计算机体系结构

Cache Coherence

第7章　多处理器及线程级并行

7.1 引言
7.2 集中式共享存储器体系结构
7.3 分布式共享存储器体系结构
7.4 存储同一性
7.5 同步与通信

7.1、引言

- 单处理机的发展正在走向尽头?
- 并行计算机在未来将会发挥更大的作用
- 获得超过单处理器的性能，最直接的方法就是把多个处理器连在一起;
- 自1985年以来，体系结构的改进使性能迅速提高，但通过复杂度和硅技术的提高而得到的性能的提高正在减小;
- 并行计算机应用软件已有稳定的发展。

并行计算机体系结构的分类

1、按照Flynn分类法，可把计算机分成

- 单指令流单数据流（SISD）
- 单指令流多数据流（SIMD）
- 多指令流单数据流（MISD）
- 多指令流多数据流（MIMD）

2、MIMD已成为通用多处理机体系结构的选择，原因：

- MIMD具有灵活性。
- MIMD可以充分利用商品化微处理器在性能价格比方面的优势。

通信模型和存储器的结构模型

1. 两种地址空间的组织方案

(1)共享存储（多处理机）：

物理上分离的多个存储器可作为一个逻辑上共享的存储空间进行编址

(2)非共享存储（多计算机）：

整个地址空间由多个独立的地址空间构成，它们在逻辑上也是独立的，远程的处理器不能对其直接寻址。

每一个处理器-存储器模块实际上是一个单独的计算机，这种机器也称为多计算机。

两种通信模型

2. 两种通信模型

(1) 共享地址空间的机器

利用Load/Store指令的地址隐含地进行数据通讯

(2)多个地址空间的机器

通过处理器间显式地传递消息完成

这种机器常称为消息传递机器。

不同通信机制的优点

1、共享存储器通信的主要优点

- 当处理器通信方式复杂或程序执行动态变化时易于编程，同时在简化编译器设计方面也占有优势。
- 当通信数据较小时，通信开销较低，带宽利用较好。
- 通过硬件控制的Cache减少了远程通信的频度，减少了通信延迟以及对共享数据的访问冲突。

2、消息传递通信机制的主要优点

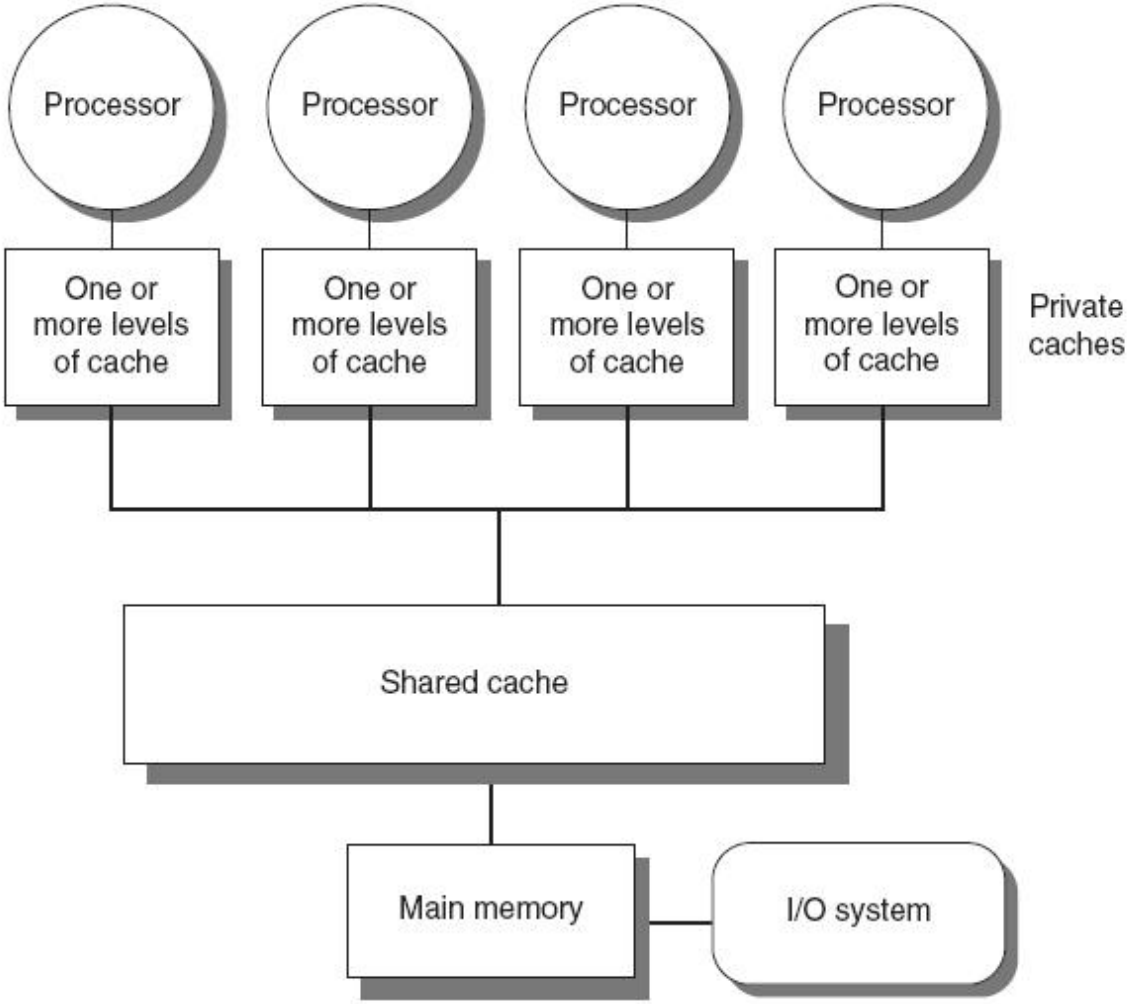
- 硬件较简单。
- 通信是显式的，从而引起编程者和编译程序的注意，着重处理开销大的通信。

基于共享存储的MIMD机器分类

根据多处理机系统中存储器组织以及处理器个数的多少，可分为两类

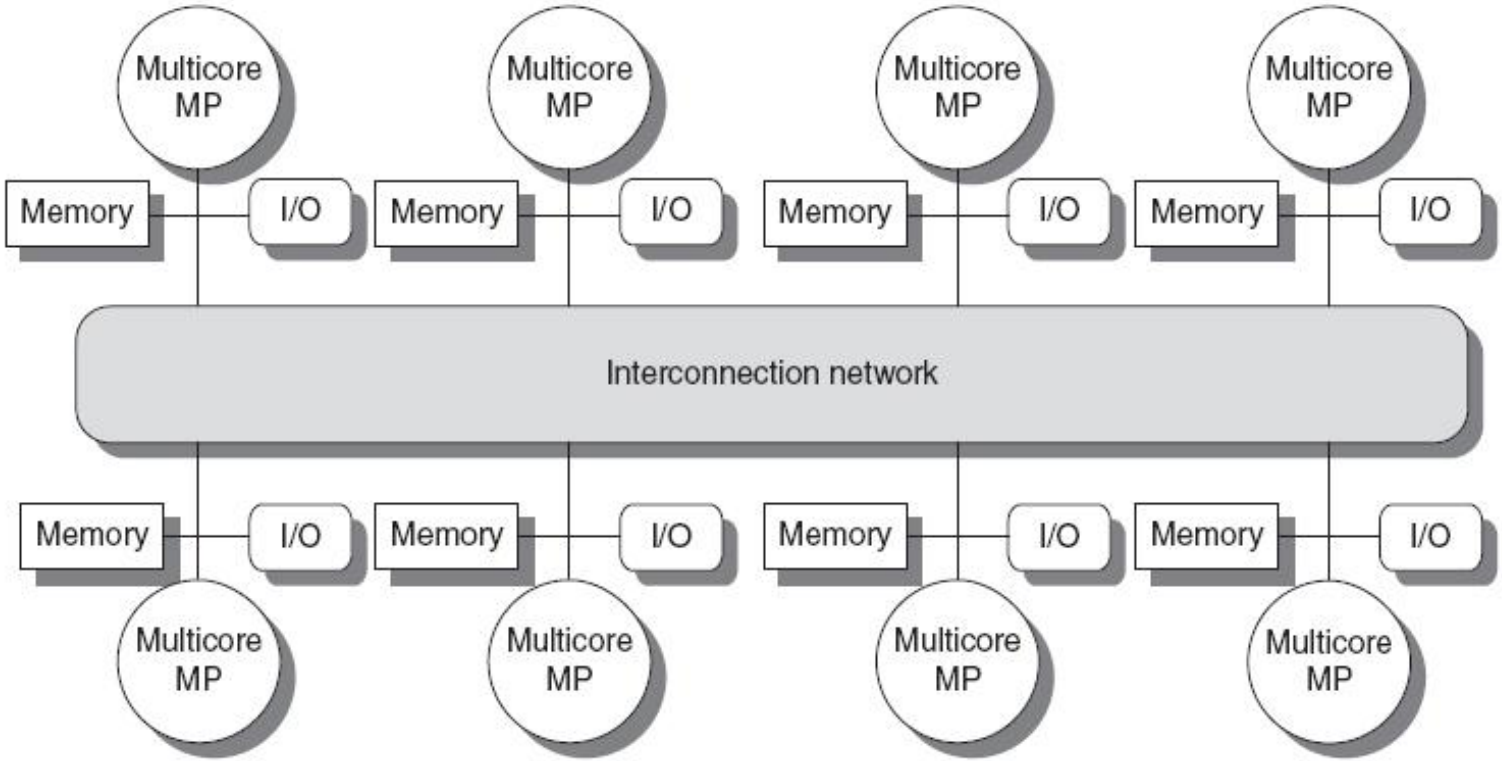
- **集中式共享存储器结构（SMP）：这类机器有时被称为UMA(uniform memory access)机器**
- **分布式共享存储器结构（DSM）：**
  - 这类机器的结构被称为分布式共享存储器(DSM)或可缩放共享存储器体系结构，DSM机器被称为NUMA(non-uniform memory access)机器
  - 每个结点包含：处理器、存储器、I/O
  - 在许多情况下，分布式存储器结构优于采用集中式共享存储器结构。
  - 分布式存储器结构需要高带宽的互连

