

# Tema 5 (II)

## Jerarquía de Memoria



Grupo ARCOS

Estructura de Computadores  
Grado en Ingeniería Informática  
Universidad Carlos III de Madrid

# Contenidos

---

## I. Memoria cache

- ▶ Introducción:
  - ▶ Acceso a bloque y principio de proximidad
- ▶ Funcionamiento
- ▶ Organización
- ▶ Estructura de la memoria caché
- ▶ Diseño de la memoria cache

# ¡ATENCIÓN!

---

- ❑ Estas transparencias son un guión para la clase
- ❑ Los libros dados en la bibliografía junto con lo explicado en clase representa el material de estudio para el temario de la asignatura
- ❑ Para la preparación de los exámenes se ha de utilizar todo el material de estudios

# Contenidos

---

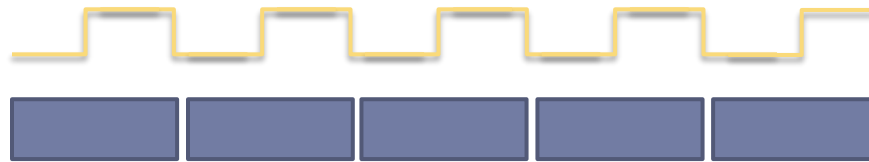
## I. Memoria cache

- ▶ **Introducción:**
  - ▶ **Acceso a bloque y principio de proximidad**
- ▶ Funcionamiento
- ▶ Organización
- ▶ Estructura de la memoria caché
- ▶ Diseño de la memoria cache

# Característica de la memoria principal

---

- ▶ Se premia el acceso a posiciones consecutivas de memoria
  - ▶ Ejemplo 1: acceder a 5 posiciones de memoria **individuales**



- ▶ Ejemplo 2: acceder a 5 posiciones de memoria **consecutivas**



# Característica de los accesos a memoria

---

- ▶ “Principio de proximidad de referencias”:

En el curso de la ejecución de un programa, las referencias a memoria tienden a estar agrupadas por:

- ▶ proximidad espacial
  - ▶ Secuencia de instrucciones
- ▶ proximidad temporal
  - ▶ Bucles

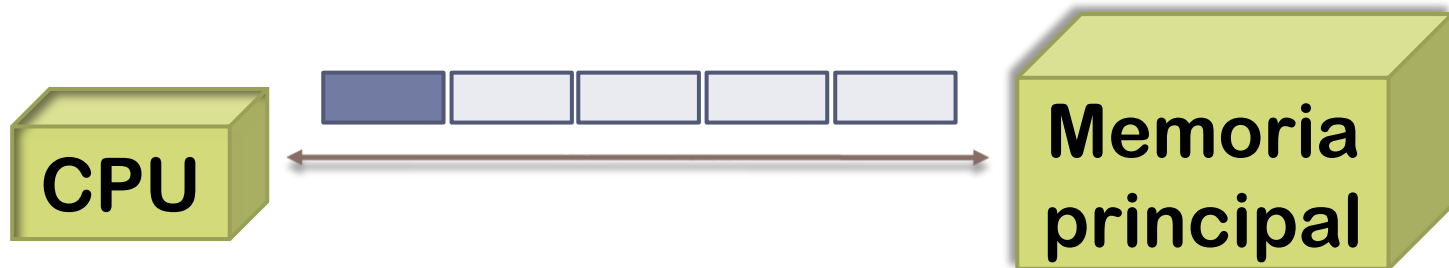
```
.data
vector: .space 4*1024

.text
main: li $t0 0
      la $t1 vector
      b2: bge $t0 1024 finb2
      mult $t2 $t0 4
      add $t2 $t1 $t2
      sw $t0 ($t2)
      add $t0 $t0 1
      b b2
      finb2: jr $ra
```

# Objetivo: aprovechar los accesos contiguos

---

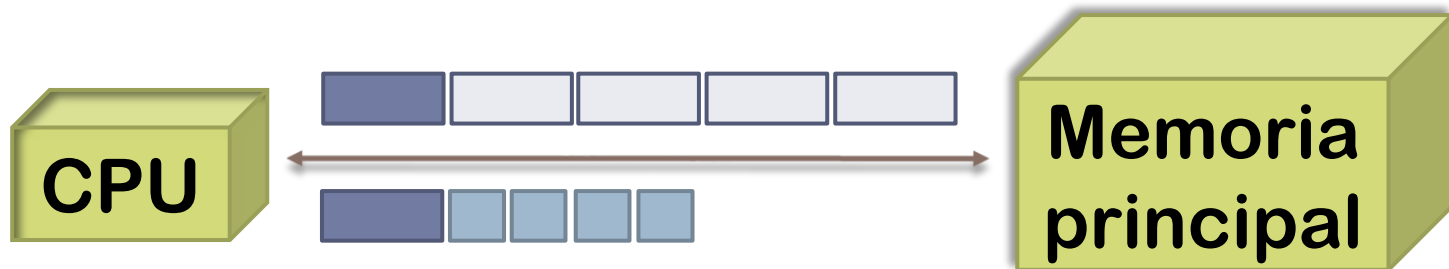
- ▶ Si cuando accedo a una posición de memoria transfiero los datos de esa posición, no aprovecho los posibles accesos contiguos.



# Objetivo: aprovechar los accesos contiguos

---

- ▶ Si cuando se accede a una posición de memoria se transfiere esos datos y los contiguos, si aprovecho los accesos contiguos

