Tema 3: Multiprocesadores
Parte 1: Coherencia SMP

Universidad de Cádiz

Introducción

Memorias rápidas → Muuuy caras (Registros)
Memorias grandes → Muuuy lentas (HDD)

Solución:
Jerarquías de memoria.

Se estructura en varios niveles

Cada nivel inferior (más cercano al procesador)
+ caro + rápido – capacidad

Un nivel inferior contiene subconjuntos de datos del nivel inmediatamente superior.

Introducción

Jerarquías de memoria.

Su funcionamiento se basa en que los accesos a memoria no son aleatorios.

Se ‘predicen’ que partes de memoria se van a usar (bloques, registros) y se **copian** a niveles inferiores.

Trabajar con copias genera grandes problemas..
…¿la copia que estoy usando está actualizada?
La respuesta a esa pregunta tendría que ser siempre SI

Introducción

Jerarquías de memoria.

Hay que asegurar que todas las copias accesibles a lectura mantengan la ultima información actualizada, es decir que todas sean iguales.

Es decir, que sean **coherentes**.

No asegurar la coherencia no es aceptable en el diseño de una arquitectura, bien de hardware, bien de software.

Introducción

Toda la memoria ha de ser coherente, desde lo que hay en el nivel mas inferior de la jerarquía hasta los niveles superiores (HDD, Discos…)

En este tema hablaremos de la **coherencia de cache** y de las técnicas usadas para asegurarla

Pese al nombre, la coherencia de caché suele referirse a la coherencia entre MP y memorias cachés

Introducción
Coherencia en monoprocesadores

Actualmente se trabaja con varios niveles de caché; el mismo dato puede estar copiado en niveles de caché además de en MP.

2 técnicas para actualizar las copias;

- **WT** (***Write-through***). Se actualizan todas las copias.
- **WB** (***Write-back***). Se permite la no coherencia (mantener copias con distinto valor) mientras no sea necesario.

Introducción
Coherencia en monoprocesadores

Write-back

Las copias se actualizan en memoria principal cuando hay que eliminar el bloque de la caché.

Es necesario usar dos bits para gestionar este método:

- ***Valid*.** Información útil (o no)
- ***Dirty*.** Dato actualizado (o no)

Este sistema reduce los accesos a memoria y es el más usado en los monoprocesadores.

Introducción
Coherencia en monoprocesadores

Algunas operaciones pueden cambiar de forma no convencional las copias.

Por ejemplo DMA.

Actualiza directamente nivel superior.

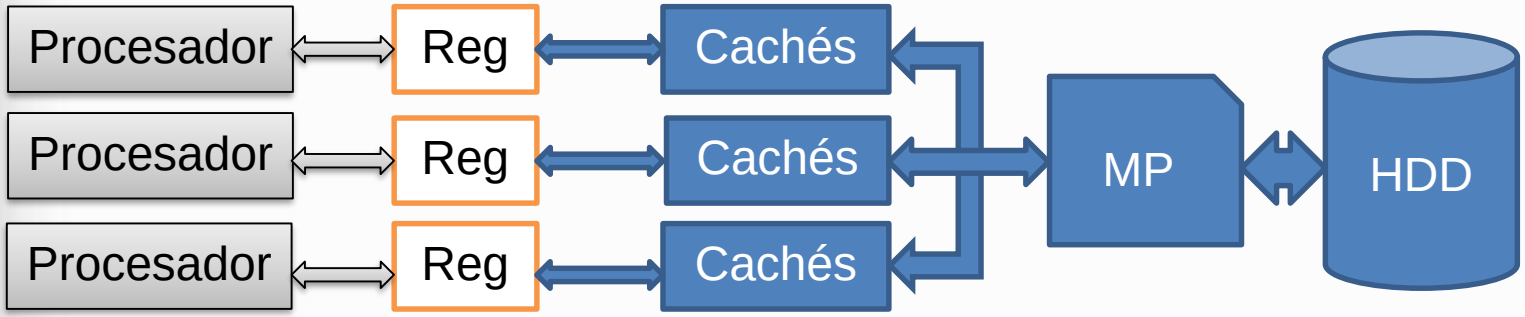
Soluciones:

- No “*cachear*” bloques de E/S
- Limpiar (*flush*) la cache

Introducción
Coherencia en SMP

En multiprocesadores la cosa se complica. Funciona en base a **variables compartidas**

- Puede existir copias distribuidas en distintos procesadores cada uno con sus niveles de caché.