集中式共享存储结构



集中式共享存储器结构（SMP）

分布式共享存储器结构（DSM）



分布式共享存储器结构（DSM）

并行处理面临的挑战

- **并行处理面临着两个重要的挑战和一个重要问题：**
  - 程序中有限的并行性
  - 相对较高的通信开销
  - 一个重要问题：存储器访问的序问题
- **挑战之一：有限的并行性使机器要达到好的加速比十分困难**
  - 例7.1: 如果想用100个处理器达到80的加速比，求原计算程序中串行部分所占比例。

例7.1

解　根据Amdahl定律加速比为：

$$\frac{1}{\frac{\text{可加速部分比例}}{\text{理论加速比}} + (1-\text{可加速部分比例})}$$

$$80 = \frac{1}{\frac{\text{并行比例}}{100} + (1-\text{并行比例})}$$

- 得出：并行比例 = 0.9975
- 可以看出要用100个处理器达到80的加速比，串行计算的部分只能占0.25%。

- **挑战之二：多处理机中远程访问的较大延迟。在现有的机器中，处理器存储器之间的数据通信大约需要35～>500个时钟周期。**
  - 同一芯片中core之间的延迟35~50cycles
  - 不同芯片间core之间的延迟100~>500 cycles

远程访问一个字的延迟时间

机　　器	通信机制	互连网络	处理机数量	典型远程存储器访问时间
SPARC Center	共享存储器	总线	≤20	1μs
SGI Challenge	共享存储器	总线	≤36	1μs
Cray T3D	共享存储器	3维环网	32－2048	1μs
Convex Exemplar	共享存储器	交叉开关＋环	8－64	2μs
KSR-1	共享存储器	多层次环	32－256	2-6μs
CM-5	消息传递	胖树	32－1024	10μs
Intel Paragon	消息传递	2维网格	32－2048	10-30μs
IBM SP-2	消息传递	多级开关	2－512	30-100μs

例7.2　一台32个处理器的计算机，对远程存储器访问时间为2000ns。除了通信以外，假设计算中的访问均命中局部存储器。当发出一个远程请求时，本处理器挂起。处理器时钟周期为10ns，如果指令基本的CPI为1.0(设所有访存均命中Cache)，求在没有远程访问的状态下与有0.5%的指令需要远程访问的状态下，前者比后者快多少?

解：　有0.5%远程访问的机器的实际CPI为：

CPI = 基本CPI + 远程访问率×远程访问开销

　　= 1.0 + 0.5%×远程访问开销

远程访问开销 = 远程访问时间/时钟时间

　　= 2000ns/10ns = 200个时钟周期

∴　　CPI = 1.0 + 0.5%×200 = 2.0

它为只有局部访问的机器的2.0 / 1.0 = 2倍，

因此在没有远程访问的状态下的机器速度是有0.5%远程访问的机器速度的2倍。

存储器访问的序问题

- **存储同一性（Consistency）：**
  - 不同处理器发出的**所有存储器操作**的顺序问题（即针对不同存储单元或相同存储单元）
  - 访问所有存储单元的**全序问题**
- **存储一致性（Coherence）：**
  - 不同处理器访问**相同存储单元**时的访问顺序问题
  - 访问每个Cache块的**局部序问题**

Sorin D, Hill M, Wood D. A Primer on Memory Consistency and Cache Coherence[J]. 2011, 6(3):212.

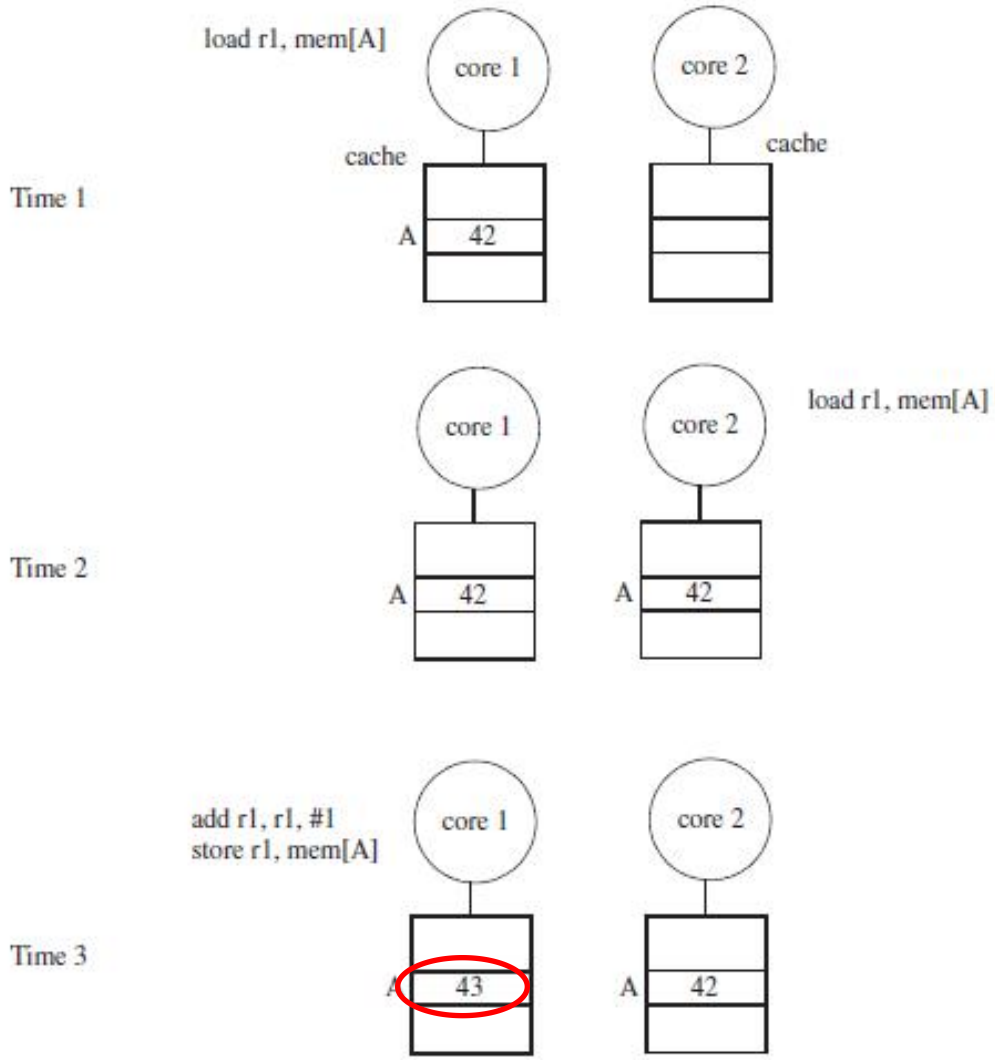
存储同一性（Memory Consistency）

TABLE 3.3: Can Both r1 and r2 be Set to 0?		
Core C1	**Core C2**	**Comments**
S1: x = NEW; L1: r1 = y;	S2: y = NEW; L2: r2 = x;	/* Initially, x = 0 & y = 0*/

可能的执行顺序（假设可全乱序|假设遵循SC Model）：

S1L1S2L2 (0,NEW)	**S2S1L1L2 (NEW, NEW)**	**L1S1S2L2 (0, NEW)**	**L2S1L1S2(0,0)**
S1L1L2S2 (0,NEW)	**S2S1L2L1(NEW,NEW)**	**L1S1L2S2(0,NEW)**	**L2S1S2L1 (NEW,0)**
S1S2L1L2(NEW,NEW)	**S2L1S1L2(NEW,NEW)**	**L1S2S1L2(0,NEW)**	**L2L1S1S2(0,0)**
S1S2L2L1(NEW,NEW)	**S2L1L2S1(NEW,0)**	**L1S2L2S1(0,0)**	**L2L1S2S1(0,0)**
S1L2L1S2(0,NEW)	**S2L2S1L1(NEW,0)**	**L1L2S1S2(0,0)**	**L2S2S1L1(NEW,0)**
S1L2S2L1(NEW,NEW)	**S2L2L1S1(NEW,0)**	**L1L2S2S1(0,0)**	**L2S2L1S1(0,0)**

