Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Tema 13 – Coherencia

Multiprocesadores

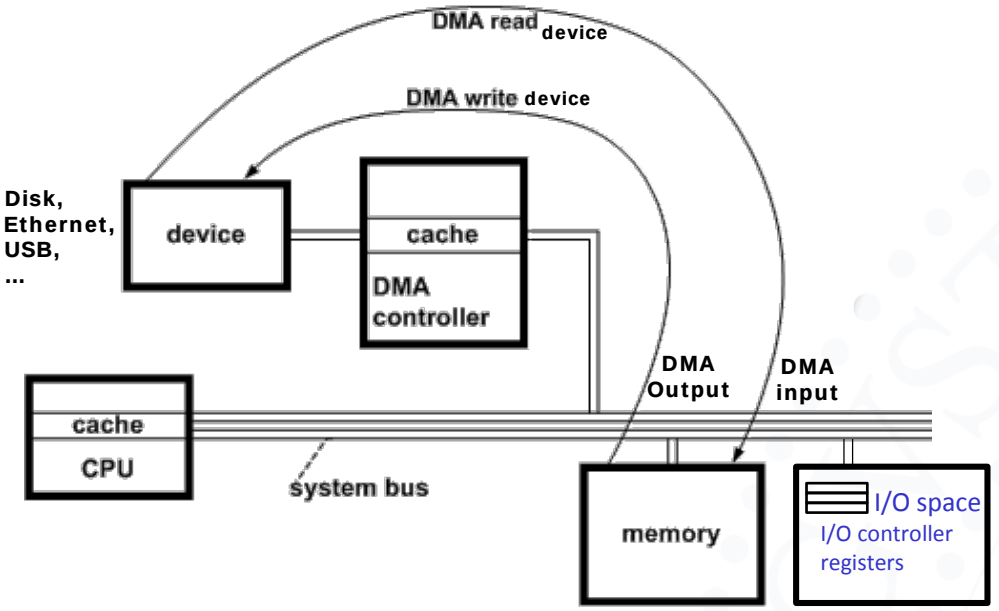
3er curso, Grado Ingeniería Informática
Especialidad Ingeniería Computadores

V. Viñals y J. Alastruey
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

Índice

- El problema de la coherencia
  - sistema, multiprocesador, cache multinivel, mas ejemplos
  - escritura retardada e inmediata (animaciones)
- El modelo de memoria
  - consistencia secuencial, pros y contras
  - una definición de coherencia
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunos transacciones de ejemplo
  - protocolo sencillo de directorio

El problema de la coherencia: sistema



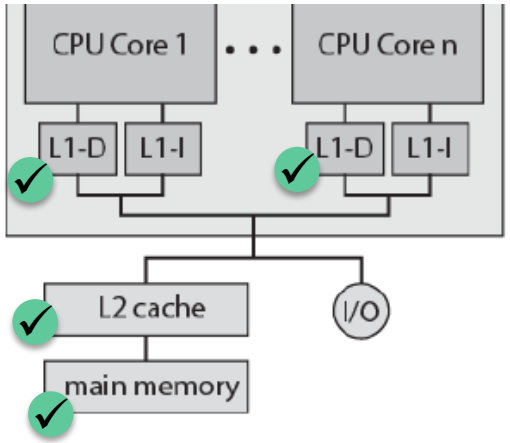
- Incluso con un solo procesador, un bloque puede estar en varios almacenes: cache(s) y memoria
- Ej.: transferencia DMA fallo página
  - El SO programa la transferencia, conoce las páginas que se mueven …

- Opción 1: páginas "no cacheables", solo residen en memoria principal
  - bit en la Tabla de Páginas (y en los TLBs)
  - útil para registros de E/S mapeados
  - in/out procesador = fallo cache
- Opción 2: comados SO → cache
  - antes de una Entrada (DMA input):
    - invalidar bloques de esa página en Cache
  - antes de una Salida
    - poner al día bloques de esa página en Mp
  - Antes de un reemplazo de página: Flush
    1. poner al día Mp, SO programa: Mc > Mp(wB, Xsucios ∈ página)
    2. invalidar página en Mc, SO programa: Mc(inv, X ∈ página)

El problema de la coherencia: multiprocesador

- Las caches mas próximas al procesador son privadas
- Pueden existir varias copias del mismo bloque **x** en varias caches privadas
- Actualizaciones locales ?
  - conducen a un estado incoherente:
    una palabra x' del bloque tienen valores diferentes en caches diferentes
  - el problema se da con cualquier política de escritura en acierto
    - escritura inmediata (*write-through*)
    - escritura retardada (*write-back = copy-back*)

El problema de la coherencia: cache multinivel

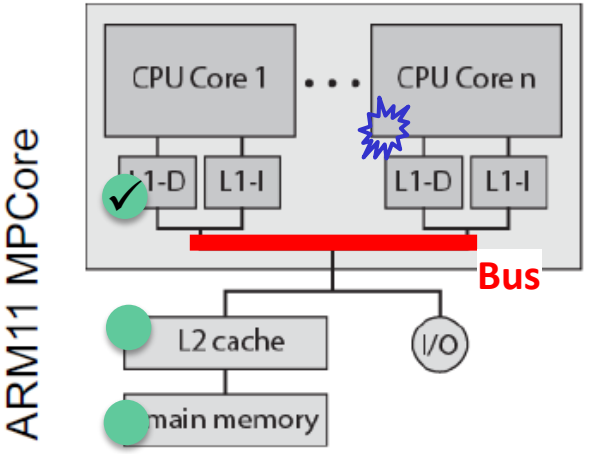


- El problema persiste
  al añadir un nivel L2 de cache compartida:
  - Un bloque x (✓) puede estar
    - en varias L1-D privadas
    - en L2 compartida
    - y por supuesto en memoria principal
  - ¿ Qué ocurre si dos cores
    quieren escribir al mismo tiempo la variable x'?
  - ¿ Qué valor entregar en las siguientes lecturas ?

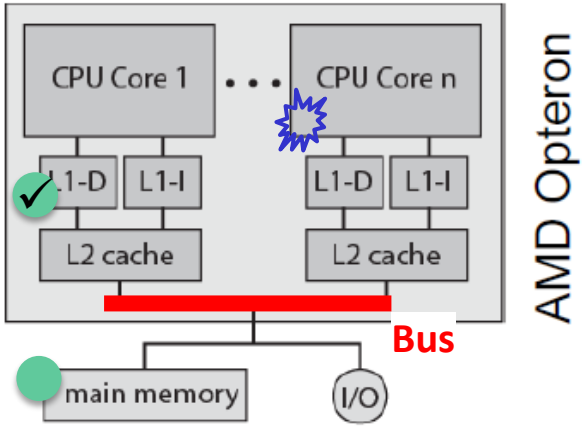
El problema de la coherencia: ejemplos caches Copy Back

BLOQUE **x**

- ✓ ultima escritura
- copia antigua

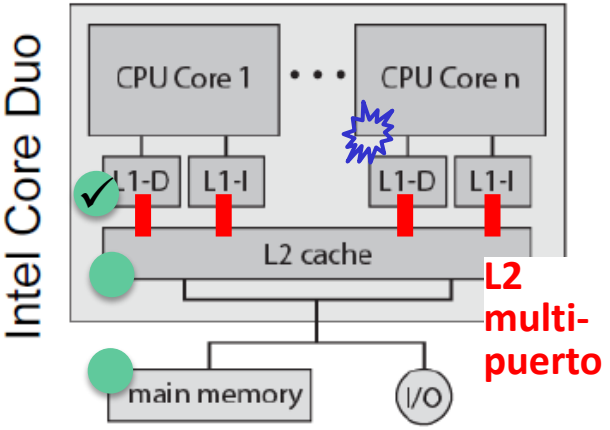


L1s privadas, L2 compartida e inclusiva

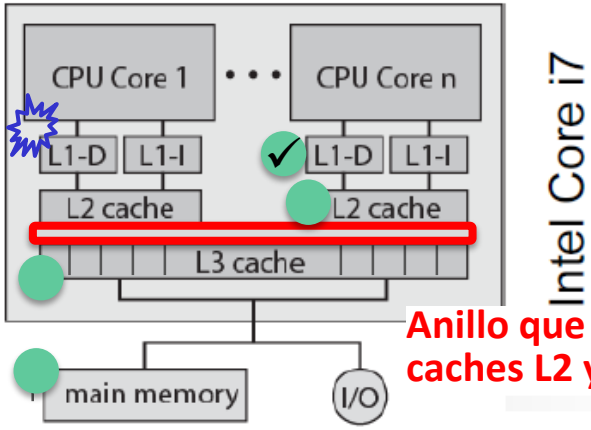


L1s y L2s privadas (y en exclusión)

¿ que ocurre con un fallo en lectura ?

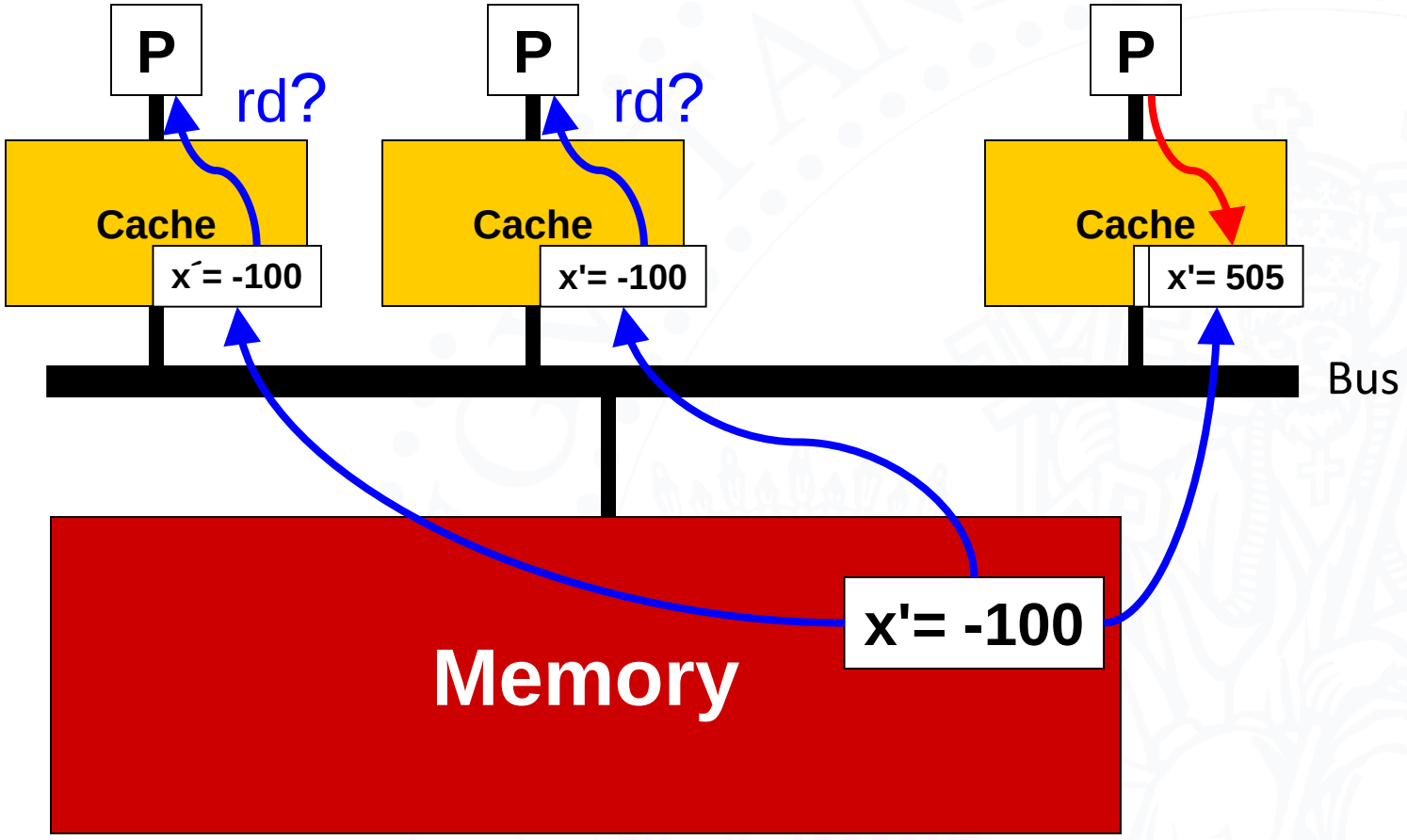


L1s privadas, L2 compartida e inclusiva

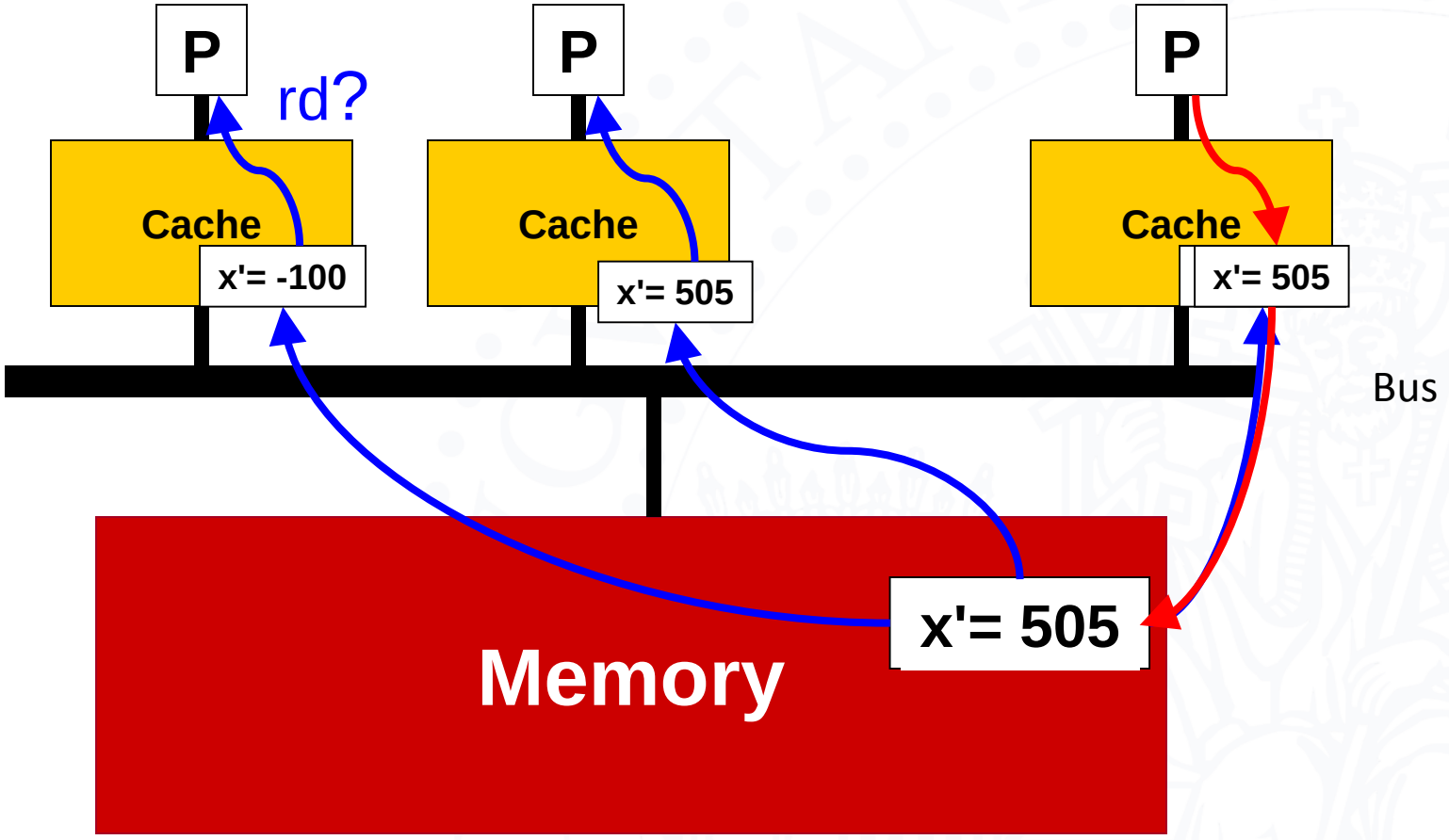


L1s y L2s privadas, L3 compartida, todas inclusivas

Problema coherencia, escritura retardada



Problema coherencia, escritura inmediata



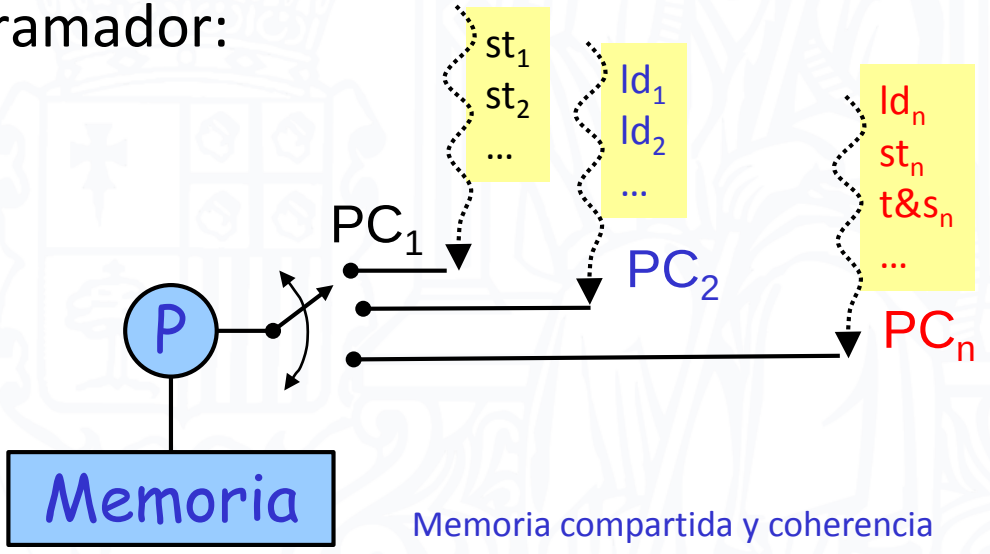
Índice

- El problema de la coherencia
  - sistema, multiprocesador, cache multinivel, mas ejemplos
  - escritura retardada e inmediata (animaciones)
- El modelo de memoria
  - consistencia secuencial, pros y contras
  - una definición de coherencia
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunos transacciones de ejemplo
  - protocolo sencillo de directorio

Modelo de memoria

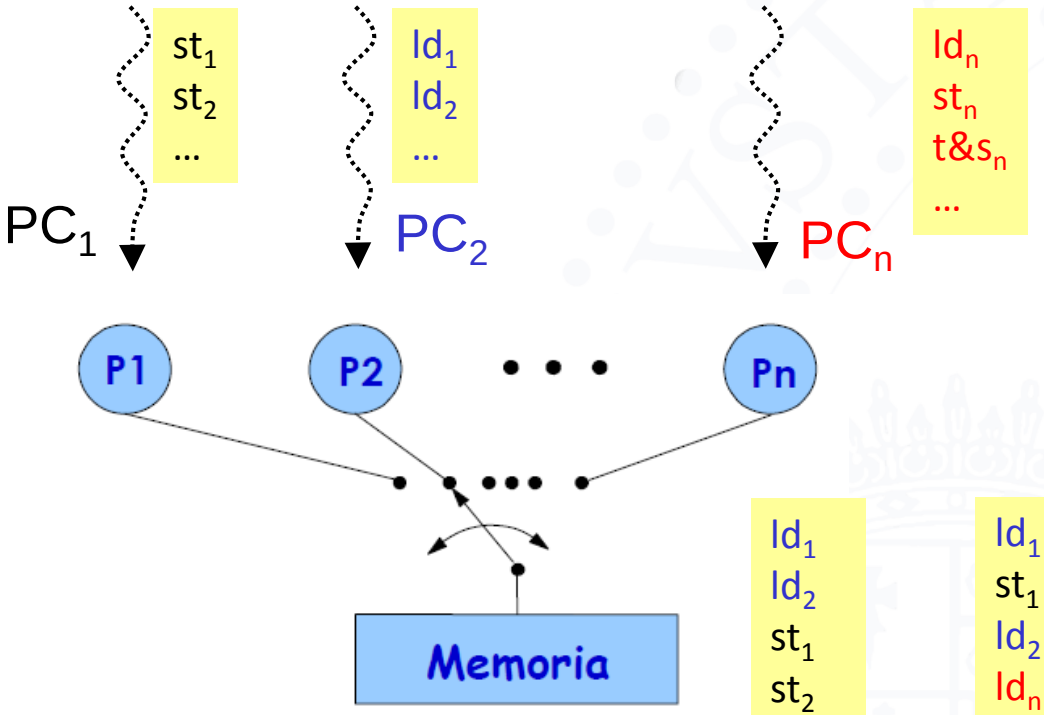
- Modelo de memoria ? → extensión del uniprocesador
  - procesos independientes
    usan regiones de memoria principal no solapadas (traducción de @)
  - procesos con hilos (*threads*) usan la memoria para
    compartir código,
    comunicar valores y sincronizar
    - Instrucciones ld, st
    - Instrucciones atómicas leer - [modificar] – escribir
    - Instrucciones encadenadas load linked – store conditional

- Modelo sencillo para el programador:
  Consistencia Secuencial
  - "similar" al uniprocesador
    que multiplexa varios **procesos**
    (tiempo compartido),
    donde cambiamos
    proceso por procesador



Consistencia Secuencial

- Mezcla arbitraria de los flujos individuales de memoria, manteniendo el orden de programa de cada flujo



El modelo de consistencia se especifica en la ALMa

ld1 ld2 st1 st2 ldn stn t&sn ...

o ...

ld1 st1 ld2 ldn stn t&sn st2 ...

o ...

st1 ldn stn ld1 t&sn ld2 st2 ...

