# Arquitectura de Computadores

## Tema 2-1

## Técnicas para Aumentar las Prestaciones de la Memoria Cache

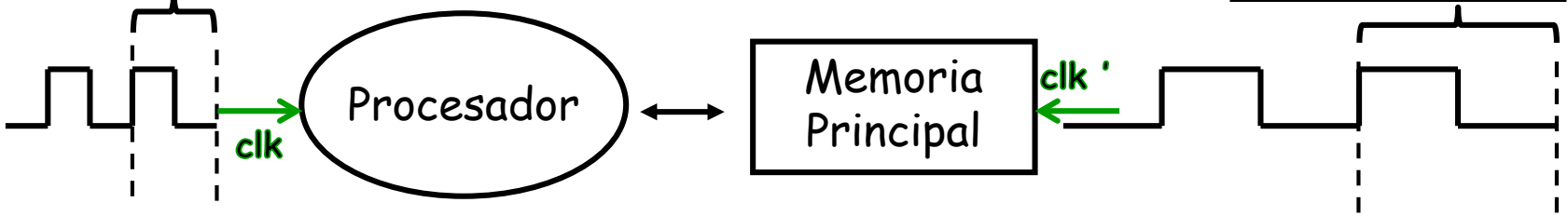
# Sumario

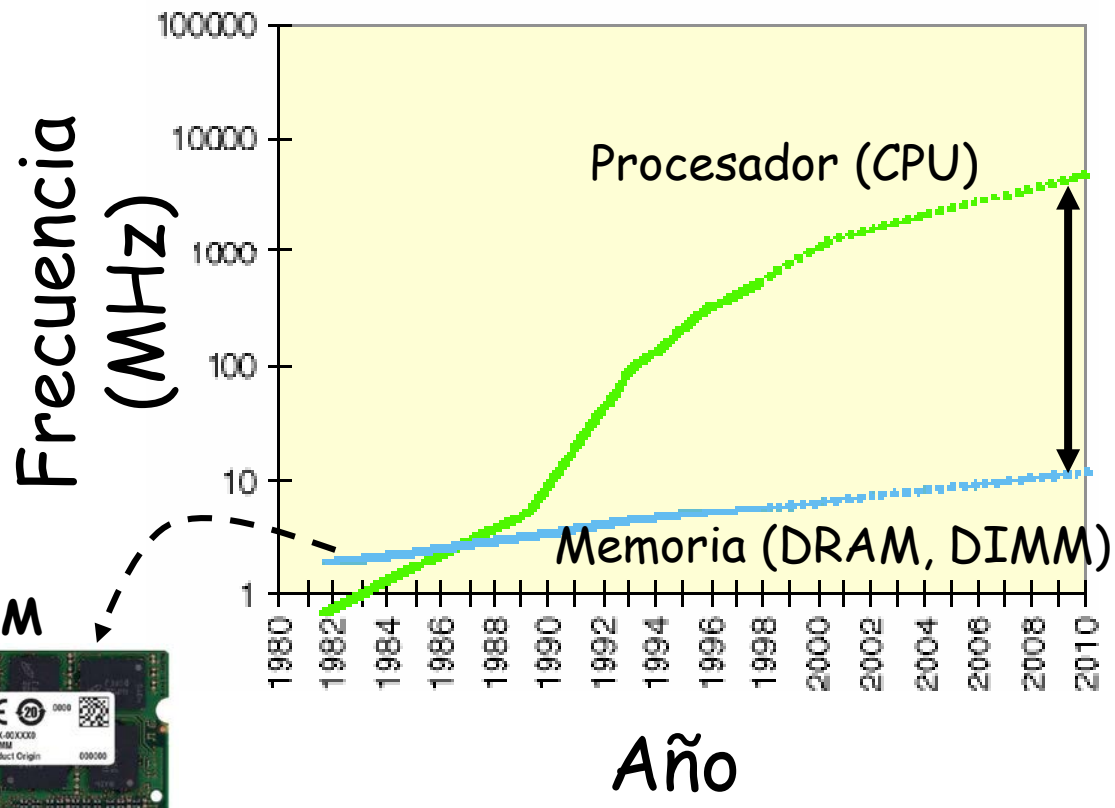
- Generalidades
  - Gap Tecnológico Procesador-Memoria
  - Problema "Memory Wall"
  - Jerarquía de Memoria
  - Repaso de Caches
- Medida de Prestaciones en la Cache
- Técnicas hardware de aumento de prestaciones
- Técnicas software de aumento de prestaciones

# ¿Por qué es necesario preocuparse del diseño de la Memoria?

~1 instrucción terminada por ciclo de reloj del procesador

~1 acceso de memoria terminado por ciclo de reloj de la memoria DRAM




DIMM

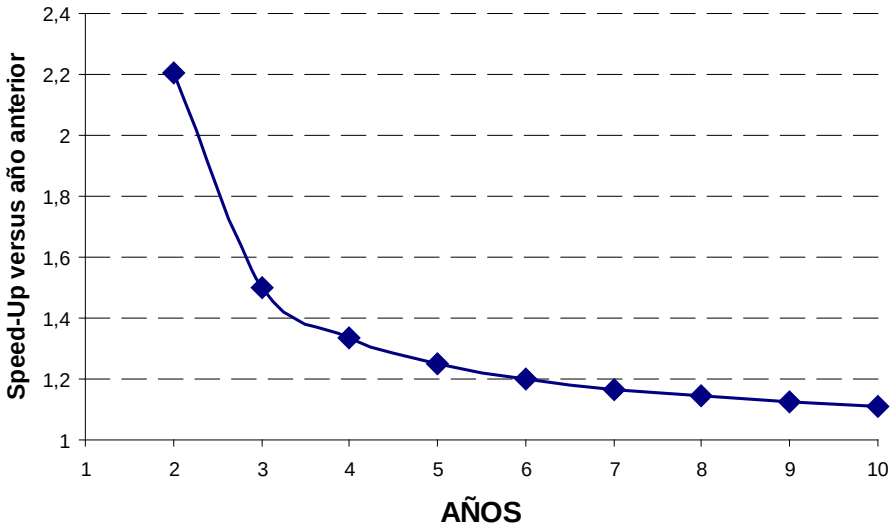

**Más de 3 órdenes de magnitud de diferencia de frecuencia de acceso: CPU-DRAM(DIMM)**

**Causa: distintas tecnologías electrónicas de fabricación producen distintos retardos en la propagación de señales en CPU y DIMM**

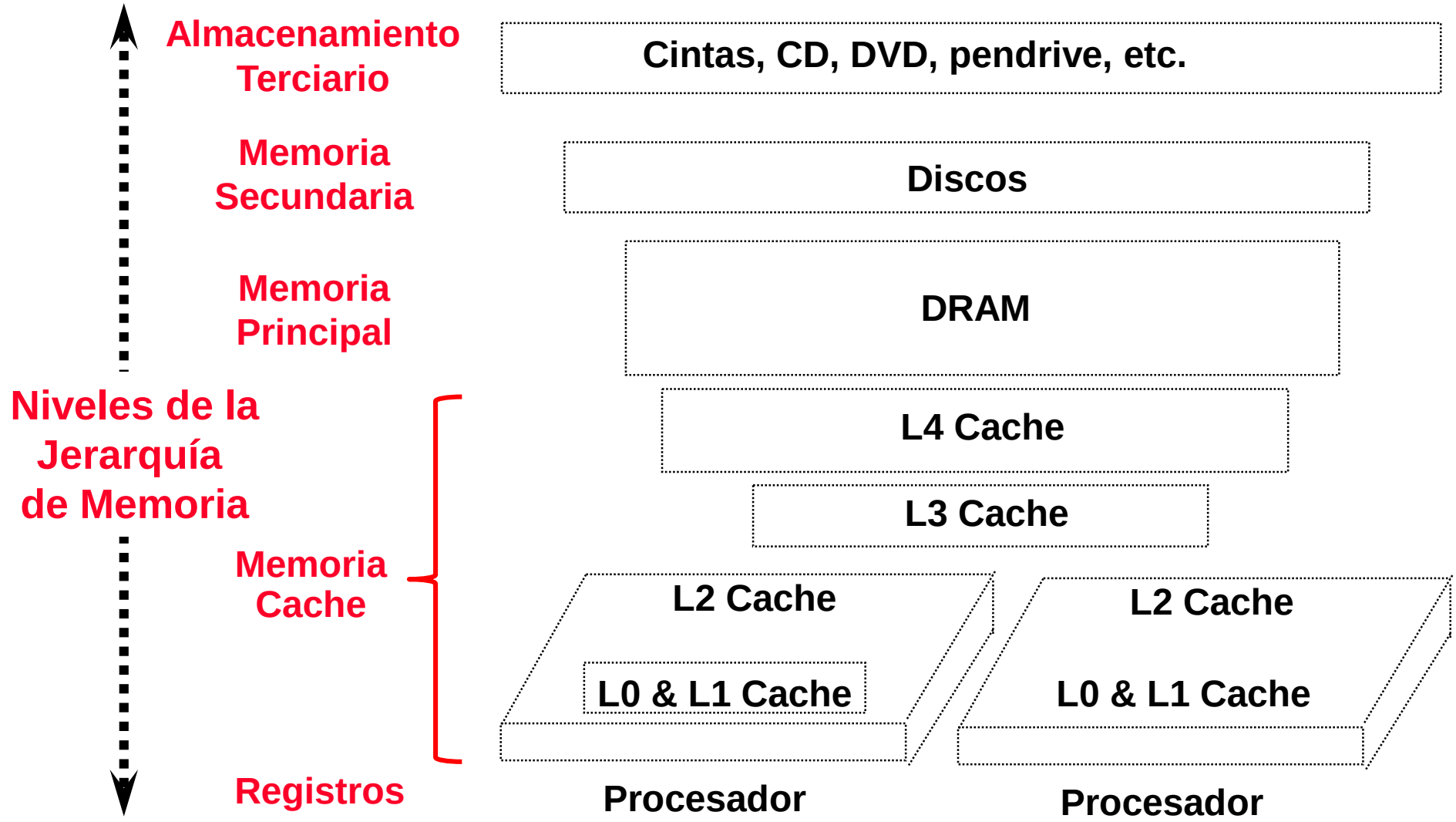
# Problema "Lentitud de la Memoria (Memory Wall)"

- Porcentaje de instrucciones de acceso a memoria: %M=30%. Resto Instrucciones: %NM=70%.
- Ley de Amdahl: Sp-Up = 1 / (%M/b + %NM/a)
- Aumento Prestaciones Lógica: a= años x 1.35/año
- Aumento Presaciones DRAM: b= años x 1.033/año
- Conclusión: las instrucciones de acceso a memoria impiden que se produzca un aumento de prestaciones del computador debido al tipo de tecnología DRAM-DIMM

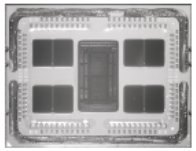
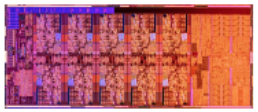
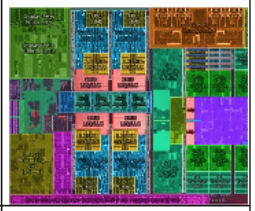
| Años | Speed-Up versus "Año 0" | Speed-Up versus año anterior |
|---|---|---|
| 1 | 1,22 | |
| 2 | 2,70 | 2,21 |
| 3 | 4,04 | 1,50 |
| 4 | 5,39 | 1,33 |
| 5 | 6,74 | 1,25 |
| 6 | 8,09 | 1,20 |
| 7 | 9,43 | 1,17 |
| 8 | 10,78 | 1,14 |
| 9 | 12,13 | 1,13 |
| 10 | 13,48 | 1,11 |



