Chapter 2
Memory Architecture for Multiprocessing

Outline

- Memory and Communication Issues
- Advanced Memory Hierarchy: Virtual Machines
- Multicore Memory Architecture: Commercial Examples

Memory and Communication Issues

- ***Memory and Communication Issues***
  - Flynn‘s Taxonomy of Computer Categorization
  - Shared Memory Architecture: SMP/DSM
  - Shared Memory Communication
  - Cache Coherence in Shared Memory Architecture
  - Distributed Memory Architecture
  - Distributed Memory Communication: Message-Passing
  - Shared Memory vs. Distributed Memory
  - Memory Consistency
  - Parallel Computer Classification via Memory View
- Advance Memory Hierarchy: Virtual Machines
- Multicore Memory Architecture: Commercial Examples

Flynn‘s Taxonomy of Computer Categorization (1/2):

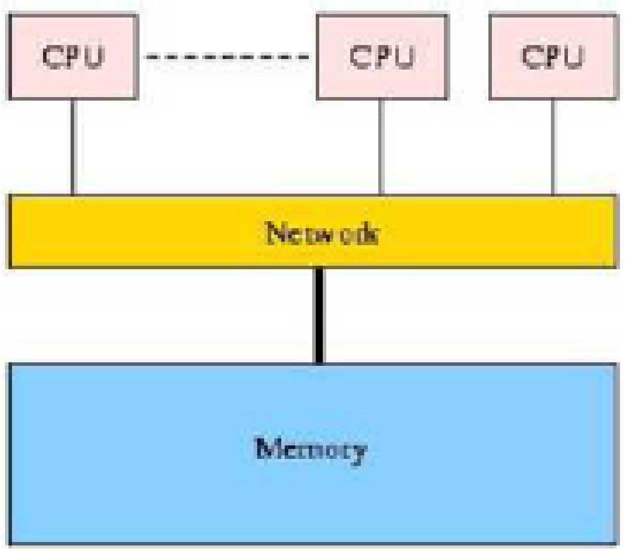
- Flynn's taxonomy is a classification of computer architectures, proposed by Michael J. Flynn in 1966
  - Single Instruction stream, Single Data stream (SISD)
    - e.g., traditional uniprocessor machines like a PC or old mainframes
  - Single Instruction stream, Multiple Data streams (SIMD)
    - e.g., an array processor, vector processing unit, or GPU
  - Multiple Instruction streams, Single Data stream (MISD)
    - Uncommon architecture; may be used for fault tolerance
    - A number of systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer
  - Multiple Instruction streams, Multiple Data streams (MIMD)
    - Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space

Flynn‘s Taxonomy of Computer Categorization (2/2):

- (Following emphasize on parallel machines)
  - SISD machines
    - systems contain one CPU and hence can accommodate one instruction stream that is executed serially (conceptually)
  - **SIMD machines**
    - Exploits data level parallelism
    - Systems often have a large number of processing units
    - A single instruction manipulates many data items in parallel
    - A subclass of SIMD systems is vector processors
  - MISD machines
    - Theoretically, in this type of machines multiple instructions act on a single data stream
  - **MIMD machines**
    - Exploits thread/program level parallelism
    - Execute several instruction streams in parallel on different data streams
    - Multiprocessors including Intel Core2Dual, etc., fall into this category

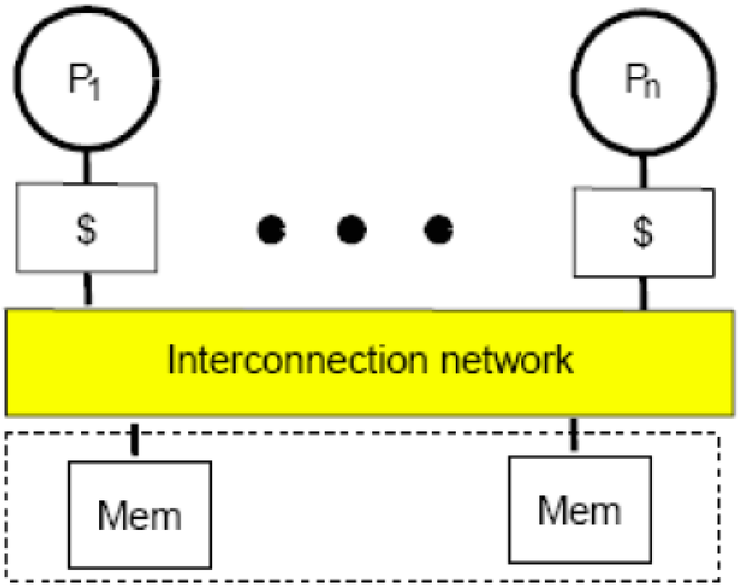
Shared Memory Architecture: SMP/DSM (1/8):

- In such architecture, memory (centralized or distributed) can be directly accessed by different CPUs in the system
- Communication between programs/threads occurs implicitly via memory instructions (e.g., loads and stores)
- A natural extension of uni-processor model
  - Shared data are location transparent



Shared Memory Architecture: SMP/DSM (2/8):
SMP: Symmetric (Shared-Memory) Multiprocessors

- Intuitive parallel processing evolution in 1980s; several processors shared a single physical memory connected by a shared bus
  - Symmetric shared memory -- each processor has the same relationship to one single shared memory
  - Scalability limited by interconnection network



Shared Memory Architecture: SMP/DSM (3/8): SMP: Symmetric (Shared-Memory) Multiprocessors

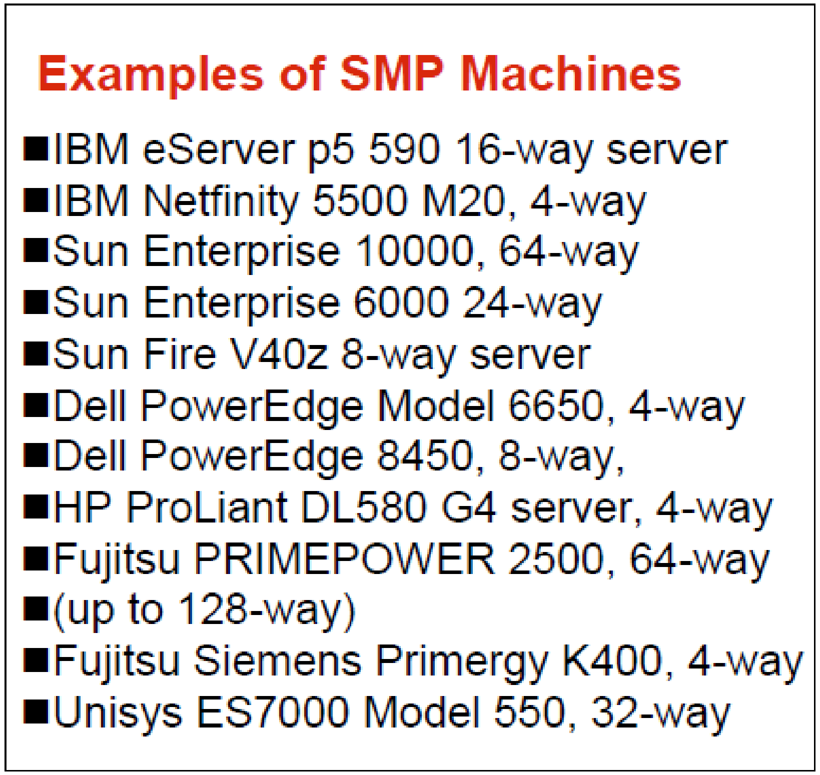
- Characteristics
  - UMA
    - Uniform memory access latency from any processor
  - Symmetric (beyond UMA)
    - Equal access to computer’s resources
    - Equal PE computing power
  - Connected by bus or crossbar switch
  - Shared-everything architecture
    - Centralized shared memory, shard I/O, same OSes
  - Cache coherence protocol is needed if caches exist
    - Usually implemented by hardware

Shared Memory Architecture: SMP/DSM (4/8): SMP: Symmetric (Shared-Memory) Multiprocessors

- Interconnection:
  - Bus
    - simple to design but cannot scale due to bus contention
  - Crossbar switch
    - Costly mechanism to avoid contention on different memory module (e.g., CPU1, CPU2, CPU3 all access different memory modules)
  - Whatever, necessary attribute is to offer equal PE-to-mem accesses
  - Difficult to scale to very large sizes:

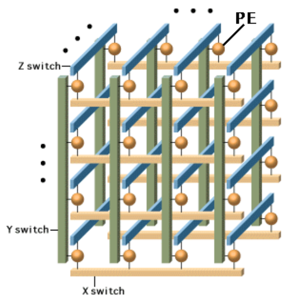
    2 to 32/128 processors today

    High cache coherence overhead when scales



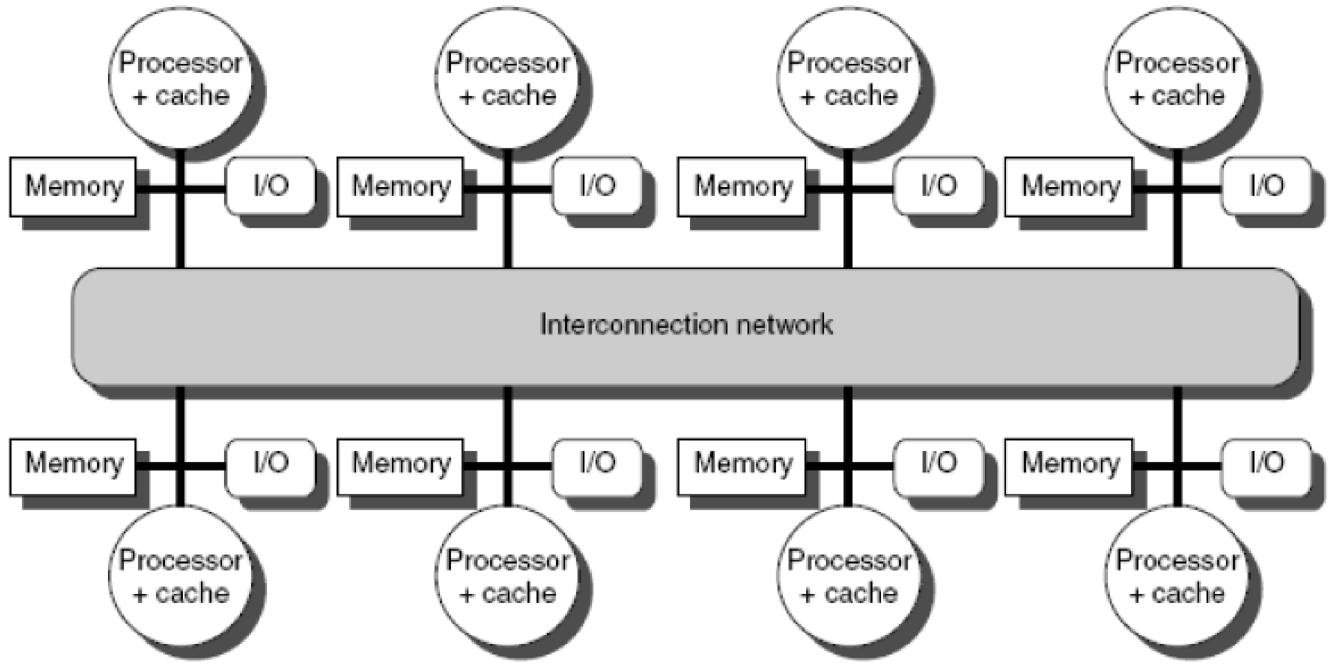
Examples of SMP Machines

- IBM eServer p5 590 16-way server
- IBM Netfinity 5500 M20, 4-way
- Sun Enterprise 10000, 64-way
- Sun Enterprise 6000 24-way
- Sun Fire V40z 8-way server
- Dell PowerEdge Model 6650, 4-way
- Dell PowerEdge 8450, 8-way,
- HP ProLiant DL580 G4 server, 4-way
- Fujitsu PRIMEPOWER 2500, 64-way
- (up to 128-way)
- Fujitsu Siemens Primergy K400, 4-way
- Unisys ES7000 Model 550, 32-way



