Estructura de Computadores

Grado de Ingeniería Informática

ETSINF

Tema 6: Jerarquía de memoria

DISCA

Objetivos

- Comprender el concepto de jerarquía de memoria y la razón que la justifica
- Conocer las características operacionales y parámetros de diseño de la memoria cache y comprender cómo estos influyen en las prestaciones del sistema de memoria
- Conocer el concepto de memoria virtual y su soporte por parte de la arquitectura del procesador

Contenido

1. Concepto de jerarquía de memoria
2. La memoria cache
   - ✓ Conceptos básicos
   - ✓ Correspondencias y políticas
   - ✓ Aspectos de rendimiento
   - ✓ Ejemplos de procesadores comerciales
3. La memoria virtual
   - ✓ Concepto y utilidad
   - ✓ Direccionamiento virtual
   - ✓ Traducción de direcciones
4. Ejemplo conjunto TLB y cache
   - ✓ El MIPS R2000 en la DECstation 3100

Bibliografía

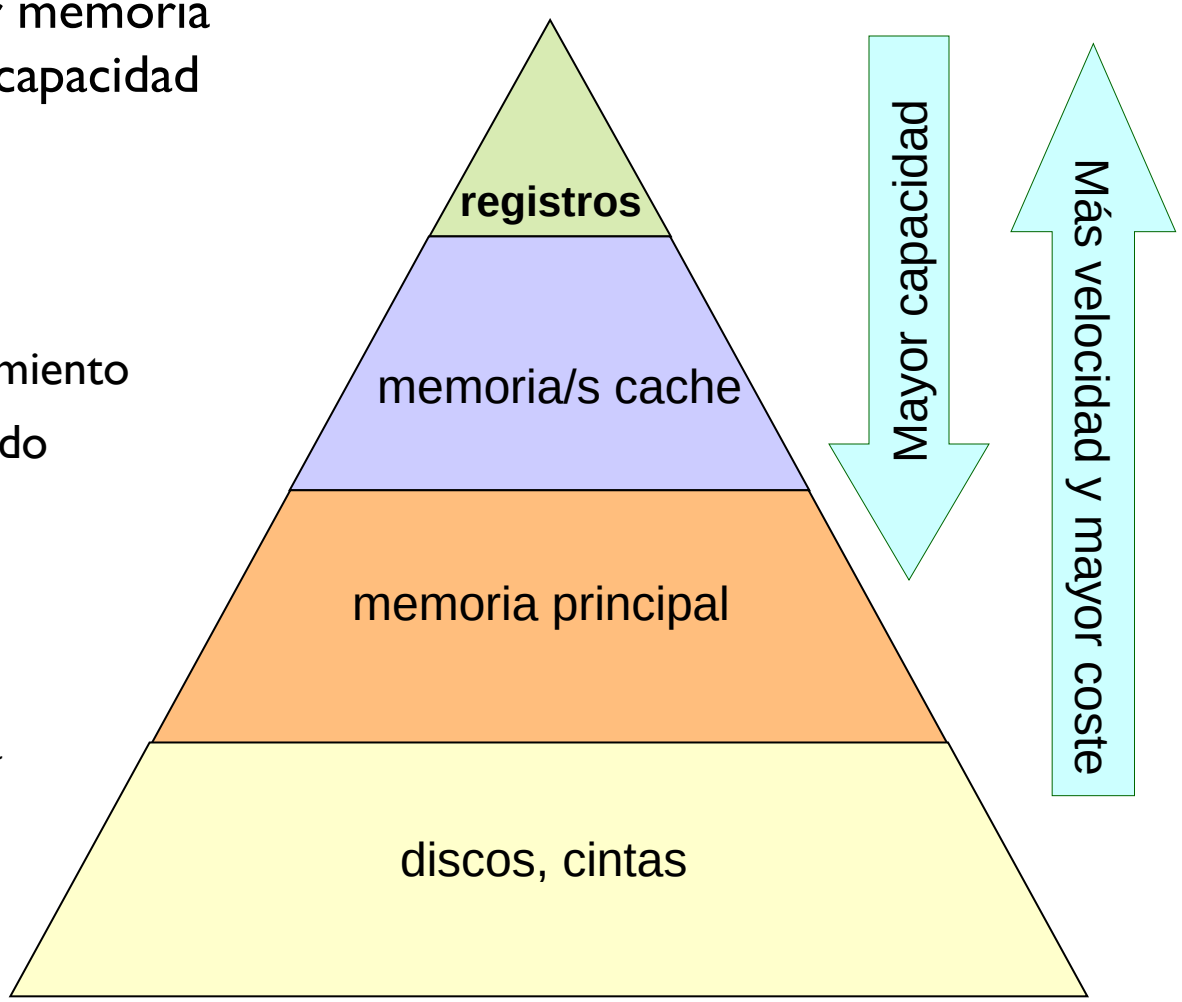
- Patterson, D.A., Hennessy, J.L.
  - ✓ Estructura y diseño de computadores. Interficie circuitería/programación. Reverté, 2000
  - ✓ Cap.5 (5.1, 5.2, 5.3, 5.4, 5.5)
- Stallings, W.
  - ✓ Organización y arquitectura de computadores. 7ª edición. Prentice Hall, 2006
  - ✓ Cap.4 (4.2)
- Hamacher, V.C., Vranesic, Z.G., Zaky, S.G.
  - ✓ Organización de computadores, 5ª edición. McGraw Hill, 2003
  - ✓ Cap.5 (5.5, 5.6, 5.7)

1. Concepto de jerarquía de memoria

- ¿Qué se entiende por jerarquía de memoria?
- Parámetros de la jerarquía
- El principio de localidad

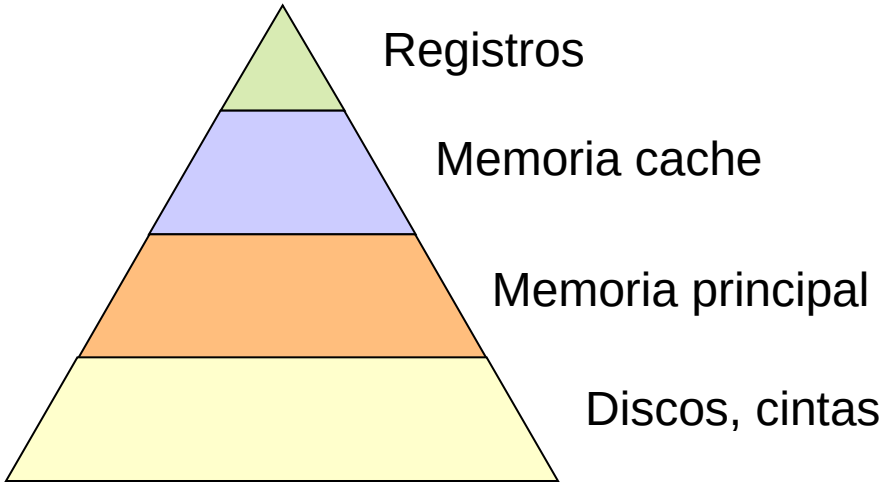
Jerarquía de memoria: ¿qué es?

- Situación ideal: disponer memoria rápida y barata de gran capacidad
- Criterios tecnológicos contrapuestos
  - ✓ Velocidad de acceso
  - ✓ Capacidad de almacenamiento
  - ✓ Coste por bit almacenado
  - ✓ Consumo de potencia
  - ✓ Fiabilidad
- Solución
  - ✓ Organización jerárquica



Parámetros jerarquía de memoria

- Modo de acceso, tiempo de acceso y capacidad
  - ✓ Registros — Aleatorio, < 0.1ns, pocos bytes
  - ✓ Memoria cache — Aleatorio, 0.1 – 10 ns, 8 KB – 32 MB
  - ✓ Memoria principal — Aleatorio, 10 – 100 ns, 256 MB – 64 GB
  - ✓ Discos — Directo, 10 ms, 100 – 1000 GB
  - ✓ Cintas — Secuencial, minutos, GB – TB



¿Por qué es eficiente la jerarquía?

- Principio de localidad de la información (datos e instrucciones)
  - ✓ **Localidad temporal**
    - Tendencia a volver a referenciar el mismo dato o instrucción
  - ✓ **Localidad espacial**
    - Tendencia a referenciar palabras en direcciones próximas
    - Este principio justifica la organización en bloques o páginas que actúan como unidad de transferencia
- Es un **principio empírico**, derivado del diseño y comportamiento de los programas
  - ✓ También se indica como la regla "20-80": el 20% del código es el responsable del 80% del tiempo de ejecución

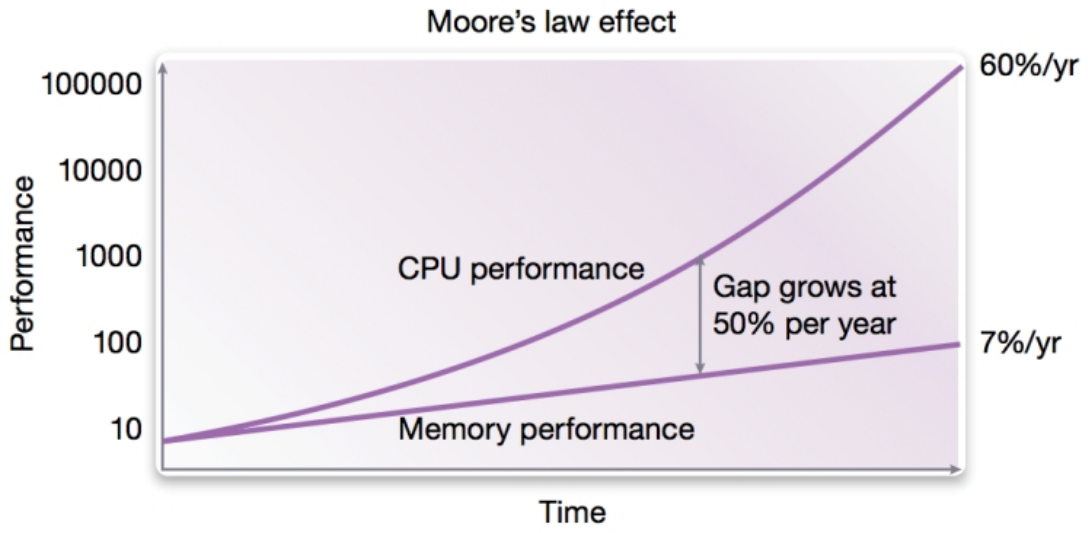
2. La memoria cache

- Conceptos básicos
- Correspondencias y políticas
- Aspectos de rendimiento

Rendimiento UCP-memoria (*performance gap*)

- Aumento del rendimiento
  - ✓ Procesador: 60% por año
  - ✓ Memoria (DRAM): 7% por año
- Por tanto, la diferencia aumenta exponencialmente

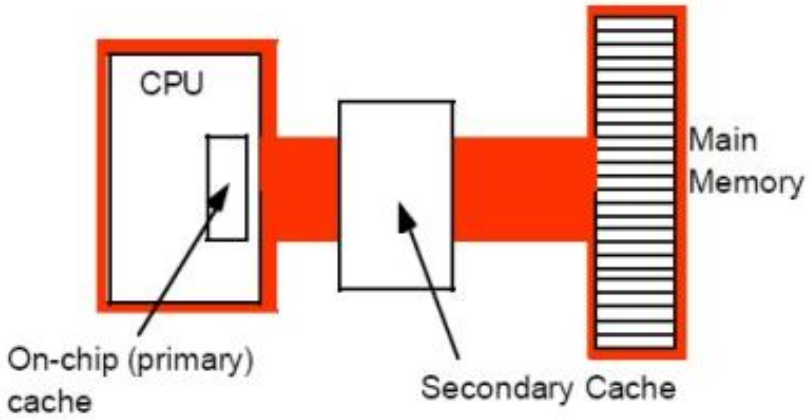
Intel Xeon (hasta 3,8 GHz)
T_{clock} = *0,263 ns*
≈ x144
DDR3-1600
Latencia (t_{RCD}+t_{CL}) ≈ *37,5 ns*



¿Cómo ocultar esta diferencia? Establecer uno o varios niveles de memoria cache (SRAM)

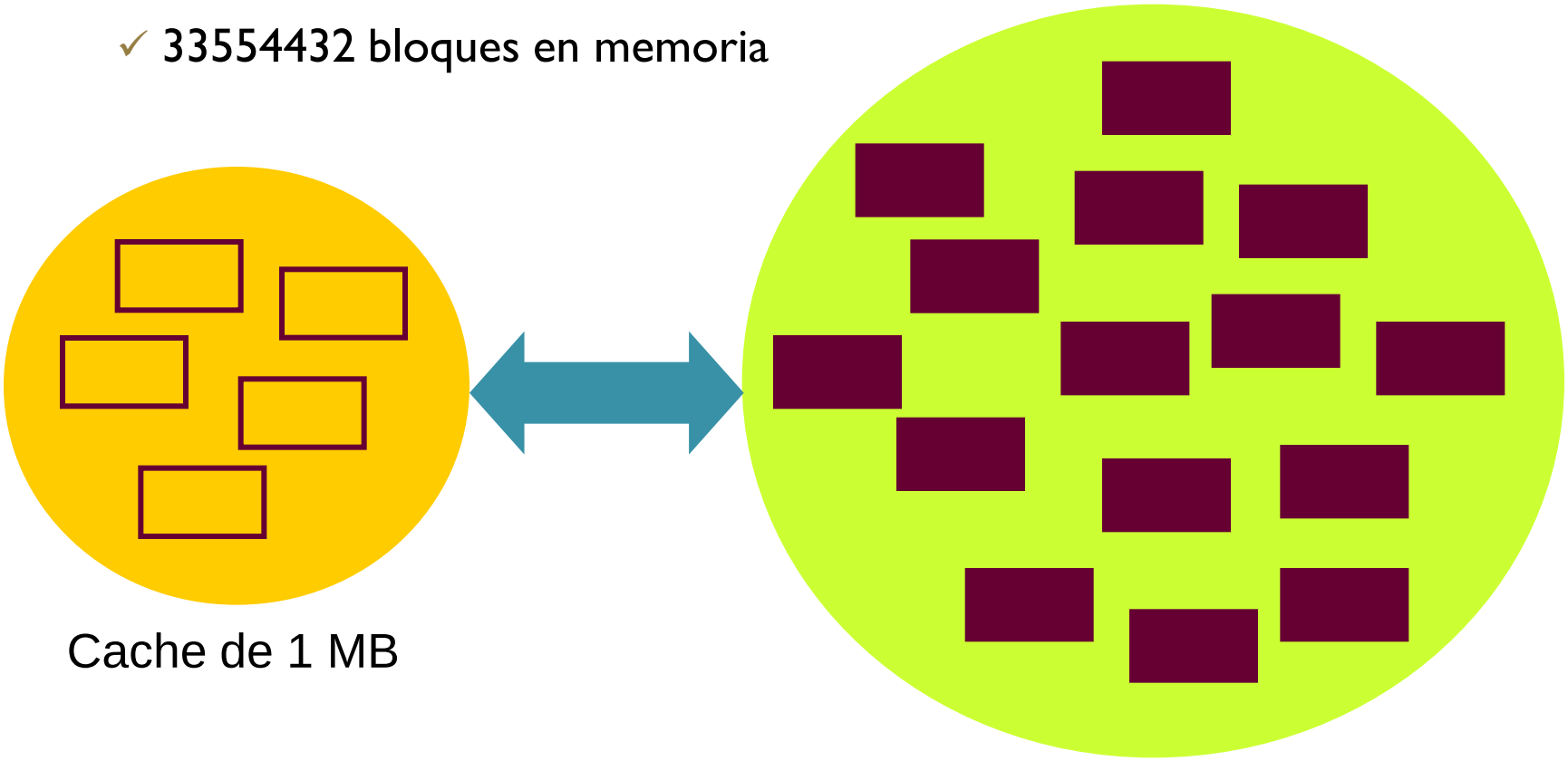
¿Qué es la memoria cache?

- Memoria rápida y de "poca capacidad" situada entre la memoria principal y el procesador
- Objetivo: reducir el tiempo de acceso a la información
  - ✓ Acceso a código (búsqueda de instrucciones)
  - ✓ Acceso a datos: `lw, lh, lb, sw, sh, sb`
- Funcionamiento
  - ✓ Primero se busca la información en la memoria cache (tecnología SRAM). Si no se encuentra aquí se accede a la memoria principal (tecnología DRAM)
    - La unidad de transferencia entre la cache y la memoria principal es el **BLOQUE** (conjunto de 4 u 8 palabras)



Relación entre memoria principal y cache

- Bloques de 4 palabras de 32 bits
  - ✓ 65536 bloques en la cache
  - ✓ 33554432 bloques en memoria

