

2º curso / 2º cuatr.
Grado en
Ing. Informática

Arquitectura de Computadores

Tema 3

Arquitecturas con paralelismo a nivel de thread (TLP)

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad
de Granada



Arquitecturas con DLP, ILP y TLP (thread=flujo de control o de instrucciones)



Arq. con **DLP**
(*Data Level Parallelism*)

Ejecutan las **operaciones** de una instrucción **concurr.** o en **paralelo**

Unidades **funcionales** vectoriales o SIMD (90)

Arq. con **ILP**
(*Instruction Level Parallelism*)

Temas[1,4], BP4

Ejecutan múltiples **instrucciones** **concurr.** o en **paralelo**

Cores escalares (60) segmentados (80), superescalares (90) o VLIW/EPIC (90)

Arq. con **TLP** (*Thread Level Parallelism*) explícito y **una** instancia de SO

Temas[1-3], BP[0-3]

Ejecutan múltiples **flujos de instrucciones** **concurr.** o en **paralelo**

Cores que modifican la arquit. ILP para ejecutar threads **concurr.** o en **paralelo** (2000)

Multi-procesadores (60): ejecutan threads en *paralelo* en un computador con múltiples cores (incluye **multicores (2000)**)

Arq. con **TLP** explícito y **múltiples** instancias SO

Ejec. múltiples **flujos de instr.** en **paralelo**

Multi-computadores (85): ejecutan threads en *paralelo* en un sistema con múltiples computadores

Nota histórica. TLP (*Thread Level Parallelism*)

➤ TLP explícito con una instancia de SO:

TEMA 3

➤ Multithread grano fino (FGMT)

■ 1975 (Denelcor HEP). **2005** (chip Sun UltraSPARC T1) ...

➤ Multithread grano grueso (CGMT)

■ 1990 (MIT Alewife). **2000** (chip IBM PowerPC RS64 IV (2)). **2006** (chip Intel Itanium Montecito (2)) ...

➤ Multithread simultánea (SMT)

■ **2002** (chip Intel Pentium 4/Xeon Hyper-Threading). **2004** (chip IBM Power5) ...

➤ Multiprocesadores en un chip (CMP) o multicores

■ **2001** (chip IBM Power4). **2004** (chip Sun UltraSPARC IV). **2006** (chip Intel Core Duo). **2008** (chip Intel Celeron Dual-core) ...

➤ Multiprocesadores

■ 1962 (Burroughs D825 - red barras cruzadas). 1966 (UNIVAC 1108 - red bus). 1985 (IBM RP3 - red multietapa). 1996 (SGI Origin 2000 -CC-NUMA, red estática+multietapa). **2006** (SGI Altix 4000) ...

➤ TLP explícito con múltiples instancias del SO (multicomputadores) **IC.SCAP**

■ 1985 (Intel iPSC1 - i286+red estática con Ethernet) ... cualquier cluster

NOTA: Destacado en *cursiva* las primeras implementaciones y en **color** más claro los chip de propósito específico

Lecciones

- Lección 7. Arquitecturas TLP
 - Clasificación de arquitecturas con TLP explícito y una instancia del SO
 - Multiprocesadores
 - Multicores
 - Cores Multithread
 - Hardware y arquitecturas TLP en un chip
- Lección 8. Coherencia del sistema de memoria
- Lección 9. Consistencia del sistema de memoria
- Lección 10. Sincronización

Objetivos Lección 7

- Distinguir entre cores multithread, multicores y multiprocesadores.
- Comparar entre cores multithread de grano fino, cores multithread de grano grueso y cores con multithread simultánea.

Bibliografía Lección 7

➤ Fundamental

- T. Rauber, G. Ründer. *Parallel Programming: for Multicore and Cluster Systems*. Springer 2010. Disponible en línea (biblioteca UGR): <http://dx.doi.org/10.1007/978-3-642-04818-0>
- Theo Ungerer, Borut Robic, Jurij Silc, “A survey of Processors with Explicit Multithreading”, ACM Computing Surveys, vol. 35, no. 1, pp. 29-63, March 2003. Disponible en línea (biblioteca UGR): <http://dl.acm.org/citation.cfm?id=641867>
- Capítulo 3, Sección 2. M. Anguita, J. Ortega. Fundamentos y problemas de Arquitectura de Computadores, Editorial Técnica Avicam. ESIIT/C.1 ANG fun

Contenido Lección 7

- Clasificación de arquitecturas con TLP explícito y una instancia del SO
- Multiprocesadores
- Multicores
- Cores Multithread
- Hardware y arquitecturas TLP en un chip

Clasificación de arquitecturas con TLP explícito y una instancia de SO

- Multiprocesador
 - Ejecutan varios threads en paralelo en un **computador** con varios cores/procesadores (cada thread en un core/procesador distinto).
 - Diversos niveles de empaquetamiento: dado, encapsulado, placa, chasis y sistema.
- Multicore o multiprocesador en un chip o CMP (*Chip MultiProcessor*)
 - Ejecutan varios threads en paralelo en un **chip de procesamiento multicore** (cada thread en un core distinto)
- Core multithread
 - **Core** que modifica su arquitectura ILP para ejecutar threads concurrentemente o en paralelo



Contenido Lección 7

- Clasificación de arquitecturas con TLP explícito y una instancia del SO
- Multiprocesadores
 - Ejecutan varios threads en *paralelo* en un **computador** con varios cores/procesadores (cada thread en un core/procesador distinto).
- Multicores
- Cores Multithread
- Hardware y arquitecturas TLP en un chip

Multiprocesadores. Criterio de clasificación: nivel de empaquet./conexión

Sistema



SGI Altix 4700

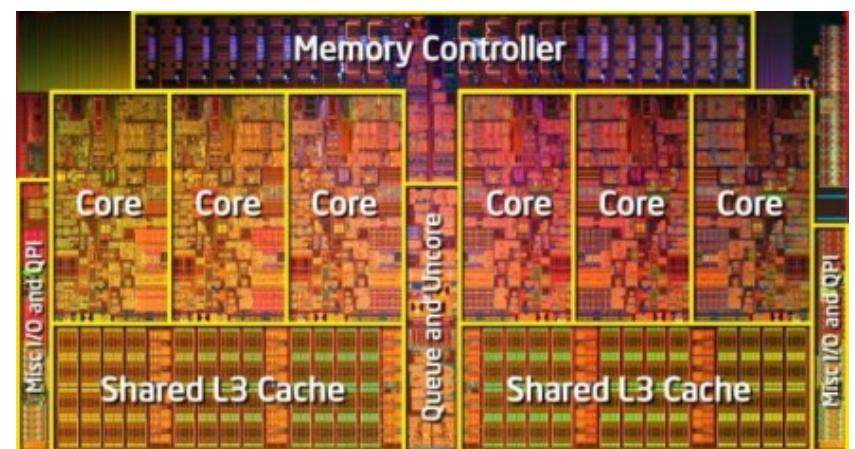
<http://www.sgi.com/products/remarketed/servers/altix4700.html>

Armario
(*cabinet*)

Placa
(*board*)
chip



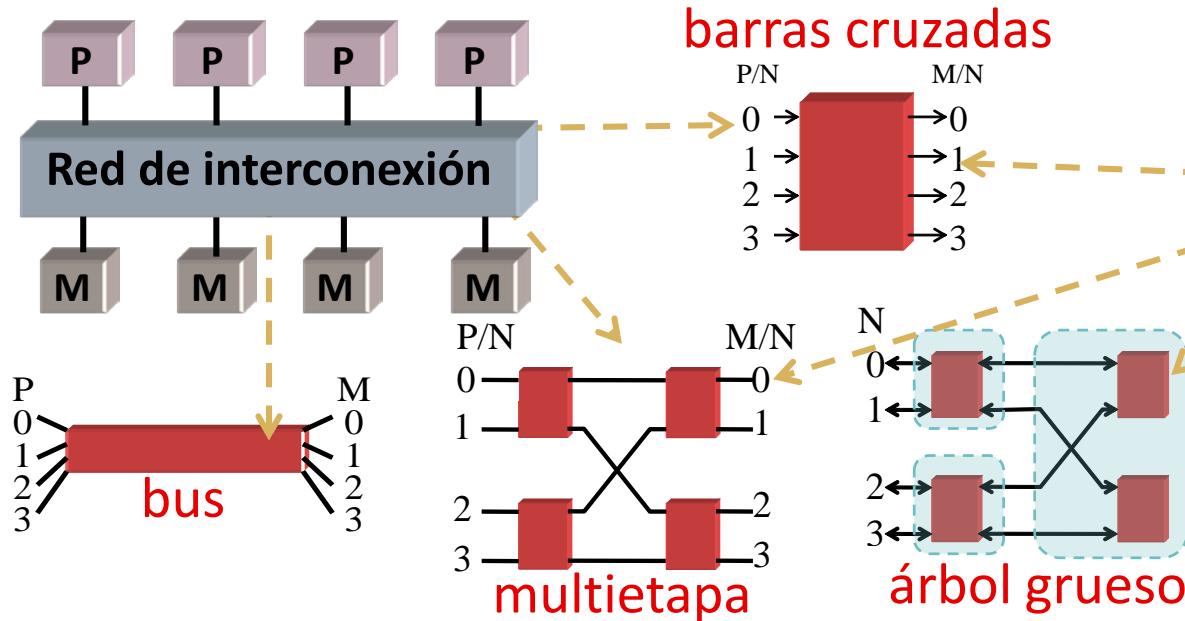
Multicore



Multiprocesadores. Criterio clasificación: sistema de memoria (Lección 2)

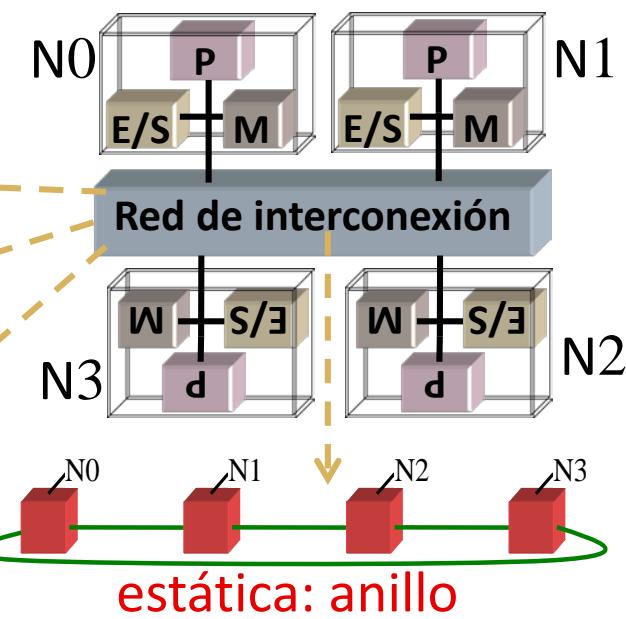
Multiprocesador con memoria centralizada (UMA) (60)

- Mayor latencia - Poco escalable

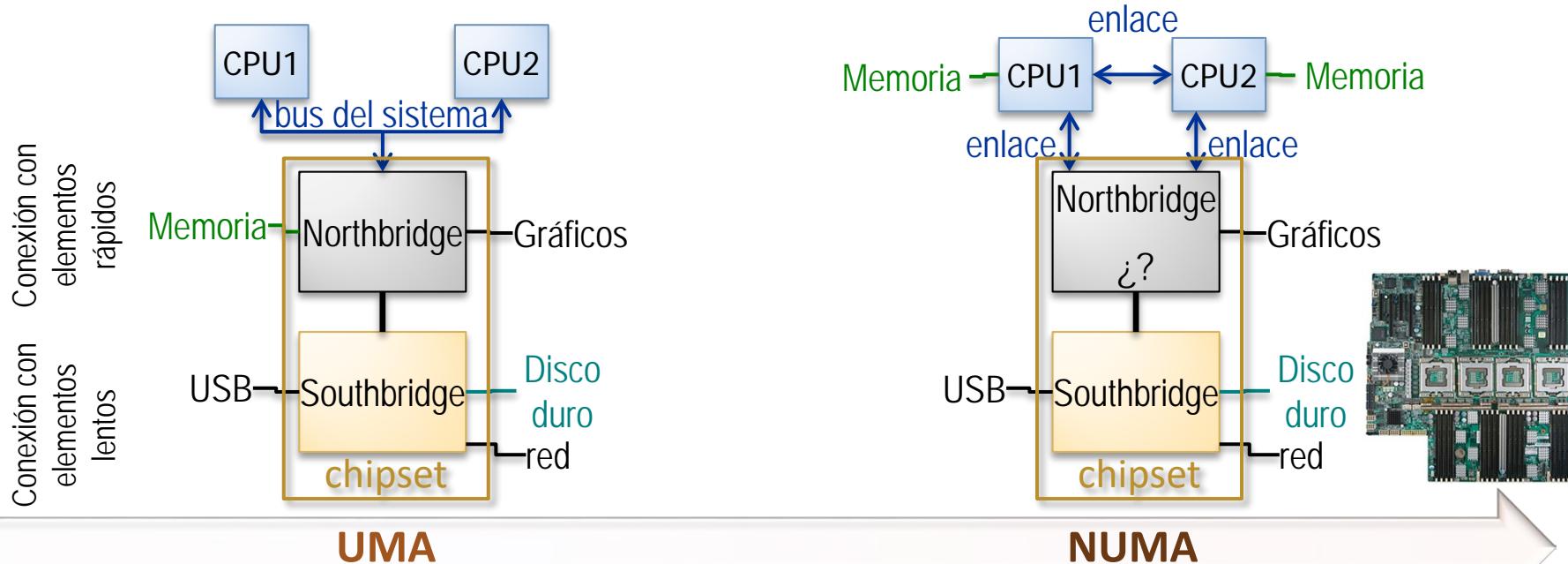


Multiprocesador con memoria distribuida (NUMA) (90)

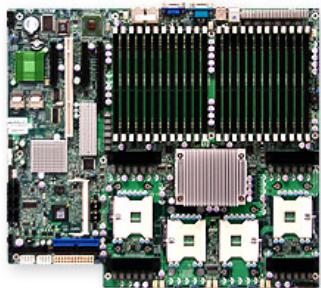
- Menor latencia - escalable pero requiere para ello distribución de datos/código



Multiprocesador en una placa: evolución de UMA a NUMA

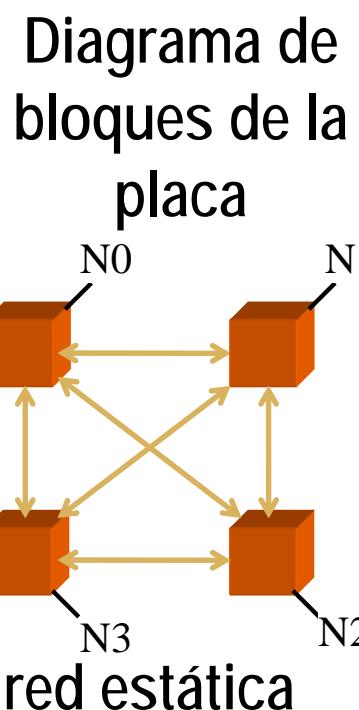
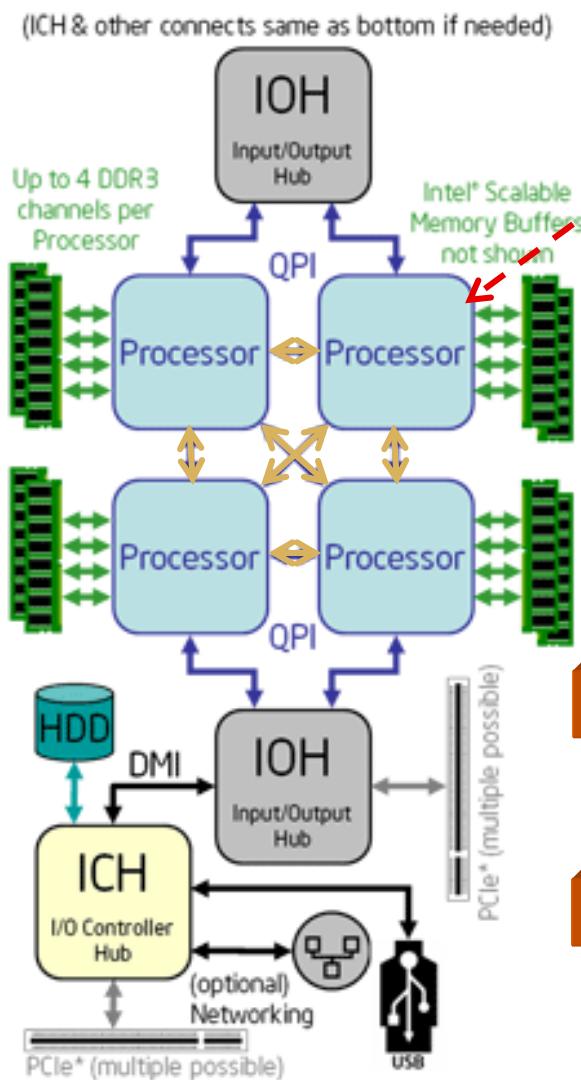


- ❖ Controlador de memoria en **chipset** (*Northbridge* chip)
- ❖ Red: bus (medio compartido)



- ❖ Controlador de memoria en chip del procesador
- ❖ Red: enlaces (conexiones punto a punto) y conmutadores (en el chip del procesador)
- ❖ Ejemplos en servidores:
 - AMD Opteron (2003): enlaces HyperTransport (2001)
 - Intel (Nehalem) Xeon 7500 (2010): enlaces QPI (*Quick Path Interconnect*, 2008)

Multiprocesador en una placa: CC-NUMA con red estática (Intel Xeon 7500)



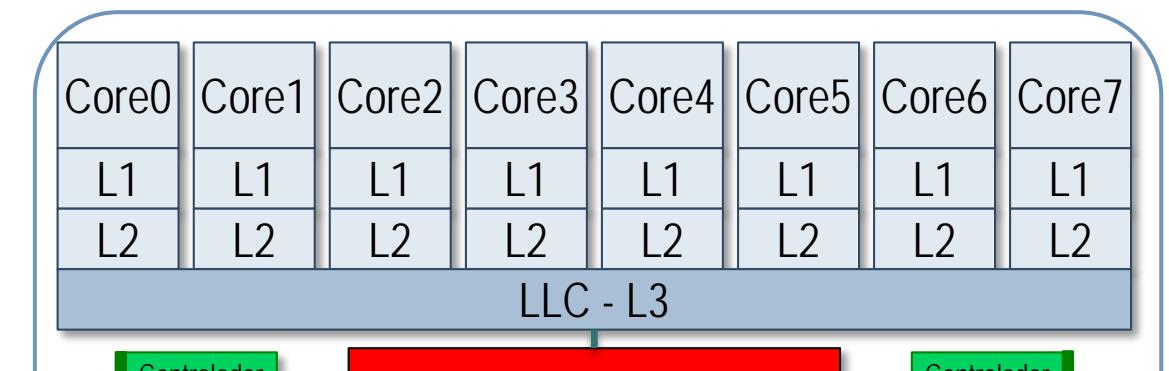
Placa para Intel Xeon 7500

Contenido Lección 7

- Clasificación y estructura de arquitecturas con TLP explícito y una instancia del SO
- Multiprocesadores
- Multicores
 - Ejecutan varios threads en paralelo en un **chip de procesamiento** multicore (cada thread en un core distinto)
- Cores Multithread
- Hardware y arquitecturas TLP en un chip

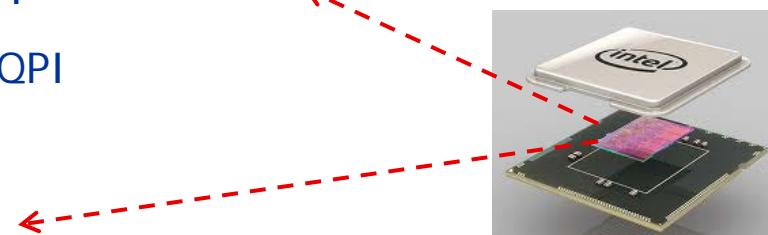
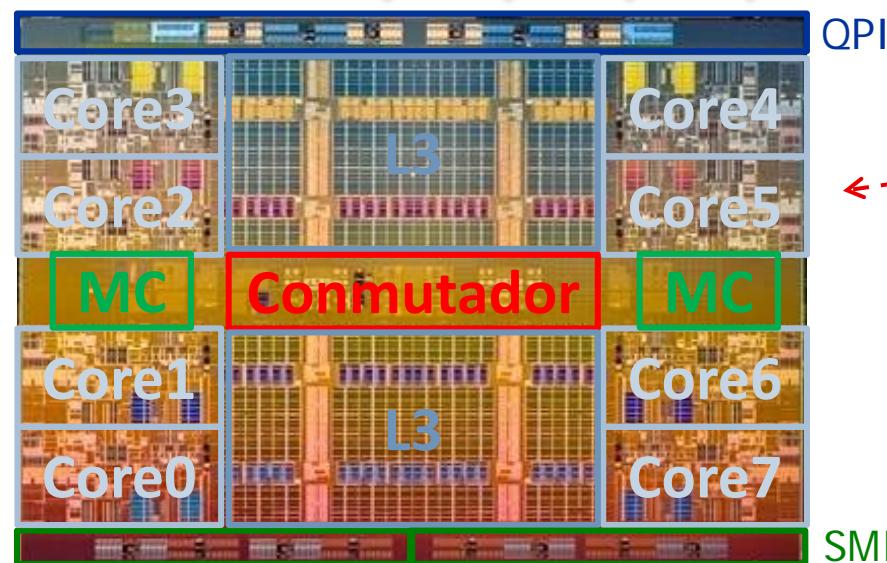
Multiprocesador en un chip o Multicore o CMP (*Chip MultiProcessor*) (2000)

Intel Xeon 7500



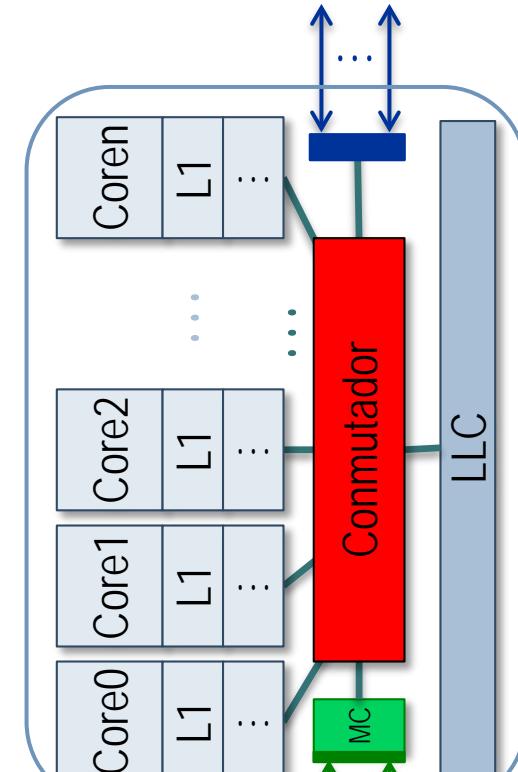
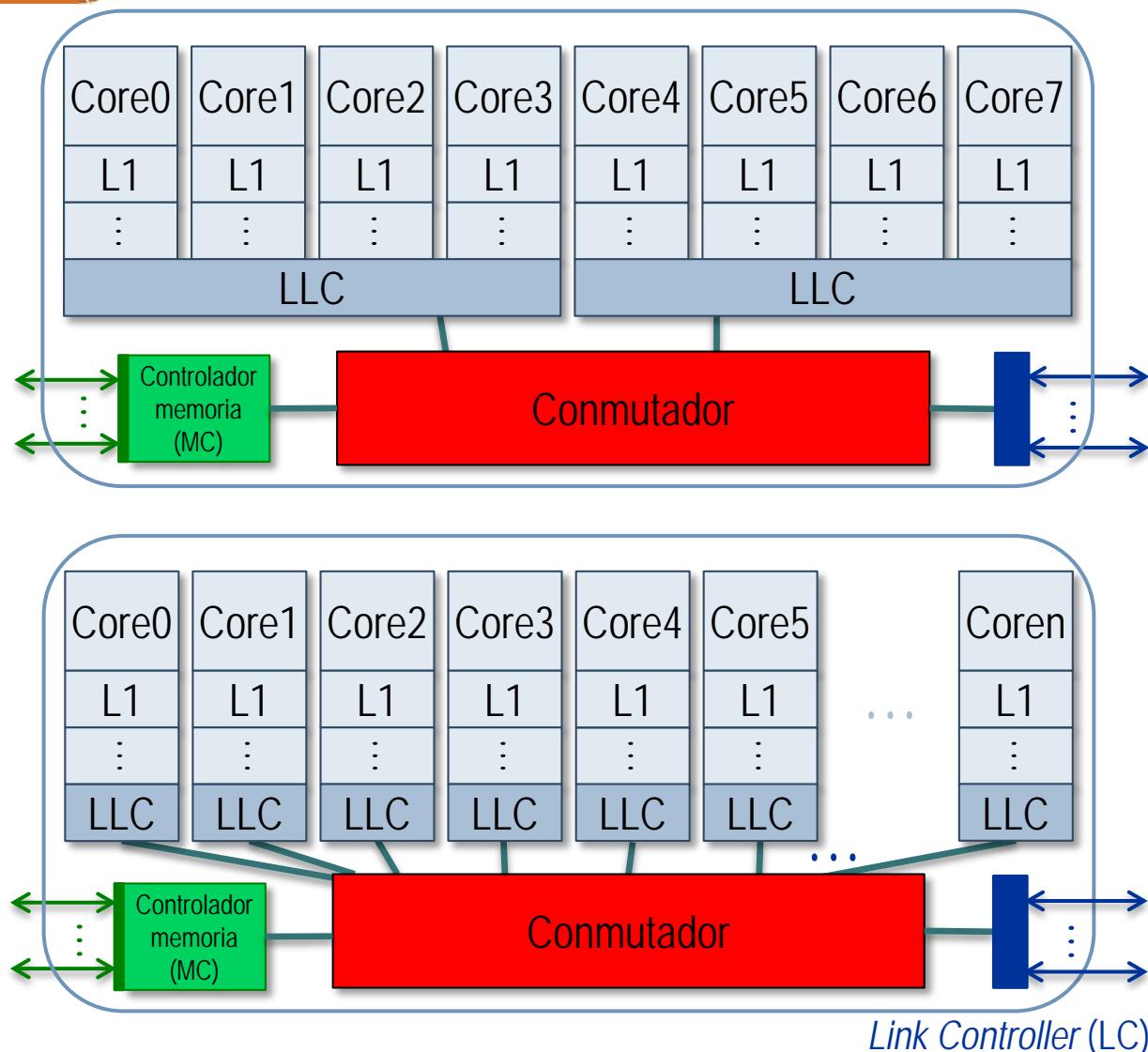
32 KB Icaché + 32KB Dcaché
8x256 KB
12 a 24 MB

