第10章 多处理机

1. 单处理机系统结构正在走向尽头

2. 多处理机正起着越来越重要的作用。近几年来，人们确实开始转向了多处理机。

- Intel于2004年宣布放弃了其高性能单处理器项目，转向多核（multi-core）的研究和开发。
- IBM、SUN、AMD等公司
- 并行计算机应用软件已有了稳定的发展。
- 充分利用商品化微处理器所具有的高性能价格比的优势。

3. 本章重点：中小规模的计算机（多处理机设计的主流）

We are dedicating all of our future product development to multicore designs. We believe this is a key inflection point for the industry.

Intel President Paul Otellini,
describing Intel's future direction at the Intel Developers Forum in 2005

The Teraflops Research Chip

Frequency	Voltage	Power	Aggregate Bandwidth	Performance
3.16 GHz	0.95 V	62W	1.62 Terabits/s	1.01 Teraflops
5.1 GHz	1.2 V	175W	2.61 Terabits/s	1.63 Teraflops
5.7 GHz	1.35 V	265W	2.92 Terabits/s	1.81 Teraflops

10.1.1 并行计算机系统结构的分类

1. Flynn分类法

SISD、SIMD、MISD、MIMD

2. MIMD已成为通用多处理机系统结构的选择，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。

计算机机群系统（cluster）是一类广泛被采用的MIMD机器。

3. 根据存储器的组织结构，把现有的MIMD机器分为两类：

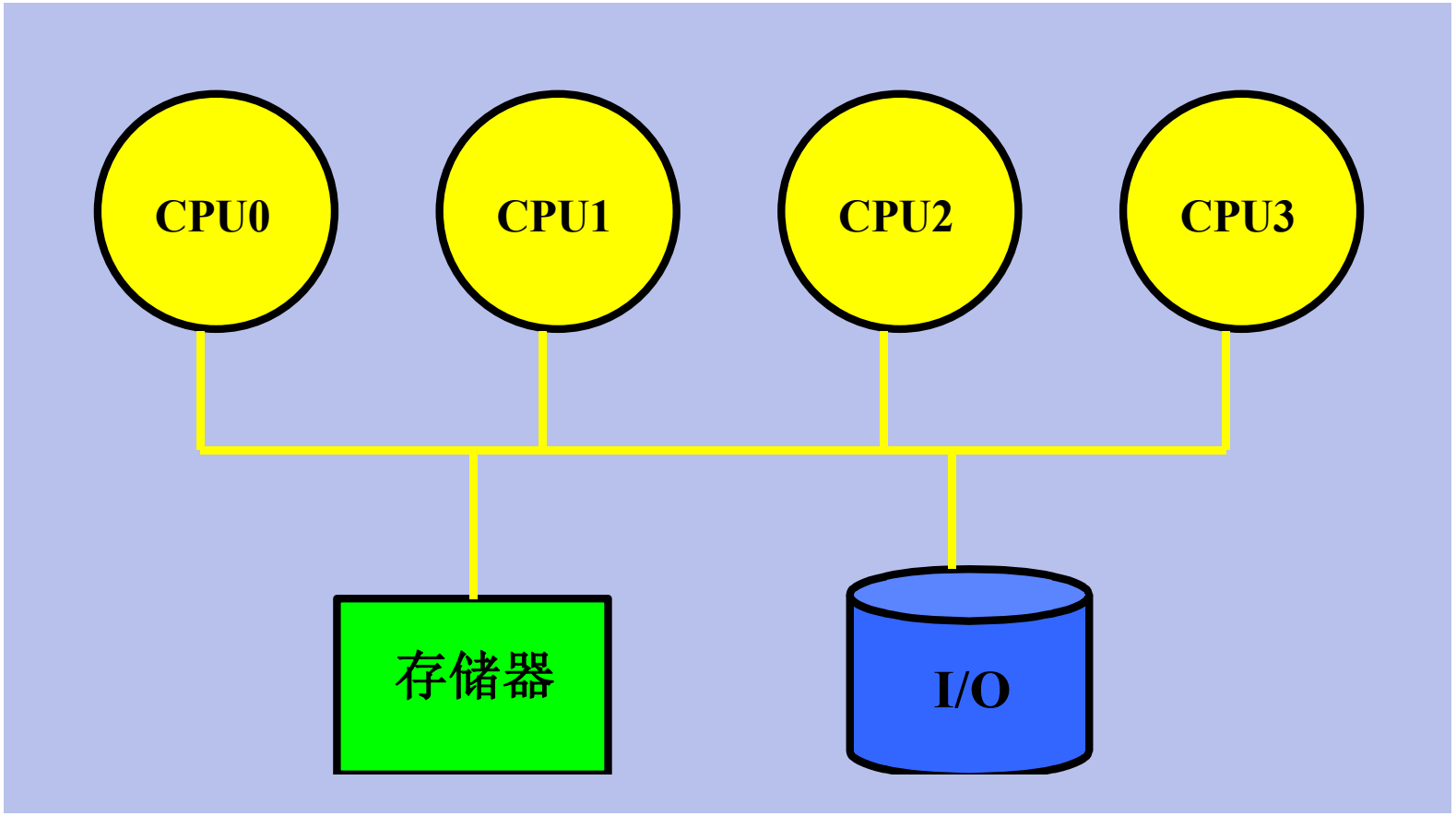
（每一类代表了一种存储器的结构和互连策略）

- 集中式共享存储器结构
  - 最多由几十个处理器构成。
  - 各处理器共享一个集中式的物理存储器。

这类机器有时被称为

- SMP机器

  （Symmetric shared-memory MultiProcessor）
- UMA机器（Uniform Memory Access）



对称式共享存储器多处理机的基本结构

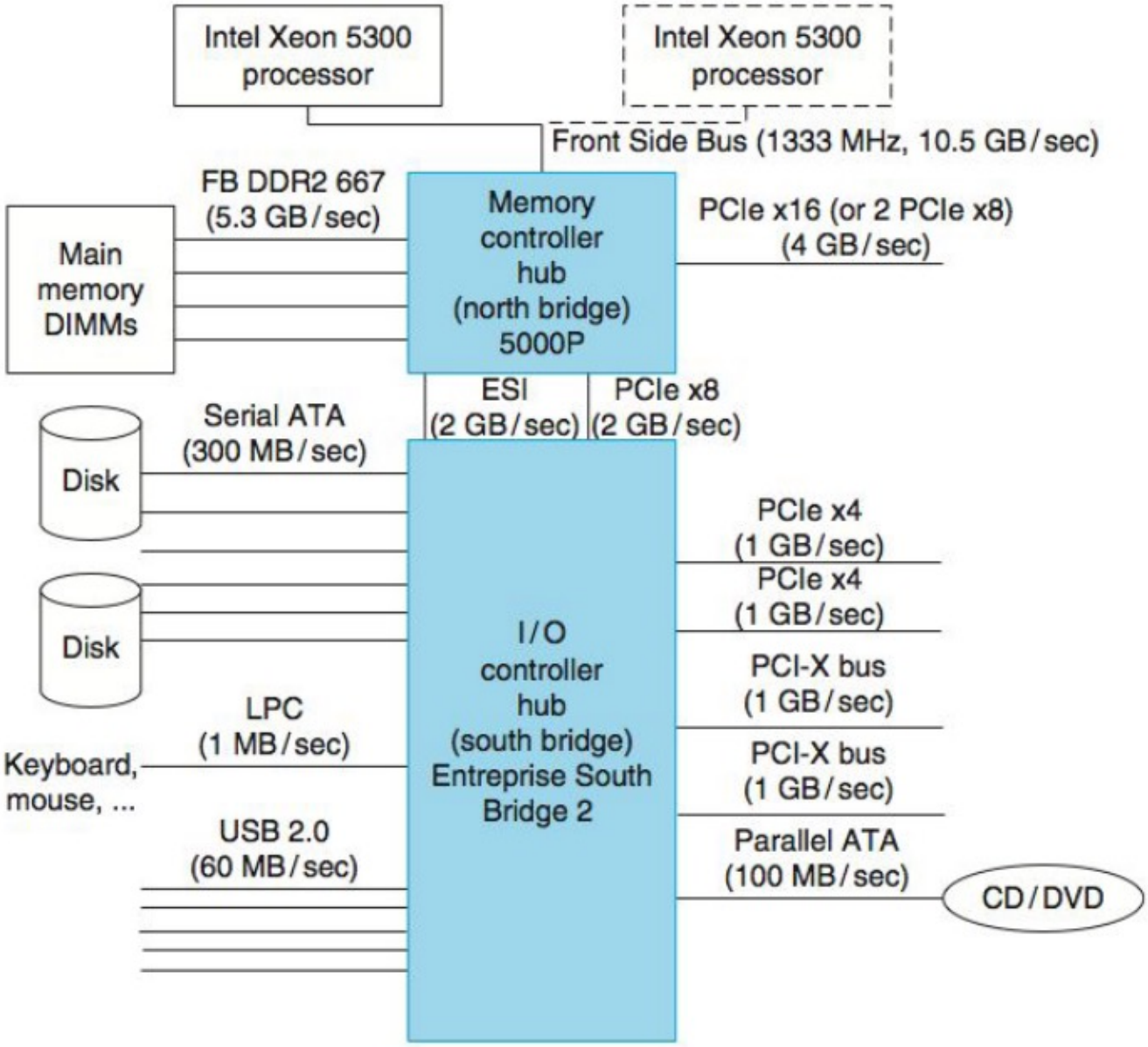
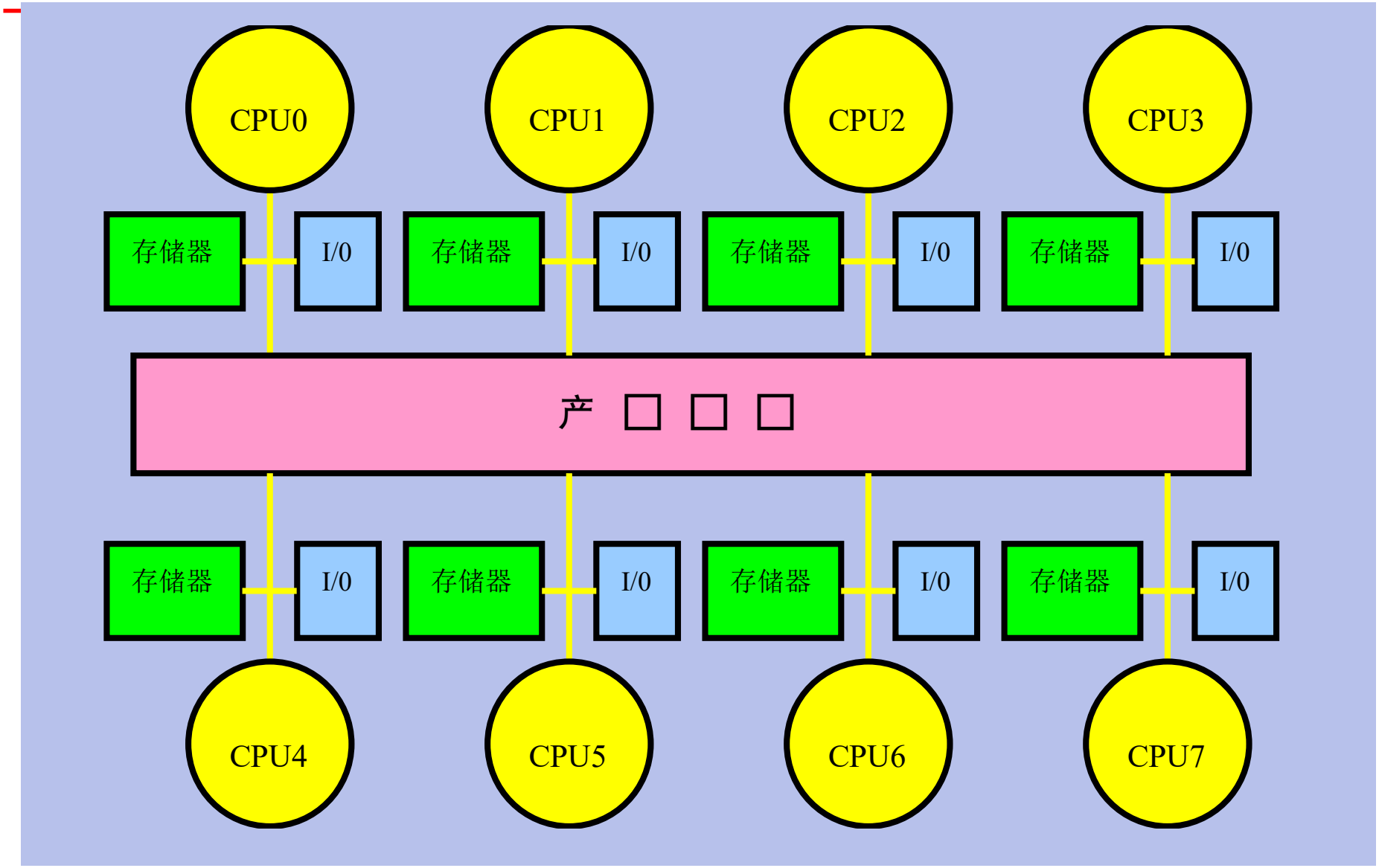
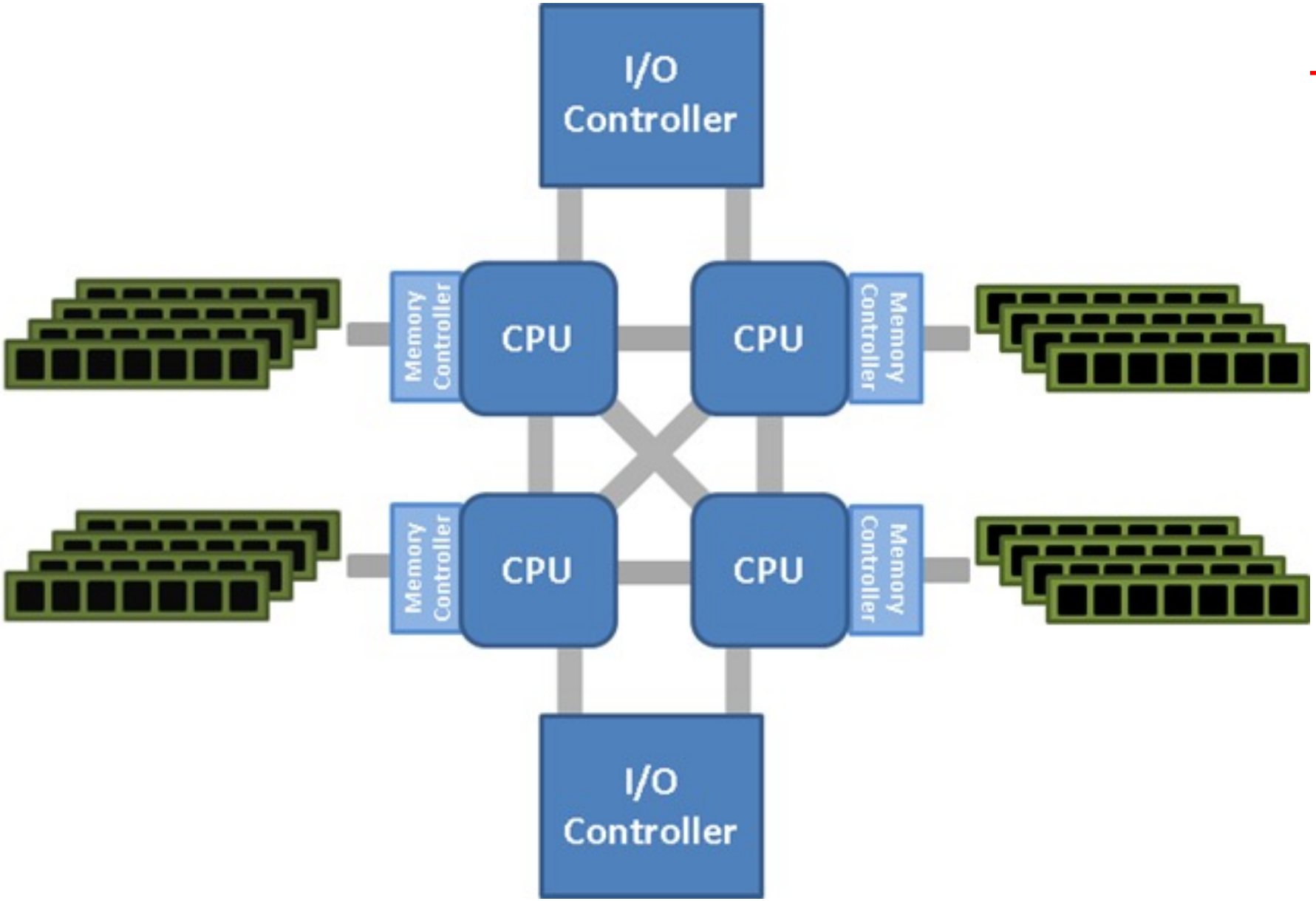