# Solución: Jerarquía de Memoria



# Ejemplos de Jerarquía de Memoria en Procesadores Reales

| Procesador | AMD Ryzen 9 4900H | Intel Core i9-10980HK | Intel Core i7-11375H |
|---|---|---|---|
| Foto |  |   |   |
| Repertorio ISA | x86-64 (64-bit) | x86-64 (64-bit) | x86-64 (64-bit) |
| Dominio Aplicaciones | Sobremesa: portátil | Sobremesa: PC/Cliente/Tableta (serie H) | Sobremesa: portátil (serie H) |
| Tamaño característico (fecha lanzamiento comercial, nombre producto comercial) | 7 nm TSMC CMOS FinFET (Q1'20, Renoir) | 14 nm ++ bulk silicon 3D Tri-Gate xtors (Q2'20, Generación 10) | 10 nm SuperFin (Q1'21, Generación 11) |
| Dado silicio / número de transistores / encapsulado | 156 $mm^2$ / 9800 millones xtores / FP6 (BGA) | ¿? $mm^2$ / ¿? Millones xtores / FCBGA1449 | ¿? $Mm^2$ / ¿? Millones xtores / FCBGA1440 |
| Núcleos/subprocesos/GPU | 8/16/8 | 8/16/1 | 4/8/1 |
| Microarquitectura / etapas segmentación | Zen 2 / 19 | Comet Lake / 14-19 | Tiger Lake / ¿? |
| Máximo número de instrucciones enviadas a ejecutar por ciclo (dispatch) | 6 | 5 | ¿? |
| Frecuencia reloj | 3,3-4,4 GHz | 2,4-5,3 GHz | 3-5 GHz |
| Cache L0 uOP (microoperaciones) | 4096 uOPs, 8 vías | 1536 uOPs, 8 vías | |
| Cache L1 Instrucciones por core | 32 KiB, 8 vías | 32 KiB, 8 vías | 32 KiB, ¿? Vías |
| Latencia L1I (ciclos) | ¿? | ¿? | ¿? |
| Cache L1 Datos por core | 32 KiB, 8 vías | 32 KiB, 8 vías | 48 KiB, ¿? Vías |
| Latencia L1D (ciclos) | 4-8 clk | 4 clk | ¿? |

L0 & L1 cache

# Ejemplos de Jerarquía de Memoria en Procesadores Reales

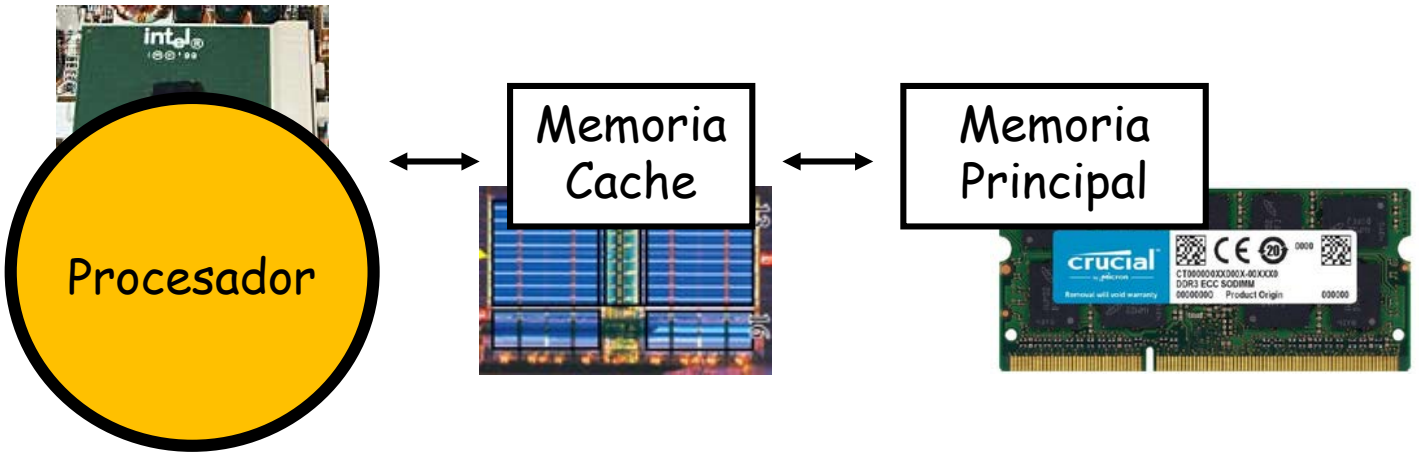
| | | | |
|---|---|---|---|
| Entradas TLB (I/D/L2 TLB) | 64@full-assoc / 64@8-vias / 512@8-vías+2048@16-vías | 128@8-vías / 64@4-vías / 1536@12-vías | ¿? |
| Tamaño página mínimo | 4 KiB | 4 KiB | ¿? |
| Cache L2 por core | 512 KiB, 8 vías | 256 KiB, 4 vías | 1280 KiB, ¿? Vías |
| Latencia L2 (ciclos) | 12 clk | 12 clk | ¿? |
| Cache L3 compartida en chip | 8 MiB, 16 Vías | 16 MiB, 16 vías | 12 MiB, ¿? Vías |
| Latencia L3 (ciclos) | 39 clk | 42 clk | ¿? |
| Tamaño bloque (L1/L2/L3 bytes) | 64 bytes | 64 bytes | ¿? |
| Bus de memoria (bits) | 64 | 64 | 64 |
| Interfaz con memoria DRAM | 2 canales; DDR4-3200: f=400MHz, ritmo=3,2Gbs; LPDDR4x-4267: f=200MHz, ritmo=4,2Gbs | 2 canales; DDR4-2933: f=366,67MHz, ritmo=2,9Gbs | DDR4-3200: f=400MHz, ritmo=3,2Gbs, 2 canales x 64-bit; LPDDR4x-4267: f=200MHz, ritmo=4,2Gbs, 4 canales x 32 bits |
| Latencia memoria | ¿? | 42 clk + 51 ns latencia | ¿? |
| Consumo de potencia | 35-54 W | 45-65 W | 28-35 W |
| Precio de salida | ¿? | $ 583 | $ 482 |

L2 & L3 cache

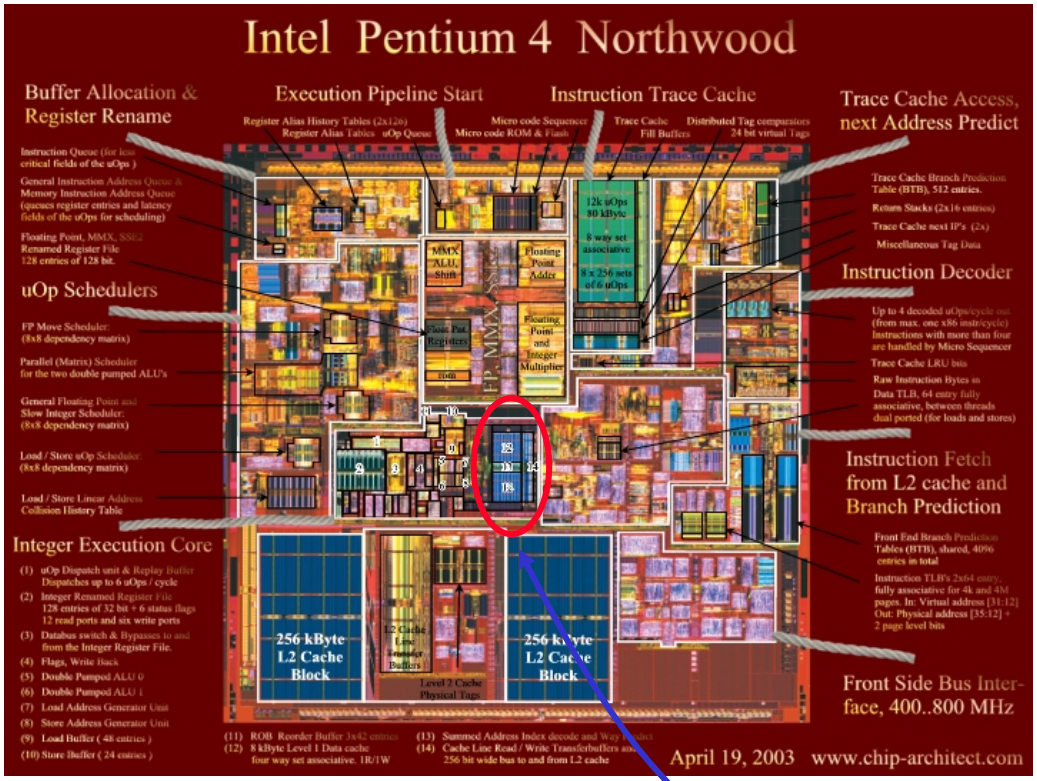
Memoria DRAM

# ¿Qué es una Memoria Cache?

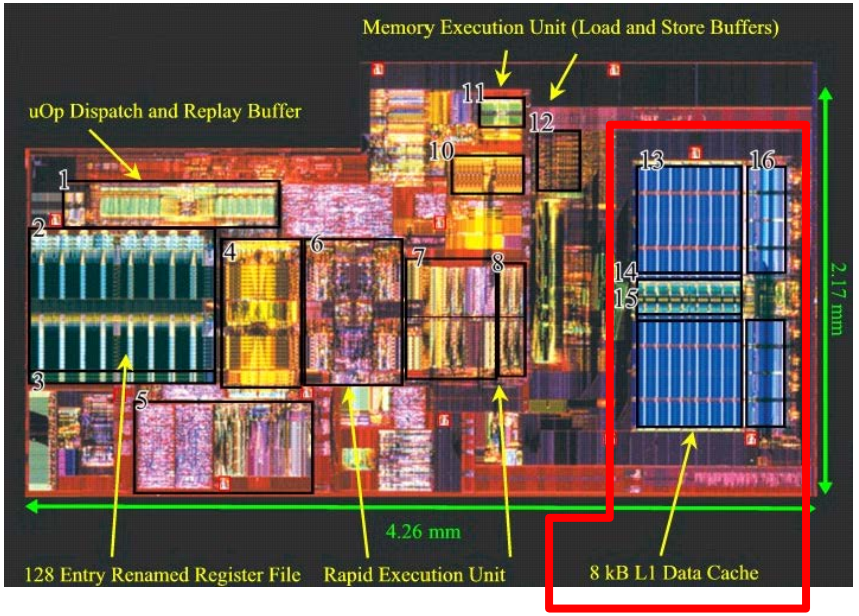
- Guarda temporalmente una réplica de los datos e instrucciones que se encuentran en memoria principal
- Pequeña: sólo permite guardar una pequeña parte de la memoria principal
- Tarda menos tiempo que la memoria principal en proporcionar datos/instrucciones al procesador
- Explota ambos tipos de localidades de los datos: temporal, espacial
- Muchas partes del computador disponen de cache



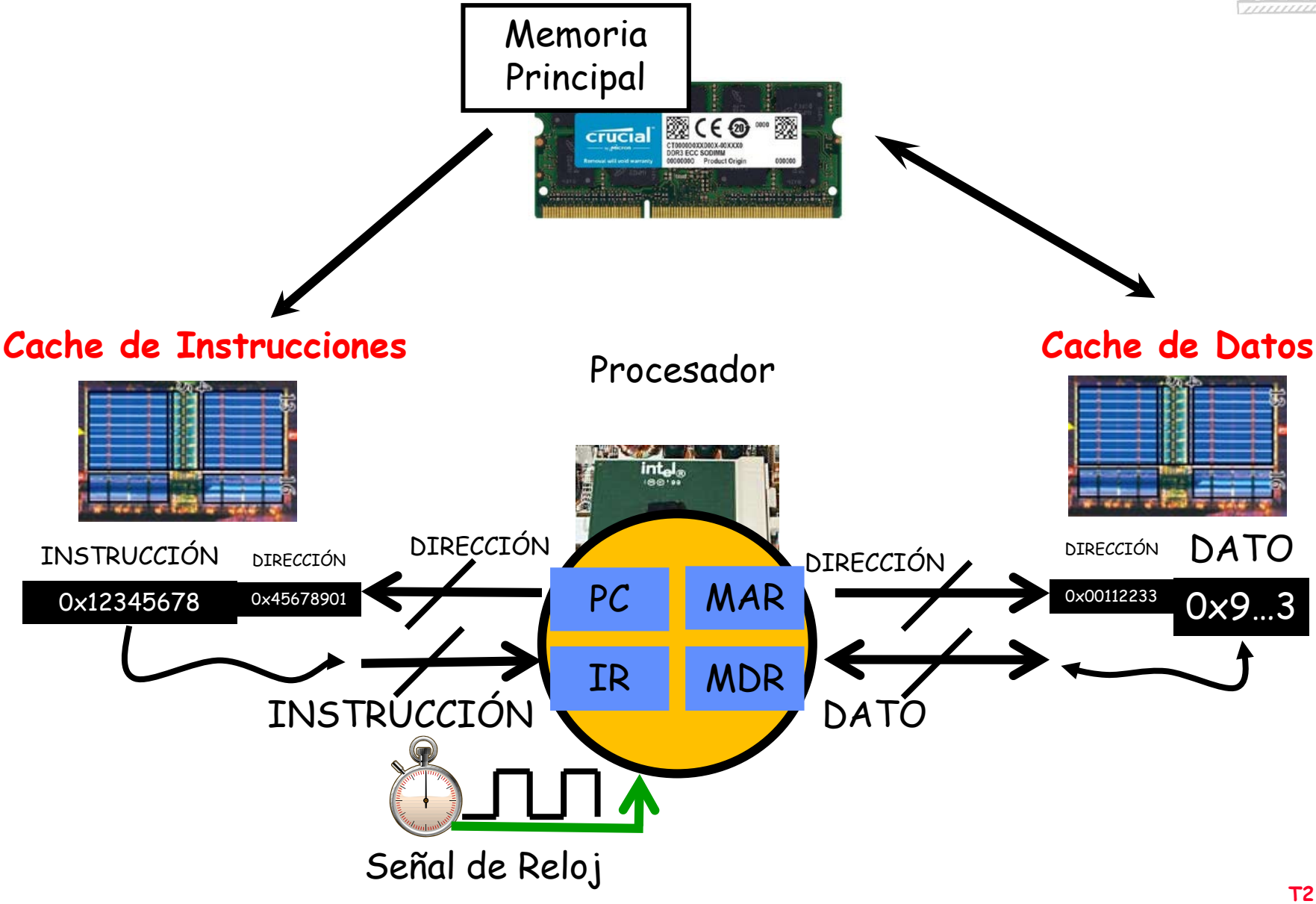
# Memoria Cache L1 de Datos en Pentium 4 "Northwood"