Diagrama de bloques
del chip



Chip o dado de silicio del chip de
procesamiento Intel Xeon 7500

Multicore: otras posibles estructuras



Contenido Lección 7

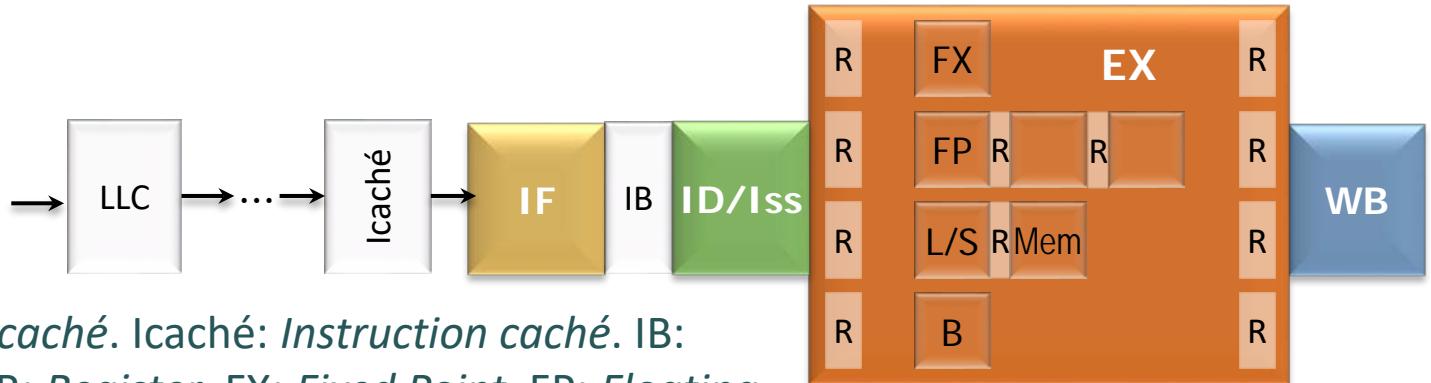
- Clasificación y estructura de arquitecturas con TLP explícito y una instancia del SO
- Multiprocesadores
- Multicores
- Cores Multithread
 - Modifican su arquitectura ILP (segmentada, superescalar o VLIW) para ejecutar threads concurrentemente o en paralelo
- Hardware y arquitecturas TLP en un chip

Arquitecturas ILP

**Escalar
segmentada**



**VLIW y
superescalar**



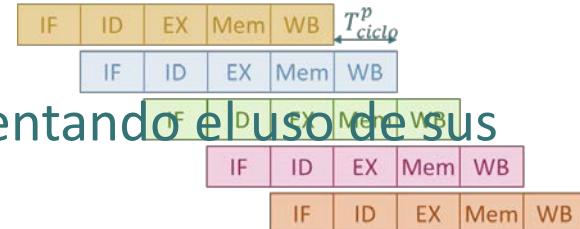
LLC: *Last Level caché*. Icaché: *Instruction caché*. IB: *Instruction Buffer*. R: *Register*. FX: *Fixed Point*. FP: *Floating Point*. L/S: *(memory) Load/Store*. B: *Branch*

- Etapa de captación de instrucciones (***Instruction Fetch***)
- Etapa de decodificación de instrucciones y emisión a unidades funcionales (***Instruction Decode/Instruction Issue***) (incluye captura de "operandos" de los registros de la arquitectura)
- Etapas de ejecución (***Execution***). Etapa de acceso a memoria (***Memory***)
- Etapa de almacenamiento de resultados en registros de la arquitectura(***Write-Back***)

Arquitecturas ILP

➤ Procesadores/cores segmentados:

- Ejecutan instrucciones **concurrentemente** segmentando el uso de sus componentes



➤ Procesadores/cores VLIW (*Very Large Instruction Word*) y superescalares:

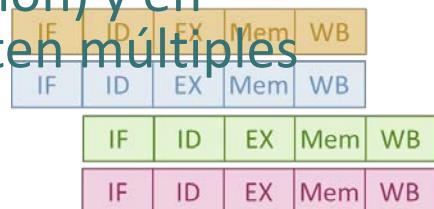
- Ejecutan instrucciones **concurrentemente** (segmentación) y en **paralelo** (tienen múltiples unidades funcionales y emiten múltiples instrucciones en paralelo a unidades funcionales)

■ VLIW:

- Las instrucciones que se ejecutan en paralelo se captan juntas de memoria.
- Este conjunto de instrucciones conforman la palabra de instrucción muy larga a la que hace referencia la denominación VLIW
- El hardware presupone que las instrucciones de una palabra son independientes: no tiene que encontrar instrucciones que pueden emitirse y ejecutarse en paralelo

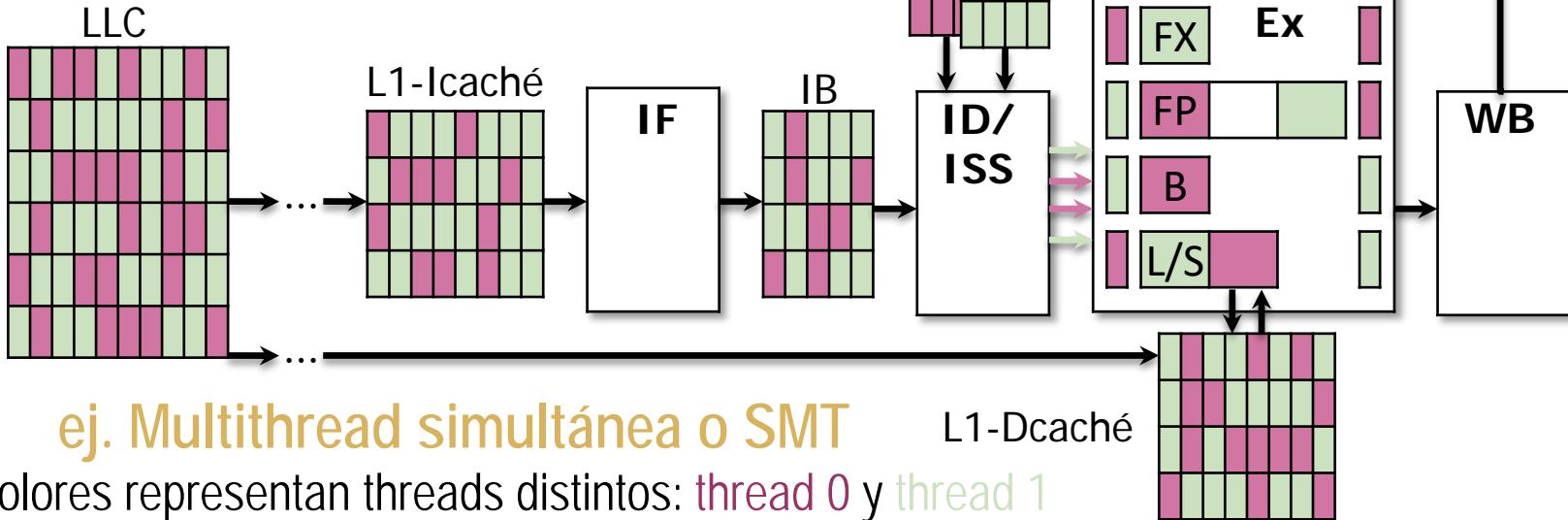
■ Superescalares:

- Tiene que encontrar instrucciones que puedan emitirse y ejecutarse en paralelo (tiene *hardware para extraer paralelismo a nivel de instrucción*)



Modificación de la arquitectura ILP en Core Multithread (ej. SMT)

LLC: *Last Level caché*. Icaché: *Instruction caché*. IB: *Instruction Buffer*. RF: *Register File*. FX: *Fixed Point*. FP: *Floating Point*. L/S: *(memory) Load/Store*. B: *Branch*



- Almacenamiento: se multiplexa, se reparte o comparte entre threads, o se replica
 - Con SMT: repartir, compartir o replicar
- Hardware dentro de etapas: se multiplexa, o se reparte o comparte entre threads
 - Con SMT: unidades funcionales (etapa Ex) compartidas, resto etapas repartidas o compartidas; multiplexación es posible (p. ej. predicción de saltos y decodificación)

Clasificación de cores multithread

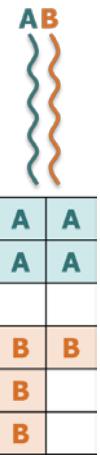
➤ *Temporal Multithreading (TMT)*

- Ejecutan varios threads **concurrentemente** en el mismo core
 - La comutación entre threads la decide y controla el hardware

- Emite instrucciones de **un único thread** en un ciclo

➤ *Simultaneous MultiThreading (SMT) o multihilo simultáneo o horizontal multithread*

- Ejecutan, en un core superescalar, varios threads en **paralelo**
- Pueden emitir (para su ejecución) instrucciones de **varios threads** en un ciclo
- No implementa comutación entre threads



Clasificación de cores con TMT



Ciclos



Ciclos

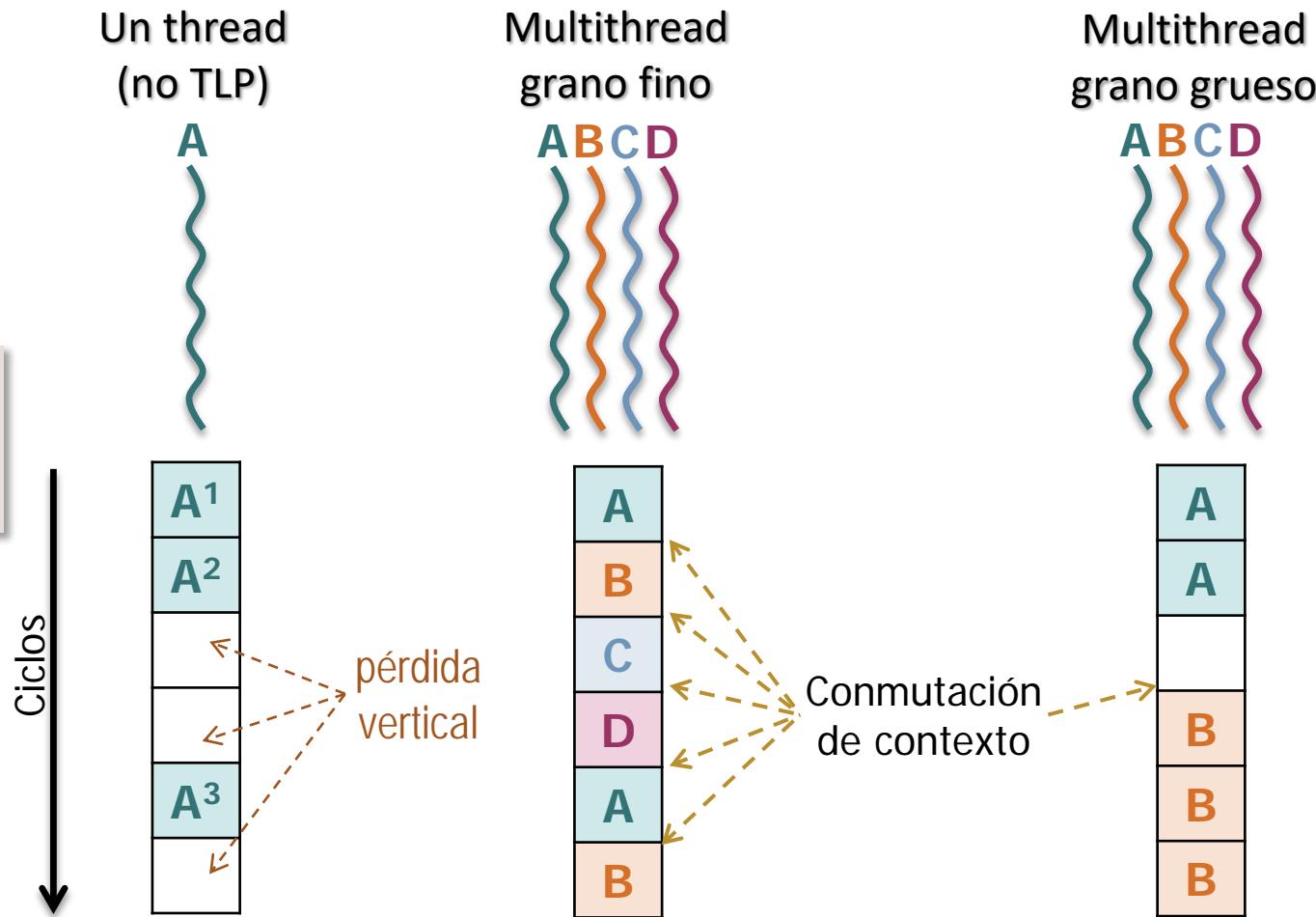
- *Fine-grain multithreading (FGMT) o interleaved multithreading*
- La conmutación entre threads la decide el hardware **cada ciclo** (coste 0)
 - por turno rotatorio (*round-robin*) o
 - por eventos de cierta latencia combinado con alguna técnica de planificación (ej. thread menos recientemente ejecutado)
 - Eventos: dependencia funcional, acceso a datos a caché L1, salto no predecible, una operación de cierta latencia (ej. div), ...
- *Coarse-grain multithreading (CGMT) o blocked multithreading*
- La conmutación entre threads la decide el hardware (coste de 0 a varios ciclos)
 - tras intervalos de tiempo prefijados (*timeslice multithreading*) o
 - por eventos de cierta latencia (*switch-on-event multithreading*).

Clasificación de cores con CGMT con comutación por eventos

- Estática:
 - Comutación
 - Explícita: instrucciones explícitas para comutación (instrucciones añadidas al repertorio)
 - Implícita: instrucciones de carga, almacenamiento, salto
 - Ventaja/Inconveniente:
 - coste cambio contexto bajo (0 o 1 ciclo) / cambios de contextos innecesarios
- Dinámica:
 - Comutación típicamente por:
 - fallo en la última caché dentro del chip de procesamiento (comutación por fallo de caché), interrupción (comutación por señal), ...
 - Ventaja/Inconveniente:
 - reduce cambios de contexto innecesarios / mayor sobrecarga al cambiar de contexto

Alternativas en un core escalar segmentado

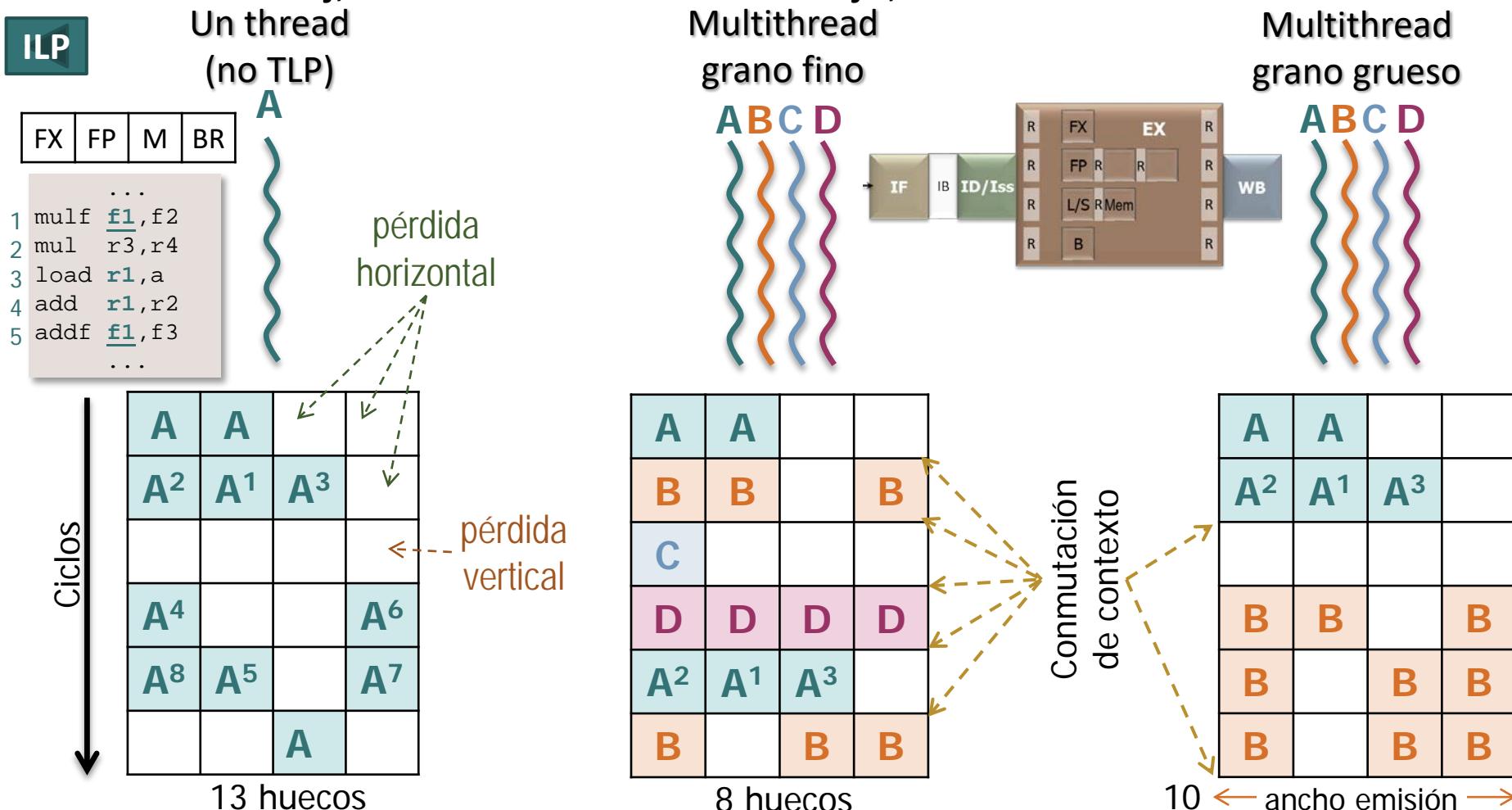
- En un core escalar se emite una instrucción cada ciclo de reloj



Alternativas en un core con emisión múltiple de instrucciones de un thread

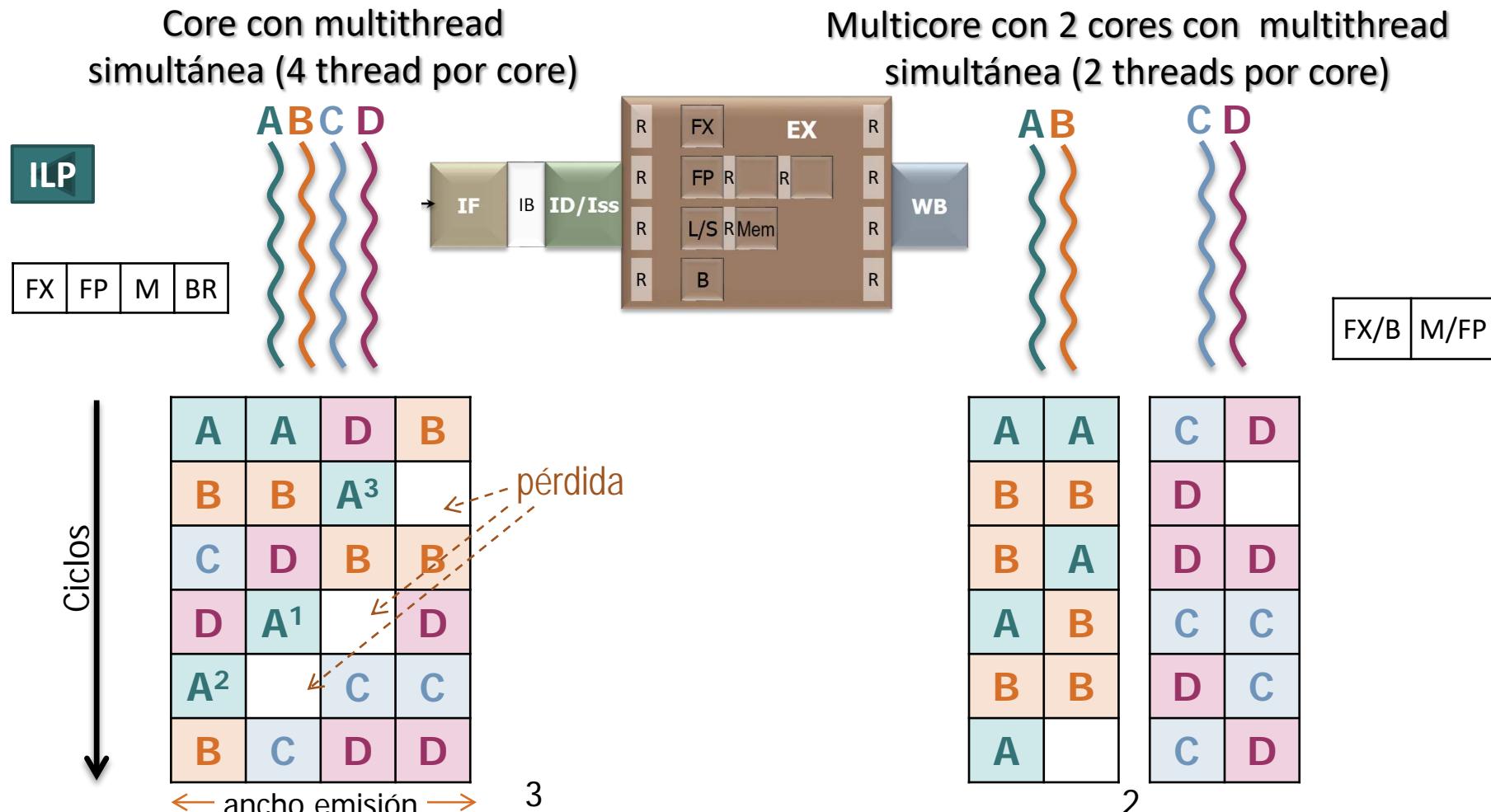
- En un core superescalar o VLIW se emiten más de una instrucción cada ciclo de reloj; en las alternativas de abajo, de un único thread

ILP



Core multithread simultánea y multicores

- En un multicore y en un core superescalar con SMT (*Simultaneous MultiThread*) se pueden emitir instrucciones de distintos threads cada ciclo de reloj



Contenido Lección 7

- Clasificación y estructura de arquitecturas con TLP explícito y una instancia del SO
- Multiprocesadores
- Multicores
- Cores Multithread
- Hardware y arquitecturas TLP en un chip

Hardware y arquitecturas TLP en un chip



Hardware	CGMT	FGMT	SMT	CMP
Registros de la arquitectura	replicado (al menos PC)	replicado	replicado	replicado
Almacenamiento	multiplexado	multiplexado, repartido, compartido o replicado	repartido, compartido o replicado	replicado
Otro hardware de las etapas del cauce	multiplexado	<u>Captación:</u> repartida o compartida; <u>Resto:</u> multiplexadas	<u>UF:</u> compartidas; <u>Resto:</u> repartidas o compartidas	replicado
Etiquetas para distinguir el thread de una instr.	Sí	Sí	Sí	No
Hardware para conmutar entre threads	Sí	Sí	No	No

Para ampliar ...

➤ Webs

- An Introduction to the Intel® QuickPath Interconnect,
<http://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html>

- Intel® QuickPath Technology Animated Demo [119 K]
https://www.youtube.com/watch?v=XPBX3cl_cIE

➤ Artículos en revistas

- Kongetira, P.; Aingaran, K.; Olukotun, K.; , "Niagara: a 32-way multithreaded Sparc processor," *Micro, IEEE* , vol.25, no.2, pp. 21-29, March-April 2005. Disponible en línea (biblioteca UGR): <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1453485&isnumber=31213>

2º curso / 2º cuatr.
Grado en
Ing. Informática

Arquitectura de Computadores

Tema 3

Arquitecturas con paralelismo a nivel de thread (TLP)

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad
de Granada



Lecciones

- Lección 7. Arquitecturas TLP
- Lección 8. Coherencia del sistema de memoria
 - Sistema de memoria en multiprocesadores
 - Concepto de coherencia en el sistema de memoria: situaciones de incoherencia y requisitos para evitar problemas en estos casos
 - Protocolos de mantenimiento de coherencia: clasificación y diseño
 - Protocolo MSI de espionaje
 - Protocolo MESI de espionaje
 - Protocolo MSI basado en directorios con o sin difusión
- Lección 9. Consistencia del sistema de memoria
- Lección 10. Sincronización

Objetivos Lección 8

- Comparar los métodos de actualización de memoria principal implementados en caché.
- Comparar las alternativas para propagar un escritura en protocolos de coherencia de caché.
- Explicar qué debe garantizar el sistema de memoria para evitar problemas por incoherencias.
- Describir las partes en las que se puede dividir el análisis o el diseño de protocolos de coherencia.
- Distinguir entre protocolos basados en directorios y protocolos de espionaje (snoopy).
- Explicar el protocolo de mantenimiento de coherencia de espionaje MSI.
- Explicar el protocolo de mantenimiento de coherencia de espionaje MESI.
- Explicar el protocolo de mantenimiento de coherencia MSI basado en directorios con difusión y sin difusión.

