

# Arquitectura e Ingeniería de Computadores



## Tema 5

Jerarquía de memoria: Cache,  
reducción de fallos, ocultación de latencia,  
memoria principal

**D**EPARTAMENTO DE  
**A**RQUITECTURA DE **C**OMPUTADORES  
Y **A**UTOMÁTICA

Curso 2009-2010

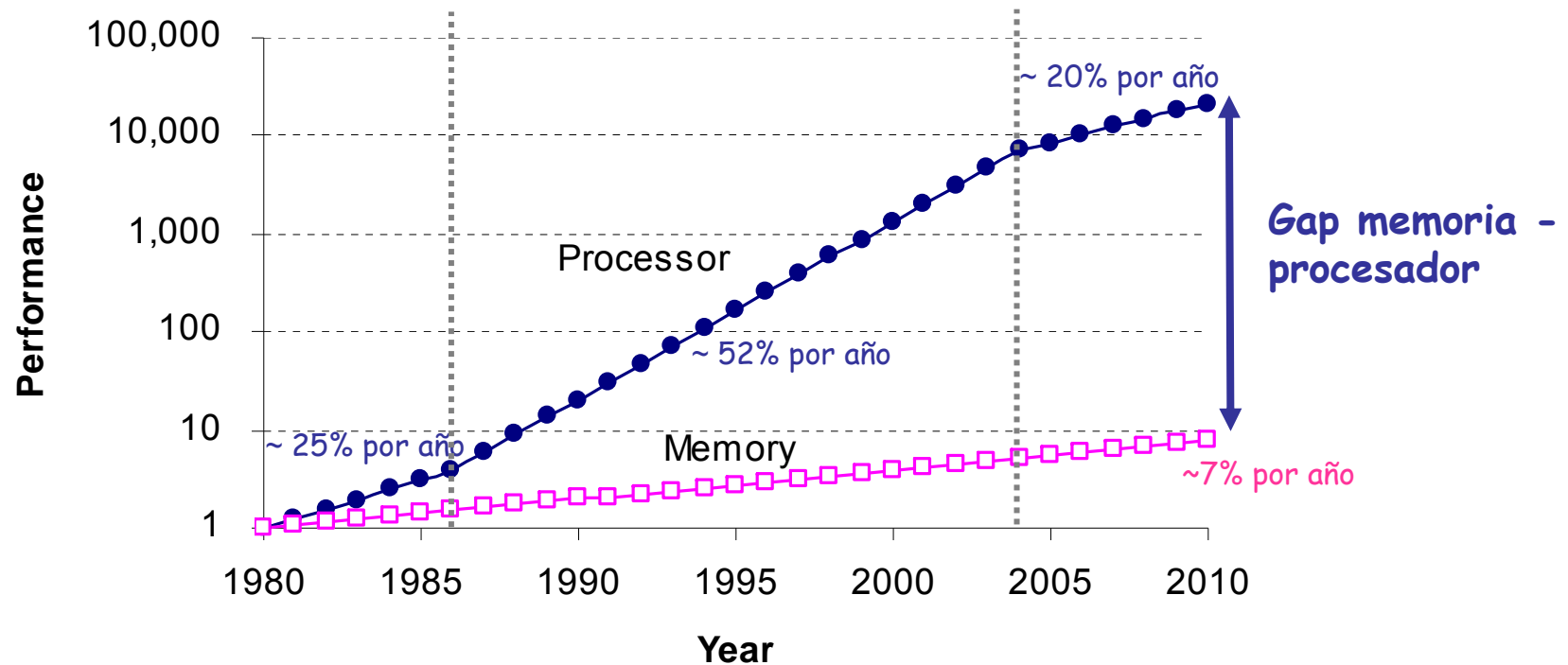
# Contenidos

---

- o Introducción: Jerarquía de memoria
- o Memoria cache: Evolución y repaso de conceptos básicos
- o Rendimiento de la memoria cache
- o Optimización de la memoria cache
  - o Reducción de la tasa de fallos de la cache
  - o Reducción de la penalización de los fallos de cache
  - o Reducción del tiempo de acierto
  - o Aumento del ancho de banda
- o La memoria principal
- o Una visión global: AMD Opteron
- o Bibliografía
  - o Capítulo 5 de Hennessy & Patterson (4th ed.)
  - o Apéndice C de Hennessy & Patterson (4th ed.)

# Introducción

## □ El problema

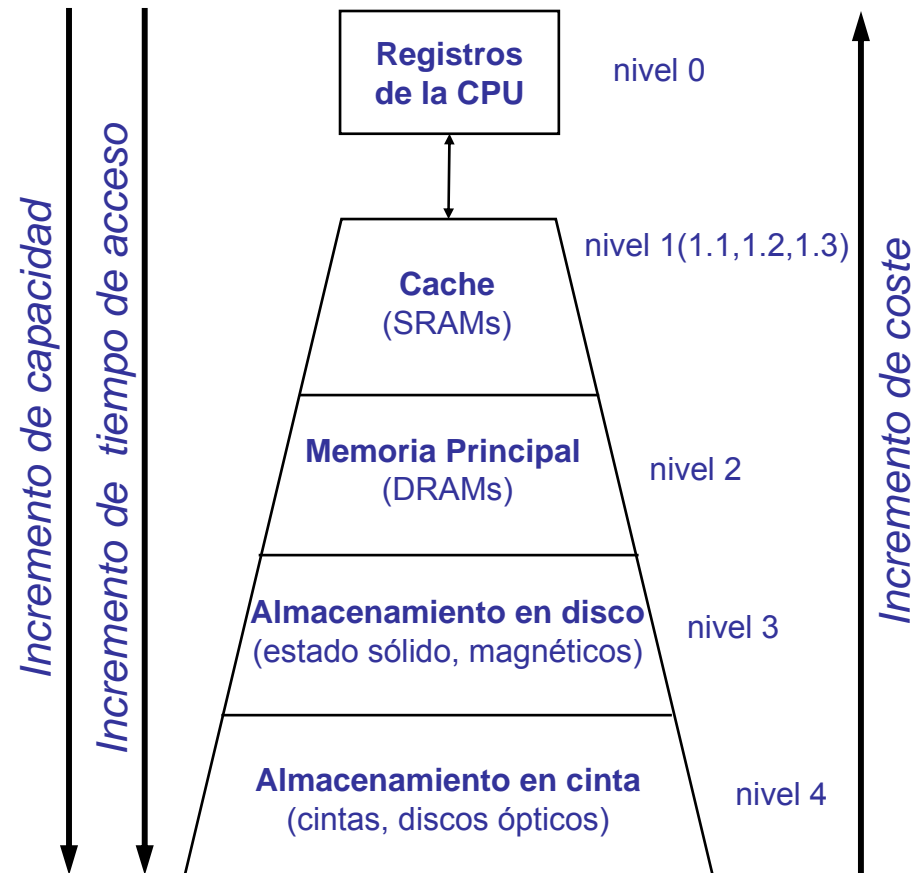


- La demanda de anchura de banda con memoria crece.
  - Segmentación, ILP
  - Ejecución especulativa
  - 1980 no caches “on chip”, 2007 2-3 niveles de cache “on chip”

# Introducción

## □ Niveles de la Jerarquía de memoria

- Un computador típico está formado por diversos niveles de memoria, *organizados de forma jerárquica*:
  - ⇒ Registros de la CPU
  - ⇒ Memoria Cache
  - ⇒ Memoria Principal
  - ⇒ Memoria Secundaria (discos)
  - ⇒ Unidades de Cinta (Back-up) y Dispositivos Ópticos
- El coste de todo el sistema de memoria excede al coste de la CPU
  - ⇒ Es muy importante optimizar su uso



# Introducción

## ❑ Tipos de Memoria

- Valores típicos para una WS o pequeño servidor (2006)

Tipo	Tamaño	Tecnología	T acceso	Ancho de banda	Costo/bit
Registros	< 1KB	Custom, CMOS	0,25-0,5 ns	50-500 GB/s	\$\$\$\$
Cache	< 16 MB	CMOS SRAM	0,5- 25 ns	5-20 GB/s	\$\$\$
Memoria principal	< 512 GB	CMOS DRAM	50-250 ns	2,5-10 GB/s	\$\$
Disco	> 1 TB	Magnética	5 ms	50-500 MB/s	~0

# Introducción

---

## ❑ Objetivo de la gestión de la jerarquía de memoria

- Optimizar el uso de la memoria
- Hacer que el usuario tenga la ilusión de que dispone de una memoria con:
  - ⇒ *Tiempo de acceso similar al del sistema más rápido*
  - ⇒ *Coste por bit similar al del sistema más barato*
- Para la mayor parte de los accesos a un bloque de información, este bloque debe encontrarse en los niveles bajos de la jerarquía de memoria

## ❑ Niveles a los que afecta la gestión de la jerarquía memoria

- Se refiere a la gestión dinámica, en tiempo de ejecución de la jerarquía de memoria
- Esta gestión de la memoria sólo afecta a los niveles 1 (cache), 2 (mem. principal) y 3 (mem. secund.)
  - ⇒ El nivel 0 (registros) lo asigna el compilador en tiempo de compilación
  - ⇒ El nivel 4 (cintas y discos ópticos) se utiliza para copias de seguridad (back-up)

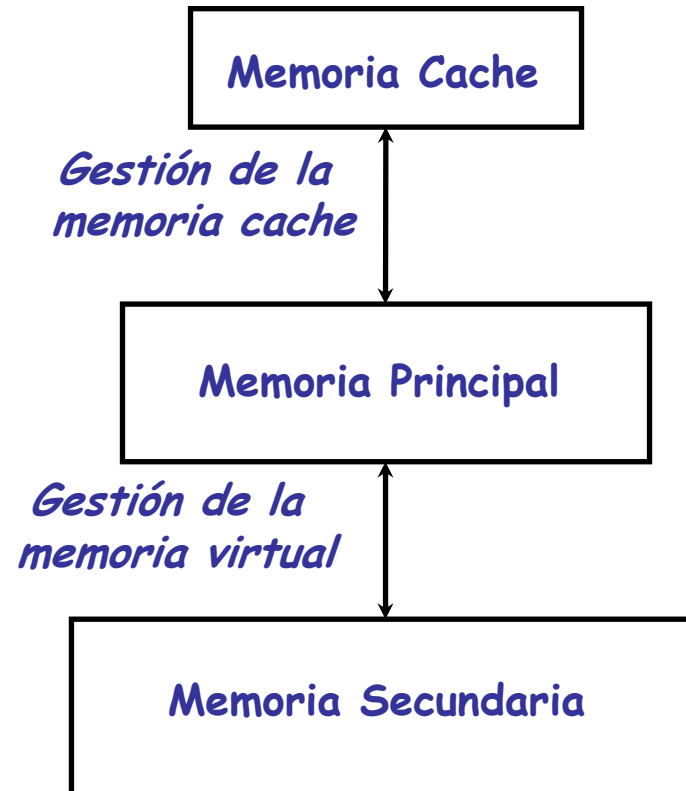
# Introducción

## ❑ Gestión de la memoria cache

- o Controla la transferencia de información entre la memoria cache y la memoria principal
- o Suele llevarse a cabo mediante Hardware específico (MMU o "Management Memory Unit")

## ❑ Gestión de la memoria virtual

- o Controla la transferencia de información entre la memoria principal y la memoria secundaria
- o Parte de esta gestión se realiza mediante hardware específico (MMU) y otra parte la realiza el S.O



# Introducción

## □ Propiedades de la jerarquía de memoria

### ➤ Inclusión

- o Cualquier información almacenada en el nivel de memoria  $M_i$ , debe encontrarse también en los niveles  $M_{i+1}$ ,  $M_{i+2}$ , ...,  $M_n$ . Es decir:  $M_1 \subset M_2 \subset \dots \subset M_n$

### ➤ Coherencia

- o Las copias de la misma información existentes en los distintos niveles deben ser coherentes
  - ⇒ Si un bloque de información se modifica en el nivel  $M_i$ , deben actualizarse los niveles  $M_{i+1}, \dots, M_n$

### ➤ Localidad

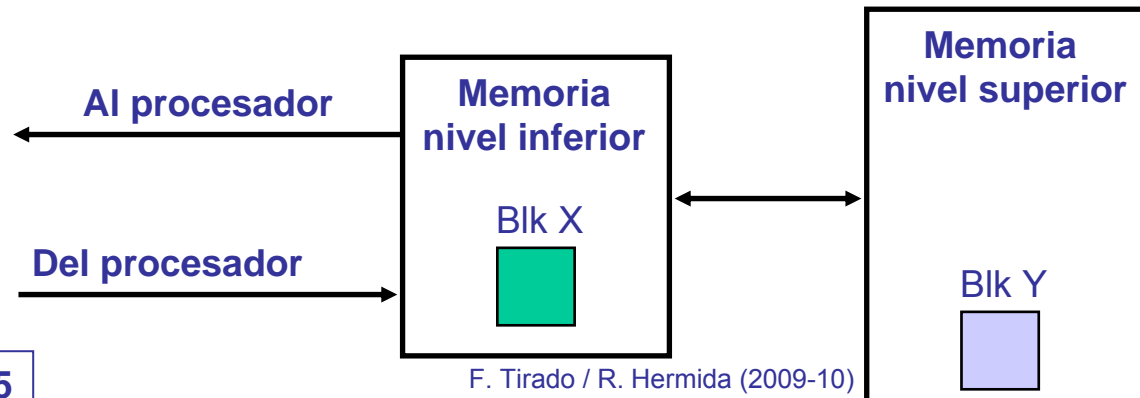
- o *Las referencias a memoria generadas por la CPU, para acceso a datos o a instrucciones, están concentradas o agrupadas en ciertas regiones del tiempo y del espacio*
- o **Localidad temporal**
  - ⇒ Las direcciones de memoria (instrucciones o datos) recientemente referenciadas, serán referenciadas de nuevo, muy probablemente, en un futuro próximo
  - ⇒ Ejemplos: Bucles, subrutinas, accesos a pila, variables temporales, etc.
- o **Localidad espacial**
  - ⇒ Tendencia a referenciar elementos de memoria (datos o instrucc.) cercanos a los últimos elementos referenciados
  - ⇒ Ejemplos: programas secuenciales, arrays, variables locales de subrutinas, etc.



# Introducción

## □ Terminología

- **Bloque:** unidad mínima de transferencia entre dos niveles
  - o En cache es habitual llamarle "línea"
- **Acierto (hit):** el dato solicitado está en el nivel  $i$ 
  - o Tasa de aciertos (hit ratio): la fracción de accesos encontrados en el nivel  $i$
  - o Tiempo de acierto (hit time): tiempo detección de acierto + tiempo de acceso del nivel  $i$ . (Tiempo total invertido para obtener un dato cuando éste se encuentra en el nivel  $i$ )
- **Fallo (miss):** el dato solicitado no está en el nivel  $i$  y es necesario buscarlo en el nivel  $i+1$ 
  - o Tasa de fallos (miss ratio):  $1 - (\text{Tasa de aciertos})$
  - o Tiempo de penalización por fallo (miss penalty): tiempo invertido para mover un bloque del nivel  $i+1$  al nivel  $i$ , cuando el bloque referenciado no está en el nivel  $i$ .
- **Requisito:** Tiempo de acierto  $\ll$  Penalización de fallo

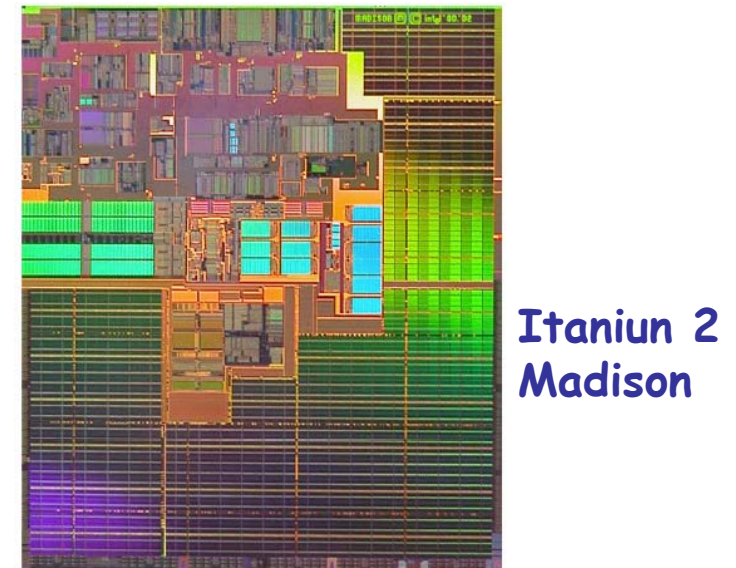


# Memoria cache: evolución

## □ Algunos datos

### □ Tamaño de la cache

Del 50 al 75 % del área. Más del 80% de los transistores



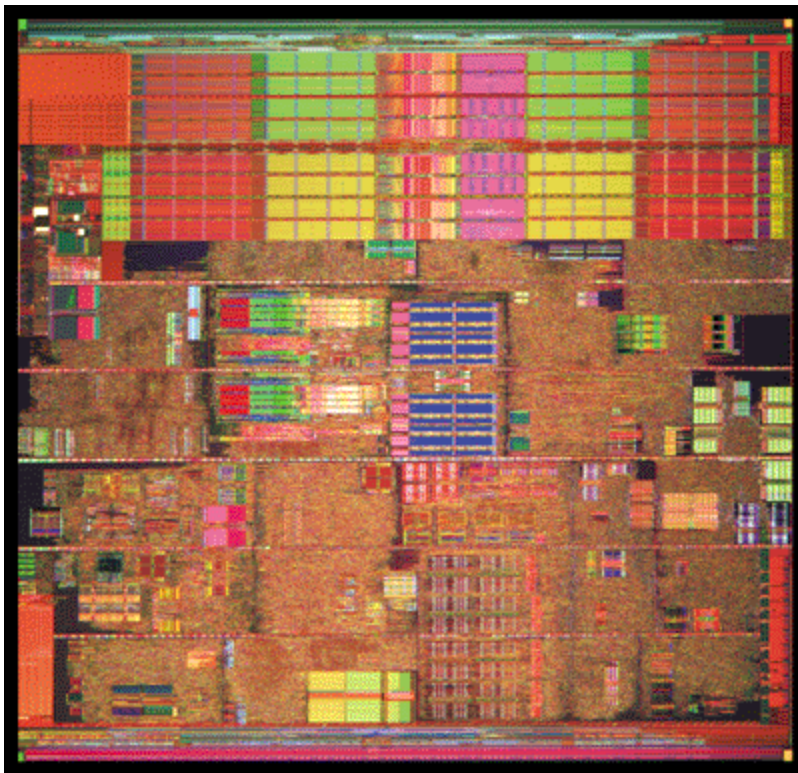
### □ Tiempo de servicio de un fallo de cache: evolución

21064 (200Mhz) 340ns,	$340/5=68$ ciclos,	$68 \times 2 = 136$ instrucciones
21164 (300Mhz) 266ns,	$266/3.3=80$ ,	$80 \times 4 = 320$ instrucciones
21264 (1Ghz) 180ns,	$180/1=180$ ,	$180 \times 6 = 1080$ instrucciones

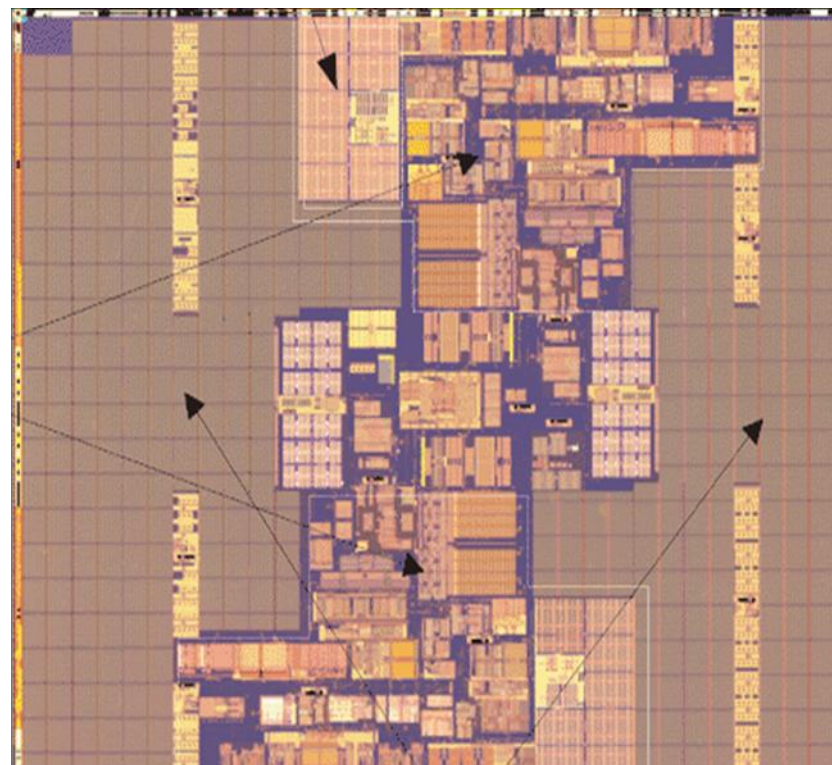
### □ Latencia:

1ciclo ( Itanium2) a 3 ciclos Power4-5

# Memoria cache: evolución



**Pentium 4**

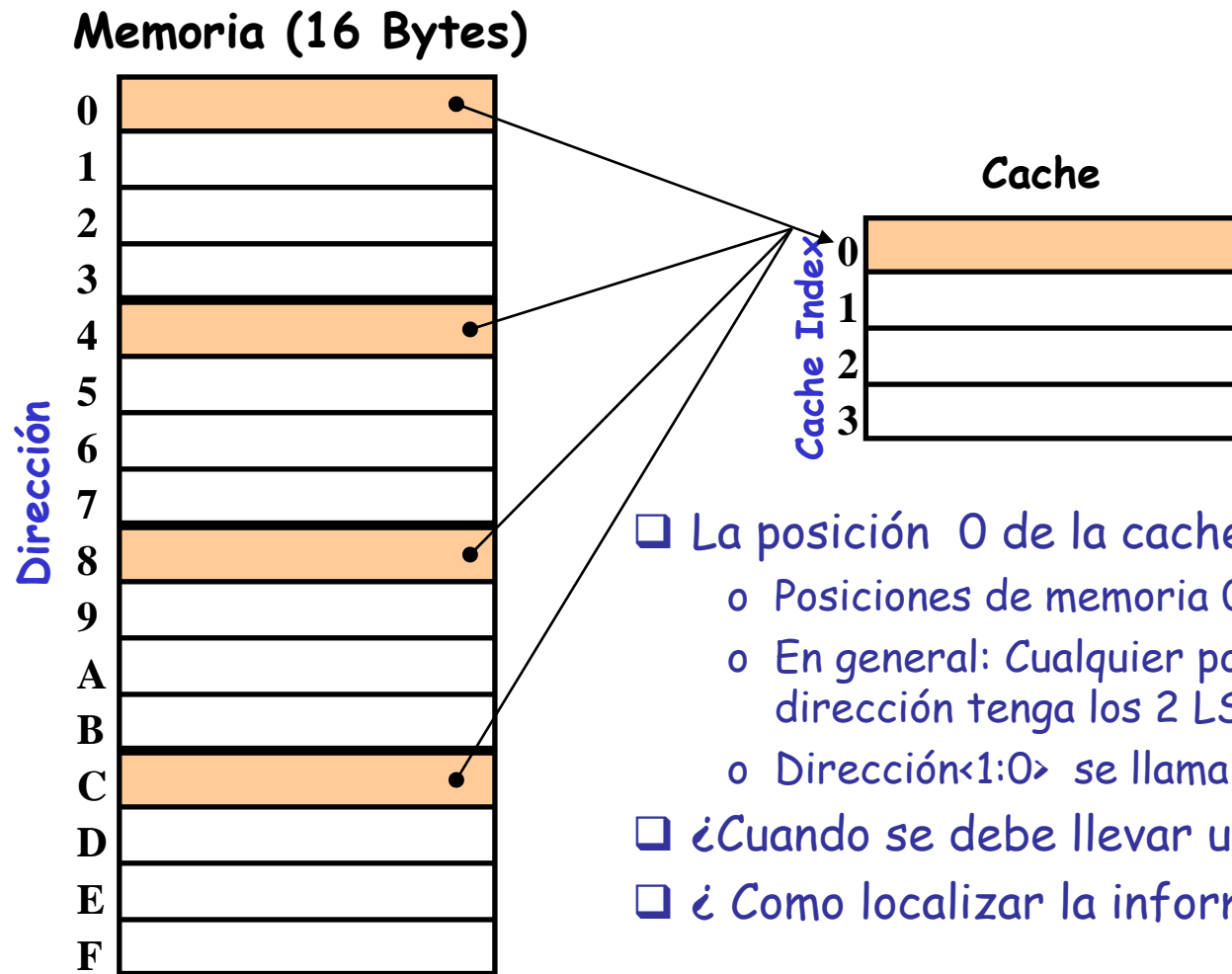


**Montecito**  
(1700Mtrans, 2 Itanium en 1 chip)

## Mc: repaso de conceptos básicos

### ❑ Política de emplazamiento: Emplazamiento directo

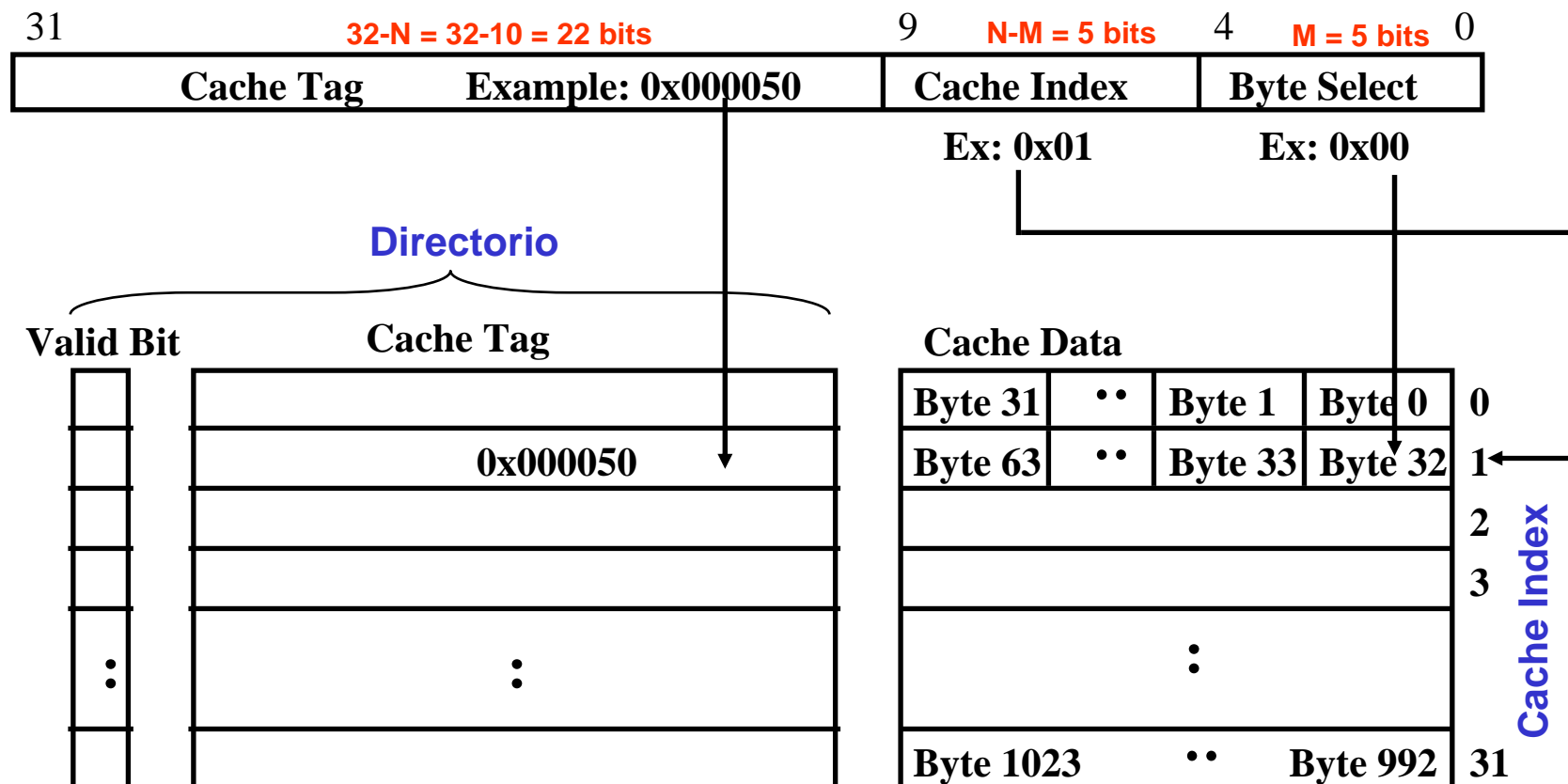
Un ejemplo trivial: Cache directa de 4 bytes, tamaño de bloque 1 byte



- ❑ La posición 0 de la cache almacenará los datos de:
  - o Posiciones de memoria 0, 4, 8, ... etc.
  - o En general: Cualquier posición de memoria cuya dirección tenga los 2 LSBs a 0
  - o Dirección<1:0> se llama "índice cache" (cache index)
- ❑ ¿Cuándo se debe llevar una información a la cache?
- ❑ ¿Como localizar la información en la cache ?

# Mc: repaso de conceptos básicos

- En general: Para una cache directa de  $2^N$  bytes con dirección de D bits:
  - o Los (D - N) bits más significativos de la dirección son el "Tag"
  - o Asumiendo un tamaño de bloque de  $2^M$  bytes, los M bits menos significativos son el selector de bytes
  - o Anchura de cache index = N-M bits
- Ejemplo: Cache directa de 1 KB ( $2^{10}$  bytes), 32 ( $2^5$ ) bytes por línea y dirección de 32 bits (N=10, M=5, Tag=32-10=22 bits, Cache index = N-M=5 bits)

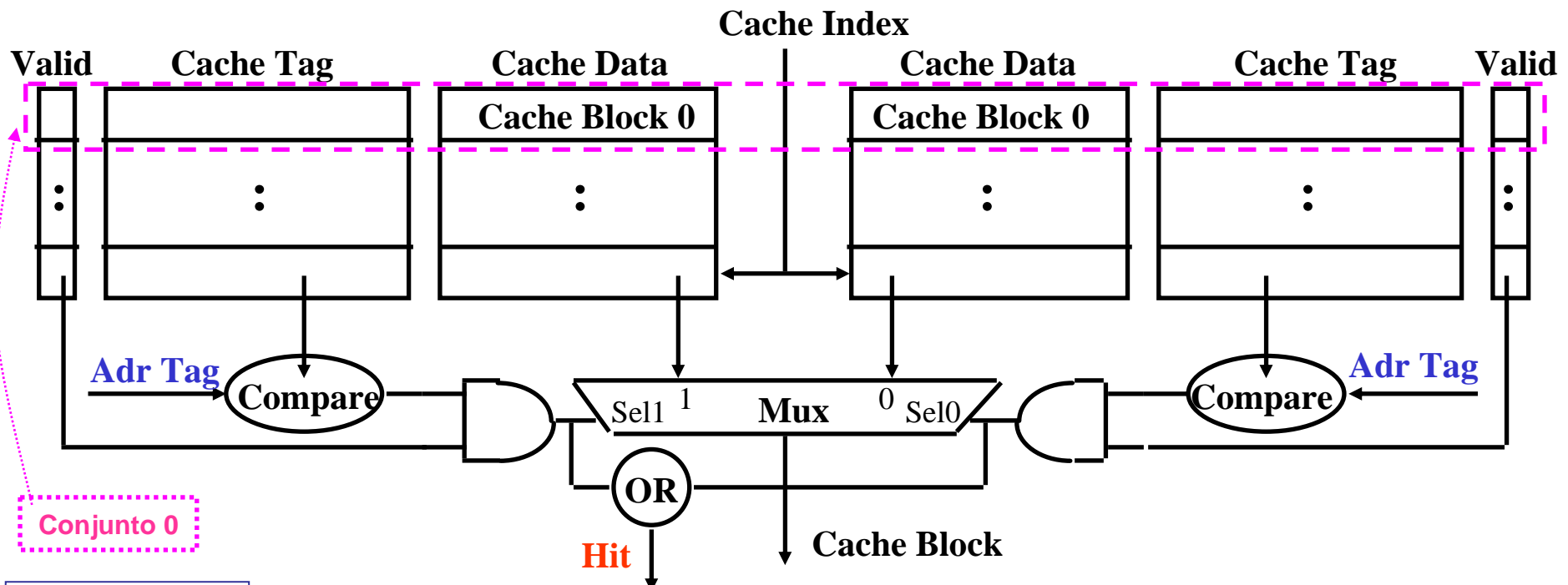




# Mc: repaso de conceptos básicos

## ❑ Emplazamiento asociativo por conjuntos

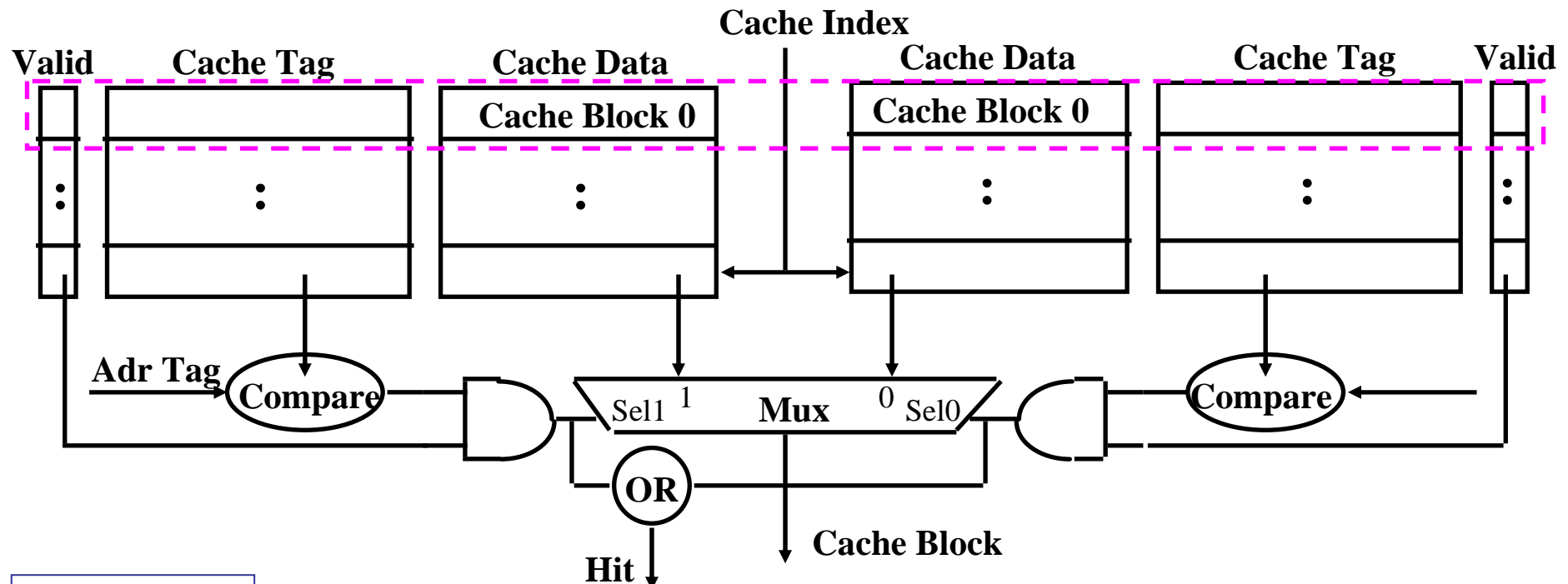
- ❑ Asociativa por conjuntos con N vías (N-way) : N líneas por conjunto
  - o N caches directas operando en paralelo (N típico 2, 4, ..)
- ❑ Ejemplo: cache asociativa por conjuntos con 2 vías
  - o Cache Index selecciona un conjunto de la cache
    - Anchura =  $\log_2(N^\circ \text{ conjuntos})$
  - o Los dos tags en el conjunto seleccionado son comparados en paralelo
  - o Se selecciona el dato en función de la comparación



# Mc: repaso de conceptos básicos

## ❑ Comparación

- ❑ Asociativa por conjuntos N-vías "versus" Directa
  - o N comparadores vs. 1
  - o Retardo extra por el MUX para los datos
  - o El dato llega después de haber verificado el éxito de alguna de las comparaciones (hit)
- ❑ En una directa, el bloque de cache es accesible antes del hit:
  - o Es posible asumir un acierto y continuar. Recuperación si fallo.



# Mc: repaso de conceptos básicos

---

## ❑ Política de actualización de Mp :Write-Through vs Write-Back

- ❑ Write-through: Todas las escrituras actualizan la cache y la memoria
  - o Al remplazar, se puede eliminar la copia de cache: Los datos están en la memoria
  - o Bit de control en la cache: Sólo un bit de validez
- ❑ Write-back: Todas las escrituras actualizan sólo la cache
  - o Al reemplazar, no se pueden eliminar los datos de la cache: Deben ser escritos primero en la memoria
  - o Bit de control: Bit de validez y bit de sucio
- ❑ Comparación:
  - o Write-through:
    - Memoria ( y otros lectores ) siempre tienen el último valor
    - Control simple
  - o Write-back:
    - Mucho menor AB, escrituras múltiples en bloque
    - Mejor tolerancia a la alta latencia de la memoria
- ❑ **Gestión de los fallos de escritura:**
  - o Write allocate (con asignación en escritura): en un fallo de escritura se lleva el bloque escrito a la cache
  - o No-write allocate (sin asignación en escritura): en un fallo de escritura el dato sólo se modifica en la Mp (o nivel de memoria siguiente)



## Mc: rendimiento

### ❑ Procesador con memoria perfecta (ideal)

- $T_{cpu} = N \times CPI \times tc$  ( $tc$  = tiempo de ciclo,  $N$  = nº de instrucciones)
- Como  $N \times CPI = N^\circ$  ciclos CPU  $\rightarrow T_{cpu} = N^\circ$  ciclos CPU  $\times tc$

### ❑ Procesador con memoria real

- $T_{cpu} = (N^\circ \text{ ciclos CPU} + N^\circ \text{ ciclos espera memoria}) \times tc$

o Cuántos ciclos de espera por la memoria?

- $N^\circ \text{ ciclos espera memoria} = N^\circ \text{ fallos} \times \text{Miss Penalty}$
- $N^\circ \text{ fallos} = N^\circ \text{ ref a memoria} \times \text{Miss Rate}$
- $N^\circ \text{ ref a memoria} = N \times AMPI$  ( $AMPI$  = Media de accesos a memoria por instr)

o Luego:

- $N^\circ \text{ ciclos espera memoria} = N \times AMPI \times \text{Miss Rate} \times \text{Miss Penalty}$

o Y finalmente

- $T_{cpu} = [ (N \times CPI) + (N \times AMPI \times \text{Miss Rate} \times \text{Miss Penalty}) ] \times tc$
- $T_{cpu} = N \times [ CPI + (AMPI \times \text{Miss Rate} \times \text{Miss Penalty}) ] \times tc$



Define el espacio de diseño para la optimización de Mc

## Mc: rendimiento

### ❑ Penalización media por instrucción

#### o Comparando

- $T_{cpu} = N \times CPI \times t_c$
- $T_{cpu} = N \times [ CPI + (AMPI \times \text{Miss Rate} \times \text{Miss Penalty}) ] \times t_c$

se pueden definir los ciclos de penalización media por instrucción debida al comportamiento de la memoria:

- Penalización media por instrucción =  
=  $AMPI \times \text{Miss Rate} \times \text{Miss Penalty}$

### ❑ Tiempo medio de acceso a memoria (TMAM)

- $TMAM = \text{Tiempo invertido en accesos a memoria} / N^\circ \text{ accesos} =$   
=  $(T \text{ de accesos a Mc} + T \text{ de penalización por fallos}) / N^\circ \text{ accesos} =$   
=  $\text{Hit time} + (N^\circ \text{ de fallos} \times \text{Miss Penalty} / N^\circ \text{ accesos})$
- $TMAM = \text{Hit time} + \text{Miss Rate} \times \text{Miss Penalty}$

# Mc: tipos de fallos

---

## □ Las 3 C's

### o Iniciales (compulsory)

- Causados por la 1ª referencia a un bloque que no está en la cache → Hay que llevar primero el bloque a la cache
- Inevitables, incluso con cache infinita
- No depende del tamaño de la Mc. Sí del tamaño de bloque.

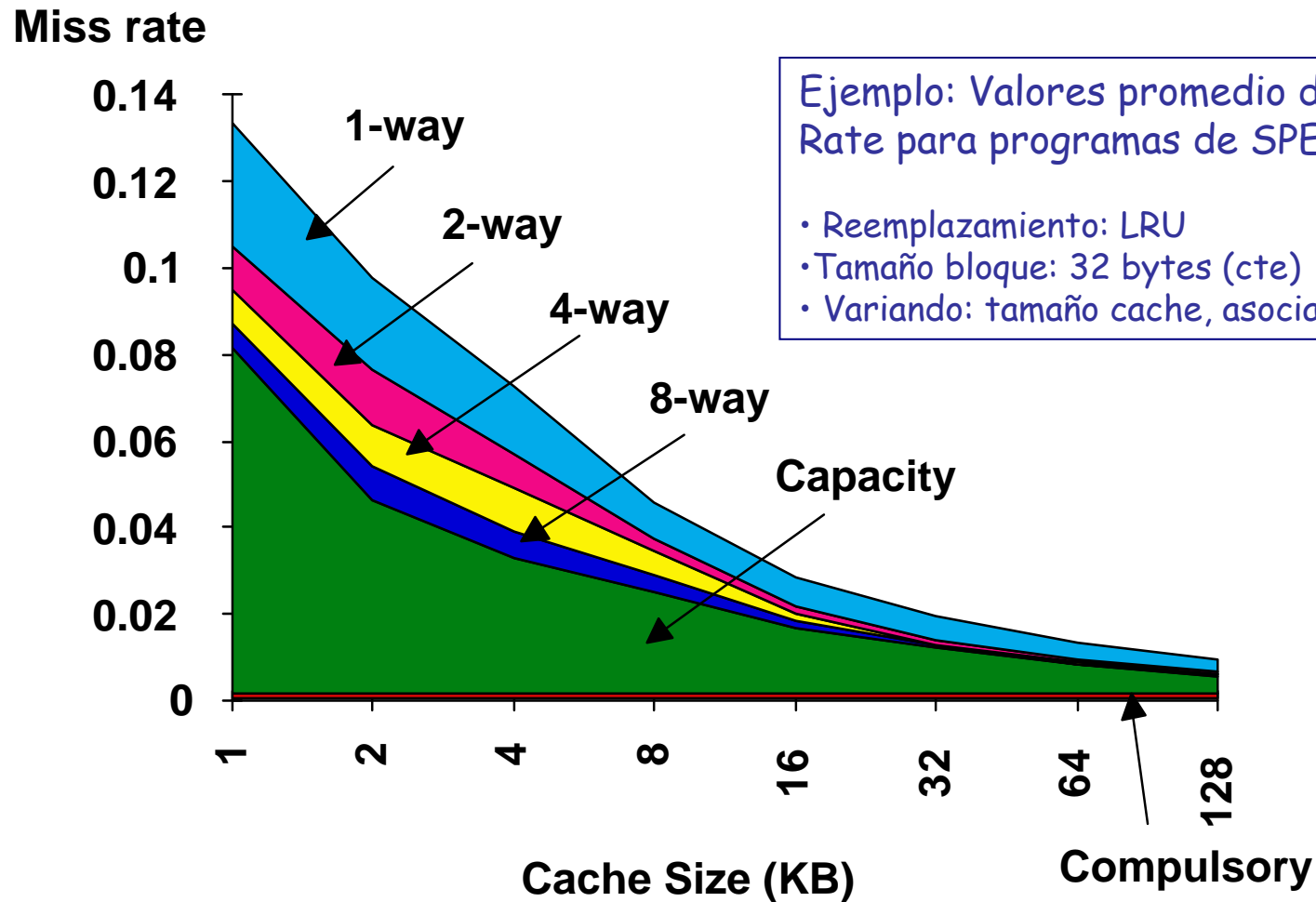
### o Capacidad

- Si la cache no puede contener todos los bloques necesarios durante la ejecución de un programa, habrá fallos que se producen al recuperar de nuevo un bloque previamente descartado

### o Conflicto

- Un bloque puede ser descartado y recuperado de nuevo porque hay otro bloque que compite por la misma línea de cache (aunque haya otras líneas libres en la cache)
- No se pueden producir en caches puramente asociativas.

