Administración de memoria

-Motivación/Objetivos
-Memoria Virtual
-Implementación de memoria virtual
-Políticas de asignación de memoria

Motivación/Objetivos

1.Realojamiento

2.Protección

3.Compartición

4.Intercambio de memoria: swapping

1)Realojamiento

- Imposible saber de antemano que otros procesos se ejecutarán al mismo tiempo.

- El programador no puede hacer referencias a posiciones de memoria absolutas

2)Protección

- Un proceso no debe tener acceso a la memoria de otro
    - TODOS los accesos a memoria se deben checar en tiempo de ejecución, para asegurarse de que se refieren al espacio de memoria del proceso actual

3)Compartición

- A veces hace falta compartir memoria entre procesos.
    - Ejemplo: 58 instancias del mismo programa pueden convenientemente compartir una sola copia del programa
    - Ejemplo 2: Consumidores/Productores deben compartir un buffer

4)Intercambio de memoria: swapping

- La RAM es típicamente pequeña: conviene usar discos para “extender” la memoria

- Se puede hacer a mano (overlay programming)
  - Complicado, Propenso a errores

- O con ayuda de la computadora (Memoria de intercambio)

Memoria Virtual

Memoria Virtual

- Mecanismo de traducción de direcciones (mapeo de direcciones)
- Distinción entre
    - direcciones de programa (programador, procesador), y
        - direcciones de memoria (hardware)
- Espacio de direcciones *virtuales*, Espacio de memoria *física*
    - *f: D -> M*

Espacio de direcciones virtual

- No necesariamente lineal

- Puede ser mayor, menor, o igual que el espacio de memoria

- El programador “ve” y “usa” una memoria virtual, diferente a la memoria física.

Implementación de memoria virtual

Registros base y límite
Paginación
Segmentación

Registros base y límite

Registros base y límite

- El registro base contiene la menor dirección accesada por el proceso
  - Todo acceso a la memoria de este proceso es relativo a esa dirección base
    - $f(a) = B + a$
- La relocalización es trivial
- La protección se lleva a cabo limitando la máxima dirección referenciable: registro límite
- Ayuda a cubrir los requisitos de relocalización y protección

Motivación/Objetivos

1.Realojamiento

2.Protección

3.Compartición

4.Intercambio (swapping)

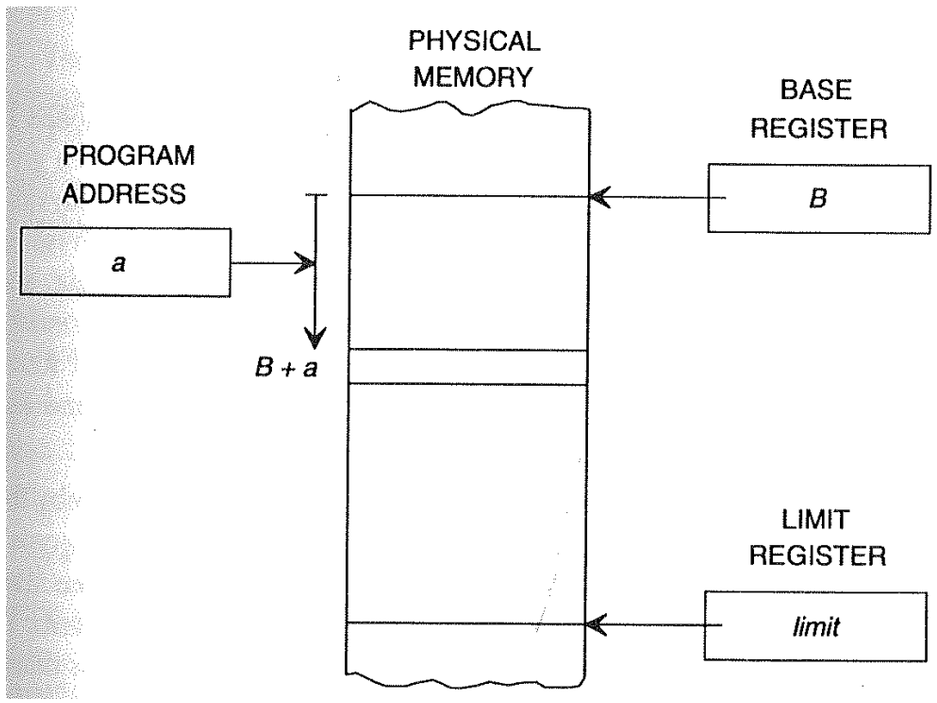
Registros base y límite (Cont.)

1)si a < 0: violación de memoria

2)a’ = B + a

3)si a’ > límite: violación de memoria

4)a’ es la dirección física solicitada



- Notemos que:
  - el espacio de direcciones es lineal
  - el espacio de direcciones es menor o igual que el espacio de memoria
  - => No hay 3)compartición

Una variación más eficiente:

- base + límite

Vs.

base + tamaño

1)si a < 0: violación de memoria

2)a’ = B + a

3)si a’ > límite: violación de memoria

4)a’ es la dirección física solicitada

1)si a < 0 o a >tamaño: violación de memoria

2)a’ = B + a

3)a’ es la dirección física solicitada

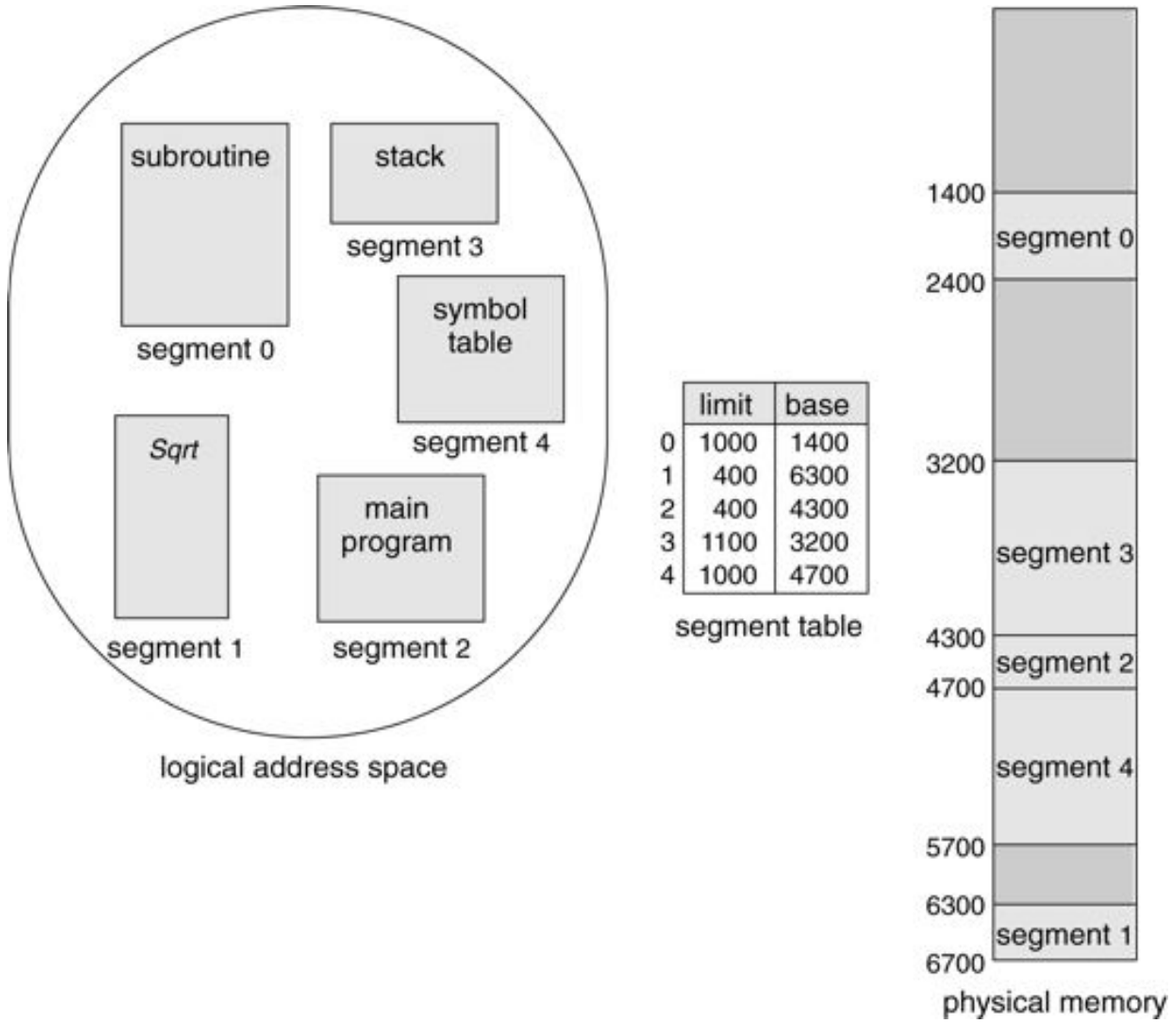
Registros base y límite (Cont.)

- Por desempeño, ambos registros deben ser registros físicos (rápidos)
- Impractico tener un par de registros por cada proceso
    - => Hay un par de registros por cada CPU, los cuales se cargan con los valores “base” y “límite” del proceso actual. Estos son parte del contexto de hardware (o entorno volatil) de cada proceso
- Procesos con el mismo ejecutable: dos conjuntos de registros.

Segmentación

Segmentación

- Es la generalización de la estrategia “registros base-límite”.
- Ahora hay varios segmentos por proceso; cada uno con su dirección “base” y “límite”.
- Su propósito es dividir el espacio de memoria virtual de un proceso, en *bloques de direcciones* lógicamente relacionadas.

