Dept. de Teoría de la Señal y Comunicaciones y Sistemas Telemáticos y Computación
Área de Telemática (GSyC)

Jerarquía de memoria

Katia Leal Algara
katia.leal@urjc.es
http://gsyc.escet.urjc.es/~katia/

Introducción

- Necesidad de una jerarquía de memoria
- Veremos:
  - Mecanismo de acceso de memoria en este tipo de jerarquía
  - Diseño y evaluación del rendimiento de la jerarquía de memoria
  - Aspectos tecnológicos y de diseño de los distintos niveles de la jerarquía de memoria

Diseño de una jerarquía de memoria básica

- El sistema de memoria de los primeros modelos de computador se descartó rápidamente por su bajo rendimiento
- En su lugar, hoy se utiliza una jerarquía de memoria organizada en niveles que incluye:
  - **Memoria caché**
  - **Memoria principal**
  - **Memoria virtual**
- Cada una:
  - se ubica físicamente en un lugar distinto
  - se fabrica con una tecnología diferente
  - se gestiona de manera independiente

Diseño de una jerarquía de memoria básica

- **Memoria caché (MC)**
- Ubicada en el mismo chip que el procesador
- Fabricada con memoria RAM estática (**SRAM**), *Static Random Access Memory*
  - Es un tipo de memoria basada en semiconductores que a diferencia de la memoria DRAM es capaz de mantener los datos sin necesidad de circuito de refresco. Sin embargo, sí son memorias volátiles, es decir que pierden la información si se les interrumpe la alimentación eléctrica
- Controlada por el **controlador de caché** incluido en el mismo chip
- Hoy en día existen varios niveles de caché
  - Hasta ahora hemos estado utilizando memoria caché de nivel 1

Diseño de una jerarquía de memoria básica

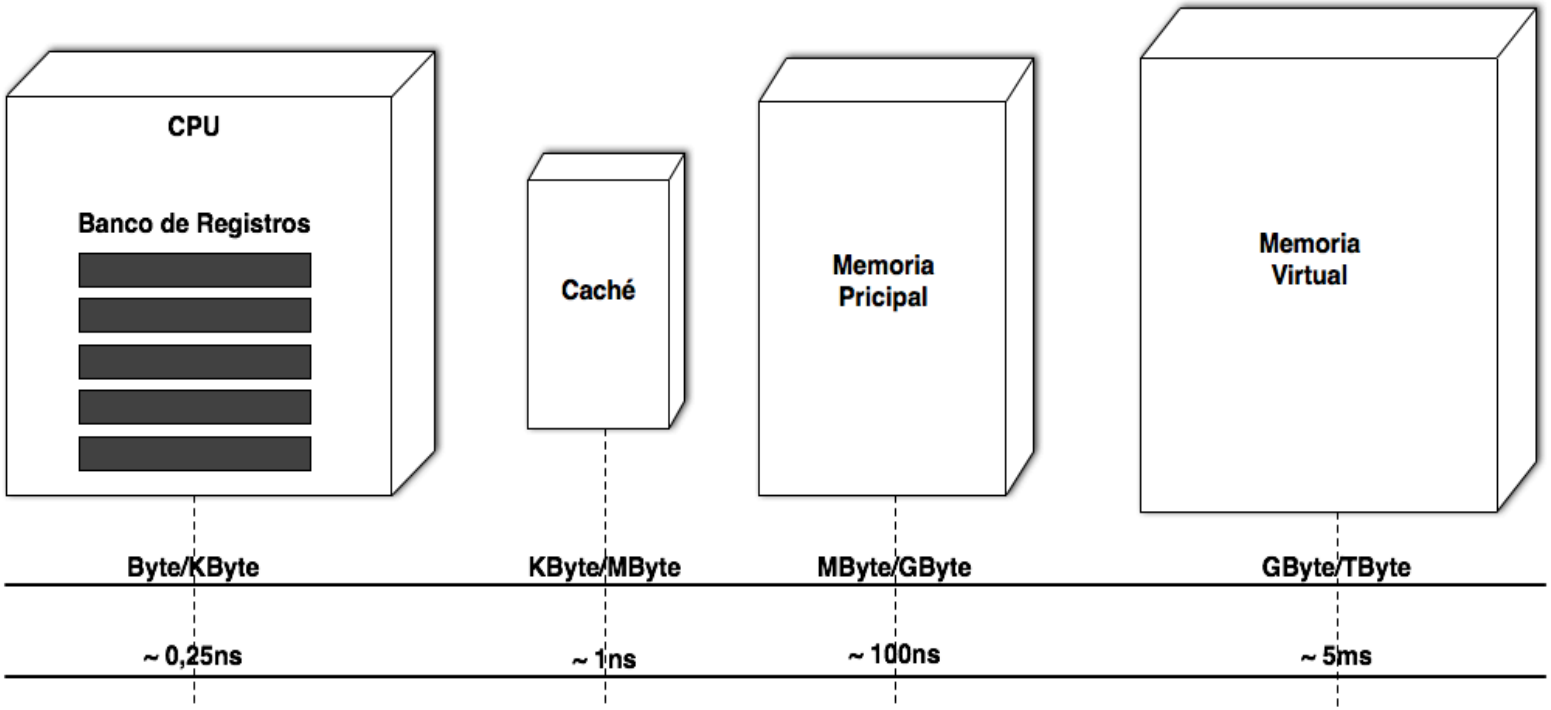
- **Memoria principal (MP)**
- Ubicada en un chip diferente al procesador
- Fabricada con memoria RAM dinámica (**DRAM**)**,** *Dynamic Random Access Memory*
  - Para mantener almacenado un dato, se requiere revisar el mismo y recargarlo cada cierto tiempo, en un ciclo de refresco. Su principal ventaja es la posibilidad de construir memorias con una gran densidad de posiciones y que todavía funcionen a una velocidad alta
- Controlada por el **controlador de memoria principal** que se encarga de planificar los accesos a la misma
- Hoy en día el controlador puede ubicarse en el mismo chip que el procesador y la memoria caché o en otro chip como el chipset norte o el MCH

Diseño de una jerarquía de memoria básica

- **Memoria virtual (MV)**
- Ubicada en el disco duro
- Fabricada con tecnología magnética
- Se controla desde el **sistema operativo** a través del controlador de disco duro

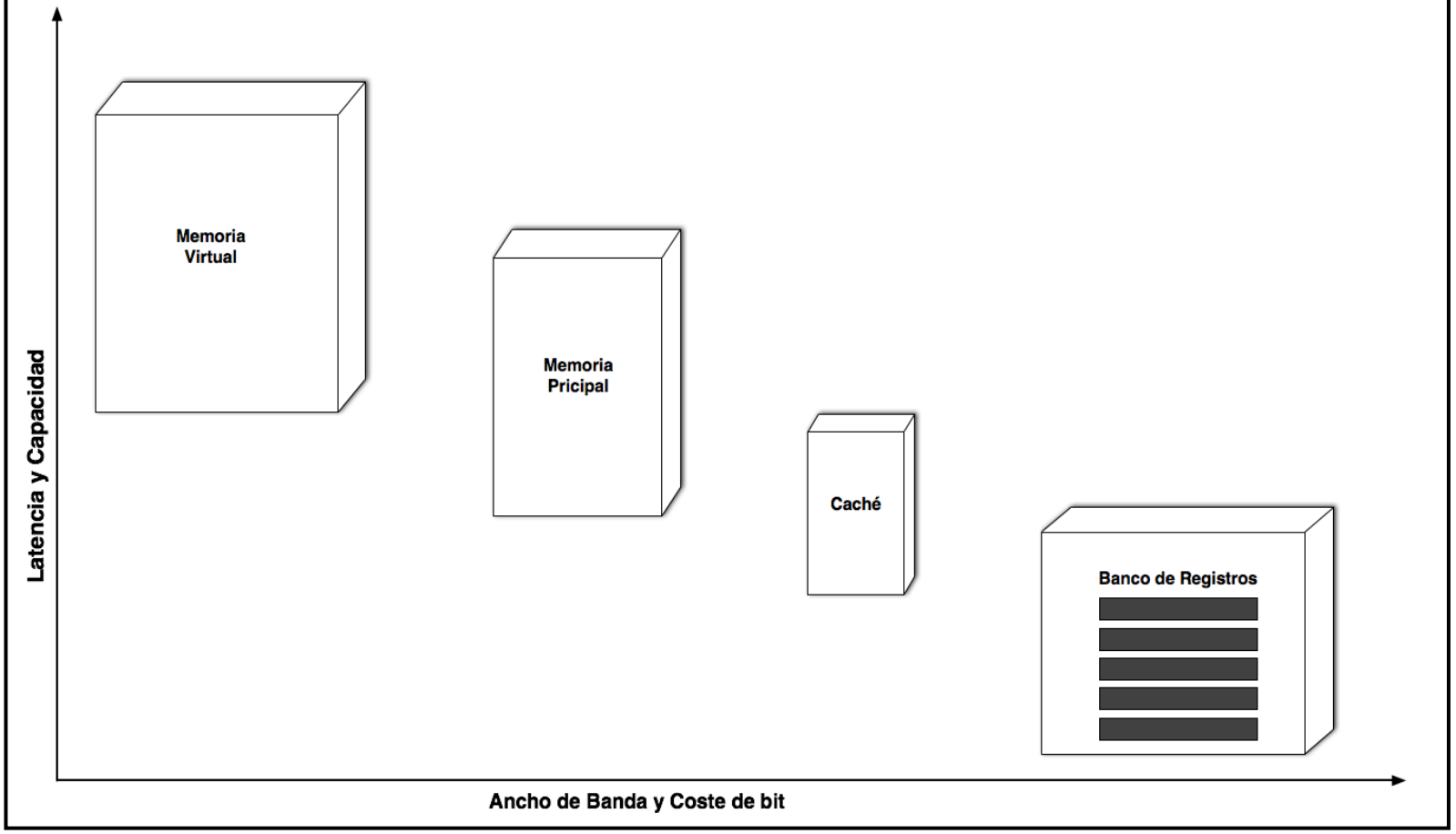
Diseño de una jerarquía de memoria básica

- Órdenes de magnitud de capacidades y tiempos de acceso



Diseño de una jerarquía de memoria básica

- Tecnologías y características de los diferentes niveles



Propiedades de una jerarquía de memoria

- **Inclusión**: cualquier información contenida en un nivel de la jerarquía debe estar también en los niveles superiores
- **Coherencia**: las copias de la misma información en los diferentes niveles son coherentes entre sí, es decir, que almacenan los mismos valores
- Debe haber una **correspondencia de direcciones** entre los distintos niveles de la jerarquía

Aciertos y fallos en el acceso a la caché

- La palabra a leer o escribir se busca en MC
  - Si la palabra se encuentra en caché, **acierto**
  - Si no se encuentra en caché, **fallo**. La penalización por fallo dependerá de la latencia de la MP y de su ancho de banda
- Si la palabra no se encuentra en la MC, se trae un **bloque** que contiene dicha **palabra** desde la MP
- ¿Qué pasaría si siempre se produjeran fallos en la caché?
  - **Principio de localidad**

