

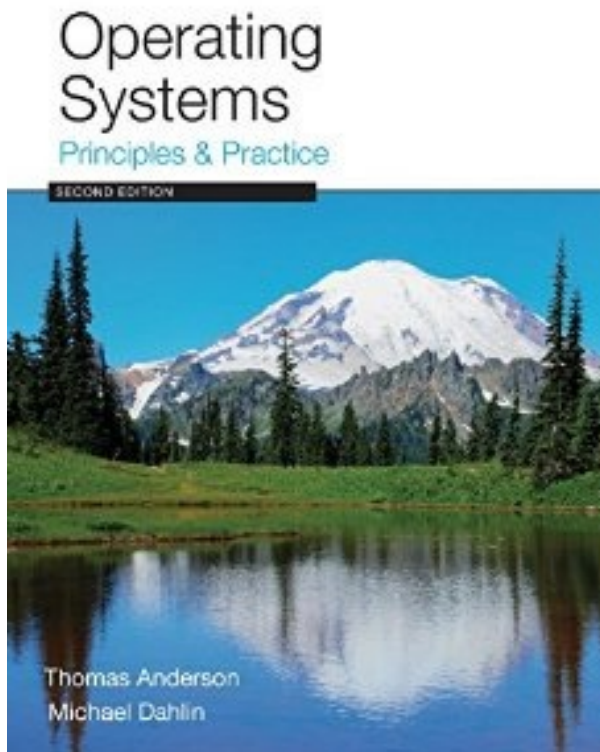


Sistemas Operativos

Capítulo 9 Memoria Caché y Memoria Virtual

Prof. Javier Cañas R.

Estos apuntes están tomados en parte del texto:
“Operating System: Principles and Practice” de T.
Anderson y M. Dahin



Temario

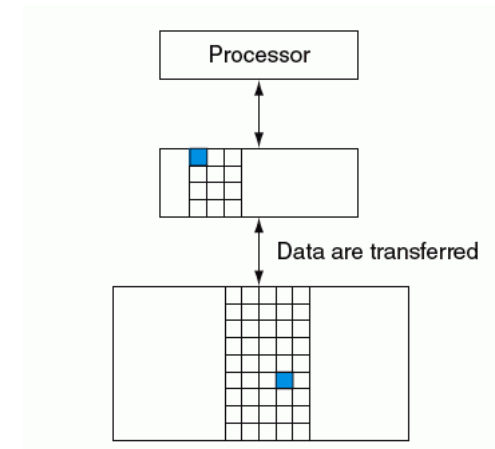
1. Introducción a las memorias caché
2. Conceptos y definiciones
3. Búsquedas en caché
4. Políticas de reemplazo
5. Archivos mapeados en memoria
6. Memoria Virtual

1 Introducción a las memorias caché

- La principal razón de tener una jerarquía de memoria es el desempeño.
- El sistema de memoria de un SO, considera otros aspectos del sistema computacional como I/O, generación de código por parte de los compiladores y diseño de aplicaciones.
- La utilización de caché se basa en la localidad espacial y temporal de los programas.

Estructura básica de una jerarquía de memoria

Speed	CPU	Size	Cost (\$/bit)	Current Technology
Fastest	Memory	Smallest	Highest	SRAM
	Memory			DRAM
Slowest	Memory	Biggest	Lowest	Magnetic Disk



Una caché simple

- En la figura la caché tiene bloques de una palabra. El procesador solicita un bloque y la caché contiene los requerimientos anteriores:

Antes de referenciar X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

Después de referenciar X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

Caché Direct Mapped

- ¿Cómo sabemos si un dato está en la caché?. Si sabemos que está, ¿cómo lo recuperamos?.
- El mapeo normalmente es:

(Direc. bloque) **mod** (número total de bloques en la caché)

Ejemplo

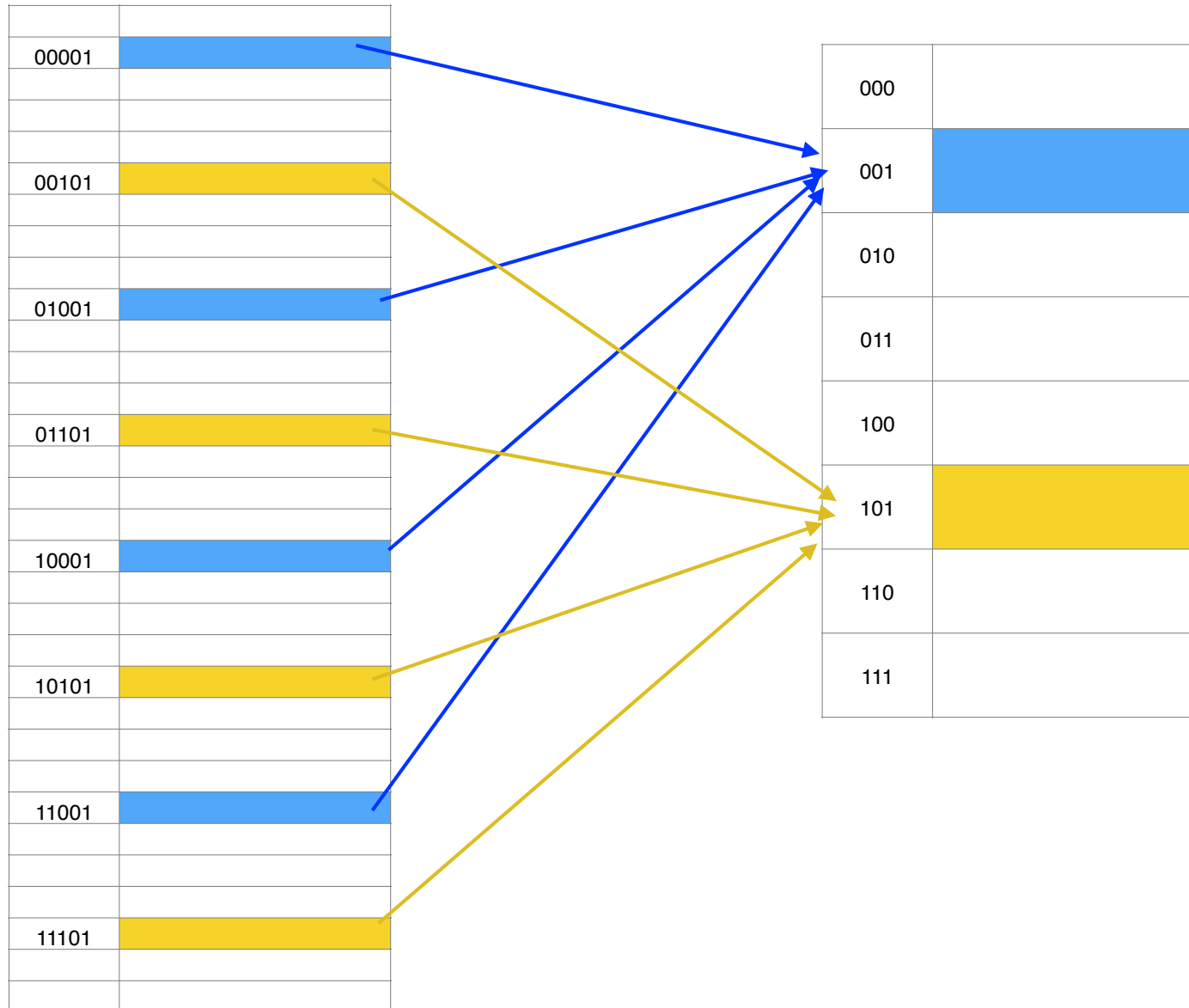
- Caché con 8 palabras. Las direcciones de memoria están entre 1_{10} (00001_2) y 29_{10} (11101_2) mapeadas a 1_{10} (001_2) y 5_{10} (101_2).

Memoria Principal

00001	
00101	
01001	
01101	
10001	
10101	
11001	
11101	

Caché

000	
001	
010	
011	
100	
101	
110	
111	



La palabra solicitada, ¿está en la caché?

- Cómo el mapeo es muchos a uno, para saber si la palabra solicitada está o no en la caché se agrega un conjunto de *tags* (etiquetas).
- El *tag* necesita sólo los bits más significativos de la dirección, es decir, los bits que no se utilizan como índices de la caché.
- En el ejemplo anterior el tamaño de la caché es 8, luego se necesitan $\log_2(8)=3$ bits. Entonces la dirección 10001_2 , está en la dirección de caché 001 y su *tag* es 10.

