



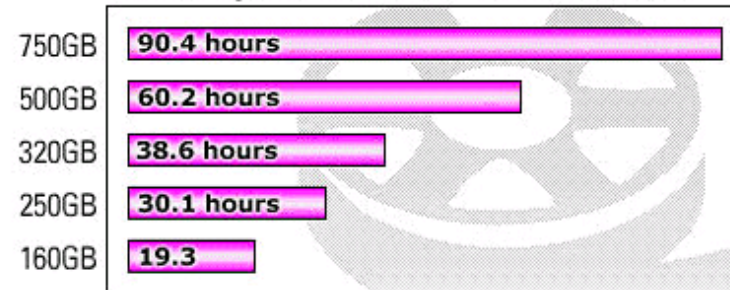
Módulo 4. Jerarquía de memoria:

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

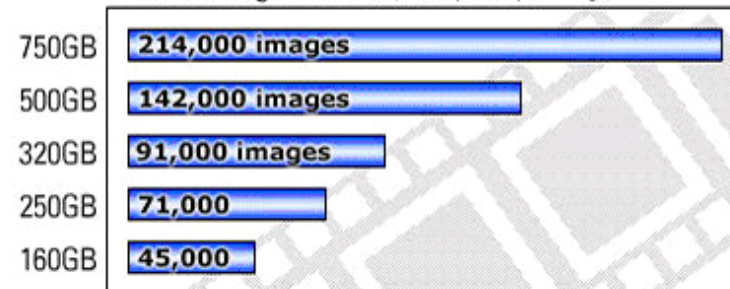
Introducción

- Los datos ocupan espacio en memoria
- Acceder a ellos implica un cierto tiempo

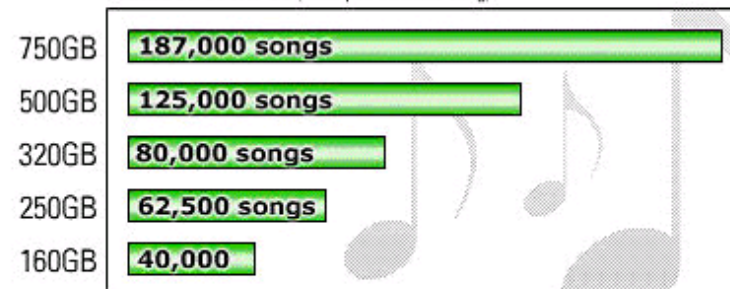
Hours of High Definition Video (Digital video - 8.3GB per hour)



Number of Digital Photos (3.5MB per 6 Mpixel image)



Number of MP3s (4MB per 4 minute song)





Introducción

- Valores típicos de tiempos de acceso y precios por Gbyte (año 2008)

Tipo de memoria	Tiempo de acceso (ns)	\$ por Gbyte
SRAM	0.5-2.5	2000-5000
DRAM	50-70	20-75
Disco magnético	5.000.000-20.000.000	0.20-2



Jerarquía de Memoria

- **Objetivo:**
 - Conseguir una memoria de gran tamaño, rápida y al menor coste posible.
 - De forma transparente al usuario
- ¿Cómo organizar la memoria?
- **Base:** Principio de localidad

“Cualquier programa accede a una porción relativamente pequeña de su espacio de direcciones en cualquier instante de tiempo”

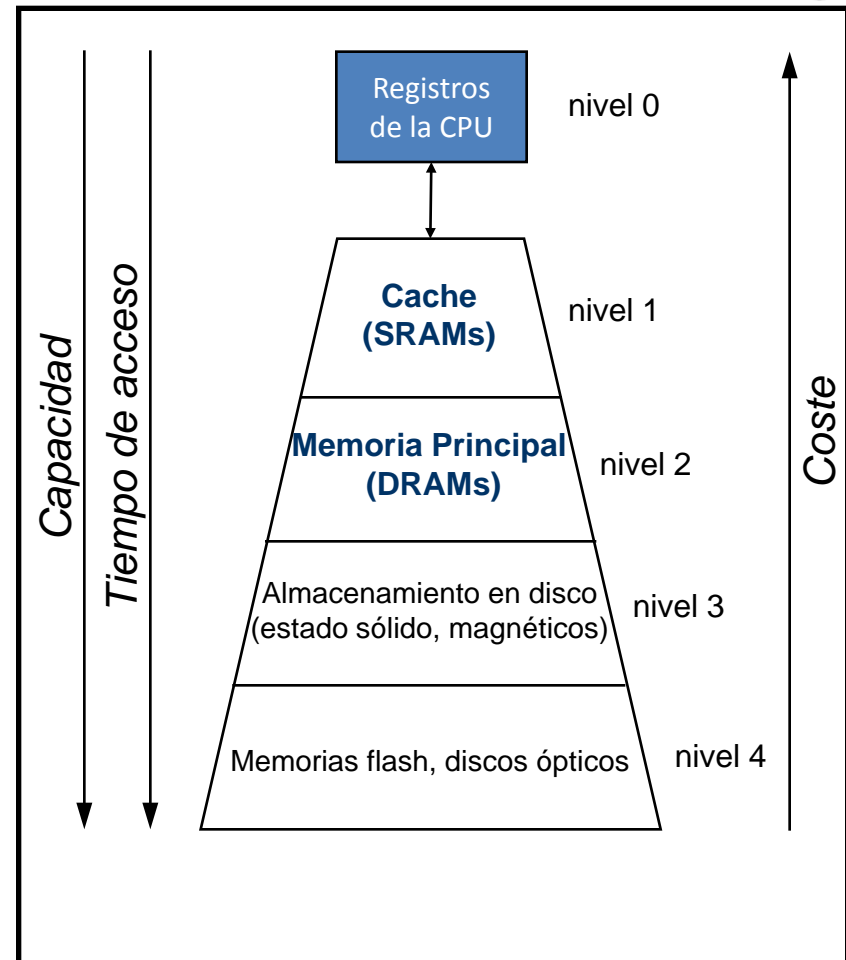


Jerarquía de memoria



Niveles de la Jerarquía de memoria

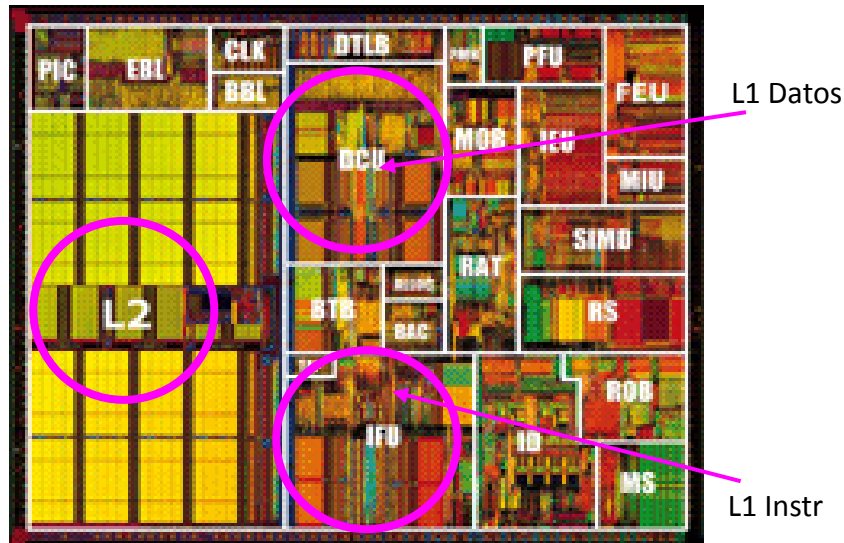
- Un computador típico está formado por diversos niveles de memoria, *organizados de forma jerárquica*:
 - Registros de la CPU
 - Memoria Cache
 - Memoria Principal
 - Memoria Secundaria (discos)
 - Memorias flash y CD-ROMs
- El coste de todo el sistema de memoria excede al coste de la CPU
 - Es muy importante optimizar su uso



Introducción

Tamaño de la cache

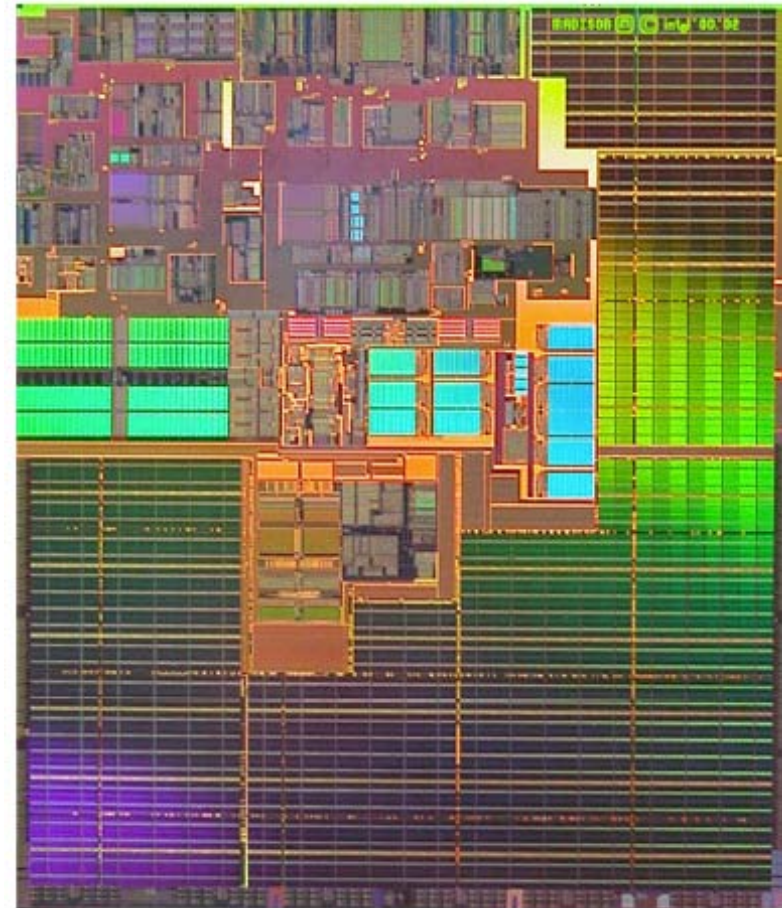
Del 50 al 75 % del área. Más del 80% de los transistores