存储一致性(Coherence)



Example of incoherence

问题的解决

- **并行性不足：**
  - 通过采用并行性更好的算法来解决
- **远程访问延迟的降低：**
  - 靠体系结构支持和编程技术
- **共享存储器访问**
  - 一致性协议

7.2 集中式共享存储器体系结构

- **多个处理器共享一个存储器。**
- **当处理器规模较小时，这种机器十分经济。**
- **支持对共享数据和私有数据的Cache缓存。**
  - 私有数据供一个单独的处理器使用，而共享数据供多个处理器使用。
- **共享数据进入Cache产生了一个新的问题：**

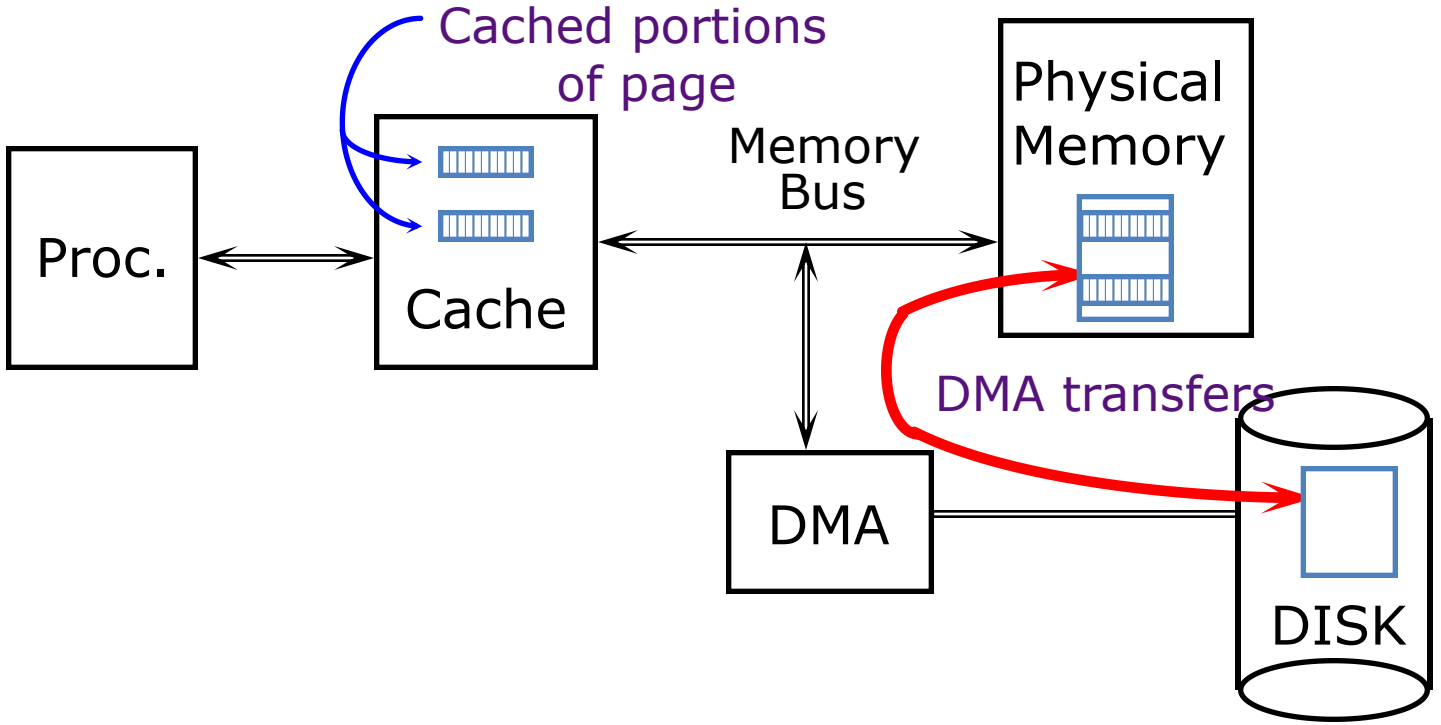
Cache的一致性问题

1、多处理机的一致性

不一致产生的原因（Cache一致性问题）

- **I / O操作**
  - Cache中的内容可能与由I / O子系统输入输出形成的存储器对应部分的内容不同。
- **共享数据**
  - 不同处理器的Cache都保存有对应存储器单元的内容

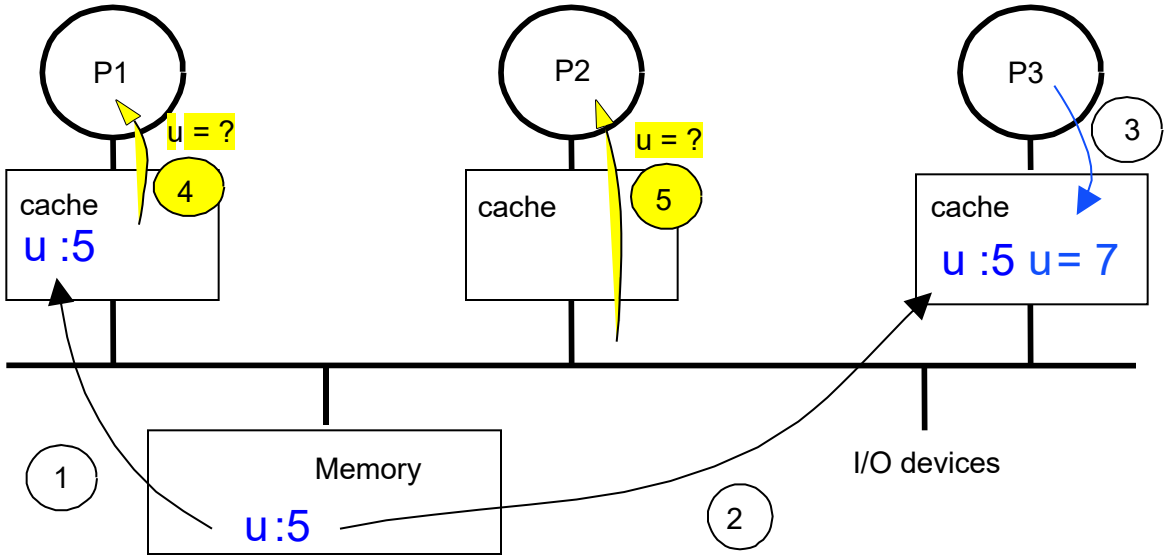
Problems with Parallel I/O



Memory → Disk: 如果cache的数据被修改过，而没有写回，存储器是陈旧数据

Disk → Memory: Cache中的数据是陈旧数据，它并不知道这次存储器写操作

Example on Cache Coherence Problem



- **P3执行了写操作后，处理器看到了u的不同值**
- **针对write back caches …**
  - 处理器访问主存可能看到一个陈旧的（不正确）值
  - 结果依赖于cache 写回的顺序
- **显然，这样会影响程序执行的结果**

存储系统是一致的

- **如果对某个数据项的任何读操作均可得到其最新写入的值，则认为这个存储系统是一致(coherent)的（非正式定义）**
- **如果存储系统行为满足条件：**
  - 处理器P对X进行一次写之后又对X进行读，读和写之间没有其它处理器对X进行写，则读的返回值总是写进的值。
  - 一个处理器对X进行写之后，另一处理器对X进行读，读和写之间无其它写，则读X的返回值应为写进的值
  - 对同一单元的写是顺序化的，即任意两个处理器对同一单元的两次写，从所有处理器看来顺序都应是相同的
- **2点假设**
  - 直到所有的处理器均看到了写的结果，一次写操作才算完成**（写传播）**
  - 允许处理器无序读，但必须以程序序进行写**（写串行化）**

另一种定义

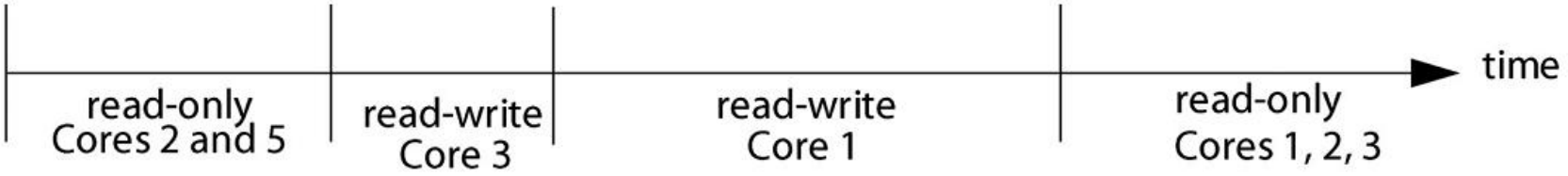


FIGURE 2.3: Dividing a given memory location's lifetime into epochs

Coherence invariants

1. **Single-Writer, Multiple-Read (SWMR) Invariant.** For any memory location A, at any given (logical) time, there exists only a single core that may write to A (and can also read it) or some number of cores that may only read A.
2. **Data-Value Invariant.** The value of the memory location at the start of an epoch is the same as the value of the memory location at the end of its last read–write epoch.

