ARQUITECTURA DE COMPUTADORES II

AUTORES:
David Expósito Singh
Florin Isaila
Daniel Higuero Alonso-Mardones
Javier García Blas
Borja Bergua Guerra

Área de Arquitectura y Tecnología de Computadores
Departamento de Informática
Universidad Carlos III de Madrid

Julio de 2012

TEMA 3:
MP DE MEMORIA COMPARTIDA (Y IV)

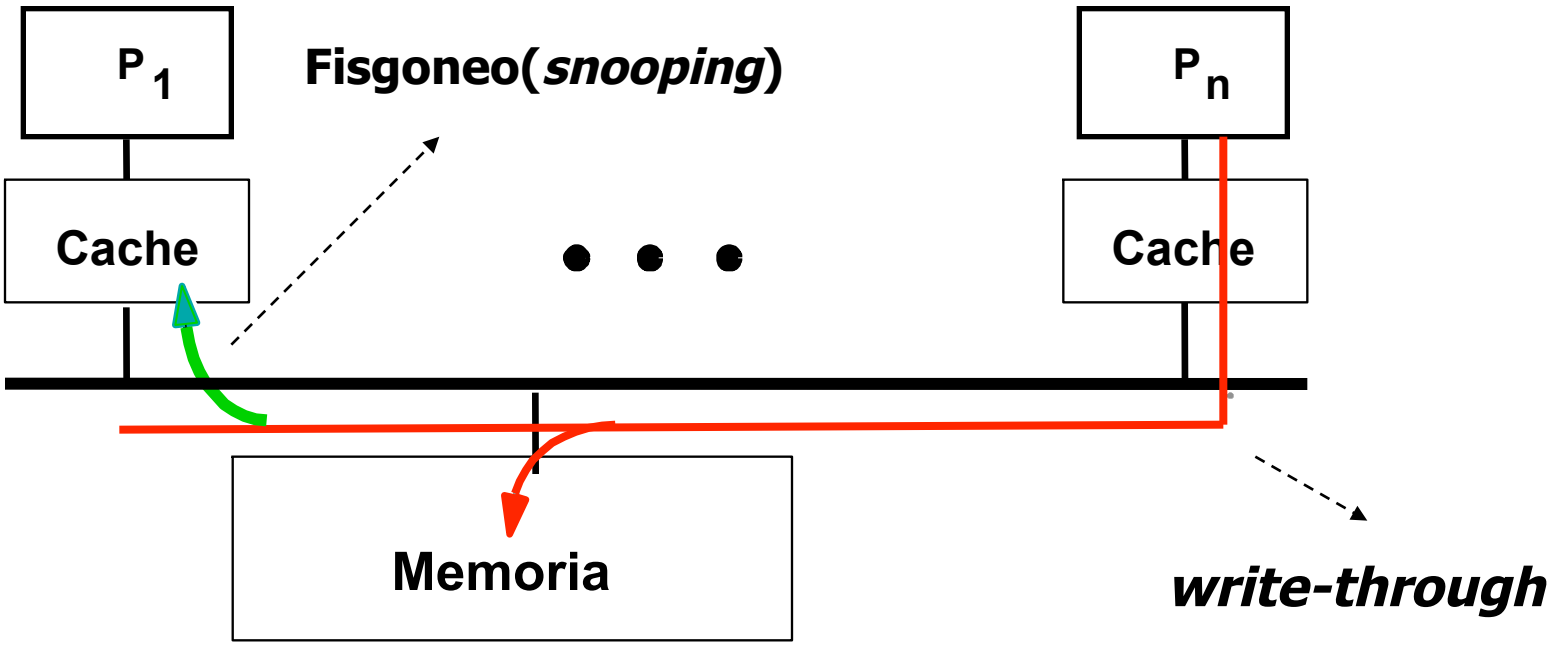
Índice (IV)