Bibliografía Lección 8

➤ Fundamental

- Cap. 3, Secc. 3. M. Anguita, J. Ortega. *Fundamentos y Problemas de Arquitectura de Computadores*. Librería Fleming/Ed. Avicam, 2016.
- Secc. 10.1. J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*. Thomson, 2005. ESII/C.1 ORT arq

➤ Complementaria

- T. Rauber, G. Ründer. *Parallel Programming: for Multicore and Cluster Systems*. Springer 2010. Disponible en línea (biblioteca UGR): <http://dx.doi.org/10.1007/978-3-642-04818-0>

Computadores que implementan en hardware mantenimiento de coherencia

Multi-computadores
Memoria no compartida

NORMA
No Remote Memory Access

nivel de sistema (*Cluster*), armario, chasis (*blade server*)

Multi-procesadores
Memoria compartida
Un único espacio de direcciones

NUMA
Non-Uniform Memory Access

NUMA (nivel de sistema, n. armario/chasis, n. placa)

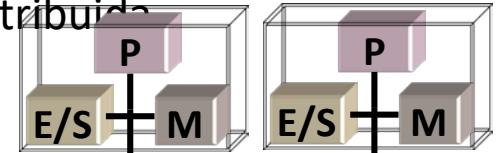
CC-NUMA (nivel de armario: SGI Altix; nivel de placa)

COMA

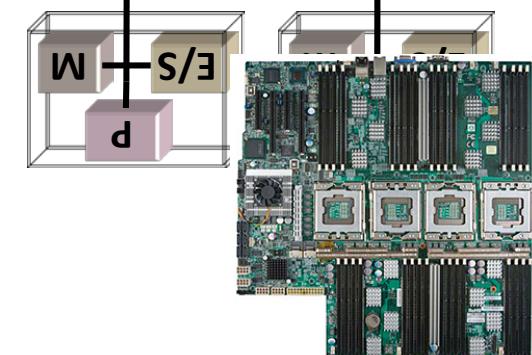
UMA
Uniform Memory Access

Coherencia por hardware
SMP Symmetric MultiProcessor (nivel de placa; nivel de chip: multicores como Intel Core i7, i5, i3)

Memoria físicamente distribuida



Red de interconexión

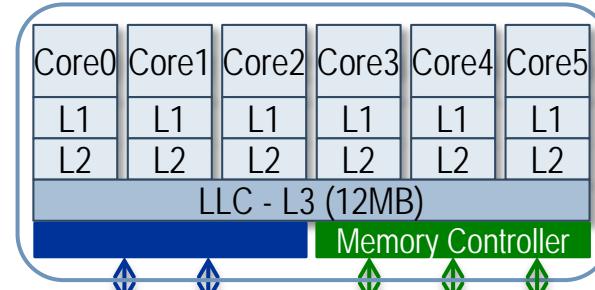


+

Escalabilidad

-

Memoria físicamente centralizada



Contenido Lección 8

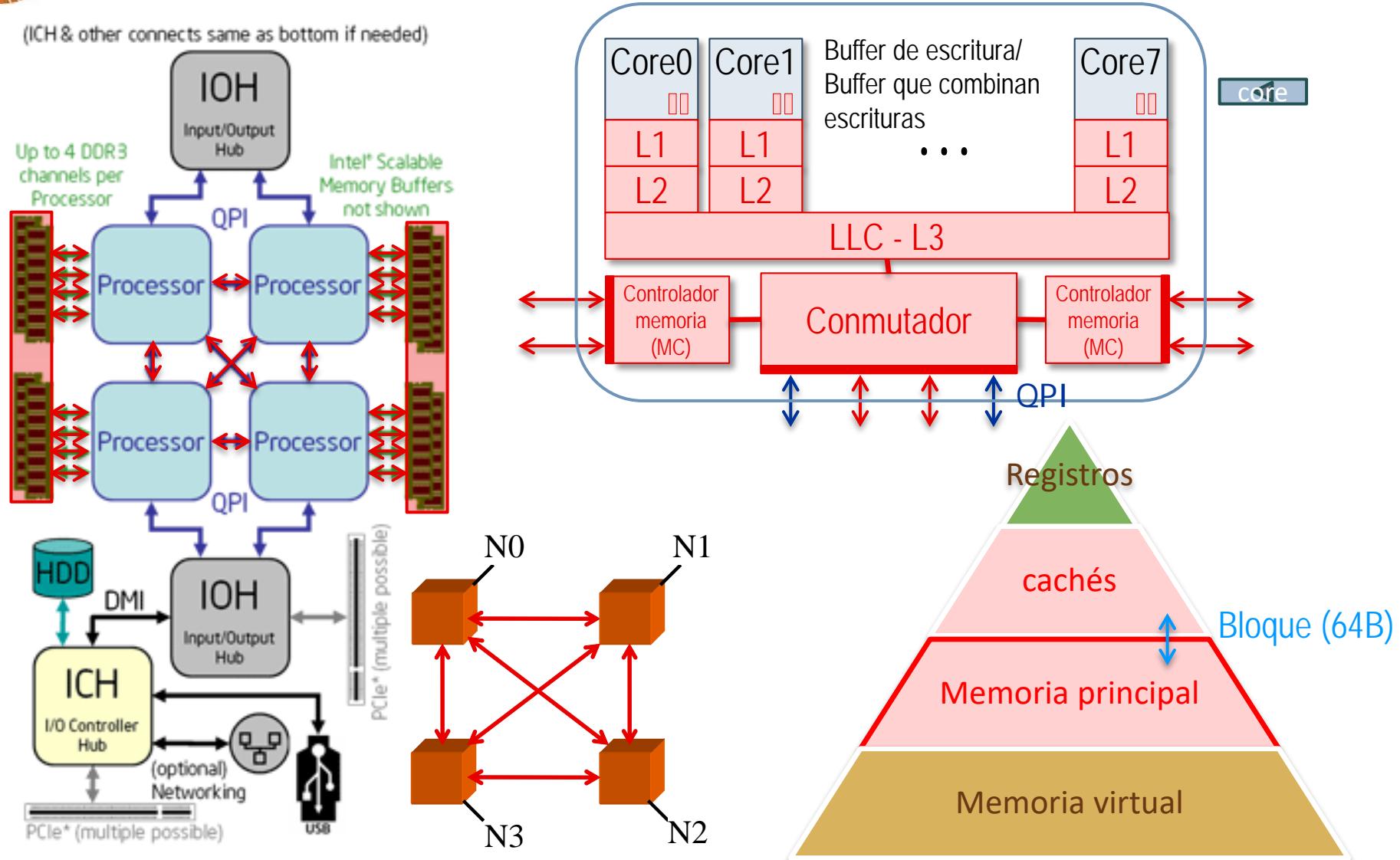
- Sistema de memoria en multiprocesadores
- Concepto de coherencia en el sistema de memoria:
situaciones de incoherencia y requisitos para evitar
problemas en estos casos
- Protocolos de mantenimiento de coherencia:
clasificación y diseño
- Protocolo MSI de espionaje
- Protocolo MESI de espionaje
- Protocolo MSI basado en directorios con o sin difusión

Sistema de memoria en multiprocesadores

- ¿Qué incluye?
 - cachés de todos los nodos
 - Memoria principal
 - Controladores
 - Buffers:
 - Buffer de escritura/almacenamiento
 - Buffer que combinan escrituras/almacenamientos, etc.
 - Medio de comunicación de todos estos componentes (red de interconexión)
- La comunicación de datos entre procesadores la realiza el sistema de memoria
 - La lectura de una dirección debe devolver lo último que se ha escrito (desde el punto de vista de todos los componentes del sistema)

comunicación

Sistema de memoria

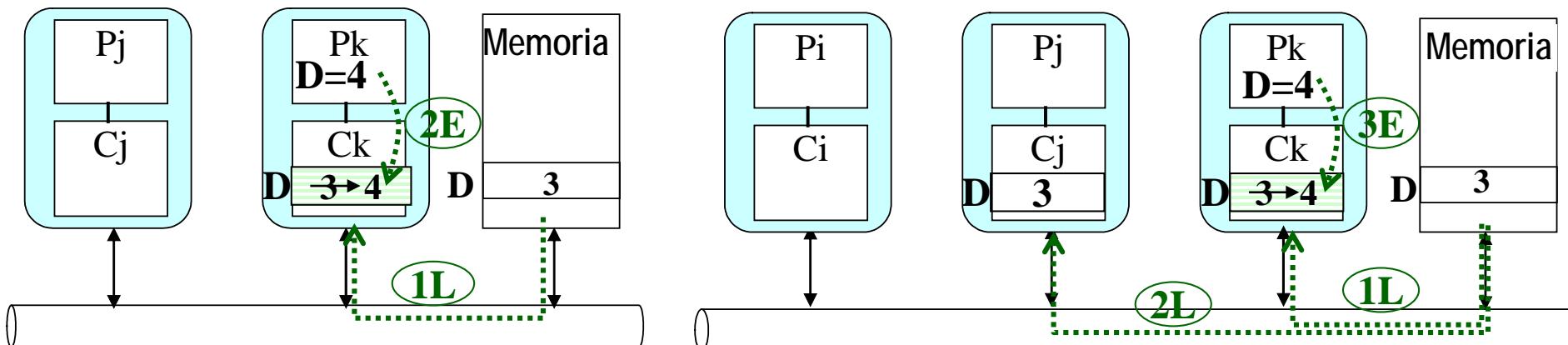


Contenido Lección 8

- Sistema de memoria en multiprocesadores
- Concepto de coherencia en el sistema de memoria:
situaciones de incoherencia y requisitos para evitar
problemas en estos casos
- Protocolos de mantenimiento de coherencia:
clasificación y diseño
- Protocolo MSI de espionaje
- Protocolo MESI de espionaje
- Protocolo MSI basado en directorios con o sin difusión

Incoherencia en el sistema de memoria

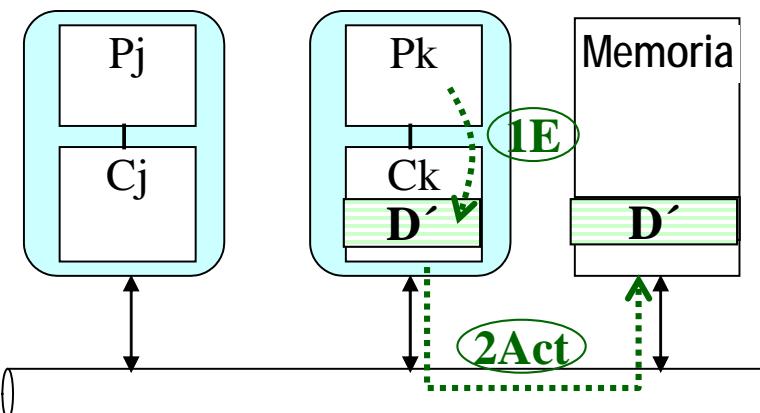
Clases de estructuras de datos	Eventos que ponen de manifiesto faltas de coherencia	Tipos de Falta de coherencia
Datos modificables	E/S	caché-MP
Datos modificables compartidos	Fallo de caché	caché-MP
Datos modificables privados	Emigra thread/proceso → Fallo caché	caché-MP
Datos modificables compartidos	Lectura de caché no actualizada	caché-caché



Métodos de actualización de memoria principal implementados en cachés

1. Escritura inmediata (*write-through*):

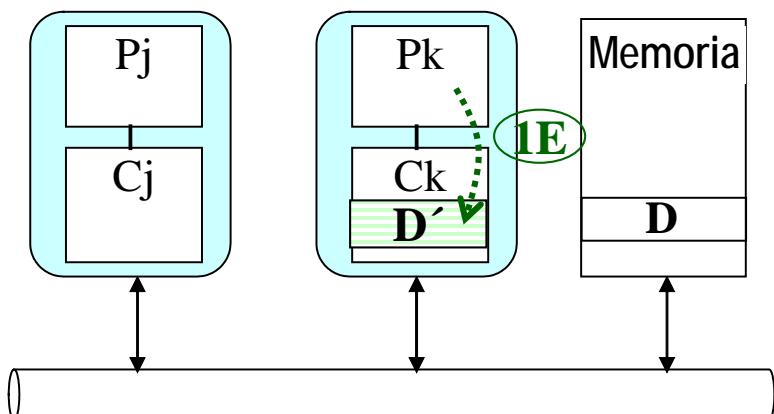
core



Cada vez que un procesador escribe en su caché escribe también en memoria principal

Por los principios de **localidad temporal** y **espacial** sería más rentable si se escribe todo el bloque en MP una vez realizadas las múltiples escrituras

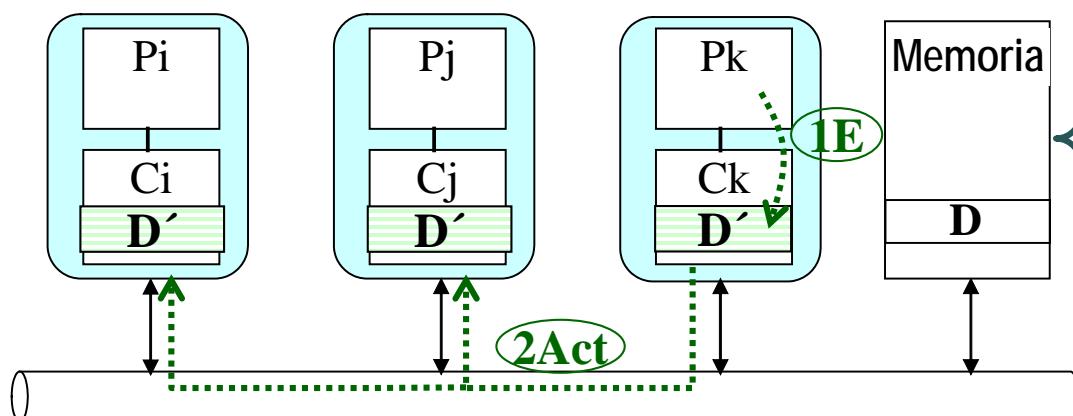
2. Posescritura (*write-back*):



Se actualiza memoria principal escribiendo todo el bloque cuando se **desaloja** de la caché

Alternativas para propagar una escritura en protocolos de coherencia de caché

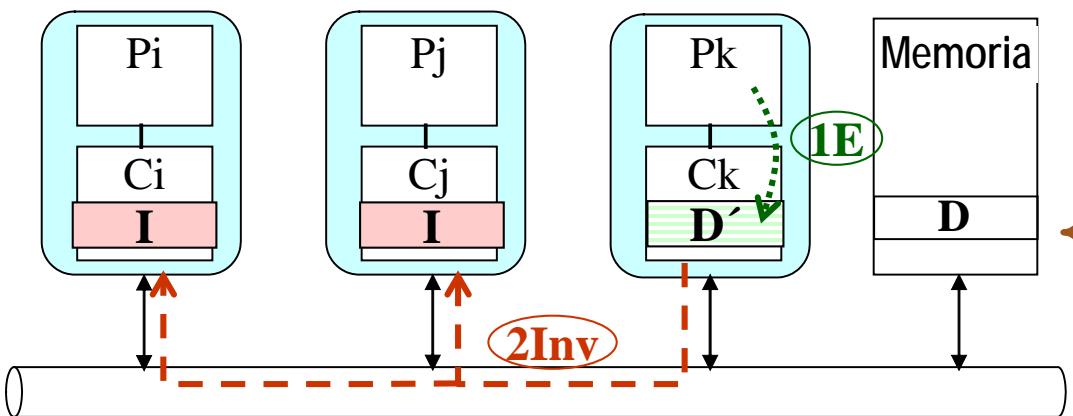
1. Escritura con actualización (*write-update*):



Cada vez que un procesador escribe en una dirección en su caché se escribe en las copias de esa dirección en otras cachés

Para reducir tráfico, sobre todo si los datos están compartidos por pocos procesadores

2. Escritura con invalidación (*write-invalidate*):



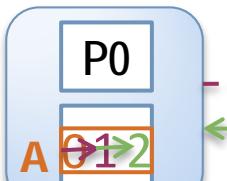
Antes que un procesador modifique una dirección en su caché se invalidan las copias del bloque de la dirección en otras cachés

Situación de incoherencia aunque se propagan las escrituras (usa difusión)

AC ATC

Orden para P0

- 1) A=1 (P0-1)
- 2) A=2 (P1-4)

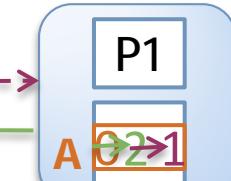


A=1

A=2

Orden para P1

- 1) A=2 (P1-2)
- 2) A=1 (P0-3)



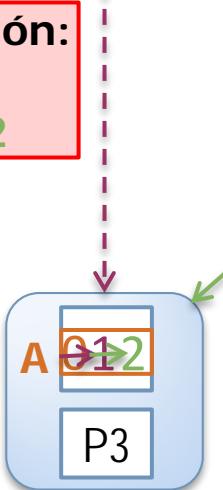
A=1

A=2

A=2

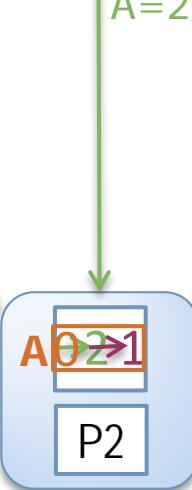
Orden generación:

- P0-1 1) A=1
P1-2 2) A=2



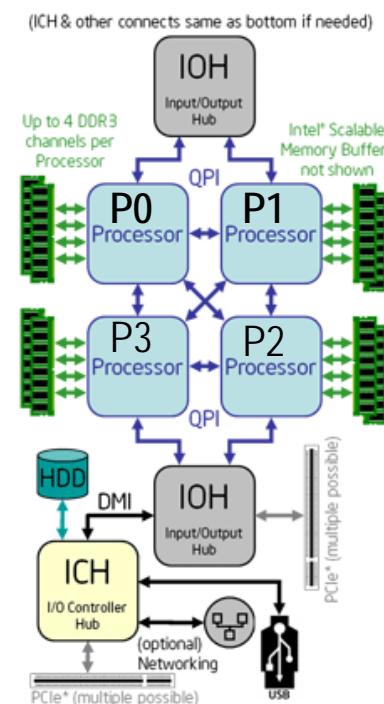
Orden para P3

- 1) A=1 (P0-3)
- 2) A=2 (P1-6)



Orden para P2

- 1) A=2 (P1-4)
- 2) A=1 (P0-5)

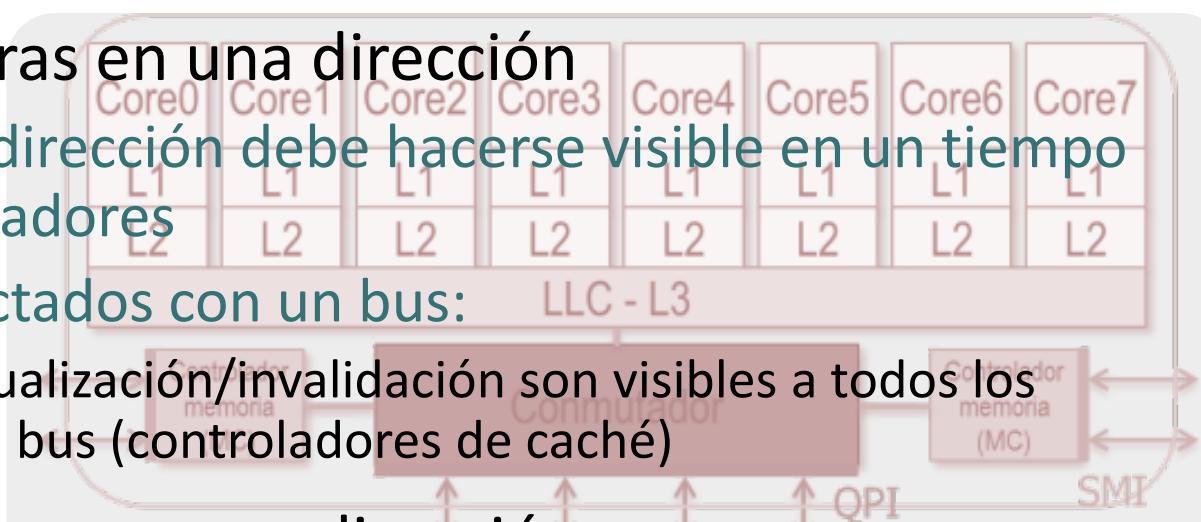


- Contenido inicial de las copias de la dirección A en **calabaza**. P0 escribe en A un 1 y, después, P1 escribe en A un 2.
- Se utiliza **actualización** para propagar las escrituras (las propagación se nota con flechas)
- Llegan en distinto orden las escrituras debido al distinto tiempo de propagación (**se está suponiendo que los Proc. están ubicados en la placa tal y como aparecen en el dibujo**). En **cursiva** se puede ver el contenido de las copias de la dirección A tras las dos escrituras.
 - Se da una situación de incoherencia aunque se propagan las escrituras: P0 y P3 acaban con 2 en A y P1 y P2 con 1.

Requisitos del sistema de memoria para evitar problemas por incoherencia I

➤ Propagar las escrituras en una dirección

- La escritura en una dirección debe hacerse visible en un tiempo finito a otros procesadores



➤ Componentes conectados con un bus:

- Los paquetes de actualización/invalidación son visibles a todos los nodos conectados al bus (controladores de caché)

➤ Serializar las escrituras en una dirección

- Las escrituras en una dirección deben verse en el mismo orden por todos los procesadores (el sistema de memoria debe **parecer** que realiza en serie las operaciones de escritura en la misma dirección)

➤ Componentes conectados con un bus:

- El orden en que los paquetes aparecen en el bus determina el orden en que se ven por todos los nodos.

Requisitos del SM para evitar problemas por incoherencia II: la red no es un bus

AC ATC

➤ Propagar escrituras en una dirección

➤ Usando difusión:

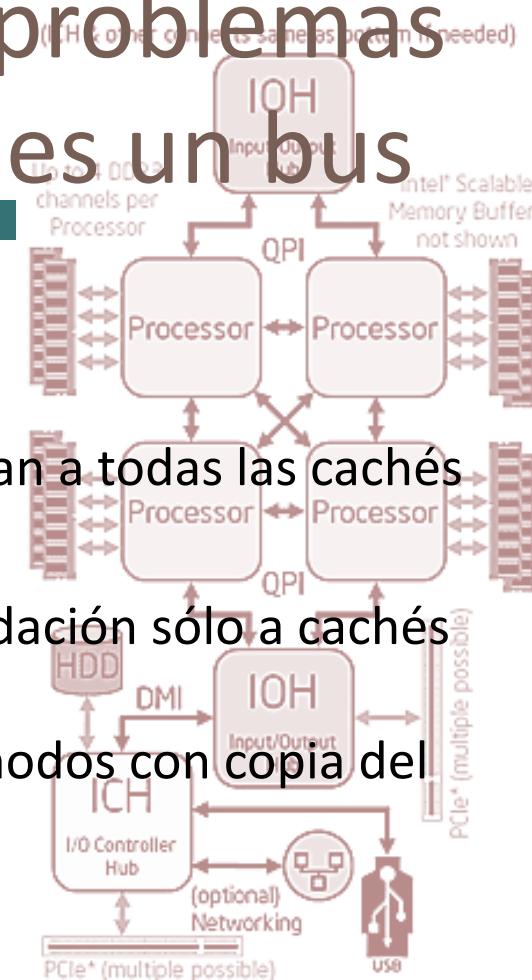
- Los paquetes de actualización/invalidación se envían a todas las cachés

➤ Para conseguir mayor escalabilidad:

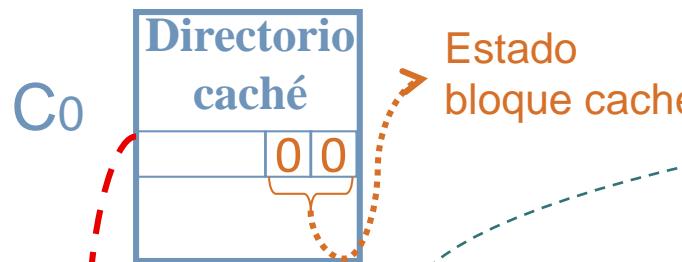
- Se debería enviar paquetes de actualización/invalidación sólo a cachés (nodos) con copia del bloque
- Mantener en un directorio, para cada bloque, los nodos con copia del mismo

➤ Serializar escrituras en una dirección

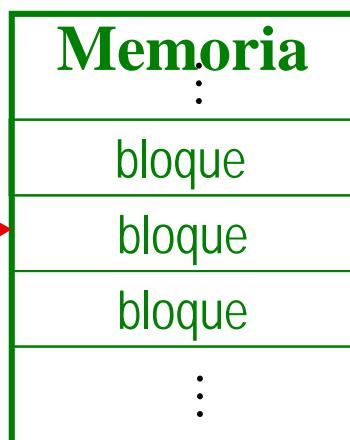
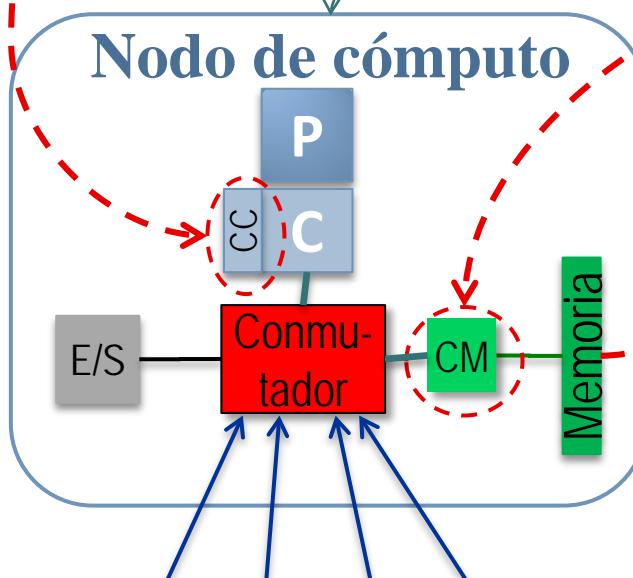
➤ El orden en el que las peticiones de escritura llegan a su *home* (nodo que tiene en MP la dirección) o al directorio centralizado sirve para serializar en sistemas de comunicación que garantizan el orden en las trasferencias entre dos puntos



Directorio de memoria principal

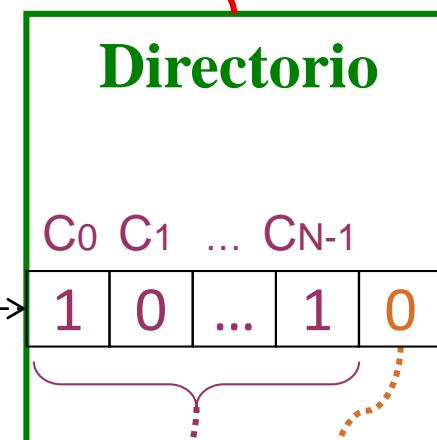
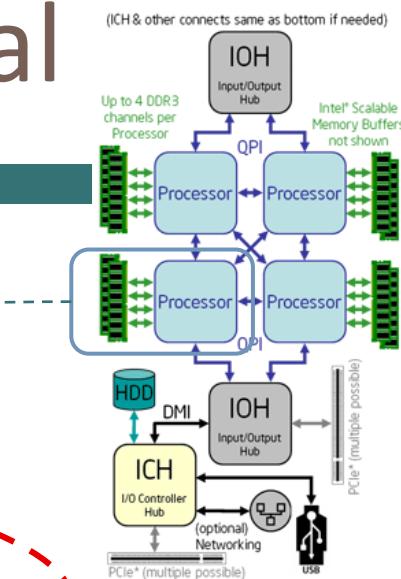


CC: Controlador caché
CM: Controlador de Memoria



Vector de bits (un bit por caché en el sistema)
con información sobre cachés con copia

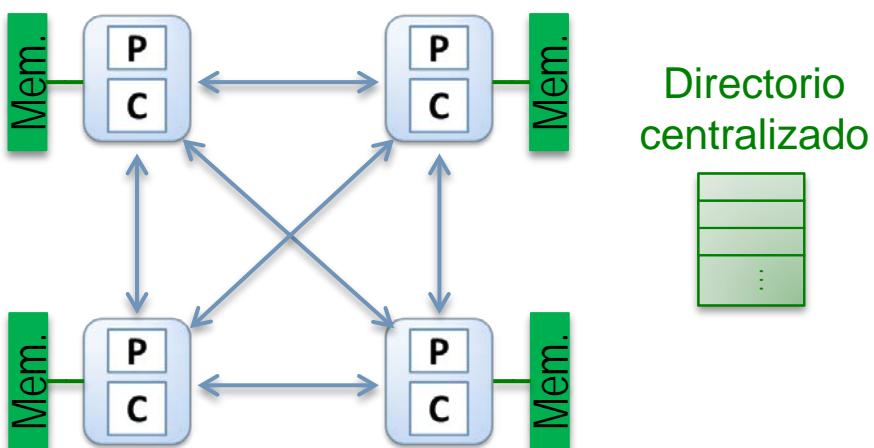
Estado bloque memoria
(0: válido, 1: inválido)



