Ingeniería de los Computadores

Unidad 5. Memoria en arquitecturas paralelas

Consistencia de memoria

Problema: Transmisión de información entre procesadores en memoria compartida

Ejemplo:

```
/* El valor inicial de A y flag es 0 */
 P1                P2
A=1               while (flag == 0); /*bucle vacío*/
flag=1            print A
```

P2 ha de realizar una **espera activa** hasta que *flag* cambie a '1'.

Después imprimirá A = '1' (suponiendo que en P1 *"A=1"* es una instrucción anterior a *"flag=1"*) → ¿COHERENCIA DEL SISTEMA DE MEMORIA? → Escrituras en orden de programa.

Pero … y si "flag=1" se ejecuta antes de "A=1" ¿?

En un **espacio de direcciones compartido** es necesario un **modelo de consistencia de la memoria**. Su objetivo es **especificar restricciones en el orden de las operaciones en la memoria** y proporcionar una visión uniforme de las mismas a los demás procesadores, logrando una abstracción de las operaciones de memoria independiente de la localización de las operaciones de memoria (módulos) y de los procesos involucrados.

Introducción

Consistencia de memoria

- "Un modelo de consistencia de memoria especifica el **orden** en el cual las operaciones de acceso a memoria deben **parecer** haberse realizado (operaciones de lectura y escritura)"

- En un procesador (o sistema **monoprocesador**), el orden en el que deben parecer haberse ejecutado los accesos a memoria es el **orden secuencial especificado por el programador** (o la herramienta de programación en el código que añade), denominado **orden de programa**.

- Tanto el hardware como la herramienta de programación si que pueden alterar dicho orden, para mejorar las prestaciones, pero debe parecer en la ejecución del programa que no se ha alterado.

Introducción

Consistencia de memoria → Se debe garantizar que:

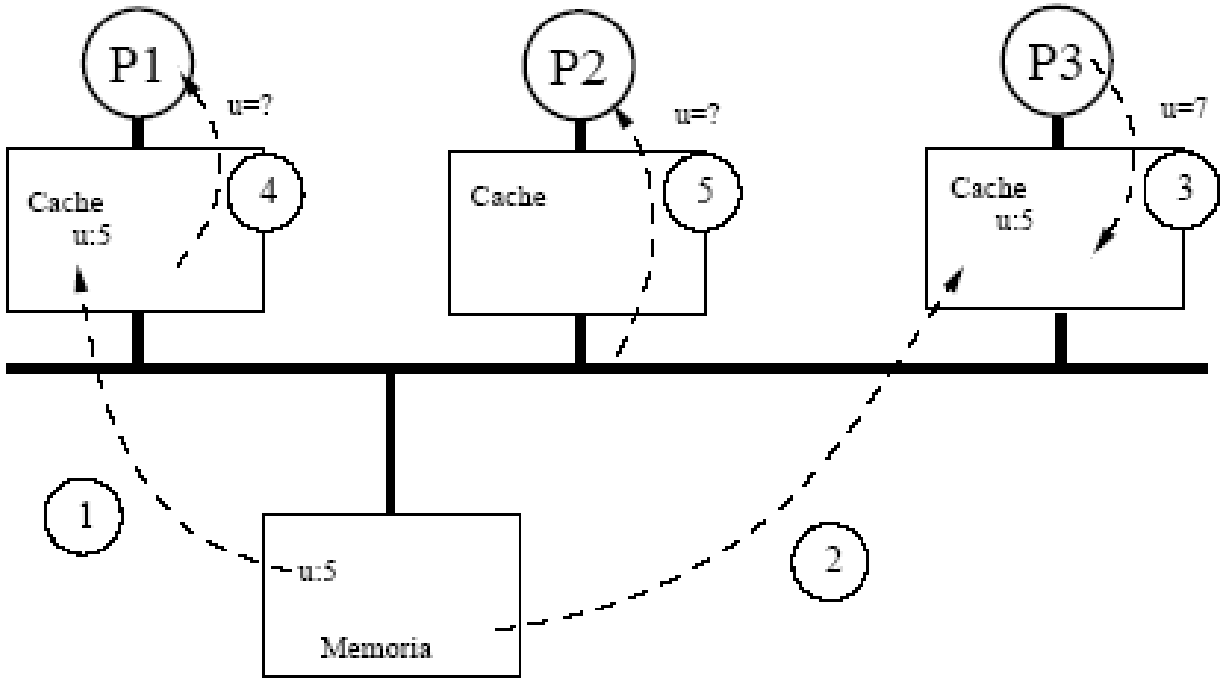
- Cada lectura de una dirección, proporcione el último valor escrito en dicha dirección **(dependencia de datos lectura después de escritura)**
- Tras varias escrituras en una dirección se debe retornar el último valor escrito **(dependencia escritura después de escritura)**
- Si se escribe en una dirección a la que previamente se ha accedido para leer, no se debe obtener en la lectura previa el valor escrito posteriormente en la secuencia del programa especificada por el programador **(dependencia escritura después de lectura)**
- No se puede escribir en una dirección si la escritura depende de una condición que no se cumple **(dependencias de control)**

Introducción

Ejemplo 1: 2P, caché *write-through* (cada vez que se escribe una línea de caché se actualiza la memoria principal)

