



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza



AOC2 Segunda Parte – Jerarquía de Memoria

1. Principios de funcionamiento de la jerarquía de memoria
2. Organización de una cache: Correspondencia
3. Tamaño de bloque y reemplazo
4. Escrituras
5. Control - comportamiento
6. Caches multinivel
7. Memoria Principal



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza



Tema 5 – Principios de Funcionamiento de la Jerarquía de Memoria

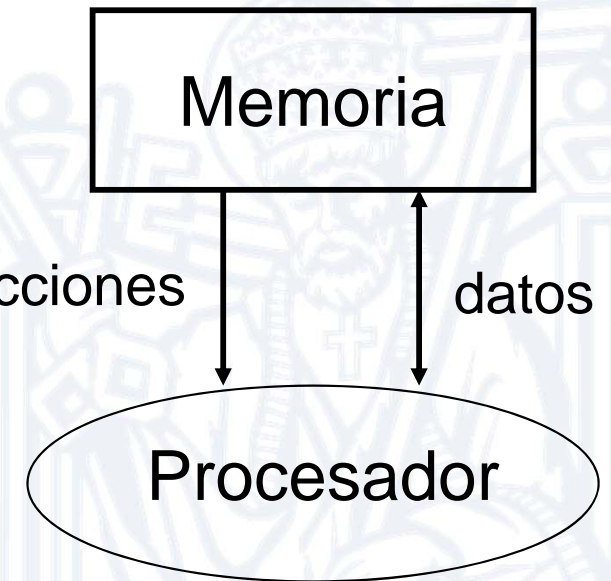
P. Ibáñez, J.L. Briz, V. Viñals, J. Alastruey, J. Resano
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

Objetivos del tema

- Motivar la existencia de la jerarquía de memoria
- Conocer los principios en que se basa la organización jerárquica
- Describir la estructura y el funcionamiento básico de una memoria cache

Guión del tema

- **Motivación: velocidad procesador vs. memoria**
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- Jerarquía de memoria
 - Memoria cache
- Ejemplos de memoria cache
- Descripción de una cache



Tiempo de ciclo del procesador

- Tabla de frecuencias y tiempo de ciclo de varios procesadores entre 1986 y 2001

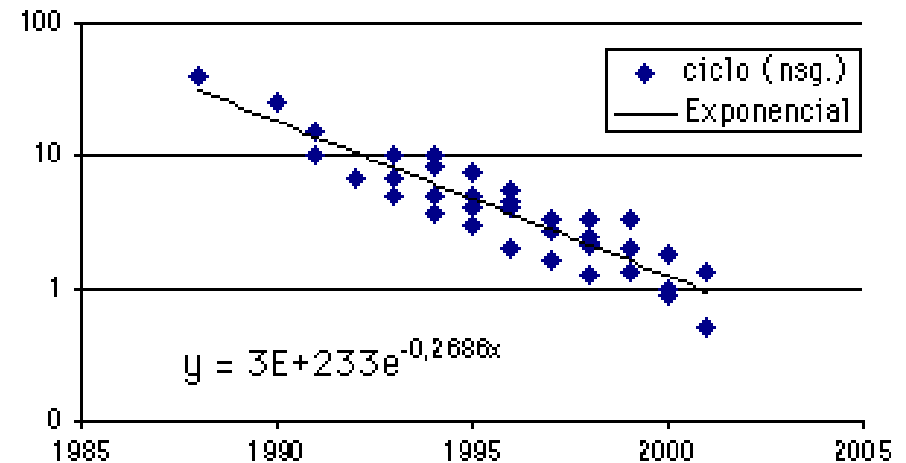
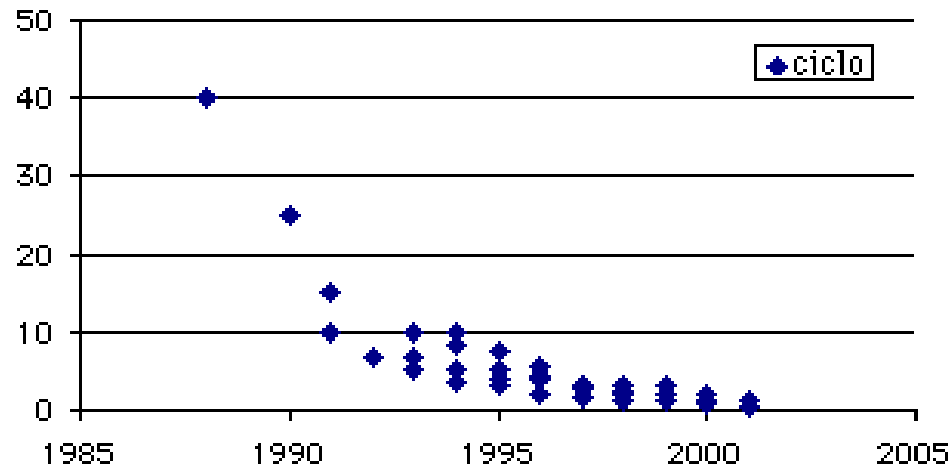
año	procesador	frecuencia (MHz)	ciclo (nsg.)
1988	motorola 88000	25	40,0
1990	intel i860	40	25,0
1991	HP 730	66	15,0
1991	mips R4000	100	10,0
1992	Alpha 21064	150	6,7
1993	Alpha 21064	200	5,0
1993	HP PA7100	100	10,0
1994	Alpha 21064	275	3,6
1994	HyperSparc	100	10,0
1995	PowerPC 604	133	7,5
1995	Alpha 21164	333	3,0
1996	Alpha 21164	500	2,0

año	procesador	frecuencia (MHz)	ciclo (nsg.)
1996	HP PA8000	180	5,6
1997	pentium II	300	3,3
1997	Alpha 21164	600	1,7
1998	Alpha 21264	800	1,3
1998	mips R12000	300	3,3
1999	pentium III	733	1,4
1999	Sparc Ultra II	300	3,3
2000	pentium III	1130	0,9
2000	HP PA8600	550	1,8
2001	pentium 4	2000	0,5
2001	PowerPC 74XX	733	1,4

Motorola
Intel
PA-Risc
Mips

Alpha
Sparc
PowerPC

Tiempo de ciclo del procesador



-23.6% cada año

■ Expresión para Tiempo de ciclo (Tc) en función del año:

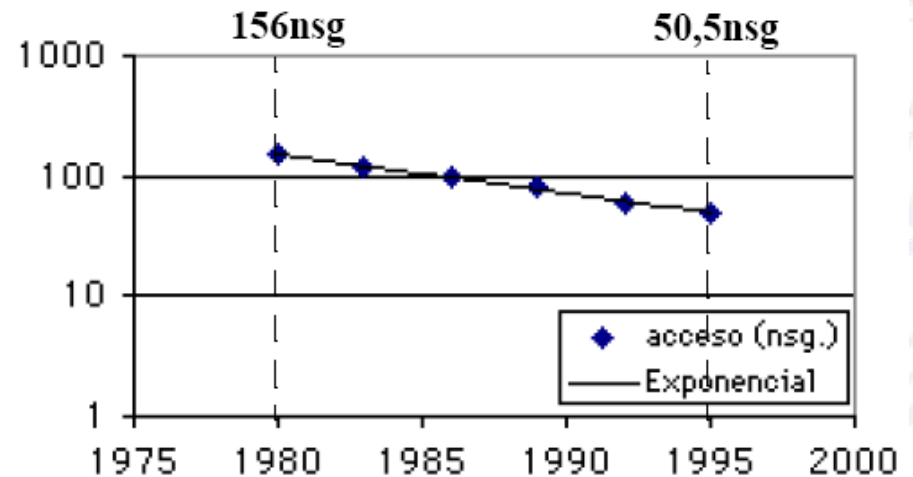
○ $T_c(\text{año}) = 21.9 * 0.764^{(\text{año}-1990)} \text{ ns}$

○ $T_c(2001) = 1.138 \text{ ns} \quad (880 \text{ MHz})$

Tiempo de acceso a memoria DRAM

■ Tabla y gráfica de tiempos de acceso a memorias DRAM

DRAM: tiempo de acceso RAS	
año	tiempo de acceso
1980	150
1983	120
1986	100
1989	80
1992	60
1995	50



-7.2% cada año

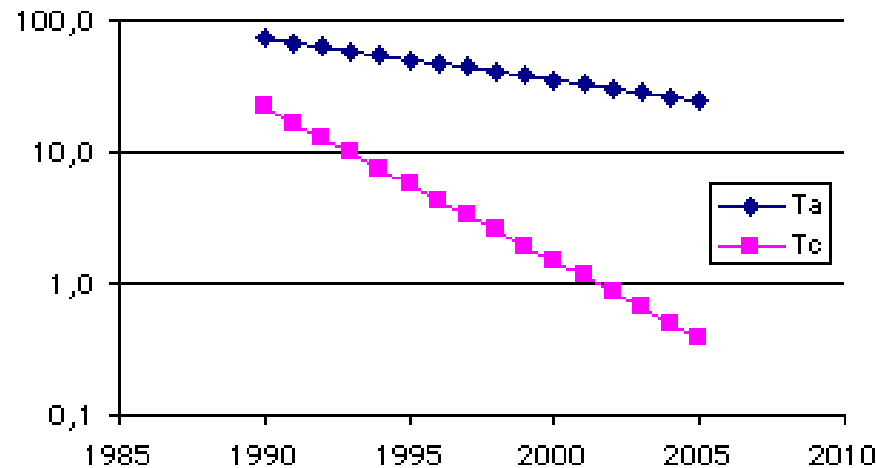
■ Expresión para Tiempo de acceso (T_a) en función del año:

○ $T_a(\text{año}) = 156 * 0,928^{(\text{año}-1980)} \text{ ns}$

○ $T_a(2001) = 32,5 \text{ ns}$ (307,7 MHz)

Discrepancia entre procesador y memoria DRAM

- Speed gap, memory wall

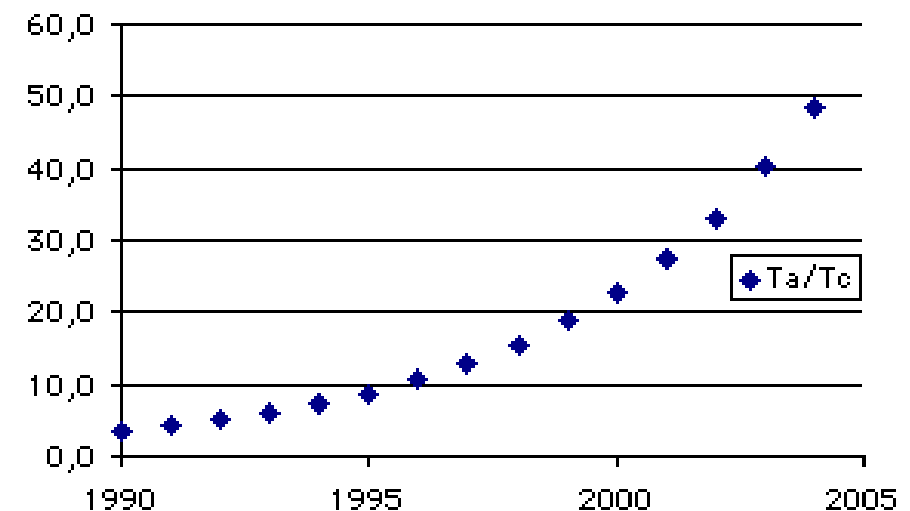


- Tiempo de acceso a memoria medido en ciclos de procesador:

$$\frac{\text{Tiempo de acceso a memoria}}{\text{Tiempo de ciclo de procesador}}$$

+21% cada año

- Ta (2001) = 27.4 ciclos por acceso



Guión del tema

- Motivación: velocidad procesador vs. memoria

- **Problema: cómo construir un almacén**

- **Grande, rápido y barato**

- Propiedad de los programas

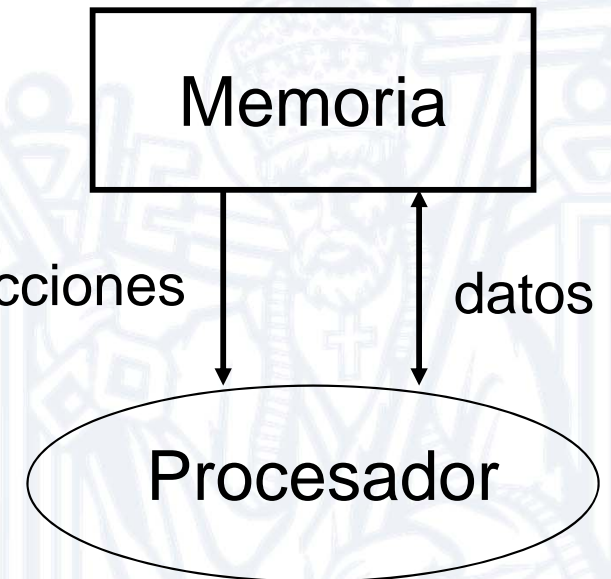
- Localidad espacial y temporal

- Jerarquía de memoria

- Memoria cache

- Ejemplos de memoria cache

- Descripción de una cache



Problema: cómo construir un almacén

- *Grande*: datos e instrucciones del programa completo
- *Rápido*: velocidad del procesador
- *Barato*: poco coste por bit (= muchos bits / mm²)