Alternativas para implementar el directorio

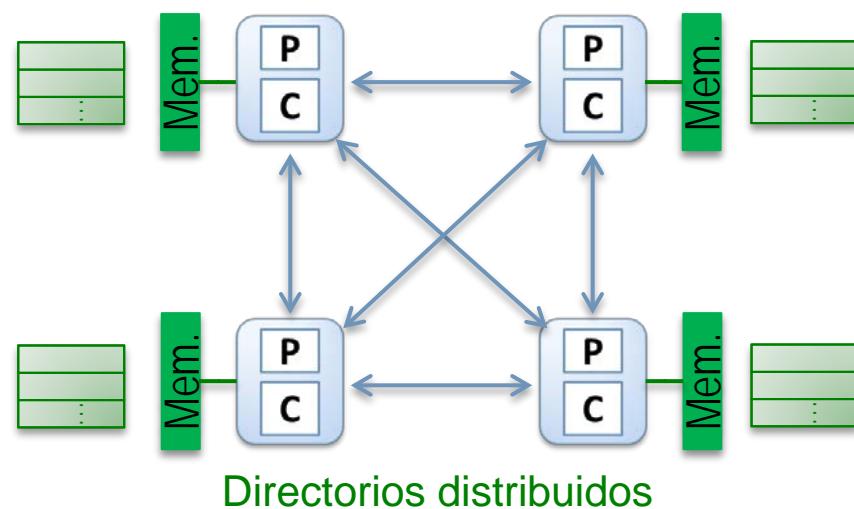
Centralizado

- Compartido por todos los nodos
- Contiene información de los bloques de todos los módulos de memoria

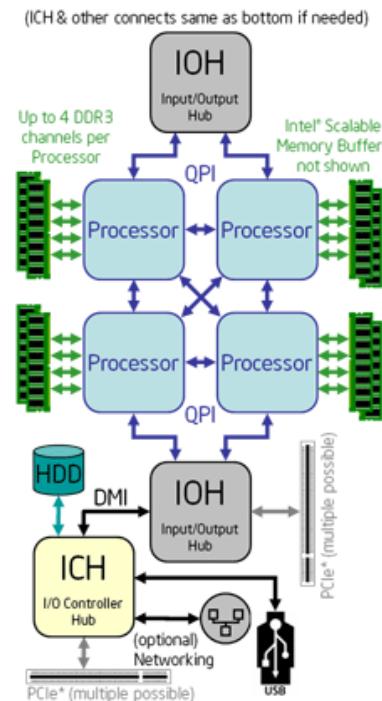
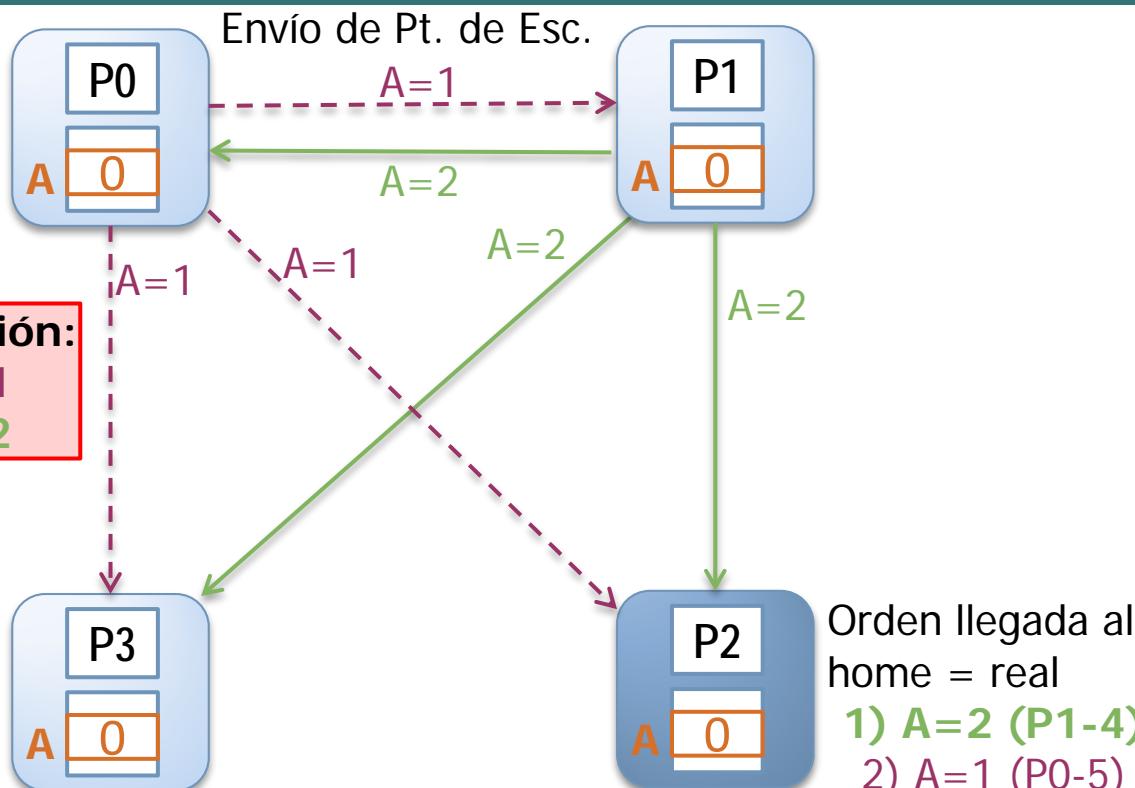


Distribuido

- Las filas se distribuyen entre los nodos
- Típicamente el directorio de un nodo contiene información de los bloques de sus módulos de memoria

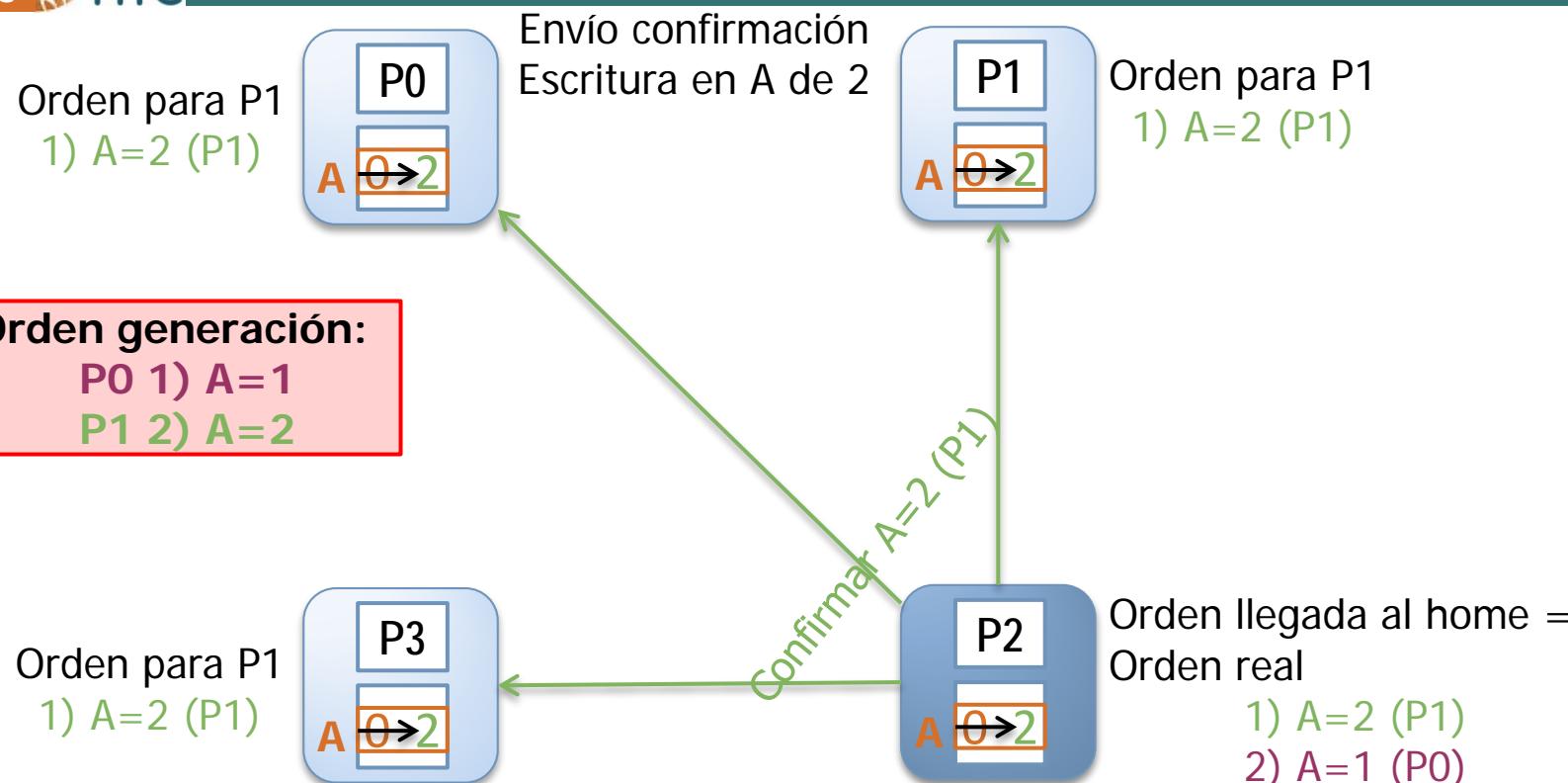


Serialización de las escrituras por el home. Usando difusión I



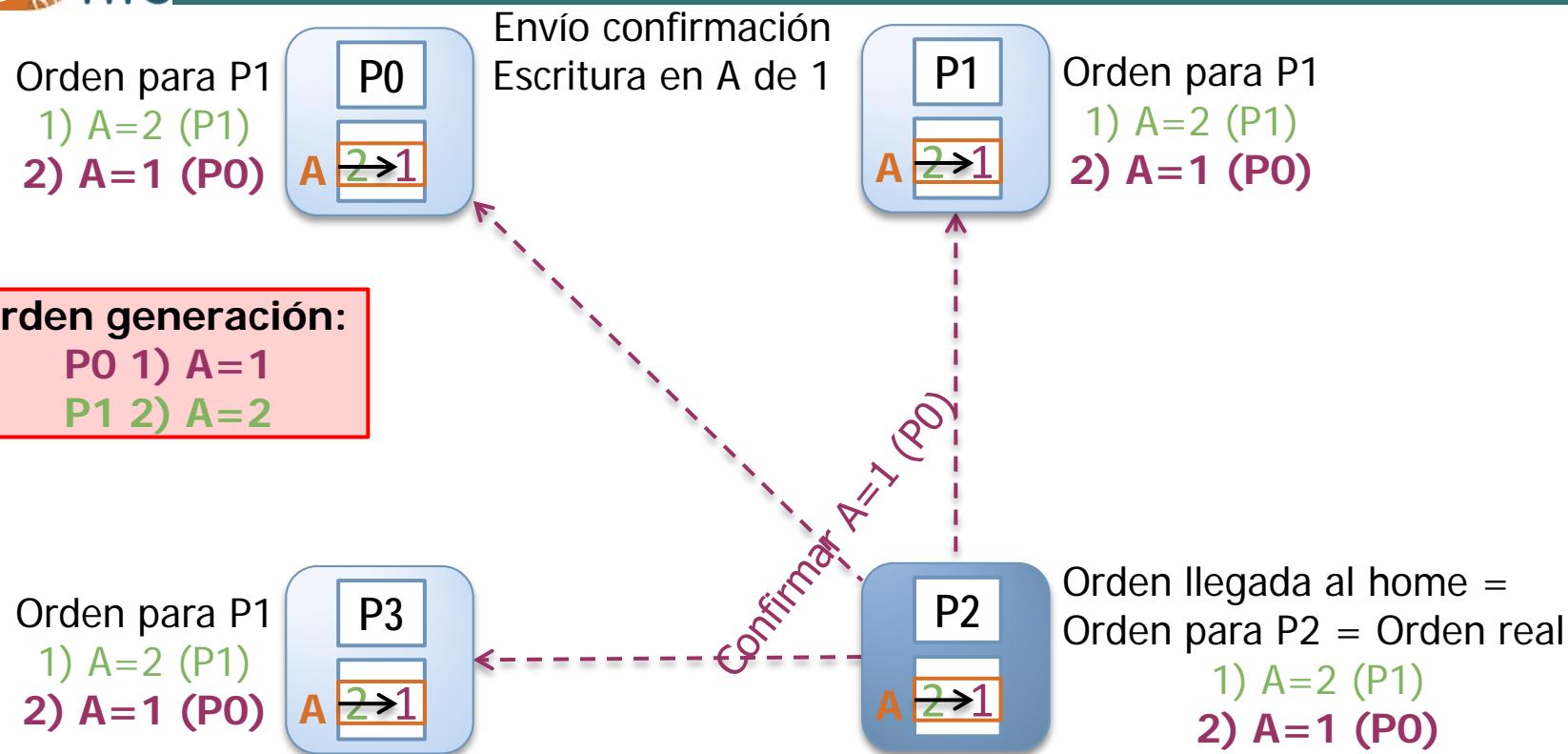
- Contenido inicial de las copias de la dirección A en **calabaza**. P0 escribe en A un 1 y, después, P1 escribe en A un 2.
- Se utiliza **actualización** para propagar las escrituras
- El orden de llegada al home es el orden real para todos

Serialización de las escrituras por el home. Usando difusión II



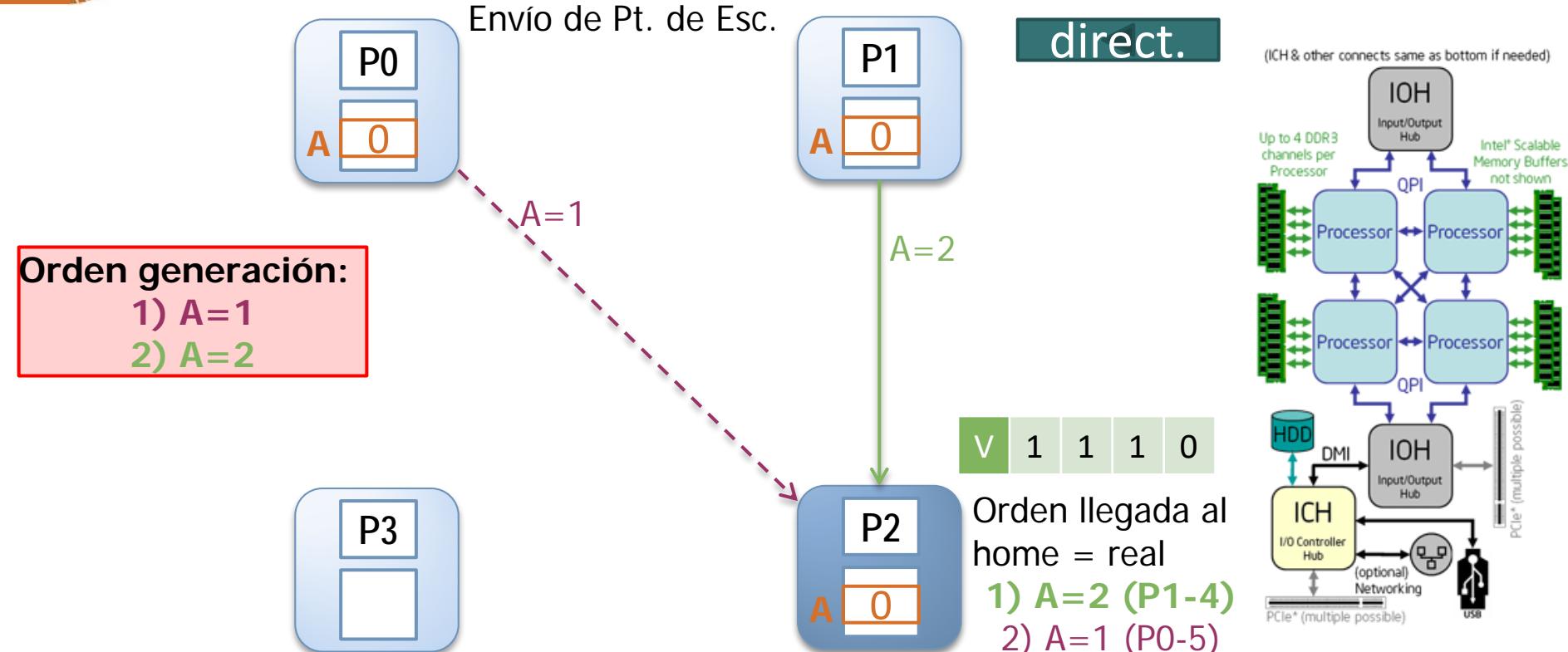
- Contenido inicial de las copias de la dirección A en **calabaza**
- Se utiliza actualización para propagar las escrituras (las propagación se nota con flechas)

Serialización de las escrituras por el home. Usando difusión III



- Se utiliza actualización para propagar las escrituras (las propagación se nota con flechas)

Serialización de las escrituras por el home. **Sin difusión y con directorio distribuido I**

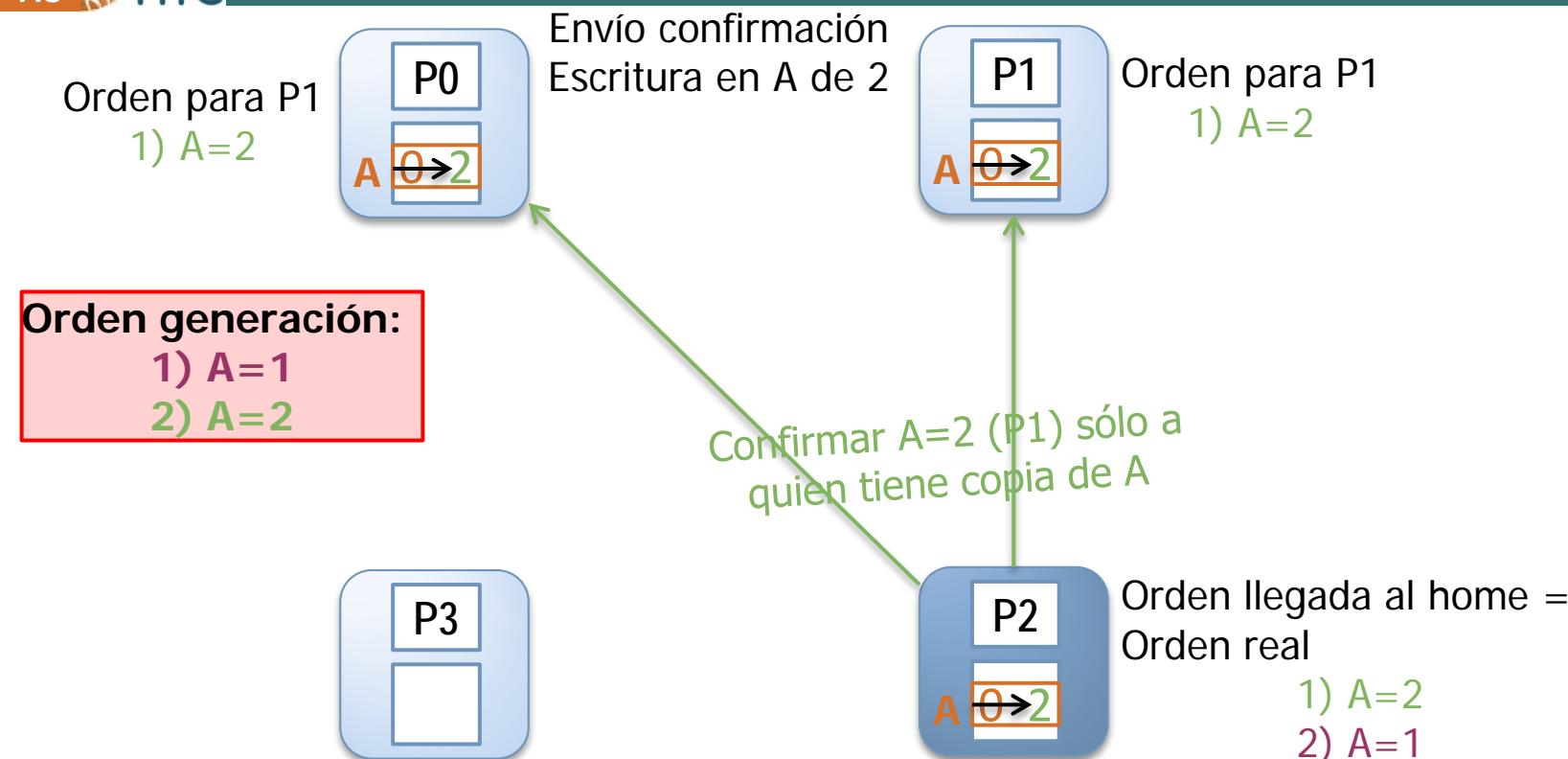


- Contenido inicial de las copias de la dirección A en **calabaza**. P0 escribe en A un 1 y, después, P1 escribe en A un 2.
 - Se utiliza actualización para propagar las escrituras (las propagación se nota con flechas)
 - El orden de llegada al home es el orden real para todos

Serialización de las escrituras por el home.

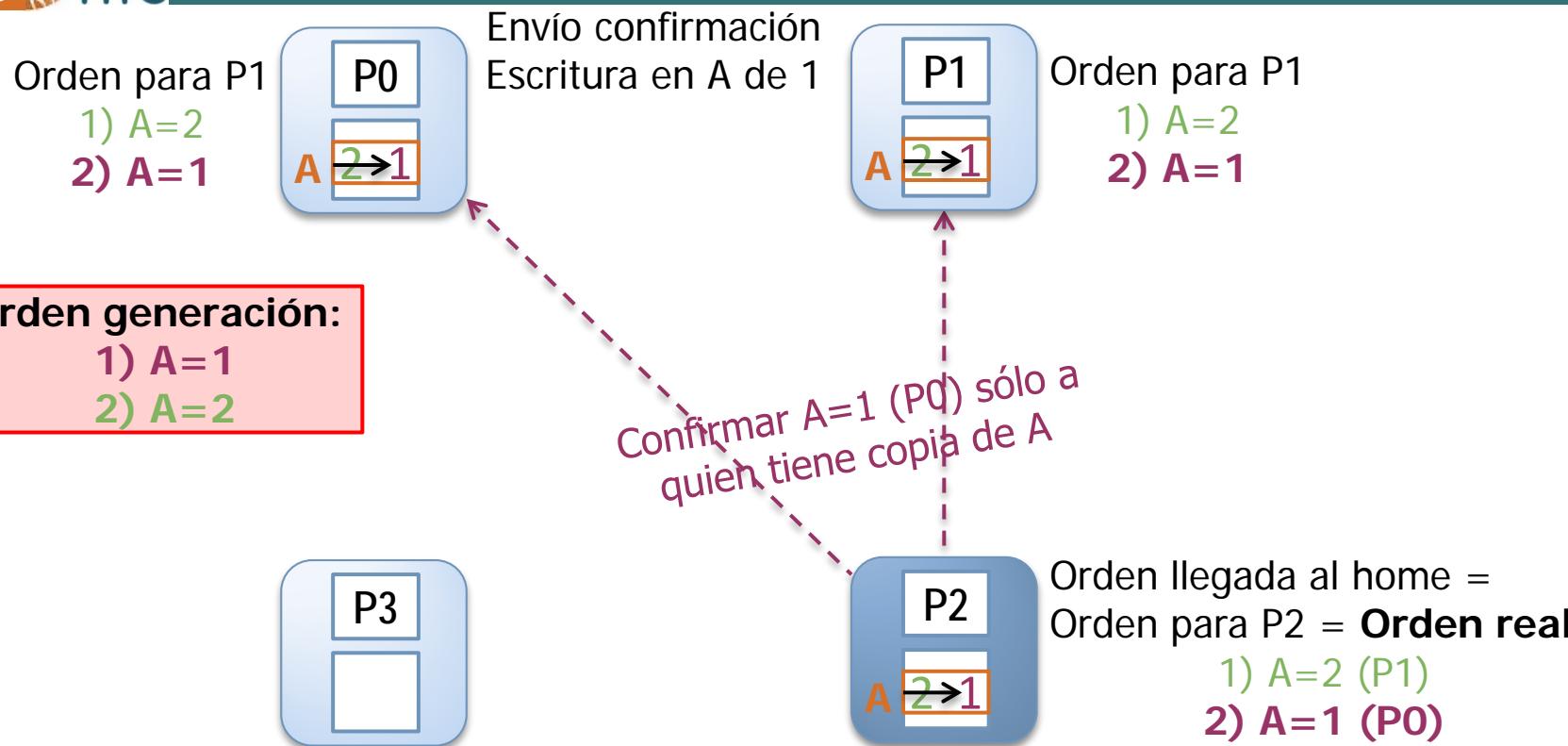
Sin difusión y con directorio distribuido II

AC ATC



- Contenido inicial de las copias de la dirección A en **calabaza**
- Se utiliza actualización para propagar las escrituras (las propagación se nota con flechas)

Serialización de las escrituras por el home. Sin difusión y con directorio distribuido III



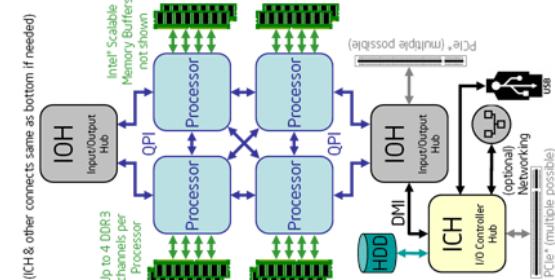
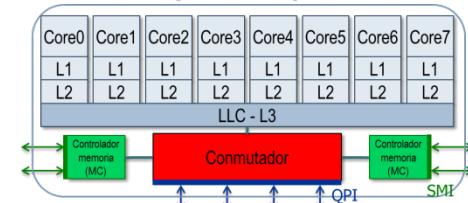
- Se utiliza actualización para propagar las escrituras (las propagación se nota con flechas)

Contenido Lección 8

- Sistema de memoria en multiprocesadores
- Concepto de coherencia en el sistema de memoria:
situaciones de incoherencia y requisitos para evitar
problemas en estos casos
- Protocolos de mantenimiento de coherencia:
clasiﬁcación y diseño
- Protocolo MSI de espionaje
- Protocolo MESI de espionaje
- Protocolo MSI basado en directorios con o sin difusión

Clasificación de protocolos para mantener coherencia en el sistema de memoria

- Protocolos de espionaje (snoopy)
 - Para buses, y en general sistemas con una difusión eficiente (bien porque el número de nodos es pequeño o porque la red implementa difusión).
- Protocolos basados en directorios.
 - Para redes sin difusión o escalables (multietapa y estáticas).
- Esquemas jerárquicos.
 - Para redes jerárquicas: jerarquía de buses, jerarquía de redes escalables, redes escalables-buses.



Facetas de diseño lógico en protocolos para coherencia

- Política de actualización de MP:
 - escritura inmediata, posescritura, mixta
- Política de coherencia entre cachés:
 - escritura con invalidación, escritura con actualización, mixta
- Describir comportamiento:
 - Definir posibles estados de los bloques en caché, y en memoria
 - Definir transferencias (indicando nodos que intervienen y orden entre ellas) a generar ante eventos:
 - lecturas/escrituras del procesador del nodo
 - como consecuencia de la llegada de paquetes de otros nodos.
 - Definir transiciones de estados para un bloque en caché, y en memoria

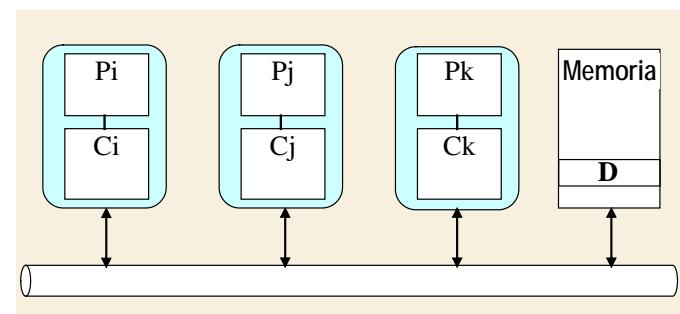
posescr



Ej. tras.