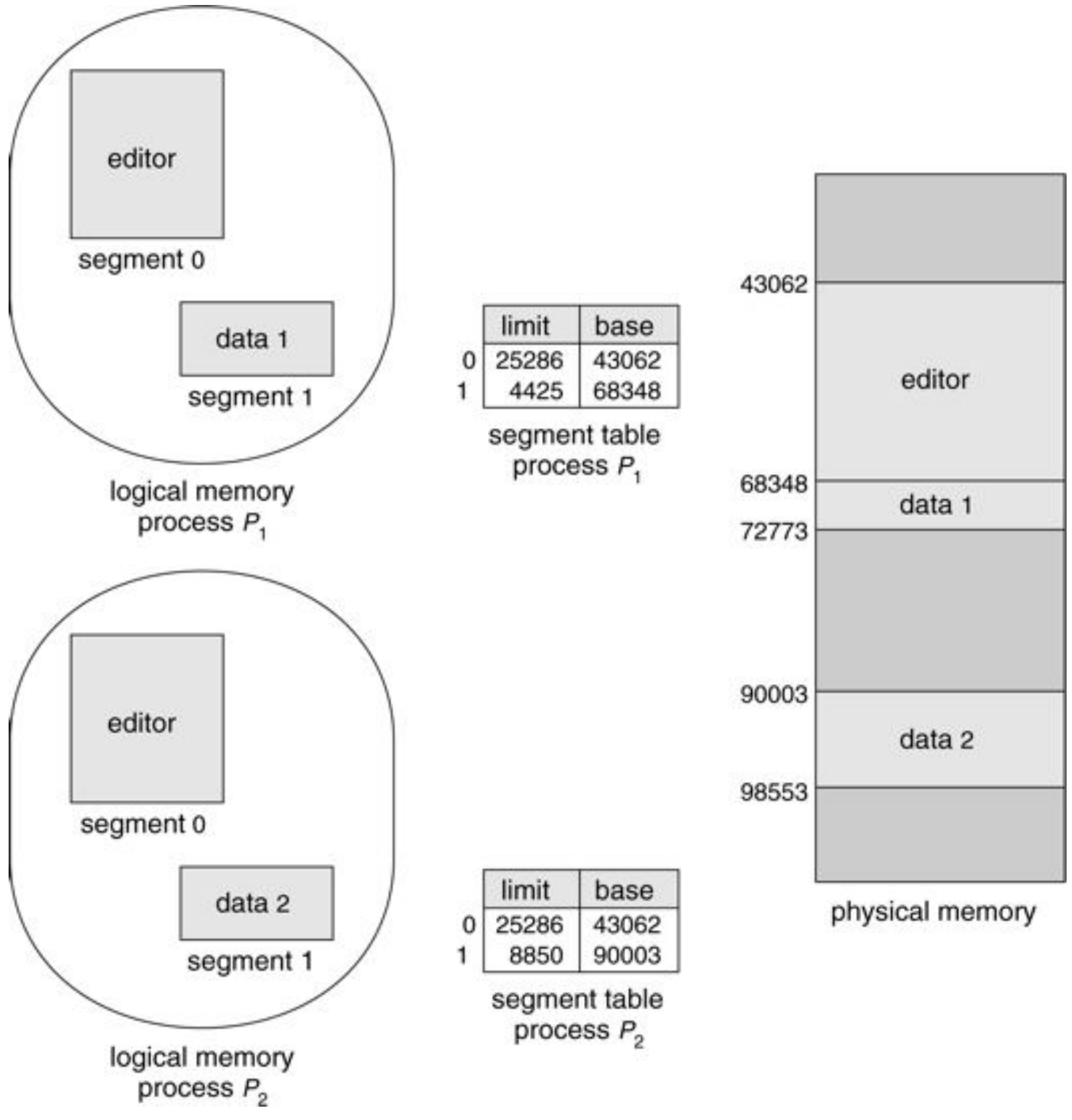


Segmentación (Cont.)

- Compartir Código de solo lectura entre aplicaciones es trivial con segmentación.

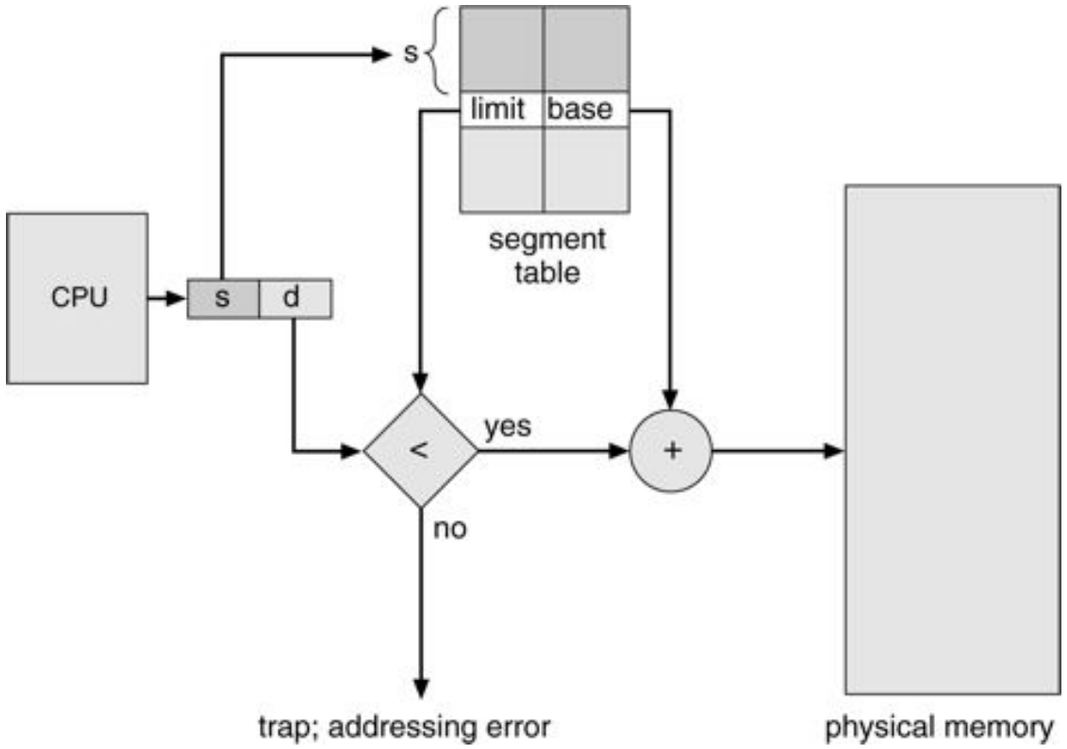
- Cada segmento se marca con ciertos permisos (lectura, escritura y ejecución).

  - Esto brinda protección a los accesos a memoria, y ayuda a descubrir errores de programación!



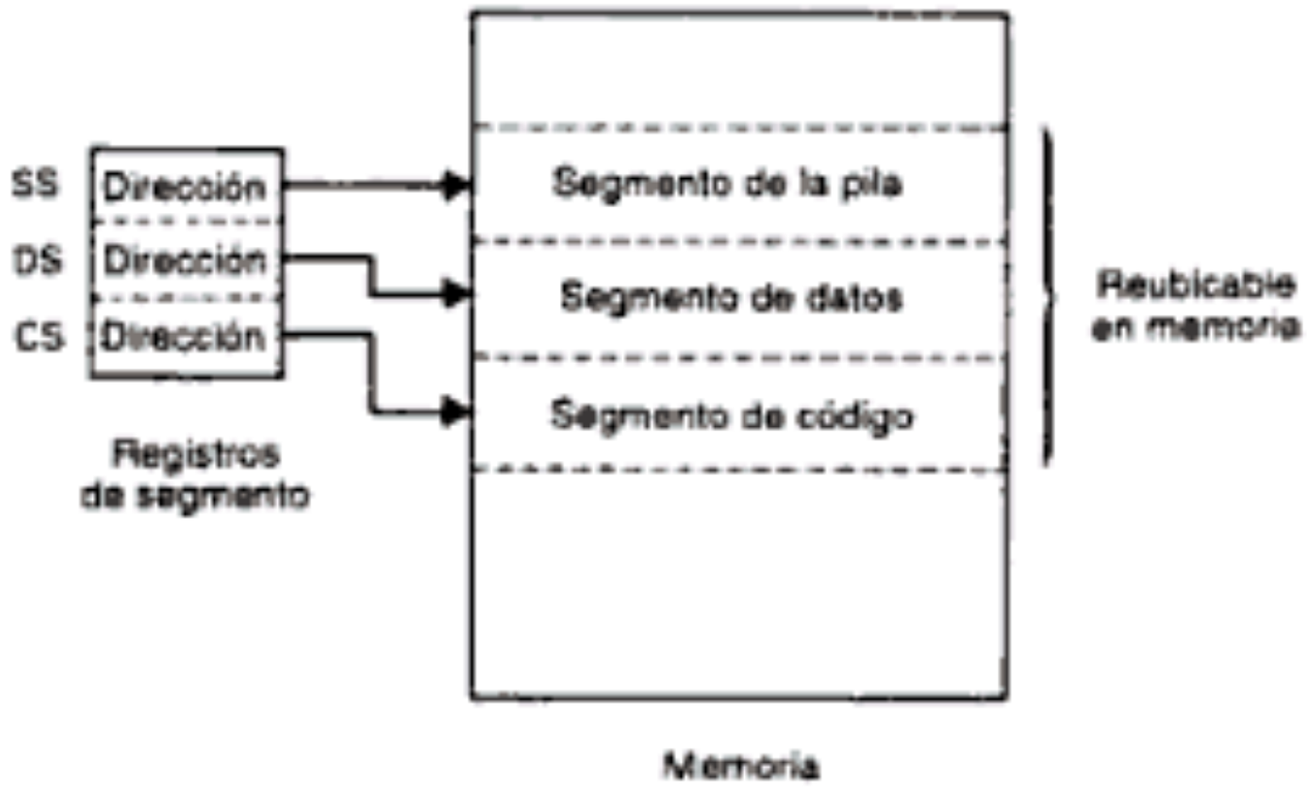
Segmentación

- Requiere ayuda del Hardware (unidad de segmentación)
- Tabla de Segmentos. Tiene una entrada por cada segmento: Base/Límite/permisos/...
  - Una por proceso!
- El formato de una dirección de memoria es ampliado a un par <segmento,dirección>.
  - Ejemplo 386: 16+32=48 bits



Segmentación (cont.)

- Ejemplo IA32:
  - Seis registros de segmento:
    - CS (Code)
    - DS (Data)
    - SS (Stack)
    - ES (Extra)
    - FS, GS (Extras)