P1, P2, … Pn **ven** el **mismo** orden global a las diferentes posiciones de memoria

Consistencia en procesadores comerciales

- Consistencia Secuencial
  - **procesadores MIPS R10K-12K-16K**
    MIPS-IV ISA, 1996-2004

- SPARC V9 ISA, define tres modelos de consistencia:
  - *Total Store Order (TSO)*, *Partial Store Order (PSO)*,
    and *Relaxed Memory Order (RMO)*
- TSO
  - *Stores* en orden de programa en cada thread
  - *Loads @x* (+ jóvenes) pueden adelantar a stores @y (+ viejos)
  - Permite *buffering* de *stores*
  - **SPARC Architecture 2015 processors** (ejecutan bien programas PSO y RMO)
  - **~ x86 processors** (Intel, AMD), aunque no existe un documento formal …
- **IBM Power** y **ARM** definen modelos relajados de memoria,
  "similares" a RMO
  - hay que programar "vallas" o "barreras de memoria"
    (*fences, membar*) para conseguir ordenaciones globales

Pros y contras

- Consistencia secuencial
  es un buen modelo para programar
- Pero para aumentar el rendimiento:
  - el compilador optimiza
  - las instrucciones se ejecutan fuera de orden
  - los stores se almacenan temporalmente (*buffering*)
  - existen niveles de cache y redes de interconexión

  } Desorden !!
- El desorden de lecturas y escrituras
  puede no respetar la consistencia secuencial
  - no es trivial "fabricar" un orden **global** de lecturas y escrituras **único** para todos los procesadores …

Ejercicio CS (1)

Can both r1 and r2 be set to 0?	
Core C1	**Core C2**
/* Initially, x = 0 & y = 0*/	
S1: Store x = NEW; L1: Load r1 = y;	S2: Store y = NEW; L2: Load r2 = x;

- Formar todos los posibles entrelazados que respetan CS
- Responder a la pregunta: ¿ r1 y r2 pueden ser 0 en los dos Cores tras la ejecución de los códigos ?

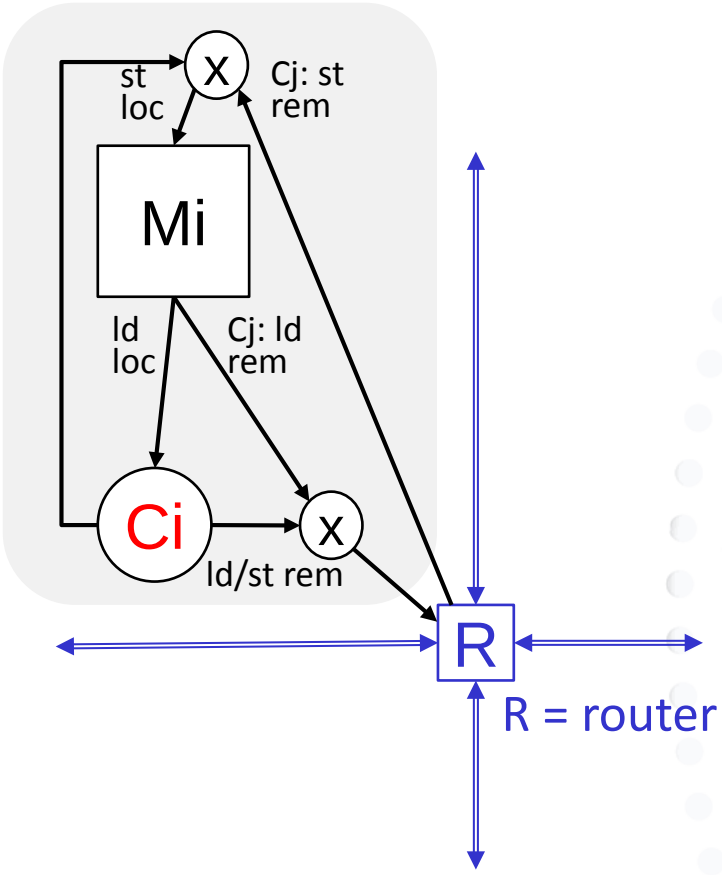
→ Inspired by Th. J. Dekker's Algorithm for ensuring mutual exclusion (early 60's)

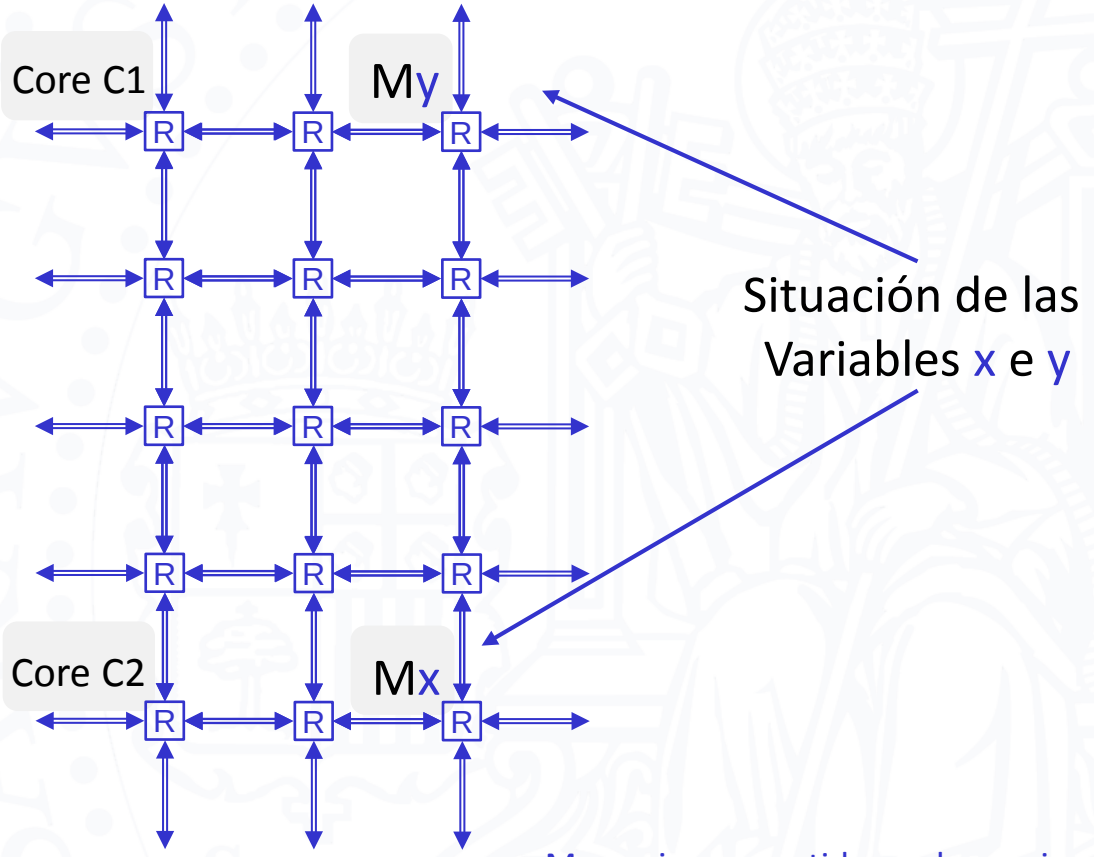
```
...
state[i] = interested;  /* declare interest */
while (state[j])  /* if the other one's interested */
{
   if (turn == j)  /* and it is its turn */
   {
      state[i] = uninterested; /* we stay uninterested */
      while (turn == j);  /* ...until our turn */
      state[i] = interested;
   }
}

< critical section >
turn = j;       /* forces alternation if both are interested */
state[i] = uninterested;   /* ask again if I want again */

 < code outside critical section >
…
```

D.J. SORIN, M.D. HILL, D.A. WOOD, A primer on memory consistency and cache coherence. Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2011.

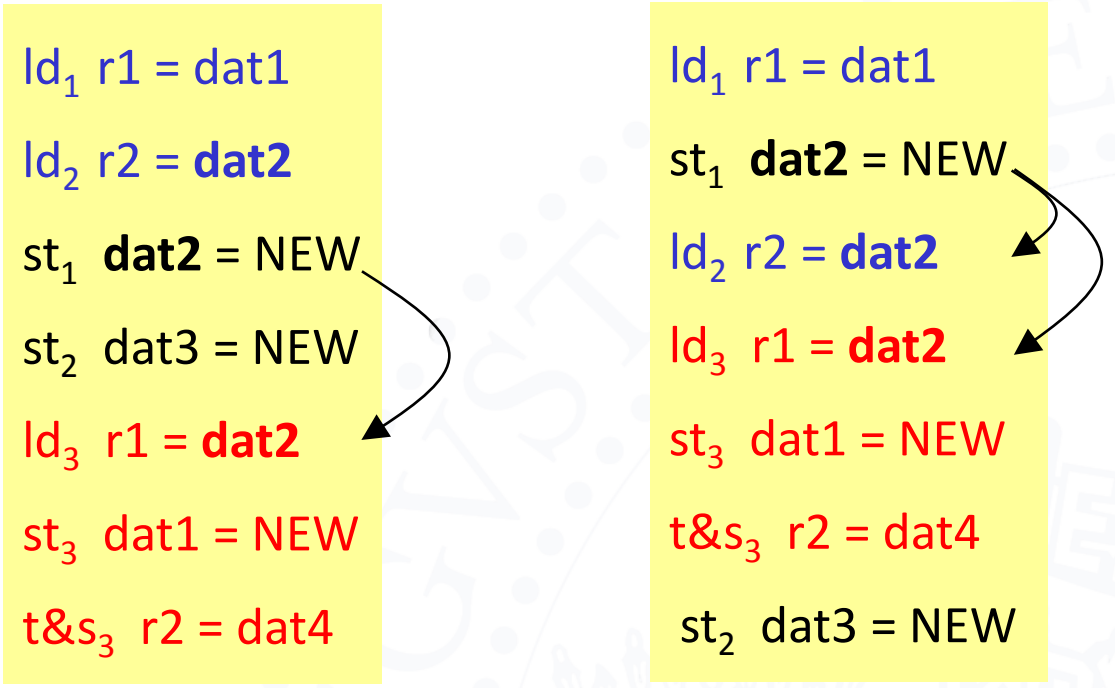
Ejercicio CS (2)

- Probar el código anterior en el multiprocesador de la figura
  - Cores en orden (Ci)
  - No hay caches, un trozo de memoria principal (Mi) en cada nodo



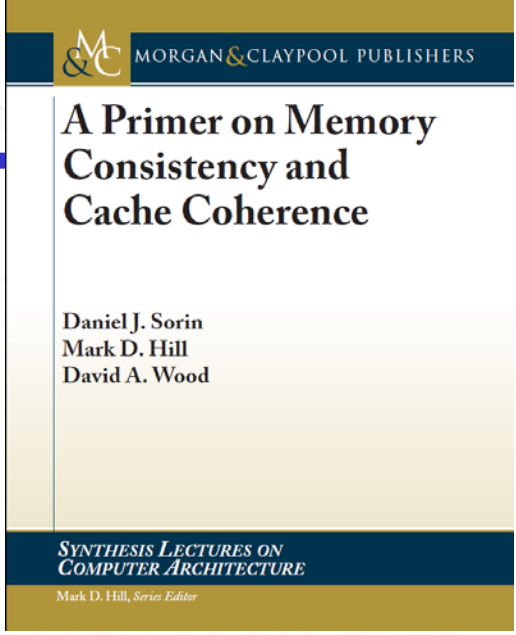


Coherencia y consistencia → dat_2



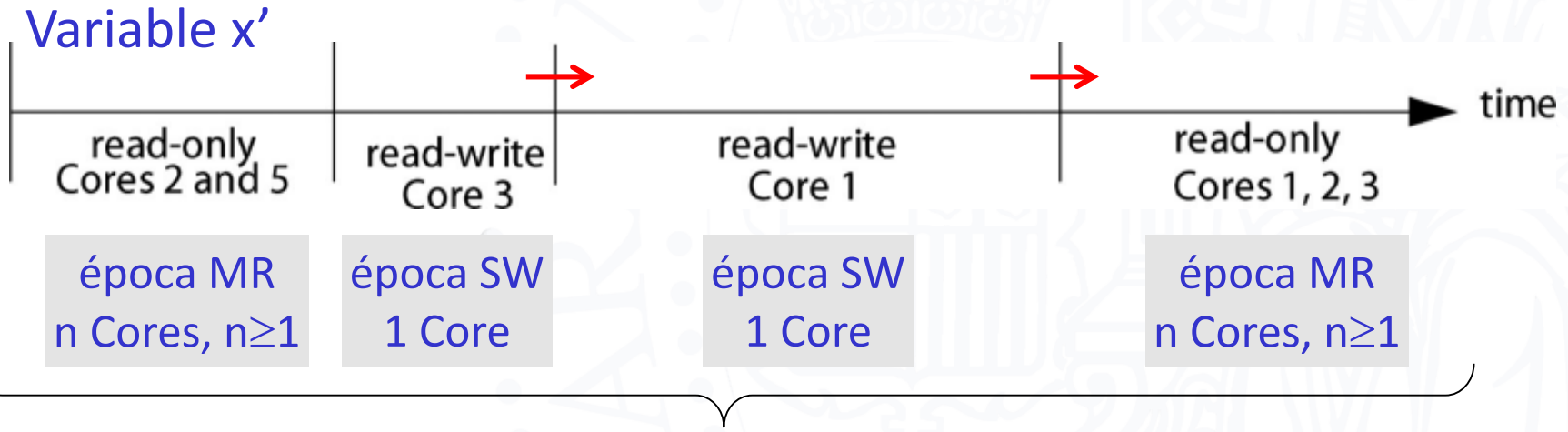
- el mecanismo de *coherencia* propaga el valor **NEW** a **dat2**
  - **st_1** → **ld_3** en la ejecución de la izquierda
  - **st_1** → (**ld_2** , **ld_3**) en la ejecución de la derecha
  - Claro, la relación productor-consumidor es indeterminista porque en este código hay *carreras de datos* (*data races*) p.e. varios accesos a **dat2**, y al menos uno de ellos es una escritura

Definición de coherencia



al leer una posición de memoria
se obtiene el valor "más reciente"
escrito en esa posición de memoria

- *Una* solución:
  - invariante *single-writer–multiple-reader* (SWMR)
  - una variable pasa por épocas SW y MR
  - el valor se propaga entre épocas consecutivas: SW → {SW, MR}



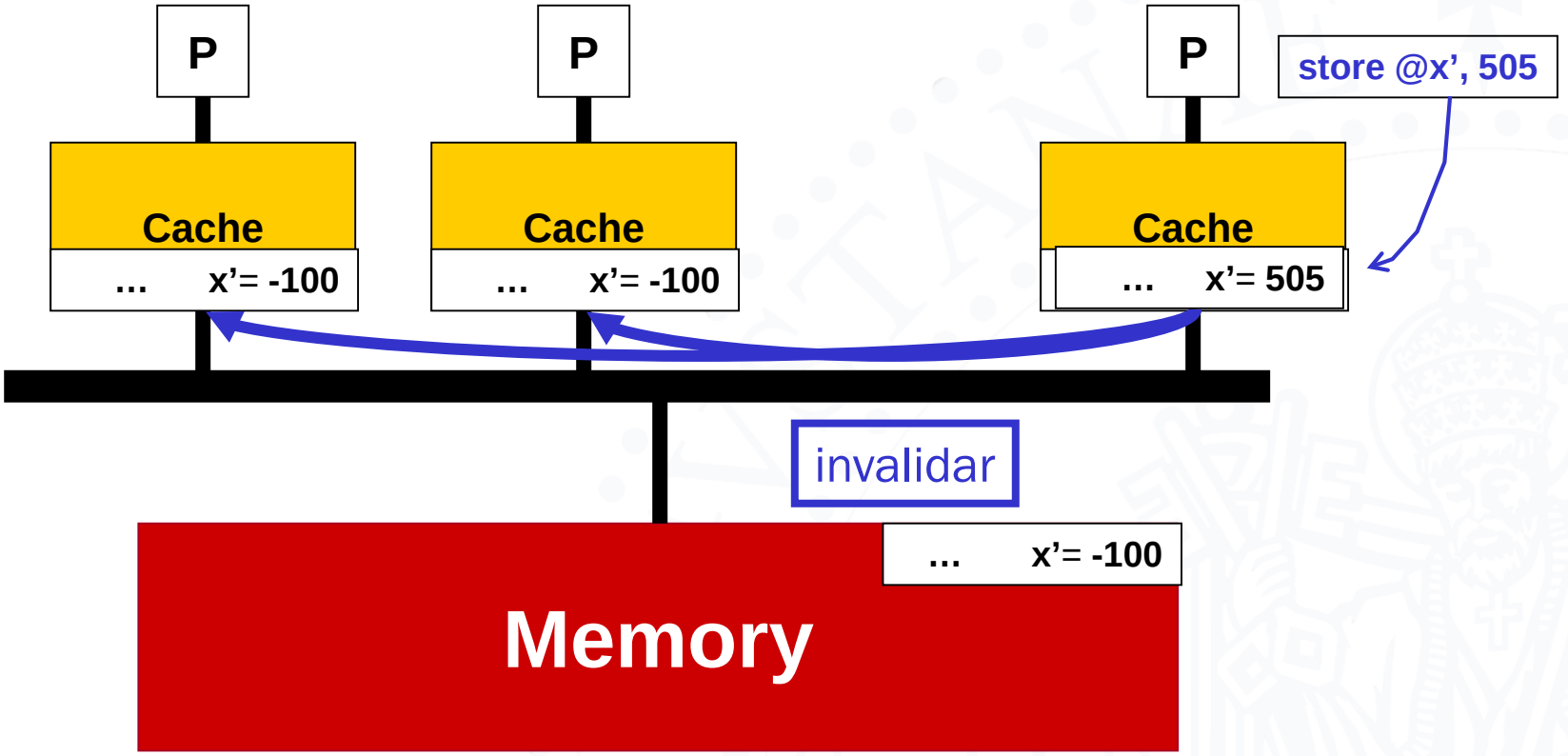
Índice

- El problema de la coherencia
  - sistema, multiprocesador, cache multinivel, mas ejemplos
  - escritura retardada e inmediata (animaciones)
- El modelo de memoria
  - consistencia secuencial, pros y contras
  - una definición de coherencia
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunos transacciones de ejemplo
  - protocolo sencillo de directorio

