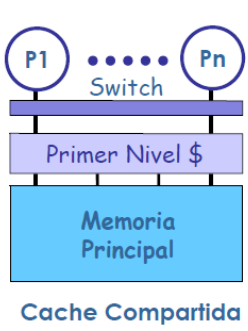
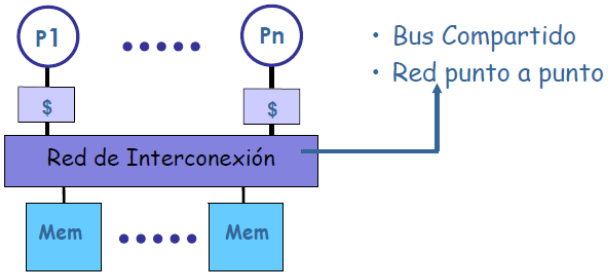
❖Tema 6. Coherencia de Memoria.

➢ 6.1 El problema de la coherencia de memoria

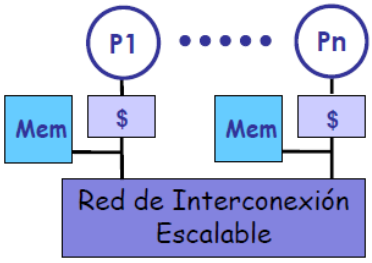
Un problema de consecuencias graves en los sistemas con más de un procesador es la coherencia de la memoria. Si existen diferentes niveles de memoria (Principal, cachés L1, L2…) existe la posibilidad de que para la misma posición de memoria se obtengan resultados diferentes dependiendo de quién lo solicite y en qué momento. Tenemos muchas formas de gestionar procesadores conectados:



Cache Compartida

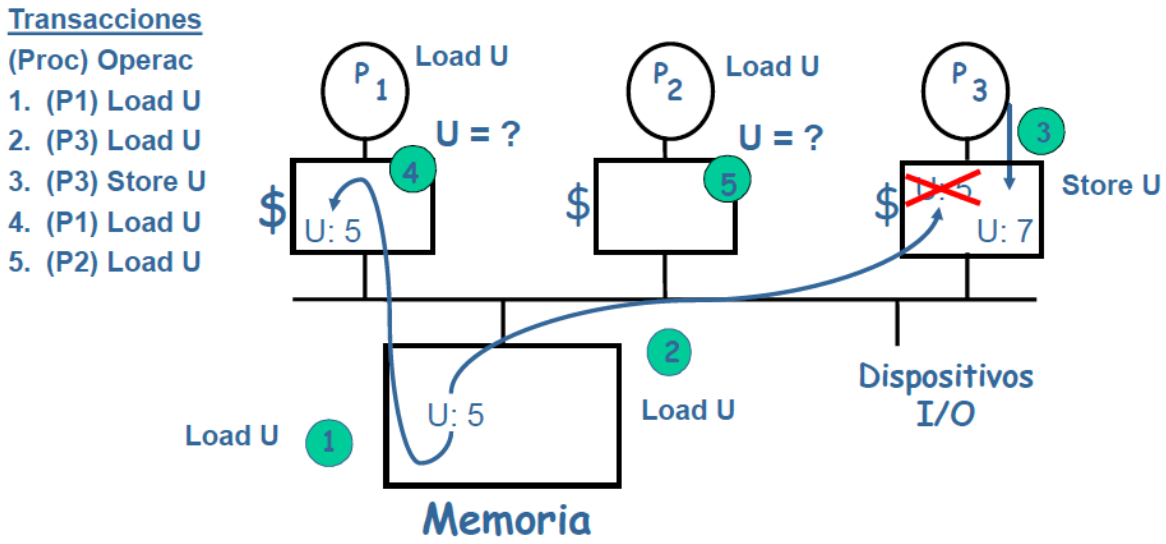


UMA, SMP



NUMA: Memoria Distribuida

Los procesadores que comparten memoria principal, pero no comparten caché, tienen el problema de la incoherencia entre los Load y los Store.



El Store del P3 si se hace sobre la caché 3 los demás procesadores no tienen conocimiento del cambio del valor de la variable U y tomarían en sus Load posteriores valores desactualizados.

6.1.1 Uniprocesadores, problemas con DMA.

Existe una situación peculiar en los sistemas con un solo procesador que también puede provocar incoherencias en la memoria. Las operaciones de Entrada/Salida gestionadas por DMA.

- La DMA escribe en memoria principal, pero en la caché del procesador sigue habiendo valores antiguos.
- La DMA lee de memoria principal, pero puede leer valores desactualizados si el procesador ha escrito en su caché y no actualizó la principal.

Para resolver las contingencias con la DMA se suelen explorar algunas "**estrategias de Grano Grueso**":

- Evitar usar la caché en operaciones que involucren E/S. Se marcan las zonas de memoria como No-Cacheables.
- Cuando el Sistema operativo detecte una operación de E/S, las páginas se borran de la caché. Cuando la DMA escriba en memoria, el procesador tendrá que tomar las páginas otra vez.
- Usar la caché con la DMA, si ésta tiene acceso a la caché se evita el problema. La cuestión es que a veces las operaciones de E/S traen mucha información de golpe para evitar tener que abrir canales constantemente y esto podría quitarnos demasiado espacio.

6.1.2 Mecanismos para controlar la coherencia en Multiprocesadores

Existen diferentes protocolos para mantener la coherencia de la memoria con estructuras diversas. En cualquiera de ellos se toman acciones de Invalidación de Escritura o de Actualización en Escritura.

Invalidación de escritura / Coherencia Dinámica: Al escribir en un bloque de caché se invalidan todas las copias de ese mismo bloque en las cachés de los otros procesadores. Ante fallos de lectura se puede usar write-through o write-back.