Principio de localidad

- **Localidad espacial**
  - Si se referencia un elemento, los elementos cercanos a él también tenderán a ser referenciados
  - La jerarquía de memoria mueve bloque con palabras contiguas en memoria a los niveles más altos de la jerarquía
  - Operaciones con matrices y arrays, ejecución secuencial de un programa
- **Localidad temporal**
  - Si se referencia un elemento, este tenderá a ser referenciado pronto
  - La jerarquía de memoria mantiene los datos accedidos recientemente lo más cerca posible del procesador
  - Estructura de los programas: datos y bucles

Aciertos y fallos en el acceso a la MP

- Relación entre MP y MV similar a la existente entre MC y MP
- La MP se divide en páginas o segmentos
- Cuando se produce un fallo de página o segmento, se debe acceder a la MV
- **La penalización en este caso es mayor** ya que la MV es la más lenta de la jerarquía
- En la gestión de este nivel **interviene el SO**. El procesador realiza un *cambio de contexto* y ejecuta otra tarea hasta que la página o segmento esté disponible en MP

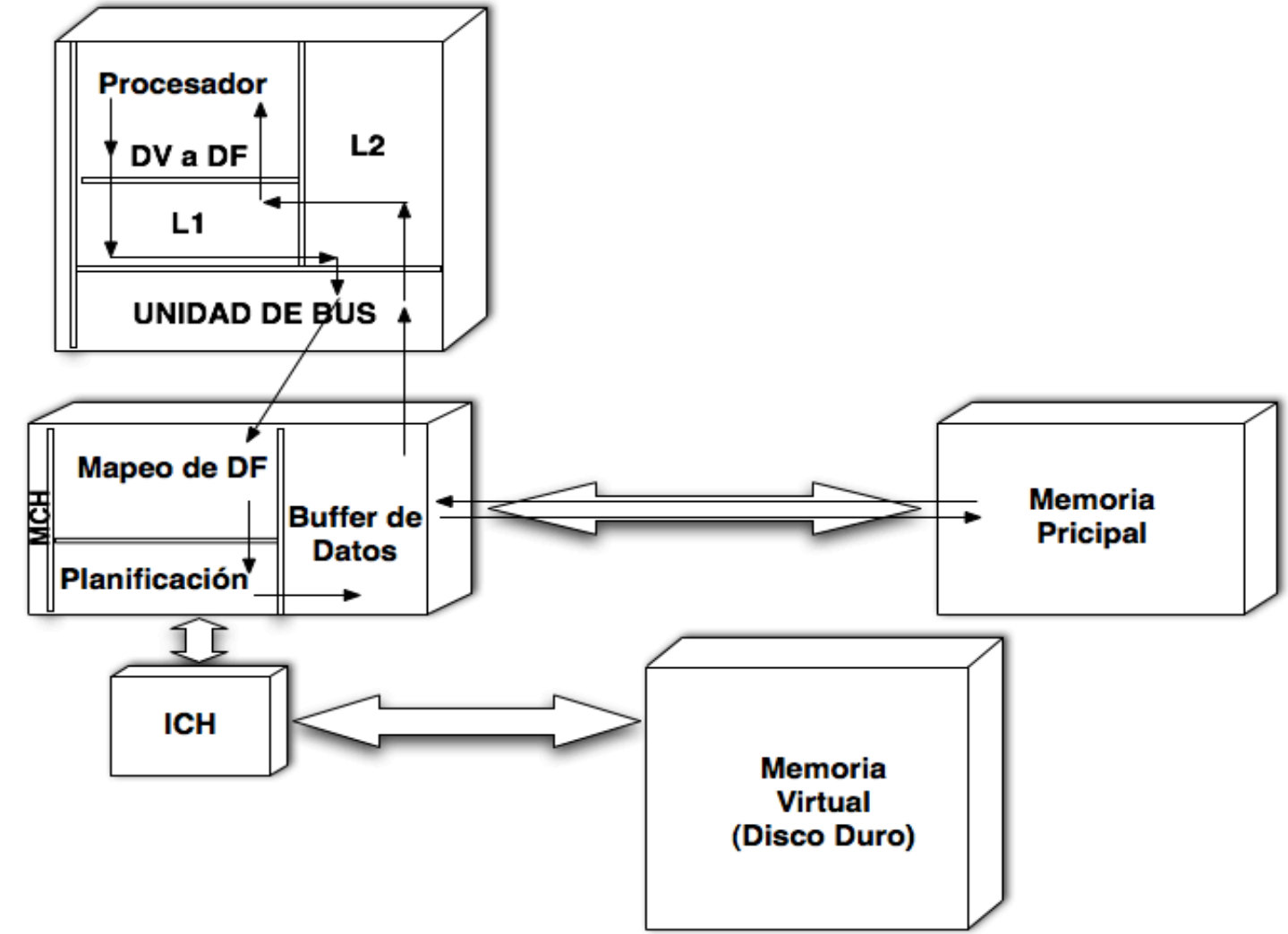
Mecanismo completo de acceso a memoria

1. Se traduce la **dirección virtual** a una **dirección física**
2. Si se traduce con éxito, es porque la palabra se encuentra en MP. Con esta dirección se accede a la memoria caché
3. Tipos de fallos en la MC:
   - **Iniciales**, cuando se referencia una palabra por primera vez
   - **De capacidad**, cuando se producen reemplazos
   - **De conflicto**, varios bloques tienen asignada la misma ubicación en memoria caché
4. En caso de fallo, **hay que pasar por el controlador de MP**, que mapeará la dirección física buscada a la ubicación física de la palabra dentro de los chips DRAM
5. El controlador planificará el acceso a la MP puesto que otros dispositivos también acceden a la MP
6. En caso de acierto, el bloque que incluye la palabra se envía a la MC
7. En caso de fallo, se debe resolver el fallo de página o segmento desde la memoria virtual
8. El SO realiza un cambio de contexto y pasa a ejecutar otro proceso
9. Una vez que la página o el segmento está en MP, se lleva el bloque correspondiente a MC y se reanuda la ejecución de la instrucción que provocó el fallo

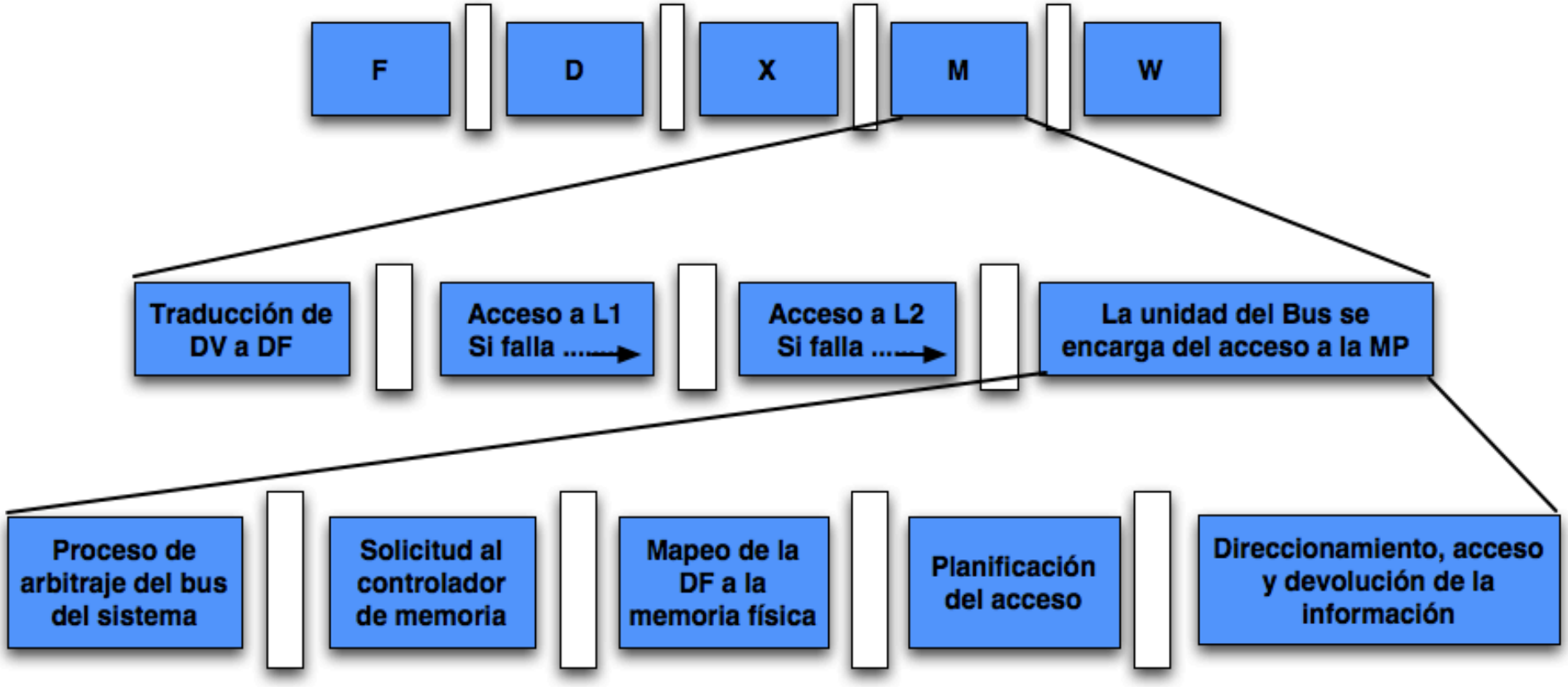
Mecanismo completo de acceso a memoria

- En caso de tener **dos niveles de MC**, el mecanismo de acceso es el mismo, salvo que cuando falla el acceso al primer nivel de caché, primero se intenta resolver desde el segundo
- Si la palabra está en el segundo nivel, se envía el bloque completo al primer nivel, completando el acceso sin salir del chip del procesador
- En caso de fallo, se debe intentar desde MP
- Siguiendo el principio de inclusión, el bloque que provocó el fallo se lleva desde la MP hasta la caché de segundo nivel, y desde ésta, a la caché de primer nivel
- En el caso de un procesador segmentado, todo este proceso (hasta llegar a la MP) debería completarse en la etapa de acceso a memoria, es decir, en un único ciclo de reloj

Mecanismo completo de acceso a memoria



Acceso a memoria del nanoMIPS en la etapa M



Evaluación de prestaciones de la jerarquía de memoria

$$t_{MEM} = t_{aciertoMC} + TF \bullet pF$$

- $t_{aciertoMC}$: Tiempo de acierto de la MC
- *TF*: Tasa de fallos de la MC

TF = núm de fallos / núm total accesos a memoria

- *pF*: Penalización por fallo en MC
- Normalmente, el tiempo invertido en acceder a la memoria se suma al tiempo de CPU

$$t = t_{CPU} + t_{MEM}$$

Evaluación de prestaciones de la jerarquía de memoria

- Métricas
  - **Latencia**: tiempo que transcurre desde que un acceso a memoria comienza hasta que finaliza
  - **Ancho de banda**: cantidad de información por unidad de tiempo que puede transferirse desde/hacia la memoria

