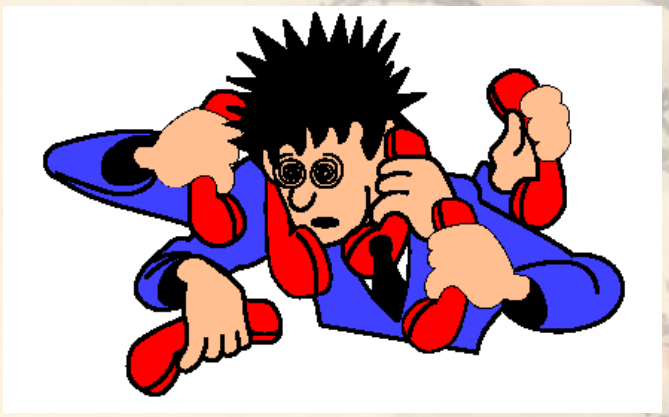
Chapter four

Multiprocessors and Thread-Level Parallelism

（续1）



陈文智

4.2.2 Basic schemes for Enforcing Coherence

一、多处理器cache的作用

1. 作用

- Caching，即在相关处理器的Caches中保留共享数据的copy。

- 因此多处理器的Cache应提供两种功能：
  - 共享数据的迁移（migration）：通过move shared data to local cache, 并且use the shared data in local cache，来达到降低访问共享数据的latency。
  - 共享数据的复制（replication）：指一旦某数据在某一Cache被改写后，应及时将改写值复制到其它Cache中去。保证多个处理器可同时读出共享数据。达到降低latency(远程调用）和减少对共享数据的竞争。

2. 实现思路

- 在小规模多处理器系统中，采用硬件解决方法，而不是软件方法；
- 引进维护Cache coherence的协议，称为cache-coherence protocol。
- 实现cache-coherence protocol的关键是：跟踪共享数据块的状态。
- 存在两类protocols,对应两种不同的跟踪共享状态的技术：基于目录的技术和监听技术。

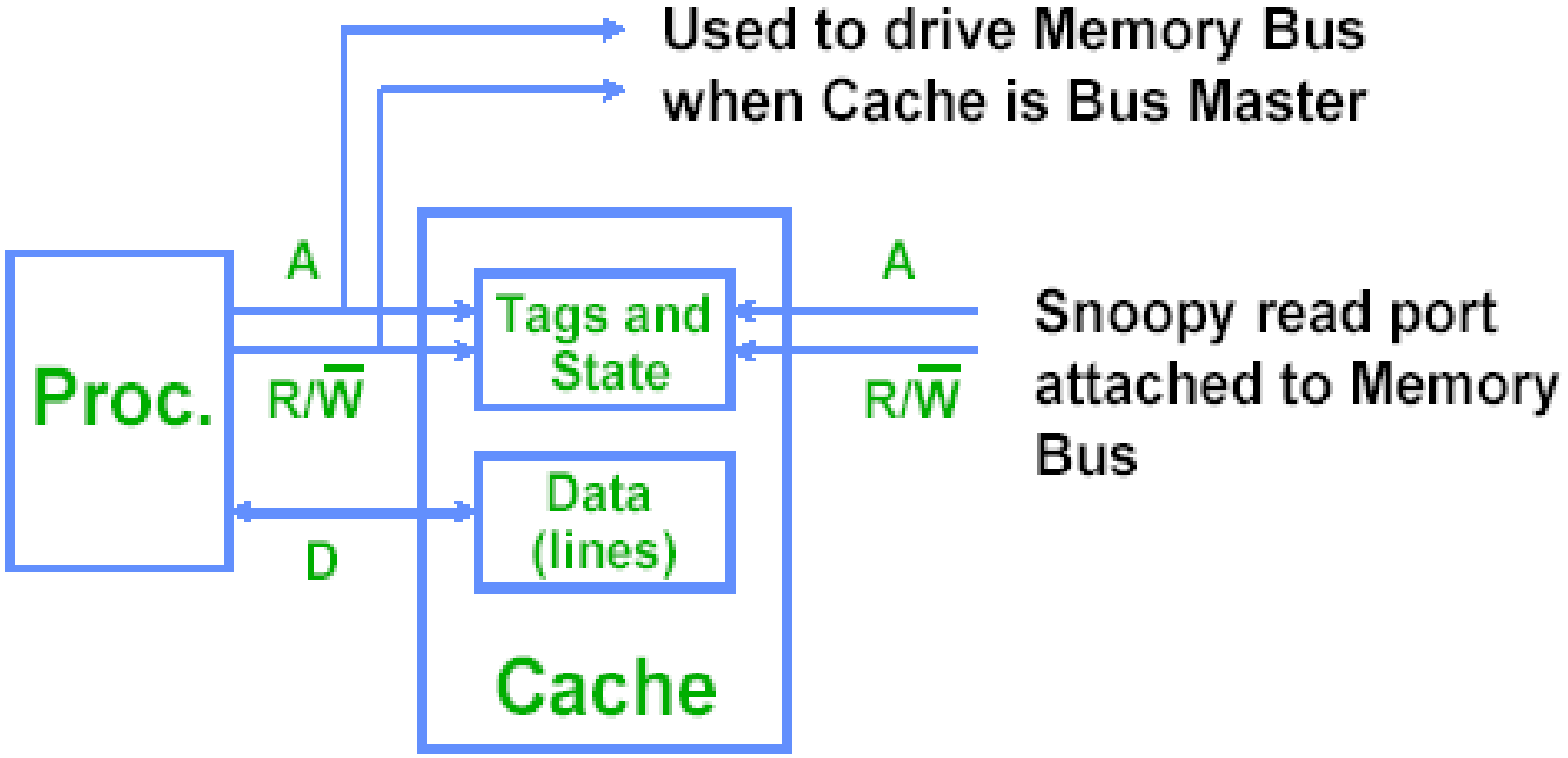
二、两类一致性协议

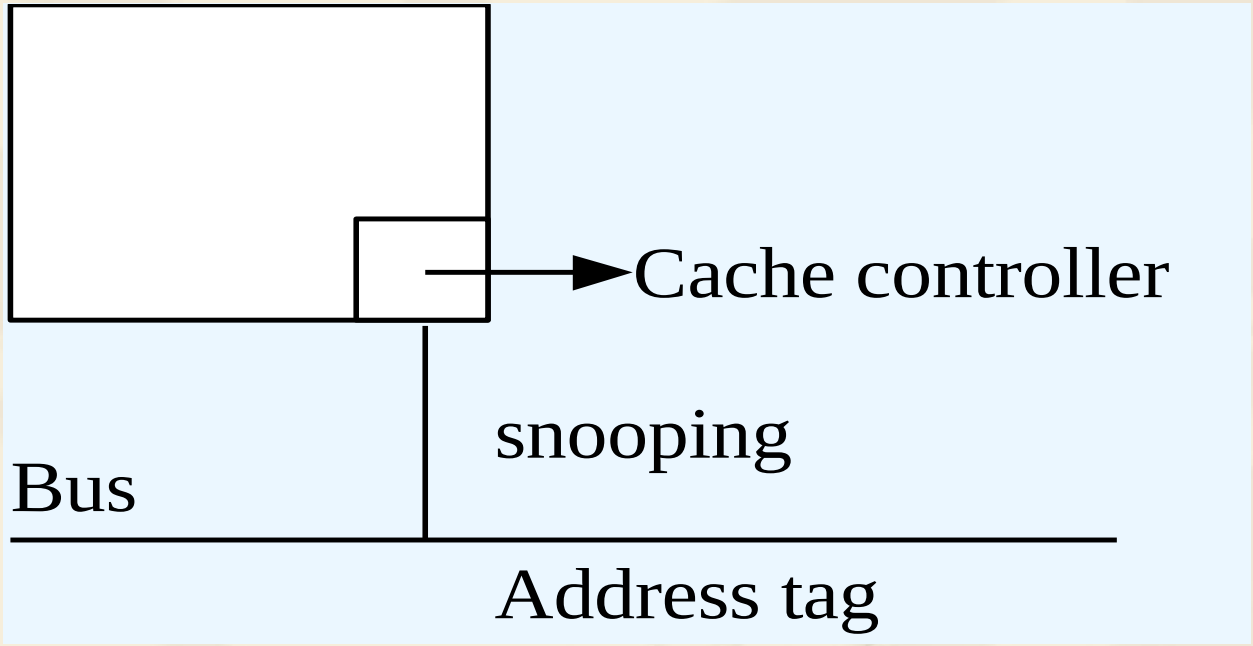
1. Directory based（基于目录）

- 把物理存储器中数据块的共享状态放在一个称为目录的结构之中。

2. Snooping（监听）

- 数据块的共享状态分散保留在每一拥有该数据块copy的Cache中，即不存在集中保留共享状态的结构。由于Cache通常是与共享存储器总线相连接，所有Cache控制器监控（monitor）or监听（snoop）总线，去发现它们（cache）是否拥有总线请求的数据块。


Used to drive Memory Bus
when Cache is Bus Master
Proc.
A
R/W
D
Tags and
State
Data
(lines)
Cache
A
R/W
Snoopy read port
attached to Memory
Bus



- **Snooping protocol在采用微处理器+Cache的共享存储多处理器系统中日益受到欢迎。因为协议可利用已存在的物理连接（bus to memory）达到查询Cache状态的目的。**

4.2.3 两种监听协议

1. Write invalidate protocol（写时无效协议）

- 在进行写操作时，该item在所有其它Cache中的copies均无效。所以一旦写过某item,则所有处理器从其cache中读该item时均发生miss（因为已无效，需从共享存储器中重新读出），即需取最新写入的copy。若两个处理器要同时写同一item时，只有一个赢得写的权利（以后再介绍，如何决定输赢），而另一个处理器必须得到该数据的新copy后再写，因此这一protocol能强制性的实现写串行化。(思考:同一个处理器连续写的情况)

