

Jerarquía de Memoria

Estructura de Computadores. Tema 6.1.

LA ABSTRACCIÓN DE MEMORIA

Escritura

- Transferir datos de CPU a memoria `movq %rax, 8(%rsp)`
- Operación “**Store**”

Lectura

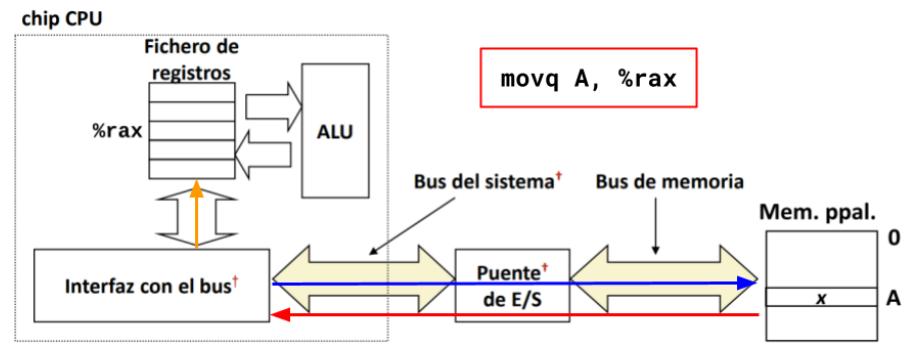
- Transferir datos de memoria a CPU `movq 8(%rsp), %rax`
- Operación “**Load**”

Estructura de buses CPU – Memoria

Un **bus** es un conjunto de cables en paralelo que transportan direcciones, datos, y señales de control. Típicamente a un bus se conectan varios dispositivos.

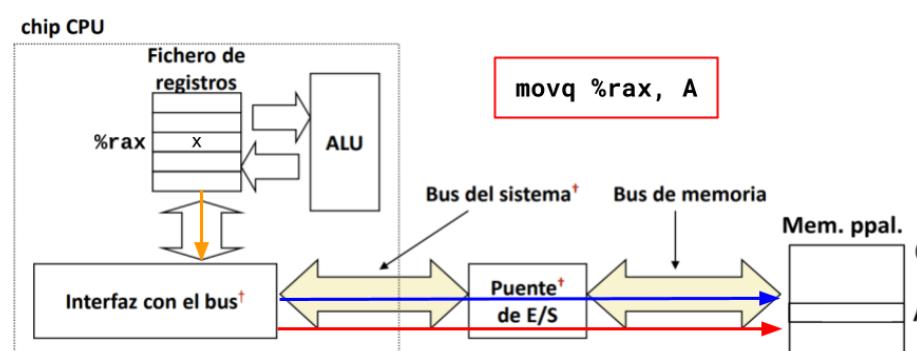
Transacción de Lectura de Memoria

1. La CPU pone la dirección A en el bus de memoria.
2. La memoria principal recibe dirección A del bus de memoria, recupera la palabra x , y la pone en el bus.
3. La CPU lee palabra x del bus y la copia al registro $%rax$



Transacción de Escritura a Memoria

1. La CPU pone dirección A en el bus. La memoria principal la recibe y espera a que llegue la palabra de datos correspondiente.
2. La CPU pone la palabra de datos x en el bus
3. La memoria principal recibe la palabra de datos x del bus y la almacena en la posición con dirección A



RAM: BLOQUE CONSTRUCTIVO DE MEMORIA PRINCIPAL

Conceptos

Tiempo de acceso. Tiempo que se requiere para leer (o escribir) un dato (palabra) en la memoria, medido en ciclos o ($n\cdot\mu\cdot m$) s.

Ancho de banda (de la memoria de un computador). Número de palabras a las que puede acceder el procesador (o que se pueden transferir entre el procesador y la memoria) por unidad de tiempo, medido en (K-M-G) B/s.

Métodos de acceso

- Aleatorio (RAM): tiempo de acceso independiente de la posición a acceder. Por ejemplo, SRAM, ROM

- Secuencial (SAM): tiempo de acceso depende de la posición de los datos a acceder. Por ejemplo, cinta magnética
- Direto (semialeatorio, DASD – direct access storage device): tiempo acceso tiene una componente aleatoria y otra secuencial. Por ejemplo, discos giratorios

Memoria de Acceso Aleatorio (RAM)

Características principales

- La RAM tradicionalmente se empaqueta **como un chip** o está **incluida** (empotrada) como parte de un chip **procesador**
- La unidad básica almacenamiento es normalmente una **celda** (1 bit/celda)
- Múltiples chips de RAM forman una memoria

La RAM tiene dos variedades

- **SRAM** (RAM estática)
 - 6 transistores / bit
 - 2 inversores ($\times 2$ tr) + 2 puertas de paso
 - Mantiene el estado indefinidamente
- **DRAM** (RAM Dinámica)
 - (1 Transistor + 1 condensador) / bit
 - Condensador orientado verticalmente
 - Debe refrescar estado periódicamente

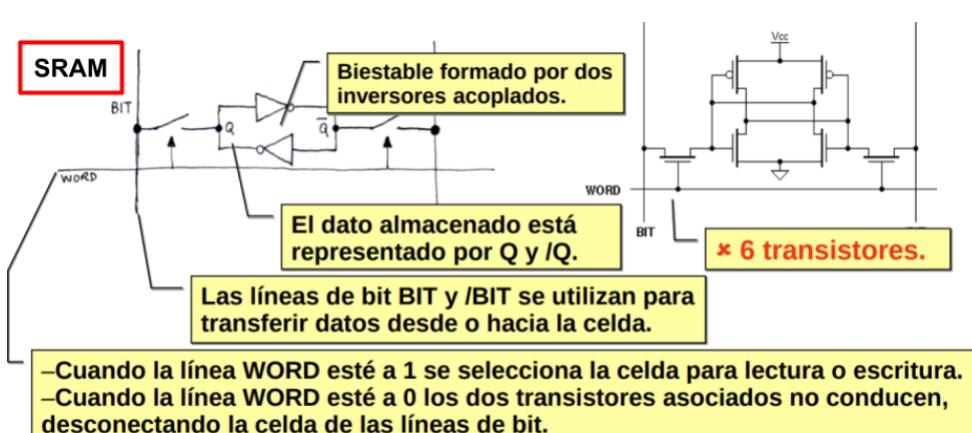
SRAM

Celda de memoria SRAM

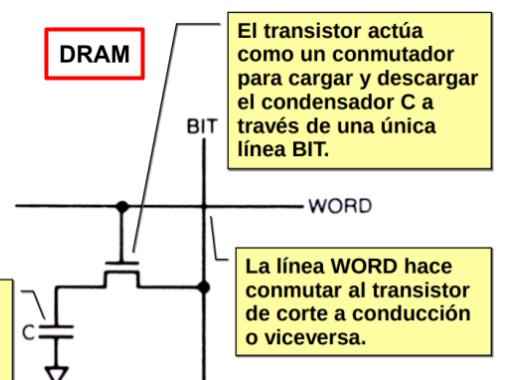
Son **estáticas** ⇒ los datos almacenados se mantienen por un tiempo indefinido si hay alimentación.

Las operaciones de **lectura** son no destructivas y tienen mayor velocidad que las DRAM.

- Se selecciona la celda poniendo WORD a 1 ⇒ Q se conecta a BIT y /Q a /BIT.



- Si $Q = 1$ ($/Q = 0$) ⇒ la línea /BIT se pone a 0. – BIT = 1 y /BIT = 0 ⇒ se lee un 1.
- Si $Q = 0$ ($/Q = 1$) ⇒ la línea BIT se pone a 0. – BIT = 0 y /BIT = 1 ⇒ se lee un 0.



DRAM

Celda de memoria DRAM

Son **dinámicas** ⇒ los datos almacenados decaen o se desvanecen y deben ser restaurados a intervalos regulares.

La celda está construida sobre el condensador para minimizar espacio

La **lectura** es destructiva, esto es, debe ir seguida de una escritura que restaure el estado original. Un circuito se encarga de refrescar y mantener los estados.

- La circuitería externa convierte a BIT en una línea de salida, poniendo la celda con WORD = 1.
- Si C está cargado (= 1) ⇒ se descarga a través de la línea BIT ⇒ se produce un pulso de corriente que es detectado por un amplificador de salida (“sense amplifier”) ⇒ aparece un 1 en la línea de datos de salida.
- Si C está descargado (= 0) ⇒ no se produce pulso de corriente ⇒ aparece un 0 en la línea de datos de salida.

Circuitos de memoria DRAM

Capacidad elevada de los chips de memoria DRAM ⇒ las direcciones han de proporcionarse multiplexadas en el tiempo.

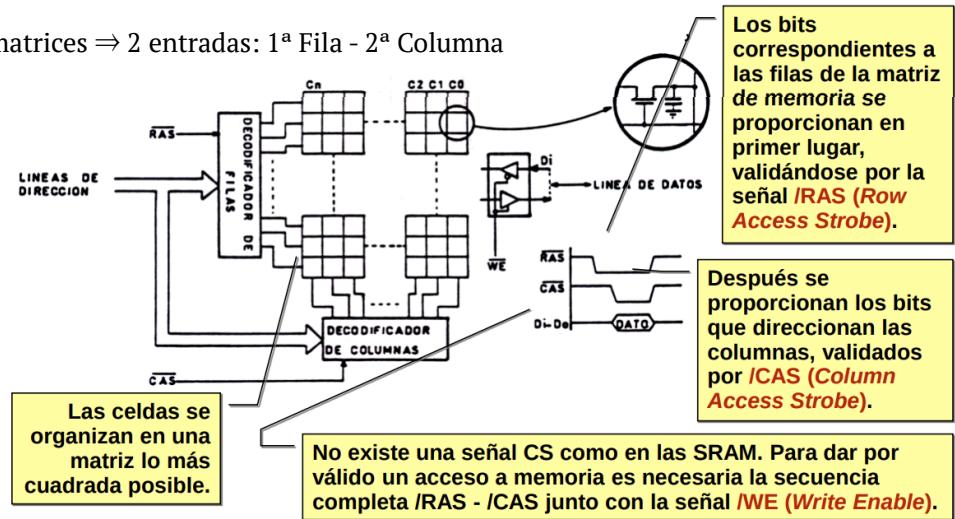
Problema. Para una memoria de 4 MB ($2^{22} \cdot 2^{20}$ B) necesitamos 22 “patillas” ⇒ ¡Muchas patillas para una memoria tan pequeña!

Solución. Organizamos la memoria por matrices ⇒ 2 entradas: 1^a Fila - 2^a Columna

Parámetros importantes

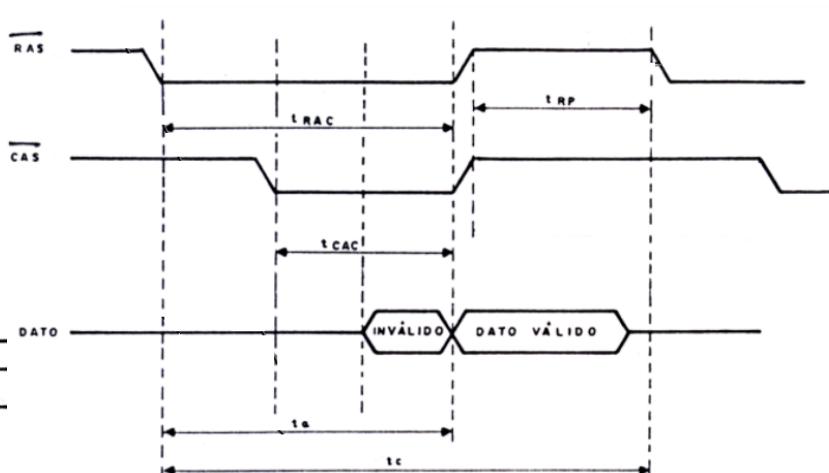
$t_{RAC} = t_a$ (tiempo de acceso):
tiempo desde que se inicia una lectura (/RAS↓) hasta que el dato está disponible en el bus de datos.

t_c (tiempo de ciclo):
tiempo mínimo entre dos accesos consecutivos.
 $t_c \cong t_{RAC} + t_{RP}$
 $t_c > t_a$

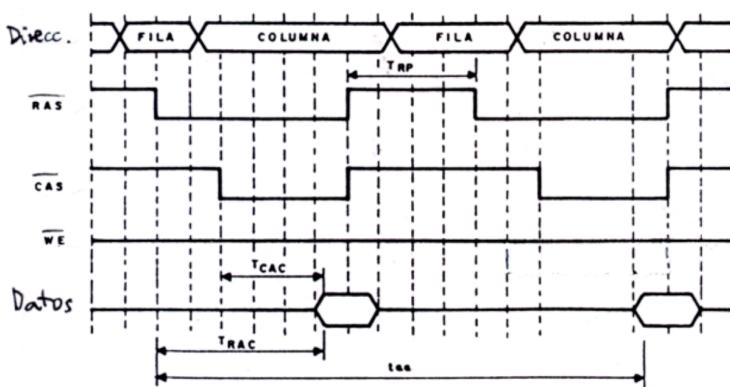


No se puede comenzar un segundo acceso tras t_a , dado que las lecturas son destructivas (el condensador de la celda de memoria se descarga) y es necesario esperar t_{RP} (tiempo de precarga de fila) para que la circuitería interna de la DRAM restaure el valor de las celdas de la fila correspondiente.

Acceso a una palabra



Acceso a dos palabras



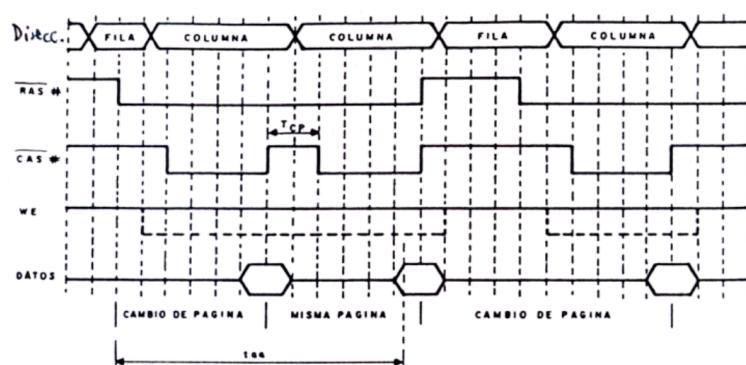
Fast Page Mode (acceso en modo página rápido)

Cuando se va a acceder a la misma fila, pueden haber dos datos columna seguidos

t_{aa} (tiempo de acceso a dos palabras en modo página):
 $t_{aa} \cong t_{RAC} + t_{CP} + t_{CAC}$



t_{aa} (tiempo de acceso a las dos palabras):
 $t_{aa} \cong t_{RAC} + t_{RP} + t_{RAC}$



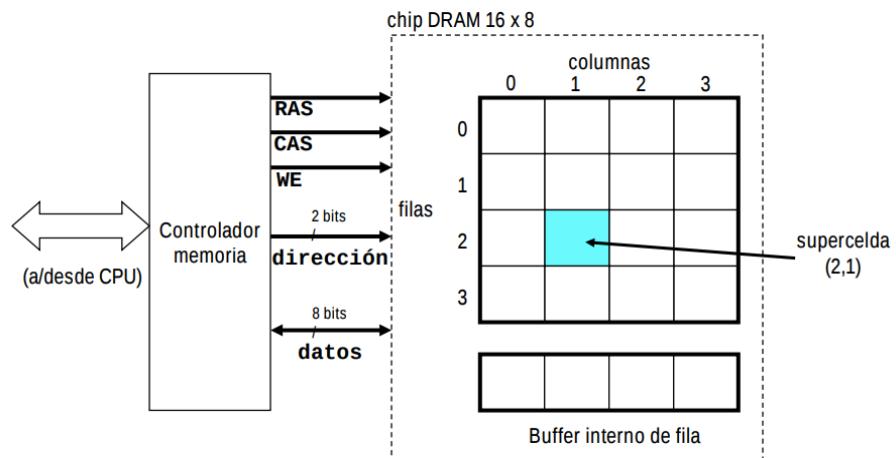
Organización DRAM convencional

- DRAM $d \times w$: $d \cdot w$ bits total organizados como d superceldas de tamaño w bits

Lectura de Supercelda DRAM (2,1)

Paso 1

- RAS (Row Address Strobe, comando Activate) selecciona fila 2
- Fila 2 copiada de array DRAM a buffer de fila



Paso 2

- CAS (Column Address Strobe, comando Read) selecciona columna 1
- Supercelda (2, 1) copiada de buffer a líneas bus datos → CPU

Paso 3. Buffer copiado de vuelta a la fila para refrescar datos

Resumen SRAM vs DRAM

	Transistores por bit	Tiempo de acceso	¿Necesita refresco?	¿Necesita ECD (detección y corrección de errores)?	Coste	Aplicaciones
SRAM	6 ú 8	1x	No	Quizás	100x	Memoria Caché
DRAM	1	10x	Sí	Sí	1x	Memoria Principal, Frame Buffers

Tendencias

- La **SRAM escala** con la tecnología de semiconductores
 - está llegando a sus límites
- Escalado **DRAM limitado** por mínima capacidad necesaria C (condensador)
 - razón de aspecto limita cómo de profundo se puede hacer el C
 - también llegando a su límite

DRAMs mejoradas

- Funcionamiento** celda DRAM **no ha cambiado** desde su invención
 - Comercializada por Intel en 1970
- Núcleos DRAM con **mejor** lógica de **interfaz** y E/S más rápida:
 - DRAM síncrona (SDRAM)
 - Usa una señal de reloj convencional en lugar de control asíncrono
 - puede aceptar un comando y transferir una palabra en cada ciclo reloj
 - puede hacer pipeline accediendo concurrentemente a distintos bancos
 - Reinterpreta señales RAS/CAS/WE/A10 como comando (Activate, Read...)‡
 - Añade 1..3 bits selección banco (BA0-2) (2,4,8 "Memory Arrays")‡
 - DRAM síncrona a doble velocidad datos (double data-rate, DDR SDRAM)
 - Temporización a doble flanco envía 2 bits por ciclo por pin
 - Diferentes tipos según el tamaño de un pequeño buffer precaptación:

- DDR (2 bits), DDR2 (4 bits), DDR3 (8 bits), DDR4 (8† bits)
- DDR4 cambia formato comandos (ACT, RAS/CAS/WE=A16-14, BC/AP=A12,10)
- Para 2010, estándar para mayoría sistemas sobremesa y servidor
- Intel Core i7 admite SDRAM DDR3 y/o DDR4 (según modelo)

Módulos de Memoria

CONFIGURACIÓN Y DISEÑO DE MEMORIAS UTILIZANDO VARIOS CHIPS

Ampliación de memoria

Presenta un principal problema:

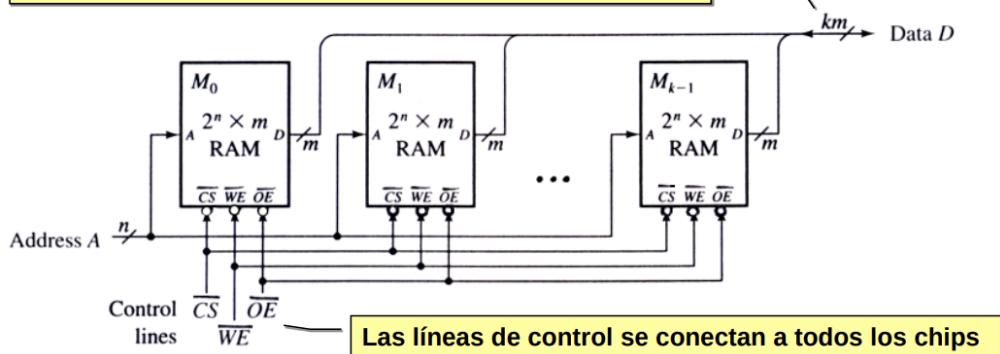
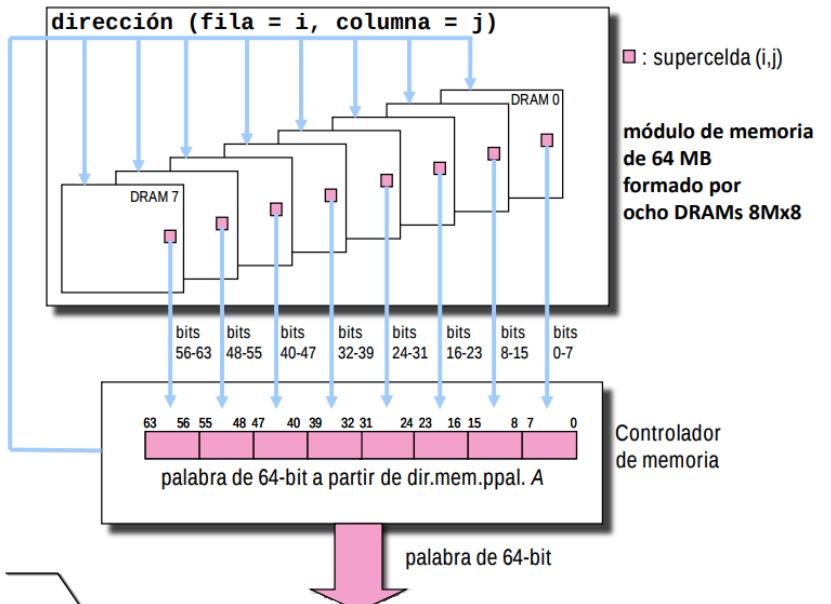
Construir una memoria de 2^M palabras de M bits a partir de chips de 2^n palabras de m bits.

Casos:

1. Incrementar el ancho de palabra de m a $k \cdot m = M$

Necesitamos k circuitos de tamaño de palabra m:

Los k chips se conectan de forma “bit-slice”. Cada chip contribuye a m bits de la palabra de datos de $k \cdot m$ bits



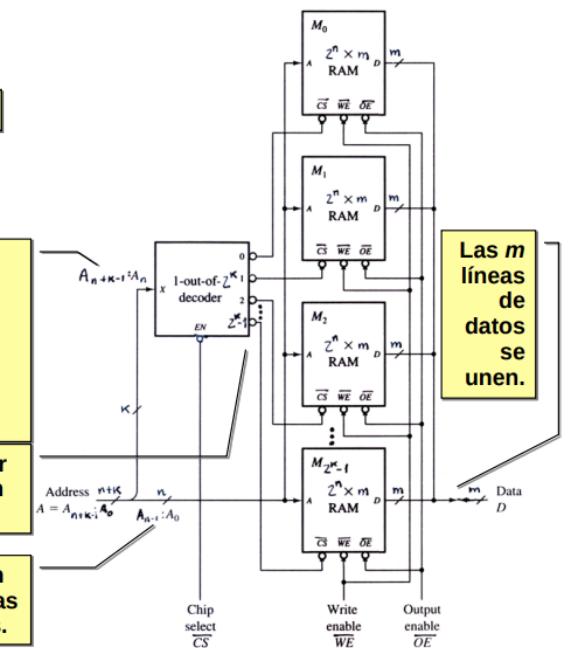
3. Incrementar simultáneamente el número de palabras y el ancho de palabra → Basta combinar las dos técnicas anteriores.

Las k líneas de dirección de orden superior ($A_{n+k-1}:A_n$) se conectan a las entradas de un decodificador de k a 2^k ⇒ cada configuración de los $n+k$ bits hace que se seleccione solamente el circuito RAM indicado por los bits de dirección $A_{n+k-1}:A_n$.

Las 2^k líneas de salida del decodificador se conectan a las entradas de selección /CS de los 2^k circuitos.

Las n líneas de dirección de orden inferior se conectan en común a las líneas de dirección de los 2^k chips.