Paginación

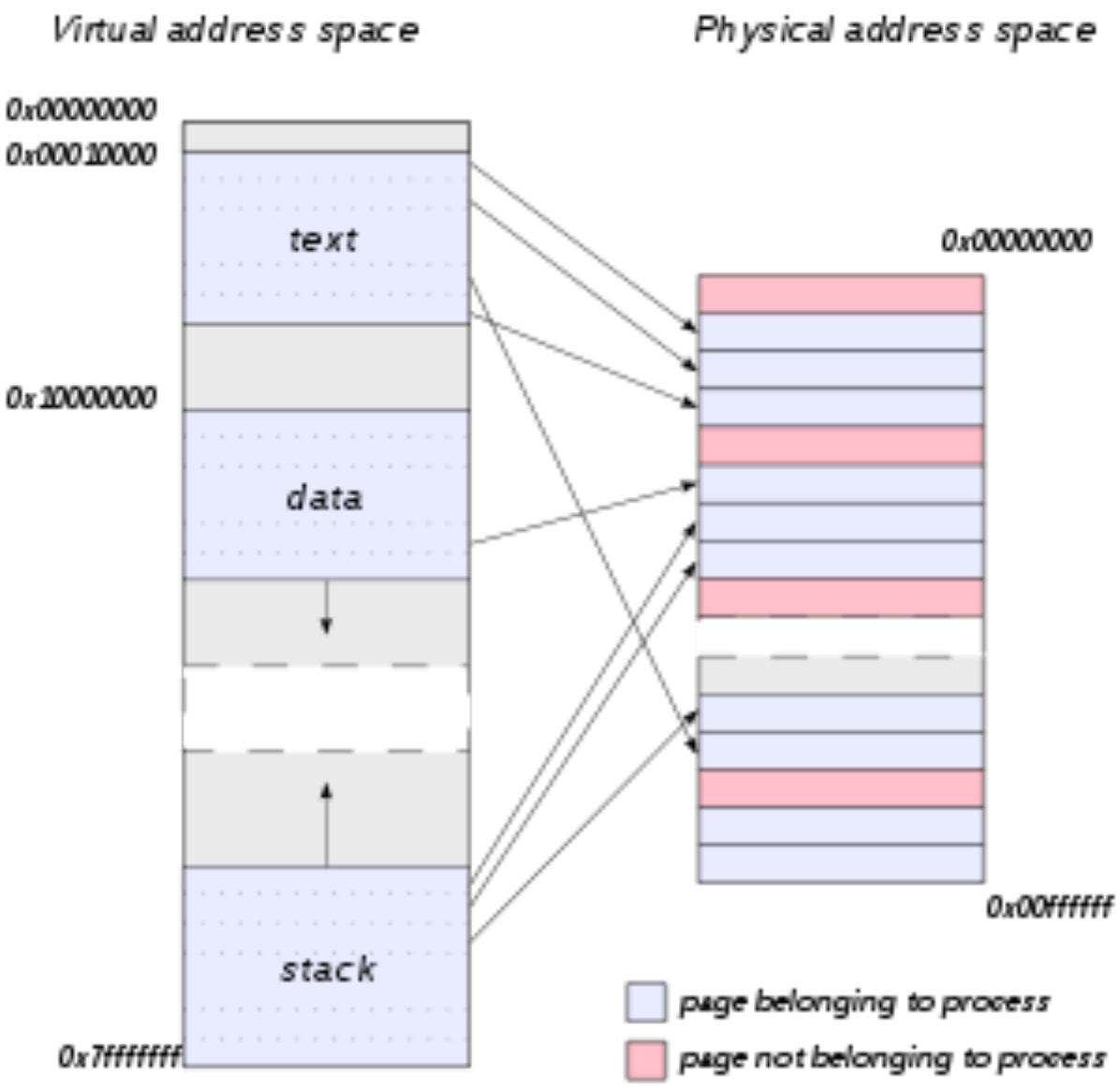
Paginación

- Un problema con registros base-límite, es que el espacio de direcciones es menor o igual que la memoria física instalada.
    - La paginación ayuda a resolver esto, dando en efecto la ilusión de tener más memoria disponible que la que realmente hay
        - Lo hace “abstrayendo” la diferencia entre memoria RAM y secundaria
- Técnica muy vieja pero de gran impacto (Universidad de Mánchester,60’s)

Páginas

- El espacio de direcciones virtual se divide en páginas
  - Bloque de direcciones de memoria de tamaño fíjo
- La memoria RAM se divide en márcos de página
  - Bloque de direcciones de memoria de tamaño fíjo
- Los marcos de página se reparten entre los procesos.
  - En un momento dado, un proceso tiene paginas

    en RAM (activas) y páginas en disco (inactivas)



Funciones del mecanismo de paginación

1. Llevar a cabo el mapeo de direcciones virtuales a físicas:
    1.1. Determinar a qué página pertenece una dirección virtual
    1.2. Determinar a que márco de página corresponde esa página
2. Transferir páginas entre disco y RAM (intercambio)
    2.1. De disco a RAM cuando se ocupe una dirección de memoria inactiva
    2.2. De RAM a disco, para liberar espacio que pueda usar otro proceso

¿En qué página se localiza el byte con dirección *A*?

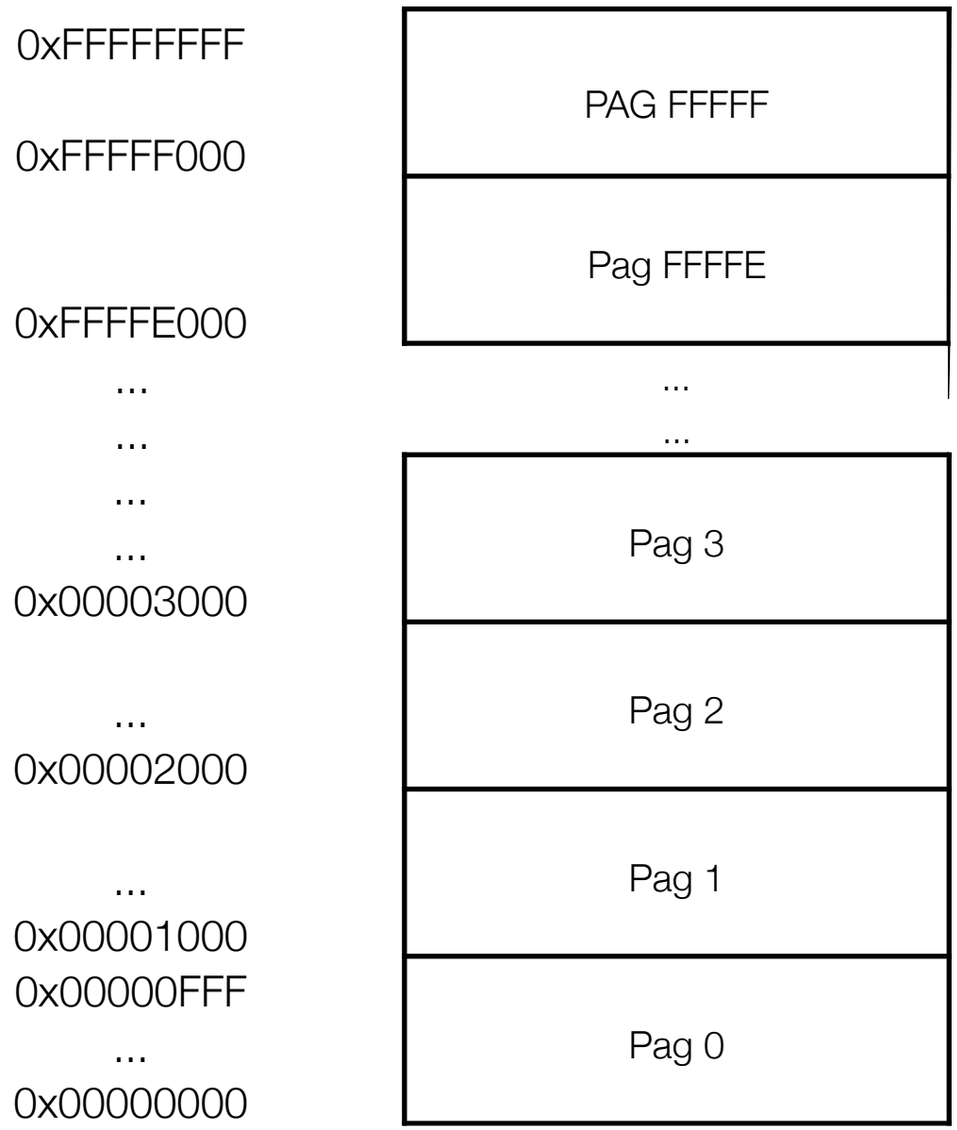
Ejemplo con direcciones de 32 bits y tamaño de página de 4KB (Linux/Intel)

pag(A)= Entera(A/4096)

byte(A)=Residuo(A/4096)

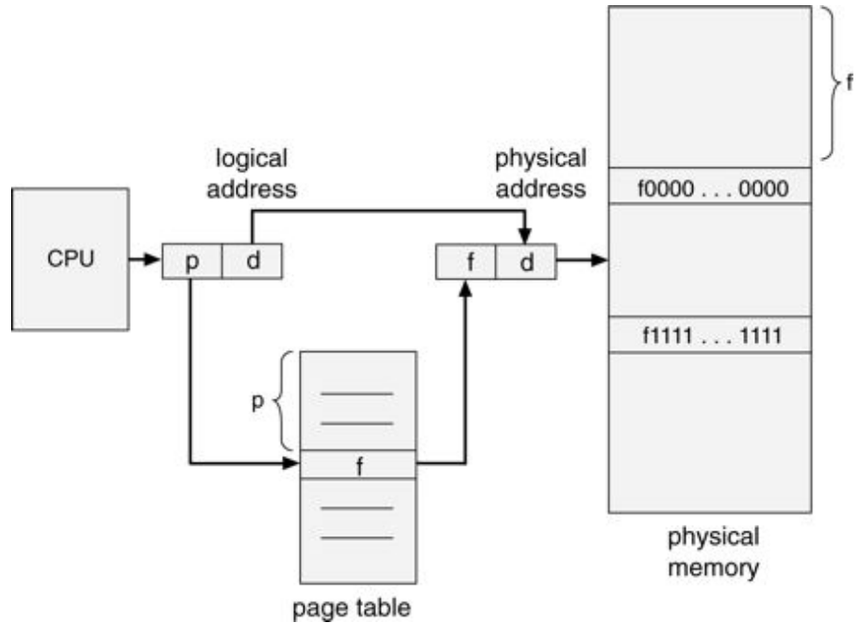
(Bits altos/bajos)

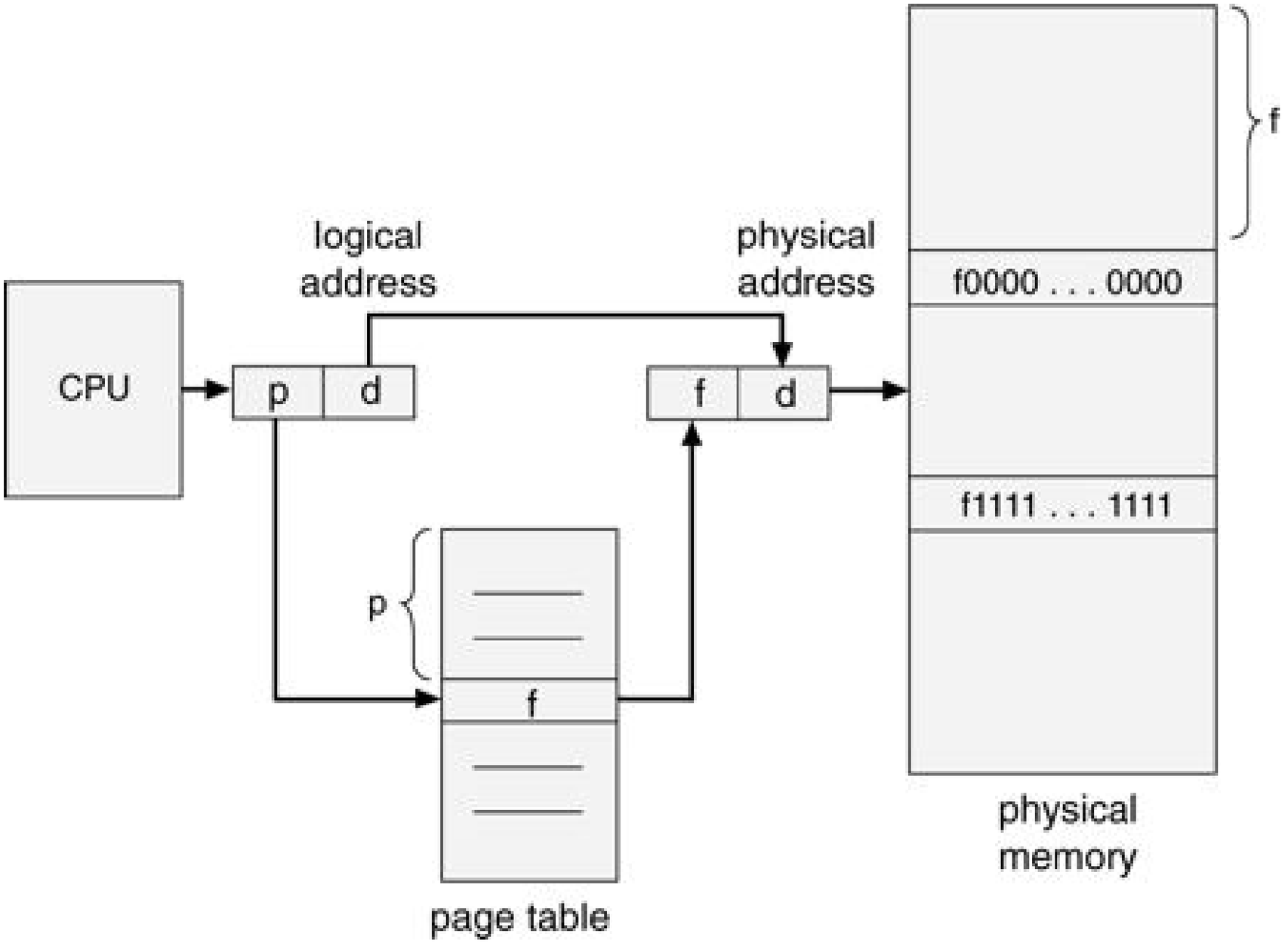
Calculadas por HW!



