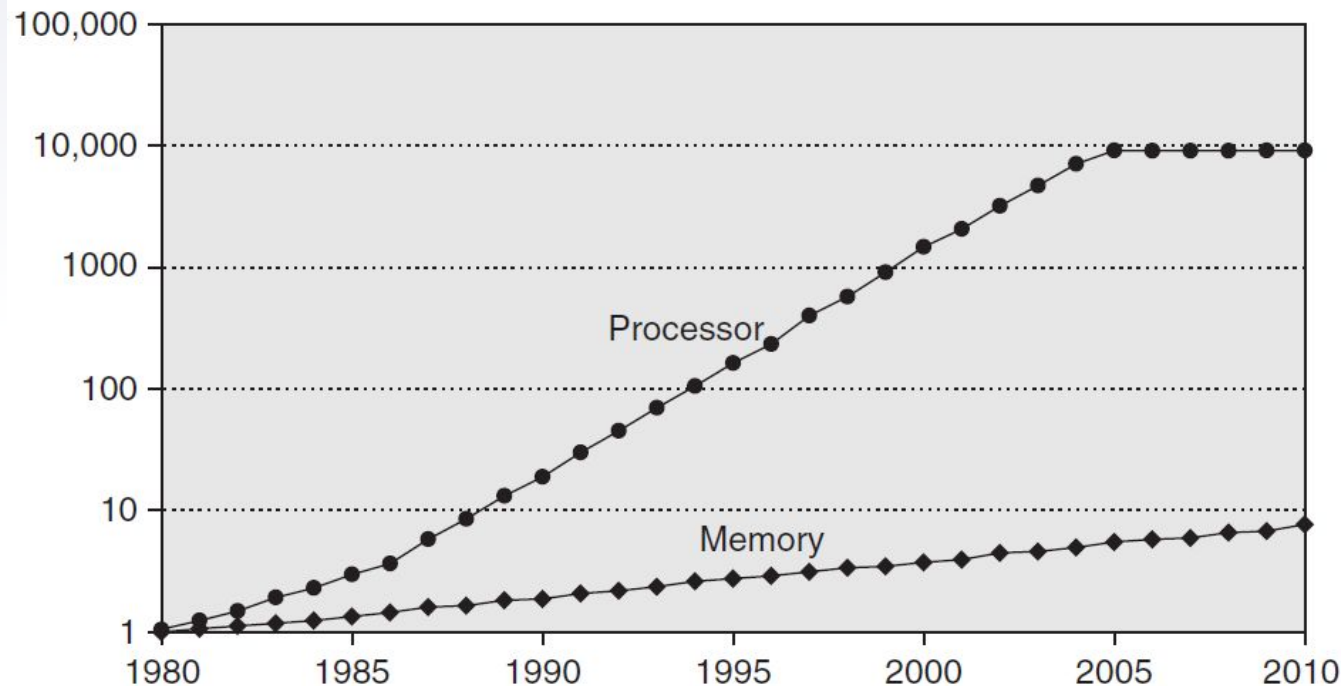


Unidad 1.1

Memoria Caché


2023 - Ing. Jaír E. Hnatiuk – Ing. Edgardo Gho - Srta. Agostina Mottura

¿Por qué se necesita una memoria caché?



Tomando como referencia 1980, se presenta en el gráfico la brecha de performance, medida como la diferencia de tiempo entre la solicitud de un dato en memoria (para un solo procesador o núcleo) y la latencia de un acceso a DRAM. Note que el eje vertical está en escala logarítmica.

Hennessy and Patterson Ed.5 Image Copyright © 2011, Elsevier Inc. All rights Reserved



Factor clave: tiempo de acceso a memoria principal. La CPU lee instrucciones y operandos desde MP, pero la diferencia de velocidad entre ambos es...

	Tiempo de acceso
1 ciclo de CPU	0,15-0,3 ns
Memoria principal	50-150 ns
Unidad SSD	50-150 μ s
Disco rígido magnético	1-10ms

Factor clave: tiempo de acceso a memoria principal. La CPU lee instrucciones y operandos desde MP, pero la diferencia de velocidad entre ambos es...

	Tiempo de acceso	Equivalente unidad humana
1 ciclo de CPU	0,15-0,3 ns	1 segundos
Memoria principal	50-150 ns	3-9 minutos
Unidad SSD	50-150 μ s	2-5 días
Disco rígido magnético	1-10ms	1-12 meses

¿Cómo logra la memoria caché acortar las diferencias?

Factor clave: tiempo de acceso a memoria principal. La CPU lee instrucciones y operandos desde MP, pero la diferencia de velocidad entre ambos es...

	Tiempo de acceso	Equivalente unidad humana
1 ciclo de CPU	0,15-0,3 ns	1 segundos
Caché de nivel 1	0,9 ns	3 segundos
Caché de nivel 2	2,8 ns	9 segundos
Caché de nivel 3	12, 9 ns	43 segundos
Memoria principal	50-150 ns	3-9 minutos
Unidad SSD	50-150 μ s	2-5 días
Disco rígido magnético	1-10ms	1-12 meses

¿Cómo logra la memoria caché acortar las diferencias?

¿Qué función cumple la memoria caché?

Reduce considerablemente el tiempo de acceso a los datos e instrucciones que requiere la CPU.

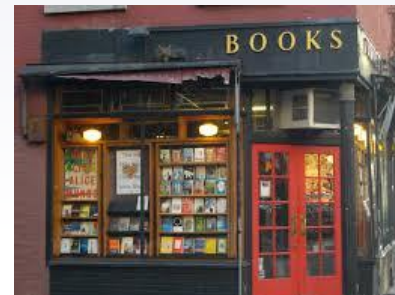
¿Cuánto tardo en acceder al contenido...



...de un libro en mi escritorio?



...de un libro que debo retirar de Biblioteca?



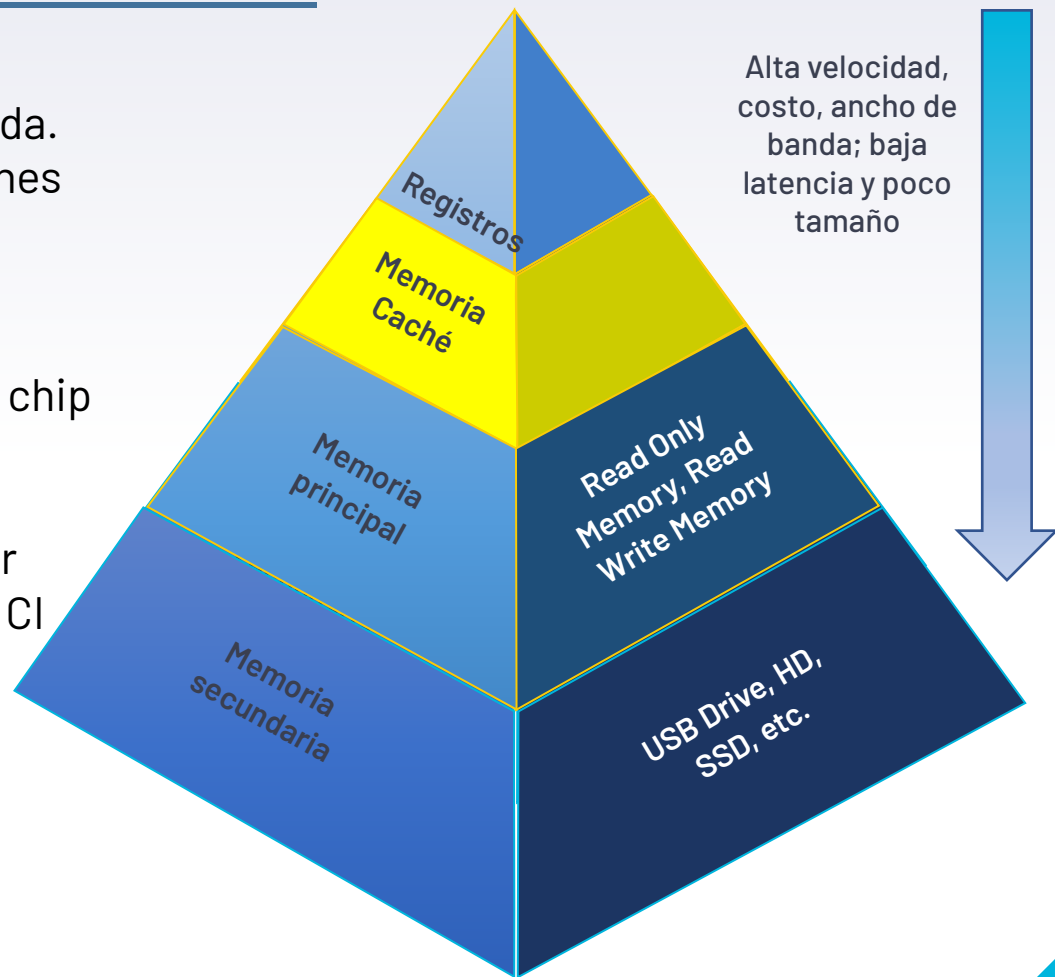
...de un libro que debo comprar?

¡Sería muy bueno tener en el escritorio todo lo que necesito!

¿Qué función cumple la memoria caché?

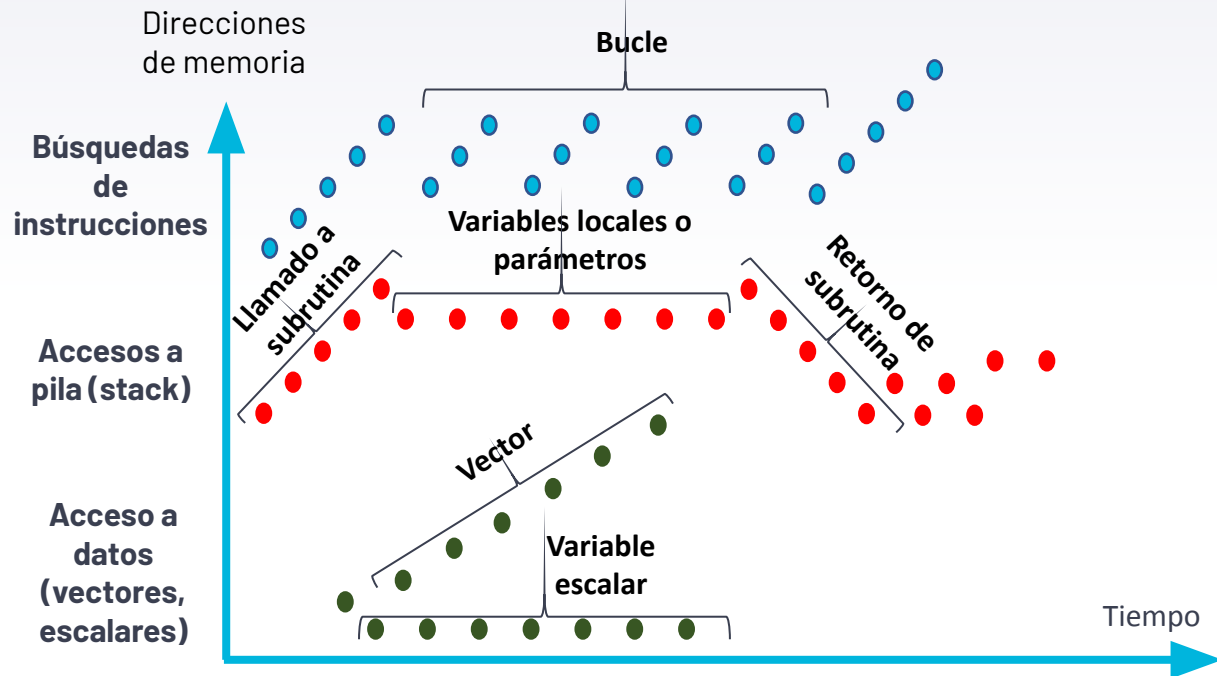
- La memoria caché es una memoria relativamente pequeña pero muy rápida.
- Su propósito es acelerar las operaciones CPU \leftrightarrow Memoria.
- Lógicamente se ubica entre CPU y la memoria principal.
- Físicamente puede estar en el mismo chip de la CPU o en un módulo separado (y ambos si es multinivel).
 - El ancho de banda es muy superior cuando se encuentra en el mismo CI (on-chip)

La CPU sigue accediendo sólo a la memoria principal. La memoria caché opera en forma transparente a las operaciones de la CPU.



Patrones comunes y predecibles en el acceso a memoria

*Si volcáramos los accesos a memoria a un gráfico...
¿cómo se vería?*



Principio de localidad temporal

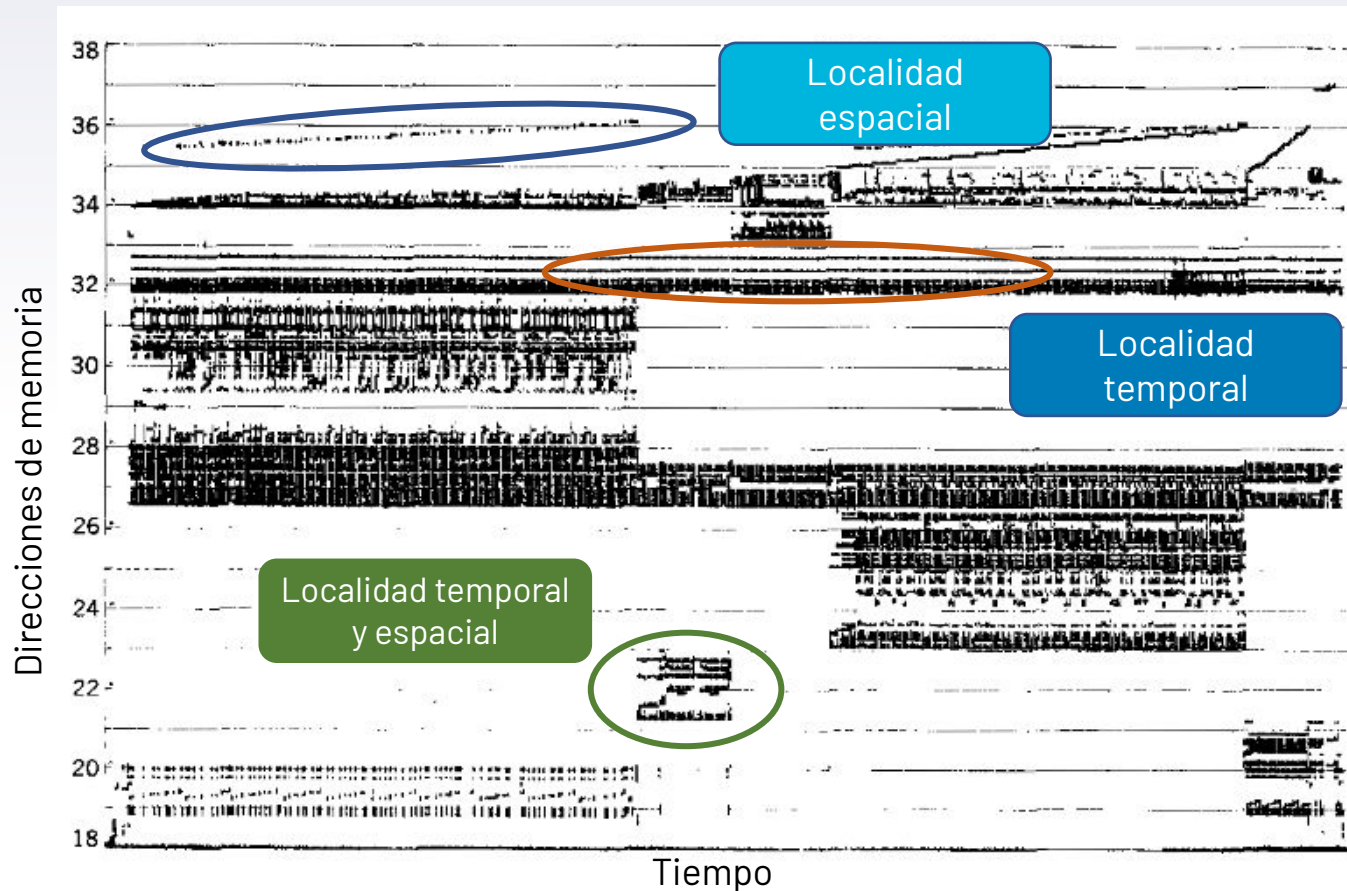
temporal: cuando se accede a una instrucción o dato es altamente probable que se la vuelva a acceder en el futuro cercano.

Principio de localidad espacial

espacial: cuando se accede a una instrucción o dato es altamente probable que las instrucciones o datos cercanos sean accedidos en el futuro cercano.

El 90% del tiempo de ejecución se consume el 10% del código

Principio de localidad temporal y espacial



¿Encuentra los patrones en el acceso a memoria?

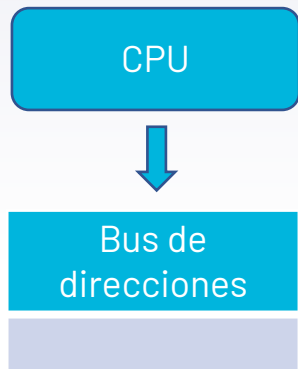
Cada punto representa un acceso a memoria.

Fuente: Program restructuring for virtual memory, DJ Hatfield, J Gerald - IBM Systems Journal, 1971.

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.



Memoria Caché

Dirección	Contenido

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	

La memoria se carga a medida que se requiere.

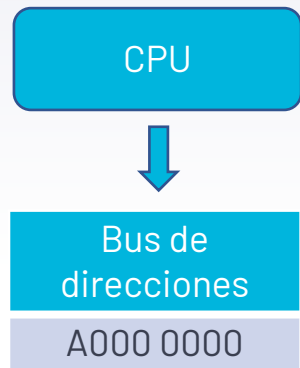
Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.



Memoria Caché

Dirección	Contenido

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	

La memoria se carga a medida que se requiere.

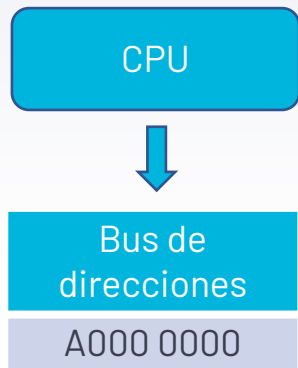
Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.

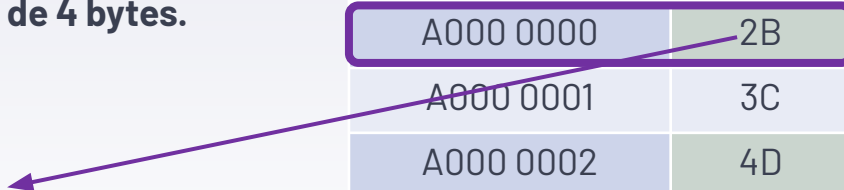


Memoria Caché

Dirección	Contenido

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	



La memoria se carga a medida que se requiere.

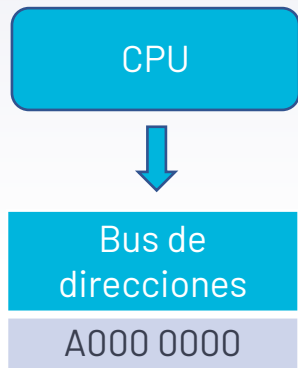
Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.

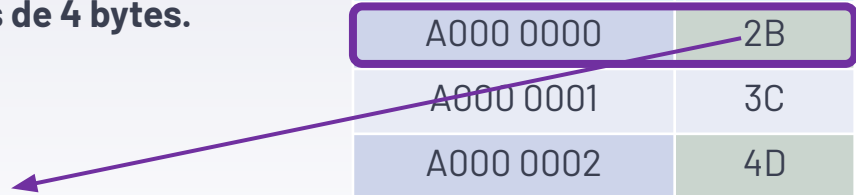


Memoria Caché

Dirección	Contenido
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	



La memoria se carga a medida que se requiere.

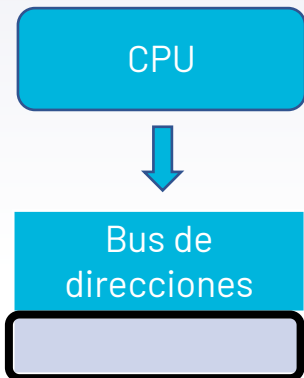
Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.

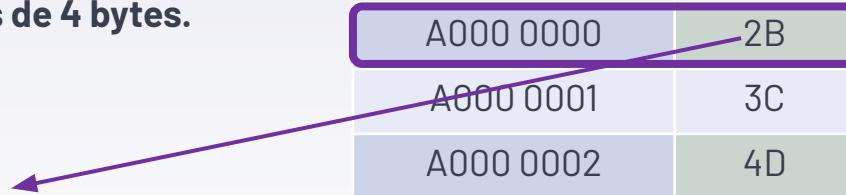


Memoria Caché

Dirección	Contenido
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	



La memoria se carga a medida que se requiere.

Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.

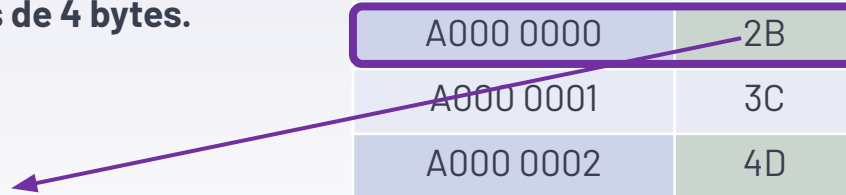


Memoria Caché

Dirección	Contenido
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	



La memoria se carga a medida que se requiere.

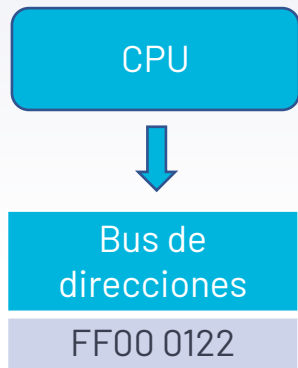
Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.



Memoria Caché

Dirección	Contenido
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	

La memoria se carga a medida que se requiere.

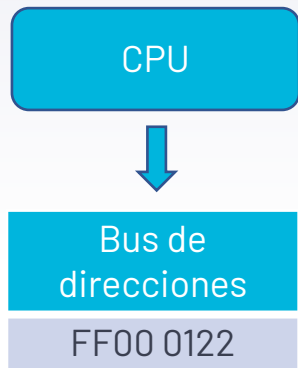
Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Una memoria más rápida

Dada una MP, implementamos una MC de 8 bytes.

Por el principio de localidad espacial usaremos bloques de 4 bytes.

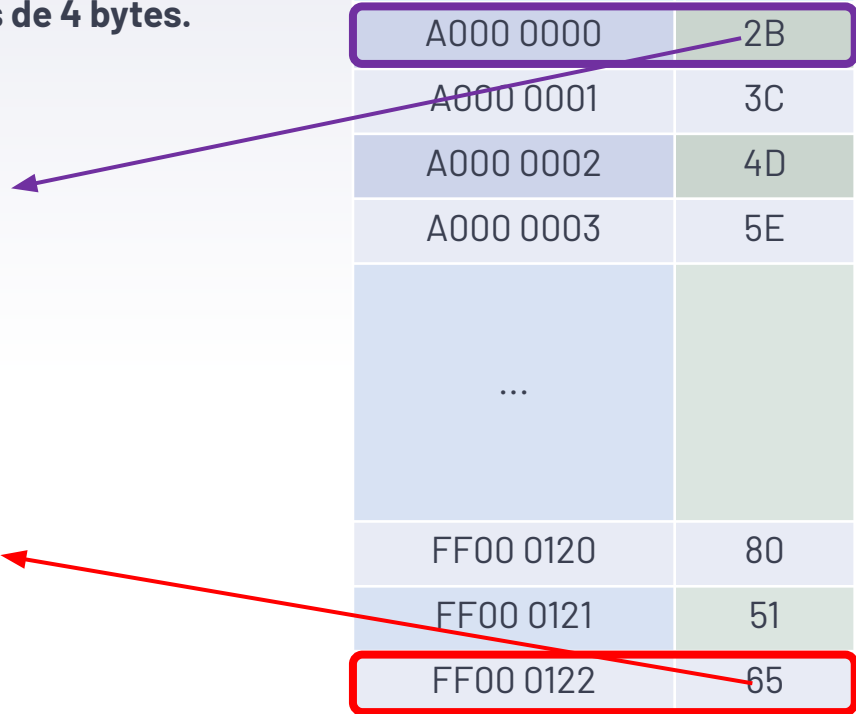


Memoria Caché

Dirección	Contenido
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	...
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02

Memoria Principal

...	
A000 0000	2B
A000 0001	3C
A000 0002	4D
A000 0003	5E
...	
FF00 0120	80
FF00 0121	51
FF00 0122	65
FF00 0123	02
...	



La memoria se carga a medida que se requiere.

Observe que se carga un bloque entero

¡Debemos almacenar *dirección y contenido*!

Agrupando de a 4, sacamos “factor común” de la dirección y almacenamos menos bits de la dirección... y la guardamos una sola vez en la ETIQUETA.

Dirección	Contenido 00	Contenido 01	Contenido 10	Contenido 11
1010 0000 0000 00 A 0 0	2B	3C	4D	5E

Dirección	Contenido 00	Contenido 01	Contenido 10	Contenido 11
1111 0001 0010 11 F 1 2	80	51	65	02

...	
A000	2B
A001	3C
A002	4D
A003	5E
...	
F12C	80
F12D	51
F12E	65
F12F	02
...	

Agrupando de a 4, sacamos "factor común" de la dirección y almacenamos menos bits de la dirección... y la guardamos una sola vez en la ETIQUETA.

Dirección	Contenido 00	Contenido 01	Contenido 10	Contenido 11
1010 0000 0000 00 A 0 0	2B	3C	4D	5E

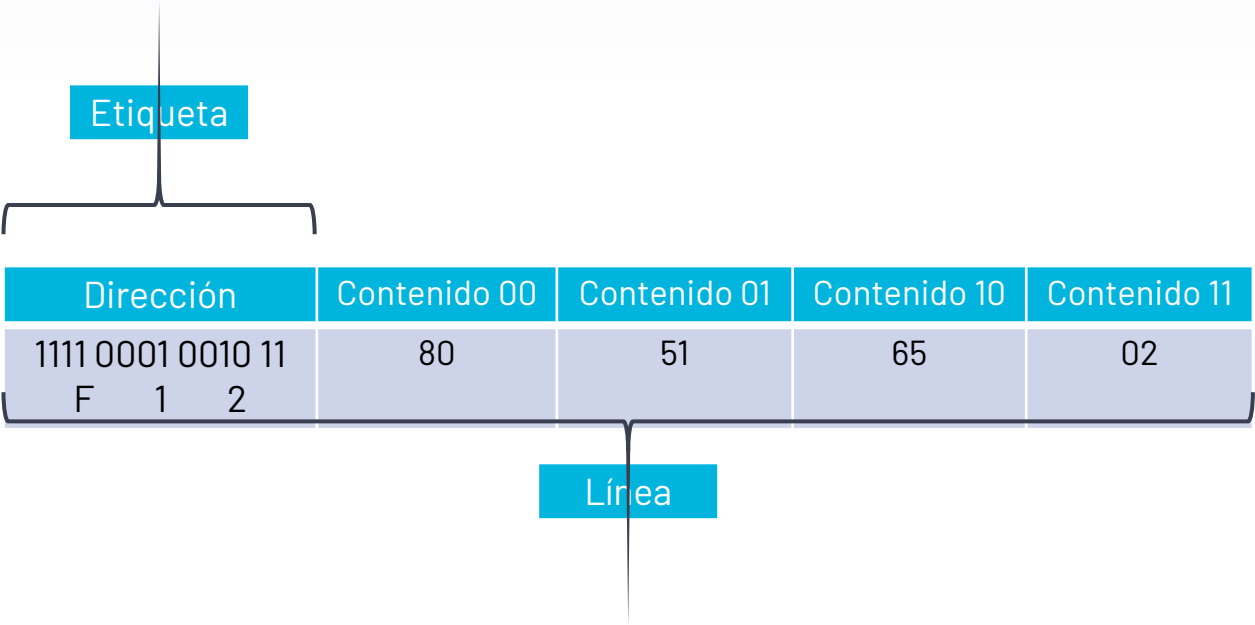
Dirección	Contenido 00	Contenido 01	Contenido 10	Contenido 11
1111 0001 0010 11 F 1 2	80	51	65	02

Línea

...	
A000	2B
A001	3C
A002	4D
A003	5E
...	
F12C	80
F12D	51
F12E	65
F12F	02
...	

Agrupando de a 4, sacamos "factor común" de la dirección y almacenamos menos bits de la dirección... y la guardamos una sola vez en la ETIQUETA.

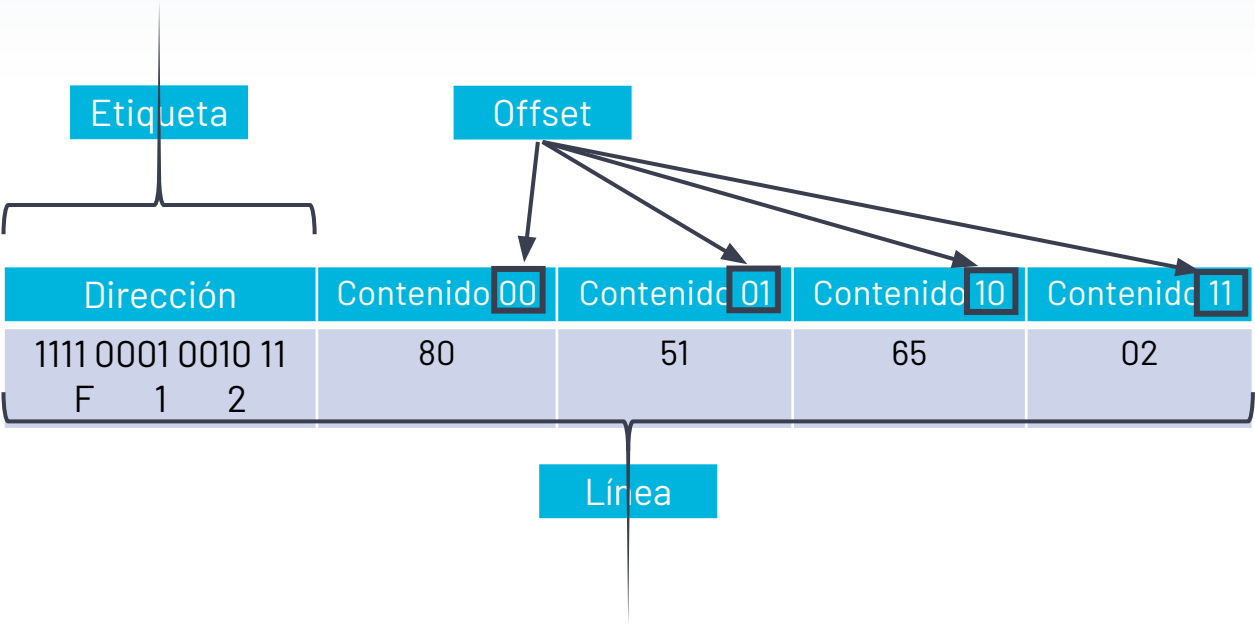
Dirección	Contenido 00	Contenido 01	Contenido 10	Contenido 11
1010 0000 0000 00 A 0 0	2B	3C	4D	5E



...	
A000	2B
A001	3C
A002	4D
A003	5E
...	
F12C	80
F12D	51
F12E	65
F12F	02
...	

Agrupando de a 4, sacamos "factor común" de la dirección y almacenamos menos bits de la dirección... y la guardamos una sola vez en la ETIQUETA.

Dirección	Contenido 00	Contenido 01	Contenido 10	Contenido 11
1010 0000 0000 00 A 0 0	2B	3C	4D	5E

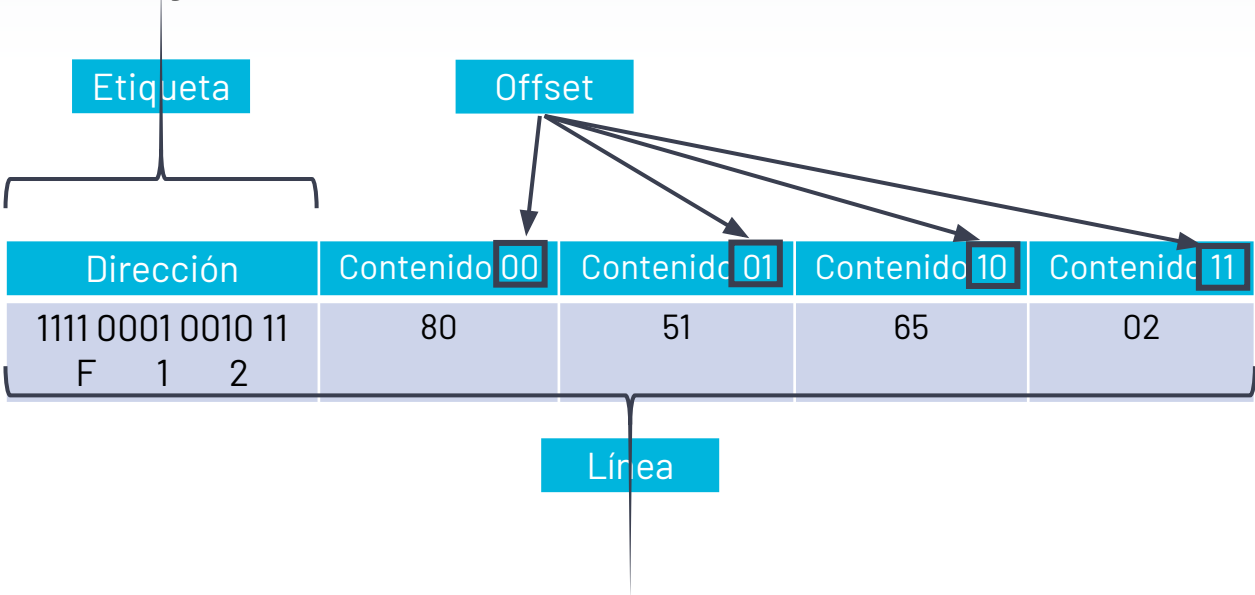


...	
A000	2B
A001	3C
A002	4D
A003	5E
...	
F12C	80
F12D	51
F12E	65
F12F	02
...	

Agrupando de a 4, sacamos "factor común" de la dirección y almacenamos menos bits de la dirección... y la guardamos una sola vez en la ETIQUETA.

Dirección	Contenido 00	Contenido 01	Contenido 10	Contenido 11
1010 0000 0000 00 A 0 0	2B	3C	4D	5E

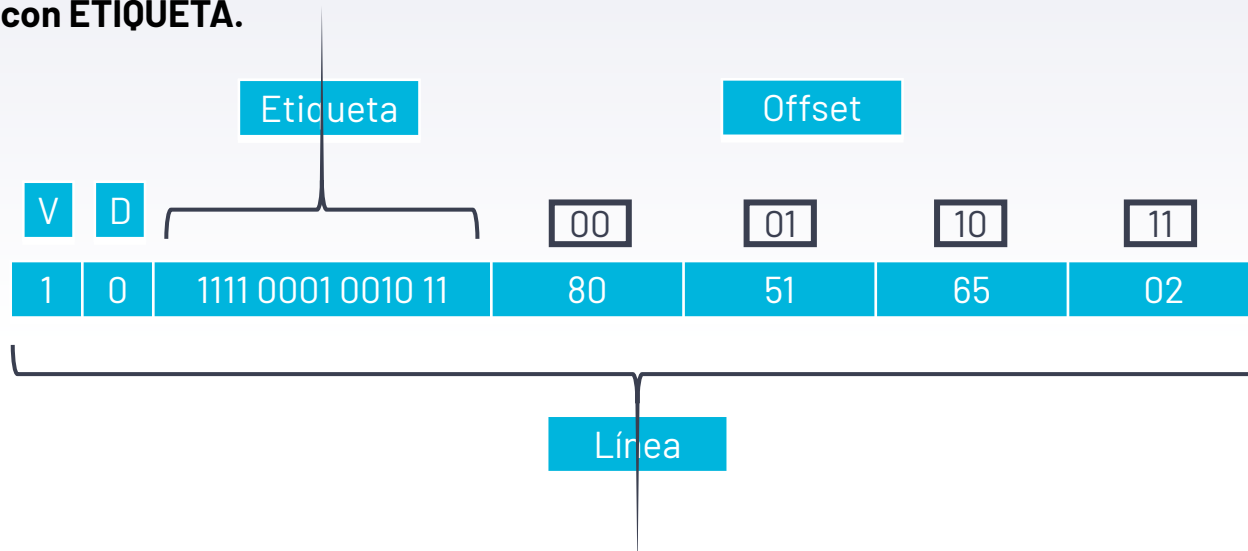
Cuando se requiera acceder a una palabra en caché se usará la parte menos significativa de la dirección como OFFSET para ubicarlo en la línea.



...	
A000	2B
A001	3C
A002	4D
A003	5E
...	
F12C	80
F12D	51
F12E	65
F12F	02
...	