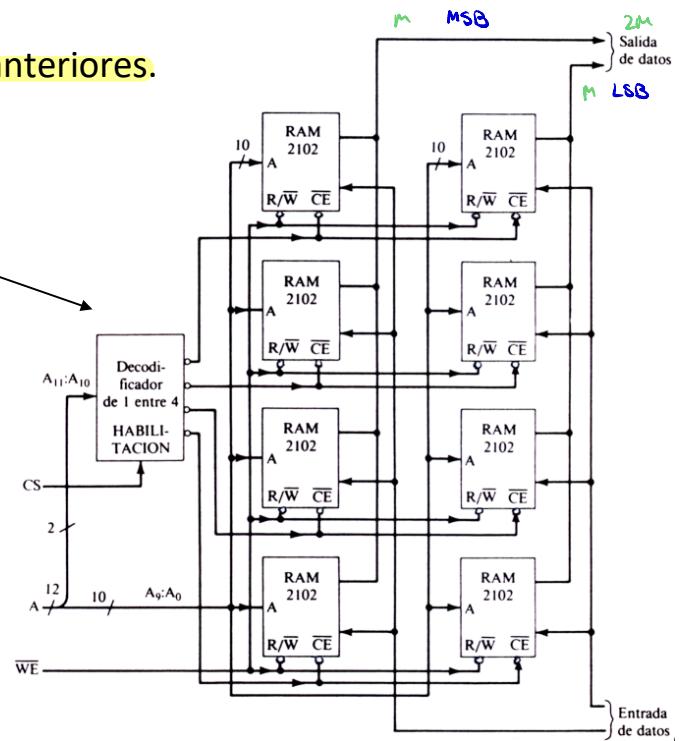
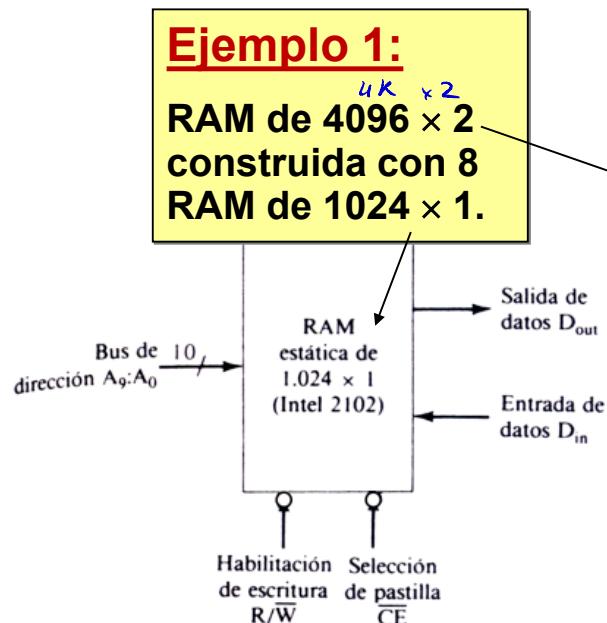
2. Incrementar el número de palabras de 2^n a $2^{n+k}=2^N$ → Necesitamos 2^k circuitos de 2^n palabras y un decodificador de 1 entre 2^k .



Ampliación de memoria

③ Incrementar simultáneamente el número de palabras y el ancho de palabra.

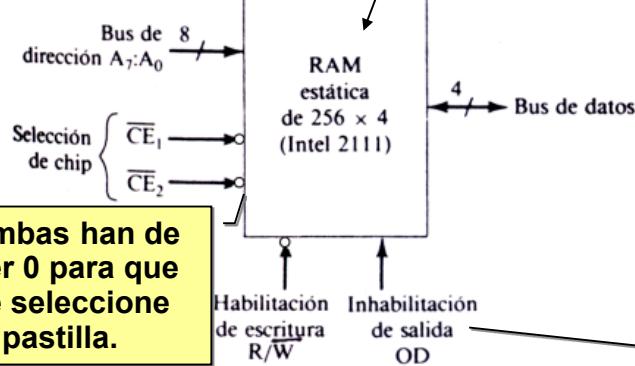
- Basta combinar las dos técnicas anteriores.



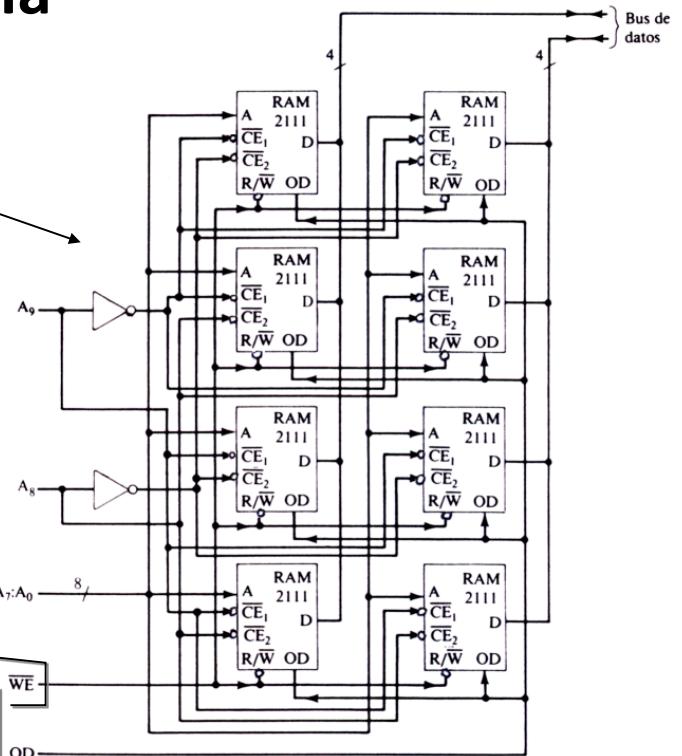
Ampliación de memoria

Ejemplo 2:

RAM de $1 K \times 8$ construida con 8 RAM de 256×4 .



Como hay un solo bus de datos \Rightarrow la línea OD (Output Disable) se activa durante los ciclos de escritura para evitar que se lean datos internos desde el chip al bus mientras el procesador usa el bus para salida datos.



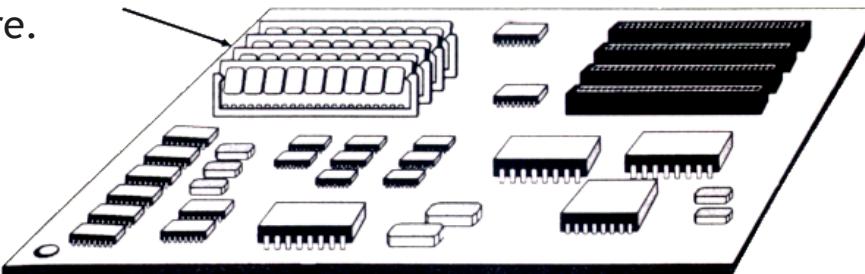
Módulos de memoria en línea

cubierta:
(RAM)

- En los 80, la memoria solía soldarse directamente en la placa madre del ordenador. Pero a medida que aumentaron los requisitos de memoria, esta técnica resultó poco factible.

- SIMM, DIMM, SODIMM:

- Varios chips DRAM en una pequeña placa de circuito impreso (PCB) que calza en un conector en la placa madre.



- ✓ método flexible para actualizar la memoria
- ✓ ocupa menos espacio en la placa madre.
- Normalmente sólo se ampliaba el bus de datos, no el de direcciones (no había necesidad de decodificador).

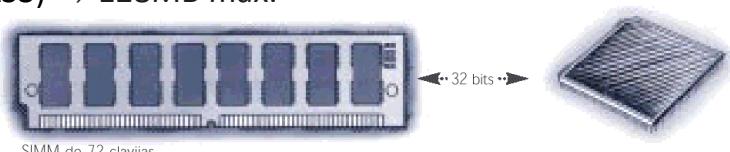
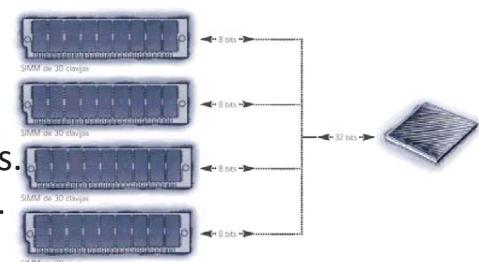
41

Módulos de memoria SIMM



- **SIMM (Single In-line Memory Module)**

- En un SIMM, cada contacto de una cara está conectado con el alineado en la otra cara para formar un único contacto.
- 80's y 90's, FPM y EDO-RAM
- 30 contactos:
 - 8 bits de datos \Rightarrow 4 SIMM para bus 32 bits.
 - 12 pines dirección \Rightarrow 24 bits dir \Rightarrow 16MB máx.
- 72 contactos:
 - 32 bits de datos (24 bits dir) \Rightarrow 64MB máx.
 - Dos rangos[†] (con /RAS1, /RAS3) \Rightarrow 128MB máx.



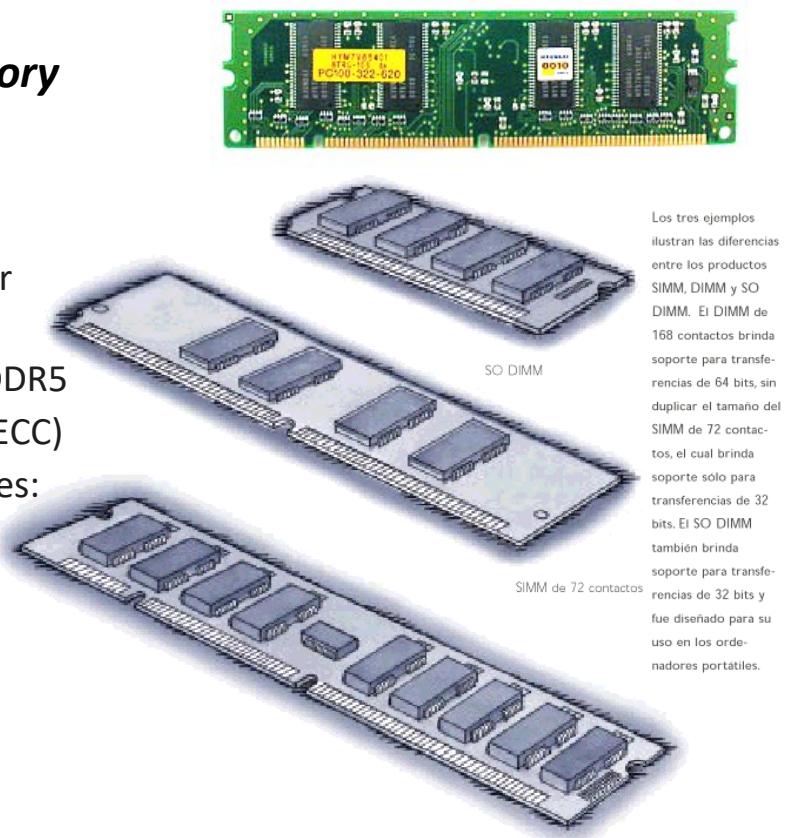
SIMM de 72 clavijas

[†] Ver https://en.wikipedia.org/wiki/SIMM#72-pin_SIMMs
https://en.wikipedia.org/wiki/Memory_rank

Módulos de memoria DIMM

■ **DIMM (Dual In-line Memory Module)**

- En DIMM los contactos opuestos están aislados eléctricamente para formar dos contactos separados.
- 90's-hoy, SDRAM-DDR...-DDR5
- 64 bits de datos (ó 72 con ECC)
- Incremento progresivo pines:
168-pin: FPM, EDO y SDRAM
184-pin: DDR
240-pin: DDR2, DDR3
288-pin: DDR4, DDR5



[†] Ver <https://en.wikipedia.org/wiki/DIMM#Ranking>

Módulos de memoria SODIMM



■ **SODIMM (Small Outline DIMM)**

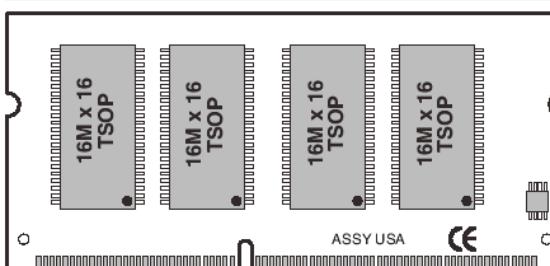
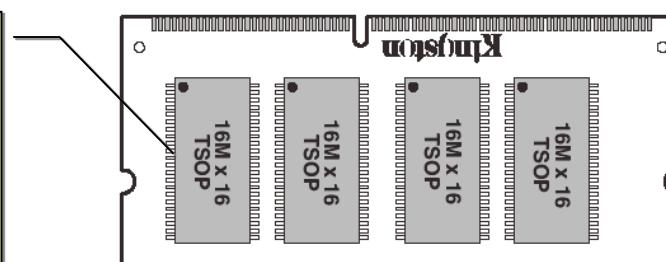
- Menos tamaño que un DIMM, menos contactos.
- Uso en portátiles. (100/144-pin SDR, 200-pin DDR-DDR2, 204-pin DDR3, 260-pin DDR4)
- Ejemplo: SODIMM SDR (144-pin) de 256 MB ($2 \times 16M \times 64$) a 133 MHz:

8 chips (4 en cada cara)
SDRAM a 133 MHz de
 $16M \times 16$

Cada chip tiene 4
bancos de $4M \times 16$

Bus de datos: 64 bits =
 $4 \text{ chips} \times 16 \text{ bits/chip}$

El módulo tiene 2 ranks
(¡en este caso sí se
amplía el número de
direcciones!)



[†] Ver <https://en.wikipedia.org/wiki/SO-DIMM>
<https://en.wikipedia.org/wiki/DIMM#Ranking>

Memoria I: Jerarquía de memoria

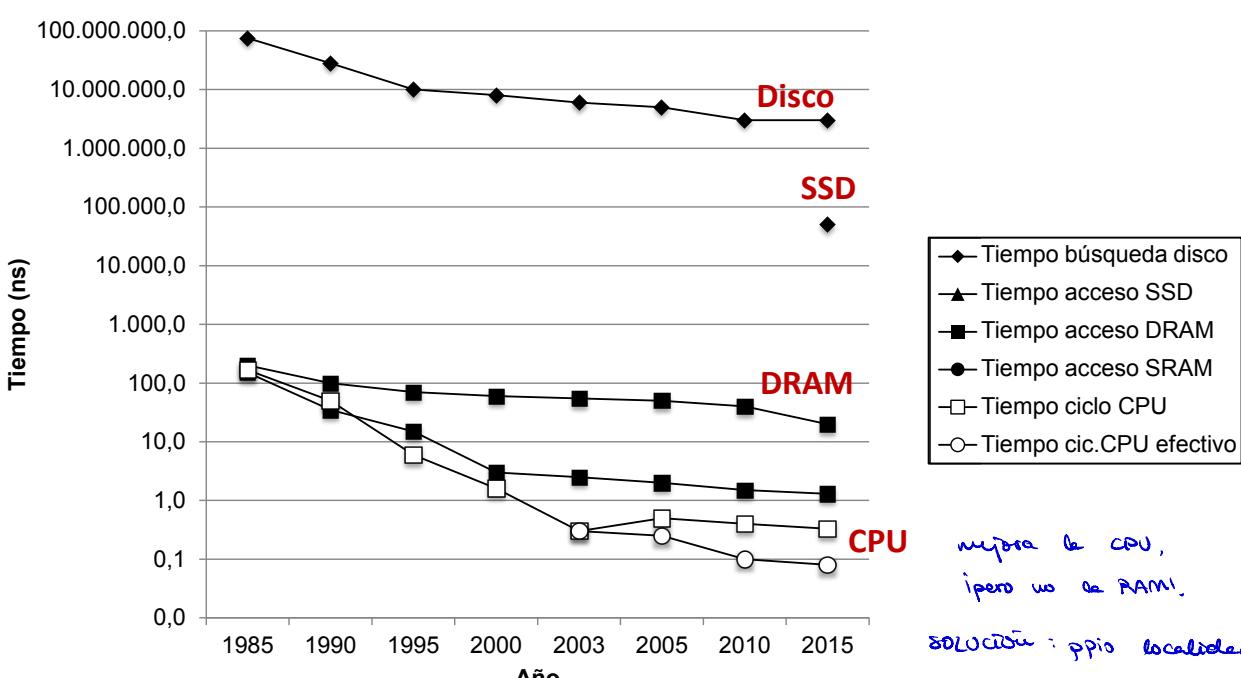
- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

45

La brecha[†] CPU-Memoria

La brecha entre velocidades de disco, DRAM y CPU se ensancha.

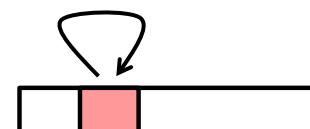


¡Localidad al rescate!

La clave para salvar esta brecha CPU-Memoria es una propiedad fundamental de los programas informáticos conocida como **localidad**

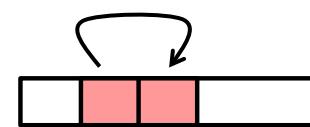
Localidad

- **Principio de localidad:** Los programas tienden a usar datos e instrucciones con direcciones iguales o cercanas a las que han usado recientemente



- **Localidad temporal:**

- Elementos referenciados recientemente probablemente serán referenciados de nuevo en un futuro próximo



- **Localidad espacial:**

- Elementos con direcciones cercanas tienden a ser referenciados muy juntos en el tiempo

Expresión matemática de localidad

- si en instante tiempo t se accede al dato/posición mem. $d(t)$...

■ Temporal

dentro de un poco \rightarrow *se accede al mismo*

- $d(t + n) = d(t)$ con n pequeño

■ Espacial

- $d(t + n) = d(t) + k$ con n, k pequeños

dentro de un poco \rightarrow *se accede a uno cercano*

49

Ejemplo de Localidad

```
sum = 0;           instrucción suma temporal
for (i = 0; i < n; i++) {
    sum += a[i];   aprox localidad espacial
}
return sum;
```

aprox localidad temporal \rightarrow *instrucciones del programa: espacial*

Localidad

Espacial o Temporal?

juntos en mem $a_2 \leftarrow a_3 \rightarrow a_4$ *lo ve cogido hacia poco* a_1, a_2, \dots, a_3

espacial

temporal

■ Referencias a datos

- Referenciar elementos array en sucesión (patrón de referencias de paso-1[†])
- Referenciar variable **sum** cada iteración

espacial

temporal

■ Referencias a instrucciones

- Referenciar instrucciones en secuencia
- Iterar bucle repetidamente

[†] stride-1 reference pattern

stride = paso, zancada

50

Estimaciones cualitativas de localidad

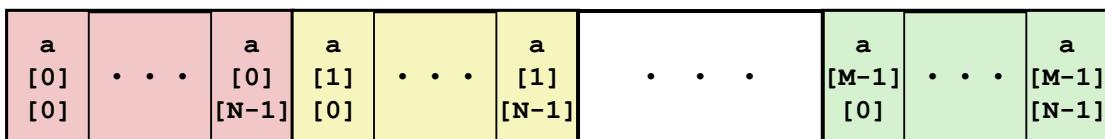
- **Afirmación:** Ser capaz de mirar un código y sacar una idea cualitativa de su localidad es una habilidad clave para un programador profesional
- **Pregunta:** ¿Tiene esta función buena localidad respecto al array a?

Pista: alm. por filas
(row-major order)

Respuesta: sí

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

51

Ejemplo de localidad

- **Pregunta:** ¿Tiene esta función buena localidad respecto al array a?

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

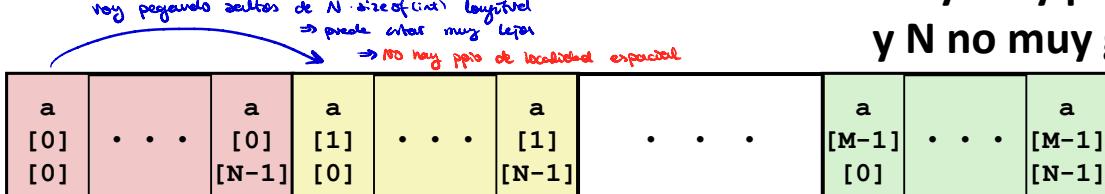
    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Justificación: las matrices se almacenan como un vector que concatena las filas

Respuesta: no, salvo si...