Contenido Lección 8

- Sistema de memoria en multiprocesadores
- Concepto de coherencia en el sistema de memoria:
situaciones de incoherencia y requisitos para evitar
problemas en estos casos
- Protocolos de mantenimiento de coherencia:
clasificación y diseño
- Protocolo MSI de espionaje
- Protocolo MESI de espionaje
- Protocolo MSI basado en directorios con o sin difusión



Protocolo de espionaje de tres estados (MSI) – posescritura e invalidación

- Estados de un bloque en caché:
 - **Modificado (M)**: es la única copia del bloque válida en todo el sistema
 - **Compartido (C,S)**: está válido, también válido en memoria y puede que haya copia válida en otras cachés
 - **Inválido (I)**: se ha invalidado o no está físicamente
- Estados de un bloque en memoria (en realidad se evita almacenar esta información):
 - **Válido**: puede haber copia válida en una o varias cachés
 - **Inválido**: habrá copia valida en una caché

Protocolo de espionaje de tres estados (MSI) – posescritura e invalidación

- Transferencias generadas por un nodo con caché (tipos de paquetes):

- Petición de lectura de un bloque (PtLec): por lectura con fallo de caché del procesador del nodo (PrLec)
- Petición de acceso exclusivo (PtLecEx): por escritura del procesador (PrEsc) en bloque **compartido** o **inválido** (alternativa: **compartido** PtEx, **inválido** PtLecEx)

- Petición de posescritura (PtPEsc): por el reemplazo del bloque **modificado** (el procesador del nodo no espera respuesta)

- Respuesta con bloque (RpBloque): al tener en estado **modificado** el bloque solicitado por una PtLec o PtLecEx recibida

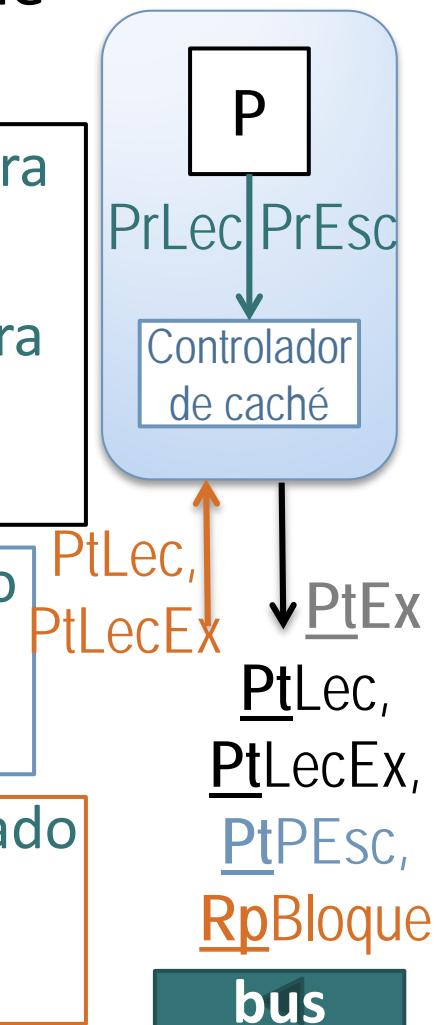


Diagrama MSI de transiciones de estados

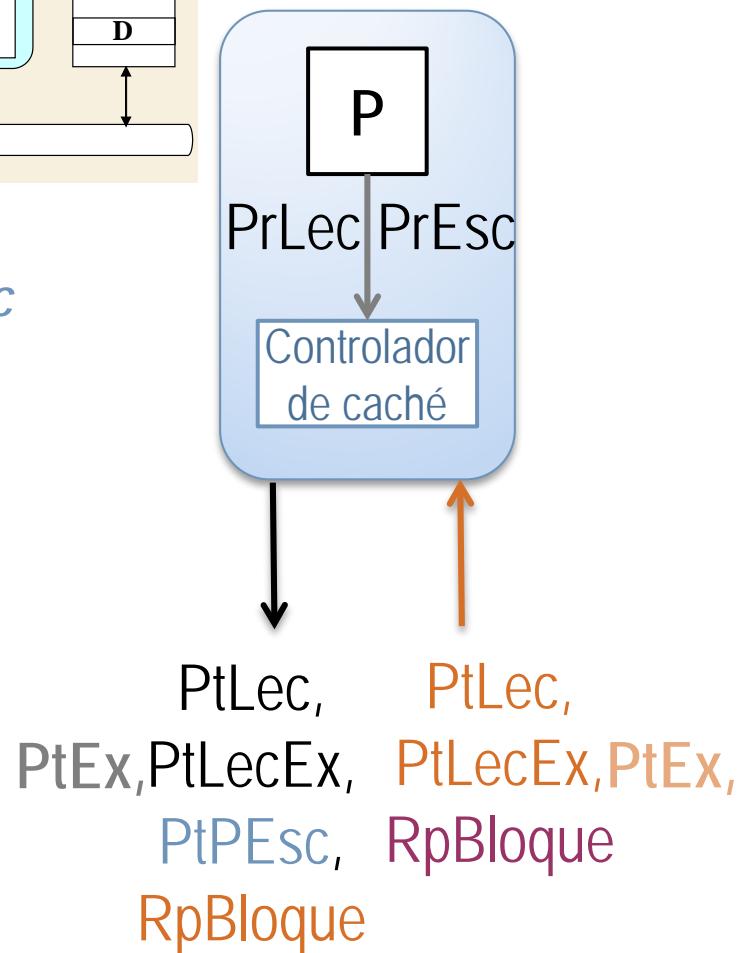
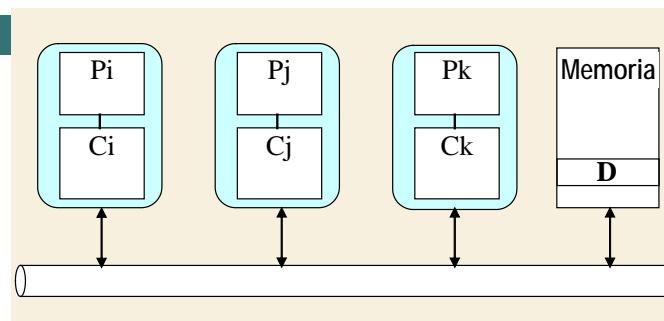
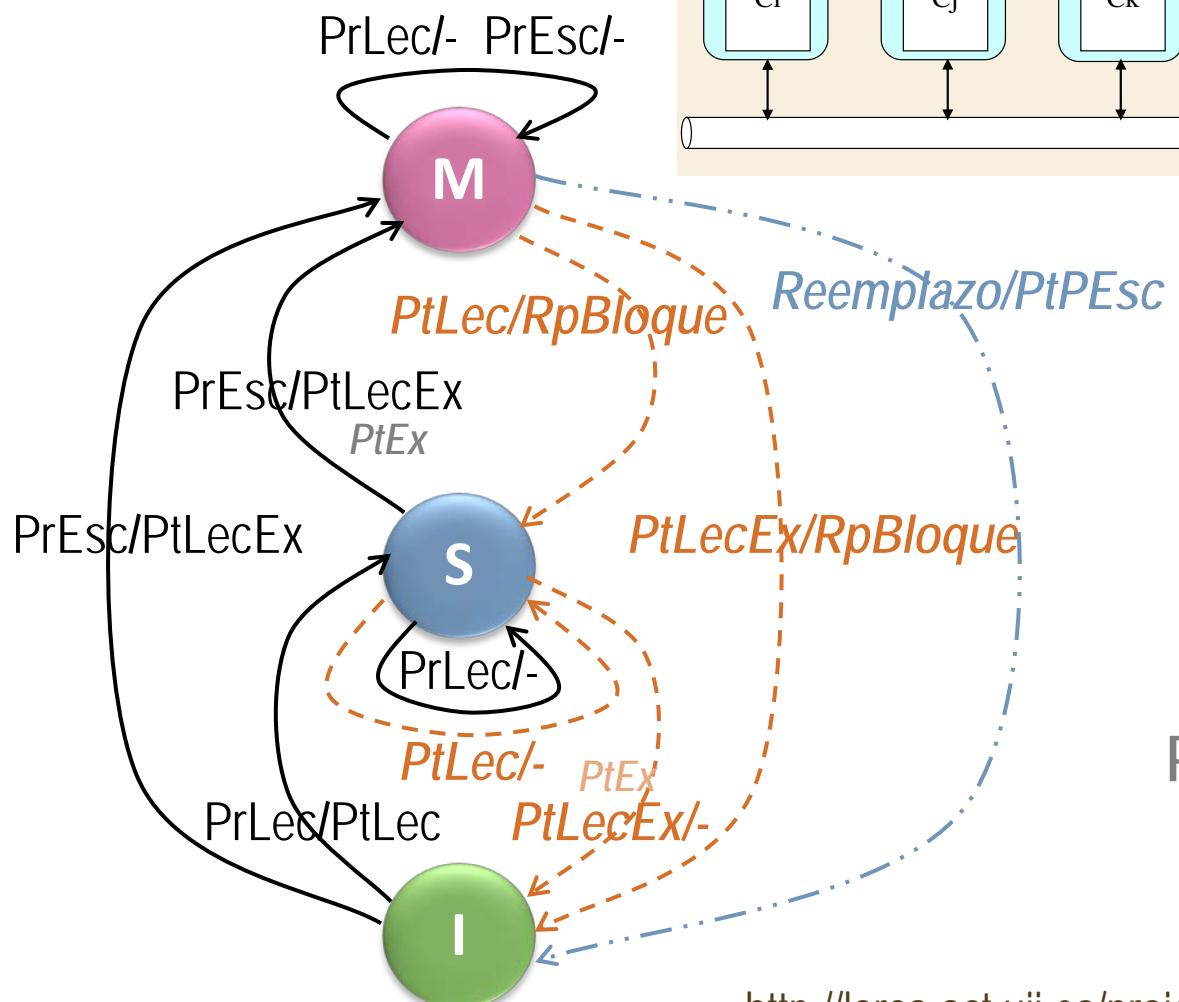
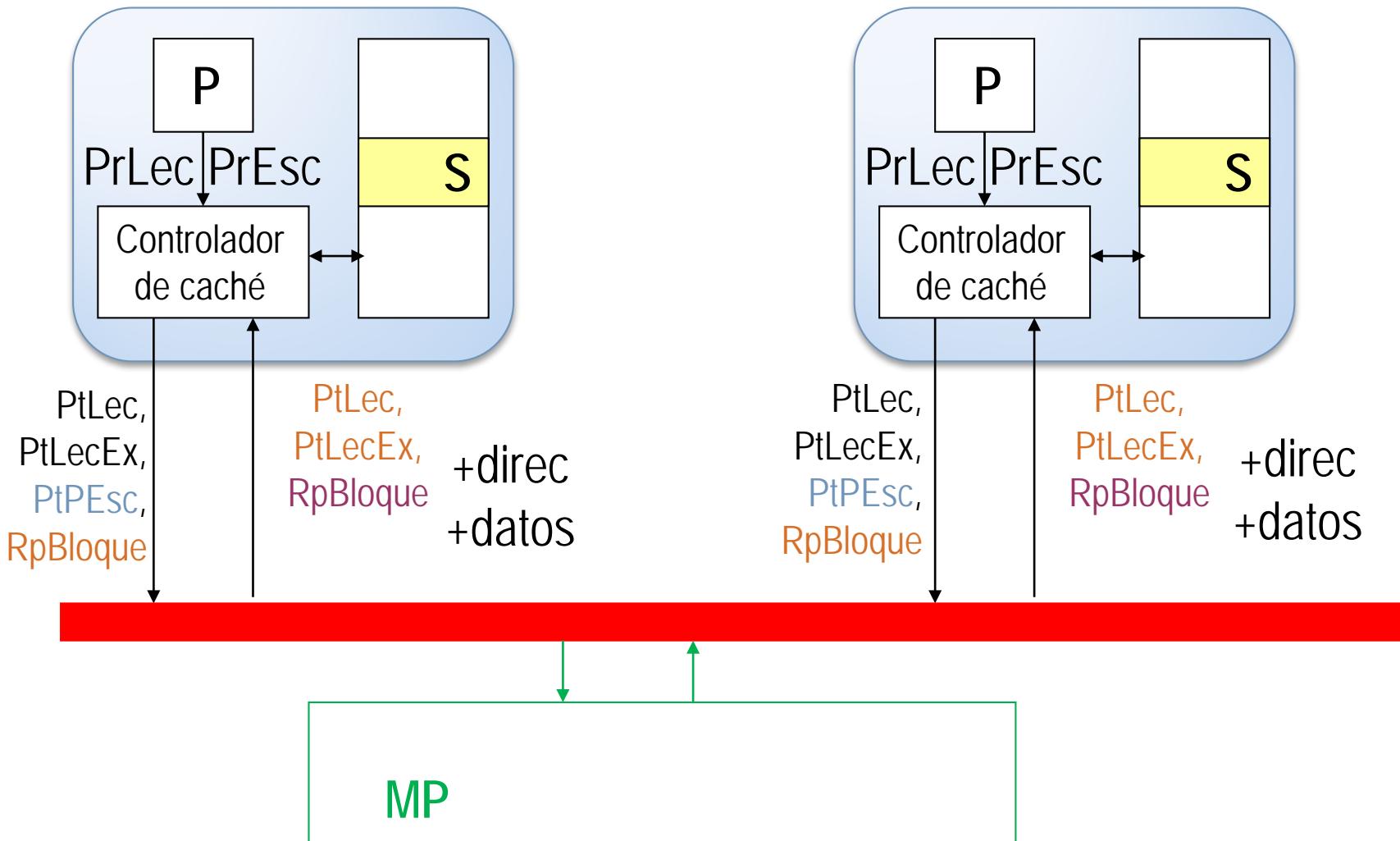


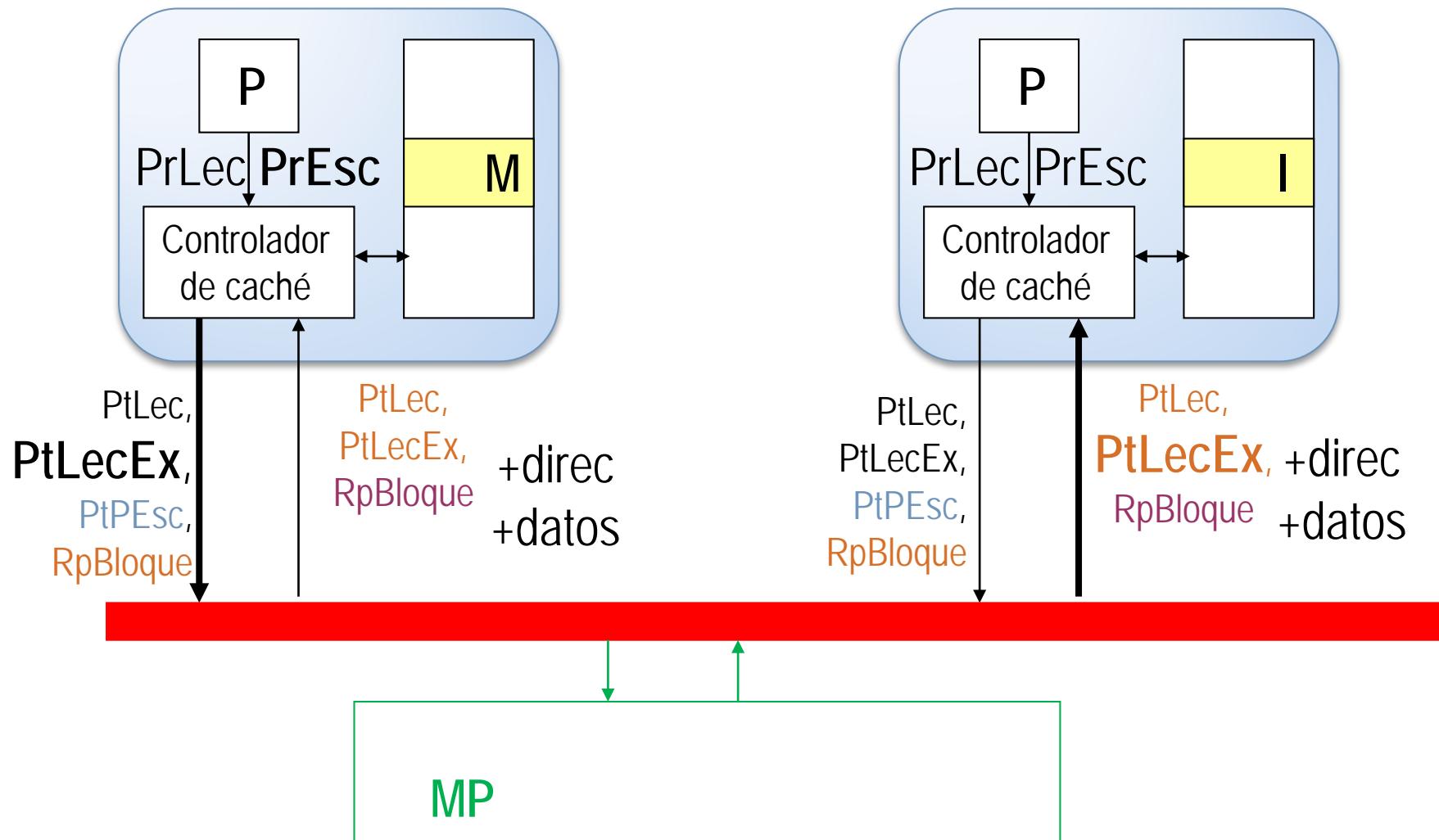
Tabla de descripción de MSI

EST. ACT.	EVENTO	ACCIÓN	SIGUIENTE
Modificado (M)	PrLec/PrEsc		Modificado
	PtLec	Genera paquete respuesta (RpBloque)	Compartido
	PtLecEx	Genera paquete respuesta (RpBloque) Invalida copia local	Inválido
	Reemplazo	Genera paquete posescritura (PtPEsc)	Inválido
Compart. (S)	PrLec		Compartido
	PrEsc <small>posescr</small>	Genera paquete PtLecEx (PtEx)	Modificado
	PtLec		Compartido
	PtLecEx <small>invalido</small>	Invalida copia local	Inválido
Inválido (I)	PrLec	Genera paquete PtLec	Compartido
	PrEsc	Genera paquete PtLecEx	Modificado
	PtLec/PtLecEx		Inválido

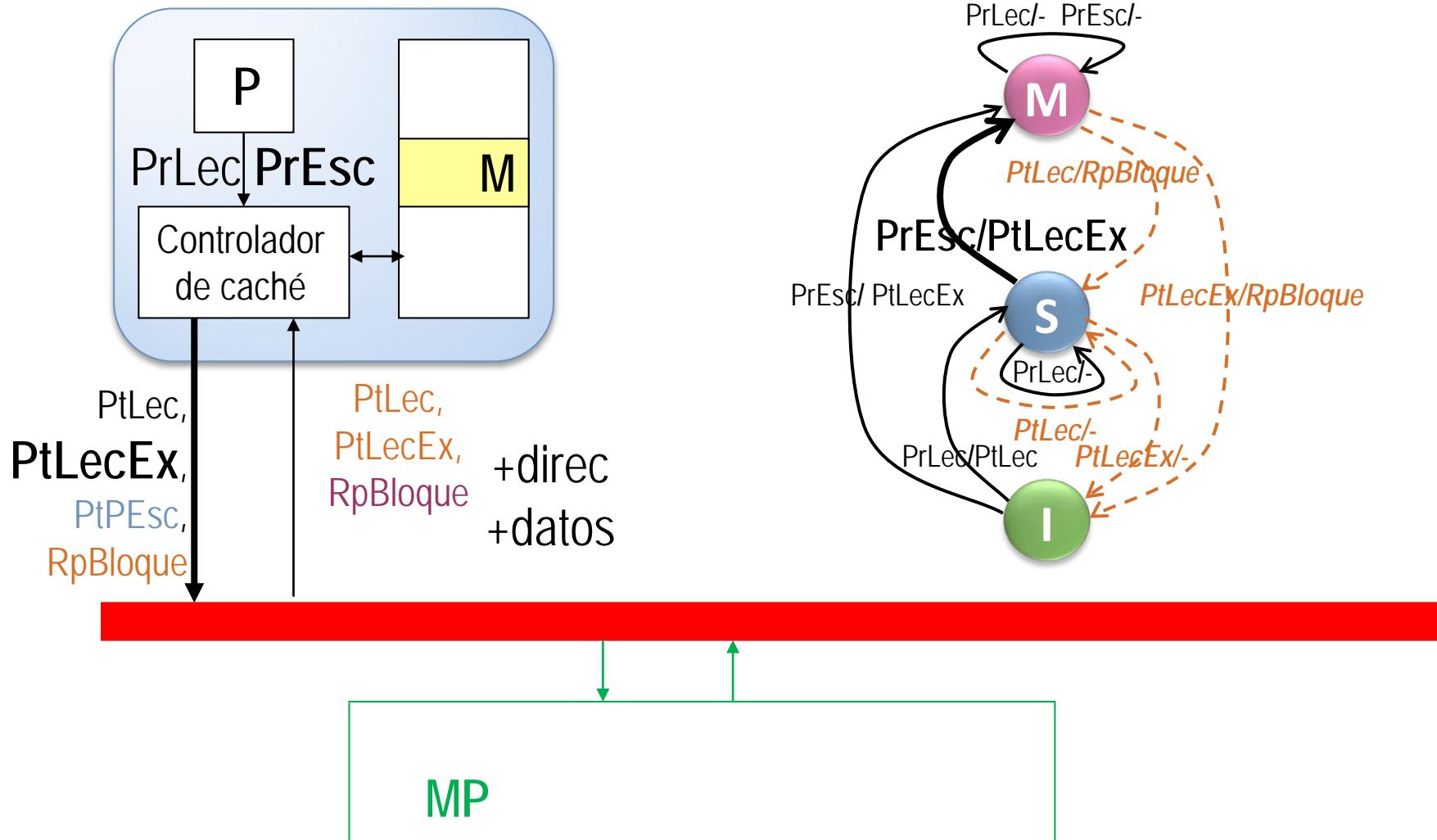
Ejemplo MSI I



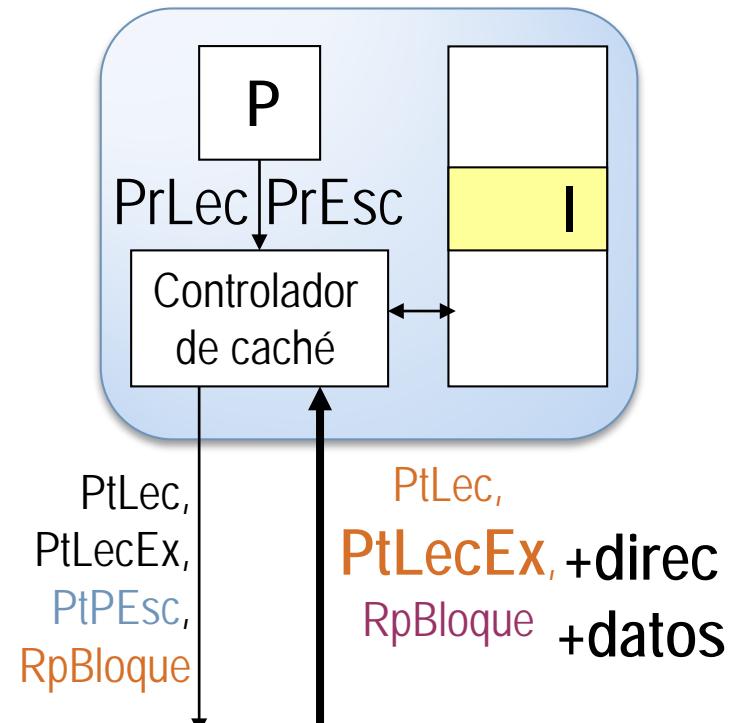
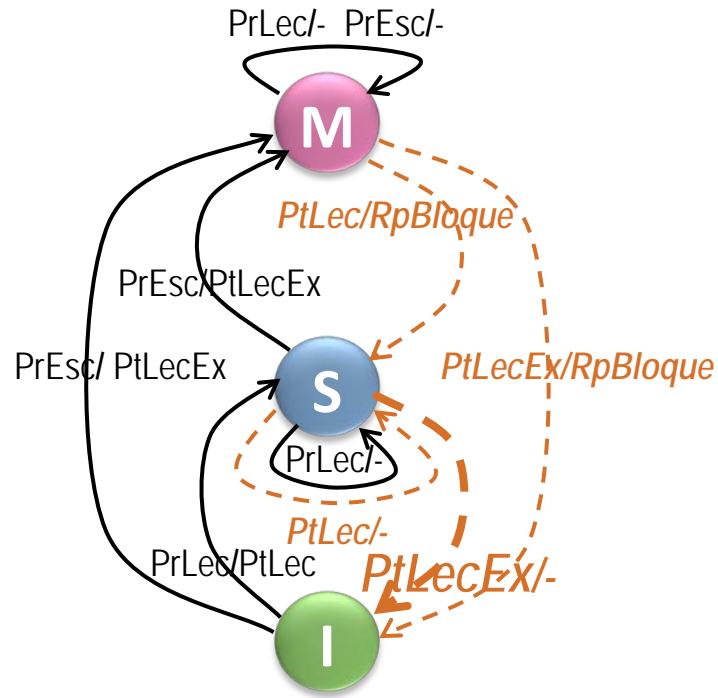
Ejemplo MSI II



Ejemplo MSI III

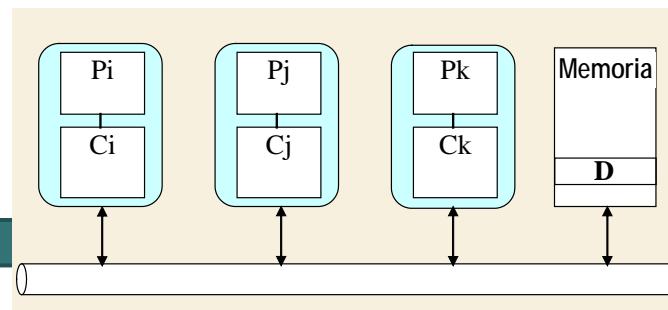


Ejemplo MSI IV



Contenido Lección 8

- Sistema de memoria en multiprocesadores
- Concepto de coherencia en el sistema de memoria:
situaciones de incoherencia y requisitos para evitar
problemas en estos casos
- Protocolos de mantenimiento de coherencia:
clasificación y diseño
- Protocolo MSI de espionaje
- Protocolo MESI de espionaje **MSI**
- Protocolo MSI basado en directorios con o sin difusión