例: 回写Cache和写无效snooping 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

两种监听协议（2）

2. Write update or write broadcast protocol（写时更新或写广播协议）

- **当某item被写后，该数据的所有copies均同时更新。为保证不超出存储器访问带宽，必须辨识被写的数据是否是共享数据，只有那些共享数据被写后才需广播，改写其需所有copies，否则就没有必要这样做。**

例：回写cache和写时更新监听协议

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	Write broadcast Of X	1	1	1
CPU B Reads X		1	1	1

三、两种协议性能的定性分析

- 对同一字的多次写操作，
  - 多次写广播；（写更新）
  - 只要一次无效化操作；（写无效）

    (思考:为什么不是多次无效化操作?)
- Cache数据块由多字组成的话，对块中每一字进行写操作：
  - 每次都要写广播；（写更新）**(write merge)**
  - 只在第一次写块中任一字时，需要产生一无效信号；（写无效）
- 从一个处理器写数到另一处理器读出写入的数的延时：
  - 写广播完成后，读命中；
  - 写无效化，读失配，stall直到得到返回值。

两种监听协议的比较：

- 两种方法自提出以来已有10年时间，目前写无效比写更新更为普及。
- 由两种协议性能的定性分析可知，写无效协议对bus和memory的带宽要求较低，因此成为几乎所有实现技术的选择。
- 在后面主要讨论写无效协议。

4.2.4 基本实现技术

一、实现写无效协议的关键是“利用总线完成无效化”。

1. 如何实现无效化监听协议

- 处理器请求总线访问；
- 处理器向总线广播将被无效的数据的地址；
- 所有Cache 控制器不断监听（snooping）总线，察看总线上的地址是否与其cache中的地址相符。若相符，则cache中该数据被无效化。
- 总线访问的顺序性（只有先取得总线访问权限的处理器才有权访问总线。）

基本实现技术（2）

2. 如何保证写操作的顺序性

- 因为两个处理器竞争同时写同一location的话，只有其中一个可获得总线的访问权，则另一处理器的Cache中的copy无效，从而使写操作按顺序执行。

基本实现技术（3）

3. 如何找到数据项最新（最近）的值？即在哪一个Cache中？
   - **对write-through cache很容易，因为Cache和memory的内容总是一致的。**
   - **对write-back cache较复杂。用snoop方法，每一Cache控制器snoops every address on the bus。若处理器找到它有一dirty copy of the requested cache block，该处理器就响应读的请求，提供该block的最新值，将该块写回存储器，并终止存储器访问。**

二、如何实现snooping?

1. 地址如何比较
   - **利用Cache地址的tag项，即将bus上的地址与其cache tags比较。**
2. 如何实现无效化？
   - **利用Cache block的Valid bit可方便地使该block无效化。**
3. 如何处理读miss?即找到该块的dirty copy?
   - **不管读miss是由无效化造成还是其他事件造成，可通过snooping获得最新的block的值。**

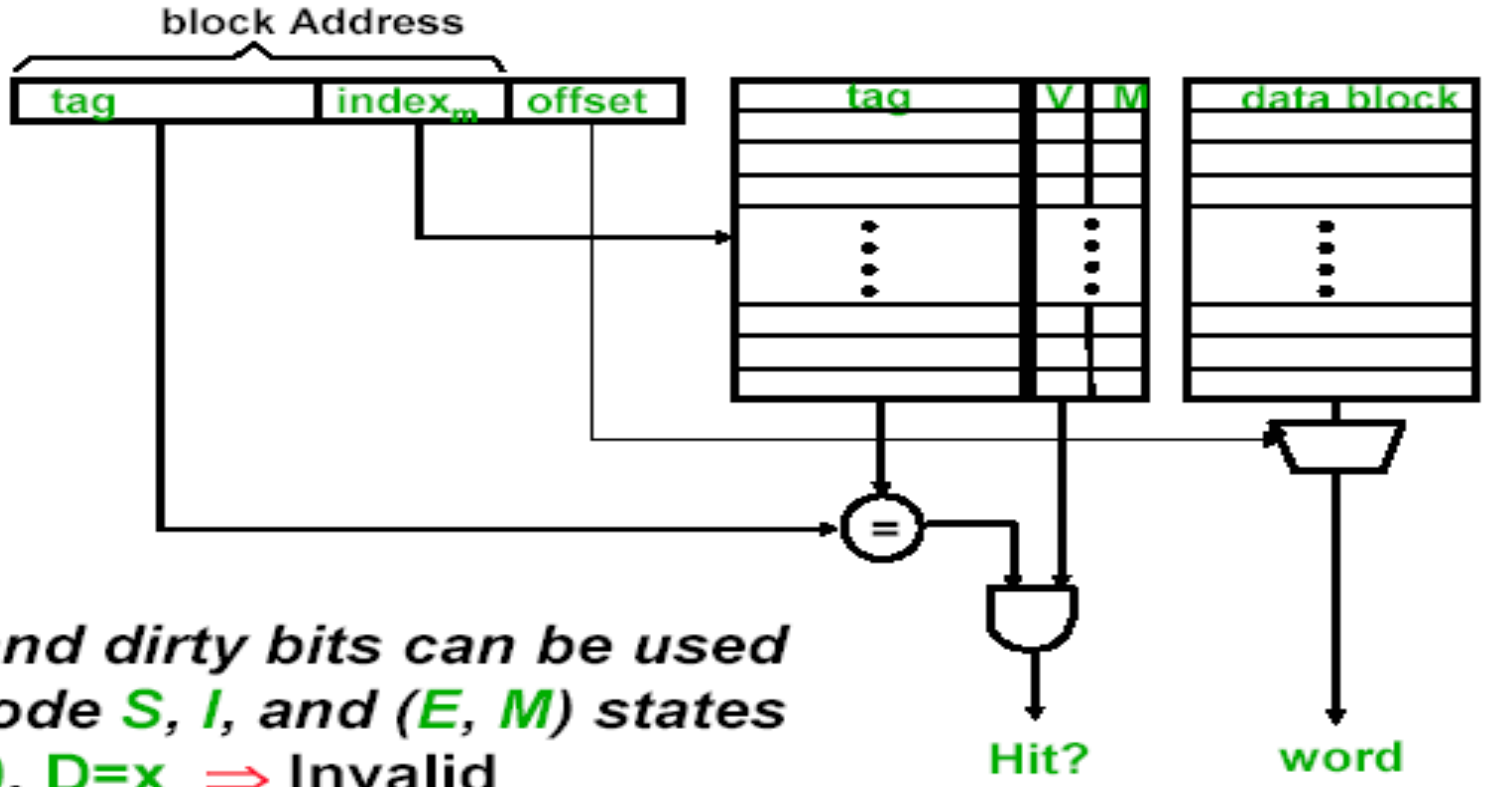
4. 如何处理写操作？

- 先区分是否是共享数据，即知道将写的block是否也在其它cache中。若无其它cached copies,（即为非共享数据），则对write-back cache讲，没有必要把写操作放到bus上，从而达到节省时间，又减轻带宽的负担。

5. 如何跟踪一cache block是否是共享的？

- 在每一block上加一新的状态位来指示该block是否是共享的（类似于valid bit或dirty bit）。
- 若写入的block处于共享状态，则cache在写后要向bus送一无效信号，并将该block标记为private。
- 然后block状态要改变。

Cache Coherence State Encoding



Valid and dirty bits can be used to encode S, I, and (E, M) states

V=0, D=x ⇒ Invalid

V=1, D=0 ⇒ Shared *(not dirty)*

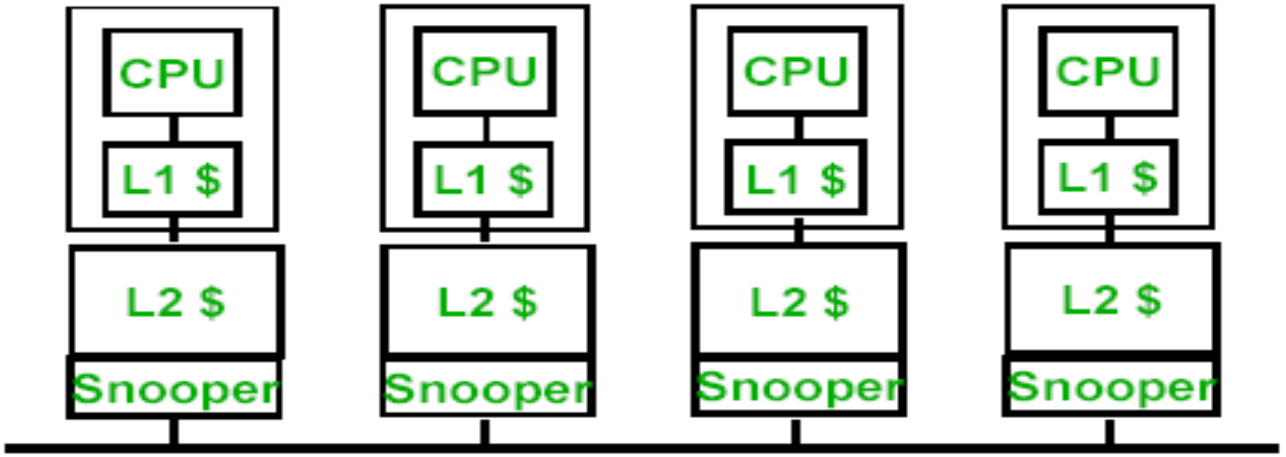
V=1, D=1 ⇒ Exclusive *(dirty)*

三、如何减少由bus检查cache-address tag 引起的对CPU cache access的干涉？

1. 复制tags: 在同一**block增加一个tags。若tags复制以后，则CPU和snooping的访问可并行进行；**
2. 多级Cache with inclusion property: **Snooping activity可对次级Cache进行，而大多数CPU activity可对初级Cache进行。**

