# Caches



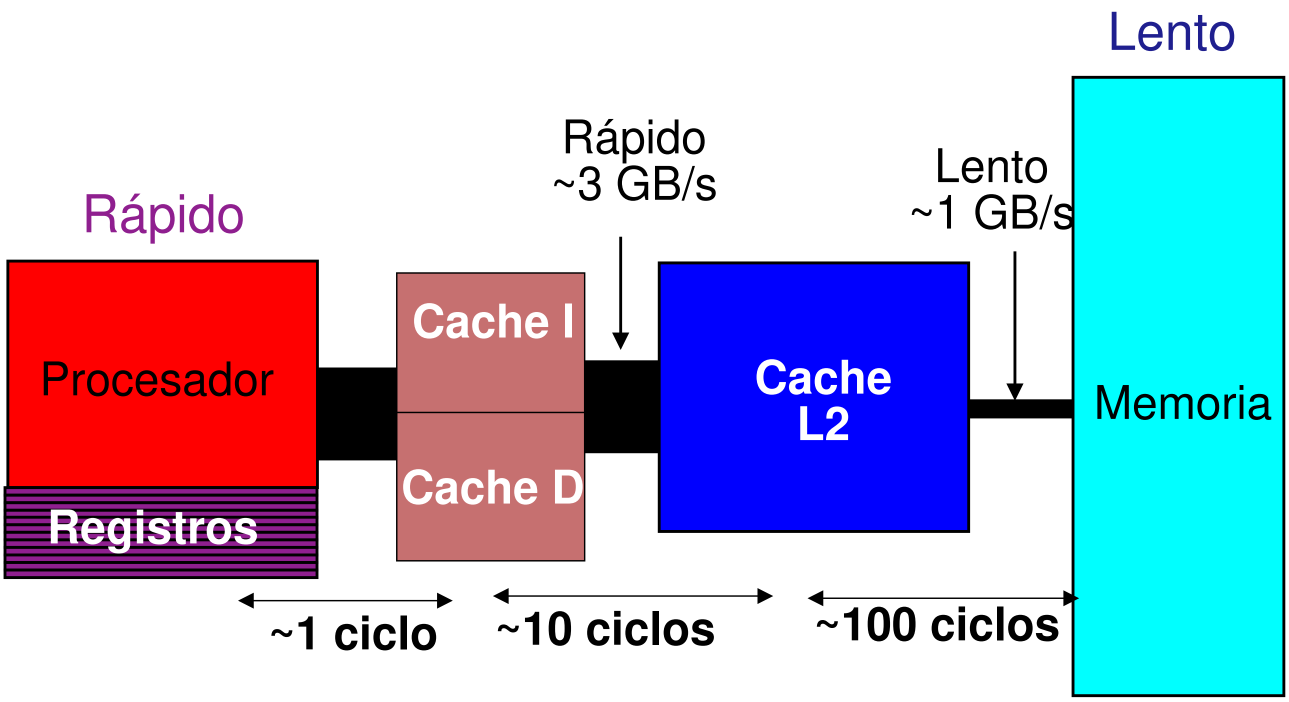


- Por qué de las caches
- Cache lines
- Tipos de cache coherency: protocolos
- estados de lineas de cache
- ejemplo
- cache directory
- cache replacement policies

  direct mapped: cache trashing
  fully associative
  set associative
- false sharing
- Transaction Lookaside Buffer (TLB)

3

## El por qué de las caches



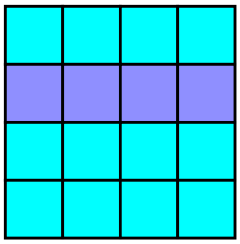

- El uso de caches minimiza los efectos perniciosos de acceso a memoria principal (gran latencia)



4

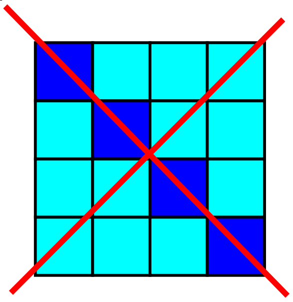
## Cache Lines

- La unidad de transferencia se llama **cache line**
- Una línea de cache consiste en posiciones <u>consecutivas</u>



