Tema 3. La jerarquía de memoria

Arquitectura de Computadores

Área de Arquitectura y Tecnología de Computadores
Departamento de Informática
Universidad de Oviedo

Curso 2023–2024

Objetivos

1.- Mejoras del rendimiento

Cambios relacionados con el incremento del rendimiento

2.- Soporte a los sistemas operativos multitarea

Funcionalidad necesaria para su correcto funcionamiento

Objetivos del sistema de memoria

1. Gran Capacidad
   - muchos programas, muchas instrucciones, con muchos datos
2. Rápido
   - muchos accesos a memoria (al menos uno por instrucción)
   - a veces la CPU espera por la memoria

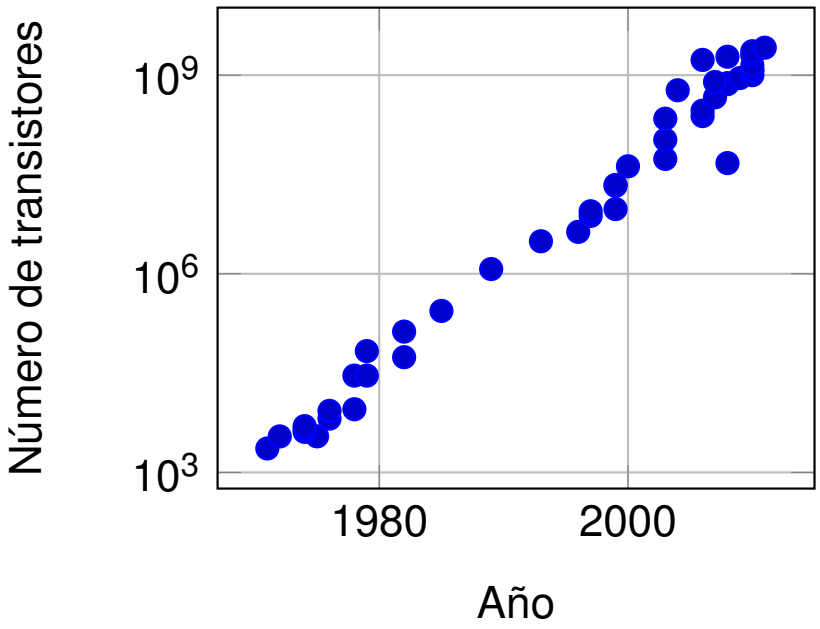
```
MOV R5, [R1]
```

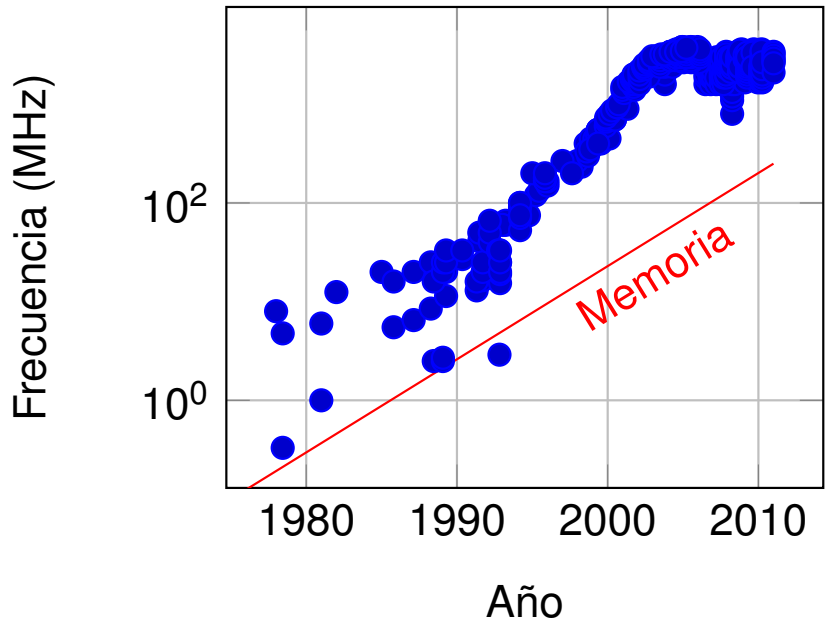
Paso		Señales de control
Búsqueda	1	PC-IB, IB-MAR, TMPE_CLR, CARRY_IN, ADD, ALU-TMPS, READ
	2	TMPS-IB, IB-PC
	3	MDR-IB, IB-IR
Ejecuc.	4	R1-IB, IB-MAR, READ
	5	Ciclo de espera
	6	MDR-IB, IB-R5, FIN

Diferencia de rendimiento

Ley de Moore

El número de transistores en un circuito integrado se dobla cada dos años





La diferencia de velocidad entre memoria y CPU es cada vez mayor

- Muro de Memoria

Efecto del *gap* memoria-CPU

Ideal

- CPU segmentada de 1 GHz $\Rightarrow$ 1000 MIPS
- Memoria rápida

Real

- Memoria más lenta
- Etapas `MEM` e `IF` largas
- CPI > 1
- Ejemplo: CPI = 1.3 $\Rightarrow$ 769.23 MIPS

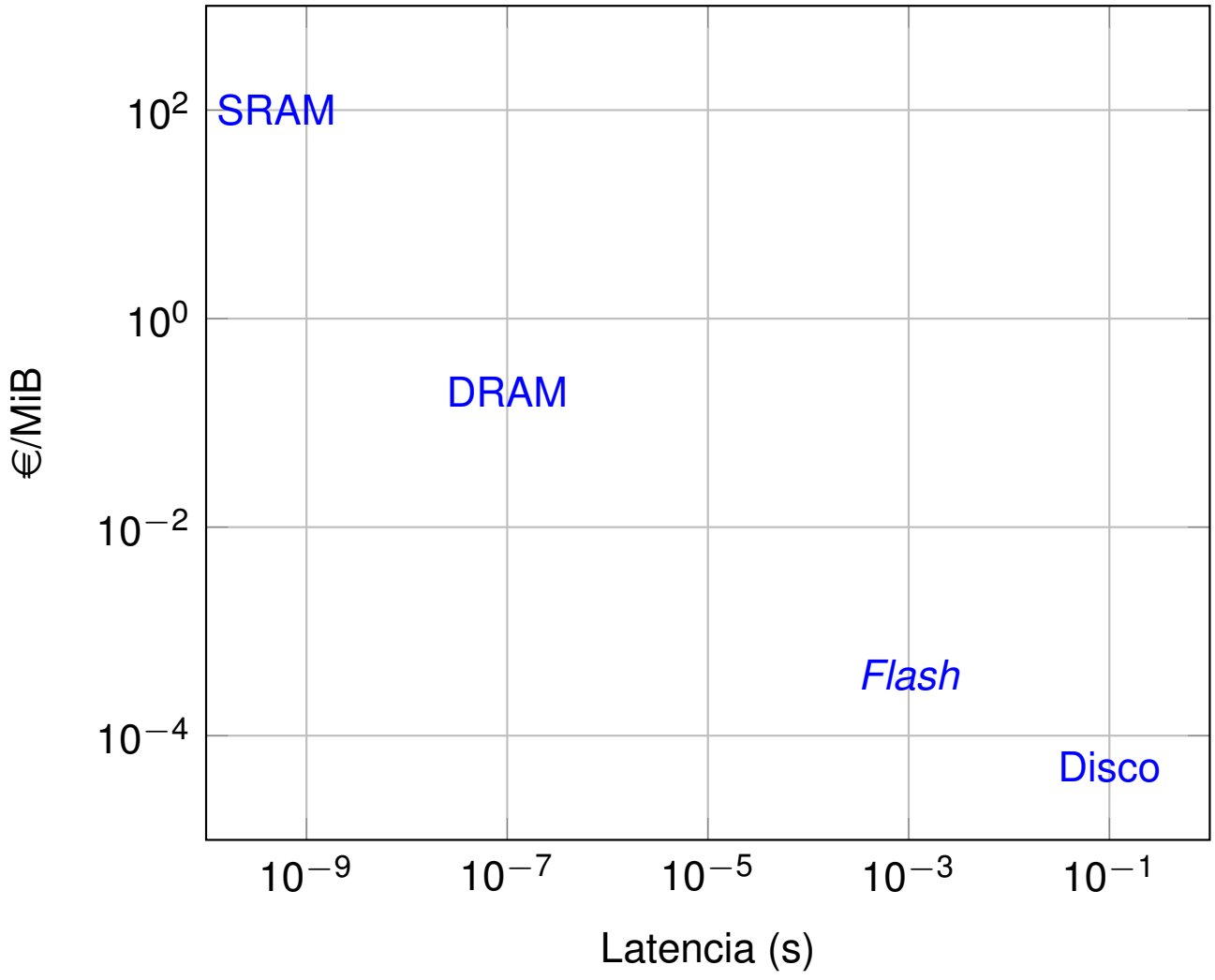
Tecnologías de memoria

	RAM estática (SRAM)	RAM dinámica (DRAM)	Almac. magnético
Celda básica	biestable	condensador	material magnético
Persistencia	necesita alimentación	necesario refresco	permanente
Latencia	$\approx$ 0.5 ns (frec. CPU)	$\approx$ 10 ns	$\approx 10^7$ ns
Coste	Alto (6 transistores)	Medio (1 transistor)	Bajo

Coste

- material para construir una celda
- energía para almacenar un bit
- coste/bit

Tecnologías de memoria



Ninguna cumple los objetivos por sí sola

La jerarquía de memoria

Se busca

- Alta velocidad (latencia y ancho de banda)
- Alta capacidad
- Bajo coste

Solución: jerarquía de memoria

Combinar tecnologías en varios niveles

Funcionará si:

- Memorias rápidas y pequeñas contienen datos con mayor probabilidad de acceso
- Memorias lentas y grandes contienen datos con menor probabilidad de acceso

Principio de localidad

Existe correlación entre los accesos a memoria

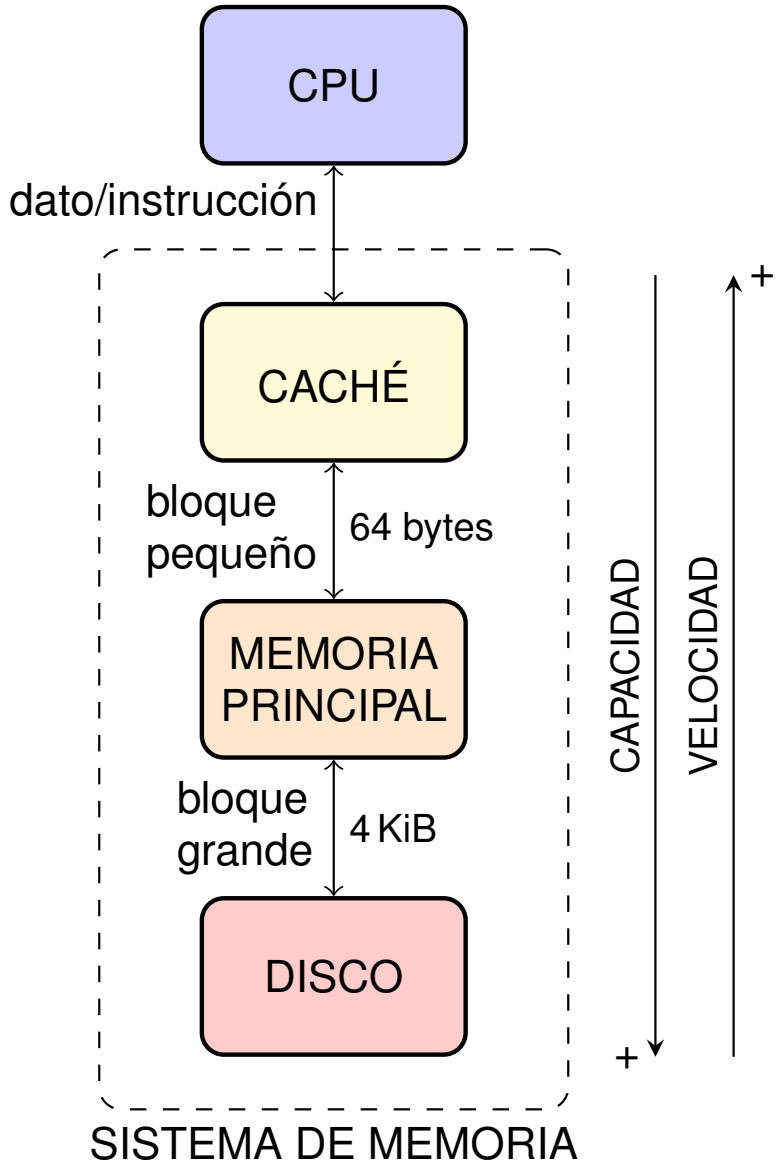
- Tipos de localidad
  - Espacial: acceso a direcciones contiguas
  - Temporal: acceso a direcciones accedidas recientemente
- Presencia
  - Código
    - espacial: ejecución secuencial
    - temporal: ejecución de bucles
  - Datos
    - espacial: acceso a arrays
    - temporal: vbles. control de bucles

- Jerarquía en varios niveles
- Se usan distintas tecnologías de memoria
- Los nieveles más cercanos a la CPU son los más rápidos
- Cuanto más lejos de la CPU, más capacidad

Funcionamiento

1. CPU quiere acceder a un dato
   - acierto caché (*hit*)
   - fallo caché (*miss*)
2. El dato se obtiene de un nivel inferior en la memoria
3. pueden encadenarse fallos

Esquema



Tiempo medio de acceso

Ejemplo

Un computador tiene una jerarquía de memoria: caché, memoria principal y disco. Los tiempos de acceso a cada nivel son:

- $t_c = 1\,\text{ns}$
- $t_p = 5\,\text{ns}$ para cada byte
- $t_d = 11\,\text{ms}$ (para cualquier tamaño de bloque)

Los tasas de acierto medias en cada nivel son:

- $A_c = 0.999 \Rightarrow 99.9\,\%$
- $A_p = 0.9999999 \Rightarrow 99.99999\,\%$

El tamaño de bloque de caché (B_c) es de 64 bytes

¿Cuál es el tiempo medio de acceso al sistema de memoria (t_{cpd})?

$$t_{cpd} = \underbrace{A_c \cdot t_c}_{\text{acierto caché}} + \underbrace{(1 - A_c) \times \left[t_{pd} \underbrace{A_p \cdot t_p \cdot B_c}_{\text{acierto M.P.}} + \underbrace{(1 - A_p) \cdot t_d}_{\text{fallo M.P.}} \right]}_{\text{fallo caché}}$$

Introducción

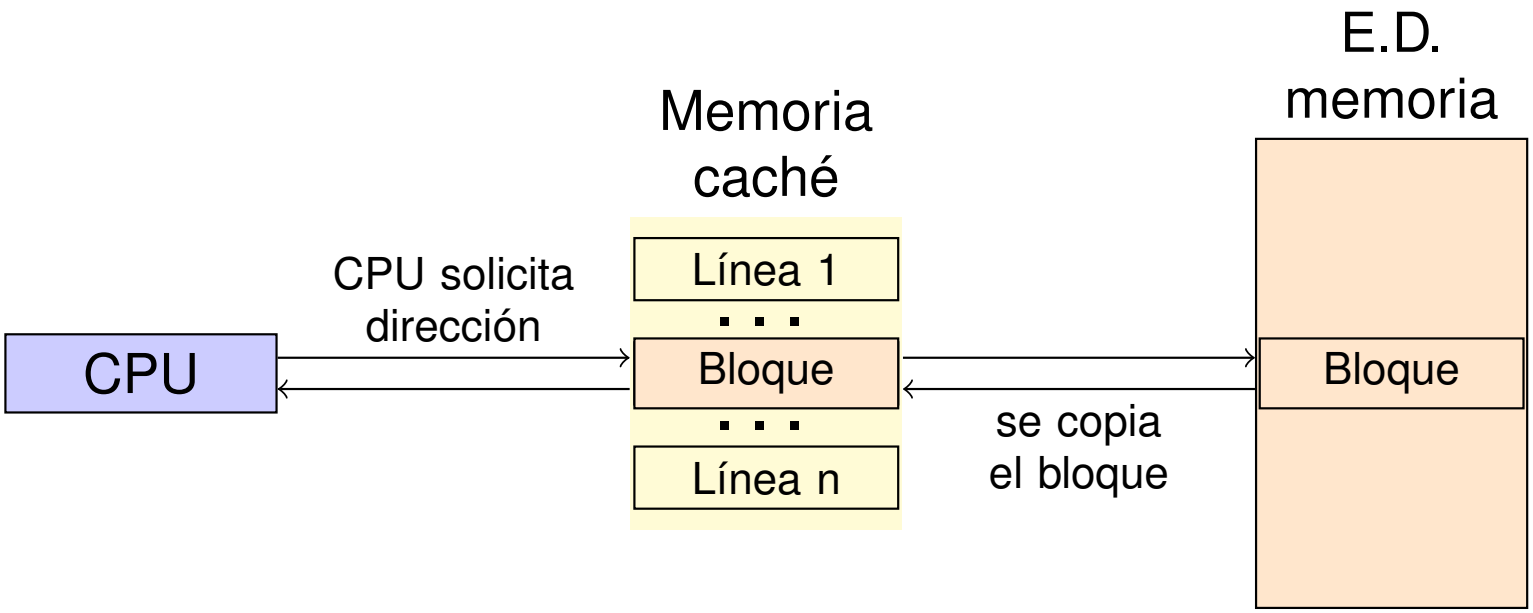
Caché

Primer nivel. Construído usando memoria SRAM

- rápida
- alto coste por bit $\Rightarrow$ baja capacidad

Organizada en líneas (bloque + metadatos)

- contienen datos de posiciones consecutivas de memoria
- todos los bloques tienen la misma capacidad



Organización en bloques

E.D. memoria $\Leftrightarrow$ gran almacén de bloques

- la caché almacena copia de alguno de ellos

Dirección emitida por CPU
- bloque de memoria
- desplazamiento dentro del bloque

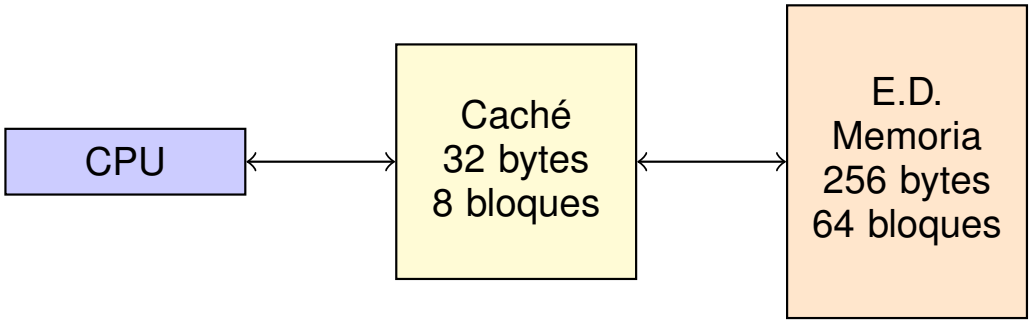
Ejemplo

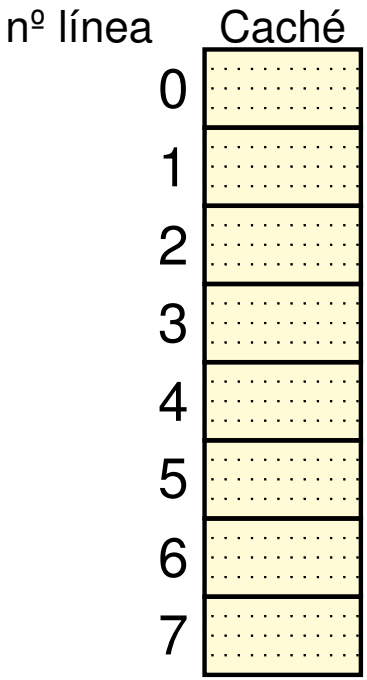
- Palabra de memoria: 1 byte
- Tamaño de caché: 32 bytes
- Tamaño del bloque: 4 bytes

} 8 bloques

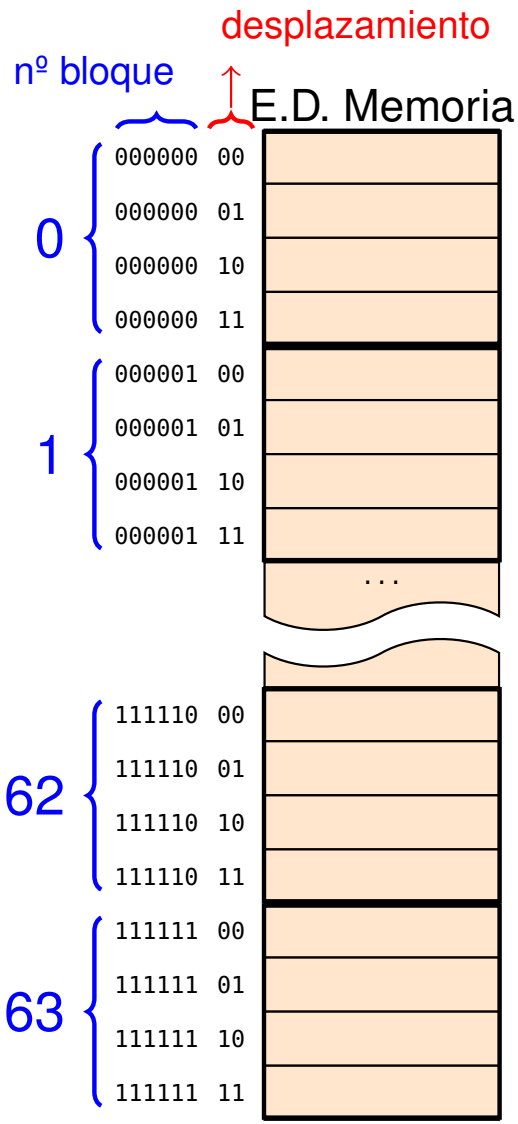
- Tamaño del E.D. memoria: 256 bytes $\Rightarrow$ 64 bloques

Organización en bloques





- Palabra de memoria: 1 byte
- Tamaño de caché: 32 bytes
- Tamaño del bloque: 4 bytes
- Tamaño del E.D. memoria: 256 bytes



Características

Controlador de caché

Gestiona la memoria caché

1. determina si una dirección produce acierto o fallo

También

2. estrategia de correspondencia
   - ¿Dónde ubicar un bloque de memoria en la caché?
3. estrategia de reemplazo
   - ¿Qué bloque sustituir en un fallo en la caché?
4. estrategia de escritura
   - ¿Qué ocurre con las escrituras?

Estrategias de correspondencia

¿Dónde ubicar un bloque de memoria principal?

- correspondencia directa
- correspondencia (totalmente) asociativa
- correspondencia asociativa por conjunto

Correspondencia directa

Modelo de asignación más simple

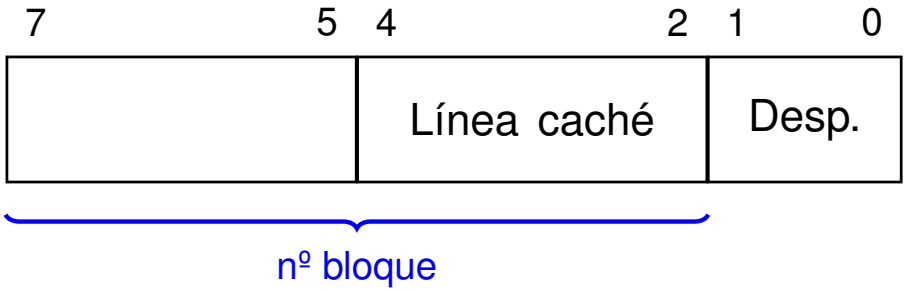
$$\text{línea caché} = (\text{nº bloque}) \text{ MOD } \underbrace{(\text{nº líneas en caché})}_{2^x}$$

$$\Updownarrow$$

línea caché = x bits menos significativos del nº bloque

Ejemplo

Dirección emitida por la CPU



$$\text{FEh} = 1111\,1110 \longrightarrow \overbrace{111}^{\text{Línea caché}}\ \overbrace{111}^{\text{Desp.}}\ 10$$

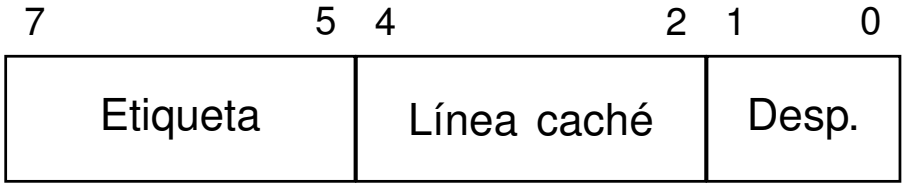
$$\text{33h} = 0011\,0011 \longrightarrow \overbrace{001}^{\text{Línea caché}}\ \overbrace{100}^{\text{Desp.}}\ 11$$

Correspondencia directa

Bits de la dirección

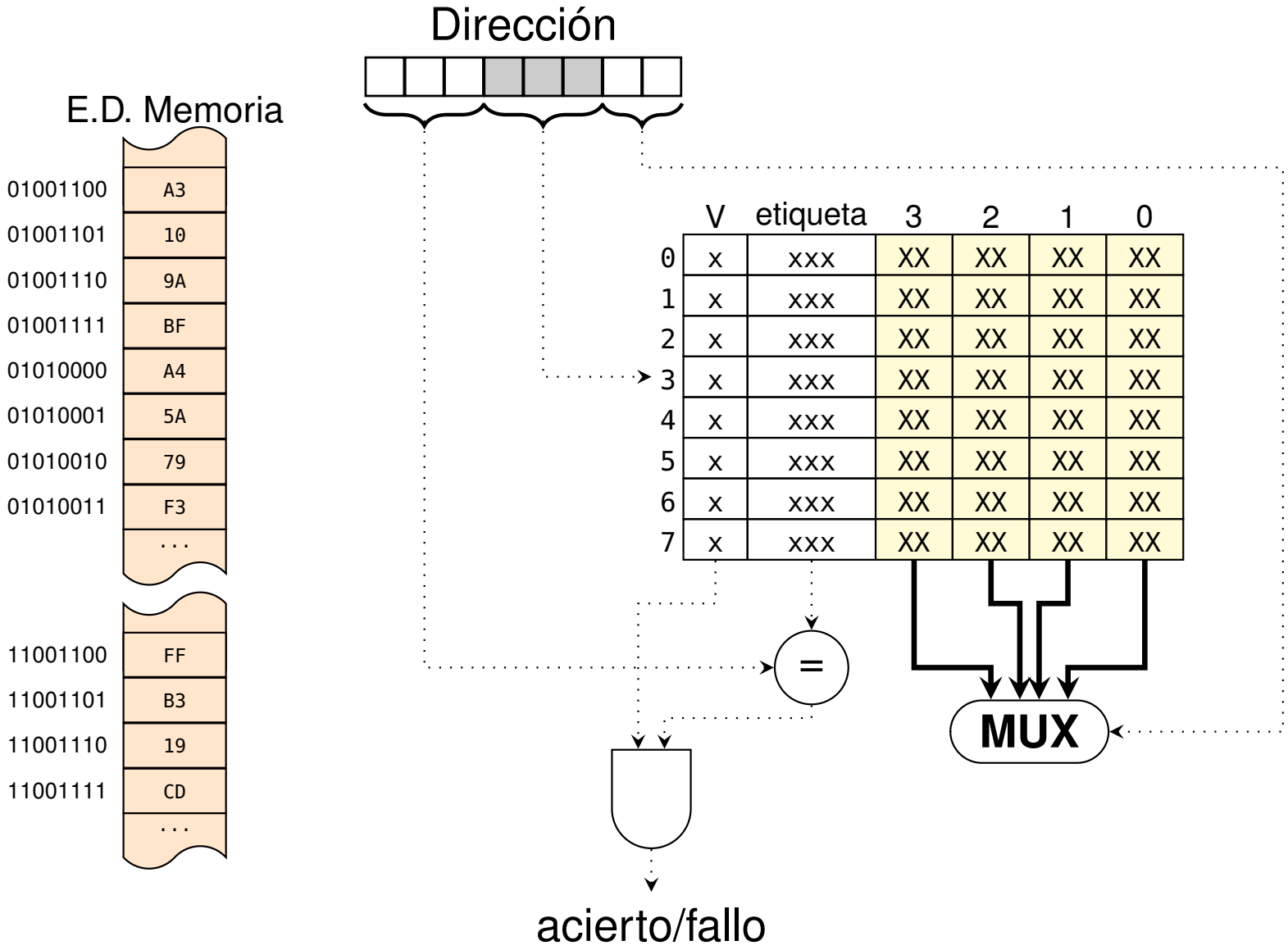
- Desp. $\rightarrow$ Posición de la palabra dentro del bloque
- Línea caché $\rightarrow$ Línea de la caché a la que se asigna
- Etiqueta $\Rightarrow$ identifica el bloque de memoria principal en la caché