- Si es write-though la MP siempre está actualizada. Se toma de ahí.
- Si es write-back se realiza la búsqueda del dato en la caché que lo tenga actualizado y se le proporciona al procesador y a la MP.

Actualización de Escritura: Al escribir en un bloque de caché se actualizan todas las copias que existan en otras cachés. Típicamente se usa con write-through.

Dependiendo de cómo estén confeccionadas las redes de interconexión de los procesadores se van a usar unos protocolos u otros.

- Broadcast de Cachés Eficiente (Protocolos Snoopy)
  - Snoopy de 2 estados
  - Snoopy con Write-Back
  - MSI (3 estados)
  - MESI (4 estados)
- Sin Broadcast (Basados en Directorio)
  - Centralizado
  - Distribuido
    - Plano
    - Jerárquico

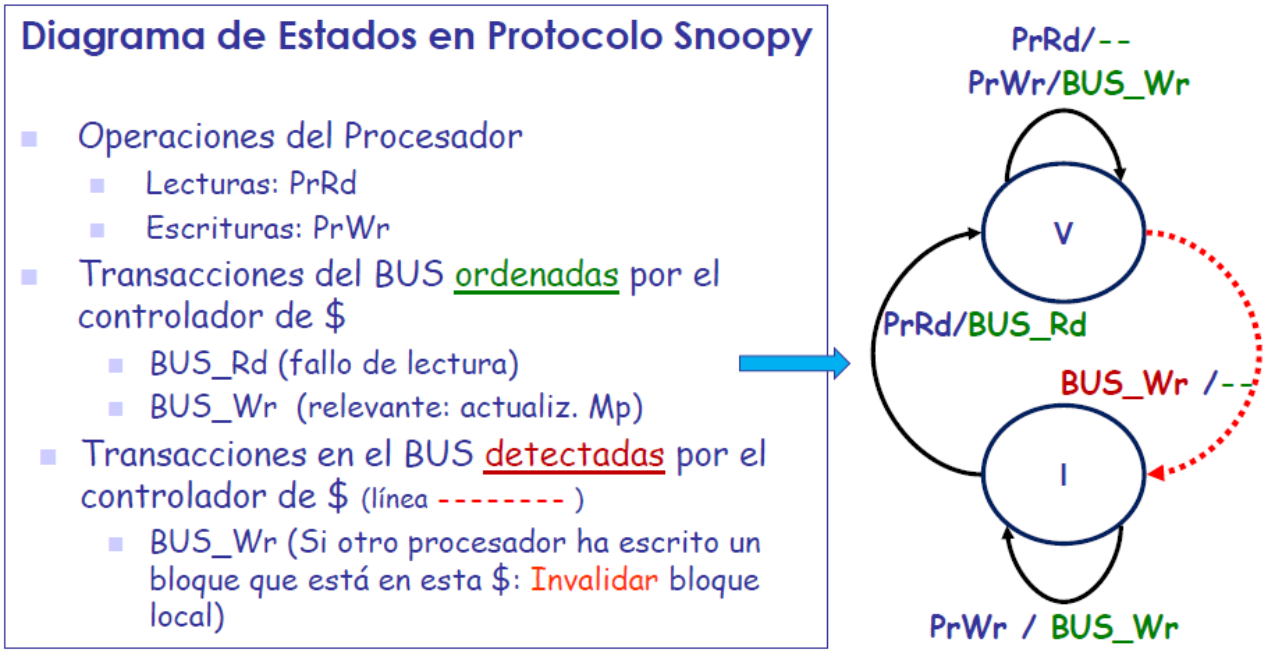
➢ 6.2 Protocolo Snoopy de 2 estados

Los protocolos Snoopy tienen todos una característica común, todas las cachés se comunican con la MP mediante un bus de datos común. El bus tiene un controlador que chequea las transacciones para detectar escrituras. Es un controlador común a todas las cachés.

Cada bloque de caché puede estar en dos estados diferentes

- Válido
- Inválido (o también ausente)

En este protocolo se considera la política de invalidación de escritura, write-through y no-write-allocate.



Cada bloque de caché, en cada caché local, está monitorizado por el controlador de Caché en una tabla. Ahí se indica su estado. Todos los bloques comienzan en estado de Inválido.

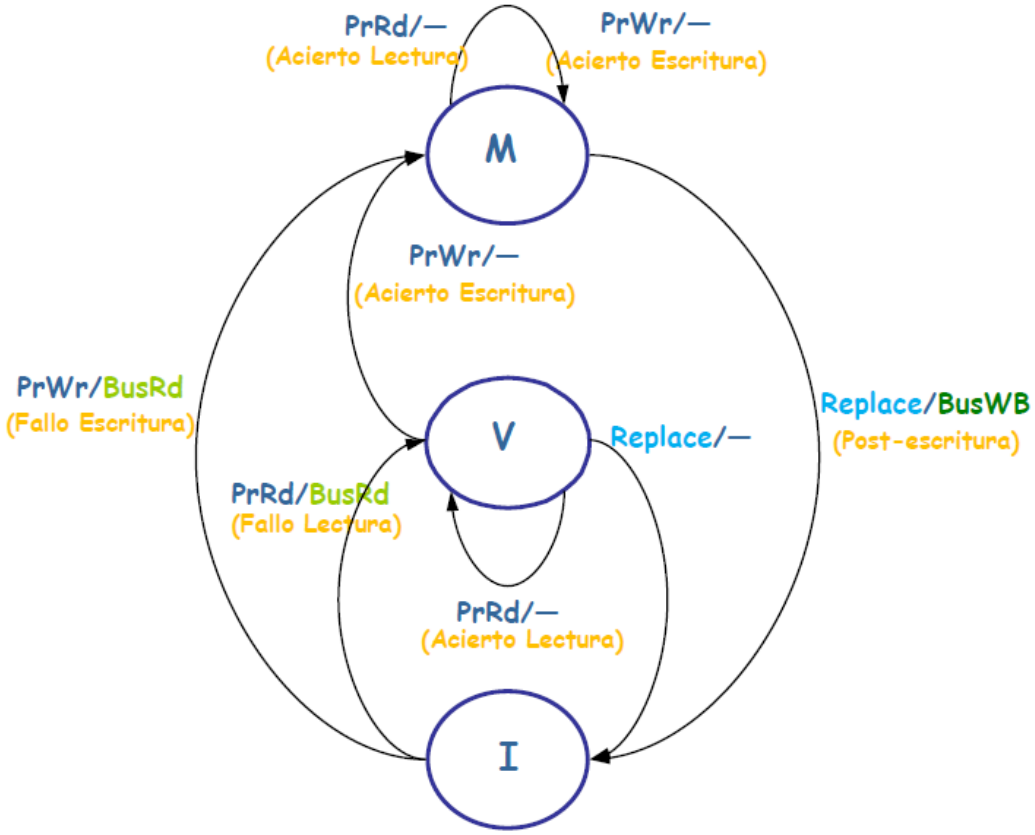
Cuando se detectan operaciones en el Bus sobre cachés externas o cuando su procesador realiza acciones de memoria se producen transiciones de estado (como una máquina de Mealy). La línea de puntos de la derecha es la invalidación del bloque al detectar escrituras de otras cachés.

