

Arquitectura de Computadoras

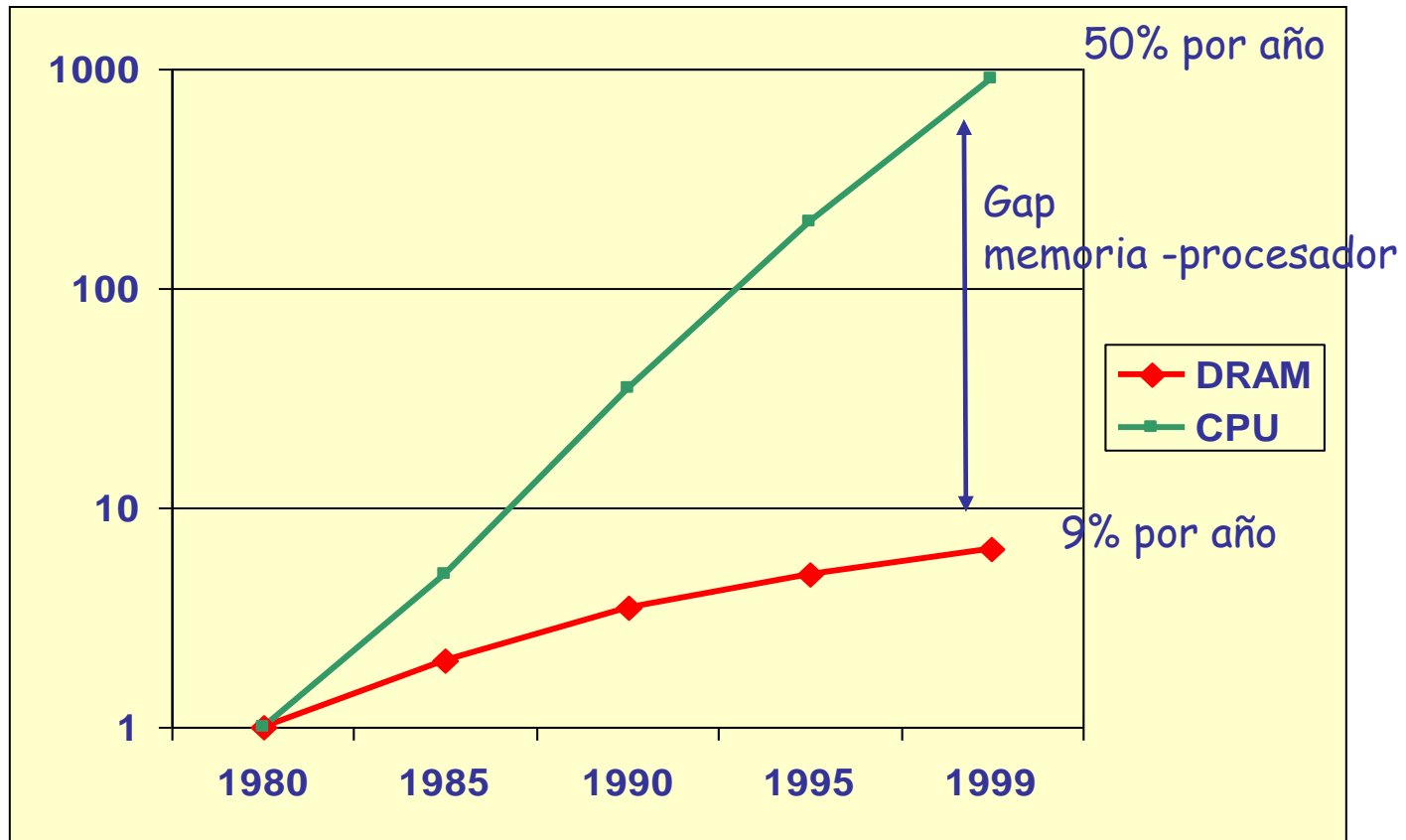
Tema 5

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

2016-2020

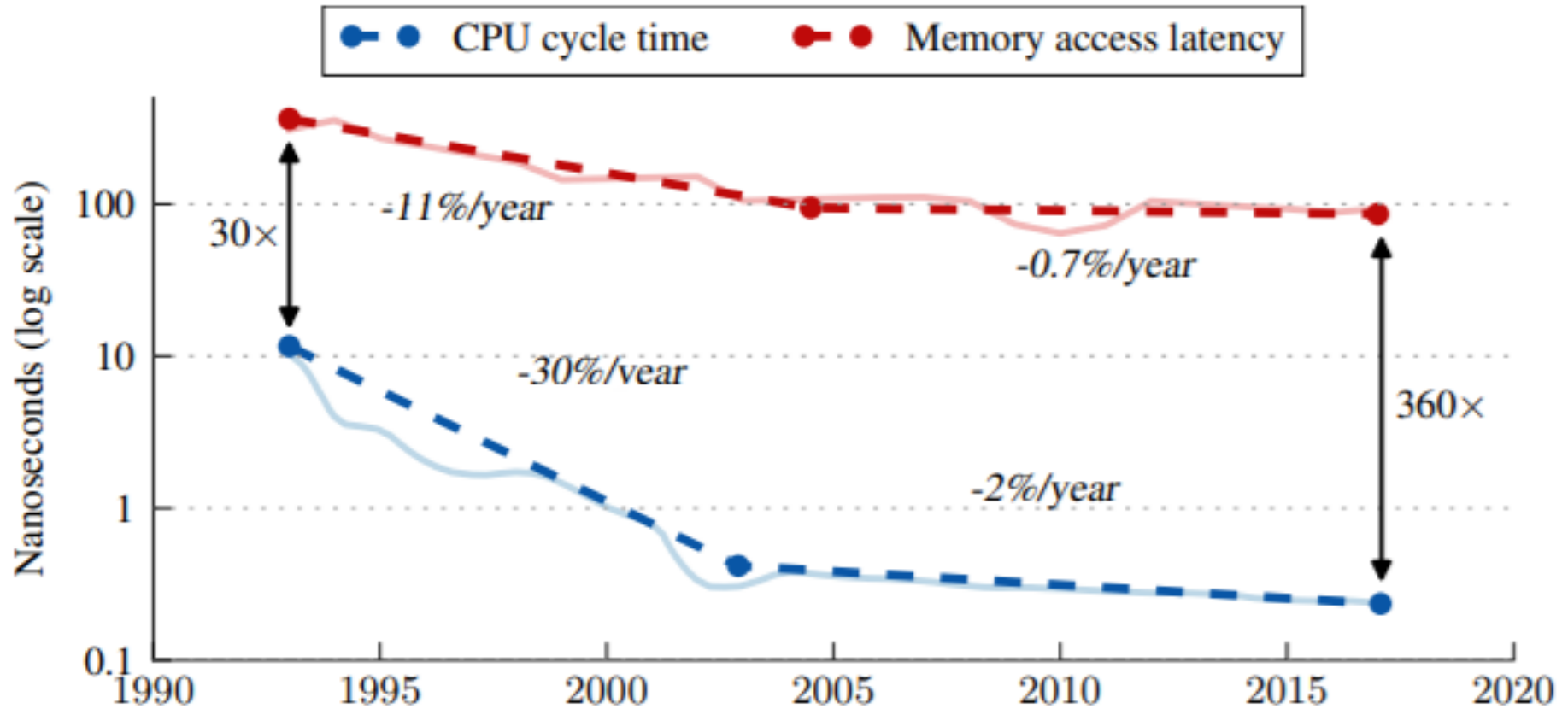
- o Introducción: Jerarquía de memoria
- o Rendimiento: mejoras
- o Reducir la tasa de fallos de la cache
- o Reducir la penalización de los fallos de cache
- o Reducir el tiempo de acceso a la cache
- o La memoria principal
- o Una visión global: Alpha 21064
- o Bibliografía
 - o Capítulo 5 de [HePa07]
 - o [JaMu98a] B. Jacob, T. N. Mudge. "Virtual Memory: Issues of Implementation". IEEE Computer Magazine. Vol. 31 (6), Junio 1998,
 - o Simulador X-CACHE

□ El problema



➤ La demanda de anchura de banda con memoria crece.

Memory Wall

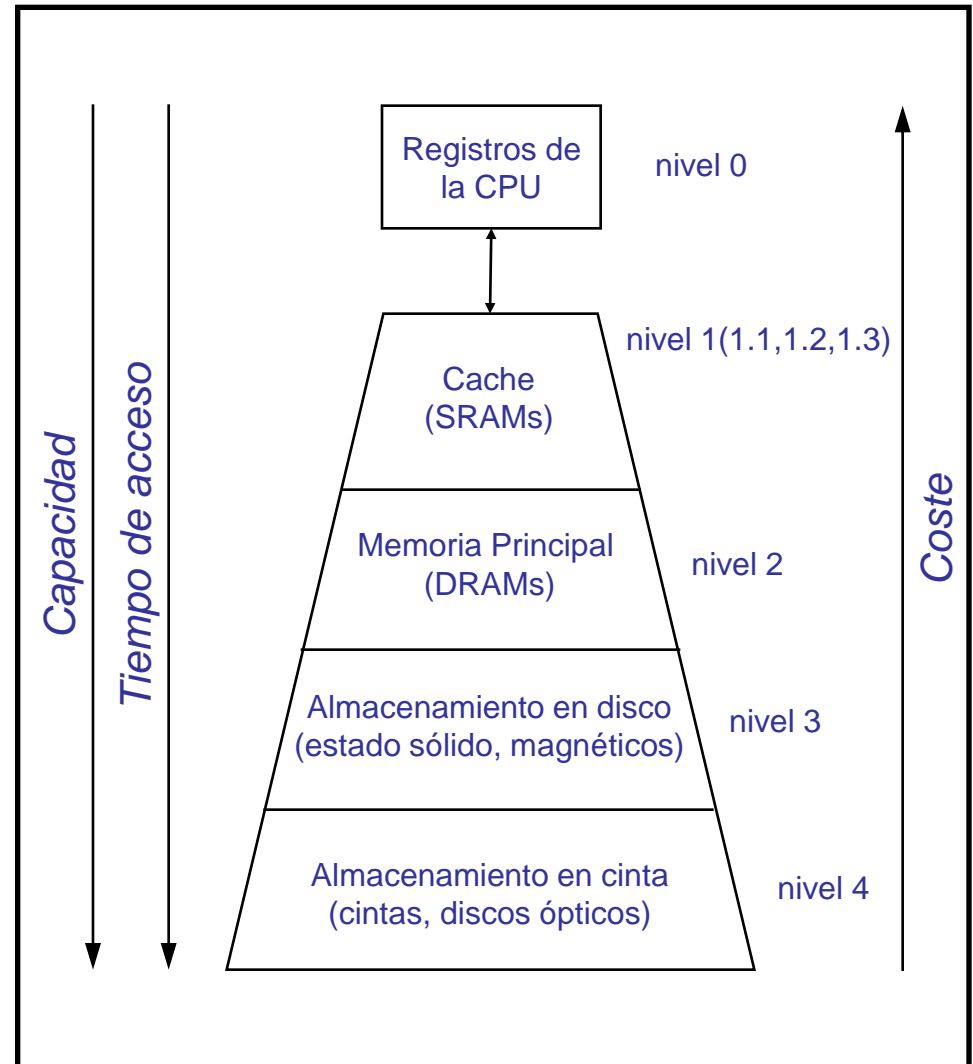


Creciente discrepancia entre la latencia de acceso a la memoria principal y el tiempo de ciclo de la CPU, durante los últimos 25 años.

Recientemente la tendencia a la baja en el tiempo de ciclo de la CPU disminuyó al 2%, mientras que el rendimiento de los procesadores individuales mejora al 3,5% por año. La disminución de la latencia de acceso a la memoria es inferior al 1%.

□ 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 CD-ROMs
- El coste de todo el sistema de memoria excede al coste de la CPU
 - ⇒ Es muy importante optimizar su uso



❑ Tipos de Memoria

Tipo	Tamaño	Velocidad	Costo/bit
Registros	< 1KB	< 0,5ns	\$\$\$\$
On-chip SRAM	8KB-6MB	< 10ns	\$\$\$
Off-chip SRAM	1Mb - 16Mb	< 20ns	\$\$
DRAM	64MB - 1TB	< 100ns	\$
Disco	40GB - 1PB	< 20ms	~0

Event	Latency
1 CPU cycle	0.3 ns
Level 1 cache access	0.9 ns
Level 2 cache access	2.8 ns
Level 3 cache access	12.9 ns
Main memory access (DRAM, from CPU)	120 ns
Solid-state disk I/O (flash memory)	50–150 µs
Rotational disk I/O	1–10 ms

❑ 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*
- La mayor parte de los accesos a un bloque de información, este bloque debe encontrarse en los niveles **altos** (bajos) de la memoria

Niveles a los que afecta la gestión de la jerarquía memoria

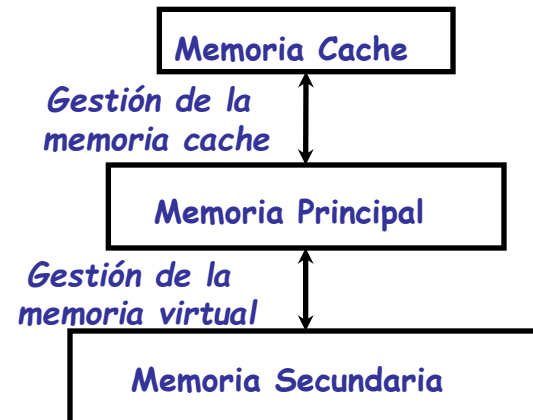
- Se refiere a la gestión dinámica, en tiempo de ejecución de la jerarquía de memoria
- Esta gestión de la memoria sólo afecta a los niveles 1 (cache), 2 (mem. principal) y 3 (mem. secund.)
 - ⇒ El nivel 0 (registros) lo asigna el compilador en tiempo de compilación
 - ⇒ El nivel 4 (cintas y CD-ROMs)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

Gestión de la memoria cache

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

Gestión de la memoria virtual

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



□ Propiedades de la jerarquía de memoria

✓ Inclusión

- 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

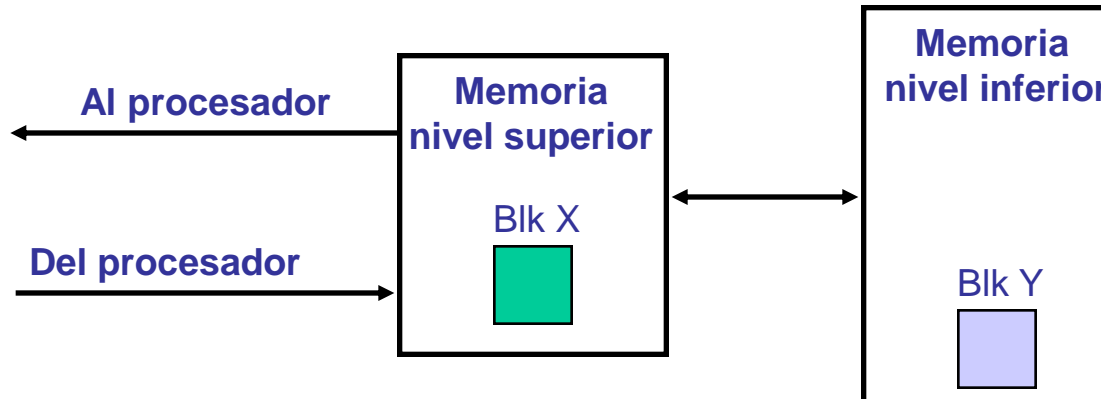
- Las copias de la misma información existentes en los distintos niveles deben ser consistentes
 - ⇒ Si un bloque de información se modifica en el nivel M_i , deben actualizarse los niveles M_{i+1} , ..., M_n
 - ⇒ Existen distintas estrategias de actualización
 - ✓ *Escritura directa (write-through)*: Cuando una palabra se modifica en el nivel M_i , inmediatamente se actualiza en el nivel M_{i+1}
 - ✓ *Post-escritura (write-back)*: La actualización del nivel M_{i+1} se retrasa hasta que el bloque que se modificó es reemplazado o eliminado en el nivel M_i

✓ Localidad

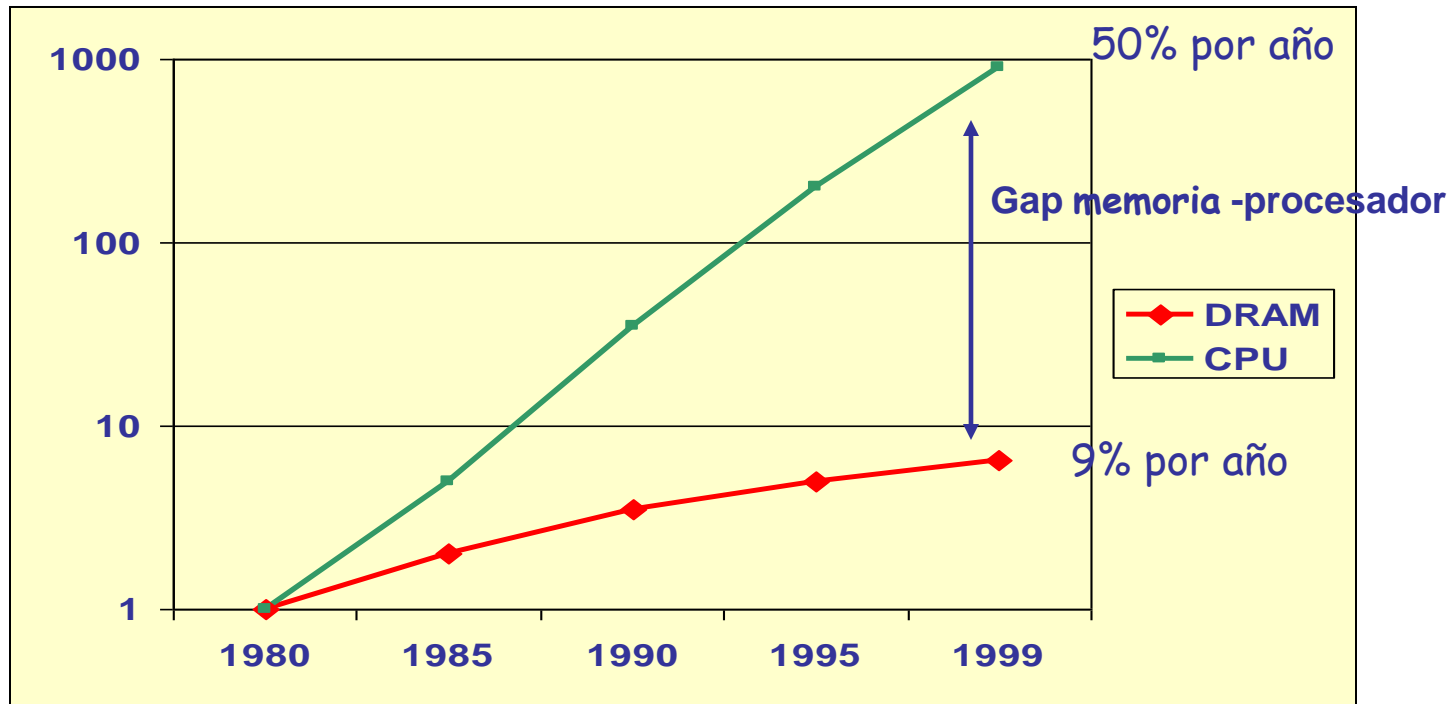
- La **propiedad de localidad** dice que las referencias a memoria generadas por la CPU, para acceso a datos o a instrucciones, están concentradas o agrupadas en ciertas regiones del tiempo y del espacio
- **Localidad temporal**
 - ⇒ Las direcciones de memoria (instrucciones o datos) recientemente referenciadas, serán referenciadas de nuevo, muy probablemente, en un futuro próximo
 - ⇒ Ejemplos: Bucles, subrutinas, accesos a pila, variables temporales, etc.
- **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.

❑ Terminología

- **Bloque:** unidad mínima de transferencia entre los dos niveles
- **Acierto (hit):** el dato solicitado está en el nivel i
 - ❖ *Tasa de aciertos (hit ratio):* la fracción de accesos encontrados en el nivel i
 - ❖ *Tiempo de acierto:* tiempo de acceso del nivel i + tiempo detección de acierto
- **Fallo (miss):** el dato solicitado no está en el nivel i y es necesario buscarlo en el nivel $i+1$
 - ❖ *Tasa de fallos (miss ratio):* $1 - (\text{Tasa de aciertos})$
 - ❖ *Tiempo de penalización por fallo:* tiempo de sustitución de un bloque del nivel i + tiempo de acceso al dato
- **Tiempo de acierto** \ll **Penalización de fallo**



□ El problema

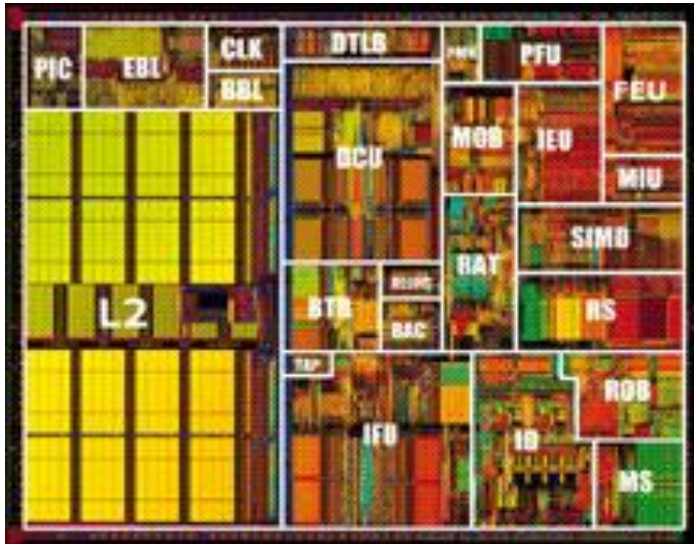


- La demanda de anchura de banda con memoria crece.
Segmentación, ILP
Ejecución especulativa
1980 no caches "on chip", 2007 2-3 niveles de cache "on chip"
- No tienen valor en sí mismas. Solo para el reducir el gap

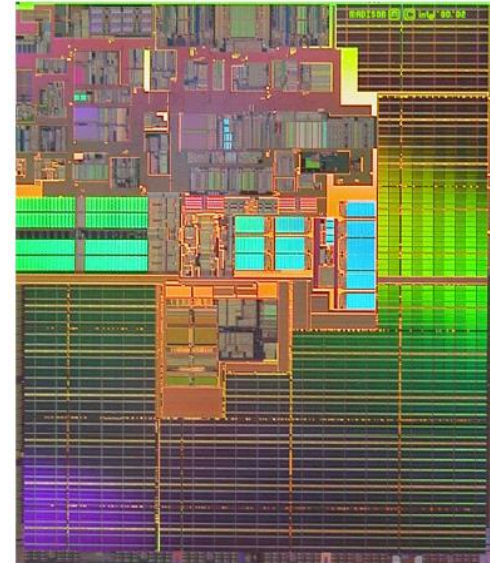
□ El problema: Algunos datos

□ Tamaño de la cache

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



PentiumIII



Itanium 2
Madison

□ 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

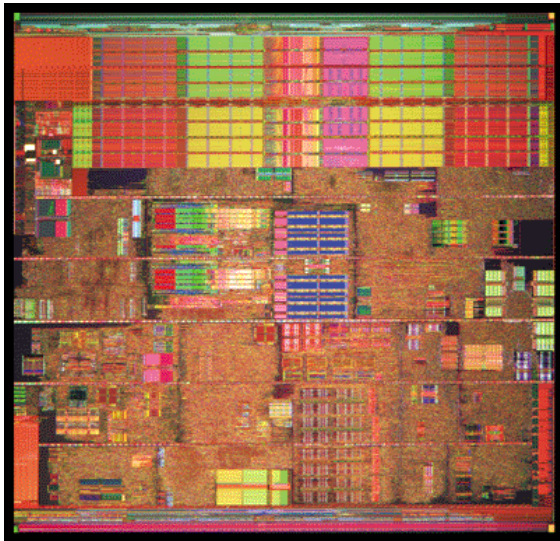
□ Latencia:

- ✓ 1ciclo (Itanium2) a 3 ciclos Power4-5

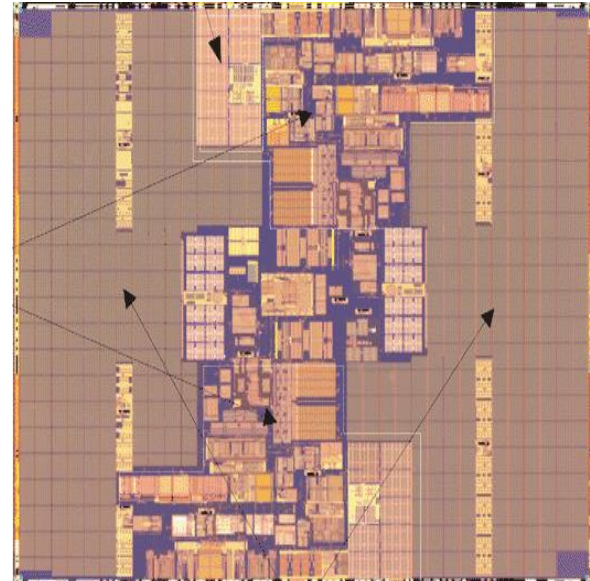
□ El problema

□ Tamaño de la cache

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

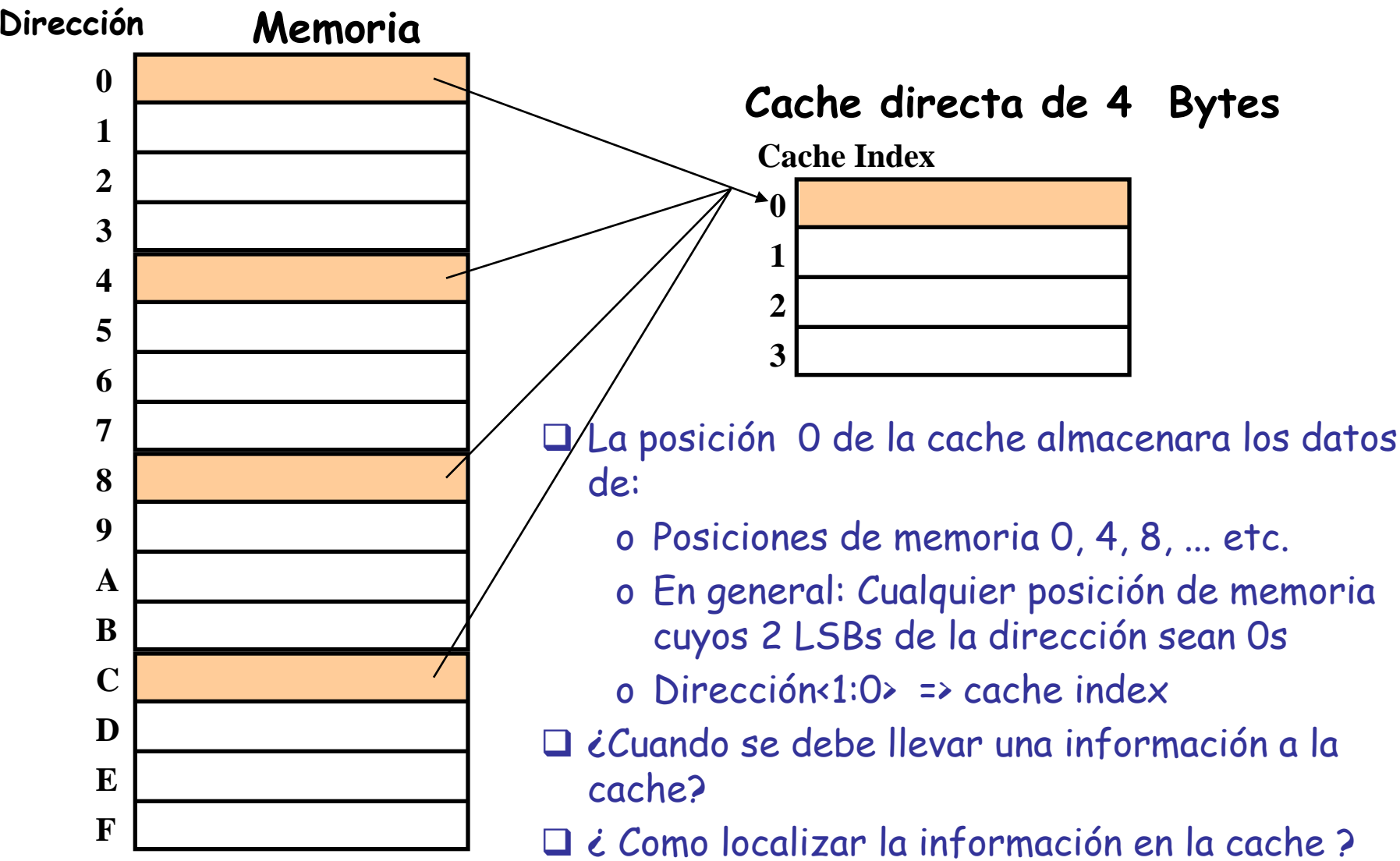


Pentium 4



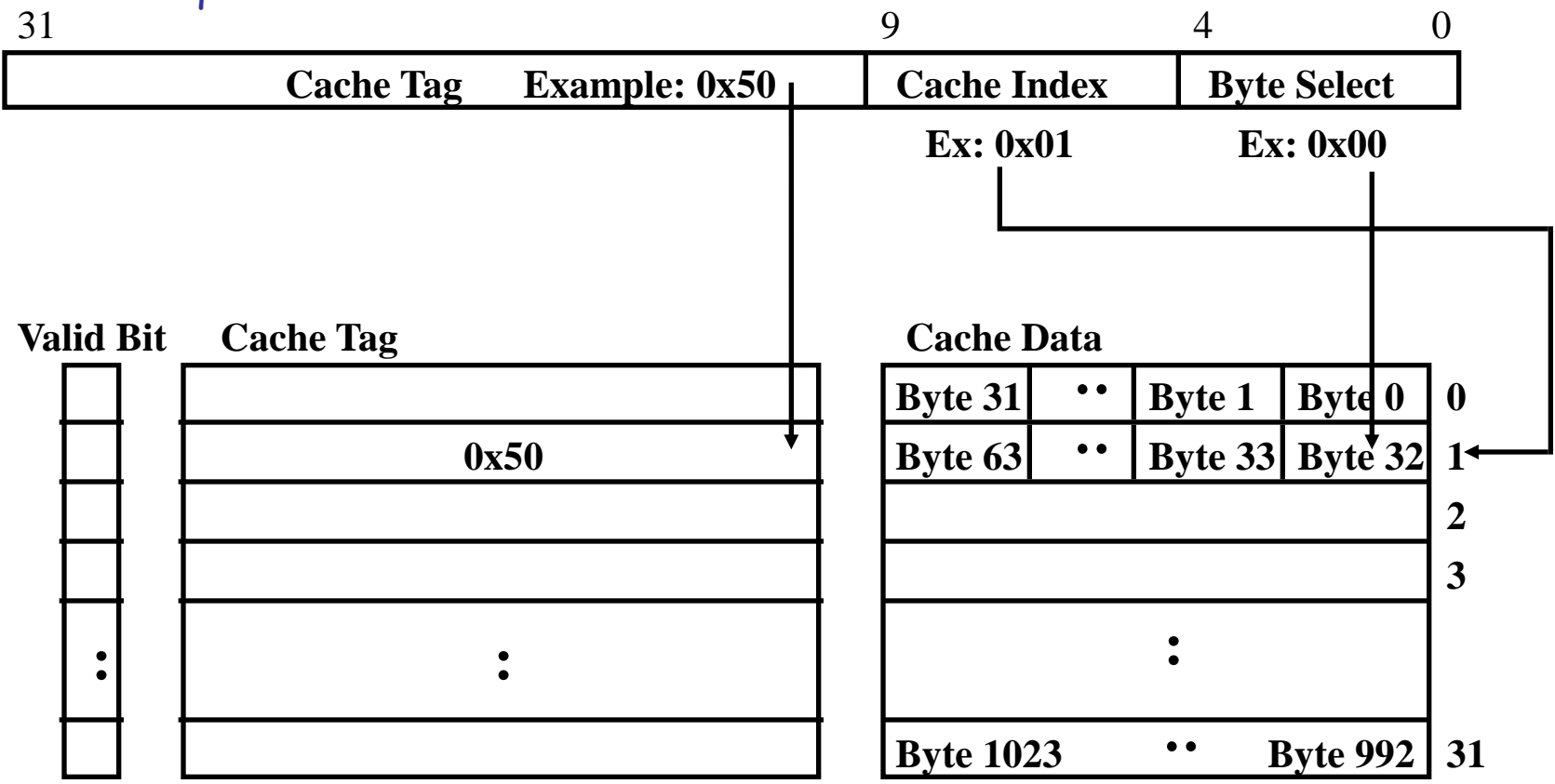
Montecito
1700Mtrans

❑ La más simple: Emplazamiento directo



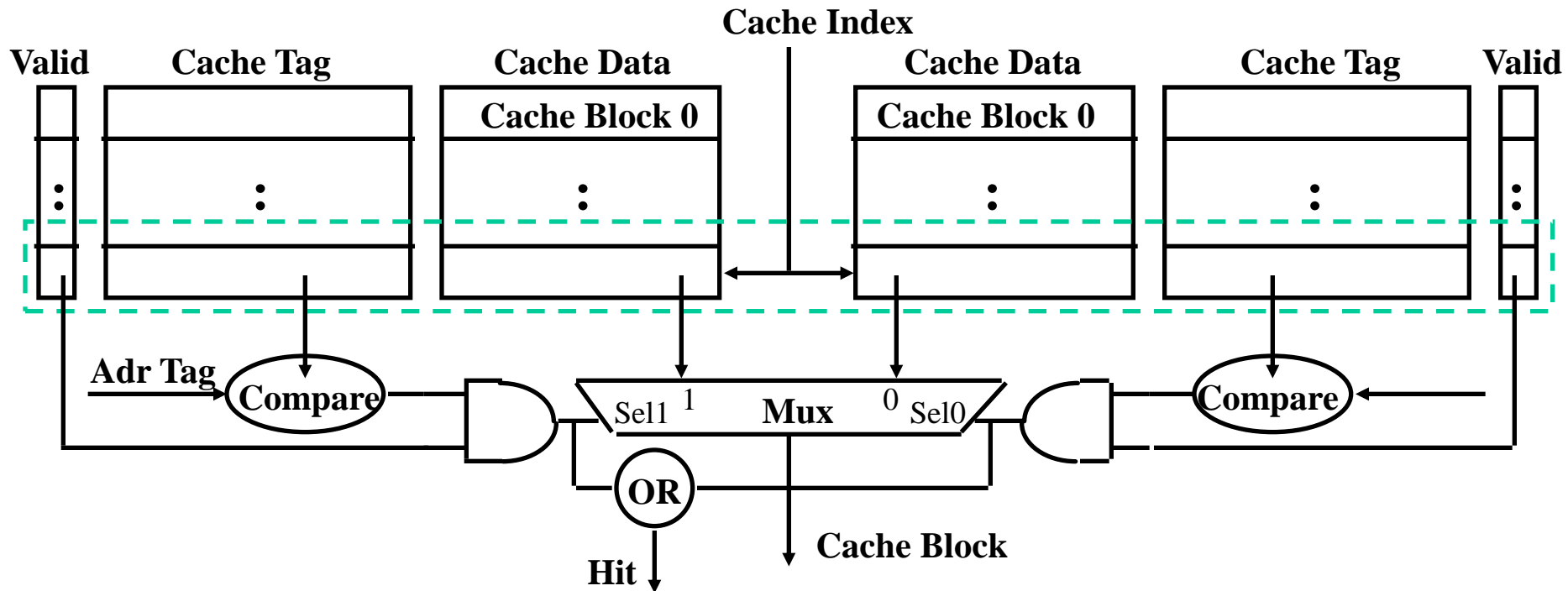
❑ Cache directa de 1 KB, 32Bytes por bloque

- ❑ Para una cache de 2^N con dirección de 32bits:
 - o Los $(32 - N)$ bits más significativos son el Tag
 - o Los M bits menos significativos son el selector de bytes (Tamaño de bloque = 2^M)



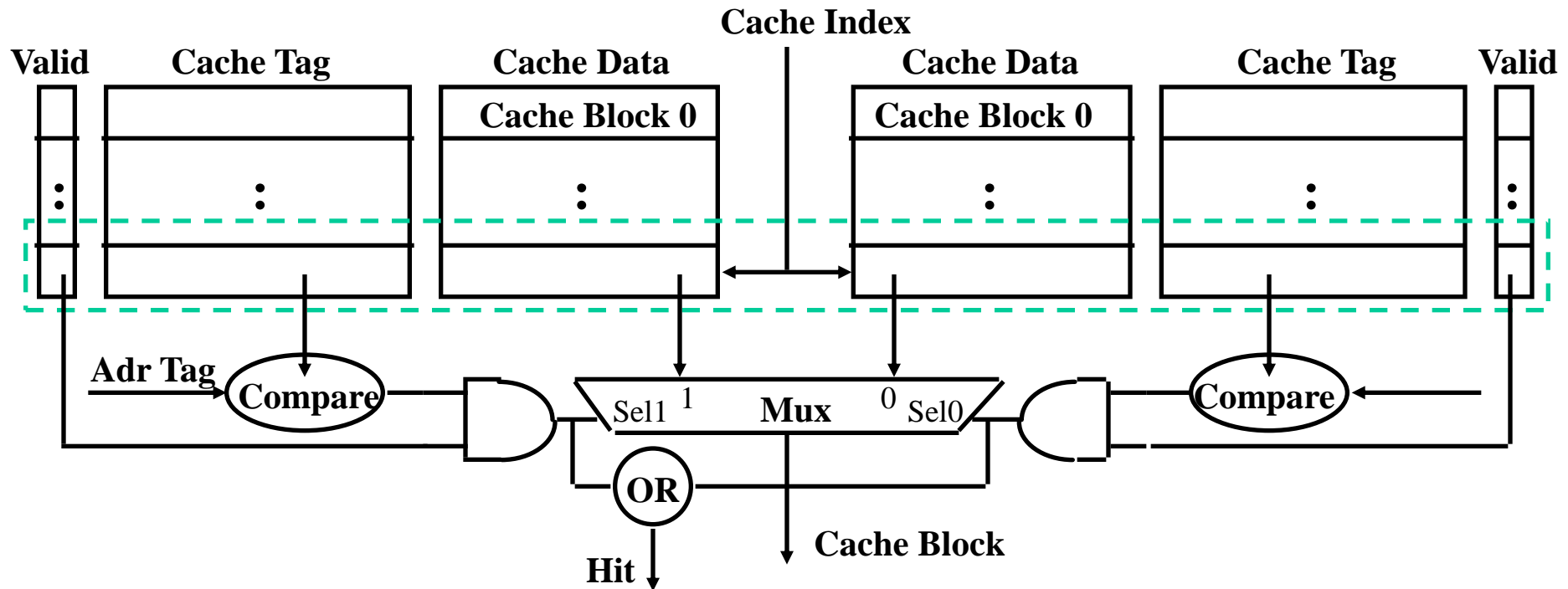
❑ Asociativa por conjuntos

- ❑ N-way asociativa : N entradas por conjunto
 - o N caches directas operando en paralelo (N típico 2 , 4)
- ❑ Ejemplo: 2-way cache asociativa
 - o Cache Index selecciona un conjunto de la cache
 - o Los dos tags en el conjunto son comparados en paralelo
 - o Se selecciona el dato en función de la comparación



❑ Comparación

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



- ❑ **Política de actualización :Write-Through vs Write-Back**
- ❑ **Write-through:** Todas las escrituras actualizan la cache y la memoria
 - o Se puede eliminar la copia de cache - Lo datos estan en la memoria
 - o Bit de control en la cache: Solo un bit de validez
- ❑ **Write-back:** Todas las escrituras actualizan solo la cache
 - o 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
- ❑ **Política de actualización :Asignación o no en fallo de escritura**

□ 4 aspectos fundamentales

- ✓ Ubicación de un bloque(línea)
 - ✓ Emplazamiento directo, asociativo, asociativo por conjuntos
- ✓ Acceso al bloque: uso del índice
 - ✓ Tag y bloque(línea)
- ✓ Reemplazamiento del bloque(línea)
 - ✓ Aleatorio, LRU
- ✓ Estrategia de escritura
 - ✓ Post-escritura(write back), escritura directa (write through)

□ Rendimiento de la cache

- ✓ Tiempo CPU = (Ciclos CPU + Ciclos de espera memoria)x Tc
- ✓ Ciclos de espera memoria= Accesos a memoria x Tasa de fallos x Tiempo de penalización de fallo

Multiplicando y dividiendo por N° de instr



- ✓ Tiempo CPU = N° de instr. x (CPI+accesos por inst x Tasa de fallos x penalización de fallo) x Tc

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \text{Miss rate} \times Miss\ penalty \right) \times Clock\ cycle\ time$$

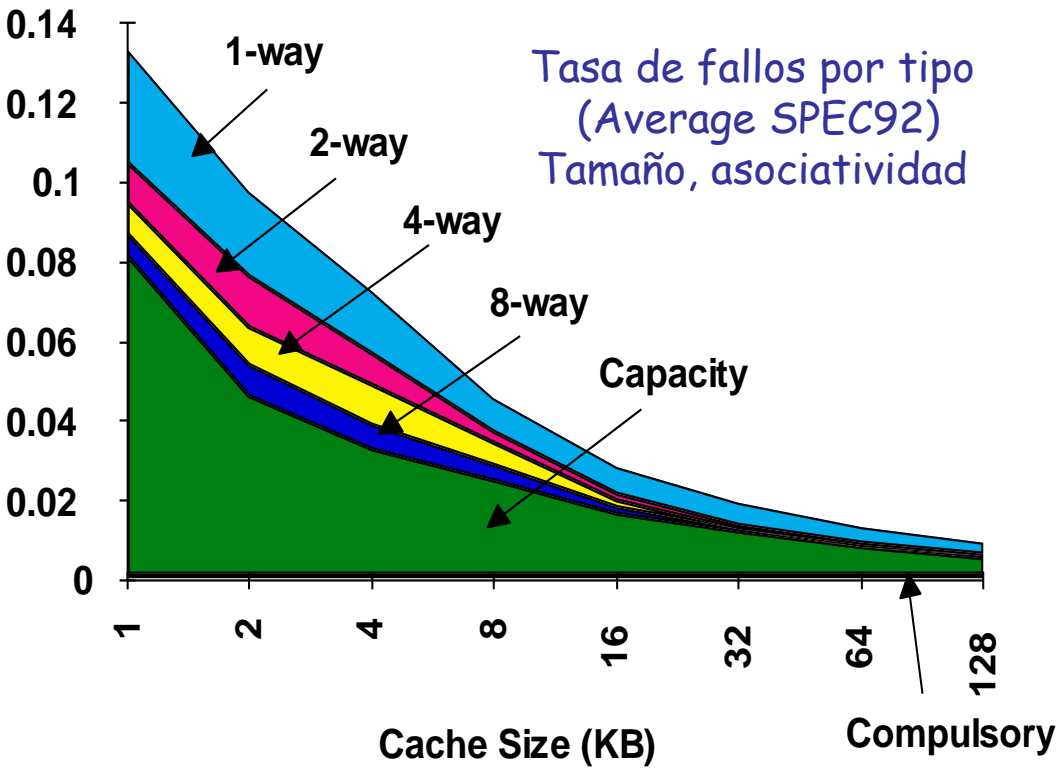
❑ ¿ Como mejorar el rendimiento de la cache

- Reducir la tasa de fallos
- Reducir la penalización del fallo
- Reducir el tiempo de acceso a la cache

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \text{Miss rate} \times Miss\ penalty \right) \times Clock\ cycle\ time$$

❑ Tipos de fallos (3 C's)

- Iniciales(Compulsory):
Fallos incluso en cache infinita
Localidad espacial
- Capacidad:
Tamaño de la cache
Localidad temporal
- Conflicto:
Política de emplazamiento

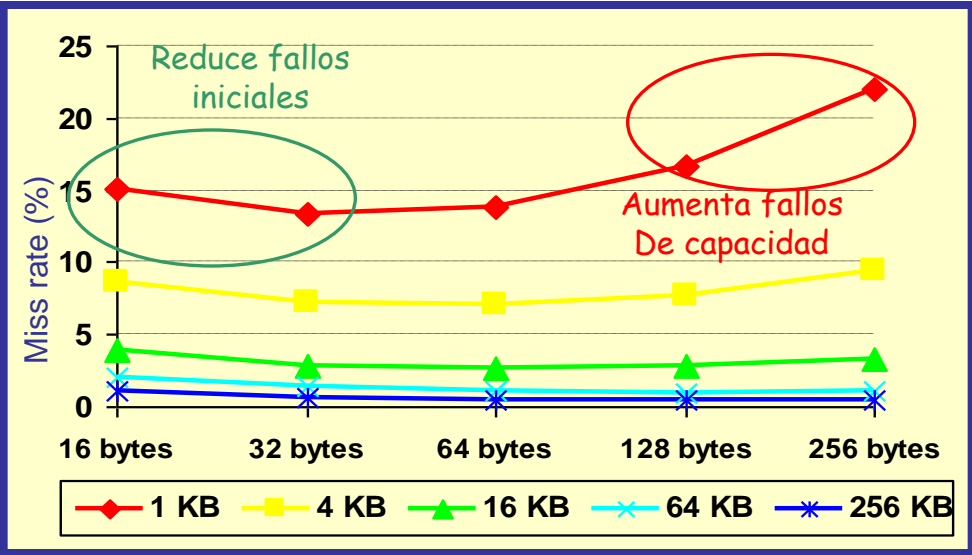


❑ Cambiando el tamaño del bloque

- Tasa de fallos



- Disminución de la **tasa de fallos de inicio** (captura mejor la localidad espacial)
- Aumento de la **tasa de fallos de capacidad** (menor *Nº Bloques* => captura peor localidad temporal)



- **Observaciones:**
 1. Valor óptimo mayor según aumenta Mc
 2. Caches integradas tamaño y bloque fijo
 3. Caches externas tamaño y bloque variable

Tamaño bloque

Tamaño cache

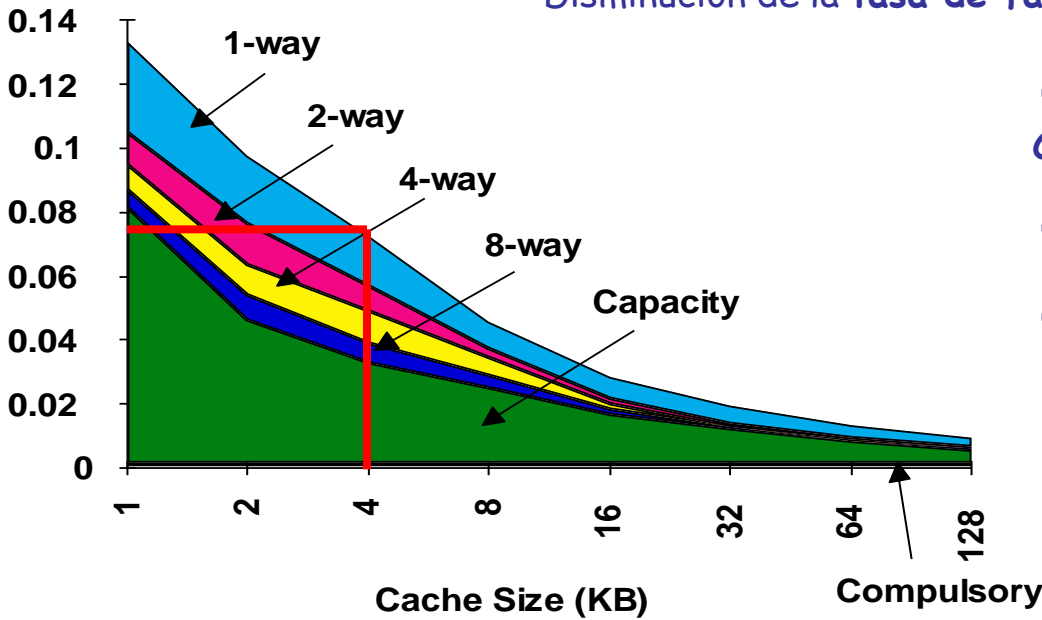
- Penalización



- Bloque del sig. nivel => Tecnología y organización del siguiente nivel

❑ Cambiando la asociatividad

- Tasa de fallos



- Disminución de la tasa de fallos de conflicto (más marcos posibles)
- E 1 Directo Asociativo por conjuntos M Asociativo