Protocolo de espionaje de cuatro estados (MESI) – posescritura e invalidación

- Estados de un bloque en caché:
 - **Modificado (M)**: es la única copia del bloque válida en todo el sistema
 - **Exclusivo (E)**: es la *única* copia del bloque válida en cachés, la memoria también está actualizada
 - **Compartido (C,Shared)**: es válido, también válido en memoria y en *al menos* otra caché
 - **Inválido (I)**: se ha invalidado o no está físicamente
- Estados de un bloque en memoria(en realidad se evita almacenar esta información):
 - **Válido**: puede haber copia válida en una o varias cachés
 - **Inválido**: habrá copia valida en una caché

Diagrama MESI de transiciones de estados

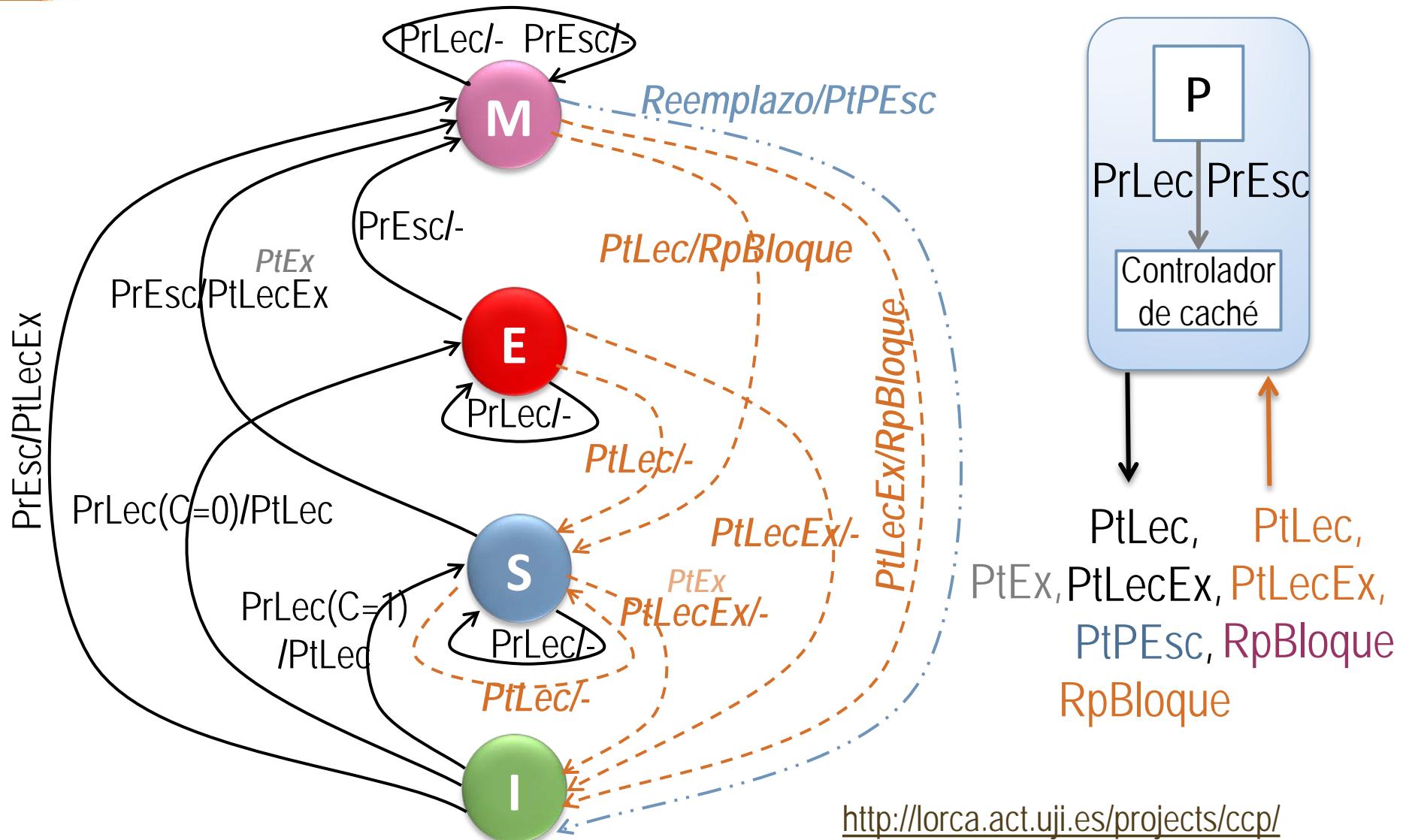


Tabla de descripción de MESI

MSI

Modificado (M)	PrLec/PrEsc		Modificado
	PtLec	Genera RpBloque	Compartido
	PtLecEx	Genera RpBloque. Invalida copia local	Inválido
	Reemplazo	Genera PtPEsc	Inválido
Exclusivo (E)	PrLec		Exclusivo
	PrEsc		Modificado
	PtLec		Compartido
	PtLecEx	Invalida copia local	Inválido
Compartido (S)	PrLec/PtLec		Compartido
	PrEsc	Genera PtLecEx (PtEx)	Modificado
	PtLecEx	Invalida copia local	Inválido
Inválido (I)	PrLec (C=1)	Genera PtLec	Compartido
	PrLec (C=0)	Genera PtLec	Exclusivo
	PrEsc	Genera PtLecEx	Modificado
	PtLec/PtLecEx		Inválido

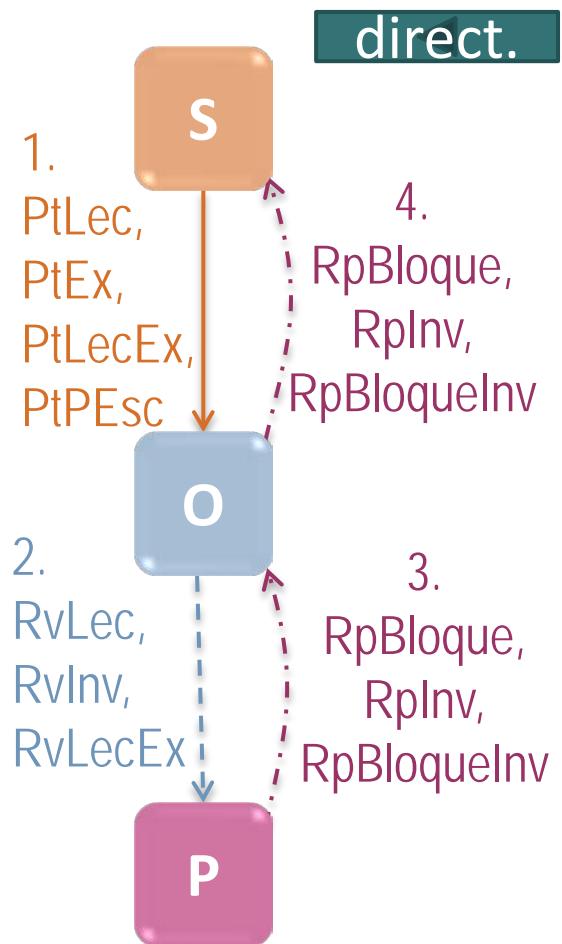
Contenido Lección 8

- Sistema de memoria en multiprocesadores
- Concepto de coherencia en el sistema de memoria:
situaciones de incoherencia y requisitos para evitar
problemas en estos casos
- Protocolos de mantenimiento de coherencia:
clasificación y diseño
- Protocolo MSI de espionaje
- Protocolo MESI de espionaje
- Protocolo MSI basado en directorios con o sin difusión

MSI

MSI con directorios (sin difusión) I

- Estados de un bloque en caché:
 - Modificado (M), Compartido (S), Inválido (I)
- Estados de un bloque en MP:
 - Válido e inválido
- Transferencias (tipos de paquetes) :
 - Tipos de nodos: solicitante (S), origen (O), modificado (M), propietario (P) y compartidor (C)
 - Petición de
 - nodo **S** a **O**: lectura de un bloque (**PtLec**), lectura con acceso exclusivo (**PtLecEx**), petición de acceso exclusivo sin lectura (**PtPEsc**)
 - Reenvío de petición de
 - nodo **O** a nodos con copia (**P**, **M**, **C**): invalidación (**RvInv**), lectura (**RvLec**, **RvLecEx**).
 - Respuesta de
 - nodo **P** a **O**: respuesta con bloque (**RpBloque**), resp. con o sin bloque confirmando inv. (**RpInv**, **RpBloqueInv**)
 - nodo **O** a **S**: resp. con bloque (**RpBloque**), resp. con o sin bloque confirmando fin inv. (**RpInv**, **RpBloqueInv**)

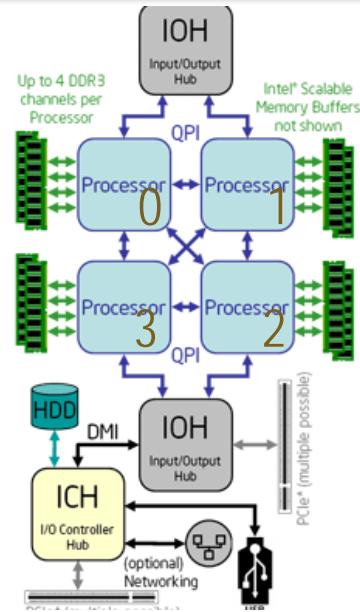
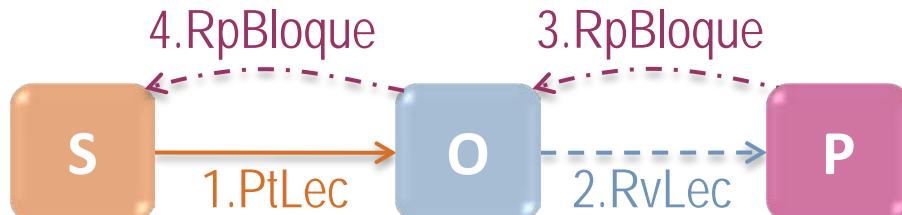


<https://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html>

MSI con directorios (sin difusión) II

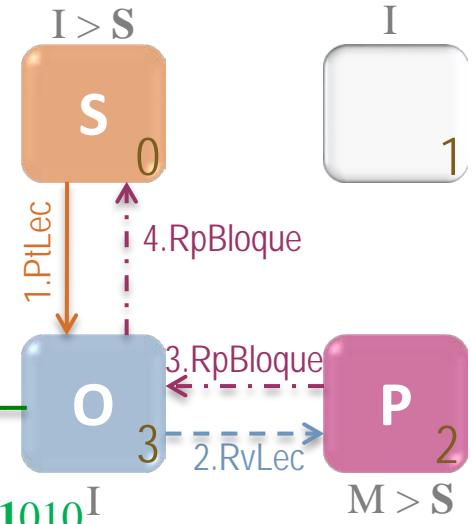
Estado inicial	C0 C1 C2 C3	Evento	Estado final
D) Inválido	I 0 0 1 0	Fallo de lectura	D) Válido
S) Inválido			S) Compartido
P) Modificado			P) Compartido
Acceso remoto			

direct.



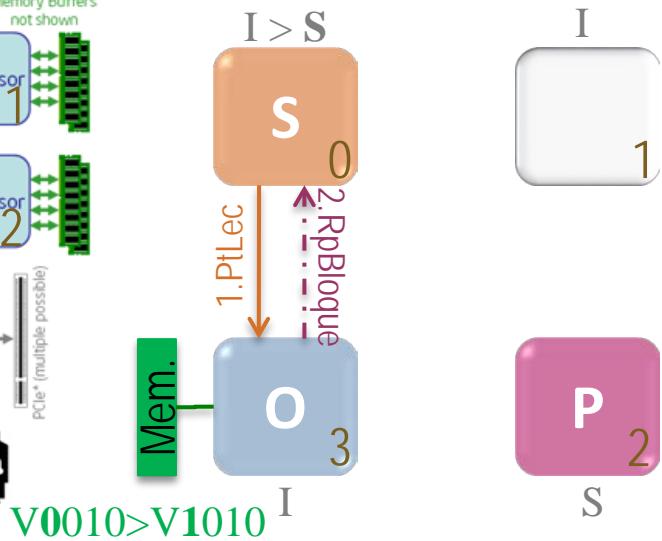
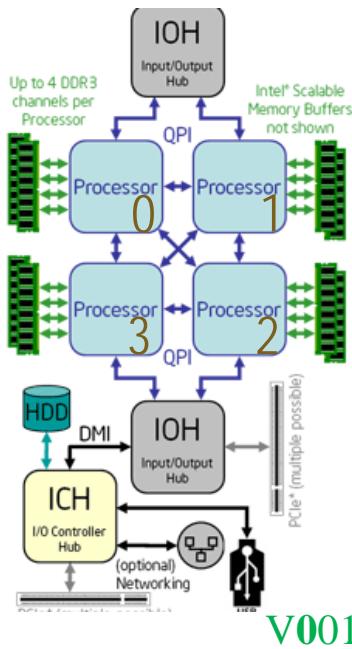
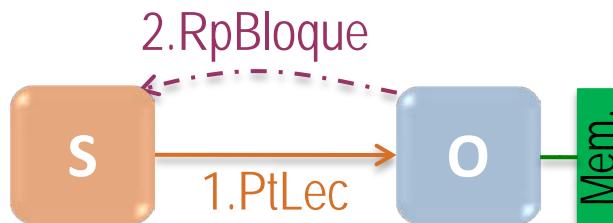
I0010>V1010

Ejemplo con 4 nodos:



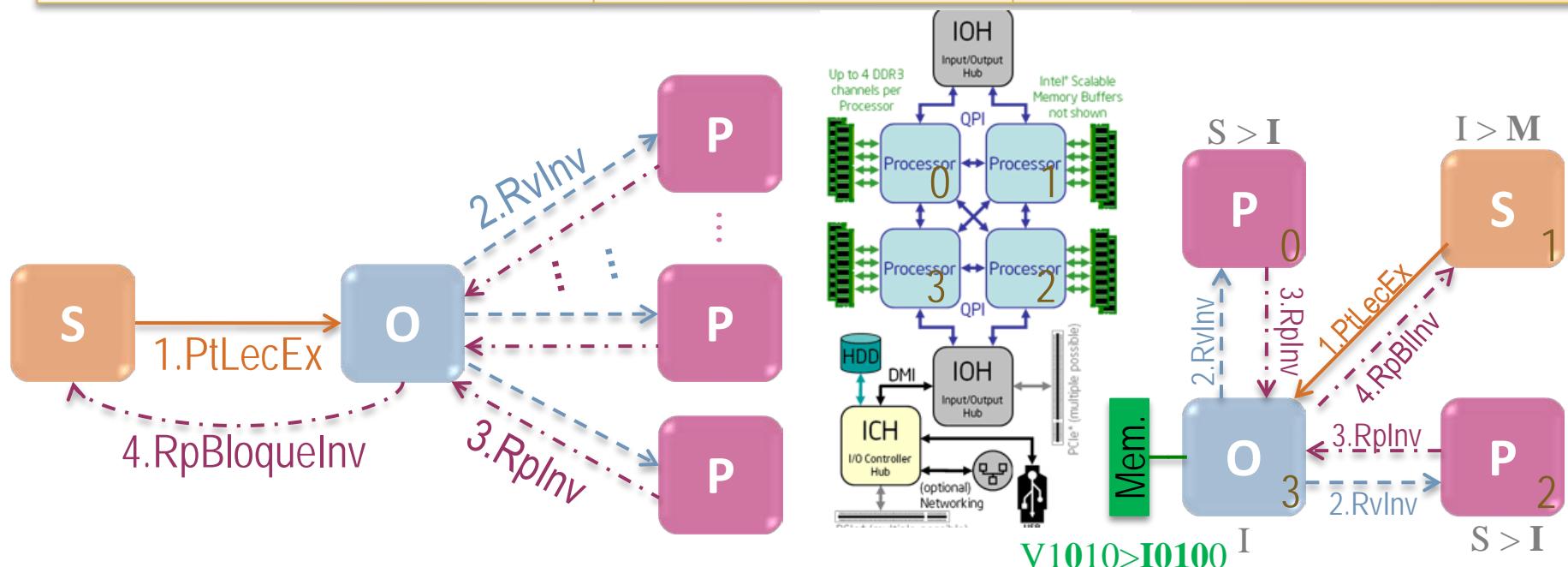
MSI con directorios (sin difusión) III

Estado inicial	Evento	Estado final										
D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="background-color: #6B8E23; color: white;">v</td><td style="background-color: #D9EAD3;">0</td><td style="background-color: #D9EAD3;">0</td><td style="background-color: #D9EAD3;">1</td><td style="background-color: #D9EAD3;">0</td></tr> </table> S) Inválido P) Compartido Acceso remoto	v	0	0	1	0	Fallo de lectura	D) Válido <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="background-color: #6B8E23; color: white;">v</td><td style="background-color: #D9EAD3;">1</td><td style="background-color: #D9EAD3;">0</td><td style="background-color: #D9EAD3;">1</td><td style="background-color: #D9EAD3;">0</td></tr> </table> S) Compartido P) Compartido	v	1	0	1	0
v	0	0	1	0								
v	1	0	1	0								



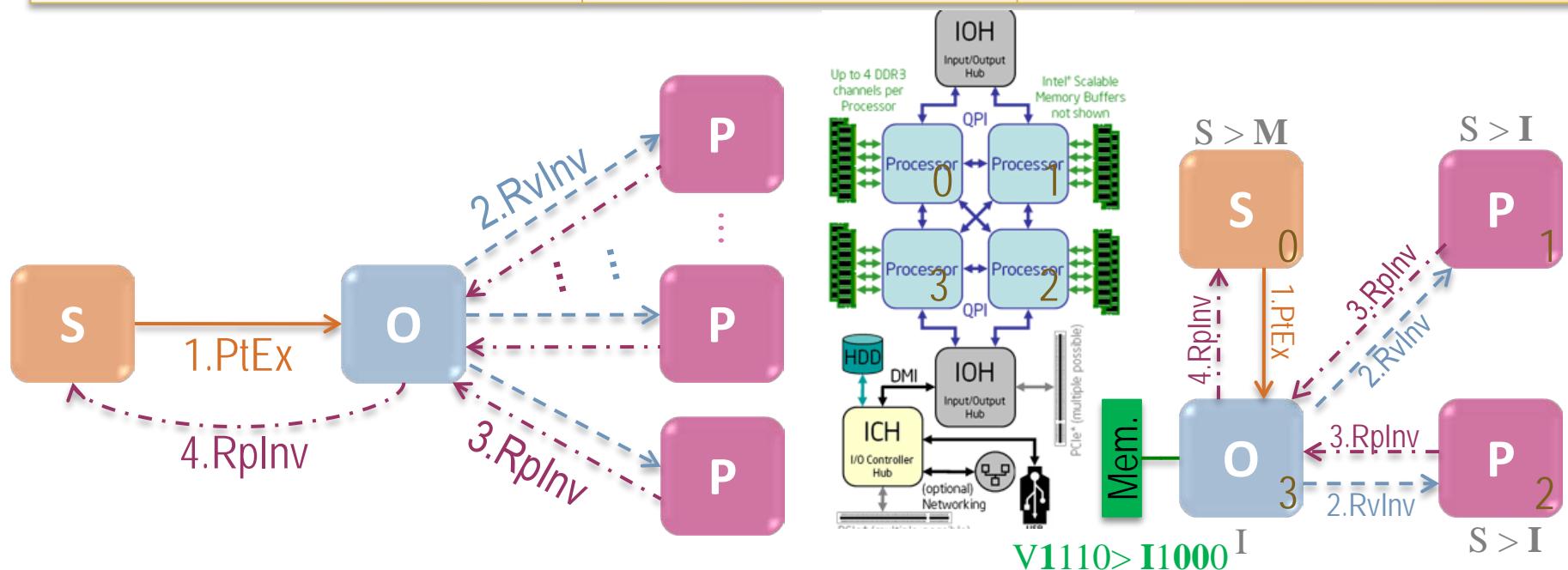
MSI con directorios (sin difusión) IV

Estado inicial	C0 C1 C2 C3	Evento	Estado final
D) Válido		Fallo de escritura	D) Inválido
S) Inválido			S) Modificado
P) Compartido			P) Inválido
Acceso remoto			



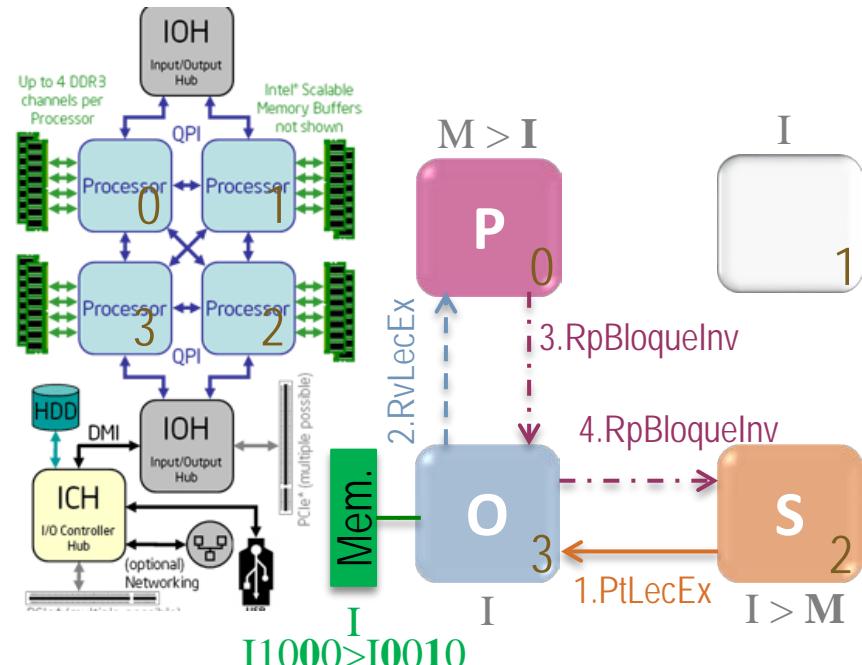
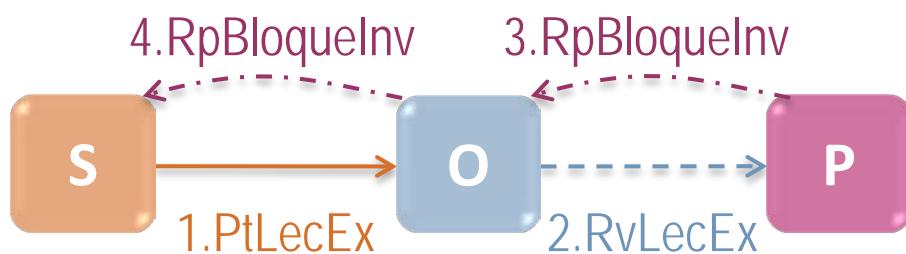
MSI con directorios (sin difusión) V

Estado inicial	Evento	Estado final
D) Válido S) Compartido P) Compartido Acceso remoto	Fallo de escritura	D) Inválido S) Modificado P) Inválido



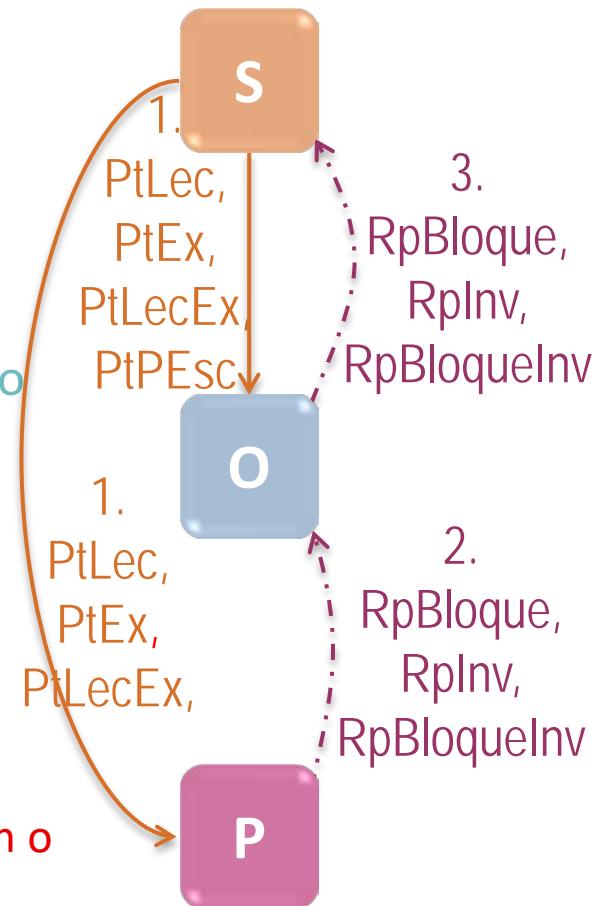
MSI con directorios (sin difusión) VI

Estado inicial	C0 C1 C2 C3	Evento	Estado final
D) Inválido	I 1 0 0 0	Fallo de escritura	D) Inválido
S) Inválido			S) Modificado
P) Modificado			P) Inválido
Acceso remoto			



MSI con directorios (con difusión) I

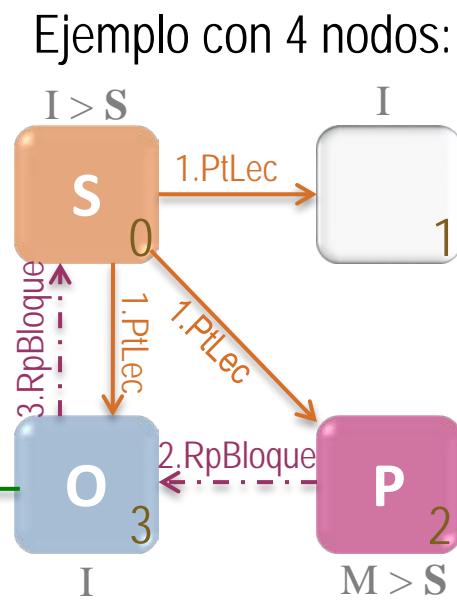
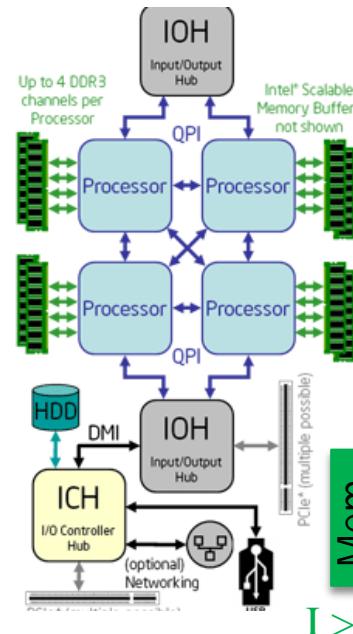
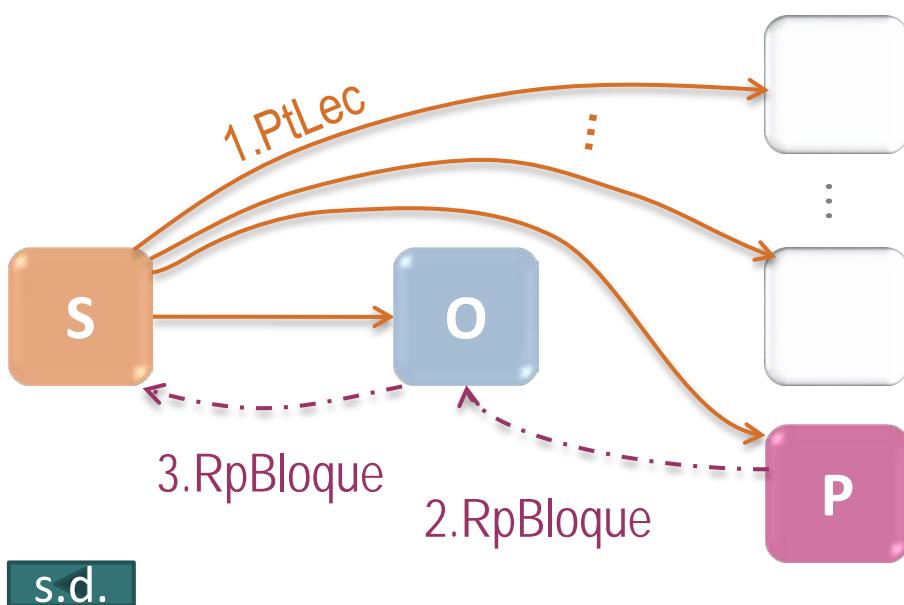
- Estados de un bloque en caché:
 - Modificado (M), Compartido (C), Inválido (I)
- Estados de un bloque en MP:
 - Válido e inválido
- Transferencias (tipos de paquetes) :
 - Tipos de nodos: solicitante (S), origen (O), modificado (M), propietario (P) y compartidor (C)
 - Difusión de petición del nodo S a
 - O y P: lectura de un bloque (PtLec), lectura con acceso exclusivo (PtLecEx), petición de acceso exclusivo sin lectura (PtEx)
 - O: posescritura (PtPEsc)
 - Respuesta de
 - nodo P a O: respuesta con bloque (RpBloque), resp. con o sin bloque confirmando inv. (Rplnv, RpBloquelnv)
 - nodo O a S: resp. con bloque (RpBloque), resp. con o sin bloque confirmando fin inv. (Rplnv, RpBloquelnv)



s.d.

