Perfil7
www.wuolah.com/student/Perfil7

ApuntesTEMA3AC.pdf

APUNTES con diapositivas y libro

2° Arquitectura de Computadores

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

- ARQUITECTURA DE COMPUTADORES -

Tema 3

Lección 7

CLASIFICACIÓN DE ARQUITECTURAS CON TLP EXPLÍCITO Y UNA INSTANCIA DE SO

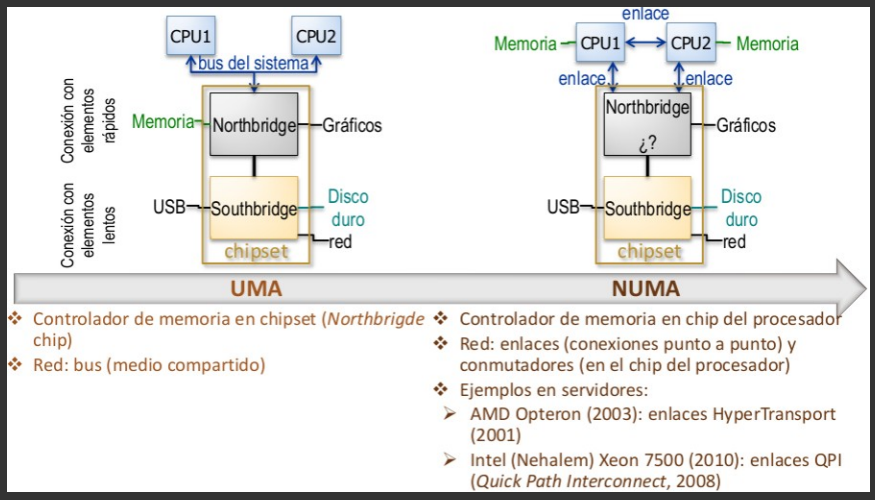
- Multiprocesador → Ejecuta varios threads en paralelo en un computador con varios cores/procesadores, donde cada thread es implementado como core/procesador distinto
- Multicore (CMP) → Ejecuta varios threads en paralelo en un chip de procesamiento multicore, donde cada thread se implementa como un core distinto
- Multithread → Core multithread que modifica su arquitectura ILP para poder implementar threads que se ejecutan concurrentemente y en paralelo

Recordatorio:

NUMA => ↑Latencia ↓Escalabilidad - Comparten memoria

UMA => ↓Latencia ↑Escalabilidad - No comparten memoria (necesita implementar sincronización, distribución de datos y código)

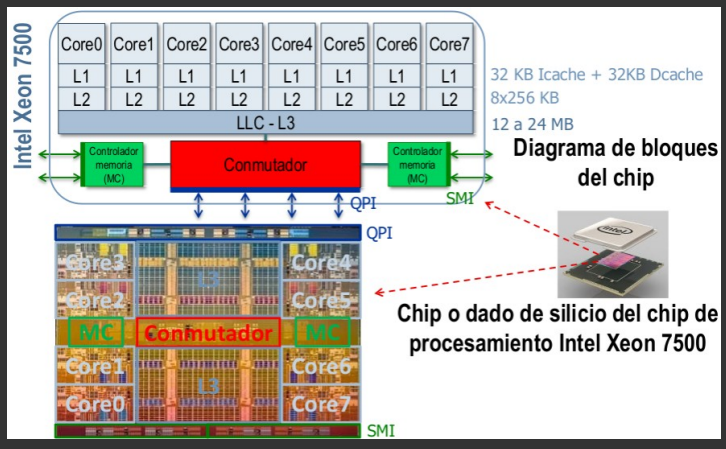
MULTIPROCESADOR EN UNA PLACA (EVOLUCIÓN DE UMA A NUMA)