Secuencia	Acción	Caché A	Caché B	X en MP
1	CPU A lee X	1	-	1
2	CPU B lee X	1	1	1
3	CPU A escribe 0 en X	0	1	**0**

Ejemplo 2: 3P, caché *write-through/write-back* (se actualiza la memoria principal escribiendo todo el bloque cuando se desaloja de la caché → aún más problemática)



Introducción

- Un sistema de memoria es **coherente si cualquier lectura** de un dato devuelve el **valor más reciente** escrito de ese dato
- Aspectos críticos del sistema de memoria compartida: los datos devueltos por una lectura (coherencia) y cuándo un valor escrito será devuelto por una lectura (consistencia)
- Un sistema de memoria es coherente si cumple:
  - ✓ **Preservación del orden del programa**: una lectura por un procesador P de una posición X, que sigue a una escritura de P a X, sin que ningún otro procesador haya escrito nada en X entre la escritura y la lectura de P, siempre devuelve el valor escrito por P.
  - ✓ **Visión coherente de la memoria**: una lectura por un procesador de la posición X, que sigue a una escritura por otro procesador a X, devuelve el valor escrito si la lectura y escritura están suficientemente separados y no hay otras escrituras sobre X entre los dos accesos.
  - ✓ **Serialización de operaciones concurrentes de escritura**: Las escrituras a la misma posición por cualquiera dos procesadores se ven en el mismo orden por todos los procesadores.

Introducción

Un sistema de memoria multiprocesador es **coherente** si el resultado de cualquier ejecución de un programa es tal que, para cada localización **es posible construir una hipotética ordenación secuencial de todas las operaciones realizadas sobre dicha localización** que sea consistente con los resultados de la ejecución y en el cuál:

1. Las **operaciones** emitidas por un procesador particular ocurren en **la secuencia indicada** y en el orden en el que dicho procesador las emite al sistema de memoria

2. El valor devuelto por cada operación de lectura es el valor escrito por la última escritura en esa localización en la secuencia indicada

Introducción

Dos estrategias para abordar la coherencia de las cachés:

- Resolución **software**: el **compilador** y el programador evitan la incoherencia entre cachés de datos compartidos.
- Hardware: transparente al programador mediante la provisión de mecanismos **hardware**, esta es la solución más utilizada para mantener la coherencia.

Políticas para mantener la coherencia:

- **Invalidación de escritura o coherencia dinámica** (*write invalidate*): siempre que un procesador modifica un dato de un bloque en la caché, invalida todas las demás copias de ese bloque guardadas en las otras cachés de los procesadores.
- **Actualización en escritura** (write-update o *write-broadcast*): esta política lo que hace es actualizar la copia en las demás cachés en vez de invalidarla.

Clasificación

Clasificación de multiprocesadores atendiendo a la distribución de memoria

- **UMA** (Uniform Memory Access)
- **NUMA** (Non-Uniform Memory Access)
- **COMA** (Cache-Only Memory Architecture)

Multiprocesadores UMA

Tipo **UMA**

- Memoria **centralizada**, **uniformemente compartida** entre procesadores (todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de la memoria)
- Sistema fuertemente acoplado: alto grado de compartición de recursos (memoria) entre los procesadores → dependencia funcional entre ellos
- Cada procesador puede disponer de caché
- Periféricos compartidos entre procesadores
- Aplicaciones de propósito general y de tiempo compartido por múltiples usuarios
- Sincronización y comunicación entre procesadores utilizando variables compartidas
- Acceso a memoria, periféricos y distribución de procesos S.O.:
  - ✓ Equitativo → Symmetric (shared-memory) Multi Processors (SMPs)
  - ✓ No equitativo → Asymmetric (shared-memory) Multi Processors (AMPs)CC-UMA (Caché-Coherent Uniform Memory Access)

Multiprocesadores NUMA

Tipo **NUMA**

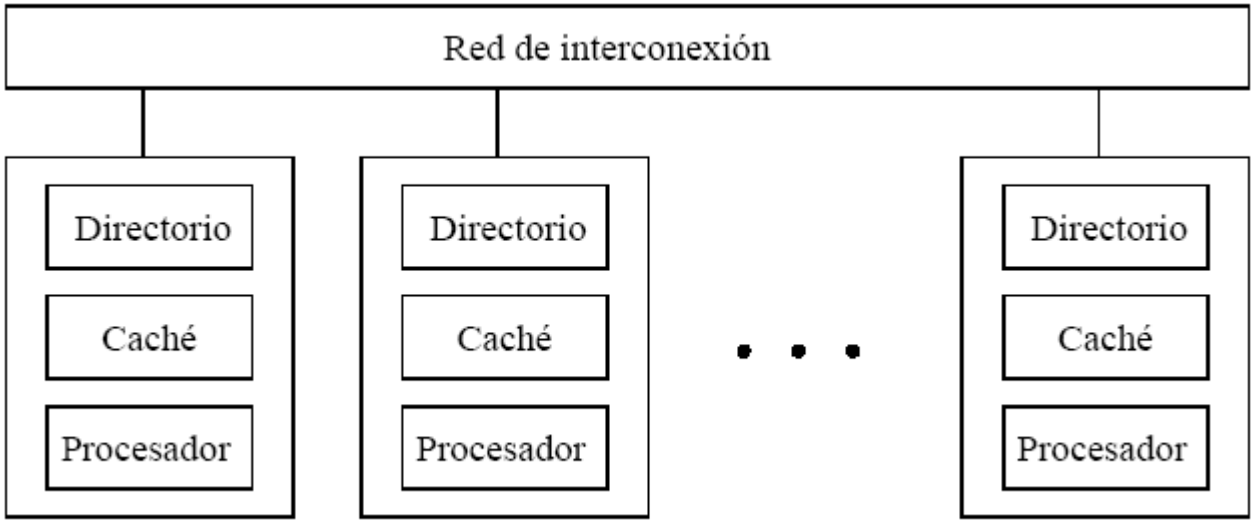
- Memoria compartida con tiempo de acceso dependiente de la ubicación de procesadores
- Sistema débilmente acoplado: cada procesador dispone de una memoria local a la que puede acceder más rápidamente
- Sistema de acceso global a memoria: local, global, local de otros módulos
- CC-NUMA (Caché-Coherent Non-Uniform Memory Access) → Memoria compartida distribuida y directorios de caché