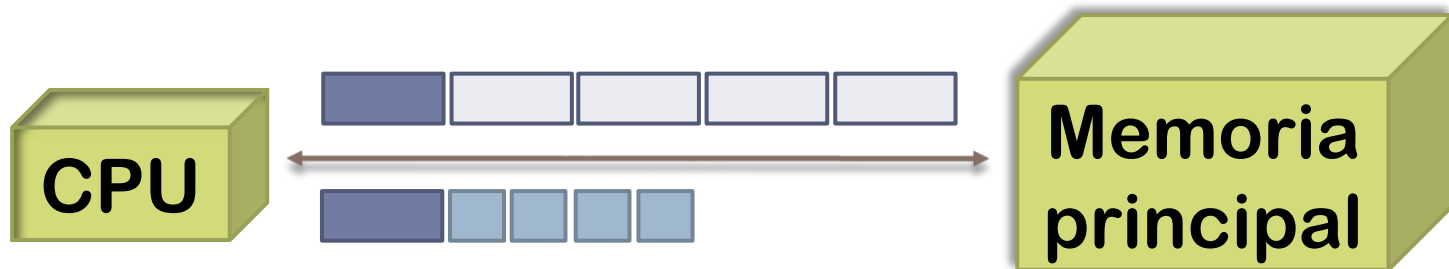




# Objetivo: aprovechar los accesos contiguos

---

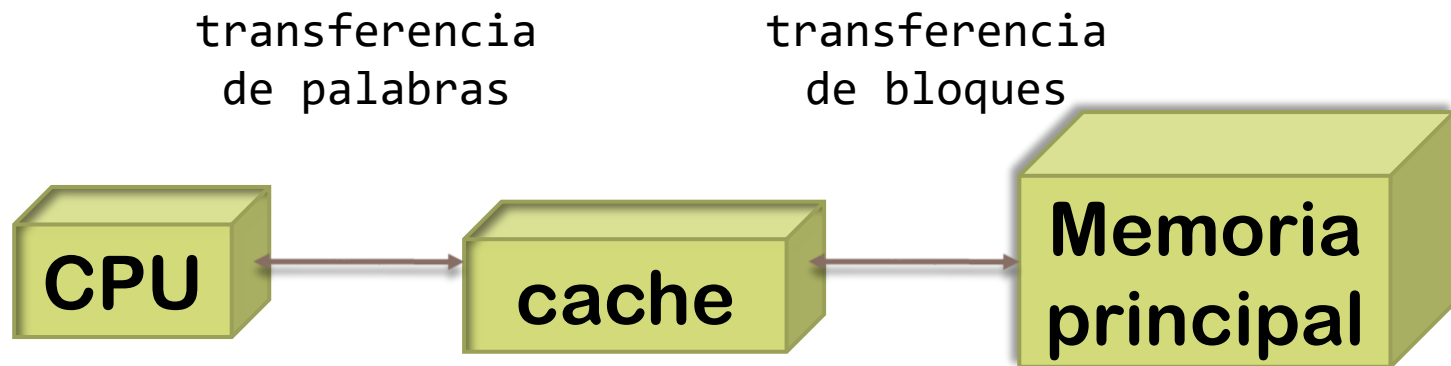
- ▶ Si cuando se accede a una posición de memoria se transfiere esos datos y los contiguos, si aprovecho los accesos contiguos
  - ▶ Transfiero de la memoria principal un bloque de palabras
  - ▶ ¿Dónde almaceno las palabras del bloque?



# Memoria Cache

---

- ▶ Cantidad pequeña de memoria rápida
- ▶ Está entre la memoria principal normal y la CPU



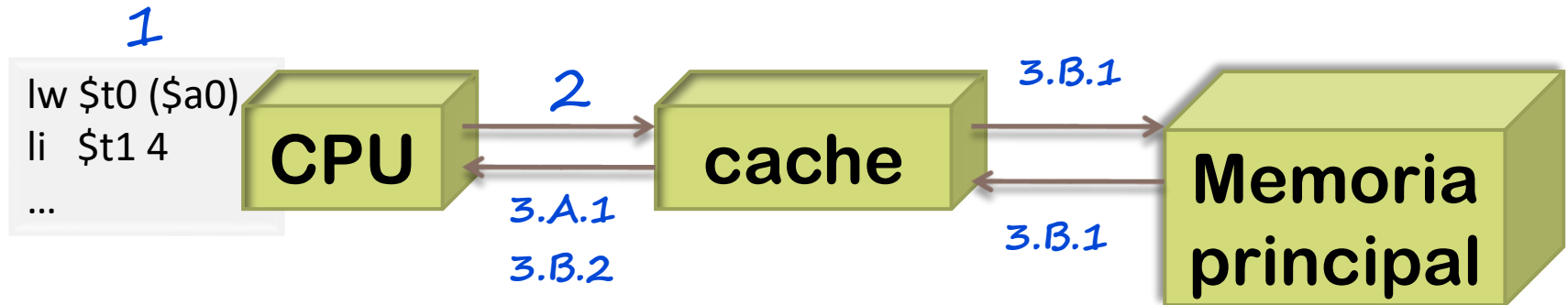
# Contenidos

---

## I. Memoria cache

- ▶ Introducción:
  - ▶ Acceso a bloque y principio de proximidad
- ▶ **Funcionamiento**
- ▶ Organización
- ▶ Estructura de la memoria caché
- ▶ Diseño de la memoria cache

# Funcionamiento general



1. La CPU solicita contenidos de una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ **Si está,**
    - 3.A.1** Se la sirve a la CPU desde la cache (rápidamente).
  - ▶ **Si no está,**
    - 3.B.1** La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2** Después la cache entrega los datos pedidos a la CPU.

# Rendimiento general

---

$$T_m = P_{ca} * T_{ca} + (1-P_{ca}) * (T_{mp} + T_{ca}) \sim P_{ca} * T_{ca} + (1-P_{ca}) * (T_{mp})$$

1. La CPU solicita contenidos de una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ **Si está,**
    - 3.A.1 Se la sirve a la CPU desde la cache (rápidamente).
  - ▶ **Si no está,**
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después la cache entrega los datos pedidos a la CPU.

# Ejemplo

---

$$T_m = P_{ca} * T_{ca} + (1-P_{ca}) * (T_{mp} + T_{ca}) \sim \\ P_{ca} * T_{ca} + (1-P_{ca}) * (T_{mp})$$

1. Tca: Tiempo de acceso a caché -> **10** ns
2. Tmp: Tiempo de acceso a memoria principal -> **120** ns
3. Pca: Probabilidad de estar en caché -> **X = 0.1, 0.2, ..., 0.9, 1.0**  
**10%, 20%, ..., 90%, 100%**

# Ejemplo

---

$$T_m = X * 10 + (1-X) * (120 + 10) \sim$$

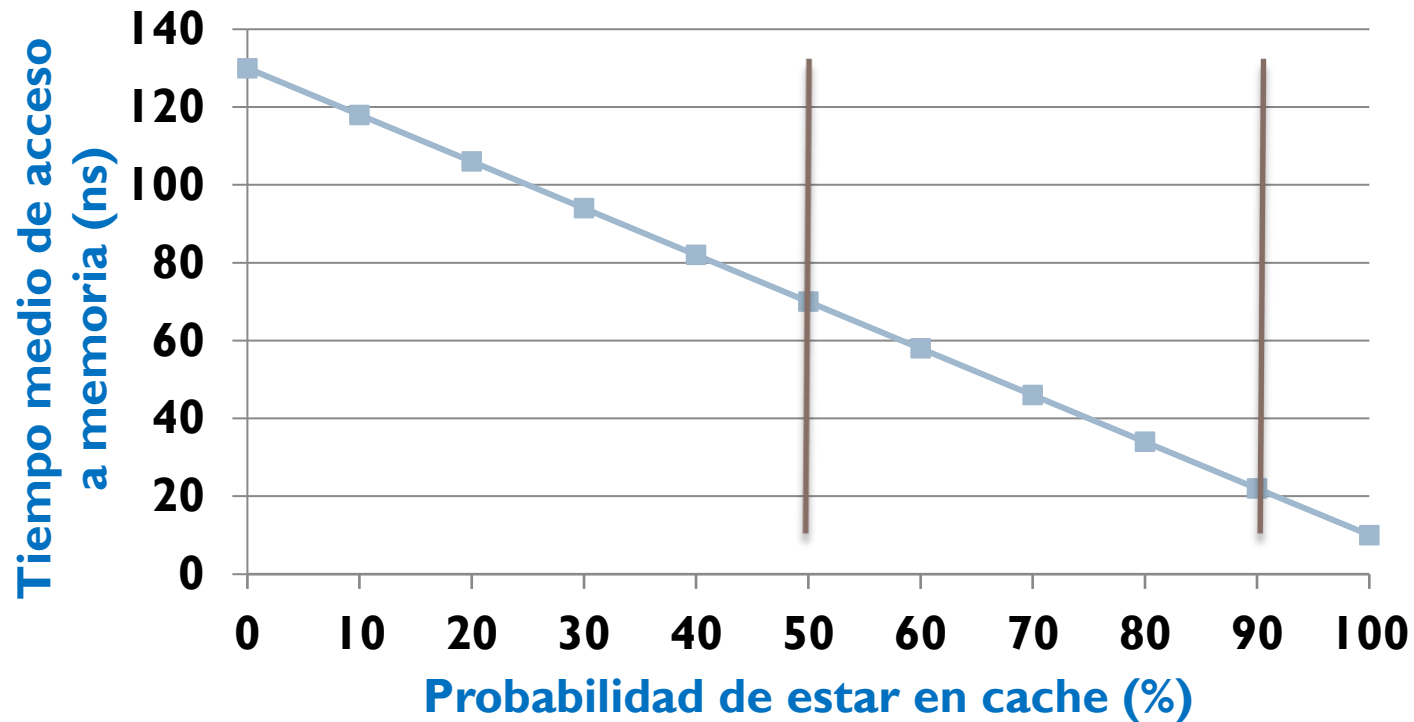
$$T_m = X * 10 + (1-X) * (120) = 120 - 110 * X$$