La ETIQUETA se guarda en cada línea. El OFFSET no se guarda, se utiliza para seleccionar la palabra requerida.

Cada línea de MC contendrá una etiqueta y el contenido de las posiciones de memoria cuyas direcciones comienzan con ETIQUETA.

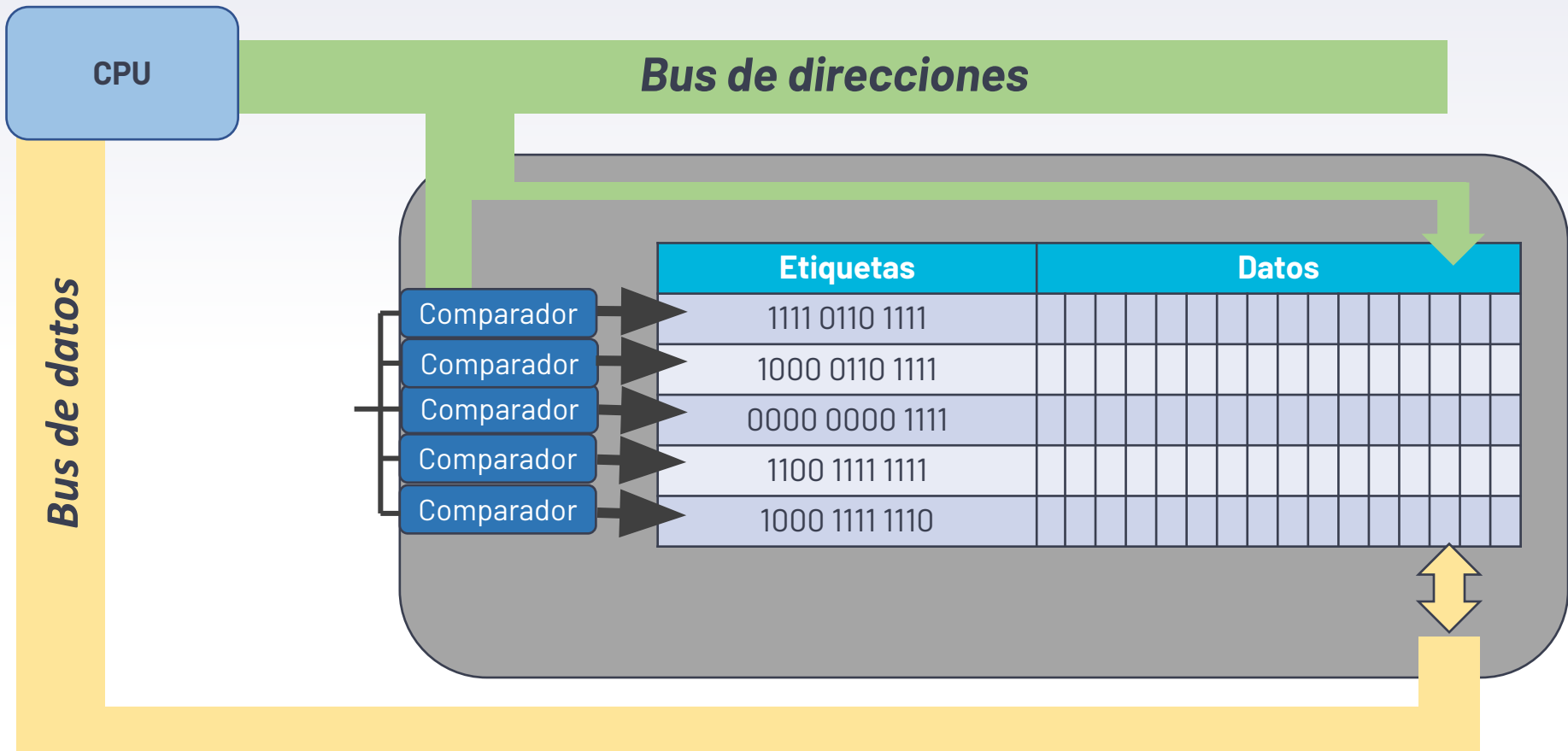


Bits de control: *(Los fija y mantiene la MMU)*

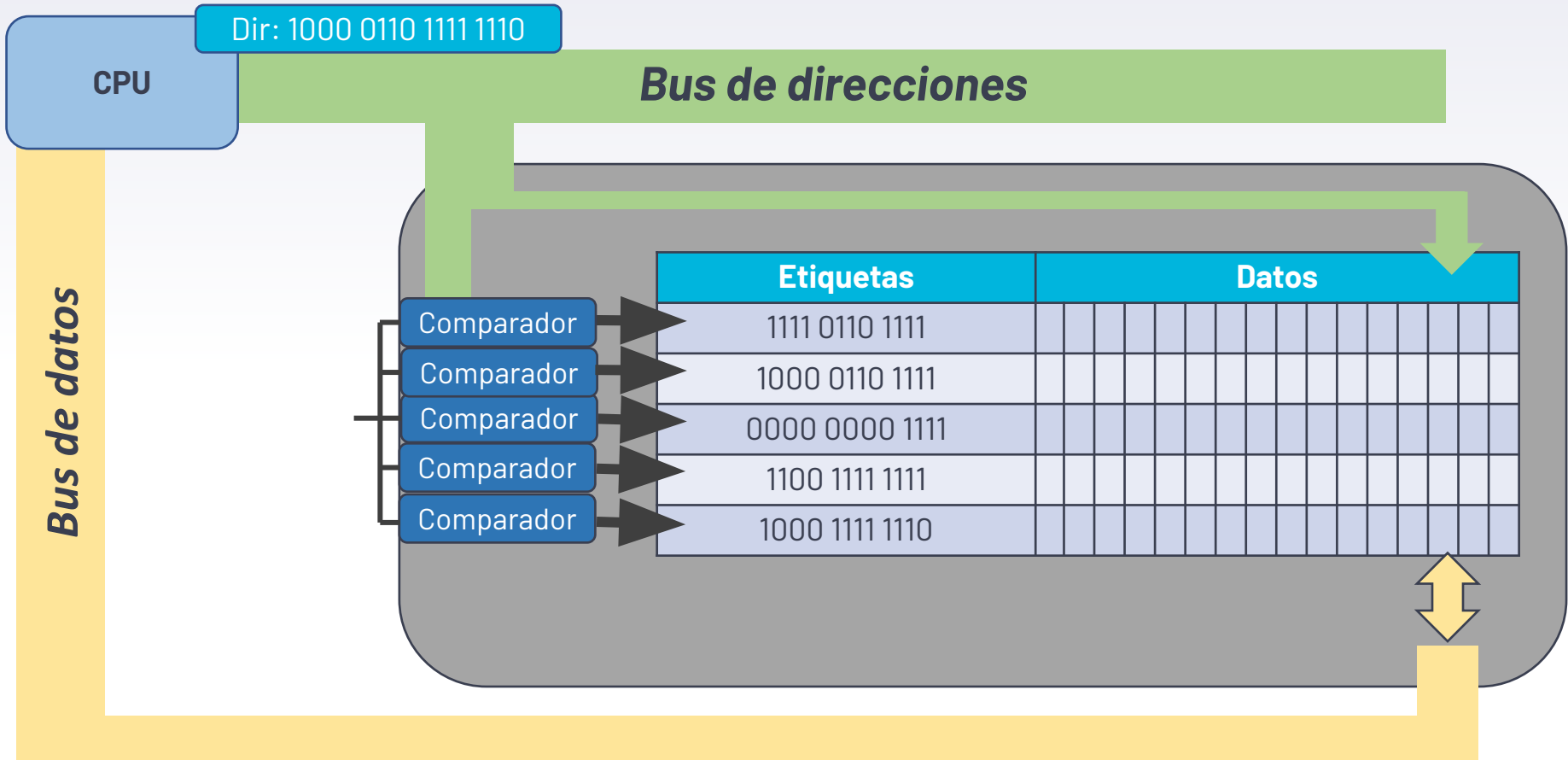
Validez: indica si el contenido de la línea debe tenerse en cuenta en los accesos a MC.

Suciedad (Dirty): indica si el contenido de la línea se modificó luego de su carga, y debe copiarse en MP antes de ser reemplazada.

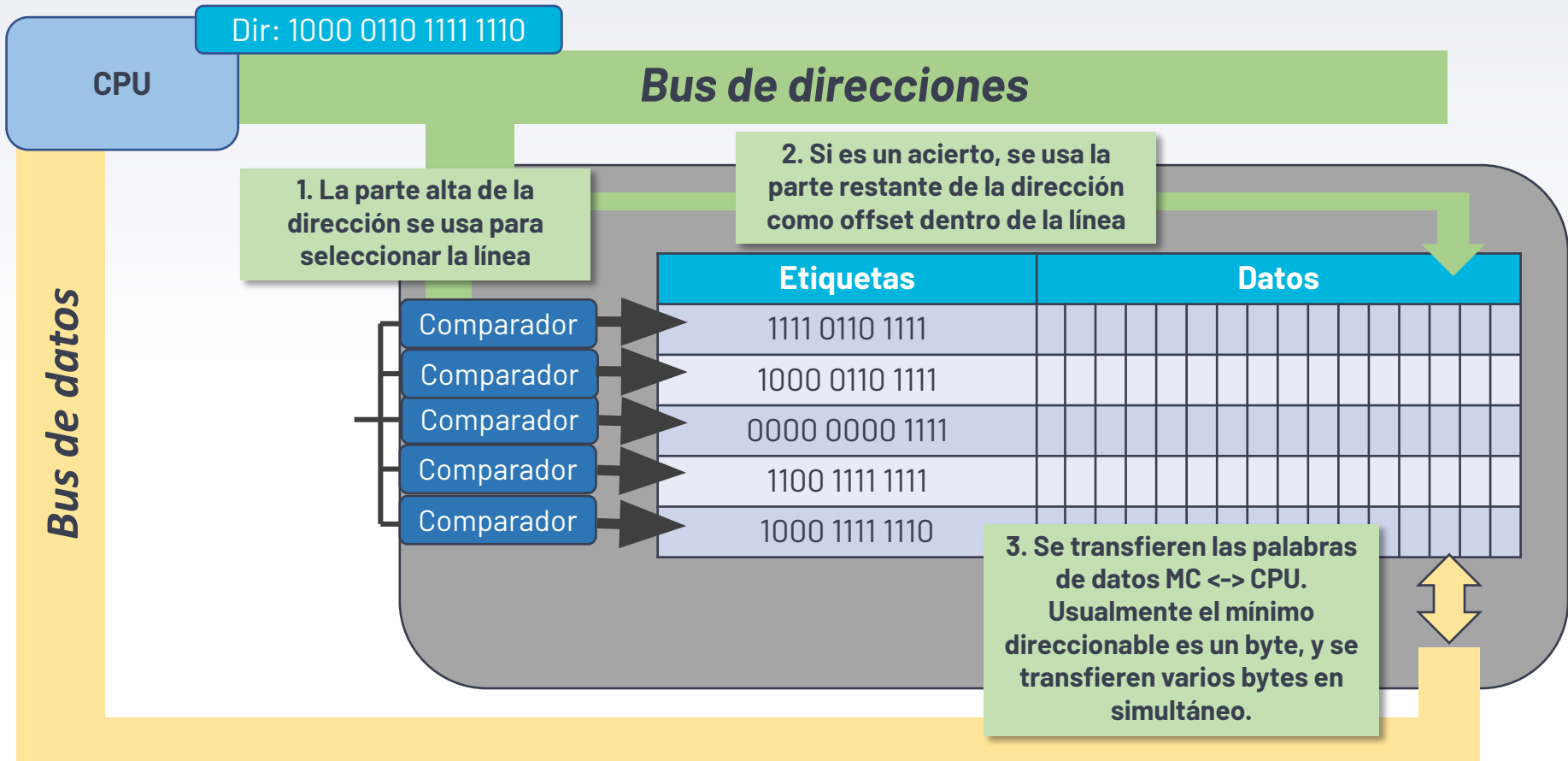
Zoom en la memoria caché



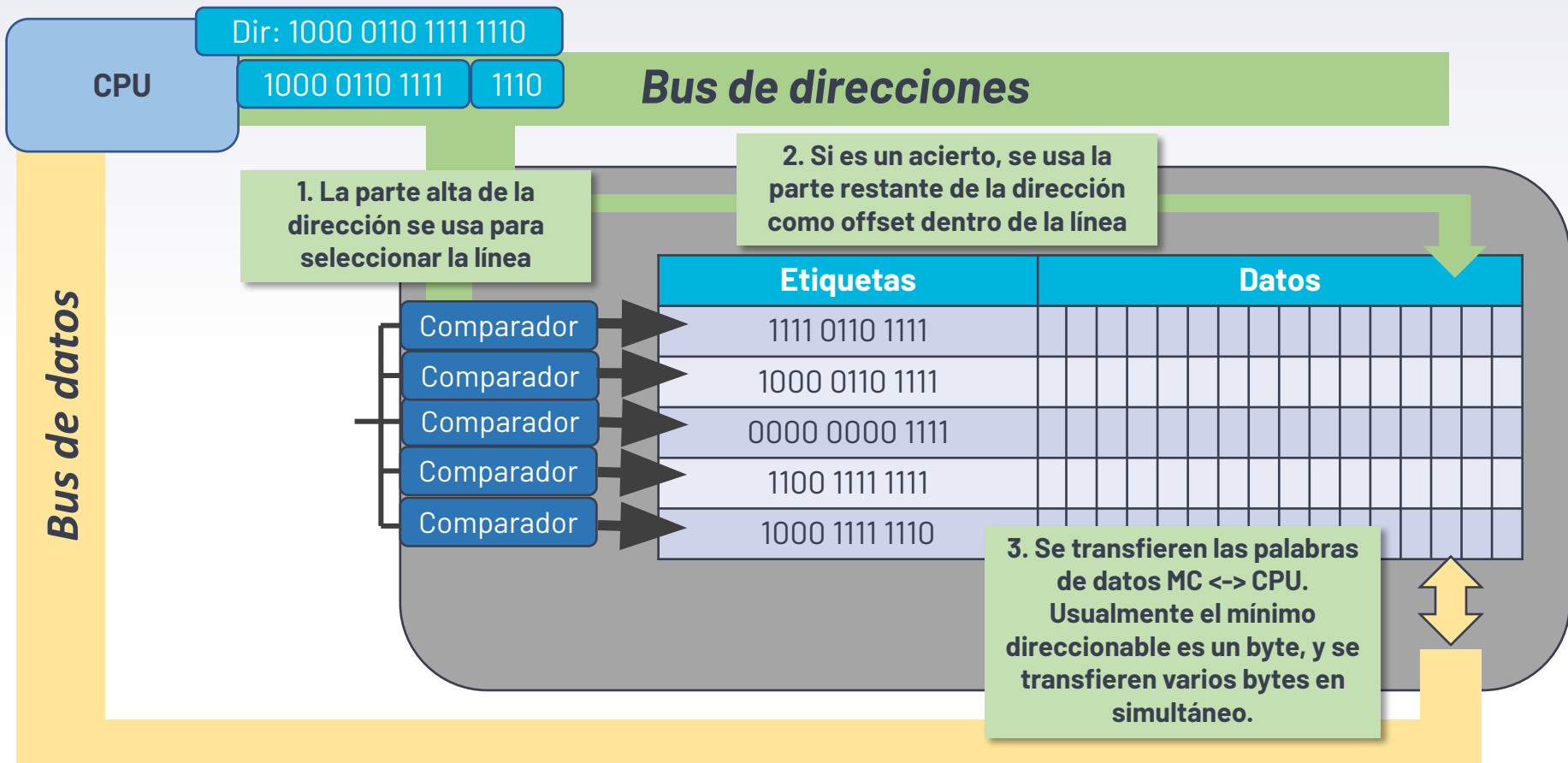
Zoom en la memoria caché



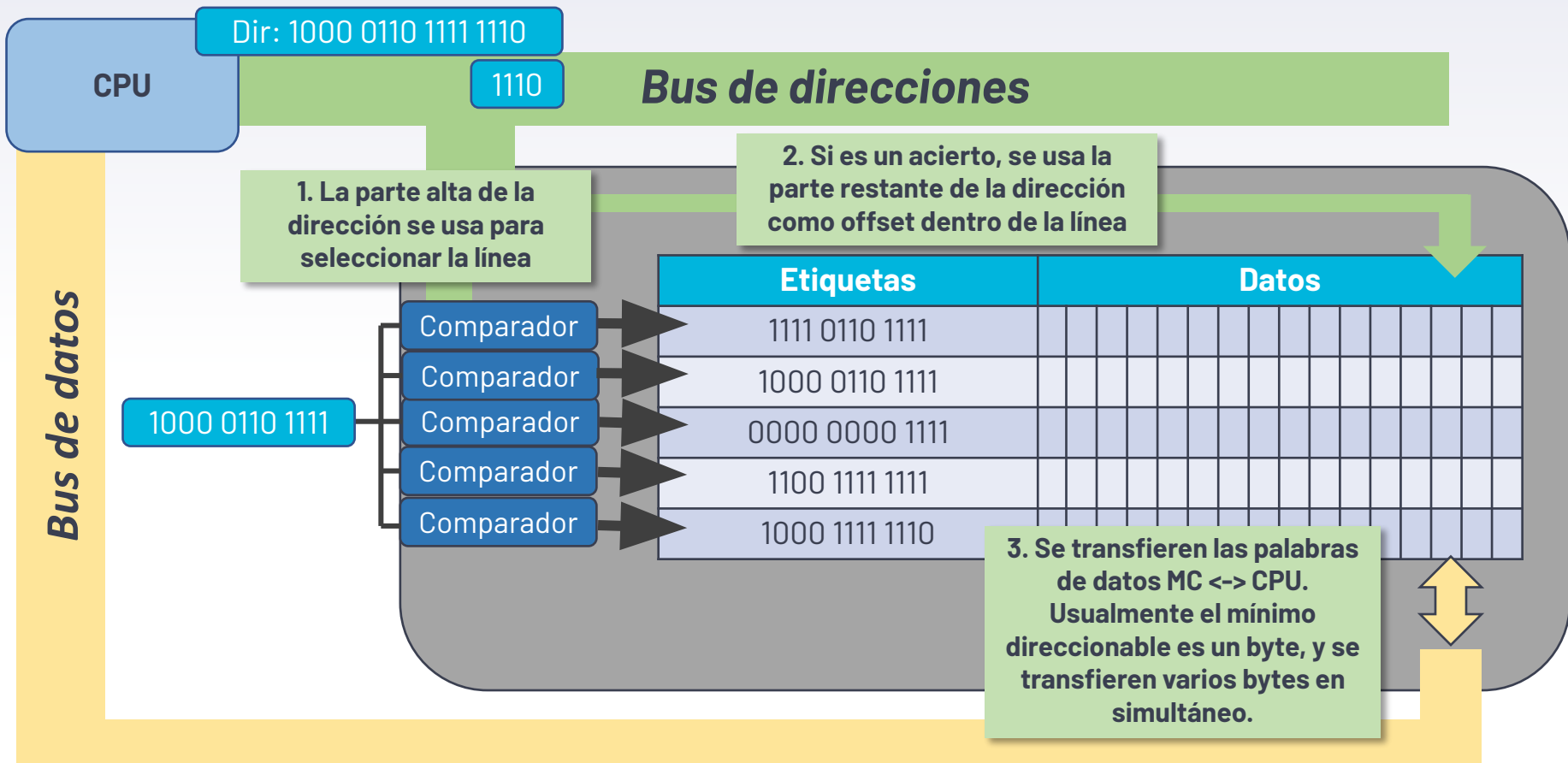
Zoom en la memoria caché



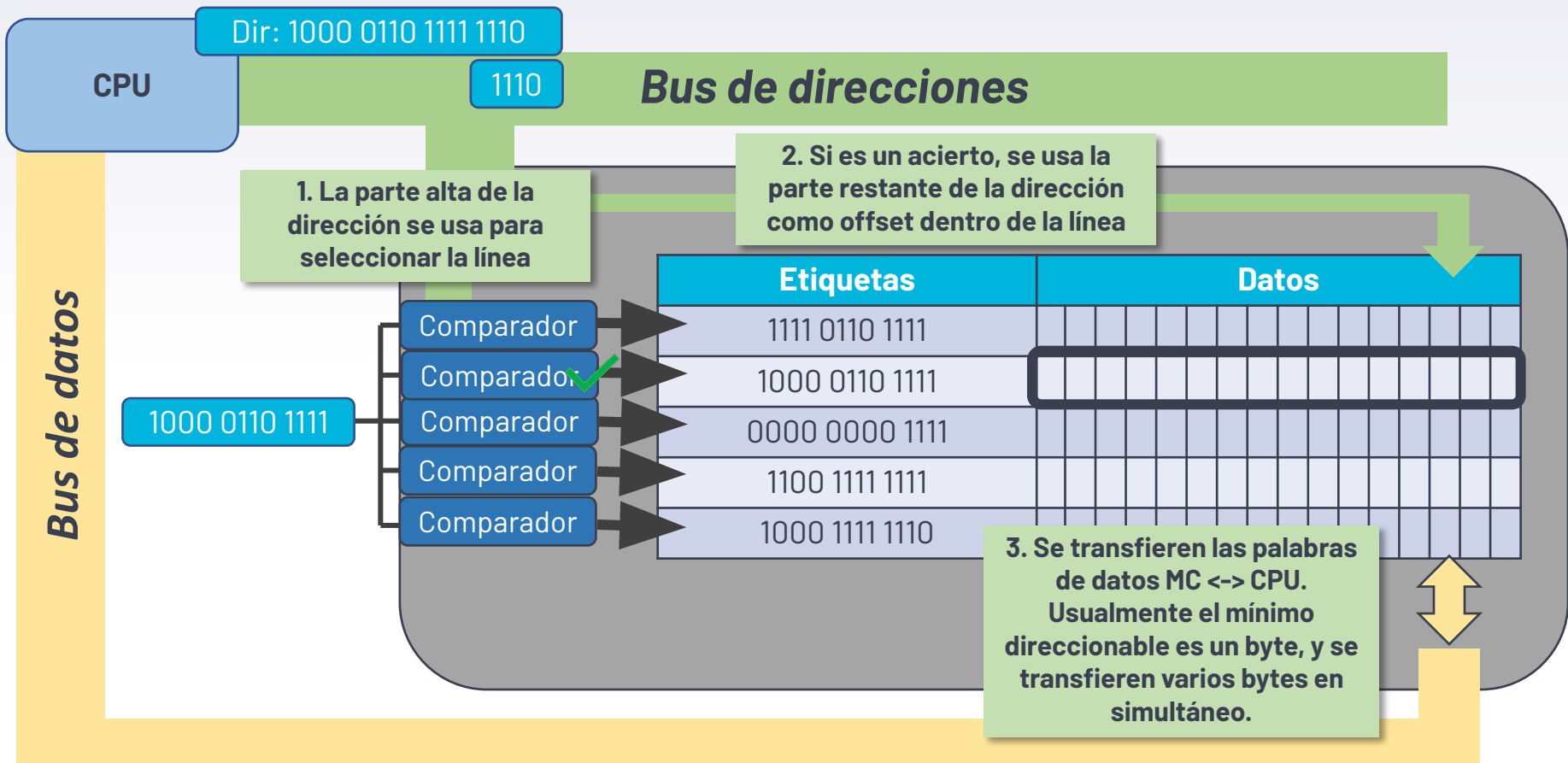
Zoom en la memoria caché



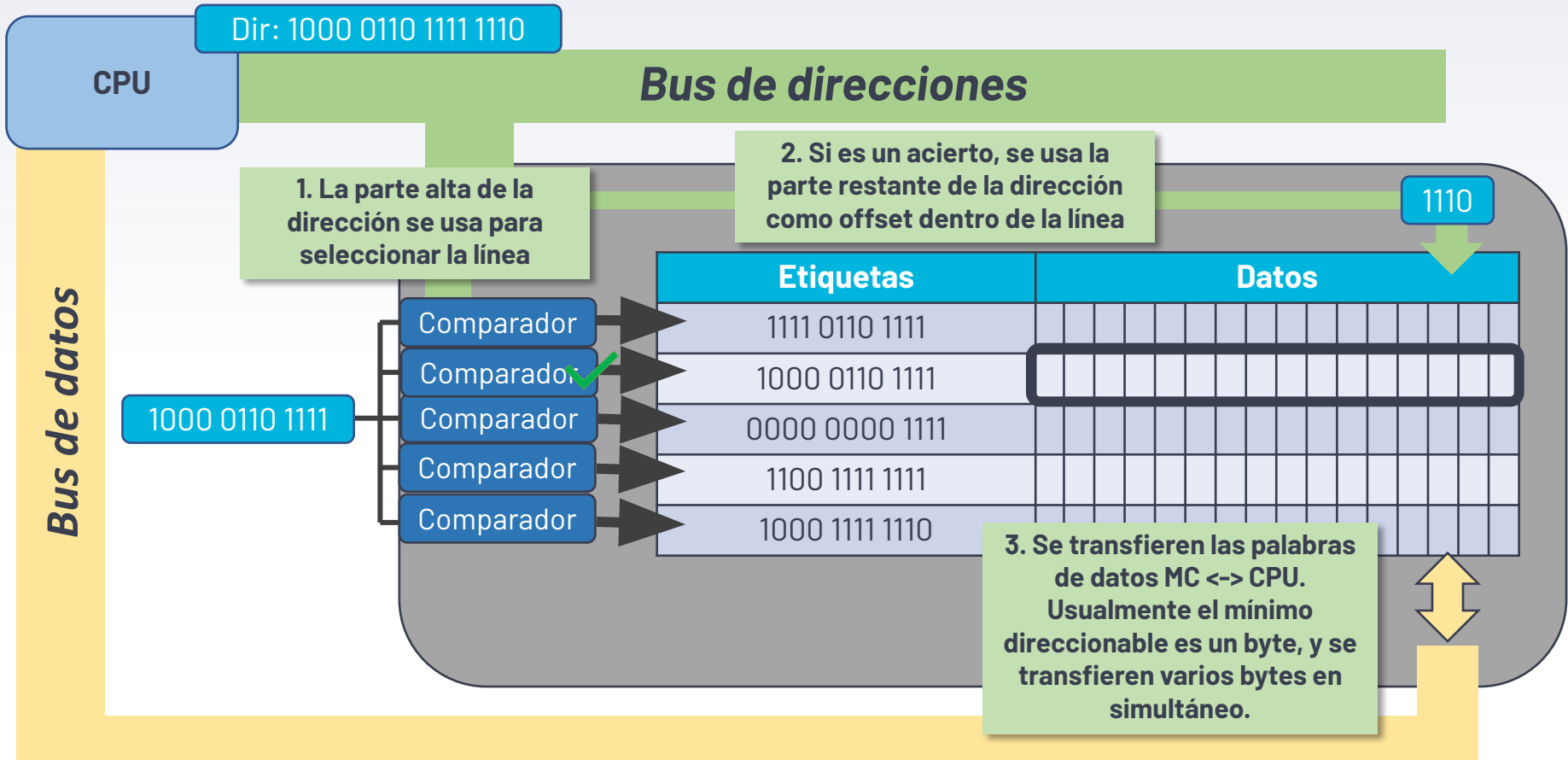
Zoom en la memoria caché



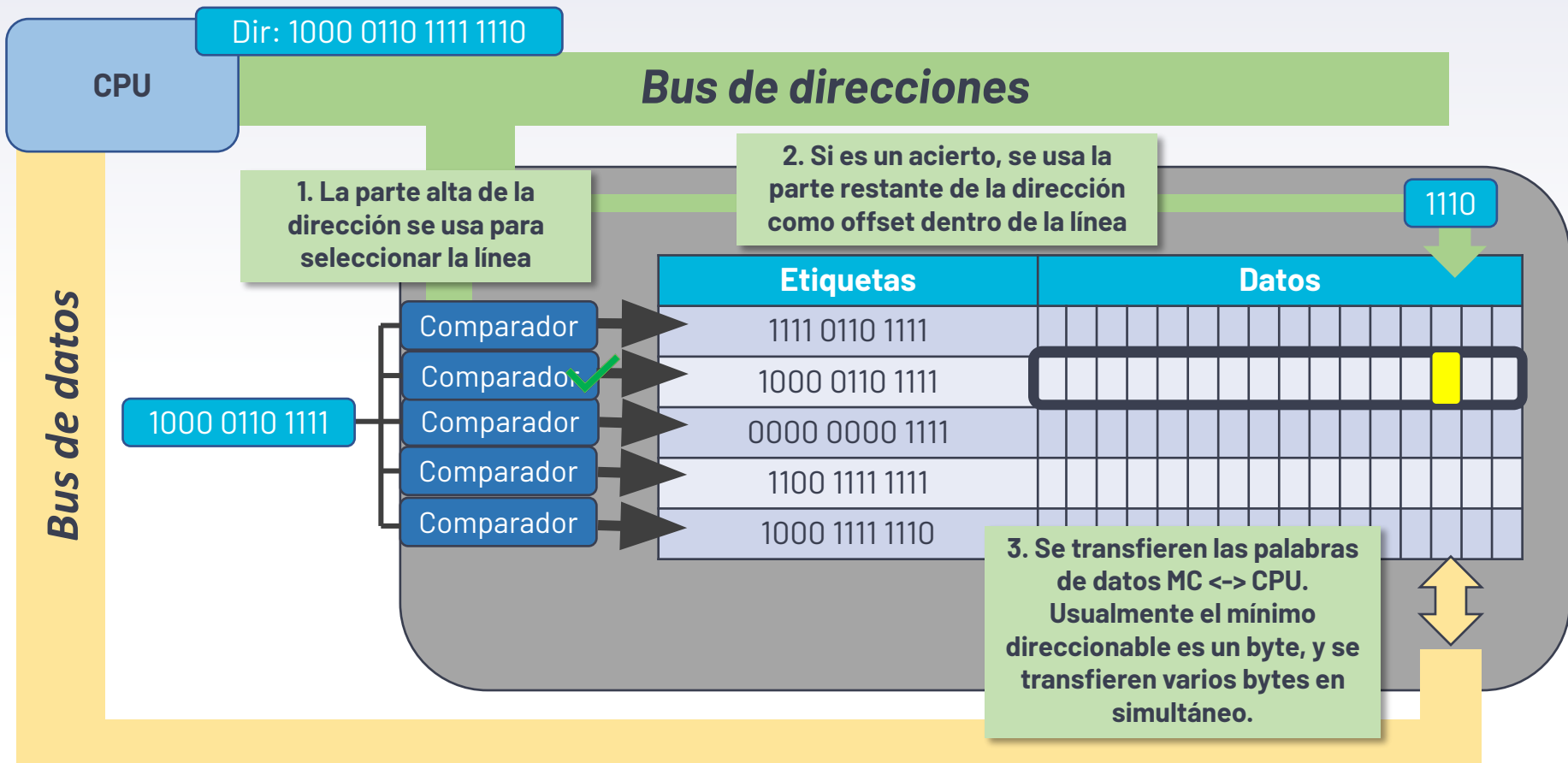
Zoom en la memoria caché



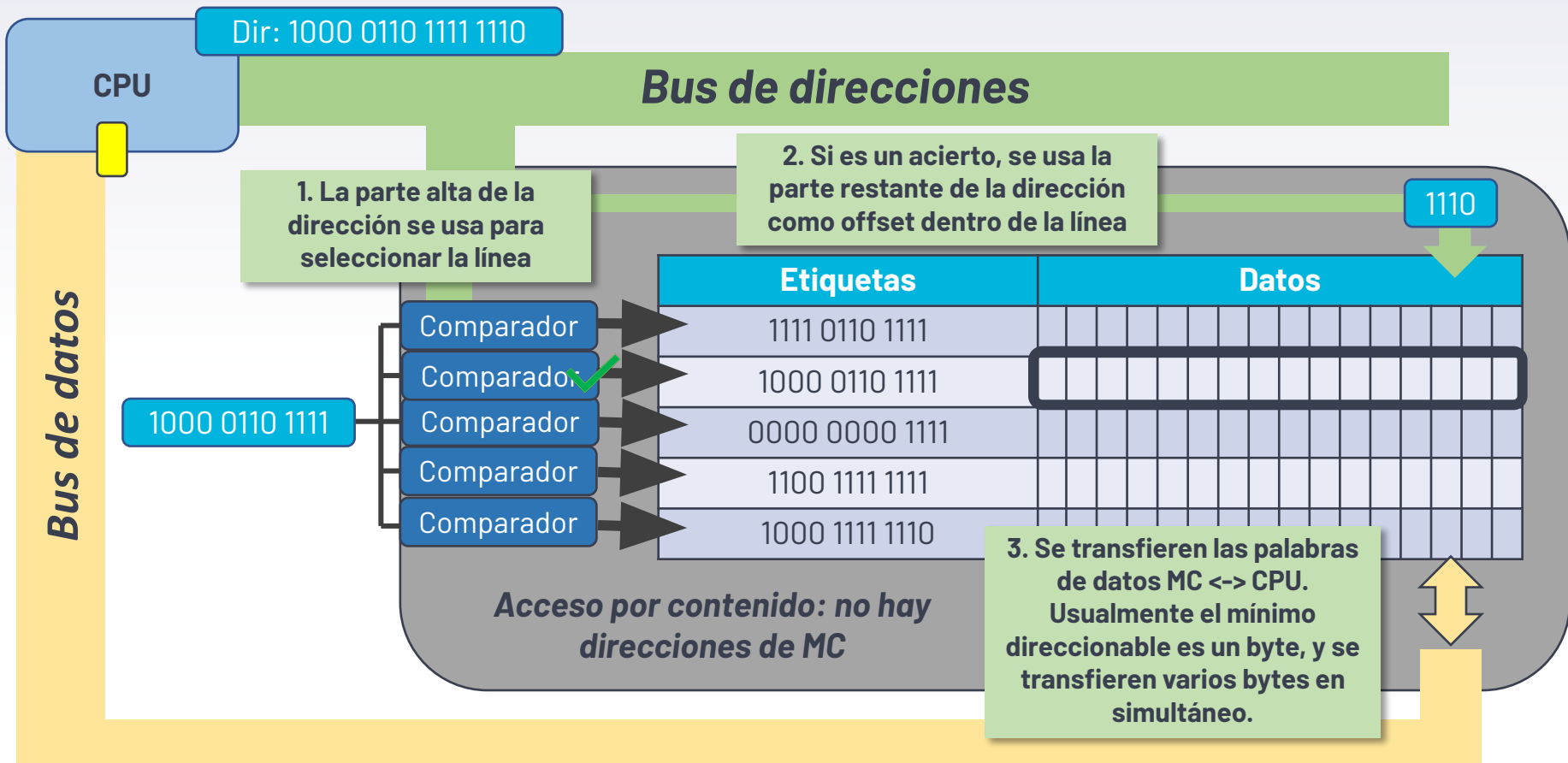
Zoom en la memoria caché



Zoom en la memoria caché



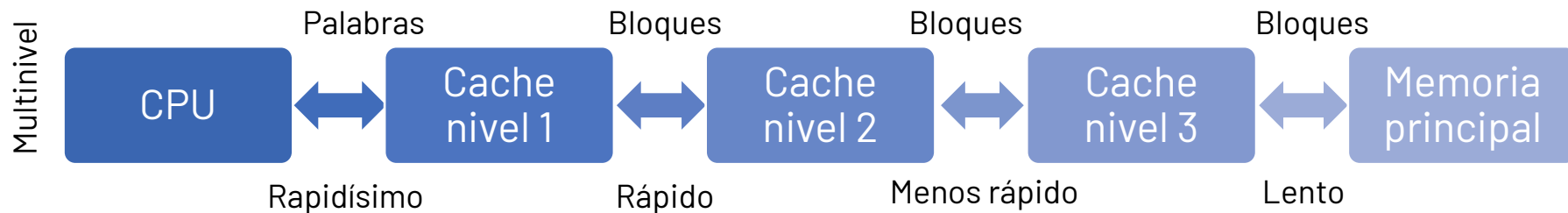
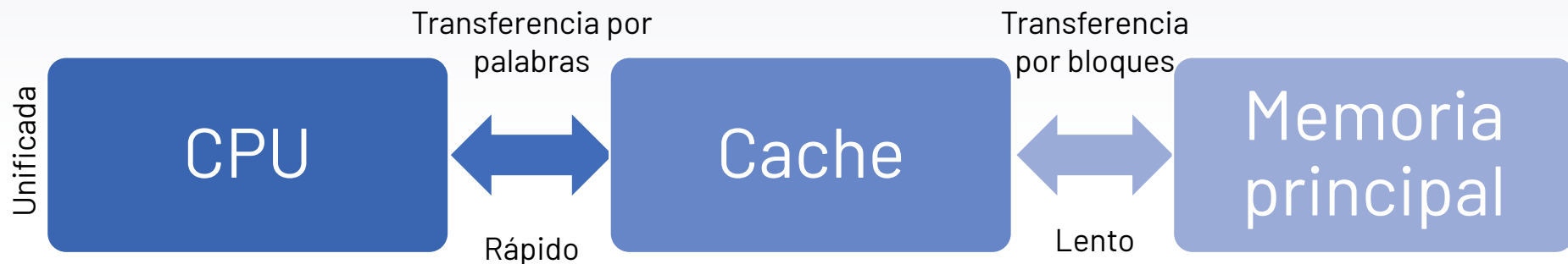
Zoom en la memoria caché



Cómo funciona la memoria caché

La memoria caché contiene una **copia** de una **porción** de memoria principal.