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

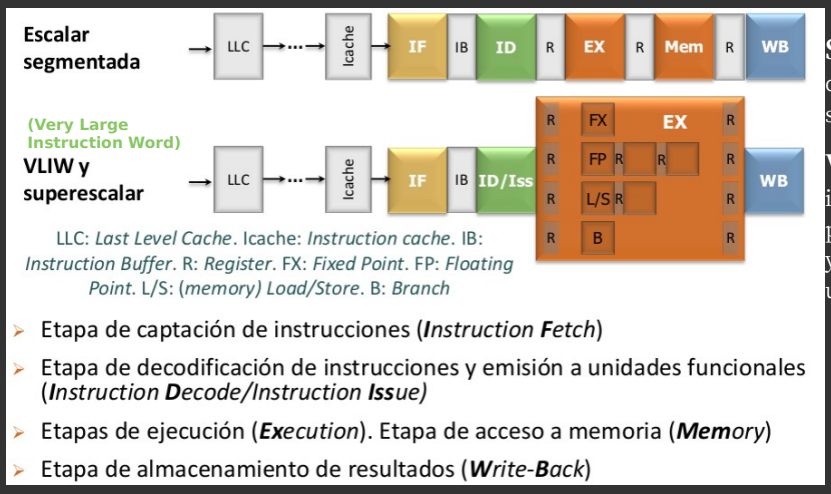
- Controlador de memoria en chip del procesador
- Red: enlaces (conexiones punto a punto) y conmutadores (en el chip del procesador)
- Ejemplos en servidores:
  - AMD Opteron (2003): enlaces HyperTransport (2001)
  - Intel (Nehalem) Xeon 7500 (2010): enlaces QPI (*Quick Path Interconnect*, 2008)

MULTIPROCESADOR EN UNA PLACA (EVOLUCIÓN DE UMA A NUMA)



Podemos encontrar otras distribuciones tales que el LLC (Last Level Cache) no sea común y agrupe varios cores, solo un core o que no haya LLC.

ARQUITECTURAS ILP



Segmentados : Ejecutan las instrucciones de forma concurrente, segmentando el uso de sus compontentes

VLIW y superescalares : Ejecutan instrucciones concurrentemente y además en paralelo (tienen múltiples unidades funcionales y emiten múltiples instrucciones en paralelo a unidades funcionales)

- VLIW : Instrucciones que se ejecutan en paralelo y se captan juntas en memoria, de manera que forman una palabra de instrucción muy grande. El hardware presupone que las instrucciones de una palabra son independientes, por tanto no tiene que encontrar instrucciones que puedan emitirse y ejecutarse en paralelo.
- Superescalares : Tiene que encontrar instrucciones que puedan emitirse y ejecutarse en paralelo, el hardware tiene que extraer paralelismo a nivel de instrucción.

En cores multithread, tanto el almacenamiento como el hardware se deben gestionar para cada thread, de forma que:

-Almacenamiento	Se multiplexa	-Hardware	Se multiplexa entre threads	NO SMT
	Se reparte		Se reparte entre threads	
	Se comparte		Se comparte entre threads	
-Almacenamiento	Se replica (lo + caro)	-Hardware	Se multiplexa entre threads	SMT
	Se reparte		Se reparte entre threads	
	Se comparte		Se comparte$_{(EX)}$ entre threads	

CLASIFICACIÓN DE CORES MULTITHREAD

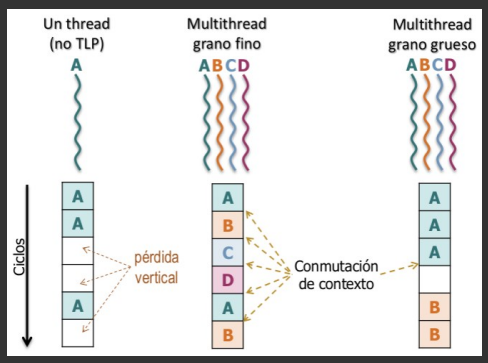
TMP Temporal Multithreading	SMT Simultaneous Multithreading	FGMT Fine-Grain Multithreading	CGMT Coarse-Grain Multithreading
• Ejecutan varios threads concurrentemente en el mismo core • La conmutación entre threads la decide el harware • Emite instrucciones de un único thread en un ciclo	• Ejecutan en un core superescalar, varios threads de forma paralela • Pueden emitir, por tanto, instrucciones de varios threads en un mismo ciclo	La conmutación entre threads la decide el hardware cada ciclo a coste 0. Lo hace implementado round robin o bien por eventos de cierta latencia combinado con técnicas de planificación	La conmutación entre threads la decide el hardware a coste de 0 a varios ciclos. Con intervalos de tiempo prefijados *timeslice multithreading* o por eventos de cierta latencia *swithc-on-event multithreading*