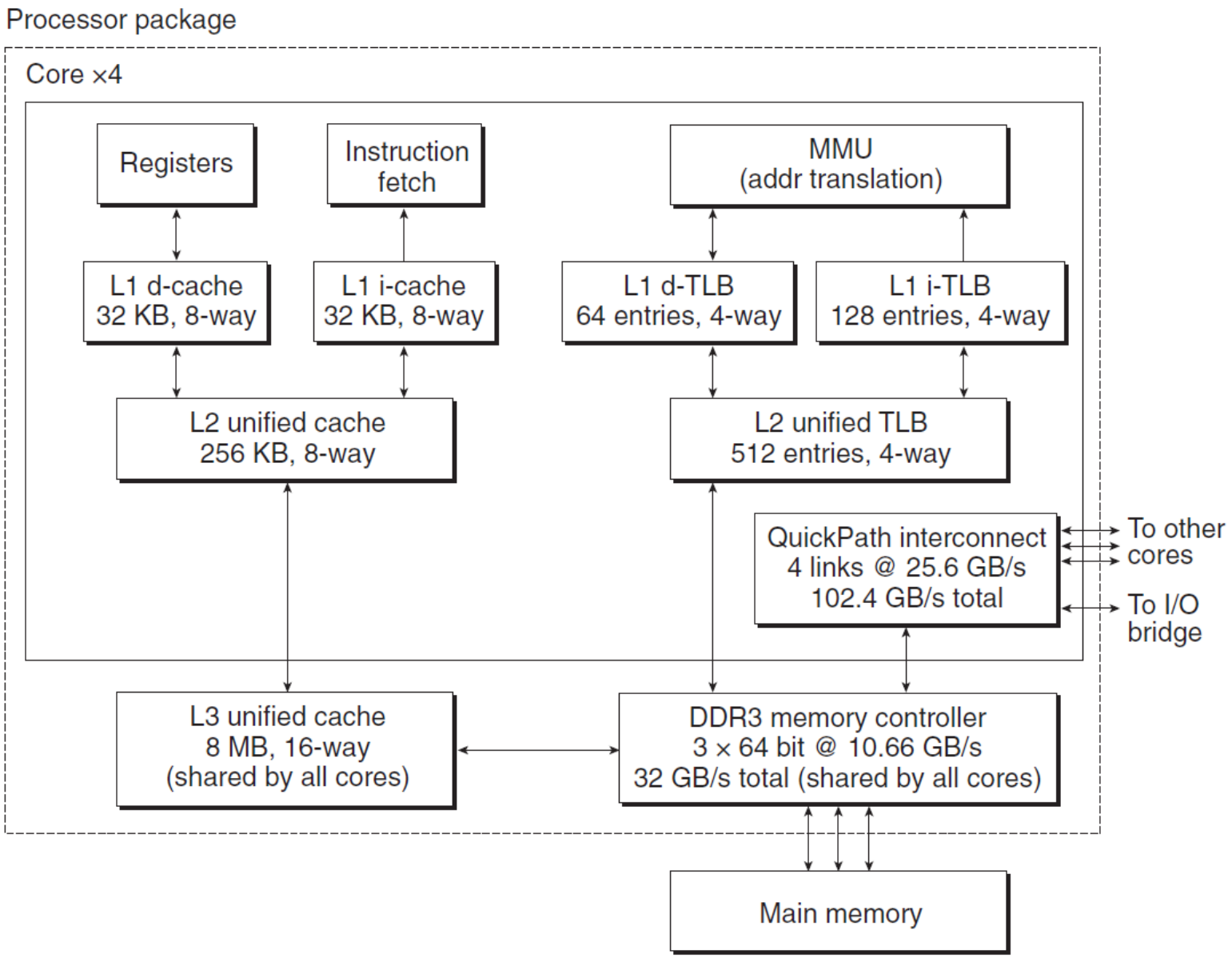
FIGURE 6.9 Organization of the I/O system on an Intel server using the Intel 5000P chip set. If you assume reads and writes are each half the traffic, you can double the bandwidth per link for PCIe.

- 分布式存储器多处理机
  - 存储器在物理上是分布的。
  - 每个结点包含：
    - 处理器
    - 存储器
    - I/O
    - 互连网络接口


CPU0
CPU1
CPU2
CPU3
存储器
I/0
存储器
I/0
存储器
I/0
存储器
I/0
产 □ □ □
存储器
I/0
存储器
I/0
存储器
I/0
存储器
I/0
CPU4
CPU5
CPU6
CPU7

- 将存储器分布到各结点有两个优点
  - 如果大多数的访问是针对本结点的局部存储器，则可降低对存储器和互连网络的带宽要求；
  - 对本地存储器的访问延迟时间小。
- 最主要的缺点
  - 处理器之间的通信较为复杂，且各处理器之间访问延迟较大。
- 簇：超级结点
  - 每个结点内包含个数较少（例如2～8）的处理器；
  - 处理器之间可采用另一种互连技术（例如总线）相互连接形成簇。


I/O Controller
Memory Controller
CPU
CPU
Memory Controller
Memory Controller
CPU
CPU
Memory Controller
I/O Controller


Processor package
Core ×4
Registers
Instruction fetch
MMU (addr translation)
L1 d-cache 32 KB, 8-way
L1 i-cache 32 KB, 8-way
L1 d-TLB 64 entries, 4-way
L1 i-TLB 128 entries, 4-way
L2 unified cache 256 KB, 8-way
L2 unified TLB 512 entries, 4-way
QuickPath interconnect 4 links @ 25.6 GB/s 102.4 GB/s total
To other cores
To I/O bridge
L3 unified cache 8 MB, 16-way (shared by all cores)
DDR3 memory controller 3 × 64 bit @ 10.66 GB/s 32 GB/s total (shared by all cores)
Main memory

10.1.2 存储器系统结构和通信机制

1. 两种存储器系统结构和通信机制

- 共享地址空间
  - 物理上分离的所有存储器作为一个统一的共享逻辑空间进行编址。
  - 任何一个处理器可以访问该共享空间中的任何一个单元（如果它具有访问权），而且不同处理器上的同一个物理地址指向的是同一个存储单元。
  - 这类计算机被称为