Pentium 4 - Northwood - 130 nm



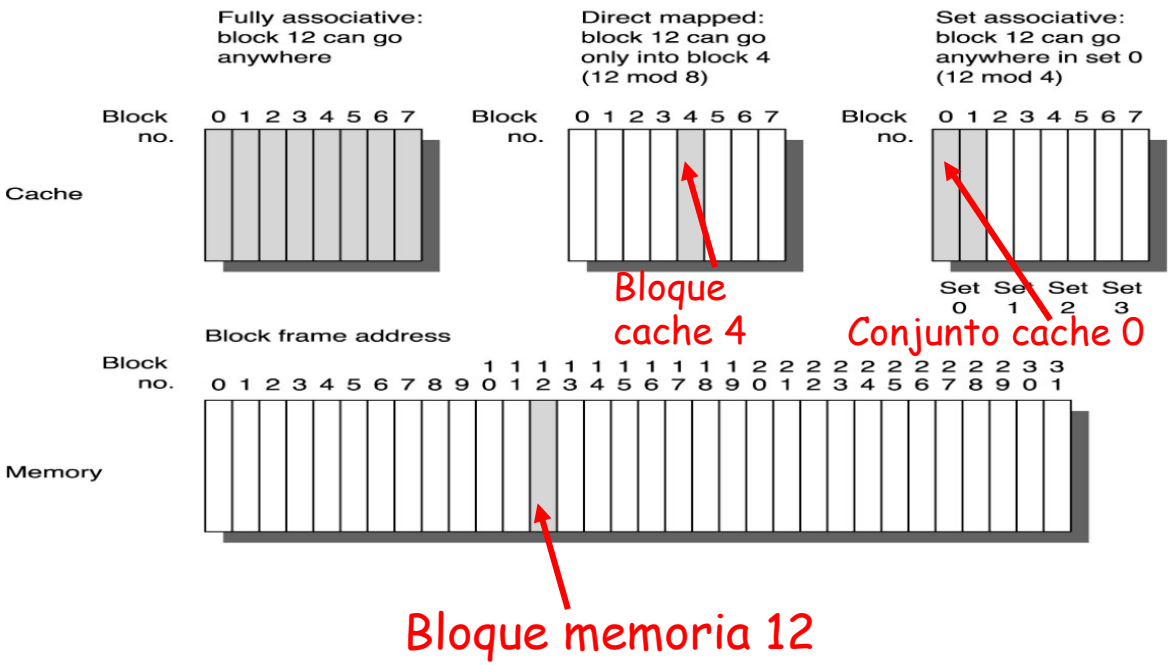
# Cache L1: Instrucciones + Datos



# 4 decisiones involucradas en el Diseño de cada nivel de la Jerarquía de Memoria y particularmente de la cache

- Q1: ¿ Dónde puede ser emplazado un bloque en un nivel de la Jerarquía ? *(Emplazamiento del Bloque)*
- Q2: ¿ Cómo se puede saber que un bloque se encuentra en un nivel determinado de la Jerarquía de Memoria ? *(Identificación del Bloque)*
- Q3: ¿ Qué bloque debe ser remplazado en un fallo ? *(Remplazamiento del Bloque)*
- Q4: ¿ Qué ocurre en una escritura ? *(Estrategia de Escritura)*

# Q1: Emplazamiento del Bloque
# ¿Dónde se "puede" encontrar un bloque de cache?




## TIPOS DE MEMORIA CACHE

**Completamente Asociativa:**
Bloque 12 en cualquier lugar

**Correspondencia Directa:**
Block no. = (Dir. Bloque) mod (No. Bloques en cache)
Bloque 12 sólo en Bloque 4 (12 mod 8)

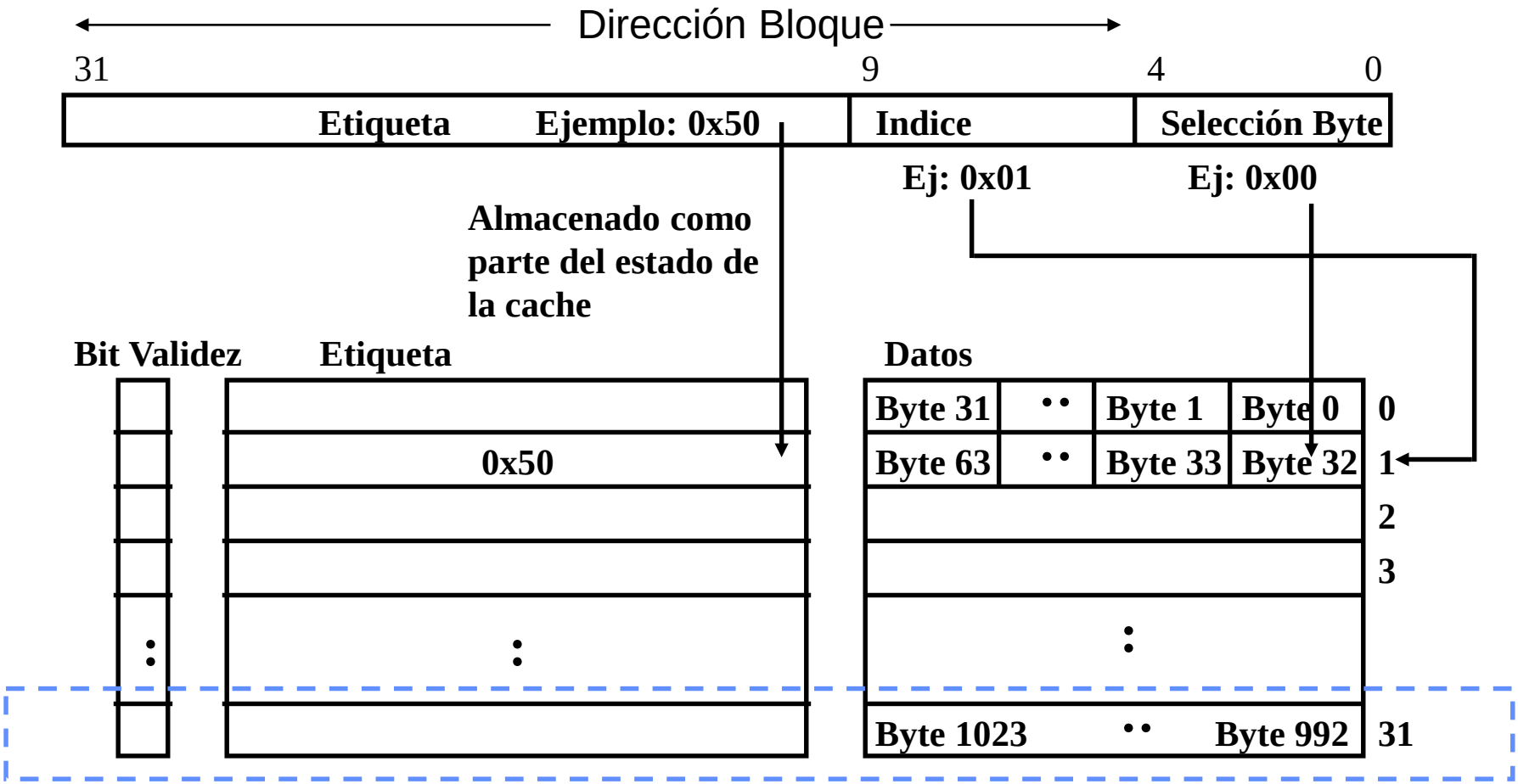
**Asociativa por Conjuntos:**
No Conjunto = (Dir. Bloque) mod (No. Conjuntos en cache)
Bloque 12 corresponde a Conjunto 0 (12 mod 4)

# Q2: Identificación Bloque
# ¿Cómo se sabe que una dirección de memoria se encuentra en la cache

- Un nivel de la jerarquía se divide en bloques de direcciones
- Dirección de acceso al nivel de la jerarquía se divide en campos:
  - Desplazamiento Bloque: selecciona un byte en el bloque
    - » tamaño campo desplazamiento = $\log_2$(tamaño bloque en bytes) [bits]
  - Indice: selecciona conjunto
    - » tamaño campo indice = $\log_2$(#bloques/asociatividad)
  - Etiqueta: determina el acierto en la cache
    - » tamaño campo etiqueta = tamaño dir - tamaño indice - tamaño desplazamiento
  - Dirección Bloque: etiqueta + indice


FIGURE 5.3 The three portions of an address in a set-associative or direct-mapped cache.

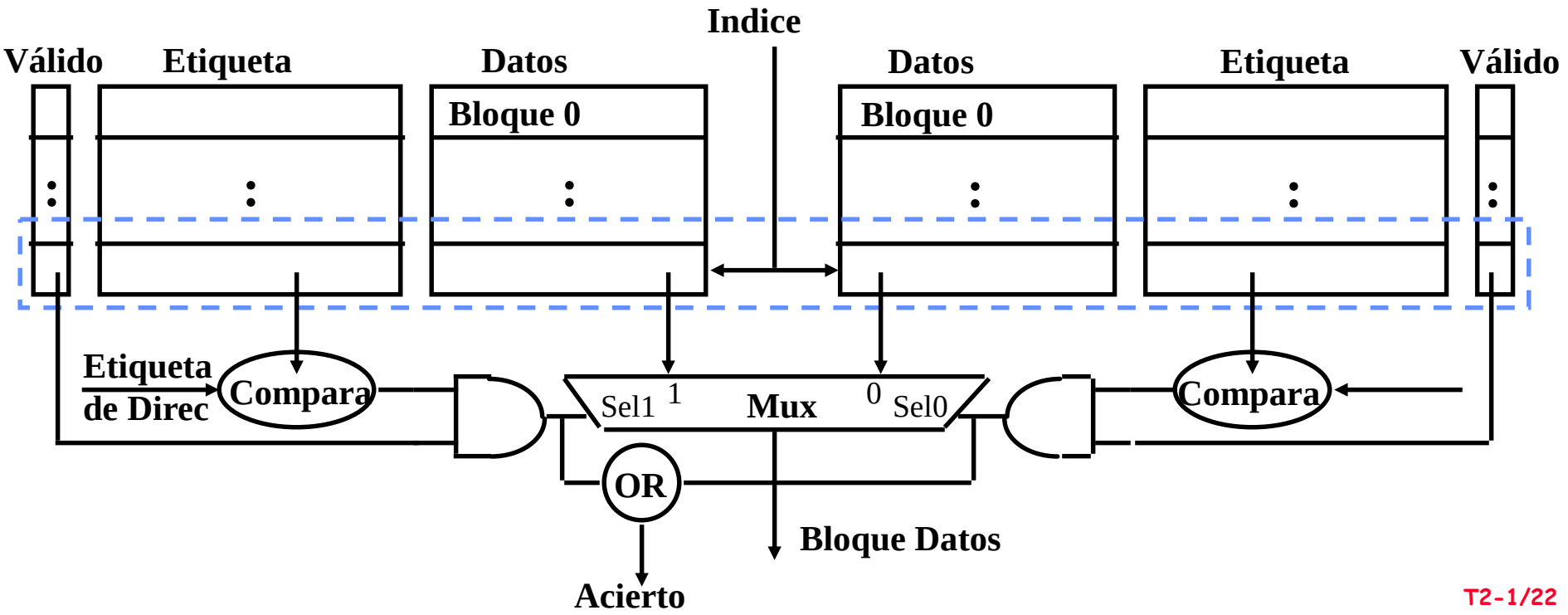
# Ejemplo: Cache 1 KB CD, 32 Bytes/bloque