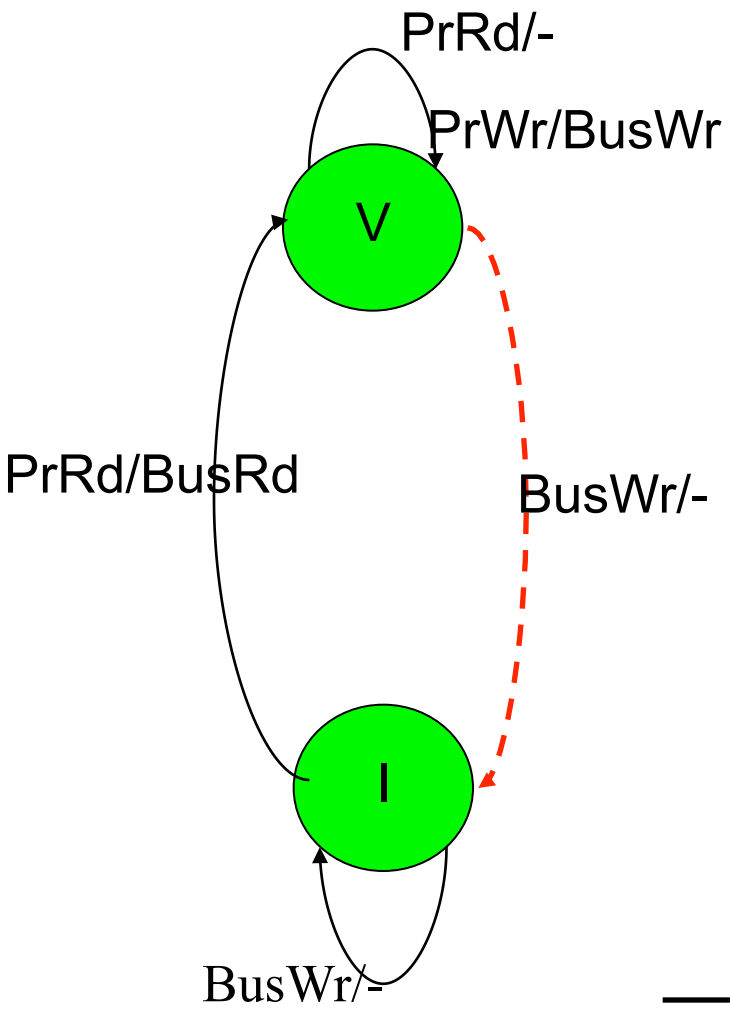
5. Coherencia de caches
   - Implementaciones
     - Hardware
       - *Snoopy*
       - Directorios
     - Software

C. CC: *snoopy: write-through*

- **Hw.s.1)** Primer intento: ***write-through***
  - Cada procesador utiliza *write-through* como política de actualización de su cache: se difunden todas las escrituras.
  - No hacen falta nuevos estados para cada bloque: Válido/ Inválido.
  - Una escritura hará que se invaliden/actualicen todas las otras copias.