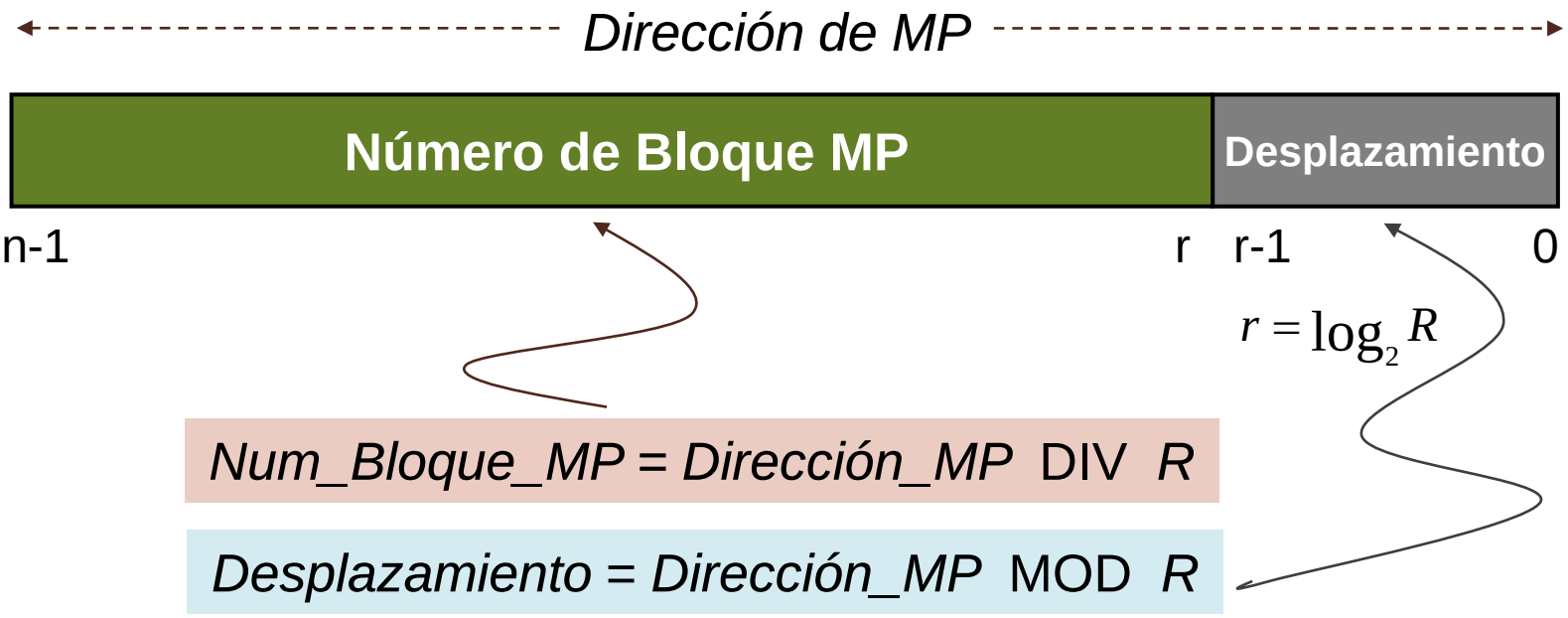


Cálculo del Número de Bloque

Espacio de direccionamiento de MP: 2^n bytes

Tamaño de bloque: R= 2^r bytes

$$Num\_Total\_Bloques\_MP = \frac{Tamaño\_espacio\_dir}{Tamaño\_Bloque}$$

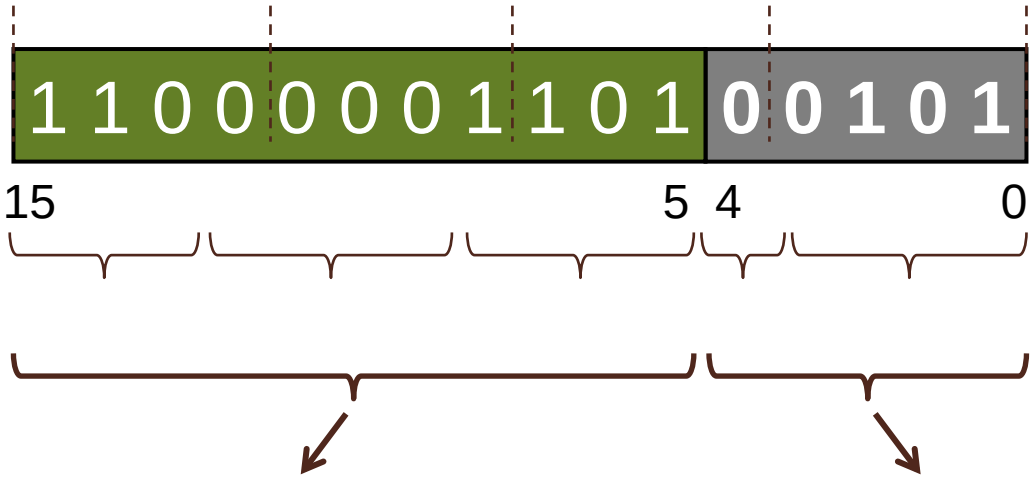


Cálculo del Número de Bloque

Espacio de direccionamiento de MP: 64 KB

Tamaño de bloque: 32 bytes

Dirección_MP: 0×C1A5



Num_Bloque_MP = 0×60D

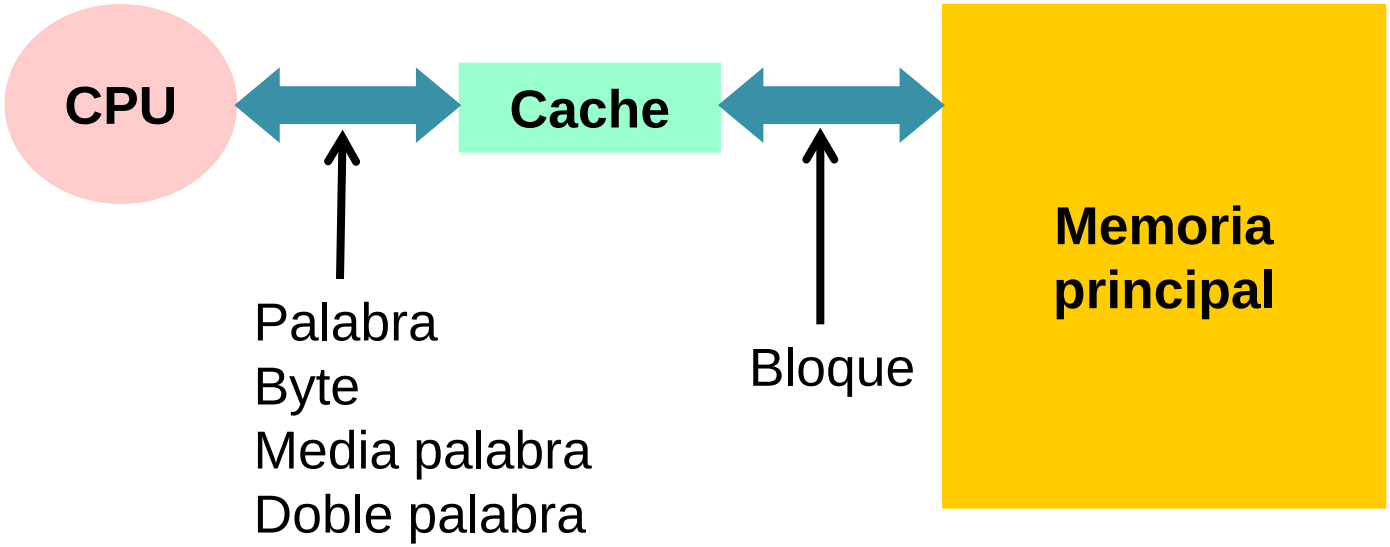
Desplazamiento = 0×05

Clasificación de los accesos

- Al hacer un acceso, la CPU puede encontrar o no la información en la memoria cache
- **Acierto** (*hit*): la información se encuentra en la cache
  - ✓ Tiempo de acierto: tiempo de acceso a la cache
- **Fallo** (*miss*): la información no se encuentra en la cache
  - ✓ Se ha de acceder a los niveles inferiores de la jerarquía de memoria
  - ✓ Tiempo en resolver el fallo: latencia de penalización

¿Cómo explota la cache la localidad espacial?

- Cuando se produce un fallo en la cache, se trae un bloque desde memoria principal
- Línea: espacio de la cache donde se copia el bloque procedente de memoria
  - ✓ El tamaño de la línea coincide con el del bloque, y se utilizan indistintamente los términos línea y bloque

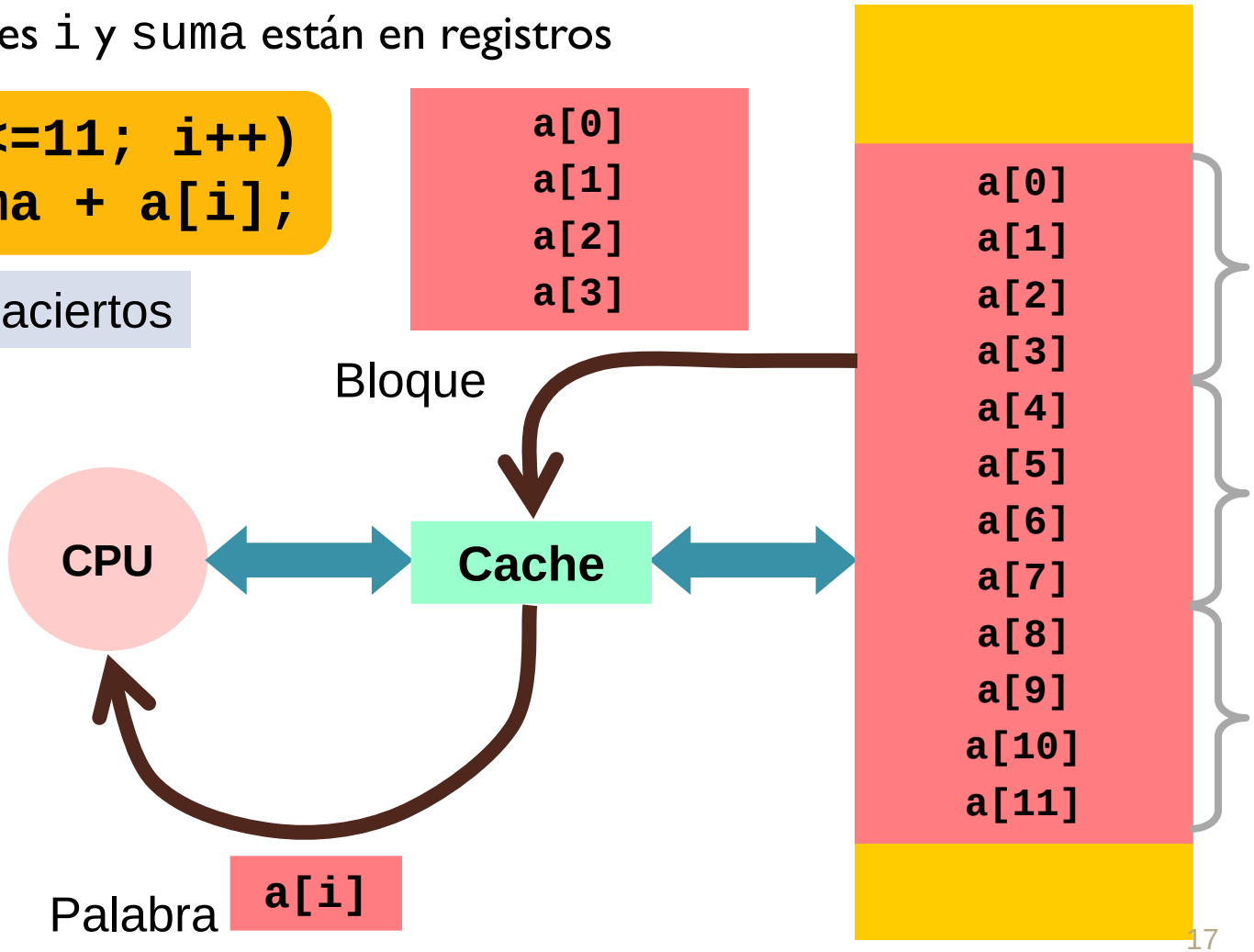


Localidad espacial en acceso a datos

- Lectura del vector `a[]` de 12 enteros de 32 bits
  - ✓ Las variables `i` y `suma` están en registros

```
for (i=0; i<=11; i++)
  suma = suma + a[i];
```

Hay 3 fallos y 9 aciertos



Tasa de aciertos y tiempo de acceso a datos

- Tasa de aciertos (H): proporción de aciertos

$$H = \frac{\text{Número de aciertos}}{\text{Número de accesos}} = \frac{\text{Número de accesos - Número de fallos}}{\text{Número de accesos}}$$

- Tasa de fallos: 1-H (proporción de fallos)

- Tiempo medio de acceso a los datos

$$T_m = H \times T_{acierto} + (1 - H) \times T_{fallo}$$

Ejemplo de cálculo

- Procesador a 1 GHz
  - ✓ Tiempo de acceso a la cache: tiempo de ciclo del procesador
  - ✓ H (en promedio): 95%
  - ✓ Resolución del fallo por la memoria principal: 40 ns (t_{RCD}+t_{CL})
- Cálculo del tiempo medio de acceso a los datos

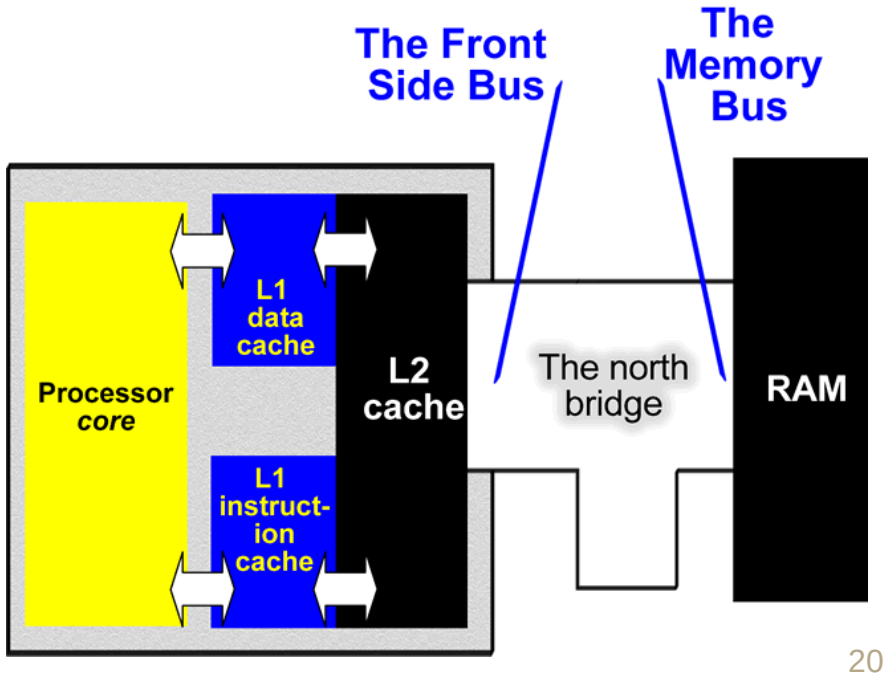
$$T_{ciclo} = \frac{1}{1\,\text{GHz}} = \frac{1}{10^9\ \text{s}^{-1}} = 1\,\text{ns}$$

$$T_m = H \times T_{acierto} + (1 - H) \times T_{fallo}$$
$$= 0.95 \times 1 + 0.05 \times 40 = 2{,}95\ \text{ns}$$

El objetivo final es que T_m sea lo más cercano posible al tiempo de acceso a la cache

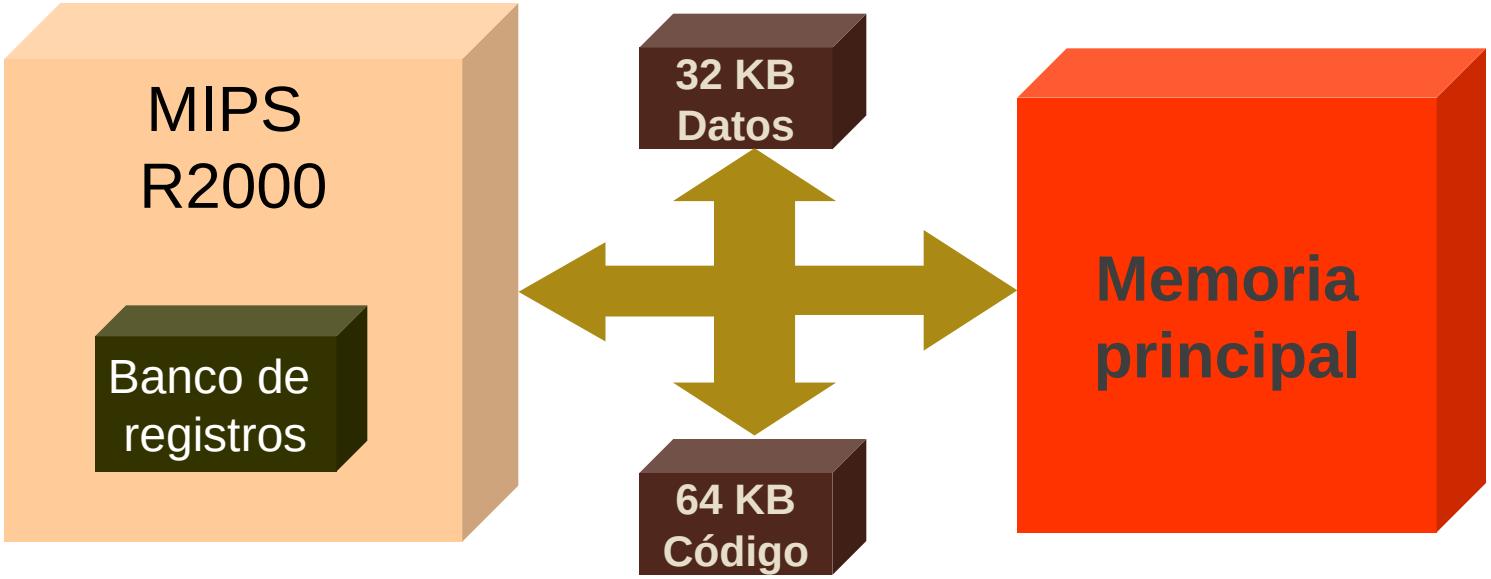
Caches unificadas y duales

- Unificadas: contienen datos e instrucciones
- Duales (*split caches*) o arquitectura Harvard
  - ✓ Hay una memoria cache para almacenar únicamente datos y otra para instrucciones en el mismo nivel
  - ✓ Se reduce la complejidad hardware y permite el acceso en paralelo a las dos caches
- En la práctica
  - ✓ L1 suele ser dual
  - ✓ L2 suele ser unificada



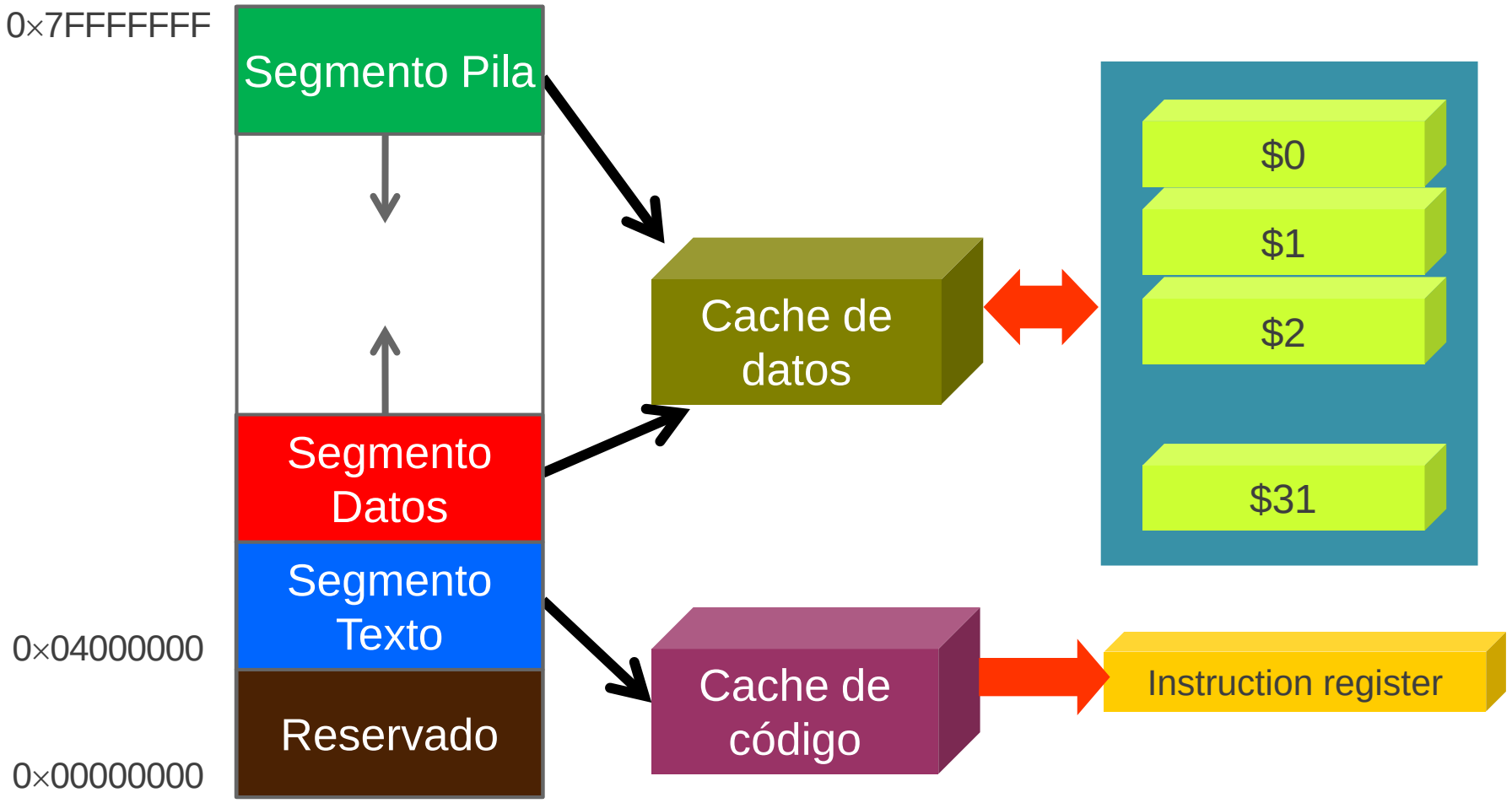
La memoria cache en el MIPS R2000 (1985)

- Solamente hay un nivel de cache (L1)
- Externa al procesador (*off chip*)
  - ✓ Cache de datos desde 4 a 64 KB (*D-cache*)
  - ✓ Cache de código desde 4 a 64 KB (*I-cache*)



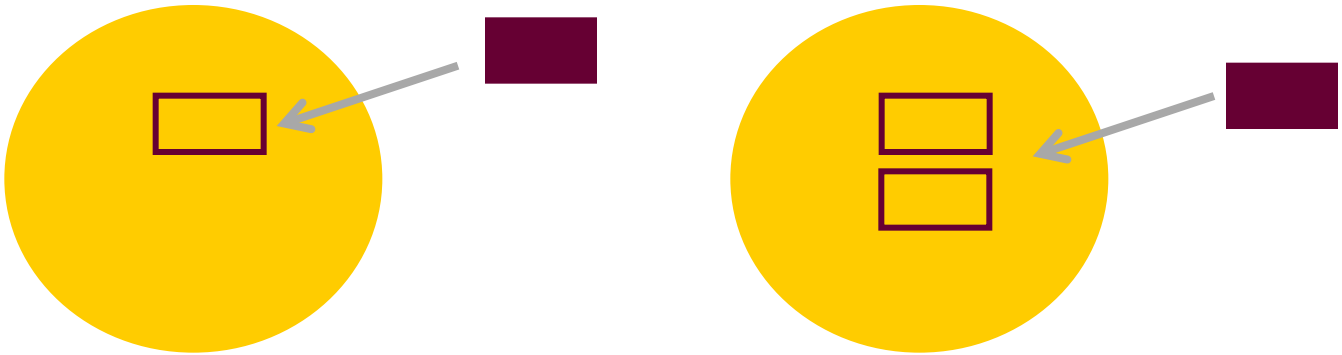
Flujo de información entre cache y registros

