Arquitectura e Ingeniería de Computadores

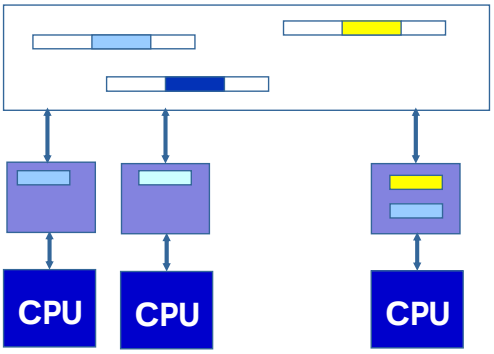
Modulo III. Multiprocesadores

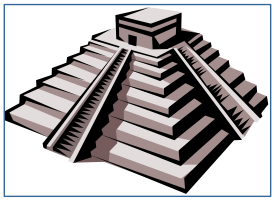
Tema 8. Coherencia. Consistencia. Sincronización.

Multiprocesadores de memoria compartida

- Esquema
  - Jerarquía de memoria extendida.
  - Alternativas de implementación.
  - Modelo intuitivo de memoria.
  - Problema de la coherencia de memoria
  - Soluciones de grano grueso, uniprocesadores.
  - Sistema de memoria coherente en multiprocesadores
  - Protocolos para implementar coherencia
    - Protocolo Snnopy

Jerarquía de memoria extendida



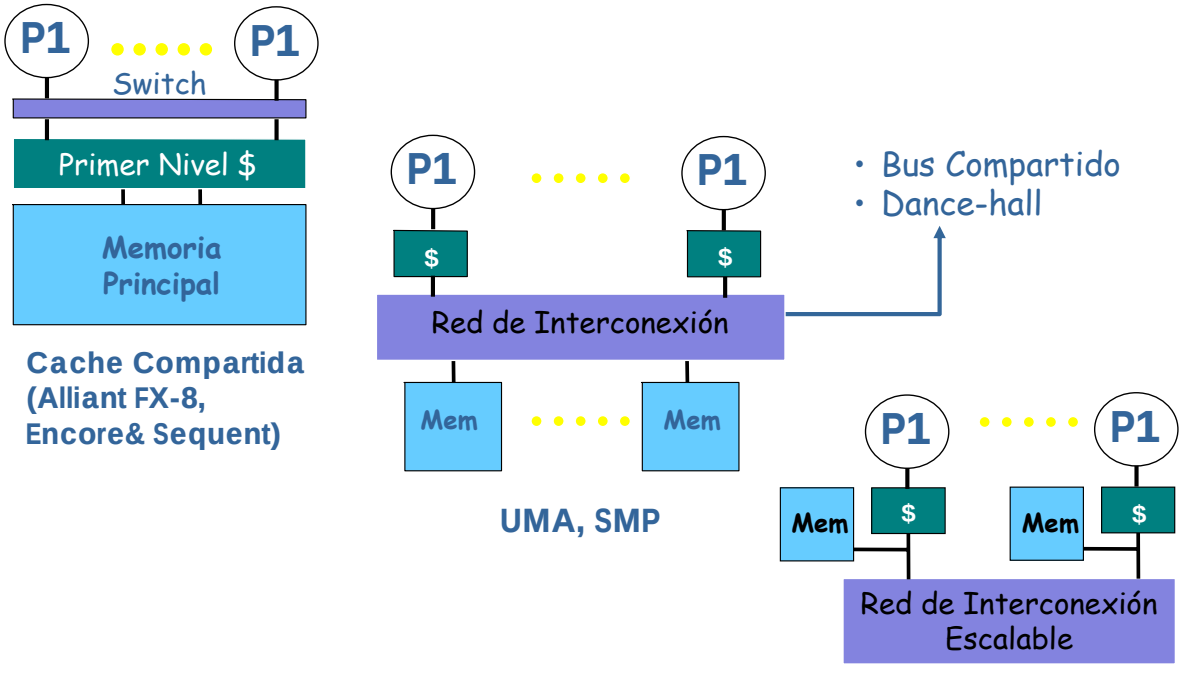


Jerarquía de Memoria

- Ocultar la latencia de los accesos
- Reducir la contención

Alternativas de implementación (1)

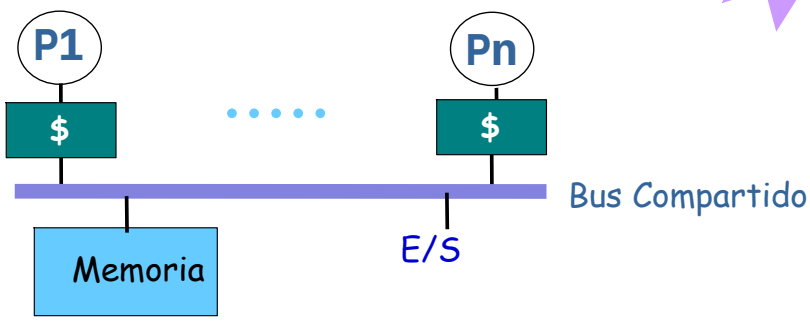
Escalabilidad



Alternativas de implementación (2)

SMP basado en Bus Compartido

Mayor éxito



- **Servidores departamentales**
- **Estaciones de trabajo**
- **Bloques de construcción básicos sistemas de gran escala**
- **Soporte en microprocesadores de propósito general**

Modelo intuitivo de memoria (1)

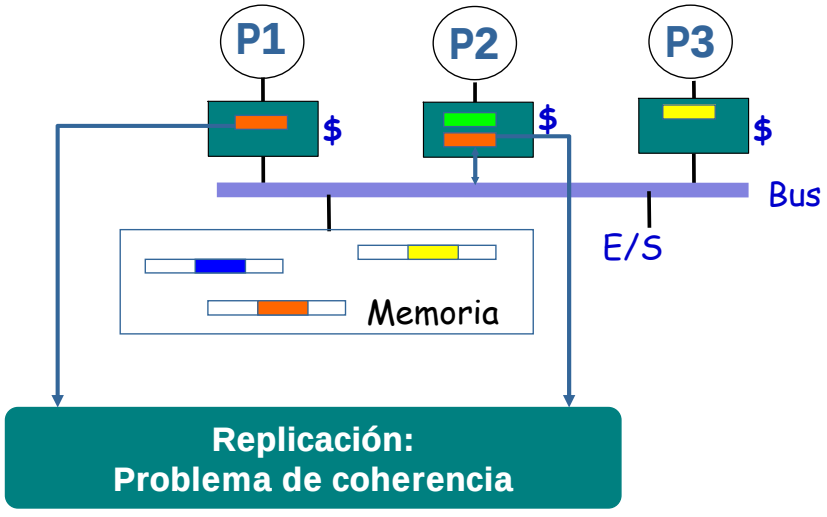
- Ejecución secuencial
  - La memoria se utiliza para “comunicar valores”
  - Cuando se lee una posición se devuelve el último valor
- Multiprogramación (uniprocesador)
  - La lectura de una posición devuelve el último valor escrito en ella
    - Independientemente del proceso (thread) que lo hizo.
  - Las caches no interfieren con el uso de múltiples procesos (threads) en un procesador
    - Todos ellos ven la memoria a través de la misma jerarquía

Modelo intuitivo de memoria (2)

- Multiprocesador **de memoria compartida**
  - Objetivo: Que el resultado de un programa sea el mismo independientemente de cómo se ejecuten los procesos
    - En paralelo
    - De forma entrelazada (multiprogramación)

Multiprocesador de memoria compartida (1)

- **Problema**:
  - Cuando dos o más procesos acceden a la memoria a través de caches diferentes existe el peligro de que vean valores inconsistentes.

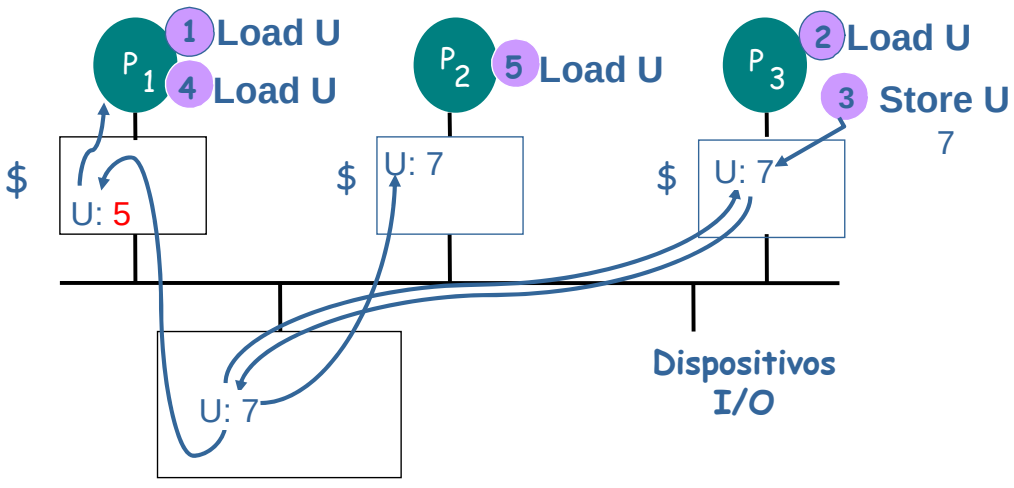


Multiprocesador de memoria compartida (3)

- Requisitos necesarios:
  - **Mantenimiento de la Coherencia**
    - Sirve para evitar que las copias de una misma dirección tengan contenidos distintos
  - **Modelo de Consistencia**
    - Sirve para garantizar que todos los componentes del sistema vean el mismo orden en los accesos a cualquier dirección
  - **Mecanismo de Sincronización**
    - Sirve para garantizar que los accesos de escritura y lectura en la misma dirección se ejecuten el orden necesario

Problema de la coherencia cache

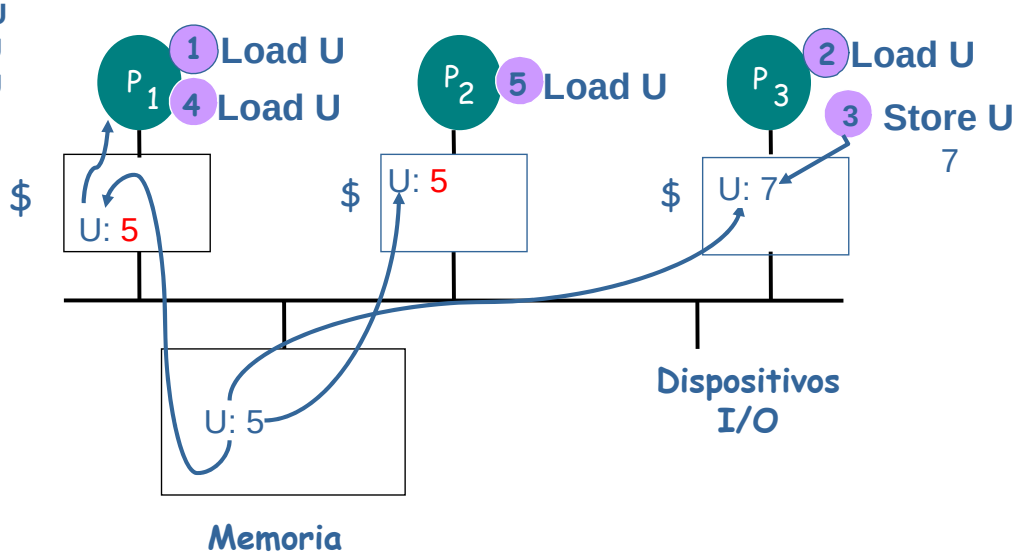
(P1) Load U
(P3) Load U
(P3) Store U
(P1) Load U
(P2) Load U



Politica reemplazamiento:
Escritura directa

Problema de la coherencia cache

(P1) Load U
(P3) Load U
(P3) Store U
(P1) Load U
(P2) Load U

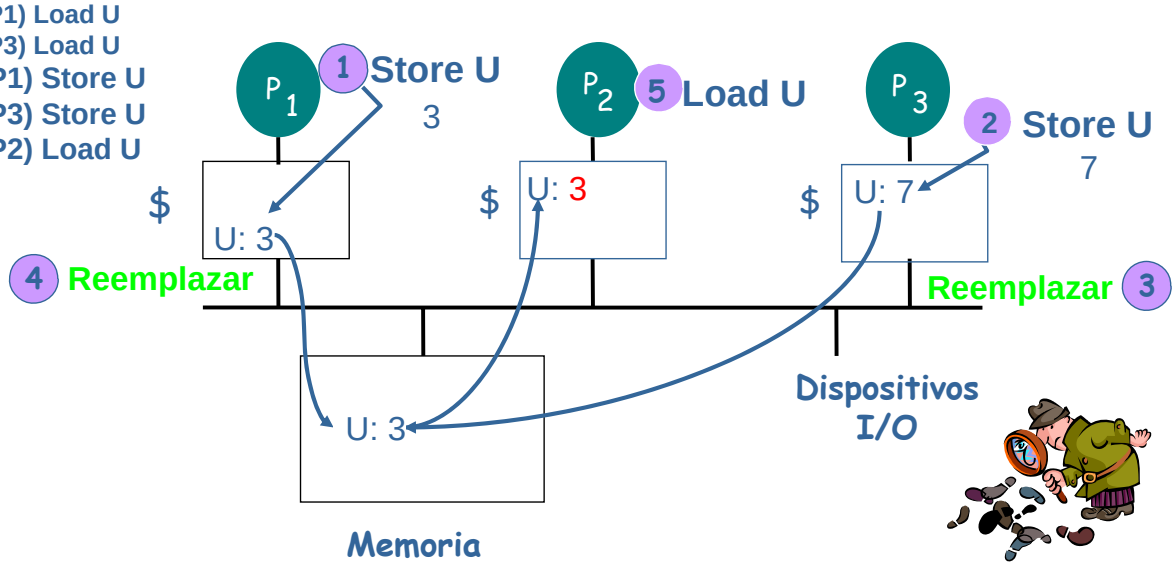


Politica reemplazamiento:

Post-Escritura

Problema de la coherencia cache

(P1) Load U
(P3) Load U
(P1) Store U
(P3) Store U
(P2) Load U

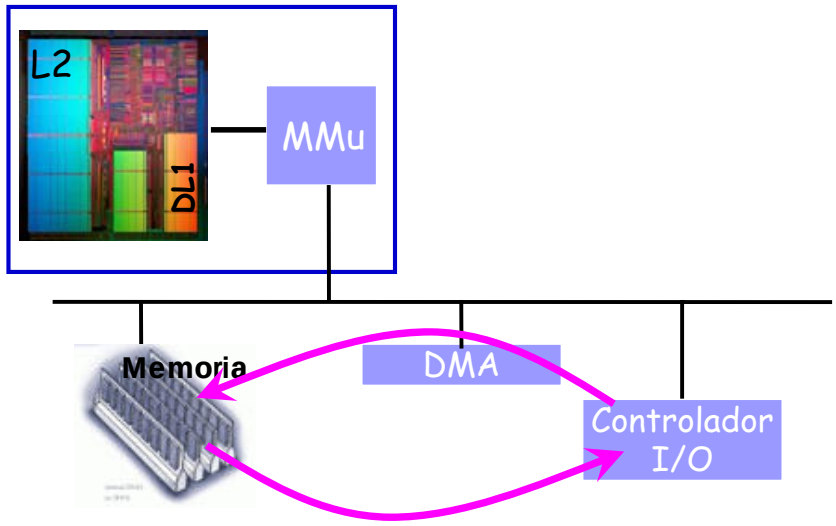


Protocolos de coherencia de cache

Siguen la pista del estado de cualquier bloque de datos compartido para evitar incoherencias

Soluciones de Grano Grueso (1)

- Problemas de coherencia en uniprocesadores
  - **Operaciones de E/S a través de dispositivos DMA:**
    - Salida de dato obsoleto (cache post-escritura)
    - Entrada de dato que no tenga efecto (acierto de cache)



Soluciones de Grano Grueso (2)

- Soluciones uniprocesador
  - Evitar usar la cache
    - Los segmentos de memoria involucrados en operaciones de E/S se marcan como *No-Cacheables*
  - Sacar de la cache antes de E/S (SO)
    - Las páginas de memoria involucradas en cualquier operación de E/S son eliminadas de la cache previamente (flush).
      - Interviene el SO
  - Usar la cache para E/S
    - El Tráfico de E/S pasa por todos los niveles de la jerarquía de memoria
      - Contamina la cache con datos que pueden no ser de interés inmediato para el procesador

Soluciones de Grano Grueso (3)

- En multiprocesadores
  - **Aparece problema de coherencia entre caches**
  - La escritura o lectura de variables compartidas es un evento frecuente.
  - No es práctico:
    - Deshabilitar la cache para datos compartidos
    - Invocar al SO en cada referencia a una variable compartida
  - Solución hardware y no software

Proveer coherencia de memoria

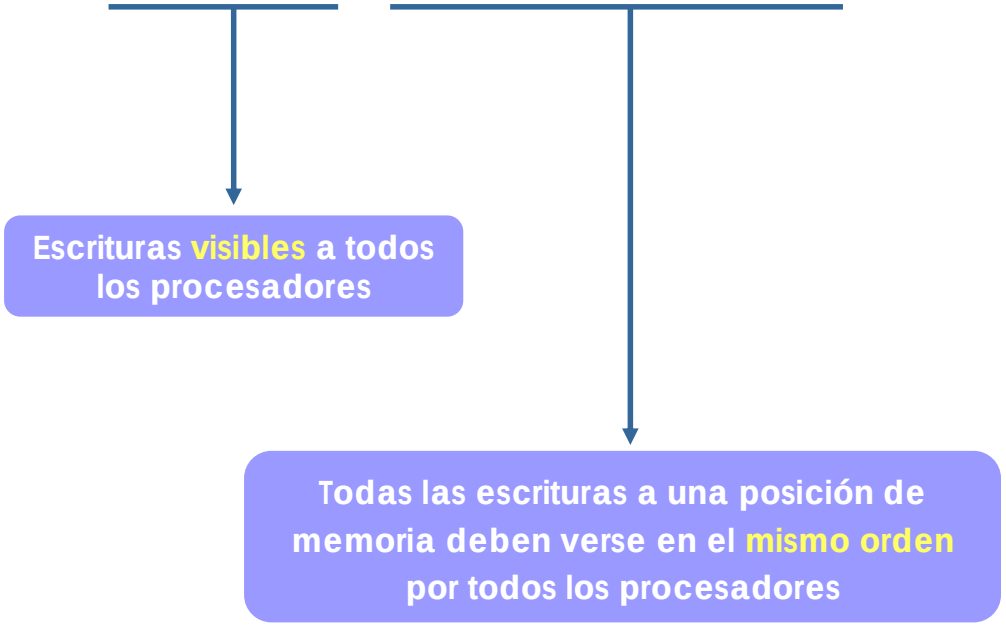
Sistema de Memoria Coherente (1)

¿Qué significa que un sistema de memoria es coherente?

Una operación de lectura retorna siempre el último valor que fue escrito en la posición de memoria correspondiente, independientemente del procesador que efectúa la lectura o escritura

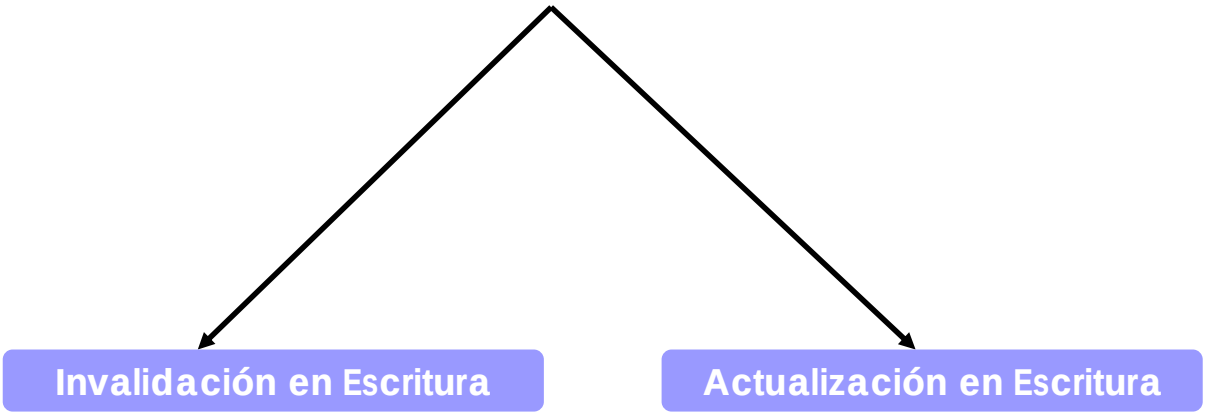
Sistema de Memoria Coherente (3)

Implícito: Propagación + Serialización de escrituras



Sistema de Memoria Coherente (4)

Políticas para mantener la coherencia

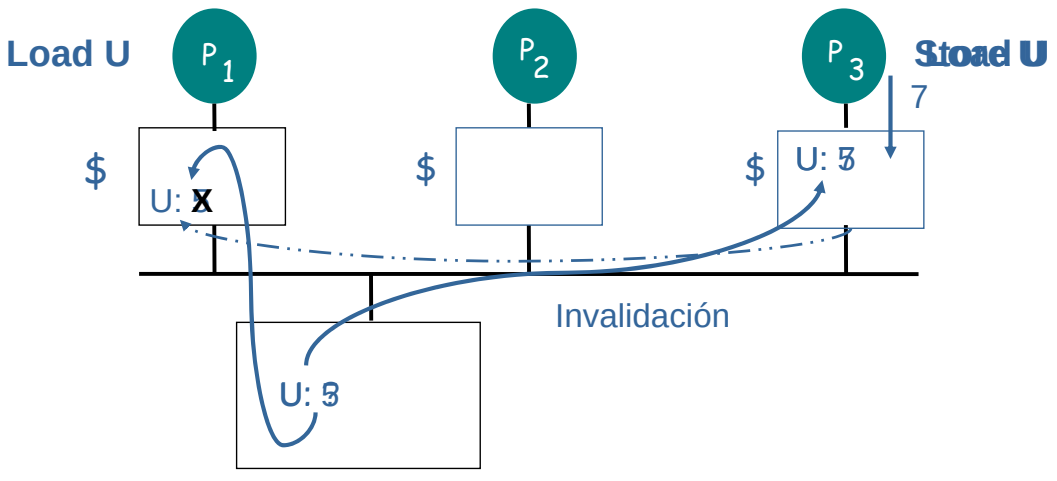


Sistema de Memoria Coherente (5)

- **Invalidación en Escritura / Coherencia Dinámica**
  - Al escribir en un bloque se invalidan todas las otras copias
    - Múltiples lectores , un solo escritor
  - Escrituras consecutivas al mismo bloque efectuadas desde el mismo procesador se realizan localmente
  - Fallo de Lectura
    - Escritura Directa (write-through):
      - la memoria esta siempre actualizada
    - Postescritura (write-back):
      - búsqueda en caches remotas para encontrar el ultimo valor

Sistema de Memoria Coherente (6)

- **Invalidación en Escritura / Coherencia Dinámica**



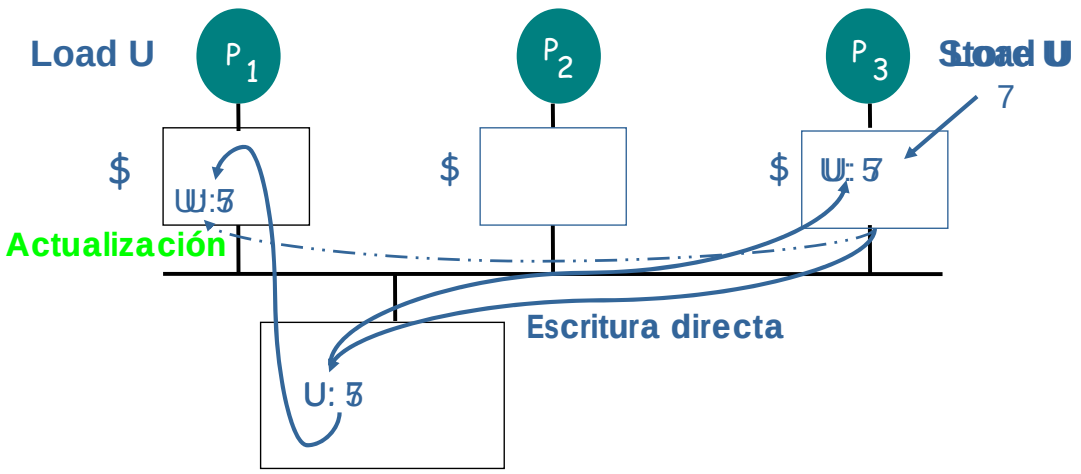
Depende de la política de actualización de la memoria

Sistema de Memoria Coherente (7)

- **Actualización en Escritura**
  - Al escribir en un bloque se actualizan todas las copias
    - Escritura directa, pocos procesadores
  - Escrituras consecutivas a la misma palabra requiere múltiples actualizaciones (*write merge*)
  - Fallo de Lectura
    - Se busca en la memoria → siempre esta actualizada

Sistema de Memoria Coherente (8)

- **Actualización en Escritura**



Sistema de Memoria Coherente (7)

Protocolos para implementar estas políticas de coherencia en caches

Dependen de la red de interconexión

Sistema de Memoria Coherente (7)

- Políticas dependen de la red de interconexión
  - **Broadcast eficiente / Buses compartidos**
    - Las operaciones de invalidación o actualización se pueden enviar de forma simultanea a todos los controladores.