- Las copias pueden depender o ser actualizadas por más de un procesador.

Introducción
Coherencia en SMP

¿Cómo sabe un procesador si una variable de su memoria caché ha sido modificada por otro procesador?

Posibles soluciones:

- No '*cachear*' variables compartidas.

  ¿Sería viable?

- Usar protocolos de coherencia.

Introducción
Conceptos y definiciones

Los procesadores pueden usar variables privadas que solo puede usar ese procesador.

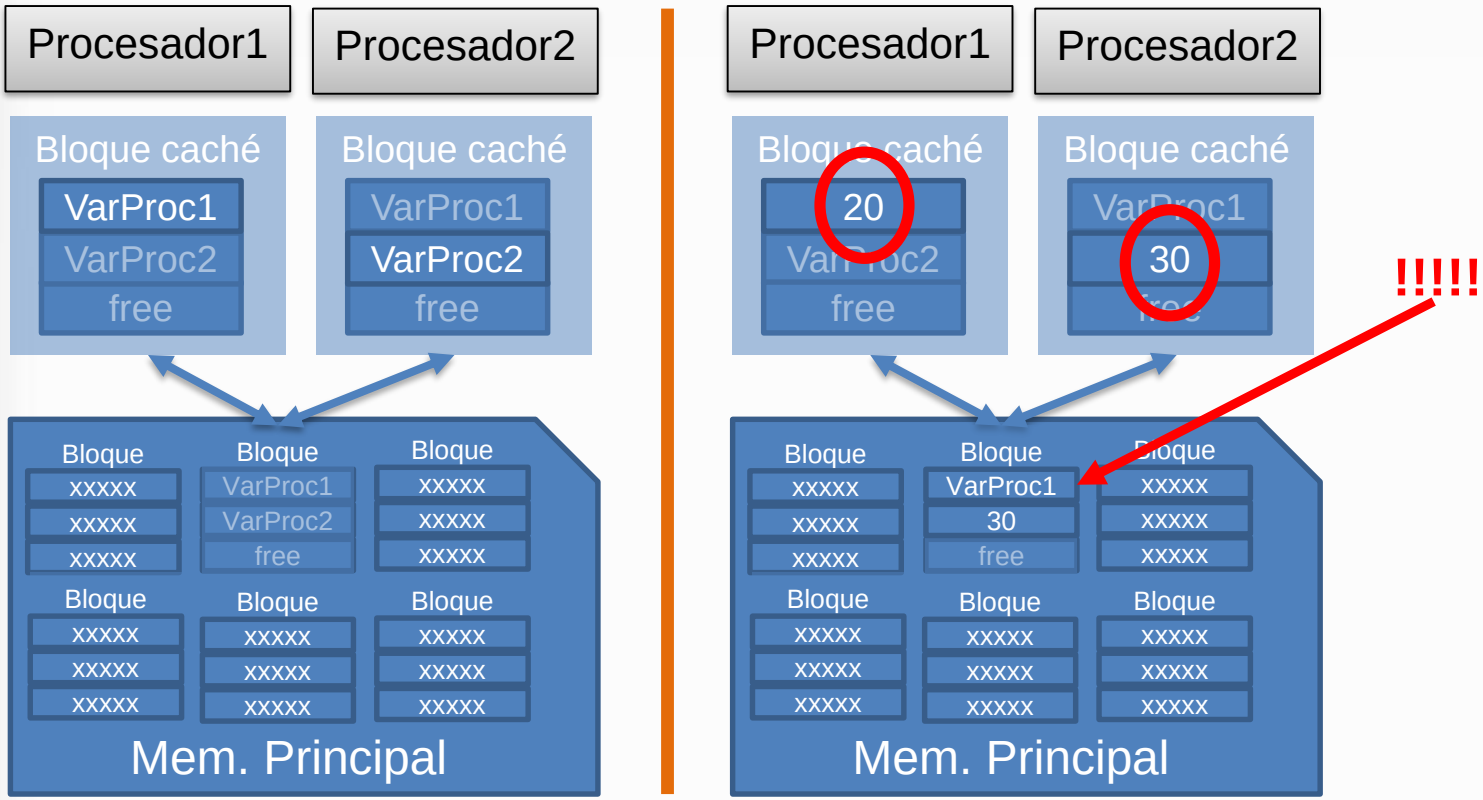
Una var. privada puede estar en un bloque con otras variables que usan otros procesadores.