- La unidad de control se encarga de la gestión



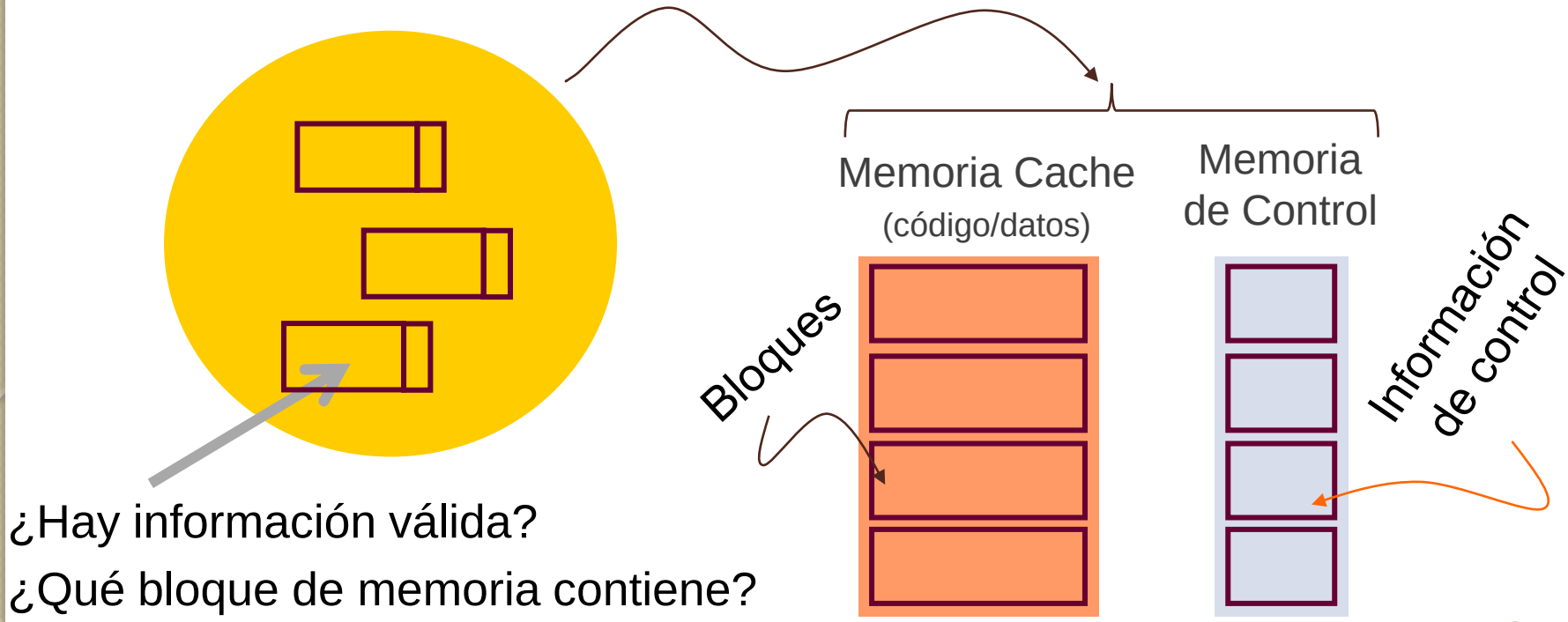
Estrategias de ubicación de información

- Cuestiones que debe resolver el sistema de cache
  - ✓ Dónde se almacena un bloque cuando se trae a la cache
  - ✓ Cómo se localiza un dato almacenado en la cache
- Para llevar a cabo ambas funciones se aplica una función de correspondencia o mapeo
- Estrategias
  - ✓ Un bloque siempre se ubicará en la misma línea
  - ✓ El bloque se puede ubicar en una de un conjunto de líneas (hay posibilidad de elección)



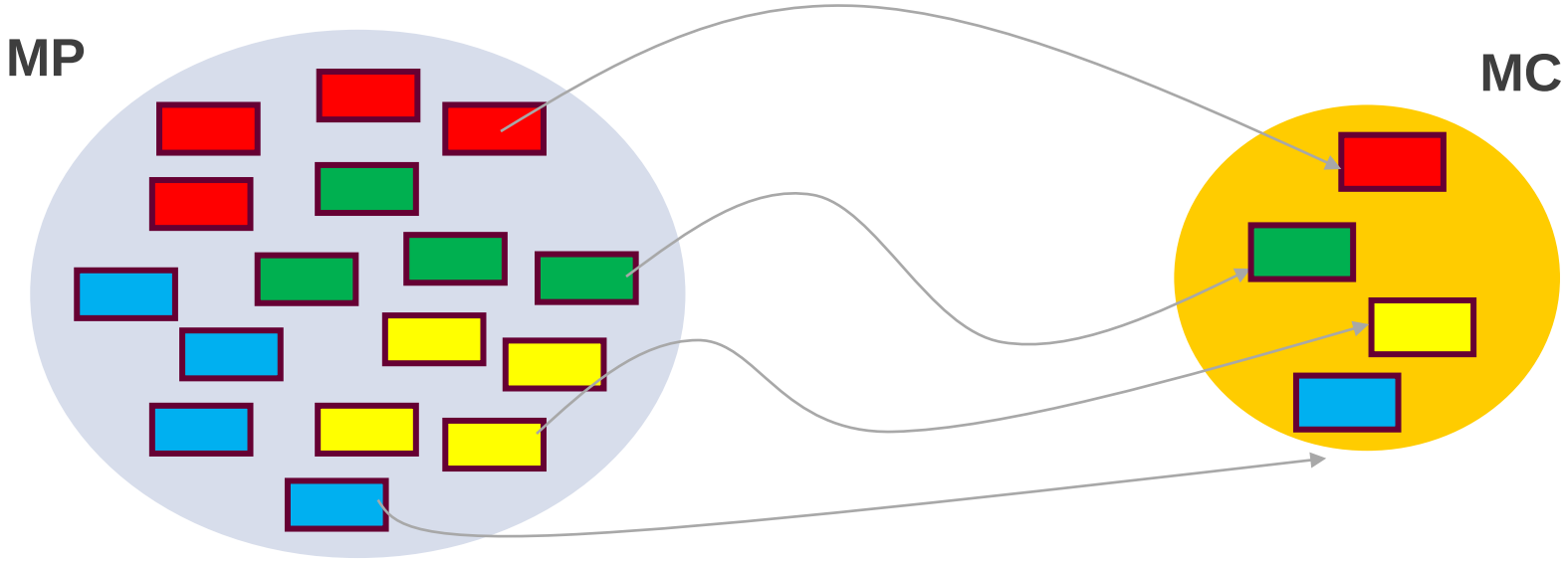
Información de control de la cache

- Información almacenada en la cache
  - ✓ Bloques de la memoria principal (código y datos)
  - ✓ Bits de control para la gestión de cada línea
    - Bit de válido: indica si la línea contiene un bloque o no
    - Etiqueta: identifica el bloque de memoria contenido en ella



Funciones de correspondencia (*mapping*)

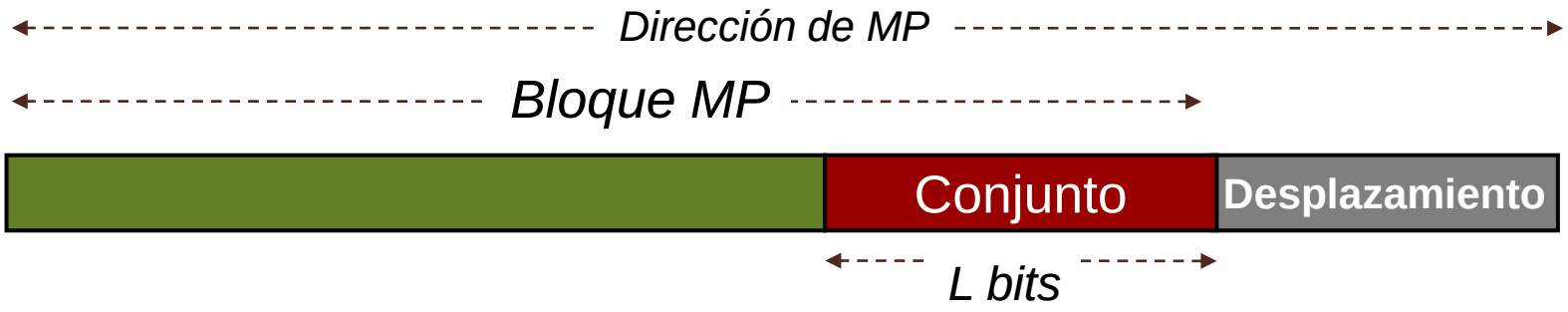
- **Objetivo**: decidir en qué línea de cache debe almacenarse un cierto bloque de MP
- Para ello, se realiza un proceso de **clasificación**:
  - Definición de cierto número de clases o recipientes en la MC
  - Agrupación de los bloque de MP por clases
  - Los bloques de MP pertenecientes a la clase (i) se hacen corresponder con el recipiente (i) de la MC



Funciones de correspondencia (*mapping*)

- A las clases o recipientes de la MC se les denomina CONJUNTOS
- El número de conjuntos de la MC es siempre potencia de 2
- Un conjunto está constituido por una o más líneas (potencias de 2)
- Los bloques de MP se clasifican en tantas clases como número de conjuntos hay en la MC

Bloque_MP $\rightarrow$ Conjunto_MC

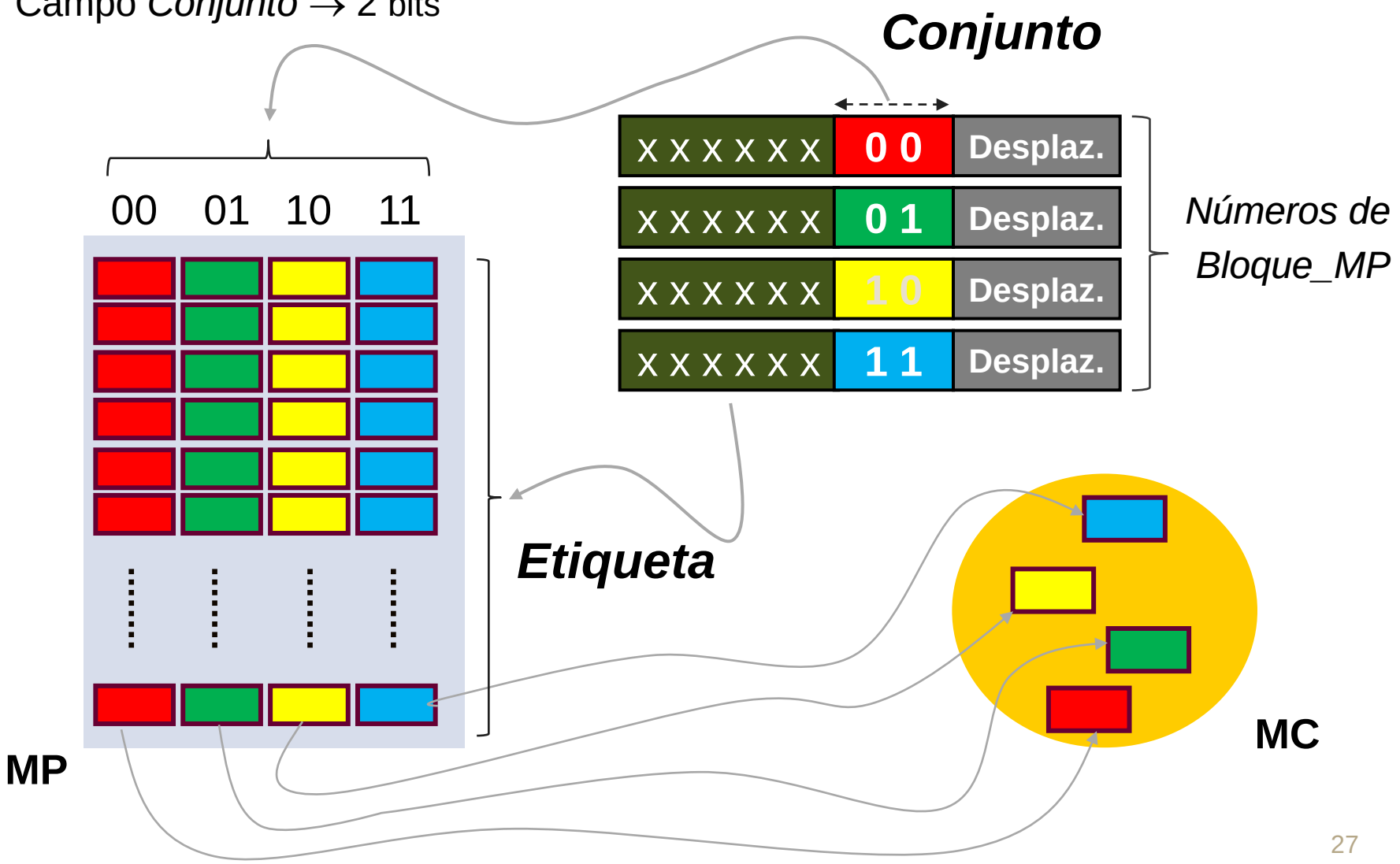


Conjunto= Bloque_MP MOD *Número_Conjuntos_MC*

Número de Conjuntos MC = 2^L

Funciones de correspondencia (*mapping*)

Número de Conjuntos MC = 4
Campo *Conjunto* → 2 bits

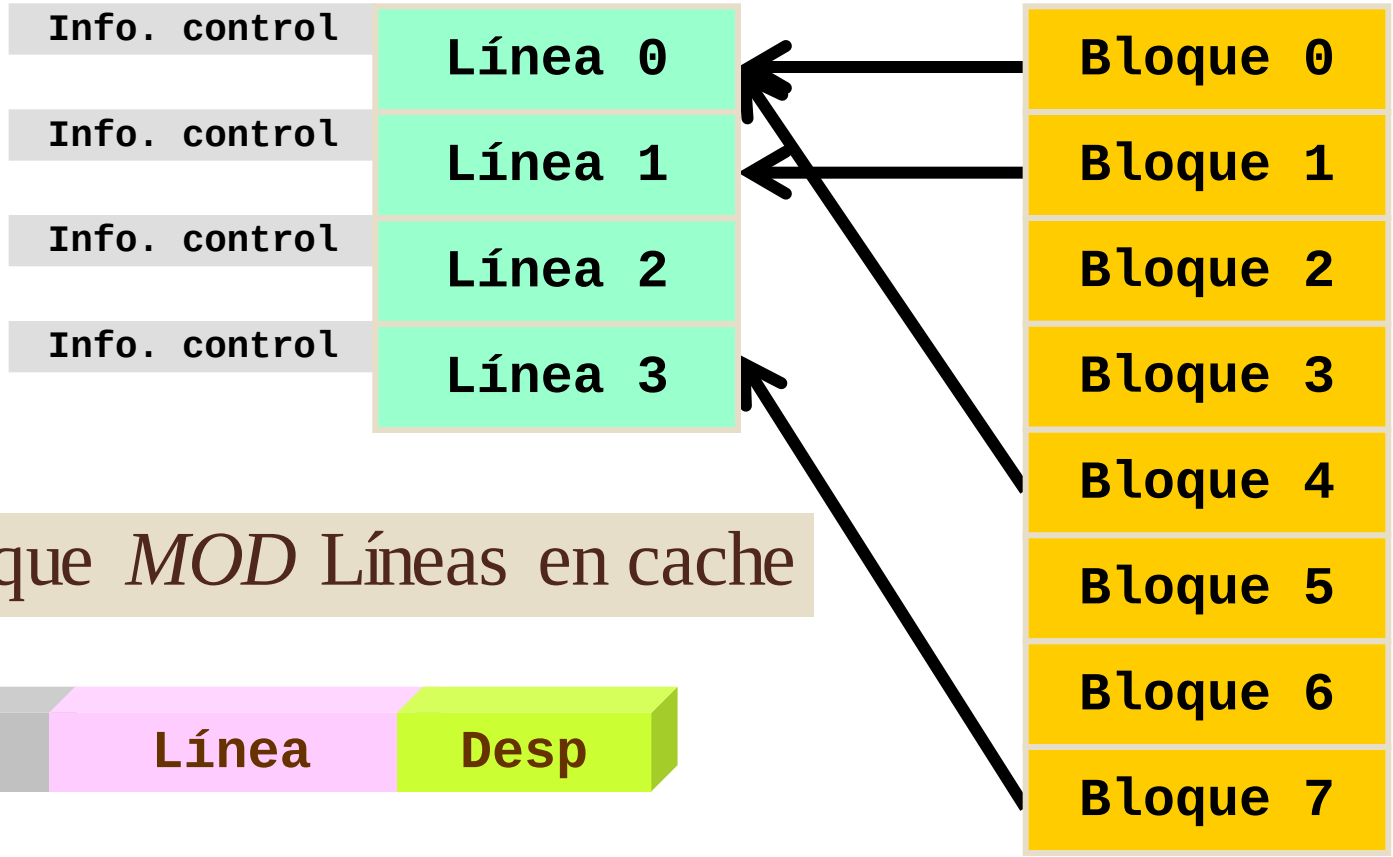


Funciones de correspondencia (*mapping*)

- **Directa**
  - ✓ Un bloque se almacena siempre en la misma línea (sólo hay una línea en cada conjunto)
- **Asociativa por conjuntos de *n* vías**
  - ✓ Un bloque se almacena siempre en cualquiera de las *n* líneas (vías) contenidas en el conjunto
- **Totalmente asociativa**
  - ✓ Un bloque se puede almacenar en cualquier línea de la memoria cache (sólo hay un conjunto)

- En la práctica la función equivale a seleccionar aquellos bits de la dirección que identifican el conjunto
- La dirección es interpretada por la cache según su organización interna

Aproximación: directa



$$\text{Línea} = \text{Bloque } \mathit{MOD} \text{ Líneas en cache}$$

Un bloque se ubica en una única línea

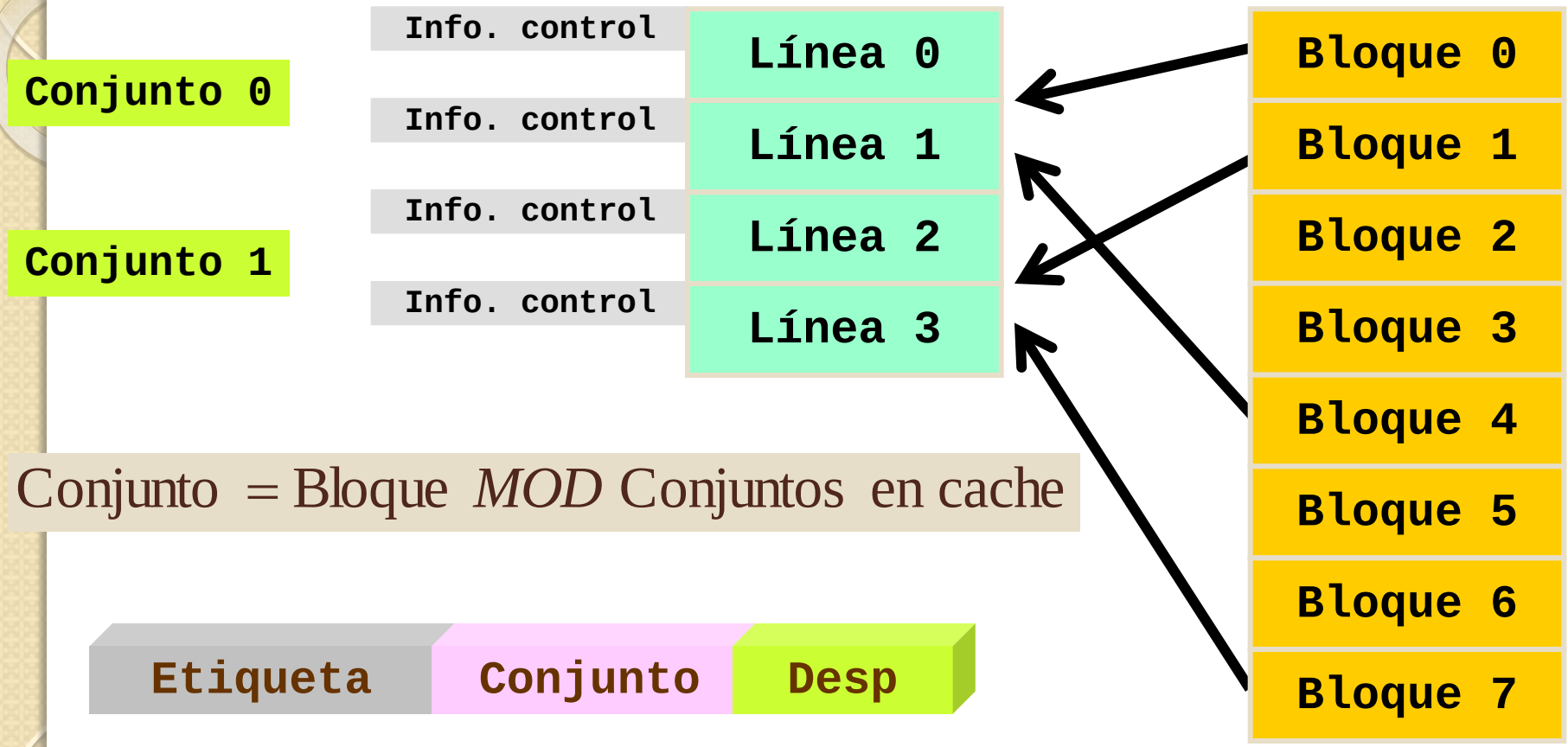
La línea 0 puede albergar los bloques 0, 2, 4 y 6 (aunque no de forma simultánea)

Correspondencia asociativa

- Inconvenientes de la correspondencia directa: baja tasa de aciertos y gran longitud de las etiquetes
- En la práctica se suele incluir cierto grado de asociatividad
- Caches de *n* vías
  - ✓ Las líneas se agrupan en conjuntos de tamaño *n*
  - ✓ Un bloque tiene *n* ubicaciones diferentes dentro del conjunto que le toca

$$\text{Número de conjuntos} = \frac{\text{Líneas en la cache}}{\text{Número de vías}}$$