Buses: Protocolo Snoopy

Observación del Bus

- El controlador de cache de cada procesador espía los paquetes que hay en el bus y actúa en consecuencia (invalidando o actualizando la copia que tiene de un bloque)

Sistema de Memoria Coherente (7)

- Políticas dependen de la red de interconexión
  - **Redes con difusión costosa** de implementar o en las que se requiere mayor escalabilidad
    - La actualización se envía únicamente a aquellas caches que tienen una copia del bloque.

Protocolos Basados en Directorio

Se indica en que caches existe copia y en que estado

Protocolos *Snoopy* (1)

- Se aplican en Redes con Broadcast eficiente, Bus compartido
- En Bus compartido es Sencillo de implementar coherencia:
  - Las operaciones de invalidación o actualización se pueden enviar de forma simultanea a todos los controladores de caché.

↓

Escrituras visibles a todos los procesadores

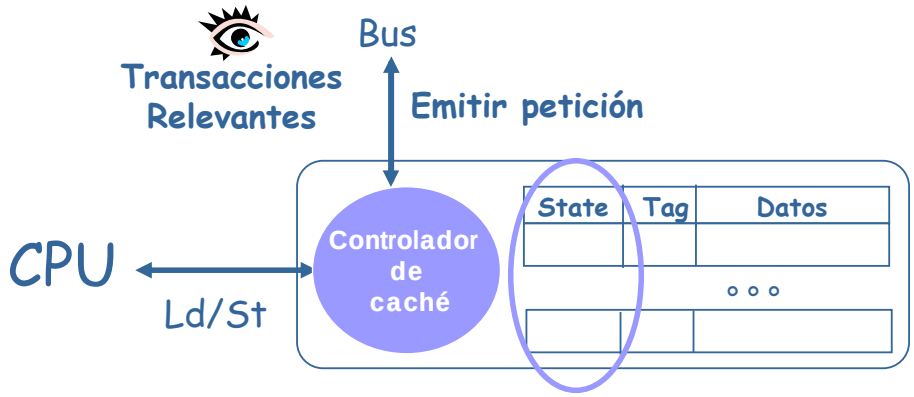
  - Es el mismo bus el que crea el orden, según cuándo le van llegando las operaciones de memoria

↓

Todas las escrituras a una posición de memoria se ven en el mismo orden por todos los procesadores

Procolos *Snoopy* (2)

- **Se basan en Observar/Espiar el bus**
  - El controlador de caché de cada procesador espía los paquetes que hay en el bus y actúan en consecuencia (invalidando o actualizando la copia que tiene de un bloque)



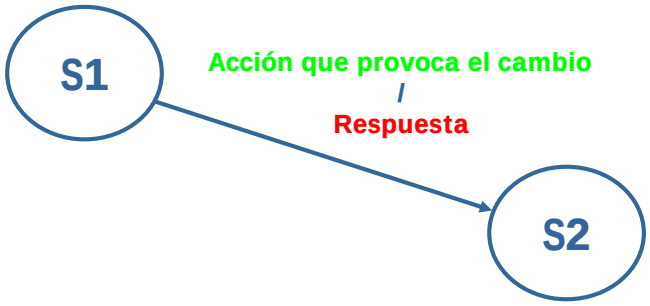
Sólo es necesario extender la funcionalidad del controlador de cache
La CPU permanece inalterada

Procolos *Snoopy* (3)

- Facetas de diseño de protocolos Snoopy
  - Definir **Política de actualización** de la memoria principal
    - Escritura inmediata
    - Post-escritura
  - Definir **Política de coherencia** de caché
    - Escritura con invalidación
    - Escritura con actualización

Procolos *Snoopy* (4)

- Facetas de diseño de protocolos Snoopy
  - Describir el comportamiento :
    - Algoritmo formado por máquinas de estado que cooperan.
      - Cada cache tiene un Diagrama de transición de estados.
      - Los Cambios de Estado son provocados por:
        - **Transacciones del bus Relevantes**
        - **Operaciones de Memoria efectuadas por el procesador local**
      - Cada cambio de estado puede producir:
        - **Acciones**
        - **Transacciones en el bus**

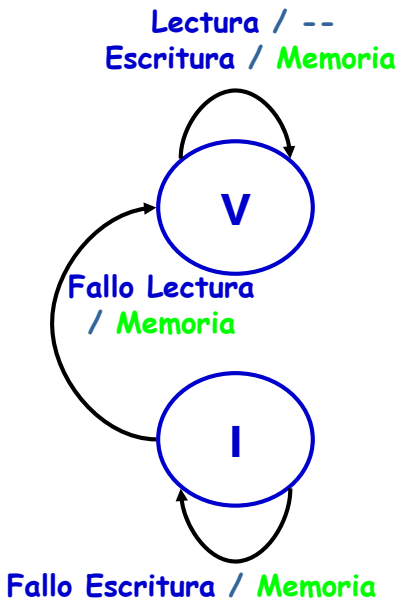


Recordatorio: caches write-through (1)

- Un único nivel de cache.
- Política de actualización de memoria
  - Escritura Directa (*write-through*)
  - Sin Asignación en Escritura (*non-allocate*)

Diagrama de Estados en Uniprocesador

- **Inicial : bloques inválidos**
- **Lectura : Inválido → Válido**
  - **Puede generar reemplazamiento**
- **Escritura : no cambia el estado**



Protocolo *Snoopy* de 2 Estados (1)

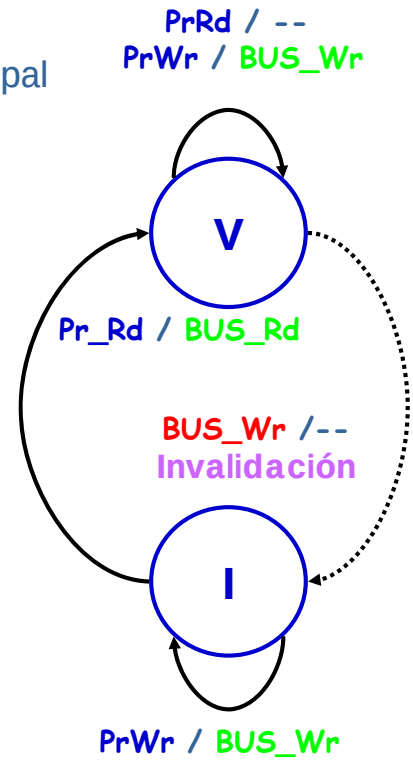
- Un nivel de cache.
- Política de actualización de la memoria principal
  - Escritura Directa (*write-through*)
  - Sin Asignación en Escritura (*non-allocate*)
- Política de Coherencia en cache
  - Invalidación

Diagrama de Estados en Protocolo Snoopy

Petición Procesador / Transacción Bus
o o
Recibir Transacción Bus / Acción

- **PrRd**: Lectura originada en el procesador
- **PrWr**: Escritura originada en el procesador
- **BusRd**: Lectura en el bus
- **BusWr**: Escritura en el bus

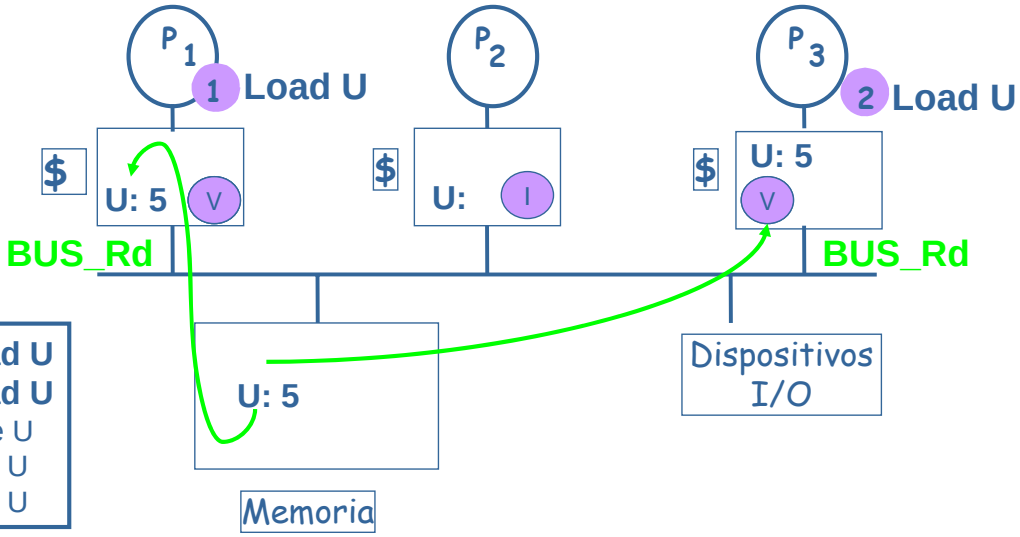
Transición iniciada en el procesador
Transición iniciada en el bus



Protocolo *Snoopy* de 2 Estados (2)

- Ejemplo:

Escritura Directa + Invalidación

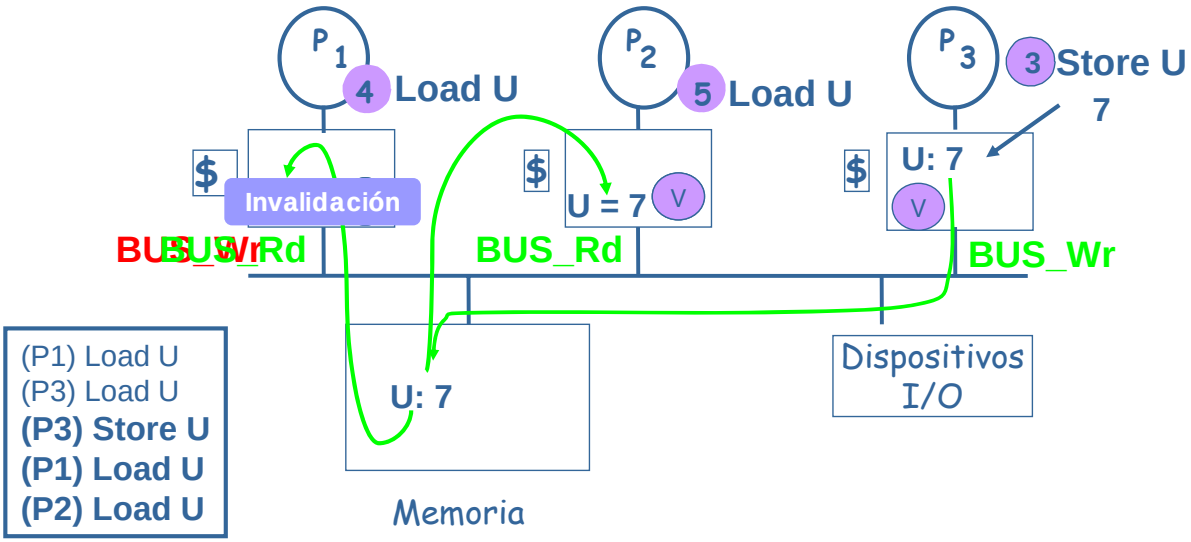


(P1) Load U
(P3) Load U
(P3) Store U
(P1) Load U
(P2) Load U

Protocolo *Snoopy* de 2 Estados (3)

- Ejemplo:

Escritura Directa + Invalidación



(P1) Load U
(P3) Load U
(P3) Store U
(P1) Load U
(P2) Load U

Protocolo *Snoopy* de 2 Estados (4)

¿**Es coherente este Protocolo** Snoopy de 2 estados?

- Los controladores de cada cache espían el bus

Propagación de escrituras

- La memoria se ocupa de las operaciones de memoria en el orden en que estas se presentan al bus (arbitraje del bus).
- Los controladores de cache las ven también en ese orden en que son presentadas al bus.
- **Escritura Directa** actualiza la memoria principal cada vez que se escribe en caché

Serialización de escrituras

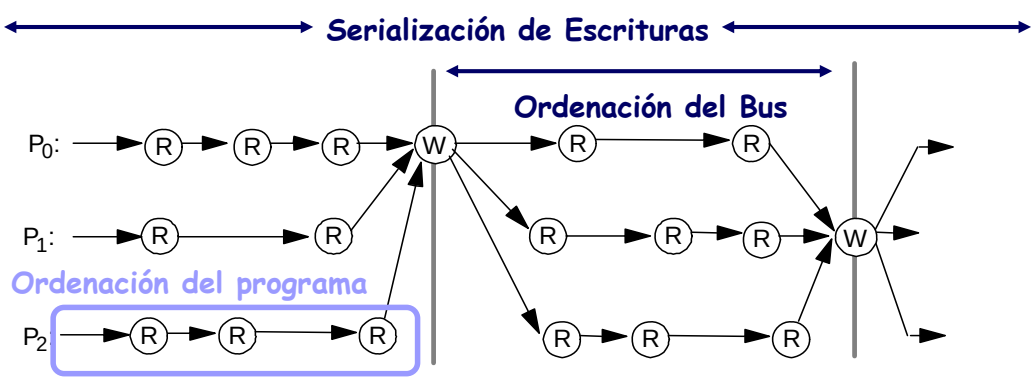
Protocolo *Snoopy* de 2 Estados (5)

- **¿Valor devuelto por lectura es el último escrito?**
  - Escrituras serializadas por el bus
  - Lecturas no totalmente serializadas
    - Los fallos de lectura → Serializados por el bus
    - Los aciertos de lectura → ????
      - El valor que tiene la cache lo puso:
        - La última escritura
        - El último fallo de lectura de ese procesador

Ambas operaciones están serializadas

Protocolo *Snoopy* de 2 Estados (6)

- ¿Es coherente el Protocolo Snoopy de 2 estados?



