

Grupo ARCOS

**uc3m** | Universidad **Carlos III** de Madrid

# Tema 5: Jerarquía de Memoria (II)

## **Estructura de Computadores**

Grado en Ingeniería Informática  
Grado en Matemática aplicada y Computación  
Doble Grado en Ingeniería Informática y Administración de Empresas

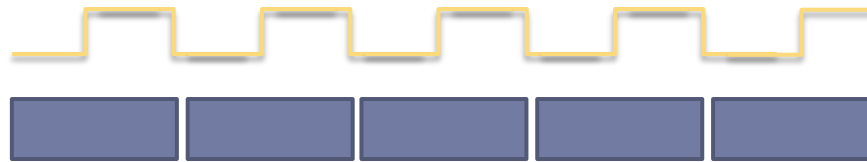


# Contenidos

1. Tipos de memoria
2. Jerarquía de memoria
3. Memoria principal
4. Memoria caché
  1. Introducción
  2. Estructura de la memoria caché
  3. Diseño y organización de la memoria caché
5. Memoria virtual

# Característica de la memoria principal

- ▶ Se premia el acceso a posiciones consecutivas de memoria
- ▶ Ejemplo 1: acceder a 5 posiciones de memoria **individuales no consecutivas**



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



# Característica de los accesos a memoria

- ▶ “Principio de proximidad o localidad de referencias”:

Durante la ejecución de un programa,  
las referencias (direcciones) a memoria tienden a estar  
agrupadas por:

- ▶ **proximidad espacial**
  - ▶ Secuencia de instrucciones
  - ▶ Acceso secuencial a arrays
- ▶ **proximidad temporal**
  - ▶ Bucles

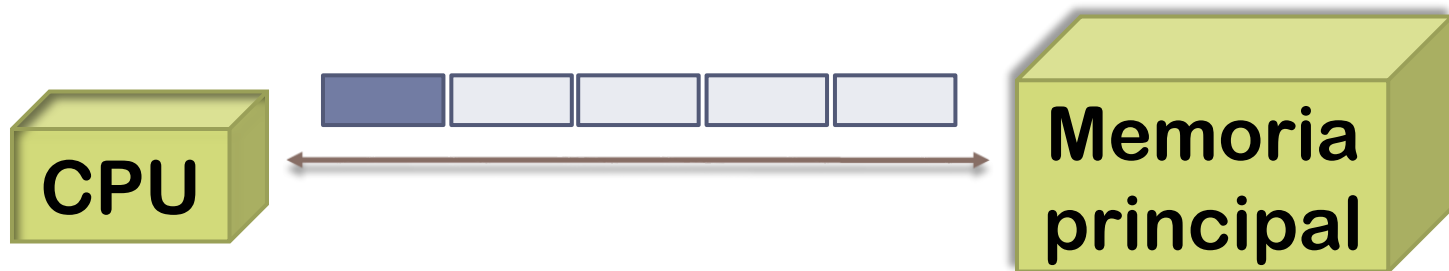
```
.data
vector: .space 4*1024

.text
main:  li  t0 0
      la  t1 vector
      li  t3 1024
      li  t4 4
b2:    bge t0 t3 fb2
      mul t2 t0 t4
      add t2 t1 t2
      sw  t0 0(t2)
      addi t0 t0 1
      j   b2
fb2:   jr  ra
```

# Objetivo de la memoria caché:

aprovechar los accesos contiguos

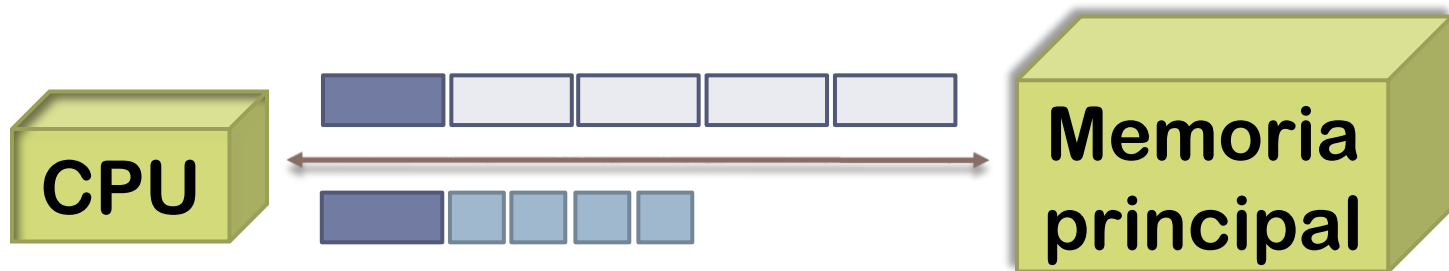
- ▶ Si cuando se accede a una posición de memoria solo se transfieren los datos de esa posición, no se aprovecha los posibles accesos a datos contiguos.



# Objetivo de la memoria caché:

aprovechar los accesos contiguos

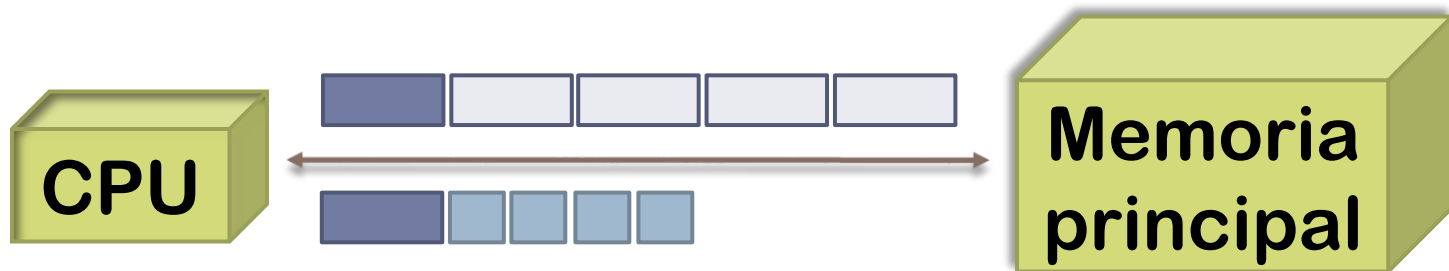
- ▶ Si cuando se accede a una posición de memoria se transfiere esos datos y los contiguos, sí se aprovecha el acceso a datos contiguos



# Objetivo de la memoria caché:

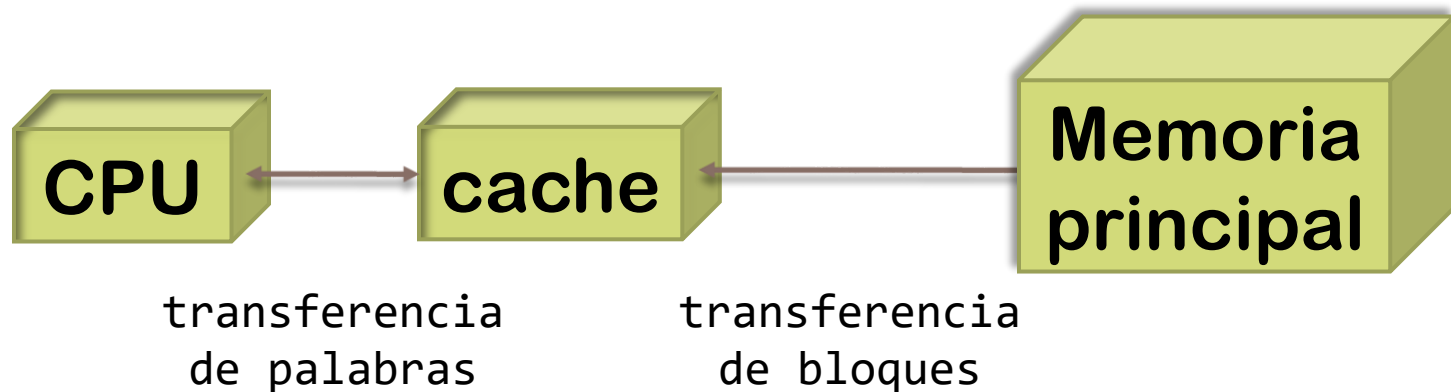
aprovechar los accesos contiguos

- ▶ Si cuando se accede a una posición de memoria se transfiere esos datos y los contiguos, sí se aprovecha el acceso a datos contiguos
  - ▶ Transfiero de la memoria principal un bloque de palabras
  - ▶ ¿Dónde se almacenan las palabras del bloque?



# Memoria Cache

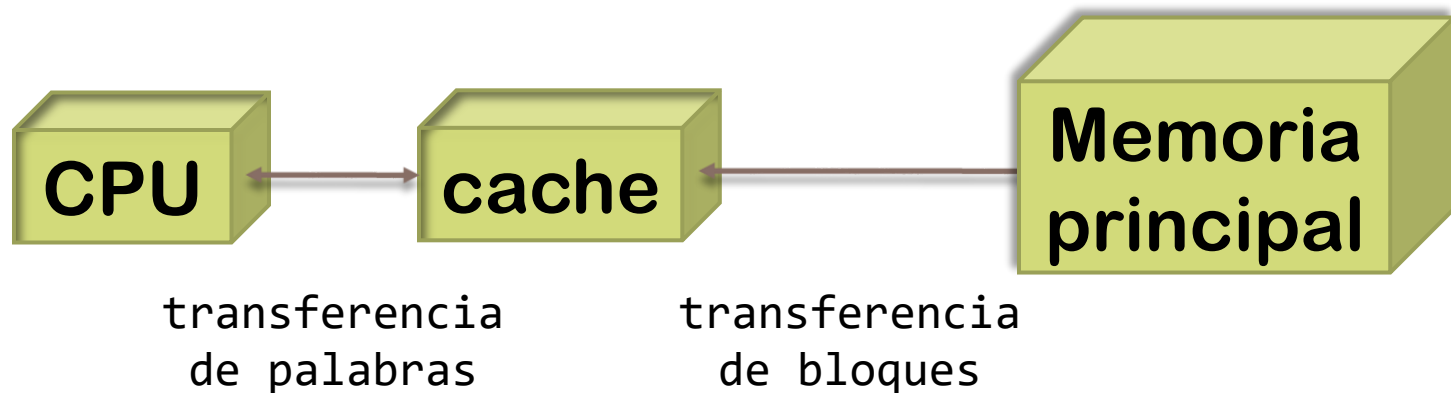
- ▶ Cantidad pequeña de memoria rápida SRAM
  - ▶ Más rápida y cara que la memoria principal DRAM
- ▶ Está **entre** la **memoria principal** y el **procesador** (CPU)
  - ▶ Integrada en el mismo procesador normalmente
- ▶ Almacena una **copia** de partes de la memoria principal





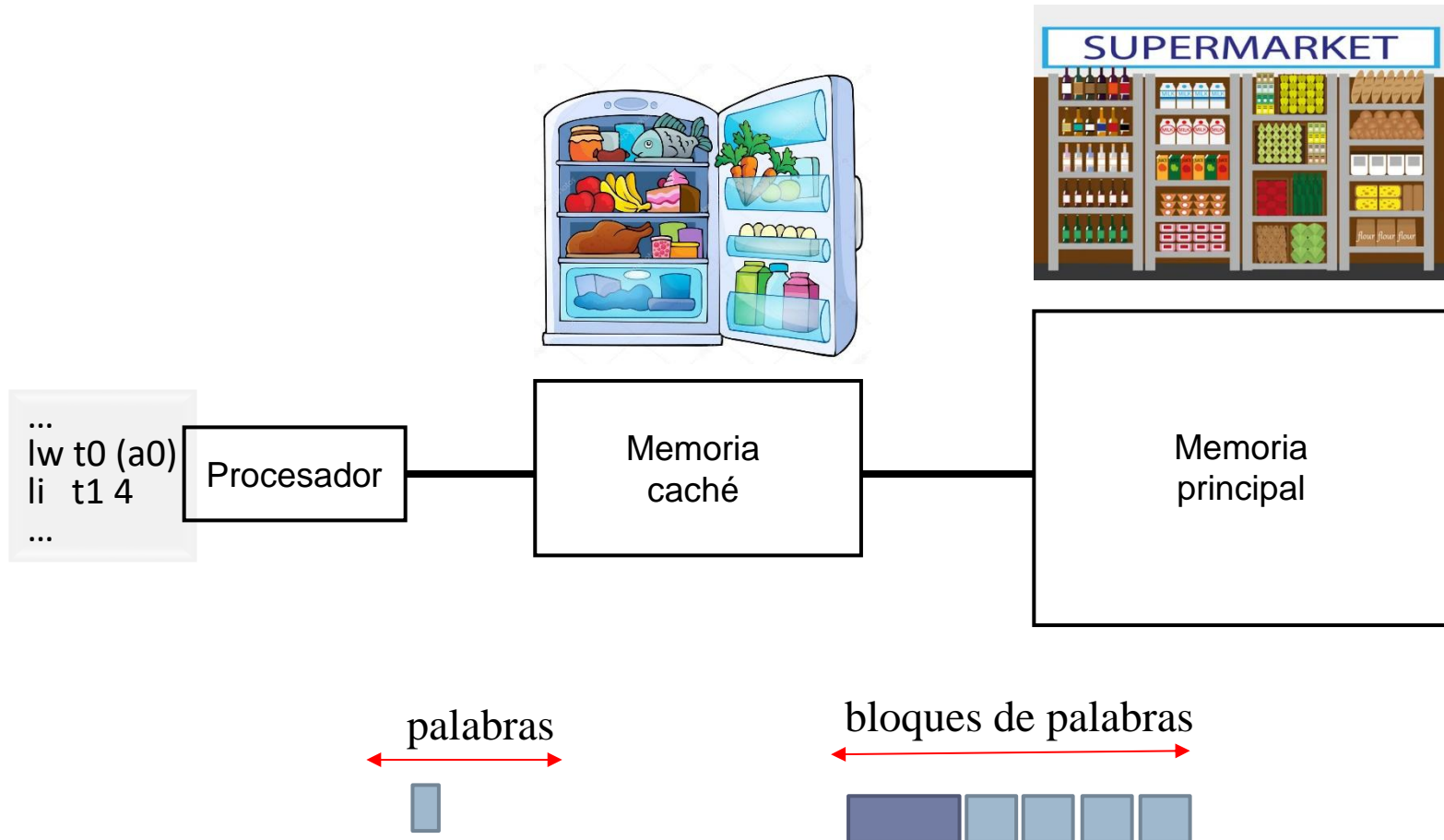
# Ejemplo de tiempos de acceso

- ▶ Memoria principal (DRAM o similar)
  - ▶ Tiempo de acceso: entre 20 y 50 ns.
- ▶ Memoria caché (SRAM o similar)
  - ▶ Tiempo de acceso: entre 1 y 2.5 ns.



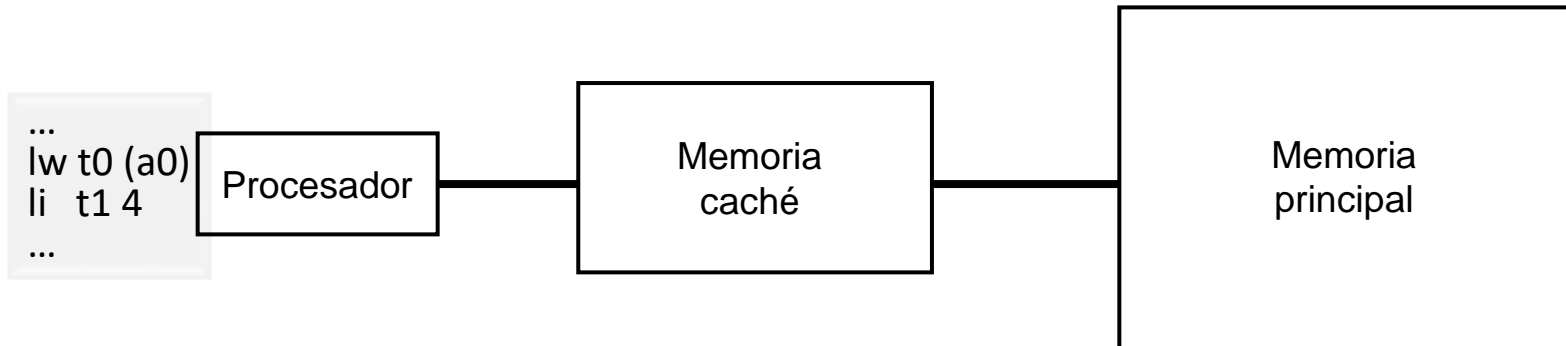
# Memoria caché

metáfora de supermercado y el frigorífico



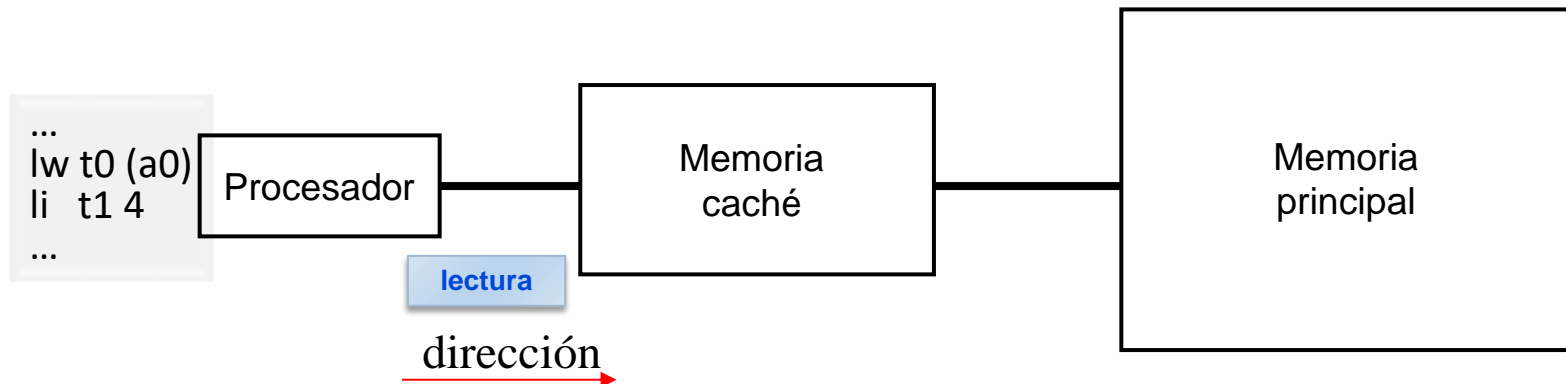
# Funcionamiento de la memoria cache

1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ Si está (**ACIERTO**)
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ Si no está (**FALLO**)
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.



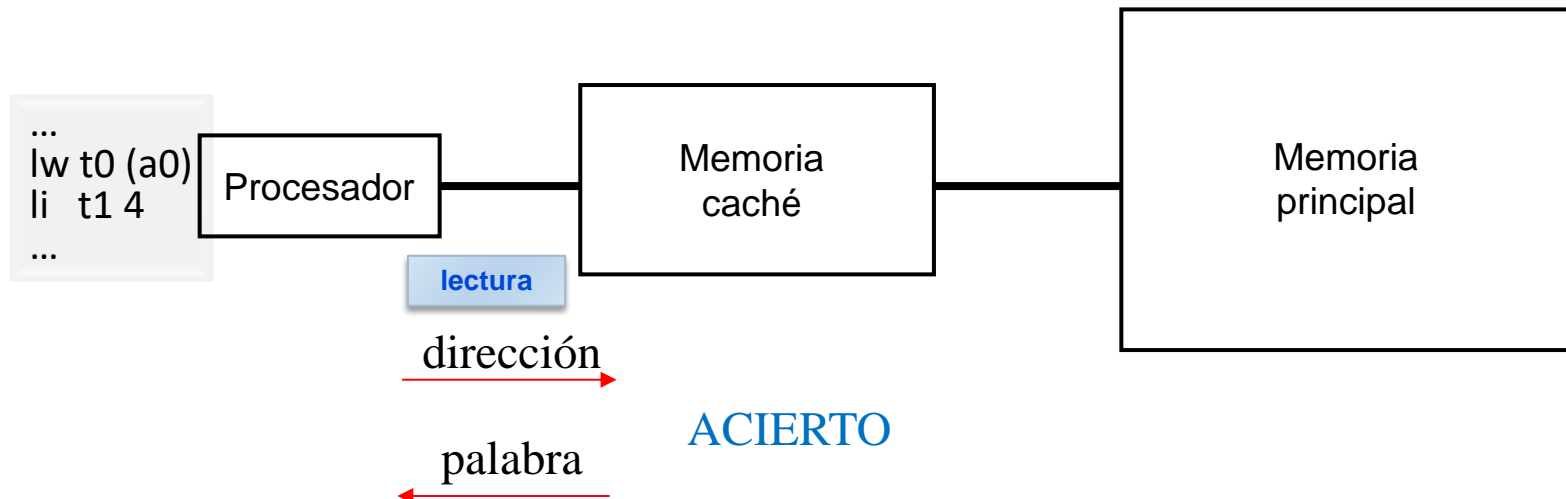
# Funcionamiento de la memoria cache

1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ Si está (ACIERTO)
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ Si no está (FALLO)
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.



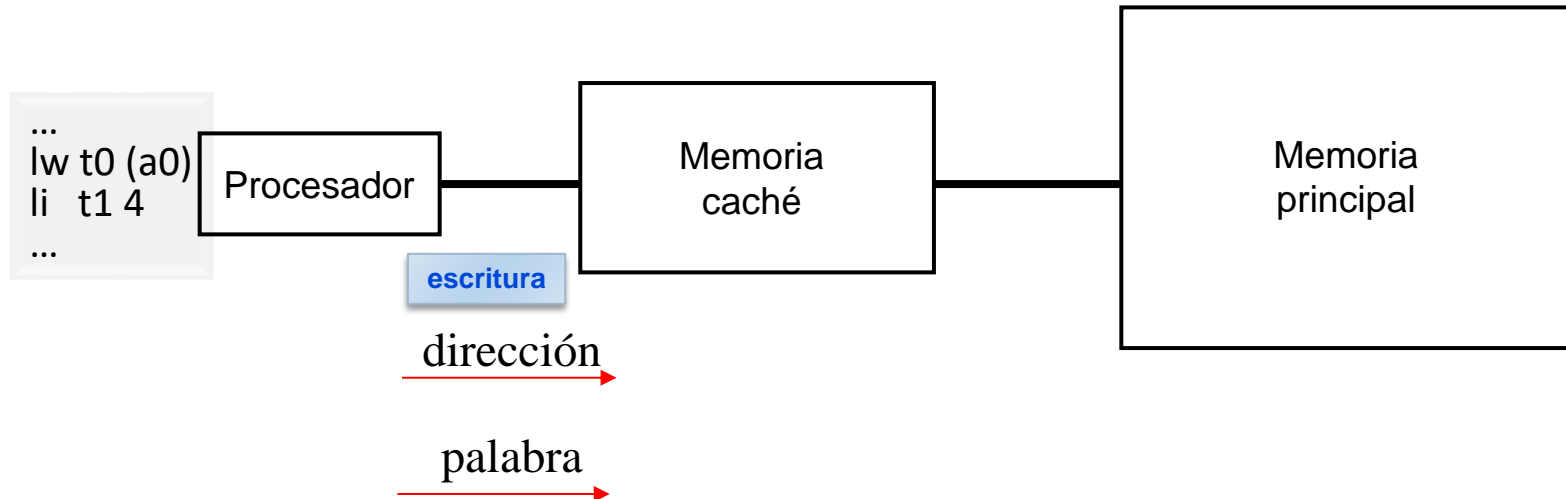
# Funcionamiento de la memoria cache

1. El procesador solicita leer/**escribir** una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ **Si está (ACIERTO)**
    - 3.A.1 Se lee de caché y se sirve al procesador / **se escribe en cache.**
  - ▶ **Si no está (FALLO)**
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / **se escribe en caché la palabra.**



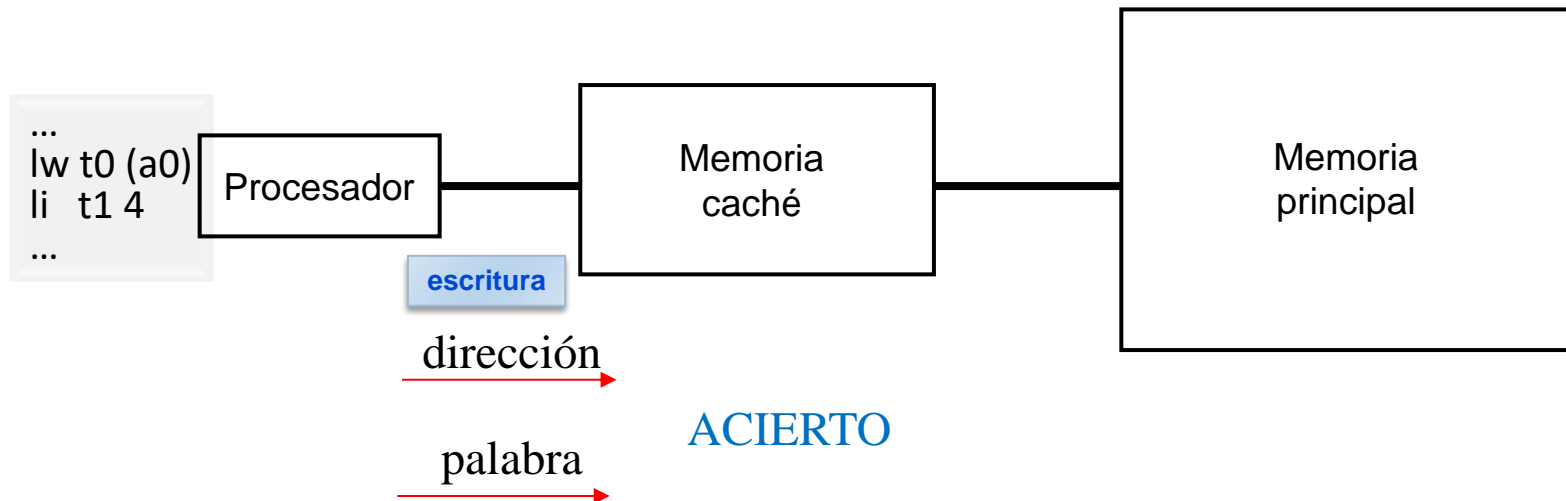
# Funcionamiento de la memoria cache

1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ Si está (ACIERTO)
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ Si no está (FALLO)
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.



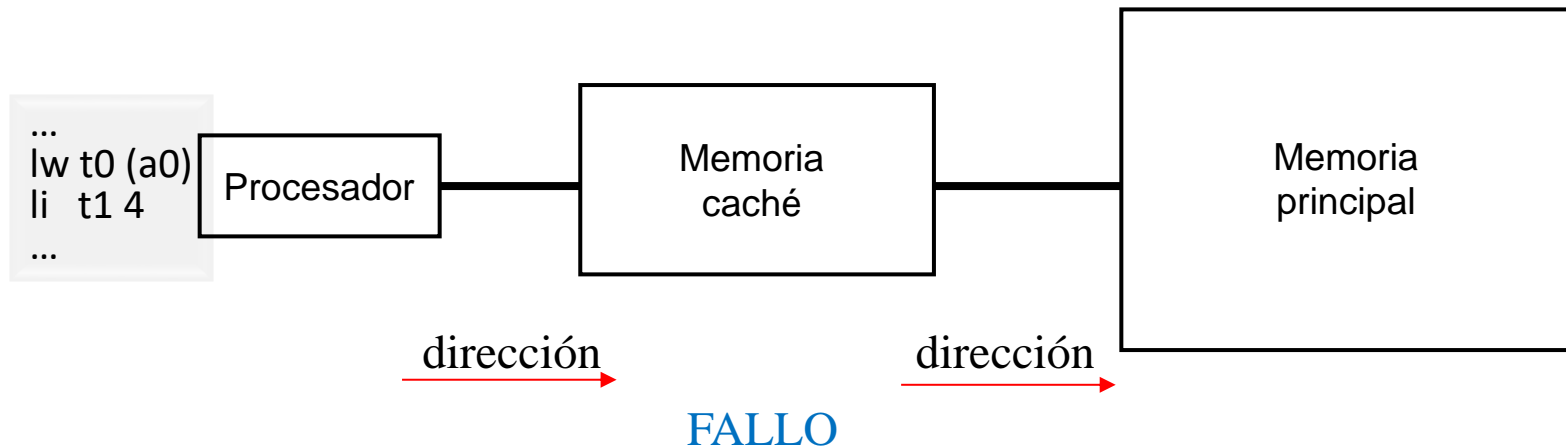
# Funcionamiento de la memoria cache

1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ **Si está (ACIERTO)**
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ **Si no está (FALLO)**
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.



# Funcionamiento de la memoria cache

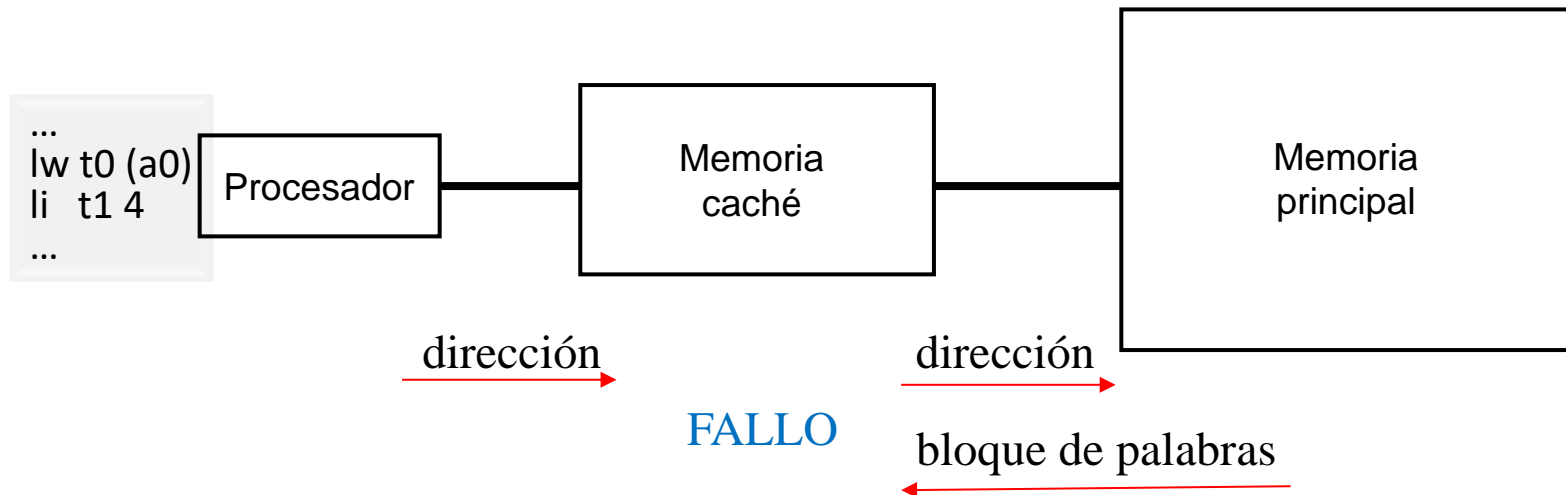
1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ Si está (**ACIERTO**)
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ Si no está (**FALLO**)
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.





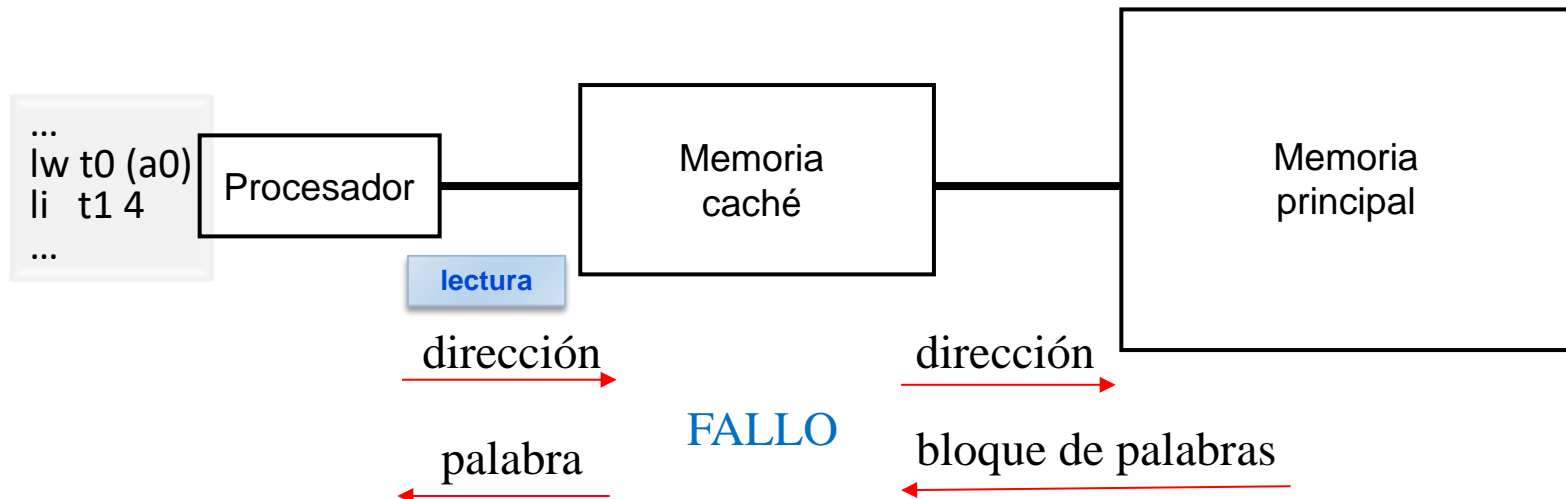
# Funcionamiento de la memoria cache

1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ Si está (**ACIERTO**)
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ Si no está (**FALLO**)
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.