Aproximación: asociativa de 2 vías

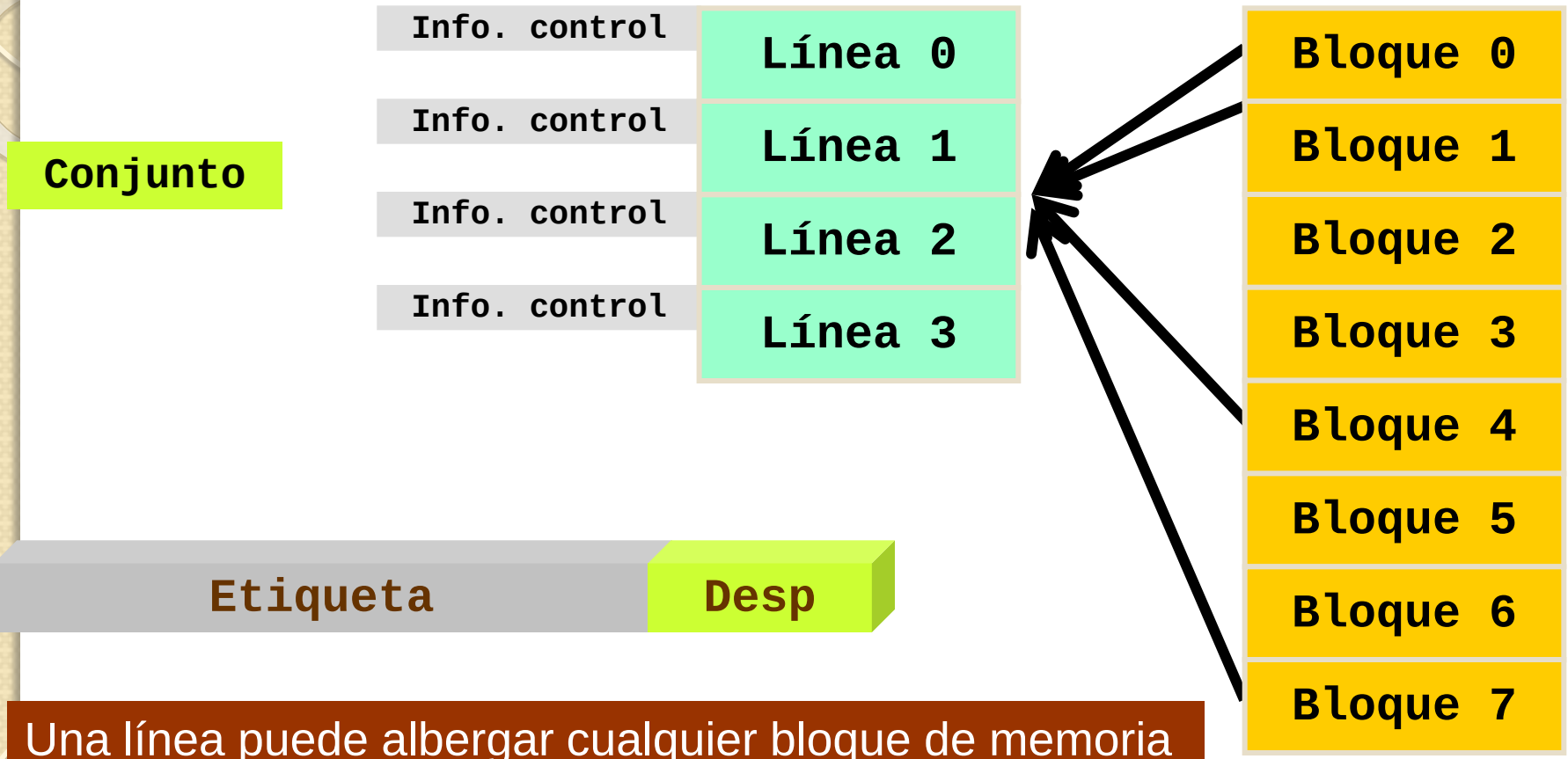


$$\text{Conjunto} = \text{Bloque } \mathit{MOD} \text{ Conjuntos en cache}$$

Un bloque se ubica en un único conjunto.

El conjunto 0 puede albergar los bloques 0, 2, 4 y 6 (aunque un máximo de dos al mismo tiempo).

Aproximación: totalmente asociativa

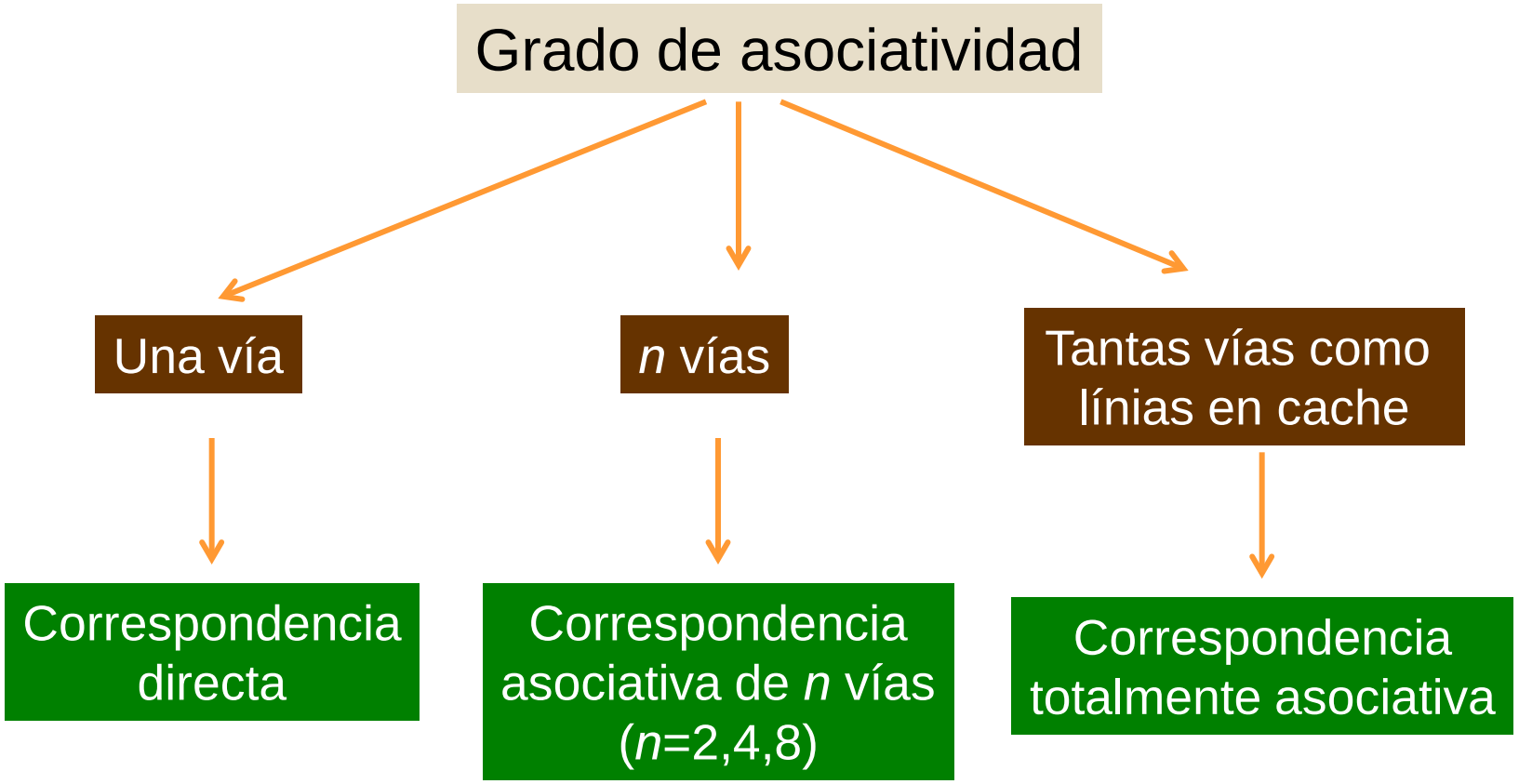


Una línea puede albergar cualquier bloque de memoria

La cache elegirá la ubicación de cada bloque según la disponibilidad de líneas en toda la cache

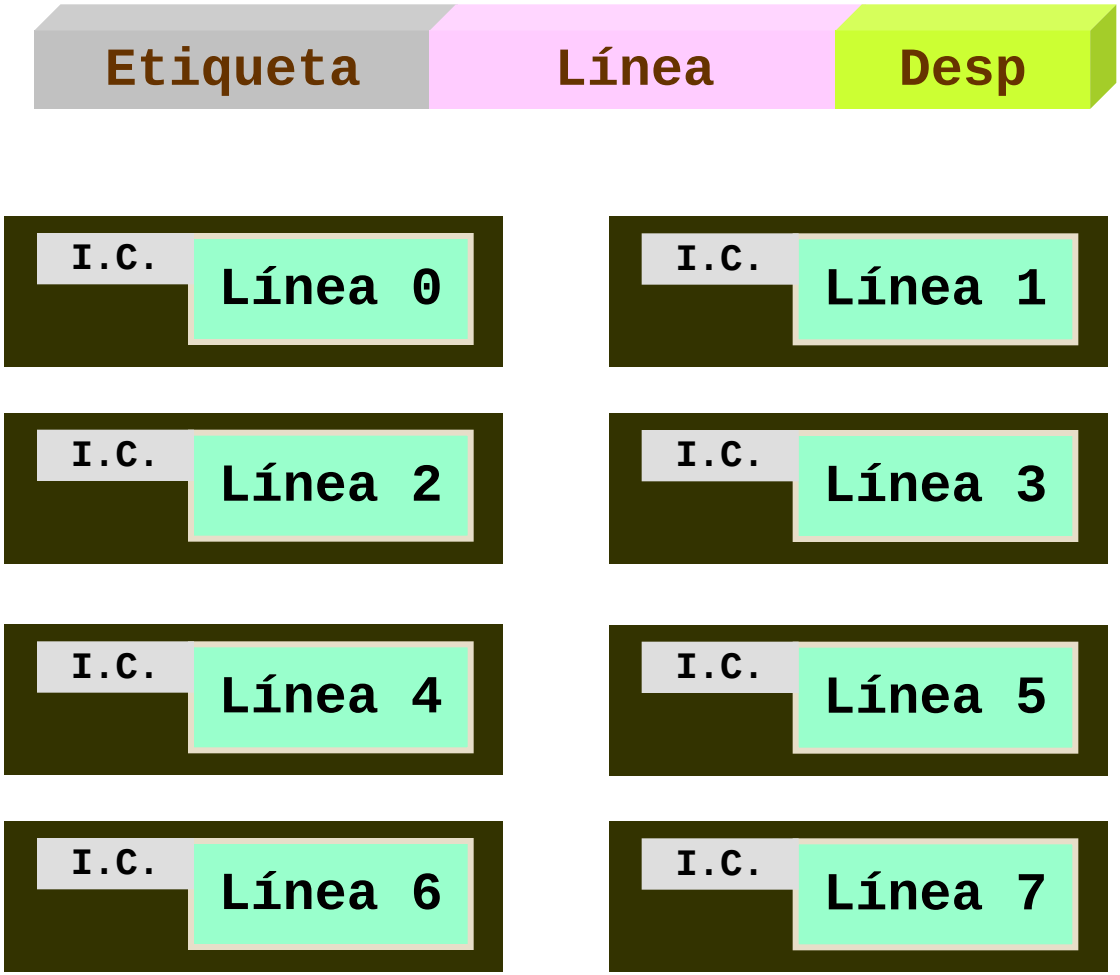
Se trata de una correspondencia asociativa de 4 vías

Resumen de correspondencias



Correspondencia asociativa de una vía

- Se trata de una correspondencia directa

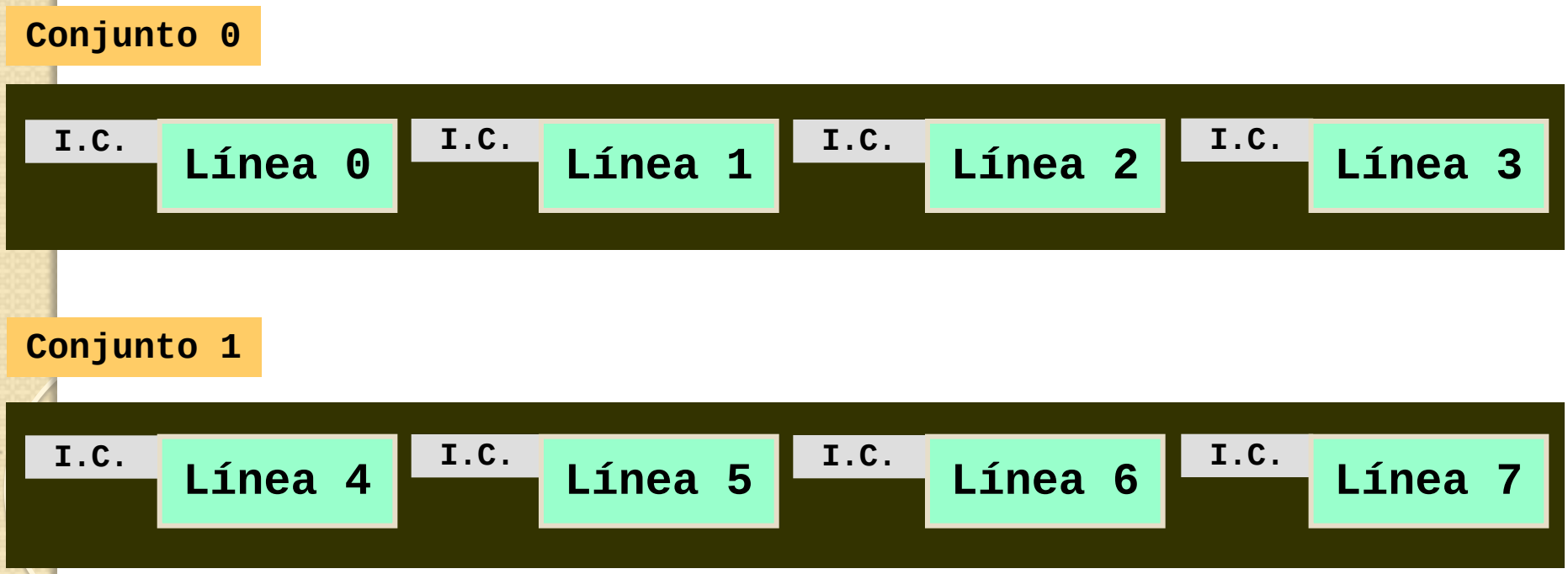


Correspondencia asociativa de 2 vías

- Hay 8 líneas agrupadas en 4 conjuntos de 2 líneas
- Cada bloque tiene dos posibles ubicaciones

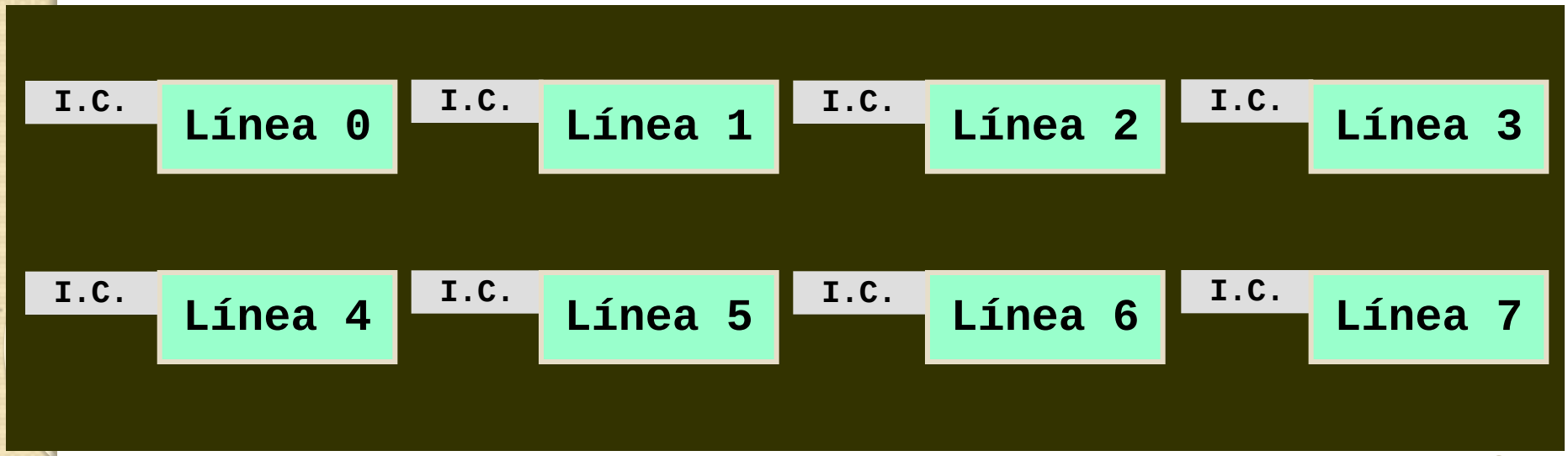
Correspondencia asociativa de 4 vías

- Hay 8 líneas agrupadas en 2 conjuntos de 4 líneas



Correspondencia asociativa de 8 vías

- Solo hay "un conjunto" que abarca todas las líneas
- Máximo grado de asociatividad

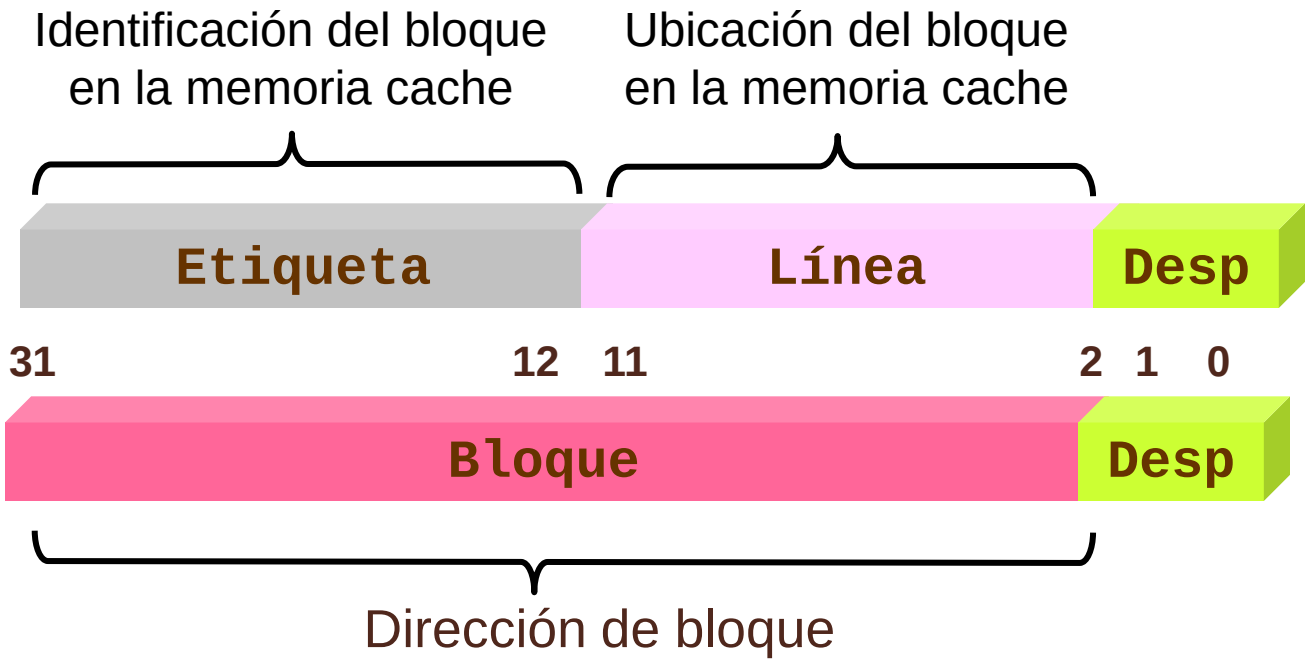


Ejercicio: rellenado de las siguientes tablas

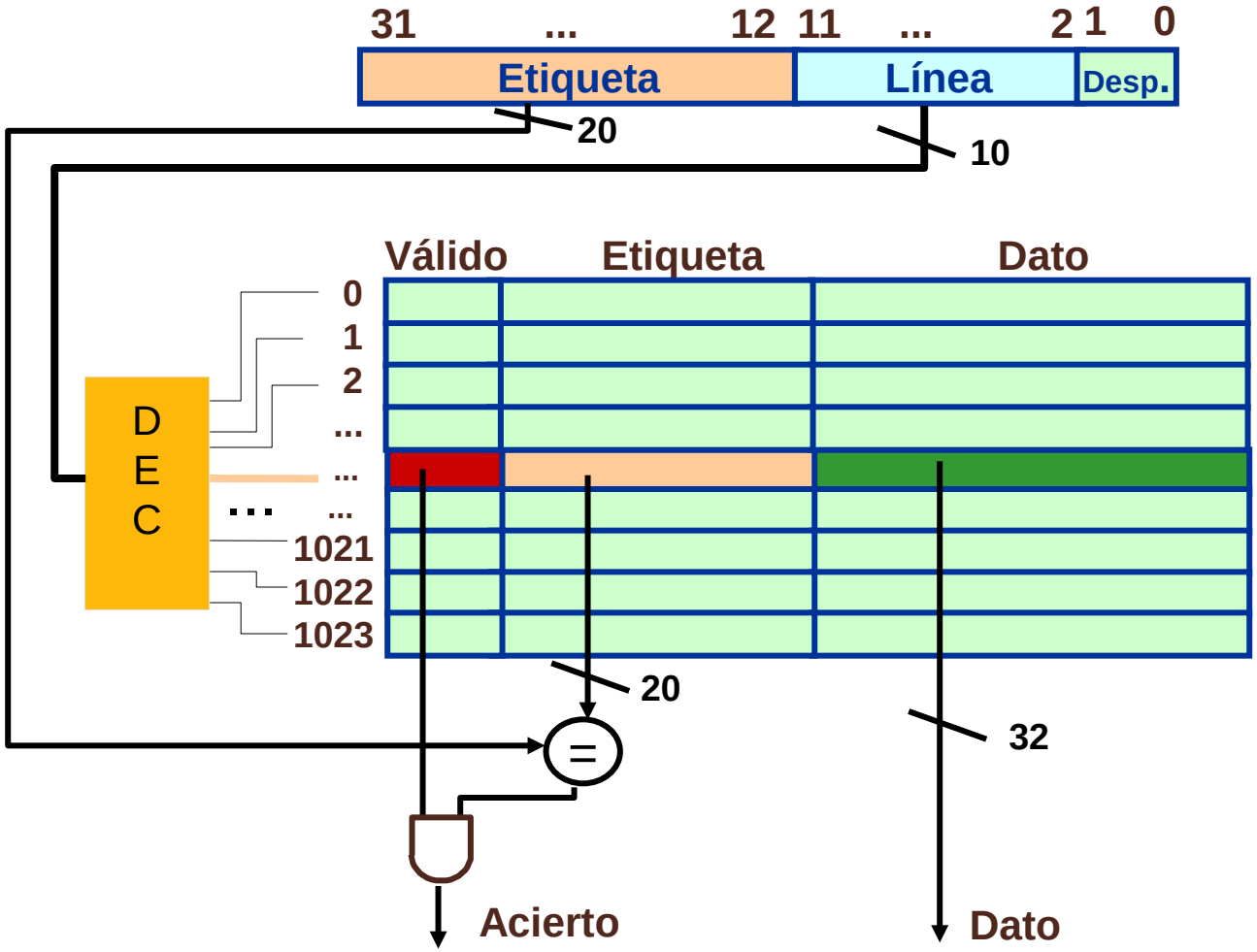
UCP		Memoria Cache					Bits dirección		
Dir.	W	Tamaño	vías	Tam. Bl.	# Bloq.	#cjtos	etiqueta	conjunto	desp
32	32	32KB	2	32B	1024	512	18	9	5
32	32	8KB	4	32B	256	64	21	6	5
24	32	32KB	1	32B	1024	1024	9	10	5
24	32	4KB	128	32B	128	1	19	0	5
36	64	32KB	2	64B	512	256	22	8	6
36	64	8KB	4	64B	128	32	25	5	6