- **由于大多数高级微处理器均采用多级Cache来降低对带宽的要求，故这一方法已被大多数设计所采用，有时还复制二级Cache的tags来进一步减少CPU和snooping的竞争。**

2-Level Caches



- Processors often have two-level caches
  - Small L1 on chip, large L2 off chip
- *Inclusion property:* entries in L1 must be in L2
  invalidation in L2 ⇒ invalidation in L1
- Snooping on L2 does not affect CPU-L1 bandwidth
- *What problem could occur?*

小结

- 保证Cache一致性的写时无效协议，**意味着：**
  - **必须对被写Cache块的其它拷贝进行无效化处理**
  - **必须能在Cache失配时找到有效数据，对直写cache----总是到memory中去找最新值；对回写cache----采用监听方法，所有CPU先比较地址，再发现是否dirty。是dirty，即最新的数据。**
- 使用监听在总线上完成判断：
  - **不断监听总线上的地址**
  - **不断检查是否与自己cache中地址相符合。如果是，则根据不同事件采取不同措施。**

五种snooping protocols

Name	Protocol type	Memory-write policy	Unique feature	Machine using
Write once	Write invalidate	Write back after first write	First snooping protocol described in literature	
Synapse N+1	Write invalidate	Write back	Explicit state Where Memory is the owner	Synapse machine; first cache-coherence machines available
Berkeley	Write invalidate	Write back	Owned shared state	Berkeley SPUR machine
Illinois	Write invalidate	Write back	Clean private State; Can supply data from any cache with a clean copy	SGI Power and Challenge series
“Firefly”	Write broadcast	Write back When private, Write through When shared	Memory updated on broadcast	No current machines; SPARCCenter 2000 closest

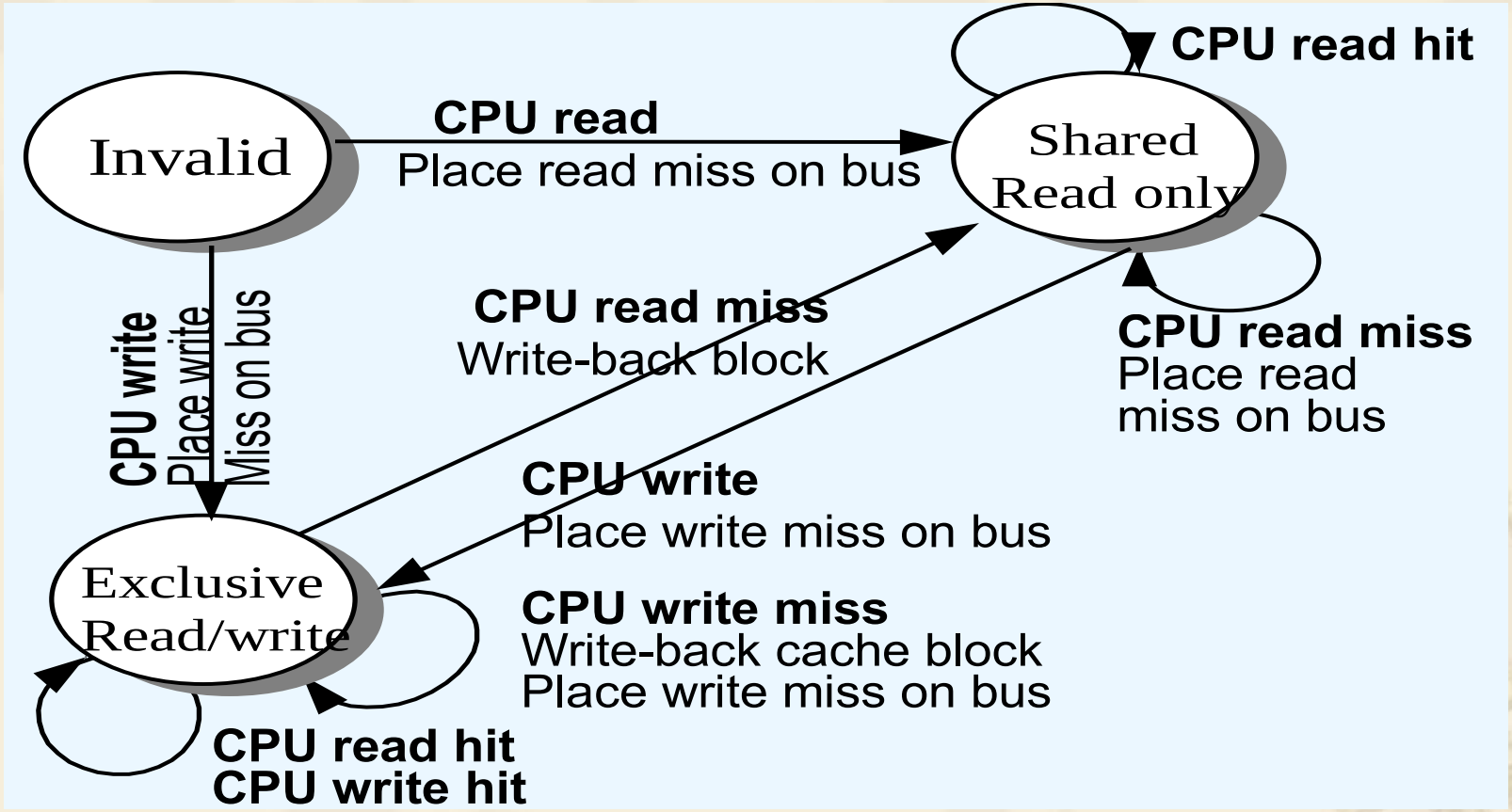
4.2.5 An example protocol

一、一致性机制的请求和操作

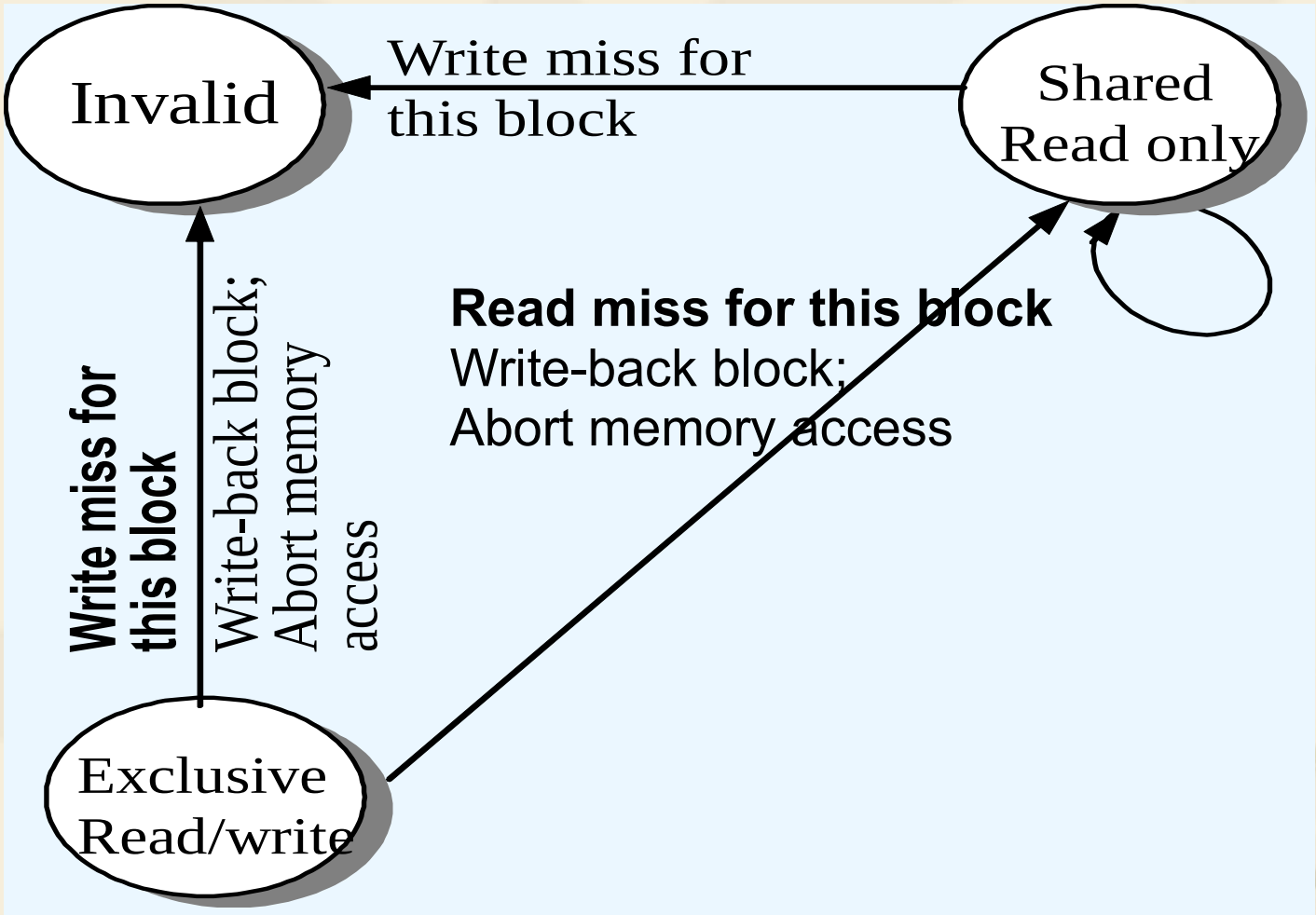
Request	Source	State of addressed cache block	Function and explanation
Read hit	Processor	Shared or Exclusive	Read data in cache
Read miss	Processor	Invalid	Place read miss on bus.
Read miss	Processor	Shared	Address conflict miss: place read miss on bus
Read miss	Processor	Exclusive	Address conflict miss: write back block, then place read miss on bus
Write hit	Processor	Exclusive	Write data in cache.
Write hit	Processor	Shared	Place write miss on bus.
Write miss	Processor	Invalid	Place write miss on bus.
Write miss	Processor	Shared	Address conflict miss: place write miss on bus
Write miss	Processor	Exclusive	Address conflict miss: write back block, then place write miss on bus
Read Miss	Bus	Shared	No action; allow memory to service read miss.
Read Miss	Bus	Exclusive	Attempt to share data: place cache block on bus and change state to Shared.
Write miss	Bus	Shared	Attempt to write shared block; invalidate the block.
Write miss	Bus	Exclusive	Attempt to write block that is exclusive elsewhere: write back the cache block and make its state Invalid.