## Mc: tipos de fallos



## Espacio de diseño para la mejora del rendimiento de Mc

---

### ❑ ¿ Como mejorar el rendimiento de la cache?

$$T_{cpu} = N \times [ CPI + (AMPI \times \text{Miss Rate} \times \text{Miss Penalty} ) ] \times t_c$$

### ❑ Estudio de técnicas para:

- o Reducir la tasa de fallos
- o Reducir la penalización del fallo
- o Reducir el tiempo de acierto (hit time)
- o Aumentar el ancho banda
  - Las dos últimas técnicas inciden sobre  $t_c$

## Espacio de diseño para la mejora del rendimiento de Mc

### ¿ Como mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Asociatividad	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Tamaño de Mc	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Algoritmo de reemplazamiento	Caches multinivel	Cache de trazas	
Cache de víctimas			
Optimización del código (compilador)			
Prebúsqueda HW			
Prebúsqueda SW			

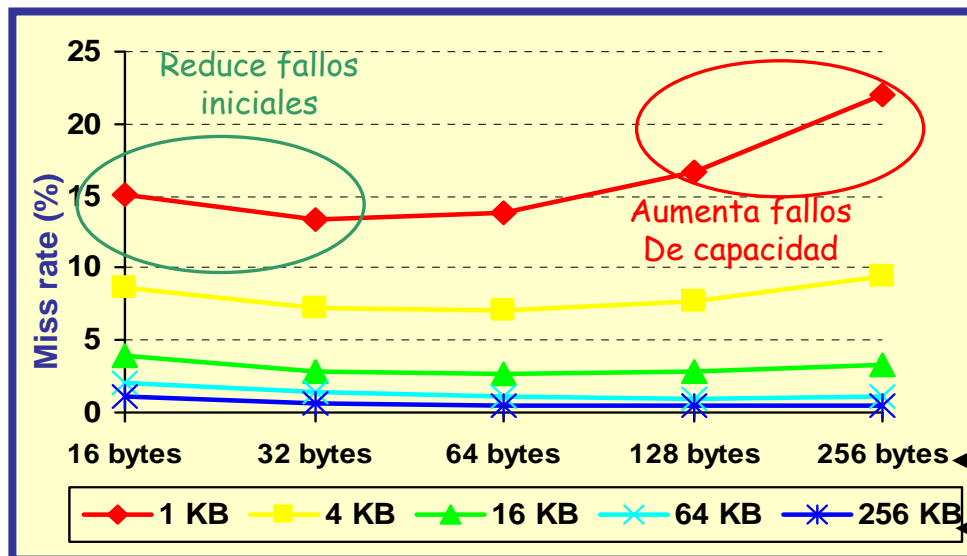
# Optimizaciones para reducir la tasa de fallos

## □ Aumento del el tamaño del bloque

### o Efecto Miss Rate



- Disminución de la **tasa de fallos iniciales** y **captura mejor la localidad espacial**
- Aumento de la **tasa de fallos de capacidad** (menor  $N^{\circ}$  Bloques  $\Rightarrow$  captura peor localidad temporal)



### Observaciones:

1. Valor óptimo mayor según aumenta  $M_c$
2. Caches integradas: tamaño y bloque fijo
3. Caches externas: tamaño y bloque variable

### o Miss Penalty

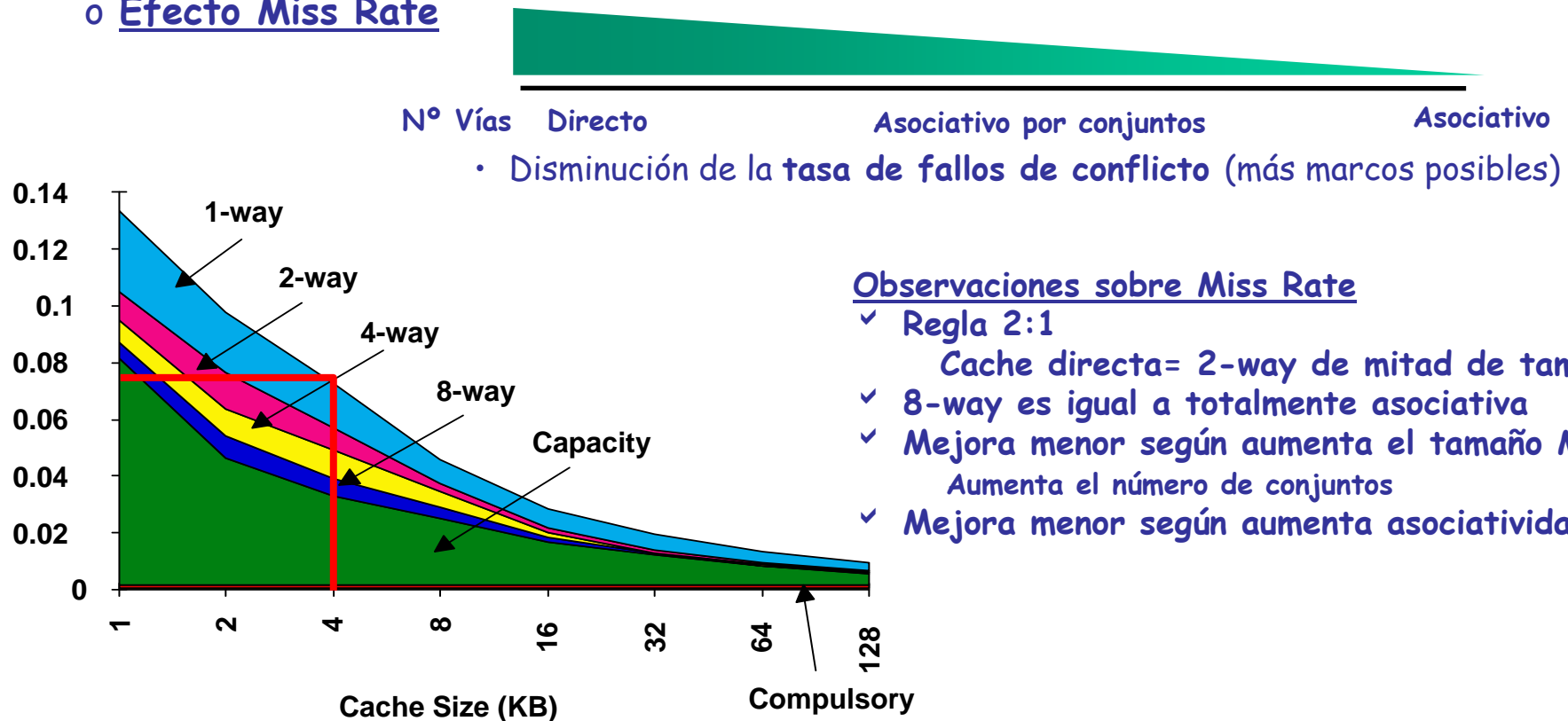


- Incremento de miss penalty puede cancelar beneficio de la mejora de miss rate (ver tr. 17 y 18)

# Optimizaciones para reducir la tasa de fallos

## □ Aumento de la asociatividad

### ○ Efecto Miss Rate



### ○ Tiempo de acceso

### ○ Coste hardware





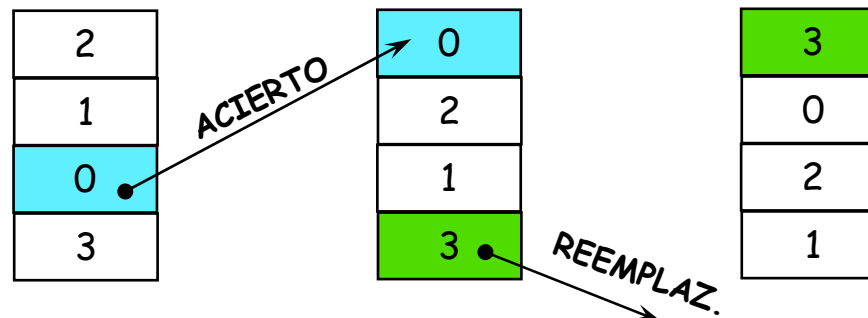
# Optimizaciones para reducir la tasa de fallos

## ❑ Aumento del tamaño de la Mc

- o Obviamente mejora la tasa de fallos (reducción de fallos de capacidad)
- o Puede empeorar tiempo de acceso, coste y consumo de energía

## ❑ Selección del algoritmo de reemplazamiento

- o **Espacio de reemplazamiento.** **Directo:** trivial; **Asociativo:** toda la cache; **Asociativo por conjuntos:** las líneas de un conjunto
- o **Algoritmos**
  - **Aleatorio:** El bloque reemplazado se escoge aleatoriamente
  - **LRU (*Least Recently Used*):** Se reemplaza el bloque menos recientemente usado. **Gestión:** pila



**Técnicas de implementación:** registros de edad, implementación de la pila, matriz de referencias

Para un grado de mayor que 4, muy costoso en tiempo y almacenamiento (actualiz. > tcache)

**LRU aproximado:** Algoritmo LRU en grupos y dentro del grupo

# Optimizaciones para reducir la tasa de fallos

## ❑ Selección del algoritmo de reemplazamiento (cont.)

### o Efecto Miss Rate



ALEATORIO                      Reemplazamiento                      LRU  
Disminuye la **tasa de fallos de capacidad** (mejora la localidad temporal)

Ejemplo: Fallos de datos por 1000 instrucciones en arquitectura Alpha ejecutando 10 programas SPEC 2000 (tamaño de bloque: 64 bytes)

Nº Vías	2		4		8	
Tamaño Mc	LRU	Aleatorio	LRU	Aleatorio	LRU	Aleatorio
16K	114.1	117.3	111.7	115.1	109.0	111.8
64K	103.4	104.3	102.4	102.3	99.7	100.5
256K	92.2	92.1	92.1	92.1	92.1	92.1

La diferencia LRU-Aleatorio disminuye al aumentar Mc

### o Tiempo de acceso

### o Coste hardware

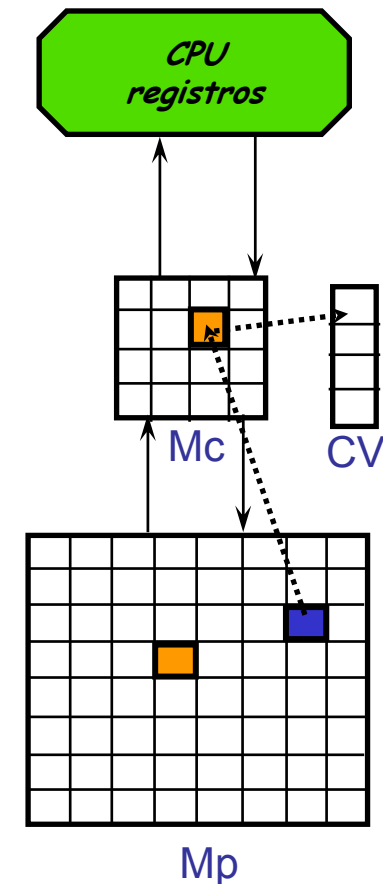


ALEATORIO                      Reemplazamiento                      LRU

# Optimizaciones para reducir la tasa de fallos

## ❑ Cache de víctimas

- o Objetivo: mantener la sencillez y rapidez de acceso de una Mc con emplazamiento directo, pero disminuyendo el impacto de los fallos de conflicto.
- o Es una memoria cache más pequeña y totalmente asociativa asociada a la memoria cache
  - Contiene los bloques que han sido **sustituidos más recientemente**
  - En un fallo primero **comprueba** si el bloque se encuentra en la cache de víctimas. En caso afirmativo, el bloque buscado se lleva de la cache de víctimas a la Mc
    - Cuanto menor es la memoria cache más efectiva es la cache víctima
    - Experimentos Jouppi (1990): En una cache de 4KB con emplazamiento directo, una cache víctima de 4 bloques elimina del 20% al 95% de los fallos, según programa.
- o Ejemplos:
  - HP 7200, 2 KB internos: 64 bloques de 32 bytes
  - ALPHA,
  - IBM Power 4-5-6, AMD Quad, ( **diferente uso** )



# Optimizaciones para reducir la tasa de fallos

## ❑ Compilador: Optimización de código

✓ **Ejemplo:** DEC Alpha 21064: Mc de 8 KBytes, directa, con 256 bloques. (Por tanto 1 bloque = 32 bytes = 4 palabras de 8 bytes)

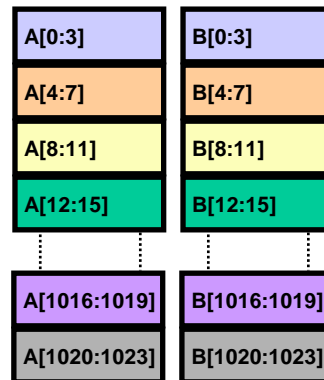
- Mc tiene 1024 palabras => Cada 1024 palabras se repite la asignación de líneas de la Mc.

### 1) Fusión de arrays: Mejora la **localidad espacial** para disminuir los **fallos de conflicto**

- Colocar las mismas posiciones de diferentes arrays en posiciones contiguas de memoria

```
double A[1024];  
double B[1024];
```

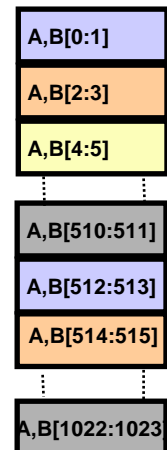
```
for (i = 0; i < 1024; i = i + 1)  
    C = C + (A[i] + B[i]);
```



**2x1024 fallos**  
2x256 de inicio  
2x3x256 de conflicto

```
struct fusion{  
    double A;  
    double B;  
} array[1024];
```

```
for (i = 0; i < 1024; i = i + 1)  
    C = C + (array[i].A + array[i].B);
```



**1024/2 fallos**  
2x256 de inicio

**Ganancia: 4**

# Optimizaciones para reducir la tasa de fallos

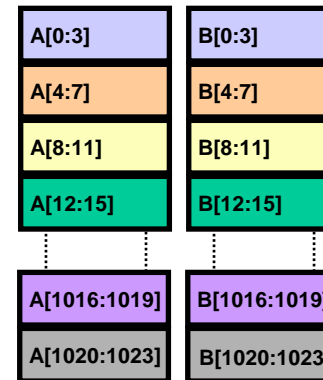
## ❑ Compilador: Optimización de código

### 2) Alargamiento de arrays: Mejora la localidad espacial para disminuir los fallos de conflicto

- Impedir que en cada iteración del bucle se compita por el mismo marco de bloque

```
double A[1024];  
double B[1024];  
  
for (i=0; i < 1024; i=i +1)  
    C = C + (A[i] + B[i]);
```

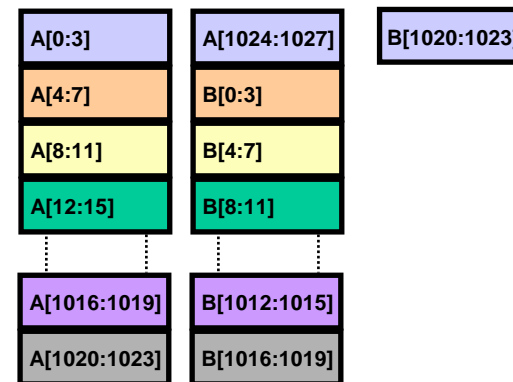
2x1024 fallos  
512 de inicio  
2x3x512 de conflicto



```
double A[1028];  
double B[1024];  
  
for (i=0; i < 1024; i=i+1)  
    C = C + (A[i] + B[i]);
```

1024/2 fallos  
2x256 de inicio

Ganancia: 4



# Optimizaciones para reducir la tasa de fallos

## ❑ Compilador: Optimización de código

### 3) Intercambio de bucles: Mejora la **localidad espacial** para disminuir los **fallos de conflicto**

- En lenguaje C las matrices se almacenan por filas, luego se debe variar en el bucle interno la columna

```
double A[128][128];  
  
for (j=0; j < 128; j=j+1)  
    for (i=0; i < 128; i=i+1)  
        C = C * A[i][j];
```

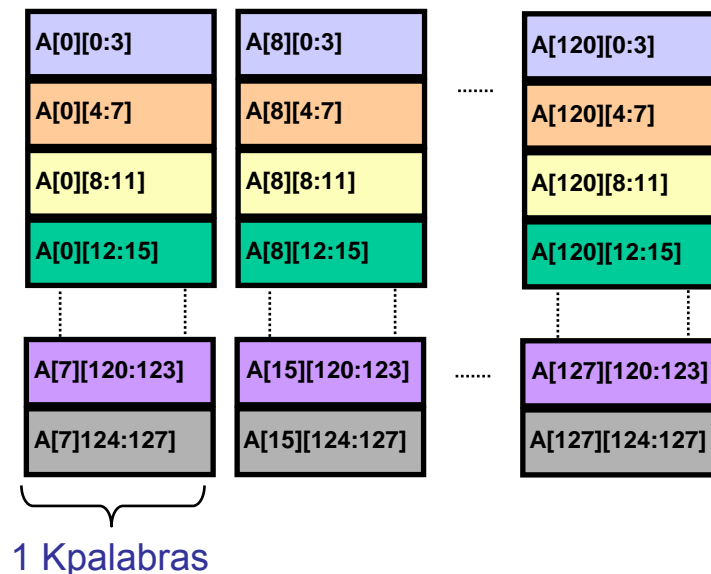
**128x128 fallos**  
16x256 de inicio  
Resto (12288) de conflicto

```
double A[128][128];  
  
for (i=0; i < 128; i=i+1)  
    for (j=0; j < 128; j=j+1)  
        C = C * A[i][j];
```

**128x128/4 fallos**  
16x256 de inicio

**Ganancia: 4**

A tiene  $2^{14}$  palabras = 16 Kpalabras =>  
es 16 veces mayor que Mc



# Optimizaciones para reducir la tasa de fallos

## ❑ Compilador: Optimización de código

### 4) Fusión de bucles: Mejora la localidad temporal para disminuir los fallos de capacidad

- Fusionar los bucles que usen los mismos arrays para usar los datos que se encuentran en cache antes de desecharlos

```
double A[64][64];

for (i=0; i < 64; i=i+1)
    for (j=0; j < 64; j=j+1)
        C = C * A[i][j];

for (i=0; i < 64; i=i+1)
    for (j=0; j < 64; j=j+1)
        D = D + A[i][j];
```

**$(64 \times 64 / 4) \times 2$  fallos**  
4x256 de inicio  
Resto (4x256) de capacidad

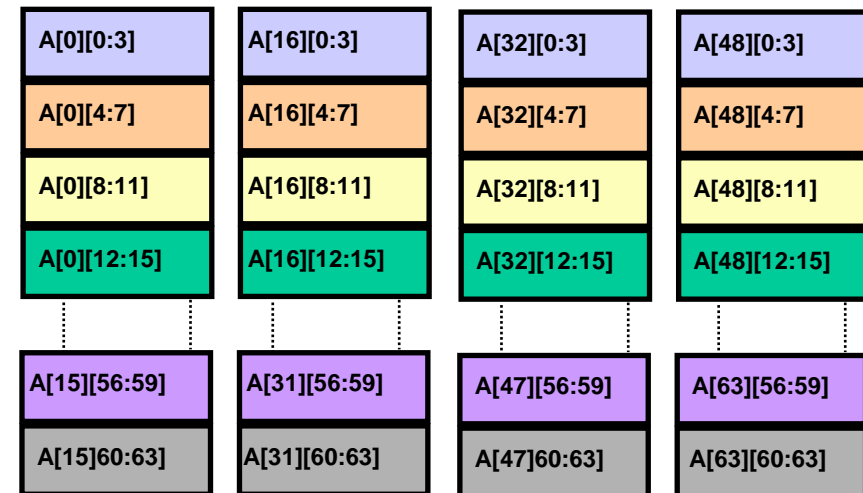
```
double A[64][64];

for (i=0; i < 64; i=i+1)
    for (j=0; j < 64; j=j+1)
    {
        C = C * A[i][j];
        D = D + A[i][j];
    }
```