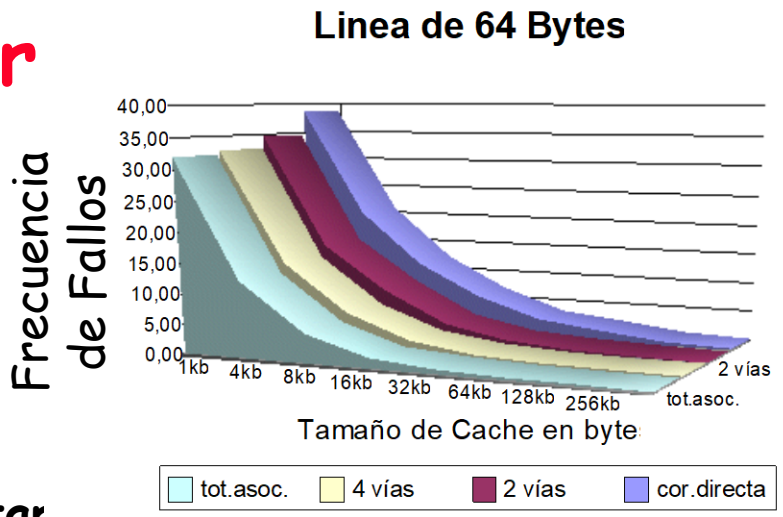

Línea de cache con un solo índice

# Cache Asociativa por Conjuntos

- Asociativa "N vías": N lineas para cada Indice
  - N caches CD en paralelo !
- Ejemplo: Asociativa 2 vías
  - Indice selecciona el Conjunto
  - Las 2 etiquetas son comparadas
  - El resultado de la comparación seleciona el dato



# Q3: ¿Qué Bloque debe ser remplazado en un Fallo ?

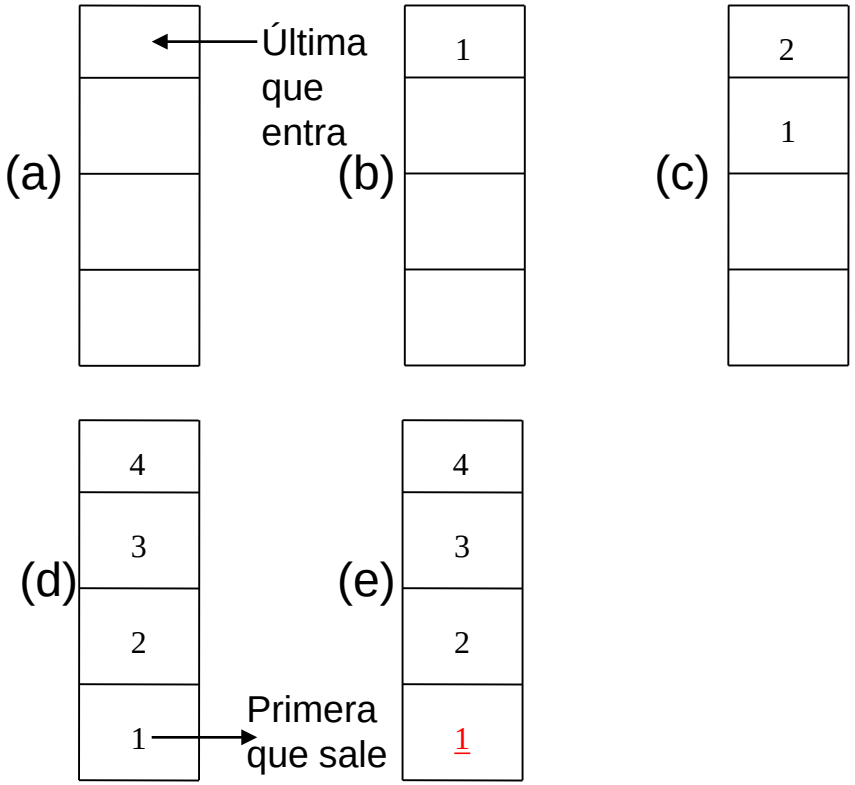



- CD: fácil !
- Asociatividad > 1:
  - Aleatorio - fácil de implementar
  - FIFO
  - Least Recently Used (LRU)
- Frecuencias de fallos para varios tipos cache

| Asocatividad: | 2-vías | | 4-vías | | 8-vías | |
|---|---|---|---|---|---|---|
| Capacidad | LRU | Aleatorio | LRU | Aleatorio | LRU | Aleatorio |
| 16 KB | 5.18% | 5.69% | 4.67% | 5.29% | 4.39% | 4.96% |
| 64 KB | 1.88% | 2.01% | 1.54% | 1.66% | 1.39% | 1.53% |
| 256 KB | 1.15% | 1.17% | 1.13% | 1.13% | 1.12% | 1.12% |

CONCLUSIÓN: Para las caches con baja frecuencia de fallos, las prestaciones de los remplazamientos aleatorio y LRU no se diferencian mucho.

# Política de Remplazamiento FIFO



# Q4-1: ¿Qué Ocurre en Aciertos de Escritura?

- **Escritura Directa (Write Through)**: La actualización se produce en la cache y en la memoria principal.
- **Escritura Retardada (Write Back)**: La actualización se produce sólo en la cache. El bloque modificado de la cache se actualiza en memoria principal sólo cuando es remplazado en la cache.
  - ¿Cómo se sabe que el bloque se debe actualizar en memoria?: bit modificado
- Pros y Contras :
  - Escritura Directa
    - » Los fallos de lectura (load) no ocasionan escrituras en MP
    - » Más fácil de implementar
    - » Siempre se combina con Búfers de Escritura para evitar las latencias de memoria
  - Escritura Retardada
    - » Menos tráfico a memoria
    - » Las escrituras se realizan a la frecuencia de la cache y no de la MP

# Q4-2: ¿Qué Ocurre en los Fallos de Escritura?

- 2 opciones:
  - Actualiza la Cache (Write Allocate)
    - » El bloque es llevado a la cache en un fallo escritura
    - » Se usa para las caches de Escritura Retardada
    - » Existe la esperanza de que las siguientes escrituras se produzca en ese bloque de cache
  - No Actualiza la Cache (No-Write Allocate)
    - » El bloque se modifica en memoria, pero no se lleva a la cache
    - » Se usa con las caches de Escritura Directa
    - » Fundamento: la escritura de datos tiene que ir a MP, entonces por qué llevar el bloque a la cache

# Medidas de Prestaciones en la Cache

- *Frecuencia de Aciertos*: % accesos aciertan en cache
  - *Frecuencia Fallos (FF) = 1 - Frecuencia de Aciertos*
- *Tiempo de Acierto ($t_{Acierto}$)*: tiempo para acceder a cache (L1-datos de Pentium 4: 2 ciclos)
- *Penalización por Fallo (P)*: tiempo necesario para remplazar un bloque en la cache desde un nivel superior
  - *Tiempo de Acceso*: tiempo para acceder al nivel superior
  - *Tiempo de Transferencia*: tiempo para transferir un bloque
  - $P = t_{Acceso} + t_{Transferencia}$
  - $t_{Transferencia}$ = Bytes / RT
  - Ritmo Transferencia [Bytes/segundo] : RT
- Tiempo de Acceso Promedio a la Memoria (AMAT)

$$AMAT = t_{Acierto} + FF \times P \quad ,, \quad [ns] \text{ ó } [ciclos]$$

# Aumento Prestaciones Cache

- **Tiempo de Acceso Promedio a la Memoria (AMAT)**

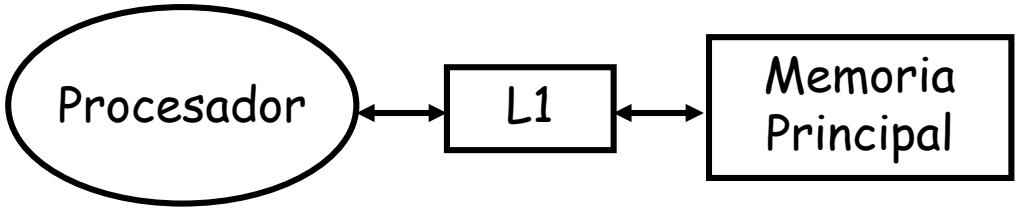
  $AMAT = t_{Acierto} + FF \times P$ ,, [ns] ó [ciclos]

- **Se mejoran las prestaciones a través de:**
  - A. Reducción de la Penalización por fallo
  - B. Reducción de la Frecuencia de Fallos
  - C. Reducción del Tiempo de Acierto

- **Técnicas para la Reducción de la Penalización**
  - 1: Caches Multinivel
  - 2: Dato crítico primero y comienzo inmediato
  - 3: Prioridad de las lecturas frente a escrituras en un fallo
  - 4: Cache de Víctimas

# A-1: Cache Multinivel

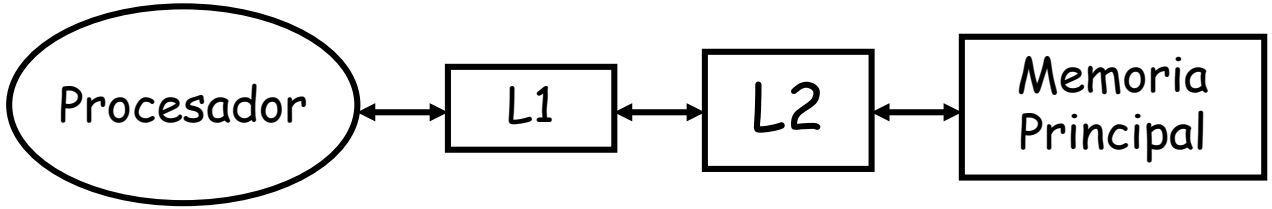
- Un segundo nivel de cache (L2) reduce la penalización por fallo al proporcionar una cache de capacidad mayor entre L1 y MP




- Ecuaciones

$$AMAT = t_{AciertoL1} + FF_{L1} \times P_{L1}$$
$$P_{L1} = t_{AciertoMP}$$
$$AMAT = t_{AciertoL1} + FF_{L1} \times t_{AciertoMP}$$

Penalización por acceso a MP




- Ecuaciones

$$AMAT = t_{AciertoL1} + FF_{L1} \times P_{L1}$$
$$P_{L1} = t_{AciertoL2} + FF_{L2} \times P_{L2}$$
$$AMAT = t_{AciertoL1} + FF_{L1} \times (t_{AciertoL2} + FF_{L2} \times P_{L2})$$

Penalización por acierto/fallo en L2

# B - Reducir Frecuencia de Fallos en la Memoria Cache

- **Clasificación de los Fallos:**
  - **Obligatorios:** 1ª vez que se accede a un bloque
  - **Capacidad:** después de que la cache esté llena con datos accedidos por el programa, el programa necesita más bloques que remplazan a los ya existentes que posteriormente vuelven a ser accedidos desde el programa
  - **Conflicto:** sin estar la cache completamente ocupada, varios bloques a los que accede el programa se "emplazan" en la misma vía, existiendo más bloques asignados a la misma vía que vías disponibles