Como esto

NO así



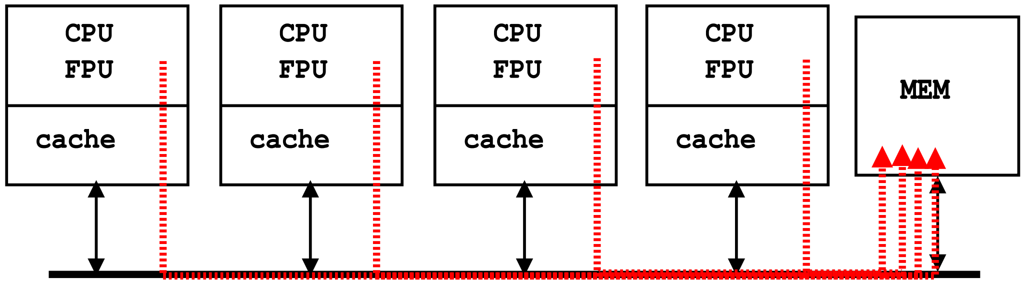
- **El tamaño** de una línea de cache es "architecture dependent"
- para máxima cache performance:

1. Re–usar los datos que hay en la cache
   Conocido como temporal locality
2. Usar todos los datos que hay en una línea de cache
   Conocido como spatial locality

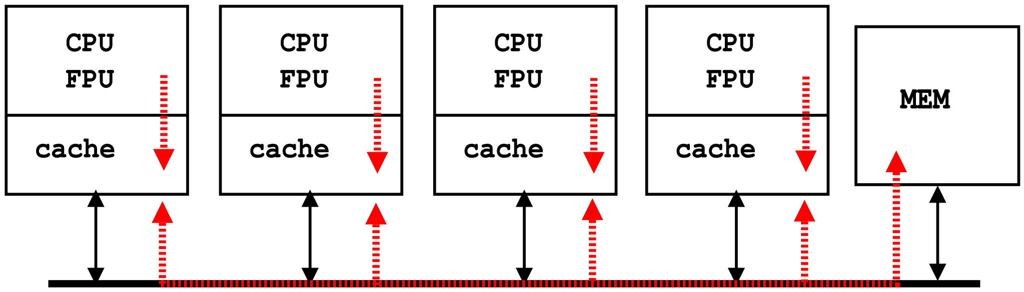
## Tipos de Multi–Processor Cache Coherency

5

**Write–Through: flush inmediato a memoria –> gran tráfico (bottleneck)**




**Write–Back: esperar hasta que se requiera**

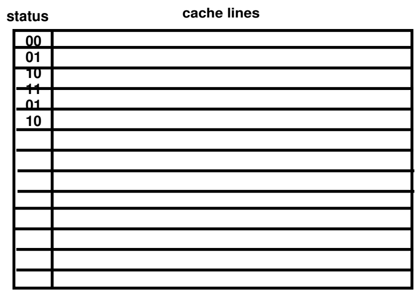





## Funcionamiento de la cache: estados

6

**En el protocolo write–back, una línea de cache puede tener 4 estados:**




***1. clean***

**Es la única copia en cache**

***2. shared***

**Hay múltiples copias en cache**

***3. dirty***

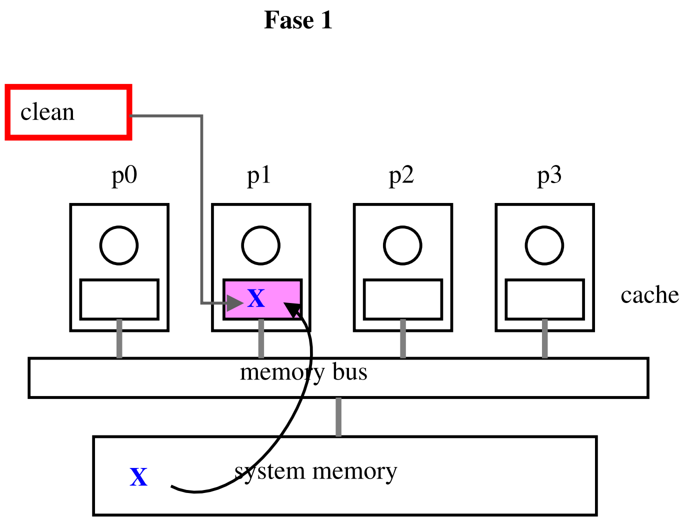
**La copia ha sido modificada y debe ser actualizada a memoria antes de ser machacada.**