**$64 \times 64 / 4$  fallos**  
4x256 de inicio

**Ganancia: 2**

A tiene  $2^{12}$  palabras = 4 Kpalabras => es  
4 veces mayor que Mc



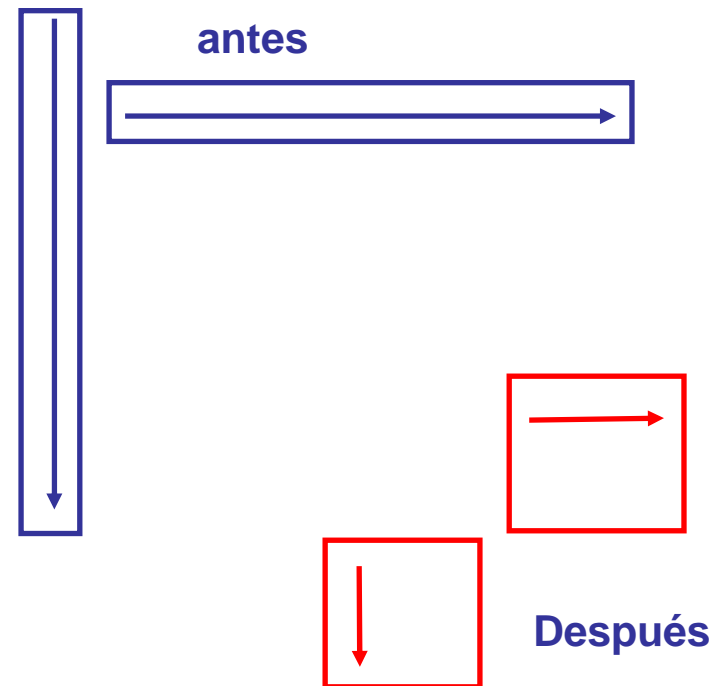
# Optimizaciones para reducir la tasa de fallos

## ❑ Compilador: Optimización de código

5) Calculo por bloques( Blocking): Mejora la localidad temporal para disminuir los fallos de capacidad

```
/* Antes */  
for (i=0; i < N; i=i+1)  
  for (j=0; j < N; j=j+1)  
    {r = 0;  
     for (k=0; k < N; k=k+1)  
       r = r + y[i][k]*z[k][j];  
     x[i][j] = r;  
    };
```

- ✓ Dos bucles internos. Para cada valor de  $i$ :
  - Lee todos los  $N \times N$  elementos de  $z$
  - Lee  $N$  elementos de 1 fila de  $y$
  - Escribe  $N$  elementos de 1 fila de  $x$
- ✓ Fallos de capacidad dependen de  $N$  y Tamaño de la cache:
- ✓ Idea: calcular por submatrices  $B \times B$  que permita el tamaño de la cache





# Optimizaciones para reducir la tasa de fallos

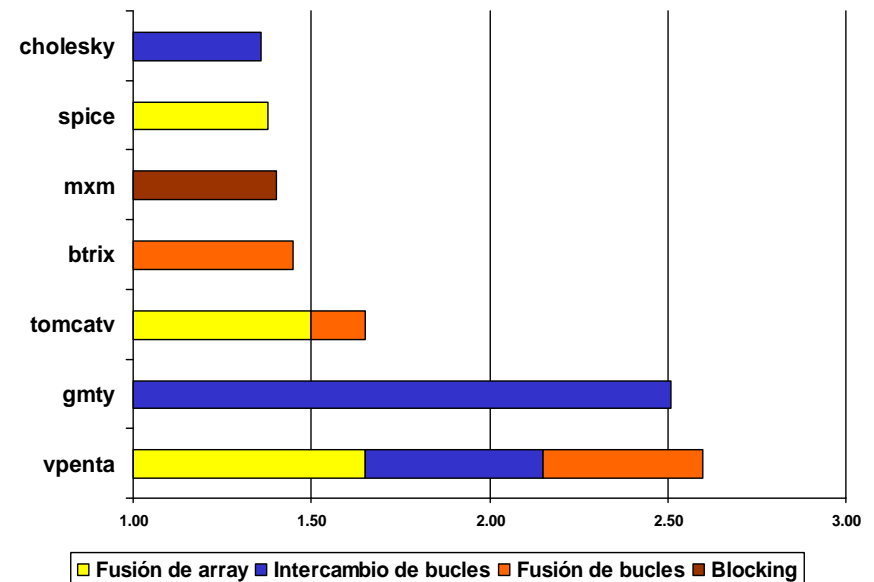
## ❑ Compilador: Optimización de código

5) Calculo por bloques( Blocking): Mejora la localidad temporal para disminuir los fallos de capacidad

```
/* Despues */  
for (jj=0; jj < N; jj=jj+B)  
for (kk=0; kk < N; kk=kk+B)  
for (i=0; i < N; i=i+1)  
    for (j=jj; j < min(jj+B-1,N); j=j+1)  
        {r = 0;  
         for (k=kk; k < min(kk+B-1,N); k=k+1)  
             r = r + y[i][k]*z[k][j];  
         x[i][j] = x[i][j]+r;  
        };
```

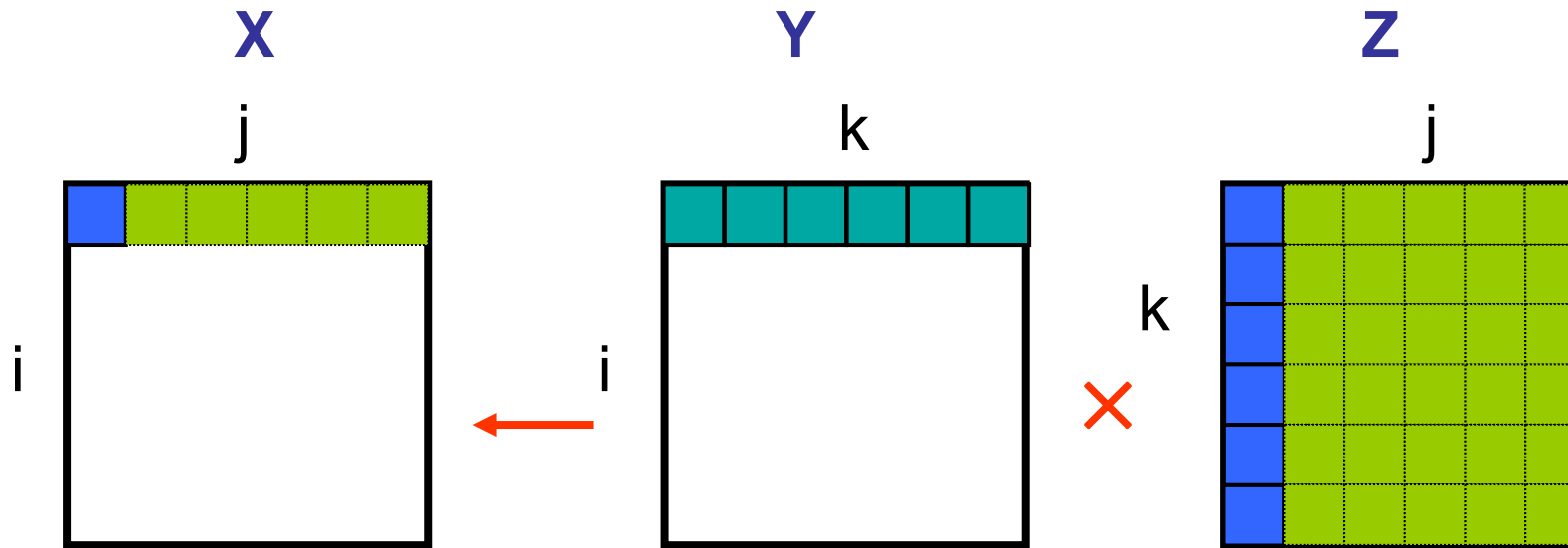
✓ B Factor de bloque (*Blocking Factor*)


## Mejora de rendimiento




# Optimizaciones para reducir la tasa de fallos

## ❑ Ejemplo: Producto de matrices 6x6 (sin blocking)



  $i = 0, j = 0, k = 0..5$

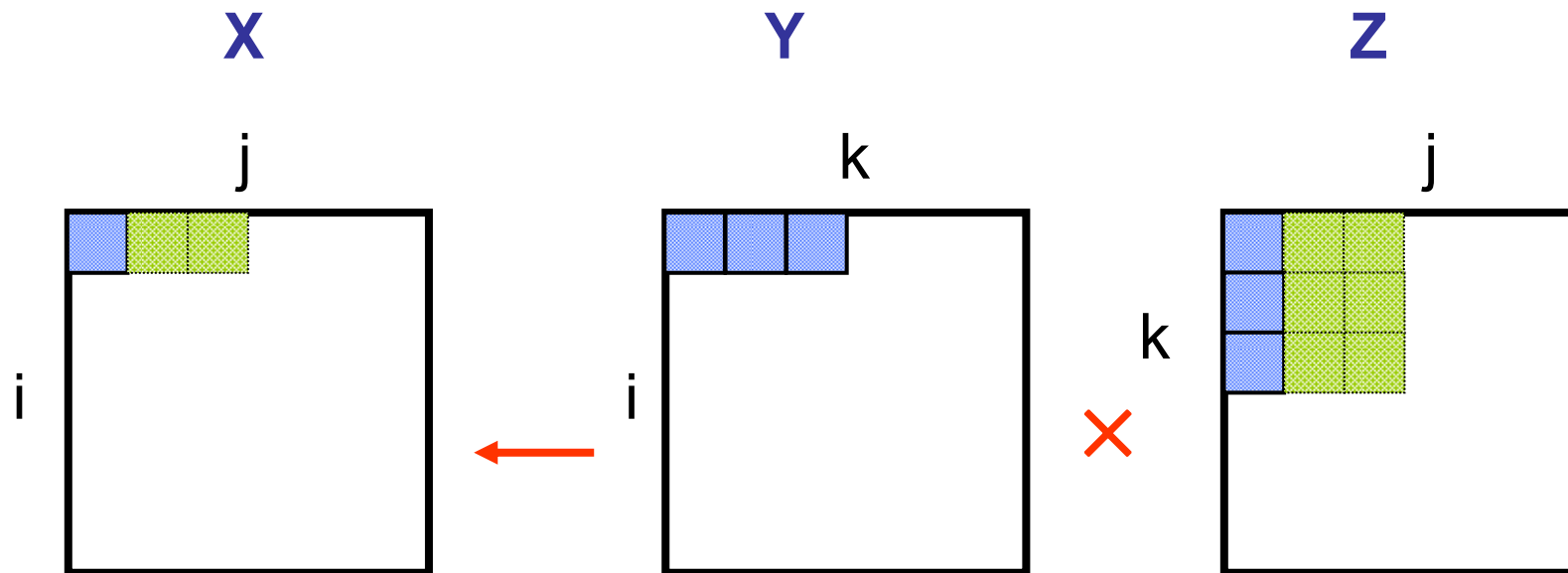
  $i = 0, j = 1..5, k = 0..5$

$$X_{ij} = \sum_k Y_{ik} Z_{kj}$$

Al procesar la 2ª fila de  $Y$  ( $i=1$ ) se necesita de nuevo 1ª col de  $Z$ :  
¿Está todavía en la cache? Cache insuficiente provoca múltiples fallos sobre los mismos datos

# Optimizaciones para reducir la tasa de fallos

## ❑ Ejemplo "blocking": Con Blocking ( $B=3$ )



$i = 0, j = 0, k = 0..2$

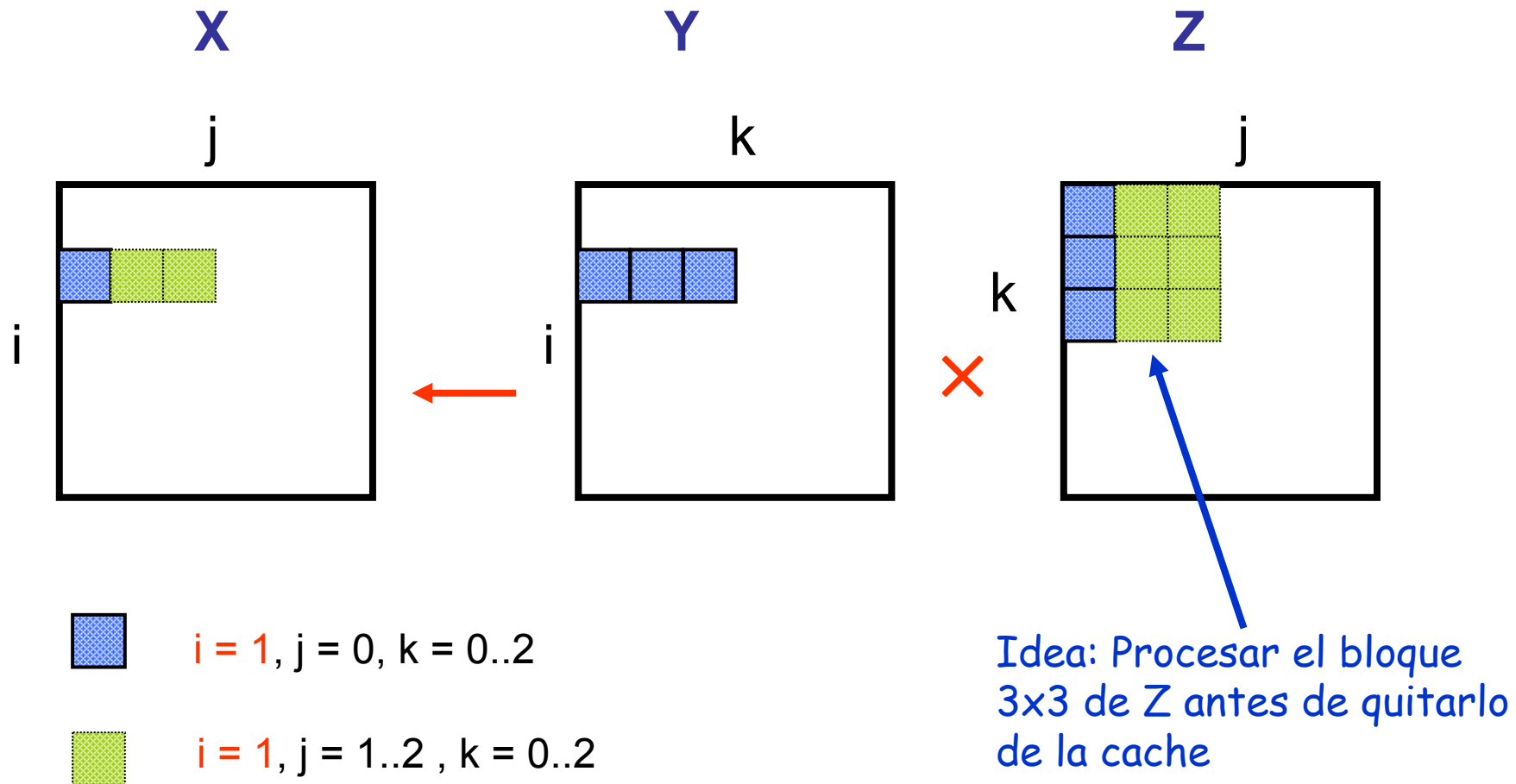


$i = 0, j = 1..2, k = 0..2$

Evidentemente, los elementos de X no están completamente calculados

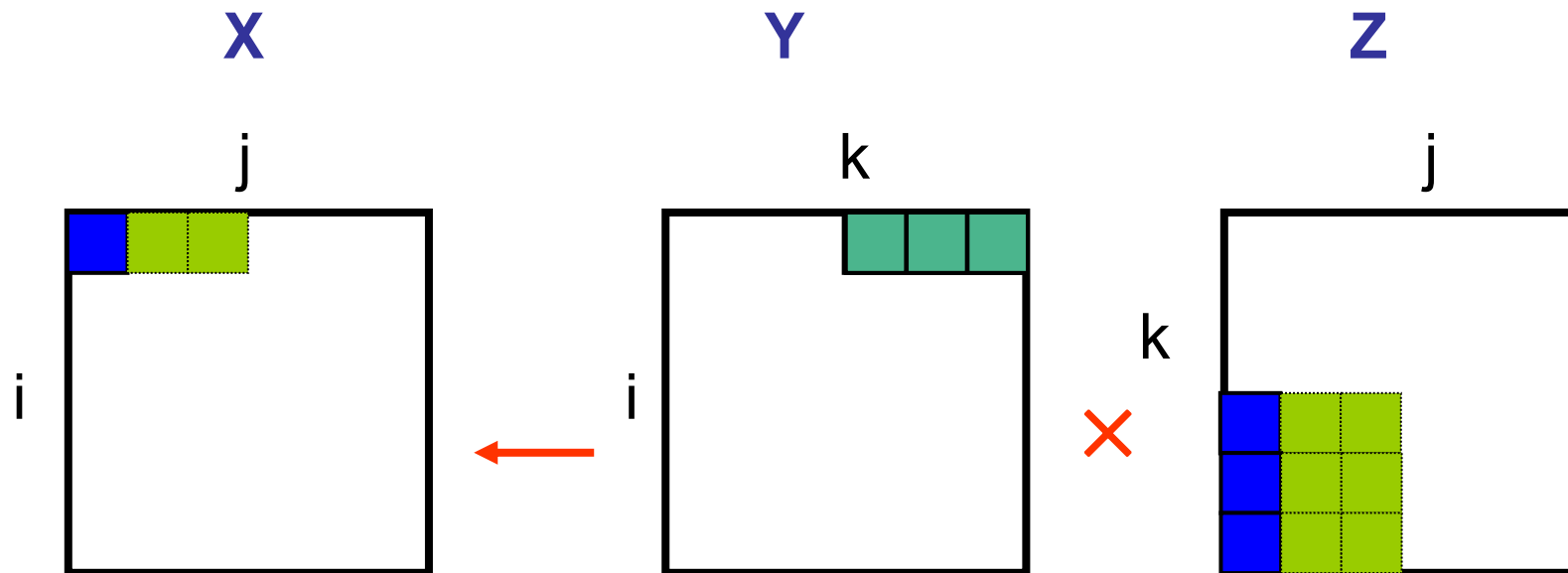
# Optimizaciones para reducir la tasa de fallos

## □ Ejemplo "blocking": Con Blocking ( $B=3$ )



# Optimizaciones para reducir la tasa de fallos

❑ Con Blocking (B=3). Algunos pasos después...



$i = 0, j = 0, k = 3..5$



$i = 0, j = 1..2, k = 3..5$

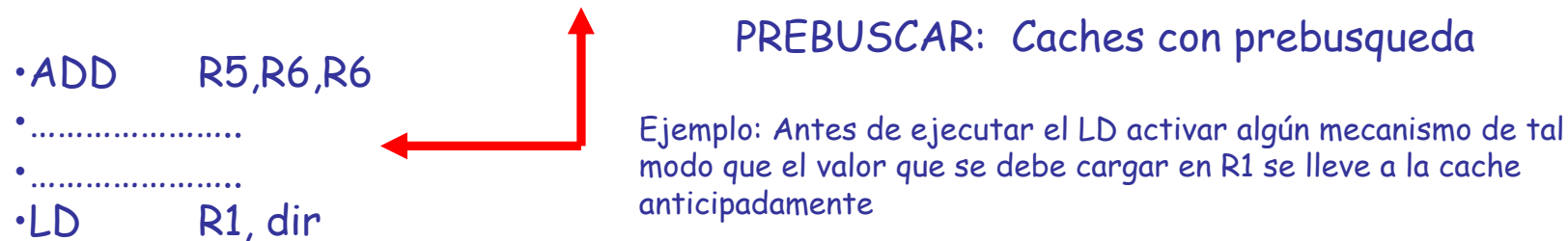
Y ya empezamos a tener  
elementos de X  
completamente calculados!