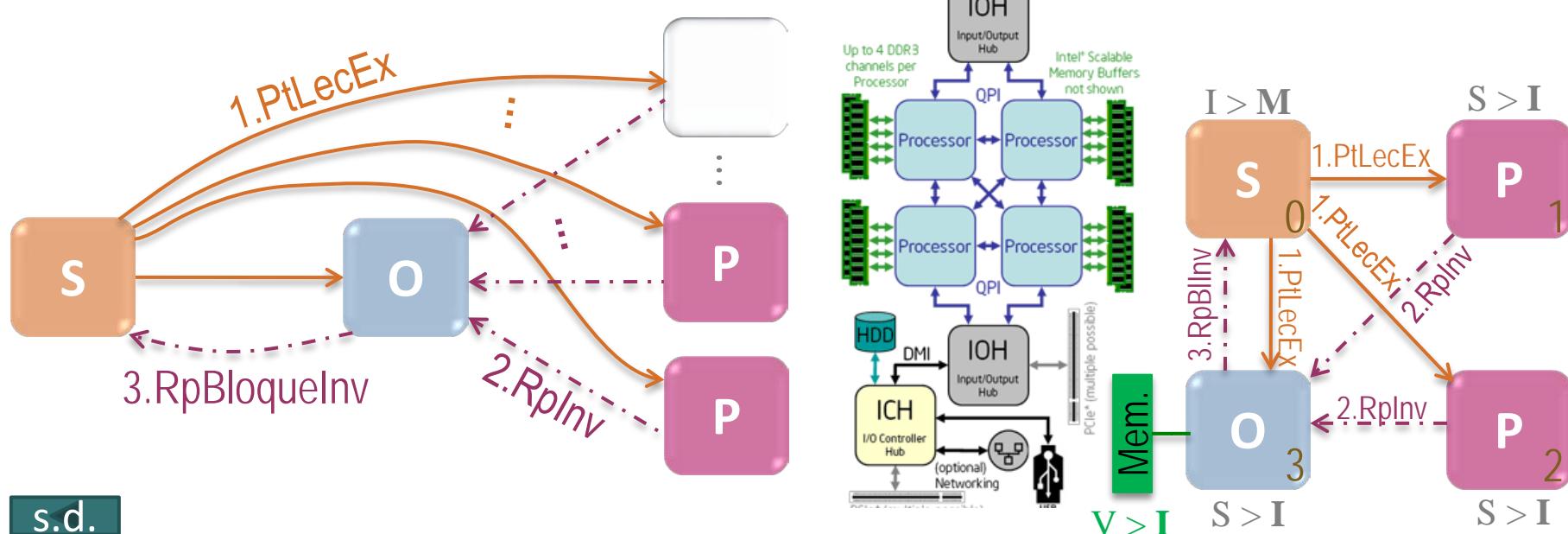
MSI con directorios (con difusión) II

Estado inicial	Evento	Estado final
D) Inválido I S) Inválido P) Modificado Acceso remoto	Fallo de lectura	D) Válido V S) Compartido P) Compartido



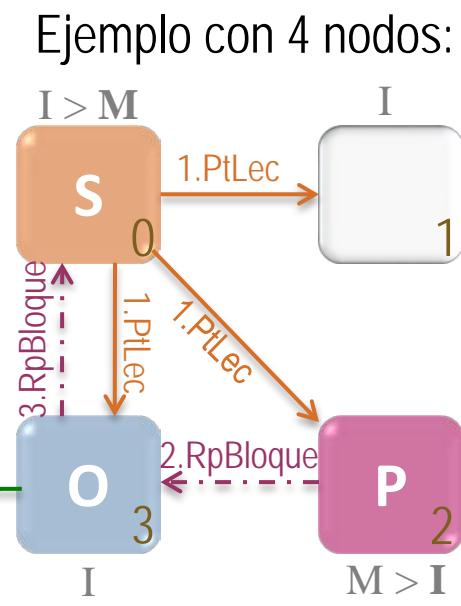
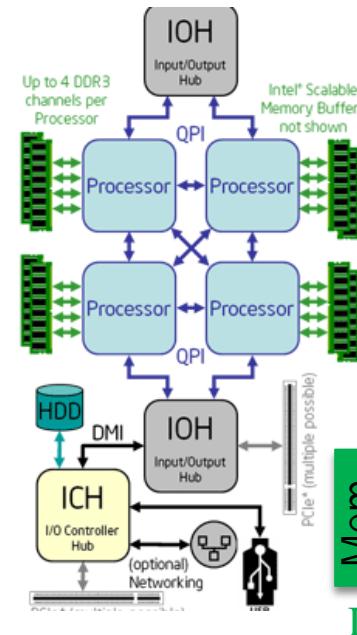
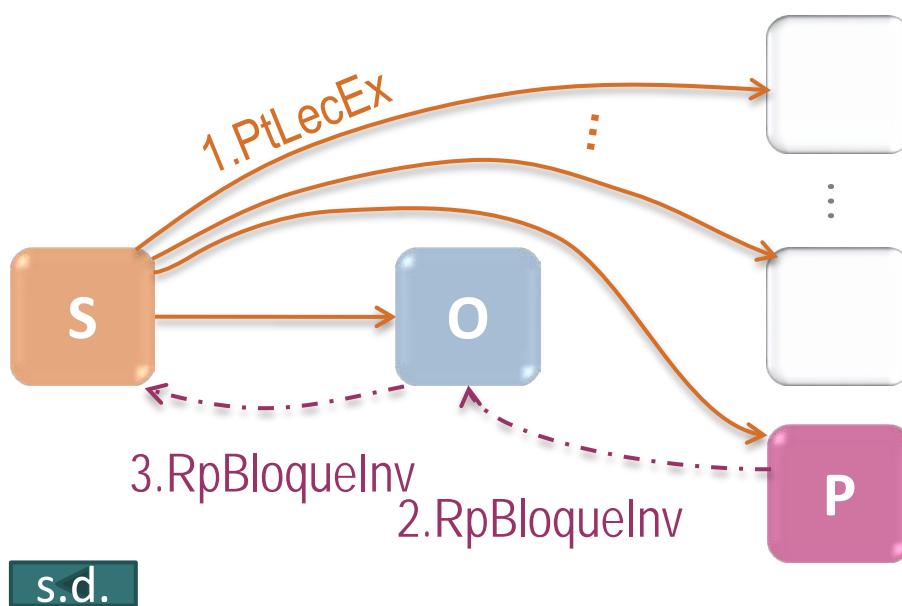
MSI con directorios (con difusión) III

Estado inicial	Evento	Estado final
D) Válido v S) Inválido P) Compartido Acceso remoto	Fallo de escritura (Procesador escribe)	D) Inválido i S) Modificado P) Inválido



MSI con directorios (con difusión) II

Estado inicial	Evento	Estado final
D) Inválido I S) Inválido P) Modificado Acceso remoto	Fallo de escritura	D) Válido I S) Modificado P) Inválido



Para ampliar ...

➤ Webs

- An Introduction to the Intel® QuickPath Interconnect,
<http://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html>
- Animaciones de protocolos de coherencia de cachés
<http://lorca.act.uji.es/projects/ccp/>

2º curso / 2º cuatr.
Grado en
Ing. Informática

Arquitectura de Computadores

Tema 3

Lección 9. Consistencia del sistema de memoria

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad
de Granada

Lecciones

- Lección 7. Arquitecturas TLP
- Lección 8. Coherencia del sistema de memoria
- Lección 9. Consistencia del sistema de memoria
 - Concepto de consistencia de memoria
 - Consistencia secuencial
 - Modelos de consistencia relajados
- Lección 10. Sincronización

Objetivos Lección 9

- Explicar el concepto de consistencia.
- Distinguir entre coherencia y consistencia.
- Distinguir entre el modelo de consistencia secuencial y los modelos relajados.
- Distinguir entre los diferentes modelos de consistencia relajados.

Bibliografía Lección 9

➤ Fundamental

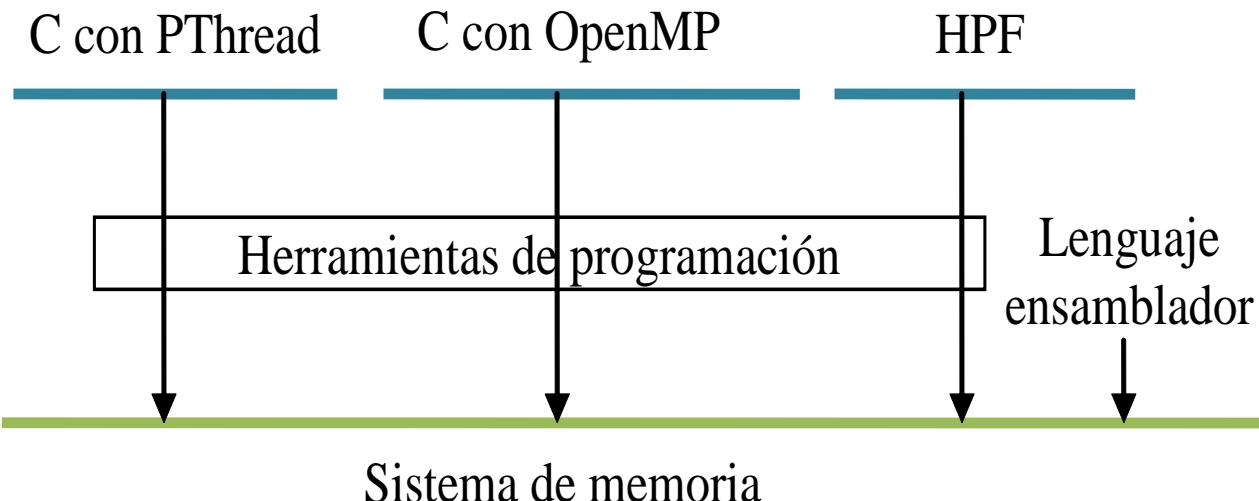
- Cap.3, Secc. 4. M. Anguita, J. Ortega. *Fundamentos y Problemas de Arquitectura de Computadores*. Librería Fleming/Ed. Avicam, 2016.
- Secc. 10.2. J. Ortega, M. Anguita, A. Prieto. “Arquitectura de Computadores”. ESII/C.1 ORT arq

Contenido Lección 9

- Concepto de consistencia de memoria
- Consistencia secuencial
- Modelos de consistencia relajados

Consistencia de memoria

Modelo de consistencia
de memoria Software



Modelo de consistencia
de memoria Hardware

- Especifica (las restricciones en) **el orden** en el cual las **operaciones de memoria** (lectura, escritura) deben **parecer** haberse realizado (operaciones a las mismas o distintas direcciones y emitidas por el mismo o distinto proceso/procesador)
- La coherencia sólo abarca operaciones realizadas por múltiples componentes (proceso/procesador) en una misma dirección

Contenido Lección 9

- Concepto de consistencia de memoria
- Consistencia secuencial
- Modelos de consistencia relajados

Consistencia secuencial (SC)

- SC es el modelo de consistencia que espera el programador de las herramientas de alto nivel
- SC requiere que:
 - Todas las operaciones de un único procesador (thread) parezcan ejecutarse en el orden descrito por el programa de entrada al procesador (**orden del programa**)
 - Todas las operaciones de memoria parezcan ser ejecutadas una cada vez (**ejecución atómica**) -> serialización global

Inicialmente A=0,k=0

P1

A=1; W(A)
k=1; W(k)

P2

while (k=0) {};
R(k)
copia=A;
R(A)

Consistencia Secuencial

Inicialmente $k1=k2=0$

P1

```
k1=1;  
if (k2=0) {  
    Sección crítica  
};
```

P2

```
k2=1;  
if (k1=0) {  
    Sección crítica  
};
```

1

¿Qué espera el programador?

P1

```
A=1;
```

Inicialmente

P2
if ($A=1$)
 $B=1;$

P3
 $A=B=0$

```
if ( $B=1$ )  
reg1=A;
```

2

¿Qué espera el programador que se almacene en reg1 si llega a ejecutarse  $reg1=A$?

Inicialmente $A=0, k=0$

P1

```
A=1;  
k=1;
```

P2

```
while ( $k=0$ ) {};  
copia=A;
```

3

¿Qué espera el programador que se almacene en copia?

sincronización

Ejemplo de Consistencia Secuencial

P1 (1) Escribir K1=1

(2) Leer K2 ($\{K2==0\}$)

1

P2 (a) Escribir K2=1

(b) Leer K1 ($\{K1==0\}$)



Escribir

Orden con Consistencia Secuencial:

(1)(2)(a)(b) (a)(b)(1)(2)

(1)(a)(2)(b) (a)(1)(b)(2)

(1)(a)(b)(2) (a)(1)(2)(b)

P1 (1) Escribir A=1

(2) Escribir K=1

3

Orden con Consistencia Secuencial:

(1)(a)....(a)(2)(a)(b)

(1)(2)(a)(b)

(a)..(a)(1)(a)..(a)(2)(a)(b)

(a)..(a)(1)(2)(a)(b)

(a) Leer K (while k==0 {})

P2 (b) Leer A



Contenido Lección 9

- Concepto de consistencia de memoria
- Consistencia secuencial
- Modelos de consistencia relajados

Modelos de consistencia relajados

- Difieren en cuanto a los requisitos para garantizar SC que relajan (los relajan para incrementar prestaciones):
 - Orden del programa:
 - Hay modelos que permiten que se relaje en el código ejecutado en un procesador el orden entre dos acceso a distintas direcciones ($W \rightarrow R$, $W \rightarrow W$, $R \rightarrow RW$)
 - Atomicidad (orden global):
 - Hay modelos que permiten que un procesador pueda ver el valor escrito por otro antes de que este valor sea visible al resto de los procesadores del sistema
- Los modelos relajados comprenden:
 - Los órdenes de acceso a memoria que no garantiza el sistema de memoria (tanto órdenes de un mismo procesador como atomicidad en las escrituras).
 - Mecanismos que ofrece el hardware para garantizar un orden cuando sea necesario.

Ejemplos de modelos de consistencia hardware relajados

Modelo	Orden del programa relajado W→R W→W R→RW			Orden global Lec. anticipada propia de otro	Instrucciones para garantizar los órdenes relajados por el modelo	
Sparc-TSO, x86-TSO	Si			Si		I-m-e (instruc. lectura-modificación-escritura atómica)
Sparc-PSO	Si	Si		Si		I-m-e, STBAR (instrucción <i>STore BARrier</i>)
Sparc-RMO	Si	Si	Si	Si		MEMBAR (instrucción <i>MEMemory BARrier</i>)
PowerPC	Si	Si	Si	Si	Si	SYNC, ISYNC (instrucciones <i>SYNChronization</i>)
Itanium	Si	Si	Si	Si		LD.ACQ, ST.REL, MF (<i>ACQuisition LoAD, RELease STore, Memory Fence</i>), y cmpxchg8.acq y otras I-m-e
ARMv7	Si	Si	Si	Si	Si	DMB (<i>Data Memory Barrier</i>)
ARMv8	Si	Si	Si	Si	Si	LDA LDAR, STL STLR (<i>LoAD-Acquire, STore-reLease 32b 64b</i>), LDAEX LDAXR , STLEX STLXR (<i>LoAD-Acquire eXclusive, Store-reLease eXclusive 32b 64b</i>), DMB

Sigue la tabla de Adve y Gharachorloo en (biblioteca ugr) <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=546611&isnumber=11956>

Consistencia secuencial

Inicialmente $k1=k2=0$

P1

```
k1=1;  
if (k2=0) {  
    Sección crítica  
};
```

P2

```
k2=1;  
if (k1=0) {  
    Sección crítica  
};
```

NO se comporta como SC los que relajan el orden **W→R**

1

P1

```
A=1;
```

Inicialmente

P2
if ($A=1$)
 $B=1;$

P3
 $A=B=0$

```
if ( $B=1$ )  
reg1=A;
```

NO se comporta como SC los que no garantizan **atomicidad**

2

Inicialmente $A=0, k=0$

P1

```
A=1;  
k=1;
```

P2

```
while ( $k=0$ ) {};  
copia=A;
```

NO se comporta como SC los que relajan el orden **W→W o R→R**

3

Para ampliar ...

➤ Artículos en revistas

- Adve, S.V.; Gharachorloo, K.; , "Shared memory consistency models: a tutorial," *Computer* , vol.29, no.12, pp.66-76, Dec 1996. Disponible en:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=546611&isnumber=11956>

2º curso / 2º cuatr.
Grado en
Ing. Informática

Arquitectura de Computadores

Tema 3

Lección 10. Sincronización

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad
de Granada

Lecciones

- Lección 7. Arquitecturas TLP
- Lección 8. Coherencia del sistema de memoria
- Lección 9. Consistencia del sistema de memoria
- Lección 10. Sincronización
 - Comunicación en multiprocesadores y necesidad de usar código de sincronización
 - Soporte software y hardware para sincronización
 - Cerrojos
 - Cerrojos simples
 - Cerrojos con etiqueta
 - Barreras
 - Apoyo hardware a primitivas software

Objetivos Lección 10

- Explicar por qué es necesaria la sincronización en multiprocesadores.
- Describir las primitivas para sincronización que ofrece el hardware.
- Implementar cerros simples, cerros con etiqueta y barreras a partir de instrucciones máquina de sincronización y ordenación de accesos a memoria.

Bibliografía Lección 10

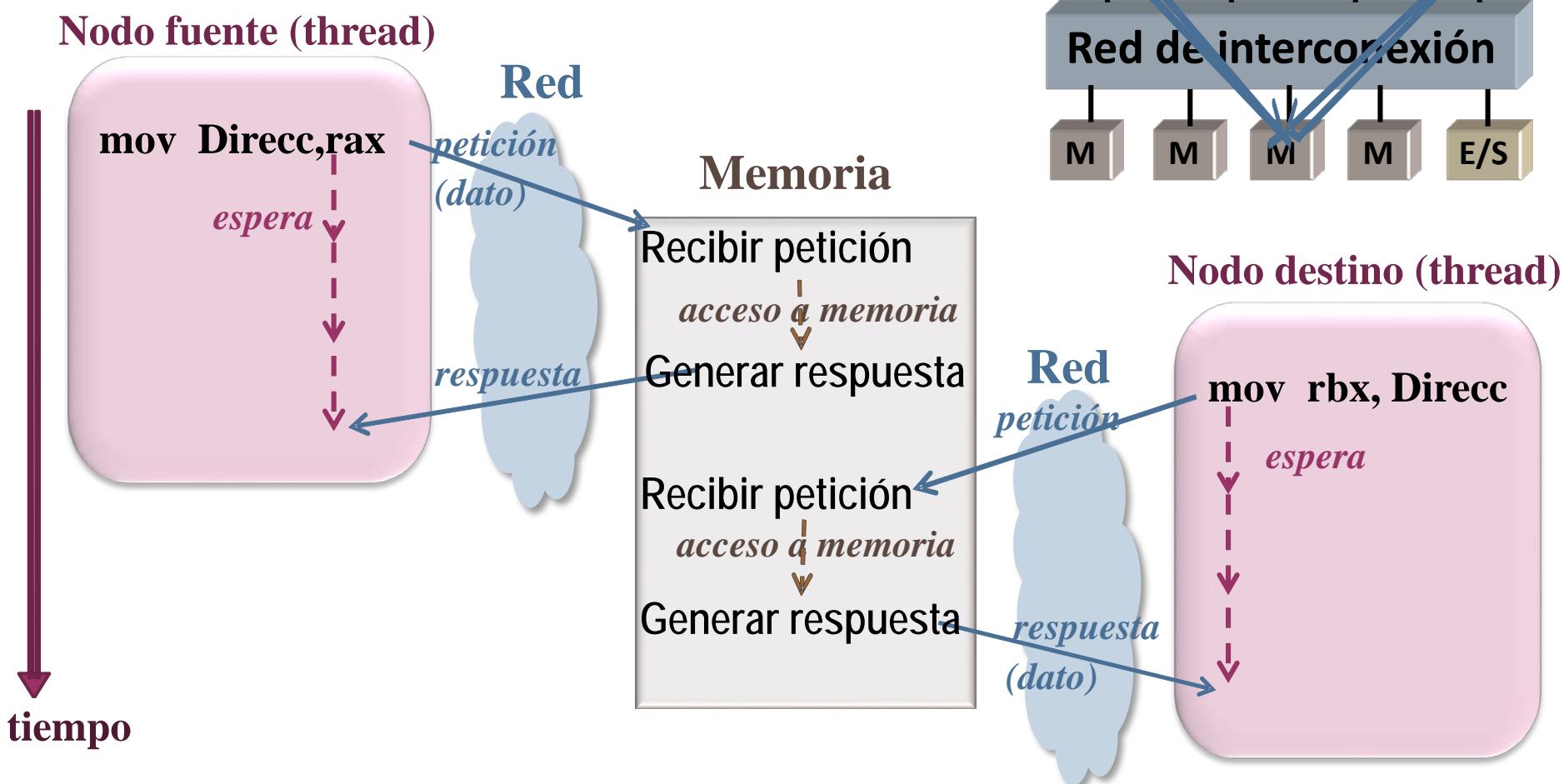
➤ Fundamental

- Cap.3, Secc. 5. M. Anguita, J. Ortega. *Fundamentos y Problemas de Arquitectura de Computadores*. Librería Fleming/Ed. Avicam, 2016.
- Secc. 10.3. J. Ortega, M. Anguita, A. Prieto. “Arquitectura de Computadores”. ESII/C.1 ORT arq

Contenido Lección 10

- Comunicación en multiprocesadores y necesidad de usar código de sincronización
- Soporte software y hardware para sincronización
- Cerrojos
- Barreras
- Apoyo hardware a primitivas software

Comunicación en un multiprocesador



Comunicación uno-a-uno

Secuencial	Paralela	
	<u>P1</u>	<u>P2</u>
... A=valor; ... copia=A; A=valor; copia=A; ...
... mov A,rx ... mov rbx,A mov A,rx mov rbx,A ...

Comunicación uno-a-uno. Necesidad de sincronización

- Se debe garantizar que el proceso que recibe lea la variable compartida cuando el proceso que envía haya escrito en la variable el dato a enviar
- Si se reutiliza la variable para comunicación, se debe garantizar que no se envía un nuevo dato en la variable hasta que no se haya leído el anterior

Paralela (inicialmente K=0)	
P1	P2
... A =1; K =1; <i>while (K==0) { };</i> copia= A ; ...