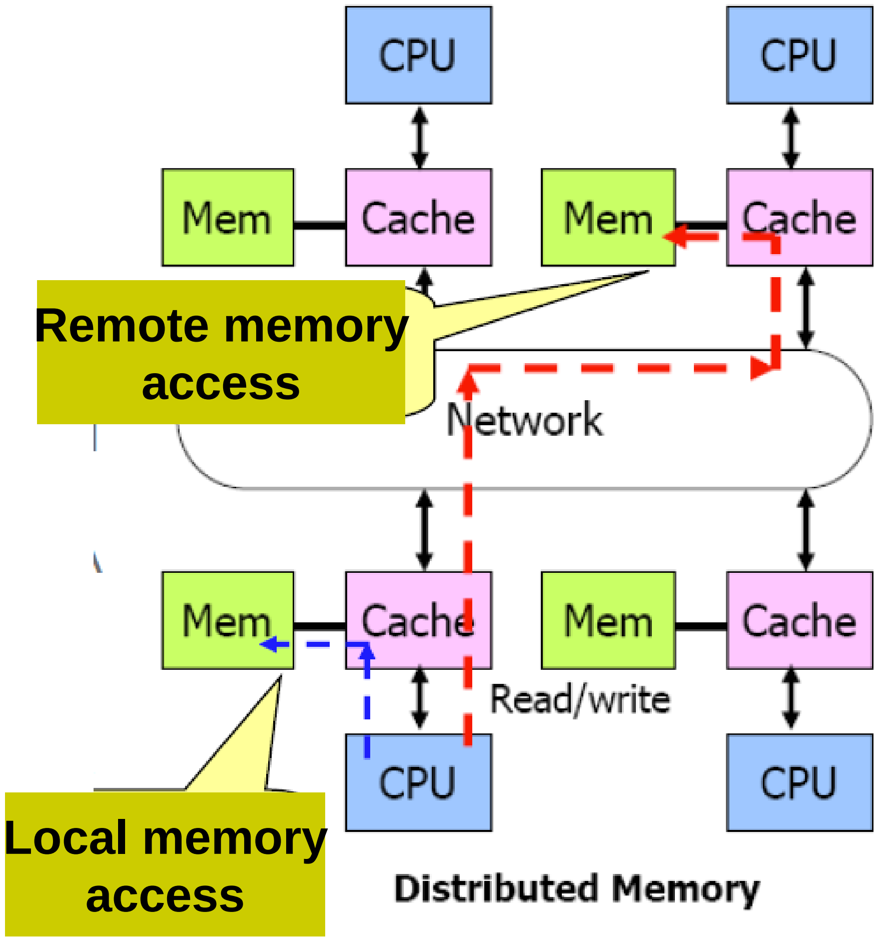
Shared Memory Architecture: SMP/DSM (5/8): DSM: Distributed Shared-Memory Multiprocessors

- DSM: Distributed Shared-Memory Multiprocessors
  - Memory is distributed among the nodes, and all node-memory pairs are interconnected by a network
  - Memory access can be either local or remote



Shared Memory Architecture: SMP/DSM (6/8): DSM: Distributed Shared-Memory Multiprocessors

- NUMA:
  - non-uniform memory accesses
    - All CPUs can read/write any part of the distributed memory, but
    - access time depends on location of a data word in memory
  - Local memory access
    - When local cache misses, and local memory hits
    - Shorter access latency than in UMA
  - Remote memory access
    - When local memory misses
    - Longer access latency than in UMA

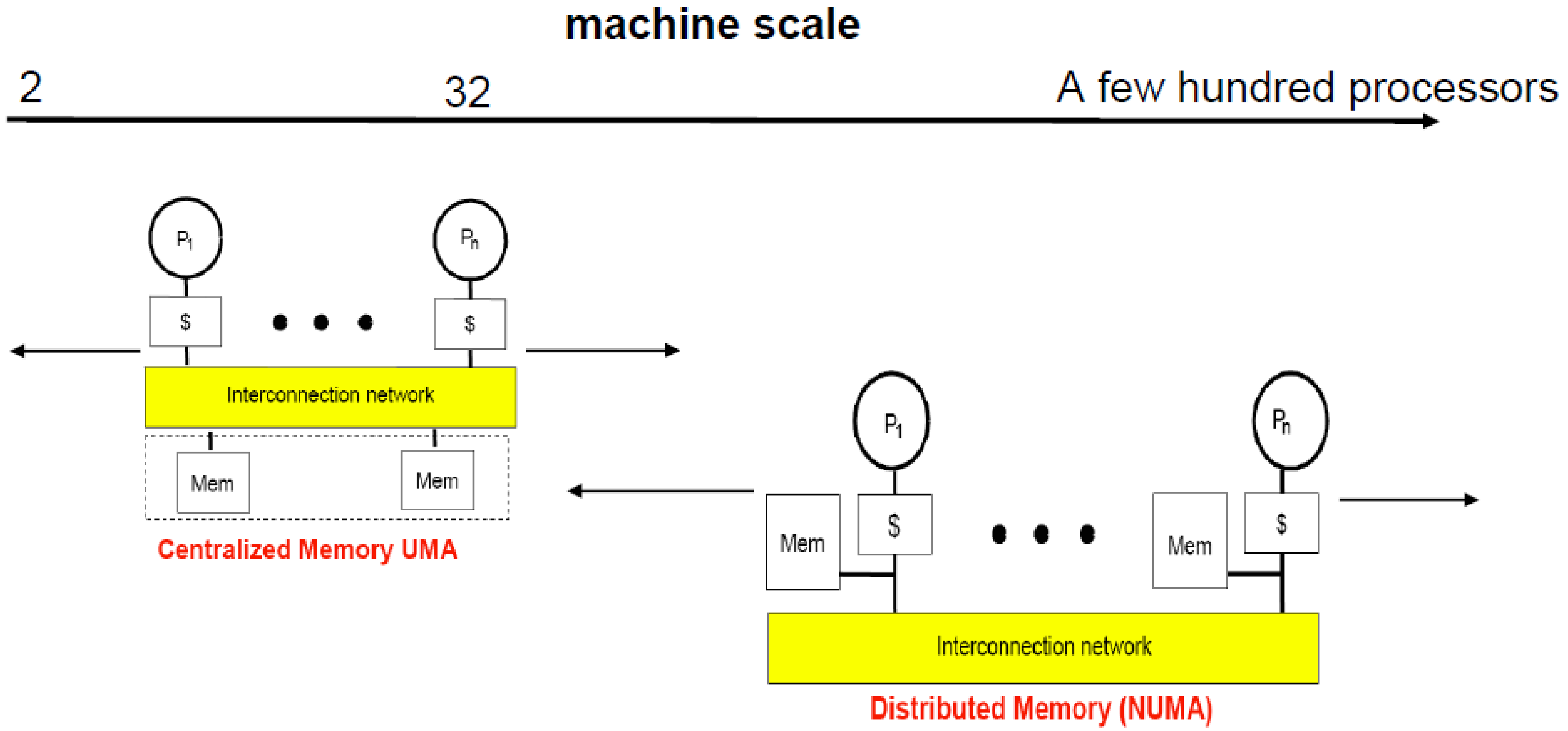


Shared Memory Architecture: SMP/DSM (7/8): DSM: Distributed Shared-Memory Multiprocessors

- Benefits:
  - A cost-effective way to reduce memory traffic on the interconnection network:
    - By making most of the accesses to local memory
  - Improves application scalability
    - By exploiting data locality
- Cautions:
  - Memory access latency depends on distance between processor and memory – software must keep data close to the processor for scalable performance
  - Communications between processors becomes more complex – must be handled carefully by the architecture

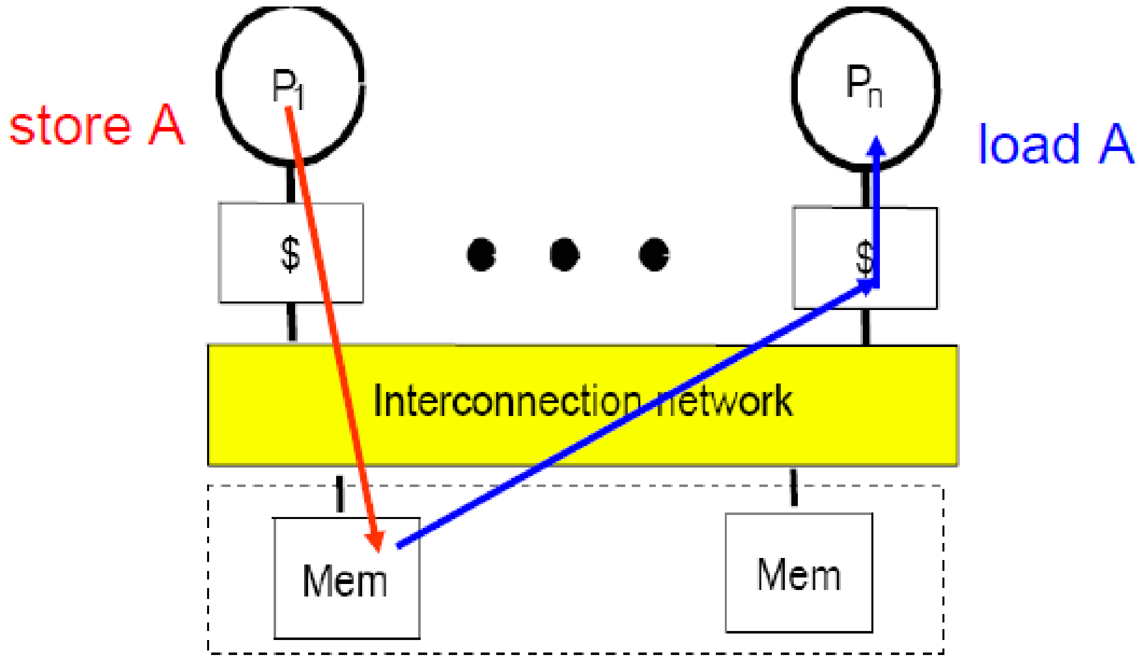
Shared Memory Architecture: SMP/DSM (8/8):

- SMP vs. DSM



Shared Memory Communication (1/2):