Ejemplo

- Comparación entre una tecnología de memoria perfecta y una jerarquía real

- Frecuencia de reloj = 2 GHz
- CPI ideal de 1
- 100 instrucciones
- 2 MC, una de datos y otra de instrucciones, ideales, nunca fallan

$$t = t_{CPU} = I \bullet CPI \bullet T = 100 \bullet 1 \bullet 1/2 \bullet 10^9 = 50\ ns$$

Diseño de la memoria caché

- Si la MC de instrucciones tiene una tasa de fallos del 4% y una penalización por fallo de 100 ns, y la MC de datos tiene una talla se fallos del 6% y una penalización por fallo de 115 ns
- Si para ejecutar las 100 instrucciones hacen falta 100 accesos a la memoria de instrucciones y 25 a la de datos, el tiempo que se invierte en acceder a la memoria es:

$$t_{MEM} = nº \text{ de acceso a MI}(t_{aciertoMI} + TF_{MI} \bullet pF_{MI}) + nº \text{ de acceso a MD}(t_{aciertoMD} + TF_{MD} \bullet pF_{MD})$$
$$= 100(0 + 0.04 \bullet 100) + 25(0 + 0.06 \bullet 115) = 572{,}25 \text{ ns}$$

- Tiempo de CPU:

$$t = t_{CPU} + t_{MEM} = 50 \text{ ns} + 572{,}5 \text{ ns} = 622{,}5 \text{ ns}$$

Diseño de la memoria caché

- Almacena bloques de información denominados **marcos**
- Para determinar qué bloque está ocupando un determinado marco, se utilizan **etiquetas o *tags***
- Las etiquetas se comparan con la del bloque buscado para indicar si se ha producido un acierto o un fallo
- Aspectos básicos de su diseño
  - Organización de la memoria caché
  - Política de ubicación
  - Política de reemplazo
  - Política de escritura

Organización de la memoria caché

- Tamaño de la memoria caché
- Tamaño de marco
- Unificación o división de las instrucciones y los datos
- Implementación de cachés multinivel

Organización de la memoria caché: *Tamaño*

- **Demasiado pequeña**
  - *Incrementa la tasa de fallos*
  - *No captura bien la localidad*
  - *Fallos de capacidad*
- **Demasiado grande**
  - *No se podrá integrar en el mismo chip que el procesador debido al consumo de área y de potencia*
  - *Más lenta, más comparaciones de etiquetas*

Organización de la memoria caché: *Marco*

- **Bloques grandes**
  - *Se captura mejor la localidad espacial*
  - *Se reducen los fallos iniciales*
  - *…*
  - *Aumenta la penalización por fallo,* se necesita más tiempo para traer los bloques del siguiente nivel
- **Se debe llegar a un compromiso** teniendo en cuenta la latencia y el ancho de banda de conexión con el siguiente de la jerarquía

Organización de la memoria caché: *Unificación o división*

- **La segmentación del procesador obliga a la división** para evitar riesgos estructurales entra las etapas F y M
- Cuando no se trata del primer nivel se puede optar por unificar en bloques de información comunes las instrucciones y los datos
- Todas estas decisiones se toman con ayuda de herramientas de simulación

Organización de la memoria caché: *Multinivel*

- Aspecto que más influye en el rendimiento
- Memoria caché de un único nivel
  - ¿pequeña y rápida o grande y lenta? **Solución**: utilizar más de un nivel de memoria caché, normalmente **dos niveles** (algunas arquitecturas empiezan a utilizar tres niveles)
- Nivel 1 (L1): cercana al procesador, pequeña y rápida
- Nivel 2 (L2): mayor tamaño, aprovecha el principio de localidad, más lenta pero menos fallos de capacidad
- La penalización por fallo es menor, en lugar de ir a MP irá a la caché de nivel 2

Ejemplo

- Comparación de una jerarquía de memoria con una sola caché con otro jerarquía de memoria con dos niveles de caché
- Procesador con un único nivel de caché unificada
  - Tiempo de acceso: 1 ns
  - Tasa de fallos del 5%
  - Penalización por fallos: 90 ns
  - Tiempo medio de acceso a memoria para realizar una lectura:

$$t_{MEM}\ (lectura) = t_{aciertoL1} + TF_{L1} \bullet pF_{L1} = 1 + 0{,}05 \bullet 90 = 5{,}5\ ns$$

Ejemplo

- Segundo nivel de memoria caché
  - Tiempo de acceso: 12 ns
  - Tasa de fallos del 10%
  - Suponemos que los movimientos de información entre los dos niveles implican un tiempo despreciable
  - Tiempo medio de acceso a memoria para realizar una lectura:

$$t_{MEM}\ (lectura) = t_{aciertoL1} + TF_{L1} \bullet pF_{L1} = 1 + 0{,}05 \bullet pF_{L1}$$

$$pF_{L1} = t_{L2} = t_{aciertoL2} + TF_{L2} \bullet pF_{L2} = 12 + 0{,}1 \bullet 90 = 21\ ns$$

$$t_{MEM}\ (lectura) = 1 + 0{,}05 \bullet 21 = 2{,}05\ ns$$

$$Ganancia = 5{,}5/2{,}05 = 2{,}68$$

Política de ubicación

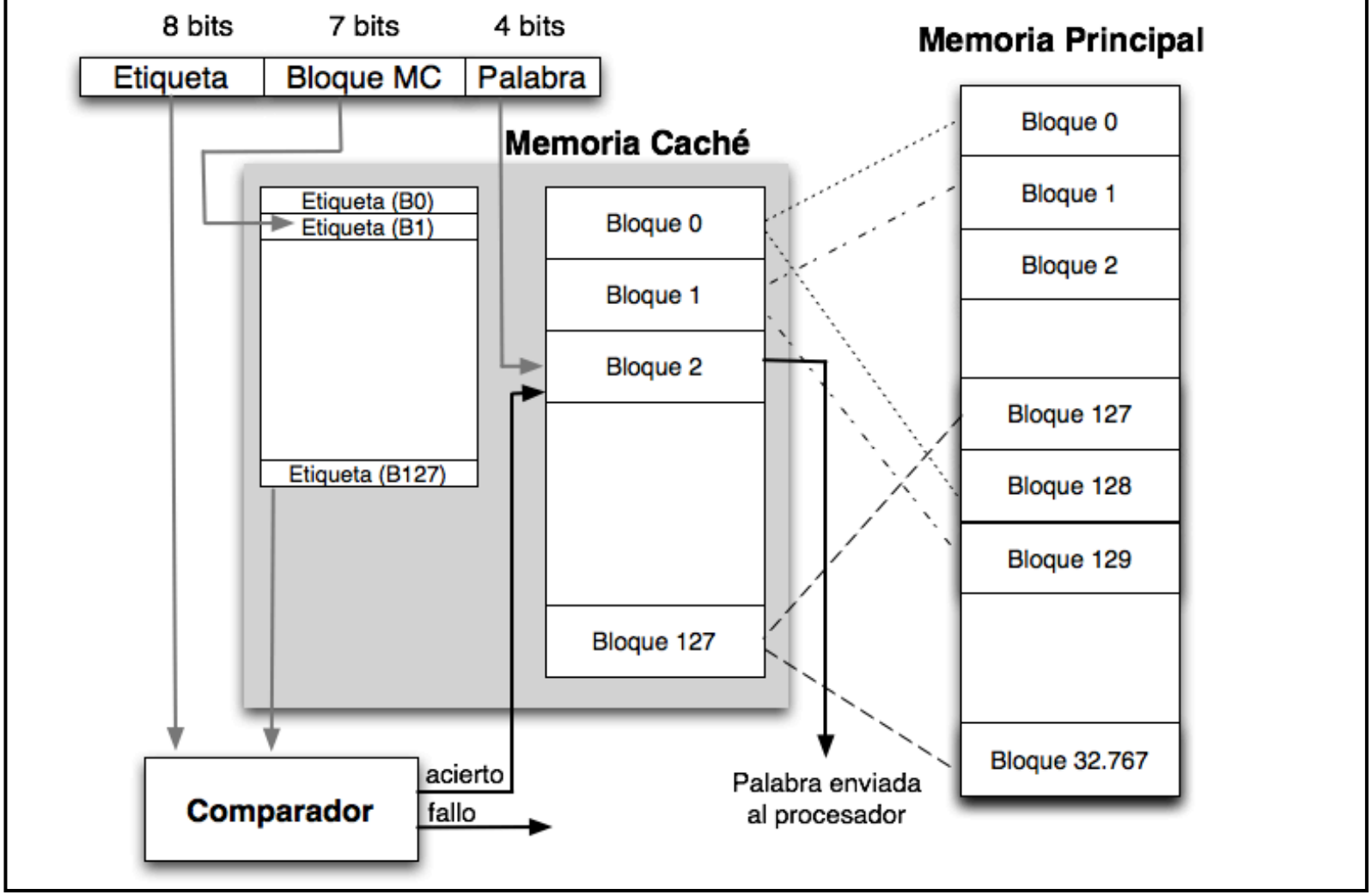
- En la MC sólo se pueden almacenar unos pocos bloques de información
- ¿En qué marco alojamos el bloque?
- Política de ubicación: tasa de fallos Vs tiempo de acceso
  - **Correspondencia directa**
  - **Totalmente asociativa**
  - **Asociativa por conjuntos**

Política de ubicación: *Correspondencia directa*

- A cada bloque de memoria principal solo le corresponde un marco de memoria caché
- Correspondencia entre memoria principal y caché
  - Dirección física: 19 bits
  - Tamaño de bloque: 16 bytes (2^4)
  - Capacidad MC: 2KB (2048B/16B = 128 bloques)
  - Capacidad MP: 512KB (512KB/16B = 32.768 bloques)
- **Ejemplo**: para ubicar el bloque 5 de MP en una MC de 128 marcos tendremos que realizar el módulo