C. CC: *snoopy: write-through*

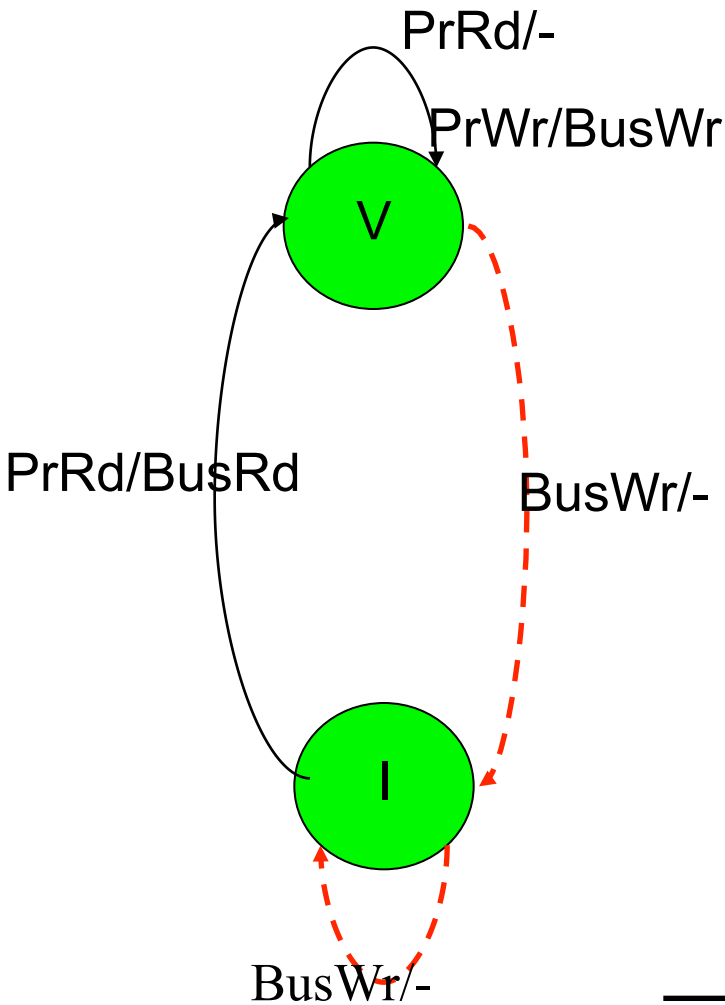




- ***Un diagrama de transición por bloque de cache.***
- Estados de un bloque: V:válido, I:inválido
  - V: el bloque contiene una copia correcta de la memoria.
  - I: el bloque no está en cache.
- Notación a/b
  - a : evento.
  - b: acción asociada al evento.
- eventos
  - Procesador puede Rd/Wr de/a un bloque.
  - Bus puede Rd/Wr de/a un bloque.

——— Acción inicada por el procesador.

– – – – – Acción iniciada por el mecanismo de *snooping*.



- La escritura invalida el resto de las caches (no afecta al estado local del bloque)
  - Se permite múltiples lecturas del bloque.
  - Escritura invalida las réplicas.
- Implementation
  - Bits de estado (implementados en hardware) associados a cada bloque almacenado en la cache.

——— Acción iniciada por el procesador.

– – – – – Acción iniciada por el mecanismo de snooping.

C. CC: *snoopy: write-through*

- **Inconvenientes**:
  - Consume mucho ancho de banda: todas las escrituras van a la memoria y al bus.
  - No se suele utilizar esta política en MP de Memoria Compartida.
- **Ventajas**:
  - Fácil de diseñar e implementar.
- Como conclusión, parece obvia la necesidad de utilizar una política de actualización *write-back* (*copy-back*) en cada procesador:
  - Las escrituras no llegan al bus: menor consumo de ancho de banda.
  - En principio, parece más difícil garantizar la coherencia que con *write-through*.
  - Los protocolos serán mucho más sofisticados: más difíciles de diseñar y de implementar.