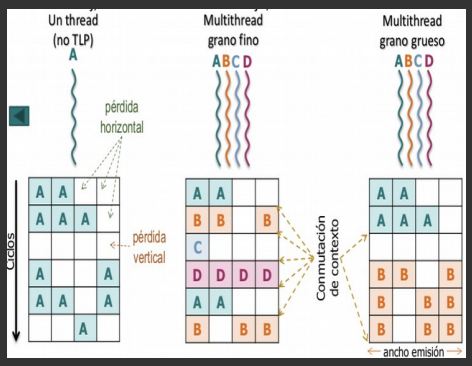
CGMT puede conmutar de forma estática o dinámica:

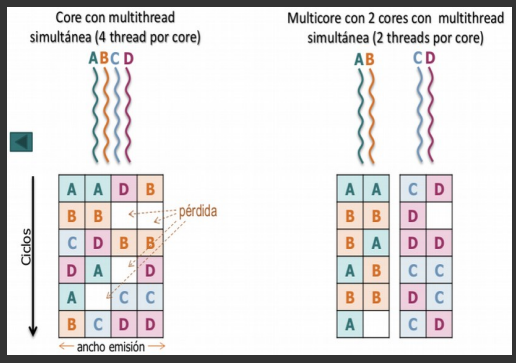
- Estática
  - Explícita: instrucciones explícitas para conmutación.
  - Implícita: instrucciones de carga, almacenamiento, salto...
  - Ventaja/Inconveniente → Coste bajo de cambio de contexto / cambios de contexto innecesarios
- Dinámica
  - Conmutación por fallo en la última caché dentro del chip de procesamiento, por interrupción, etc
  - Ventaja/Inconveniente → Reduce cambios de contexto innecesarios / coste mayor cambio contexto

Trazas o ejemplos de ejecuciones:

Core escalar **TMP?** **SMP**







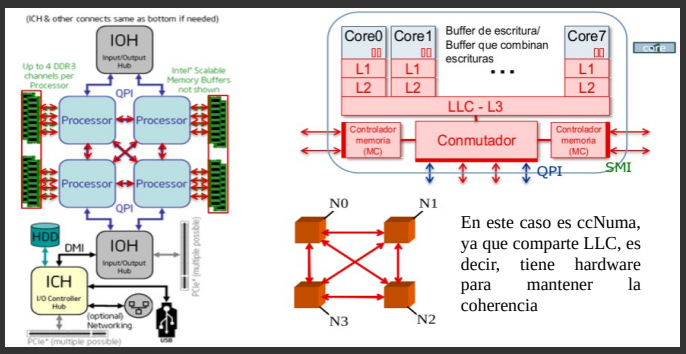
HARDWARE Y ARQUITECTURAS TLP EN UN CHIP

Hardware	CGMT	FGMT	SMT	CMP
Registros	**replicado** (al menos PC)	**replicado**	**replicado**	**replicado**
Almacenamiento	multiplexado	multiplexado, compartido, repartido o **replicado**	compartido, repartido o **replicado**	**replicado**
Otro hardware de las etapas del cauce	multiplexado	Captación: repartida o compartida; Resto: multiplexadas	UF: compartidas; Resto: 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**

SISTEMA DE MEMORIA EN MULTIPROCESADORES

Dichos sistemas incluyen:

- Cachés de todos los nodos
- Memoria principal
- Controladores
- Buffers
- Red de interconexión



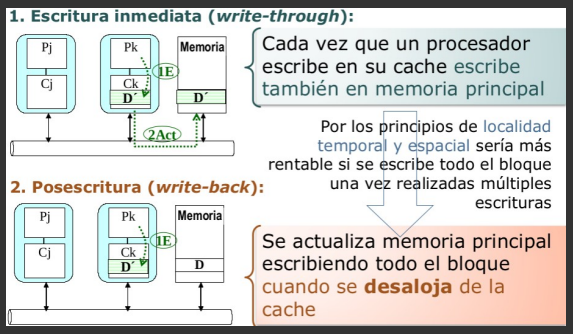
En este caso es ccNuma, ya que comparte LLC, es decir, tiene hardware para mantener la coherencia