Ejemplo para el MIPS R2000 (directa)

- Bus de direcciones de 32 bits
- Tamaño de bloque: 1 palabra (4 bytes)
- Tamaño de cache: 4 KB
  - ✓ Líneas en cache: 1024
- Interpretación de los campos de dirección



Correspondencia directa: organización interna

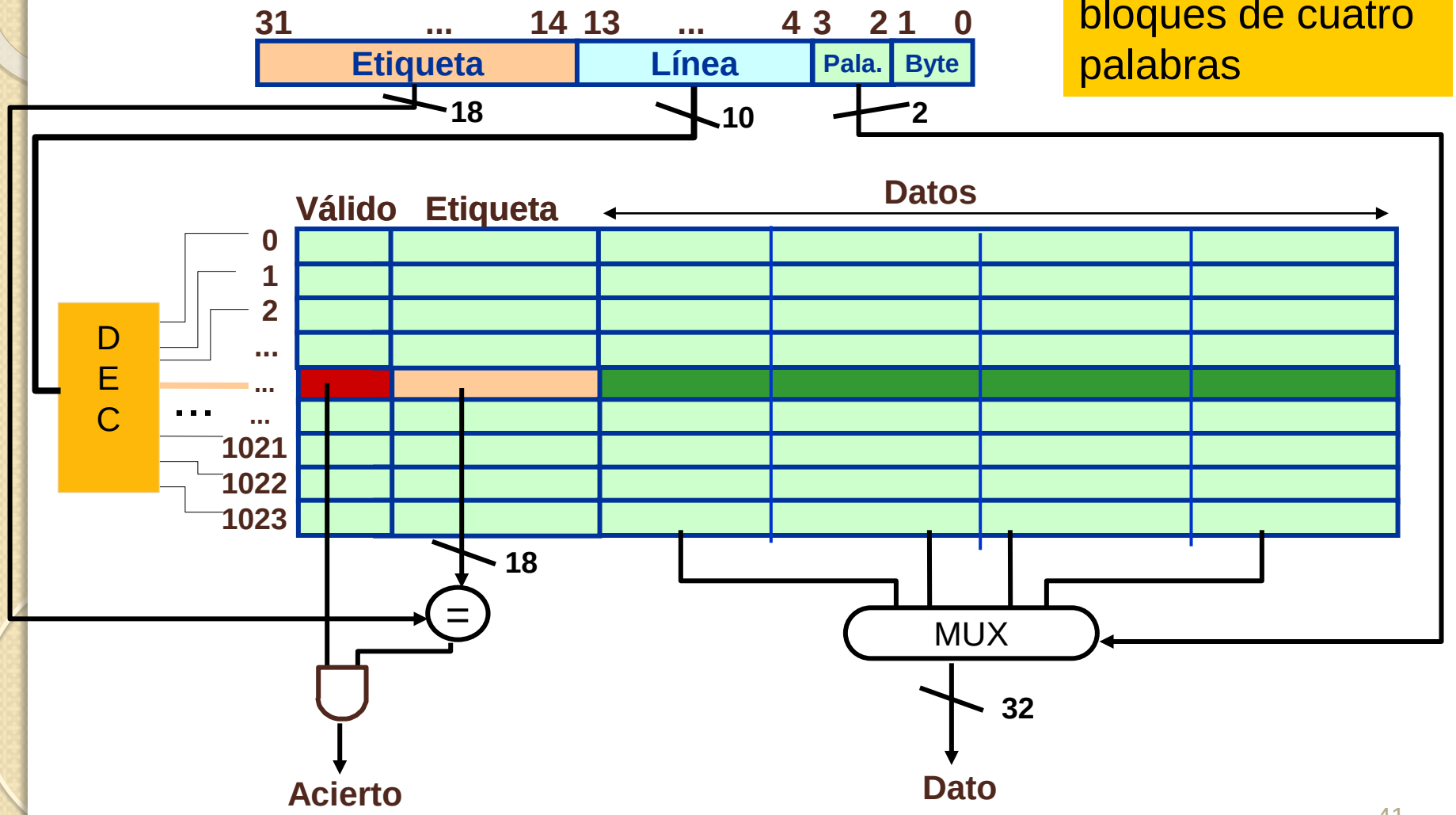


Cache de 4 KB, bloques de una palabra

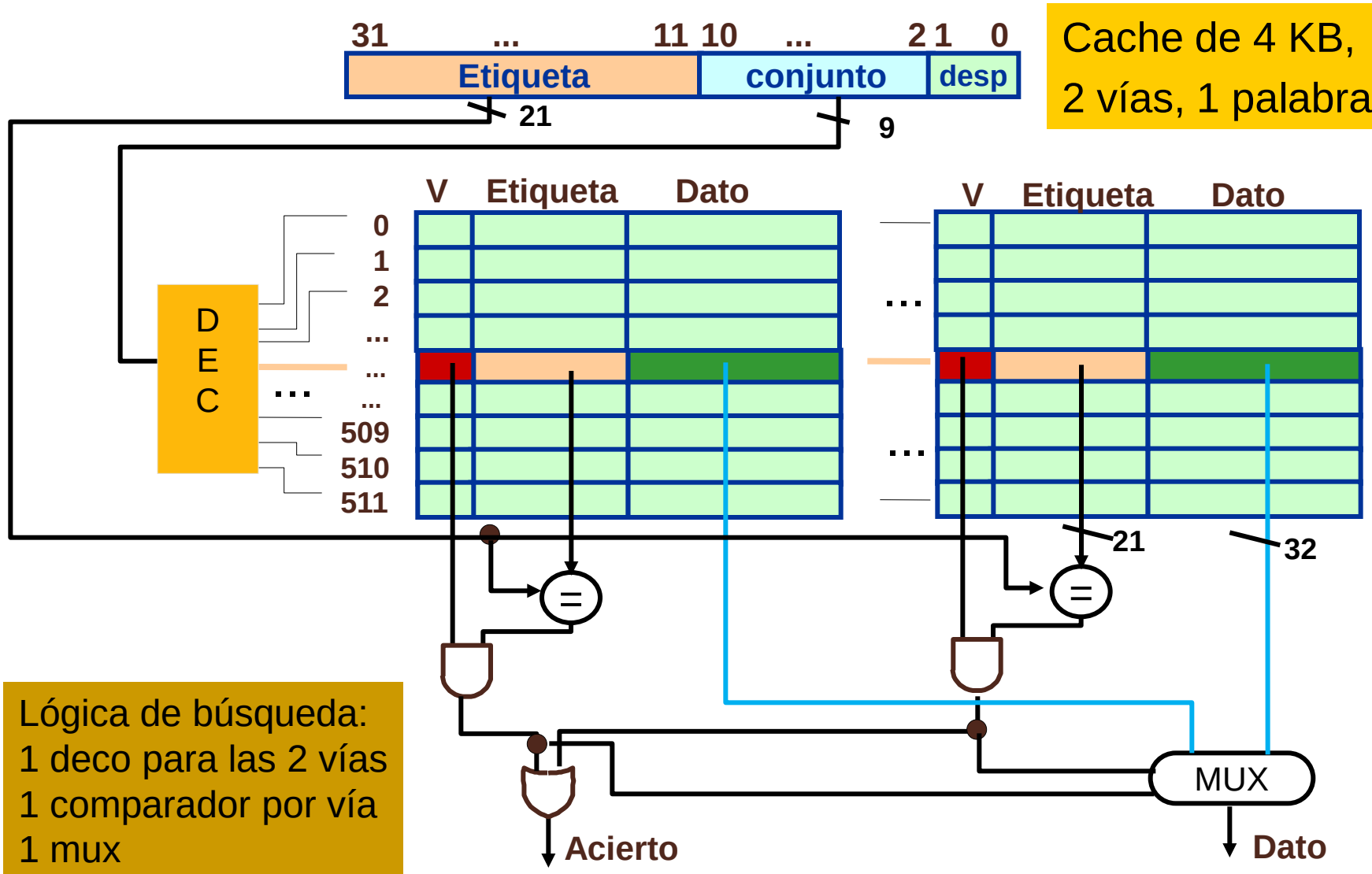
Lógica de búsqueda: 1 descodificador, 1 comparador, 1 puerta AND

Correspondencia directa: organización interna

Cache de 16 KB, bloques de cuatro palabras



Implementación de una cache asociativa



Cache de 4 KB,
2 vías, 1 palabra

Lógica de búsqueda:
1 deco para las 2 vías
1 comparador por vía
1 mux

Ejemplo de acceso a un vector

```
          .data 0x10000000
          .word 2,6,5,7,8,3,4,1,9,0
          .globl __start

          .text 0x00400000

__start:
          lui  $t0, 0x1000
          addi $t1, $zero, 10
          addi $a0, $zero, $zero
bucle:    lw   $t2, 0($t0)
          add  $a0, $a0, $t2
          addi $t2, $t2, 1
          addi $t0, $t0, 4
          addi $t1, $t1, -1
          bnez $t1, bucle
```

Segmento de datos:
10 palabras de 32 bits

Segmento de código:
9 instrucciones de 32 bits

Direcciones emitidas por el procesador

Segmento de datos

```
0x10000000
0x10000004
0x10000008
0x1000000C
0x10000010
0x10000014
0x10000018
0x1000001C
0x10000020
0x10000024
```

10 accesos de lectura (efecto de la instrucción **`lw`**)

Segmento de código

```
0x00400000
0x00400004
0x00400008
0x0040000C
0x00400010
0x00400014
0x00400018
0x0040001C
0x00400020
0x0040000C
0x00400010
0x00400014
0x00400018
0x0040001C
0x00400020
...
```

Direcciones de las instrucciones del bucle

3+10 × 6=63 accesos de lectura (lectura de las instrucciones por la unidad de control del procesador)

Tasa de aciertos en el acceso a datos

- Se supone cache directa de 32 KB y bloques de 16 bytes
  - ✓ Los datos se ubican en 3 bloques
  - ✓ 17 bits de etiqueta, 11 de línea, 4 de desplazamiento

```
00010000000000000 00000000000 0000
00010000000000000 00000000000 0100
00010000000000000 00000000000 1000
00010000000000000 00000000000 1100

00010000000000000 00000000001 0000
00010000000000000 00000000001 0100
00010000000000000 00000000001 1000
00010000000000000 00000000001 1100

00010000000000000 00000000010 0000
00010000000000000 00000000010 0100
```

$$H = \frac{10-3}{10} = 0{,}7$$

Tasa de aciertos en el acceso a código

- Se supone cache directa de 64 KB y bloques de 16 bytes
  - ✓ 18 bits de etiqueta, 12 de línea, 4 de desplazamiento

```
000000001000000 000000000000 0000
000000001000000 000000000000 0100
000000001000000 000000000000 1000
000000001000000 000000000000 1100

000000001000000 000000000001 0000
000000001000000 000000000001 0100
000000001000000 000000000001 1000
000000001000000 000000000001 1100

000000001000000 000000000010 0000
000000001000000 000000000000 1100
000000001000000 000000000001 0000
000000001000000 000000000001 0100
```

Direcciones de las instrucciones del bucle

$$H = \frac{63-3}{63} = 0{,}95$$

Tipos de fallos (son exclusivos)

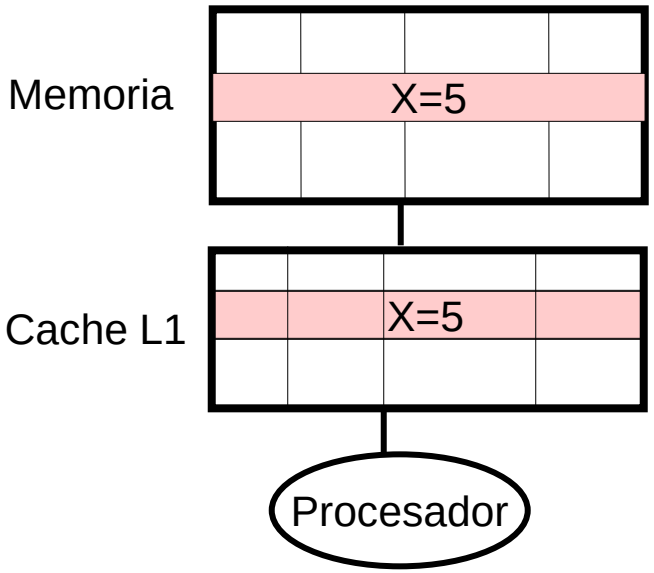
- **Fallos de arranque**
  - ✓ Se producen la primera vez que se referencia un bloque (no ha estado todavía en la cache)
- **Fallos de capacidad**
  - ✓ Se producen debido a la limitación del tamaño de la cache
- **Fallos conflicto (o colisión)**
  - ✓ Se producen el conjunto al que corresponde está lleno pero la cache no
  - ✓ Solo se producen en caches de correspondencia directa y asociativa por conjuntos, pero no en las completamente asociativas
- Una cache de mayor tamaño reduce los fallos de capacidad y un mayor número de vías los de conflicto; pero podría incrementar el tiempo de acceso

Políticas de lectura

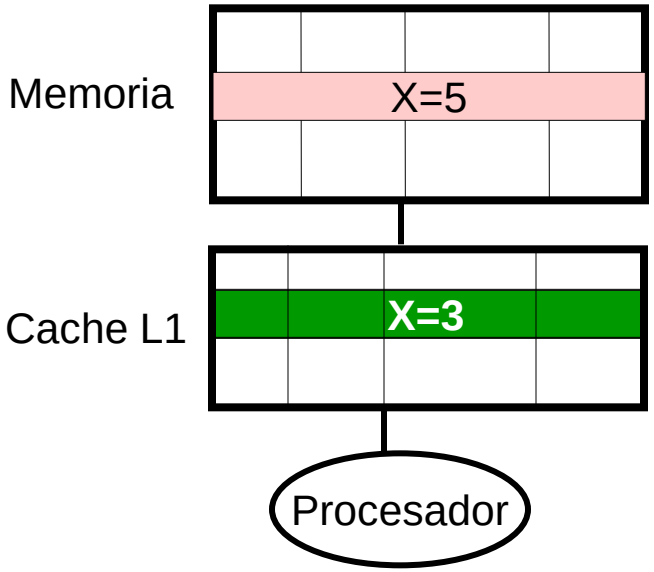
- Acierto de lectura
  - ✓ ¡Situación deseada!
- Fallo de lectura
  - ✓ Hay que traer el bloque desde la memoria principal (o del nivel inferior de la jerarquía) a la memoria cache
    - El tiempo de acceso se incrementa
  - ✓ La memoria principal, al leer el bloque, ofrece primero la palabra que causó el fallo (*critical word first*)
    - El resto del bloque se acaba de traer en orden circular
  - ✓ Por tanto, el tiempo para resolver el fallo es la latencia de acceso a los datos $t_{RCD}+t_{CL}$

El problema de la coherencia

- Las escrituras pueden provocar incoherencia



Situación 1: fallo de lectura



Situación 2: acierto de escritura
Si procesador escribe solo en L1 hay incoherencia entre L1 y Memoria

Solución al problema: políticas de escritura

Políticas de acierto en escritura (*Políticas de actualización en escritura*)

- Acierto de escritura
  - ✓ Escritura directa (*write through*)
    - Se escribe el dato en la cache y en la memoria principal (nivel inferior)
  - ✓ Escritura posterior (*write back*)
    - Se escribe en la cache pero no en la memoria principal (nivel inferior)
    - Cuando se reemplaza el bloque se escribe en el nivel inferior
      - Se requiere un bit adicional: bit MODIFICADO (*dirty bit*)

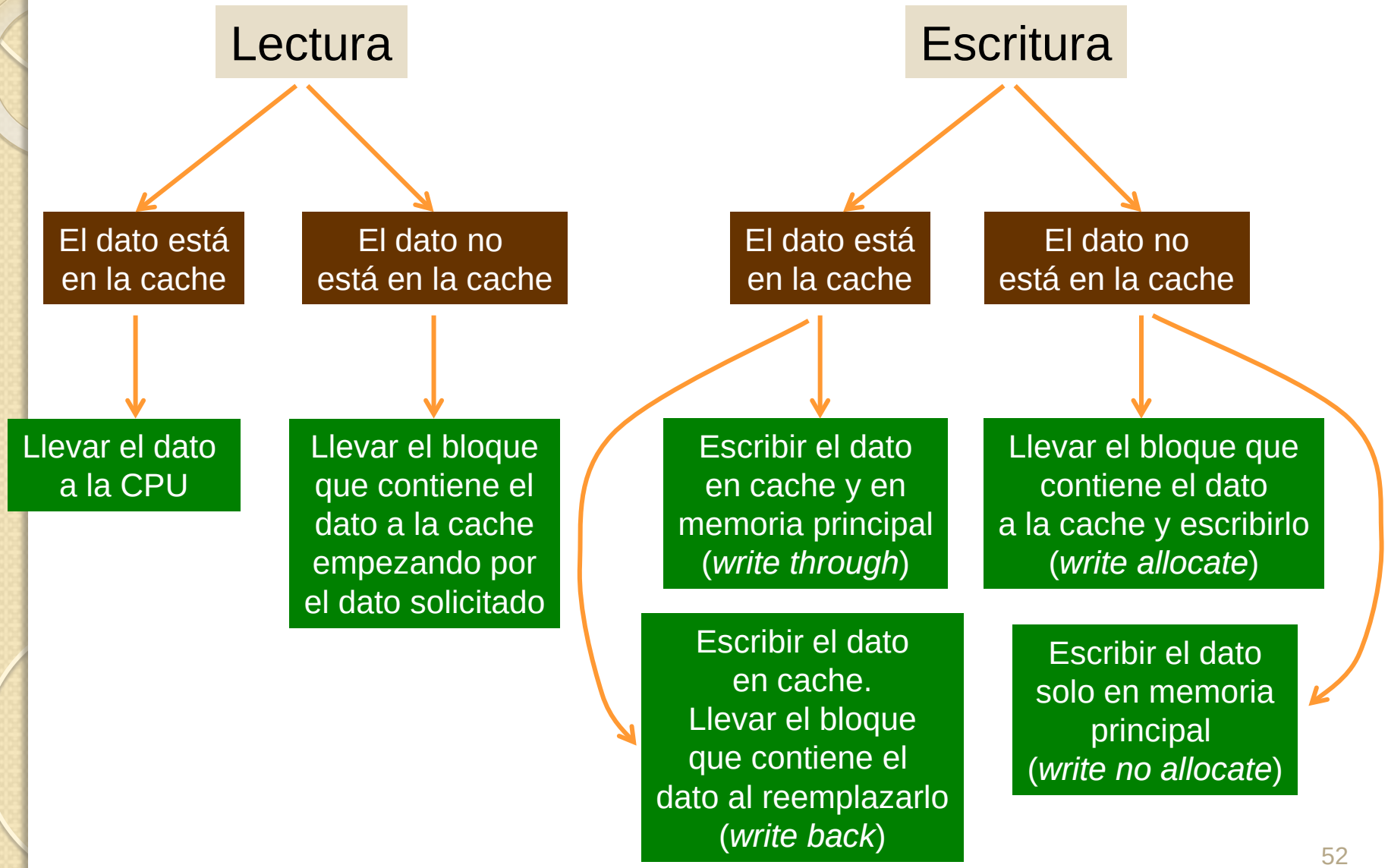
El elevado número de ciclos de reloj que conlleva la escritura en memoria principal podría penalizar significativamente al procesador, especialmente en el caso de *escritura directa*