¿A qué página física corresponde la página virt. *p*?

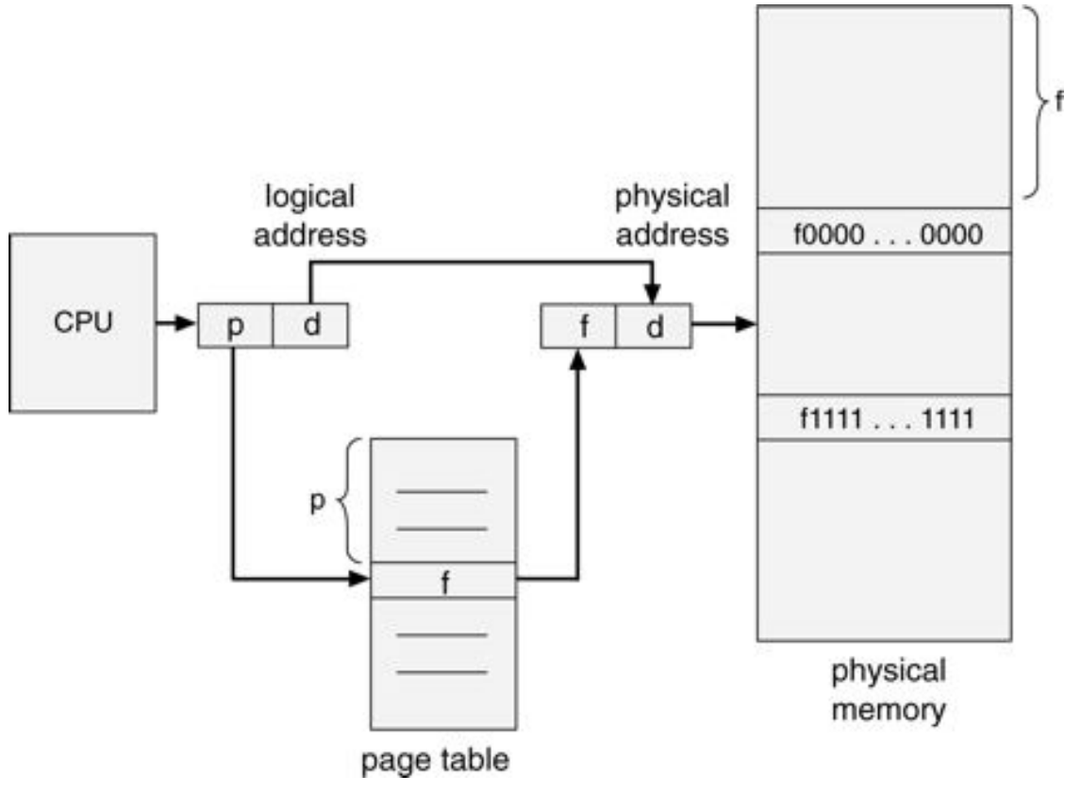
- Tabla de páginas (esquema simple):
  - Una tabla por proceso: un apuntador a la tabla del proceso actual se carga en un registro del procesador.
  - La entrada *p*-ésima indica el marco de página que le corresponde a la página *p,* y un poco de información administrativa:
    - Cuantas veces la página se ha referenciado
    - Tiempo de última referencia
    - Si se ha escrito a esa dirección (dirt page)




logical
address
physical
address
CPU
p
d
f
d
f
f0000 . . . 0000
f1111 . . . 1111
p
f
page table
physical
memory

Dirección virtual -> Dirección física

- Mapeo de direcciones
  - *A=(p,b)* // Dividimos la dirección en Página/Byte
  - *f(A) = f(p,b) = p' + b*
- Donde:
  - *z :=* tamaño de página
  - *p* := Entera(A/z),
  - *b* := Residuo(A/z)
  - *p'* := Tabla_de_páginas[p]

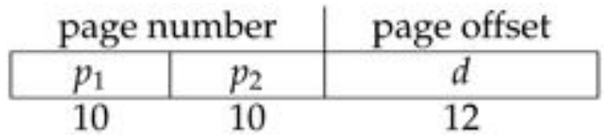


Problema con una tabla tan simple

- Tabla enorme (hagamos cuentas)
    - Tamaño de página 4K, Espacio de memoria 4GB:
        - 2^32/2^12= 2^20 entradas, cada una de cuatro bytes:
            - el tamaño es 2^22 bytes = 4MB
- => No podemos implementarla en registros
- Una tabla por proceso!
    - 100 procesos: 400 MB sólo de tablas de páginas!!

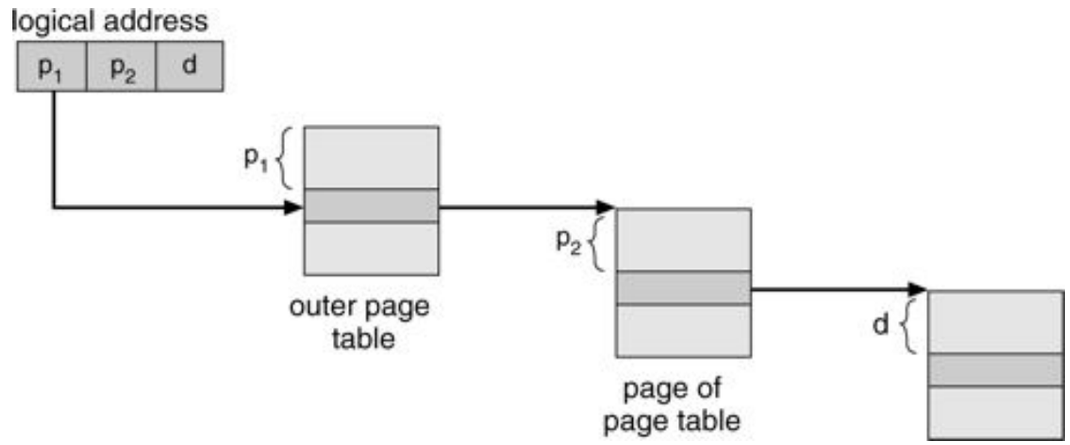
Múltiples niveles de tablas

- Se divide el total de páginas en “directorios” de tamaño fijo. (se puede repetir varias veces.)
  - Pentium y posteriores: dos niveles.



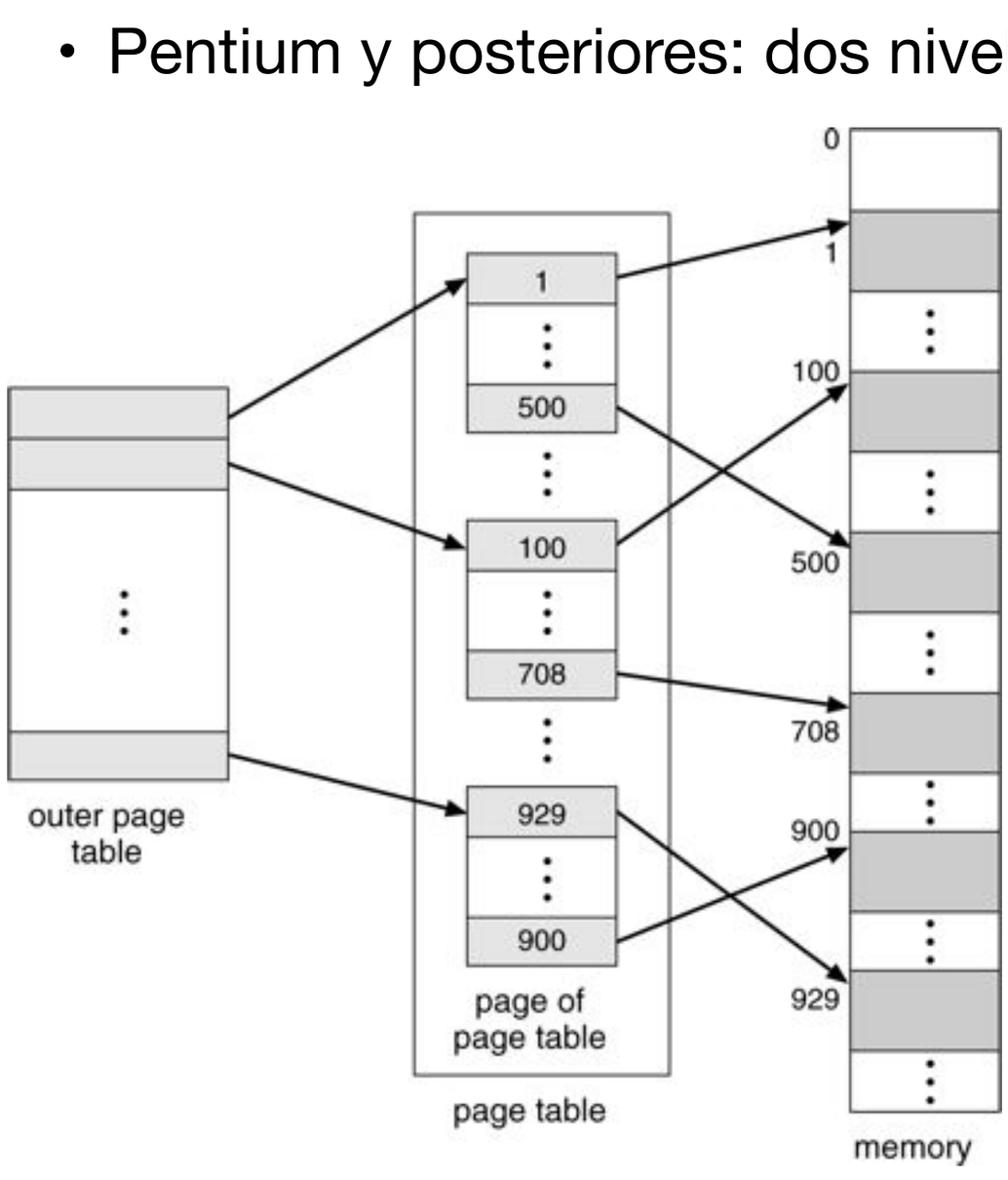
  - Sparc (32 bits): 3 niveles.
  - Motorola 68030 (32 bits): 4 niveles.

