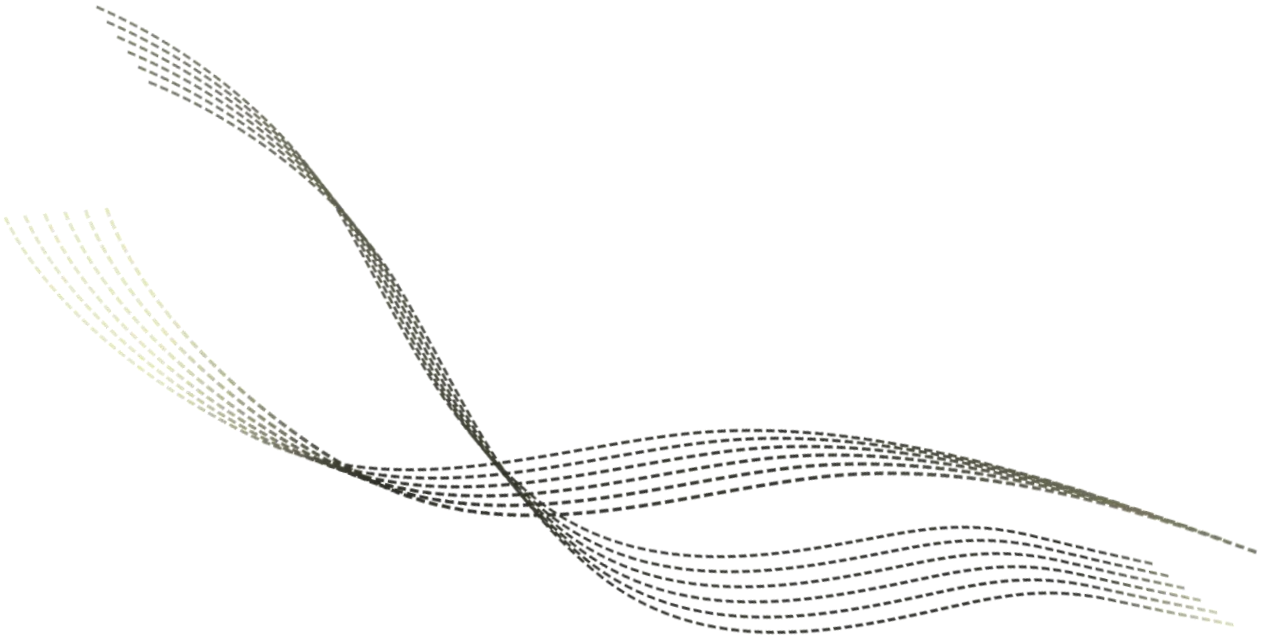
高级计算机体系结构

Advanced Computer Architecture



多处理器系统

沈明华

目录

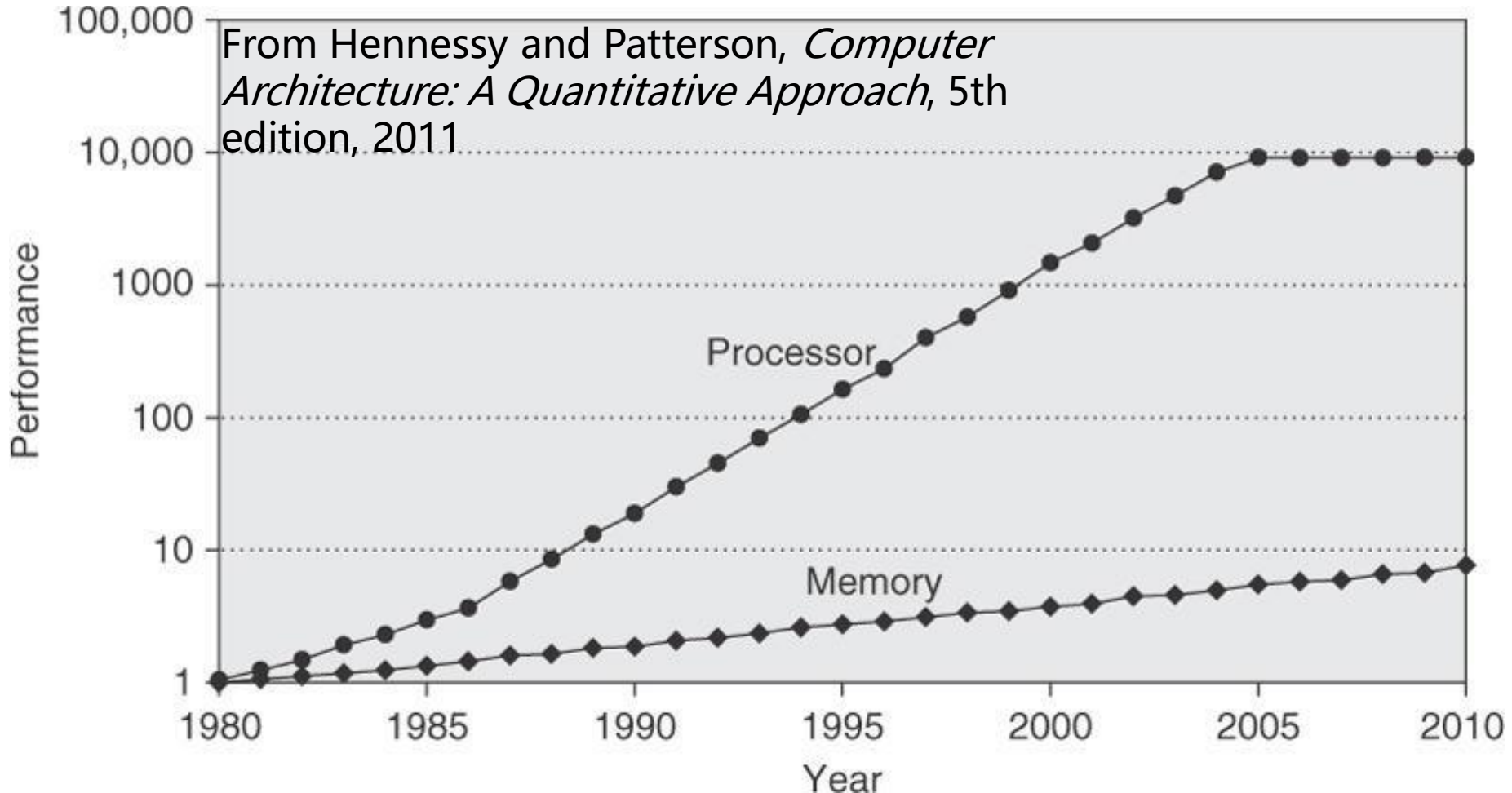
CONTENTS

PART 01 多处理器系统

■ 背景

□ 单一处理器性能提升遇到瓶颈

– 从2005年后开始，单处理器性能提升放缓



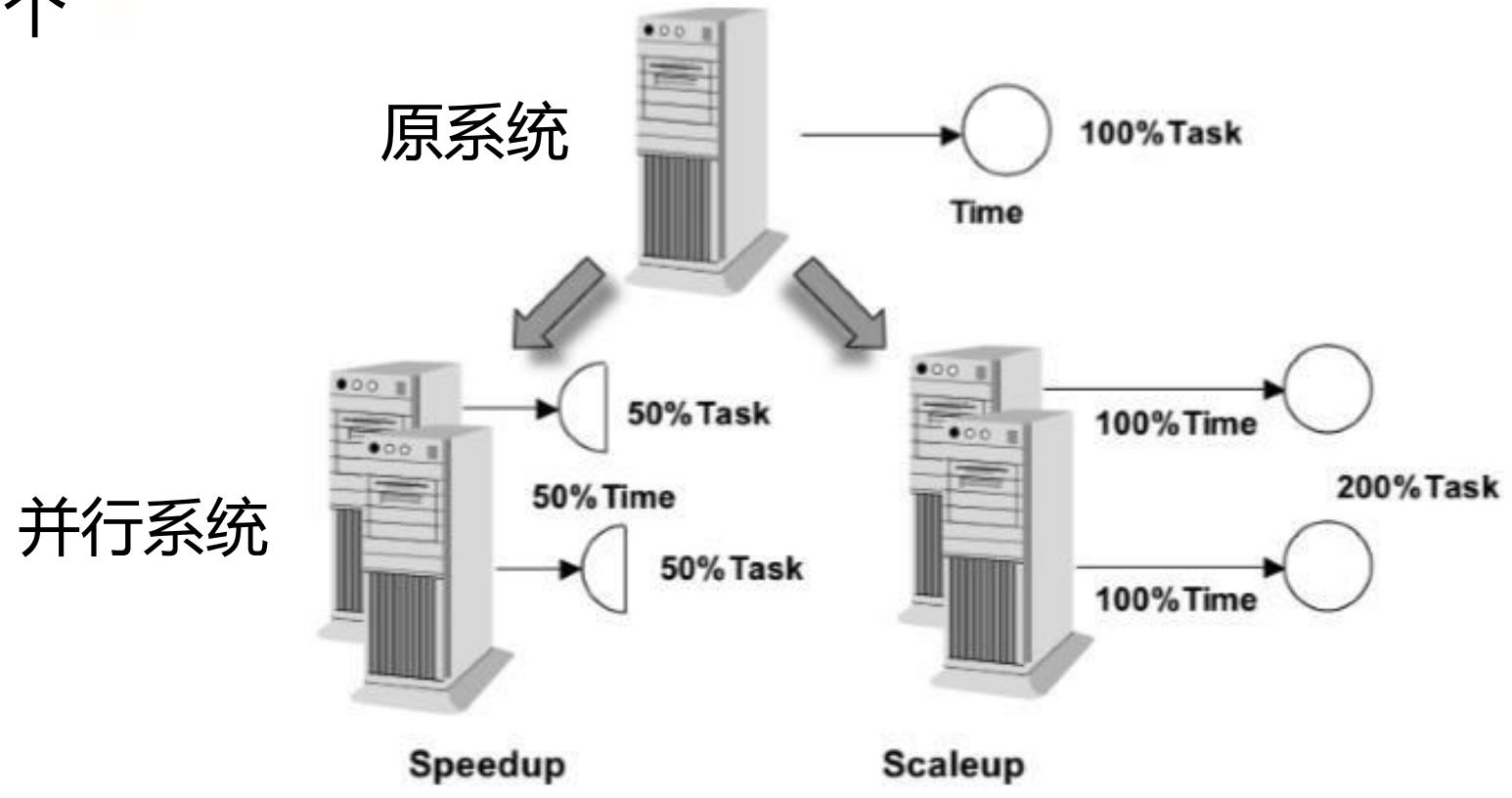
■ 背景

□ 多种因素促进多处理器系统研究

- 数据驱动应用的增长
  - 例如数据库系统、大数据系统……多任务并行需求增加
- 对服务器性能的关注
  - 云服务、云计算开始出现
  - 提高处理器性能的重要性下降，提高计算并行性的重要性上升
- 加深对多处理器系统的理解
  - 服务器的计算任务天生适合并行
- 并行计算机定义
  - A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast ——Almasi and Gottlieb, Highly Parallel Computing ,1989

■ 背景

- 并行计算机系统的可扩展性分为两个方向
  - 加速(speedup)扩展：投入n倍的计算设备，使得完成一件任务的用时变成原来的1/n
  - 规模(scaleup)扩展：投入n倍的计算设备，使得单位时间内完成的任务数量变成n个