Cache' state transitions based on requests

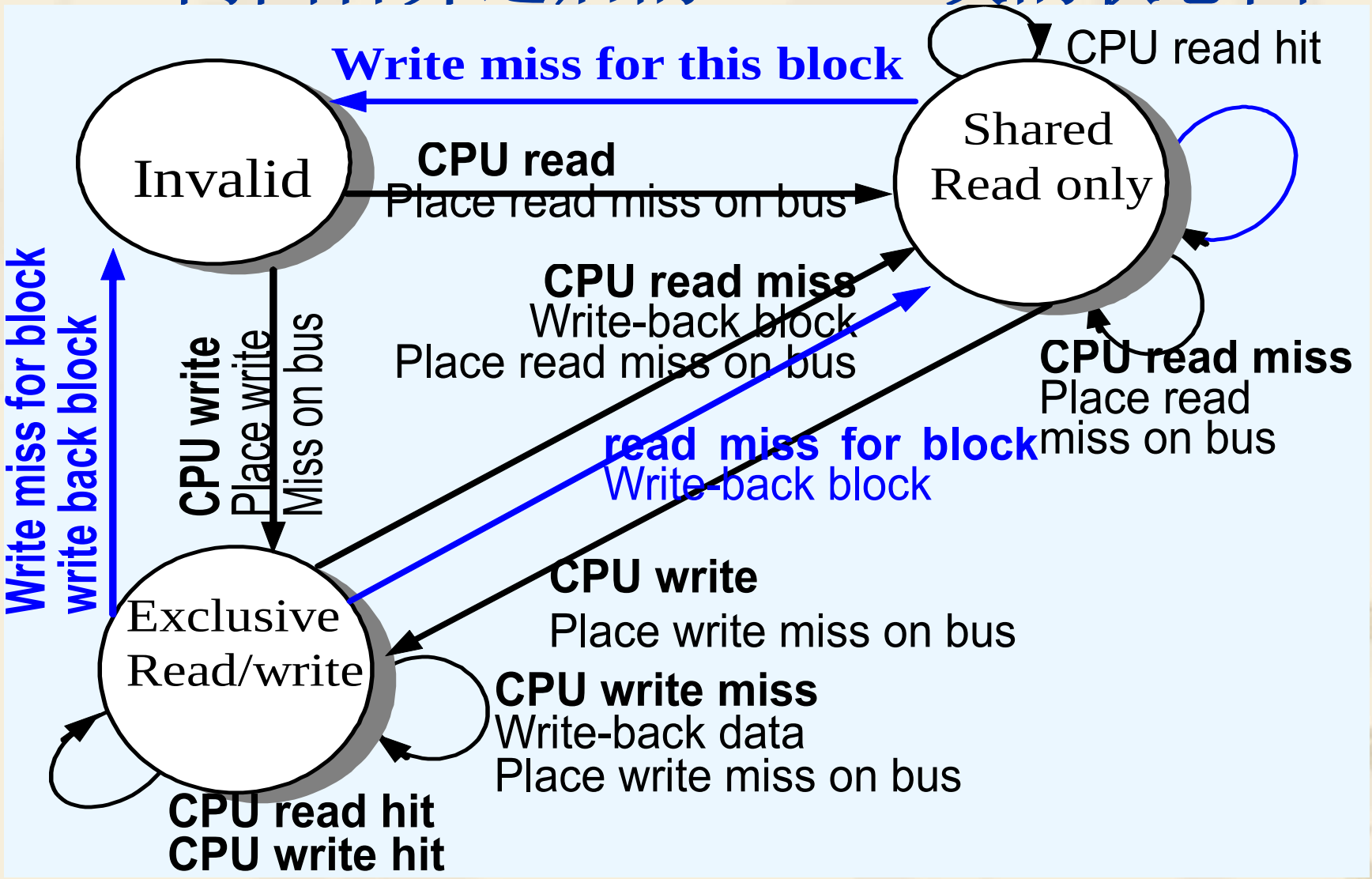
1. from CPU



Cache' state transitions based on requests 2. from the bus



3. 两图合并之后的Cache块的状态图



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

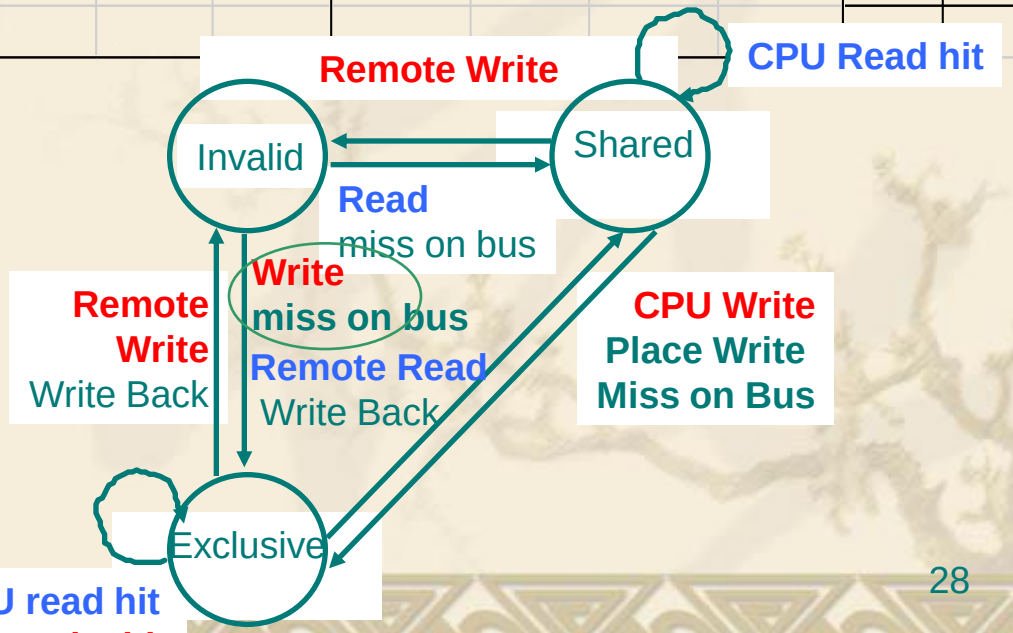
Example: step 1

	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												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state

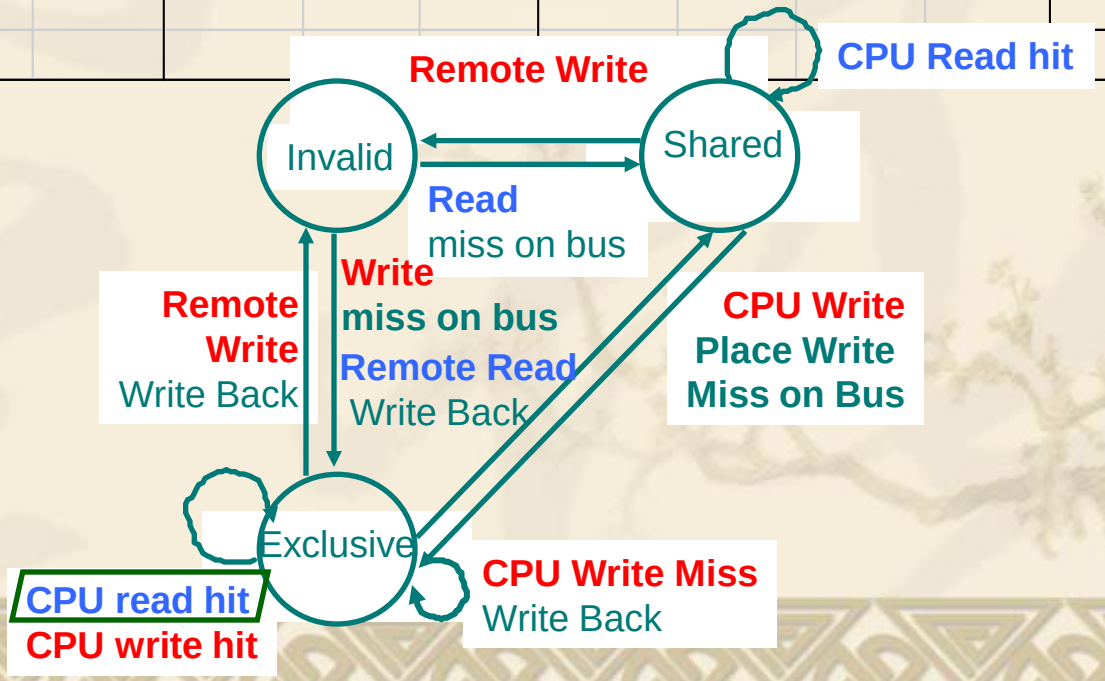
is invalid and A1 and A2 map to same cache block,

but A1 != A2.