Solución → **Buffers de Escritura**

Políticas de fallo en escritura

- Fallo de escritura
  - ✓ Ubicación/reserva en escritura (*write-allocate*)
    - Gestión del fallo similar a un fallo de lectura
    - Se trae el bloque y se escribe el dato (escritura directa o posterior)
  - ✓ No ubicación/reserva en escritura (*no-write-allocate*)
    - Se escribe el dato en el nivel inferior, no en la cache
    - Solo se utiliza con la política de escritura directa

Resumen de políticas de lectura y escritura



Algoritmos de reemplazo

- Se aplican cuando un bloque debe almacenarse en un conjunto que se encuentra lleno
- Se encuentran implementados en el hardware
- Algoritmos más empleados
  - ✓ Menos recientemente usado (LRU, *least recently used*)
    - Implementación
      - Se utiliza un contador por línea para mantener el orden de referencia
      - Tamaño del contador en bits:

$$n = \log_2(\text{Número de vías})$$

  - ✓ Orden de llegada (FIFO, *first in first out*)
    - Bits adicionales (misma cantidad que LRU)
  - ✓ ALEATORIO (*random*)
    - No es necesario añadir bits

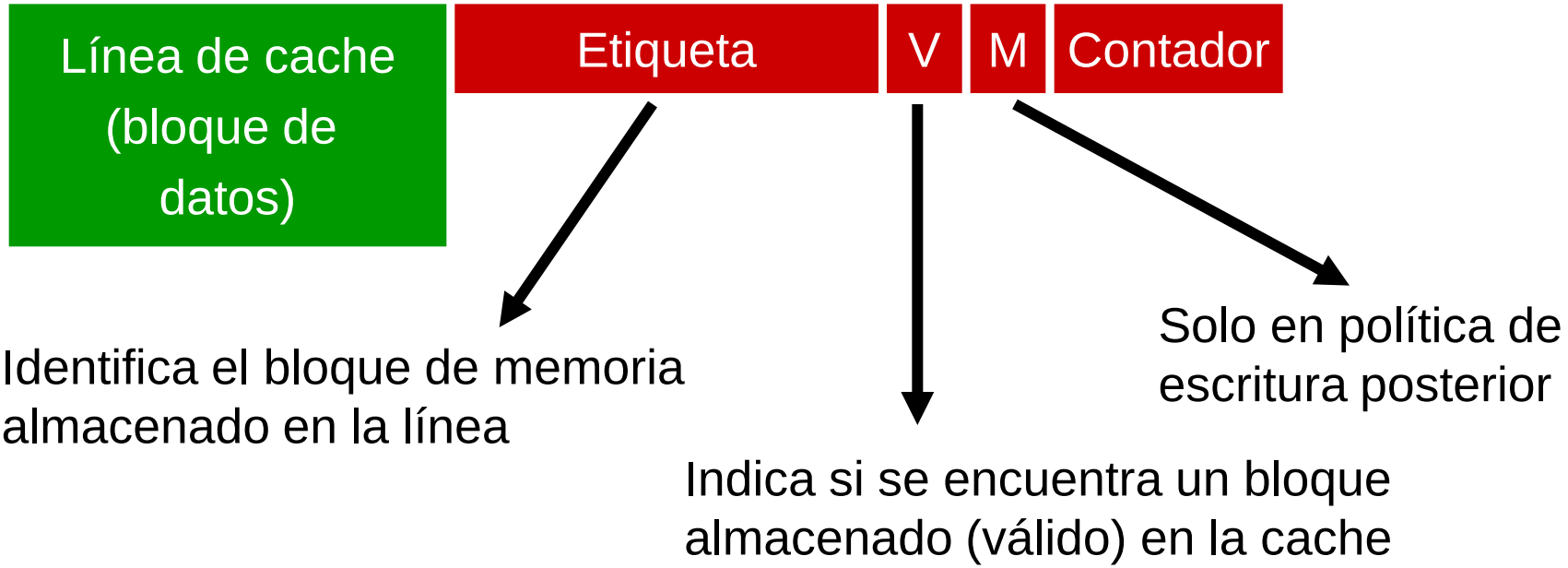
Detalles del algoritmo LRU

- Fallo en la cache
  - ✓ Si conjunto lleno (todos los bits de línea válida activos)
    - Eliminar línea con valor de contador máximo (ctr = 2^n-1)
    - Para el resto de líneas hacer ctr++
- Acierto
  - ✓ Para todo ctr < $ctr_{línea\_referenciada}$ hacer ctr++
- En cualquier caso, $ctr_{línea\_referenciada} = 0$

- Ejemplo de contadores de un conjunto de 4 vías
  - Ctr_vía_0 = 1
  - Ctr_vía_1 = 3 ← línea LRU (*least recently used*)
  - Ctr_vía_2 = 2
  - Ctr_vía_3 = 0 ← línea MRU (*most recently used*)

Información de control: directorio

- Información adicional al bloque almacenado en una línea
- Su volumen depende de la configuración de la cache
  - ✓ Bits de la etiqueta
  - ✓ Bit de válido, bit de modificado (solo política *writeback*)
  - ✓ Bits del contador del algoritmo de reemplazo



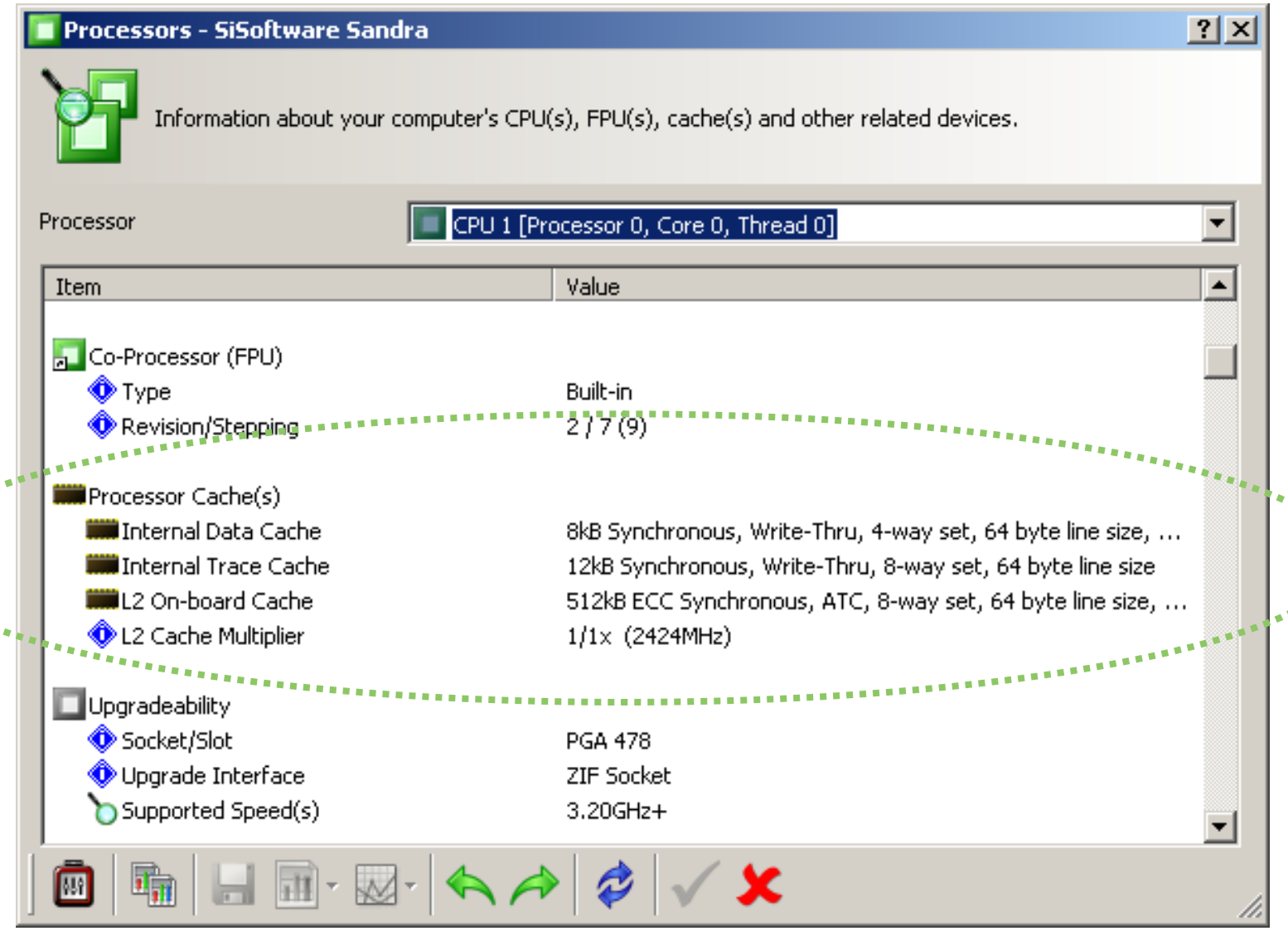
Cálculo del volumen del directorio

- Procesador con 32 bits de dirección
- Cache de 512 líneas de 16 bytes (8 KB)
  - ✓ Asociativa por conjuntos de 4 vías (128 conjuntos)
  - ✓ Política de escritura: actualización directa (*write through*)
  - ✓ Algoritmo de reemplazo FIFO (se escoge un bloque entre cuatro)

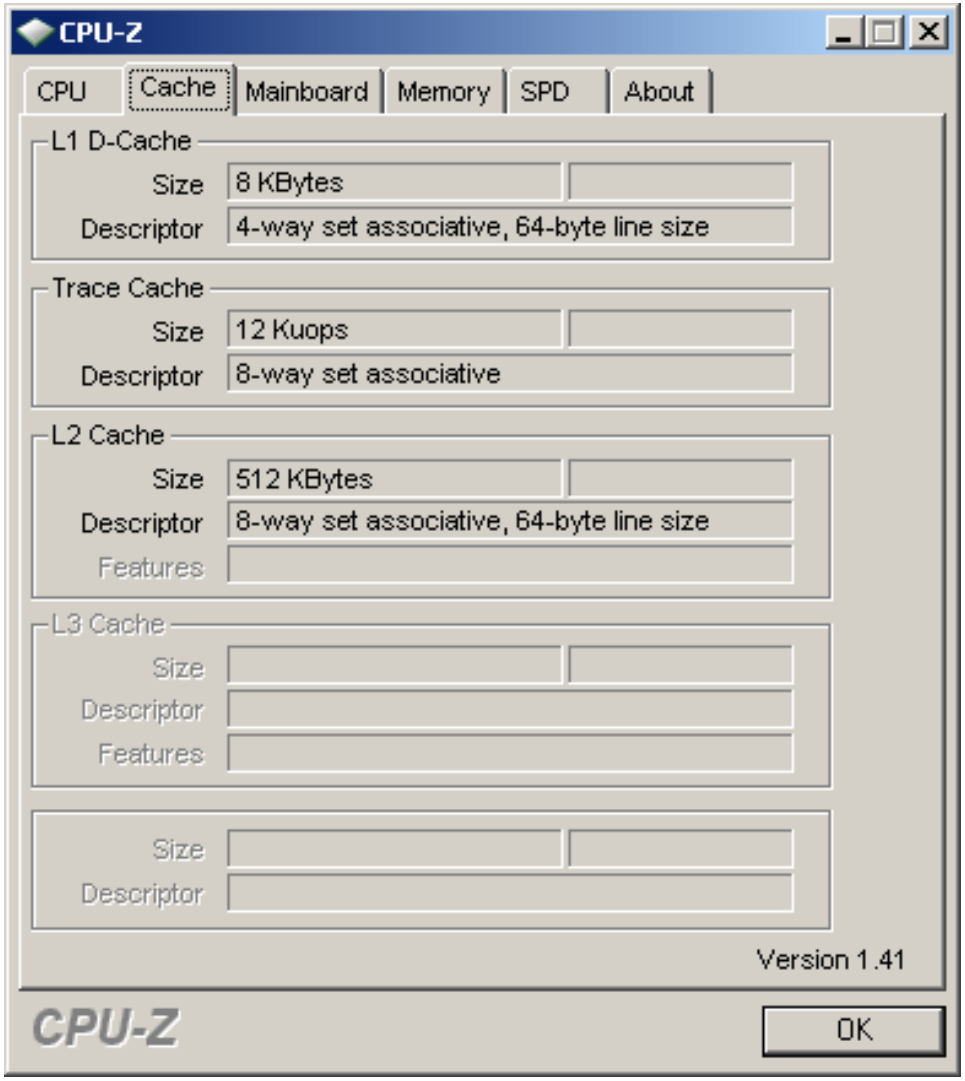
Etiqueta: 21 bits
Bit de válido: 1 bit
Contador: 2 bits

$$V = 512 \times (21+1+2)$$
$$= 12288 \text{ bits}$$
$$= 1.5 \text{ KB}$$

SiSoftware Sandra: memoria cache



CPU-Z: memoria cache



¿Cómo mejorar el rendimiento?

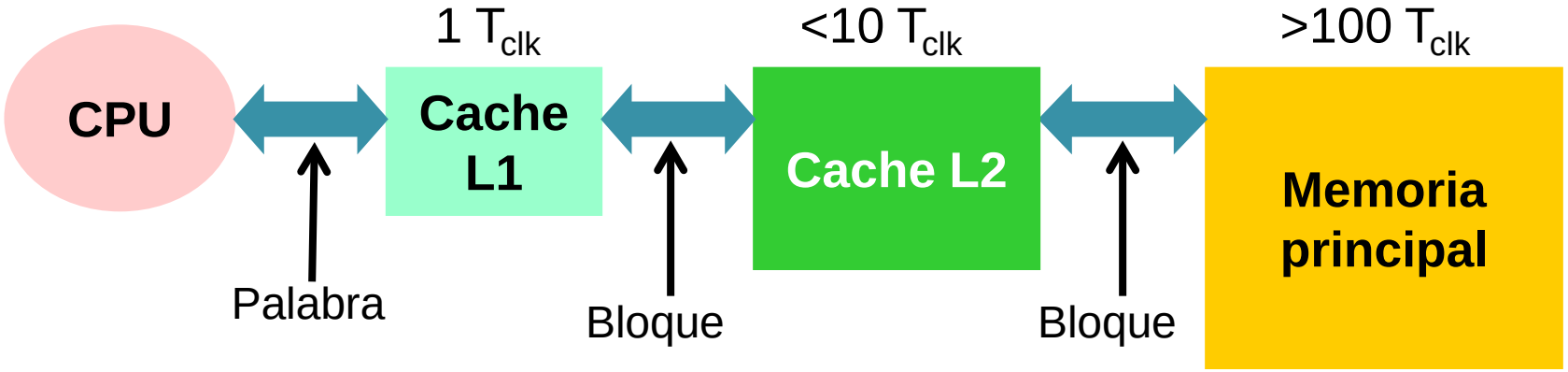
- Tiempo medio de acceso a los datos:

$$T_m = H \times T_{acierto} + (1 - H) \times T_{fallo}$$

- Estrategias de mejora
  - ✓ Reducir el tiempo de acierto: cuestiones tecnológicas
  - ✓ Disminuir la tasa de fallos: organizaciones más eficientes (añadir más vías)
  - ✓ Reducir el tiempo para resolver el fallo
    - Caches multinivel
      - 2 y hasta 3 niveles de cache
      - Nombres usuales: L1, L2, L3
    - Buses más anchos entre niveles de memoria cache

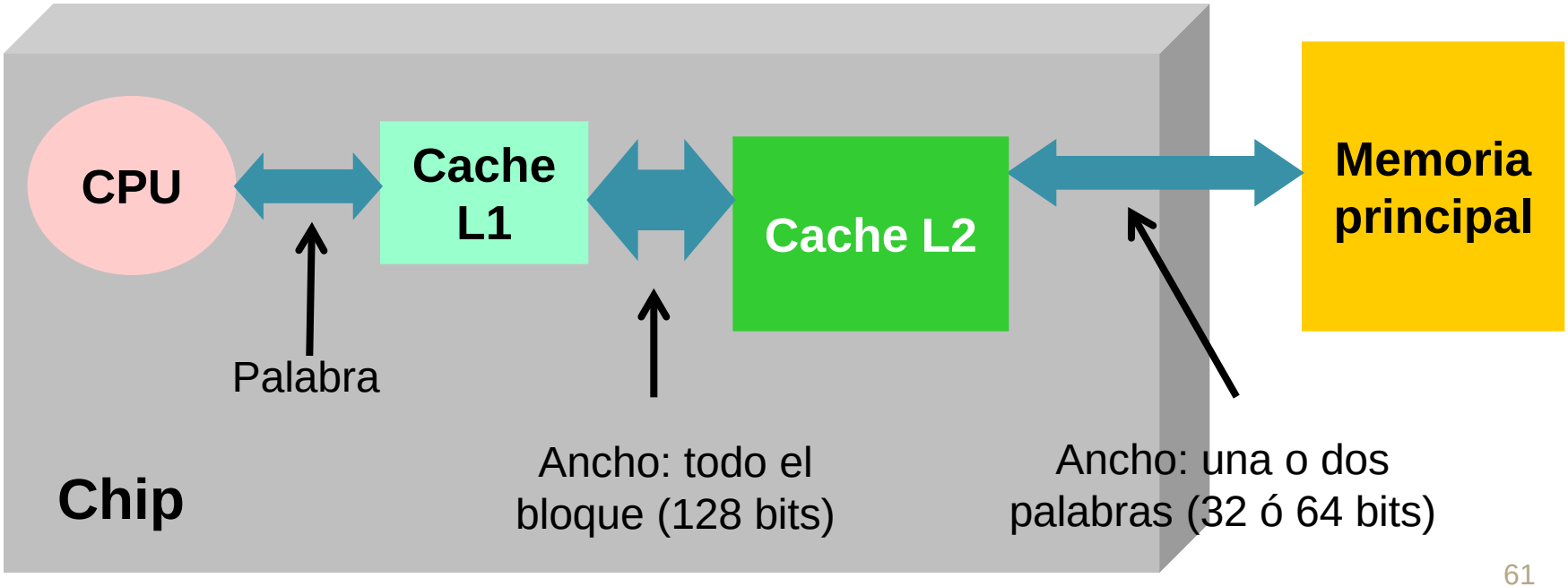
Memorias cache multinivel

- Cache L2: es mucho más grande y bastante más lenta que la cache L1
- Organizaciones independientes
  - ✓ Las dos caches tienen organizaciones y políticas independientes
  - ✓ Ejemplo: Pentium 4
    - Cache L1: escritura directa, no ubicación
    - Cache L2: escritura posterior, ubicación



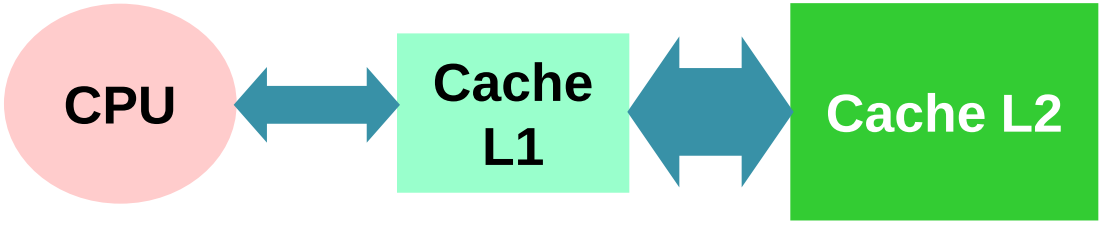
Ancho del bus en caches multinivel

- Si la cache L2 está integrada en el procesador el ancho del bus L2-L1 puede ser todo el bloque (no hay problemas con el número de pines)
  - ✓ El ancho del bus entre la memoria y las caches es de una o dos palabras: transferencia por bloques
  - ✓ Ejemplo: bloques de cuatro palabras de 32 bits



Penalización en el acceso a la cache L1

- Existen dos formas de acceder a la cache L2 en relación con el acceso a L1
  - ✓ Secuencialmente
    - Cuado se detecta un fallo en L1 se accede a L2
  - ✓ En paralelo
    - Se accede a L1 y a L2 al mismo tiempo (mismo bus)
    - Si hay acierto en L1 se aborta la operación en L2
    - En caso de fallo, la penalización de L1 es (caso de acierto) el tiempo de acceso a L2



Rendimiento y parámetros

- El diseño de una memoria cache contempla
  - ✓ el número de niveles, la capacidad, la asociativada, el tamaño de bloque, las políticas de lectura y escritura, las políticas de reemplazo
- El rendimiento de un diseño de memoria cache depende
  - ✓ de las características de los programas que ejecuta el procesador (tipo de datos, localidad espacial y temporal de los accesos)
  - ✓ del ancho de banda demandado por el procesador (depende la frecuencia de accesos a la memoria y del ancho de palabra)
  - ✓ del ancho de banda de la memoria principal

iPad 4

iPhon 5

Procesador A6X: 32-bit system-on-a-chip (SoC)