Como usamos política no-write-allocate los fallos de escritura no hacen que salgamos de Inválido, el bloque no se trae a caché.

La memoria principal siempre está actualizada porque tenemos write-through.

➢ 6.3 Protocolo Snoopy con Write-Back

El protocolo de dos estados tiene un uso poco eficiente del ancho de banda de memoria. Para mejorarlo añadimos un 3er estado y cambiamos a política de write-back y write-allocate.



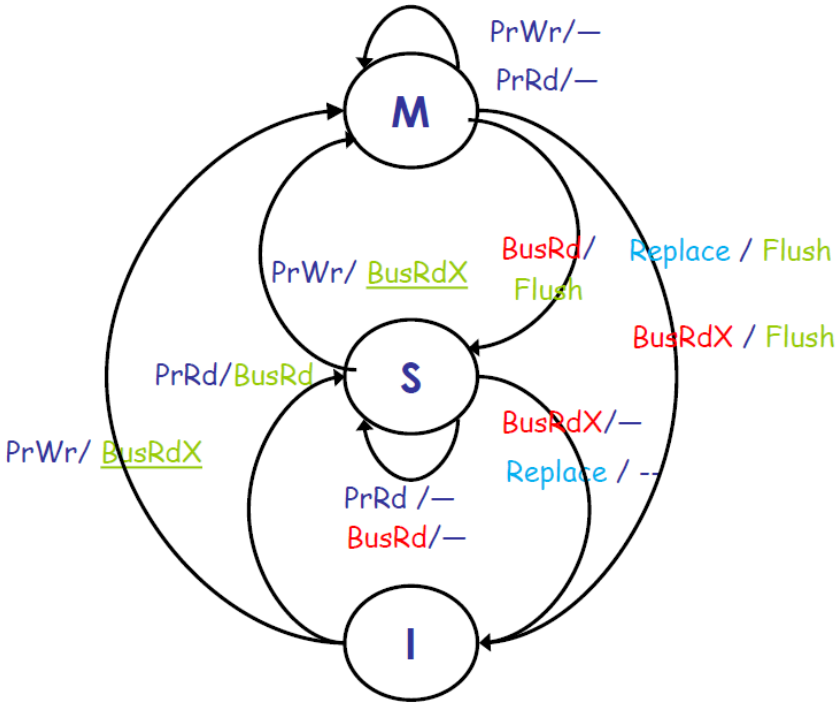
El estado Modificado (dirty) nos indica que el bloque tiene una escritura pendiente de pasar a la MP. Solo se produce la Post-escritura cuando el bloque es desalojado (y por tanto pasa a inválido) desde el estado M.

En este protocolo se usa la Actualización de Escritura en vez de la invalidación. Se puede ver que no hay ninguna transición forzada por las operaciones en el Bus de Memoria. Se pasa a estado de Inválido si el bloque es reemplazado por sobreescritura, y si estaba Modificado se vuelca en el Bus.

➢ 6.4 Protocolo MSI

Cada bloque de caché puede estar en 3 estados.

- Modificado (en exclusiva).
- Shared.
- Inválido.



La acción de "Flush" implica volcar el bloque en el Bus de memoria para que pueda ser leído por la MP o por otras cachés.

El estado M se caracteriza por indicarnos que el bloque está modificado, pero en exclusiva, es decir, somos la única caché que tiene el valor actual. La MP está desactualizada. Podemos seguir haciendo escrituras en nuestro bloque M sin tener que notificarlo a nadie.

Si en el Bus se produce una petición de lectura de nuestro bloque (exclusivo), pasamos a ser compartido. Esto no cambia mientras no haya nuevas escrituras.

La acción importante sobre el Bus es "BusRDX". Esta acción les dice a los demás bloques de caché que acabas de hacer una escritura sobre un bloque y por tanto te conviertes en exclusivo. Podemos ver que si recibimos BusRdX del Bus, siempre pasamos a inválido.