1. CPU solicita acceso a una dirección de memoria
2. El administrador de la memoria caché (MMU) verifica si la dirección existe en la caché
3. Si está presente (HIT, éxito) se lee desde caché
4. Si no está presente en la caché (MISS, falla), se lee el bloque correspondiente de MP (*prefetch*) y se actualiza la MC



Características de la memoria caché

Construcción: memoria estática de alta velocidad (SRAM)

Direccionamiento: por contenido (CAM: *content addressable memory*).

No existen direcciones de caché, sino que se almacena la dirección y el contenido.

Utiliza comparadores para verificar si una dirección de MP está presente en MC.

Política de carga y escritura: en qué momento se actualiza la memoria principal cuando se modifica la caché. Cómo se realiza la lectura de MP para completar la MC.

Organización: Debe determinarse la existencia de una palabra en caché de forma extremadamente rápida (se analizan a continuación tres formas).

Tamaño de la memoria y del bloque: impactan en la performance por lo que se busca un equilibrio

Costo: la memoria caché se encarece al aumentar de tamaño por su propia construcción y por los comparadores que requiere para operar.

Organización: Asignación asociativa (fully associative)

Cada bloque de memoria principal puede ubicarse en cualquier línea de caché.

La búsqueda de etiquetas se realiza en paralelo, la palabra buscada puede estar en cualquier línea de caché!

Cada bloque de MP (auto) puede utilizar cualquier ubicación en el estacionamiento.



Asignación asociativa (fully associative)

Cualquier bloque de memoria principal puede ir a parar a cualquier línea de caché

Tamaño de la MC / tamaño de línea =
cantidad de líneas

Memoria principal
Bloque 0
Bloque 1
Bloque 2
Bloque 3
Bloque 4
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

Memoria Caché
Línea 0
Línea 1
Línea 2
Línea 3

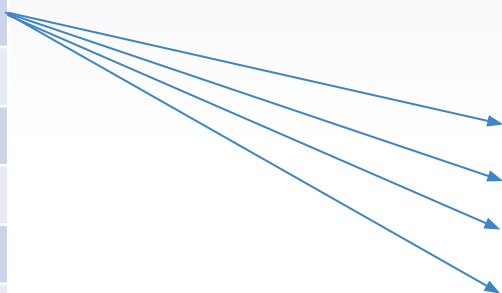
Línea MC	Bloque de MP que alojará
0	Cualquiera
1	Cualquiera
2	Cualquiera
3	Cualquiera

Asignación asociativa (fully associative)

Cualquier bloque de memoria principal puede ir a parar a cualquier línea de caché

Tamaño de la MC / tamaño de línea =
cantidad de líneas

Memoria principal
Bloque 0
Bloque 1
Bloque 2
Bloque 3
Bloque 4
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N



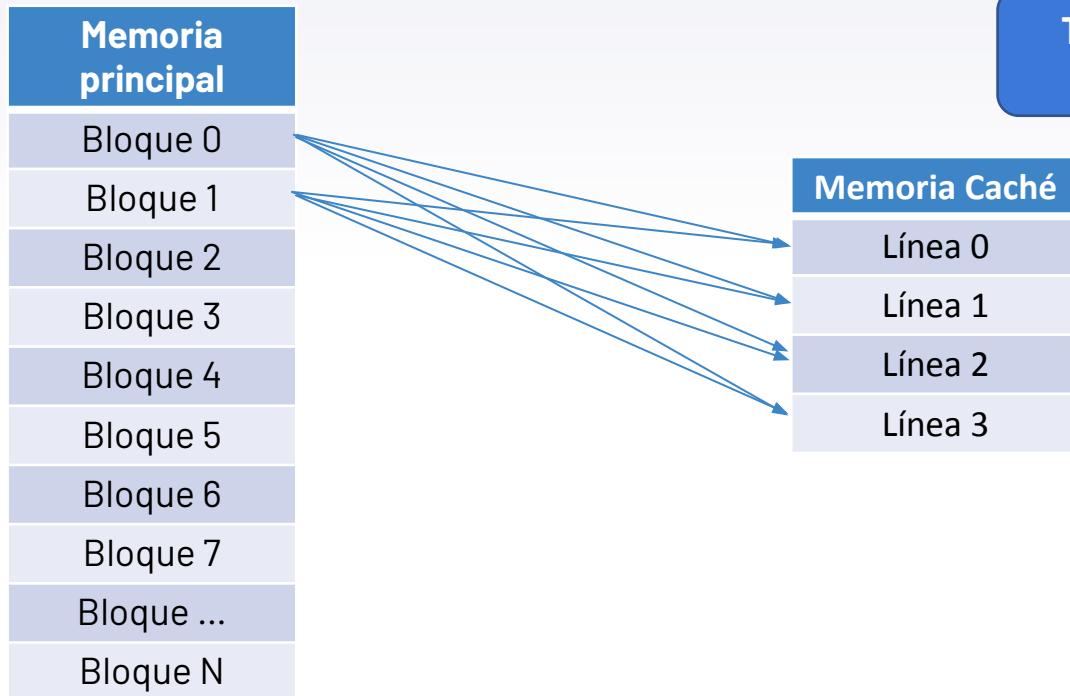
Memoria Caché
Línea 0
Línea 1
Línea 2
Línea 3

Línea MC	Bloque de MP que alojará
0	Cualquiera
1	Cualquiera
2	Cualquiera
3	Cualquiera

Asignación asociativa (fully associative)

Cualquier bloque de memoria principal puede ir a parar a cualquier línea de caché

Tamaño de la MC / tamaño de línea =
cantidad de líneas

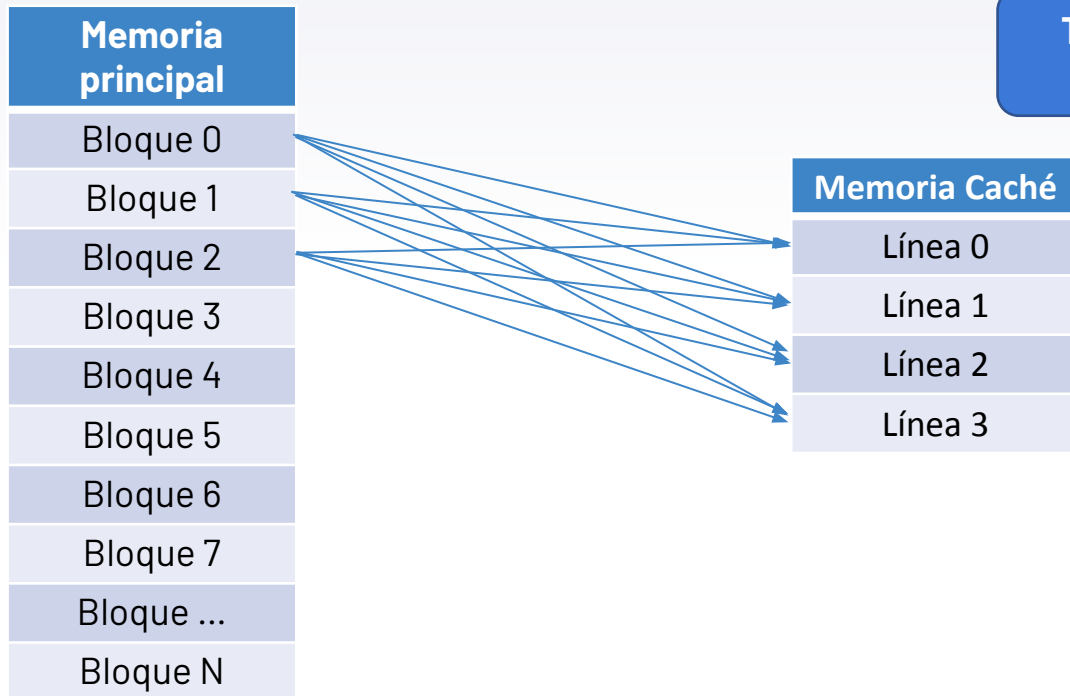


Línea MC	Bloque de MP que alojará
0	Cualquiera
1	Cualquiera
2	Cualquiera
3	Cualquiera

Asignación asociativa (fully asociative)

Cualquier bloque de memoria principal puede ir a parar a cualquier línea de caché

Tamaño de la MC / tamaño de línea =
cantidad de líneas

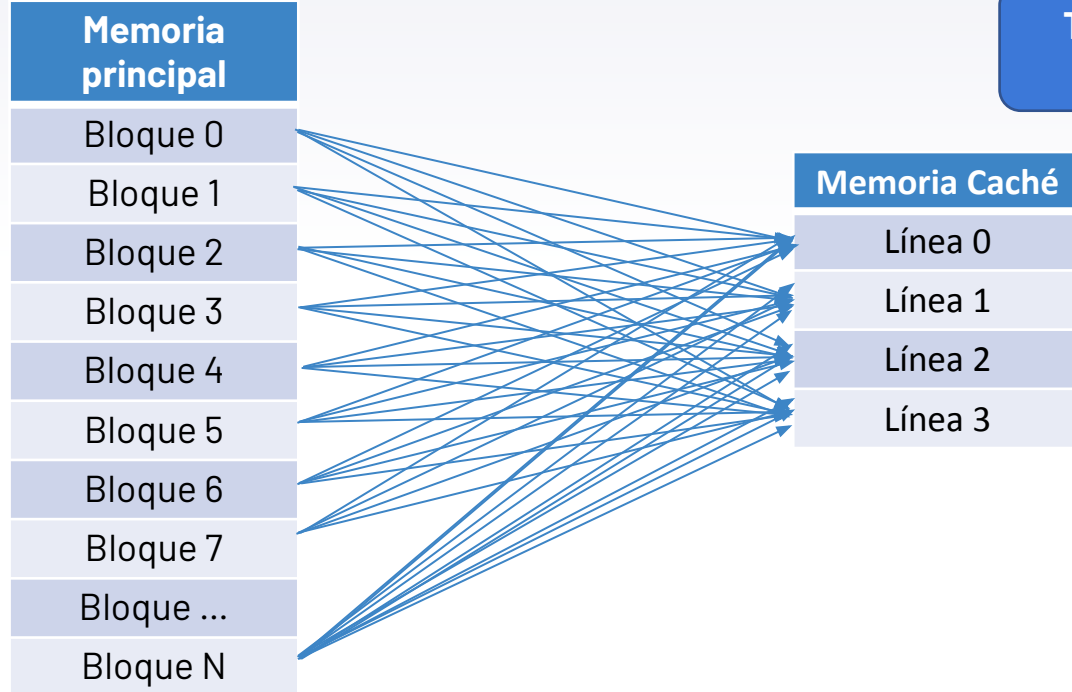


Línea MC	Bloque de MP que alojará
0	Cualquiera
1	Cualquiera
2	Cualquiera
3	Cualquiera

Asignación asociativa (fully asociative)

Cualquier bloque de memoria principal puede ir a parar a cualquier línea de caché

Tamaño de la MC / tamaño de línea =
cantidad de líneas

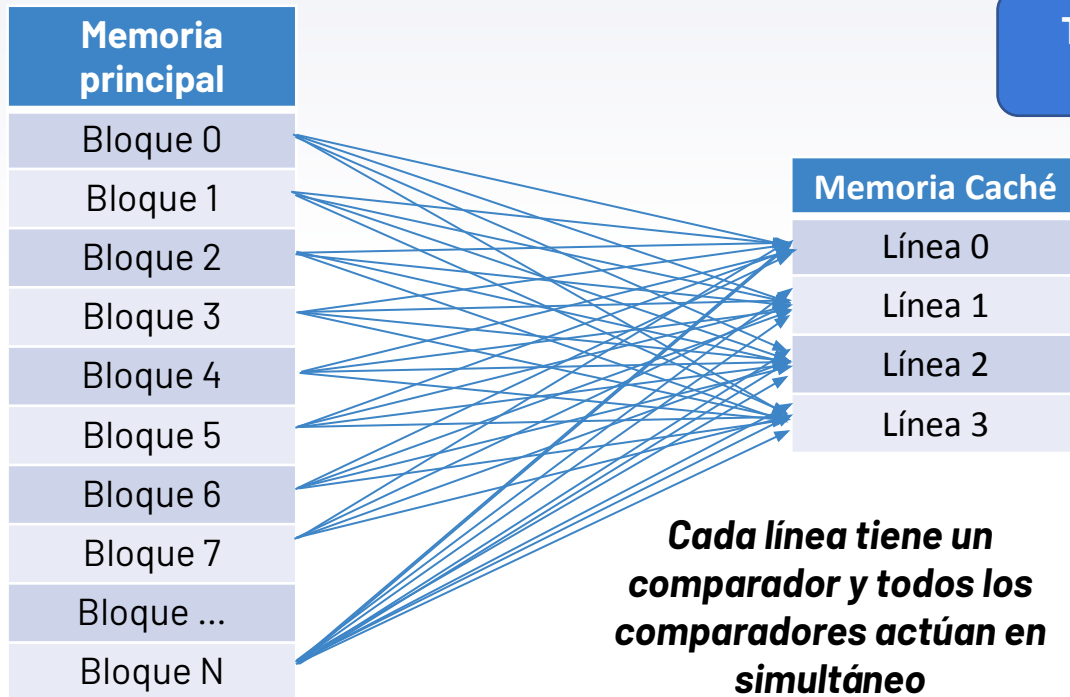


Línea MC	Bloque de MP que alojará
0	Cualquiera
1	Cualquiera
2	Cualquiera
3	Cualquiera

Asignación asociativa (fully asociative)

Cualquier bloque de memoria principal puede ir a parar a cualquier línea de caché

Tamaño de la MC / tamaño de línea =
cantidad de líneas



Línea MC	Bloque de MP que alojará
0	Cualquiera
1	Cualquiera
2	Cualquiera
3	Cualquiera

Asignación asociativa (fully associative)

Interpretación de cada dirección de memoria según la CPU la coloca en el bus.

$S+W$ = tamaño de dirección de memoria (ancho bus de direcciones)

Especifica un bloque de MP o línea de MC

Define la palabra dentro de la línea

Etiqueta (tag)

Palabra (offset)

$S = \log_2(\text{cantidad de bloques})$
 $S = \log_2(\text{tamaño MP/tamaño bloque})$

$W = \log_2(\text{tamaño de línea})$

Para determinar si una palabra se encuentra en MC se deben comparar todas las etiquetas de cada línea de caché en paralelo. Se necesita una memoria especial que permita dicha búsqueda (asociativa).

La comparación se efectúa solamente sobre el campo de etiqueta.

Asignación asociativa (fully associative)

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 32 bytes, tamaño de caché 128 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Etiqueta (tag)

Offset (palabra)

¿Debo multiplicar los 4G por 8? ¿O los 128K por 8? ¿Se multiplica por el tamaño de palabra?

Asignación asociativa (fully associative)

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 32 bytes, tamaño de caché 128 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea =

Etiqueta (tag)	Offset (palabra)
----------------	------------------

¿Debo multiplicar los 4G por 8? ¿O los 128K por 8? ¿Se multiplica por el tamaño de palabra?

Asignación asociativa (fully associative)

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 32 bytes, tamaño de caché 128 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = $128 \text{ KB} / 32 = 2^{17} / 2^5 = 2^{12} = \mathbf{4096 \text{ líneas}}$

Etiqueta (tag)

Offset (palabra)

¿Debo multiplicar los 4G por 8? ¿O los 128K por 8? ¿Se multiplica por el tamaño de palabra?

Asignación asociativa (fully associative)

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 32 bytes, tamaño de caché 128 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = $128 \text{ KB} / 32 = 2^{17} / 2^5 = 2^{12} = \mathbf{4096 \text{ líneas}}$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(32) = 5$



5 bits

¿Debo multiplicar los 4G por 8? ¿O los 128K por 8? ¿Se multiplica por el tamaño de palabra?

Asignación asociativa (fully associative)

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 32 bytes, tamaño de caché 128 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = 128 KB / 32 = $2^{17} / 2^5 = 2^{12} = \mathbf{4096 \text{ líneas}}$

Cantidad de bloques =

tamaño MP / tamaño de línea = 4 GB / 32 bytes = $2^{32} / 2^5 = 2^{27} = \mathbf{128 \text{ M bloques}}$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(32) = 5$



¿Debo multiplicar los 4G por 8? ¿O los 128K por 8? ¿Se multiplica por el tamaño de palabra?

Asignación asociativa (fully associative)

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 32 bytes, tamaño de caché 128 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = 128 KB / 32 = $2^{17} / 2^5 = 2^{12} = \mathbf{4096 \text{ líneas}}$

Cantidad de bloques =

tamaño MP / tamaño de línea = 4 GB / 32 bytes = $2^{32} / 2^5 = 2^{27} = \mathbf{128 \text{ M bloques}}$

Longitud del campo etiqueta =

$\log_2(\text{cantidad de bloques}) = \log_2(128 \text{ M}) = \log_2(2^{27}) = 27$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(32) = 5$



¿Debo multiplicar los 4G por 8? ¿O los 128K por 8? ¿Se multiplica por el tamaño de palabra?

Asignación asociativa (fully associative)

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =

Etiqueta (tag) – 27 bits	Palabra (offset) – 5 bits

En asignación asociativa se comparan todas las etiqueta en paralelo

Asignación asociativa (fully associative)

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
01111101011101110001101100111000 base 2

Etiqueta (tag) – 27 bits	Palabra (offset) – 5 bits

En asignación asociativa se comparan todas las etiqueta en paralelo

Asignación asociativa (fully associative)

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
011111010111011100011011001 11000 base 2

Etiqueta (tag) – 27 bits	Palabra (offset) – 5 bits
011111010111011100011011001	11000

En asignación asociativa se comparan todas las etiqueta en paralelo

Asignación asociativa (fully associative)

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
011111010111011100011011001 11000 base 2

Etiqueta (tag) – 27 bits	Palabra (offset) – 5 bits
011111010111011100011011001	11000

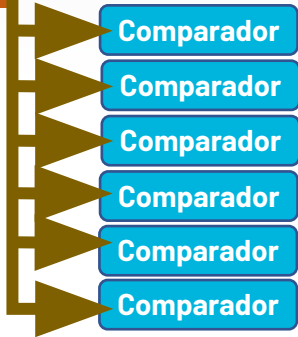
- La etiqueta de la dirección (011111010111011100011011001) se compara en forma simultánea con las etiquetas de todas las líneas de caché. Si existe en MC una línea cuya etiqueta tenga el mismo valor, es un **ACIERTO**. Por lo tanto se accede a la palabra ubicada en el offset 11000 (24_{10}) de la línea.
- Si ninguna de las etiquetas de MC tiene el valor buscado, es una **FALLA**.

En asignación asociativa se comparan todas las etiqueta en paralelo

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Etiqueta (tag)	Offset (palabra)
011111010111011100011011001	11000

Ej: 32 palabras de 1 byte c/u



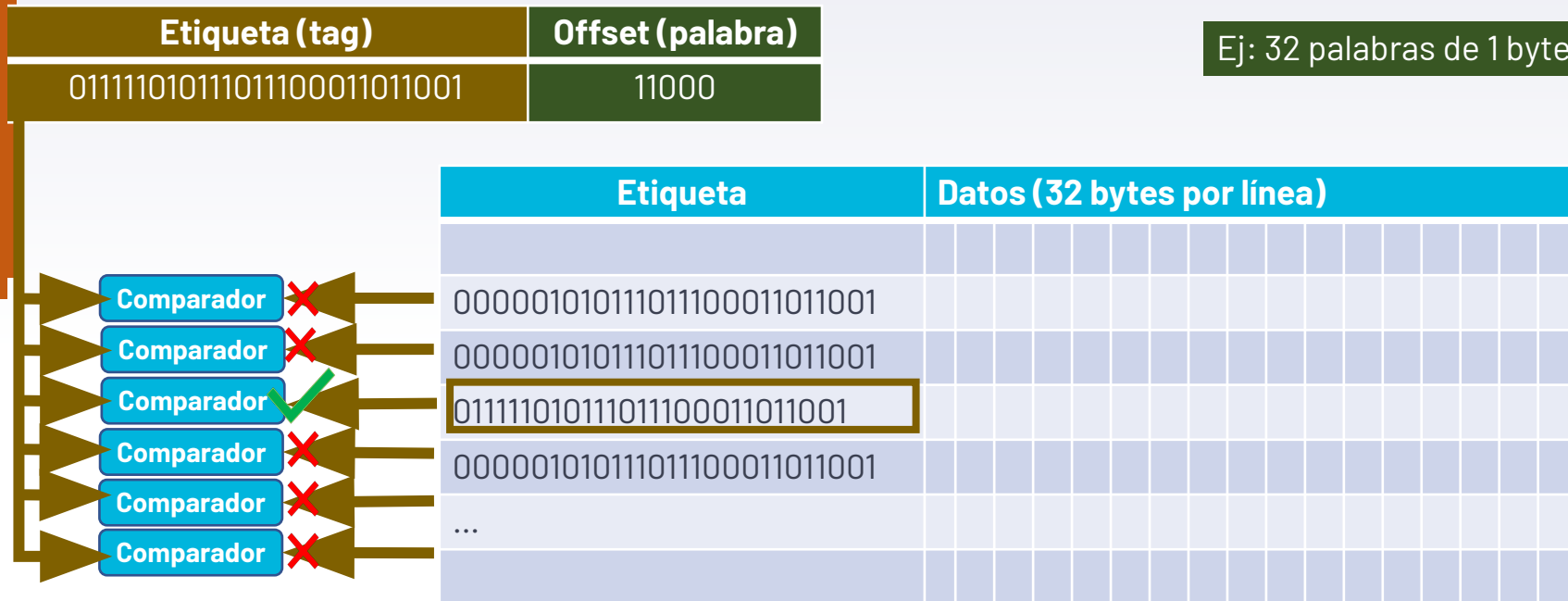
Etiqueta	Datos (32 bytes por línea)																														
000001010111011100011011001																															
000001010111011100011011001																															
011111010111011100011011001																															
000001010111011100011011001																															
...																															

Memoria Caché

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Ej: 32 palabras de 1 byte c/u



Memoria Caché

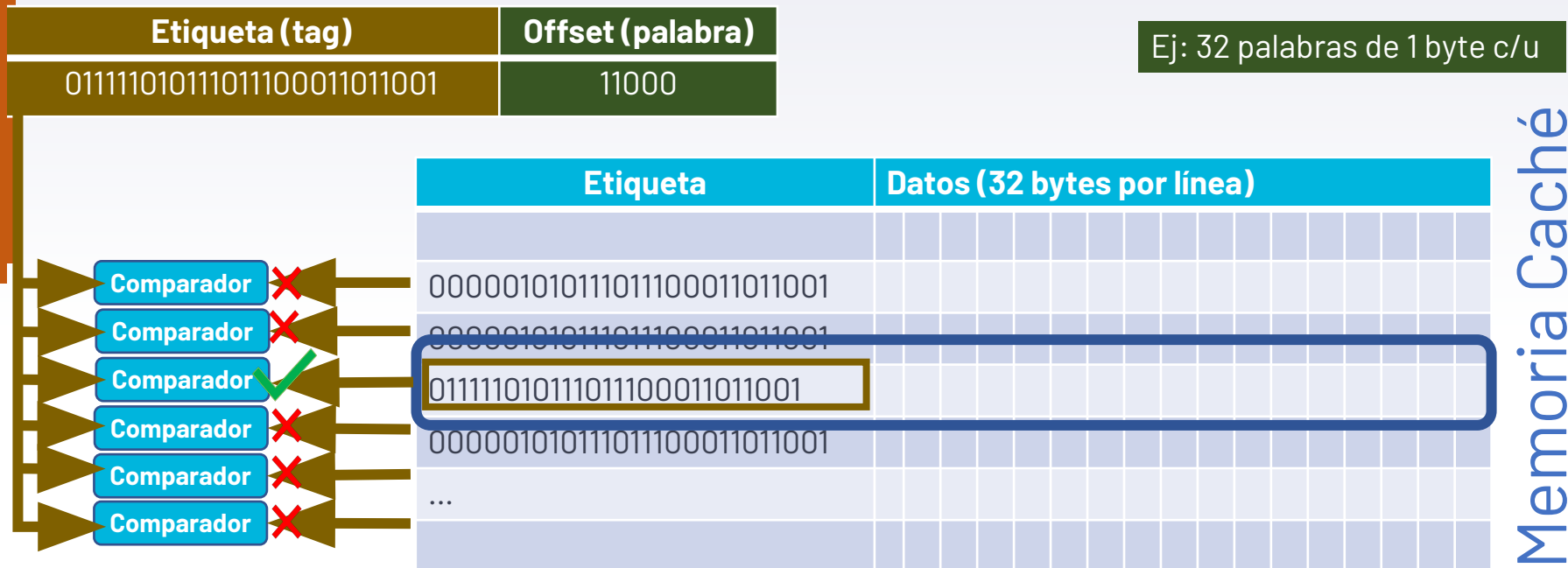
1. Se comparan *simultáneamente* todas las etiquetas de MC

Bus de datos

¿Cómo funciona?

La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

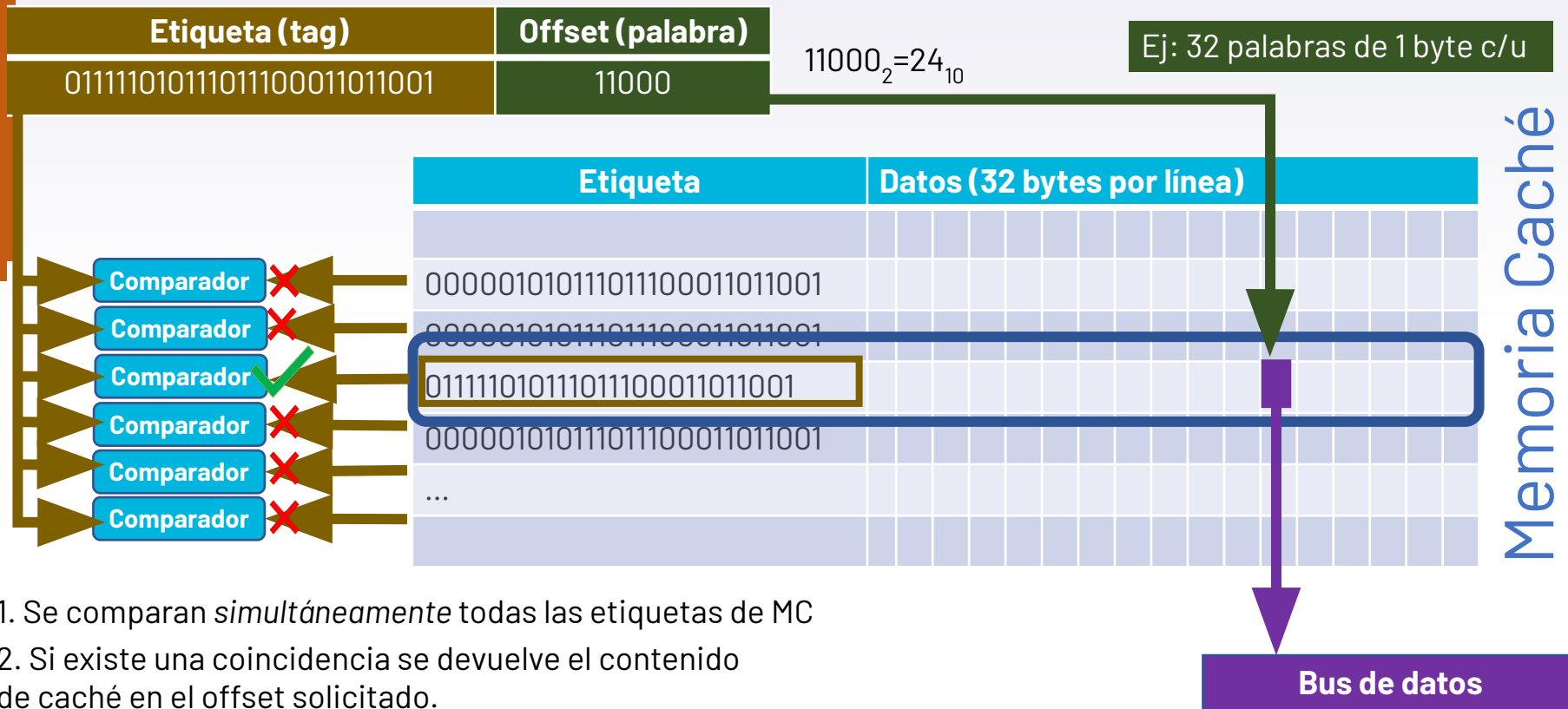
Ej: 32 palabras de 1 byte c/u



1. Se comparan *simultáneamente* todas las etiquetas de MC
2. Si existe una coincidencia se devuelve el contenido de caché en el offset solicitado.

Bus de datos

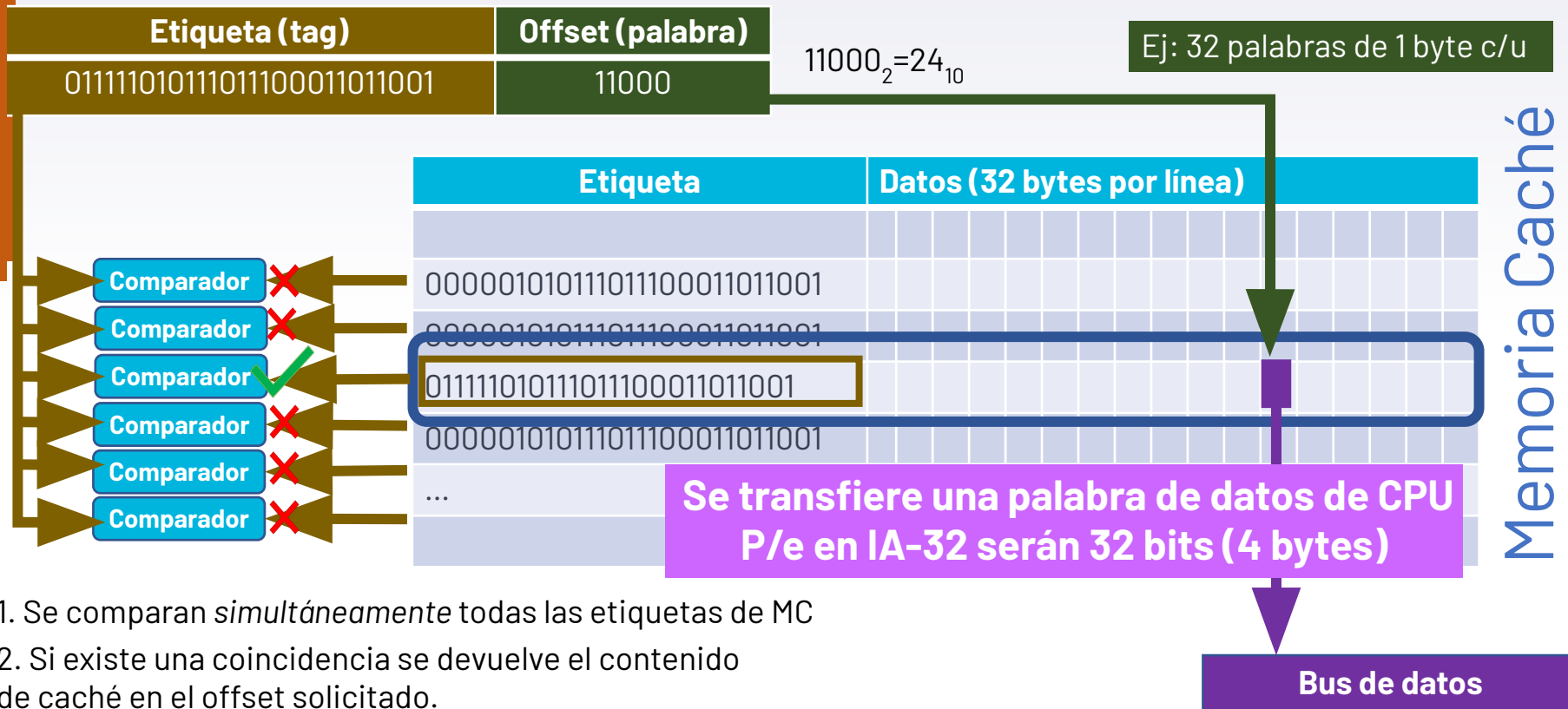
¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)



1. Se comparan *simultáneamente* todas las etiquetas de MC
2. Si existe una coincidencia se devuelve el contenido de caché en el offset solicitado.

¿Cómo funciona?

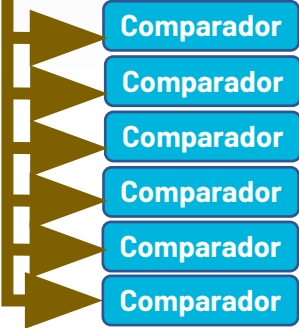
La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)



1. Se comparan *simultáneamente* todas las etiquetas de MC
2. Si existe una coincidencia se devuelve el contenido de caché en el offset solicitado.

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra **NO** está en la memoria caché (**MISS**)

Etiqueta (tag)	Offset (palabra)
011111010111011100011011001	11000

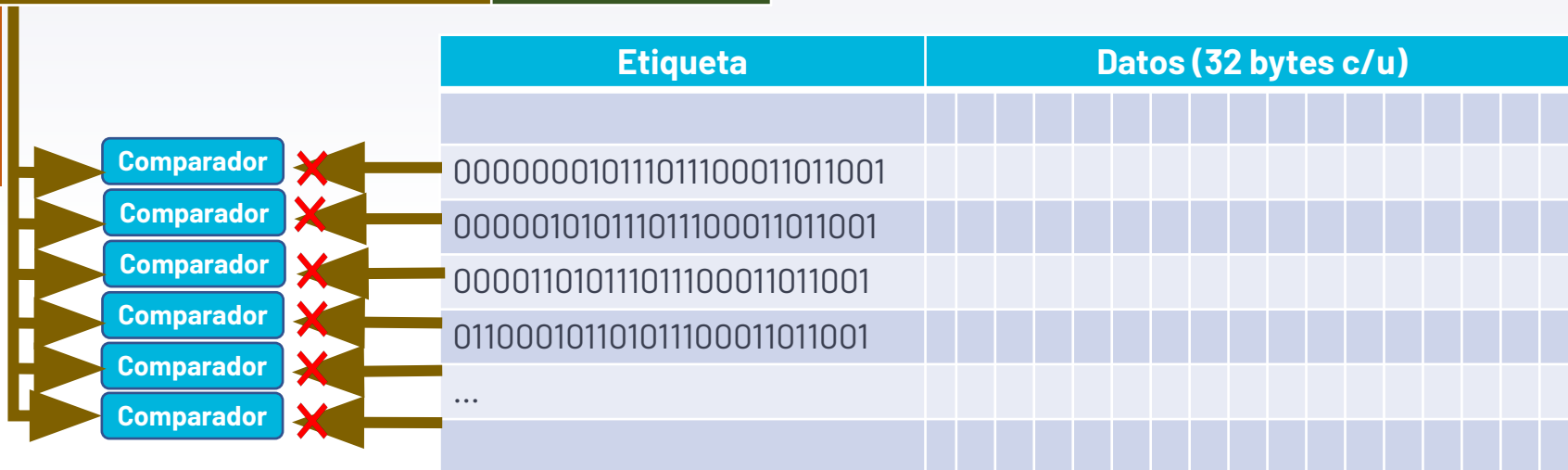


Etiqueta	Datos (32 bytes c/u)															
000000010111011100011011001																
000001010111011100011011001																
000011010111011100011011001																
011000101101011100011011001																
...																