Se genera entonces una **falsa compartición**.

Los protocolos de coherencia no gestionan variables privadas por lo que pueden dar conflictos al actualizar un bloque de cache

Introducción
Conceptos y definiciones

Falsa compartición.



Un procesador, al gravar todo el bloque, hace que se pierdan los datos de los anteriores.

Introducción
Conceptos y definiciones

Falsa compartición.

Caso: Dos procesadores con variables privadas en el mismo bloque intentan actualizar sus variables.

¿Solución?

- Distribución adecuada de los datos en memoria (ubicando variables de distintos procesadores en bloques distintos).

- Conveniente que los bloques de memoria sean algo mas pequeños.

Introducción
Conceptos y definiciones

- El sistema de memoria tiene que ser **coherente**: todos los procesos tienen que utilizar la **misma información y actualizada**.

- Se dice que un sistema es **coherente** si:
  - Al leer una variable, se obtiene el **último valor escrito** en esa variable (si ha transcurrido suficiente tiempo desde la escritura).
  - Todas las escrituras sobre una variable se "ven" en el **mismo orden** en todos los procesadores.

Introducción
Conceptos y definiciones

- Multiprocesadores (SMP)
  - Memoria compartida
  - Pocos procesadores (de 2 a 16) conectados a un BUS
    - Protocolos tipo **Snoopy**
- Muticomputadores
  - Memoria distribuida (DSM)
  - Muchos procesadores conectados a una RED
    - Protocolos basados en **directorios**

Protocolo Snoopy

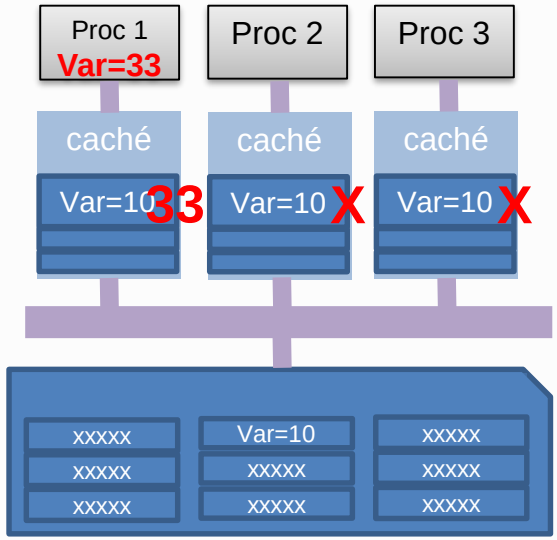
Snoopy (entrometido/fisgón)

- La coherencia se realizará por hardware.
- La memoria se conecta a los procesadores por un BUS.
  - Todas las operaciones son publicas y cualquier procesador puede ver lo que otros hacen.
- Snoopy usa ese concepto ‘espiando’ o ‘fisgando’ las operaciones de los otros procesadores.
- Dependiendo de lo que encuentre ‘fisgando’ decidirá que hacer en las cachés locales **envia**ndo **señales de control** especiales a los procesadores.

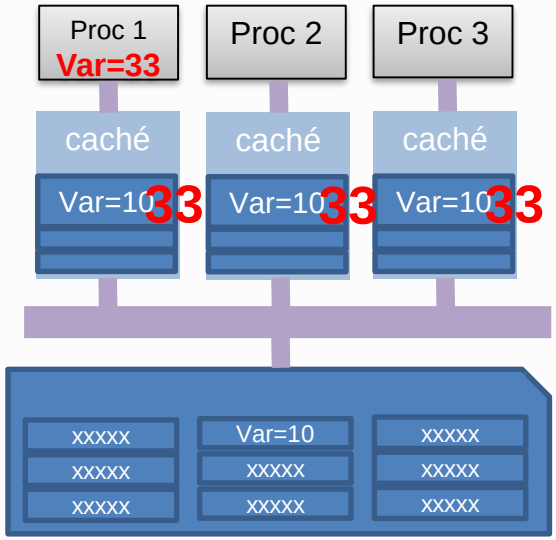
Protocolo Snoopy

¿Qué hacer al detectar una actualización de un dato que está copiado?