Estructura de caché de mapeo directo

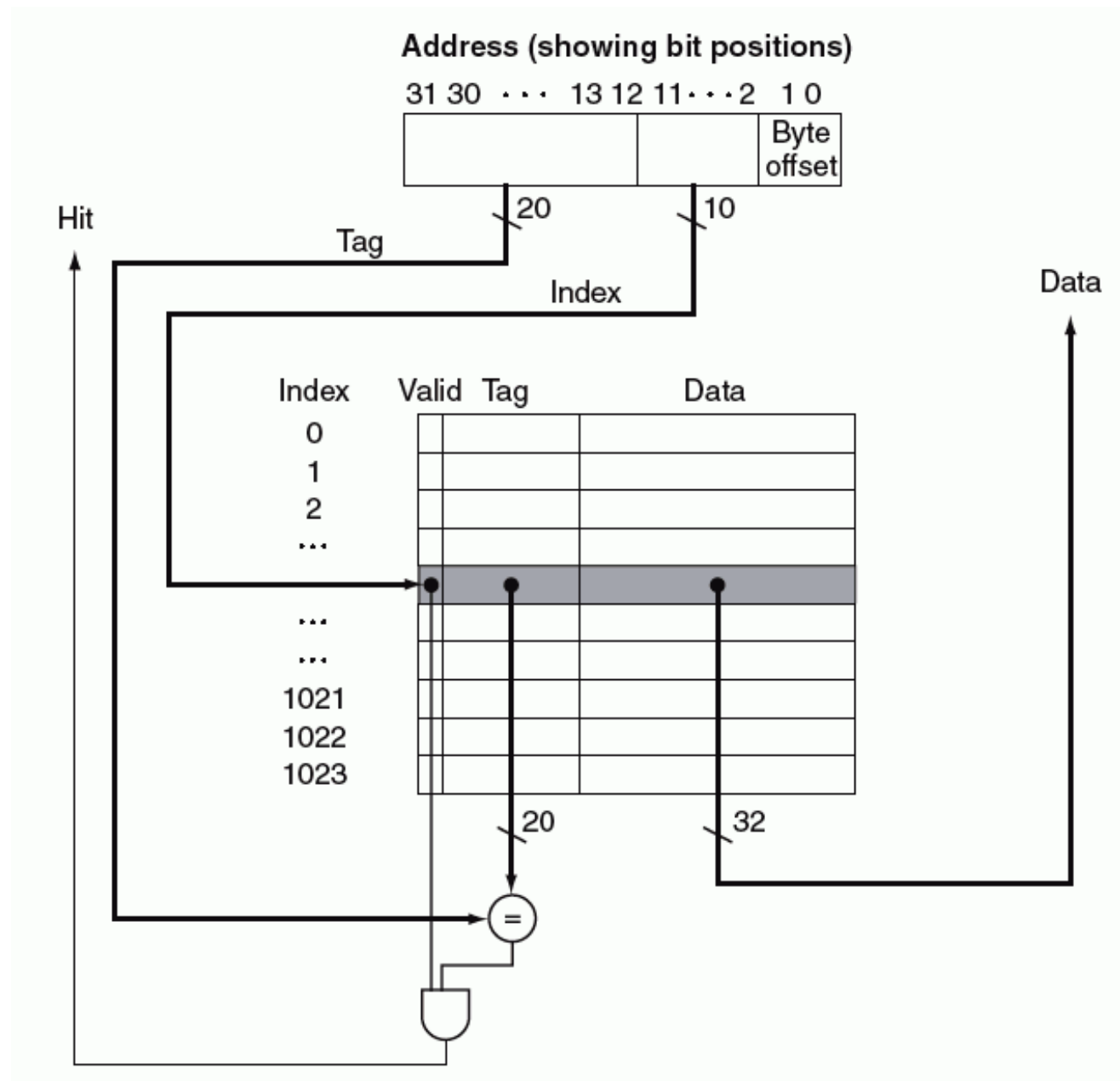
Se muestra el contenido después de manejar un miss de la dirección 10110

Índice	V	Tag	Dato
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Memoria(10110)
111	N		

Ejercicio

Considerando la misma caché anterior, se generan las siguientes referencias a memoria. Establecer como va cambiando el contenido de la caché. ¿Cuántas fallas de Caché se producen?

Dirección decimal	Dirección binaria
22	10110
26	11010
22	10110
26	11010
16	10000
3	00011
16	10000
18	10010



Quiz

- ¿Cuántos bits se requieren en una caché direct-mapped de 16KB de datos y bloques de 4 palabras, asumiendo direcciones de 32b?
- **Sol:**
 - ▶ 16KB son 4Kpalabras, 2^{12} como el tamaño de bloque es de 4 palabras (2^2), hay 2^{10} bloques.
 - ▶ Cada bloque tiene 4×32 o sea, 128b de datos + tag.
 - ▶ El tag tiene= $32-10-2-2$ más un bit de validez.
 - ▶ El tamaño total es= $2^{10} \times (128+(32-10-2-2) + 1)=2^{10} \times 147=147\text{Kbits}$ o 18.4KB para una caché de 16KB
 - ▶ 1.15 veces lo necesario para almacenar datos.

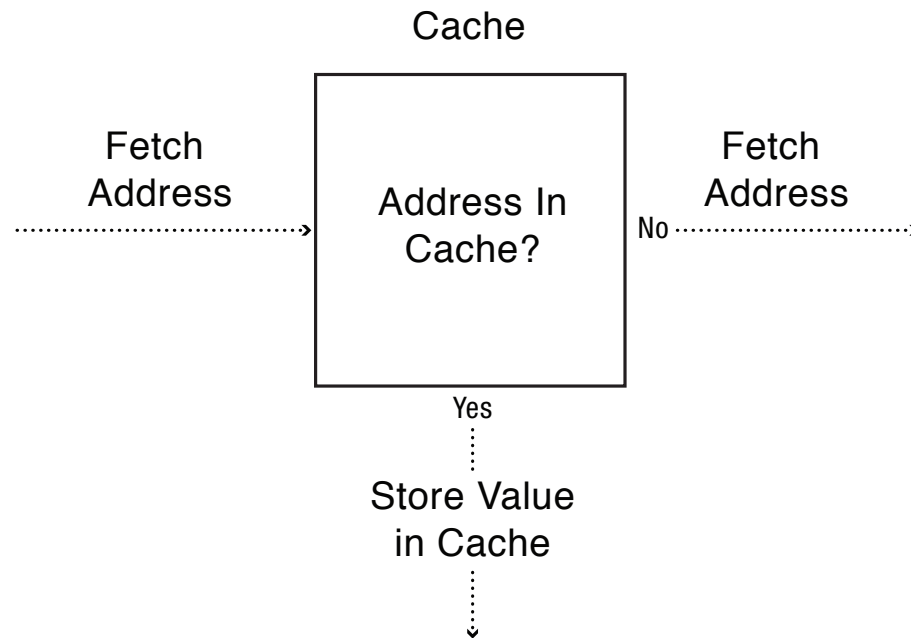
2 Conceptos y Definiciones

- Una caché es una copia de programas o datos que puede ser accesada en menos tiempo que la original.
- Las caches funcionan porque tanto los usuarios y los programas tienen un comportamiento predecible.
- Ejemplos de caché:
 - La TLB: búsquedas rápidas en estructuras de páginas multinivel.
 - Caché de direcciones virtuales: Cada entrada almacena el valor de memoria asociado a una dirección virtual. (Nivel1)
 - Caché de direcciones físicas: Cada entrada almacena el valor de memoria asociado a una dirección física. (Nivel 2 o 3)