N es muy pequeño

...o M muy muy pequeño
y N no muy grande



Ejemplo de Localidad

pensar

- **Pregunta:** Se pueden permutar los bucles de forma que la función recorra el array 3-d a con patrón de referencias de paso-1 (y tenga por tanto buena localidad espacial)?

```
int sum_array_3d(int a[M] [N] [N] )
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];
    return sum;
}
```

Respuesta: sí: **for k,i,j, bucles en mismo orden índices** (segunda opción, peor) **for i,k,j,** haría NxM bucles (de N elem, paso-1)

Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

Jerarquía de Memoria

- Algunas propiedades fundamentales y perdurables del hardware y software:
 - Las tecnologías de almacenamiento rápidas cuestan más por byte, tienen menor capacidad y requieren más potencia (¡ calor!)
 - La brecha velocidad CPU-memoria principal se está ampliando
 - Los programas bien escritos tienden a exhibir buena localidad
- Estas propiedades fundamentales se complementan muy convenientemente
- Sugieren un enfoque para organizar sistemas de memoria y almacenamiento conocido como jerarquía de memoria

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

55

Ejemplo Jerarquía

Memoria



Caches

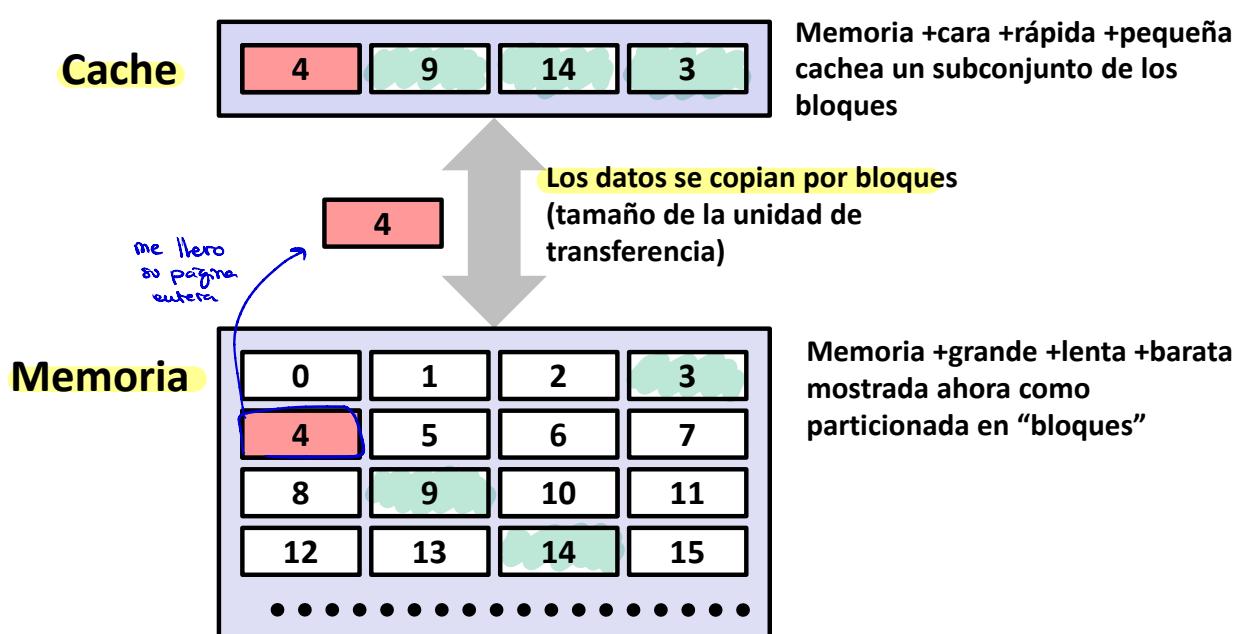
- **Cache:** Un dispositivo de almacenamiento más rápido y pequeño que funciona como zona de trabajo temporal para un subconjunto de los datos de otro dispositivo mayor y más lento
- Idea fundamental de una jerarquía de memoria:
 - $\forall k$, el dispositivo a nivel k (+rápido, +pequeño) sirve de cache para el dispositivo a nivel $k+1$ (+lento, +grande)
- ¿Por qué funcionan bien las jerarquías de memoria?
 - Debido a la localidad, los programas suelen acceder a los datos a nivel k más a menudo que a los datos a nivel $k+1$
 - Así, el almacenamiento a nivel $k+1$ puede ser más lento, y por tanto más barato (por bit) y más grande
- **Idea Brillante (ideal):** La jerarquía de memoria conforma un gran conjunto de almacenamiento que cuesta como el más barato pero que proporciona datos a los programas a la velocidad del más rápido

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

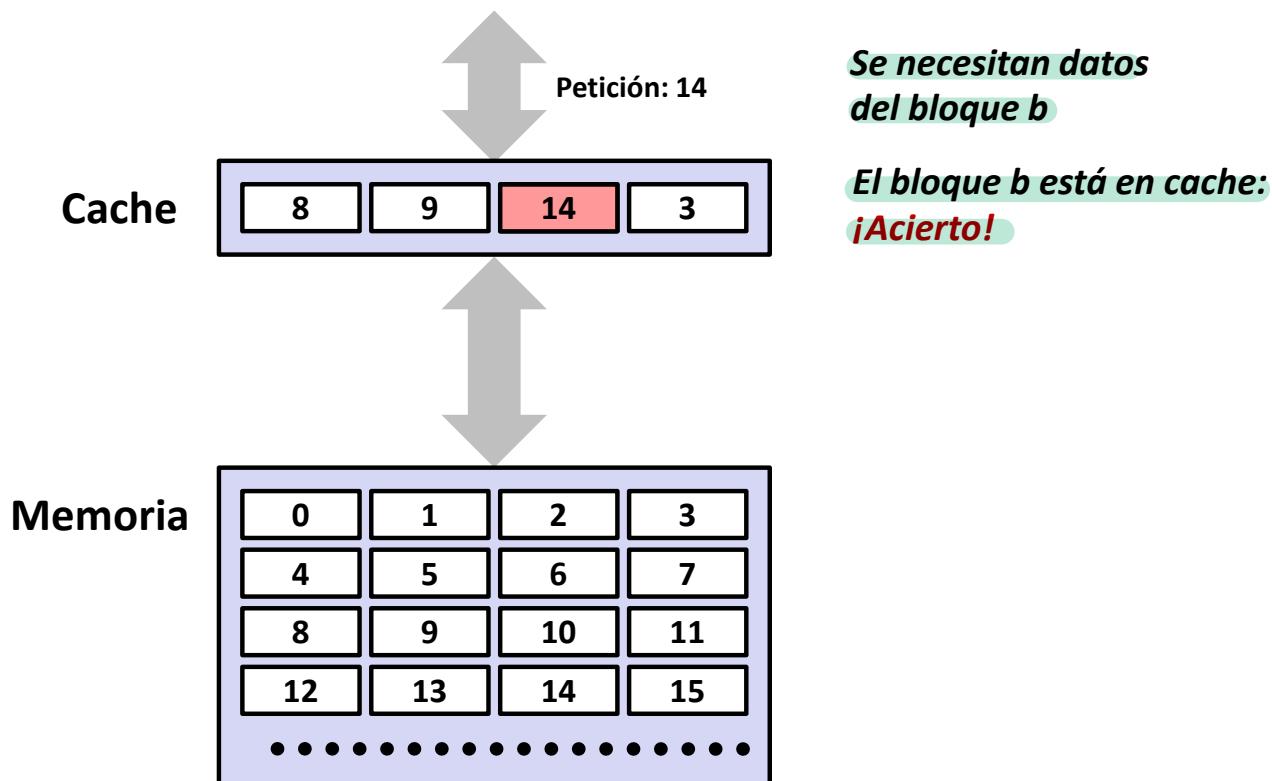
57

Conceptos Generales de Cache

memoria principal y cache se dividen en páginas (paginación)



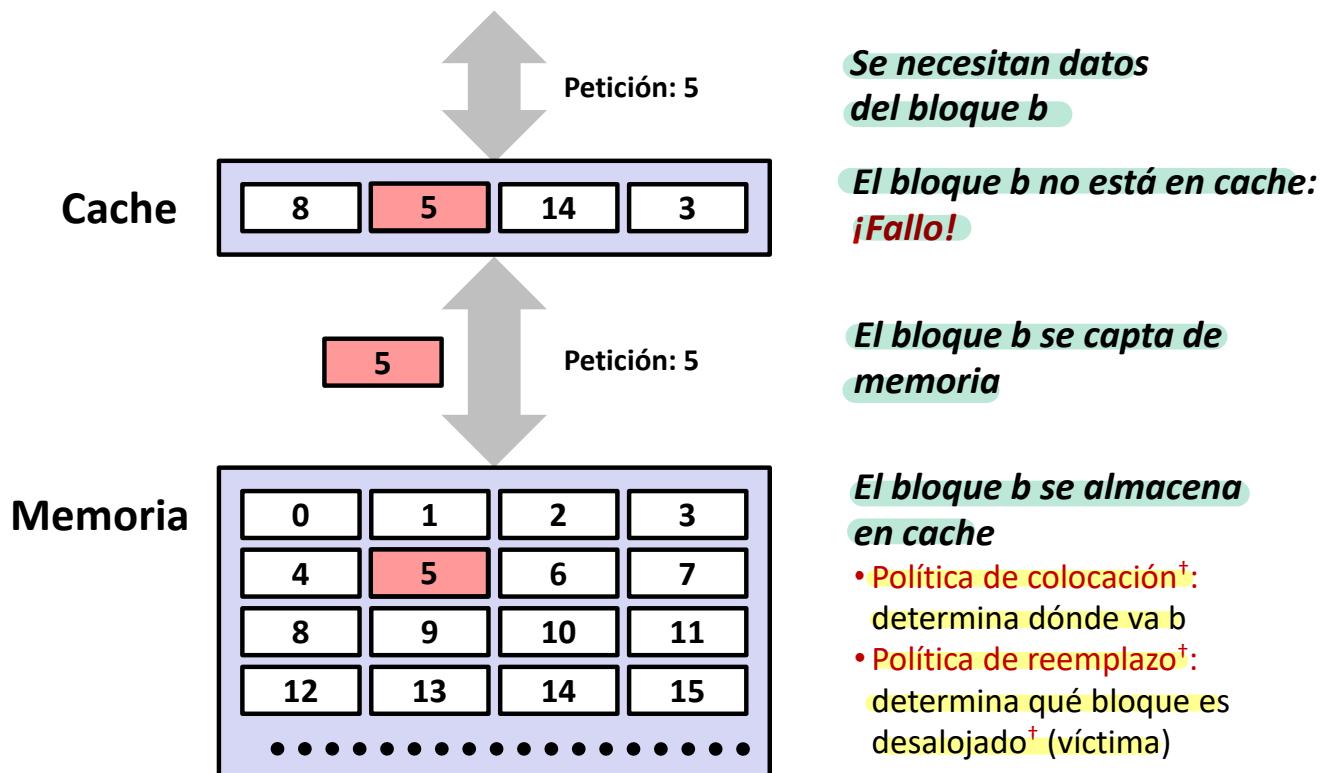
Conceptos Generales de Cache: Acierto[†]



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

[†] cache Hit 59

Conceptos Generales de Cache: Fallo[†]



Conceptos Generales de Cache:

3 Tipos de Fallo de Cache

■ Fallos en frío (obligados)

- Los fallos en frío ocurren porque la cache empieza vacía y esta es la primera referencia al bloque

Cuando acabamos de encender el PC

La cache empieza sin nada

■ Fallos por capacidad

- Ocurren cuando el conjunto de bloques activos (**conjunto de trabajo**) es más grande que la cache

Algunas algo para meter un nuevo dato, tengo que saber qué algo es en cache

■ Fallos por conflicto

- Mayoría caches limitan que los bloques a nivel $k+1$ puedan ir a pequeño subconjunto (a veces unitario) de las posiciones de bloque a nivel k
 - P.ej. Bloque i a nivel $k+1$ debe ir a bloque $(i \bmod 4)$ a nivel k (corr. directa)
- Fallos por conflicto ocurren cuando cache nivel k suficientemente grande pero a varios datos les corresponde ir al mismo bloque a nivel k
 - P.ej. Referenciar bloques $0, 8, 0, 8, 0, 8, \dots$ fallaría continuamente (ejemplo anterior con correspondencia directa)

No toda la cache completa
pero si la zona que recientemente

los fallos de colisión en una
tabla hash

Ejemplos de cacheado en Jerarquía Memoria



Tipo de Cache	Qué se cachea?	Dónde se cachea?	Latencia (ciclos)	Gestionado por
Registros	Palabras de 4-8 B	CPU (on-chip)	0	Compilador
TLB [†]	Trad. de direcciones	TLB (on-chip)	0	MMU [†] (hardw)
cache L1	Bloques de 64 bytes	L1 (on-chip)	4	Hardware
cache L2	Bloques de 64 B	L2 (on-chip)	10	Hardware
cache L3	Bloques de 64 B	L3 (on-chip)	50	Hardware
Memoria Virtual	Páginas de 4 KB	Memoria principal RAM	200	Hardware + SO
Buffer de disco	Partes de ficheros	Memoria principal	200	SO
cache de disco	Sectores de disco	Controladora de disco	100,000	Firmware disco
Buffer disco red	Partes de ficheros	Disco local	10,000,000	Cliente NFS [†]
cache Navegador	Páginas Web	Disco local	10,000,000	Navegador web
cache Web	Páginas Web	Discos de servidores remotos	1,000,000,000	Servidor web proxy [†]

[†] Translation Lookaside Buffer, Memory Management Unit,

Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

64

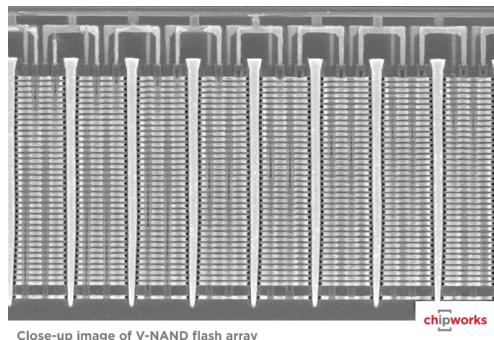
Tecnologías de almacenamiento

■ Discos Magnéticos



- Almacenamiento en medio magnético
- Acceso electromecánico

■ Memoria No-volátil (Flash)



- Almacenamiento como carga persistente

■ Implementado con estructura 3-D[†]

- +100 niveles de celdas
- 3 bits de datos por celda

[†] Ver https://en.wikipedia.org/wiki/Flash_memory#Vertical_NAND

¿Qué hay dentro de una unidad de disco?

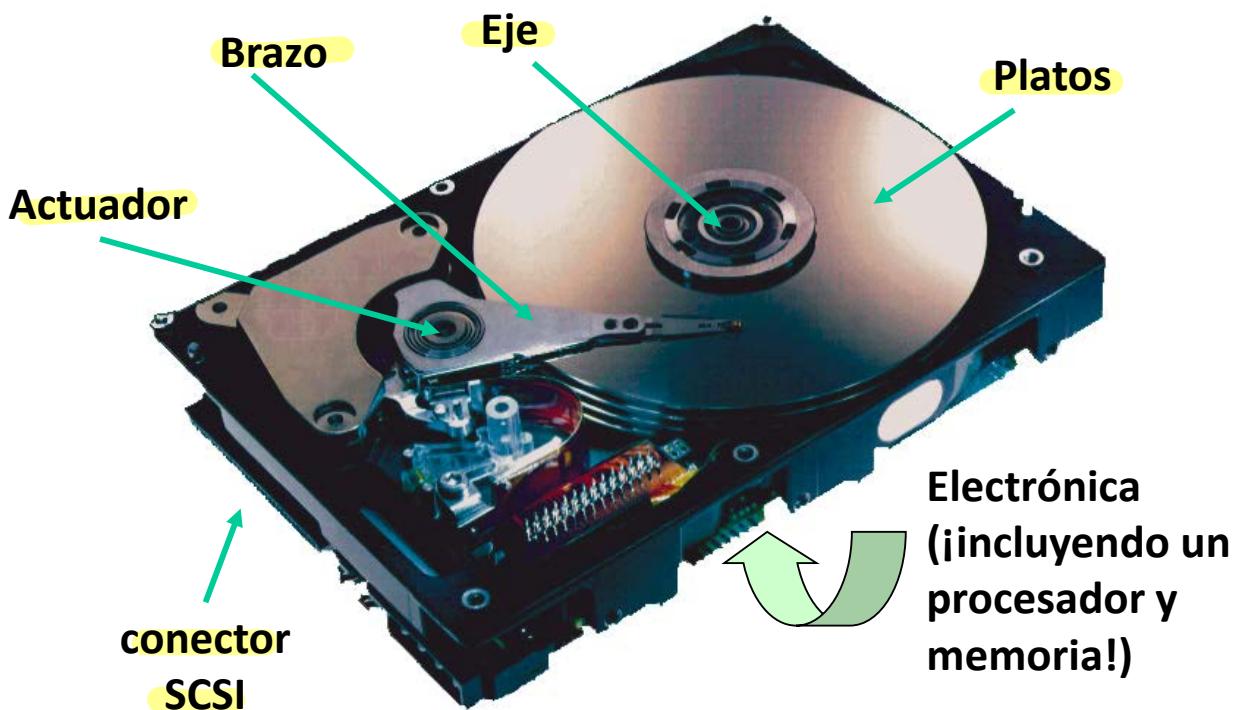


Imagen cortesía de Seagate Technology

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

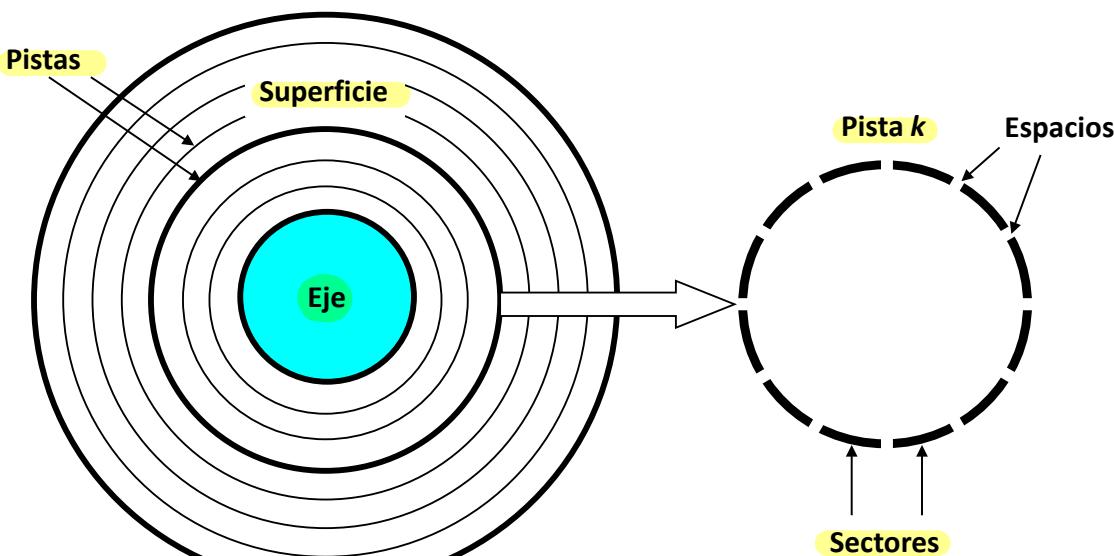
66

Grado Informática, 2º Curso

Estructura de Computadores

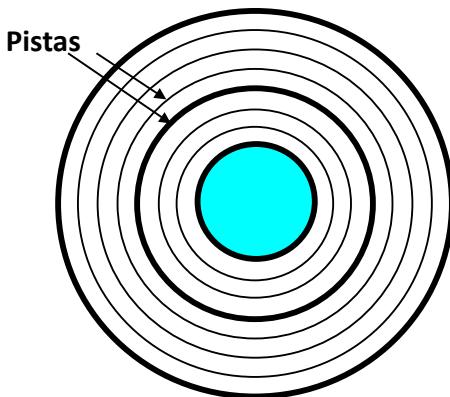
Geometría de un disco

- Los discos consisten en **platos** con dos **superficies (caras)**[†]
- Cada cara consiste en **anillos concéntricos** llamados **pistas**
- Cada pista consiste en **sectores** separados por **espacios**[†]



Capacidad de un disco

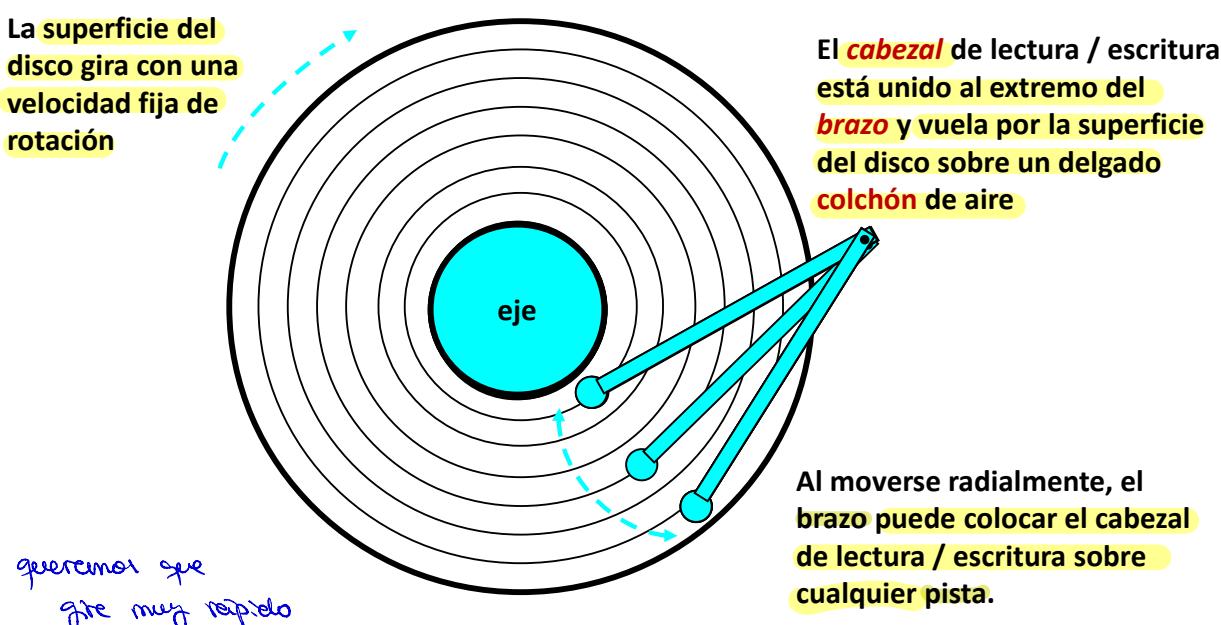
- **Capacidad:** máximo número de bits que se pueden almacenar
 - Los proveedores expresan la capacidad en unidades de gigabytes (GB) o terabytes (TB), donde 1 GB = 10^9 Bytes y 1 TB = 10^{12} Bytes
- La capacidad queda determinada por estos factores tecnológicos:
 - Densidad de grabación (bits/pulgada[†]): nº de bits que se pueden comprimir en un tramo de pista de 1 pulgada
 - Densidad de pistas (radial) (pistas/pulgada): nº de pistas que se pueden comprimir en un tramo radial de 1 pulgada
 - Densidad superficial (bits/pulg²): producto de ambas densidades (grabación y radial)



[†] bits/in, bits per inch 68

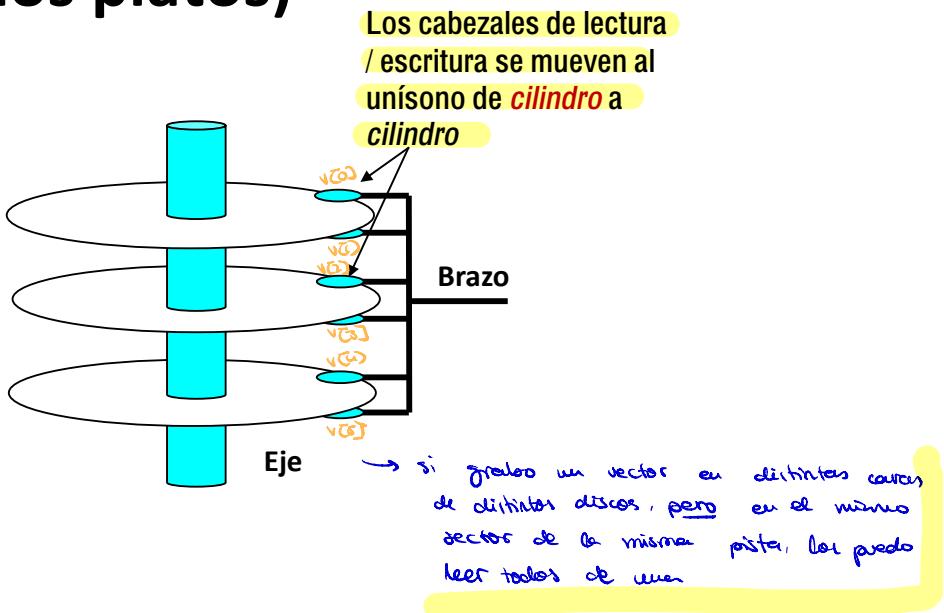
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Funcionamiento de un disco (Visto en un solo plato)



Funcionamiento de un disco

(Visto en varios platos)



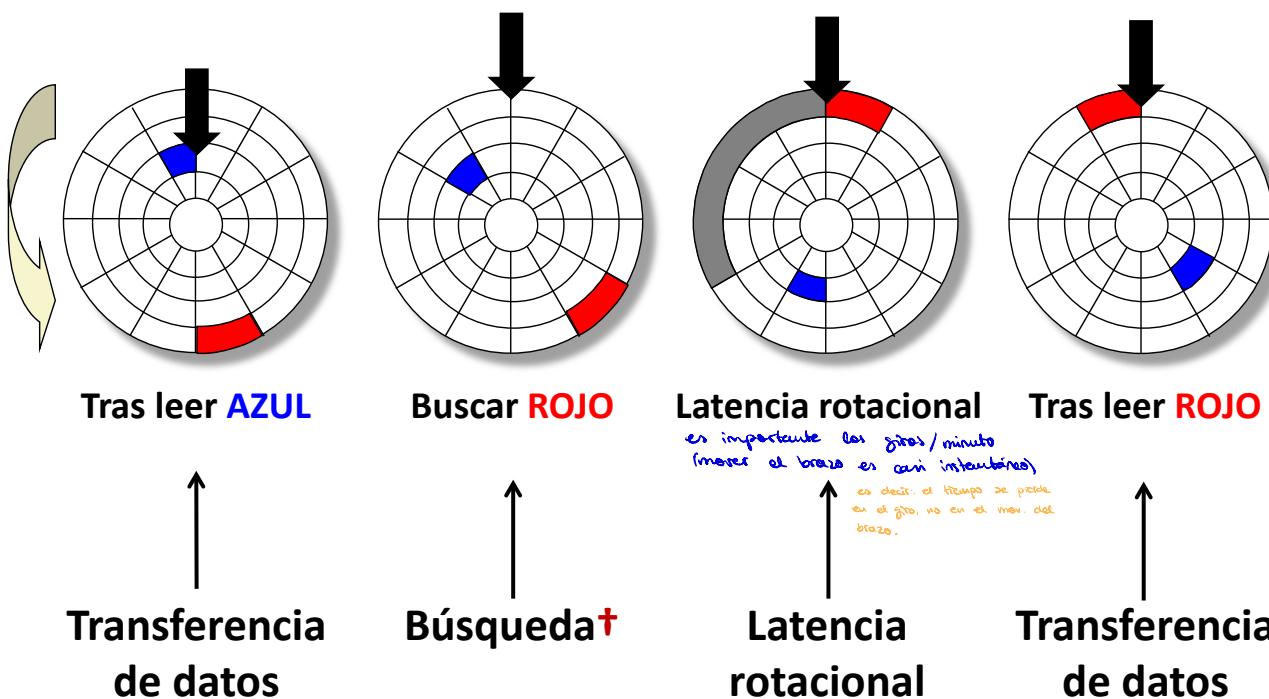
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