Memoria dinámica
DRAM

Grande y barata

Memoria estática
SRAM

Rápida

- *... y que además consume poca energía ? ...*

Memoria SRAM vs. DRAM¹

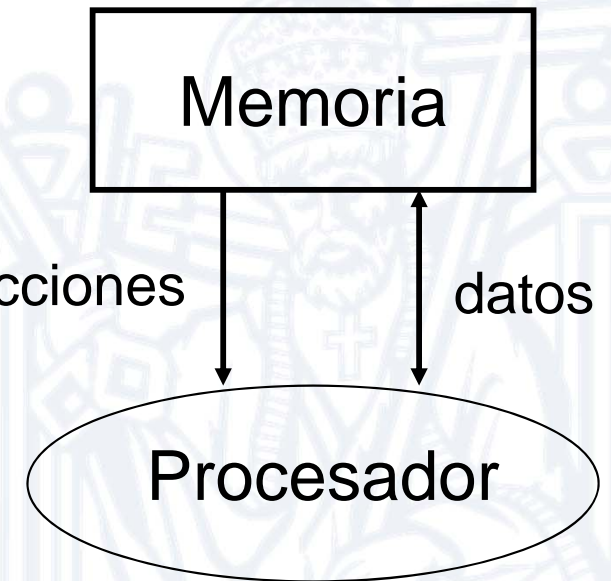
	Celda de almacenamiento	Capacidad (chip 2008)	Precio/MByte	Velocidad
DRAM	1 Condensador + 1 Transistor alta densidad	grande 2048 Mbits	barato 0,18\$	lenta 12.5 ns
SRAM	4Trans. + 2Trans. baja densidad (16-25x DRAM)	pequeña 32 Mbits	caro ~ 18\$	rápida 0.4 ns

- Ejemplo: 2048 Mbits de memoria principal para un PC
 - DRAM: 16 chips, 20 €
 - SRAM: 64 chips, ~2.000 € (¡ sólo los chips !)
- Otros problemas de SRAM:
 - Encaminamiento aumentaría mucho el tiempo de acceso de SRAM
 - Consumo mucho mayor en SRAM

1. The Stacked Capacitor DRAM Cell and Three-Dimensional Memory. Mitsumasa Koyanagi, January 2008 IEEE Solid State Circuits Society Journal.
http://www.ieee.org/portal/site/sscs/menuitem.f07ee9e3b2a01d06bb9305765bac26c8/index.jsp?&pName=sscs_level1_article&TheCat=2171&path=sscs/08Winter&file=Koyanagi.xml

Guión del tema

- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- **Propiedad de los programas**
 - **Localidad espacial y temporal**
- Jerarquía de memoria
 - Memoria cache
- Ejemplos de memoria cache
- Descripción de una cache



Localidad espacial y temporal

- Propiedad de los programas:

Si un procesador referencia una posición de memoria

- Es probable que vuelva a referenciar pronto

la misma posición de memoria

→Localidad temporal

- Es probable que referencie pronto

posiciones de memoria **cercanas**

→Localidad espacial

Localidad en código

■ Secuenciamiento implícito

- Después de la instrucción i se ejecuta la instrucción $i+1$

=> Localidad espacial

■ Ejecución de bucles

- Cada N instrucciones se ejecuta la instrucción i

=> Localidad temporal

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

@ memoria

```
500    mov    0, r4
```

bucle:

```
504    sll    r4, 2, r5
```

```
508    fload  [r6+r5], f2
```

```
512    fload  [r7+r5], f3
```

```
516    fadd   f2, f3, f2
```

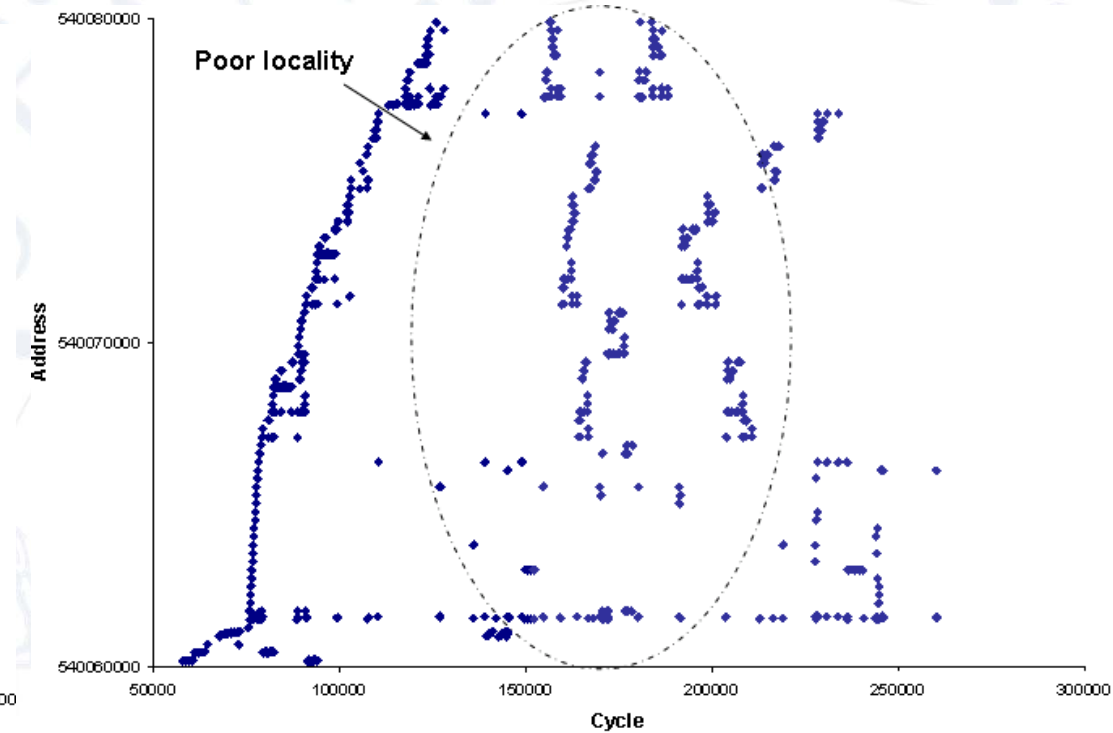
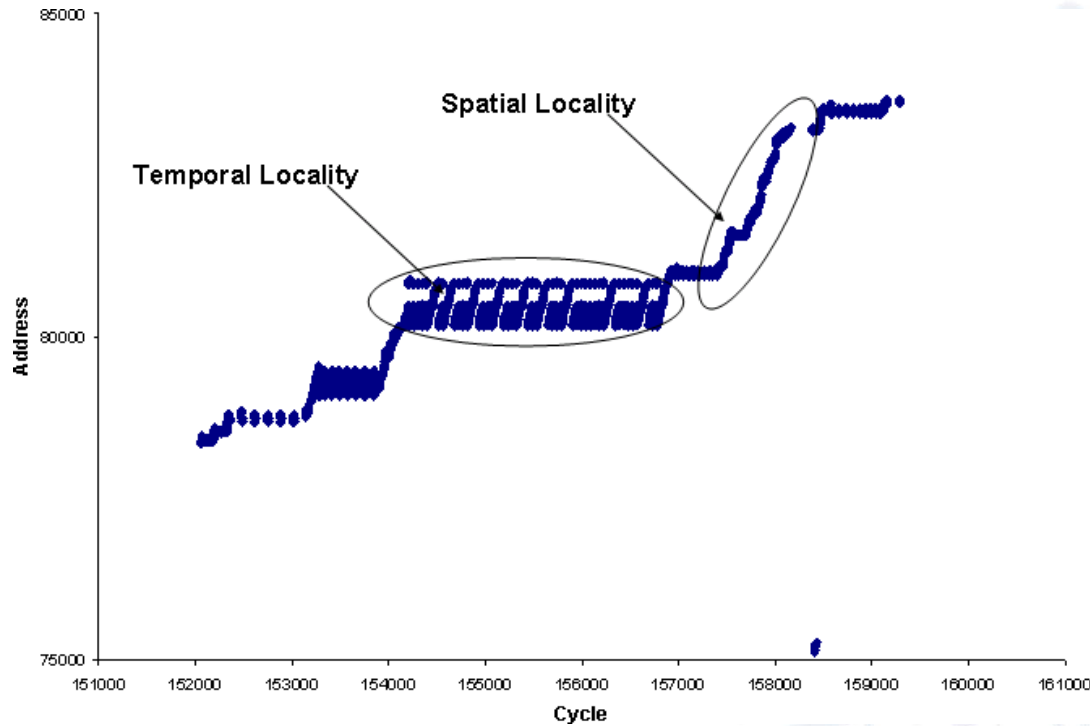
```
520    fstore f2, [r8+r5]
```

```
524    add    r4, 1, r4
```

```
528    cmp    r4, max
```

```
532    bl     bucle
```

Localidad de instrucciones en acción

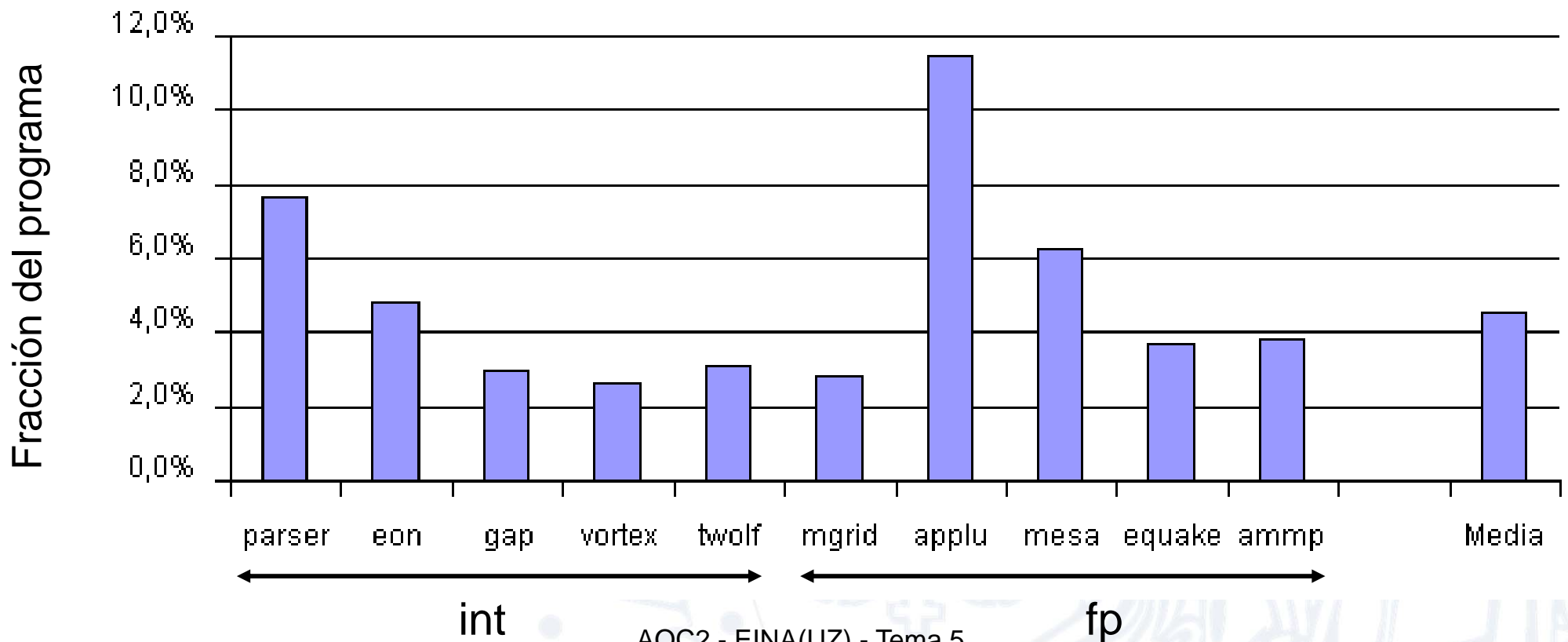


Localidad en código

■ Regla empírica

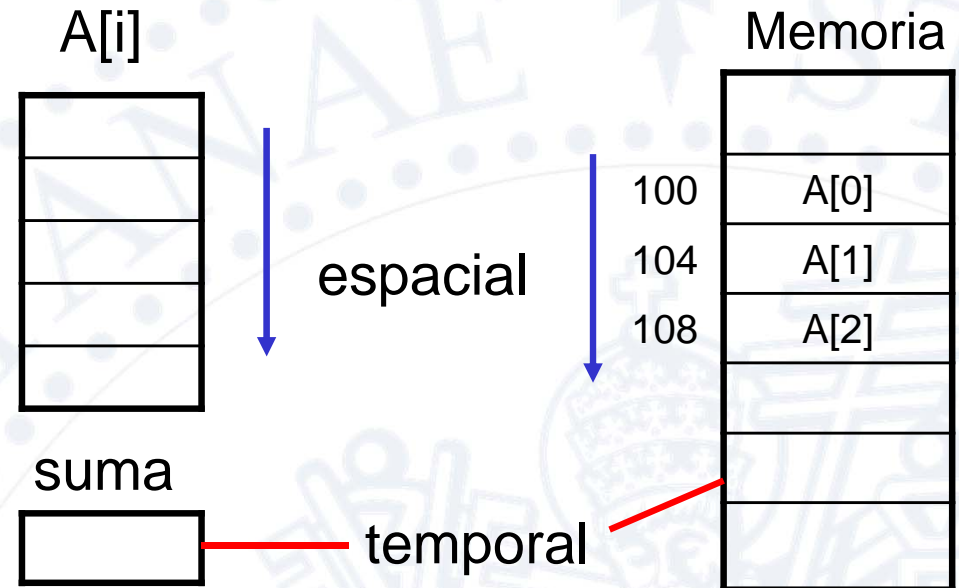
El 90% de las instrucciones ejecutadas (dinámico)
es atribuible a un 10% de las instrucciones (estático)