Caché simple

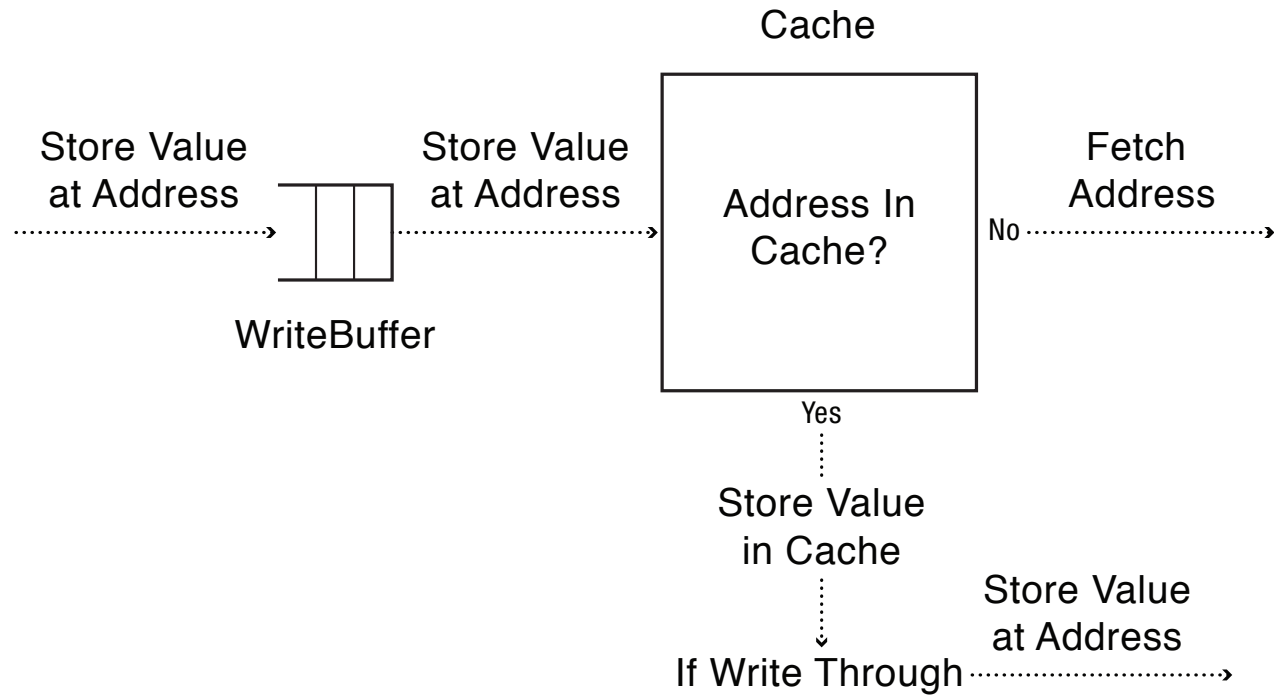
- Lo más simple es una memoria caché que almacena pares dirección-valor.
- Para que sea útil: el costo de recuperar datos de la caché debe ser significativamente menor que recuperar desde memoria, y la tasa de éxito (*hit*) debe ser alta.
- **Localidad temporal**: los programas tienden a referenciar las mismas instrucciones y datos recientemente accedidos.
- **Localidad espacial**: Los programas tienden a referenciar datos cerca de otros datos recientemente accedidos.

Lectura



$$\text{Latencia (read)} = \text{Pr(caché hit)} \times \text{Latencia (caché hit)} + \text{Pr(caché miss)} \times \text{Latencia (caché miss)}$$

Escritura



Escritura de caché

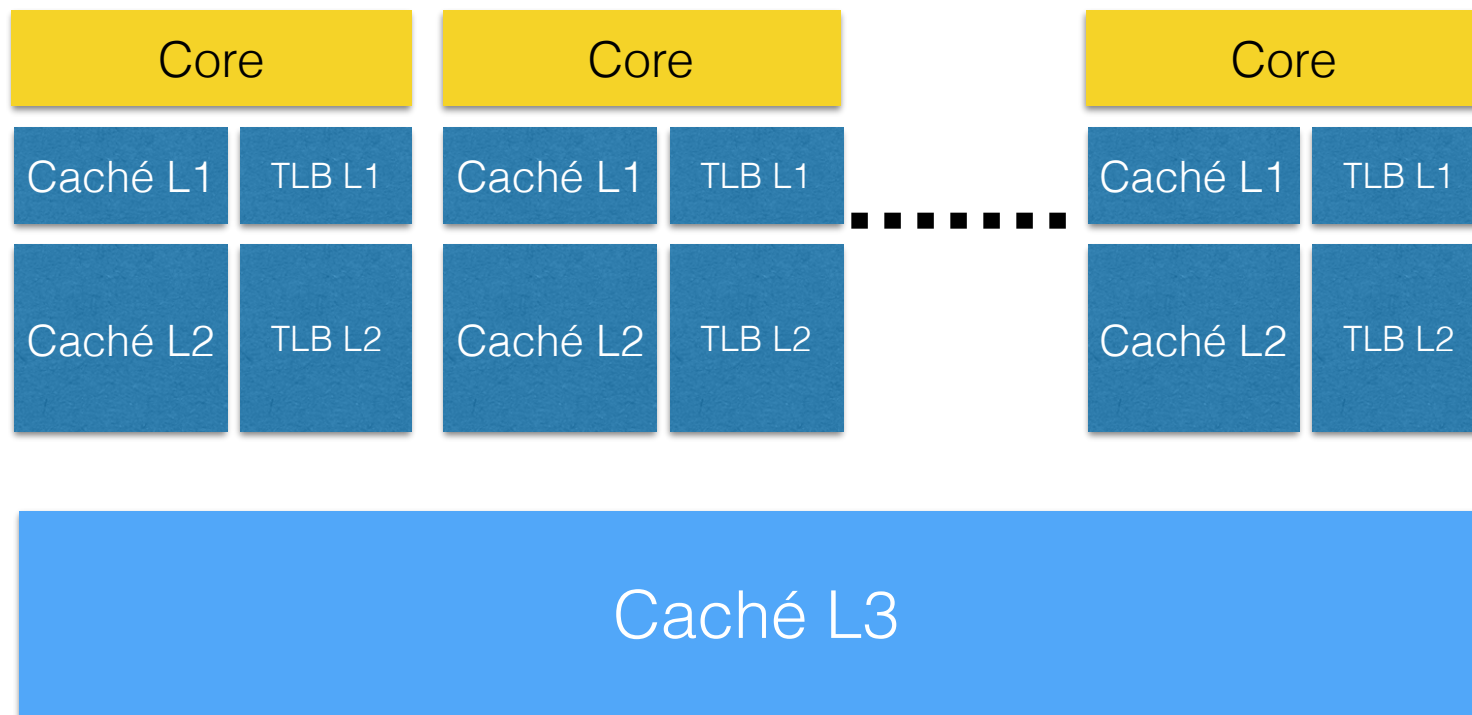
- Muchos sistemas escriben en un buffer, lo que permite escribir inmediatamente y una hebra del kernel en trasfondo escribe en la caché.
- Al utilizar buffer, al leer se debe verificar el buffer y la caché.
- La caché puede ser:
 - **write-through**: todas las actualizaciones se envían inmediatamente a la memoria.
 - **write-back**: todas las actualizaciones se almacenan en la caché y sólo se envían a la memoria cuando no hay más espacio.

Jerarquía de memoria

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μ s	100 TB
Local non-volatile memory	100 μ s	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

El Intel i7 tiene 8MB como caché nivel 3 compartida. El nivel 2 es por núcleo

Intel i7



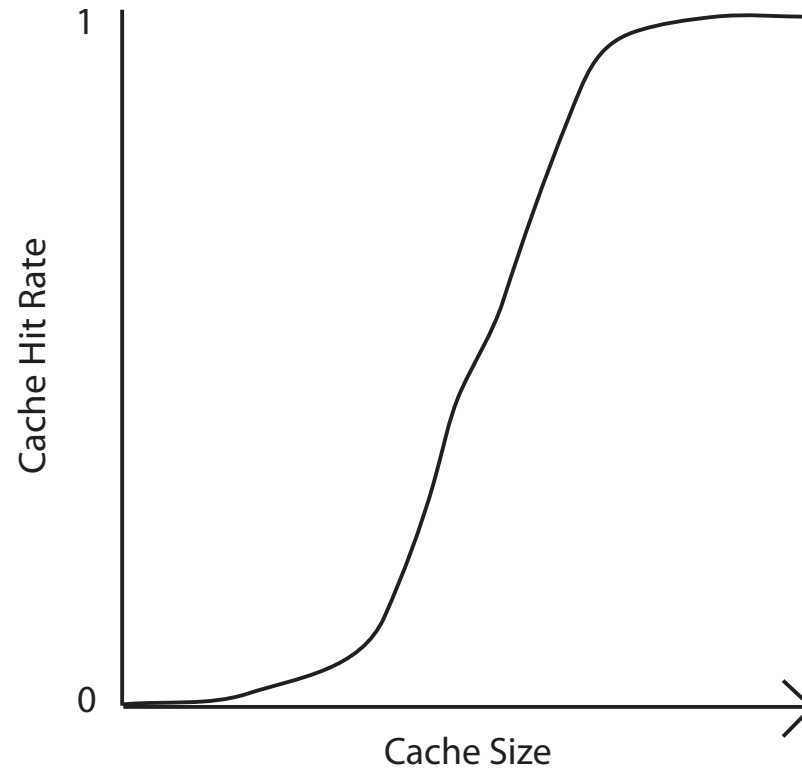
Efectividad de la caché

- Un mismo programa podrá tener diferente comportamiento de caché dependiendo de como se use.
- Por ejemplo un programa que escribe en una tabla de hash. Si la tabla cabe en la caché de nivel 1, el acceso será rápido. Si es muy grande requerirá muchos accesos a disco.
- No sólo la dependencia es con el tamaño de la caché.