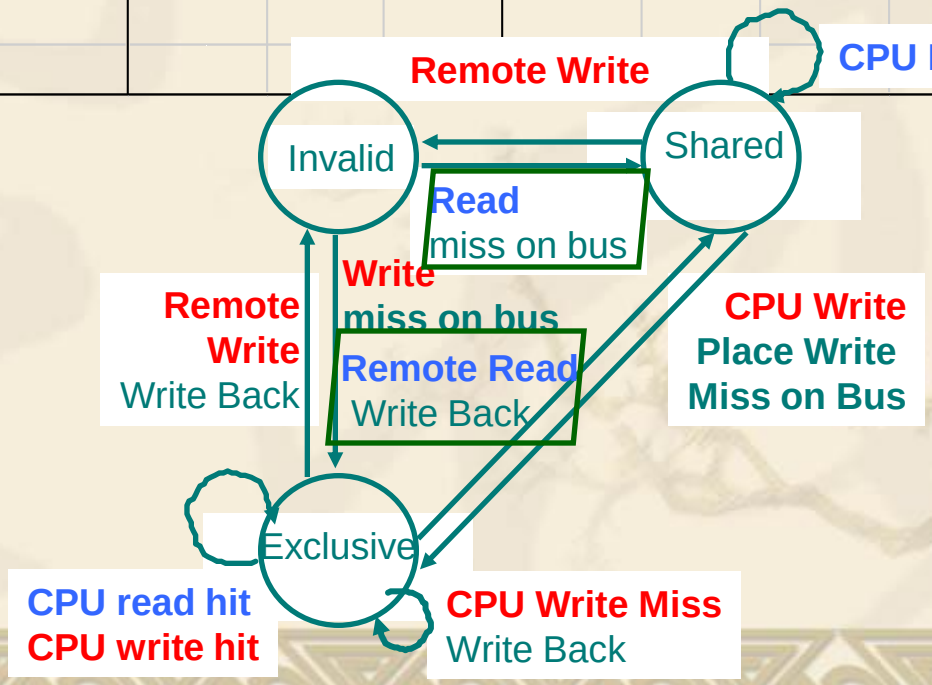
Example: step 2

	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												



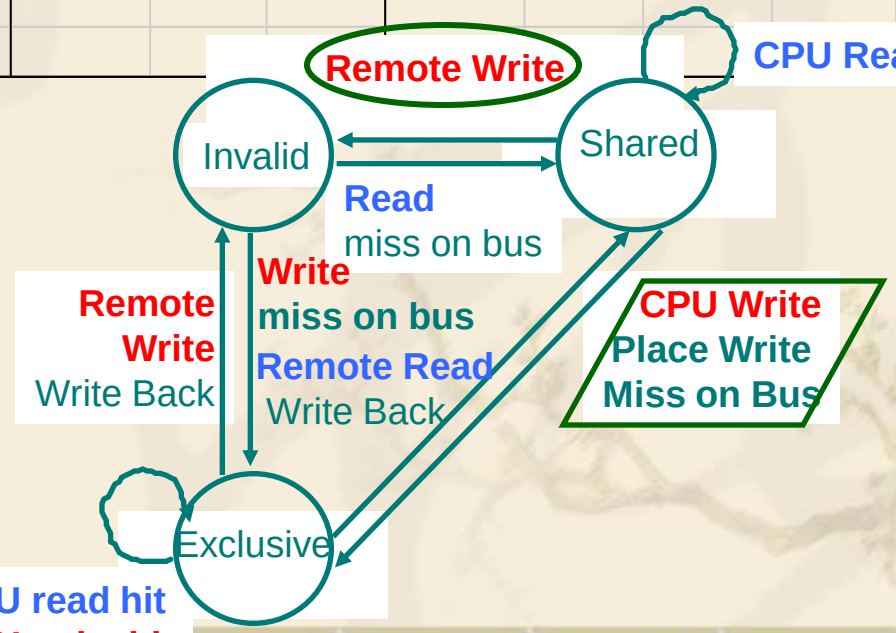
Example:step 3

	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				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1												
P2: Write 40 to A2												



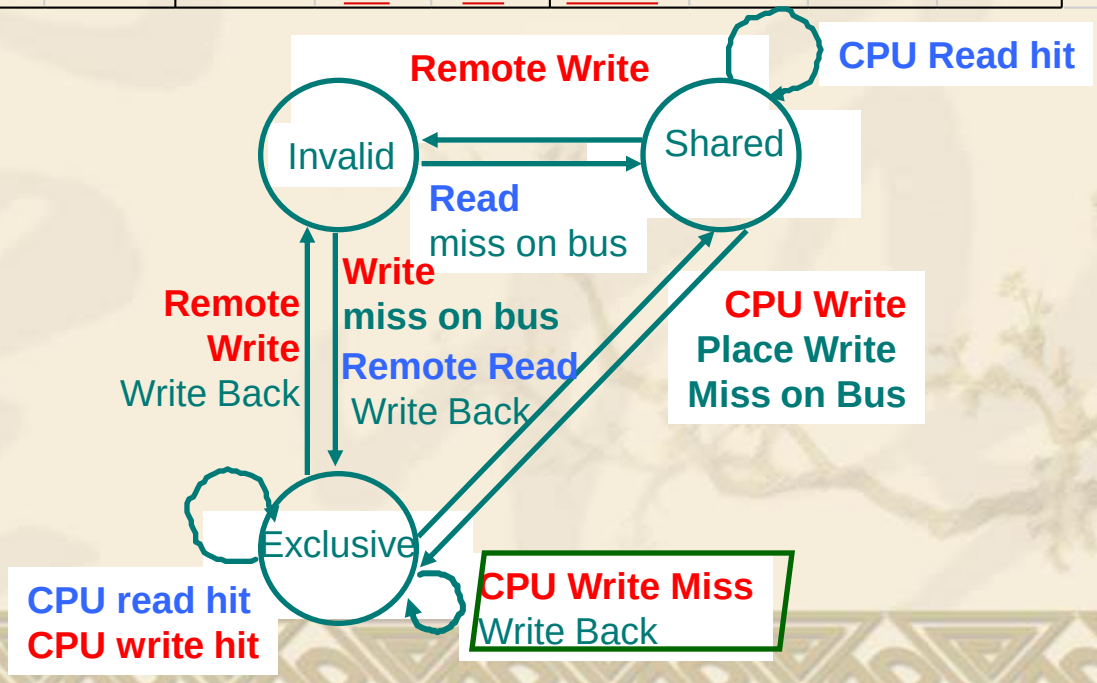
Example: step4

	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				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2												



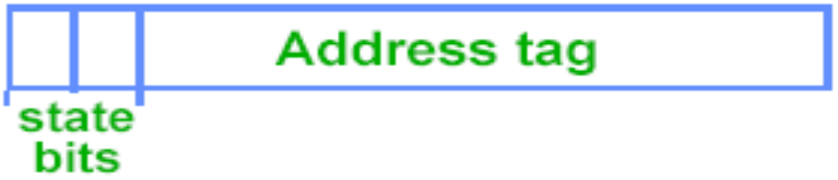
Example:step 5

step	P1 State	P1 Addr	P1 Value	P2 State	P2 Addr	P2 Value	Bus Action	Bus Proc.	Bus Addr	Bus Value	Memory Addr	Memory 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				*WrBk*	P1	A1	10	A1	*10*
				Shar.	A1	*10*	*RdDa*	P2	A1	10	A1	10
P2: Write 20 to A1	*Inv.*			*Excl.*	A1	*20*	*WrMs*	P2	A1		A1	10
P2: Write 40 to A2							*WrMs*	P2	A2		A1	10
				Excl.	*A2*	*40*	*WrBk*	P2	A1	20	A1	*20*