- ✓ Regla 2:1
Cache directa= 2way de mitad de tamaño
- ✓ 8 way es igual a totalmente asociativa
- ✓ Tiempo de acceso
- Observaciones:
 1. Mejora menor según aumenta el tamaño M_c
- Aumenta el número de conjuntos
 2. Mejora menor según aumenta asociatividad

- Tiempo de acceso
- Coste hardware

- E 1 Directo Asociativo por conjuntos M Asociativo
- Número de comparadores => tecnología

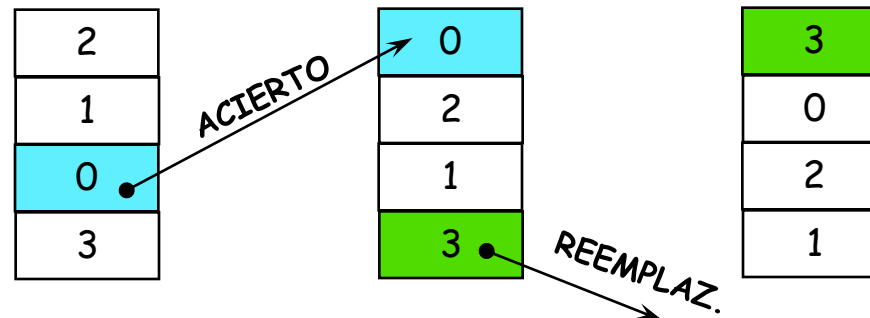
❑ Algoritmo de reemplazamiento

✓ Espacio de reemplazamiento

- **Directo:** Trivial
- **Asociativo:** Toda la cache
- **Asociativo por conjuntos:** Los marcos de un conjunto

✓ Algoritmos

- **Aleatorio:** El bloque reemplazado se escoge aleatoriamente
- **LRU (Least Recented Used):** Se reemplaza el bloque menos recientemente usado
 - **Gestión:** pila



- **Implementación:** registros de edad, implementación de la pila y matriz de referencias
- Para un grado de mayor que 4, muy costoso en tiempo y almacenamiento (actualiz. > t_{cache})
- **LRU aproximado:** Algoritmo LRU en grupos y dentro del grupo

❑ Cache de víctimas

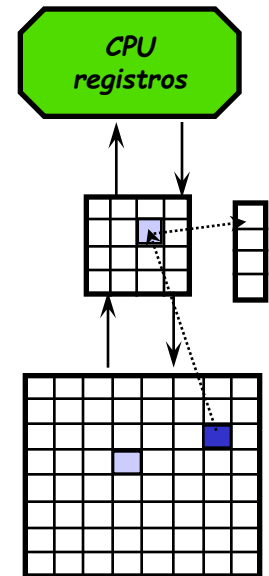
- Memoria cache más pequeña totalmente asociativa asociada a la memoria cache
 - => Contiene los bloques que han sido **sustituidos más recientemente**
 - => En un fallo primero **comprueba** si el bloque se encuentra en la cache víctima

😊 Baja la **tasa de fallos de conflicto** en caches con emplazamiento directo

- Cuanto menor es la memoria cache más efectiva es la cache víctima
- En una cache de 4KB con emplazamiento directo, una cache víctima de 4 bloques elimina del 20% al 95% de los fallos, según programa.

• Ejemplo:

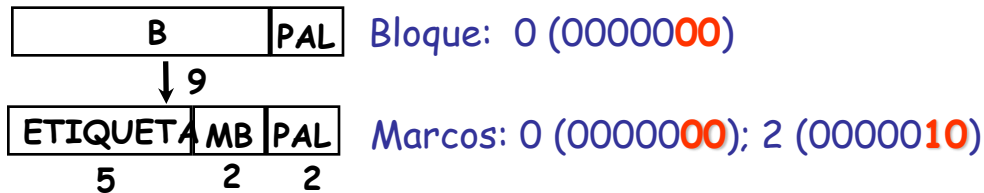
- HP 7200, 2 KB internos: 64 bloques de 32 bytes
- ALPHA,
- Power 4-5-6, AMD Quad, (**diferente uso**)



❑ Cache pseudo-asociativas

- Trata de combinar las ventajas de la cache directa y la asociativa
- Memoria cache de emplazamiento directo:

=> En un fallo busca el bloque en otro marco dado por la inversión del bit MS del índice de marco



Hit Time



Pseudo Hit Time



Miss Penalty



Time

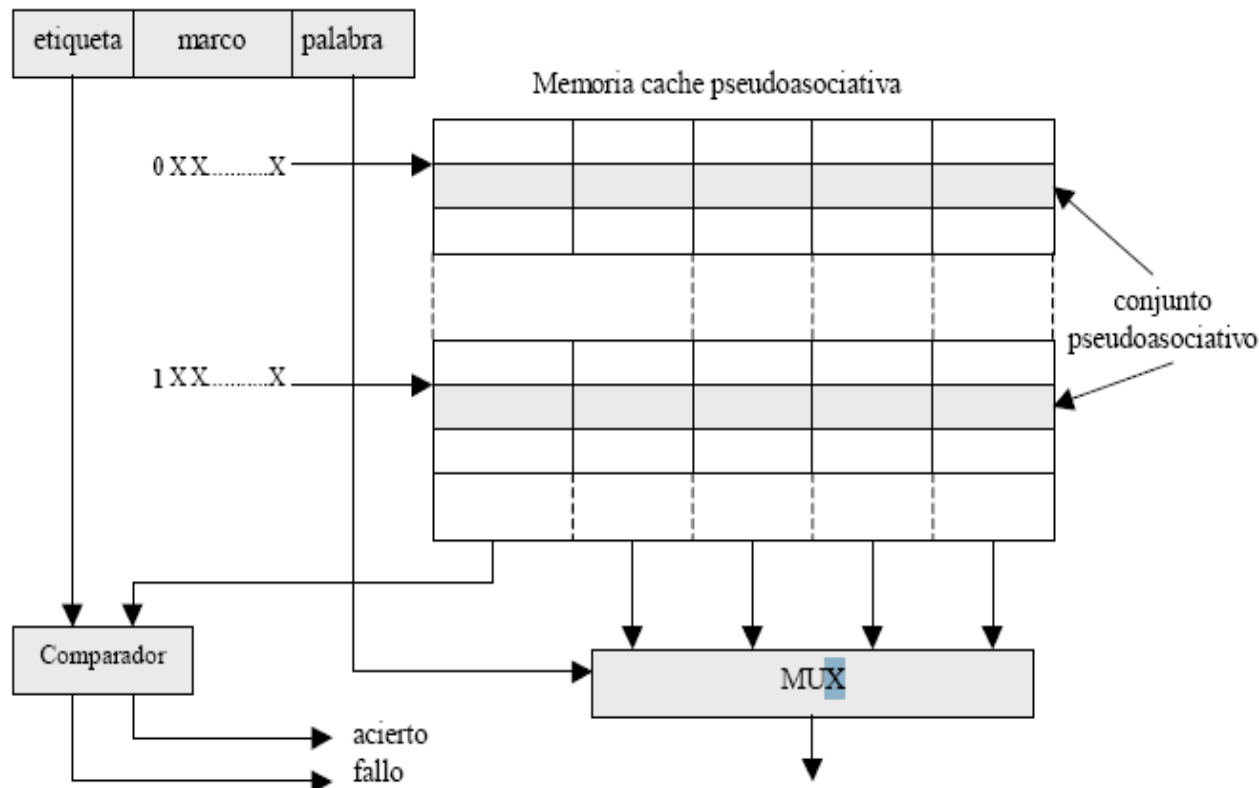
☹ Dos tiempos de acceso a cache => Gestión por parte del procesador

- ♦ Más útiles en cache "off chip" (L2)
- ♦ Ejemplos: R10000 L2, UltraSparc L2

- **Caches pseudo-asociativas (o columna asociativa):**
 - Consiste en utilizar toda la capacidad de la cache para reubicar algunos bloques extra en bloques que en principio no les pertenece
 - Implementación: Cuando en una cache de correspondencia directa se falla, antes de ir a buscar en la memoria principal puede intentarse en otro bloque (el correspondiente al índice pero con el bit más significativo invertido) del “pseudo conjunto”

❑ Cache pseudo-asociativas

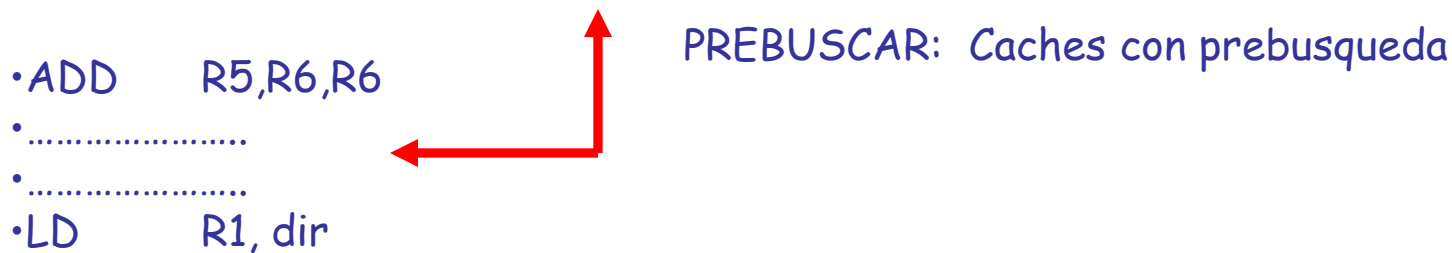
- Caché de correspondencia directa con una modificación para que se comporte como asociativas.
- Se permite que un bloque de Mpse pueda ubicar en dos (pseudoasociativade 2 vías) marcos de Mc:
 - el que le corresponde (por la correspondencia directa)
 - el que resulta de conmutar el bit más significativo de la dirección del bloque



❑ 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

- ✓ Solapa acceso a los datos con instrucciones anteriores a la que usa el dato
- ✓ No se relaciona con los registros. No genera riesgos
- ✓ Posibilidad de búsquedas innecesarias

➤ **Dos tipos**

- ✓ Prebusqueda SW
- ✓ Prebusqueda HW

Reducir la tasa de fallos

❑ Cache con prebusqueda

✓ Prebusqueda SW

Instrucciones especiales de prebusqueda introducidas por el compilador

La eficiencia depende del compilador y del tipo de programa

Prebusqueda con destino cache (MIPS IV, PowerPC, SPARC v. 9), o registro (HP-PA)
Instrucciones de prebusqueda no producen excepciones. Es una forma de especulación.

Funciona bien con bucles y pattern de acceso a array simples. Aplicaciones de cálculo

Funciona mal con aplicaciones enteras que presentan un amplio reuso de Cache

Overhead por las nuevas instrucciones. Más búsquedas. Más ocupación de memoria

❑ Cache con prebúsqueda (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).

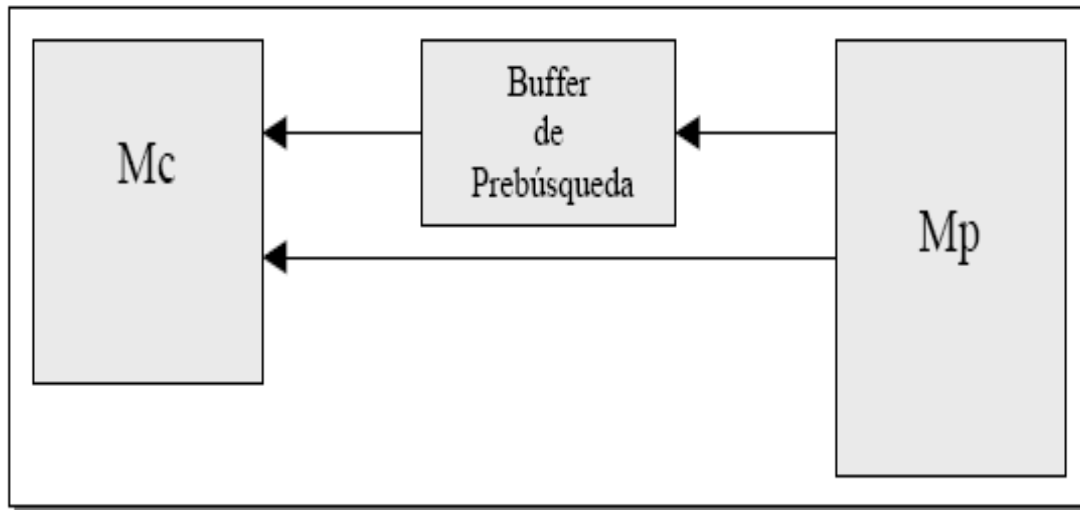
$$\text{Fallos "a"} = (3 \times 100) / 2 = 150$$

• 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

Prebúsqueda de instrucciones y datos

- La prebúsqueda de instrucciones y datos antes de ser demandados disminuye la tasa de fallos.
- Las instrucciones o datos prebuscados son llevados directamente a la caché o a un buffer externo
- El Alpha AXP 21064 prebusca bloques cuando ocurre un fallo, el que contiene la palabra causante del fallo y el siguiente. El primero lo coloca en MCy el segundo en el *buffer de prebúsqueda*
- *Experimentalmente se ha comprobado que un buffer de prebúsqueda simple elimina el 25 % de los fallos de una caché de datos con correspondencia directa de 4 KB*



❑ Cache con prebúsqueda (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 = $251 - 19 = 232$

❑ Cache con prebusqueda

✓ Prebusqueda HW

Prebusqueda secuencial de un bloque adicional

➤Tipos

Prebusqueda sobre fallo

Prebusca el bloque $b+1$ si el acceso b produce un fallo

Prebusqueda marcada

Se incluye un tag en cada bloque. Si un bloque marcado es buscado o prebuscado el siguiente se prebusca. Sigue bien la localidad espacial

Prebusqueda adaptativa

El grado de prebusqueda (numero de bloques prebuscados) se ajusta en función del comportamiento

Prebusqueda con "stride" arbitrario

Reducir la tasa de fallos

❑ Cache con prebúsqueda

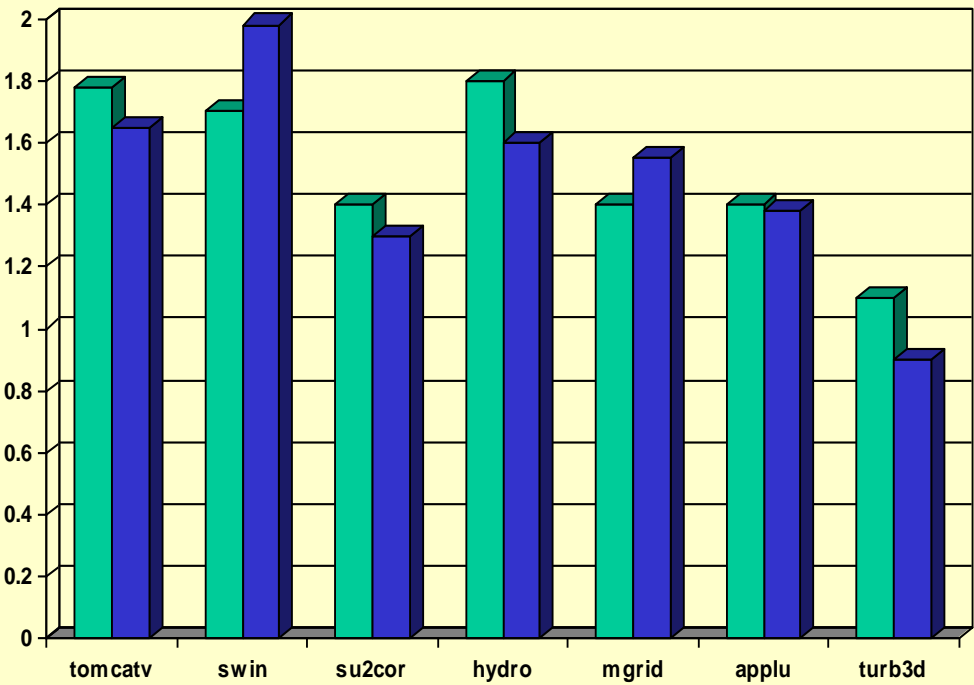
✓ Comparación

HP 7200 Prebúsqueda HW - HP8000 Prebúsqueda SW

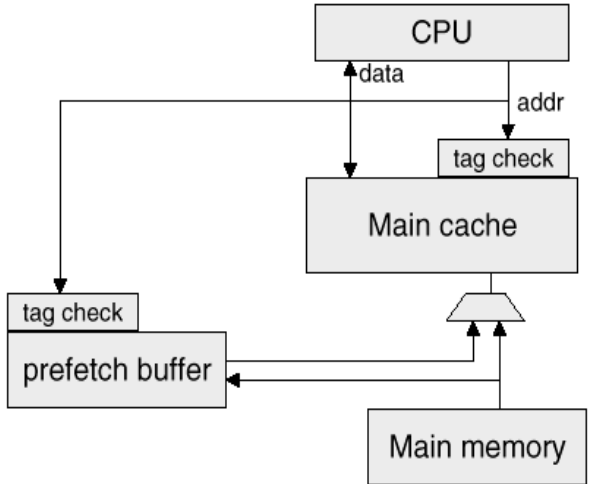
✓ Implementacion

Rendimiento
Relativo

■ HP7200 ■ HP8000



Specfp95



➤ Estado de los bloques prebuscados (stream buffer)
Bloque prebuscado pasa a buffer
Al ser referenciado pasa a Cache
Alpha busca dos bloques en fallo

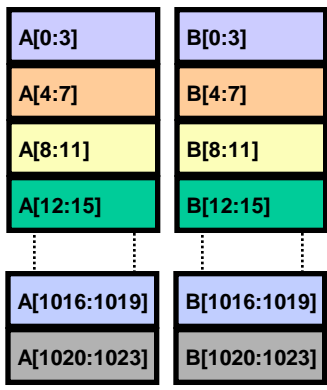
❑ Compilador: Optimización de código

- ✓ **Ejemplo:** DEC Alpha 21064: Mc de 8 KB, E = 1, M = 256(numero de bloques), Bloque = 4 palabras de 64 bits (32 bytes)
 - Cada 1024 palabras se repite la asignación de marcos de bloques.

- 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];

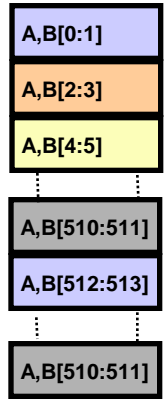
for (i = 0; i < 1024; i = i + 1)
    C = C + (A[i] + B[i]);
```



2x1024 fallos
2x256 de inicio
1536 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

Reducir la tasa de fallos

❑ Compilador: Optimización de código

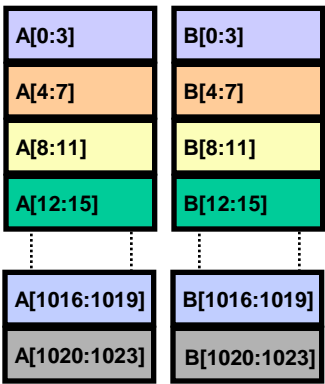
2) Alargamiento de arrays: Mejora la localidad espacial para disminuir los fallos de conflicto

- 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
1536 de conflicto

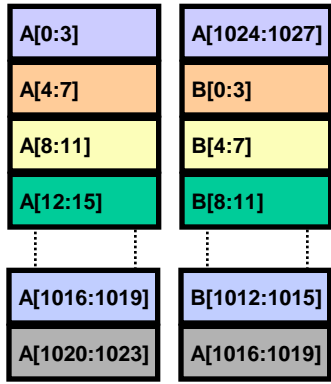


```
double A[1028];
double B[1024];

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

1024/2 fallos
512 de inicio

Ganancia: 4



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];
```

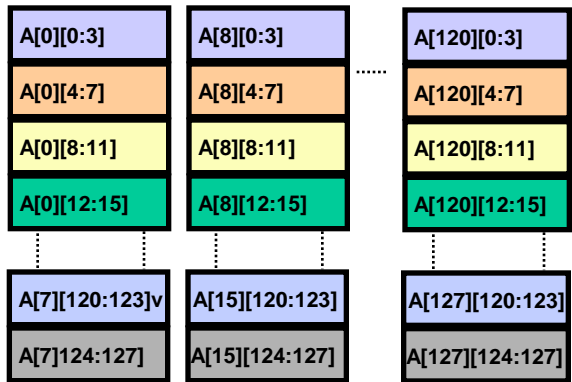
128x128 fallos
16x256 de inicio
12288 de conflicto

```
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



❑ Compilador: Optimización de código

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

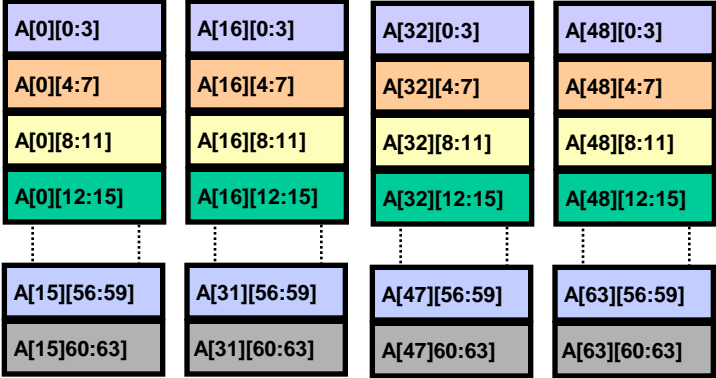
- Fusionar los bucles que usen los mismos arrays para usar los datos que se encuentran en cache antes de desecharlos

```
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];
```