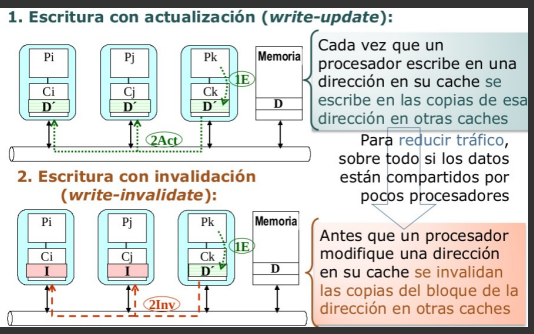
INCOHERENCIA EN EL SISTEMA DE MEMORIA

Estas pueden ser, por ejemplo:

Datos modificables que en E/S dan falta de coherencia en caché-MP
Datos modificables privados que al emigrar thread/proceso dan falta de coherencia de caché-MP
Datos modificables compartidos que al encontrar fallo de cache, se genera falta de coherencia caché-MP
Datos modificables compartidos que hacen lectura de caché no actualizada, falta de coherencia cache-cache

Para evitar estas faltas de cache, existen métodos de actualización de memoria principal para cache





El sistema de memoria necesita implementar de alguna forma protocolos para afrontar la incoherencia en sistemas de memoria:

Propagar las escrituras en una dirección → La escritura se hace visible durante un tiempo prudencial a otros procesadores. Los componentes conectados mediante un bus trabajan a través de paquetes de actualización/validación visibles a todos los nodos del bus (controladores de cache).

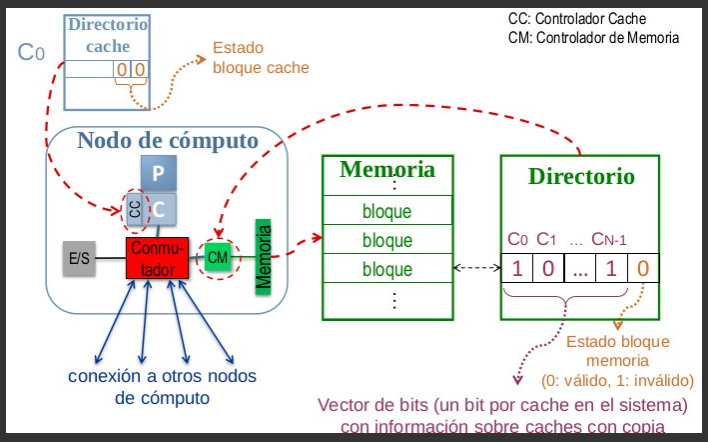
Mediante difusión, es decir, los paquetes de actualización/validación se envían a todas las caches.

Por otro lado, para una mayor escalabilidad, solo se envían los paquetes a los nodos con copia del bloque. Cada bloque tiene un directorio donde se guardan los nodos con copia del mismo.

Serializar las escrituras en una dirección → Las escrituras en una dirección deben realizarse en el mismo orden para todos los procesadores. Es equivalente a una implementación de las operaciones del sistema de memoria en serie. Para los componentes conectados por un bus, dicho orden es determinado por el orden en que los paquetes aparecen en el bus.

El ordenen el que las peticiones de escritura llegan al home(nodo que tiene en MP la dirección), o al directorio centralizado, sirven como referencia de orden.

DIRECTORIO DE MEMORIA PRINCIPAL



Podemos implementar los directorios de forma centralizada:

Todos los nodos comparten la información de todos los módulos de memoria en un directorio centralizado.

o de forma distribuida:

Las filas se distribuyen entre los nodos. De forma que cada nodo tiene acceso únicamente a su módulo de memoria distribuido.

CLASIFICACIÓN DE PROTOCOLOS PARA MANTENER COHERENCIA EN MEMORIA

- Protocolos de espionaje (snoopy)

  Para buses y sistemas con una difusión eficiente. Y se basa en que todos los nodos pueden qué datos se están usando mediante espionaje al bus.

- Protocolos basados en directorios

  Para redes sin difusión o escalables, donde solo se pasa la información del bloque a los nodos implicados.