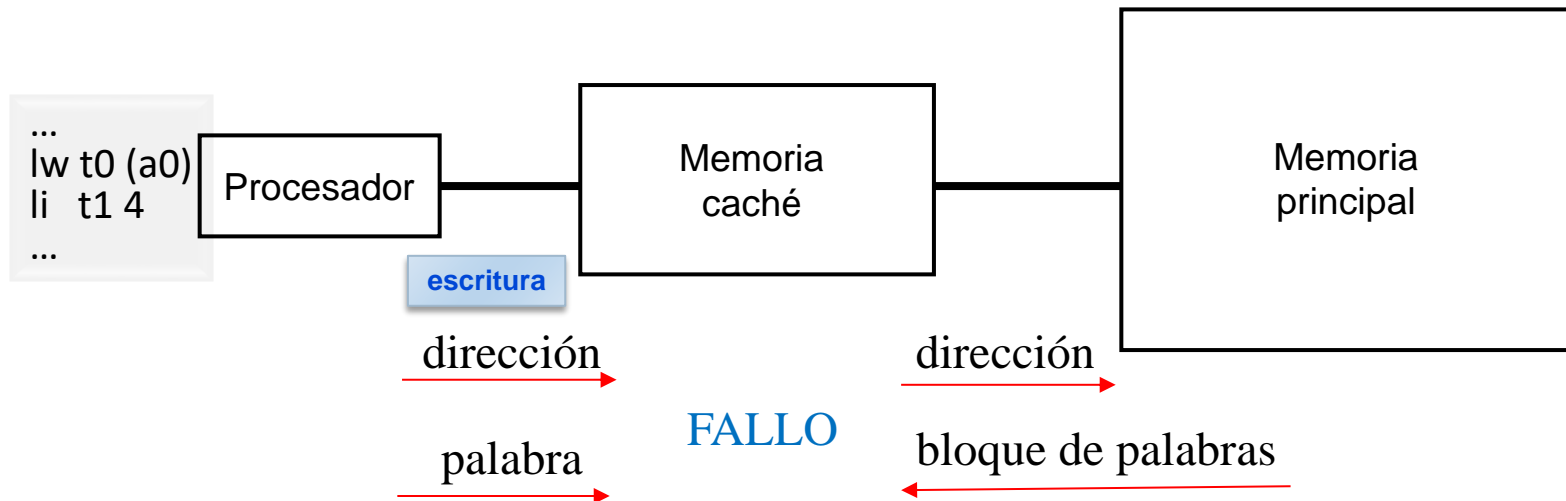
# Funcionamiento de la memoria cache

1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ Si está (**ACIERTO**)
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ Si no está (**FALLO**)
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.



# Funcionamiento de la memoria cache

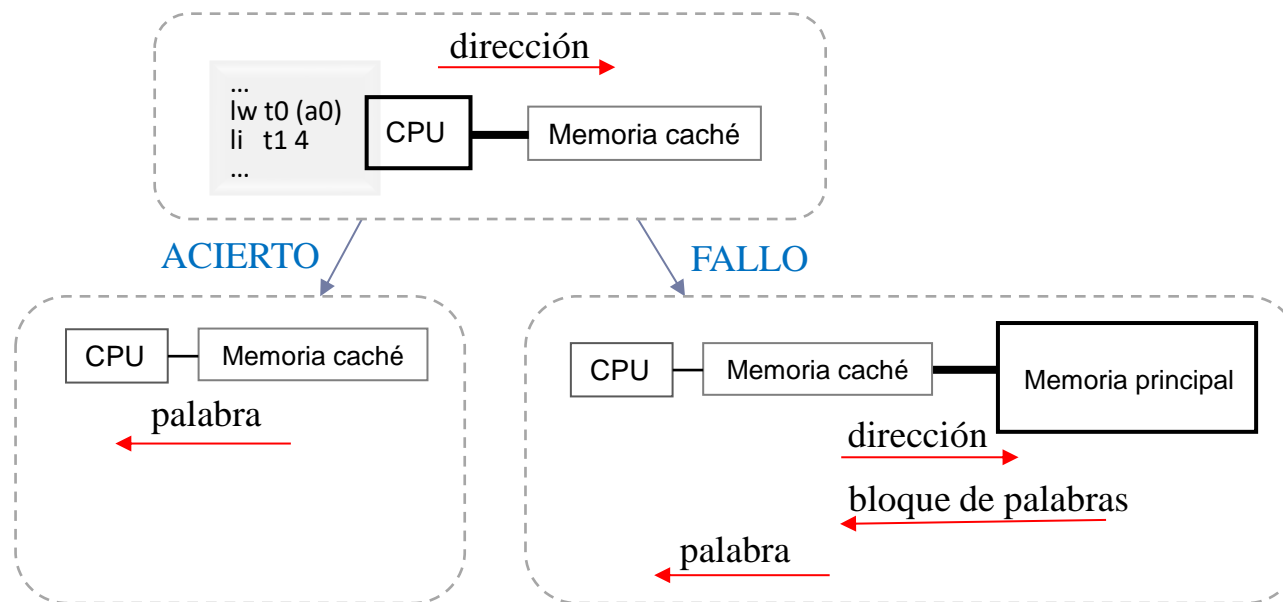
1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ Si está (**ACIERTO**)
    - 3.A.1 Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ Si no está (**FALLO**)
    - 3.B.1 La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2 Después, se lee los datos desde caché / se escribe en caché la palabra.



# Funcionamiento de la memoria caché

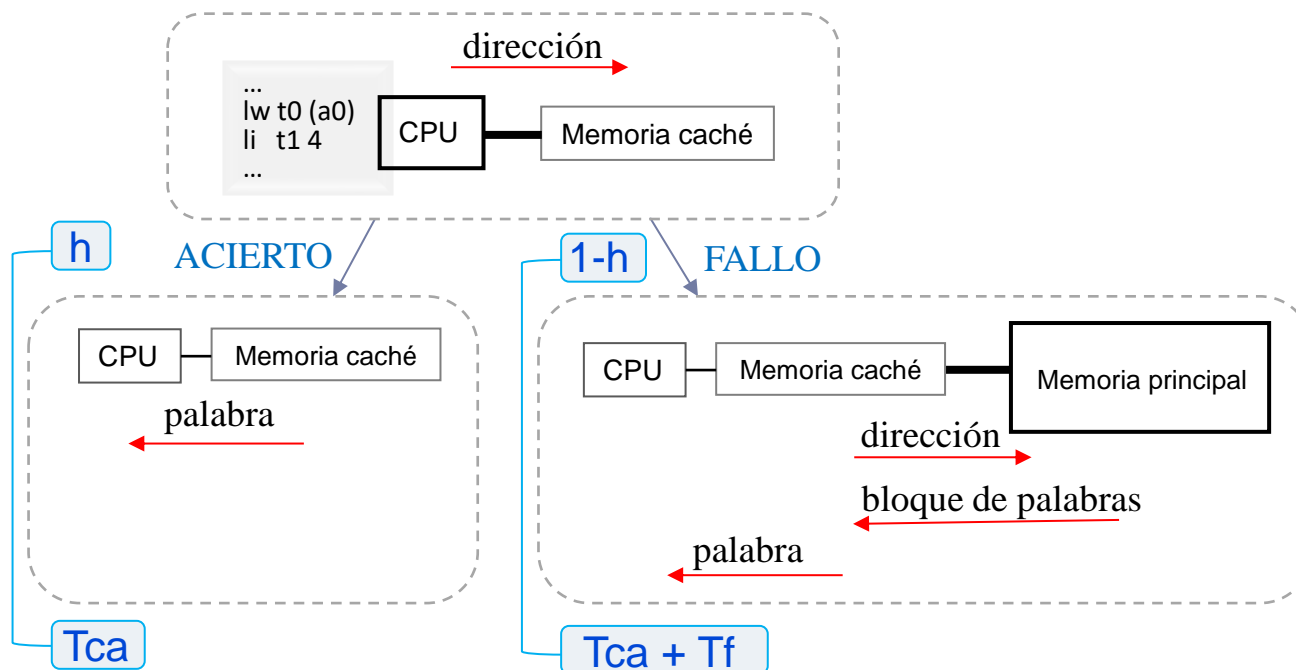
## Resumen

1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ **Si está (ACIERTO)**
    - 3.A.1** Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ **Si no está (FALLO)**
    - 3.B.1** La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2** Después, se lee los datos desde caché / se escribe en caché la palabra.



# Funcionamiento de la memoria caché

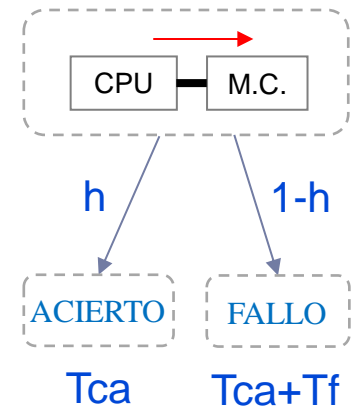
1. El procesador solicita leer/escribir una posición de memoria.
2. La cache comprueba si ya están los datos de esta posición:
  - ▶ **Si está (ACIERTO)**
    - 3.A.1** Se lee de caché y se sirve al procesador / se escribe en cache.
  - ▶ **Si no está (FALLO)**
    - 3.B.1** La cache transfiere de memoria principal el bloque asociado a la posición.
    - 3.B.2** Después, se lee los datos desde caché / se escribe en caché la palabra.



# Tiempo medio de acceso a caché

- ▶ Tiempo medio de acceso a un sistema de memoria con dos niveles:

$$\begin{aligned} T_m &= h \cdot T_a + (1-h) \cdot (T_a + T_f) \\ &= T_a + (1-h) \cdot T_f \end{aligned}$$



- ▶ **Ta**: tiempo de acceso a la caché
- ▶ **Tf**: tiempo en tratar el fallo
  - ▶ Incluye el tiempo en remplazar un bloque, traer un bloque de memoria principal a caché, etc.
- ▶ **h**: tasa de aciertos

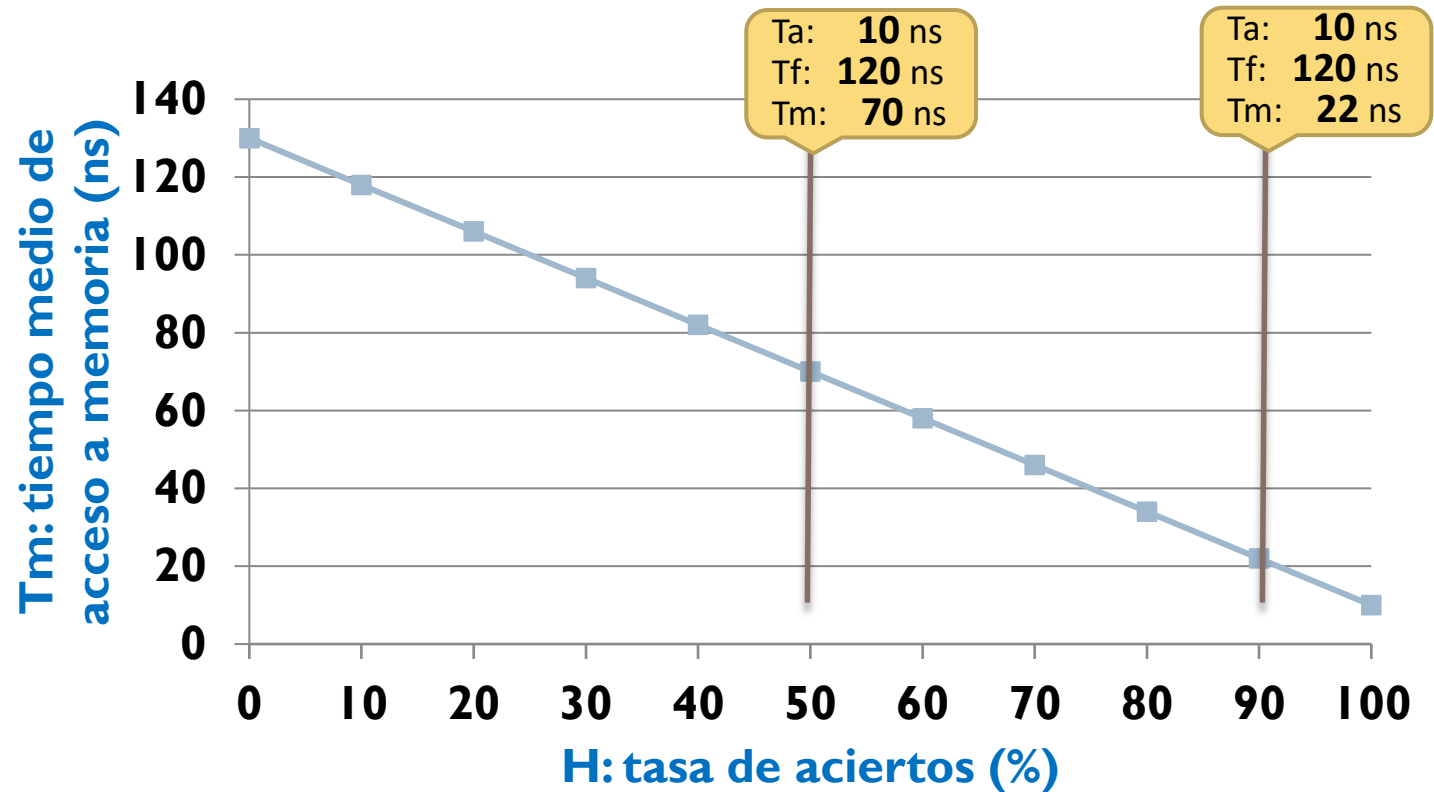
# Ejemplo

$$\begin{aligned}T_m &= h \cdot T_a + (1-h) \cdot (T_a + T_f) \\ &= T_a + (1-h) \cdot T_f\end{aligned}$$

1.  $T_a$ : Tiempo de acceso a caché = **10 ns**
2.  $T_f$ : Tiempo de acceso a memoria principal = **120 ns**
3.  $h$ : tasa de aciertos  $\rightarrow X = 0.1, 0.2, \dots, 0.9, 1.0$   
**10%, 20%, ..., 90%, 100%**

# Ejemplo

$$T_m = h \cdot T_a + (1-h) \cdot (T_a + T_f)$$
$$= T_a + (1-h) \cdot T_f$$





# Ejercicio

- ▶ **Computador:**
  - ▶ Tiempo de acceso a caché: 4 ns
  - ▶ Tiempo de acceso a bloque de MP: 120 ns.
- ▶ Si se tiene una tasa de aciertos del 90%. ¿Cuál es el tiempo medio de acceso?
- ▶ Tasas de acierto necesarias para que el tiempo medio de acceso sea menor de 5 ns.

# Ejercicio (solución)

- ▶ **Computador:**

- ▶ Tiempo de acceso a caché: 4 ns
- ▶ Tiempo de acceso a bloque de MP: 120 ns.

- ▶ Si se tiene una tasa de aciertos del 90%. ¿Cuál es el tiempo medio de acceso?

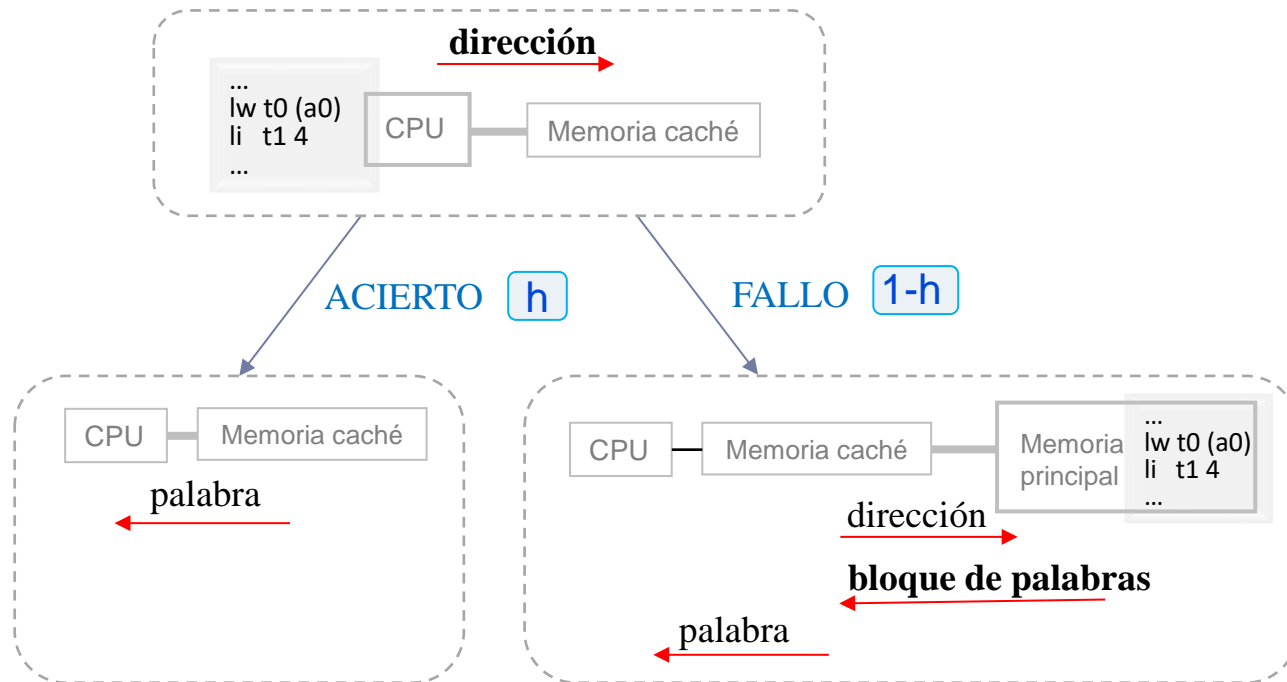
$$T_m = 4 \times 0.9 + (120 + 4) \times 0.1 = 16 \text{ ns}$$

- ▶ Tasas de acierto necesarias para que el tiempo medio de acceso sea menor de 5 ns.

$$\begin{aligned} 5 &= 4 \times h + (120 + 4) \times (1 - h) \\ \Rightarrow h &> 0.9916 \end{aligned}$$

# Tasa de aciertos de un fragmento de código

- ▶ La tasa de aciertos  $h$  depende de:
  - ▶ Disposición en memoria ppal. y caché (bloques afectados).
  - ▶ Traza de accesos (lista de direcciones) que genera al ejecutarse.
  - ▶ El comportamiento de la memoria caché (tiempo de búsqueda, remplazo, etc.)



# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;
```

```
li    t0, 0      # s  
li    t1, 0      # i  
li    t2, 1000  
bucle: bge    t1, t2, fin  
add   t0, t0, t1  
addi  t1, t1, 1  
j     bucle  
fin:  ...
```

- ▶ Ejemplo:
  - ▶ Acceso a M.caché: 2 ns
  - ▶ Acceso a M.P.: 120 ns
  - ▶ Bloque de caché: 4 palabras
  - ▶ Transferencia de un bloque entre memoria principal y caché: 200 ns

# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;
```

```
li    t0, 0      # s  
li    t1, 0      # i  
li    t2, 1000  
bucle: bge    t1, t2, fin  
add   t0, t0, t1  
addi  t1, t1, 1  
j     bucle  
fin:  ...
```

## ► Sin memoria caché:

- Número de accesos a memoria =  $3 + 4 \times 1000 + 1 = 4004$  accesos
- Tiempo de acceso a memoria =  $4004 \times 120 = 480480$  ns = 0,480 ms

# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;
```

```
li    t0, 0      # s  
li    t1, 0      # i  
li    t2, 1000  
bucle: bge    t1, t2, fin  
add   t0, t0, t1  
addi  t1, t1, 1  
j     bucle  
fin:  ...
```

- ▶ **Con** memoria caché (bloque de 4 palabras):
  - ▶ Número de accesos = 4004 accesos
  - ▶ Número de bloques = ?
  - ▶ Número de fallos = ?
  - ▶ Tiempo de acceso = ?

# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;
```

bucle:

```
li    t0, 0      # s  
li    t1, 0      # i  
li    t2, 1000  
bge   t1, t2, fin
```

fin:

```
add   t0, t0, t1  
addi  t1, t1, 1  
j     bucle  
...
```

## ► **Con** memoria caché (bloque de 4 palabras):

- Número de accesos = 4004 accesos
- Número de bloques = 2
- Número de fallos = ?
- Tiempo de acceso = ?

(1/2) estudio de bloques:  
análisis de bloques de datos y  
código afectados

# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;
```

bucle:

```
li    t0, 0      # s  
li    t1, 0      # i  
li    t2, 1000  
bge   t1, t2, fin
```

fin:

```
add   t0, t0, t1  
addi  t1, t1, 1  
j     bucle  
...
```

## ► Con memoria caché (bloque de 4 palabras):

- Número de accesos = 4004 accesos
- Número de bloques = 2
- Número de fallos = ?
- Tiempo de acceso = ?

(2/2) estudio de referencias  
generadas por ejecución: accesos  
a código (fetch) y datos



# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas

procesador

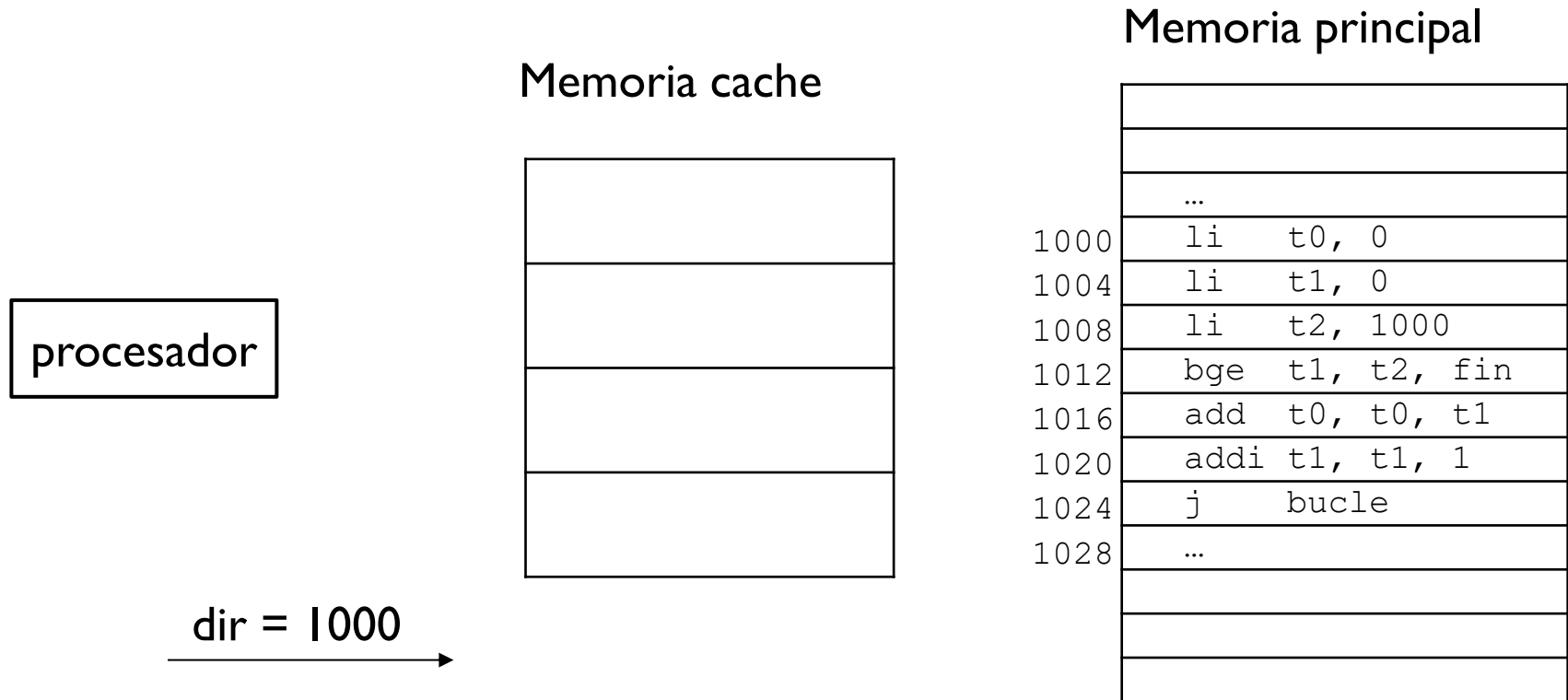
Memoria cache


Memoria principal

	...
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	...

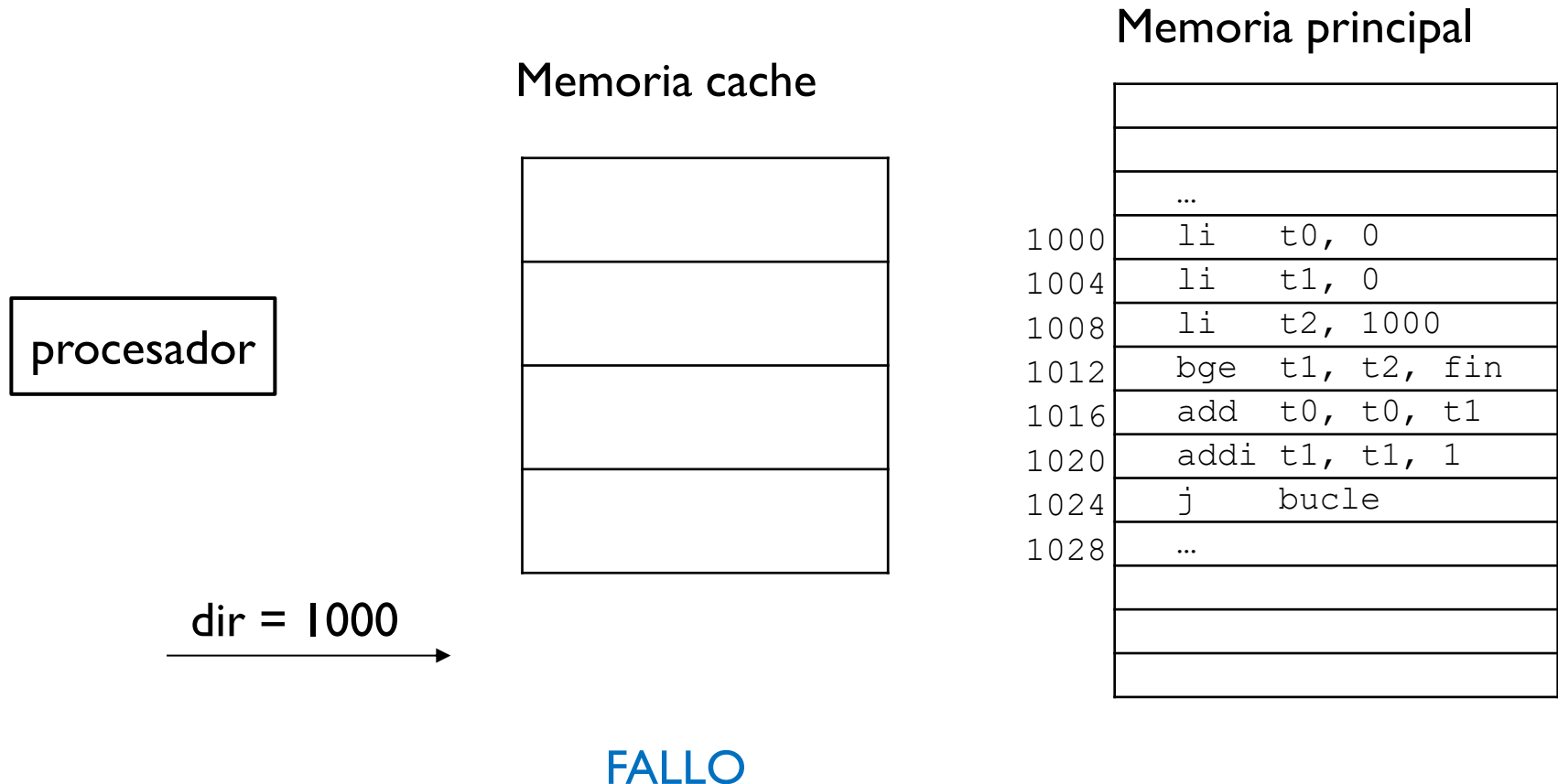
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



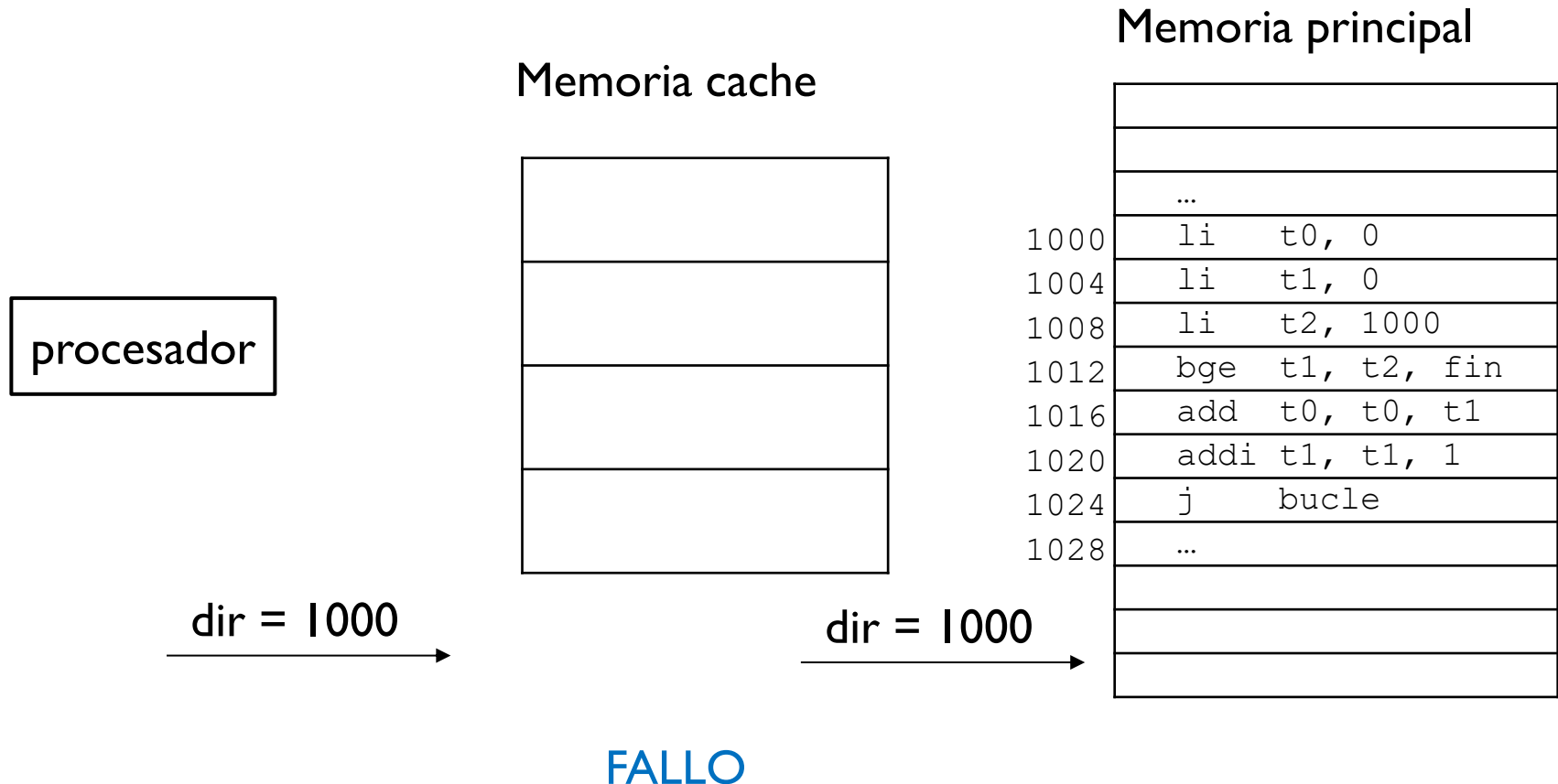
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



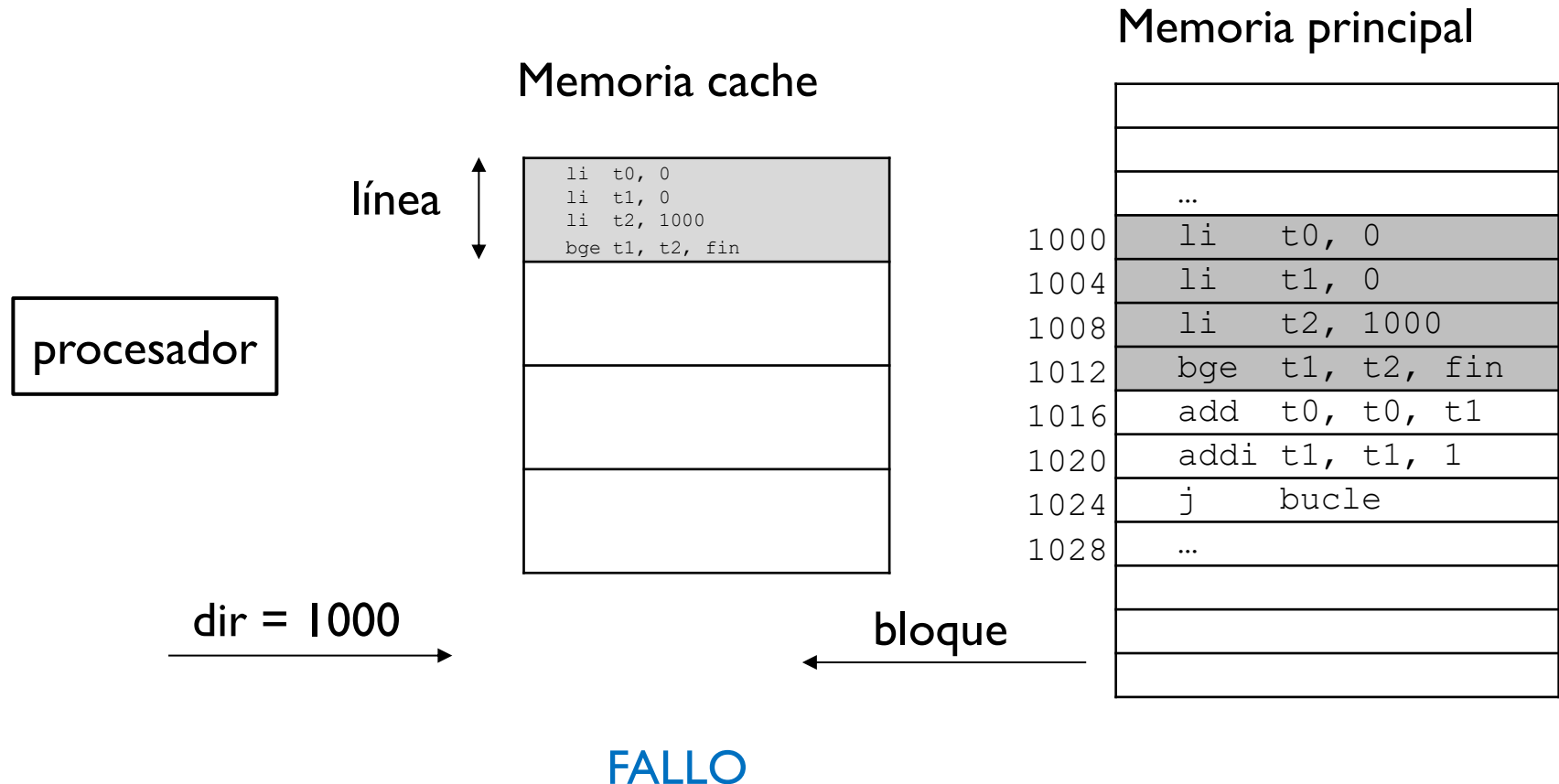
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



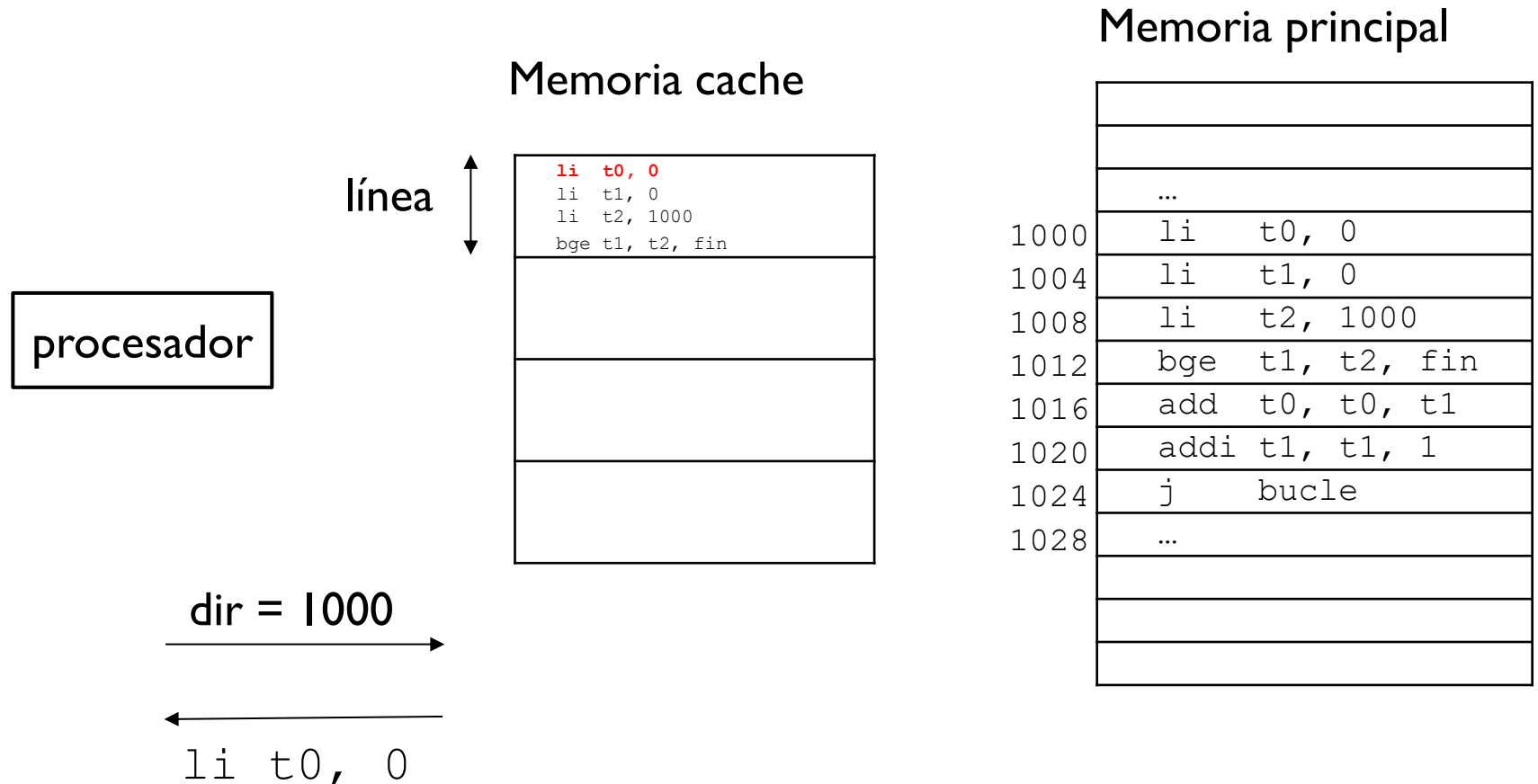
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



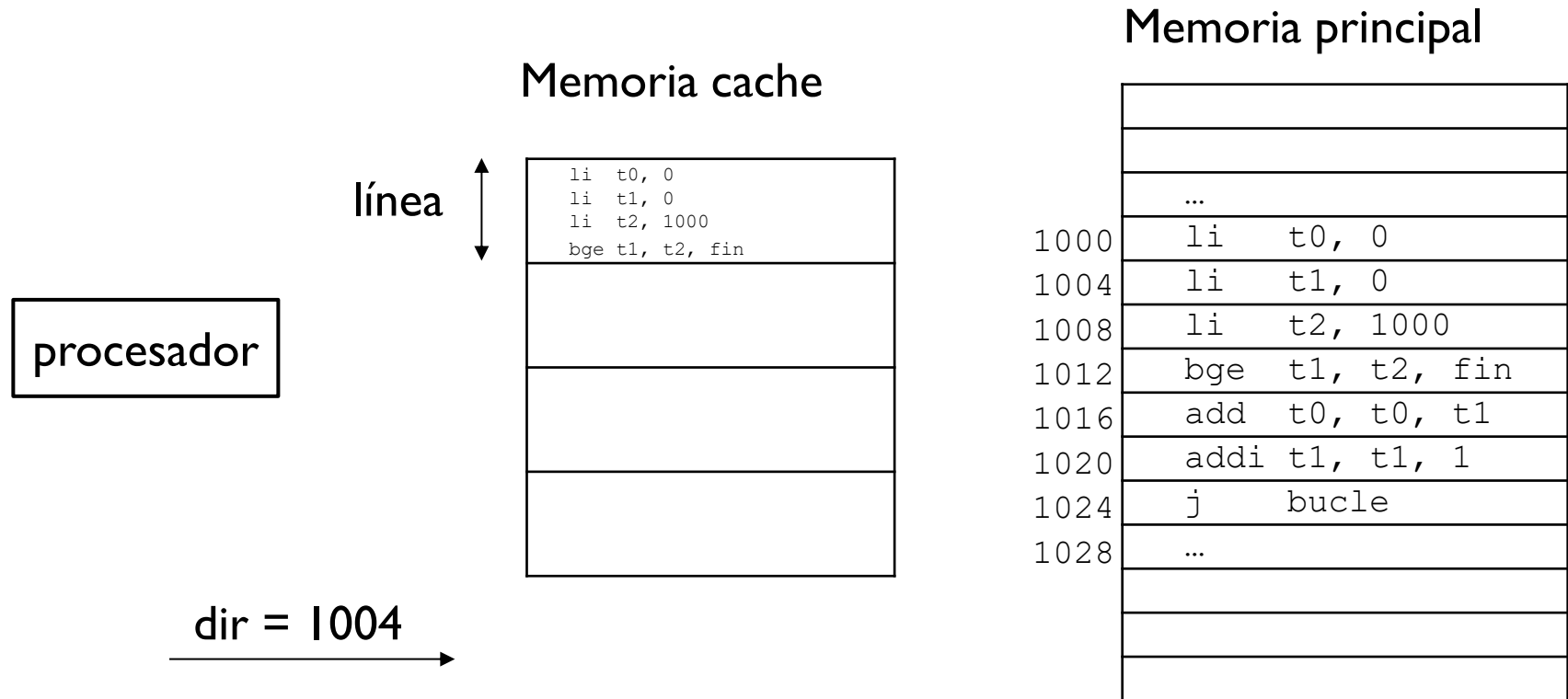
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



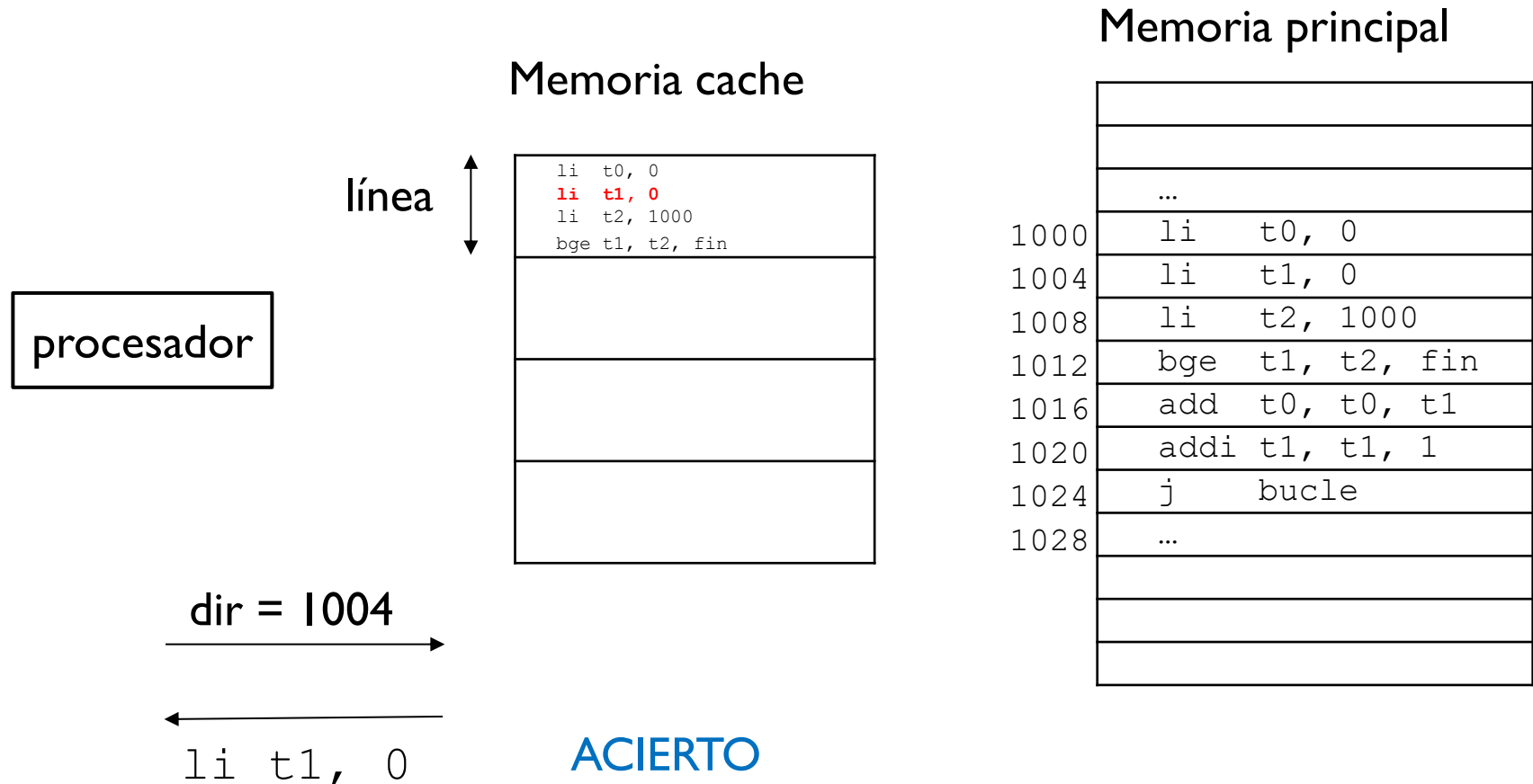
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



# Ejemplo de funcionamiento

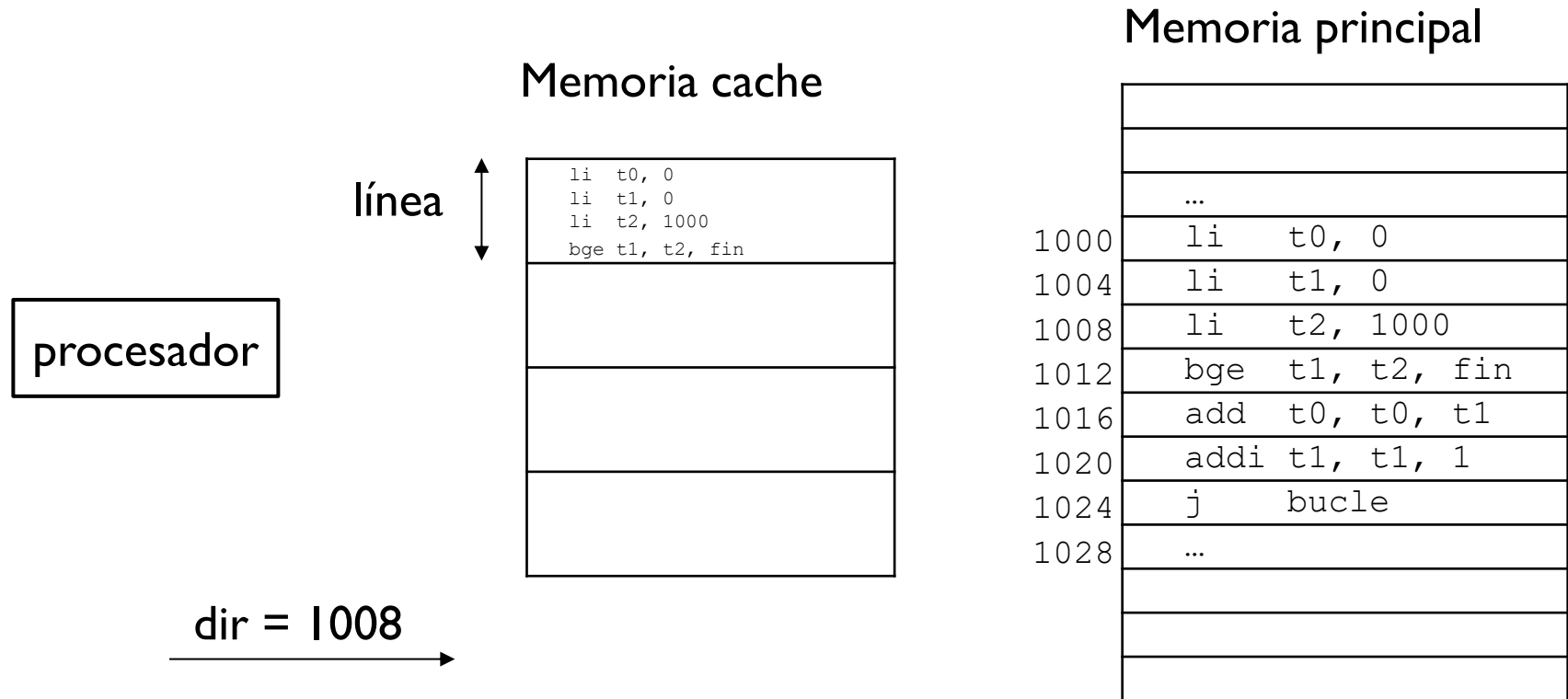
(2/2) estudio de referencias generadas





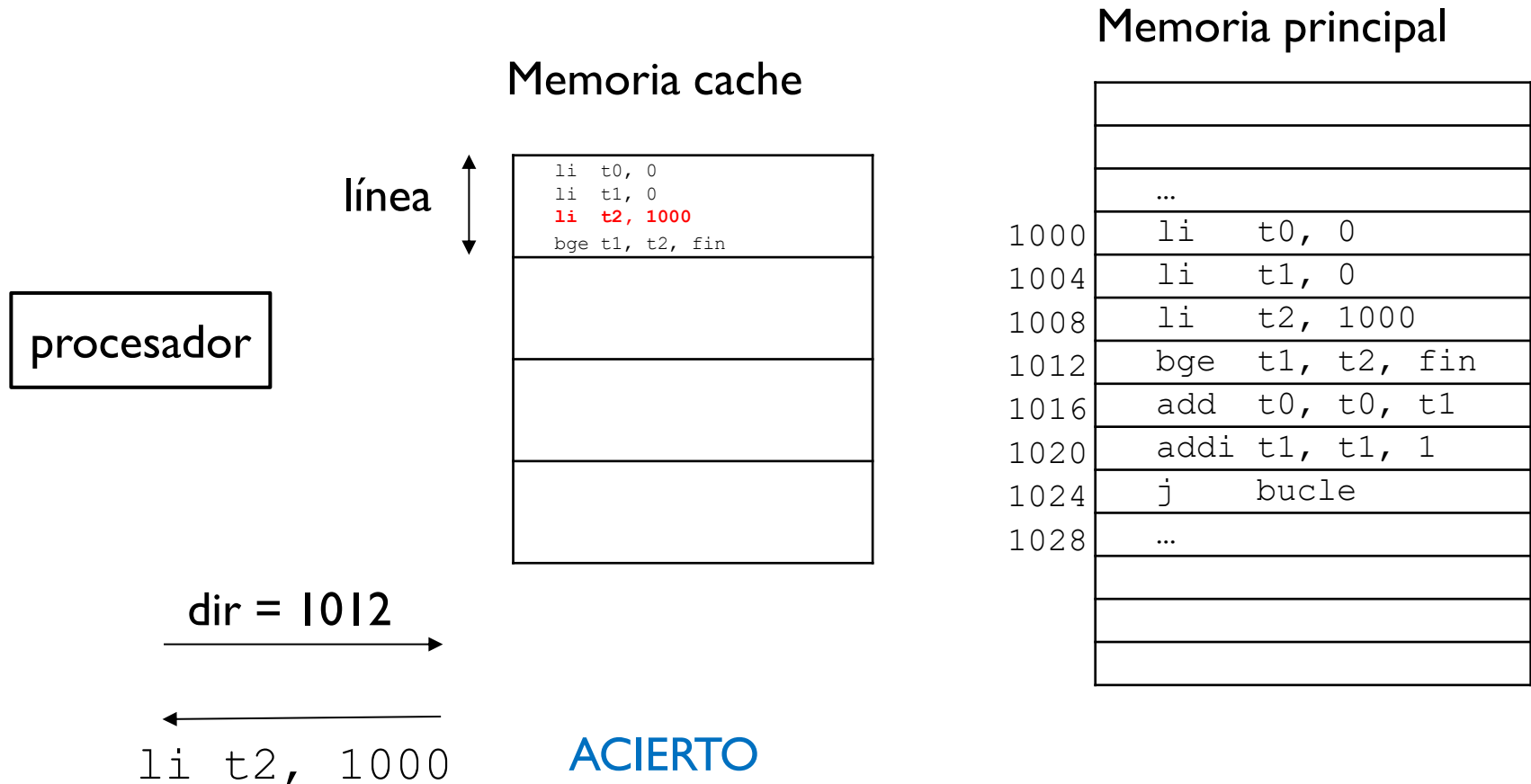
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



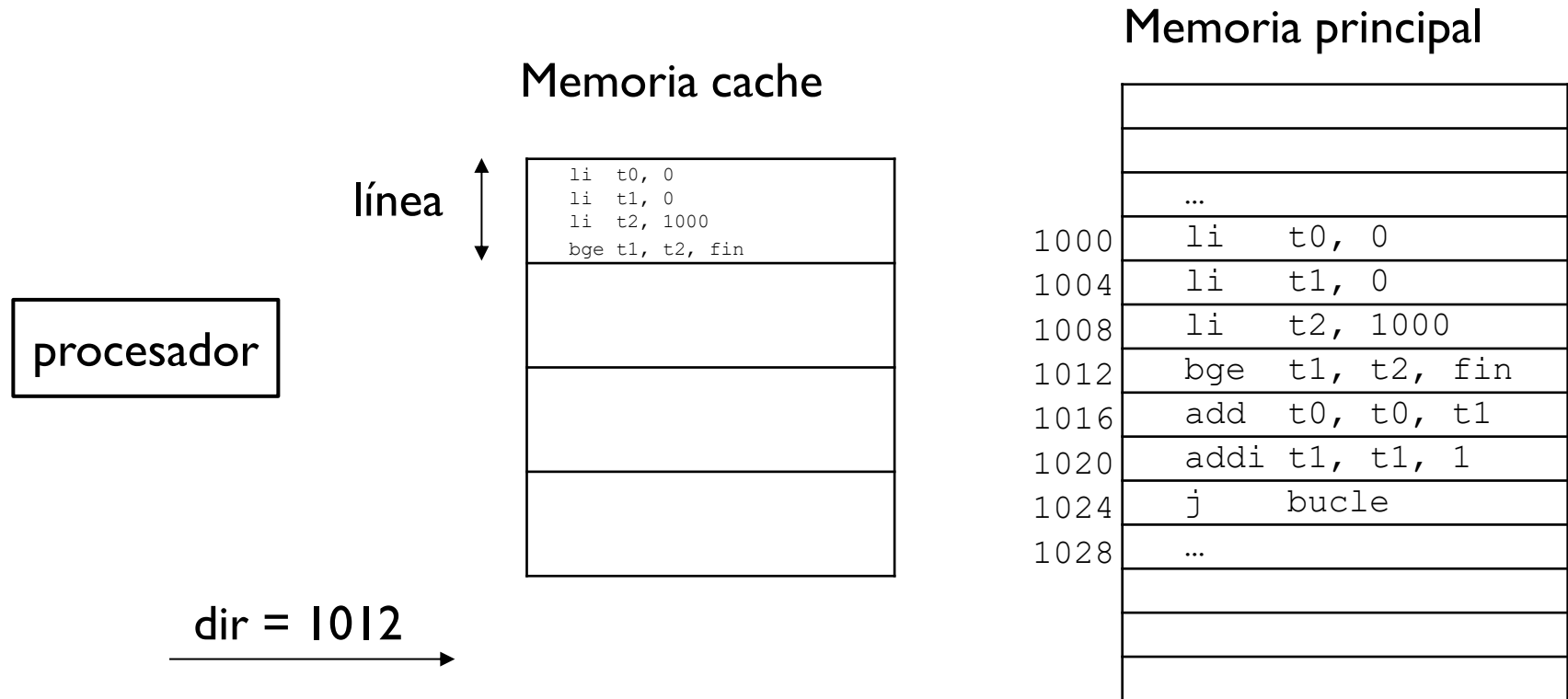
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



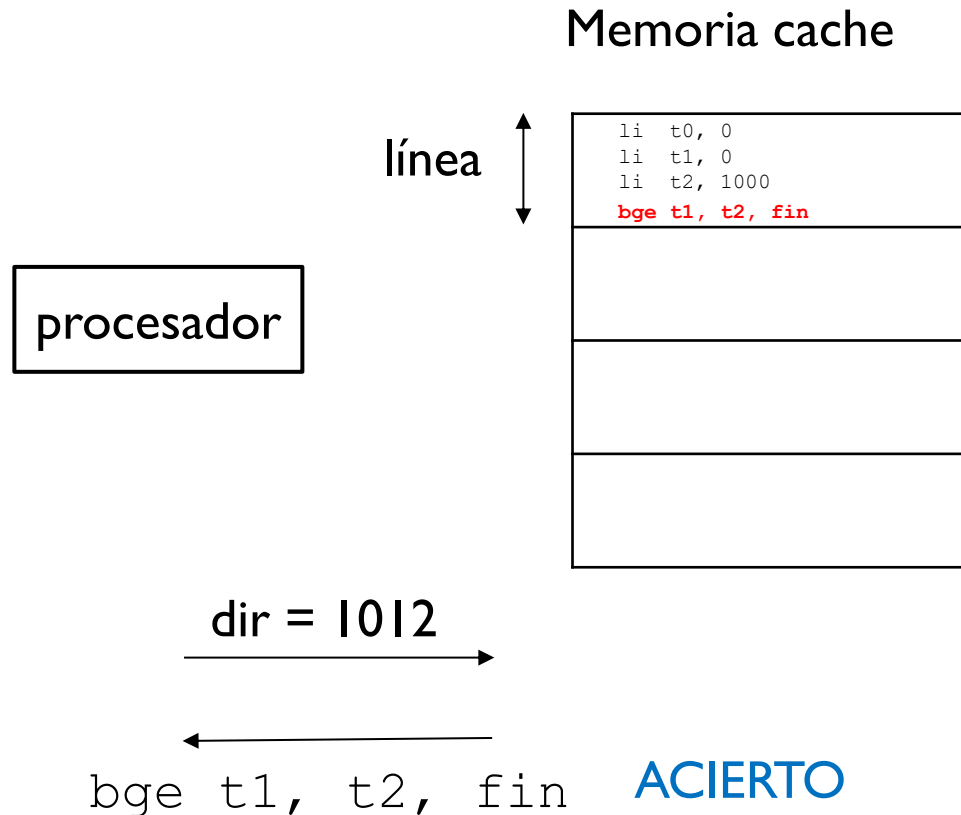
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas

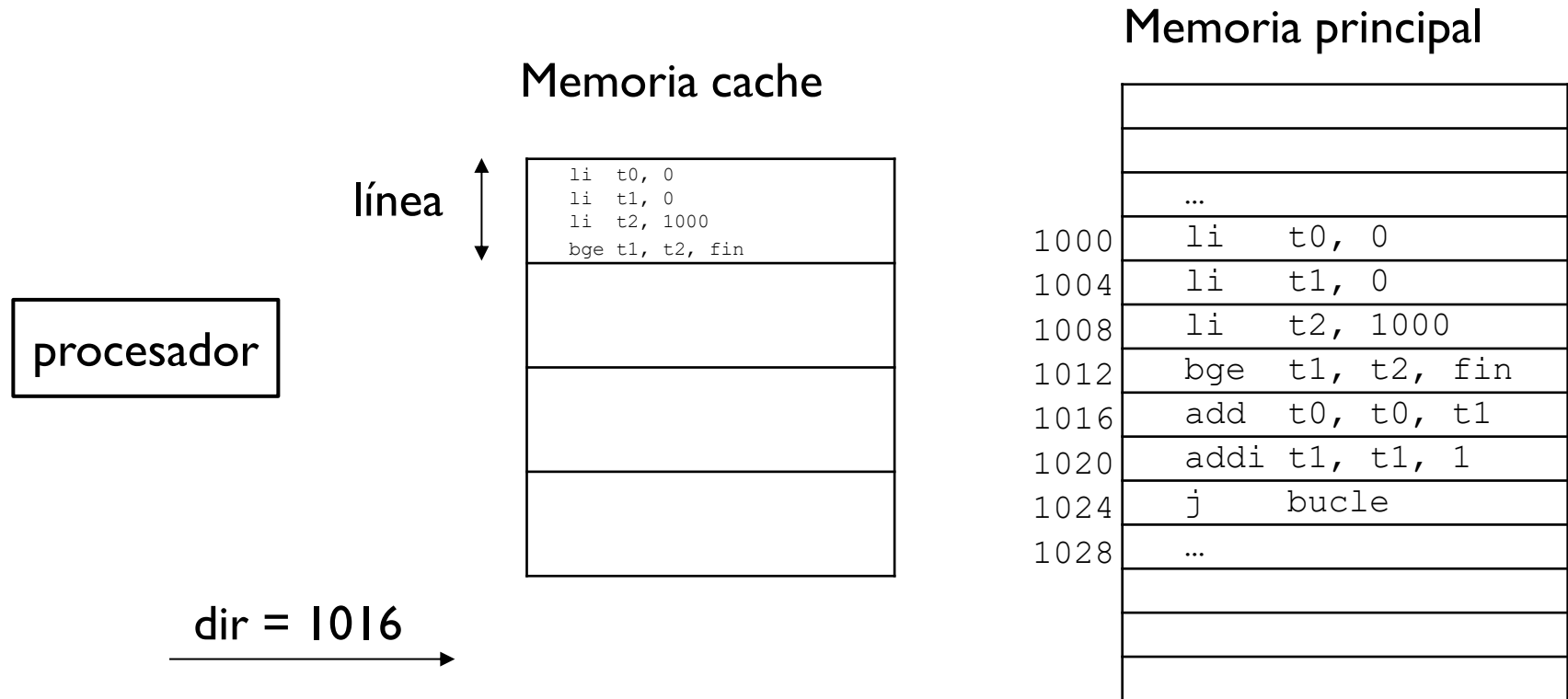


## Memoria principal

	...
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	...

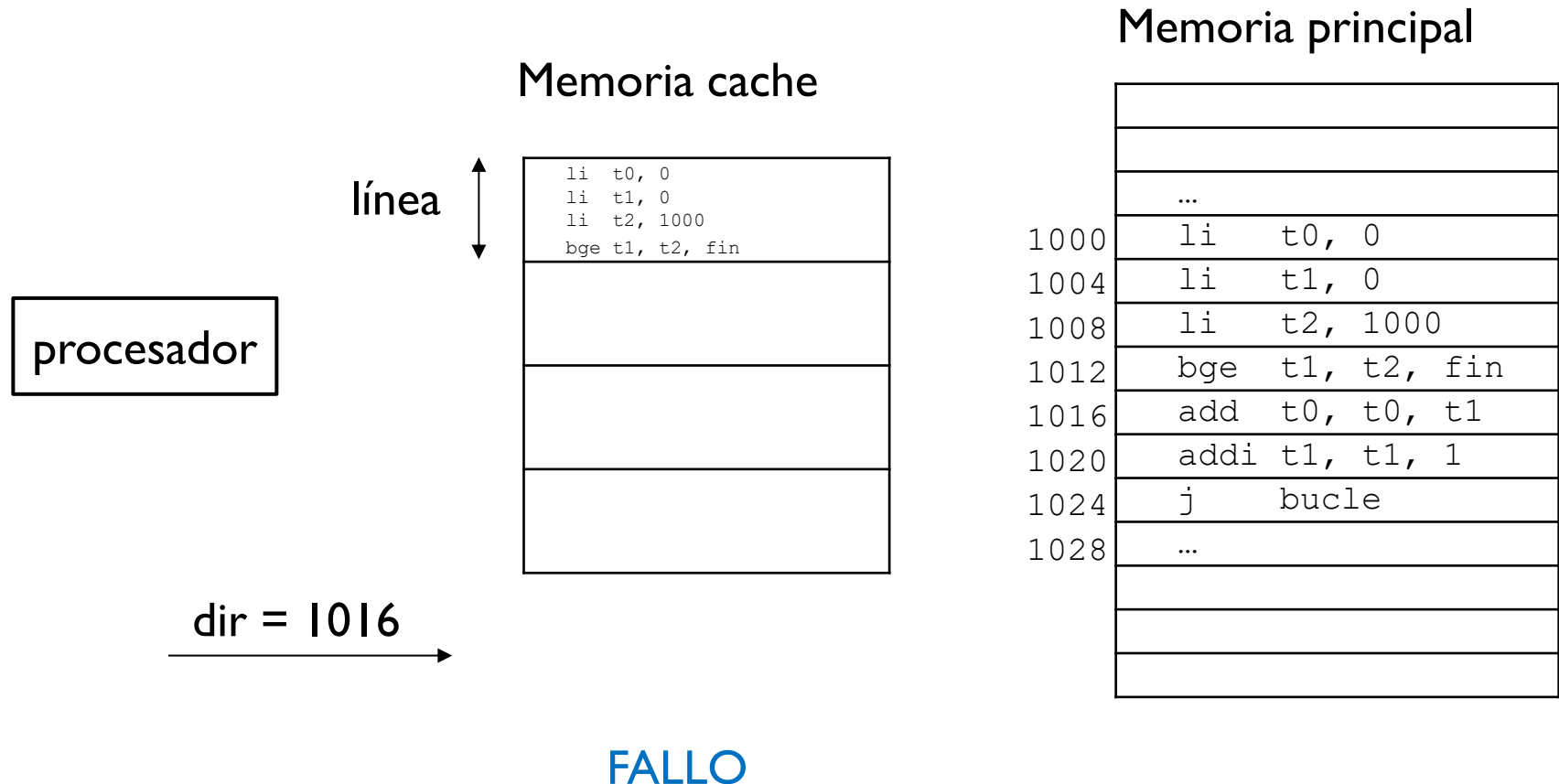
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



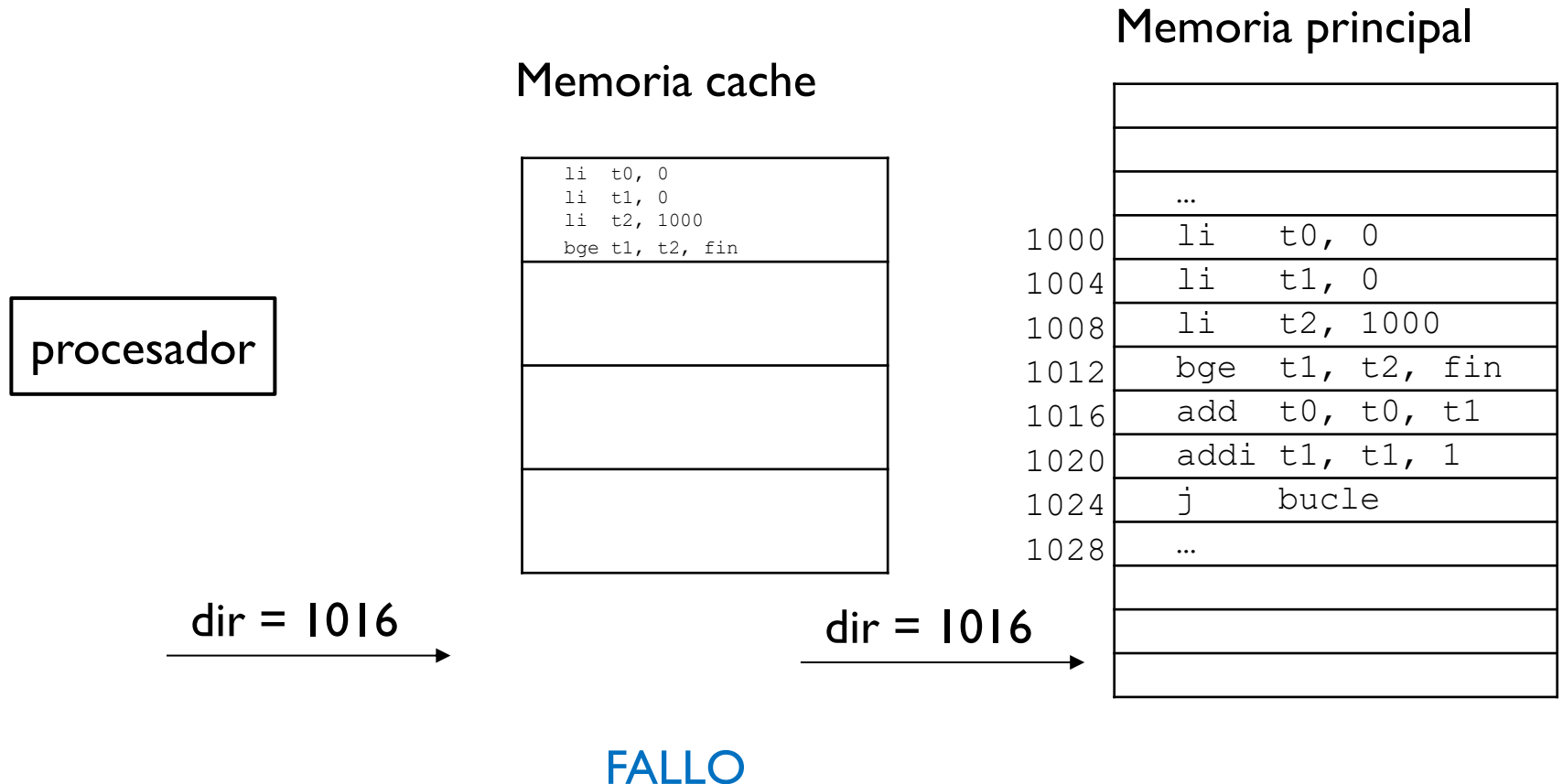
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



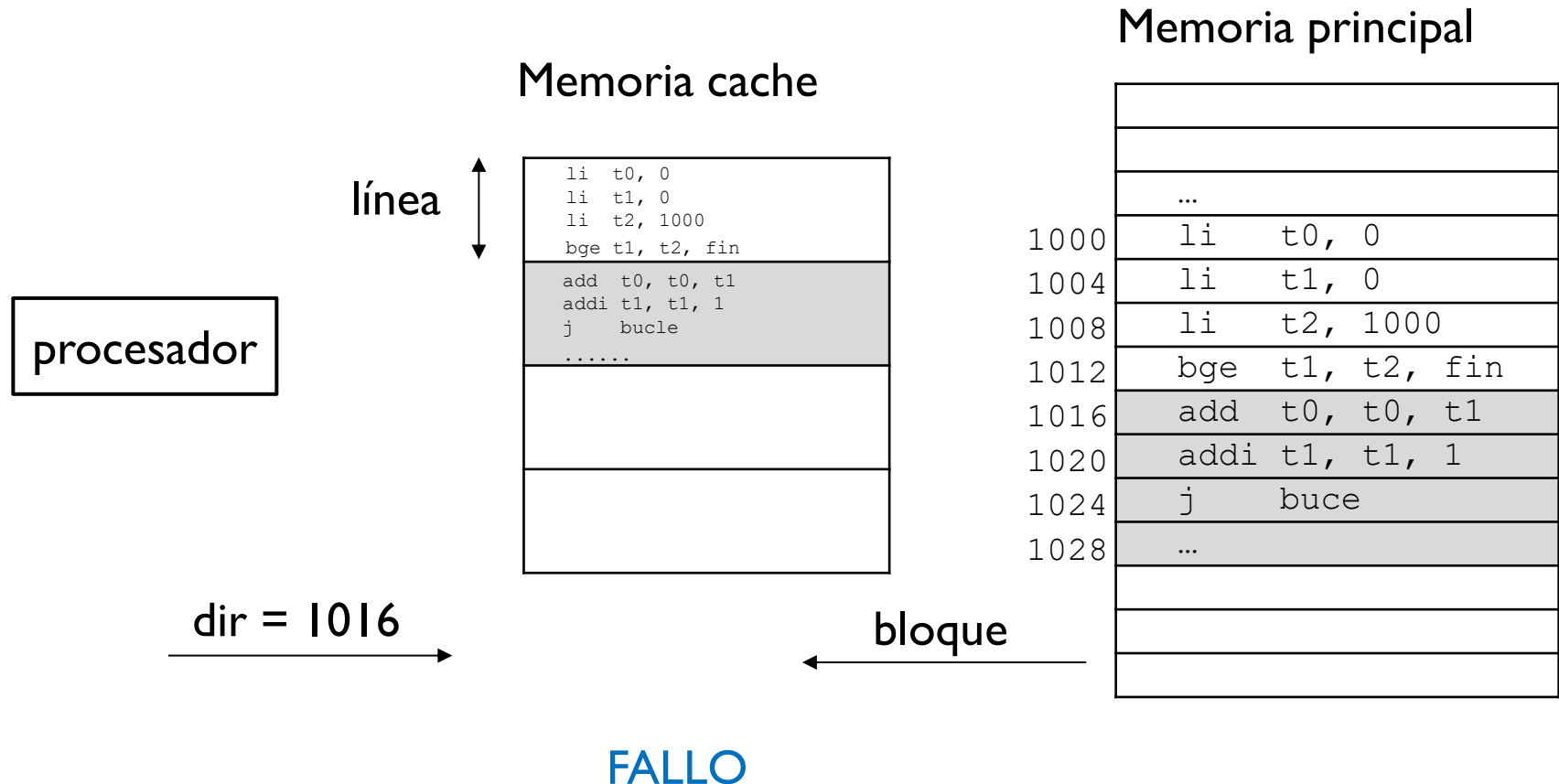
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



# Ejemplo de funcionamiento

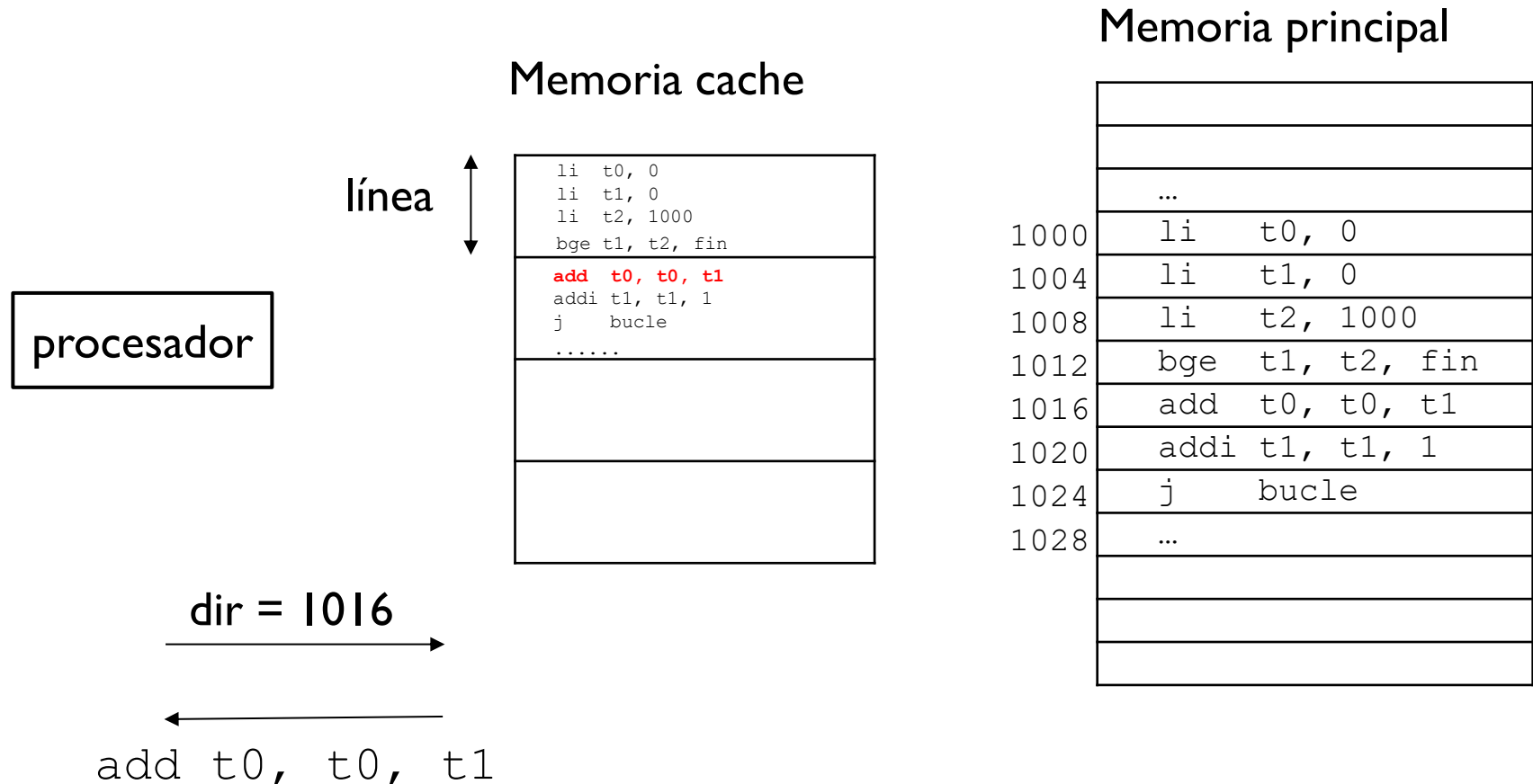
(2/2) estudio de referencias generadas





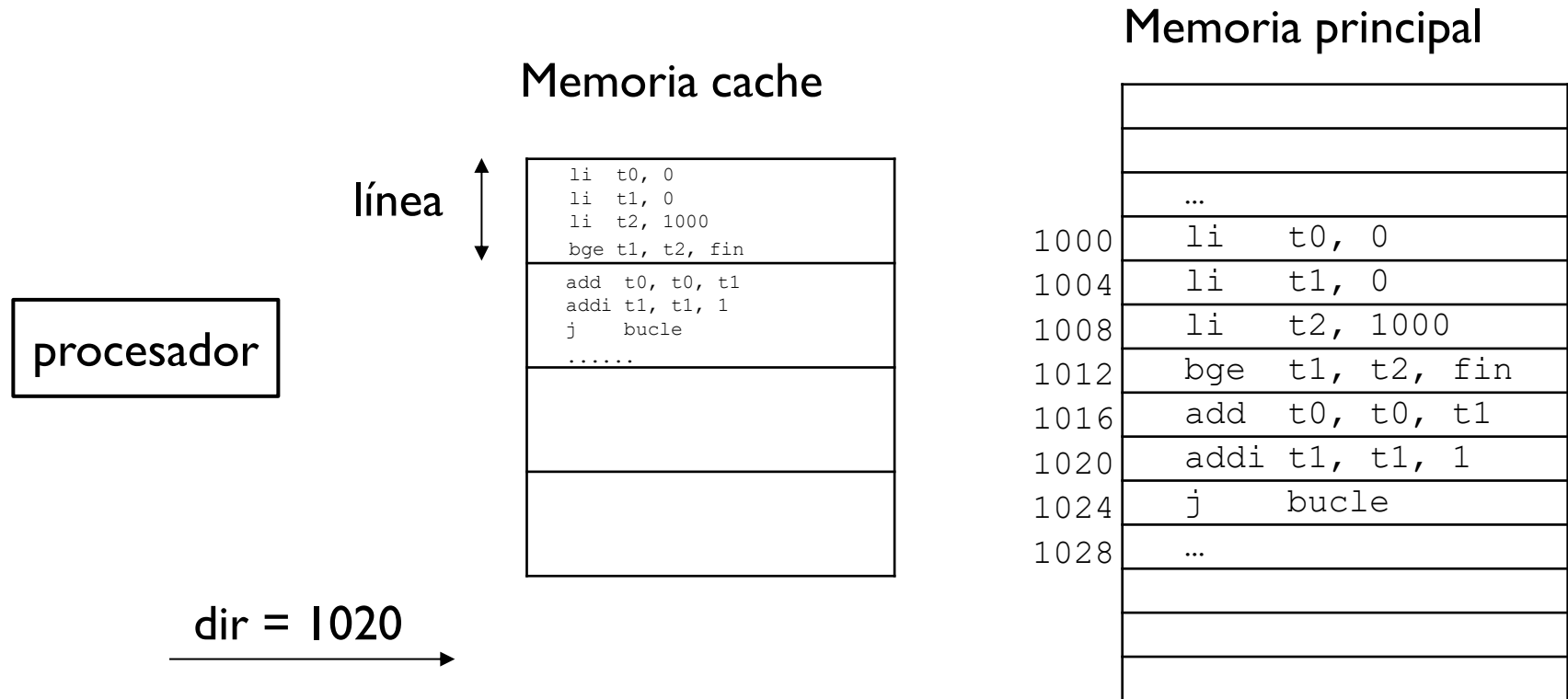
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



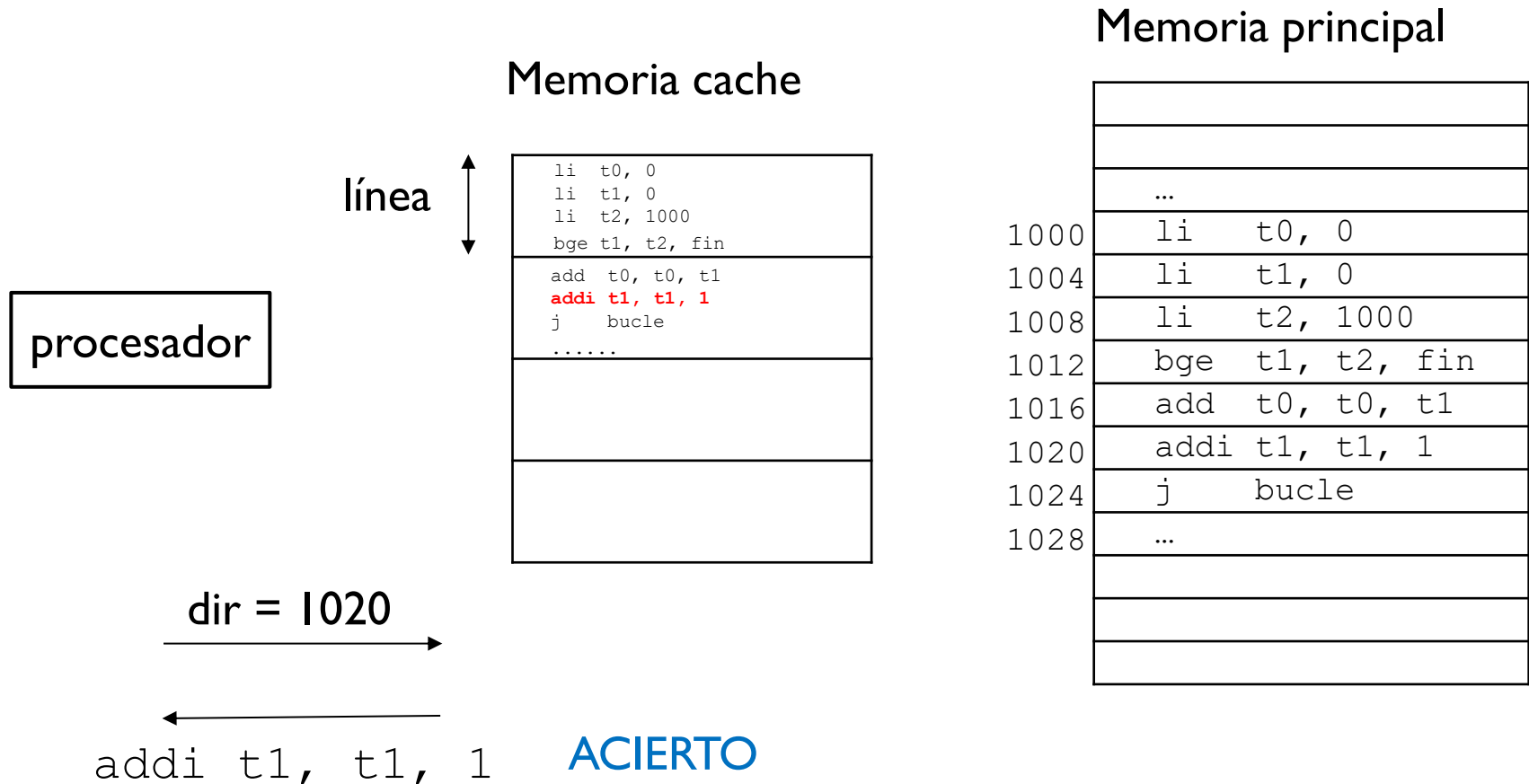
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



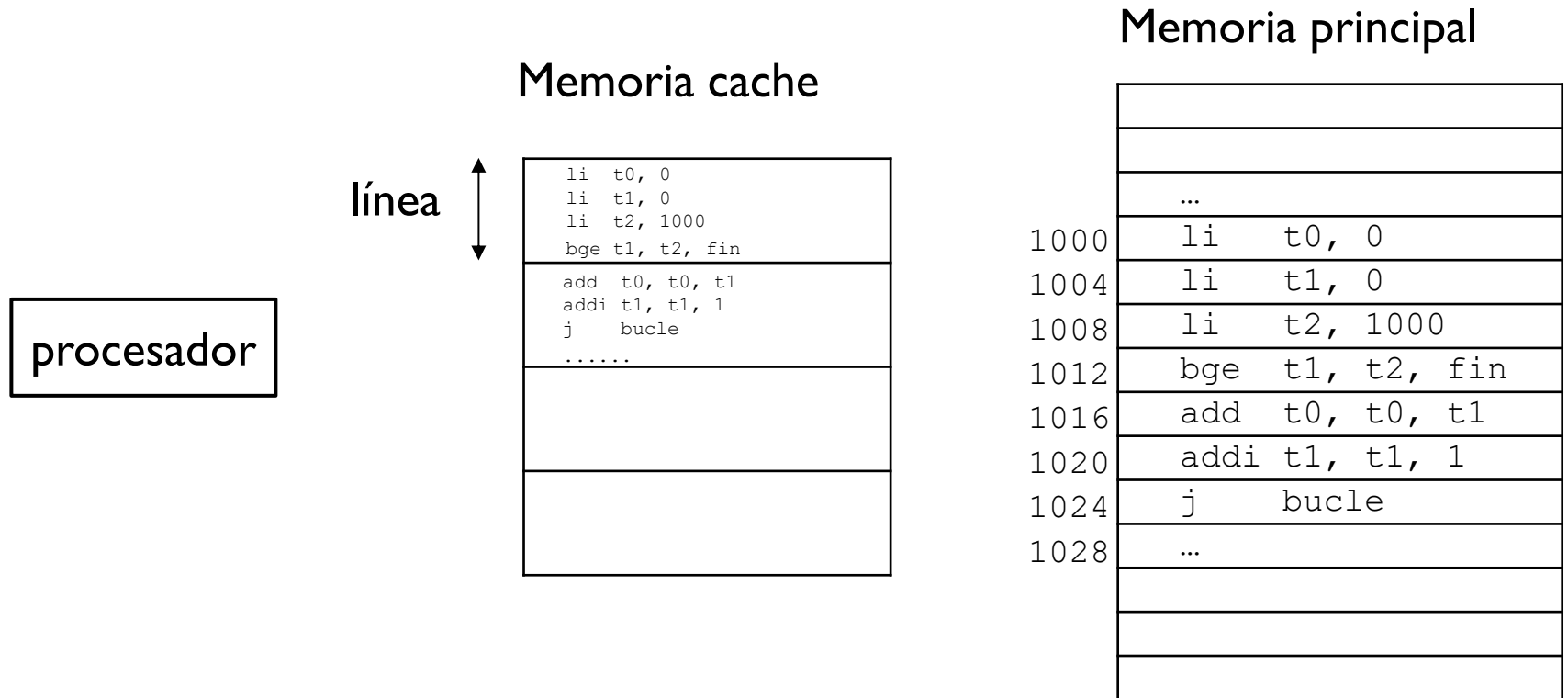
# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



# Ejemplo de funcionamiento

(2/2) estudio de referencias generadas



El resto de los accesos son **ACIERTOS**

# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;
```