    分布式共享存储器系统

    （DSM: Distributed Shared-Memory）

    NUMA机器 （NUMA: Non-Uniform Memory Access）

- 把每个结点中的存储器编址为一个独立的地址空间，不同结点中的地址空间之间是相互独立的。
  - 整个系统的地址空间由多个独立的地址空间构成
  - 每个结点中的存储器只能由本地的处理器进行访问，远程的处理器不能直接对其进行访问。
  - 每一个处理器-存储器模块实际上是一台单独的计算机
  - 现在的这种机器多以集群的形式存在

2. 通信机制

- 共享存储器通信机制
  - 共享地址空间的计算机系统采用

- 处理器之间是通过用load和store指令对相同存储器地址进行读/写操作来实现的。

➢ 消息传递通信机制

- 多个独立地址空间的计算机采用
- 通过处理器间显式地传递消息来完成
- 消息传递多处理机中，处理器之间是通过发送消息来进行通信的，这些消息请求进行某些操作或者传送数据。

例如：一个处理器要对远程存储器上的数据进行访问或操作：

- 发送消息，请求传递数据或对数据进行操作；

  远程进程调用(RPC， Remote Process Call)

- 目的处理器接收到消息以后，执行相应的操作或代替远程处理器进行访问，并发送一个应答消息将结果返回。

- 同步消息传递

请求处理器发送一个消息后一直要等到应答结果才继续运行。

- 异步消息传递

数据发送方知道别的处理器需要数据，通信也可以从数据发送方来开始，数据可以不经请求就直接送往数据接受方。发送方在发出消息后，可立即继续执行原来的程序。

10.1.3 并行处理面临的挑战

并行处理面临着两个重要的挑战

- 程序中的并行性有限
- 远程访问延迟

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

1. 第一个挑战

有限的并行性使计算机要达到很高的加速比十分困难。

例10.1 假设想用100个处理器达到80的加速比，求原计算程序中串行部分最多可占多大的比例？

解 Amdahl定律为：

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

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

由上式可得：并行比例＝0.9975

2. 第二个挑战：多处理机中远程访问的延迟较大

- 在现有的机器中，处理器之间的数据通信大约需要50～1000个时钟周期。
- 主要取决于：

  通信机制、互连网络的种类和多处理机的规模

例10.2 假设有一台32台处理器的多处理机，对远程存储器访问时间为200ns。除了通信以外，假设所有其它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。处理器的时钟频率为2GHz，如果指令基本的CPI为0.5（设所有访存均命中Cache），求在没有远程访问的情况下和有0.2%的指令需要远程访问的情况下，前者比后者快多少?

解 有0.2%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

＝0.5＋0.2%×远程访问开销

远程访问开销为：

远程访问时间/时钟周期时间＝200ns/0.5ns＝400个时钟周期

∴ CPI＝0.5＋0.2%×400＝1.3

因此在没有远程访问的情况下的机器速度是有0.2%远程访问的机器速度的1.3/0.5=2.6倍。

- 问题的解决
  - 并行性不足： 采用并行性更好的算法
  - 远程访问延迟的降低：靠系统结构支持和编程技术

3. 并行程序的计算 / 通信比率

- 反映并行程序性能的一个重要的度量：

  计算与通信的比率

- 计算 / 通信比率随着处理数据规模的增大而增加；随着处理器数目的增加而减少。

10.2 对称式共享存储器系统结构

- 多个处理器通过共享总线共享一个存储器
- 大容量、多级Cache降低了对内存带宽和总线带宽的要求

  2005，AMD & Intel two processor for server

  2006，Sun T1 eight processor multicore
- 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

  共享数据则是供多个处理器使用
- 共享数据进入Cache产生了一个新的问题

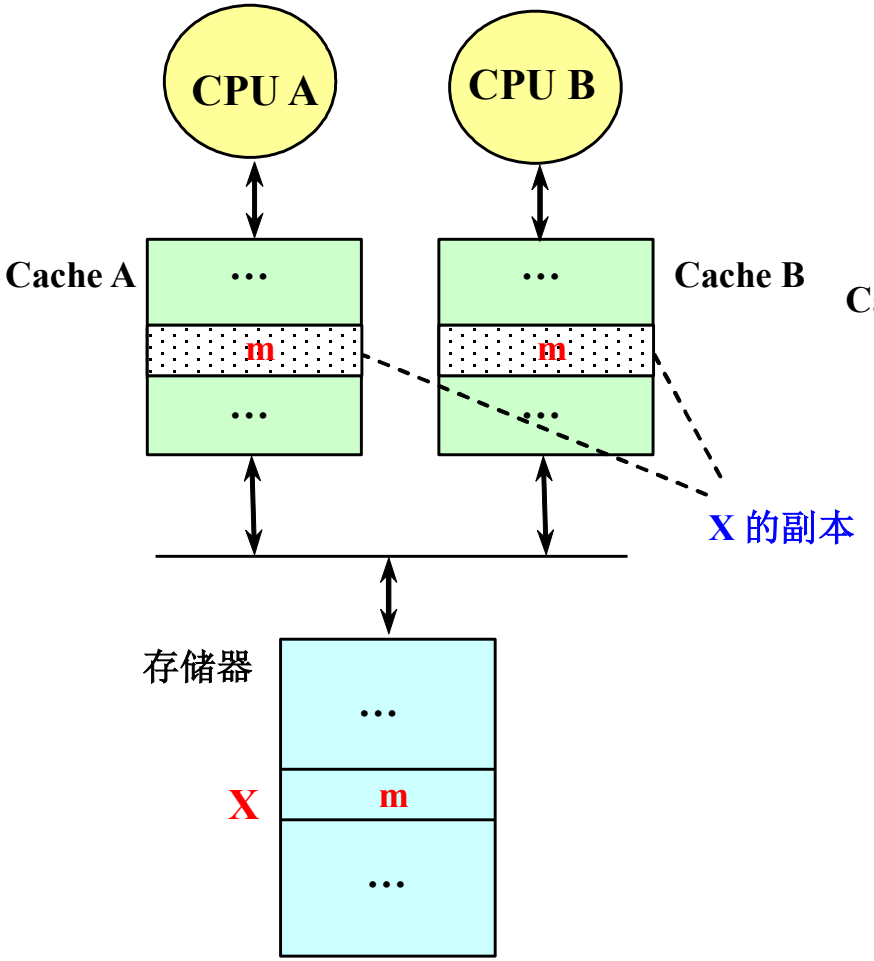
  Cache的一致性问题

10.2.1 多处理机Cache一致性

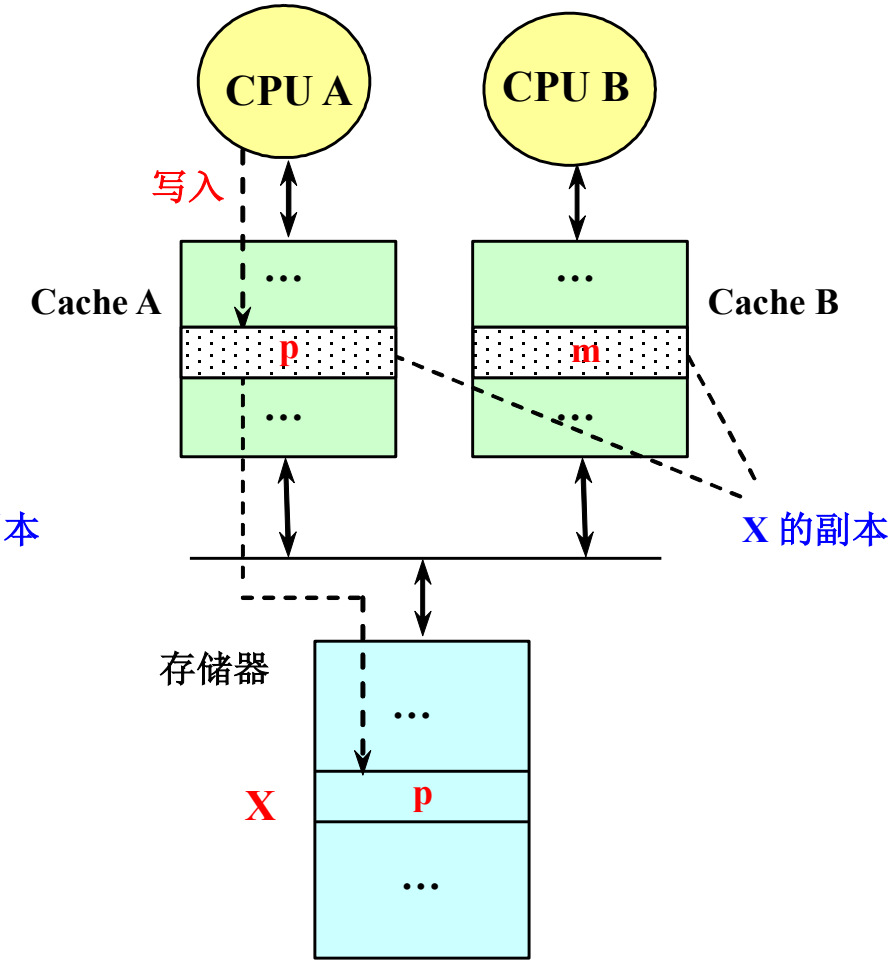
1. 多处理机的Cache一致性问题

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

例 由两个处理器（A和B）读写引起的Cache一致性问题



（a） CPU A 写入前



（b） CPU A 将 p 写入 X，p≠m

2. 存储器的一致性

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的。

- 存储系统行为的两个不同方面
  - What：读操作得到的是什么值
  - When：什么时候才能将已写入的值返回给读操作(When a written value must be seen by a read)
- 需要满足以下条件
  - 处理器P对单元X进行一次写之后又对单元X进行读，读和写之间没有其它处理器对单元X进行写，则P读到的值总是前面写进去的值。

- 处理器P对单元X进行写之后，另一处理器Q对单元X进行读，读和写之间无其它写，则Q读到的值应为P写进去的值。(if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.)
- 对同一单元的写是串行化的.

➢ 由于When a written value must be seen by a read，在后面的讨论中，我们假设：

- 直到所有的处理器均看到了写的结果，这个写操作才算完成；
- 处理器的任何访存均不能改变写的顺序。就是说，允许处理器对读进行重排序，但必须以程序规定的顺序进行写。

10.2.2 实现一致性的基本方案

（首先，共享数据的多个副本为什么产生？）

在支持Cache一致性的多处理机中，Cache提供两种功能：

- 共享数据的迁移

  减少了对远程共享数据的访问延迟，也减少了对共享存储器带宽的要求。

- 共享数据的复制

  不仅减少了访问共享数据的延迟，也减少了访问共享数据所产生的冲突。

1. Cache一致性协议（硬件实现）

在多个处理器中用来维护一致性的协议。

- 关键：跟踪**记录**共享数据块的**状态**
- 两类协议（采用不同的技术跟踪共享数据的状态）
  - 目录式协议（directory）
    - 物理存储器中数据块的共享状态被保存在一个称为目录的地方。（一个共享数据块的状态只在一个地方有）
  - 监听式协议（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。（每个Cache中都有）

- 所有Cache都可通过某种广播介质访问（Bus ,Switch），所有Cache控制器监听总线来判断它们是否有总线上请求的数据块。
- Limitation？