Protocolo *Snoopy* de 2 Estados (7)

- Problema: Ancho de Banda
  - Ejemplo:
    - Procesador a 2000 MHz
    - CPI = 1
    - 15% stores de 8 bytes
    - 300 Millones de stores por segundo por procesador
    - 2400 MB/s por procesador
    - Un bus con un ancho de banda de 10GB/s sólo puede soportar 4 procesadores sin saturarse

Solución: utilizar post-escritura (Write-back)

Recordatorio: caches write-back (1)

- Política de actualización de la memoria principal
  - **Post-Escritura** (*write-back*)
  - Asignación en Escritura (*write-allocate*)
- Diagrama de transición de estados de la caché

  Tres estados:

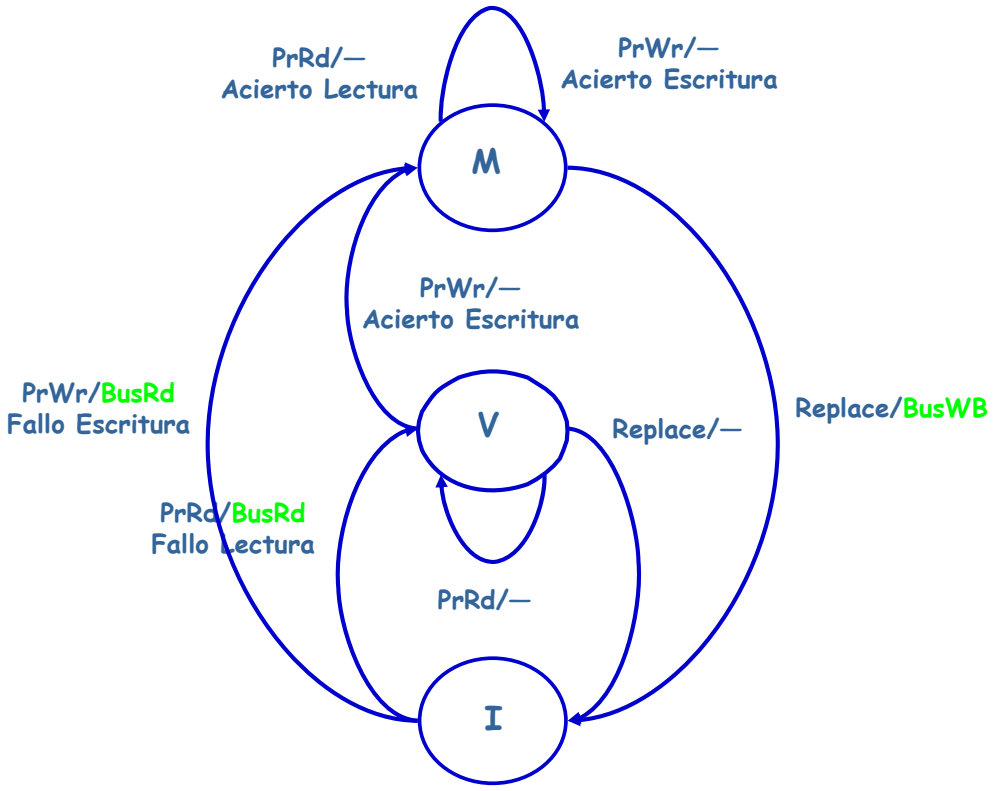
- El bloque está en la cache y no ha sido modificado
- La memoria principal actualizada

- El bloque está en la cache y ha sido modificado
- La memoria principal NO actualizada

Recordatorio: caches write-back (2)

- Diagrama de transición de estados de la cache
  - Accesos a memoria / Transiciones de Bus
    - **Fallo de lectura / Petición de lectura de un bloque *(BusRd)***
    - **Fallo de escritura / Petición de lectura de un bloque *(BusRd)*** (asignación en escritura)
    - **Reemplazamiento / Petición de postescritura *(BusWB)***
  - Acierto Escritura y Lectura no generan transición de bus

Recordatorio: caches write-back (3)



Protocolo *Snoopy* de 3 Estados(1)

- Extensión del protocolo de monoprocesador a multiprocesador → **MSI**
  - Política de actualización de la memoria principal
    - **Post-Escritura** (*write-back*)
    - Asignación en Escritura (*write-allocate*)
  - Política de coherencia en cache
    - **Invalidación**

Protocolo *Snoopy* de 3 Estados(2)

- Diagrama de transición de estados de la caché: 3 estados

Lectura y Escritura

- Sólo hay una cache con copia válida
- La copia de la memoria principal está anticuada
-Exclusividad implica que la cache puede modificar el bloque sin notificárselo a nadie

Sólo lectura

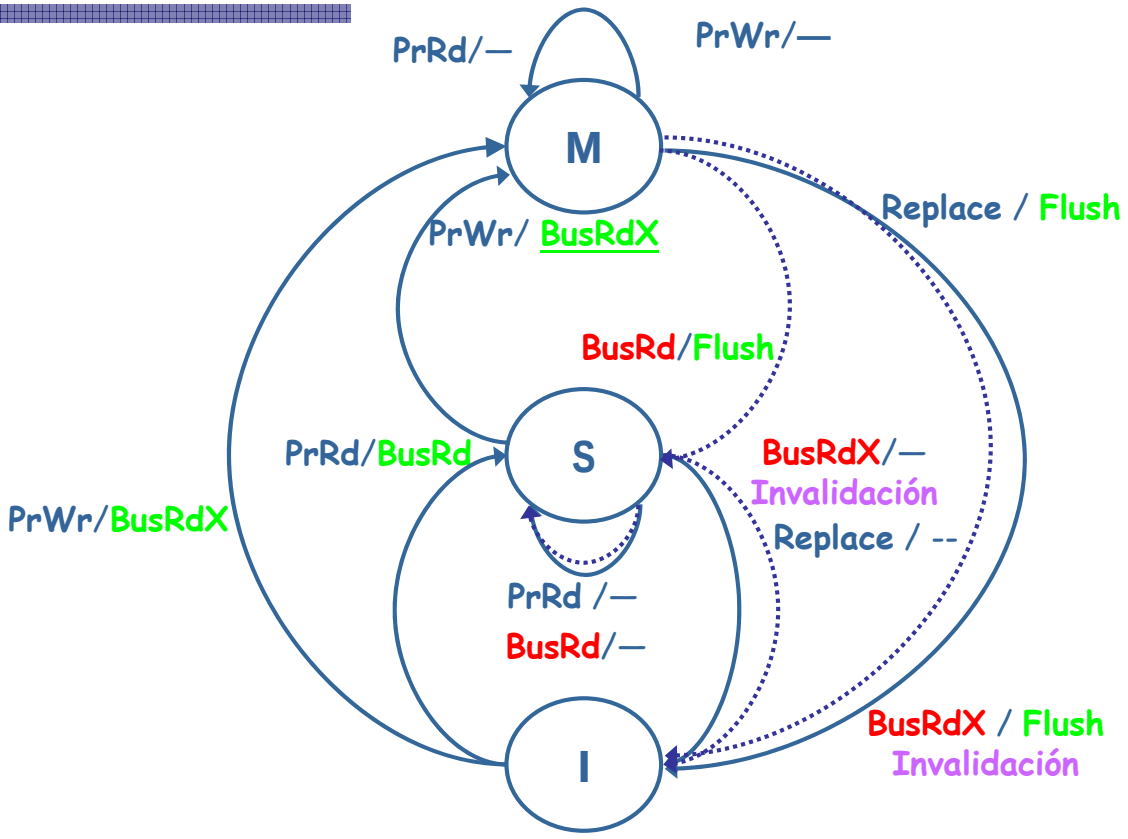
- El bloque está en la cache y no ha sido modificado
- La memoria principal actualizada
- Otras caches pueden tener copia

O no presente

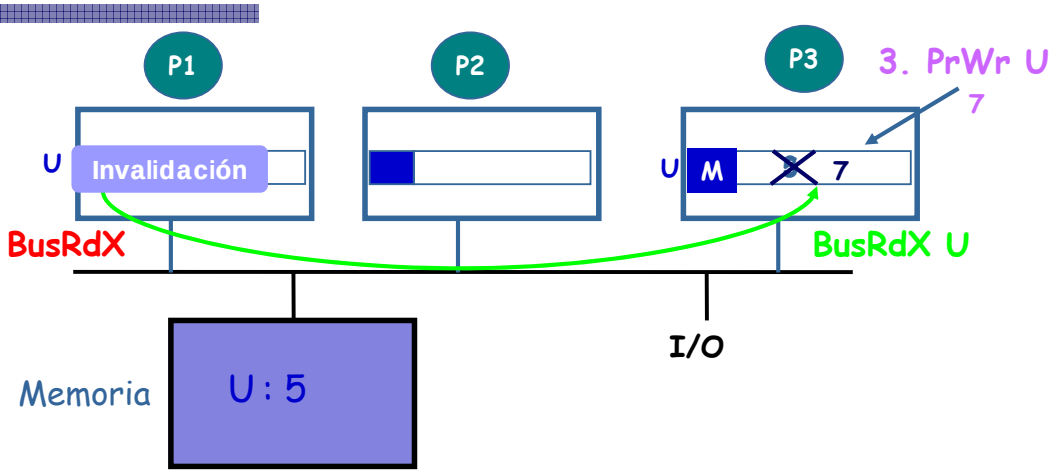
Protocolo *Snoopy* de 3 Estados(3)

- Diagrama de transición de estados de la caché
  - Transacciones de bus
    - BusRd → Lectura sin intención de modificar el dato.
      - Se genera con fallo de lectura.
      - Dirección en el bus y espera recibir una copia del bloque
      - El sistema de memoria (u otro cache) suministra los datos
    - **BusRdX** → Lectura con intención de modificarlo. (Nuevo)
      - Se genera con fallo de Esc o una escritura estando en estado "V"
      - Resto de caches **invalidan** su copia.
      - Dirección en el bus y espera recibir una copia del bloque
      - El sistema de memoria (u otro cache) suministra los datos
      - El procesador no escribe hasta recibir reconocimiento de la transacción
    - BusWB o Flush →
      - Se genera con un reemplazamiento.
      - El controlador vuelca al bus la dirección y el contenido del bloque.
      - La memoria principal se actualiza.

Protocolo MSI: invalidación de 3 estados (1)

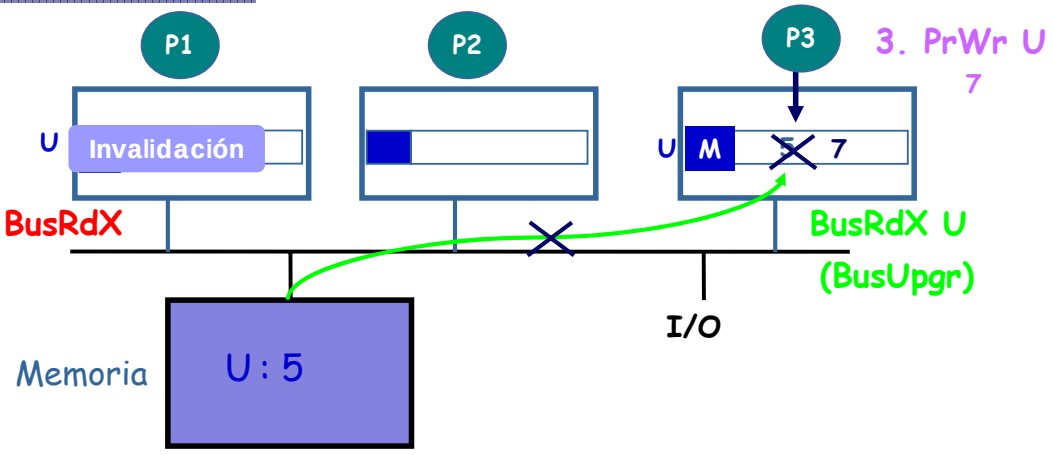


Protocolo MSI: invalidación de 3 estados (2)