Protocolos de coherencia = trabajo extra en las escrituras

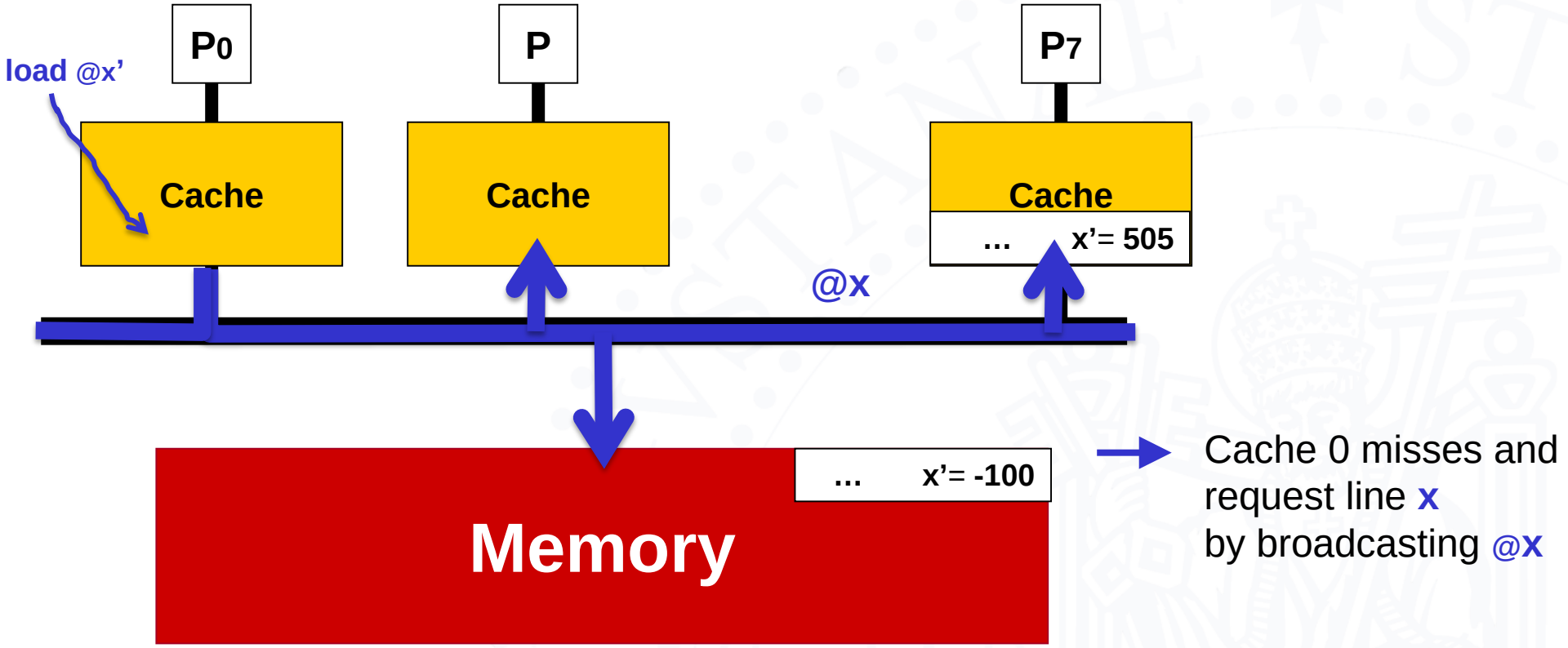
- Solución SWMR para propagar escrituras al cambiar de época: invalidar el bloque **x** en las caches privadas que tengan copia
  - el fallo posterior asegura la propagación
- Localizar las copias del bloque **x ?** → dos formas:
  - **difundir** dirección @x a todas las caches privadas
    … las que tienen ese bloque, reaccionan y lo invalidan
    - st @x´, val → difusión (@x) → los nodos con copia hacen **Mc – x**
  - **enviar invalidaciones de forma selectiva**
    manteniendo un **directorio** de "copias" de cada bloque.
    La invalidación se envía *primero* al directorio,
    después, el directorio reenvía *únicamente* a los nodos con copia
    - st @x´, val → enviar @x a DIR → DIR notifica a los nodos con copia
      → los nodos notificados hacen **Mc – x**

Invalidación + difusión en caches Copy-Back



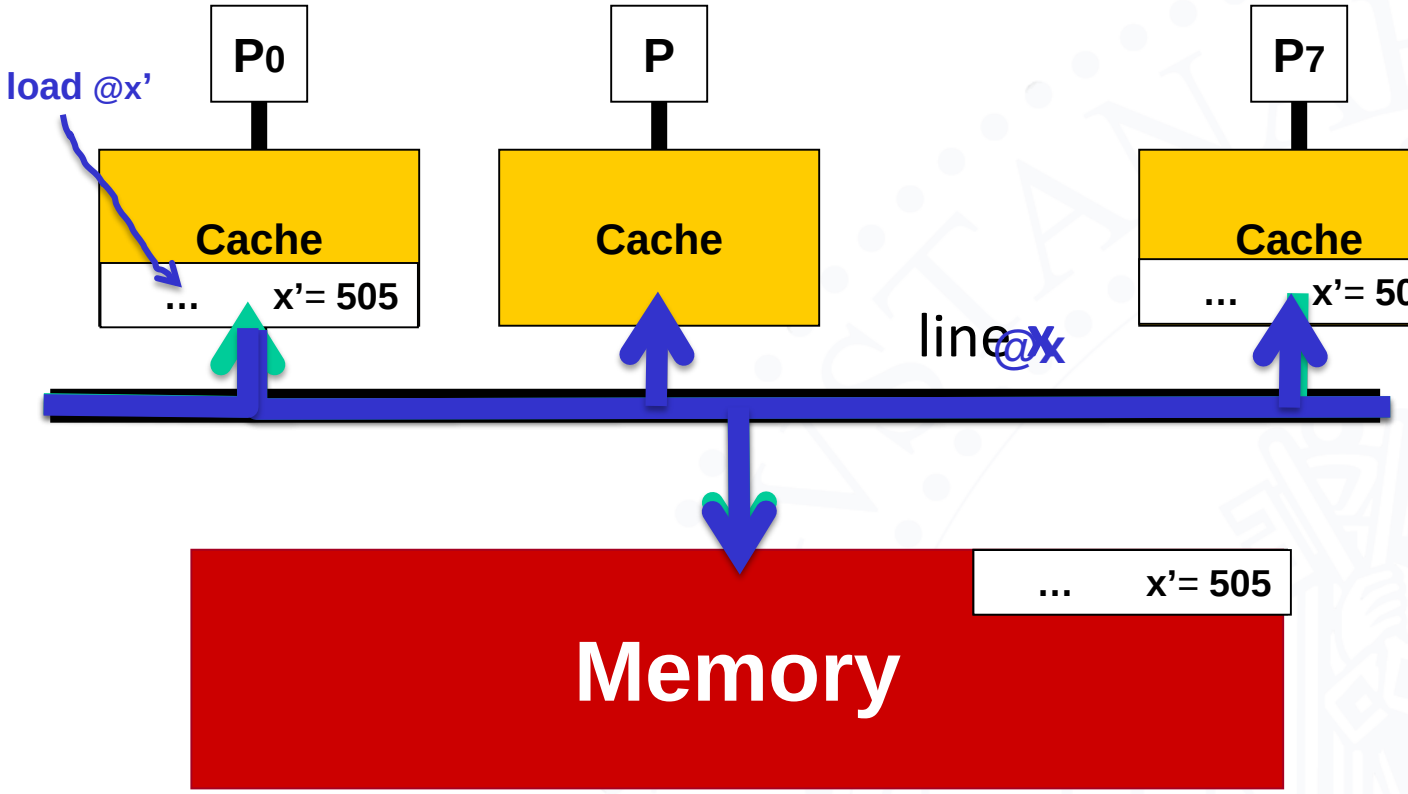
- Copy-back = escritura retardada = las escrituras no salen de las caches
  - se consigue guardando estado en cada bloque {Inválido, Limpio, Sucio}
- Con varias caches privadas, escribir = invalidar el resto de copias del bloque **x** (eliminar el grupo de compartición)

Invalidación + difusión en caches Copy-Back



- La propagación de escritura se consigue tras el fallo

Invalidación + difusión en caches Copy-Back



- La propagación de escritura se consigue tras el fallo

Cache 7 is snooping

- @x hits
- takes control of the bus
- sends the whole line **x** to Mem as well as Cache 0

Índice

- El problema de la coherencia
  - sistema, multiprocesador, cache multinivel, mas ejemplos
  - escritura retardada e inmediata (animaciones)
- El modelo de memoria
  - consistencia secuencial, pros y contras
  - una definición de coherencia
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunos transacciones de ejemplo
  - protocolo sencillo de directorio

Notación

- Items:
  - Bloque referenciado /expulsado: x / u
  - Palabra: minúscula con prima: x'
  - Dirección de: @ @x', @x
  - Bloque y dirección: mayúsculas X = < @x, x >
  - Palabra y dirección: mayúsculas X'= < @x', x' >
- Movimiento entre componentes de la jerarquía:

evento: comp_destino (comando, items)

rh	read hit
wh	write hit
rm	read miss
wm	write miss
rpl	replacement

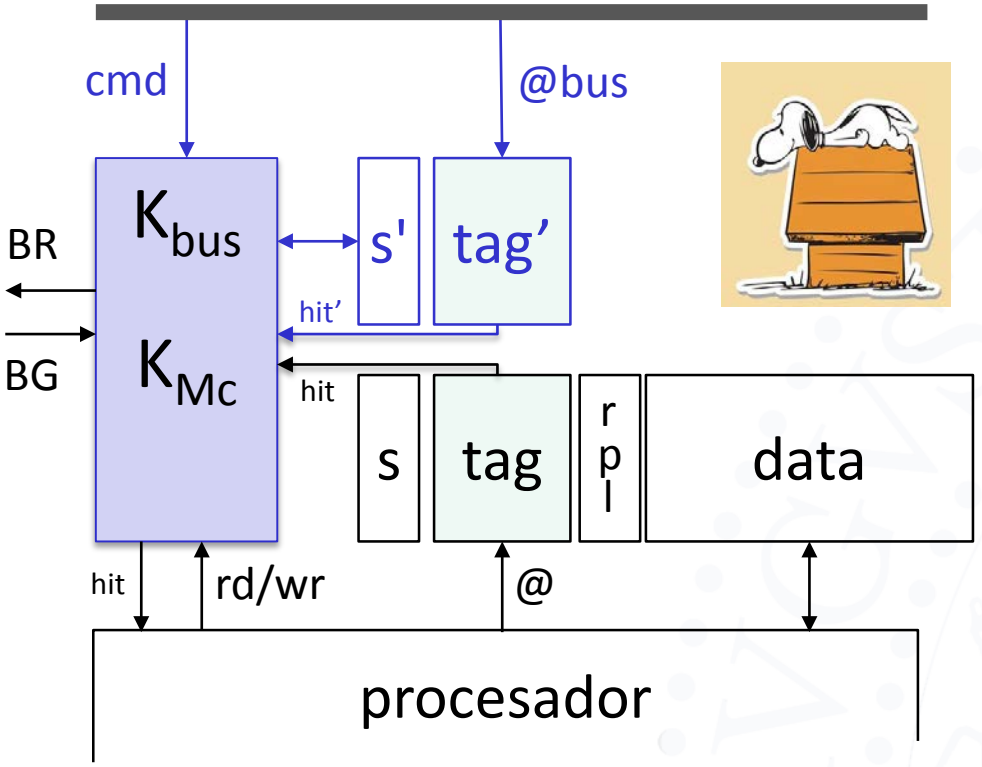
Mc
Mp
all

rB	read block
wB	write block
rW	read word
wW	write word
inv	invalidate block
rBinv	= rB + inv

X, @x, …

- Carga de bloques o palabras en un componente: operador "**+**"
  p.e. Mc **+** x; después de un fallo al bloque x
- Invalidación de bloque en una Mc: operador "**-**"
  p.e. Mc **-** u; al reemplazar el bloque u

Modelo de controlador de cache/coherencia



Kbus/KMc mira el estado de coherencia de cada bloque referenciado, cambia estados y ejecuta acciones

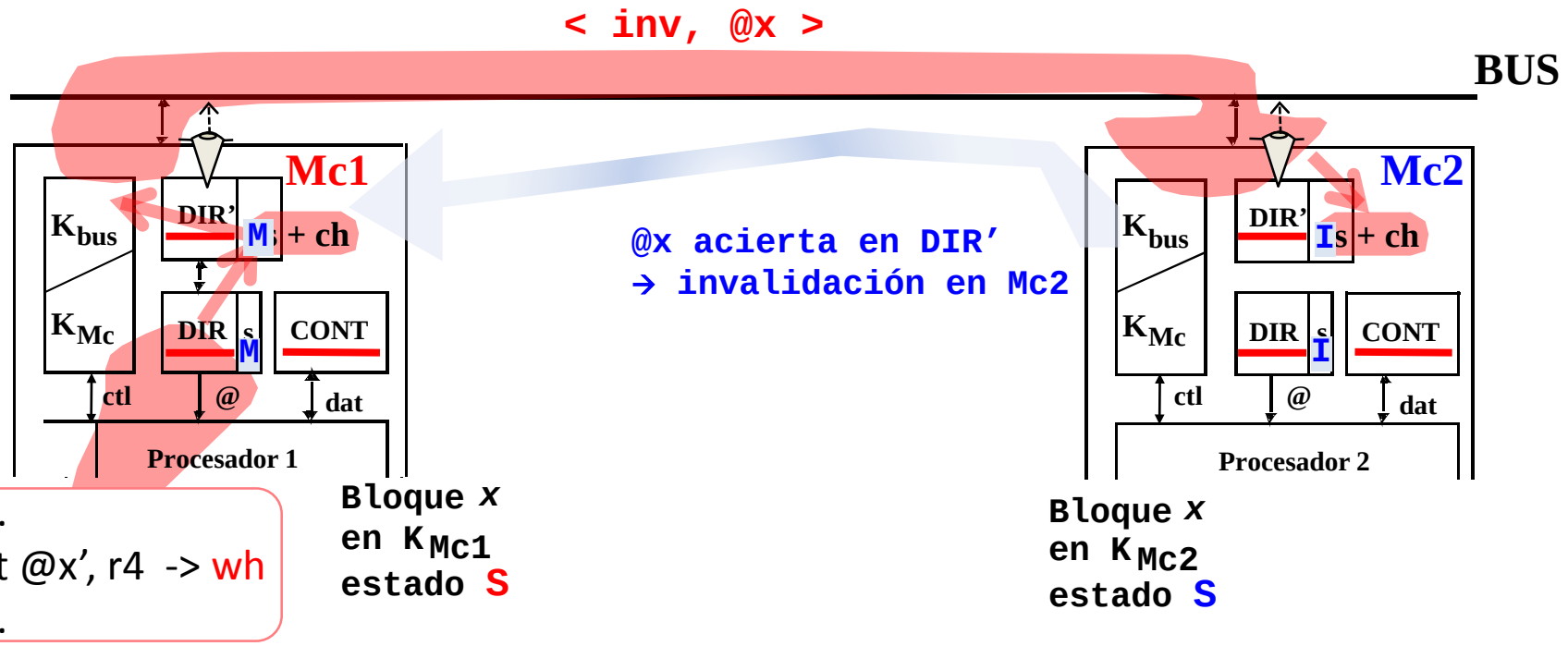
- Snoopy caches
- Duplicar estado y etiquetas
  - **s , s'** *bits de estado* que informan de las propiedades de coherencia de cada bloque
    - "ampliación" del estado de bloque en el uniprocesador
  - **tag'** *duplicado de las etiquetas* = nº conjuntos y asociatividad
- Observar las transacciones de bus es buscar en tag' la dirección de bus
  - En caso de acierto Kbus/KMc reacciona según el tipo de comando de bus

Protocolo MSI: invalidación + difusión en caches Copy-back

- Protocolo de coherencia **MSI**
  - **M = modified**
  - **S = shared**
  - **I = Invalid**

- **M** bloque sucio, en propiedad → **M** = (1 Mc ≠ Mp)
  - este estado permite propagar las escrituras
  - los fallos de otra cache deben servirse desde la cache en este estado
- **S** bloque limpio, quizás compartido → **S** = (nMc = Mp; n≥1)
- **I** bloque inválido

Ejemplo de transacción en MSI: wh en Mc1 e invalidación desde Mc1 (a Mc2)

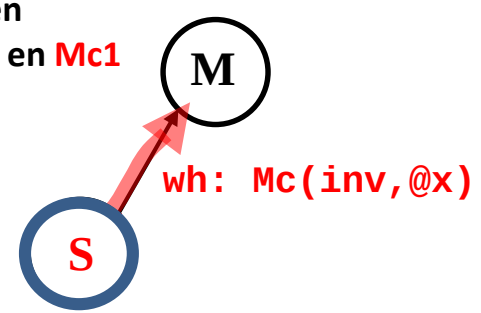


...
st @x', r4 -> wh
...