bucle:

```
li    t0, 0    # s  
li    t1, 0    # i  
li    t2, 1000  
bge   t1, t2, fin
```

fin:

```
add   t0, t0, t1  
addi  t1, t1, 1  
j     bucle  
...
```

## ► Con memoria caché (bloque de 4 palabras):

- Número de accesos = 4004 accesos
- Número de bloques = 2
- Número de fallos = 2
- Tiempo de acceso = ?

(2/2) estudio de referencias generadas por ejecución: accesos a código (fetch) y datos

# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;
```

bucle:

```
li    t0, 0      # s  
li    t1, 0      # i  
li    t2, 1000  
bge   t1, t2, fin
```

fin:

```
add   t0, t0, t1  
addi  t1, t1, 1  
j     bucle  
...
```

## ► Con memoria caché (bloque de 4 palabras):

- Número de accesos = 4004 accesos
- Número de bloques = 2
- Número de fallos = 2
- **Tiempo de acceso = 8408 ns**

- Tiempo para transferir 2 bloques =  $200 \times 2 = 400$  ns

- Tiempo de acceso a la caché =  $4004 \times 2 = 8008$  ns

- Acceso a M.caché: 2 ns
- Acceso a M.P.: 120 ns
- Bloque de caché: 4 palabras
- Transferencia de un bloque entre memoria principal y caché: 200 ns

# Ejemplo de funcionamiento

```
int i;  
int s = 0;  
for (i=0; i < 1000; i++)  
    s = s + i;  
  
bucle:  li    t0, 0      # s  
        li    t1, 0     # i  
        li    t2, 1000  
        bge   t1, t2, fin  
        add   t0, t0, t1  
        addi  t1, t1, 1  
        j     bucle  
fin:    ...
```

- ▶ Sin memoria caché = 480480 ns
- ▶ Con memoria caché = 8408 ns
- ▶ Tasa de aciertos a la caché =  $4002 / 4004 \Rightarrow 99,95 \%$

¡ x57 !

# Ejercicio

## Calcular tasa de aciertos

```
int v[1000]; // global
...
int i;
int s;
for (i=0; i < 1000; i++)
    s = s + v[i];
```

- ▶ **Ejemplo:**
  - ▶ Acceso a caché: 2 ns
  - ▶ Acceso a MP: 120 ns
  - ▶ Bloque de MP: 4 palabras
  - ▶ Transferencia de un bloque entre memoria principal y caché: 200 ns