Dirección emitida por la CPU

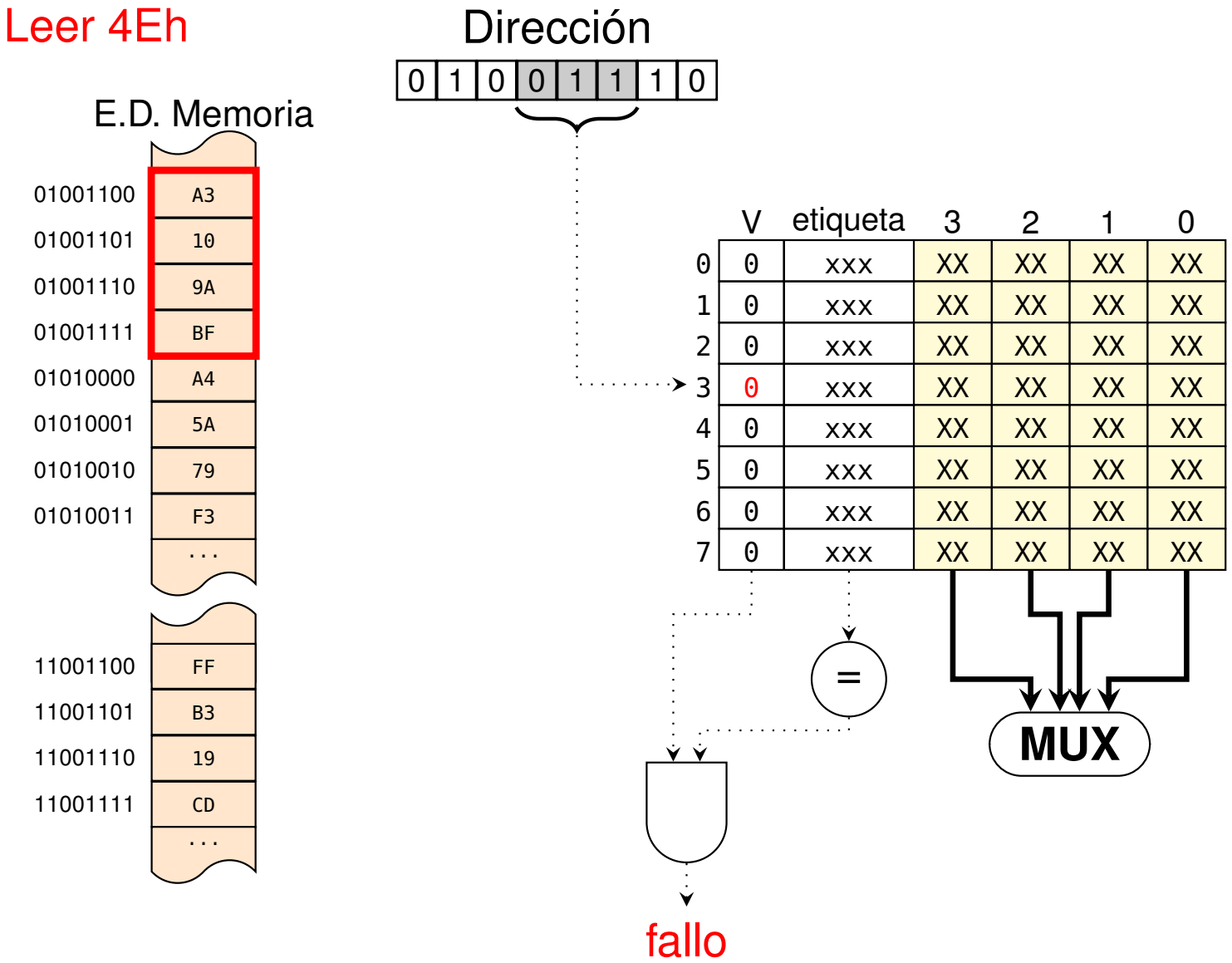


- Se asocia un bit de validez a cada línea

Correspondencia directa



Correspondencia directa



Correspondencia directa

Leer 4Eh

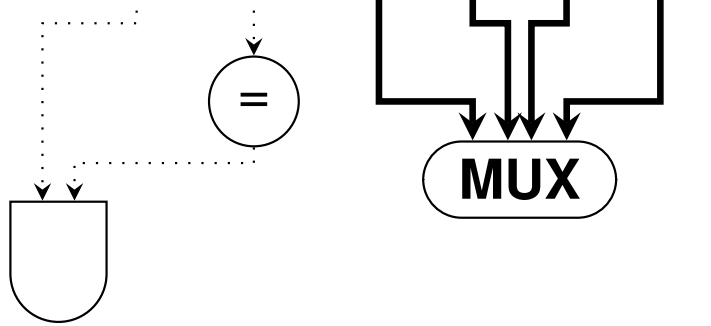
Dirección

0	1	0	0	1	1	1	0

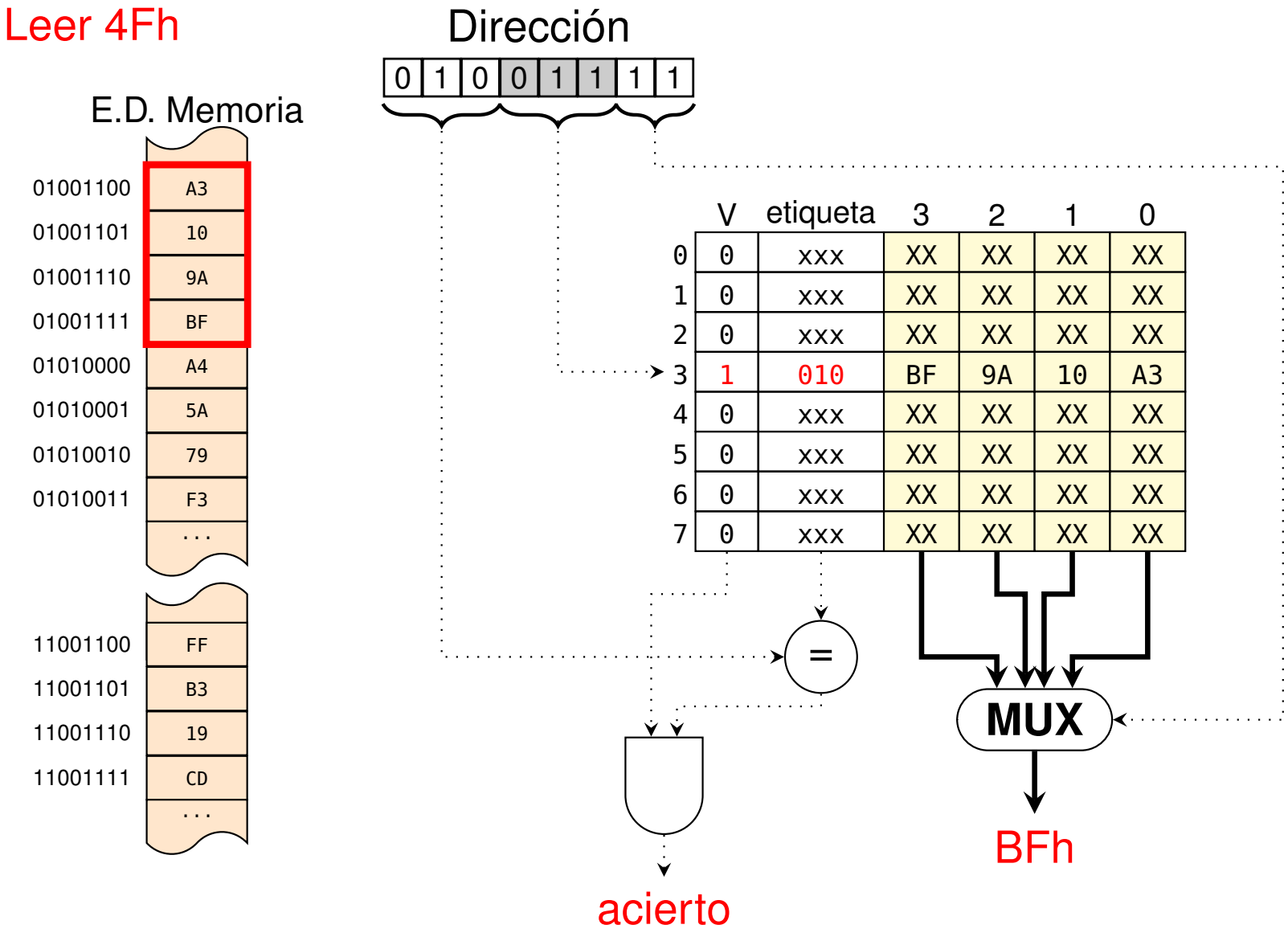
E.D. Memoria

01001100	A3
01001101	10
01001110	9A
01001111	BF
01010000	A4
01010001	5A
01010010	79
01010011	F3
...	
11001100	FF
11001101	B3
11001110	19
11001111	CD
...	

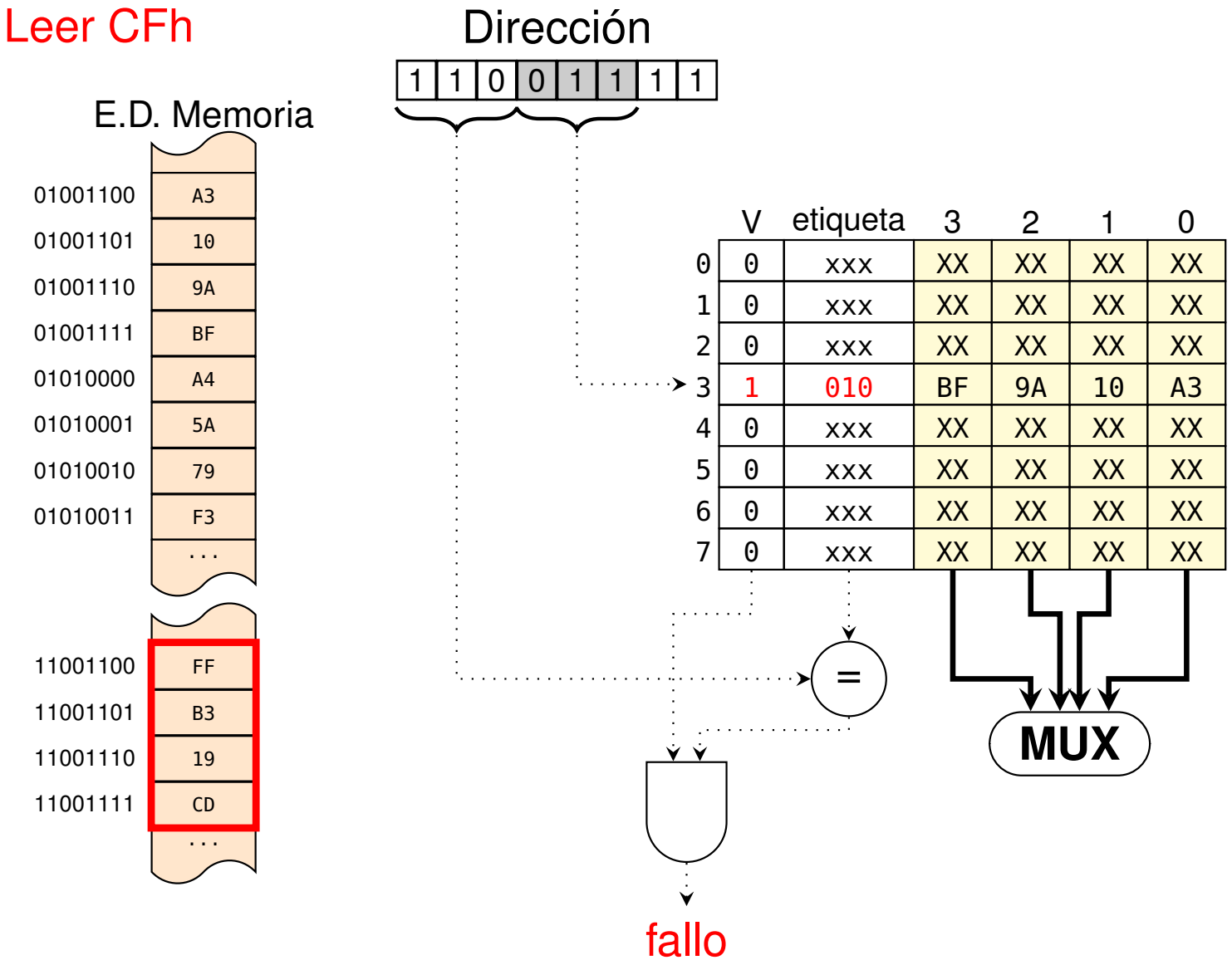
	V	etiqueta	3	2	1	0
0	0	xxx	XX	XX	XX	XX
1	0	xxx	XX	XX	XX	XX
2	0	xxx	XX	XX	XX	XX
3	1	010	BF	9A	10	A3
4	0	xxx	XX	XX	XX	XX
5	0	xxx	XX	XX	XX	XX
6	0	xxx	XX	XX	XX	XX
7	0	xxx	XX	XX	XX	XX



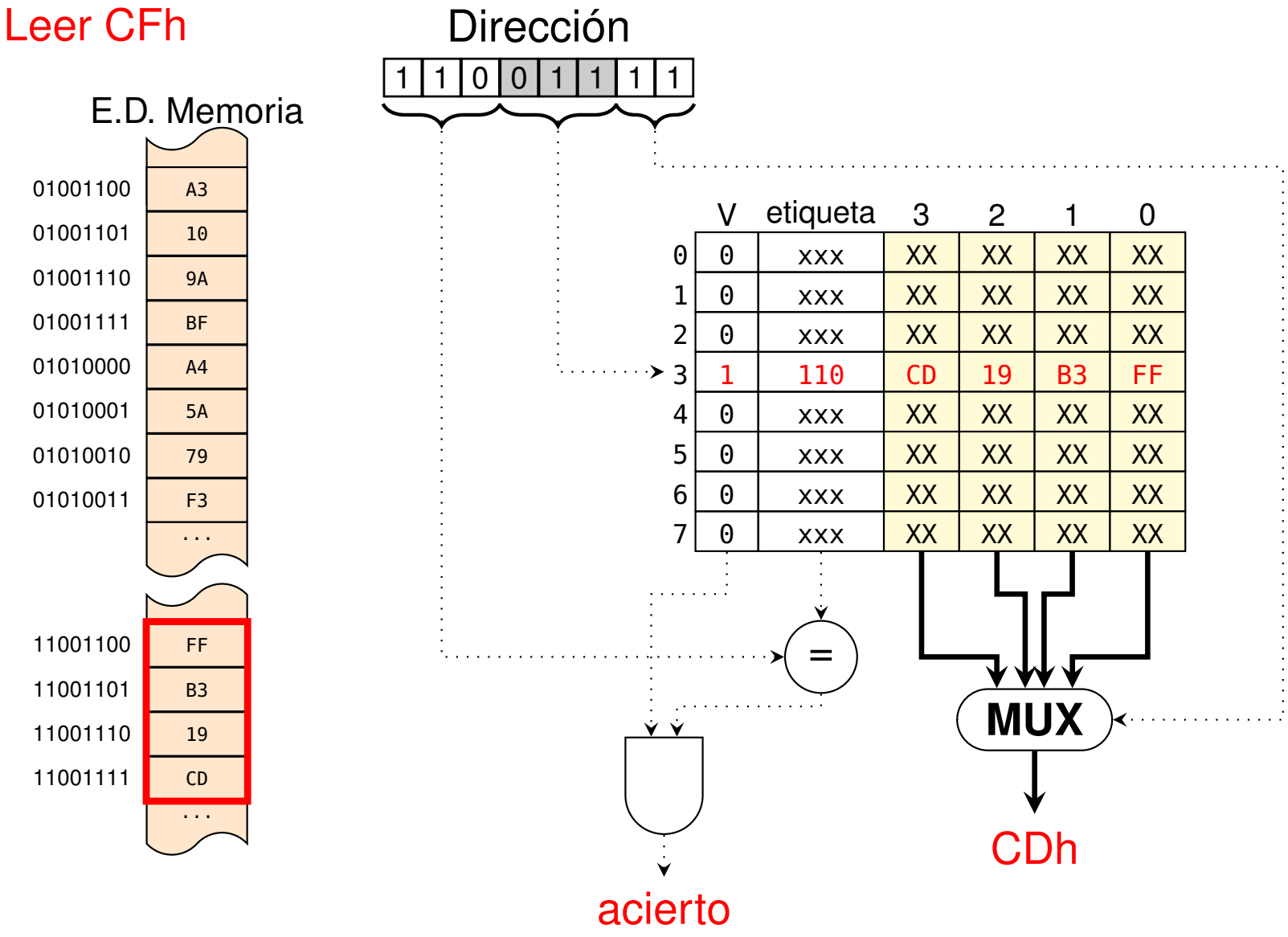
Correspondencia directa



Correspondencia directa



Correspondencia directa



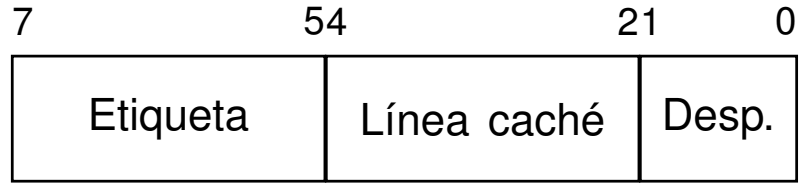
Correspondencia asociativa

Problema de la correspondencia directa

Simple, pero nula flexibilidad

Ejemplo

- Palabra de memoria: 1 byte
- Tamaño de caché: 32 bytes
- Tamaño del bloque: 4 bytes
- Tamaño del E.D. memoria: 256 bytes



$$37\text{h} = 0011\,0111 \longrightarrow 001\,101\,11$$
$$\text{F4h} = 1111\,0100 \longrightarrow 111\,101\,00$$
$$56\text{h} = 0101\,0110 \longrightarrow 010\,101\,10$$

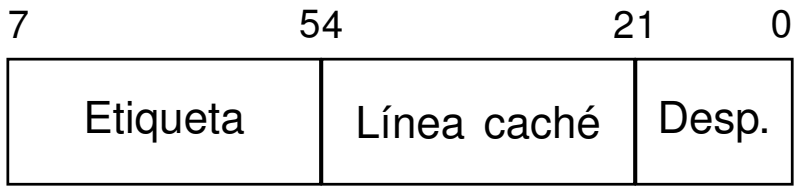
todos a la misma línea de caché

Correspondencia asociativa

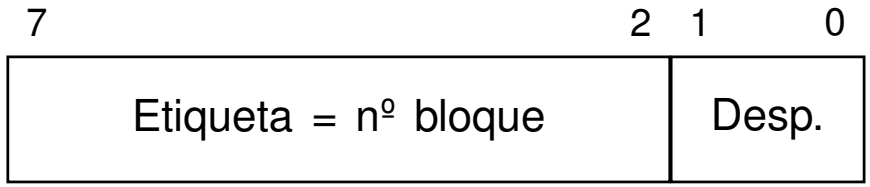
Libertad total

Permite ocupar cualquier línea en caché

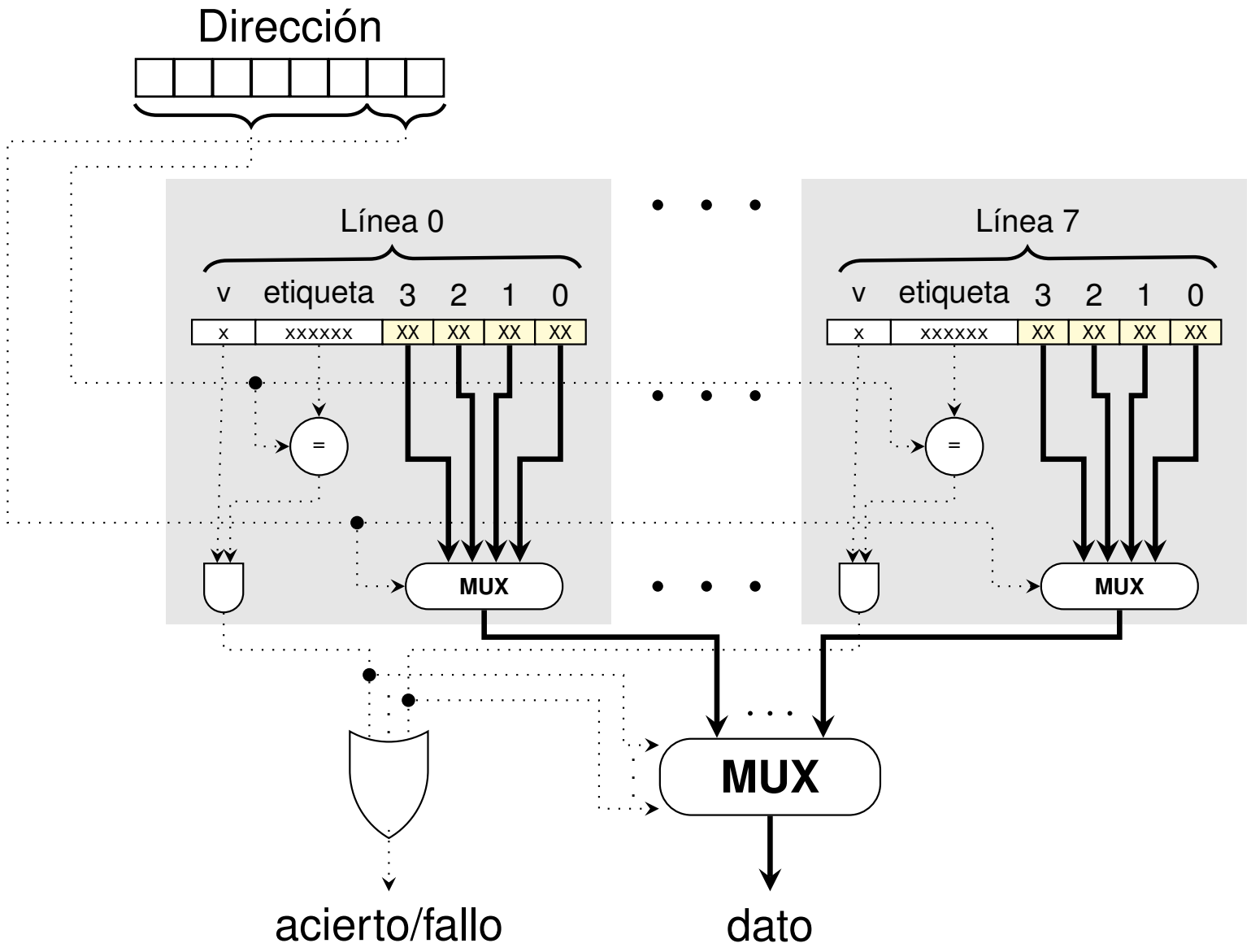
Correspondencia directa



Correspondencia asociativa



Correspondencia asociativa



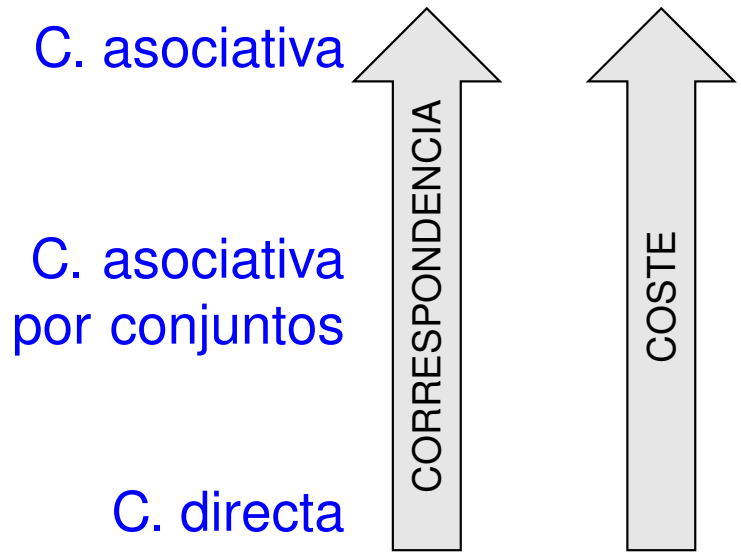
Correspondencia asoc. por conjuntos

Problema de la correspondencia totalmente asociativa

Eficiente pero muy costosa

- número de comparadores

Correspondencia $\begin{cases} \text{directa} \Rightarrow \text{uno para toda la caché} \\ \text{totalmente asociativa} \Rightarrow \text{uno por línea} \end{cases}$