Localidad en instrucciones SPEC 2000



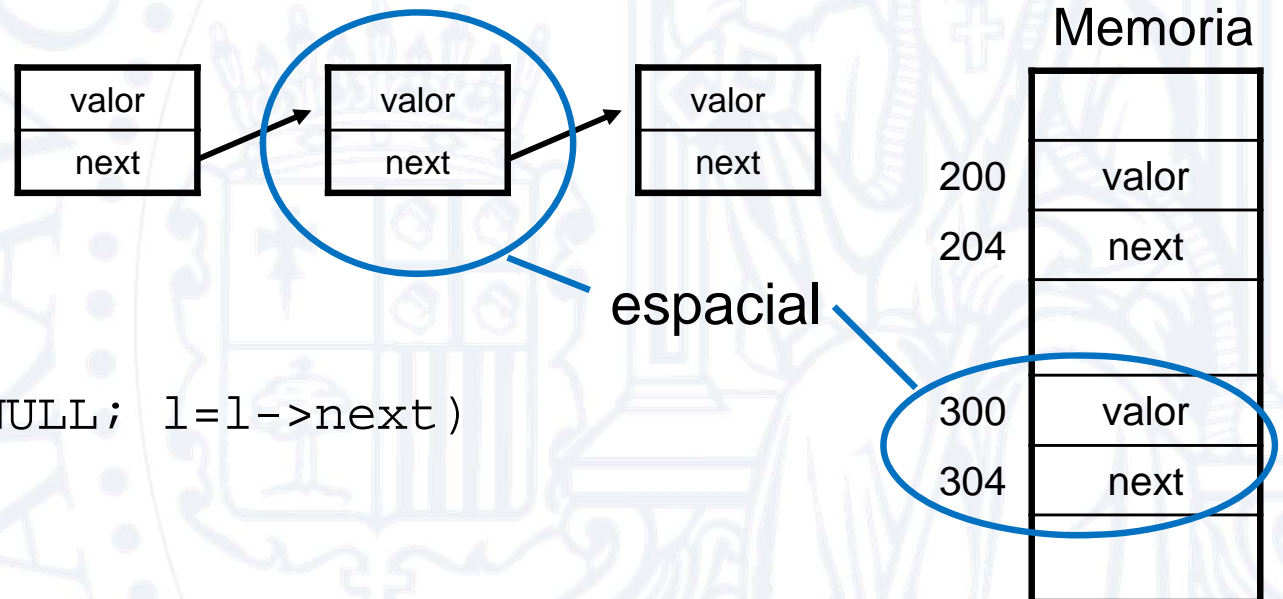
Localidad en datos

```
for (i=0; i<max; i++)  
    suma += A[i];
```



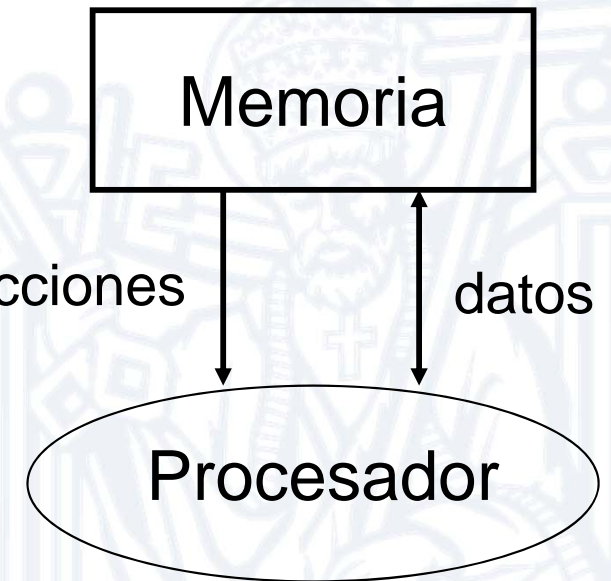
```
struct nodo {  
    int valor;  
    struct nodo *next;  
}
```

```
for (l=lista; l->next!=NULL; l=l->next)  
    suma += l->valor;
```



Guión del tema

- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- **Jerarquía de memoria**
 - **Memoria cache**
- Ejemplos de memoria cache
- Descripción de una cache



Jerarquía de memoria: Cache

Características de las memorias

- DRAM barata y lenta
- SRAM cara y rápida

Propiedad de los programas

- Localidad espacial y temporal
(+ regla 90/10)

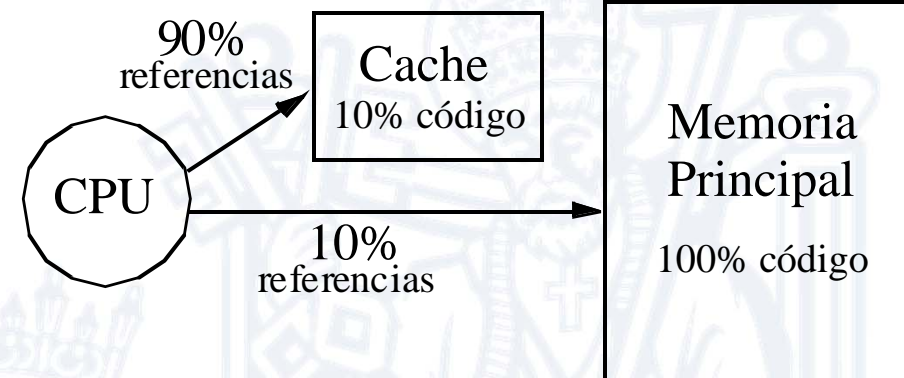
■ Solución: dos almacenes

- Mem. principal: DRAM, grande y lenta

almacén global

- Mem. cache: SRAM, pequeña y rápida

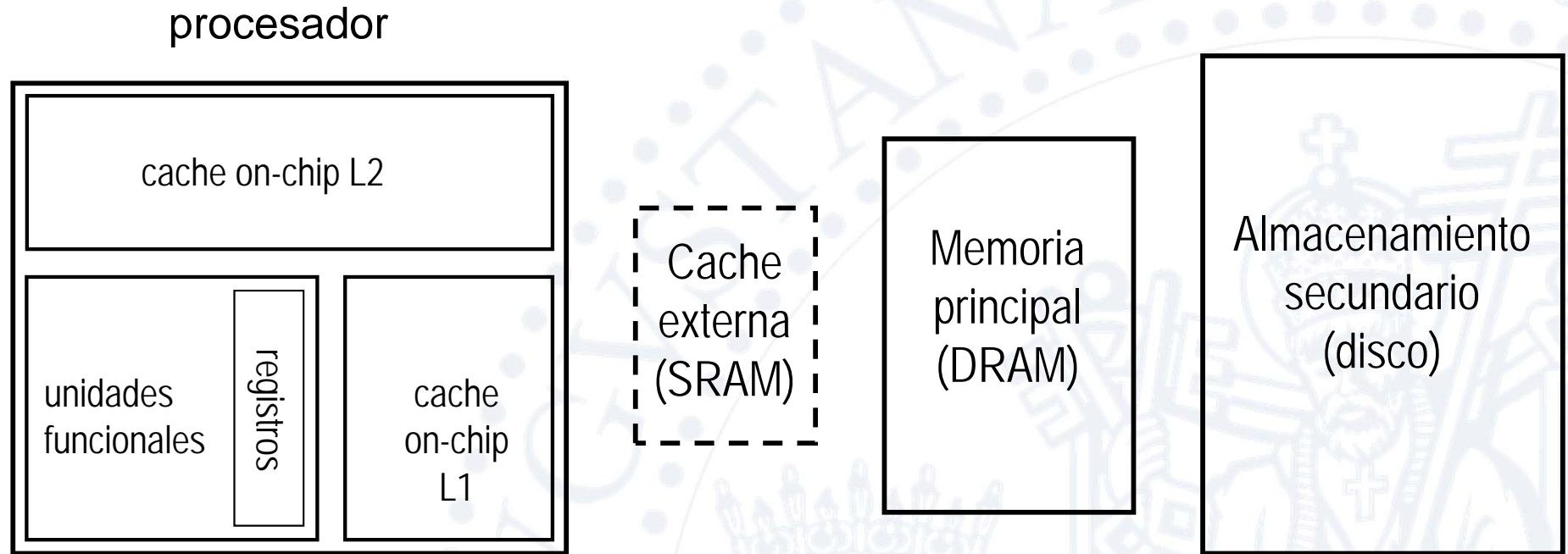
subconjunto más usado



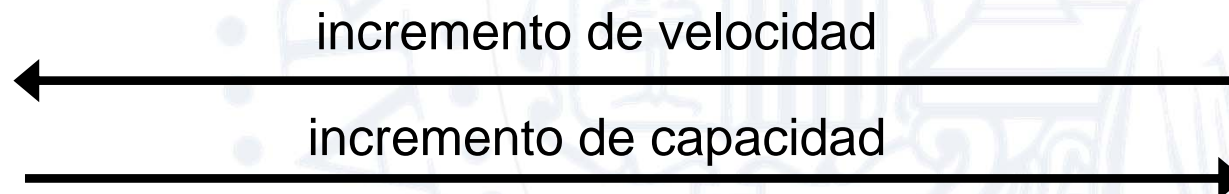
Memoria Cache: memoria pequeña y rápida que contiene el subconjunto más usado de memoria principal

Jerarquía de memoria

■ Varios niveles de almacenamiento

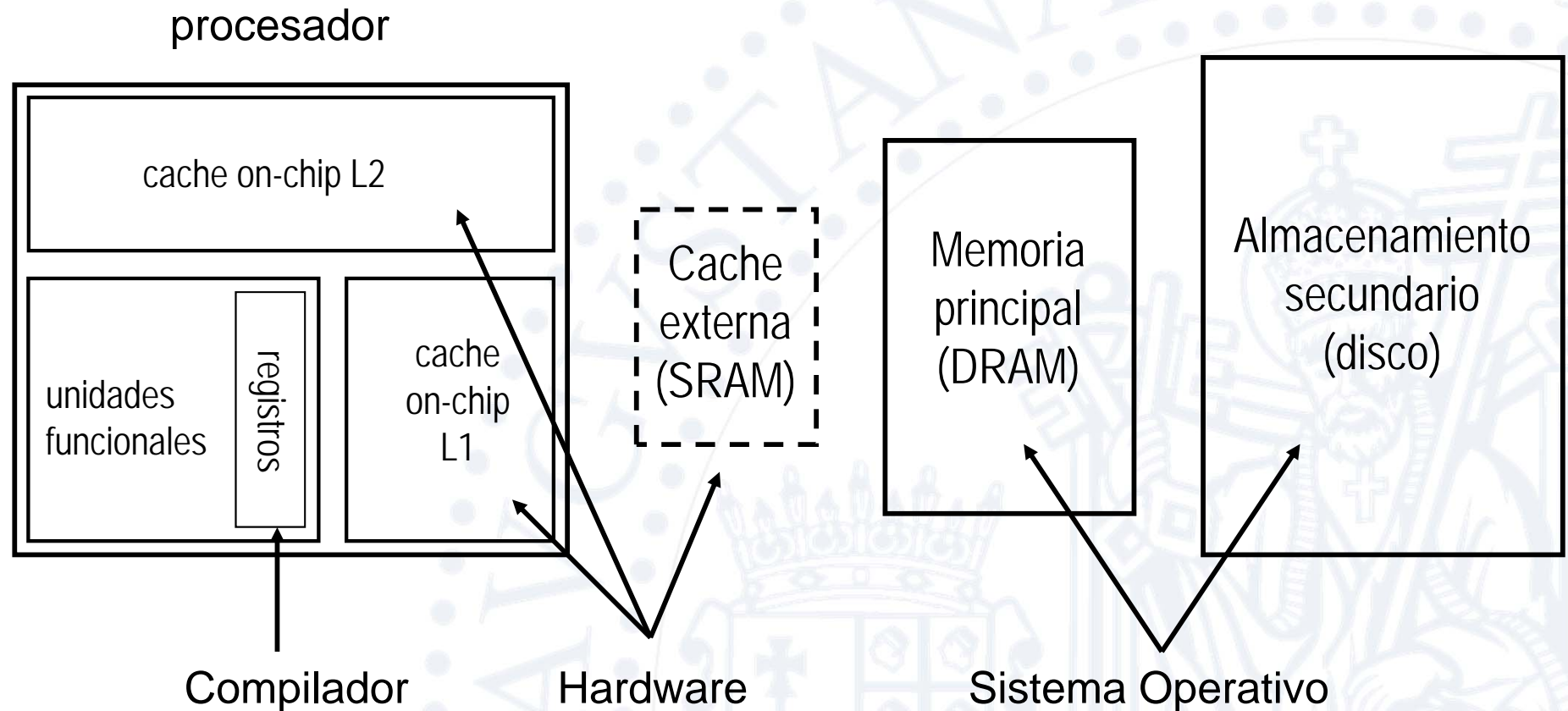


Tiempo (ciclos)	1	1-2	5-10	40	~ 80	10 M
Capacidad (bytes)	160 * 8	8-64 KB	~ 1 MB	~ 32 MB	Gigabytes	Terabytes