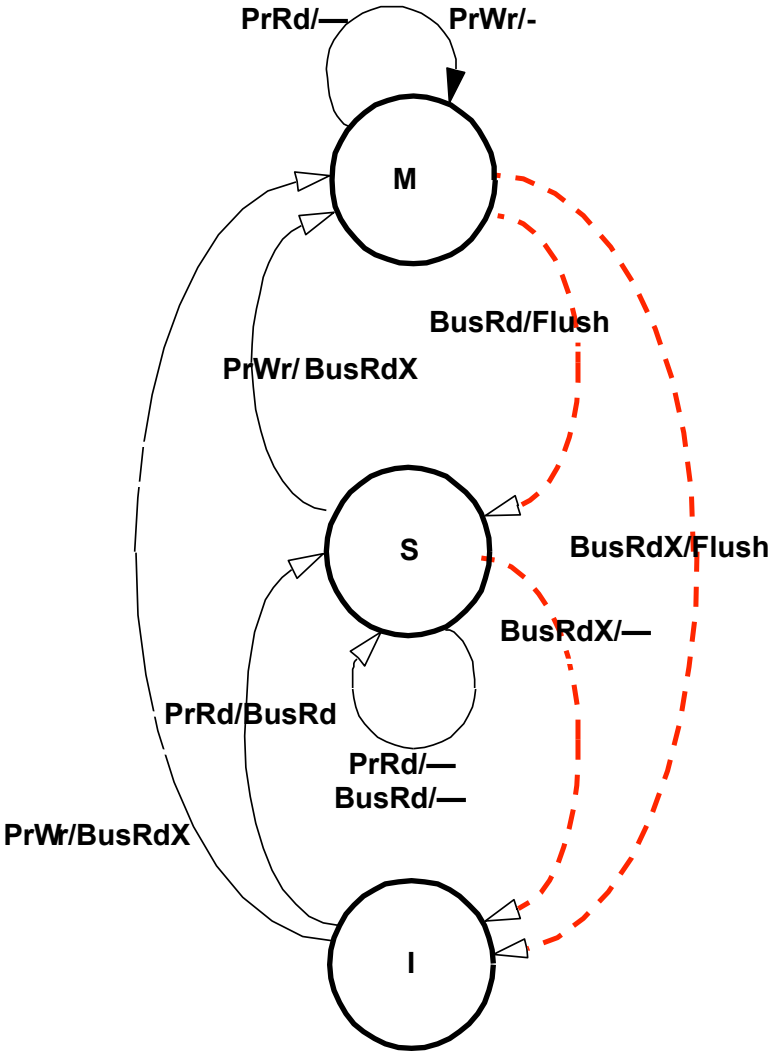
Procotolos Write-Back Snoopy

- No hay necesidad de cambiar el procesador, memoria principal, cache …
  - Requiere de una extensión del controlador de cache (no sólo estados V y I) y añadir mecanismos de serialización en el acceso al bus.
- Dirty state ahora también referencia una pertenencia exclusiva del bloque.
  - Exclusive: la copia válida del bloque está sólo en caché (la memoria principal también puede tenerlo.)
  - Owner: propietario de la copia válida y responsable de proveerlo si existen peticiones del bloque.
- Dos tipos de protocolos:
  - Basados en la invalidación.
  - Basados en la actualización.

Protocolo de invalidación básico MSI Write-back

- Estados:
  - Invalid (I)
  - Shared (S): una o varias copias del bloque.
  - Dirty or Modified (M): sólo una.
- Eventos del procesador:
  - PrRd (read)
  - PrWr (write)
- Transacciones del procesadores:
  - BusRd (read): solicita una copia del bloque sin modificarla.
  - BusRdX (read exclusive): solicita una copia para modificarla.
  - BusWB (write back): actualiza la memoria.
- Acciones:
  - Actualiza el estado, realiza la transacción a través del bus y vuelca el valor al bus.