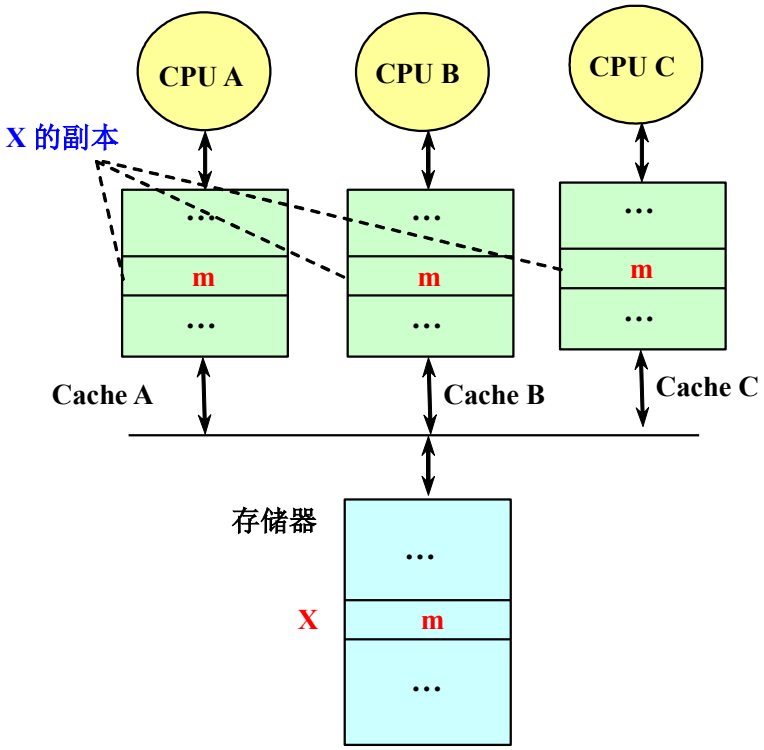
2. 采用两种方法来解决Cache一致性问题。

➢ 写作废协议

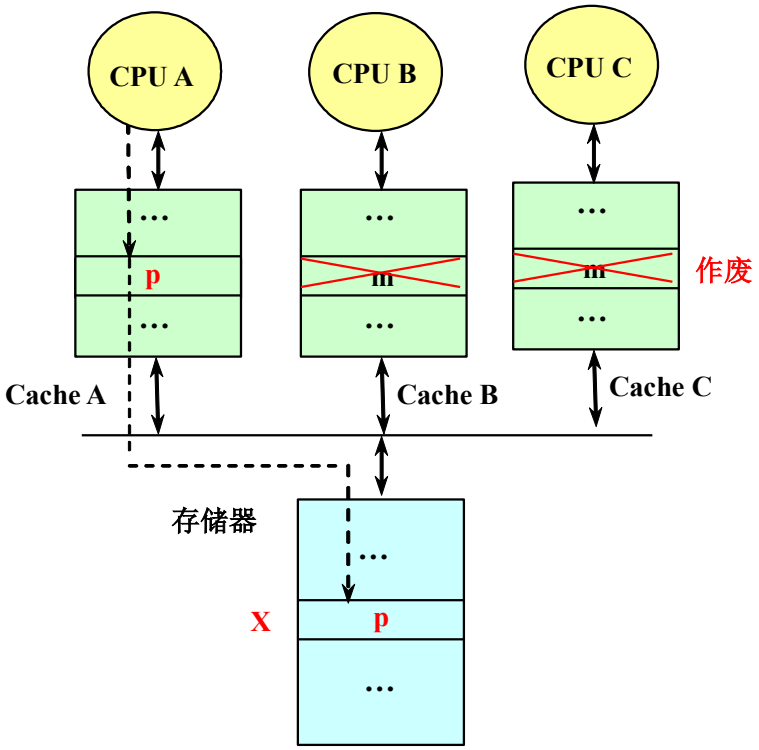
- 在处理器对某个数据项进行写入之前，保证它拥有对该数据项的唯一的（排它的）访问权。(作废其它的副本)

例：监听总线、写作废协议举例（采用写直达法）

初始状态：CPU A、CPU B、CPU C都有X的副本。在CPU A要对X进行写入时，需先作废CPU B和CPU C中的副本，然后再将p写入Cache A中的副本，同时用该数据更新主存单元X。



（a） CPU A 写入前



（b） CPU A 将 p 写入 X 后，作废其他 Cache 中的副本

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

Figure 4.4 An example of an invalidation protocol working on a snooping bus for a single cache block (X) with write-back caches. We assume that neither cache initially

➢ 写更新协议

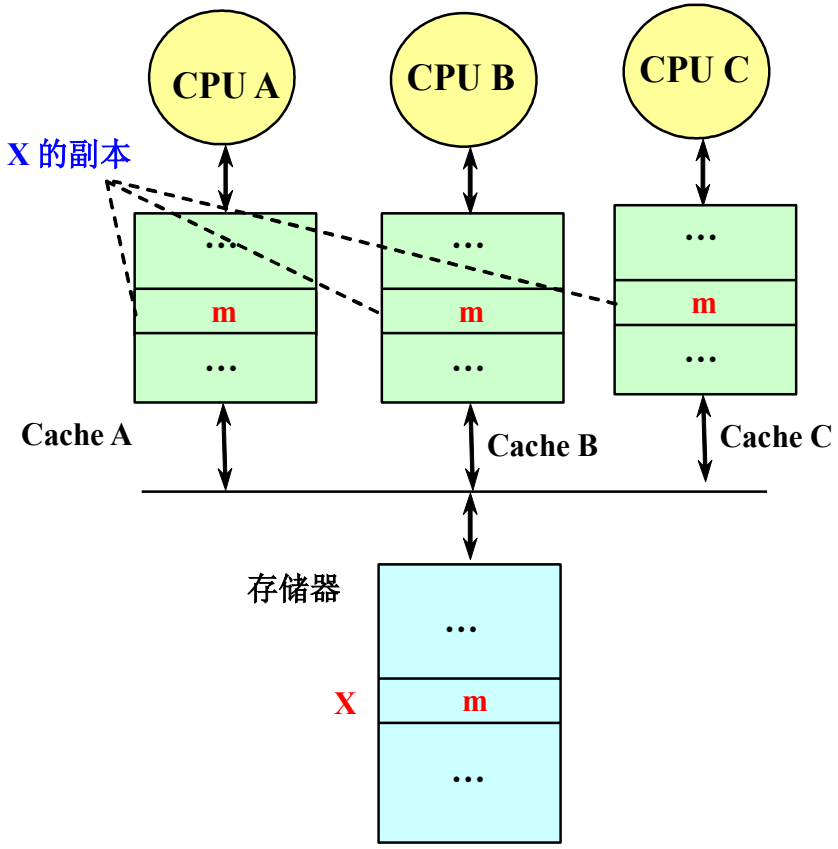
- 当一个处理器对某数据项进行写入时，通过广播使其它Cache中所有对应于该数据项的副本进行更新。

例：监听总线、写更新协议举例（采用写直达法）

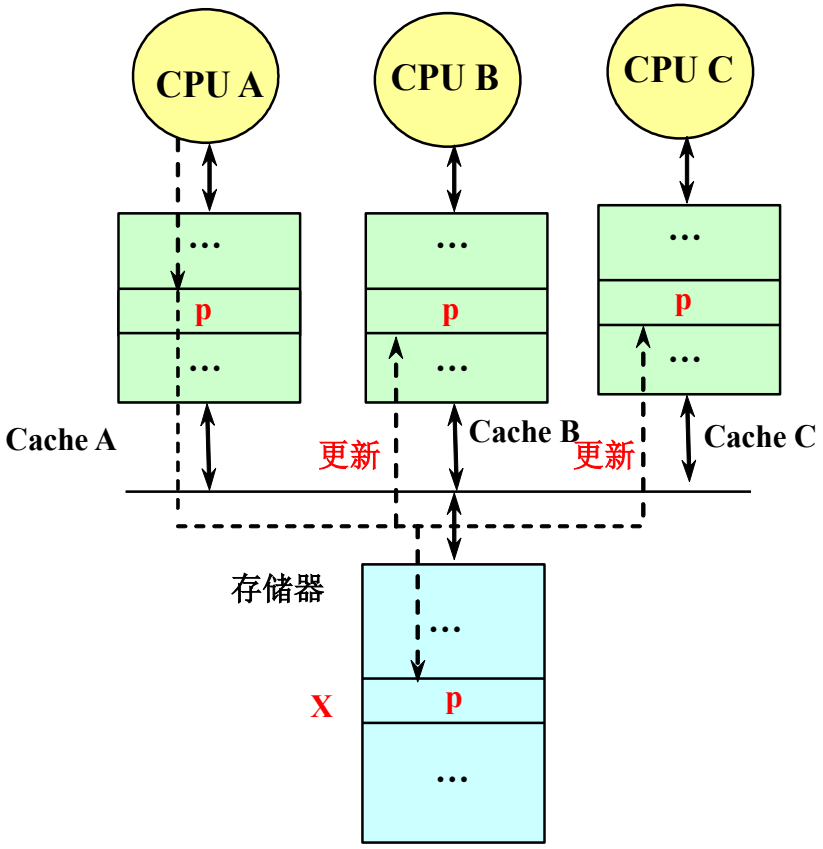
假设：3个Cache都有X的副本。

当CPU A将数据p写入Cache A中的副本时，将p广播给所有的Cache，这些Cache用p更新其中的副本。

由于这里是采用写直达法，所以CPU A还要将p写入存储器中的X。如果采用写回法，则不需要写入存储器。



（a） CPU A 写入前



（b） CPU A 将 p 写入 X 后，更新其他 Cache 中的副本

- 写更新和写作废协议性能上的差别主要来自：
  - 在对同一个数据进行多次写操作而中间无读操作的情况下，写更新协议需进行多次写广播操作，而写作废协议只需一次作废操作。
  - 在对同一Cache块的多个字进行写操作的情况下，写更新协议对于每一个写操作都要进行一次广播，而写作废协议仅在对该块的第一次写时进行作废操作即可。

    写作废是针对Cache块进行操作，而写更新则是针对字（或字节）进行。
  - 考虑从一个处理器A进行写操作后到另一个处理器B能读到该写入数据之间的延迟时间。

    写更新协议的延迟时间较小。

10.2.3 监听协议的实现

1. 监听协议的基本实现技术

- 实现写作废协议的关键：使用**广播介质**执行作废

  （假定采用的是总线）

  - 一个处理器取得总线访问权，将要作废的地址在总线上广播。
  - 所有处理器都一直在监听总线，它们检测总线上的地址是否在它们的Cache中。若在，则作废Cache中对应的数据。

- 写操作的串行化：由总线仲裁实现

2. Cache发送到总线上的消息主要有以下两种：
   - **RdMiss——读不命中**
   - **WtMiss——写不命中**
   - 需要通过总线找到相应数据块的最新副本，然后调入本地Cache中。
     - **写直达Cache：因为所有写入的数据都同时被写回主存，所以从主存中总可以取到其最新值。**
     - **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。**

（后面的讨论中，只考虑写回法Cache）

- 有的监听协议还增设了一条Invalidate消息，用来通知其他各处理器作废其Cache中相应的副本。
- Cache的标识（tag）可直接用来实现监听。
- 作废一个块只需将其有效位置为无效。
- 给每个Cache块增设一个共享位
  - 为“1”：该块是被多个处理器所共享
  - 为“0”：仅被某个处理器所独占