PentiumIII

Latencia:

1ciclo (Itanium2) a 3 ciclos Power4-5



**Itanium 2
Madison**

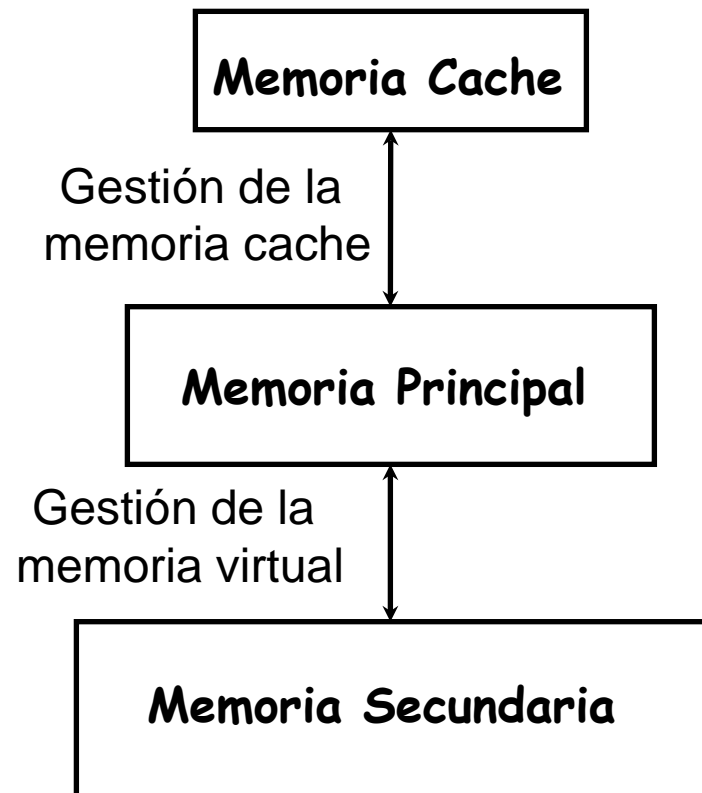
Introducción

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





Introducción

- **Jerarquía de memoria:** múltiples niveles de memoria de diferentes velocidades y tamaños
- 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 coherentes
 - Si un bloque de información se modifica en el nivel M_i , deben actualizarse los niveles M_{i+1}, \dots, M_n
 - Localidad
 - 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



Principio de localidad

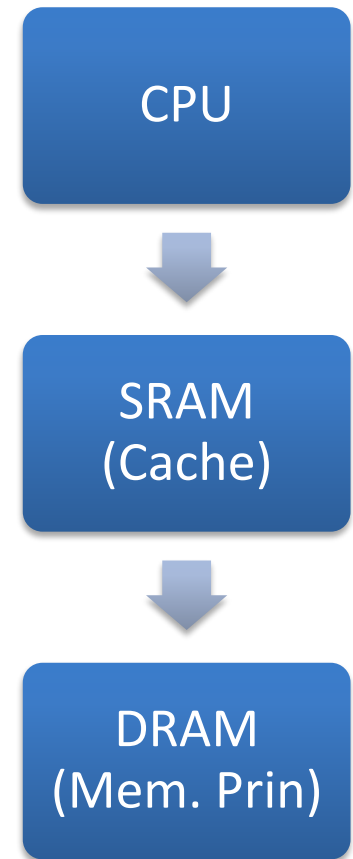
- **Localidad temporal:** Un dato usado en un determinado instante tiende a ser pronto reutilizado
 - Es habitual que un programa acceda la misma variable varias veces. Lo óptimo es mantenerla en registro, pero no siempre es posible
- **Localidad espacial:** Si un dato es utilizado en un determinado instante, es muy probable que los datos cercanos a él sean también pronto utilizados
 - Es habitual que un programa acceda datos que están almacenados en posiciones consecutivas
 - Los arrays se almacenan en posiciones consecutivas y se suelen acceder secuencialmente



¿Cómo explotar la localidad temporal?



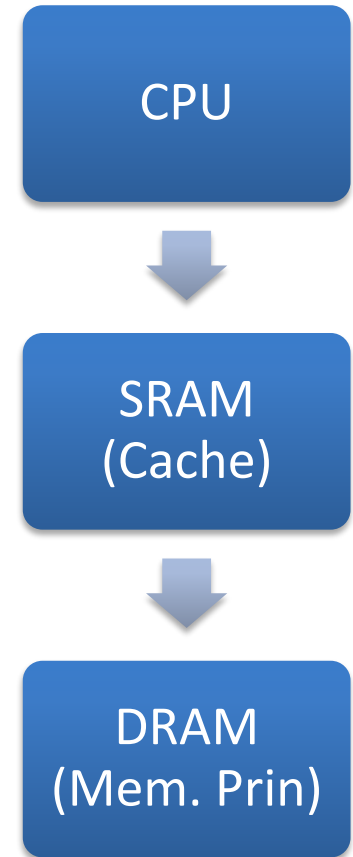
- Cada vez que se accede a memoria principal, llevamos una copia del dato a una memoria más pequeña y rápida (SRAM)
 - Introduce una penalización en el primer acceso
 - Será amortizada si se producen más accesos, pues se usará la copia y se evitará un acceso a memoria principal.





¿Cómo explotar la localidad espacial?

- Cada vez que se accede a la dirección i , llevamos una copia del dato a la cache
- Pero además, se copian varios datos más
 - Por ejemplo, si se accede al elemento $a[0]$, se traen a la cache $a[0], a[1], a[2]$ y $a[3]$
 - De nuevo, hay una penalización inicial que suele amortizarse.



Jerarquía de memoria

- La memoria más rápida (cache) se coloca más cerca del procesador → Accesos más rápidos
- Y el nivel de memoria más lento será el más alejado
- La búsqueda de los datos comienza siempre en la cache, y de no encontrarse en ella, continúa por los demás niveles de memoria

Ilusión: memoria rápida



Gestión de la jerarquía de memoria



- Vamos a trabajar con una memoria en 2 niveles
 - Cache y memoria principal
- **Bloque:** unidad mínima de transferencia entre los dos niveles
- **Acierto (hit):** el dato solicitado está en la cache
 - Tasa de aciertos (hit ratio): la fracción de accesos encontrados en el nivel i
 - 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 la cache y es necesario buscarlo en la memoria principal
 - Tasa de fallos (miss ratio): $1 - (\text{Tasa de aciertos})$
 - 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 .

Memoria Cache. Acceso Directo



- **Memoria pequeña y rápida** situada entre el procesador y la memoria principal

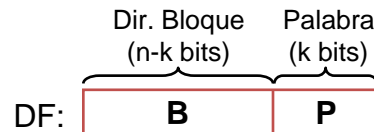
- Almacena una copia de la porción de información actualmente en uso de la memoria principal

- **Objetivo:** disminuir el tiempo de acceso a memoria

- **Estructura del sistema memoria cache/principal:**

- *MP (memoria principal):*

- formada por 2^n palabras direccionables
- “dividida” en nB bloques de tamaño fijo de 2^k palabras por bloque
- Campos de una dirección física:

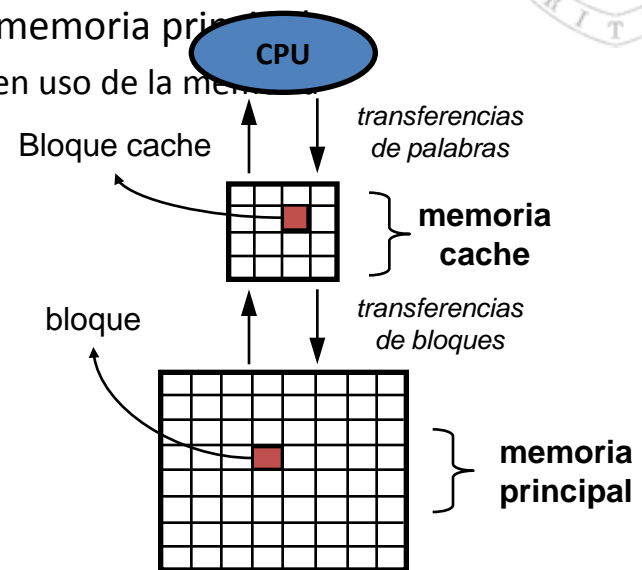


- *MC (memoria cache):*

- formada por nM bloques (o líneas) de 2^k palabras cada uno ($nM \ll nB$)

- *Directorio (en memoria cache):*

- Para cada bloque de MC, indica cuál es el bloque de MP que está alojado en él.