State Transition Diagram

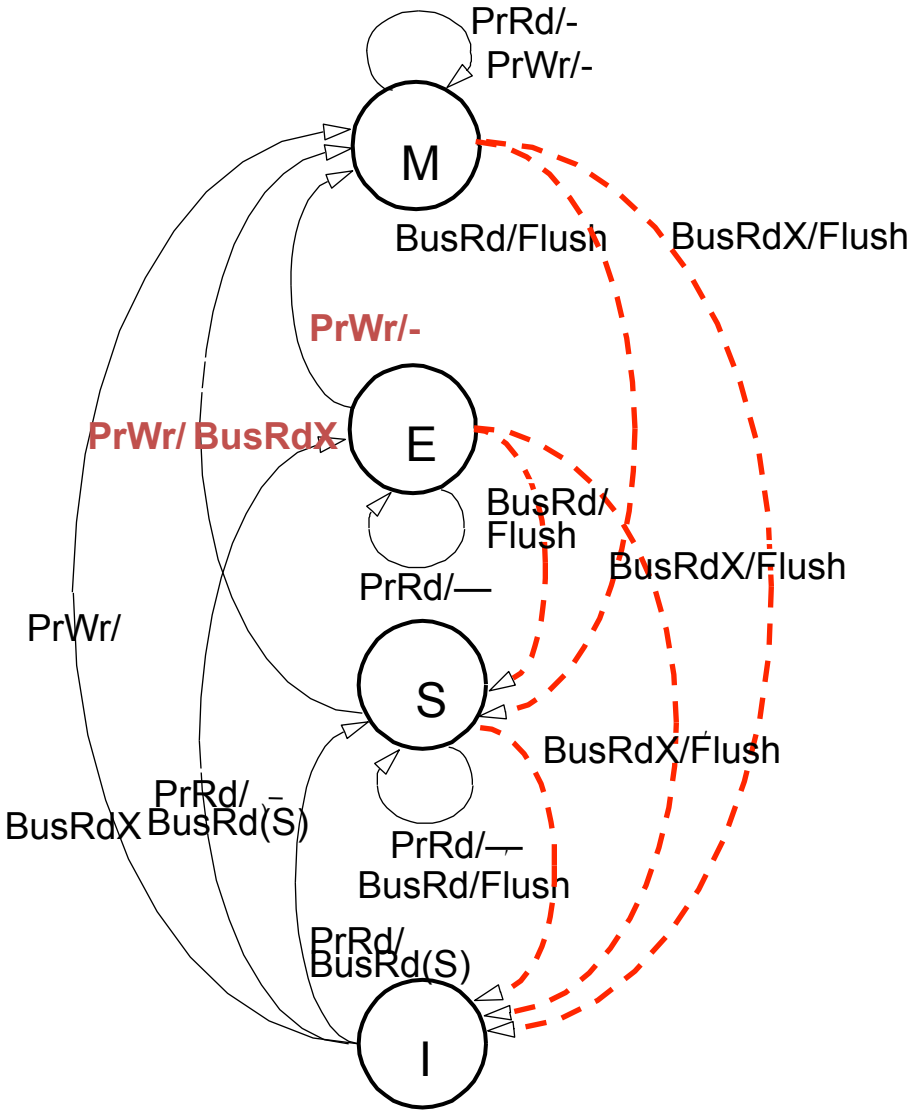


- No se muestran ni reemplazos ni write backs.
- Rd/Wr en estado M y Rd en estado S no causan transacciones al bus.
- Rd/Wr en estado I causan dos transacciones al bus.
- Wr en estado S causa dos transacciones de bus:
  - Dato pasa a modificado.
  - Se puede evitar la transición porque ya tiene el último dato: se puede usar actualizar (BusUpgr) en lugar de BusRdX

Protocolo de invalidación MESI (4-estados)

- Problema con protocolo MSI
  - Leer y modificar los datos son dos transacciones de bus incluso si no se comparten.
    - Esto sucede incluso en un programa secuencial!.
    - BusRd (I->S) seguido por BusRdX o BusUpgr (S->M)
- Solución: añadir estado *exclusive*. Si el bloque reside sólo en la caché local, entonces no se modifica. Se escribe localmente sin transacción de bus.
- Estados:
  - invalid
  - exclusive o *exclusive-clean* (existe una única copia en cache pero no es modificada)
  - shared (dos o más cachés tienen copias)
  - modified (dirty)

Diagrama de transición MESI



- Acierto: no hay transacción en bus dentro del estado E.
  - ¿Cuándo I cambia a E y cuándo cambia a S?
  - I -> E en PrRd si ningún otro procesador tiene una copia.
  - I -> S en caso contrario.
- S señal adicional de bus.
- BusRd(S): con señal BusRd, si un procesador mantiene el bloque lo asegura poniendo la señal a valor 1.

C. CC.: directorios

- Implementaciones con Directorios:
  - Se utilizan en MP en los que la red de interconexión no es un bus: CC-NUMAs.
  - Idea básica:
    - Se mantiene siempre anotado dónde existen copias de los bloques y en qué estado se encuentran.
    - *multicast*: operaciones "pro-coherencia" dirigidos exclusivamente a las caches que tengan copia del bloque: G*ossip vs. Snooping*.
  - Variación:
    - Jerarquías: *snooping* + directorios: redes de MP (biprocesadores) basados en buses; jerarquías de directorios.
  - Implementación:
    - Muchas posibilidades:

      ¿RENDIMIENTO vs. TAMAÑO DIRECTORIO vs. 'ESCALABILIDAD'?

C. CC.: directorios

RENDIMIENTO = grado de "contención" + Vel. de búsqueda

- Propuestas:
  - A) Directorios completos (*full-map*)
  - B) Directorios limitados (*limited*)
  - C) Directorios encadenados (*chained*)

↓ **'escalabilidad'**

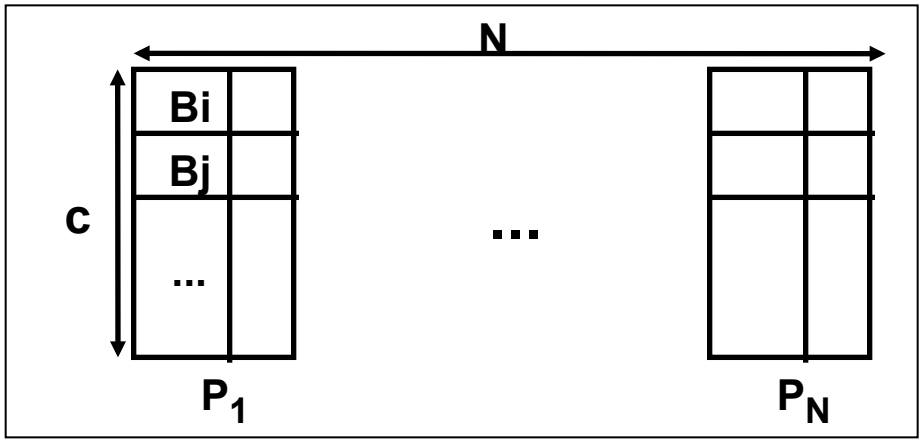
- **1.d.A. Directorios completos:**
  - Idea: mantienen información sobre todas las caches en el sistema.
  - Inconveniente: no 'escalabilidad'.
  - Propuestas:
    - » A.1. Tang: un único directorio centralizado conteniendo copia de cada directorio local: bit de *dirty* por bloque: búsqueda exhaustiva.
    - » A.2. Censier: un único directorio centralizado: bit de *dirty* por bloque: un "bit de presencia" para cada bloque: búsqueda fácil.
    - » A.3. Stenström: similar a Censier pero distribuido por las caches, en cada cache sólo existe información de sus bloques.