Bloque 5 *mod* 128 marcos = marco 5

Política de ubicación: *Correspondencia directa*



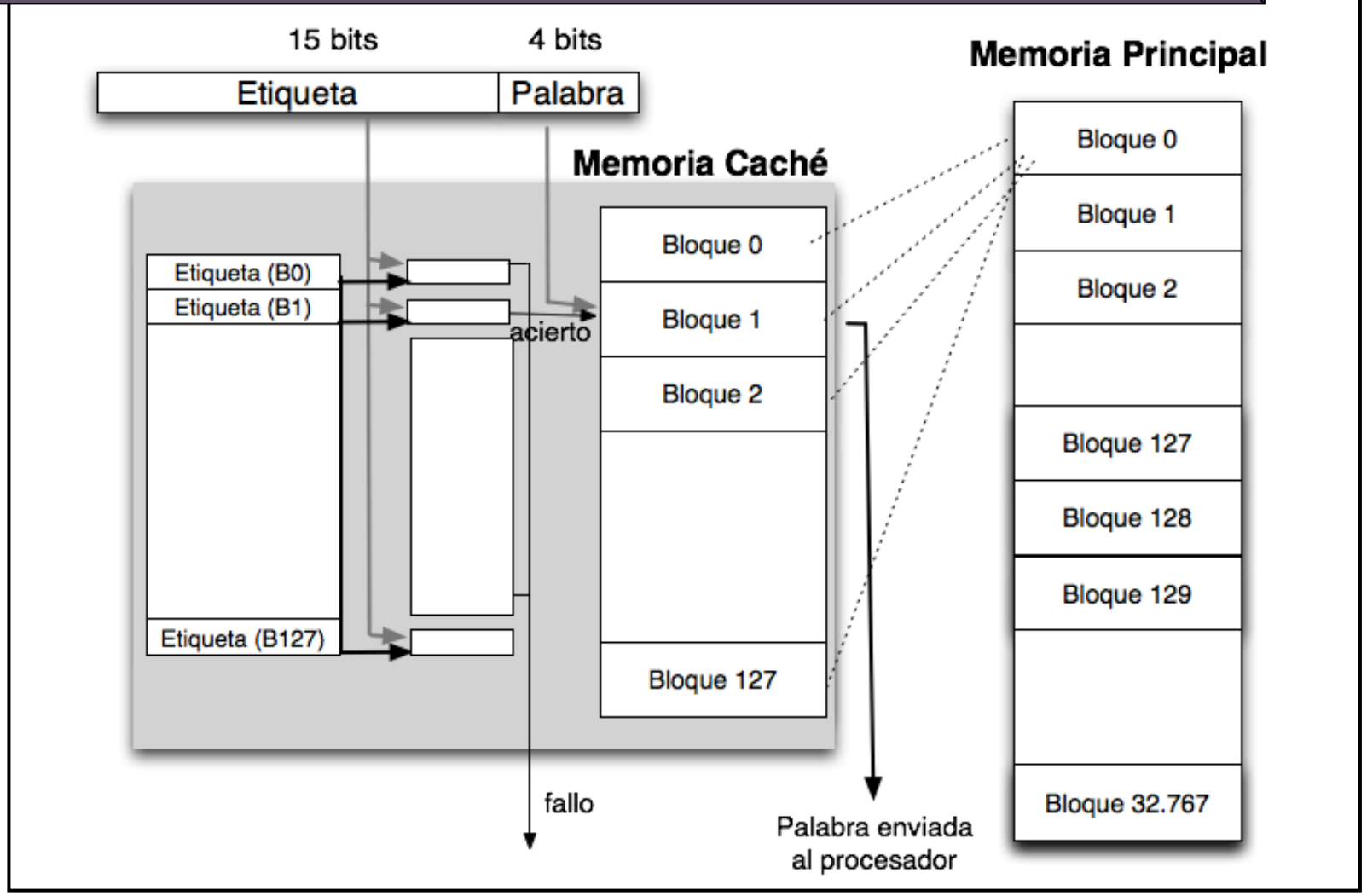
Política de ubicación: *Correspondencia directa*

- **Ventajas**
  - La lectura permite el acceso simultáneo al **directorio** y a la palabra dentro del bloque de MC
  - Algoritmo de reemplazo trivial
- **Inconvenientes**
  - Incremento de la tasa de fallos cuando dos bloques de MP correspondientes al mismo bloque de MC son accedidos de forma alternativa

Política de ubicación: *Totalmente asociativa*

- Cualquier bloque de MP se puede ubicar en cualquier bloque de MC
- La etiqueta se compara con todas las etiquetas almacenadas en caché
- **Ventajas**
  - Flexibilidad, permite implantar gran variedad de algoritmos de reemplazo
  - Presenta la mayor tasa de aciertos
- **Inconvenientes**
  - Mayor tiempo de acceso

Política de ubicación: *Totalmente asociativa*



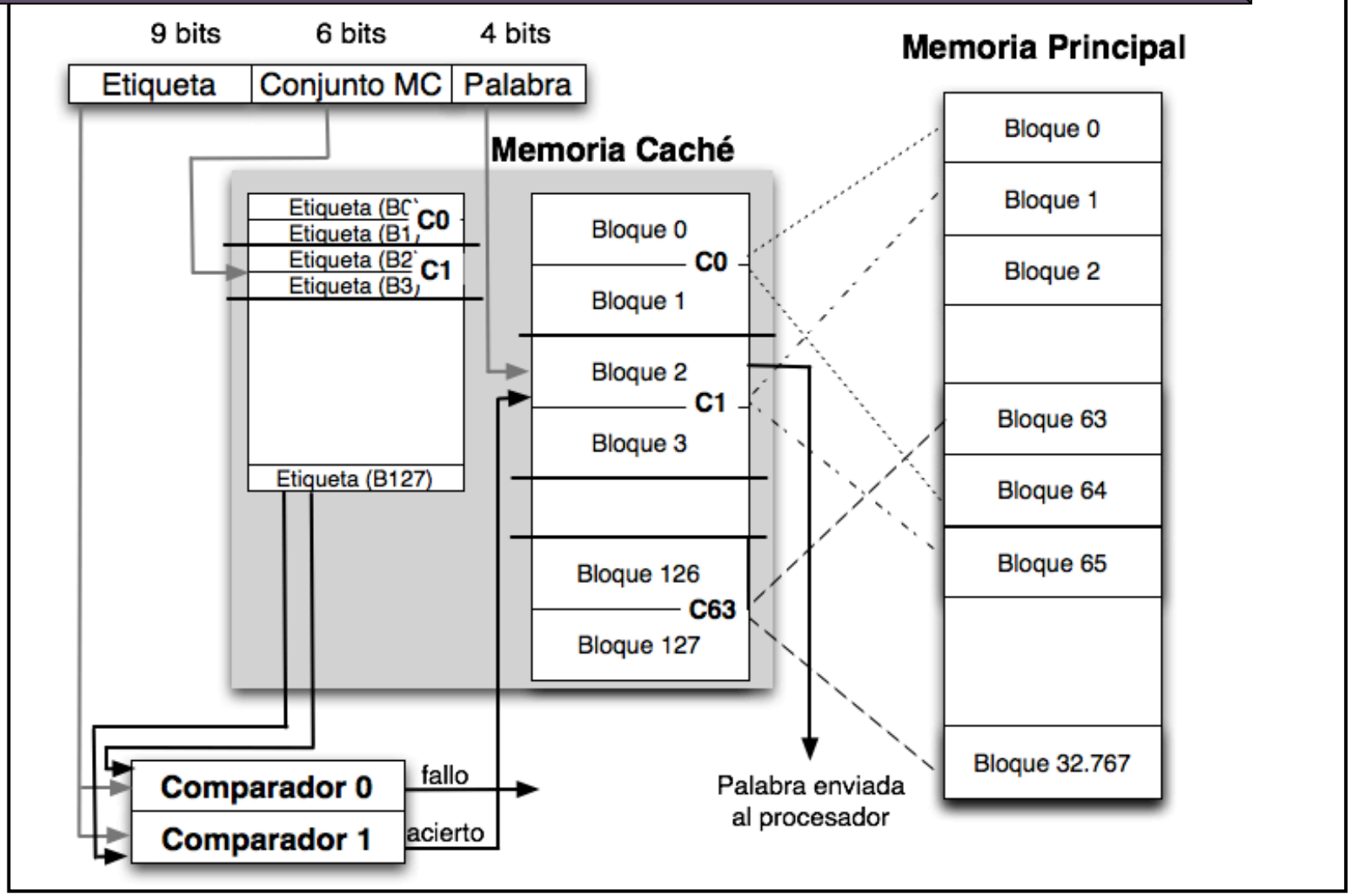
Política de ubicación: *Asociativa por conjuntos*

- Dividir la MC en *C* **conjuntos** de *B* bloques o *vías*
- Se aplica:
  - Correspondencia directa a nivel de conjuntos
  - Correspondencia asociativa a nivel de bloques
- Solución intermedia
- Reduce el tiempo de acceso de la memoria totalmente asociativa e incrementa la tasa de aciertos de la directa
- Ajustar los valores de C y B!
- **Ejemplo**: para ubicar el bloque 5 de MP en una MC asociativa de 64 conjuntos de 2 vías, tendremos que realizar el módulo

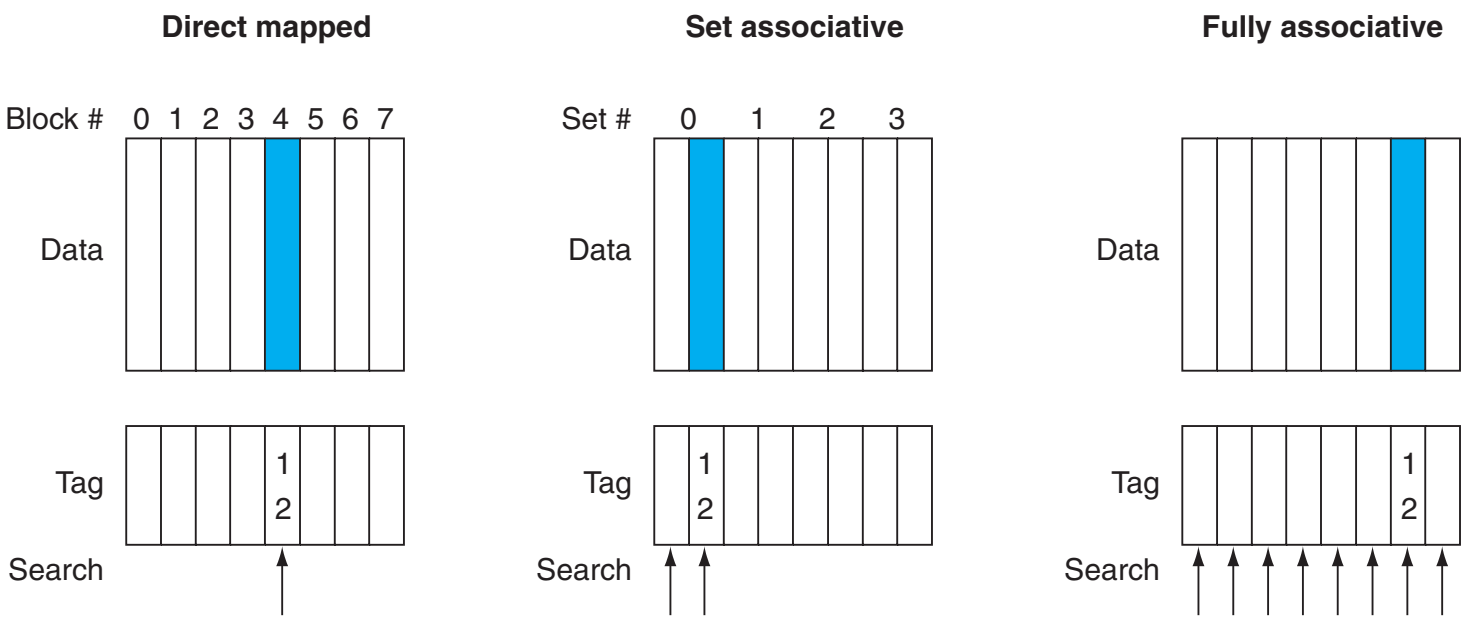
  Bloque 5 *mod* 64 conjuntos = conjunto 5