- Invalidar todas las copias (**protocolos de invalidación**)



- Actualizar todas las copias (**protocolos de actualización**)



La memoria se actualizará o no dependiendo de la política de escritura utilizada (WT/WB)(Write-through/Write-Back)

Protocolo Snoopy

Acciones Snoopy (Para gestionar los estados y generar señales de control)

Frente al BUS

BR *(Bus Read)* Solicitud de un bloque actualizado.
Todos han de actualizar (si lo tuviesen).

INV *(Invalidate)* Señal para invalidar un bloque concreto.
Todos han de responder. (Prot. de Invalidación)

BC *(Broadcast)* Orden para actualizar todas las copias.
Han de responder confirmando. (Prot. de actualización)

Otras

BW *(Bus Write)* Escritura de un bloque en **MP**.

Protocolo Snoopy

Acciones Snoopy (Para gestionar los estados y generar señales de control)

Frente al procesador Local

PR *(Processor Read)* El procesador lee una variable en caché, si acierto perfecto, si fallo ha de leer bloque con **BR**.

PW *(Processor Write)* Procesador escribe una variable en caché. Hay que avisar para actualizar o bien con un **INV** o bien con un **BC**. Si antes de actualizar encontramos fallo en caché habría que hacer un **BR**.

(Entendemos por variable una palabra, no un bloque)

Estados y señales de control

- Para gestionar la coherencia se añaden bits de control al bloque.
- Estos bits permiten marcar cada bloque con 5 posibles estados:

I	Invalido
E	Exclusivo: **una** copia y **coherente** con MP
M	Modificado: **una** copia y **NO coherente** con MP
S	Compartido: **varias** copias y **todas coherentes**
O	Propietario: **varias** copias (en S) y **una coherente** (en O) que puede no ser coherente con MP

Estados y señales de control

Son necesarios tres bits para identificar estos estados.
(valid – dirty – shared)

	Valid	Dirty	Shared
(Invalido) **I**	0	-	-
(Exclusivo) **E**	1	0	0
(Modificado) **M**	1	1	0
(Compartido) **S**	1	0	1
(Propietario) **O**	1	1	1

Estados y señales de control

En función de los

- Estados (Valid, Dirty, Shares)
- Política de escritura (Write through/back)
- Estrategia para gestión de copias,
- Jerarquías
- etc.

….Se obtienen distintos protocolos de tipo snoopy.

Los protocolos quedan definidos con un autómata finito donde partiendo de un estado queda definido el siguiente estado tras una señal de control

Estados y señales de control

- Los protocolos quedan definidos con un **autómata finito** donde se detalla las ***transiciones*** y las ***señales de control*** que producen al recibir un determinado estado una determinada señal de control

- También podemos definirlos a través de un **diagrama de grafos** que represente estas transiciones

Protocolos de invalidación

Al realizar una actualización de una variable compartida ….
Se invalidan todas las copias distribuidas

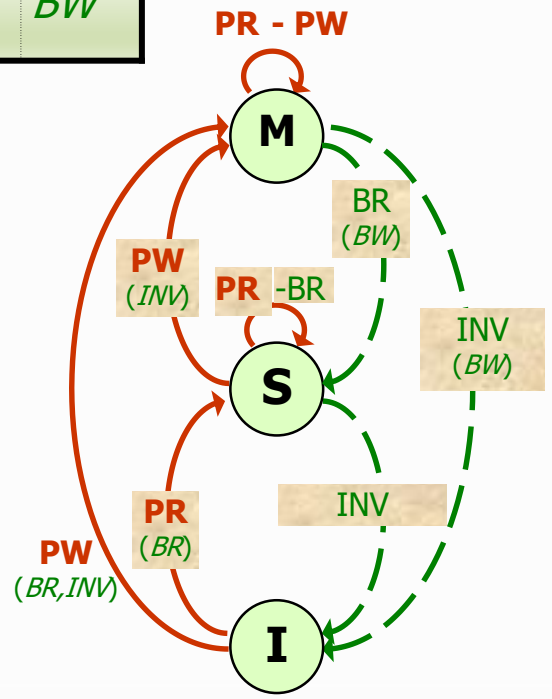
Existen varios protocolos:

- Los más simples tienen 2 estados y usan políticas de escritura WT *(Write-through)*

- Nosotros veremos protocolos de al menos 3 estados y con políticas de escrituras WB *(Write-back)*

Protocolo de tres estados MSI

		PR		PW		BR		INV	
fallo	I, -	S	*BR*	M	*BR,INV*				
acierto.	S	S		M	*INV*	S		I	
	M	M		M		S	*BW*	I	*BW*



El ancho de banda del BUS puede ser un cuello de botella importante en el sistema.