- Esquemas jerárquicos

  Para redes jerárquicas: jerarquías de buses, de redes escalables o de redes escalables-buses.

Para tales protocolos debemos tener en cuenta que en su diseño se utiliza:

- Política de actualización de MP

  Escritura inmediata(write through), posescritura(write back) o mixta.

- Política de coherencia entre cachés

  Escritura con invalidación, escritura con actualización y mixta.

- Describir comportamiento

  Definir posibles estados de los bloques en cache y en memoria

  Definir transferencias(nodos que intervienen y orden) ante eventos

  Definir transiciones de estado para un bloque en cache y en memoria

PROTOCOLO DE ESPIONAJE : MSI

Modificado (M) : La única copia del bloque válida de todo el sistema

Compartido (C,S) : Es válido, pero también es válido en memoria y puede que se encuentre en otras caches

Invalido (I) : No es válido o ni siquiera está físicamente

En un bloque de memoria también se puede almacenar info Válido(puede haber copia válida en una o varias caches) o Inválido(habrá copia válida en una cache), pero se evita.

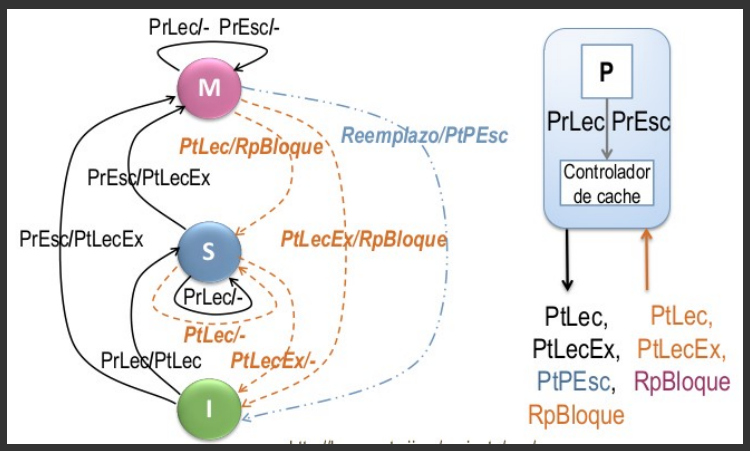
Se pueden dar las siguientes transferencias generadas por un nodo con cache (tipos de paquetes):

-Petición de lectura de un bloque (PtLec) → Se da cuando el procesador desea acceder a un dato que no tiene en la caché del nodo. Se llama al controlador de cache mediante esta operación y este devuelve el dato de cache o de MP. Debido a que pasa la info del bloque al bus, el bloque siempre acaba en compartido tras la ejecución.

-Petición de acceso exclusivo (PtLecEx) → Se da cuando el procesador trata de escribir (PrEsc) en un bloque compartido o inválido.

-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 las anteriores PtLec o PtLecEx.



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 (posescr)	Genera paquete PtLecEx (PtEx)	Modificado
	PtLec		Compartido
	PtLecEx (invalido)	Invalida copia local	Inválido
Inválido (I)	PrLec	Genera paquete PtLec	Compartido
	PrEsc	Genera paquete PtLecEx	Modificado
	PtLec/PtLecEx		Inválido

PROTOCOLO DE ESPIONAJE : MSI

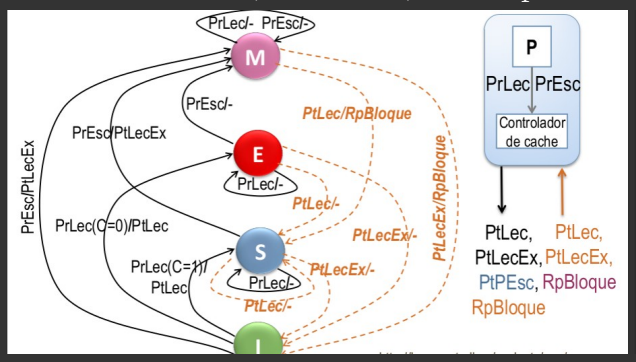
Modificado (M) : La única copia del bloque válida de todo el sistema

Exclusivo (E) : Es la única copia válida del bloque en caches, la memoria también está actualizada

Compartido (C,S) : Es válido, pero también es válido en memoria y puede que se encuentre en otras caches

Invalido (I) : No es válido o ni siquiera está físicamente

En un bloque de memoria también se puede almacenar info Válido(puede haber copia válida en una o varias caches) o Inválido(habrá copia válida en una cache), pero se evita.