- Ventajas
  - ✓ Escalado de memoria con + coste/rendimiento
  - ✓ Reducción de latencia de acceso a memorias locales

Multiprocesadores COMA

Tipo **COMA**

- Sólo se usa una caché como memoria
- Es un caso particular de NUMA donde las memorias distribuidas se convierten en cachés
- Las cachés forman un mismo espacio global de direcciones
- Acceso a las cachés por **directorio distribuido**



Protocolos de coherencia

- Protocolos de **sondeo** o ***snoopy***:
  - Utilizado en redes donde la difusión (broadcast) es posible
  - Redes basadas en bus
  - Cada caché monitoriza el estado del bus y las transacciones de las demás cachés

- Protocolos basados en directorio:
  - Utilizado en redes donde el broadcast no es posible a causa de la degradación
  - Redes multietapa
  - Se utiliza para ello un **directorio centralizado o distribuido**

Protocolos de coherencia

- Protocolos de **sondeo** o ***snoopy***
  - **Objetivo**: garantizar las transacciones necesarias en el bus para operaciones de memoria y que los controladores de caché observen y actúen en transacciones relevantes
  - **Todas las transacciones** aparecen en el bus y **son visibles** para los procesadores en el mismo orden en el que se producen
  - Utilizado en sistemas multiprocesador con bus y pocos elementos conectados
  - Cada procesador indica el estado de cada línea de su caché
  - La red de interconexión debe permitir broadcast (difusión)
  - Cada caché monitoriza las transacciones de las demás cachés observando el bus
  - Se utiliza un algoritmo distribuido representado como un conjunto de máquinas de estados finitos que cooperan entre sí:
    - Conjunto de estados asociado con los bloques de memoria en las cachés
    - Diagrama de transición entre estados
    - Acciones asociadas a las transiciones entre estados

Protocolos de coherencia

Protocolos de sondeo o *snoopy* – Protocolo de **invalidación en escritura** (***write invalidate***)

- Constituyen las estrategias **más robustas y extendidas**
- Basado en asegurar que un procesador tiene acceso exclusivo a un dato antes de que acceda a él
- Se consigue invalidando todas la líneas de todas las cachés que contengan ese dato que esta siendo escrito en ese momento
- Cada controlador de caché observa las transacciones de memoria (observa el bus) de los otros controladores para mantener su estado interno
- Control de coherencia de caché mediante transacciones de lectura y de lectura exclusiva
- Cuando un procesador quiera leer un dato invalidado, falle su caché y tenga que ir a memoria a buscarlo
- Ejemplo: Protocolos **MSI**, **MESI**, Write Once y Berkeley

Protocolos de coherencia

Protocolos de sondeo o *snoopy* – Protocolo de **actualización en escritura** (***write-update***)

- Menos utilizados que el anterior (*write-invalidate*)
- Cada vez que un procesador escribe un dato, se actualizan las cachés que contienen el dato en los demás procesadores
- Problemas para mantener el ancho de banda bajo control debido a la alta cantidad de transacciones que se generan
- Mejora: escritura en una localización → actualización de cachés relacionadas
- Ejemplo: Protocolos *Dragon* y *Firefly*

Protocolos de coherencia

Protocolos de sondeo (*snoopy*) – **MSI**

- Protocolo de invalidación básico para cachés write-back
- Estados: Inválido (**I**), Compartido (**S**), Modificado (**M**)

  - ✔ **Inválido**: no es válido el bloque
  - ✔ **Compartido**: el bloque esta presente en la caché y no ha sido modificado, **la memoria principal esta actualizada** y **cero o más** cachés pueden tener también una copia actualizada (compartida)
  - ✔ **Modificado**: únicamente este procesador tiene una copia válida, la copia de la memoria principal esta anticuada y ninguna otra caché puede tener una copia válida del bloque (ni en estado modificado ni compartido)

