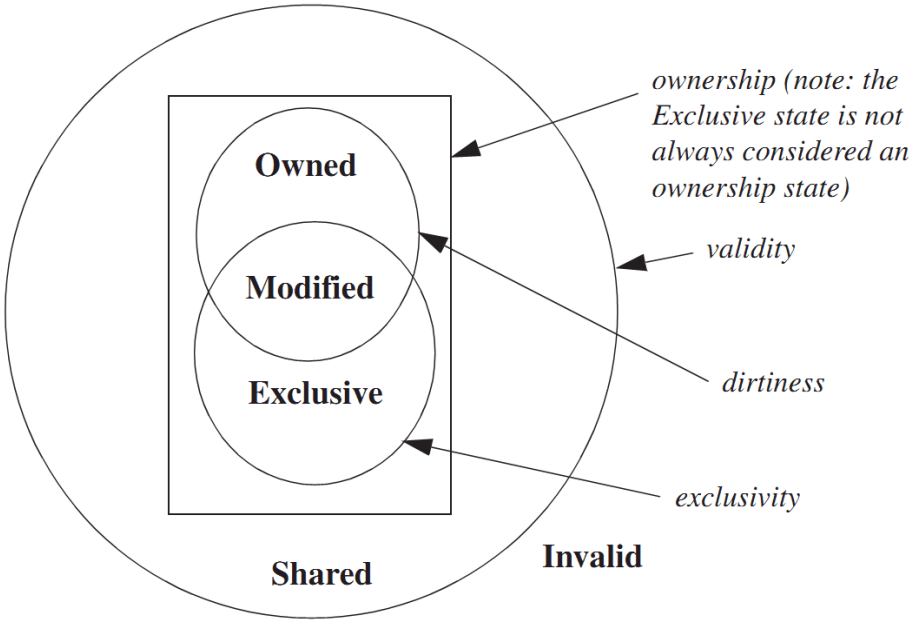
计算机体系结构

Cache Coherence

Review

- **共享数据块的跟踪：监听和目录**
- **Cache一致性协议实现：写作废和写更新**
- **集中式共享存储 Cache一致性协议**
  - Snooping协议：MSI，MESI，MOESI
- **Coherency Misses**
  - True Sharing
  - False Sharing



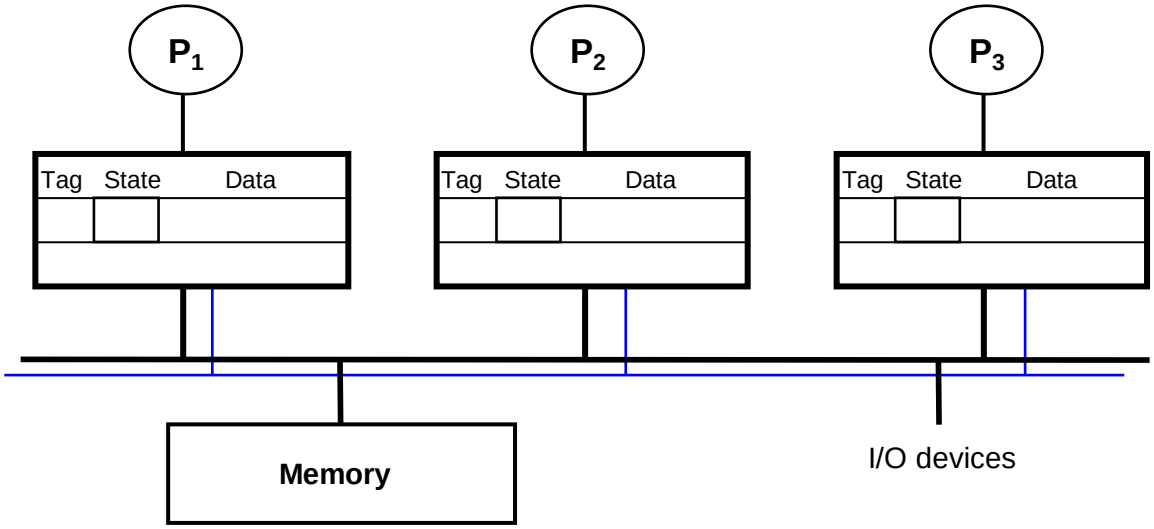
MESI Write-Back Invalidation Protocol

- **MSI Protocol的缺陷：**
  - 读进并修改一个block，产生2 个总线事务
    - 首先是读操作产生 BusRd (I→S)，并置状态为Shared, 写更新时产生 BusRdX (S→M)
    - 即使一个块是Cache独占的这种情况仍然存在
    - 使用多道程序负载时，这种情况很普遍
- **增加exclusive state, 减少总线事务**
  - Exclusive state 表示仅当前Cache包含该块，并且是干净的块
  - 区分独占块的“clean”和“dirty”
  - 一个处于exclusive state的块，更新时不产生总线事务

Four States: MESI

- **M: Modified**
  - 仅当前Cache含有该块，并且该块被修改过
  - 内存中的Copy是陈旧的值
- **E: Exclusive or exclusive-clean**
  - 仅当前Cache含有该块，并且该块没被修改过
  - 内存中的数据是最新的
- **S: Shared**
  - 多个Cache中都含有本块，而且都没有修改过
  - 内存中的数据是最新的
- **I: Invalid**
- **也称Illinois protocol**
  - 首先是由Illinois的研究人员研制并发表论文
  - MESI协议的变种广泛应用于现代微处理器中

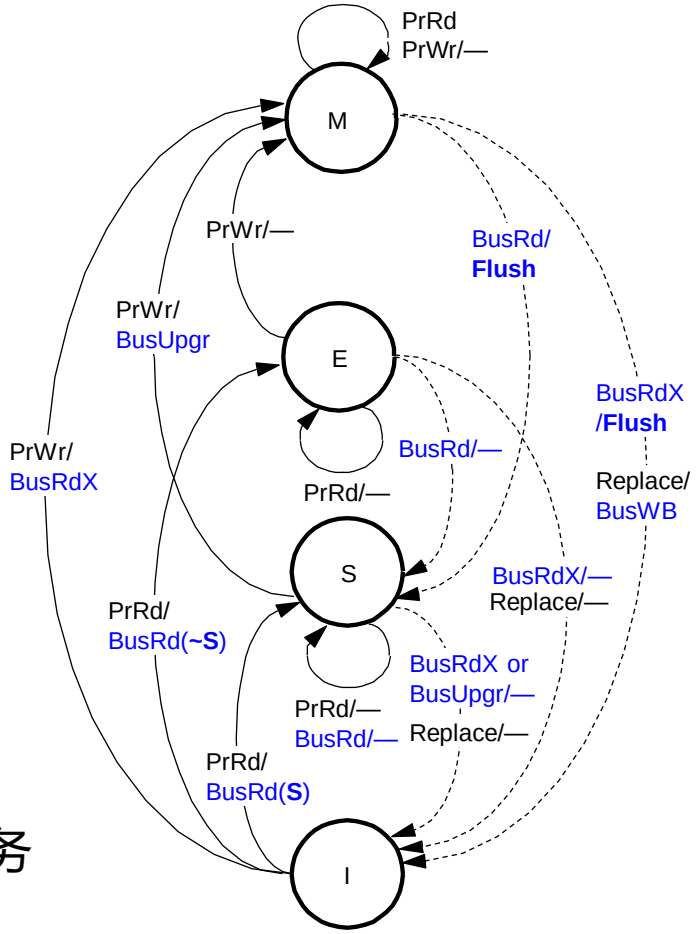
Hardware Support for MESI



- **总线互连的新要求**
  - 增加一个称为shared signal S, 必须对所有Cache控制器可用
  - 可以实现成 wired-OR line
- **所有cache controllers 监测 BusRd**
  - 如果所访问的块的状态是（state S, E, or M)
  - 请求Cache 根据shared signal选择E或S

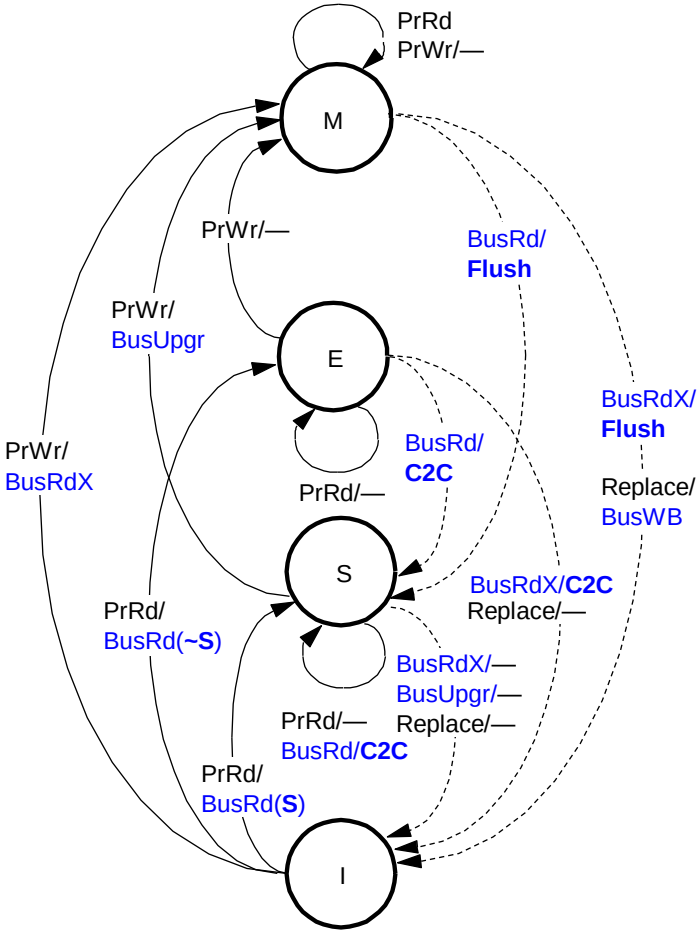
MESI State Transition Diagram

- **Processor Read**
  - 读失效时产生BusRd事务
  - BusRd(S) => shared line asserted
    - 在其他Cache中有有效的copy
    - Goto state *S*
  - BusRd(~S) => shared line not asserted
    - 在其他Cache中不存在该块
    - Goto state *E*
  - 读命中时不产生总线事务
- **Processor Write**
  - 该Cache块的状态转至 *M*
  - 在I或S态(命中）产生 BusRdX / BusUpgr
    - 作废其他Cache中的Copies
  - Cache块处于状态E and M时，不产生总线事务
  - E或S态写失效时，引起replace动作
  - M态写失效时，引起replace和BusWB



