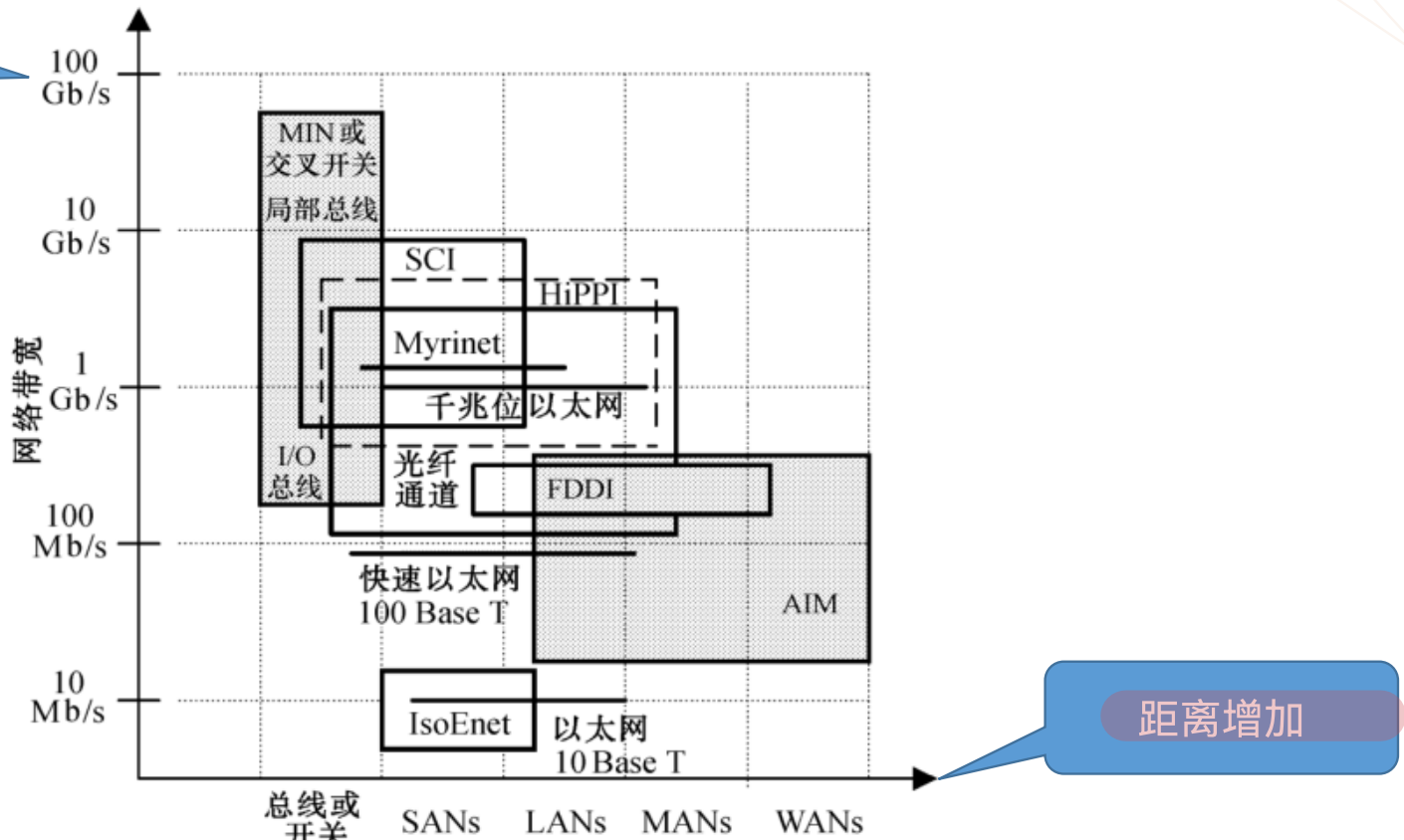
第5章 – 线程级并行：多处理机

1. 计算机构成的两个最基本的要素

- 点：包括小到CPU内部的寄存器、ALU、控制器，到存储模块、外设，乃至多处理机的计算节点，都可以视为点；
- 互连网络：按照一定拓扑结构和控制方式，将点连接起来。

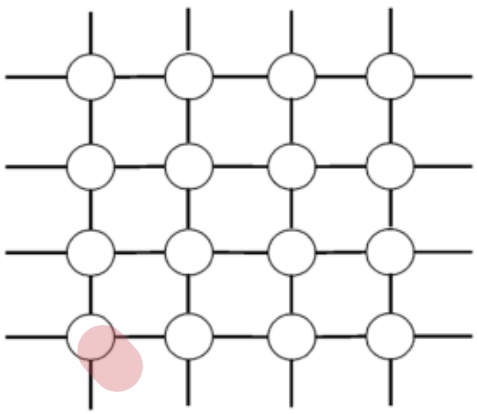
系统互连

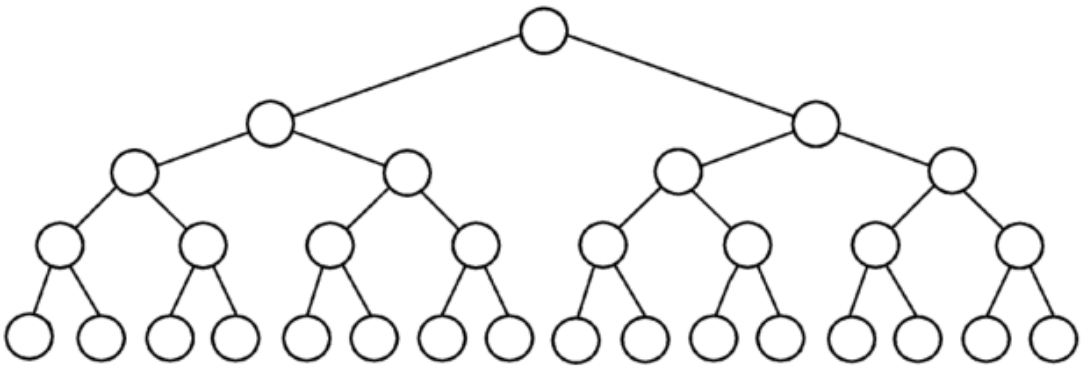
单位时间
信息量增加

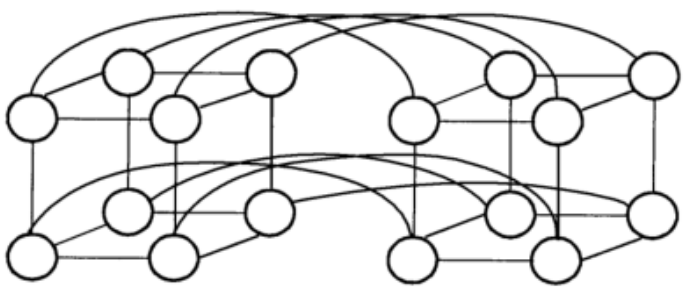


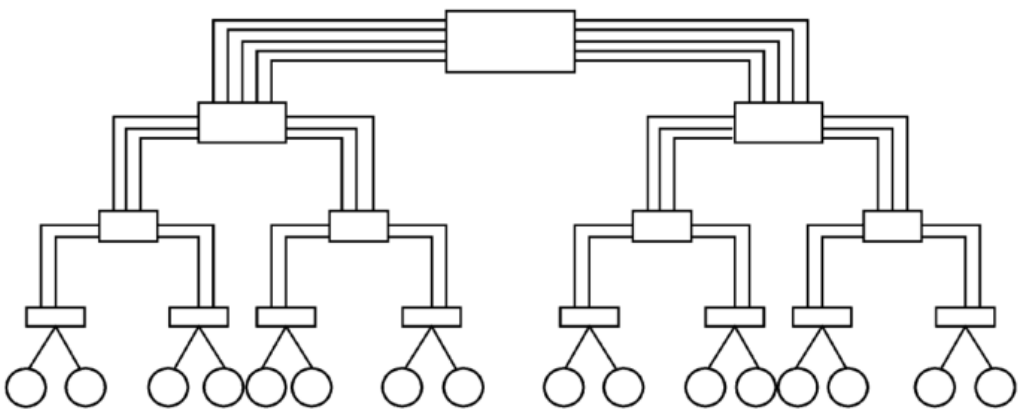
距离增加

静态互连网络：指处理单元间有固定连接的一类网络，在程序执行期间，这种点到点的链接保持不变。

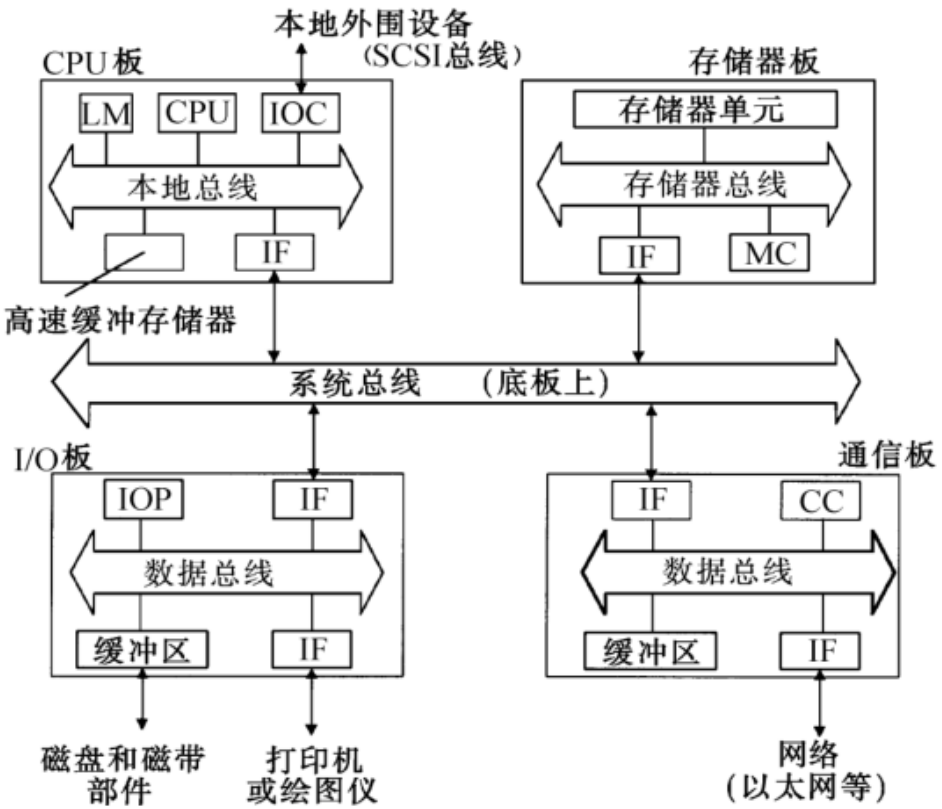




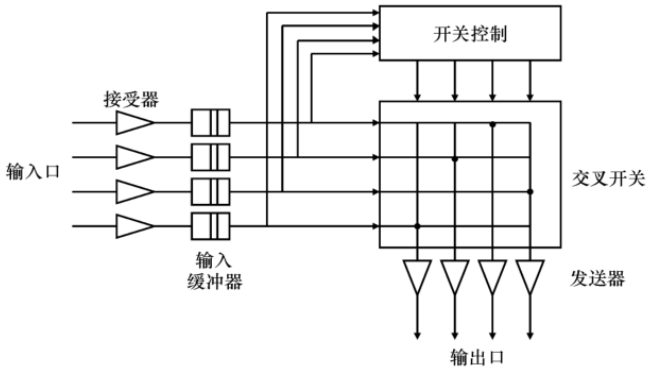




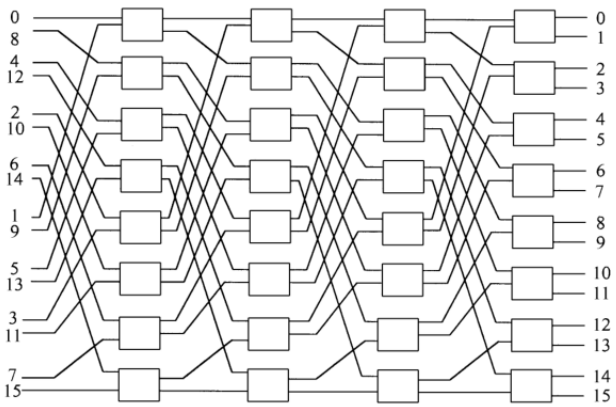
动态互连网络：由开关单元构成，可以按照应用程序的要求动态的改变连接组态。



各种总线



交叉开关网络

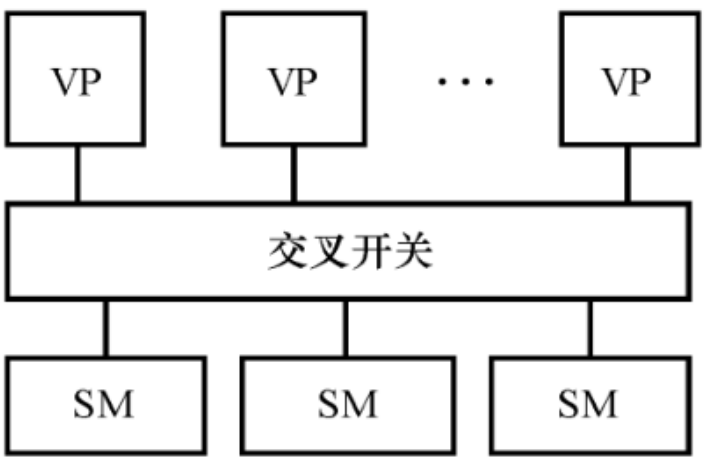


多级互连网络

2. 几种基本的MIMD并行机结构模型

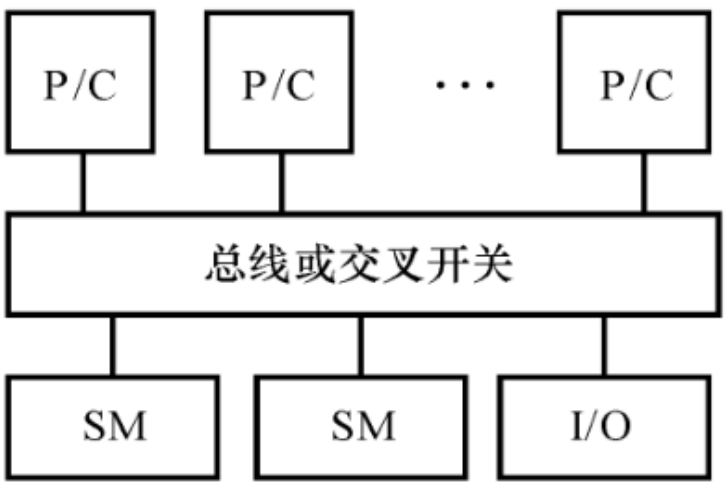
- PVP，Parallel Vector Processor, 并行向量处理机
- SMP, Symmetric Multiprocessor, 对称多处理机
- MPP, Massively Parallel Processor, 大规模并行处理机
- DSM，Distributed Shared Memory, 分布式共享存储多处理机
- COW，Cluster of Workstations, 工作站集群

PVP——并行向量处理机



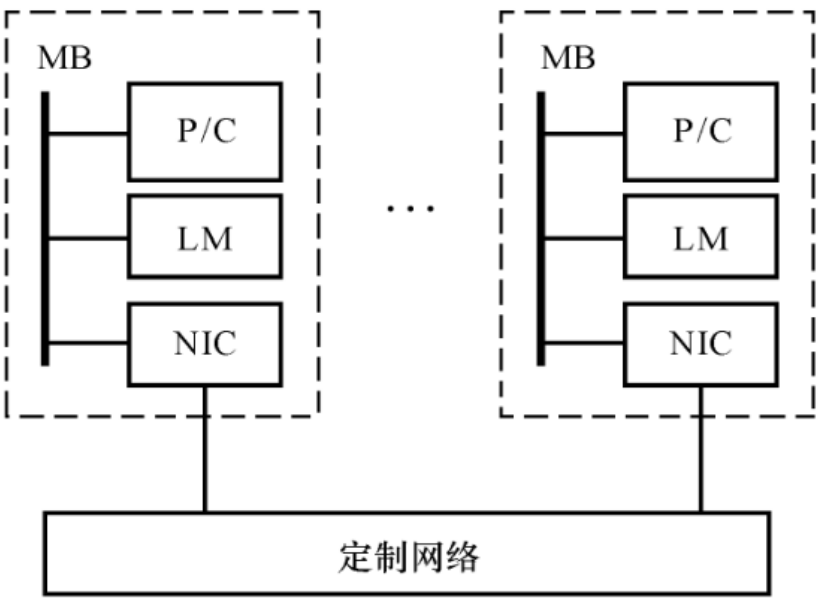
- 这样的系统中包含了少量的高性能专门设计定制的向量处理器ＶＰ，每个至少具有１Ｇflops的处理能力；
- 存储器以兆字节每秒的速度向处理器提供数据。
- 向量处理器VP和共享存储模块通过高带宽的交叉开关网络互连；
- 这样的机器通常不使用高速缓存，而是使用大量的向量寄存器和指令缓冲器；
- 例如：Cray90、NECSX-4和我国的银河１号等都是ＰＶＰ。

SMP——对称多处理机



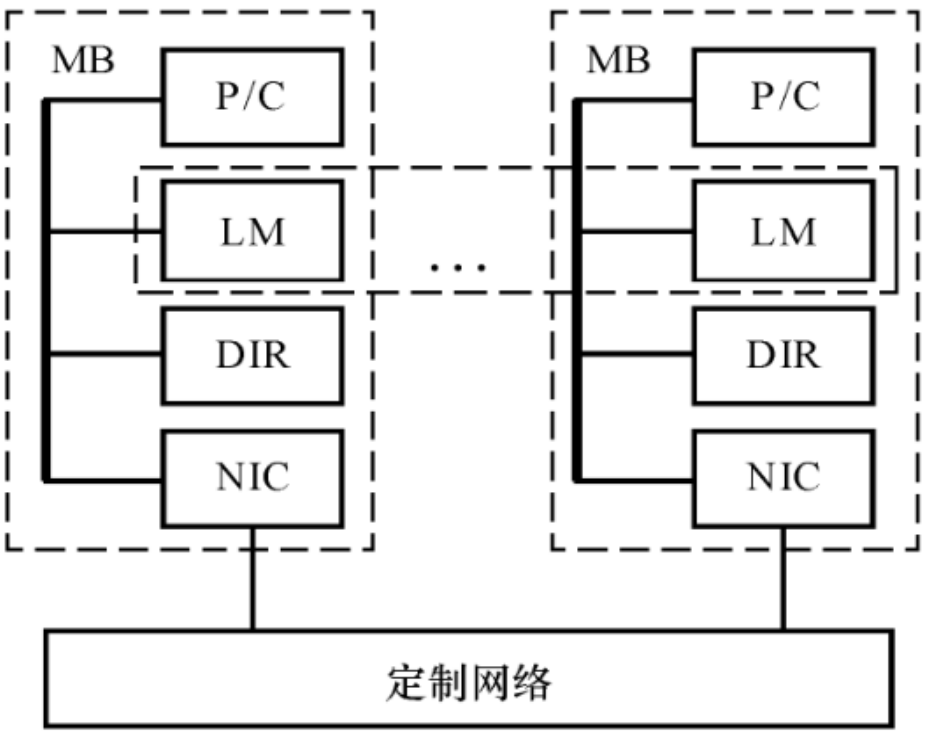
- ＳＭＰ系统使用商品微处理器（具有片上或外置高速缓存）；
- 它们经由高速总线（或交叉开关）连向共享存储器和I/O；
- 这种机器主要应用于商务，例如数据库、在线事务处理系统和数据仓库等；
- 重要的是系统是对称的，每个处理器可等同的访问共享存储器、Ｉ／Ｏ设备和操作系统服务。正是对称，才能开拓较高的并行度；也正是共享存储，限制系统中的处理器不能太多（一般少于６４个），同时总线和交叉开关互连一旦作成也难于扩展。
- 例如：IBM R50、SGI Power Challenge、DEC Alpha服务器8400和我国的曙光１号等都是这种类型的机器。

MPP——大规模并行处理机



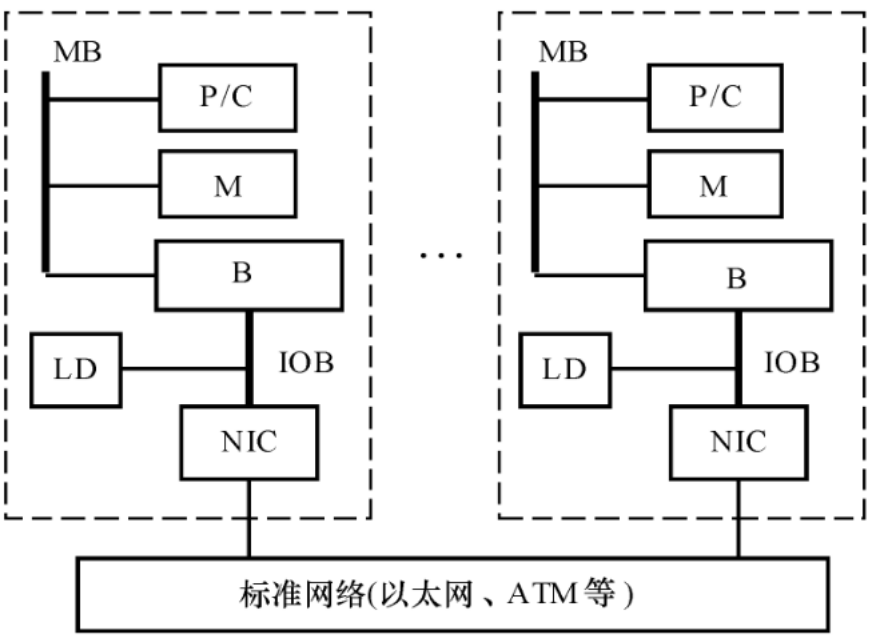
- ＭＰＰ一般是指超大型计算机系统；
- 处理节点采用商品微处理器；每个节点上有自己的局部存储器；采用高通信带宽和低延迟的互连网络（专门设计和定制的）进行节点互连；
- 能扩放至成百上千乃至上万个处理器；
- 它是一种异步的ＭＩＭＤ机器，程序系由多个进程组成，每个都有其私有地址空间，进程间采用传递消息相互作用；
- ＭＰＰ的主要应用是科学计算、工程模拟和信号处理等以计算为主的领域。
- 例如：Intel Paragon、Cray T3E、IntelOption Red和我国的曙光-１０００等都是这种类型的机器。

DSM——分布式共享存储多处理机



- 物理上有分布在各节点中的局部存储器，但是对用户而言，系统硬件和软件提供了逻辑上单地址的编程空间。
- 高速缓存目录DIR用以支持分布高速缓存的一致性。
- ＤＳＭ 相对于 ＭＰＰ的优越性是编程较容易。
- 例如：Stanford DASH、Cray T3D和SGI/Cray Origin 2000等。

COW——工作站集群

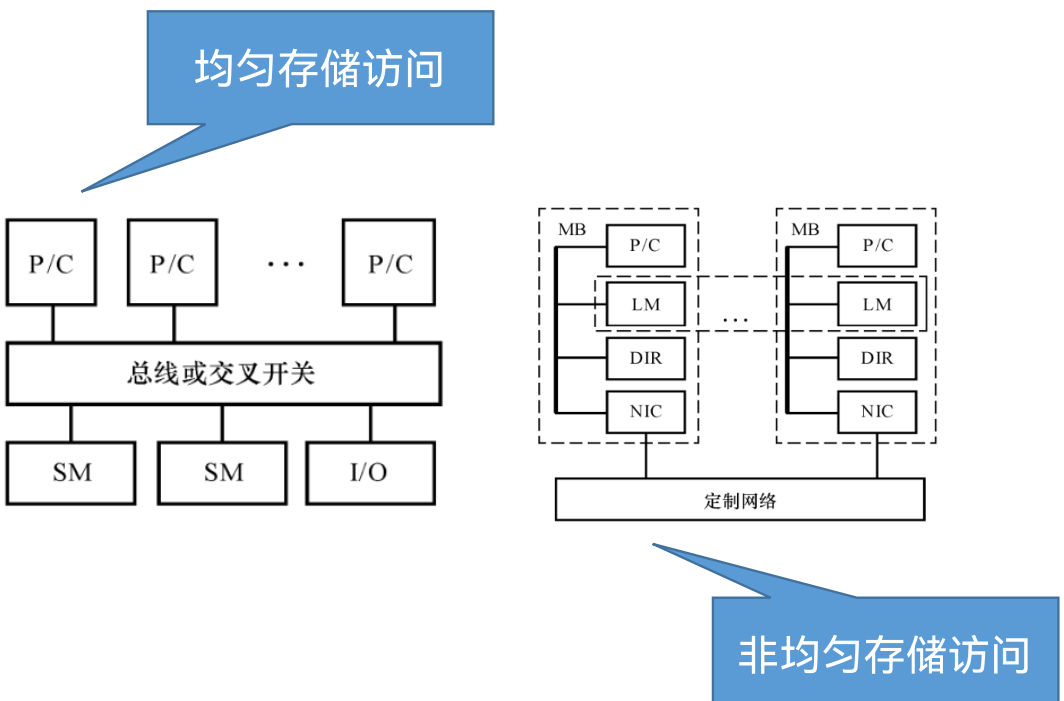


- 在有些情况下，机群往往是低成本的变形的ＭＰＰ；
- ＣＯＷ 的重要界线和特征是：
  ①ＣＯＷ 的每个节点都是一个完整的工作站（不包括监视器、键盘、鼠标等），这样的节点有时叫作“无头工作站”，一个节点也可以是一台ＰＣ或ＳＭＰ；
  ②各节点通过一种低成本的商品（标准）网络（如以太网、ＦＤＤＩ和 ＡＴＭ 开关等）互连（有的商用机群也使用定做的网络）；
  ③各节点内总是有本地磁盘，而 ＭＰＰ节点内却没有；
  ④节点内的网络接口是松散耦合到Ｉ／Ｏ 总线上的，而 ＭＰＰ内的网络接口是连到处理节点的存储总线上的，因而可谓是紧耦合式的；
  ⑤一个完整的操作系统驻留在每个节点中，而 ＭＰＰ中通常只是个微核，ＣＯＷ 的操作系统是工作站ＵＮＩＸ，加上一个附加的软件层以支持单一系统映像、并行度、通信和负载平衡等。
- Berkeley NOW、Alpha Farm、Digital Truclster等都是ＣＯＷ 结构。