nB: número de bloques

nM: número de bloques de cache

B: dirección del bloque

M: dirección del bloque de cache

P: palabra dentro del bloque



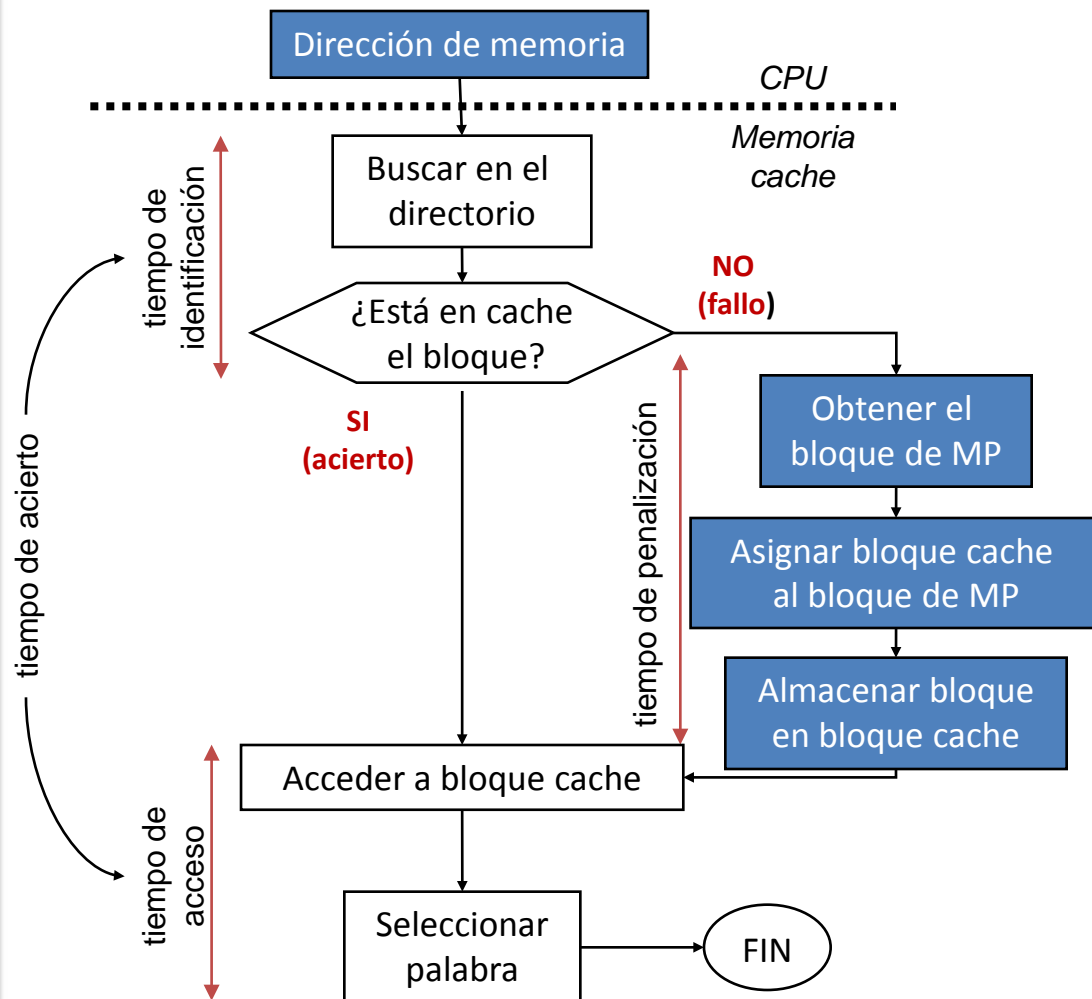
Gestión de la jerarquía de memoria



- Cuando la CPU genera una referencia, busca en la cache
 - Si la referencia no se encuentra en la cache: **FALLO**
 - Cuando se produce un fallo, no solo se transfiere una palabra, sino que se lleva un **BLOQUE** completo de información de la MP a la MC
 - *Por la propiedad de localidad temporal*
 - Es probable que en una próxima referencia se direcciona la misma posición de memoria
 - Esta segunda referencia no producirá fallo: producirá un **ACIERTO**
 - *Por las propiedades de localidad espacial*
 - Es probable que próximas referencias sean direcciones que pertenecen al mismo bloque
 - Estas referencias no producen fallo



Memoria cache



- Maximizar la tasa de aciertos
- Minimizar el tiempo de acceso
- Minimizar el tiempo de penalización
- Reducir el coste hardware

$$T_{total} = T_{acierto} + (1 - H)T_{penalizacion}$$

(frecuencia de hits)



Memoria Cache

- ¿Cómo sabemos que un dato está en la cache?
- Y si está, ¿cómo lo encontramos?



POLÍTICAS DE EMPLAZAMIENTO





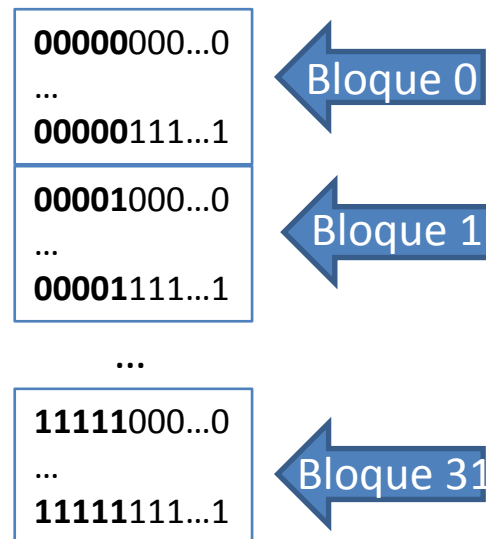
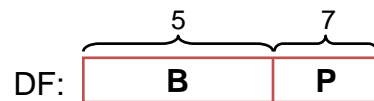
Políticas de emplazamiento

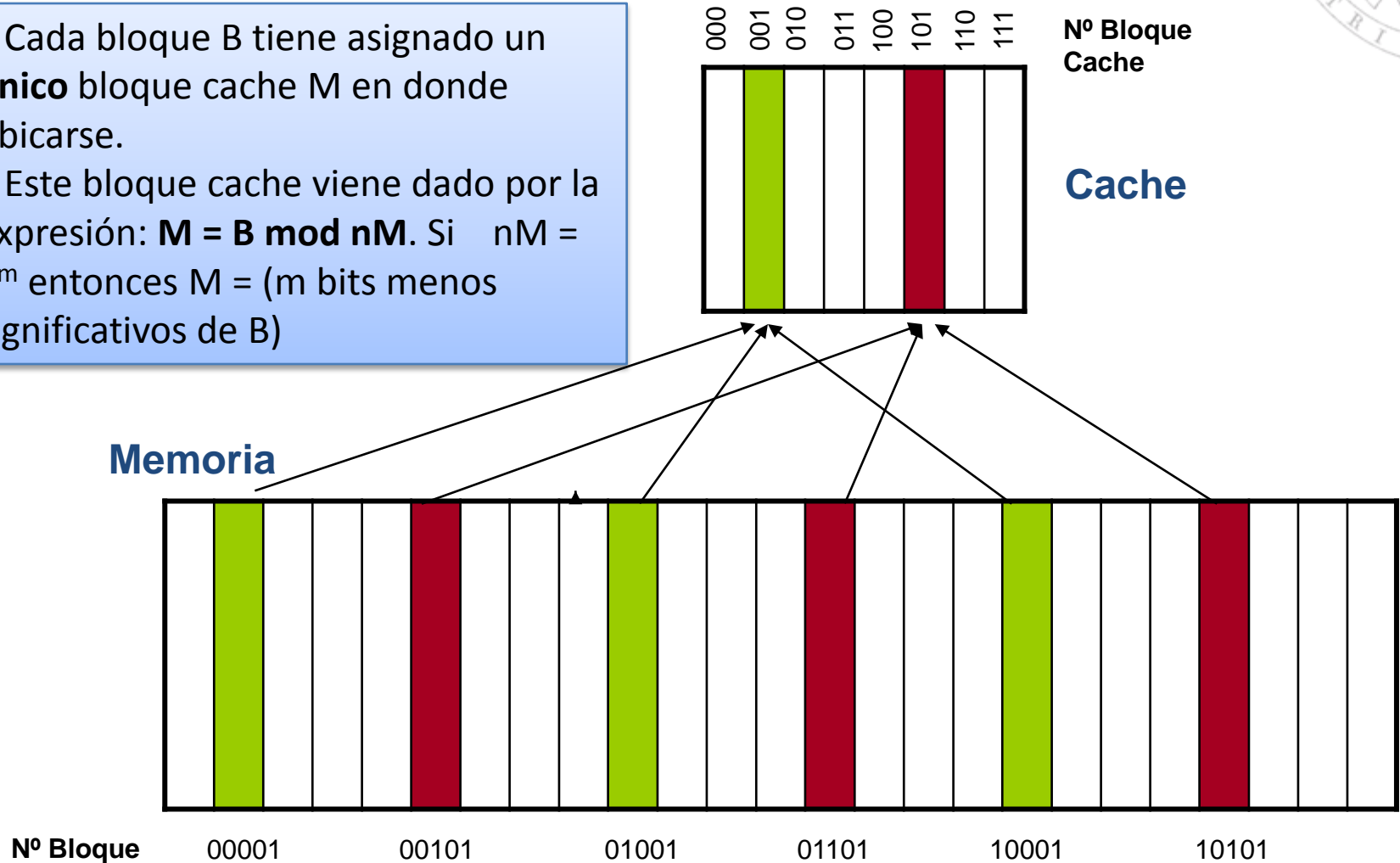
- Política de emplazamiento:
 - Necesaria ya que existen menos bloques en MC que bloques en MP
 - Determina en qué bloque, o bloques, de MC, puede cargarse cada bloque de MP
- Existen diferentes políticas:
 - Emplazamiento directo
 - Emplazamiento asociativo
 - Emplazamiento asociativo por conjuntos