Jerarquía de memoria

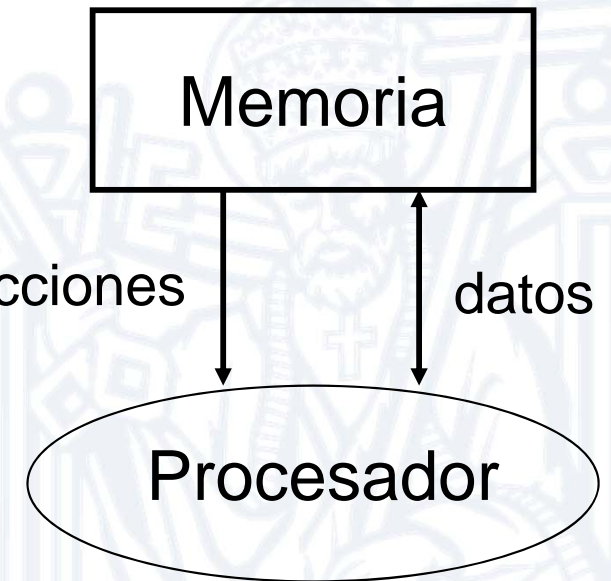
- Responsable de cada nivel



La memoria cache es transparente al nivel de lenguaje máquina

Guión del tema

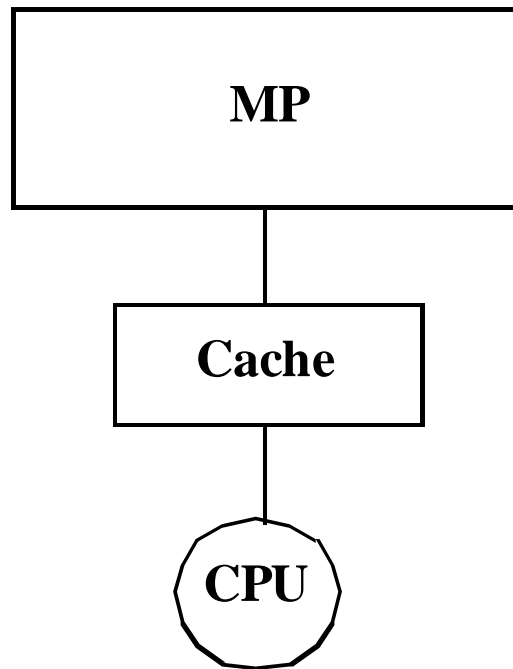
- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- Jerarquía de memoria
 - Memoria cache
- **Ejemplos de memoria cache**
- Descripción de una cache



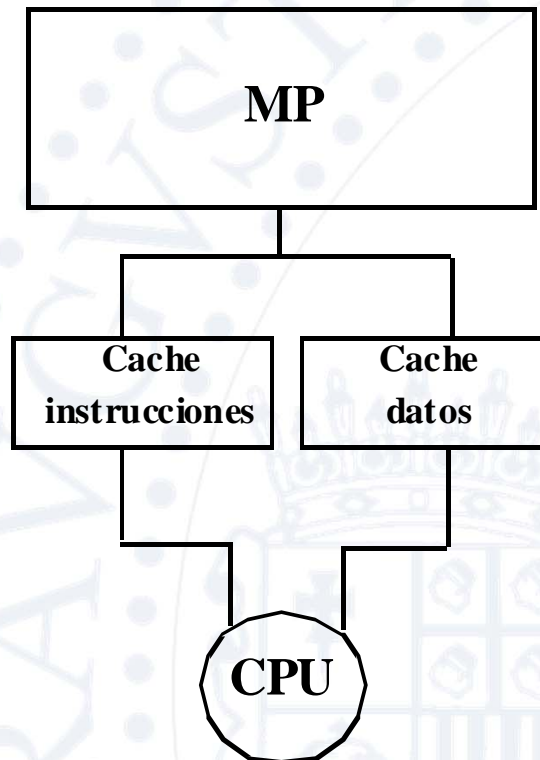
Ejemplos de memoria cache

- Organización de la memoria cache

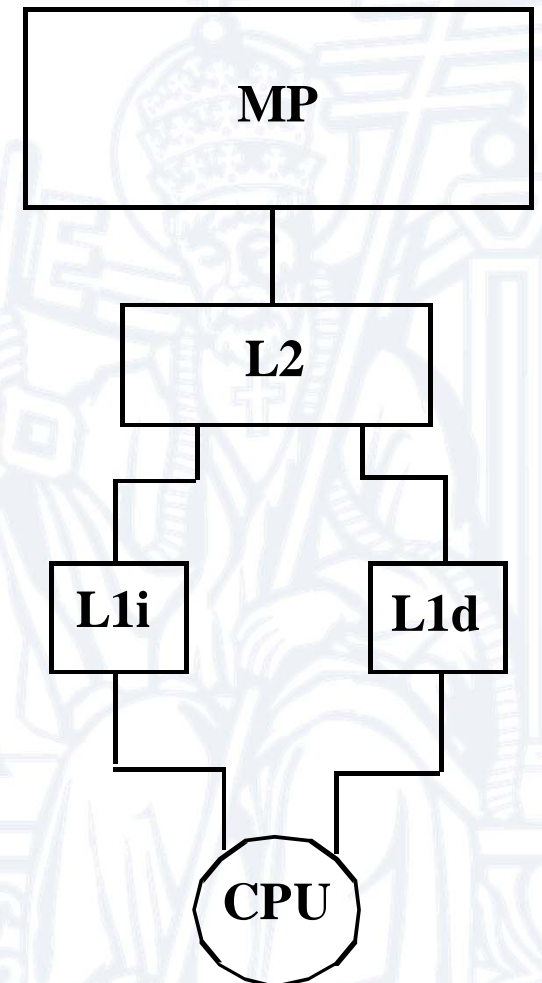
Cache unificada



Cache separada

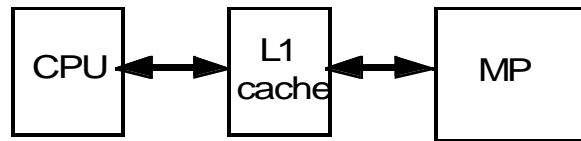


Cache multinivel

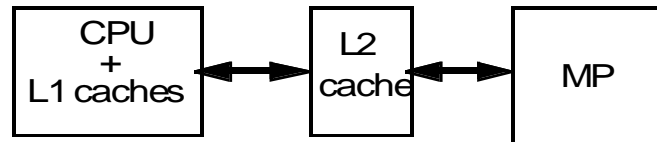


Evolución de la jerarquía en la familia Intel

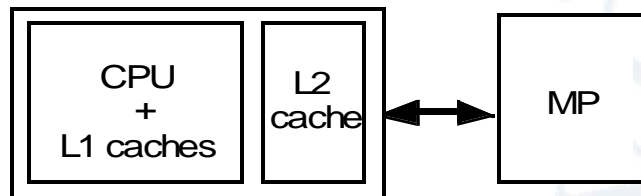
486 (1989)



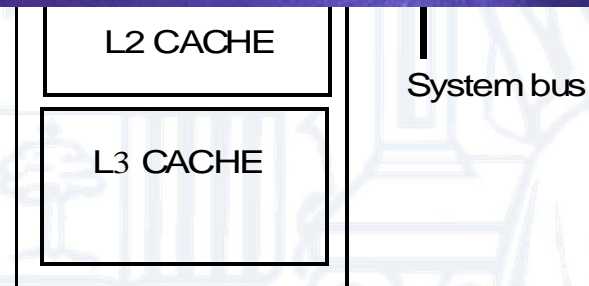
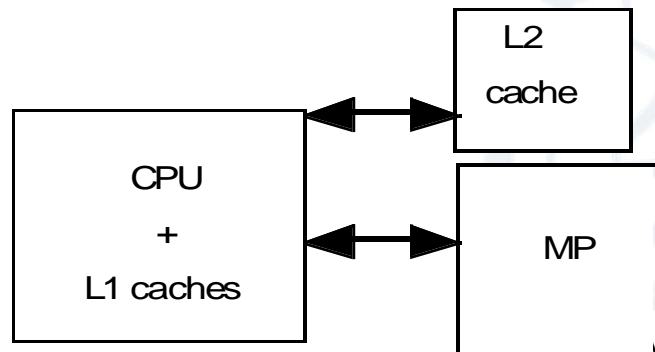
Pentium (1993)



Pentium Pro (1995)



Pentium II-III (1997-1999)



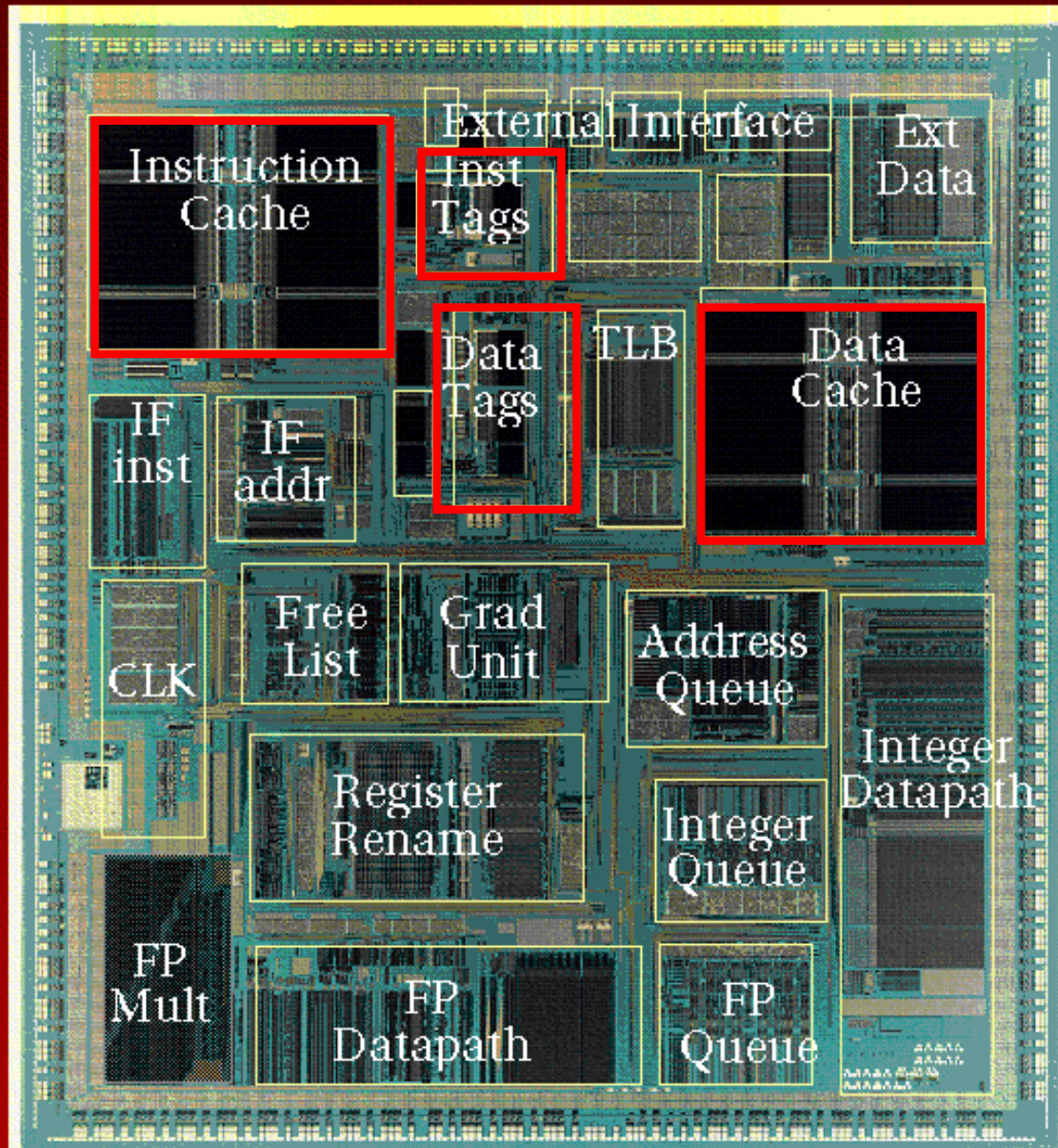
Ejemplos de memoria cache: capacidad

■ Chart Watch (nov-2009)