■ 背景

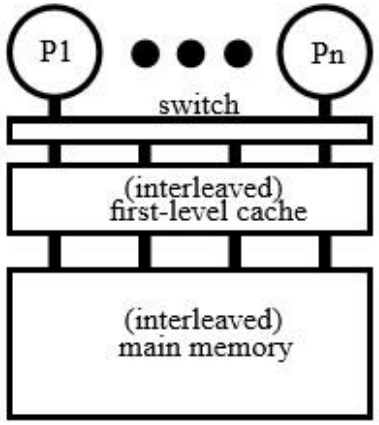
□ Flynn并行架构划分

- SISD(single instruction single data)
  - 一条指令流对应一条数据流
  - 例如通用处理器
- SIMD(single instruction multiple data)
  - 一条指令流同时作用到多条数据流上
  - 例如RISC-V的RVA23规范、x86的AVX指令集、ARM的neon指令集
- MISD(multiple instruction single data)
  - 没有类似的商业产品
- MIMD(multiple instruction multiple data)
  - 每个节点有自己的指令流和数据流　并行计算
  - 例如多处理器系统

■ 多处理器系统

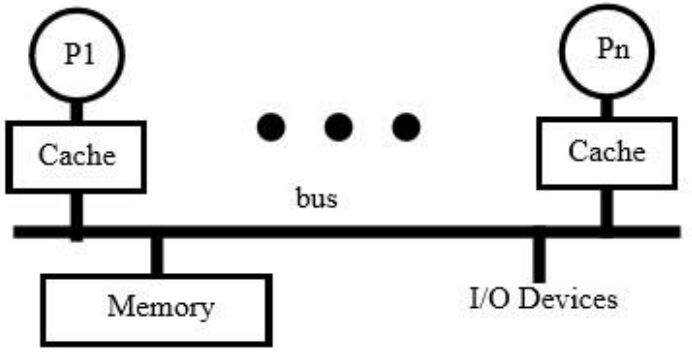
□ 几种常见的多处理器系统架构

所有处理器都通过一个开关连接到一级缓存
缺点是可能造成严重的拥堵



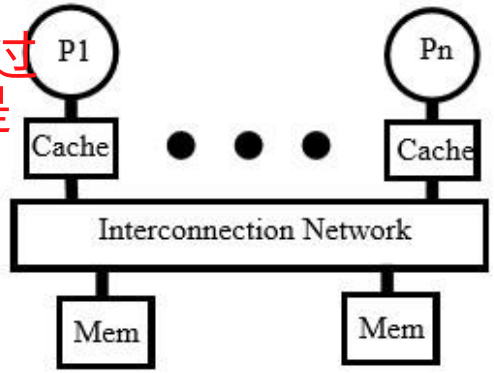
(a) Shared Cache

基于总线的共享内存，每个处理器都有缓存，通过总线共享内存；早期多核和对称多处理的常见形式。
缺点是总线带宽是瓶颈



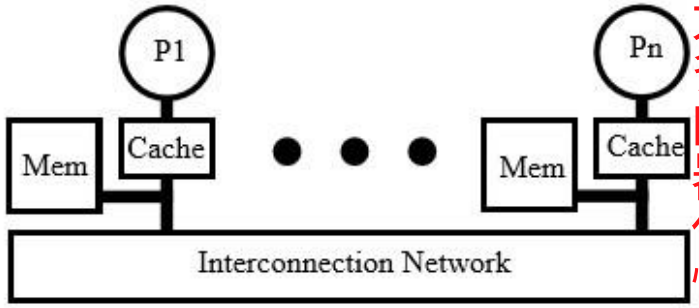
(b) Bus-based Shared Memory

舞厅模型：UMA，处理器通过互联网络访问内存，特点是任何一块CPU访问任何一块内存的延迟基本一样的



(c) Dance-hall

分布式内存架构，是目前大规模超级计算机和现代多路服务器（NUMA架构）的主流设计。特点是处理器之间通过互联网络通信，访问自己的内存非常快，其他的比较慢



(d) Distributed Memory

■ 多处理器系统

□ 两类多处理器系统结构

- 集中式共享内存(centralized shared-memory)
  - 多处理器共享同一个物理内存(UMA架构, uniform memory access)
- 分布式共享内存(distributed shared-memory)
  - 内存物理上分开，每个处理器有自己独有的可快速访问的内存(NUMA架构, non-uniform memory access)
- 共享内存：多个处理器的内存地址空间一致