***4. invalid***

**Ya no es válida. Alguna otra copia es más reciente.**

## Funcionamiento de la cache: estados

7

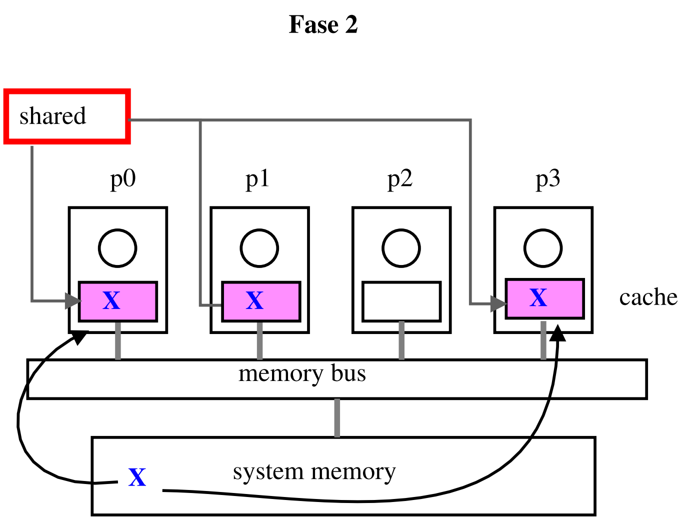



- El Procesador P1 ha cargado (LOAD) X en su cache
- Nadie más tiene una copia, por tanto el estado es: "clean"



## Funcionamiento de la cache: estados

8

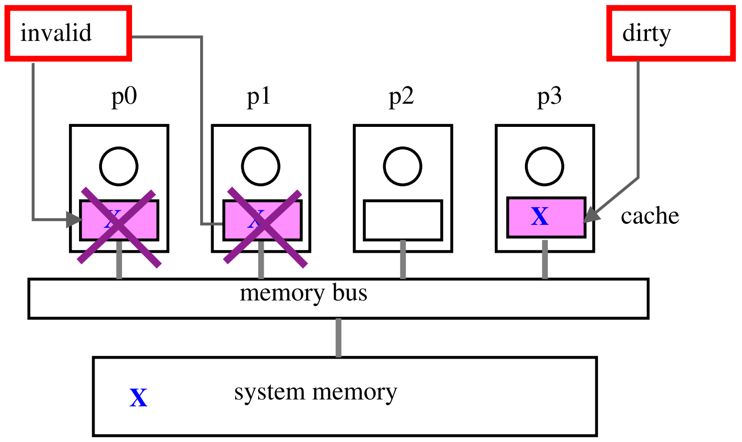



- También los procesadores P0 y P1 han cargado X en su cache
- La línea de cache que contiene X tiene ahora status "shared"