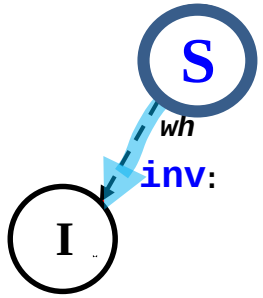
Bloque *x* en K_{Mc1} estado **S**

Bloque *x* en K_{Mc2} estado **S**

Acierto en escritura en Mc1



Estados y transiciones

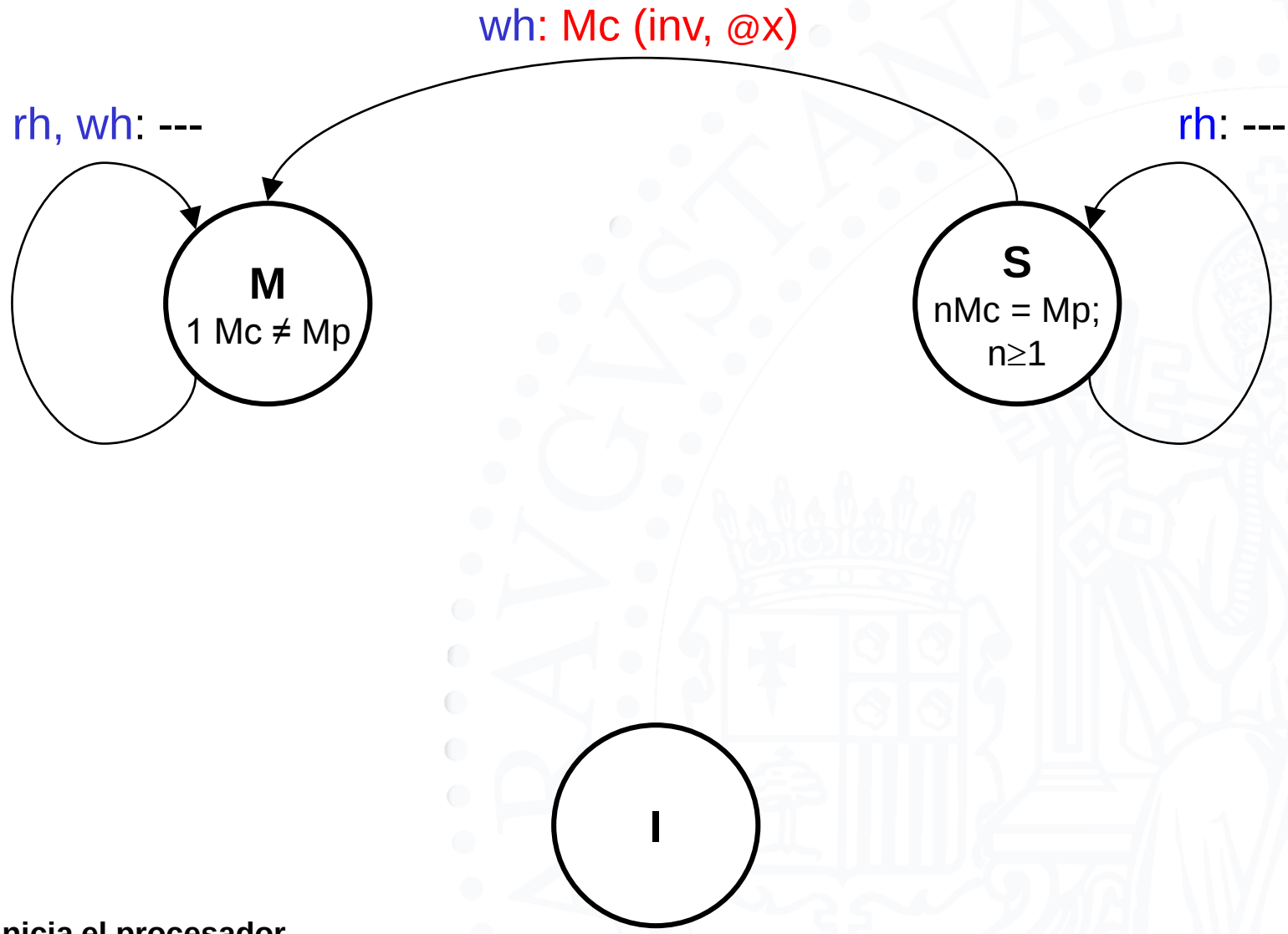


Snoopy =

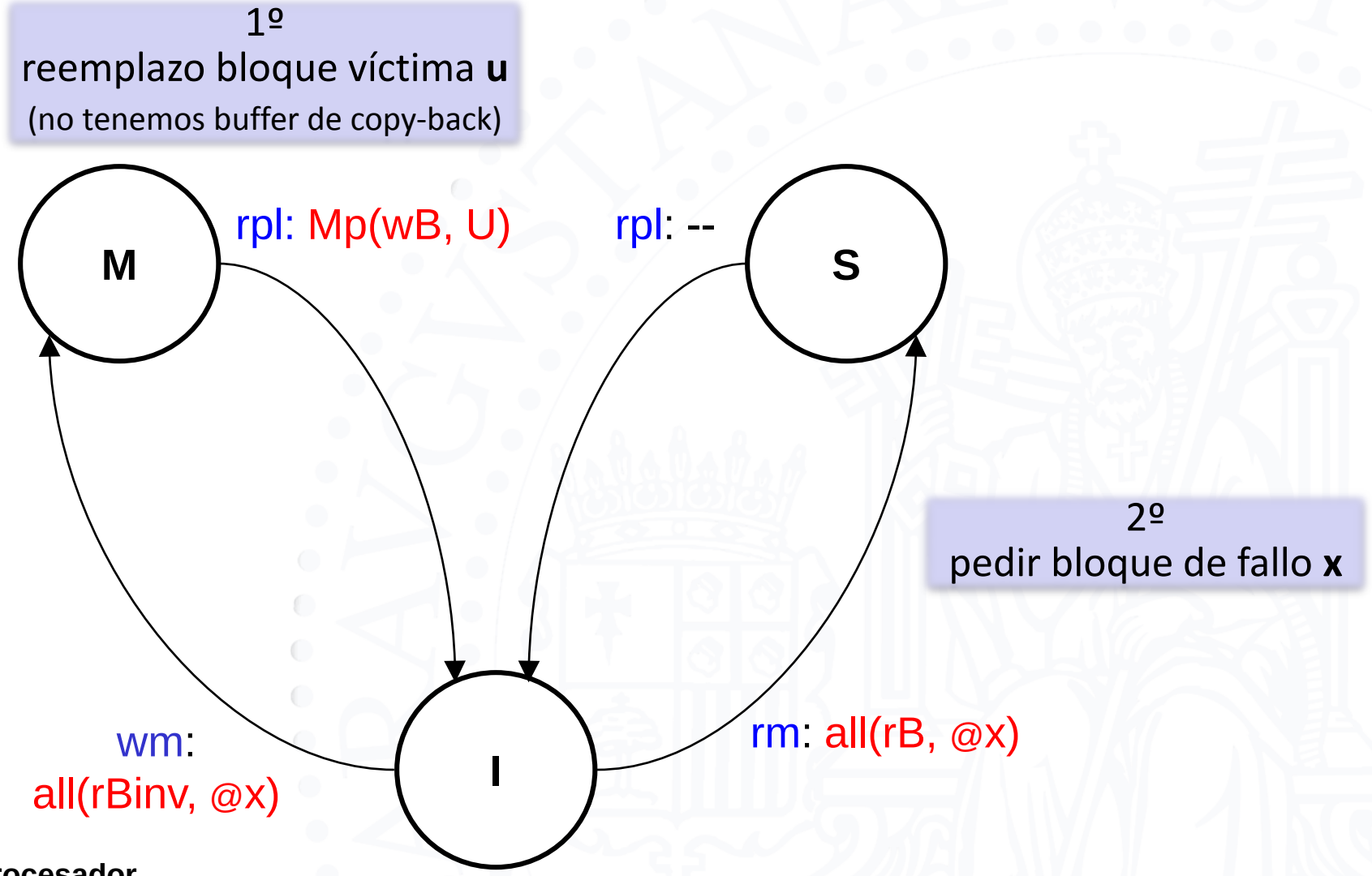
Índice

- El problema de la coherencia
  - sistema, multiprocesador, cache multinivel, mas ejemplos
  - escritura retardada e inmediata (animaciones)
- El modelo de memoria
  - consistencia secuencial, pros y contras
  - una definición de coherencia
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunos transacciones de ejemplo
  - protocolo sencillo de directorio

Protocolo MSI, aciertos procesador



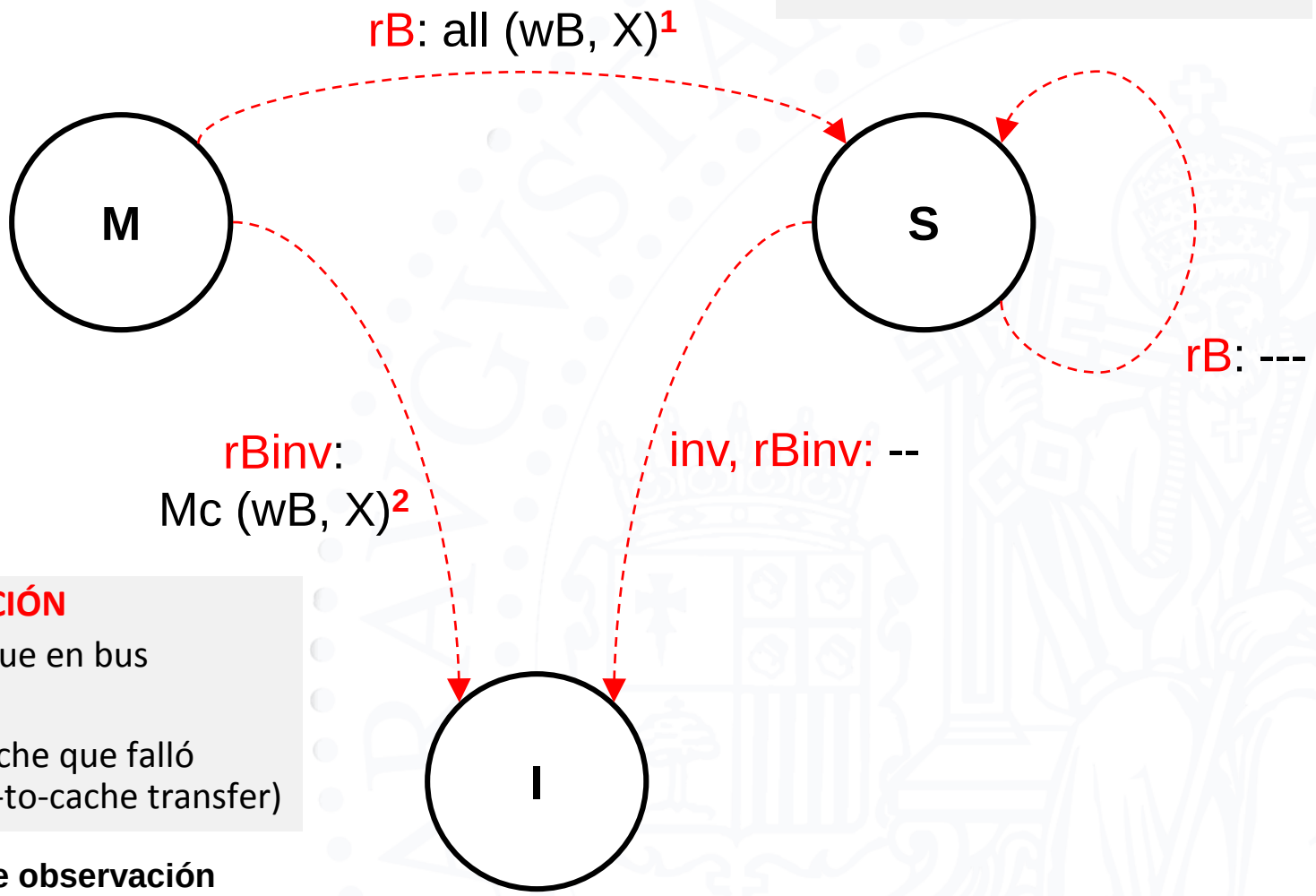
Protocolo MSI, fallos procesador



Protocolo MSI, aciertos de observación

- rB
- inv
- rBinv
- ~~wB~~

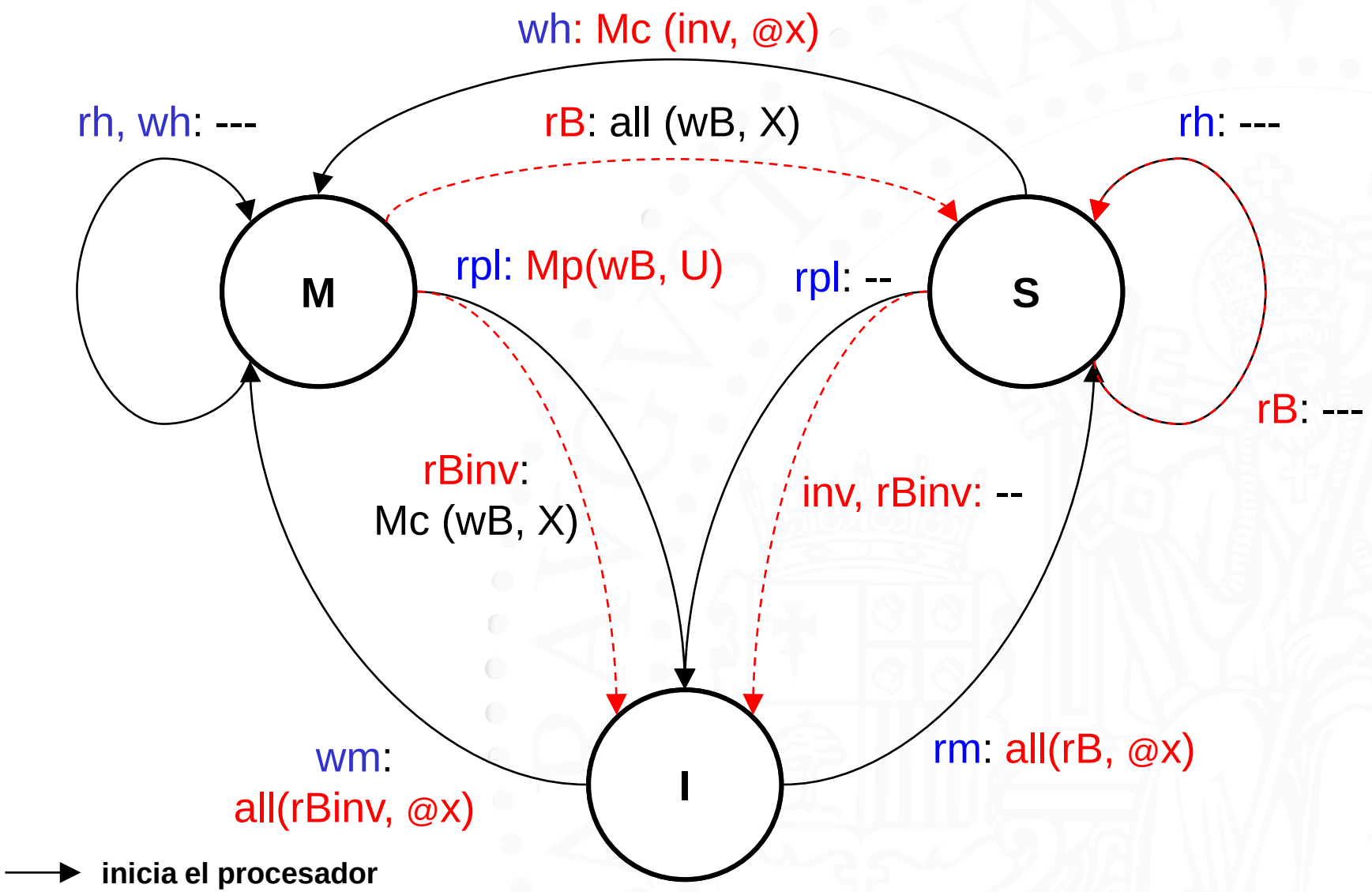
1. INTERVENCION
- colocar bloque en bus
- bloque → (Mp, cache que falló)



2. INTERVENCIÓN
- colocar bloque en bus
- inhibir Mp
- bloque → cache que falló
  (C2C: cache-to-cache transfer)

aciertos de observación

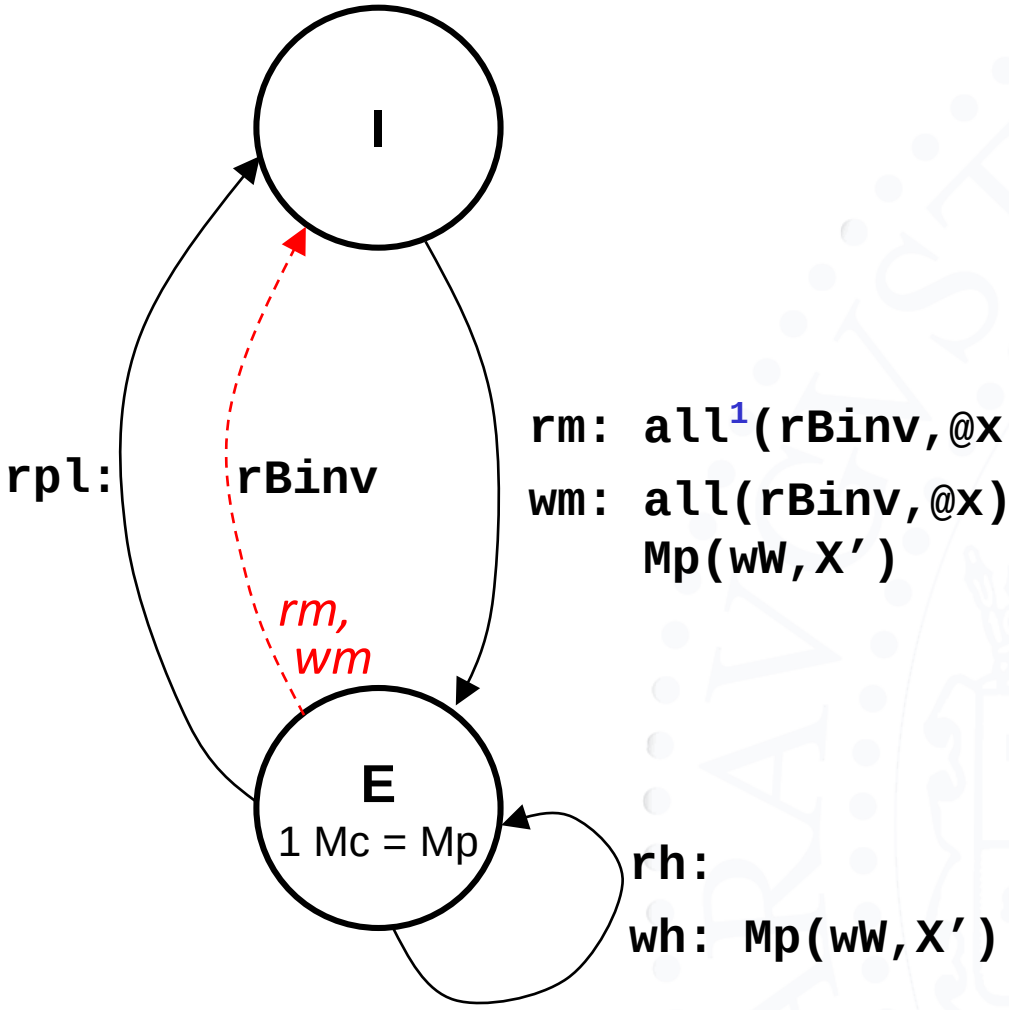
Protocolo MSI completo



Protocolo MSI: resumen

- Petición de procesador, dos tipos:
  - rd, lectura → acierto o fallo: rh o rm
  - wr, escritura → acierto o fallo: wh o wm
- Cuatro comandos de bus desde el controlador:
  - **rB**, petición de lectura de bloque, en caso de fallo en lectura (rm)
    - Mp u otra cache suministra el bloque
  - **rBinv**, petición de bloque en exclusiva, para ser propietario, en caso de fallo en escritura (wm)
    - Mp u otra cache suministra el bloque
  - **inv**, reclamar bloque en exclusiva, en caso de acierto en escritura (wh) en estado S
  - **wB**, colocar bloque en bus, puede ir a parar a Mp y/o a una Mc
    - en caso de reemplazo de bloque sucio → rpl
    - y también para propagar de escrituras

Ejercicio: protocolo **EI**, caches Write-Through WA



- E = Exclusive
  - 1 Mc = Mp
  - propiedad
  - coherente con Mc
- O sea, un bloque solo puede estar en una Mc, cuyo contenido es coherente con Mp
- Propagación escrituras:
  - Invalidar – escribir Mp – leer Mp