块的拥有者：拥有该数据块的唯一副本的处理器。

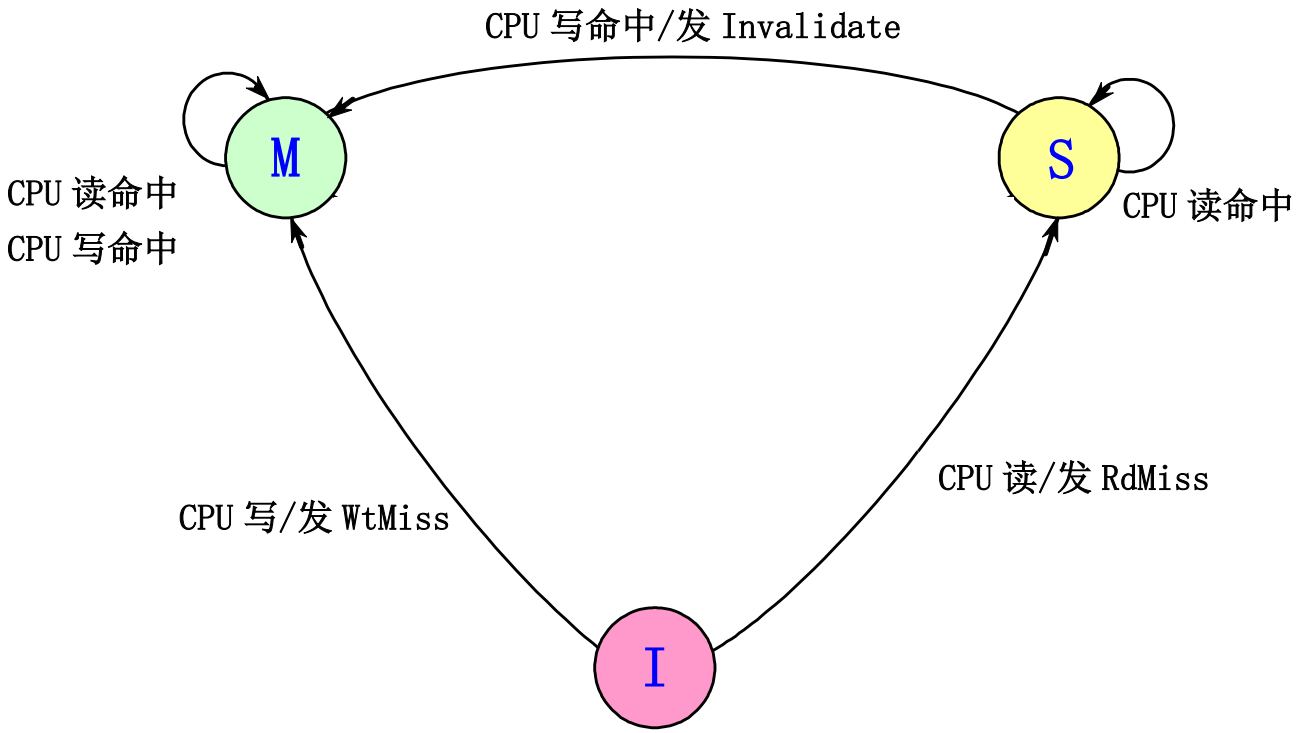
3. 监听协议举例

- 在每个结点内嵌入一个有限状态控制器。
  - **该控制器响应处理器或总线的请求，改变所选Cache块的状态，使用总线访问或作废数据。**
- 每个数据块的状态取以下3种状态中的一种：
  - 无效（简称**I**）：**Cache中该块的内容为无效。**
  - 共享（简称**S**）：**该块可能处于共享状态。**
    - 在多个处理器中都有副本。这些副本都相同，且与存储器中相应的块相同。
  - 已修改（简称**M**）：**该块已经被修改过，并且还没写入存储器（exclusive）。**

    （块中的内容是最新的，系统中唯一的最新副本）

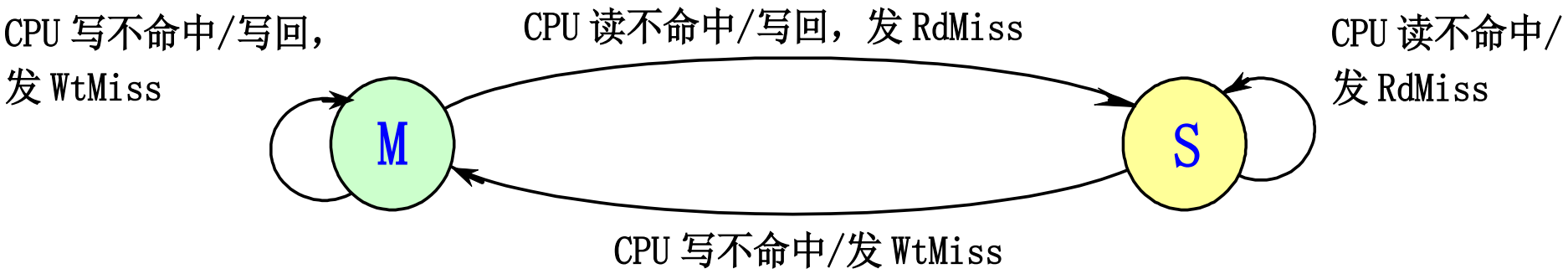
➢ 响应来自处理器的请求

- 不发生替换的情况



写作废协议中（采用写回法），Cache块的状态转换图

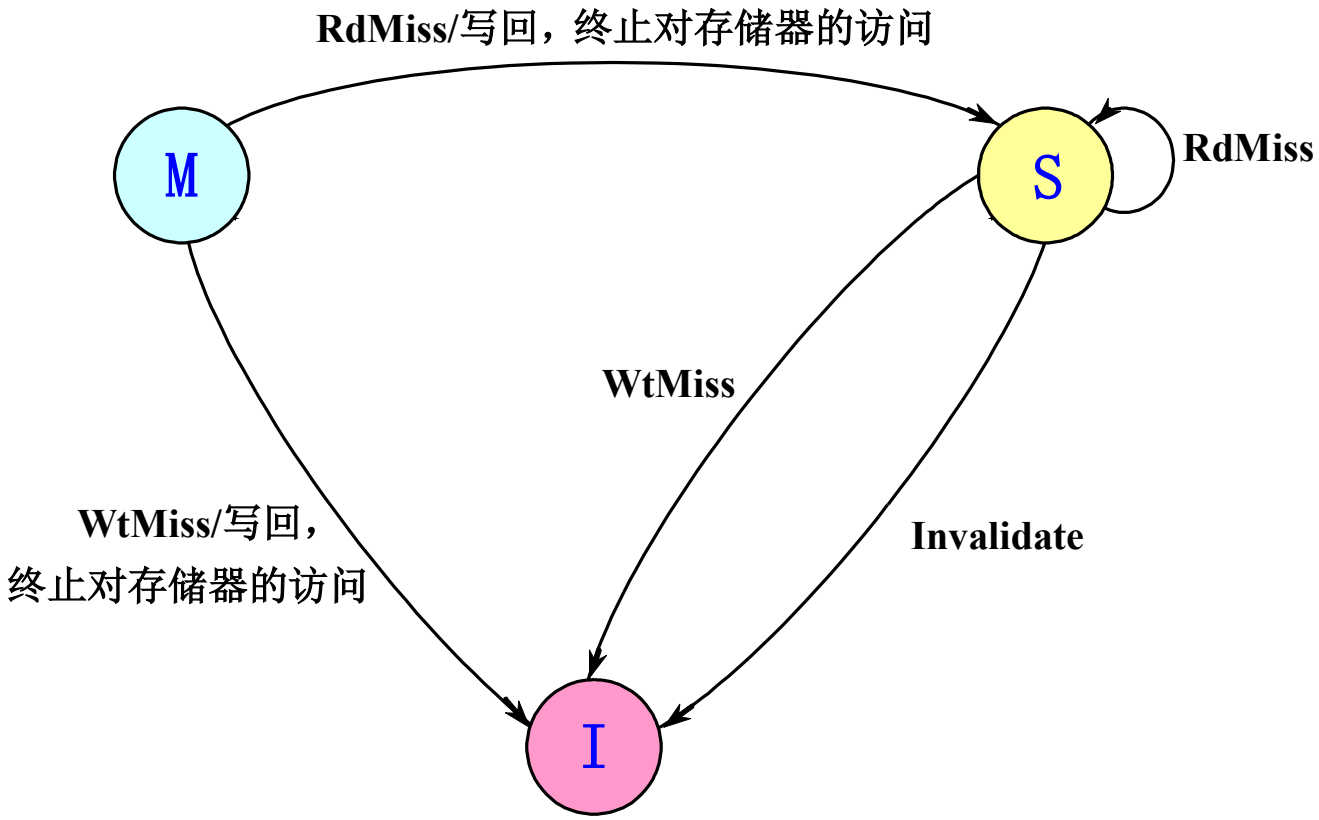
- 发生替换的情况



写作废协议中（采用写回法），Cache块的状态转换图

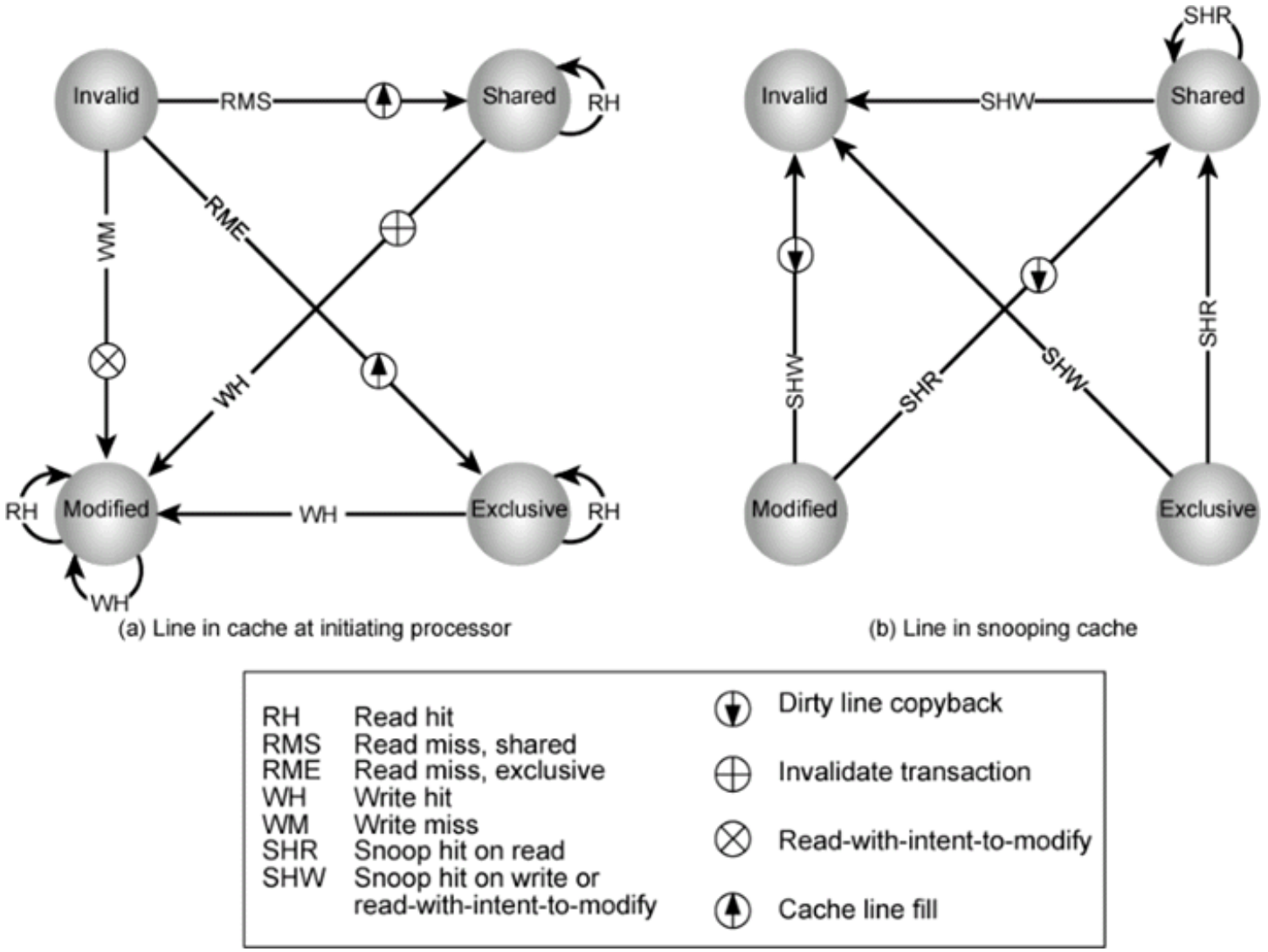
➢ 响应来自总线的请求

- 每个处理器都在监视总线上的消息和地址，当发现有与总线上的地址相匹配的Cache块时，就要根据该块的状态以及总线上的消息，进行相应的处理。



写作废协议中（采用写回法），Cache块的状态转换图

- **Modified**：cache行已被修改，且仅位于这个cache
- **Exclusive**：cache行同主存，且仅位于这个cache中
- **Shared**：cache行同主存，且位于多个cache中
- **Invalid**：此cache行不含有效数据