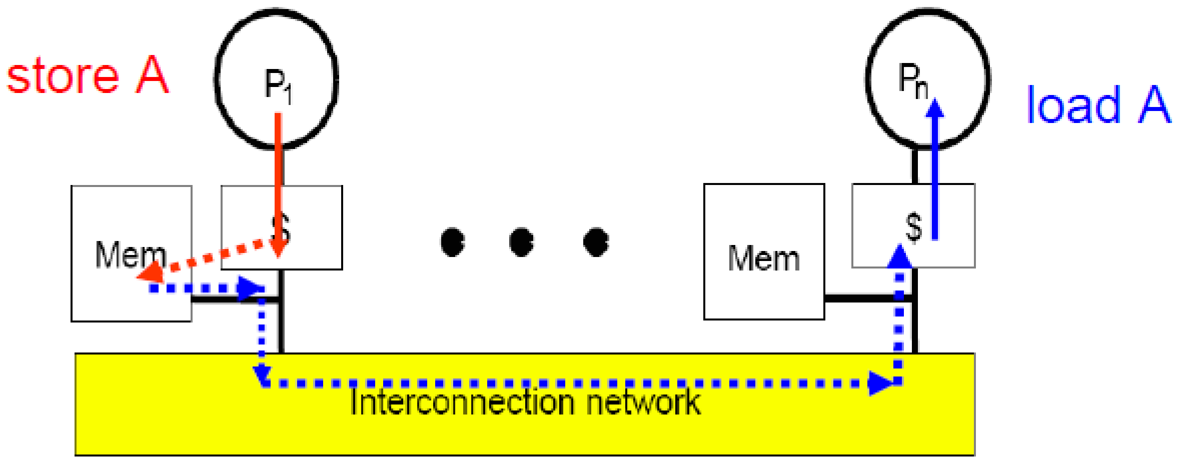
- In shared memory systems, all processors share the same address space (main memory)
  - Address locations can be used to communicate implicitly via memory instructions (e.g. load and store)
- Small-scale systems usually centralize memory; result is a UMA



Centralized Memory UMA

Shared Memory Communication (2/2):

- Large-scale multiprocessors must use multiple memories that are physically distributed among the processors; NUMA (small-scale multiprocessors? SMP is possible)
  - For communicating data among processors:
    - Physically separate memories can be addressed as one logically shared address space
  - The same physical address on two processors refers to the same location in memory (e.g., store A in P1, load A in Pn)



Distributed Memory (NUMA)

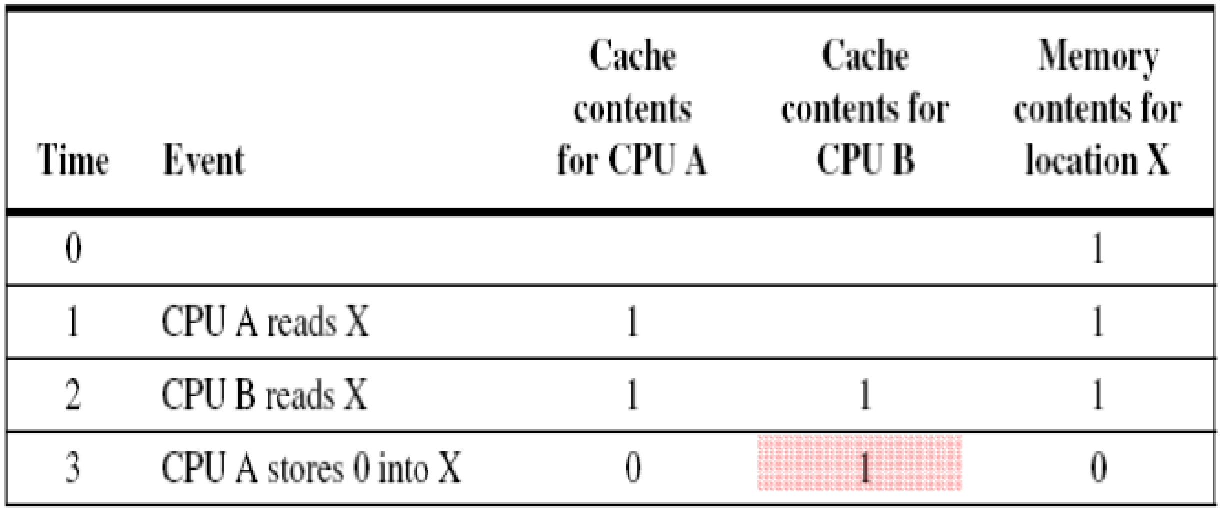
Cache Coherence in Shared Memory Architecture (1/19): Basic Schemes for Enforcing Coherence

- Programs on multiple processors may have copies of the same data in several caches
  - Most SMPs use HW schemes to maintain data coherence in the caches
  - Accessing data in memory/remote cache may take much longer time than accessing local cache
- Techniques to improve data locality
  - Migration - data can be moved to a local cache and used there in a transparent fashion
    - Reduces both
      - latency to access shared data that is allocated remotely and
      - bandwidth demand on the shared memory
  - Replication – for shared data being simultaneously read, since caches make a copy of data in local cache
    - Reduces both
      - latency of access and
      - contention for read shared data

Cache Coherence in Shared Memory Architecture (2/19):

- Coherence and consistency
  - Conherence defines **values** returned by a read
  - Consistency determines **when** a written value will be returned by a read
- Cache Coherence problem
  - Processors see different values for X at time = 3

Write-through cache



Time	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

Cache Coherence in Shared Memory Architecture (3/19):

- Defining Coherent Memory System
  - 1. **Preserve Program Order**: A read by processor P to location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P
  - 2. **Coherent view of memory**: Read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses
  - 3. **Write serialization**: 2 writes to same location by any 2 processors are seen in the same order by all processors
    - If not, a processor could keep value 1 since saw as last write
    - For example, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1

Cache Coherence in Shared Memory Architecture (4/19):

- Cache coherence protocols maintain coherence for multiple processors
- 2 Classes of Cache Coherence Protocols
  - Bus based -- **Snooping**
    - Every cache with a copy of data has a copy of sharing status of block, but no centralized state is kept
    - All caches are accessible via some broadcast medium (a bus or switch)
    - All cache controllers monitor or snoop on the bus to determine whether or not they have a copy of a block that is requested on the bus
  - **Directory based**
    - The sharing status of a block of physical memory is kept in just one location

Cache Coherence in Shared Memory Architecture (5/19):

- Example Write-Back Snoopy Protocol
  - Invalidation protocol, write-back cache
    - Snoops every address on bus
    - If it has a dirty copy of requested block, provides that block in response to the read request and aborts the memory access
  - Each memory block is in one state (but not to be indicated):
    - Clean in all caches and up-to-date in memory (Shared in cache)
    - OR Dirty in exactly one cache (Exclusive in cache)
    - OR not in any caches
  - Each cache block is in one state (track these):
    - Shared : block can be read
    - OR Exclusive : cache has only copy, it is writeable, and dirty
    - OR Invalid : block contains no data
  - Read misses: caches with that block respond
  - Writes to shared blocks are treated as misses (since cause same bus activities; simplifies design)

An Example of Invalidation Protocol

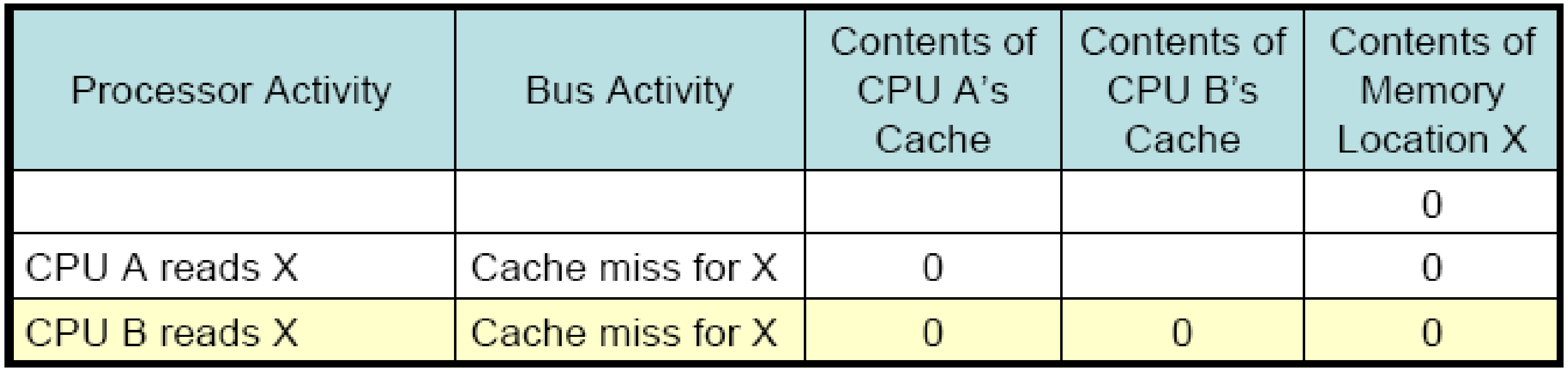
Processor Activity	Bus Activity	Contents of CPU A's Cache	Contents of CPU B's Cache	Contents of Memory Location X
				0

- **An example of an invalidation protocol working on a snooping bus for a single cache block (X) with write-back caches.**

An Example of Invalidation Protocol

Processor Activity	Bus Activity	Contents of CPU A's Cache	Contents of CPU B's Cache	Contents of Memory Location X
				0
CPU A reads X	Cache miss for X	0		0

An Example of Invalidation Protocol



Processor Activity	Bus Activity	Contents of CPU A's Cache	Contents of CPU B's Cache	Contents of Memory Location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0

An Example of Invalidation Protocol

Processor Activity	Bus Activity	Contents of CPU A's Cache	Contents of CPU B's Cache	Contents of Memory Location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0

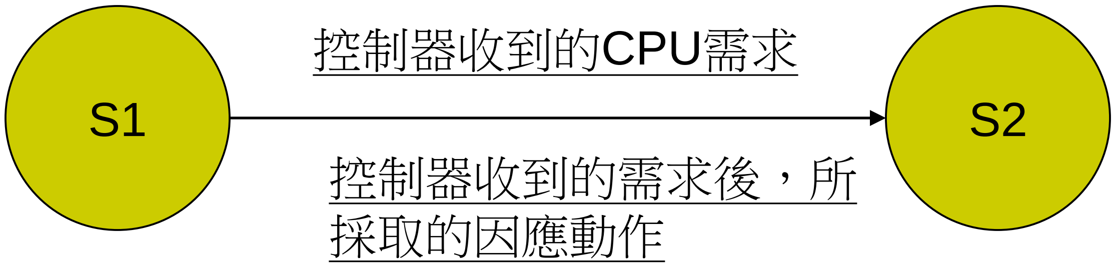
- **For a write, we require that the writing processor have exclusive access, preventing any other processor from being able to write simultaneously.**

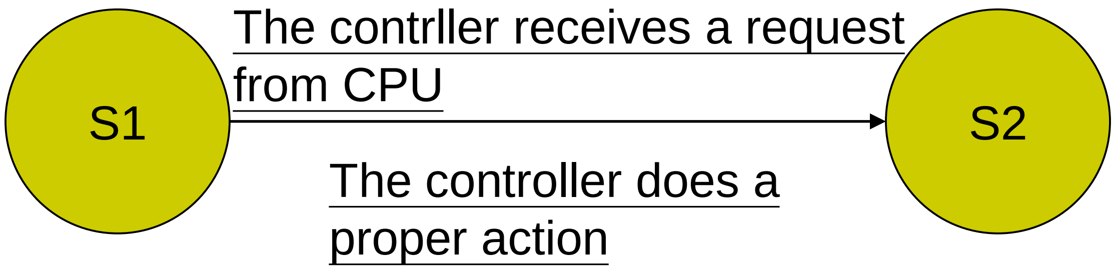
An Example of Invalidation Protocol

Processor Activity	Bus Activity	Contents of CPU A's Cache	Contents of CPU B's Cache	Contents of Memory Location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