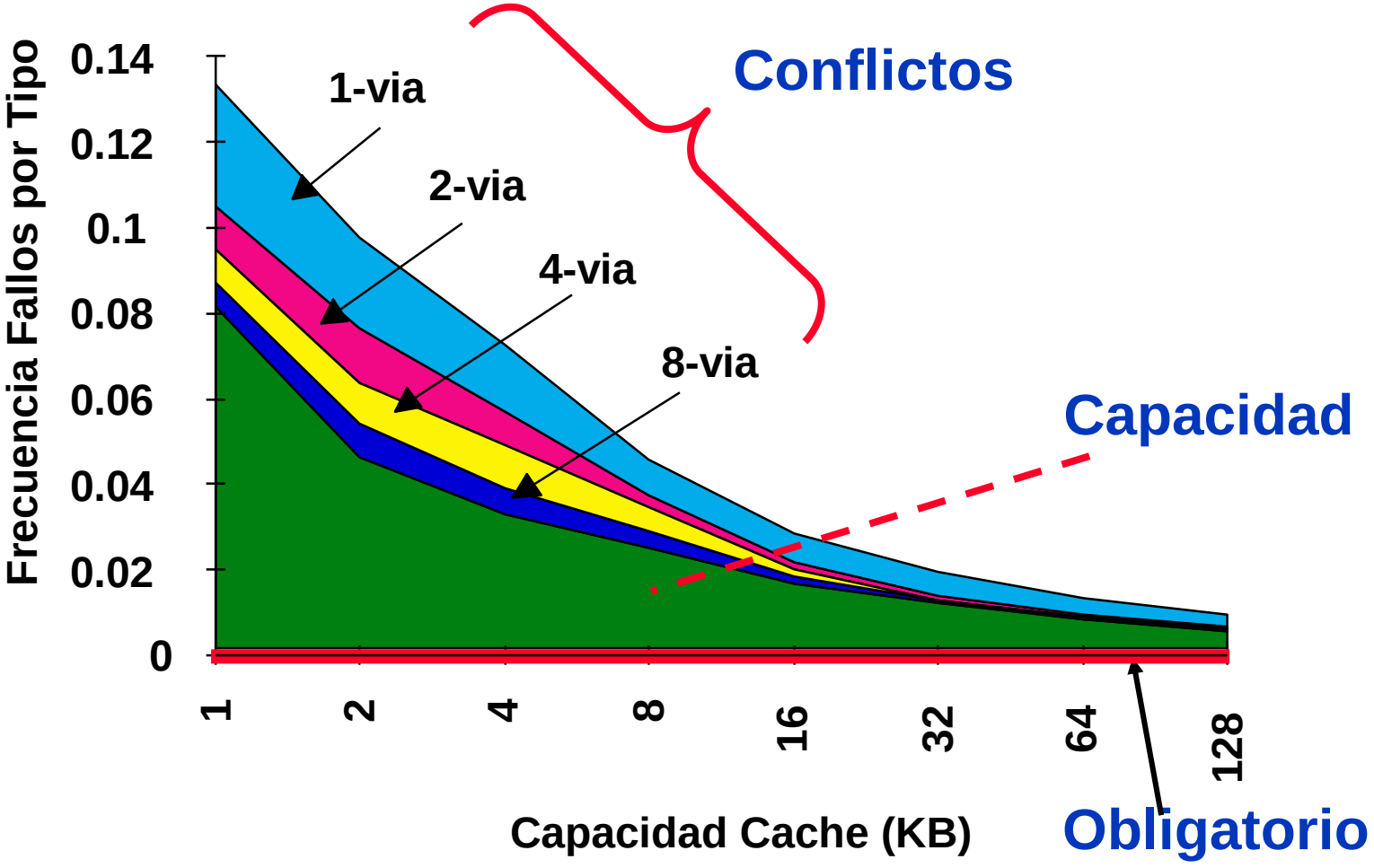
**TECNICAS de OPTIMIZACION HW PARA REDUCIR LA FRECUENCIA DE FALLOS**

- 1: Aumentar Capacidad
- 2: Aumentar Tamaño Bloque
- 3: Aumentar Asociatividad
- 4: Cache de Trazas

# B-1. Aumentar la Capacidad

- **Una forma de reducir la frecuencia de fallos consiste en incrementar la capacidad de datos que se pueden alojar**
  - Disminuye los conflictos por coincidencia de posiciones de memoria
- **Sin embargo, existen desventajas**
  - Puede incrementar el tiempo de acierto

# B-1. Aumentar la Capacidad: Prestaciones

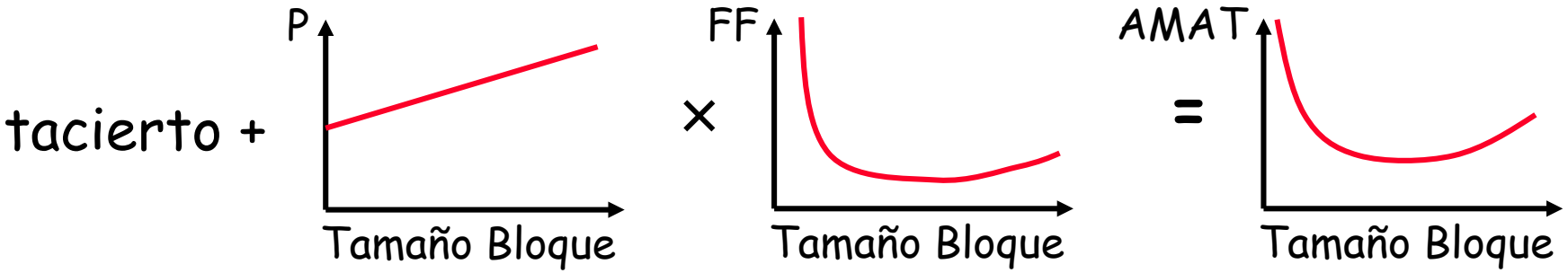


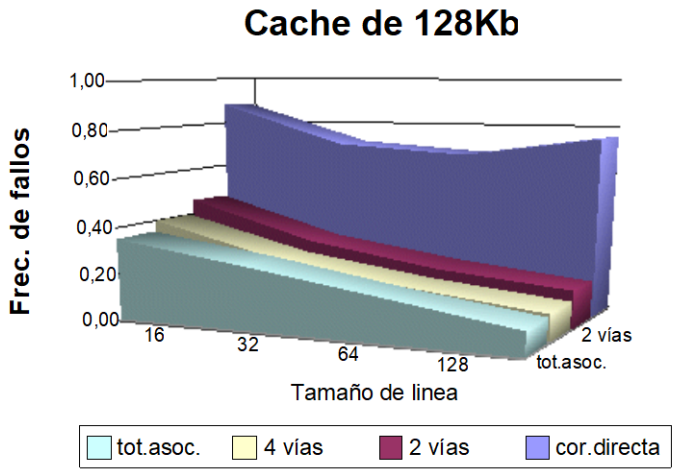
# B-2. Aumentar el Tamaño del Bloque

- **Una forma de reducir la frecuencia de fallos consiste en incrementar el tamaño del bloque de datos**
  - Aprovecha la localidad espacial
  - Reduce los fallos obligatorios – por qué?
- **Sin embargo, existen desventajas**
  - Puede incrementar la penalización por fallo (requiere más datos por línea)
  - Puede incrementar el tiempo de acierto (lee más datos y/o mux más grande)
  - Puede incrementar los fallos por conflictos – por qué?

# B-2. Aumentar el Tamaño del Bloque

- AMAT= $t_{acierto}$ + FF x P
- Aumentar el tamaño de bloque generalmente:
  - aumenta la Penalización por Fallo (P)
  - disminuye la Frecuencia de Fallos (FF)

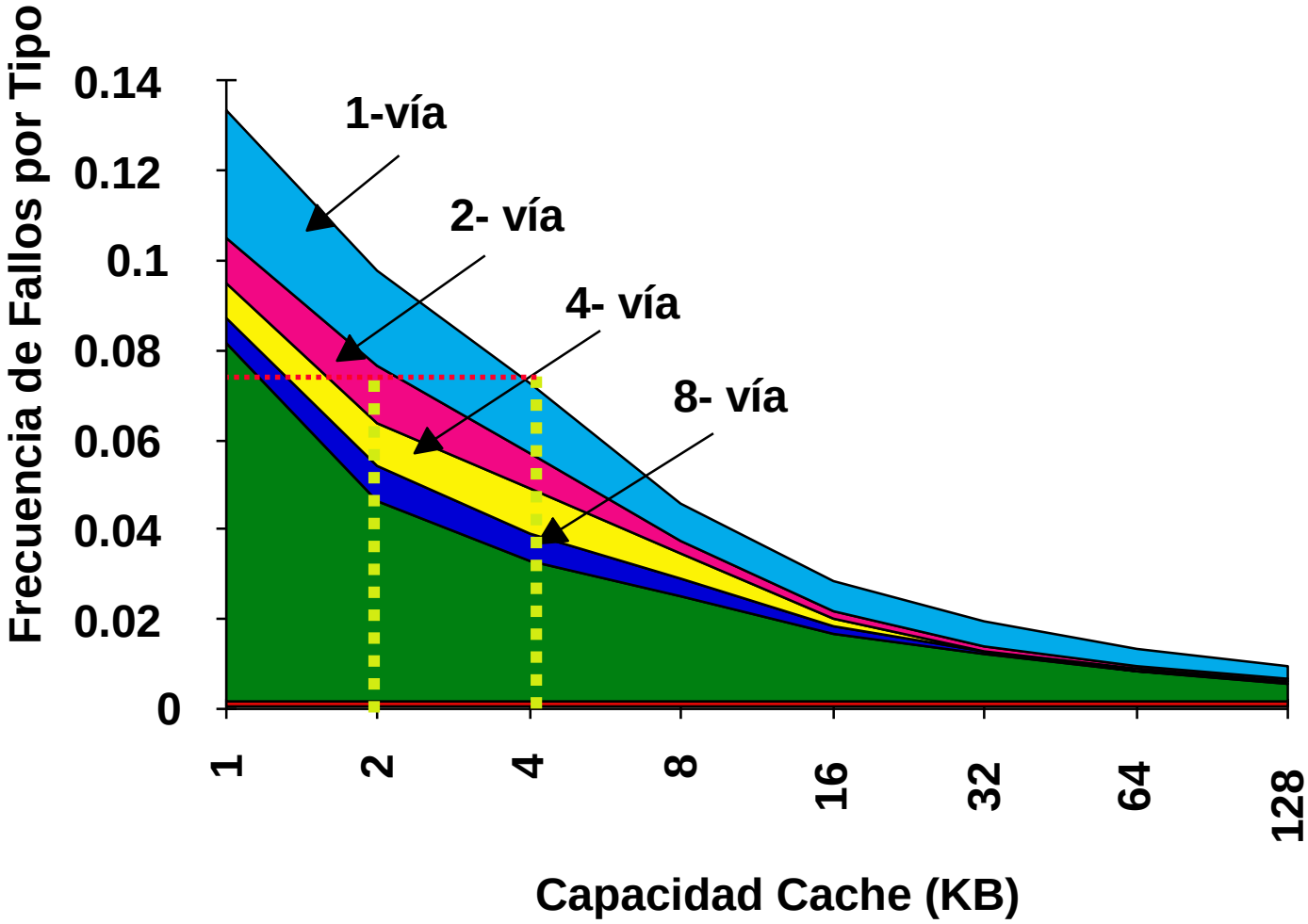


# B-3. Aumentar la Asociatividad

- **Aumentar la asociatividad ayuda a reducir los fallos por conflictos**
- **Regla 2:1 :**
  - La FF de una cache CD de tamaño N es aproximadamente igual a la FF de una cache asociativa 2-vías de tamaño N/2
- **Desventajas de la mayor asociatividad**
  - Requiere realizar mayor número de comparaciones
  - Requiere un mux n:1 para una asociatividad de n vías
  - Aumenta el tiempo de acierto

# Regla de la Cache "2:1"



# Ejemplo: AMAT vs. Asociatividad

- Suponer las siguientes caches: T=1 para CD, T=1.10 para 2-vías, T=1.12 para 4-vías, T=1.14 para 8-vías

| Tamaño (KB) | Asociatividad CD | 2-vías | 4-vías | 8-vías |
|---|---|---|---|---|
| 1 | 7.65 | 6.60 | 6.22 | 5.44 |
| 2 | 5.90 | 4.90 | 4.62 | 4.09 |
| 4 | 4.60 | 3.95 | 3.57 | 3.19 |
| 8 | 3.30 | 3.00 | 2.87 | 2.59 |
| 16 | 2.45 | 2.20 | 2.12 | 2.04 |
| 32 | 2.00 | 1.80 | 1.77 | 1.79 |
| 64 | 1.70 | 1.60 | 1.57 | 1.59 |
| 128 | 1.50 | 1.45 | 1.42 | 1.44 |

**AMAT:** tiende a disminuir a medida que aumenta la asociatividad y capacidad

Los datos en ROJO indican que AMAT no disminuye por aumentar la asociatividad

Estos datos no toman en cuenta el efecto de un reloj más lento sobre el resto del programa (sólo son medidas de AMAT y no de tCPU)

T: Tiempo de acierto de la memoria cache

# C - Reducción del Tiempo de Acierto

- 1. Caches Sencillas y Pequeñas
- 2. Caches Segmentadas

# C-1. Caches Sencillas y Pequeñas

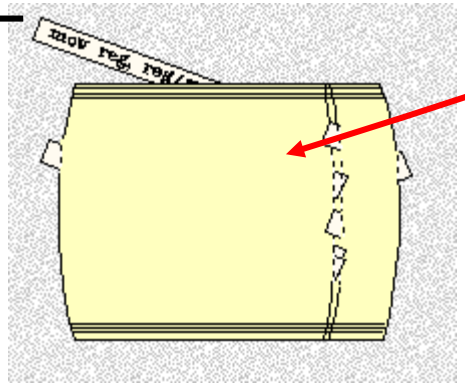
- Con caches de CD, se puede solapar la validación de la etiqueta con la transmisión de los datos al procesador
- La cache L1 se encuentra en el camino crítico de la ejecución de las instrucciones. En muchos procesadores determina el periodo de reloj ya que los aciertos en la cache L1 se realizan en 1 ciclo
- Ejemplos
  - Intel Pentium 4 "Presscott": L1 dCache 16 KB, 8 vías, 64 B/linea, 2 clk INT, 6 clk FP; L2 unificada 1 MB, 8 vías, 128 B/linea
  - AMD Opteron: L1 dCache 64 KB; L2 unificada 128KB

# Optimizaciones Software (Cache de Instrucciones)

- **Objetivo: reducir los fallos en la Cache de Instrucciones**
  - Reordenar la localización de procedimientos para reducir los fallos - ¿Qué tipo de fallos se reducen: obligatorios, capacidad, conflictos?
  - Realizar depuración para determinar cuándo se va a producir conflictos.
  - Reducir el tamaño del código para disminuir los fallos de capacidad.