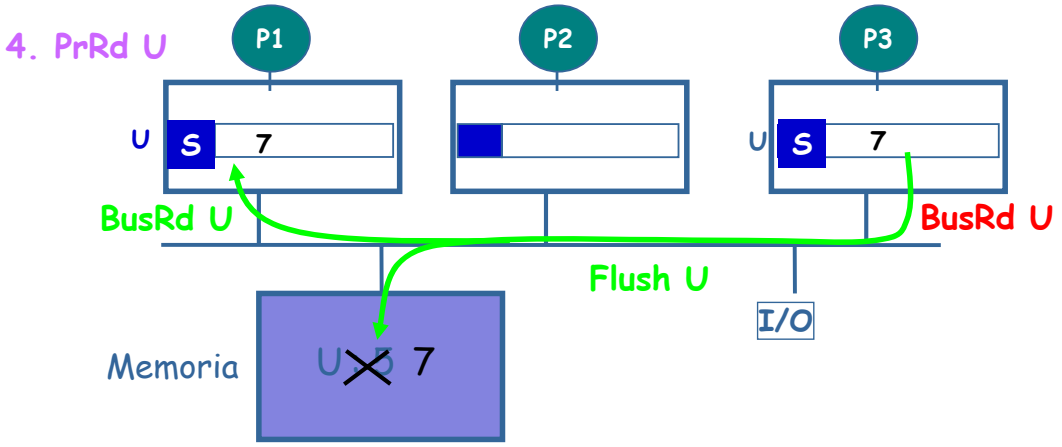
Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 escribe U	M	—	I	BusRd	Memoria
P3 escribe U	I	—	M	BusRdX	Cache P1

Protocolo MSI: invalidación de 3 estados (2)



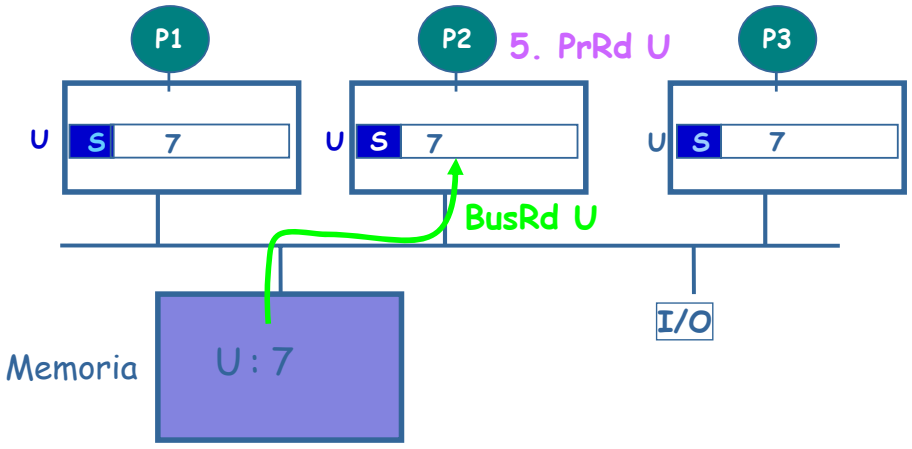
Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	S	—	—	BusRd	Memoria
P3 Lee U	S	—	S	BusRd	Memoria
P3 escribe U	I	—	M	BusRdX (BusUpgr)	Memoria (ignorados)
P1 Lee U					
P2 Lee U					

Protocolo MSI: invalidación de 3 estados (3)



Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	S	—	—	BusRd	Memoria
P3 Lee U	S	—	S	BusRd	Memoria
P3 escribe U	I	—	M	BusRdX (BusUpg)	Memoria (ignorados)
P1 Lee U	S	—	S	BusRd	Cache P3
P2 Lee U					

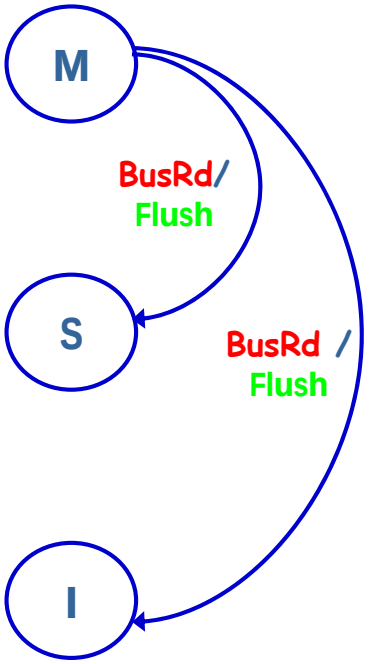
Protocolo MSI: invalidación de 3 estados (4)



Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	S	—	—	BusRd	Memoria
P3 Lee U	S	—	S	BusRd	Memoria
P3 escribe U	I	—	M	BusRdX (BusUpg)	Memoria (ignorados)
P1 Lee U	S	—	S	BusRd	Cache P3
P2 Lee U	S	S	S	BusRd	Memoria

Protocolo MSI: Problemas

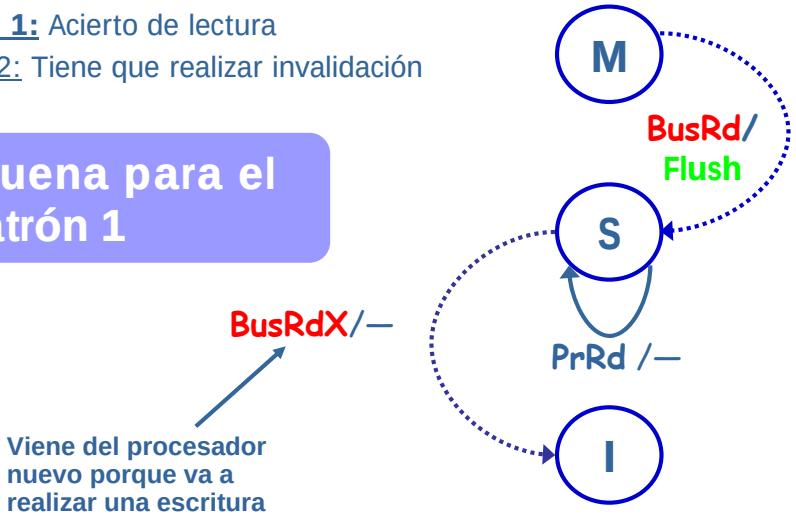
- **Problema1: Decisiones de bajo nivel**
  - ¿Qué transición debe producirse cuando se recibe **BusRd** de un bloque en estado modificado?



Protocolo MSI: Problemas

- **La elección depende de las expectativas (patrón de acceso)**
  - Patrón 1: el procesador "original" vuelve a realizar una lectura del bloque
  - Patrón 2: el procesador "nuevo" realiza una escritura del bloque
- **Modificada--Compartida**
  - **Patrón 1:** Acierto de lectura
  - Patrón 2: Tiene que realizar invalidación

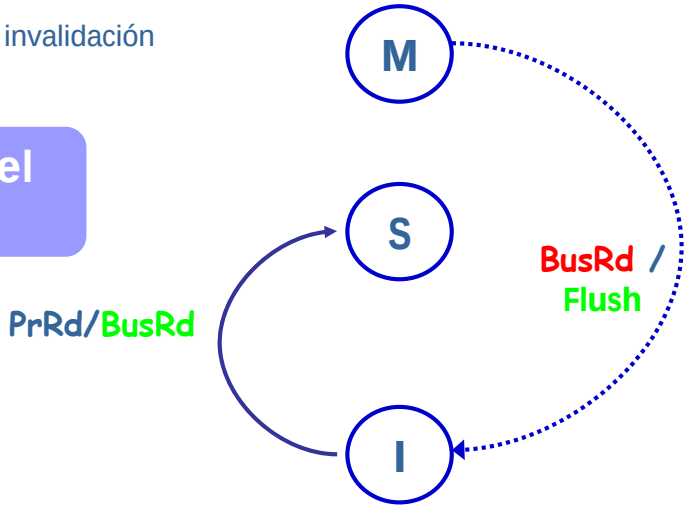
Opción buena para el patrón 1



Protocolo MSI: Problemas

- **La elección depende de las expectativas (patrón de acceso)**
  - Patrón 1: el procesador "original" vuelve a realizar una lectura del bloque
  - Patrón 2: el procesador "nuevo" realiza una escritura del bloque
- **Modificada--Invalido**
  - Patrón 1: Fallo de lectura
  - Patrón 2: No es necesaria invalidación

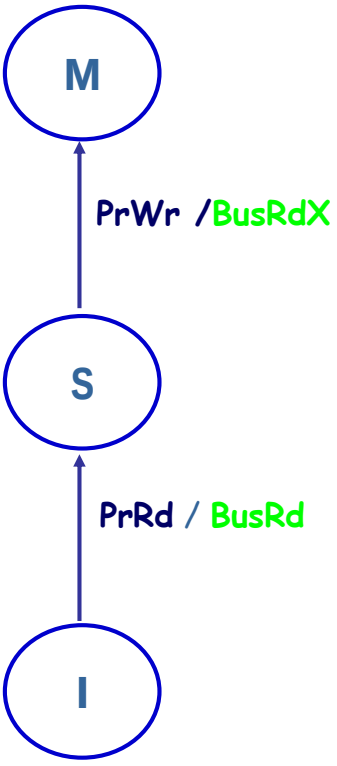
Opción buena para el patrón 2



Protocolo MSI: Problema

- Problema2:
  - **Lectura-Modificación de un dato en un mismo procesador**:
    - **2 transacciones de bus:**
      - BusRd:
        - Pone el bloque en estado compartido
      - BusRdX:
        - Pasa de S a M
        - Invalida el resto de copias.

  - Si el bloque no existe en ninguna otra cache:
    - Se puede ir directamente a Modificado
      - Ahorra una transición
    - No hace falta invalidar (BusRdX)



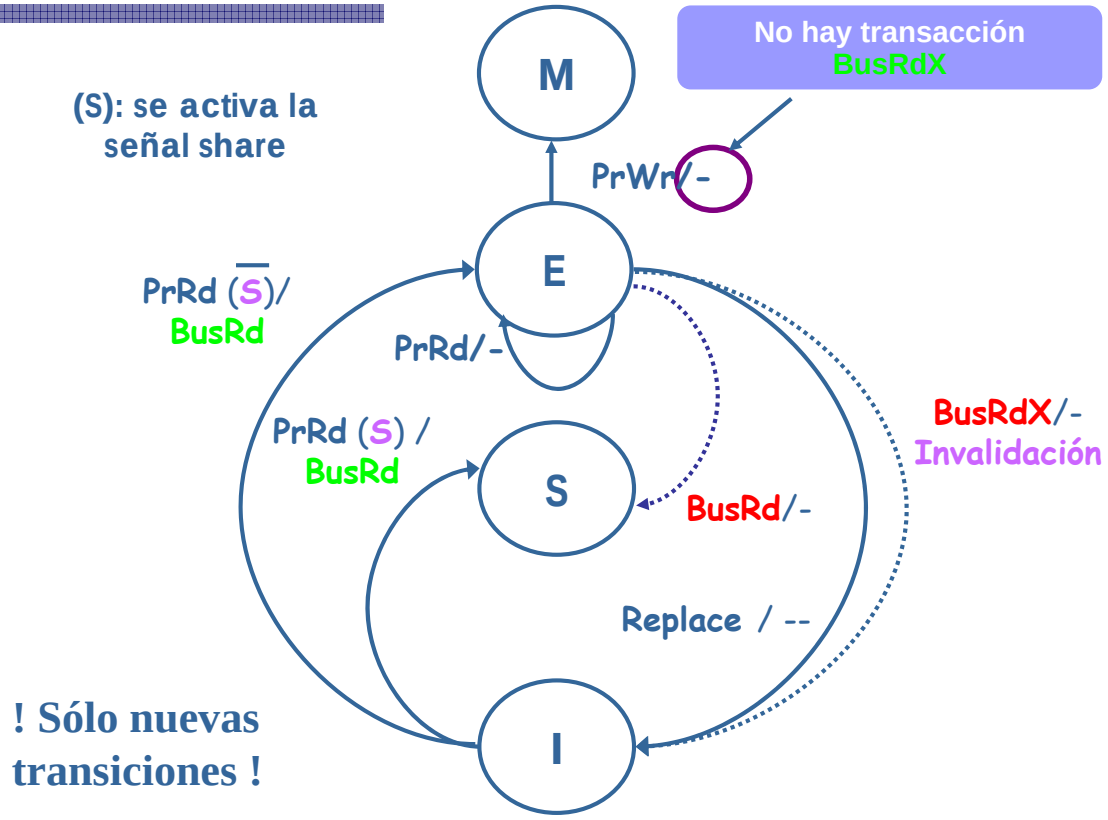
Protocolo MESI: Invalidación de 4 estados (1)

- Protocolo MESI
  - 4 Estados :
    - (M) Modificado
    - (S) Compartido
    - (I) Inválido
    - **(E) Exclusivo**
  - Nuevo estado (Exclusive Clean / Exclusive)
    - Indica que el bloque **sólo está en una cache**
    - El bloque **no ha sido modificado**
    - La **memoria** principal está **actualizada**
    - Implica Exclusividad:
      - Puede pasarse a (M) sin transacción de bus
    - No implica Pertenencia:
      - Al contrario que estado (M) el controlador de cache no debe responder a transacción BusRd con el bloque

Protocolo MESI: Invalidación de 4 estados (2)