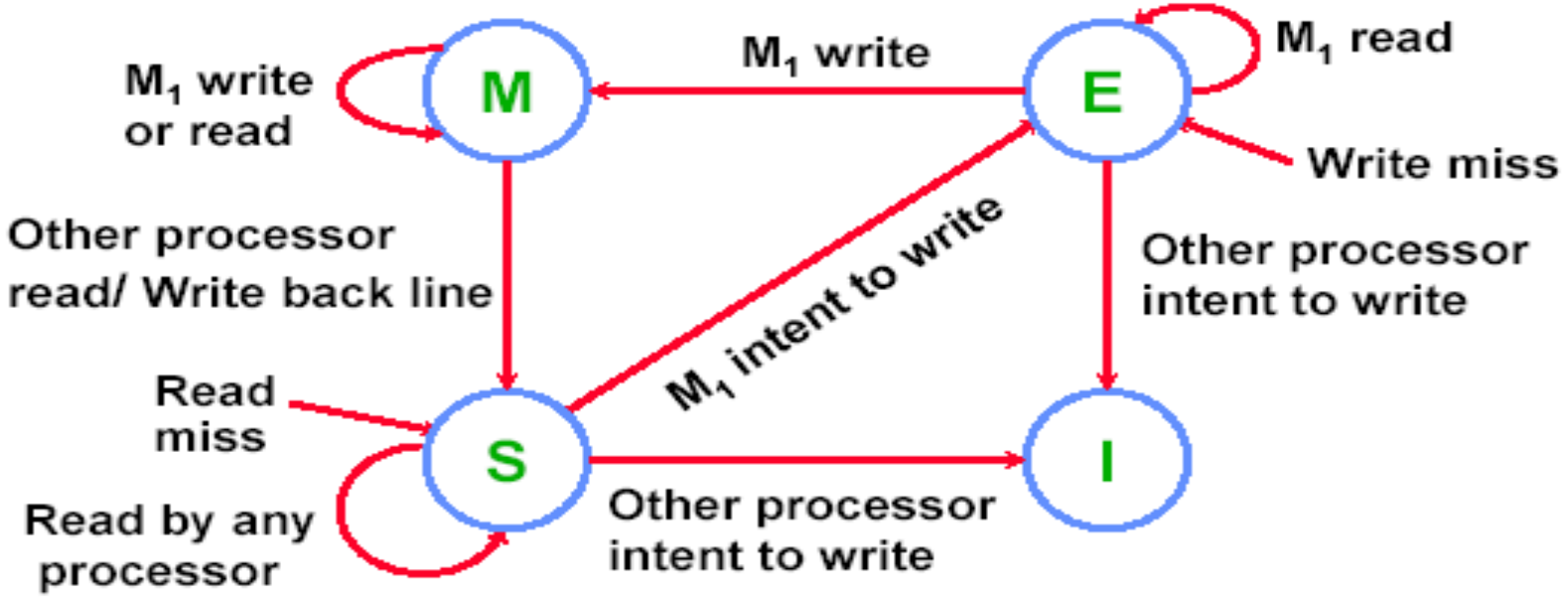


Cache State Transition Diagram

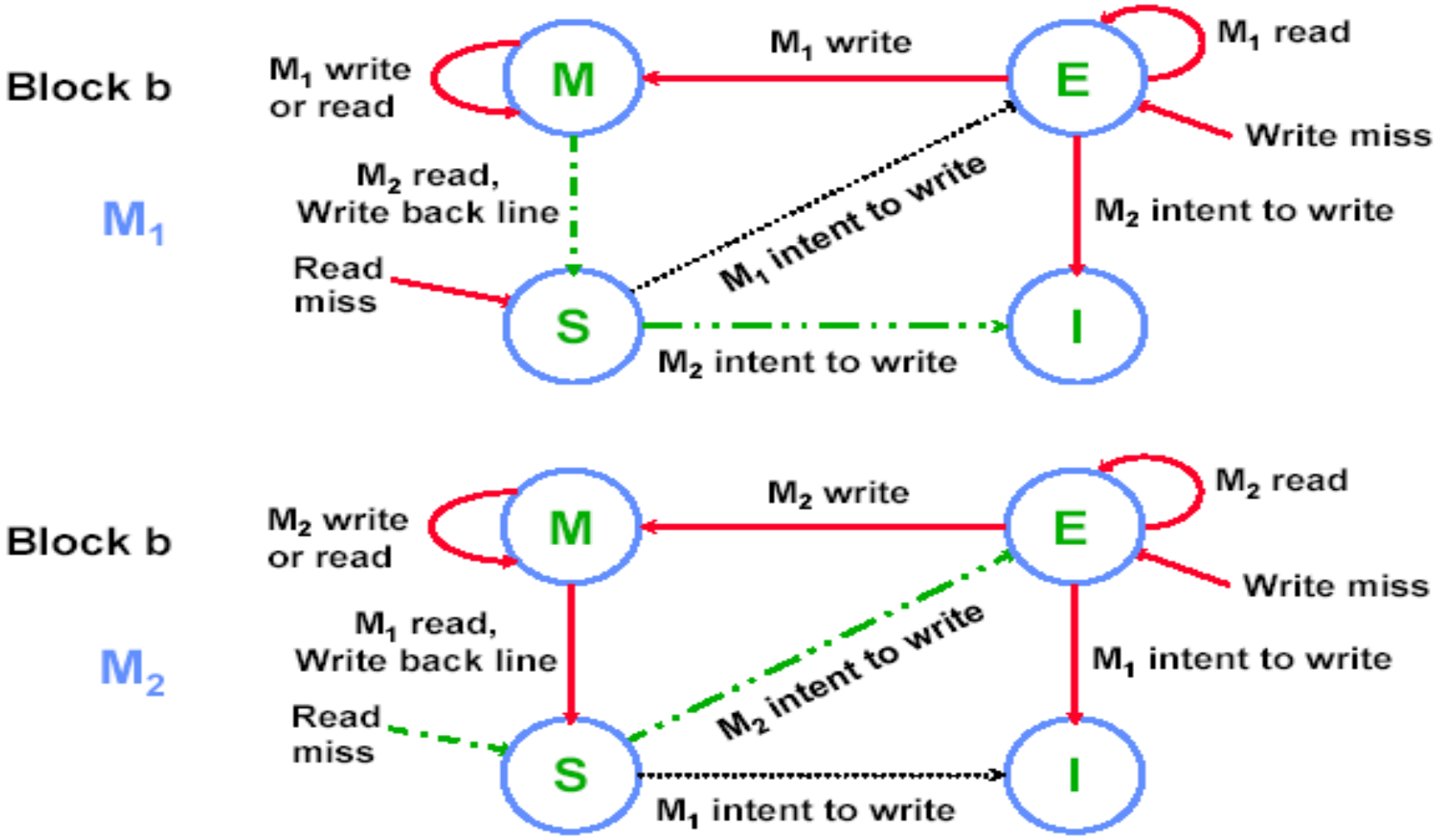
Each cache line tag



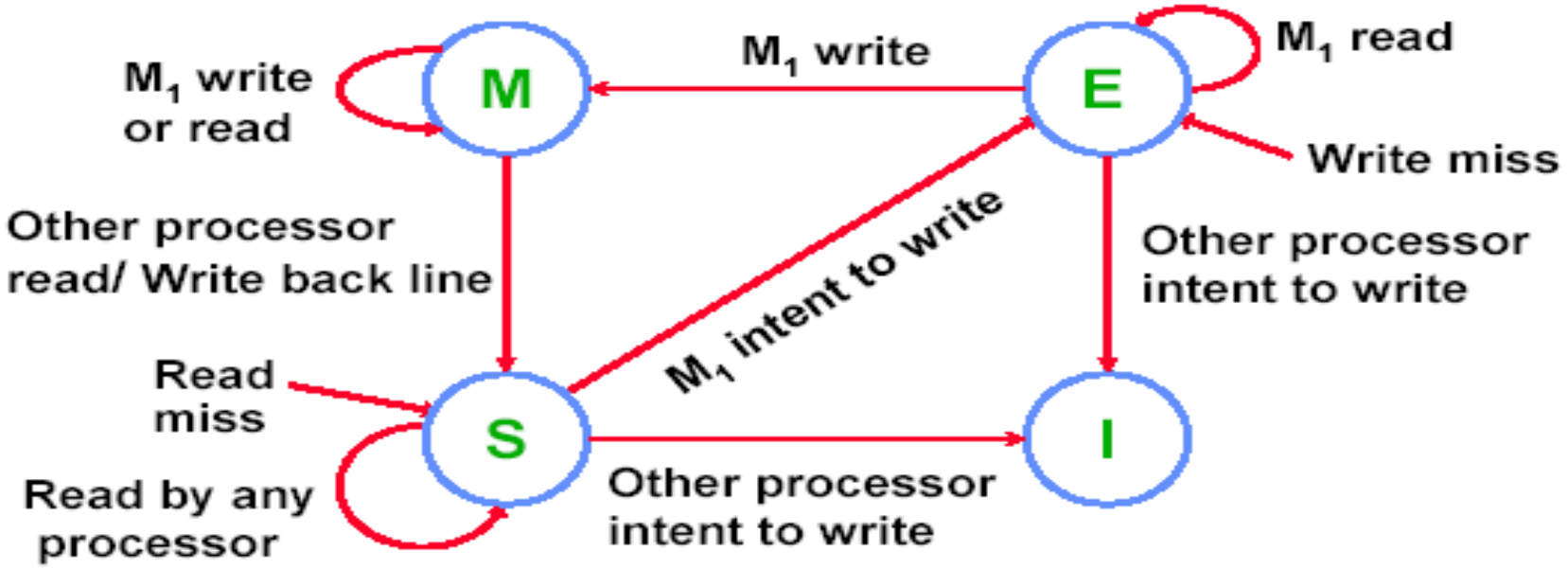
M: Modified Exclusive
E: Exclusive, unmodified
S: Shared
I: Invalid



2 Processor Example



Observation



If a line is in **M** or **E**, then no other cache has a valid copy of the line!

- Memory stays coherent, multiple differing copies cannot exist

4.3 Distributed shared-Memory Architectures

- 4.3.1 分布式存储器结构

一、在这一体系结构下有两种结构

- **无Cache coherence**
  - **共享数据不进入缓存**

  例子：**Cray T3D,注意力集中在可缩放存储系统**
- **有Cache coherence**

二、无Cache coherence（1）

- memory distributed among the nodes
- all nodes are interconnected by a network
- access can be either local or remote, 由node中的控制器根据地址决定数据是在本地存储器还是在远程存储器。
- 若需远程访问，则发消息给远程存储器的控制器去访问数据；
- 这类系统是带cache的，但为避免coherence问题，将那些共享数据标记为uncacheable, 只允许private data 存放于Cache。因为共享数据不会在Cache，所以远程访问只能按字，而不能按block进行。

无Cache coherence（2）

- 仍然可用软件显式地将共享数据放进cache,然后由软件来管理coherence。（通常，cache是由硬件管理，而不是由软件管理）

- 优点：不需要任何硬件
- 缺点：
  - 不能实现基于compiler的透明的软件cache coherence;
  - 因为无cache coherence,丧失了取single cache block的优点，也丧失了利用共享数据的空间局部性（因为每次远程访问只能按字进行）
  - 失去了prefetch等带来的tolerating latency 机制。

- 根据上述分析，小规模多处理器(集中式存储)比较容易支持 **cache coherence**；对大规模多处理器体系结构(分布式存储)而言，还存在其它挑战，如缺乏监听**coherence**机制的缩放性问题。