- … uso de simuladores para determinar el diseño de caché que más nos conviene

Política de ubicación: *Asociativa por conjuntos*



Ubicación del bloque 12 con las tres políticas



Políticas de reemplazo

- Cuando se produce un fallo en MC hay que determinar qué bloque de MC desalojar para traer el bloque que ha producido el fallo desde MP
- En el caso de una caché con correspondencia directa solo se puede desalojar un bloque
- Dicha selección puede reducir la tasa de fallos de los accesos posteriores a MC
- A tener en cuenta
  - Probabilidad de uso de una página
  - Coste del intercambio (modificada?)
- Algoritmos más utilizados
  - **Aleatorio**
  - **FIFO**
  - **LRU**

Políticas de reemplazo

- **Aleatorio:** se utiliza un generador de números aleatorios para escoger el bloque a reemplazar
  - Más sencilla de implementar y la menos costosa
- **FIFO** - *First In First Out*, reempleza el bloque que ha permanecido en MC el mayor periodo de tiempo
  - Más costoso a medida que aumenta al número de bloques, se debe mantener una lista ordenada
- **LRU** - *Least Recently Used*, reemplaza el bloque de memoria que lleva más tiempo sin utilizarse
  - Más complejo y costoso en cuanto a recursos, pero es el qué mejores resultados ofrece
- Tasa de aciertos Vs Tamaño caché (número de bloques)

Políticas de escritura

- Las escrituras llevan más tiempo puesto que no se puede realizar trabajo en paralelo
  - En el caso de las lecturas, se recupera la información de un marco determinado al tiempo que se realiza la comparación de las etiquetas
- Políticas
  - **Escritura directa**
  - **Post-escritura**

Políticas de escritura: *Escritura Directa - Write through*

- Se escribe a la vez en el primer nivel de memoria caché y en el siguiente nivel de la jerarquía
- En caso de fallo, se trae el bloque de MP a MC y una vez en esta, se procede a realizar la escritura
- **Ventajas**
  - Fácil de implementar
  - Asegura la coherencia
- **Inconvenientes**
  - Se genera mucho tráfico con MP
  - El procesador tiene que esperar (diferencia tiempos de escritura en MC y MP) a que se complete la escritura
    - Solución**: buffer de escritura**

Políticas de escritura: *Post-escritura - Write back*

- Cuando se modifica una palabra solo se hace en el primer nivel de caché
- Para no actualizar la memoria con cada reemplazo, se activa un bit, el **bit de sucio** o *dirty bit*, que indicará si el bloque ha sido modificado o no
- La escritura del bloque en MP se realiza cuando el bloque sucio en MC se reemplaza
- **Ventajas**
  - Menos tráfico de información
  - Los aciertos en escritura se llevan a cabo a la velocidad de la MC
- **Inconvenientes**
  - Diseño más complejo, requiere más recursos
- Teniendo en cuenta posibles fallos transitorios de la MC, es preferible la escritura directa en L1

Buffer de escritura

- **Estructura hardware** en la que se realizan las escrituras en primera instancia
- Posteriormente se solapa la escritura con el siguiente nivel de la jerarquía con la ejecución de las siguientes instrucciones
- Estructura pequeña y rápida organizada como una memoria caché
- Las escrituras en el buffer suponen una penalización menor que hacerlas en el siguiente nivel de la jerarquía de memoria
- Si el buffer está lleno, el procesador debe parar hasta que quede una posición vacía

Buffer de escritura: *escritura directa*

- Las escrituras se hacen palabra a palabra en caché y en este buffer en lugar de en el siguiente nivel de la jerarquía de memoria
- El contenido del buffer se volcará en el siguiente nivel:
  - Cuando esté **lleno**
  - Cuando se produzca un **fallo de lectura** y sea necesario actualizar el siguiente nivel con el contenido del buffer antes de resolver el fallo

Buffer de escritura: *post-escritura*

- El buffer se utiliza para volcar los bloques sucios que van a ser reemplazados en memoria caché
- No hay que esperar a que se escriba el bloque sucio en el siguiente nivel, puesto que se almacena temporalmente en el buffer
- El contenido del buffer se volcará en el siguiente nivel:
  - Cuando esté lleno
  - Cuando se produzca un fallo

Políticas de escritura

- Si el acceso a MC es para escritura y el bloque no se encuentra, se produce un ***fallo de escritura***
- **Escritura con ubicación** *(Write with allocate)*
  - Se suele asociar con *post-escritura*
  - Se lleva el bloque de MP a MC donde se realiza la escritura
- **Escritura sin ubicación** *(Write with no allocate)*
  - Se suele asociar con *escritura directa*
  - Sólo se escribe sobre la MP

Ejemplo

- Comparación ancho de banda de MP consumido por las dos políticas de escritura
- Frecuencia de reloj: 2,5 GHz
- Accesos a memoria: $2 \cdot 10^7$
- Único nivel de MC unificada
- Tasa de fallos MC: 12%
- Tamaño de bloque: 16 palabras
- 70% lecturas, 30% escrituras
- MC con escritura directa
- Escritura con ubicación

Ejemplo (continuación)

Tipo de acceso	Acciones	Tráfico con MP
Acierto de lectura	- Lectura MC	0
Fallo de lectura	- Traer bloque de MP - Lectura en la MC	16 palabras
Acierto de escritura	- Escritura en MC y en MP	1 palabra
Fallo de escritura	- Traer bloque de MP - Escribir en la MC y en la MP	17 palabras

- Ancho de banda consumido:

AB_{MP} = *referencias a memoria por segundo • (%fallos lectura•16 + %aciertos escritura•1 + %fallos escritura•17) =*

$2 \cdot 10^7 \cdot (0{,}12 \cdot 0{,}7 \cdot 16 + 0{,}88 \cdot 0{,}3 \cdot 1 + 0{,}12 \cdot 0{,}3 \cdot 17) =$

4,44 •10^7 palabras por segundo

Ejemplo (continuación)

Tipo de acceso	Acciones	Tráfico con MP
Acierto de lectura	- Lectura MC	0
Fallo de lectura, reemplazamiento bloque limpio	- Traer bloque de MP - Lectura en la MC	16 palabras
Fallo de lectura, reemplazamiento bloque sucio	- Escritura de bloque en MP - Traer bloque de MP - Lectura en MC	32 palabras
Acierto de escritura	- Escritura en MC	0
Fallo de escritura, reemplazamiento bloque limpio	- Traer bloque de MP - Escritura en la MC	16 palabras
Fallo de escritura, reemplazamiento bloque sucio	- Escritura de bloque en MP - Traer bloque de MP - Escritura en MC	32 palabras

Ejemplo (continuación)

- En post-escritura, el 10% de los bloques son modificados cuando se encuentran ubicados en caché
- Ancho de banda consumido:

AB_{MP} = referencias a memoria por segundo • (%fallos con reemplazamiento bloque limpio•16 + %fallos con reemplazamiento bloque sucio•32) =

$2 \cdot 10^7 \cdot (0{,}12 \cdot 0{,}9 \cdot 16 + 0{,}12 \cdot 0{,}1 \cdot 32) =$

$4{,}22 \cdot 10^7$ palabras por segundo

Ejemplo

- Comparación del tiempo medio de acceso a memoria con las dos políticas de escritura
- Tº medio acceso a MC es de 4ns
- Latencia de acceso a MP es de 85ns
- Tamaño bloque: 16 palabras
- MC con escritura directa

t_{MEM} (lectura) = $t_{acierto}$ + TF•pF

t_{MEM} (escritura) = $t_{acierto}$ + $latencia_{MP}$ + TF•pF

pF = 16•($latencia_{MP}$) = 16 • (85) = **1360 ns**

t_{MEM} (lectura) = $t_{acierto}$ + TF•pF = 4 + 0,12 •1360 = **167,2 ns**

t_{MEM} (escritura) = $t_{acierto}$ + $latencia_{MP}$ + TF•pF =

4 + 85 + 0,12 •1360 = **252,2 ns**

Ejemplo (continuación)

- En media:

t_{MEM} = %lectura • t_{MEM} (lectura) + %escritura • t_{MEM} (escritura)
= 0,7 • 167,2 + 0,3 • 252,2 = **192,7 ns**

- Con las post-escritura tenemos el mismo comportamiento para lecturas y escrituras:

$t_{MEM} = t_{acierto} + TF \bullet pF$

- Pero la penalización por fallo se modifica respecto de la escritura directa

$pF = 16 \bullet (latencia_{MP}) + \%sucios \bullet 16 \bullet (latencia_{MP}) =$
16•85 + 16•0,1•85 = **1496 ns**

$t_{MEM} = t_{acierto} + TF \bullet pF =$ 4 + 0,12 • 1496 = **183,52 ns**

Ejemplo

- Diseño de una jerarquía de memoria con dos niveles de caché unificados
  - *El primer nivel de escritura directa*
  - *El segundo nivel de post-escritura con un 26% de bloques modificados*
- No se solapan accesos y transferencias
- L1, tamaño de bloque de 8 palabras, tº de acceso de 1 ns, tasa de fallos del 5%
- L2, tamaño de bloque de 16 palabras, tº de acceso de 9 ns, tasa de fallos del 9%
- Transferencia de una palabra entre MP y L2 supone 0,5 ns
- Transferencia de una palabra entre L1 y L2 supone 0,1 ns
- Latencia acceso MP de 85 ns

$$t_{MEM}\ (lectura) = t_{aciertoL1} + TF_{L1} \bullet pF_{L1}$$

$$t_{MEM}\ (escritura) = t_{aciertoL1} + t_{aciertoL2} + TF_{L1} \bullet pF_{L1}$$

$$pF_{L1} = 8 \bullet (t_{L2} + t_{busL1L2})$$

$$t_{L2} = t_{aciertoL2} + TF_{L2} \bullet pF_{L2}$$

Ejemplo (continuación)

$pF_{L2} = 16 \bullet (\text{latencia}_{MP} + t_{busL2MP}) + 16 \bullet \%\text{sucios} \bullet (\text{latencia}_{MP} + t_{busL2MP})$

$pF_{L2} = 16 \bullet (0{,}85+0{,}5) + 16 \bullet 0{,}26 \bullet (0{,}85+0{,}5) = 1723{,}68$ ns

$t_{L2} = 9 + 0{,}09 \bullet 1723{,}68 = 164{,}13$ ns

$pF_{L1} = 8\ (164{,}13 + 0{,}1) = 1313{,}85$ ns

t_{MEM} (lectura) $= 1 + 0{,}05 \bullet 1313{,}85 =$ **66,69 ns**

t_{MEM} (escritura) $= 1 + 9 + 0{,}05 \bullet 1313{,}85 =$ **75,69 ns**

$\mathbf{t_{MEM}} = \%\text{lectura} \bullet t_{MEM}\text{ (lectura)} + \%\text{escritura} \bullet t_{MEM}\text{ (escritura)} =$