ARMv7s Cortex-A9 Swift (32-bit dual core processor; 1.4GHz)

quad-core PowerVR SGX554MP4 graphics processing unit (GPU) 300MHz

1 GB of LPDDR2-1066 RAM

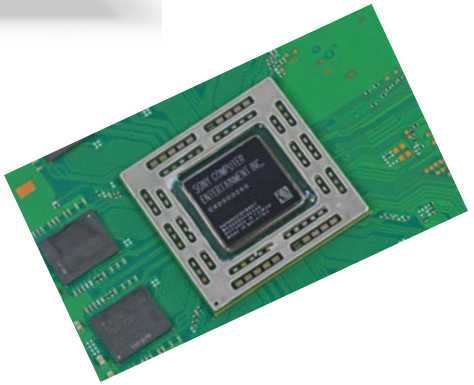
L1 (dual) { L1I: 32KB 4-way ; L1D: 32KB 4-way } *cada core*

L2 (unificada): 1MB 4-16 way *;compartida por los dos cores*

Cache line: 32B

PlayStation 4

Xbox One



Procesador AMD Jaguar (64-bit; dual-module; 4 cores/module; 1.6/1.75GHz)

AMD's GPGPU-capable Radeon GCN architecture 800MHz

PlayStation: 8 GB of GDDR5-256 bits RAM 176GBps

L1 (dual) { L1I: 32KB 2-way ; L1D: 32KB 8-way } *cada core*

L2 (unificada): 2MB 4-16 way *;compartida por los 4 cores*

Cache line: 64B

ASUS laptop

Procesador Intel Core i7-4700HQ (64 bits; quad core) 2.4GHz

8 GB of DDR3-1600 RAM

L1 (dual) { L1I: 32KB 4-way; L1D: 32KB 8-way } *cada core*

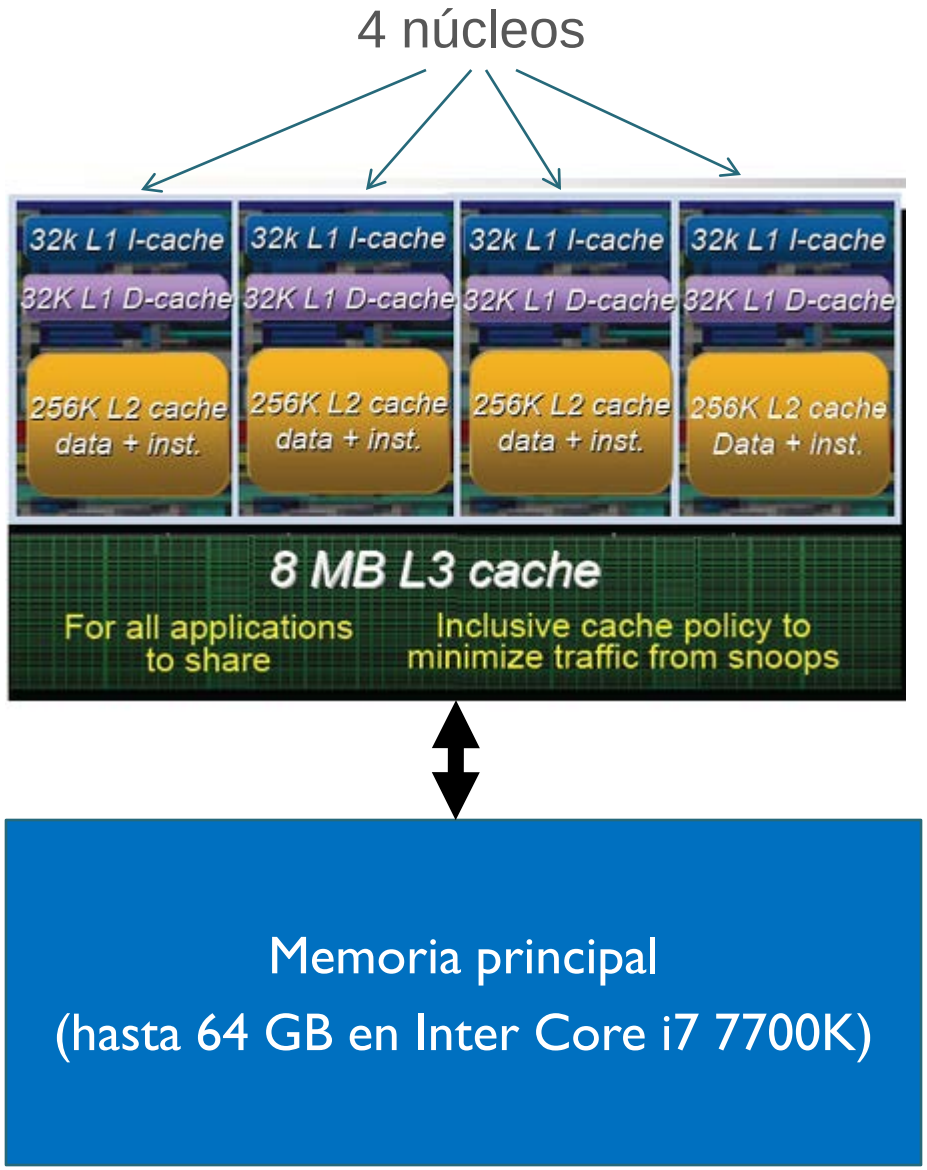
L2 (unificada): 256KB 8-way

L3 (unificada): 6MB 16-way *;compartida por los cuatro cores*

Cache line: 64B

Write-back

Memoria cache en un Intel Core i7



Intel server board S4600LH product family

Procesador Intel Xeon E5-4650L (64 bits; 8 cores) 2.6GHz

4 procesadores (32 cores)

1.5 TB of DDR3-1600 RAM

L1 (dual) { L1I: 32KB 8-way; L1D: 32KB 8-way } *cada core*

L2 (unificada): 256KB 8-way

L3 (unificada): 25MB 16-way *;compartida por los 8 cores*

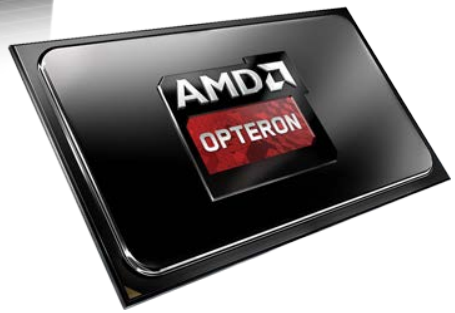
Cada procesador

Cache line: 64B

Write-back

Cray XK7 (4º en TOP500_2017)

"Interlagos"



Procesador AMD Opteron 6274 (64 bits; 16 cores) 2.4GHz

18688 procesadores (299008 cores); > 27 PataFlops

710 TB of DDR3-1600 RAM

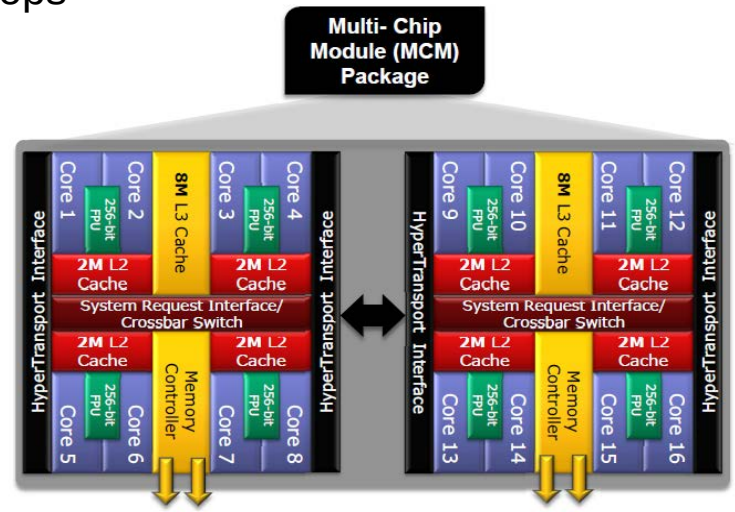
L1 (dual) { L1I: 2x64KB 2-way compartida; L1D: 16x16KB 4-way por core } *cada procesador*

L2 (unificada): 8x2MB 16-way

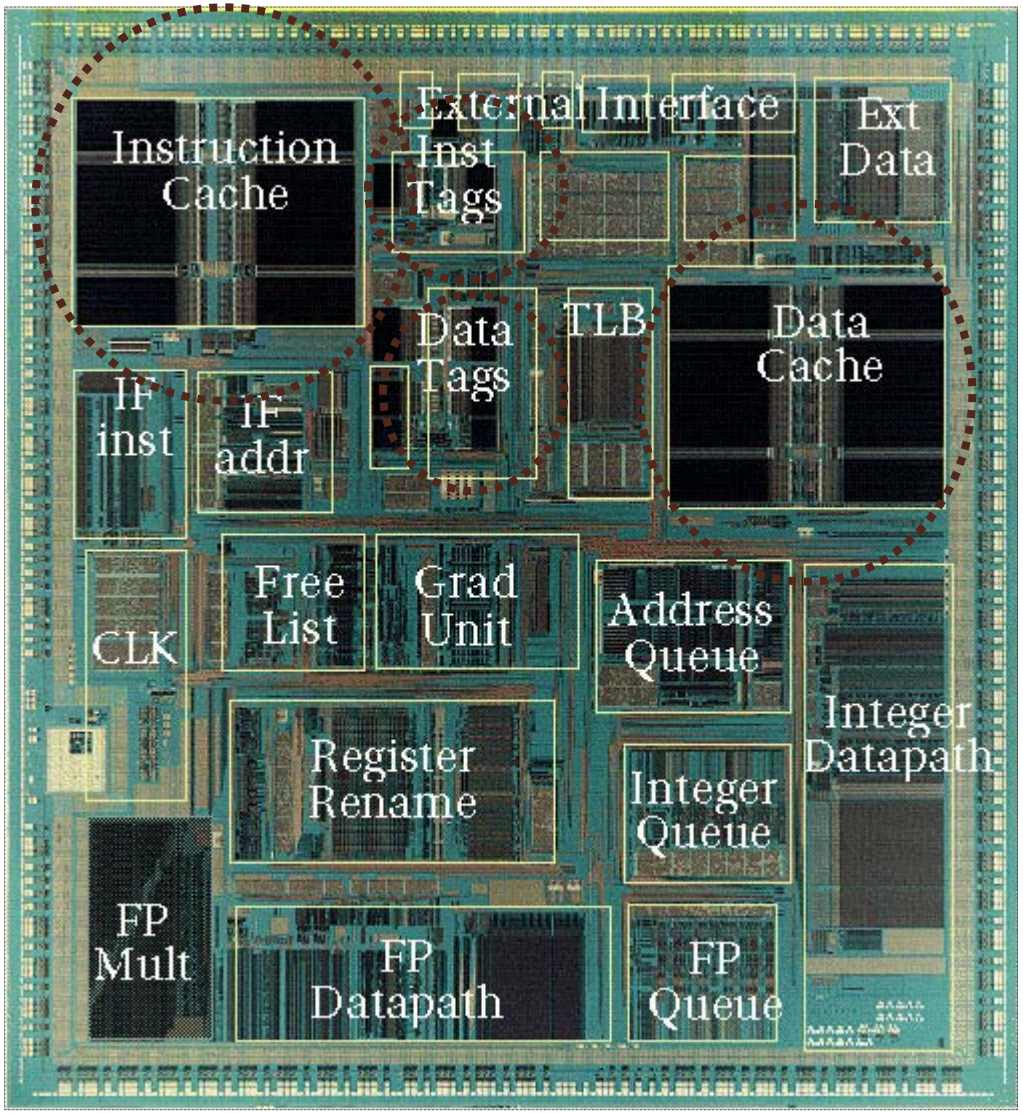
L3 (unificada): 2x8MB 64-way

Cache line: 64B

Write-back



Memoria cache en un MIPS R10000



L1: código (32 KB, 2 vías)

datos (32 KB, 2 vías)

L2: externa, 2 vías (0,5-16 MB)

Caches en procesadores MIPS: evolución

	Nivel L1				Nivel L2		
ID-MHz	Código	Datos					
R3000-33	32 KB	32 KB	Directa	Off chip			
R4000-100	8 KB	8 KB	Directa	On chip	1 MB	Directa	Off chip
R5000-200	32 KB	32 KB	2 vías	On chip	1 MB	Directa	Off chip

	Nivel L1				Nivel L2		
RM7000-250	16 KB	16 KB	4 vías	On chip	256 KB	4 vías	On chip

	Nivel L3		
RM7000-250	8 MB	Directa	Off chip

Tendencias en el diseño de la cache en MIPS

- El tamaño de la L1 se halla limitado por las restricciones de área de silicio y por la frecuencia de reloj del procesador, requiriéndose niveles adicionales de cache
- Cuanto mayor sea la frecuencia de reloj del procesador, mayor el número de niveles de cache requeridos
- Es habitual acceder a L1 en un ciclo de reloj del procesador
- L2 suele ser directa para evitar a los circuitos comparadores trabajar con muchas etiquetas en paralelo
- Las caches más recientes suelen usar
  - ✓ Escritura posterior
  - ✓ 2 ó 4 vías
  - ✓ La gestión se hace con la instrucción máquina **cache**

3. La memoria virtual

- Concepto y utilidad
- Direccionamiento virtual
- Traducción de direcciones

El problema

Múltiples programas/tareas capaces de acceder a 256 TB de memoria cada uno

?

Memoria Principal
~ 4 GB

¿ Cómo posibilitar la ejecución de tales programas teniendo en cuenta el reducido tamaño relativo de la memoria principal ?

La memoria virtual: utilidad

- Si no se dispone de memoria virtual el máximo espacio direccionable por el procesador sería el tamaño de la memoria principal o memoria física

- El uso de memoria virtual permite que los programas dispongan de un espacio de direcciones mayor que el de la memoria física
  - ✓ En un momento dado, la mayor parte de la MV direccionable se encuentra en disco
  - ✓ Para que los programas y datos sean accesibles por el procesasor deben encontrarse en memoria
  - ✓ El sistema operativo se encarga de realizar las gestiones oportunas (asignatura FSO)

Visión del programador

- El programador no se preocupa de la cantidad de memoria física del computador
- Procesador MIPS R2000
  - ✓ 32 bits de direcciones: 4 GB de espacio direccionable dividido en dos partes iguales (usuario y sistema)
  - ✓ La memoria principal suele ser de menor tamaño

Espacio visto por el programador

```
0x00000000
0x00000001
0x00000010
...
0xFFFFFFFF
```

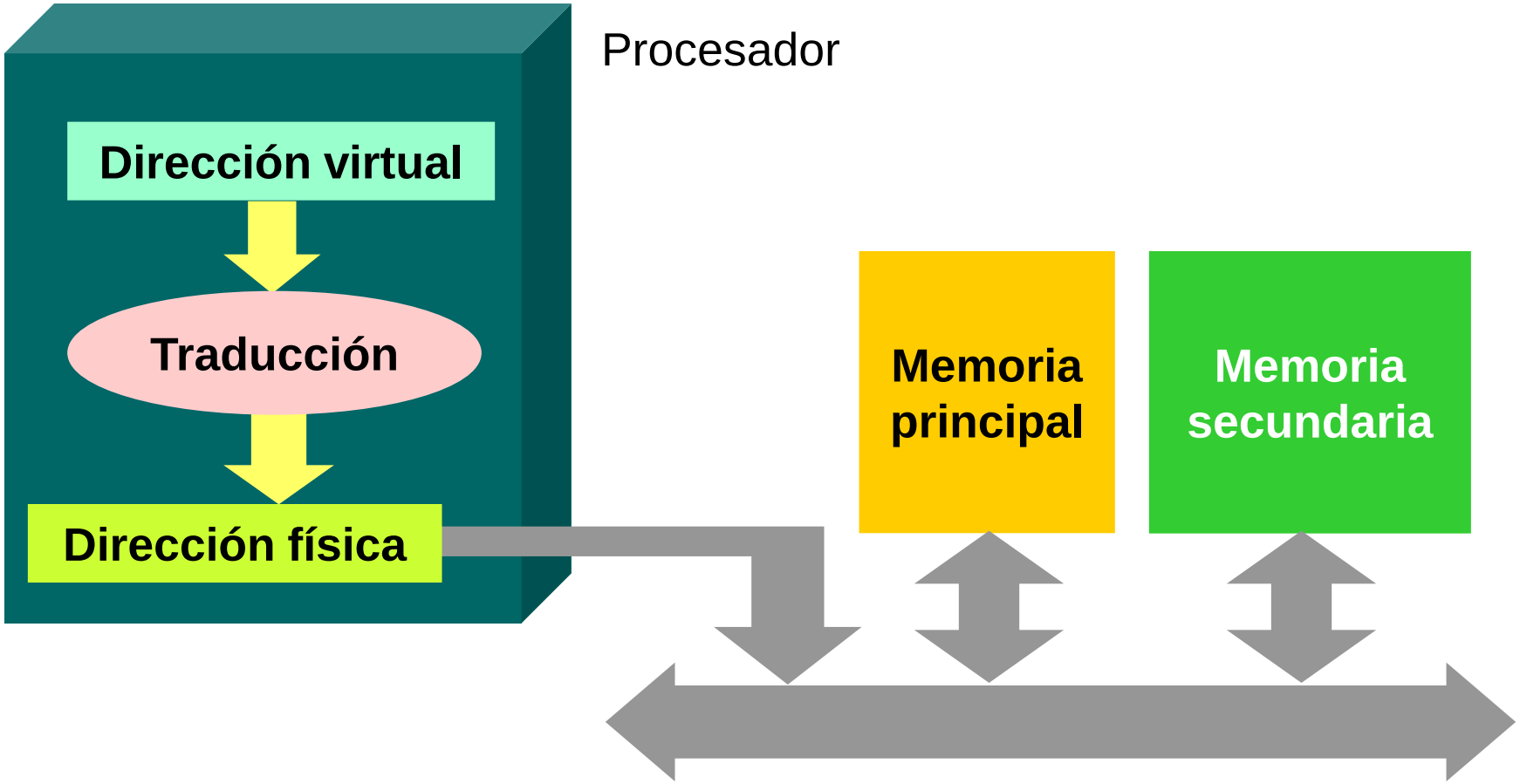
Implementación física del espacio direccionable gestionada por el SO

Direcciones virtuales y físicas

- Cuando se usa MV se puede distinguir
  - ✓ **Dirección virtual**
    - Dirección de un objeto utilizada por el procesador
    - El objeto puede estar en MP o en MS
  - ✓ **Dirección física**
    - Dirección de un objeto en MP
    - Son las direcciones emitidas por el bus de direcciones
- Es necesario obtener la dirección física a partir de la dirección virtual para acceder al objeto
  - ✓ Esta correspondencia se realiza en base a páginas de memoria
    - La MV se divide en páginas
    - La MP se divide en marcos de página
    - Cada marco alberga una página

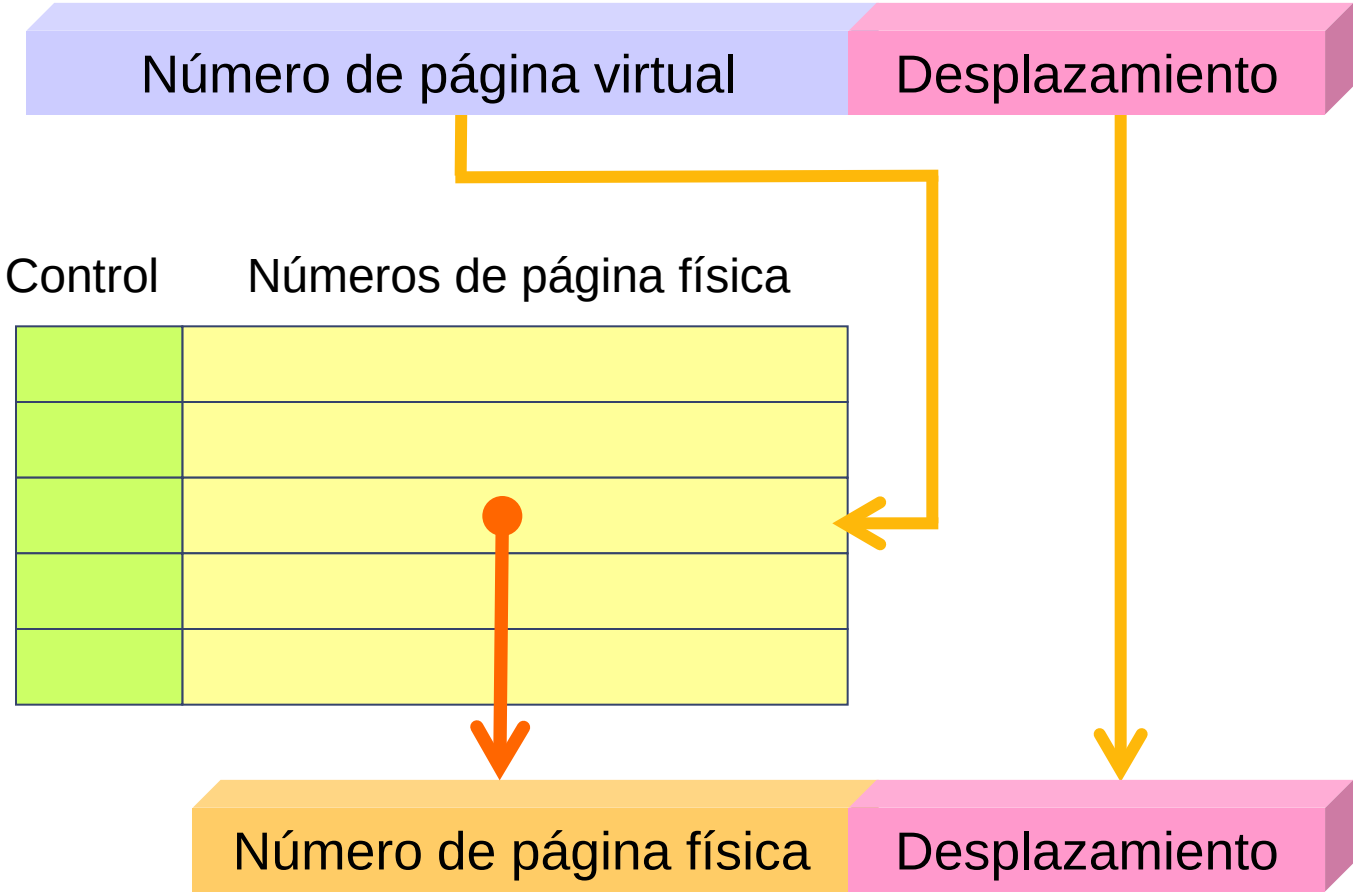
Direcciones virtuales y físicas

- Los programas trabajan con direcciones virtuales
  - ✓ Antes de salir al bus se han de traducir a direcciones físicas