Protocolo de sondeo (*snoopy*) - MSI

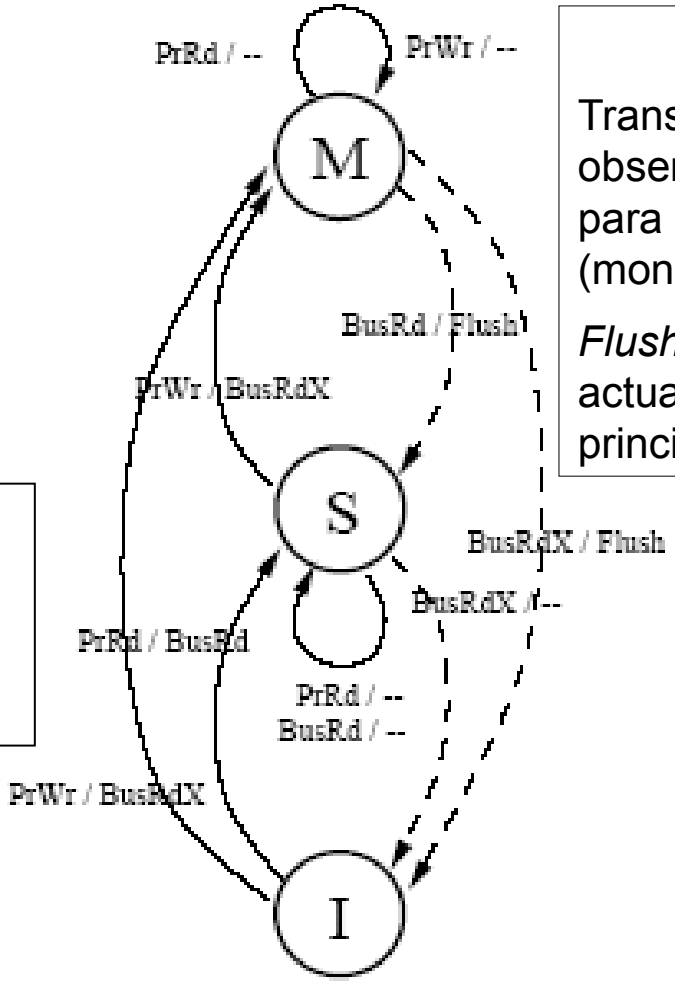
Operaciones del procesador + transacciones que se generan en el bus por dichas operaciones de procesador

Fallo lectura [**miss**] (I) → M,S

Fallo escritura [**miss**] (I) → M,S

Acierto escritura [**hit**] (S) → S, (M) → M

Acierto lectura [**hit**] (S,M)



Transacciones en el bus observadas por las cachés para cambiar su estado (monitorización)

Flush → dato en bus + actualización de memoria principal

Protocolo de sondeo (*snoopy*) – **MESI** (o Illinois)

- Ampliación del protocolo de invalidación de 3 estados. Refinamiento para aplicaciones "secuenciales" que corren en **multiprocesadores** (carga común usada en multiprocesadores de pequeña escala)
- En el MSI el programa cuando lee y modifica un dato tiene que generar 2 transacciones incluso en el caso de que no exista compartición (solo presente en una caché) del dato (*BusRd* y *BusRdX*)
- Estados: Modificado (**M**), *Exclusivo (**E**), Compartido (**S**), Inválido (**I**)
  El estado exclusivo (E): indica que el bloque es la **única copia** (y por tanto exclusiva) del sistema multiprocesador, que **no está modificada** y la **memoria principal está actualizada**
- Al ser exclusivo es posible realizar una escritura o pasar al estado modificado sin ninguna transacción en el bus, al contrario que en el caso de estar en el estado compartido; pero no implica pertenencia, así que al contrario que en el estado modificado la caché no necesita responder al observar una petición de dicho bloque (la memoria tiene una copia válida)
- Publicado por la Universidad de Illinois en 1984

Protocolos de coherencia

Protocolo de sondeo (*snoopy*) - MESI

BusRd(S) → cuando ocurre la transacción de lectura en el bus, se activa la señal S.

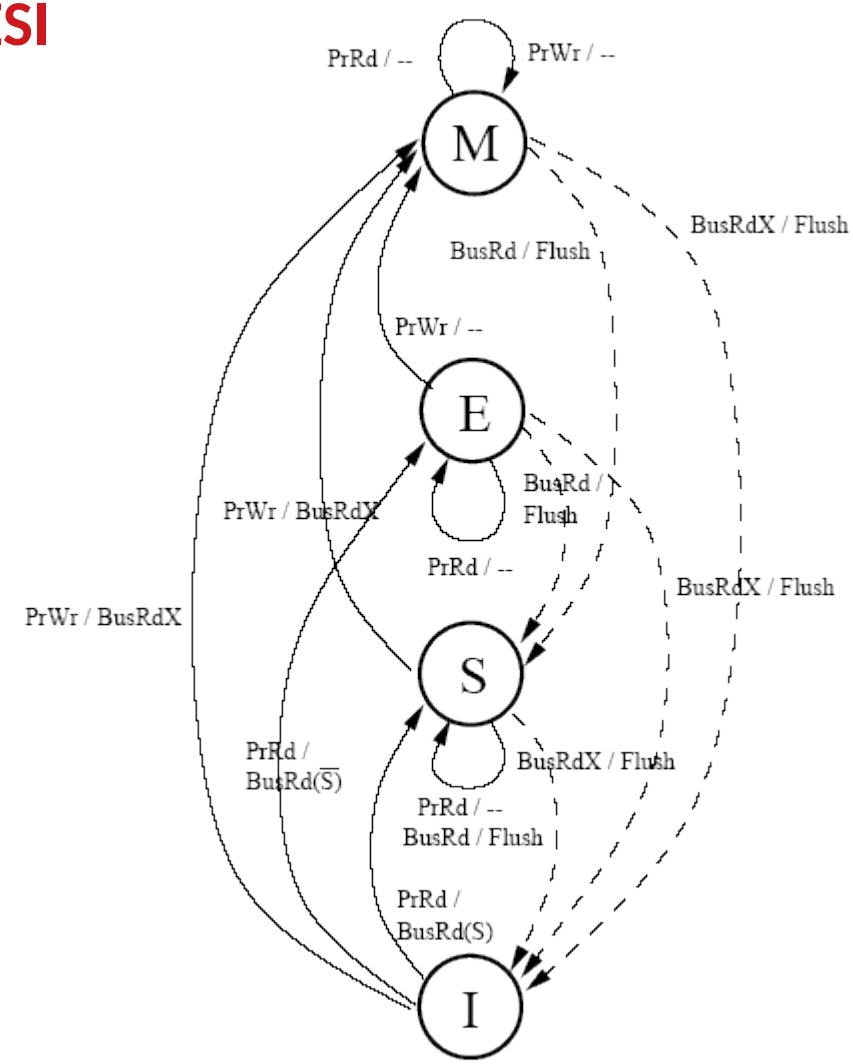
Este protocolo necesita una señal adicional en el bus (señal S) para que los controladores puedan determinar en una transacción *BusRd* si existe otra caché que tenga el mismo bloque