- Un nuevo requisito en la interconexión física del bus
  - Señal adicional
    - **S** (señal de bloque compartido)
    - Es una **Or-cableada**
  - Cuando un procesador solicita un bloque
    - Mediante **S** Los controladores de cache pueden determinar si hay otra cache con el bloque
      - **SI**
        - El bloque solicitado pasa a estado **Compartido**
      - **NO**
        - El bloque solicitado pasa a estado **Exclusivo**

Protocolo MESI: Invalidación de 4 estados (3)

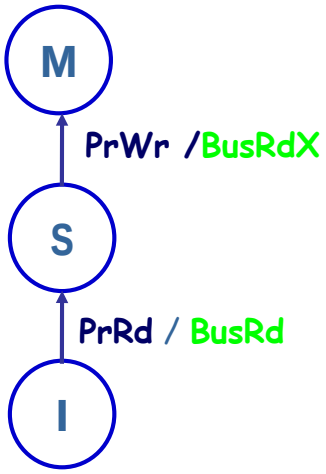


Protocolo MESI versus MSI

- **Lectura-Modificación de un dato en un mismo procesador**:
  - Si el bloque no existe en ninguna otra cache:

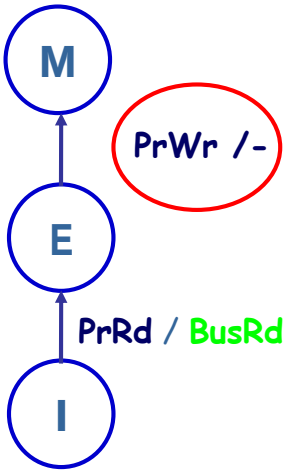
2 transacciones en el bus

MSI

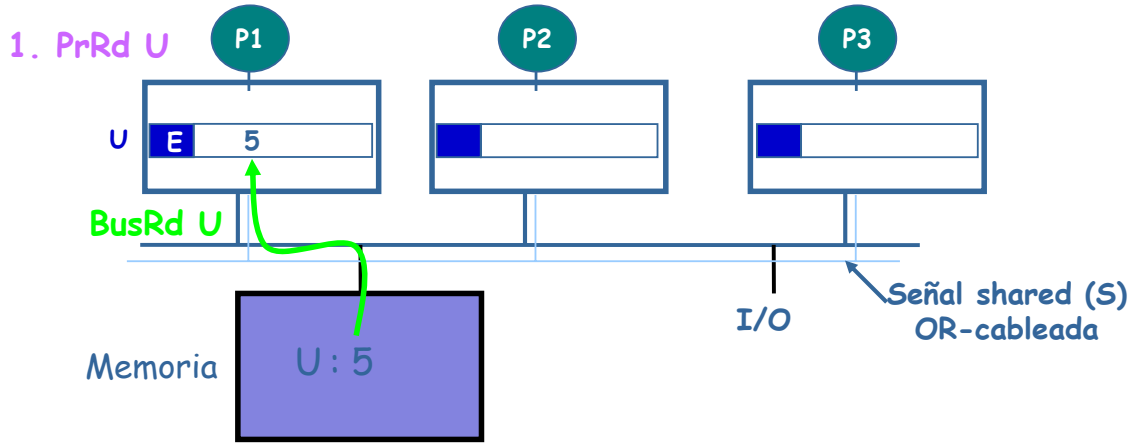


1 transacciones en el bus

MESI

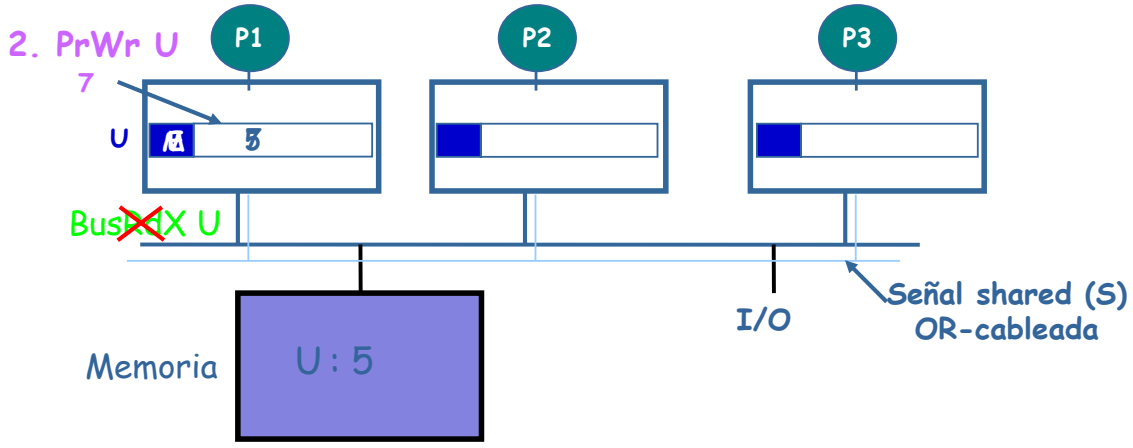


Protocolo MESI: Invalidación de 4 estados (4)



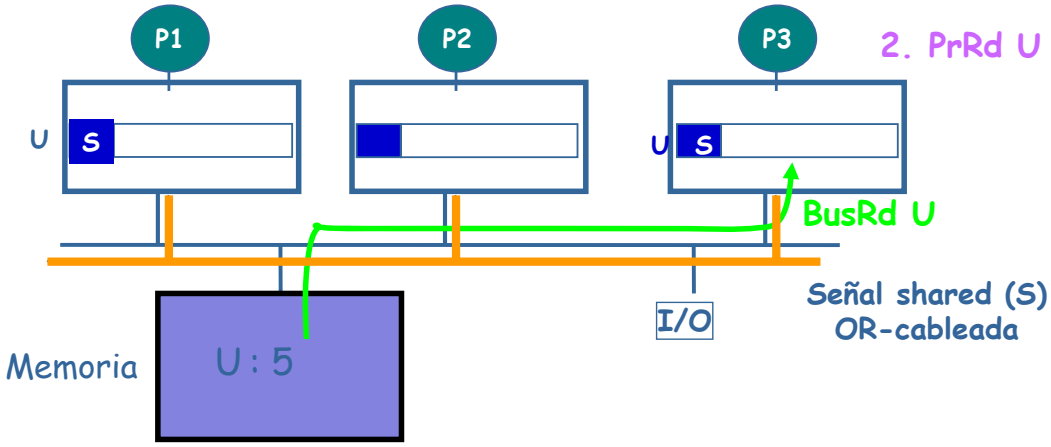
Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria

Protocolo MESI: Invalidación de 4 estados (5)



Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria
P1 Escribe U	M	—	—	—	—

Protocolo MESI: Invalidación de 4 estados (6)



Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria
P3 Lee U	S	—	S	BusRd	Memoria

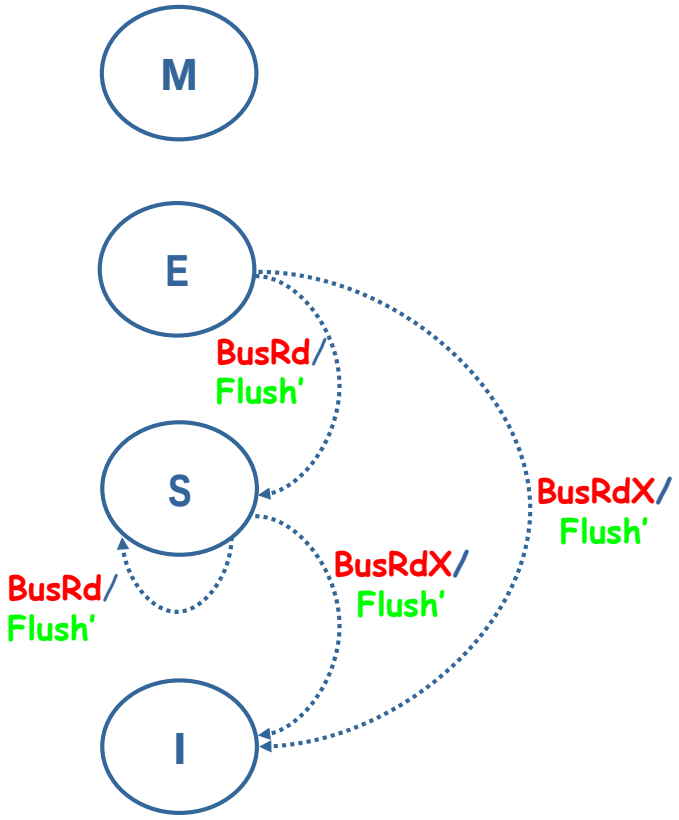
Protocolo Illinois: MESI Original (1)

- ¿**Quién debería suministrar el bloque** en una transacción **BusRd** cuando tanto la memoria como otra cache tienen una copia válida?
  - **Cache-to-Cache sharing**
    - Nueva transacción bus **(Flush')**
      - La cache pone los datos en el bus
      - No se escribe en memoria, sólo se escribe en la cache que solicita el bloque
    - ¿Por qué esta tansición?
      - Teóricamente SRAM (Caches) suministran los datos más rápidamente que DRAM
      - **Muy útil en arquitecturas NUMA**
        - La latencia para obtener datos desde una cache cercana es más pequeña que la de una unidad de memoria principal lejana

Protocolo Illinois: MESI Original (2)

-Flush': se envia el bloque a otra cache

-No actualiza memoria

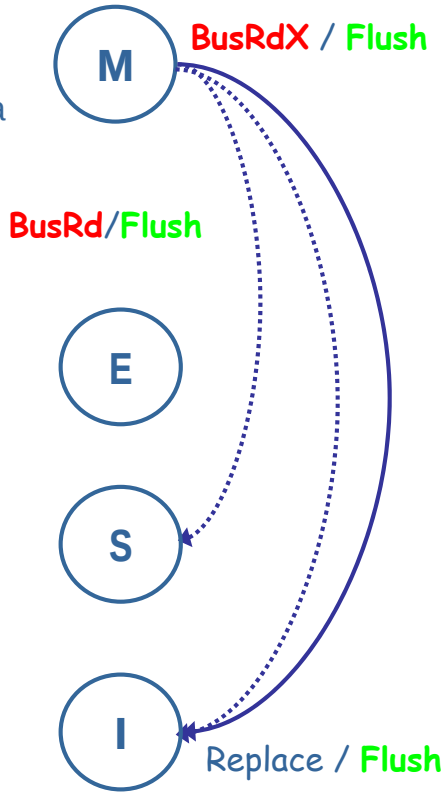


Protocolo Illinois: MESI Original (3)

- **Cache-to-Cache sharing**
  - Problema
    - Añade complejidad:
      - Múltiples caches: **Algoritmo de selección**
      - **La memoria tiene que esperar un cierto tiempo para asegurarse que no suministrará los datos una cache**

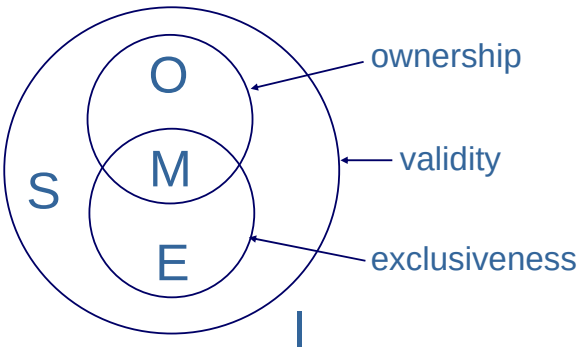
Protocolo MSI y MESI: problema

- **Transacción Flush**
  - Transfiere el bloque a la cache que lo solicita
  - Actualiza el contenido de la memoria
- Podría evitarse esta actualización
  - **Transacción Flush´**
    - Transfiere el bloque a la cache que lo solicita
- Hacer la actualización sólo con "Replace"



Protocolo MOESI: invalidación de 5 estados (2)

- Estados
  - (M) Modificado
  - (S) Compartido
  - (I) Inválido
  - (E) Exclusivo
  - **(O) Owned**