(64x64/4)x2 fallos
4x256 de inicio
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];  
  }
```

64x64/4 fallos
4x256 de inicio



Ganancia: 2

❑ Compilador: Optimización de código

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

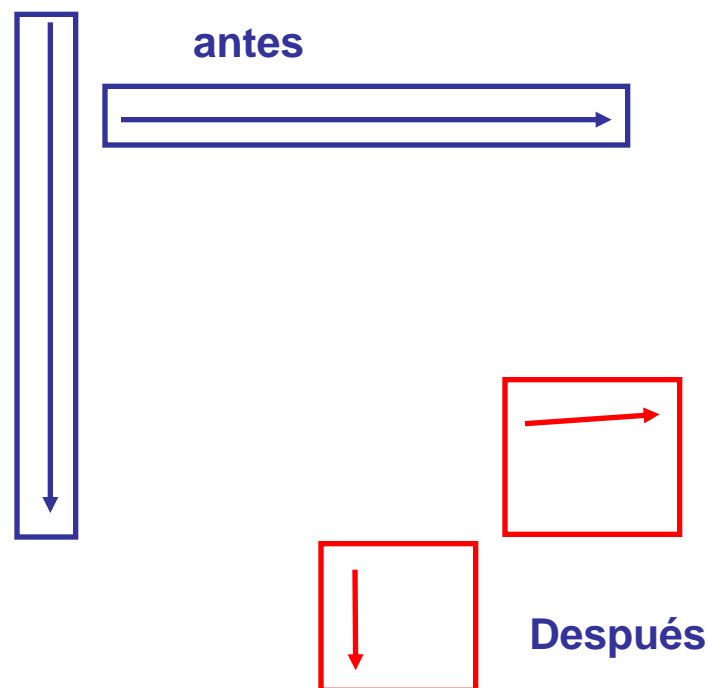
```
/* Antes */  
for (i=0; i < N; i=i+1)  
  for (j=0; j < N; j=j+1)  
    {r = 0;  
      for (k=0; k < N; k=k+1){  
        r = r + y[i][k]*z[k][j];};  
      x[i][j] = r;  
    };
```

✓ Dos bucles internos:

- Lee todos los NxN elementos de **z**
- Lee N elementos de 1 fila **y** en cada iteración
- Escribe N elementos de 1 fila de **x**

✓ Fallos de capacidad dependen de N y Tamaño de la cache:

✓ Idea: calcular por submatrices BxB que permita la cache



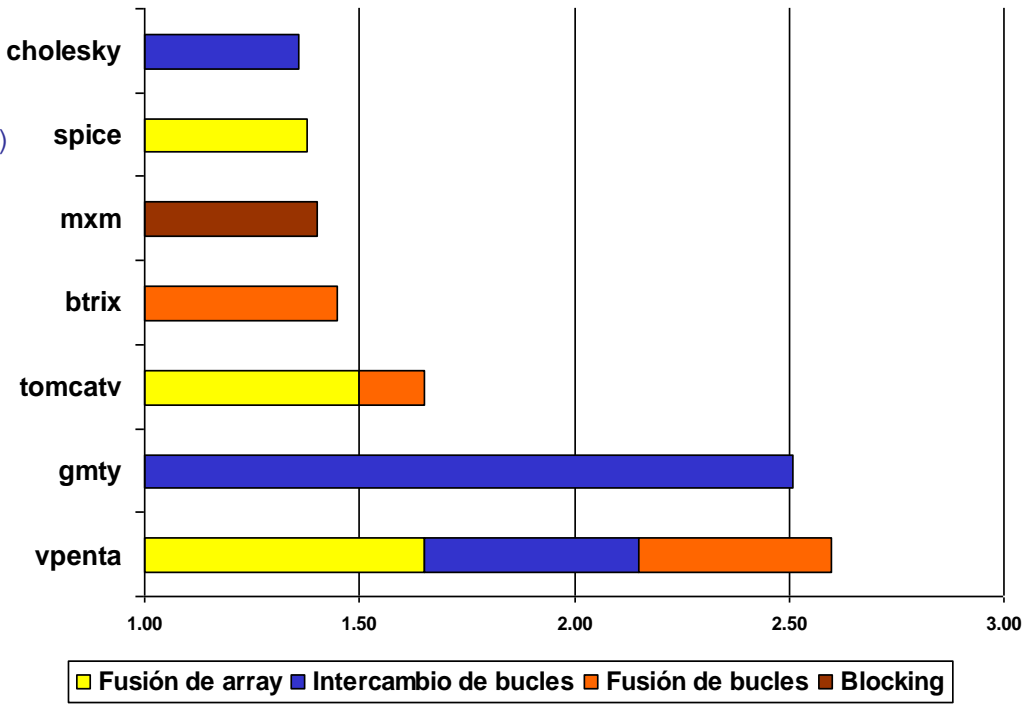
❑ Compilador: Optimización de código

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

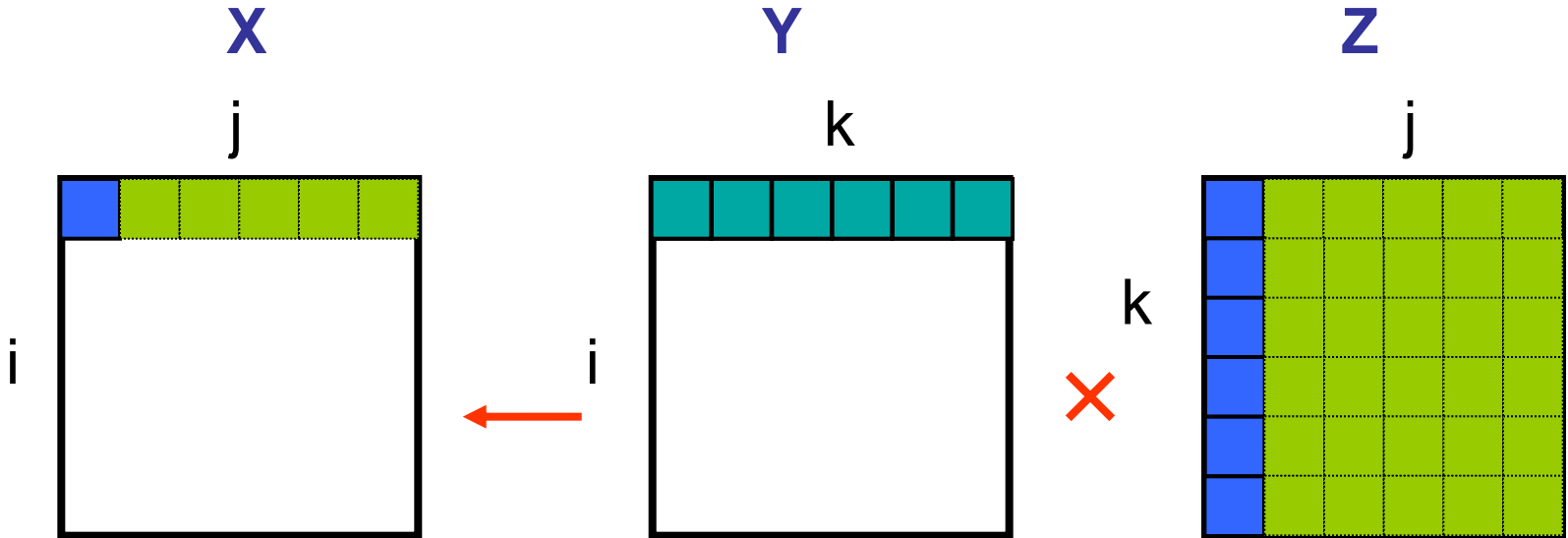
```
/* Despues */
for (jj=0; jj < N; jj=jj+B)
for (kk=0; kk < N; kk=kk+B)
for (i=0; i < N; i=i+1)
  for (j=jj; j < min(jj+B-1,N); j=j+1)
    {r = 0;
     for (k=kk; k < min(kk+B-1,N); k=k+1) {
       r = r + y[i][k]*z[k][j];};
     x[i][j] = x[i][j]+r;
    };
```

✓ B Factor de bloque (Blocking Factor)

Mejora de rendimiento



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



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

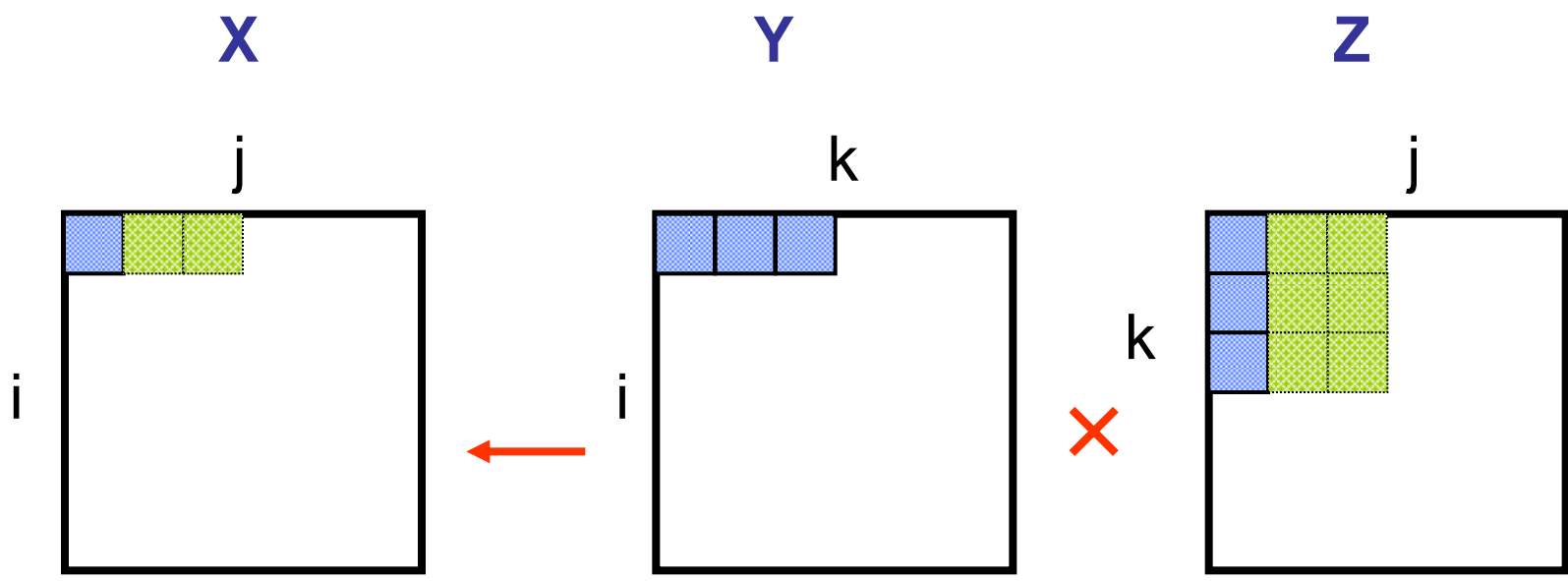




$i = 0, j = 1..5, k = 0..5$

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

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

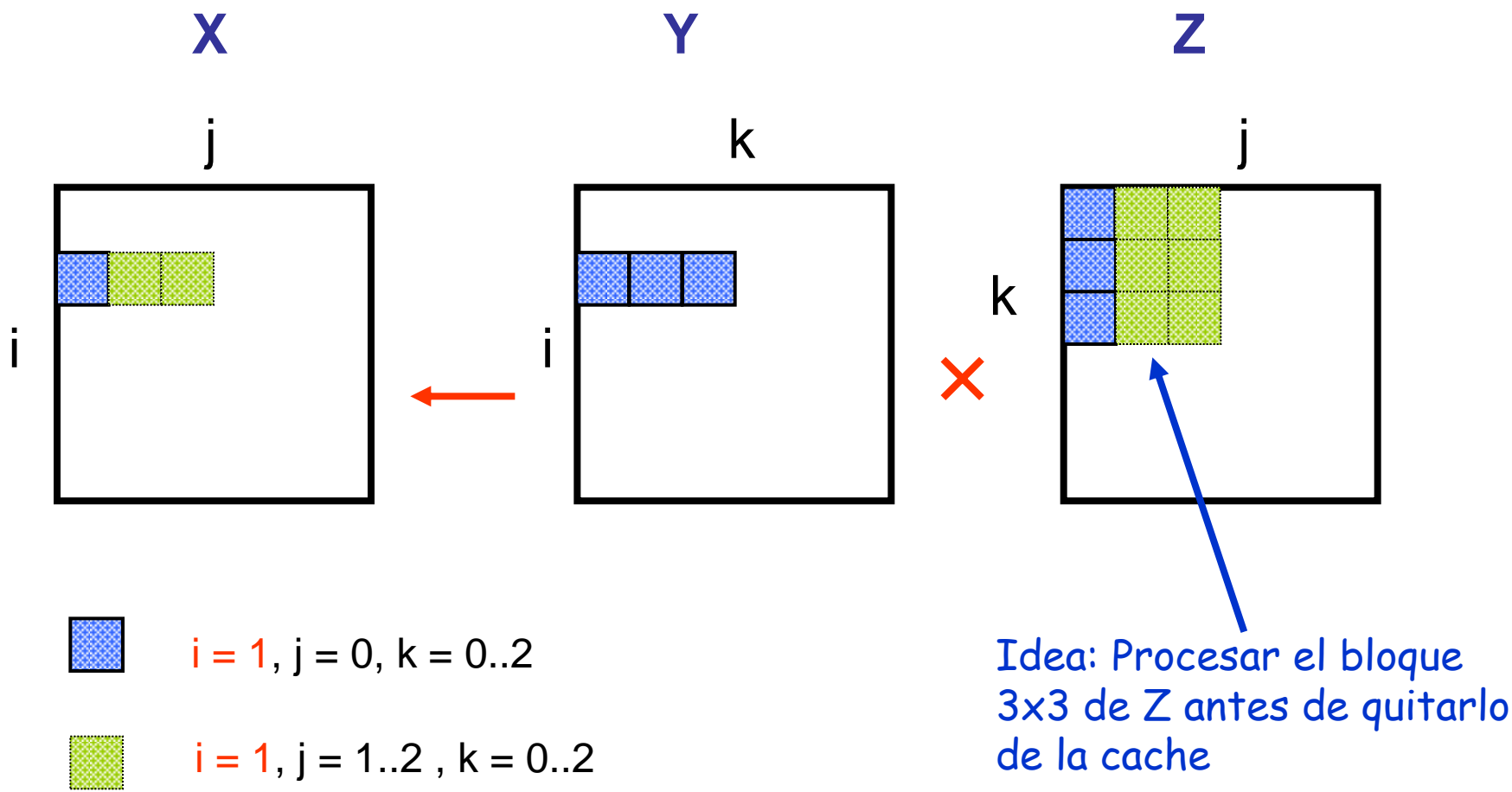
❑ Ejemplo "blocking": Con Blocking (B=3)



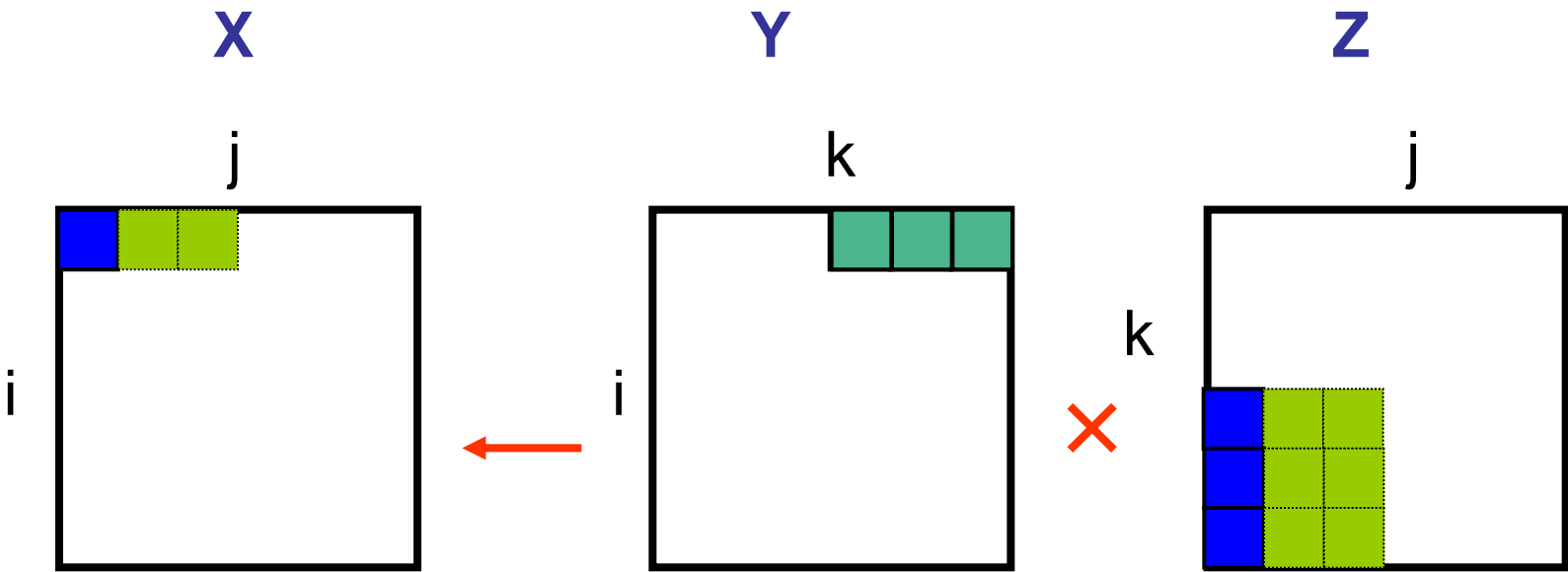
-  $i = 0, j = 0, k = 0..2$
-  $i = 0, j = 1..2, k = 0..2$

Evidentemente, los elementos de X no están completamente calculados

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



❑ 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!

□ Resumen

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \text{Miss rate} \times Miss\ penalty \right) \times Clock\ cycle\ time$$

- 1) Tamaño del bloque
- 2) Asociatividad
- 3) Algoritmo de reemplazamiento
- 4) Cache de víctimas
- 5) Cache pseudo-asociativas
- 6) Cache con prebúsqueda
- 7) Compilador: Optimización de código

□ ¿ Como mejorar el rendimiento de la cache

- Reducir la tasa de fallos
- Reducir la penalización del fallo
- Reducir el tiempo de acceso a la cache

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \text{Miss rate} \times \text{Miss penalty} \right) \times Clock\ cycle\ time$$

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

- ❑ Write-through: Todas las escrituras actualizan la cache y la memoria
 - o Se puede eliminar la copia de cache - Los datos están en la memoria
 - o Bit de control en la cache: Solo un bit de validez
- ❑ Write-back: Todas las escrituras actualizan solo la cache
 - o 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

❑ Política de actualización :Asignación o no en fallo de escritura

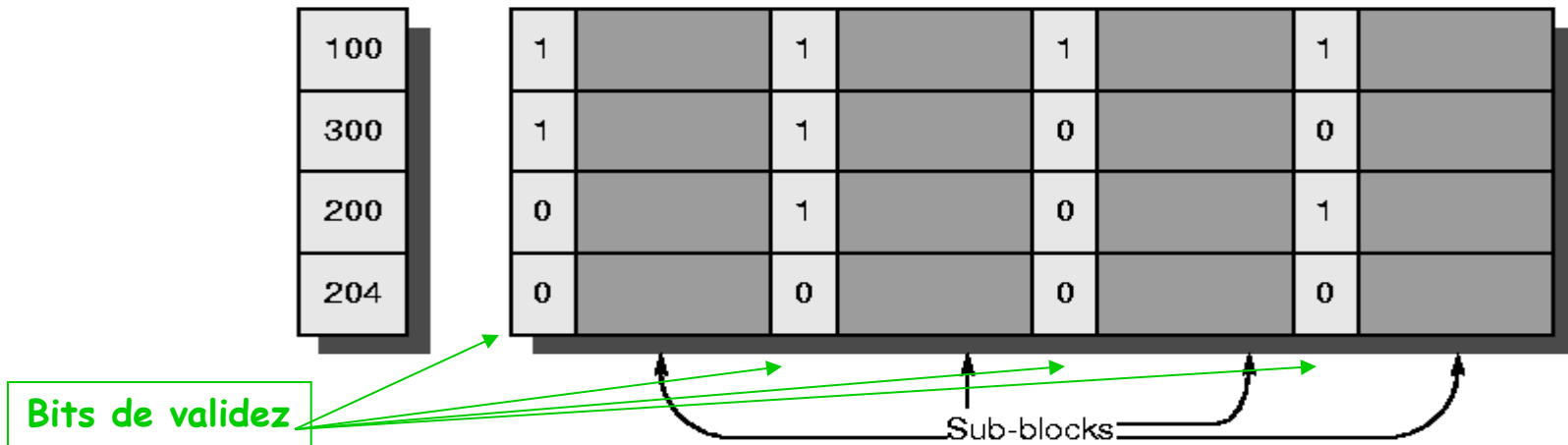
Reducir la penalización del fallo

❑ Dar prioridad a la lecturas sobre las escrituras

- ✓ Con escritura directa (write through). Buffer de escrituras. Check buffer si no hay conflicto prioridad lecturas
- ✓ Con post-escritura. Buffer para bloque sucio y leer primero bloque en fallo

❑ Reemplazamiento de subbloques