1. Tca: Tiempo de acceso a caché -> **10** ns
2. Tmp: Tiempo de acceso a memoria principal -> **120** ns
3. Pca: Probabilidad de estar en caché -> **X = 0.1, 0.2, ..., 0.9, 1.0**  
**10%, 20%, ..., 90%, 100%**

# Ejemplo

$$T_m = X * 10 + (1-X) * (120 + 10) \sim$$

$$T_m = X * 10 + (1-X) * (120) = 120 - 110 * X$$





# Ejercicio

## ▶ Datos:

- ▶ Tmp: 120 ns

## ▶ Tiempos de acceso: *!?*

```
int vector[1024];

int main ( void ) {
    int i;
    for (i = 0; i<1024; i++)
        vector[i] = i;
}
```

```
.data
vector: .space 4*1024

.text
.globl main

main: li $t0 0
      la $t1 vector
      b2: bge $t0 1024 finb2
      mult $t2 $t0 4
      add $t2 $t1 $t2
      sw $t0 ($t2)
      add $t0 $t0 1
      b b2
      finb2: jr $ra
```

# Ejercicio (sol.)

## ▶ Datos:

- ▶ Tmp: 120 ns

## ▶ Tiempos de acceso:

### ▶ Por datos:

- ▶  $N_a = 1.024$
- ▶  $T = 1.024 * 120 = 122.880$

### ▶ Por instrucciones:

- ▶  $N_a = 2 + 6 * 1.024 + 2 = 6.148$
- ▶  $T = 6.148 * 120 = 737.760$

```
int vector[1024];

int main ( void ) {
    int i;
    for (i = 0; i<1024; i++)
        vector[i] = i;
}
```

```
.data
vector: .space 4*1024

.text
.globl main

main: li $t0 0
      la $t1 vector
      b2: bge $t0 1024 finb2
      mult $t2 $t0 4
      add $t2 $t1 $t2
      sw $t0 ($t2)
      add $t0 $t0 1
      b b2
      finb2: jr $ra
```

T. sin caché = 860.640 ns

# Ejercicio

## ▶ Datos:

- ▶ Tca: 10 ns
- ▶ Tmp: 120 ns
- ▶ Tamaño línea: 8 palabras

## ▶ Tiempos de acceso: *!?*

```
int vector[1024];

int main ( void ) {
    int i;
    for (i = 0; i<1024; i++)
        vector[i] = i;
}
```

```
.data
vector: .space 4*1024

.text
.globl main

main: li $t0 0
      la $t1 vector
      b2: bge $t0 1024 finb2
      mult $t2 $t0 4
      add $t2 $t1 $t2
      sw $t0 ($t2)
      add $t0 $t0 1
      b b2
      finb2: jr $ra
```

# Ejercicio (sol.)

## ▶ Datos:

- ▶ Tca: 10 ns
- ▶ Tmp: 120 ns
- ▶ Tamaño línea: 8 palabras

## ▶ Tiempos de acceso:

### ▶ Por instrucciones:

- ▶  $N_a = 2 + 6 * 1.024 + 1 = 6.148$ 
  - Primer acceso: 120 ns
  - Sigüientes  $1 + 6 * 1.024$ : 10 ns
  - Último acceso: 120 ns
- ▶  $T = 120 + 6.145 * 10 + 120 = 61.690$

### ▶ Por datos:

- ▶  $N_a = 1.024$ 
  - Primer acceso: 120 ns
  - Sigüientes 7 pal.: 10 ns
- ▶  $T = 1 * 128 * 120 + 7 * 128 * 10 = 33.280$

```
int vector[1024];

int main ( void ) {
    int i;
    for (i = 0; i<1024; i++)
        vector[i] = i;
}
```

```
.data
vector: .space 4*1024

.text
.globl main

main: li $t0 0
      la $t1 vector
      b2: bge $t0 1024 finb2
      mult $t2 $t0 4
      add $t2 $t1 $t2
      sw $t0 ($t2)
      add $t0 $t0 1
      b b2
      finb2: jr $ra
```

T. sin caché = 860.640 ns  
T. con caché = 94.970 ns

Una mejora de casi x9 en este ejemplo

# Contenidos

---

## I. Memoria cache

- ▶ Introducción:
  - ▶ Acceso a bloque y principio de proximidad
- ▶ Funcionamiento
- ▶ **Organización**
- ▶ Estructura de la memoria caché
- ▶ Diseño de la memoria cache

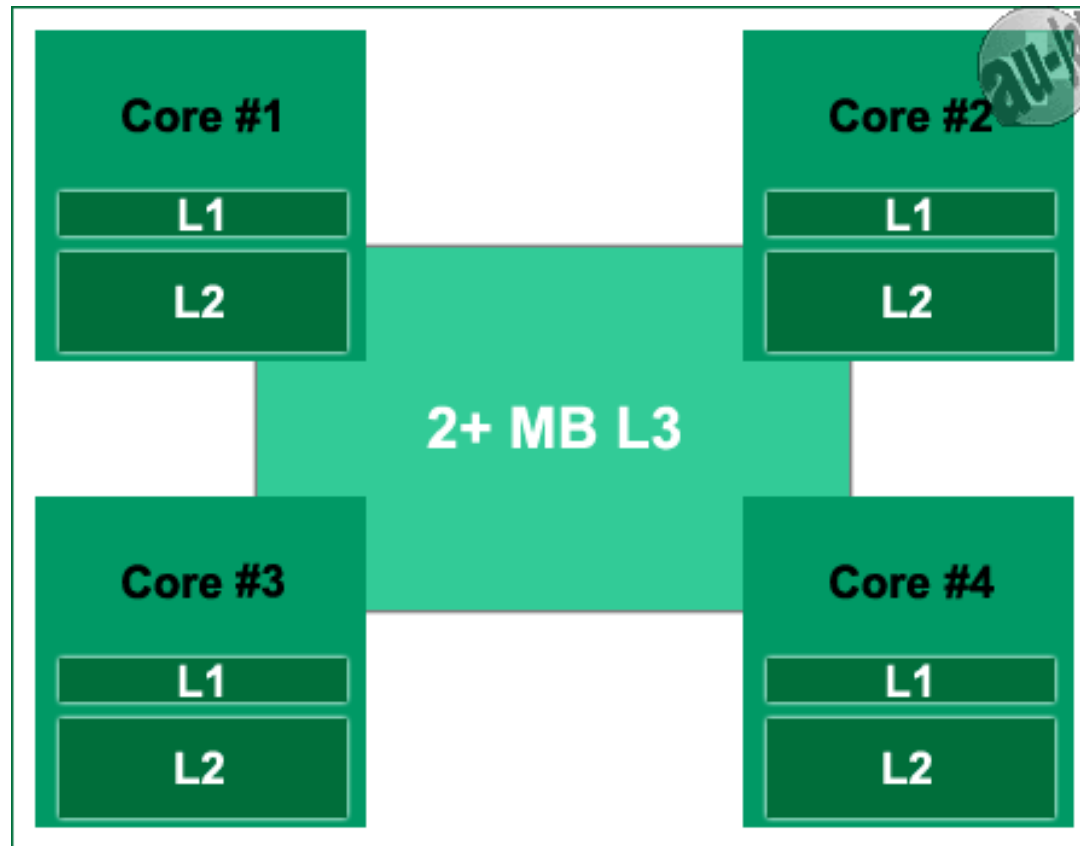


# Memoria Cache: niveles

---

- ▶ Es habitual encontrar tres niveles:
  - ▶ **L1 o nivel 1:**
    - ▶ Caché interna: la más cercana a la CPU
    - ▶ Tamaño pequeño (8KB-128KB) y máxima velocidad
  - ▶ **L2 o nivel 2:**
    - ▶ Cache externa: entre L1 y L3 (o entre L1 y M.Principal)
    - ▶ Tamaño mediano (256KB – 4MB) y menor velocidad que L1
  - ▶ **L3 o nivel 3:**
    - ▶ último nivel antes de M. Principal
    - ▶ Tamaño mayor y menor velocidad que L2