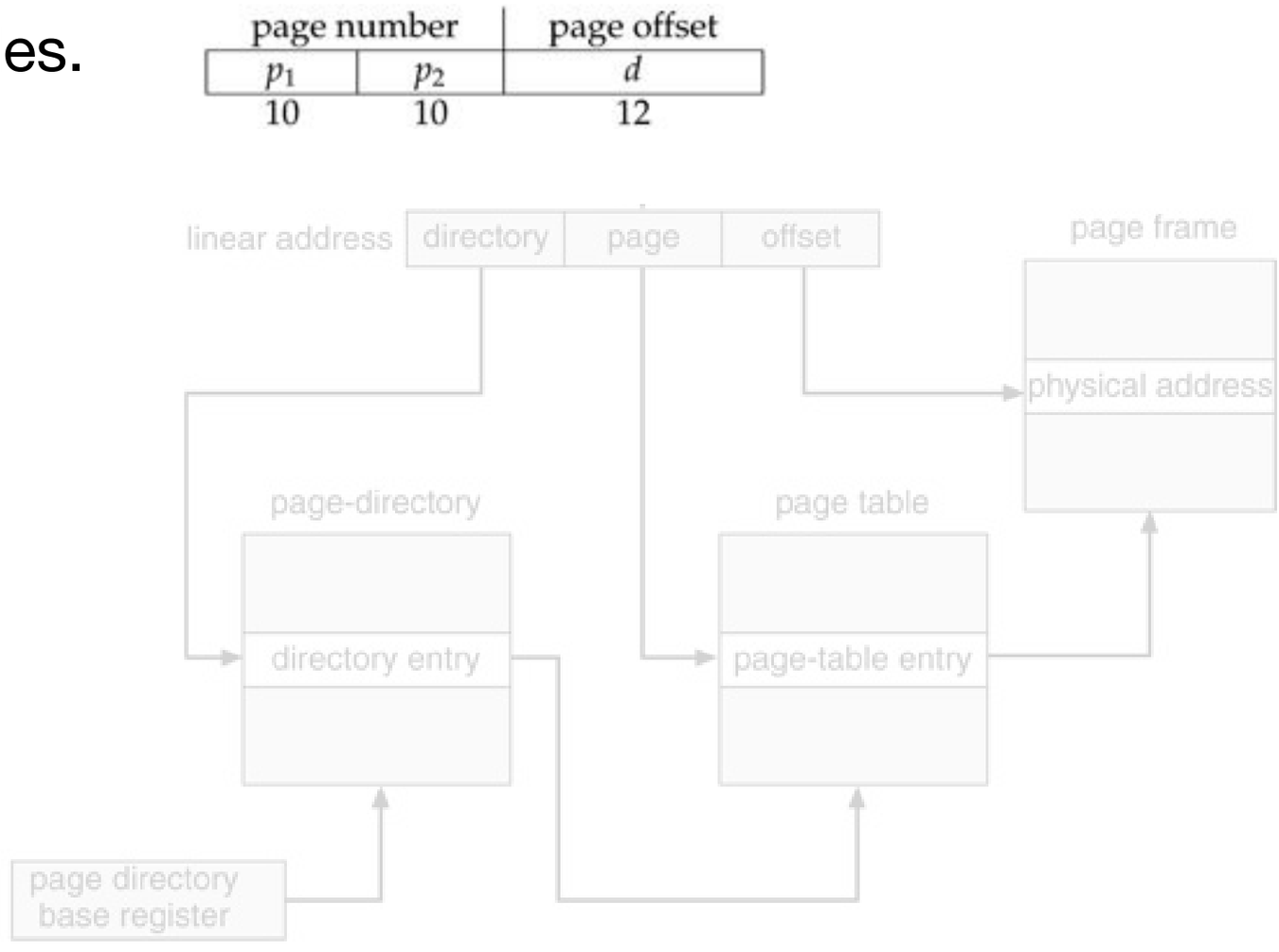
- Se realiza un acceso a RAM por cada nivel de “pagina”



o

- Pentium y posteriores: dos niveles.

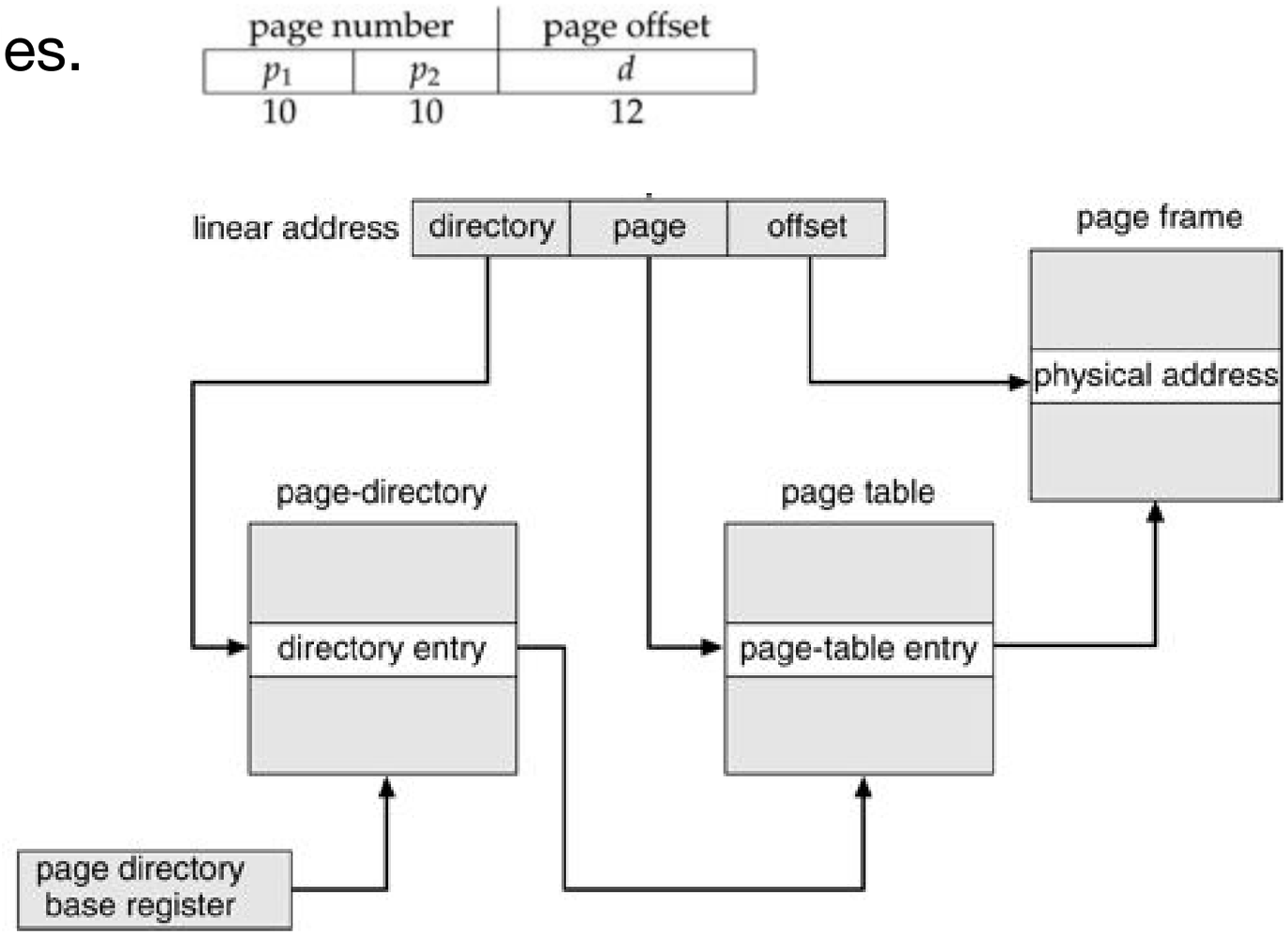




- Se realiza un acceso a RAM por cada nivel de “pagina”.

Múltiples niveles de tablas (cont.)

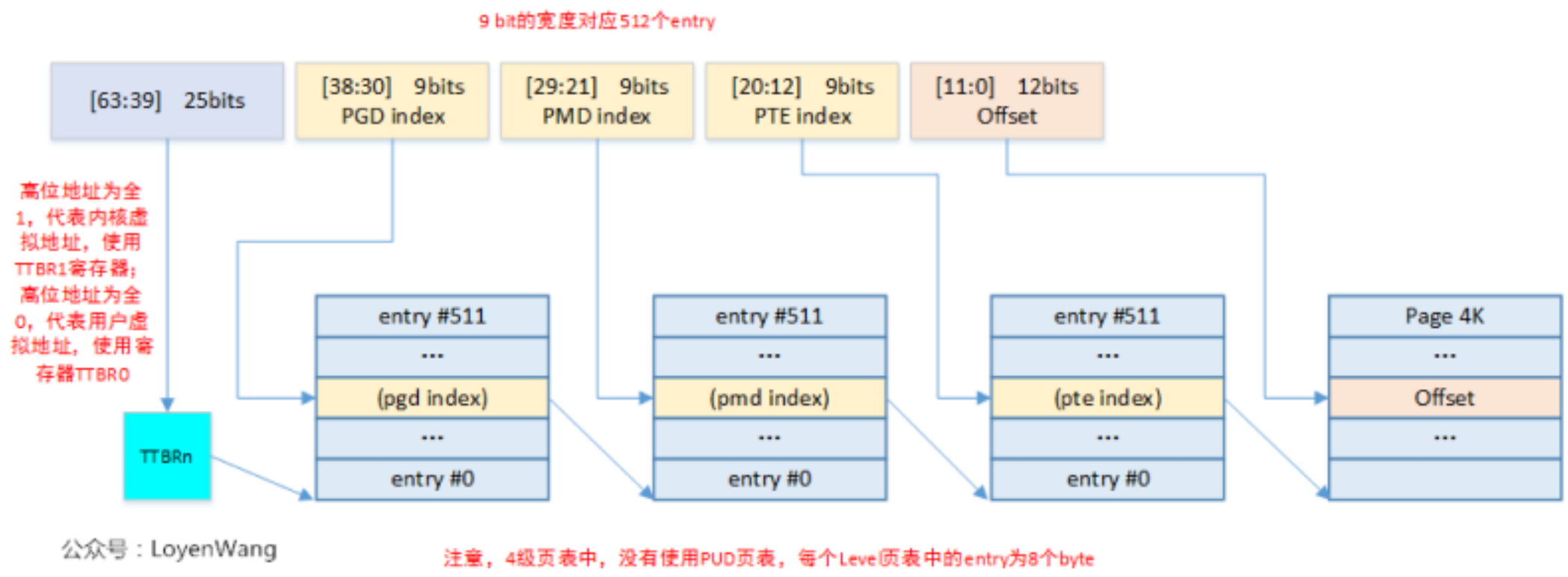
- Pentium y posteriores: dos niveles.



- Se realiza un acceso a RAM por cada nivel de “pagina”.

Múltiples niveles de tablas

- Se divide el total de páginas en “directorios” de tamaño fijo. (se puede repetir varias veces.)