3. 从存储角度来看MIMD

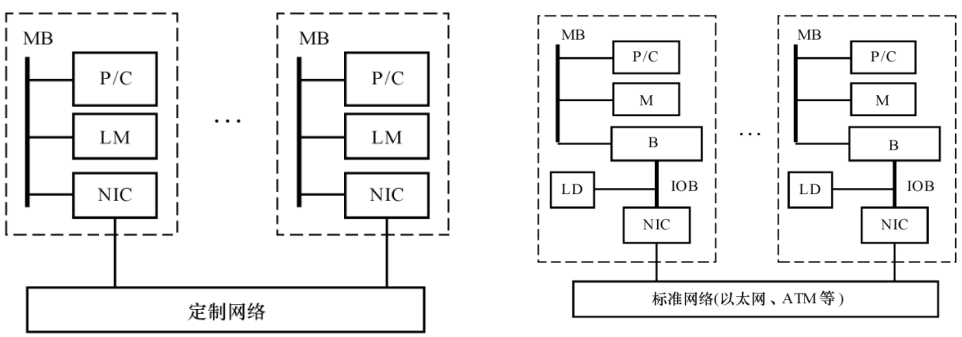
- 单地址空间共享存储

存储器可以是物理上集中的或者分布的，但是所有存储单元有统一的地址空间，并被所有的处理器所访问。



- 多地址空间非共享存储

所有的存储器都是私有的、仅能由其处理器所访问。



3. 从存储角度来看MIMD

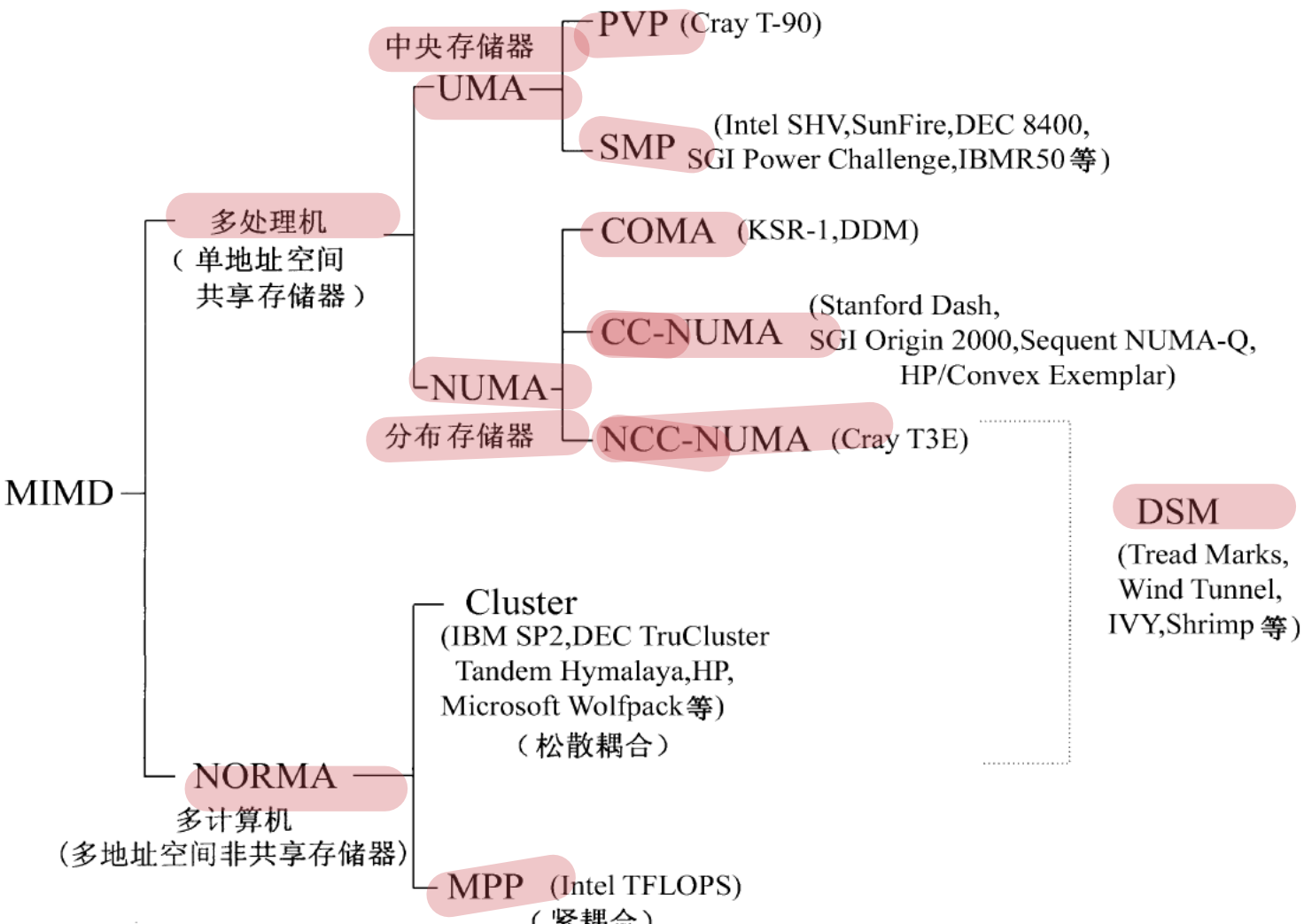
- 从存储结构的角度
  - 单地址空间共享存储
  - 多地址空间非共享存储
- 共享存储的访问
  - 均匀存储访问——UMA（Uniform Memory Access）
  - 非均匀存储访问——NUMA（Nonuniform Memory Access）

UMA

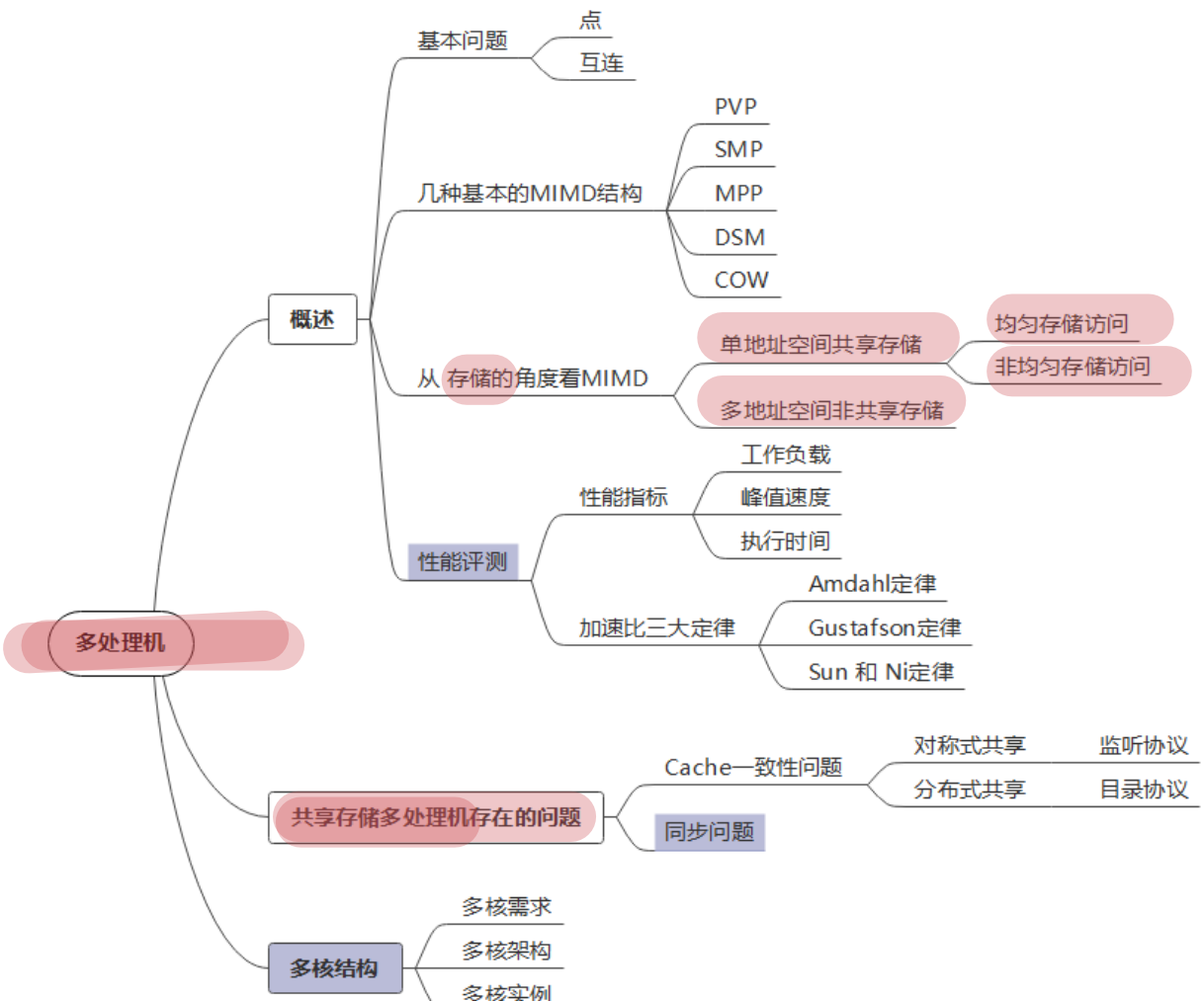
- 物理存储器被所有处理器均匀共享;
- 所有处理器访问任何存储单元取相同时间
- 每台处理器课带私有高速缓存
- 外围设备也可以一定形式共享

NUMA

- 被共享的存储器在物理上是分布在所有的处理器中的，其所有本地存储器的集合就组成了全局地址空间
- 处理器访问存储器的时间是不一样的：访问本地存储器 LM 或群内共享存储器 CSM 较快，而访问外地的存储器或全局共享存储器 GSM 较慢
- 每台处理器照例可带私有高速缓存，且外设也可以某种形式共享


MIMD
多处理机
（单地址空间
共享存储器）
中央存储器
UMA
PVP (Cray T-90)
SMP (Intel SHV,SunFire,DEC 8400,
SGI Power Challenge,IBMR50等)
NUMA
分布存储器
COMA (KSR-1,DDM)
CC-NUMA (Stanford Dash,
SGI Origin 2000,Sequent NUMA-Q,
HP/Convex Exemplar)
NCC-NUMA (Cray T3E)
DSM
(Tread Marks,
Wind Tunnel,
IVY,Shrimp等)
NORMA
多计算机
（多地址空间非共享存储器）
Cluster
(IBM SP2,DEC TruCluster
Tandem Hymalaya,HP,
Microsoft Wolfpack等)
（松散耦合）
MPP (Intel TFLOPS)
（紧耦合）

本章组织结构

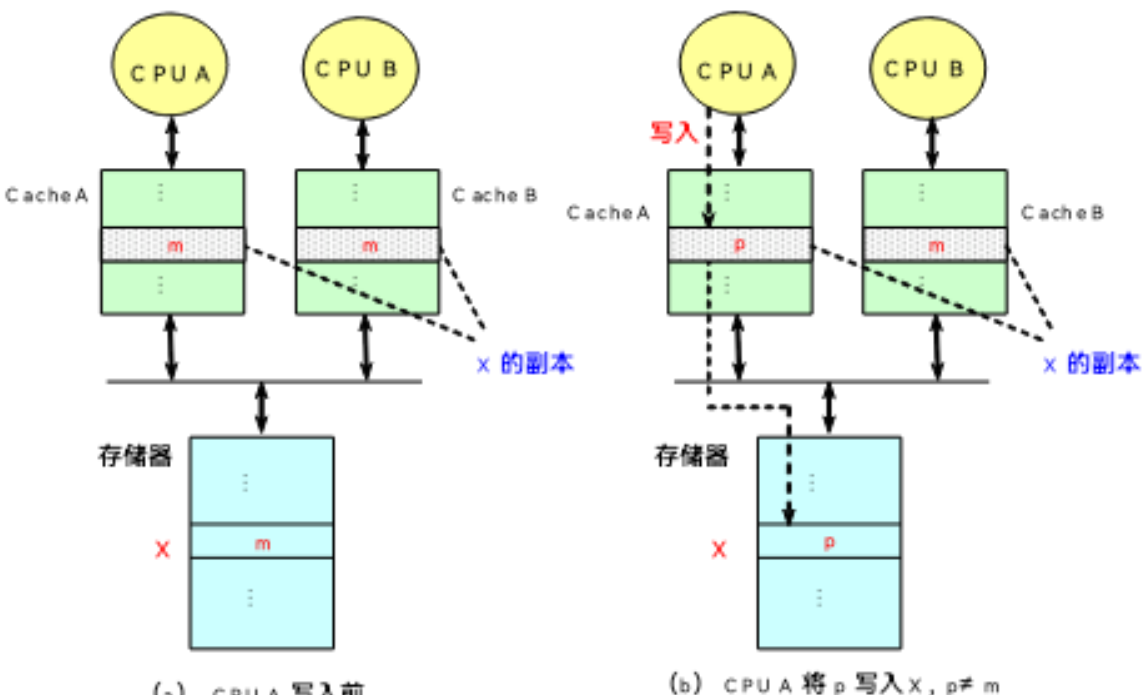


注释：着色部分为自学内容

5.2 多处理机CACHE一致性

- 什么是多处理机的Cache一致性问题

允许共享数据进入Cache，就可能出现多个处理器的Cache中都有同一存储块的副本，当其中某个处理器对其Cache中的数据进行修改后，就会使得其Cache中的数据与其他Cache中的数据不一致。



- 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
  - 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，读和写之间没有其他处理器对单元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，读和写之间无其他写，则Q读到的值应为P写进去的值。
    - 对同一单元的写是串行化的，即任意两个处理器对同一单元的两次写，从各个处理器的角度看来顺序都是相同的。(写串行化)

- 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。

- 在一致的多处理机中，Cache提供两种功能：

迁移
复制

  - 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。

一般情况下，小规模多处理机是采用硬件的方法来实现Cache的一致性。

- Cache一致性协议

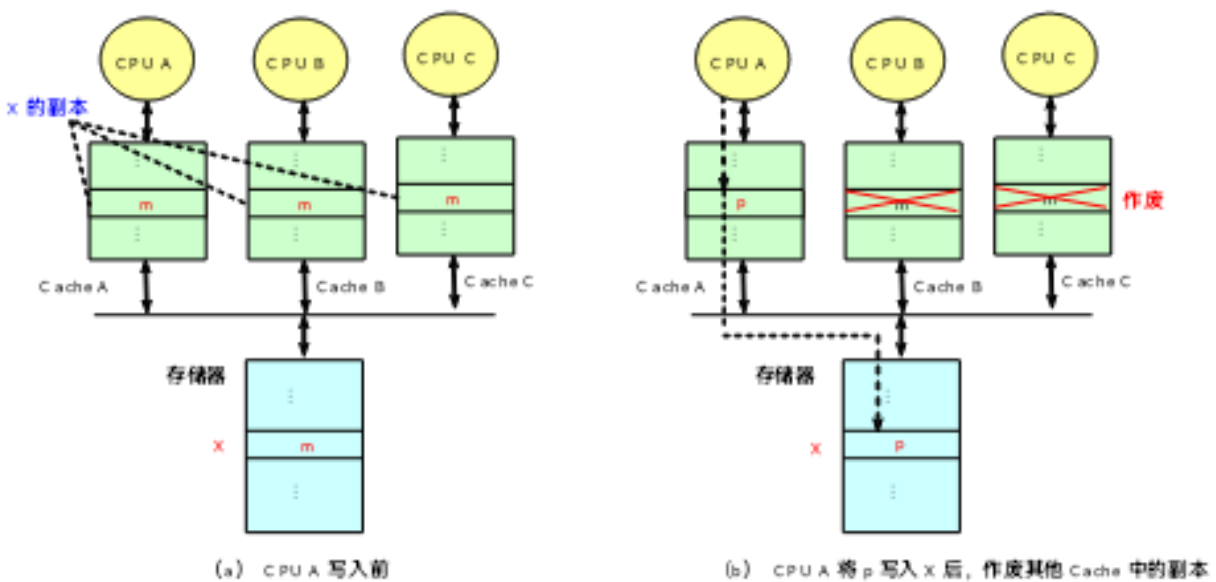
  在多个处理器中用来维护一致性的协议。

  - 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。
    - 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。
      - ✔ Cache通常连在共享存储器的总线上，当某个Cache需要访问存储器时，它会把请求放到总线上广播出去，其他各个Cache控制器通过监听总线（它们一直在监听）来判断它们是否有总线上请求的数据块。如果有，就进行相应的操作。

- 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访问权。(作废其他的副本)

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。

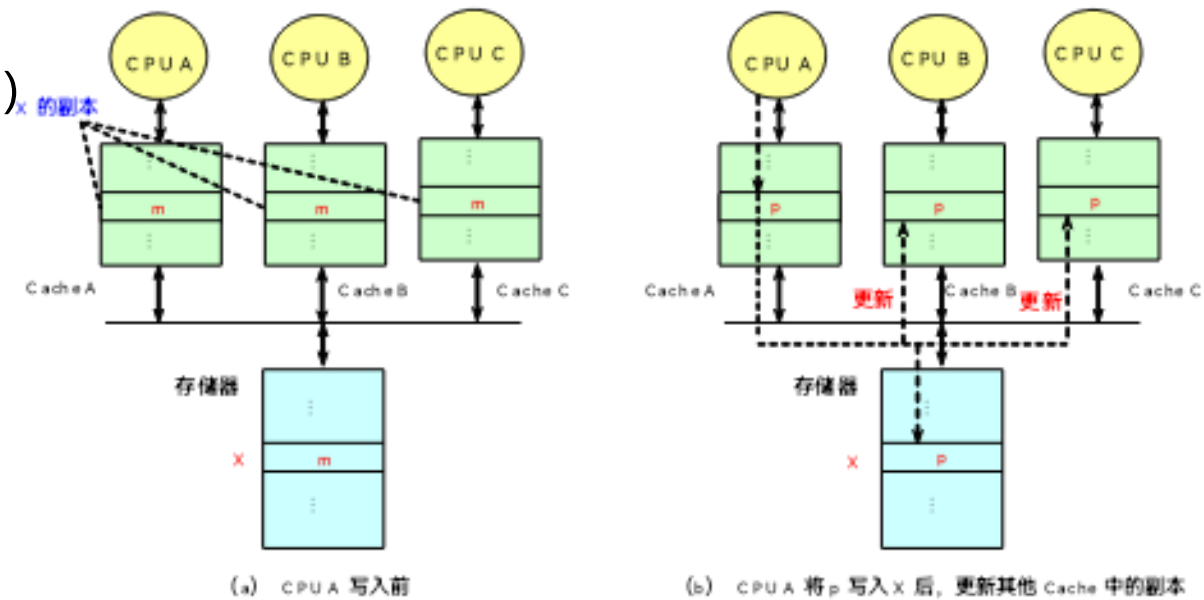


- 写更新协议：当一个处理器对某数据项进行写入时，通过广播使其他Cache中所有对应于该数据项的副本进行更新。

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。



- 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。

5.2.1 监听协议的实现

监听协议的基本实现技术

- 实现监听协议的关键有3个方面
  - 处理器之间通过一个可以实现广播的互连机制相连。

    通常采用的是总线。
  - 当一个处理器的Cache响应本地CPU的访问时，如果它涉及全局操作，其Cache控制器就要在获得总线的控制权后，在总线上发出相应的消息。
  - 所有处理器都一直在监听总线，它们检测总线上的地址在它们的Cache中是否有副本。若有，则响应该消息，并进行相应的操作。

- 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

  再通过总线找到相应数据块的最新副本，然后调入本地Cache中。
- 有的监听协议还增设了一条Invalidate消息（作废），用来通知其他各处理器作废其Cache中相应的副本。
  - 与WtMiss的区别：Invalidate不引起调块
- 写操作的串行化：由总线实现（获取总线控制权的顺序性）
  - 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）

- 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。

监听协议举例

- 在每个结点内嵌入一个有限状态控制器。
  - 该控制器根据来自处理器或总线的请求以及Cache块的状态，做出相应的响应。
- 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。

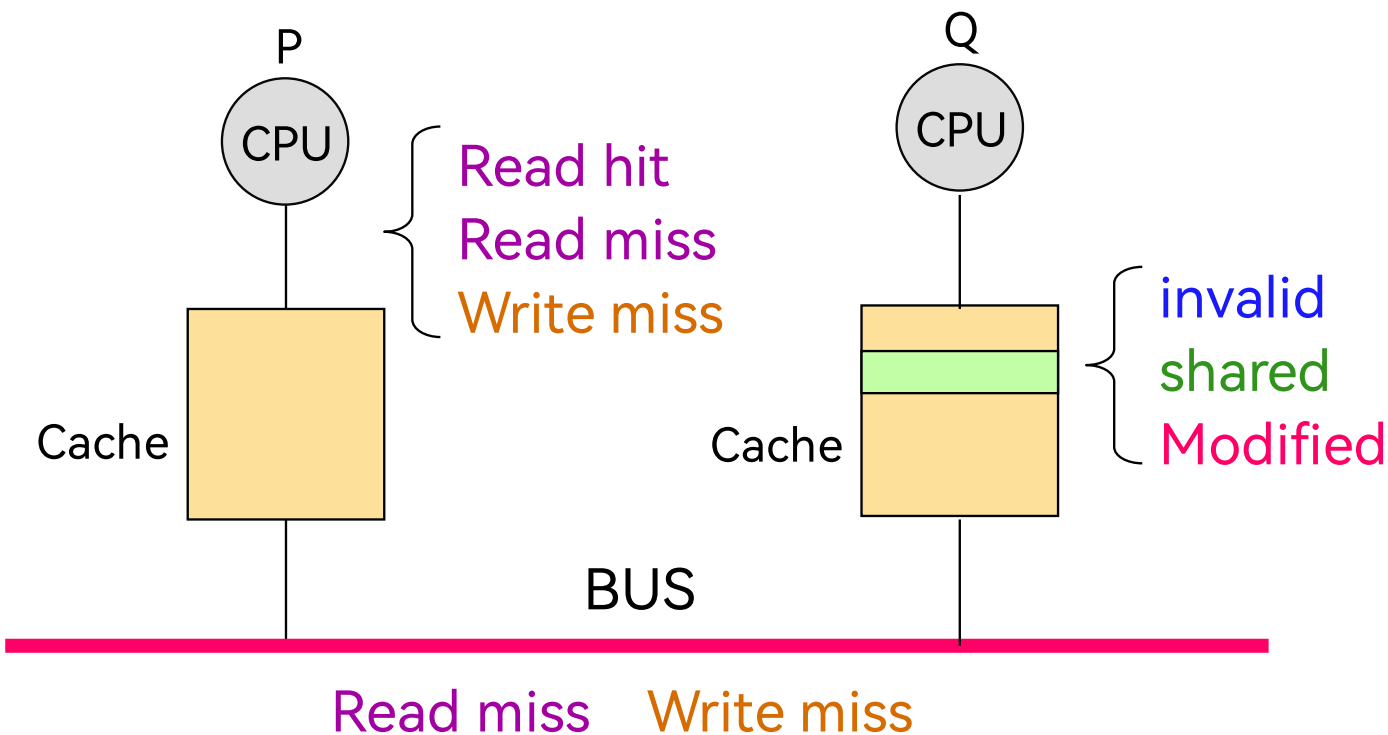
    在多个处理器中都有副本。这些副本都相同，且与存储器中相应的块相同。
  - 已修改（简称M）：该块已经被修改过，并且还没写入存储器。

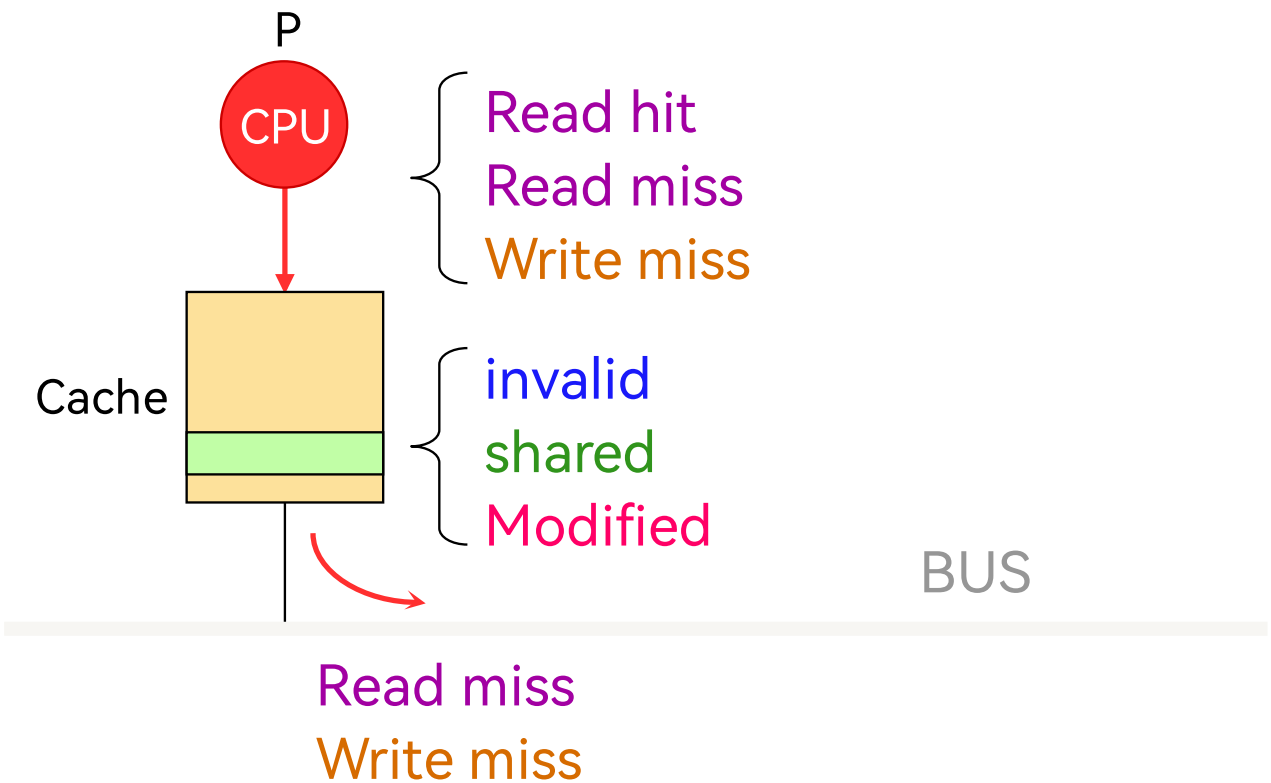
  （块中的内容是最新的，系统中唯一的最新副本）
- 因为每次总线任务均要检查Cache的地址位，这可能与CPU对Cache的访问冲突。可通过下列两种技术之一降低冲突：
  - 复制标志位
  - 采用多级包容Cache （许多系统采用）