Cache llena

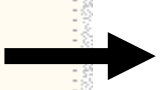
Hay que desalojar instrucciones que ya están



# Optimizaciones Software (Cache de Instrucciones)

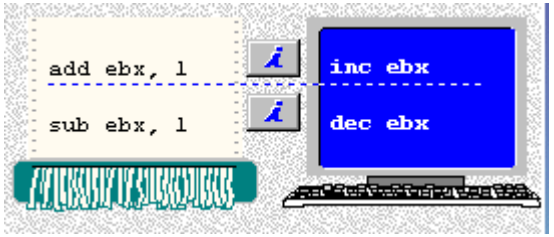
- **Reducción del tamaño de código para disminuir fallos por capacidad**
  - Cacular la dirección en la propia instrucción de acceso a memoria

```
1 shl edx, 3
2 add ebx, edx
3 add ebx, 32
4 mov ecx, [ebx]
```

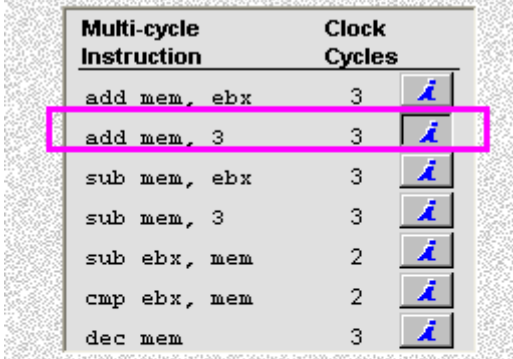


```
1 mov ecx, [ebx+edx*8+32]
```

  - Utilizar códigos de operación más pequeños




  - Sustituir varias Instrucciones Monociclo por una Instrucción Multiciclo



# Optimizaciones Software

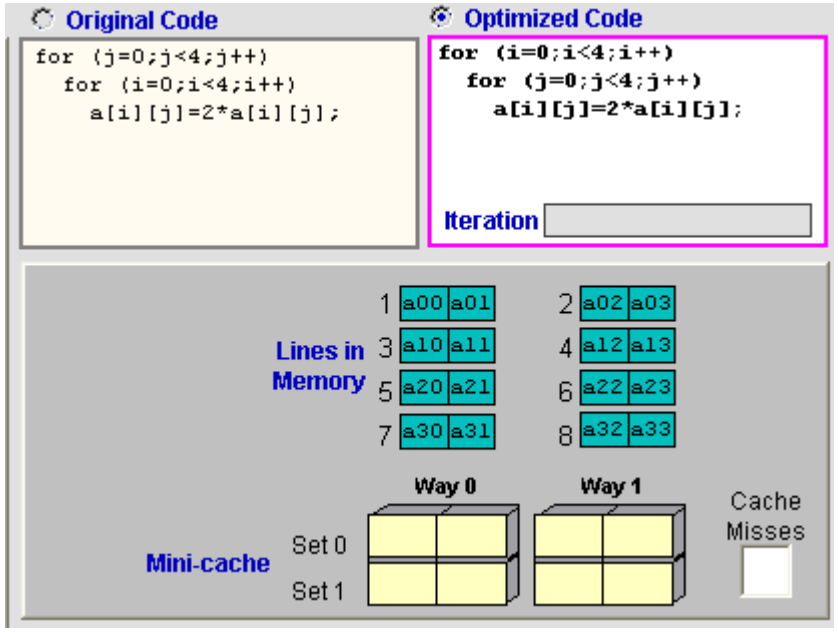
- Optimizaciones para la Cache de Datos
  - *Unión de Matrices*: mejora la localidad espacial del emplazamiento de matrices

# Optimizaciones Software

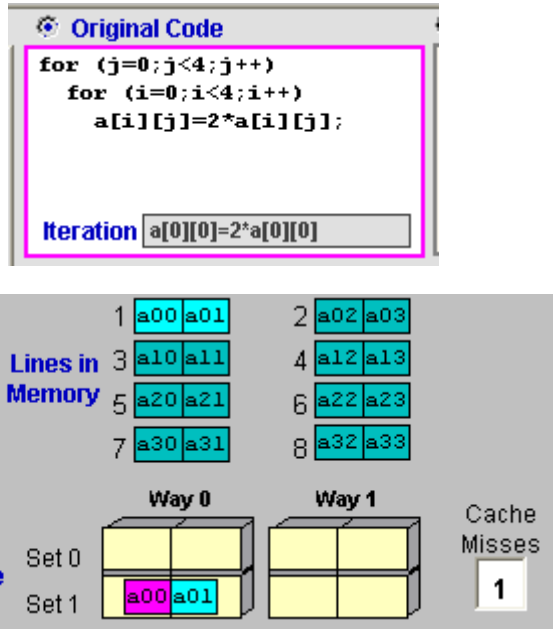
- **Optimizaciones para la Cache de Datos**
  - ***Intercambio de Bucles*: aprovecha la localidad espacial de los datos**

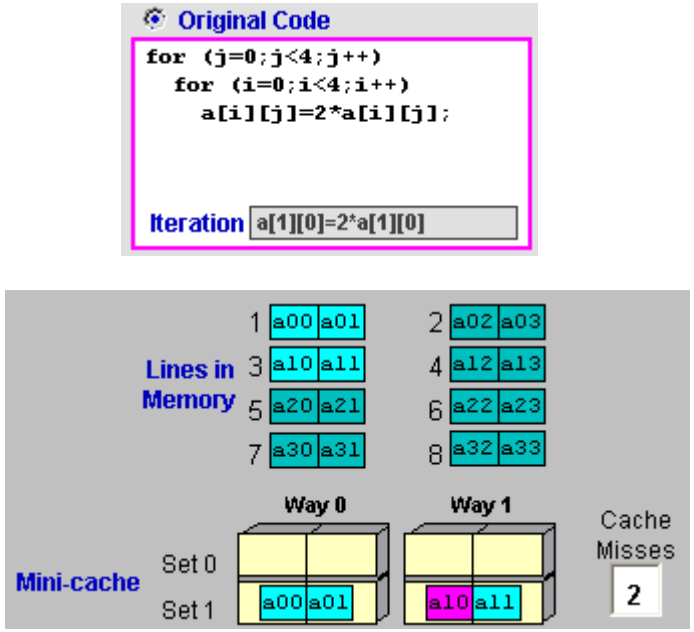
```
)0;
```

# Ejemplo: intercambio de bucles



# Ejemplo: intercambio de bucles



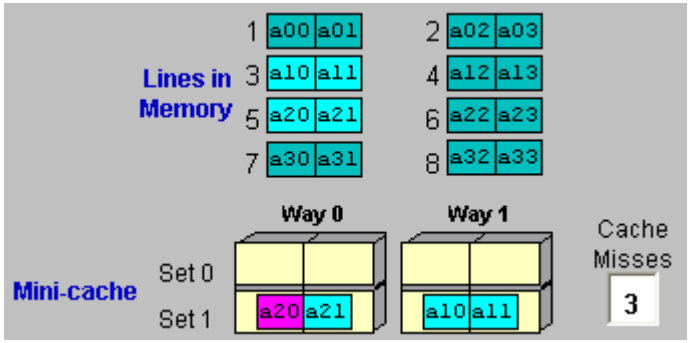



# Ejemplo : intercambio de bucles

Original Code

```
for (j=0;j<4;j++)
  for (i=0;i<4;i++)
    a[i][j]=2*a[i][j];
```

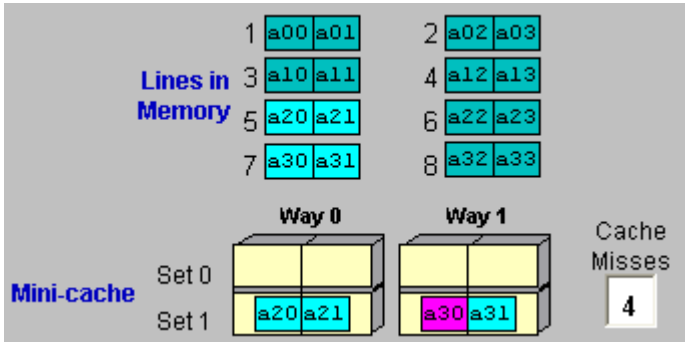
Iteration: a[2][0]=2*a[2][0]




Original Code

```
for (j=0;j<4;j++)
  for (i=0;i<4;i++)
    a[i][j]=2*a[i][j];
```

Iteration: a[3][0]=2*a[3][0]

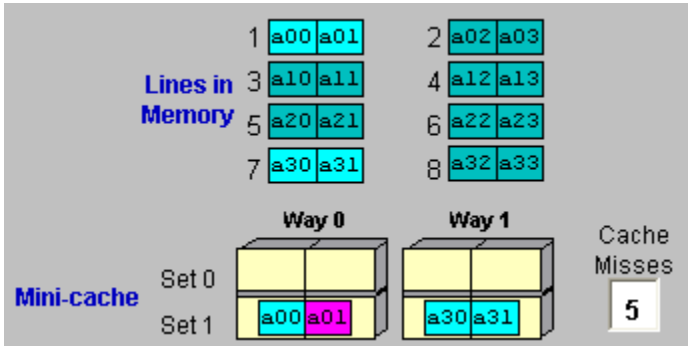


# Ejemplo : intercambio de bucles

Original Code

```
for (j=0;j<4;j++)
  for (i=0;i<4;i++)
    a[i][j]=2*a[i][j];
```

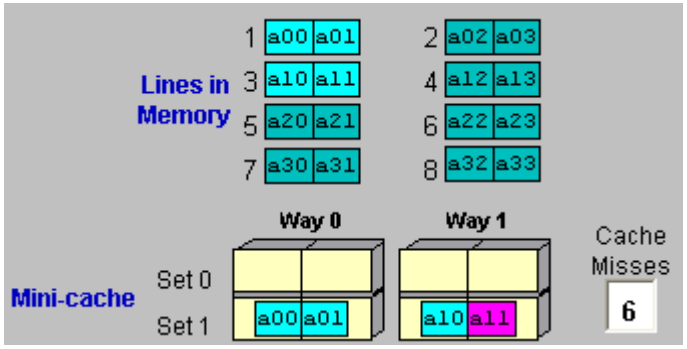
Iteration a[0][1]=2*a[0][1]




Original Code

```
for (j=0;j<4;j++)
  for (i=0;i<4;i++)
    a[i][j]=2*a[i][j];
```

Iteration a[1][1]=2*a[1][1]

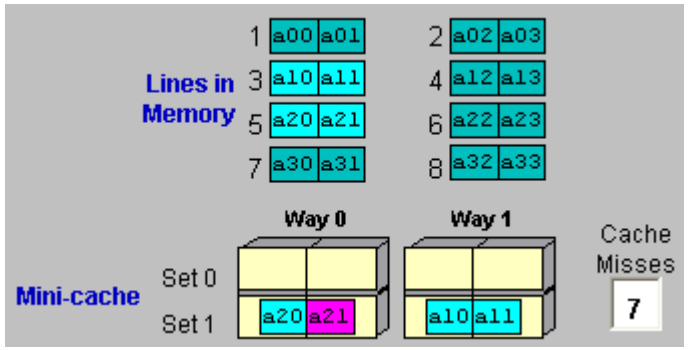


# Ejemplo : intercambio de bucles

Original Code

```
for (j=0;j<4;j++)
  for (i=0;i<4;i++)
    a[i][j]=2*a[i][j];
```