Modelo Working Set

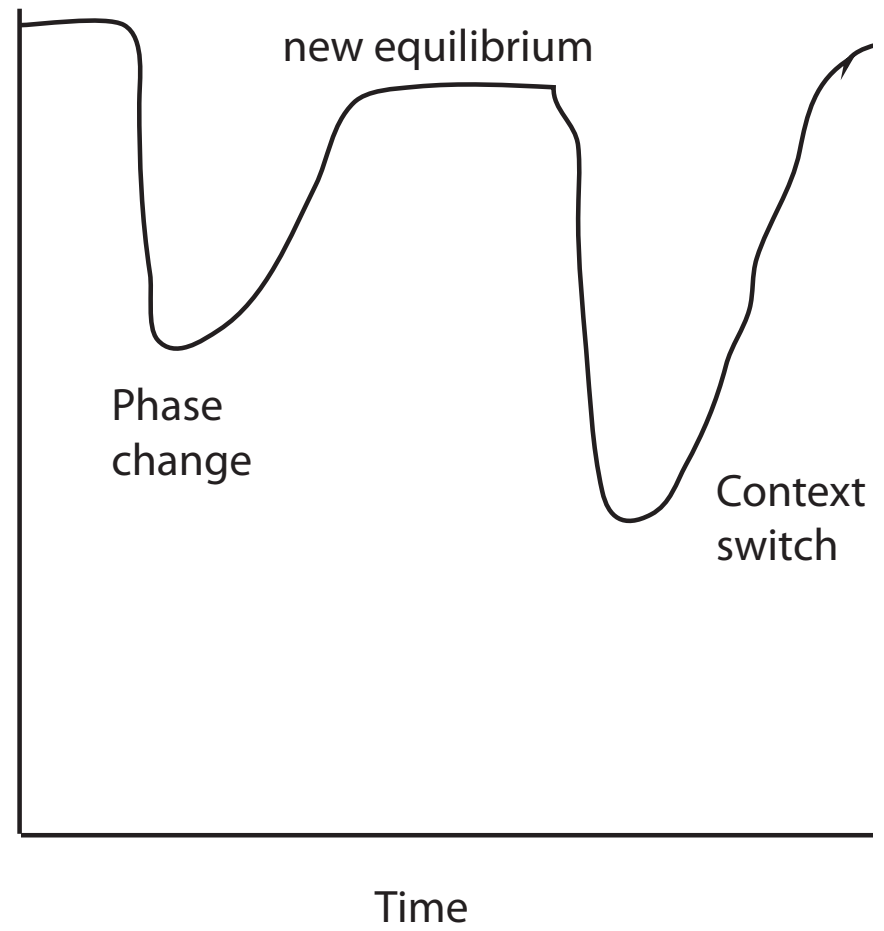
- El hit rate está asociado directamente con el tamaño de la caché.
- Todo programa exhibe un punto de inflexión en el cual la masa crítica de un programa entra completamente en la caché.
- **Working Set:** localizaciones de memoria que se necesitan para tener un hit rate adecuado.
- **Thrashing:** Cuando la caché es muy pequeña toda referencia genera un miss.

... Modelo Working Set



Cambio de fase en working set

- Los programas cambian dinámicamente su working set.
- El context switch también lo cambia



3 Configuraciones de cachés

- Una memoria caché mapea un conjunto disperso de direcciones a valores de datos almacenados en estas direcciones.
- La caché se puede ver como una gran tabla que tiene dos columnas: <dirección> <dato>.
- Para explotar bien la localidad espacial, cada entrada almacena los valores de un bloque de memoria, no sólo una palabra simple.

... Configuraciones de cachés

- Procesadores modernos utilizan bloques de 64B. Para sistemas operativos, el tamaño del bloque es típicamente el tamaño de una página (4KB en procesadores Intel)
- Existen cachés del SO y cachés de HW.

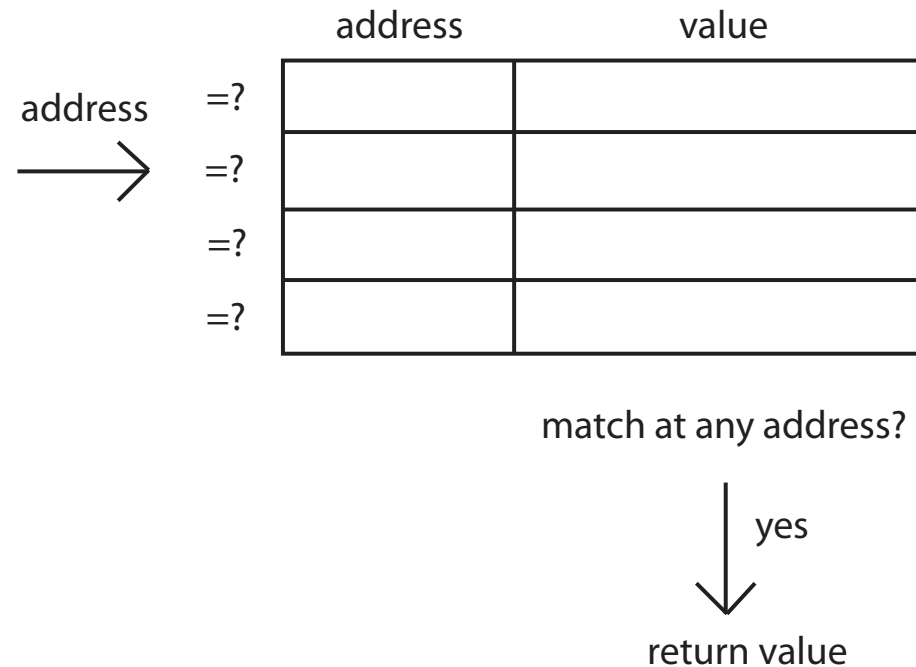
Búsquedas en cachés

- Para el SO las opciones de búsqueda son: lista enlazada, árboles multinivel y tablas de hash.
- Las cachés de HW tienen mayores restricciones. El overhead debe ser mínimo. Tradeoff: rapidez en la búsqueda vs. costo de caché miss.
- Las caché de HW se organizan de 3 formas:
 - ***Totalmente asociativas.***
 - ***Direct mapped***
 - ***Set associative***

Cachés totalmente asociativas

- La dirección se puede almacenar en cualquier parte de la tabla. La búsqueda es por contenido y se hace en paralelo.
- Ejemplo: la TLB es totalmente asociativa. También la memoria física es totalmente asociativa ya que cualquier “page frame” puede contener cualquier página virtual, pero en este caso la búsqueda no es paralela.