```
.data
    v: .space 4000 # 4*1000

.text
main:
    li    t0, 0      # i
    li    t1, 0      # i de v
    li    t2, 1000   # num. eltos.
    li    t3, 0      # s
bucle: bge    t0, t2, fin
    lw     t4, v(t1)
    add    t3, t3, t4
    addi   t0, t0, 1
    addi   t1, t1, 4
    j      bucle
fin:     ...
```



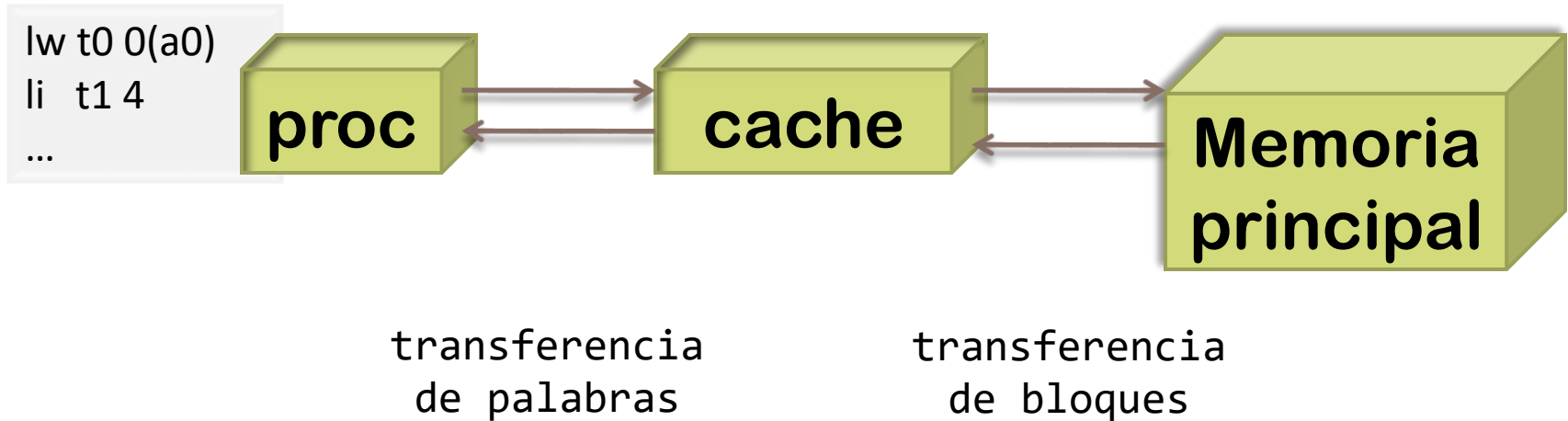
# ¿Por qué funciona?

- ▶ Tiempo de acceso a caché mucho menor que tiempo de acceso a memoria principal.
- ▶ A la memoria principal se accede por bloques.
- ▶ Cuando un programa accede a una dirección, es probable que vuelva a acceder a ella en el futuro cercano.
  - ▶ Proximidad o Localidad temporal.
- ▶ Cuando un programa accede a una dirección, es probable que en el futuro cercano acceda a posiciones cercanas.
  - ▶ Proximidad o Localidad espacial.
- ▶ Tasa de aciertos: probabilidad de que un dato accedido esté en la caché

**Tasa de aciertos elevada**

# Funcionamiento general

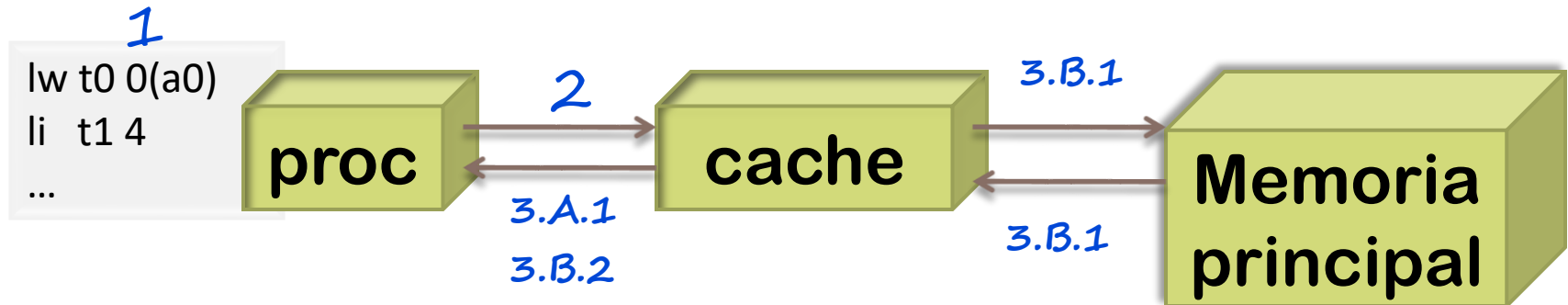
## resumen



- ▶ Se construye con tecnología SRAM
  - ▶ Integrada en el mismo procesador.
  - ▶ Más rápida y más cara que la memoria DRAM.
- ▶ Mantiene una **copia** de partes de la memoria principal.

# Funcionamiento general

## resumen



**1.** El procesador solicita contenidos de una posición (dirección) de memoria.

**2.** La cache comprueba si ya están los datos de esta posición:

**Si está (ACIERTO),**

**3.A.1** Se la sirve al procesador desde la cache (rápidamente).

**Si no está (FALLO),**

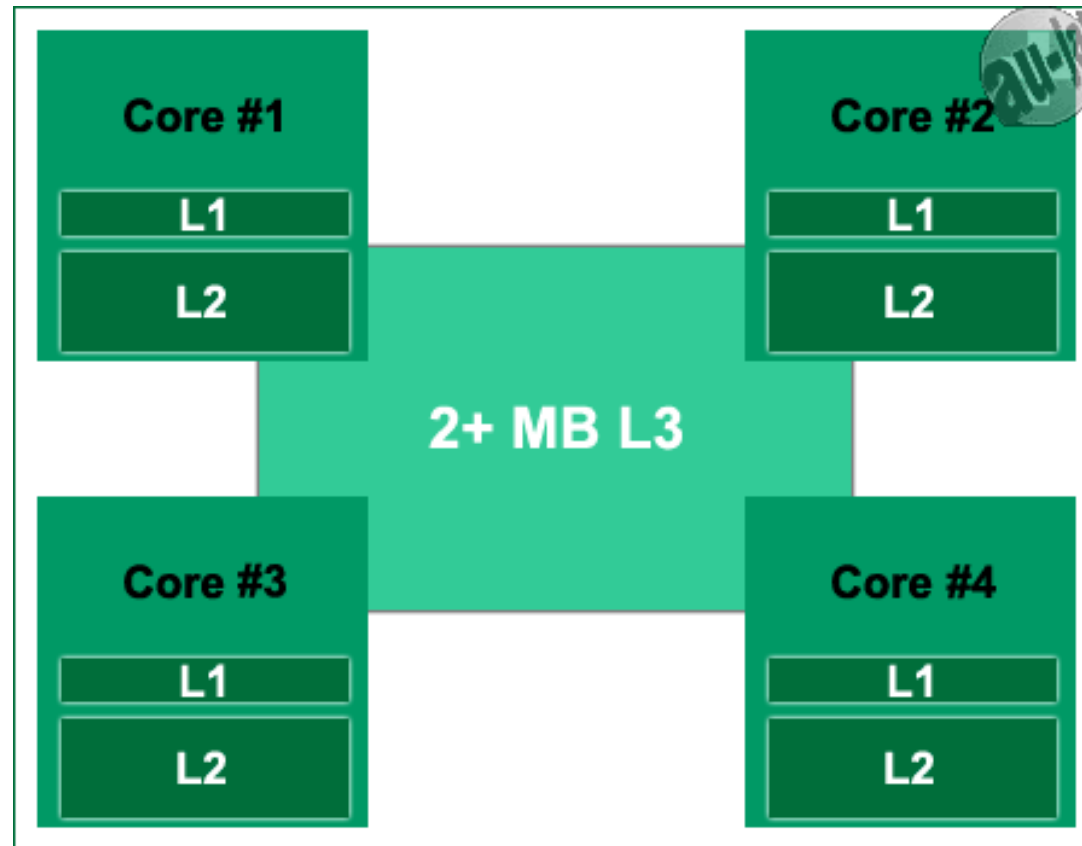
**3.B.1** La cache transfiere de memoria principal el bloque de palabras asociado a la posición.

**3.B.2** Después, la cache entrega los datos pedidos a la procesador.

# Niveles de memoria caché

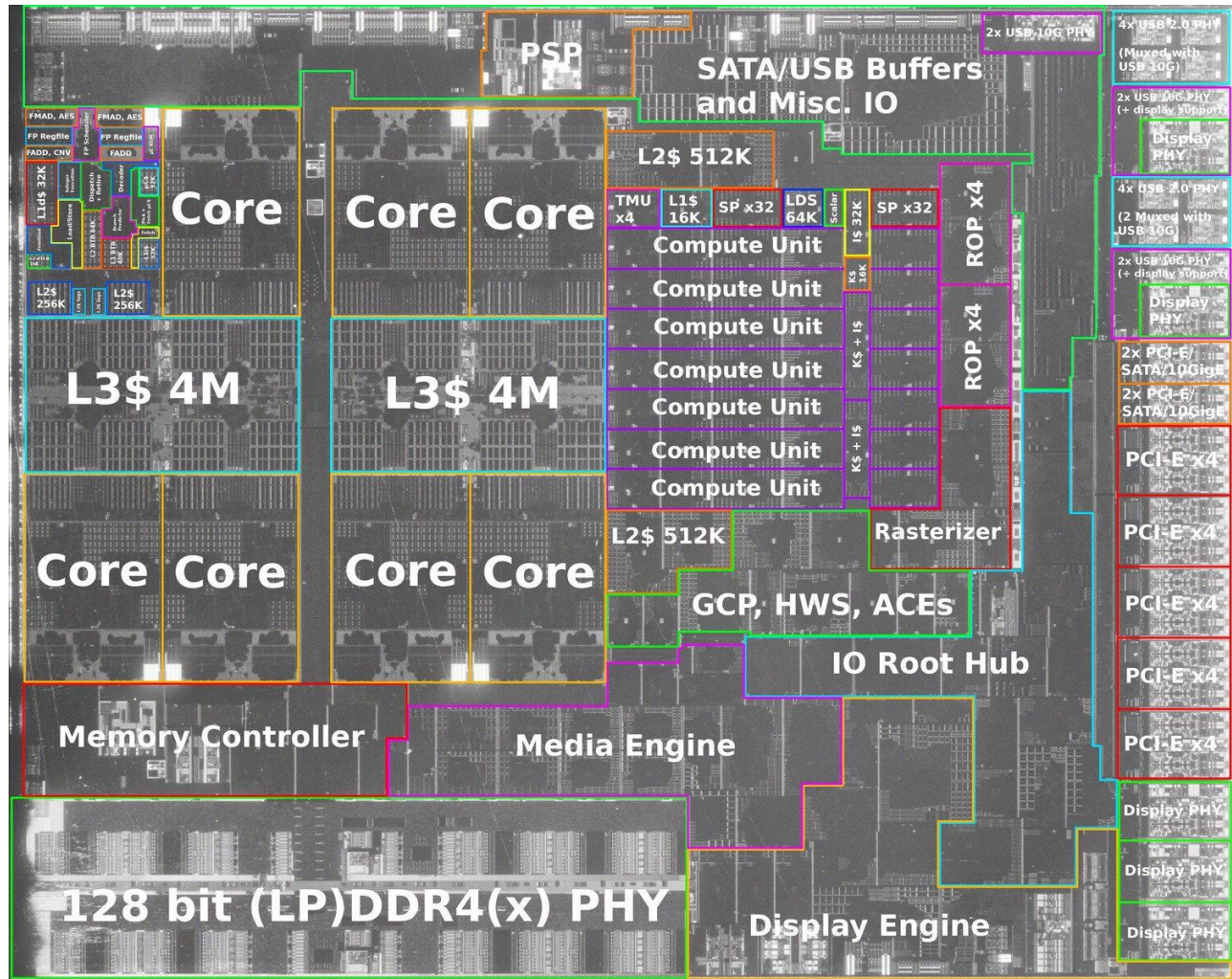
- ▶ Es habitual encontrar tres niveles:
  - ▶ **L1 o nivel 1:**
    - ▶ Caché interna: la más cercana al procesador
    - ▶ Tamaño pequeño (8KB-128KB) y máxima velocidad
    - ▶ Pueden separarse para instrucciones y datos
  - ▶ **L2 o nivel 2:**
    - ▶ Caché interna
    - ▶ Entre L1 y L3 (o entre L1 y M. Principal)
    - ▶ Tamaño mediano (256KB – 4MB) y menor velocidad que L1
  - ▶ **L3 o nivel 3:**
    - ▶ Típicamente último nivel antes de M. Principal
    - ▶ Tamaño mayor y menor velocidad que L2
    - ▶ Interna o externa al procesador

# Ejemplo: AMD quad-core



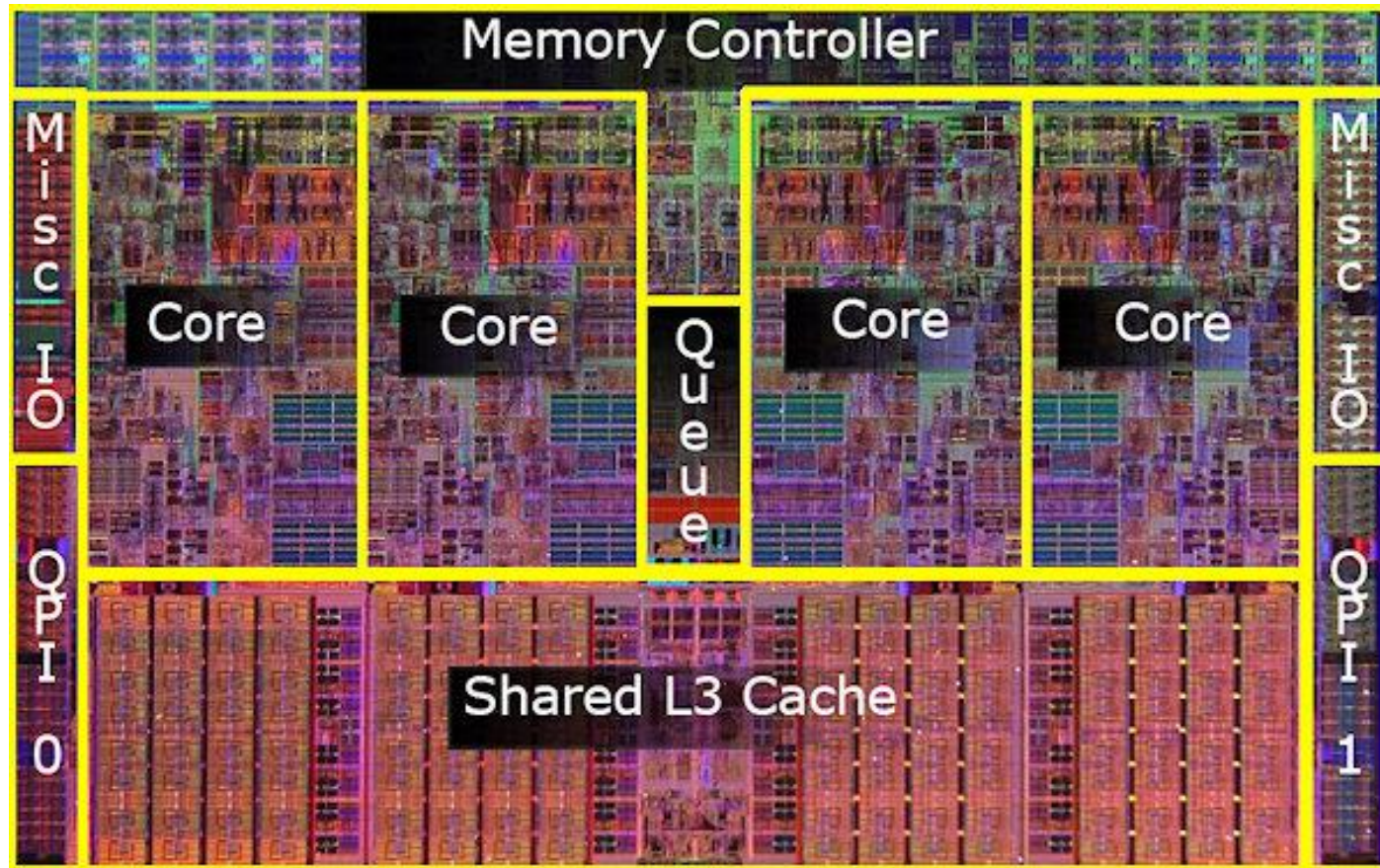
Quad-Core CPU mit gemeinsamen L3-Cache

# Ejemplo: AMD Ryzen 4000





# Ejemplo: Intel Core i7



# Contenidos

1. Tipos de memoria
2. Jerarquía de memoria
3. Memoria principal
4. Memoria caché
  1. Introducción
  2. Estructura de la memoria caché
  3. Diseño y organización de la memoria caché
5. Memoria virtual



# Acceso a una palabra en m. principal

Recordatorio

## ► Ejemplo:

- Computador de 32 bits
- Memoria direccionada por bytes
- Acceso a memoria principal por palabras
- Acceso a la palabra con dirección  
64 en decimal

← Palabra de 32 bits →

0				
4				
8				
12				
16				
20				
24				
28				
32				
36				
40				
44				
48				
52				
56				
60				
64				

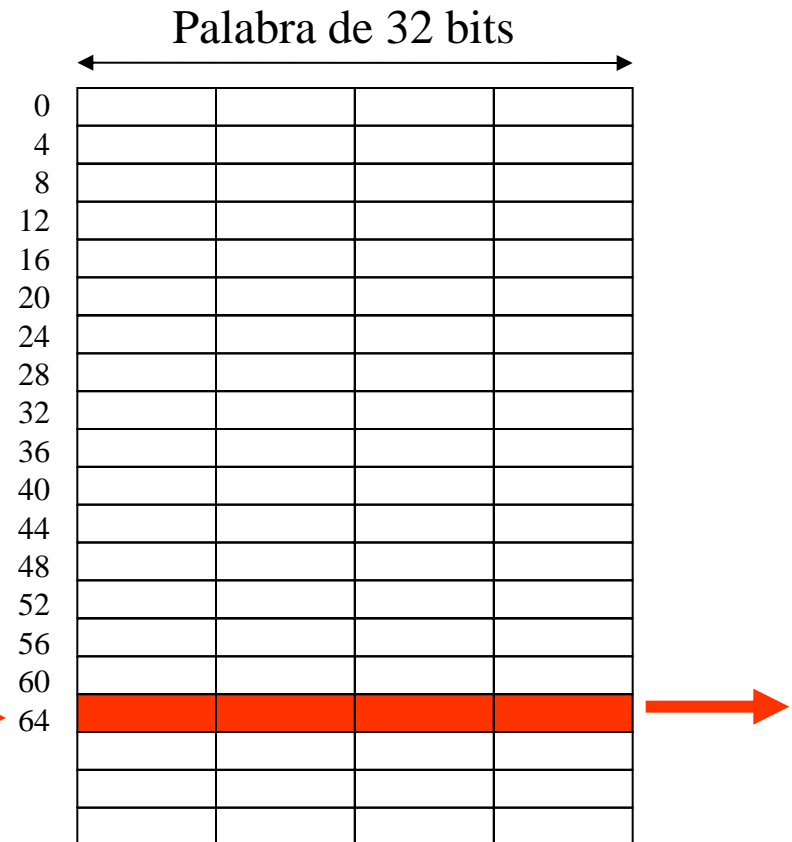
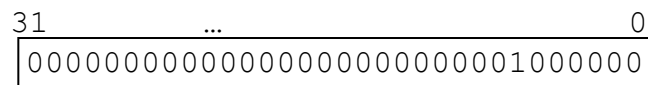
# Acceso a una palabra en m. principal

Recordatorio

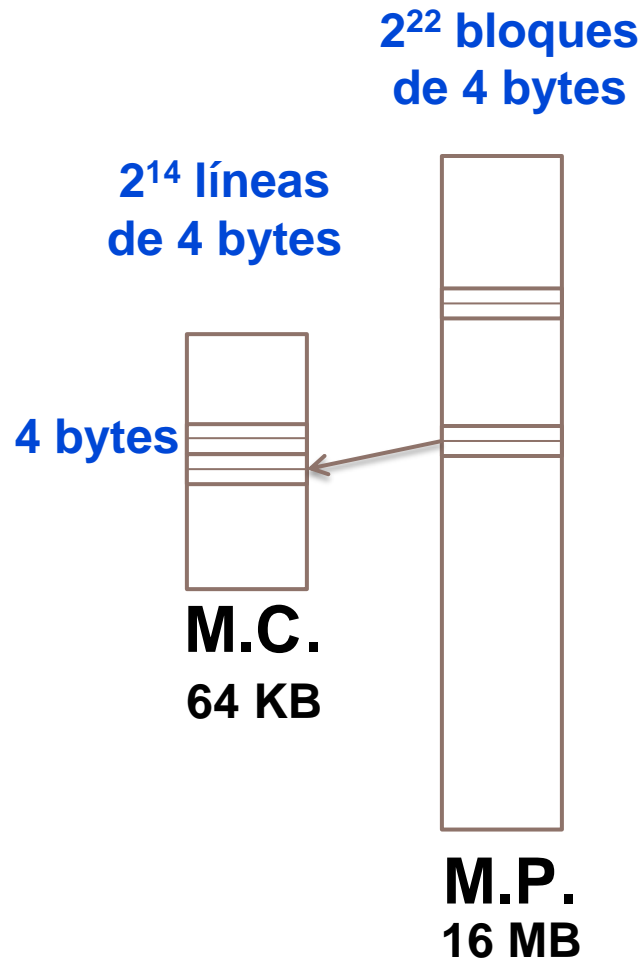
## ► Ejemplo:

- Computador de 32 bits
- Memoria direccionada por bytes
- Acceso a memoria principal por palabras
- Acceso a la palabra con dirección

64 en decimal

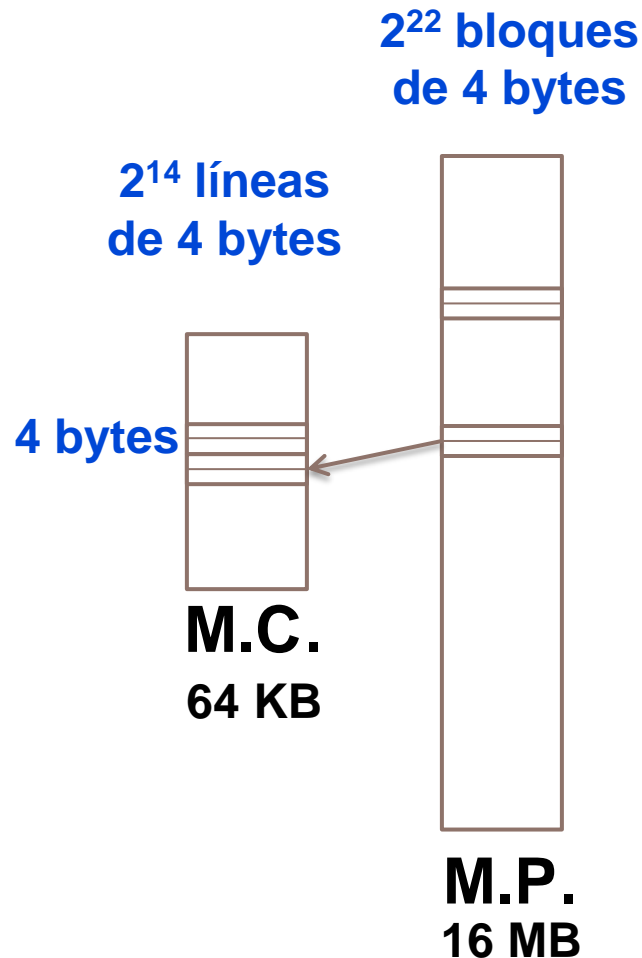


# Estructura de la memoria caché



- ▶ Se divide la M.P. y la M.C. (de forma lógica) en bloques de igual tamaño
  - ▶ El bloque en caché se le llama **línea**
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)
- ▶ El tamaño de la M.C. es menor:
  - ▶ El número de bloques en la memoria caché es pequeño.

# Estructura de la memoria caché

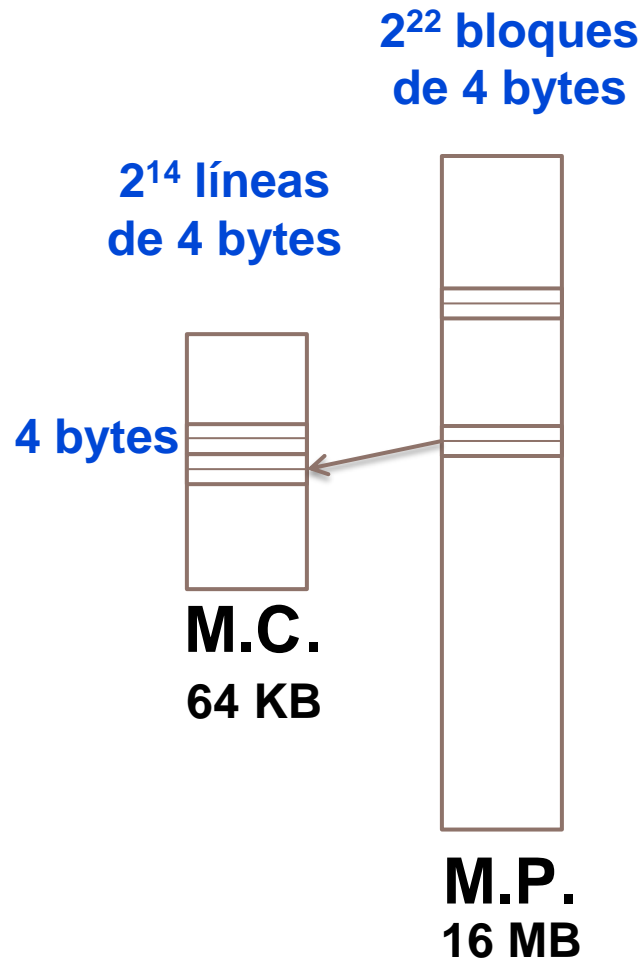


- ▶ Se divide la M.P. y la M.C. (de forma lógica) en bloques de igual tamaño
  - ▶ El bloque en caché se le llama **línea**
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)
- ▶ El tamaño de la M.C. es menor:
  - ▶ El número de bloques en la memoria caché es pequeño.

¿Cuántos bloques de 4 palabras caben en una m. caché de 64 KiB para una CPU de 32 bits?

- ▶ Solución:  
 $2^6 \cdot 2^{10} \text{ bytes} / 2^4 \text{ bytes} = 2^{11} \text{ bloques} = 2048 \text{ bloques} = 2048 \text{ líneas}$

# Estructura de la memoria caché



- ▶ Se divide la M.P. y la M.C. (de forma lógica) en bloques de igual tamaño
  - ▶ El bloque en caché se le llama **línea**
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)
- ▶ El tamaño de la M.C. es menor:
  - ▶ El número de bloques en la memoria caché es pequeño.

¿Cuántos bloques de 4 palabras de 4 bytes hay en una memoria de 1 GiB?

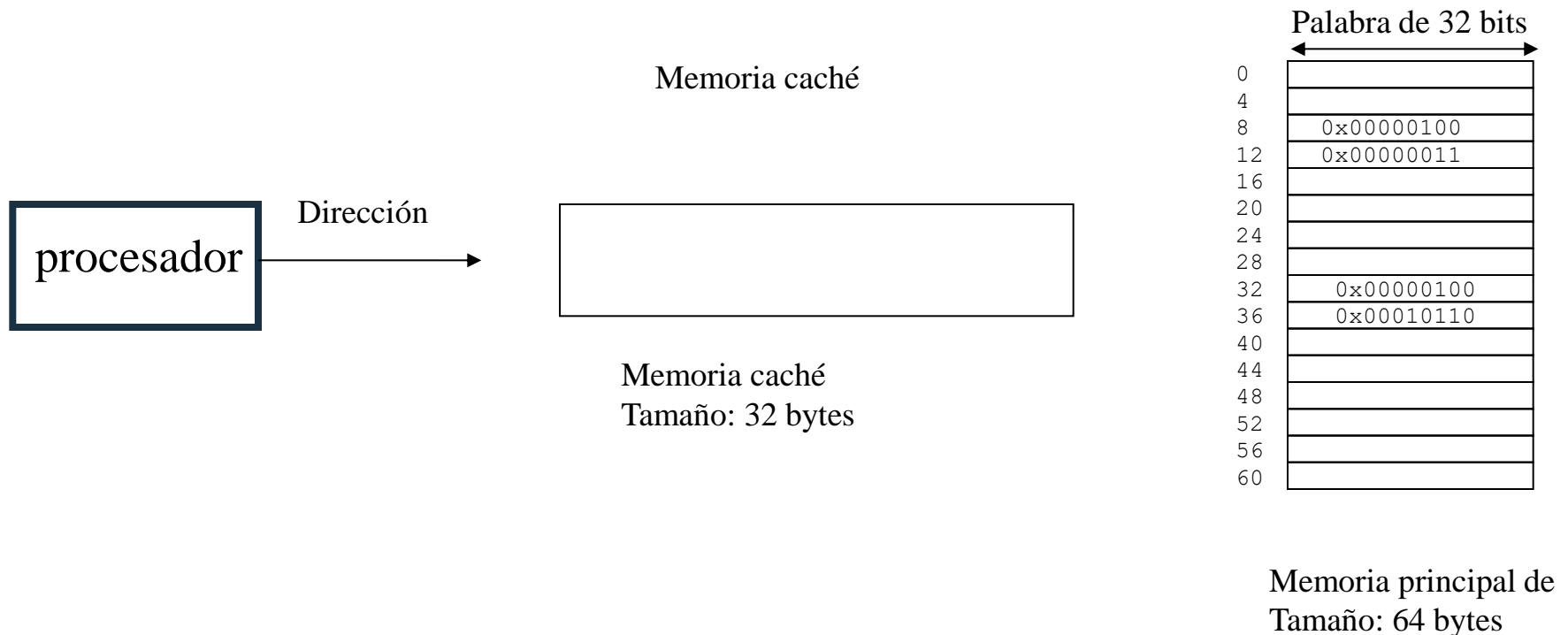
- ▶ Solución:

$$2^{30} \text{ b} / 2^4 \text{ b} = 2^{30-4} \text{ b} = 2^{26} \text{ b} = 64 \text{ megabloques} = \sim 64 \text{ millones}$$

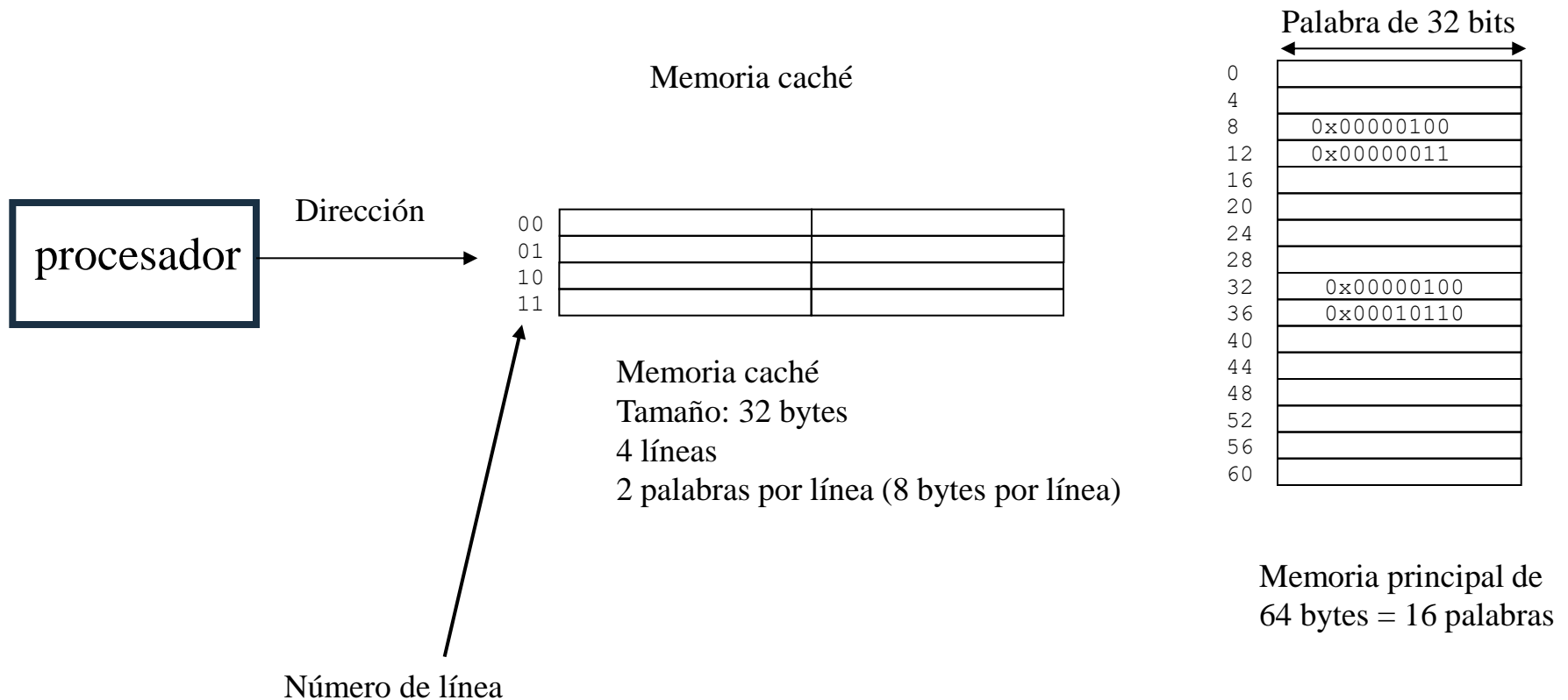
# ¿Cómo buscar una palabra en caché?

## Ejemplo:

Con líneas de dos palabras  
¿cuántas líneas tiene la caché?



# ¿Cómo buscar una palabra en caché?

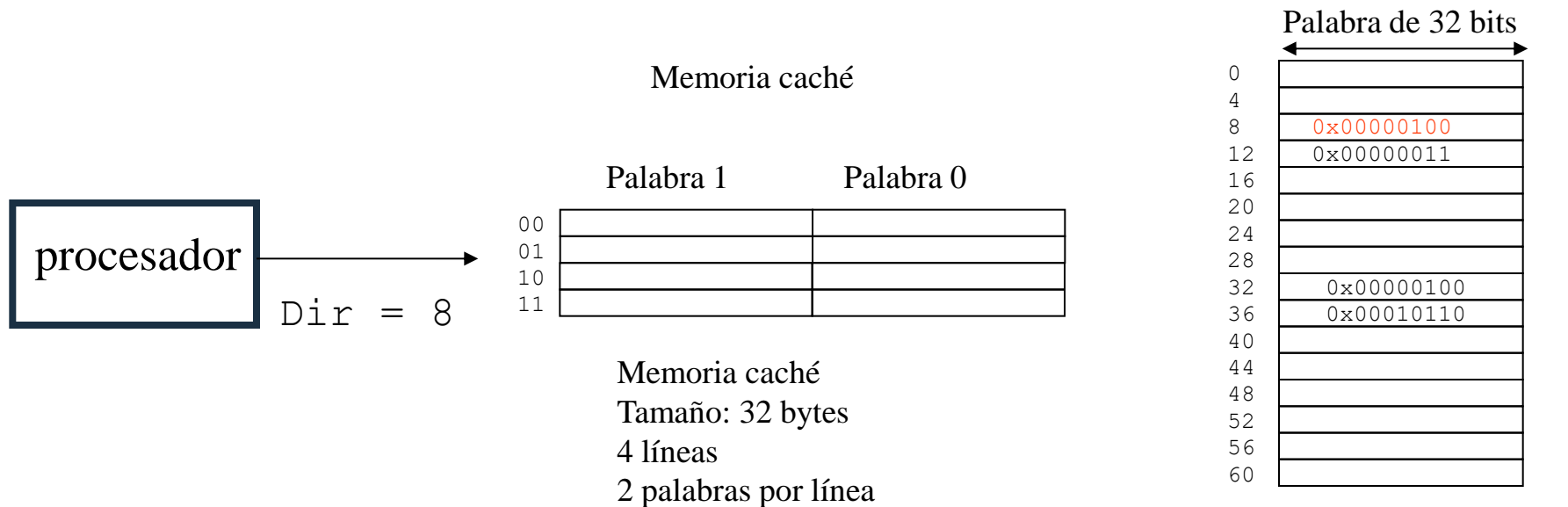


# ¿Cómo buscar una palabra en caché?





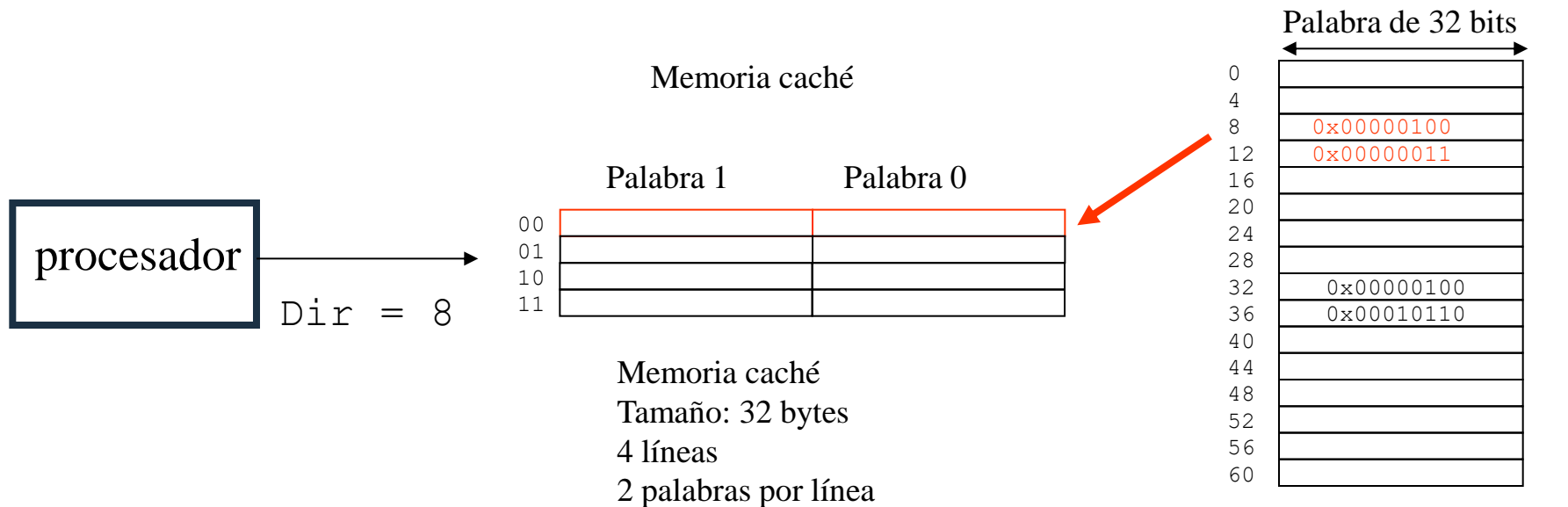
# ¿Cómo buscar una palabra en caché?



**FALLO**  
**¿Cómo se sabe?**

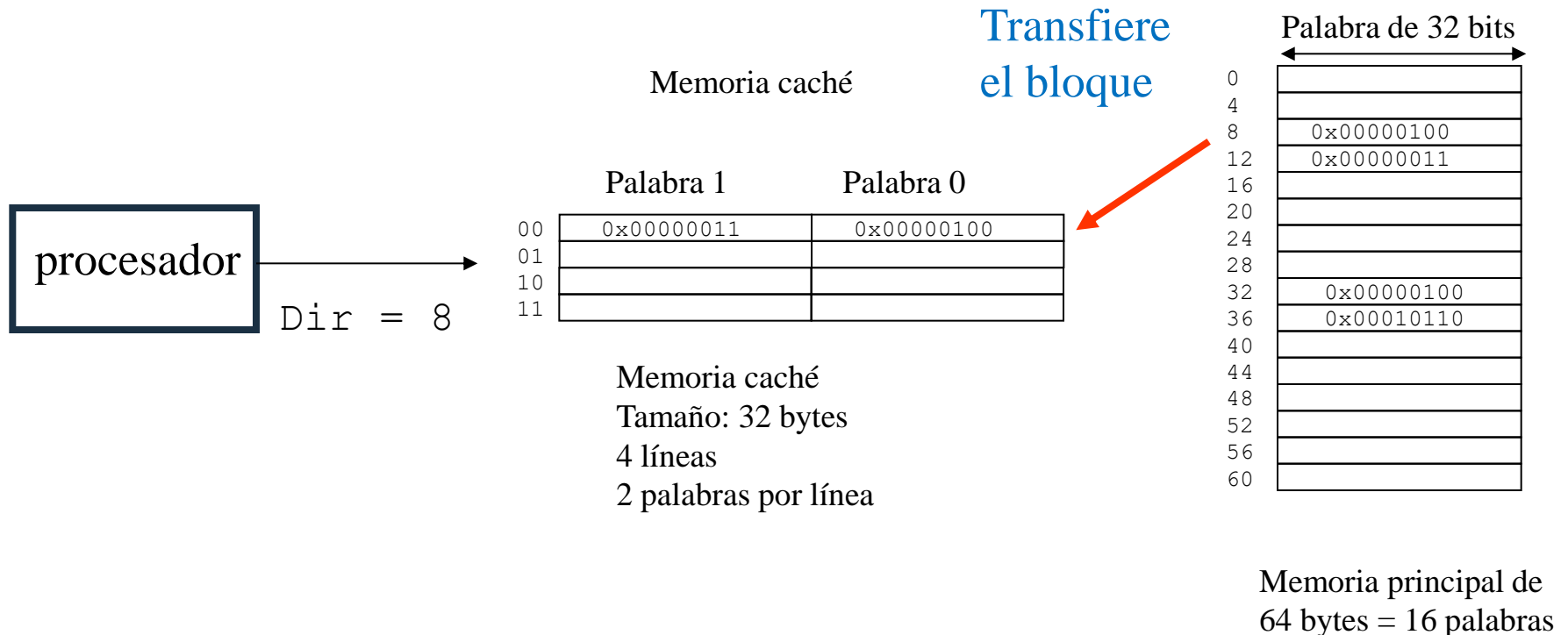
Memoria principal de  
64 bytes = 16 palabras

# ¿Cómo buscar una palabra en caché?

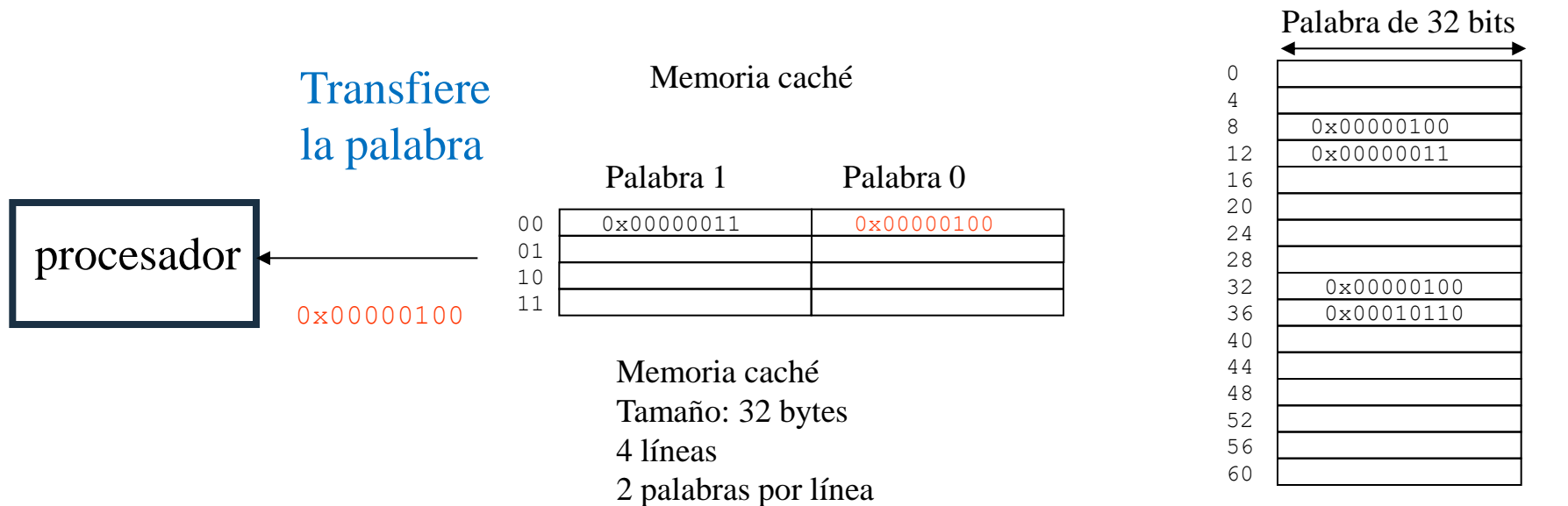


Se elige una línea en la caché  
¿Qué línea?

# ¿Cómo buscar una palabra en caché?



# ¿Cómo buscar una palabra en caché?



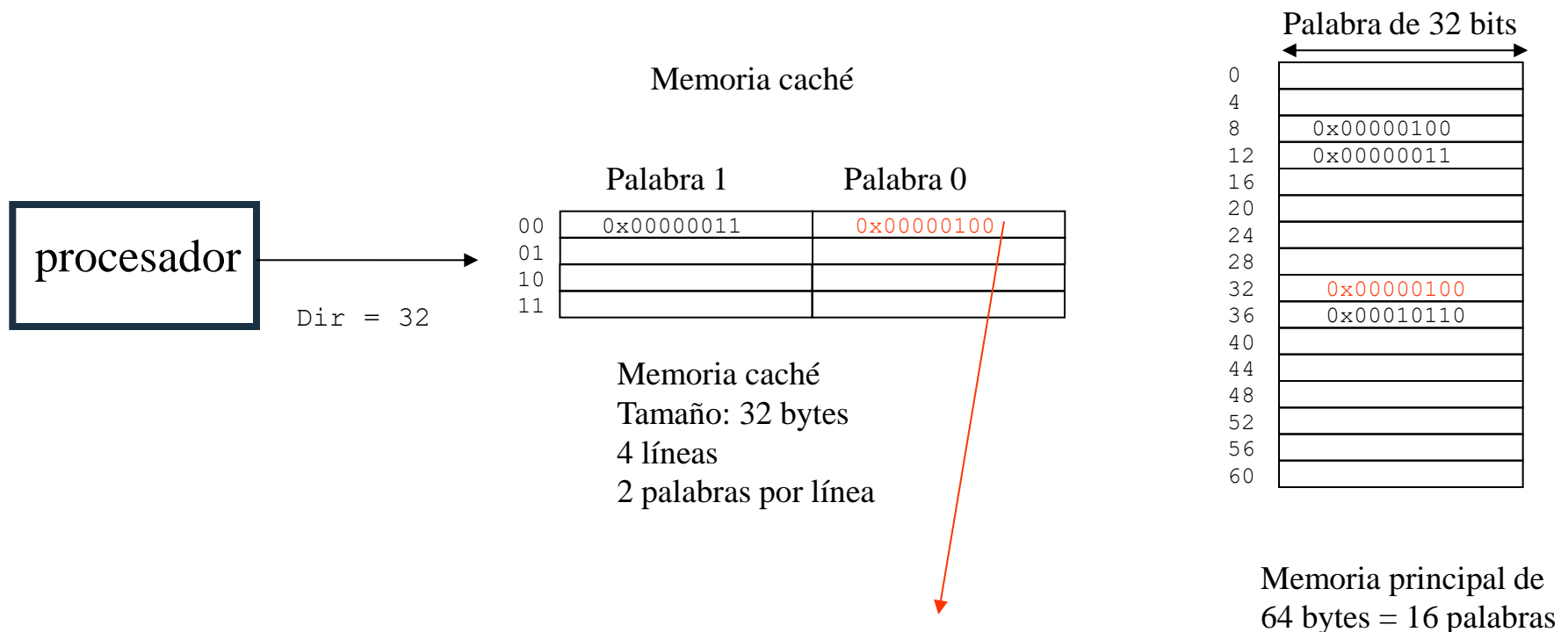