# Optimizaciones para reducir la tasa de fallos o la penalización por fallo

## ❑ Cache con prebúsqueda

✓ Idea

Ocultar la latencia de un fallo de cache solapándolo con otras instrucciones independientes



➤ Anticipa los fallos de Cache anticipando las búsquedas antes de que el procesador demande el dato o la instrucción que provocarían un fallo

- ✓ Se hace una búsqueda en memoria sin que la instrucción o el dato buscado haya sido referenciado por el procesador
  - Si la información prebuscada se lleva a Mc => **reducción tasa fallos**
  - Si la información prebuscada se lleva a buffer auxiliar => **reducción penalización**
- ✓ El acceso a memoria se solapa con la ejecución normal de instrucciones en el procesador
- ✓ No se relaciona con los registros. No genera riesgos
- ✓ Existe la posibilidad de que se hagan búsquedas innecesarias

➤ **Dos tipos**

- ✓ Prebúsqueda SW
- ✓ Prebúsqueda HW

# Optimizaciones para reducir la tasa de fallos o la penalización por fallo

## ❑ Cache con prebúsqueda hardware

### o Prebúsqueda de instrucciones

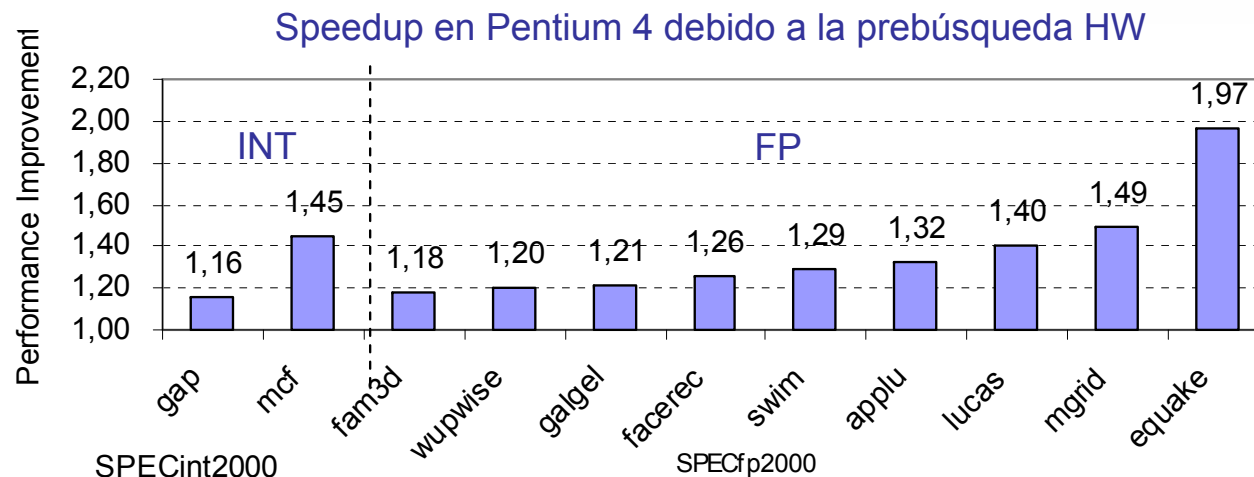
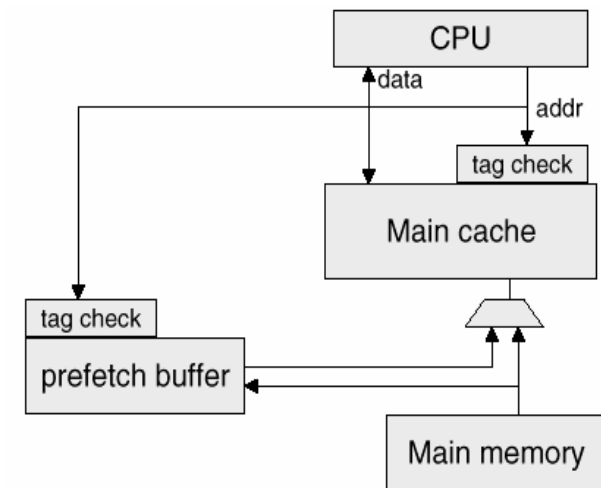
- Típicamente: la CPU busca dos bloques en un fallo (el referenciado y el siguiente)
- El bloque buscado se lleva a Mc
- El prebúsqueda se lleva a un buffer ("prefetch buffer" o "stream buffer"). Al ser referenciado pasa a Mc

### o Prebúsqueda de datos

- Ejemplo Pentium 4: La prebúsqueda se activa si se producen dos fallos sucesivos en L2 dentro de una misma página de memoria y la distancia entre los bloques referenciados es  $< 256$  bytes

## Implementación

(Prebúsqueda de un bloque)



# Optimizaciones para reducir la tasa de fallos o la penalización por fallo

---

## ❑ Cache con prebúsqueda software

- o Instrucciones especiales de prebúsqueda introducidas por el compilador
- o La eficiencia depende del compilador y del tipo de programa
- o Prebúsqueda con destino en cache (MIPS IV, PowerPC, SPARC v. 9), o en registro (HP-PA)
- o Instrucciones de prebúsqueda no producen excepciones. Es una forma de especulación.
- o Funciona bien con bucles y patrones simples de acceso a arrays.  
Aplicaciones de cálculo
- o Funciona mal con aplicaciones enteras que presentan un amplio reuso de Cache
- o Overhead por las nuevas instrucciones. Más búsquedas. Más ocupación de memoria



# Optimizaciones para reducir la tasa de fallos o la penalización por fallo

---

## ❑ Cache con prebúsqueda software (ejemplo)

- ✓ Cache 8 KB directa, bloque:16 bytes, write-back (con asignación en escritura)
- ✓ Datos: a(3,100), b(101,3). Elemento arrays = 8 bytes. Cache inicialmente vacía.  
Ordenación en memoria: por filas
  - 1 bloque cache = 2 palabras (elementos)

- ✓ Programa (sin prebúsqueda):

```
for (i:=0; i<3; i:=i+1)
    for (j:=0; j<100; j:=j+1)
        a[i][j] := b[j][0] * b[j+1][0]
```

- ✓ Fallos

- Acceso a elementos de "a": Se escriben y acceden en cache tal como están almacenados en memoria. Cada acceso a memoria proporciona dos palabras (beneficio de localidad espacial).

$$\text{Fallos "a"} = (3 \times 100) / 2 = 150$$

- Acceso a elementos de "b" (si ignoramos fallos de conflicto): Un fallo por cada valor de j cuando i=0 => 101 fallos. Para los restantes valores de i, los elementos de b ya están en la cache.
- Total fallos: 150+101 = 251

# Optimizaciones para reducir la tasa de fallos o la penalización por fallo

## ❑ Cache con prebúsqueda software (ejemplo)

- ✓ Suposición: La penalización por fallo es de tal duración que se necesita iniciar la prebúsqueda 7 iteraciones antes.

- ✓ Idea: partir bucle

```
/* para i=0 (prebusca a y b) */
for (j:=0; j<100; j:=j+1) {
    prefetch (b[j+7][0]); /* b[j][0] para 7 iteraciones más tarde */
    prefetch (a[0][j+7]); /* a[0][j] para 7 iteraciones más tarde */
    a[0][j] := b[j][0] * b[j+1][0] ; }
Fallos:  $\lceil 7/2 \rceil$            Fallos: 7

/* para i=1,2 (prebusca sólo a, ya que b ya está en cache) */
for (i:=1; i<3; i:=i+1)
    for (j:=0; j<100; j:=j+1) {
        prefetch (a[i][j+7]); /* a[i][j] para 7 iteraciones más tarde */
        a[i][j] := b[j][0] * b[j+1][0] ; }
Fallos:  $2 * \lceil 7/2 \rceil$  (para i=1,2)
```

- ✓ Total fallos  $3 * \lceil 7/2 \rceil + 7 = 19$  fallos
- ✓ Instrucciones extra (los prefetch):  $100*2 + 200*1 = 400$
- ✓ Fallos evitados =  $251 - 19 = 232$

# Espacio de diseño para la mejora del rendimiento de Mc

## ¿ Como mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Asociatividad	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV $\Rightarrow$ DF	Cache multibanco
Tamaño de Mc	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Algoritmo de reemplazamiento	Cache multinivel	Cache de trazas	
Cache de víctimas			
Optimización del código (compilador)			
Prebúsqueda HW			
Prebúsqueda SW			

# Optimizaciones para reducir la penalización por fallo

## □ Dar prioridad a la lecturas sobre las escrituras

- Un fallo de lectura puede impedir la continuación de la ejecución del programa; un fallo de escritura puede ocultarse

- Con escritura directa ( write through)

- Buffer de escrituras (rápido). Depositar en buffer las palabras que tienen que ser actualizadas en Mp y continuar ejecución.
- La transferencia del buffer a Mp se realiza en paralelo con la ejecución del programa
- Riesgo: El valor más reciente de una variable, puede estar en buffer y no todavía en Mp

- Ejemplo: cache directa con write-through

SW 512(R0), R3;                      M[512] <- R3                      (línea 0 de Mc)

LW R1, 1024(R0);                      R1 <- M[1024]                      (línea 0 de Mc)

LW R2, 512(R0);                      R2 <- M[512]                      (fallo lectura: línea 0 de Mc)

(Dependencia LDE. Con buffer de escrituras ¿tienen R3 y R2 el mismo valor?)

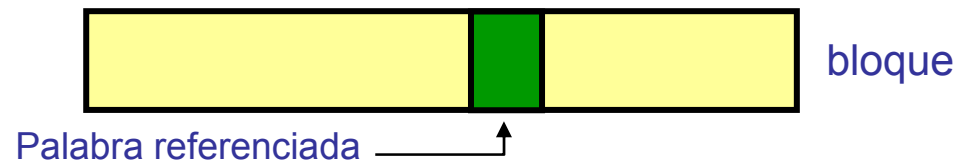
- En fallo de lectura chequear contenido del buffer de escrituras. Si no hay conflicto, dar prioridad a la lectura y proseguir la ejecución del programa.

- Con post-escritura (write back)

- Se puede aplicar la misma idea, disponiendo de un buffer donde quepa un bloque completo
- Si un fallo de lectura implica remplazar un bloque sucio => mover bloque sucio a buffer y leer primero bloque en fallo.
- Riesgo: similar al caso anterior

# Optimizaciones para reducir la penalización por fallo

- ❑ **Envío directo de la palabra solicitada al procesador**
  - o **Carga anticipada (early restart):** Cuando la palabra solicitada se carga en memoria cache se envía al procesador, sin esperar a la carga del bloque completo
- ❑ **Primero la palabra solicitada (critical word first)**
  - o Primero se lleva al procesador y a memoria cache la palabra solicitada
  - o El resto del bloque se carga en memoria cache en los siguientes ciclos
- ❑ **La eficiencia de estas técnicas depende del tamaño del bloque. Útil con bloques grandes.**
  - o Para bloques pequeños la ganancia es muy pequeña
  - o Problema. Localidad espacial: alta probabilidad de acceder a continuación a la siguiente palabra en secuencia.



# Optimizaciones para reducir la penalización por fallo

## ❑ Fusión de buffers de escritura

- o Idea: incluir en un mismo buffer múltiples palabras consecutivas para:
  - Optimizar las transferencias buffer - memoria principal
  - Reducir los campos de dirección de las entradas del buffer
- o Si el buffer contiene bloques modificados, chequear la dirección del nuevo dato para ver si coincide con la dirección de alguna entrada del buffer. Si hay coincidencia, no asignar una nueva entrada del buffer, combinar los nuevos datos en la misma entrada
- o Utilizado en Sun T1 (Niagara)

## ❑ Ejemplo: buffer de escritura sin y con fusión

Cada escritura con su dirección

Write address	V	V	V	V		
100	1	Mem[100]	0	0	0	0
108	1	Mem[108]	0	0	0	0
116	1	Mem[116]	0	0	0	0
124	1	Mem[124]	0	0	0	0

Una sola dirección para 4 escrituras consecutivas

Write address	V	V	V	V				
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

- Buffer de escritura con 4 entradas
- Cada entrada: 4 palabras de 8 bytes

© 2007 Elsevier, Inc. All rights reserved.

# Optimizaciones para reducir la penalización por fallo

## □ Cache multinivel (L2, L3, ...)

- o Tiempo medio de acceso a memoria (TMAM): Un nivel

- $TMAM = \text{Hit time} + \text{Miss Rate} \times \text{Miss Penalty}$

- o Tiempo medio de acceso a memoria: Dos niveles

$$TMAM = \text{Hit Time } L1 + \text{Miss Rate } L1 \times \text{Miss Penalty } L1$$

$$\text{Miss Penalty } L1 = \text{Hit Time } L2 + \text{Miss Rate } L2 \times \text{Miss Penalty } L2$$

Luego:

$$TMAM = \text{Hit Time } L1 + \text{Miss Rate } L1 \times [\text{Hit Time } L2 + (\text{Miss Rate } L2 \times \text{Miss Penalty } L2)]$$

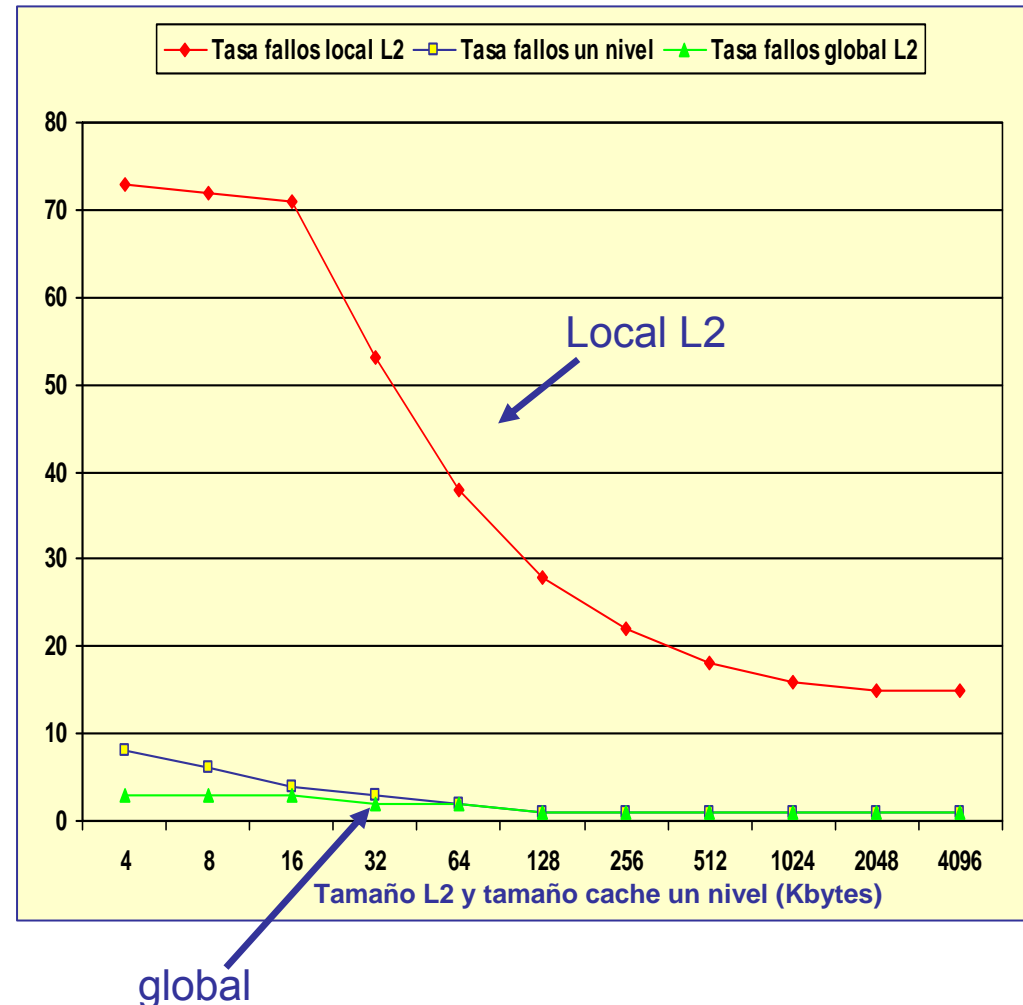
- o Definiciones:

- *Tasa de fallos local en una cache (Lx):* fallos en cache Lx dividido por el numero total de accesos a la cache Lx
- *Tasa de fallos global en una cache (Lx):* fallos en cache Lx dividido por el numero total de accesos a memoria generados por el procesador
  - Consecuencia: Tasa de fallos local en L1 = Tasa de fallos global en L1
- La tasa de fallos global es lo importante
  - L1: Afecta directamente al procesador => Acceder a un dato en el ciclo del procesador
  - L2: Afecta a la penalización de L1 => Reducción del tiempo medio de acceso

# Optimizaciones para reducir la penalización por fallo

## ❑ Cache multinivel ( L2,L3,...)

- o Comparación: Tasa de fallos con:
  - Cache un nivel, varios tamaños
  - Cache 2 niveles
    - L1 32Kbytes
    - L2 varios tamaños
- o Tasa de fallos local no es una medida muy relevante
- o El tiempo de acceso de L2 solo afecta al tiempo de penalización
- o Importante que tamaño de L2  $\gg$  L1
- o Reducción de fallos en L2: mismas técnicas que L1 (asociatividad, tamaño de bloque,...)
- o Costo





# Espacio de diseño para la mejora del rendimiento de Mc

## ¿ Como mejorar el rendimiento de la cache?

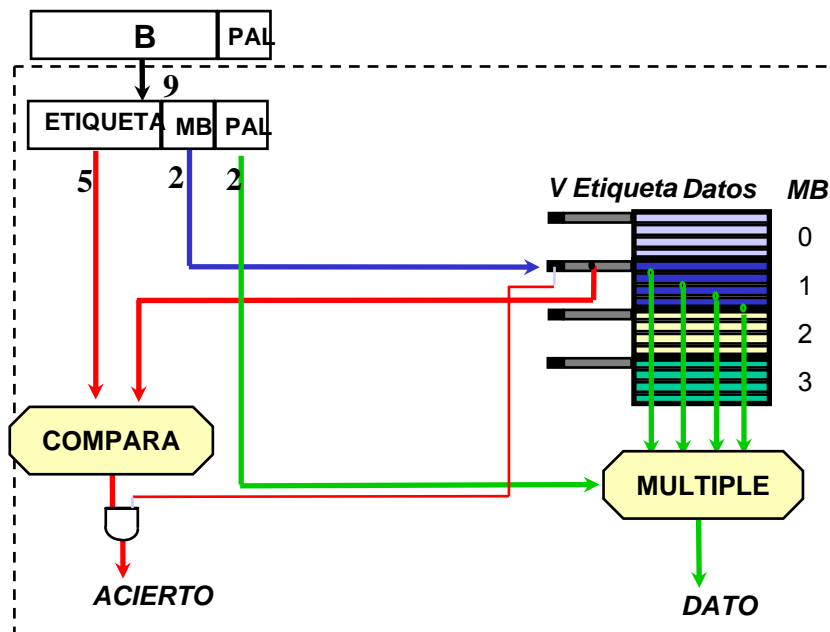
Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Asociatividad	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Tamaño de Mc	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Algoritmo de reemplazamiento	Cache multinivel	Cache de trazas	
Cache de víctimas			
Optimización del código (compilador)			
Prebúsqueda HW			
Prebúsqueda SW			

# Optimizaciones para reducir el tiempo de acierto

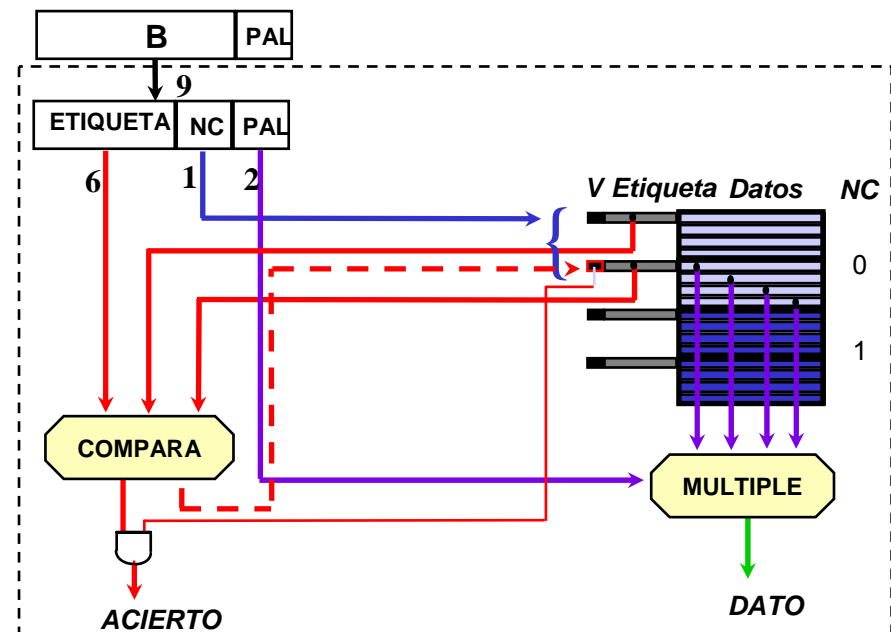
## ❑ Caches simples y pequeñas

- o El acceso al directorio y la comparación de tags consume tiempo
- o Ejemplo: Comparación de acceso a un dato en cache directa y en cache asociativa por conjuntos con 2 vías

Identificación+Comparación+Lectura



Directo

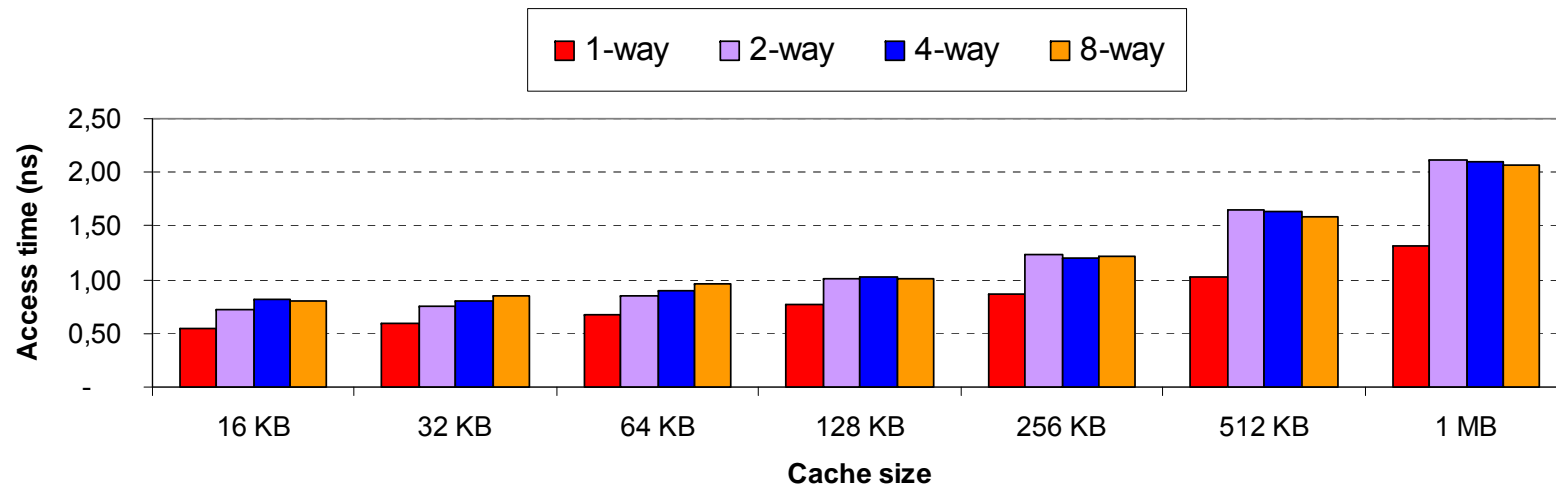


Asociativo por conjuntos 2 vías

# Optimizaciones para reducir el tiempo de acierto

## ❑ Caches simples y pequeñas

- o Una cache pequeña se pueda integrar junto al procesador
  - evitando la penalización en tiempo del acceso al exterior
  - Tiempo de propagación versus tiempo de ciclo del procesador
- o Ejemplo: tres generaciones del procesadores AMD (K6, Athlon y Opteron) han mantenido el mismo tamaño para las caches L1
- o Simple (cache directa o grado de asociatividad pequeño)
  - En cache directa se puede solapar chequeo de tags y acceso al dato, puesto que el dato sólo puede estar en un lugar
  - El aumento del número de vías puede aumentar los tiempos de comparación de tags
- o Ejemplo: impacto del tamaño de la cache y la asociatividad sobre el tiempo de acceso (tecnología 90 nm)



# Optimizaciones para reducir el tiempo de acierto

## ❑ Ocultar la latencia de traducción DV => DF

### o Cache de direcciones virtuales

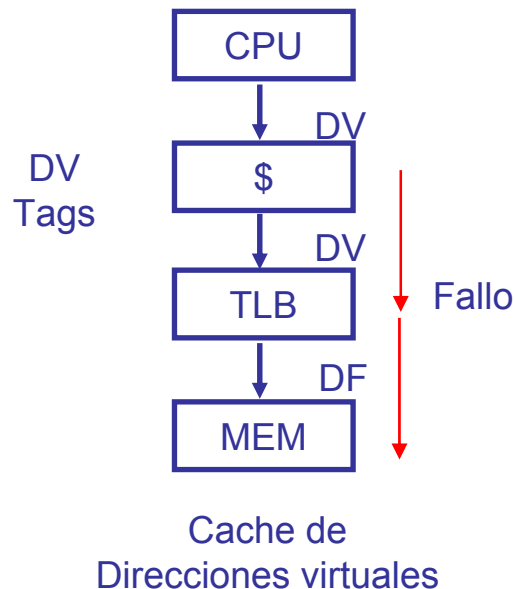
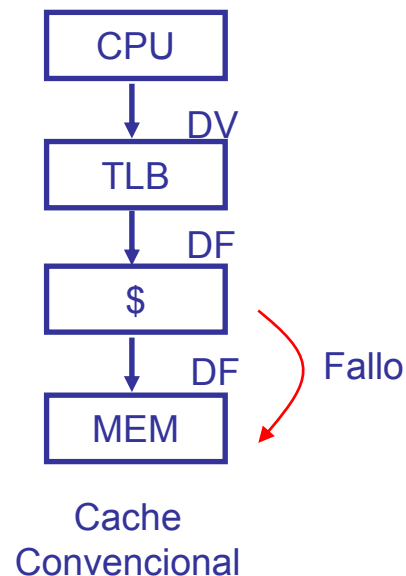
- Acceder a la cache con la dirección virtual (sin traducción previa a DF)

#### ✓Problema del cambio de contexto:

- Al cambio de contexto borrar la cache. De lo contrario: falsos aciertos
- Costo = tiempo de borrado ( flush) + fallos iniciales
- Solución: Añadir un identificador de proceso (PID) a la DV. Permite distinguir DVs de dos procesos diferentes.