Fallo lectura (I) → M,S,E

Fallo escritura (I) → M,S,E

Acierto escritura (S) → S, (M,E) → O

Acierto lectura (S,M,E)



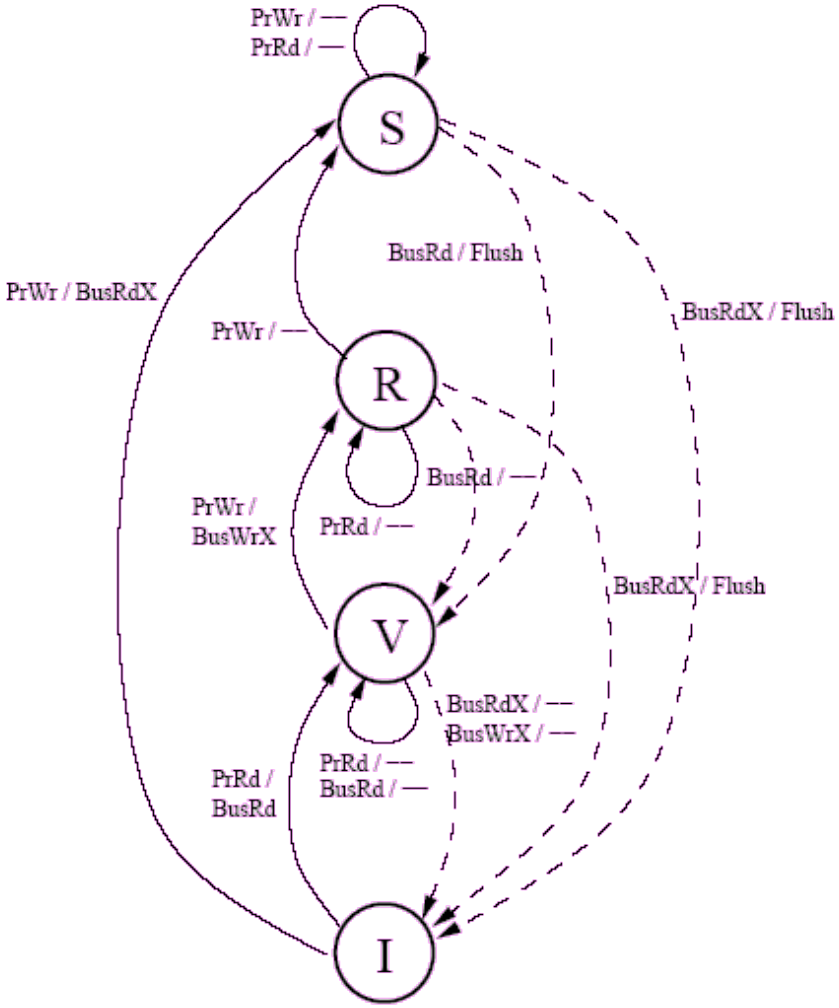
Protocolo de sondeo (snoopy) – Write Once

- Cada línea de caché tiene **dos bits** extra para almacenar el estado de esa línea
- Líneas adicionales de control para inhibir la memoria principal
- Estados líneas de caché:
  - Válida (**V**): la línea de caché, es consistente con la copia de memoria, ha sido leída de la memoria principal y no ha sido modificada
  - Inválida (**I**): la línea no se encuentra en la caché o no es consistente con la copia en memoria
  - Reservada (**R**): los datos han sido escritos una única vez desde que se leyó de la memoria compartida, la línea de caché es consistente con la copia en memoria que es la única otra copia
  - Sucia (**S**): la línea de caché ha sido escrita más de una vez, y la copia de la caché es la única en el sistema (por lo tanto inconsistente con el resto de copias)
- Transiciones según operaciones sobre cachés: fallo de lectura, acierto de escritura, fallo de escritura, acierto de lectura y cambio de línea