Políticas de emplazamiento

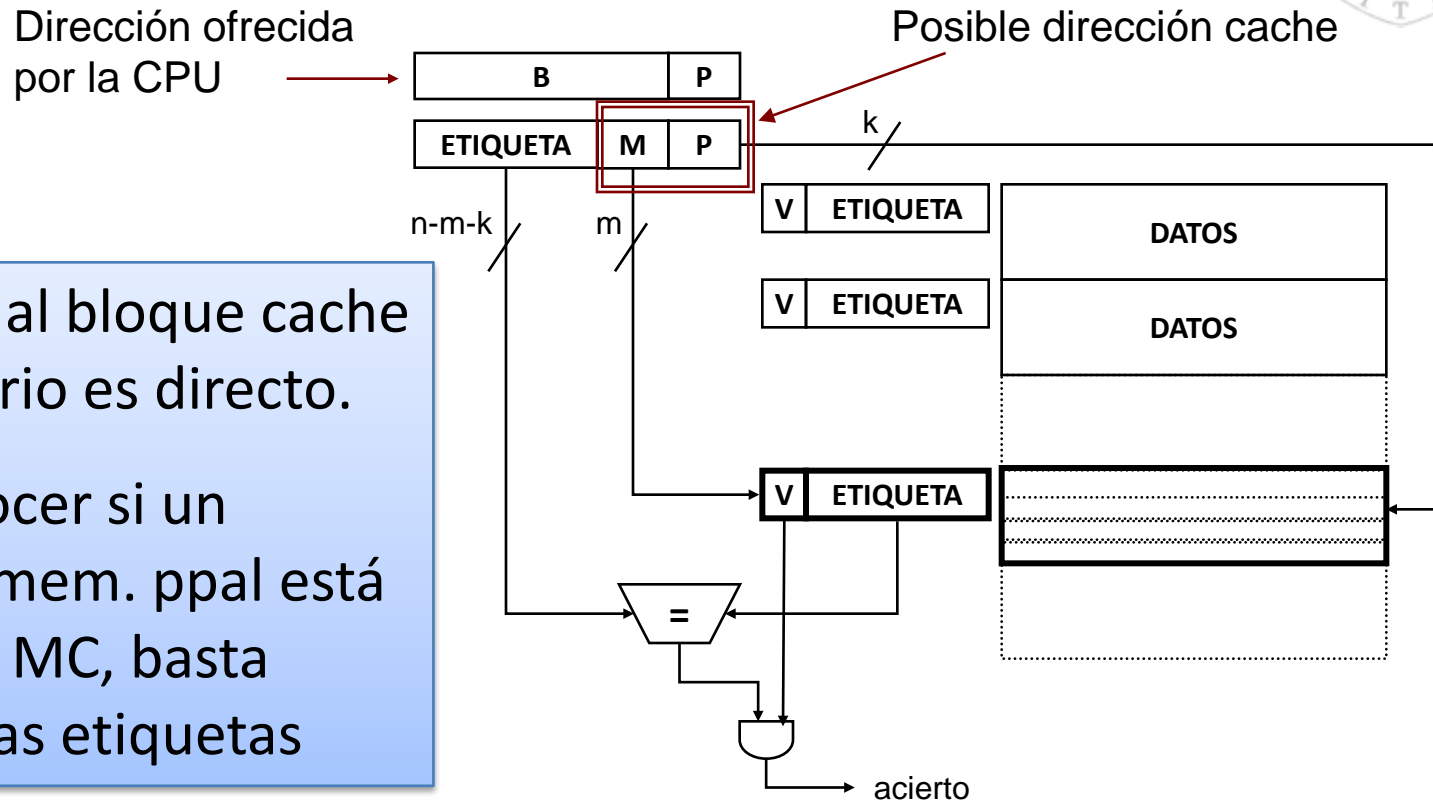
- Para todos los ejemplos:
 - Tamaño de bloque 128 palabras $\Rightarrow k=7$
 - Memoria cache con 8 bloques
 - Memoria principal 4k palabras $\Rightarrow n = 12$





Emplazamiento directo

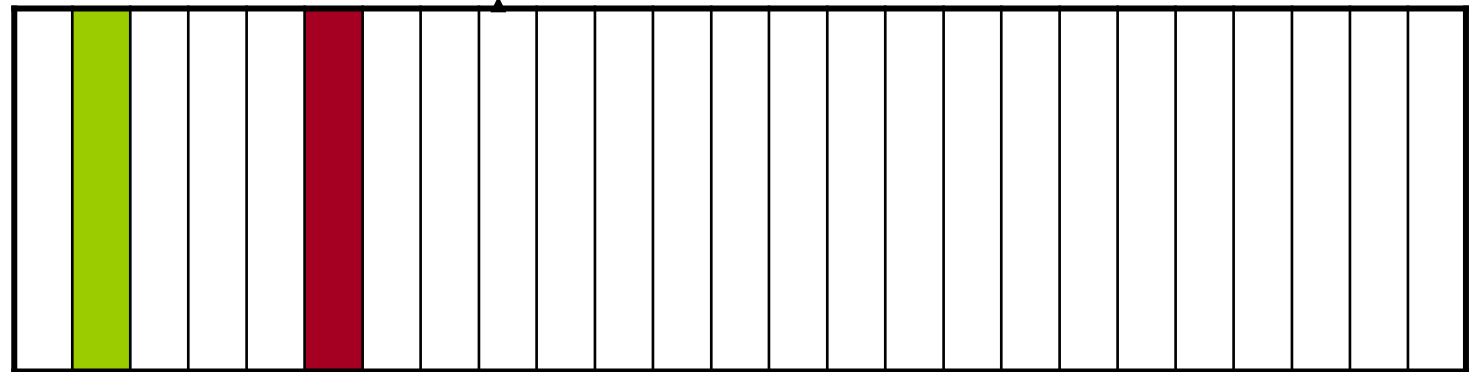
- El acceso al bloque cache y al directorio es directo.
- Para conocer si un bloque de mem. ppal está cargado en MC, basta comparar las etiquetas



¿Cuánto vale m para los datos del ejemplo?

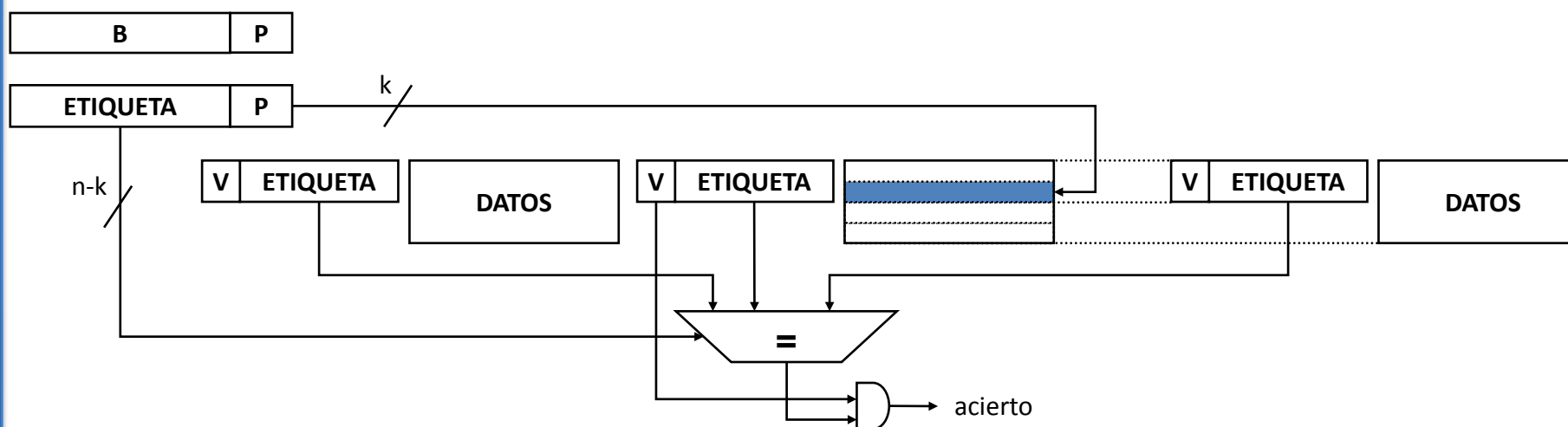


- | N° Bloque
Cache | Cache |
|--------------------|-------|
| 000 | |
| 001 | |
| 010 | |
| 011 | |
| 100 | |
| 101 | |
| 110 | |
| 111 | |



10101

Emplazamiento asociativo

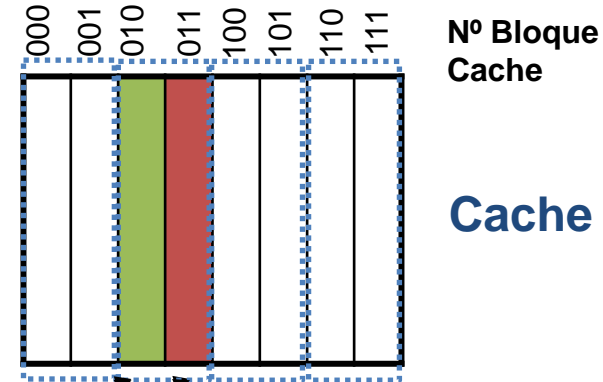


- Para conocer si un bloque de mem. ppal está cargado en MC, hay que comparar todas las etiquetas