Procesador	L1i + L1d	L2	L3
Intel 2-core Xeon X5270	2 x (32 + 32) KB	6 MB	-
AMD 2-core Opteron X5270	2 x (64 + 64) KB	2 x 1 MB	-
Intel 4-core Xeon X7350	4 x (32 + 32) KB	2 x 4 MB	-
AMD 4-core Opteron 8360E	4 x (64 + 64) KB	4 x 512 KB	2 MB
Intel 6-core Xeon X7460	6 x (32 + 32) KB	3 x 3 MB	16 MB
Intel Itanium 2 9150M	2 x (16 + 16) KB	2 x (1MB + 256 KB)	12 MB
IBM Power6 (2-core)	2 x (64 + 64) KB	2 x 4 MB	32 M (off)
Fujitsu SPARC64 VII	4 x (64 + 64) KB	6 MB	-
Sun UltraSPARC T2+ (8-core)	8 x (8 + 16) KB	4 MB	-

MIPS R10K (1996-1998)

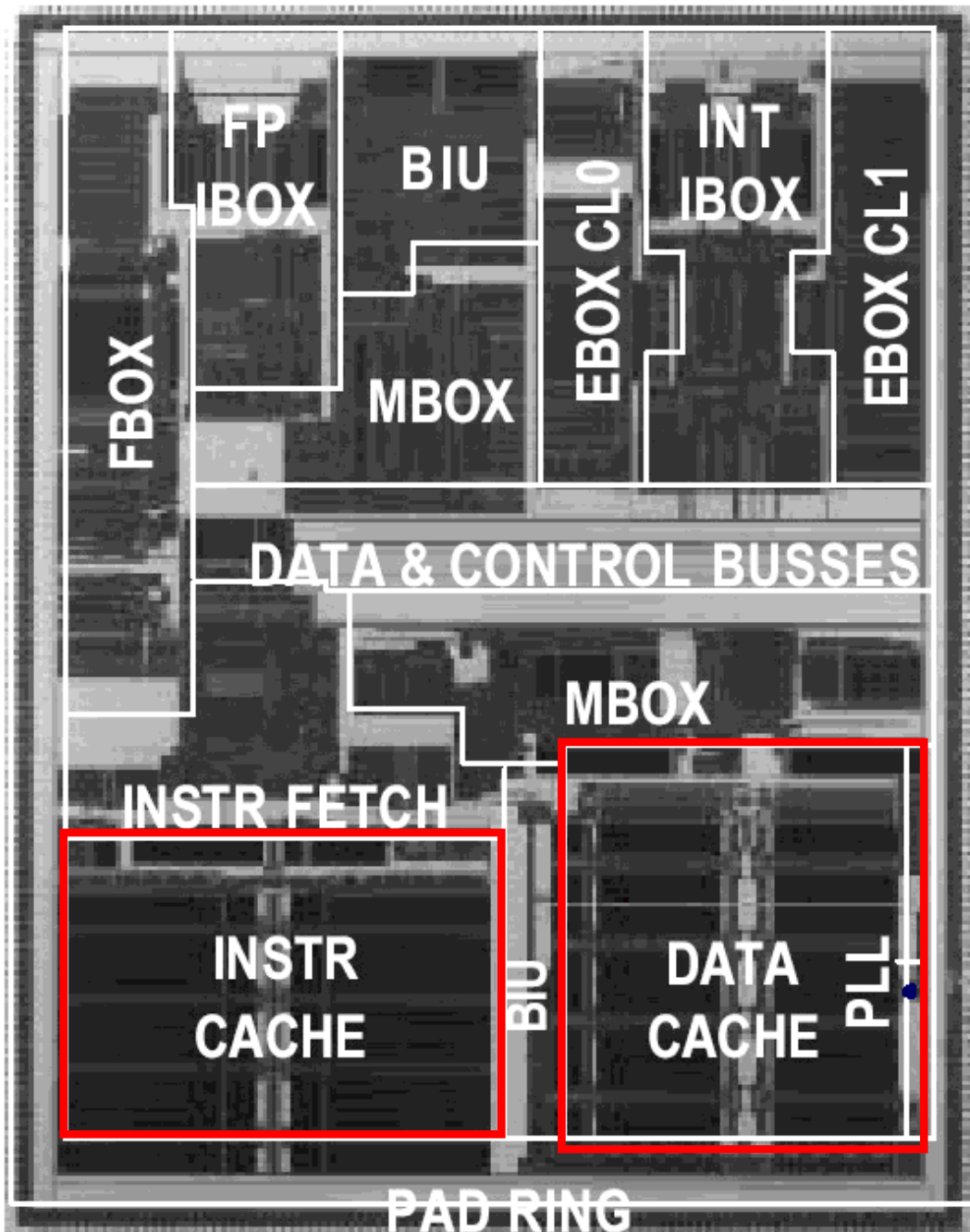
R10K die size 16.6mm x 17.9mm



	i	d
Capacidad	32 KB	32 KB
Total	64 KB	
% transistores	65%	
% area	~19%	

6,8 Mtransistores
297 mm²

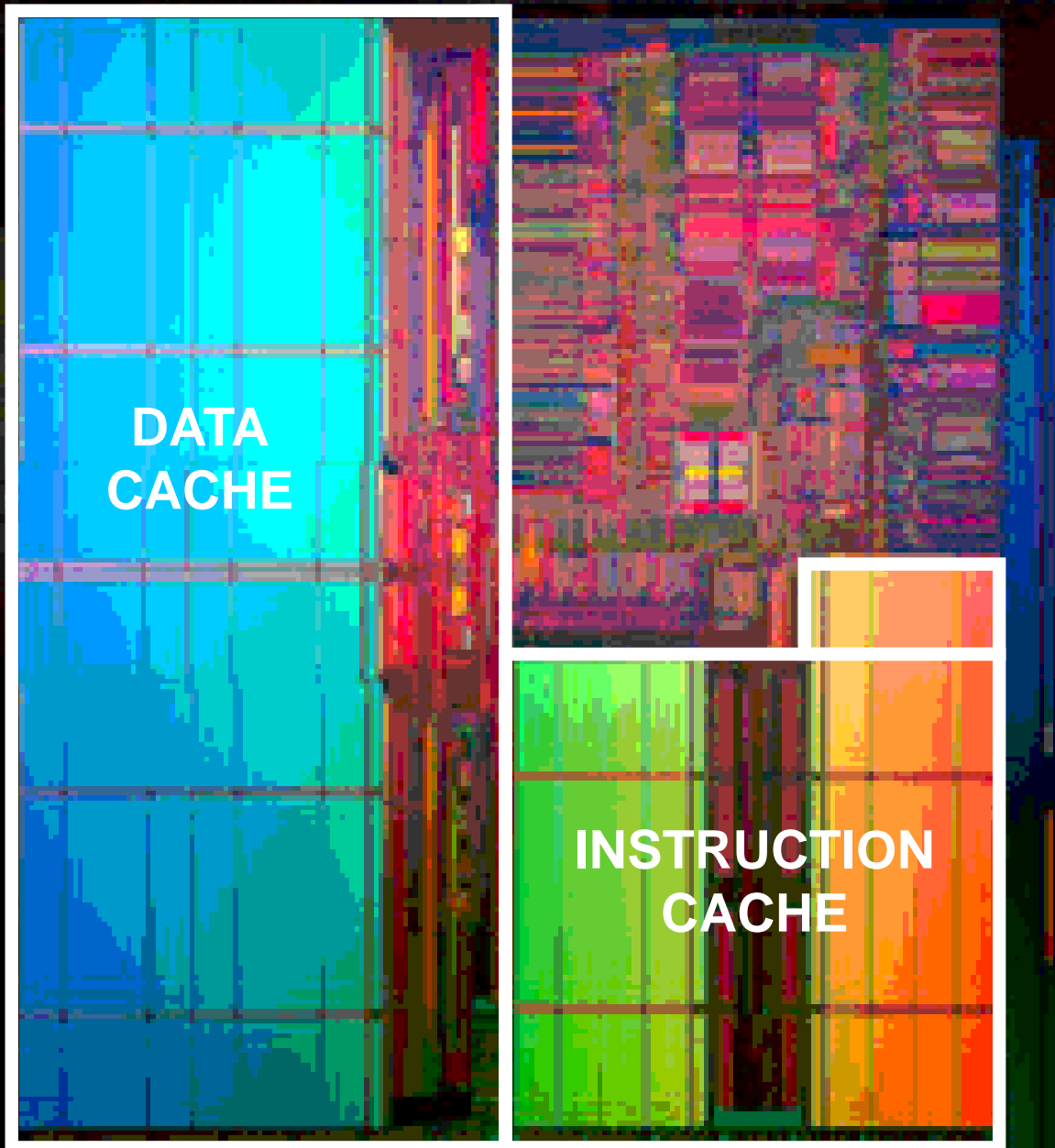
Alpha 21264 (1998-2000)



	i	d
Capacidad	64 KB	64 KB
Total	128 KB	
% transistores	nd	
% area	~27%	

15 Mtransistores
 15.8 x 19 mm = 300 mm²

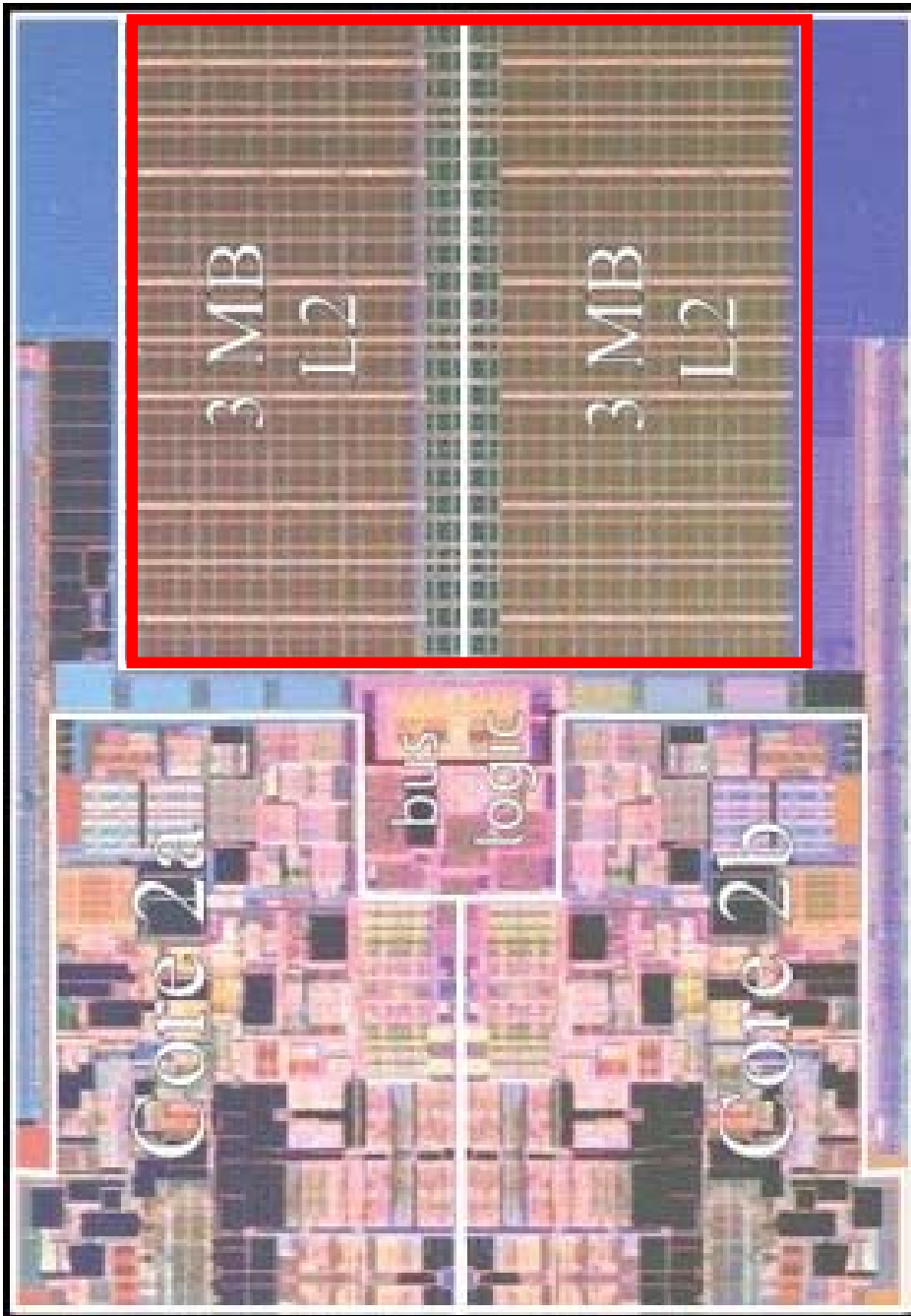
HP PA-8700 (2001-2002)



	i	d
Capacidad	768 KB	1536 KB
Total	2.25 MB	
% transistores	~90%	
% area	~75%	

186 Mtransistores
16 x 19 mm = 304 mm²

Intel Penryn Dual-Core (2008)