MESI State Transition Diagram – contd

- **Observing a BusRd**
  - 该块的状态从E降至 S
    - 因为存在其他copy
  - 该块从*M降至*S
    - 将引起更新过的块刷新操作
    - 刷新内存和其他有需求的Cache
- **Observing a BusRdX or BusUpgr**
  - 将作废相应的 block
  - 对于处于modified状态的块，将产生flush事务
  - BusUpgr：仅引起作废其他块，不产生读块操作
- **Cache-to-Cache (C2C) Sharing**
  - 原来的Illinois version支持这种共享
  - 由Cache 提供数据，而不是由内存提供数据



MESI Lower-level Design Choices

- **在E或S态时，由谁为BusRd/BusRdx事务提供数据**
  - Original, Illinois MESI: cache, 因为它假设Cache比memory更快
- **但 cache-to-cache 共享增加了实现的复杂性**
  - 这种实现的代价高于从memory获取数据
  - 存储器如何知道它该提供数据 (must wait for caches)
  - 如果多个Cache共享数据，要有Selection过程
- **当块状态为Modified，总线上的刷新（Flushing）数据操作**
  - 需要更新的块以及存储器 接收数据，但存储器速度比Cache的速度慢。
  - 是否可以仅让Cache接收数据，memory不接收数据?
    - 这就要求第5个状态: Owned state ⇒ MOESI Protocol
    - Owned 态是共享的Modified态，此时存储器不是最新数据
    - 该块可以被多个Cache共享，但所有者（owner)只有一个

MOESI中的Owned 和Shared 状态

- **Owned位。**
  - O位为1表示在当前Cache 块中包含的数据是当前处理器系统最新的数据拷贝，而且在其他CPU中一定具有该Cache块的副本，其他CPU的Cache块状态为S。
  - 如果存储器的数据在多个CPU的Cache中都具有副本时，有且仅有一个CPU的Cache块状态为O，其他CPU的Cache块状态只能为S。
  - 与MESI协议中的S状态不同，状态为O的Cache块中的数据与存储器中的数据并不一致。
- **Shared位。**
  - 当Cache块状态为S时，其包含的数据并不一定与存储器一致。
  - 如果在其他CPU的Cache中不存在状态为O的副本时，该Cache块中的数据与存储器一致；
  - 如果在其他CPU的Cache中存在状态为O的副本时，Cache块中的数据与存储器不一致。

Performance of Symmetric Shared-Memory Multiprocessors

- **Cache performance 由两部分构成:**
  - 单处理器 cache miss的通信量（Traffic）
  - 通信引起的通信量：由于作废机制导致后面的访问失效
- **Coherence misses**
  - 有时也称为 Communication miss
  - 4th C of cache misses along with Compulsory, Capacity, & Conflict.

Coherency Misses

- **由于对共享块的写操作引起**
  - 共享块在多个本地Cache有副本
  - 当某处理器对共享块进行写操作时会 作废其他处理器的本地Cache的副本
  - **其他处理器对共享块进行读操作时 会有**coherence miss
- **True sharing misses： Cache coherence 机制引起的数据通信**
  - 通常是不同的处理器写或读 同一个变量
  - 对S态块的写操作会作废其他cache中的共享块
  - 处理器试图读一个存在于其他处理器的cache中并且已经修改过的字（modified），这会导致失效，并将当前cache中的对应块写回
  - **即使块大小为1个字，失效仍然会发生**
- **False sharing misses： 由于某个字在某个失效块中**
  - 读写同一块中的不同变量
  - 失效并没有通过通信产生新的值，仅仅是产生了额外的失效
  - 块是共享的，但块中没有真正共享的字
    ⇒**如果块的大小为1个字，那么就不会产生这种失效**

Example of True & False Sharing Misses

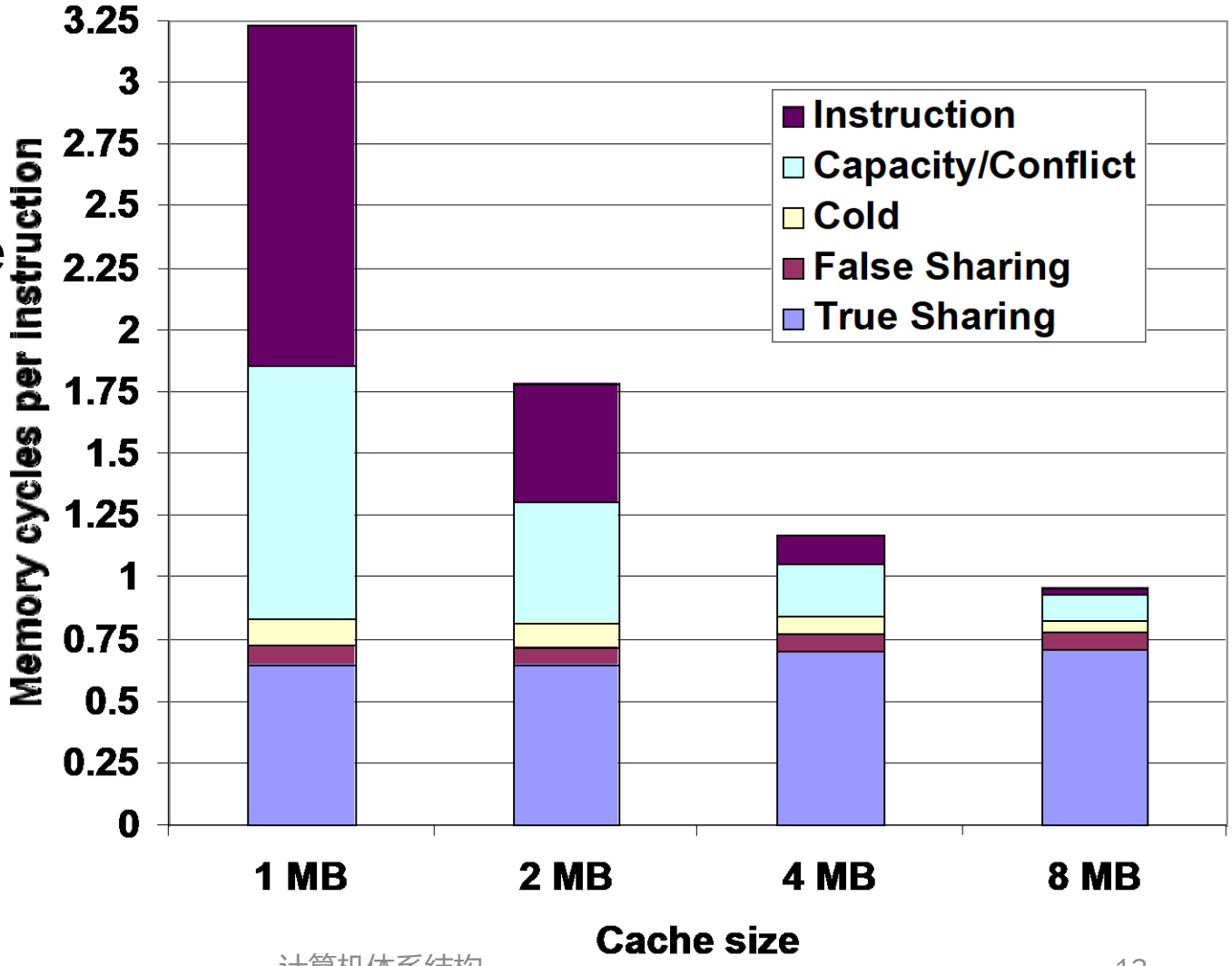
变量X和Y 属于同一cache块

初始状态为： P1和P2均读取了共享变量X, Block(X, Y) 在P1, P2中处于Shared 态

Request	P1 Cache State	P2 Cache State	Explanation
P1: Write X	Shared (X , Y)	Shared (X , Y)	True Sharing Miss (P2 read X)
	Modified (X , Y)	Invalid (X , Y)	P1 invalidates block (X , Y) in P2
P2: Read Y	Modified (X , Y)	Invalid (X , Y)	False Sharing Miss (Y not modified)
	Shared (X , Y)	Shared (X , Y)	Write-Back & Copy block from P1 to P2
P1: Write X	Shared (X , Y)	Shared (X , Y)	False Sharing Miss (P2 did not read X)
	Modified (X , Y)	Invalid (X , Y)	P1 invalidates block (X , Y) in P2
P2: Write Y	Modified (X , Y)	Invalid (X , Y)	False Sharing Miss (P1 did not read Y)
	Invalid (X , Y)	Modified (X , Y)	Write-Back & Copy block from P1 to P2
P1: Read Y	Invalid (X , Y)	Modified (X , Y)	True Sharing Miss (P2 modified Y)
	Shared (X , Y)	Shared (X , Y)	Write-Back & Copy block from P2 to P1