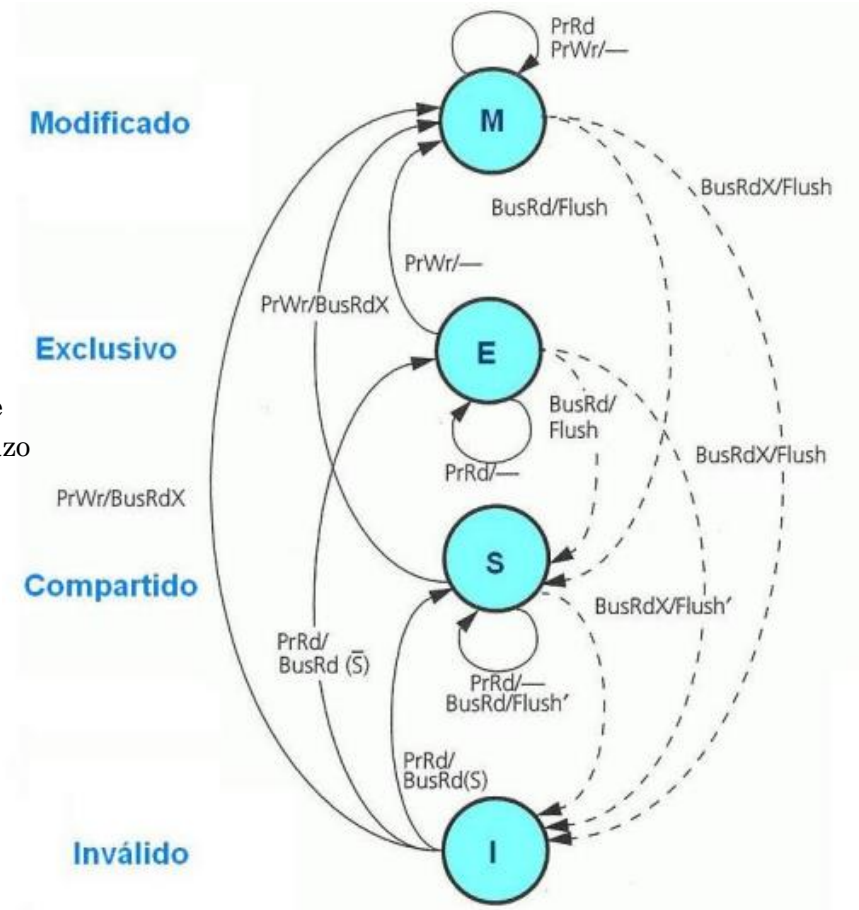
La transición S→M se conoce con el nombre de "BusUpgrade". Esto es porque la caché ya tenía el bloque de datos y simplemente notifica a las demás que ha modificado el contenido. El resto de cachés tendrán que pasar a inválido.

➢ 6.5 Protocolo MESI

Cada bloque de caché puede estar en 4 estados. Añadimos uno intermedio al MSI.

- Modificado.
- Exclusivo. (Clean)
- Shared.
- Inválido.

(*) Faltan las líneas de transición por reemplazo de bloques.

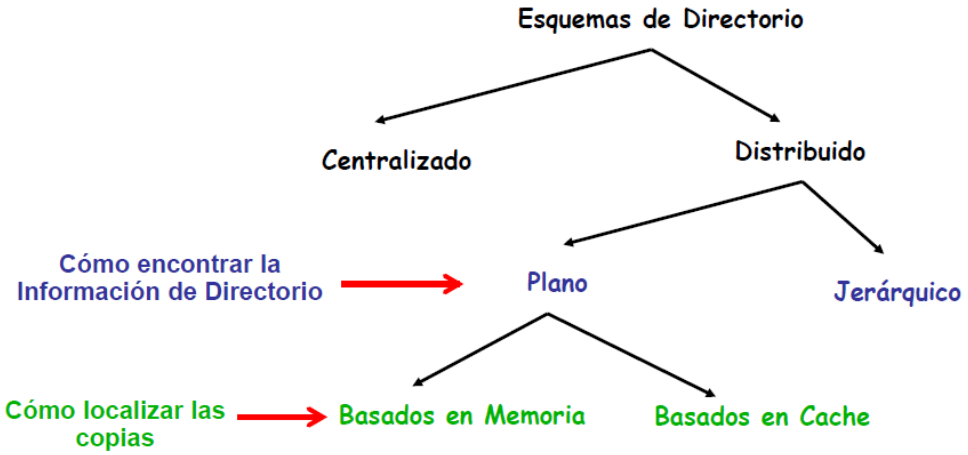


El nuevo estado E nos indica que el bloque es exclusivo, es la única copia entre todas las cachés y además no está modificado respecto de la MP. Por eso si escribimos en él pasamos a estado M, pero no necesitamos hacer notificaciones al Bus.

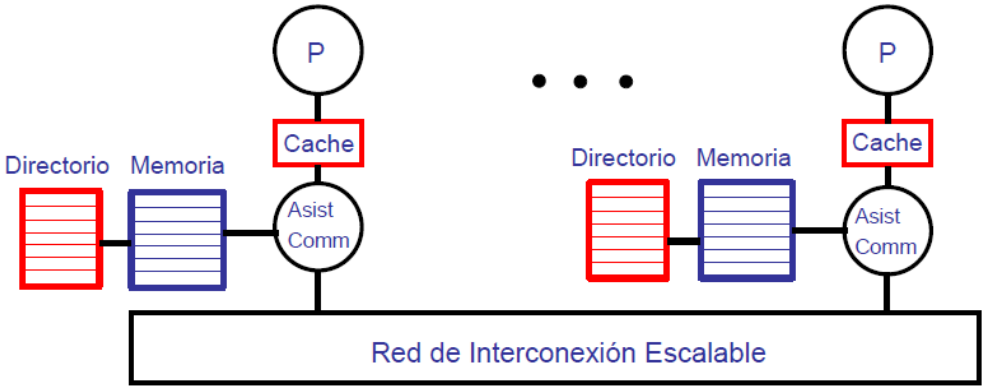
El estado M sigue siendo exclusivo.

Existe una línea en el bus llamada "Línea S". Cuando hacemos una petición de lectura desde I, se provoca un fallo de lectura y pasamos a S o a E dependiendo de si hay respuesta por esta línea. Lo que hacemos es preguntar si otra caché tiene el bloque ya cargado o no, y pasaríamos a Shared en caso de que sí.