70

Grado Informática, 2º Curso

Estructura de Computadores

Acceso a disco: componentes del tiempo de servicio



Tiempo de acceso a disco

■ Tiempo promedio acceso algún sector determinado, aprox:

- $T_{\text{acceso}} = T_{\text{prom b\u00fasqueda}} + T_{\text{prom rotaci\u00f3n}} + T_{\text{prom transferencia}}$

■ Tiempo de b\u00fasqueda ($T_{\text{prom b\u00fasqueda}}$)

- Tiempo para colocar cabezales sobre el cilindro que contiene el sector
- Valores t\u00edpicos $T_{\text{prom b\u00fasqueda}} = 3 - 9 \text{ ms}$

■ Latencia rotacional ($T_{\text{prom rotaci\u00f3n}}$)

- Tiempo esperando a que pase bajo cabezales el primer bit del sector
- $T_{\text{prom rotaci\u00f3n}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ s}/1 \text{ min}$
- Velocidad rotacional t\u00edpica = 7,200 RPMs ($\Rightarrow 4.17 \text{ ms}$)

■ Tiempo de transferencia ($T_{\text{prom transferencia}}$)

- Tiempo para leer los bits del sector
- $T_{\text{prom transferencia}} = 1/\text{RPMs} \times 1/(\# \text{ sectores/pista prom}) \times 60 \text{ s}/1 \text{ min}$

Tiempo para una rotaci\u00f3n (minutos) fracci\u00f3n de rotaci\u00f3n a leer

Ejemplo de tiempo de acceso a disco

■ Dados:

- Velocidad rotacional = 7,200 RPM
- Tiempo b\u00fasqueda promedio = 9 ms
- # sectores/pista promedio = 400

■ Calcular:

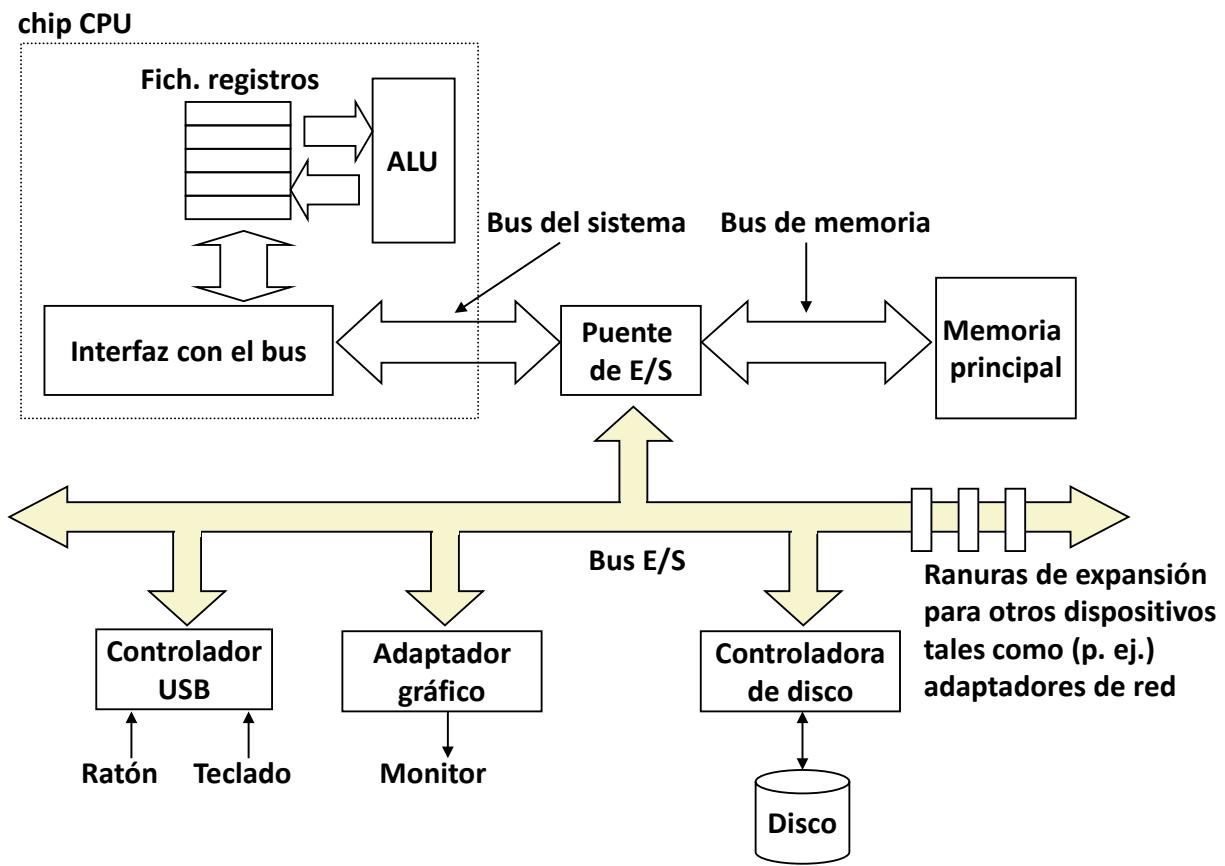
$$\frac{1}{2} \cdot \left(\frac{60 \text{ s}}{7200 \text{ RPM}} \right) \cdot 1000 \text{ ms/s} = 4.17 \text{ ms}$$

- $T_{\text{prom rotaci\u00f3n}} = 1/2 \times (60 \text{ s}/7200 \text{ RPM}) \times 1000 \text{ ms/s} = 4.17 \text{ ms}$
- $T_{\text{prom transferencia}} = 60/7200 \times 1/400 \times 1000 \text{ ms/s} = 0.02 \text{ ms}$
- $T_{\text{acceso}} = 9 \text{ ms} + 4.17 \text{ ms} + 0.02 \text{ ms} = 13.19 \text{ ms}$

■ Puntos importantes:

- Tiempo acceso dominado por tiempo b\u00fasqueda y latencia rotacional
- El primer bit en un sector es el m\u00e1s caro (lento), el resto sale gratis.
- *Tiempo acceso SRAM aprox. 4ns/palabra 64b, DRAM aprox. 60ns*
 - *Disco es aprox. 40.000 veces m\u00e1s lento que SRAM,*
 - *2.500 veces m\u00e1s lento que DRAM^t*

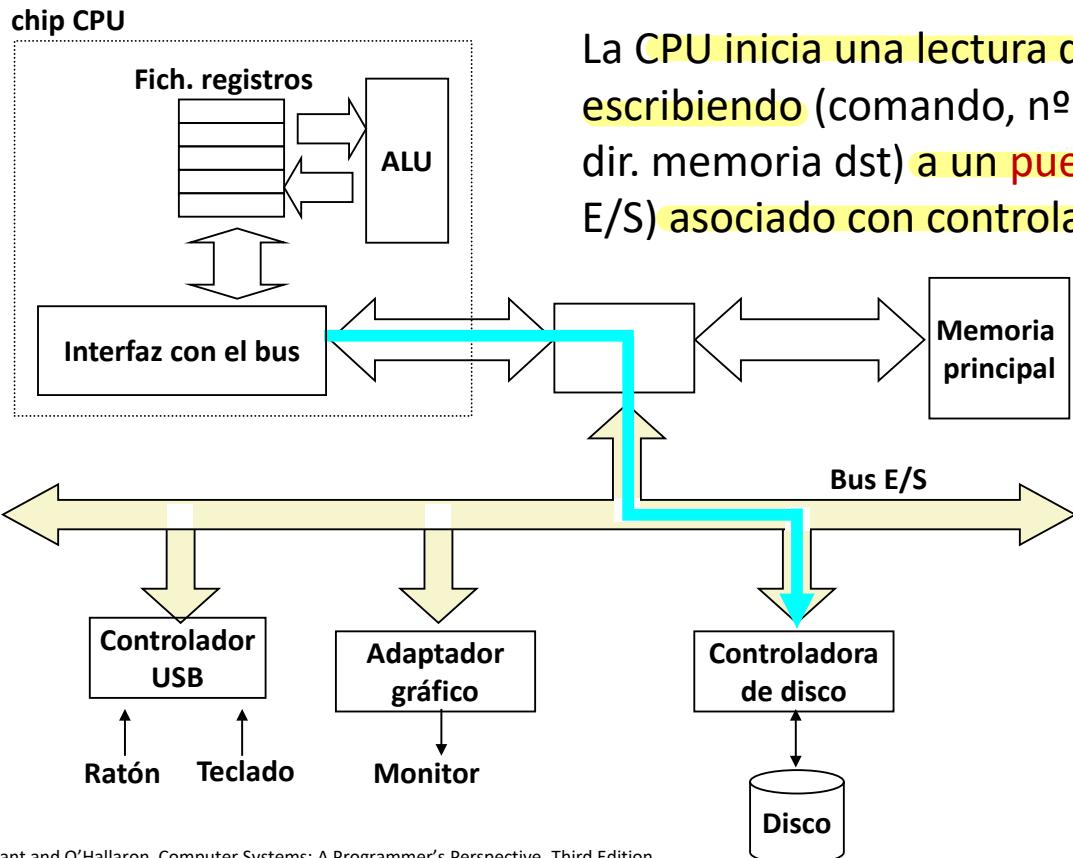
Bus de E/S



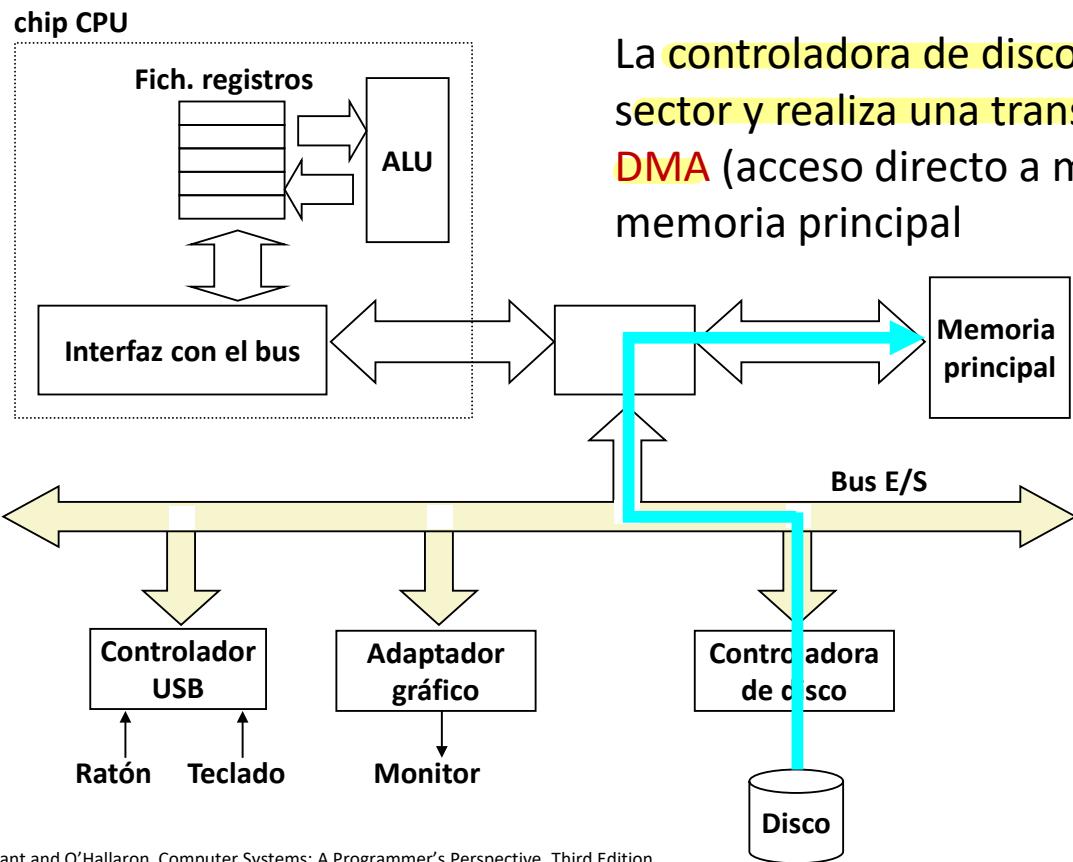
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

74

Lectura de un sector de disco (1)



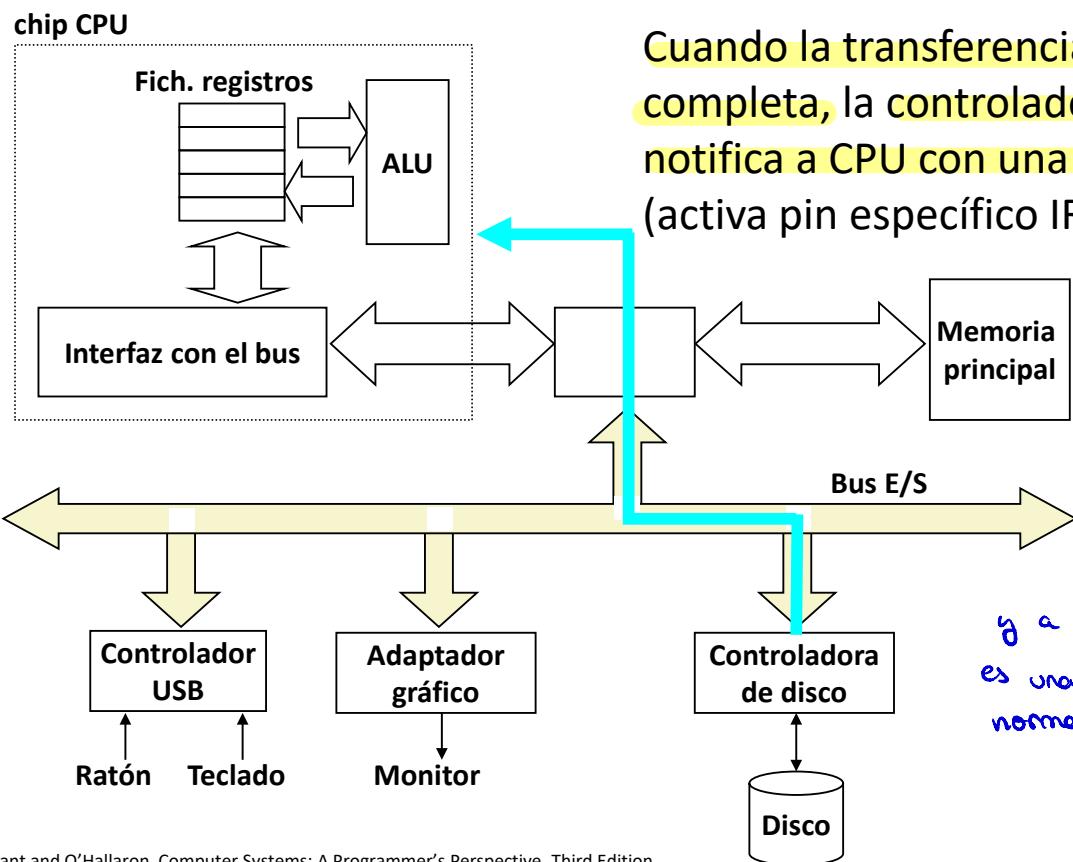
Lectura de un sector de disco (2)



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

76

Lectura de un sector de disco (3)



Memorias no volátiles

- **SRAM y DRAM son memorias volátiles**
 - Pierden información si se apagan (*sin electricidad*)
- **Las memorias no volátiles retienen valores incluso si se apagan**
 - funcionan ordenador desde la BIOS (memoria no volátil)*
 - Memoria de sólo lectura (ROM):
 - programada durante su fabricación
 - PROM progr. usr. irreversible, EPROM borrible (luz UV) → 
 - EEPROM: PROM borrible eléctricamente
 - Memorias Flash: EEPROMs con capacidad borrado parcial (por bloques)
 - se desgastan tras aprox. 100.000 borrados
 - 3D XPoint[†] (Intel Optane) & NVMs emergentes
 - nuevos materiales



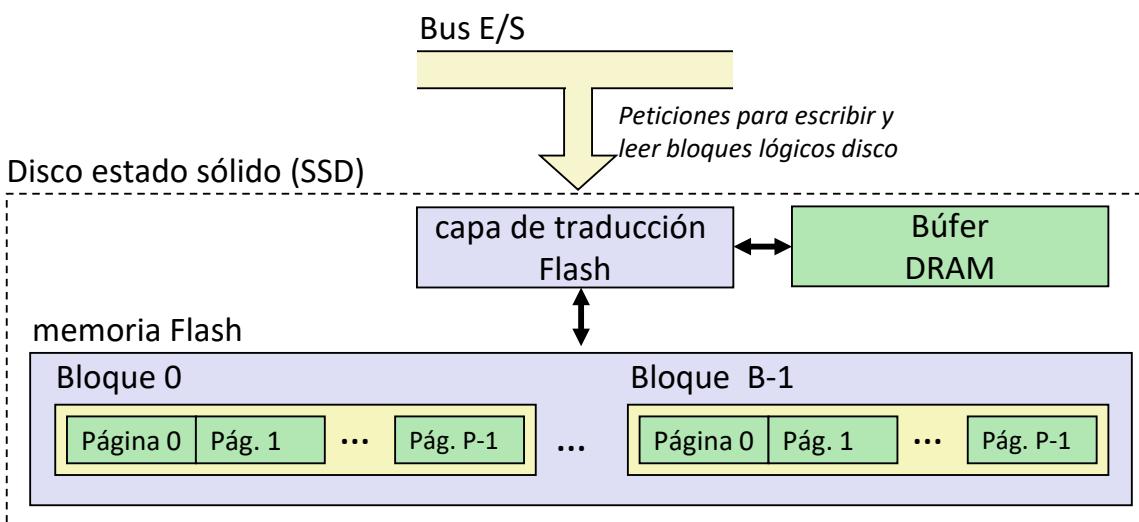
■ Aplicaciones de las memorias no volátiles

- Programas firmware almacenados en una ROM (BIOS, controladoras de disco, tarjetas de red, aceleradores gráficos, subsistemas seguridad...)
- Discos de estado sólido (reemplazando discos giratorios)
- Caches de disco

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

[†] cross point, ver https://en.wikipedia.org/wiki/3D_XPoint 78

Discos de estado sólido (SSDs)



- **Páginas: de 4KB a 512KB, Bloques: de 32 a 128 páginas**
- **Datos escritos/leídos en unidades de páginas**
- **Una página se puede escribir sólo tras borrar su bloque**
- **Un bloque se desgasta tras unas 100.000 escrituras**

Prestaciones características SSD

■ Benchmark[‡] de un Samsung 970 EVO Plus

<https://ssd.userbenchmark.com/SpeedTest/711305/Samsung-SSD-970-EVO-Plus-250GB>

Rendimiento lectura secuencial	2.126 MB/s	Rendmto. escritura sec.	1.880 MB/s
Rendimiento [†] lectura aleatoria	140 MB/s	Rendmto. escritura aleat.	59 MB/s

■ Acceso secuencial mucho más rápido que aleatorio

- Tema omnipresente en jerarquías de memoria

■ Escrituras aleatorias son especialmente lentas

- Borrar un bloque lleva mucho tiempo (~1ms)
- Modificar una página de un bloque requiere que todas las otras se copien a un nuevo bloque
- La capa de traducción Flash permite acumular una serie de pequeñas escrituras antes de realizar una escritura de bloque

[‡] test, prueba de referencia

[†] throughput

80

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

SSD frente a Discos giratorios

■ Ventajas

- No partes móviles → más rápidos, menor consumo, más resistentes

■ Desventajas

▪ Eventual desgaste

- mitigado por “lógica nivelado desgaste” en capa traducción flash
- p.ej. Samsung 970 EVO Plus garantiza que se pueda escribir 600x la capacidad del disco antes de desgastarse[†]
- controladora migra datos para repartir/minimizar nivel desgaste
- En 2019, aprox. 4x más caro por byte (que giratorios)
- y seguirá cayendo ese coste relativo

■ Aplicaciones

- reproductores MP3, smartphones, portátiles
- cada vez más común en servidores y PC sobremesa

[†] 600x << 100.000 borrados bloques

ver folleto en <https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/970evoplus/>

Memoria I: Jerarquía de memoria

- La abstracción de memoria (concepto de Lectura y Escritura)
- RAM: bloque constructivo de memoria principal
- Configuración y diseño de memorias utilizando varios chips
- Localidad de las referencias
- Jerarquía de memoria
- Tecnologías de almacenamiento, y tendencias

Resumen

- **La brecha de velocidad entre CPU, memoria y almacenamiento masivo continúa ampliándose.**
- **Los programas bien escritos exhiben una propiedad llamada localidad.**
- **Las jerarquías de memoria, basadas en cacheado, cierran la brecha al explotar la localidad.**
- **El progreso en memoria flash está sobreponiendo a todas las demás tecnologías de memoria y almacenamiento (DRAM, SRAM, disco magnético)**
 - Capaz de apilar celdas en tres dimensiones

Memorias Cache

Estructura de Computadores
Semana 15

Bibliografía:

[BRY16] Cap.6 Computer Systems: A Programmer's Perspective 3rd ed. Bryant, O'Hallaron. Pearson, 2016
Signatura ESIIT/[C.1 BRY com](#)

Transparencias del libro CS:APP, Cap.6
Introduction to Computer Systems: a Programmer's Perspective
Autores: Randal E. Bryant y David R. O'Hallaron
<http://www.cs.cmu.edu/afs/cs/academic/class/15213-f15/www/schedule.html>

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

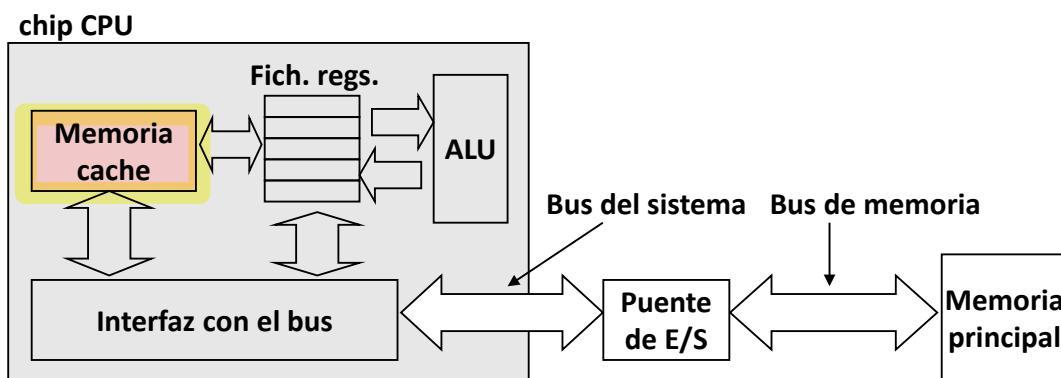
1

Memoria II: Cache

- **Organización y Funcionamiento de la memoria cache**
- **Impacto de la cache en el rendimiento**
 - Modelo de evaluación
 - La montaña de memoria
- **Programación de código aprovechando la cache**