2、实现一致性的基本方案

- **在一致的多处理机中，Cache提供两种功能**
  - 共享数据的迁移:
    - 降低了对远程共享数据的访问延迟和对共享存储器的带宽要求。
  - 共享数据的复制
    - 不仅降低了访存的延迟，也减少了访问共享数据所产生的冲突。
- **小规模多处理机不是采用软件而是采用硬件技术实现Cache一致性。**

基本模型

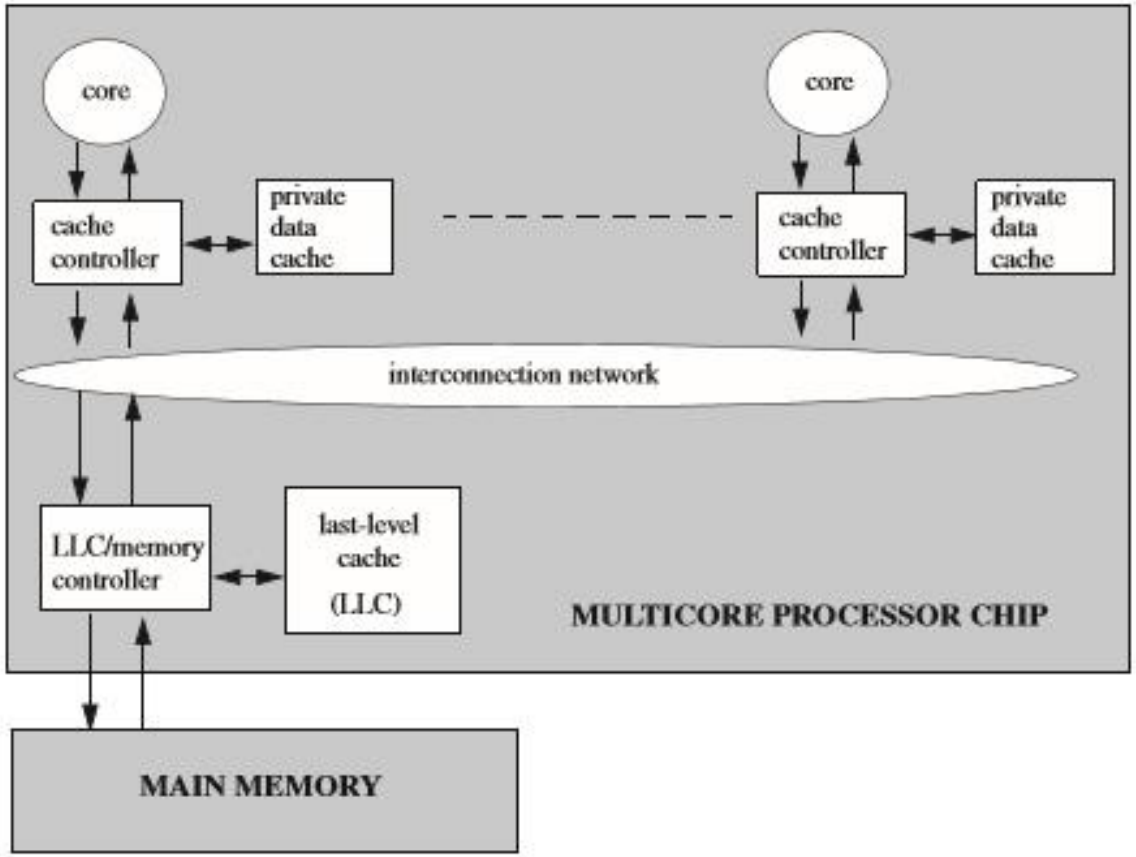


FIGURE 2.1: Baseline system model used throughout this primer.

Cache一致性协议

- **对多个处理器维护一致性的协议**
- **关键：跟踪共享数据块的状态**
- **共享数据状态跟踪技术**
  - 目录 (directory): 物理存储器中共享数据块的状态及相关信息均被保存在一个称为目录的地方。
  - 监听 (bus snooping): 每个Cache除了包含物理存储器中块的数据拷贝之外，也保存着各个块的共享状态信息。

3、基于监听的两种协议

- **写作废(write-invalidate)协议**
  - 在一个处理器写某个数据项之前保证它对该数据项有唯一的访问权
  - 例 在写回Cache的条件下，监听总线中写作废协议的实现。

处理器行为	总线行为	CPU A Cache内容	CPU B Cache内容	主存X单元内容
				0
CPU A 读X	Cache失效	0		0
CPU B 读X	Cache失效	0	0	0
CPUA将X单元写1	作废X单元	1		0
CPU B 读X	Cache失效	1	1	1

- **写更新(Write Update)协议**
  - 当一个处理器写某数据项时，通过广播使其它Cache中所有对应的该数据项拷贝进行更新。
  - 例 在写回Cache的条件下，监听总线中写更新协议的实现。

处理器行为	总线行为	CPUA Cache内容	CPUB Cache内容	主存X单元内容
				0
CPU A 读X	Cach失效	0		0
CPU B 读X	Cach失效	0	0	0
CPUA将X单元写1	广播写X单元	1	1	1
CPU B 读X		1	1	1

写更新和写作废协议性能上的差别

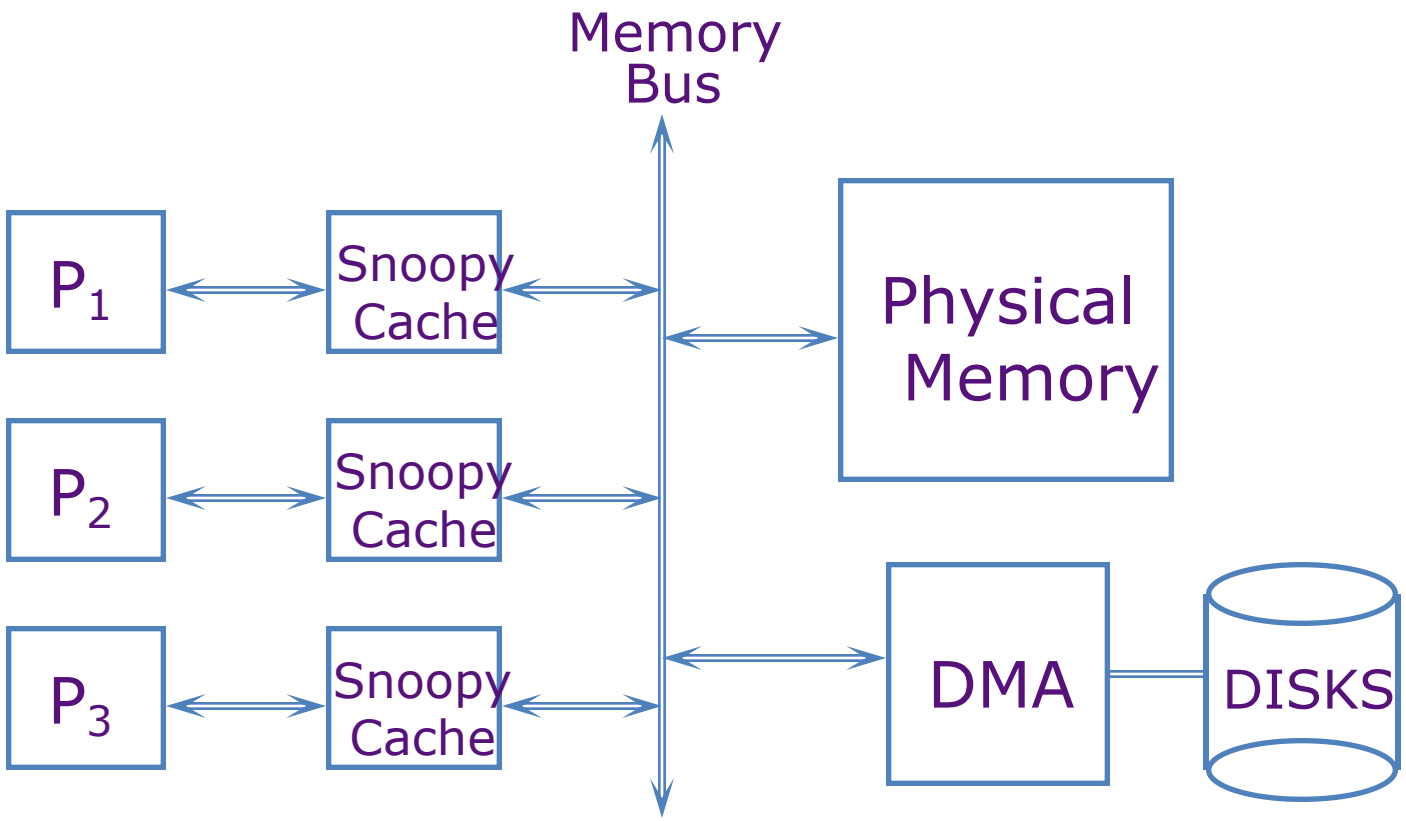
- 对同一数据的多个写而中间无读操作的情况,写更新协议需进行多次写广播操作，而在写作废协议下只需一次作废操作
- 对同一块中多个字进行写，写更新协议对每个字的写均要进行一次广播，而在写作废协议下仅在对本块第一次写时进行作废操作
- 一个处理器写到另一个处理器读 之间的延迟通常在写更新模式中较低。而在写作废协议中，需要读一个新的拷贝