## Funcionamiento de la cache: estados

9

**Fase 3**



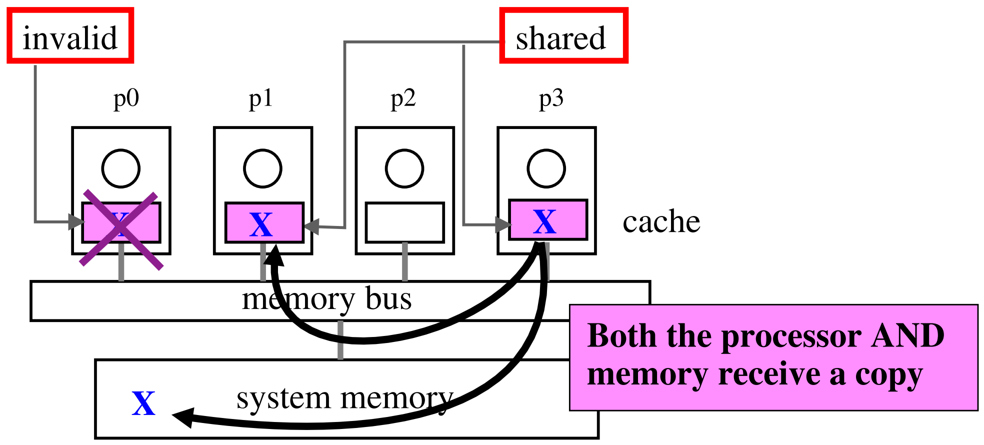

- Supongamos que ahora P3 modifica X
- La línea de cache en P3 queda ahora en estado "dirty"
- Pero antes las líneas de cache de P0 and P1 se marcan "invalid"



## Funcionamiento de la cache: estados

10

**Fase 4**




- Una línea de cache modificada ("dirty") no se escribirá a memoria (write back) a menos que:
  - Su espacio se requiere para otros datos
  - Otro procesador necesita el mismo dato (como P1 aqui)

## Cache coherency en S2MP:  directory map. HUB features

11

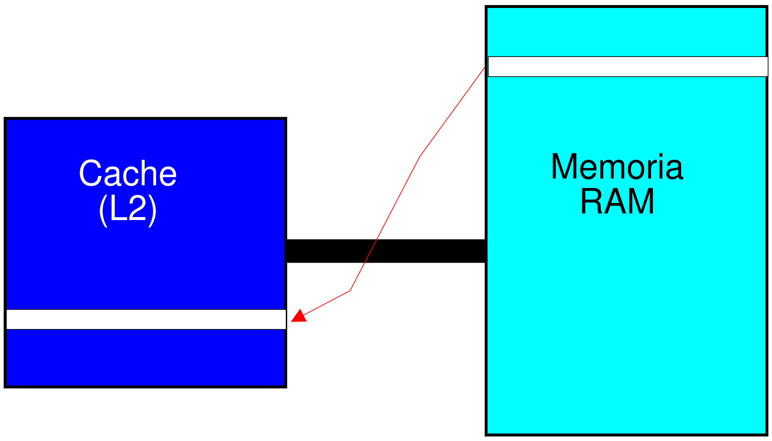
- Existe un directorio de quién tiene copias de cada línea de cache ( 1 bit por cada cpu y línea de cache –>  0=no_copia, 1=copia a esa cpu)
- En lugar de hacer un snoopy–bus (coste lineal con el N de cpu´s) se hace un broadcast sólo a las cpu´s con copia de la línea (bit a 1). Esto posibilita la escalabilidad de la arquitectura en memoria compartida.
- El HUB mantiene una estadística de accesos "locales" / "remotos" para decidir si migrar páginas a otros nodos.
- El HUB además tiene(*)  un mecanismo HW de copia de páginas que permite el page migration y el page replication  a gran velocidad.