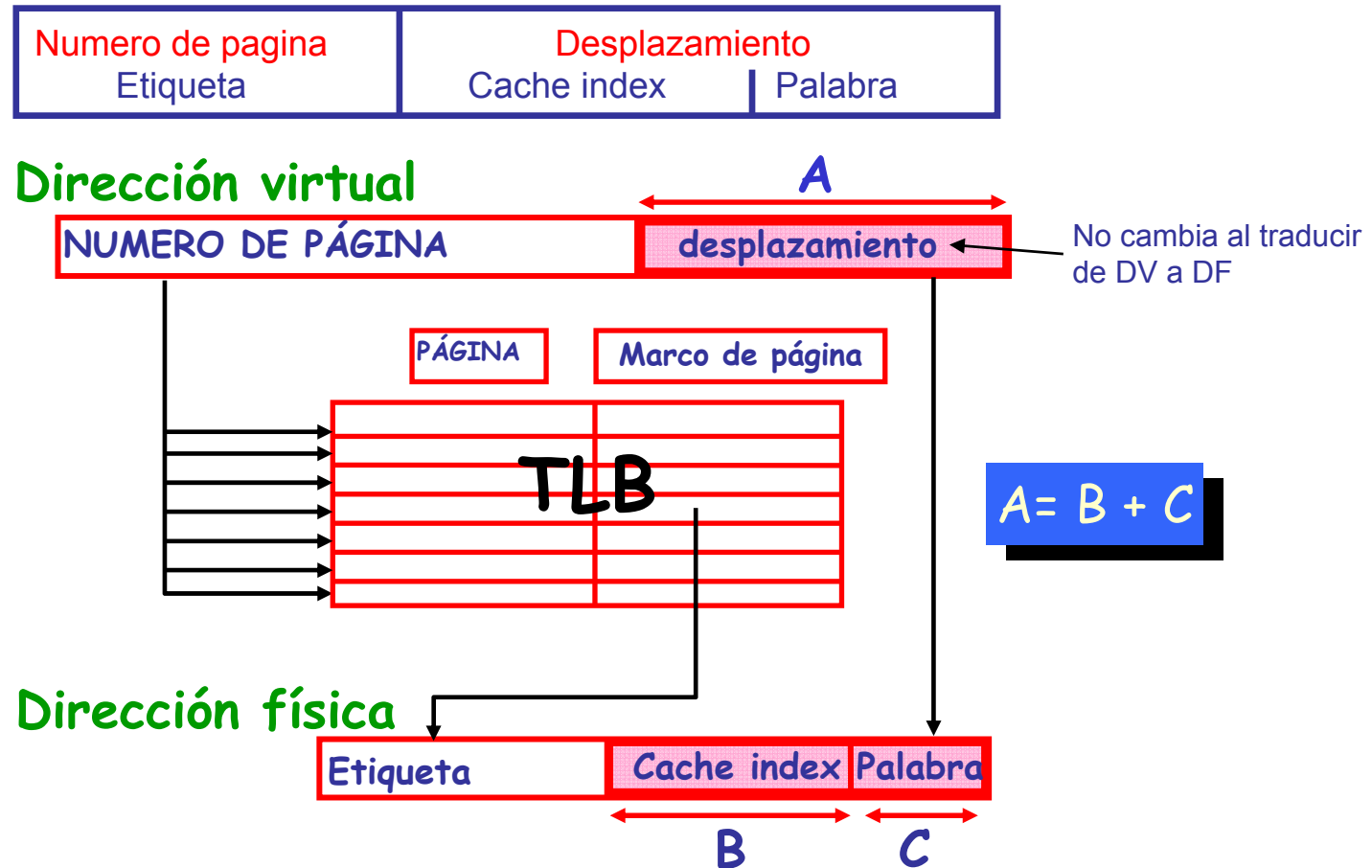
#### ✓Problema de sinónimos

- Dos direcciones virtuales apuntan la misma dirección física. Implica que dos copias de la misma DF pueden residir en la cache
- Solución: mecanismos HW para garantizar que cada bloque de la cache se corresponde con un bloque de Mp diferente



# Optimizaciones para reducir el tiempo de acierto

- ❑ Ocultar la latencia de traducción DV => DF  
o Cache virtualmente accedida, físicamente marcada



- o Se solapa la traducción del numero de pagina con acceso a los tag del marco de bloque
- o Con cache directa, limita el tamaño de la cache al tamaño de pagina
- o Un modo de aumentar el tamaño máximo de la cache es aumentar la asociatividad

# Optimizaciones para reducir el tiempo de acierto

## □ Predicción de vía

- o Permite combinar el rápido tiempo de acierto de una cache directa con la menor tasa de fallos de conflicto de una cache asociativa por conjuntos
- o Cada bloque de la cache contiene bits de predicción que indican cuál será la vía más probable del siguiente acceso
- o El multiplexor selecciona la vía predicha antes de completar la comparación de tags
- o En caso de fallo de la predicción, completar la comparación de tags en todas las líneas del conjunto seleccionado



- o Se han alcanzado tasas de éxito en la predicción en torno al 85% con una cache asociativa por conjuntos con 2 vías
- o Problema: diferentes tiempos en caso de acierto
- o Ejemplo: Se utiliza en Pentium 4

# Optimizaciones para reducir el tiempo de acierto

---

## ❑ Cache de trazas

- o Empleada por primera (y por el momento, única) vez en el Pentium 4
- o La cache guarda secuencias dinámicas de instrucciones ejecutadas en lugar de secuencias estáticas de instrucciones en orden de almacenamiento en memoria
  - El predictor de saltos queda incluido en la secuencia dinámica de instrucciones que almacena la cache
  - Problema: Cuando una instrucción de salto presenta diferentes comportamientos, las mismas instrucciones pueden aparecer en distintas trazas => aparecen múltiples veces en la cache
- o Optimización adicional en Pentium 4. Se almacenan secuencias de  $\mu$ -instrucciones ya descodificadas => ahorro tiempo
- o Relativamente costosa en términos de área, consumo y complejidad en comparación con los beneficios

# Espacio de diseño para la mejora del rendimiento de Mc

## ¿ Como mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Asociatividad	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV $\Rightarrow$ DF	Cache multibanco
Tamaño de Mc	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Algoritmo de reemplazamiento	Cache multinivel	Cache de trazas	
Cache de víctimas			
Optimización del código (compilador)			
Prebúsqueda HW			
Prebúsqueda SW			



# Optimizaciones para aumentar el ancho de banda

## ❑ Cache sin bloqueo ( non-blocking, lockup-free )

### o Idea

Ocultar la latencia de un fallo de cache solapándolo con otras instrucciones independientes

```
•ADD    R5,R6,R6
•.....
•.....
•LD      R1, dir
•.....
•.....
•ADD     R4,R4,R1
```

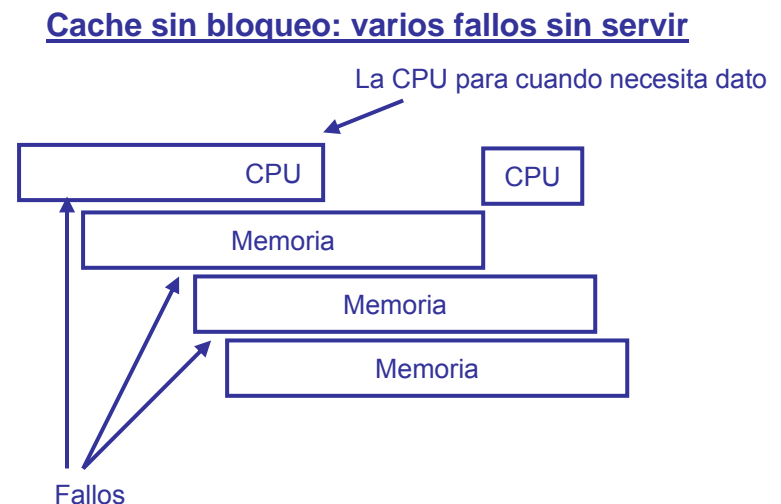
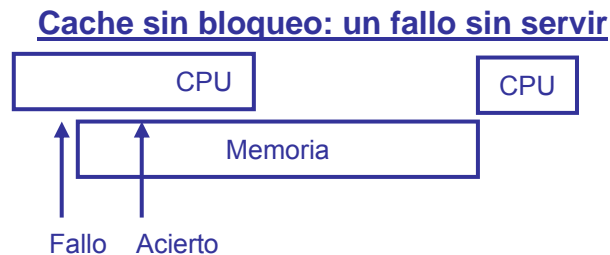
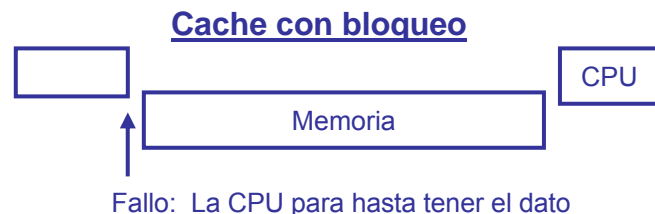
PREBUSCAR: Caches con prebúsqueda

NO BLOQUEAR: Cache que no bloquean  
(Se siguen ejecutando instrucciones después del LD. El ADD no se ejecuta hasta que R1 está disponible)

# Optimizaciones para aumentar el ancho de banda

## ❑ Cache sin bloqueo ( non-blocking, lockup-free )

- o Permite que la ejecución siga aunque se produzca un fallo mientras no se necesite el dato. (Se aplica a la cache de datos).
- o Un fallo sin servir (hit under 1 miss). Sigue ejecutando y proporcionando datos que están en cache
  - HP7100, Alpha 21064
- o Múltiples fallos sin servir (hit under multiple misses)
  - R12000 (4) , Alpha21264 (8), HP8500 (10), PentiumIII y 4 ( 4)
- o Los beneficios dependen de la planificación de instrucciones
- o Requiere interfase de memoria más complejo ( múltiples bancos )



# Optimizaciones para aumentar el ancho de banda

## ❑ Cache sin bloqueo ( non-blocking, lockup-free )

o Hay que asociar un registro a la petición cuando se inicia un load sin bloqueo

LD      R1,dir

o Información necesaria en el control de la función

Dirección del bloque que produce el fallo

Registro destino donde se almacena el dato

Formato del dato (Byte, half-word, word...)

Ítem en el bloque que produce el fallo

o Implementación (MSHR, Miss Status Holding Register):

Bit valido	Dirección bloque
---------------	---------------------

Bit valido	Destino	Formato
Bit valido	Destino	Formato
Bit valido	Destino	Formato
Bit valido	Destino	Formato

Palabra 0

Palabra 1

Palabra 2

Palabra 3

### Tipos de fallos

Fallo primario (1º de bloque)

Fallo secundario (restantes  
mismo bloque)

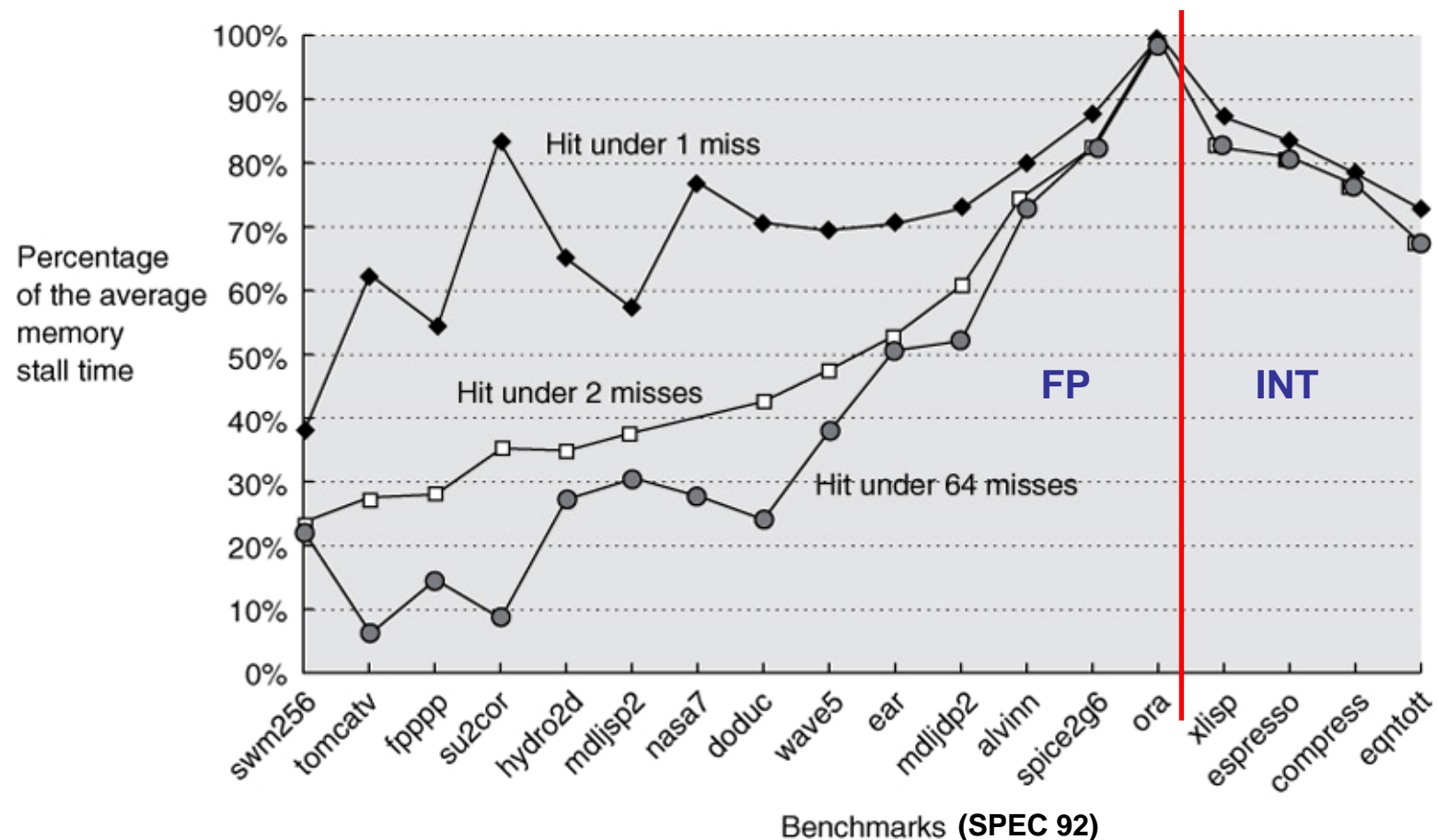
Estructura del MSHR para  
bloque de 4 palabras  
(Solo un fallo por palabra)

# Optimizaciones para aumentar el ancho de banda

## ❑ Cache sin bloqueo ( non-blocking, lockup-free )

o Porcentaje de tiempo de parada del procesador por fallos en la cache (Caso base: cache con bloqueo = 100%)

o Datos experimento: Cache directa 8 KB, Bloque 32 B, Miss Penalty 16 ciclos



# Optimizaciones para aumentar el ancho de banda

## ❑ Cache multibanco

- o Dividir la cache en bancos independientes que puedan soportar accesos simultáneos.
  - Ejemplo: L2 de SUN T1 (Niágara) tiene 4 bancos, L2 de AMD Opteron tiene 2 bancos
- o La organización en bancos funciona bien cuando los accesos se dispersan de forma natural entre los diferentes bancos
- o El entrelazamiento de orden bajo suele funcionar bien
  - Bloques consecutivos están en bancos consecutivos
- o Ejemplo: Ubicación de bloques en una cache con 4 bancos con entrelazamiento de orden bajo

Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

© 2007 Elsevier, Inc. All rights reserved.

# Optimizaciones para aumentar el ancho de banda

---

## ❑ Cache segmentada

- o Segmentar los accesos a la cache permite aumentar el ancho de banda.
- o Problema: incremento de los ciclos de latencia (ver evolución caches). Más ciclos de reloj entre el lanzamiento de un LD y el uso de los datos que el LD proporciona
- o Más problemas: Mayor penalización en los saltos mal predichos
- o Ejemplos: N° de etapas del acceso a la cache en diferentes procesadores
  - Pentium 1 etapa
  - De Pentium Pro a Pentium III 2 etapas
  - Pentium 4 4 etapas

## Caches Resumen (I)

Técnica	Tasa fallos	Penal fallo	Tiempo acierto	Ancho banda	Coste HW / Complejidad	Comentario
Aumento tamaño de bloque	+	-			0	Trivial. L2 de Pentium 4 usa 128 bytes
Aumento asociatividad	+		-		1	Ampliamente usado
Aumento tamaño de Mc	+		-		1	Ampliamente usado, especialmente en L2
Mejora algoritmo reemplazamiento	+		-		1	LRU (o pseudo) bastante usado
Cache de víctimas	+	-			1	Bastante sencillo
Optimización del compilador	+				0	El software presenta oportunidades de mejora. Algunos computadores tienen opciones de optimización
Prebúsqueda HW	+	+			2 instr., 3 data	Muchos procesadores prebuscan instrucciones. AMD Opteron y Pentium 4 prebuscan datos.
Prebúsqueda SW	+	+			3	Necesita cache no bloqueante. En muchas CPUs.