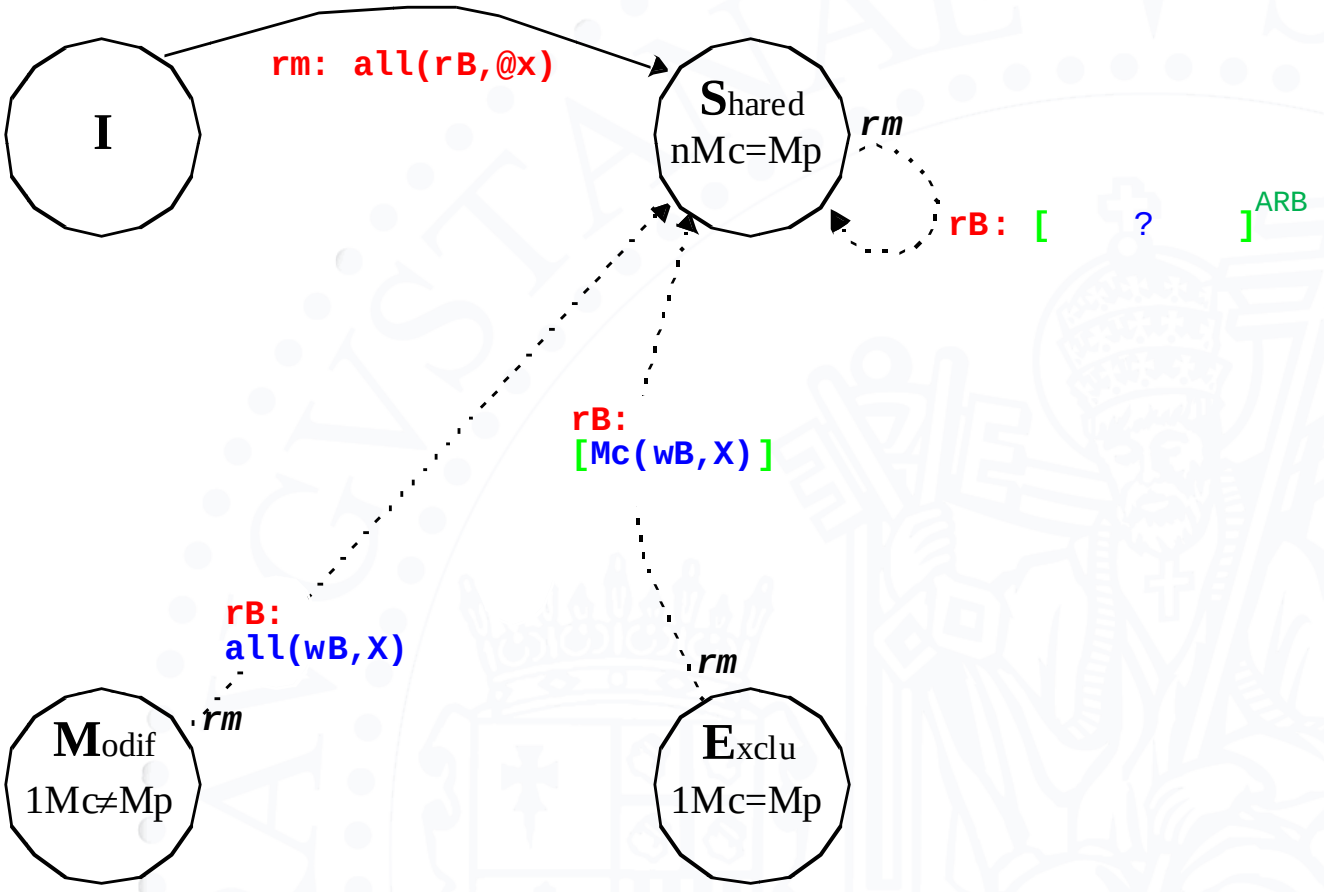
1. Mp(rB,@x)//Mc(inv,@x)= all(rBinv,@x)

Write Once[1] (1)

Acierto en Lectura, rh. 1ª trans. de fallo: reemplazo, rpl

1. Goodman, J. R. (1983, June). **Using cache memory to reduce processor-memory traffic**. In ACM SIGARCH Computer Architecture News (Vol. 11, No. 3, pp. 124-131). ACM.

Write Once (2)

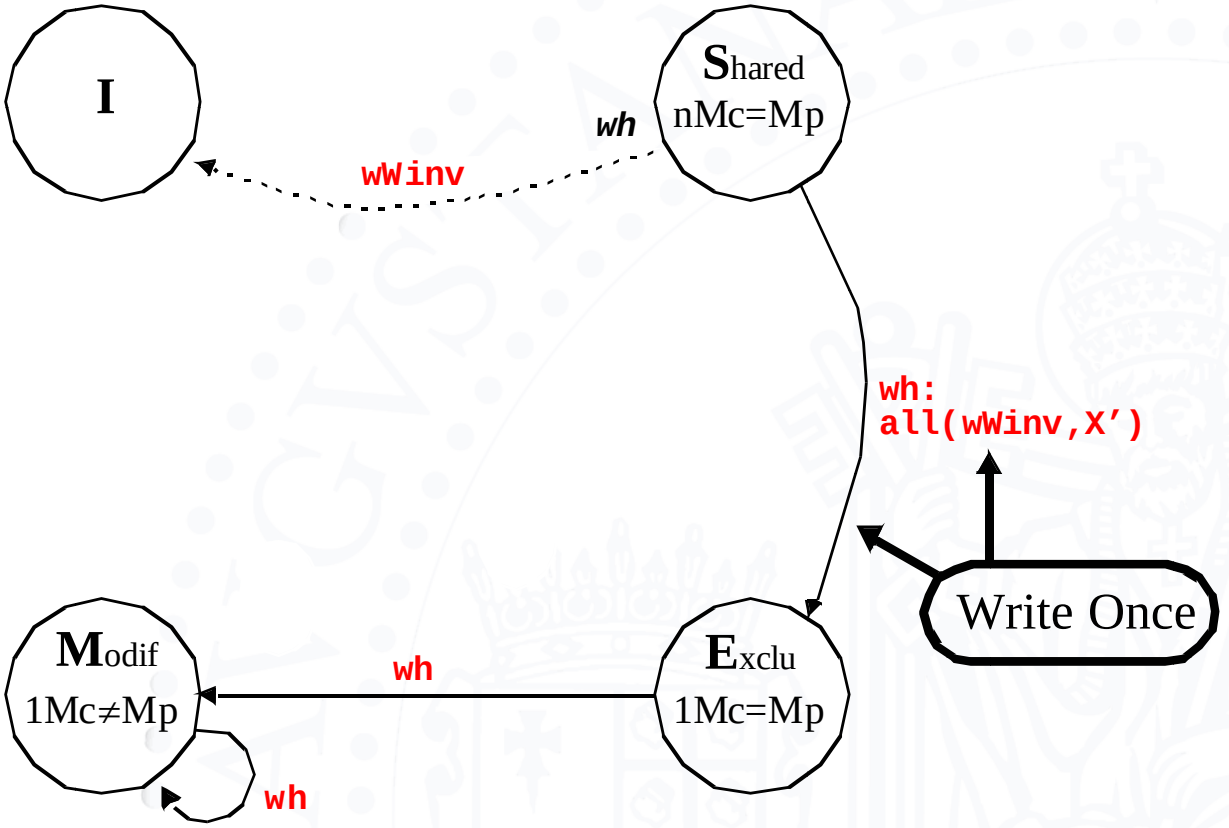


2ª transición de fallo: fallo lectura, rm

[Optativo]: Intervención = servicio de "copia rápida"

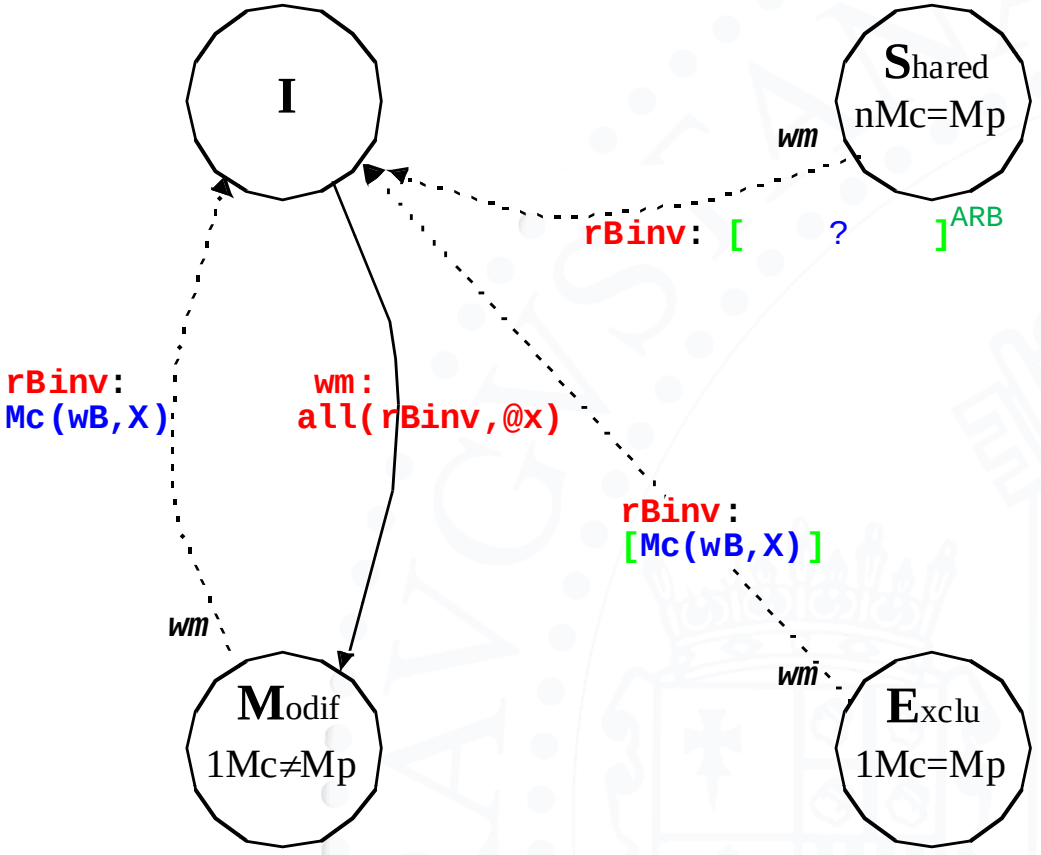
= C2C, cache-to-cache transfer

Write Once (3)



Acierto en escritura, wh

Write Once (4)



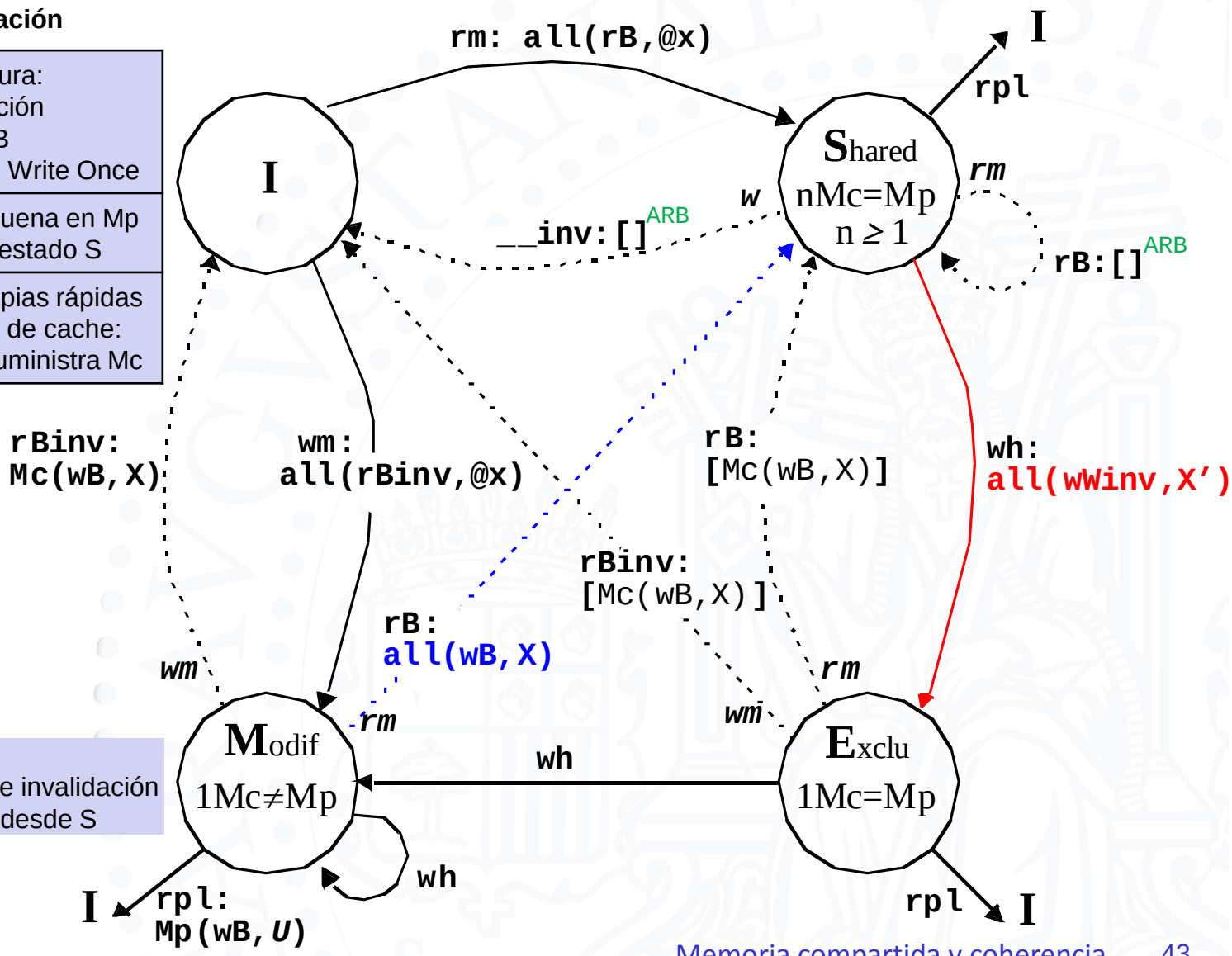
2ª transición de fallo: fallo escritura, wm

[Optativo]: Intervención = servicio de "copia rápida"

= C2C, cache-to-cache transfer

Write Once (5)

comando	cuando?	explicación
`all(wWinv`	`wh^S`	Primera escritura: WT + invalidación Siguientes: CB Solo existe en Write Once
`all(wB`	`rB^M`	queda copia buena en Mp y 2 copias en estado S
`[ Mc(wB`	`rB^E` `rBinv^E`	Servicio de copias rápidas = intervención de cache: - inhibe Mp, suministra Mc



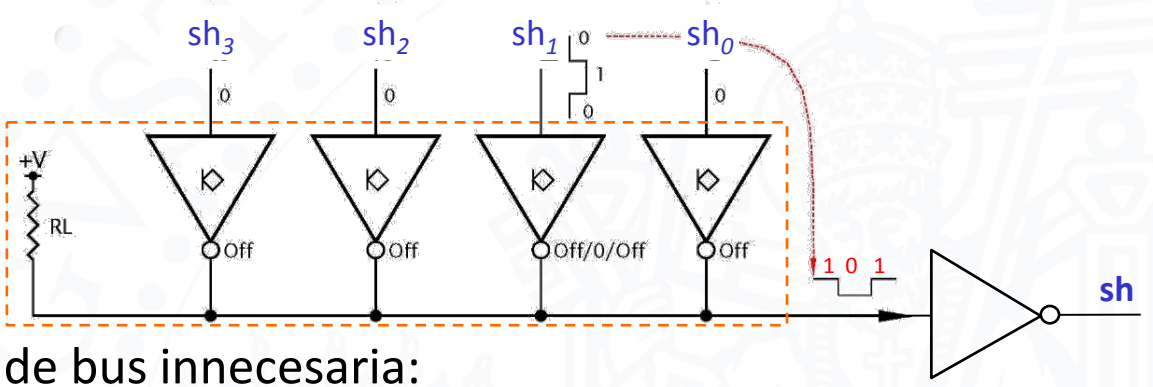
Escritura de variables locales ?
- pasamos a M y no generamos tráfico de invalidación

Es necesario arbitrar si se sirven copias desde S

Otro protocolo MESI en caches Copy-Back

- Cómo entrar en estado **Exclusivo** sin escritura a través ?
  - **E** = (1 Mc = Mp)
  - Entrar en **E** → soporte hw para detectar compartición: línea **sh**

{$sh_i = 1$} → $cache_i$ tiene el bloque
- NOR cableada + inversor
- línea **sh** = OR(sh_3, sh_2, sh_1, sh_0)



  - Reduce una transacción de bus innecesaria:
    - inv que convierte un bloque de S a M
      todo el mundo observa, pero estoy solo

- P.e. protocolo Univ. Illinois, Pamarcos y Patel, 1984
  → Motorola M88110
- Empleado en procesadores Intel, IBM PowerPC, MIPS

1 to 48-core
MIPS64 64-bit
"network processor"

Ejemplos comerciales: solo invalidación

Compañía	Estados	Red
ARM9 TDI (embedded SoC)	MESI	Bus, línea sh
AMD Athlon MP, Opteron	MOESI	Punto a punto. Difusión o directorio. *Hyper Transport Link*
Hewlett-Packard Alpha	MOESI	Bus
IBM PowerPC 603, 755	MEI	Bus
IBM PowerPC 740, 750	MESI	Bus
IBM Power4, 5, 6, 7	MESI[a] mejorado	Punto a Punto
Intel Pentium Pro, Pentium II/ Xeon, Pentium III/Xeon, Itanium, Itanium 2	datos: MESI[b] instr: SI	Bus, línea sh
Sun UltraSPARC	MOESI[c]	Bus
Intel Xeon 7300 (2007), Xeon 7400(2008), Nehalem-EX (2009), Westmere-EX (2010)	MESIF	Punto a punto. Difusión o directorio. QPI: *QuickPath Interconnect*[d]
Oracle Sun T1, T2, T3	MI WT en L1 + MESI en L2	crossbar entre L1s y L2

a. Incluye estados adicionales en las caches L2 y L3.
b. Introduced by Intel in the Pentium processor to "support the more efficient writeback cache in addition to the write-through cache previously used by the Intel 486 processor." The MESI protocol is also known as the Illinois protocol.
c. Nomenclatura alternativa de Sun: M = Exclusive Modified, O = Shared Modified, E = Exclusive Clean, S = Shared Clean.
d. The *QuickPath* Architecture is implemented jointly by I/O Hubs and Caches, and has four layers. The Physical layer consists of the actual connection between components. The Link layer is responsible for flow control and the reliable transmission of data. The Routing layer is responsible for the routing of QPI data packets. Finally, the Protocol layer is responsible for high-level protocol communications, including the implementation of a MESIF (Modify, Exclusive, Shared, Invalid, Forward) cache coherence protocol.