Memoria principal de  
64 bytes = 16 palabras

# ¿Cómo buscar una palabra en caché?



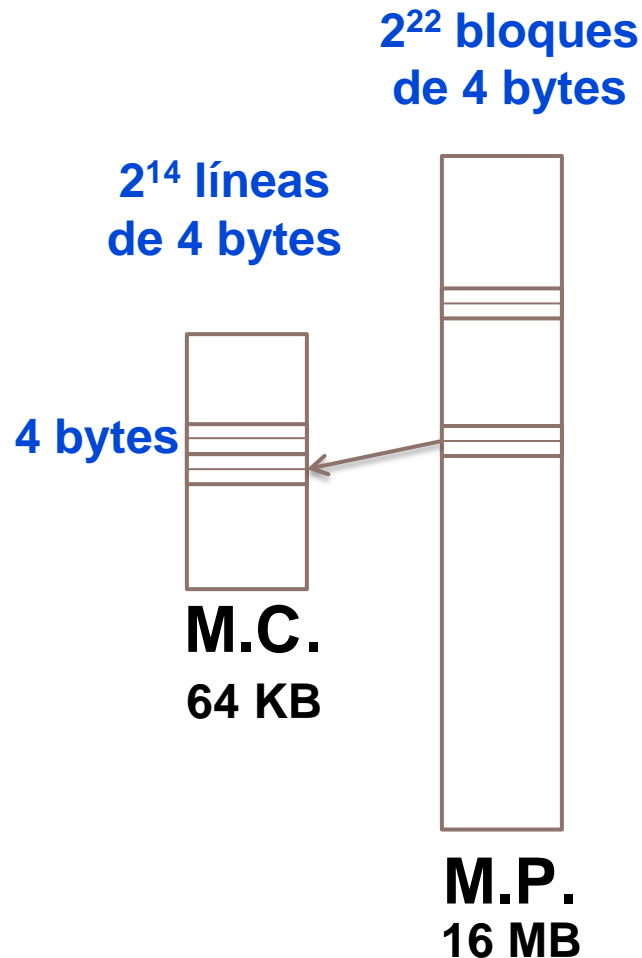
¿Cómo saber si está en la caché?

# ¿Cómo buscar una palabra en caché?



Recordando: el contenido es de la dirección 8 no de la 32

# Estructura de la memoria caché



- ▶ Se divide la M.P. y la M.C. (de forma lógica) en bloques de igual tamaño
  - ▶ El bloque en caché se le llama **línea**
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)
- ▶ El tamaño de la M.C. es menor:
  - ▶ El número de bloques en la memoria caché es pequeño.

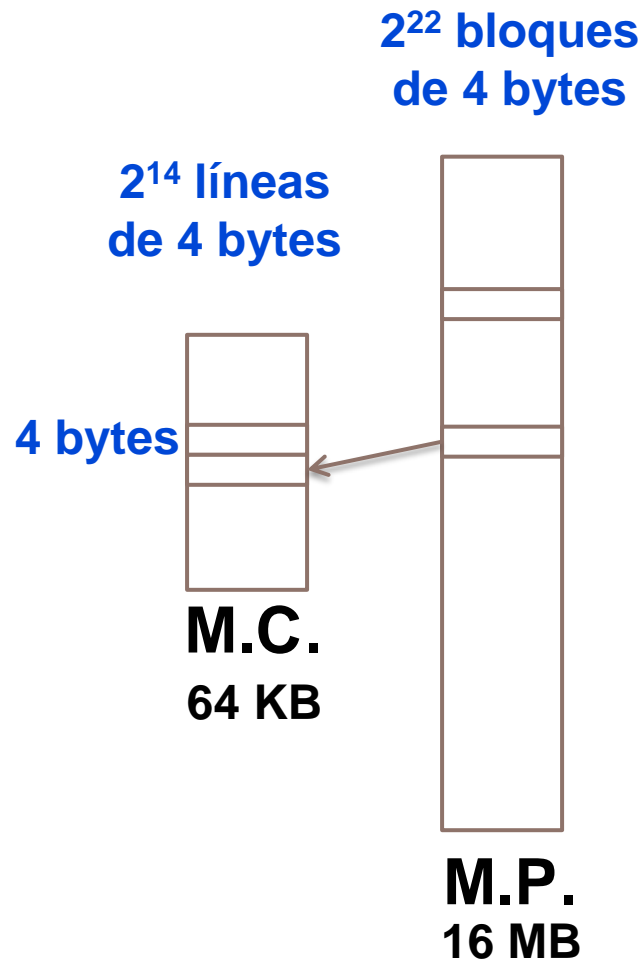
1. ¿Dónde se ubica un bloque de M.P.?
2. ¿Cómo se identifica un bloque de M.P.?
3. En caso de fallo y M.C. llena...  
¿Qué bloque debe remplazarse?
4. En caso de escritura...  
¿Qué debe actualizarse? ¿M.C.? ¿M.P. y M.C.?

# Contenidos

1. Tipos de memoria
2. Jerarquía de memoria
3. Memoria principal
4. Memoria caché
  1. Introducción
  2. Estructura de la memoria caché
  3. Diseño y organización de la memoria caché
5. Memoria virtual

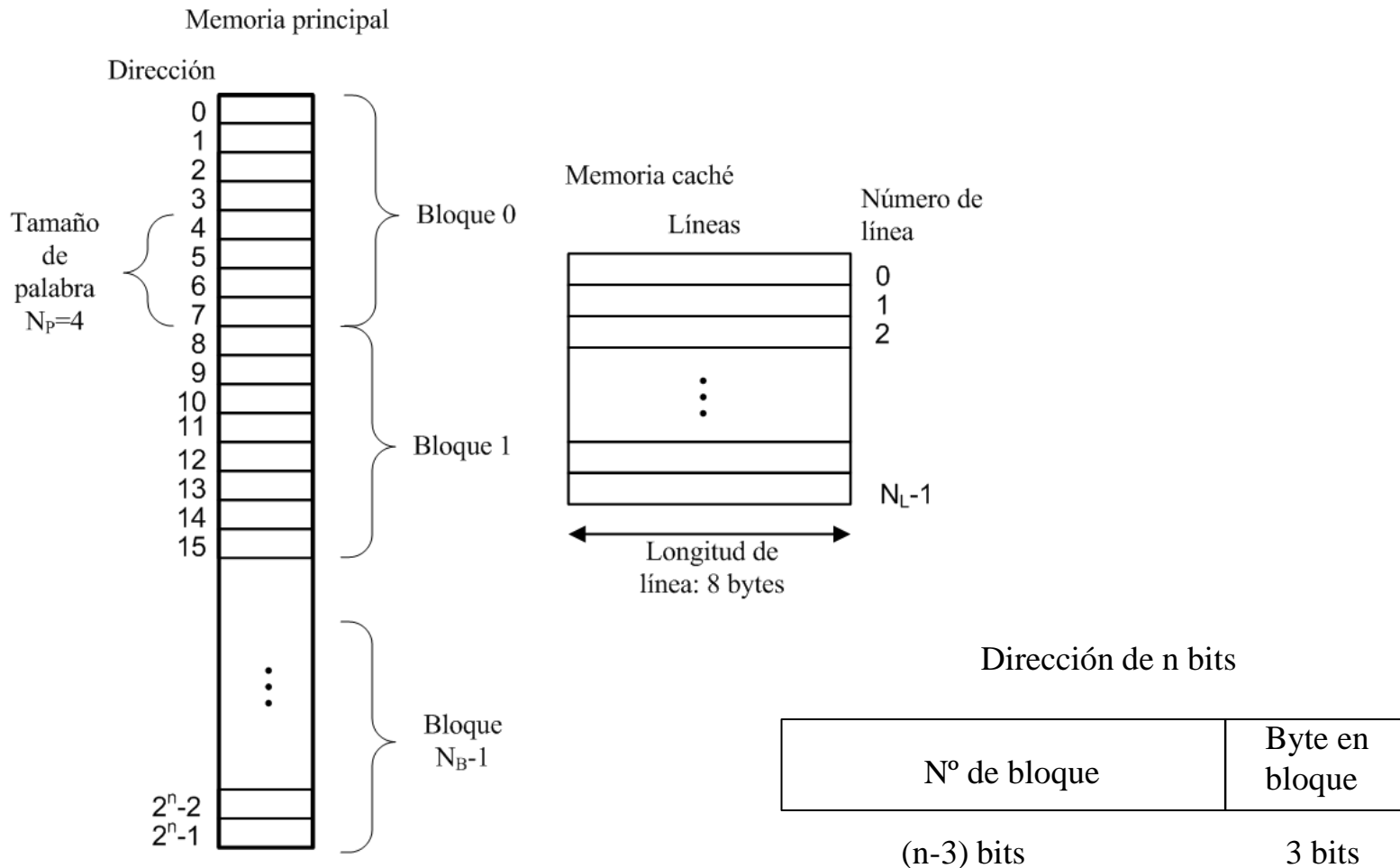


# Estructura y diseño de la cache

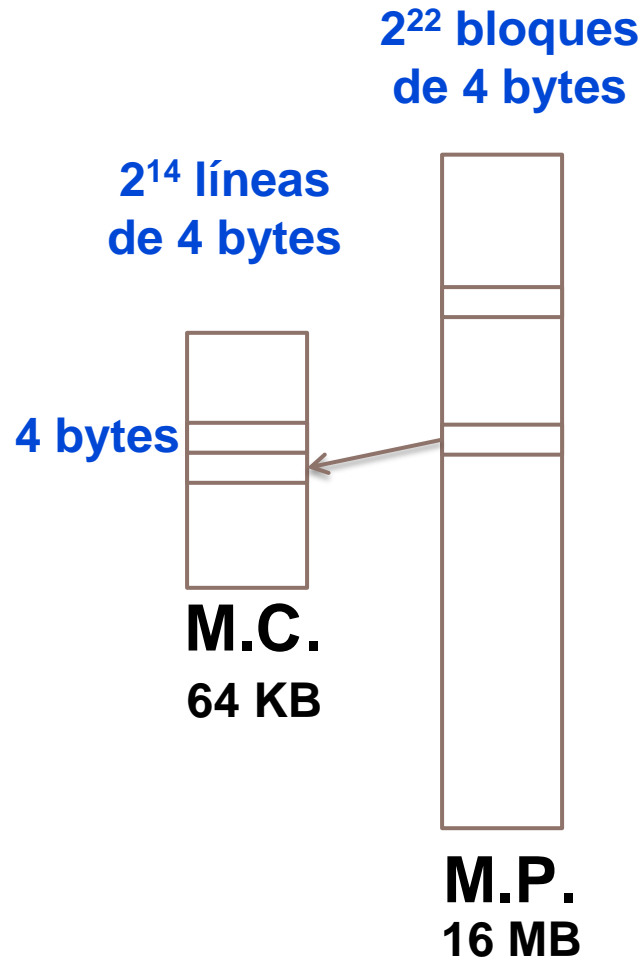


- ▶ Se divide la M.P. y la M.C. en bloques de igual tamaño
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)

# Ejemplo de organización de la memoria caché

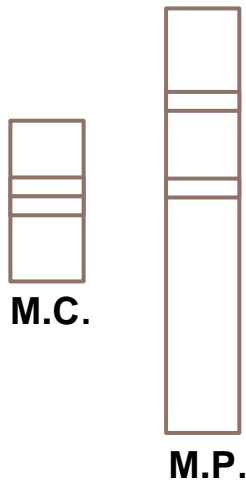


# Estructura y diseño de la cache



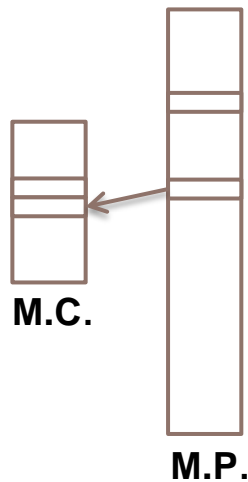
- ▶ Se divide la M.P. y la M.C. en bloques de igual tamaño
- ▶ A cada bloque de M.P. le corresponderá **una línea de M.C.** (bloque en caché)
- ▶ En el diseño se determina:
  - ▶ Tamaño
  - ▶ Función de correspondencia
  - ▶ Algoritmo de sustitución
  - ▶ Política de escritura
- ▶ Es habitual distintos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

# Tamaño de caché



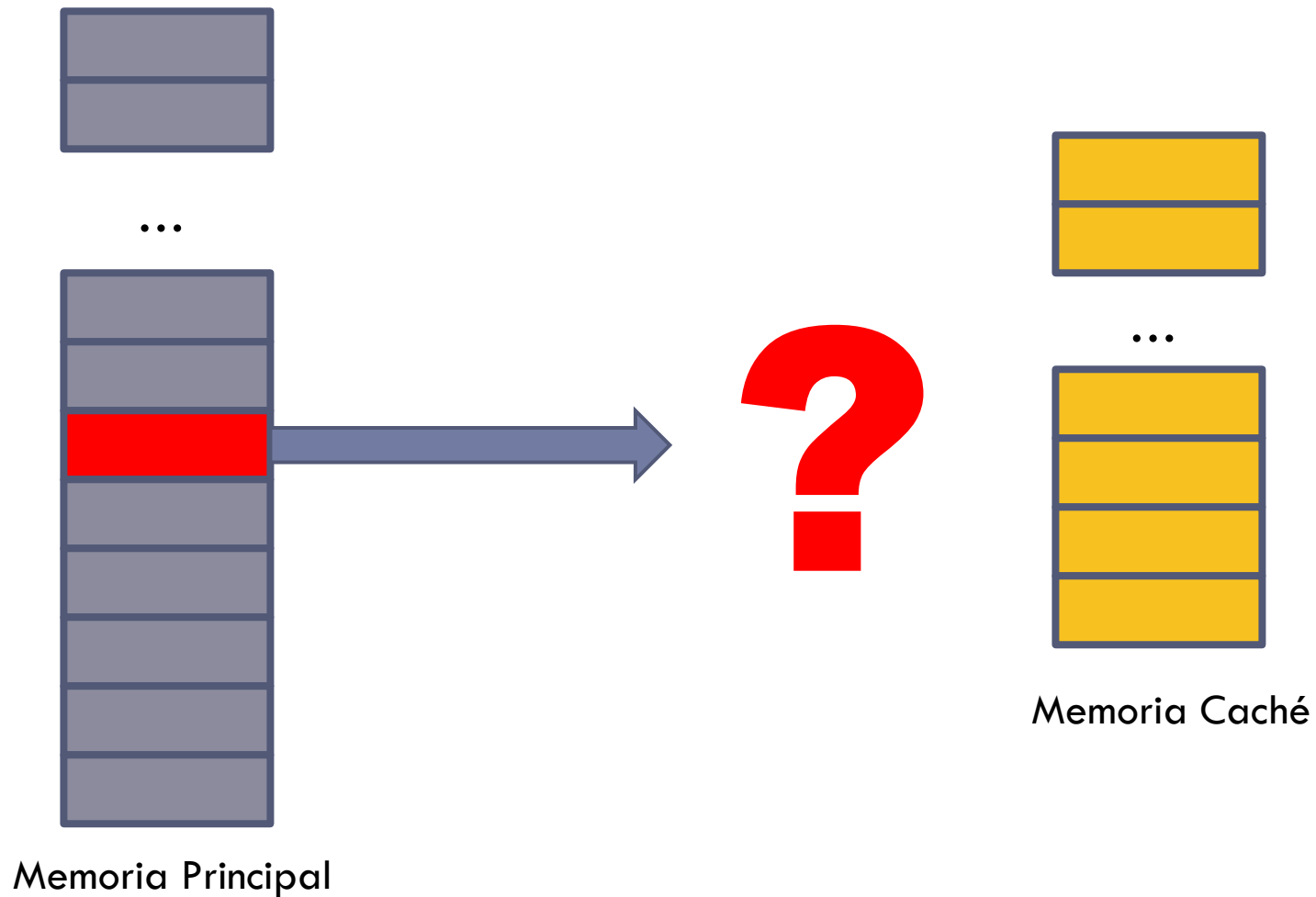
- ▶ **Tamaños:**
  - ▶ Total de la memoria caché.
  - ▶ De las líneas en los que se organiza la memoria caché.
- ▶ Se determina por estudios sobre códigos muy usados

# Función de correspondencia

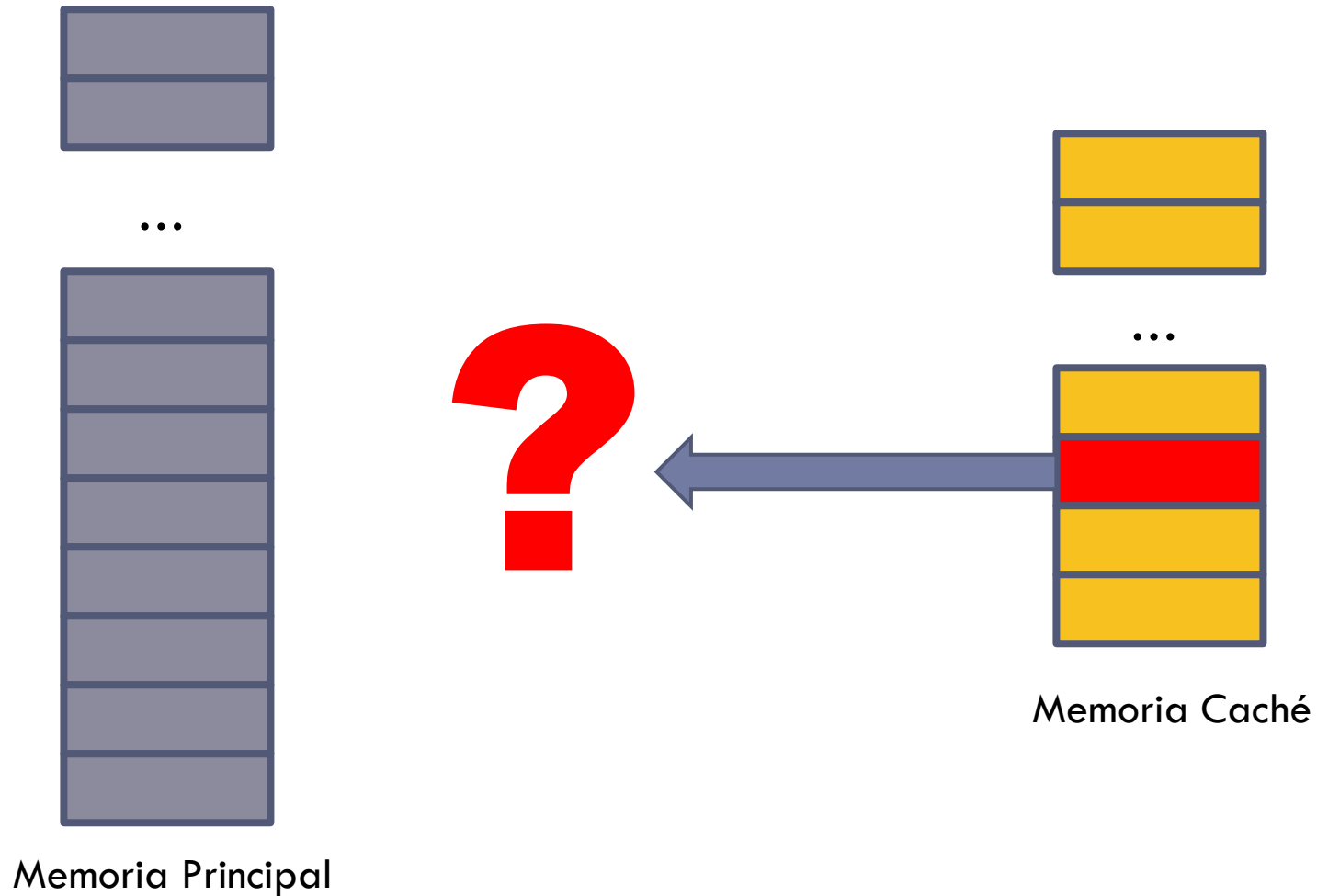


- ▶ Algoritmo que determina, para un bloque concreto de la memoria principal, en qué lugar o lugares de la memoria caché se puede almacenar.
- ▶ Un mecanismo que permita saber para una línea de la memoria caché, qué bloque concreto de memoria principal está en ese momento (o si está libre).
  - ▶ Se asocian a las líneas etiquetas.
  - ▶ Las etiquetas se basan en la dirección de comienzo de la línea.

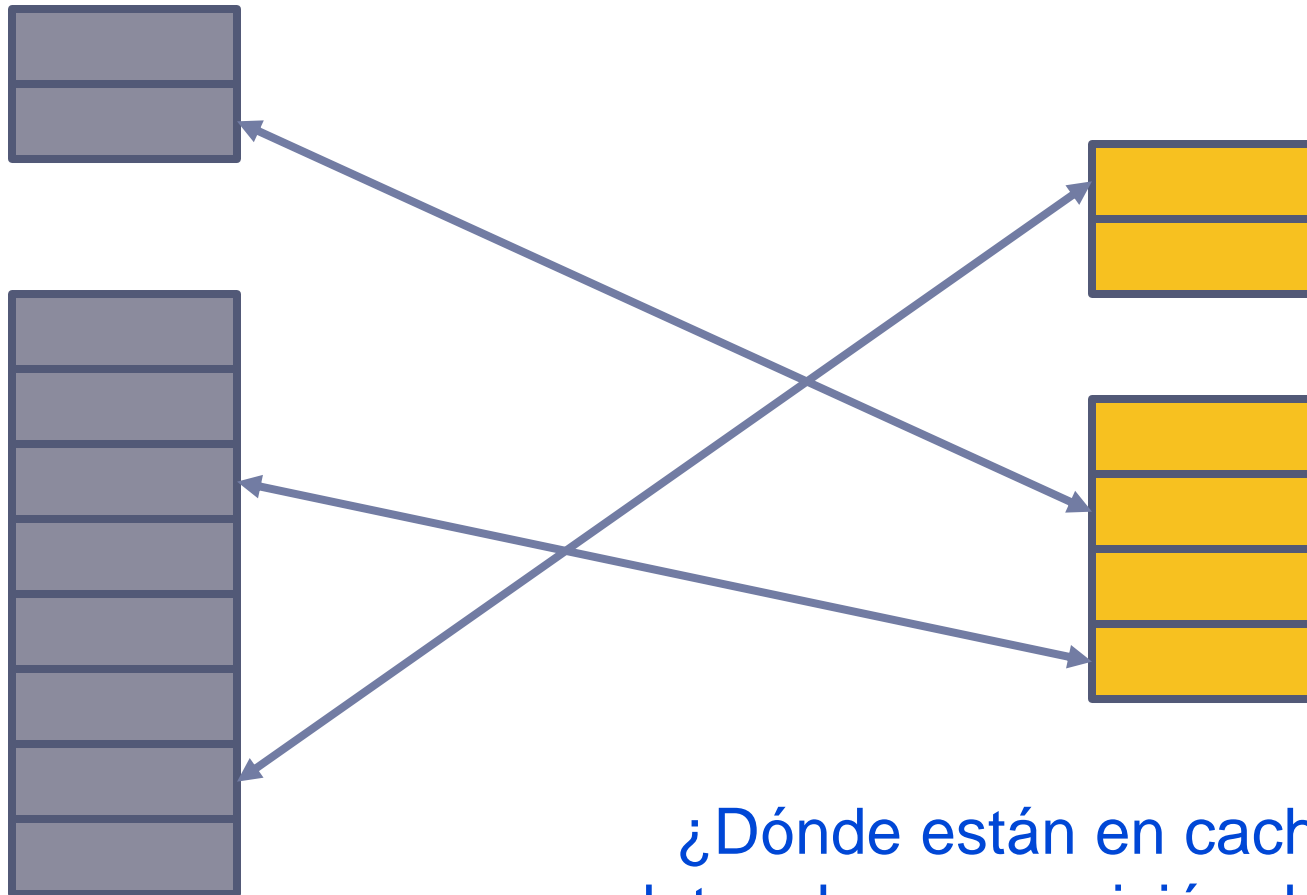
# Ubicación en caché



# Ubicación en memoria principal



# Función de correspondencia



¿Dónde están en caché los  
datos de una posición de M.P.?



# Funciones de correspondencia

- ▶ Función de correspondencia **directa**
- ▶ Función de correspondencia **asociativa**
- ▶ Función de correspondencia **asociativa por conjuntos**

## Correspondencia **directa**

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

## Memoria principal

Palabra de 32 bits

[illegible]

# Correspondencia directa

Bloque 0 - línea 0

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

## Memoria principal

Palabra de 32 bits

[illegible]

# Correspondencia directa

## Bloque 1 - línea 1

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

## Memoria principal

Palabra de 32 bits

[illegible]

# Correspondencia directa

Bloque 2 - línea 2

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

## Memoria principal

Palabra de 32 bits

0  
4  
8  
12  
16  
20  
24  
28  
32  
36  
40  
44  
48  
52  
56  
60  
64  
.  
.  
.

# Correspondencia directa

Bloque 3 - línea 3

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

## Memoria principal

Palabra de 32 bits

[illegible]

# Correspondencia directa

## Bloque 4 - línea 0

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

## Memoria principal

Palabra de 32 bits

Diagram illustrating a 32x4 grid structure, likely representing a memory layout or data organization. The grid is divided into 8 rows of 4 cells each. The rows are numbered 0 to 31 on the left. The grid is colored in a repeating pattern of 4 rows each:

- Rows 0-3: Light blue
- Rows 4-7: Light blue
- Rows 8-11: Light blue
- Rows 12-15: Light blue
- Rows 16-19: Light blue
- Rows 20-23: Light blue
- Rows 24-27: Light blue
- Rows 28-31: Light blue

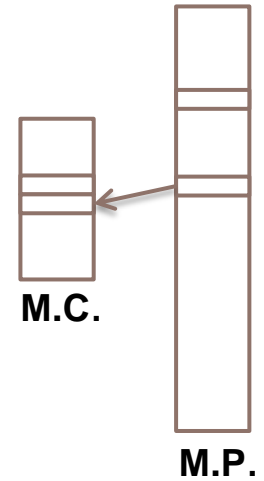
A double-headed arrow at the top indicates a width of 4 units.

# Correspondencia directa

## ► En general:

- En una memoria caché con  $NL$  número de líneas, el bloque de memoria  $K$  se almacena en la línea:

$$K \text{ módulo } NL$$





# Correspondencia directa

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

**Varios bloques en la misma línea**

Memoria principal	
	Palabra de 32 bits
0000000	
0000100	
0001000	
0001100	
0010000	
0010100	
0011000	
0011100	
0100000	
0100100	
0101000	
0101100	
0110000	
0110100	
0111000	
0111100	
1000000	
1000100	
1001000	
1001100	
1010000	
1010100	
1011000	
1011100	
1100000	
1100100	
1101000	
1101100	
1110000	
1110100	
1111000	
1111100	

# Correspondencia directa

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

¿Cómo se sabe **qué** bloque de memoria se encuentra una determinada línea?  
Ejemplo: la dirección 0100100

Memoria principal	
	Palabra de 32 bits
0000000	
0000100	
0001000	
0001100	
0010000	
0010100	
0011000	
0011100	
0100000	
0100100	
0101000	
0101100	
0110000	
0110100	
0111000	
0111100	
1000000	
1000100	
1001000	
1001100	
1010000	
1010100	
1011000	
1011100	
1100000	
1100100	
1101000	
1101100	
1110000	
1110100	
1111000	