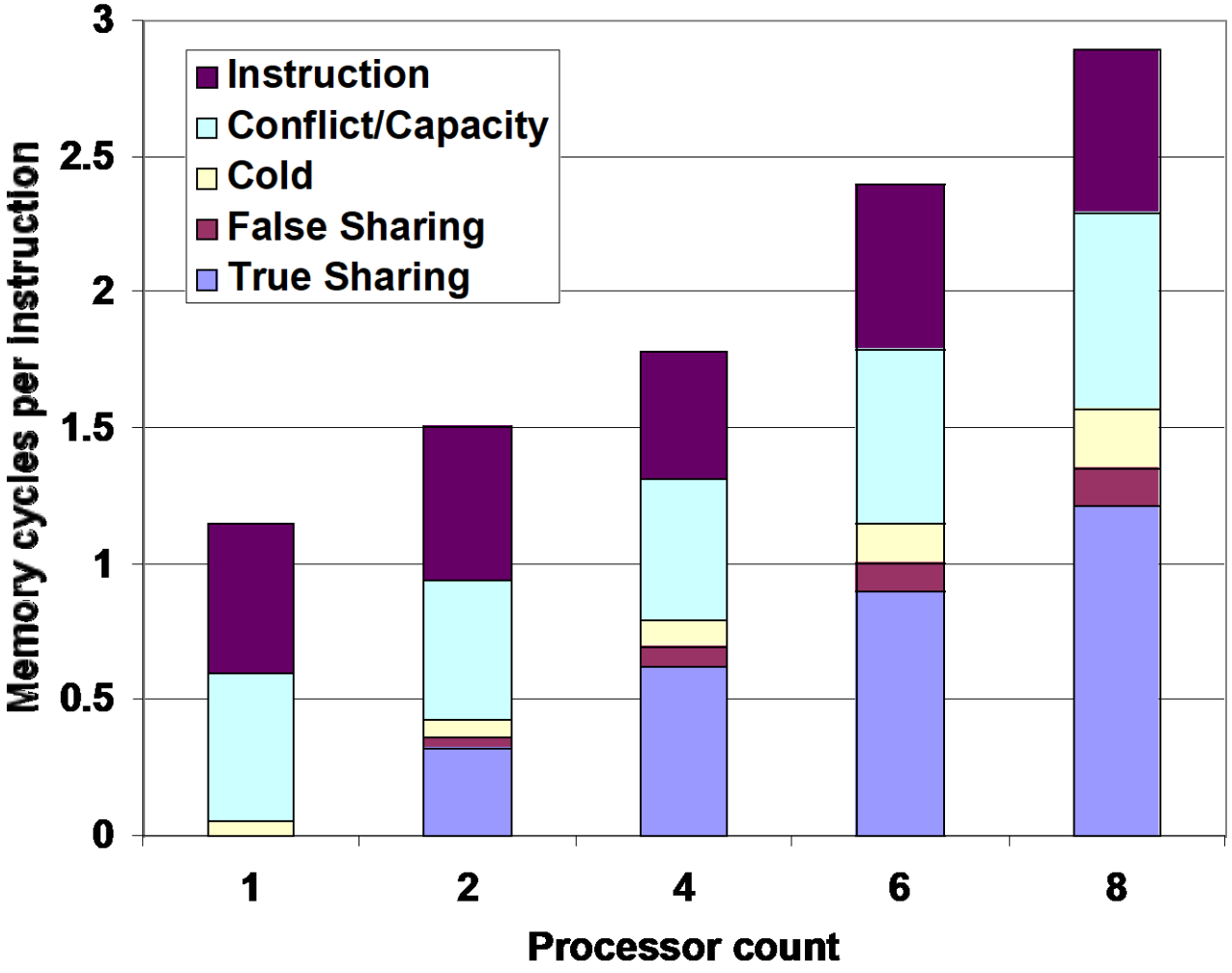
MP Performance 4 Processor Commercial Workload: OLTP, Decision Support (Database), Search Engine

- Uniprocessor cache misses improve with cache size increase (Instruction, Capacity/Conflict, Compulsory)
- True sharing and false sharing unchanged going from 1 MB to 8 MB (L3 cache)



MP Performance 2MB Cache
Commercial Workload: OLTP, Decision Support (Database), Search Engine

- True sharing, false sharing increase going from 1 to 8 CPUs



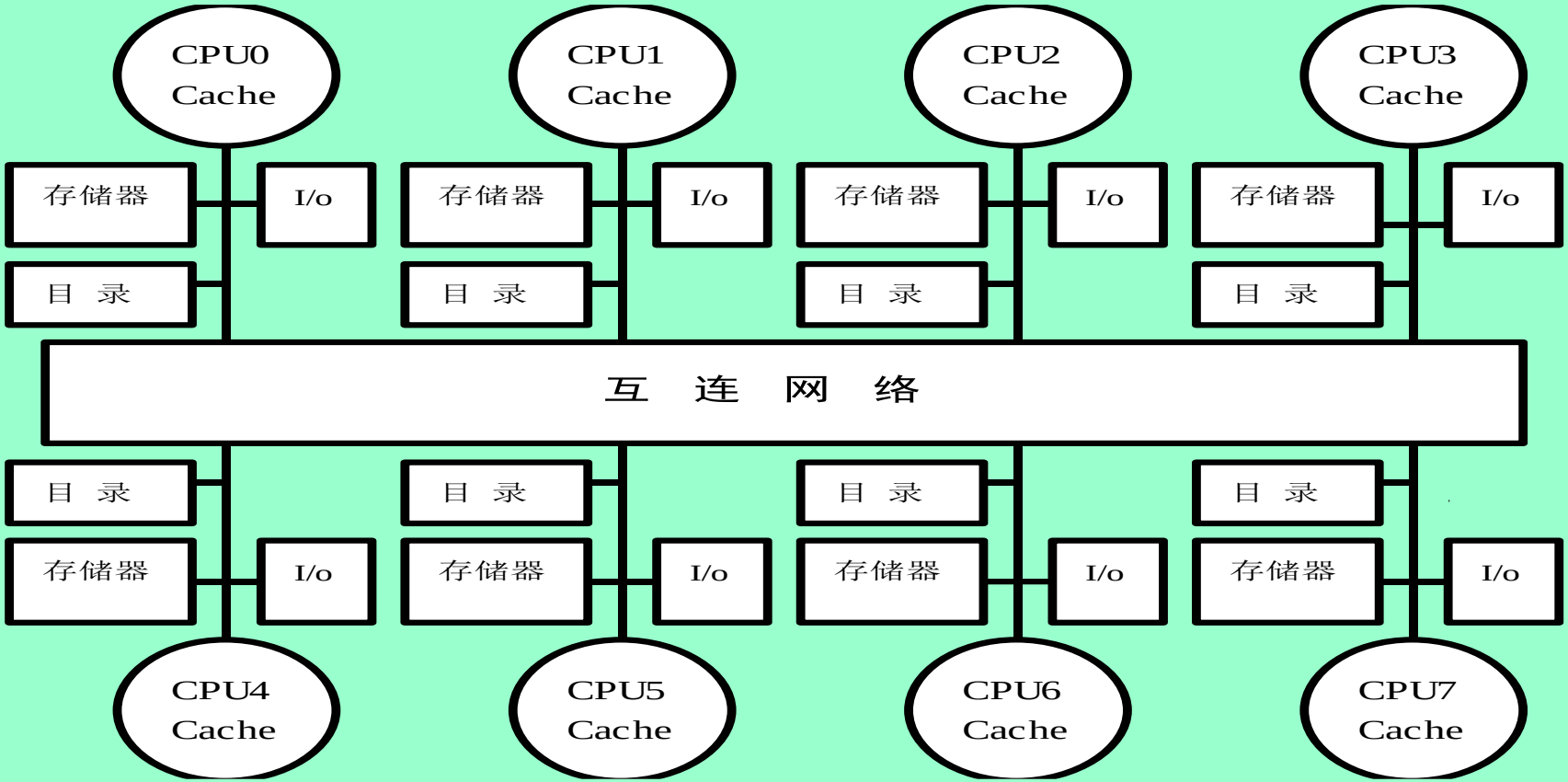
Limitations of Snooping Protocols

- **总线的可扩展性受到一定限制**
  - 总线上能够连接的处理器数目有限
  - 共享总线存在竞争使用问题
  - 在由大量处理器构成的多处理器系统中，监听带宽是瓶颈
- **解决方案之一：片上互连网络→并行通信**
  - 多个处理器可并行访问共享的Cache banks
  - 允许片上多处理器包含有更多的处理器
  - 可扩展性仍然受到限制。
- **在非总线或环的网络上监听是比较困难的**
  - 必须将一致性相关信息广播到所有处理器，这是比较低效的
- **如何不采用广播方式而保持 cache coherence**
  - 使用目录（directory)来记录每个 Cached 块的状态
  - 目录项说明了哪个私有Cache包含了该块的副本