# Ejemplo: AMD quad-core

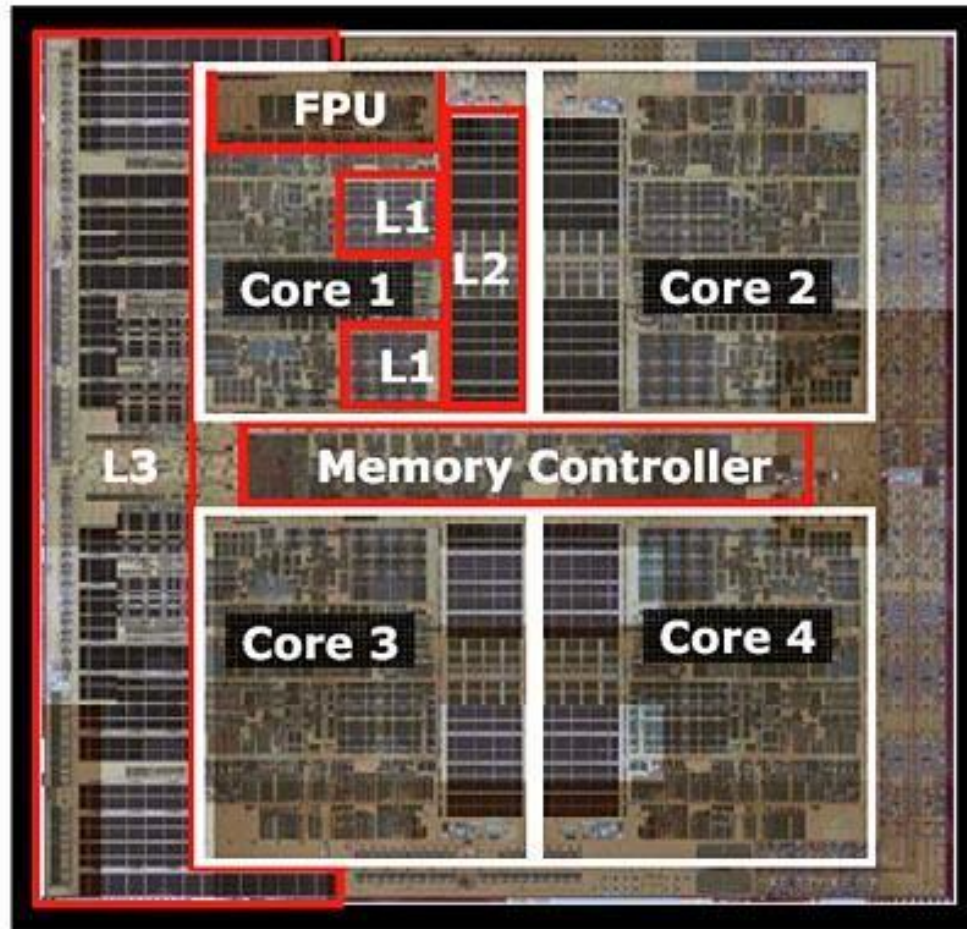


Quad-Core CPU mit gemeinsamen L3-Cache



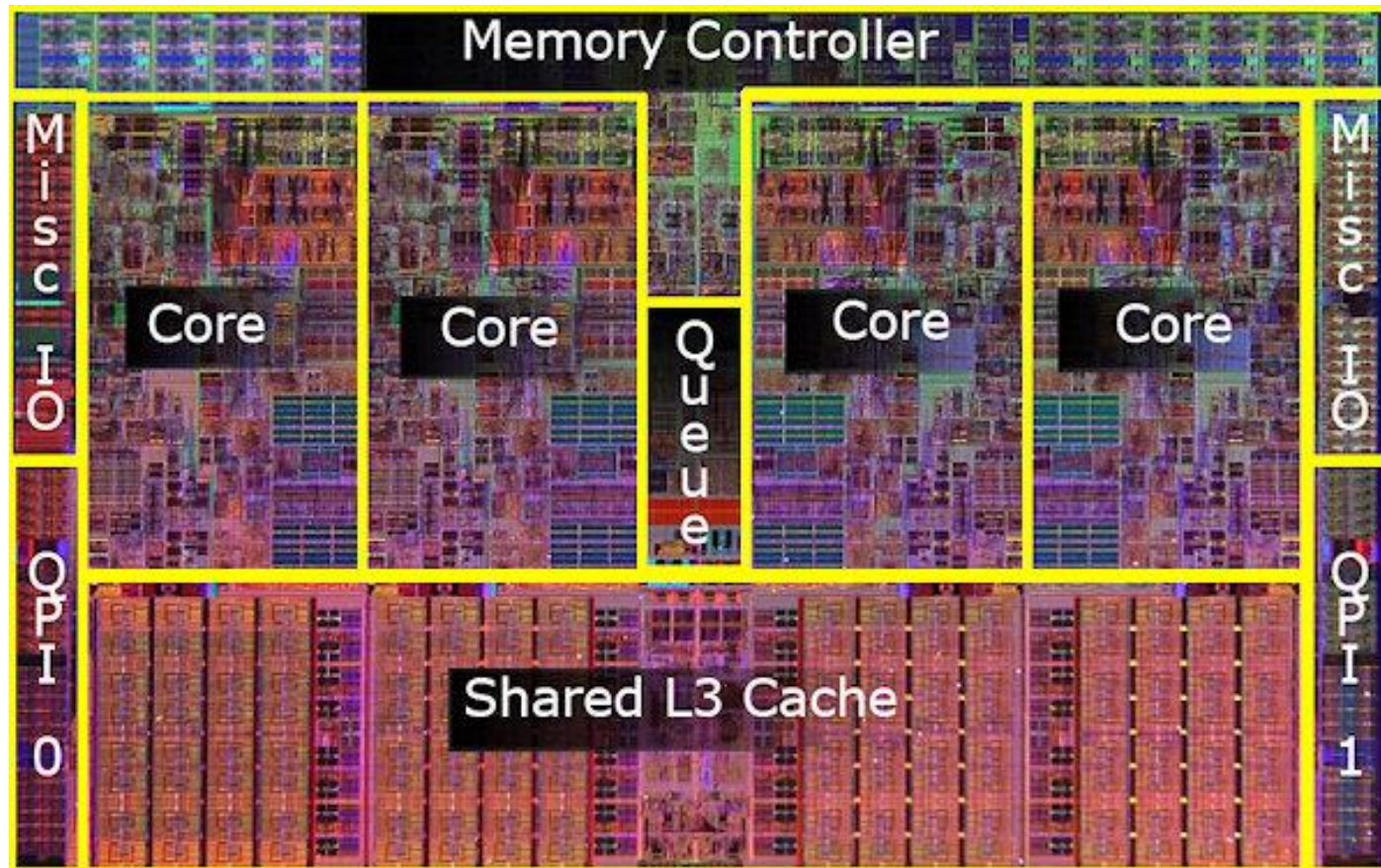
# Ejemplo: AMD quad-core

---



# Ejemplo: Intel Core i7

---



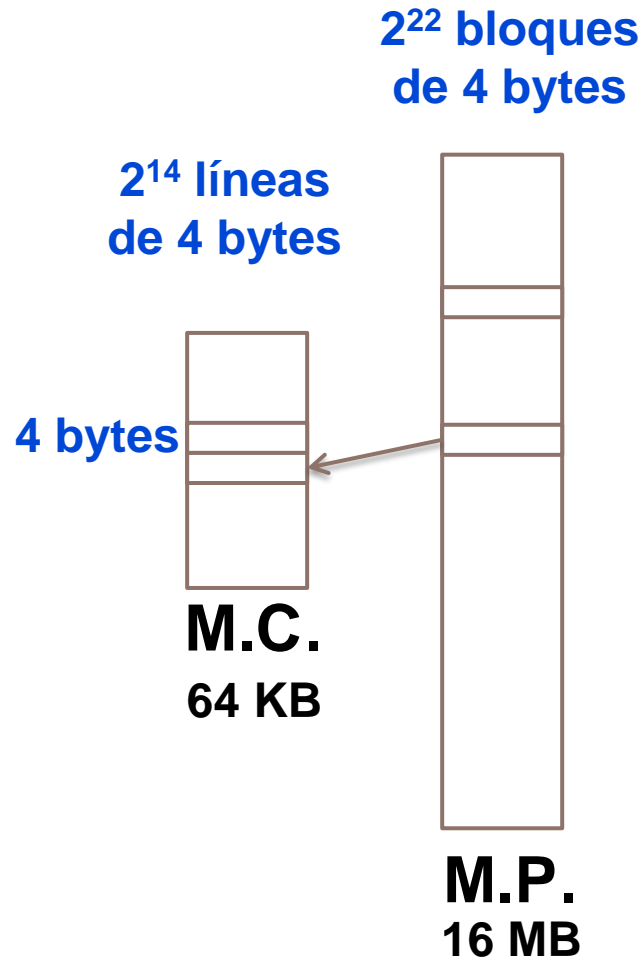
# Contenidos

---

## I. Memoria cache

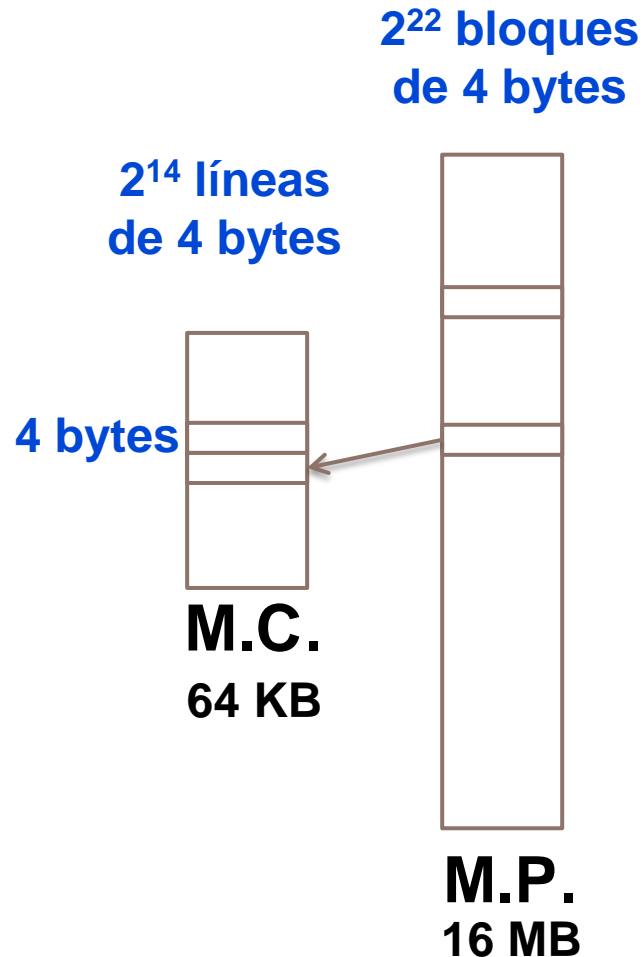
- ▶ Introducción:
  - ▶ Acceso a bloque y principio de proximidad
- ▶ Funcionamiento
- ▶ Organización
- ▶ **Estructura de la memoria caché**
- ▶ **Diseño de la memoria cache**

# Estructura y diseño de la cache



- ▶ Se divide la M.P. y la M.C. en bloques de igual tamaño