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	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

Lección 9

CONSISTENCIA DE MEMORIA

La consistencia de memoria es la que especifica el orden en el que se deben realizar las operaciones de memoria (R/W) para que no se pierda información y el resultado tenga sentido lógico.

La coherencia solo abarca operaciones realizadas en una misma dirección.

CONSISTENCIA SECUENCIAL (SC)

Es el modelo de consistencia que un programador espera de las herramientas de alto nivel.

De forma que todas las operaciones de un único procesador parezcan ejecutarse en el orden establecido en el programa y todas las operaciones de memoria parezcan ser ejecutadas atómicamente.

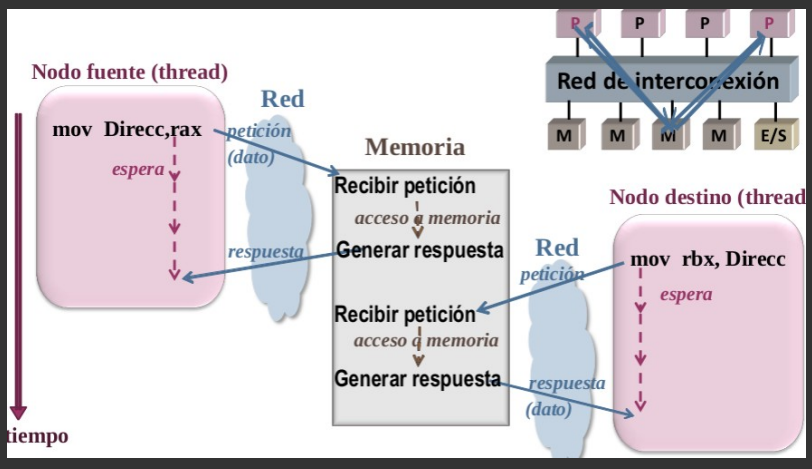
MODELOS DE CONSISTENCIA RELAJADOS

Estos modelos se usan como recurso para incrementar prestaciones y se basan en la relajación de los ordenes del código, como pueden ser W→W, W→R o R→RW

También, hay algunos de estos modelos que permiten la atomicidad (mientras un procesador accede a un valor, el resto de procesadores no tienen acceso).

Lección 10

MODELOS DE CONSISTENCIA RELAJADOS



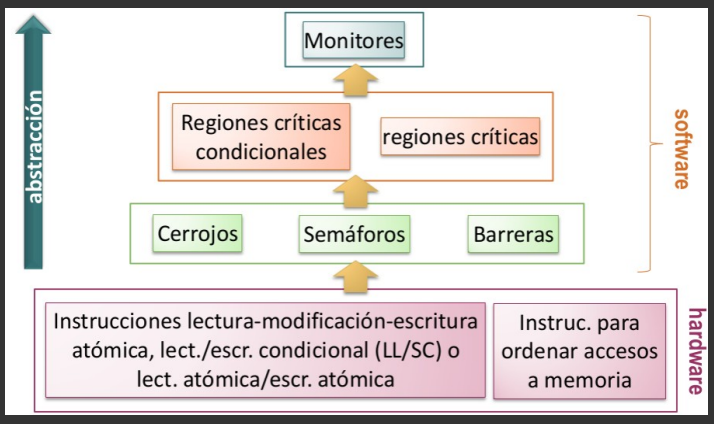
Comunicación uno-a-uno

Necesita sincronización

Comunicación colectiva

Usa cerrojos y barreras para poder colectivizar una tarea.

SOPORTE SOFTWARE Y HARDWARE DE SINCRONIZACIÓN



CERROJOS

Basados en la sincronización mediante dos operaciones:

- Cierre del cerrojo lock(k) → adquiere el derecho a acceder a una sección crítica cerrando el cerrojo k
- Apertura del cerrojo unlock(k) → libera a uno de los threads que esperan el acceso a una sección crítica