- ✓ No reemplazar el bloque completo sobre un fallo
- ✓ Localidad y tamaño (Usado también para reducir almacenamiento de "tags")



❑ Envío directo de la palabra solicitada al procesador

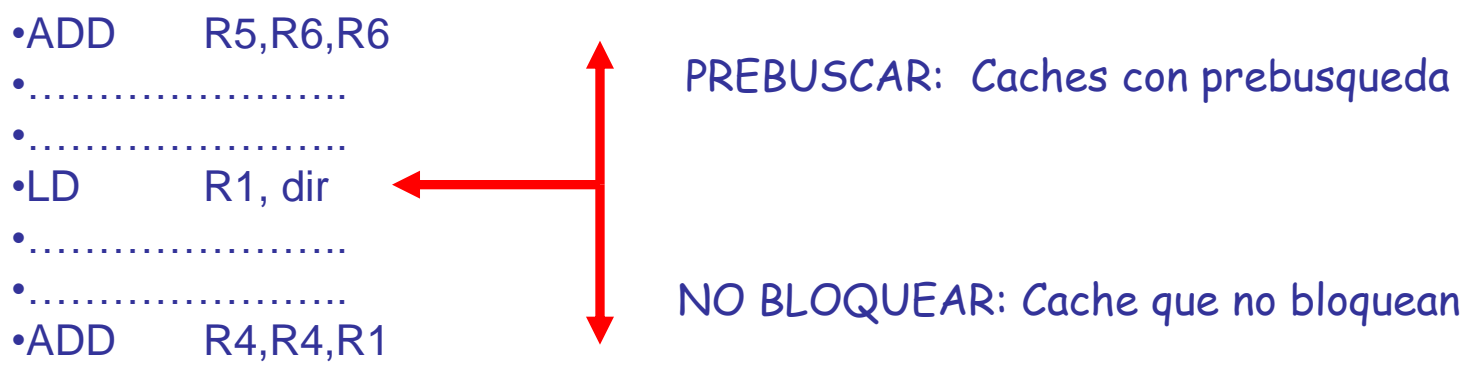
- ✓ **Carga anticipada (early restart)**: Cuando la palabra solicitada se carga en memoria cache se envía al procesador
- ✓ **Primero la palabra solicitada (critical word first)**: Primero se lleva al procesador y a memoria cache la palabra solicitada
- ✓ El resto del bloque se carga en memoria cache en los siguientes ciclos
- ✓ Su eficiencia depende del **tamaño del bloque** . Util con bloques grandes
 - => para bloques pequeños la ganancia es muy pequeña
- ✓ Problema: localidad espacial, después la siguiente palabra

Reducir la penalización del fallo

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

✓ Idea

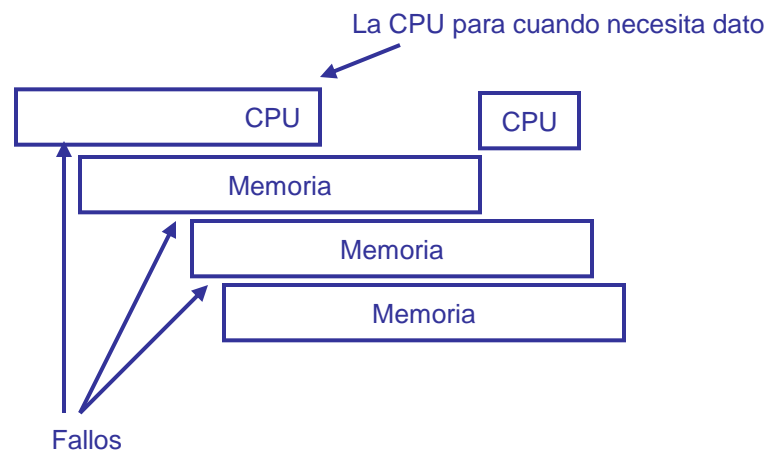
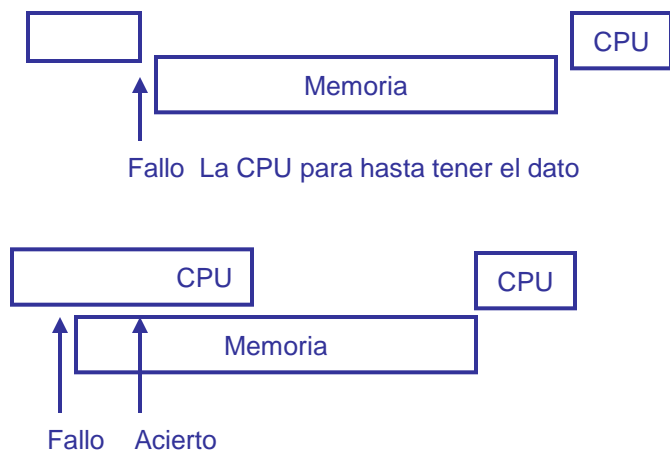
Ocultar la latencia de un fallo de cache solapandolo con otras instrucciones independientes



Reducir la penalización del fallo

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

- Permite que la ejecución siga aunque se produzca un fallo mientras ni se necesita el dato
- Un fallo sin servir. Sigue ejecutando y proporcionando datos que están en cache
 - HP7100, Alpha 21064
- Múltiples fallos sin servir. R12000 (4) , Alpha21264 (8), HP8500 (10), PentiumIII y 4 (4)
- Los beneficios dependen de la planificación de instrucciones
- Requiere interfase de memoria más complejo (múltiples bancos)



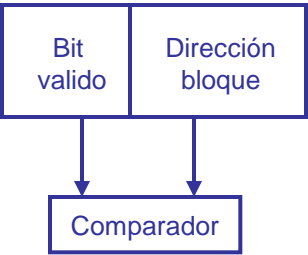
Reducir la penalización del fallo

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

Hay que asociar un registro a la petición cuando se inicia un load sin bloqueo
LD R1,dir

- Información necesaria en el control de la función
- Dirección del bloque que produce el fallo
- Entrada de la Cache para el bloque
- Ítem en el bloque que produce el fallo
- Registro donde se almacena el dato

Implementación (MSHR Miss Status Holding Register):



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

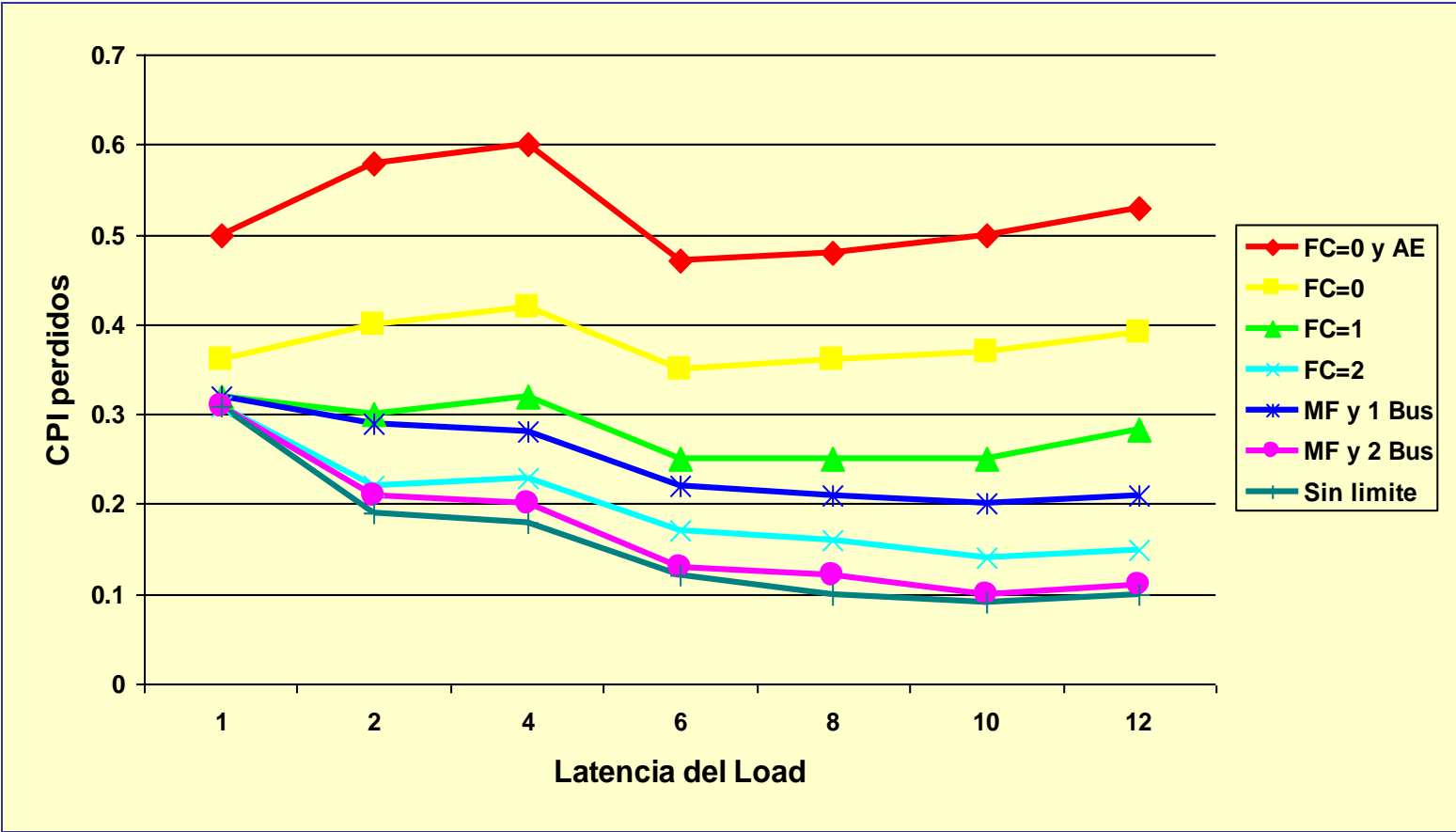
Palabra 0
Palabra 1
Palabra 2
Palabra 3

Fallo primario
Fallo secundario

→ Estructura del MSHR para
bloque de 32 byte, palabra 8
bytes
(Solo un fallo por palabra)

Reducir la penalización del fallo

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



FC nº de fallos
AE actualización en escritura

MF múltiples fallos y nº de búsquedas

Reducir la penalización del fallo

❑ Cache multinivel (L2,L3,...)

o Más grande, más rápida \Rightarrow Dos niveles de cache

o L2 Tiempo de acceso medio a memoria (TAMA)

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

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

$$\text{TAMA} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

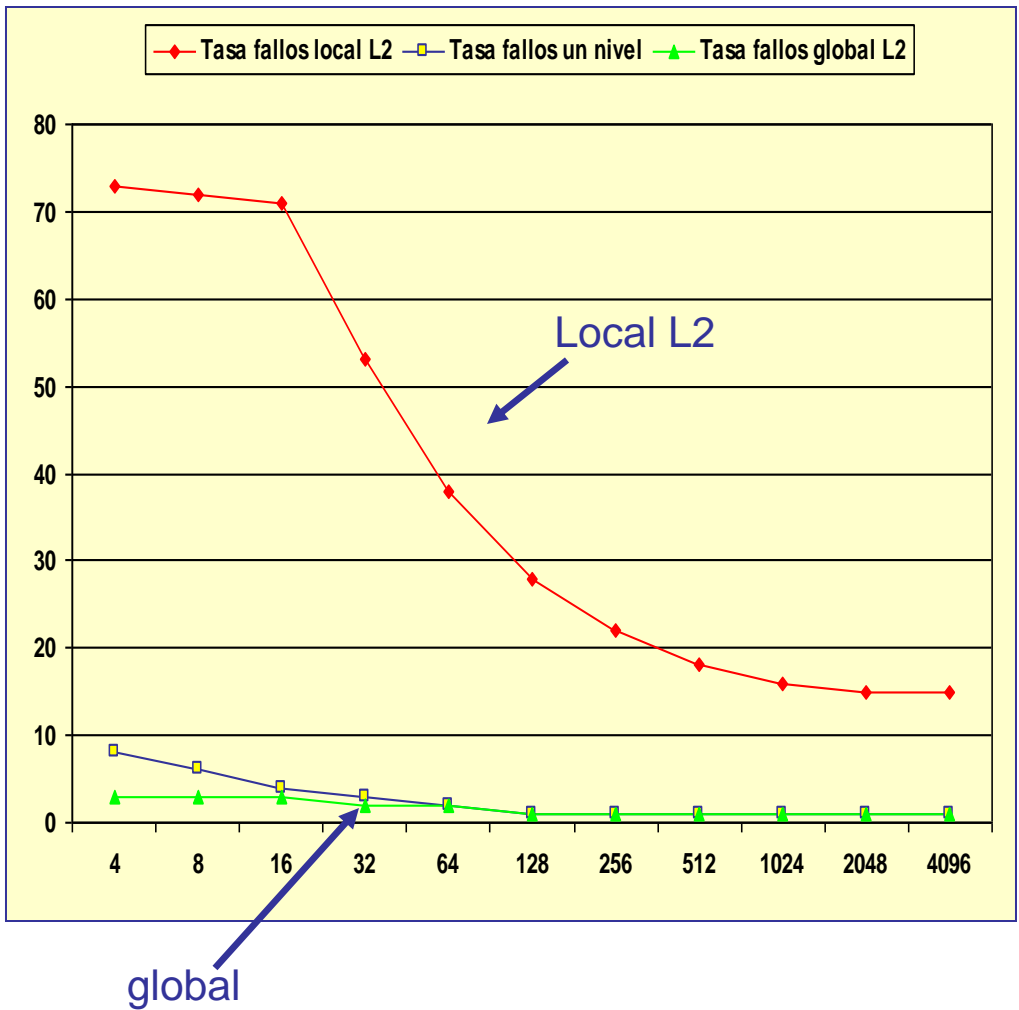
o Definiciones:

- ✓ **Tasa de fallos local**— fallos en esta cache dividido por el numero total de accesos a esta cache (Miss rate_{L2})
- ✓ **Tasa de fallos global**—fallos en esta cache dividido por el numero total de accesos a memoria generados por el procesador
- ✓ La tasa de fallos global es lo importante
- ✓ - **L1**: Afecta directamente al procesador \Rightarrow Acceder a un dato en el ciclo del procesador
- ✓ - **L2**: Afecta a la penalización de L1 \Rightarrow **Reducción del tiempo medio de acceso**