Memorias Cache

- Las **memorias cache** son **memorias pequeñas y rápidas basadas en SRAM gestionadas automáticamente por hardware**
 - Retiene **bloques de memoria principal accedidos frecuentemente**
- La **CPU busca los datos primero en caché**
- Estructura típica del sistema:



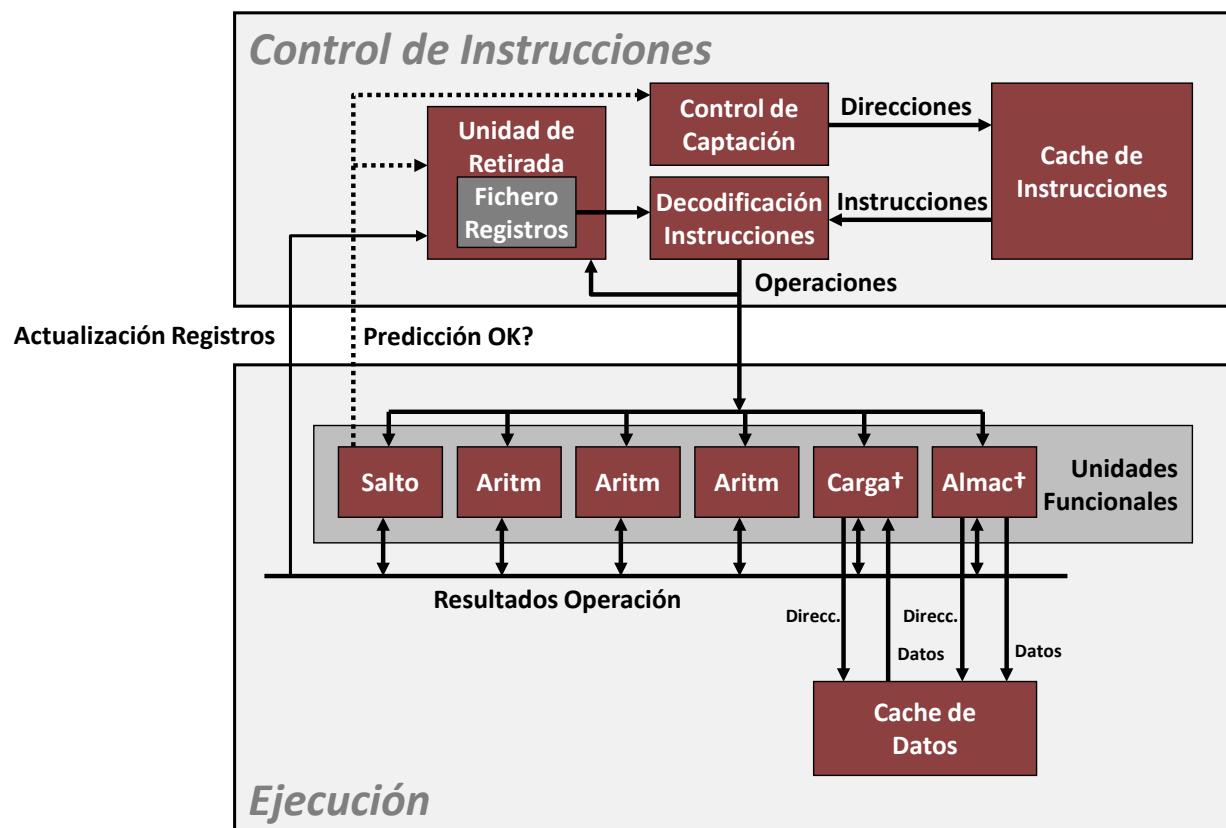
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

10

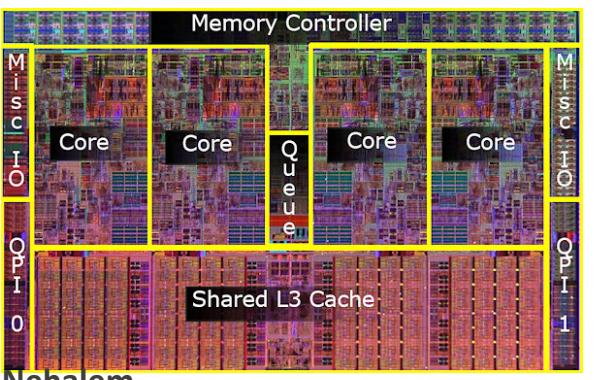
Grado Informática, 2º Curso

Estructura de Computadores

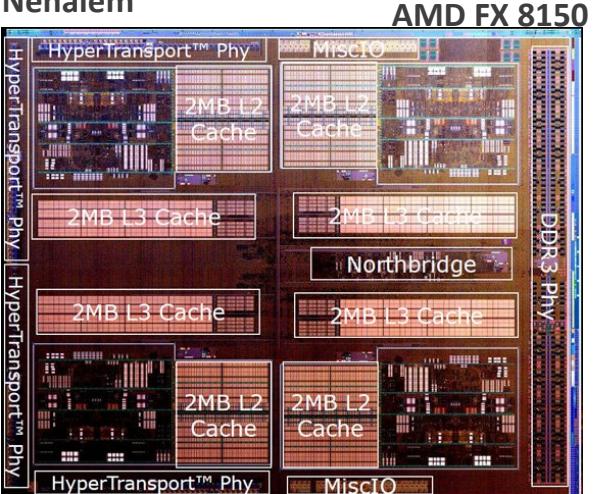
Diseño moderno de CPU



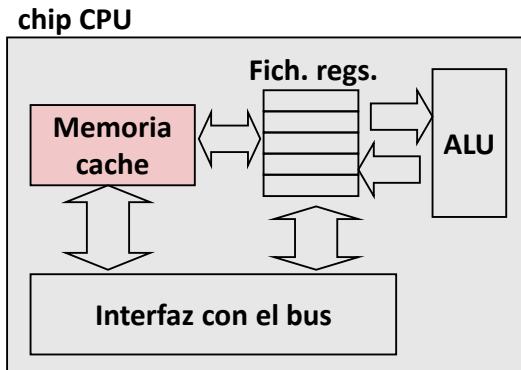
El aspecto que tiene realmente



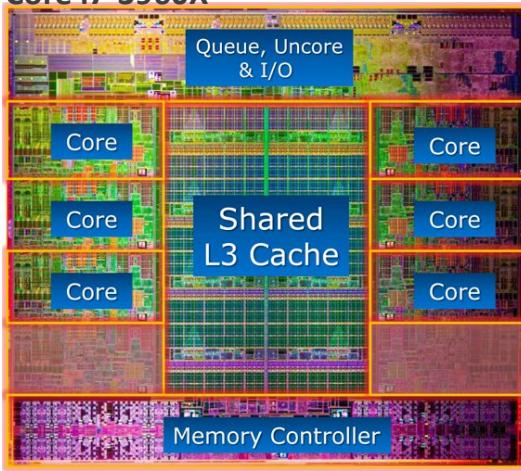
Nehalem



AMD FX 8150



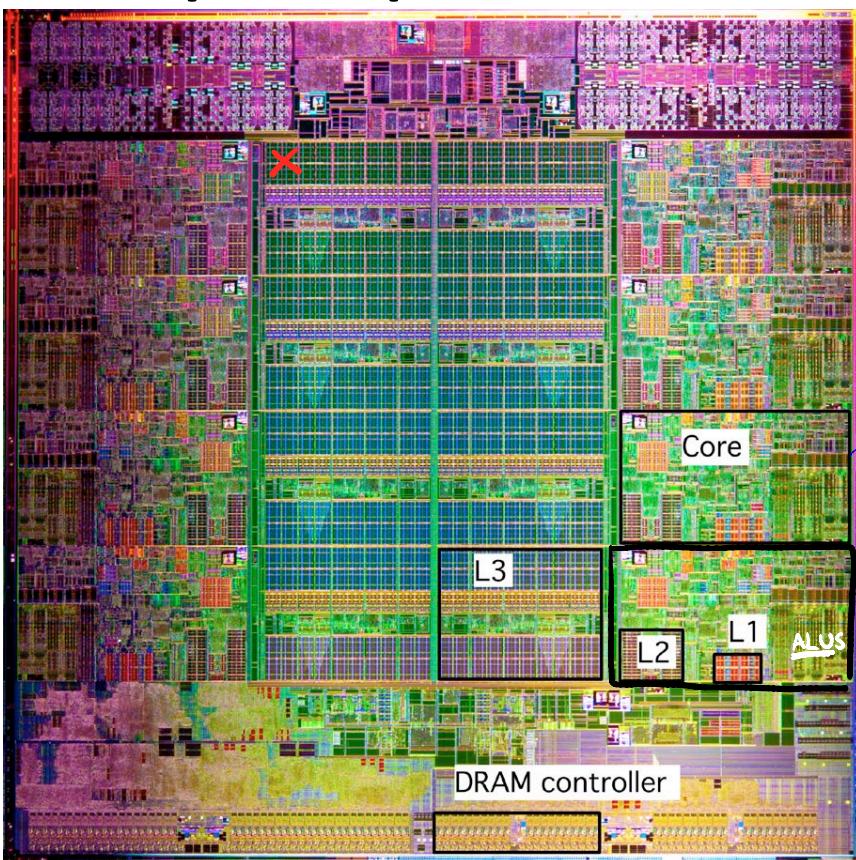
chip CPU



Core i7-3960X

12

Chip El aspecto que tiene realmente (Cont.)

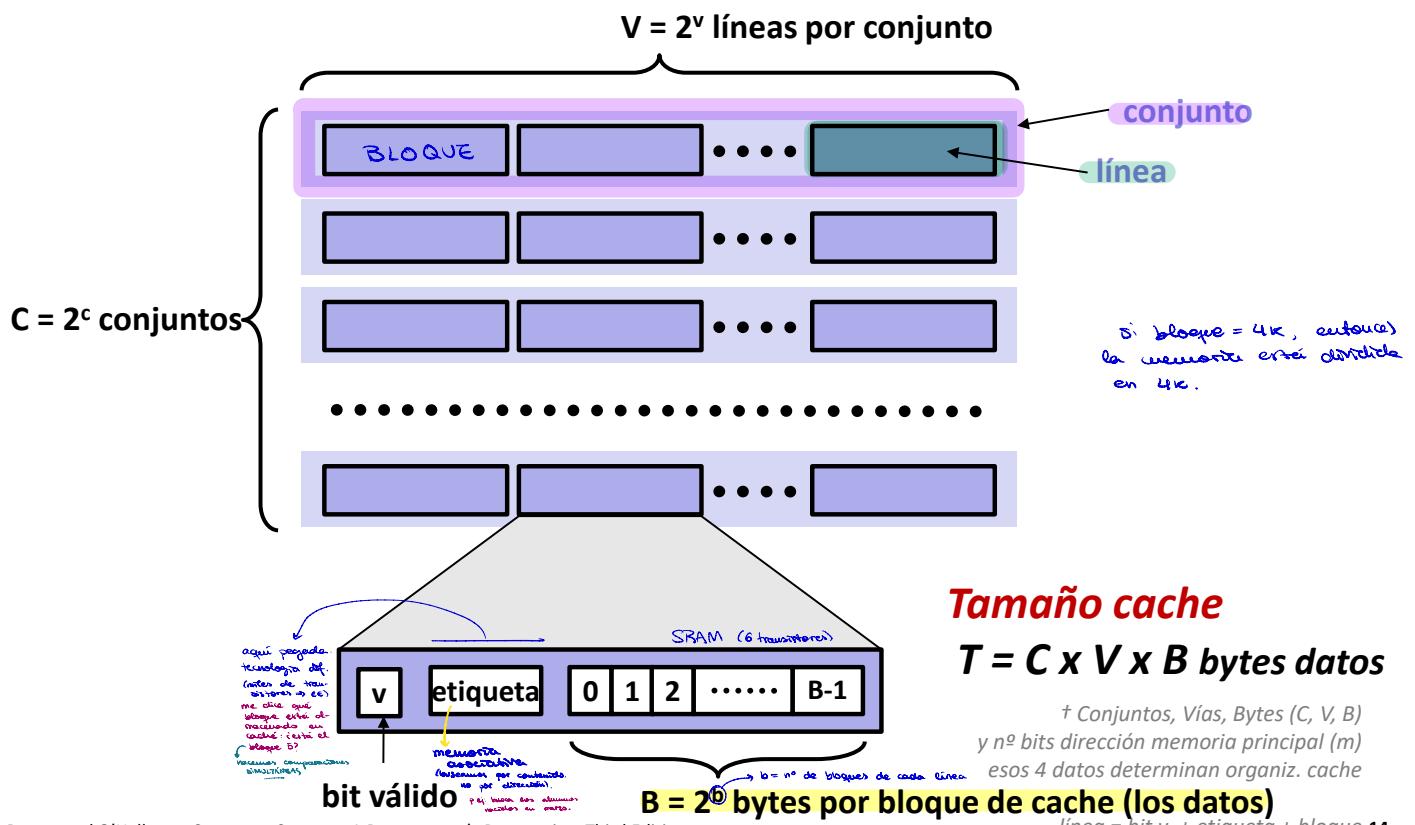


múltiples ALU (unidades repetidas)

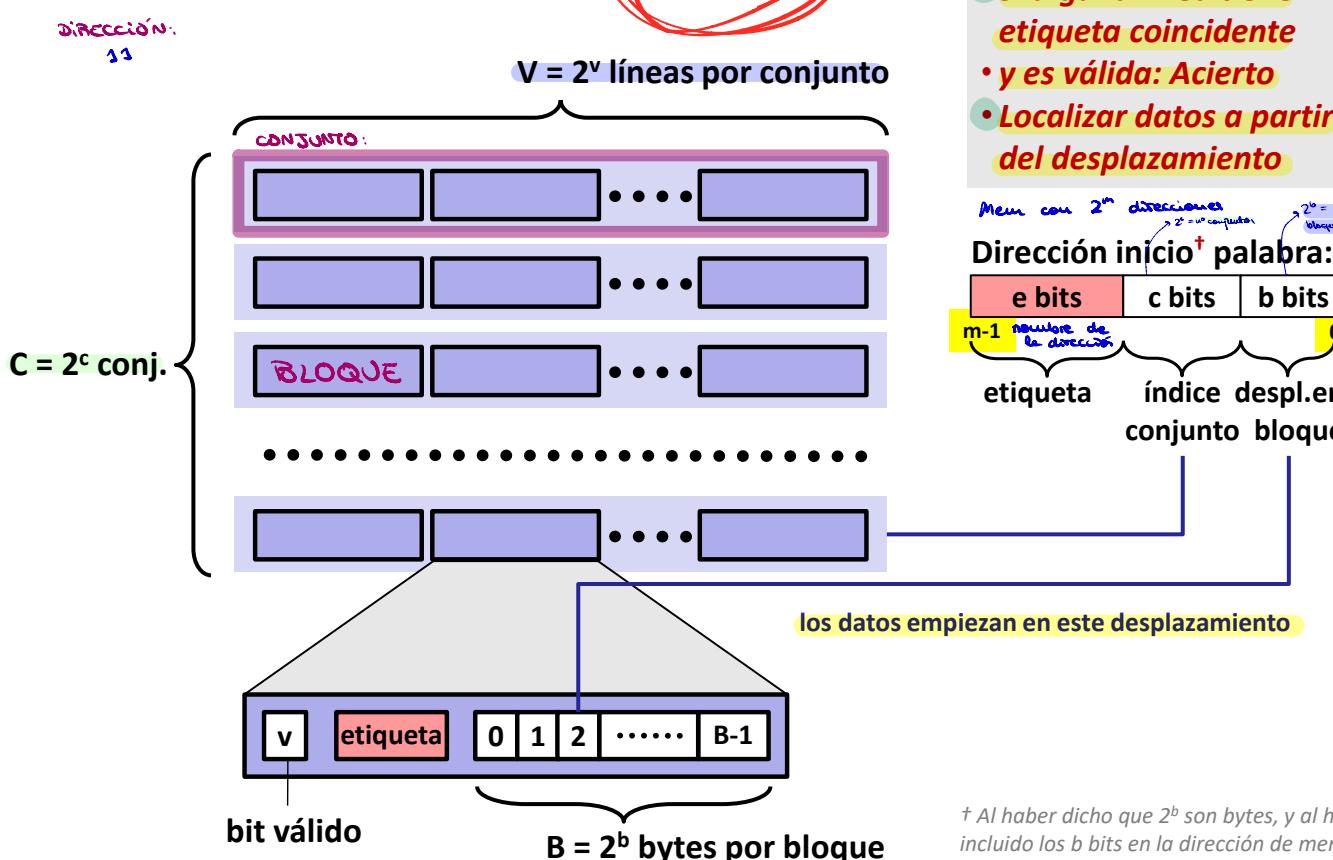
Dado + Procesador Intel Sandy Bridge

- L1: 32KB Instrucciones + 32KB Datos
 - L2: 256KB
 - L3: 3–20MB
- Compartida entre todos los procesadores (solo del chip en su modo, puede ser unida x j perder memoria trae mas tiempo)
- Die = dado, Wafer = oblea, que se corta en dados

Organización General de Cache (C, V, B, m^+)



Lectura de Cache



Ej: Cache con Correspondencia Directa ($V = 1$)

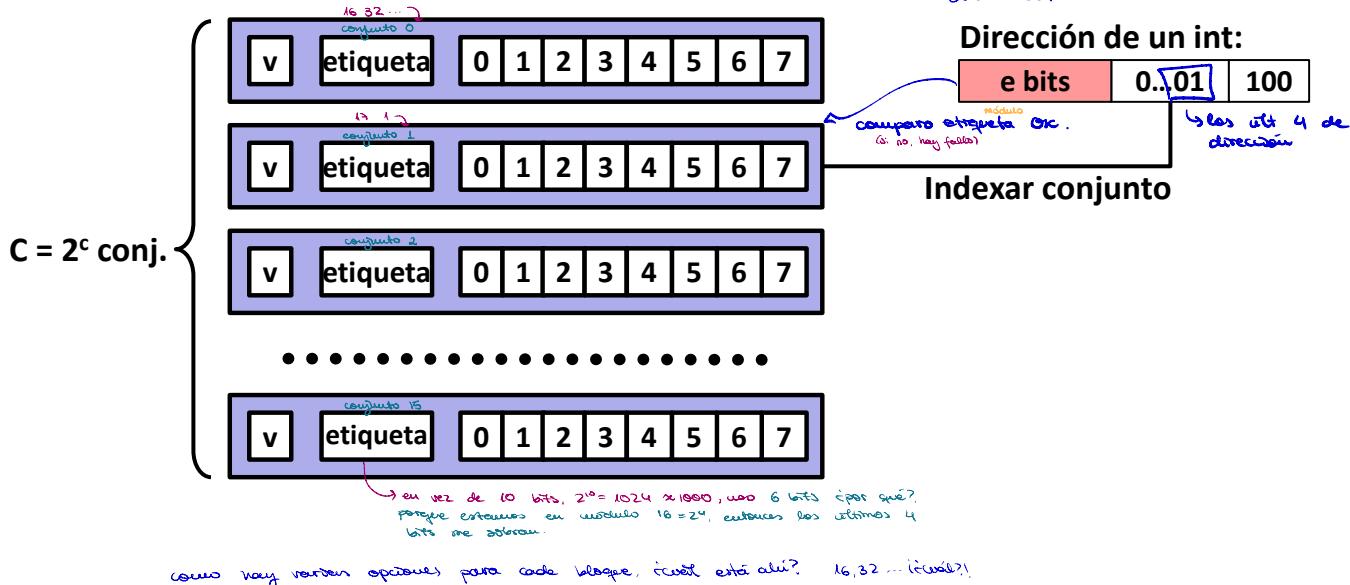
Correspondencia directa: Una línea por conjunto

Suponer: tamaño bloque cache $B=8$ bytes



Cuando hago algo de memoria en cache hay distintas alternativas

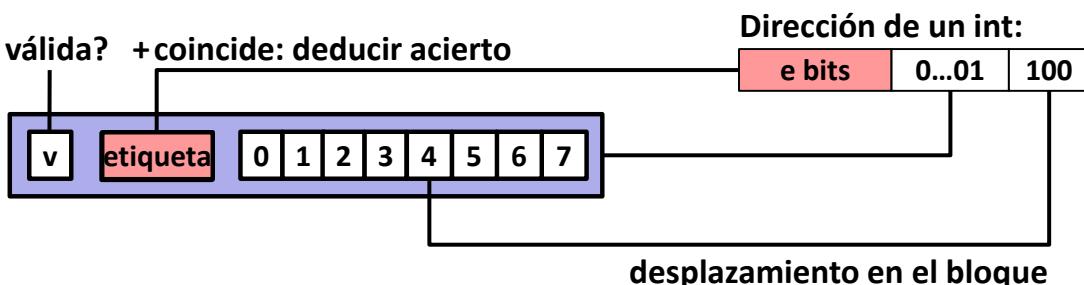
→ todos los conjuntos sólo pueden almacenar un bloque



Ej: Cache con Correspondencia Directa ($V = 1$)

Correspondencia directa: Una línea por conjunto

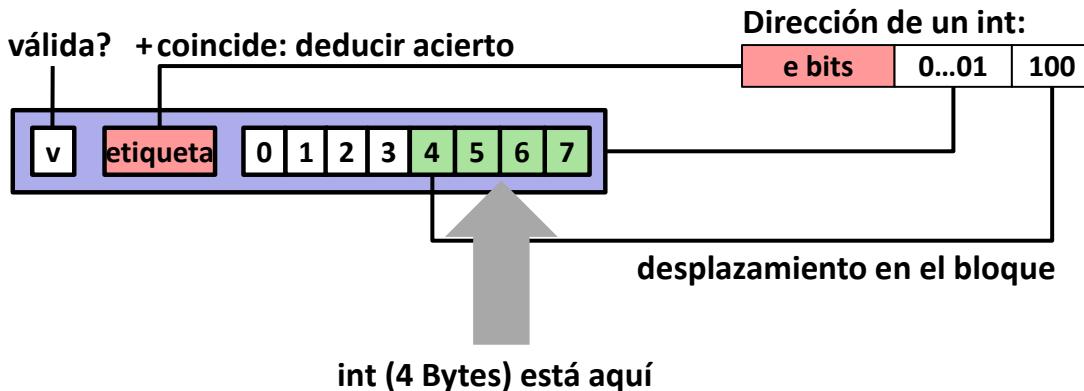
Suponer: tamaño bloque cache $B=8$ bytes



Ej: Cache con Correspondencia Directa ($V = 1$)

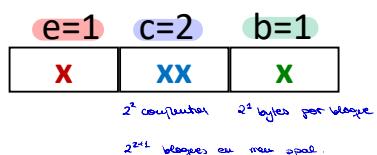
Correspondencia directa: Una línea por conjunto