解决Cache一致性问题的关键

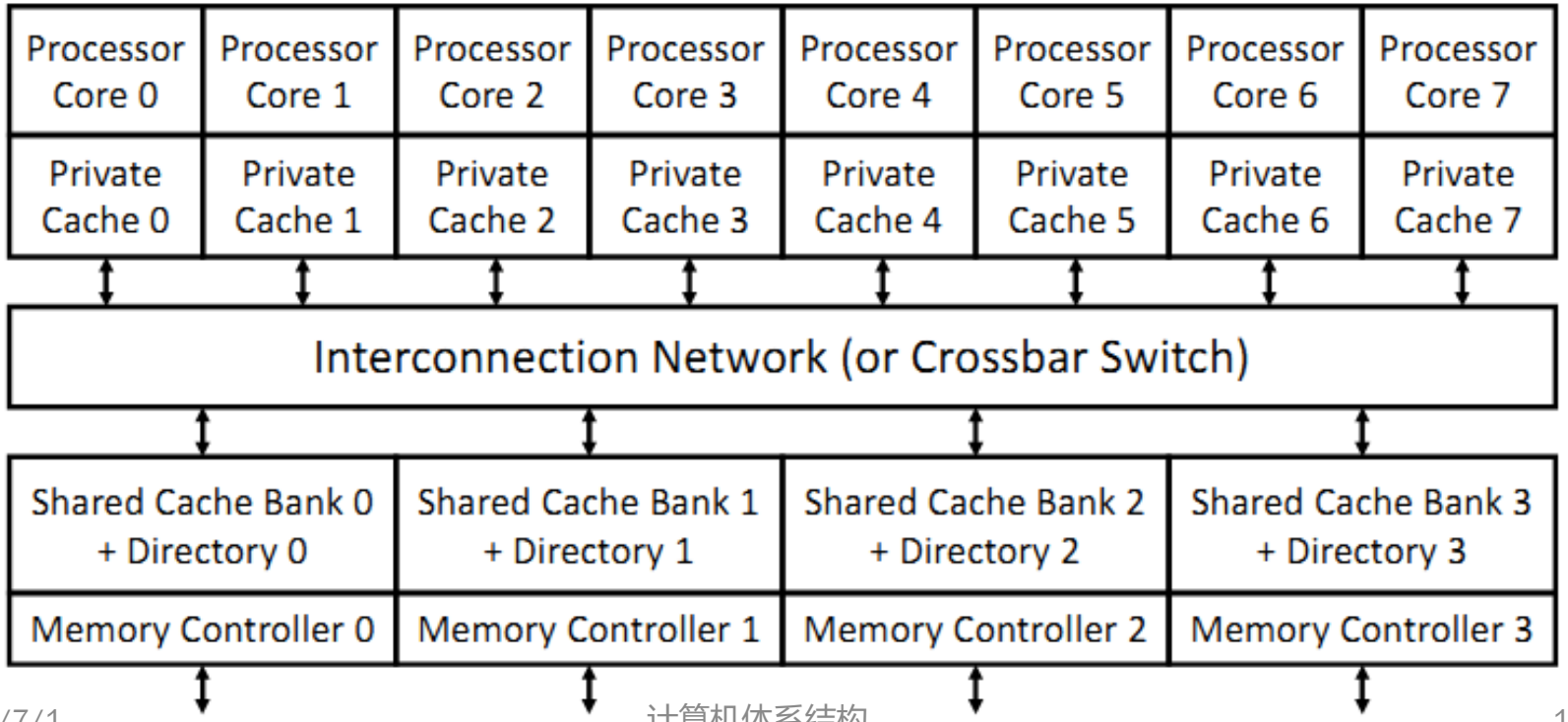
- **寻找替代监听协议的一致性协议。**
- **目录协议**
  - 目录：用于记录共享块相关信息的数据结构，它记录着可以进入Cache的每个数据块的访问状态、该块在各个处理器的共享状态以及是否修改过等信息。
- **对每个结点增加目录表后的分布式存储器的系统结构**

分布式共享存储结构


CPU0 Cache
CPU1 Cache
CPU2 Cache
CPU3 Cache
存储器
I/o
目 录
互 连 网 络
CPU4 Cache
CPU5 Cache
CPU6 Cache
CPU7 Cache

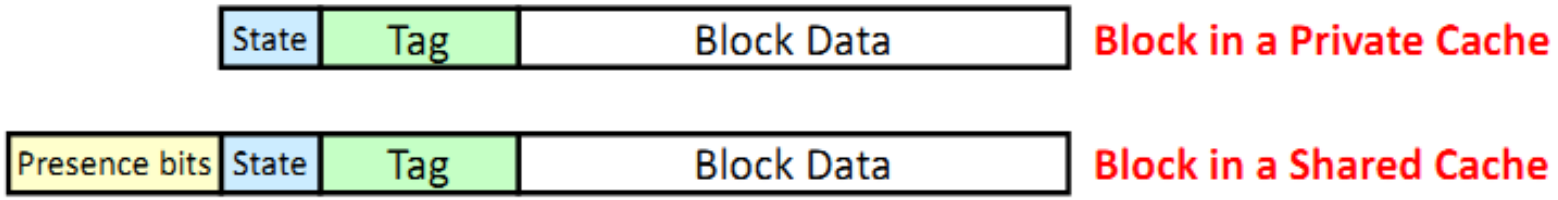
Directory in a Chip Multiprocessor

- **目录在所有处理器共享的最外层Cache中**
  - 目录记录了每个私有Cache中块的相关信息
- **最外层Cache分成若干个banks，以便并行访问**
  - Cache的banks数可以与cores的数量相同，也可以不同



Directory in the Shared Cache

- **Shared Cache 包含所有的私有Cache**
  - 共享Cache是私有cache块的超集
  - Example: Intel Core i7
- **目录在共享cache中**
  - 共享cache中的每个块增加若干presence bits
  - 如果有k个processors那么共享cache中每个块含有presence bits(k位) + state位
  - Presence bits 指示了包含该块copy的cores
  - 每个块都有其在私有cache和共享cache中的状态信息
  - State = M (Modified), S (Shared), or I (Invalid) in private cache



一些术语

- **本地或私有Cache （Local (or Private) Cache)**
  - 处理器请求的源
- **目录（Home Directory)**
  - 存放Cache块相关信息
  - 目录使用presence bits 和 state 追踪cache块
- **远程Cache（Remote Cache)**
  - 该Cache中包含一个Cache块的副本，处于modified 或shared 态
- **Cache一致性：即要保证Single-Writer, Multiple-Readers**
  - 如果一个块在本地Cache中处于Modified态，那么只有一个有效的副本存在（共享的Cache和存储器还没有更新）
- **无总线，不用广播方式到所有处理器核**
  - 所有消息都有显式的回复

States for Local and Shared Cache

对于本地（私有）cache 块，存在3种状态:

1. Modified: 仅当前Cache具有该块修改过的副本

2. Shared: 该块可能在多个Cache中有副本

3. Invalid: 该块无效

对于共享Cache中的块，存在4种状态:

1. Modified: 只有一个本地Cache是这个块的拥有者

只有一个本地Cache具有该块修改后的副本

2. Owned: 共享Cache是modified块的拥有者

Modified block被写回到共享Cache，但不是内存

处于owned态的块可以被多个本地Cache共享

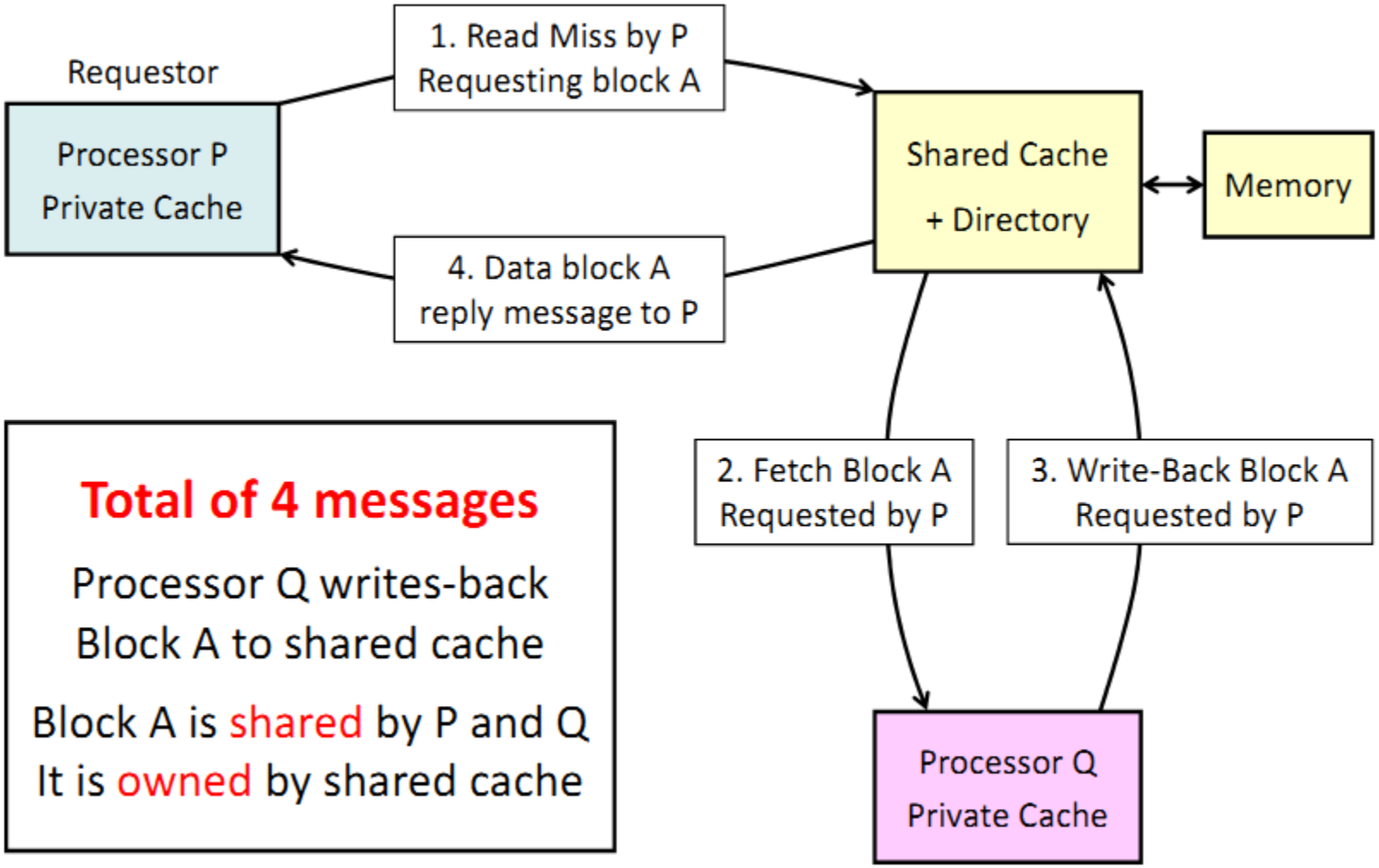
3. Shared: 该块可能被复制到多个cache中

4. Uncached: 该块不在任何本地或共享Cache中

Read Miss by Processor P

- **Processor P 发送 Read Miss 消息给 Home directory**
- **Home Directory: block 是 Modified态**
  - Directory 发送 Fetch message 给拥有该块的remote cache
  - Remote cache发送 Write-Back message 到 directory (shared cache)
  - Remote cache 将该块状态修改为shared
  - Directory 将其所对应的共享块状态修改为 owned
  - Directory 发送数据给P, 并将对应于P的presence bit置位
  - P的Local cache 将所接收到的块状态置为 shared
- **Home Directory: block 是Shared or Owned态**
  - Directory发送数据给P，并将对应 P的presence bit置位
  - P的Local cache 将所接收到的块状态置为 shared
- **Home Directory: Uncached -> 从存储器中获取块**