- **When the second miss by B occurs, CPU A responds with the value and cancels the response from memory. In addition, both the contents of B's cache and the memory contents of X are updated.**

狀態機的狀態改變 (Finite State Machine: FSM)



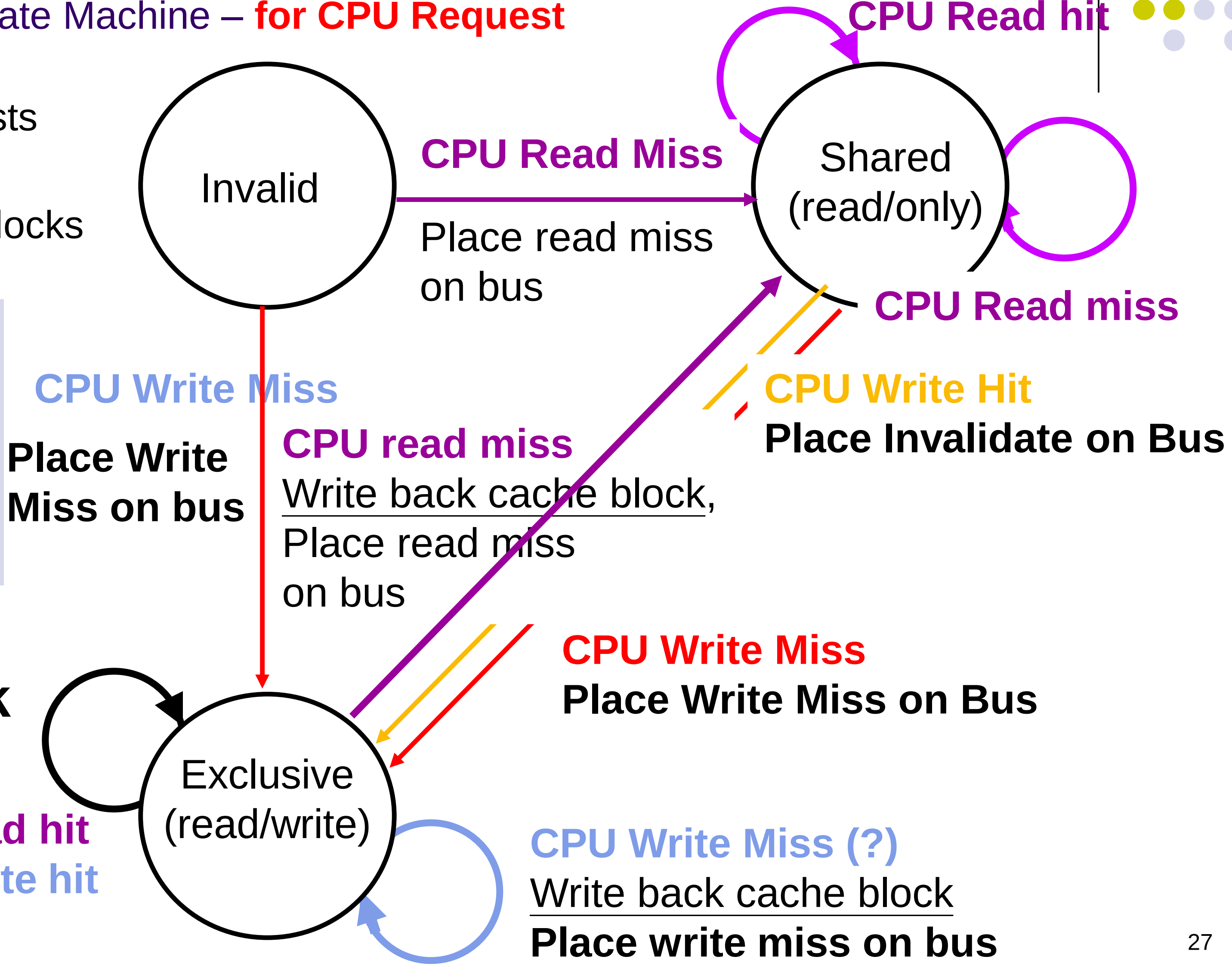


S1
S2
The contrller receives a request from CPU
The controller does a proper action

Cache Coherence in Shared Memory Architecture (6/19): Write-Back State Machine – for CPU Request

- State machine for CPU requests for each cache block
- Non-resident blocks invalid

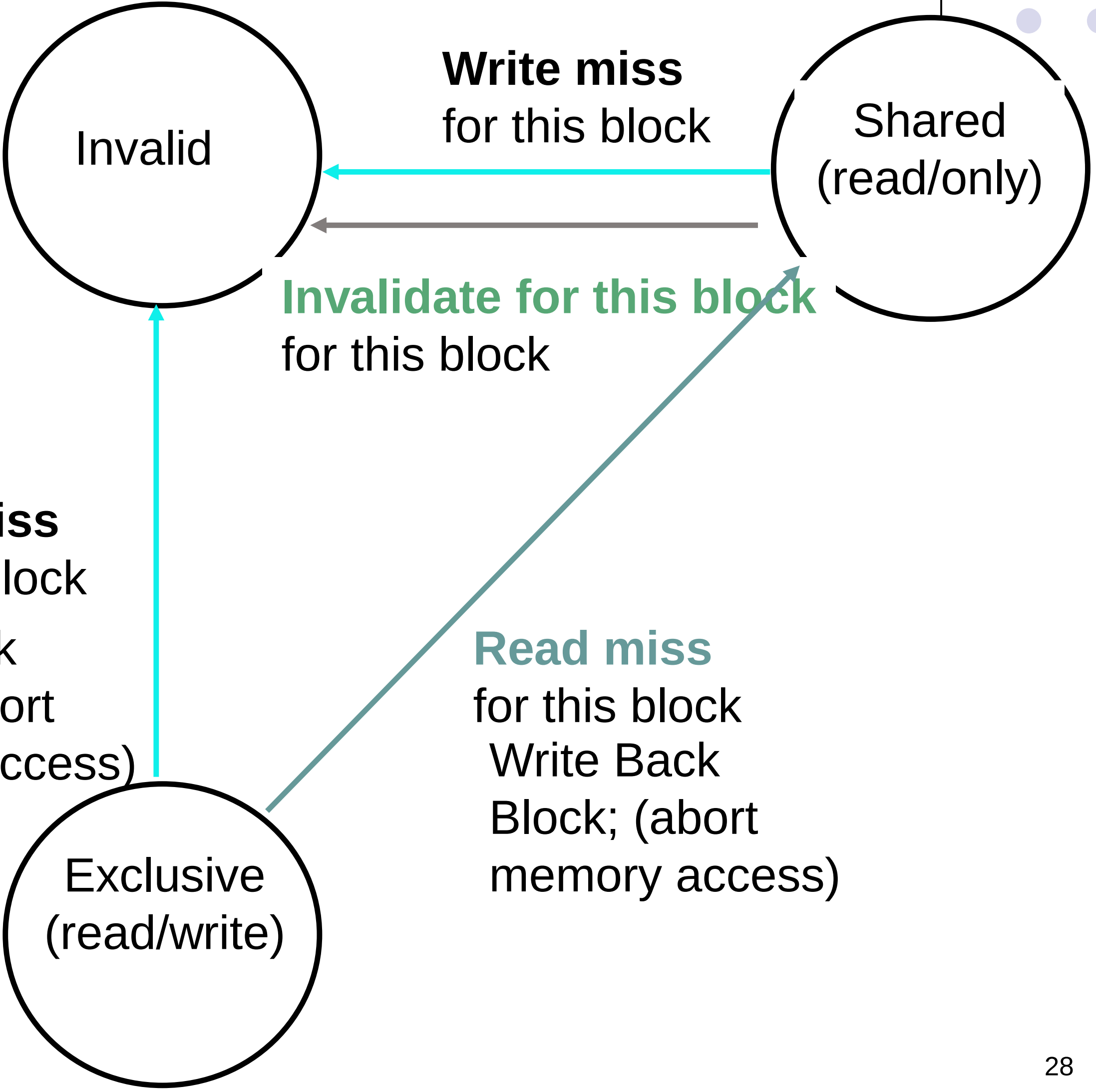
Cache controller幫每個Cache block維護一個對應的狀態機。Cache controller maintains a finite state machine for each cache block.



Cache Block State

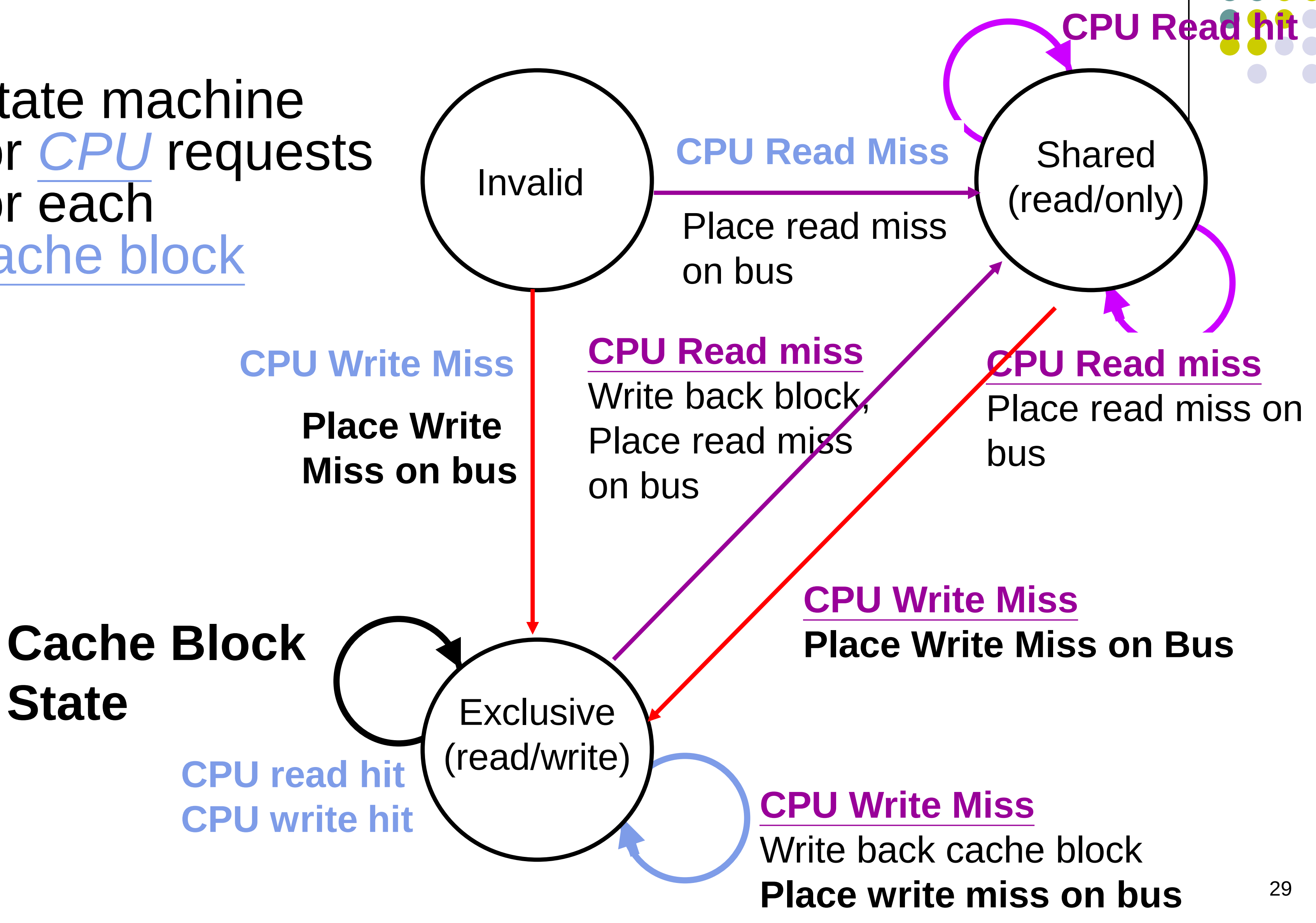
Cache Coherence in Shared Memory Architecture (7/19): Write-Back State Machine – for Bus Requests