Suponer: tamaño bloque cache $B=8$ bytes



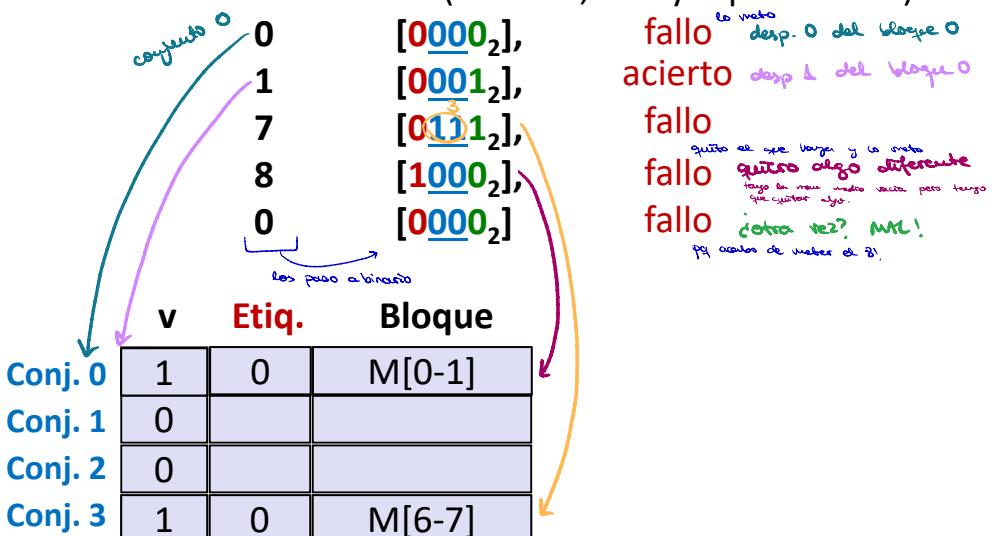
**Si etiqueta no coincide (= fallo): vieja línea desalojada y reemplazada
(nuevos datos y nueva etiqueta)**

Simulación cache correspondencia directa



1 vía / conjunto
 $\rightarrow 4 \cdot 1 \cdot 2 = 1 = e$
 2^4 Bytes
 direcciones 4 bits (espacio dirección tam M=16 bytes)
 C=4 conjuntos, V=1 vía (bloq./conj.), B=2 bytes/bloque
 $4 \text{ conj} \rightarrow 2^2 \text{ conj}$
 $c=2$
 $B=2^b$ B/bloque $\rightarrow b=1$

Traza de direcciones (lecturas, un byte por lectura):

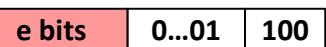


Cache Asociativa por Conjuntos de V vías (con V=2)

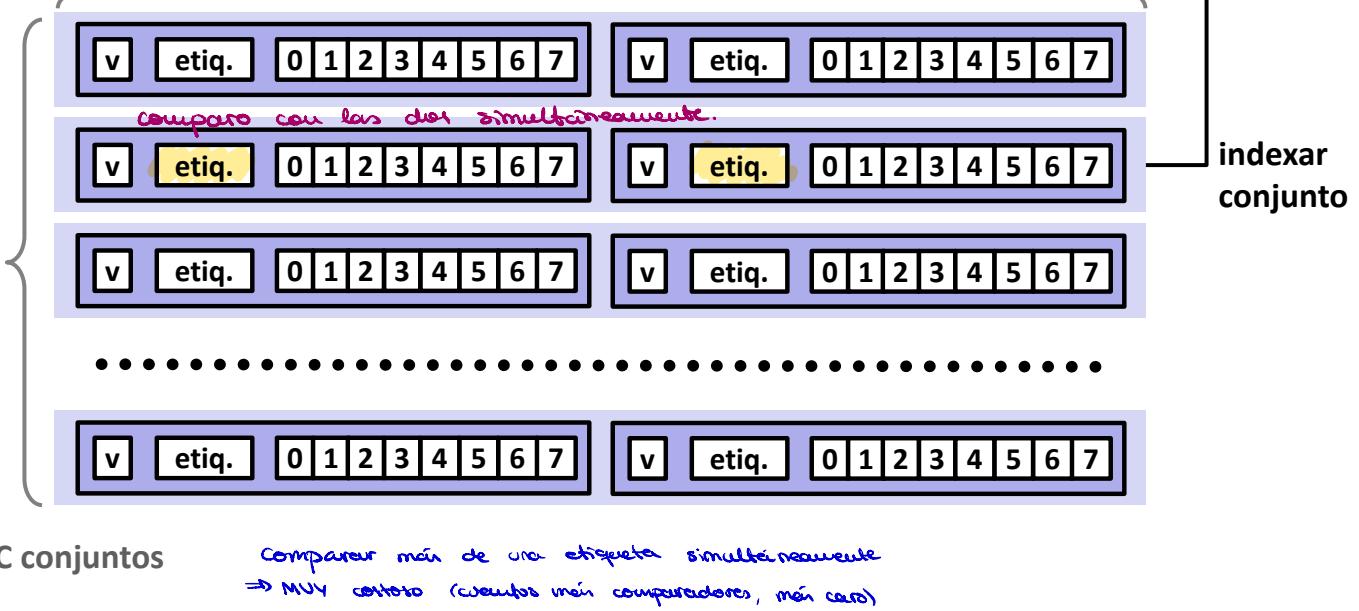
V = 2: Dos líneas por conjunto

Suponer: tamaño bloque cache B=8 bytes

Dirección de un short int:



2 líneas por conjunto



C conjuntos

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

20

Grado Informática, 2º Curso

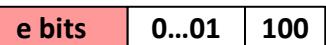
Estructura de Computadores

Cache Asociativa por Conjuntos de V vías (aquí V=2)

V = 2: Dos líneas por conjunto

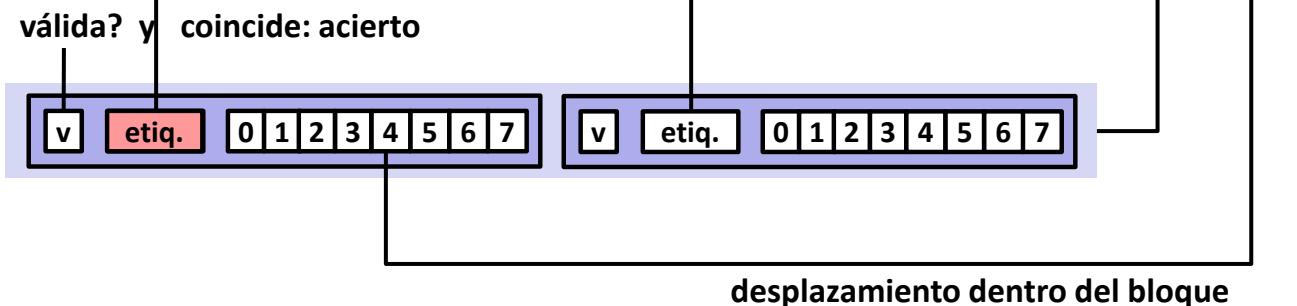
Suponer: tamaño bloque cache B=8 bytes

Dirección de un short int:



comparar ambas

válida? y coincide: acierto

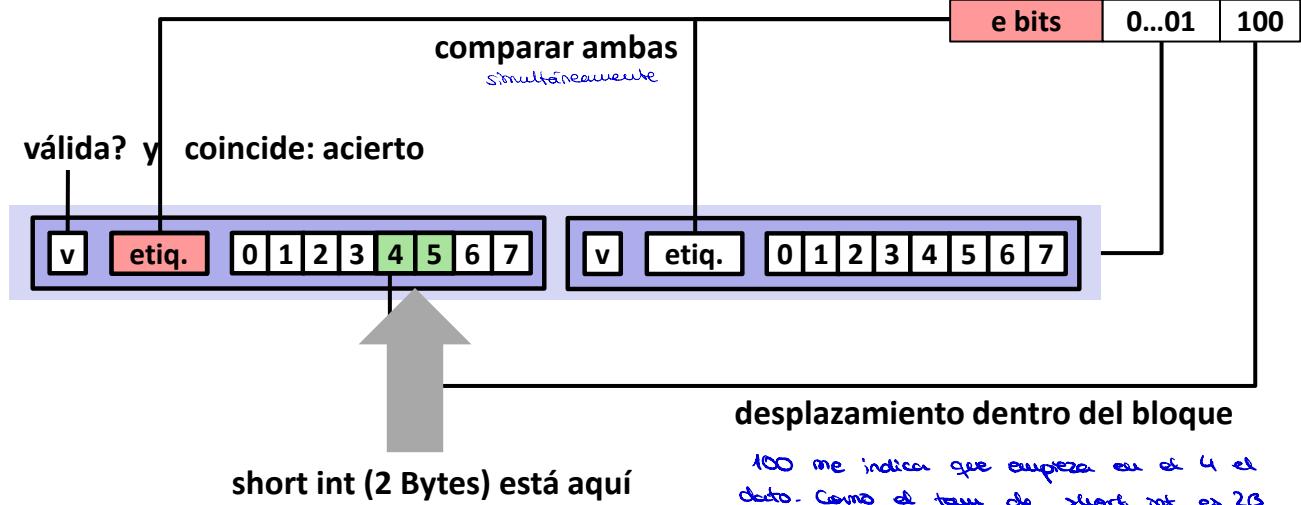


Cache Asociativa por Conjuntos de V vías (aquí V=2)

V = 2: Dos líneas por conjunto

Suponer: tamaño bloque cache B=8 bytes

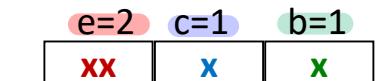
Dirección de un short int:



Si ninguna coincide o no válida (= fallo):

- Se escoge una línea del conjunto para desalojo y reemplazo
- Políticas de reemplazamiento: aleatoria, menos rec.uso (LRU), ...

Simulación cache asociativa conjuntos 2-vías



→ 4-1-1=2=c
direcciones 4 bits (M=16 bytes)

C=2 conjuntos, V=2 vías (bloq./conj.), B=2 bytes/bloque

2¹ conj → c=1

B=2⁰ B/bloq → b=1



Traza de direcciones (lecturas, un byte por lectura):

0	[0000] ₂ ,
1	[0001] ₂ ,
7	[0111] ₂ ,
8	[1000] ₂ ,
0	[0000] ₂

fallo → lo meto en conj 0 pq aun no hay nada (en frío)

acierto → lo saco de meter

fallo

fallo

lo meto en conj 0 ya lo tengo del fallo de antes!

acierto → como b=1, cada bloq. tiene que tener desplaz. 1

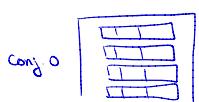
Conj. 0	Etiq.	Bloque
1	00	M[0-1]
1	10	M[8-9]

→ b=0: empieza desde 0
→ b=1: empieza desde 1

Conj. 1	Etiq.	Bloque
1	01	M[6-7]
0		

Simulación asociativa

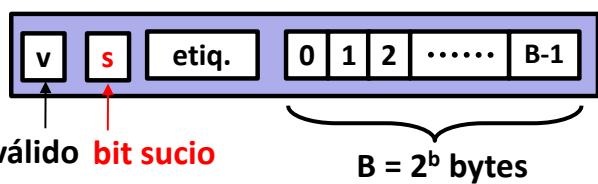
un sólo conjunto (4 bloques en nuestro ejemplo). Necesitamos 3 bits para etiqueta, 0 para c, y necesitaremos 4 comparadores hardware → es el trío correcto



¿Qué hay de las escrituras?

Múltiples copias de los datos:

- L1, L2, L3, Mem. principal, Disco



¿Qué hacer en acierto escritura?

- NUNCA pierde mucho tiempo si modifica un dato en la memoria*
- Write-through (escribir inmediatamente en la memoria) (Escritura directa)
 - Write-back (diferir escritura hasta reemplazo de la línea) (Escritura diferida)
 - Cada línea caché necesita un **bit sucio** (=1 si línea difiere de bloque M)
- =1 ⇒ se alteró esto respecto de lo que hay en MP*
- Si tengo que escribir de cache y llevarlo a MP, recorriendo sobre la pila de memoria con info actualizada*

¿Qué hacer en fallo escritura?

- Write-allocate (cargar y actualizar línea en cache) (Asignación en Escritura)
 - Conveniente si van a haber más escrituras a ese bloque
- No-write-allocate (escribir directo a memoria, sin cargar en cache)

Usual

- Write-through + No-write-allocate (Escritura directa sin Asignación en Escritura)
- Write-back + Write-allocate (Post-Escritura con Asignación en Escritura)

¿Por qué indexar con los bits intermedios?

Correspondencia directa: Una línea por conjunto

Suponer: tamaño bloque cache $B=8$ bytes

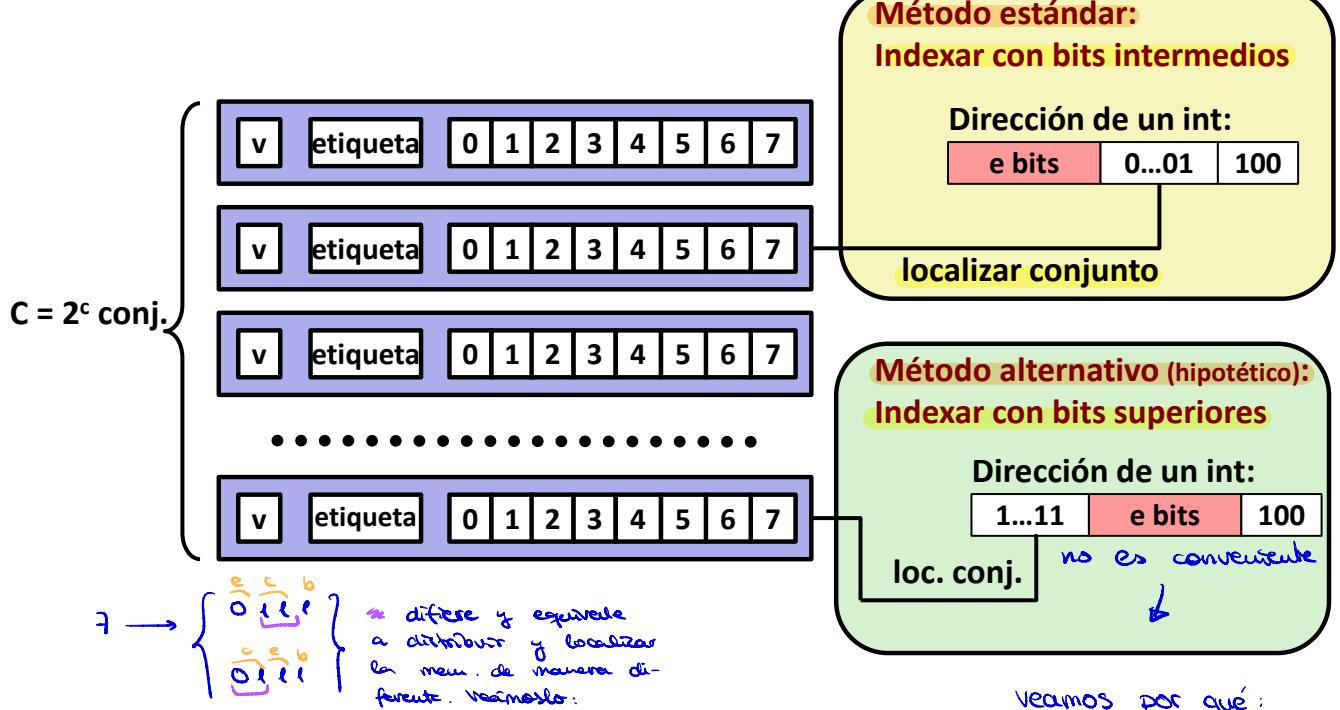
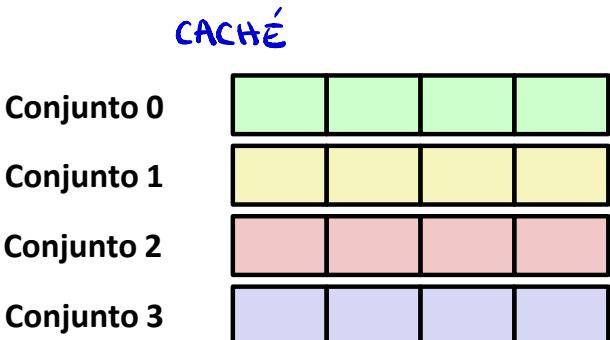


Ilustración de los métodos de indexación

- Memoria de 64 bytes
 - Direcciones de 6 bits
- Cache de 16B, corresp. directa
- T. bloque B=4 ($\Rightarrow C=4$ conj. ¿Por qué?)
- 2 bits etiq., 2 bits índ., 2 bits despl.



MEM PRINCIPAL			
0000xx			
0001xx			
0010xx			
0011xx			
0100xx			
0101xx			
0110xx			
0111xx			
1000xx			
1001xx			
1010xx			
1011xx			
1100xx			
1101xx			
1110xx			
1111xx			

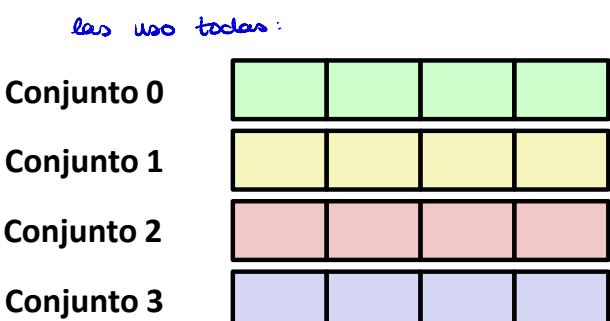
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

método extendido

Indexado c/bits intermedios

- Direcciones de la forma **TTSSBB**
 - **TT** bits etiqueta
 - **SS** bits índice conjunto
 - **BB** bits desplazamiento bloque
- Hace buen uso de localidad espacial

correspondencia directa: color con color



MEM PRINCIPAL			
0000xx			
0001xx			
0010xx			
0011xx			
0100xx			
0101xx			
0110xx			
0111xx			
1000xx			
1001xx			
1010xx			
1011xx			
1100xx			
1101xx			
1110xx			
1111xx			

método hipotético (malo)

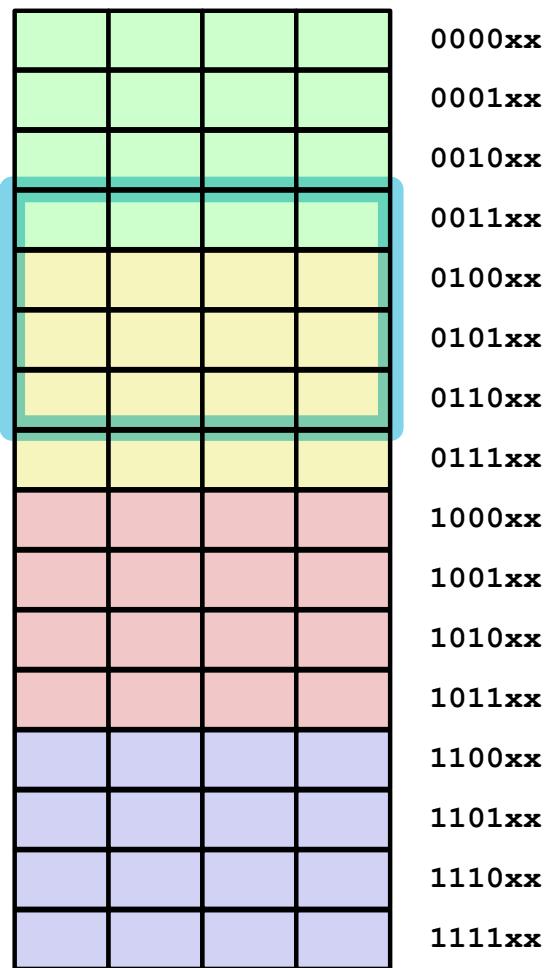
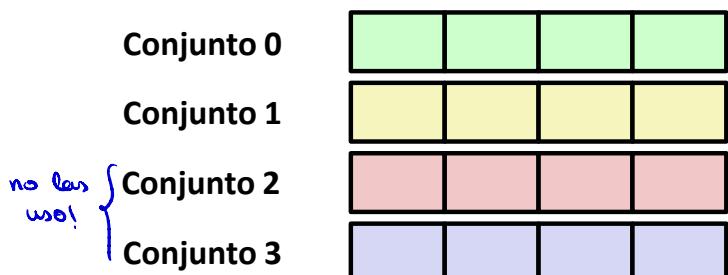
Indexado c/bits superiores

Direcciones de la forma SS_TT_BB_B

- **SS** bits índice conjunto
- **TT** bits etiqueta
- **BB** bits desplazamiento bloque

quiero
esta zona
me faltan
espacio para
2 anavistas

Programa con alta localidad espacial generaría muchos conflictos (fallos por conflicto)

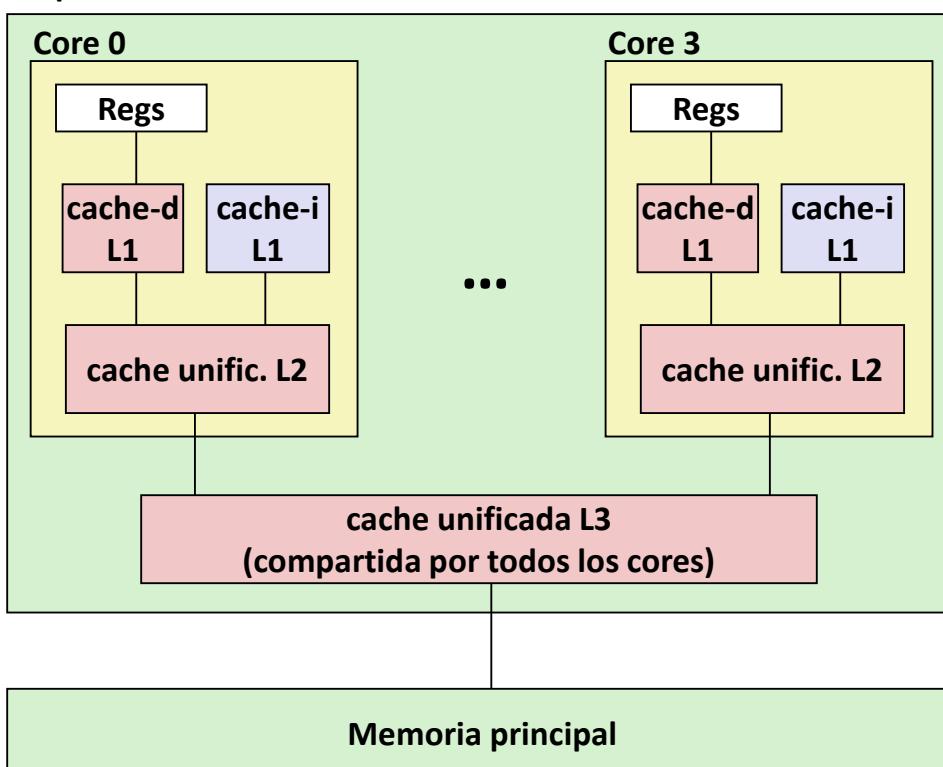


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

28

Jerarquía de caches del Intel Core i7

Paquete Procesador



cache-i y cache-d L1:

32 KB, 8-vías,
Acceso: 4 ciclos
(tarde pegatinas!)

cache unificada L2:

256 KB, 8-vías,
Acceso: 10 ciclos

cache unificada L3:

8 MB, 16-vías,
Acceso: 40-75 ciclos

Tamaño de bloque: 64 B en todas las caches

afecta a MC y MP porque tienen bloques del mismo tamaño.