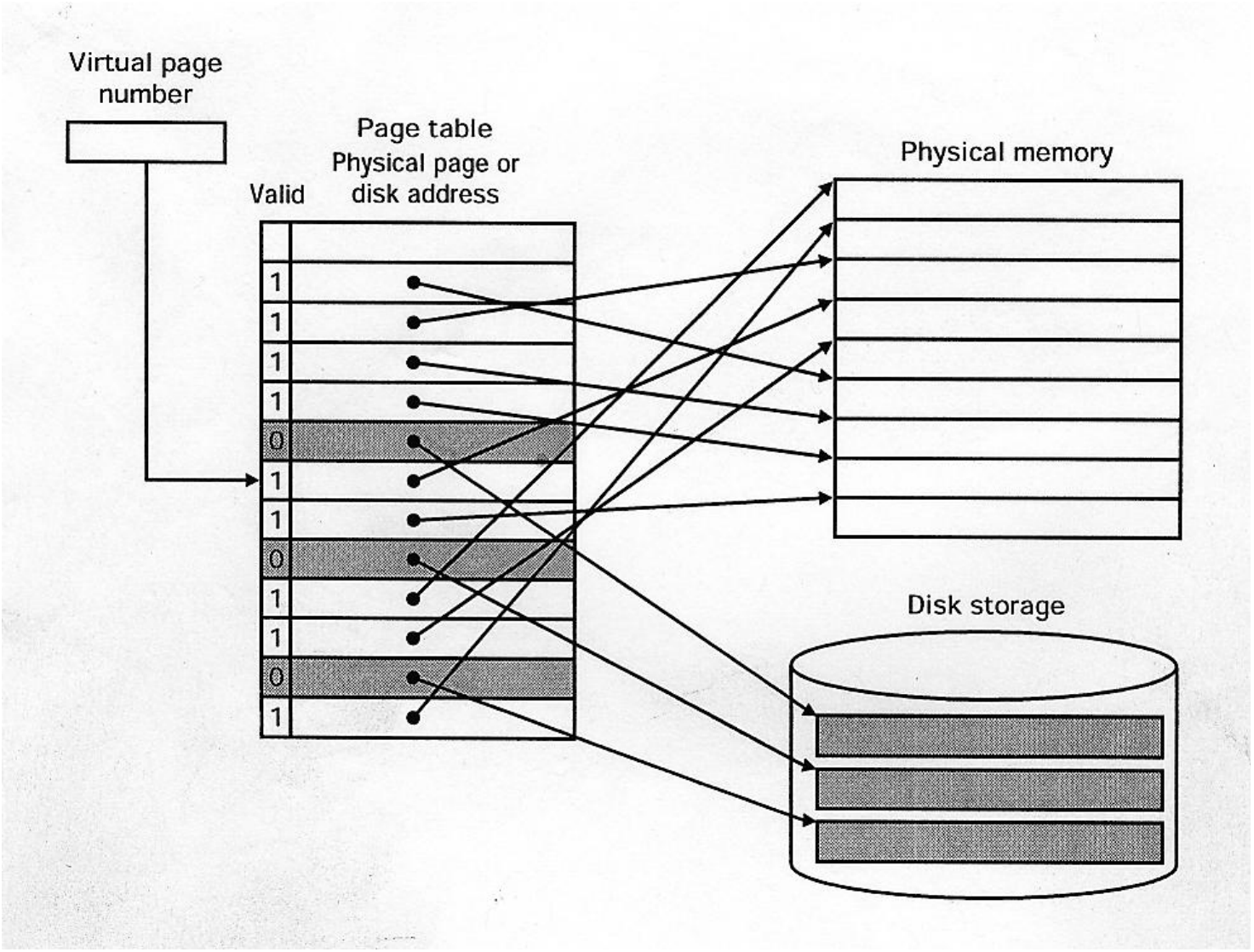


Traducción mediante una tabla de páginas

- El número de página virtual sirve de índice para acceder a la tabla y recuperar el número de página física



Interpretación de la información en la TP

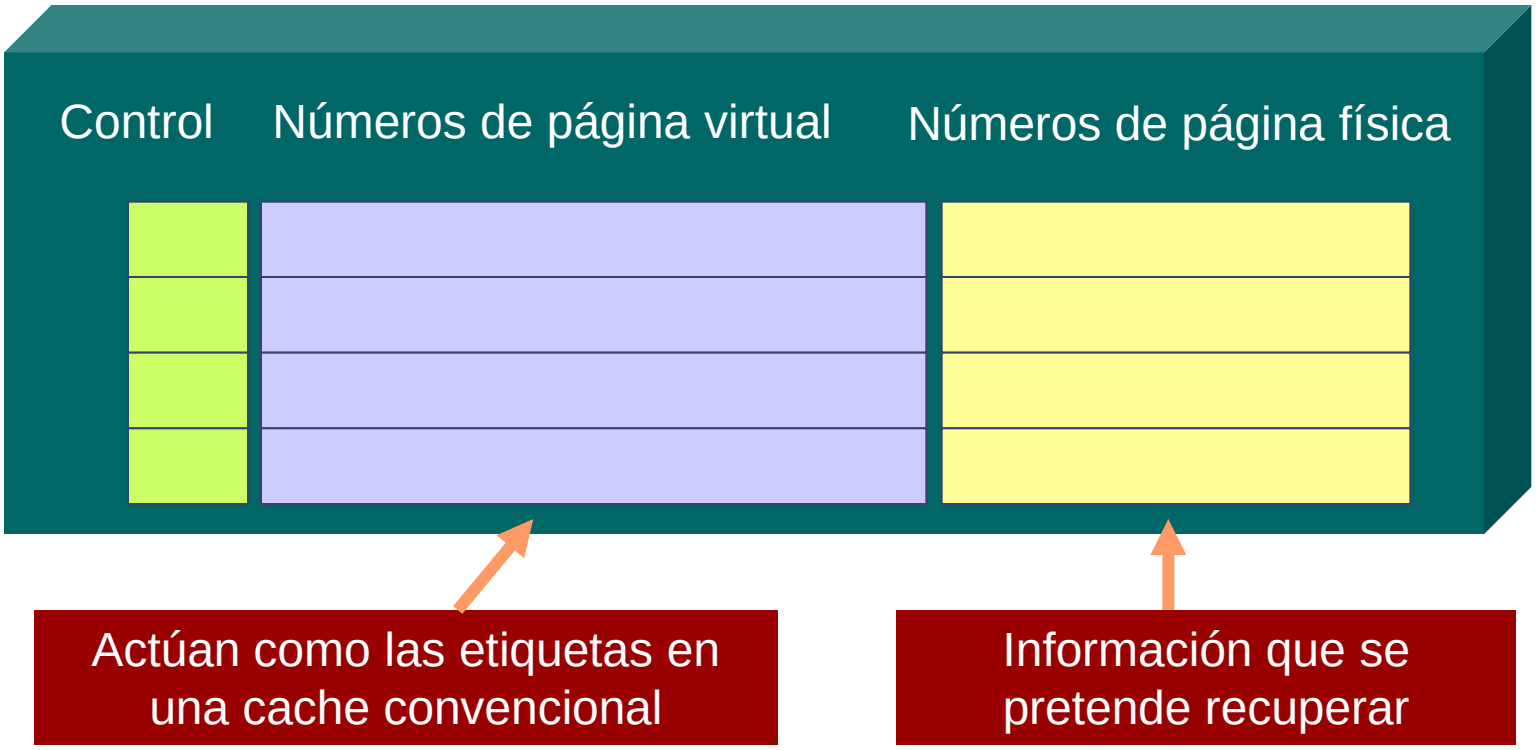


Traducción de direcciones: características

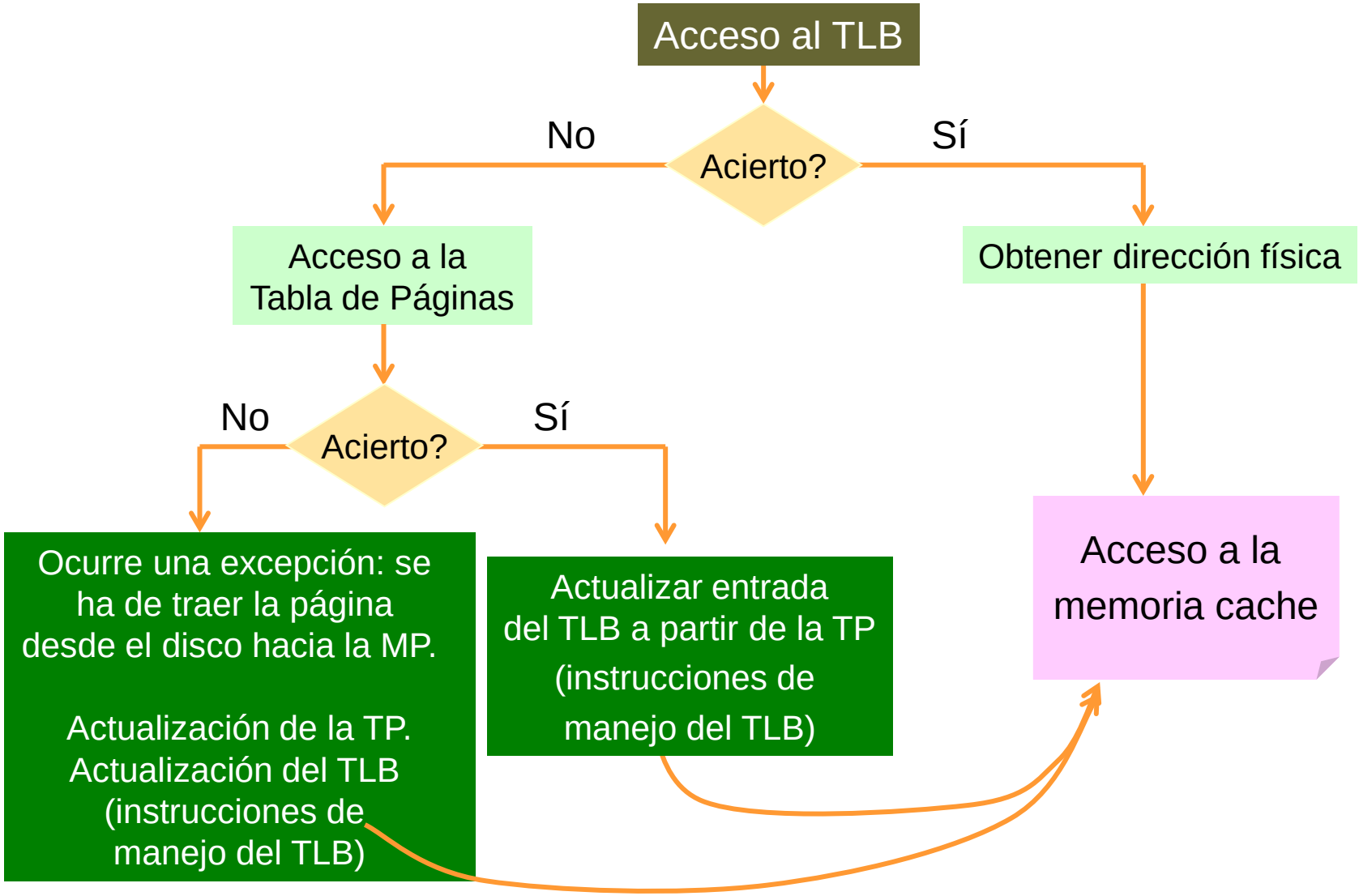
- La tabla de páginas se almacena en memoria principal
- Hay una entrada por cada página virtual
- Información almacenada en la tabla
  - ✓ Bit de válido: indica si la página está en memoria o en disco
  - ✓ Número de página física (en su caso)
  - ✓ Permisos de lectura, escritura, etc
- Problema: es necesario mucho espacio para almacenar la tabla y se utilizan muy pocas páginas
  - ✓ Solución: organización mediante tablas con estructura jerárquica que se actualizan dinámicamente

Traducción rápida de direcciones: el TLB

- Hacen falta dos accesos a MP: lentitud
- Solución: usar una "cache" de la tabla de páginas en el procesador: TLB (*Translation Lookaside Buffer*)

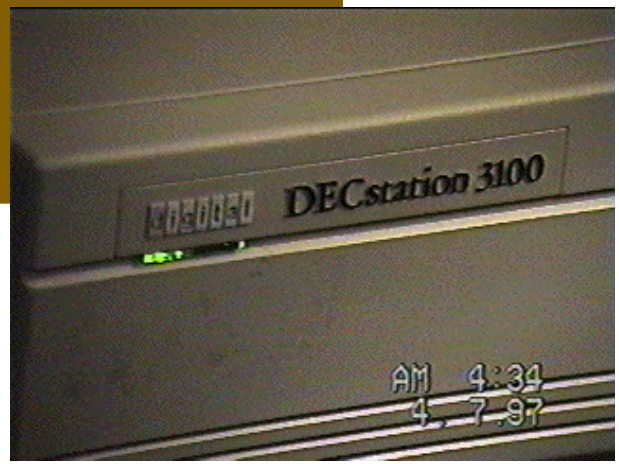


Relación entre Tabla de Pàgines (TP) y TLB



4. Ejemplo conjunto TLB y cache

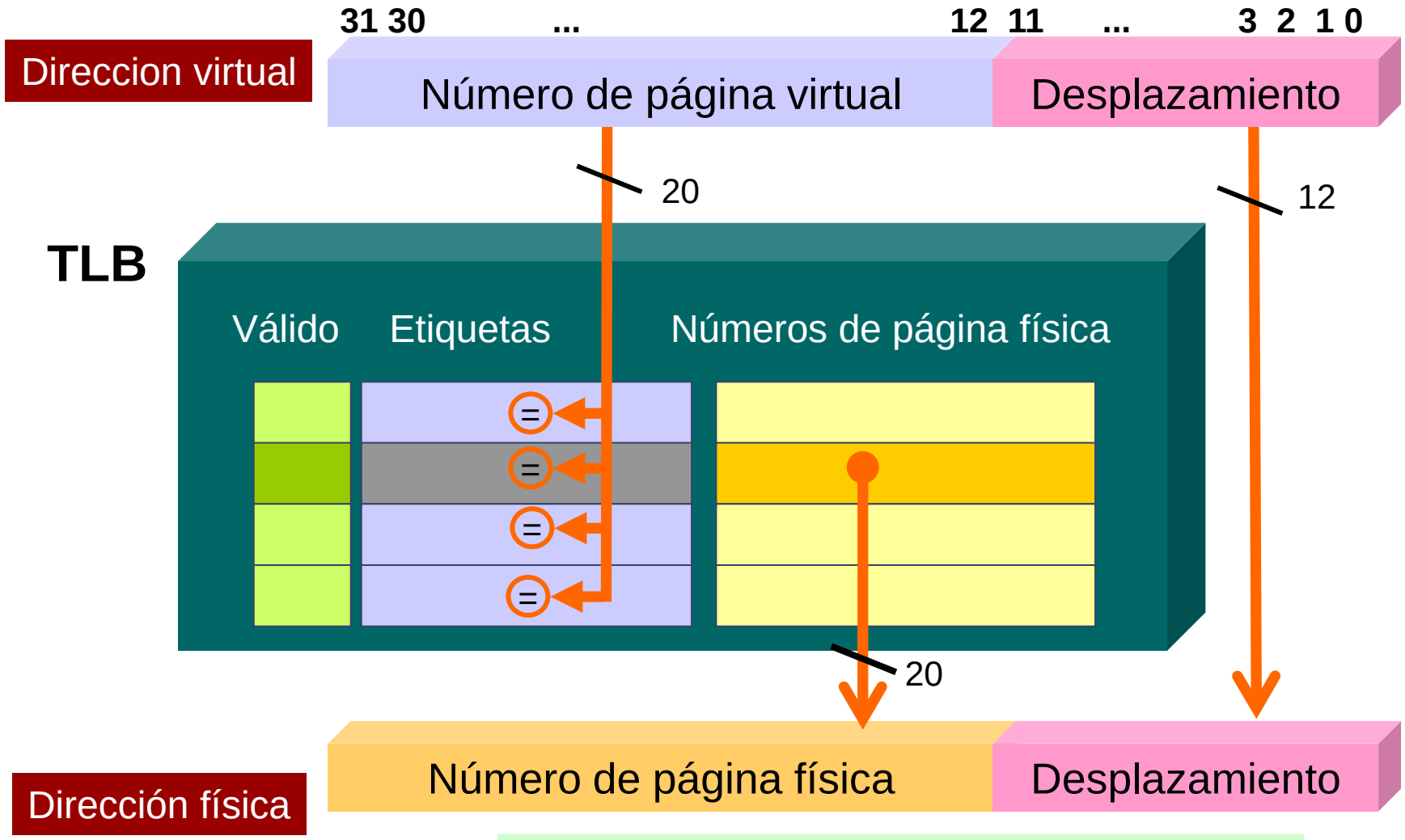
- Computador DECstation 3100



La memoria virtual en el MIPS R2000

- *Memory Management Unit* (MMU)
- Características de direccionamiento de memoria
  - ✓ Direcciones físicas: 32 bits
  - ✓ Direcciones virtuales: 32 bits
  - ✓ Tamaño de las páginas: 4 KB (2^{12} bytes)
- TLB situado dentro del procesador (*on chip*)
  - ✓ Traducción de direcciones de código y datos
  - ✓ Correspondencia completamente asociativa
  - ✓ Tiene 64 entradas: se puede traducir y acceder rápidamente a un total de 64 páginas de 4 KB de memoria virtual
  - ✓ Cada entrada tiene 64 bits de longitud
    - Control: bits de válido, *cacheable*, etc.

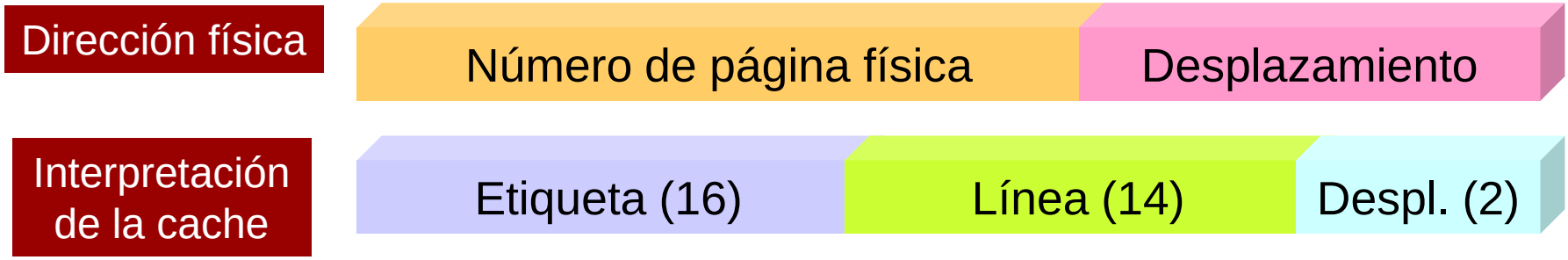
Traducción de la dirección virtual en la TLB



Esta dirección llegará a la memoria cache L1

La memoria cache en el MIPS R2000

- Configuración en la DECstation 3100
  - ✓ Nivel L1 externo al procesador
    - Caches separadas de datos y código
  - ✓ Correspondencia directa
  - ✓ Tamaño de bloque: 4 bytes (una palabra)
  - ✓ Políticas de escritura
    - Aciertos: escritura directa (*write through*)
    - Fallos: no ubicación (*no write allocate*)
- La cache representada (una de las dos) tiene 64 KB de capacidad



Interpretación de la dirección física por la cache