Protocolos de coherencia

Protocolo de sondeo (snoopy) – **Write Once**

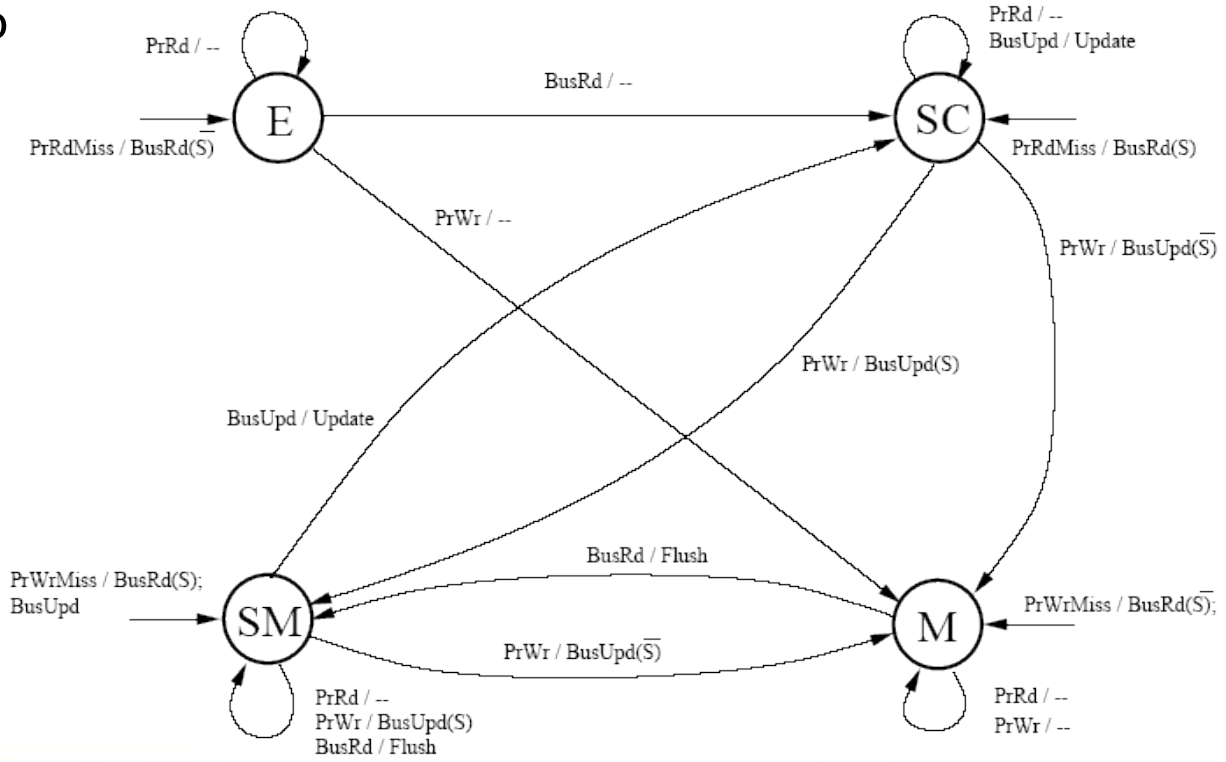
Diagrama de transiciones entre estados



Protocolos de coherencia

Protocolo de sondeo (snoopy) - **Dragon**

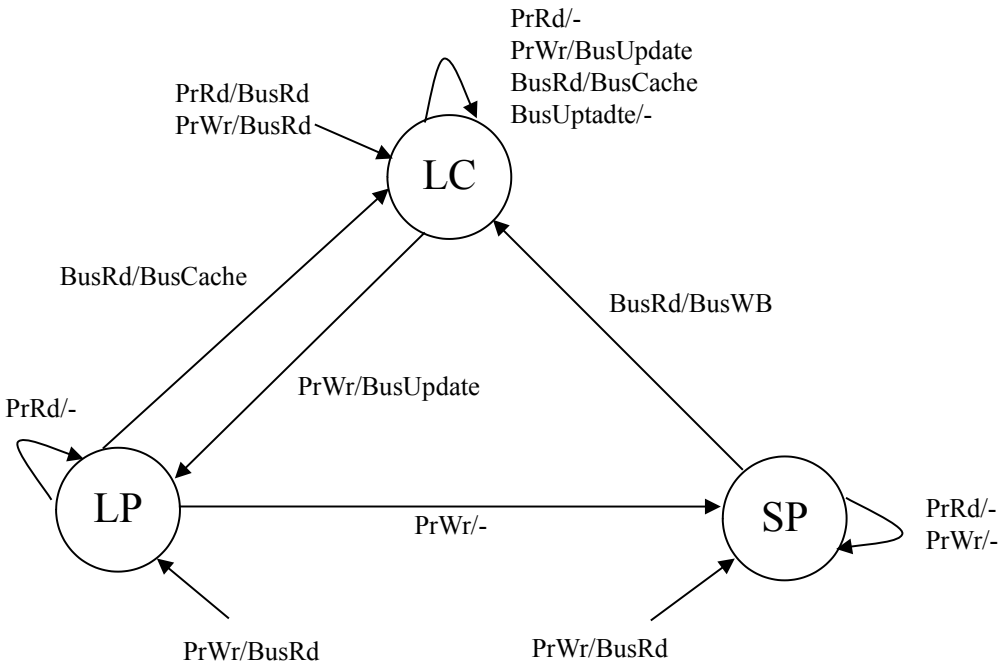
- Protocolo de actualización en escritura básico para cachés *write-back*
- Estados: Exclusivo (**E**), Compartido (**C**), Compartido_Modificado (**SM**) y Modificado (**M**)
- Transiciones según operaciones sobre cachés: fallo de lectura, escritura, reemplazo