C. CC.: dirs.: *full-map: Tang*

Comparación propuestas dir. completos (*full-map*):

L: tamaño bloque; C: capacidad cache; M: ídem M; c = C/L; m = M/L; N: # proces.

1.d.A.1: Tang



I) Rendimiento:

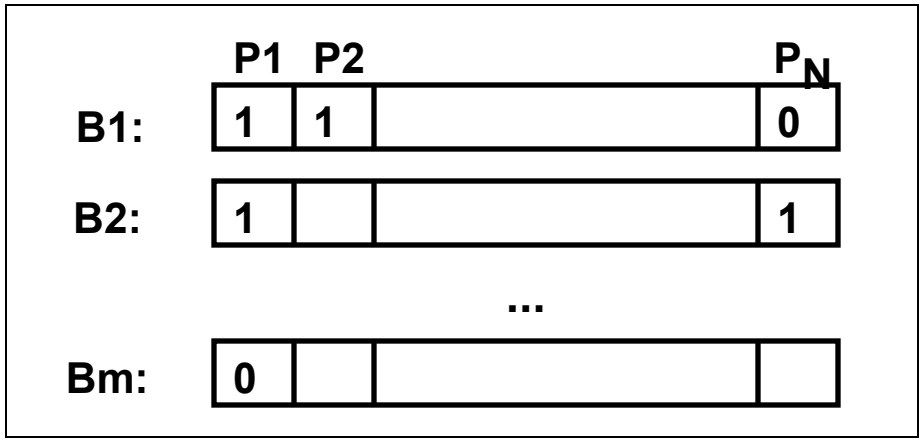
. mucha 'contención'

. búsqueda exhaustiva: lenta

II) Tamaño:

N * c * (tag+#bits de estado bloque): ∝ c*N

1.d.A.2: Censier



I) Rendimiento:

. mucha 'contención'

. búsqueda rápida

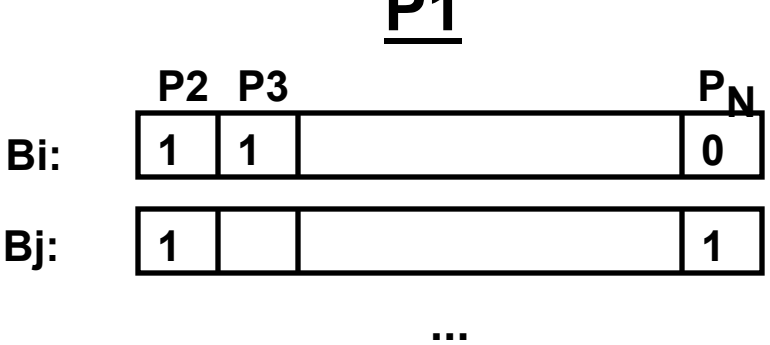
II) Tamaño:

N * m*(#bits de estado bloque): ∝ N*m

1.d.A.3: Stenström

Información sólo de los bloques que tiene copiados.

Cuando hay fallo, se consulta un dir. completo en M.