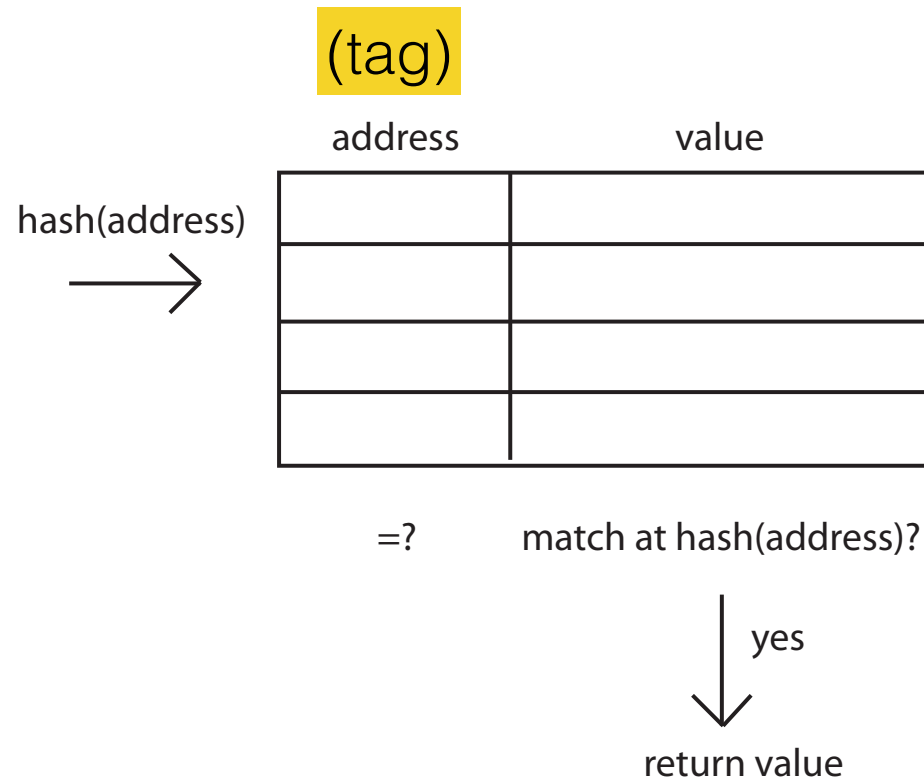
Caché totalmente asociativa



Caché de mapeo directo

- Corresponde a la caché descrita en el punto 1 “Introducción a las memorias caché”.
- Una dirección sólo puede ser almacenada en una entrada de la tabla. La búsqueda es simple aplicando una función de hash. Hay un hit si hay un calce con la entrada de la tabla.
- La búsqueda es eficiente, pero puede ocurrir que dos direcciones diferentes y muy utilizadas, tengan el mismo valor de hash.

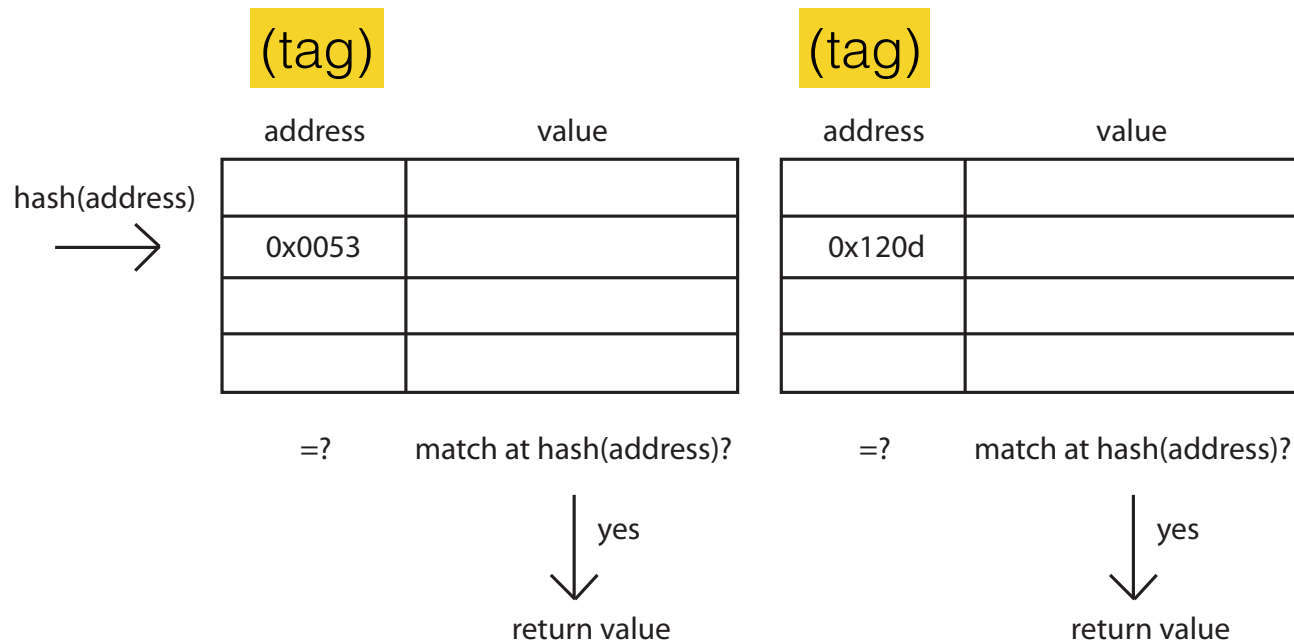
Caché de mapeo directo



Caché set asociativa

- Fusiona las dos estrategias anteriores logrando una búsqueda un poco más lenta que con mapeo directo, pero mantiene la flexibilidad de la caché totalmente asociativa.
- Una caché k -set asociativa tiene k replicas. Un bloque particular puede estar en cualquiera de las k réplicas.
- Equivale a una tabla de hash con bucket de tamaño k .

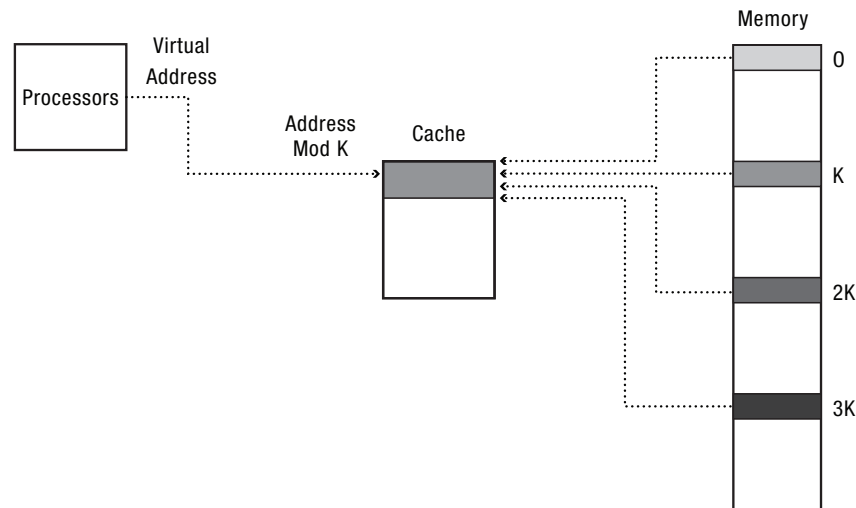
Caché set asociativa



Actualmente todas las cachés de HW y TLB son set asociativas utilizando 8 conjuntos

Coloreo de páginas

- Si el tamaño de la caché es mayor que el tamaño de una página, podría ocurrir que múltiples páginas queden mapeadas en un mismo lugar de la caché:



... Coloreo de páginas

- El desempeño puede deteriorarse considerablemente.
- Para mejorar el desempeño de cachés, los SO utilizan el concepto de coloreo de páginas.
- **Coloreo de páginas:** un frame de página es particionado en conjuntos basado en cuál bucket utilizará.

- Ejemplo: caché set asociativa de 2MB 8-way.
- Páginas de 4KB
- 2MB se direccionan con 21b. Como la página tiene 4KB, tiene 12b de dirección.
- $21b - 12b = 9b$. Si la memoria fuera 1-way, 1 de cada $2^9 = 512$ páginas caerían en el mismo lugar de la caché, pero como es de 8-way, 1 de cada $512/8 = 64$ páginas caen en el mismo lugar. Entonces hay 64 conjuntos separados o colores.
- El SO asigna frames para esparcir páginas en los distintos colores.

4 Políticas de reemplazo

- Si en el proceso de búsqueda nos encontramos con un **miss**, ¿qué bloque elegimos para reemplazar?.
- Con caché de mapeo directo, no hay opción de elección.
- En general, se puede elegir, y esta elección tiene gran impacto sobre la tasa de hit de la caché.
- No hay una respuesta simple!.

... Políticas de reemplazo

- Para cachés de HW hay más restricciones y las decisiones se deben tomar en forma muy rápida.
- Para el SO hay más tiempo y más opciones. Se puede ver la memoria principal como caché de disco.
- Aún dentro del SO, las políticas para el buffer caché de archivos es distinta que para memoria virtual paginada bajo demanda.
- Las políticas son: RANDOM, FIFO, MIN, LRU, LFU.