- State machine for *bus* requests for each cache block


Invalid
Shared (read/only)
Exclusive (read/write)
Write miss for this block
Invalidate for this block for this block
Write miss for this block
Write Back Block; (abort memory access)
Read miss for this block
Write Back Block; (abort memory access)

Cache Coherence in Shared Memory Architecture (8/19): Block-replacement

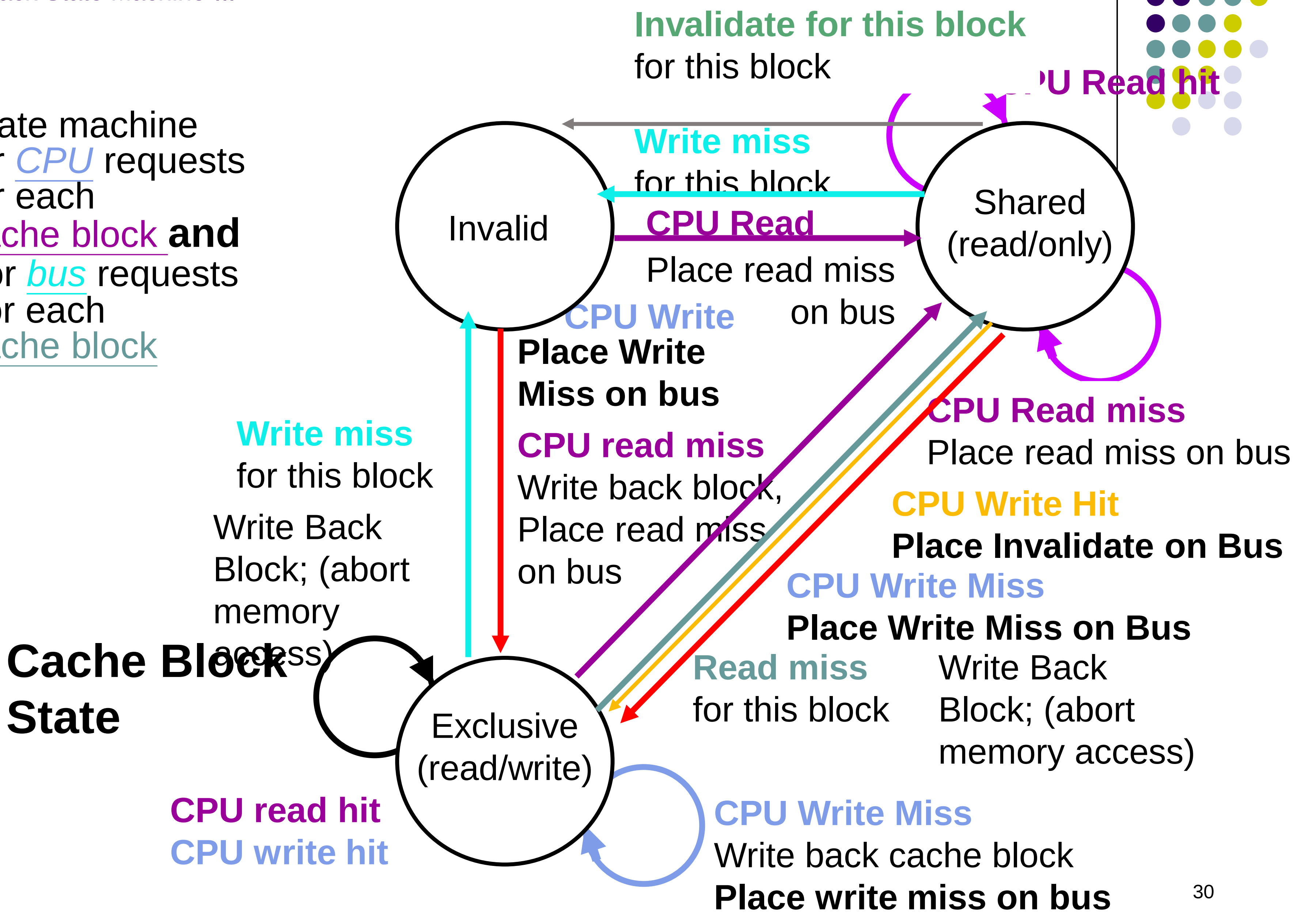
- State machine for *CPU* requests for each cache block



Cache Coherence in Shared Memory Architecture (9/19):

Write-back State Machine-III

- State machine for *CPU* requests for each cache block **and** for *bus* requests for each cache block



Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1												
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block,
initial cache state is invalid

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Valu	Actior	Proc.	Addr	Value	Add.	Valu
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	*State*	*Addr*	*Value*	*State*	*Add*	*Valu*	*Actic*	*Pro*	*Addr*	*Valu*	*Addr*	*Value*
P1 Write 10 to A1	*Excl.*	*A1*	*10*				*WrMs*	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				*Shar.*	*A1*		*RdMs*	P2	A1			
	Shar.	A1	10	Shar.	A1	*10*	*WrBk*	P1	A1	10	A1	*10*
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Valu	Actior	Proc.	Addr	Value	Add	Valu
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10	Shar.	A1	10	WrBk	P1	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	Inv	P2	A1		A1	10
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Valu	Actio	Proc.	Addr	Value	Add	Valu
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10	Shar.	A1	10	WrBk	P1	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	Inv	P2	A1		A1	10
P2: Write 40 to A2							WrBk	P2	A1	20	A1	20
				Excl.	A2	40	WrMs	P2	A2		A1	20

Assumes A1 and A2 map to same cache block,
but A1 != A2

Cache Coherence in Shared Memory Architecture (10/19): Coherency Misses

為了確保coherence之下，導致原已讀取的block被丟棄，重新再讀取。

- 4th C: *coherence miss*
  - Joins Compulsory, Capacity, Conflict
    - 1. True sharing misses arise from the communication of data through the cache coherence mechanism
      - The first write by a processor to a shared cache block causes an invalidation to establish ownership of that block.
      - When another processor attempts to read a modified word in that cache block, a miss occurs and the resultant block is transferred.
      - Miss would still occur if block size were 1 word
    - 2. False sharing misses when a block is invalidated because some words in the block, other than the one being read, is written into
      - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
      - Block is shared, but no word in block is actually shared
        ⇒ <u>miss would not occur if block size were 1 word</u>

Cache Coherence in Shared Memory Architecture (11/19): Coherency Misses

- Example: True vs. False Sharing vs. Hit?
  - Assume x1 and x2 in same cache block
    - P1 and P2 both read x1 and x2 before

Time	P1	P2	True, False, Hit? Why?
1	Write x1		Write hit; place invalidate on bus; invalidate x1 in P2
2		Read x2	**False miss; x1 irrelevant to P2**
3	Write x1		Write hit; place invalidate on bus; invalidate x1 in P2
4		Write x2	**False miss; invalidate x2 in P1**
5	Read x2		**True miss**

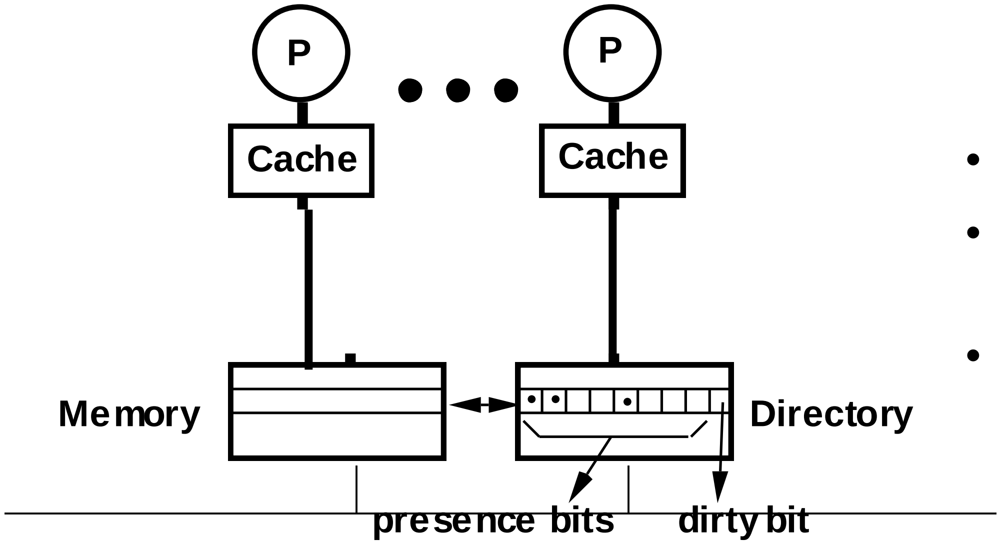
Cache Coherence in Shared Memory Architecture (12/19): Directory Protocol

- For larger-scale architectures, memory bandwidth also need to be scaled
  - Bus-based coherence lacks scalability
  - Scalable Approach: Directories
    - A directory keeps the state of every memory block that may be cached.
      - Information in the directory includes which caches have copies of the memory block, whether it is dirty, and so on.