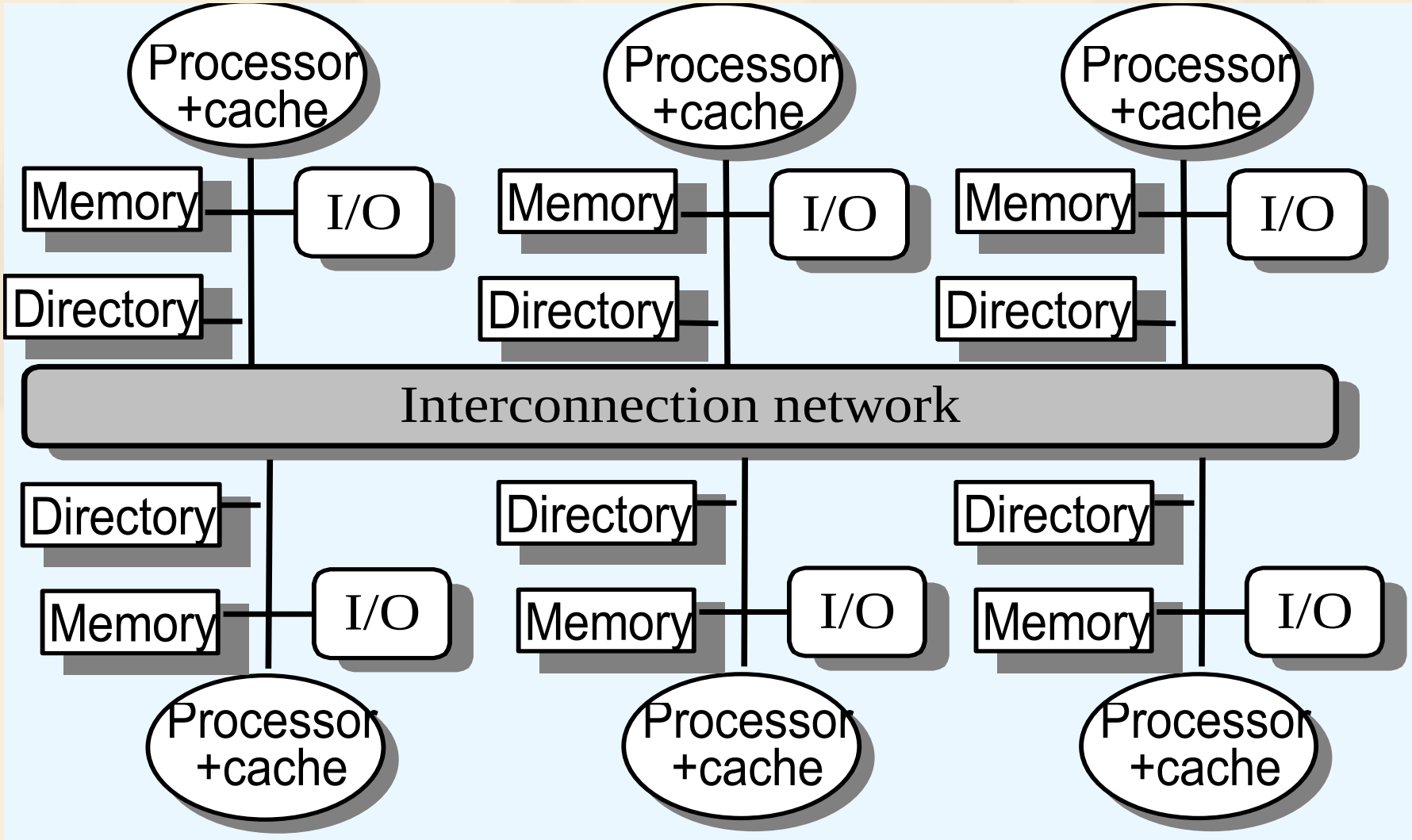
三、有Cache coherence

- 关键：找到另一种不同于snooping protocol的coherence protocol，这就是directory protocol。因为在分布存储器结构中，不能采用总线作为仲裁，而且处理器之间通信要采用显式的消息发送。
- 目录保存的信息
  - 每一可能存入cache的block的状态：包括共享（shared)、未进入cache（uncached）、独占的（exclusive）
  - 哪些cache拥有该block的copies，block是否dirty等等。

Directory protocol 实现方法

- 目录的entry与存储器的block相联系的方法。
  - 典型的基于目录协议的所含的信息量与各存储器中block数与处理器数的乘积成正比。对于处理器个数小于100的机器，这样的信息量还是可允许的，但对于大规模多处理器而言，必须设法减小目录的信息量。
- 常用减少信息量的方法：
  - 在目录里只放少数block的信息，而不是针对存储器中所有的block;
  - 每一entry缩减bits数。

- 为了防止访问目录成为瓶颈，目录的**entries**可分布存放在存储器上，成为分布式目录。
- 每个目录负责跟踪拥有本节点存储器部分地址的**cache**。


Processor +cache
Processor +cache
Processor +cache
Memory
I/O
Memory
I/O
Memory
I/O
Directory
Directory
Directory
Interconnection network
Directory
Directory
Directory
Memory
I/O
Memory
I/O
Memory
I/O
Processor +cache
Processor +cache
Processor +cache

4.3.2 基于目录的cache一致性协议基础

一、基于目录协议的两种基本操作：

- 处理读失配；
- 处理对共享的干净的cache块的写入

❖ 实现关键：

- 目录必须跟踪每一cache块的状态。每一个处理器跟踪各自cache中每一数据块的状态。
- 必须跟踪保存了共享块拷贝的各处理器，因为一旦某处理器对该共享数据写过一次后，须对其它copies作无效处理。

二、目录中Cache 块可能的状态

- 共享（shared）
  - 该块的**copies**存在于一个或多个**processor**的**caches**中；
- 未进入Cache（uncached）
  - 没有一个处理器将此块拷入其**cache**中；
- 独占（exclusive）
  - 只有一个处理器保存此块的拷贝，并更新过数据，于是内存中的数据已过时。此处理器为此数据的拥有者（**owner**）。

三、如何表示共享block被哪一处理器共享？（其copies在哪些处理器的Cache中？）

1. 每一个data block设置一个bit vectors
   - bit vector 中的每一位表示一个对应的处理器，将其bit置位表示该处理器共享该block（保存有该块的copy），可用该bit vector表示该block处于exclusive状态时的owner。
   - 置位表示对应处理器拥有共享数据

2. 目录协议的操作约定（与监听技术相同的假设）

- 写入非独占数据时，一定会导致Cache写失配，且处理器将暂停直到一次访问结束。

3. 基于目录的Cache一致性协议与snooping 不同之处：

- Snooping把总线（互连机制）作为判断点，起仲裁作用。Director协议不能把互连网络作为判断点。
- Director写是面向消息的，（不象总线是面向事务的，可采用中断方式），所有消息必须明确应答。

四、处理器与目录间传递消息的种类

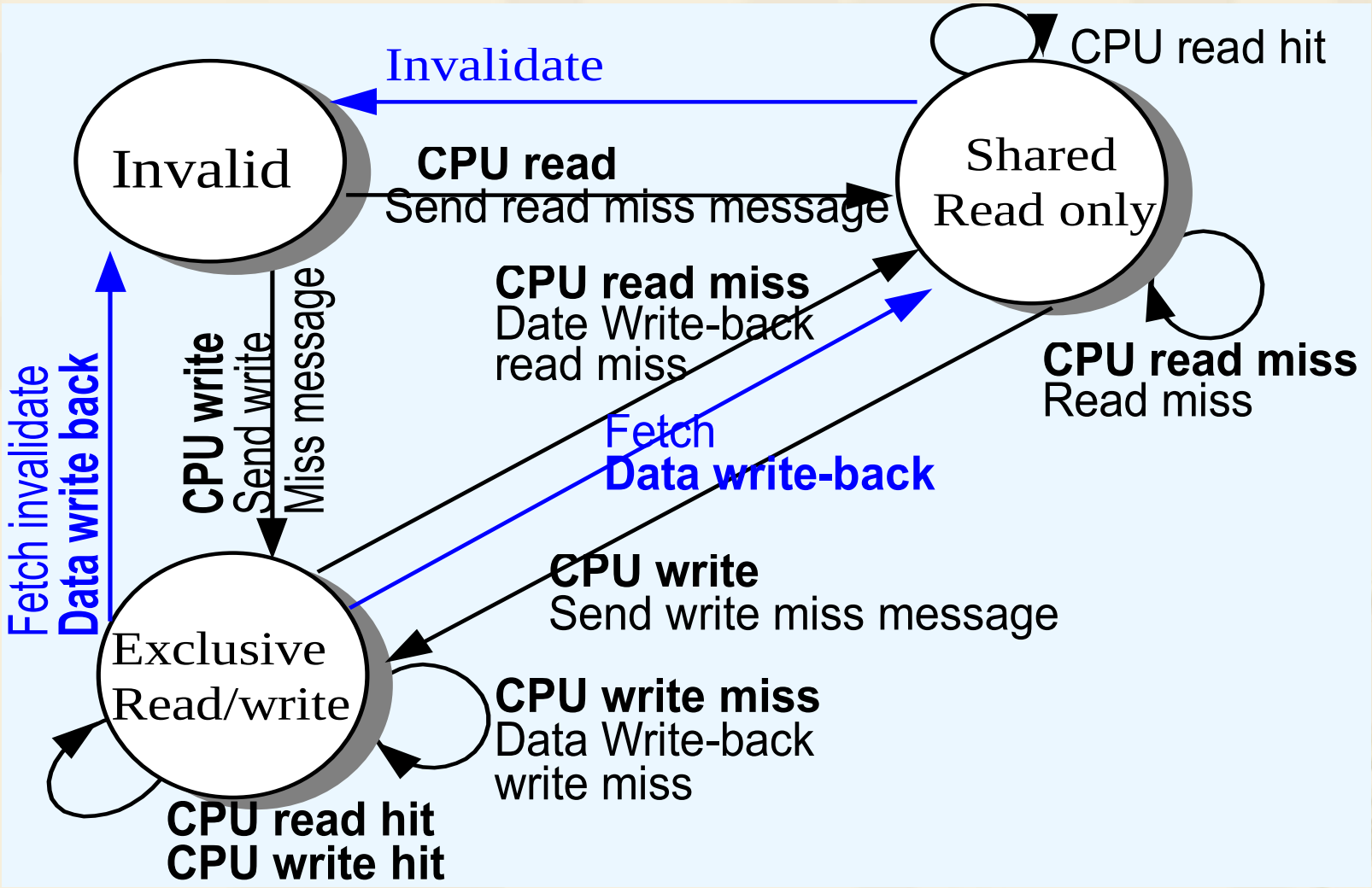
Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Processor P has a write miss at address A; — request data and make P the exclusive owner.
Invalidate	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 from the home memory.
Data write back	Remote cache	Home directory	A, D	Write back a data value for address A.