L1 = desplazamiento: 6 bits $\rightarrow 2^6$ bloques
• T1s: 64 bytes $\rightarrow 2^{10}$ bytes
desplazamiento: $\frac{2^{10}}{2^6} = 2^4$ bloques
desplazamiento = 2⁴ = 16 bytes
... pero el propio Intel prefiere Paquete

MINI EJERCICIO

Ejemplo: cache de datos L1 del Core i7

$$\rightarrow 2^5 \cdot 2^{10} B = 2^{15} B$$

32 KB 8-vías (asoc.conj.)

64 bytes/bloque $2^6 \rightarrow b=6$

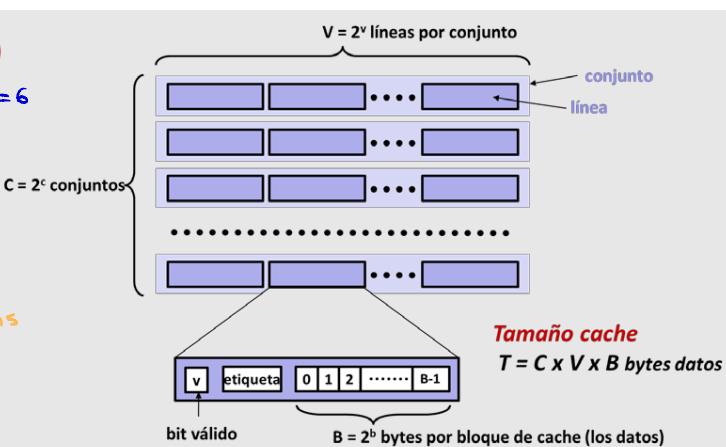
47 bit rango direcciones

$B = 64, \quad b = 6$

$V = 8, \quad v = 3$

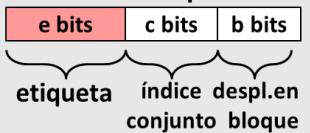
$C = 64, \quad c = 6 \quad]\text{Repasso}$

$T = C \cdot V \cdot B = 64 \cdot 8 \cdot 64 = 2^{15}$



	Hex	Decimal	Binary
0	0	0000	
1	1	0001	
2	2	0010	
3	3	0011	
4	4	0100	
5	5	0101	
6	6	0110	
7	7	0111	
8	8	1000	
9	9	1001	
A	10	1010	
B	11	1011	
C	12	1100	
D	13	1101	
E	14	1110	
F	15	1111	

Dirección inicio palabra:



despl. bloque: 6 bits

índice conj.: 6 bits

etiqueta: 35 bits

$47 - 6 - 6 = 35$

Dirección de Pila:

0x00007f7262a1e010



despl. bloque:

índice conj.:

etiqueta:

0x01
0x??
0x0
0x??
0x??
0x7f7262a1e

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

30

Ejemplo: cache de datos L1 del Core i7

32 KB 8-vías (asoc.conj.)

64 bytes/bloque

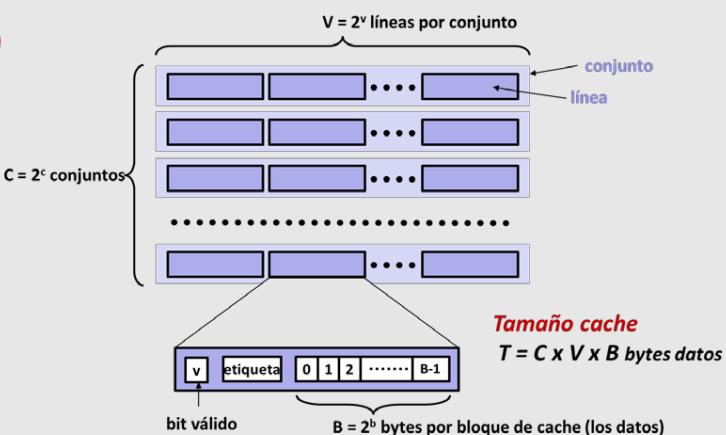
47 bit rango direcciones

$B = 64, \quad b = 6$

$V = 8, \quad v = 3$

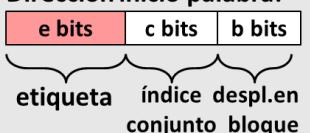
$C = 64, \quad c = 6$

$T = 64 \times 64 \times 8 = 32,768$



	Hex	Decimal	Binary
0	0	0000	
1	1	0001	
2	2	0010	
3	3	0011	
4	4	0100	
5	5	0101	
6	6	0110	
7	7	0111	
8	8	1000	
9	9	1001	
A	10	1010	
B	11	1011	
C	12	1100	
D	13	1101	
E	14	1110	
F	15	1111	

Dirección inicio palabra:



despl. bloque: 6 bits

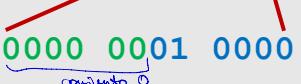
índice conj.: 6 bits

etiqueta: 35 bits

Dirección de Pila:

0x00007f7262a1e010

número extra etiqueta en los 8 bloques del conjunto 0.



despl. bloque: 0x10

índice conj.: 0x0

etiqueta: 0x7f7262a1e

Cache: resumen de políticas de colocación

Organización (C,V,B,m)

- La caché tiene $2^c \times 2^v = 2^{c+v}$ líneas. Un bloque tiene 2^b bytes. Una dirección física tiene m bits. La MP tiene 2^m bytes (2^{m-b} bloques).

Correspondencia directa

- Bloque i de MP \Rightarrow línea $i \bmod 2^l$ de cache (L líneas = $2^l = 2^{c+v} = 2^c$ con $v=0$)

1 bloque / conjunto
- recibiendo un sólo comparador pq compara en un sólo lugar
- es la más barata

Correspondencia totalmente asociativa

- Bloque i de MP \Rightarrow cualquier línea de cache

una locura

1 comparador / bloque
- direcciones más largas

Correspondencia asociativa por conjuntos

- Bloque i de MP \Rightarrow conjunto $i \bmod 2^c$ de cache (cualquier línea del conjunto)

intermedio
varios bloques / conjunto. varios comparadores

2 pasos:
 \rightarrow conjunto
 \rightarrow sitio (tot. asociativo)

Consideraremos el "Ejemplo 1":

- Tamaño de caché: 2K bytes. 2^b Bytes $\Rightarrow c+v+b=11$
- 16 bytes por bloque. 2^b Bytes $\sim b=4 \Rightarrow b=4 \Rightarrow c+v=7$, **128 líneas en cache**
- Memoria principal máx: 256K bytes. $\Rightarrow m=18$, **16K bloques en MP**
- $(CxV=128, B=16, m=18)$, $c+v=7, b=4, c+v+b=11, CxVxB=2K$

• Ancho del address bus: 256KB $\rightarrow 2^9 b \rightarrow 16$ bits ancho de bus
• bloques en la caché? $256/16B = 2^{10} 2^4 B/16B \rightarrow 128$ bloques

32

Ejemplo 1 de política de colocación

Cache: política de colocación

la más simple a nivel hardware
la más barata

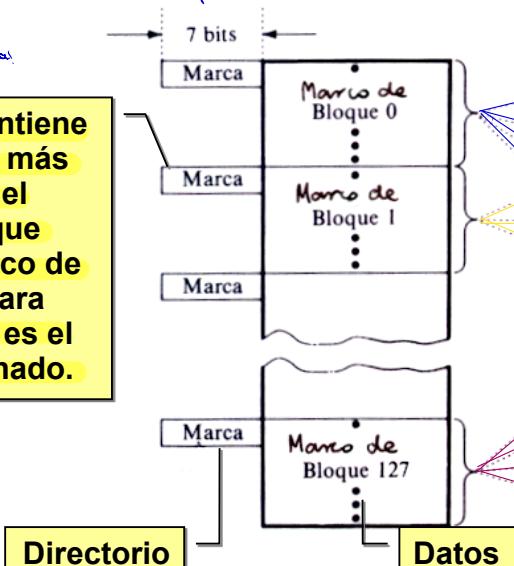
Correspondencia directa

- Bloque i de MP \Rightarrow línea $i \bmod 2^l$ de caché.
- A cada línea le corresponde sólo un subconjunto de bloques de MP.

ejemplo a este resultado
pido 0 \rightarrow falló
pido 128 \rightarrow falló
pido 0 \rightarrow falló
pido 128 \rightarrow falló
y la caché está prácticamente vacía!

Cada marca contiene los $m-(l+b)$ bits más significativos del bloque de MP que hay en ese marco de bloque. Sirve para identificar cuál es el bloque almacenado.

De cada marco, los últimos b bits son iguales para todos los bloques $2^b = 128$



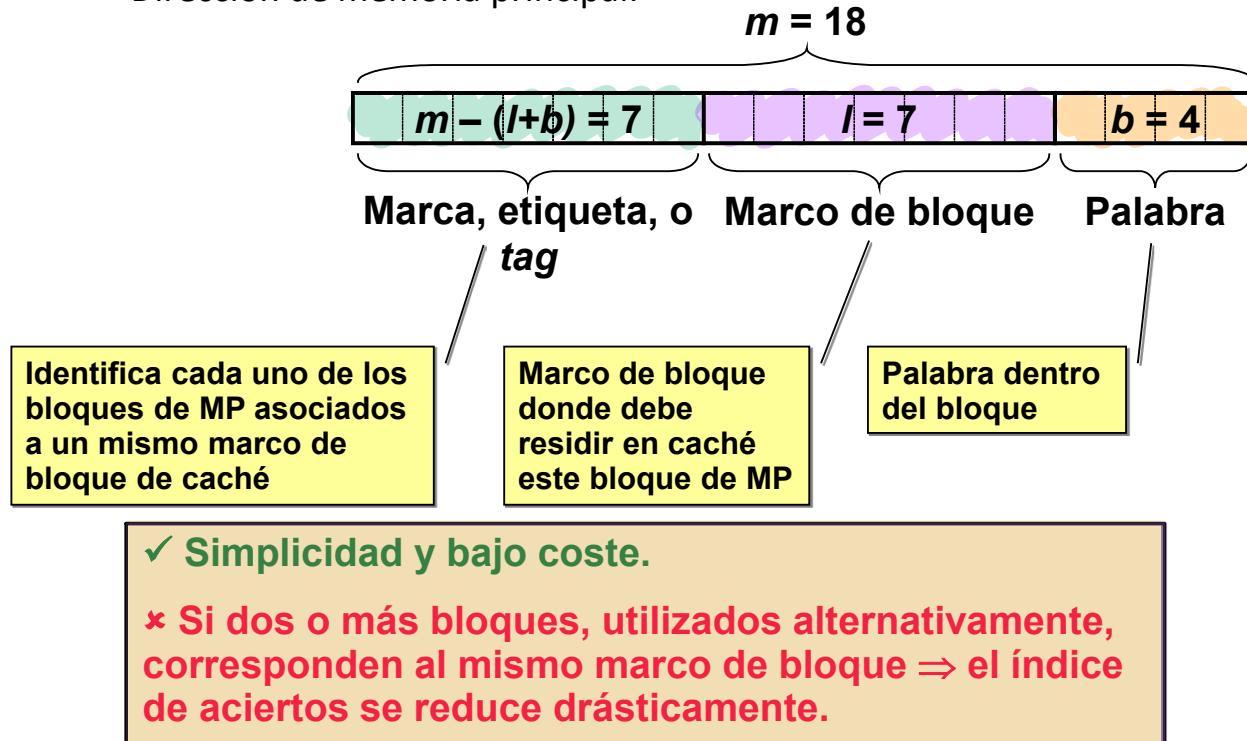
Ejemplo 1

Memoria principal
Bloque 0
Bloque 1
Bloque 2
⋮
Bloque 127
Bloque 128
Bloque 129
⋮
Bloque 255
Bloque 256
Bloque 257
⋮
Bloque 4095
Bloque 4096
⋮
Bloque 16383

33

Cache: política de colocación

- Dirección de memoria principal:



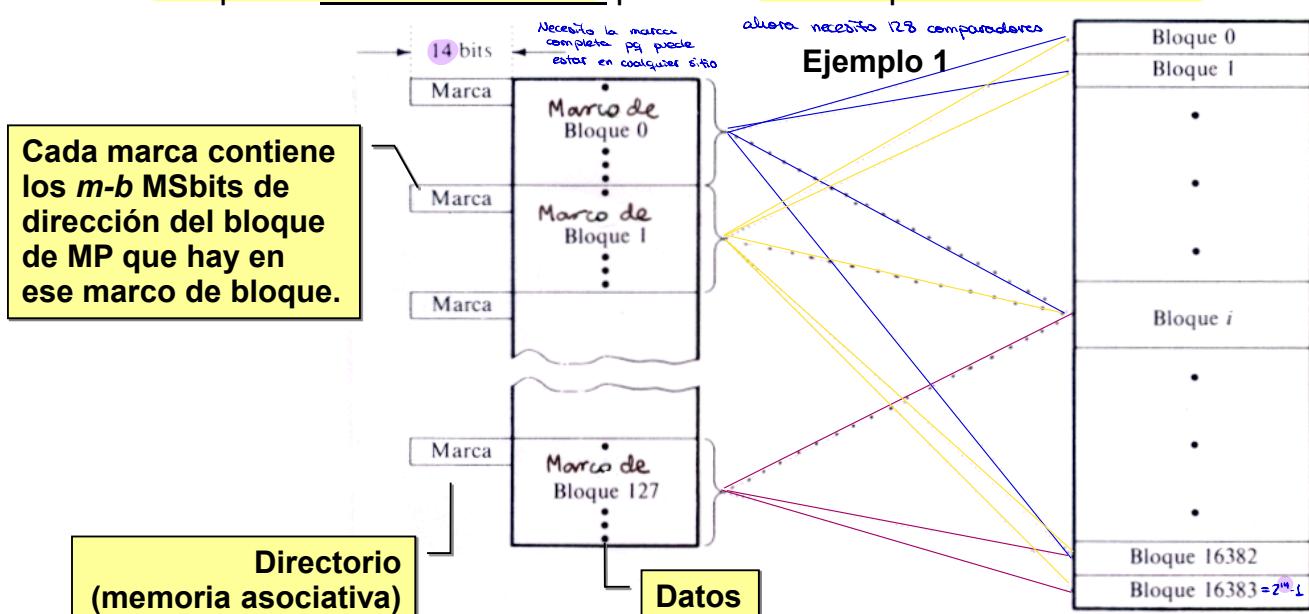
34

Cache: política de colocación

| la más cara \rightarrow tanto que no se hace
| la más flexible (cualquier combinación de datos)

Correspondencia totalmente asociativa

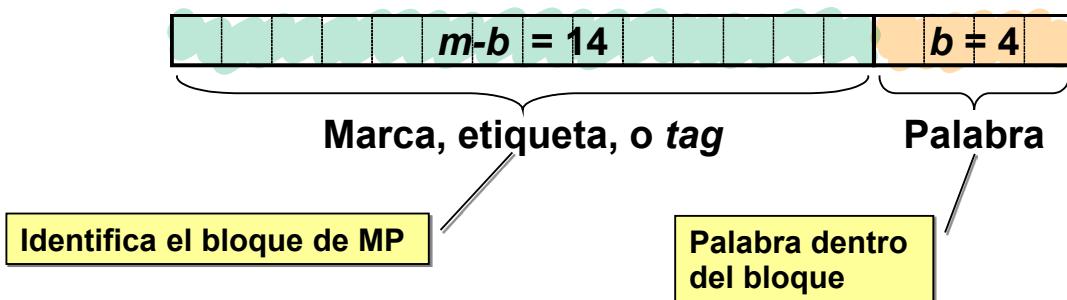
- Un bloque de MP puede residir en cualquier marco de bloque de caché.
- Cuando se presenta una solicitud a la caché, todas las marcas se comparan simultáneamente para ver si el bloque está en la caché.



35

Cache: política de colocación

- Dirección de memoria principal:



✓ **Flexible.** Permite cualquier combinación de bloques de MP en la caché. Elimina en gran medida conflictos entre bloques.

✗ **Compleja y costosa de implementar (por la memoria asociativa).**

36

Cache: política de colocación

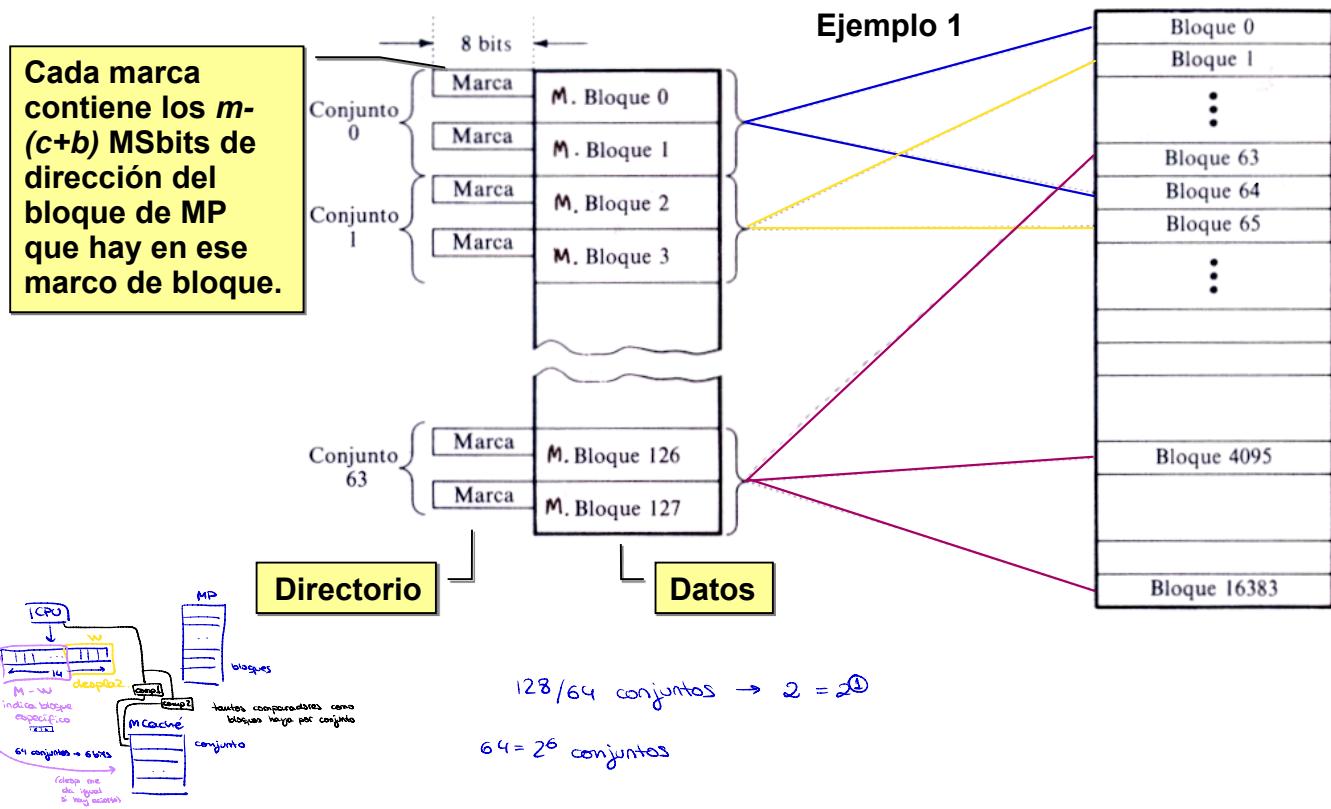
■ Correspondencia asociativa por conjuntos

meet me halfway

- La caché se subdivide en 2^c conjuntos disjuntos
 - 2^v marcos de bloque / conjunto
- Bloque i de MP \Rightarrow conjunto $i \bmod 2^c$ de caché. Dentro de ese conjunto puede estar en cualquier marco de bloque.
- Hay dos fases en el acceso a caché:
 - Selección directa del conjunto donde puede estar ese bloque.
 - Búsqueda asociativa (dentro del conjunto) de la marca.
- A la correspondencia asociativa por conjuntos con 2^v marcos de bloque / conjunto también se le llama **correspondencia asociativa de 2^v vías**.
 - La vía i está formada por todos los marcos de bloque de la caché que ocupan el lugar i -ésimo dentro de su conjunto.
 - Completar enunciado del Ejemplo 1 fijando $v = 1, 2^v = 2$ vías.

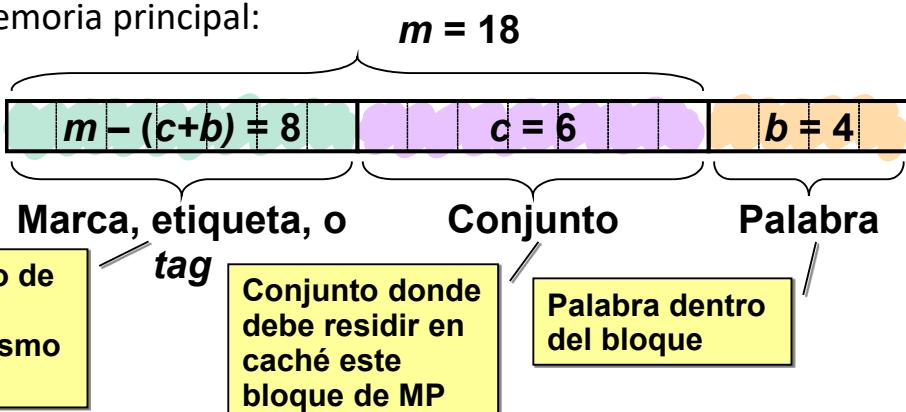
37

Cache: política de colocación



Cache: política de colocación

- Dirección de memoria principal:



$c = 0$ (1 conjunto) \Rightarrow corresp. totalmente asociativa.

$c = n$ (1 marco de bloque / conjunto) \Rightarrow corresp. directa.

$0 < c < n \Rightarrow$ se pretende reducir el **coste de la totalmente asociativa** manteniendo un **rendimiento similar** \Rightarrow es la técnica más utilizada.

✓ Resultados experimentales demuestran que un tamaño de conjunto de 2 a 16 marcos de bloque funciona casi tan bien como una corresp. totalmente asociativa con un incremento de coste pequeño respecto de la corresp. directa.

Métricas para prestaciones de cache



- **Tasa de Fallo**
 - Fracción de referencias a memoria no encontradas en caché (fallos / accesos) = $1 - \text{tasa de acierto}$
 - Valores típicos (en porcentaje):
 - 3-10% para L1
 - puede ser bastante pequeño (por ejemplo, <1%) para L2, dependiendo del tamaño, etc.
- **Tiempo en Acierto[†]** (tiempo de acceso en caso de acierto)
 - Tiempo para entregar una línea de cache al procesador
 - incluye el tiempo para determinar si la línea está en cache
 - Valores típicos :
 - 4 ciclos reloj para L1
 - 10 ciclos reloj para L2
- **Penalización por Fallo**
 - Tiempo adicional requerido debido a un fallo
 - típicamente 50-200 ciclos para M.principal (Tendencia: aumentando!)

Reflexionemos sobre esos valores

- **Enorme diferencia entre acierto y fallo**
 - Podría llegar a 100x, si solo L1 y Memoria principal
- **¿Verosímil que 99% acierto es doble de bueno que 97%?**
 - Considerar este ejemplo simplificado:
 - tiempo en acierto cache de 1 ciclo
 - penalización por fallo de 100 ciclos
 - Tiempo medio de acceso:
 - 97% aciertos: 1 ciclo + 0.03×100 ciclos = **4 ciclos**
 - 99% aciertos: 1 ciclo + 0.01×100 ciclos = **2 ciclos**
- **Por eso se usa “tasa de fallo” en lugar de “tasa de acierto”**

Memoria II: Cache

- Organización y Funcionamiento de la memoria cache
- Impacto de la cache en el rendimiento
 - Modelo de evaluación
 - La montaña de memoria
- Programación de código aprovechando la cache

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

43

Grado Informática, 2º Curso

Estructura de Computadores

Jerarquía de memoria

- Parámetros que caracterizan cada nivel i
 - Los dispositivos de almacenamiento (registros, memorias, discos y unidades de cinta), se caracterizan por:
 - Tiempo de acceso (t_i):
 - Tiempo desde que se inicia una lectura hasta que llega la palabra deseada.
 - Tamaño de la memoria (s_i):
 - Número de bytes, palabras, sectores, etc., que se pueden almacenar en el dispositivo de memoria.
 - Coste por bit o por byte (c_i). €
 - Ancho de banda (b_i):
 - Velocidad a la que se transfiere información desde un dispositivo.
 - Unidad de transferencia (x_i):
 - Tamaño de la unidad de información que se transfiere entre el nivel i y el $i+1$.

