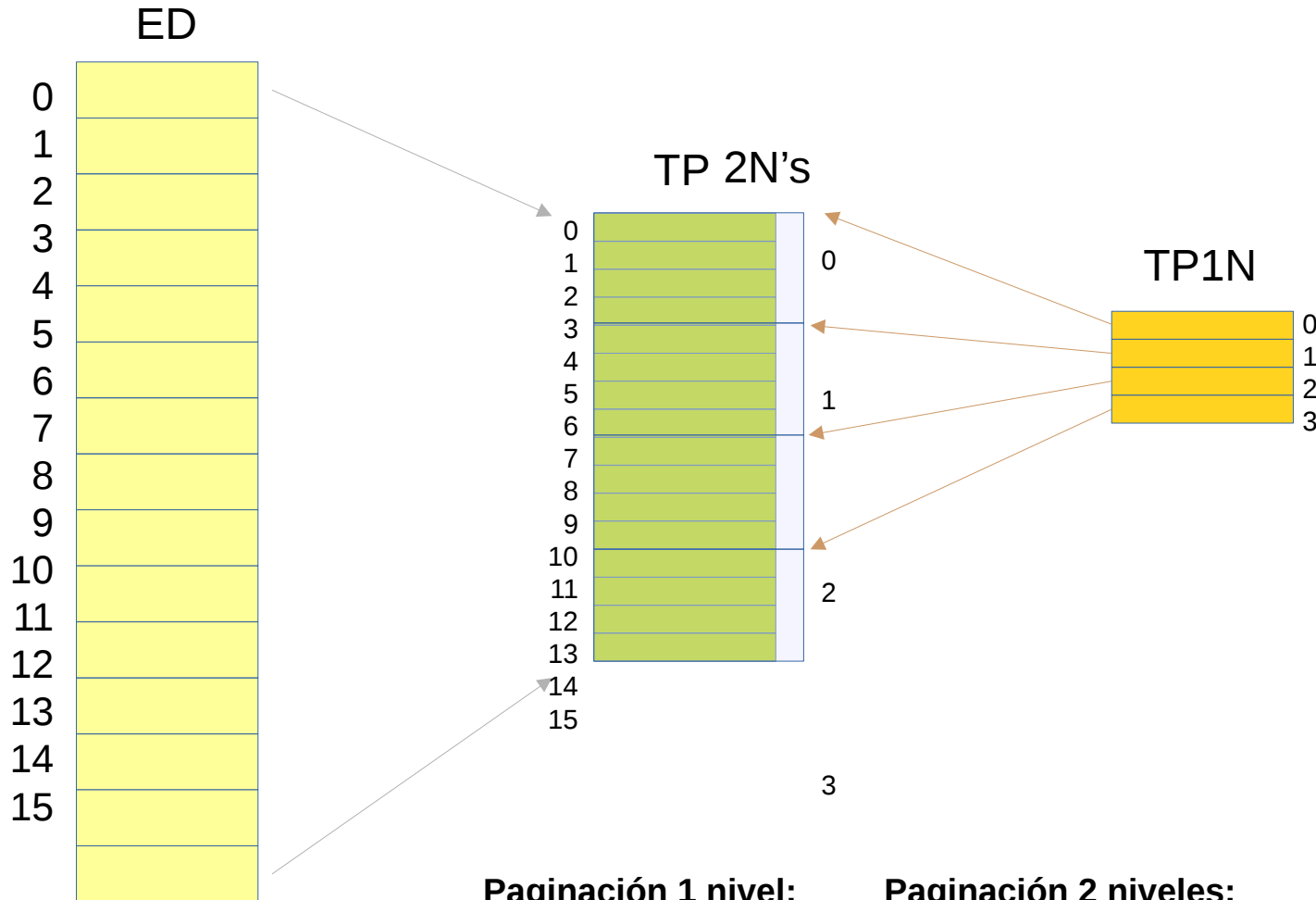
Dirección virtual de n bits



Paginación a 2 niveles: esquema



Paginación: 1 y 2 niveles



Paginación 1 nivel:

Paginación 2 niveles:

DL:

4	X
---	---

Tamaño de página: 2^X
N.º de PTEs: $2^4 = 16$

2^X
 $2^2 = 4$

DL:

2	2	X
---	---	---

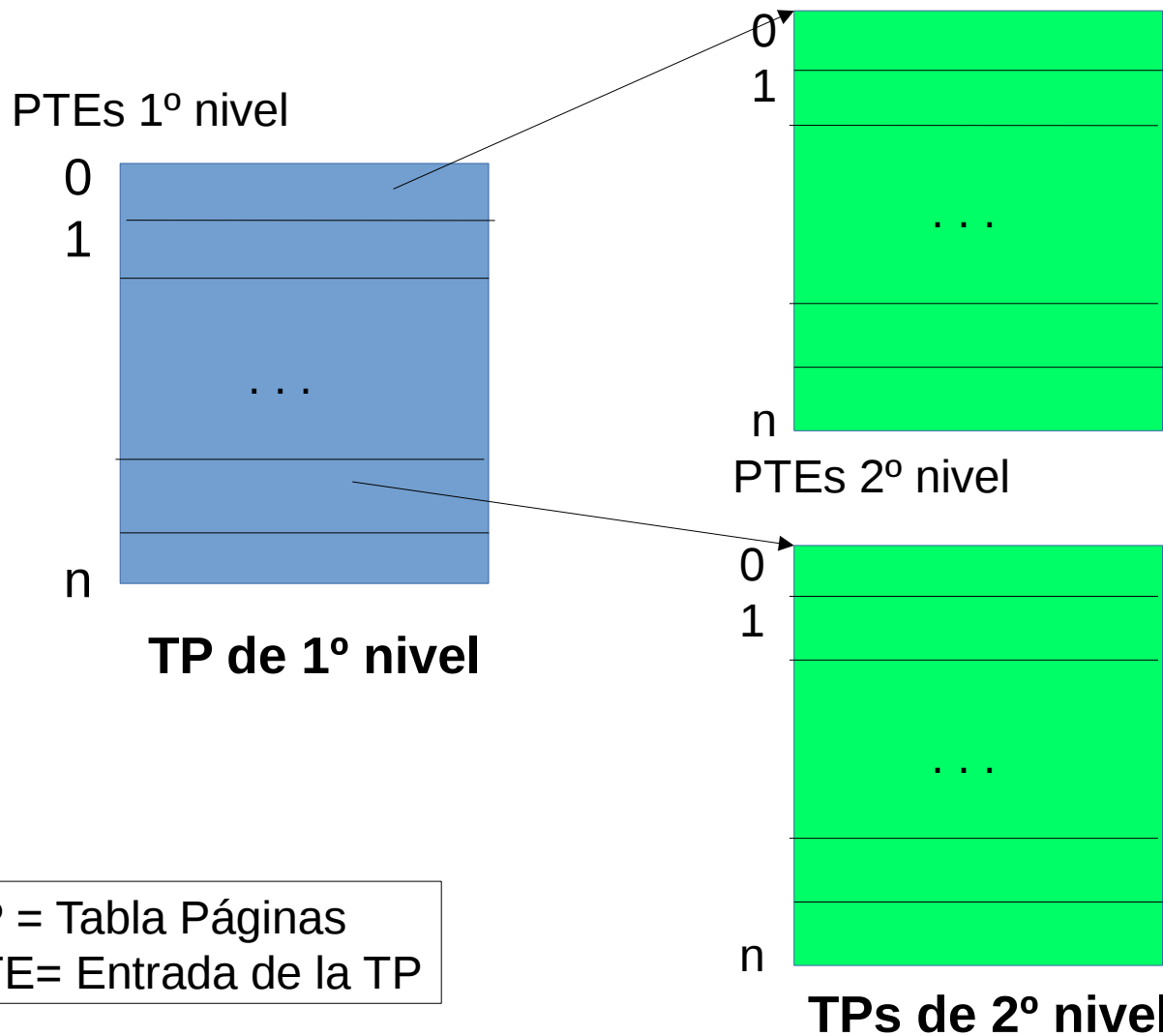
Paginación multinivel

- ▷ ¿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 está 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 está 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