Reducir la penalización del fallo

❑ Cache multinivel (L2,L3,...)

- ✓ Cache L1 32Kbytes, cache L2 diferentes tamaños
- ✓ Tasa de fallos local no es una medida
- ✓ El tiempo de acceso de L2 solo afecta al tiempo de penalización
- ✓ Tamaño de L2>>L1
- ✓ Reducción de fallos en L2 igual que L1 asociatividad, tamaño de bloque,...
- ✓ Costo



Reducir la penalización del fallo

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty \right) \times Clock\ cycle\ time$$

- 1) Dar prioridad a la lecturas sobre las escrituras
- 2) Reemplazamiento de subbloques
- 3) Envío directo de la palabra solicitada al procesador
- 4) Cache sin bloqueo (non-blocking, lockup-free)
- 5) Cache multinivel (L2,L3,...)

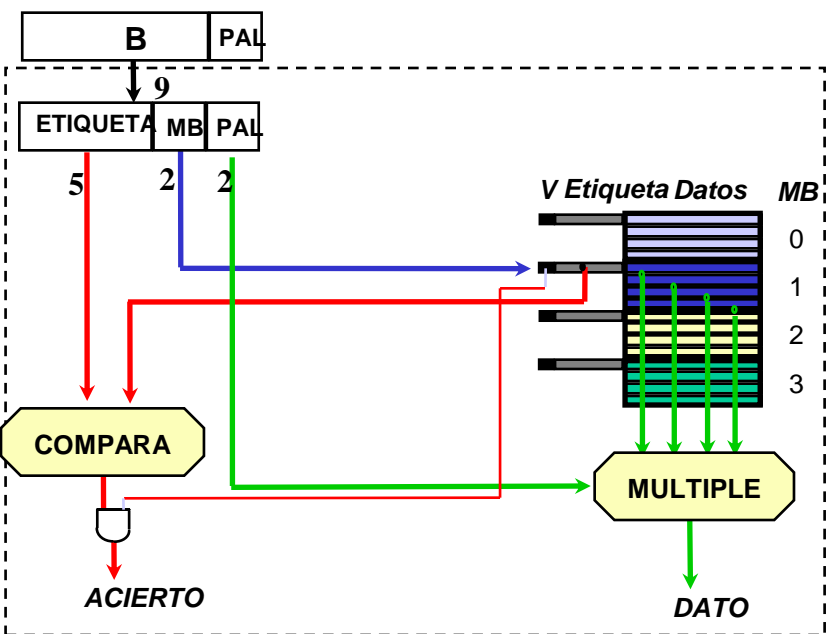
□ ¿ Como mejorar el rendimiento de la cache

- Reducir la tasa de fallos
- Reducir la penalización del fallo
- Reducir el tiempo de acceso a la cache

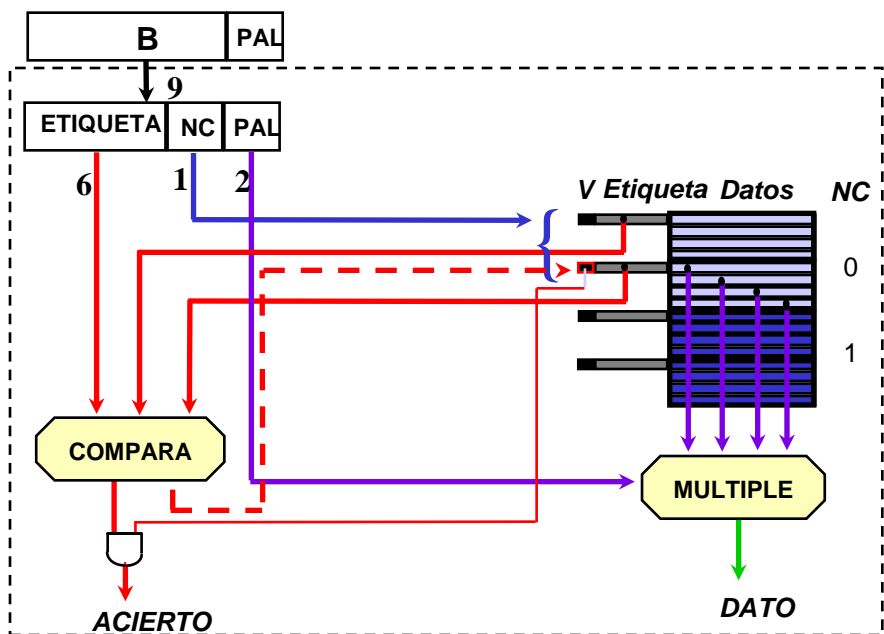
❑ Caches simples y pequeñas

- ✓ **Pequeña** para que: Se pueda **integrar** junto al procesador
 - ⇒ evitando la penalización en tiempo del acceso al exterior
 - ⇒ Tiempo de propagación versus tiempo de ciclo del procesador
- ✓ **Simple** (grado de asociatividad pequeño)
 - ⇒ **solapamiento** y **tiempos de acceso** menores

• Identificación+Comparación+Lectura



Directo

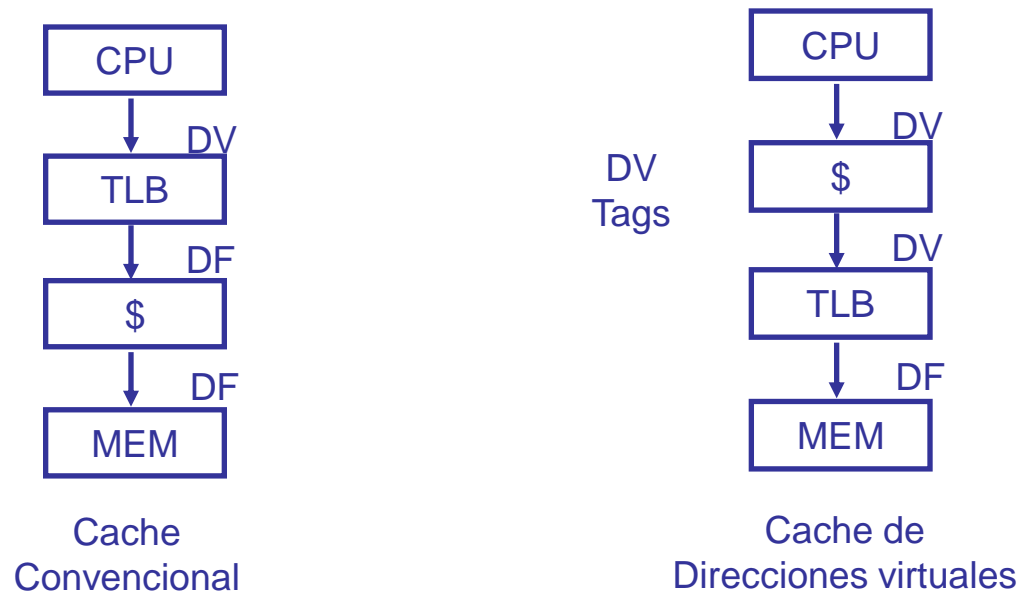


Asociativo por conjuntos

Reducir el tiempo de acceso

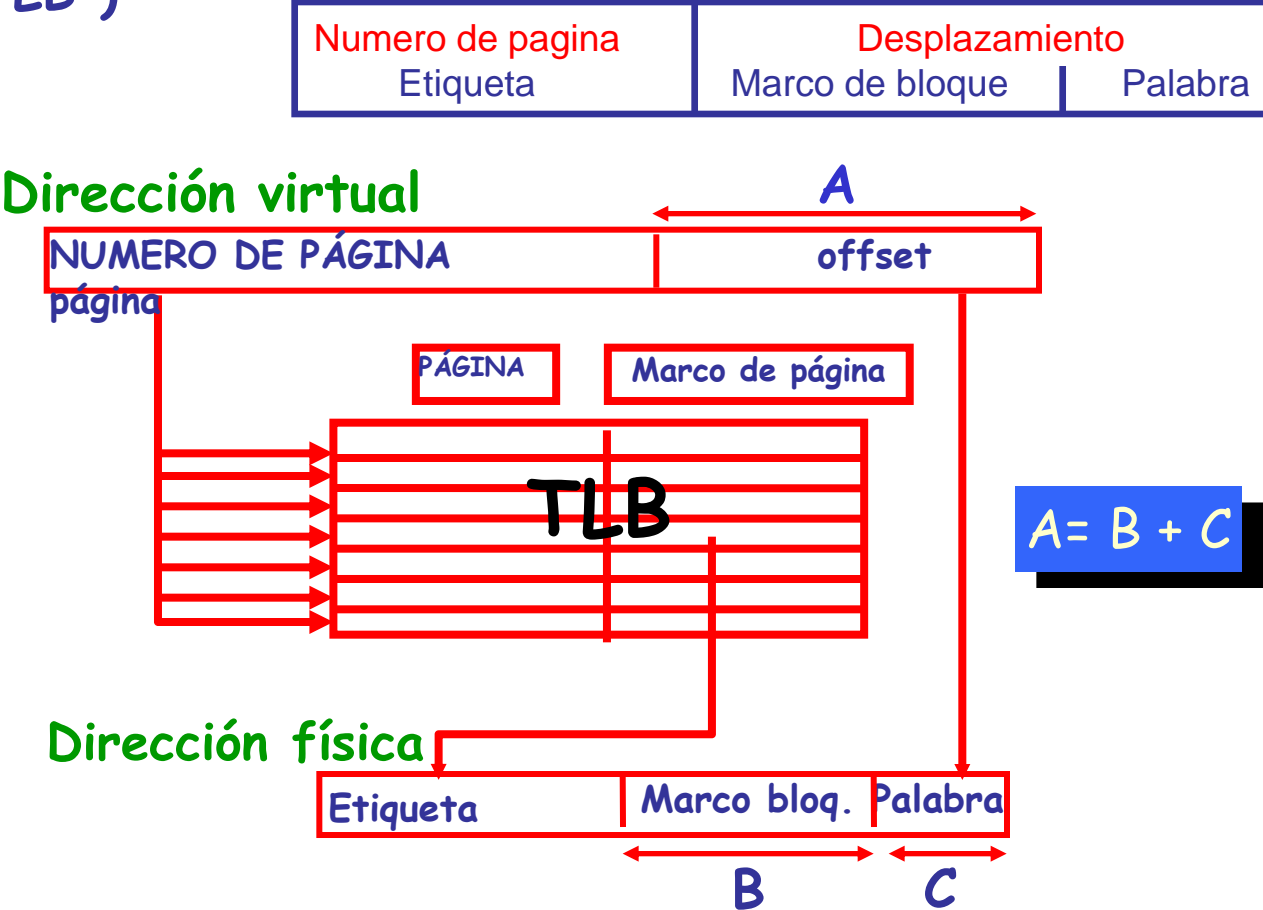
❑ Cache de direcciones virtuales (evita espera TLB)

- Acceder a la cache con la dirección virtual
 - ✓ Al cambio de contexto borrar la cache. Falsos aciertos
 - Costo = tiempo de borrado (flush) + fallos iniciales
 - ✓ No borrado, problema de sinónimos
 - Dos direcciones virtuales apuntan la misma dirección física
- Solución a los sinónimos y borrado
 - ✓ Añadir un identificador de proceso a cada bloque de cache.



Reducir el tiempo de acceso

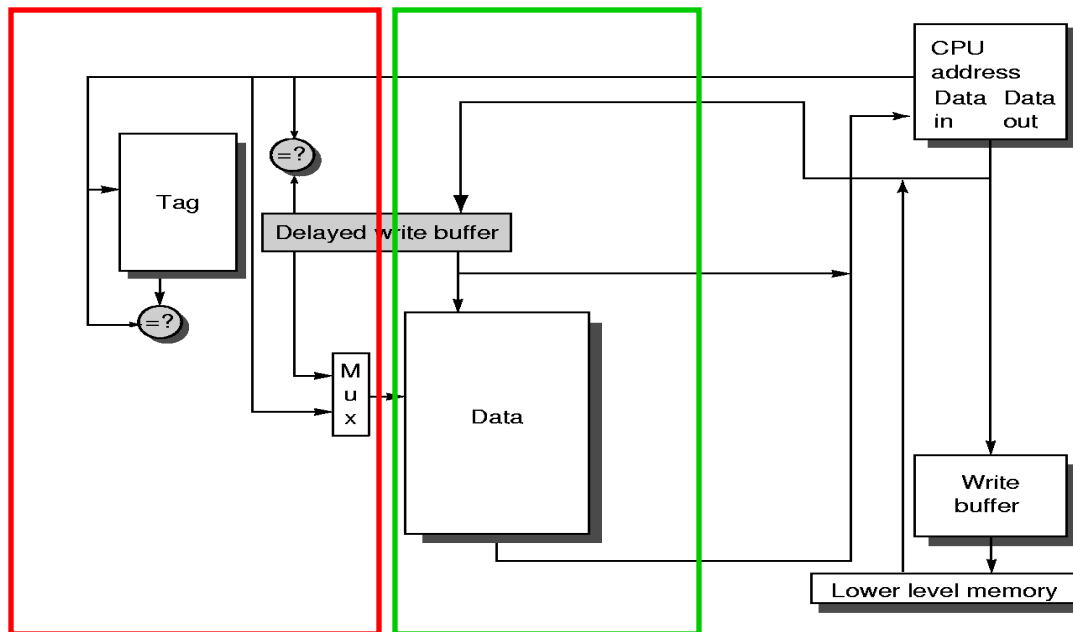
❑ Caches virtualmente accedidas físicamente marcadas (evita espera TLB)



- o Se solapa la traducción del numero de pagina con el acceso a los tag del marco de bloque
- o 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

❑ Segmentar escrituras

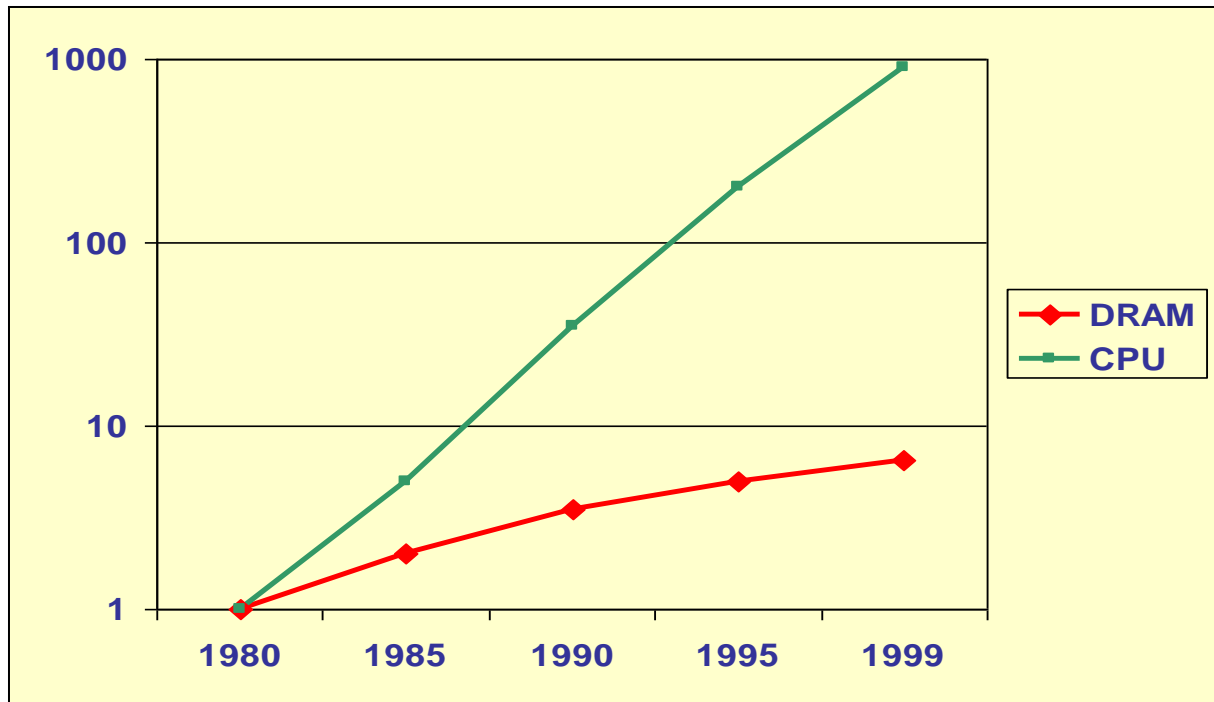
- ✓ Las operaciones de escrituras más lentas (se escribe después de chequear el tag)
- ✓ Se segmenta el acceso: 1º se chequea el tag, 2º se actualiza el dato
- ✓ Un write chequea el tag, el anterior esta escribiendo el dato en la cache (Latencia de escrituras)
- ✓ Los accesos de escritura deben usar el Delayed Write Buffer



Caches Resumen

	Técnica	TF	PF	TA	Complejidad
Tasa de fallo	Tamaño grande del bloque	+	-		0
	Alta asociatividad	+		-	1
	Cache de victimas	+			2
	Caches pseudo-Asociativas	+			2
	Prebusqueda Hw	+			2
	Prebusqueda Sw compilador	+			3
	Optimizaciones del compilador	+			0
Penalización de fallo	Prioridad de lecturas		+		1
	Reemplazamiento de subbloques		+	+	1
	Early Restart & Critical Word 1st		+		2
	Caches Non-Blocking		+		3
	Caches L2		+		2
Tiempo acceso	Caches pequeñas y sencillas	-		+	0
	Evitar/solapar traducción de Dir.			+	2
	Segmentar escrituras			+	1

□ El problema



1960-1985 rendimiento función de nº de operaciones

1990's Ejecución segmentada, superescalar, fuera de orden, especulación,...

1999 rendimiento función de accesos fuera de la cache

¿ Que Implica?