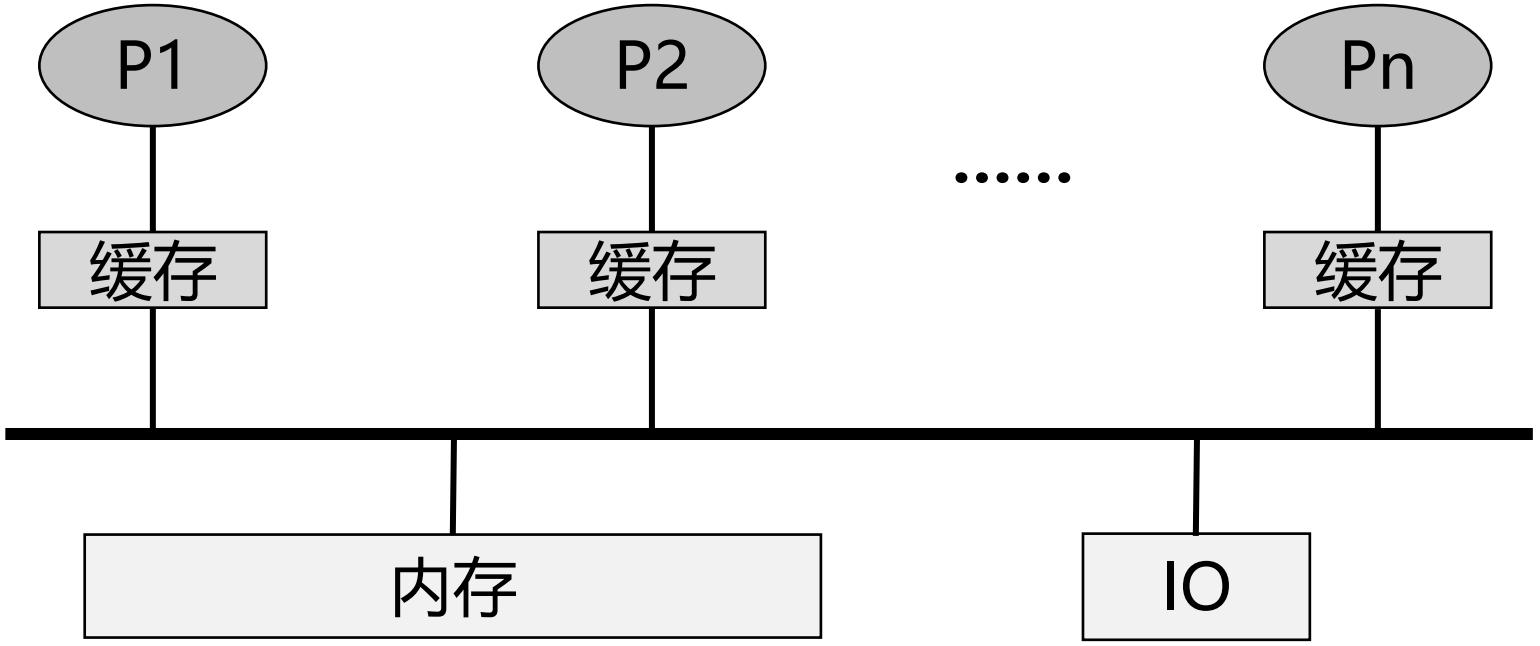
虽然物理分割，但具有连续的内存地址空间

■ 多处理器系统

□ 集中式共享内存

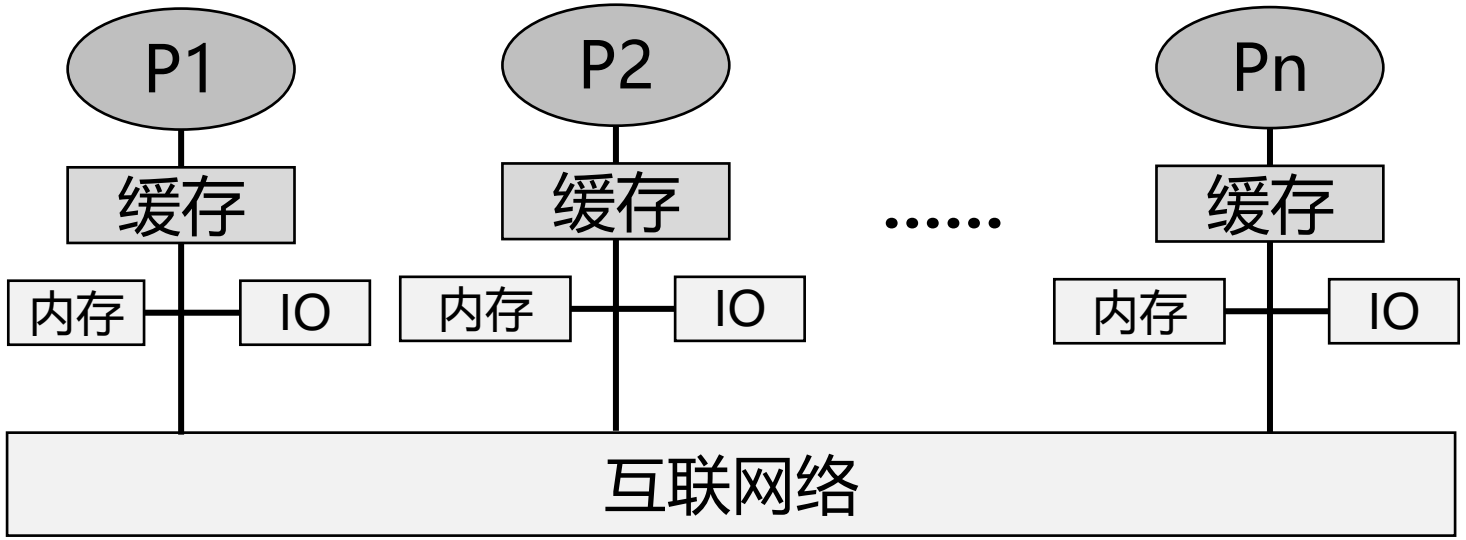
对称：所有核心地位完全平等；都挂靠在总线上访问唯一的内存和IO，所以核心多了会竞争，扩展性差

- 通常又被称为对称多处理器系统(SMP, symmetric multiprocessors)
- SMP所有处理器访问任意内存地址的时间一致



■ 多处理器系统

- 分布式共享内存(DSM, distributed shared-memory)
  - 优势：有效提高内存带宽，降低访问本地内存的延迟
  - 劣势：复杂的通信逻辑，较高节点间通信开销



■ 多处理器系统

□ 单机与多机条件下的MIMD系统

- 单机器(无论SMP还是DSM)

紧耦合

  - 是一个紧密结合的架构，处理器通过数据总线或互联网络实现互联
  - 包含的处理器较少(差不多就2个)，只有一个共享的内存地址空间
  - 通过load/store指令显式交换数据
  - 仅能支持线程/进程级的并行
- 多机器(例如仓库级计算机，WSCs, warehouse-scale computer)
  - 连接相对松散，每台独立的机器接入局域网
  - 可以包含大量的计算机器，且存在私有内存地址空间
  - 通过消息传递完成数据交换
  - 支持任务级并行