Ejemplo: dirección virtual

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



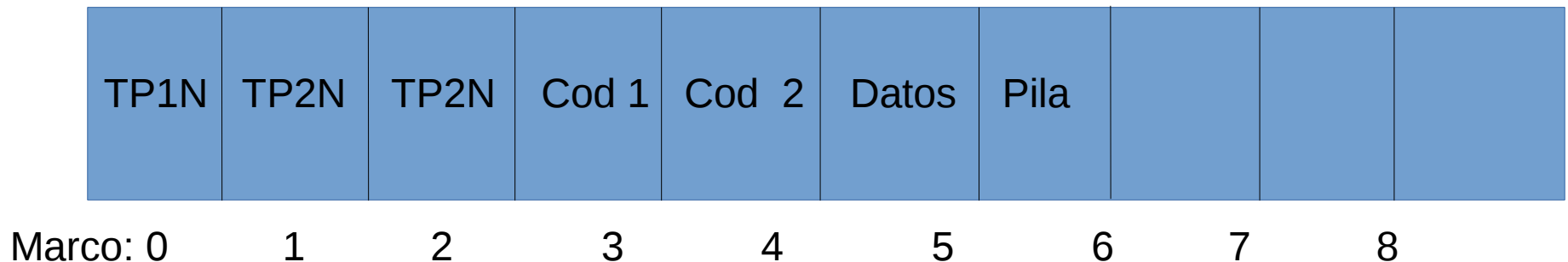
▷

▷ De aquí, calculamos:

- Tamaño de las páginas a partir del desplazamiento de página:
 $2^{14} = 4 \text{ K direcciones}$. Como cada dirección es a un octeto
→ tamaño página = 4 KB
- N° 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:



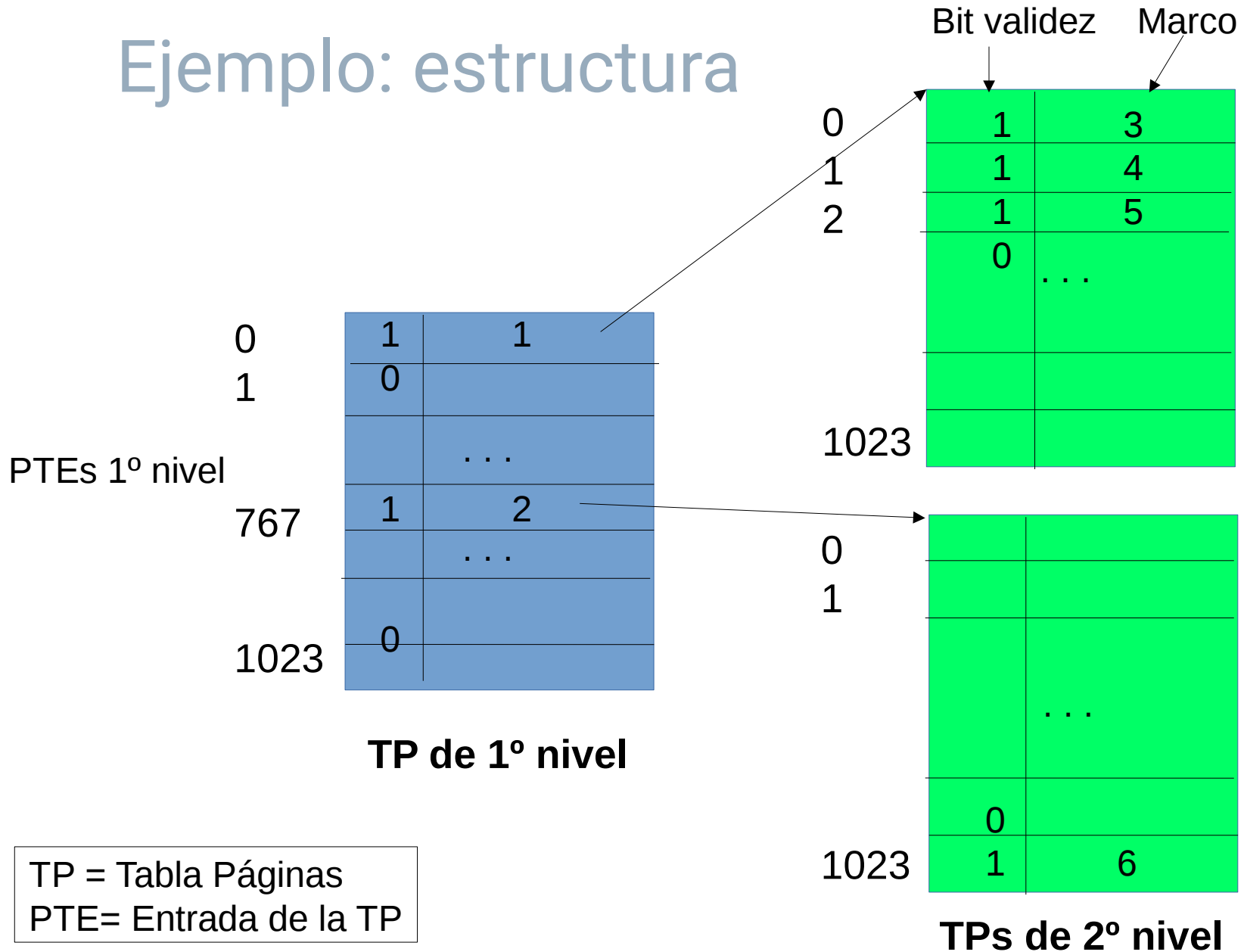
Ejemplo: rellenando las TPs

- ▷ 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 direccionar 4MB (direcciona una TP de 2º nivel que permite direccionar $1024 \times 4\text{KB} = 4\text{ MB}$), 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:
 - $12\text{k}-1/4\text{M} \rightarrow \text{cociente}=0$ (entrada 0 de la PTE 1º nivel) \rightarrow resto: 12K-1
 - $\text{Resto}/4\text{K} \rightarrow \text{cociente}=2$ (PTE 2) y resto = 4095 (offset)

Ejemplo: rellenando las Tps (ii)

- *Pila*: La base de la pila esta en la dirección 3GB-> necesito $3\text{GB}/4\text{MB}$ 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 2º nivel, como $3\text{GB} \% 4\text{MB} = 0$ 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 2º Nivel.

Ejemplo: estructura



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)

- $5678 / 4M \rightarrow \text{cociente}=0$
 - $\rightarrow \text{resto } 5678$
- 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:
- $5678 / 4 \times 1024 = \text{Cociente} = 1$ y $\text{Resto} = 1582 \rightarrow$ Entrada 1 de la TP 2º nivel, que como tiene el bit de validez a 1, se puede utilizar y apunta al marco 4 y desplazamiento 1582
- La dirección física será $= 4 \times 4096 + 1582 = 17966$.

Ejemplo: traducción de (b)

- Al igual que el primer caso, vemos la entrada de la TP1N que debemos usar:
 - $13631488 / 4 \times 1024 \times 1024 \rightarrow \text{cociente} = 3$
- 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