Correspondencia asoc. por conjuntos

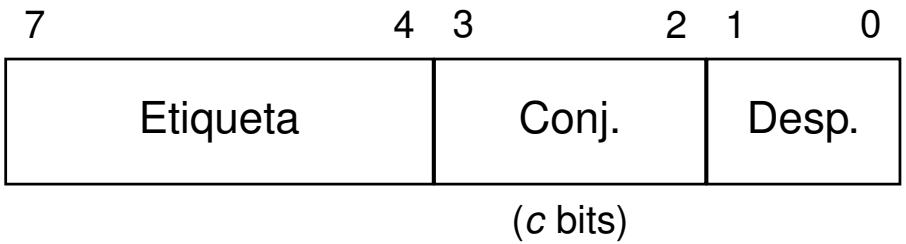
Clave

Agrupación de líneas de caché en conjuntos

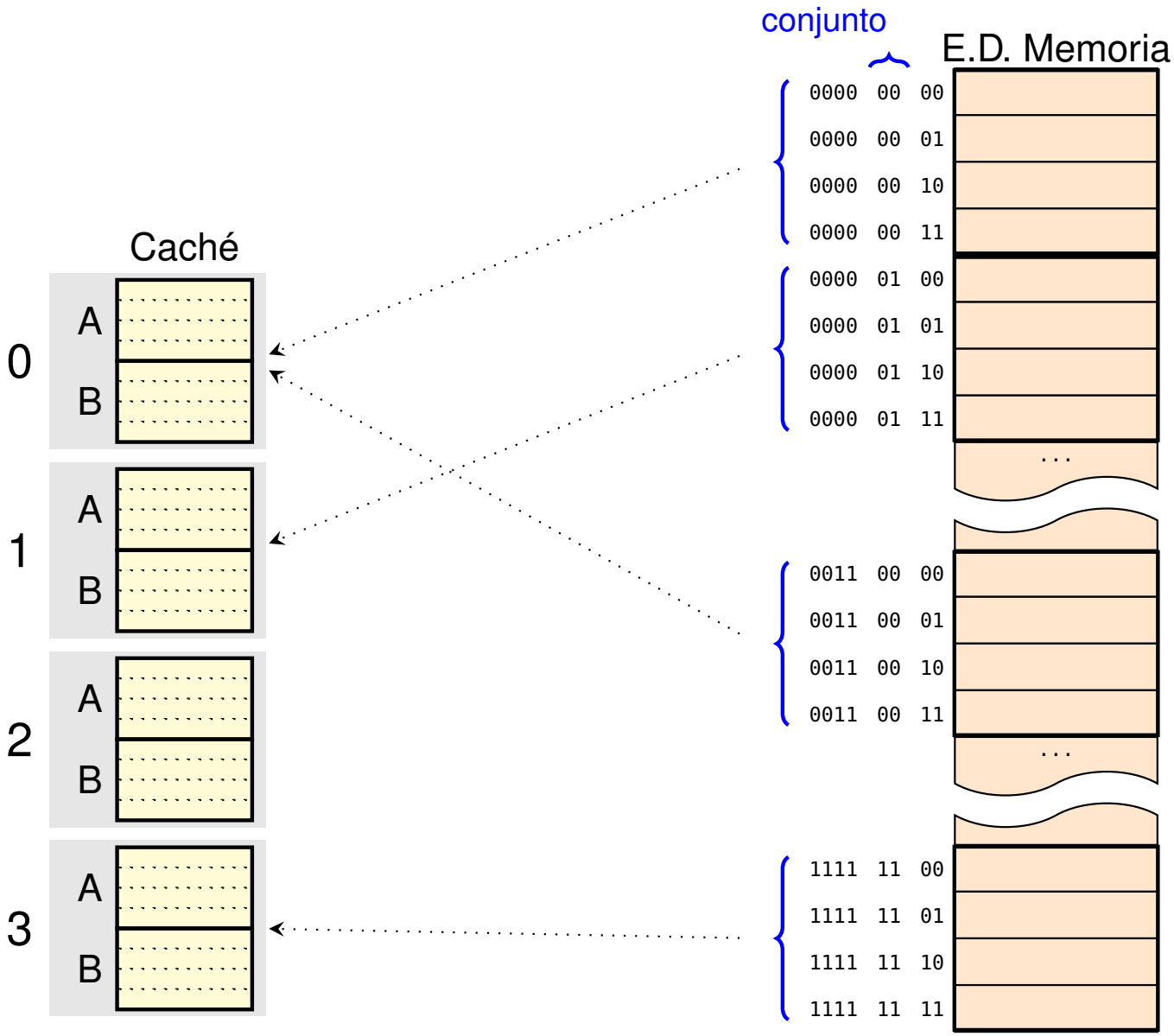
- correspondencia directa al conjunto
- correspondencia totalmente asociativa dentro del conjunto

Ejemplo

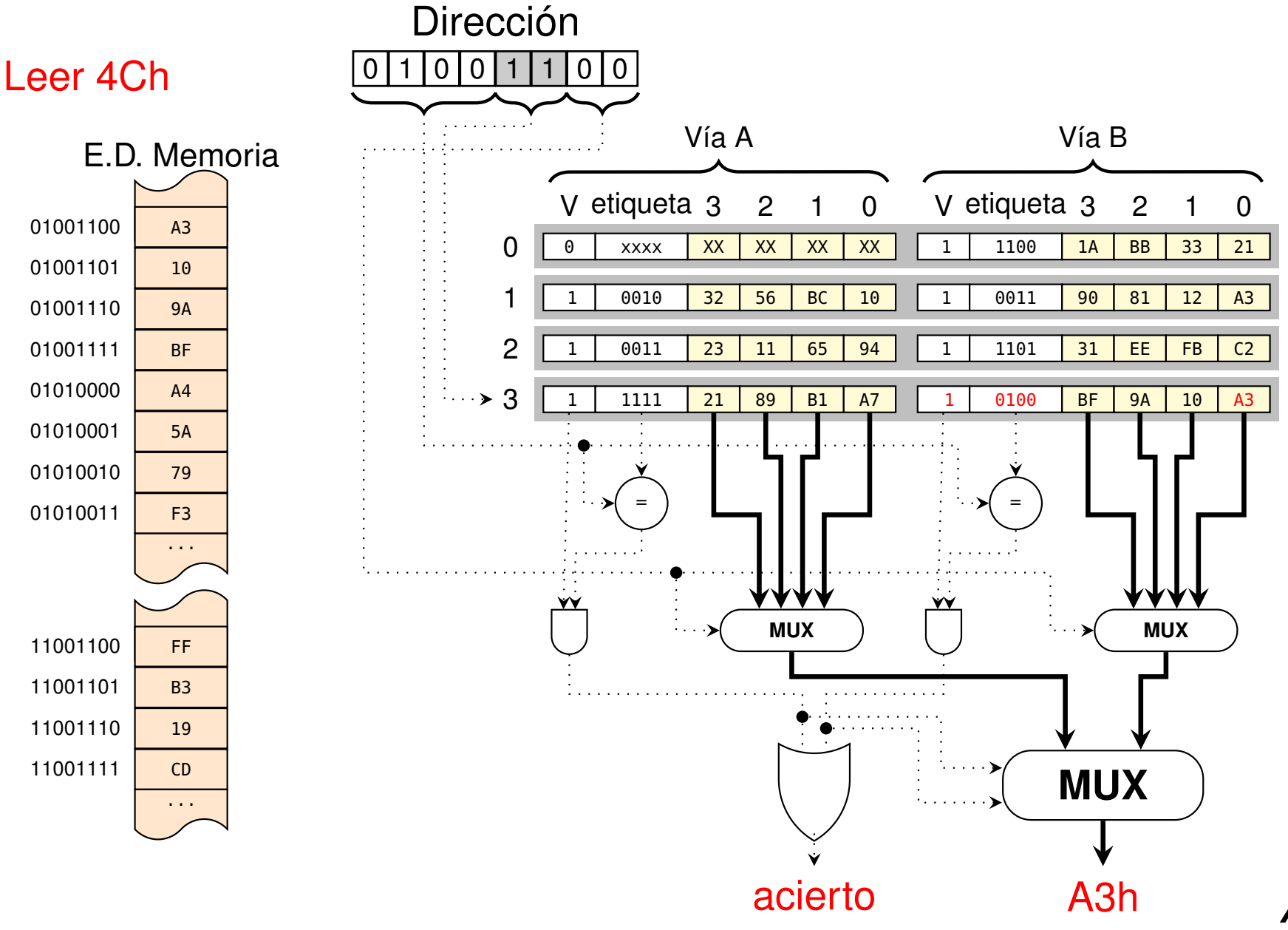
Dirección emitida por la CPU



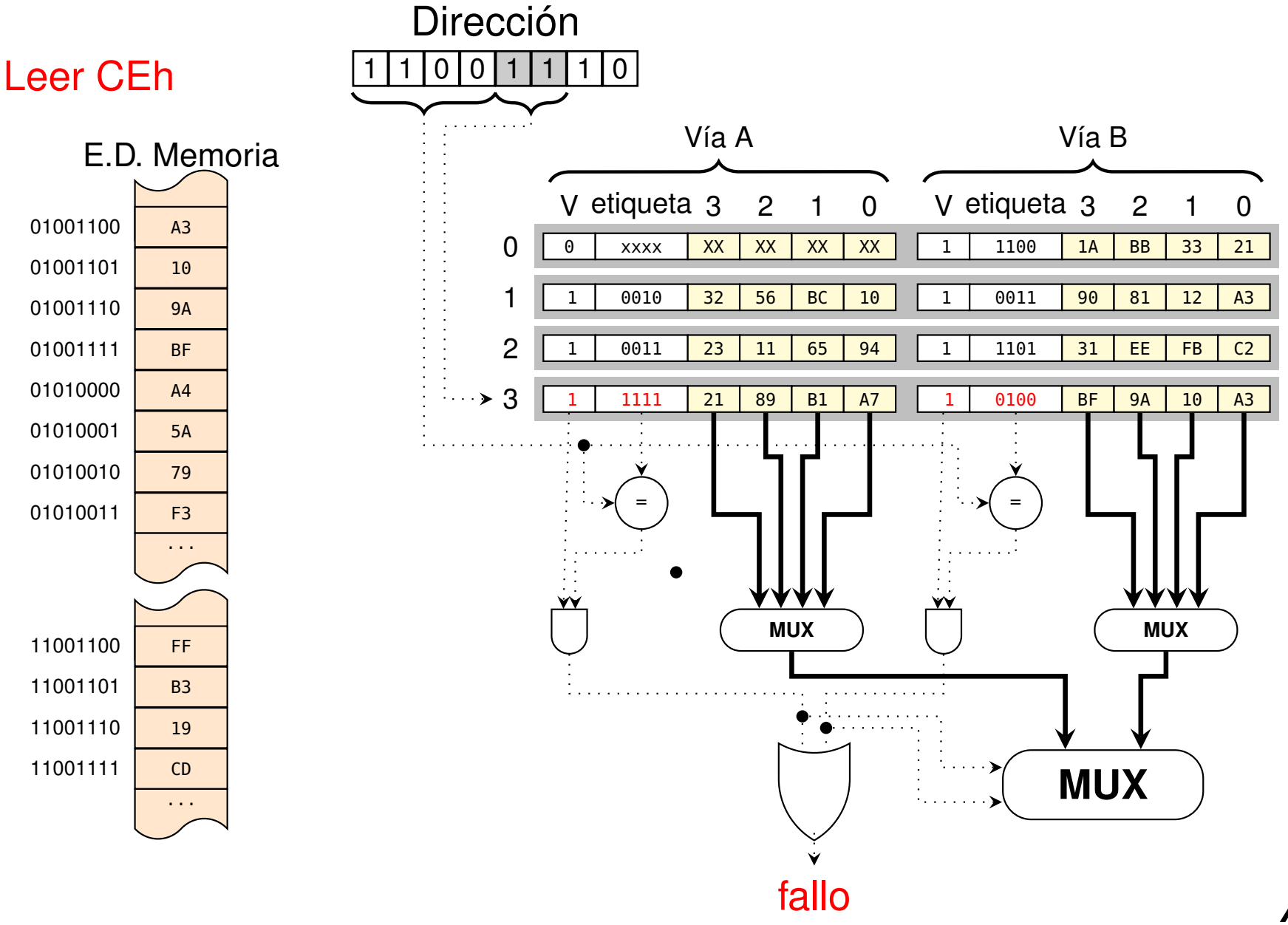
Correspondencia asoc. por conjuntos



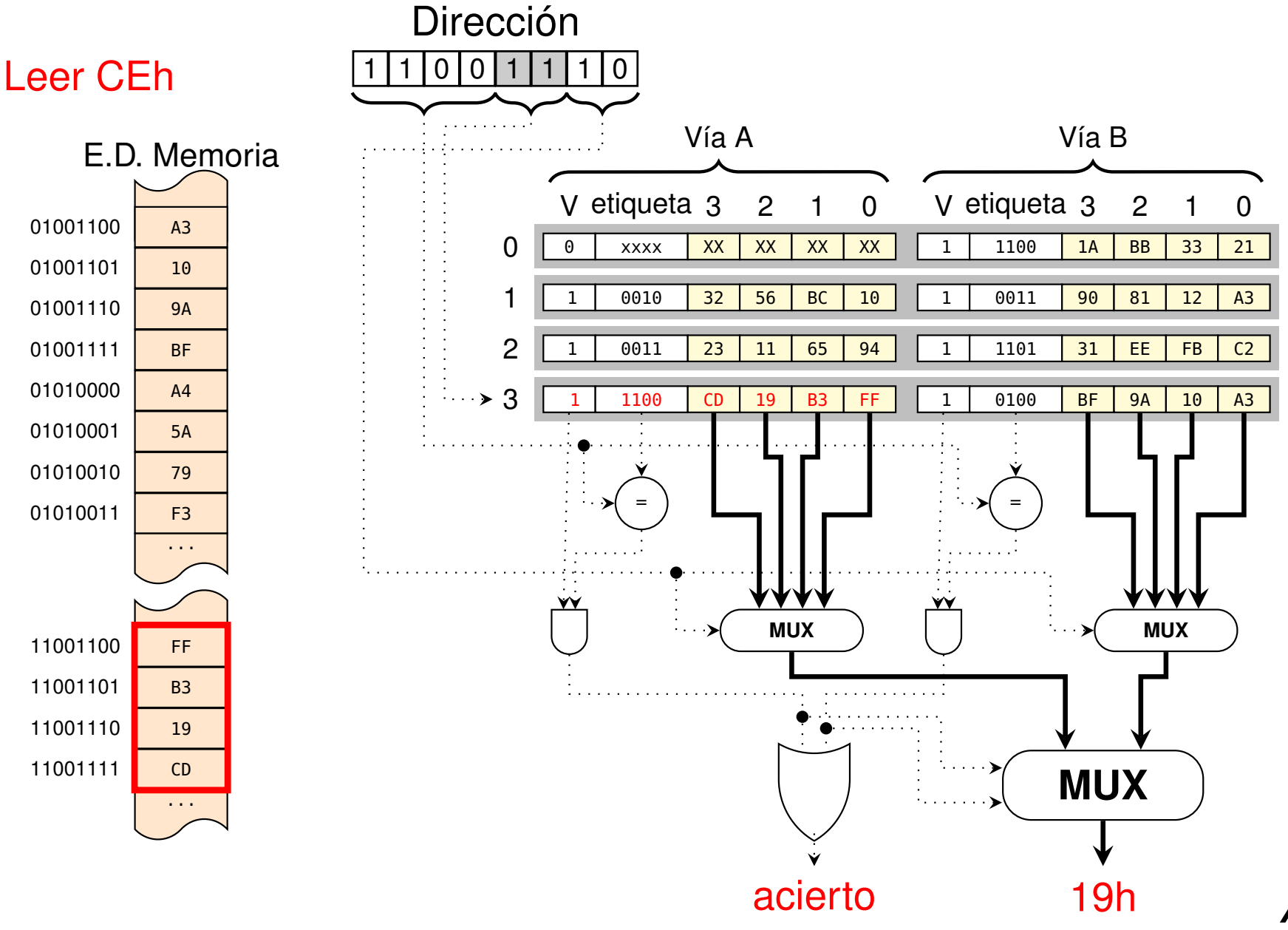
Correspondencia asoc. por conjuntos



Correspondencia asoc. por conjuntos



Correspondencia asoc. por conjuntos



Estrategias de reemplazo

Si hay fallo de caché, ¿qué bloque dejará su sitio?

Elegir el bloque que se reemplaza

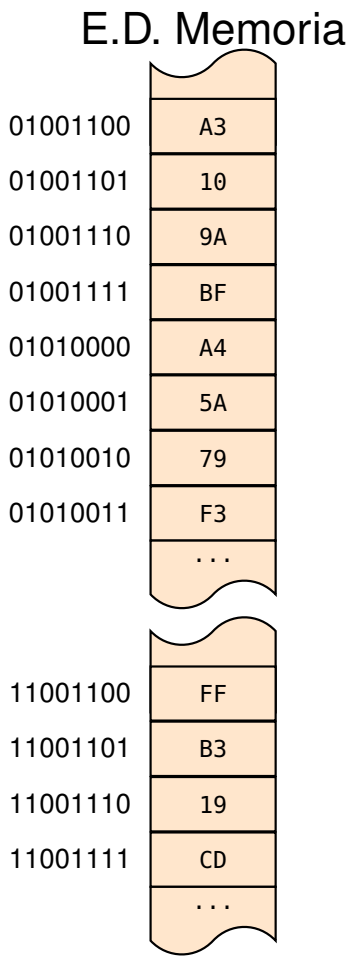
1. existen varias alternativas de ubicación
2. todas las alternativas están ocupadas

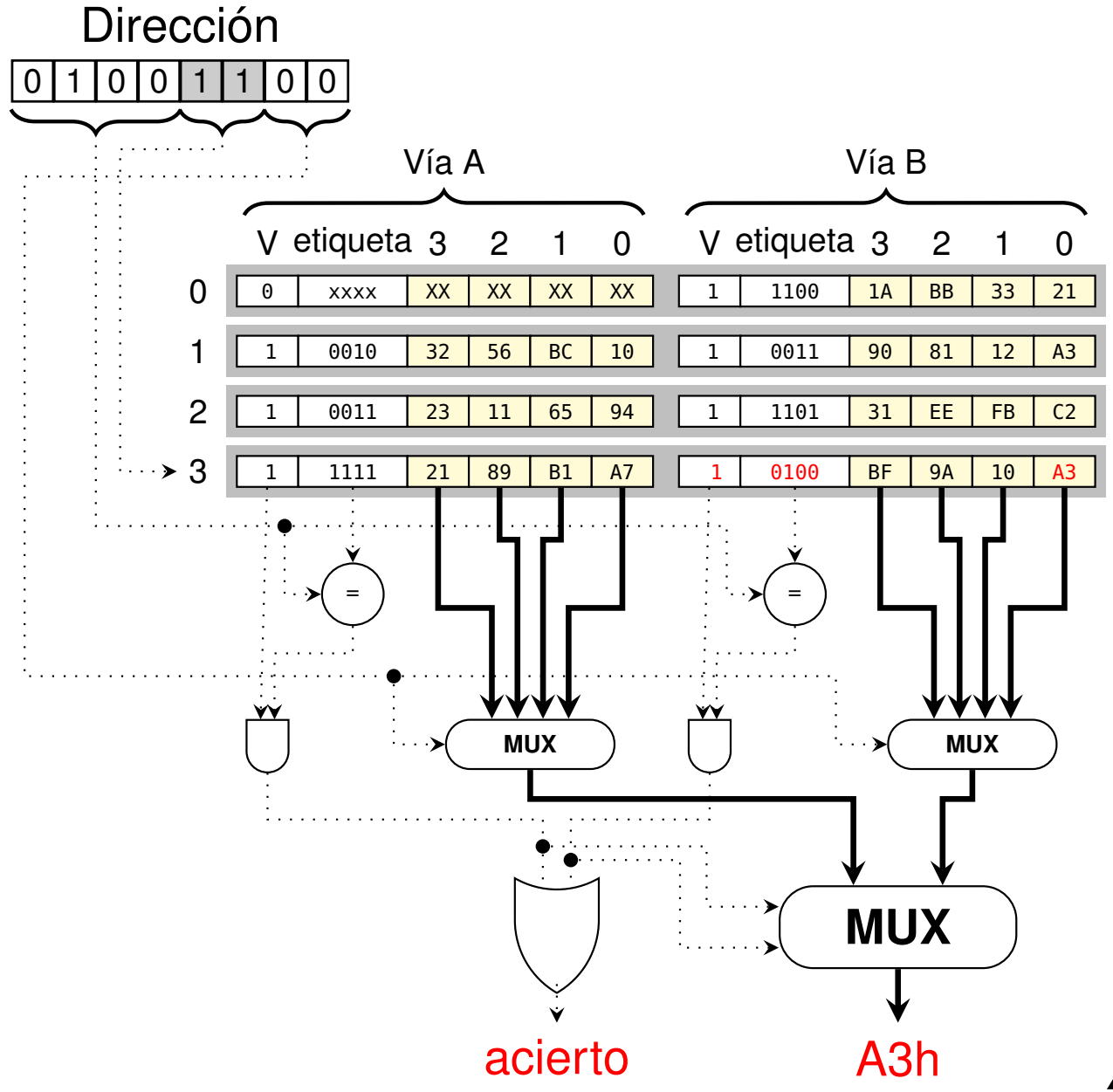
Estrategias

- *Least Recently Used* (LRU) $\Rightarrow$ bloque que más tiempo lleva sin usarse
- Aleatoria $\Rightarrow$ bloque aleatorio entre todos los candidatos

Ejemplo

¿Reemplazo?
Leer 4Ch

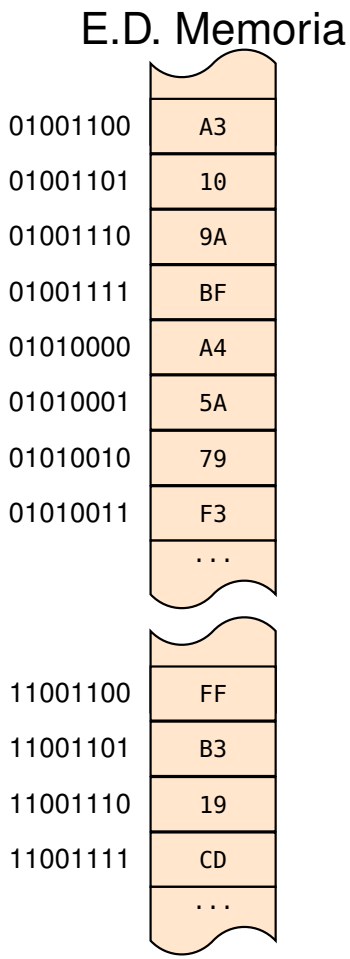


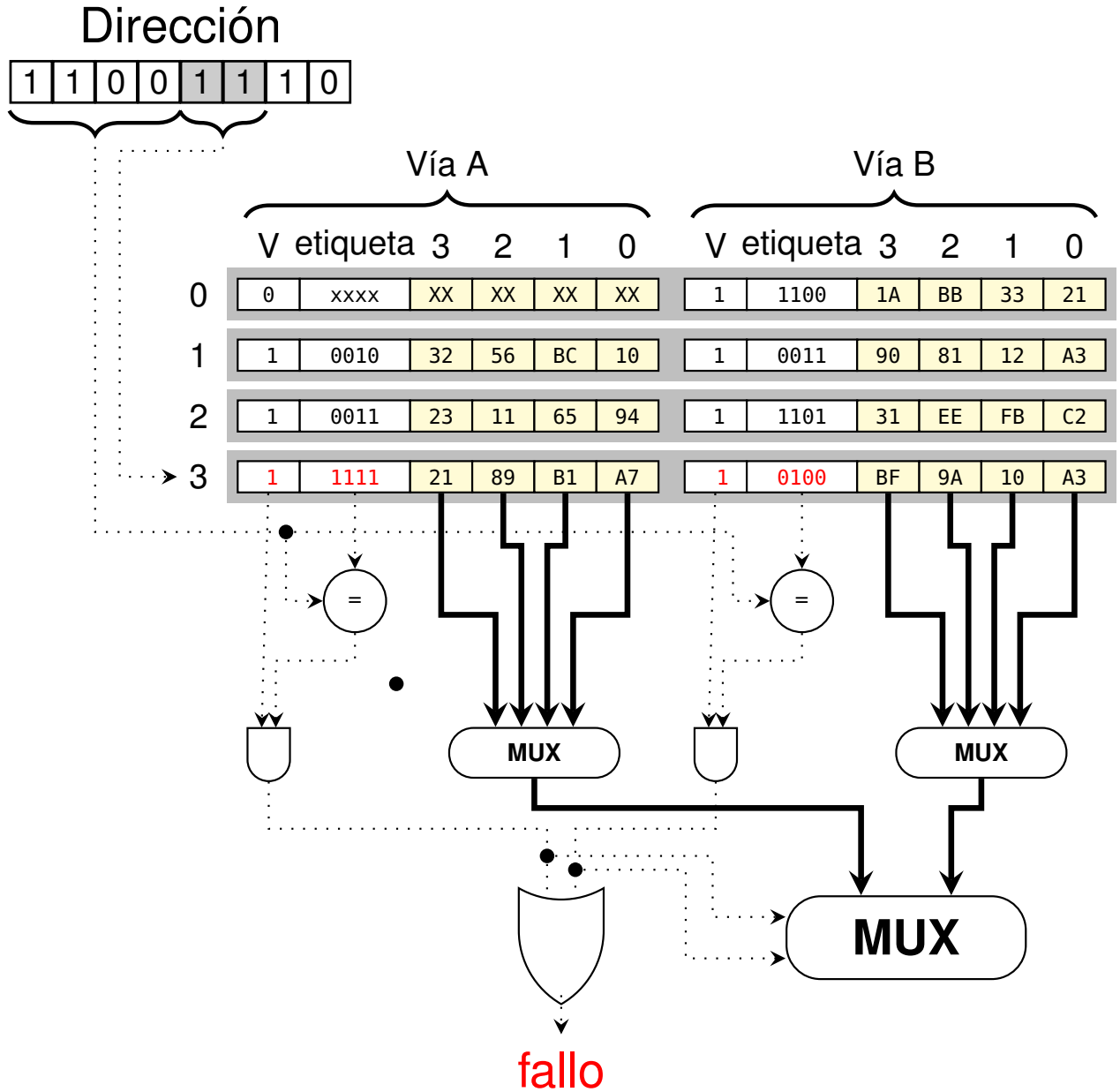


Ejemplo

¿Reemplazo?