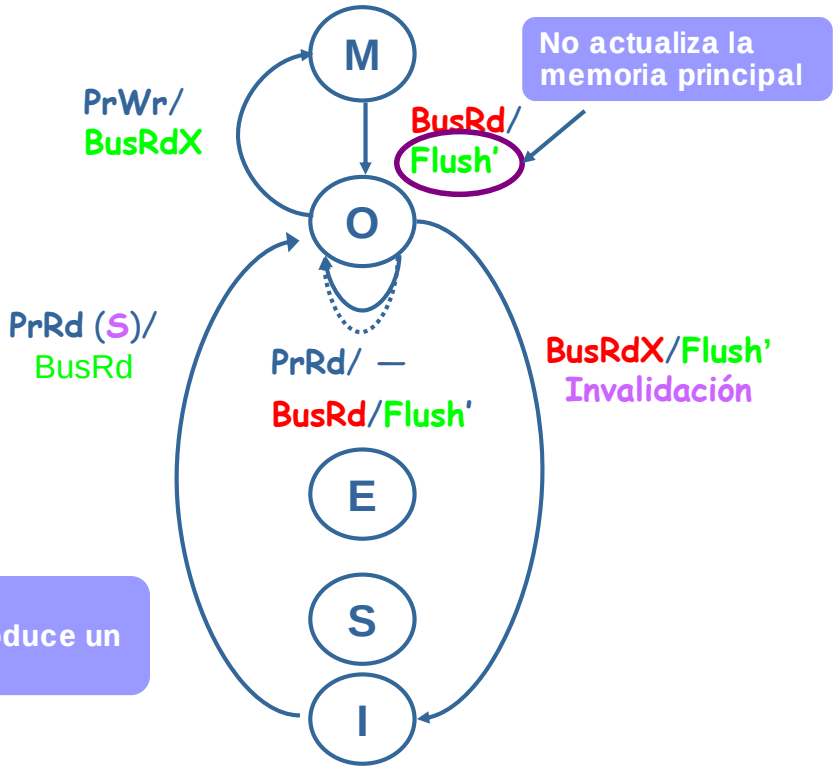
- Nuevo estado (O) Owned
  - No es Exclusivo, existen otras copias
  - Es Propietario, suministra los datos en caso de:
    - BusRd
    - BusRdX

[Sweazey & Smith ISCA86]

Protocolo MOESI: invalidación de 5 estados (2)

-Flush': se envia el bloque a otra cache

-No actualiza memoria



La memoria principal se actualiza cuando se produce un reemplazamiento

Calculo ancho de banda

- Ancho de banda
  - ¿Cómo se calcula?
    - Transiciones/S → Transacciones/s → Bytes/s

Dependiente Workload

From/To	NP	I	E	S	M
NP			BusRd 6+64	BusRd 6+64	BusRdX 6+64
I			BusRd 6+64	BusRd 6+64	BusRdX 6+64
E					
S			NA		BusUpg 6
M	Flush 6+64	Flush 6+64	NA	Flush 6+64	

Soponiendo:

Bloque = 64B

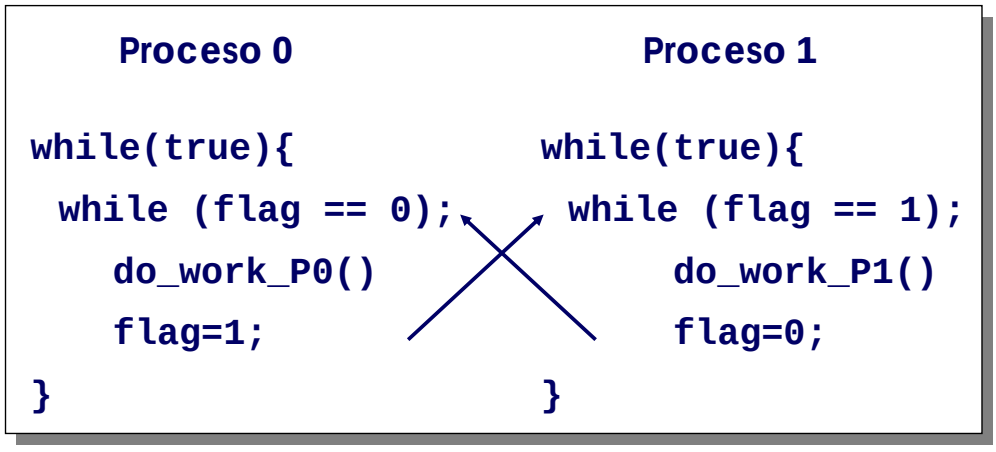
Dir+cmd = 6B

Resumen: protocolos de invalidación

- Snoopy 2 estados
  - Problema latencia y ancho de banda (escritura directa)
- MSI: BusUpgrd vs BusRdX
  - BusUpgrd (S→M): ahorro 10-20% respecto a BusRdX
- MSI vs. MESI
  - La transición E→M no produce tráfico en el bus
  - Pequeño **Invalidación**
    - Pocas transiciones E→M
    - Con BusUpgrd, el ahorro es pequeño (Ej: 6B)
- MOESI vs MESI
  - Mismo ancho banda, menor latencia
- Inconveniente
  - Fallos de lectura por invalidación **Actualización**

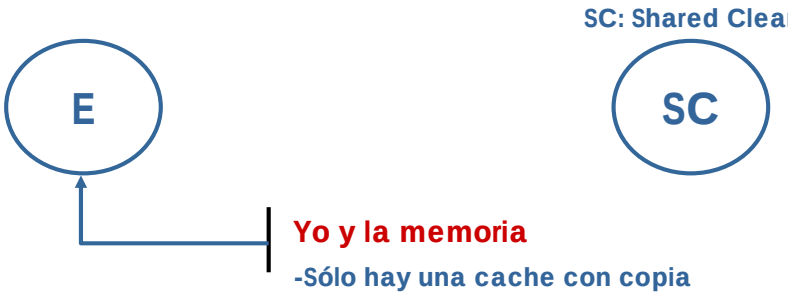
Problema patrones migratorios

- Problema patrones migratorios
  - Cuando hay un patrón de acceso migratorio, los protocolos basados en invalidación son ineficientes.
    - Ejemplo: flag de sincronización



¿Cuantas transacciones son necesarias en un protocolo de invalidación?

Protocolo Dragón: actualización 4 estados (1)



Yo y la memoria

- -Sólo hay una cache con copia
- -El bloque no ha sido modificado
- -La memoria está actualizada

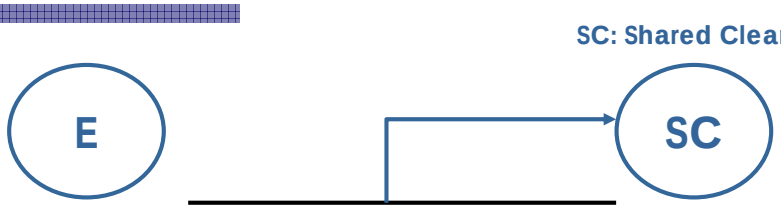
SM: Shared Modified

Yo y nadie más

- -El bloque ha sido modificado
- -La memoria no está actualizada
- -Esta es la única copia (Propiedad Exclusiva)

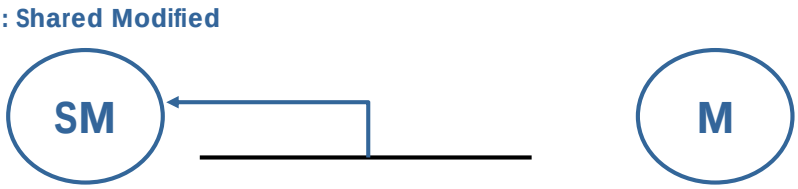
Dragón de Xerox PARC (1984)
Sun SparcServer (1997)

Protocolo Dragón: actualización 4 estados (2)



Yo, otros y puede que la memoria, pero no soy el propietario

- -Potencialmente 2 o más procesadores tienen este bloque.
- -La memoria principal puede estar o no actualizada



Yo y otros pero no la memoria, soy el propietario

- -Potencialmente 2 más procesadores tienen este bloque.
- -La memoria principal no está actualizada
- -Este procesador es responsable de actualizar la memoria principal cuando este bloque sea reemplazado

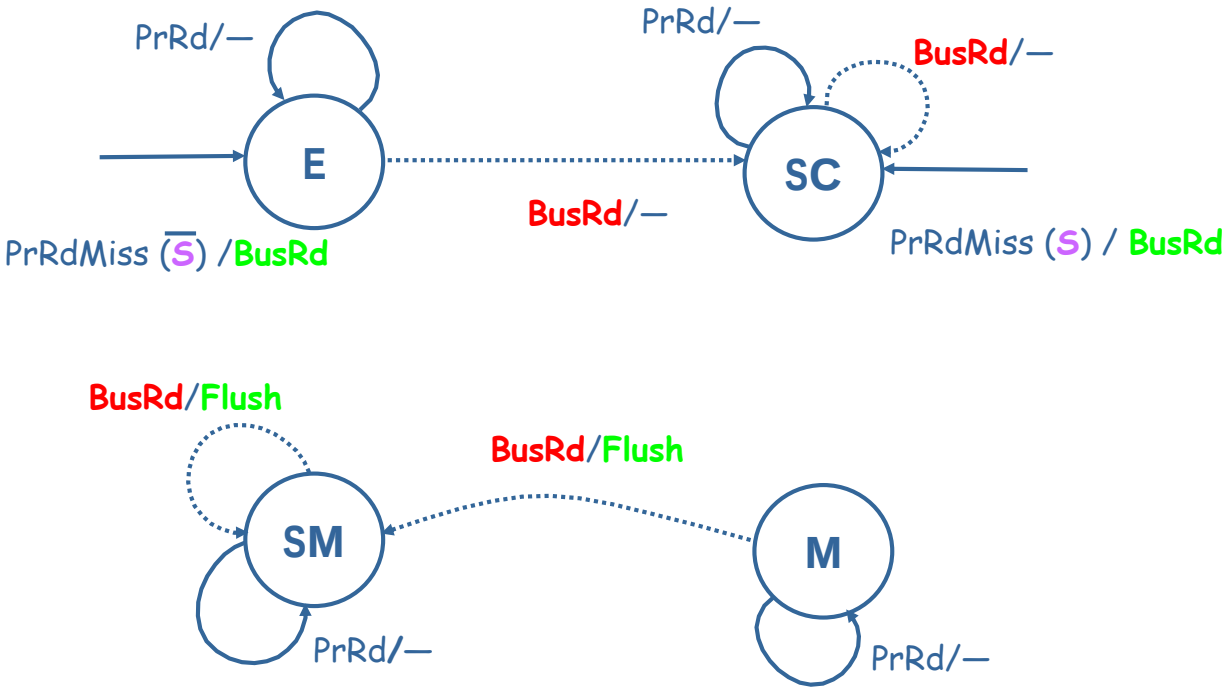
Protocolo Dragón: actualización 4 estados (3)

- **No existe estado invalido**
  - Porque está basado en actualización, no en invalidación.
    - Los bloques en cache siempre actualizados.
    - Siempre se pueden usar los datos presentes en la cache.
- ¿Qué pasa con los bloques que no están en cache?
  - Eventos Procesador
    - PrRd / PrWr
    - **PrRdMiss / PrWrMiss**

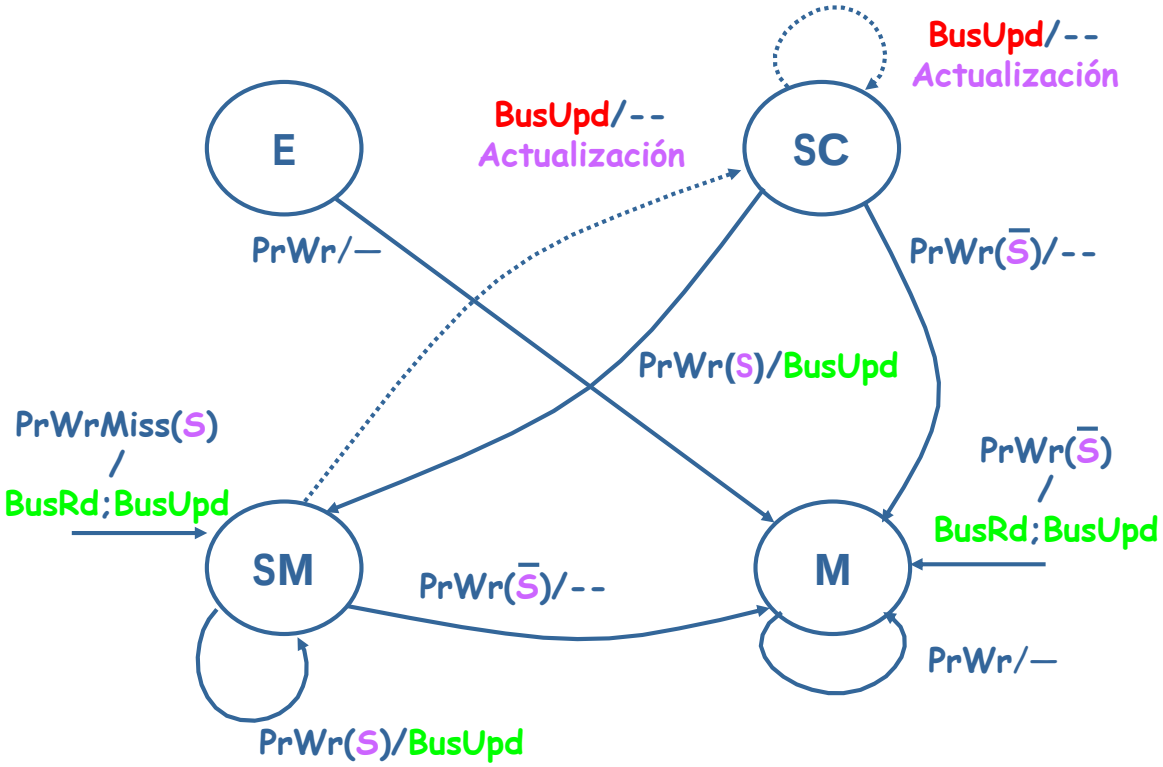
Protocolo Dragón: actualización 4 estados (2)