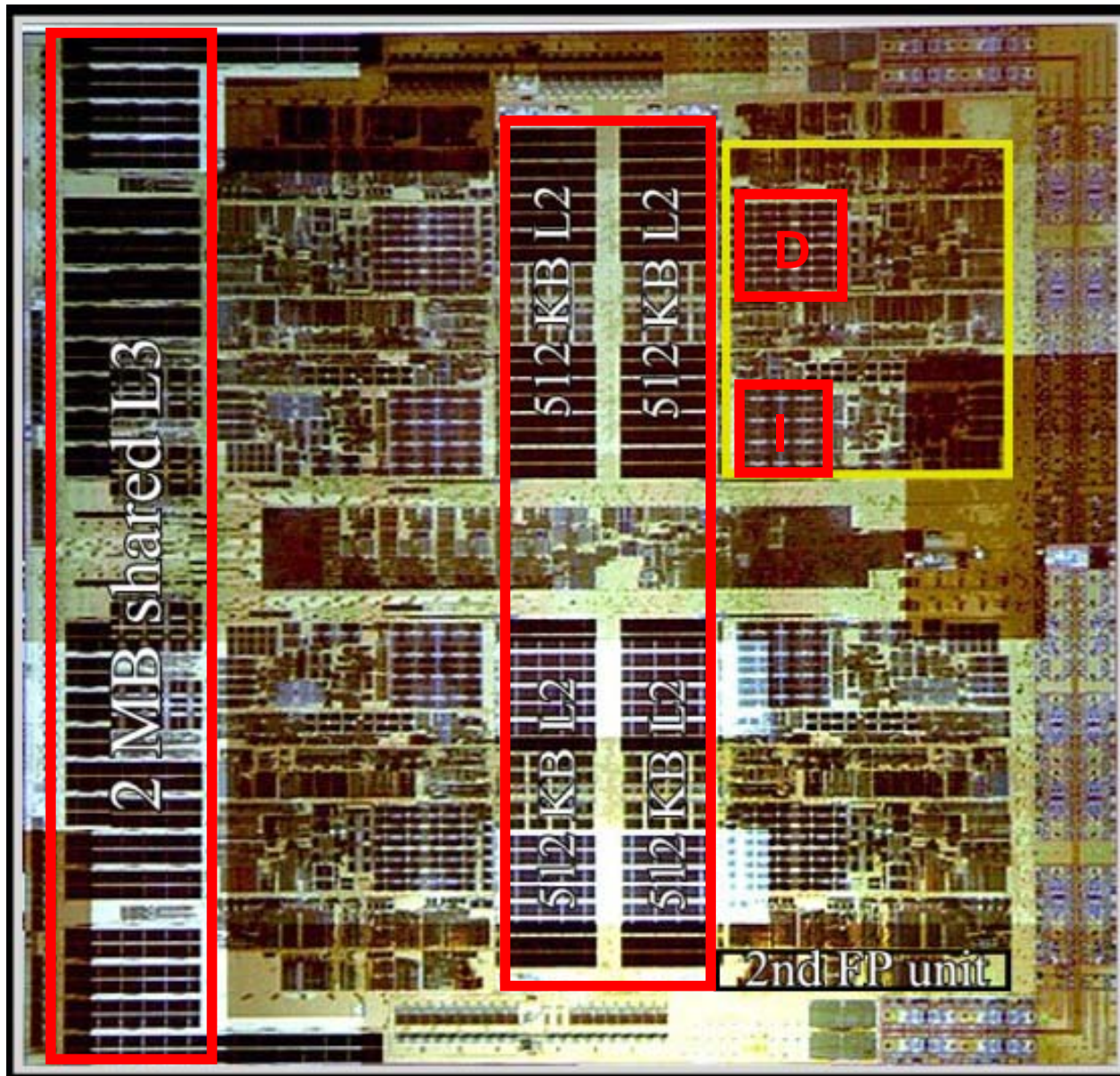


	i	d
L1	2x32 KB	2x32 KB
L2	2 x 3 MB	
L3	-	
Total	6.125 MB	
% transist.	~80%	
% area	~60% (a ojo)	

410 Mtransistores

12.3 x 8.6 mm = 107 mm² @ 65 nm

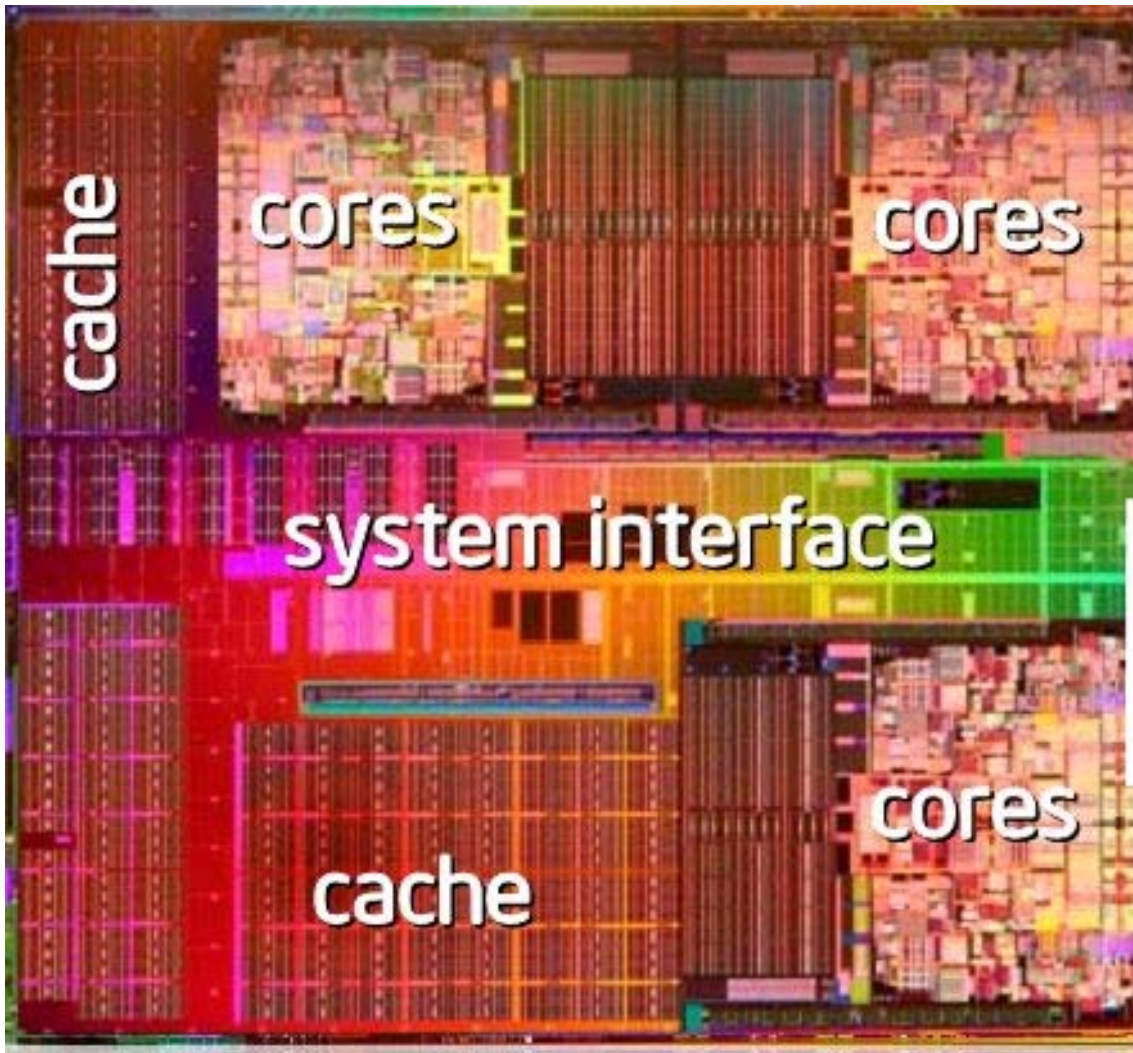
AMD Quad Core, K10 Barcelona (2008)



	i	d
L1	4x64 KB	4x64 KB
L2	4 x 512 KB	
L3	2 MB	
Total	4.5 MB	
% transist.	~50%	
% area	nd	

463 Mtransistores
283 mm² @ 65 nm

Intel Dunnington: 6-core Xeon X7460 (3Q 2008)

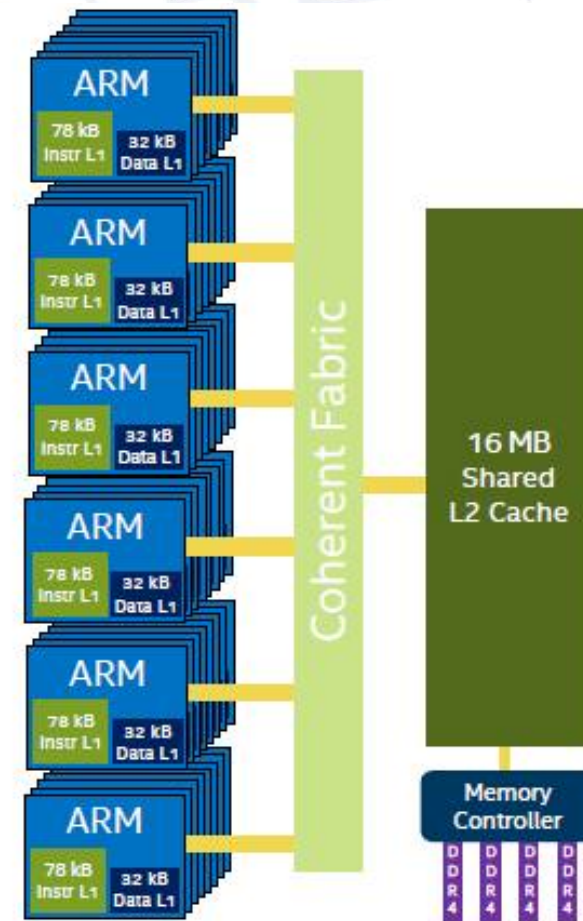
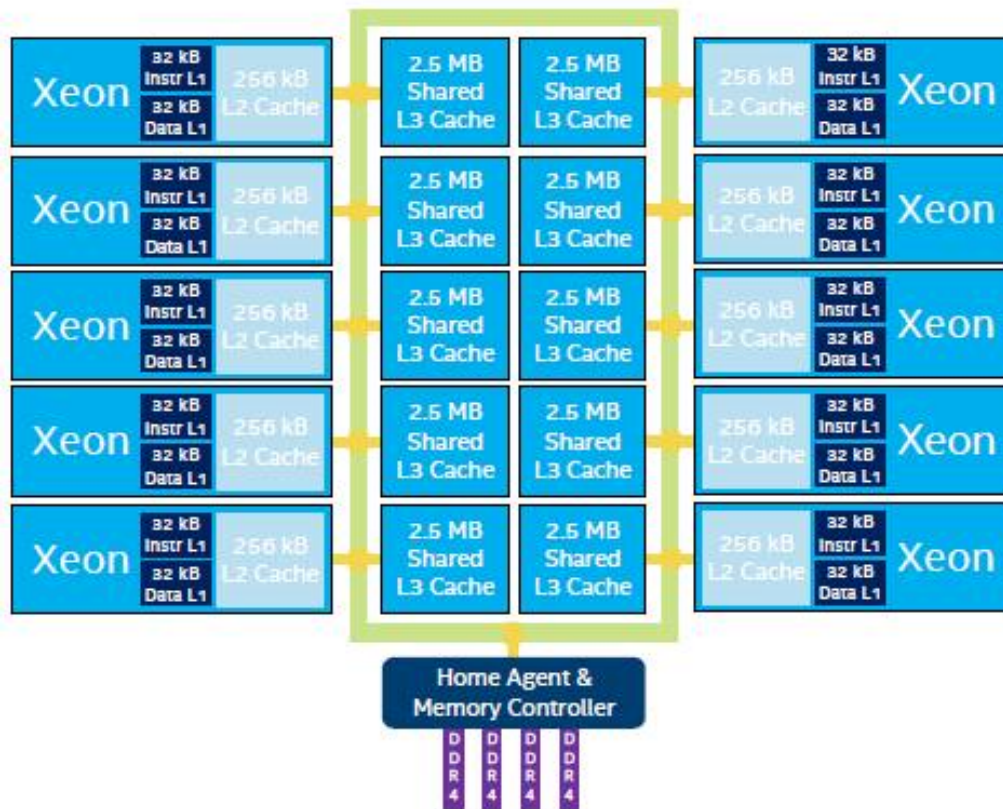


	i	d
L1	6x32 KB	6x32 KB
L2	3 x 3 MB	
L3	16 MB (≈ 100 ciclos)	
Total	25.3 MB	
% transist.	nd	
% area	nd	