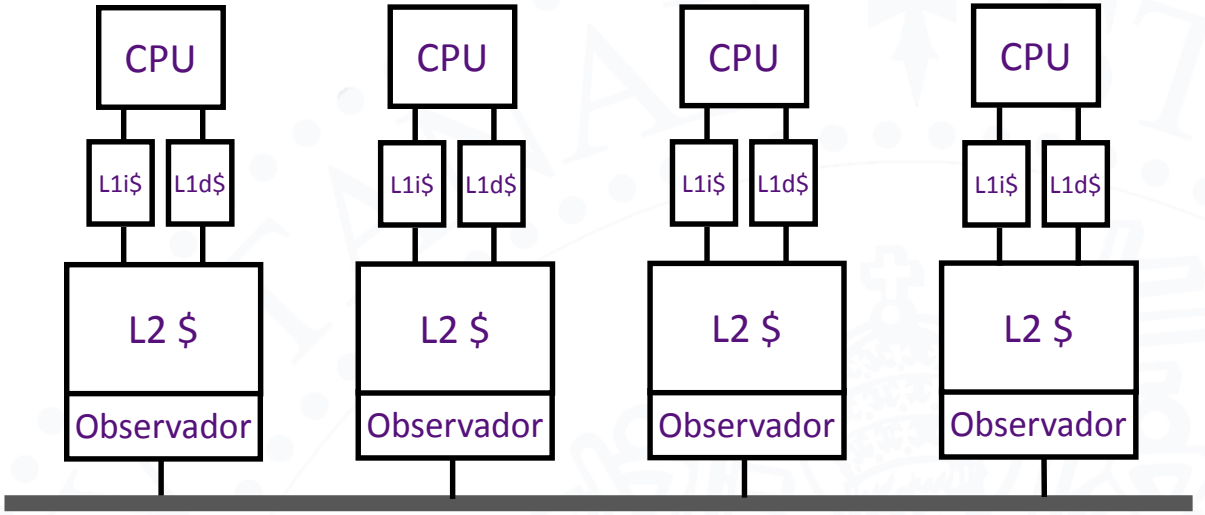
- Un ejemplo del 2009: servidor 1U Dell PowerEdge R610, con 2 chips Intel Xeon E5620 quad- core.

Índice

- Algunas organizaciones de multis en chip
- Modelo de memoria y consistencia secuencial
  - coherencia
  - serialización de escrituras
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunos transacciones de ejemplo
  - protocolo sencillo de directorio

Jerarquía multinivel

Chip multiprocesador con dos niveles de caches privadas



- La observación se hace en eventos de L2, no afecta directamente a L1
- Posibilidad 1: inclusión de contenidos: L1 $\subset$ L2
  - cada bloque en L2 tiene un bit de "presencia en L1"
  - invalidación en L2 → invalidación en L1 (Intel, IBM, Oracle)
  - reemplazo en L2 → posible reemplazo en L1: víctimas de inclusión
- Posibilidad 2: exclusión de contenidos: L1 ≠ L2
  - mayor tamaño neto L1+L2, permite reducir el tamaño de L2 (AMD)
  - para no molestar a L1 en cada observación *negativa* de L2, replicar las etiquetas de L1 en L2
    - o sea: 1º observar L2 y en caso de fallo, 2º observar etiquetas replicadas de L1
      Inclusión de directorio
- Posibilidad 3: no inclusión – no exclusión de contenidos: L1 $\cap$ L2 ≠ $\varnothing$ (AMD)

Algunos problemas de la colección

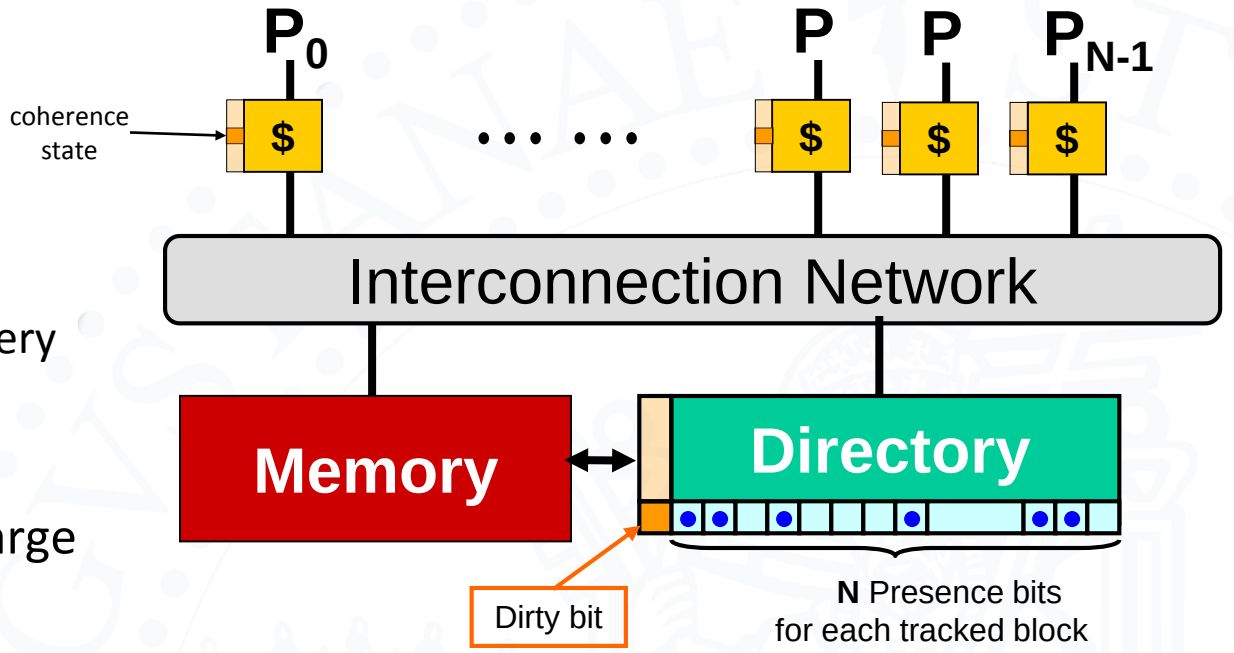
- **Problema 18.** Protocolo MESI de invalidación con línea **sh**. También conocido como Illinois, por la Universidad que lo propuso. HACERLO !!!
- **Problemas 20, 23 y 24.** Coherencia en caches multinivel.
- **Problema 21.** Primitivas de sincronización Fetch&Inc y su implementación en un protocolo MSI.

Índice

- Algunas organizaciones de multis en chip
- Modelo de memoria y consistencia secuencial
  - coherencia
  - serialización de escrituras
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunas transacciones de ejemplo
  - protocolo sencillo de directorio

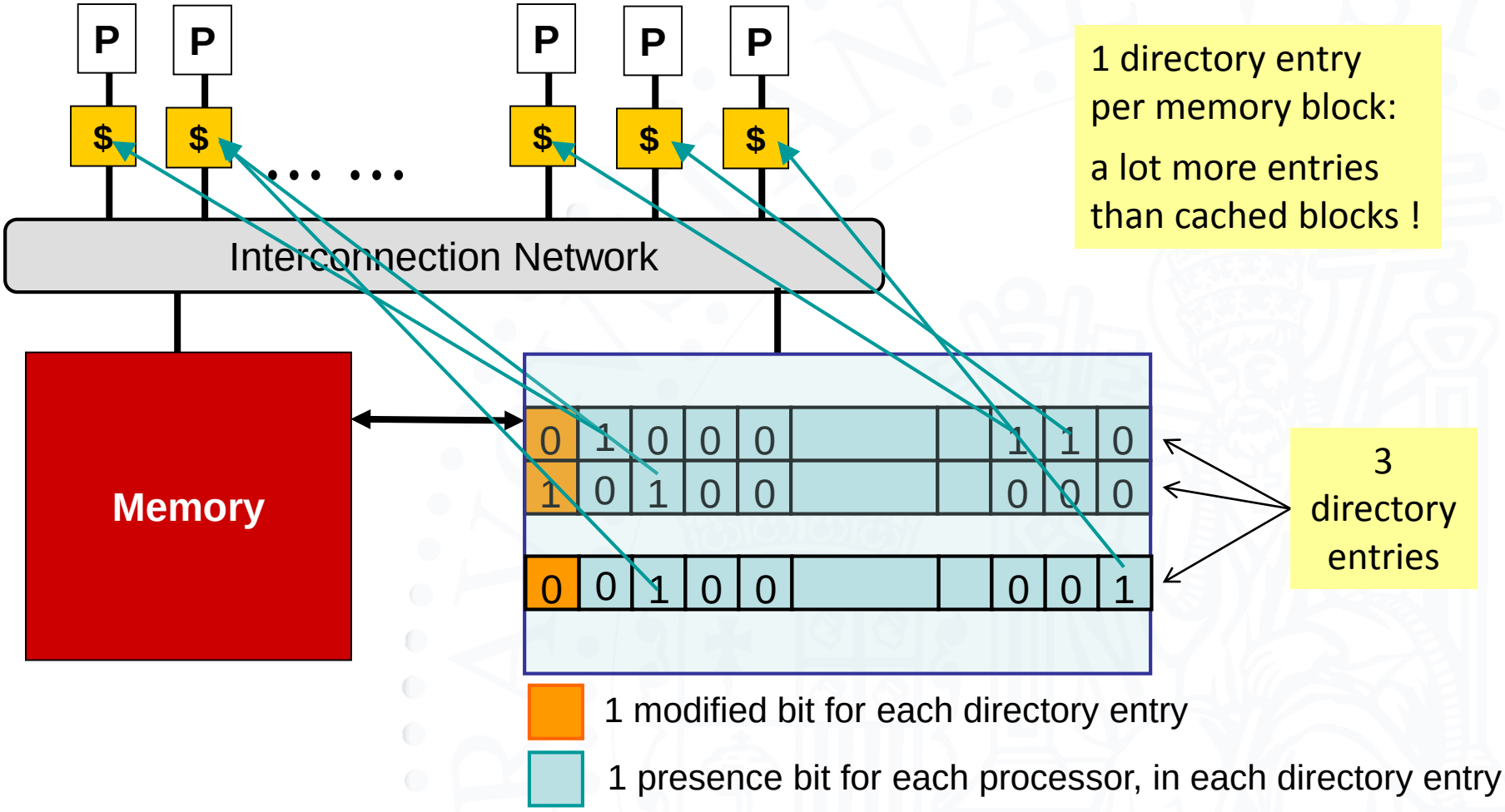
Directory-based Coherence Protocols

- Snooping-based protocol
  - **N** lookups / broadcast (↑ energy)
    - all caches need to watch every bus request from every processor
  - Not a scalable solution for maintaining coherence in large shared-memory systems



- Directory protocol:
  a directory keeps a list of sharers for each block
  - HW overhead to keep the directory in main memory
    ≈ *# lines in MM * # processors*
    - alternatively, the directory can be in the shared last-level cache (SLLC)
      overhead ≈ *# lines in SLLC * # processors*

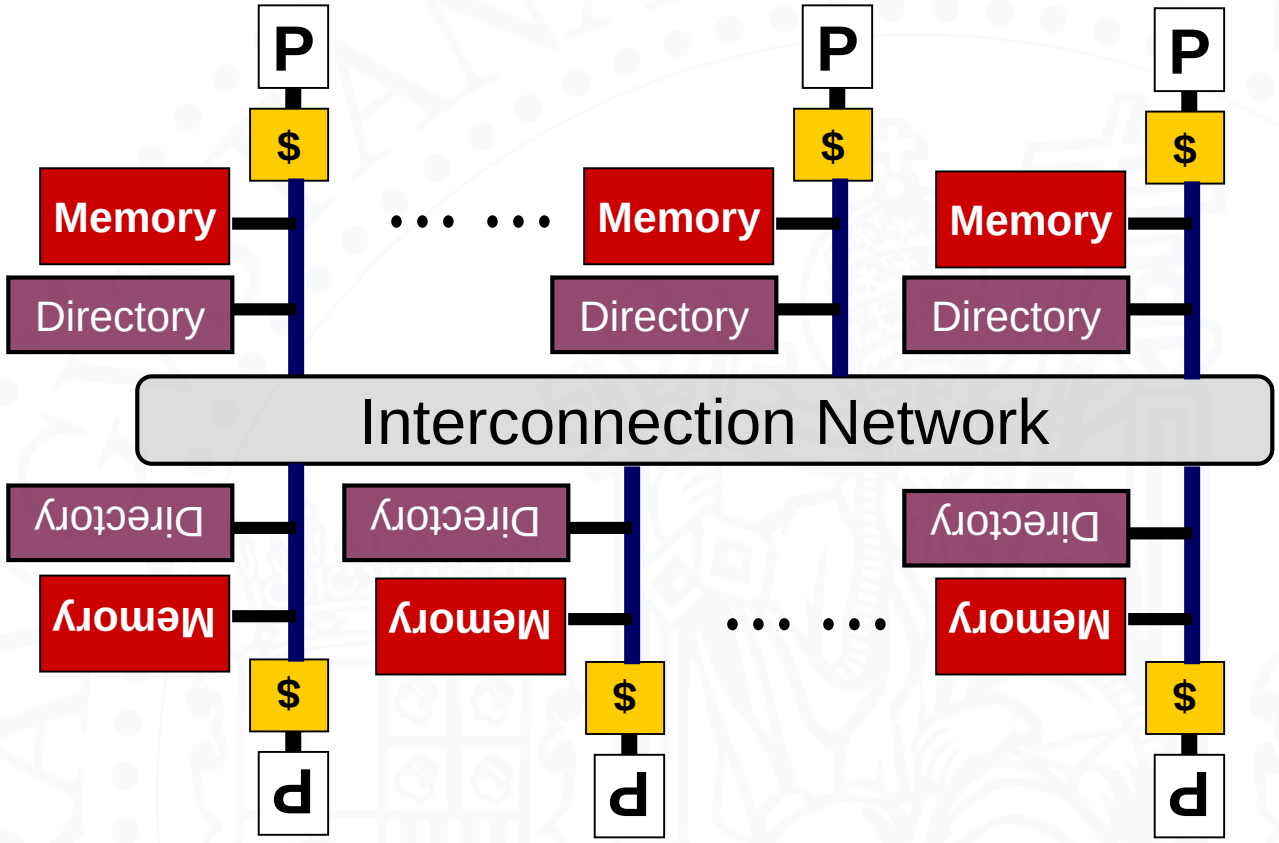
Directory-based Coherence Protocols



Presence bit vector $\underline{P} = (p_n .. p_i .. p_{N-1})$ 0 1 0 0 0 0 1

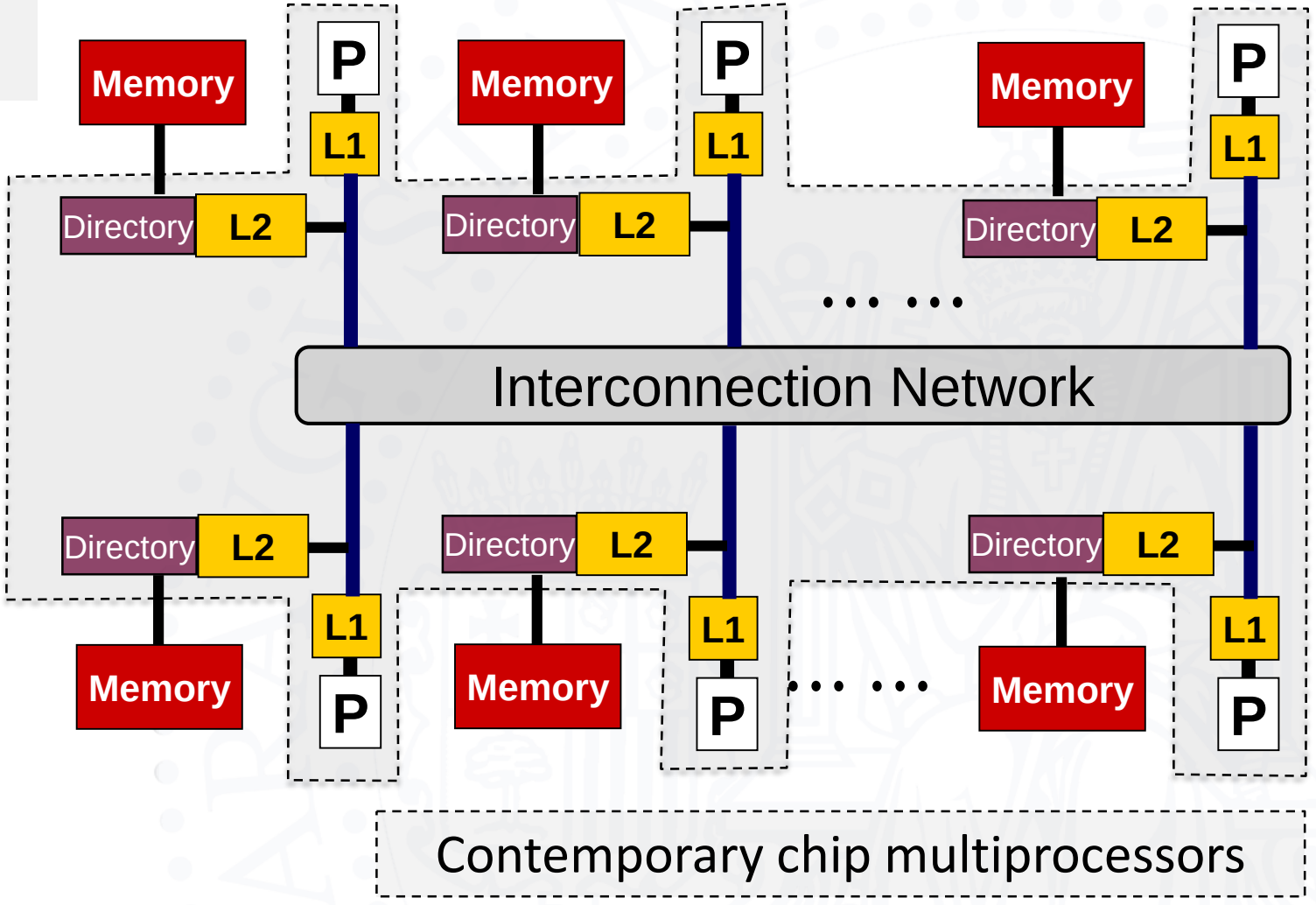
Distributed Directory (1)

- Centralized directory is not scalable (contention)
- To build large MP systems, several processing nodes (P) are tied by means of a powerful interconnection network (no longer a shared bus)
- Nodes are made up of processor, caches, and a fraction of directory and main memory:
  - Distributed shared memory (DSM) *Stanford*
  - Cache-coherent Non-Uniform-Memory-Architecture (CC-NUMA) *Berkeley*
- Each block has a "home" {0 ..N-1} $\log_2$ N bits from the Most Significant Bits of address
- 1 directory entry per memory block (a lot more entries than cached blocks !)