在基于总线的多处理机中，写作废协议成为绝大多数系统设计的选择。

4. 监听协议的基本实现技术

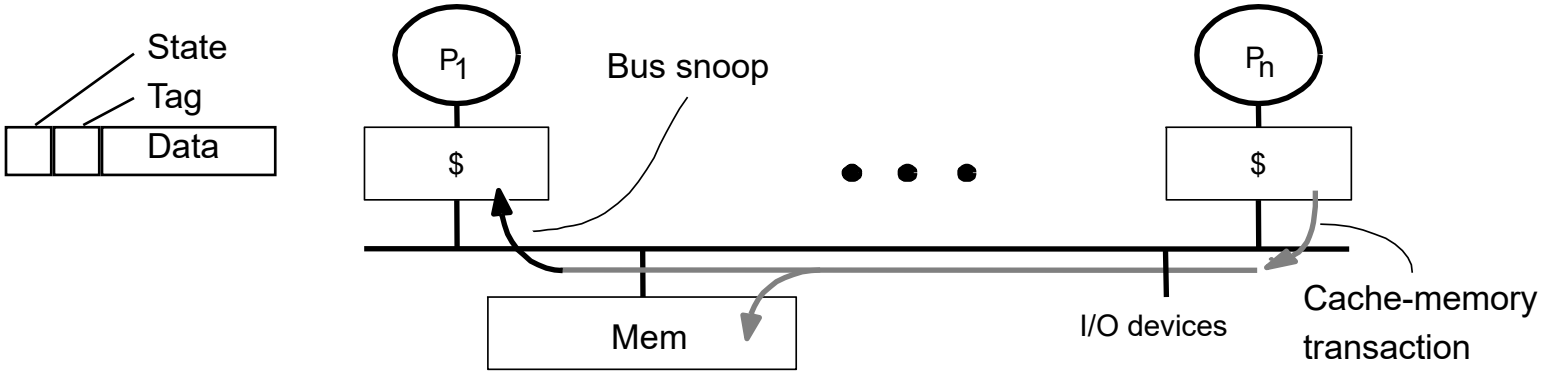
- **小规模多处理机中实现写作废协议的关键利用总线进行作废(invalidate)操作,每个块的有效位使作废机制的实现较为容易。**
- **写直达Cache，因为所有写的数据同时被写回主存，则从主存中总可以取到最新的数据值。**
- **对于写回Cache，得到数据的最新值会困难一些，因为最新值可能在某个Cache中，也可能在主存中。**
- **在写回Cache条件下的实现技术**
  - 用Cache中块的标志位实现监听过程。
  - 给每个Cache块加一个特殊的状态位说明它是否为共享。

Shared Memory Multiprocessor

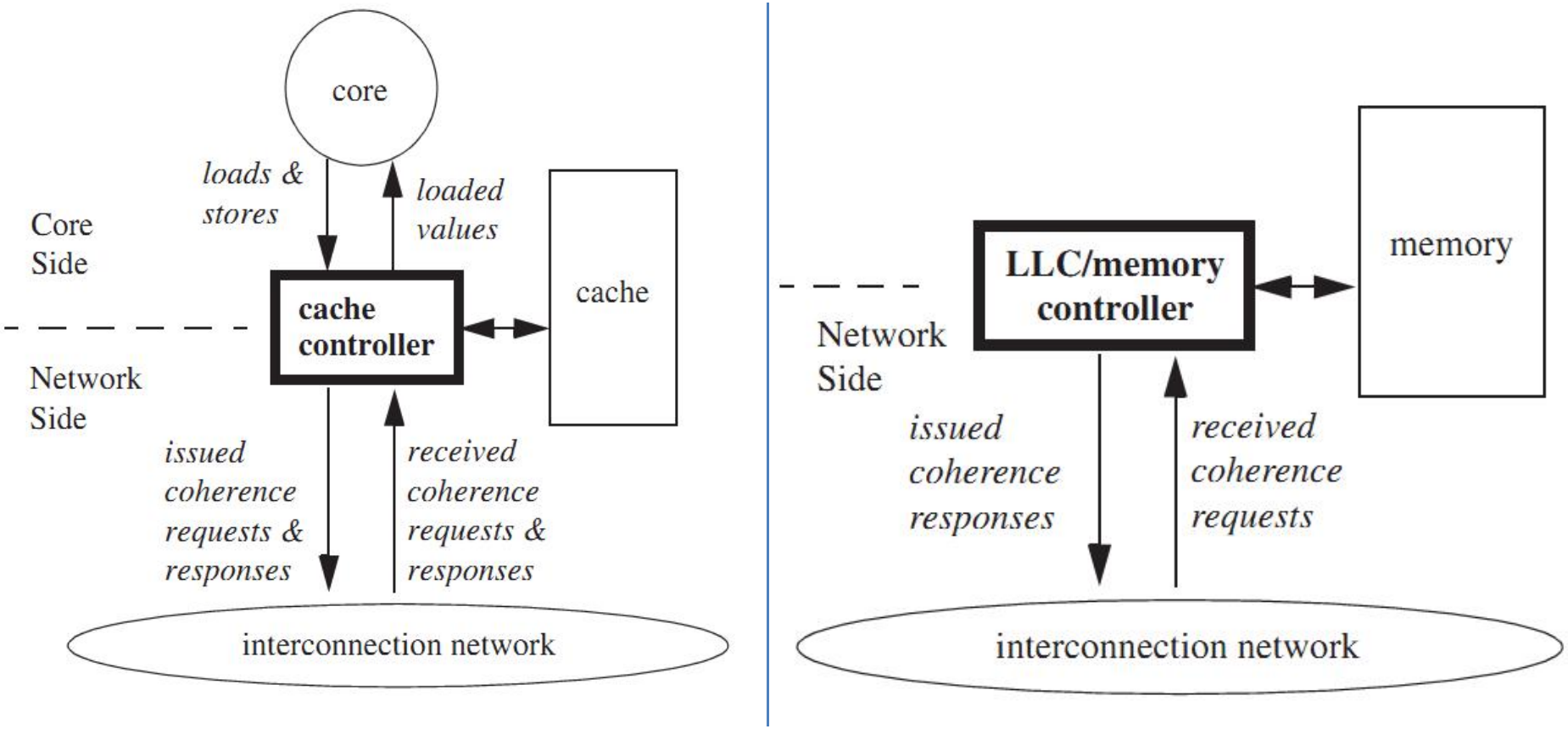


利用监听机制来保持处理器看到的存储器的视图一致

Snoopy Cache-Coherence Protocols



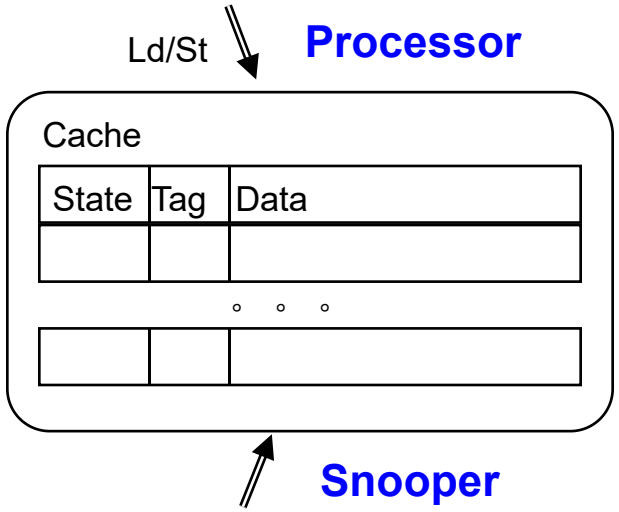
- **总线作为广播的媒介&Caches可知总线的行为**
  - 总线上的事务对所有Cache是可见的
  - 这些事务对所有控制器以同样的顺序可见
- **Cache 控制器监测 （snoop）共享总线上的所有事务**
  - 根据Cache中块的状态不同会产生不同的事务
  - 通过执行不同的总线事务来保证Cache的一致性
    - Invalidate, update, or supply value