监听协议举例

写作废，写回法

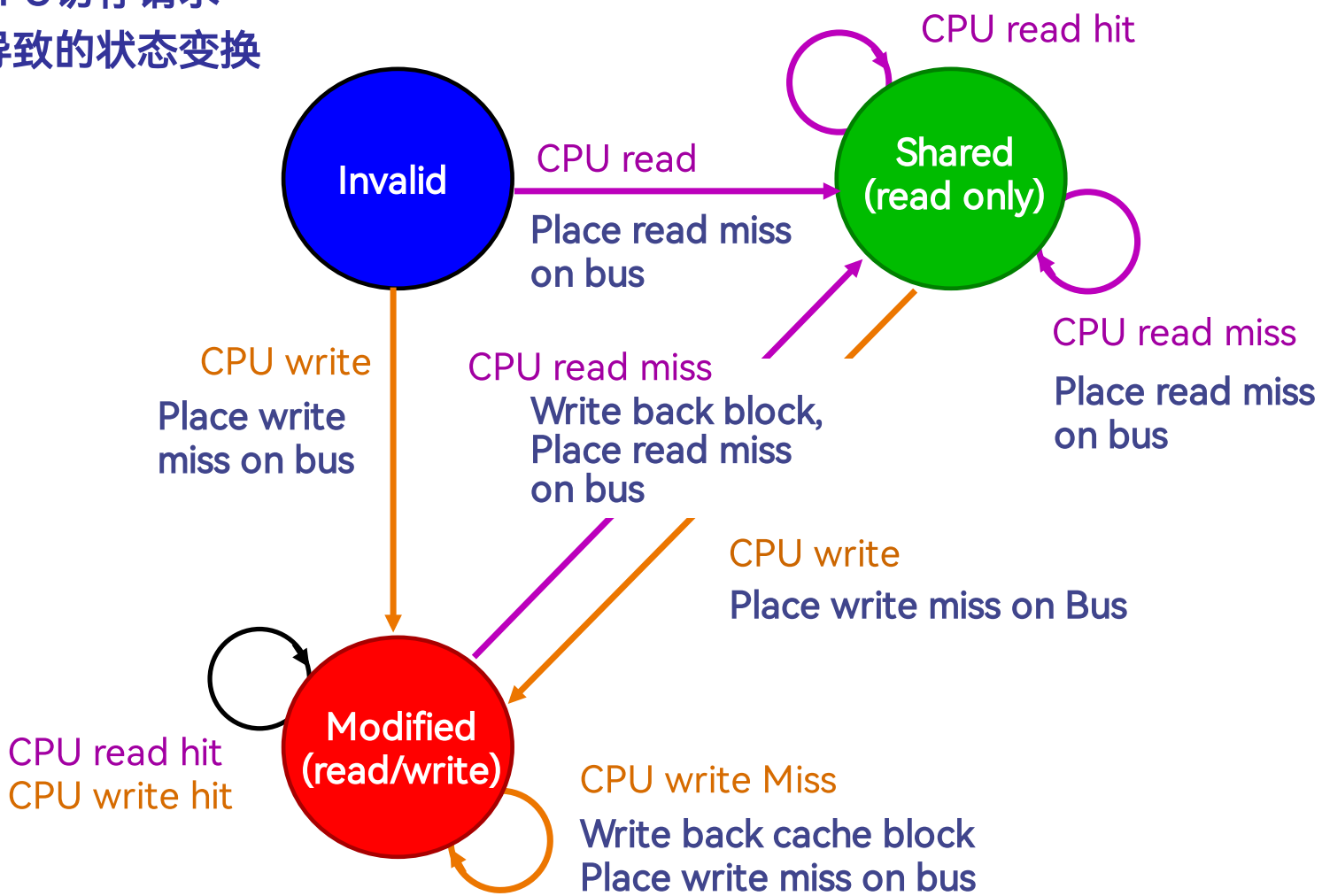
为简单起见，对 Write hit 和 Write miss 不加区分，都按 Write miss 处理


P
CPU
Read hit
Read miss
Write miss
Cache
Q
CPU
invalid
shared
Modified
Cache
BUS
Read miss
Write miss


P
CPU
Read hit
Read miss
Write miss
Cache
invalid
shared
Modified
BUS
Read miss
Write miss