$0{,}7 \bullet 66{,}69 + 0{,}3 \bullet 75{,}69 =$ **69,39 ns**

Diseño de la memoria virtual

- La memoria virtual se introduce en la jerarquía de memoria porque:
  - Permite la **multiprogramación** ➔ protección
  - Permite ejecutar procesos más grandes que la MP
  - Permite independencia de las referencias con respecto a la localización de los procesos en MP
- La MV no se controla exclusivamente por hardware
- La tecnología de la MV es el **almacenamiento magnético**
- La unidad de información no es el bloque, sino el **segmento o la página**, mucho mayores que el bloque que se maneja entre MC y MP
- El alojamiento es siempre **asociativo**
- La política de escritura es siempre **post-escritura**

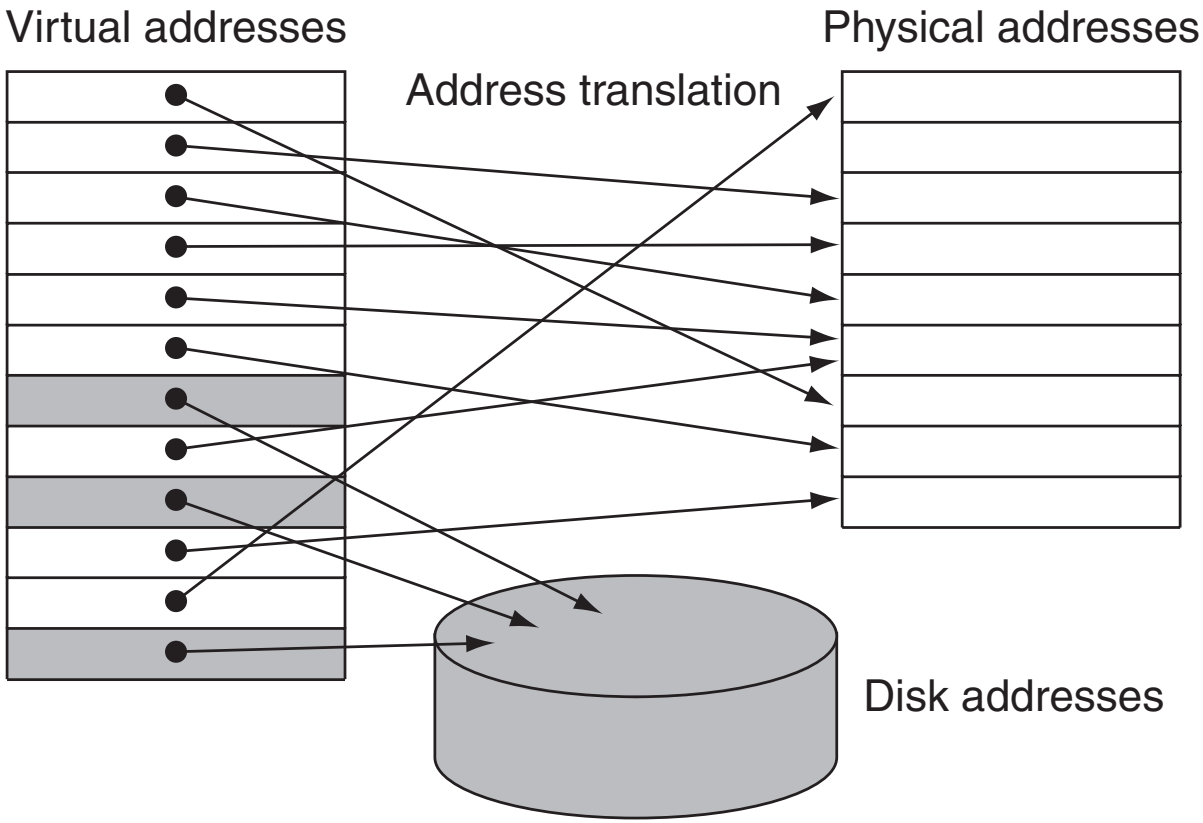
Diseño de la memoria virtual

- Más compleja y difícil de gestionar
- Latencia del orden de *ms* y no de *ns*
- Resolver un **fallo de página** implica un **cambio de contexto** para evitar la enorme penalización por fallo
- Decisiones clave a la hora de diseñar un sistema de memoria virtual:
  - El **tamaño de la página** debe ser lo suficientemente grande para amortizar el elevado tiempo de acceso
    - Tamaño típico entre 4KB y 16KB
    - Nuevos servidores y desktops se están desarrollando para soportar páginas de 32KB y 64KB
    - Los nuevos sistemas empotrados utilizan páginas de 1KB
  - Priman las **políticas que reducen el número de fallos de pagina**, por eso la técnica de alojamiento empleada es totalmente asociativa
  - **Los fallos de página se manejan por software**, puesto que la sobrecarga introducida por estos algoritmos es mucho menor que el tiempo de acceso a disco
  - La *escritura-directa* queda descartada en MV. En su lugar se emplea ***post-escritura***

Organización de la memoria virtual

- **Dirección virtual**→ dirección generada por el procesador
- **Dirección física** → dirección que maneja la unidad de memoria
- *Memory Management Unit (**MMU**)*:
  - Trabaja en tiempo de ejecución de manera transparente a la CPU
  - Traduce direcciones virtuales a direcciones físicas
  - Protección de la memoria: evita que los programas accedan a porciones de memoria prohibidas

Organización de la memoria virtual



Asignación de memoria

- Tres métodos de asignación de memoria
  - **Paginación**: tamaño fijo de bloque de información
  - **Segmentación**: tamaño variable de bloque de información
  - Técnica híbrida, **segmentación paginada**: los segmentos se componen de un número entero de páginas

Paginación

- La paginación se gestiona además de con el hardware, con la colaboración del SO
- Es tan importante reducir los fallos de página que los diseñadores se centran en desarrollar algoritmos de ubicación de páginas lo más óptimos posible
- El problema de utilizar ubicación totalmente asociativa radica en localizar una entrada: una búsqueda completa es impracticable!
- En su lugar, se localizan las páginas por medio de unas tablas que indexan la memoria
- A esta estructura se la denomina **tabla de páginas** y reside en memoria

Paginación

- El espacio de direcciones virtuales de un proceso no es contiguo, pudiendo alojarse en cualquier posición de memoria física
- La memoria física se divide en bloques de tamaño fijo denominados **marcos**
- La memoria virtual se divide en bloques del mismo tamaño denominados **páginas**
- Se mantiene una lista de todos los marcos libres
- Para ejecutar un programa de *n* páginas, hay que encontrar *n* marcos libres y cargar el programa
- Después, se debe inicializar la **tabla de páginas** para traducir direcciones virtuales a físicas

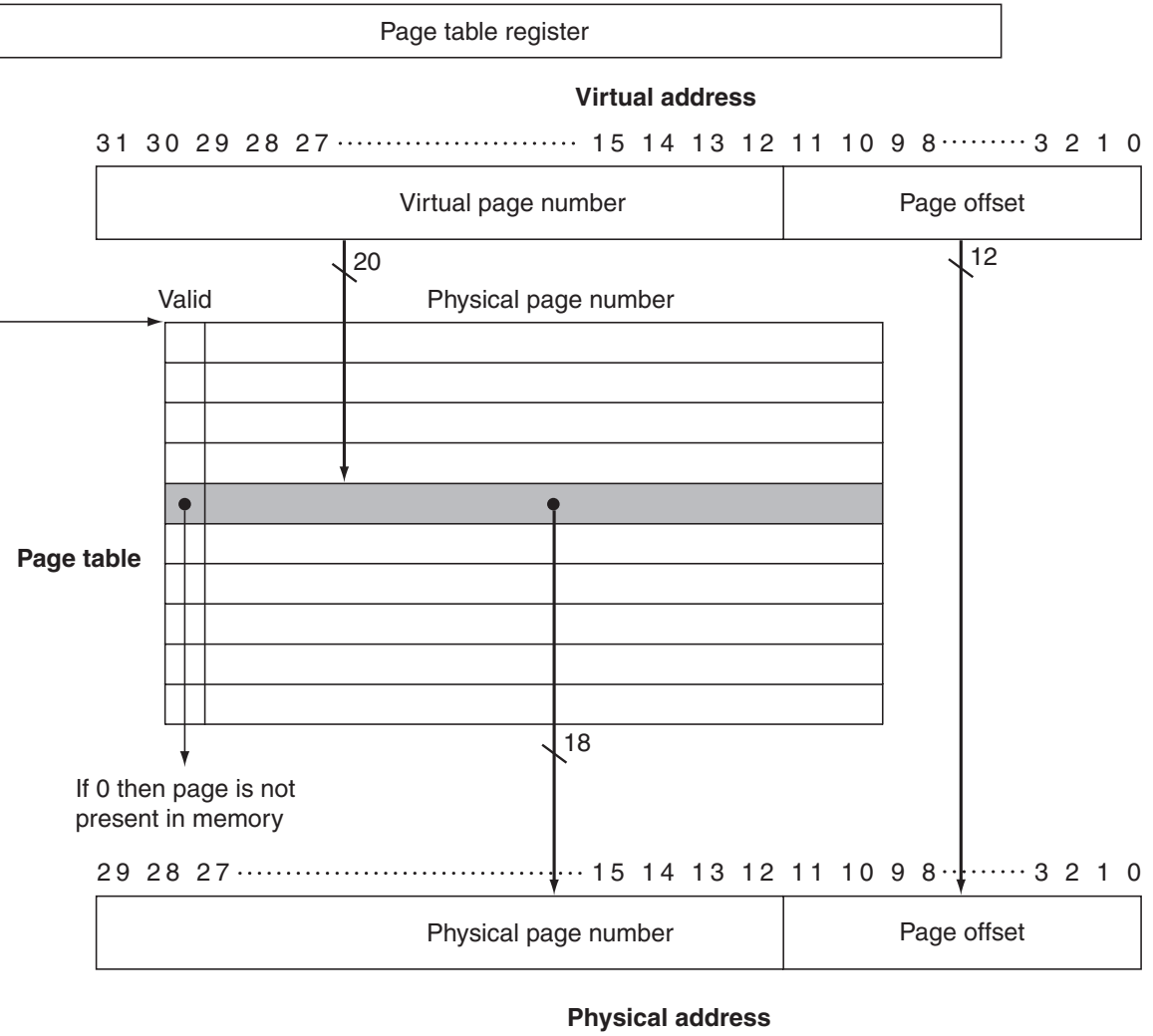
Ubicación y búsqueda de páginas

- Debido a la utilización de una política de ubicación totalmente asociativa, el SO puede implementar algoritmos de reemplazo más inteligentes para reducir los fallos de página
- La verdadera dificultad radica en encontrar una entrada: una búsqueda exhaustiva es impracticable
- En su lugar se emplea una **tabla de páginas** que reside en memoria:
  - Indexada con el número de página de la dirección virtual
  - Permite obtener la página física en la que se ubica la página virtual buscada
- Cada programa del sistema tiene su propia tabla de páginas
- Para indicar la localización de la tabla de páginas en MP, el hardware incluye un registro que apunta al comienzo de la tabla de páginas, el ***page table register***