Distributed Directory (2)

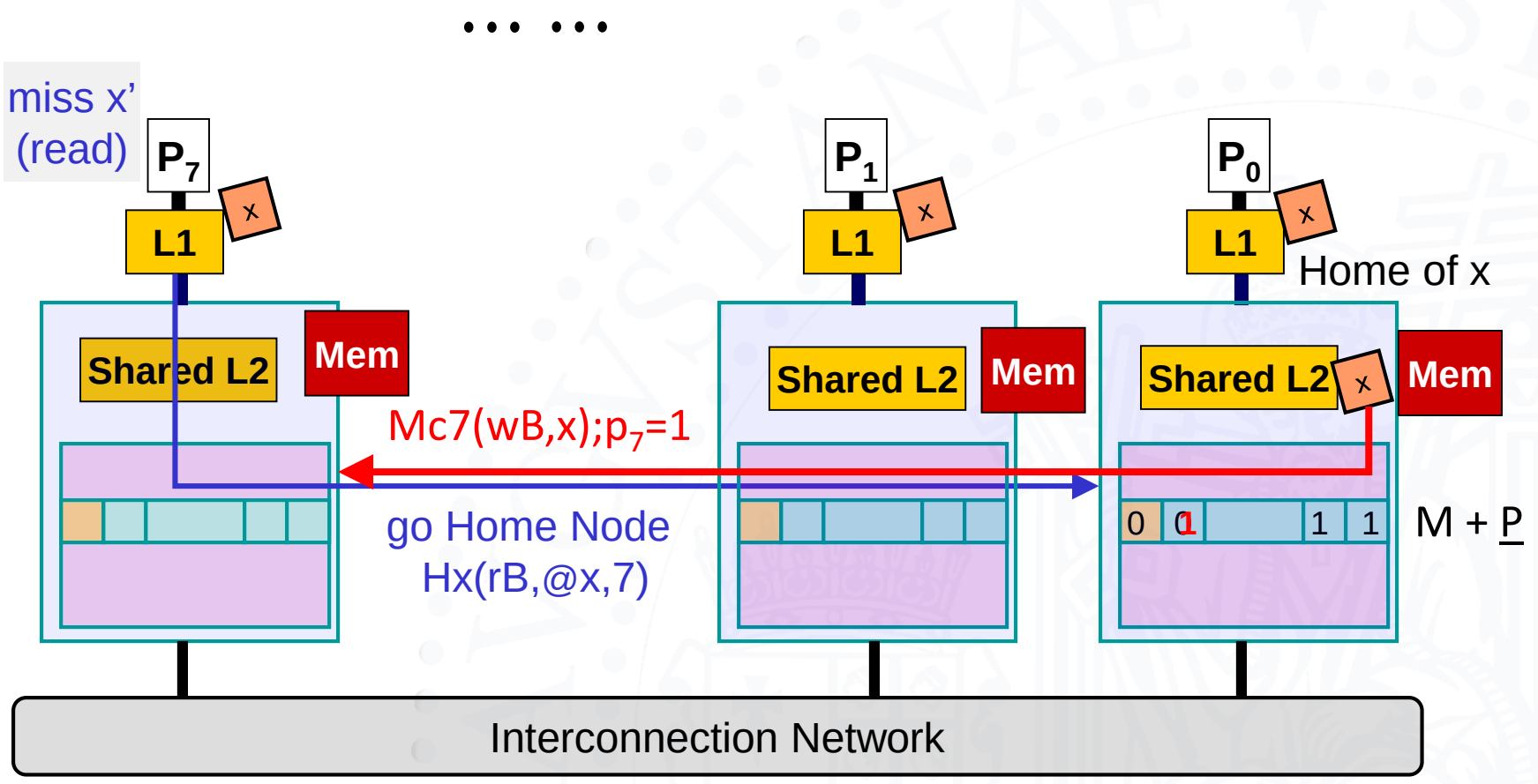
- 1 directory entry per shared last-level cache block



L1 Privadas

L2 Compartidas "Homes"

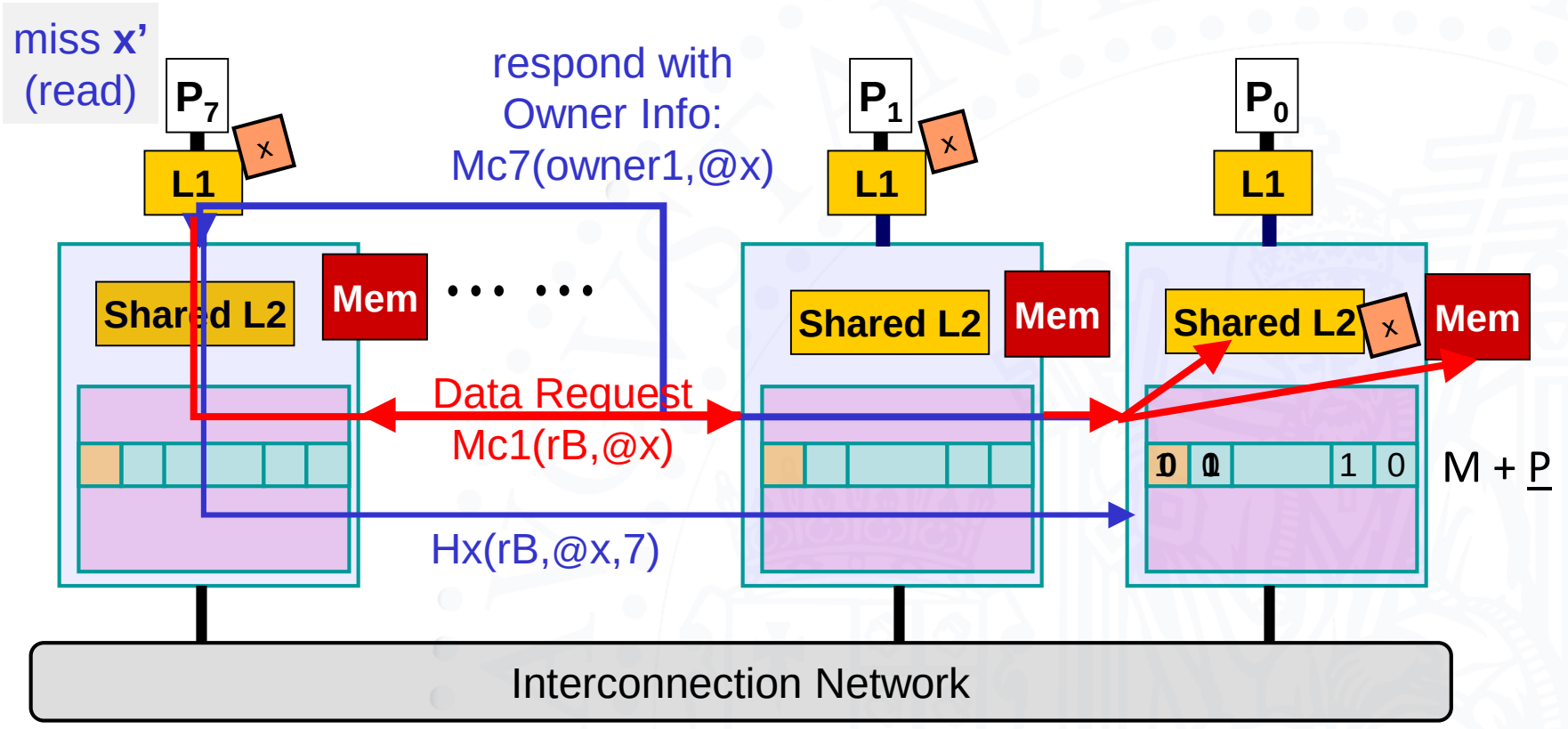
Example: Read Miss on cached a Clean Line



Line **x** is Shared in $L1_0$ and $L1_1$

The shared L2 cache in Node 0 has a clean copy in state **S** (nMc1 = Mp)

Example: Read Miss on a cached Dirty Line



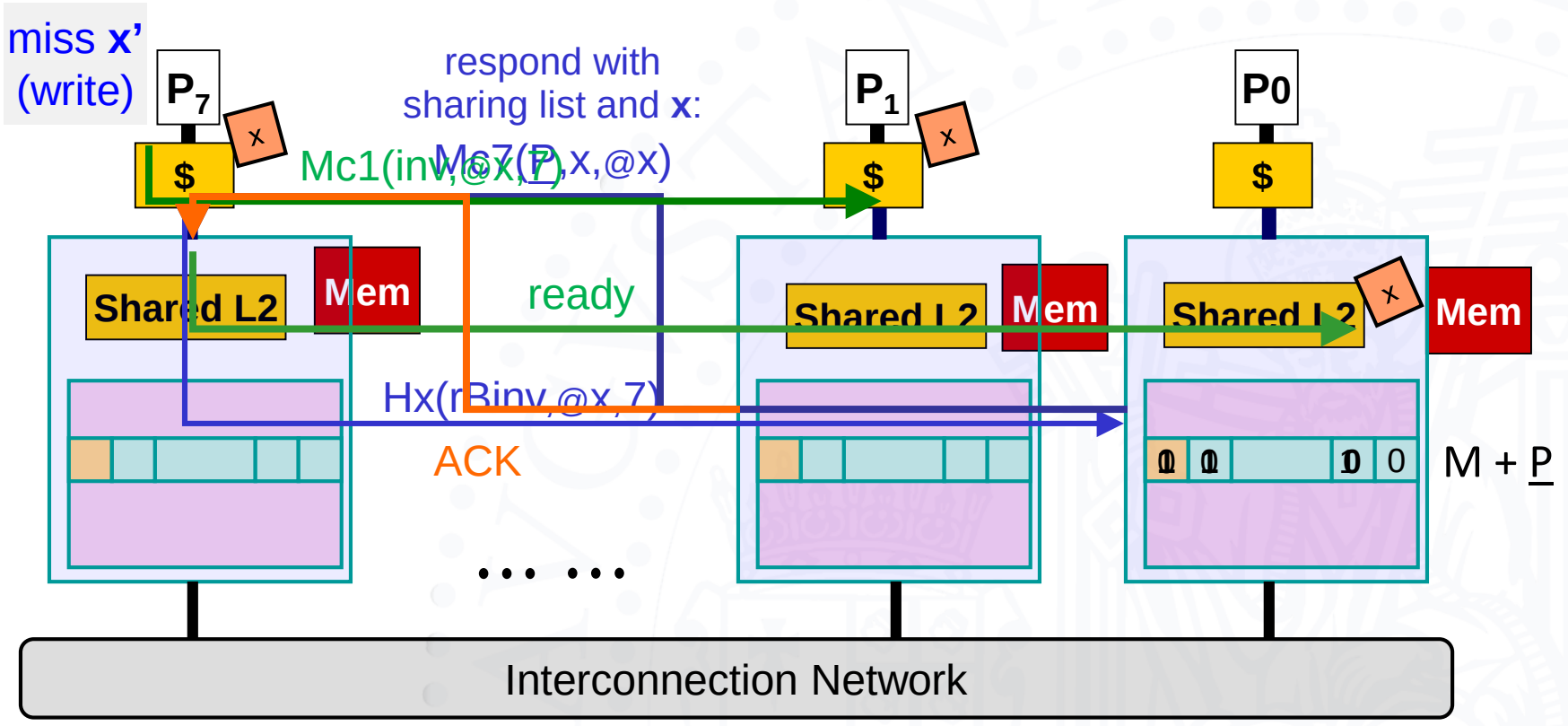
Line **x** is **Dirty in $L1_1$ (M state)**

Eventually, line **x** is Clean (S state)
in $L1_1$ and $L1_7$,
in $Home_0$ shared L2, and in Mem

Example: Write Miss (WA) on cached clean line

Line **x** is Shared in $L1_1$

The shared L2 cache in Node 0 has a clean copy in state **S** (nMc1 = Mp)



Write x' can proceed in P_7

There are many other ways to manage messages in a write miss.
The best one solution has only 3 messages in the critical path !

Índice

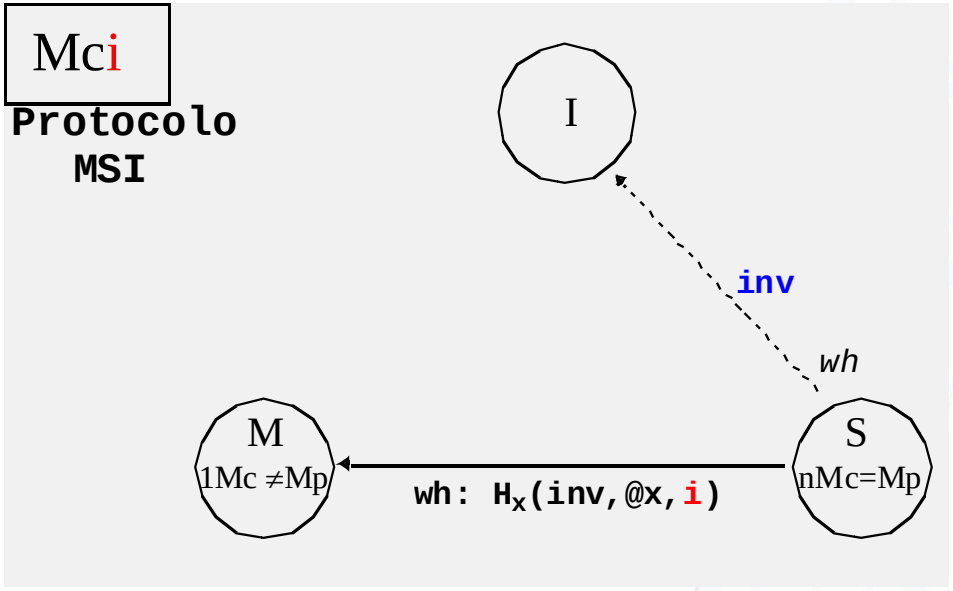
- Algunas organizaciones de multis en chip
- Modelo de memoria y consistencia secuencial
  - coherencia
  - serialización de escrituras
- Protocolos de coherencia basados en difusión
  - Invalidación. Difusión vs. envío selectivo
  - ejemplos invalidación + CB + Bus: MSI, EI, Write Once, MESI
  - Protocolos comerciales
- Jerarquía de caches multinivel
- Protocolos de coherencia basados en directorio
  - requisitos hw y algunas transacciones de ejemplo
  - protocolo sencillo de directorio

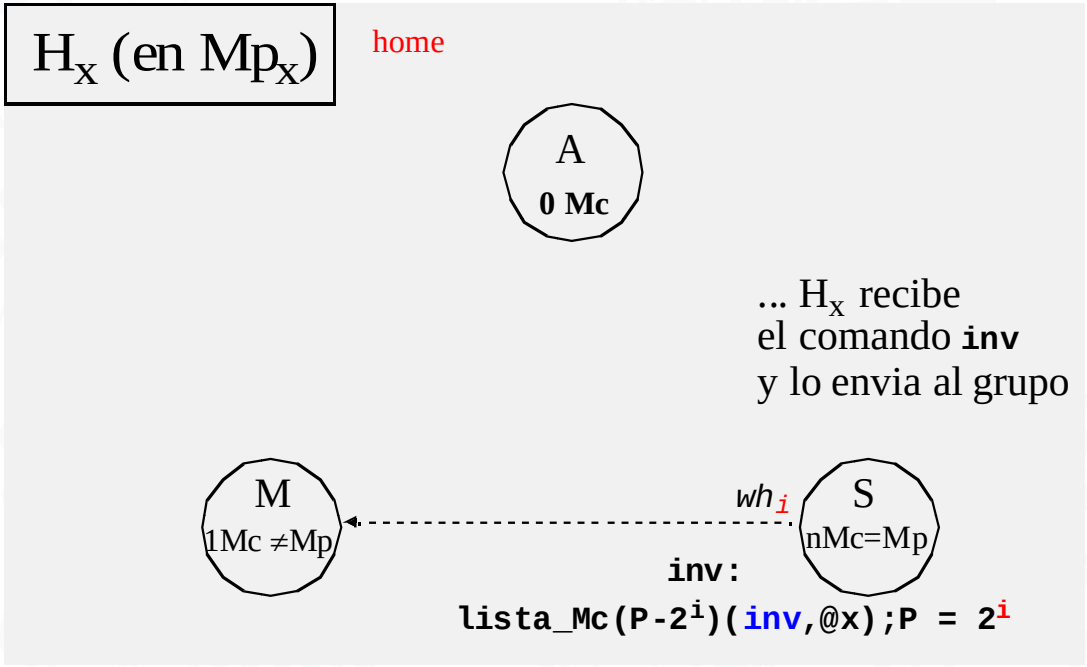
Protocolo sencillo de directorio

- $N=2^n$ nodos con una memoria cache privada (Mci) y un trozo de Mp (Mpi) con un "Hogar-Directorio" de coherencia (Hx) en cada nodo
- Cada entrada del directorio indica estado de un bloque de Mpi
  - vector de presencia con N bits: **P** = (p_{N-1}, ... , p_1, p_0)
  - si (p_i==1) ent x presente en Mci
    cc x ausente de Mci
  - 1 bit M $\in$ { limpio=0, sucio=1 }, junto con **P** permite 3 estados:
    - **A** absent (0 Mc) $\leftrightarrow$ P = 0
    - **S** shared (nMc = Mp) $\leftrightarrow$ P $\neq$ 0 $\wedge$ LS = Limpio
    - **M** modified (1Mc $\neq$ Mp) $\leftrightarrow$ P = 2^i $\wedge$ LS = Sucio
- En las caches privadas un protocolo MSI

Ejemplo: acierto en escritura (1)

- La cache **Mc**i acierta en escritura
  en un bloque ***x*** en estado **S**
  y envía un mensaje de invalidación al controlador del directorio $\mathbf{H_x}$
  H_x consulta el vector de presencia **P** del bloque ***x*** y actúa ...





Ejemplo: acierto en escritura (2)

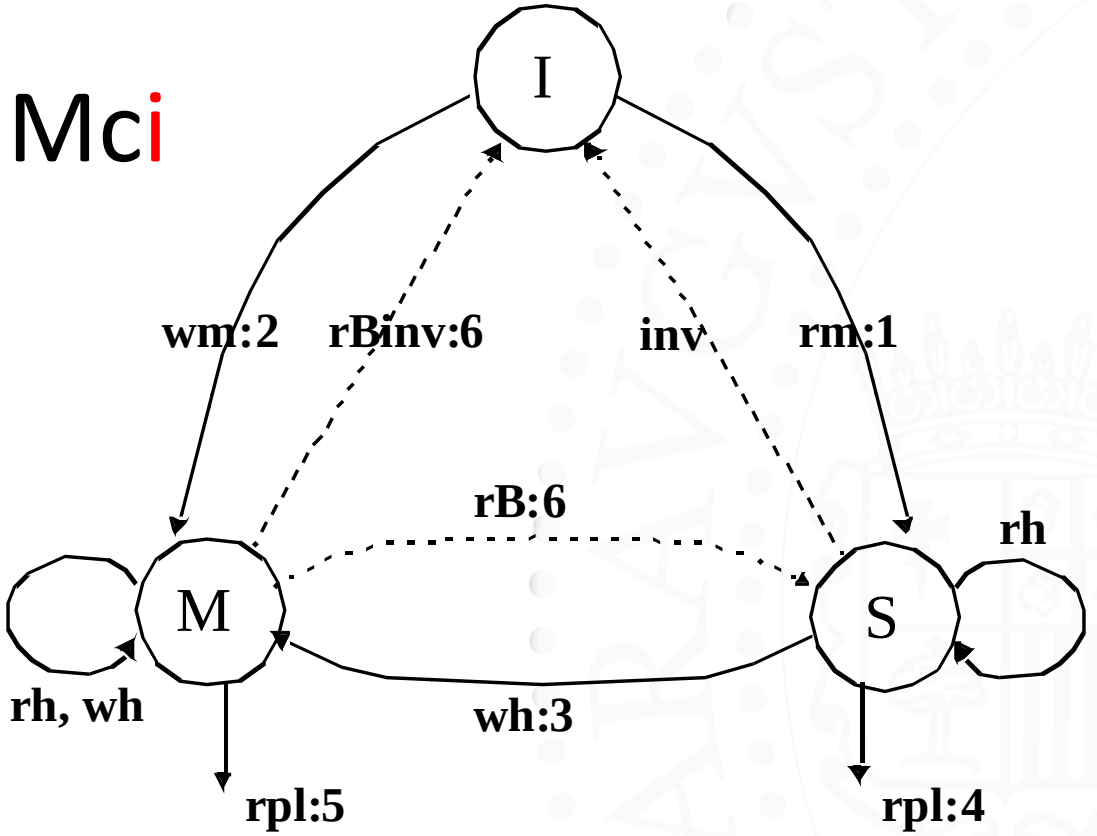
- Consistencia Secuencial
  - una escritura detiene al procesador que la realiza hasta que su efecto se nota en todas las Mc que comparten el bloque → esperar mensaje de confirmación

t ↓

P1	Mc4	Mc2	Mc1	Directorio(x) en H_x	Comentario
actividad	est	est	est	est $p_7p_6p_5p_4p_3p_2p_1p_0$	
`st @x', ...`	S	S	S	S 0 0 0 1 0 1 1 0	Situación inicial: **3Mc=Mp**
wh: H_x(inv,@x,1)	S	S	S		Mensaje Mc1>H_x
bloqueo hasta conseguir permiso de escritura				**Mc2(inv,@x) Mc4(inv,@x)**	Se consulta directorio e invalida a Mc2 y Mc4
	I **H_x(ack)**	I **H_x(ack)**	S		Mc's cambian estado y devuelven confirmación
	I	I		M 0 0 0 0 0 0 1 0 **Mc1(ack)**	Cuando H_x recibe las confirmaciones actualiza directorio y avisa a Mc1
P1 sigue ...			M	M 0 0 0 0 0 0 1 0	Mc1 recibe confirmación, cambia de estado y libera a P1