SNOOPY-CACHE 状态机-I

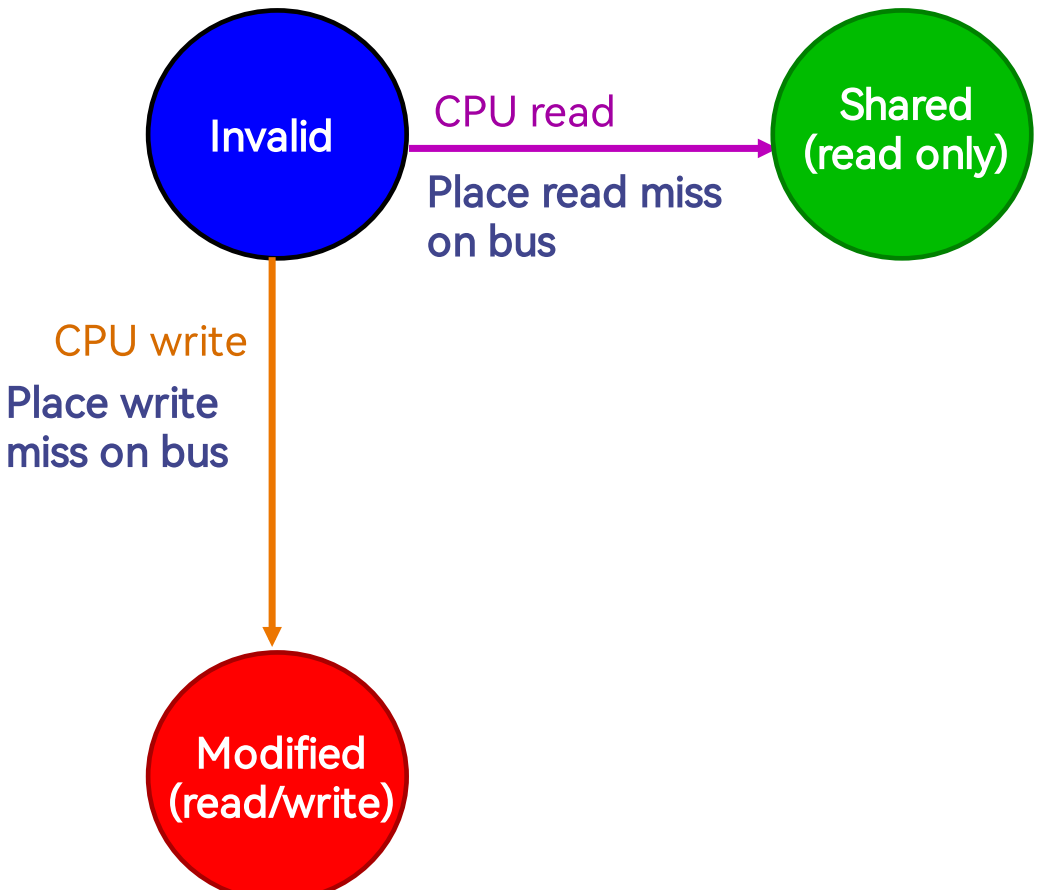
CPU访存请求
导致的状态变换



SNOOPY-CACHE 状态机-I

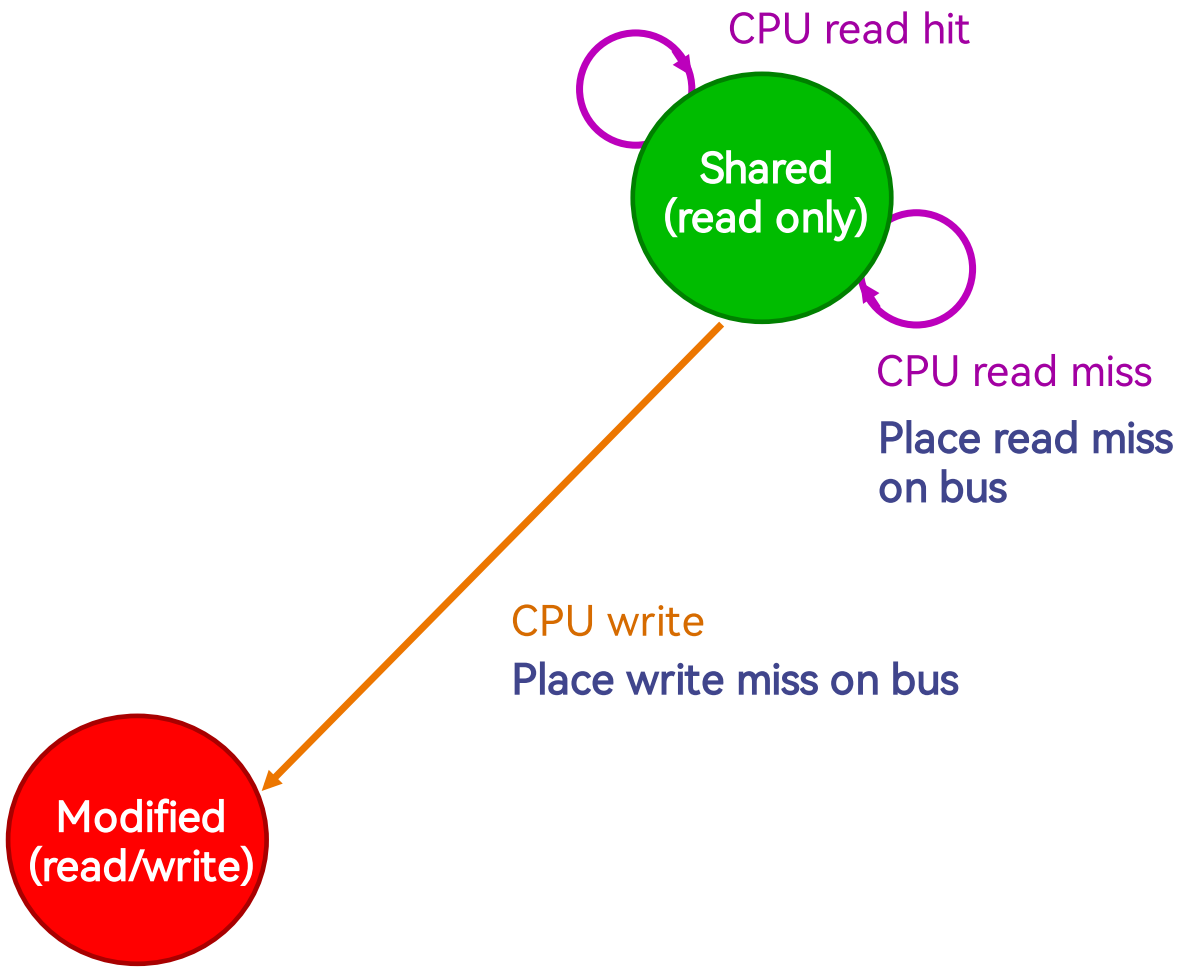
CPU访存请求

导致的状态变换



SNOOPY-CACHE 状态机-I

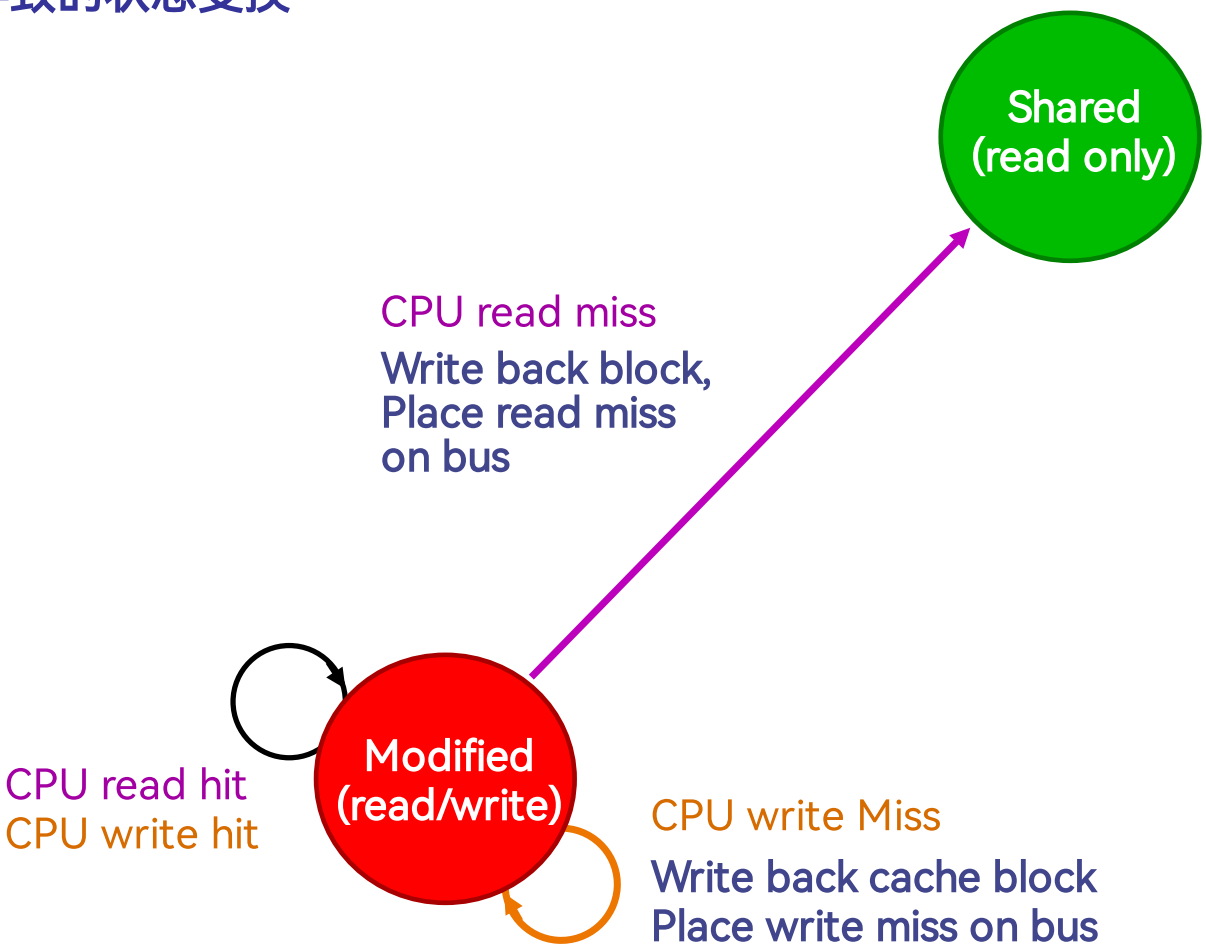
CPU访存请求
导致的状态变换

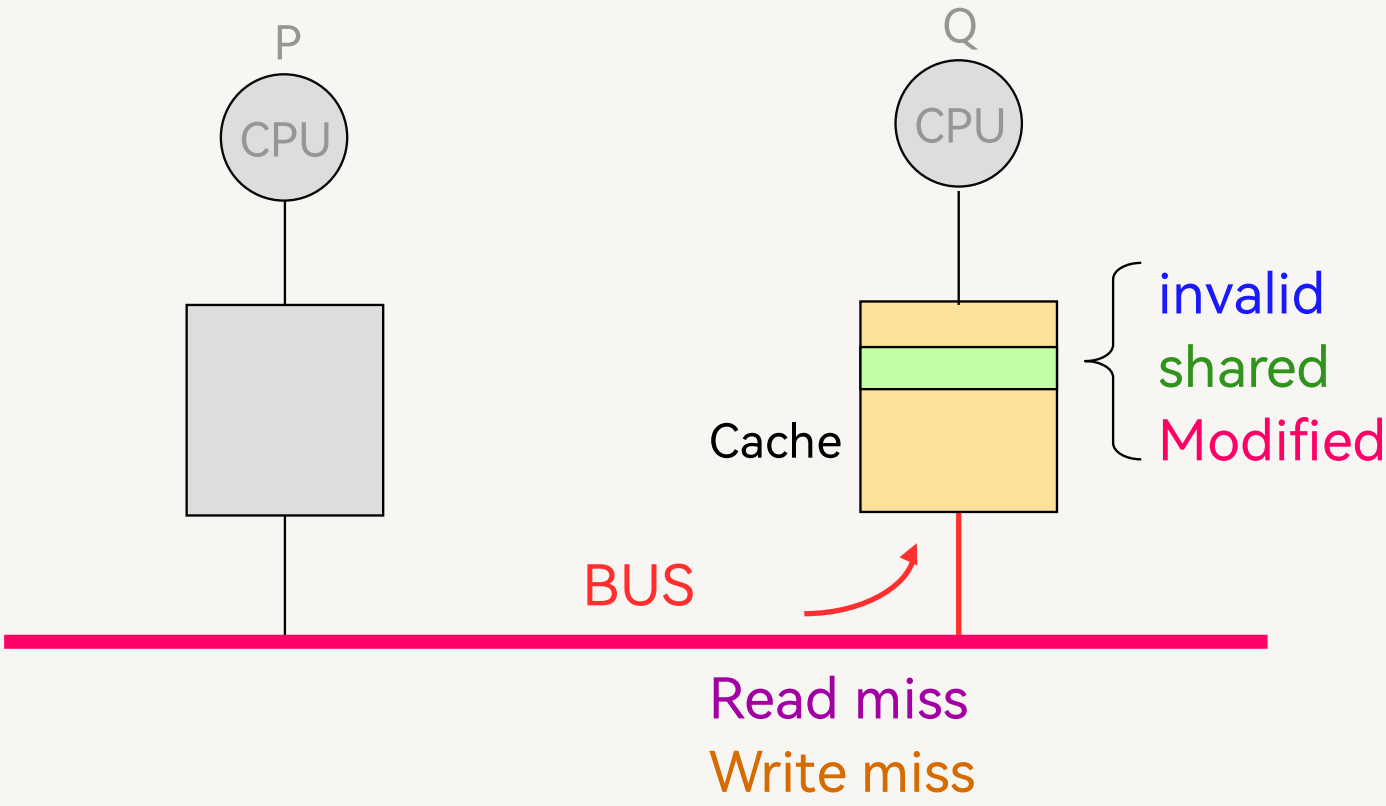


SNOOPY-CACHE 状态机-I

CPU访存请求

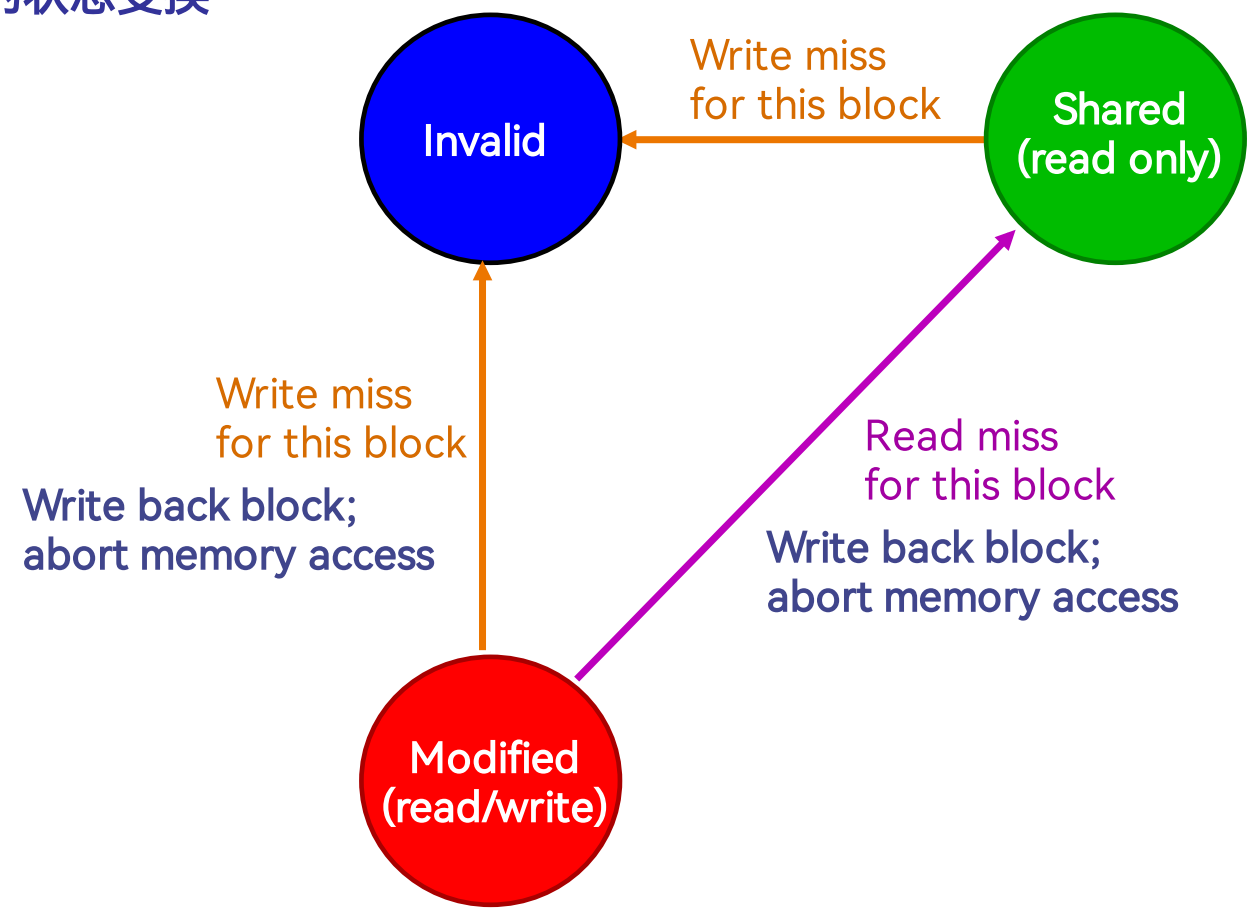
导致的状态变换

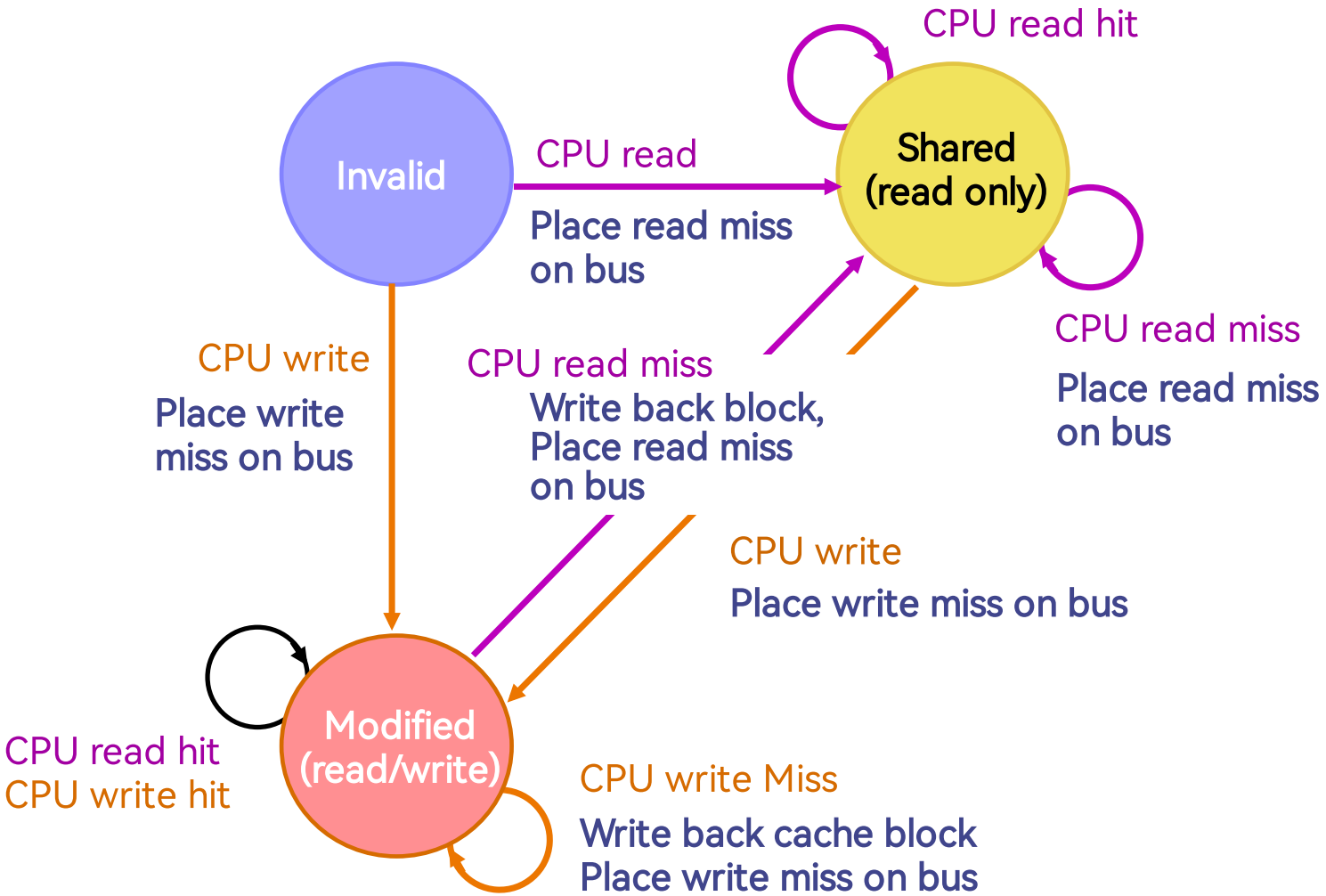



P
CPU
Q
CPU
invalid
shared
Modified
Cache
BUS
Read miss
Write miss

Snoopy-Cache 状态机-II

总线上的请求
导致的状态变换

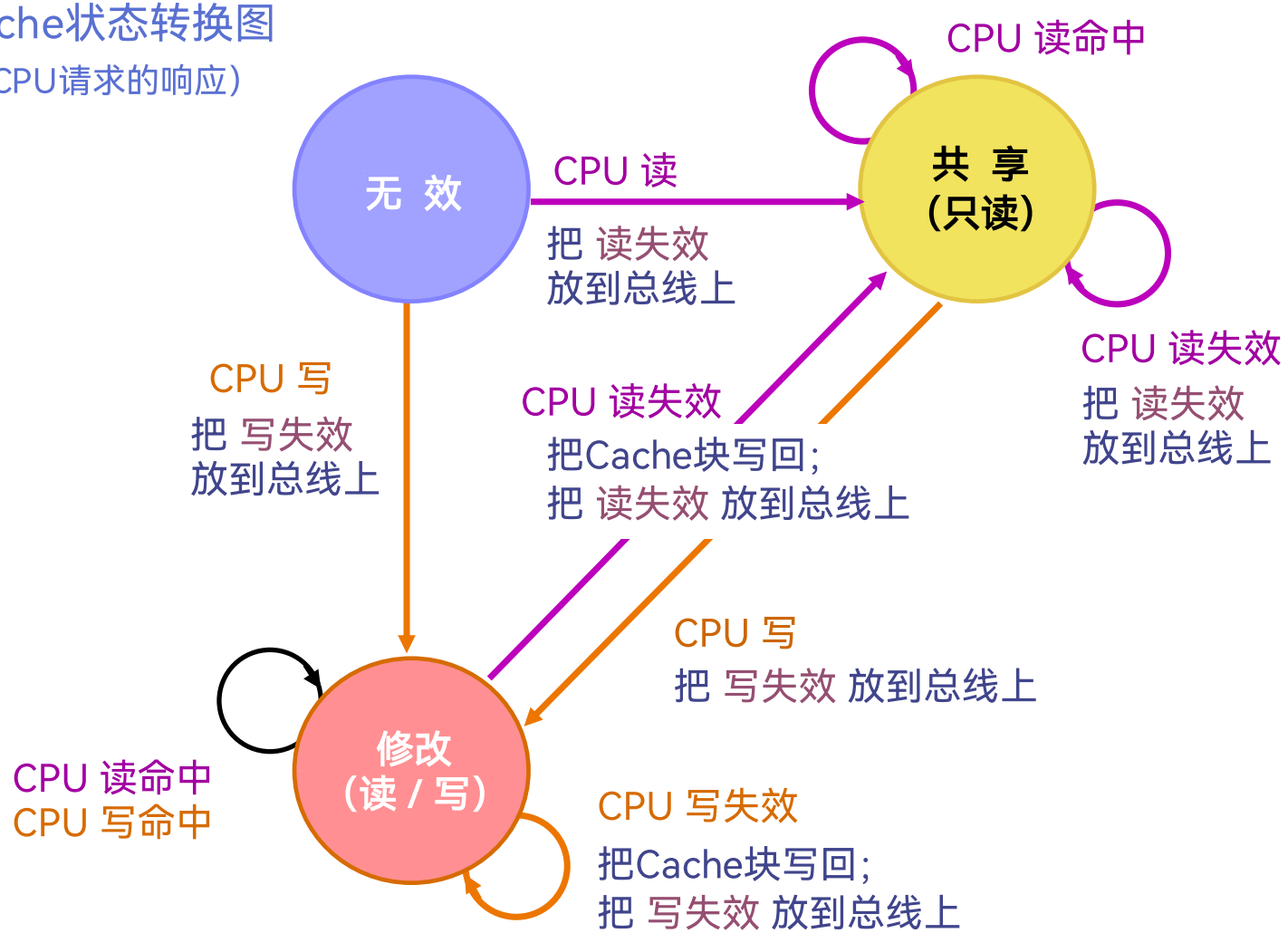



CPU read hit
Invalid
CPU read
Shared
(read only)
Place read miss
on bus
CPU read miss
Place read miss
on bus
CPU write
Place write
miss on bus
CPU read miss
Write back block,
Place read miss
on bus
CPU write
Place write miss on bus
CPU read hit
CPU write hit
Modified
(read/write)
CPU write Miss
Write back cache block
Place write miss on bus

Snooping 协议

Cache状态转换图

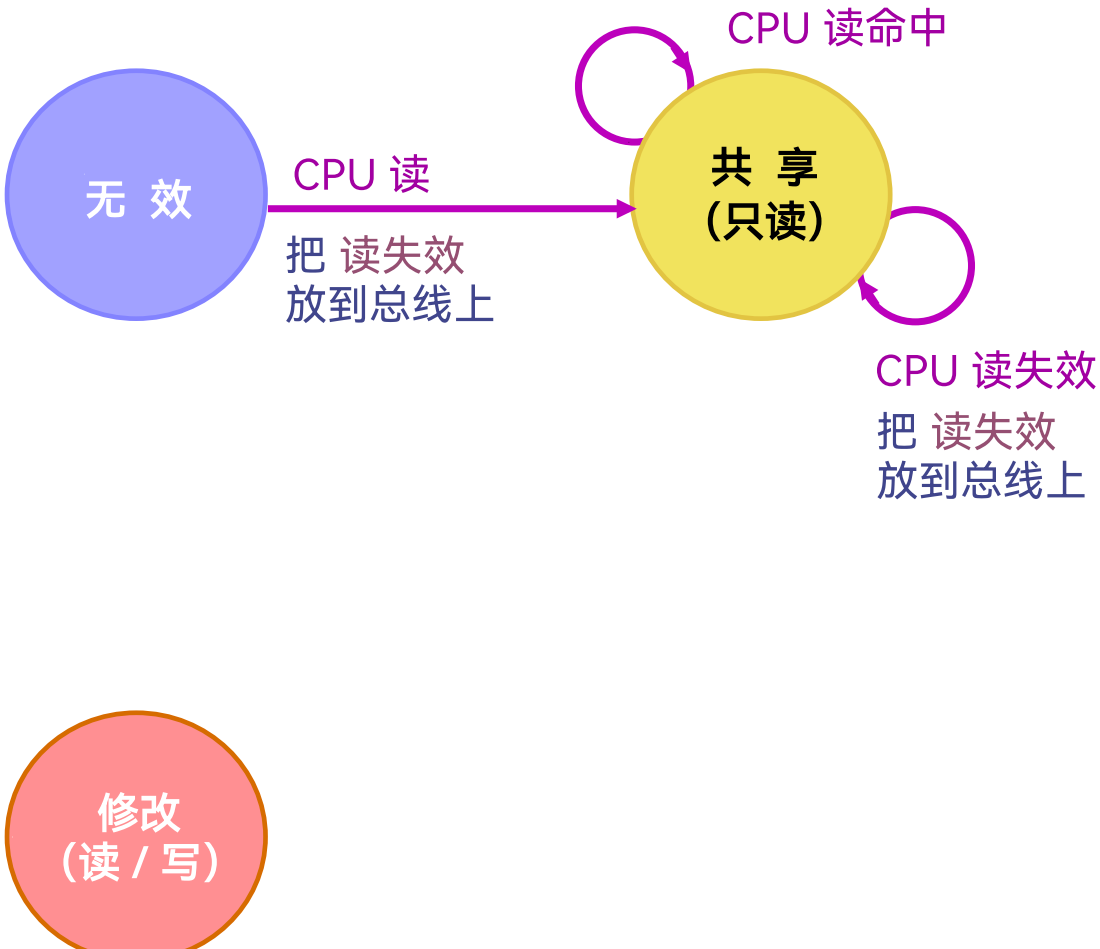
(对CPU请求的响应)

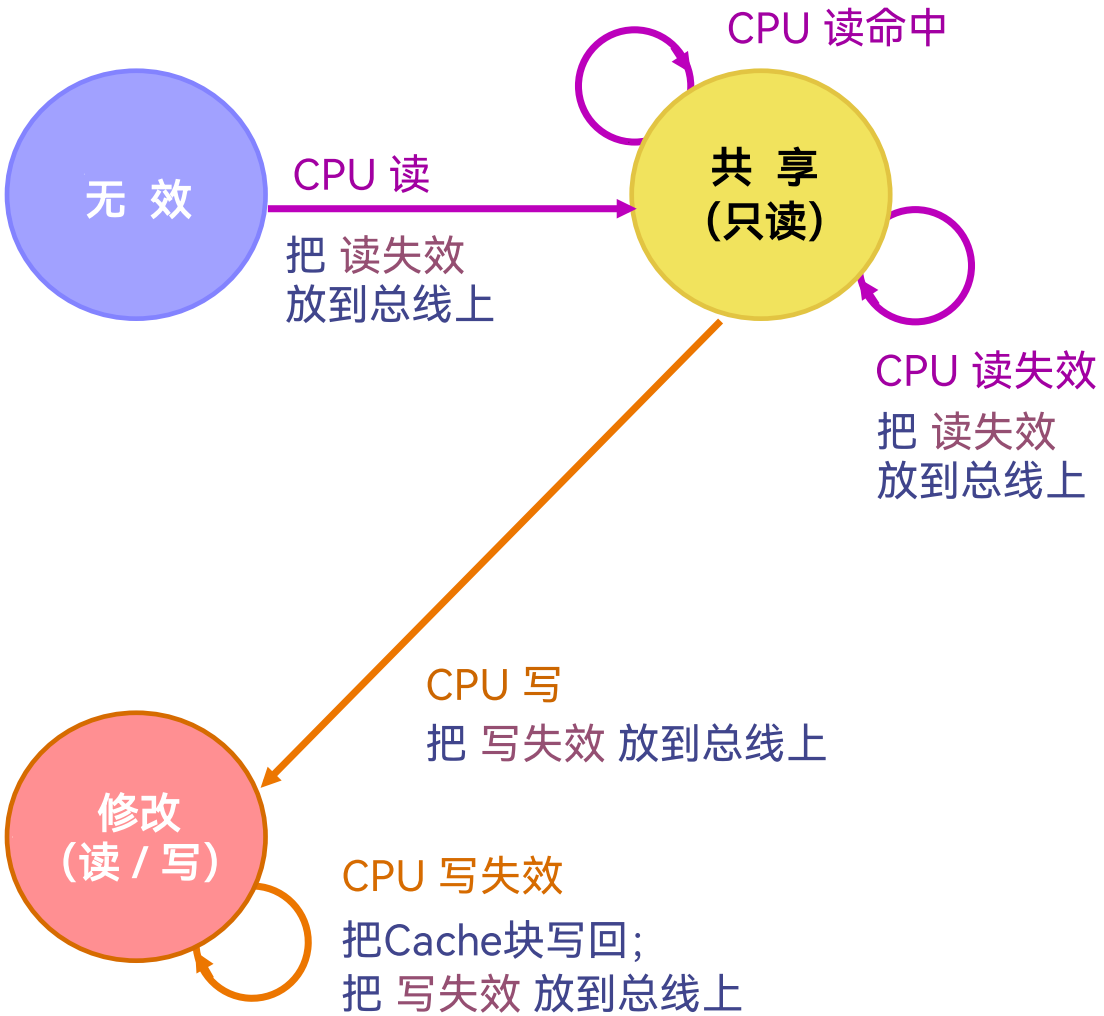


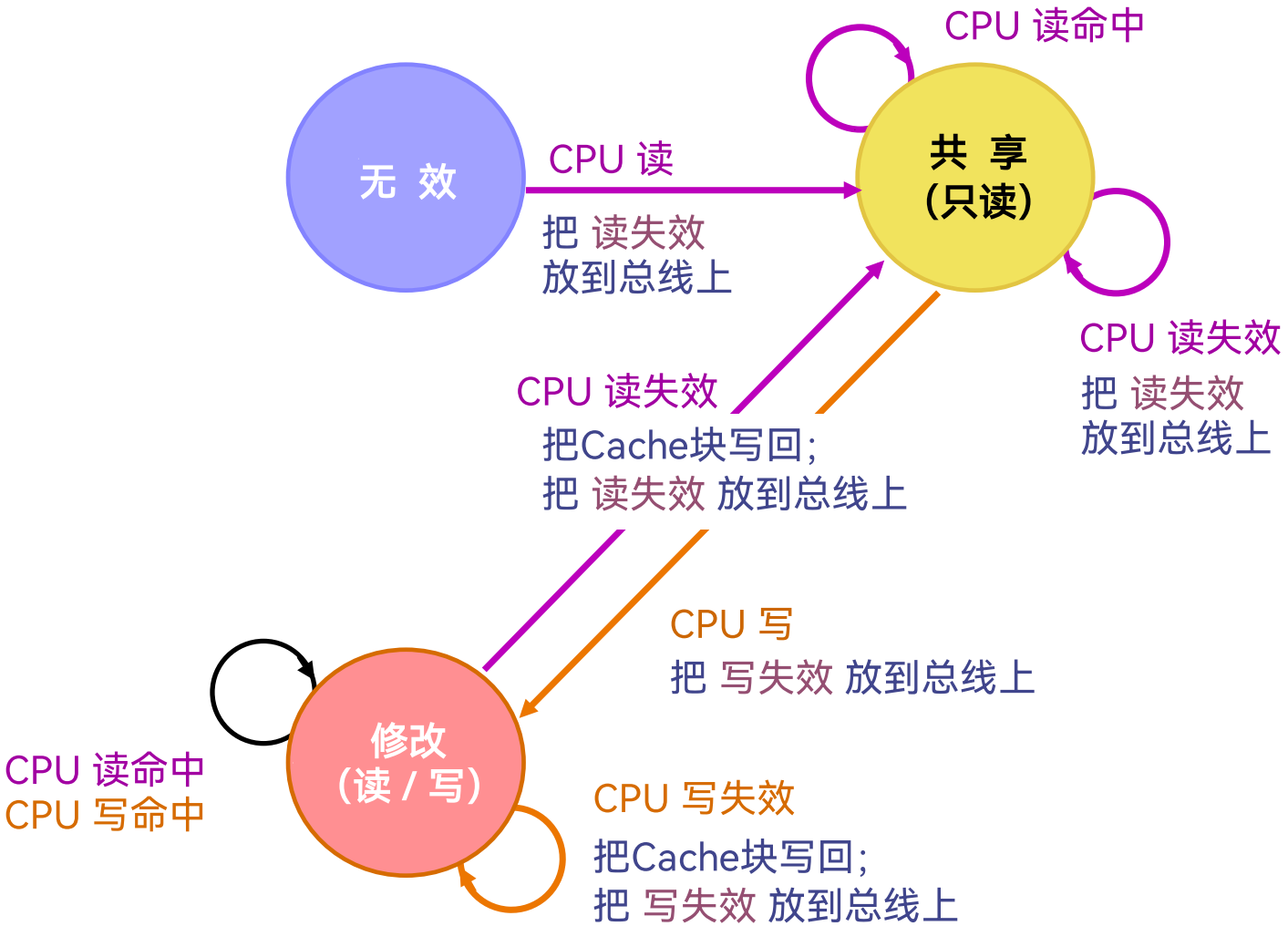

无 效

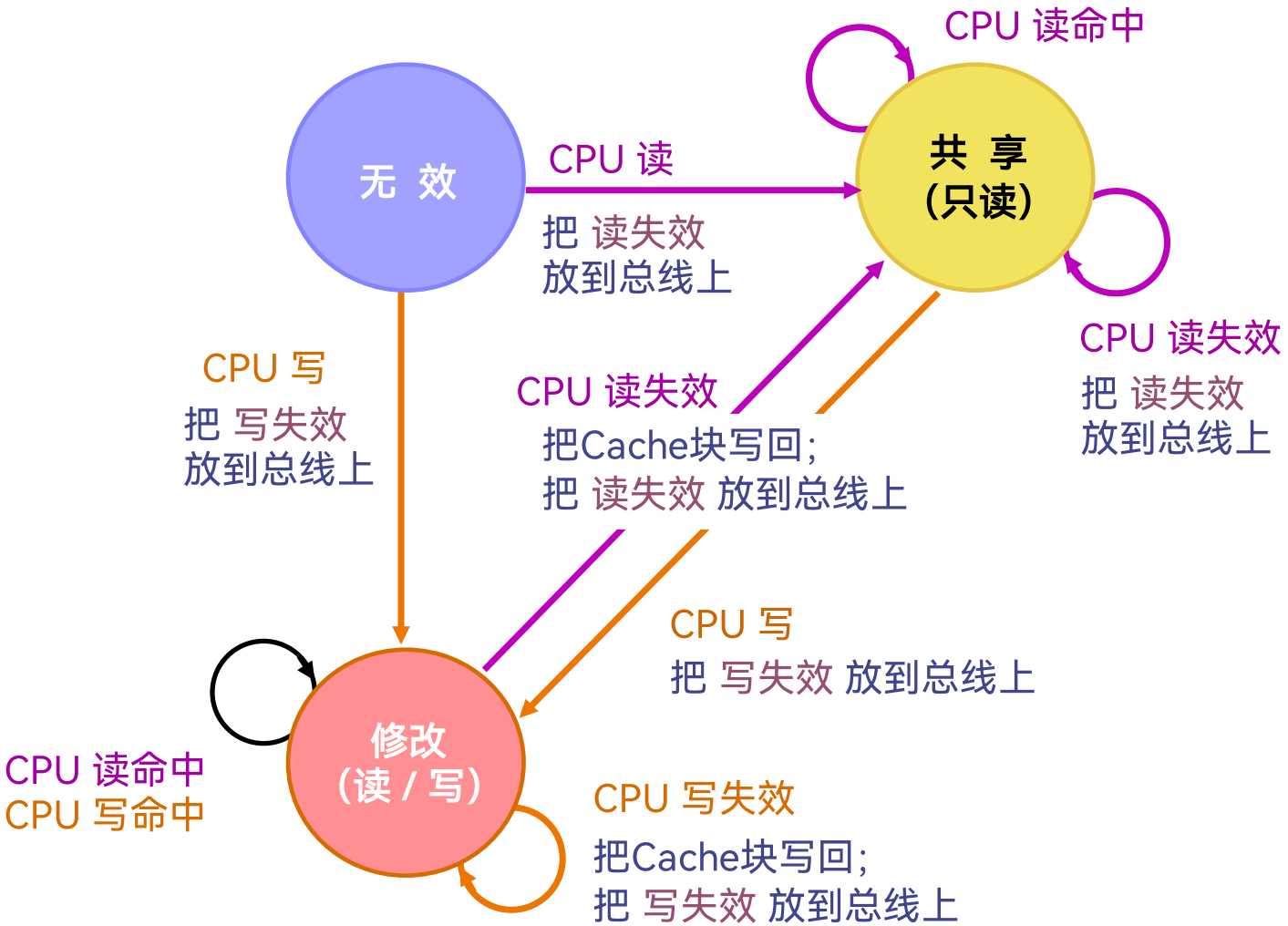

共 享
(只读)


修改
(读 / 写)


CPU 读命中
无 效
CPU 读
共 享
(只读)
把 读失效
放到总线上
CPU 读失效
把 读失效
放到总线上
修改
(读 / 写)


CPU 读命中
无 效
CPU 读
共 享
(只读)
把 读失效
放到总线上
CPU 读失效
把 读失效
放到总线上
CPU 写
把 写失效 放到总线上
修改
(读 / 写)
CPU 写失效
把Cache块写回;
把 写失效 放到总线上

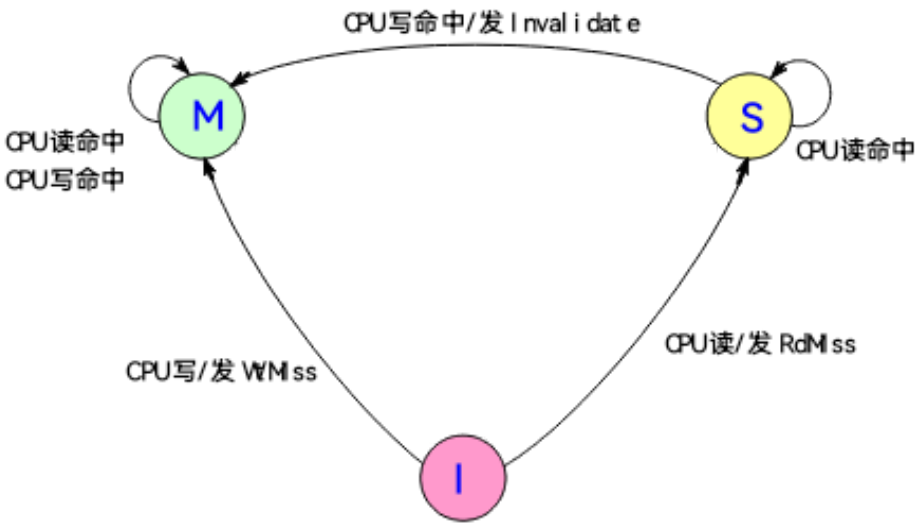

CPU 读命中
无 效
CPU 读
把 读失效
放到总线上
共 享
(只读)
CPU 读失效
把 读失效
放到总线上
CPU 读失效
把Cache块写回;
把 读失效 放到总线上
CPU 写
把 写失效 放到总线上
修改
(读 / 写)
CPU 读命中
CPU 写命中
CPU 写失效
把Cache块写回;
把 写失效 放到总线上


CPU 读命中
无 效
CPU 读
共 享
(只读)
把 读失效
放到总线上
CPU 读失效
把 读失效
放到总线上
CPU 写
把 写失效
放到总线上
CPU 读失效
把Cache块写回;
把 读失效 放到总线上
CPU 写
把 写失效 放到总线上
修改
(读 / 写)
CPU 读命中
CPU 写命中
CPU 写失效
把Cache块写回;
把 写失效 放到总线上

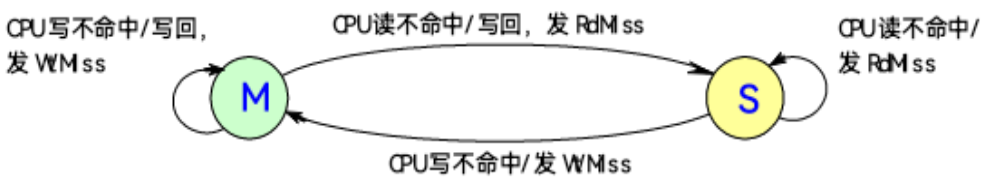
下面来讨论在各种情况下监听协议所进行的操作。

➤响应来自处理器的请求

- 不发生替换的情况



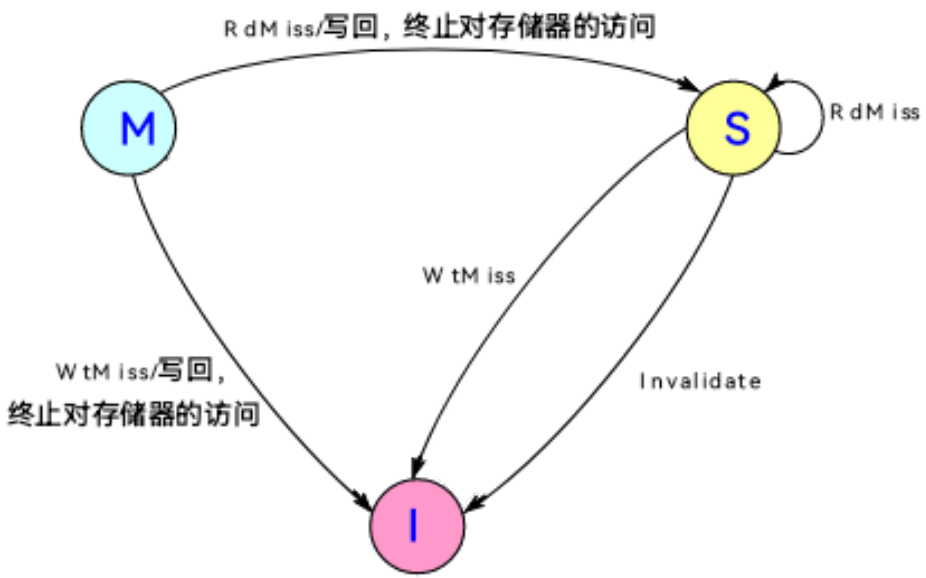
- 发生替换的情况



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

Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	Processor	Shared or modified	Normal hit	Read data in local cache.
Read miss	Processor	Invalid	Normal miss	Place read miss on bus.
Read miss	Processor	Shared	Replacement	Address conflict miss: place read miss on bus.
Read miss	Processor	Modified	Replacement	Address conflict miss: write-back block; then place read miss on bus.
Write hit	Processor	Modified	Normal hit	Write data in local cache.
Write hit	Processor	Shared	Coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, because they do not fetch the data but only change the state.
Write miss	Processor	Invalid	Normal miss	Place write miss on bus.
Write miss	Processor	Shared	Replacement	Address conflict miss: place write miss on bus.
Write miss	Processor	Modified	Replacement	Address conflict miss: write-back block; then place write miss on bus.
Read miss	Bus	Shared	No action	Allow shared cache or memory to service read miss.
Read miss	Bus	Modified	Coherence	Attempt to read shared data: place cache block on bus, write-back block, and change state to shared.
Invalidate	Bus	Shared	Coherence	Attempt to write shared block; invalidate the block.
Write miss	Bus	Shared	Coherence	Attempt to write shared block; invalidate the cache block.
Write miss	Bus	Modified	Coherence	Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache.