## Caches Resumen (II)

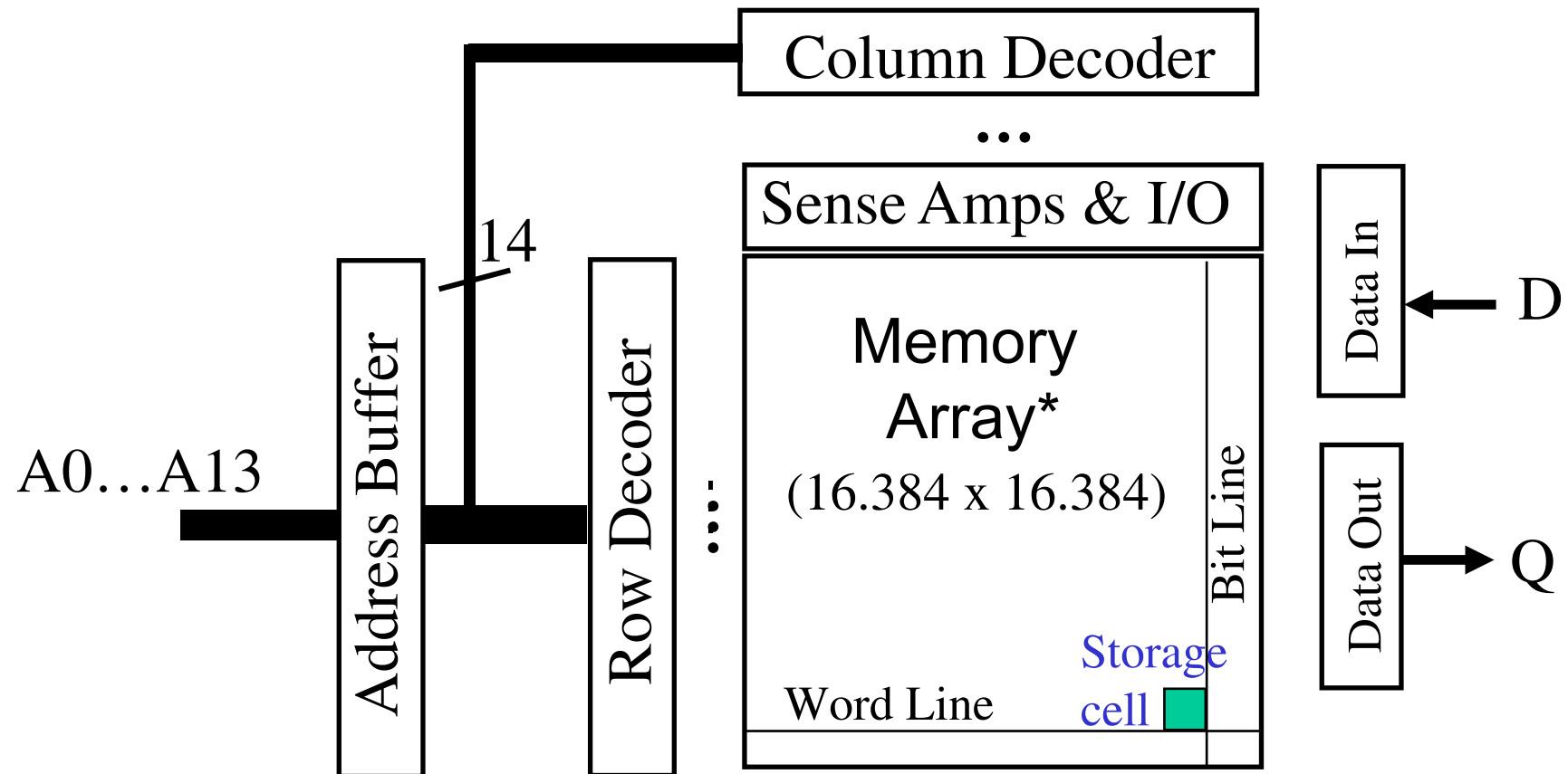
Técnica	Tasa fallos	Penal fallo	Tiempo acierto	Ancho banda	Coste HW / Complejidad	Comentario
Prioridad a las lecturas		+			1	Ampliamente usado
Prioridad a la palabra crítica		+			2	Ampliamente usado
Fusión de buffers de escritura		+			1	Ampliamente usado con write through
Cache multinivel		+			2	Ampliamente usado. Más complejo si tamaño de bloque en L1 y L2 distintos.
Cache pequeña y sencilla	-		+		0	Trivial; ampliamente usado.
Ocultar latencia traducción DV => DF			+		1	Ampliamente usado
Predicción de vía			+		1	Usado en Pentium 4
Cache de trazas			+		3	Usado en Pentium 4
Cache no bloqueante		+		+	3	Ampliamente usado
Cache multibanco				+	1	En L2 de Opteron y Niagara
Cache segmentada			-	+	1	Ampliamente usado



## ❑ Conceptos básicos

- Rendimiento
  - ✓ **Latencia**: Penalización del fallo
    - *Tiempo de acceso*: tiempo entre la petición y la llegada del dato
    - *Tiempo de ciclo*: tiempo entre peticiones sucesivas
  - ✓ **Ancho de Banda**: Penalización de fallo de bloques grandes (L2)
- Construida con **DRAM**: Dynamic Random Access Memory (2007 2Gb)
  - ✓ Necesita ser refrescada periódicamente (2 a 8 ms, <5% tiempo)
  - ✓ Direccionamiento en dos fases (El chip es una matriz de celdas 2D):
    - **RAS** : Row Access Strobe
    - **CAS** : Column Access Strobe
  - ✓ Tiempo ciclo doble tiempo de acceso( 2000; 40ns acceso, 90ns ciclo)
- Cache usan **SRAM**: Static Random Access Memory (2007 64Mb)
  - ✓ No necesitan refresco
  - ✓ **Tamaño**: 4-6 transistores/bit vs. 1 transistor
  - ✓ **Capacidad**: DRAM/SRAM - 4-8,
  - ✓ **Costo**: SRAM/DRAM - 8-16
  - ✓ **Tiempo de acceso**: SRAM/DRAM - 8-16

## DRAM: organización lógica (256 Mbit)



\* Puede estar internamente organizado como varios módulos

# DRAM: Mejora del rendimiento

---

## 1. Fast Page mode

- o Añade señales de timing para permitir repetir los accesos al row buffer sin un nuevo acceso al array de almacenamiento.
- o Este buffer existe en el array de almacenamiento y puede tener un tamaño de 1024 a 2048 bits.

## 2. Synchronous DRAM (SDRAM)

- o Añade una señal de reloj al interfase de la DRAM, de manera que transferencias sucesivas no necesitan sincronizar con el controlador de la DRAM.

## 3. Double Data Rate (DDR SDRAM)

- o Transfiere datos en ambos flancos de la señal de reloj de la DRAM  $\Rightarrow$  dobla el ancho de banda ( peak data rate)
- o DDR2 reduce consumo, reduciendo el voltaje desde 2.5 a 1.8 volts + trabaja a mayor frecuencia hasta 400 MHz
- o DDR3 baja el voltaje a 1.5 volts + mayor frecuencia hasta 800 MHz

❑ Mejora en AB, no en latencia

# DDR SDRAM: AB y nomenclatura

DRAM: el nombre se basa en AB de chip (Peak Chip Transfers / Sec)

DIMM: el nombre se basa en AB del DIMM (Peak DIMM MBytes / Sec)

Stan- dard	Clock Rate (MHz)	M transfers / second	DRAM Name	Mbytes/s/ DIMM	DIMM Name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800

x 2

x 8

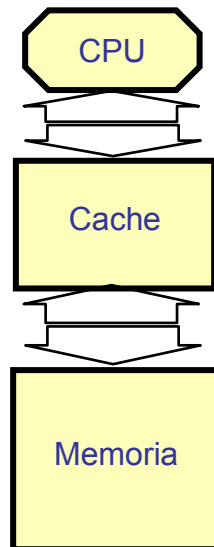
# AB entre Memoria principal y Mc

- ❑ **Objetivo:** Aumento del ancho de banda en la transferencia de un bloque manteniendo la misma latencia. (Permite aumento del tamaño de bloque sin gran impacto en miss penalty).

- o Ejemplo: 4 ciclos en enviar la dirección, 24 ciclos en el acceso y 4 ciclos en el envío del dato. Tamaño de palabra 4 bytes.

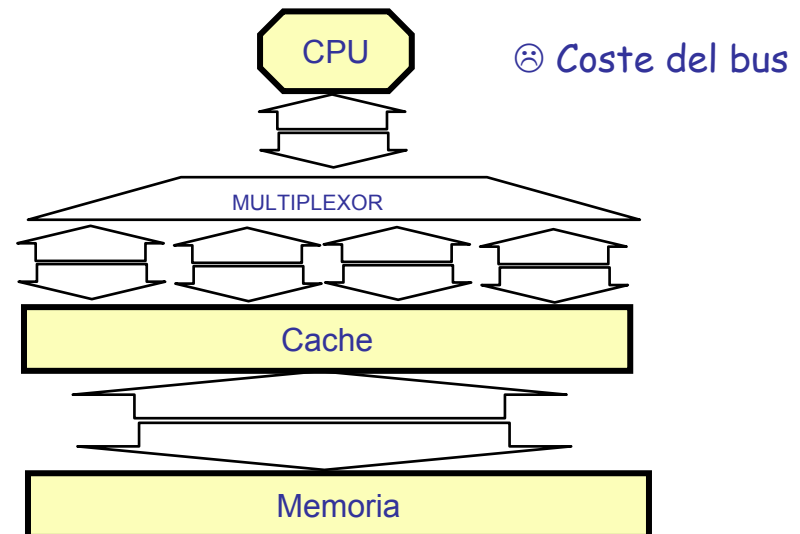
- o Cache: Bloques de tamaño 4 palabras (= 16 bytes)

**BUS Y MEMORIA DE 1 PALABRA  
ACCESO SECUENCIAL**



Miss penalty =  $4 \times (4+24+4) = 128$  ciclos  
AB =  $16 \text{ bytes} / 128 \text{ ciclos} = 0,125 \text{ bytes/ciclo}$

**BUS Y MEMORIA DE 4 PALABRAS  
ACCESO SECUENCIAL (256,512)**

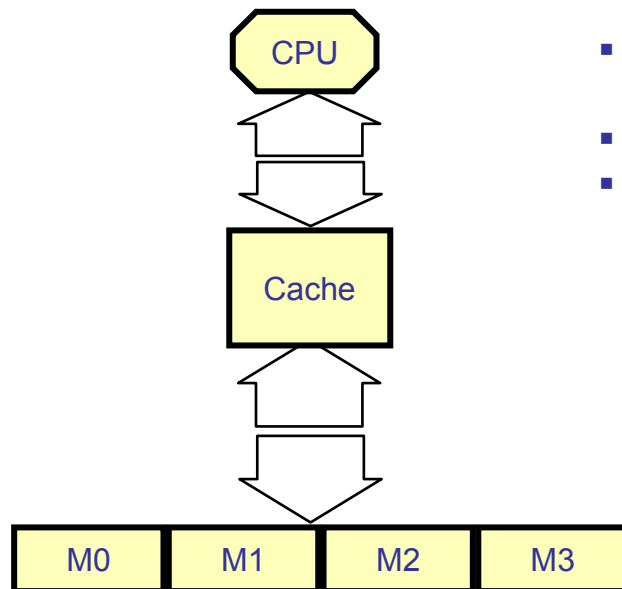


Miss penalty =  $4+24+4 = 32$  ciclos  
AB =  $16 \text{ bytes} / 32 \text{ ciclos} = 0,5 \text{ bytes/ciclo}$

# AB entre Memoria principal y Mc

## ENTRELAZAMIENTO de orden bajo

ANCHURA DE BUS DE 1 PALABRA  
4 MÓDULOS DE MEMORIA DE 1 PALABRA ENTRELAZADOS  
ACCESO SOLAPADO A LOS MÓDULOS



- Se envía una misma dirección de palabra (N-2 bits) a los 4 módulos: 4 ciclos
- Se accede a los cuatro módulos en paralelo: 24 ciclos
- Cada módulo proporciona una palabra a través del bus: 4x4 ciclos

Miss penalty: 44 ciclos

AB: 16 bytes/44 ciclos = 0,36 bytes ciclo

☺ Muy buena **relación coste/rendimiento**

Funciona bien con accesos secuenciales.( Reemplazamiento de un bloque)  
No soluciona el problema de accesos independientes

# AB entre Memoria principal y Mc

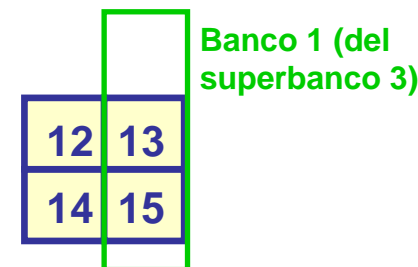
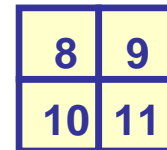
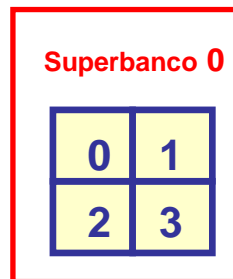
## BANCOS INDEPENDIENTES (superbancos): entrelazamiento mixto

- o En cache no bloqueante, varios fallos de cache tienen que poderse servir simultáneamente
- o Organizar la memoria con entrelazamiento de orden bajo no soluciona el problema
- o Se puede solucionar mezclando los tipos de entrelazamiento (entrelazamiento mixto)
- o Cada banco necesita interfaz con el sistema. Controlador, buses separados de direcciones y datos

Numero de superbanco	Desplazamiento superbanco	
	Desplaz. banco	Numero de banco

**Ejemplo:** Memoria de 16 palabras  
4 accesos independientes  
Cada acceso dos palabras

0 a 3 (2bits)	0 a 1 (1bit)	0 a 1 (1bit)
---------------	--------------	--------------



## Visión global: AMD Opteron

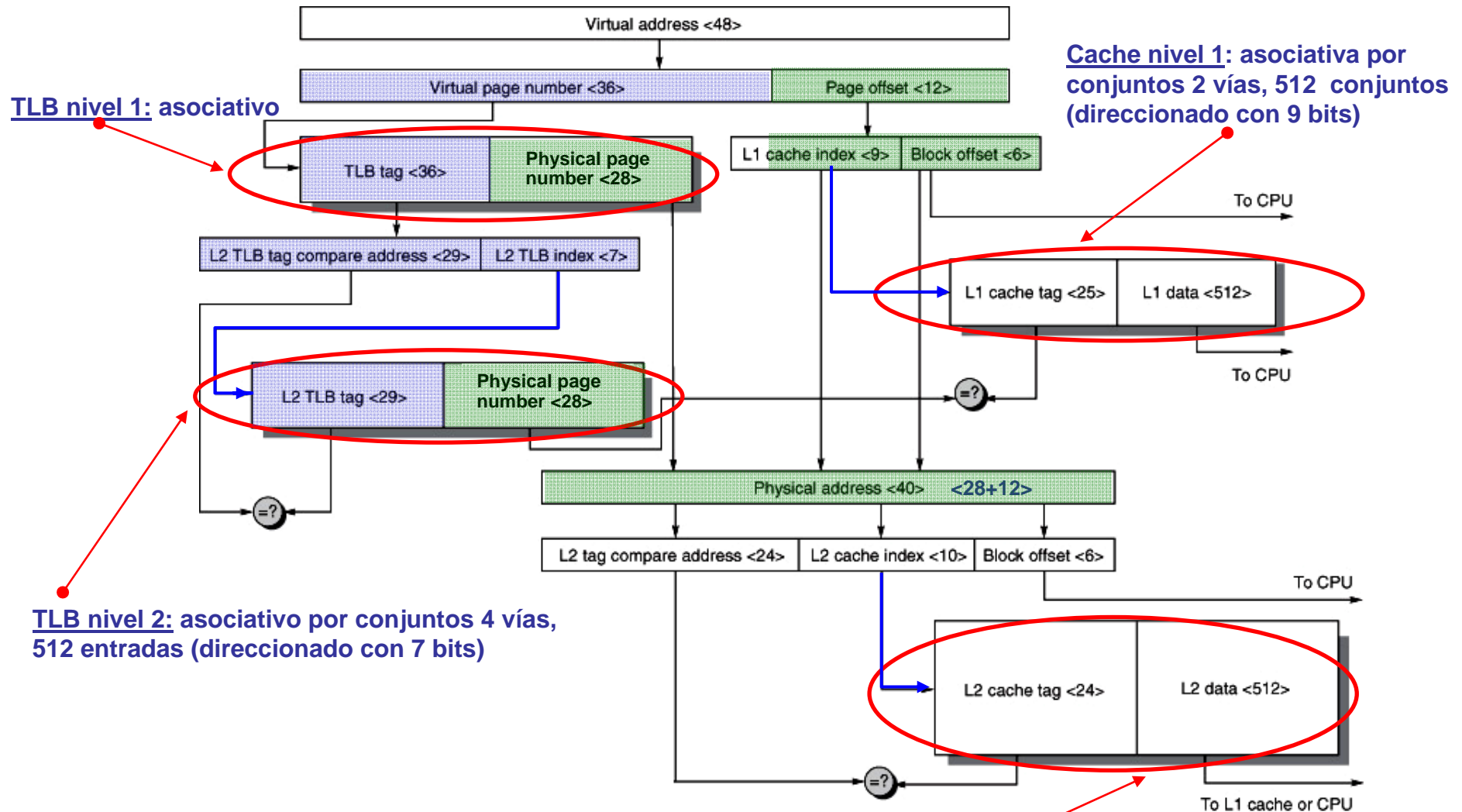
---

- ❑ Pipeline entero de 12 etapas, con max. frecuencia de reloj 2,8 GHz y velocidad de memoria hasta PC3200 DDR SDRAM (2006)
- ❑ Dirección virtual: 48 bits; dirección física 40 bits
- ❑ Cache L1: I-Cache y D-Cache. Cada una: 64 KB, 2 vías, bloque 64 bytes, LRU. (Nº conjuntos = 512).
- ❑ Cache L2: 1 MB, 16 vías, bloque 64 bytes, pseudo LRU. (Nº conjuntos = 1024)
- ❑ Política de escritura en L2 y D-Cache L1: Post-escritura, con asignación en escritura
- ❑ Caches de L1 virtualmente accedidas, físicamente marcadas
- ❑ TLBs separados para instrucciones y datos, organizados en dos niveles
  - o I-TLB nivel 1 y D-TLB nivel 1: asociativo, 40 entradas
  - o I-TLB nivel 2 y D-TLB nivel 2: 4 vías, 512 entradas
- ❑ Controlador de memoria gestiona hasta 10 fallos de cache: 2 de I-Cache y 8 de D-Cache



# Visión global: AMD Opteron

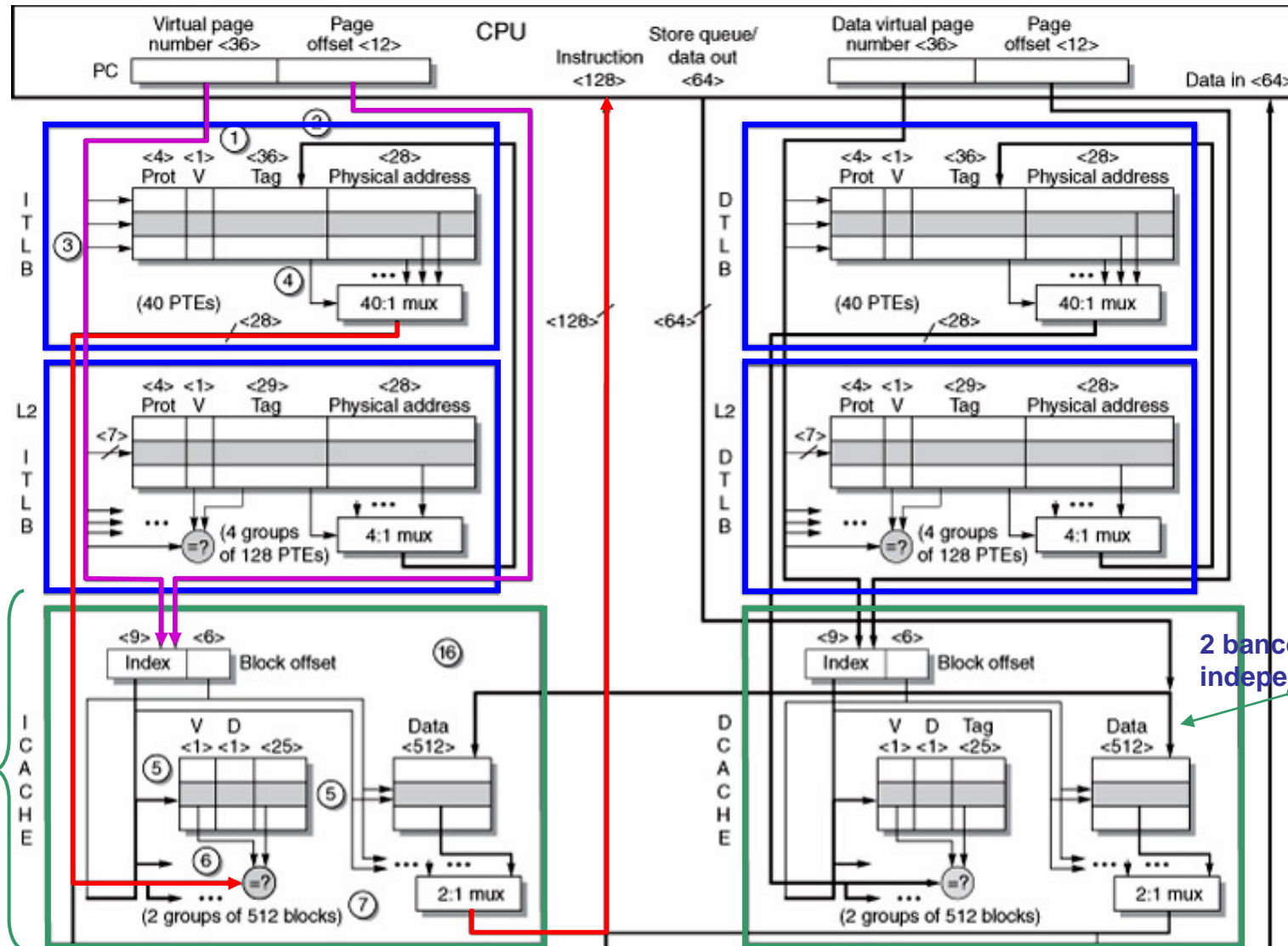
## ❑ Caches L1 y L2, junto con la jerarquía de TLBs



© 2007 Elsevier, Inc. All rights reserved.