1.9 Gtransistores
503 mm² @ 45 nm high-K
2.66 GHz, 130W
\$2729 1K unidades Feb 2010
2 threads/core

SoCs para servidores: ARM vs Intel 2016

Cache and Memory Hierarchy






48 cores ARMv8 a 2 GHz

SoCs para servidores: ARM vs Intel 2016

Cache and Memory Latency Comparisons

Specifications

	Cavium ThunderX <ul style="list-style-type: none">Dedicated L1 cache (78kB instr/32kB data)Shared L2 cache (16MB behind crossbar)No L3 cacheDDR4: 4 channels/CPU, up to 2400
	Intel® Xeon® D-1581 <ul style="list-style-type: none">Dedicated L1 cache (32kB instr/32kB data)Dedicated L2 cache (256kB/core)Shared L3 cache (2.5MB per core)DDR4: 2 channels/CPU, up to 2133¹
	Intel® Xeon® E5-2640 v4 <ul style="list-style-type: none">Dedicated L1 cache (32kB instr/32kB data)Dedicated L2 cache (256kB/core)Shared L3 cache (2.5MB per core)DDR4: 4 channels/CPU, up to 2133¹

Measurements (nsec)

Lower is better

Latency Measurement	Cavium ThunderX_CP (48C, 2GHz)	Intel® Xeon® D-1581 (16C, 1.8GHz)	Intel® Xeon® E5-2640 v4 (10C, 2.6GHz)
L1 Cache read	1.51	1.67 Up to 10% slower	1.17 Up to 22% faster
L2 Cache read	20.4	5.01 Up to 75% faster	3.53 Up to 82% faster
L3 Cache read	No L3 cache	19.3	14.9
DDR4 read	142-218**	63.7 Up to 55% - 70% faster	61.1 Up to 56% - 71% faster

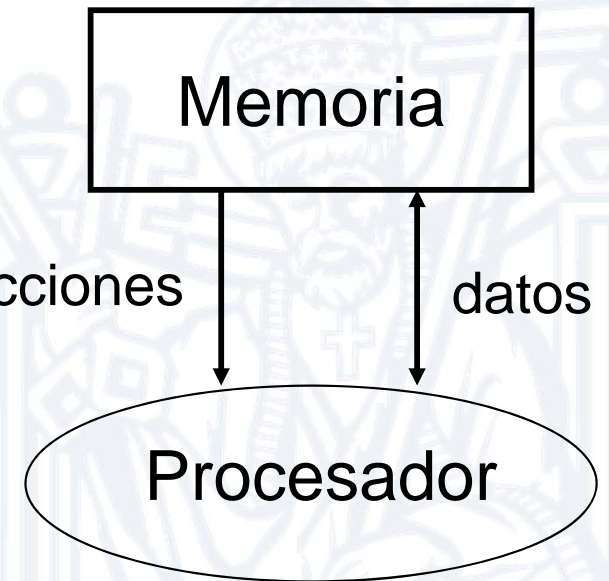
**Memory latencies vary from 142 to 218 nsec for unknown reasons

"Workloads that contain large sequential portions or are highly sensitive to latency are a poor match [for ThunderX]." *David Kanter, Microprocessor Report, 2/1/2016*

Intel va a añadir pronto una L4 en los Skylake Xeons!

Guión del tema

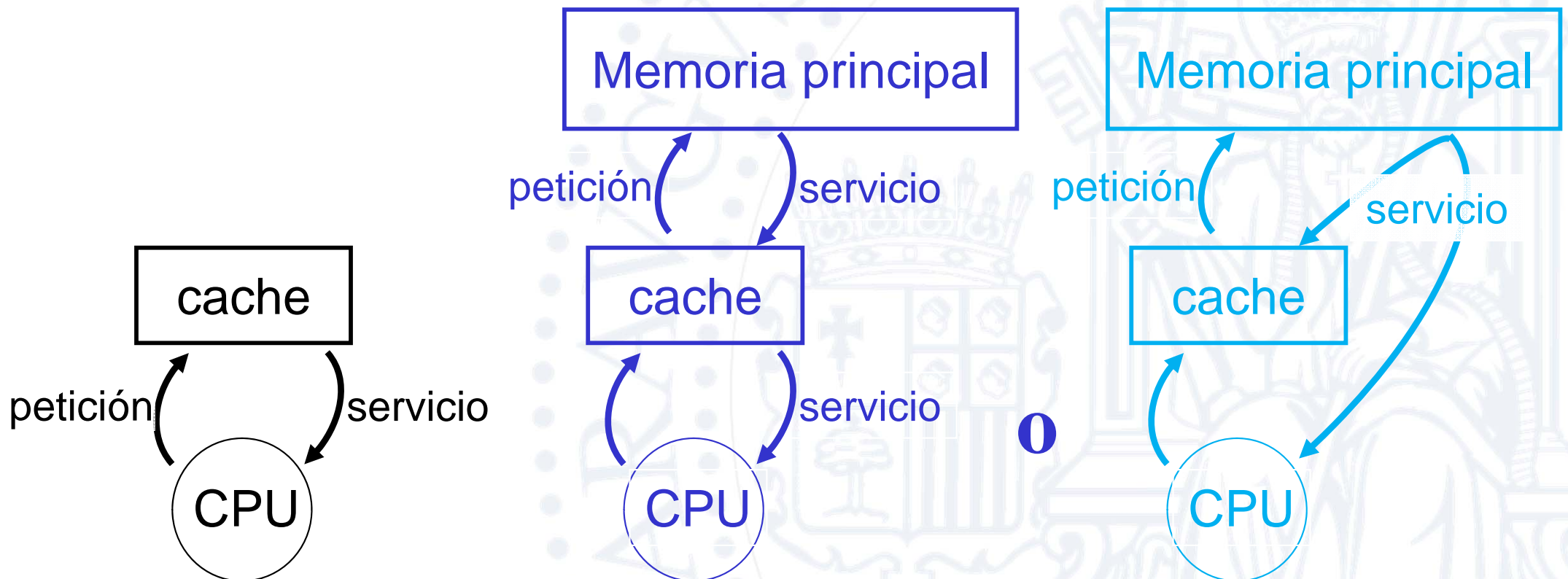
- Motivación: velocidad procesador vs. memoria
- Problema: cómo construir un almacén
 - Grande, rápido y barato
- Propiedad de los programas
 - Localidad espacial y temporal
- Jerarquía de memoria
 - Memoria cache
- Ejemplos de memoria cache
- **Descripción de una cache**



Descripción funcional

■ Acceso del procesador

- Primero se mira en cache
 - ◆ Acierto (*hit*) : se encuentra la palabra buscada
 - ◆ Fallo (*miss*): no se encuentra la palabra buscada
- Sólo en caso de fallo se accede a memoria principal



Descripción funcional: gestión de contenidos

- Capacidad de Cache \ll capacidad de Memoria

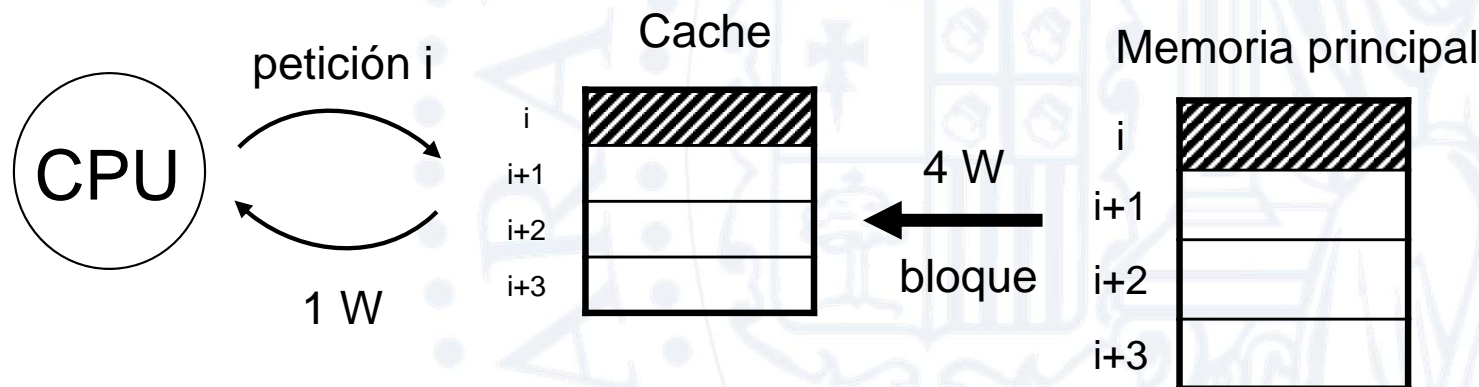
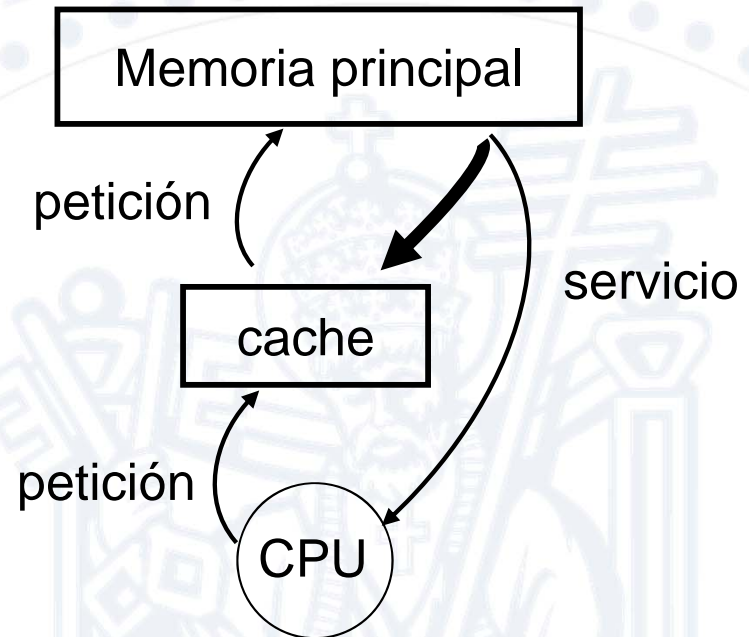
- ¿qué datos se guardan en cache?

- Aprovechar localidad temporal

- Si CPU pide una palabra, probablemente la volverá a pedir
 - ◆ Guardar palabra en cache

- Aprovechar localidad espacial

- Si CPU pide una palabra, probablemente pedirá las cercanas
 - ◆ Guardar varias palabras contiguas: bloque



Prestaciones

■ Tasas

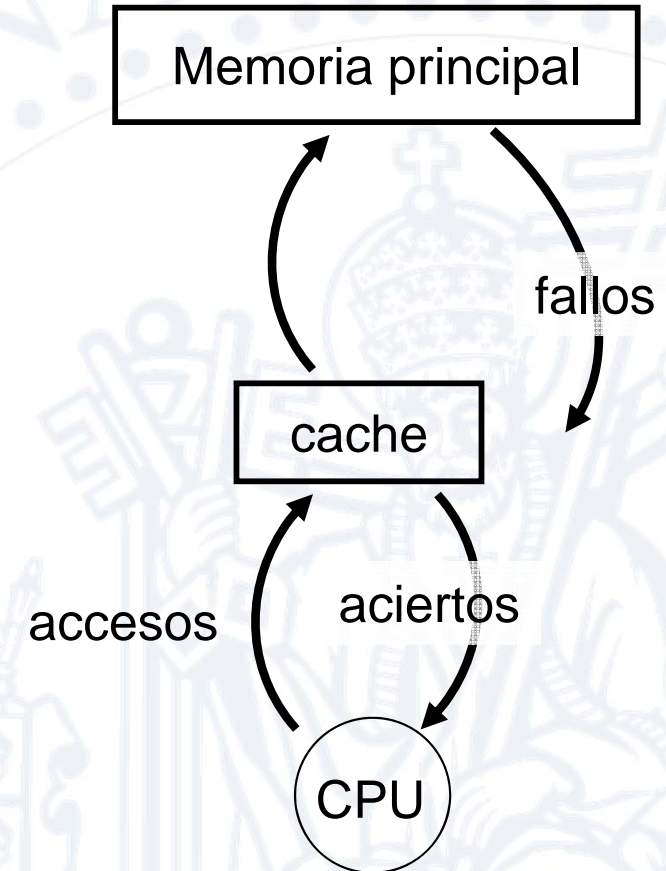
- Tasa de fallos (m)

$$m = \text{fallos} / \text{accesos}$$

- Tasa de aciertos (h)