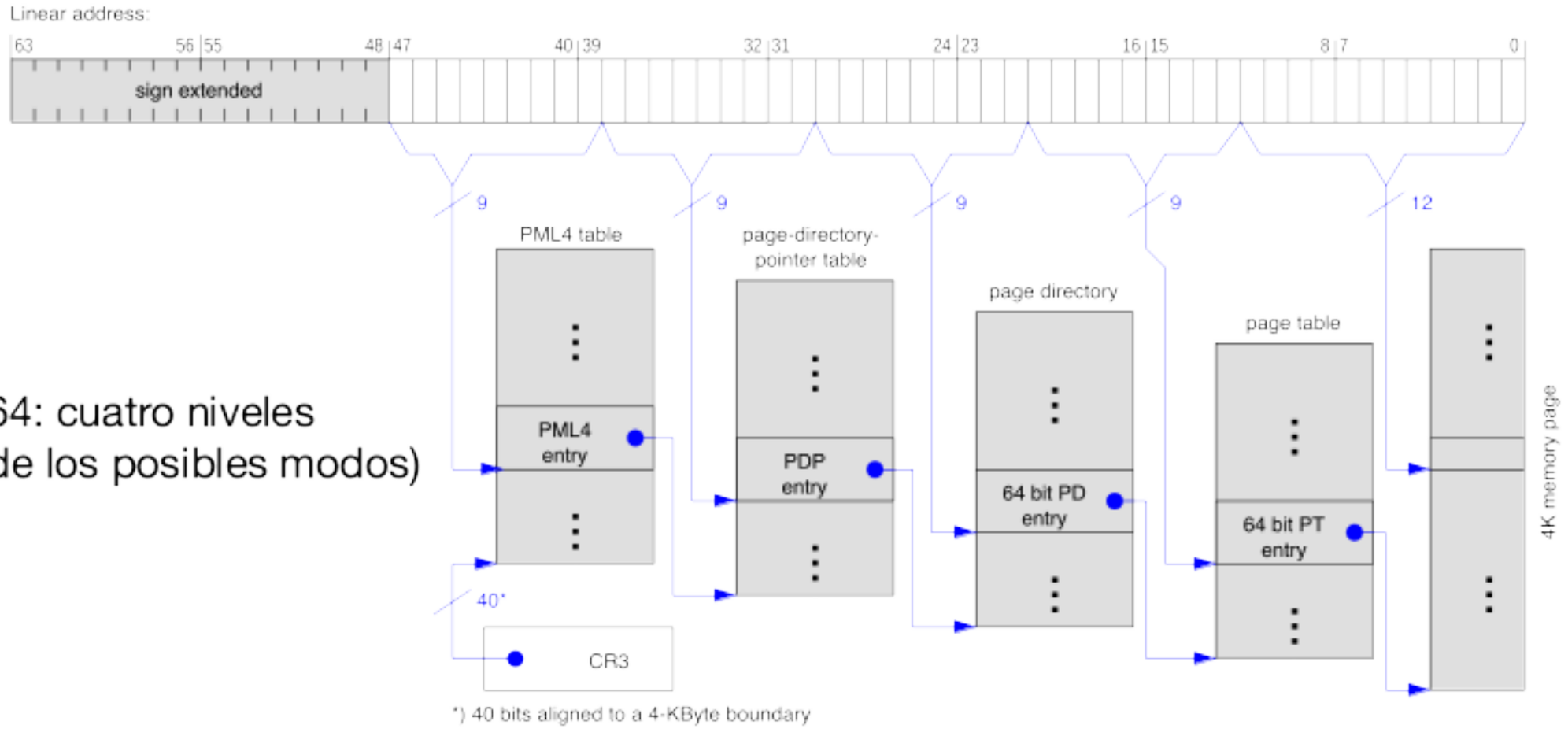
ARM64: tres niveles (hay más posibilidades)

- Se realiza un acceso a RAM por cada nivel de “pagina”

Múltiples niveles de tablas

- Se divide el total de páginas en “directorios” de tamaño fijo. (se puede repetir varias veces.)

AMD64: cuatro niveles
(uno de los posibles modos)

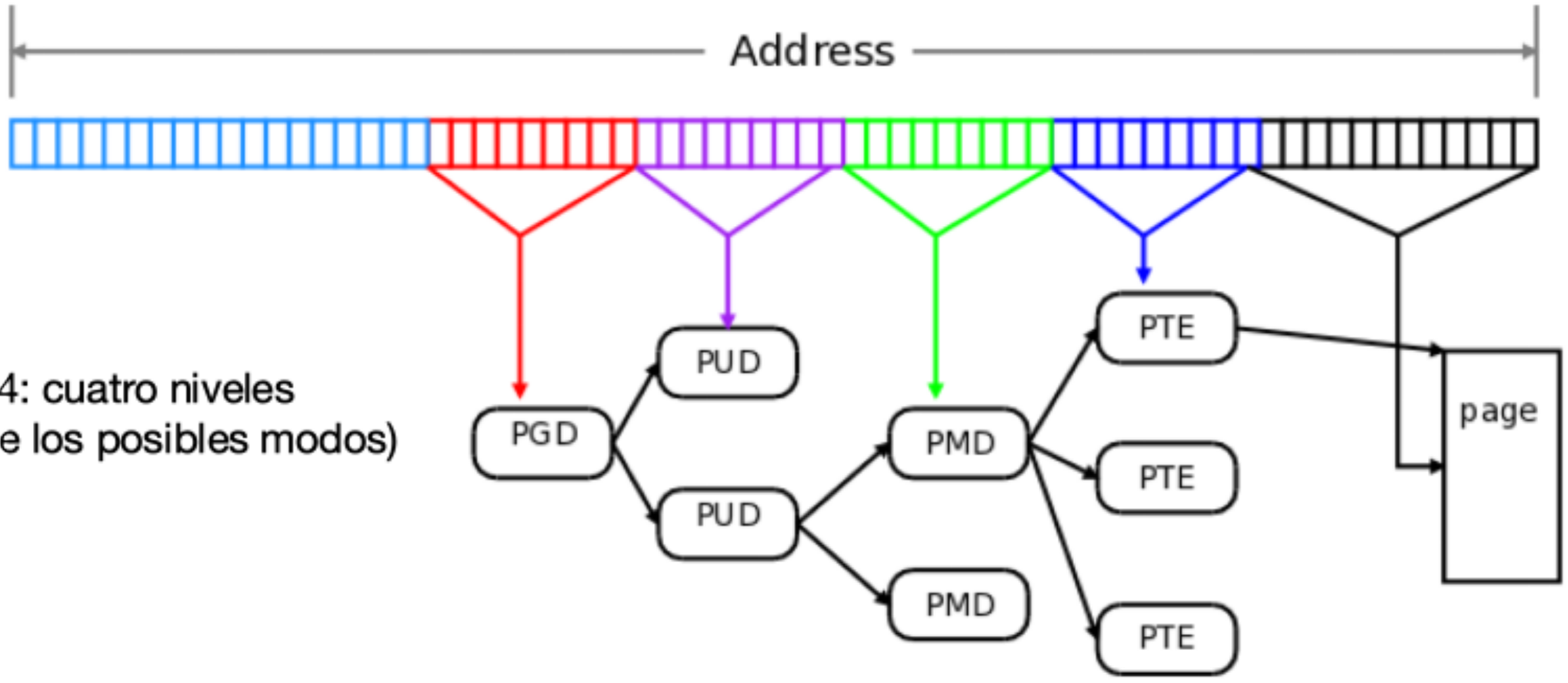


- Se realiza un acceso a RAM por cada nivel de “pagina”

Múltiples niveles de tablas

- Se divide el total de páginas en “directorios” de tamaño fijo. (se puede repetir varias veces.)

AMD64: **cuatro niveles**
(uno de **los posibles modos**)



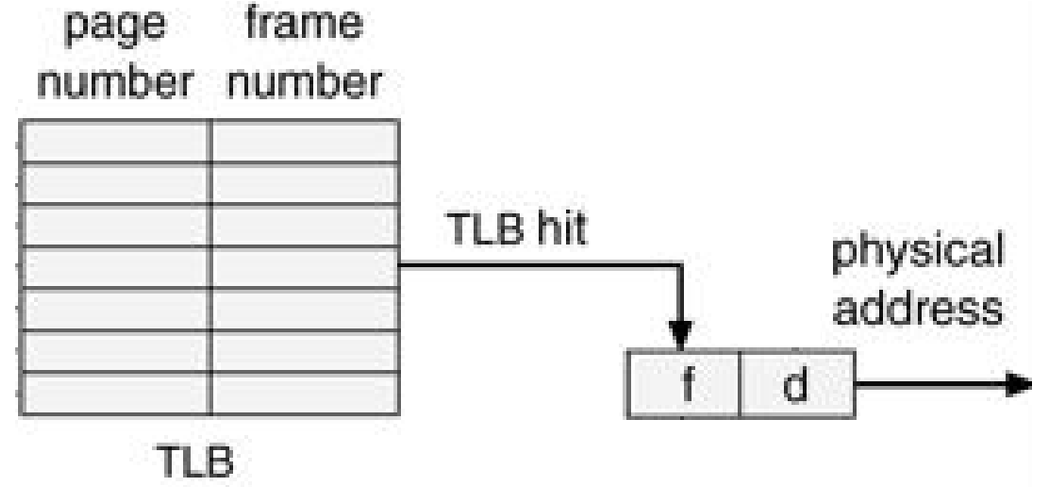
- Se realiza un acceso a RAM por cada nivel de “pagina”

Múltiples niveles de tablas

- Pros: El espacio usado por la tabla de páginas puede disminuir mucho.
    - Subárboles que sólo contienen hojas no “activas” no ocupan memoria
    - Ejemplo: Intel32.
- Cons: Cada nivel requiere UN acceso a memoria.
    - En niveles muy profundos, esto aumenta mucho el tiempo total de acceso a una dirección física a partir de una virtual.

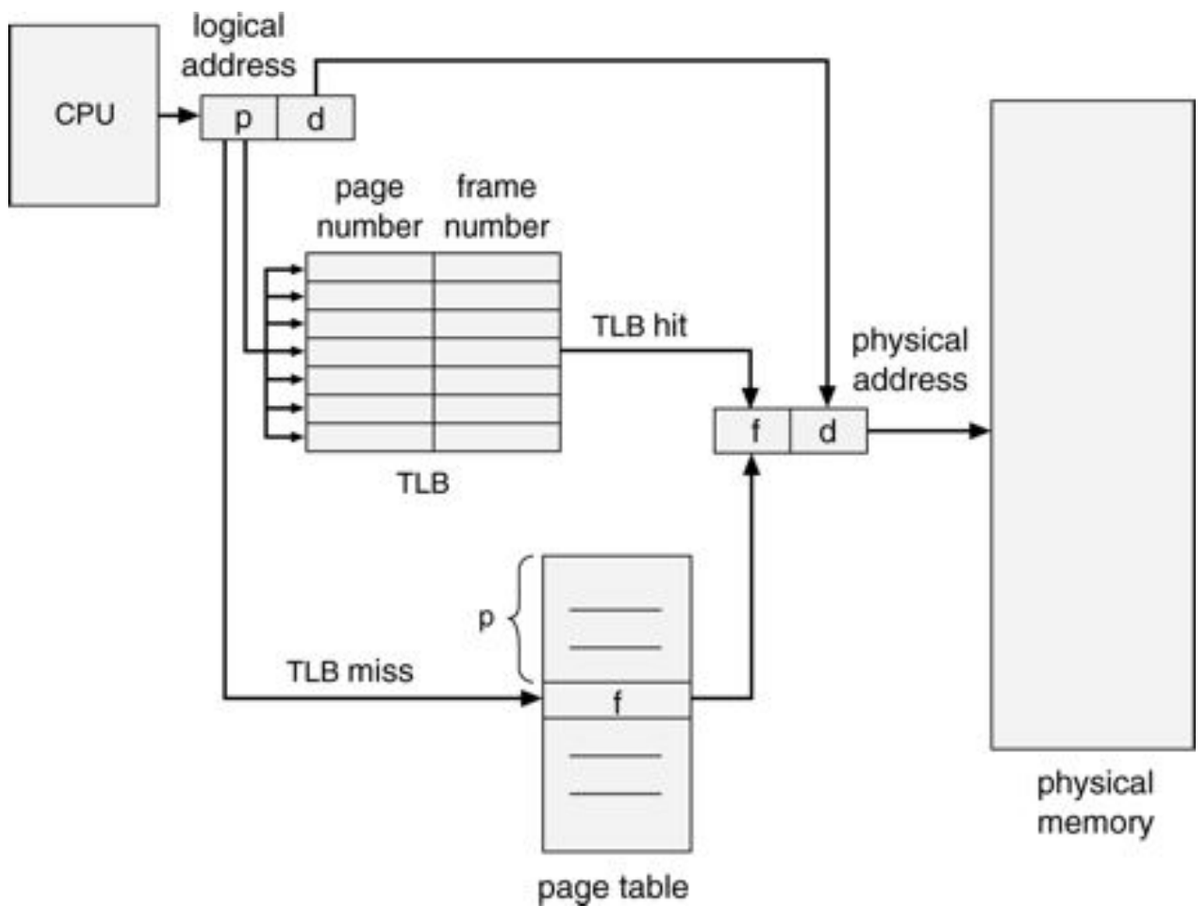
Solución: TLB (tabla asociativa)