PART 02 缓存一致性问题

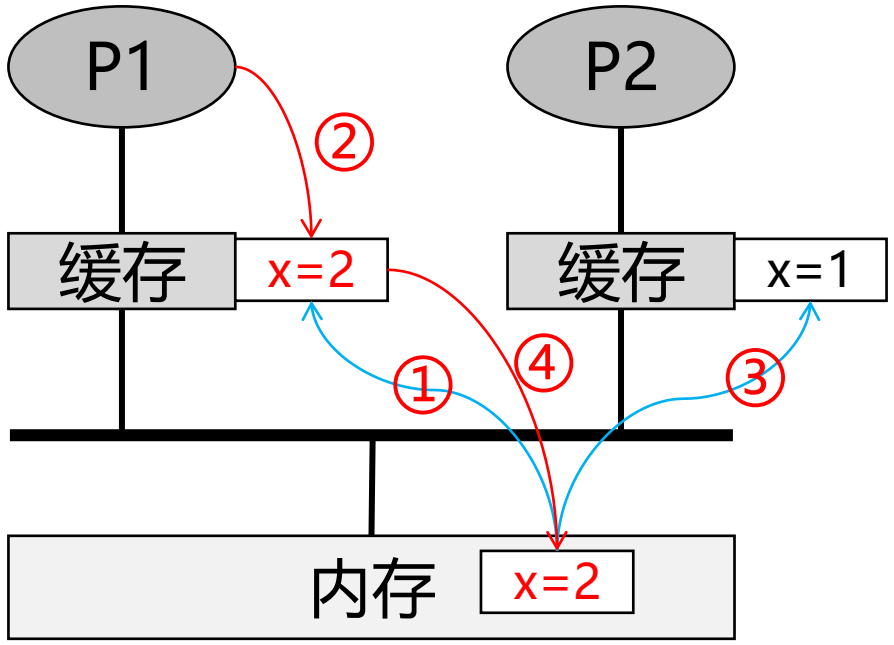
■ 背景

- 多处理器系统引入并行计算从而提高单台计算机的性能
  - 需要保证并行程序运行结果的正确
    - 无论底层是多个处理器还是单个多核处理器的系统，同一并行程序的运行结果应一致
  - 但缓存的存在，且处理器进行计算时仅从缓存读取数据的特性
    - 有可能导致处理器使用错误的数据执行计算
    - 进而导致并行程序运行结果的错误

■ 背景

□ 缓存一致性(cache coherence)

- 初始内存中x=1
- 处理器P1从共享内存读取x=1
- 然后处理器P1改写x=2，但没有立即写回内存
- 然后处理器P2读取到x=1
- 处理器P1的结果同步到内存
- **但处理器P1的写入没有同步到P2**
  - P2看到的x仍然是1
  - 出现缓存不一致问题



■ 缓存一致性

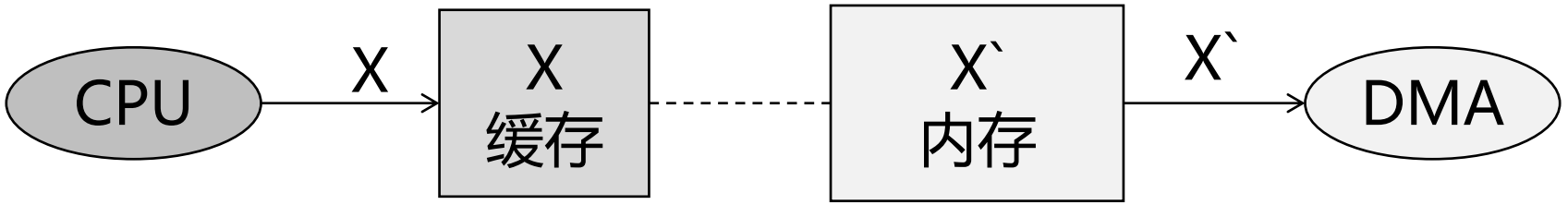
□ 缓存一致性问题

- 缓存一致性问题：不同处理器同一时刻看到的同一内存地址的值不同
- 无论多处理器还是单处理器都存在这个问题
  - ➢ 多处理器系统，每个处理器内部都有各自的缓存，处理器间存在一致性问题
  - ➢ 单个多核处理器，L1和L2缓存是每个核心独立，核心间存在一致性问题
  - ➢ 甚至单核处理器+IO操作，也可能存在一致性问题

■ 缓存一致性问题

□ 单核处理器+IO操作产生的一致性问题

- 首先，内存可以通过DMA(direct memory access)组件访问，DMA可以代替CPU完成磁盘和内存间数据传输
- 然后修改的值X在缓存中，没有来得及同步到内存
- 此时执行IO操作，DMA读取内存数据写入磁盘，把旧值X`写入磁盘
- 出现一致性错误，CPU和DMA两个组件看到同一内存地址的数据不同