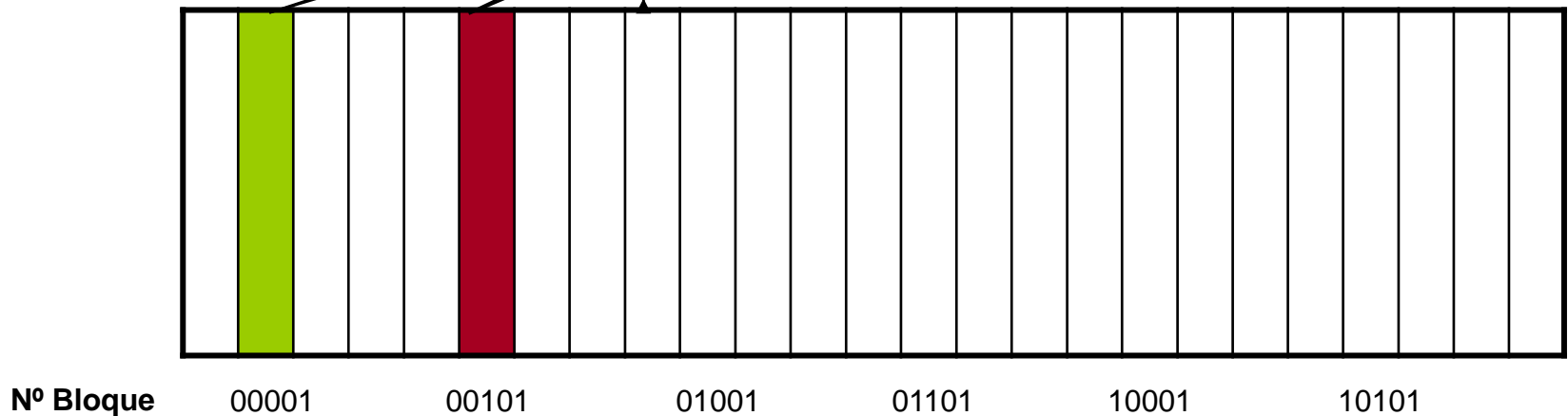
Emplazamiento asociativo por conjuntos



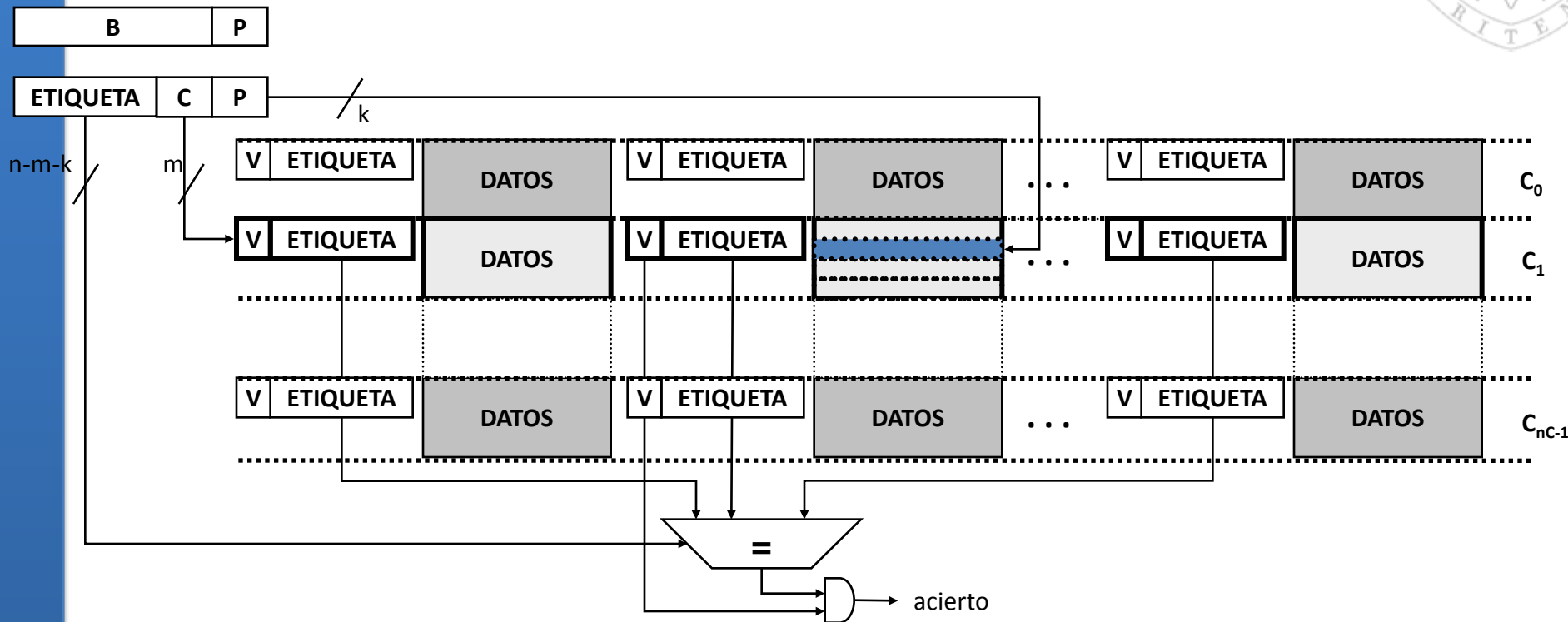
- Cada bloque B tiene asignado un conjunto fijo **C** y puede ubicarse en cualquiera de los bloques de cache que componen dicho conjunto
- Este conjunto viene dado por la expresión: $C = B \bmod nC$



Memoria



Emplazamiento asociativo por conjuntos



- El directorio almacena para cada bloque de cache una etiqueta con los $n-k-m$ bits que completan la dirección del bloque de principal almacenado
- El acceso al conjunto es directo y al bloque de cache dentro del conjunto asociativo

¿Cuánto vale m para los datos del ejemplo?

Emplazamiento asociativo por conjuntos



- El valor nM/nC se denomina grado de asociatividad o número de vías de la MC
 - Grado de asociatividad = 1, equivale a emplazamiento directo
 - Grado de asociatividad = nM , equivale a emplazamiento asociativo

Políticas de emplazamiento



Emplazamineto	Ventajas	Inconvenientes
Directo	Acceso simple y rápido	Alta tasa de fallos cuando varios bloques compiten por el mismo marco
Asociativo	Máximo aprovechamiento de la MC	Una alta asociatividad impacta directamente en el acceso a la MC
Asociativo por conjuntos	Es un enfoque intermedio entre emplazamiento directo y asociativo El grado de asociatividad afecta al rendimiento, al aumentar el grado de asociatividad disminuyen los fallos por competencia por un marco Grado óptimo: entre 2 y 16 Grado más común: 2	Al aumentar el grado de asociatividad aumenta el tiempo de acceso y el coste hardware



Política de actualización

- **¿Qué ocurre cuando se produce una escritura en memoria?**
 - ¿Se escribe sólo en la cache? ¿Sólo en la memoria principal? ¿En las dos?
- **Write-through:** Todas las escrituras actualizan la cache y la memoria
 - Al remplazar, se puede eliminar la copia de cache: Los datos están en la memoria
 - Bit de control en la cache: Sólo un bit de validez
- **Write-back:** Todas las escrituras actualizan sólo la cache
 - Al reemplazar, no se pueden eliminar los datos de la cache: Deben ser escritos primero en la memoria
 - Bit de control: Bit de validez y bit de sucio
- **¿Qué ocurre cuando se produce un fallo de escritura?**
- **Write allocate** (con asignación en escritura): en un fallo de escritura se lleva el bloque escrito a la cache
- **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)



Políticas de actualización

- Comparación:
 - Write-through:
 - Memoria (y otros lectores) siempre tienen el último valor
 - Control simple
 - Write-back:
 - Mucho menor ancho de banda, escrituras múltiples en bloque
 - Mejor tolerancia a la alta latencia de la memoria

Políticas de reemplazamiento



- **¿Qué ocurre si la MC está llena? ¿Qué bloque de cache se debe desalojar?**
- Espacio de reemplazamiento: conjunto de posibles bloques que pueden ser reemplazados por el nuevo bloque
 - **Directo**: el bloque que reside en marco que el nuevo bloque tiene asignado. Al no existir alternativas no se requieren algoritmos de re-emplazamiento
 - **Asociativo**: cualquier bloque que resida en de la cache
 - **Asociativo por conjuntos**: cualquier bloque que resida en el conjunto que el nuevo bloque tiene asignado
- Algoritmos (implementados en hardware):
 - **Aleatorio**: se escoge un bloque del espacio de re-emplazamiento al azar
 - **FIFO**: se sustituye el bloque del espacio de re-emplazamiento que lleve más tiempo cargado
 - **LRU** (least recently used): se sustituye el bloque del espacio de re-emplazamiento que lleve más tiempo sin haber sido referenciado
 - **LFU** (least frequently used): se sustituye el bloque del espacio de re-emplazamiento que haya sido referenciado en menos ocasiones



Rendimiento

■ Procesador con memoria perfecta (ideal)

- $T_{cpu} = N \times CPI \times T_c$
- Como $N \times CPI = N^{\circ}$ ciclos CPU $\rightarrow T_{cpu} = N^{\circ}$ ciclos CPU $\times T_c$

■ Procesador con memoria real

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

— Cuántos ciclos de espera por la memoria?

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

— Luego:

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

— Y finalmente

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

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



Define el espacio de diseño para la optimización de M_c



Rendimiento

■ Penalización media por instrucción

— 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:

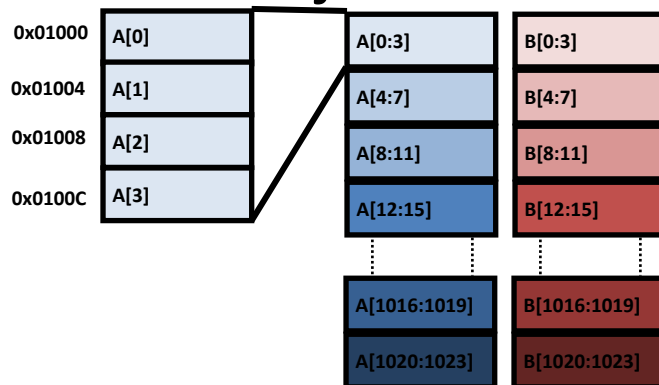
- Penalización media por instrucción =
= $AMPI \times \text{Miss Rate} \times \text{Miss Penalty}$

■ Tiempo medio de acceso a memoria (TMAM)

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

Ejemplo

- Se tienen dos arrays de 1024 palabras, se accede a los arrays para ejecutar el siguiente código: $\text{valor} = A[i] + B[i]$
- Se tiene una memoria cache de 128B con bloques de 16B
- A está alojado en la dirección: 0x01000
- B está alojado en la dirección: 0x04000



- Acceso directo
- Asociativo con 2 vías (LRU)
- Asociativo con 4 vías (FIFO)
- Completamente asociativa



¿Existe la cache perfecta?

■ Las 3 C's

– 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.