Cache通常连在共享存储器的总线上，各个Cache控制器通过监听总线来判断它们是否有总线上请求的数据块。

Implementing a Snooping Protocol

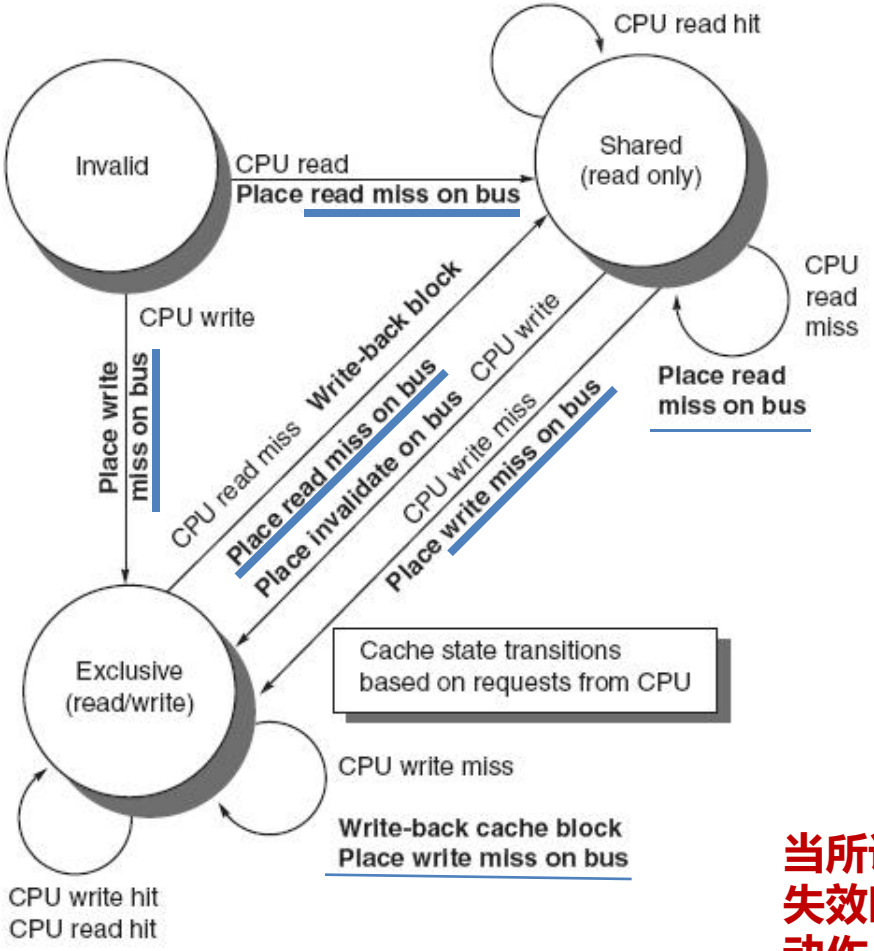
- **Cache 控制器接收两方面的请求输入：**
  - 处理器的请求 (load/store)
  - 监测器 （snooper)的总线请求/响应
- **Cache 控制器根据这两方面的输入产生动作**
  - 更新Cache块的状态
  - 提供数据
  - 产生新的总线事务

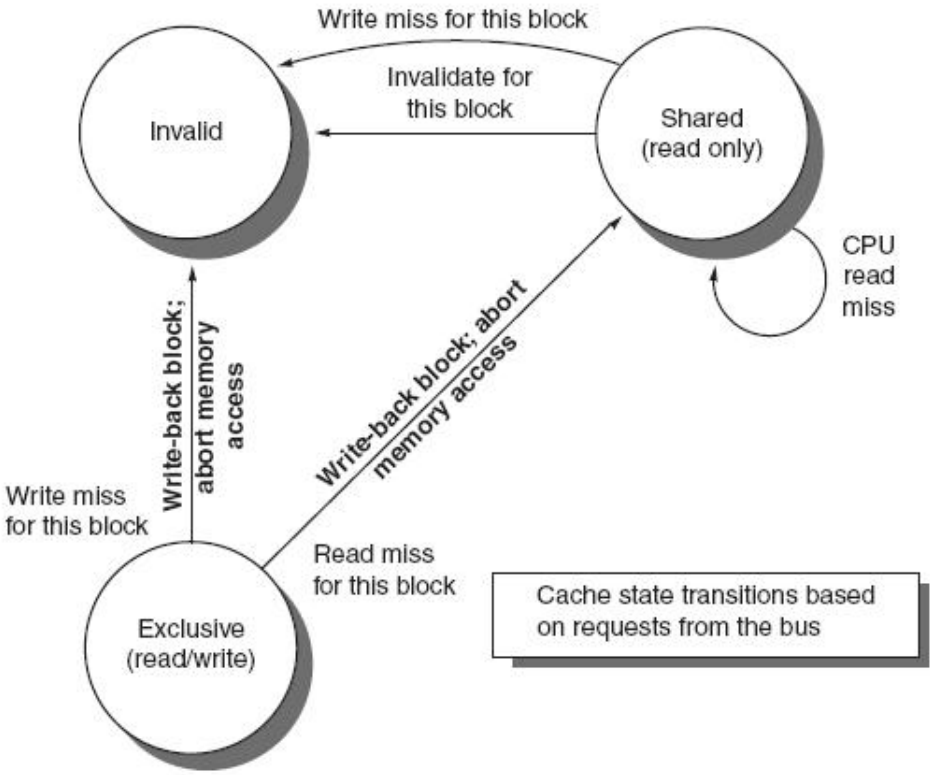


MSI Write-Back Invalidate Protocol

- **3 states:**
  - Modified: 仅该cache拥有修改过的、有效的该块copy
  - Shared: 该块是干净块，其他cache中也可能含有该块，存储器中的内容是最新的
  - Invalid: 该块是无效块（invalid）
- **4 bus transactions:**
  - Read Miss : 服务于Read Miss on Bus
  - Write Miss: 服务于Write Miss on Bus,得到一个独占的块
  - Invalidate: 作废该块在其他处理器中的Copy
  - Write back：替换操作将修改过的块写回
- **写操作时，作废所有其他块**
  - 直到Invalidate transaction出现在总线上，写操作才算完成
  - 写串行化：总线事务在总线上串行化

MSI Snoopy Cache Coherence Protocol





当所访问的块的最新数据在某个私有Cache时，在读写失效时，数据的提供者是拥有该块数据的私有Cache。
动作：Write-back block; abort memory access

MSI Snoopy Cache Coherence Protocol

MSI Snoopy Cache Coherence Protocol

Request	Source	State Transition	Action and Explanation
Read Hit	Processor	Shared or Modified	Normal Hit: Read data in private data cache (no transaction)
Read Miss	Processor	Invalid → Shared	Normal Miss: Place **read miss on bus**, change state
Read Miss	Processor	Shared	Replace block: Place **read miss on bus**
Read Miss	Processor	Modified → Shared	**Write-Back** block, Place **read miss on bus**, change state
Write Hit	Processor	Modified	Normal Hit: Write data in private data cache (no transaction)
Write Hit	Processor	Shared → Modified	Coherence: Place **invalidate on bus** (no data), change state
Write Miss	Processor	Invalid → Modified	Normal Miss: Place **write miss on bus**, change state
Write Miss	Processor	Shared → Modified	Replace block: Place **write miss on bus**, change state
Write Miss	Processor	Modified	**Write-Back** block, Place **write miss on bus**
Read Miss	Bus	Shared	**Serve read miss** from shared cache or memory
Read Miss	Bus	Modified → Shared	Coherence: **Write-Back & Serve read miss**, change state
Invalidate	Bus	Shared → Invalid	Coherence: **Invalidate shared block** in other private caches
Write Miss	Bus	Shared → Invalid	Coherence: **Invalidate shared block** in other private caches
Write Miss	Bus	Modified → Invalid	Coherence: **Write-Back & Serve write miss, Invalidate**

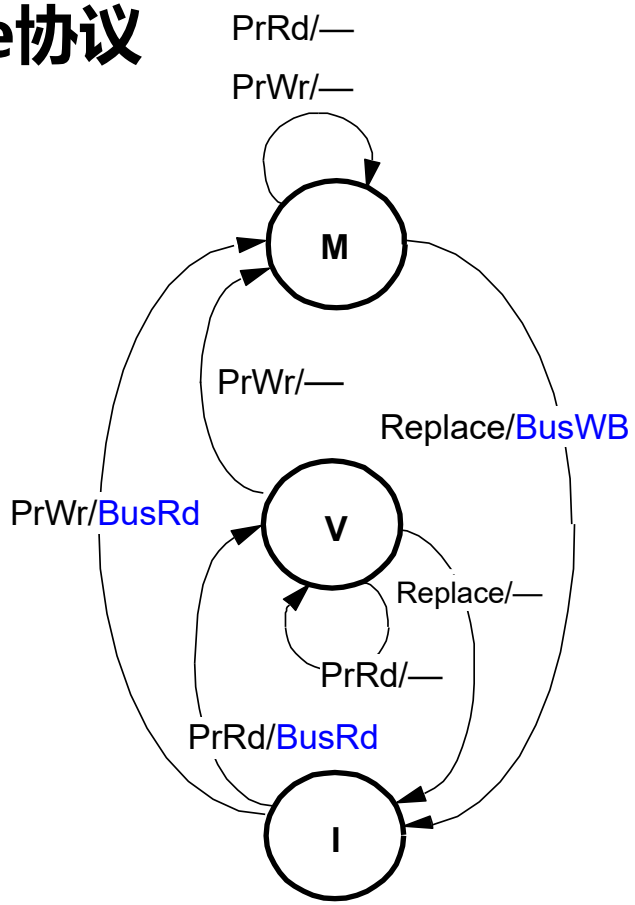
Example on MSI Cache Coherence

Request	Processor P1			Processor P2			Bus			Memory	
	State	Addr	Value	State	Addr	Value	Proc	Addr	Action	Addr	Value
P1: Write 10 to A1							P1	A1	Wr Miss	A1	15
	M	A1	10								
P1: Read A1 (Hit)	M	A1	10								
P2: Read A1							P2	A1	Rd Miss		
	S	A1	10				P1	A1	Wr Back	A1	10
				S	A1	10	P2	A1	Transfer		
P2: Write 20 to A1							P2	A1	Invalidate		
	I	A1	10	M	A1	20				A1	10
P2: Write 40 to A2				M	A2	40				A2	25