Transiciones cache privada

- Suministro de bloques centralizado en Mp
- No hay suministro entre caches



1: H_x(rB,@x,**i**)

2: H_x(rBinv,@x,**i**)

3: H_x(inv,@x,**i**)

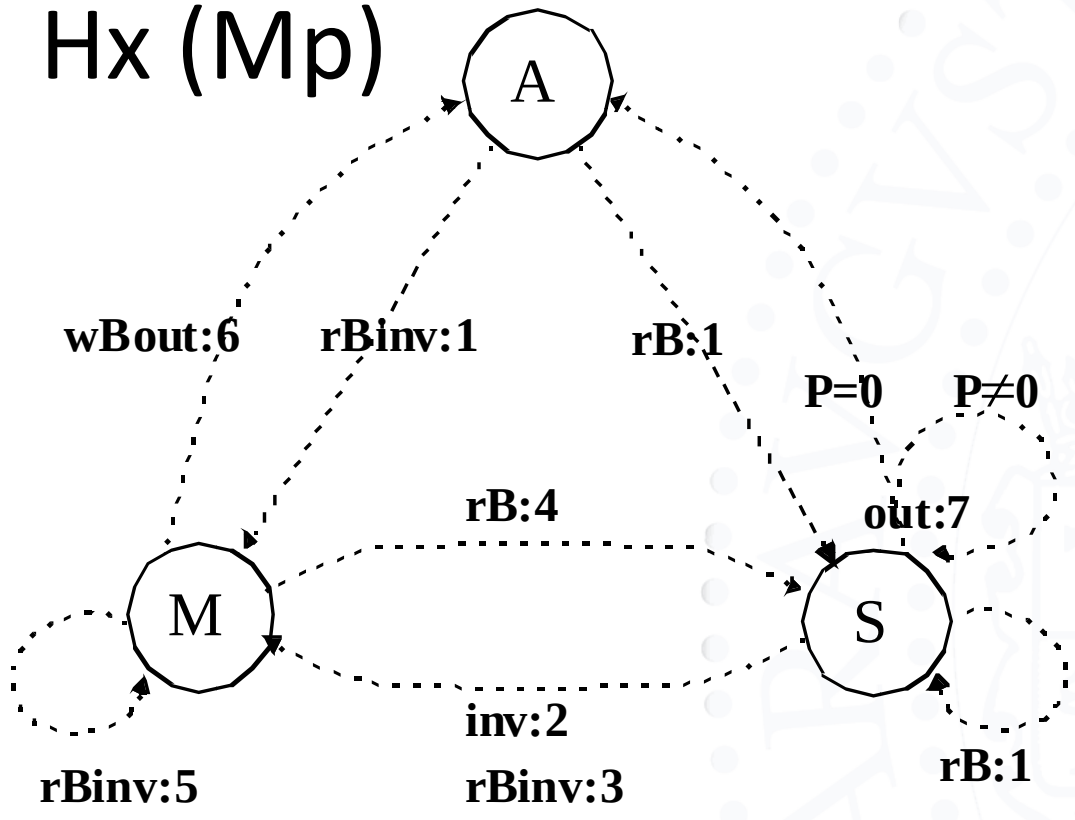
4: H_x(out,@x,**i**)

5: H_x(wBout,U,**i**)

6: H_x(ack,X,**i**)

"**ack**" mensajes de respuesta

Transiciones en el directorio



- Mci ha enviado el comando

1. Mci(wB,x); p_i=1
2. lista_Mc(P-2^i)(inv,@x); P=2^i
3. lista_Mc(P-2^i)(inv,@x); P=2^i; Mci(wB,x)
4. lista_Mc(rB,@x); Mp+x; Mci(wB,x); p_i=1
5. lista_Mc(rBinv,@x); Mci(wB,x); P=2^i
6. Mp+u; p_i=0
7. p_i=0

- en 5. y 6. lista_Mc solo tiene un “1”

Ejercicio: poner puntos de espera “wait” y mensajes de confirmación “ack” para consistencia secuencial

Ejercicio: modificar el protocolo anterior

- Ahora, $N=2^n$ nodos con una memoria cache privada (Mci) y un trozo de cache compartida de último nivel (SLLCi) que actúa de "Hogar-Directorio" de coherencia (Hx)
- Cada entrada del directorio indica el estado de un bloque SLLCi
  - vector de presencia con N bits: **P** = (p_{N-1}, ... , p_1, p_0)
  - si (p_i==1) ent x presente en Mci
    cc x ausente de Mci
  - 1 bit M ∈ { limpio=0, sucio=1 }
    1 bit I ∈ { inválido=0, válido =1 }, junto con **P** permite 4 estados:
    - **A** absent (0 Mc) ↔ P = 0
    - **S** shared (nMc = Mp) ↔ P ≠ 0 ∧ LS = Limpio
    - **M** modified (1Mc ≠ Mp) ↔ P = 2^i ∧ LS = Sucio
    - **I** invalid llego en Reset y primera transición de fallo en SLLCi
- Inclusión de contenidos:
  al expulsar un bloque en SLLC hay que asegurarla !

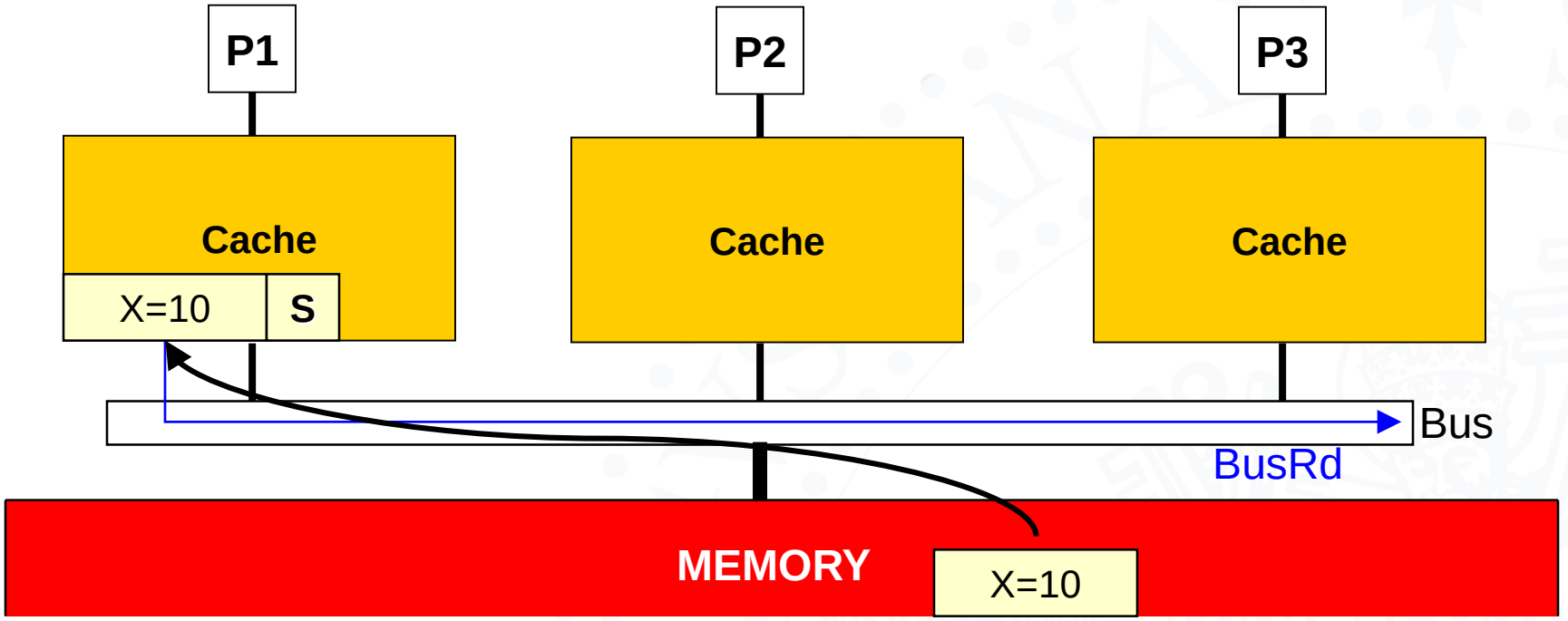
Escuela de Ingeniería y Arquitectura Universidad Zaragoza
Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

Para saber más …

- Jean-Loup Baer. Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors, Cambridge University Press, 2010.
- J. Hennessy and D. Patterson. Computer Architecture: a Quantitative Approach, 5th Ed., ISBN 9780123838728, Morgan Kaufmann, 2011
- M. Dubois, M. Annavaram, P. Stenström. Parallel Computer Organization and Design, Cambridge University Press, 2012

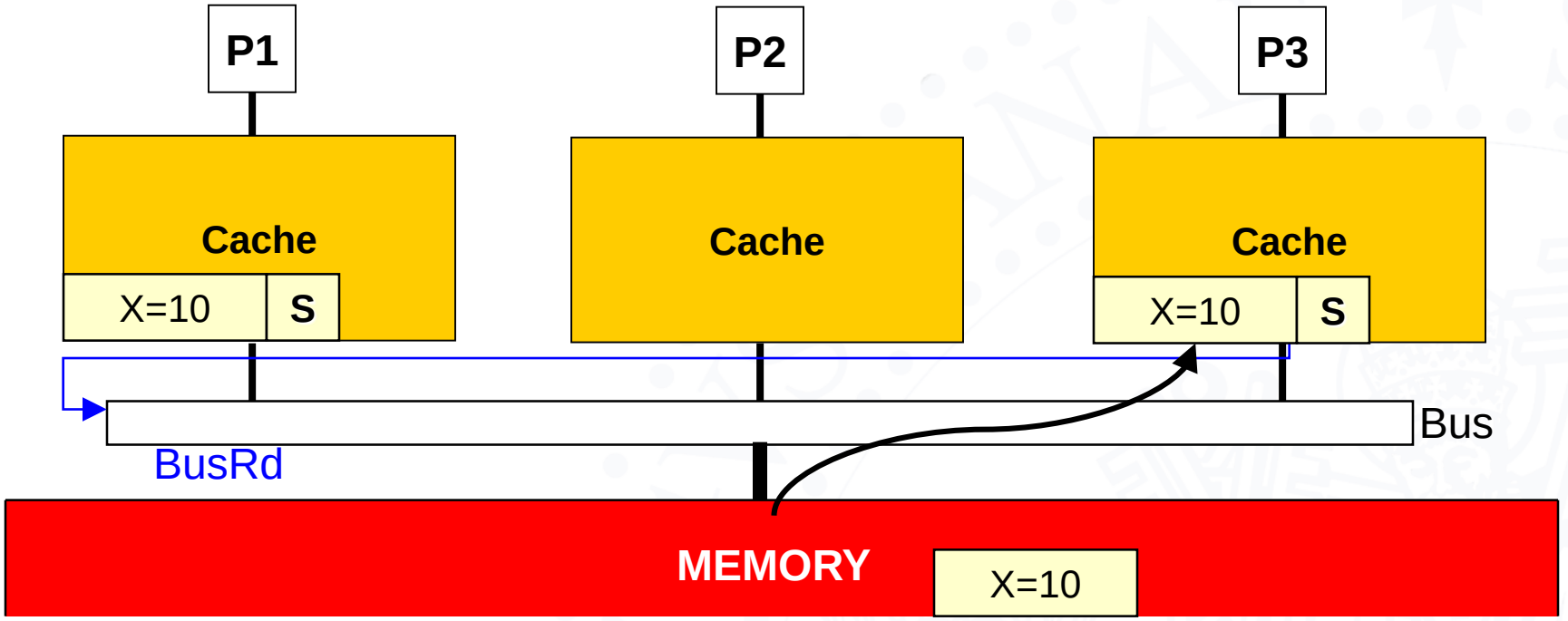
- … y un ejemplo animado de MSI → →

Ejemplo MSI → trabajarlo con los nombres de comandos de la pag 24.



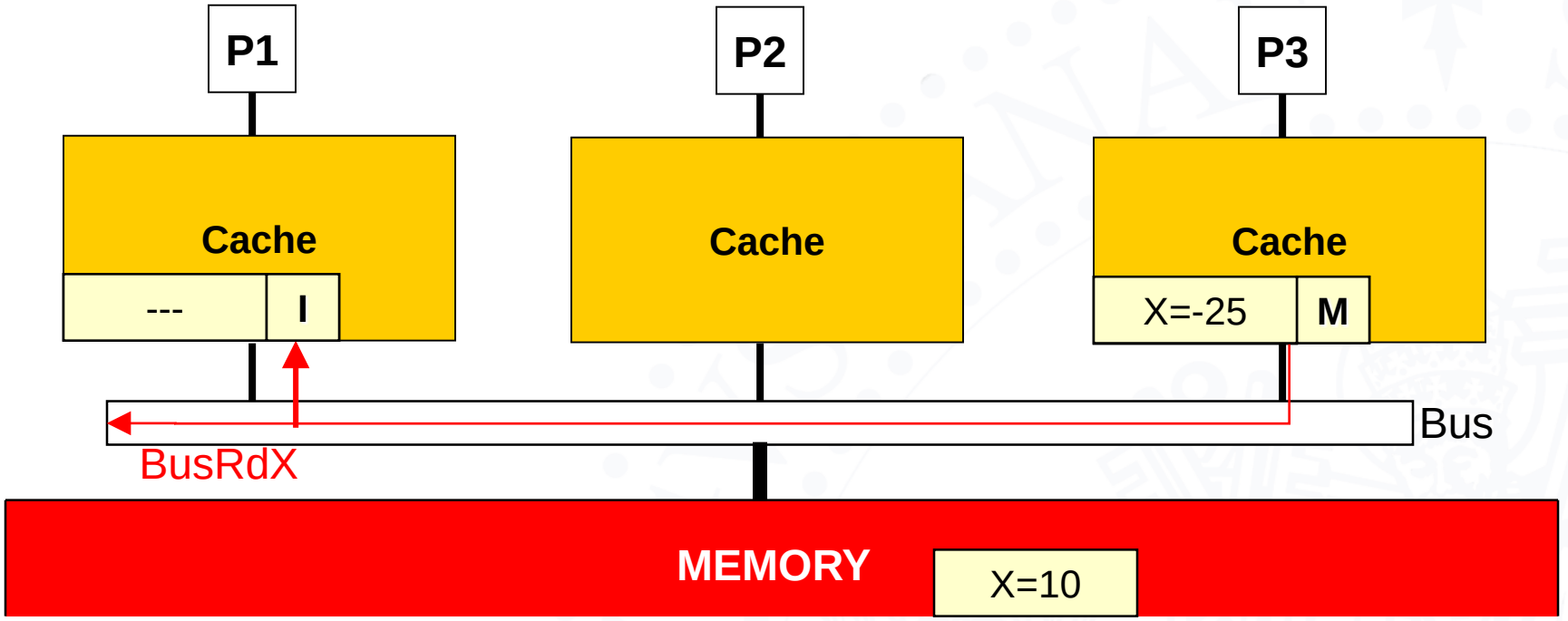
Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory

Ejemplo MSI → trabajarlo con los nombres de comandos de la pag 24.



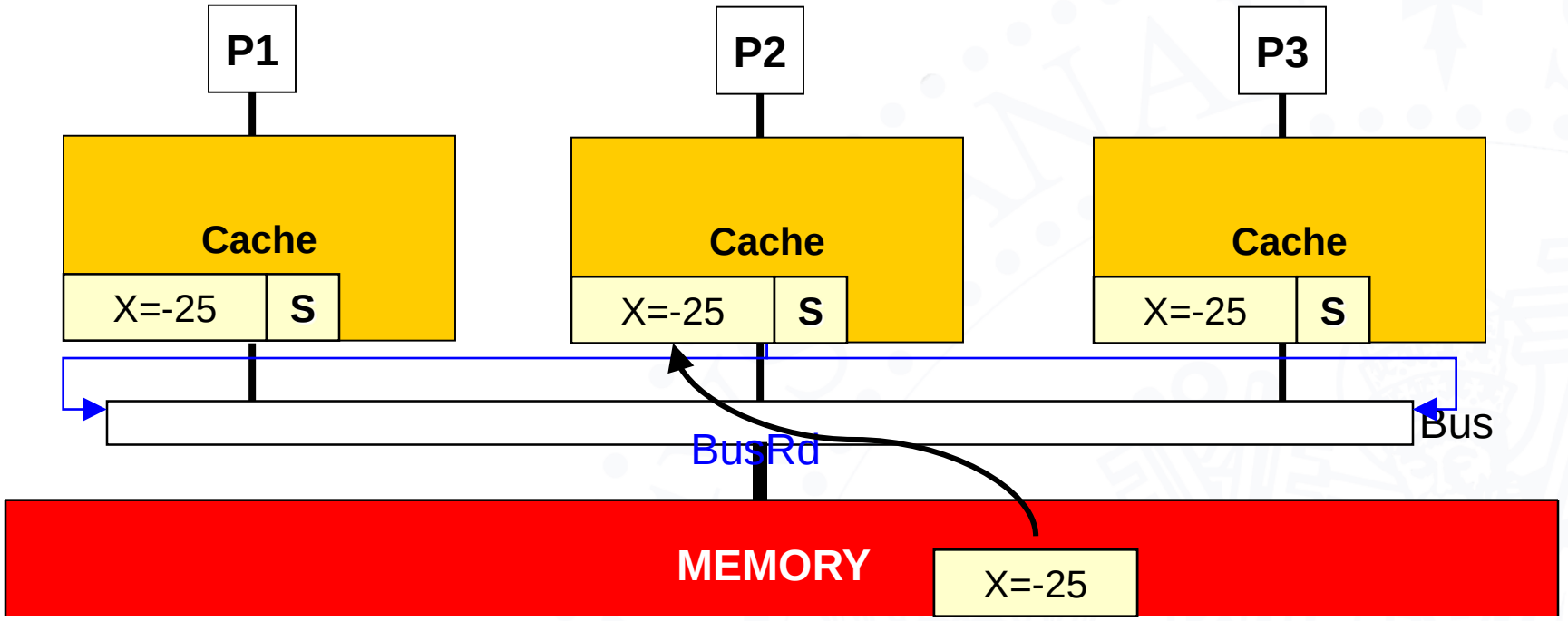
Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory
P3 reads X	S	---	S	BusRd	Memory

Ejemplo MSI → trabajarlo con los nombres de comandos de la pag 24.



Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory
P3 reads X	S	---	S	BusRd	Memory
P3 writes X	I	---	M	BusRdX	

Ejemplo MSI → trabajarlo con los nombres de comandos de la pag 26



Processor Action	State in P1	State in P2	State in P3	Bus Transaction	Data Supplier
P1 reads X	S	---	---	BusRd	Memory
P3 reads X	S	---	S	BusRd	Memory
P3 writes X	I	---	M	BusRdX	
P1 reads X	S	---	S	BusRd	P3 Cache
P2 reads X	S	S	S	BusRd	Memory