Cache Coherence in Shared Memory Architecture (13/19): Directory Protocol

- **Similar to Snoopy Protocol: Three states for each memory block**
  - Shared**: One or more processors have the block cached, and the value in memory is up to date.**
  - Uncached**: No processor has a copy of the memory block.**
  - Exclusive**: Exactly one processor (owner) has a copy of the memory block, and it has written the block, so the memory copy is out of date.**
- **In addition to cache state, we must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)**

Cache Coherence in Shared Memory Architecture (14/19): Directory Protocol



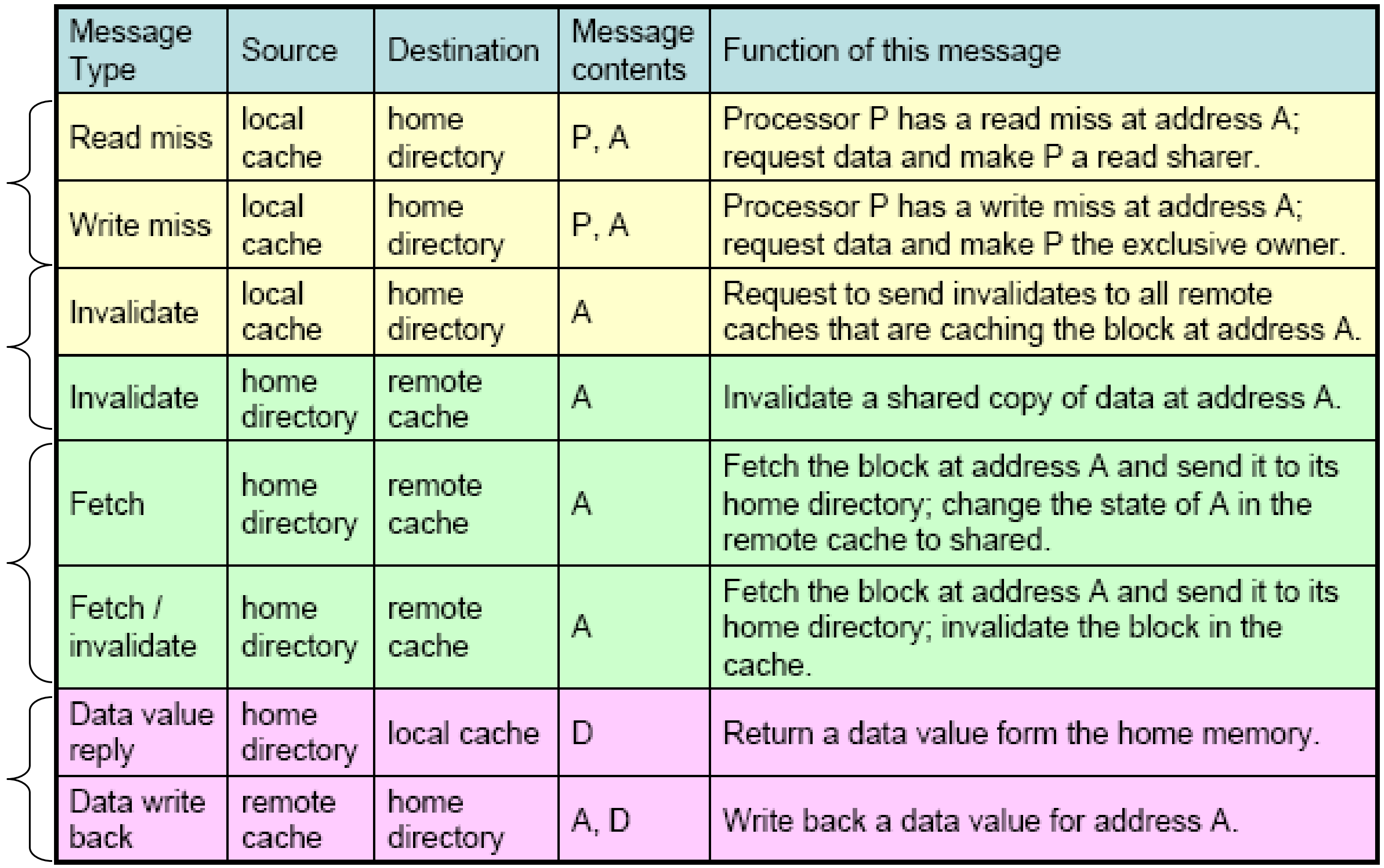
- k processors.
- With each block in memory:
  k presence-bits, 1 dirty-bit
- With each block in cache:
  1 valid bit, and 1 dirty (owner) bit

- Read from main memory by processor i:
  - If dirty-bit OFF then { read from main memory; turn p[i] ON; }
  - if dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}
- Write to main memory by processor i:
  - If dirty-bit OFF then { supply data to i; send invalidations to all caches that have the block; turn dirty-bit ON; turn p[i] ON; ... }
  - ...

Cache Coherence in Shared Memory Architecture (15/19): Directory Protocol

- **Terms: Typically 3 processors involved**
  - Local node **where a read/write** **request** **originates**
  - Home node **where the** **memory location** **and the** **directory** **entry of an address reside**
  - Remote node **has a** **copy** **of a cache block, whether exclusive or shared**
- **Physical address space**
  - **The high-order bits may provide the node number**
  - **The low-order bits provide the offset within the memory on that node**

Cache Coherence in Shared Memory Architecture (16/19): Directory Protocol



Message Type	Source	Destination	Message contents	Function of this message
Read miss	local cache	home directory	P, A	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	local cache	home directory	P, A	Processor P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	local cache	home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	home directory	remote cache	A	Invalidate a shared copy of data at address A.
Fetch	home directory	remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch / invalidate	home directory	remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	home directory	local cache	D	Return a data value form the home memory.
Data write back	remote cache	home directory	A, D	Write back a data value for address A.

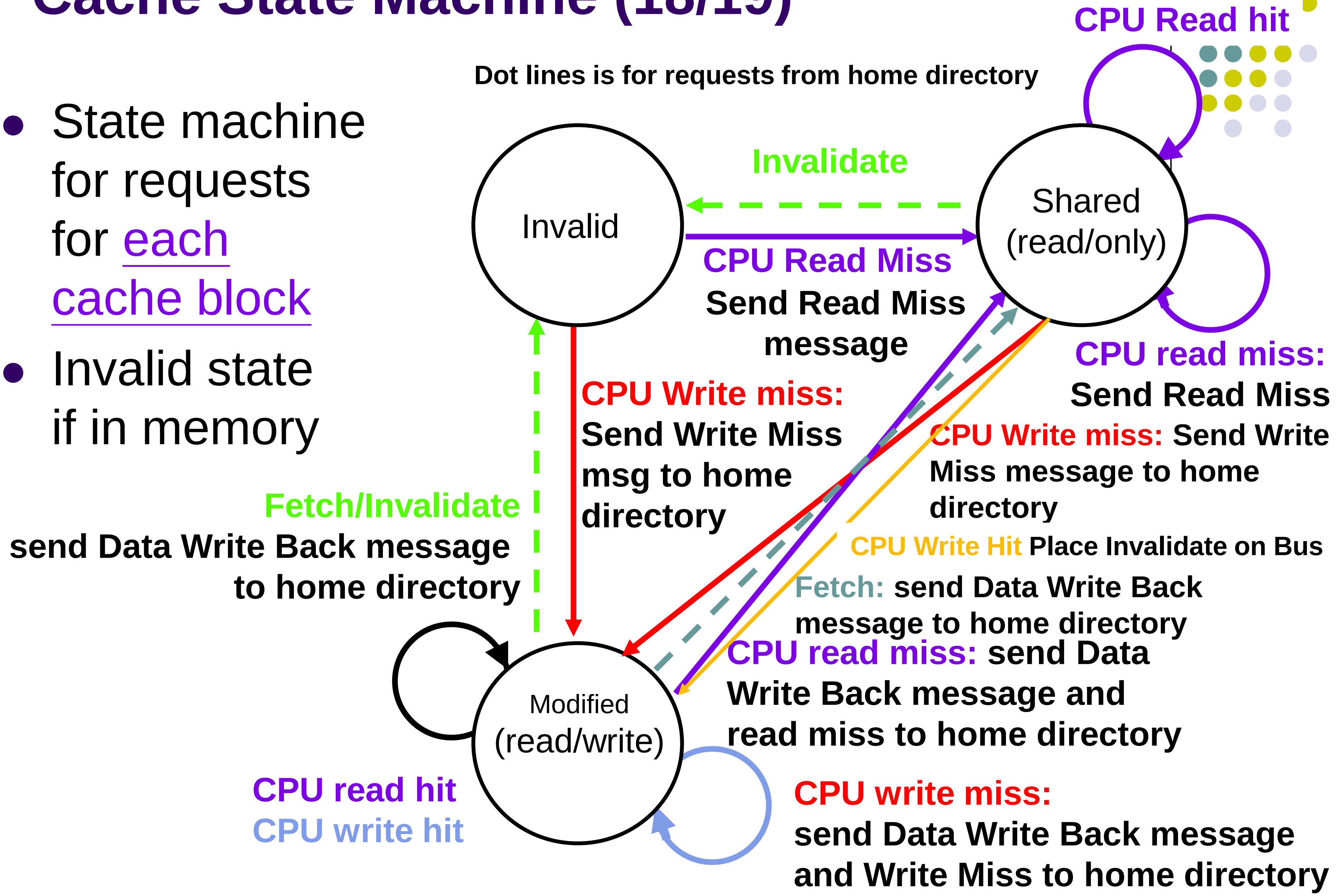
P = requesting processor number
A = requested address
D = data contents

Cache Coherence in Shared Memory Architecture (17/19): Directory Protocol

- **Cache states identical to snoopy case; transactions very similar**
- **State transitions for a cache caused by read misses, write misses, invalidates, data fetch requests**
- **An individual cache generates read miss, write miss, and invalidate message to home directory**
- **Write misses that were broadcast on the bus for snooping ⇒ explicit invalidate & data fetch requests**

Cache State Machine (18/19)

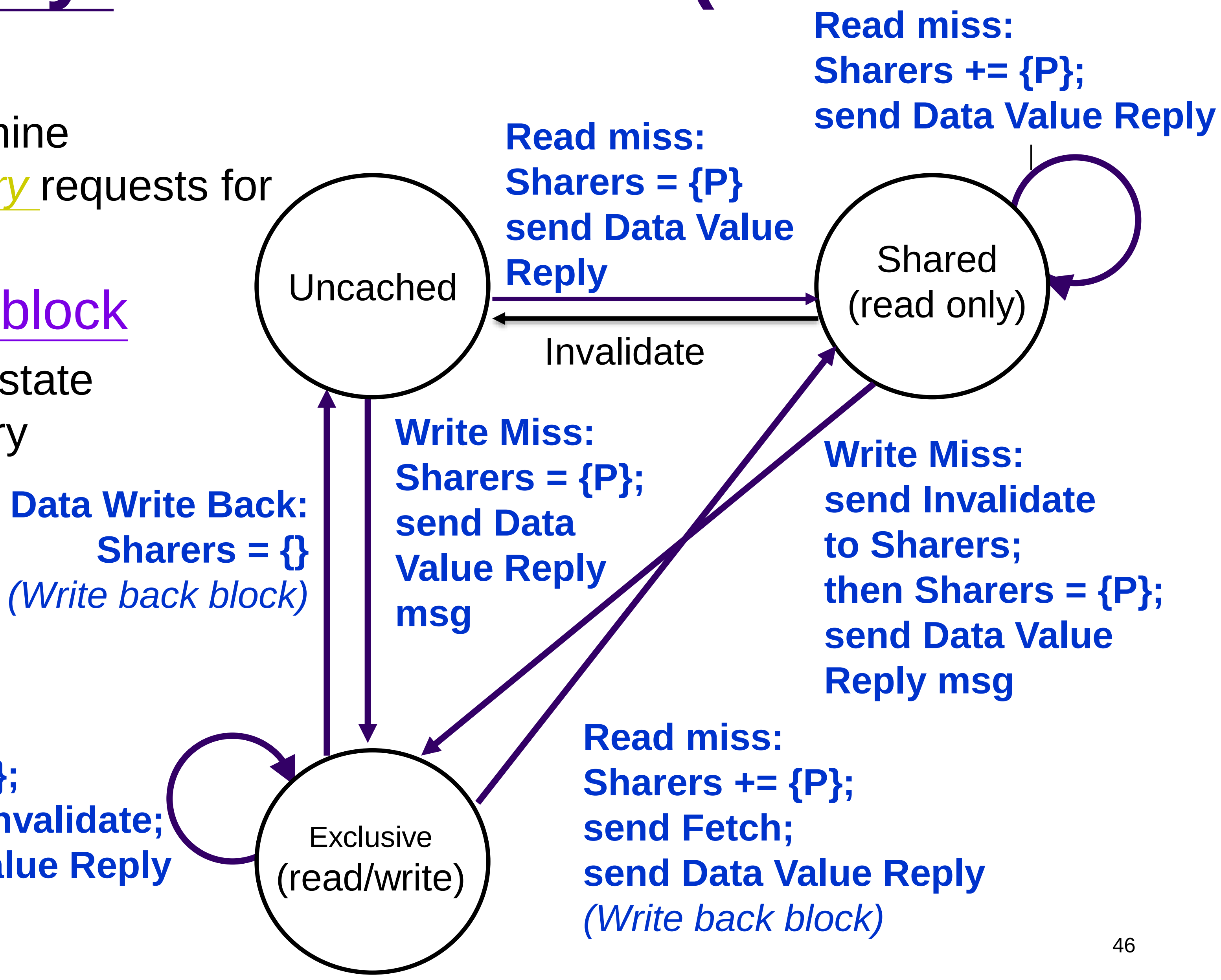
- State machine for requests for each cache block
- Invalid state if in memory