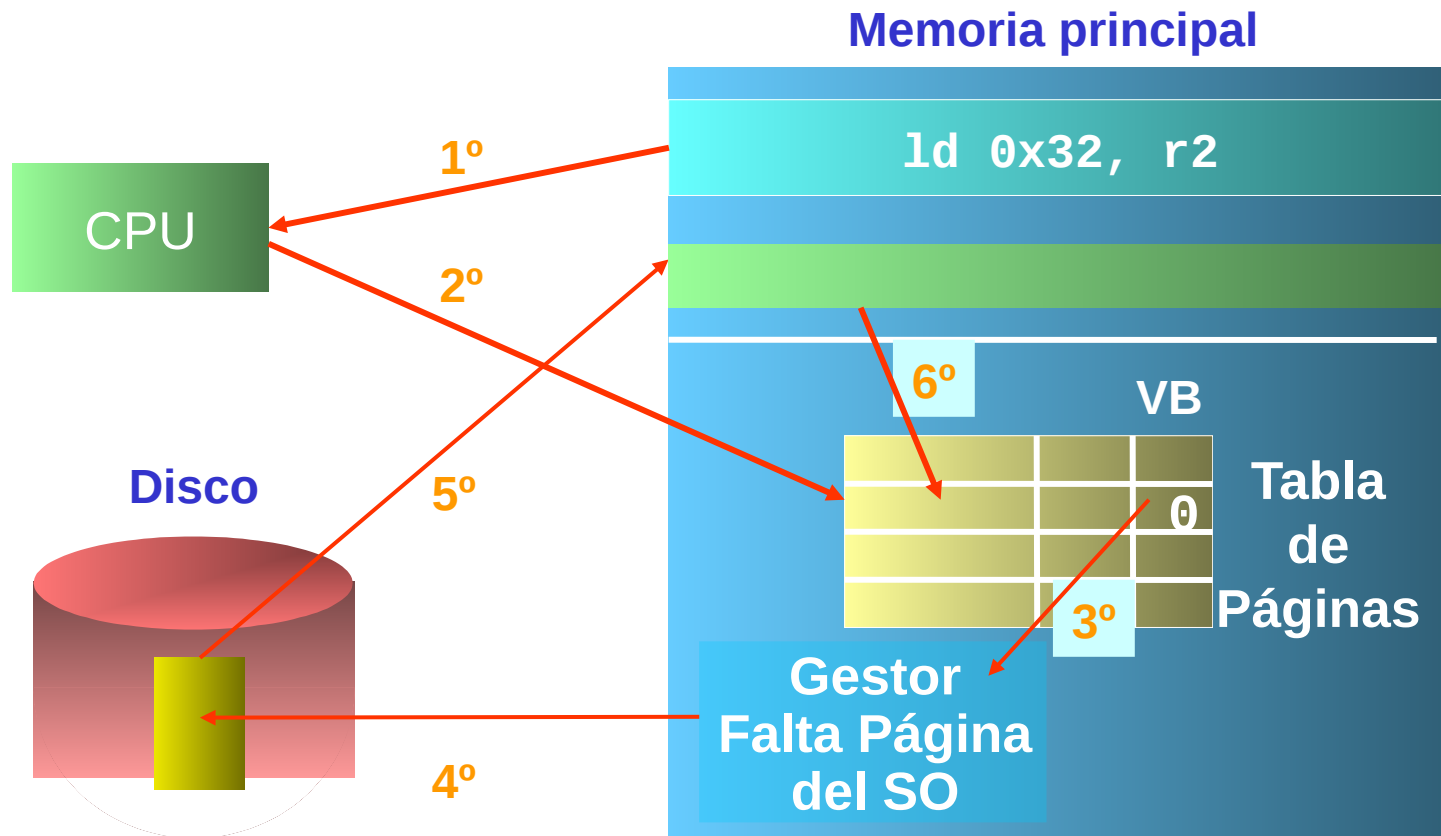
Falta de página

- ▷ 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 Recently 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.