➢ 6.6 Coherencia basada en Directorio

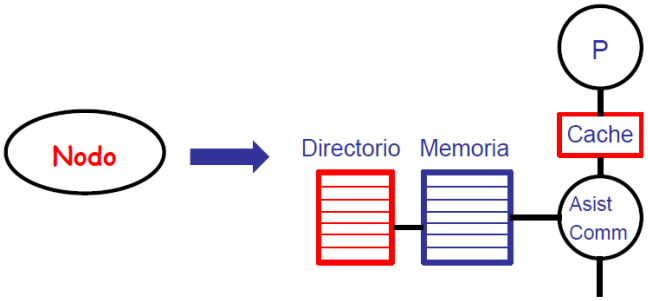


(*) Sólo vamos a estudiar las estrategias de Directorio Plano. Tanto basado en Memoria como en Caché.



Se dispone de un esquema de memoria Distribuido con cachés separadas. Cada procesador "ve" un espacio global de direcciones físicas compartido. La memoria física total es la suma de todas las memorias de los procesadores, por lo tanto podemos decir que "cada procesador posee una porción de la memoria física".

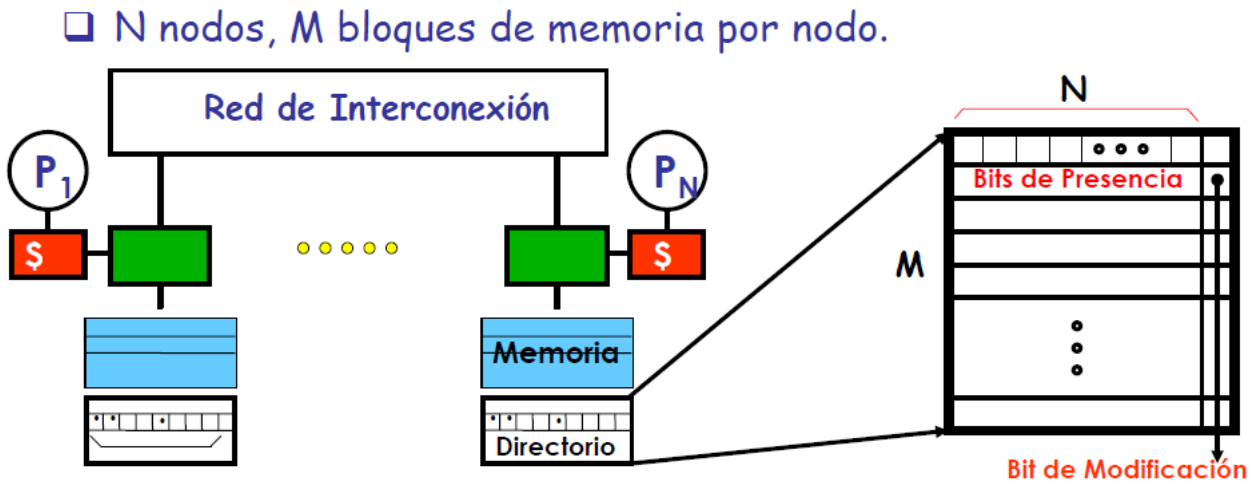
Definición de NODO:



Cada nodo está compuesto por el procesador, su caché, un asistente de comunicación, una porción de la memoria y un directorio. P se comunica con el resto de componentes mediante el Asistente de Comunicación. El asistente se encarga de multitud de tareas, atender al procesador, a la red de interconexión (otros nodos), gestionar las transacciones con la memoria física "local" y actualizar el contenido del directorio.

6.6.1 Directorio plano basado en Memoria

El protocolo de comunicaciones entre nodos tiene cierta complejidad y está influido por la información que contiene el directorio. El directorio que vamos a describir ahora se conoce como **directorio sin optimización** (Full bit Vector). Está localizado en una zona fija y por tanto se dice que es directorio Plano.



Cada directorio es una tabla de M filas y N+1 columnas. La porción de memoria física que me pertenece (a mi nodo) contiene M bloques y por tanto mi directorio debe saber quién contiene copias de mis bloques y si éstos están modificados o no (y si lo están, mi memoria está desactualizada ya que se usa política de Write-Back).

Los bits de presencia me informan de en qué nodos (cachés) está ese bloque cargado (incluida mi propia caché).

Para entender el funcionamiento de la red haremos unas definiciones conceptuales que nos van a simplificar los procesos:

Nodo Peticionario (Local): es el nodo que emite una petición de bloque ya sea para lectura o para escritura que no pertenece a su espacio de memoria.

Nodo Origen (Home): es el nodo que posee en su porción de memoria física el bloque que solicita el nodo Local. Esto se obtiene debido a que las emisiones de direcciones son globales y el asistente de comunicación informa de que esa porción de memoria está fuera de nuestro nodo. Hay que pedirlo a Home.

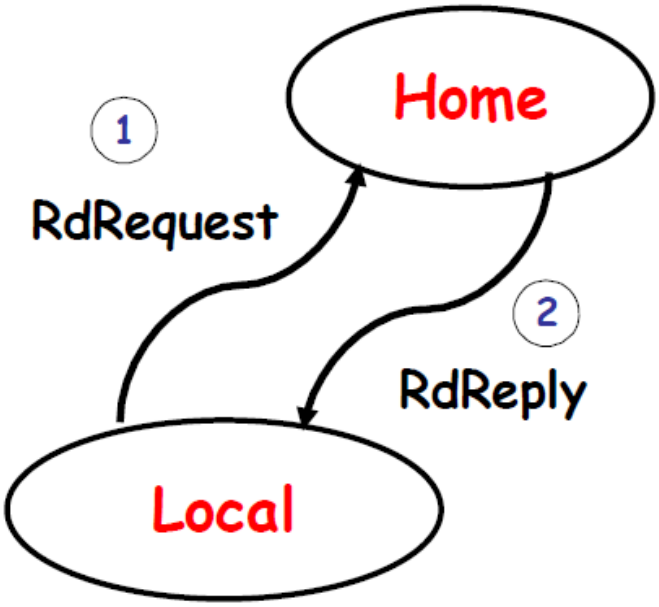
Nodo Propietario (Owner): si el bloque pedido por Local no ha sido cargado por nadie, el propietario es simplemente Home. Sin embargo si otro nodo lo pidió para llevarlo a su caché y ha realizado una escritura en él, entonces el Owner es el nodo que contiene la copia más reciente del bloque en su caché y además es exclusivo (no puede haber 2 owner de un bloque).