CPU Read hit
Dot lines is for requests from home directory
Invalidate
Invalid
Shared (read/only)
CPU Read Miss
Send Read Miss message
CPU read miss:
Send Read Miss
CPU Write miss:
Send Write Miss msg to home directory
CPU Write miss: Send Write Miss message to home directory
Fetch/Invalidate
send Data Write Back message to home directory
CPU Write Hit Place Invalidate on Bus
Fetch: send Data Write Back message to home directory
CPU read miss: send Data Write Back message and read miss to home directory
Modified (read/write)
CPU read hit
CPU write hit
CPU write miss:
send Data Write Back message and Write Miss to home directory

Directory State Machine (19/19)

- State machine for *Directory* requests for each memory block
- Uncached state if in memory


Uncached
Shared (read only)
Exclusive (read/write)
Read miss: Sharers = {P} send Data Value Reply
Invalidate
Read miss: Sharers += {P}; send Data Value Reply
Write Miss: Sharers = {P}; send Data Value Reply msg
Data Write Back: Sharers = {} (Write back block)
Write Miss: send Invalidate to Sharers; then Sharers = {P}; send Data Value Reply msg
Write Miss: Sharers = {P}; send Fetch/Invalidate; send Data Value Reply
Read miss: Sharers += {P}; send Fetch; send Data Value Reply (Write back block)

State Transition Diagram for Directory

- Same states & structure as the transition diagram for an individual cache
- A message sent to a directory causes two different types of actions:
  - update the directory state
  - send messages to satisfy the request
- Unlike snoopy, the directory state indicates the state of all the cached copies of a memory block, rather than for a single cache block.
- The directory must track the set of processors that have a copy of a block; we use a set called Sharers to perform this function.

Example

Processor 1 Processor 2 Interconnect Directory Memory

	P1			*P2*			*Bus*				*Directory*			*Memory*
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Action*	*Proc.*	*Addr*	*Value*	*Addr*	*State*	*{Procs}*	*Value*
P1: Write 10 to A1														
P1: Read A1														
P2: Read A1														
P2: Write 20 to A1														
P2: Write 40 to A2														

A1 and A2 map to the same cache block

Example

Step4

Step3

Step1

Step2

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			*P2*			*Bus*				*Directory*			*Memor*
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Action*	*Proc.*	*Addr*	*Value*	*Addr*	*State*	*{Procs}*	*Value*
P1: Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1														
P2: Read A1														
P2: Write 20 to A1														
P2: Write 40 to A2														

A1 and A2 map to the same cache block

Example

	Processor 1			Processor 2			Interconnect				Directory			Memory
	P1			*P2*			*Bus*				*Directory*			*Memory*
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Action*	*Proc.*	*Addr*	*Value*	*Addr*	*State*	*{Procs}*	*Value*
P1 Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1	Excl.	A1	10								*A1*	*Ex*	*{P1}*	
P2: Read A1														
P2: Write 20 to A1														
P2: Write 20 to														
P2: Write 40 to														

A1 and A2 map to the same cache block

Example

Step3 Step5 Step1 Step2 Step4

Processor 1 Processor 2 Interconnect Directory Memory

	P1			P2			Bus				Directory			Memory
step	*State*	*Addr*	*Value*	*State*	*Addr*	*Value*	*Action*	*Proc.*	*Addr*	*Value*	*Addr*	*State*	*{Procs}*	*Value*
P1 Write 10 to A1							*WrMs*	P1	A1		*A1*	*Ex*	*{P1}*	
	Excl.	*A1*	*10*				*DaRp*	P1	A1	0				
P1: Read A1	Excl.	A1	10								*A1*	*Ex*	*{P1}*	
P2: Read A1							*RdMs*	P2	A1					
	Shar.	A1	10				*Ftch*	P1	A1					*A1=10*
				Shar.	A1	*10*	*DaRp*	P2	A1	10	A1	*Shar.*	*{P1,P2}*	
P2: Write 20 to A1														
P2: Write 40 to A2														

A1 and A2 map to the same cache block

Example

Step2 Step3 Step1 Step2

Processor 1 Processor 2 Interconnect Directory Memory

	P1			P2			Bus				Directory			Memory
step	*State*	*Addr*	*Valu*	*State*	*Add*	*Valu*	*Action*	*Proc*	*Add.*	*Value*	*Addr*	*State*	*{Procs}*	*Value*
P1 Write 10 to A1							WrMs	P1	A1		A1	Ex	{P1}	
	Excl.	A1	10				DaRp	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1							RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10				A1=10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	{P1,P2}	
P2: Write 20 to A1							Inval.	P1	A1					
	Inv.			Excl.	A1	20					A1	Excl.	{P2}	
P2: Write 40 to A2														

A1 and A2 map to the same cache block

Example

Step1 Step2 Step3

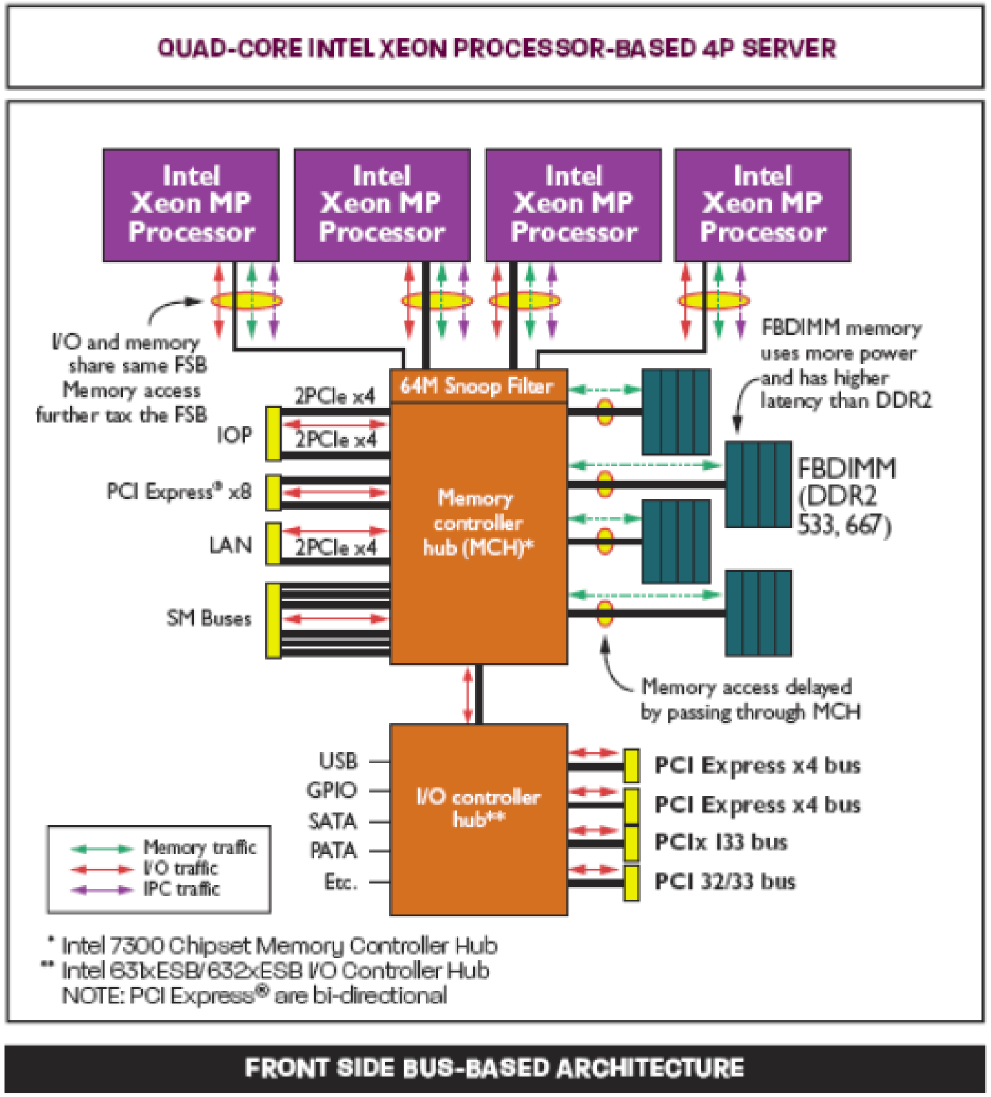
Processor 1 Processor 2 Interconnect Directory Memory

	P1			P2			Bus				Directory			Memory
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	State	{Procs}	Value
P1 Write 10 to A1							WrMs	P1	A1		A1	Ex	{P1}	
	Excl.	A1	10				DaRp	P1	A1	0				
P1: Read A1	Excl.	A1	10											
P2: Read A1							RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10				A1=10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	{P1,P2}	
P2: Write 20 to A1				Excl.	A1	20								
	Inv.						Inval.	P1	A1		A1	Excl.	{P2}	
P2: Write 40 to A2							WrMs	P2	A2		A2	Excl.	{P2}	
							WrBk	P2	A1	20	A1	Unca.	{}	A1=20
				Excl.	A2	40	DaRp	P2	A2	0				

A1 and A2 map to the same cache block
(but different memory block addresses A1 ≠ A2)