Comunicación colectiva

Secuencial	Paralela (sum=0)
<pre>for (i=0 ; i<n ; i++) { sum = sum + a[i]; } printf(sum);</pre>	<pre>for (<i>i</i>=<i>ithread</i> ; <i>i</i>< n ; <i>i</i>=<i>i</i>+<i>nthread</i>) { <i>sump</i> = <i>sump</i> + a[<i>i</i>]; } sum = sum + <i>sump</i>; /* SC, sum compart. */ if (<i>ithread</i>==0) printf(sum);</pre>

Race condition

- Ejemplo de comunicación colectiva: suma de n números:
 - La lectura-modificación-escritura de **sum** se debería hacer en exclusión mutua (es una sección crítica) => **cerrojos**
 - Sección crítica: Secuencia de instrucciones con una o varias direcciones compartidas (variables) que se deben acceder en exclusión mutua
 - El proceso 0 no debería imprimir hasta que no hayan acumulado **sump** en **sum** todos los procesos => **barreras**

Comunicación colectiva en multiprocesadores (carrera)

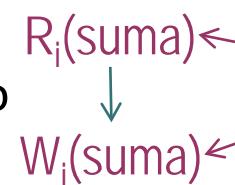
Sistema de memoria

	Ej.1	Ej.2	Ej.1	Ej.2	
R ₀ (suma)	1.0	2.0	R ₂ (suma)	3.1	1.0
W ₀ (suma)	2.1	3.1	W ₂ (suma)	4.4	4.3
R ₁ (suma)	5.4	7.7	R ₃ (suma)	7.6	5.3
W ₁ (suma)	6.6	8.9	W ₃ (suma)	8.10	6.7

Thread 0 **Thread 1**

Orden.resultado

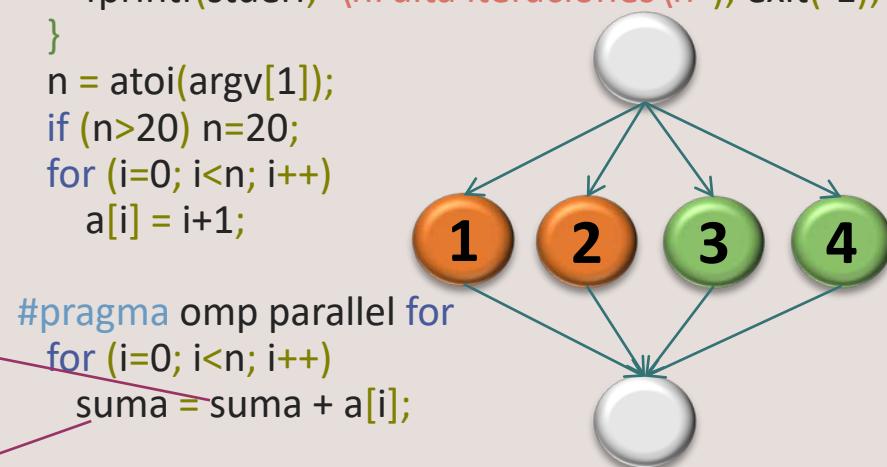
- Ej. para n=4, el compilador no optimiza
- a={1,2,3,4}
- R_i (suma): Lectura de suma en la iteración i



sin exclusión mutua en el acceso a suma

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
main(int argc, char **argv) {
    int i, n=20, a[n],suma=0;
    if(argc < 2) {
        fprintf(stderr,"\\nFalta iteraciones\\n"); exit(-1);
    }
    n = atoi(argv[1]);
    if (n>20) n=20;
    for (i=0; i<n; i++)
        a[i] = i+1;
```

```
#pragma omp parallel for
for (i=0; i<n; i++)
    suma = suma + a[i];
printf("Fuera de 'parallel' suma=%d\\n",suma);
return(0);
```

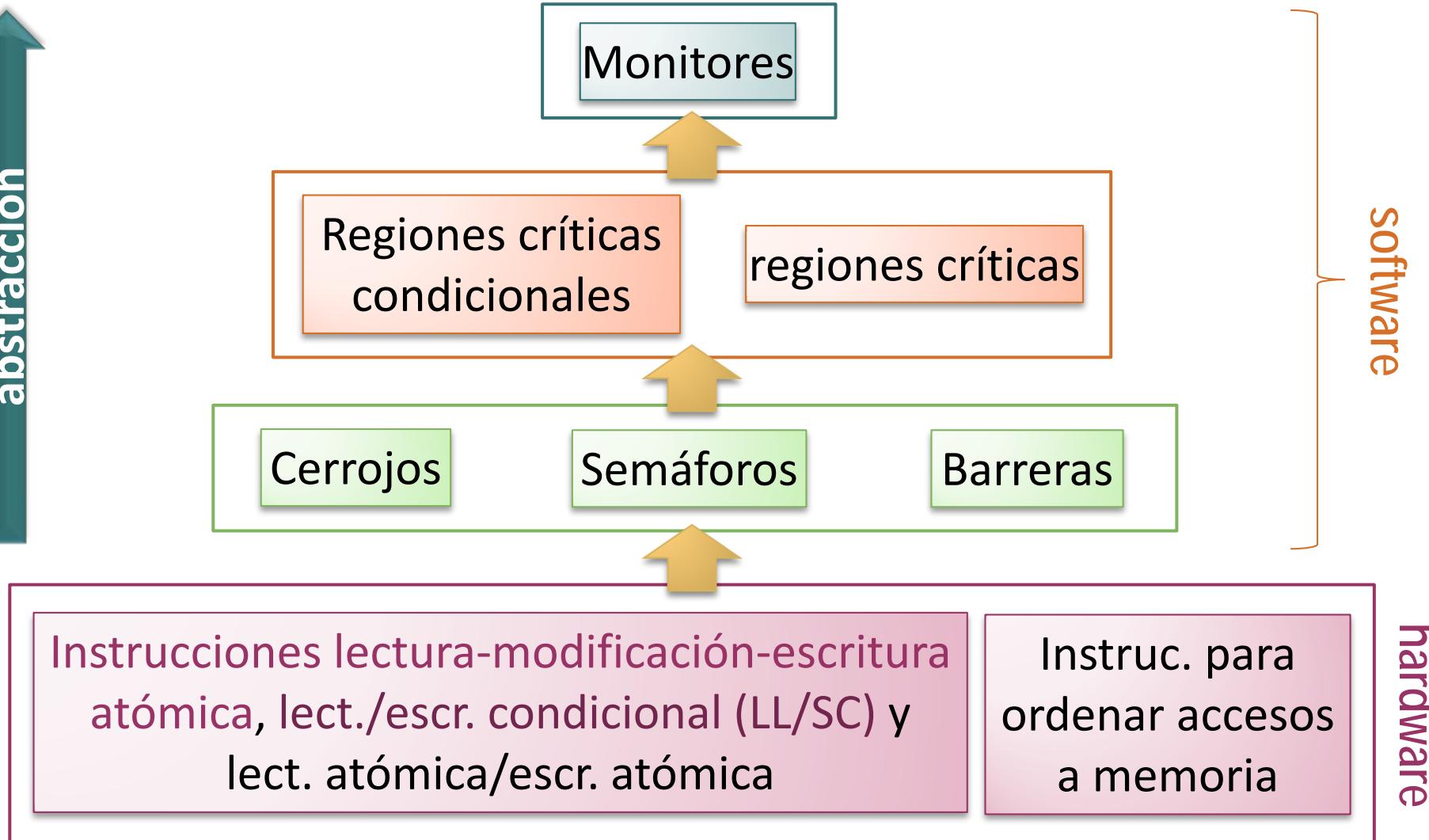


Contenido Lección 10

- Comunicación en multiprocesadores y necesidad de usar código de sincronización
- Soporte software y hardware para sincronización
- Cerrojos
- Barreras
- Apoyo hardware a primitivas software

Soporte software y hardware de sincronización

abstracción



Contenido Lección 10

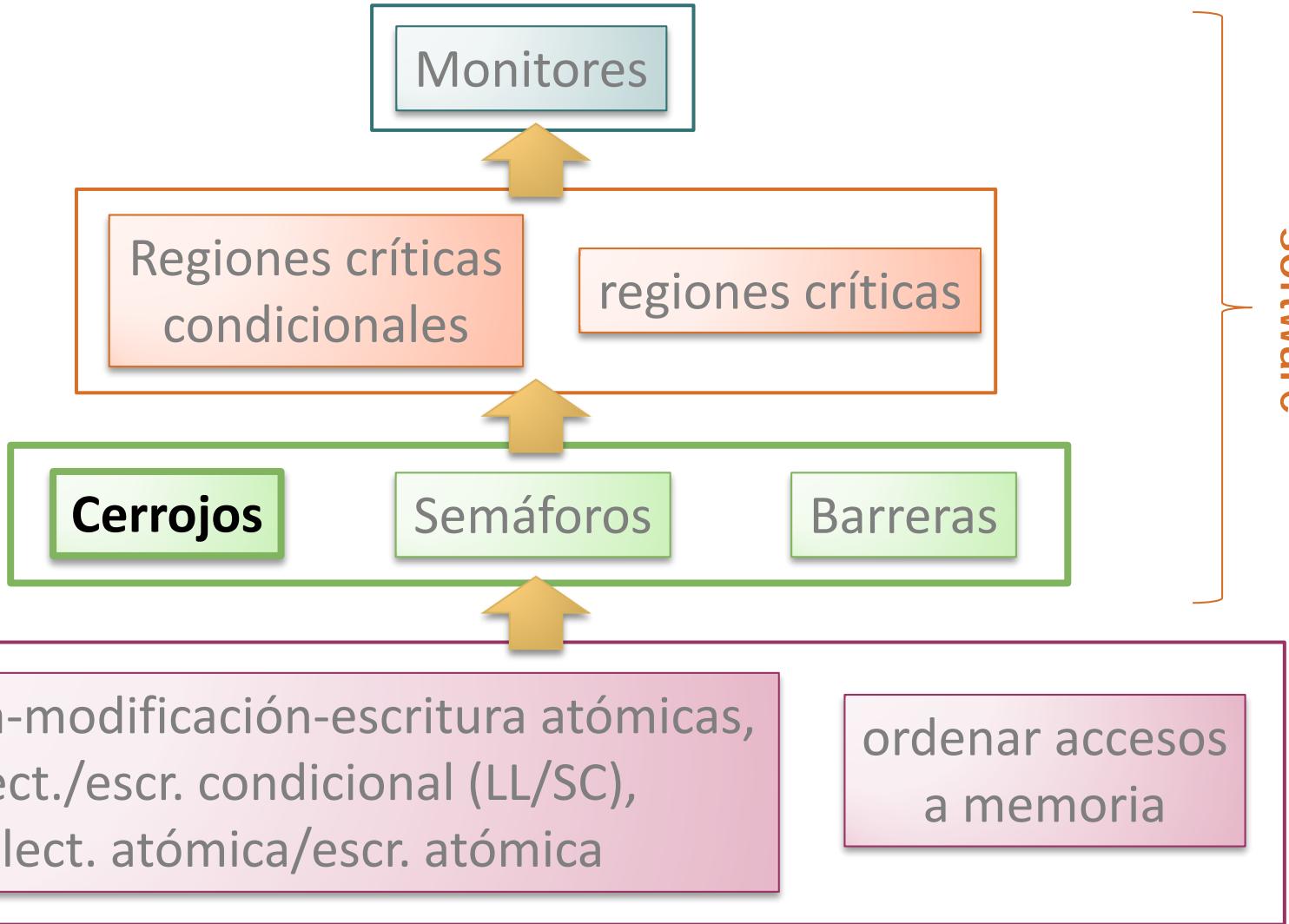
- Comunicación en multiprocesadores y necesidad de usar código de sincronización
- Soporte software y hardware para sincronización
- Cerrojos
 - Cerrojos simples
 - Cerrojos con etiqueta
- Barreras
- Apoyo hardware a primitivas software

Soporte software y hardware de sincronización

abstracción

software

hardware



Cerrojos

- Permiten sincronizar mediante dos operaciones:
 - Cierre del cerrojo o lock (k): intenta **adquirir** el derecho a acceder a una sección crítica (*cerrando* o *adquiriendo* el cerrojo k).
 - Si varios **threads** intentan la **adquisición** (cierre) a la vez, sólo uno de ellos lo debe conseguir, el resto debe pasar a una *etapa de espera*.
 - Todos los **threads** que ejecuten **lock ()** con el cerrojo cerrado deben quedar **esperando**.
 - Apertura del cerrojo o unlock (k): **libera** a uno de los **threads** que esperan el acceso a una sección crítica (éste *adquiere* el cerrojo).
 - Si no hay **threads** en **espera**, permitirá que el siguiente **thread** que ejecute la función **lock ()** adquiera el cerrojo k sin espera

Cerrojos en ejemplo suma

Secuencial	Paralela
<pre>for (i=0 ; i<n ; i++) { sum = sum + a[i]; }</pre>	<pre>for (<i>i</i>=<i>ithread</i> ; <i>i</i><<i>n</i> ; <i>i</i>=<i>i</i>+<i>nthread</i>) { <i>sump</i> = <i>sump</i> + a[<i>i</i>]; } lock(k); //código de adquisición sum = sum + sump; /* SC, sum compart. */ unlock(k); //código de liberación</pre>

- Alternativas para implementar la espera:
 - Espera ocupada.
 - Suspensión del proceso/thread, éste queda esperando en una cola, el procesador conmuta a otro proceso/thread.

Componentes en un código para sincronización

➤ Método de adquisición

➤ Método por el que un thread trata de adquirir el derecho a pasar a utilizar unas direcciones compartidas. Ej.:

- Utilizando lectura-modificación-escritura atómicas: Intel x86, Intel Itanium, Sun Sparc
- Utilizando **LL/SC (Load Linked / Store Conditional)**: IBM Power/PowerPC, ARMv7, ARMv8

➤ Método de espera

➤ Método por el que un thread espera a adquirir el derecho a pasar a utilizar unas direcciones compartidas:

- Espera ocupada (*busy-waiting*)
- Bloqueo

➤ Método de liberación

➤ Método utilizado por un thread para liberar a uno (cerrojo) o varios (barrera) threads en espera

Cerrojo Simple I

- Se implementa con una variable compartida `k` que toma dos valores: abierto (0), cerrado (1)
- **Apertura** del cerrojo, `unlock(k)`: abre el cerrojo escribiendo un 0 (*operación invisible*)
- **Cierre** del cerrojo, `lock(k)`: Lee el cerrojo y lo cierra escribiendo un 1.
 - **Resultado de la lectura:**
 - si el cerrojo **estaba cerrado** el thread espera hasta que otro thread ejecute `unlock(k)`,
 - si **estaba abierto** adquiere el derecho a pasar a la sección crítica.
 - **leer-assignar_1-escibir en el cerrojo debe ser invisible (atómica)**

Cerrojo Simple II

- Se debe añadir lo necesario para garantizar el acceso en exclusión mutua a k y el orden imprescindible en los accesos a memoria

lock (k)

```
lock(k) {
    while (leer-asignar_1-escribir(k) == 1) {};
} /* k compartida */
```

unlock (k)

```
unlock(k) {
    k = 0 ;
} /* k compartida */
```

Cerrojos en OpenMP

Descripción	Función de la biblioteca OpenMP
Iniciar (estado unlock)	omp_init_lock(&k)
Destruir un cerrojo	omp_destroy_lock(&k)
Cerrar el cerrojo lock(k)	omp_set_lock(&k)
Abrir el cerrojo unlock(k)	omp_unset_lock(&k)
Cierre del cerrojo pero sin bloqueo (devuelve 1 si estaba cerrado y 0 si está abierto)	omp_test_lock(&k)

Cerrojos con etiqueta

- Fijan un orden FIFO en la adquisición del cerrojo (se debe añadir lo necesario para garantizar el acceso en exclusión mutua al contador de adquisición y el orden imprescindible en los accesos a memoria):

lock (contadores)

```
contador_local_adq = contadores.adq;  
contadores.adq = (contadores.adq + 1) mod max_flujos;  
while (contador_local_adq != contadores.lib) {};
```

unlock (contadores)

```
contadores.lib = (contadores.lib + 1) mod max_flujos;
```

Contenido Lección 10

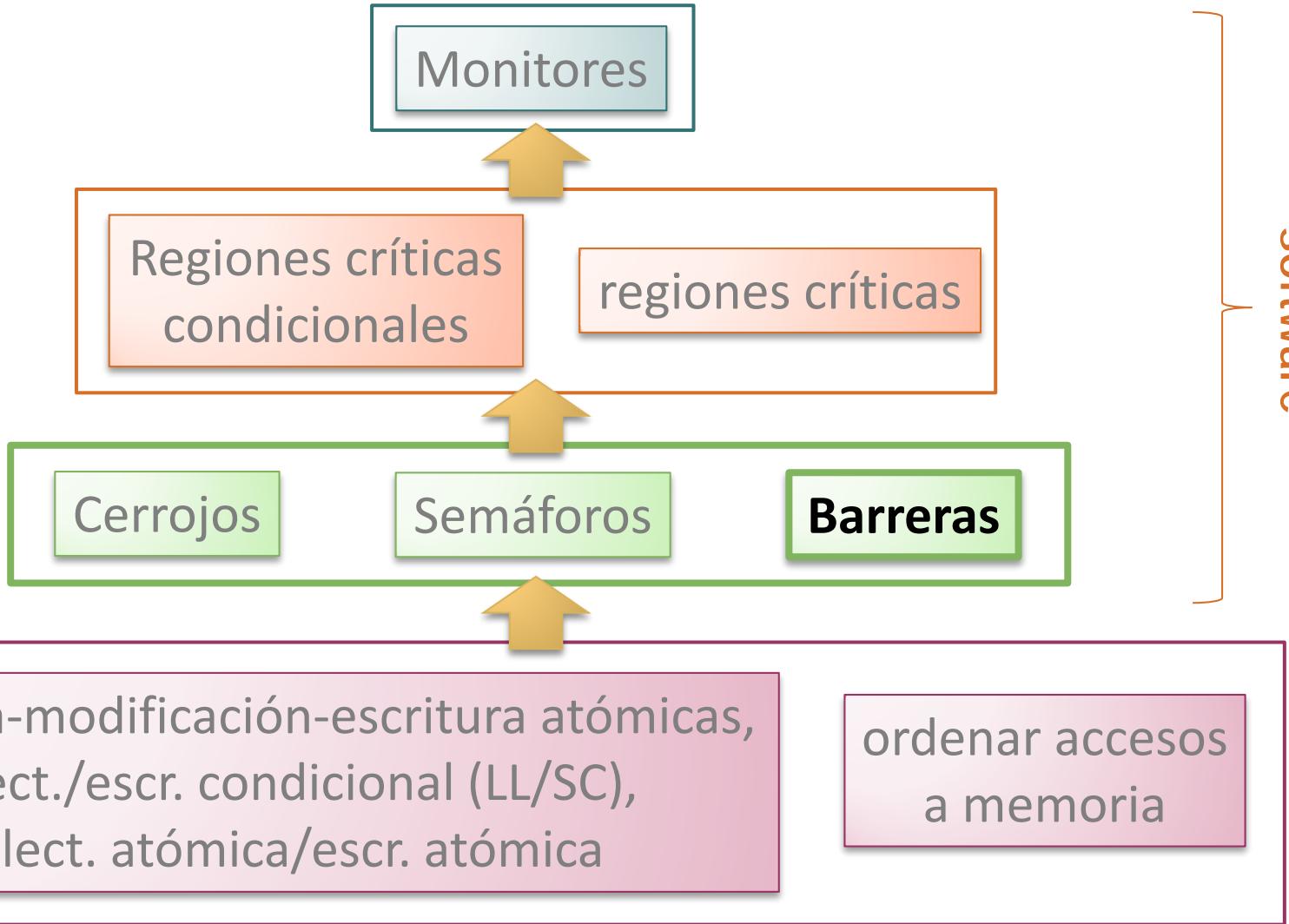
- Comunicación en multiprocesadores y necesidad de usar código de sincronización
- Soporte software y hardware para sincronización
- Cerrojos
- Barreras
- Apoyo hardware a primitivas software

Soporte software y hardware de sincronización

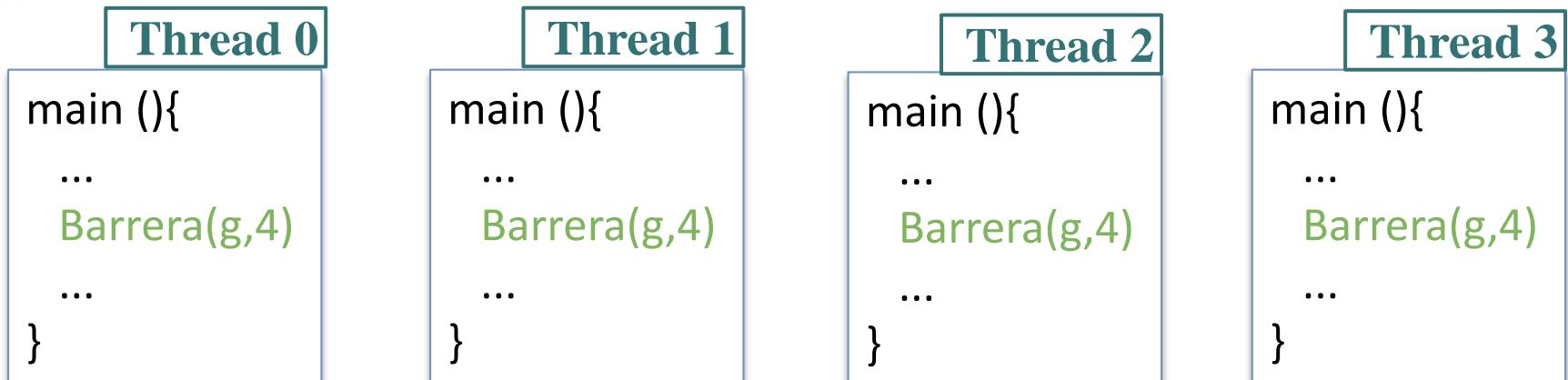
abstracción

software

hardware



Barreras



Barrera(id, num_threads) {

if (**bar[id].cont==0**) **bar[id].bandera=0**;

cont_local = ++bar[id].cont;

if (**cont_local ==num_threads**) {

bar[id].cont=0;

bar[id].bandera=1;

}

else **espera mientras bar[id].bandera=0;**

}

- Acceso Ex. Mutua.

- Implementar **espera**. Si espera ocupada:

while (bar[id].bandera==0) {};

Barreras sin problema de reutilización

Barrera *sense-reversing*

Barrera(id, num_procesos) {

bandera_local = !(*bandera_local*) //se complementa bandera local
 lock(**bar[id].cerrojo**);

cont_local = ++**bar[id].cont** //*cont_local* es privada

 unlock(**bar[id].cerrojo**);

 if (*cont_local* == num_procesos) {

bar[id].cont = 0; //se hace 0 el cont. de la barrera

bar[id].bandera = *bandera_local*; //para liberar thread en espera

 }

 else while (**bar[id].bandera** != *bandera_local*) {}; //espera ocupada

}

Contenido Lección 10

- Comunicación en multiprocesadores y necesidad de usar código de sincronización
- Soporte software y hardware para sincronización
- Cerrojos
- Barreras
- Apoyo hardware a primitivas software
 - Instrucciones de lectura-modificación-escritura atómicas
 - Instrucciones LL/SC (*Load Linked / Store Conditional*)

Soporte software y hardware de sincronización



Instrucciones de lectura-modificación-escritura atómicas

Test&Set (x)

```
Test&Set (x) {  
    temp = x ;  
    x = 1 ;  
    return (temp) ;  
}  
/* x compartida */
```

x86

```
mov reg,1  
xchg reg,mem  
reg ↔ mem
```

Fetch&Oper(x,a)

```
Fetch&Add(x,a) {  
    temp = x ;  
    x = x + a ;  
    return (temp);  
}/* x compartida,  
a local */
```

x86

```
lock xadd reg,mem  
reg ← mem |  
mem ← reg+mem
```

Compare&Swap(a,b,x)

```
Compare&Swap(a,b,x){  
    if (a==x) {  
        temp=x ;  
        x=b; b=temp ; }  
}/* x compartida,  
a y b locales */
```

x86

```
lock cmpxchg mem,reg  
if eax=mem  
then mem ← reg  
else eax ← mem
```

Cerrojos simples con Test&Set y Fetch&Or

con Test&Set (x)

```
lock(k) {  
    while (test&set(k)==1) {};  
}  
/* k compartida */
```

x86

```
lock:    mov    eax,1  
repetir: xchg   eax,k  
          cmp    eax,1  
          jz     repetir
```

con Fetch&Oper(x,a)

```
lock(k) {  
    while (fetch&or (k,1)==1) {};  
}  
/* k compartida */
```

{ true (1, cerrado)
 false (0, abierto)

Cerrojo simple

Cerrojos simples con Compare&Swap

con **Compare&Swap(a,b,x)**

```
lock(k) {  
    b=1  
    do  
        compare&swap(0,b,k) ;  
    while (b==1);  
}  
/* k compartida, b local */
```

```
compare&swap(0,b,k){  
    if (0==k) { b=k | k=b; }  
}
```

Cerrojo simple en Itanium (consistencia de liberación) con Compare&Swap



lock:	//lock(M[lock])	Compare&Swap
mov ar.ccv = 0	// cmpxchg compara con ar.ccv	
	// que es un registro de propósito específico	
mov r2 = 1	// cmpxchg utilizará r2 para poner el cerrojo a 1	
spin:	// se implementa espera ocupada	
ld8 r1 = [lock] ;;	// carga el valor actual del cerrojo en r1	
cmp.eq p1,p0 = r1, r2;	// si r1=r2 entonces cerrojo está a 1 y se hace p1=1	
(p1) br.cond.spnt spin ;;	// si p1=1 se repite el ciclo; spnt indica que se // usa una predicción estática para el salto de “no tomar”	
cmpxchg8.acq r1 = [lock], r2 ;;	// intento de adquisición escribiendo 1 // IF [lock]=ar.ccv THEN [lock]←r2; siempre r1←[lock]	
cmp.eq p1, p0 = r1, r2	// si r1!=r2 (r1=0) => cer. era 0 y se hace p1=0	
(p1) br.cond.spnt spin ;;	// si p1=1 se ejecuta el salto	
unlock:	//unlock(M[lock])	Consistencia
st8.rel [lock] = r0 ;;	//liberar asignando un 0, en Itanium r0 siempre es 0	

Cerrojo simple en PowerPC(consistencia débil) con LL/SC implementando Test&Set

lock: #lock(M[r3]) Test&Set

 li r4,1 #para cerrar el cerrojo

bucle: **lwarx** r5,0,r3 #carga y reserva: $r5 \leftarrow M[r3]$

 cmpwi r5,0 #si está cerrado (a 1)

 bne- bucle #esperar en el bucle, en caso contrario

stwcx. r4,0,r3 #poner a 1 ($r4=1$): $M[r3] \leftarrow r4$

 bne- bucle #el thread repite si ha perdido la reserva

isync #accede a datos compartidos cuando sale del bucle



unlock: # unlock(M[r3])

sync #espera hasta que terminen los accesos anteriores

 li r1,0

stw r1,0(r3) #abre el cerrojo

Cerrojo simple en ARMv7 (consistencia débil) con LL/SC implementando Test&Set

lock:

#lock(M[r1])

Test&Set

bucle:

mov r0, #1 #Para posteriormente cerrar el cerrojo asigna a r0 un 1

ldrex r5, [r1] #Lee cerrojo

cmp r5, #0 #Comprueba si el cerrojo es 0 (abierto)

strexeq r5, r0, [r1] #Si el cerrojo está abierto intenta escribir (un 1)

cmpeq r5, #0 #Comprueba si ha tenido éxito la escritura

bne bucle #Vuelve a intentarlo si no ha tenido éxito

dmЬ #Para asegurar que se ha adquirido el cerrojo antes de ...

#realizar los accesos a memoria que hay después del lock

Consistencia

unlock:

unlock(M[r1])

dmЬ #Espera a que terminen los accesos anteriores ...

mov r0, #0 #antes de abrir el cerrojo

str r0, [r1] #Abre el cerrojo

Cerrojo simple en ARMv8 (consistencia liberación) con LL/SC para Test&Set

lock:

#lock(M[r1])

Test&Set

bucle: **ldae**x r5, [r1]
 cmp r5, #0

#Para posteriormente cerrar el cerrojo asigna a r0 un 1
#Lee cerrojo con ordenación de adquisición
#Comprueba si el cerrojo es 0 (abierto)

strexeq r5, r0, [r1] #Si el cerrojo está abierto intenta escribir (un 1)
cmpeq r5, #0
bne bucle

#Comprueba si ha tenido éxito la escritura
#Vuelve a intentarlo si no ha tenido éxito

unlock:

unlock(M[r1])

Consistencia
de liberación

mov r0, #0
stl r0, [r1]

#Abre el cerrojo usando almacenamiento con liberación

Algoritmos eficientes con primitivas hardware

Suma con fetch&add

```
for (i=ithread; i<n; i=i+nthread)
    fetch&add(sum,a[i]);
/* sum variable compartida */
```

Suma con fetch&add

```
for (i=ithread; i<n; i=i+nthread)
    sump = sump + a[i];
fetch&add(sum,sump);
/* sum variable compartida */
```



Suma con compare&swap

```
for (i=ithread; i<n; i=i+nthread)
    sump = sump + a[i];
do
    a = sum;
    b = a + sump;
    compare&swap(a,b,sum);
    while (a!=b);
/* sum variable compartida */
```

Para ampliar ...

➤ Webs

- Implementación en el kernel de linux de cerrojos con etiqueta <http://lxr.free-electrons.com/source/arch/x86/include/asm/spinlock.h>

➤ Artículos en revistas

- Graunke, G.; Thakkar, S.; , "Synchronization algorithms for shared-memory multiprocessors," *Computer* , vol.23, no.6, pp.60-69, Jun 1990. Disponible en (biblioteca ugr):
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=55501&isnumber=2005>