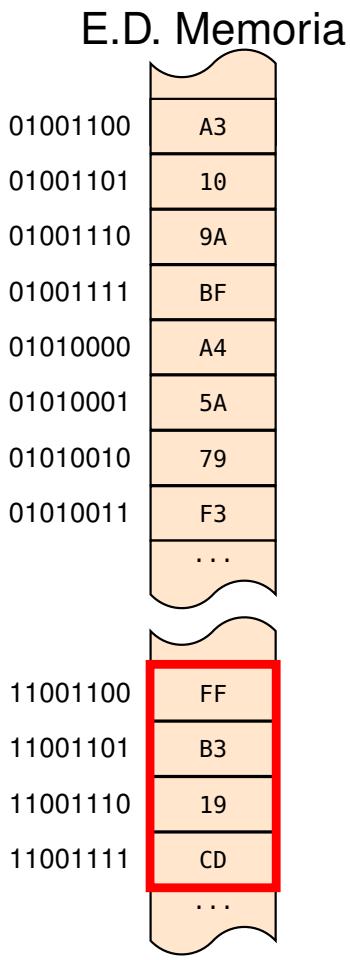
Leer CEh

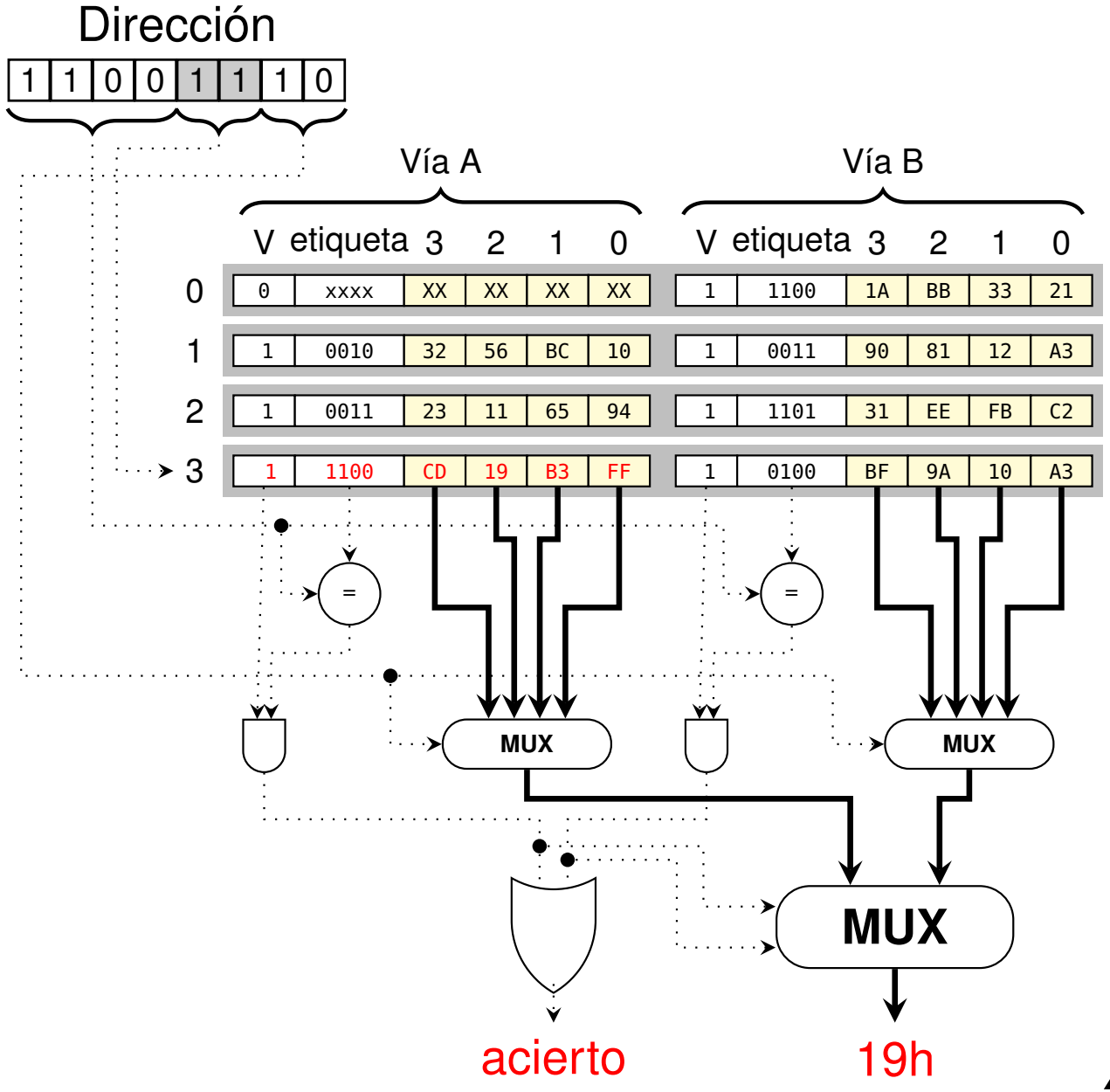




Ejemplo

¿Reemplazo?
Leer CEh





Estrategias de escritura

¿Qué sucede cuando la CPU accede para escribir?

- La información en caché es una copia de niveles inferiores
- La escrituras se pueden cachear o no

Estrategias

- *Write-through* o escritura a través
  - escritura simultánea en varios niveles de la jerarquía
- *Write-back* o diferida
  - bloque se escribe en el siguiente nivel cuando se reemplaza

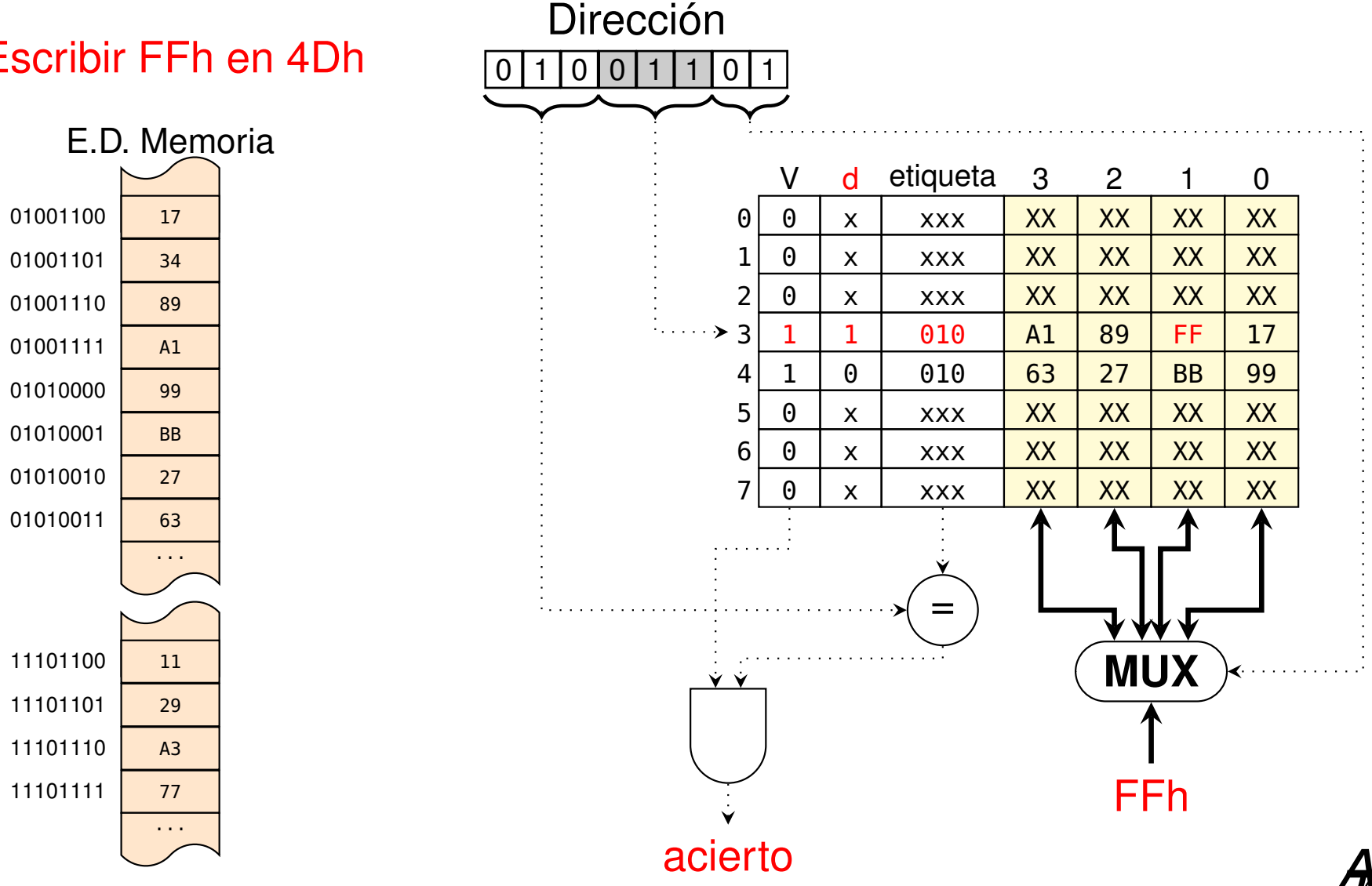
Fallo de caché

- *Write allocate* o asignación en escritura
  - se trae el bloque a la caché y después se escribe
- *No write allocate* o no asignación en escritura
  - solo se escribe en memoria principal (no se cachea)

Escritura *write-back*

Mayor complejidad hardware ⇒ bit *dirty*

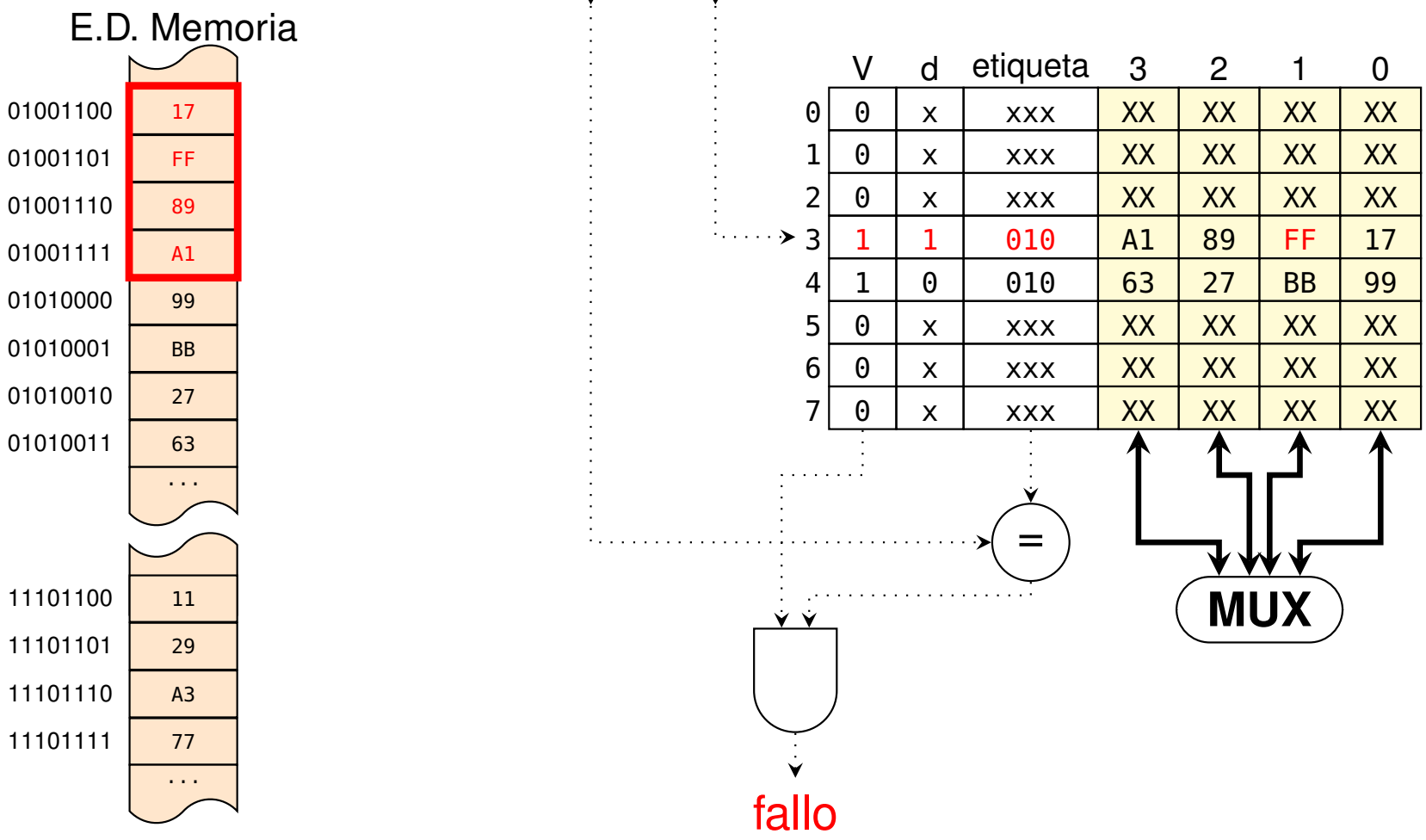
Escribir FFh en 4Dh



Escritura *write-back*

Mayor complejidad hardware ⇒ bit *dirty*

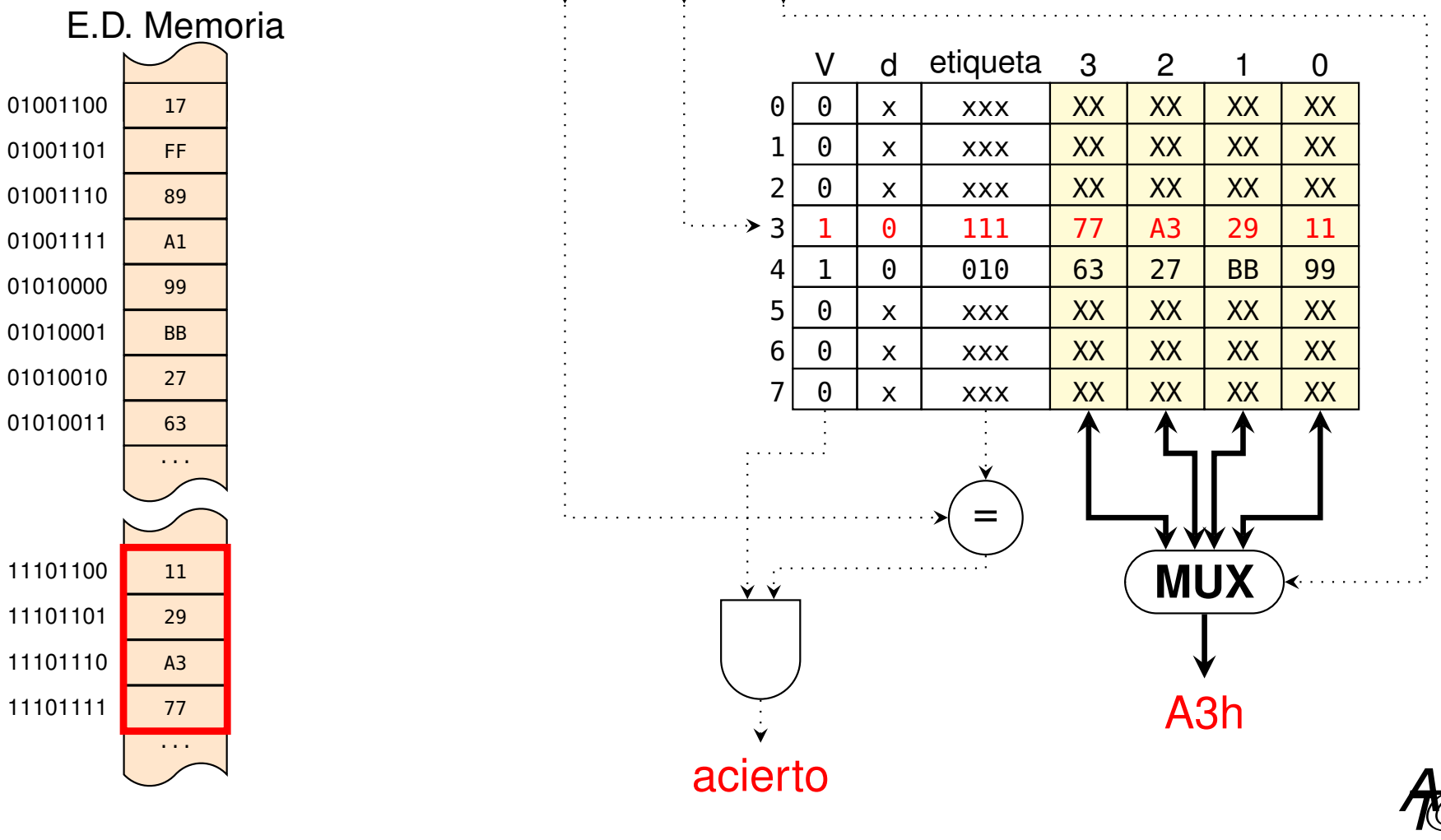
Leer EEh



Escritura *write-back*

Mayor complejidad hardware ⇒ bit *dirty*

Leer EEh



Organización de la caché

- Numero de niveles de caché
- Tipo de información almacenada

Niveles de cachés

Problema

Gran diferencia de velocidad entre la caché y la memoria principal

Solución

Compromiso entre la velocidad de acceso y la tasa de aciertos
Usando varios niveles de caché $\Rightarrow$ reduce la penalización de rendimiento

- se denominan L1, L2, L3, etc.
- velocidad y capacidad intermedias
- normalmente, tres niveles (L3)

Tipo de información

Caché unificada

Una sola caché para almacenar todo tipo de información

- ✔ es más simple, un único hardware
- ✔ aprovecha mejor los bloques
- ✔ mayor tasa de aciertos

Caché dividida

Una caché para datos y otra para código

- ✘ hardware duplicado
- ✘ capacidad de cada caché fija
- ✘ menor tasa de aciertos
- ✔ accesos paralelos $\Rightarrow$ L1 es siempre dividida

¿Cuál obtiene mejor rendimiento?

Problema de coherencia

- CPU trabaja siempre con la caché
- la caché guarda copias de E.D. memoria

} Posición de memoria con distintos valores

¿Y si algún otro dispositivo accede al E.D. memoria?

Pueden aparecer problemas de coherencia

1. se modifica el E.D. memoria. $\Rightarrow$ CPU accede a un dato obsoleto
2. CPU escribe en caché $\Rightarrow$ dato en E.D. memoria obsoleto

Problema de coherencias

Dos escenarios

1. Un único nivel de caché con una única CPU
2. Múltiples niveles de caché con múltiples CPUs

¿Qué dispositivos acceden al E.D. memoria?

1. Interfaces E/S mapeadas en el E.D. memoria
2. Interfaces E/S con acceso directo a memoria (DMA)

Soluciones

1. Marcar zonas como no cacheables
   - no óptimo para interfaces con DMA
2. Espionaje del bus (*snooping*)
   - observa líneas de control y direcciones
   - detiene a la interfaz para deshacer incoherencias

Problemas de coherencia

Interfaz lee del E.D. memoria

- ✔ *Write-through* $\Rightarrow$ no hay problema
- ✘ *Write-back* $\Rightarrow$ problema si el bloque está cacheado y sucio

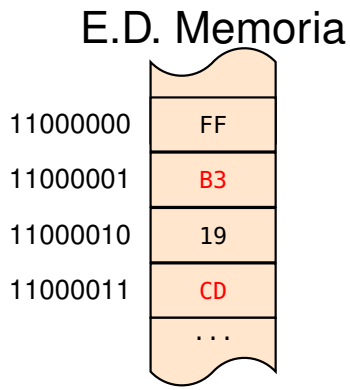
Interfaz escribe en E.D. memoria

- ✘ *Write-through* $\Rightarrow$ problema si el bloque está cacheado
- ✘ *Write-back* $\Rightarrow$ problema si el bloque está cacheado (además puede estar sucio)

Interfaz lee + *write-back*

El bloque a leer está marcado como sucio

1. bloque de caché está sucio $\Rightarrow$ incoherente con E.D. memoria
2. periférico solicita leer sobre el bloque (11000010)
3. controlador de caché detiene la lectura y actualiza el bloque
4. se permite la lectura

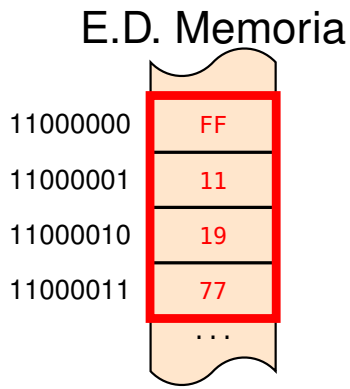


	V	d	etiqueta	3	2	1	0
0	1	1	110	77	19	11	FF

Interfaz lee + *write-back*

El bloque a leer está marcado como sucio

1. bloque de caché está sucio $\Rightarrow$ incoherente con E.D. memoria
2. periférico solicita leer sobre el bloque (11000010)
3. controlador de caché detiene la lectura y actualiza el bloque
4. se permite la lectura

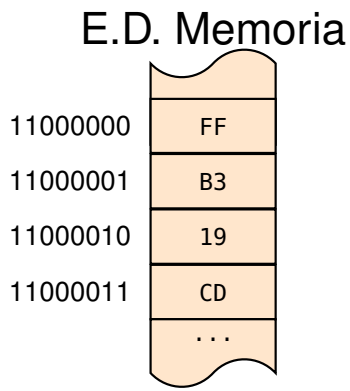


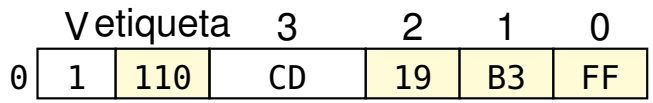
	V	d	etiqueta	3	2	1	0
0	1	0	110	77	19	11	FF

Interfaz escritura + *write-through*

El bloque está cacheado

1. bloque cacheado y siempre coherente con E.D. memoria
2. periférico solicita escribir sobre el bloque (11000011)
3. controlador de caché permite la escritura
4. se invalida la línea de en caché

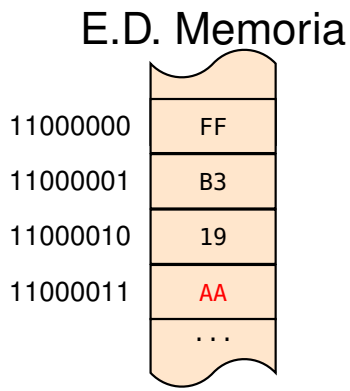


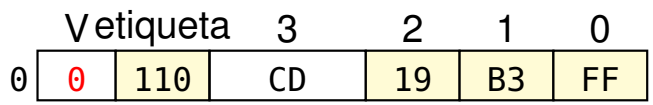


Interfaz escritura + *write-through*

El bloque está cacheado

1. bloque cacheado y siempre coherente con E.D. memoria
2. periférico solicita escribir sobre el bloque (11000011)
3. controlador de caché permite la escritura
4. se invalida la línea de en caché

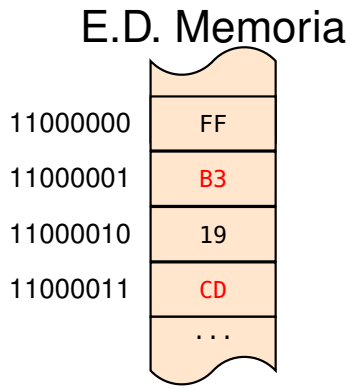




Interfaz escritura + *write-back*

El bloque está cacheado (y sucio)

1. bloque de caché está sucio $\Rightarrow$ incoherente con E.D. memoria
2. periférico solicita escribir sobre el bloque (11000011)
3. controlador de caché detiene la escritura y actualiza el bloque
4. se permite la escritura y se invalida la línea de caché

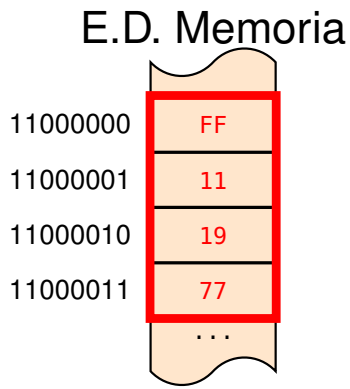


	V	d	etiqueta	3	2	1	0
0	1	1	110	77	19	11	FF

Interfaz escritura + *write-back*

El bloque está cacheado (y sucio)

1. bloque de caché está sucio $\Rightarrow$ incoherente con E.D. memoria
2. periférico solicita escribir sobre el bloque (11000011)
3. controlador de caché detiene la escritura y actualiza el bloque
4. se permite la escritura y se invalida la línea de caché

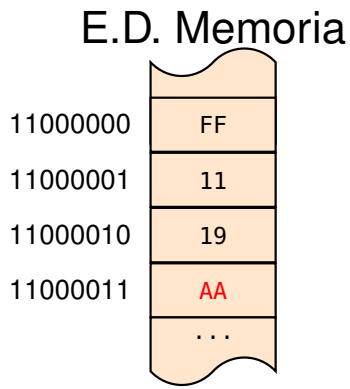


	V	d	etiqueta	3	2	1	0
0	1	0	110	77	19	11	FF

Interfaz escritura + *write-back*

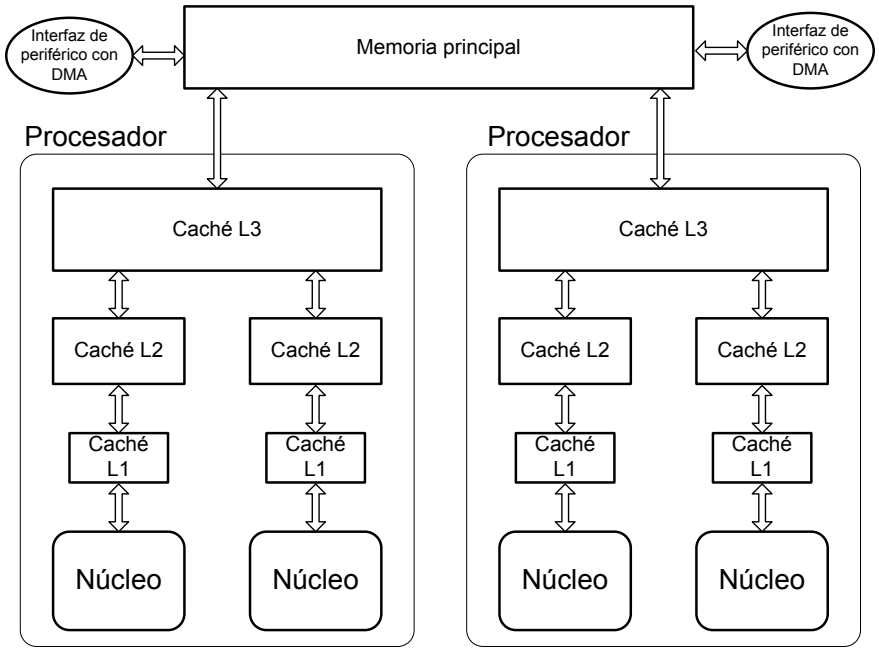
El bloque está cacheado (y sucio)

1. bloque de caché está sucio $\Rightarrow$ incoherente con E.D. memoria
2. periférico solicita escribir sobre el bloque (11000011)
3. controlador de caché detiene la escritura y actualiza el bloque
4. se permite la escritura y se invalida la línea de caché