El protocolo ha de reducir el trafico.

Se puede ahorrar tiempo de transferencia cargando un bloque de caché desde otra caché (en vez de desde MP)

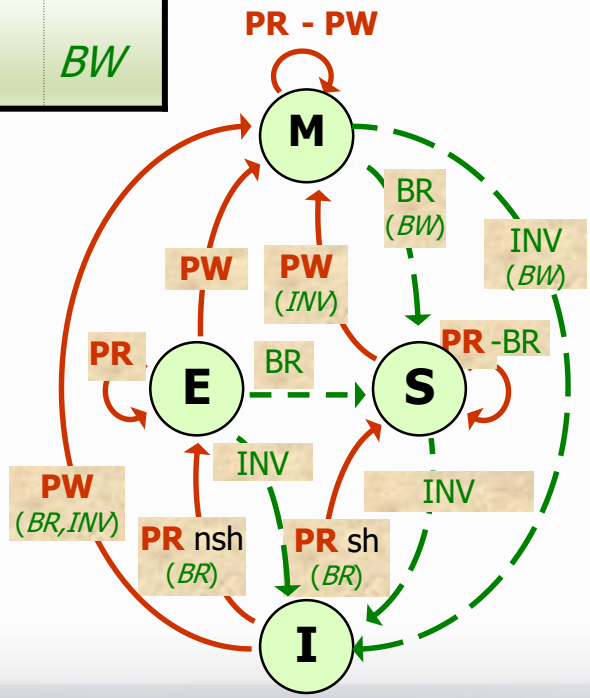
- Invalida (temp) el acceso del procesador origen con su mem. caché
- No descarga trafico del bus.
- Si origen está en M, habría que descargar en MP (si no lo hiciésemos tendríamos dos copias con M)

Protocolo MESI *(Illinois)*

- Mejora del MSI para minimizar las invalidaciones y reducir el trafico del bus
- Se introduce un **estado E** (Exclusive) que garantiza **solo una copia** del bloque.
  (Nos permite ahorrarnos trafico al modificarlo)
- Para distinguir entre E y S se introduce una **señal de control *sh****(share)-nsh*
  Al hacer un BR si sh es 0 se marca el bloque con E.
  (Al recibir un BR si un procesador tiene el bloque envía un sh=1)

Protocolo MESI *(Illinois)*

		PR		PW		BR		INV	
fallo	I, -	nsh: E sh: S	*BR*	M	*BR,INV*				
acierto	E	E		M		S		I	
	S	S		M	*INV*	S		I	
	M	M		M		S	*BW*	I	*BW*

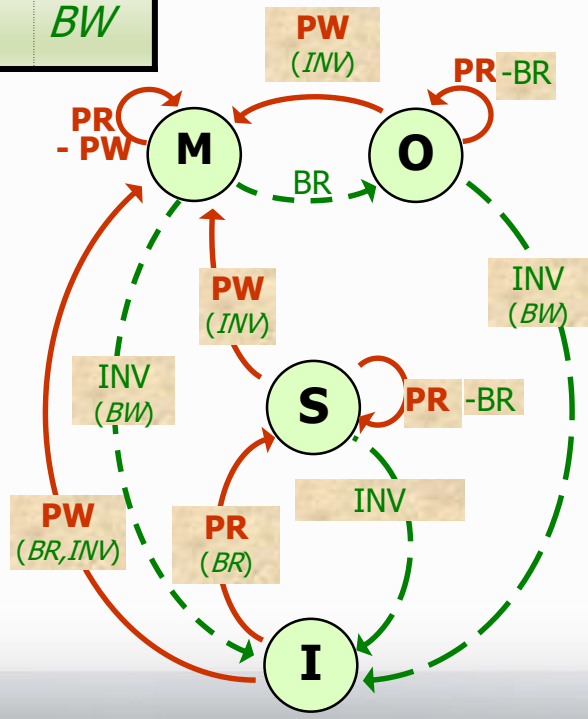


Protocolo MOSI *(Berkeley)*

- Se introduce un **estado O** (*Owner*) que permite tener múltiples copias no coherentes con memoria pero coherentes entre si.
- Solo una copia tiene estado O, el resto S
- Conseguimos ahorrar el tiempo de actualización de MP hasta que sea necesario (cuando un bloque sea invalidado o reemplazado)