Memoria Caché

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra **NO** está en la memoria caché (**MISS**)

Etiqueta (tag)	Offset (palabra)
011111010111011100011011001	11000



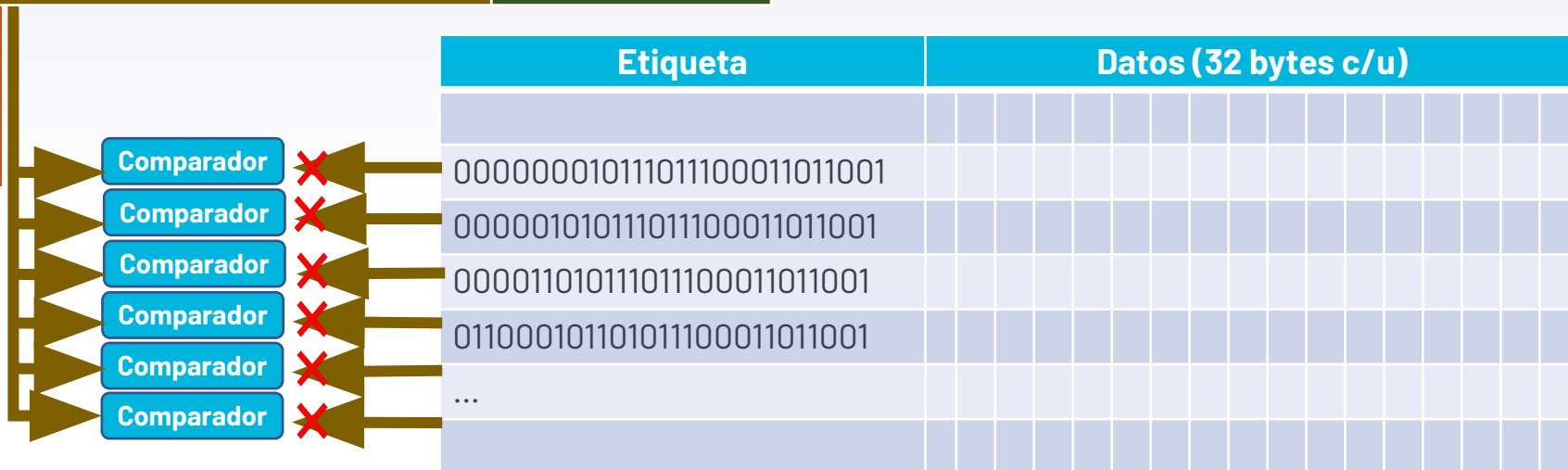
Memoria Caché

1. Se comparan *simultáneamente* todas las etiquetas de MC

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra **NO** está en la memoria caché (**MISS**)

Etiqueta (tag)	Offset (palabra)
011111010111011100011011001	11000

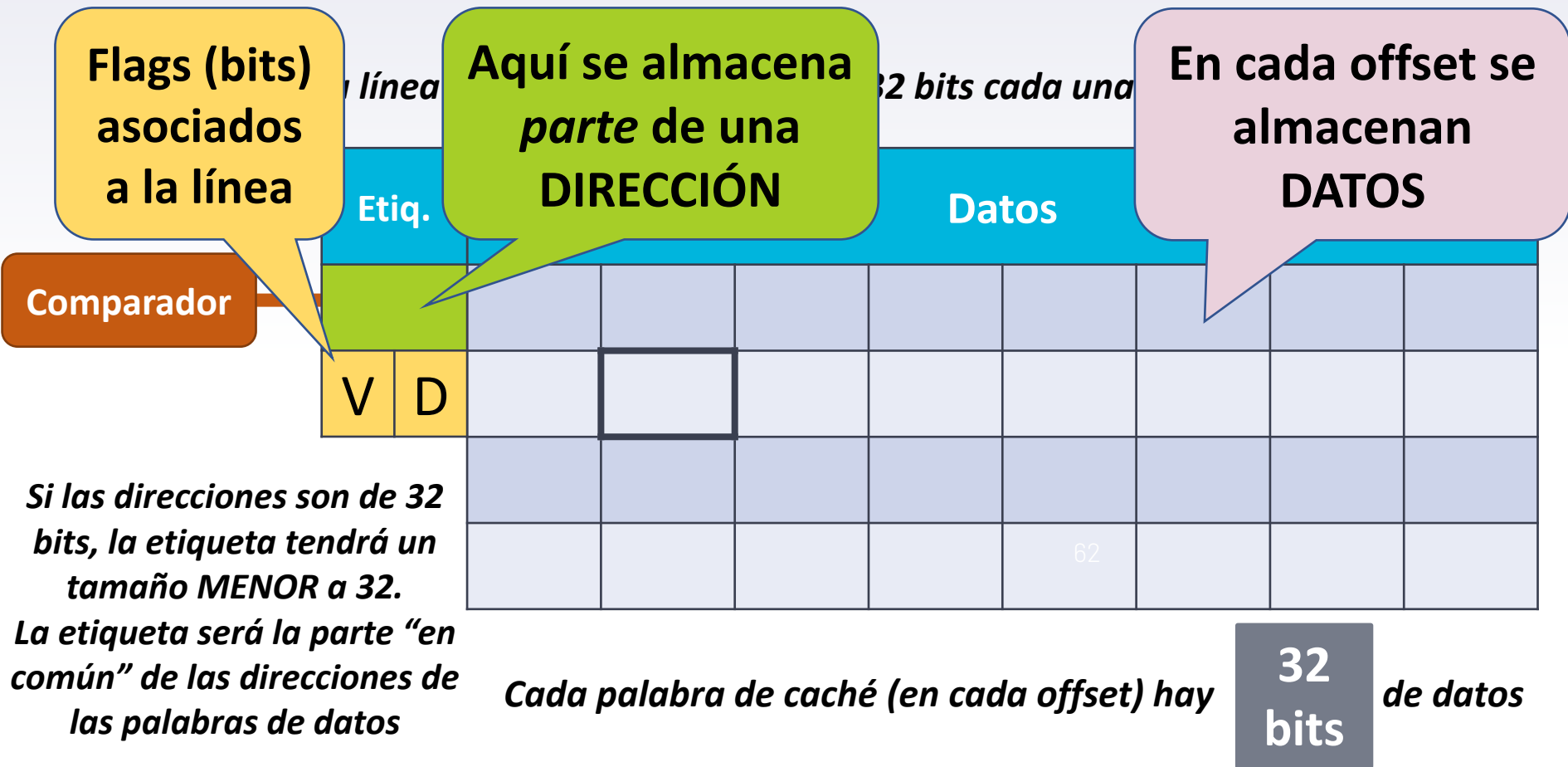


Memoria Caché

1. Se comparan *simultáneamente* todas las etiquetas de MC
2. Si no existe coincidencia se debe acceder a MP.

Bus de datos

Zoom en una línea de memoria caché: más cerca de las dimensiones reales



Organización: Mapeo directo

Cada dirección de memoria principal tiene un lugar específico donde alojarse en caché.

Pero dado que la MC es mucho más pequeña, icada ubicación de MC tiene asignadas muchas porciones de MP!

Dado que **cada bloque de MP solo puede ubicarse en una línea específica de MC**, cuando se produzca una falla siempre se reemplazará el bloque que estaba ocupando el mismo lugar.

Cada bloque de MP (auto) tiene asignada una ubicación en el estacionamiento.



Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea =
cantidad de líneas

Memoria principal

Bloque 0

Bloque 1

Bloque 2

Bloque 3

Bloque 4

Bloque 5

Bloque 6

Bloque 7

Bloque ...

Bloque N

Memoria Caché

Línea 0

Línea 1

Línea 2

Línea 3

Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

*Cada línea de caché normalmente
contiene múltiples palabras.*

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea =
cantidad de líneas

Memoria principal
Bloque 0
Bloque 1
Bloque 2
Bloque 3
Bloque 4
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

Memoria Caché
Línea 0
Línea 1
Línea 2
Línea 3

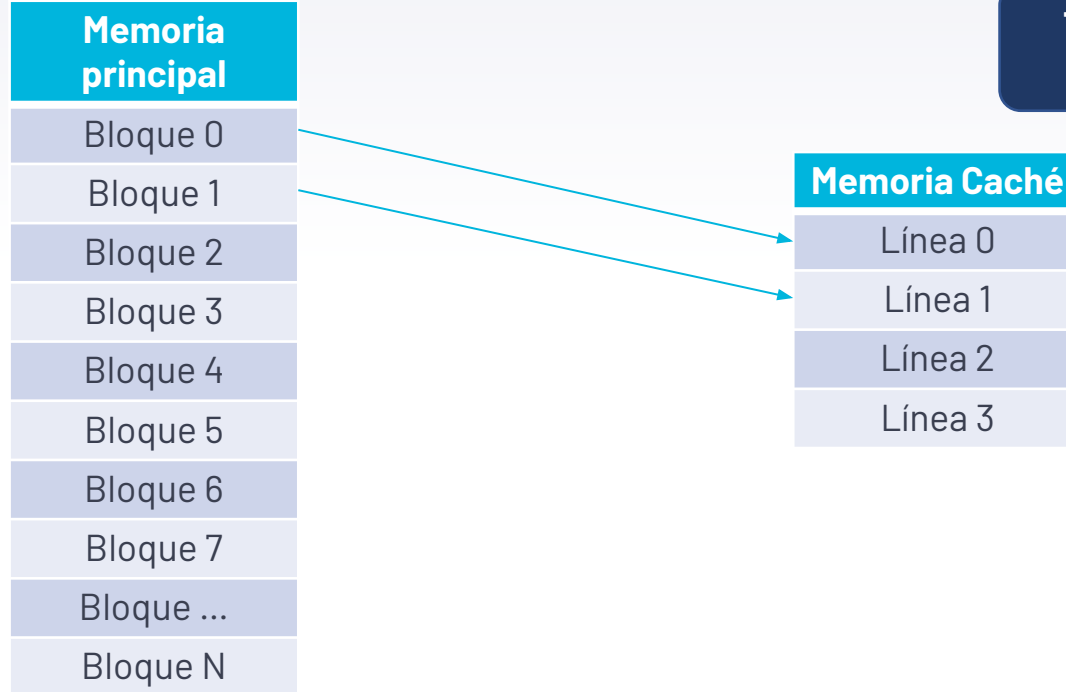
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

Cada línea de caché normalmente
contiene múltiples palabras.

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea =
cantidad de líneas



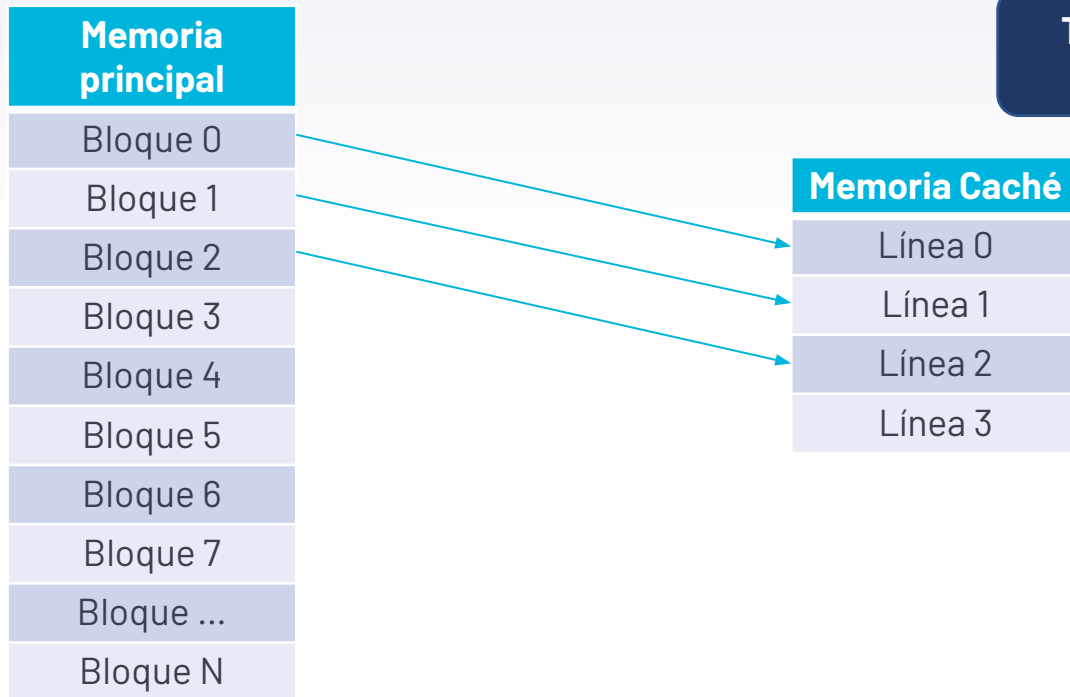
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

*Cada línea de caché normalmente
contiene múltiples palabras.*

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea = cantidad de líneas



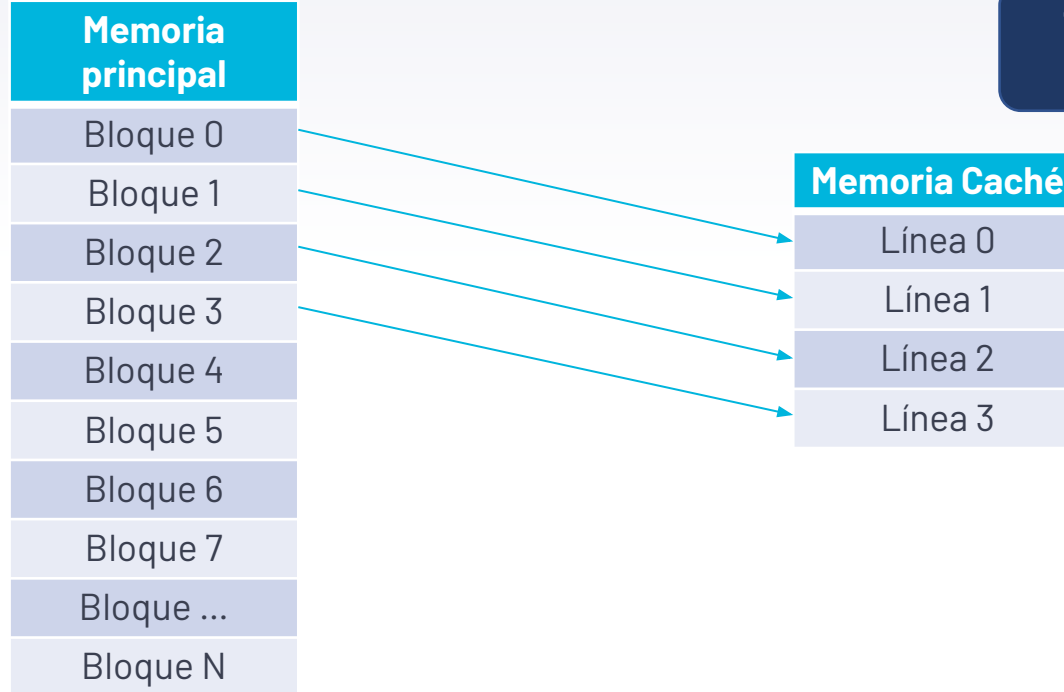
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

Cada línea de caché normalmente contiene múltiples palabras.

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea =
cantidad de líneas



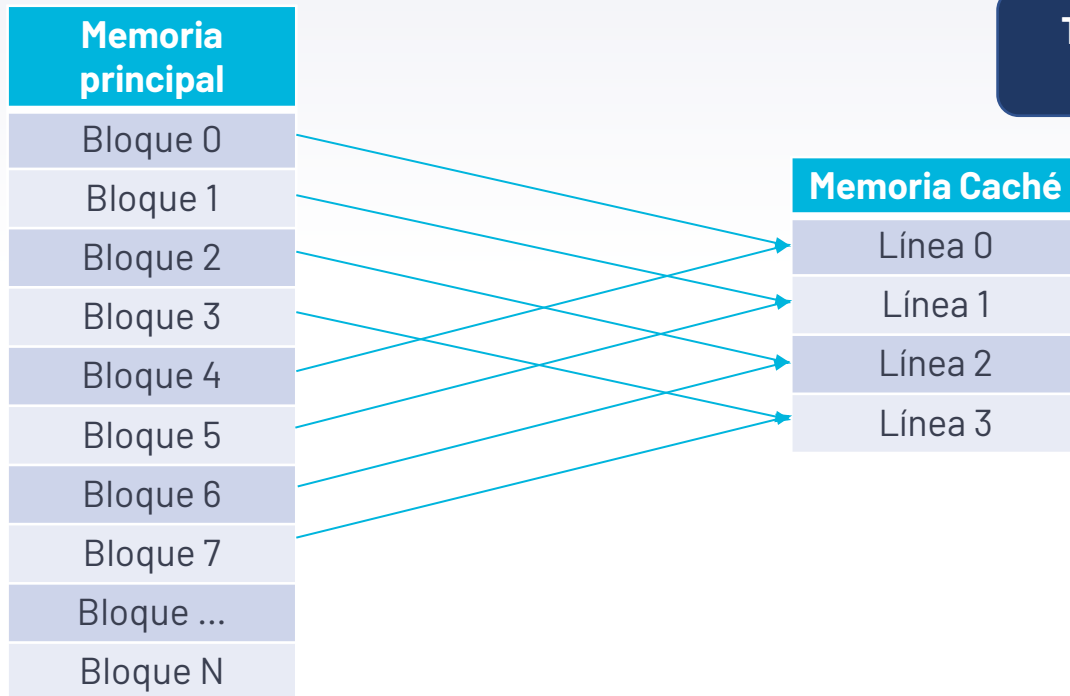
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

*Cada línea de caché normalmente
contiene múltiples palabras.*

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea = cantidad de líneas



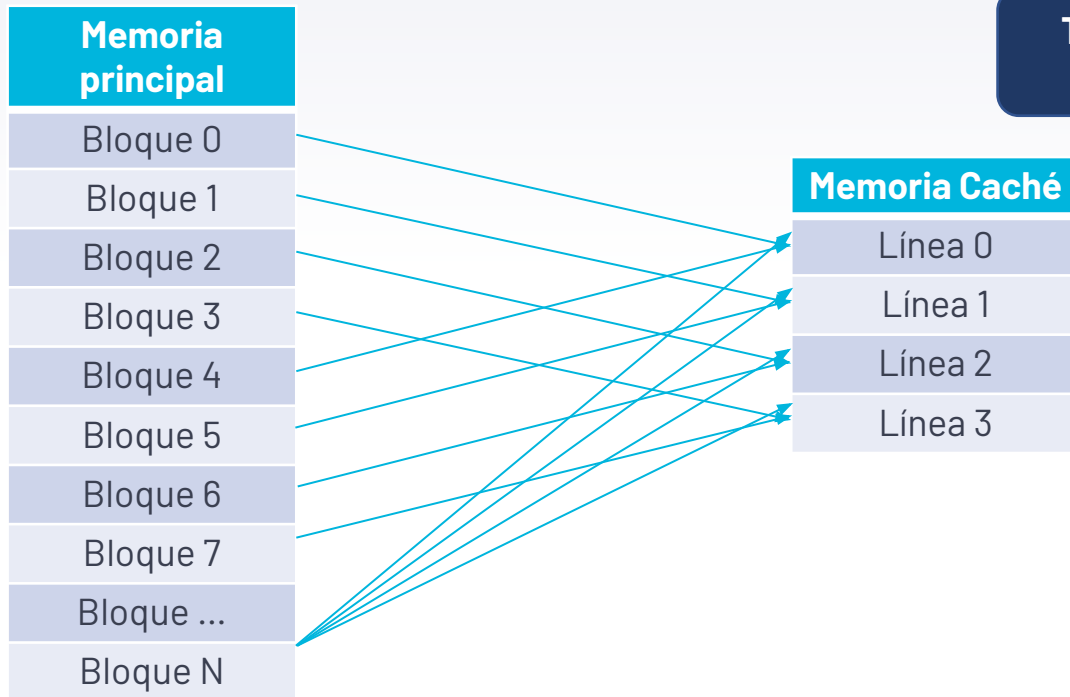
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

Cada línea de caché normalmente contiene múltiples palabras.

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea = cantidad de líneas



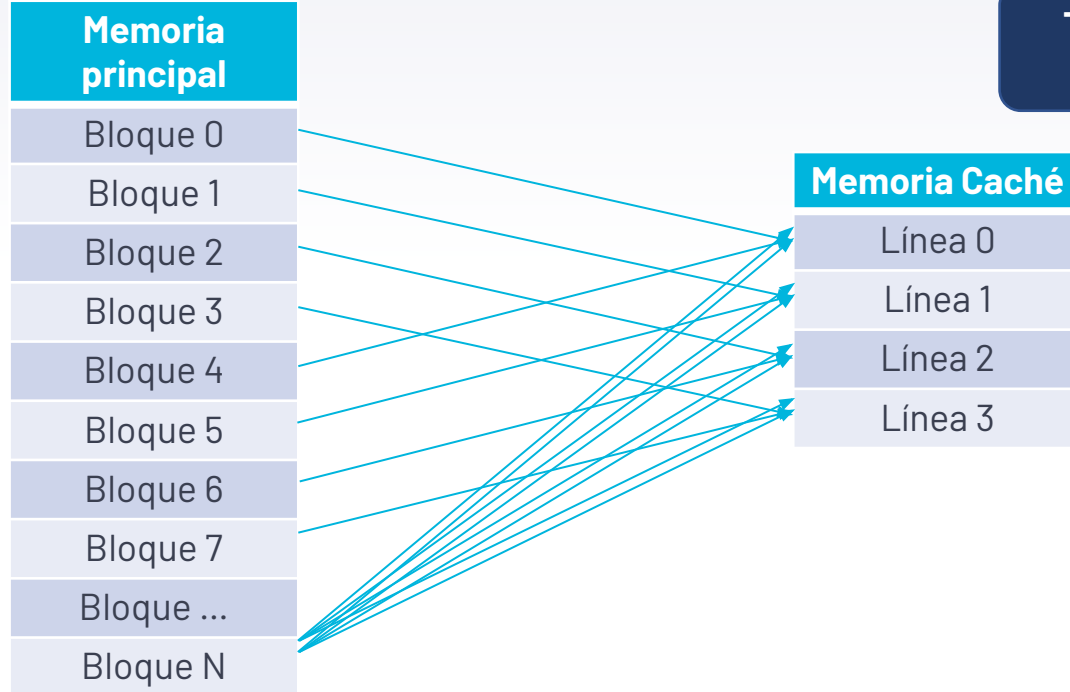
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

Cada línea de caché normalmente contiene múltiples palabras.

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea = cantidad de líneas



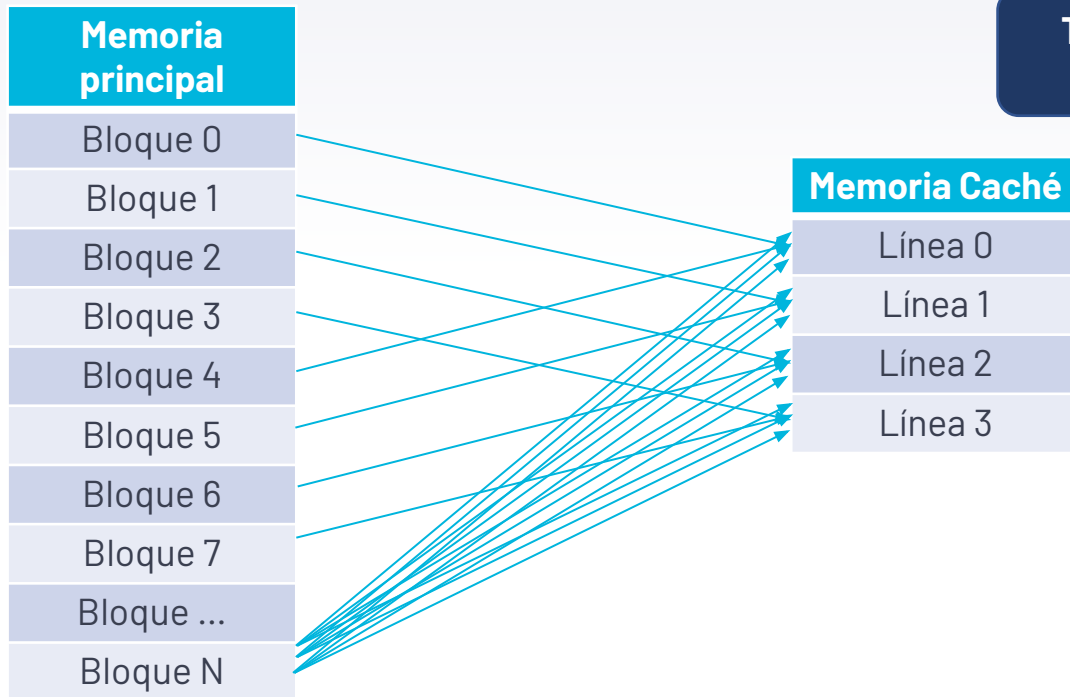
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

Cada línea de caché normalmente contiene múltiples palabras.

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea =
cantidad de líneas



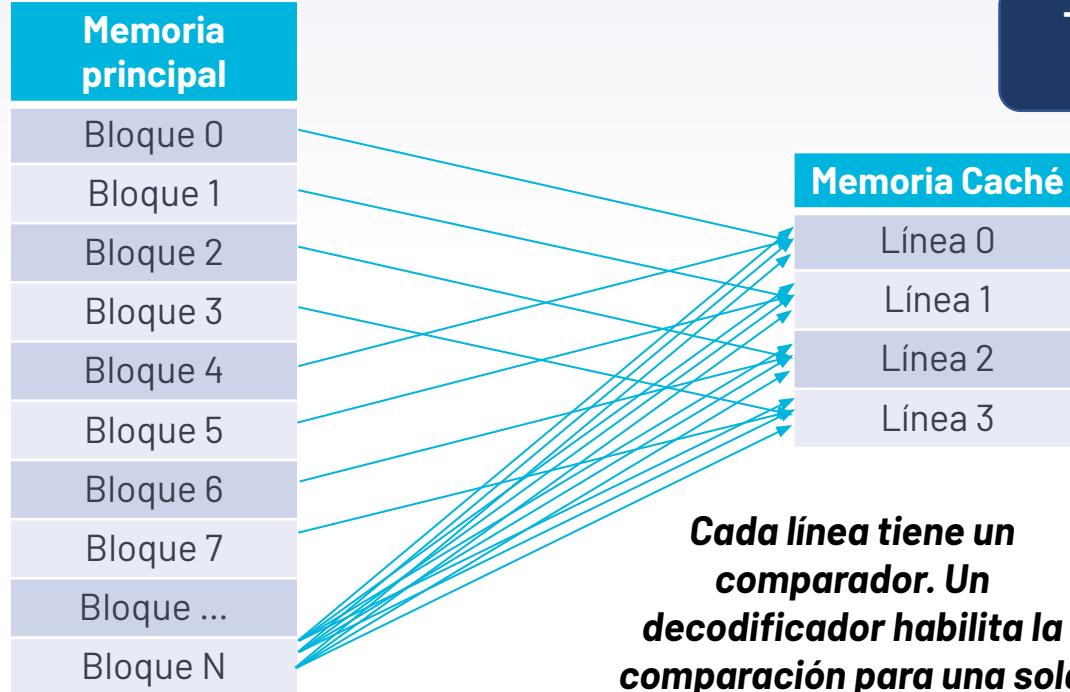
Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

*Cada línea de caché normalmente
contiene múltiples palabras.*

Mapeo directo

Una misma línea de caché puede alojar distintos bloques de MP. **Pero cada bloque de MP tiene asignado solamente una línea en la MC.**

Tamaño de la MC / tamaño de línea =
cantidad de líneas



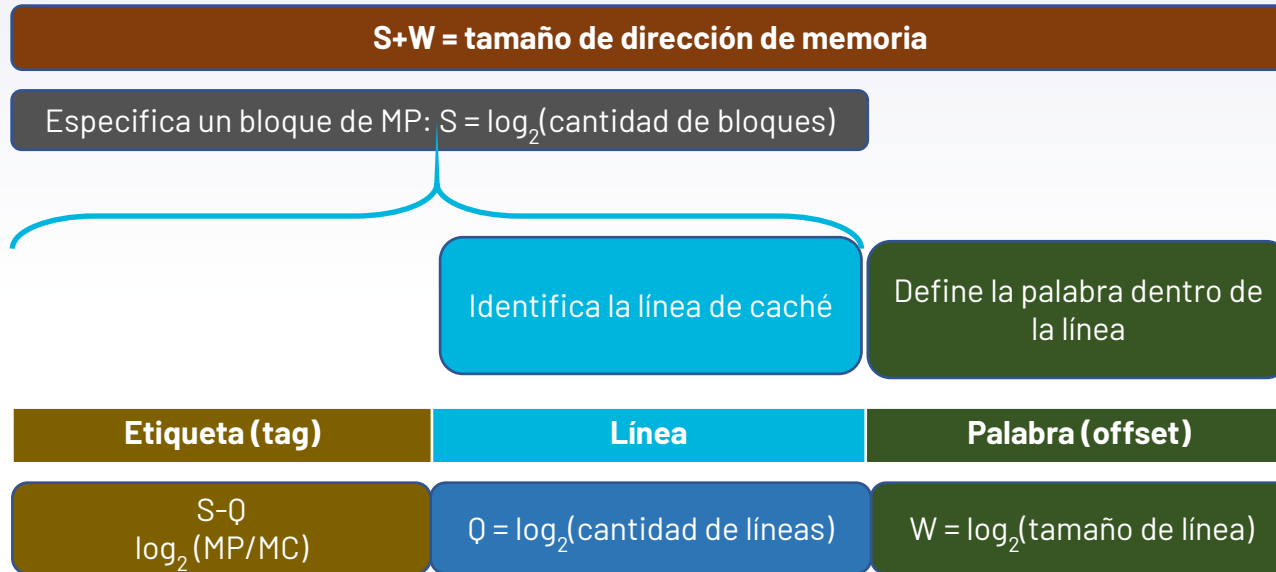
Cada línea tiene un comparador. Un decodificador habilita la comparación para una sola línea a la vez.

Línea MC	Bloque de MP que alojará
0	0, 4, 8, ...
1	1, 5, 9, ...
2	2, 6, 10, ...
3	3, 7, 11, ...

Cada línea de caché normalmente contiene múltiples palabras.

Mapeo directo

Interpretación de cada dirección de memoria según la CPU la coloca en el bus.



Para determinar si una palabra se encuentra en MC se ubica la línea de caché (siempre corresponderá una sola para cada bloque de memoria) y luego se compara la etiqueta.

La comparación se efectúa solamente sobre el campo de etiqueta.

Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Etiqueta (tag)

Línea

Palabra (offset)

Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas = tamaño de MC / tamaño de línea =



Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas = tamaño de MC / tamaño de línea = 32 KB / 64 = **512**



Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas = tamaño de MC / tamaño de línea = 32 KB / 64 = **512**

Longitud del campo offset =
 $\log_2(\text{tamaño de línea}) = \log_2(64) = 6$



Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas = tamaño de MC / tamaño de línea = 32 KB / 64 = **512**

Longitud del campo offset =
 $\log_2(\text{tamaño de línea}) = \log_2(64) = 6$



Longitud del campo línea = $\log_2(\text{cantidad de líneas}) = \log_2(512) = 9$

Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas = tamaño de MC / tamaño de línea = 32 KB / 64 = **512**

Cantidad de bloques =

tamaño MP / tamaño de línea = 4 GB / 64 bytes = $2^{32}/2^6=2^{26}=64$ M bloques

Longitud del campo offset =

$\log_2(\text{tamaño de línea})=\log_2(64)=6$



Longitud del campo línea= $\log_2(\text{cantidad de líneas})=\log_2(512)=9$

Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas = tamaño de MC / tamaño de línea = 32 KB / 64 = **512**

Cantidad de bloques =

tamaño MP / tamaño de línea = 4 GB / 64 bytes = $2^{32}/2^6=2^{26}=64$ M bloques

Longitud del campo etiqueta =

$\log_2(\text{cantidad de bloques}) - \log_2(\text{cantidad de líneas}) = \log_2(2^{26}) - \log_2(512) = 26 - 9 = 17$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(64) = 6$

Etiqueta (tag)	Línea	Palabra (offset)
17 bits	9 bits	6 bits

Longitud del campo línea = $\log_2(\text{cantidad de líneas}) = \log_2(512) = 9$

Mapeo directo

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas = tamaño de MC / tamaño de línea = 32 KB / 64 = **512**

Cantidad de bloques =

tamaño MP / tamaño de línea = 4 GB / 64 bytes = $2^{32}/2^6=2^{26}=64$ M bloques

Longitud del campo etiqueta =

$\log_2(\text{cantidad de bloques}) - \log_2(\text{cantidad de líneas}) = \log_2(2^{26}) - \log_2(512) = 26 - 9 = 17$

Longitud del campo etiqueta (bis) =

$\log_2(\text{tamaño MP} / \text{tamaño MC}) = \log_2(4 \text{ GB} / 32 \text{ KB}) = \log_2(2^{32}/2^{15}) = \log_2(2^{17}) = 17$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(64) = 6$

Etiqueta (tag)	Línea	Palabra (offset)
17 bits	9 bits	6 bits

Longitud del campo línea = $\log_2(\text{cantidad de líneas}) = \log_2(512) = 9$

Mapeo directo

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =



En asignación directa se compara sólo la etiqueta que corresponde a la línea en que debería encontrarse el bloque buscado.

Mapeo directo

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
01111101011101110001101100111000 base 2

Etiqueta (tag) – 17 bits	Línea – 9 bits	Palabra (offset) – 6 bits

En asignación directa se compara sólo la etiqueta que corresponde a la línea en que debería encontrarse el bloque buscado.

Mapeo directo

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
01111101011101110 001101100 111000 base 2

Etiqueta (tag) – 17 bits	Línea – 9 bits	Palabra (offset) – 6 bits
01111101011101110	001101100	111000

En asignación directa se compara sólo la etiqueta que corresponde a la línea en que debería encontrarse el bloque buscado.

Mapeo directo

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
01111101011101110 001101100 111000 base 2

Etiqueta (tag) – 17 bits	Línea – 9 bits	Palabra (offset) – 6 bits
01111101011101110	001101100	111000

- La porción de línea de la dirección ($001101100 = 108_{10}$) determina que se verificará la línea 108_{10} de caché. Si en esa línea el campo etiqueta tiene el valor 01111101011101110, es un **ACIERTO**. Por lo tanto se accede a la palabra ubicada en el offset 111000 (56_{10}) de la línea.
- Si en la línea 108 el campo etiqueta tiene un valor distinto, es una **FALLA**.

En asignación directa se compara sólo la etiqueta que corresponde a la línea en que debería encontrarse el bloque buscado.

¿Cómo funciona?

La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (HIT)

Etiqueta (tag)	Línea	Palabra (offset)
01111101011101110	001101100	111000

Comparador

Línea	Etiqueta	Datos (64 bytes c/u)
106	011110000011110000	
107	01111101011101110	
108	01111101011101110	
109	11001101011101010	

Memoria Caché

Bus de datos

¿Cómo funciona?

La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Bus de direcciones

Etiqueta (tag)	Línea	Palabra (offset)
01111101011101110	001101100	111000



Línea	Etiqueta	Datos (64 bytes c/u)
106	01111000011110000	
107	01111101011101110	
108	01111101011101110	
109	11001101011101010	

Memoria Caché

Comparador

1. Se selecciona la línea 108 (001101100)

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Bus de direcciones

Etiqueta (tag)	Línea	Palabra (offset)
01111101011101110	001101100	111000

Comparador

Línea	Etiqueta	Datos (64 bytes c/u)
106	01111000011110000	
107	01111101011101110	
108	01111101011101110	
109	11001101011101010	

Memoria Caché

1. Se selecciona la línea 108 (001101100)
2. Se compara la etiqueta en MC con la porción de etiqueta de la dirección

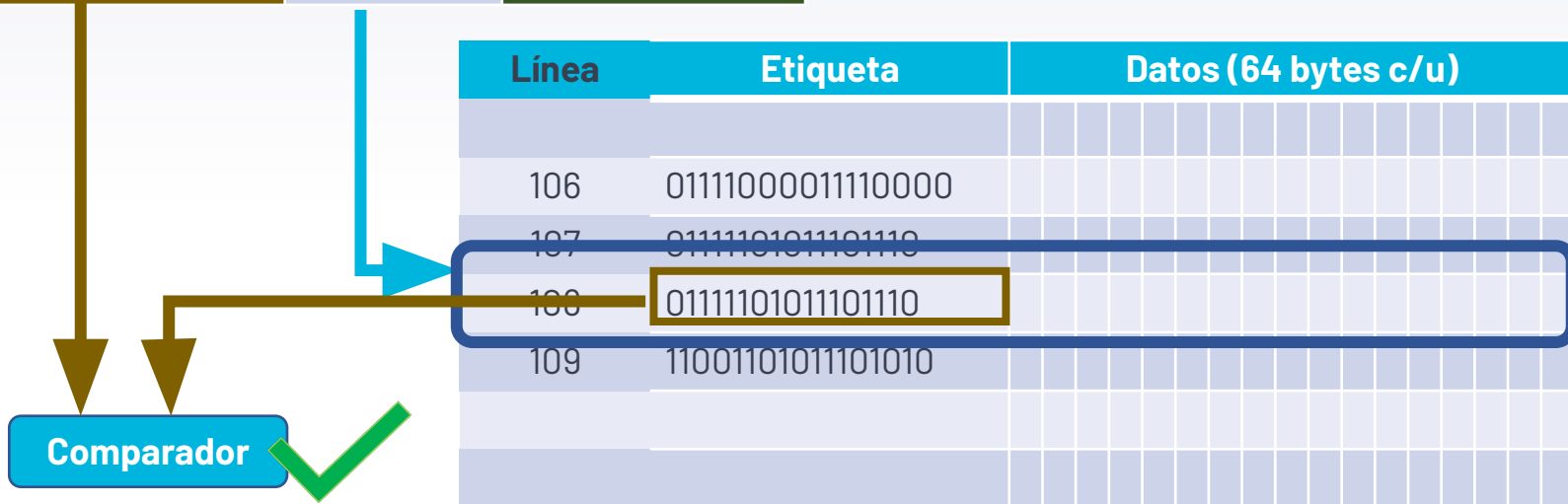
Bus de datos

¿Cómo funciona?