	V	d	etiqueta	3	2	1	0
0	0	0	110	77	19	11	FF

Coherencia en multiprocesadores



Alternativas

Impacto en rendimiento ⇒ transmitir información coherencia

- Actualizar en escritura
- Invalidar en escritura

Escritura ⇒ cambio en varias cachés

Transmitir coherencia

Snooping

- Operaciones visibles a las cachés
- Núcleo escribe en posición
- Se invalida en otras cachés si cacheada

Directorio

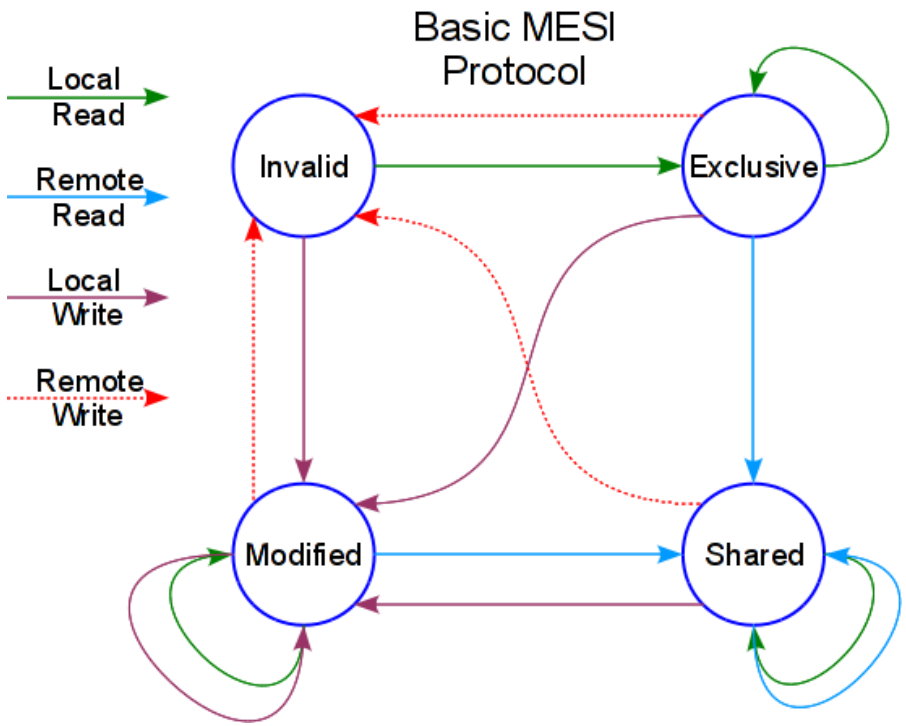
Dispositivo adicional gestiona coherencia

- Toda operación consulta el directorio
- Puede modificar su estado

Protocolo MESI

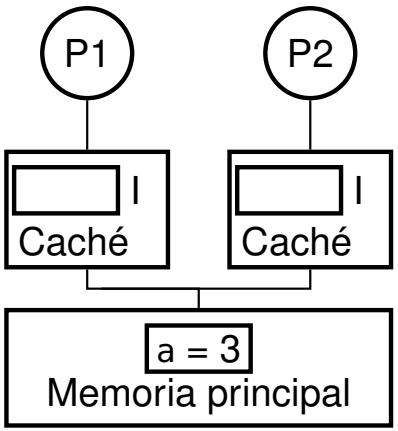
Características

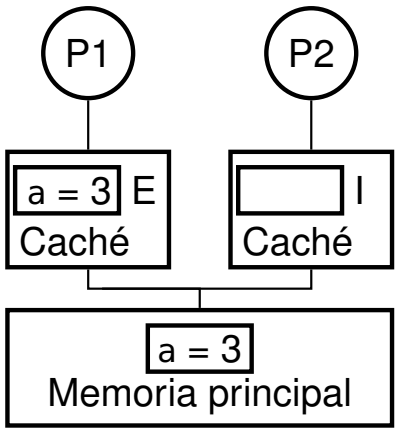
- Write-invalidate
- Basado en *snooping*
- Define estados para la línea de caché
- Base de otros protocolos

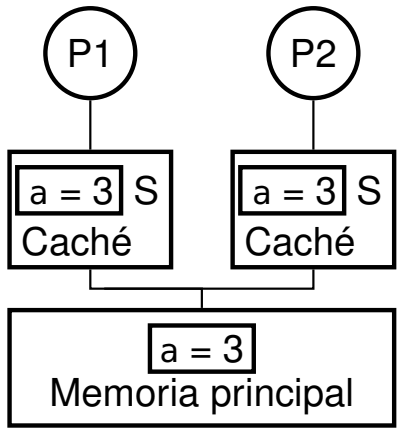


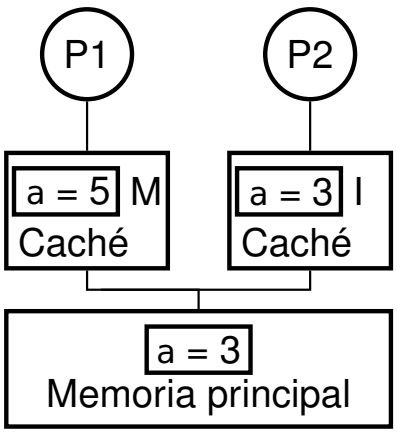
Protocolo MESI

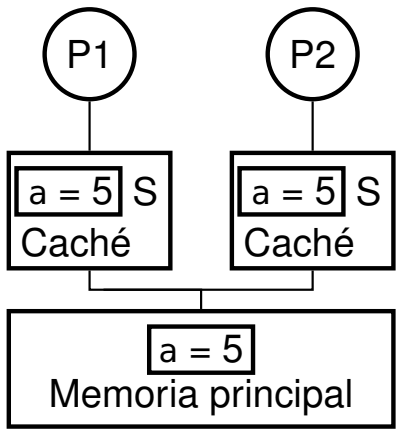
Ejemplo











Ejemplo: Intel Core i5 6600K

Caché L1 dividida: datos (4x) y código (4x)

- 4×32 KiB
- 64 bytes por línea
- 8 vías

} 64 conjuntos

Caché L2 unificada (4x)

- 4×256 KiB
- 64 bytes por línea
- 4 vías

} 1024 conjuntos

Caché L3 unificada

- 6 MiB
- 64 bytes por línea
- 12 vías

} $2^{13} = 8192$ conjuntos

Introducción

Memoria virtual

Utilizar la memoria principal como caché del disco

- solo una parte del disco usada como nivel de la jerarquía
- reducir latencia
- principio de localidad

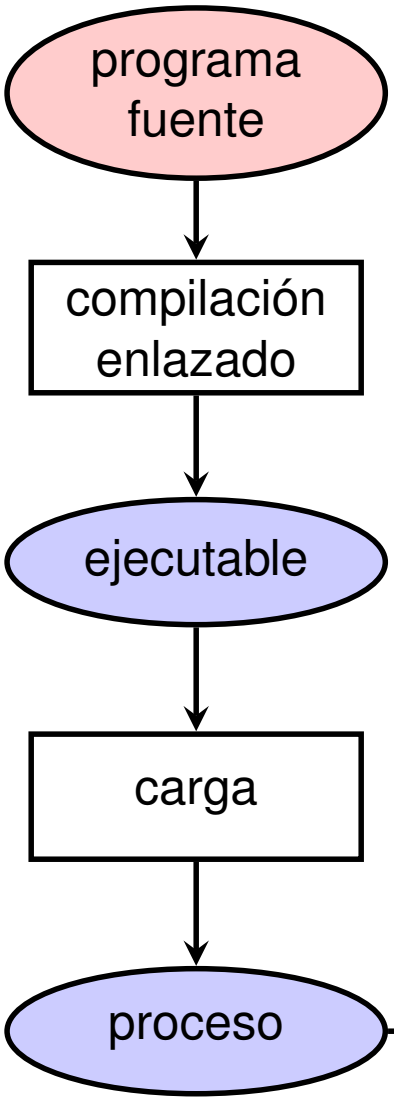
Ventajas

- aumentar el tamaño del sistema de memoria
- incrementar su funcionalidad

Limitaciones

- aumenta la complejidad del sistema de memoria
- necesario soporte del S.O.

Asociación de direcciones



Direcciones simbólicas

Cadenas de caracteres

- nombres de variable

Direcciones reubicables

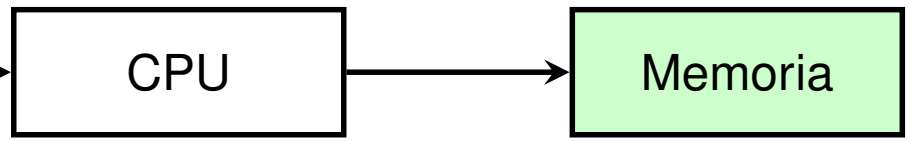
Espacio de direcciones propio del proceso

- direcciones virtuales

Direcciones absolutas

Posición absoluta de un dato en memoria

- llegan a los dispositivos de memoria



Objetivos

- O1 Ampliar la capacidad del sistema de memoria
- O2 Soportar la protección de memoria
- O3 Compartir áreas de memoria
- O4 Simplificar las herramientas de desarrollo y carga

O1: Ampliación de la memoria

Utilizar memoria como caché del disco

- ejecutar programas más grandes que la memoria instalada
- ejecutar muchos programas simultáneamente

Zona del disco

- Windows $\Rightarrow$ archivo de paginación
- Unix/Linux $\Rightarrow$ partición de intercambio

Direcciones virtuales

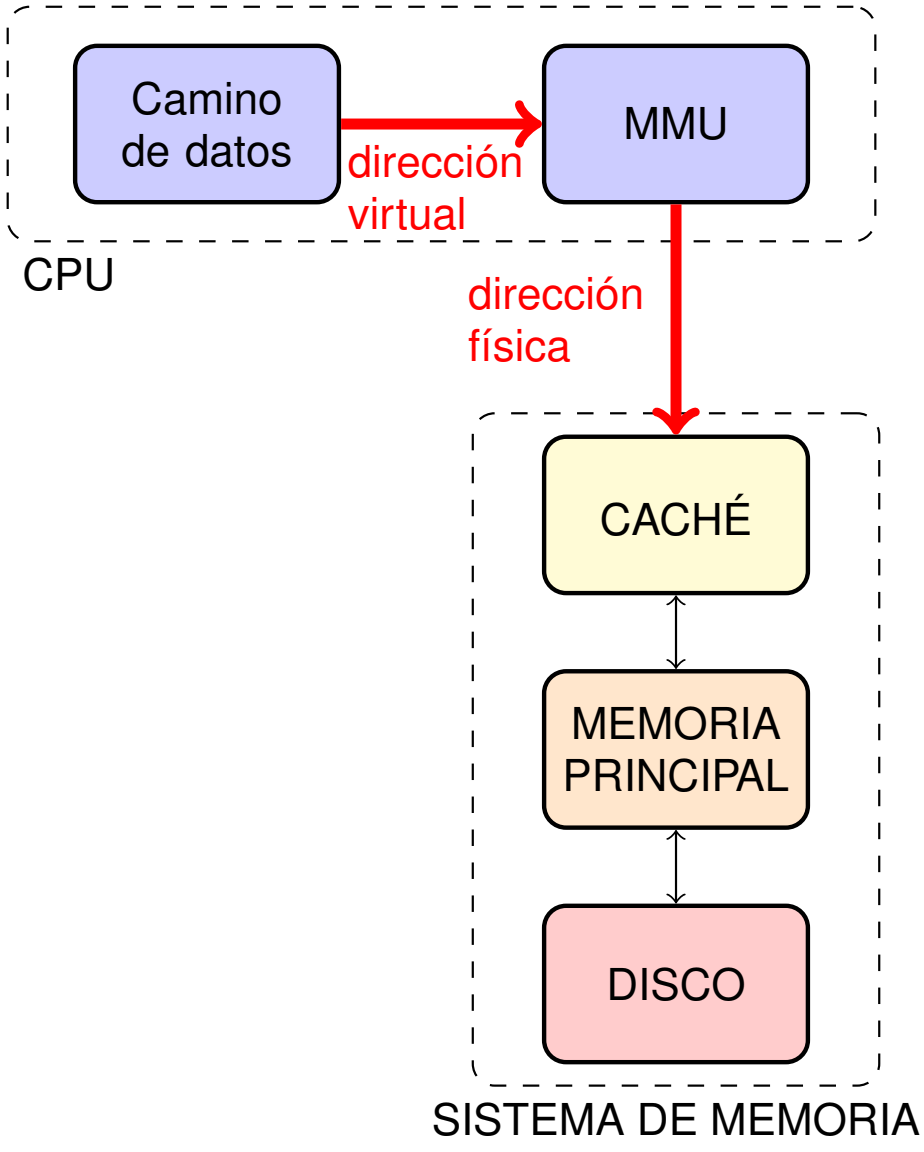
Dos tipos de direcciones

- virtuales: usadas por programas
- físicas: usadas en el sistema de memoria

¿Cómo traducir?

Memory Management Unit

- hardware
- utiliza tablas de traducción



Espacios de direcciones

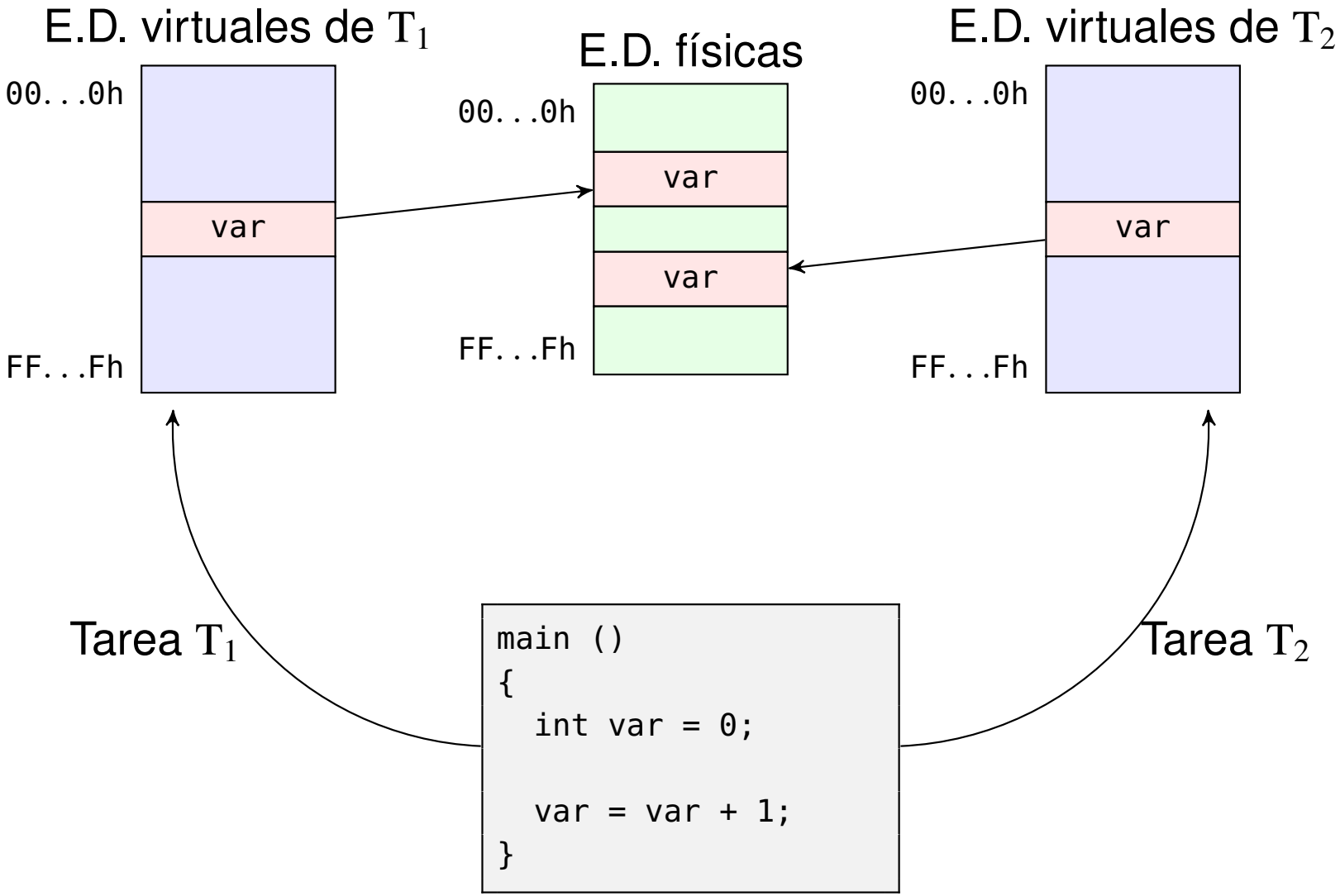
Espacio de direcciones virtuales

- conjunto de todas las direcciones virtuales posibles
- fijo, depende de la capacidad de direccionamiento de la CPU
- uno completo para cada tarea

Espacio de direcciones físicas

- conjunto de todas las direcciones físicas posibles
- variable, depende de la memoria física instalada
- único para todo el computador

Ejemplo

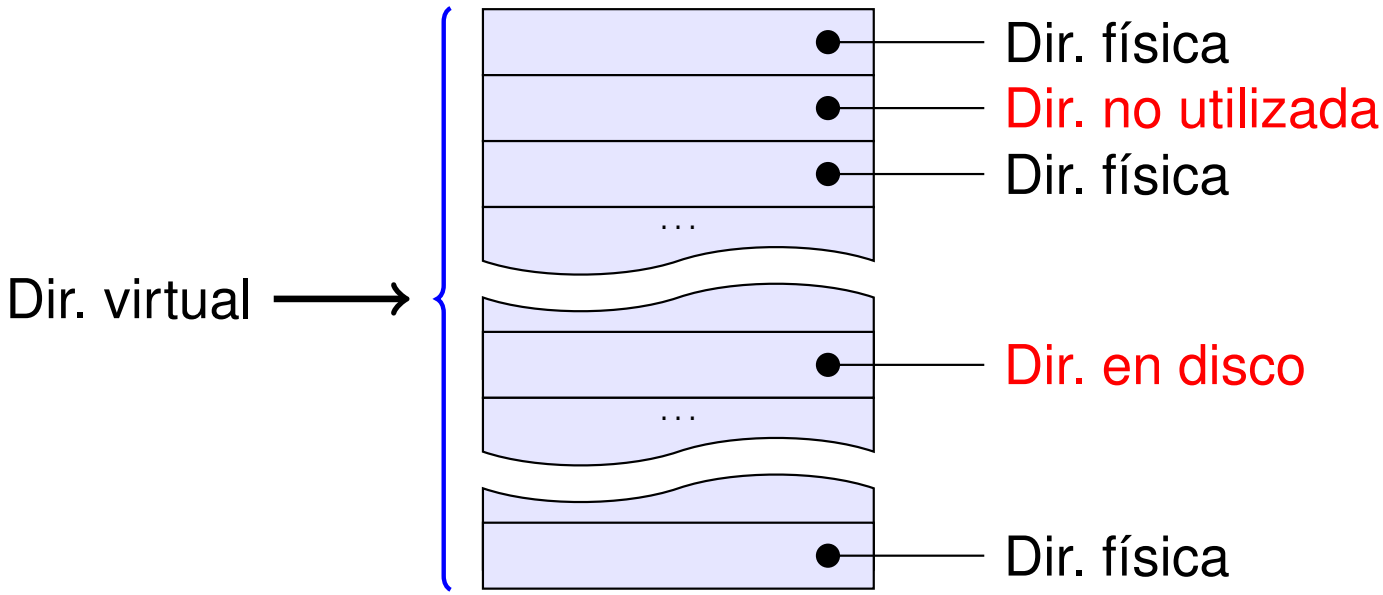

E.D. virtuales de T_1
E.D. físicas
E.D. virtuales de T_2
00...0h
var
FF...Fh
00...0h
var
var
FF...Fh
00...0h
var
FF...Fh
Tarea T_1
Tarea T_2
main ()
{
int var = 0;
var = var + 1;
}

Traducción simplificada

Tabla de traducción

Una por tarea

- dirección virtual $\Rightarrow$ dirección física / no utilizada / en disco



Funcionamiento

Parecido a la caché

1. La unidad de ejecución emite una dirección virtual
2. La MMU traduce la dirección
   - ✔ acierto $\Rightarrow$ se accede al dato en M.P.
   - ✘ fallo $\Rightarrow$ excepción
     - S.O. determina que el dato está en el disco

       1.- se copia el dato (bloque) desde el disco a M.P.
       2.- se actualiza la tabla de traducción
       3.- se repite el acceso
     - S.O. determina que el dato no está en el disco

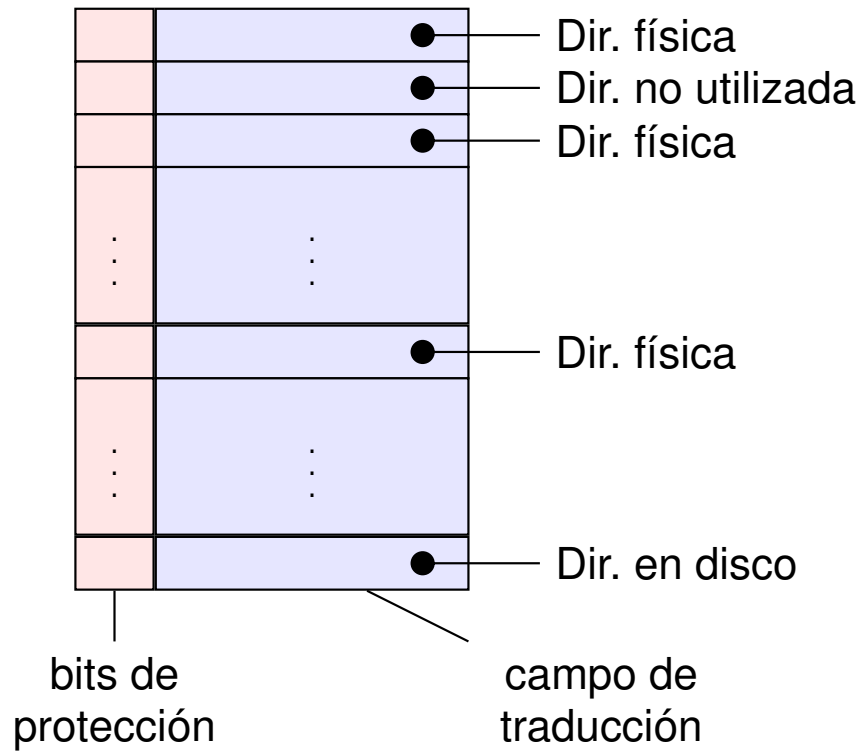
O2: Protección de memoria

Dos vertientes

- proteger al S.O. frente a las tareas
- proteger a unas tareas de otras

Solución

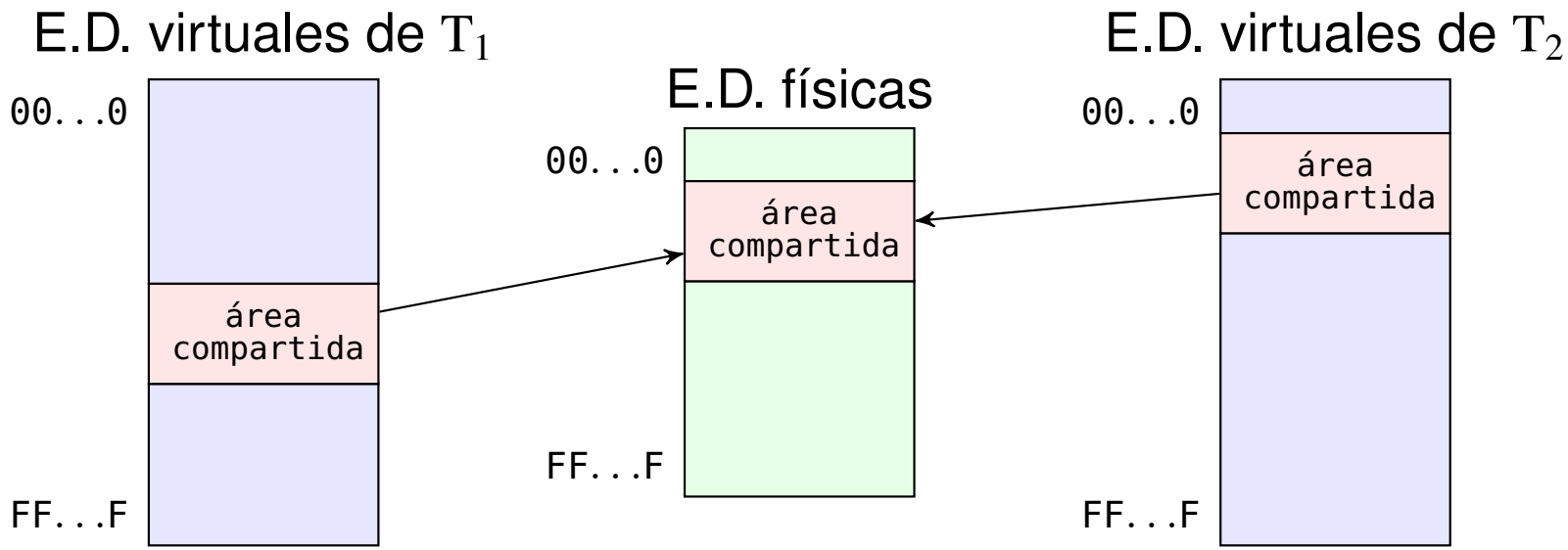
- una tabla de traducción por tarea
- bits de protección



O3: Compartición de memoria

Entre dos o más tareas

- comunicar tareas
- compartir código



O4: Herramientas de desarrollo y carga

Memoria virtual facilita la tarea de

- compiladores y enlazadores
- carga de programas

Espacios de direcciones virtuales independientes

- ubicar estructuras de datos y código
- todas las direcciones disponibles
- no preocuparse de otras tareas