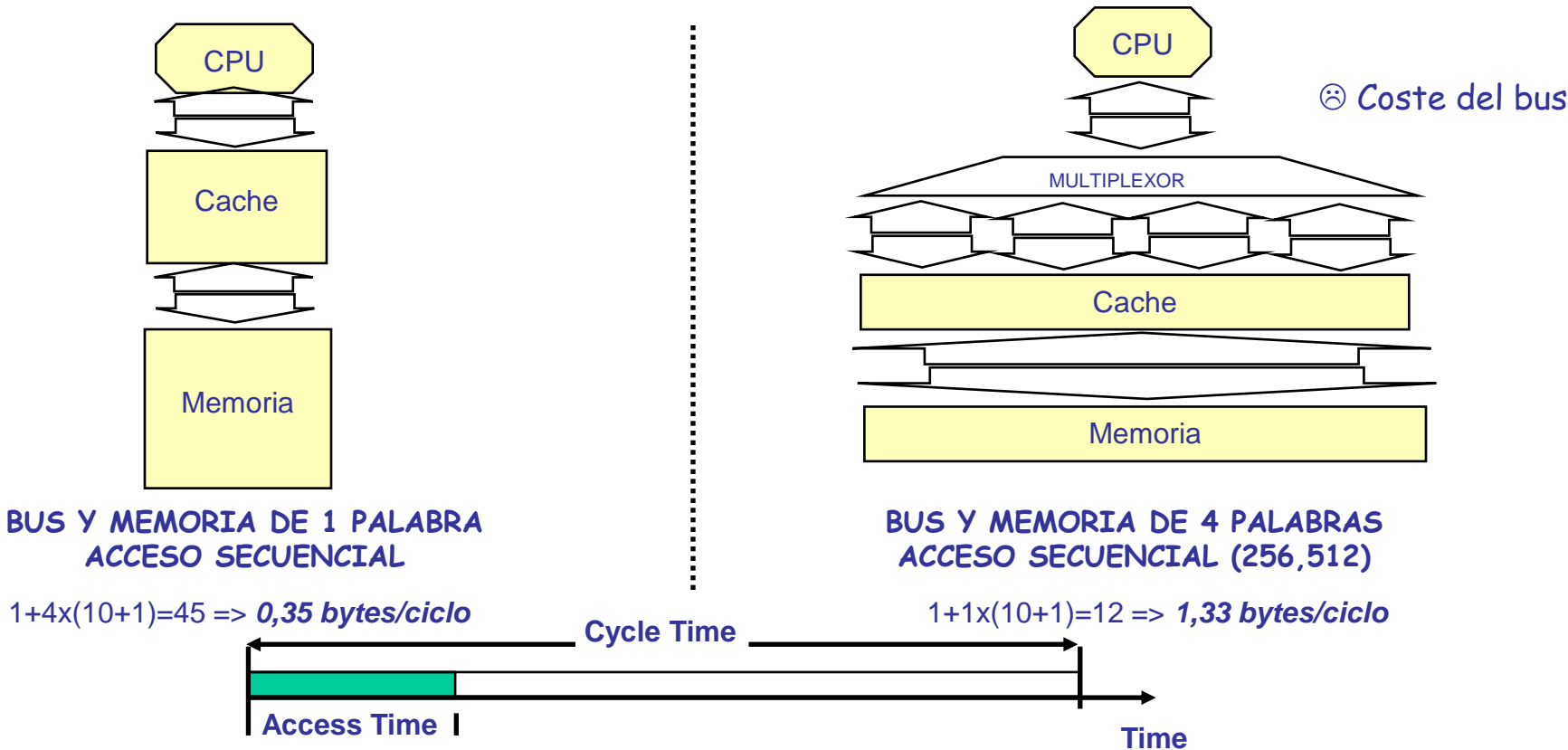
Compiladores, SO, Algoritmos, estructuras de datos,...

❑ 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, 1% 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

❑ Aumentar el ancho de banda con memoria

- **Aumento del ancho de banda** en la transferencia de un bloque manteniendo la misma latencia
 - **DRAM:** 1 ciclo en enviar la dirección, 10 ciclos en el acceso y 1 ciclo en el envío del dato, tamaño de palabra 32bits.
 - **Cache:** Bloques de tamaño 4 palabras de 4 bytes

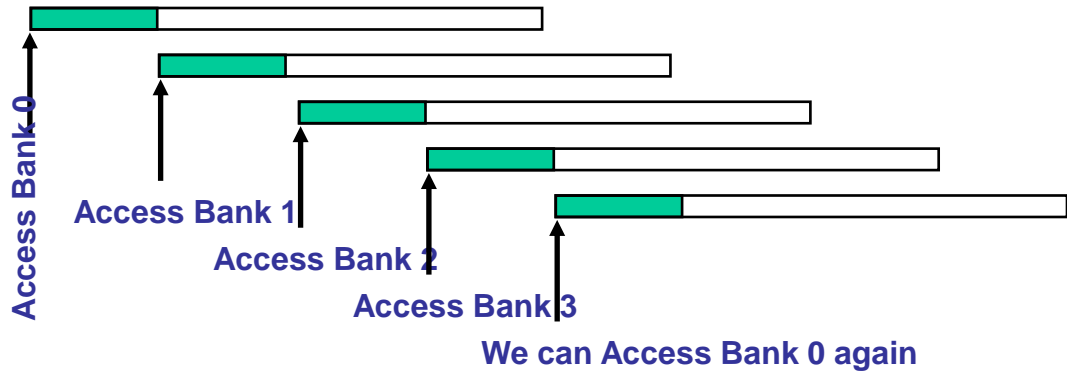
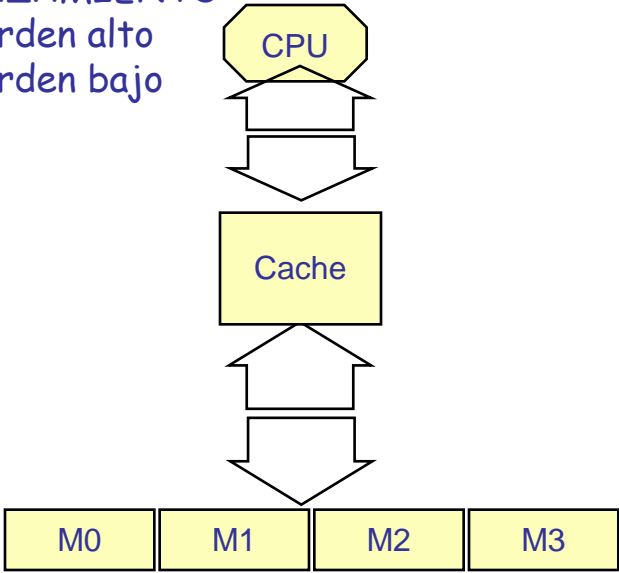


❑ Aumentar el ancho de banda con memoria

- **Aumento del ancho de banda** en la transferencia de un bloque manteniendo la misma latencia
 - DRAM: 1 ciclo en enviar la dirección, 10 ciclos en el acceso y 1 ciclo en el envío del dato, tamaño de palabra 32bits.
 - Cache: Bloques de tamaño 4 palabras de 4 bytes

ENTRELAZAMIENTO

- Orden alto
- Orden bajo



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

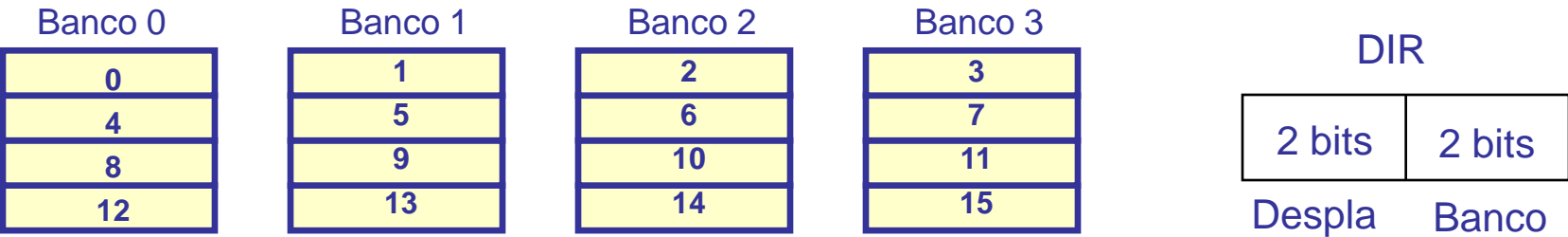
$1+(10+4)\times 1=15 \Rightarrow 1 \text{ byte/ciclo}$

☺ Muy buena **relación coste/rendimiento**

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

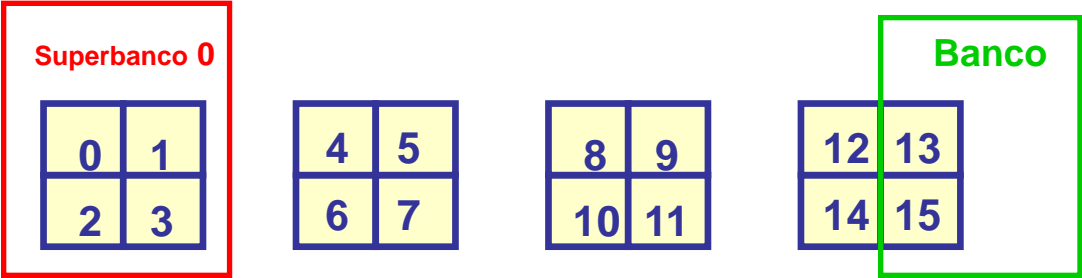
❑ Aumentar el ancho de banda con memoria

Distribución de direcciones con entrelazamiento de orden bajo



- Accesos independientes a cada banco (caches que no bloquean)
 - Cada banco necesita interfase con el sistema. Controlador, buses separados de direcciones y datos
 - Mezclar los tipos de entrelazamiento (entrelazamiento mixto)

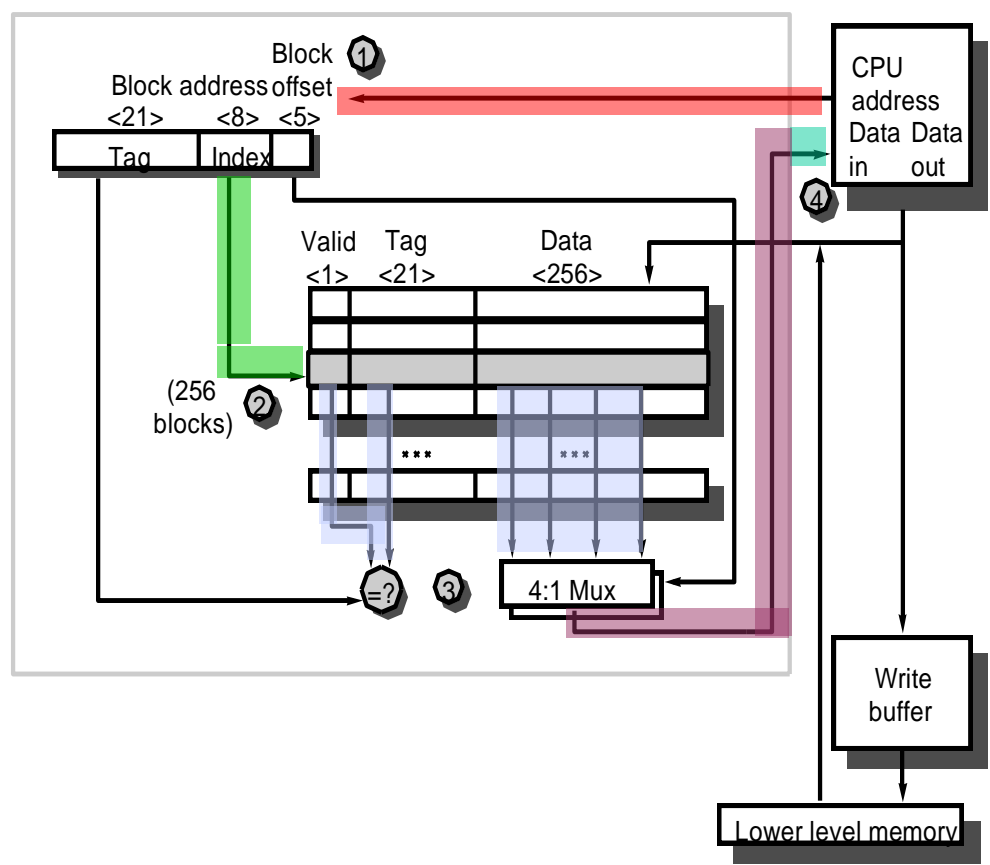
Numero de superbanco	Desplazamiento superbanco	
	Desplazamiento	Numero de banco



Ejemplo:
4 accesos independientes
Cada acceso dos palabras

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

- **CACHE de primer nivel L1**
- **Tamaño:** 8 KB datos y 8 KB de instrucciones (solo lectura)
- **Tamaño del bloque:** 32 bytes (4 palabras de 64 bits)
- **Número de bloques:** 256
- **Política de emplazamiento:** Directo
- **Política de reemplazamiento:** Trivial
- **Política de actualización:** Escritura directa

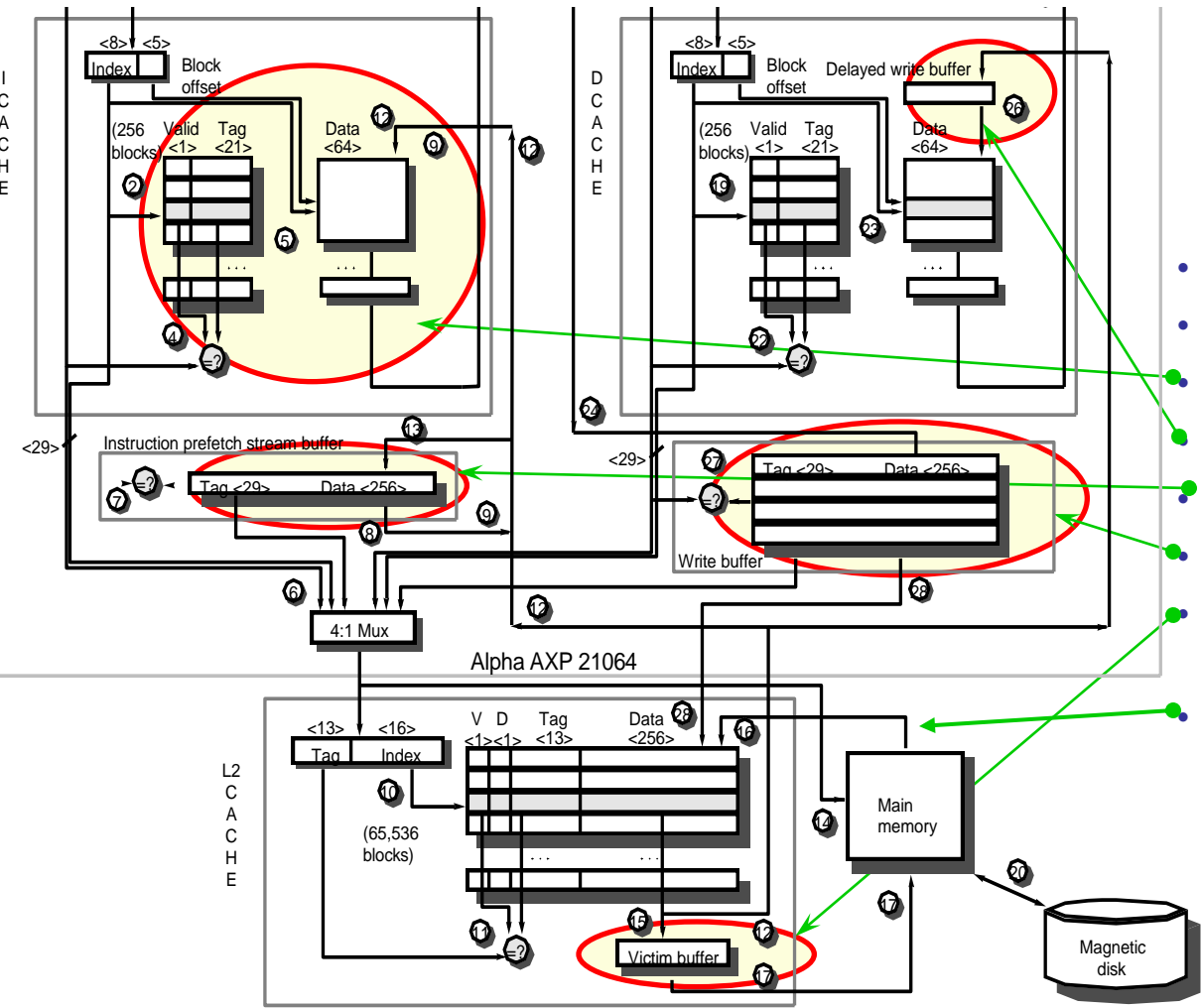


1. Dirección física de 34 bits (división en campos)
 - MS: cacheable o E/S (no cacheable)
2. Selección de marco de bloque por el índice
3. Comparación y lectura simultánea
4. Informar que el dato está listo en cache

En caso de fallo

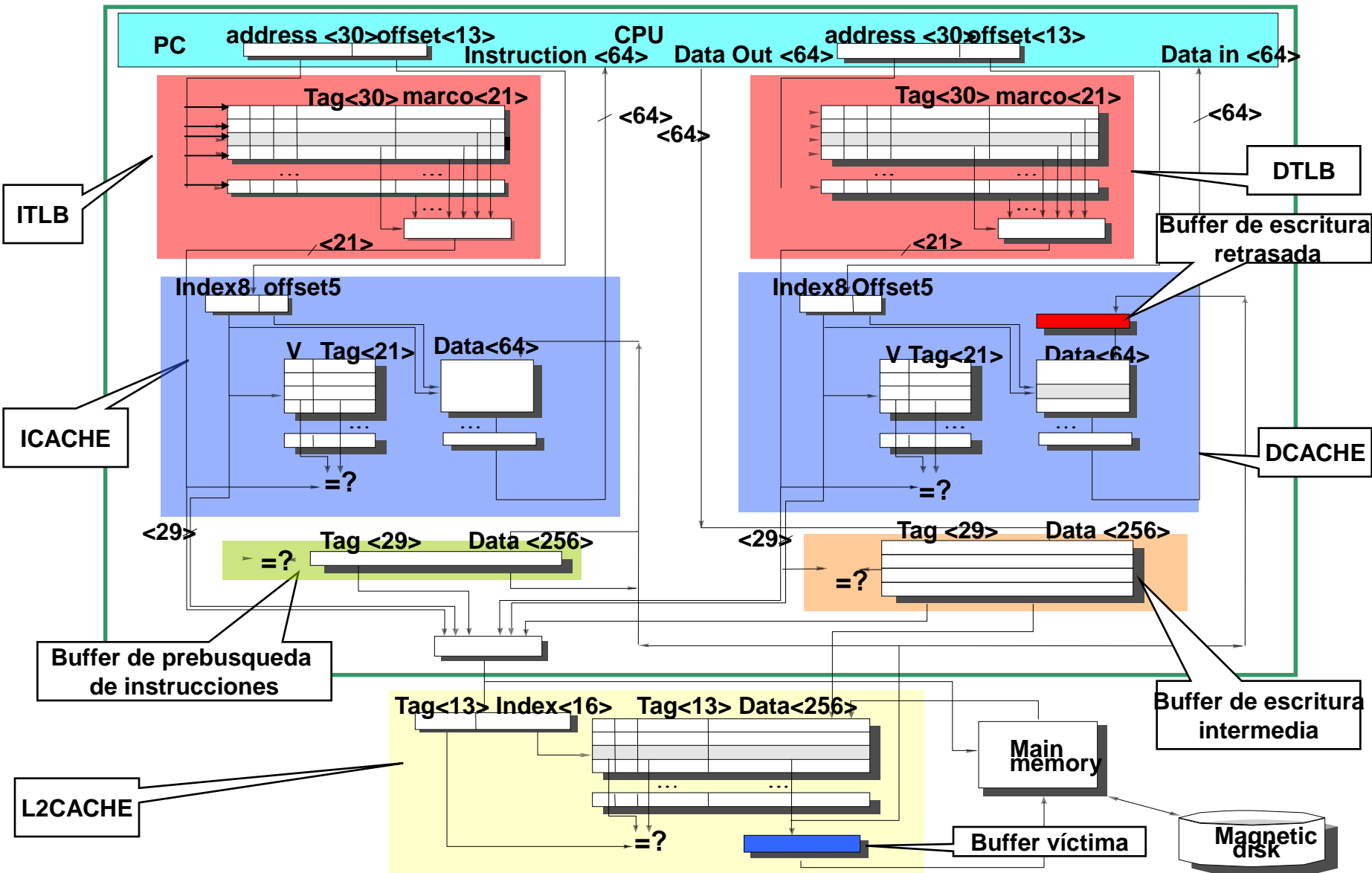
4. Indica al procesador que espere
5. Lectura del bloque desde siguiente nivel (Mp o L2)

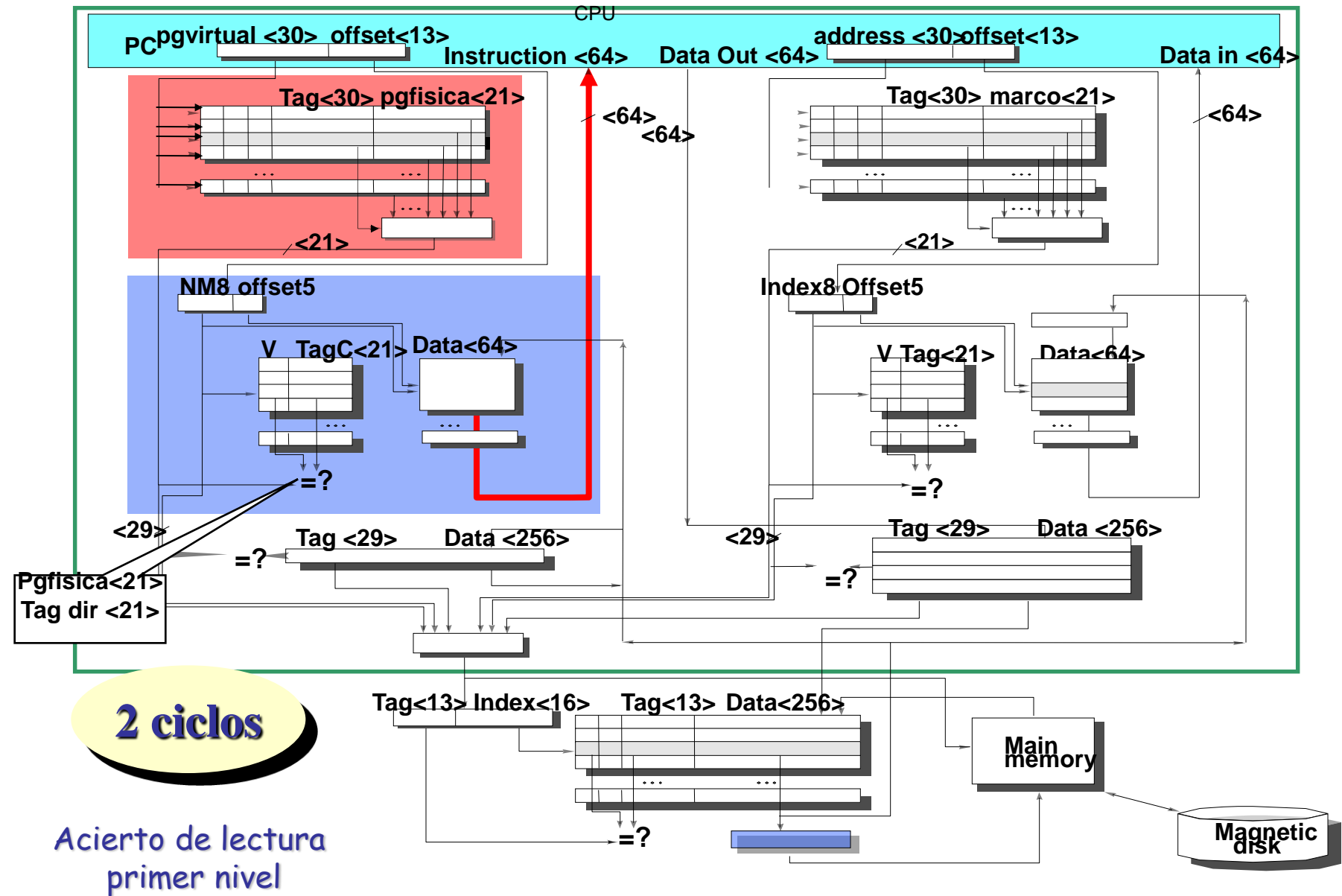
Visión global Alpha 21064



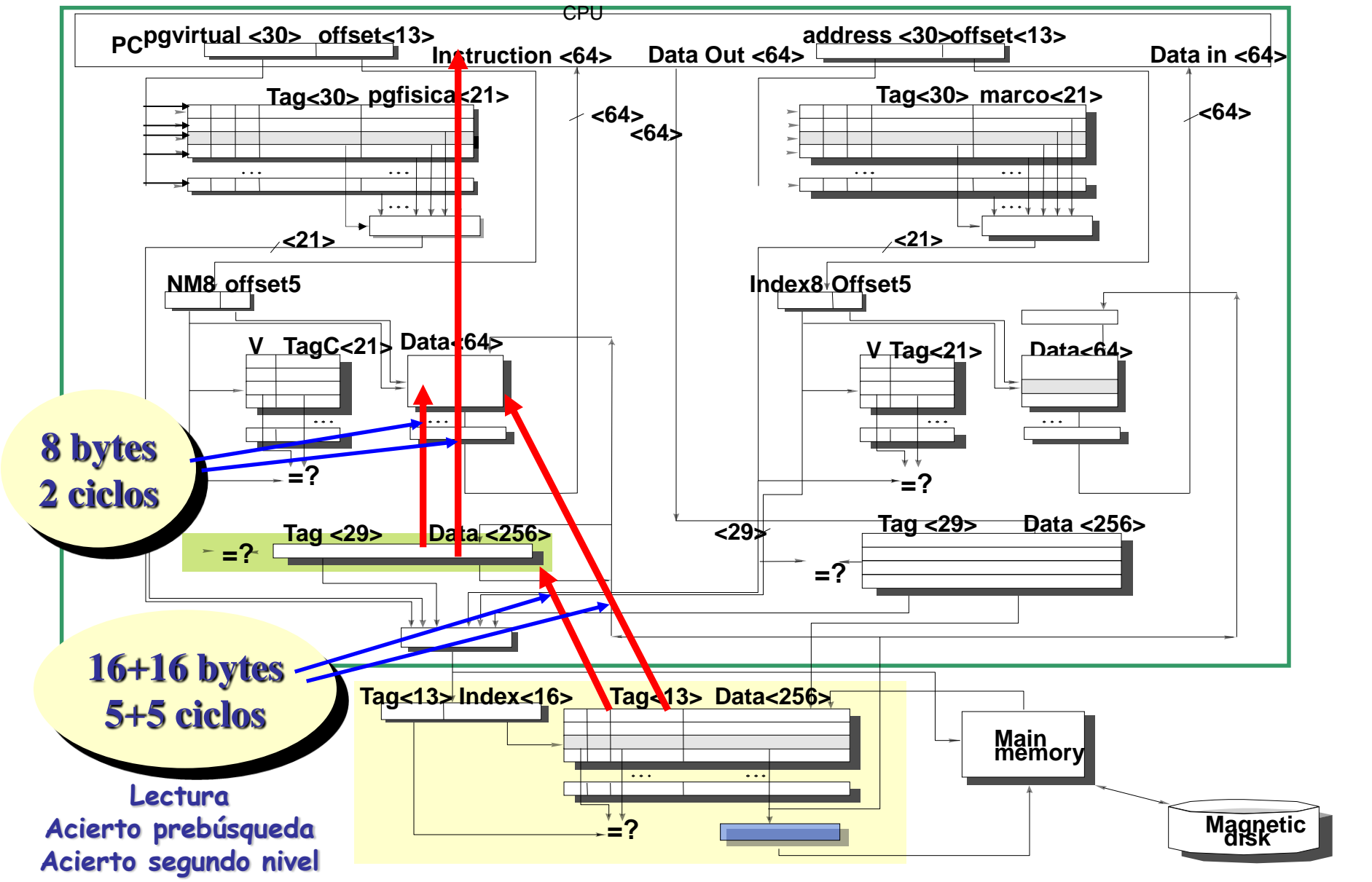
- División de memorias cache
- TLB asociativo
- Solapamiento de comparación y la lectura
- Solapamiento de escrituras consecutivas
- Prebúsqueda hw de instrucciones (1 bloque)
- Buffer de escritura de 4 bloques
- 2 MB L2 de cache de emplazamiento directo con cache víctima de 1 bloque
- 256 bit a memoria

Visión global Alpha 21064

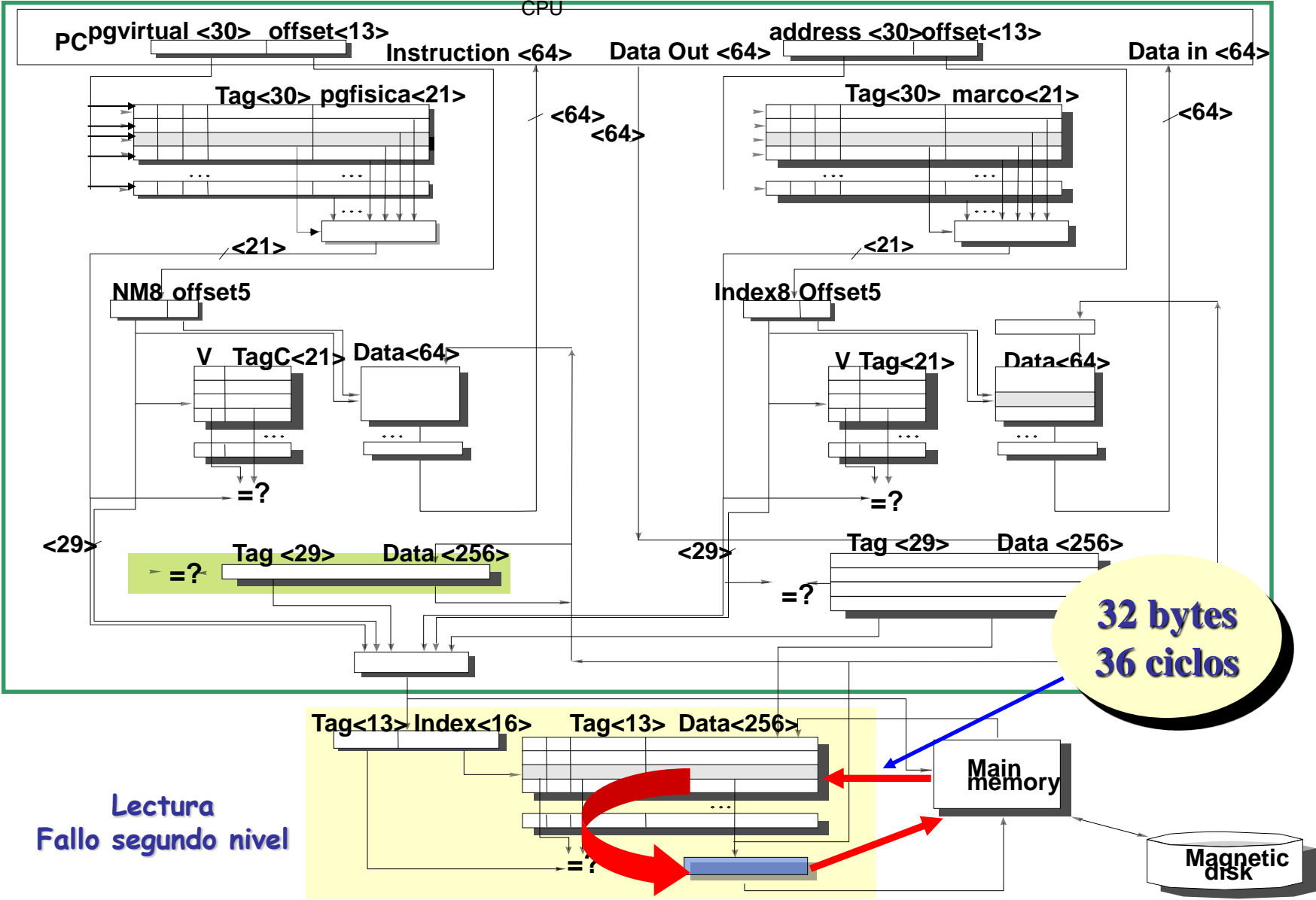




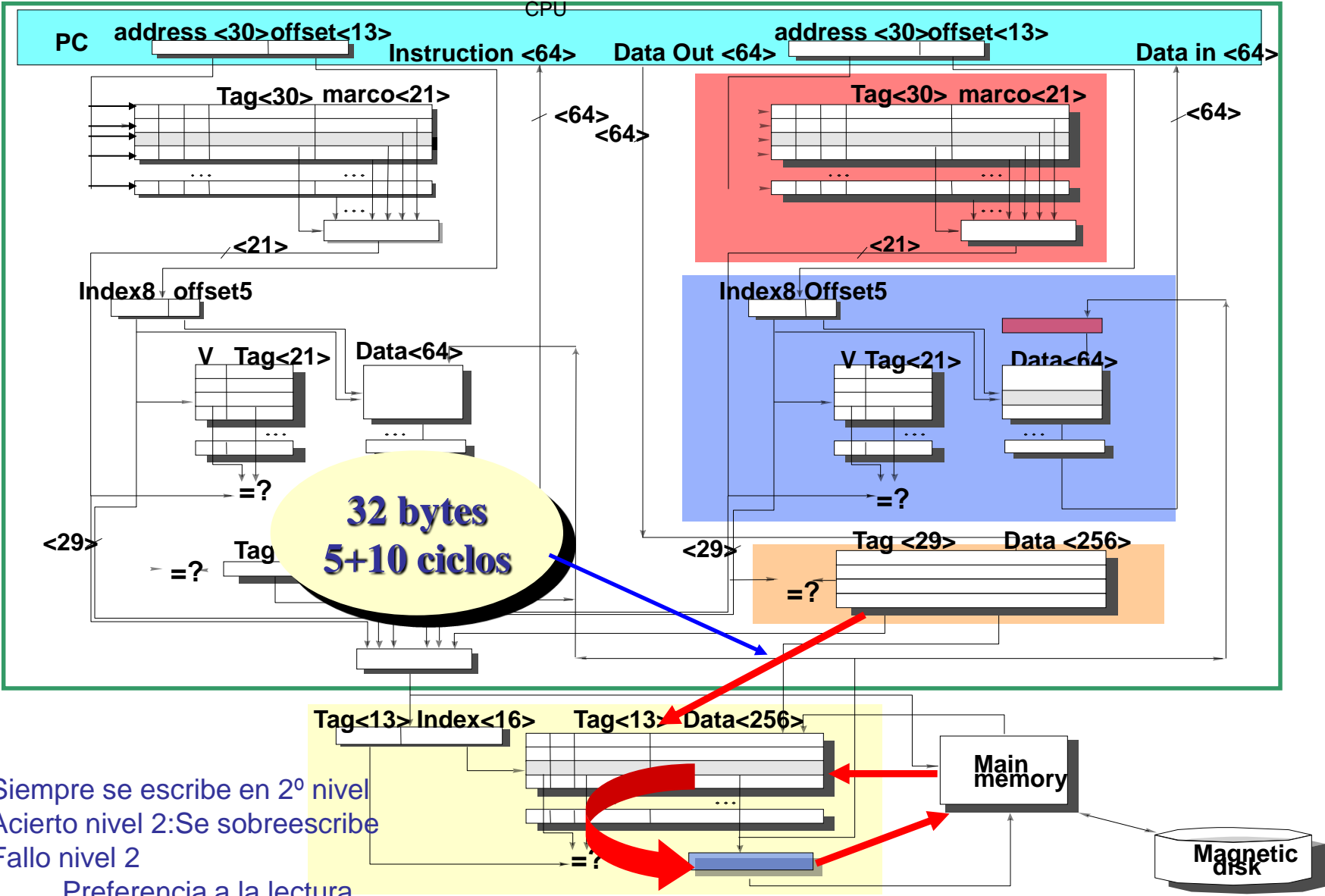
Visión global Alpha 21064



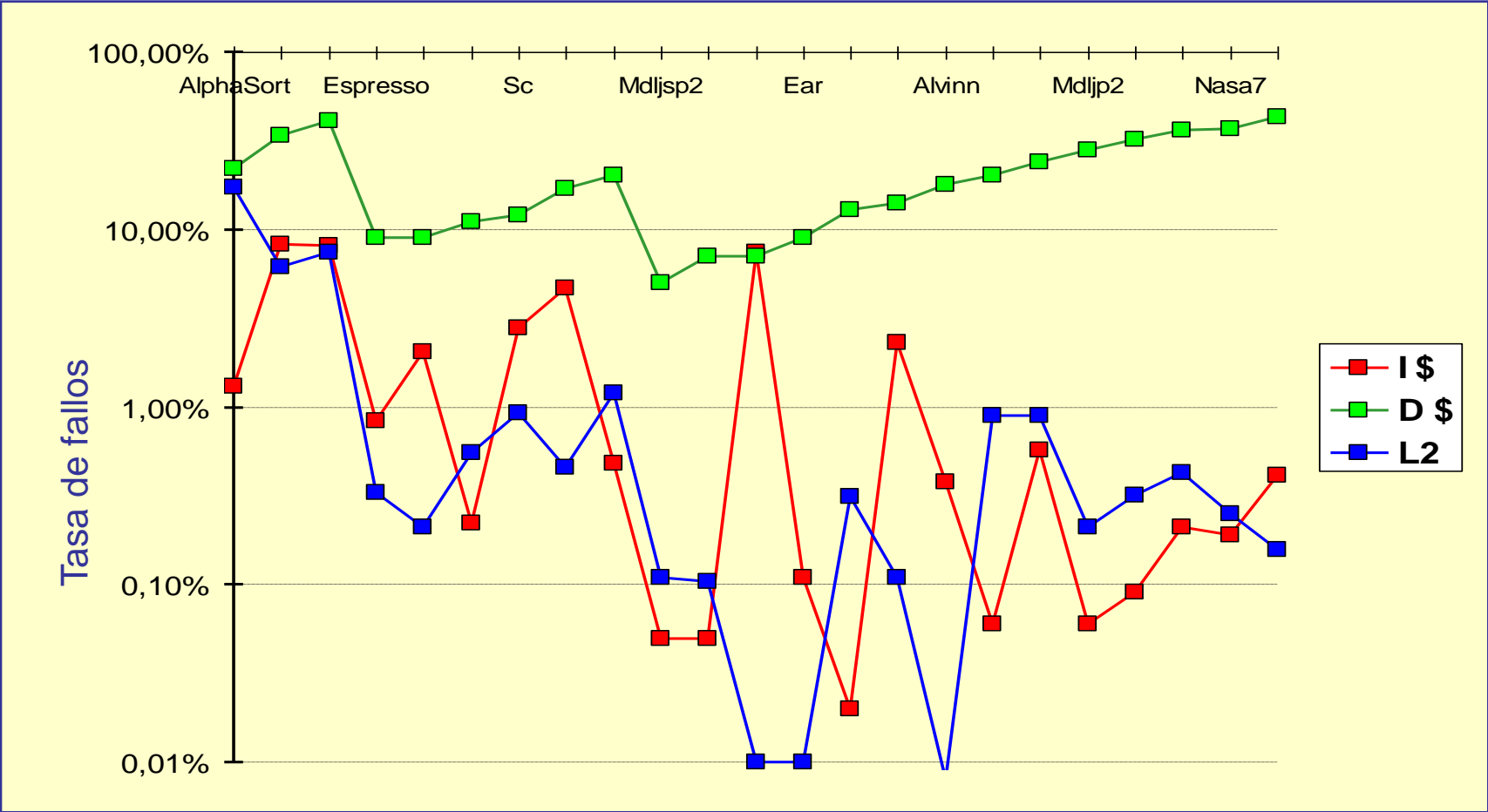
Visión global Alpha 21064



Visión global Alpha 21064



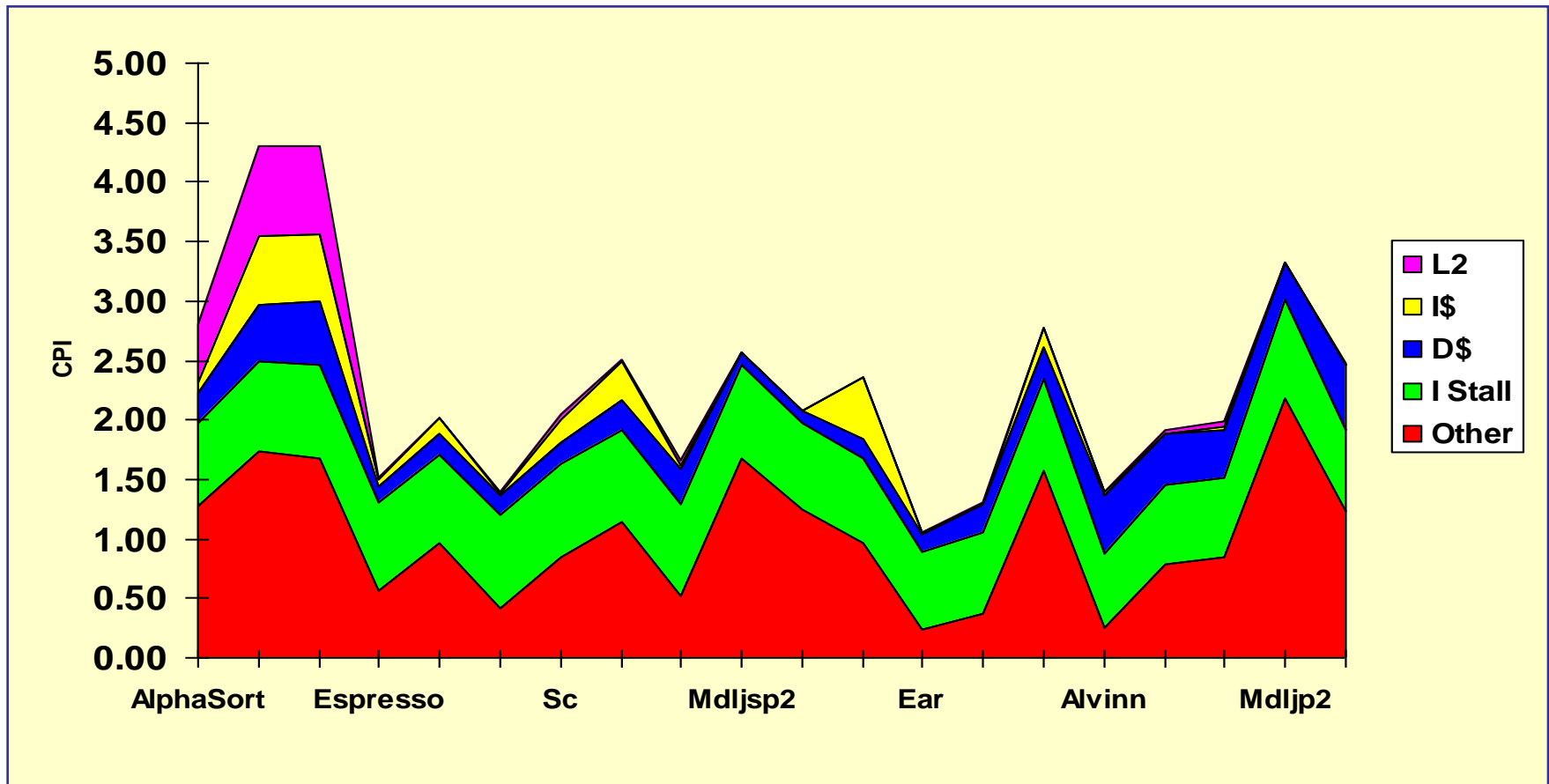
❑ Rendimiento SPEC92



Specint: I\$ miss = 2%, D\$ miss = 13%, L2 miss = 0.6%