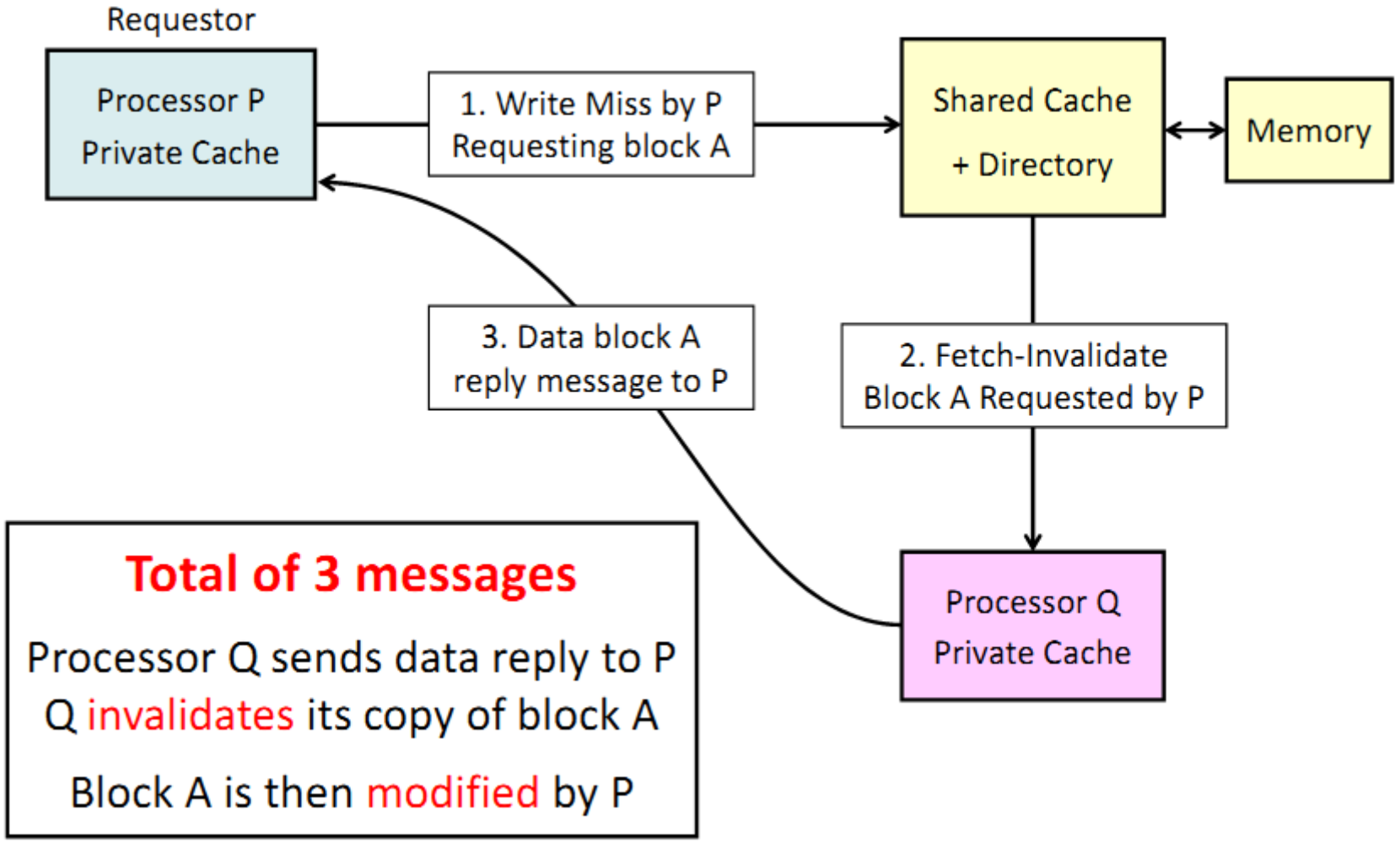
Read Miss to a Block in Modified State



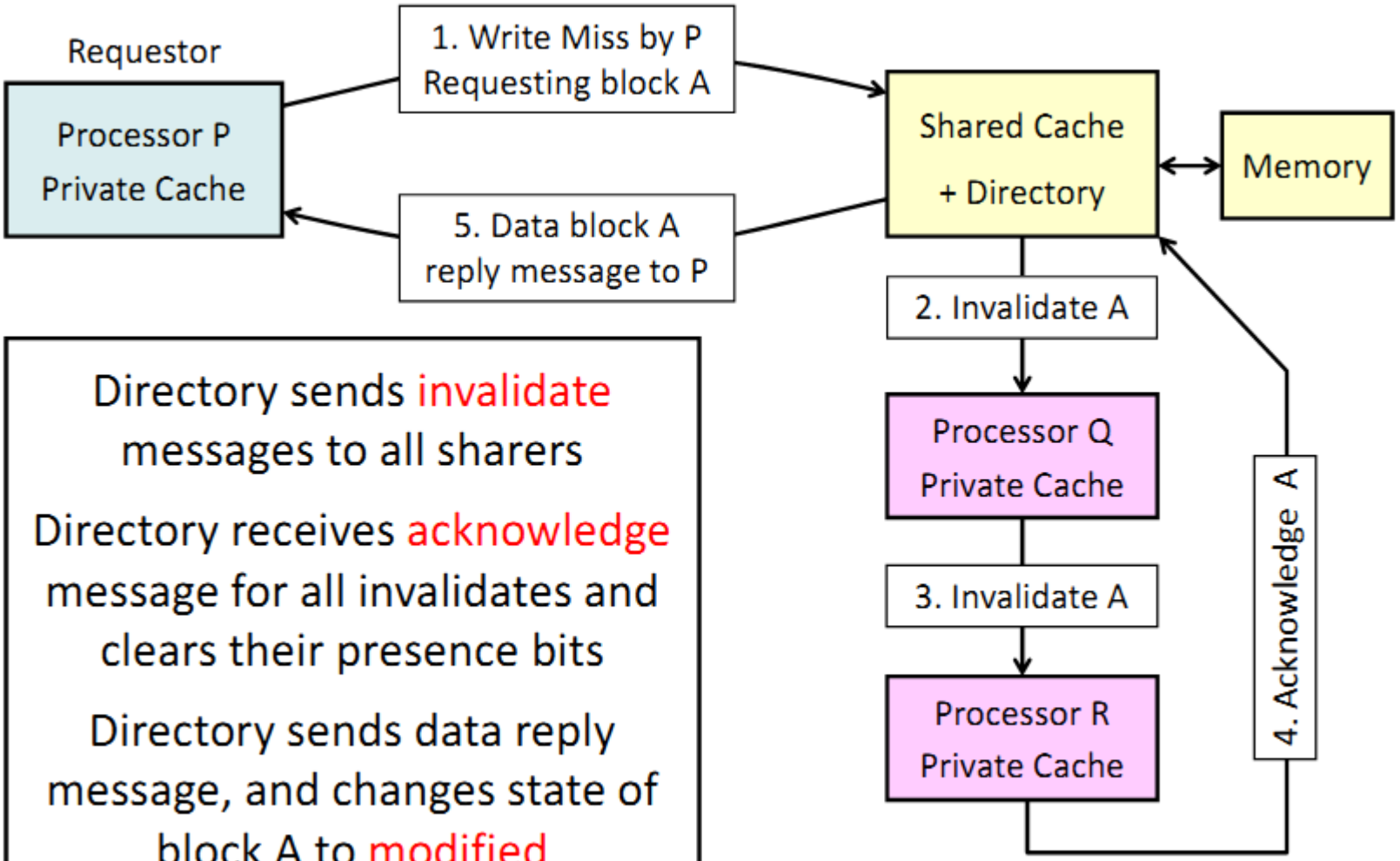
Write Miss Message by P to Directory

- **Home Directory: block 是Modified态**
  - Directory 发送 Fetch-Invalidate message 给处理器Q的Cache (Remote Cache)
  - 处理器Q的cache 直接发送数据应答消息给P
  - Q的cache将对应块的状态修改为invalid
  - P的cache (Local) 将接收到的块的状态信息修改为modified
  - Directory 将对应于Q的 presence bit复位，并将对应于P的 presence bit 置位
- **Home Directory: block 是 Shared or Owned态**
  - Directory 根据presence bit位给所有的共享者发送invalidate messages
  - Directory接收 acknowledge消息并将对应的presence bits复位
  - Directory 发送数据回复信息给P, 并将P对应的 presence bit 置位
  - P的cache 和directory 将该块的状态修改为 modified
- **Home Directory: Uncached -> 从存储器获取数据**