Protocolo MOSI *(Berkeley)*

		PR		PW		BR		INV	
fallo	I, -	S	*BR*	M	*BR,INV*				
acierto	S	S		M	*INV*	S		I	
	M	M		M		O		I	*BW*
	O	O		M	*INV*	O		I	*BW*

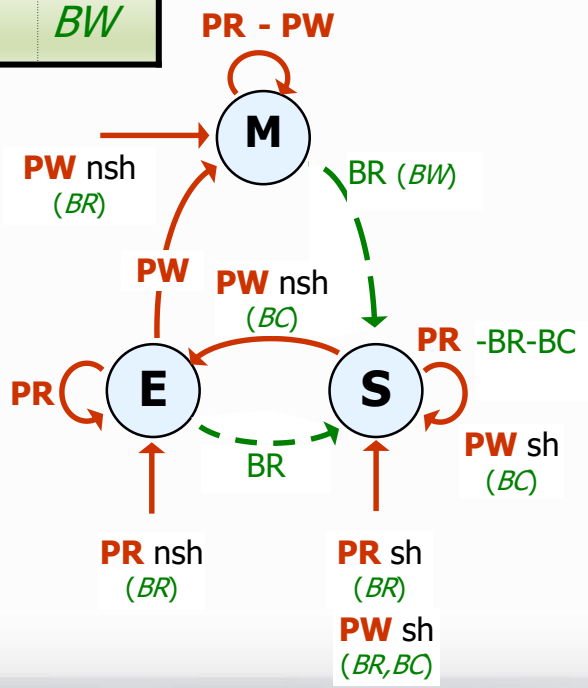


Protocolos de actualización

- En los protocolos de actualización todas las copias se mantienen actualizadas.
  (snoopy informará a las caches del la actualización correspondiente)
- En este protocolo usamos la señal ***broadcast (BC)***.
- Mantener actualizando todos los datos consume más ancho del BUS, por lo que no siempre es adecuado usarlos.
- Mientras el BUS actualiza una cache el procesador no puede usarla.

Protocolo MSE *(Firefly)*

fallo \ acierto	PR	PW	BR	BC
-	nsh: **E** sh: **S** *BR*	nsh: **M** sh: **S** *BR* *BR,BC*		
E	**E**	**M**	**S**	**S**
S	**S**	nsh: **E** sh: **S** *BC*	**S**	**S**
M	**M**	**M**	**S** *BW*	**S** *BW*



- Incorpora la política de escritura **WT** *(WriteThrough)* cuando hay varias copias.
- Usa la línea de control **SH** *(Shared)*.

Protocolo MOES *(Dragon)*

fallo \ acierto	PR	PW	BR	BC
-	nsh: **E** sh: **S** *BR*	nsh: **M** *BR* sh: **O** *BR,BC*		
E	**E**	**M**	**S**	**S**
S	**S**	nsh: **M** - sh: **O** *BC*	**S**	**S**
M	**M**	**M**	**O**	**S**
O	**O**	nsh: **M** - sh: **O** *BC*	**O**	**S**