Lento

## Cache replacement policies

12




**Problema:** La cache no es suficiente para todos los datos

**Pregunta:** Cómo reemplazar los datos en la cache ?

**Respuesta:** según una cache replacement policy

13

### 1. Direct Mapped

**Formula de reemplazo:**

**cache location = memory address modulo cache size**
(the memory address can be virtual or physical)

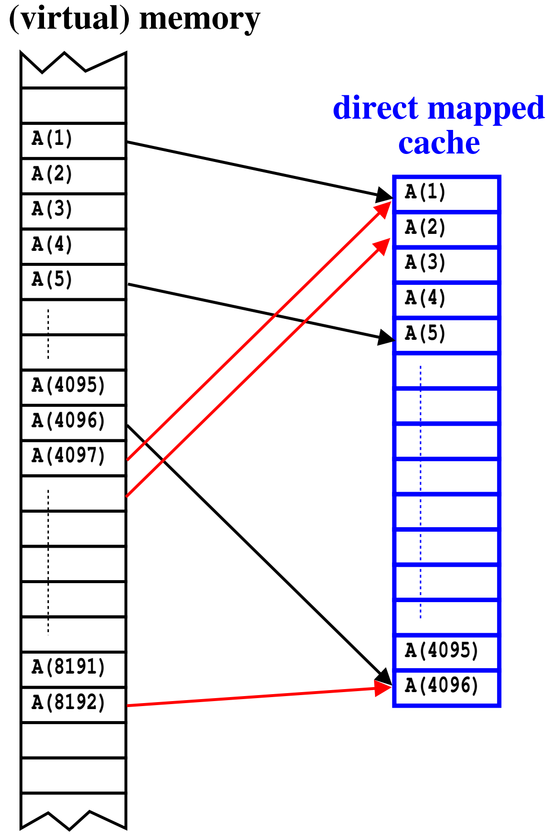
### Un ejemplo:

- **Supongamos: cache size = 16 KByte = 16*1024 bytes**
- **Esto corresponde a: 4*1024 = 4096 32–bit words**
- **Cargamos un 32–bit array A with 8192 elements**





14




**memory location:**

A(4097) = A(1) + 4096*4

**cache location:**

A(4097) = A(1)

### 1. Direct Mapped

- **Un conocido efecto de este diseño es el:**

  **"TRASHING"**

- **Los elementos que todavía son necesarios son machacados continuamente.**
- **Especialmente cuando hay múltiples arrays involucrados este esquema puede ser muy ineficiente.**
- **Normalmente la solución es modificar el mapping a memoria, pero no es nada trivial.**
- **La detección se hace por comparación entre los resultados del PCsampling con el Basic Block Counting**

## Fully Associative Caches

15

### 2. Fully Associative

- EL algoritmo de reemplazo es un LRU: Least–Recently Used
- Las líneas más antiguas, son las reemplazadas
- En la mayoría de casos es lo más lógico
- Normalmente ayuda mucho con múltiples arrays

Atención, pregunta:

Por qué no son todas las caches Fully Associative ???

Respuesta: POR COSTE !!!

Afortunadamente hay una alternativa...

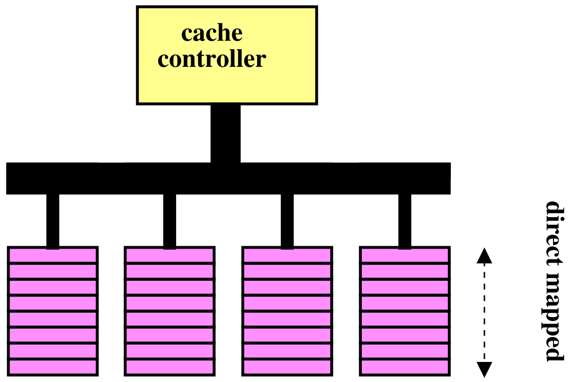


## N–Set Associative Caches

16

- La cache contiene varias direct mapped caches
- Las líneas de cache pueden ir a cualquiera de ellas (denominadas "set")
- La selección de cúal es normalmente (semi–) LRU

Example: 4–way set associative cache




- Es una solución cost–effective
- 2, 3 o 4 sets es lo habitual
- Elimina la mayoria de problemas de trash que aparecen en las direct–mapped
- El Trashing puede todavía darse dentro del set

## False Sharing

1 7

**Supongamos que ejecutamos este loop en 2 procesadores:**

```
for (i=0; i<10; i++)
      a[i] = b[i] + c[i];
```

| | | |
|---|---|---|
| procesador 1 | para | i=0,2,4,6,8 |
| procesador 2 | para | i=1,3,5,7,9 |

**Esto <u>no</u> es una manera muy eficiente de paralelizar. Por qué ?**

**Por el uso de las líneas de cache:**

| | |
|---|---|
| b[] and c[]: | usamos la mitad de los datos |
| a[]: | efecto de false sharing |