10.3 分布式共享存储器系统结构

10.3.1 目录协议的基本思想

- 广播和监听的机制使得监听一致性协议的可扩放性很差。
- 替代监听协议的一致性协议。

（采用目录协议）

1. 目录协议

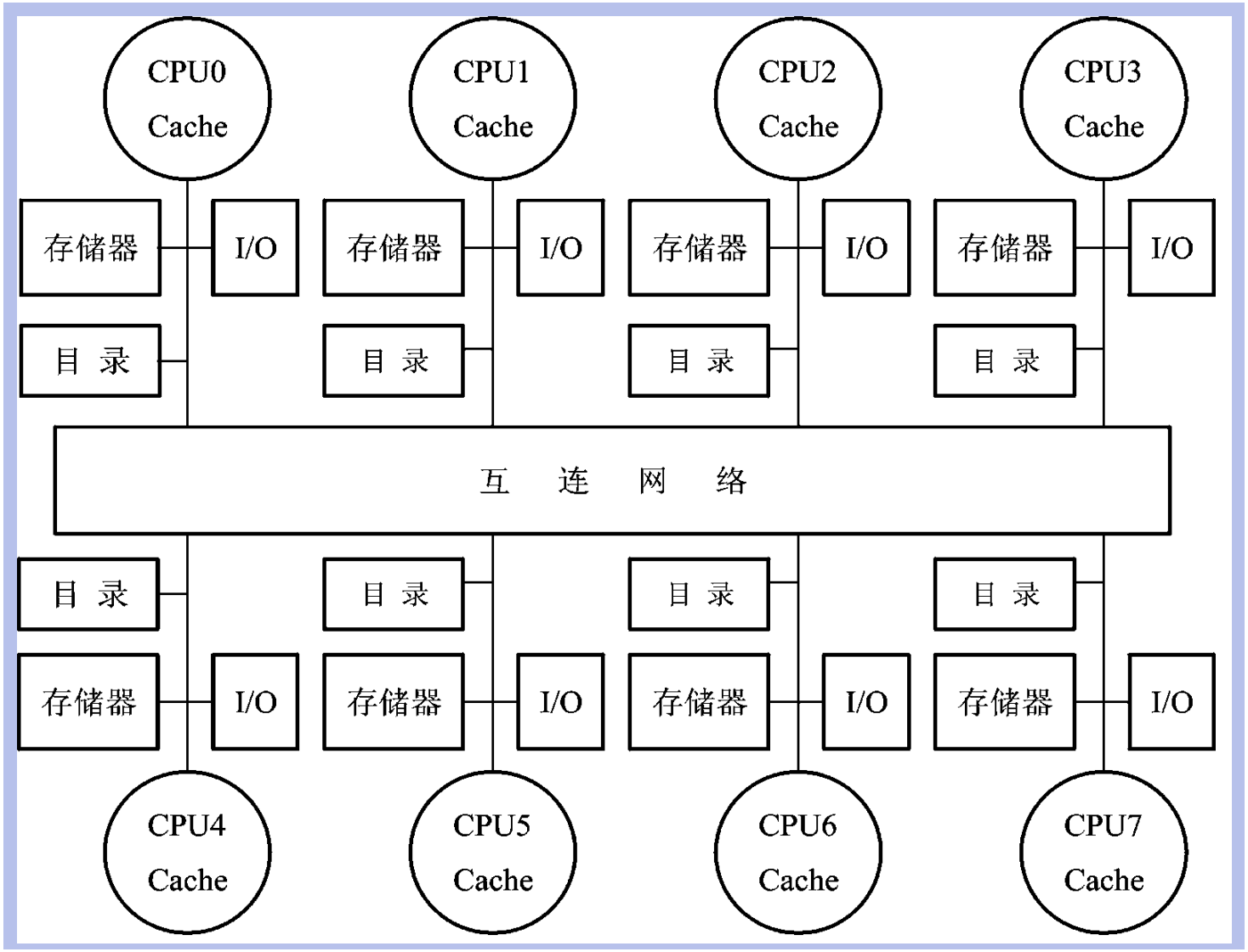
- 目录：一种集中的数据结构。对于**存储器**中的每一个**可以调入Cache的数据块**，在目录中设置一条目录项，用于记录该块的状态以及哪些Cache中有副本等相关信息。

- 特点：

  对于任何一个数据块，都可以快速地在唯一的一个位置中找到相关的信息。这使一致性协议避免了广播操作。

- 位向量：记录哪些Cache中有副本。
  - 每一位对应于一个处理器。
  - 长度与处理器的个数成正比。
  - 由位向量指定的处理机的集合称为共享集S。
- 分布式目录
  - 目录与存储器一起分布到各结点中，从而对于不同目录内容的访问可以在不同的结点进行。

对每个结点增加目录后的分布式存储器多处理机



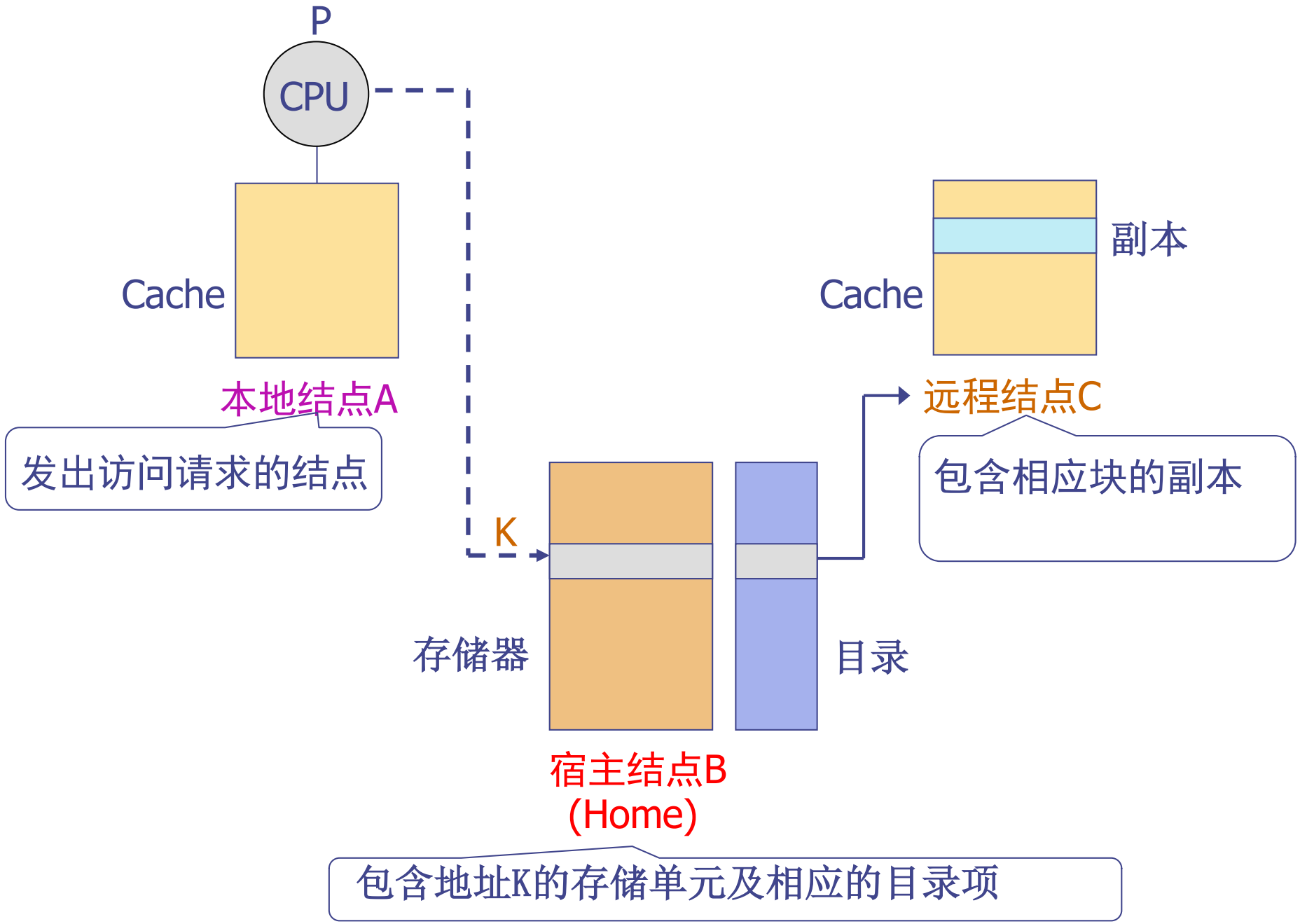
2. 在目录协议中，存储块的状态有3种：

- 未缓冲：该块尚未被调入Cache。所有处理器的Cache中都没有这个块的副本。
- 共享：该块在一个或多个处理机上有这个块的副本，且这些副本与存储器中的该块相同。
- 独占：仅有一个处理机有这个块的副本，且该处理机已经对其进行了写操作，所以其内容是最新的，而存储器中该块的数据已过时。

这个处理机称为该**块的拥有者**。

3. 本地结点、宿主结点以及远程结点的关系

- 本地结点：发出访问请求的结点
- 宿主结点：包含所访问的存储单元及其目录项的结点
- 远程结点：包含Cache block的副本



本地结点、宿主结点以及远程结点的关系

4. 在结点之间发送的消息

- 本地结点发给宿主结点（目录）的消息

说明：括号中的内容表示所带参数。

P：发出请求的处理机编号

K：所要访问的地址

- **RdMiss（P，K）**

处理机P读取地址为A的数据时不命中，请求宿主结点提供数据（块），并要求把P加入共享集。

- **WtMiss（P，K）**

处理机P对地址A进行写入时不命中，请求宿主结点提供数据，并使P成为所访问数据块的独占者。

- **Invalidate（K）**

  请求向所有拥有相应数据块副本（包含地址*K*）的远程Cache发Invalidate消息，作废这些副本。

➢ 宿主结点（目录）发送给远程结点的消息

- **Invalidate（K）**

  作废远程Cache中包含地址K的数据块。

- **Fetch（K）**

  从远程Cache中取出包含地址*K*的数据块，并将之送到宿主结点。把远程Cache中那个块的状态改为“共享”。

- **Fetch&Inv（K）**

- 从远程Cache中取出包含地址*K*的数据块，并将之送到宿主结点。然后作废远程Cache中的那个块。

➢ 宿主结点发送给本地结点的消息

DReply（D）

- D表示数据内容。
- 把从宿主存储器获得的数据返回给本地Cache。

➢ 远程结点发送给宿主结点的消息

WtBack（K，D）

- 把远程Cache中包含地址*K*的数据块写回到宿主结点中， 该消息是远程结点对宿主结点发来的“取数据”或“取/作废”消息的响应。

- 本地结点发送给被替换块的宿主结点的消息
  - **MdSharer（P，K）**

    用于当本地Cache中需要替换一个包含地址K的块、且该块未被修改过的情况。这个消息发给该块的宿主结点，请求它将P从共享集中删除。如果删除后共享集变为空集，则宿主结点还要将该块的状态改变为“未缓存”（U）。