- Assume that A1 and A2 map to same cache block
- Initial cache state is invalid

Write-back Cache

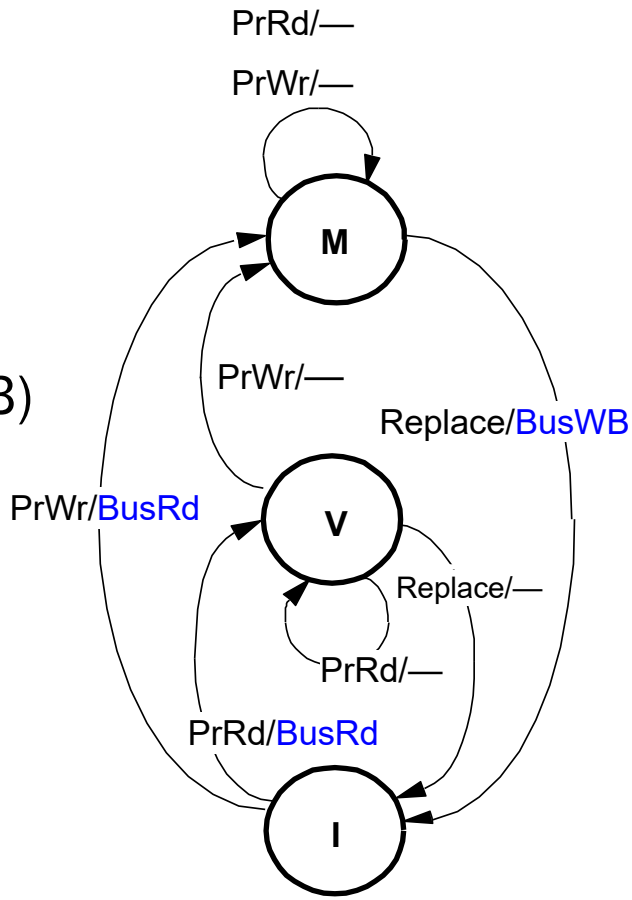
- **另外一种MSI写回cache的invalidate协议**
- **先看单个处理器+单个Cache**
- **Cache块状态**
  - Invalid, Valid (clean), Modified (dirty)
- **Processor / Cache 操作**
  - PrRd, PrWr, block Replace
- **总线事务**
  - Bus Read (BusRd), Write-Back (BusWB)
  - 仅传送cache-block



Write-back Cache

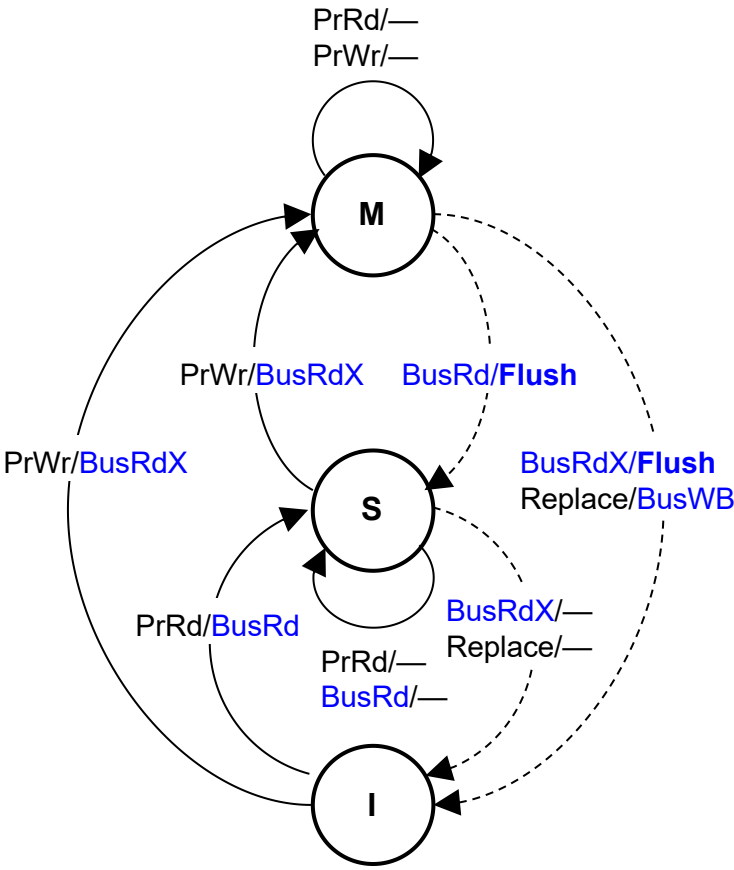
- **Cache块状态**
  - Invalid, Valid (clean), Modified (dirty)
- **Processor / Cache 操作**
  - PrRd, PrWr, block Replace
- **总线事务**
  - Bus Read (BusRd), Write-Back (BusWB)
  - 仅传送cache-block

- **针对Cache一致性的块状态调整**
  - Treat Valid as Shared
  - Treat Modified as Exclusive
- **引入新的总线事务**
  - Bus Read-eXclusive (BusRdX)
  - 其基本动作是：读进并意图修改



MSI Write-Back Invalidate Protocol

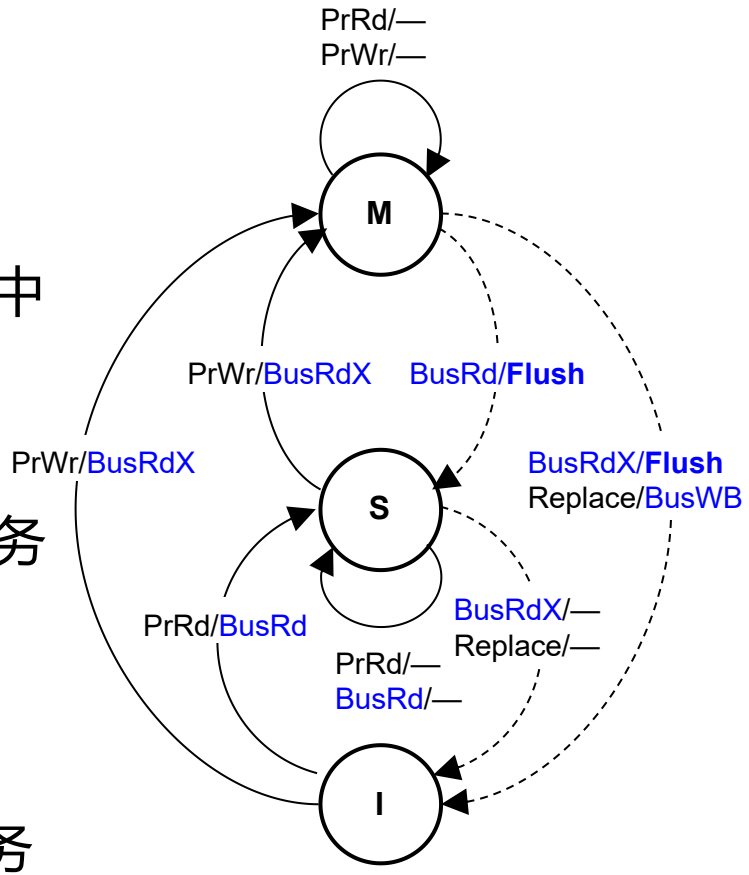
- **3 states:**
  - Modified: 仅该cache拥有修改过的、有效的该块copy
  - Shared: 该块是干净块，其他cache中也可能含有该块，存储器中的内容是最新的
  - Invalid: 该块是无效块（invalid）
- **4 bus transactions:**
  - Bus Read: 读失效时产生BusRd总线事务
  - Bus Read Exclusive（总线排他读）: BusRdX
    - 得到独占的（exclusive）cache block
    - 其基本动作为读进并意图修改
  - Bus Write-Back: BusWB用于cache 块的替换
  - Flush on BusRd or BusRdX
    - Cache将数据块放到总线上（而不是从存储器取数据）完成 Cache-to-cache的传送，并更新存储器