1111100	

# Correspondencia directa

00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

¿Cómo se sabe **qué** bloque de memoria se encuentra una determinada línea?

Ejemplo: la dirección 0100100

**Se añade a cada línea una etiqueta**

Memoria principal	
	Palabra de 32 bits
0000000	
0000100	
0001000	
0001100	
0010000	
0010100	
0011000	
0011100	
0100000	
0100100	
0101000	
0101100	
0110000	
0110100	
0111000	
0111100	
1000000	
1000100	
1001000	
1001100	
1010000	
1010100	
1011000	
1011100	
1100000	
1100100	
1101000	
1101100	
1110000	
1110100	
1111000	
1111100	

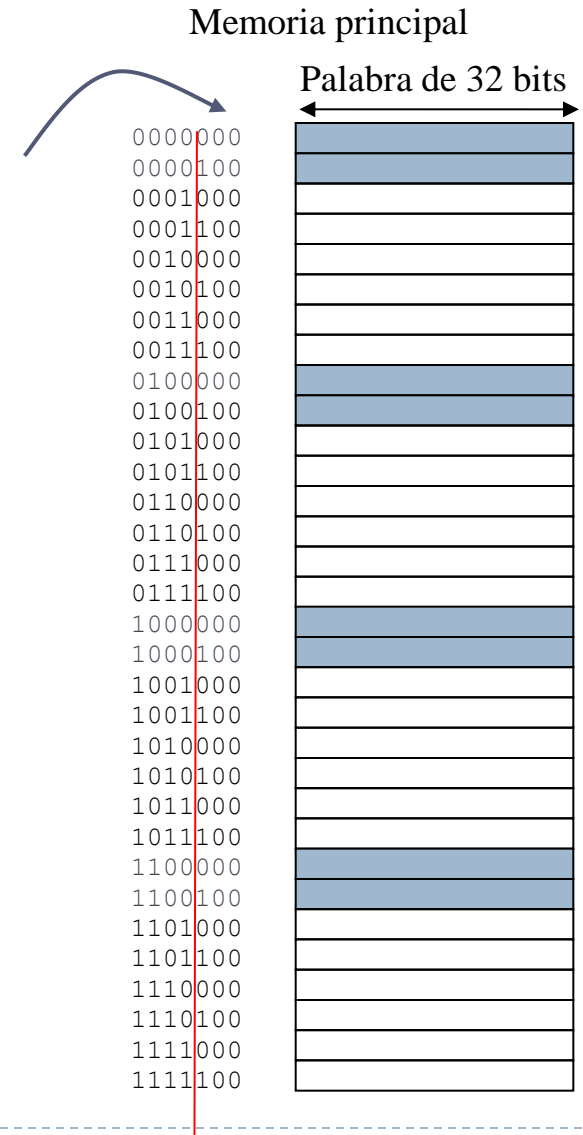
# Correspondencia directa

## Idea

¿qué byte dentro de la línea?  
Líneas de 8 bytes

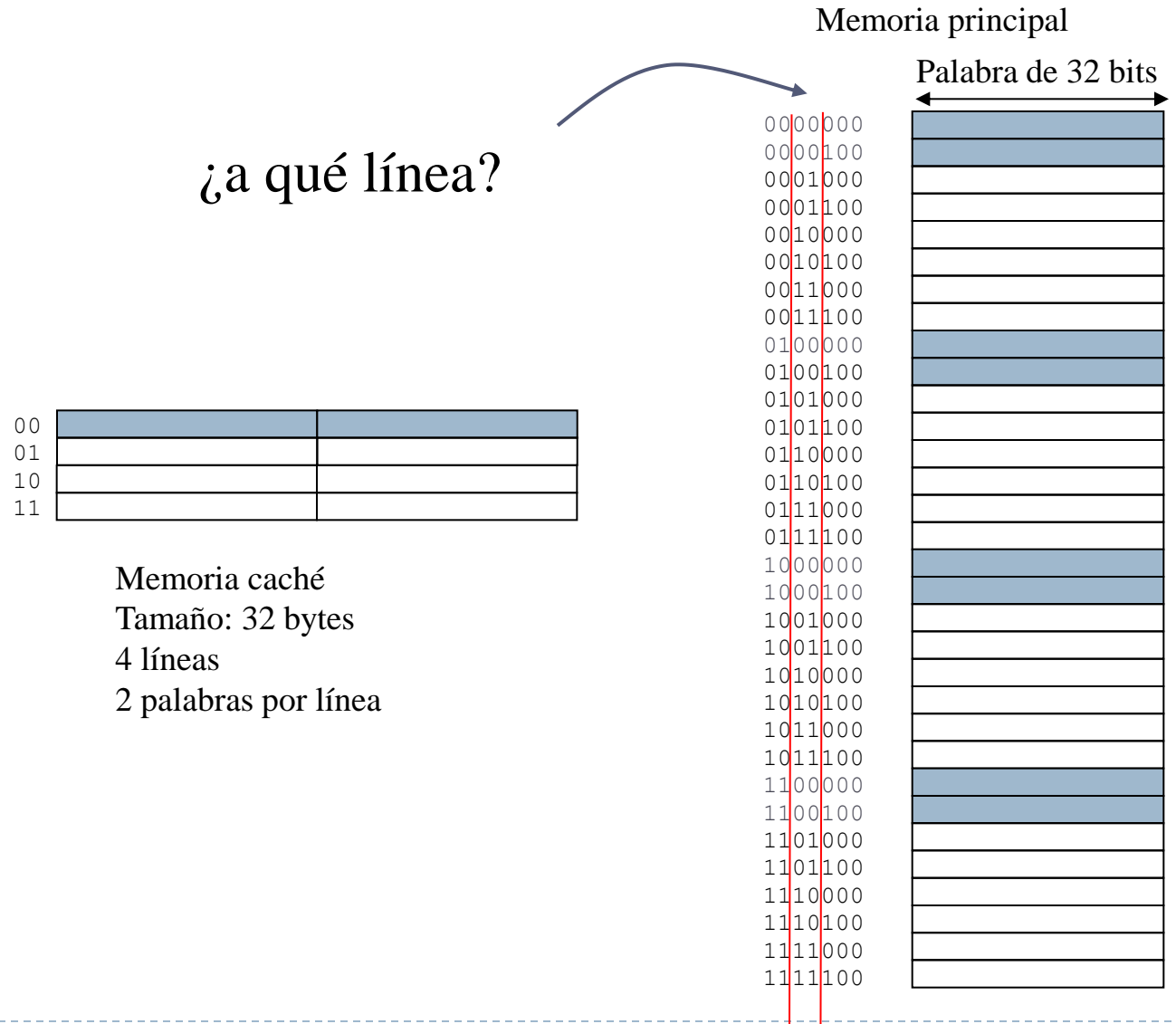
00		
01		
10		
11		

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea



# Correspondencia directa

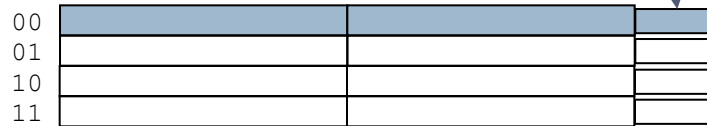
## Idea



# Correspondencia directa

## Idea

etiqueta asociada a la línea  
que diferencia los bloques  
que van a la misma línea

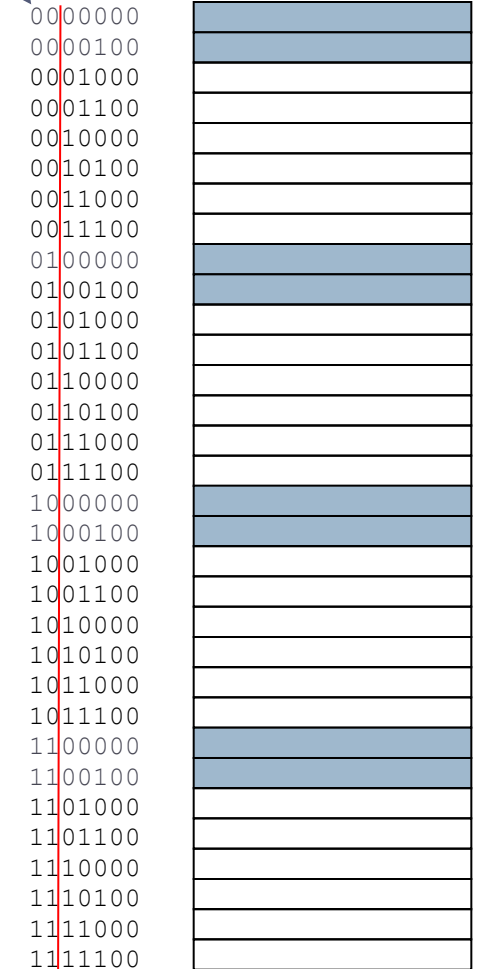


Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

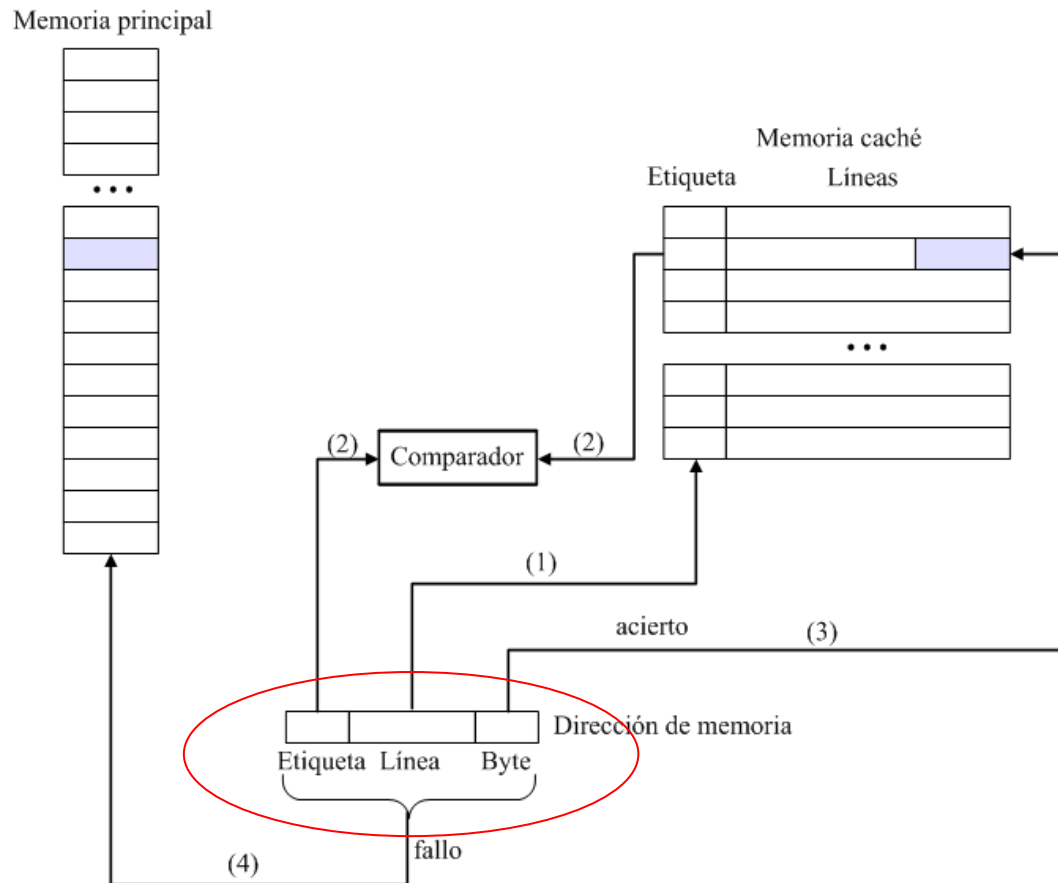
La etiqueta (bits superiores de la dirección) se almacena también en la memoria caché

## Memoria principal

Palabra de 32 bits



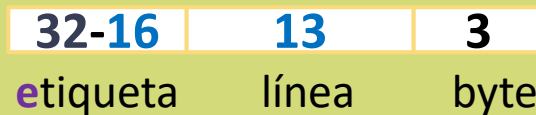
# Organización de una memoria caché con correspondencia directa



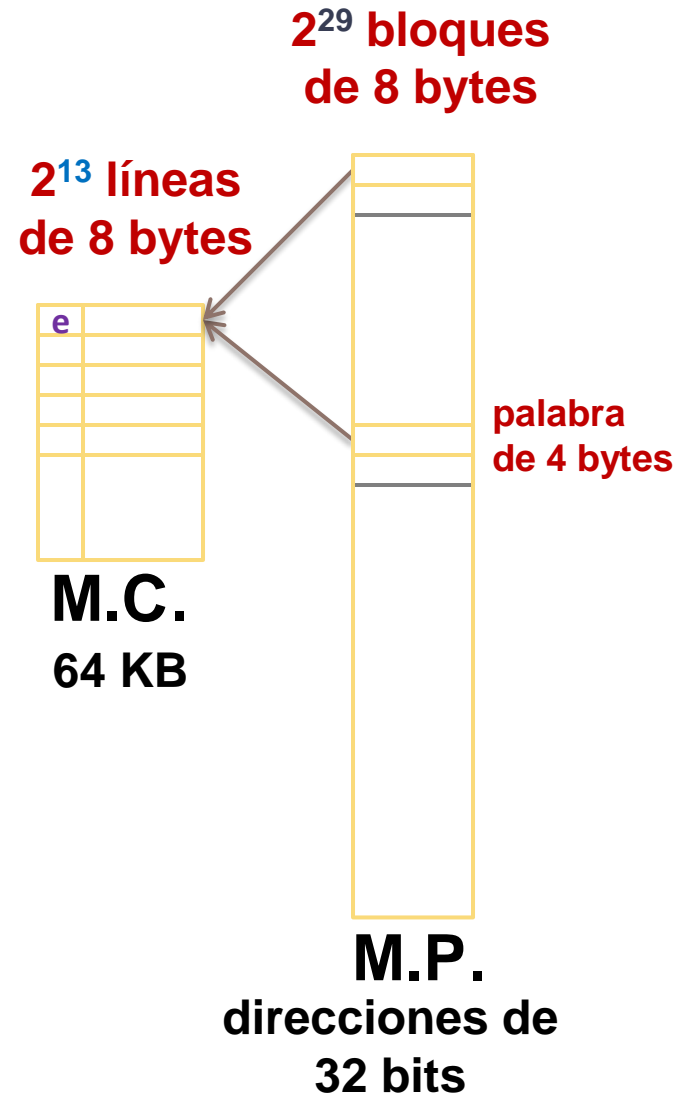
# Función de correspondencia directa

## Ejemplo

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



- ▶ Si en 'línea' está 'etiqueta', entonces está el bloque en caché
- ▶ Simple, poco costosa, pero puede provocar muchos fallos dependiendo del patrón de accesos



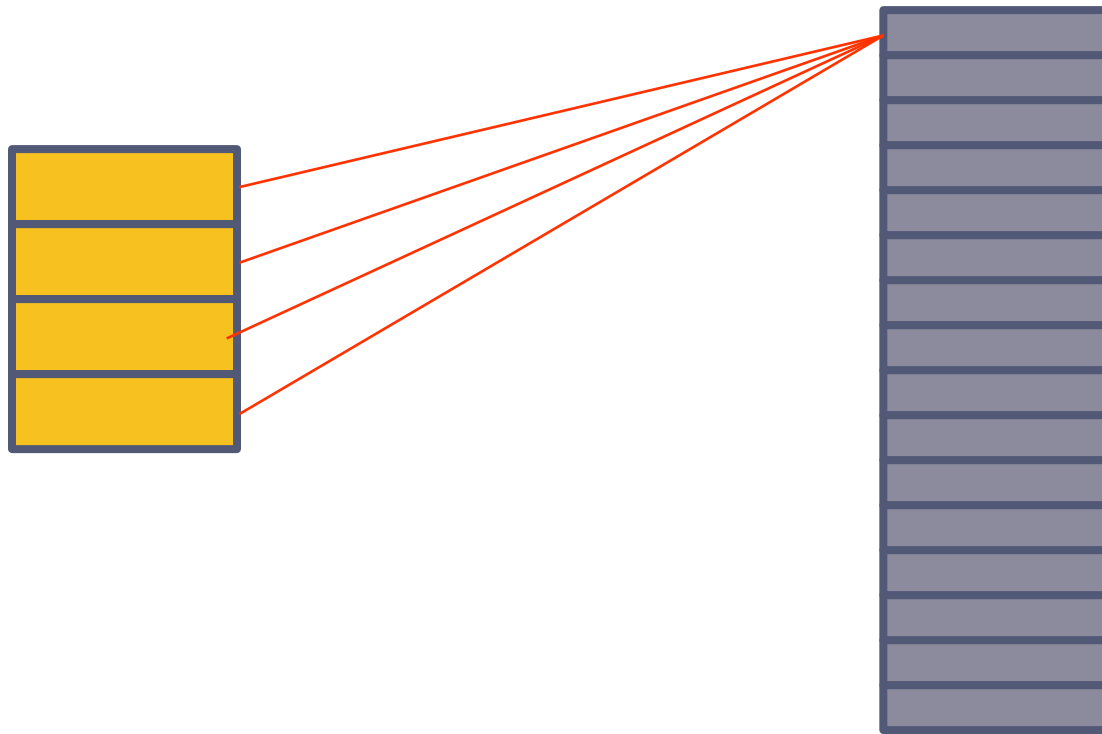


# Ejercicio

- ▶ Dado un computador de 32 bits con una memoria caché de 64 KB y bloques de 32 bytes. Si se utiliza correspondencia directa
  - ▶ ¿En qué línea de la memoria caché se almacena la palabra de la dirección 0x0000408A?
  - ▶ ¿Cómo se puede obtener rápidamente?
  - ▶ ¿En qué línea de la memoria caché se almacena la palabra de la dirección 0x1000408A?
  - ▶ ¿Cómo sabe la caché si la palabra almacenada en esa línea corresponde a la palabra de la dirección 0x0000408A o a la palabra de la dirección 0x1000408A?

# Correspondencia **asociativa**

- ▶ Cada bloque de M.P. puede almacenarse en cualquier línea de la caché



# Correspondencia asociativa

## Idea

etiqueta asociada a la línea  
que diferencia los bloques  
que van a la misma línea

00			
01			
10			
11			

Memoria caché  
Tamaño: 32 bytes  
4 líneas  
2 palabras por línea

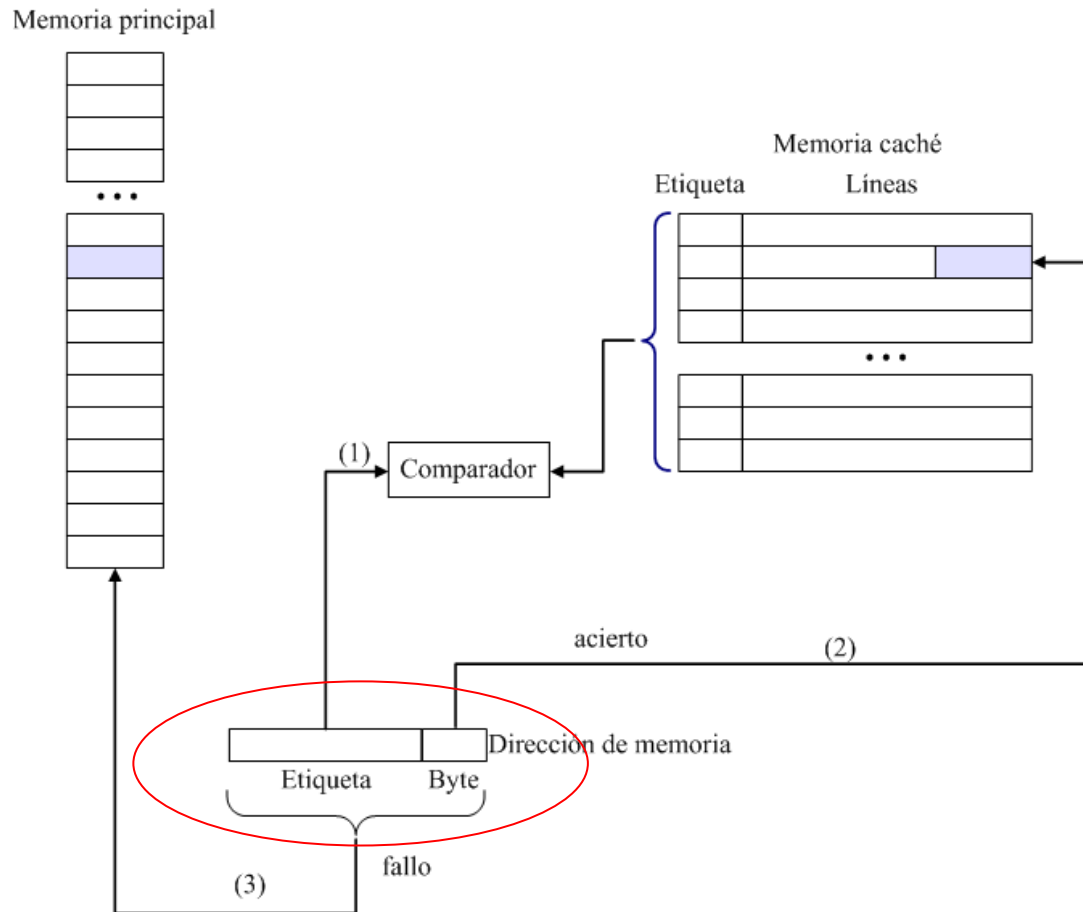
La etiqueta (bits superiores de la dirección)  
se almacena también en la memoria caché

Memoria principal

Palabra de 32 bits

0000	000	
0000	100	
0001	000	
0001	100	
0010	000	
0010	100	
0011	000	
0011	100	
0100	000	
0100	100	
0101	000	
0101	100	
0110	000	
0110	100	
0111	000	
0111	100	
1000	000	
1000	100	
1001	000	
1001	100	
1010	000	
1010	100	
1011	000	
1011	100	
1100	000	
1100	100	
1101	000	
1101	100	
1110	000	
1110	100	
1111	000	
1111	100	

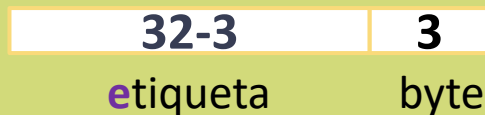
# Organización de una memoria caché con correspondencia asociativa



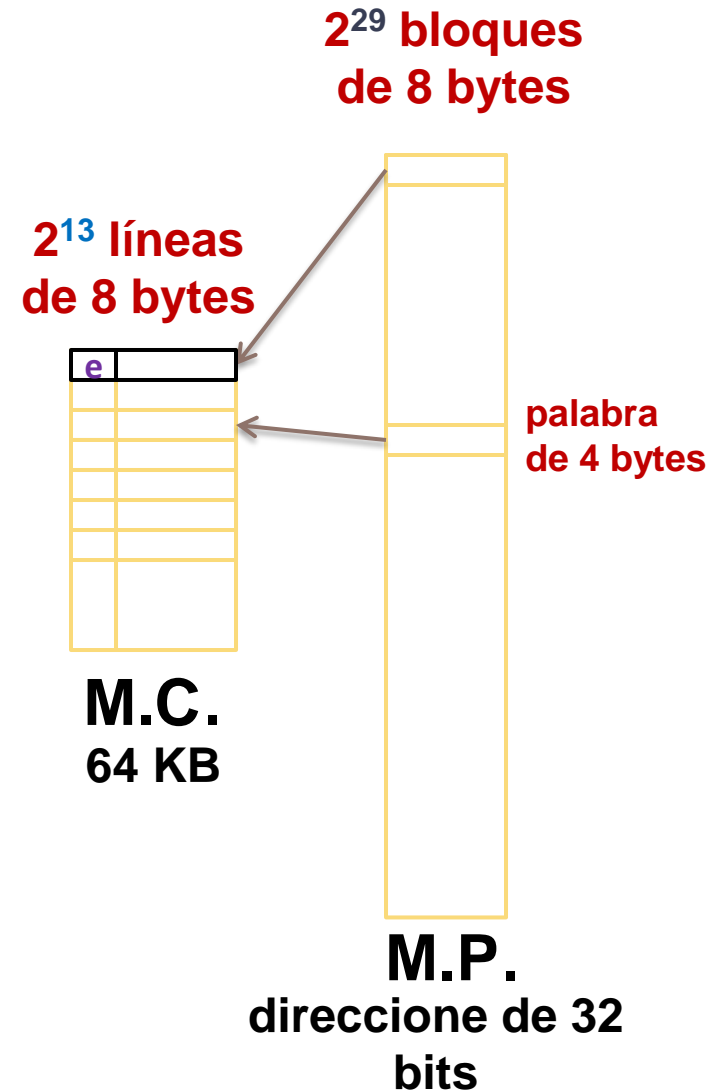
# Función de correspondencia asociativa

## Ejemplo

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



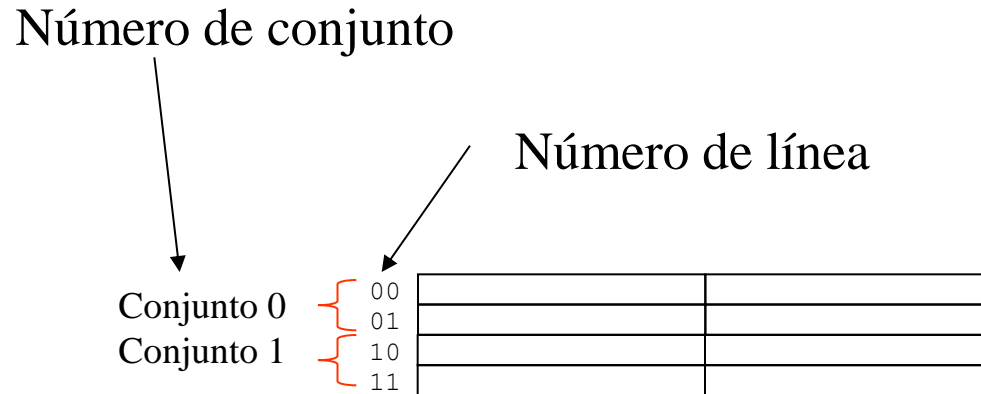
- ▶ Si hay una línea con 'etiqueta' en la caché, está allí el bloque
- ▶ Independiente del patrón de acceso, búsqueda costosa
- ▶ Etiquetas más grandes: cachés más grandes



# Correspondencia **asociativa por conjuntos**

- ▶ La memoria se organiza en **conjuntos** de líneas
  - ▶ Las líneas en un conjunto se le denominan vías.
- ▶ Una memoria **caché asociativa por conjunto de K vías**:
  - ▶ Cada conjunto almacena K líneas
- ▶ Cada bloque siempre se almacena en el mismo conjunto...
  - ▶ El bloque B se almacena en el conjunto:
    - ▶  $B \bmod \text{número de conjuntos}$
- ▶ ...dentro de un conjunto el bloque se puede almacenar en cualquiera de las líneas de ese conjunto

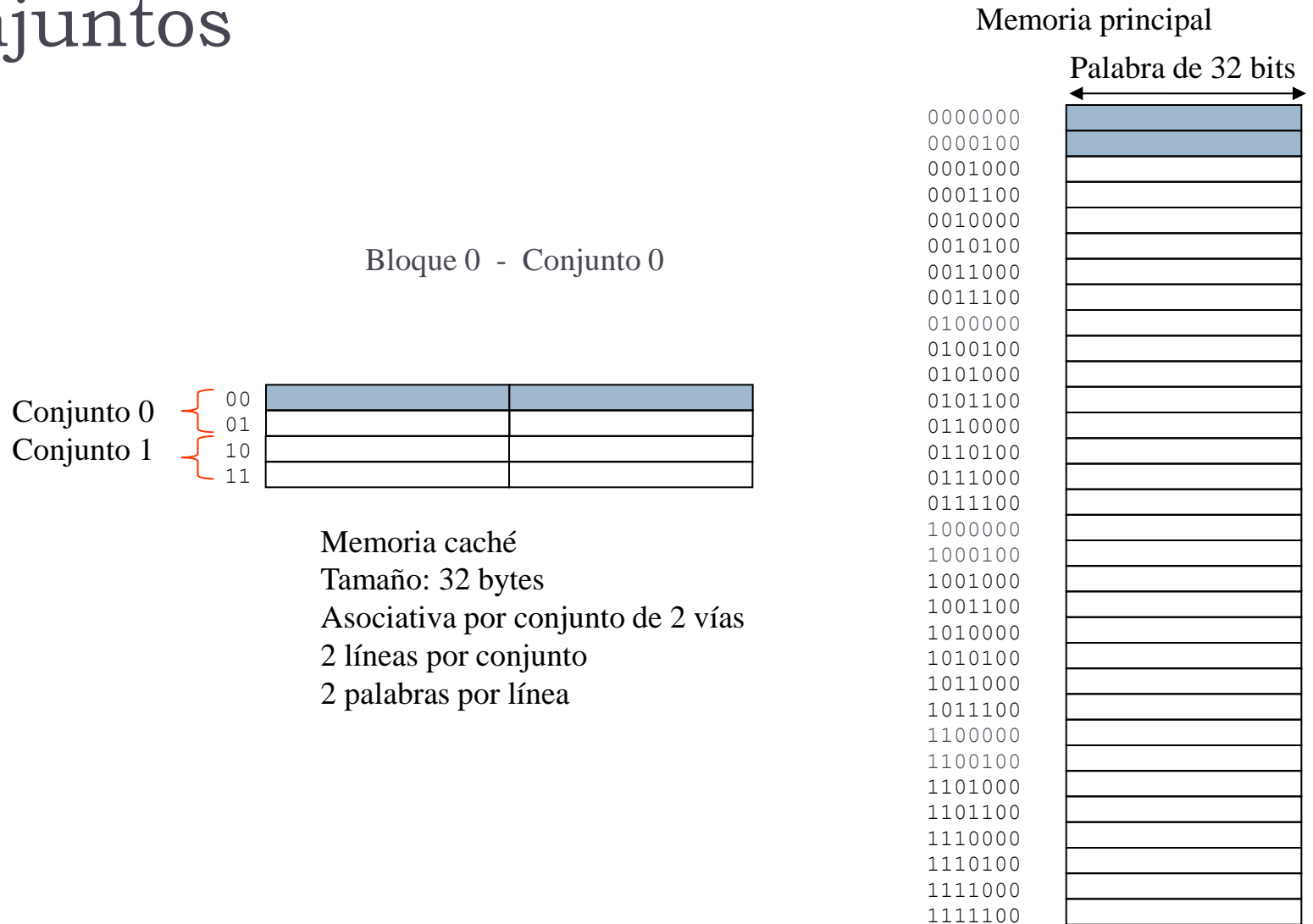
# Correspondencia asociativa por conjuntos



Memoria caché  
Tamaño: 32 bytes  
Asociativa por conjunto de 2 vías  
2 líneas por conjunto  
2 palabras por línea

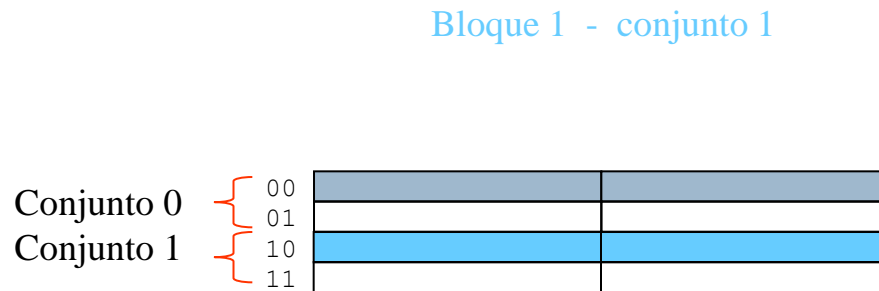
Memoria principal	
	Palabra de 32 bits
00000000	
00001000	
00010000	
00011000	
00100000	
00101000	
00110000	
00111000	
01000000	
01001000	
01010000	
01011000	
01100000	
01101000	
01110000	
01111000	
10000000	
10001000	
10010000	
10011000	
10100000	
10101000	
10110000	
10111000	
11000000	
11001000	
11010000	
11011000	
11100000	
11101000	
11110000	
11111000	

# Correspondencia asociativa por conjuntos

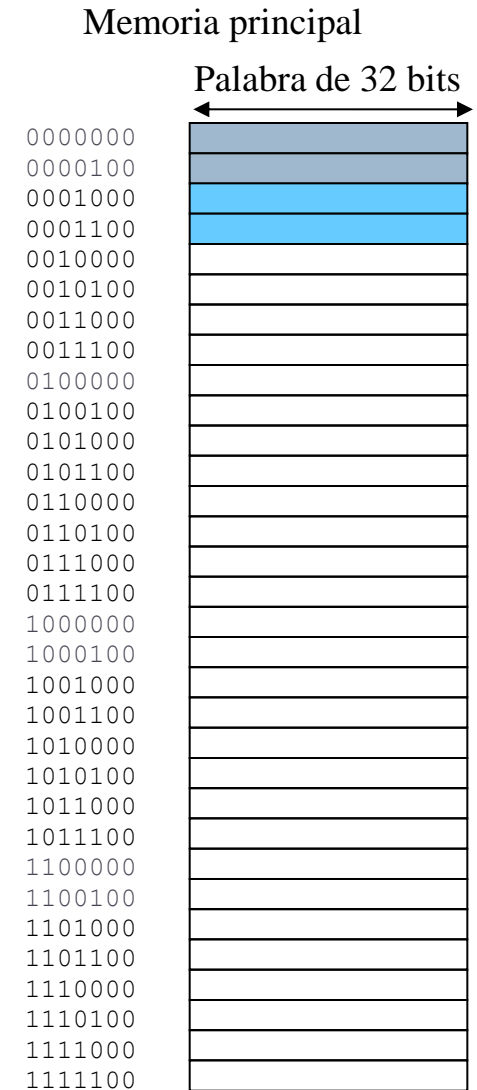




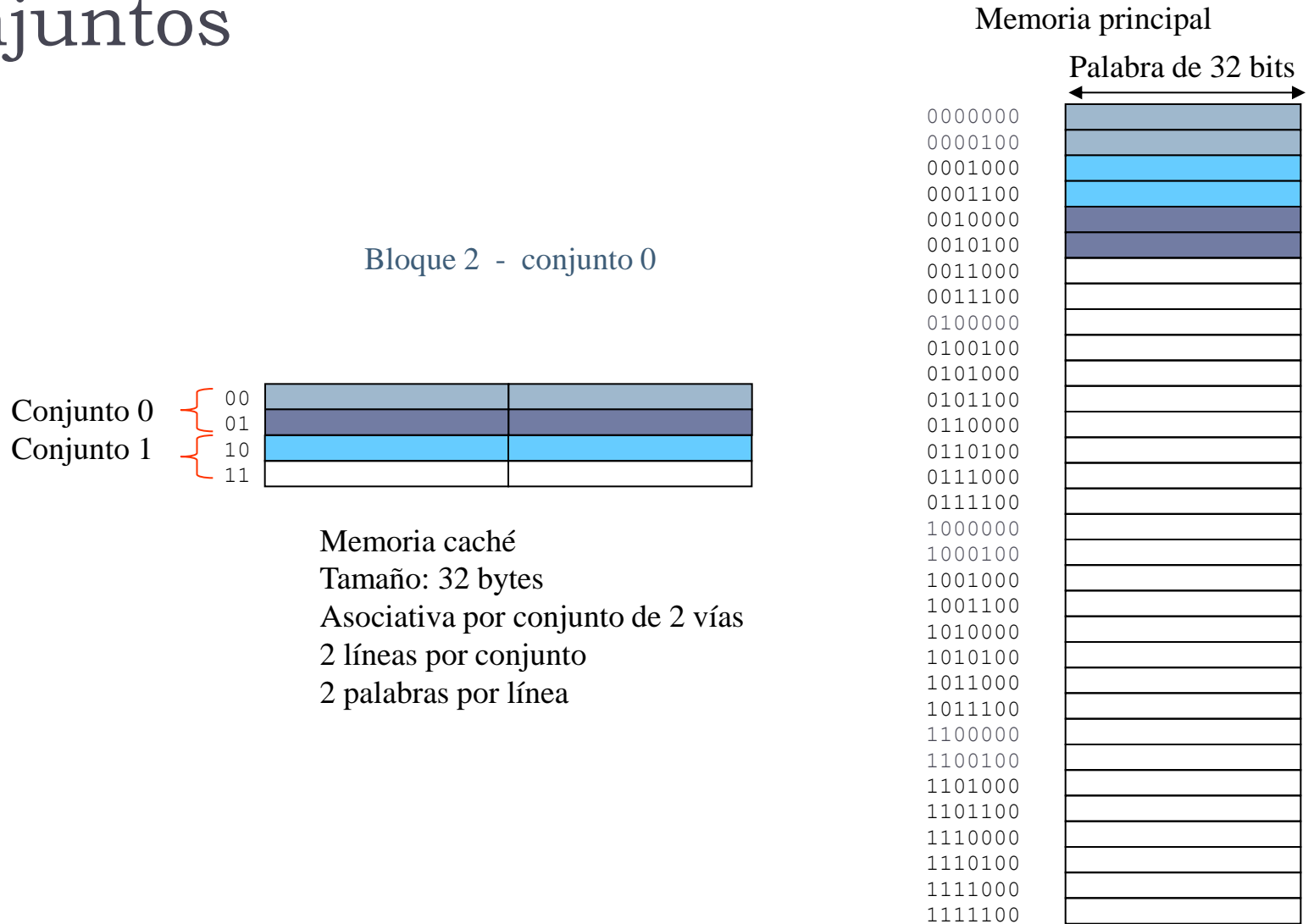
# Correspondencia asociativa por conjuntos



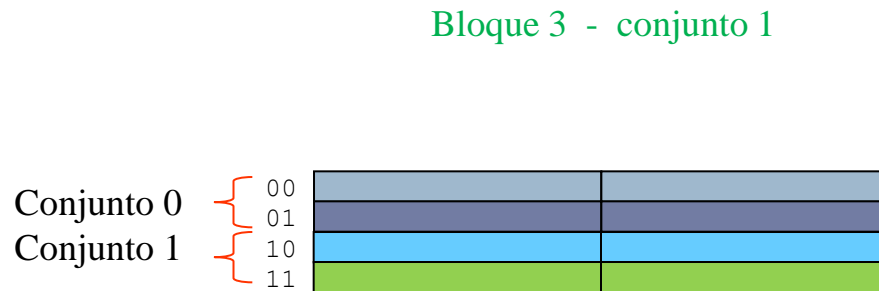
Memoria caché  
 Tamaño: 32 bytes  
 Asociativa por conjunto de 2 vías  
 2 líneas por conjunto  
 2 palabras por línea



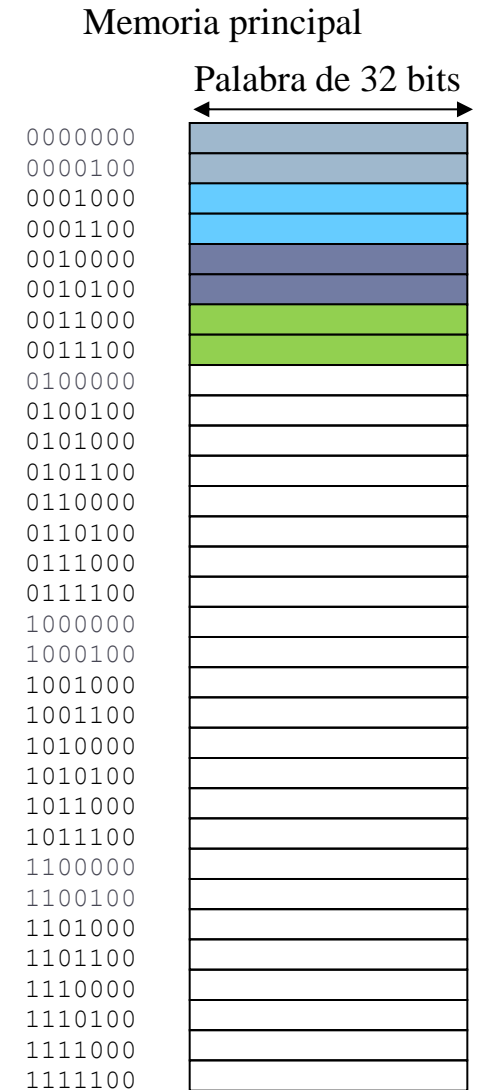
# Correspondencia asociativa por conjuntos



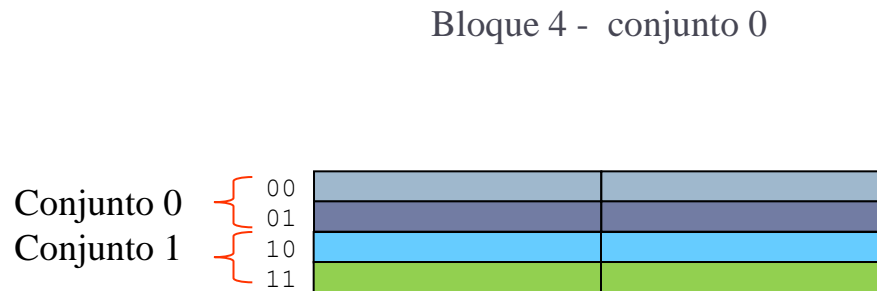
# Correspondencia asociativa por conjuntos



Memoria caché  
Tamaño: 32 bytes  
Asociativa por conjunto de 2 vías  
2 líneas por conjunto  
2 palabras por línea

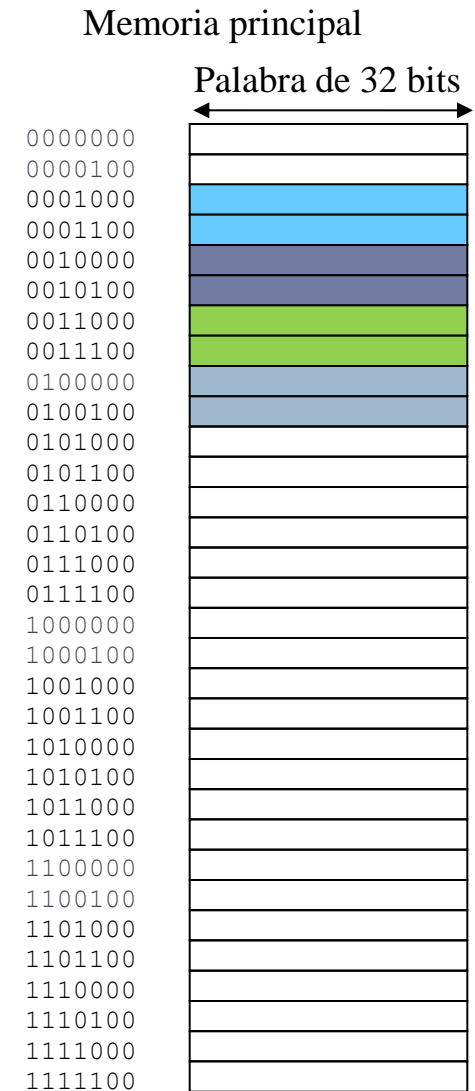


# Correspondencia asociativa por conjuntos



Memoria caché  
Tamaño: 32 bytes  
Asociativa por conjunto de 2 vías  
2 líneas por conjunto  
2 palabras por línea

Habría que eliminar la línea que estaba antes

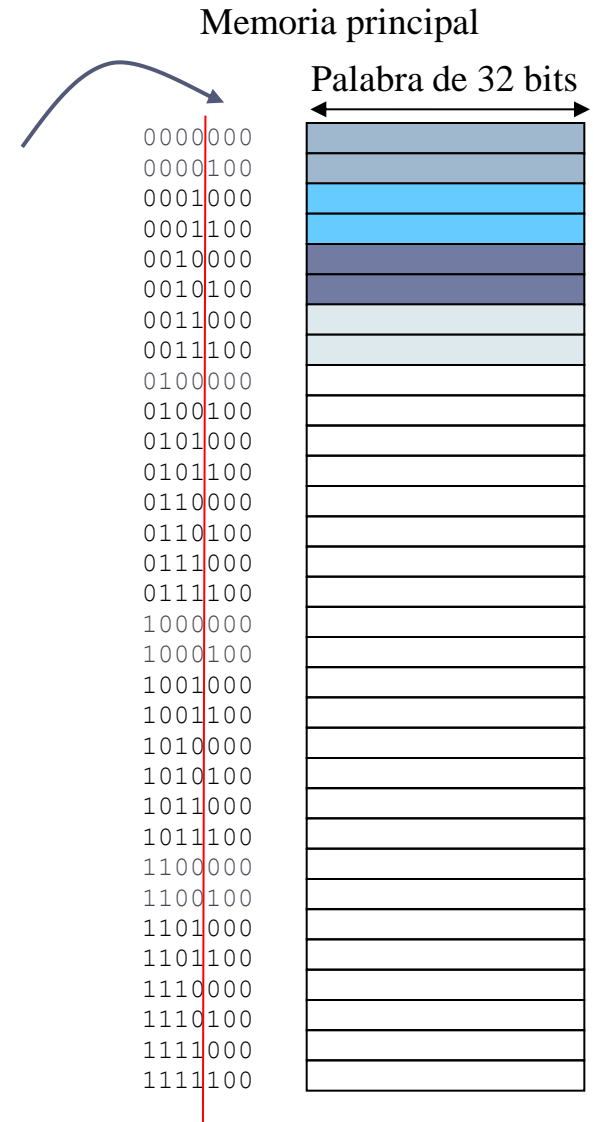