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)

# Estructura y diseño de la cache

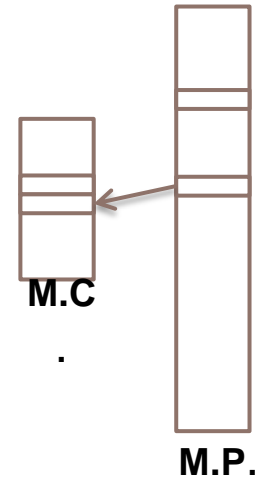


- ▶ Se divide la M.P. y la M.C. en bloques de igual tamaño
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)
- ▶ En el diseño se determina:
  - ▶ **Tamaño**
  - ▶ **Función de correspondencia**
  - ▶ **Algoritmo de sustitución**
  - ▶ **Política de escritura**
- ▶ Es habitual distintos diseños para L1, L2, ...

# Diseño de la cache

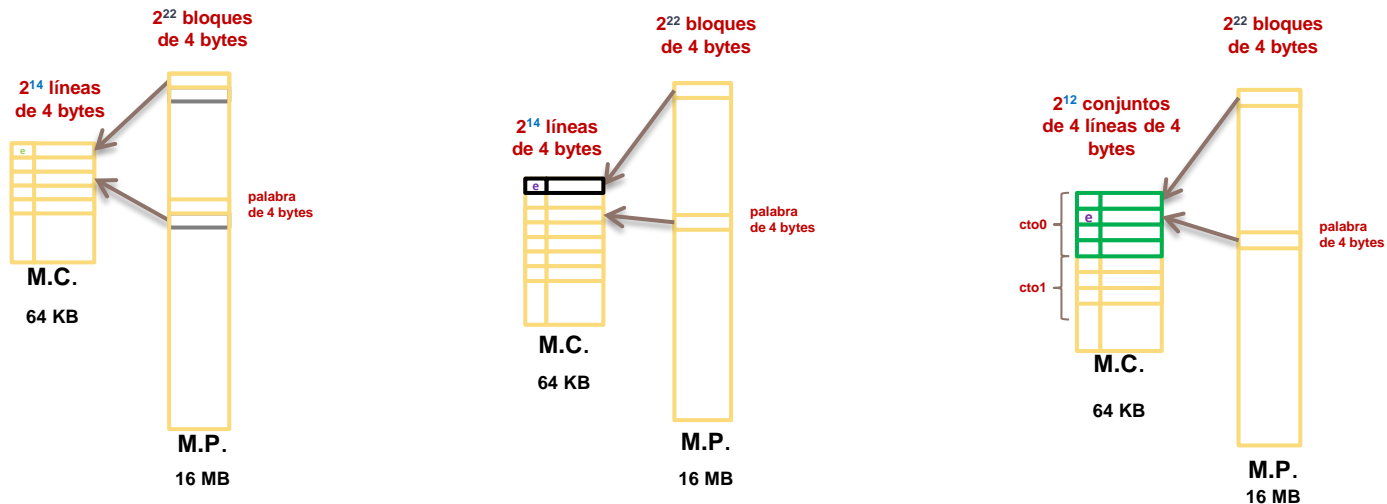
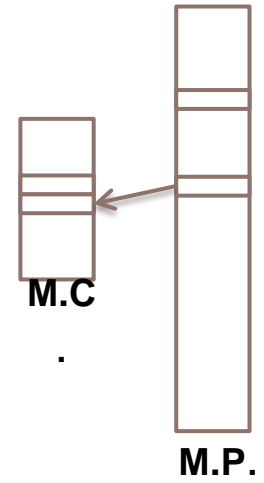
---