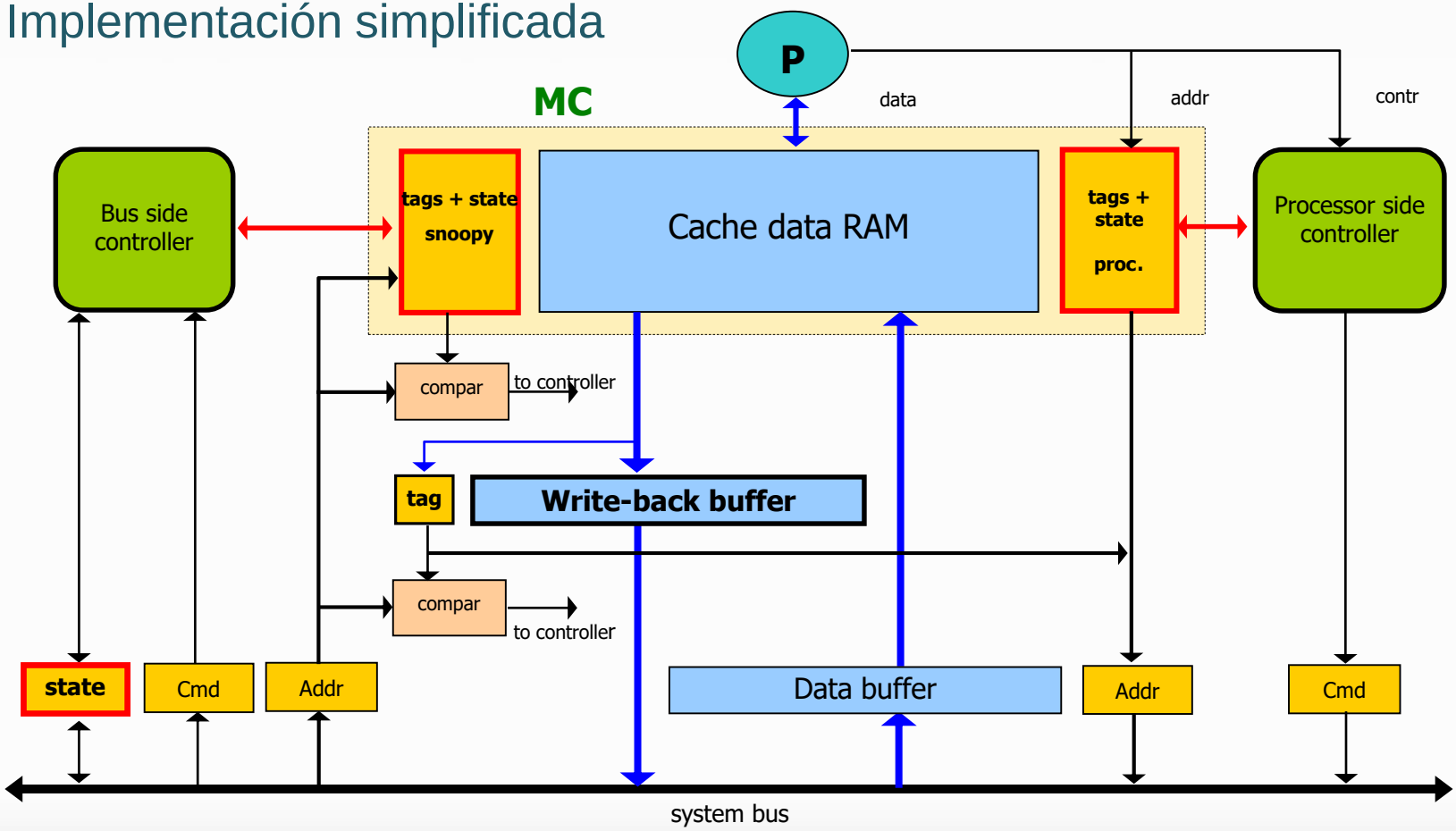
- Prácticamente igual que **MSE** pero con el estado O.
- Volvemos a la política de escritura **WB** (Gracias al estado **O**)
- Usa la línea de control **SH** *(Shared).*

Implementación Snoopy

- El protocolo Snoopy es suficiente para aplicarse a los multiprocesadores.
- Tiene una lógica sencilla pero con una implementación complicada.
- La implementación genera nuevos problemas que no se puede permitir un protocolo de coherencia.
  (Un protocolo de coherencia debería ser 100% fiable)
- Los problemas están asociados a su:
  - Distribución (Autómata distribuido en P procesadores)
  - NO Atomicidad

Implementación Snoopy

Ejemplo de Implementación simplificada



Problemas y soluciones

Directorio de la memoria cache

- Surge al intentar acceder tanto el procesador como el bus a la memoria caché
- Para evitar conflictos se obliga a uno de los dos a espere a que el otro acabe.
  - Esto ralentiza todo el sistema.
- Normalmente se soluciona duplicando el directorio de la caché
  - Esto no implica eliminar conflictos pero los reduce notablemente.
  - Los dos directorios han de ser coherentes.

Problemas y soluciones

Bufferes BW

- Se generan tiempos de espera al acceder a memoria en ciertas secuencias de operaciones comunes. (Imagina un remplazo de un bloque en M)

- Lo común es actualizar (BW) y leer (BR).