Nodo Compartido (Sharer): como estudiamos en el protocolo Snoopy-MESI (aquí usamos también 4 estados) los bloques vistos desde una caché pueden estar en 4 diferentes estados. En la política de directorio plano basada en memoria un bloque sólo puede existir en estado modificado en un único nodo (owner). Sin embargo puede haber muchos nodos con el bloque cargado, pero siempre tiene que estar limpio, es decir, puede haber muchos Sharer para el mismo bloque, pero todos tienen que contener la misma información. Además si el bloque no tiene un nodo que lo posea (en estado M) quiere decir que la memoria física del nodo Home tiene la información actualizada.

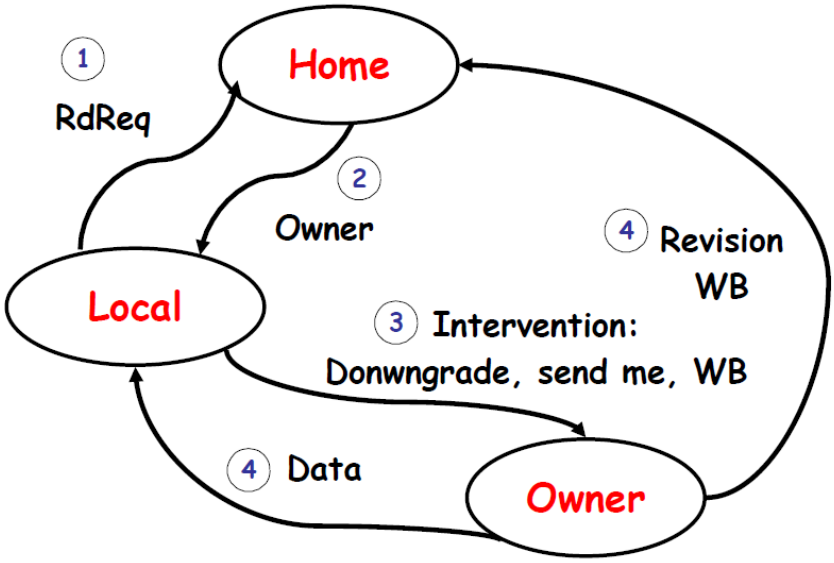
Nodo Exclusivo (Exclusive): igual que en MESI este nodo posee en exclusiva el bloque cargado en su caché (incluso aunque sea de su propia memoria), pero siempre en estado de limpio. Si es exclusivo y sucio se considera estado M.

Ahora vamos a estudiar el protocolo de comunicaciones que tiene lugar cuando se producen lecturas o escrituras, con fallo en caché y salida a la red.

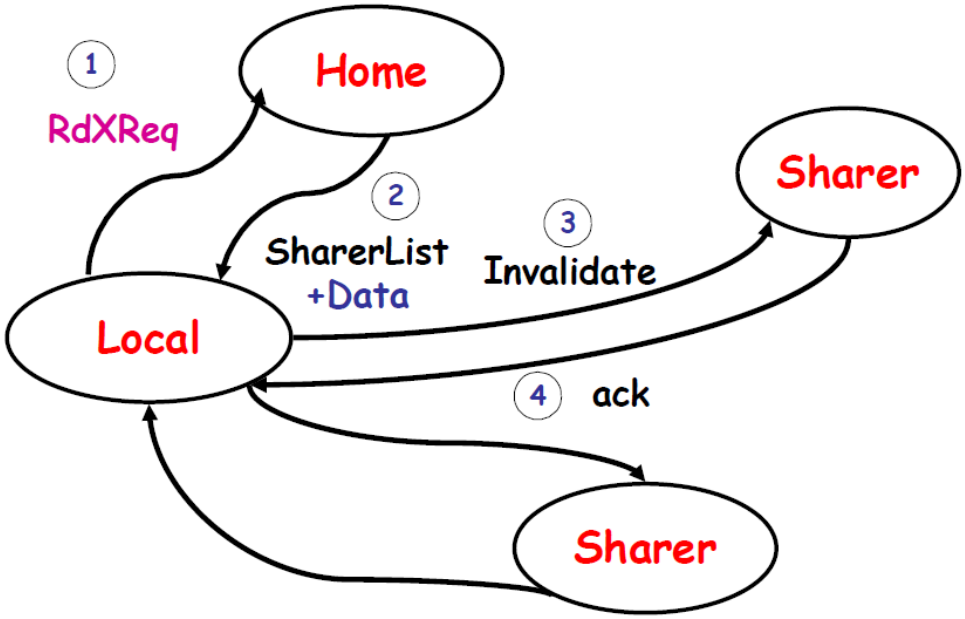
- Obtener bloque para lectura
  - Caso 1: Dirty bit OFF (el bloque está actualizado en la memoria del nodo Home)
    - 1. El nodo local pide el bloque
    - 2. El Home lo proporciona (y actualiza el directorio)
    - Sólo son necesarias dos transacciones en la red



- Obtener bloque para lectura
  - Caso 2: Dirty bit ON (El bloque no está actualizado en memoria del nodo Home)
    - 1. El nodo local pide el bloque
    - 2. El Home proporciona la identidad del nodo Propietario
    - 3. El nodo local hace una petición de lectura al propietario
    - 4. El nodo propietario proporciona el bloque y hace WB al Home