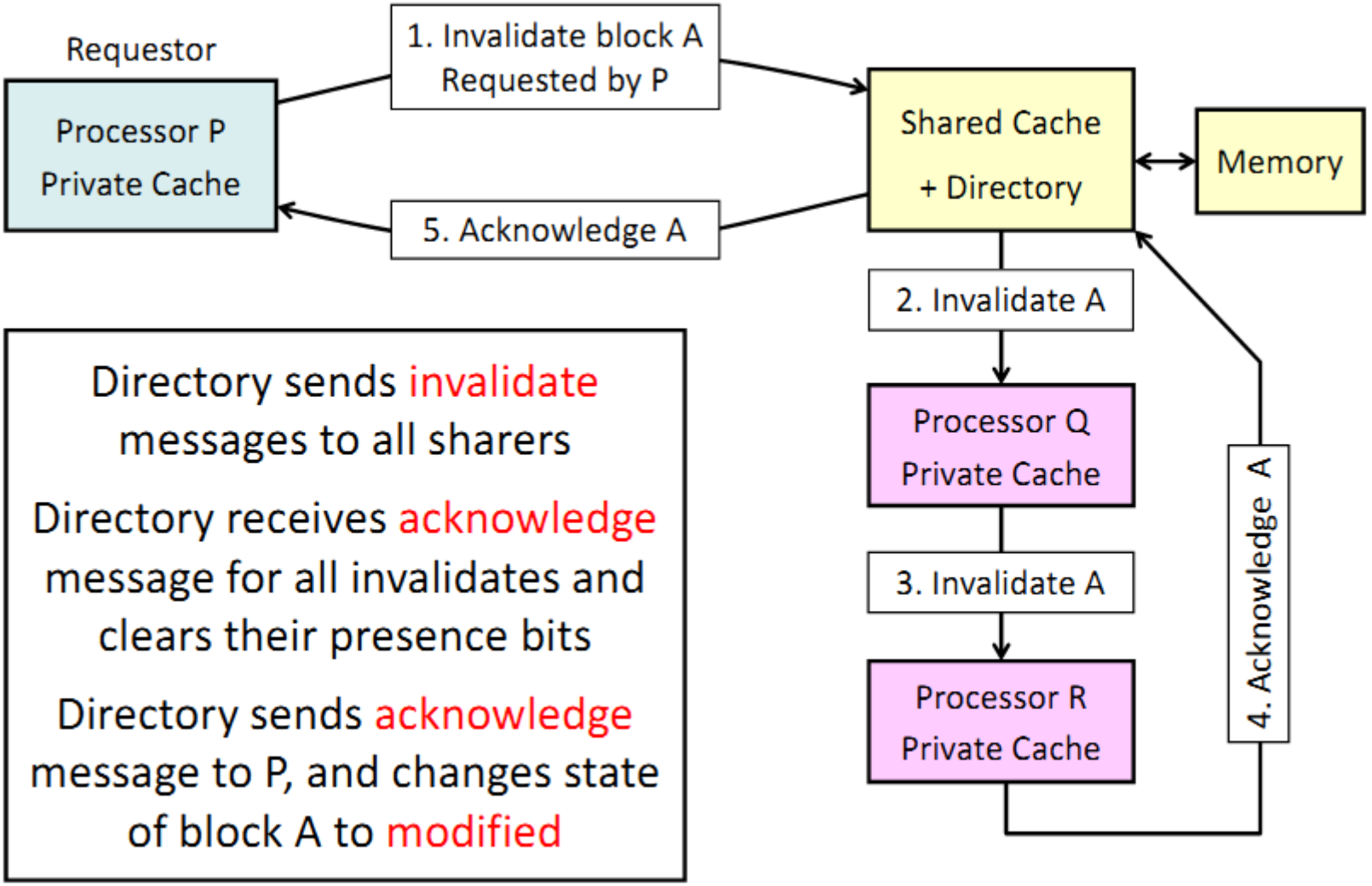
Write Miss to a Block in Modified State



Write Miss to a Block with Sharers



Invalidating a Block with Sharers

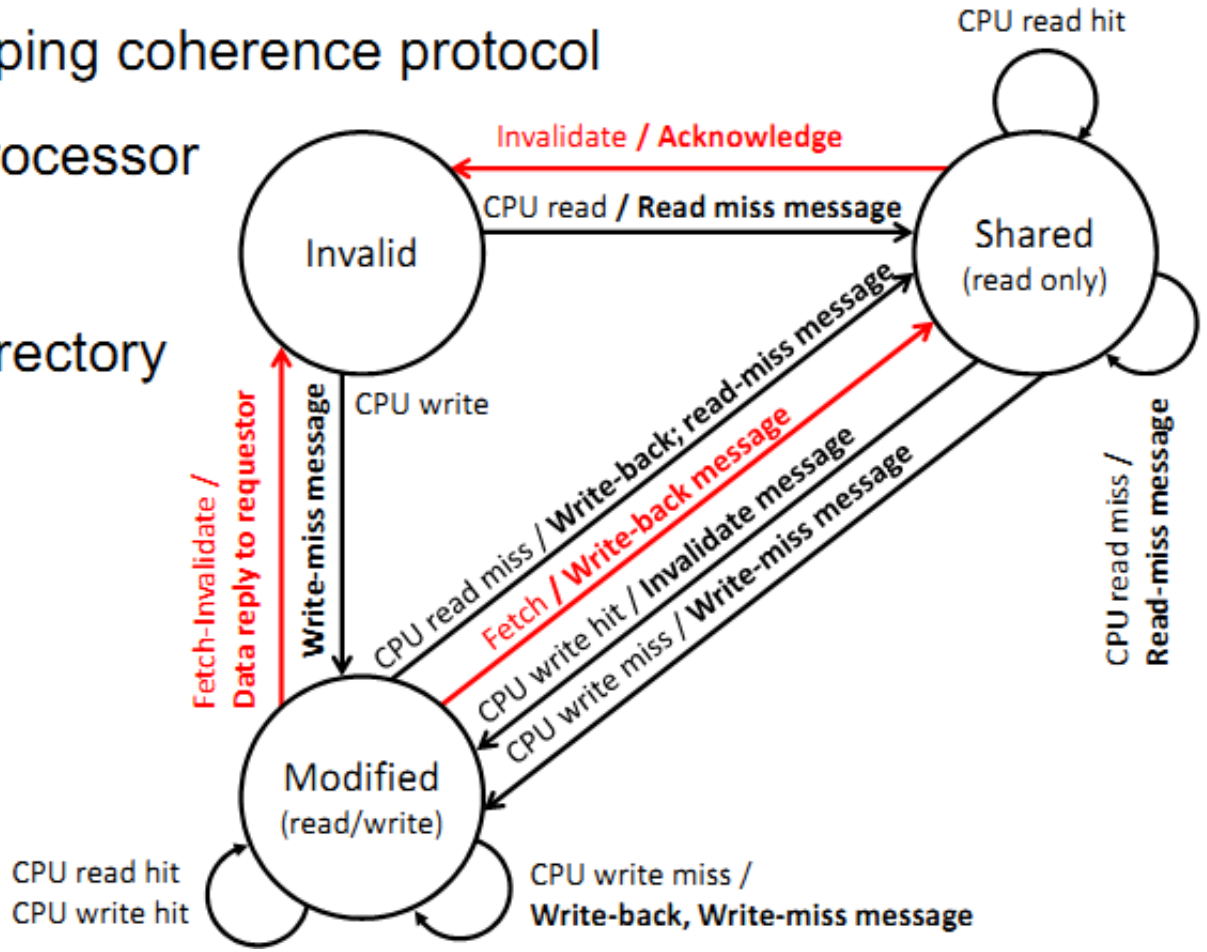


Directory Protocol Messages

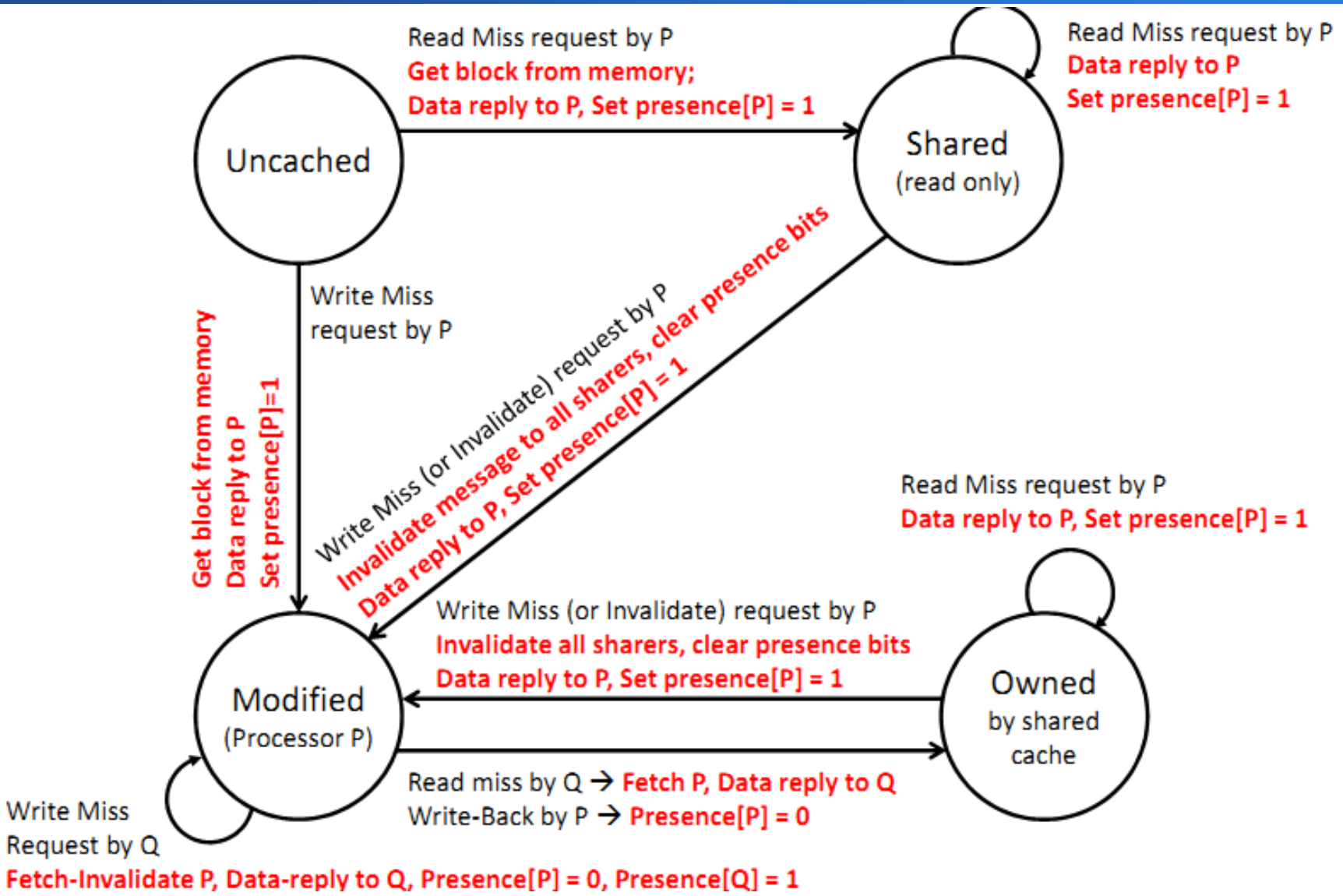
Message Type	Source	Destination	Message Function
Read Miss	Local Cache	Home Directory	Processor P has a read miss at address A Request data and make P a read sharer
Write Miss	Local Cache	Home Directory	Processor P has a write miss at address A Request data and make P the exclusive owner
Invalidate	Local Cache	Home Directory	Processor P wants to invalidate all copies of the same block at address A in all remote caches
Invalidate	Home Directory	Remote Caches	Directory sends invalidate message to all remote caches to invalidate shared block at address A
Acknowledge	Remote Cache	Home Directory	Remote cache sends an acknowledgement message back to home directory after invalidating last shared block A
Acknowledge	Home Directory	Local Cache	Directory sends acknowledgment message back to local cache of P after invalidating all shared copies of block A
Fetch	Home Directory	Remote Cache	Directory sends a fetch message to a remote cache to fetch block A and to change its state to shared
Fetch & Invalidate	Home Directory	Remote Cache	Directory sends message to a remote cache to fetch block A and to change its state to invalid
Data Block Reply	Directory or Cache	Local Cache	Directory or remote cache sends data block reply message to local cache of processor P that requested data block A
Data Block Write Back	Remote Cache	Home Directory	Remote Cache sends a write-back message to home directory containing data block A