La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Bus de direcciones

Etiqueta (tag)	Línea	Palabra (offset)
01111101011101110	001101100	111000



Memoria Caché

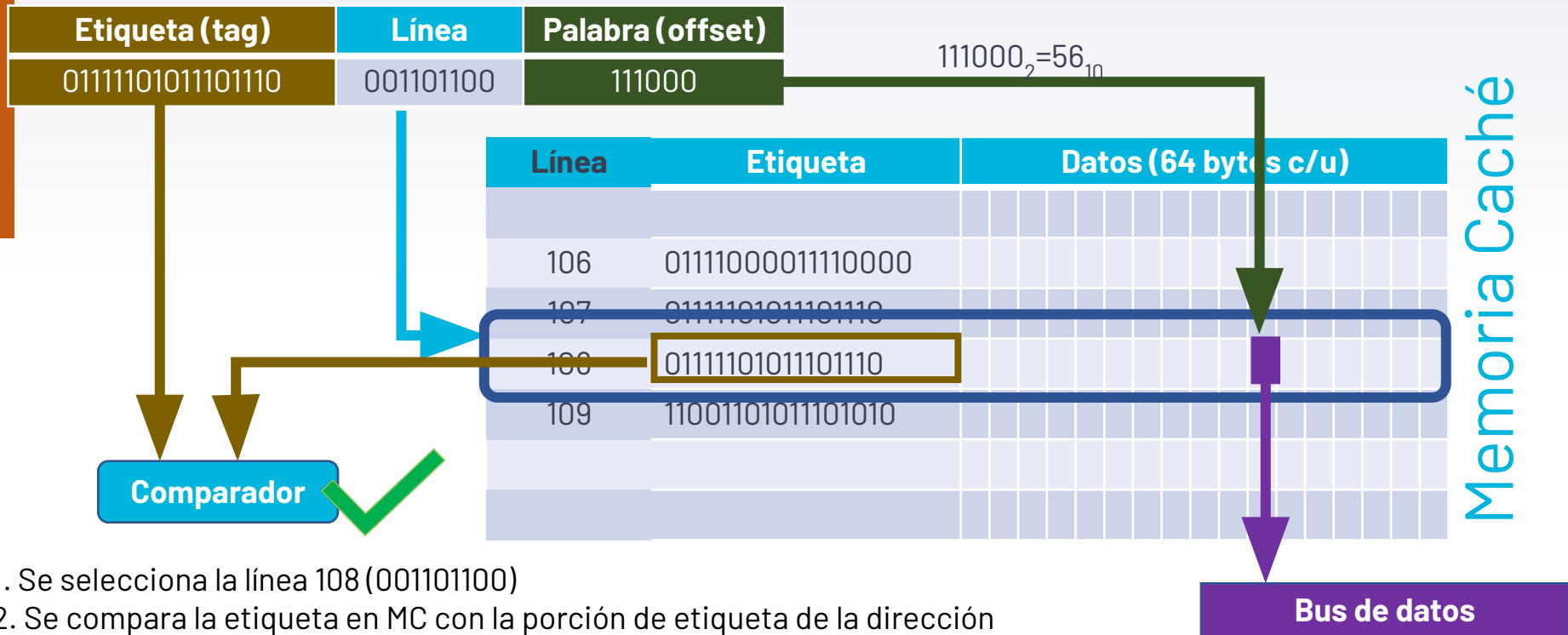
1. Se selecciona la línea 108 (001101100)
2. Se compara la etiqueta en MC con la porción de etiqueta de la dirección

Bus de datos

¿Cómo funciona?

La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (HIT)

Bus de direcciones



1. Se selecciona la línea 108 (001101100)
2. Se compara la etiqueta en MC con la porción de etiqueta de la dirección
3. Si son iguales, se devuelve el contenido de caché en el offset solicitado

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra **NO** está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Línea	Palabra (offset)
01111101011101110	001101100	111000

Línea	Etiqueta	Datos (64 bytes c/u)
106	011110000011110000	
107	01111101011101110	
108	010000000011111110	
109	11001101011101010	

Comparador

Memoria Caché

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra **NO** está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Línea	Palabra (offset)
01111101011101110	001101100	111000



Línea	Etiqueta	Datos (64 bytes c/u)
106	01111000011110000	
107	01111101011101110	
108	01000000011111110	
109	11001101011101010	

Memoria Caché

Comparador

1. Se selecciona la línea 108 (001101100)

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra **NO** está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Línea	Palabra (offset)
0111101011101110	001101100	111000



Línea	Etiqueta	Datos (64 bytes c/u)
106	01111000011110000	
107	0111101011101110	
108	01000000011111110	
109	11001101011101010	

Memoria Caché

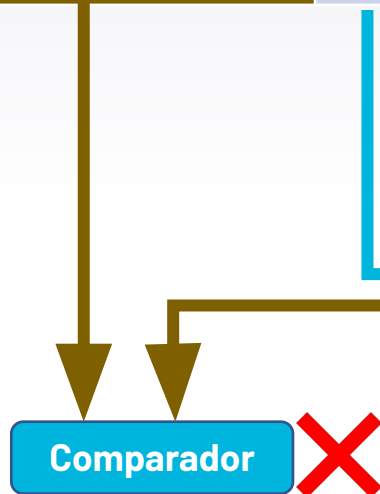
1. Se selecciona la línea 108 (001101100)
2. Se compara la etiqueta en MC con la porción de etiqueta de la dirección

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra **NO** está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Línea	Palabra (offset)
01111101011101110	001101100	111000



Línea	Etiqueta	Datos (64 bytes c/u)
106	01111000011110000	
107	01111101011101110	
108	01000000011111110	
109	11001101011101010	

Memoria Caché

1. Se selecciona la línea 108 (001101100)
2. Se compara la etiqueta en MC con la porción de etiqueta de la dirección
3. Si son distintas, se produce una falla: debe acceder a MP

Bus de datos

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.

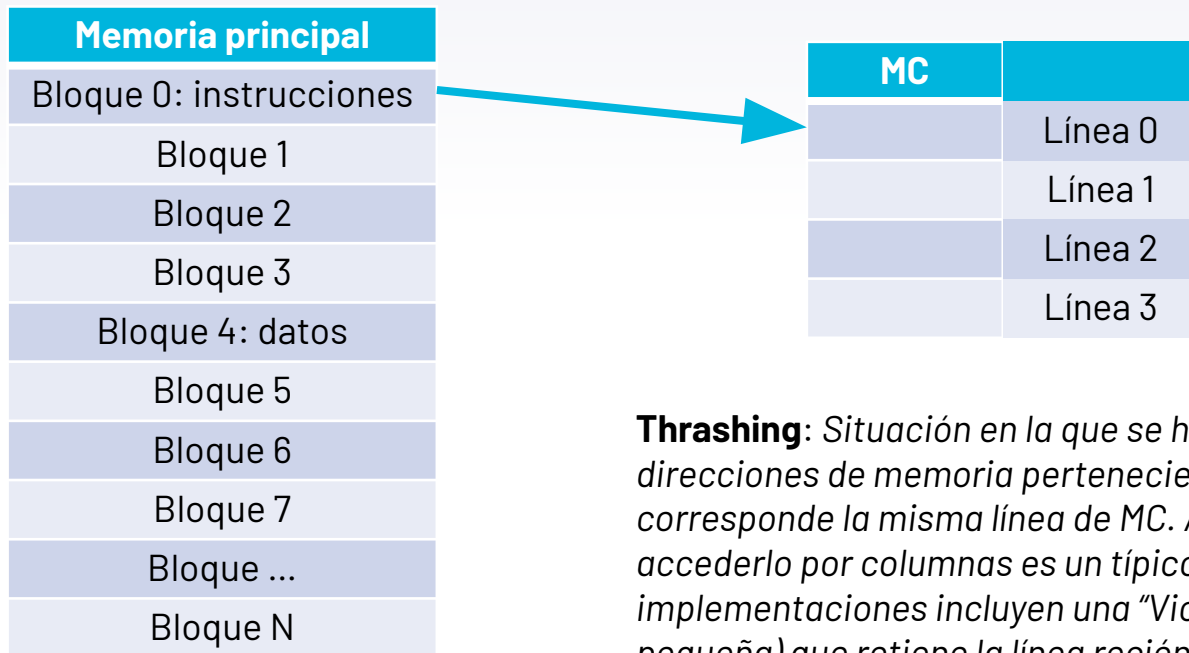
Memoria principal
Bloque 0: instrucciones
Bloque 1
Bloque 2
Bloque 3
Bloque 4: datos
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

MC	
	Línea 0
	Línea 1
	Línea 2
	Línea 3

Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.



Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.

Memoria principal
Bloque 0: instrucciones
Bloque 1
Bloque 2
Bloque 3
Bloque 4: datos
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N



MC	
Bloque 0	Línea 0
	Línea 1
	Línea 2
	Línea 3

Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.

Memoria principal
Bloque 0: instrucciones
Bloque 1
Bloque 2
Bloque 3
Bloque 4: datos
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

MC	
Bloque 0	Línea 0
	Línea 1
	Línea 2
	Línea 3

Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.

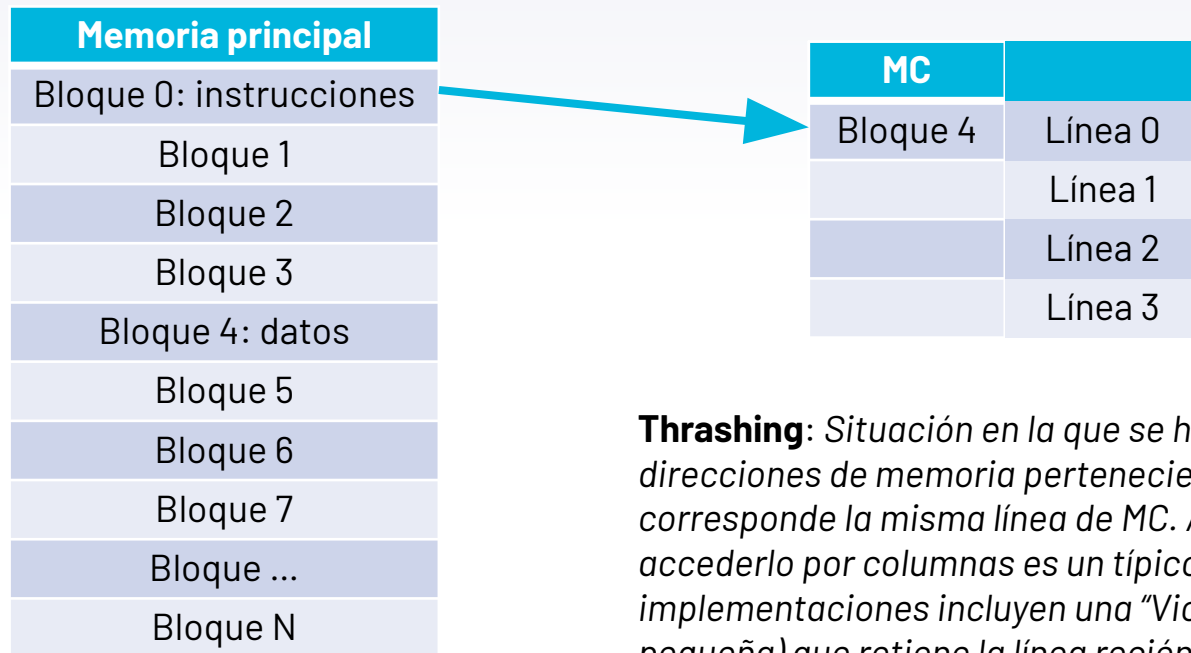
Memoria principal
Bloque 0: instrucciones
Bloque 1
Bloque 2
Bloque 3
Bloque 4: datos
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

MC	
Bloque 4	Línea 0
	Línea 1
	Línea 2
	Línea 3

Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.



Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.

Memoria principal
Bloque 0: instrucciones
Bloque 1
Bloque 2
Bloque 3
Bloque 4: datos
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N



MC	
Bloque 0	Línea 0
	Línea 1
	Línea 2
	Línea 3

Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.

Memoria principal
Bloque 0: instrucciones
Bloque 1
Bloque 2
Bloque 3
Bloque 4: datos
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

MC	
Bloque 0	Línea 0
	Línea 1
	Línea 2
	Línea 3



Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Mapeo directo

Si se acceden repetidamente dos ubicaciones de MP mapeadas a la misma línea de caché se producen muchas fallas (MISS), esta condición se denomina **thrashing**.

Memoria principal
Bloque 0: instrucciones
Bloque 1
Bloque 2
Bloque 3
Bloque 4: datos
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

MC	
Bloque 4	Línea 0
	Línea 1
	Línea 2
	Línea 3

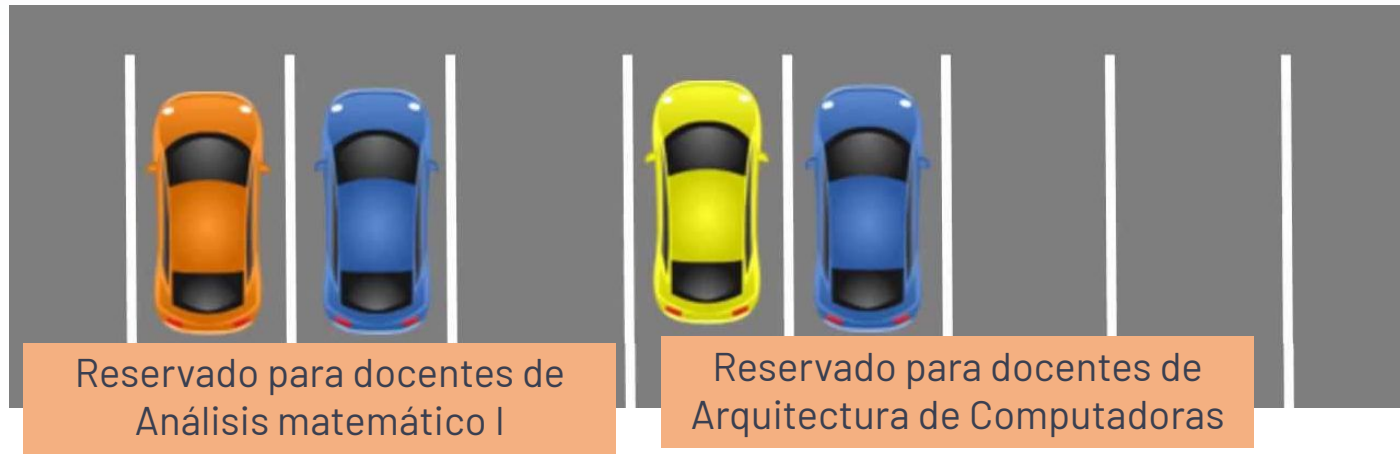
Thrashing: Situación en la que se hacen **referencias sucesivas** a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Asignación asociativa por conjuntos

Cada bloque de memoria principal puede ubicarse en cualquier línea de caché dentro de un conjunto acotado (normalmente de 2, 4, 8, o 16 líneas o **vías**).

La búsqueda de etiquetas se realiza en paralelo pero solo para las líneas del conjunto.

Cada bloque de MP (auto) puede utilizar cualquier ubicación en el estacionamiento dentro del conjunto.



Asignación asociativa por conjuntos

Un bloque de memoria principal puede ir a parar a cualquier línea de caché dentro de un conjunto determinado.

Tamaño de la MC / tamaño de línea = cantidad de líneas

Memoria principal
Bloque 0
Bloque 1
Bloque 2
Bloque 3
Bloque 4
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

Memoria Caché	
Línea 0	Conjunto 0
Línea 1	
Línea 2	Conjunto 1
Línea 3	

Línea MC	Bloque de MP que alojará
0	0, 2, 4, 8....
1	
2	1, 3, 5, 7...
3	

Asignación asociativa por conjuntos

Un bloque de memoria principal puede ir a parar a cualquier línea de caché dentro de un conjunto determinado.

Tamaño de la MC / tamaño de línea = cantidad de líneas

Memoria principal
Bloque 0
Bloque 1
Bloque 2
Bloque 3
Bloque 4
Bloque 5
Bloque 6
Bloque 7
Bloque ...
Bloque N

Memoria Caché	
Línea 0	Conjunto 0
Línea 1	
Línea 2	Conjunto 1
Línea 3	

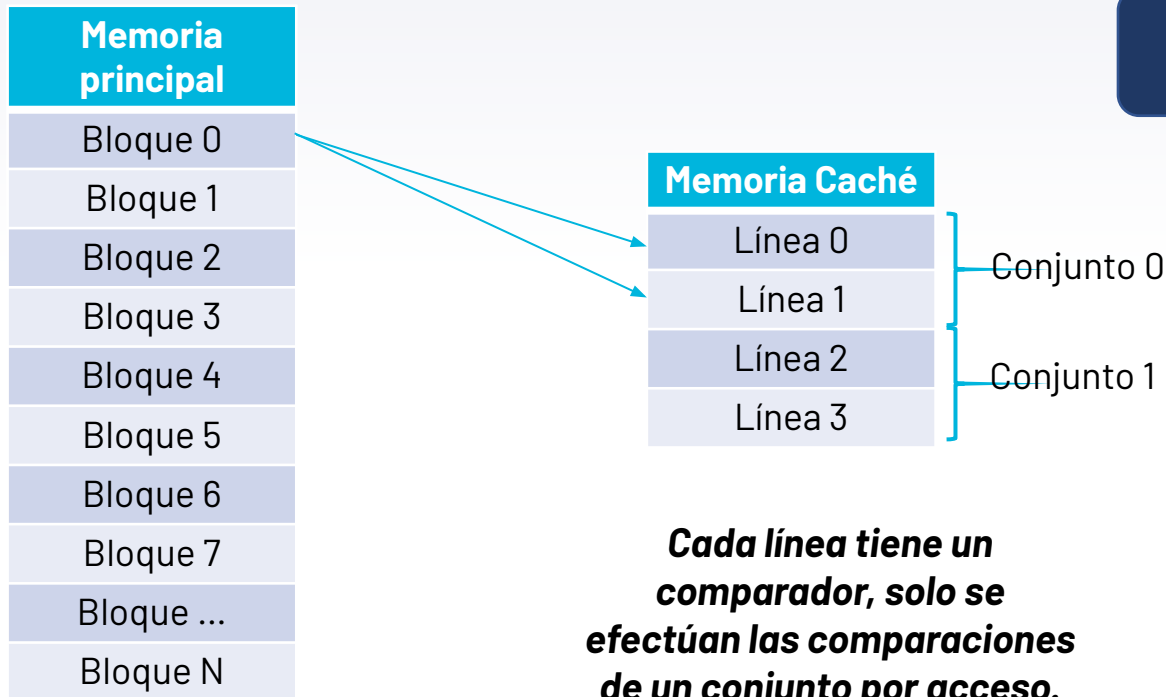
Cada línea tiene un comparador, solo se efectúan las comparaciones de un conjunto por acceso.

Línea MC	Bloque de MP que alojará
0	0, 2, 4, 8....
1	
2	1, 3, 5, 7...
3	

Asignación asociativa por conjuntos

Un bloque de memoria principal puede ir a parar a cualquier línea de caché dentro de un conjunto determinado.

Tamaño de la MC / tamaño de línea = cantidad de líneas



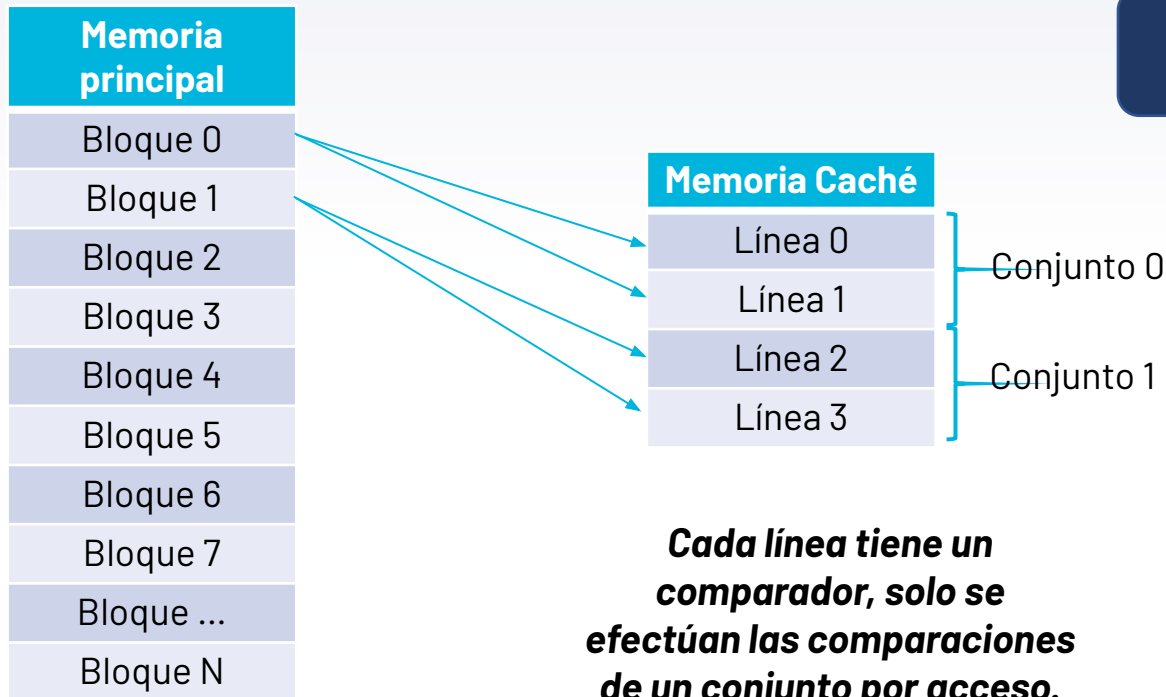
Cada línea tiene un comparador, solo se efectúan las comparaciones de un conjunto por acceso.

Línea MC	Bloque de MP que alojará
0	0, 2, 4, 8....
1	
2	1, 3, 5, 7...
3	

Asignación asociativa por conjuntos

Un bloque de memoria principal puede ir a parar a cualquier línea de caché dentro de un conjunto determinado.

Tamaño de la MC / tamaño de línea = cantidad de líneas



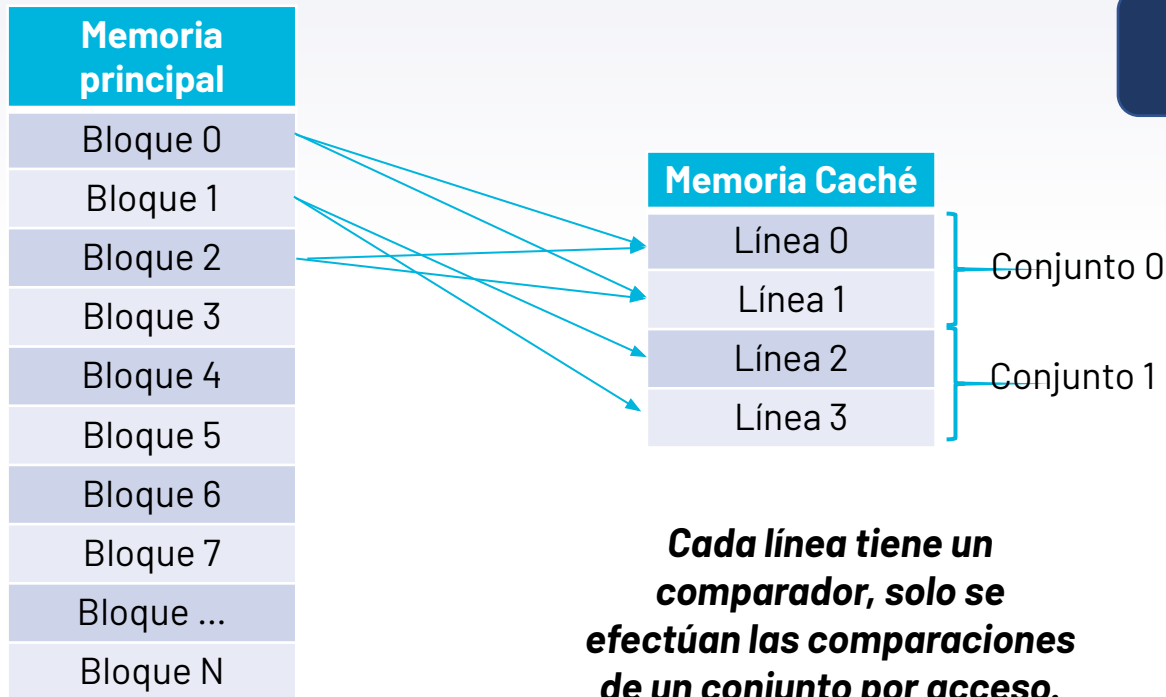
Cada línea tiene un comparador, solo se efectúan las comparaciones de un conjunto por acceso.

Línea MC	Bloque de MP que alojará
0	0, 2, 4, 8....
1	
2	1, 3, 5, 7...
3	

Asignación asociativa por conjuntos

Un bloque de memoria principal puede ir a parar a cualquier línea de caché dentro de un conjunto determinado.

Tamaño de la MC / tamaño de línea = cantidad de líneas



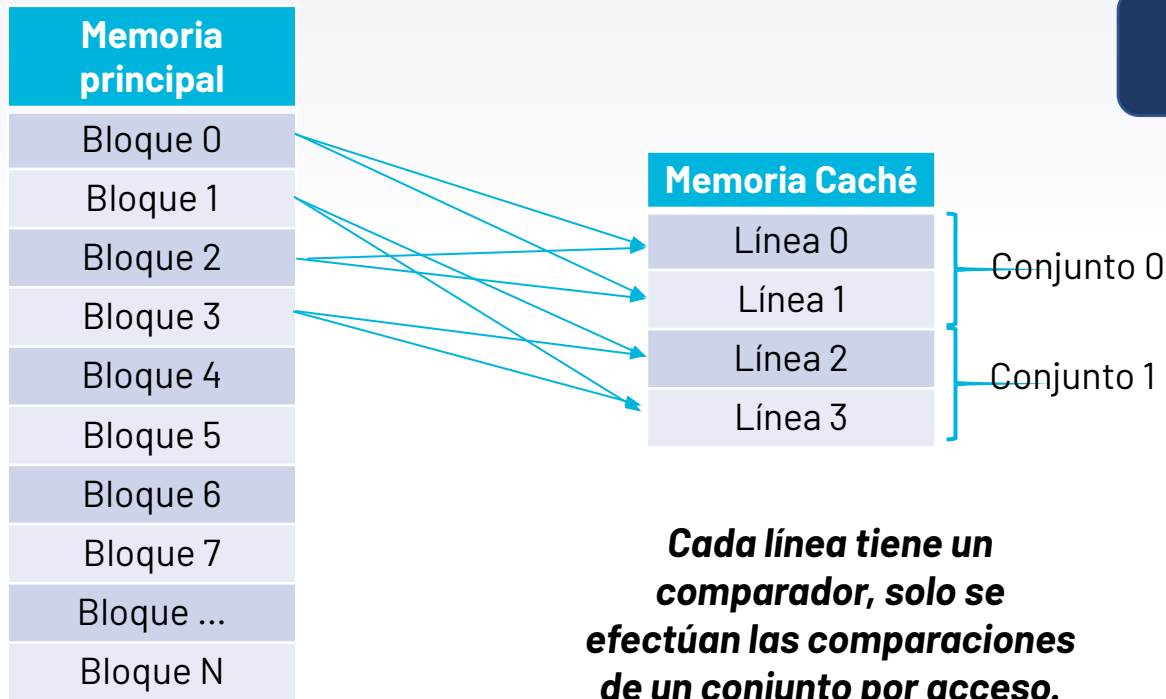
Cada línea tiene un comparador, solo se efectúan las comparaciones de un conjunto por acceso.

Línea MC	Bloque de MP que alojará
0	0, 2, 4, 8....
1	
2	1, 3, 5, 7...
3	

Asignación asociativa por conjuntos

Un bloque de memoria principal puede ir a parar a cualquier línea de caché dentro de un conjunto determinado.

Tamaño de la MC / tamaño de línea = cantidad de líneas



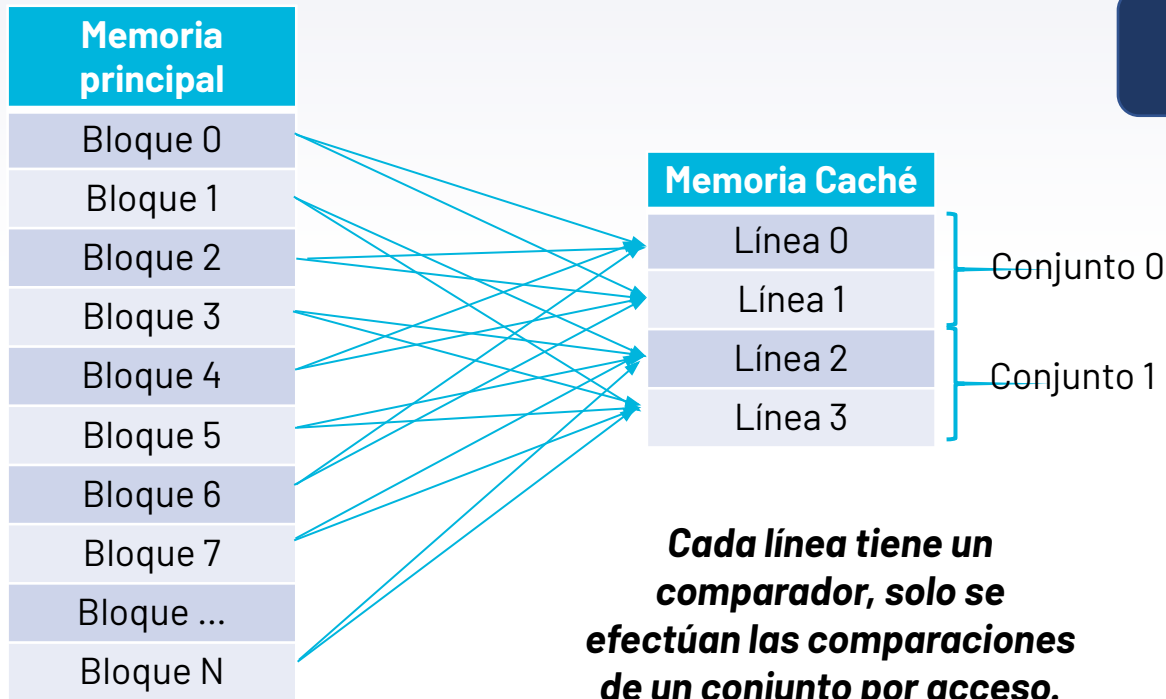
Cada línea tiene un comparador, solo se efectúan las comparaciones de un conjunto por acceso.

Línea MC	Bloque de MP que alojará
0	0, 2, 4, 8....
1	
2	1, 3, 5, 7...
3	

Asignación asociativa por conjuntos

Un bloque de memoria principal puede ir a parar a cualquier línea de caché dentro de un conjunto determinado.

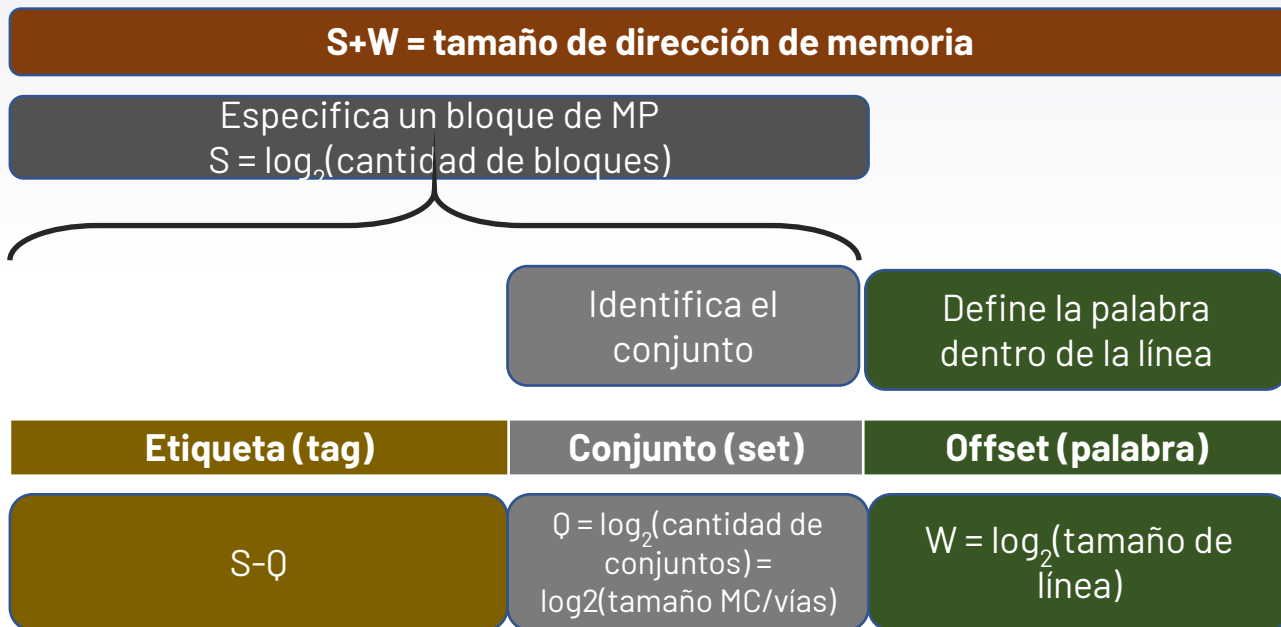
Tamaño de la MC / tamaño de línea = cantidad de líneas



Línea MC	Bloque de MP que alojará
0	0, 2, 4, 8....
1	
2	1, 3, 5, 7...
3	

Asignación asociativa por conjuntos

Interpretación de cada dirección de memoria según la CPU la coloca en el bus.



Una porción de la dirección (conjunto) se utiliza para especificar qué conjunto le corresponde. Se puede utilizar cualquier línea dentro de dicho conjunto.

La comparación se efectúa solamente sobre el campo de etiqueta para las líneas del conjunto.

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Etiqueta (tag)

Conjunto (set)

Palabra (offset)

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea =

Etiqueta (tag)

Conjunto (set)

Palabra (offset)

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea $\approx 32 \text{ KB} / 64 = 2^{15} / 2^6 = 2^9 = \mathbf{512 \text{ líneas}}$

Etiqueta (tag)

Conjunto (set)

Palabra (offset)

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = $32 \text{ KB} / 64 = 2^{15} / 2^6 = 2^9 = \mathbf{512 \text{ líneas}}$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(64) = 6$

Etiqueta (tag)

Conjunto (set)

Palabra (offset)

6 bits

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = $32 \text{ KB} / 64 = 2^{15} / 2^6 = 2^9 = \mathbf{512 \text{ líneas}}$

Cantidad de bloques =

tamaño MP / tamaño de línea = $4 \text{ GB} / 64 \text{ bytes} = 2^{32} / 2^6 = 2^{26} = 64 \text{ M bloques}$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(64) = 6$

Etiqueta (tag)

Conjunto (set)

Palabra (offset)

6 bits

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = $32 \text{ KB} / 64 = 2^{15} / 2^6 = 2^9 = \mathbf{512 \text{ líneas}}$

Cantidad de bloques =

tamaño MP / tamaño de línea = $4 \text{ GB} / 64 \text{ bytes} = 2^{32} / 2^6 = 2^{26} = 64 \text{ M bloques}$

Cantidad de conjuntos =

cantidad de líneas / líneas por conjunto = $512 / 4 = 2^9 / 2^2 = 2^7 = 128 \text{ conjuntos}$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(64) = 6$

Etiqueta (tag)

Conjunto (set)

Palabra (offset)

6 bits

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = $32 \text{ KB} / 64 = 2^{15} / 2^6 = 2^9 = \mathbf{512 \text{ líneas}}$

Cantidad de bloques =

tamaño MP / tamaño de línea = $4 \text{ GB} / 64 \text{ bytes} = 2^{32} / 2^6 = 2^{26} = 64 \text{ M bloques}$

Cantidad de conjuntos =

cantidad de líneas / líneas por conjunto = $512 / 4 = 2^9 / 2^2 = 2^7 = 128 \text{ conjuntos}$

Longitud del campo de conjunto =

$\log_2(\text{cantidad de conjuntos}) = \log_2(128) = \log_2(2^7) = 7$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(64) = 6$

Etiqueta (tag)

Conjunto (set)

Palabra (offset)

7 bits

6 bits

Asignación asociativa por conjuntos

Ejemplo: Supongamos direcciones de 32 bits (total direccionable = 4GB), líneas de 64 bytes, tamaño de caché 32 KB. Se trata de una memoria caché asociativa de 4 vías.