Reemplazo RANDOM

- Consiste en tomar un bloque en forma aleatoria para reemplazar.
- Puede ser útil para cachés de nivel 1 donde hay muy poco tiempo para actuar.
- En promedio no será muy malo, pero es impredecible y puede frustrar una aplicación diseñada para utilizar cuidadosamente los diferentes niveles de caché.

Reemplazo FIFO

- Una política razonable es sacar el bloque (o página) que ha estado más tiempo en la memoria.
- Lamentablemente FIFO tiene el peor comportamiento posible para cargas de trabajo que se dan en forma frecuente, por ejemplos ciclos sobre un arreglo que es muy grande para entrar totalmente en la caché.

FIFO

Reference	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
1	A				E				D				C		
2		B				A				E				D	
3			C				B				A				E
4				D				C				B			

Reemplazo MIN (Óptimo)

- ¿Cuál política de reemplazo es la óptima para minimizar miss de caché?.
- MIN consiste en reemplazar el bloque que será utilizado más lejanamente en el futuro.
- Si pudiéramos predecir el futuro, se podría hacer algo mucho mejor: Lograr que los bloques lleguen “just in time”, exactamente cuando se necesiten.

Reemplazo LRU

- Una manera de predecir el futuro es mirar al pasado. Las localidades se mantienen (por un tiempo).
- LRU significa elegir el bloque que sea menos recientemente usado.
- La implementación es simple (pero poco eficiente). Se mantiene una lista. En cada hit se mueve el bloque al frente de la lista y en cada miss, se remueve el bloque que está al final de la lista.

... LRU

- En algunos casos LRU se aproxima al óptimo, pero podría ser también un pésimo algoritmo, por ejemplo cuando el menos recientemente usado es el siguiente en ser referenciado.
- La siguiente figura muestra el comportamiento de 3 algoritmos utilizando el mismo patrón de referencia.

LRU															
Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
1	A		+				+				+			+	
2		B			+								+		
3				C					E			+			
4						D		+		+					C
FIFO															
1	A		+				+		E						
2		B			+						A			+	
3				C								+	B		
4						D		+		+					C
MIN															
1	A		+				+				+			+	
2		B			+								+		C
3				C					E			+			
4						D		+		+					

+: indica un hit de caché

Exploración secuencial de memoria con LRU y MIN

LRU															
Reference	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
1	A				E				D				C		
2		B				A				E				D	
3			C				B				A				E
4				D				C				B			

MIN															
1	A					+					+			+	
2		B					+					+	C		
3			C					+	D					+	
4				D	E					+					+

LRU se comporta igual que FIFO

LFU Menos frecuentemente usado

- Una estrategia que sigue una distribución de Zipf es LFU.
- LFU descarta el bloque que ha sido usado con menos frecuencia, es decir mantiene las páginas más populares aún cuando páginas menos populares han sido referenciadas recientemente.
- Tanto LRU como LFU intentan predecir el futuro.

Anomalía de Belady

- Intuitivamente, si se aumenta el tamaño de la caché se mejora la tasa de hit. Para muchas políticas de reemplazo, esta intuición es cierta, pero en algunos casos falla.
- El algoritmo de reemplazo FIFO presenta esta anomalía como se puede apreciar en las siguientes secuencias de referencias.

FIFO: Anomalía de Belady

FIFO (3 slots)												
Reference	A	B	C	D	A	B	E	A	B	C	D	E
1	A			D			E					+
2		B			A			+		C		
3			C			B			+		D	
FIFO (4 slots)												
1	A				+		E				D	
2		B				+		A				E
3			C						B			
4				D						C		

5 Paginación bajo demanda

- Con paginación bajo demanda, las aplicaciones pueden tener acceso a más memoria de la físicamente presente en la máquina. Para esto se usan páginas de memoria (frames) como caché de bloques de disco.
- Cuando una aplicación accesa una página que no está en memoria (miss), el SO transparentemente la trae del disco. Esta técnica se puede utilizar en **Archivos Mapeados en Memoria** y en **Memoria Virtual**.

Archivos mapeados en memoria

- La mayoría de los programas utilizan llamadas al sistema explícitas para realizar I/O sobre archivos. Estas llamas al sistema son **read** y **write** y para esto:
 - Un programa hace un **open** a un archivo y luego los datos son copiados al espacio usuario (**read**)
 - El programa modifica los datos sin afectar el archivo.
 - Para escribir los cambios en el archivo se utiliza **write**.
- Una alternativa es utilizar archivos mapeados en memoria utilizando paginación bajo demanda.

... Archivos mapeados en memoria

- En un archivo mapeado en memoria, el SO genera la ilusión que el archivo es un segmento de programa y se puede leer y escribir utilizando instrucciones `lw` y `sw`. La memoria es entonces una caché (tipo write back) del disco.
- El proceso es similar a cargar una imagen ejecutable en memoria. El SO crea la tabla de página que apunta a los frames de memoria y puede comenzar la ejecución cuando el primer frame es inicializado.
- Normalmente un archivo mapeado en memoria tendrá una tabla de página simple compartida entre todos los procesos que utilizan el mismo archivo.

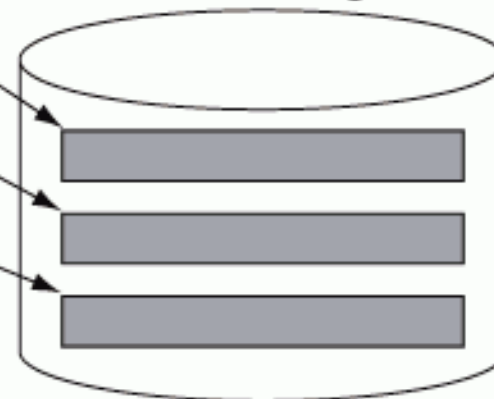
Virtual page number

Page table
Valid Physical page or
 disk address

1	●
1	●
1	●
1	●
0	●
1	●
1	●
0	●
1	●
1	●
0	●
1	●

Physical memory

Disk storage



Swap Partition

Ventajas

- **Transparencia:** El programa opera sobre el archivo como si fuera parte de la memoria.
- **Zero copy I/O:** no se necesita copiar datos del archivo entre los buffer del kernel a la memoria y vice versa. La copia es directa a los frames.
- **Pipelining:** El programa comienza a operar en el momento que la tabla esté inicializada sin necesidad de estar completa.
- **Comunicación entre procesos (IPC):** Dos o más procesos pueden compartir información instantáneamente sin necesidad de utilizar la asistencia del kernel ni I/O de disco.

Implementación

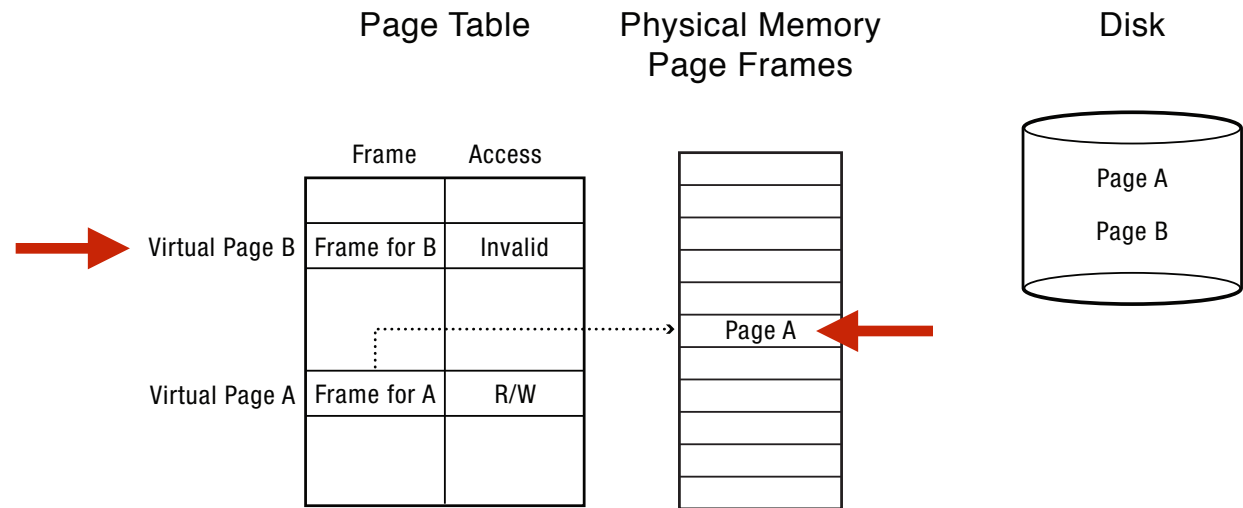
- El SO proporciona una llamada al sistema para mapear un archivo en una parte del espacio virtual del proceso (`mmap`).
- Al ejecutar esta llamada al sistema, el kernel inicializa entradas de la tabla de página de la región destinada a archivo marcando entradas de la tabla como inválidas. El kernel retorna a usuario.