- TLB: Translation Lookaside Buffer
- También llamado “associative store”
- Permite hacer búsquedas en paralelo
- Consta de parejas página/marco



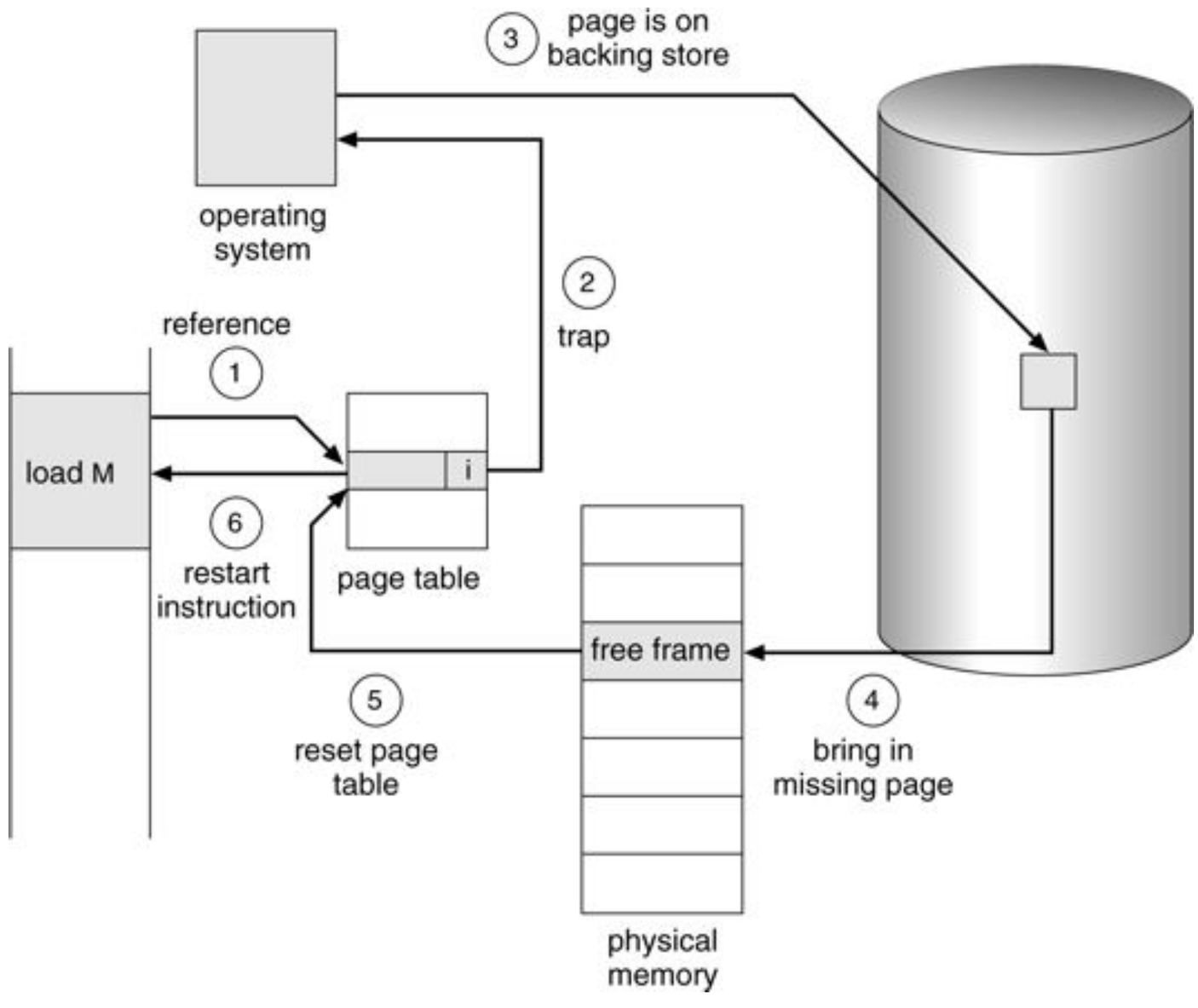
Práctica: Combinación de TLB y PT

- Casi toda la operación hecha por HW
- El sistema operativo entra cuando la página no se encuentra en la tabla.



¿Y si la página no se encuentra en la PT?

- El kernel toma el control



Diferentes espacios de memoria para diferentes procesos...

- un apuntador a la tabla del proceso actual se carga en un registro del procesador.

- Para cambiar de proceso actual,

- recuperamos la dirección de la TP del nuevo proceso a partir de su descriptor de HW, y la escribimos en el registro adecuado de la MMU.

- Invalidamos las entradas del TLB.

- Restauramos el resto del entorno volatil del proceso a ejecutar

Ejemplos de aplicaciones de paginación:

- Compartir memoria:
    - Páginas virtuales de diferentes procesos se asocian a la misma página física.
- Si la memoria no es “totalmente compartida”: copy-on-write
    - Es un mecanismo que permite posponer la copia de bloques de memoria hasta que uno de los procesos hace una modificación sobre ese bloque.
- Archivos mapeados a memoria:
    - Es una técnica para facilitar el acceso a archivos: el archivo se vé como un “arreglo” en memoria.
- ZSWAP: Compresión de memoria en lugar de mandarla a almacenamiento secundario (en la práctica: combinación de ZSWAP e intercambio)

Estrategias de remplazo de páginas

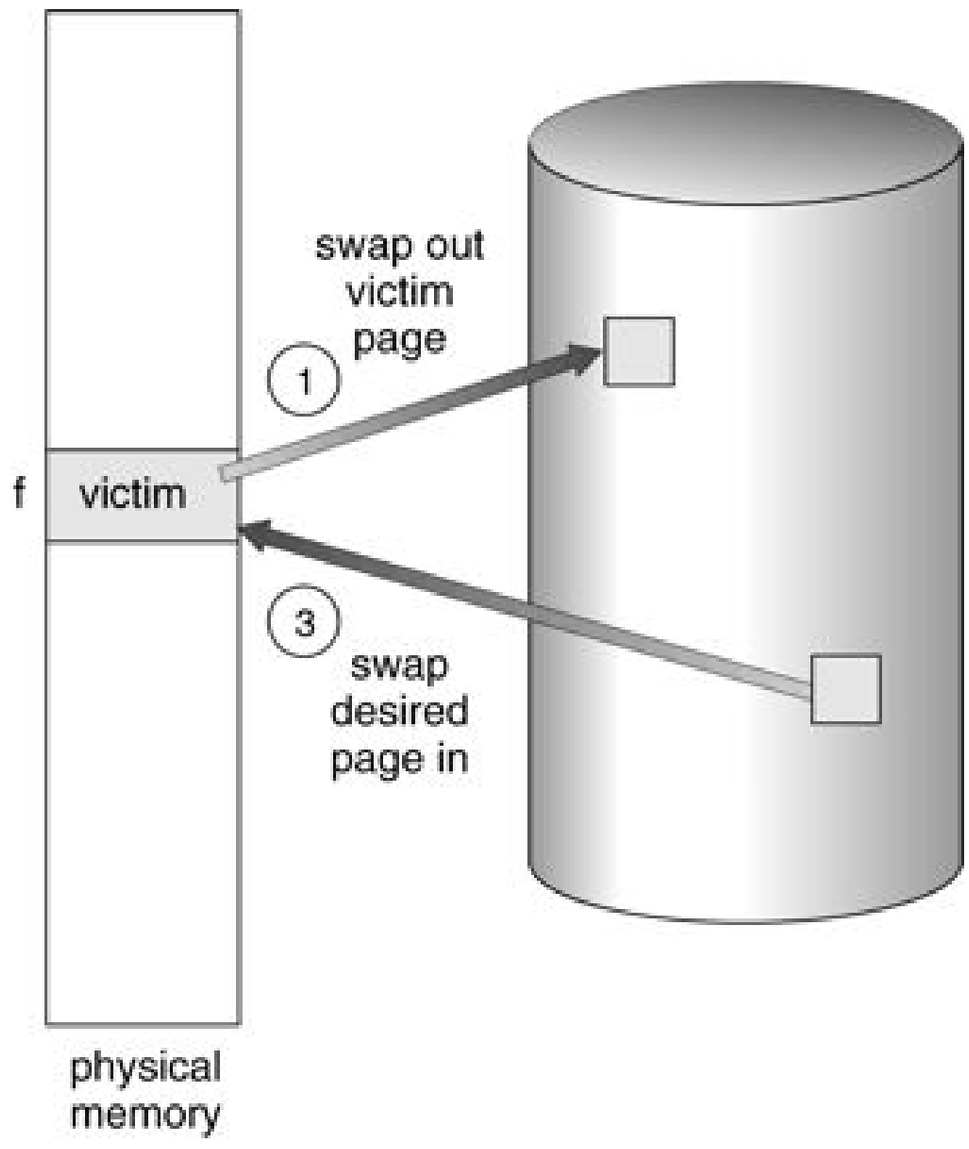
Motivación

- Cuando una falla de página ocurre, puede ser necesario traer la página referenciada de memoria secundaria a RAM.
    - !Es posible que no haya una página física libre para poder copiar esta página en ella!
- Debemos entonces “hacer espacio” en la RAM.
- ¿De qué manera podemos “hacemos espacio”?
    - Mandar “todo” un proceso a disco.
    - !Estrategias de reemplazo de páginas!

Estrategias de reemplazo de páginas

- Las que “no” usan estadísticas de uso.