# Visión global: AMD Opteron

## Esquema de la jerarquía de memoria (I)



# Visión global: AMD Opteron

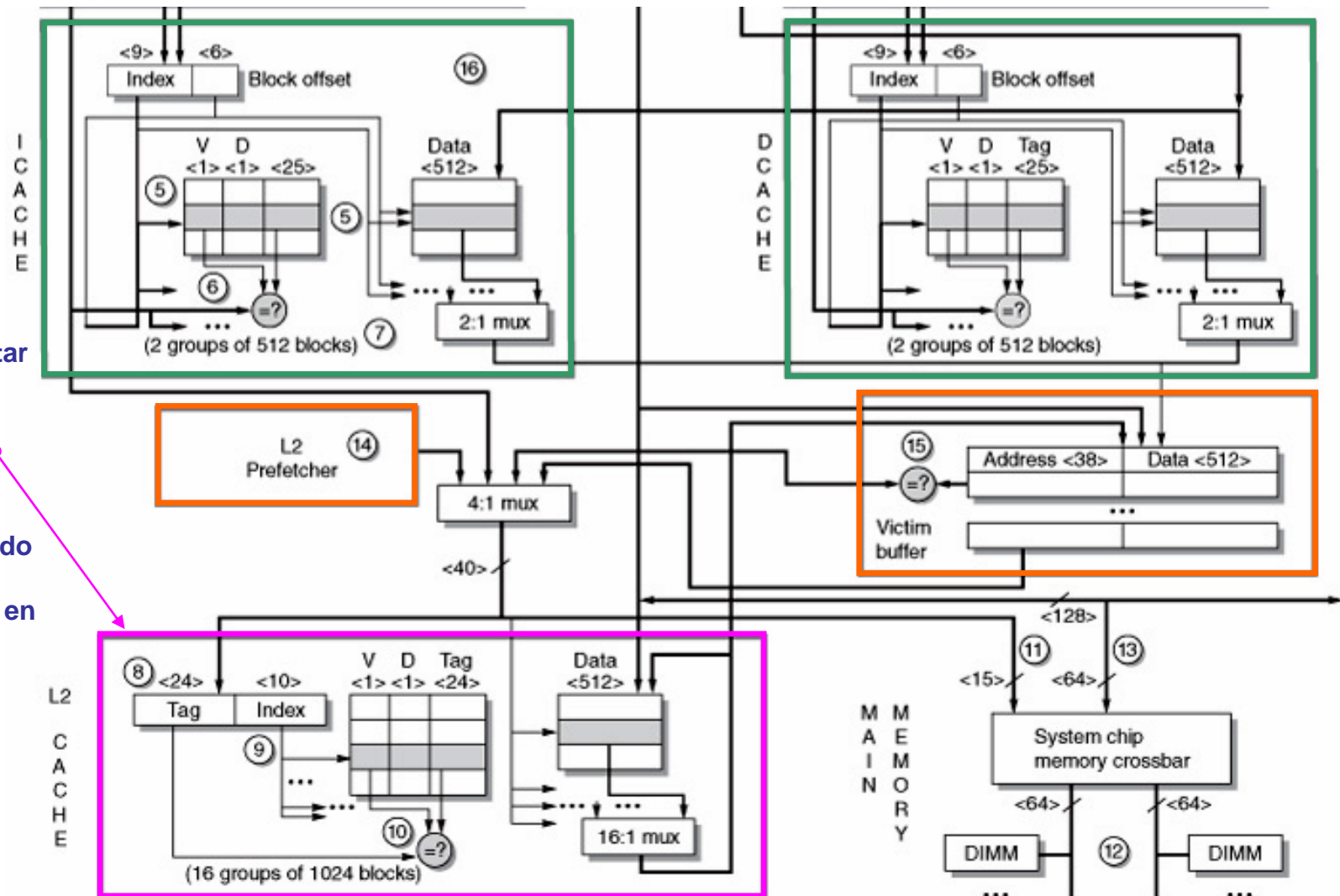
## ❑ Esquema de la jerarquía de memoria (II)

- Fallo L1/ acierto  
L2: 7 ciclos (abortar  
acceso a Mp)

- L2 no inclusiva

-Fallo en L1 y L2:  
Bloque referenciado  
se lleva sólo a L1.  
Bloque eliminado en  
L1 se lleva a L2

- Latencia instr.  
obtenida en Mp >  
100 ciclos



© 2007 Elsevier, Inc. All rights reserved.

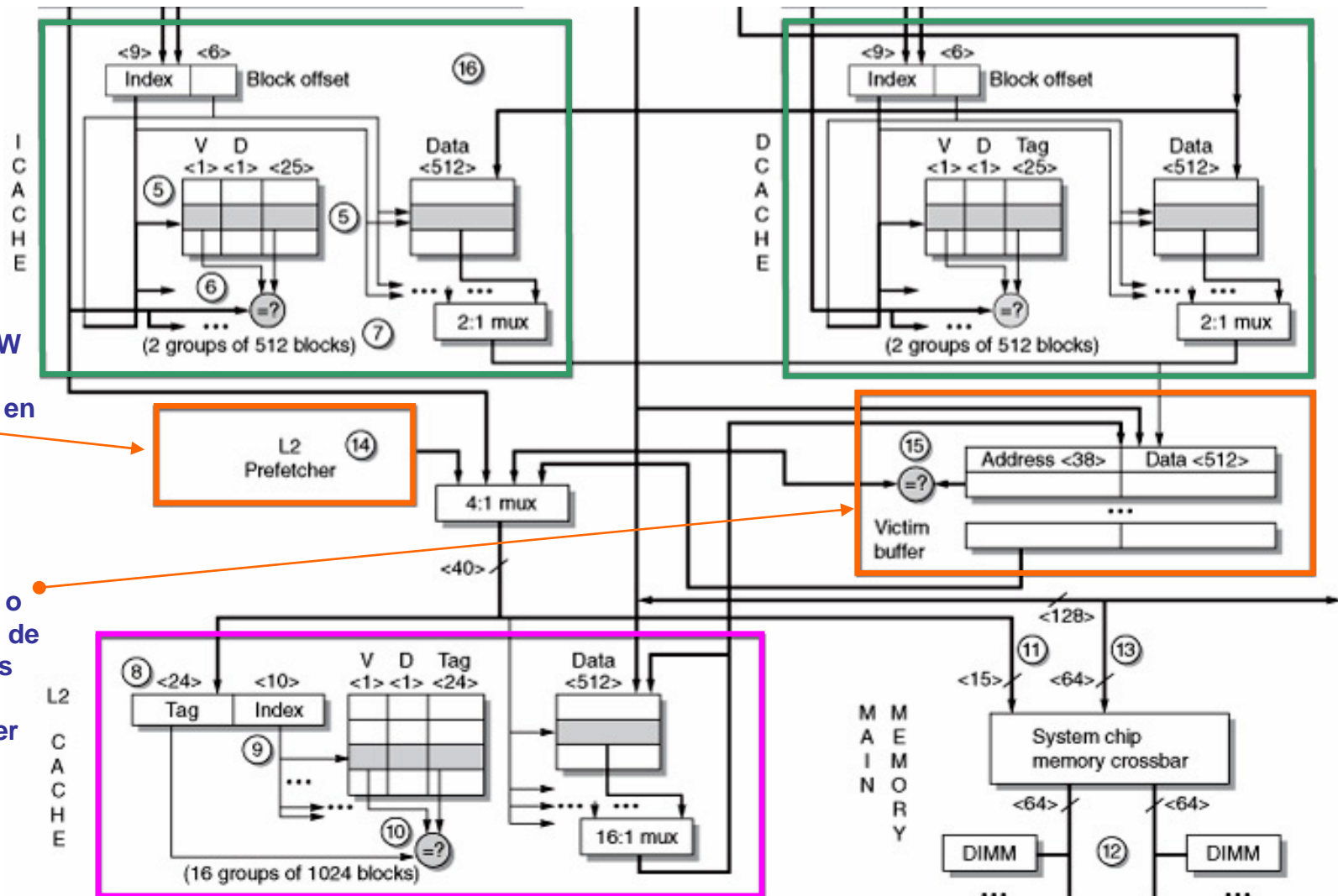
# Visión global: AMD Opteron

## ❑ Esquema de la jerarquía de memoria (III)

- Prebúsqueda HW asociada con L2: bloque siguiente en secuencia ascendente o descendente

- Escritura de bloque desde D\$ o L2 a Mp: a través de buffer de víctimas

- Capacidad buffer de víctimas: 8 bloques

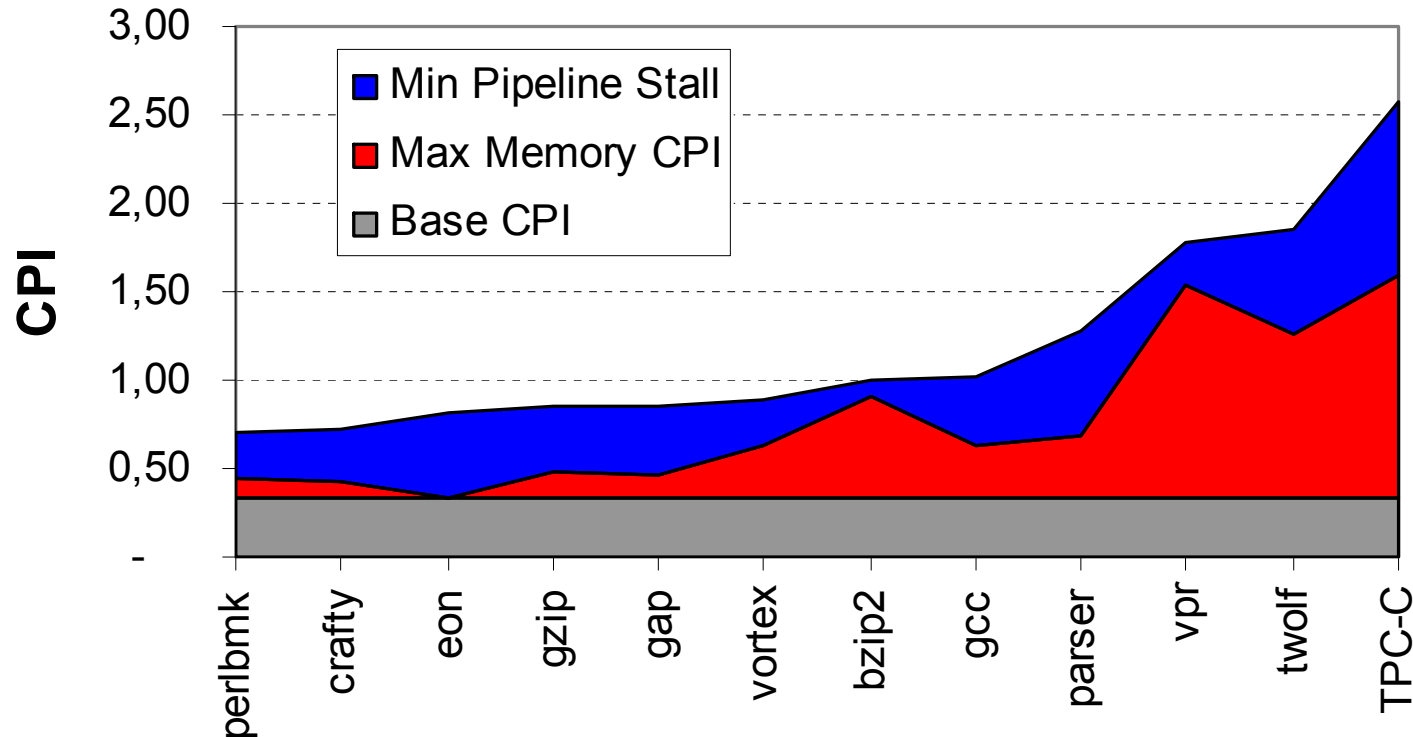


© 2007 Elsevier, Inc. All rights reserved.

## Visión global: AMD Opteron

### □ Rendimiento: efecto de la jerarquía de memoria

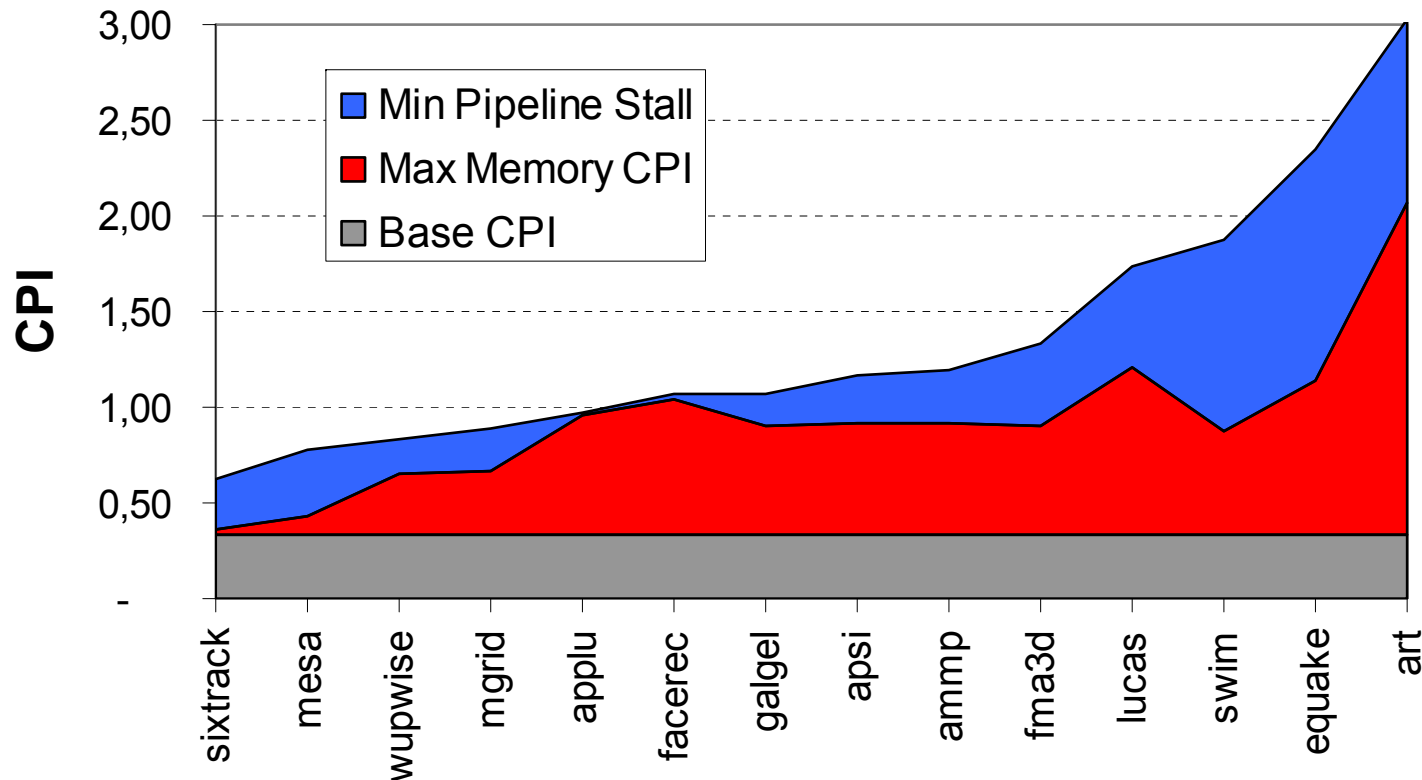
- o CPI real medido para SPECint 2000 + TPC-C



- o CPI base (Opteron lanza 3 instr/ciclo => CPI base es 0,33)
- o Max Memory CPI (suponiendo no solapamiento con paradas del procesador).  
Estimación: (Fallos/instrucción en diferentes niveles) x (Penalización de cada nivel)
- o Min Pipeline Stall:  $\text{CPI real} - (\text{CPI base} + \text{Max Memory CPI})$
- o La diferencia entre CPI real y base que es atribuible a la memoria (zona roja) ronda el 50% en promedio.

## Visión global: AMD Opteron

- Rendimiento: efecto de la jerarquía de memoria
  - o CPI real medido para SPECfp 2000



- o La penalización en el CPI atribuible a la jerarquía de memoria es algo mayor que en el caso de los programas enteros (alrededor del 60%)

## Comparación: AMD Opteron - Intel Pentium 4

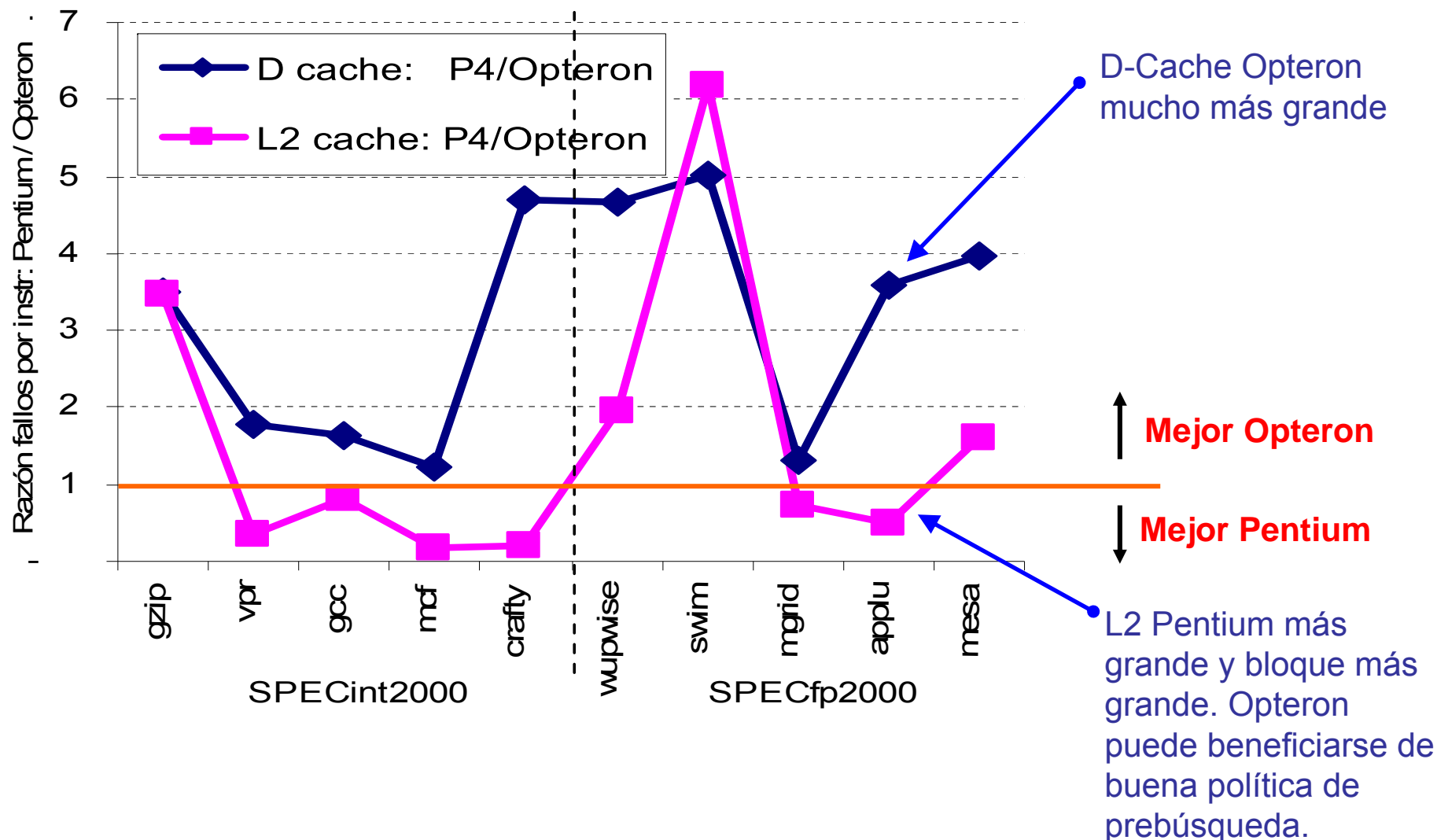
CPU	Pentium 4 (3.2 GHz*)	Opteron (2.8 GHz*)
I-Cache	Trace Cache (8K micro-ops)	Asoc. cjtos. 2-vías, 64 KB, bloque 64B
D-Cache	Asoc. cjtos. 8-vías, 16 KB, bloque 64B	Asoc. cjtos. 2-vías, 64 KB, bloque 64B
Cache L2	Asoc. cjtos. 8-vías, 2 MB, bloque 128B, inclusiva de D-Cache	Asoc. cjtos. 16-vías, 1 MB, bloque 64B, exclusiva de D-Cache
Prebúsqueda	8 streams a L2	1 stream a L2
Memoria	200 MHz x 64 bits	200 MHz x 128 bits

\* Frecuencias referidas al año 2005. Hay versiones posteriores.



## Comparación: AMD Opteron - Intel Pentium 4

- ❑ Cociente entre fallos por instrucción Pentium y fallos por instrucción Opteron (Fallos en D-Cache y L2)





## Otros ejemplos

### ❑ IBM Power 5-6

	Power 5	Power 6
L1 Instrucciones	64KB, directa, 128 bytes, read only	64KB, 4 ways, 128 bytes, read only
L1 Datos	32KB, 2 ways, 128 bytes, write-through	64KB, 8 ways, 128 bytes, write-through
L2	Compartida, 1,92MB, 10ways, 128 bytes, pseudoLRU, copy-back	Propietaria de cada core, 4MB, 8ways, 128 bytes, pseudoLRU, copy-back
L3 Off-chip	Compartida, Victimas 36MB, 12 ways, 256 bytes, pseudoLRU	Compartida, Victimas 32MB, 16 ways, 128 bytes, pseudoLRU

## Otros ejemplos

---

### □ Sun Niagara I-II

	Niagara I	Niagara II
L1 Instrucciones	16KB, 4 ways, 32 bytes, read only, aleatorio	16KB, 8 ways, 32bytes, read only, aleatorio
L1 Datos	8KB, 4 ways, 16 bytes, write-through, aleatorio	8KB, 8 ways, 16 bytes, write-through, aleatorio
L2	Compartida, 3MB, 12ways, 64 bytes, copy-back	compartida, 4MB, 16ways, 64 bytes, copy-back

## Otros ejemplos

### □ Intel CORE2 - AMD Opteron

	Opteron	Core 2
L1 Instrucciones	64KB, 2 ways, 64bytes, read only, LRU	32KB, 8 ways, 64 bytes, read only,
L1 Datos	64KB, 2 ways, 64bytes, LRU, copy back, no inclusiva	32KB, 8 ways, 64 bytes, write-through
L2	Compartida, 1MB, 16ways, 64bytes, pseudoLRU, copy-back	Compartida, 4MB, 16ways, o 6MB 24 ways, 128 bytes, copy-back