  - **WtBack2（P，K，D）**

    用于当本地Cache中需要替换一个包含地址K的块、且该块已被修改过的情况。这个消息发给该块的宿主结点，完成两步操作：①把该块写回；②进行与MdSharer相同的操作。

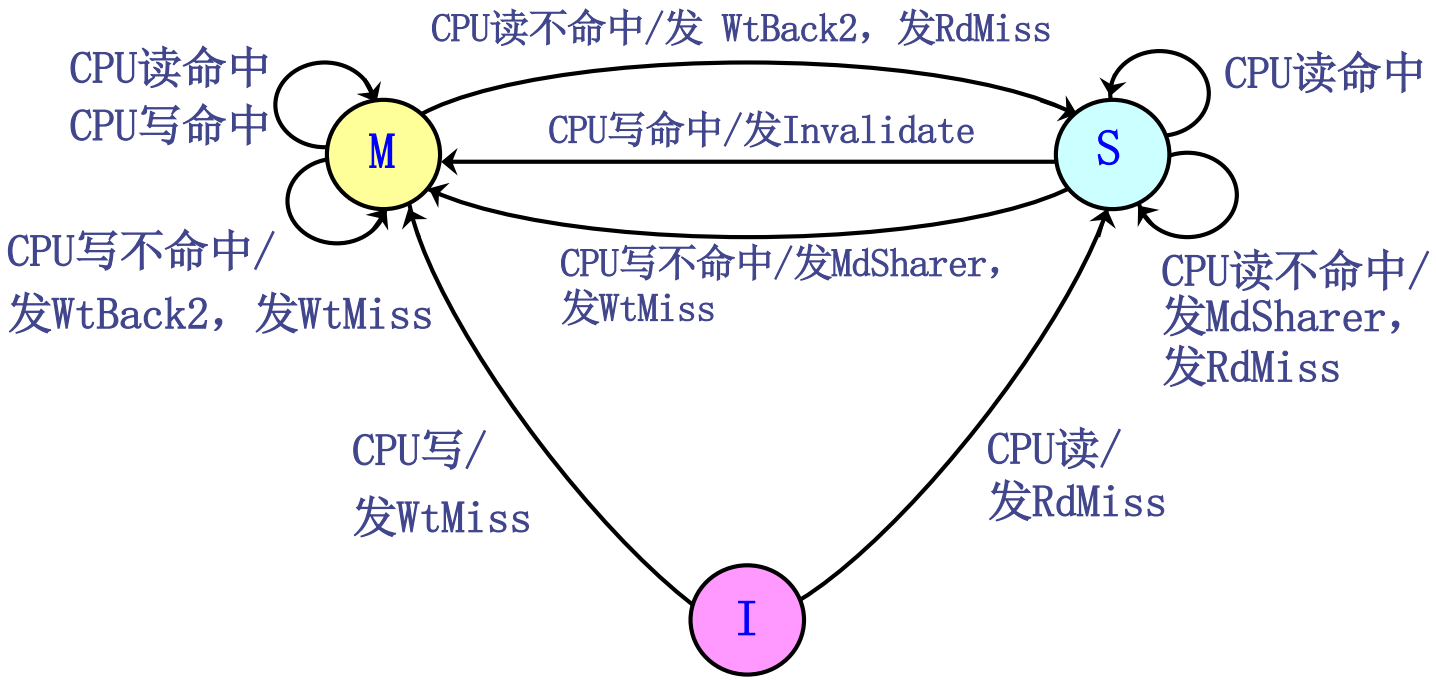
Message type	Source	Destination	Message contents	Function of this message
Read miss	local cache	home directory	P, A	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	local cache	home directory	P, A	Processor P has a write miss at address A; request data and make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

10.3.2 目录协议实例

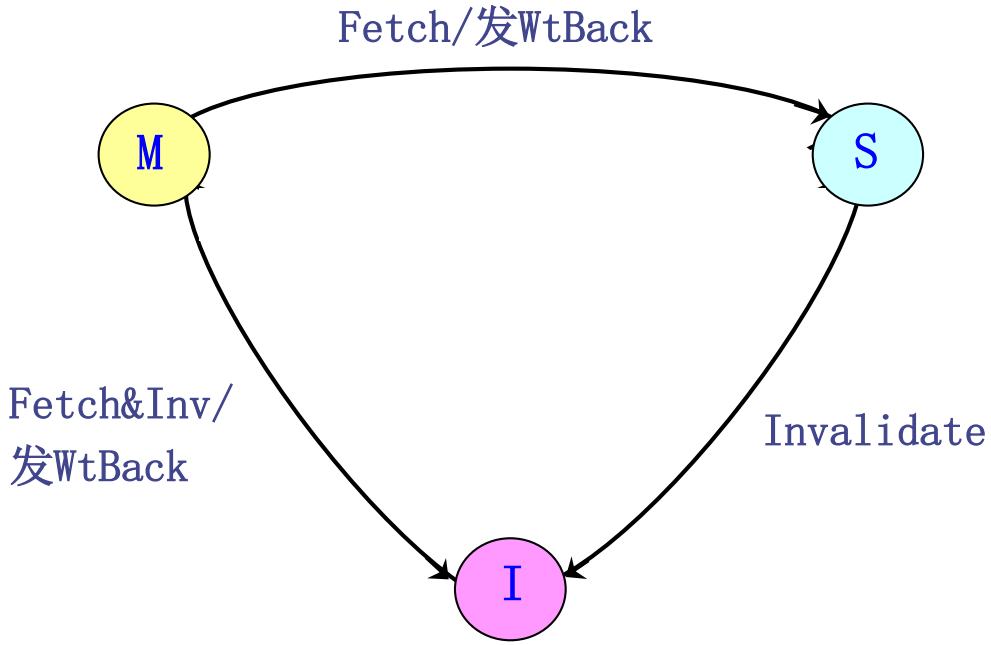
- 本地结点把请求发给宿主结点中的目录，再由目录控制器有选择地向远程结点发出相应的消息。
  - 发出的消息会产生两种不同类型的动作：
    - 更新目录状态
    - 使远程结点完成相应的操作

1. 在基于目录协议的系统中， Cache块的状态转换图。

- 响应本地Cache CPU请求

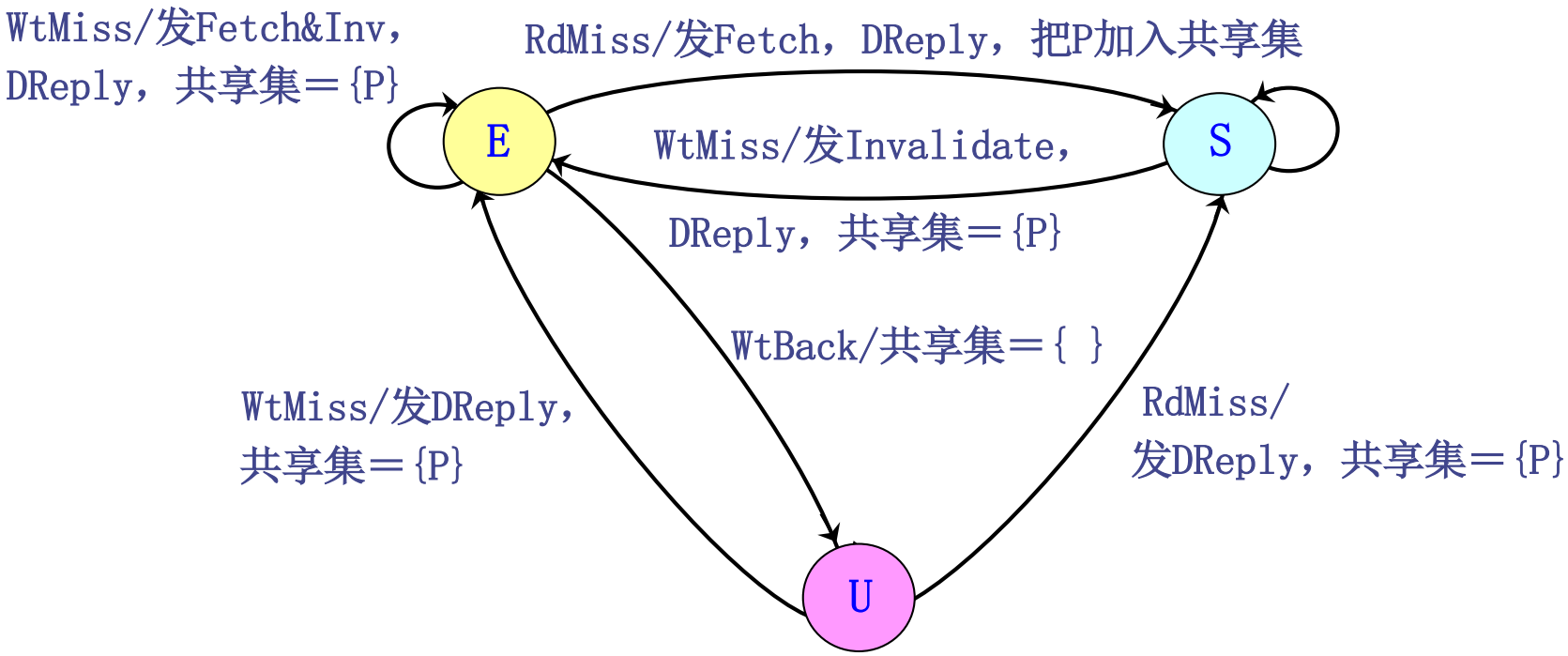


- 远程结点中Cache块响应来自宿主结点的请求的状态转换图



2. 目录的状态转换及相应的操作

- 目录中存储器块的状态有3种
  - 未缓存
  - 共享
  - 独占
- 位向量记录拥有其副本的处理器的集合。这个集合称为共享集合。
- 目录请求需要更新共享集合，读取共享集合执行作废操作。
- 目录可能接收到3种不同的请求
  - 读不命中
  - 写不命中
  - 数据写回



U：未缓存（Uncached）　　S：共享（Shared）：只读

E：独占（Exclusive）：可读写　　P：本地处理器

目录的状态转换及相应的操作

- 当一个块处于未缓存状态时，对该块发出的请求及处理操作为：
  - RdMiss（读不命中）
    - 将所要访问的存储器数据送往请求方处理机，且该处理机成为该块的唯一共享结点，本块的状态变成共享。
  - WtMiss（写不命中）
    - 将所要访问的存储器数据送往请求方处理机，该块的状态变成独占，表示该块仅存在唯一的副本。其共享集合仅包含该处理机，指出该处理机是其拥有者。

- 当一个块处于共享状态时，其在存储器中的数据是当前最新的，对该块发出的请求及其处理操作为：
  - **RdMiss**
    - 将存储器数据送往请求方处理机，并将其加入共享集合。
  - **WtMiss**
    - 将数据送往请求方处理机，对共享集合中所有的处理机发送作废消息，且将共享集合改为仅含有该处理机，该块的状态变为独占。