- ▶ **Tamaño**
  - ▶ El tamaño total y de los bloques en los que se organiza
  - ▶ Por estudios sobre códigos muy usados se determina



# Diseño de la cache

- ▶ **Tamaño**
  - ▶ El tamaño total y de los bloques en los que se organiza
  - ▶ Por estudios sobre códigos muy usados se determina
- ▶ **Función de correspondencia**
  - ▶ ¿Dónde está en caché los datos de una posición de M.P.?
  - ▶ Ej.: Directa, Asociativa y asociativa por conjuntos





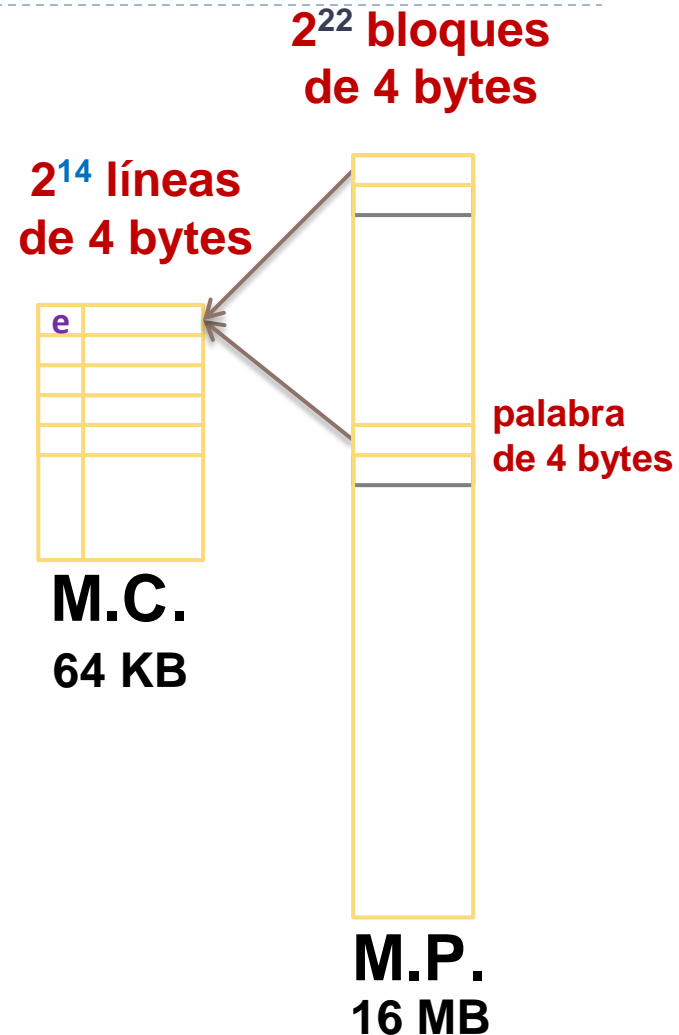
# Función de correspondencia

## ▶ Directa:

- ▶ Cada bloque de M.P. le corresponde una sola línea de caché (siempre la misma)
- ▶ La dirección de M.P. la determina:

22-14	14	2
etiqueta	línea	palabra

- ▶ Si en 'línea' está 'etiqueta', entonces está el bloque en caché
- ▶ Ej: línea caché      bloque de M.P. asociado.  
0                      0,  $2^{14}$ ,  $2 * 2^{14}$ ,  $3 * \dots$   
1                      1,  $2^{14} + 1$ , ...
- ▶ [V] simple, poco costosa  
[I] mal patrón acceso → ↑ fallos

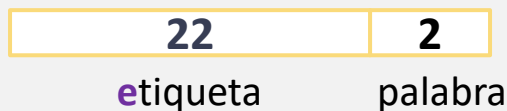




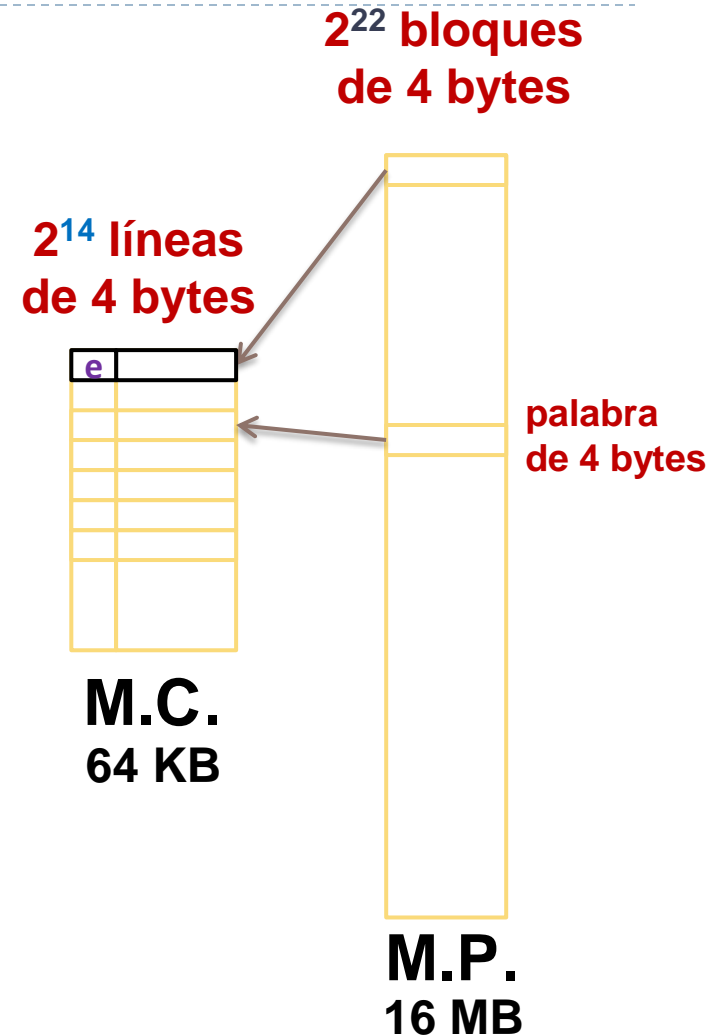
# Función de correspondencia

## ▶ Asociativa:

- ▶ Un bloque de M.P. puede cargarse en cualquier línea de caché
- ▶ La dir. de M.P. se interpreta como:



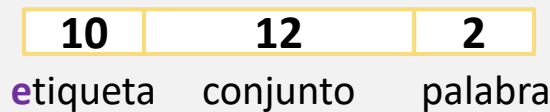
- ▶ Si hay un línea con 'etiqueta' en la caché, está allí el bloque
- ▶ [V] indep. del patrón de acceso  
[I] búsqueda costosa