■ 缓存一致性

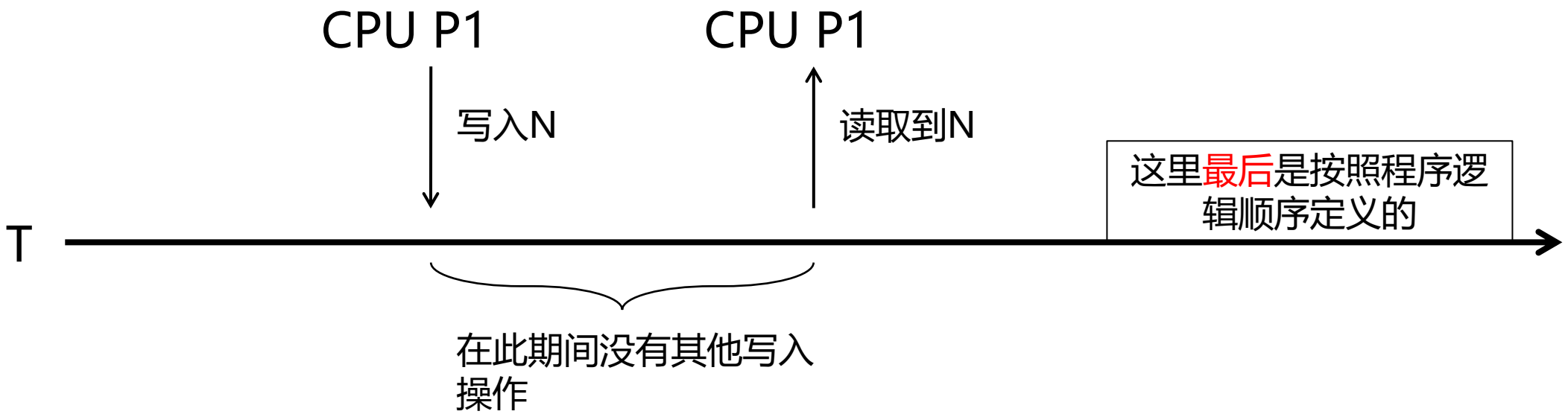
□ 缓存一致性应该保证

- 对于内存来说
  - ➢ 当读取某个地址的数据时，内存需要保证读取的值一定是最后写入的值
- 而对于多处理器的共享内存系统来说
  - ➢ 某个处理器读取内存地址X的值，读取到的结果一定是所有处理器最后向地址X写入的结果
- 在不同状况下，“最后”的定义略有不同

■ 缓存一致性要求

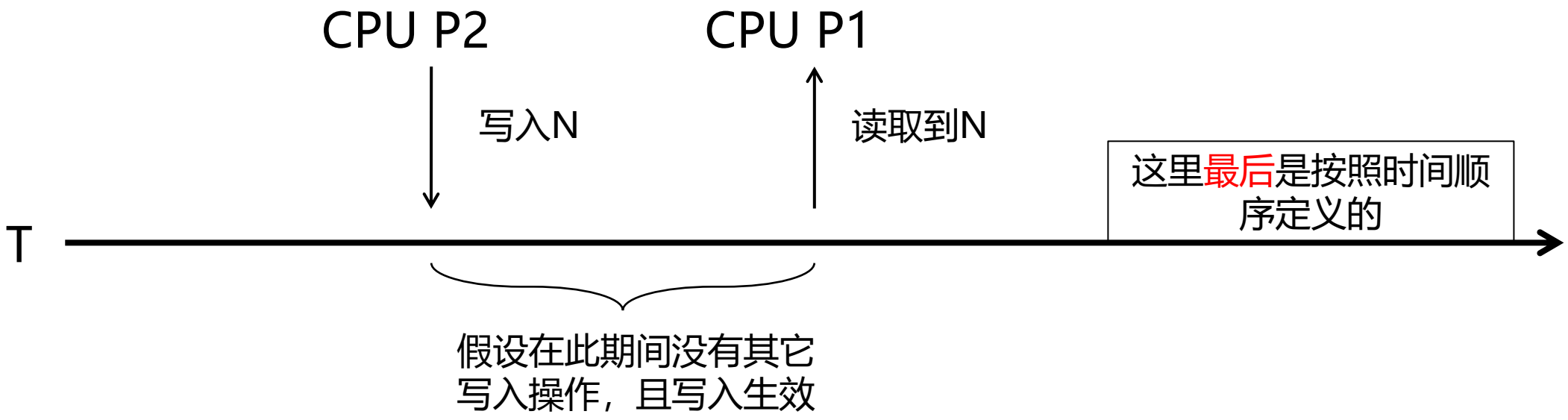
□ 情况一：RAW的一致性

– 假设P1处理器在内存地址X处写入值N，那么P1处理器执行后续指令读取地址X得到的值必须是N



■ 缓存一致性要求

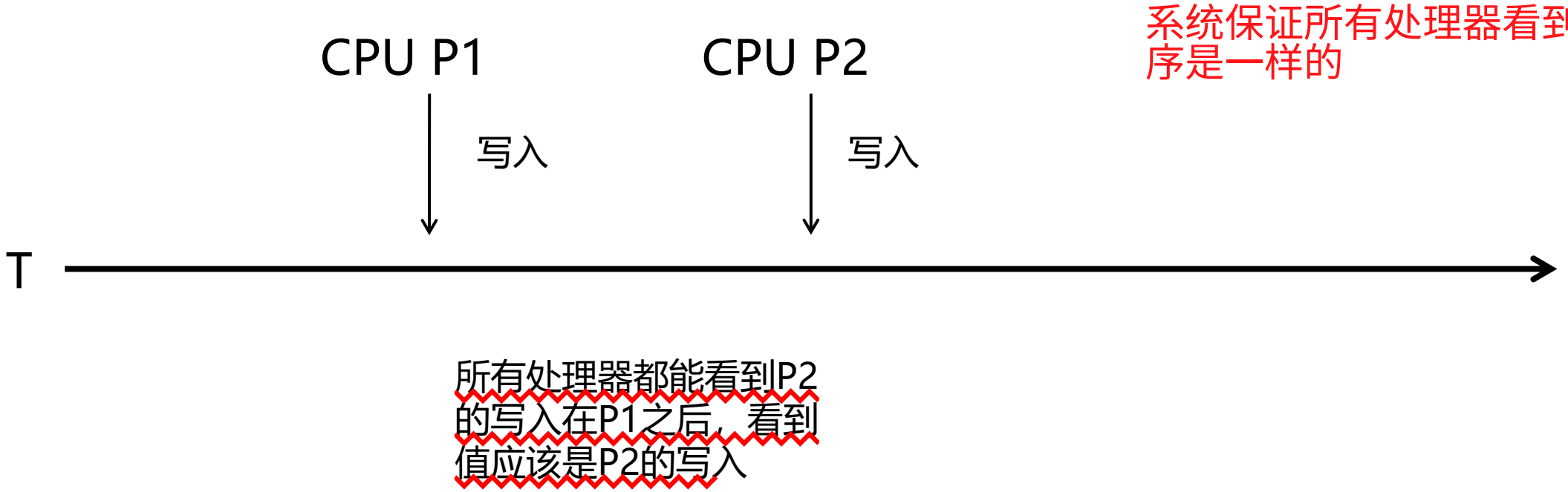
- 情况二：写入传播(write propagation)
  - 若处理器P2在地址X处写入值N，那么处理器P1能在地址X处读取到新值N
  - 只要处理器P1的读取操作发生在P2写入操作之后