- 当某块处于独占状态时，该块的最新值保存在共享集合所指出的唯一处理机（拥有者）中。

  有三种可能的请求：

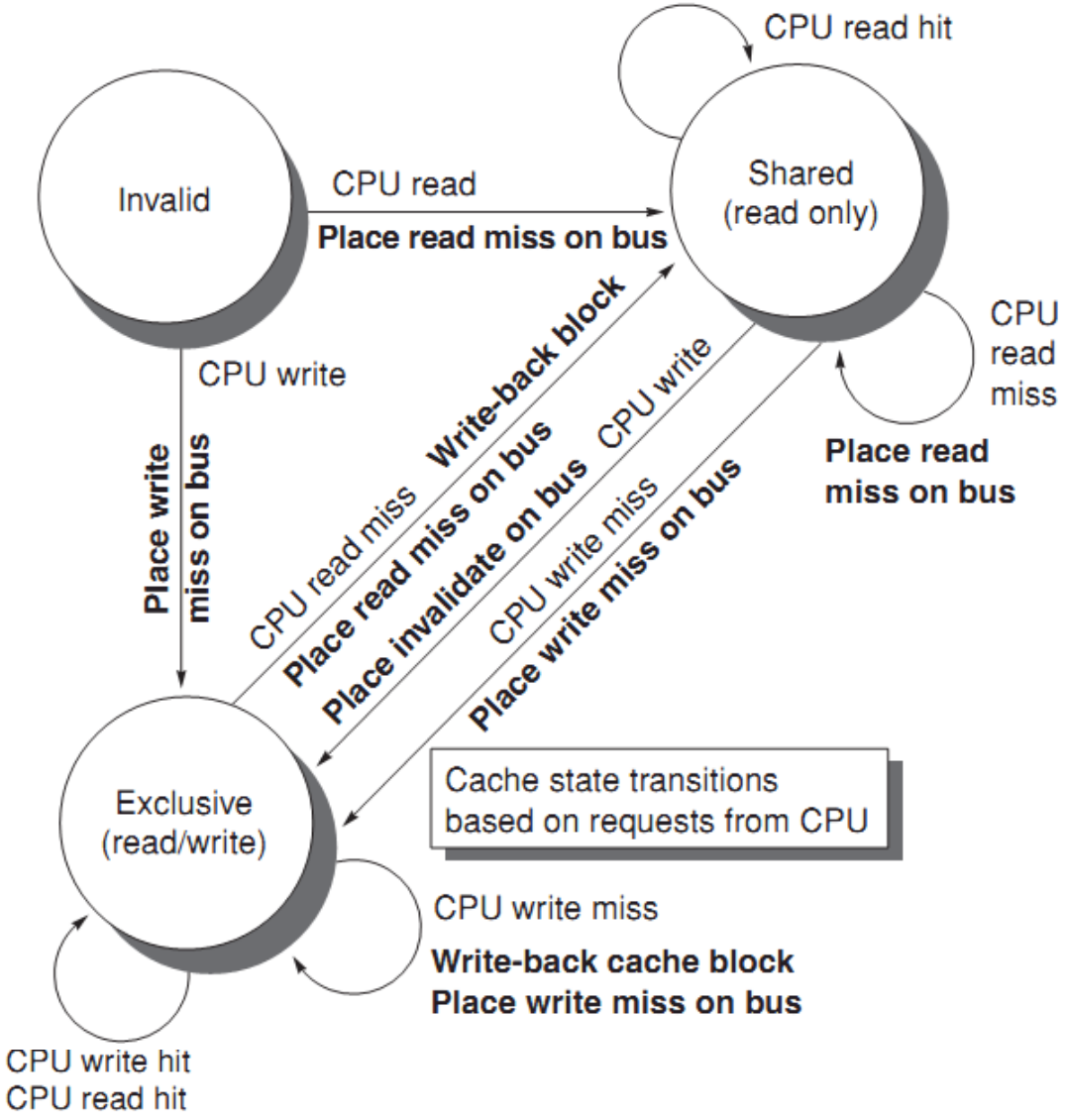
  - **RdMiss**
    - 将“取数据”的消息发往拥有者处理机，将它所返回给宿主结点的数据写入存储器，并进而把该数据送回请求方处理机，将请求方处理机加入共享集合。
    - 此时共享集合中仍保留原拥有者处理机（因为它仍有一个可读的副本）。
    - 将该块的状态变为共享。

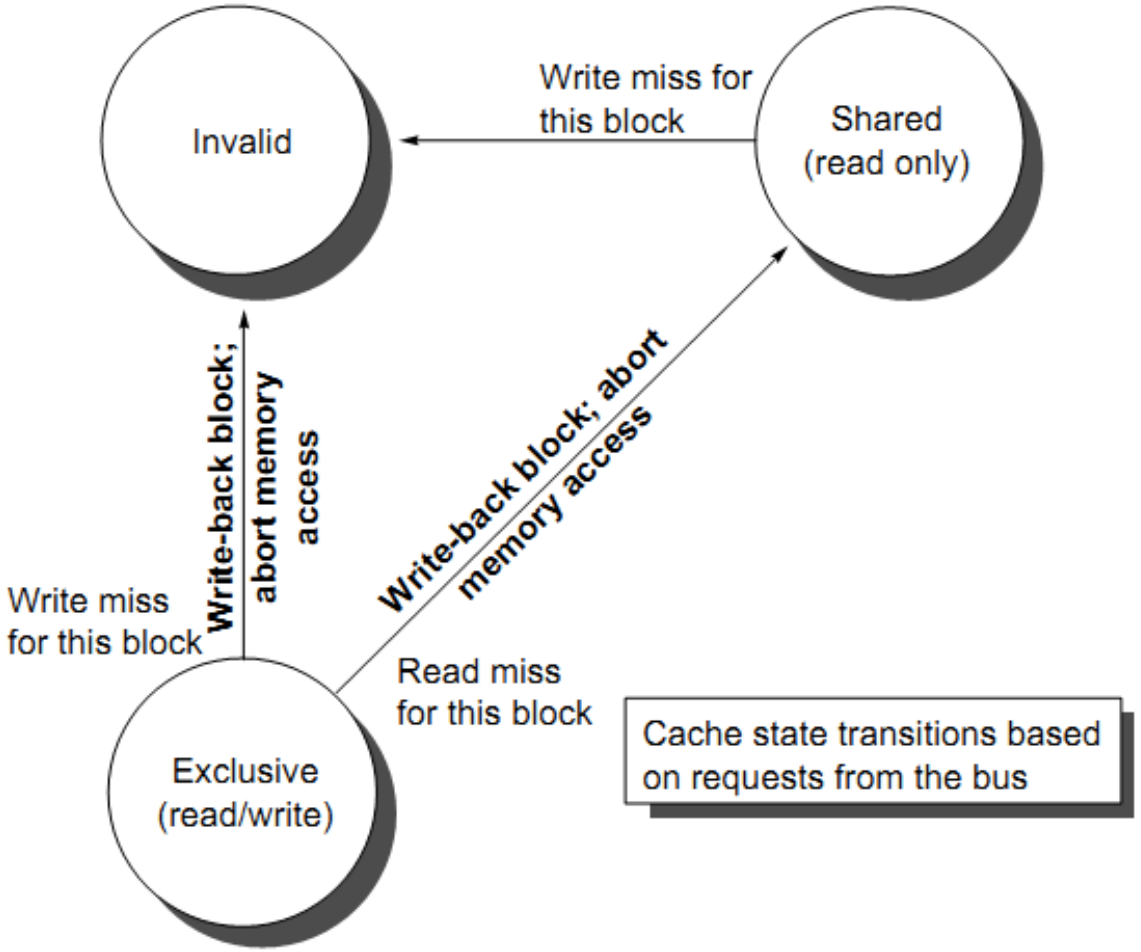
- **WtMiss**
  - 该块将有一个新的拥有者。
  - 给旧的拥有者处理机发送消息，要求它将数据块送回宿主结点写入存储器，然后再从该结点送给请求方处理机。
  - 同时还要把旧拥有者处理机中的该块作废。把请求处理机加入共享者集合，使之成为新的拥有者。
  - 该块的状态仍旧是独占。
- **WtBack**（写回）
  - 当一个块的拥有者处理机要从其Cache中把该块替换出去时，必须将该块写回其宿主结点的

存储器中，从而使存储器中相应的块中存放的数据是最新的（宿主结点实际上成为拥有者）；

- 该块的状态变成未缓冲，其共享集合为空。

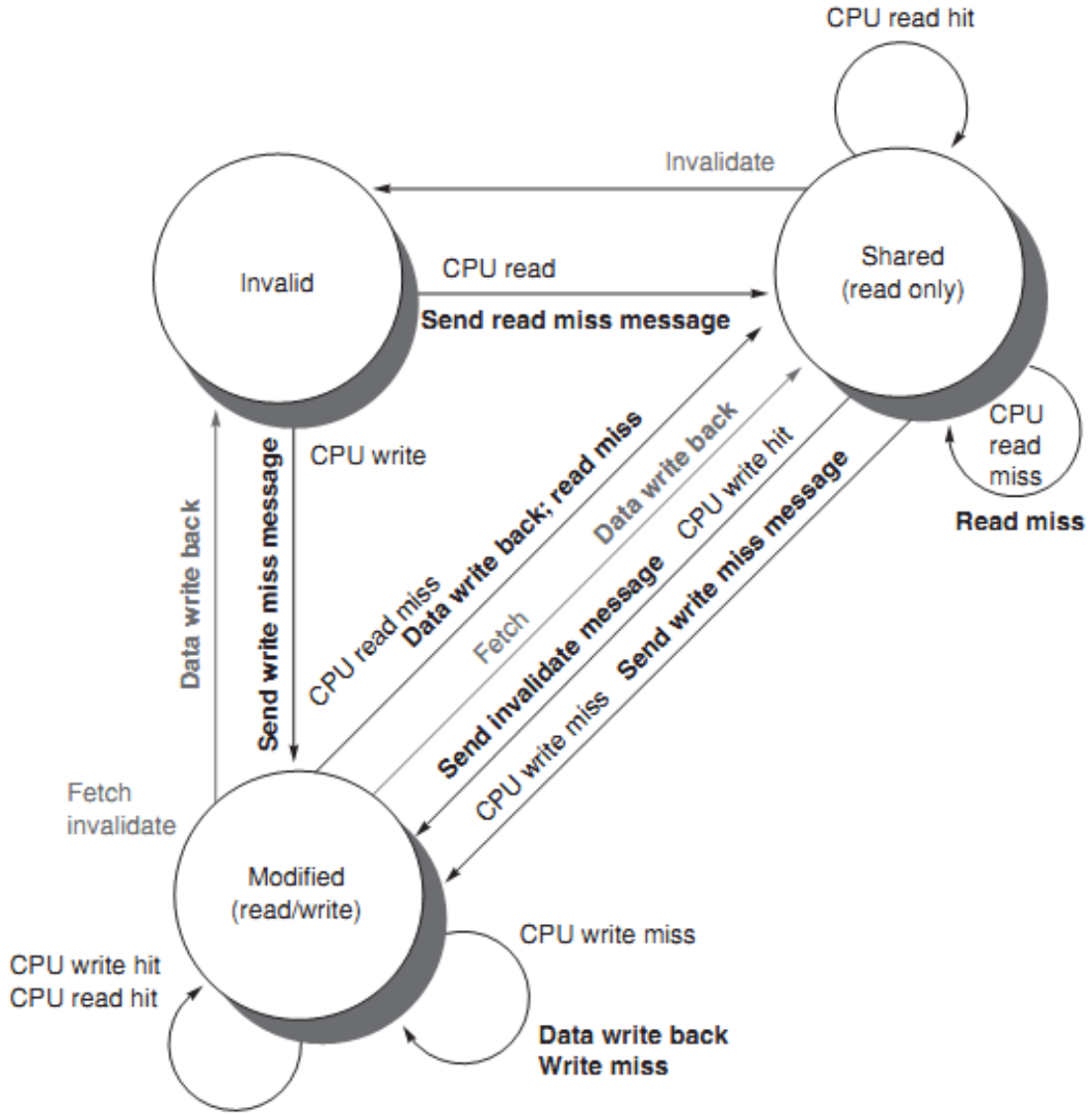
A write invalidate, cache coherence protocol for a private write-back cache showing the states and state transitions for each block in the cache





A write-invalidate, cache-coherence protocol for a write-back cache showing the states and state transitions for each block in the cache.

State transition diagram for an individual cache block in a directory-based system



The state transition diagram for the directory