5.2.2 目录协议

1. 目录协议基本思想

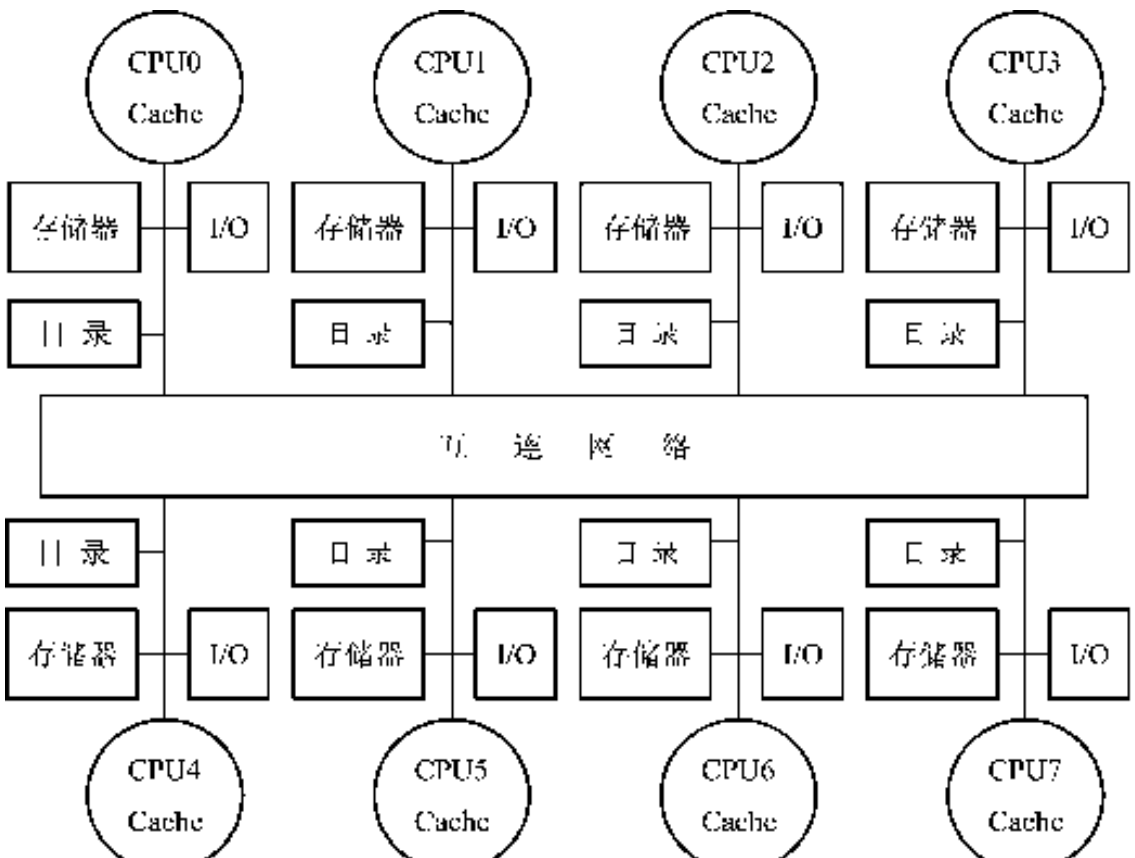
- 广播和监听的机制使得监听一致性协议的可扩放性很差。
- 寻找替代监听协议的一致性协议。

（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



- 目录法最简单的实现方案：对于存储器中每一块都在目录中设置一项。目录中的信息量与M×N成正比。

  其中：

  - M：存储器中存储块的总数量
  - N：处理器的个数
  - 由于M=K×N，K是每个处理机中存储块的数量，所以如果K保持不变，则目录中的信息量就与N^2成正比。

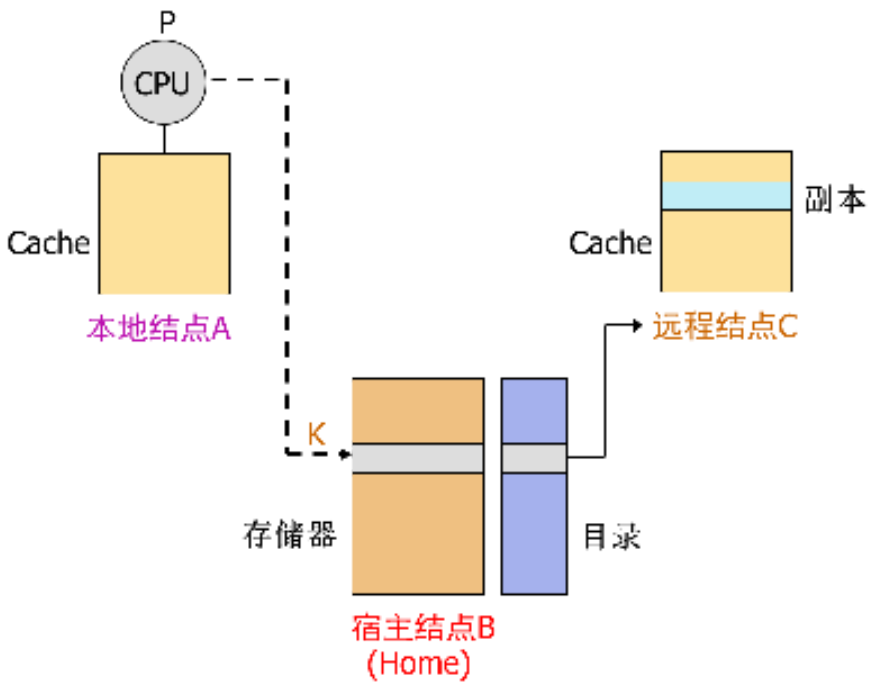
② 在目录协议中，存储块的状态有3种：

- 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
- 远程结点可以和宿主结点是同一个结点，也可以不是同一个结点。



宿主结点：存放有对应地址的存储器块和目录项的结点

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K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。

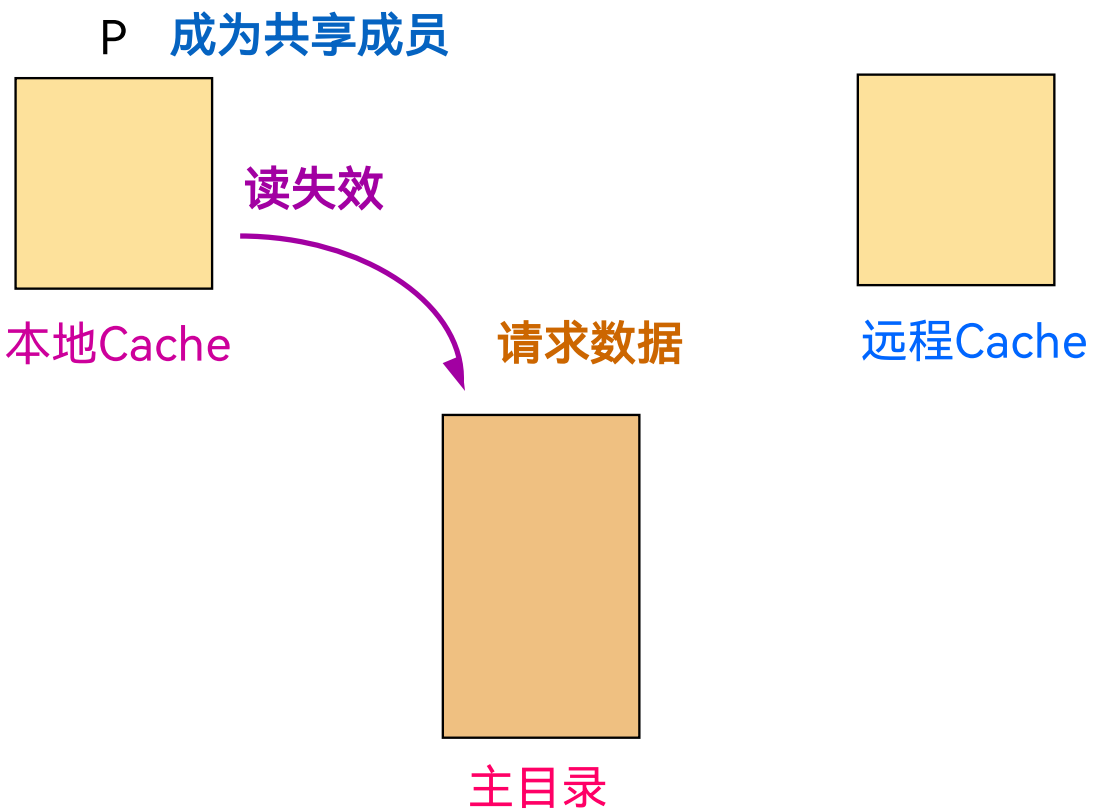
□WtMiss（P，K）

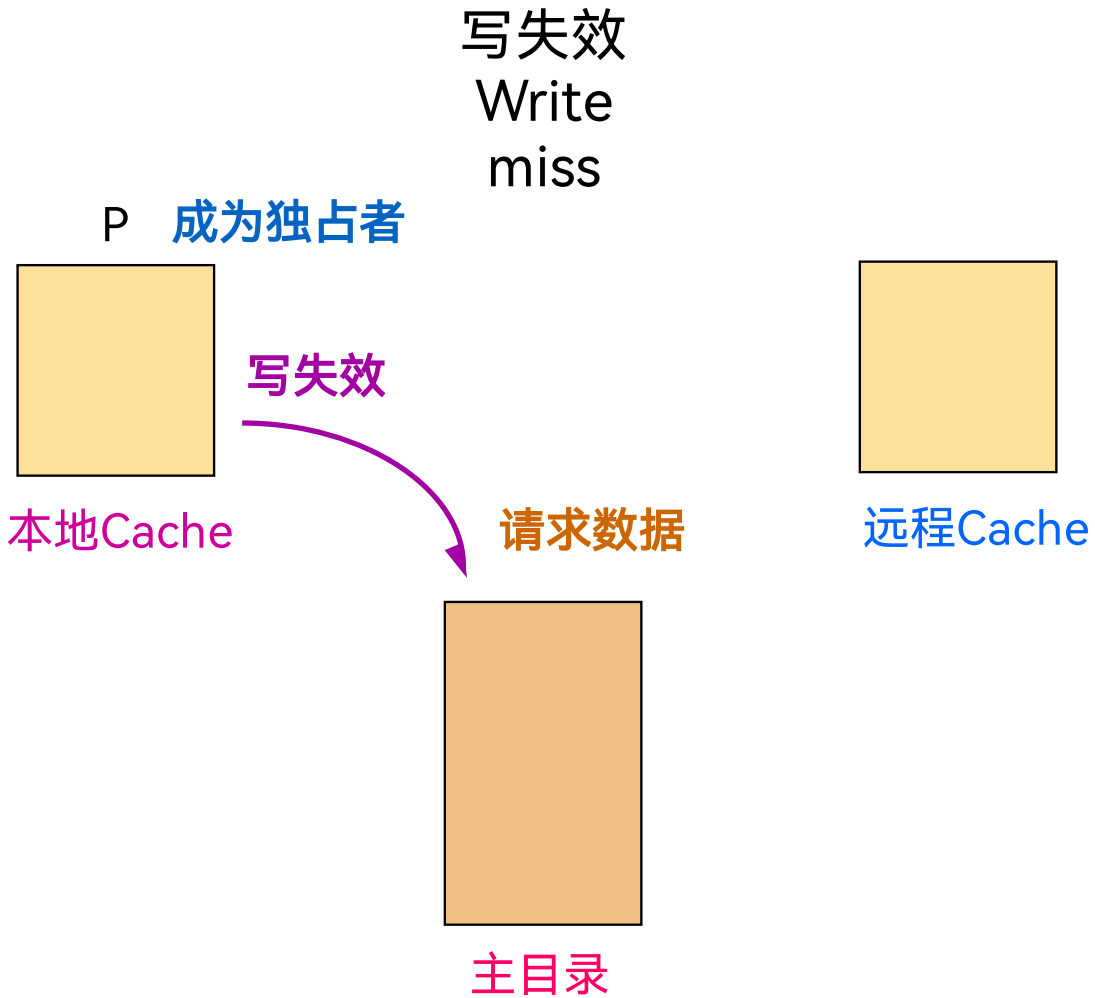
处理机P对地址K进行写入时不命中，请求宿主结点提供数据，并使P成为所访问数据块的独占者。

□Invalidate（K）

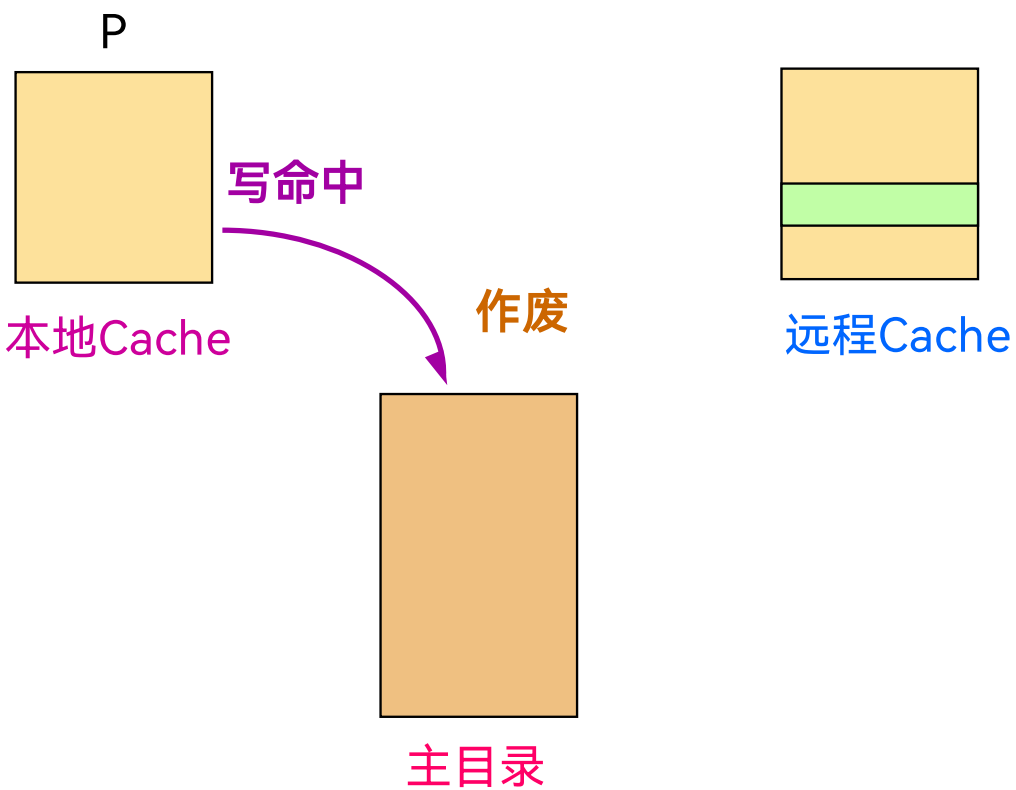
请求向所有拥有相应数据块副本（包含地址*K*）的远程Cache发Invalidate消息，作废这些副本。

读失效
Read miss




写失效
Write
miss
P
成为独占者
写失效
本地Cache
请求数据
远程Cache
主目录

作　废
invalidate



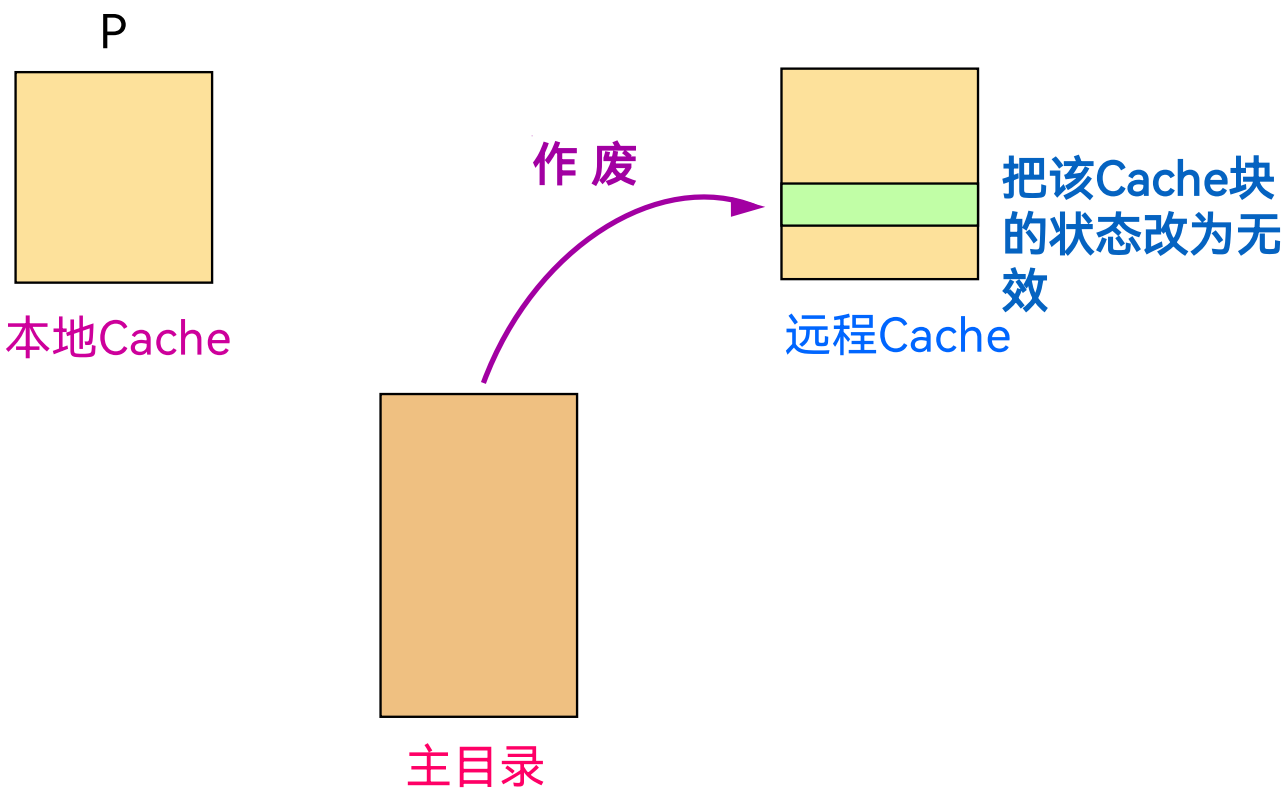
- 宿主结点（目录）发送给远程结点的消息
  - Invalidate（K）

    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）

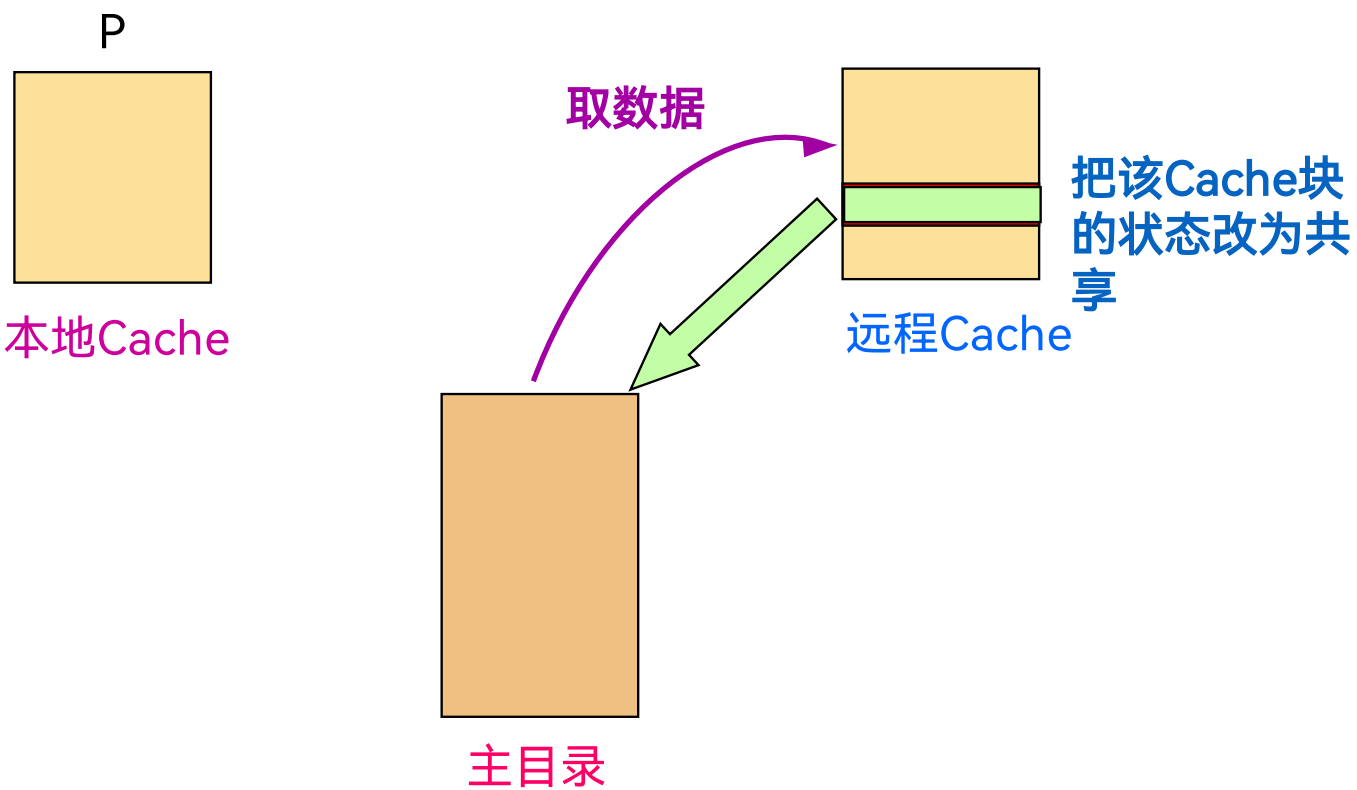
    从远程Cache中取出包含地址*K*的数据块，并将之送到宿主结点。然后作废远程Cache中的那个块。