MSI State Diagram for a Local Cache

- Three states for a cache block in a local (private) cache
- Similar to snooping coherence protocol
- Requests by processor
  - **Black arrows**
- Requests by directory
  - **Red arrows**

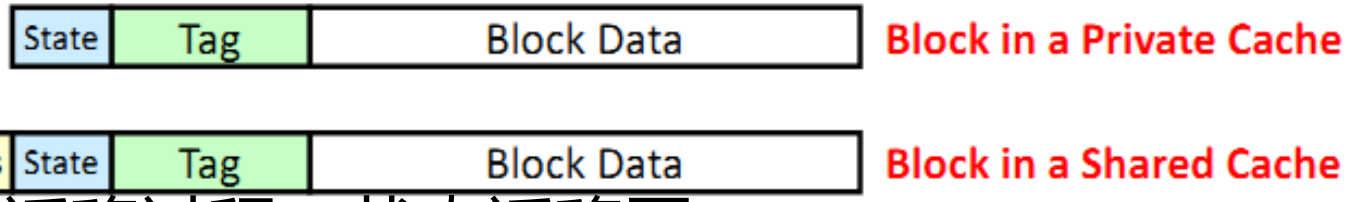


MOSI State Diagram for Directory



-Review

- **分布式共享存储的Cache一致性协议**
  - Cache块的状态：
    - 私有Cache中块的状态/目录中Cache块的状态



  - 状态迁移过程：状态迁移图
- **存储同一性问题**
  - Consistency研究不同处理器访问存储器操作的定序问题，即所有处理器发出的所有访问存储器操作(所有地址）的全序
  - Coherence研究不同处理器访问存储器相同地址操作的定序问题，即访问每个Cache块的局部序问题

7.4 Models of Memory Consistency

- **什么是存储同一性?**
- **隐式存储同一性模型（顺序存储同一性）**
- **放松的存储同一性**

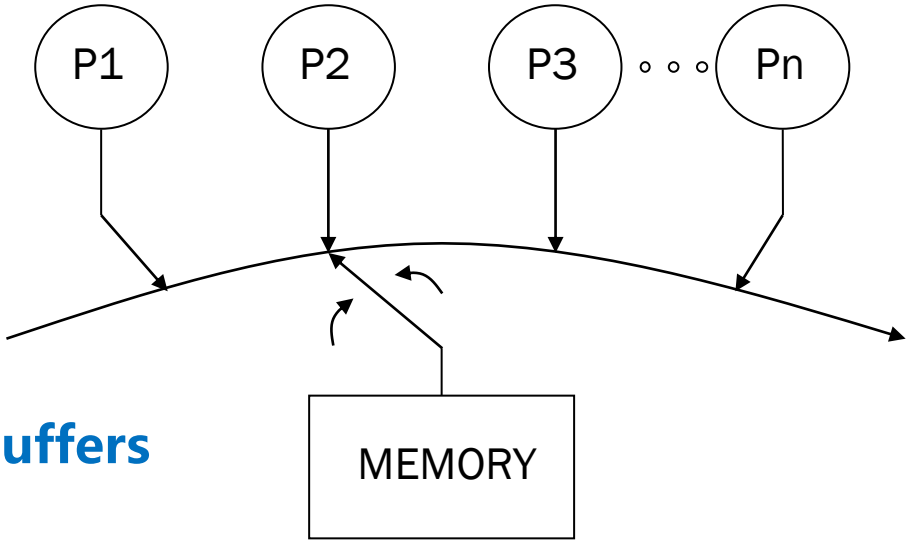
存储同一性的定义

- **定义: 共享地址空间的存储同一性模型是在多个处理器对不同存储单元并发读写操作时，每个进程看到的这些操作被完成的序的一种约定。**
  - 存储一致性保证的是当对共享存储空间中的某一单元修改后，对所有读取者是可见的。即每个单元都能“返回最后一次写操作的值”
  - 没有明确所写入的数据何时成为可见的
  - 一致性协议没有涉及处理器P1和P2对不同地址单元的访问顺序
  - 一致性协议没有涉及到P2对不同存储单元的读操作相对于P1所见到的顺序

```
P1              P2    (A, flag are zero initial)
A=1             while(flag == 0);
flag=1                  print A;
```

Implicit Memory Model

- **顺序同一性(Sequential Consistency) [Lamport]：该模型要求所有处理器的读、写和交换(swap)操作以某种序执行所形成的全局存储器次序，符合各处理器的原有程序次序。即： 不论指令流如何交叠执行，全局次序必须保持所有进程的程序次序**
  - 所有读写操作执行以某种顺序执行
  - 每一进程的操作以程序序执行



- **No caches, no write buffers**

Understanding Program Order – Example 1

- Initially X = 2

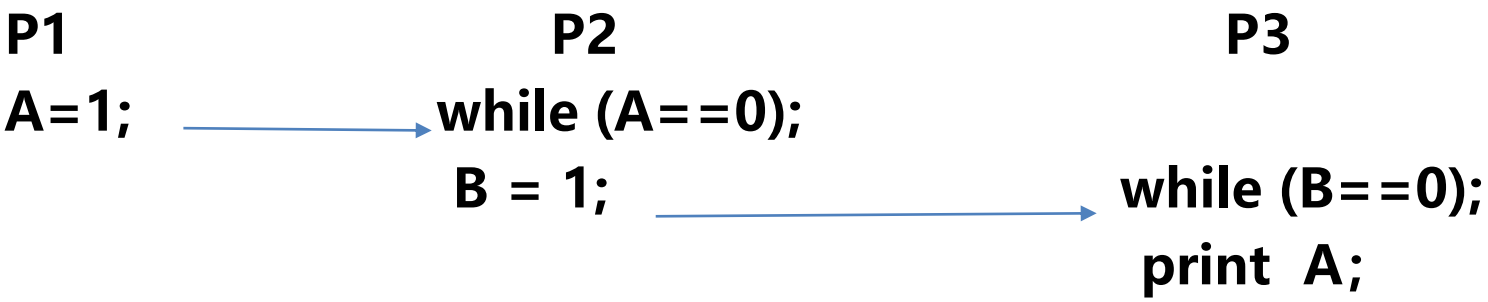
P1	P2
.....	.....
r0=Read(X)	r1=Read(x)
r0=r0+1	r1=r1+1
Write(r0,X)	Write(r1,X)
.....	.....

- Possible execution sequences:

```
P1:r0=Read(X)
P2:r1=Read(X)
P1:r0=r0+1
P1:Write(r0,X)
P2:r1=r1+1
P2:Write(r1,X)
    x=3
```

```
P2:r1=Read(X)
P2:r1=r1+1
P2:Write(r1,X)
P1:r0=Read(X)
P1:r0=r0+1
P1:Write(r0,X)
    x=4
```

Understanding Program order-Example 2

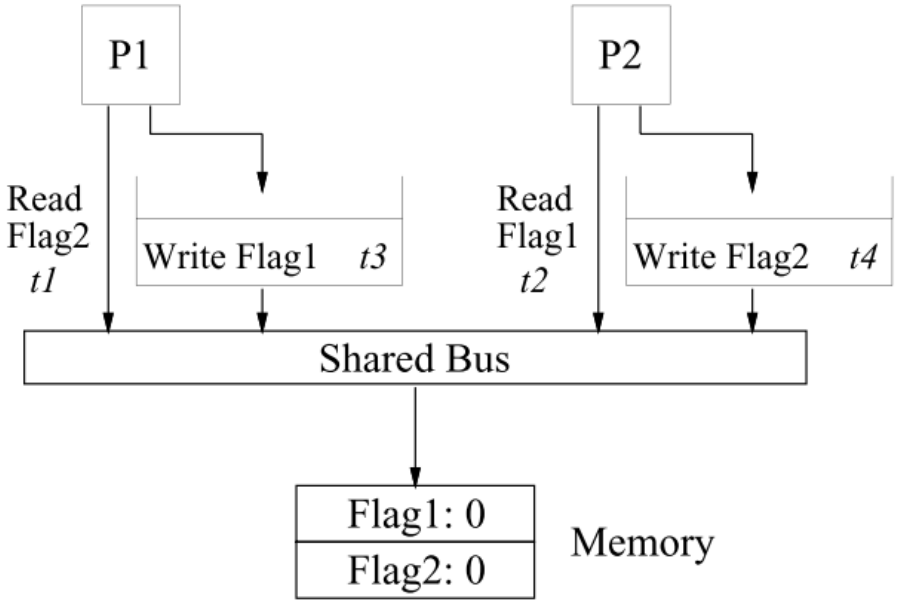


假设A,B的初始值为0；

从程序员角度看；P3应该输出 A=1；

如果P2被允许越过对变量A的读操作，在P3看见A的新值前对B进行写操作，那么P3就可能读出B的新值和A的旧值（例如从cache)，这种情况就不满足顺序同一性要求。