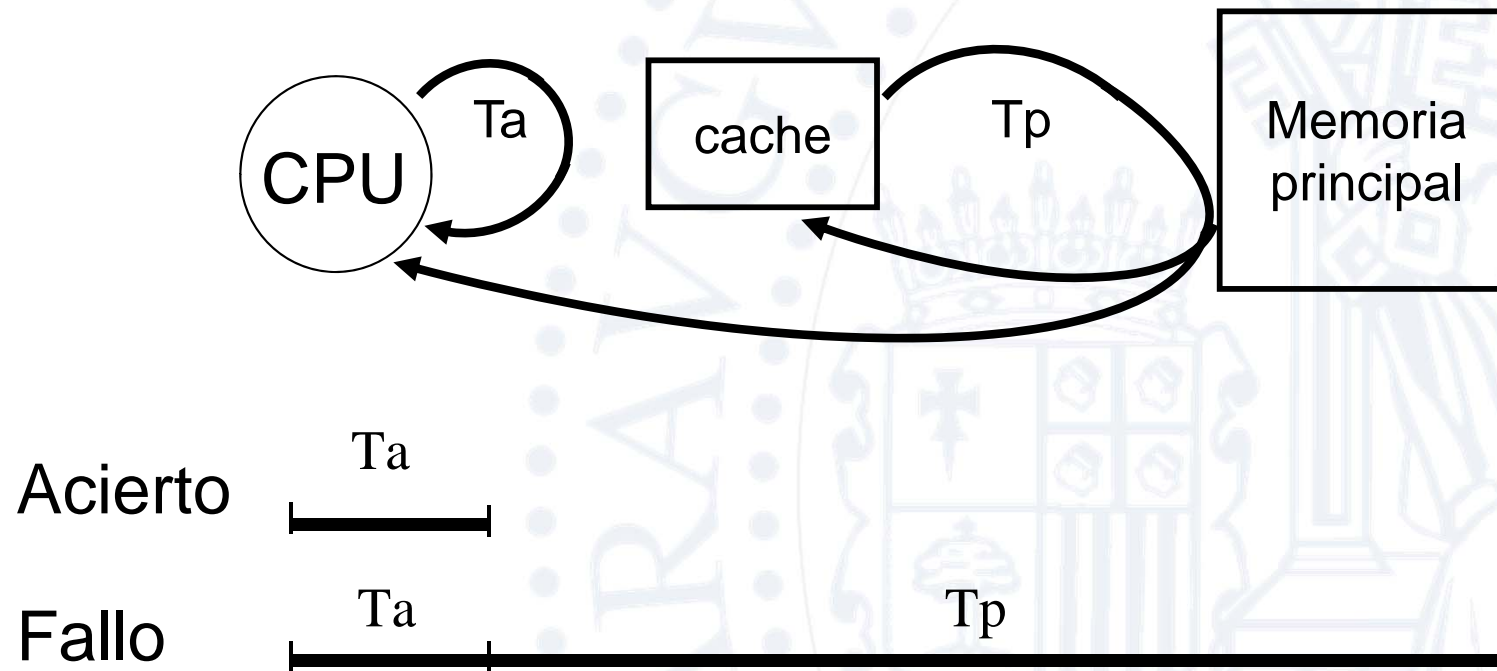
$$h = \text{aciertos} / \text{accesos}$$

$$h + m = 1$$



Costes (en tiempo o en nº de ciclos)

- Coste de acierto: T_a (ns) o C_a (ciclos)
 - Mirar en cache y entregar palabra al procesador
- Penalización de fallo: T_p (ns) o C_p (ciclos)
 - Desde que se detecta un fallo hasta que se sirve la palabra al procesador (leer bloque de MP, escribir bloque en cache ...)



Coste efectivo de acceso a memoria (T_{ef} , C_{ef})

■ Tiempo medio de acceso a memoria



$$T_{ef} = h \cdot T_a + m(T_a + T_p)$$

$$T_{ef} = T_a + \underbrace{m \cdot T_p}_{\text{penalización media}}$$

■ Ejemplo

$h = 0.9$ $m = 0.1$ $C_a = 1$ ciclo $C_p = 50$ ciclos	Sin cache	$C_{ef} = 50$ ciclos
	Con cache	$C_{ef} = 1 + 0.1 \cdot 50 = 6$ ciclos

Ejercicio

- Hay que seleccionar entre dos alternativas de cache:
 - a) 4 KB
 - b) 32 KB
- Tasas de fallos
 - a) 10%
 - b) 7.5%
- Calcula el CPI en cada caso sabiendo que:
 - El número medio de accesos a memoria por instrucción es 1,3
 - El CPI *ideal* es 3 (ideal = todos los accesos a memoria tardan un ciclo)
 - El coste de acierto es un ciclo, y la penalización de fallo son 10 ciclos
- La opción a) permite una frecuencia (procesador y cache) de 150MHz, mientras que la opción b) permite sólo 100MHz.
¿Qué opción da mejor rendimiento?

Parámetros de diseño

■ Capacidad de la cache

- Cuántos bloques puede albergar



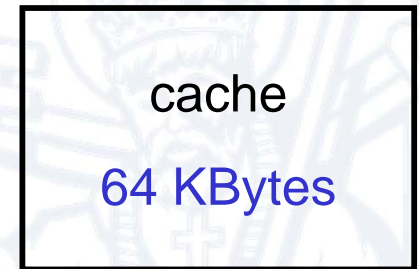
■ Tamaño de bloque

- Cuántos bytes tiene un bloque

ó

■ Correspondencia

- Regla de colocación y localización



■ Algoritmo de reemplazo

- Si la cache está llena, qué bloque expulsamos

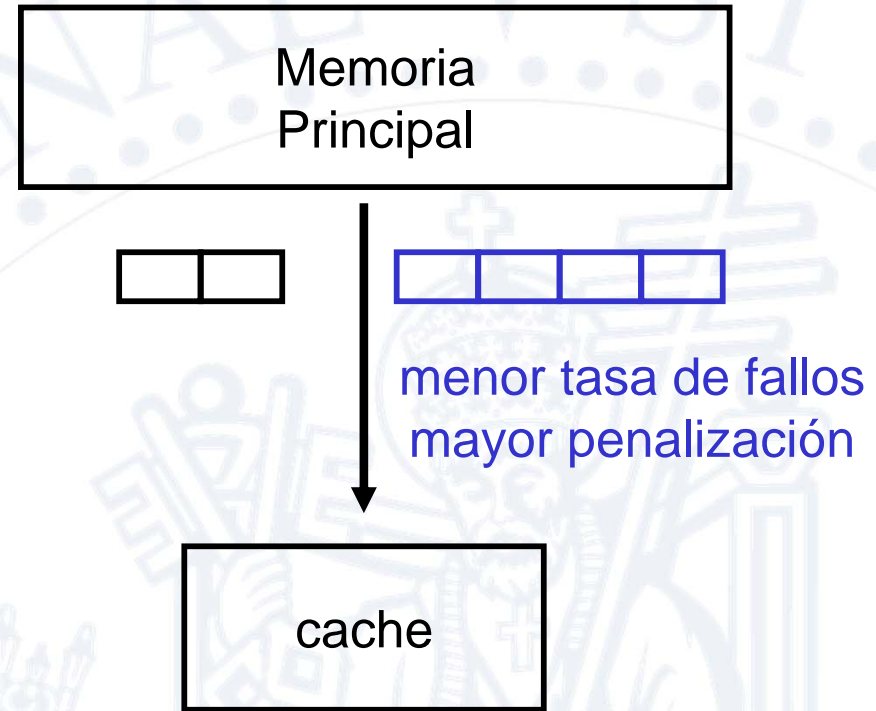
menor tasa de fallos
mayor tiempo de acierto

■ Política de escritura

- Qué ocurre cuando el procesador escribe

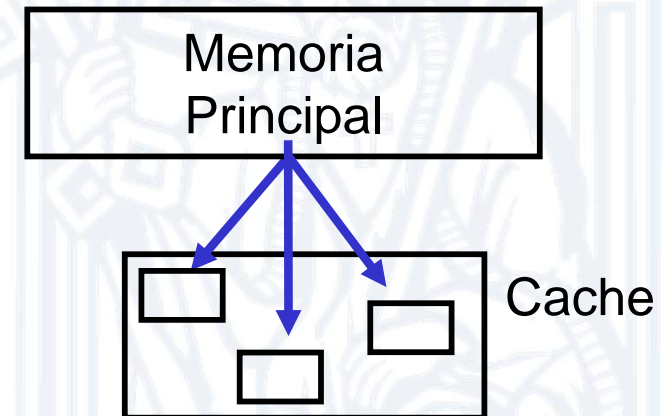
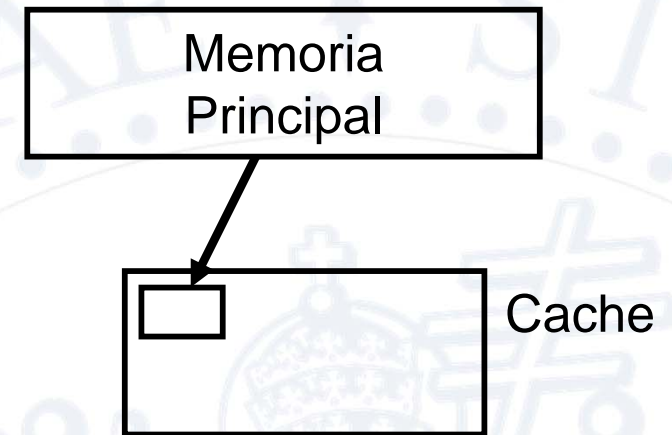
Parámetros de diseño

- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



Parámetros de diseño

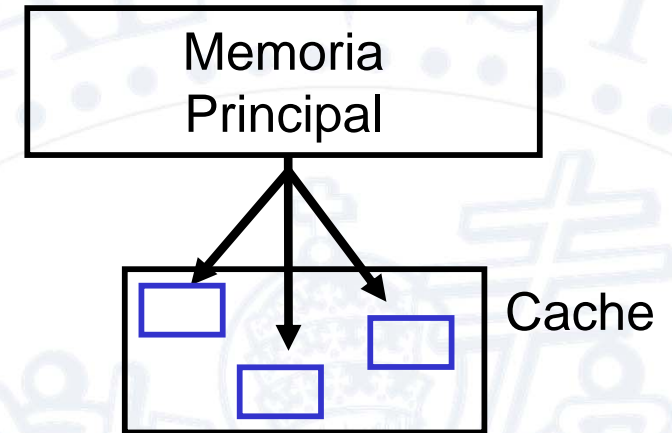
- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



menor tasa de fallos
mayor tiempo de acierto

Parámetros de diseño

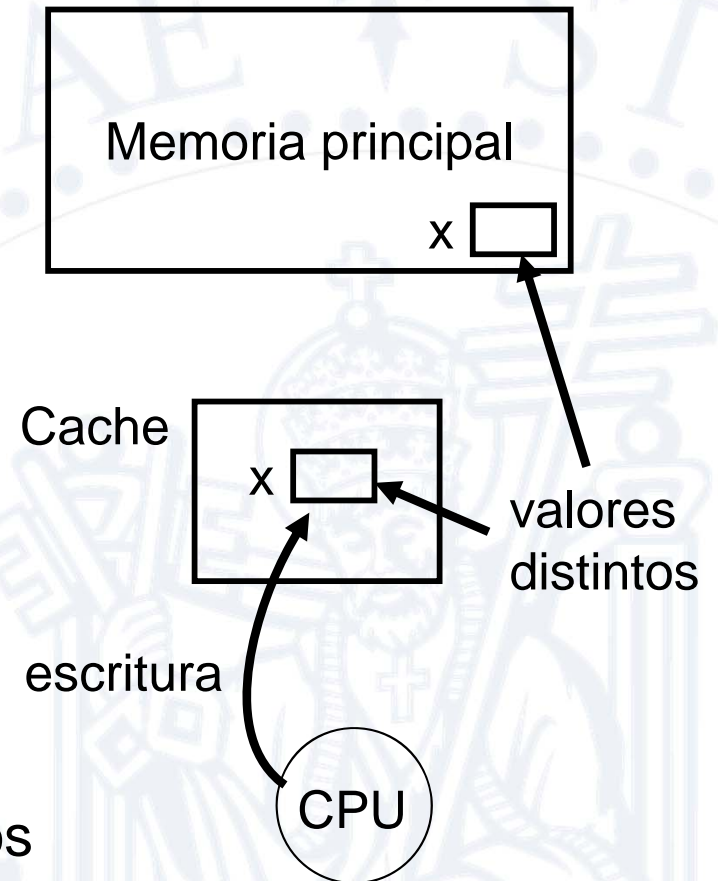
- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



¿qué bloque se expulsa?

Parámetros de diseño

- Capacidad de la cache
 - Cuántos bloques puede albergar
- Tamaño de bloque
 - Cuántos bytes tiene un bloque
- Correspondencia
 - Regla de colocación y localización
- Algoritmo de reemplazo
 - Si la cache está llena, qué bloque expulsamos
- Política de escritura
 - Qué ocurre cuando el procesador escribe



Resumen

- Memorias: las DRAM son muy lentas, las SRAM muy caras
- Localidad: los programas acceden a memoria con alta localidad espacial y temporal
- Jerarquía de memoria: almacenes cercanos al procesador rápidos y pequeños, almacenes lejanos lentos y grandes
- Memoria cache: memoria pequeña y rápida que contiene el subconjunto más usado de memoria principal
- Modelo de prestaciones

$$T_{ef} = T_a + m \cdot T_p$$

Ejercicio de clase

- Calcular la tasa de fallos de una cache de datos.
Suponer que:
 - El tamaño de bloque es de K elementos
 - El número de bloques de la cache de datos (capacidad) es menor que el número de filas de la matriz A

Alg. 1

```
for (i = 0; i < max; i++)  
  for (j = 0; j < max; j++)  
    A[i][j] = 0;
```

Alg. 2

```
for (i = 0; i < max; i++)  
  for (j = 0; j < max; j++)  
    A[j][i] = 0;
```