作　废
invalidate

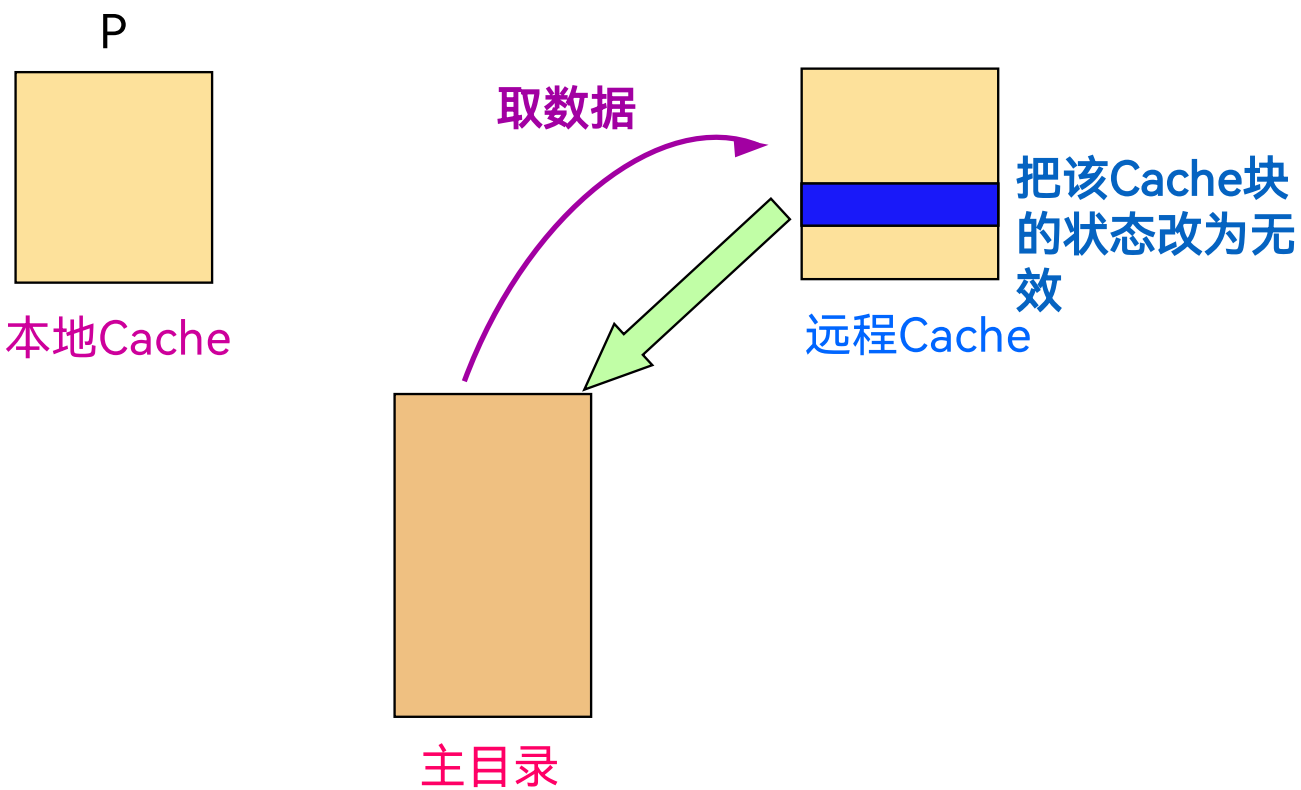


取数据

fetch



取数据 / 作废
fetch/invalidate



- 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。

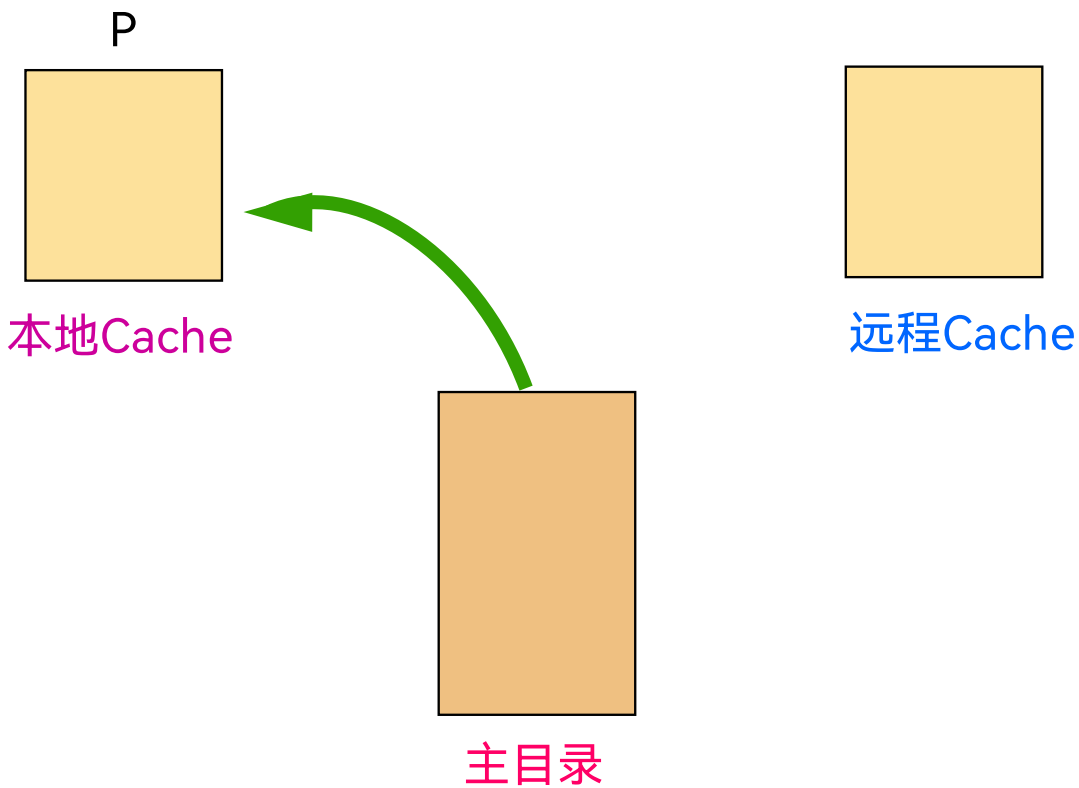
- 远程结点发送给宿主结点的消息

  WtBack（K，D）

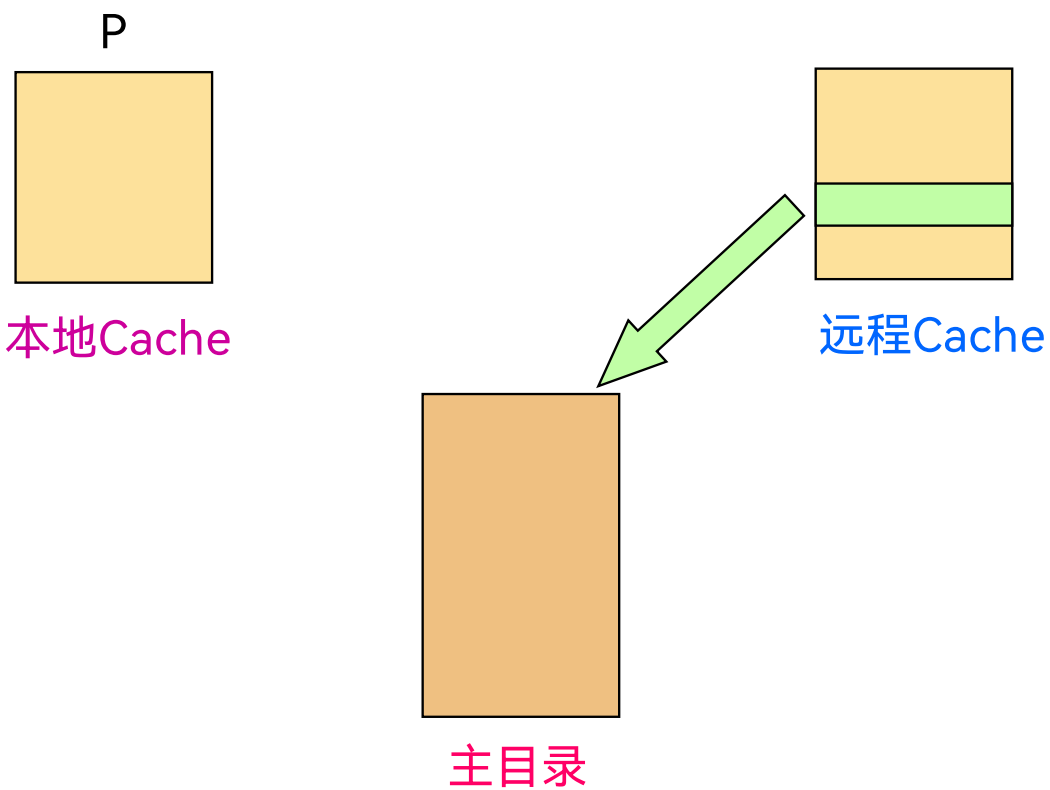
  - 把远程Cache中包含地址*K*的数据块写回到宿主结点中， 该消息是远程结点对宿主结点发来的“取数据”或“取/作废”消息的响应。

返回数据

data value reply



数据写回
data write back



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

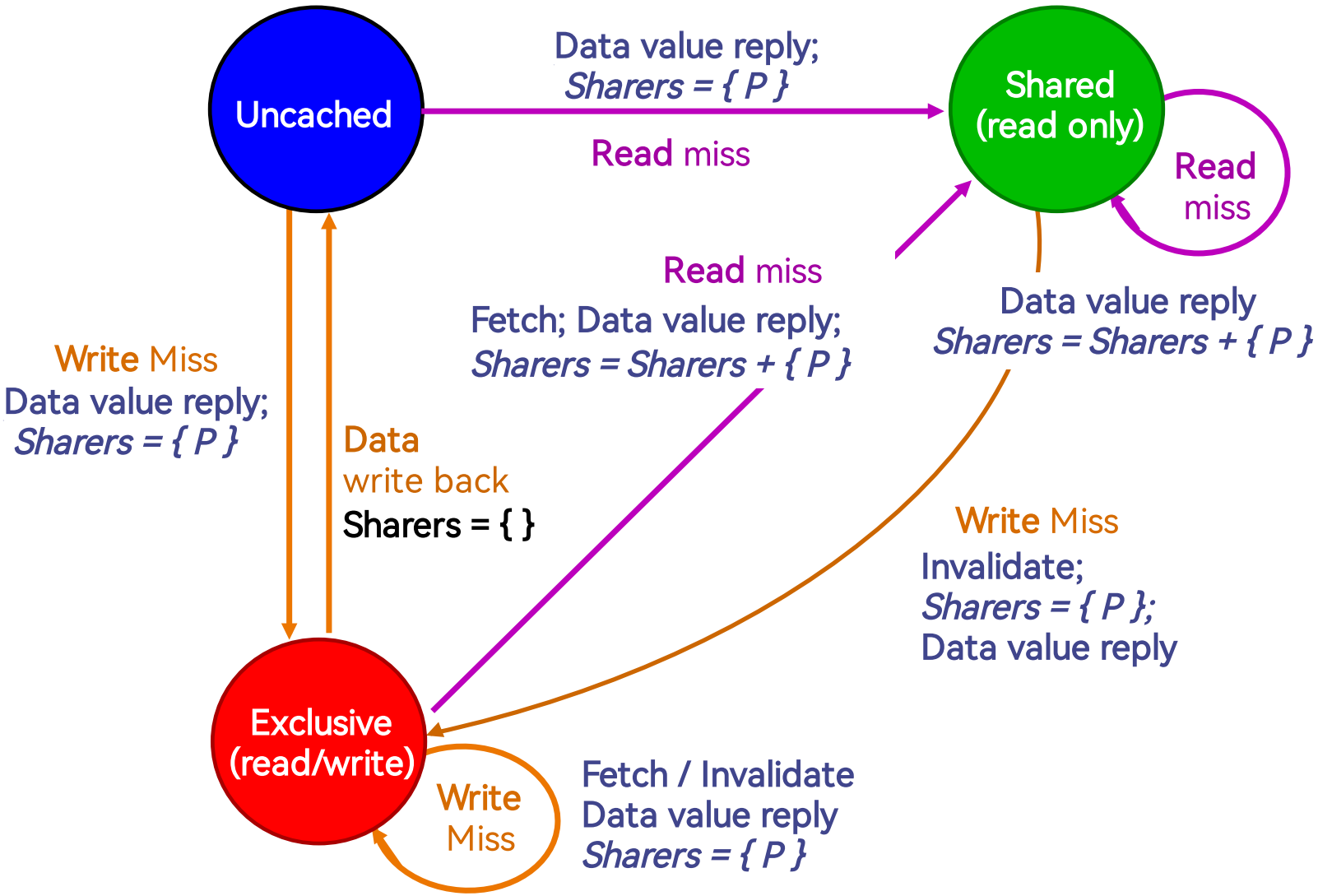
    用于当本地Cache中需要替换一个包含地址K的块、且该块未被修改过的情况。这个消息发给该块的宿主结点，请求它将P从共享集中删除。如果删除后共享集变为空集，则宿主结点还要将该块的状态改变为“未缓存”（U）。
  - WtBack2（P，K，D）

    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。

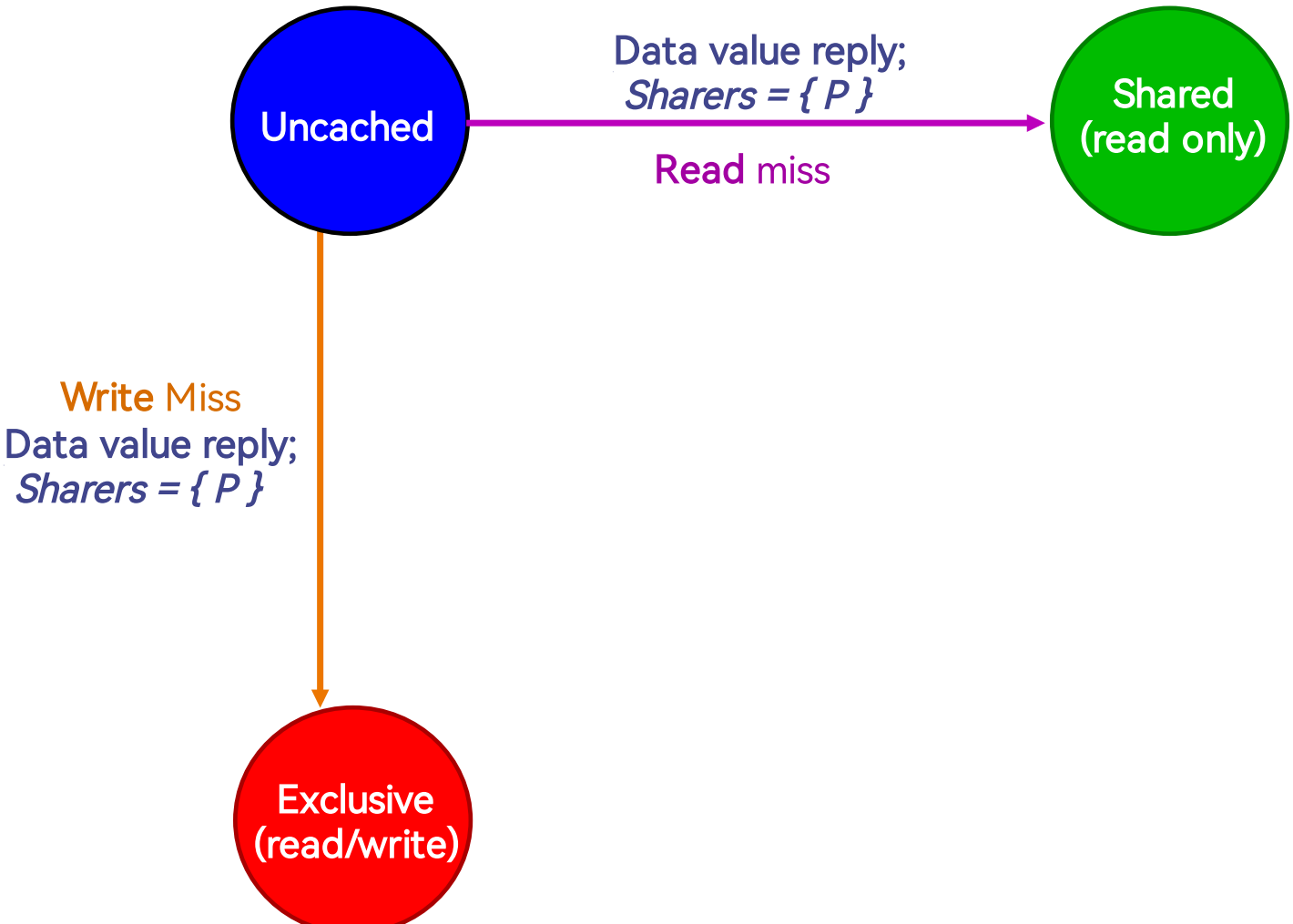
2. 目录协议实例

- 在基于目录的协议中，目录承担了一致性协议操作的主要功能。
  - 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

目录的状态转换图

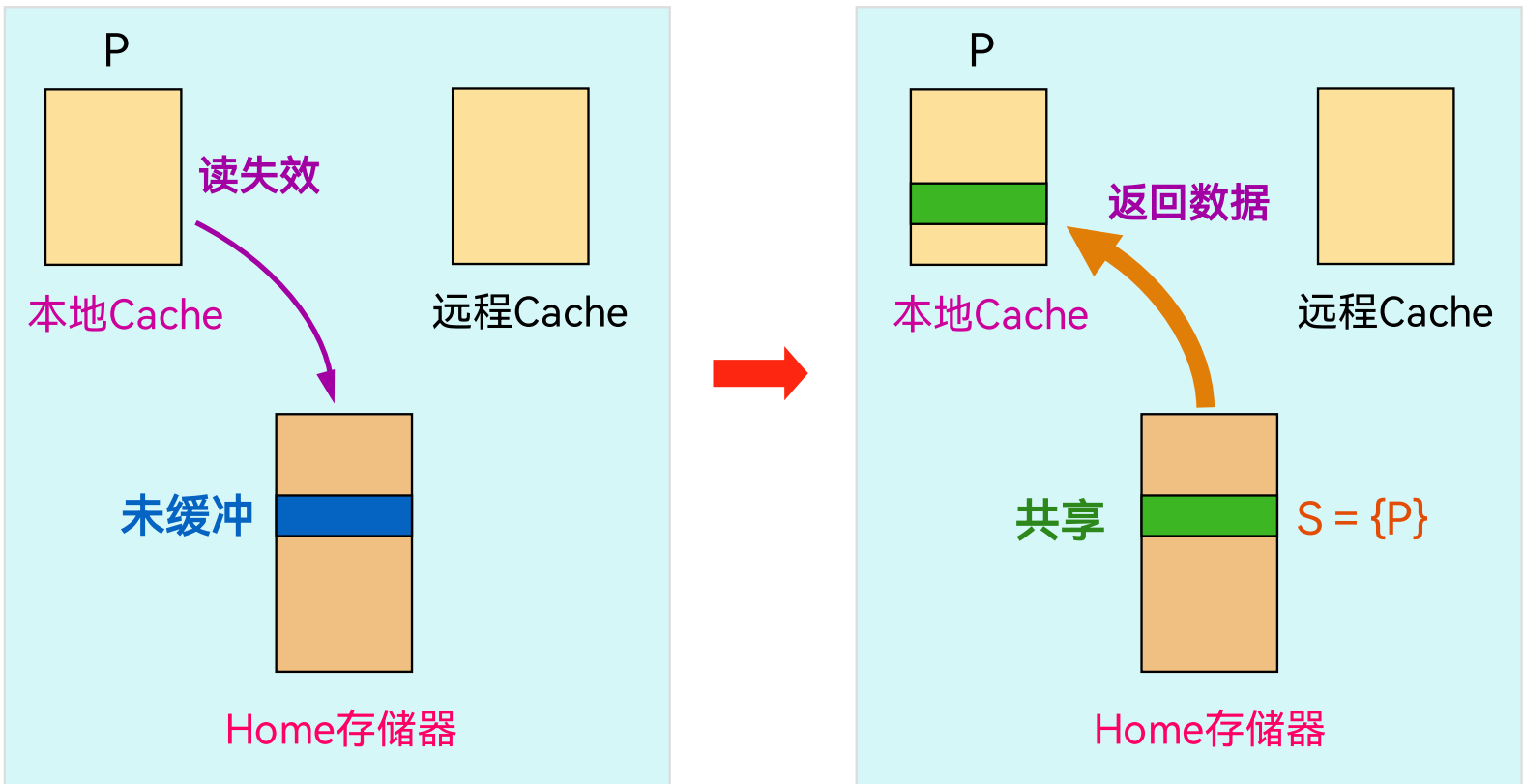

Uncached
Data value reply;
Sharers = { P }
Read miss
Shared
(read only)
Read
miss
Read miss
Fetch; Data value reply;
Sharers = Sharers + { P }
Data value reply
Sharers = Sharers + { P }
Write Miss
Data value reply;
Sharers = { P }
Data
write back
Sharers = { }
Write Miss
Invalidate;
Sharers = { P };
Data value reply
Exclusive
(read/write)
Write
Miss
Fetch / Invalidate
Data value reply
Sharers = { P }

目录的状态转换图


Uncached
Data value reply;
Sharers = { P }
Read miss
Shared
(read only)
Write Miss
Data value reply;
Sharers = { P }
Exclusive
(read/write)

- 数据块的状态为未缓冲

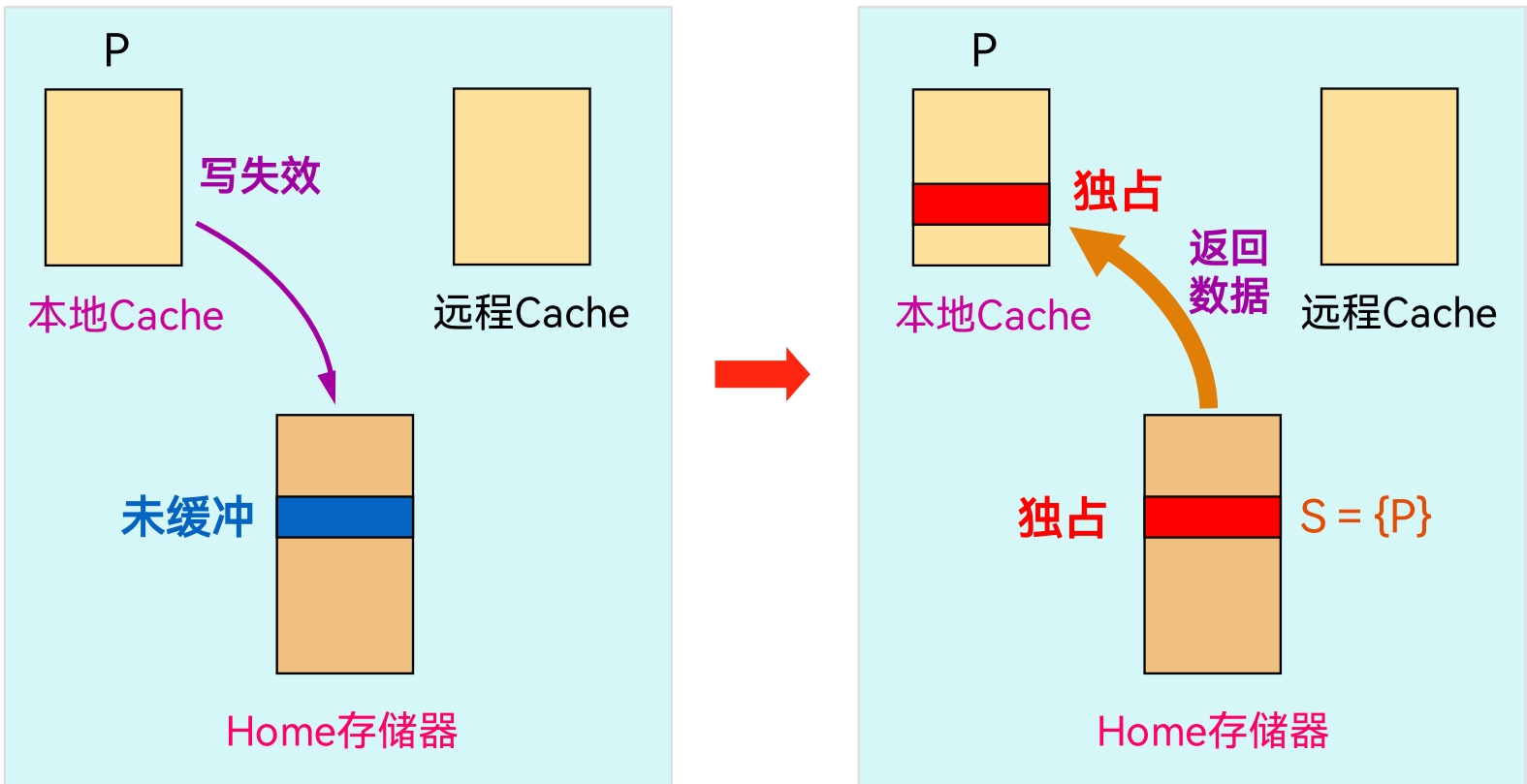
读失效



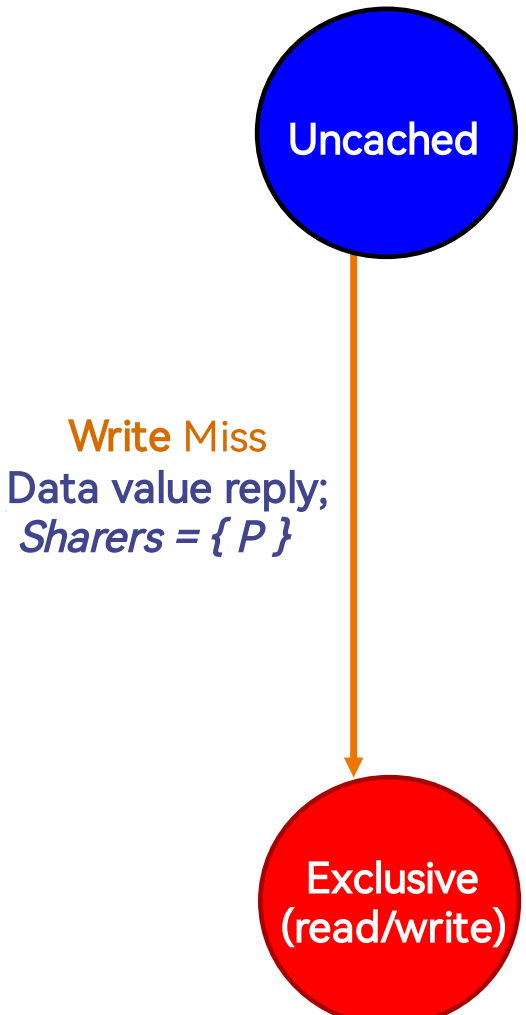

Uncached
Data value reply;
Sharers = { P }
Read miss
Shared
(read only)

- 数据块的状态为未缓冲

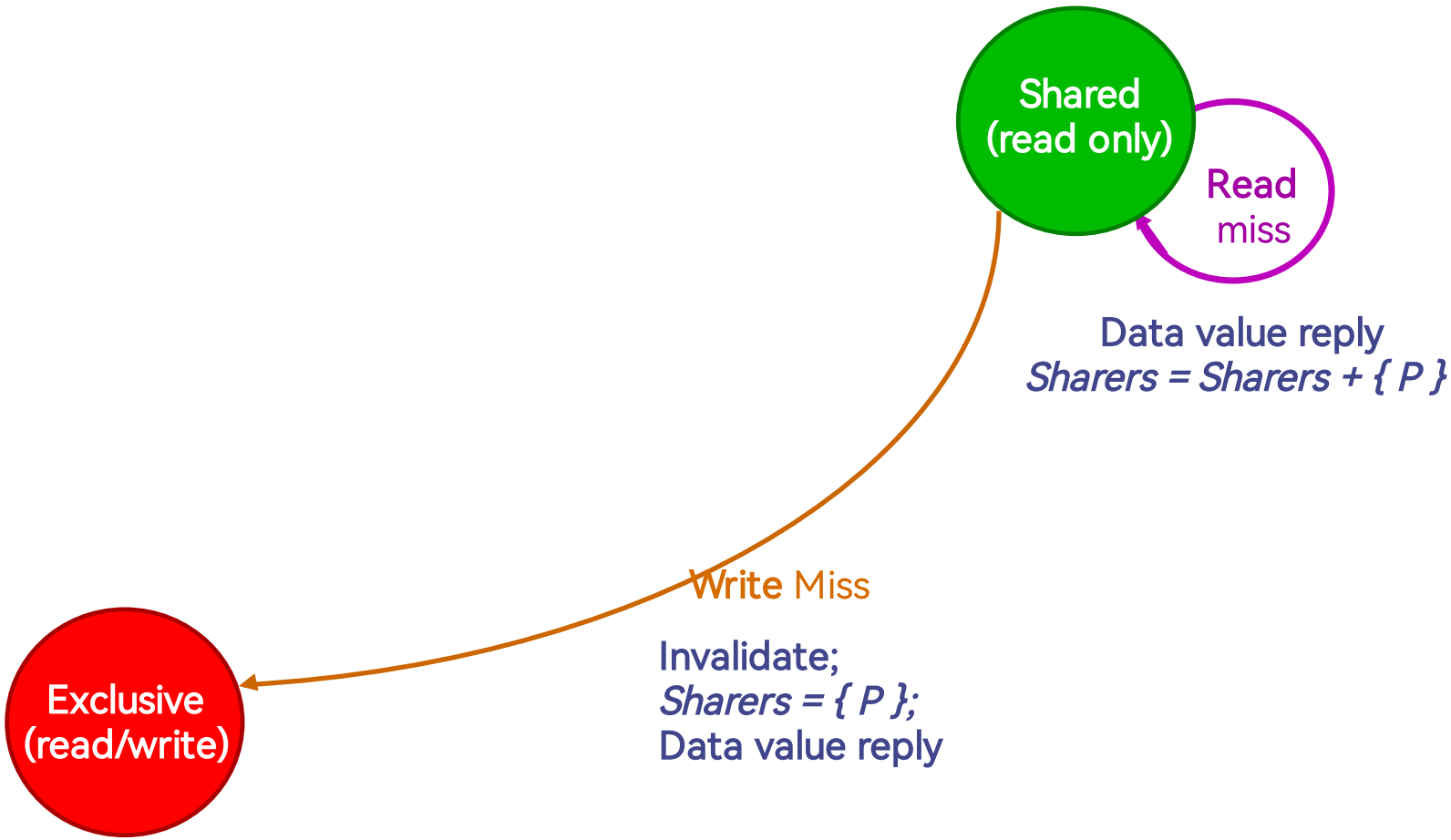
写失效



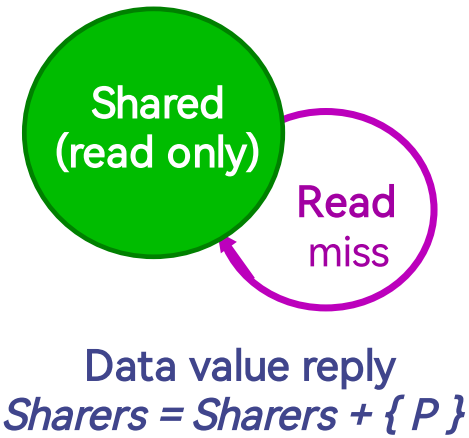
目录的状态转换图


Uncached
Write Miss
Data value reply;
Sharers = { P }
Exclusive
(read/write)