Optimization 1: Write Buffers with Bypassing Capability

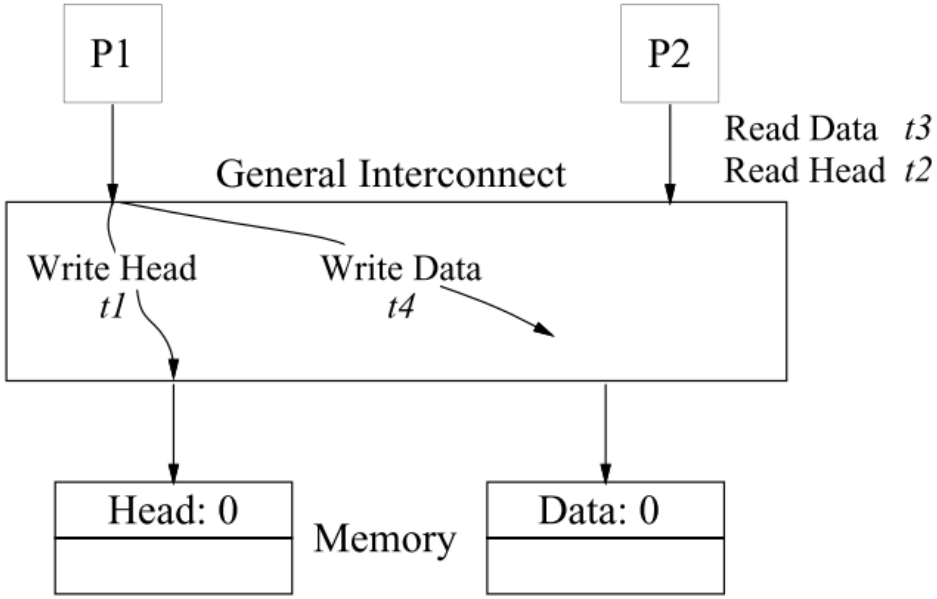


P1	P2
Flag1 = 1	Flag2 = 1
if (Flag2 == 0)	if (Flag1 == 0)
critical section	*critical section*

(a) write buffer

- **Flag1和Flag2的新值都在write buffer中**
- **导致存储器操作的序与程序序不同，违反SC规则，P1和P2可同时进入临界区**

Optimization 2: Overlapping Write Operations

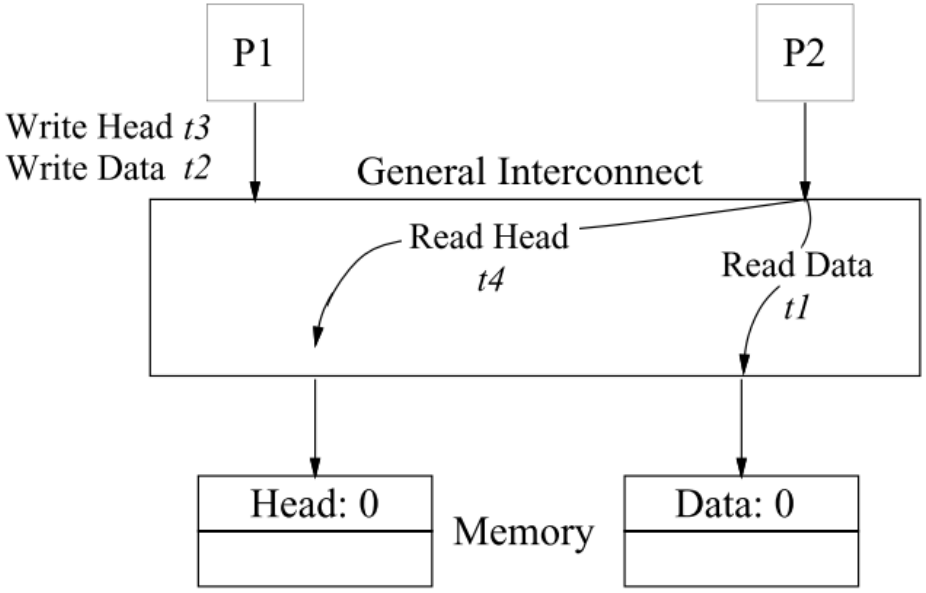


P1	P2
Data = 2000	while (Head == 0) {;}
Head = 1	... = Data

(b) overlapped writes

- **非总线互联网络：避免总线的性能瓶颈**
- **多存储器模块：具有并行读写特性，提高读写性能**
  - 导致write Data 与 Write Head的完成序 与 程序序 相反
  - 进而 导致 P2 首先读到Head的新值，Data 的旧值，违反SC 规则

Optimization 3: Non-blocking reads



P1	P2
Data = 2000	while (Head == 0) {;}
Head = 1	... = Data

(c) non−blocking reads

- **假设P1写操作按照程序序执行存储器操作，P2允许以overlapped 的方式执行读操作 （non-blocking read, speculative execution, and dynamic scheduling)**
- **则：可能会产生P2 Read Data 提前于 P1的Write Data的情况，导致违反SC规则**

多处理器操作的困难

- **大多数并行计算机体系结构研究有关：**
  - 如何克服顺序执行和并行执行的瓶颈，以得到更高的性能和效率
  - 如何为用户提供良好的编程模型，以便编写正确而高性能的并行程序
- **操作的顺序问题**
  - Operations: A, B, C, D
    - 硬件以何种顺序执行（和报告结果）这些操作?
  - 程序员与微结构设计人员的协议由ISA来约定
  - 保留程序员所希望的执行顺序
    - 可降低编程的难度，如：易于 debugging; 易于状态恢复、异常处理等
    - 通常会使得硬件设计变得困难，特别是当我们的设计目标为高性能处理器时，乱序load-store的执行，使得问题变得复杂

单个处理器存储器操作的序

- **操作顺序由von Neumann 模型约定**
- **顺序串行执行**
  - 硬件执行load和store操作以程序序顺序执行
- **乱序执行不改变程序语义**
  - 硬件以程序序报告load和store操作的结果
- **优点 1) 在执行时机器状态是确定的。2)程序的不同次运行机器状态是一致的，有利于程序调试**
- **缺点: 维护这种序的额外开销，降低了性能，增加了复杂性，降低了可扩放性**

数据流处理器的存储器操作的序

- 当操作数准备好就可以执行存储器操作
- 操作的顺序仅仅由数据依赖性来确定
- 相互独立的操作可以以任意序执行和提交结果

- 优点: 并行度高，性能高
- 缺点: 相同程序的不同次运行次序可以不同，使得调试困难

MIMD处理器中的存储器操作序

- **每个处理器的存储器操作以顺序序执行对应于运行在该处理器上的一个线程（假设每个处理器符合von Neumann模型）**
- **多个处理器并发地执行存储器操作**
- **存储器如何看来自所有处理器的存储器操作?**
  - 即不同处理器发出的存储器操作,在共享存储器端看到的应该是什么序?

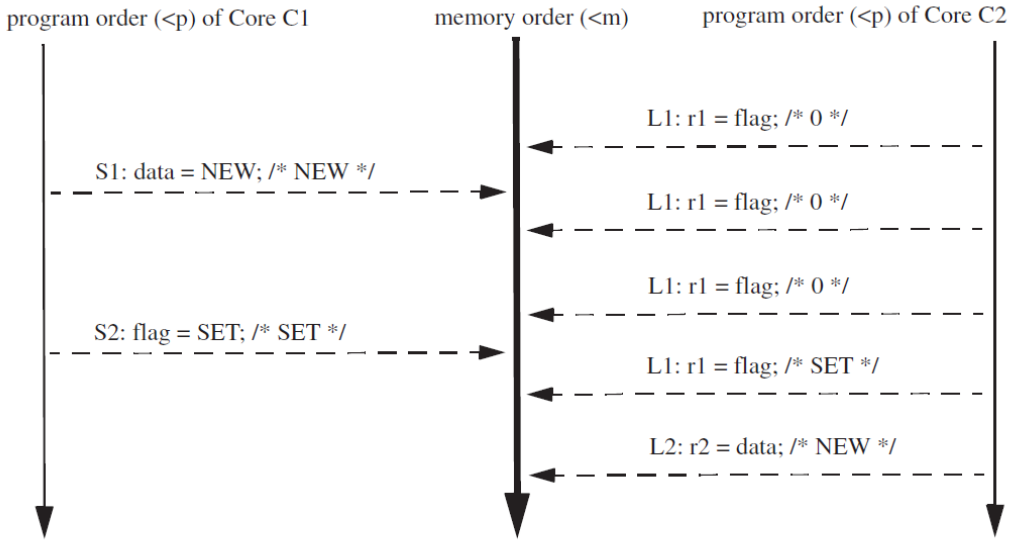


FIGURE 3.1: A Sequentially Consistent Execution of Table 3.1's Program.

序的重要性

- **易于调试**
  - 若程序的每次执行都是相同的序有利于程序的调试
- **正确性**
  - 从不同处理器看到的存储器操作序可能会不同
- **性能和代价权衡**
  - 强制符合严格 “sequential ordering”使得硬件设计人员实现性能增强技术变得十分复杂 (例如., OoO 执行, caches)
- **<p ： 程序序(program order)**
  <m： 存储器操作序(memory order)

Acknowledgements

- **These slides contain material developed and copyright by:**
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - David Patterson (UCB)
  - Chenxi Zhang (Tongji)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**