Protocolos de coherencia

Protocolo de sondeo (*snoopy*) - **Firefly**

- Protocolo de actualización en escritura, que hace uso de una línea compartida especial de bus
- Estados de línea de caché: Lectura_Privada(**LP**) (Exclusive), Lectura_Compartida (**LC**) (Shared) y Sucia_Privada (**SP**) (Modified) (MESI)
- Transiciones según operaciones sobre cachés: fallo de lectura, escritura, reemplazo



Protocolos de coherencia

Protocolos de sondeo (snoopy): Rendimiento

- Consideraciones a tener en cuenta para evaluar el rendimiento de un protocolo:
  - Tráfico causado por fallos de caché
  - Tráfico de comunicación entre cachés
- Diferencias de rendimiento entre protocolos de invalidación y actualización:
  - Varias escrituras a la misma palabra sin lecturas intermedias
  - Líneas de caché de varias palabras: los de invalidación trabajan sobre el bloque y los de actualización sobre palabras para aumentar la eficiencia
  - Retraso entre escritura de palabra en un procesador y la lectura por parte de otro → mejor en los protocolos de actualización
- Rendimiento según los requisitos:
  - **Para aprovechar mejor el ancho de banda del bus y la memoria, se utilizan los protocolos de invalidación**
  - Migración de procesos o sincronización intensivas: cuando se requiera mucha migración de procesos o mucha sincronización, un protocolo de invalidación va a trabajar mucho mejor que uno de actualización

Protocolos de directorio (centralizado o distribuido)

- Sistemas multiprocesador con varias subredes locales de interconexión y muchos elementos conectados
- Directorio común que almacena el estado de las líneas de caché
- La red de interconexión no soporta *broadcast* o provoca mucha degradación
- El comando de invalidación/actualización se envía a aquellas caches que disponen de copia local de un bloque
- La entrada del directorio tiene un **bit de permiso de actualización**
- Existen **varias copias compartidas de la misma línea de caché** para mejorar el rendimiento sin incrementar el tráfico

Protocolos de directorio

Protocolos de **directorio centralizado**

- Tabla **centralizada** que almacena ubicación de cada copia de caché
- Tamaño grande → búsqueda asociativa
- Inconvenientes: competencia de acceso y tiempos largos de búsqueda
- Protocolos de mapeado completo o limitado