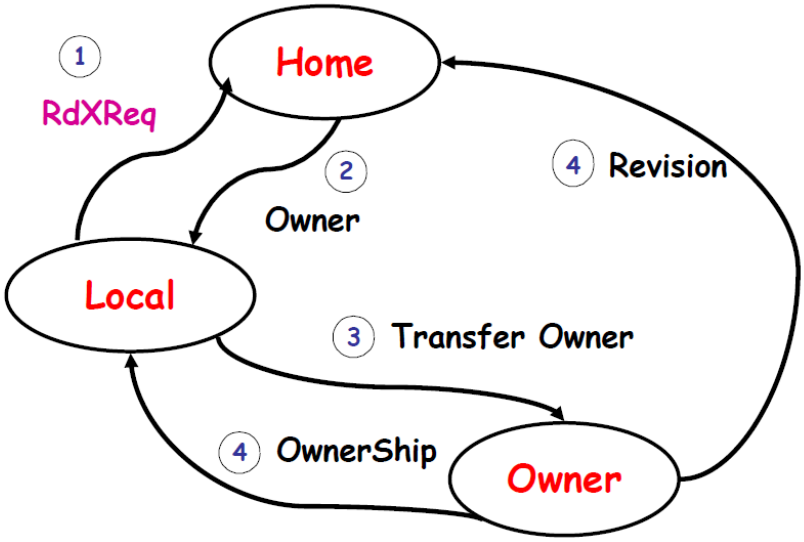


- Obtener bloque para escritura
  - Caso 1: Dirty bit OFF (hay varias copias válidas. Mp actualizada)



❑ Obtener bloque para escritura

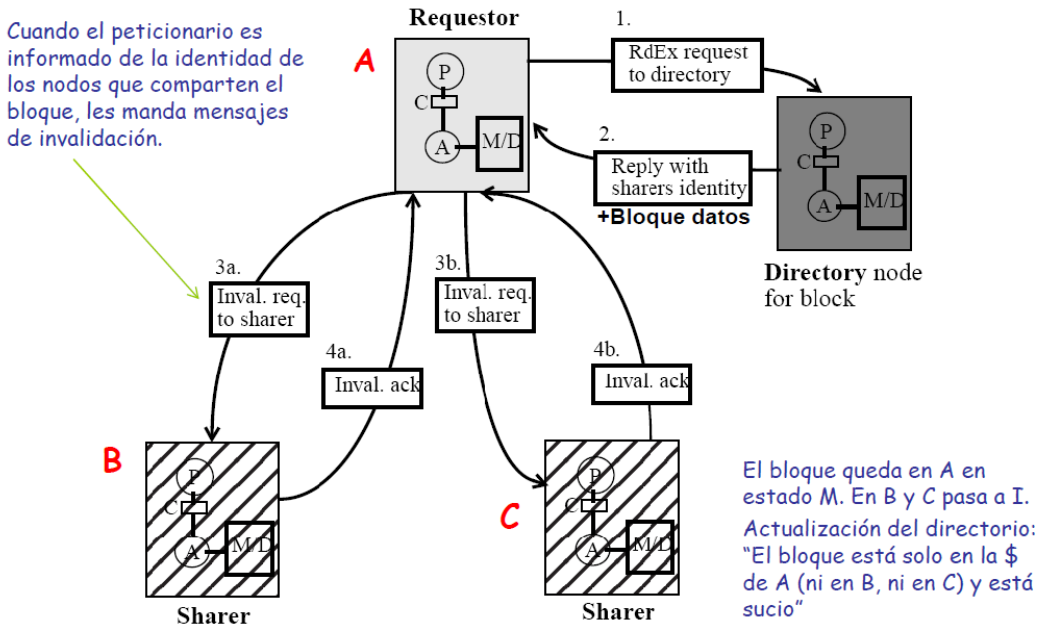
- Caso 2: Dirty bit ON (hay una copia válida en el propietario. Mp no actualizada)



En todos los esquemas de transacciones que acabamos de ver se tiene que estar actualizando el directorio del Nodo Home. Por cambio de Owner, o porque Local se convierte en un nuevo Sharer.

Un esquema que no requiere de peticiones de bloque, pero sí de comunicación exterior ocurre cuando un nodo escribe sobre un bloque de su caché. Debe notificarlo a la red por si hubiese nodos con ese bloque en estado Sharer ya que deben ser invalidados. Si hay fallo en escritura ocurriría todo esto:

❑ Fallo de escritura sobre un bloque compartido por dos procesadores (bloque limpio)



El fenómeno de la sobrecarga

Es interesante preguntarse dónde se guardan los directorios y cuánto espacio ocupan. El porcentaje de espacio de almacenamiento perdido para crear los directorios se conoce como sobrecarga de almacenamiento.

La sobrecarga es proporcional a N*M, donde recordemos que N es el número de nodos y M la cantidad de bloques que gestiona cada nodo.