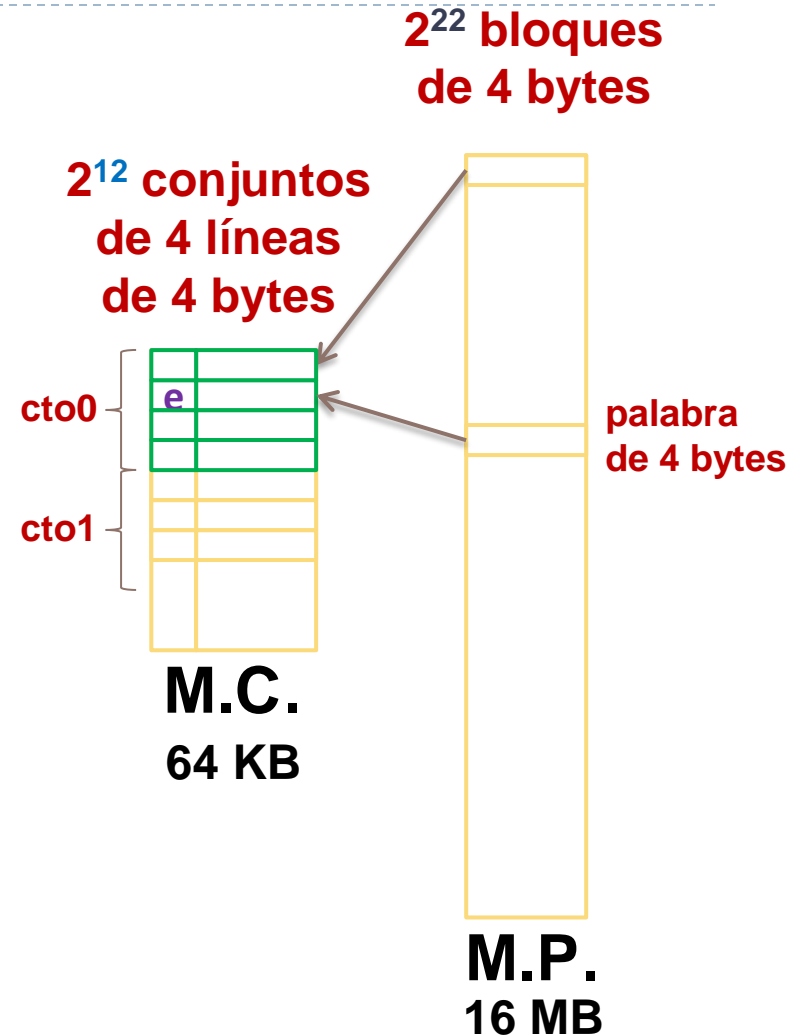
# Función de correspondencia

## ▶ Asociativa por conjuntos:

- ▶ Un bloque de M.P. puede cargarse en cualquier línea de caché (vía) de un conjunto determinado
- ▶ La dir. de M.P. se interpreta como:



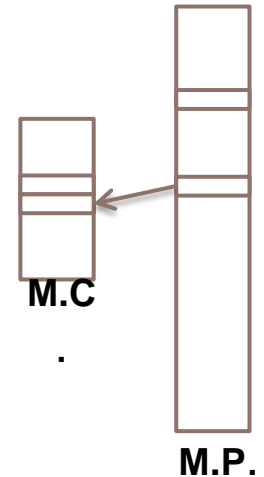
- ▶ Si hay un línea con 'etiqueta' en el conjunto 'conjunto', está allí el bloque en caché
- ▶ [V] lo mejor de directa y asociativa  
[I] búsqueda menos costosa



# Diseño de la cache

---

- ▶ **Tamaño**
  - ▶ El tamaño total y de los bloques en los que se organiza
  - ▶ Por estudios sobre códigos muy usados se determina
- ▶ **Función de correspondencia**
  - ▶ ¿Dónde está en caché los datos de una posición de M.P.?
  - ▶ Ej.: Directa, Asociativa y asociativa por conjuntos
- ▶ **Algoritmo de sustitución**
  - ▶ Si la caché está llena y hay un fallo, ¿Qué bloque se quita?
  - ▶ Ej.: LRU, FIFO, LFU, etc.



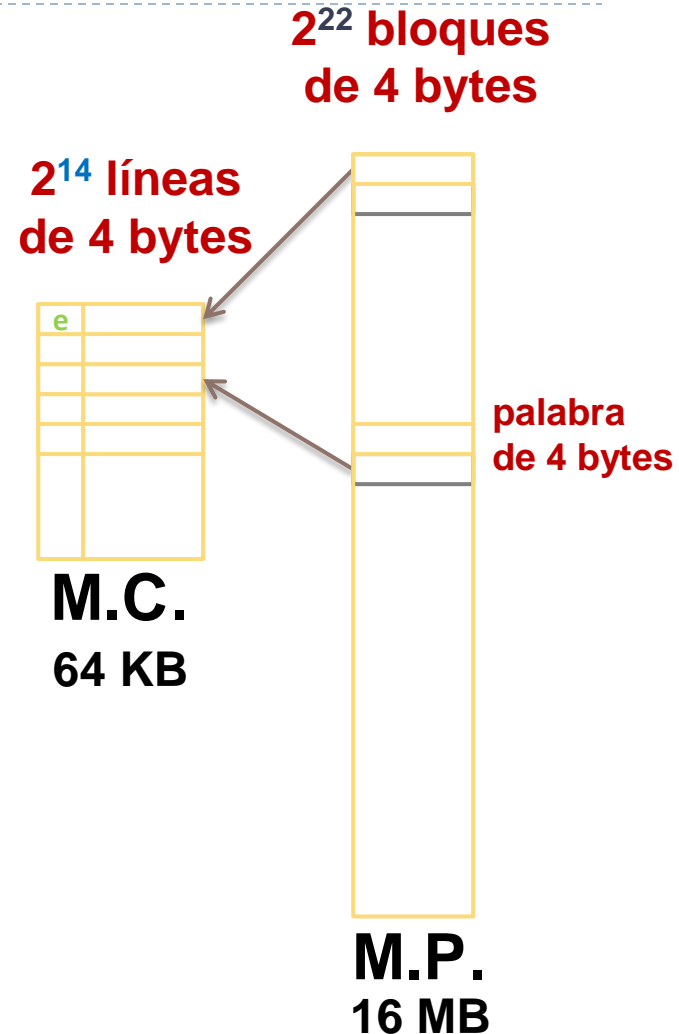
# Algoritmo de sustitución

## ▶ Correspondencia directa:

- ▶ No hay elección posible
- ▶ Para cada bloque/palabra solo hay una posible línea

## ▶ Correspondencia asociativa y asociativa por conjuntos

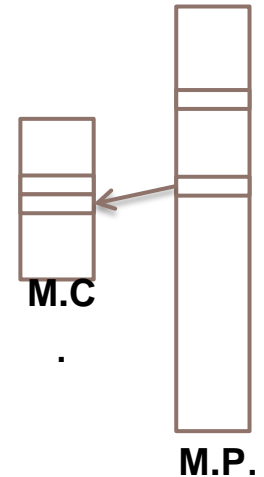
- ▶ Algoritmos implementados en hardware para buscar velocidad y mínimo número de fallos en cache
- ▶ Elección de bloque/palabra según:
  - ▶ **LRU**: utilizado menos recientemente
  - ▶ **FIFO**: primero en entrar, primero en salir
  - ▶ **LFU**: utilizado menos frecuentemente
  - ▶ **Aleatoria**: una línea al azar



# Diseño de la cache

---

- ▶ **Tamaño**
  - ▶ El tamaño total y de los bloques en los que se organiza
  - ▶ Por estudios sobre códigos muy usados se determina
- ▶ **Función de correspondencia**
  - ▶ ¿Dónde está en caché los datos de una posición de M.P.?
  - ▶ Ej.: Directa, Asociativa y asociativa por conjuntos
- ▶ **Algoritmo de sustitución**
  - ▶ Si la caché está llena y hay un fallo, ¿Qué bloque se quita?
  - ▶ Ej.: LRU, FIFO, LFU, etc.
- ▶ **Política de escritura**
  - ▶ Las escrituras se pueden hacer solo en el bloque correspondiente de M.C. o también en el de M.P. asociado
  - ▶ Ej.: write-back (velocidad) o write-through (coherencia -multicore, I/O DMA, etc.-)