目录的状态转换图

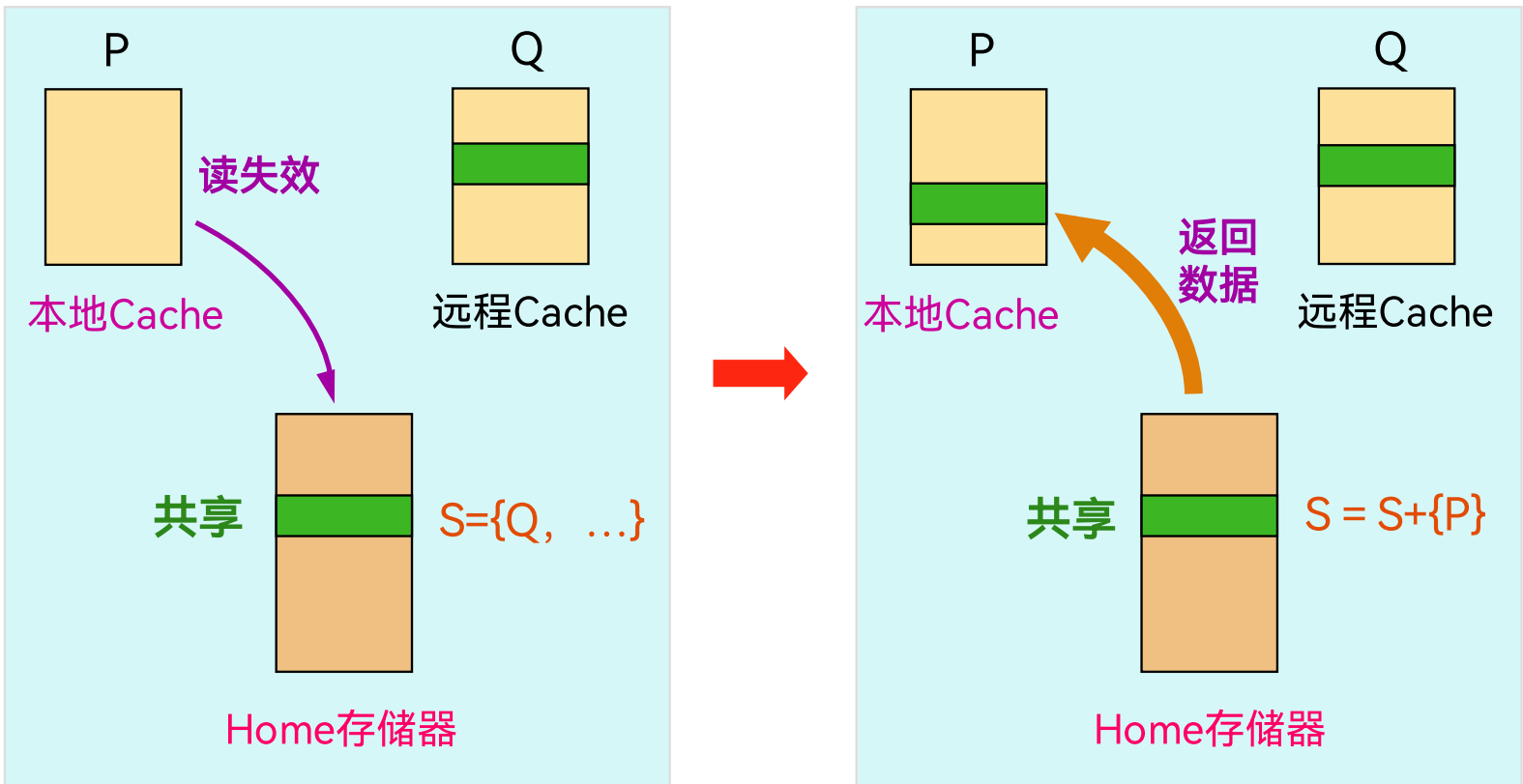

Shared
(read only)
Read
miss
Data value reply
Sharers = Sharers + { P }
Write Miss
Invalidate;
Sharers = { P };
Data value reply
Exclusive
(read/write)

目录的状态转换图

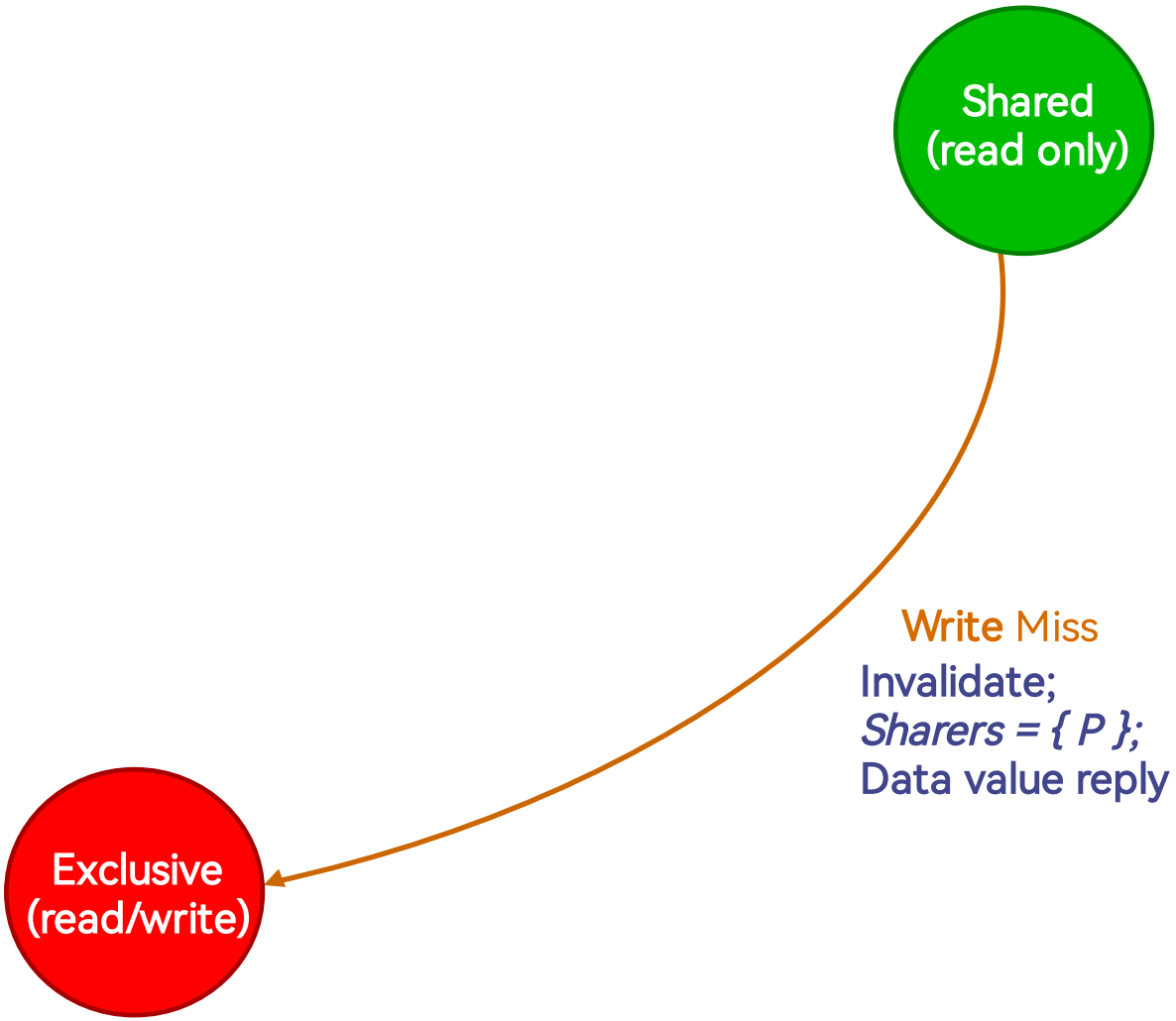

Shared
(read only)
Read
miss
Data value reply
Sharers = Sharers + { P }

- 数据块的状态为共享

读失效

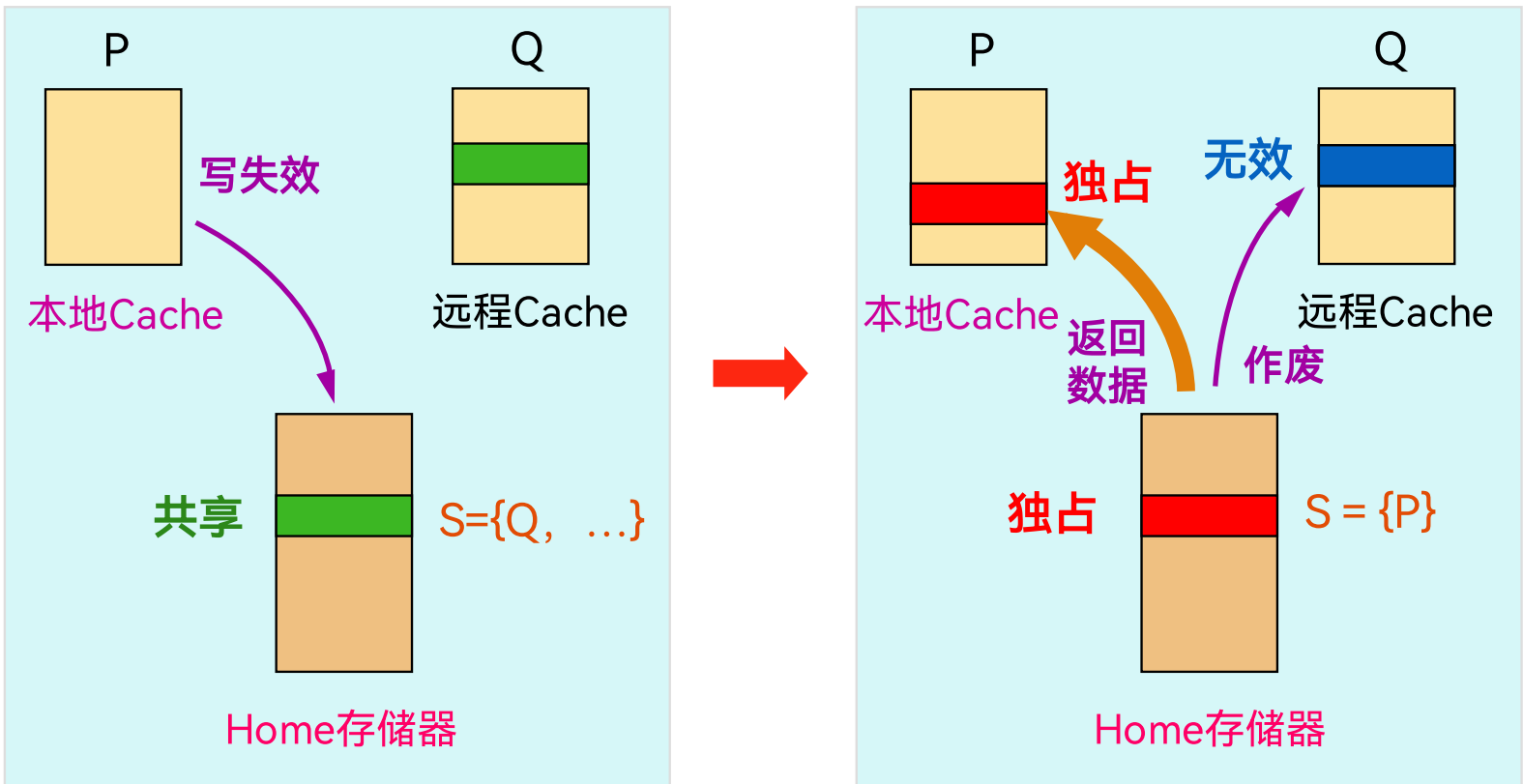


目录的状态转换图

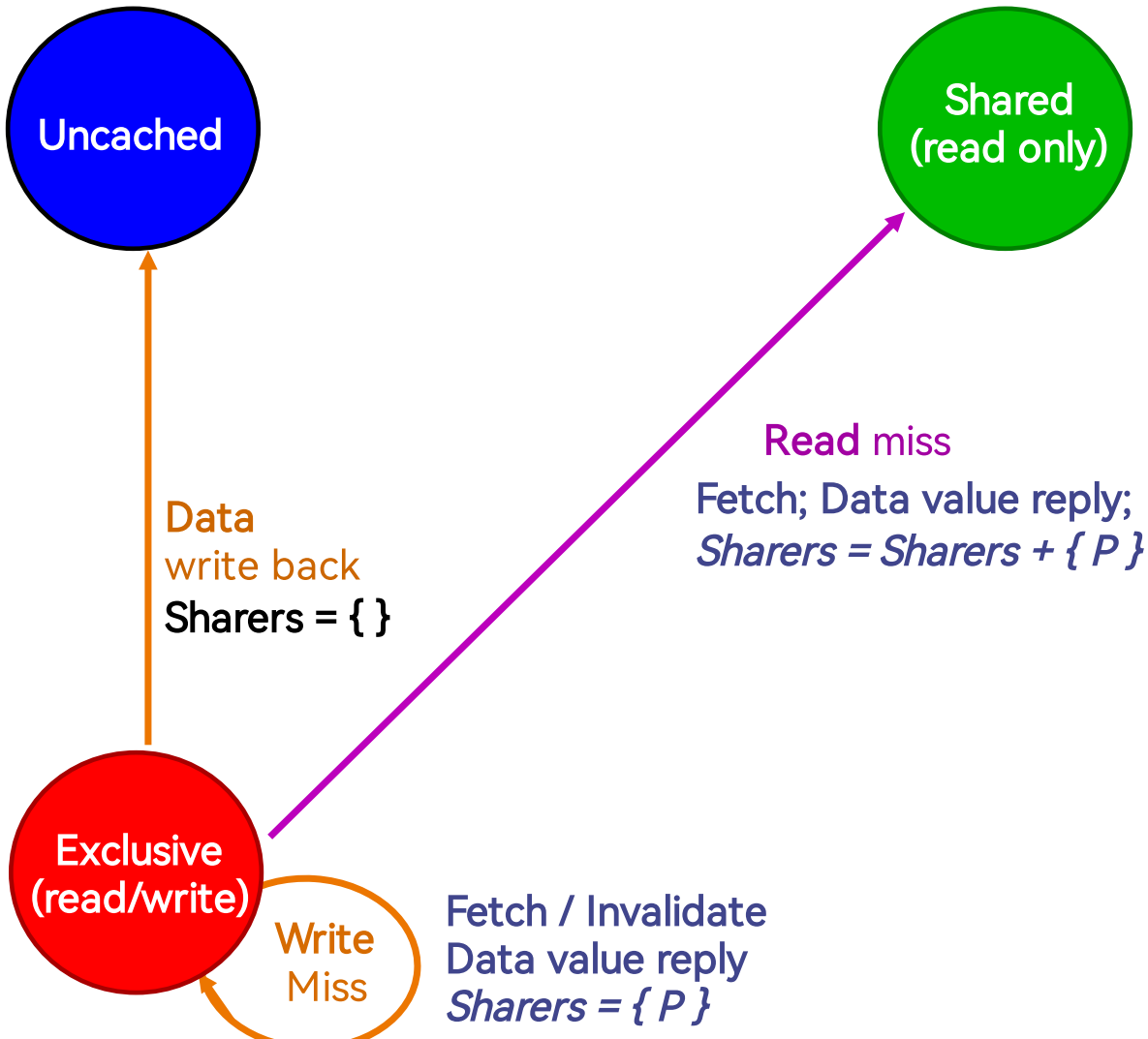

Shared
(read only)
Write Miss
Invalidate;
Sharers = { P };
Data value reply
Exclusive
(read/write)

- 数据块的状态为共享

写失效

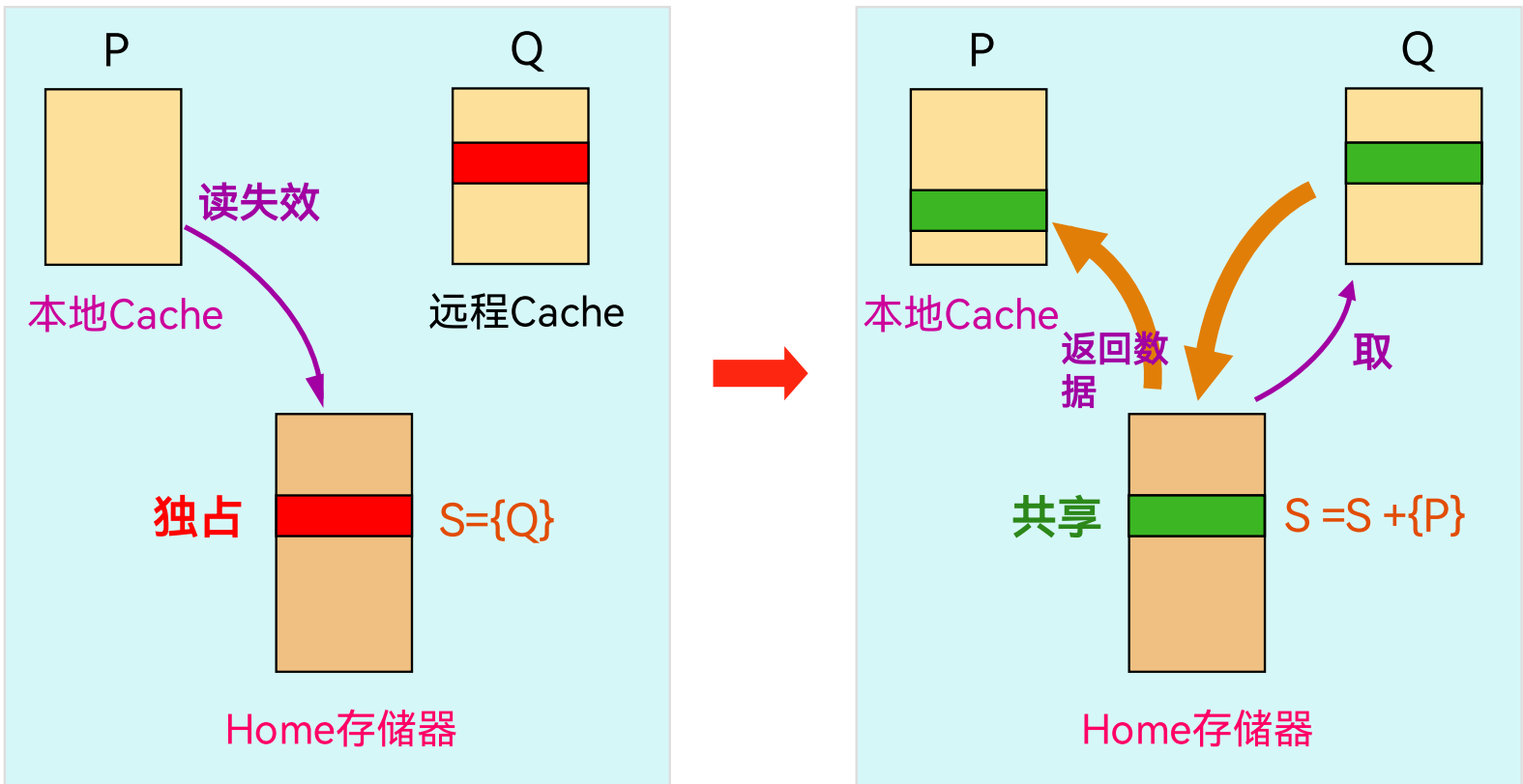


目录的状态转换图

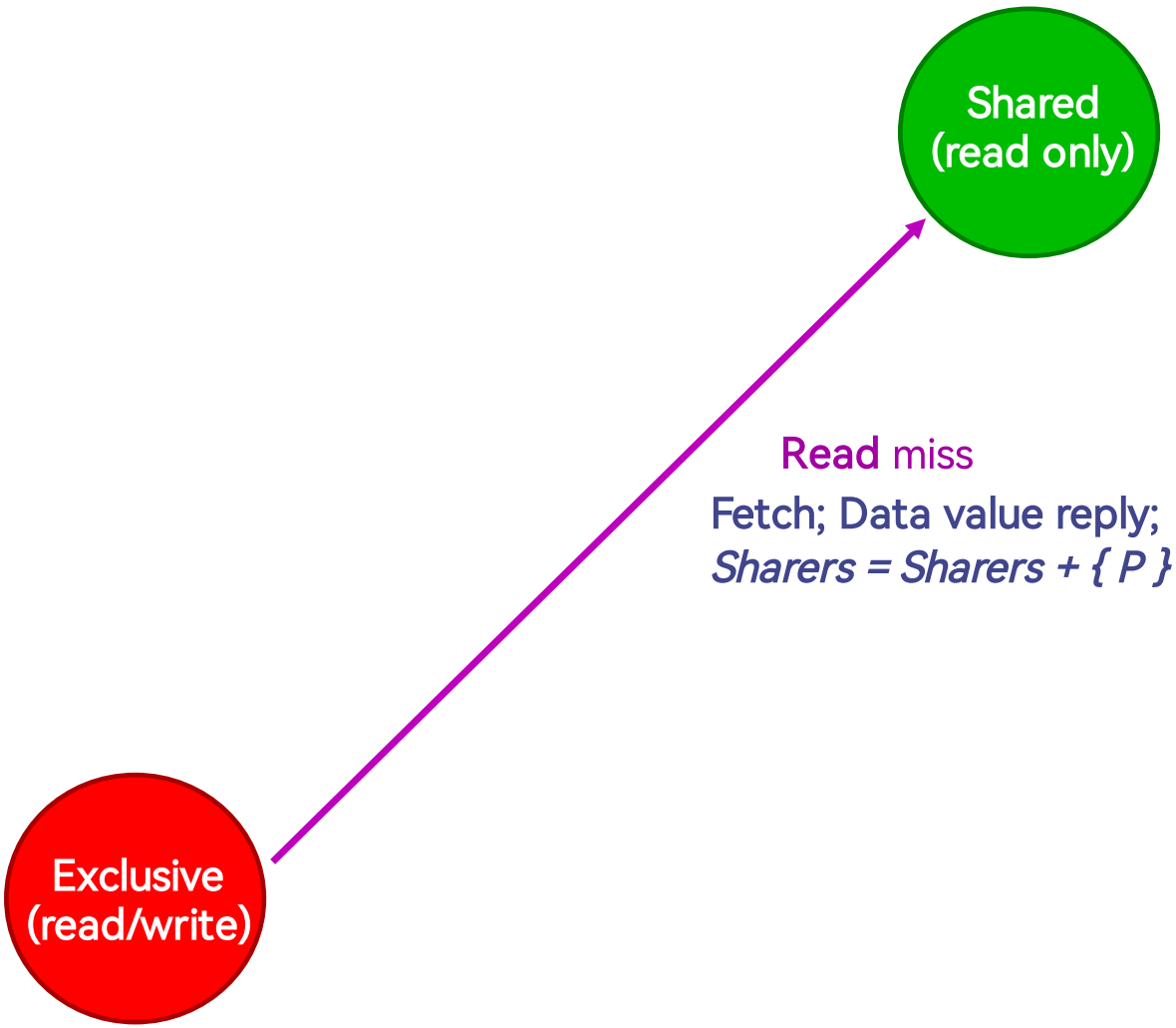

Uncached
Shared
(read only)
Read miss
Fetch; Data value reply;
Sharers = Sharers + { P }
Data
write back
Sharers = { }
Exclusive
(read/write)
Write
Miss
Fetch / Invalidate
Data value reply
Sharers = { P }