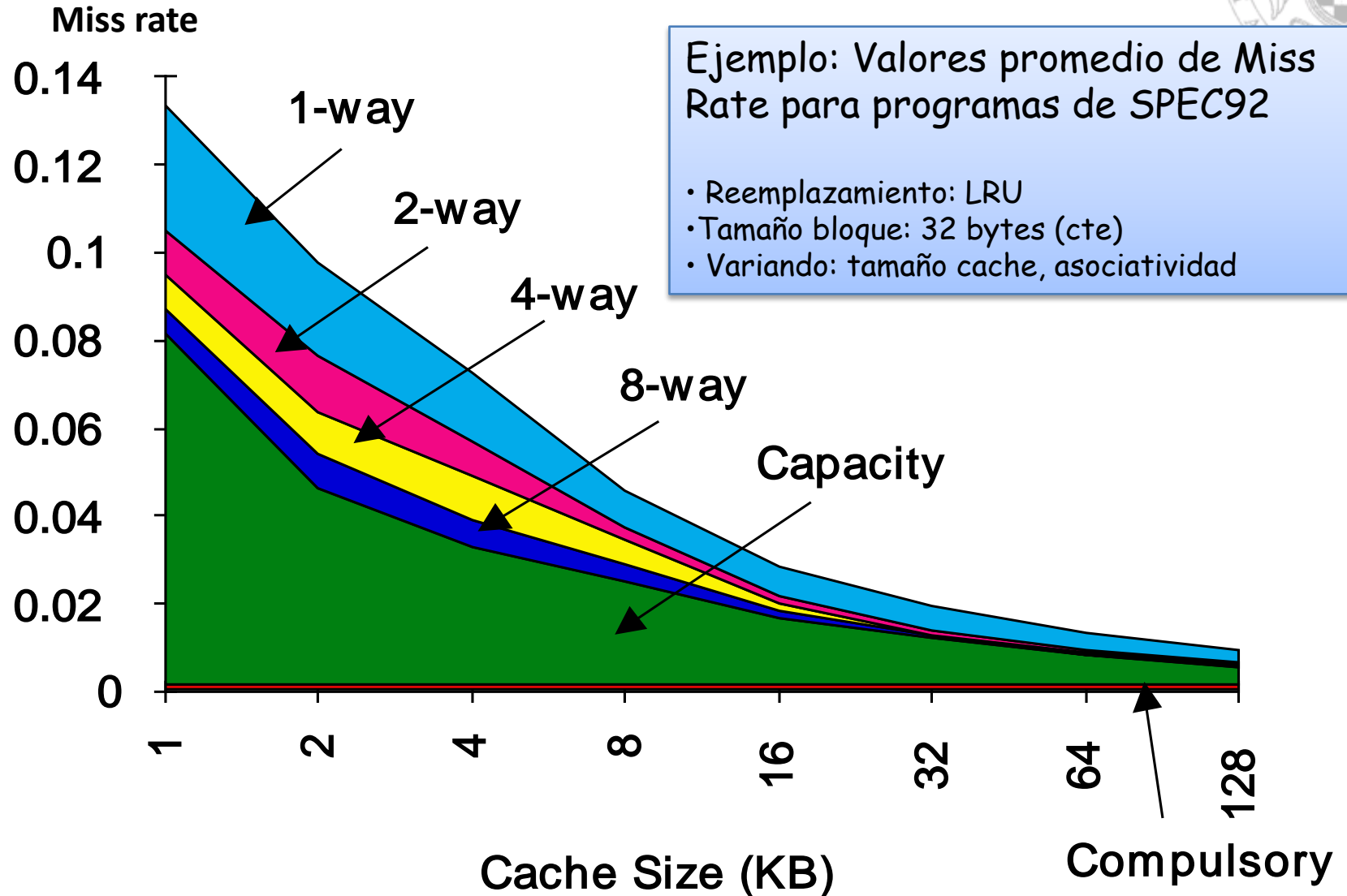
– Capacidad

- 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

– 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.

Políticas de emplazamiento



¿Cómo 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:
 - Reducir la tasa de fallos
 - Reducir la penalización del fallo
 - Reducir el tiempo de acierto (hit time)
 - Aumentar el ancho banda

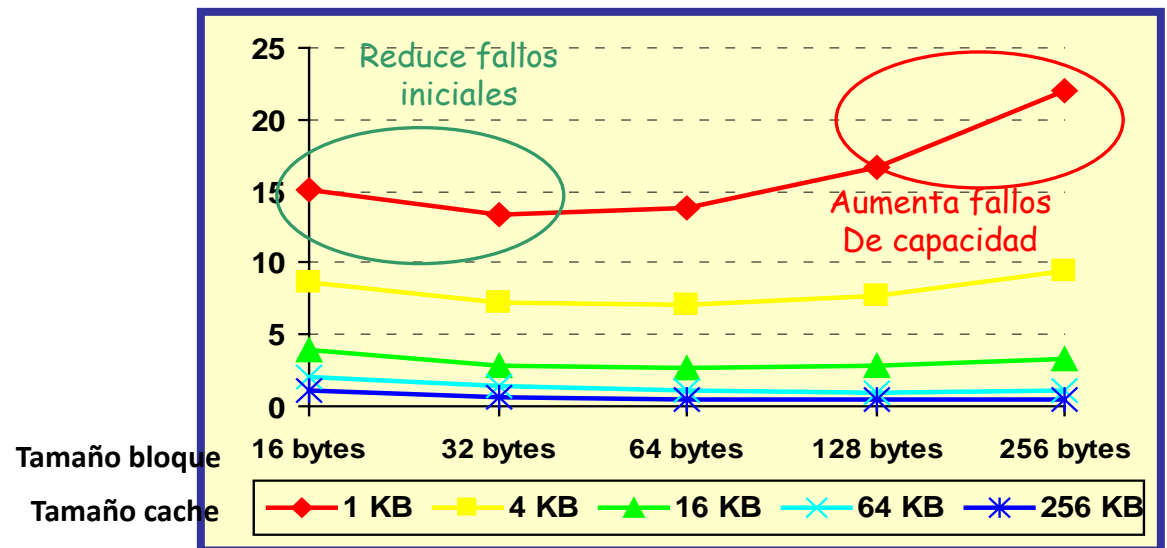
¿Cómo mejorar el rendimiento de la cache?



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	

Tamaño de bloque

- Aumentar el tamaño del bloque disminuye la tasa de fallos iniciales y **captura mejor la localidad espacial**
- Pero se aumenta la tasa de fallos de capacidad (menor N° Bloques => **captura peor localidad temporal**)
- Al tener bloques más grandes aumenta el **miss-rate**



¿Cómo mejorar el rendimiento de la cache?



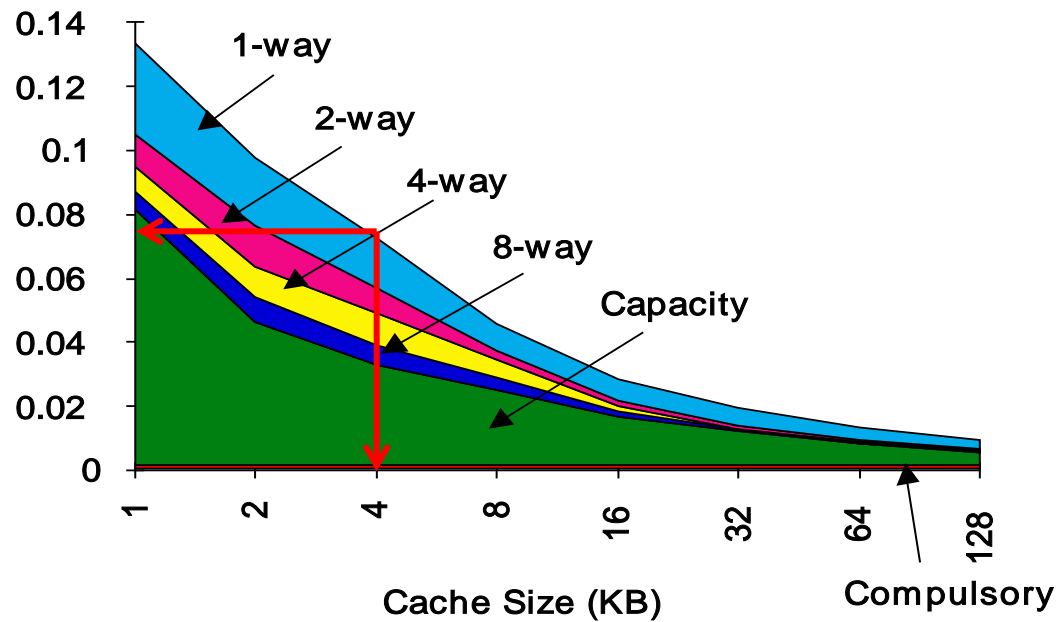
Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Asociatividad	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV =>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



Asociatividad

- El aumento de la asociatividad disminuye la tasa de fallos
- El aumento de la asociatividad aumenta el hardware y el tiempo de acceso
- Del estudio de la gráfica
 - El paso de emplazamiento directo a asociativa con 2 conjuntos permite disminuir el tamaño de la cache a la mitad
 - Manteniendo el tamaño de la cache, no se consiguen grandes mejoras con más de 8 vías

Asociatividad



¿Cómo 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	

Algoritmo de reemplazamiento



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

Fallos de datos por 1000 instrucciones

- La política de reemplazamiento aleatoria necesita menos HW y es más rápida que el LRU, pero produce una mayor tasa de fallos

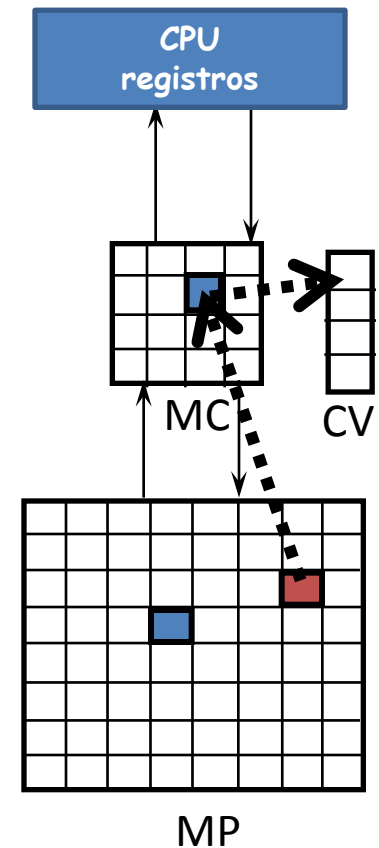
¿Cómo mejorar el rendimiento de la cache?