# Correspondencia asociativa por conjuntos

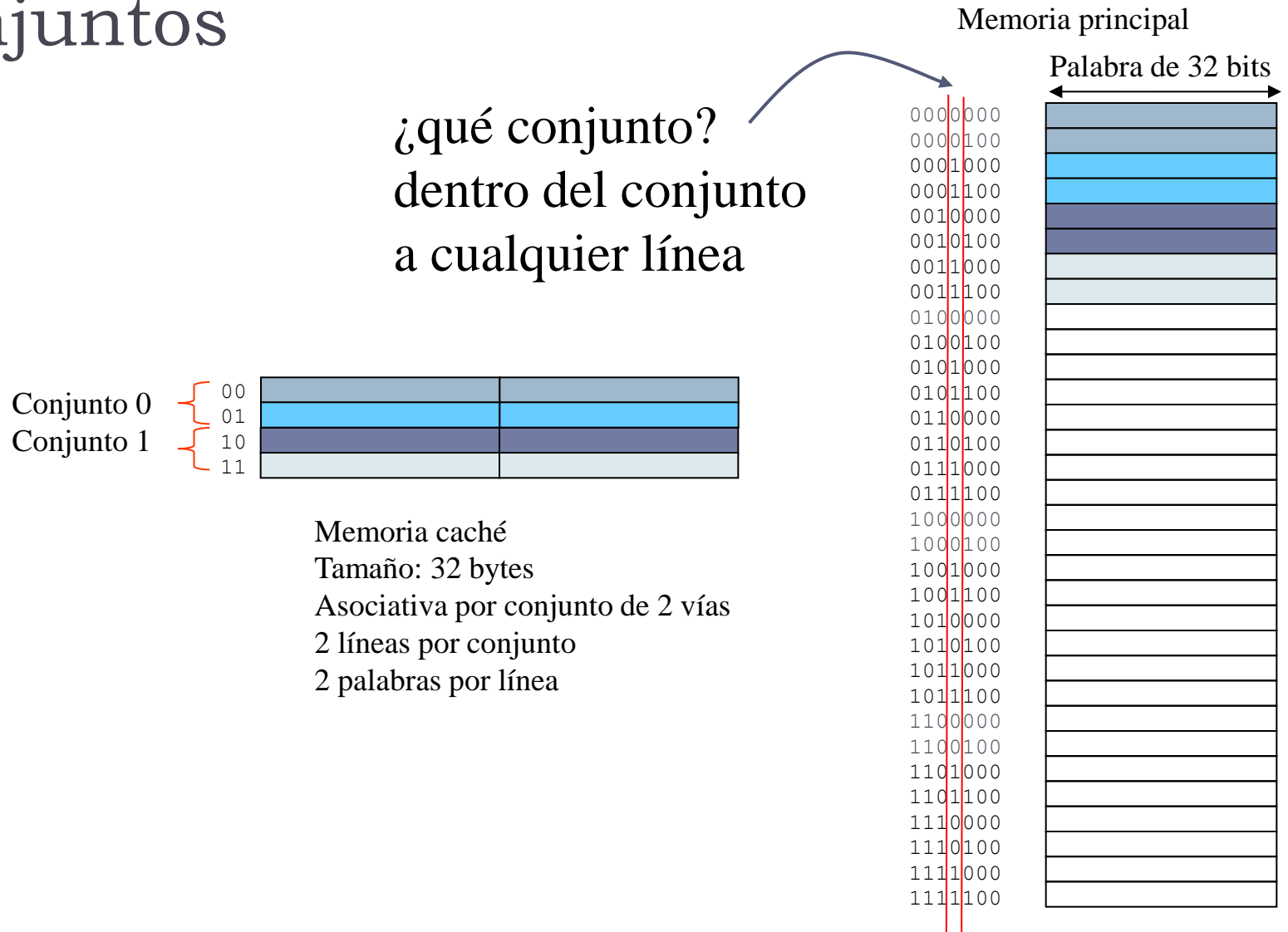
¿qué byte dentro de la línea?  
Líneas de 8 bytes



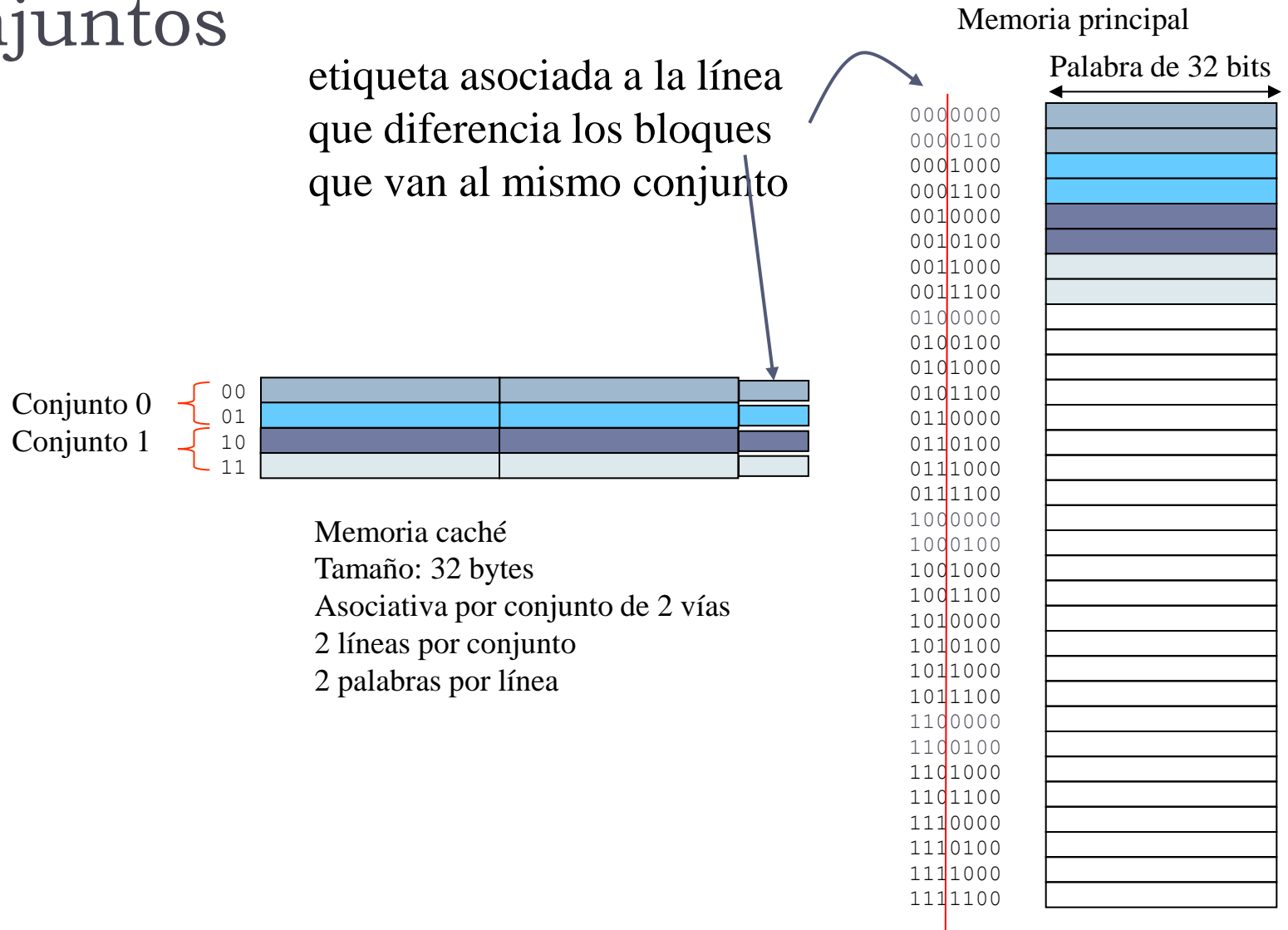
Memoria caché  
Tamaño: 32 bytes  
Asociativa por conjunto de 2 vías  
2 líneas por conjunto  
2 palabras por línea



# Correspondencia asociativa por conjuntos



# Correspondencia asociativa por conjuntos

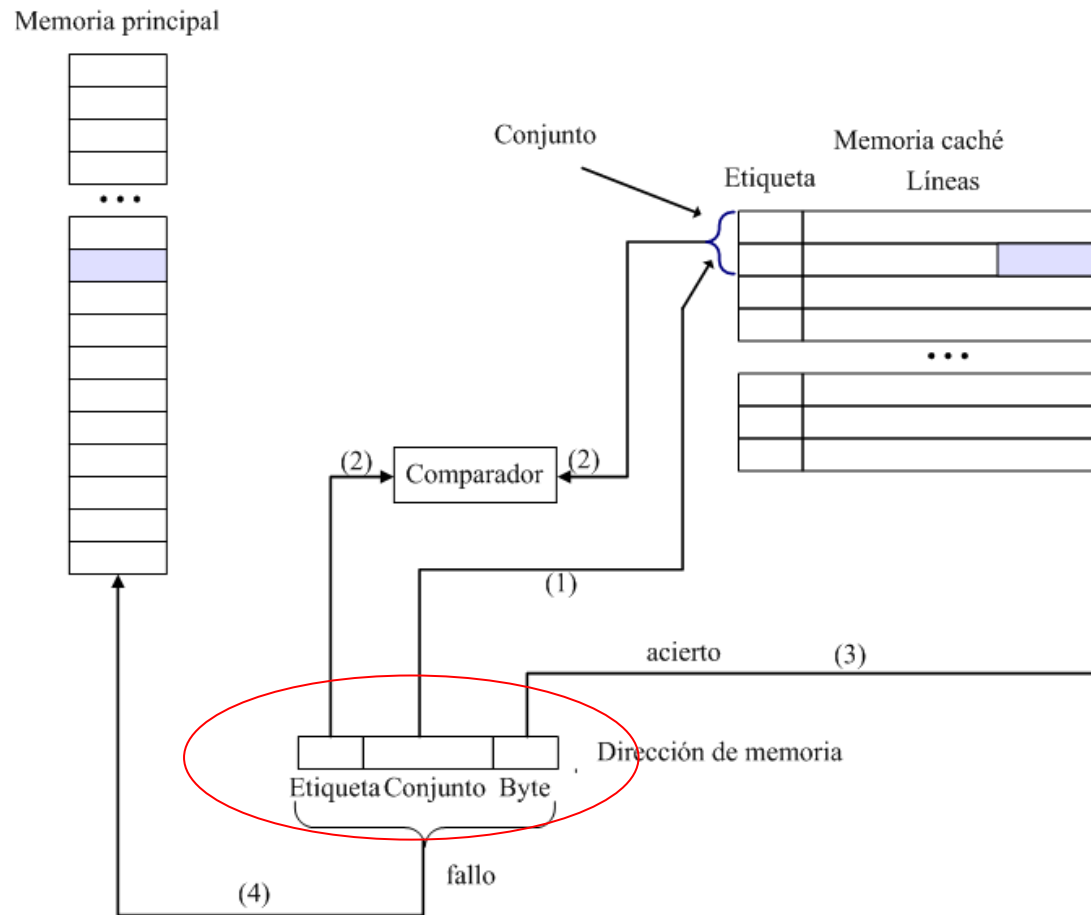


# Correspondencia asociativa por conjuntos

- ▶ Establece un compromiso entre flexibilidad y coste.
  - ▶ Es más flexible que la correspondencia directa.
  - ▶ Es menos costosa que la correspondencia asociativa.



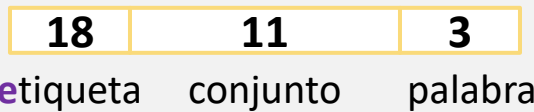
# Organización de una memoria caché asociativa por conjuntos



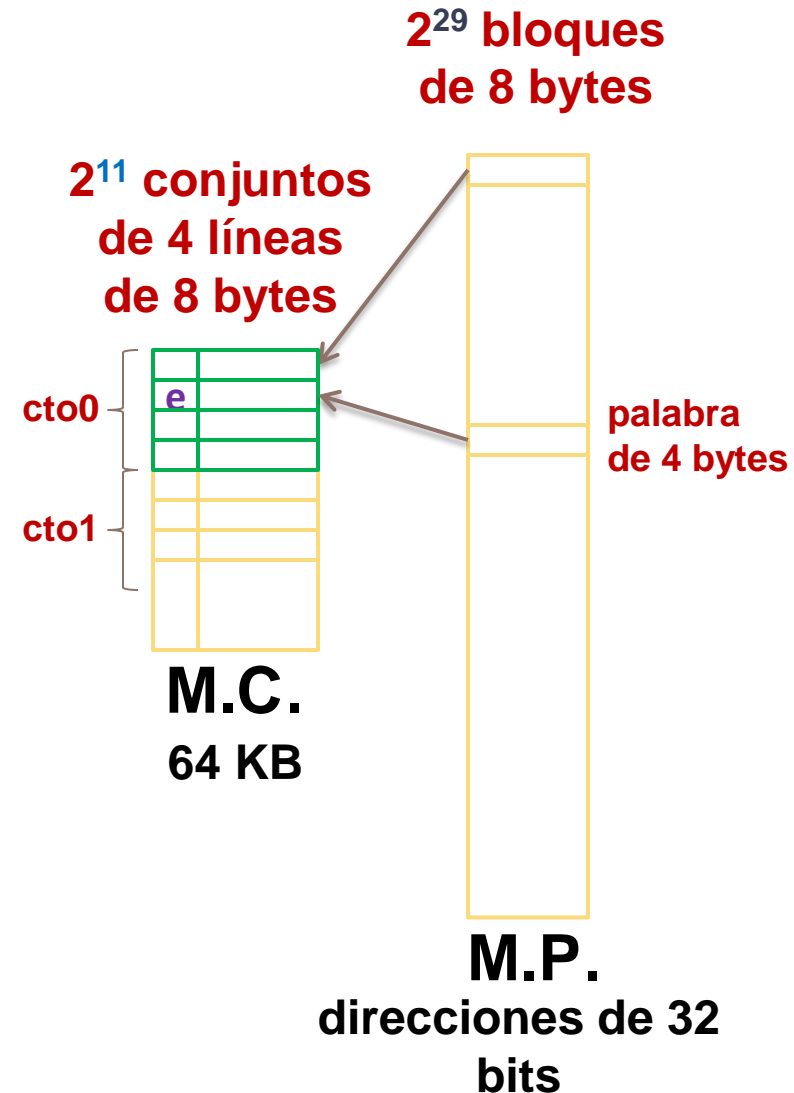
# Función de correspondencia asociativa por conjuntos. Ejemplo

## ► 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



# Sustitución de bloques

- ▶ Cuando todas las entradas de la caché contienen bloques de memoria principal:
  - ▶ Hace falta seleccionar una línea que hay que dejar libre para traer un bloque de la M.P.
    - ▶ Directa: no hay posible elección
    - ▶ Asociativa: seleccionar una línea de la caché.
    - ▶ Asociativa por conjuntos: seleccionar una línea del conjunto seleccionado.
- ▶ Existen diversos algoritmos para seleccionar la línea de la caché que hay que liberar (para asociativa y asociativa por conjuntos)

# Algoritmos de sustitución

- ▶ **FIFO**

- ▶ *First-In-First-Out*
- ▶ Sustituye la línea que lleva más tiempo en la caché.

- ▶ **LRU:**

- ▶ *Least Recently Used*
- ▶ Sustituye la línea que lleva más tiempo sin usarse.

- ▶ **LFU:**

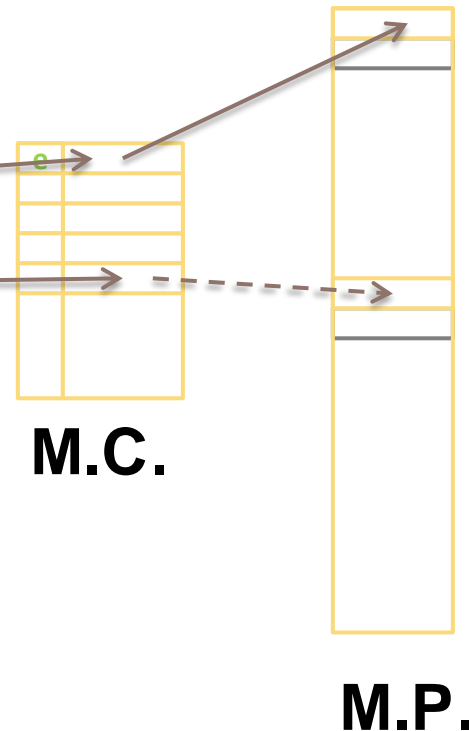
- ▶ *Least Frequently Used*
- ▶ Sustituye la línea que se ha usado menos veces.

# Políticas de escritura

- ▶ Cuando se modifica un dato en memoria caché, **en caso de acierto**, hay que actualizar en algún momento la memoria principal

- ▶ Alternativas:

- ▶ Escritura inmediata (*write through*).
- ▶ Post-escritura (*write back*).



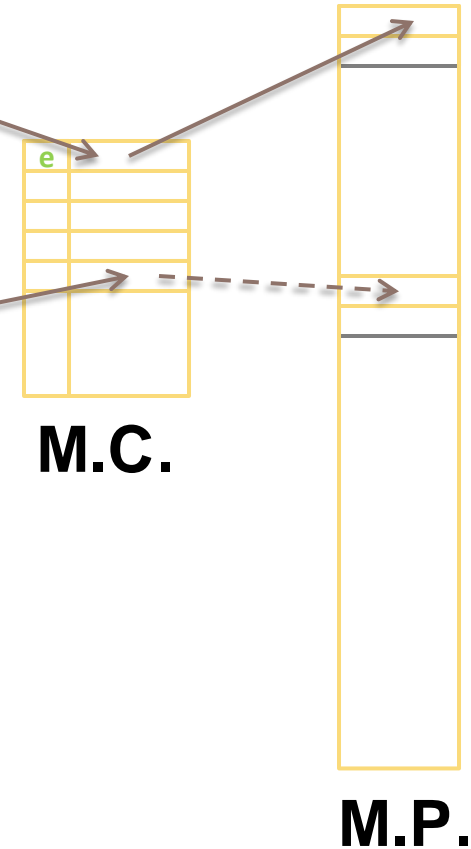
# Políticas de escritura

## ► Escritura **inmediata**:

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

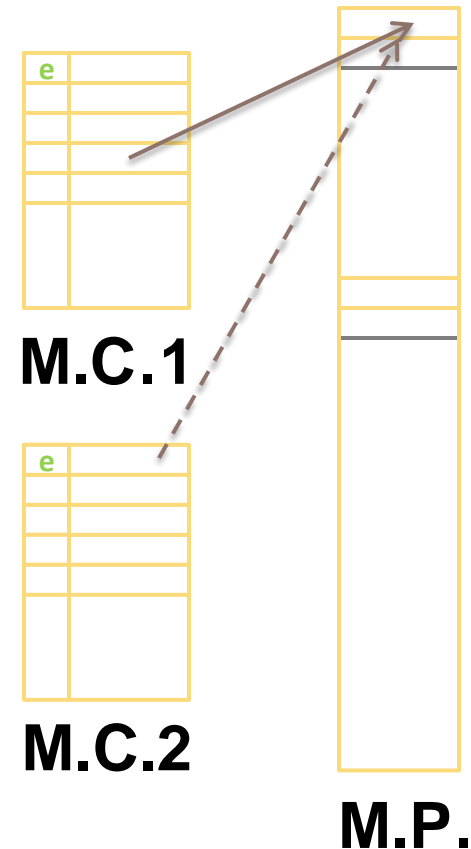
## ► Escritura **diferida**:

- 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



# Políticas 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%.

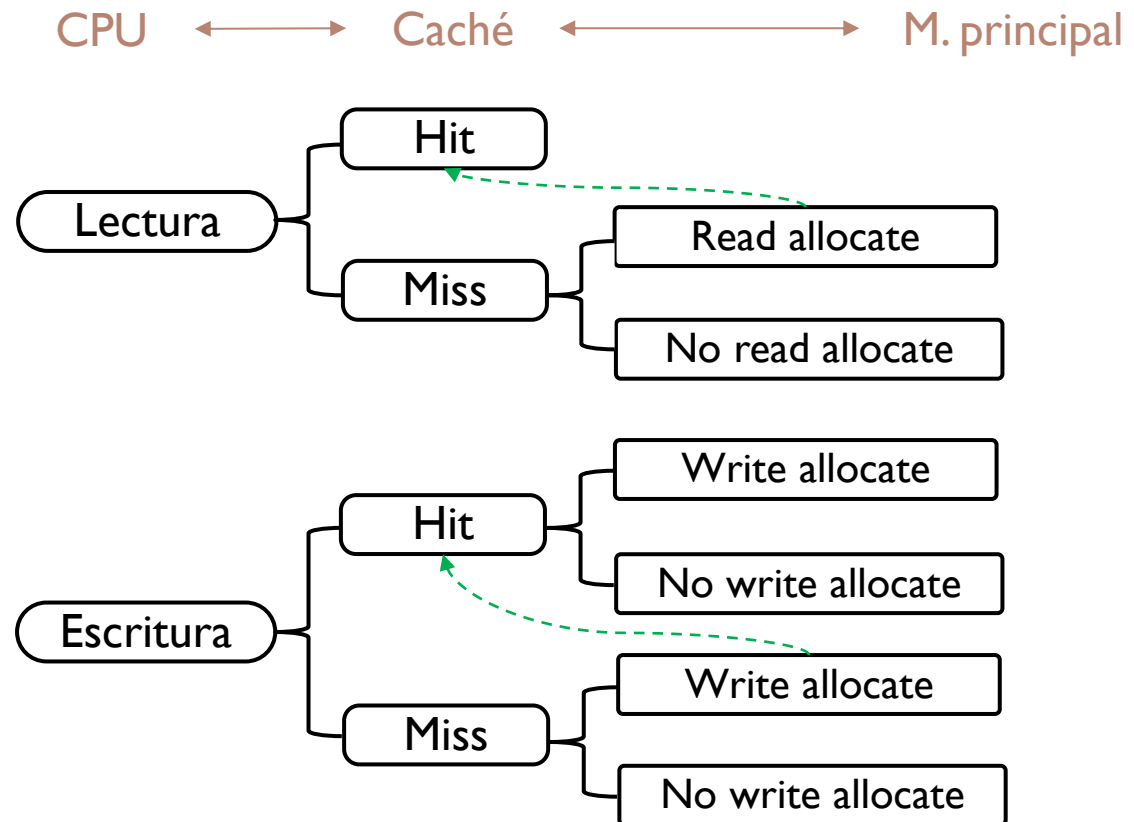


# Políticas de escritura

- ▶ Cuando se modifica un dato en memoria caché, en caso de **fallo**, hay que traer en algún momento el bloque de memoria principal
- ▶ Alternativas:
  - ▶ **Write-allocate**: traer bloque a caché y actualizar en caché.
    - ▶ Bueno si siguientes lecturas/escrituras al mismo bloque
  - ▶ **No-write-allocate**: escribir directamente en memoria
    - ▶ Bueno si siguientes operaciones no al mismo bloque



# Políticas de escritura



- ▶ Típico:
  - ▶ write-through + no-write-allocate
  - ▶ write-back + write-allocate

# Tipos de fallos en caché

- ▶ *Compulsory* (obligatorios)

- ▶ La primera vez que se accede a una palabra de una línea que no está
- ▶ Los fallos en una caché infinita

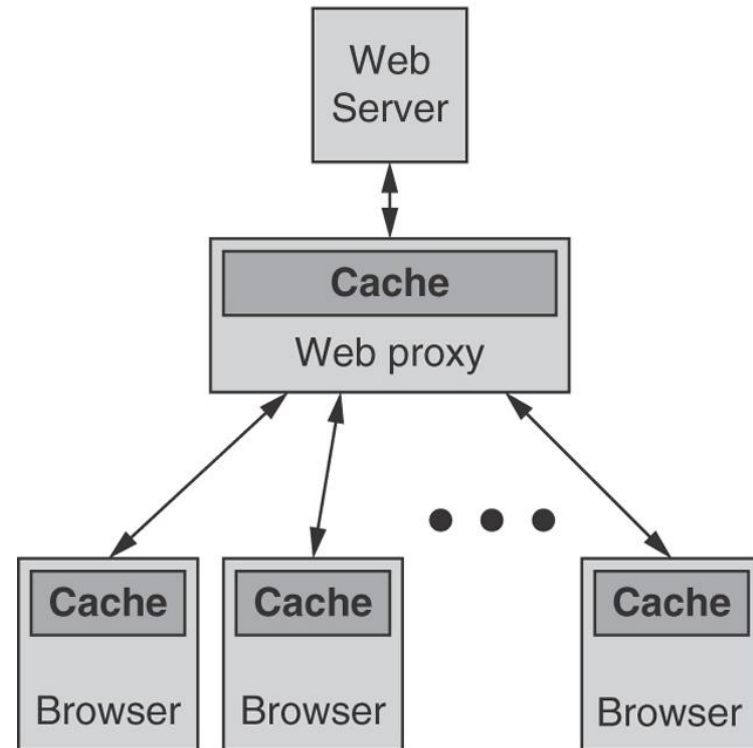
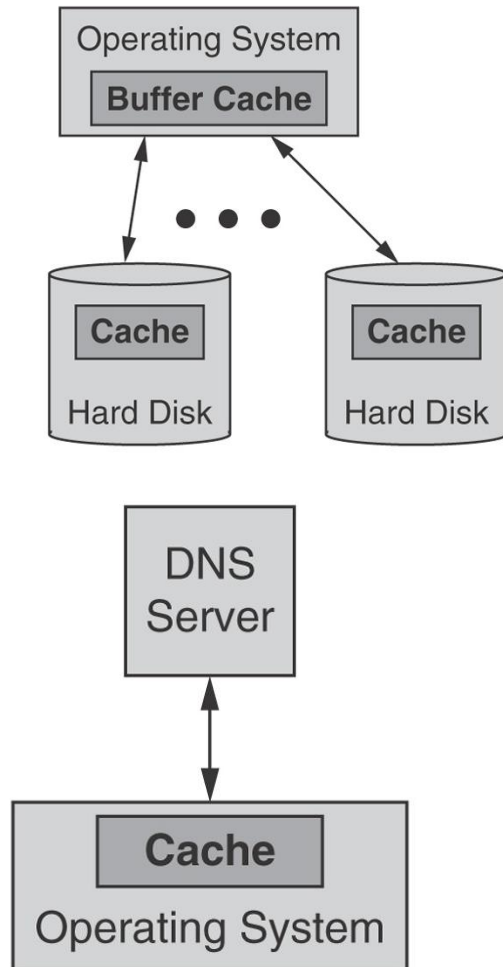
- ▶ *Capacity* (capacidad)

- ▶ Cuando se acceden a muchas otras palabras antes de volver a usar una palabra inicial.
- ▶ Los fallos en una caché totalmente asociativa

- ▶ *Conflict* (conflictos)

- ▶ Dos palabras van a la misma línea de caché
- ▶ Los fallos en correspondencia directa por accesos distintos a misma línea.

# Ejemplos de cachés en otros sistemas



Memory Systems  
Cache, DRAM, Disk  
Bruce Jacob, Spencer Ng, David Wang  
Elsevier

Grupo ARCOS

**uc3m** | Universidad **Carlos III** de Madrid

# Tema 5: Jerarquía de Memoria (II)

## **Estructura de Computadores**

Grado en Ingeniería Informática  
Grado en Matemática aplicada y Computación  
Doble Grado en Ingeniería Informática y Administración de Empresas



# Ejercicio

- ▶ 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

## ► Se pide:

- a) ¿Cuántos bloques tiene la memoria principal?
- b) ¿Cuántos conjuntos tiene la memoria caché?
- 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.
- 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?
- 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?

# Ejercicio (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

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

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

# Ejercicio (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:

Etiqueta (18 bits)	Conjunto (8 bits)	Byte (6 bits)
--------------------	-------------------	---------------



# Ejercicio (solución)

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

Etiqueta (18 bits)	Conjunto (8 bits)	Byte (6 bits)
--------------------	-------------------	---------------

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:

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

$$\triangleright 8 = 5 + (1-h) * 100$$

Despejando h, se tiene  $h = 97/100 = 0,97$  (tanto por uno)

Es decir, **una tasa de acierto del 97 %**

# Ejercicio

- ▶ Sea un computador con una memoria caché y principal con las siguientes características:
  - ▶ Tiempo de acceso a memoria caché de 4 ns
  - ▶ Tiempo de acceso a memoria principal de 80 ns
  - ▶ Tiempo para servir un fallo de caché de 120 ns
  - ▶ Política de escritura inmediata

En este computador se ha observado que la tasa de aciertos a la memoria caché es del 95 % y que cada 100 accesos, 90 son de lectura. Calcular el tiempo medio de acceso a memoria.