1. ¿En cuántas líneas se divide la memoria caché?
2. ¿Cuál es el tamaño de cada campo de la dirección?

Cantidad de líneas =

tamaño de MC / tamaño de línea = $32 \text{ KB} / 64 = 2^{15} / 2^6 = 2^9 = \mathbf{512 \text{ líneas}}$

Cantidad de bloques =

tamaño MP / tamaño de línea = $4 \text{ GB} / 64 \text{ bytes} = 2^{32} / 2^6 = 2^{26} = 64 \text{ M bloques}$

Cantidad de conjuntos =

cantidad de líneas / líneas por conjunto = $512 / 4 = 2^9 / 2^2 = 2^7 = 128 \text{ conjuntos}$

Longitud del campo de conjunto =

$\log_2(\text{cantidad de conjuntos}) = \log_2(128) = \log_2(2^7) = 7$

Longitud del campo offset =

$\log_2(\text{tamaño de línea}) = \log_2(64) = 6$

Longitud del campo de etiqueta =

Longitud de dirección - Campo conjunto - Campo palabra = $32 - 7 - 6 = 19$

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
19 bits	7 bits	6 bits

Asignación asociativa por conjuntos

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =

Etiqueta (tag) – 19 bits	Conjunto (set) – 7 bits	Palabra (offset) – 6 bits

En asignación asociativa por conjuntos se comparan en paralelo las etiquetas de las líneas del conjunto.

Asignación asociativa por conjuntos

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
01111101011101110001101100111000 base 2

Etiqueta (tag) – 19 bits	Conjunto (set) – 7 bits	Palabra (offset) – 6 bits

En asignación asociativa por conjuntos se comparan en paralelo las etiquetas de las líneas del conjunto.

Asignación asociativa por conjuntos

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
0111110101110111000 1101100 111000 base 2

Etiqueta (tag) – 19 bits	Conjunto (set) – 7 bits	Palabra (offset) – 6 bits
0111110101110111000	1101100	111000

En asignación asociativa por conjuntos se comparan en paralelo las etiquetas de las líneas del conjunto.

Asignación asociativa por conjuntos

¿Cómo funciona? Supongamos que la CPU solicita la dirección 0x7D771B38...

7D771B38 base 16 =
0111110101110111000 1101100 111000 base 2

Etiqueta (tag) – 19 bits	Conjunto (set) – 7 bits	Palabra (offset) – 6 bits
0111110101110111000	1101100	111000

La etiqueta de la dirección (0111110101110111000) se compara en forma simultánea con las etiquetas de las líneas de caché del conjunto 1101100₂ (108₁₀). Si existe en MC una línea cuya etiqueta tenga el mismo valor, es un **ACIERTO**. Por lo tanto se accede a la palabra ubicada en el offset 111000 (56₁₀) de la línea. Si ninguna de las etiquetas de MC del conjunto tiene el valor buscado, es una **FALLA**.

En asignación asociativa por conjuntos se comparan en paralelo las etiquetas de las líneas del conjunto.

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000

- Comparador
- Comparador
- Comparador
- Comparador

Conjunto	Etiqueta	Datos (64 bytes c/u)															
108	11101111000011110000																
108	11011111101011101110																
108	01000101011101110001																
108	11111001101011101010																
109																	
...																	

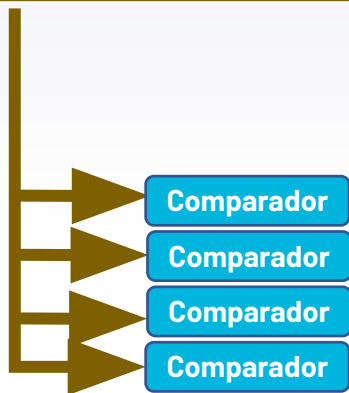
Memoria Caché

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Bus de direcciones

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000



Conjunto	Etiqueta	Datos (64 bytes c/u)
108	11101111000011110000	
108	11011111101011101110	
108	01000101011101110001	
108	11111001101011101010	
109		
...		

Memoria Caché

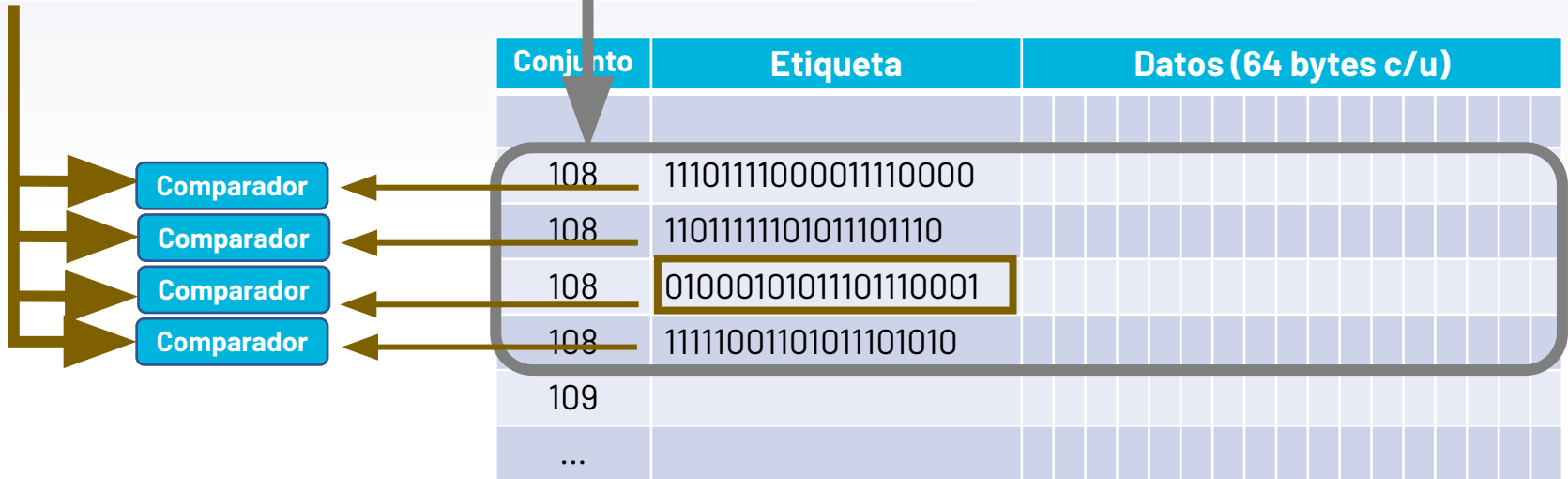
Bus de datos

1. Se comparan *simultáneamente* las etiquetas de MC del conjunto seleccionado

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Bus de direcciones

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000



Memoria Caché

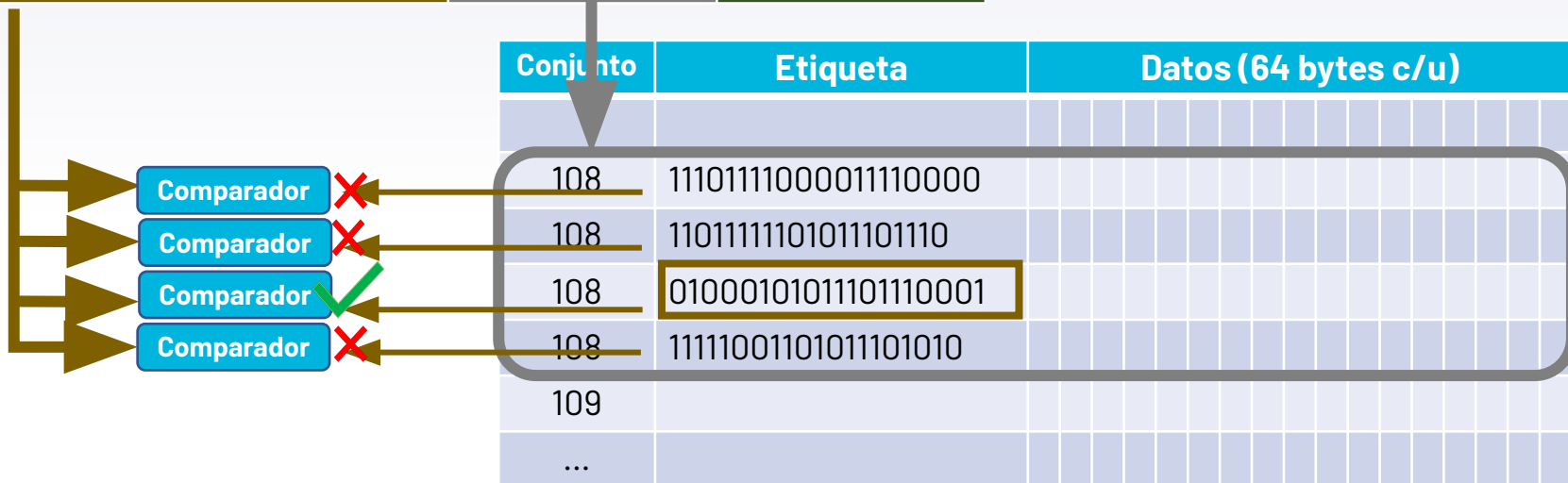
Bus de datos

1. Se comparan *simultáneamente* las etiquetas de MC del conjunto seleccionado

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Bus de direcciones

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000



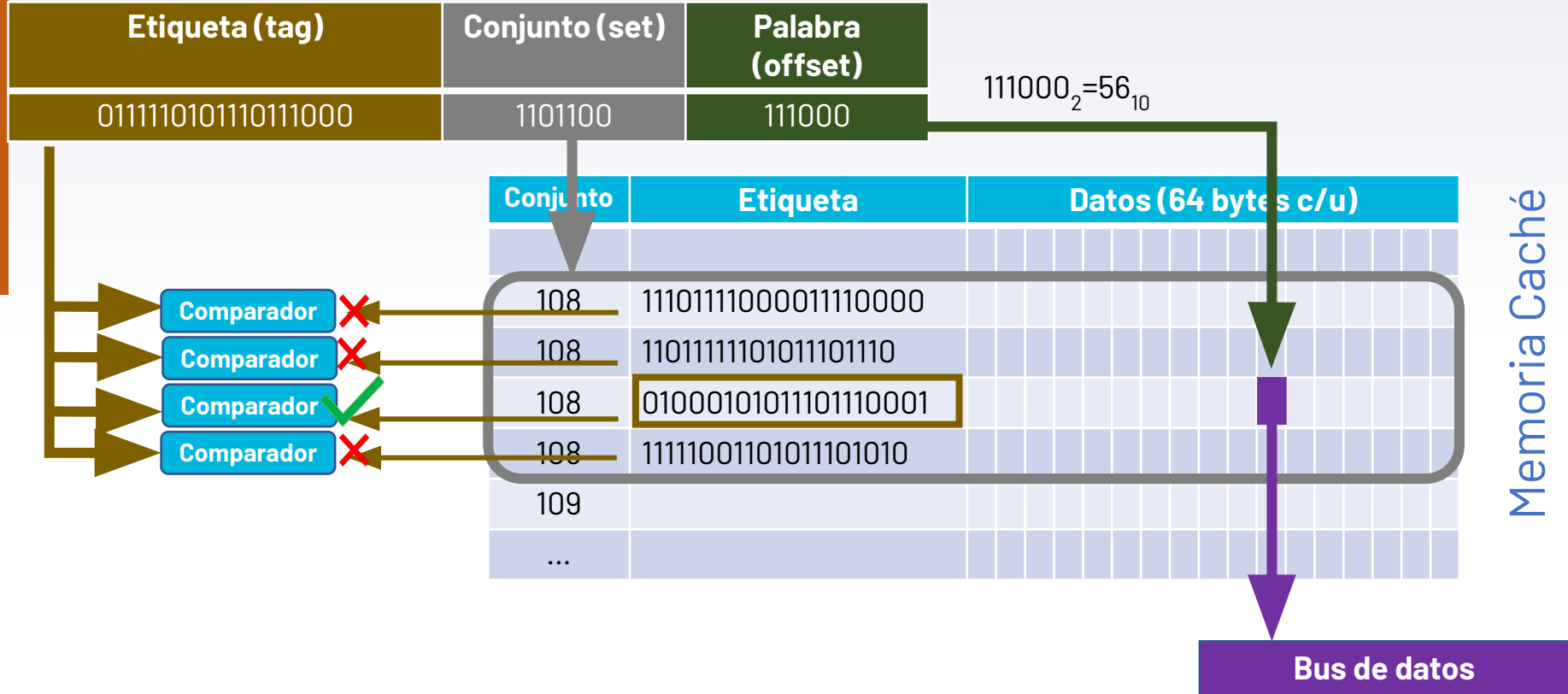
Memoria Caché

Bus de datos

1. Se comparan *simultáneamente* las etiquetas de MC del conjunto seleccionado

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra está en la memoria caché (**HIT**)

Bus de direcciones



1. Se comparan *simultáneamente* las etiquetas de MC del conjunto seleccionado
2. Si existe una coincidencia se devuelve el contenido de caché en el offset solicitado.

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra NO está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000

Comparador

Comparador

Comparador

Comparador

Conjunto	Etiqueta	Datos (64 bytes c/u)															
108	11101111000011110000																
108	1101111101011101110																
108	01000101011101110001																
108	11111001101011101010																
109																	
...																	

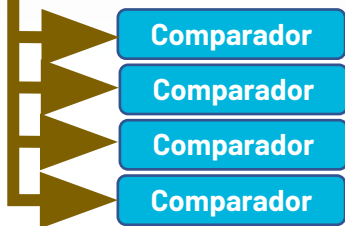
Memoria Caché

Bus de datos

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra NO está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000



Conjunto	Etiqueta	Datos (64 bytes c/u)
108	11101111000011110000	
108	11011111101011101110	
108	01000101011101110001	
108	11111001101011101010	
109		
...		

Memoria Caché

Bus de datos

1. Se comparan *simultáneamente* las etiquetas de MC del conjunto seleccionado

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra NO está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000



Memoria Caché

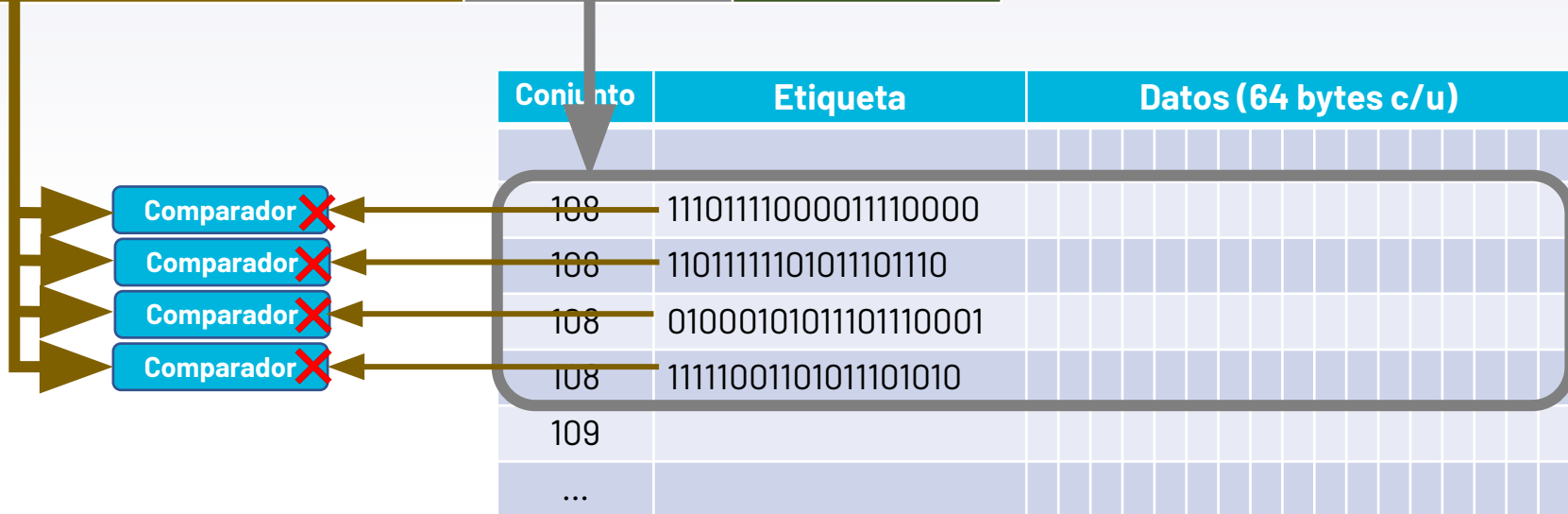
Bus de datos

1. Se comparan *simultáneamente* las etiquetas de MC del conjunto seleccionado

¿Cómo funciona? La CPU solicita la dirección 0x7D771B38, y la palabra NO está en la memoria caché (**MISS**)

Bus de direcciones

Etiqueta (tag)	Conjunto (set)	Palabra (offset)
0111110101110111000	1101100	111000



Memoria Caché

Bus de datos

1. Se comparan *simultáneamente* las etiquetas de MC del conjunto seleccionado
2. Si no existe coincidencia se debe acceder a MP.

Comparativa: Ventajas y desventajas

- Combina simpleza de Directa con eficiencia de Asociativa.
- Cuanto más grande el conjunto (n- vías) mayor la performance pero también la complejidad y costo.
- Obliga a implementar políticas de reemplazo
- Mejor performance

Intel Pentium 4 usa 8 vías en la MC de datos de L1. También de 4, 8, 16 o 24 vías para L2.
ARM implementa asociativa por conjuntos de 8, 32 y 64 vías!



- Diseño simple: menor tamaño de la memoria de etiquetas y comparadores más reducidos
 - No requiere búsqueda asociativa.
 - Se suele usar para caché de instrucciones.
 - Riesgo de thrashing
 - Peor performance
-
- Buena performance, pero también diseño más complejo.
 - Comparadores de mayor tamaño
 - Obliga a implementar políticas de reemplazo
 - Se utiliza en el TLB con unas pocas docenas de entradas (muy pequeña)

Asignación asociativa y asociativa por conjuntos

Políticas de reemplazo

En asignación directa cada bloque de MP tiene una sola línea de MC donde puede alojarse. Pero en mapeo asociativo y en asociativo por conjuntos, si la memoria caché (o el conjunto requerido) se llenó y se produce una falla... ¿a quién se reemplaza?



- **LRU (least recently used)**: Reemplazo el que hace más tiempo que no uso.
- **LFU (least frequently used)**: Reemplazo el que menos veces usé.
- **FIFO (first in first out)**: Reemplazo el más viejo de los que tengo.
- **RANDOM**: Reemplazo cualquiera.

Políticas de reemplazo

No se aplica en memoria caché de mapeo directo

LRU (least recently used)

Se basa en la localidad temporal. A cada línea de caché se le agrega una marca de tiempo (time stamp). En cada acceso a MC se actualiza la marca de la línea accedida. Se reemplaza la línea que hace más tiempo no se acede. Cada marca de tiempo requiere almacenamiento adicional y los circuitos para actualizarse.

LFU (least frequently used)

A cada línea de caché se le agrega un contador. En cada acceso a MC se actualiza el contador de la línea accedida. Se reemplaza la línea cuyo valor sea el inferior.

FIFO (first in first out, también llamado Round Robin)

Al completarse la MC se reemplaza la línea que entró primero. Luego la segunda, y así. Es sencillo de implementar con la técnica de búffer circular o round robin.

RANDOM

No requiere lógica adicional... y funciona bastante bien!

Pseudo-LRU (pseudo least recently used)

En asociativa por conjuntos: se asigna un bit uso por vía para cada línea. Cuando se accede a un conjunto, el bit de la línea accedida se activa. Si todos los bits están activos, se desactivan todos con excepción del último. Cuando una línea debe reemplazarse, se escoge la que no esté activa (si hay más de una.... usualmente random)

Pseudo-LRU

Ejemplo: asociativa por conjuntos de 2 vías

Se reemplaza la línea en el conjunto que ha estado más tiempo en la caché sin que se haga referencia a ella. Cuando se accede a una línea se pone en 1 el bit de uso de la misma y en 0 los bits de uso de las otras líneas del conjunto. (Caso particular de 2 vías).

Al momento de deber reemplazarse una línea, la que tenga el bit de uso en cero será la *víctima*.

Uso	Etiqueta	Datos															
0	10101111																
1	11001111																
1	00001001																
0	11111000																

Conjunto n

Conjunto n+1

Etiqueta	Conjunto	Palabra
10101111	n	xxxxxxx

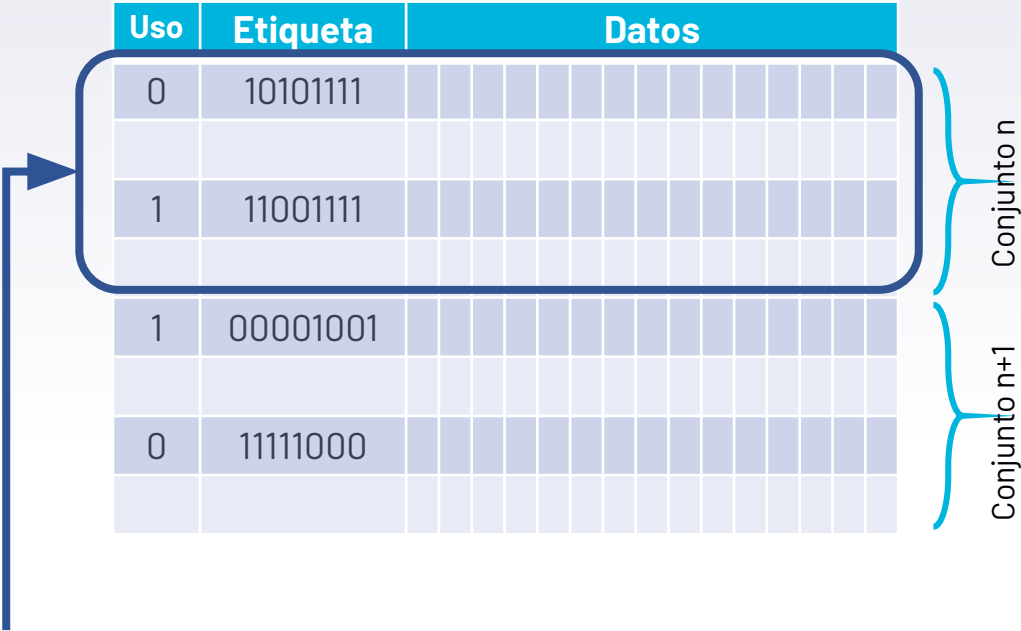


Bus de direcciones

Pseudo-LRU

Ejemplo: asociativa por conjuntos de 2 vías

Se reemplaza la línea en el conjunto que ha estado más tiempo en la caché sin que se haga referencia a ella. Cuando se accede a una línea se pone en 1 el bit de uso de la misma y en 0 los bits de uso de las otras líneas del conjunto. (Caso particular de 2 vías). Al momento de deber reemplazarse una línea, la que tenga el bit de uso en cero será la víctima.



Etiqueta	Conjunto	Palabra
10101111	n	xxxxxx

Bus de direcciones

Pseudo-LRU

Ejemplo: asociativa por conjuntos de 2 vías

Se reemplaza la línea en el conjunto que ha estado más tiempo en la caché sin que se haga referencia a ella. Cuando se accede a una línea se pone en 1 el bit de uso de la misma y en 0 los bits de uso de las otras líneas del conjunto. (Caso particular de 2 vías).

Al momento de deber reemplazarse una línea, la que tenga el bit de uso en cero será la víctima.

Uso	Etiqueta	Datos									
0	10101111										
1	11001111										
1	00001001										
0	11111000										

Conjunto n

Conjunto n+1

Etiqueta	Conjunto	Palabra
10101111	n	xxxxxxx

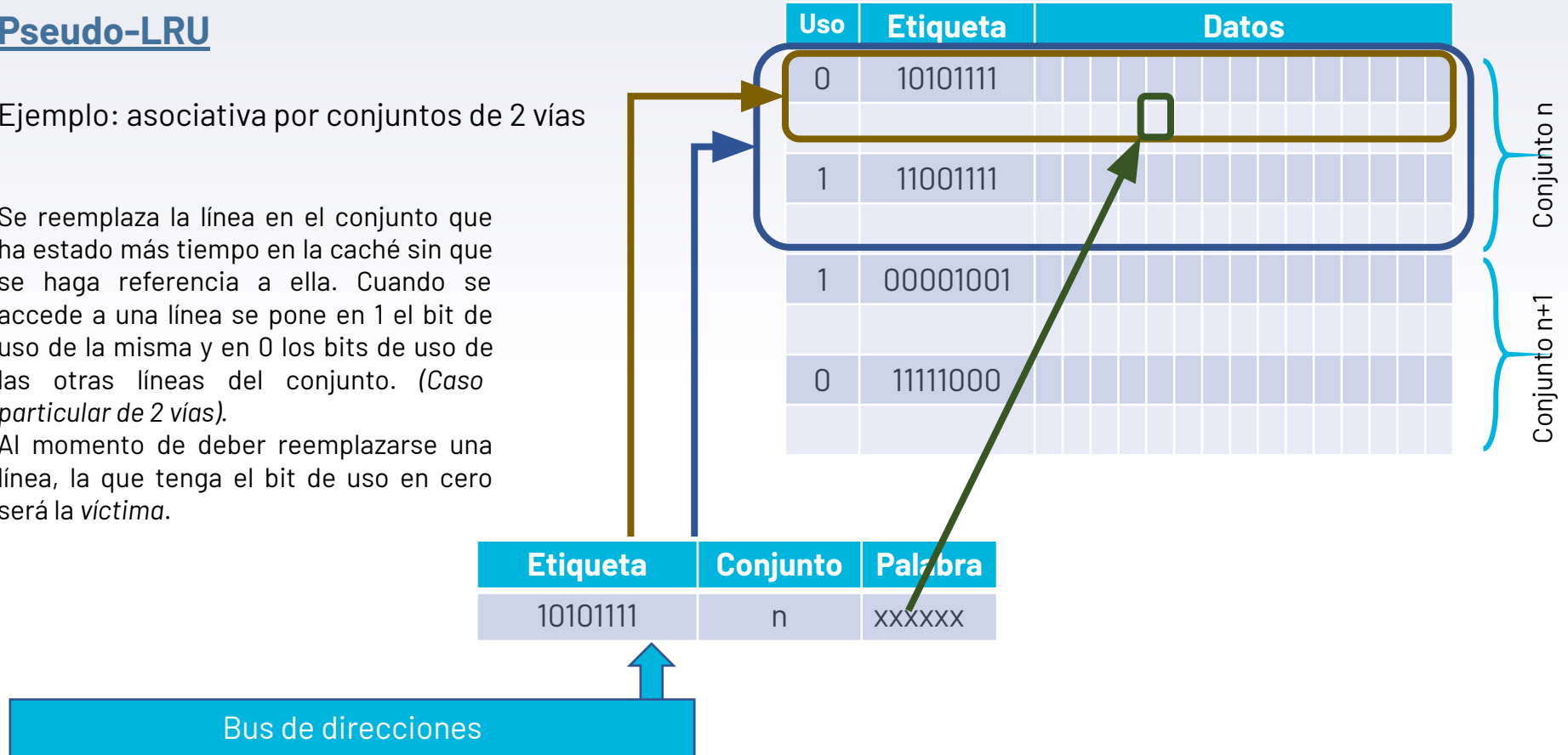
Bus de direcciones

Pseudo-LRU

Ejemplo: asociativa por conjuntos de 2 vías

Se reemplaza la línea en el conjunto que ha estado más tiempo en la caché sin que se haga referencia a ella. Cuando se accede a una línea se pone en 1 el bit de uso de la misma y en 0 los bits de uso de las otras líneas del conjunto. (Caso particular de 2 vías).

Al momento de deber reemplazarse una línea, la que tenga el bit de uso en cero será la víctima.

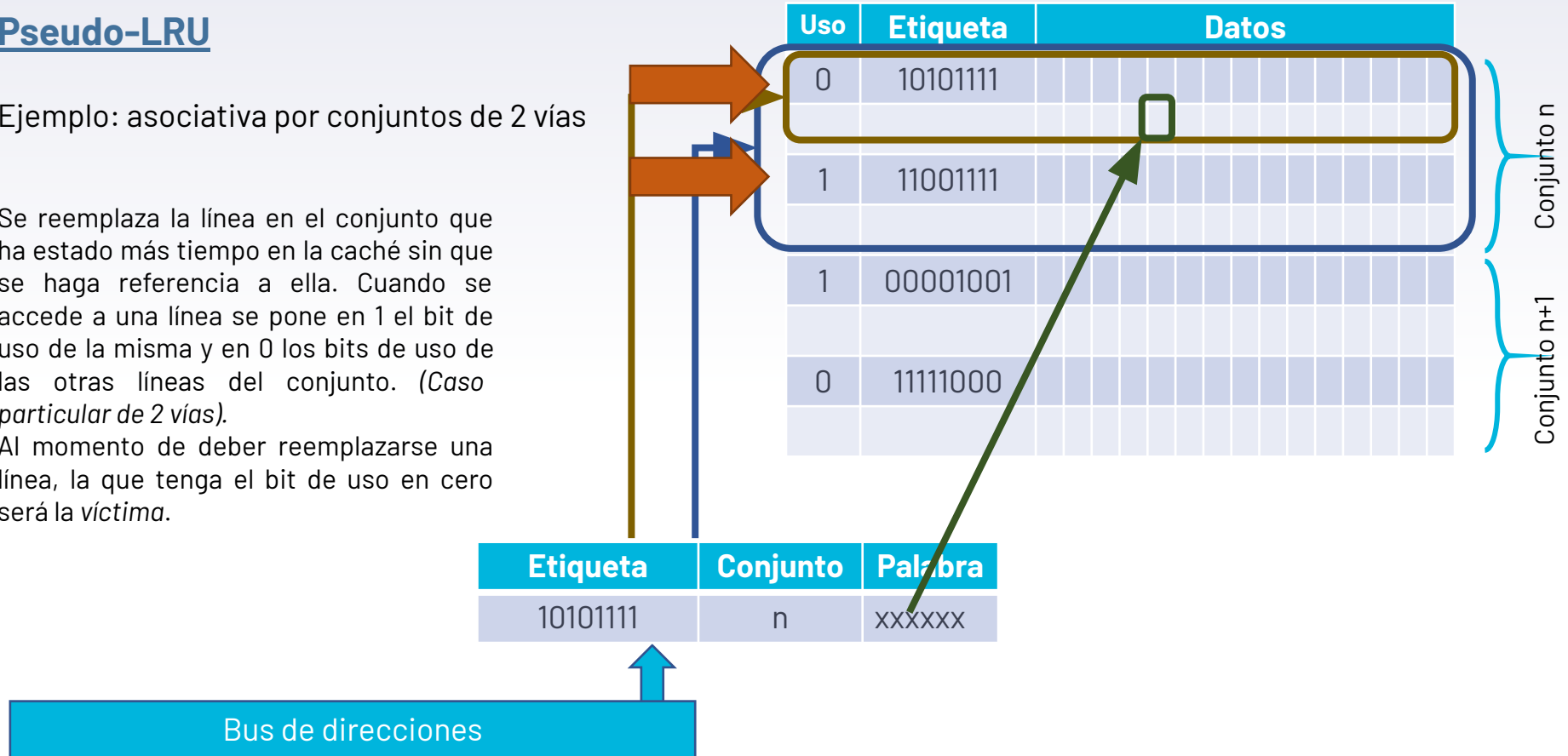


Pseudo-LRU

Ejemplo: asociativa por conjuntos de 2 vías

Se reemplaza la línea en el conjunto que ha estado más tiempo en la caché sin que se haga referencia a ella. Cuando se accede a una línea se pone en 1 el bit de uso de la misma y en 0 los bits de uso de las otras líneas del conjunto. (Caso particular de 2 vías).

Al momento de deber reemplazarse una línea, la que tenga el bit de uso en cero será la víctima.

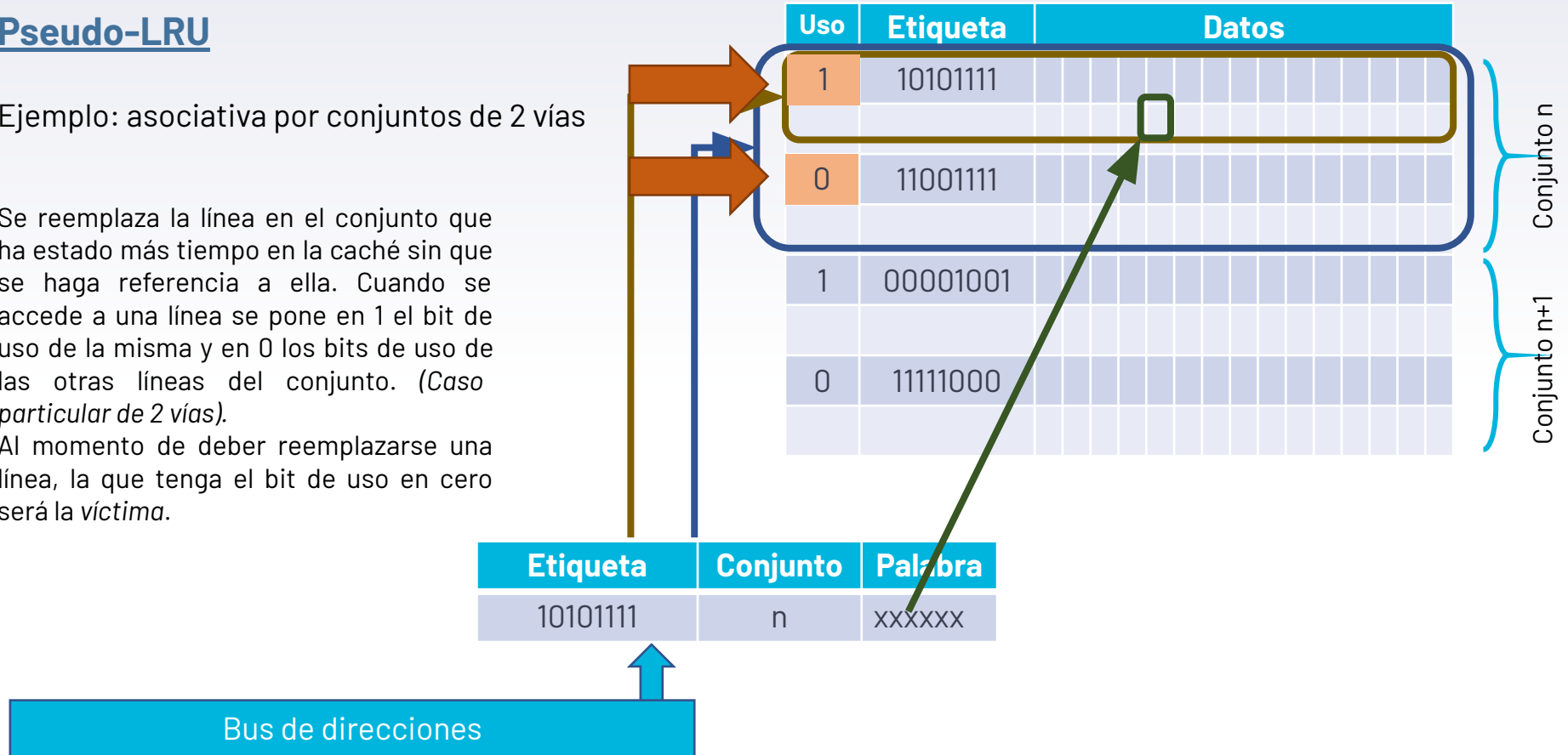


Pseudo-LRU

Ejemplo: asociativa por conjuntos de 2 vías

Se reemplaza la línea en el conjunto que ha estado más tiempo en la caché sin que se haga referencia a ella. Cuando se accede a una línea se pone en 1 el bit de uso de la misma y en 0 los bits de uso de las otras líneas del conjunto. (Caso particular de 2 vías).

Al momento de deber reemplazarse una línea, la que tenga el bit de uso en cero será la víctima.