- También se puede **reducir la tasa de fallos**:
 - Cache víctima
 - Optimizando el código
 - Pre-búsqueda HW
 - Pre-búsqueda SW

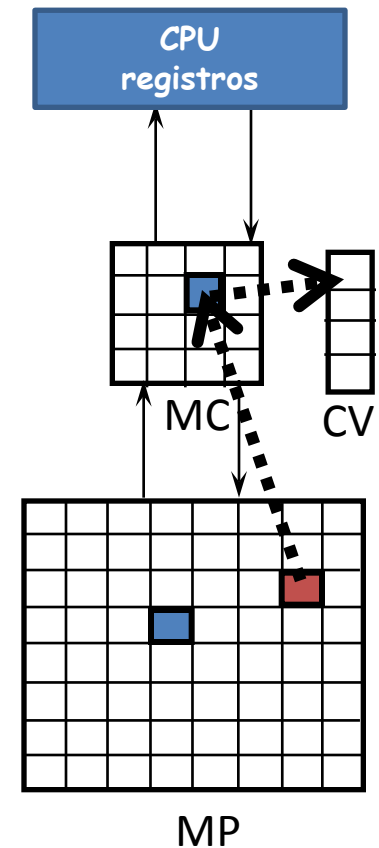
Cache Víctima

- Se propuso por primera vez en 1990, la idea es utilizar una cache pequeña completamente asociativa (CV) conjuntamente con una cache grande con una asociatividad más limitada (MC)
- Se intenta conseguir que MC tenga la misma tasa de fallos que si fuera completamente asociativa
 - Ej: MC de acceso directo + CV asociativa tiene la misma tasa de fallos que MC con 2 vías



Cache Víctima

- Un bloque puede estar en MC o en CV pero nunca en las dos
- En la cache víctima estarán aquellos bloques que han sido re-emplazados de la memoria cache
 - Un bloque pasa de MC a CV cuando se produce un re-emplazo
 - Un bloque pasa de CV a MC cuando es referenciado





Optimización de Código

- Para todos los ejemplos:
 - MC de 4 Kbytes
 - Acceso directo
 - 256 bloques
 - 1 bloque = 16 bytes
 - Datos:
 - `int A[1024];`
 - `int B[1024];`
 - ¿Cuántas posiciones del array por bloque?
 - ¿Cuántas posiciones del array por MC?

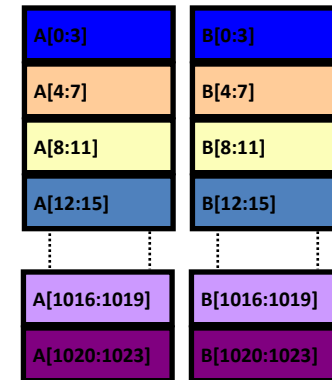
Optimización de Código

- Código original:

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

Si MC tuviera 2 vías,
¿se reducirían los fallos?

2x1024 accesos
2x1024 fallos
 2x256 de inicio
 Resto 2x3x256

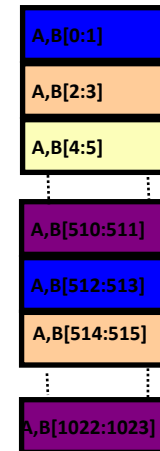


- **Fusión de arrays:** Mejora la localidad espacial para disminuir los fallos

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

```
struct fusion{
    int A;
    int 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

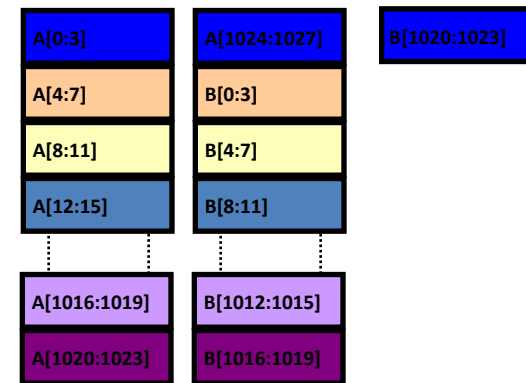


Ganancia: 4

Optimización de Código

- **Alargamiento de arrays:** Mejora la localidad espacial para disminuir los fallos
 - Impedir que en cada iteración del bucle se compita por el mismo marco de bloque

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



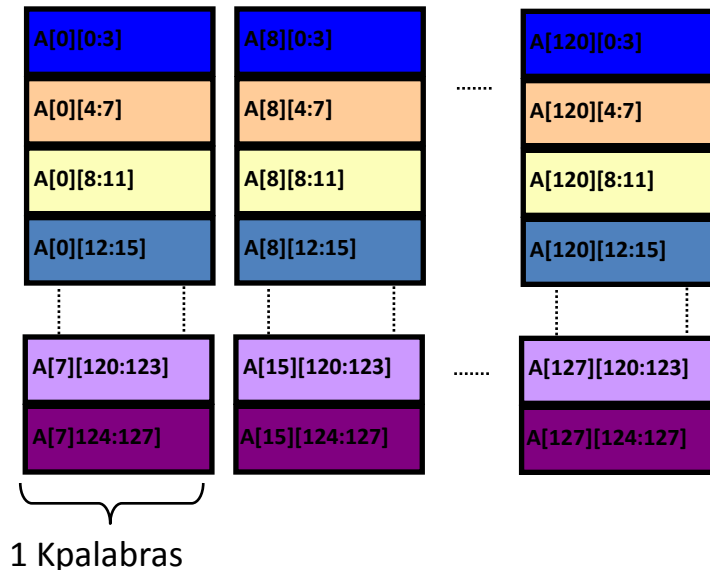
Si MC tuviera 2 vías,
¿se reducirían los fallos?

1024/2 fallos
2x256 de inicio

Ganancia: 4

Optimización de Código

- **Intercambio de bucles:** Mejora la localidad espacial para disminuir los fallos
 - En lenguaje C las matrices se almacenan por filas, luego se debe variar en el bucle interno la columna (`int A[128][128];`)



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



Optimización de Código

■ Intercambio de bucles:

– Acceso por filas

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

128x128 accesos
128x128 fallos
16x256 de inicio
Resto (12288)

Si MC tuviera 2 vías,
¿se reducirían los fallos?

– Acceso por columnas

```
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



Optimización de Código

- Fusión de bucles:
 - Mejora la localidad temporal para disminuir los fallos
 - Fusionar los bucles que usen los mismos arrays para usar los datos que se encuentran en cache antes de desecharlos

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

128x128x2 accesos
(128x128/4)x2 fallos

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

(128x128/4)x2 fallos

Ganancia: 4

¿Y si aplicamos fusión de bucles e intercambio de bucles?

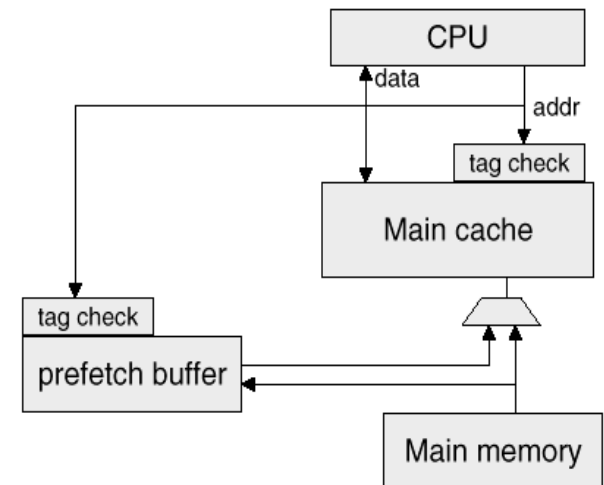


Cache con Pre-búsqueda

- Anticipa los fallos de Cache anticipando las búsquedas antes de que el procesador demande el dato o la instrucción que provocarían un fallo
 - Esta técnica reduce la tasa de fallos, pero principalmente la penalización por fallo
 - Este tipo de cache sólo es efectiva con un adecuado ancho de banda ya que se van a producir más transferencias de bloques de las necesarias
 - Esta técnica es más efectiva si se dispone de una memoria cache y de un buffer, en caso contrario los bloques pre-buscados (que puede que no se usen) están desalojando de MC bloques activos

Cache con Pre-búsqueda

- Pre-búsqueda Hardware:
 - Típicamente: la CPU busca dos bloques en un fallo (el referenciado y el siguiente)
 - El bloque buscado se lleva a Mc
 - El prebuscado se lleva a un buffer (“prefetch buffer” o “stream buffer”). Al ser referenciado pasa a MC





Cache con Pre-búsqueda

- Pre-búsqueda Software:
 - Instrucciones especiales de pre-búsqueda introducidas por el compilador
 - La eficiencia depende del compilador y del tipo de programa
 - Pre-búsqueda con destino en cache (MIPS IV, PowerPC, SPARC v. 9), o en registro (HP-PA)
 - Instrucciones de pre-búsqueda no producen excepciones. Es una forma de especulación.
 - Funciona bien con bucles y patrones simples de acceso a arrays. 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

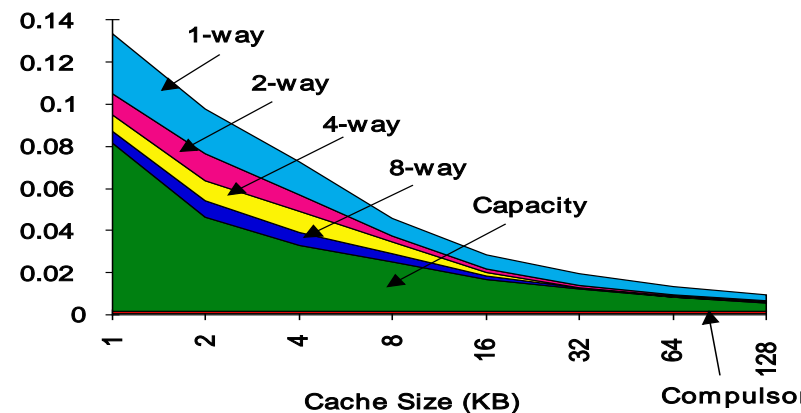
¿Cómo mejorar el rendimiento de la cache?