Specfp: I\$ miss = 1%, D\$ miss = 21%, L2 miss = 0.3%
WL comercial: I\$ miss = 6%, D\$ miss = 32%, L2 miss = 10%

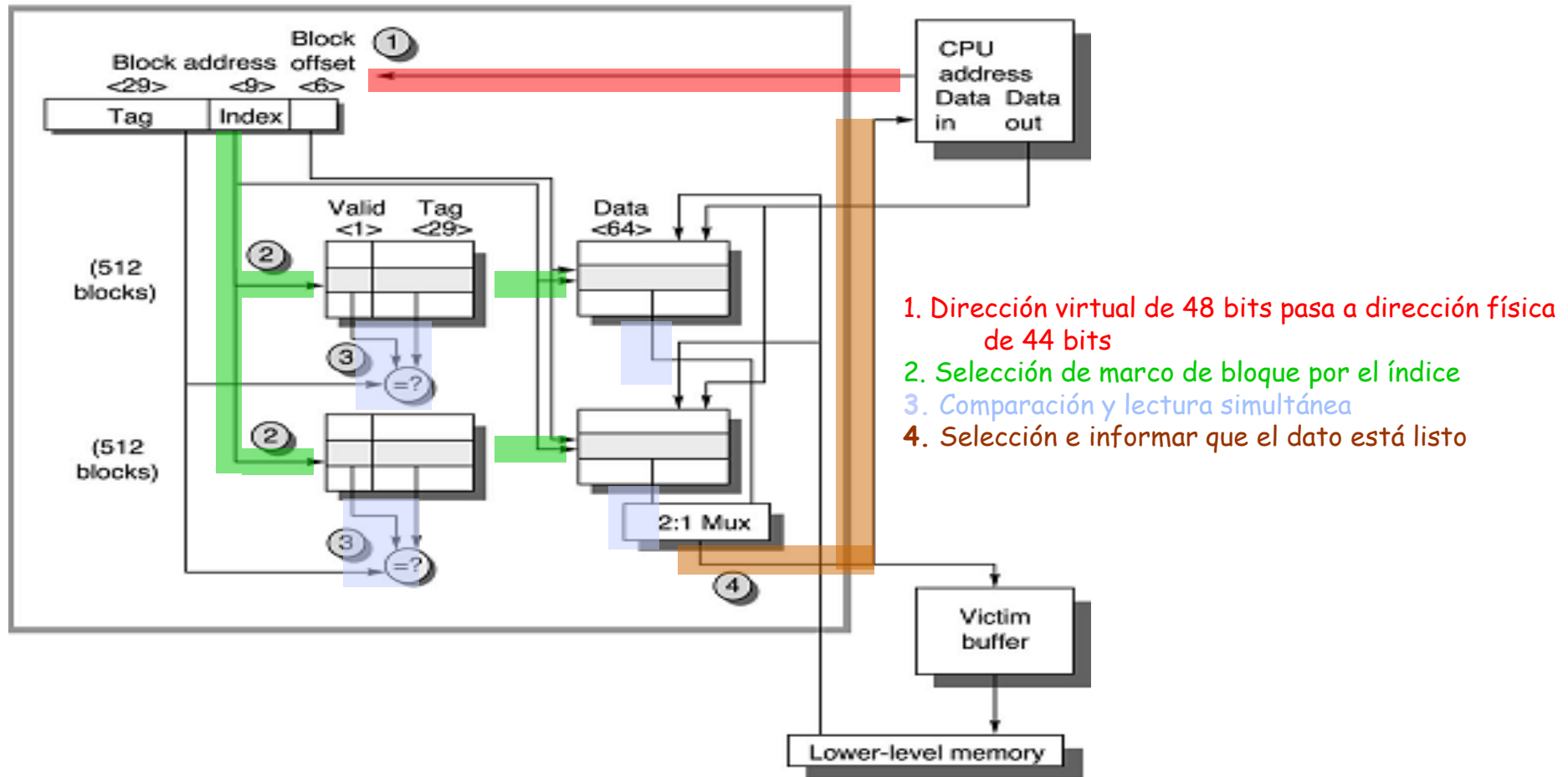
□ Rendimiento CPI ideal 0.5

- ✓ Parada en el lanzamiento de instrucciones saltos, dependencias
- ✓ Esperas de las diferentes caches
- ✓ Otros conflictos estructurales, registros,...

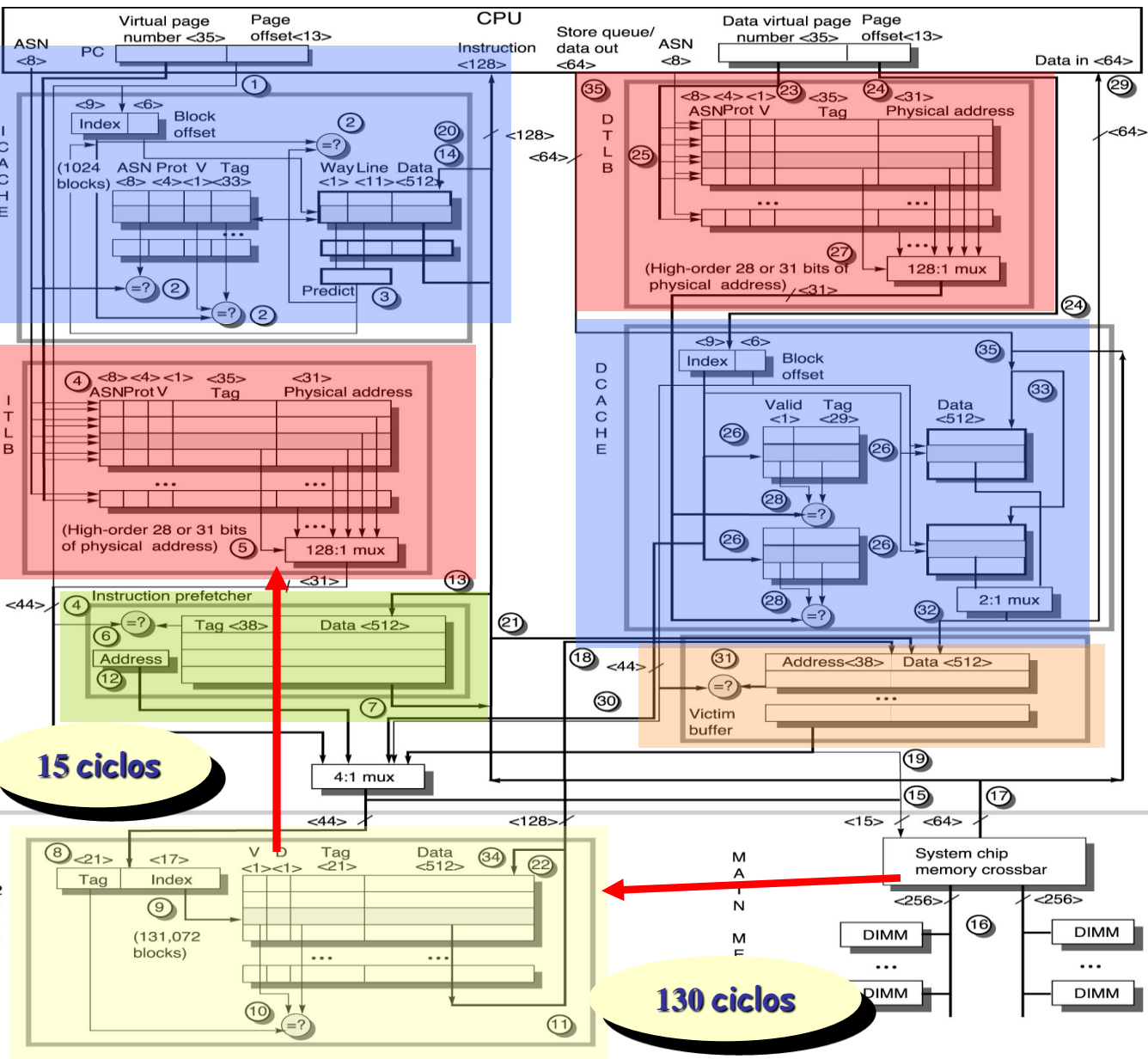


Visión global Alpha 21264 (ES40)

- **CACHE de DATOS L1**
- **Tamaño:** 64KB datos , ASOCIATIVA 2 vias
- **Tamaño del bloque:** 64 bytes (8 palabras de 64 bits), 1024 bloques



Visión global Alpha 21264 (ES40)



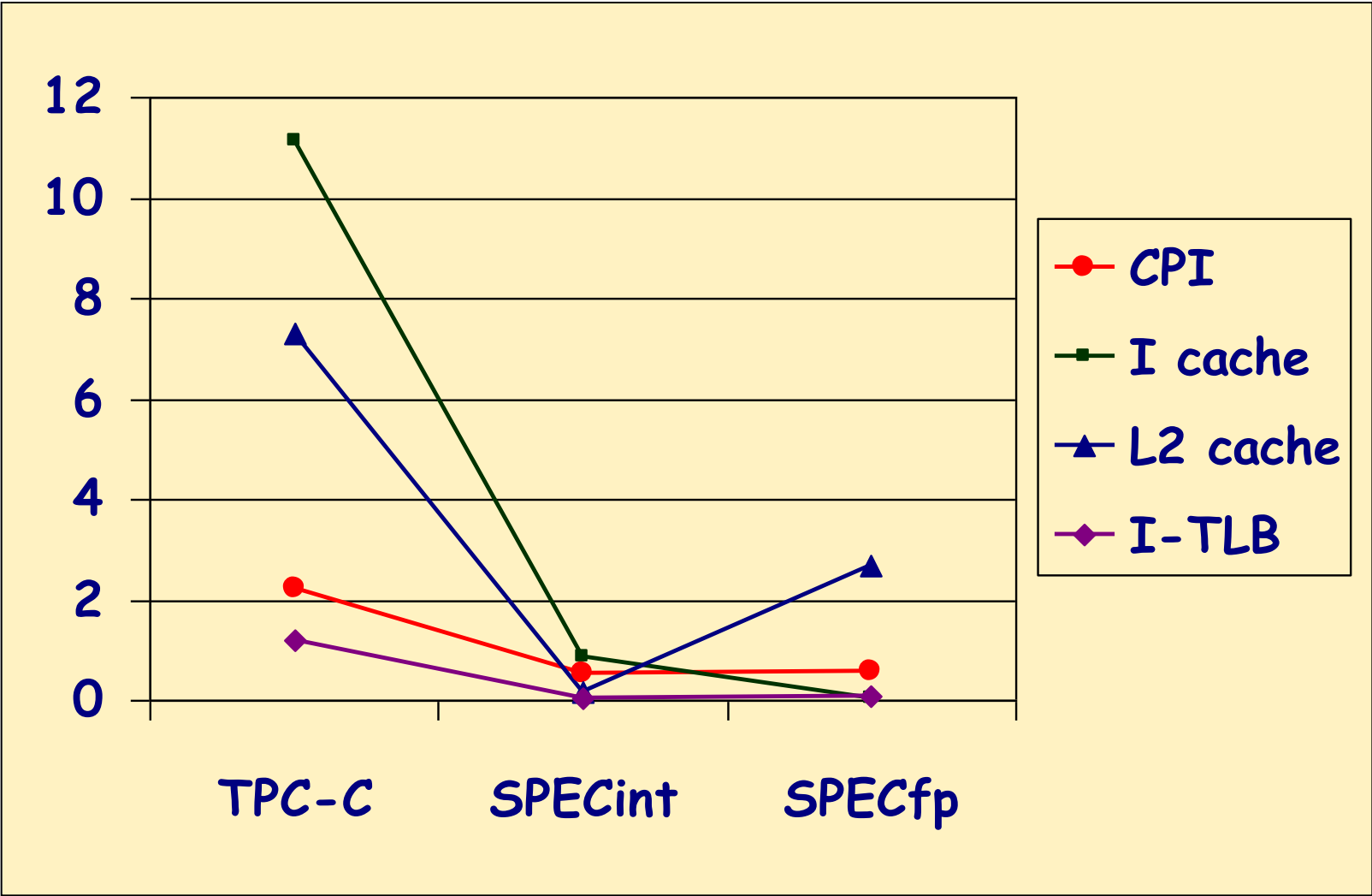
Caches L1

- o Cache de instrucciones acceso virtual
- o Escritura directa
- o Buffer de escritura intermedia para acelerar la escritura directa
- o Buffer de escritura retrasada para segmentar las escrituras
- o Buffer de prebúsqueda de un bloque de instrucciones

Cache L2

- o Post-escritura
- o Buffer víctima para dar preferencia a la lectura
- o Directa
- o 8Mb, 64 bytes bloque
- o Critico primero (16)

❑ Rendimiento SPEC95



Fallos por 1000 instrucciones y CPI para diferentes cargas de trabajo

❑ 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-throught	64KB, 8 ways, 128 bytes, write-throught
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

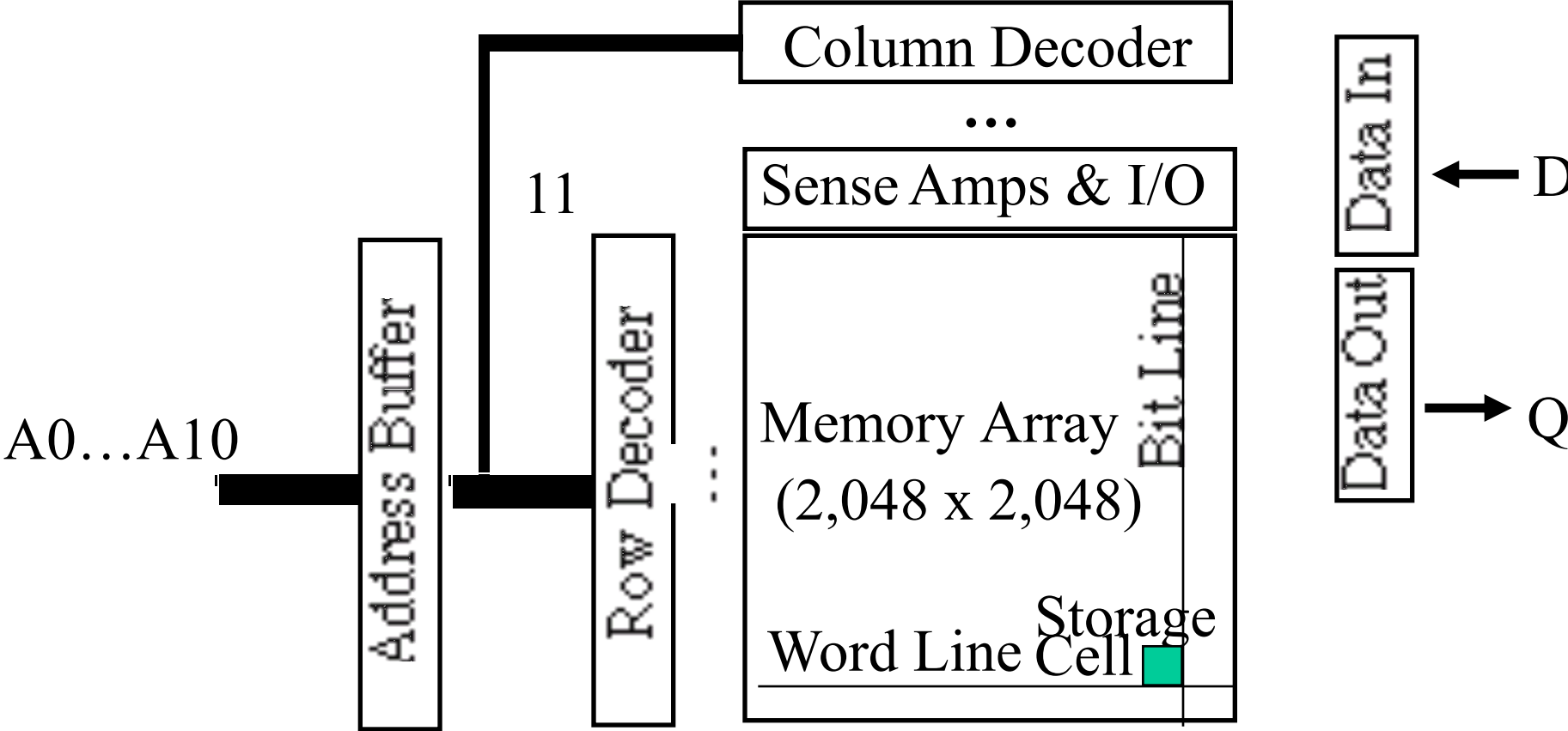
☐ Sun Niagara I-II

	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

❑ 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

DRAM logical organization (4 Mbit)



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.

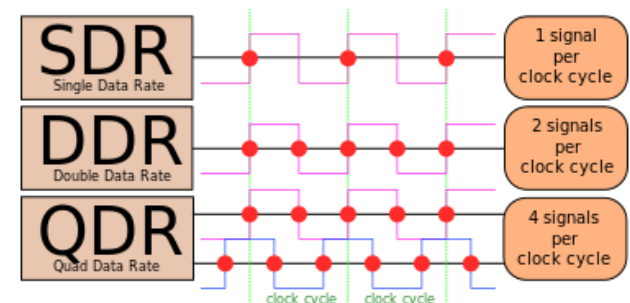
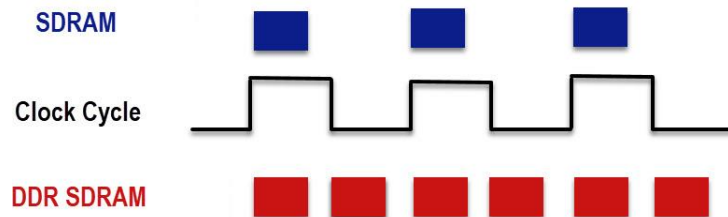
2. Synchronous DRAM (SDRAM)

- 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, I/O bus clock hasta 800 MHz
- o DDR4-3200, PC4-25600, Memory clock 400 MHz, I/O bus clock 1600 MHz, Theoretical bandwidth 3.2 GT/s 25.6 GB/s

Mejora en AB, no en latencia



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

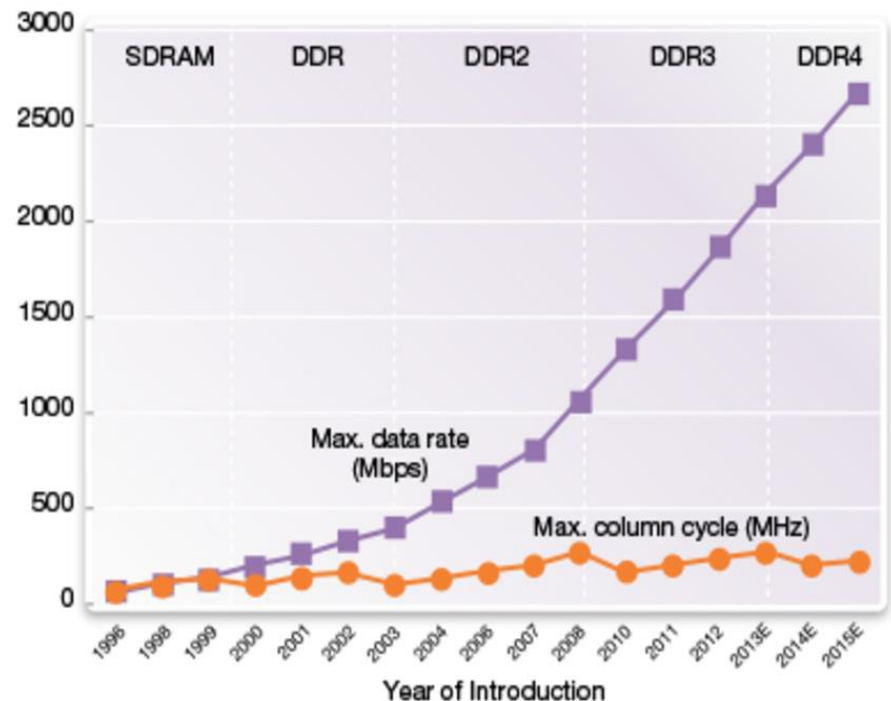
x 2

x 8

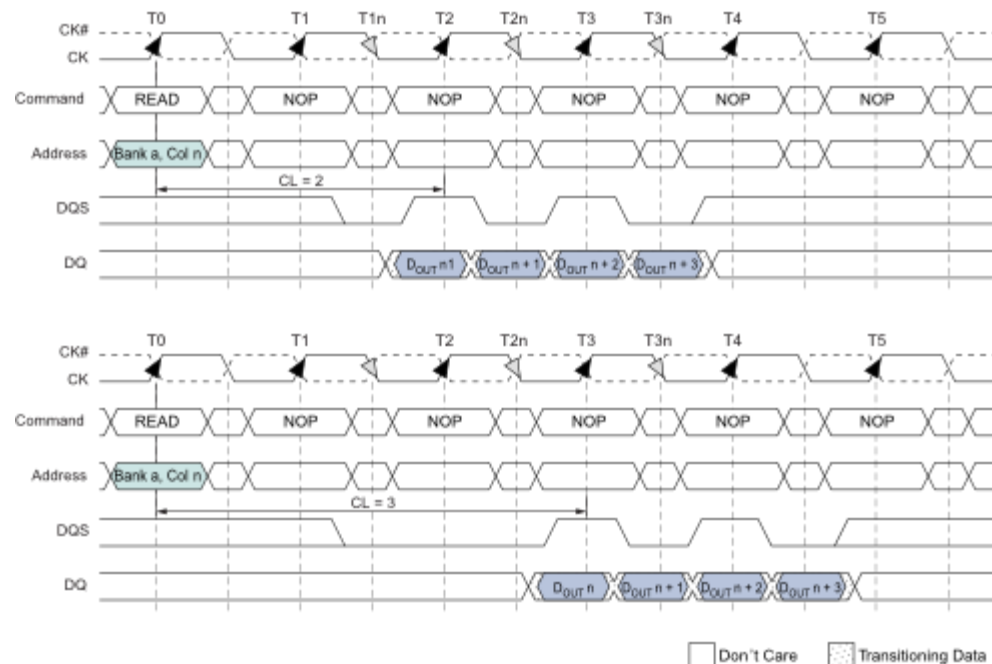
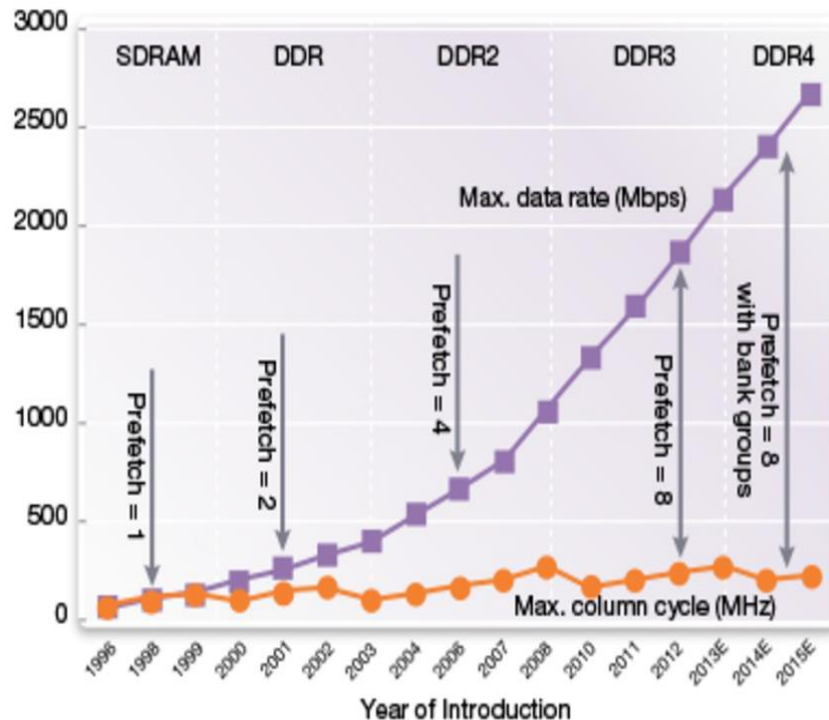
- ❑ La función de grupos de bancos, utilizada en las SDRAM DDR4, se tomó de las memorias gráficas GDDR5.
- ❑ Para comprender la necesidad de grupos de bancos, se debe entender el concepto de recuperación previa de SDRAM DDR.
- ❑ Prefetch es el término que describe cuántas palabras de datos se recuperan cada vez que se ejecuta un comando de columna con memorias DDR.

DDR4 Prefetch y rendimiento

- ❑ La Figura muestra la velocidad de datos máxima para cada generación de SDRAM en comparación con el ciclo de columna máximo.
- ❑ La Figura muestra que el núcleo es bastante lento y ha cambiado poco con el tiempo, mientras que la velocidad de la interfaz ha aumentado significativamente con el tiempo.



- ❑ Cada vez que se ejecuta un ciclo de columnas, se accede a una palabra de datos de la SDRAM.
- ❑ Ahora, mientras que la SDRAM realmente tenía algo llamado una regla 2N, que podía acomodar una búsqueda previa de dos
- ❑ Sin embargo, una vez que se introdujo la SDRAM DDR, el núcleo ya no podía mantenerse con el ancho de banda requerido de la SDRAM.
- ❑ Con la captación previa de cuatro de DDR2 y la captación previa de ocho de DDR3, la separación se hizo aún más amplia. DDR4 sigue utilizando una captación previa de ocho



❑ DDR4 y Grupos de Bancos