Gestión de memoria para procesos

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

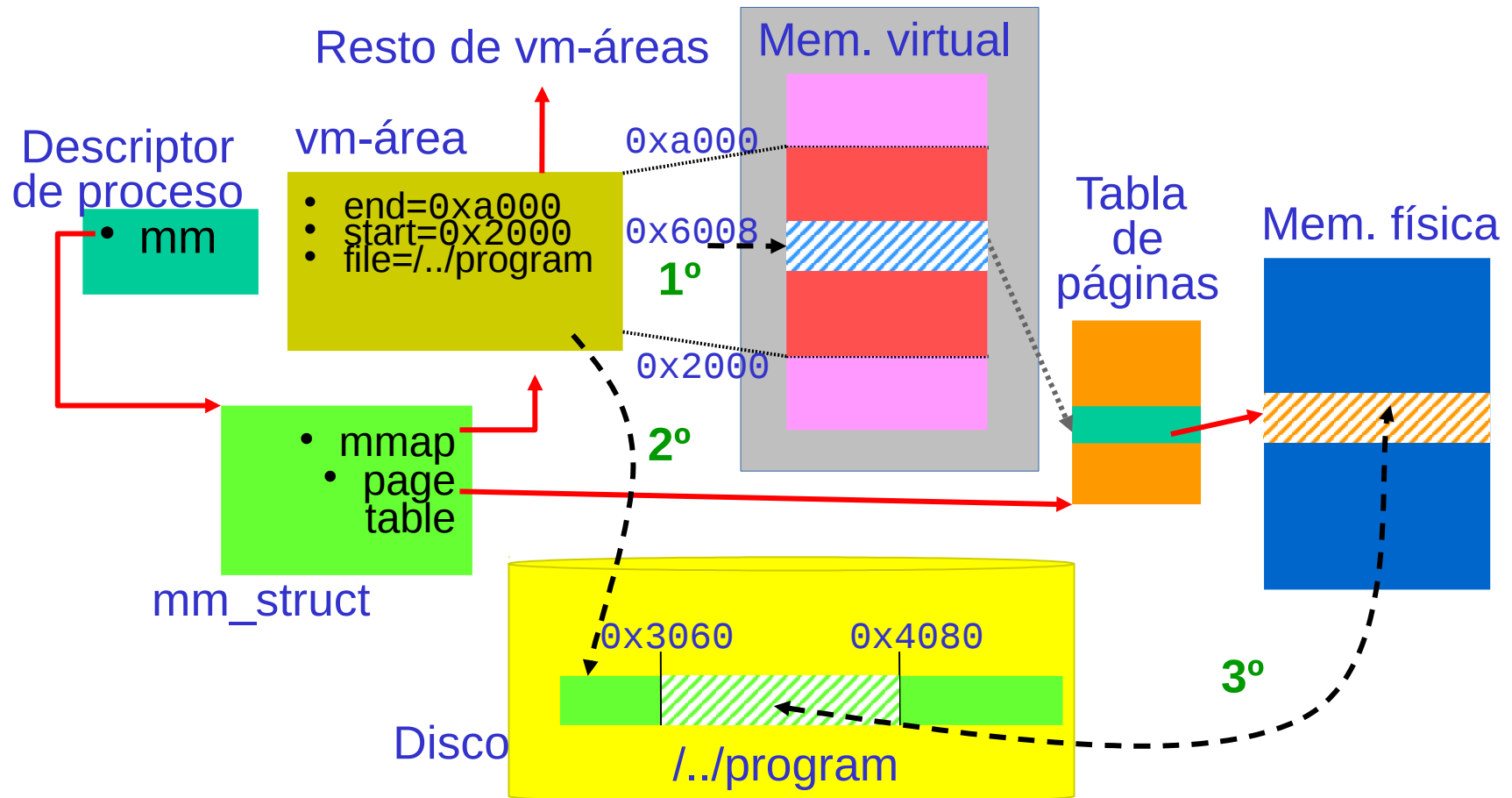
- > El puntero de operaciones VM proporciona a las vm-áreas características de orientación objetos: cada tipo de vm-área puede tener diferentes manejadores.
- > Cualquier objeto (sistema de archivos, dispositivo de caracteres, ..) proyectado en memoria de usuario (mmap) puede suministrar sus propias operaciones. Algunas de ellas son:
 - `open() / close()` – crear/destruir un vm-área.
 - `fault()` – manejador de la falta de página que no esta en TP.

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