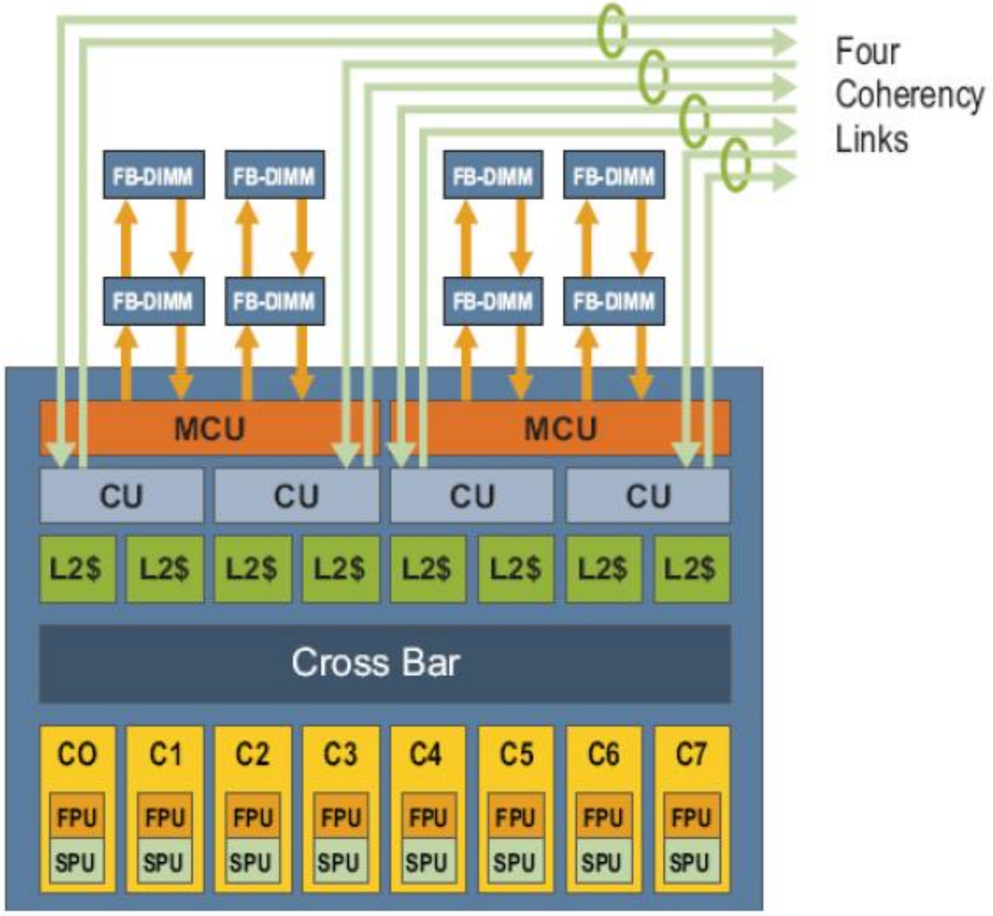
Cache Coherence in Shared Memory Architecture

- Intel QuadCore SMP
  - Snoopy protocol
  - Snoop Filter cache


QUAD-CORE INTEL XEON PROCESSOR-BASED 4P SERVER
Intel Xeon MP Processor
Intel Xeon MP Processor
Intel Xeon MP Processor
Intel Xeon MP Processor
I/O and memory share same FSB
Memory access further tax the FSB
FBDIMM memory uses more power and has higher latency than DDR2
64M Snoop Filter
2PCIe x4
IOP
2PCIe x4
PCI Express® x8
LAN
2PCIe x4
SM Buses
Memory controller hub (MCH)*
FBDIMM (DDR2 533, 667)
Memory access delayed by passing through MCH
USB
GPIO
SATA
PATA
Etc.
I/O controller hub**
PCI Express x4 bus
PCI Express x4 bus
PCIx 133 bus
PCI 32/33 bus
Memory traffic
I/O traffic
IPC traffic
* Intel 7300 Chipset Memory Controller Hub
** Intel 631xESB/632xESB I/O Controller Hub
NOTE: PCI Express® are bi-directional
FRONT SIDE BUS-BASED ARCHITECTURE

Cache Coherence in Shared Memory Architecture

- Example:
  - UltraSPARC T2 Plus
    - NUMA architecture
    - 4 Coherency Units (CUs).
    - cache coherence (snoopy)

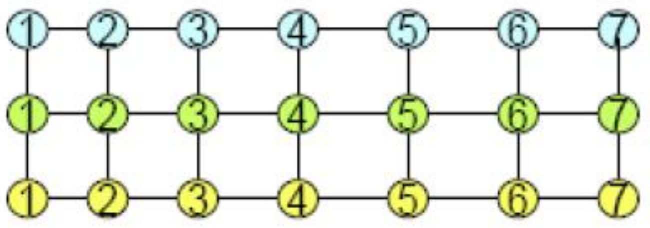


Distributed Memory Architecture (1/2):

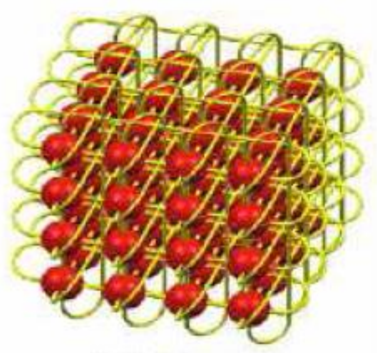
- Each CPU has its own memory view and associated memory
  - CPUs exchange data between their respective memory spaces when required, via message passing
  - Hence called **NORMA**: NO Remote-Memory Access
- User must be aware of sender/receiver of communicated data, and will have to move or distribute these data explicitly when needed
- Distributed-memory MIMD systems exhibit a large variety in their connecting topology (due to better flexibility)

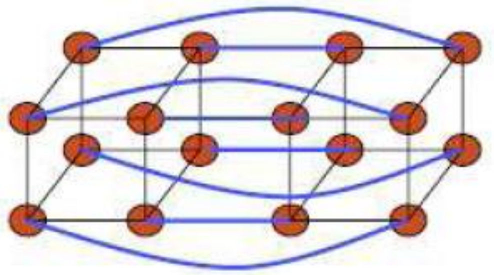
Distributed Memory Architecture (2/2):

- NORMA machine examples:
  - MPP (massively parallel processors), cluster, …
- Popular topologies:
  - (Fat) tree, mesh/torus, cube, …



2D Mesh


3D Tours

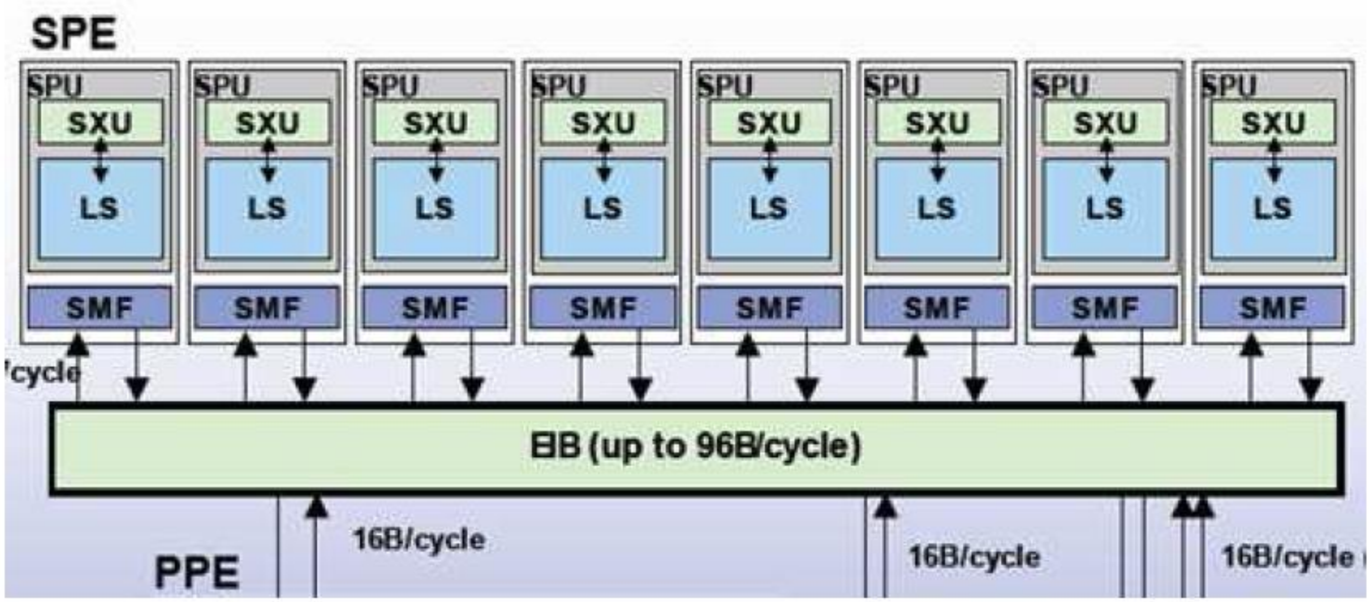

Hypercube (d=4)

Distributed Memory Communication: Message-Passing (1/2):

- Message passing multiprocessors
  - Multiprocessors are with multiple address spaces
    - communication of data is done by explicitly passing messages among the processors
- Message passing communication
  - Communications occurs by sending/receiving messages
    - request and deliver data just as with the network protocols
  - Can be *synchronous* or *asynchronous*
    - **Synchronous**: the initiating processor sends a request and waits until the reply is returned before continuing
    - **Asynchronous**: the initiating processor sends a request, yet does not wait for the reply; it immediately continues execution

Distributed Memory Communication: Message-Passing (2/2):

- Example:
  - IBM Cell SPEs
    - each SPE is an independent processor which can run its own programs
    - A shared, coherent memory and a rich set of DMA commands provide for communications between all Cell processing elements



Shared Memory vs. Distributed Memory (1/3): Advantages of Shared Memory

- Compatibility
  - Same communication mechanism works for all machines. Can have standardized programming interface (as OpenMP has proposed)
- Ease of programming
  - Obvious when communication patterns are complex or vary dynamically during execution. Similar advantage simplifies compiler design
  - Programmer allowed to focus attention only on performance-critical accesses
- Lower overhead for communication and better use of bandwidth when communicating small items
  - Arises from the implicit nature of communication and use of memory mapping to implement protection in hardware, rather than through I/O system
- Ability to use hardware-controlled caching to reduce remote communications
  - By supporting automatic caching of all data, both shared and private.

Shared Memory vs. Distributed Memory (2/3): Advantages of Distributed Memory

- Hardware can be simpler
  - in comparison with a shared memory implementation that supports coherent caching of remote data
- Communication is explicitly written in programs via sending messages
- Explicit communication captures programmer attention on this costly aspect of parallel computation
  - sometimes leading to improved structure in a multiprocessor program.
- Synchronization is naturally associated with sending/receiving messages
  - reducing possibility for errors introduced by incorrect synchronization
- Easier to use sender-initiated communication
  - may have some performance advantage

Shared Memory vs. Distributed Memory (3/3):

- Shared memory architecture is more power than distributed memory architecture
  - Cache coherence hardware moves data automatically
  - Can run both shared-memory and message-passing programs
- Distributed memory architecture is simpler and may take considerably less execution time than shared memory architecture
  - Software is responsible for moving data
  - Runs only message-passing programs
- In general,
  - Parallelizing programs for shared-memory is easier and takes less time, but it takes further efforts to both debug and scale the performance
  - Parallelizing programs for distributed-memory is harder and takes longer time, but programs are better behaved, and higher performance can be gained

Summary

- Shared memory
- Cache coherence problem
- Snooping protocol
- Directory protocol
- Distributed memory