- Transacciones del Bus
  - BusRd / BusWB o Flush
  - **BusUpd** (Nueva) en lugar de BusRdX
    - Broadcast de la **palabra modifica** (no de todo el bloque) a todos los procesadores para que se actualicen las copias.
    - También actualiza la memoria principal
- Acciones
  - **Actualizar** (Nueva) en lugar de Invalidar
    - Controlador de cache actualiza un bloque si lo tiene en su cache y está siendo “broadcasted” en el bus
- Señal S
  - Para soportar el estado Exclusivo (igual que MESI)

Protocolo Dragón: actualización 4 estados (3)



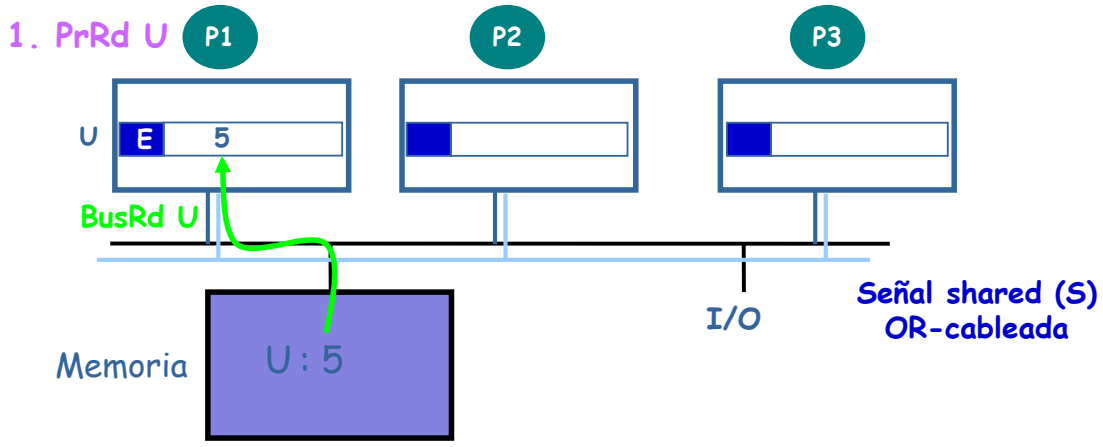
Protocolo Dragón: actualización 4 estados (4)



Protocolo Dragón: actualización 4 estados (5)

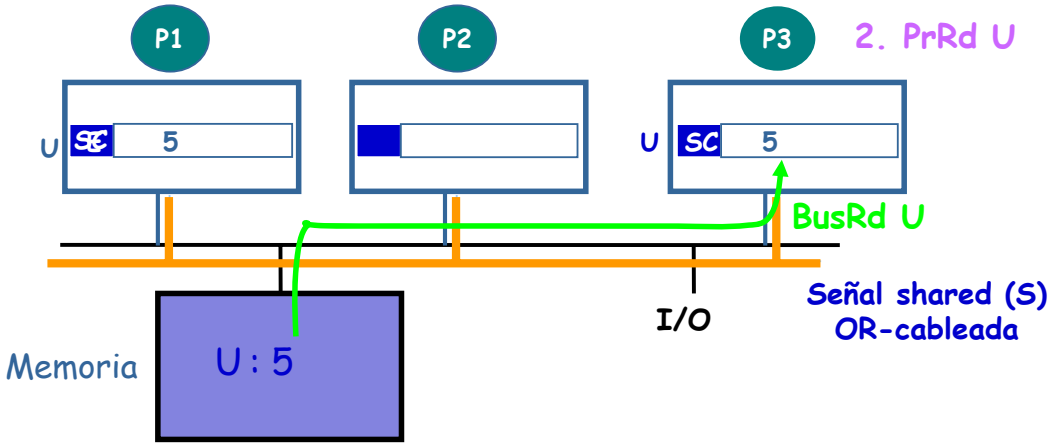
- Elecciones de diseño implícitas en este protocolo
  - ¿Puede eliminarse el estado SM?
    - NO
      - La actualización de SRAM caches es menos costosa que la actualización de memoria.
        - No compensa actualizar memoria en cada BusUp
    - SI
      - Añadir a la transacción BusUpd actualización de la memoria
        - Multiprocesador Firefly de DEC
  - ¿Debe hacerse un broadcast al resto de las controladores para informar de un reemplazo de un bloque en estado SC?
    - En caso de que haya sólo otra copia, permite que pase al estado E (no genera actualizaciones)
  - Acierto Escritura ¿Puede actualizarse la copia local antes de que adquiera el control del bus?
    - Puede complicar serialización de escrituras

Protocolo Dragón: actualización 4 estados (6)



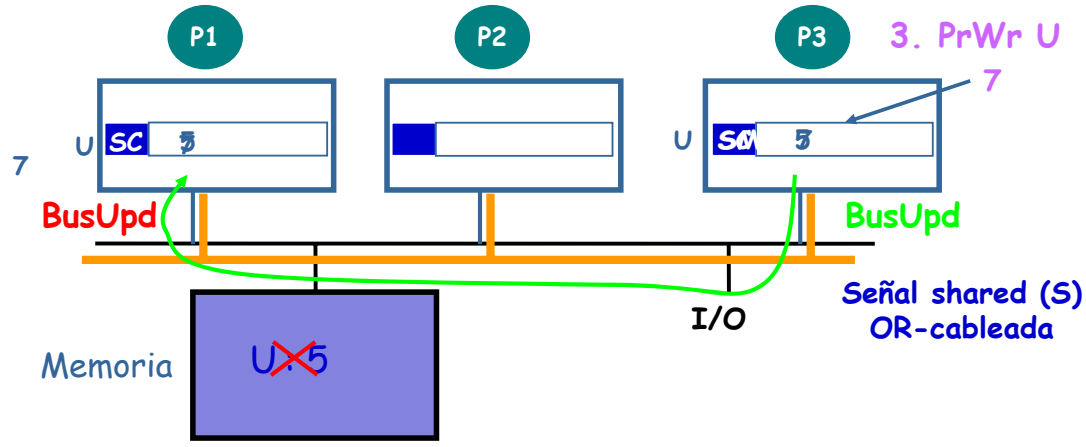
Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria
P3 Lee U					
P3 escribe U					
P1 Lee U					
P2 Lee U					

Protocolo Dragón: actualización 4 estados (7)



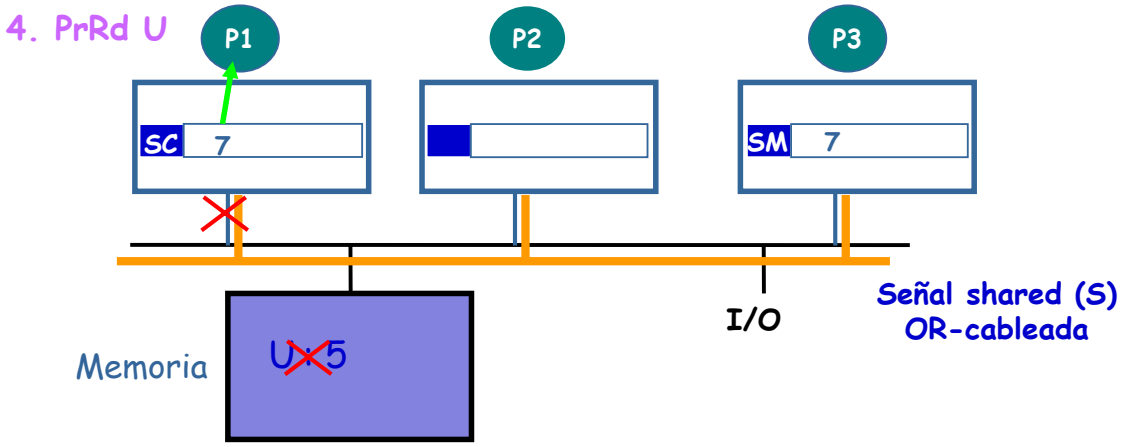
Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria
P3 Lee U	SC	—	SC	BusRd	Memoria
P3 escribe U					
P1 Lee U					
P2 Lee U					

Protocolo Dragón: actualización 4 estados (8)



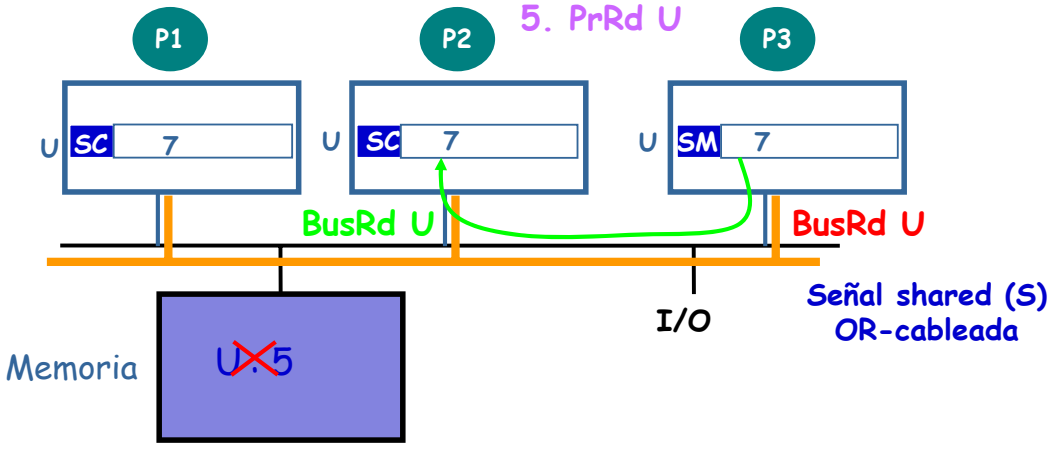
Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria
P3 Lee U	SC	—	SC	BusRd	Memoria
P3 escribe U	SC	—	SM	BusUpd	Cache P3
P1 Lee U					
P2 Lee U					

Protocolo Dragón: actualización 4 estados (9)



Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria
P3 Lee U	SC	—	SC	BusRd	Memoria
P3 escribe U	SC	—	SM	BusUpd	Cache P3
P1 Lee U	SC	—	SM	—	Cache P1
P2 Lee U					

Protocolo Dragón: actualización 4 estados (10)



Operación	Estado P1	Estado P2	Estado P3	Transacción Bus	Datos Suministrados
P1 Lee U	E	—	—	BusRd	Memoria
P3 Lee U	SC	—	SC	BusRd	Memoria
P3 escribe U	SC	—	SM	BusUpd	Cache P3
P1 Lee U	SC	—	SM	—	Cache P1
P2 Lee U	SC	SC	SM	BusRd	Cache P3

Invalidación vs Actualización (1)

- Patrón 1

```
for i = 1 to k
    P1(write, x);
    P2-PN-1 (read, x);
end for I
```

Una escritura antes de lectura

- Patrón 2

```
for i = 1 to k
    for j = 1 to m
      P1 (write, x);
    end for j
    P2 (read, x);
end for i
```

Varias escritura antes de lectura

Invalidación vs Actualización (2)

- Patrón 1
  - N = 16, M = 10, K = 10
  - Actualización
    - Iteración 1: N fallos de cache (70 bytes)
    - Iteraciones restantes: 1 actualización/iteración (14 bytes; 6 cntrl, 8 datos)
    - Tráfico total = 16*70 + 9*14 = 1246 bytes
  - Invalidación
    - Iteración 1: N fallos de cache (70 bytes)
    - Iteraciones restantes: P1 invalida (6bytes, BusUpgrd), los otros producen fallos de lectura (70bytes)
    - Tráfico total = 16*70 + 9*6 + 15*9*17 = 3469 bytes

Invalidación vs Actualización (3)

- Patrón 2
  - N = 16, M = 10, K = 10
  - Actualización
    - Iteración 1: 2 fallos de cache (70 bytes), M actualizaciones (14 bytes; 6 cntrl, 8 datos)
    - Iteraciones restantes: M actualización/iteración

      Tráfico total = 2*70 + 10*14 = 1540 bytes
  - Invalidación
    - Iteración 1: 2 fallos de cache (70 bytes)
    - Iteraciones restantes: P1 invalida (6bytes, BusUpgrd), P2 produce fallo de lectura (70bytes)
    - Tráfico total = 2*70 + 9*(70+6) = 824 bytes

Resumen

- Protocolos de coherencia *Snoopy*
  - Invalidación
    - Snoopy 2 estados
    - MSI (BusRdX vs BusUpgrd)
    - MESI (Illinois)
    - MOESI
  - Actualización
    - Dragón
- Medio de comunicación = bus
  - Ventajas
    - Orden = Arbitraje de bus
  - Desventajas
    - Escalabilidad limitada
      - Longitud y nº de conexiones