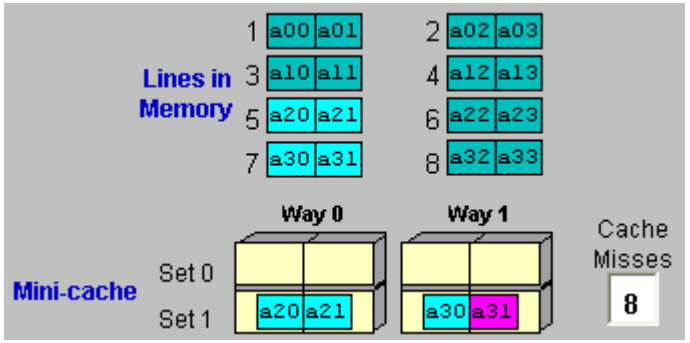
Iteration a[2][1]=2*a[2][1]



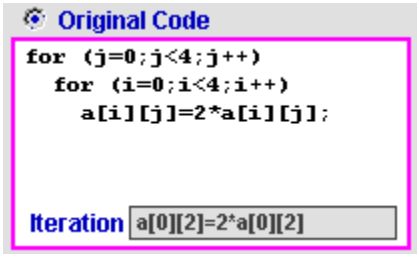

Original Code

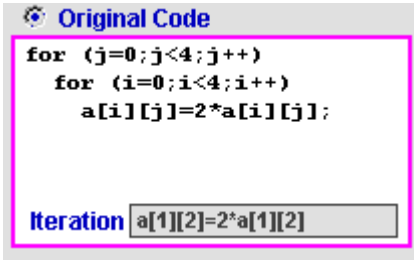
```
for (j=0;j<4;j++)
  for (i=0;i<4;i++)
    a[i][j]=2*a[i][j];
```

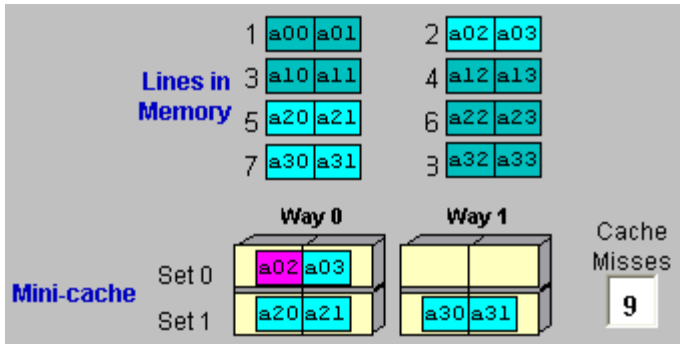
Iteration a[3][1]=2*a[3][1]

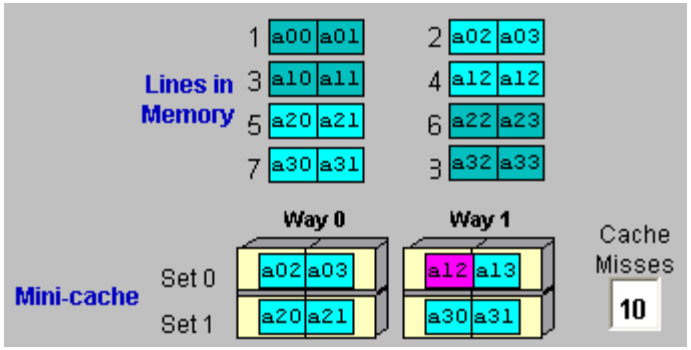


# Ejemplo : intercambio de bucles




Adelantamos varias iteraciones hasta la última de ellas …

# Ejemplo : intercambio de bucles

**Original Code**

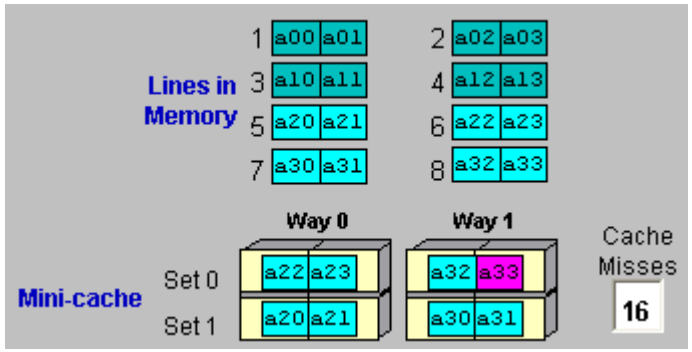
```
for (j=0;j<4;j++)
  for (i=0;i<4;i++)
    a[i][j]=2*a[i][j];
```

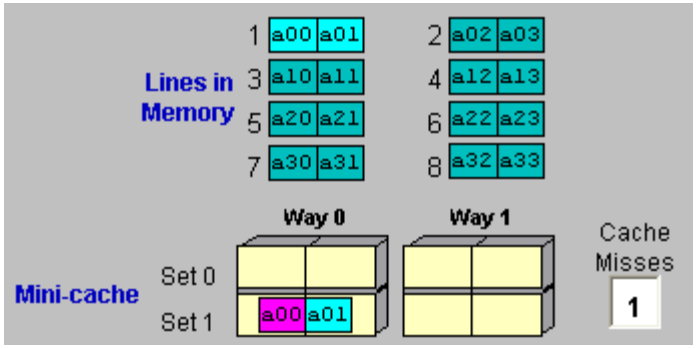
Iteration: a[3][3]=2*a[3][3]

**Optimized Code**

```
for (i=0;i<4;i++)
  for (j=0;j<4;j++)
    a[i][j]=2*a[i][j];
```

Iteration: a[0][0]=2*a[0][0]



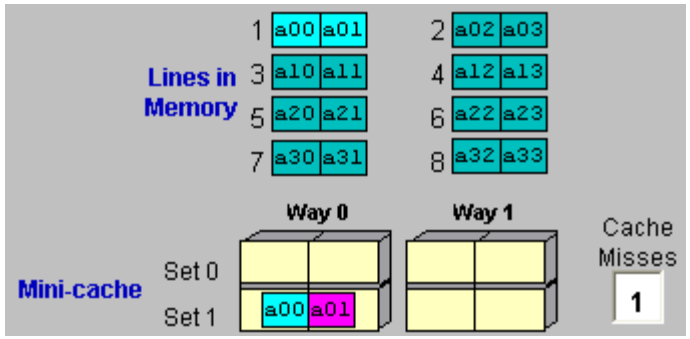



# Ejemplo : intercambio de bucles

Optimized Code

```
for (i=0;i<4;i++)
  for (j=0;j<4;j++)
    a[i][j]=2*a[i][j];
```

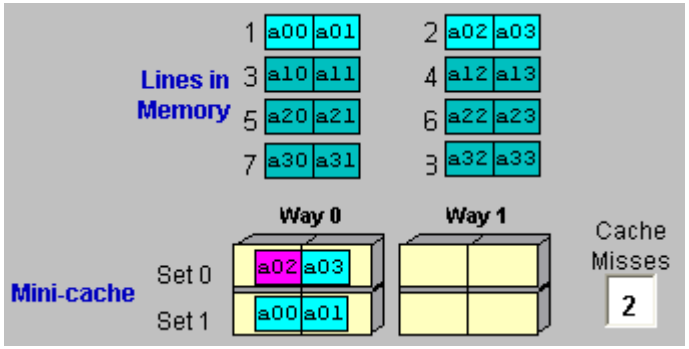
Iteration a[0][1]=2*a[0][1]




Optimized Code

```
for (i=0;i<4;i++)
  for (j=0;j<4;j++)
    a[i][j]=2*a[i][j];
```

Iteration a[0][2]=2*a[0][2]



# Ejemplo : intercambio de bucles

Optimized Code

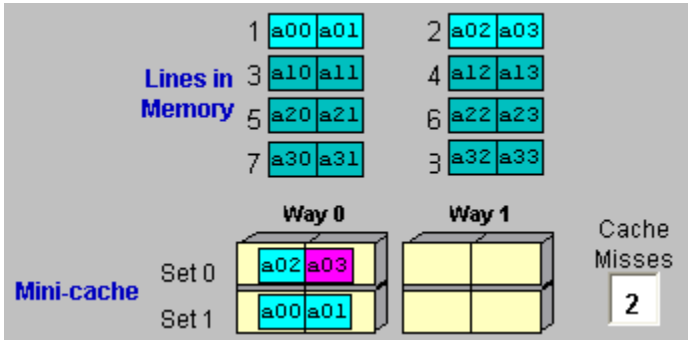
```
for (i=0;i<4;i++)
  for (j=0;j<4;j++)
    a[i][j]=2*a[i][j];
```

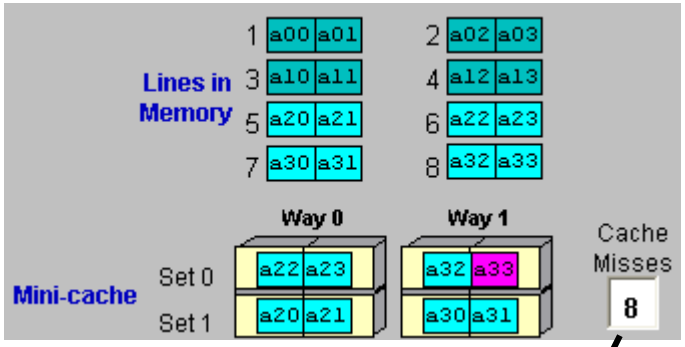
Iteration a[0][3]=2*a[0][3]

Optimized Code

```
for (i=0;i<4;i++)
  for (j=0;j<4;j++)
    a[i][j]=2*a[i][j];
```

Iteration a[3][3]=2*a[3][3]



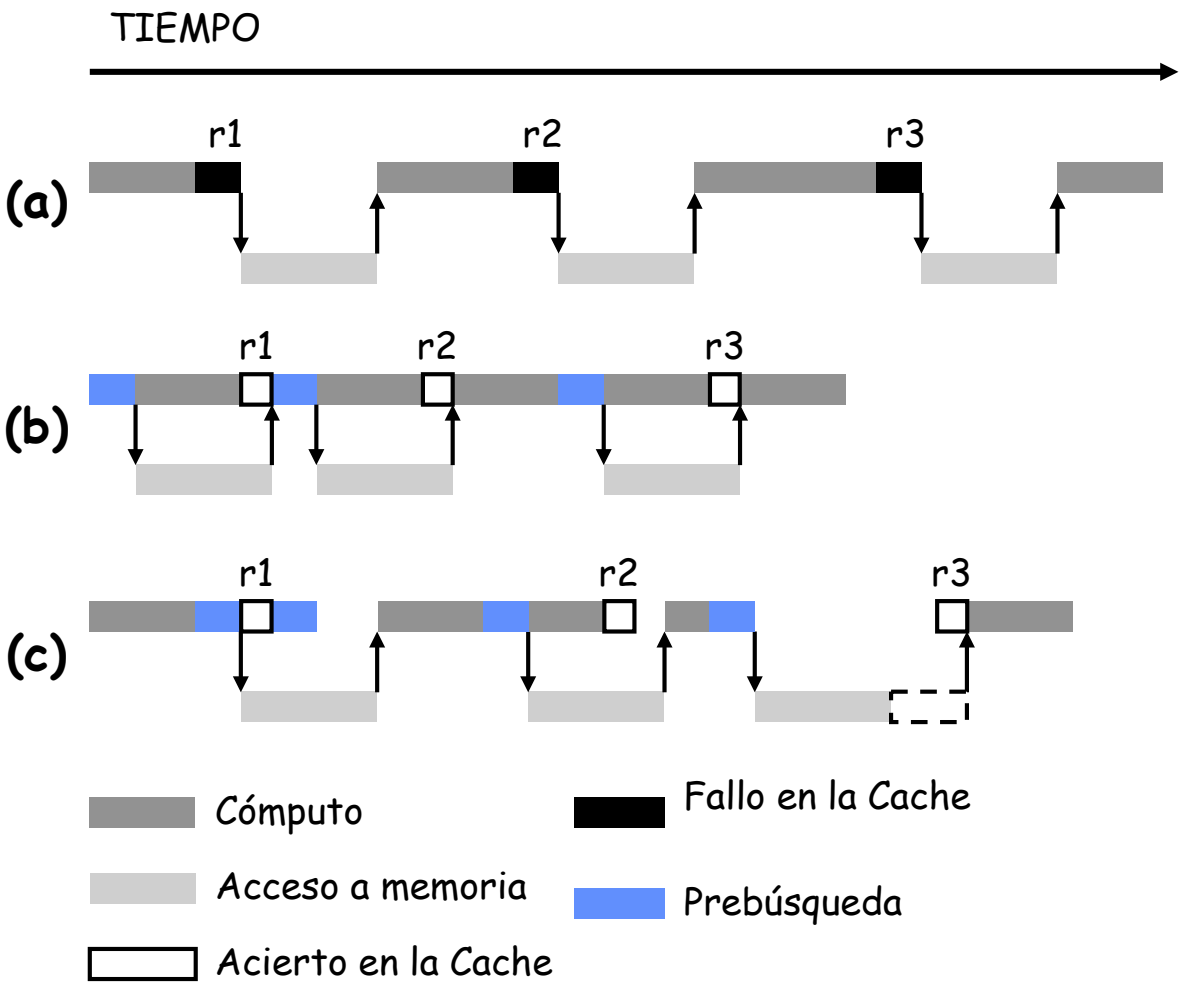




Reducción del 50% de fallos de cache

Adelantamos varias iteraciones hasta la última de ellas …

# *Unificación de Bucles*: aprovecha la localidad temporal

# Concepto de Prebúsqueda



```
for (i = 0; i < N; i++)
      ip = ip + a[i]*b[i];
```

(a)