Pongamos un ejemplo, bloques de caché de 64Bytes (llamado también tamaño de la línea). Si disponemos de $N = 64$ nodos quiere decir que cada entrada del directorio debe tener mapeado los nodos con 64 bits.

$$Sobrecarga = \frac{64\ bits}{64 \cdot 8\ bits} = 0{,}125 \quad \rightarrow \quad 12{,}5\%$$

Con bloques de 64Bytes y 1024 nodos:

$$Sobrecarga = \frac{1024\ bits}{64 \cdot 8\ bits} = 2 \quad \rightarrow \quad 200\%$$

Estrategias para reducir la sobrecarga:

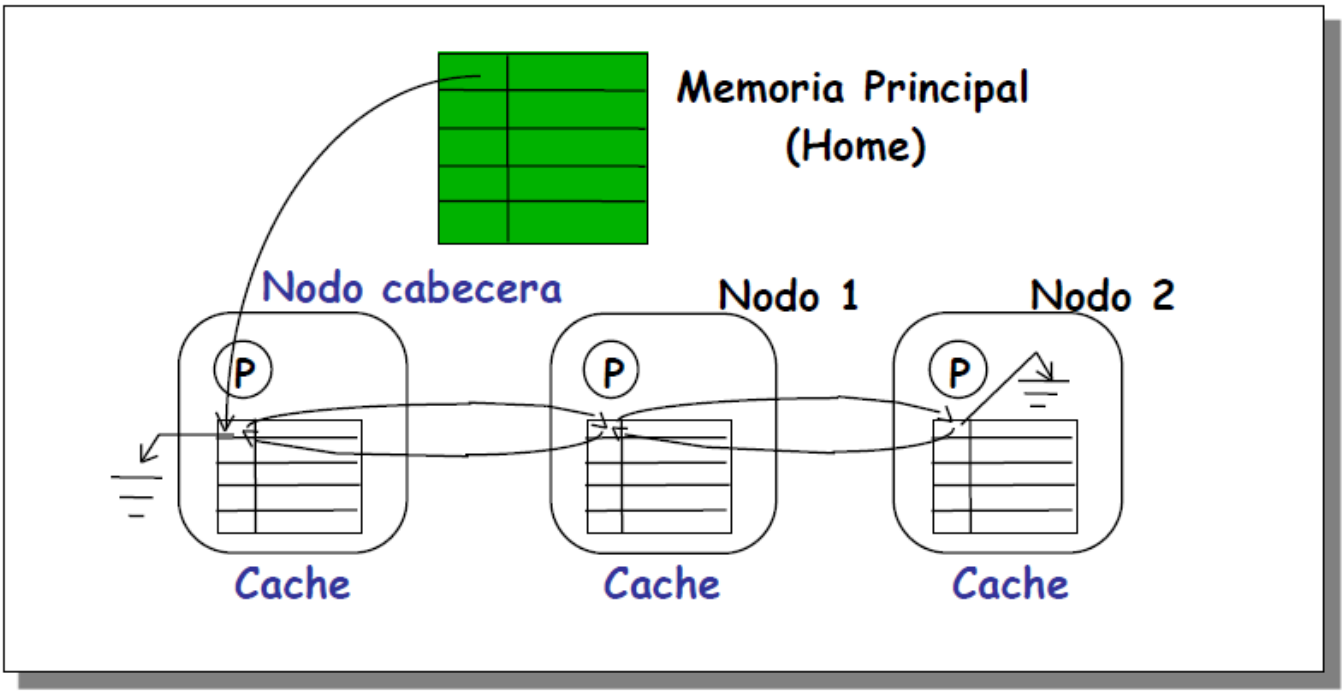
- Utilizar nodos con varios procesadores de manera que se usa un solo bit por nodo reduciendo el ancho del directorio.

- En vez de almacenar un bit por nodo se almacena una cantidad de punteros a nodo que tienen cargado ese bloque. Este número de punteros no debe ser muy grande para que disminuya el ancho del directorio.

- Organizar el directorio como una caché en lugar de tener una fila para cada bloque que tenga que gestionar mi nodo. Se hacen entradas para representar el estado de un bloque que ha sido sacado de la memoria física de tu dominio. Esto puede crear conflictos de reemplazamiento, pero reduce la altura del directorio.

6.6.2 Directorio plano basado en Caché

El sistema de directorio plano basado en memoria tiene un alto gasto de memoria por sobrecarga de almacenamiento y además requiere de muchísimo tráfico de mensajes en la red de interconexión dando un enorme trabajo a los asistentes de comunicación (y esto puede provocar latencias de espera por canales ocupados).



Existe una única memoria física externa a todos los nodos. El nodo Home es simplemente la memoria principal. Los bloques de memoria que se pueden pasar a las cachés de los nodos con procesador están mapeados en el directorio de Home.

A diferencia del directorio basado en memoria el directorio de Home no contiene la lista de todos los nodos que han cargado un determinado bloque sino que simplemente contiene la dirección del primer nodo que lo solicitó (nodo de cabecera).

Todos los demás nodos que tienen copia de ese bloque están enlazados mediante punteros con una estructura de lista enlazada.

Fallo de Lectura

El nodo peticionario (Local) envía a Home una petición de lectura de un bloque. Si no lo tiene nadie se lo facilita y se convierte en el nodo de cabecera.
En caso de que lo tuviesen otros Home le envía la identidad del nodo de cabecera. El nodo Local envía petición a Cabecera para convertirse él en el nuevo nodo de cabecera.
Los datos son enviados por la Cabecera a Local y se actualiza el estado de cabecera en el directorio de Home.

Fallo de Escritura

El nodo local puede que ya estuviese en la lista enlazada de nodos que comparten el bloque, pero quiere modificarlo. Se procede a recorrer toda la lista para invalidar las copias de todos los nodos y convertirse en cabecera. Todos los nodos deben enviar un ACK de invalidación al nodo Local.

Igual que en la estrategia de directorio plano basado en memoria si el bloque de caché está sucio solo puede haber un nodo que lo posea. Con una petición de lectura o escritura posterior se tiene que realizar un Write-Back además del intercambio del nodo de cabecera.

Ventajas frente al esquema basado en Memoria

- Menor sobrecarga de almacenamiento. En el directorio de Home sólo hay un puntero a cabecera y en los nodos con caché dos punteros (uno para anterior y otro para siguiente).

- Descentralización del trabajo del asistente de conexión. Los envíos de invalidación no se hacen todos desde el mismo nodo sino que se van enlazando.

- La lista enlazada guarda de forma implícita el orden de los nodos que han ido pidiendo el bloque de memoria.