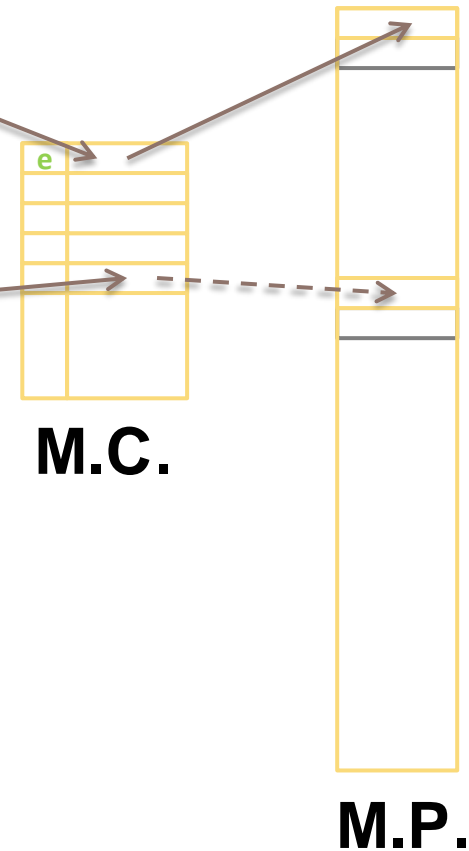
# Política de escritura

## ▶ Escritura **inmediata**:

- ▶ La escritura se hace tanto en M.P. como en cache
- ▶ [V] Coherencia
- ▶ [I] Mucho tráfico

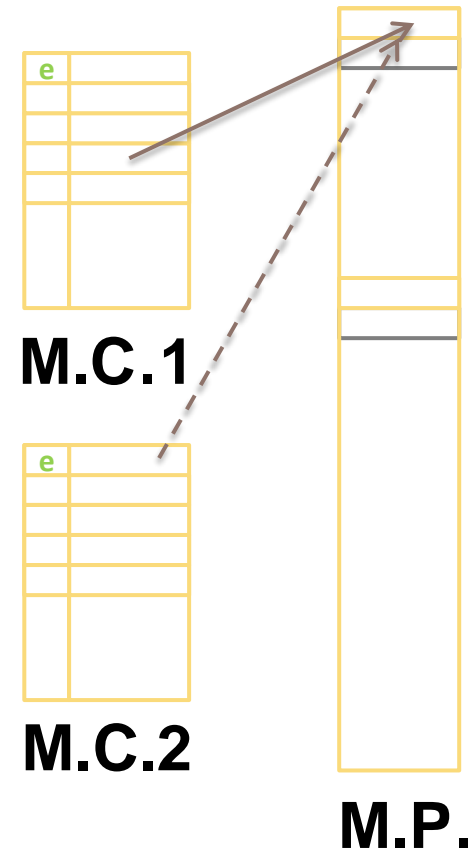
## ▶ Escritura **diferida**:

- ▶ La escritura solo se hace en la caché, indicando en un bit que no está volcada la línea en M.P.
- ▶ Al sustituir el bloque (o cuando ↓ tráfico con M.P.) se escribe en M.P.
- ▶ [V] Velocidad
- ▶ [I] Coherencia + inconsistencia



# Política de escritura

- ▶ Ej: CPU multicore con caché por core
  - ▶ Las escrituras en caché solo son vistas por un core
  - ▶ Si cada core escribe sobre una misma palabra, ¿cuál es el resultado final?
- ▶ Ej: módulo de E/S con acceso directo a M.P.
  - ▶ Actualización por DMA puede no ser coherente
- ▶ Porcentaje de referencias a memoria para escritura del orden del 15%.



# Ejercicio (1 / 2)

---

- ▶ Considere un computador de 32 bits con las siguientes características:
  - Memoria física instalada de 256 MB con un tiempo de acceso de 70 ns.
  - Direccionamiento de la memoria por bytes.
  - Tamaño de la memoria caché de 64 KB.
  - Tamaño de la línea 64 bytes.
  - La caché es asociativa por conjuntos de 4 vías.
  - El tiempo de acceso a la caché es de 5 ns y el tiempo de penalización en caso de fallo es de 100 ns.



# Ejercicio (2/2)

---

► Se pide:

- a) ¿Cuántos bloques tiene la memoria principal? (0,25 puntos)
- b) ¿Cuántos conjuntos tiene la memoria caché? (0,25 puntos)
- c) Dada una dirección de memoria, indique qué partes de la dirección se utilizan para identificar la etiqueta, el conjunto y el byte dentro de la línea.  
Indique también el número de bits de cada parte. (1 punto)
- d) Dada la siguiente dirección de memoria  
0000 0011 1100 0011 0000 0000 1111 1000.  
En caso de encontrarse en la memoria caché  
¿en qué conjunto se almacenaría? (0,25 puntos)
- e) Si el tiempo medio de acceso al sistema de memoria es de 8 ns ¿cuál es tasa de acierto necesaria para lograr este tiempo? (0,25 puntos)

# Solución

---

- a) La memoria tiene un tamaño de línea de 64 bytes =  $2^6$  bytes.  
Por tanto, el número de bloques de memoria principal será

$$\begin{aligned} \text{nbloques} &= \text{tamaño memoria} / \text{tamaño de línea} = \\ & 256 \text{ MB} / 64 = \\ & 256 \times 2^{20} / 64 = 256 * 2^{14} \text{ bloques} \end{aligned}$$

- b) El número de líneas de memoria caché es

$$\begin{aligned} \text{nlineas} &= \text{tamaño memoria} / \text{tamaño de línea} = \\ & 64 \text{ KB} / 64 \text{ bytes} = \\ & 2^{16} / 2^6 = 2^{10} = 1024 \text{ líneas} \end{aligned}$$

$$\text{Conjuntos} = \text{N}^\circ \text{ líneas} / \text{N}^\circ \text{ vías} = 1024 / 4 = 256 \text{ conjuntos.}$$

# Solución

---

- c) La dirección de una caché asociativa por conjuntos se divide en tres partes: etiqueta, conjunto y byte dentro de la línea.
- ▶ Byte: el tamaño de la línea es 64 bytes =  $2^6$  bytes.  
Se necesitan, por tanto 6 bits para identificar el byte dentro de la línea.
  - ▶ Conjunto: Hay 256 conjuntos =  $2^8$ ,  
por lo que se necesitan 8 bits para identificar un conjunto
  - ▶ Etiqueta: para la etiqueta se emplean el resto de los bits de la dirección =  
 $32 - 6 - 8 = 18$

La dirección quedaría:



# Solución

---

d) Utilizamos el formato de la dirección del apartado anterior:



El byte asociado a esta dirección se encontraría en el **conjunto 3**


e) El cálculo del tiempo medio de acceso a memoria se hace con la siguiente fórmula:

▶  $T_{\text{medio}} = h * t_c + (1-h) * t_{\text{fallo}}$

▶  $8 = h * 5 + (1-h) * 100$

Despejando h, se tiene  $h = 92/95 = 0,967$  (tanto por uno)

Es decir, **una tasa de acierto del 96,7 %**



# Tema 5 (II)

## Jerarquía de Memoria



Grupo ARCOS

Estructura de Computadores  
Grado en Ingeniería Informática  
Universidad Carlos III de Madrid