- 数据块的状态为独占

读失效

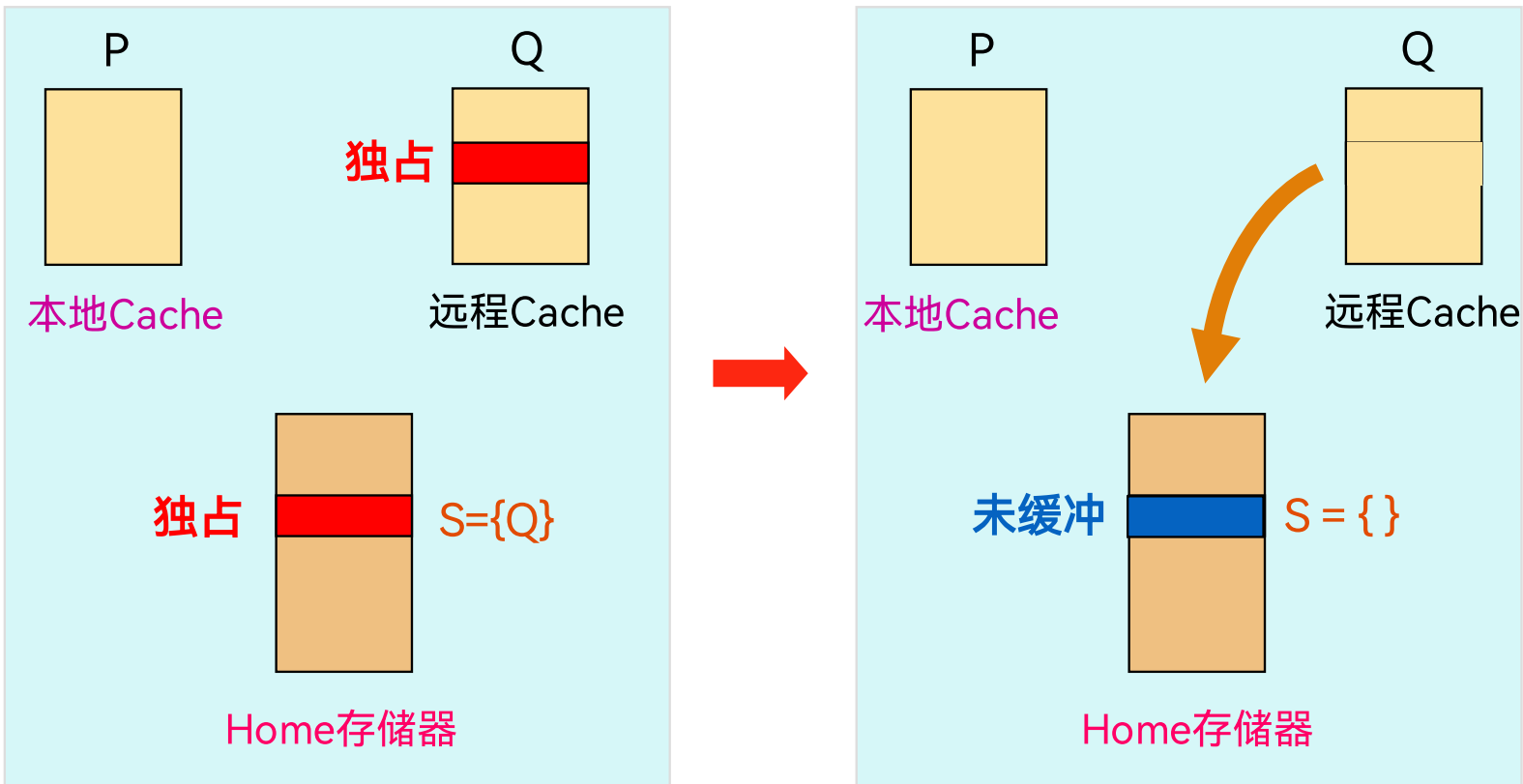


目录的状态转换图

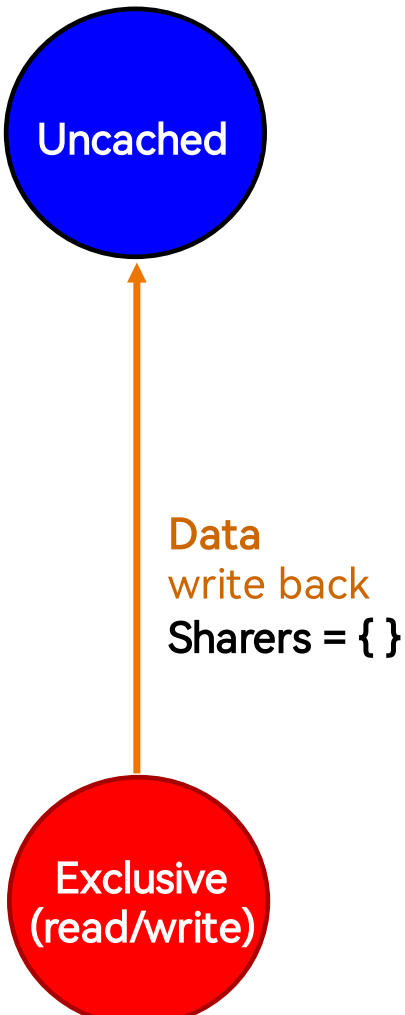

Shared
(read only)
Read miss
Fetch; Data value reply;
Sharers = Sharers + { P }
Exclusive
(read/write)

- 数据块的状态为独占

  数据写回（替换时发生）

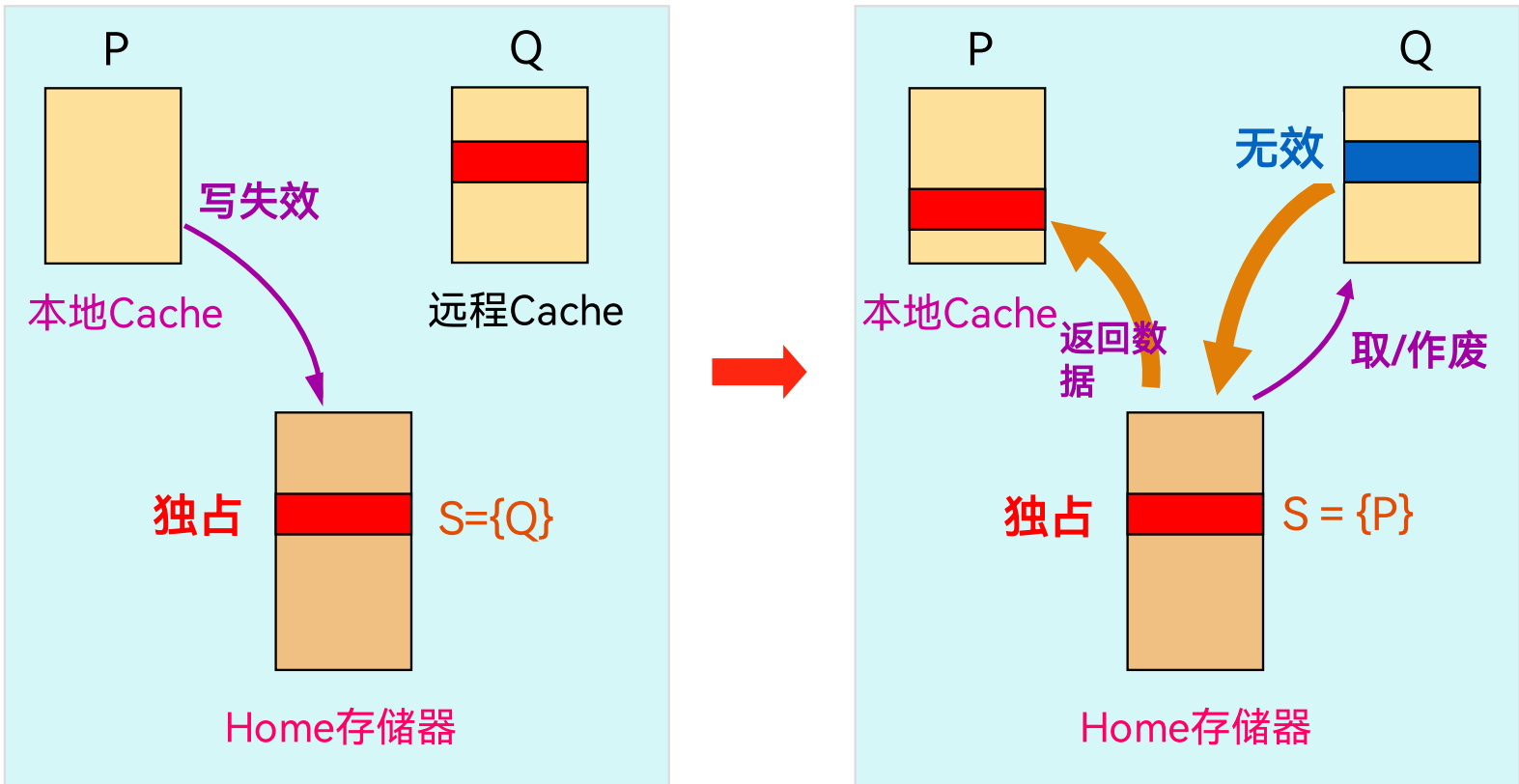


目录的状态转换图



- 数据块的状态为独占

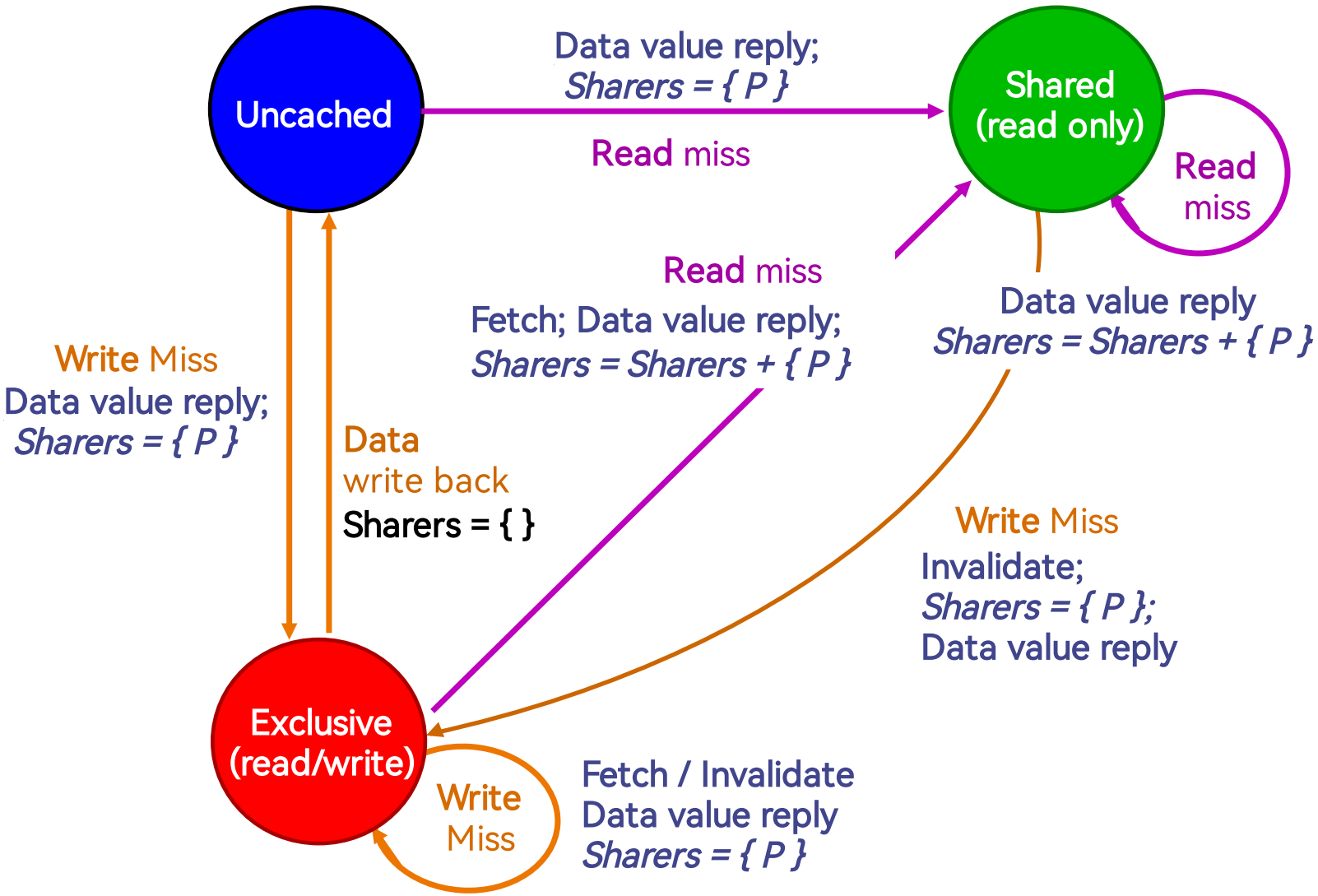
写失效



目录的状态转换图

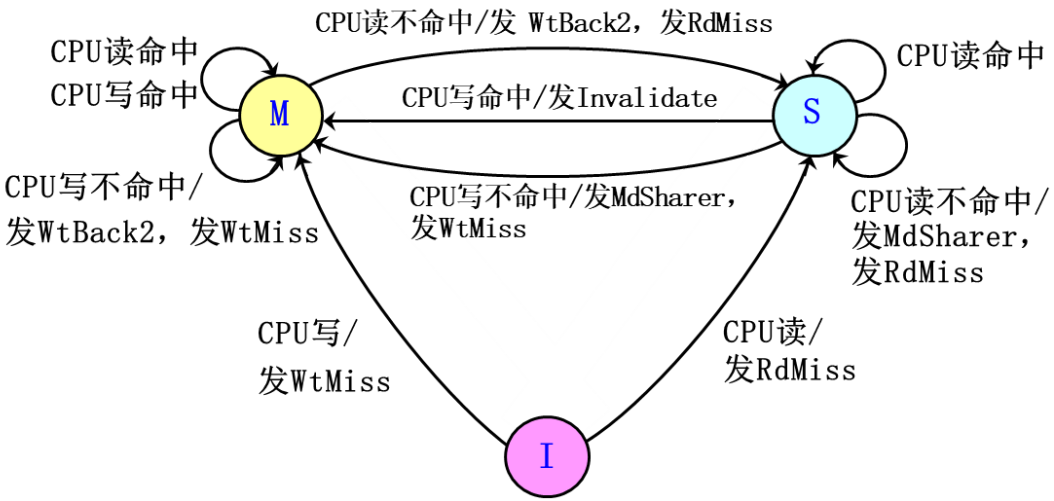

Exclusive
(read/write)
Write
Miss
Fetch / Invalidate
Data value reply
Sharers = { P }

目录的状态转换图

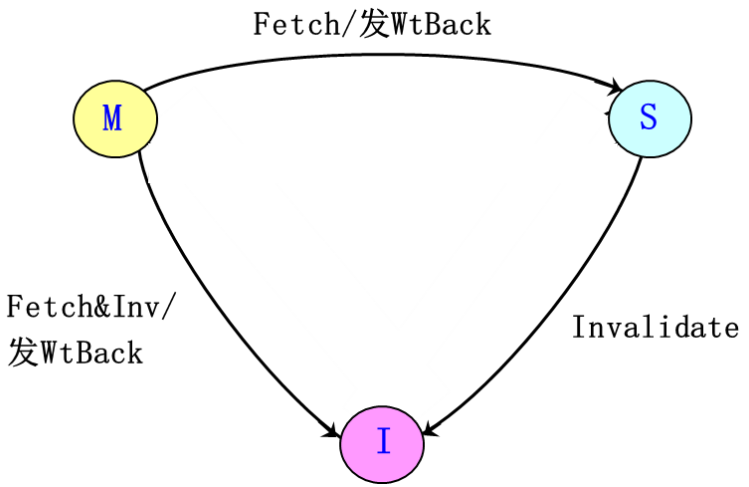

Uncached
Data value reply;
Sharers = { P }
Read miss
Shared
(read only)
Read
miss
Read miss
Fetch; Data value reply;
Sharers = Sharers + { P }
Data value reply
Sharers = Sharers + { P }
Write Miss
Data value reply;
Sharers = { P }
Data
write back
Sharers = { }
Write Miss
Invalidate;
Sharers = { P };
Data value reply
Exclusive
(read/write)
Write
Miss
Fetch / Invalidate
Data value reply
Sharers = { P }

① 在基于目录协议的系统中， Cache块的状态转换图。

➤ 响应本地 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

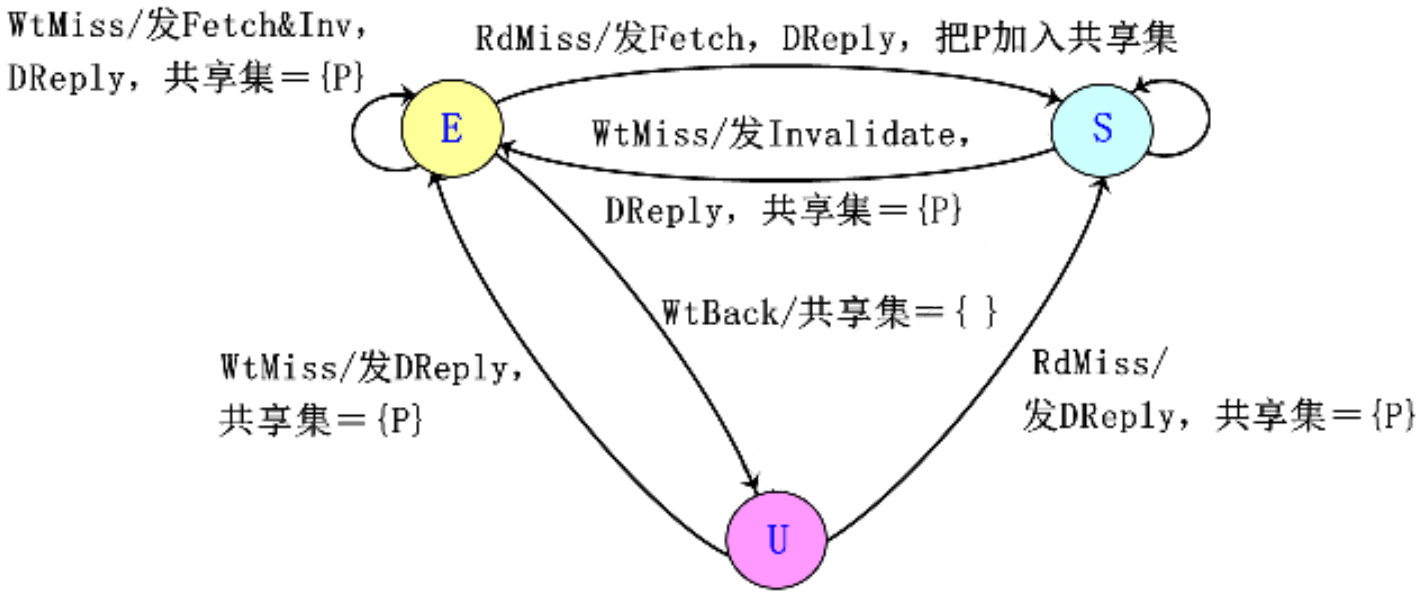
如前所述：

- 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。
- 对于从本地结点发来的请求，目录所进行的操作包括：
  - 向远程结点发送消息以完成相应的操作。这些远程结点由共享集合指出；
  - 修改目录中该块的状态；
  - 更新共享集合。

➤ 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

（假设这些操作是原子的）



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。

3. 目录的三种结构

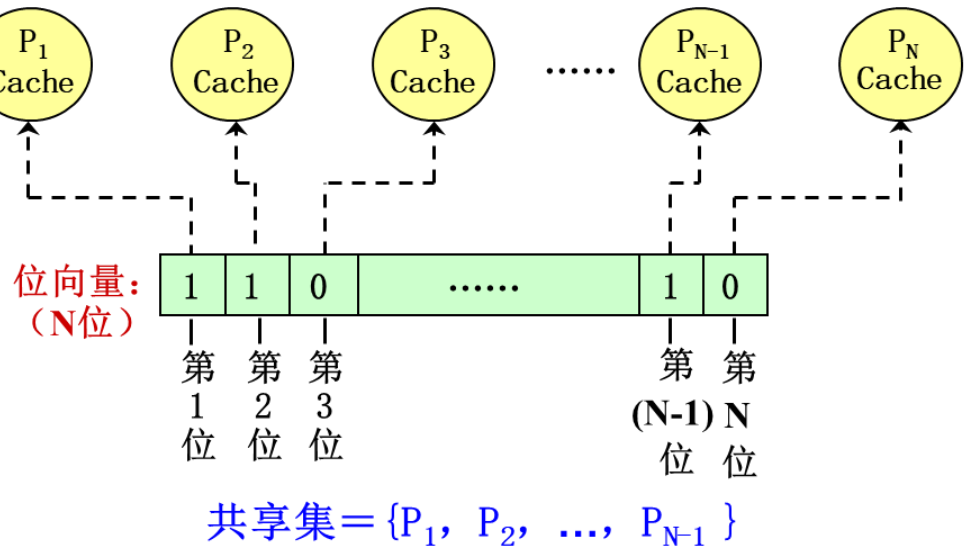
- 不同目录协议的主要区别主要有两个
  - 所设置的存储器块的状态及其个数不同
  - 目录的结构
- 目录协议分为3类
  - 全映像目录
  - 有限映像目录
  - 链式目录

① 全映像目录

每一个目录项都包含一个N位（N为处理机的个数）的位向量，其每一位对应于一个处理机。

- 优点：处理比较简单，速度也比较快。
- 缺点：
  - 存储空间的开销很大。
  - 目录项的数目与处理机的个数N成正比，而目录项的大小（位数）也与N成正比，因此目录所占用的空间与N^2成正比。
  - 可扩放性很差。

➤ 举例

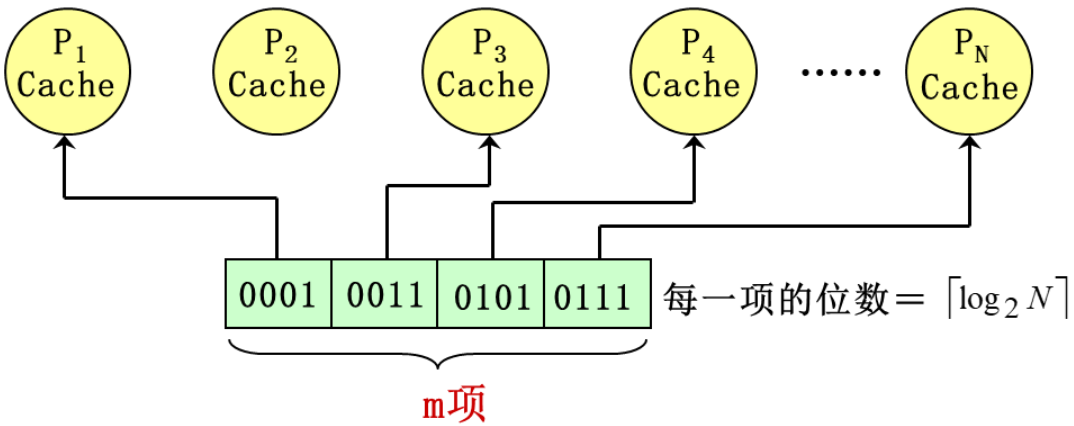


- 当位向量中的值为“1”时，就表示它所对应的处理机有该数据块的副本；否则就表示没有。
- 在这种情况下，共享集合由位向量中值为“1”的位所对应的处理机构成。

② 有限映像目录

- 提高其可扩放性和减少目录所占用的空间。
- 核心思想：采用位数固定的目录项目
  - 限制同一数据块在所有Cache中的副本总数。
  - 例如，限定为常数m。则目录项中用于表示共享集合所需的二进制位数为：$m \times \log_2 N$。
  - 目录所占用的空间与$N \times \lceil \log_2 m \rceil$成正比。
- 缺点
  - 当同一数据的副本个数大于m时，必须做特殊处理。当目录项中的m个指针都已经全被占满，而某处理机又需要新调入该块时，就需要在其m个指针中选择一个，将之驱逐，以便腾出位置，存放指向新调入块的处理机的指针 。

➤ 举例



共享集＝{P_1，P_3，P_4，P_7 }

有限映像目录（m = 4，N≥8的情况）

③ 链式目录

- 用一个目录指针链表来表示共享集合。当一个数据块的副本数增加（或减少）时，其指针链表就跟着变长（或变短）。
- 由于链表的长度不受限制，因而带来了以下优点：既不限制副本的个数，又保持了可扩展性。
- 链式目录有两种实现方法

  - 单链法

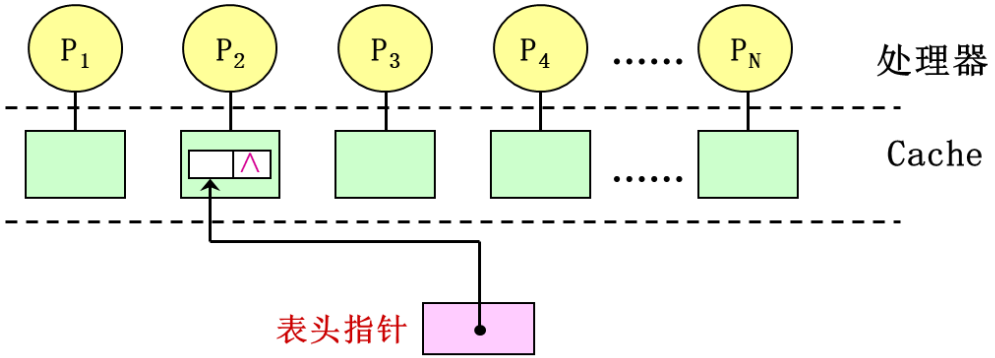
    当Cache中的块被替换出去时，需要对相应的链表进行操作——把相应的链表元素（假设是链表中的第i个）删除。实现方法有以下两种：

    - 沿着链表往下寻找第i个元素，找到后，修改其前后的链接指针，跳过该元素。
    - 找到第i个元素后，作废它及其后的所有元素所对应的Cache副本。

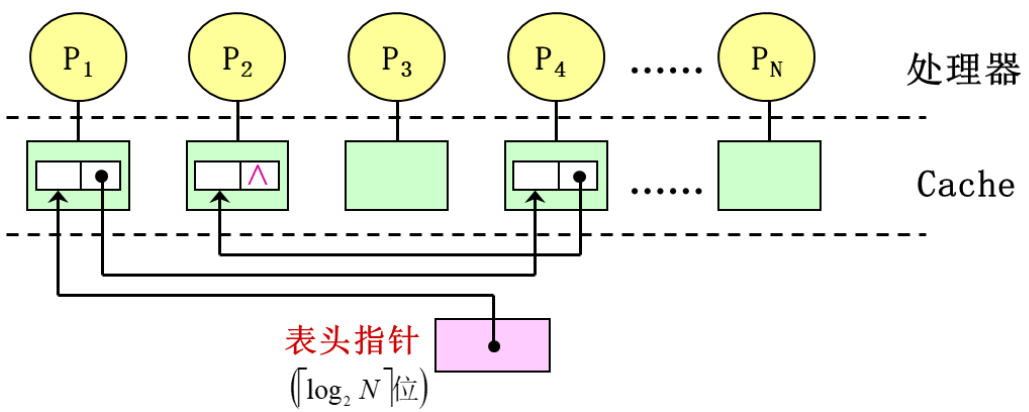
  - 双链法

    - 在替换时不需要遍历整个链表。
    - 节省了处理时间，但其指针增加了一倍，而且一致性协议也更复杂了。

采用单向链法的示意图



共享集 = { P_2 }



共享集 = { P_1, P_4, P_2 }