- Las que “sí” usan estadísticas de uso.
  - referenced bit
  - dirty bit

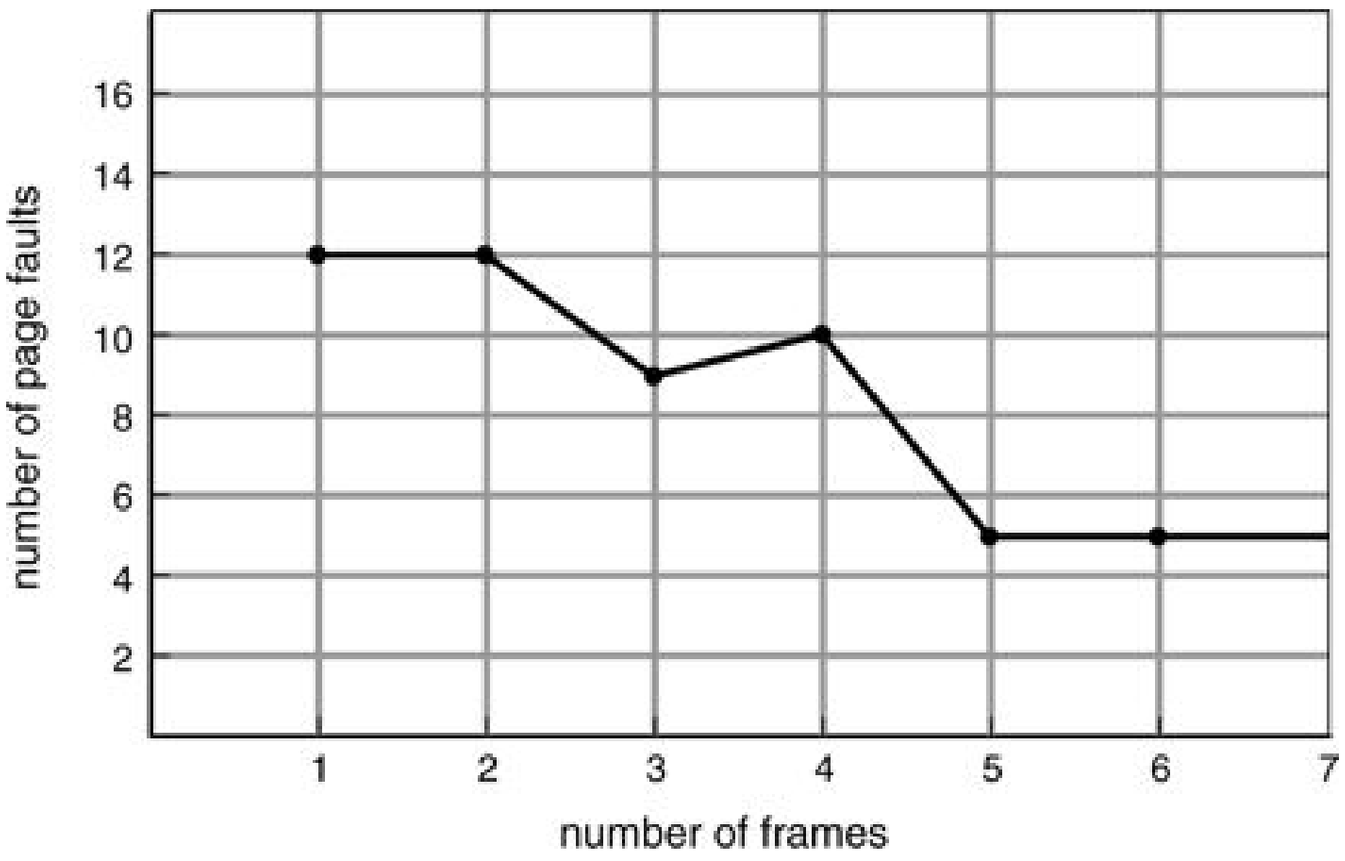


Las que “no” usan estadísticas de uso

- FIFO: El primero en entrar es el primero en salir.
    - Muy fácil de implementar. Sólo hace falta implementar una cola (lista doblemente ligada p. ej.).
- Desempeño pobre:
    - Ignora que la página más vieja puede ser la más referenciada!
    - Puede suceder el fénomeno de la “anomalía de Belady”:
        - ¡¡¡Incrementar la memoria baja el desempeño!!!

Anomalía de Belady

- Secuencia de referencias:
    - 1,2,3,4,1,2,5,1,2,3,4,5
- Gráfica de fallos de página contra número de páginas físicas:



Las que “sí” usan estadísticas de uso

- Intentan predecir el uso futuro a partir del historial de uso anterior.
    - Esto se justifica mediante el “principio de localidad de referencia”.
        - Reloj: Modificación simple de FIFO. (Esquema más simple usado en la práctica)
        - LRU:La del último acceso “más viejo”. (Usada en la práctica.)
        - LFU: Menos usadas frecuentemente.
- La implementación es un poco más complicada y requiere ayuda de HW.
- Se puede demostrar que LRU (y cualquier otra técnica con la *stack property*) no presenta la anomalía de Belady.

En un mundo utópico: técnica de reemplazo OPT

- Si conocemos de antemano la secuencia de referencias, podemos tener una técnica óptima (genera el mínimo número posible de fallos de página).
    - Regla greedy simple: la página víctima será la que tarde más tiempo en volver a ser reverenciada.
- No realizable en la práctica, pero ayuda a diseñar otras técnicas.
    - LRU se puede comprender como una implementación de OPT, bajo la hipótesis de que *el futuro es un reflejo del pasado*.

Hiperpaginación

(T*h*rashing)

thrash | *θ* ra sh |

verb [trans.]
beat (a person or animal) repeatedly and violently with a stick or whip *: she thrashed him across the head and shoulders* | [as n.] (**thrashing**) *what he needs is a good thrashing.*
noun
1 [usu. in sing.] a violent or noisy movement, typically involving hitting something repeatedly *: the thrash of the waves.*
2 (also **thrash metal**) a style of fast, loud, harsh-sounding rock music, combining elements of punk and heavy metal.

Hiperpaginación

- “La situación en la que una creciente cantidad de recursos son utilizados para hacer una cantidad de trabajo cada vez menor”

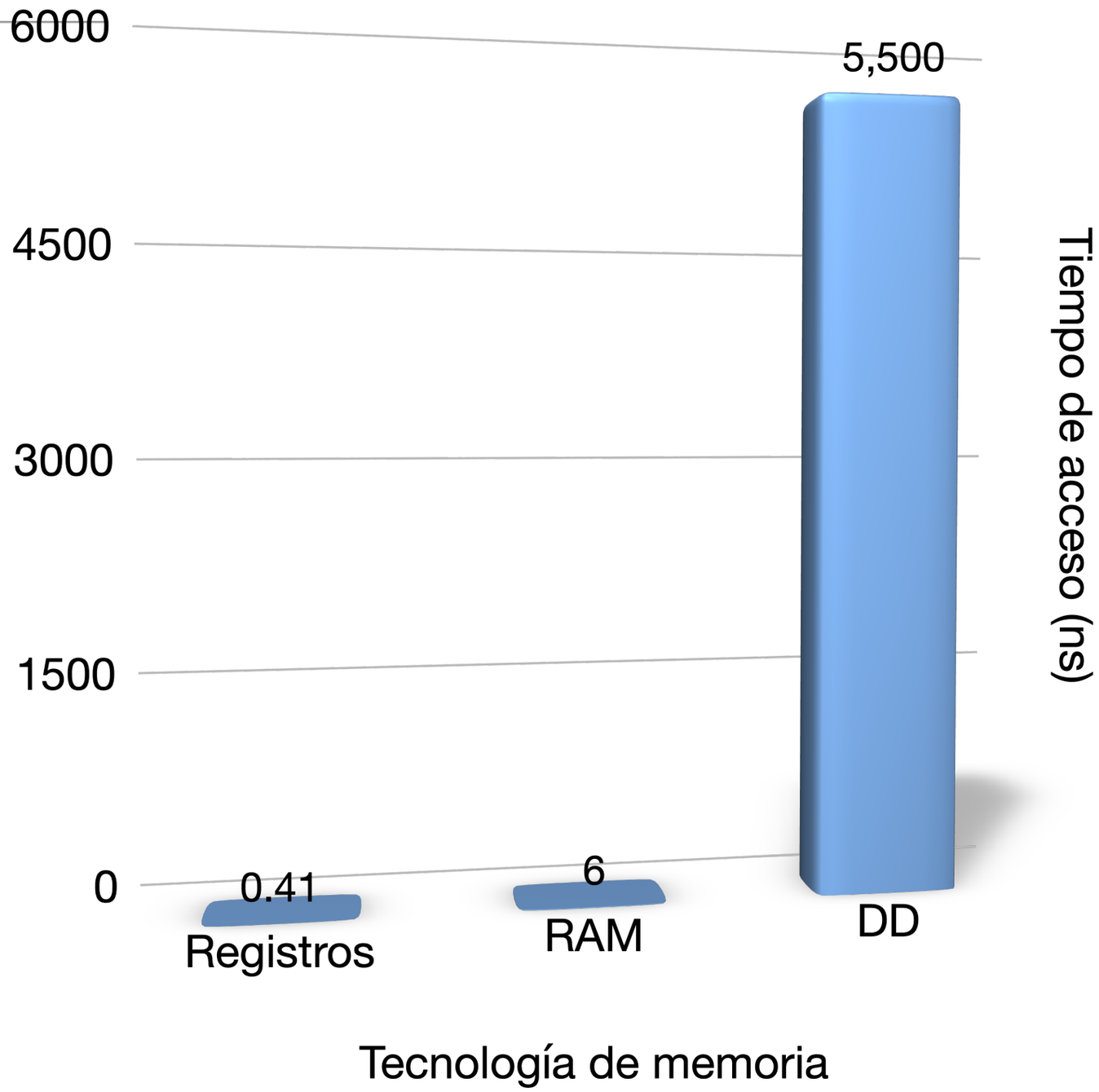
- Todos la hemos vivido alguna vez: muchos procesos en computadoras con poca RAM.

    - El sistema casi deja de responder: pasa la mayor parte del tiempo moviendo páginas entre disco y RAM para hacer espacio para nuevos procesos.

        - El acceso a disco es lentísimo!!!

Jerarquía de memoria simplificada

- Registros
  - RAM
    - Almacenamiento secundario



Posibles estrategias para minimizarla

- Añadir más memoria principal
- Escoger una buena estrategia de reemplazo de páginas
- Disminuir el número de procesos
- Reemplazar programas que usan mucha memoria con equivalentes que usan menos
- **Implementar una política de “conjunto de trabajo”**

Modelo de “conjunto de trabajo”

Peter J. Denning, 1968. [USA]

“Resource allocation in multiprocess computer systems"

Ph.D. Thesis, MIT.

Modelo de “conjunto de trabajo”

Peter J. Denning, 1968. [1942,USA]

Un proceso puede estar en RAM sí y solo sí todas las páginas que está usando (Considerando las páginas que ha usado recientemente) pueden estar en RAM.

Es un “todo o nada”: Si el uso de páginas aumenta y no hay RAM, *todo* el proceso es enviado a disco para liberar memoria para otros procesos.

-- Es una estrategia que combina administración de memoria con administración de tiempo de CPU.

Modelo de “conjunto de trabajo”

w(t,h) = {página *i* I *i* aparece en las últimas *h* referencias}

Hay que buscar el valor de *h* adecuado.

Un proceso puede estar en RAM sí y solo sí todas las páginas que está usando (Considerando las páginas que ha usado recientemente) pueden estar en RAM.

Es un “todo o nada”: Si el uso de páginas aumenta y no hay RAM, *todo* el proceso es enviado a disco para liberar memoria para otros procesos.

Segmentación + paginación