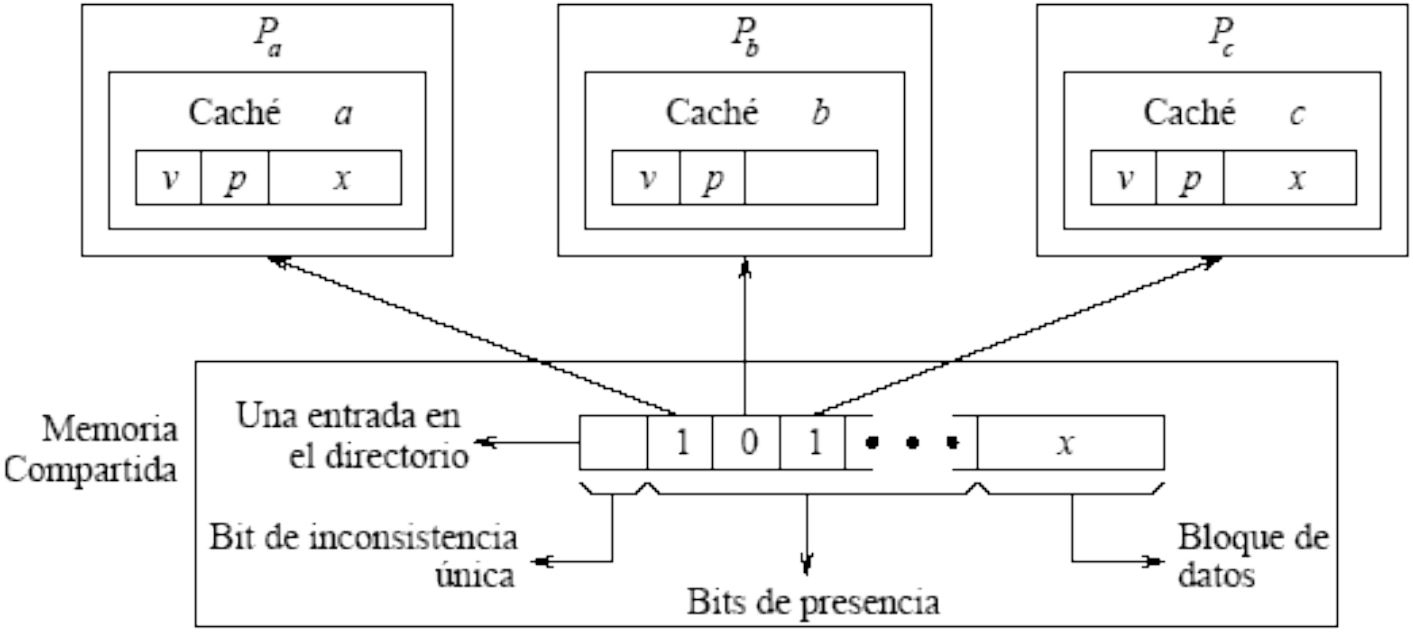
Protocolos de directorio

Protocolos de **directorio de mapeado completo**

- Protocolo $Dir_N\ NB$
- En el directorio: cada entrada de bloque tiene unos bits de presencia por cada caché y un bit de inconsistencia única
  - ✔ Bits de presencia: especifica la presencia en las cachés de copias del bloque de memoria
  - ✔ Bit de inconsistencia única: cuando este bit está activado, sólo uno de los bits presencia está a uno, es decir, sólo existe una caché con la copia de ese bloque o línea, con lo que sólo esa caché tiene permiso para actualizar la línea
- Por cada caché:
  - ✔ Bit de validación (**v**): indica si la copia es válida
  - ✔ Bit de privacidad (**p**): indica si la copia tiene permiso de escritura, es decir, cuando este bit es 1 entonces es la única copia que existe de esta línea en las cachés y por tanto tiene permiso para escribir

Protocolos de directorio

Protocolos de **directorio de mapeado completo**



Protocolos de directorio

Protocolos de **directorio de mapeado completo**

- Fallo de lectura:
  - Supongamos que una caché envía una petición de fallo de lectura a la memoria
  - Si el bit de inconsistencia simple está activado, la memoria envía una petición de actualización a la caché que tenga el bit de privado activo, o sea, a la única caché que tenga la línea
    - ✔ La caché devuelve el bloque a la memoria y desactiva su bit de privado
    - ✔ El bit de inconsistencia simple también es desactivado del directorio central
    - ✔ La memoria activa el bit de presencia de la caché correspondiente que solicitó el bloque y envía una copia a dicha caché
    - ✔ Cuando la caché recibe la copia, activa el bit de válido y desactiva el de privado
  - Si el bit de inconsistencia simple está desactivado, trae el bloque de la memoria activando el de válido y desactivando el de privado como en el caso anterior

Protocolos de directorio

Protocolos de **directorio de mapeado completo**

- Fallo de escritura:
  - Supongamos que una caché envía una petición de fallo de escritura a la memoria
  - La memoria envía peticiones de invalidación a todas las cachés que tienen copias de ese bloque poniendo a cero sus bits de presencia
  - Las cachés aludidas invalidan sus líneas poniendo a 0 el bit de línea válida
  - Si hay alguna caché con una copia del bloque con el bit de privado activado, la memoria actualiza su copia
  - Una vez la memoria recibe todos los reconocimientos, activa el bit de presencia en la caché y manda la copia dicha caché
  - El bit de inconsistencia simple se pone a 1 puesto que solo hay una copia
  - La caché que recibe la copia, se modifica y pone los bit de válido y privado a uno

Protocolos de directorio

Protocolos de **directorio de mapeado completo**

- Acierto de escritura:
  - Si el bit de privado es 0, la caché envía una petición de privacidad a la memoria
    - ✔ La memoria invalida todas las demás cachés que tienen una copia del bloque
    - ✔ Se pone el bit de inconsistencia a 1
    - ✔ Cuando la caché recibe el reconocimiento de que las demás cachés han sido invalidadas, modifica su bloque pone el bit de privado a uno
  - Si el bit de privado es 1, entonces escribe sin más puesto que es la única caché que tiene copia de esa línea

Protocolos de directorio

Protocolos de **directorio de mapeado completo**

- Principales problemas:
  - ✔ Tamaño_directorio = (1 + Núm_cachés) · Núm_líneas_memoria
  - ✔ Crecimiento del directorio proporcional a N^2
  - ✔ Para evitar el crecimiento cuadrático del mapeado completo es posible restringir el número de copias de caché activas de forma simultánea de un bloque. De esta forma se limita el crecimiento del directorio hacia un valor constante

Protocolos de directorio

Protocolos de **directorio de mapeado limitado**

- $Dir_i NB$, $i < N$
- Idéntico al protocolo de mapeado completo pero reduciendo el número de copias de caché activas simultáneas de un bloque:
  - ✔ Cada entrada de bloque tiene un **bit de presencia** en cada caché y un **bit de inconsistencia única (BIU)**
  - ✔ Cada caché tiene un bit de validación (**v**) y un bit de privacidad (**p**)
- Punteros limitados
  - ✔ Llamada de desalojo: la memoria invalida una de las cachés que utilizaba el puntero y le asigna ese puntero libre a la nueva caché
  - ✔ Tamaño punteros: $\log_2 N$ bits
- Tamaño del directorio = (1 + Punteros · $\log_2 N$) · Bloques_memoria
- Crecimiento proporcional a $N \cdot \text{Log}_2 N$
- Protocolo “escalable”

Protocolos de directorio

Protocolos de **directorio de distribuido**

- Tabla distribuida entre cachés
- Cada tabla local almacena el estado de la caché y de las copias de los bloques
- Protocolos de mapeado distribuidos de directorio:
  - Encadenados: listas de cachés encadenadas
  - Jerárquicos: división del directorio entre grupos de procesadores interconectados (*clusters*)

Protocolos de directorio

Protocolos de directorio de distribuido encadenado

Lista simple o doble enlazada de punteros entre distintas cachés con copia de un mismo bloque:

- Cada entrada del directorio apunta a una caché con copia del bloque y esta caché a su vez tiene un puntero que apunta a otra, etc.
- Una entrada en el directorio contiene un único puntero que apunta a la cabeza de la lista