... Implementación

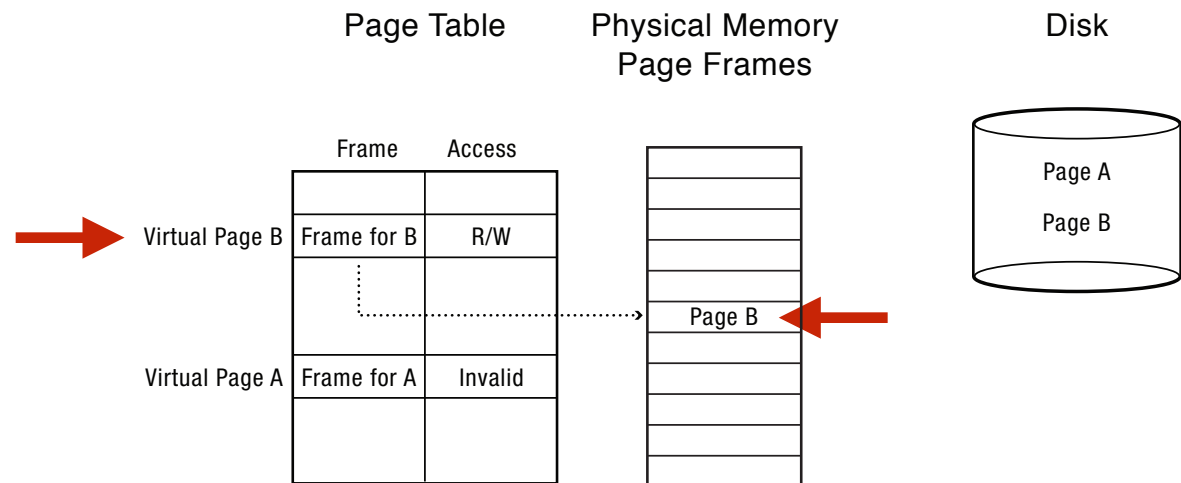
- Frente a una entrada inválida de la tabla de página:
 1. TLB miss: Este evento genera una búsqueda en HW.
 2. Excepción de tabla de página: El HW encuentra una entrada que es inválida. Trap al kernel.
 3. Conversión de dirección virtual a offset de archivo.
 4. Lectura de bloque de disco: el kernel asigna un frame de memoria vacío a un bloque de disco. Genera un wait.
 5. Interrupción de disco: El Scheduler reanuda el thread del kernel que maneja la excepción.
 6. Actualizar tabla de página: la tabla de página ahora apunta al frame de memoria donde está el bloque.

6. Actualizar tabla de página: la tabla de página ahora apunta al frame de memoria donde está el bloque.
7. Se reasume el proceso: El SO reasume el proceso en la instrucción que generó la excepción.
8. TLB miss: La TLB aún no contiene una entrada válida para la página disparando una búsqueda completa de la página.
9. Fetch de tabla de página: El HW hace nuevamente una búsqueda multinivel, encuentra ahora una entrada válida y retorna el número de frame al procesador. El procesador carga la TLB con la nueva conversión desalojando la entrada previa de la TLB

Antes de la
falla de página



Después de la
falla de página



Reemplazo de página

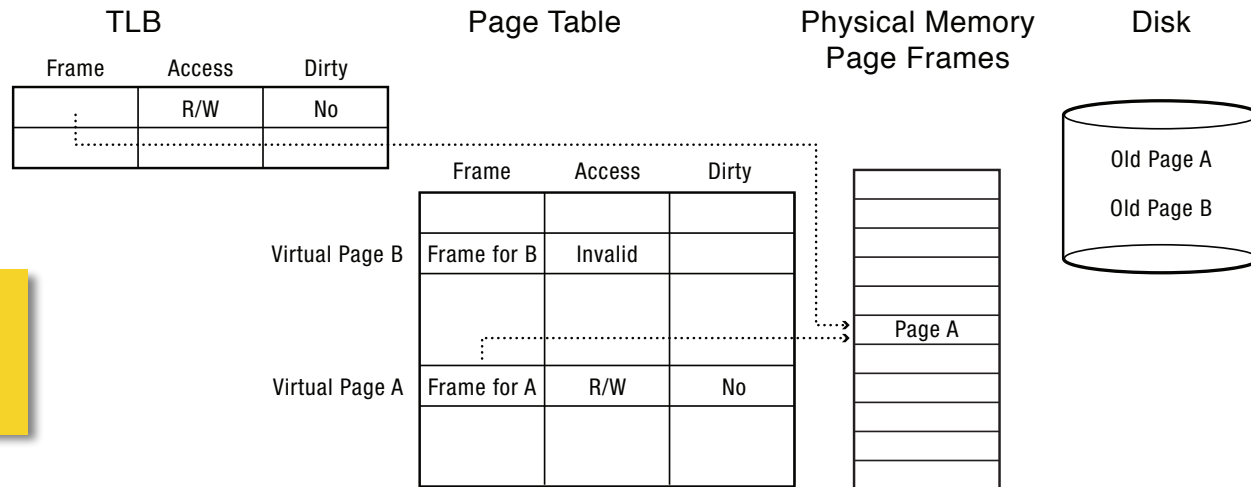
- Para crear un frame libre, el SO debe:
 1. Seleccionar una página para desalojar.
 2. Encontrar la entrada de la tabla de página que apunta a la página desalojada. Esto se hace con una estructura llamada core map, arreglo sobre cada página física, incluida entrada de tablas que tienen punteros a una página particular.
 3. Marcar la entrada de la tabla de página como inválida. Es necesario resetear la TLB.
 4. Escribir todos los cambios a disco de la página desalojada.

El bit “sucio”

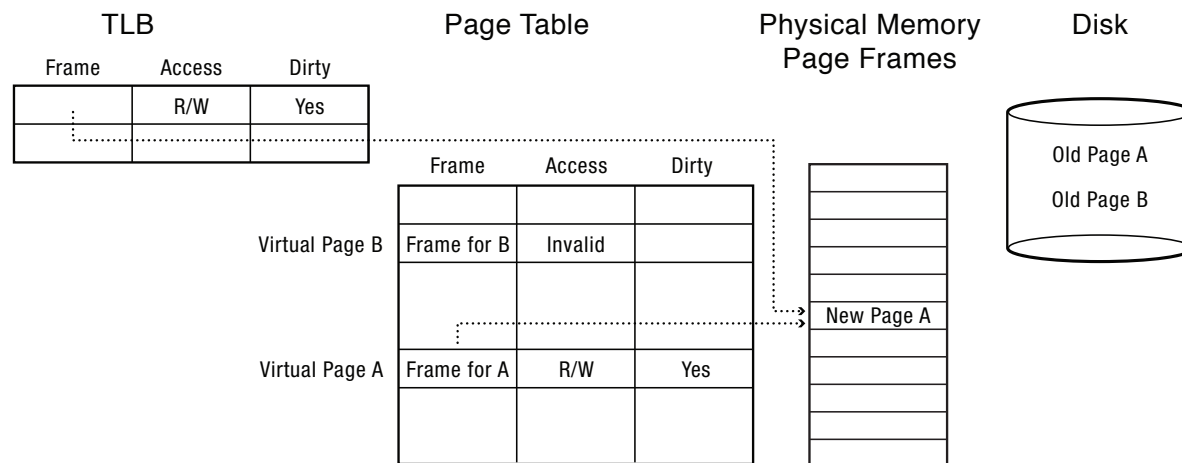
- ¿Cómo sabe el SO si una página ha sido modificada?. Una solución eficiente es incorporar una entrada en la tabla de página llamada “dirty bit”.
- El SO lo inicializa en cero, y el HW lo cambia automáticamente cuando de ejecuta una instrucción store en esa página virtual.
- Como la TLB contiene una copia de a entrada de la tabla de página, también incorpora un dirty bit.