节点分类

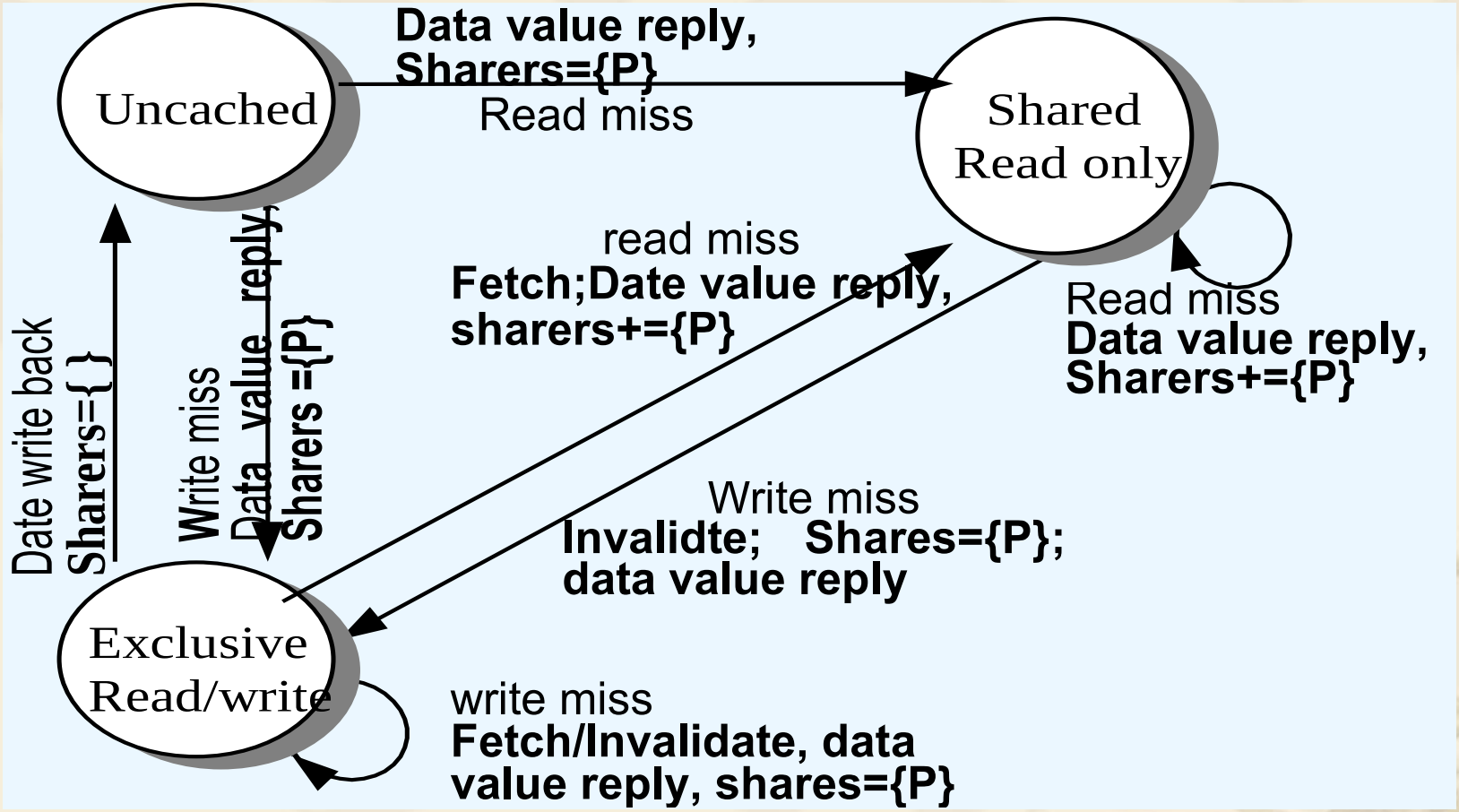
- 本地节点（local node）
  - 指产生访问请求的节点
- 家节点（home node）
  - 指该节点拥有要访问地址的存储器单元和目录项（即要访问的数据的家）
- 远程节点（remote node）
  - 指拥有要访问数据拷贝的节点。

4.3.3 目录协议中的范例

一、目录协议的Cache中数据块状态转换图



二、目录中数据块记录的状态转换图



Example

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														
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			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			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	Excl.	A1	10											
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			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	Excl.	A1	10											
P2: Read A1				Shar.	A1		RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10	A1			10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	P1,P2}	10
P2: Write 20 to A1														
P2: Write 40 to A2														

Write Back

A1 and A2 map to the same cache block

Example

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	Excl.	A1	10											
P2: Read A1				Shar.	A1		RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10	A1			10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	P1,P2}	10
P2: Write 20 to A1				Excl.	A1	20	WrMs	P2	A1					10
	Inv.						Inval.	P1	A1		A1	Excl.	{P2}	10
P2: Write 40 to A2														

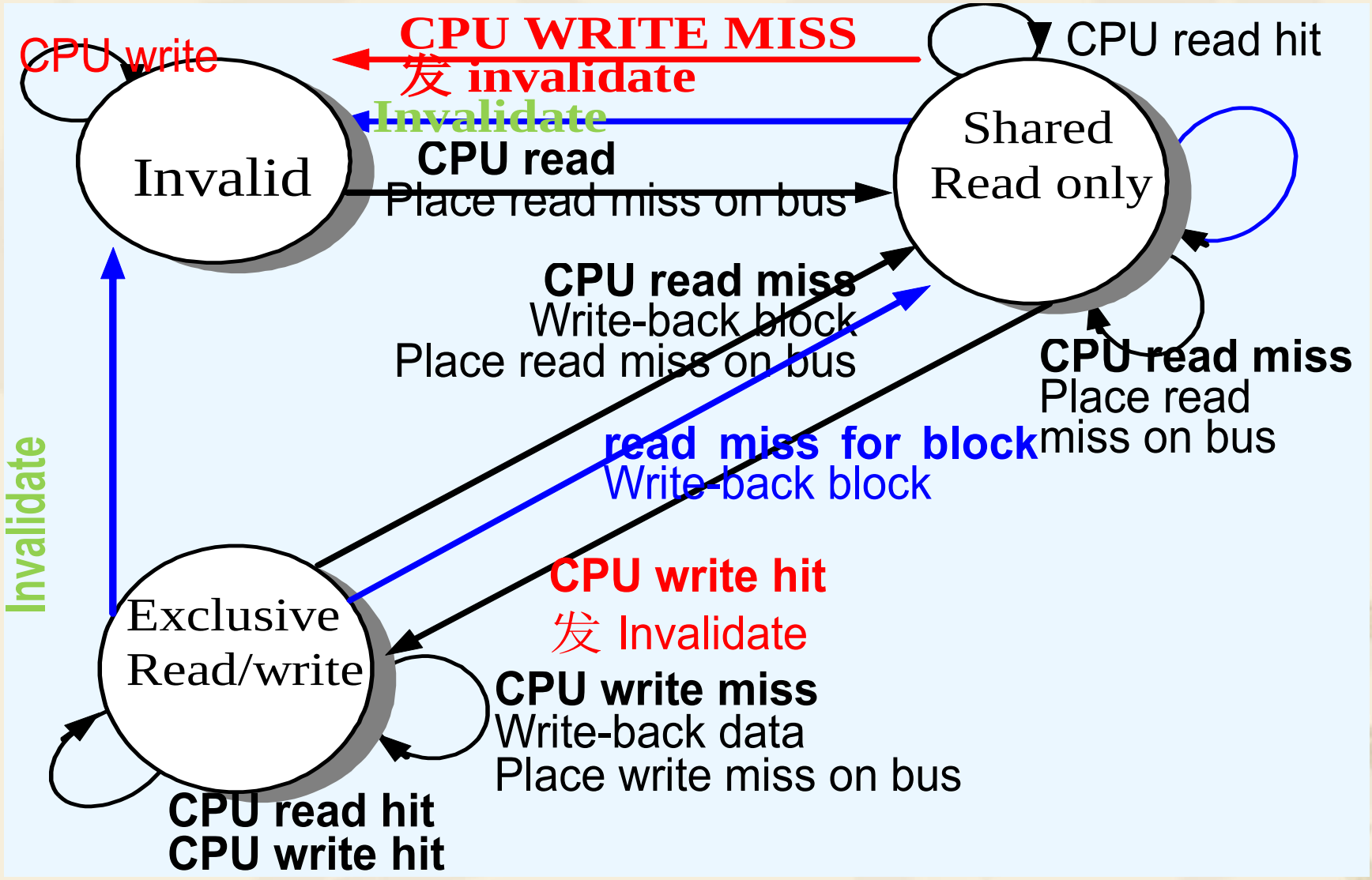
A1 and A2 map to the same cache block

Example

Processor 1 Processor 2 Interconnect Directory Memory

	P1			P2			Bus				Directory			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	Excl.	A1	10											
P2: Read A1				Shar.	A1		RdMs	P2	A1					
	Shar.	A1	10				Ftch	P1	A1	10	A1			10
				Shar.	A1	10	DaRp	P2	A1	10	A1	Shar.	P1,P2}	10
P2: Write 20 to A1				Excl.	A1	20	WrMs	P2	A1					10
	Inv.						Inval.	P1	A1		A1	Excl.	{P2}	10
P2: Write 40 to A2							WrMs	P2	A2		A2	Excl.	{P2}	0
							WrBk	P2	A1	20	A1	Unca.	{}	20
				Excl.	A2	40	DaRp	P2	A2	0	A2	Excl.	{P2}	0

A1 and A2 map to the same cache block


CPU write
CPU WRITE MISS
发 invalidate
CPU read hit
Invalidate
Invalid
CPU read
Place read miss on bus
Shared
Read only
CPU read miss
Write-back block
Place read miss on bus
CPU read miss
Place read
miss on bus
read miss for block
Write-back block
Invalidate
CPU write hit
发 Invalidate
Exclusive
Read/write
CPU write miss
Write-back data
Place write miss on bus
CPU read hit
CPU write hit