Asignación asociativa por conjuntos

Políticas de reemplazo

Asociatividad									
Dos vías				Cuatro vías			Ocho vías		
Tamaño	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114,1	117,3	115,5	111,7	115,1	113,3	109,0	111,8	110,4
64 KB	103,4	104,3	103,9	102,4	102,3	103,1	99,7	100,5	100,3
256 KB	92,2	92,1	92,5	92,1	92,1	92,5	92,1	92,1	92,5

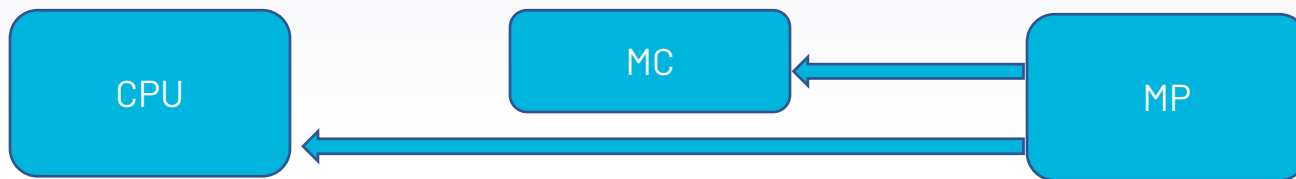
Fallas en caché cada 1000 instrucciones comparando los algoritmos de reemplazo LRU, RANDOM y FIFO para distintos tamaños de caché y número de conjuntos. Hay muy poca diferencia entre LRU y random para los tamaños mayores de caché, aunque LRU se impone en memorias pequeñas. El benchmark se hizo con tamaño de bloque de 64 bytes para arquitectura Alpha.

Política de carga y escritura

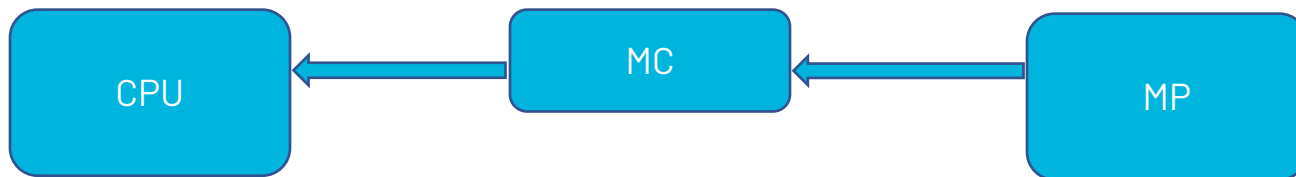
¿Cómo se realiza la carga de MC cuando se produce un **falla**?

Carga inmediata vs carga diferida (Load through vs load back)

- Carga inmediata (load through): Se carga el bloque de MC y en forma simultánea se transfiere la palabra a la CPU.



- Carga diferida (load back): Se actualiza el bloque en MC y luego la CPU accede a la MC.

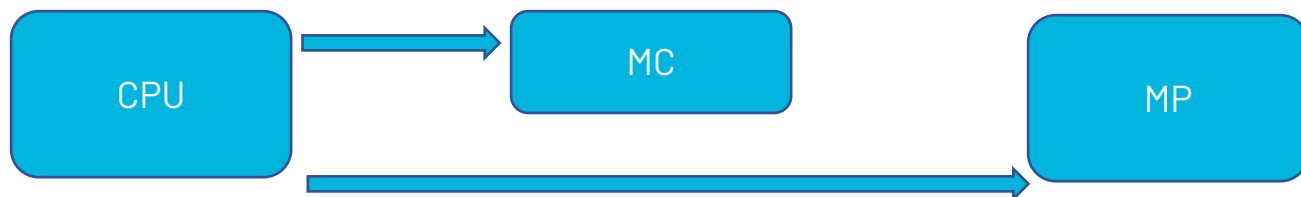


Política de carga y escritura

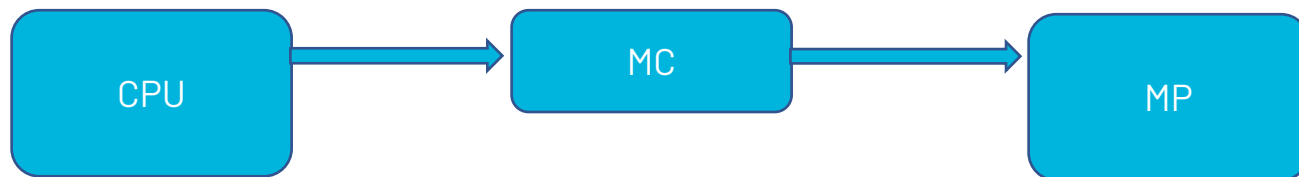
¿Cómo se realiza la escritura desde CPU a MP?

Acierto: Escritura inmediata vs escritura diferida (Write through vs write back)

- Escritura inmediata (write through): Se actualiza la MC y en forma simultánea se actualiza MP.
 - Segura (coherencia) y sencilla de implementar.
 - Peor performance, tiende a cuellos de botella y exceso de tráfico a memoria.
 - El procesador puede que deba demorarse (*stall*) durante la escritura. Se usan *write buffers* para reducir esto.



- Escritura diferida (write back): Se actualiza el bloque en MC; cuando se reemplaza la línea se actualiza MP.
 - Se utiliza un bit de suciedad (**dirty bit**) para indicar que la línea fue modificada.
 - Si se producen lecturas/escrituras Mem <-> E/S o en entornos multiprocesador puede generar inconsistencias.



Política de carga y escritura

¿Cómo se realiza la escritura desde CPU a MP?

Falla: Asignación y escritura vs escritura sin asignación (Write allocate vs write no allocate)

- Asignación y escritura (write allocate): se carga el bloque en MC y se actualiza



- Escritura sin asignación (write no allocate): se actualiza en MP sin cargar en MC

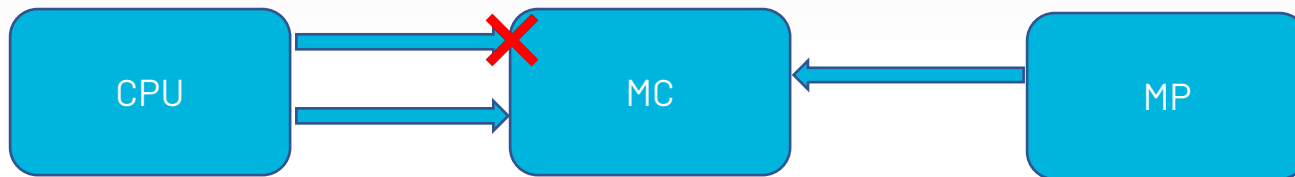


Política de carga y escritura

¿Cómo se realiza la escritura desde CPU a MP?

Falla: Asignación y escritura vs escritura sin asignación (Write allocate vs write no allocate)

- Asignación y escritura (write allocate): se carga el bloque en MC y se actualiza



- Escritura sin asignación (write no allocate): se actualiza en MP sin cargar en MC

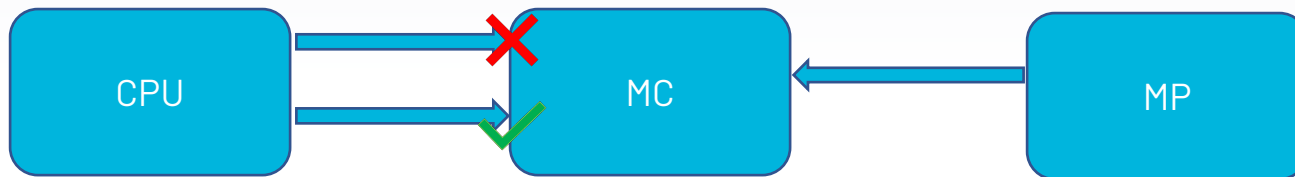


Política de carga y escritura

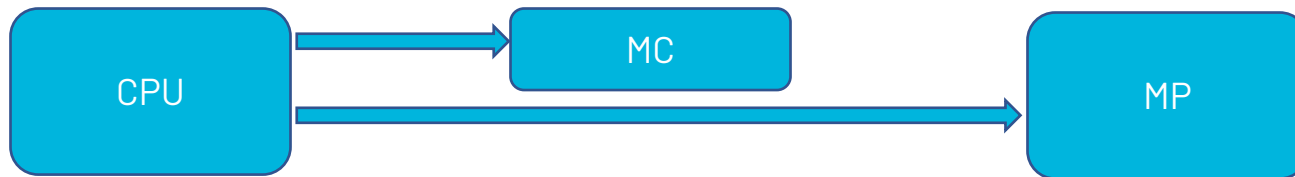
¿Cómo se realiza la escritura desde CPU a MP?

Falla: Asignación y escritura vs escritura sin asignación (Write allocate vs write no allocate)

- Asignación y escritura (write allocate): se carga el bloque en MC y se actualiza



- Escritura sin asignación (write no allocate): se actualiza en MP sin cargar en MC

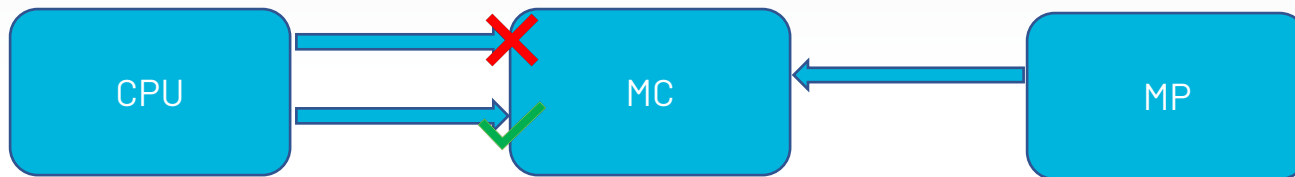


Política de carga y escritura

¿Cómo se realiza la escritura desde CPU a MP?

Falla: Asignación y escritura vs escritura sin asignación (Write allocate vs write no allocate)

- Asignación y escritura (write allocate): se carga el bloque en MC y se actualiza



- Escritura sin asignación (write no allocate): se actualiza en MP sin cargar en MC

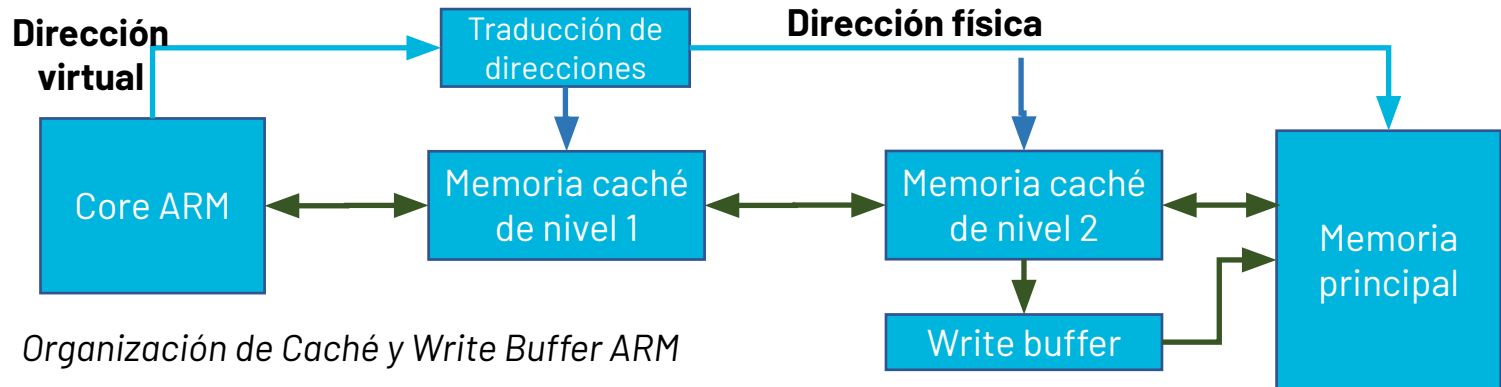


Write buffer: mejora para escrituras de ARM

- Consiste en un pequeño buffer (conjunto de direcciones y un conjunto de palabras) FIFO que se ubica entre la MC y la MP.
- Es más chico que la MC: puede contener hasta cuatro direcciones independientes.

Cómo funciona

1. Cuando el procesador realiza una escritura (CPU → MC), los datos se colocan en el buffer de escritura a la velocidad de reloj del procesador (CPU → MC → WB), y el procesador continúa la ejecución.
2. Luego se realiza la escritura desde el write buffer en paralelo.
Si el write buffer está lleno el procesador se demora hasta que haya suficiente espacio en él.
3. Mientras se realizan operaciones que no son de escritura, el write buffer continúa escribiendo en memoria principal hasta que esté completamente vacío.



Bits de control

Se asocian a cada **línea** de memoria caché.

Bit de suciedad (dirty bit / use bit)

Cada línea tiene asociado un bit de suciedad que indica si la línea sufrió modificaciones y por lo tanto debe actualizarse la MP antes de ser reemplazada. *Dirty* (sucio) equivale a modificado. *Clean* es no-modificado.

Bit de validez

Cada línea tiene asociado un bit de validez. Cuando una línea de MC aun no ha sido cargada se indica con este bit que no es válida. Si este bit no está activo, la etiqueta de la línea no coincidirá en ninguna comparación. Se usa para indicar si la línea contiene o no un bloque que pertenezca al proceso en ejecución. *Limpiar* la MC equivale a poner los bits de validez en cero.



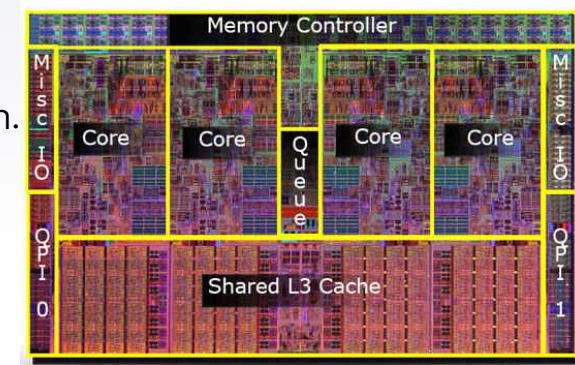
Cache Disable (CD, solo Pentium 4)

Habilita/deshabilita el mecanismo de carga de la línea de caché.

Si CD=1 la caché no se llenará con nuevos datos cuando se produzcan fallos, pero continuará funcionando para los aciertos. Si CD=0 los fallos producirán que se actualice la MC.

Not Write-Through (NW, solo Pentium 4)

Selecciona el modo de operación para la caché de datos. Si NW=1 la caché operará en Write Back; Si está en cero operará en Write-Through.



Core valid bit (Intel Nehalem y posteriores)

Se agrega un bit por core a cada línea de MC L3 en procesadores multicore (NEHALEM en adelante; luego vino SANDY BRIDGE y IVY BRIDGE). De esa forma detecta quien está accediendo a cada línea.

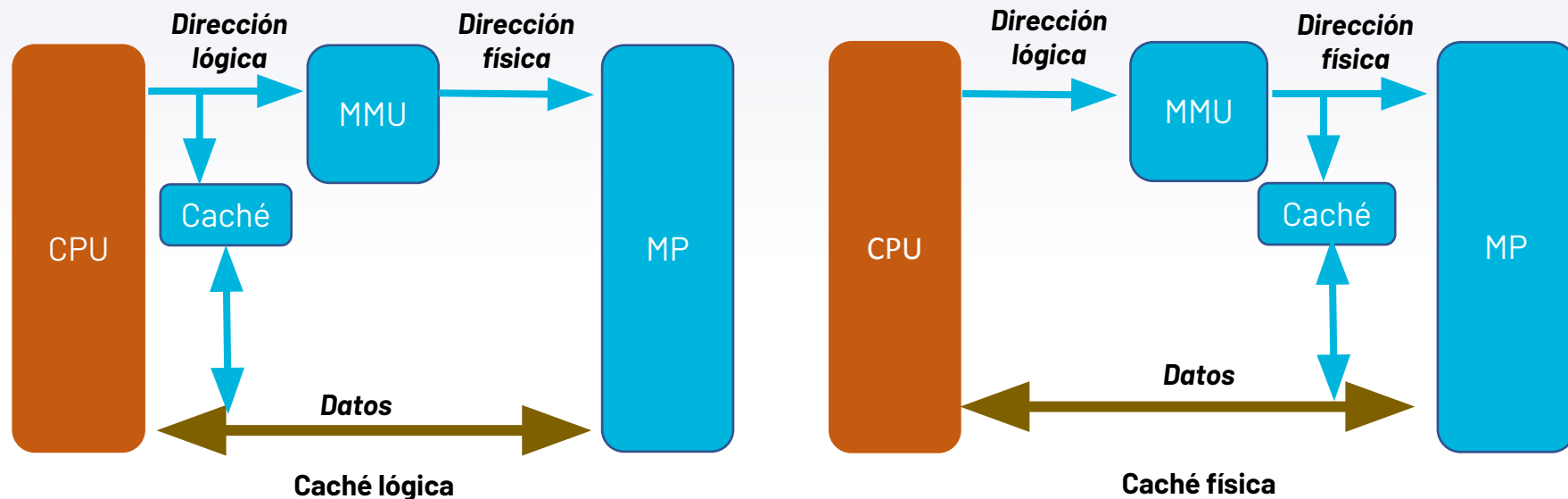
Si la línea puede que esté en L1/L2 del core, se indica con Core valid bit=1. Si más de un bit=1 la línea no se puede modificar en ningún core.

Write through / write back (Motorola PPC 601)

Permite que se asigne a las líneas de MC la política write through (contra write back que es default). En un esquema de 3 buses puede ser necesario para no generar inconsistencias con interfaces de E/S.

Direccionamiento virtual

Los procesos manejan las direcciones desde un punto de vista lógico, sin preocuparse por el total de memoria física disponible. Los campos de dirección de las instrucciones de máquina contienen direcciones virtuales.



- La CPU accede a MC sin pasar por la MMU (más rápido)
- Pero dado que la mayoría de los sistemas de memoria virtual le dan a cada proceso el mismo espacio de direccionamiento virtual, la MC debe limpiarse en cada cambio de contexto, o deben agregarse bits para identificar el espacio de direccionamiento virtual.
- L1 puede ser lógica y L2+ física en un mismo sistema.

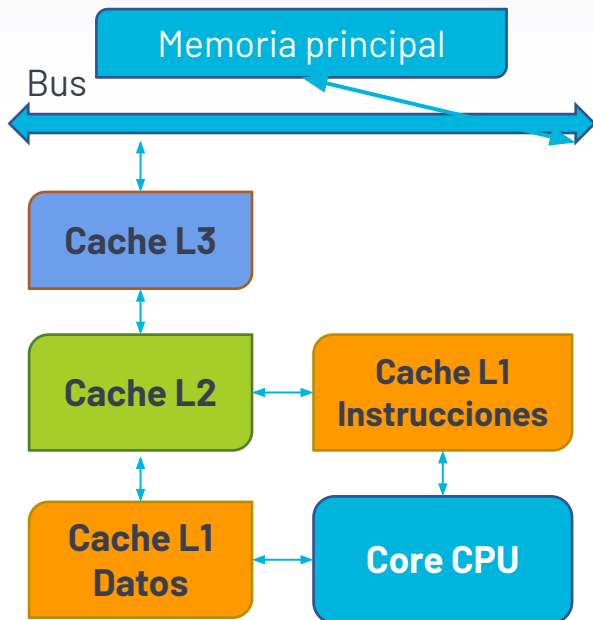
Cache unificada (unified) vs fraccionada o dividida (Split)

Caché unificada: almacena tanto instrucciones como datos

- Mayor tasa de aciertos
- Balancea la carga de datos e instrucciones
- Sencilla: una sola para diseñar e implementar
- Se suele implementar en L2, L3

Caché dividida: existen dos MC, una para instrucciones y otra para datos

- Elimina posibles bloqueos entre la unidad de búsqueda/decodificación y ejecución
- Importante para pipelining
- Se suele implementar en L1



Fallas cada mil instrucciones

Tamaño (KB)	Cache de instrucciones	Caché de datos	Caché unificada
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

74% de referencias a instrucciones y 26% a datos. MC asociativa de 2 vías, bloques de 64 bytes, arquitectura Alpha.

Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

Asociativa

Línea:	0	1	2	3	4	5	6	7
Memoria caché								

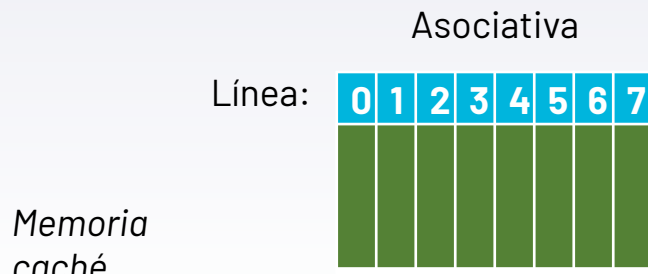
Directa

Línea:	0	1	2	3	4	5	6	7

Bloque:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Memoria principal																																

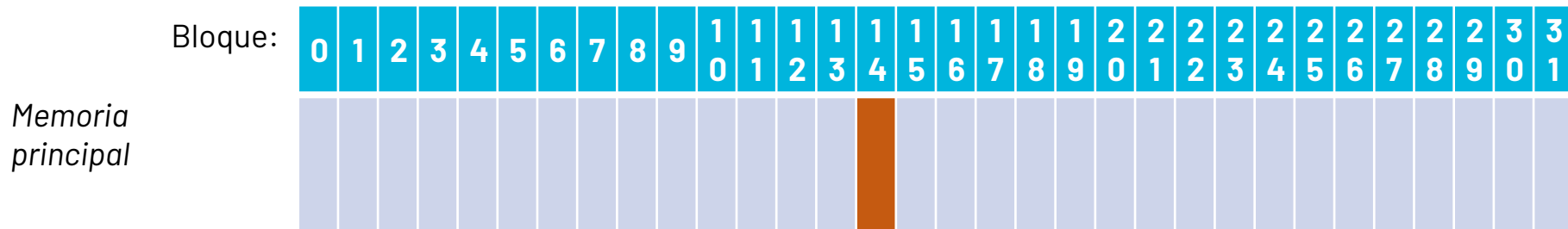
Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.



El bloque 14 podrá ubicarse en cualquier línea libre

El bloque 14 se ubicará en la línea 6 (resto entre 14 y 8)

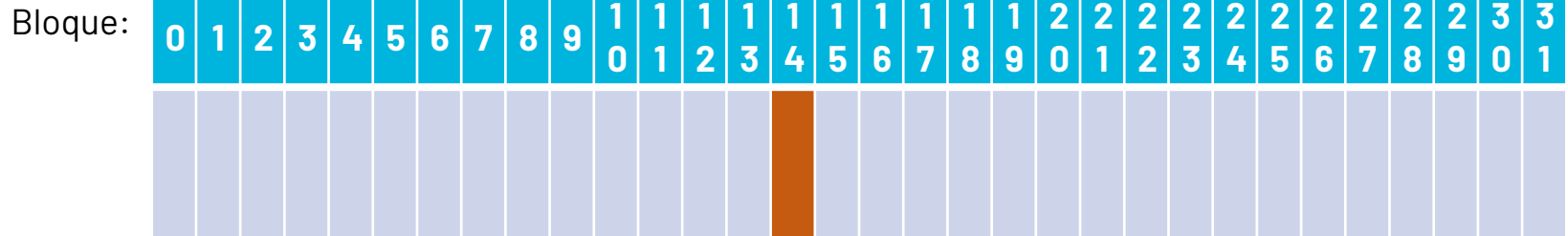
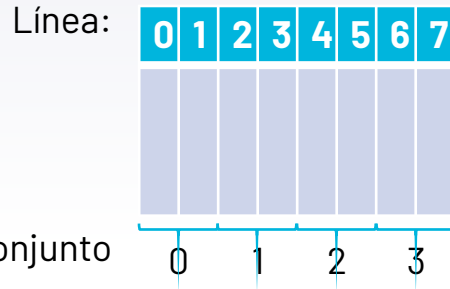


Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

Dos formas de razonarlo...

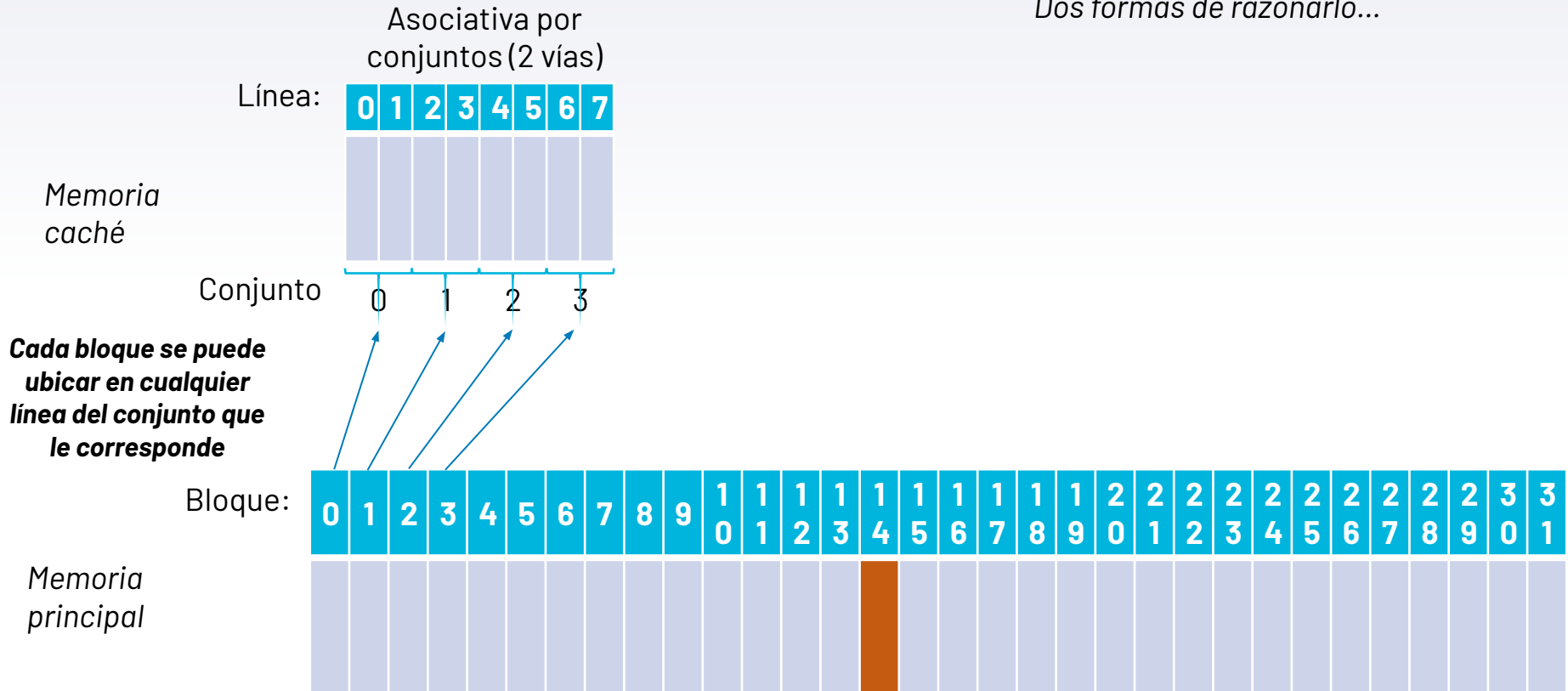
Asociativa por
conjuntos (2 vías)



Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

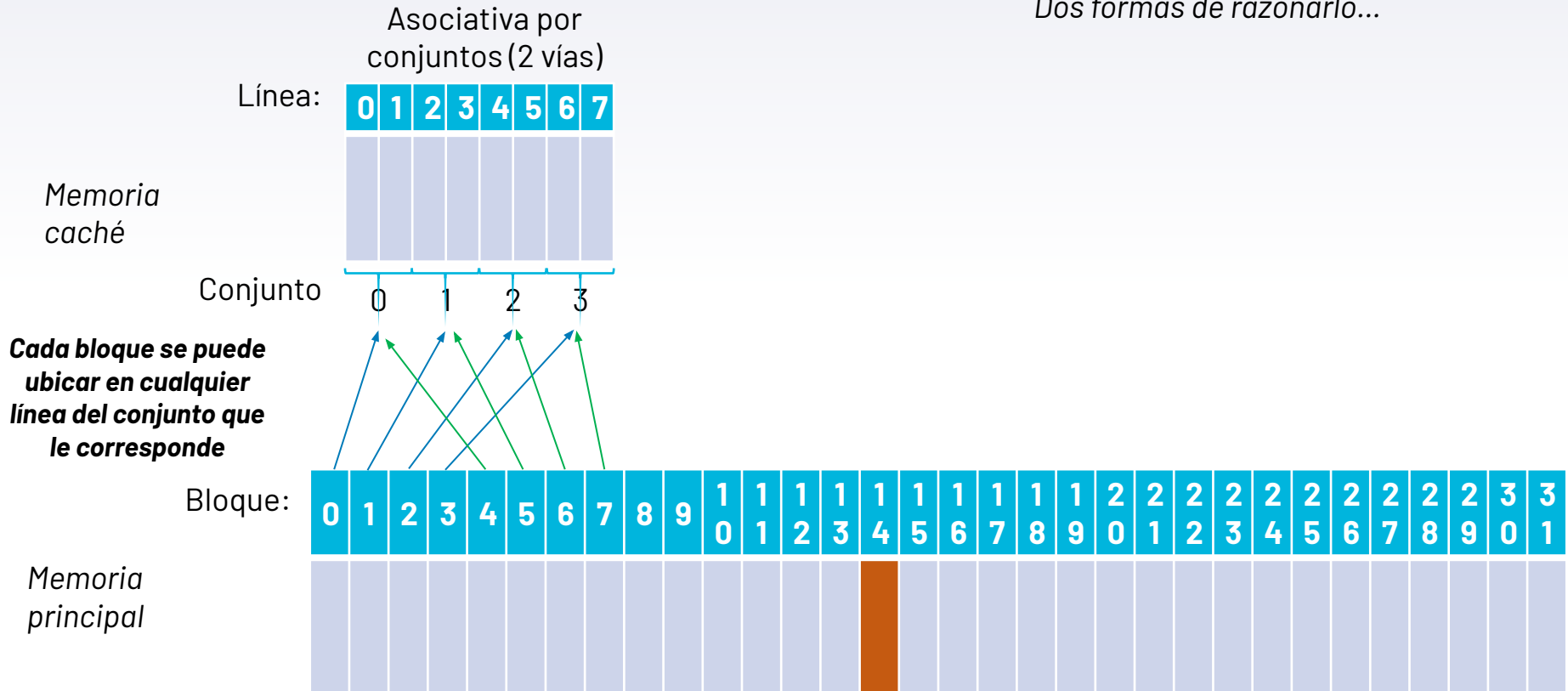
Dos formas de razonarlo...



Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

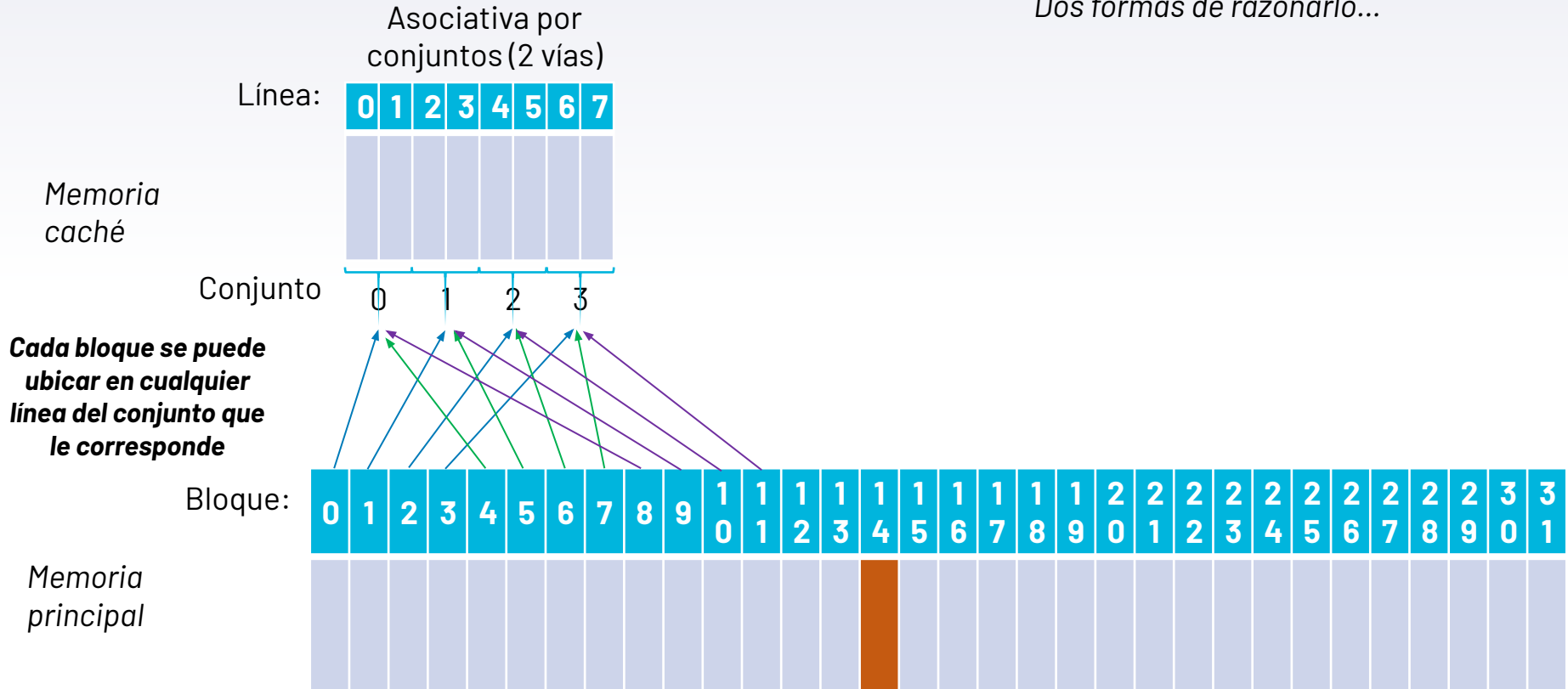
Dos formas de razonarlo...



Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

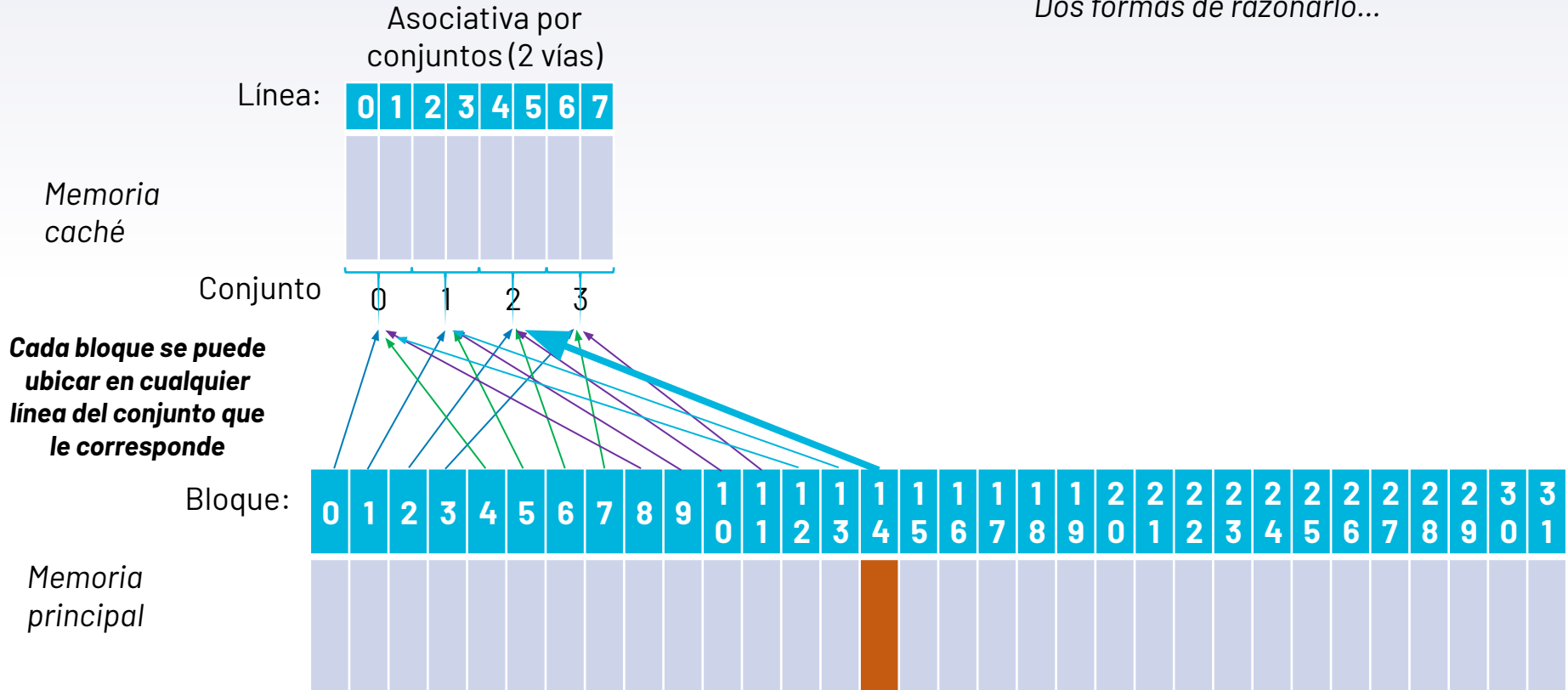
Dos formas de razonarlo...



Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

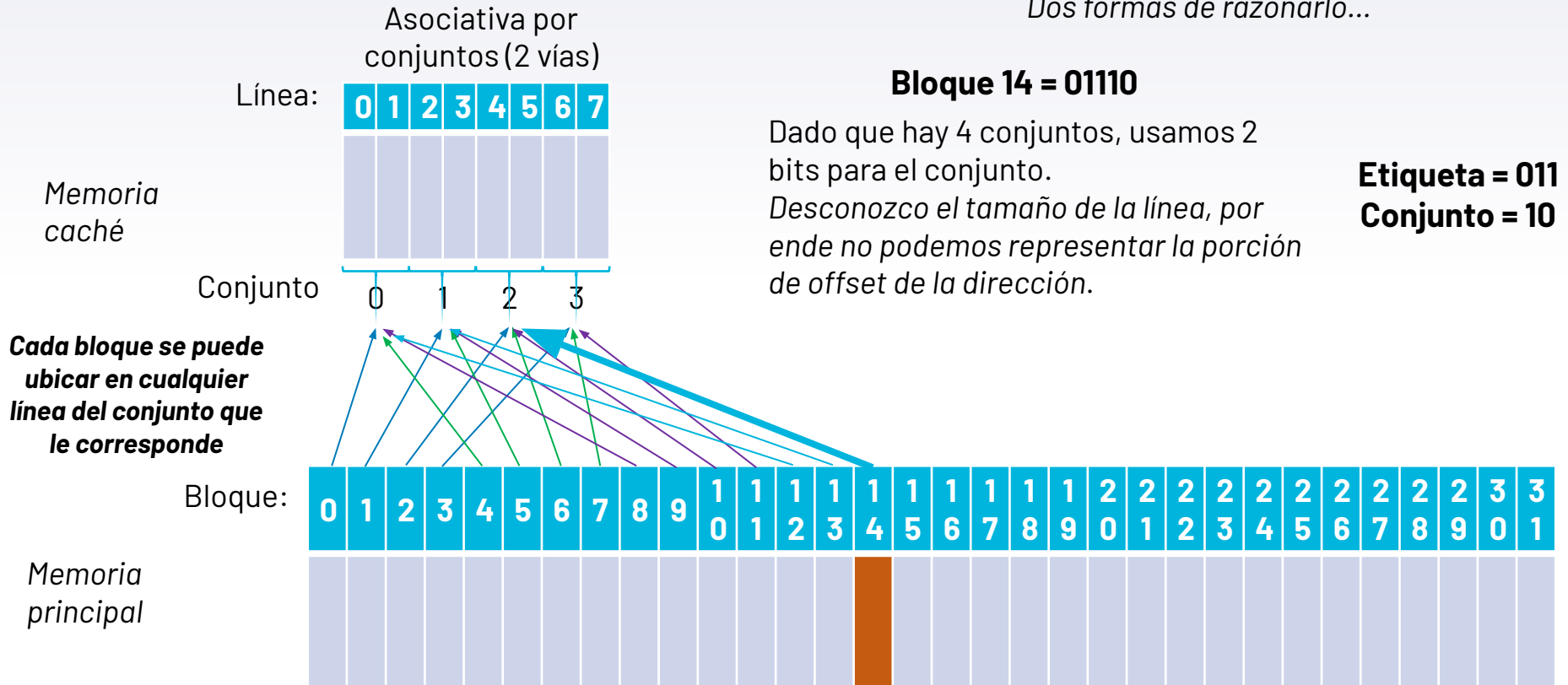
Dos formas de razonarlo...



Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

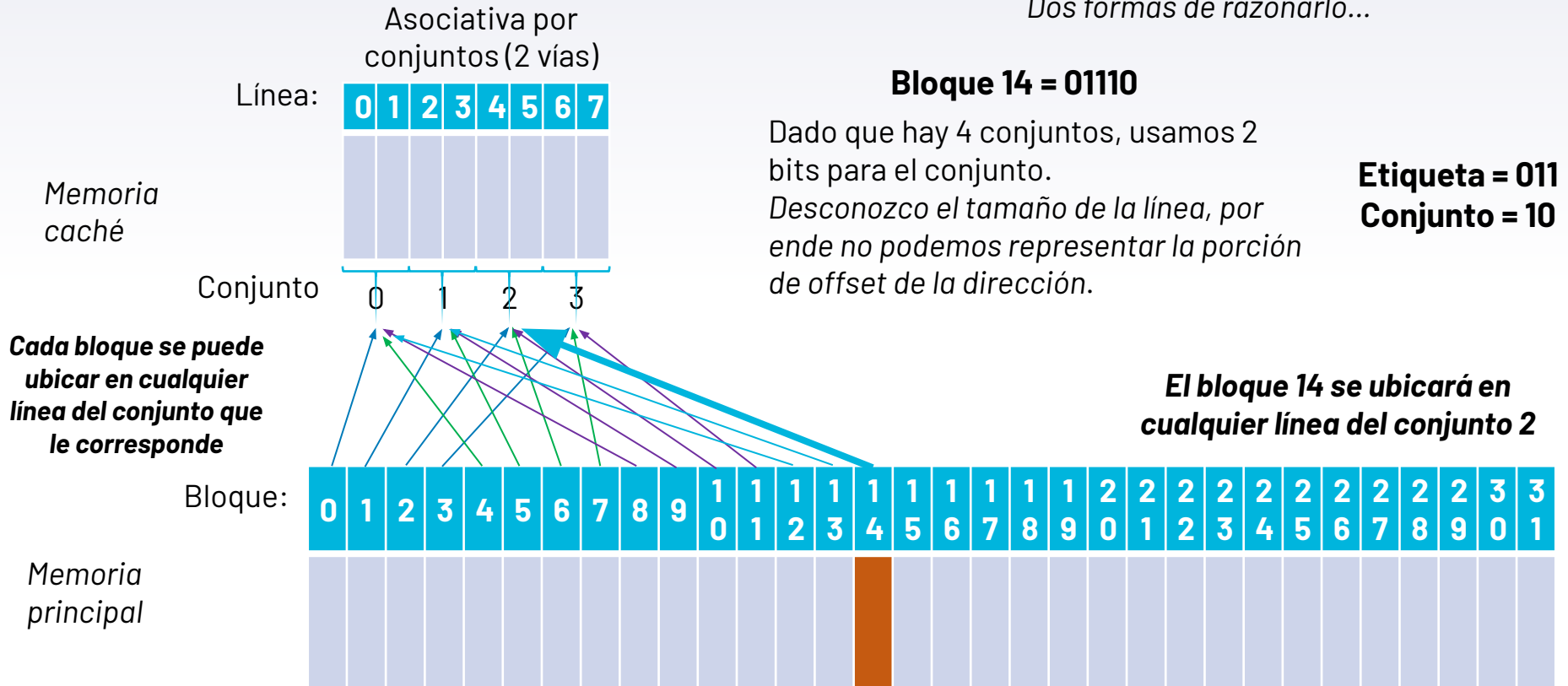
Dos formas de razonarlo...



Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

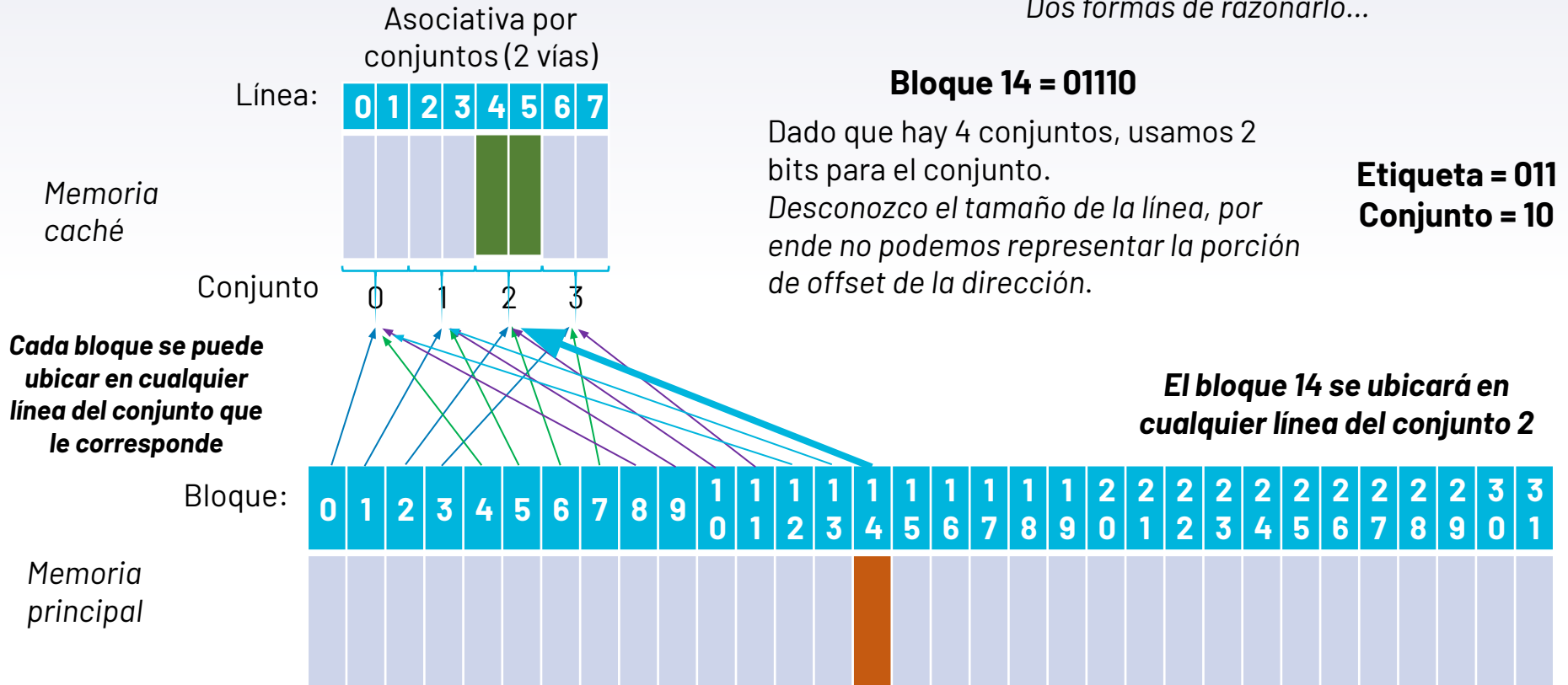
Dos formas de razonarlo...



Comparativa: ¿Dónde puede ubicarse el bloque 14 en cada organización de MC?

Ejemplo: la MC tiene 8 líneas y la MP tiene 32 bloques.

Dos formas de razonarlo...



Tiempo de acceso

$$\text{Tasa de aciertos (HR)} = \frac{\text{cantidad de aciertos}}{\text{cantidad de accesos a memoria}}$$

$$\text{Tasa de fallas (MR)} = \frac{\text{cantidad de fallas}}{\text{cantidad de accesos a memoria}}$$

$$HR + MR = 1 \text{ (Miss Rate y Hit Rate son complementarios)}$$

$$\text{Tiempo de acceso medio } (t_{acc}) [\text{seg}] = HR \times t_{acc} MC + MR \times \text{Tiempo de penalización}$$

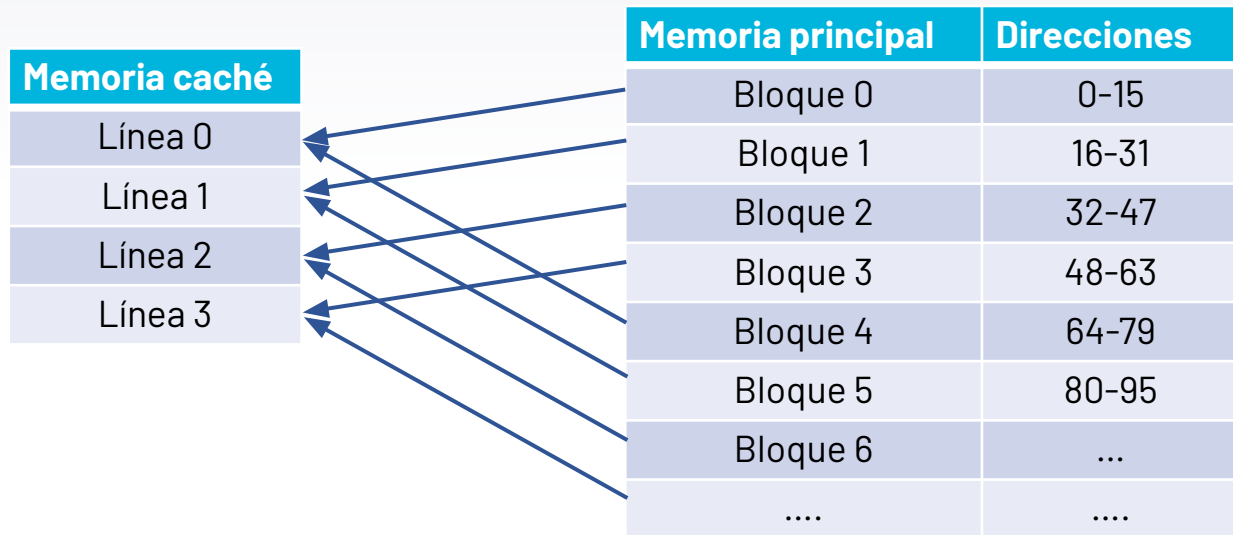
$$\text{Tiempo de penalización} [\text{seg}] = t_{acc} MC + (t_{acc} MP \times \text{Tamaño de bloque})$$

$$t_{acc} = HR \times t_{acc} MC + MR \times [t_{acc} MC + (t_{acc} MP \times \text{Tamaño de bloque})]$$

**Cuando se produce una falla debe cargarse un bloque entero de MP en MC.
El tamaño de bloque se expresa en cantidad de palabras que se leen por acceso a MP.**

Tiempo de acceso: ejemplo

Contamos con una MC de un solo nivel, cuyo tiempo de acceso es 3ns y una MP de tiempo de acceso de 60ns. En cada acceso a MP se lee una palabra. La MC está organizada con mapeo directo. No hay escrituras. Aplica política de carga *Load through*.



El programa ejecuta desde la dirección 48 a la 95 una vez. Luego ejecuta desde la dirección 15 a 31 diez veces en un bucle, y finaliza.
¿Cuánto demora su ejecución?

$$\text{Tiempo de acceso medio} = \text{Tiempo de acceso total} / \text{Cantidad total de palabras leídas por CPU}$$

Tiempo de acceso: ejemplo (respuesta)

*Se podría sumar 3ns x 5 fallas al tiempo de acceso final.
Corresponde al tiempo de buscar en MC en cada falla.*

Evento	Ubicación	T	Comentarios	Línea 0	Línea 1	Línea 2	Línea 3
1 falla	48	960 ns	16 lecturas x 60 ns. Bloque 3 -> Línea 3				B3
15 aciertos	49-63	45 ns	15 aciertos x 3 ns de Tiempo de acceso a MC				B3
1 falla	64	960 ns	16 lecturas x 60 ns. Bloque 4 -> Línea 0	B4			B3
15 aciertos	65-79	45 ns		B4			B3
1 falla	80	960 ns	16 lecturas x 60 ns. Bloque 5 -> Línea 1	B4	B5		B3
15 aciertos	81-95	45 ns		B4	B5		B3
1 falla	15	960 ns	16 lecturas x 60 ns. Bloque 0 -> Línea 0	B0	B5		B3
1 falla	16	960 ns	16 lecturas x 60 ns. Bloque 1 -> Línea 1	B0	B1		B3
15 aciertos	17-31	45 ns		B0	B1		B3
9 aciertos	15	27 ns	Últimas iteraciones del bucle	B0	B1		B3
144 aciertos	16-31	432 ns	Últimas iteraciones del bucle	B0	B1		B3
Total 5 fallas / 213 aciertos		5439 ns	Tiempo promedio de acceso = 5439 ns / 218 accesos = 24,9 ns				

¿Cuál es el HIT RATE? ¿Qué impacto tiene la MC en la performance de este sistema? ¿Cómo podría mejorarse?

Tiempo de acceso en estructura multinivel

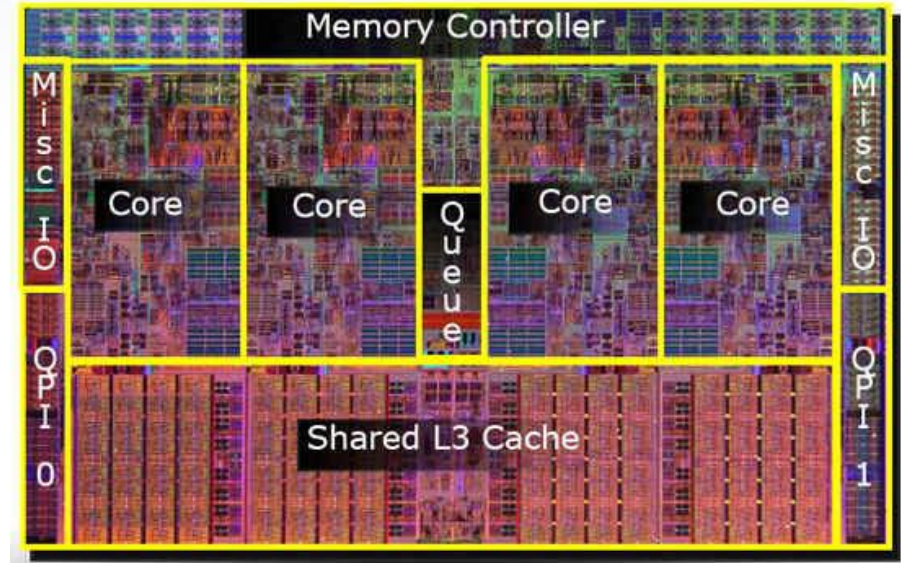
Supongamos una estructura de dos niveles de MC (L1 y L2) de tipo unificada (datos e instrucciones comparten cada nivel). Los fallos de L1 se buscarán en L2. Los fallos de L2 se buscarán en MP.

$$\text{Tiempo de acceso medio} = T_{\text{acc L1}} + MR_{L1} \times (HR_{L2} \times T_{\text{acc L2}} + MR_{L2} \times T_{\text{acc MP}})$$

Los términos de la ecuación de tiempo de acceso medio a memoria pueden medirse en unidades de tiempo absolutas (usualmente nanosegundos) o el número de ciclos de reloj que el procesador debe esperar.

El diagrama corresponde a la línea Intel Nehalem. Los niveles L1 y L2 de caché son privados por core, mientras que la caché L3 es compartida.

QPI (Quick Path Interconnect) es el reemplazo de FSB (Front Side Bus). Permite que un core obtenga datos de la MC de otro core.



Configuración de memoria caché en algunos procesadores

Procesador	Tipo	Lanzamiento	Caché L1	Caché L2	Caché L3
IBM 360/85	Mainframe	1968	16–32 kB		
PDP-11/70	Minicomputadora	1975	1 kB		
VAX 11/780	Minicomputadora	1978	16 kB		
Intel 80486	PC	1989	8 kB		
Pentium	PC	1993	8 kB/8 kB	256–512 kB	
PowerPC 601	PC	1993	32 kB		
PowerPC G4	PC/Server	1996	32Kb/32kB	256 kB–1 MB	2 MB
Pentium 4	PC/Server	2000	8 kB/8 kB	256 kB	
CRAY MTA	Supercomputadora	2000	8 kB	2 MB	
Itanium	PC/Server	2001	16 kB/16 kB	96 kB	4 MB
ICRAY XD-1	Supercomputadora	2004	64 kB/64 kB	1 MB	
Intel Core i7 EE 990	Workstation/Server	2011	6x32kB/32kB	1.5 MB	12 MB
Intel Xeon E7 v4	Server	2017	24x32 kB/32 kB	24x256 kB	60 MB

Optimizaciones a la memoria caché

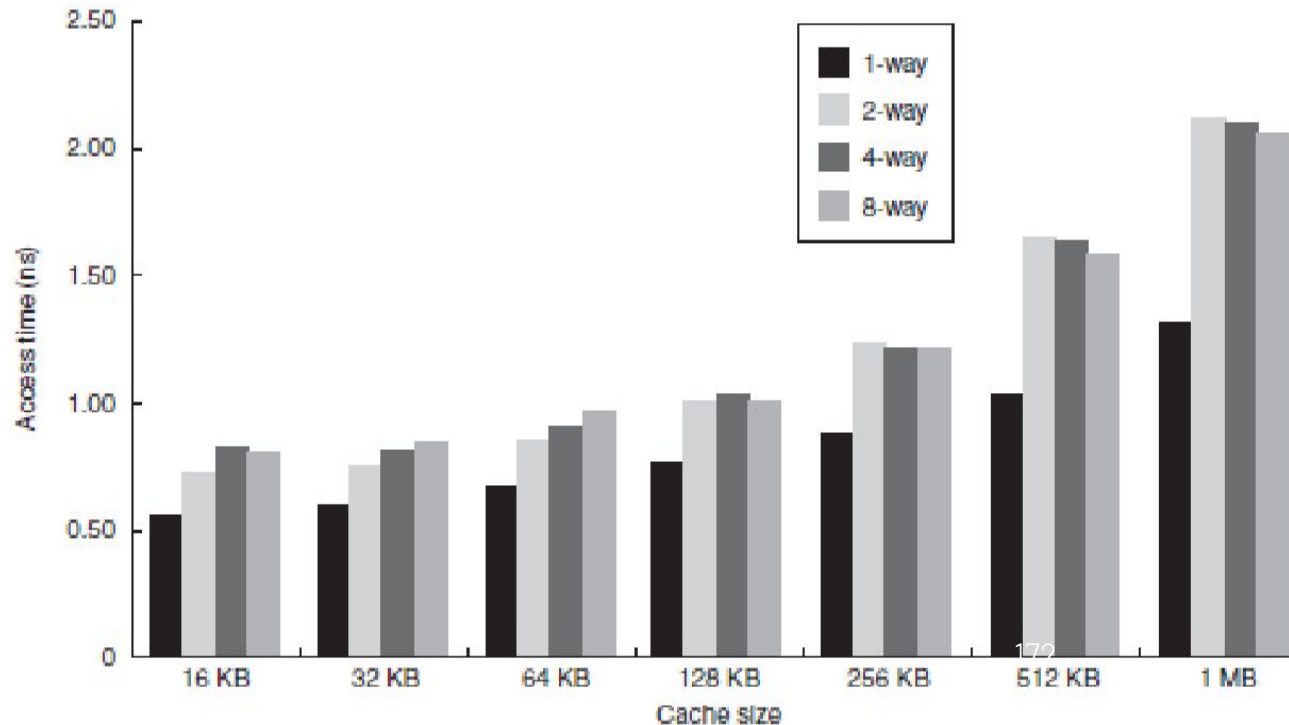
¿Por qué se producen las fallas (miss)?

- **Compulsory** (obligatorio). El primer acceso a un bloque inevitablemente generará un falla. También se les llama fallas de inicio frío o de primera referencia.
- **Capacity** (capacidad). Si la caché no puede contener todos los bloques necesarios para ejecutar un programa, estas fallas ocurrirán cuando deba descartar y luego volver a cargar bloques.
- **Conflict**. En asignación directa o asociativa por conjuntos puede ocurrir que un bloque se descarte y se vuelva a leer por haber muchos bloques necesarios para la ejecución del proceso que se mapean al mismo conjunto o línea.

¿Cómo se soluciona?

- **Utilizar organización asociativa (fully asociative)**. Pero es costosa y aumenta el tiempo de acierto (hit time).
- **Incrementar el tamaño de bloque**. Reducirá el miss rate, pero incrementará la penalización.
- **Incrementar el tamaño de caché**. Potencialmente aumenta el hit time y el consumo de energía.
- **Incrementar el grado de asociatividad**. La organización de 8 vías a los efectos prácticos logra reducir fallas igual que la totalmente asociativa.
- **Agregar niveles**: el nivel debe ser tan rápido como la CPU y los subsiguientes ser grandes para disminuir la penalización por falla.

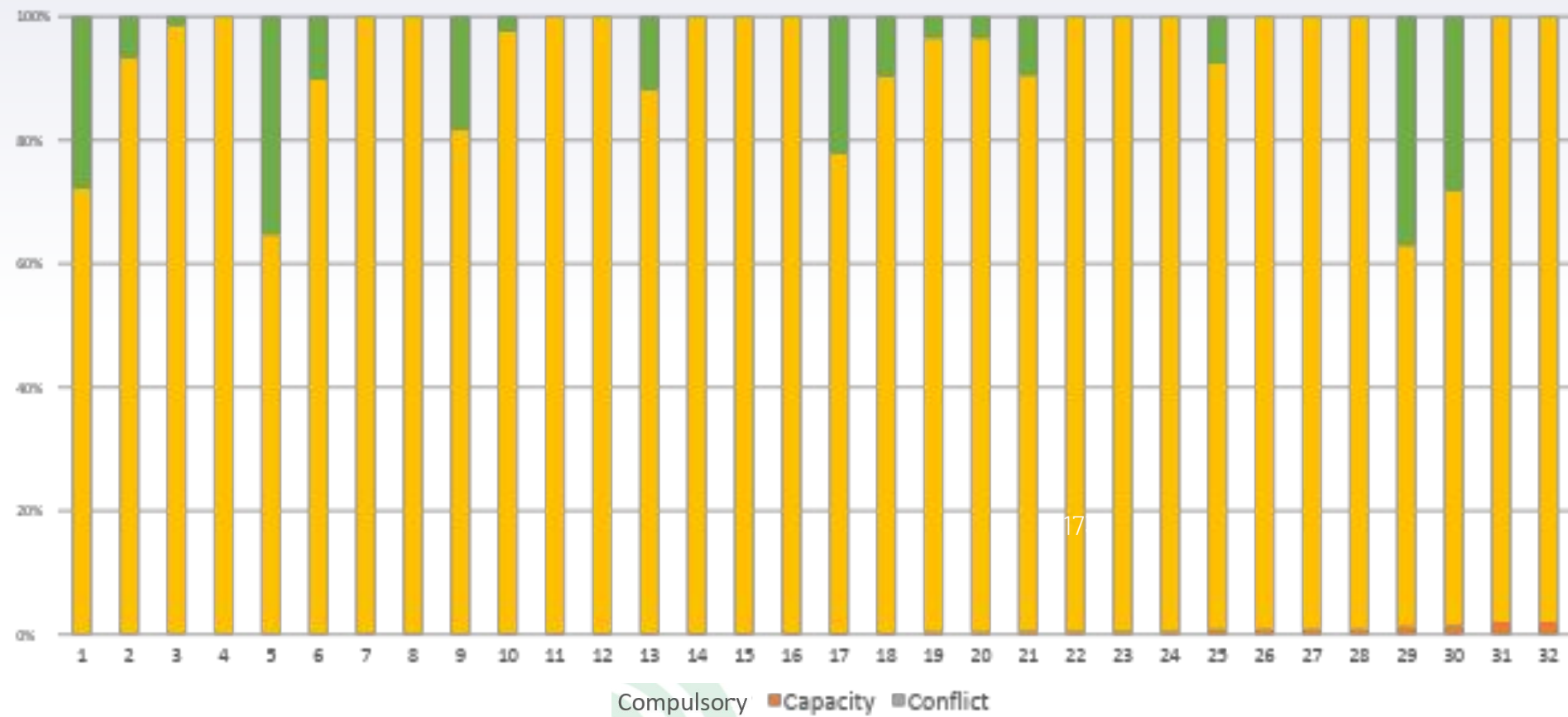
Tiempo de acceso



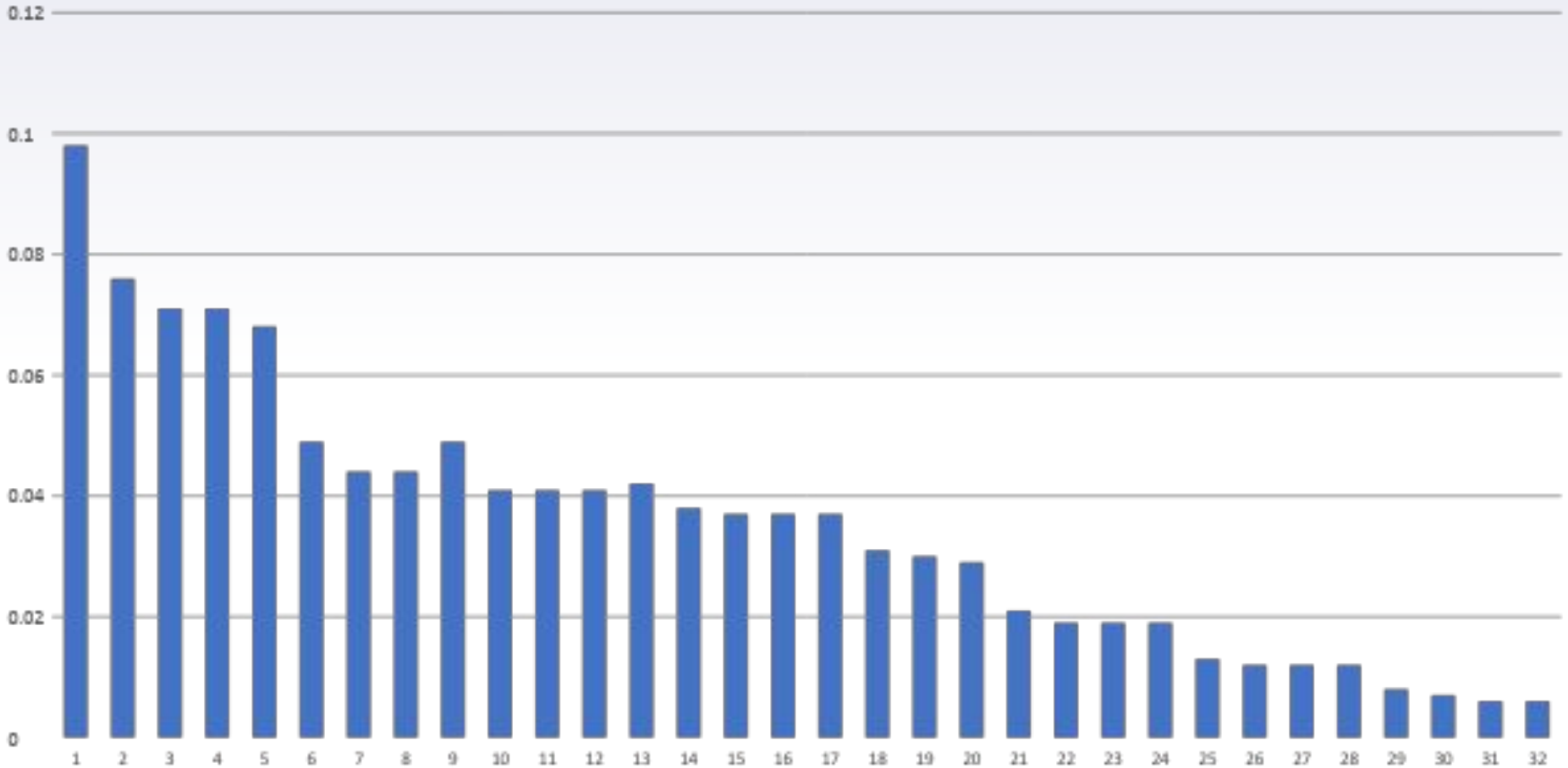
Variación del tiempo de acceso en una memoria caché CMOS de acuerdo a tamaño y grado de asociatividad.

Basado en modelo CACTI 4, asumiendo tecnología de 90nm, un solo banco y bloques de 64 bytes.

Composición de la tasa de fallos



Tasa de fallos en distintos grados de asociatividad



Composición de la tasa de fallos (parte 1)

Tamaño de caché (KB)	Asociatividad	Tasa de fallos total	Componentes de la tasa de fallos (porcentaje relativo) (suma el 100% de la tasa de fallos total)					
			Compulsory		Capacity		Conflict	
4	1 vía	0,098	0,0001	0,10%	0,07	72%	0,027	28%
4	2 vías	0,076	0,0001	0,10%	0,07	93%	0,005	7%
4	4 vías	0,071	0,0001	0,10%	0,07	99%	0,001	1%
4	8 vías	0,071	0,0001	0,10%	0,07	100%	0,000	0%
8	1 vía	0,068	0,0001	0,10%	0,044	65%	0,024	35%
8	2 vías	0,049	0,0001	0,10%	0,044	90%	0,005	10%
8	4 vías	0,044	0,0001	0,10%	0,044	99%	0,000	1%
8	8 vías	0,044	0,0001	0,10%	0,044	100%	0,000	0%
16	1 vía	0,049	0,0001	0,20%	0,04	82%	0,009	17%
16	2 vías	0,041	0,0001	0,20%	0,04	98%	0,001	2%
16	4 vías	0,041	0,0001	0,20%	0,04	99%	0,000	0%
16	8 vías	0,041	0,0001	0,20%	0,04	100%	0,000	0%
32	1 vía	0,042	0,0001	0,20%	0,037	175 89%	0,005	11%
32	2 vías	0,038	0,0001	0,20%	0,037	99%	0,000	0%
32	4 vías	0,037	0,0001	0,20%	0,037	100%	0,000	0%
32	8 vías	0,037	0,0001	0,20%	0,037	100%	0,000	0%
64	1 vía	0,037	0,0001	0,20%	0,028	77%	0,008	23%
64	2 vías	0,031	0,0001	0,20%	0,028	91%	0,003	9%
64	4 vías	0,03	0,0001	0,20%	0,028	95%	0,001	4%
64	8 vías	0,029	0,0001	0,20%	0,028	97%	0,001	2%

Composición de la tasa de fallos (parte 2)

Tamaño de caché (KB)	Asociatividad	Tasa de fallos total	Componentes de la tasa de fallos (porcentaje relativo) (suma el 100% de la tasa de fallos total)					
			Compulsory		Capacity		Conflict	
64	1 vía	0,037	0,0001	0,20%	0,028	77%	0,008	23%
64	2 vías	0,031	0,0001	0,20%	0,028	91%	0,003	9%
64	4 vías	0,03	0,0001	0,20%	0,028	95%	0,001	4%
64	8 vías	0,029	0,0001	0,20%	0,028	97%	0,001	2%
128	1 vía	0,021	0,0001	0,30%	0,019	91%	0,002	8%
128	2 vías	0,019	0,0001	0,30%	0,019	100%	0,000	0%
128	4 vías	0,019	0,0001	0,30%	0,019	100%	0,000	0%
128	8 vías	0,019	0,0001	0,30%	0,019	100%	0,000	0%
256	1 vía	0,013	0,0001	0,50%	0,012	94%	0,001	6%
256	2 vías	0,012	0,0001	0,50%	0,012	99%	0,000	0%
256	4 vías	0,012	0,0001	0,50%	0,012	99%	0,000	0%
256	8 vías	0,012	0,0001	0,50%	0,012	176 99%	0,000	0%
512	1 vía	0,008	0,0001	0,80%	0,005	66%	0,003	33%
512	2 vías	0,007	0,0001	0,90%	0,005	71%	0,002	28%
512	4 vías	0,006	0,0001	1,10%	0,005	91%	0,000	8%
512	8 vías	0,006	0,0001	1,10%	0,005	95%	0,000	4%

Conclusiones

- En la medida que los procesadores sigan mejorando sus prestaciones pero la memoria no logre acompañar ese crecimiento al mismo ritmo, la brecha se mantendrá y aumentará.
- Las técnicas avanzadas de optimización (solo expuestas en forma parcial y muy breve en este material) son un intento de disminuir dicha diferencia, pero será necesario seguir trabajando fuerte en este campo.
- Es indiscutible que los diseñadores de sistemas operativos y particularmente de compiladores pueden hacer mucho para que las memorias caché aprovechen el principio de localidad al máximo.



Bibliografía

- Computer's Architecture: A Quantitative Approach 5th Ed.
Hennessy – Patterson, Morgan Kaufmann, 2012, ISBN 978-0-12-383872-8
- Computer Organization and Architecture. Designing for performance, 9th Ed.
William Stallings, Pearson Education, 2013. ISBN 13: 978-0-13-293633-0
- MUR00 Principios de arquitectura de computadoras, 1ra Ed.
Murdocca – Heuring, Prentice Hall, 2002, ISBN 987-9460-69-3
- Microprocesadores Intel 7ma ed
Barry Brey, Prentice Hall, 2006, ISBN970-26-0804-X
- Program restructuring for virtual memory
DJ Hatfield, J Gerald – IBM Systems Journal, 1971 – ieeexplore.ieee.org
- An Introduction to the Intel QuickPath Interconnect" (PDF).
Intel Corporation. January 30, 2009. Retrieved June 14, 2011.
- Computer Architecture ELE 475 / COS 475 Slide Deck 3: Cache Review,
David Wentzlaff, Department of Electrical Engineering, Princeton University
- Intel® Xeon® Processor E7-8890 v4 Product Specification
- Memoria cache (Perez Berro – Perrone – Rodríguez, UNLAM)
- PowerPC 601 RISC Microprocessor. Technical Summary.
<https://www.nxp.com/docs/en/data-sheet/MPC601.pdf>