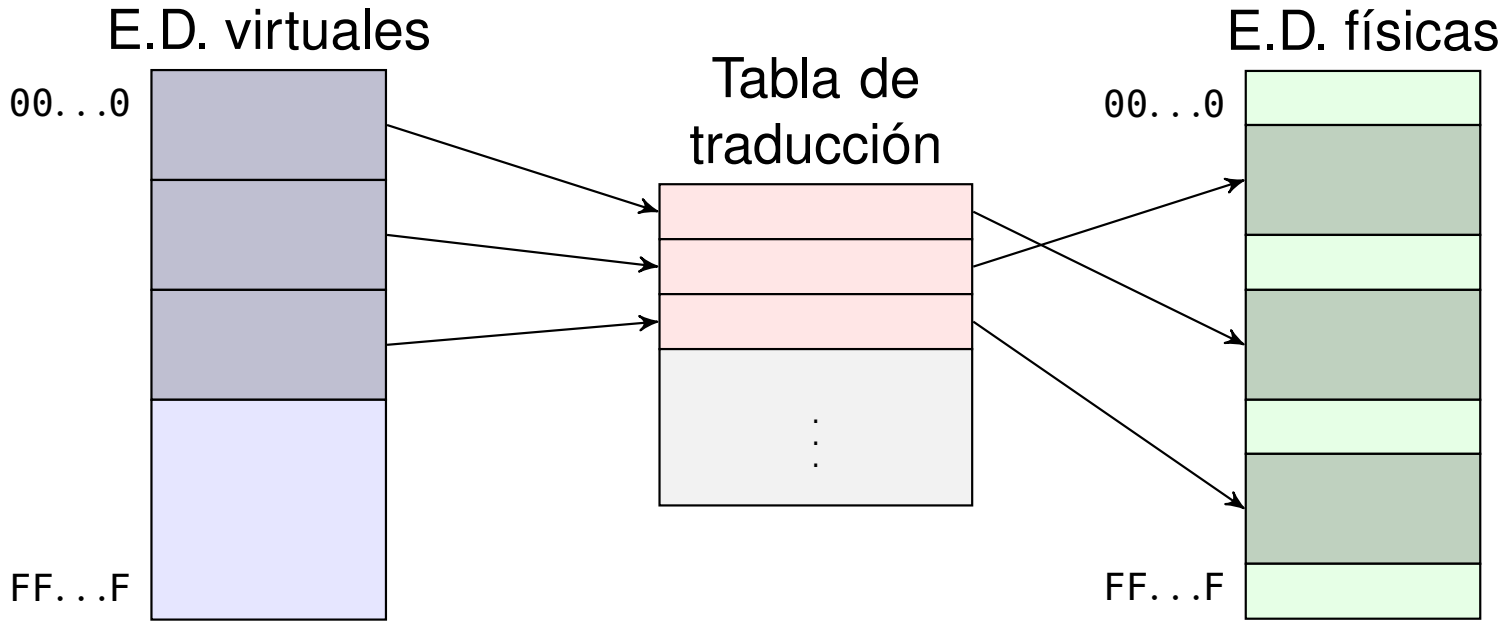
Traducción dir. virtual $\Rightarrow$ dir. física

- ubicar tareas en cualquier sitio
- registro de huecos libres

La tabla de traducción

Traducción dir. virtual $\Rightarrow$ dir. física

- no dirección a dirección de una en una (coste)
- en bloques

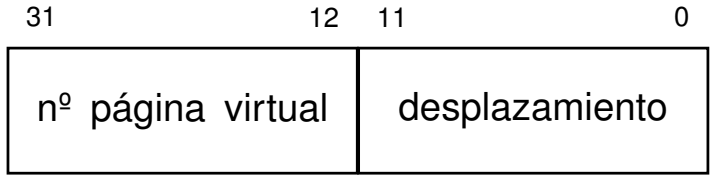


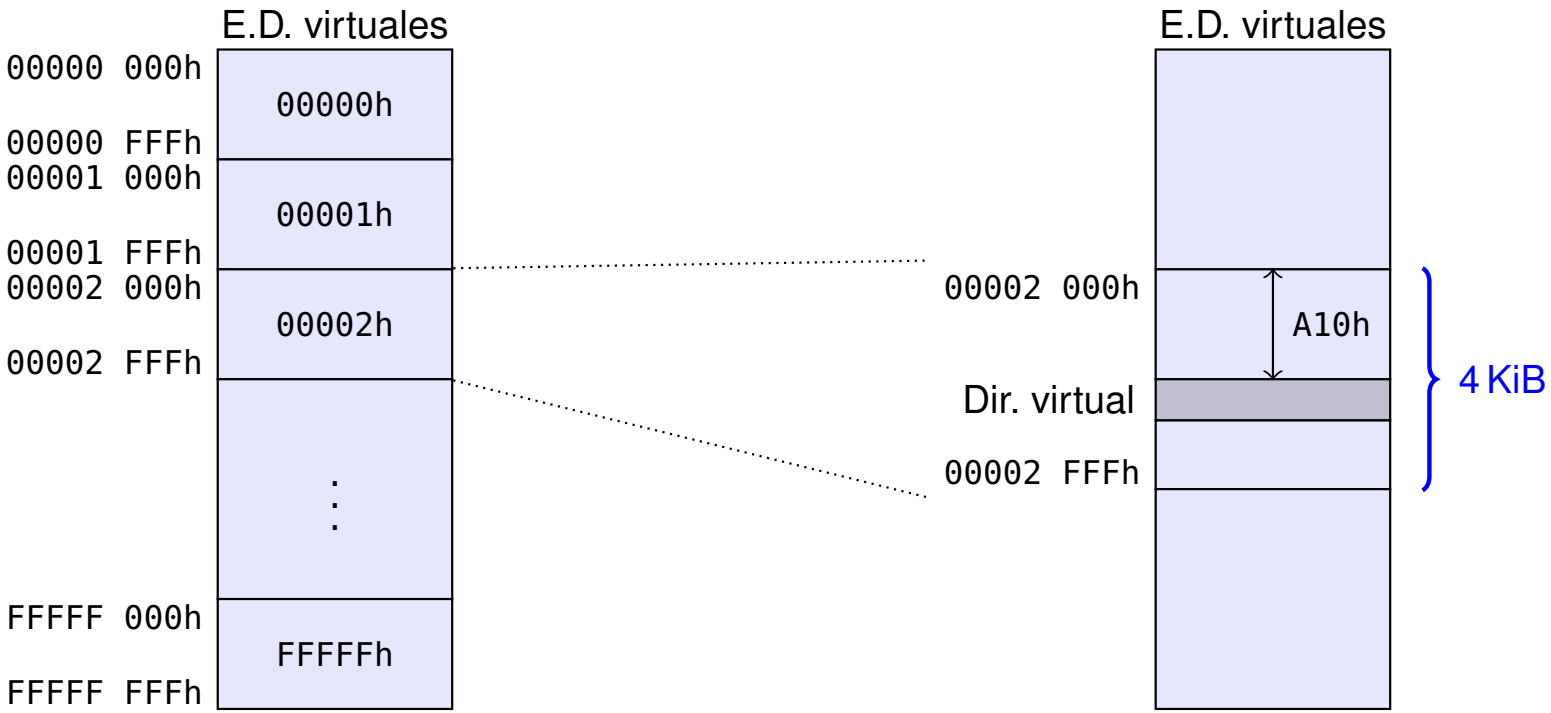
Paginación

Dirección virtual

- número de página virtual
- desplazamiento

`00002A10`



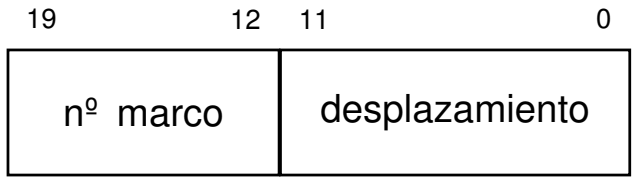


Paginación

Dirección física

- número de marco
- desplazamiento

`5FA10`



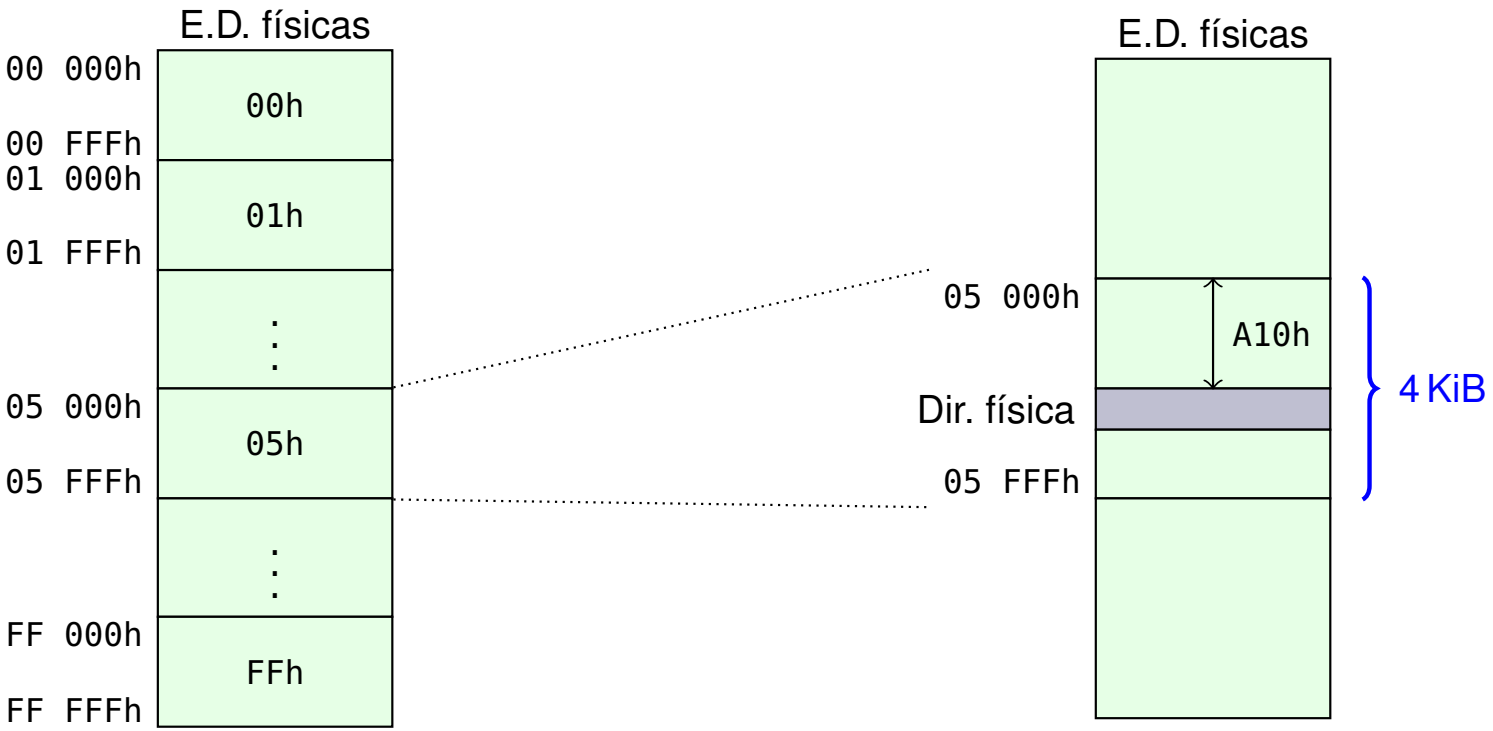


Tabla de páginas

Traducción

Página virtual $\Rightarrow$ página física (marco)

- el desplazamiento se mantiene

Características

- una por tarea
- gestionada por el sistema operativo
- a partir del registro de tabla de páginas
- tantas entradas como páginas virtuales

Tabla de páginas

Campos

- Bit de presencia
  - ✔ página en memoria (accesible)
  - ✘ página en disco o no válida
- Marco de página/Ubicación en disco: depende del bit de presencia
  - ✔ marco de página
  - ✘ ubicación en disco o página no válida
- Bits de protección
  - quién y para qué se accede a la página
- Bits de estado: facilitar la gestión $\approx$ caché
  - *dirty* $\Rightarrow$ indica si la página está modificada
  - acceso $\Rightarrow$ indica si la página ha sido accedida recientemente

Ejemplo de tabla de páginas

Dirección virtual de 32 bits

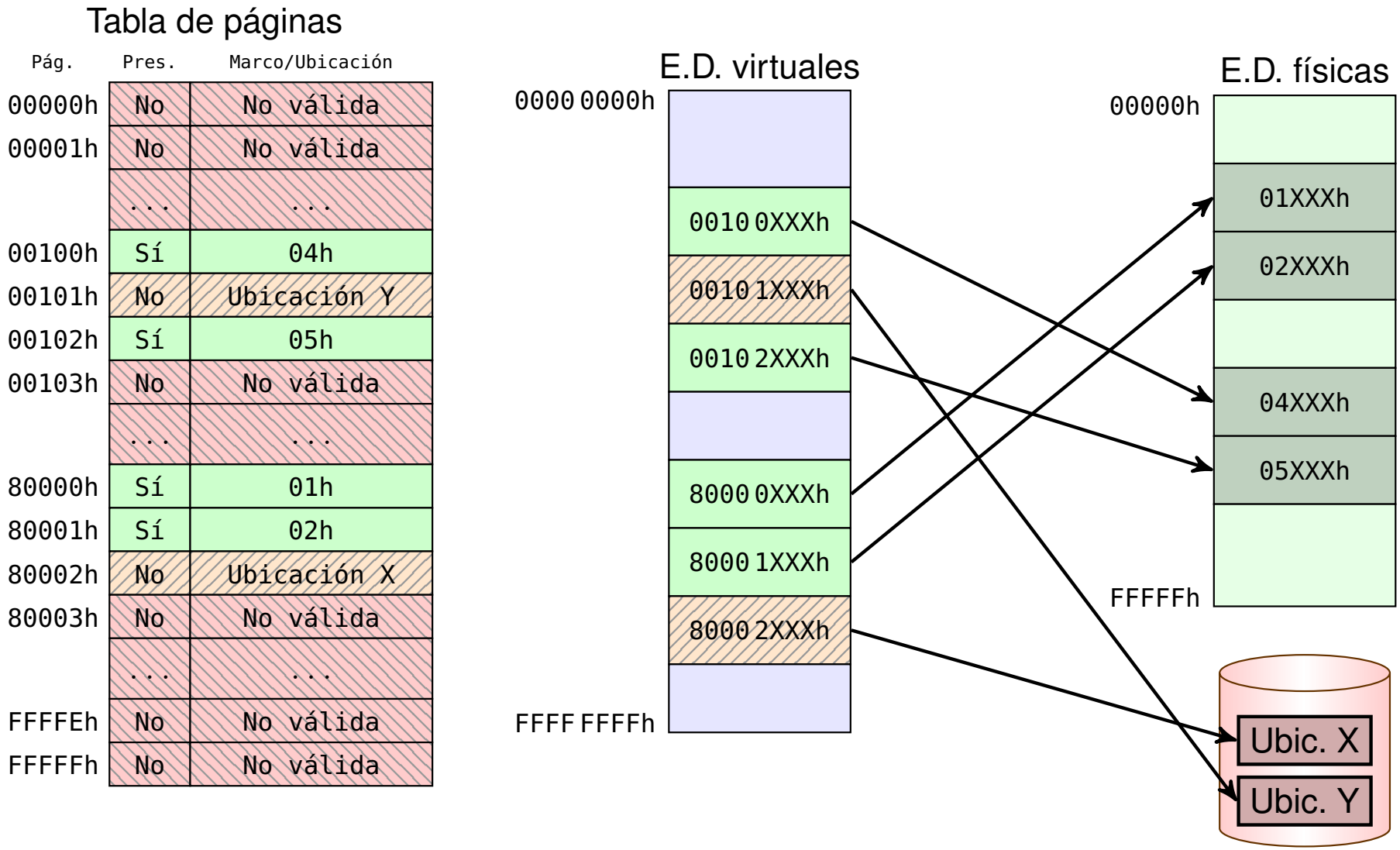
- página: 20 bits $\Rightarrow 2^{20}$ páginas
- desplazamiento: 12 bits $\Rightarrow$ páginas de 4 Kipalabras

Dirección física de 20 bits

- marco: 8 bits $\Rightarrow 2^{8}$ marcos
- desplazamiento: 12 bits $\Rightarrow$ páginas de 4 Kipalabras

Palabra de 1 byte $\Rightarrow$ página de 4 KiB

Ejemplo de tabla de páginas



Ejemplo de traducción

Dir. virtual

80001 5AB

traducción $\rightarrow$

Dir. física

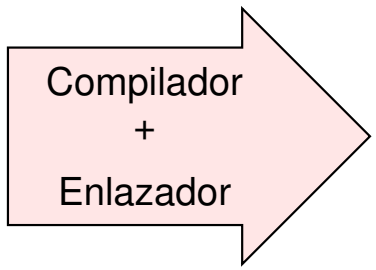
02 5AB

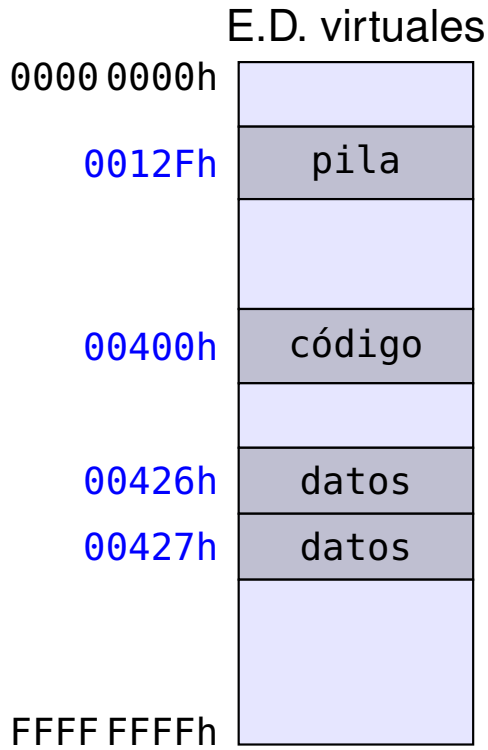
Pág.	Pres.	Marco/Ubicación
00000	No	No válida
00001	No	No válida
	...	...
00100	Sí	04
00101	No	Ubicación Y
00102	Sí	05
00103	No	No válida
	...	...
80000	Sí	01
80001	Sí	02
80002	No	Ubicación X
80003	No	No válida
	...	...
FFFFE	No	No válida
FFFFF	No	No válida

Tareas en memoria

Máquina de 32 bits con páginas de 4 KiB

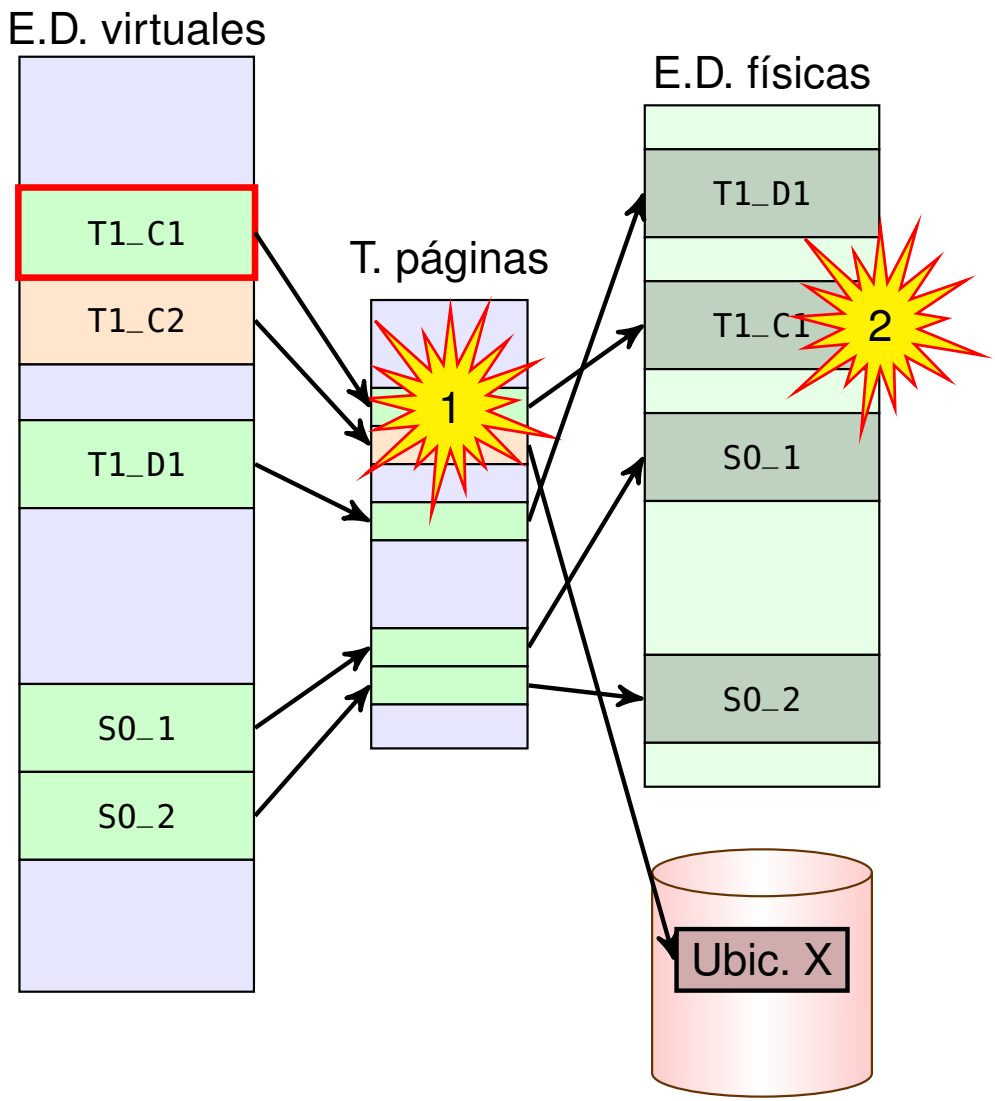
```
float var[1500];

main ()
{
  int i;

  for (i = 1; i < 1500; i++)
    var[i] = 0;
}
```





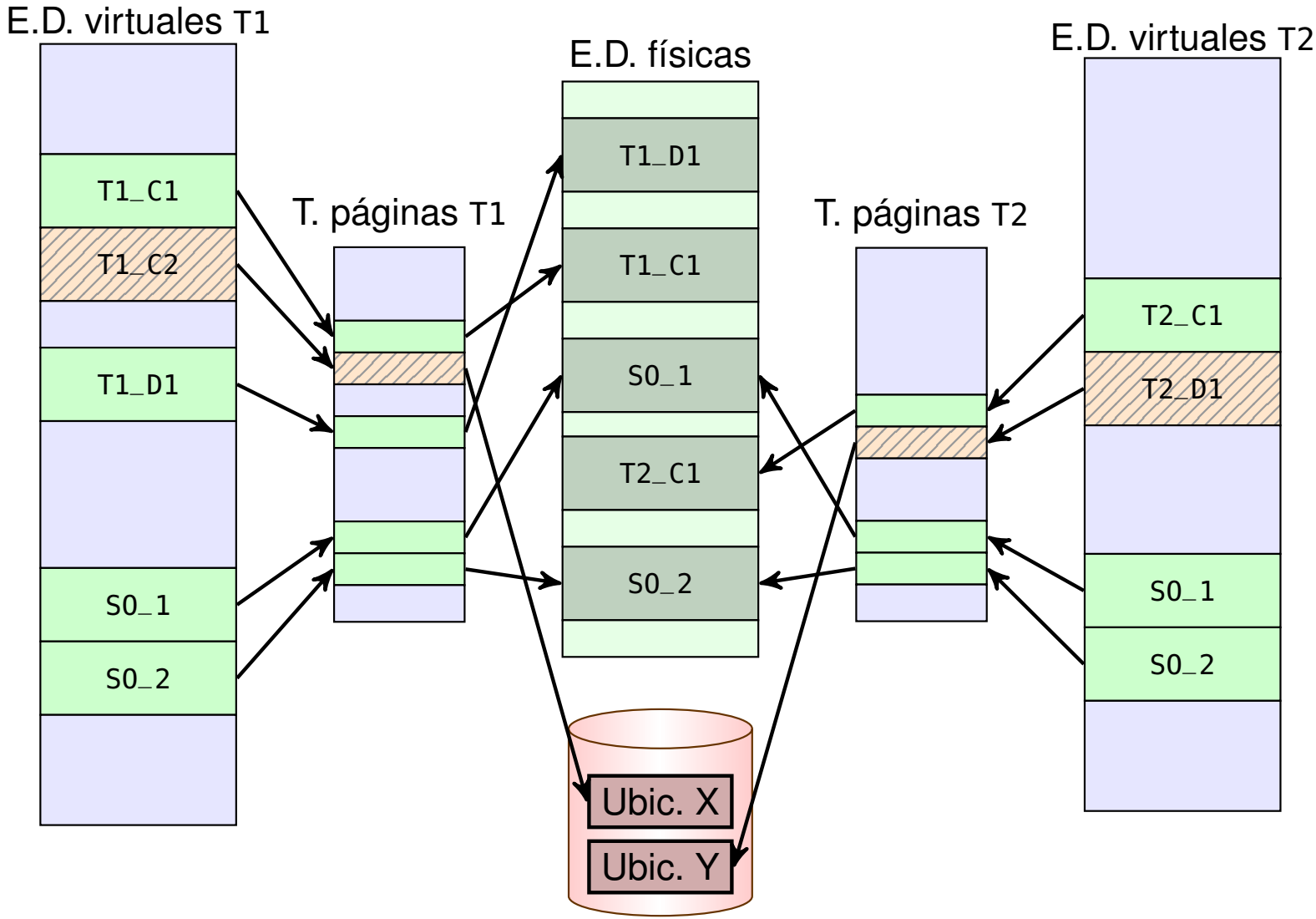
Añadiendo el S.O.

- Forma parte del E.D. virtuales de todas las tareas
- Recursos en la tabla de páginas
- Tabla de páginas traduce direcciones de la tarea y del S.O.