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	

Caches Multinivel

- El aumento del tamaño de MC reduce la tasa de fallos
- El aumento del tamaño de la MC aumenta la penalización por fallo
- El aumento del tamaño de MC aumenta el coste del procesador
- Si se quiere una MC rápida y barata, ésta tiene que ser pequeña
- Teniendo diferentes niveles de MC se puede conseguir reducir la penalización por fallo
 - No es lo mismo transferir de MP a MC que de L1 a L2
 - Tener L2 permite aplicar políticas de pre-búsqueda con menos inconvenientes





Cache Multinivel

■ Cache multinivel (L2, L3, ...)

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

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

- Tiempo medio de acceso a memoria: Dos niveles

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

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

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

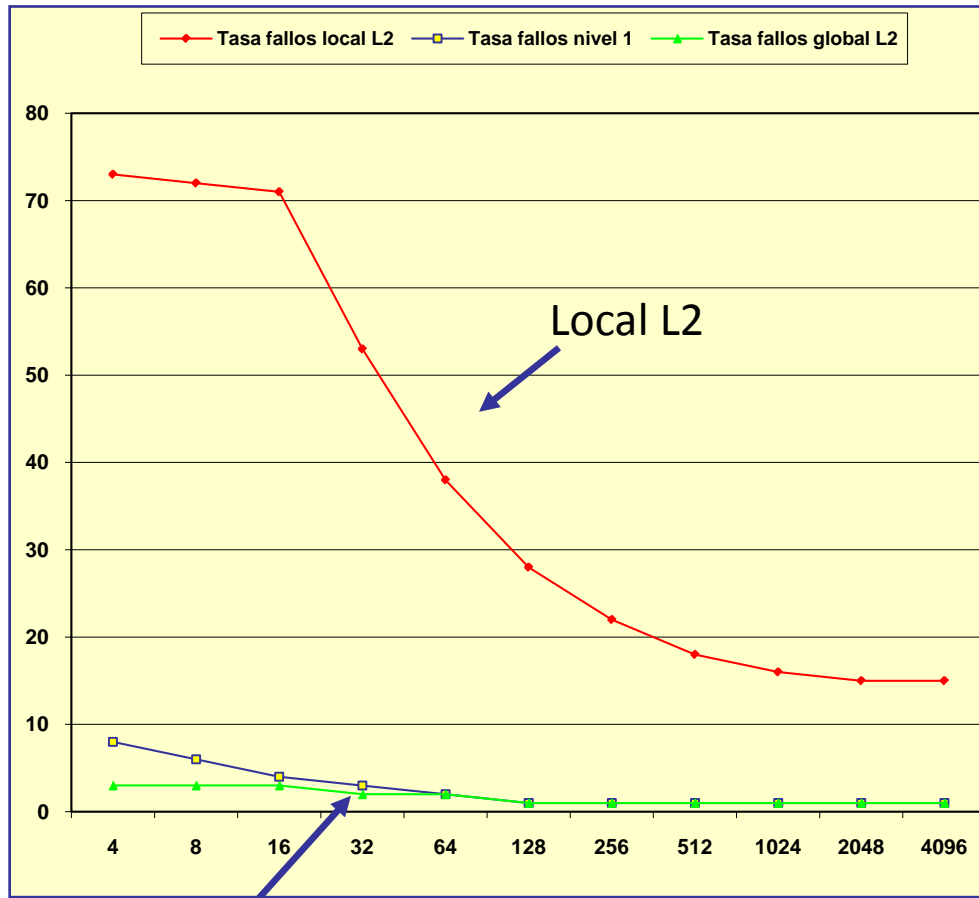
- Definiciones:

- Tasa de fallos local en una cache (Lx): fallos en cache Lx dividido por el número total de accesos a la cache Lx
 - Tasa de fallos global en una cache (Lx): fallos en cache Lx dividido por el número 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 \Rightarrow Acceder a un dato en el ciclo del procesador
 - L2: Afecta a la penalización de L1 \Rightarrow Reducción del tiempo medio de acceso

Cache Multinivel



global

¿Cómo 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	

Reducción penalización por fallo



- Dar prioridad a las lecturas sobre las escrituras
 - Un fallo de lectura puede impedir la continuación de la ejecución del programa; un fallo de escritura puede ocultarse
 - 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
 - PROBLEMA: podemos estar intentando leer una palabra que todavía está en el buffer
- Dar prioridad a palabras críticas
 - Cuando la palabra solicitada se carga en memoria cache se envía al procesador, sin esperar a la carga del bloque completo
 - Problema. Localidad espacial: alta probabilidad de acceder a continuación a la siguiente palabra en secuencia.

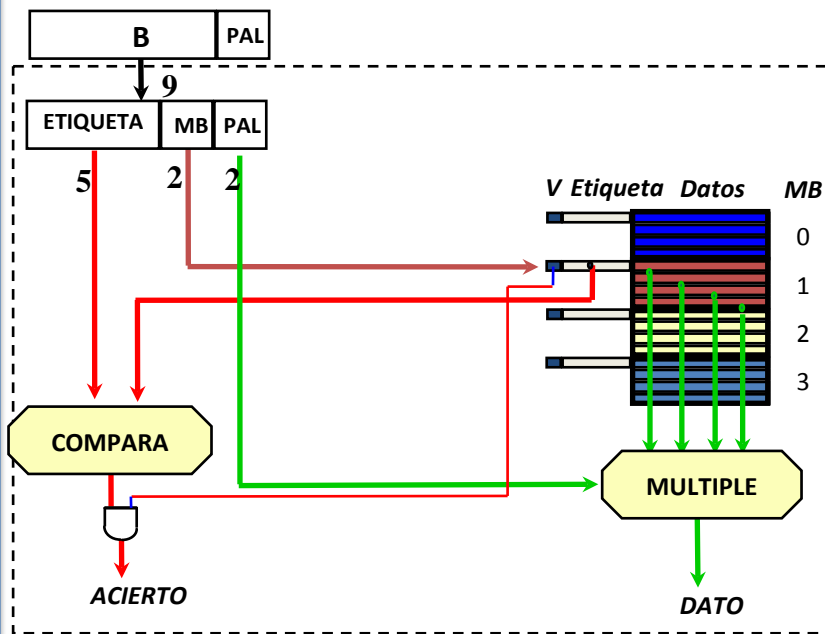
¿Cómo mejorar el rendimiento de la cache?



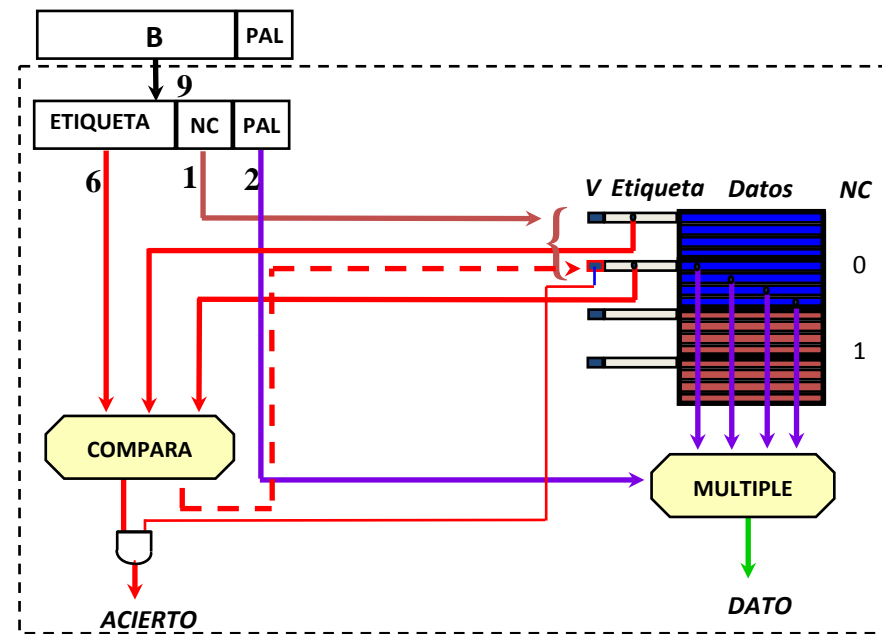
Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Asociatividad	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => 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 L1 pequeña y sencilla

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



Directo



Asociativo por conjuntos 2 vías

¿Cómo 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 Segmentada

- Cache segmentada
 - Segmentar los accesos a la cache permite aumentar el ancho de banda.
 - 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
 - Más problemas: Mayor penalización en los saltos mal predichos
 - Ejemplos: N^o 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



Cache 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
Cache Víctima	+	-			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.

Cache 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
Cache multinivel		+			2	Ampliamente usado. Más complejo si tamaño de bloque en L1 y L2 distintos.
Cache pequeña y sencilla	—		+		0	Trivial; ampliamente usado.
Cache multibanco				+	1	En L2 de Opteron y Niagara
Cache segmentada			—	+	1	Ampliamente usado