Observar el bit sucio en uno tanto en la TLB como la tabla de página

Antes de escribir



Después de escribir

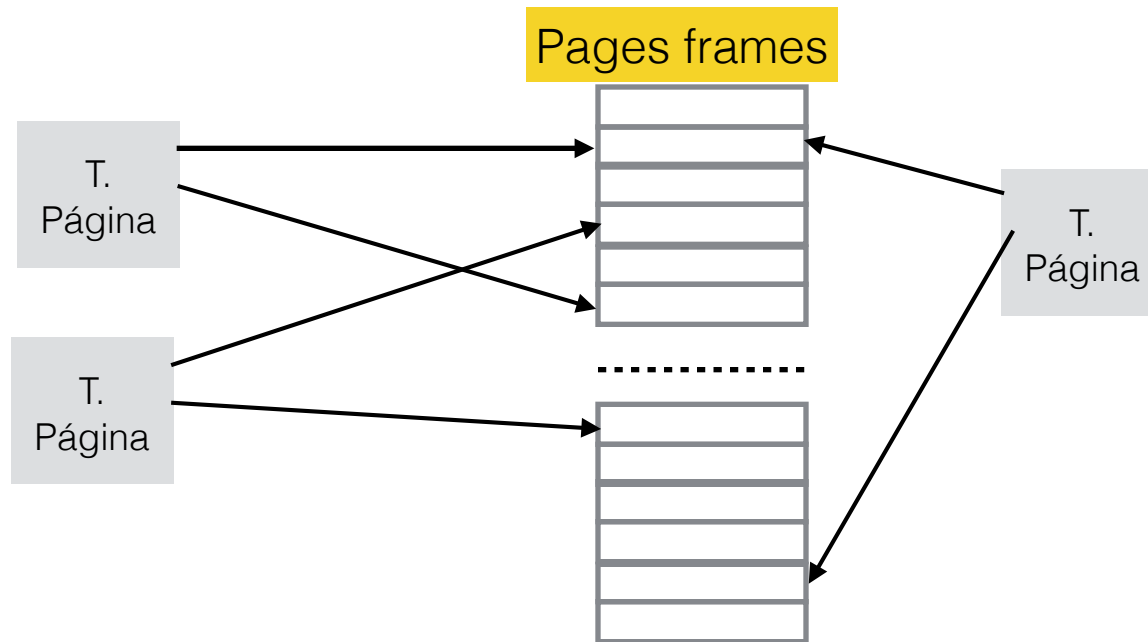


Aproximación a LRU

- LRU se aproxima al algoritmo MIN, pero el HW no lleva un registro de las páginas menos recientemente usadas.
- Implementar LRU con listas resulta ineficiente por el movimiento de punteros que se debe realizar.
- ***bit de uso***: muchos procesadores mantienen un bit de uso en cada página (junto al bit sucio). Cuando la página es inicializada, se pone en cero. Por HW se pone en uno cuando la página es llevada a la TLB.

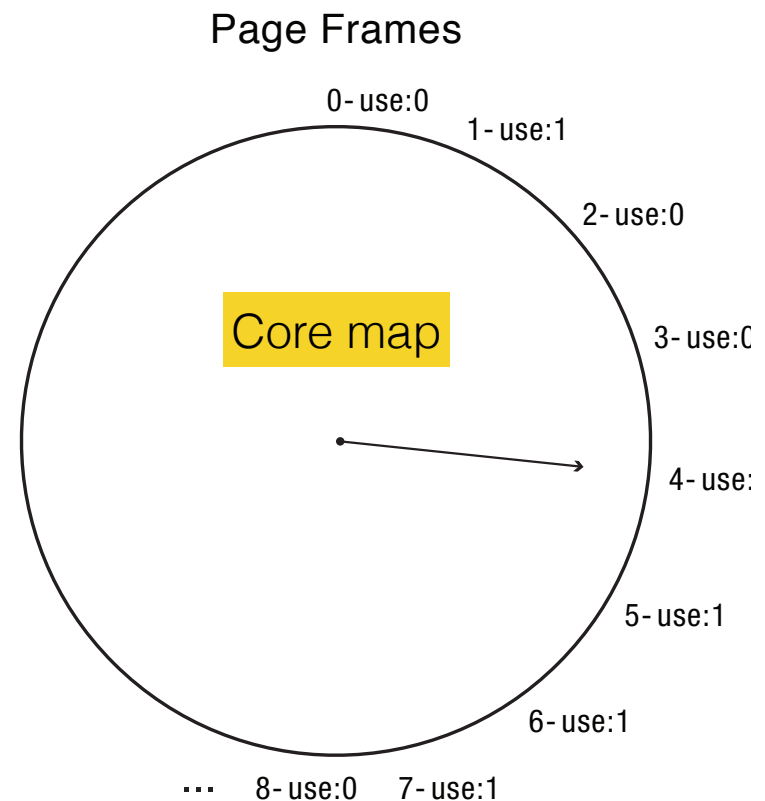
Algoritmo del reloj

- Una forma de sacar provecho del bit de uso es en el llamado “Algoritmo del Reloj”.
- Se utiliza el **core map**:



...Algoritmo del reloj

- Periódicamente el SO barre el core map de frames.
- Para cada frame, escribe en el core map, el valor del bit de uso de la T. de Página que apunta a ese frame y lo pone en cero.
- Si no está en uso, la página se puede sacar.
- Si está en uso marcarla como no usada y continuar con el siguiente.



6 Memoria Virtual

- Se generaliza el concepto de archivo mapeado en memoria asociando ahora segmentos de memoria con un archivo en disco. Esta técnica se denomina **Memoria Virtual**.
- Programas ejecutables, bibliotecas, datos estáticos, stack y segmentos del heap pueden demandar páginas al disco.
- A diferencia de los archivos mapeados en memoria, cuando un proceso termina, no hay necesidad de escribir datos modificados al disco.

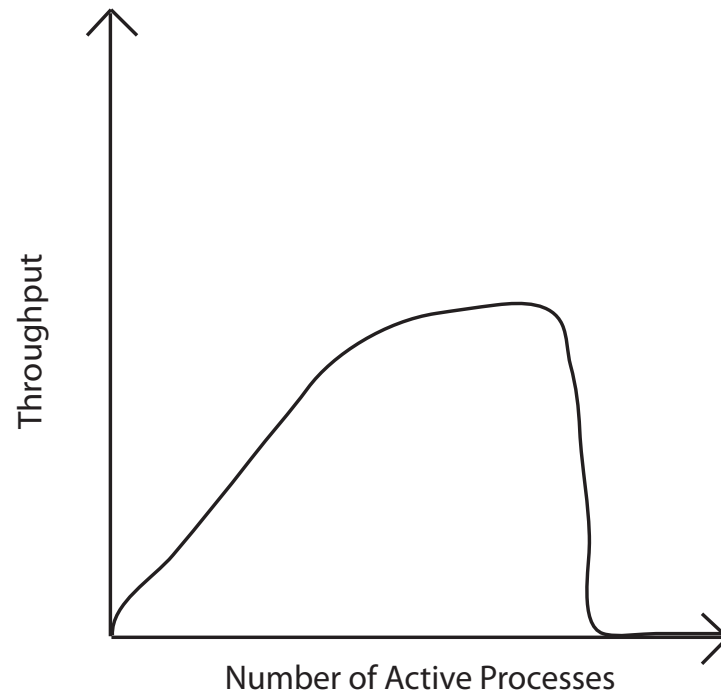
... Memoria Virtual

- La gran ventaja de la memoria virtual es su flexibilidad. El sistema puede continuar funcionando aún cuando el usuario puede activar más procesos de los que pueden entrar simultáneamente en memoria.
- Todos los mecanismos vistos para archivos son aplicables, con excepción de un detalle: Es necesario balancear la asignación de frames entre procesos.

Thrashing

- Un disco actual puede manejar a lo más 100 fallas de página por segundo,
- Un procesador multicore puede ejecutar 10^{10} instrucciones por segundo.
- ¿Qué pasa en el desempeño si se aumenta el número de procesos?.
- WS_i = (working set del Proceso P_i), M = tamaño memoria
- Si $\sum WS_i \geq M \Rightarrow$ Thrashing

Thrashing



Herramientas para conocer estado de memoria virtual

- `vm_stat` (Mac OS X). En Unix `vmstat`

[illegible]



Sistemas Operativos

Capítulo 9 Memoria Caché y Memoria Virtual

Prof. Javier Cañas R.