Añadiendo el S.O. (2 tareas)

Tareas comparten la memoria del sistema operativo



Protección de memoria

Independencia de espacios de direcciones entre tareas

- cada tarea su propia tabla de páginas
- se evitan solapamientos en memoria física

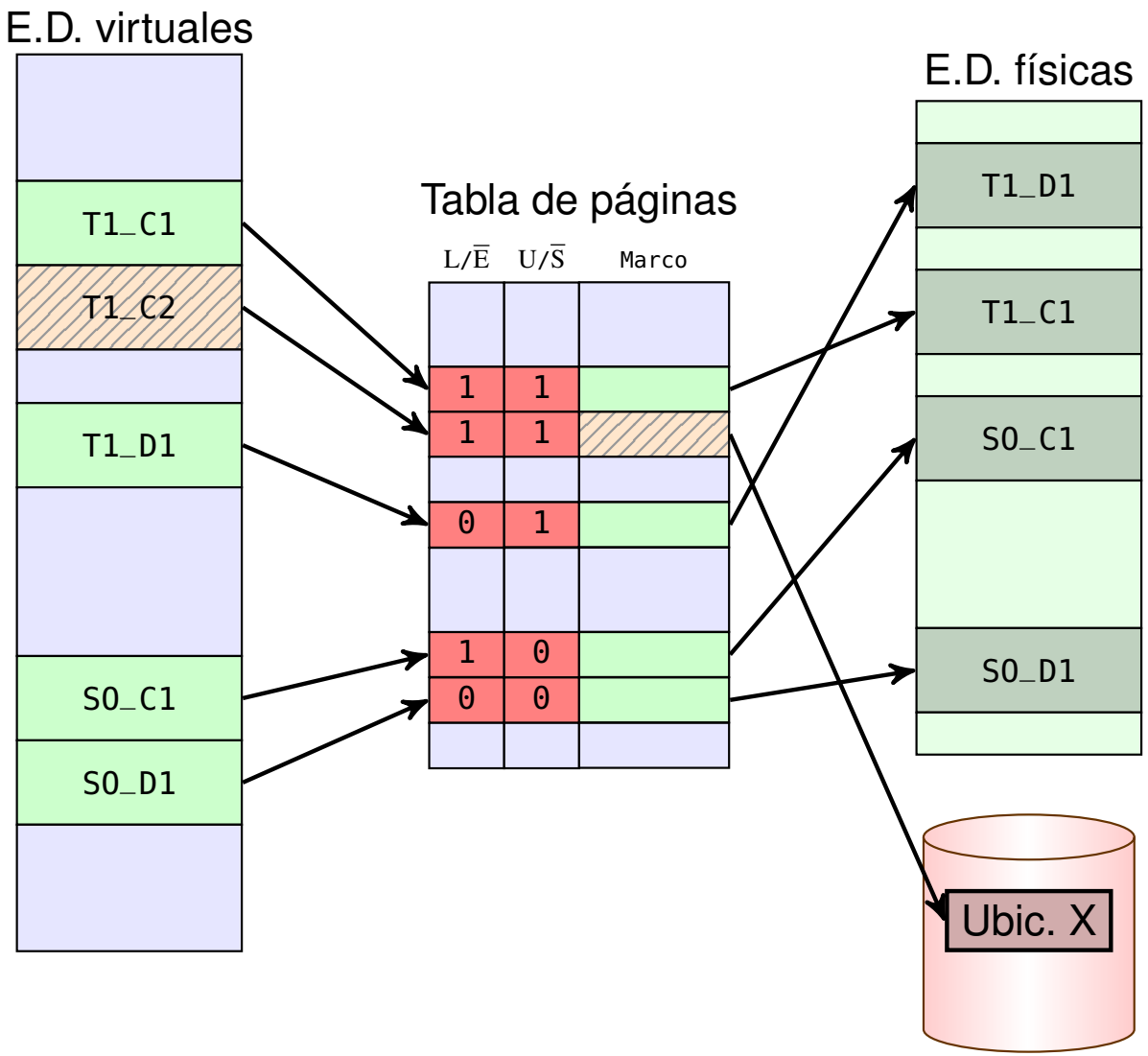
Tipo de acceso

- bits adicionales en cada entrada de la T.P.
- para qué puede ser accedida una página

Nivel de privilegio

- asociado un nivel de privilegio a cada página

Bits de protección



- Bit L: solo lectura
- Bit $\overline{\mathrm{E}}$: lectura y escritura
- Bit U: nivel de usuario
- Bit $\overline{\mathrm{S}}$: nivel de supervisor

Ejemplo de protección

Windows 32 bits

Memoria virtual dividida en dos rangos

- `0000 0000h - 7FFF FFFFh` (2 GiB): accesible a la tarea
- `8000 0000h - FFFF FFFFh` (2 GiB): solo al sistema operativo

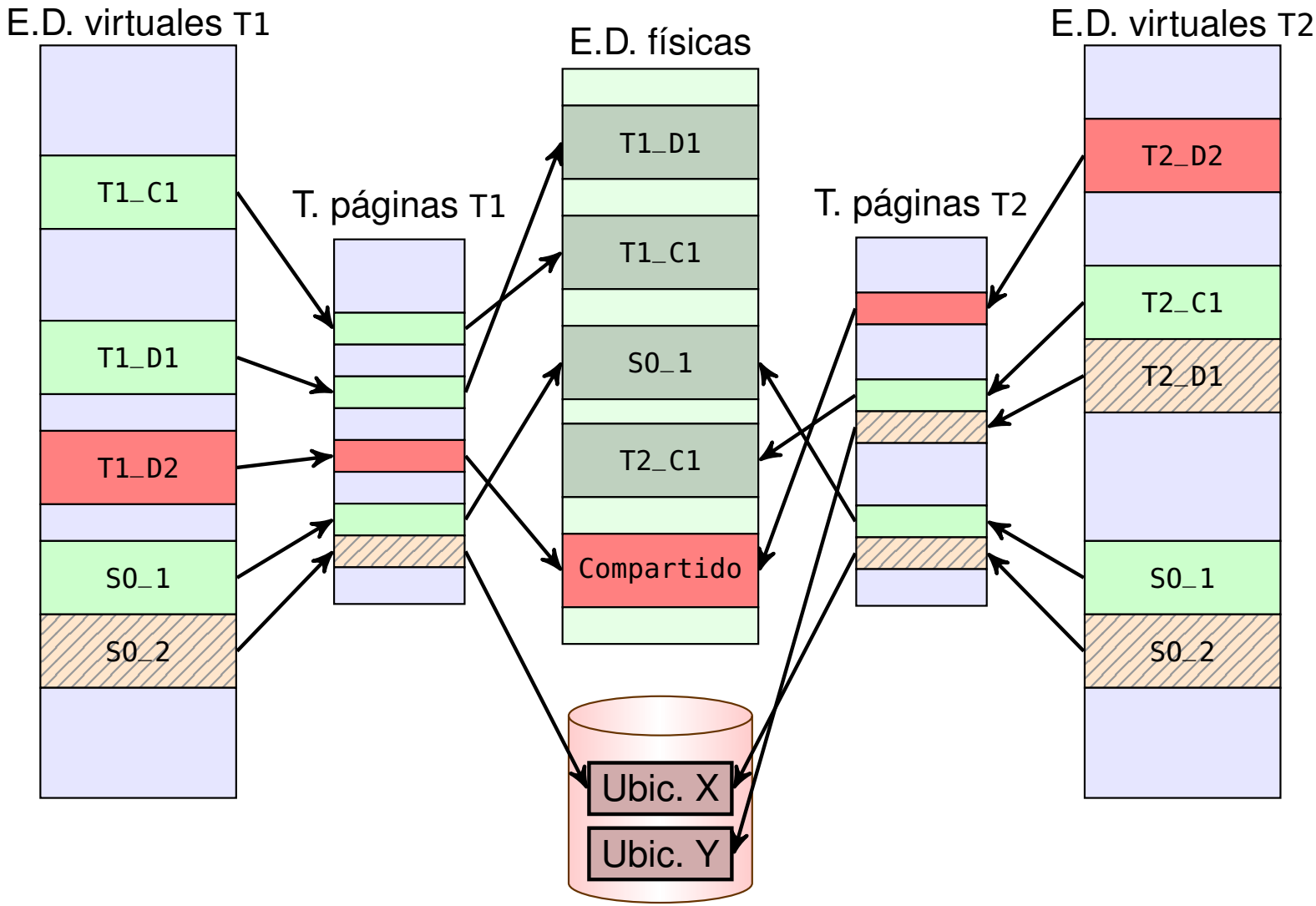
Linux 32 bits

Memoria virtual dividida en dos rangos

- `0000 0000h - BFFF FFFFh` (3 GiB): accesible a la tarea
- `C000 0000h - FFFF FFFFh` (1 GiB): solo al sistema operativo

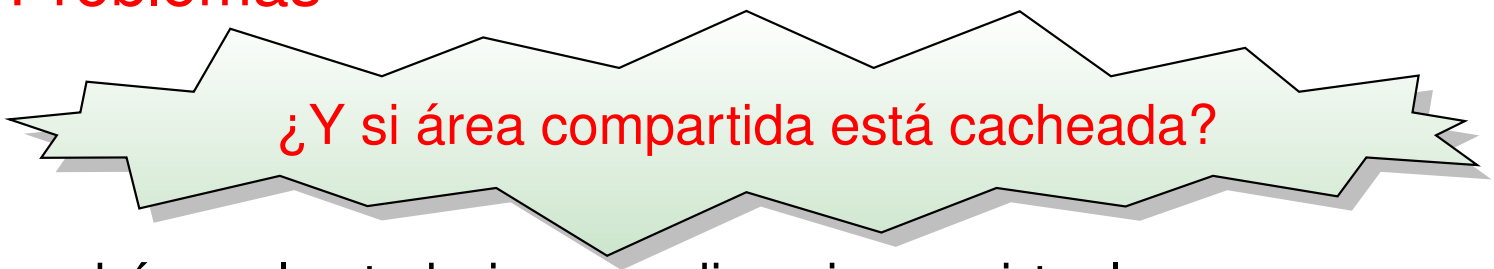
Compartición de memoria

Se solicita al sistema operativo

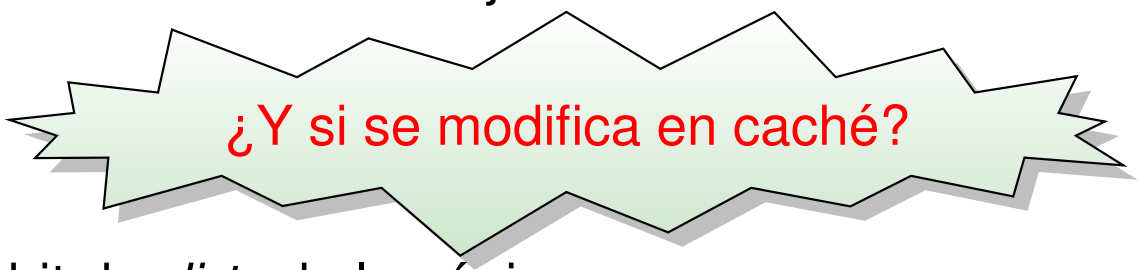


Compartición de memoria

Problemas



cachés reales trabajan con direcciones virtuales



bit de *dirty* de la página

Coherencia entre tablas de páginas

Fallo de página

Se produce

Dirección virtual emitida no está en memoria física

- bit de presencia desactivado

¿Qué sucede?

Excepción por «fallo de página»

- se ejecuta manejador del sistema operativo

Sistema operativo determina tipo de fallo

- dirección en página sin almacenamiento
- dirección en página en disco

Fallo de página

Fallo no recuperable

Página sin almacenamiento asignado

Acceso de lectura

00103 42Ch

Pág.	Pres.	Marco/Ubicación
00000	No	No válida
00001	No	No válida
	...	...
00100	Sí	04
00101	No	Ubicación Y
00102	Sí	05
00103	No	No válida
	...	...
80000	Sí	01
80001	Sí	02
80002	No	Ubicación X
80003	No	No válida
	...	...
FFFFE	No	No válida
FFFFF	No	No válida

Fallo de página

Fallo recuperable

Página en disco

Acceso de lectura

`00101 190h`

Sistema operativo debe decidir:

1. Dónde cargará la página
   - reemplazo
2. Actualizar página saliente
   - bit *dirty*
3. Carga la página y actualizar T.P.
4. Volver a ejecutar instrucción

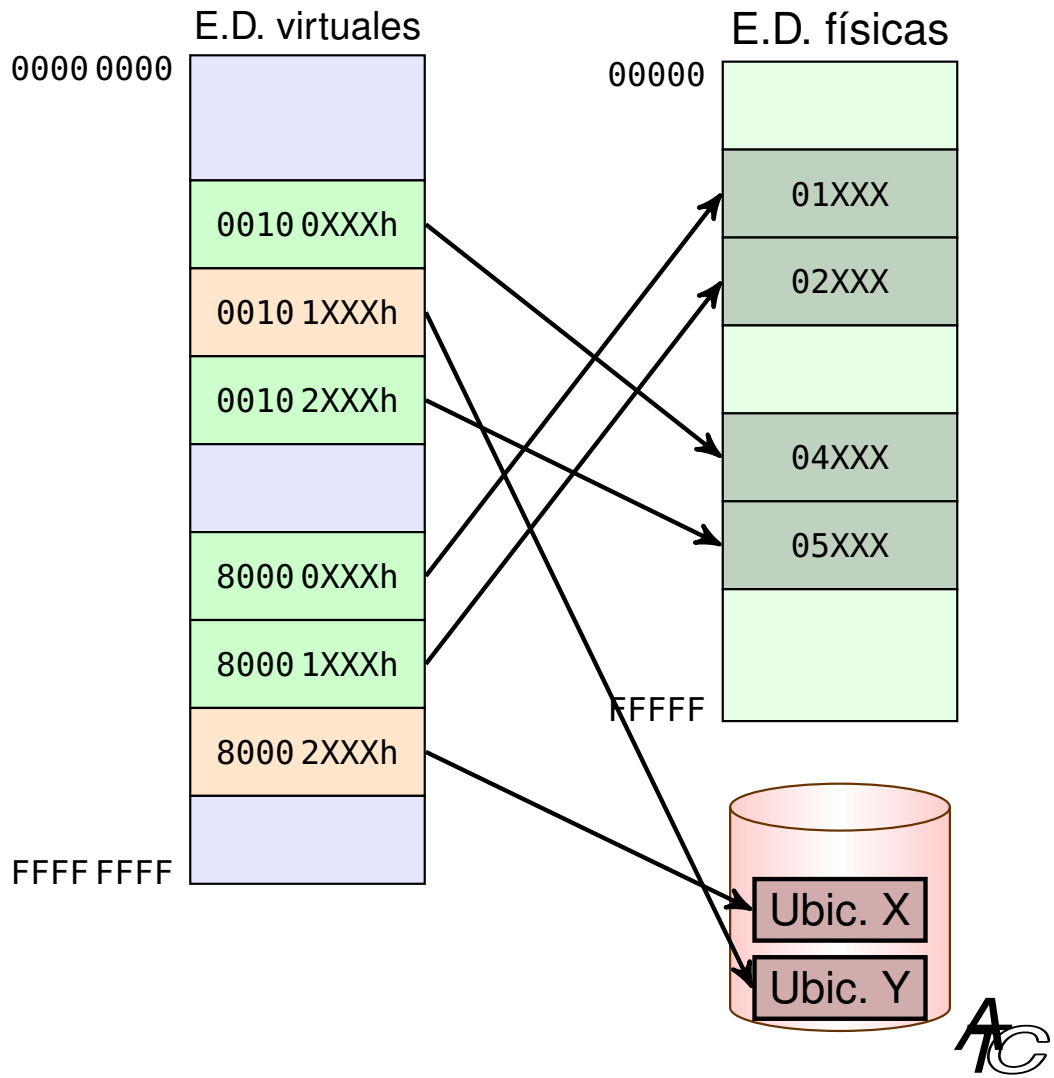
Pág.	Pres.	Marco/Ubicación
00000	No	No válida
00001	No	No válida
	...	...
00100	Sí	04
00101	No	Ubicación Y
00102	Sí	05
00103	No	No válida
	...	...
80000	Sí	01
80001	Sí	02
80002	No	Ubicación X
80003	No	No válida
	...	...
FFFFE	No	No válida
FFFFF	No	No válida

Fallo de página

Acceso de lectura

00101 190h ⇒ reemplazar página 02. *Working set*

Pág.	Pres.	*Dirty*	Marco/Ubicación
00000	No	-	No válida
00001	No	-	No válida
	...	...	...
00100	Sí	No	04
00101	No	-	Ubicación Y
00102	Sí	Sí	05
00103	No	-	No válida
	...	...	...
80000	Sí	Sí	01
80001	Sí	Sí	02
80002	No	-	Ubicación X
80003	No	-	No válida
	...	...	...
FFFFE	No	-	No válida
FFFFF	No	-	No válida

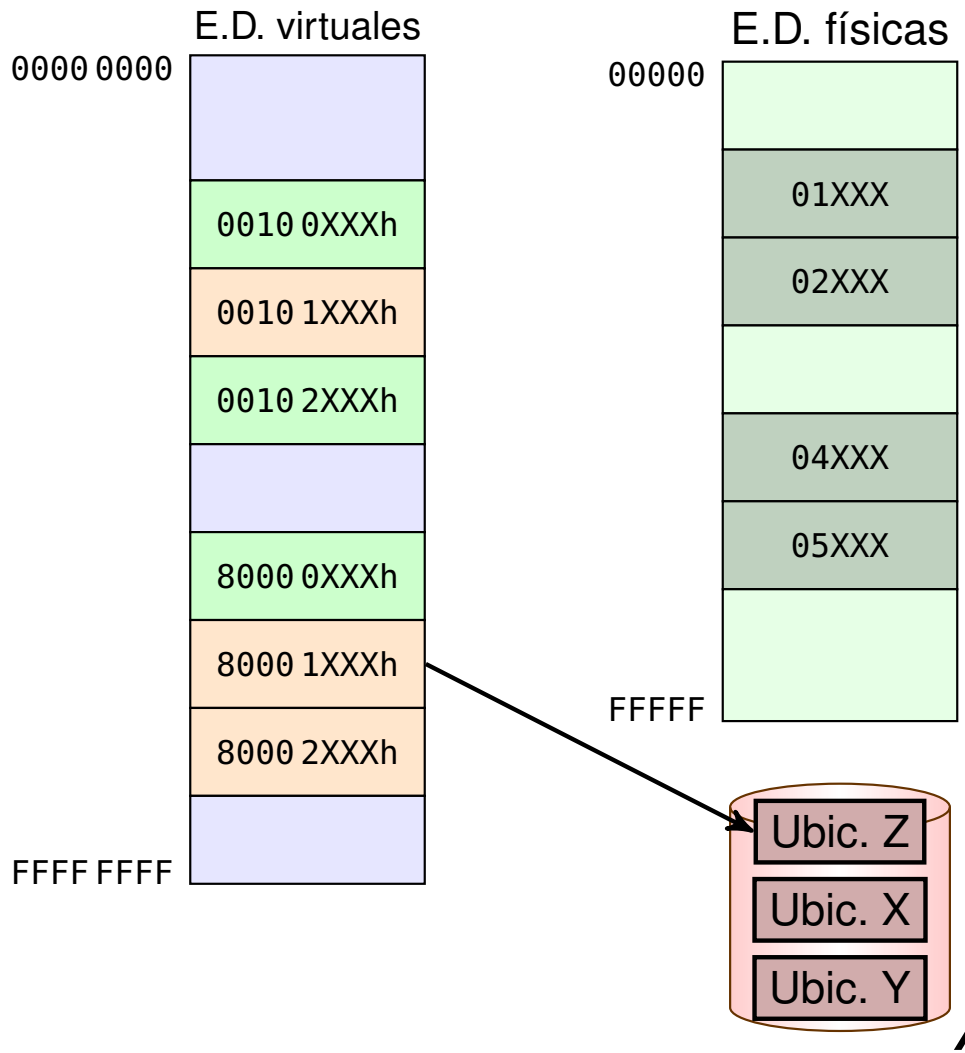


Fallo de página

Acceso de lectura

00101 190h ⇒ reemplazar página 02. *Working set*

Pág.	Pres.	*Dirty*	Marco/Ubicación
00000	No	-	No válida
00001	No	-	No válida
	...	...	...
00100	Sí	No	04
00101	No	-	Ubicación Y
00102	Sí	Sí	05
00103	No	-	No válida
	...	...	...
80000	Sí	Sí	01
80001	No	-	Ubicación Z
80002	No	-	Ubicación X
80003	No	-	No válida
	...	...	...
FFFFE	No	-	No válida
FFFFF	No	-	No válida

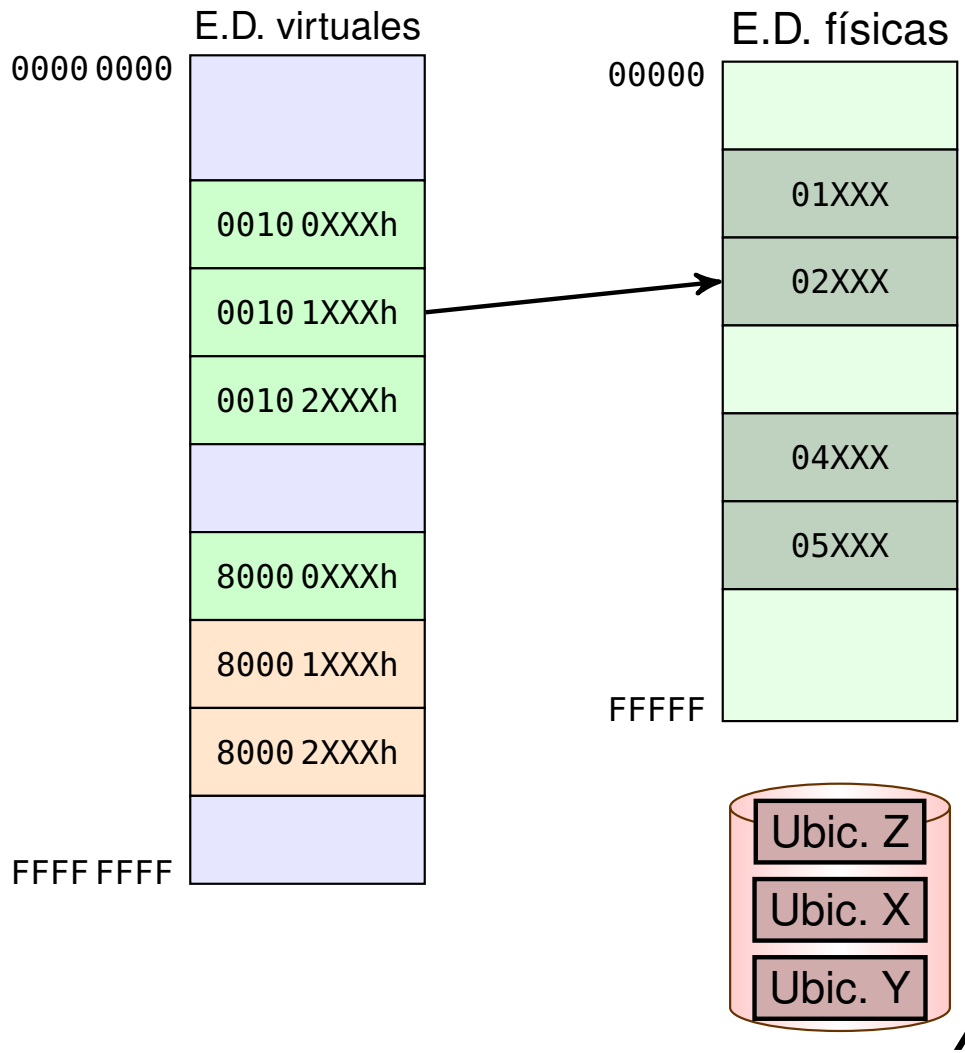


Fallo de página

Acceso de lectura

00101 190h ⇒ reemplazar página 02. *Working set*

Pág.	Pres.	*Dirty*	Marco/Ubicación
00000	No	-	No válida
00001	No	-	No válida
	...	...	...
00100	Sí	No	04
00101	Sí	No	02
00102	Sí	Sí	05
00103	No	-	No válida
	...	...	...
80000	Sí	Sí	01
80001	No	-	Ubicación Z
80002	No	-	Ubicación X
80003	No	-	No válida
	...	...	...
FFFFE	No	-	No válida
FFFFF	No	-	No válida



Fallo de página

Fallo de TLB

- Excepción $\Rightarrow$ manejador del sistema operativo
- Se busca en la tabla de páginas la entrada
  - traducción a dirección física
- Se copia la entrada al TLB
- Todas ocupadas? $\Rightarrow$ política de reemplazo

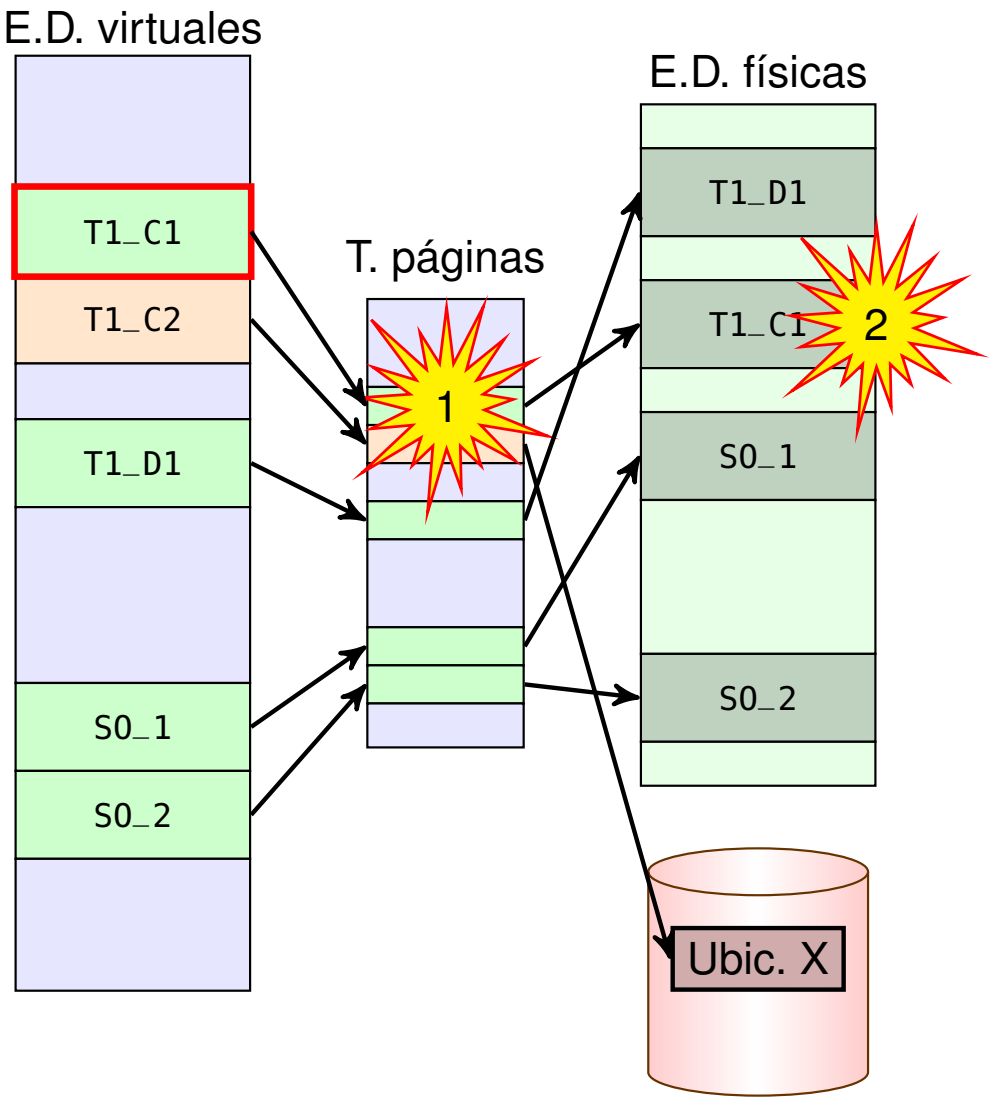
Problema de rendimiento

Doble acceso a memoria

1. entrada en la tabla de páginas
   - traducir dirección
2. dato

Solución

El TLB (*Translation lookaside buffer*)



El TLB

- Principio de localidad de referencias
  - acceso a pocas páginas virtuales
  - se necesita acceder a pocas entradas en la tabla de páginas
- Se puede utilizar una caché
- Entradas más usadas se almacenan en TLB
  - *Translation Lookaside Buffer*
  - caché específica
  - dentro de la CPU