# Prebúsqueda Software

# Ejemplo 1

```
.org 0x0000
_start:  movia r2, A
         movia r3, B
         movia r4, N
         ldw r4, 0(r4)
         add r5, r0, r0
LOOP:    ldw r6, 0(r2)
         ldw r7, 0(r3)
         mul r8, r6, r7
         add r5, r5, r8
         addi r2, r2, 4
         addi r3, r3, 4
         subi r4, r4, 1
         bgt r4, r0, LOOP
         stw r5, D(r0)
STOP:    br STOP
```

Índices de los 4 bloques de 16 bytes en la memoria cache de instrucciones de correspondencia directa

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |

```
.org 0x1000
N:       .word 20 /* número de elementos*/
A:       .word 1,2,3,4,5,6,7,8,9,…,19,20 /* elementos del vector A */
B:       .word 20,19,18,17,16,15,…,2,1 /* elementos del vector B */
D:       .skip 4
```

# Ejemplo 1

| DIRECCIONES iniciales palabra | | |
|---|---|---|
| | .org 0x0000 | |
| 0x0000 | _start: | movia r2, A |
| 0x0004 | | movia r3, B |
| 0x0008 | | movia r4, N |
| 0x000C | | ldw r4, 0(r4) |
| 0x0010 | | add r5, r0, r0 |
| 0x0014 | LOOP: | ldw r6, 0(r2) |
| 0x0018 | | ldw r7, 0(r3) |
| 0x001C | | mul r8, r6, r7 |
| 0x0020 | | add r5, r5, r8 |
| 0x0024 | | addi r2, r2, 4 |
| 0x0028 | | addi r3, r3, 4 |
| 0x002C | | subi r4, r4, 1 |
| 0x0030 | | bgt r4, r0, LOOP |
| 0x0034 | | stw r5, D(r0) |
| 0x0038 | STOP: | br STOP |

# Ejemplo 1

| DIRECCIONES iniciales palabra | | |
|---|---|---|
| | .org 0x0000 | |
| 0x0000 | _start: | movia r2, A |
| 0x0004 | | movia r3, B |
| 0x0008 | | movia r4, N |
| 0x000C | | ldw r4, 0(r4) |
| 0x0010 | | add r5, r0, r0 |
| 0x0014 | LOOP: | ldw r6, 0(r2) |
| 0x0018 | | ldw r7, 0(r3) |
| 0x001C | | mul r8, r6, r7 |
| 0x0020 | | add r5, r5, r8 |
| 0x0024 | | addi r2, r2, 4 |
| 0x0028 | | addi r3, r3, 4 |
| 0x002C | | subi r4, r4, 1 |
| 0x0030 | | bgt r4, r0, LOOP |
| 0x0034 | | stw r5, D(r0) |
| 0x0038 | STOP: | br STOP |

Dirección termina en bits
…xx xx00 xxxx

Índice

Accesos a la cache = 4

Fallos de la cache = 1

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | 0x00 movia | 0x04 movia | 0x08 movia | 0x0C ldw |
| 1 | | | | |
| 2 | Fallo | Acierto | Acierto | Acierto |
| 3 | | | | |

# Ejemplo 1

```
DIRECCIONES
iniciales        .org 0x0000
0x0000    _start:  movia r2, A
0x0004             movia r3, B
0x0008             movia r4, N
0x000C             ldw r4, 0(r4)
0x0010             add r5, r0, r0
0x0014    LOOP:    ldw r6, 0(r2)
0x0018             ldw r7, 0(r3)
0x001C             mul r8, r6, r7
0x0020             add r5, r5, r8
0x0024             addi r2, r2, 4
0x0028             addi r3, r3, 4
0x002C             subi r4, r4, 1
0x0030             bgt r4, r0, LOOP
0x0034             stw r5, D(r0)
0x0038    STOP:    br STOP
```

Dirección termina en bits
...xx xx01 xxxx

Índice

Accesos a la cache = 8

Fallos de la cache = 2

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | 0x00 movia | 0x04 movia | 0x08 movia | 0x0C ldw |
| 1 | 0x10 add | 0x14 ldw | 0x18 ldw | 0x1C mul |
| 2 | | | | |
| 3 | | | | |

Fallo Acierto Acierto Acierto

# Ejemplo 1

| DIRECCIONES iniciales | | |
|---|---|---|
| | .org 0x0000 | |
| 0x0000 | _start: | movia r2, A |
| 0x0004 | | movia r3, B |
| 0x0008 | | movia r4, N |
| 0x000C | | ldw r4, 0(r4) |
| 0x0010 | | add r5, r0, r0 |
| 0x0014 | LOOP: | ldw r6, 0(r2) |
| 0x0018 | | ldw r7, 0(r3) |
| 0x001C | | mul r8, r6, r7 |
| 0x0020 | | add r5, r5, r8 |
| 0x0024 | | addi r2, r2, 4 |
| 0x0028 | | addi r3, r3, 4 |
| 0x002C | | subi r4, r4, 1 |
| 0x0030 | | bgt r4, r0, LOOP |
| 0x0034 | | stw r5, D(r0) |
| 0x0038 | STOP: | br STOP |

Dirección termina en bits
…xx xx10 xxxx

Índice

Accesos a la cache = 12

Fallos de la cache = 3

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | 0x00 movia | 0x04 movia | 0x08 movia | 0x0C ldw |
| 1 | 0x10 add | 0x14 ldw | 0x18 ldw | 0x1C mul |
| 2 | 0x20 add | 0x24 addi | 0x28 addi | 0x2C subi |
| 3 | Fallo | Acierto | Acierto | Acierto |

# Ejemplo 1

| DIRECCIONES iniciales | | |
|---|---|---|
| | .org 0x0000 | |
| 0x0000 | _start: | movia r2, A |
| 0x0004 | | movia r3, B |
| 0x0008 | | movia r4, N |
| 0x000C | | ldw r4, 0(r4) |
| 0x0010 | | add r5, r0, r0 |
| 0x0014 | LOOP: | ldw r6, 0(r2) |
| 0x0018 | | ldw r7, 0(r3) |
| 0x001C | | mul r8, r6, r7 |
| 0x0020 | | add r5, r5, r8 |
| 0x0024 | | addi r2, r2, 4 |
| 0x0028 | | addi r3, r3, 4 |
| 0x002C | | subi r4, r4, 1 |
| 0x0030 | | bgt r4, r0, LOOP |
| 0x0034 | | stw r5, D(r0) |
| 0x0038 | STOP: | br STOP |

Dirección termina en bits
...xx xx11 xxxx

Índice

Accesos a la cache = 13

Fallos de la cache = 4

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | 0x00 movia | 0x04 movia | 0x08 movia | 0x0C ldw |
| 1 | 0x10 add | 0x14 ldw | 0x18 ldw | 0x1C mul |
| 2 | 0x20 add | 0x24 addi | 0x28 addi | 0x2C subi |
| 3 | 0x30 bgt | 0x34 stw | 0x38 br | 0x3C &%$# |

Fallo

# Ejemplo 1

| DIRECCIONES iniciales | | |
|---|---|---|
| | .org 0x0000 | |
| 0x0000 | _start: | movia r2, A |
| 0x0004 | | movia r3, B |
| 0x0008 | | movia r4, N |
| 0x000C | | ldw r4, 0(r4) |
| 0x0010 | | add r5, r0, r0 |
| 0x0014 | LOOP: | ldw r6, 0(r2) |
| 0x0018 | | ldw r7, 0(r3) |
| 0x001C | | mul r8, r6, r7 |
| 0x0020 | | add r5, r5, r8 |
| 0x0024 | | addi r2, r2, 4 |
| 0x0028 | | addi r3, r3, 4 |
| 0x002C | | subi r4, r4, 1 |
| 0x0030 | | bgt r4, r0, LOOP |
| 0x0034 | | stw r5, D(r0) |
| 0x0038 | STOP: | br STOP |

Instrucciones = 8

Accesos a la cache = 21

Fallos de la cache = 4

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | 0x00 movia | 0x04 movia | 0x08 movia | 0x0C ldw |
| 1 | 0x10 add | 0x14 ldw | 0x18 ldw | 0x1C mul |
| 2 | 0x20 add | 0x24 addi | 0x28 addi | 0x2C subi |
| 3 | 0x30 bgt | 0x34 stw | 0x38 br | 0x3C &%$# |

# Ejemplo 1

| DIRECCIONES iniciales | | |
|---|---|---|
| | .org 0x0000 | |
| 0x0000 | _start: | movia r2, A |
| 0x0004 | | movia r3, B |
| 0x0008 | | movia r4, N |
| 0x000C | | ldw r4, 0(r4) |
| 0x0010 | | add r5, r0, r0 |
| 0x0014 | LOOP: | ldw r6, 0(r2) |
| 0x0018 | | ldw r7, 0(r3) |
| 0x001C | | mul r8, r6, r7 |
| 0x0020 | | add r5, r5, r8 |
| 0x0024 | | addi r2, r2, 4 |
| 0x0028 | | addi r3, r3, 4 |
| 0x002C | | subi r4, r4, 1 |
| 0x0030 | | bgt r4, r0, LOOP |
| 0x0034 | | stw r5, D(r0) |
| 0x0038 | STOP: | br STOP |

Accesos a la cache = 165

Fallos de la cache = 4

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | 0x00 movia | 0x04 movia | 0x08 movia | 0x0C ldw |
| 1 | 0x10 add | 0x14 ldw | 0x18 ldw | 0x1C mul |
| 2 | 0x20 add | 0x24 addi | 0x28 addi | 0x2C subi |
| 3 | 0x30 bgt | 0x34 stw | 0x38 br | 0x3C &%$# |

# Ejemplo 1

Accesos a la cache = 167

Fallos de la cache = 4

Frecuencia Fallos

FF = 100% x 4 / 167 = 2,4%

| DIRECCIONES iniciales | | |
|---|---|---|
| 0x0000 | | .org 0x0000 |
| 0x0000 | _start: | movia r2, A |
| 0x0004 | | movia r3, B |
| 0x0008 | | movia r4, N |
| 0x000C | | ldw r4, 0(r4) |
| 0x0010 | | add r5, r0, r0 |
| 0x0014 | LOOP: | ldw r6, 0(r2) |
| 0x0018 | | ldw r7, 0(r3) |
| 0x001C | | mul r8, r6, r7 |
| 0x0020 | | add r5, r5, r8 |
| 0x0024 | | addi r2, r2, 4 |
| 0x0028 | | addi r3, r3, 4 |
| 0x002C | | subi r4, r4, 1 |
| 0x0030 | | bgt r4, r0, LOOP |
| 0x0034 | | stw r5, D(r0) |
| 0x0038 | STOP: | br STOP |

| Índice | Palabra0 | Palabra1 | Palabra2 | Palabra3 |
|---|---|---|---|---|
| 0 | 0x00 movia | 0x04 movia | 0x08 movia | 0x0C ldw |
| 1 | 0x10 add | 0x14 ldw | 0x18 ldw | 0x1C mul |
| 2 | 0x20 add | 0x24 addi | 0x28 addi | 0x2C subi |
| 3 | 0x30 bgt | 0x34 stw | 0x38 br | 0x3C &%$# |

# Ejemplo 2: 4 bloques de 8 bytes

Contención o lucha por usar el bloque de datos durante bucle

| Índice | Palabra0 | Palabra1 |
|---|---|---|
| 0 | 0x00, 0x20 | 0x04, 0x24 |
| 1 | 0x08, 0x28 | 0x0C, 0x2C |
| 2 | 0x10, 0x30 | 0x14, 0x34 |
| 3 | 0x18, 0x38 | 0x1C, 0x3C |

Accesos a la cache = 167

Fallos de la cache = 46

Frecuencia Fallos
FF = 100% x 46 / 167 = 27,5%