## False Sharing

1 8

**<u>Processor 1</u>**

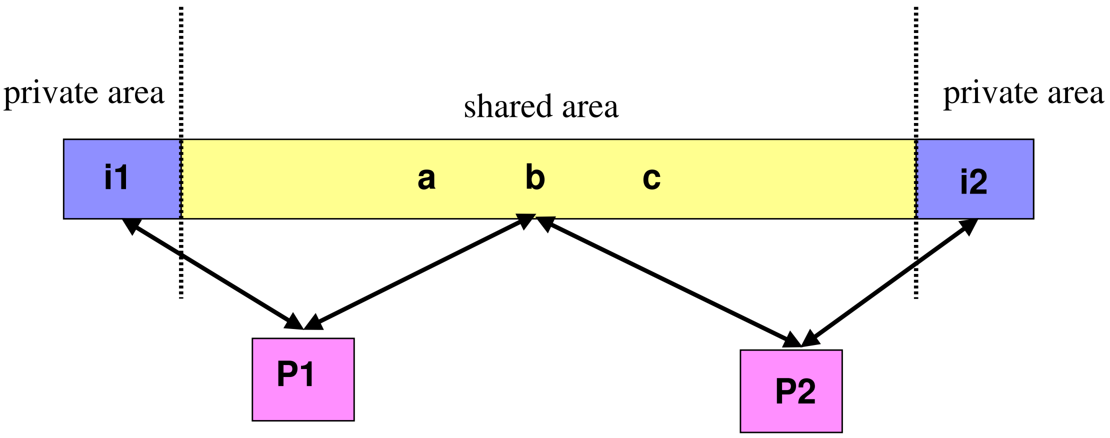
for i1 = 0, 2, 4, 6, 8 do:
a[i1] = b[i1] + c[i1];

Read: b,c Write: a

**<u>Processor 2</u>**

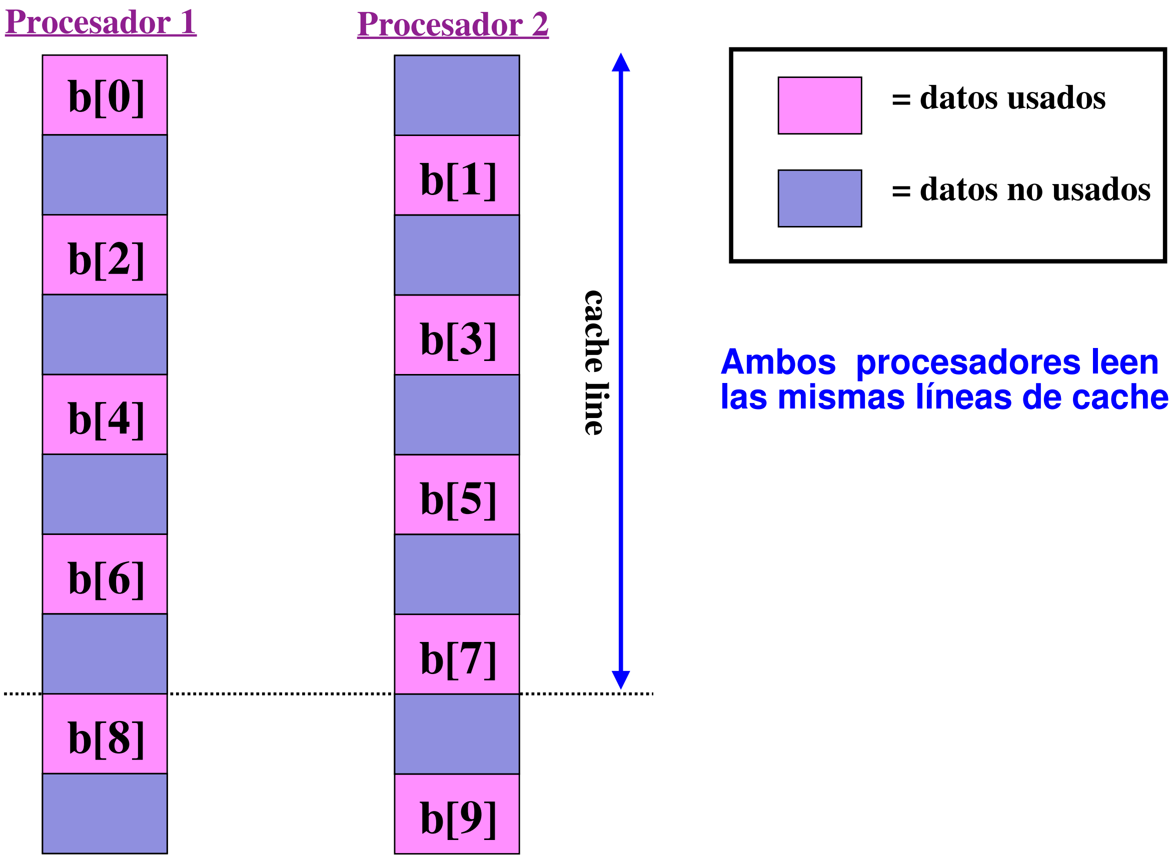
for i2 = 1, 3, 5, 7, 9 do:
a[i2] = b[i2] + c[i2];