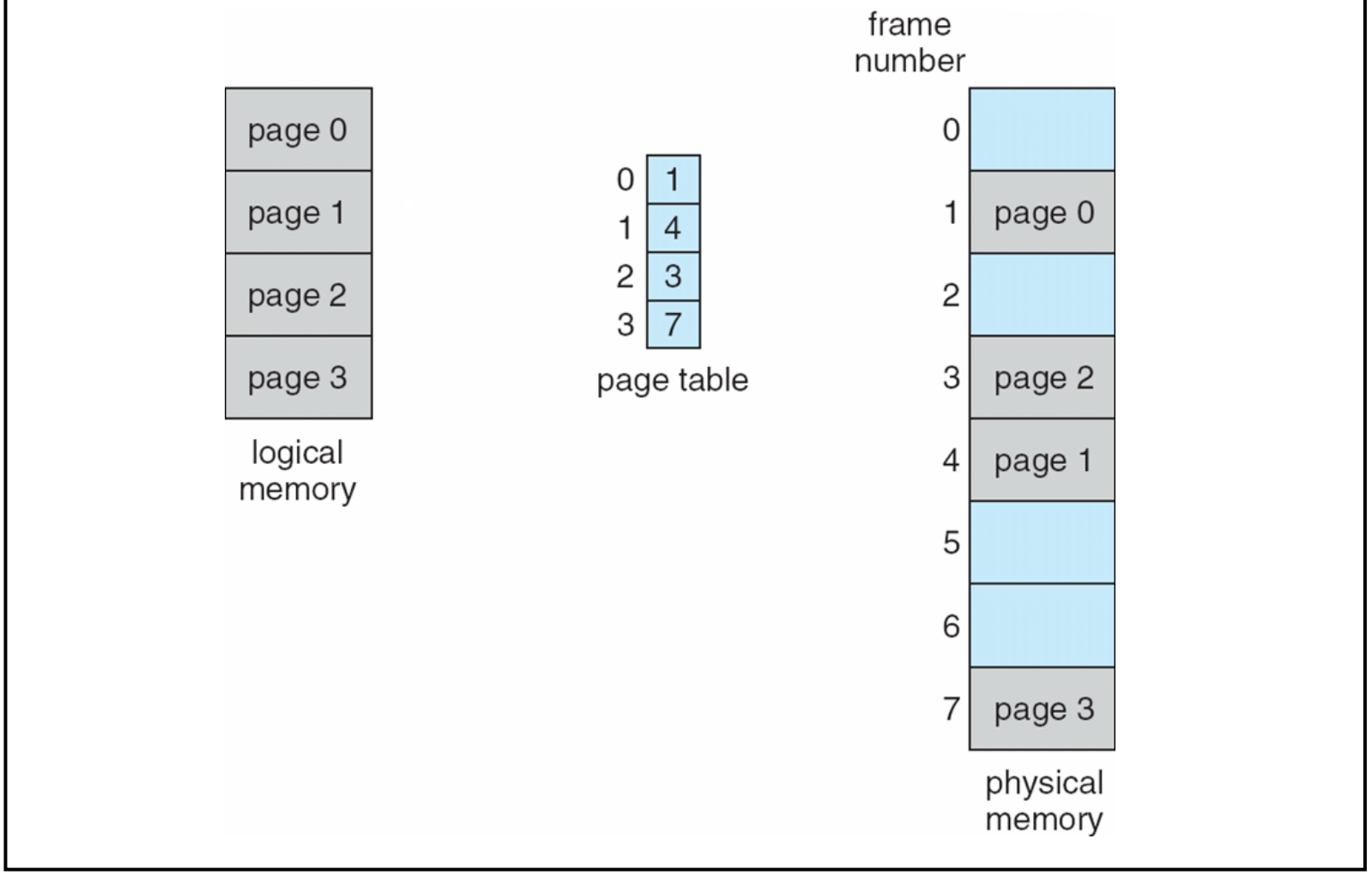
Ubicación y búsqueda de páginas

- Para evitar colisiones entre el espacio de direcciones virtuales de los distintos procesos, el SO se encarga de:
  - La asignación de la memoria física
  - Actualizar las tablas de páginas ➔ protección
- Un **bit de válido** se utiliza en cada entrada de la tabla de páginas para indicar si la página está o no presente en la MP:
  - ***Off*** ➔ la página no está presente en MP, fallo de página
  - ***On*** ➔ la página está presente en MP, acceso a la MC

Esquema de traducción de direcciones



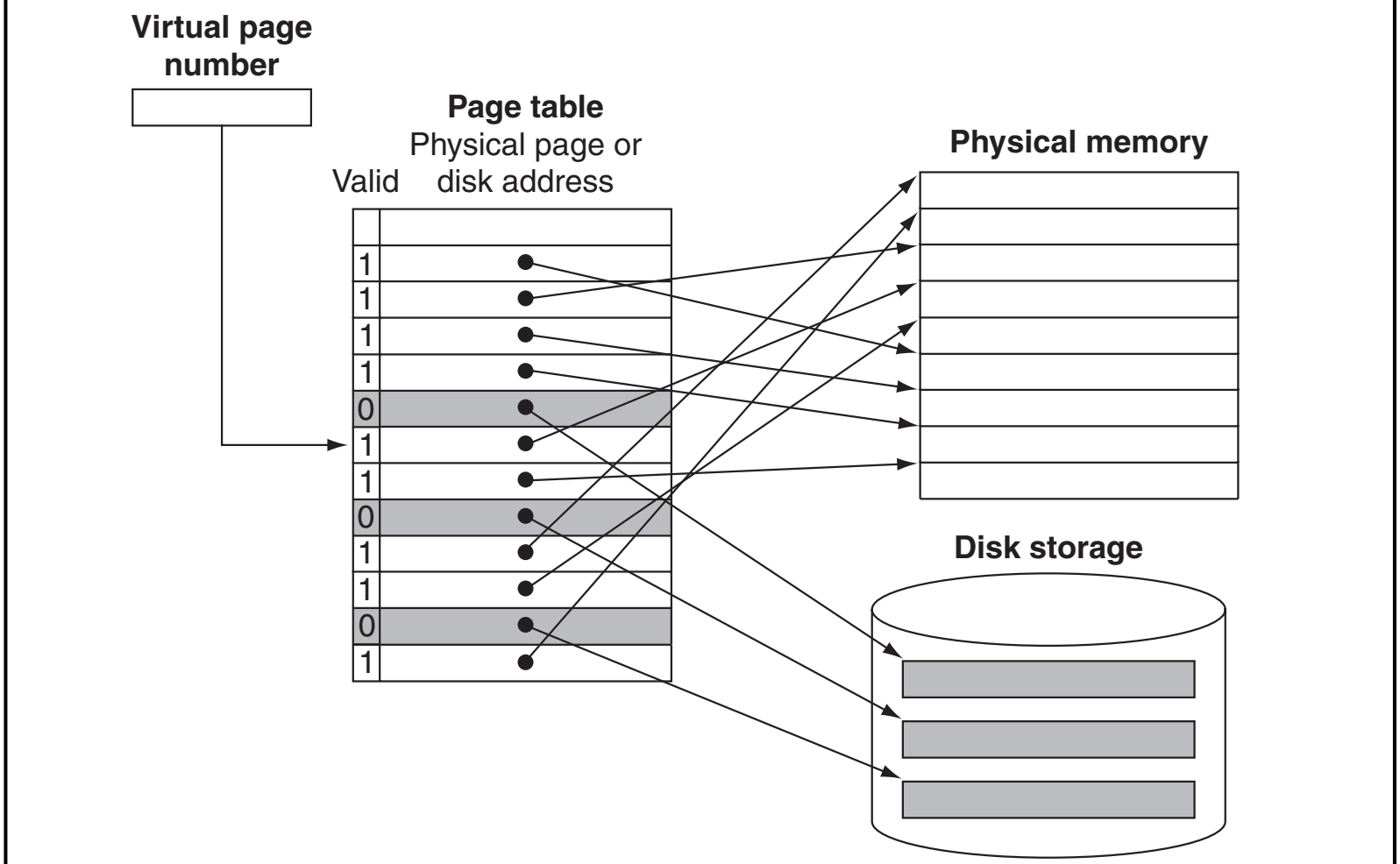
Esquema de traducción de direcciones



Fallo de página

- Si el bit de válido está a off ➔ se produce un fallo de página
- Se debe transferir el control al SO: mecanismo de excepción
- Una vez que el SO tiene el control debe:
  - Buscar la página en el siguiente nivel de la jerarquía, en el disco magnético
  - Decidir en qué marco de MP alojar la página
- Se debe llevar un control de la localización en el disco de las páginas virtuales
- El SO reserva espacio en disco para todas las páginas de un proceso: **swap space**
- Al mismo tiempo crea una estructura de datos para almacenar la ubicación en el disco de cada página virtual
- Esta nueva estructura puede formar parte de la tabla de páginas o ser una estructura auxiliar también indexada

Fallo de página



Fallo de página

- El SO mantiene otra estructura de datos en la que almacena qué procesos y qué direcciones virtuales utiliza cada página física
- Cuando todas las páginas físicas están ocupadas, la mayoría de los SSOO emplean una aproximación del algoritmo LRU (*Least Recently Used*) para reemplazar una página que lleva cierto tiempo sin ser referenciada
- Para ayudar al SO, algunos ordenadores proporcionan el **bit de referencia** o bit de uso, que se pone a 1 cuando se accede a la página
- Si la página a reemplazar ha sido modificada, en los sistemas de MV se emplea ***write-back***, de tal manera que la página se modifica en cada escritura en MP y solo se copia en el disco duro cuando es sustituida
- Para saber si una página ha sido modificada, a la tabla de páginas se le añade el ***dirty bit***