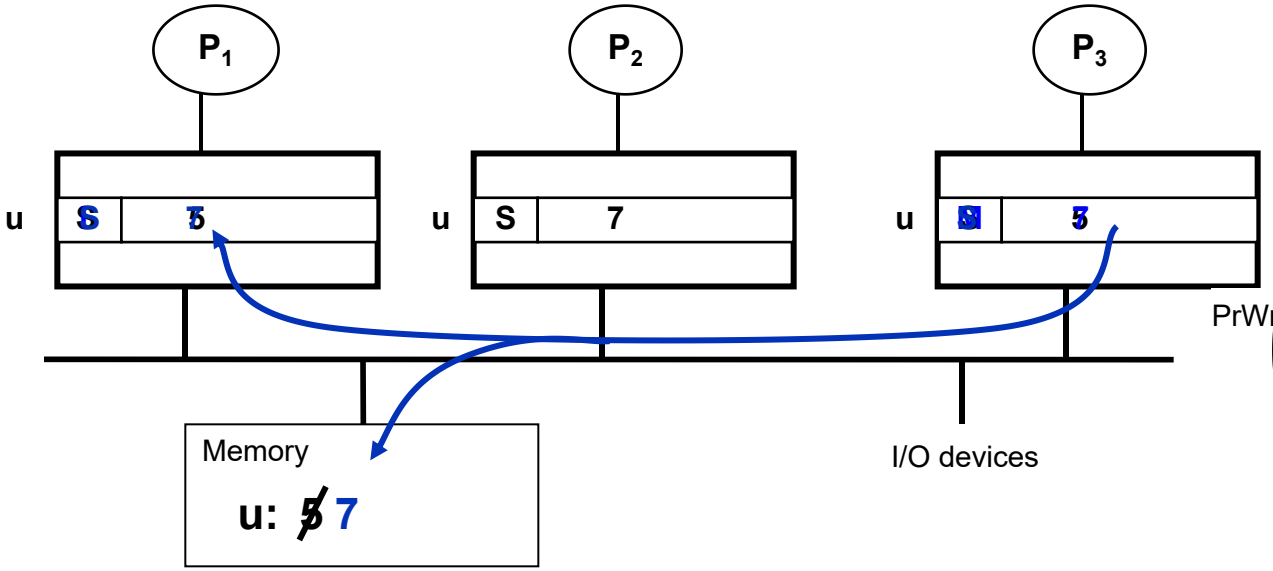
readx vs read + write

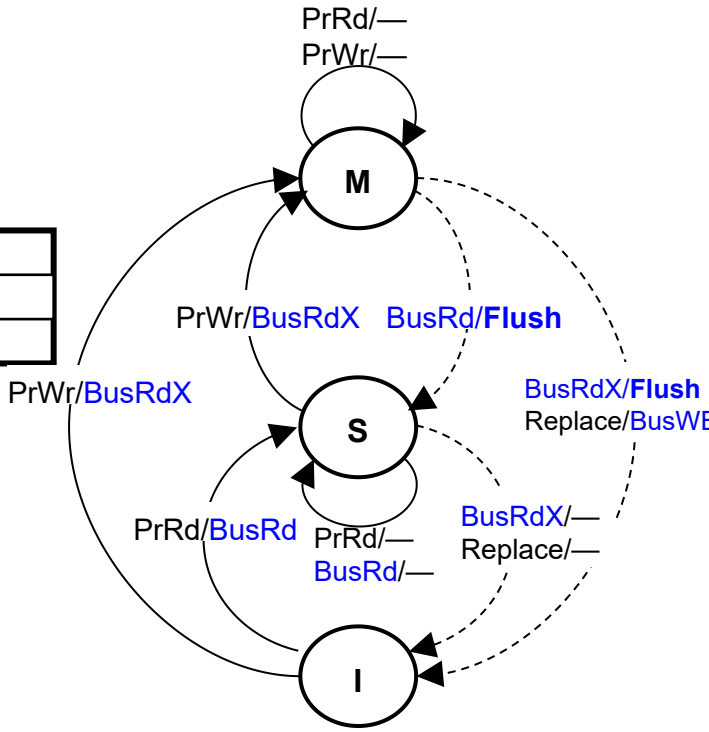
State Transitions in the MSI Protocol

- **Processor Read**
  - Cache miss ⇒ 产生BusRd事务
  - Cache hit (S or M) ⇒ 无总线动作
- **Processor Write**
  - 当在非Modified状态时，产生总线BusRdX 事务，BusRdX 导致其他Cache中的对应块作废（invalidate）
  - 当在Modified状态时，无总线动作
- **Observing a Bus Read**
  - 如果该块是 Modified, 产生Flush总线事务
    - 更新存储器和有需求的Cache
    - 引起总线事务的Cache块状态⇒ Shared
- **Observing a Bus Read Exclusive**
  - 作废相关block
  - 如果该块是modified, 产生Flush总线事务



Example on MSI Write-Back Protocol





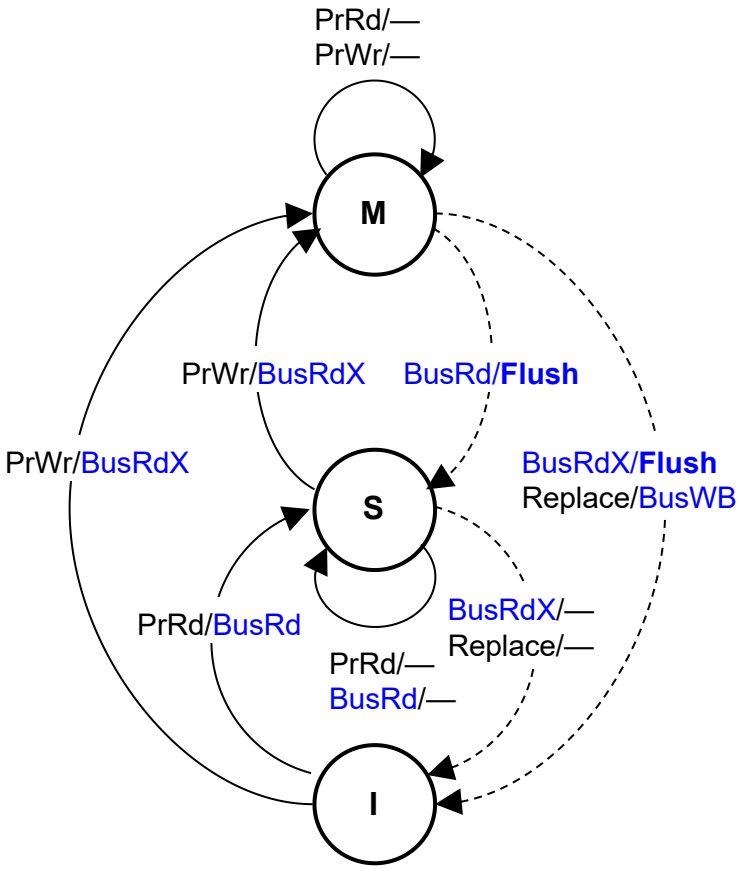
Processor Action	State P1	State P2	State P3	Bus Action	Data from
1. P1 reads u	S			BusRd	Memory
2. P3 reads u	S		S	BusRd	Memory
3. P3 writes u	I		M	BusRdX	Memory
4. P1 reads u	S		S	BusRd, Flush	P3 cache
5. P2 reads u	S	S	S	BusRd	Memory

Lower-level Design Choices

- **引入Bus Upgrade (BusUpgr) 将Cache块状态从S到M**
  - 引起作废操作 (类似 BusRdX)，但避免块的读操作
- **当 M态的块观察到BusRd 时，变迁到哪个态**
  - M → S or M → I 取决于访问模式
- **Transition to state S**
  - 如果不久会有本地读操作，而不是其他处理器的写操作
  - 比较适合于经常发生读操作的访问模式
- **Transition to state I**
  - 经常发生其他处理器写操作
  - 比较适合数据迁移操作：即本地写后，其他处理器将会发出读和写请求，然后本地又进行读和写。即连续的对称式访问模式。
- **不同选择方案会影响存储器的性能**

Satisfying Coherence

- **写传播(Write propagation)**
  - 对一个shared 或invalid块的写，其他cache都可见
    - 使用Bus Read-exclusive (BusRdX) 事务，Bus Read-exclusive 事务作废其他Cache中的块
    - 其他处理器在未看到该写操作的效果前体验到的是Cache Miss
- **写串行(Write serialization)**
  - 所有出现在bus上的写操作(BusRdX)被总线串行化
    - 所有处理器（包括发出写操作的处理器）以同样的方式排序
    - 首先更新发出写操作的处理器的本地cache，然后处理其他事务
  - 并不是所有的写操作都会出现在总线上
    - 对modified 块的写序列来自同一个处理器（P）将不会产生总线事务
    - 同一处理器是串行化的写：由P进行读操作将会看到串行序的写序列
    - 其他处理器对该块的读操作：会导致一个总线事务，这保证了写操作的顺序对其他处理器而言也是串行化的。



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**