Read: b,c Write: a



## False Sharing

19




**Ambos procesadores leen las mismas líneas de cache**



## False Sharing

20

**Procesador 1**

a[0] = b[0] + c[0]

Escribe en la línea que contine a[0]

Esto marca la línea de cache que contiene a[0] como "dirty"

a[2] = b[2] + c[2]

Detecta que la línea de a[2] es dirty

Coge una nueva copia (del procesador 2)

Escribe en la línea que contiene a[2]

Esto marca la línea que contiene a[2] como "dirty"

**Procesador 2**

a[1] = b[1] + c[1]

Detecta la línea con a[1] como dirty

Coge una nueva copia (del procesador 1)

Escribe en la línea que contiene a[1]

Esto marca la línea de cache que continene a[1] como "dirty"

a[3] = b[3] + c[3]

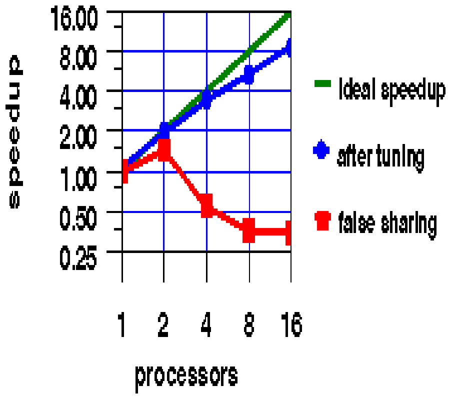
Detecta que la línea con a[3] es dirty

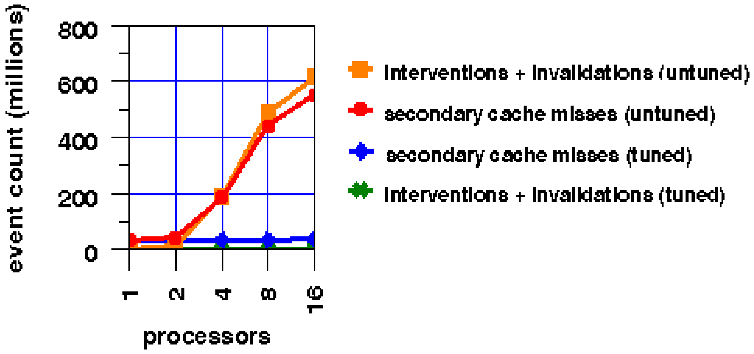
ETC

tiempo

## False Sharing

21






## Transaction Lookaside Buffer (TLB)

22

- El cálculo de direcciones reales de memoria es un cálculo costoso en tiempo. Por ello se guarda una cache de direcciones llamada Transaction Lookaside Buffer (TLB)
- Los fallos de TLB (TLB mises) son como fallos de cache. Lo único que en lugar de ir a buscar el dato a memoria hay que volver a calcularlo.
- Los procesares suelen tener un tamaño de TLB limitado (64 entries en R10000).
- Se puede mitigar bastante aumentando el tamaño de página