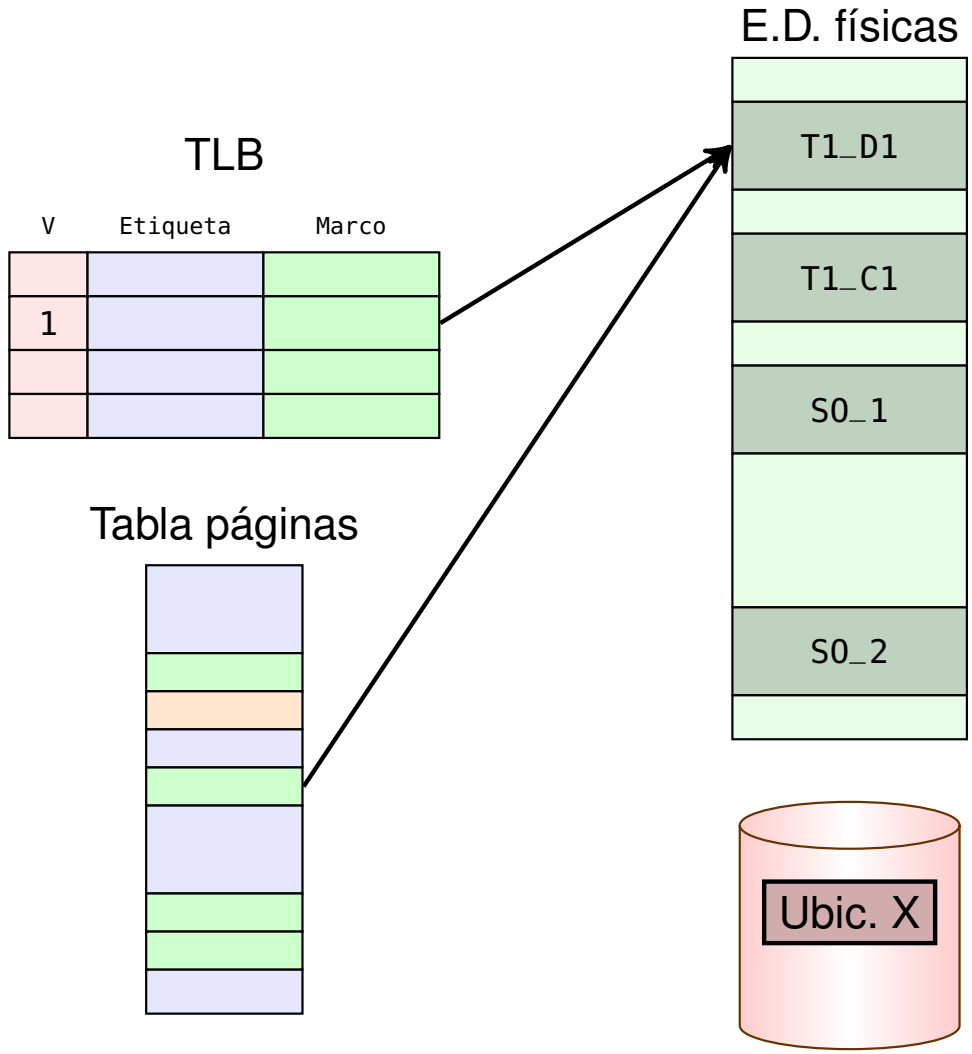
El TLB

Antes

1. tabla de páginas
2. dato

Con TLB

1. TLB (bit validez)
2. dato



El TLB

Características

- Memoria caché
- Almacena entradas de la tabla de páginas más accedidas
- Bits de validez, de acceso y etiqueta
- Caché totalmente asociativa
- Políticas de reemplazo y escritura
- Gestionado por hardware (IA32) / software (MIPS)

El TLB

Funcionamiento en una tarea

Buscar entradas de las tablas de páginas en el TLB

- ✔ acierto $\Rightarrow$ se obtiene rápidamente el número de marco
- ✘ fallo $\Rightarrow$ actualizar la entrada del TLB

Problema durante un cambio de contexto

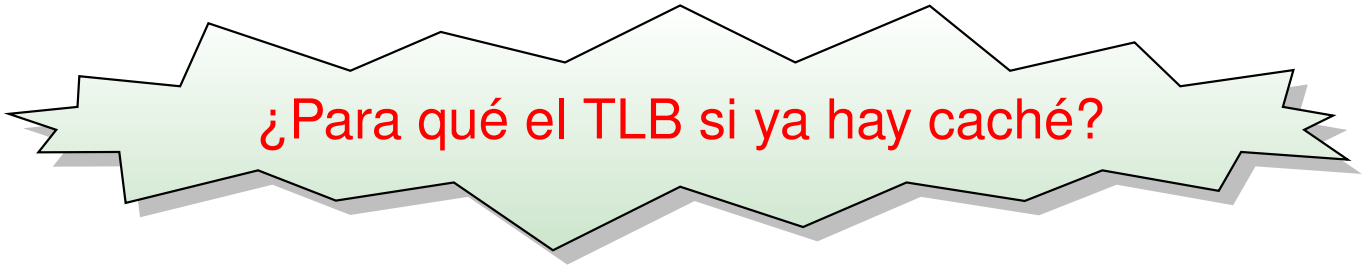
- cambio de tarea $\Rightarrow$ cambio de tabla de páginas
- invalidar muchas entradas de TLB $\Rightarrow$ fallos de TLB
  - penalización del rendimiento

Solución

Bits extra para cada entrada del TLB

- identificar la tarea
- evitar invalidación del TLB completa
- entradas en la tabla de páginas del S.O. marcadas como globales

El TLB



- caché podría almacenar entradas en la tabla de páginas
- cachés unificadas son más eficientes

Ventaja

Accesos «simultáneos» a caché y TLB

- traducir y acceder a dirección

Combinando TLB y caché

Traducir dirección virtual

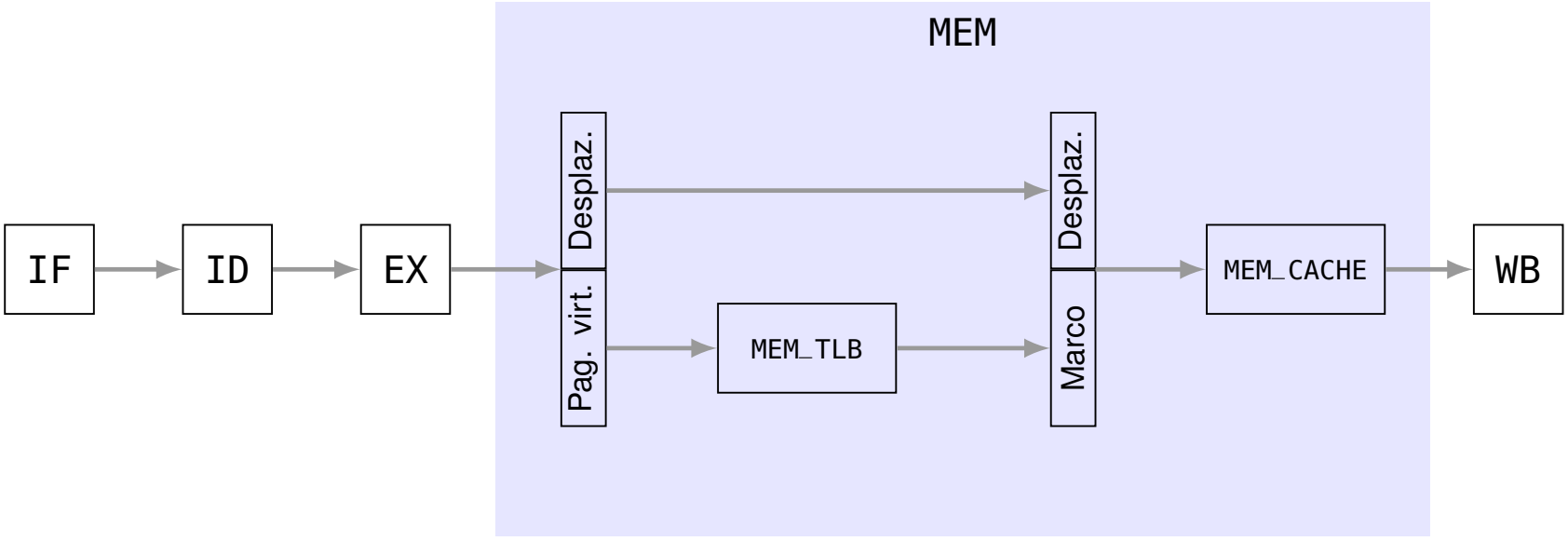
1. Se accede al TLB con el número de página virtual
2. Acierto $\Rightarrow$ se obtiene número de marco
   - se realiza la traducción
3. Dirección física se interpreta según la caché
4. Buscamos dato en la caché

Problema

No es posible el acceso simultáneo a TLB y caché

- acceso segmentado
- cachés virtuales

Acceso segmentado TLB - Caché



Problema

- Necesarias cargas/almacenamientos consecutivos
- Poco efectivo

Cachés virtuales

Hasta ahora

Physically indexed/*Physically tagged* (P/P)

- Conjunto (*index*) $\leftarrow$ dir. física
- Etiqueta (*tag*) $\leftarrow$ dir. física

Idea (solo L1)

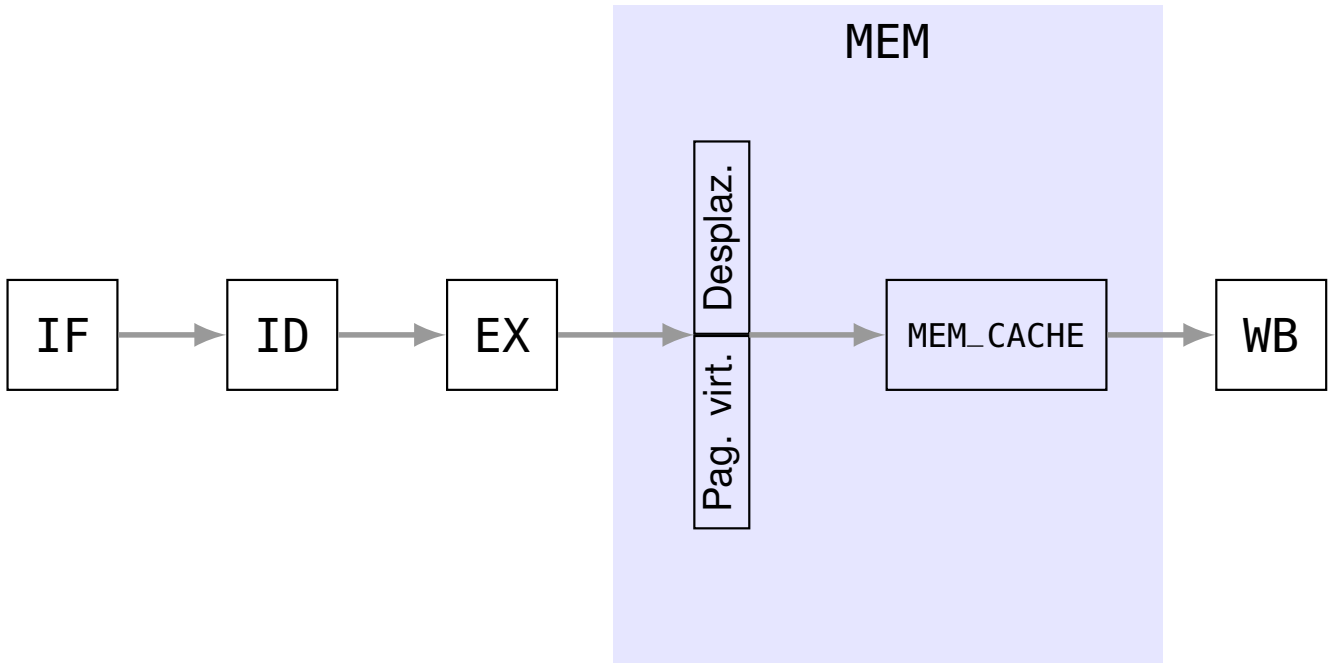
Usar direcciones virtuales

- Total/parcialmente
- Evitar esperar por TLB

Cachés virtuales (V/V)

Virtually indexed/Virtually tagged

- Conjunto (*index*) $\leftarrow$ dir. virtual
- Etiqueta (*tag*) $\leftarrow$ dir. virtual

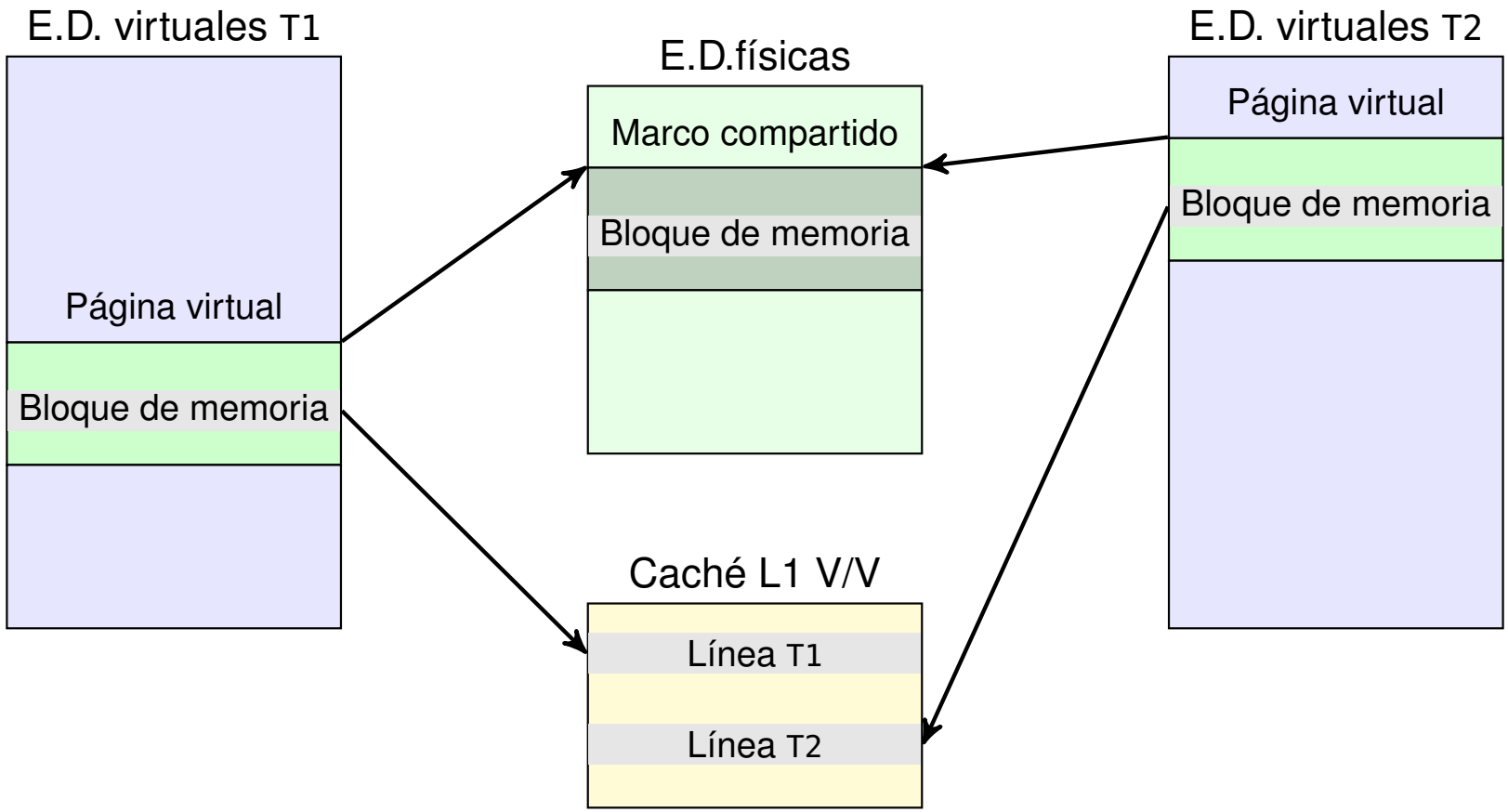


- Acierto $\Rightarrow$ 1 ciclo
- Fallo de caché $\Rightarrow$ traducir dirección

Cachés virtuales (V/V)

Problemas

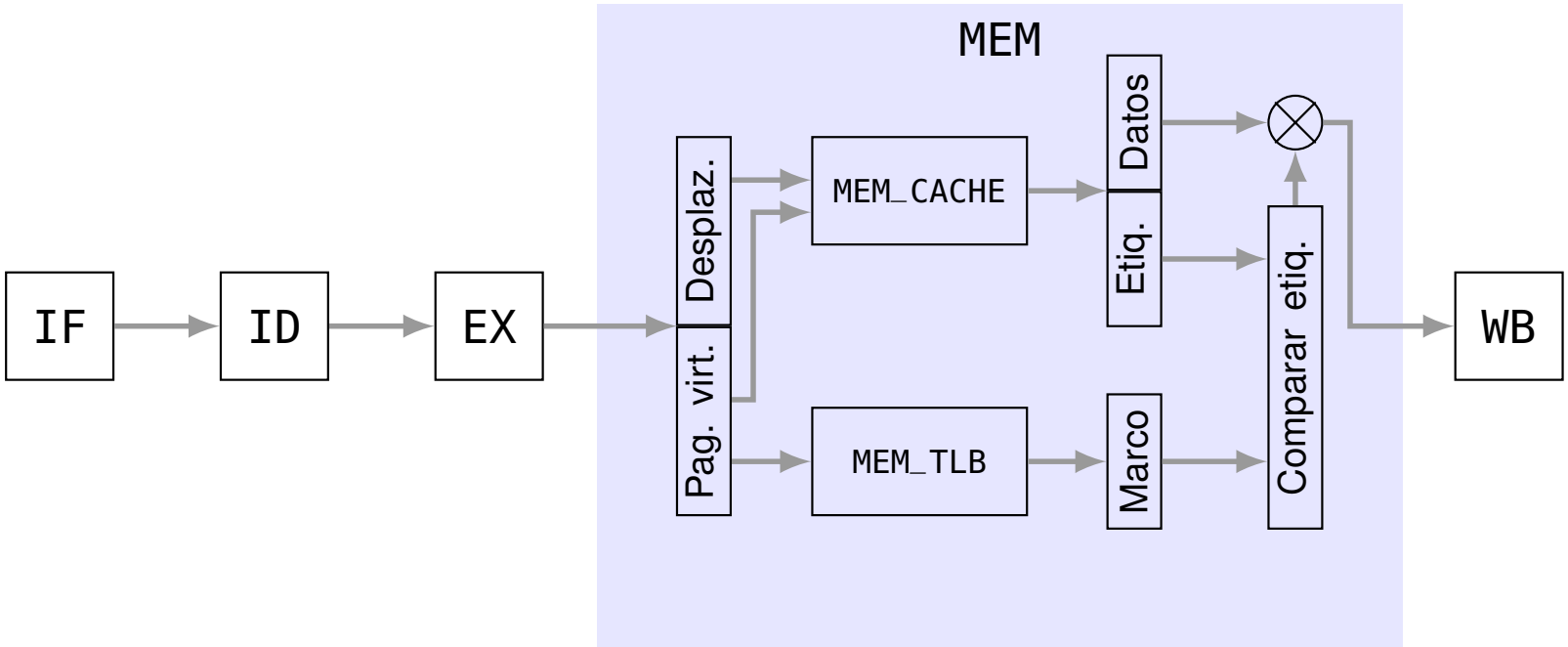
- Id. tarea / invalidar caché en cambio de contexto
- Caché implementa protección memoria
- Sinónimos



Cachés virtuales (V/P)

Virtually indexed/Physically tagged

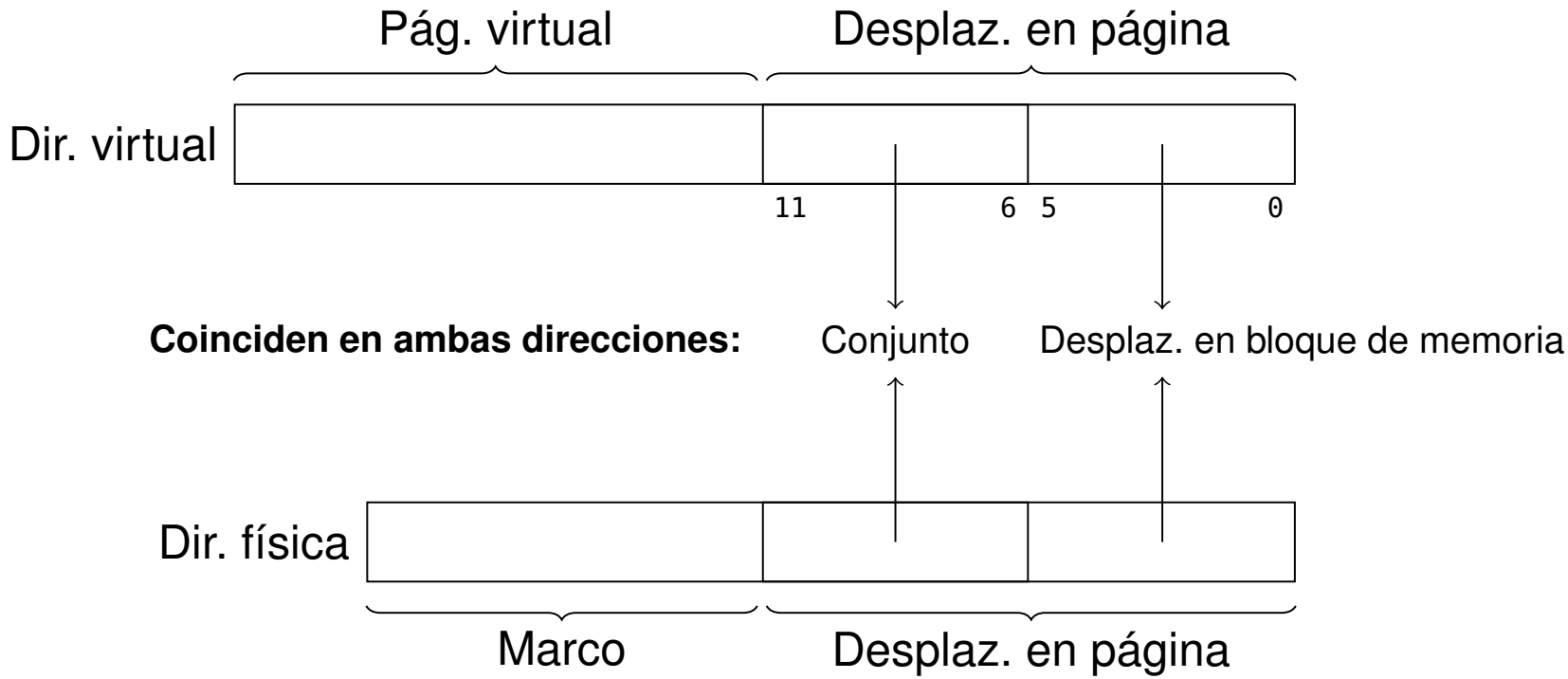
- Conjunto (*index*) $\leftarrow$ dir. virtual
- Etiqueta (*tag*) $\leftarrow$ dir. física



Cachés virtuales (V/P)

Características

- Siempre acceso a TLB $\Rightarrow$ protección
- Solución más simple a sinónimos



Limitación

Tam. caché $\leq$ Tam. Pág. virtual $\times$ Núm. vías

Virtualización

Objetivo

Compartir memoria física del E. D. entre máquinas virtuales

- Gestionada por el hipervisor
- Nuevo nivel de virtualización entre la memoria virtual y la física

Espacios de direcciones

- Direcciones virtuales del invitado (*Guest Virtual Addresses (GVAs)*)
- Direcciones físicas del invitado (*Guest Physical Addresses (GPAs)*)
- Direcciones físicas del anfitrión (*Host Physical Addresses (HPAs)*)

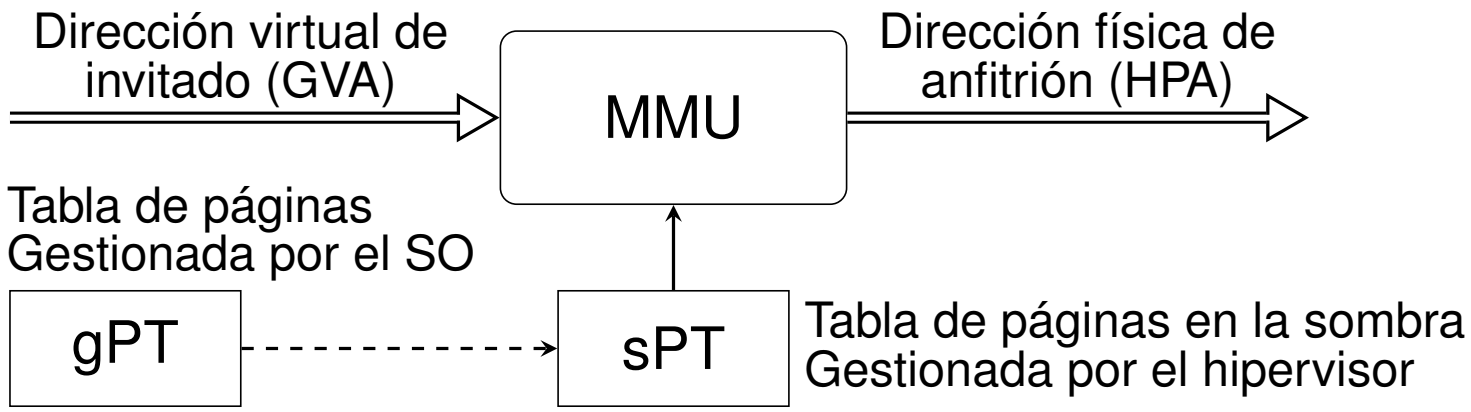
Traducciones

- GVA $\rightarrow$ GPA (MMU)
- GPA $\rightarrow$ HPA (?)

Traducción GPA $\rightarrow$ HPA

Tablas de páginas en la sombra (*Shadow page tables*)

- Una tabla de páginas en la sombra (sPT) por cada tarea
- Gestionada por el hipervisor
- No se usa la tabla de páginas del invitado (gPT)
- La MMU trabaja con la sPT
- ✘ Muchos cambios de contexto



El hipervisor intercepta operaciones sobre la gPT y las lleva a la sPT

Traducción GPA → HPA

Traducción de segundo nivel (*Second Level Address Translation (SLAT)*)

1. GVA → GPA (MMU usando gPT)
   - O.S. anfitrión gestiona gPT + TLB
2. GPA → HPA (MMU usando nPT)
   - gestionada por el hipervisor
   - El TLB almacena la traducción GVA → HPA

✘ En un fallo de TLB se leen 2 entradas de tabla de páginas (gPT + nPT)