■ 缓存一致性要求

◘ 情况三：串行写入

- 若处理器P1和P2同时写入地址X，那么所有其他处理器看到的写入顺序一致
  - ➢ 广播延迟可能导致处理器收到的写入顺序不一致
  - ➢ 但一致性要求所有处理器看到P1先写，P2后写的结果



■ 缓存一致性

□ 缓存一致性的常规定义

- 对于并行程序访问的任意一个地址X，都可以构建一个串行写入的记录，这个记录与指令执行顺序一致，并且
  - 由任何处理器发出的内存访问操作，会按照指令发出的顺序串行执行
  - 任何一次读取操作，都会返回地址X的串行执行记录中最后一次写入的结果
- 对于任意处理器，读取/写入一个地址的顺序只能是由程序定义
  - 带有一致性支持的存储系统保证所有处理器看到的写入/读取顺序一致

一致性与一贯性

- 一致性(coherence)强调读取值的异同
  - 定义哪些值可以被后续的读取看到
  - 更关心一个地址
- 一贯性(consistency)强调读取的时间概念
  - 定义何时写入的值可以被后续的读取看到
  - 需要同时关注多个地址
- 举个例子
  - 右边的并行程序初始化A=0，flag=0
  - 只有一致性不能保证flag=1时A=1

关心读到的值是不是最新的，大家看到的数据要一致

事件的发生要符合逻辑

```
P1              P2
A = 1;          while (flag == 0);
flag = 1;       print A
```

■ 关键点

- 关键点在于需要一种策略可以追踪数据块的状态变化，有两种经典协议
  - 窥探(Snooping)协议
    - 通过互联总线关注每一次数据更改(snooping)
    - 每个缓存都记录共享数据的状态
    - 若总线带宽足够高，可以非常快
  - 目录(directory)协议
    - 不通过总线广播数据更改
    - 通过目录记录缓存的共享状态
    - 更适合大量处理器的互联需求