  Pero MP es muy lenta

Una opción es usar un buffer BW siendo la operación:

Copiar a Buffer / BR/ BW

❖ Más complicado pues ahora para comprobar si un bloque está en caché hay que mirar también en el buffer. Duplicamos así controladora y comparadores

Problemas y soluciones

Peticiones

- En una petición BR ¿Quién proporciona el bloque?
  - MP lo tiene siempre pero es muy lenta.
  - Caché es más rápida pero implica complejidad.

Estrategias para solicitar bloque a otras caches:

- Esperar tiempo máximo de tiempo, dando tiempo a que todas las chaches contesten. Tiempo siempre fijo y grande.
- Esperar un tiempo variable (*handshake*), hasta que todos hayan contestado. Tiempo mínimo pero complicado de implementar.
- Añadir un bit a todos los bloques de MP para saber si están cacheados. Solución muy compleja.

Problemas y soluciones

Peticiones

- Se optimiza solicitando a MP el bloque:
  - Si cache proporciona antes se cancela la lectura a MP.
  - Si MP acaba antes se espera hasta que cache confirmen que no hay ningún M.
- Para poder usar estas estrategias hay que usar tres líneas de control:
  - ***sh*** *(Shared)*. Indica si el bloque está en alguna caché.
  - ***dirty***. Indica si el bloque ha sido modificado.
  - ***Inh*** *(inhibir)*. Para abortar lectura de MP.

Problemas y soluciones

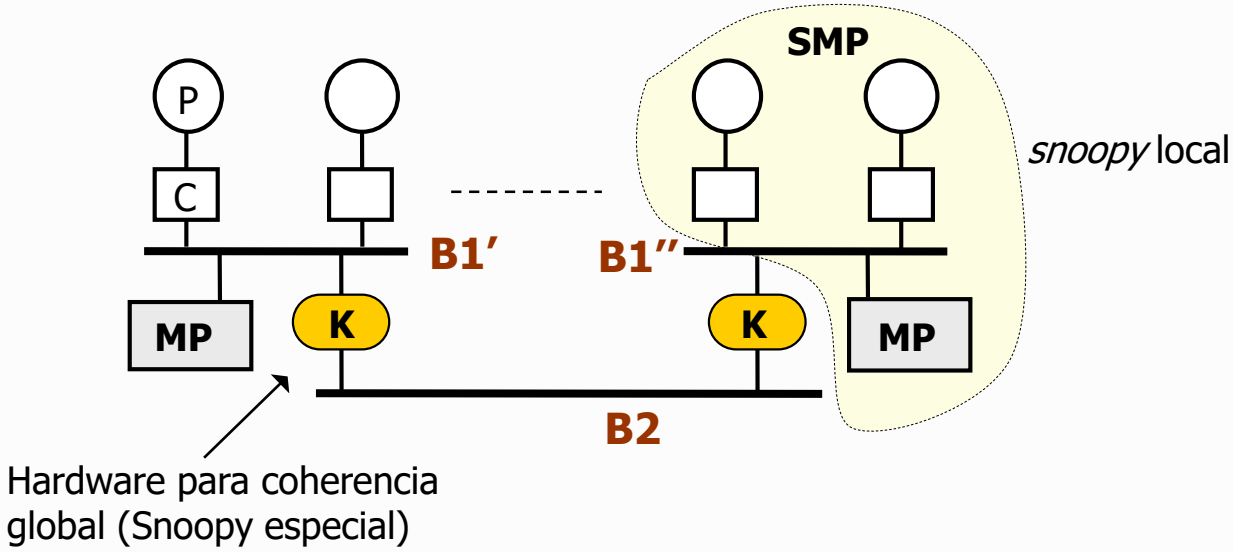
Atomicidad

- Una operación es **atómica** si ejecuta toda la operación sin ninguna interferencia.
- Los autómatas de los protocolos no son por definición atómicos y pueden solaparse operaciones generando grandes conflictos mezclando acciones de distintos procesadores sobre el mismo bloque.
- Es labor del protocolo asegurar la atomicidad.

Snoopy jerárquico

No se pueden conectar muchos procesadores a un BUS.

Para solucionarlo se pueden jerarquizar los BUSES.



La memoria aún compartida esta distribuida físicamente.
(Por lo que tendríamos un sistema NUMA)