Protección

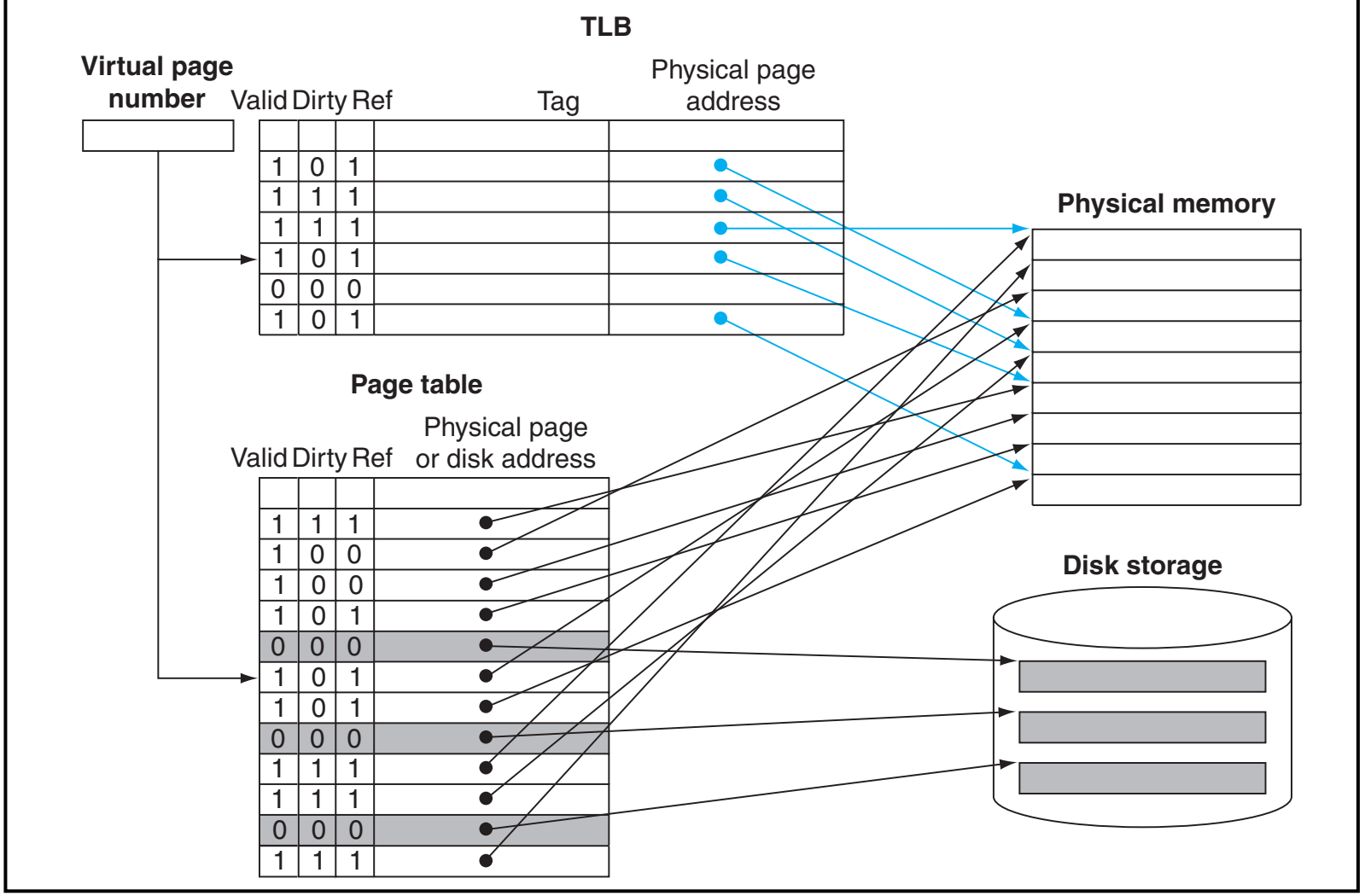
- Cuando varios procesos compartan la MP, se debe evitar la lectura o escritura en zonas de memoria que no pertenecen a un proceso
- Un bit de permiso de escritura en el TLB puede proteger a una página de ser escrita
- El hardware debe proporcionar al SO los siguientes mecanismos:
  - Soportar dos modos de ejecución: **supervisor** y usuario
  - Proporcionar una parte del estado del procesador en modo sólo lectura de tal manera que el usuario no la pueda escribir:
    - Bit de modo usuario/supervisor
    - Puntero de tabla de páginas
    - TLB

  Para modificar estos elementos, el SO utiliza instrucciones especiales sólo disponibles en modo supervisor
  - Proporcionar mecanismos para cambiar de modo usuario a modo supervisor, como por ejemplo, las llamadas al sistema: *syscall exception*
    - Se salva el PC en el registro EPC
    - Se pone el procesador en modo supervisor
    - Para volver en modo usuario a la dirección almacenada en el EPC, se utiliza la instrucción **`ERET`**
- Las tablas de páginas se almacenan en el espacio de direcciones del SO

Optimización de la traducción de direcciones: **TLB**

- Como las tablas de páginas están almacenadas en la MP, cada acceso a la memoria por un programa implica al menos dos accesos a memoria
- Debido al principio de localidad, sabemos que las palabras de una página serán referenciadas pronto, por lo que la traducción de una página virtual a física debería conservarse
- Es por esto que los procesadores actuales incluyen una caché especial que mantiene las traducciones recientemente utilizadas
- A esta caché de traducción de direcciones se la conoce comúnmente como ***translation-lookaside buffer*** (TLB)

Hardware de paginación con TLB



Optimización de la traducción de direcciones: **TLB**

- En cada referencia a memoria, primero buscamos si la traducción está en el TLB.
- **TLB *Hit***: la traducción *sí* está en el TLB
  - Nos ahorramos el acceso a memoria para consultar la tabla de páginas.
  - Se comprueba el bit de válido, si es 0 ➔ ***page fault***, la página no está en memoria. El procesador genera una excepción para invocar al SO y que éste resuelva el fallo de página.
  - Si el bit de válido está a 1, se procede a actualizar el bit de referencia y el de sucio (sólo en caso de escritura).
- **TLB *Miss***: la traducción *no* está en el TLB, se genera una excepción
  - No nos ahorramos el acceso a memoria.
  - Traemos la traducción de la tabla de páginas sin comprobar el bit de válido.
  - Se re-ejecuta la instrucción, pero en este caso la traducción sí que estará.
  - El TLB miss se puede atender tanto por hardware como por software.
- En el TLB se utiliza ***write-back***, de tal manera que los bits con información solo se copian en la tabla de páginas cuando la entrada en el TLB es reemplazada
- Los diseñadores de TLBs utilizan diferentes combinaciones:
  - LRU por hardware es demasiado caro, tampoco por software
  - Muchos sistemas dan soporte para selección de una entrada aleatoria (***random***).

TLB miss

Register	CP0 register number	Description
EPC	14	Where to restart after exception
Cause	13	Cause of exception
BadVAddr	8	Address that caused exception
Index	0	Location in TLB to be read or written
Random	1	Pseudorandom location in TLB
EntryLo	2	Physical page address and flags
EntryHi	10	Virtual page address
Context	4	Page table address and page number

- Registros de control de MIPS: están en el coprocesador 0.
- Se leen por medio de la instrucción `mfc0` y se escriben con `mtc0`

TLB miss

- Código MIPS para un manejador típico de un fallo de TLB:

```
TLBmiss:
 mfc0  $k1,Context    # copy address of PTE into temp $k1
 lw    $k1, 0($k1)    # put PTE into temp $k1
 mtc0  $k1,EntryLo    # put PTE into special register EntryLo
 tlbwr                # put EntryLo into TLB entry at Random
 eret                 # return from TLB miss exception
```

- Instrucciones especiales para actualizar el TLB:
  - `tlbwr` copia el registro de control `EntryLo` en la entrada del TLB seleccionada por el registro de control `Random`
  - `Random` implementa un algoritmo de reemplazo aleatorio
  - `$k1` y `$k0` son registros temporales utilizados por el kernel sin necesidad de salvarlos
- El SO carga la entrada en el TLB sin comprobarla y re-ejecuta la instrucción
- Si la entrada resulta ser inválida, se genera un fallo de página

TLB miss

- Los fallos de página en el acceso a datos son más complicados:
  1. Ocurren en mitad de la ejecución de la instrucción
  2. La instrucción no se puede completar antes de resolver la excepción
  3. Después de tratar la excepción, la instrucción debe comenzar su ejecución de nuevo como si nada hubiera ocurrido
- En este caso, las instrucciones deben ser rearrancables, algo sencillo en la arquitectura MIPS
- Cuando se produce un ***TLB miss***:
  - El hardware del MIPS salva el número de página en el registro `BadVAddr`
  - Genera una excepción
- La excepción invoca al SO, que trata el fallo vía software
  - Se transfiere el control a la dirección 0X8000 0000, la dirección de la rutina software (***handler***, manejador) encargada de tratar el TLB miss

Fallo de página

- Cuando se produce un fallo de página se transfiere el control a la dirección **0X8000 0180**
  - Dirección general para una excepción
  - El *TLB miss* tiene un punto de entrada especial para reducir la penalización por fallo del TLB
  - El SO utiliza el registro `Cause` para determinar la causa de la excepción
- Como se trata de un fallo de página:
  - El SO salva el estado del proceso activo: registros de propósito general, de coma flotante, registro de dirección de tabla de páginas y los registros `EPC` y `Cause`

Memoria virtual

Save state			
Save GPR	addi	$k1,$sp, -XCPSIZE	# save space on stack for state
	sw	$sp, XCT_SP($k1)	# save $sp on stack
	sw	$v0, XCT_V0($k1)	# save $v0 on stack
	...		# save $v1, $a*i*, $si, $ti, ...on stack
	sw	$ra, XCT_RA($k1)	# save $ra on stack
Save Hi, Lo	mfhi	$v0	# copy Hi
	mflo	$v1	# copy Lo
	sw	$v0, XCT_HI($k1)	# save Hi value on stack
	sw	$v1, XCT_LI($k1)	# save Lo value on stack
Save Exception Registers	mfc0	$a0, $cr	# copy cause register
	sw	$a0, XCT_CR($k1)	# save $cr value on stack
	...		# save $v1, ...
	mfc0	$a3, $sr	# copy Status Register
	sw	$a3, XCT_SR($k1)	# save $sr on stack
Set sp	move	$sp, $k1	# sp = sp - XCPSIZE
Enable nested exceptions			
	andi	$v0, $a3, MASK1	# $v0 = $sr & MASK1, enable exceptions
	mtc0	$v0, $sr	# $sr = value that enables exceptions
Call C exception handler			
Set $gp	move	$gp, GPINIT	# set $gp to point to heap area
Call C code	move	$a0, $sp	# arg1 = pointer to exception stack
	jal	xcpt_deliver	# call C code to handle exception
Restoring state			
Restore most GPR, Hi, Lo	move	$at, $sp	# temporary value of $sp
	lw	$ra, XCT_RA($at)	# restore $ra from stack
	...		# restore $t0, ..., $a1
	lw	$a0, XCT_A0($k1)	# restore $a0 from stack
Restore Status Register	lw	$v0, XCT_SR($at)	# load old $sr from stack
	li	$v1, MASK2	# mask to disable exceptions
	and	$v0, $v0, $v1	# $v0 = $sr & MASK2, disenable exceptions
	mtc0	$v0, $sr	# set Status Register
Exception return			
Restore $sp and rest of GPR used as temporary registers	lw	$sp, XCT_SP($at)	# restore $sp from stack
	lw	$v0, XCT_V0($at)	# restore $v0 from stack
	lw	$v1, XCT_V1($at)	# restore $v1 from stack
	lw	$k1, XCT_EPC($at)	# copy old $epc from stack
	lw	$at, XCT_AT($at)	# restore $at from stack
Restore ERC and return	mtc0	$k1, $epc	# restore $epc
	eret	$ra	# return to interrupted instruction

Thrashing – Working set

- Si un programa accede a una MV mayor que la MP disponible, se ejecutará muy despacio
- Dicho programa estará intercambiando páginas constantemente entre la memoria y el disco, ***thrashing***
- Si un programa entra en *thrashing*, lo mejor es ejecutar dicho programa en una máquina con más memoria o añadir más memoria a la actual
- Otra alternativa es rediseñar los algoritmos y las estructuras de datos de nuestro programa para tratar de reducir el número de páginas que un programa utiliza de forma simultánea, reducir el ***working set***