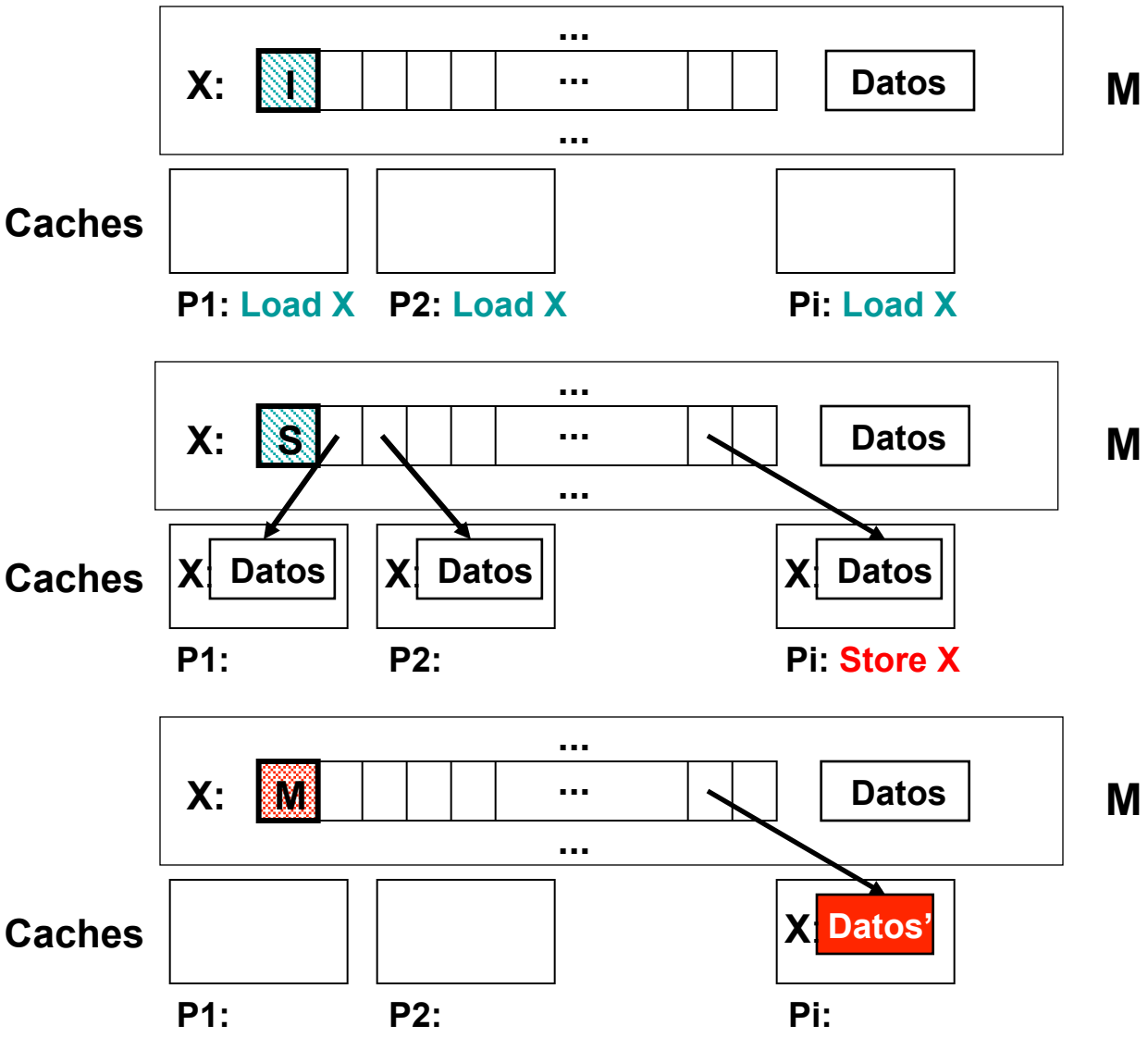
I) Rendimiento:

. poca 'contención'

. búsqueda rápida

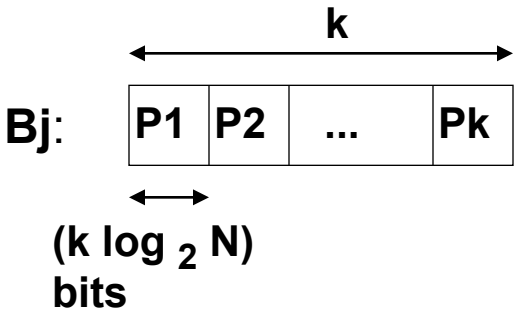
II) Tamaño:

(N * c) * N: ∝ c


X:
I
Datos
M
Caches
P1: Load X
P2: Load X
Pi: Load X
X:
S
Datos
M
Caches
X Datos
X Datos
X Datos
P1:
P2:
Pi: Store X
X:
M
Datos
M
Caches
X Datos'
P1:
P2:
Pi:

C. CC.: dirs.: limitados

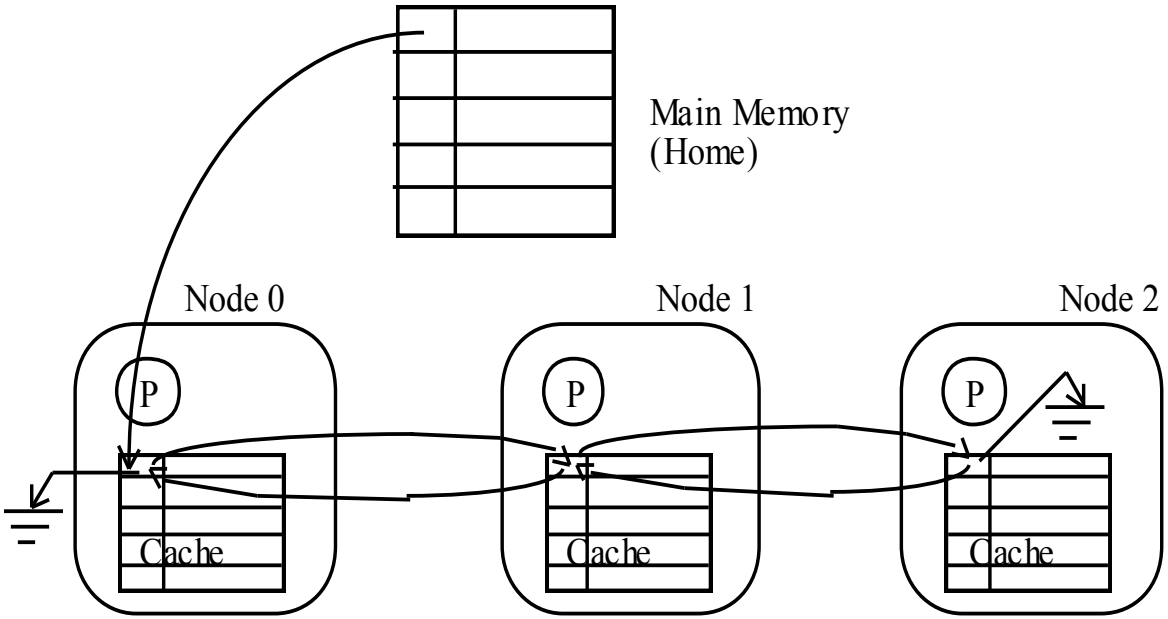
- **1.d.B. Directorios limitados**:
  - Idea: mantienen información sobre un número 'k' de caches, k < N: si hay más de 'k' copias:
    - no permitir más que 'k' copias
    - hacer "*broadcasting*"
  - Ventaja: más 'escalable'
  - Inconvenientes:
    - el éxito dependerá del grado de "sharing" (¿más de 'k' copias?)
  - Consideraciones de rendimiento:



C. CC.: dirs.: encadenados

- **1.d.C. Directorios encadenados:**
  - Idea: para cada bloque copiado, hay un "puntero" al "siguiente" procesador que también tiene copia
  - Ventaja: completamente 'escalable'
  - Inconvenientes:
    - El éxito dependerá del grado de "sharing": se necesita recorrer toda la "lista" de directorios
    - Estándar ISO/ANSI/IEEE P1596, Scalable Coherent Interface (SCI): 1992: HP/Convex Exemplar :
      - www.SCIzzL.com

C. CC.: dirs.: encadenados



C. CC.: bibliografía

- Libros "de texto":
  - El Culler *et al*. y el Sima *et al*.
- Libros específicos: [compilaciones de "papers"]
  - M. Tomásevic, V. Milutinovic: *The Cache Coherence Problem in Shared-Memory Multiprocessors:* ***Hardware Solutions***, IEEE Computer Society Press, 1993
  - I. Tartalja, V. Milutinovic: *The Cache Coherence Problem in Shared-Memory Multiprocessors:* ***Software Solutions***, IEEE Computer Society Press, 1996