Se verifica que:

$$t_i < t_{i+1}$$

$$s_i < s_{i+1}$$

$$c_i > c_{i+1}$$

$$b_i > b_{i+1}$$

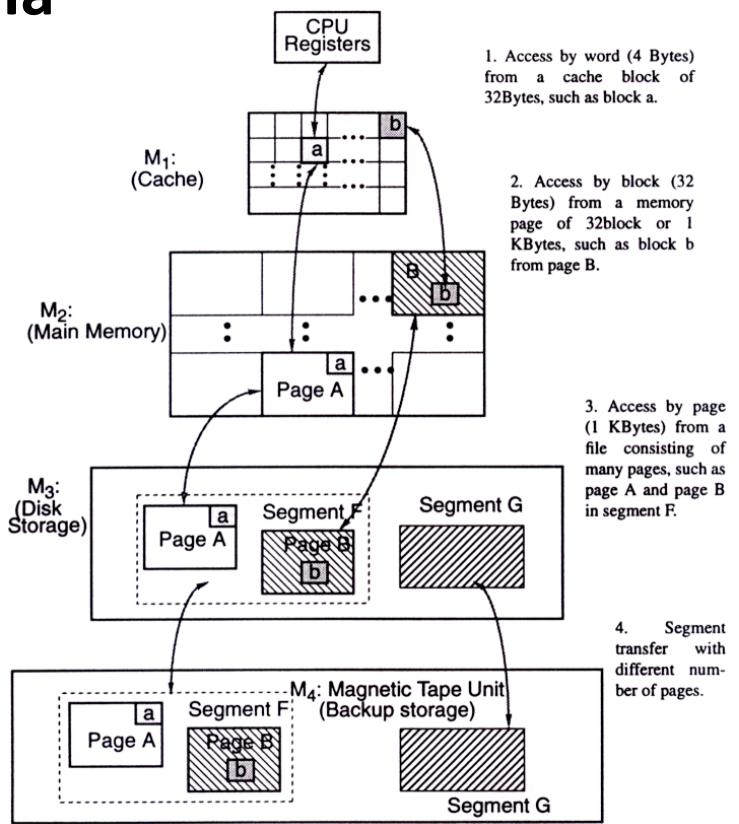
$$x_i < x_{i+1}$$

todo lo que hay en memoria es igual a lo que voy en mp. \Rightarrow cuando quite un bloque modificado de cache, modifica la pcp

Jerarquía de memoria

■ Propiedad de inclusión

- $M_1 \subset M_2 \subset M_3 \subset \dots \subset M_n$
- Si una palabra se encuentra en $M_i \Rightarrow$ copias de esa palabra también se encuentran en $M_{i+1}, M_{i+2}, \dots, M_n$.
- Sin embargo, una palabra almacenada en M_{i+1} puede no estar en M_i , y si es así, tampoco estará en $M_{i-1}, M_{i-2}, \dots, M_1$.



Modelo de evaluación de la jerarquía

■ Modelo de evaluación del rendimiento de una jerarquía de memoria (C. K. Chow, 1974*)

- Se asume que caches son inclusivas ($\sum_{i=1}^j a_i = A_j$)
 - y que compilador no reutiliza registros ($A_0 = 0$)
- Tasa de aciertos A_i (hit ratio)
 - Porcentaje de información buscada que está en el nivel i .
 - En una jerarquía con n niveles:
 - $A_0 = 0$
 - $A_n = 1$
 - A_i depende de:
 - Capacidad del nivel i (s_i).
 - Granularidad de la transferencia de información.
 - Estrategia de administración de memoria.

Modelo de evaluación de la jerarquía

- Tasa de fallos F_i : (miss ratio)
 - $F_i = 1 - A_i, i=0, \dots, n.$
 - $F_0 = 1$
 - $F_n = 0$
- Tasa de aciertos específica del nivel i a_i :
 - Frecuencia de accesos con *primer éxito* al nivel i
 - Probabilidad de acceder con éxito a una información en nivel i y que esa información no se encuentre en los niveles 0 a $i-1$.
$$\sum_{i=1}^n a_i = 1$$
 - Dado que la información de M_j está también en $M_k, k > j$:
 - $a_i = A_i - A_{i-1}, i = 1, \dots, n$
 - $a_1 = A_1$
 - $a_n = 1 - A_{n-1}$

47

Modelo de evaluación de la jerarquía

- Los objetivos al diseñar una memoria con n niveles son:
 - ① Obtener un rendimiento cercano al de la memoria M_1 (la más rápida).
 - ② Obtener un coste por bit cercano al de la memoria M_n (la más barata).

① Rendimiento:

- Cuantificable con el tiempo medio de acceso a la jerarquía (\bar{T}).

Tiempo de acceso efectivo al nivel i -ésimo de la jerarquía (T_i):

$$T_i = \sum_{j=1}^i t_j$$

*es lo que me interesa
(tiempo de acceso efectivo)*

t_j = tiempo de acceso específico del nivel j

$\bar{T} = \sum_{i=1}^n a_i T_i$

Sumatoria de la tasa de acierto específica de M_i , multiplicada por el tiempo de acceso efectivo a M_i

48

Modelo de evaluación de la jerarquía

$$\bar{T} = \sum_{i=1}^n a_i T_i \quad a_i = A_i - A_{i-1}$$

■ Sustituyendo a_i y T_i :

$$\begin{aligned} \bar{T} &= \sum_{i=1}^n \left[(A_i - A_{i-1}) \sum_{j=1}^i t_j \right] = (A_1 - A_0)t_1 + (A_2 - A_1)(t_1 + t_2) + (A_3 - A_2)(t_1 + t_2 + t_3) + \\ &\quad + \dots + (A_n - A_{n-1})(t_1 + t_2 + t_3 + \dots + t_n) = \\ &= (A_1 - A_0 + A_2 - A_1 + A_3 - A_2 + \dots + A_n - A_{n-1})t_1 + \\ &\quad + (A_2 - A_1 + A_3 - A_2 + \dots + A_n - A_{n-1})t_2 + \\ &\quad + (A_3 - A_2 + \dots + A_n - A_{n-1})t_3 + \\ &\quad + \dots + (A_n - A_{n-1})t_n = \\ &= \sum_{i=1}^n (A_n - A_{i-1})t_i = \sum_{i=1}^n (1 - A_{i-1})t_i = \\ &= \sum_{i=1}^n F_{i-1}t_i \end{aligned}$$

49

Modelo de evaluación de la jerarquía

■ ②Coste por bit:

- El coste por bit promedio $c(n)$ de un sistema de n niveles es:

$$c(n) = \frac{\sum_{i=1}^n c_i s_i}{\sum_{i=1}^n s_i} + c_0$$

coste total
coste de interconexión entre niveles
tamaño total

Cache: modelo de evaluación

■ Modelo aplicado a un sistema de solo 2 niveles (memoria cache y memoria principal).

- $t_1 = t_c$: tiempo de acceso de la memoria cache (específico).
- $a_1 = A_1 = A$: tasa de acierto (hit ratio) de la memoria cache.
- $t_2 = t_m$: tiempo de acceso a la memoria principal (específico).
- Tiempo medio de acceso:

$$\bar{T} = At_c + (1 - A)(t_c + t_m)$$

veces que voy a caché · tiempo de acceso mc prob de que no esté en caché · tiempo acceso mc + mp


La CPU pide simultáneamente a la MP y a la MC un dato (lo pide por bus), entonces nos quedamos con la fórmula:
 $T_2 = At_c + (1-A)t_m$

Acierto \Rightarrow no se accede a MP

Fallo \Rightarrow se accede a caché y a MP

- γ : razón entre los tiempos de acceso a la MP y a la cache

$$\gamma = \frac{t_m}{t_c}$$

51

Caché: modelo de evaluación

- Eficiencia de un sistema que emplea memoria cache:

$$E = \frac{t_c}{\bar{T}}, \quad 0 < E \leq 1$$

$$E = \frac{t_c}{At_c + (1 - A)(t_c + t_m)} = \frac{1}{A + (1 - A)\left(1 + \frac{t_m}{t_c}\right)} =$$

$$= \frac{1}{A + (1 - A)(1 + \gamma)} = \frac{1}{A + 1 - A + \gamma(1 - A)} = \frac{1}{1 + \gamma(1 - A)}$$

- E máxima cuando A=1 (todas las referencias en cache).

▪ $\gamma \uparrow \Rightarrow E \downarrow$; $A \downarrow \Rightarrow E \downarrow$. Interesa γ baja y A alta.

- Ejemplo:

$$\bullet t_c = 15 \text{ ns} \quad \bar{T} = At_c + (1 - A)(t_c + t_m) = 0,9 \cdot 15 + 0,1 \cdot 115 = 25 \text{ ns}$$

$$\bullet t_m = 100 \text{ ns} \quad \gamma = \frac{t_m}{t_c} = \frac{100}{15} = 6,6 \quad E = \frac{1}{1 + \gamma(1 - A)} = \frac{1}{1 + 6,6 \cdot 0,1} = 0,6$$

52

Caché separada / unificada

■ Caches separadas de datos / instrucciones

- Es común particionar la caché en dos módulos diferentes:
 - **Caché de datos.**
 - La **localidad** de los **datos** no es tan buena como la de las **instrucciones** ⇒ la caché de datos es **menos eficiente**.
 - Es **más compleja** por la posibilidad de **modificación** de los **datos**.
⇒ Muchos sistemas no admiten caché de datos. lectura/escritura
 - **Caché de instrucciones.**
 - Es fácil que bucles y pequeñas rutinas entren totalmente en caché, permitiendo su ejecución sin necesidad de acceder a MP.
 - Se **simplifica la caché**, ya que es habitual que se prohíba escribir en las localizaciones donde se encuentran las instrucciones ⇒ **caché de sólo lectura**.
- ✓ Se pueden emitir direcciones de instrucción y dato a la vez, doblando el ancho de banda entre caché y procesador.
- ✓ Se puede optimizar cada caché por separado:
 - Diferentes capacidades, tamaños de bloque, asociatividades, etc.

53

Caché separada / unificada

→ algo más costosa

Las caches de instrucciones tienen menor frecuencia de fallos que las de datos.

¿Cuál tiene una frecuencia de fallos menor: una caché de instrucciones de 16 KB + una caché de datos de 16 KB o una caché unificada de 32 KB?

Frecuencia de fallos para la caché particionada:

$$53\% \cdot 3,6\% + 47\% \cdot 5,3\% = 4,4\%$$

Una caché unificada de 32 KB tiene una frecuencia de fallos similar (4,3%).

Tamaño	Instrucción sólo	Sólo datos	Unificada
0,25 KB	22,2 %	26,8 %	28,6 %
0,50 KB	17,9 %	20,9 %	23,9 %
1 KB	14,3 %	16,0 %	19,0 %
2 KB	11,6 %	11,8 %	14,9 %
4 KB	8,6 %	8,7 %	11,2 %
8 KB	5,8 %	6,8 %	8,3 %
16 KB	3,6 %	5,3 %	5,9 %
32 KB	2,2 %	4,0 %	4,3 %
64 KB	1,4 %	2,8 %	2,9 %
128 KB	1,0 %	2,1 %	1,9 %
256 KB	0,9 %	1,9 %	1,6 %

Frecuencia de fallos para caches de distintos tamaños de sólo datos, sólo instrucciones, y unificadas. Los datos son para una cache asociativa de 2 vías utilizando reemplazo LRU con bloques de 16 bytes para un promedio de trazas de usuario/sistema en la VAX-11 y trazas de sistema en el IBM 370 [Hill 1987]. El porcentaje de referencias a instrucciones en estas trazas es aproximadamente del 53 por 100.

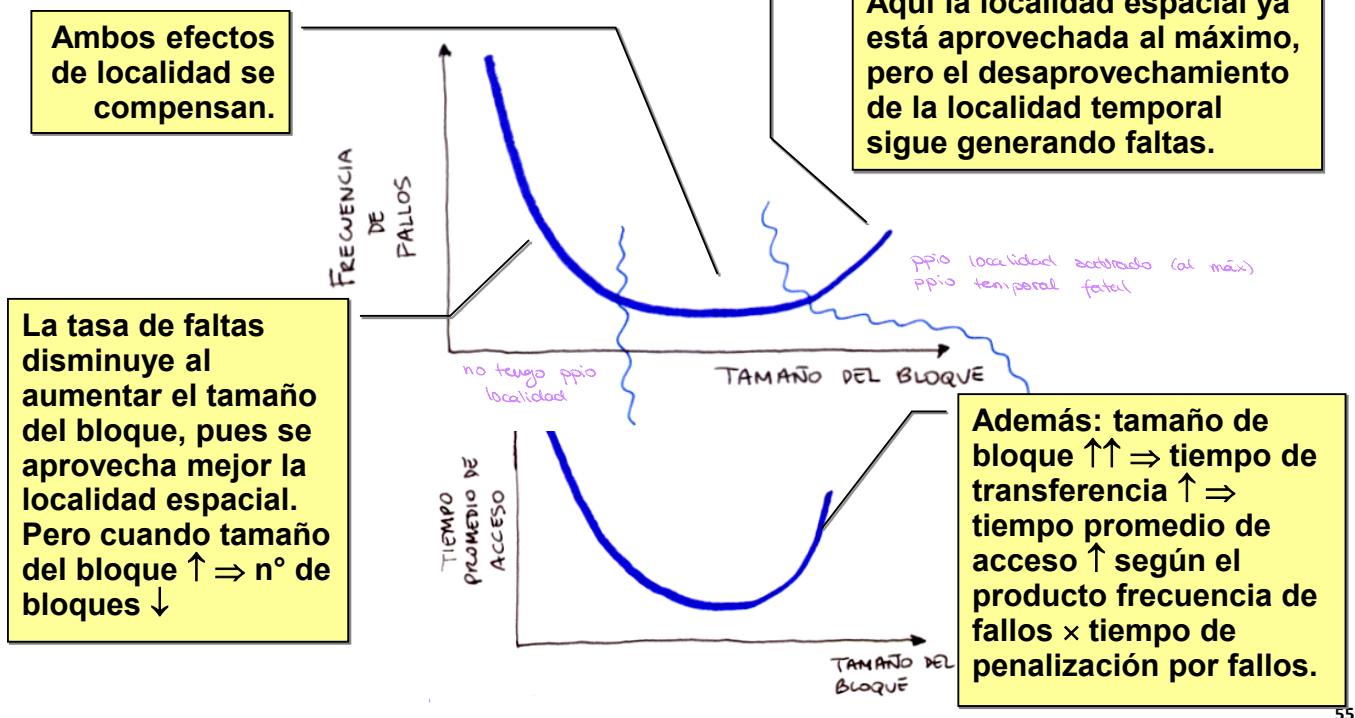
(Ver, sin embargo, las ventajas de la transparencia anterior).

54

Caché: tamaño

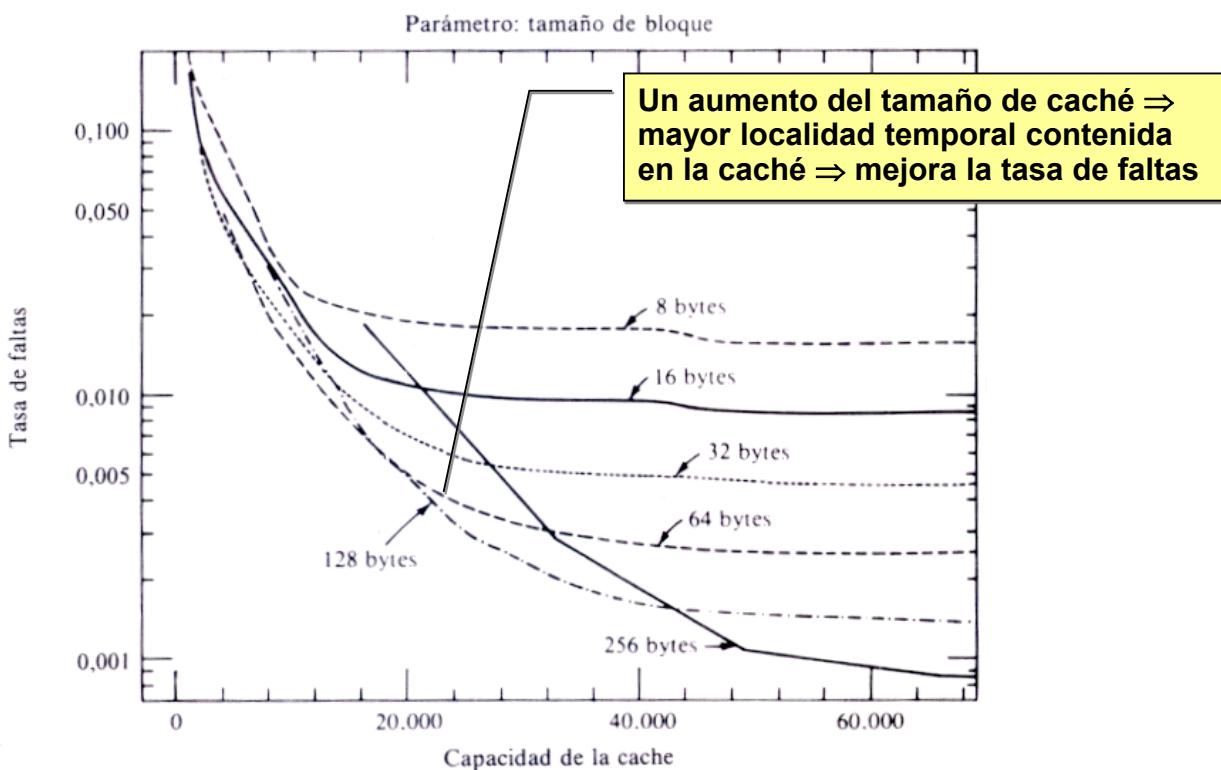
■ Tamaños del bloque y de la caché

- Para un tamaño de caché fijo:



Caché: tamaño

- Para un tamaño de bloque fijo:



Memoria II: Cache

- Organización y Funcionamiento de la memoria cache
- Impacto de la cache en el rendimiento
 - Modelo de evaluación
 - La montaña de memoria
- Programación de código aprovechando la cache

La montaña de memoria

- **Rendimiento de lectura** (ancho de banda de lectura)
 - Número de bytes leídos de memoria por segundo (MB / s)
- **Montaña de memoria:** rendimiento de lectura medido en función de la localidad espacial y temporal.
 - Manera compacta de caracterizar el rendimiento del sistema de memoria.

Función test para montaña memoria

```

long data[MAXELEMS]; /* Array global a recorrer */

/* test - Iterar sobre los primeros "elems" elementos
 *         del array "data" con paso de "stride",
 *         usando desenrollado de bucles x4
 */
int test(int elems, int stride) {
    long i, sx2=stride*2, sx3=stride*3, sx4=stride*4;
    long acc0 = 0, acc1 = 0, acc2 = 0, acc3 = 0;
    long length = elems, limit = length - sx4;

    /* Combinar 4 elementos a la vez */
    for (i = 0; i < limit; i += sx4) {
        acc0 = acc0 + data[i];
        acc1 = acc1 + data[i+stride];
        acc2 = acc2 + data[i+sx2];
        acc3 = acc3 + data[i+sx3];
    }

    /* Terminar con los elementos restantes */
    for (; i < length; i++) {
        acc0 = acc0 + data[i];
    }
    return ((acc0 + acc1) + (acc2 + acc3));
}

```

mountain/mountain.c

Llamar a `test()` con muchas combinaciones de `elems` y `stride`

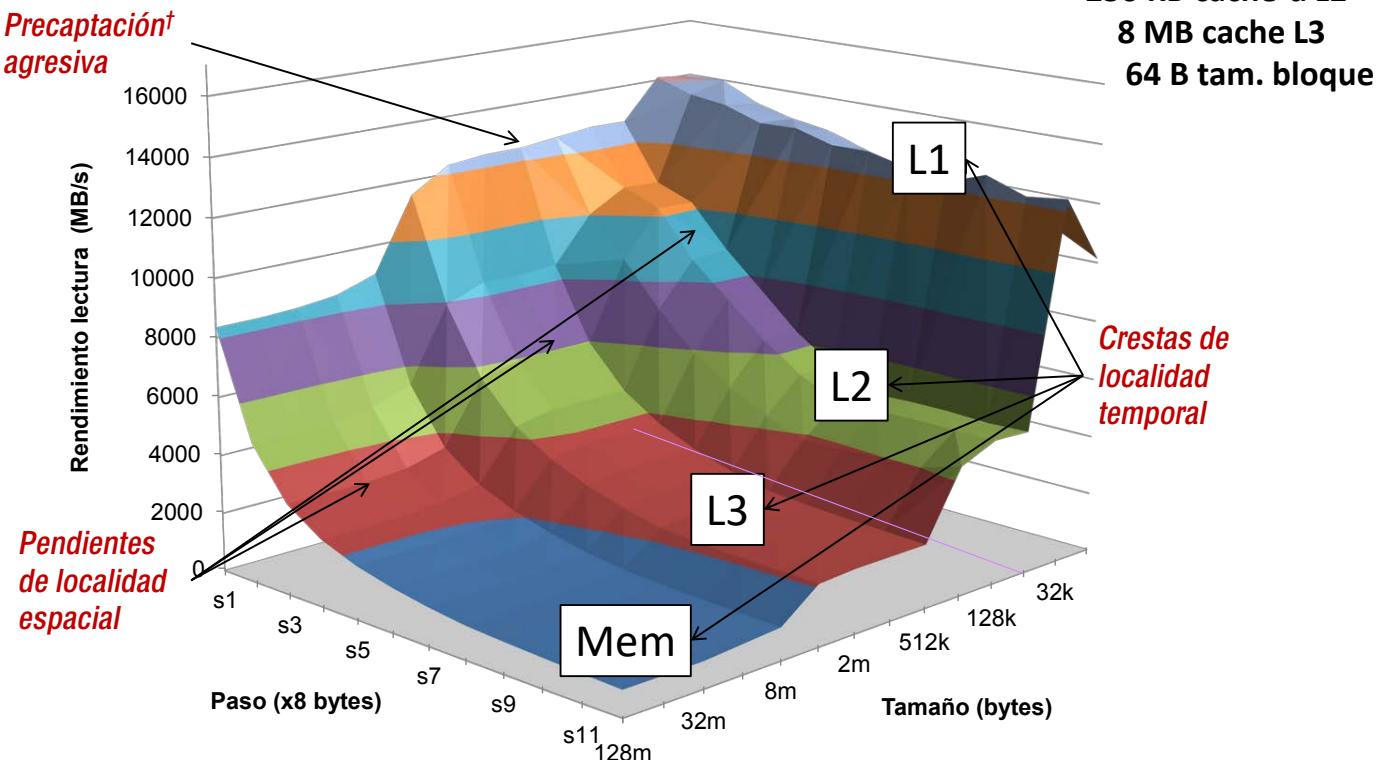
Para cada `elems` y `stride`:

1. Llamar una vez a `test()` para calentar las caches

2. Llamar a `test()` otra vez y medir el rendimiento de lectura (MB/s)

[†] for (...; i<=limit; ...)

La Montaña de Memoria



Memoria II: Cache

NO ENTRA LO DE ABASO

- Organización y Funcionamiento de la memoria cache
- Impacto de la cache en el rendimiento
 - La montaña de memoria
- Programación de código aprovechando la cache

Ejercicios interesantes:

- Datos de mem ←
si MC es asociativa x conjuntos dtaun etiqueta?