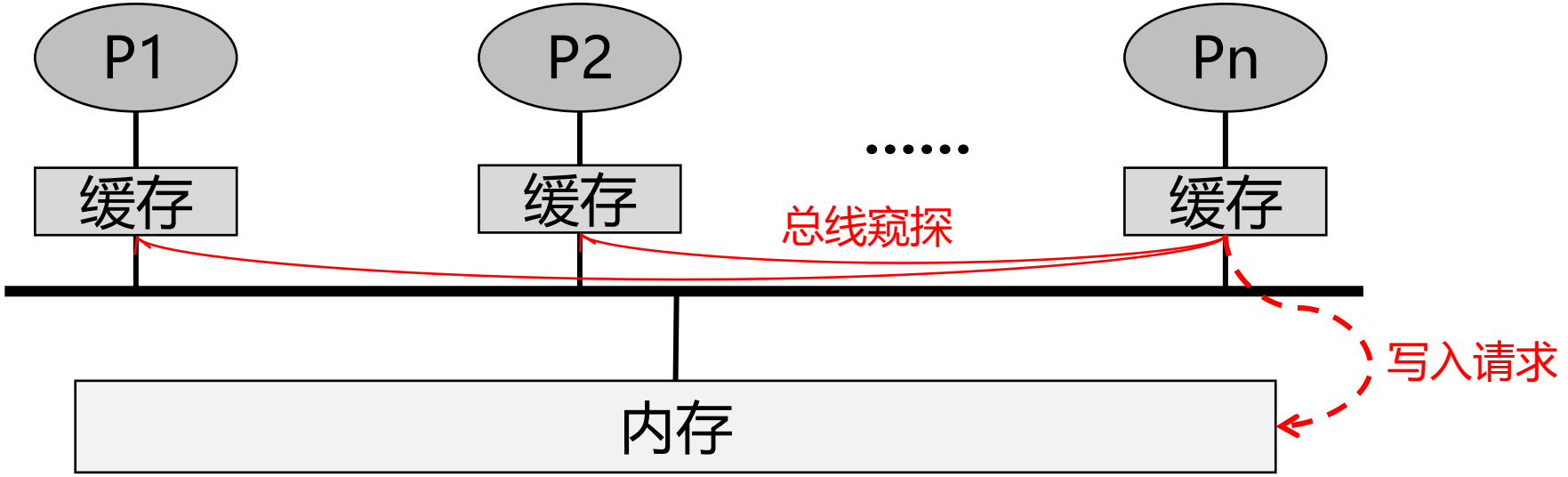
PART 03

Snooping协议

■ Snooping协议

□ 基本结构

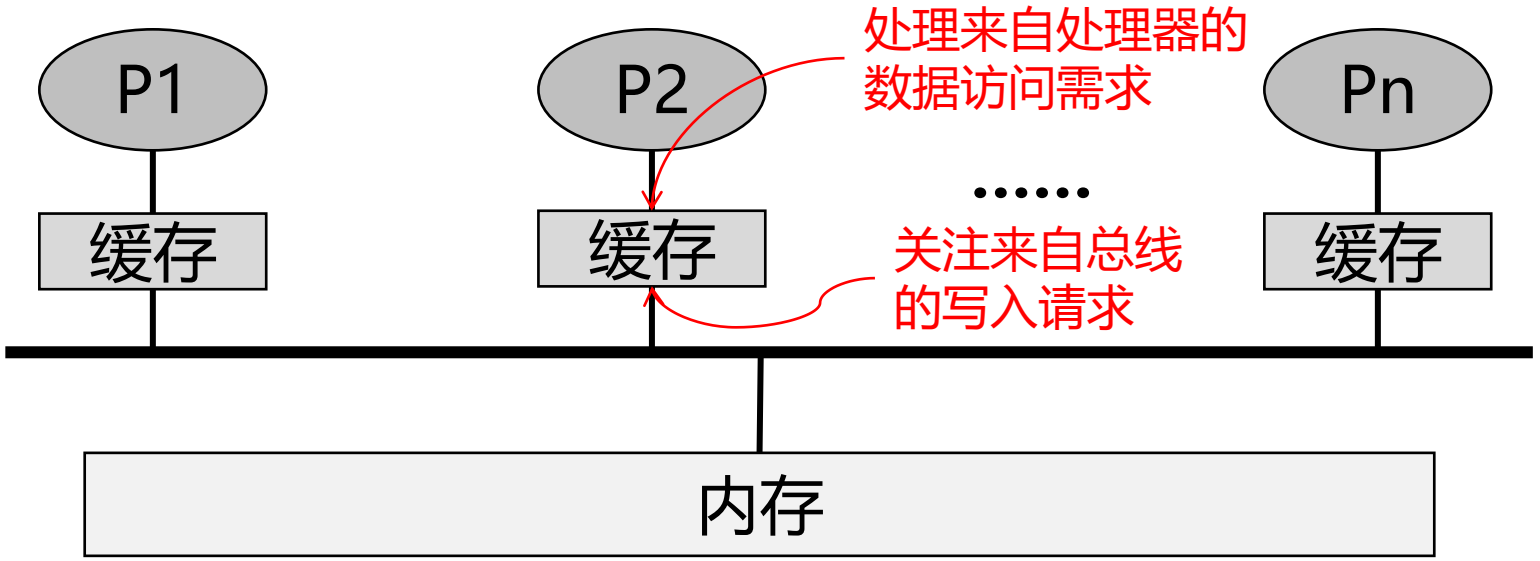
- 讨论Snooping协议之前设定的基本前提条件1
  - ➢ 带缓存的处理器用一条总线连接
  - ➢ 所有写入请求都通过总线传递到内存
  - ➢ 写入请求对任意处理器可见，且见到的写入顺序一致
  - ➢ 所有处理器不断的窥探总线，记录写入情况



■ Snooping协议

□ 基本结构

- 讨论Snooping协议之前设定的基本前提条件2
  - ➢ 增强缓存功能，缓存需要同时兼顾两个方向的数据，存在两个缓存控制器
  - ➢ 一个用于处理来自处理器的数据访问需求，正常从内存中获得数据
  - ➢ 另一个关注来自总线的写入请求，及时更新最新数据(snooper)



■ Snooping协议

□ Snooping协议

- 是一个分布式的协议
  - ➢ 每个本地缓存行都有自己的状态
  - ➢ 缓存行的状态转化过程是一个有限自动机
  - ➢ 每个读取、写入操作都触发状态转换
- 分为两类
  - ➢ 基于更新策略的：在写入时，更新其他处理器上的缓存，广播最新值
  - ➢ 基于失效策略的：在写入时，令其他处理器上的缓存失效，使得其他处理器读取时必须再次从内存读取
  - ➢ 结合缓存的写直通(write through)和写回(write back)两种策略，排列组合一共有4种不同的一致性协议设计

■ Snooping协议

- 以写直通失效策略为例
  - 每个缓存行包含两个状态
    - 数据在缓存中，且是最新的值，为有效状态V
    - 若当前数据在缓存，但不是最新值，为无效状态I
    - 若不在缓存中，同为无效状态I
  - 另要求更新过程原子化，单位时间只能处理一个写入请求

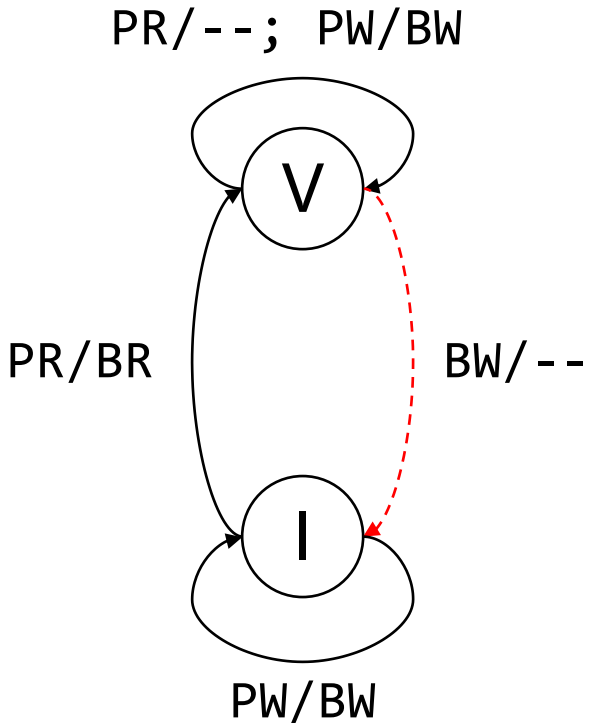
数据过期了，只能从内存拉过来

■ Snooping协议

□ 写直通失效策略Snooping协议

– 有限状态机转化如右图所示，其中

- V表示缓存有效，可直接使用，又称共享状态(shared)；
  I表示缓存失效，需要重新读取
- A/B的标记表示若观察到A就产生B操作
- P表示处理器主动产生的请求，B表示来自总线的请求
- R表示读取，W表示写入
- PR表示处理器读取请求，其他组合以此类推



■ Snooping协议

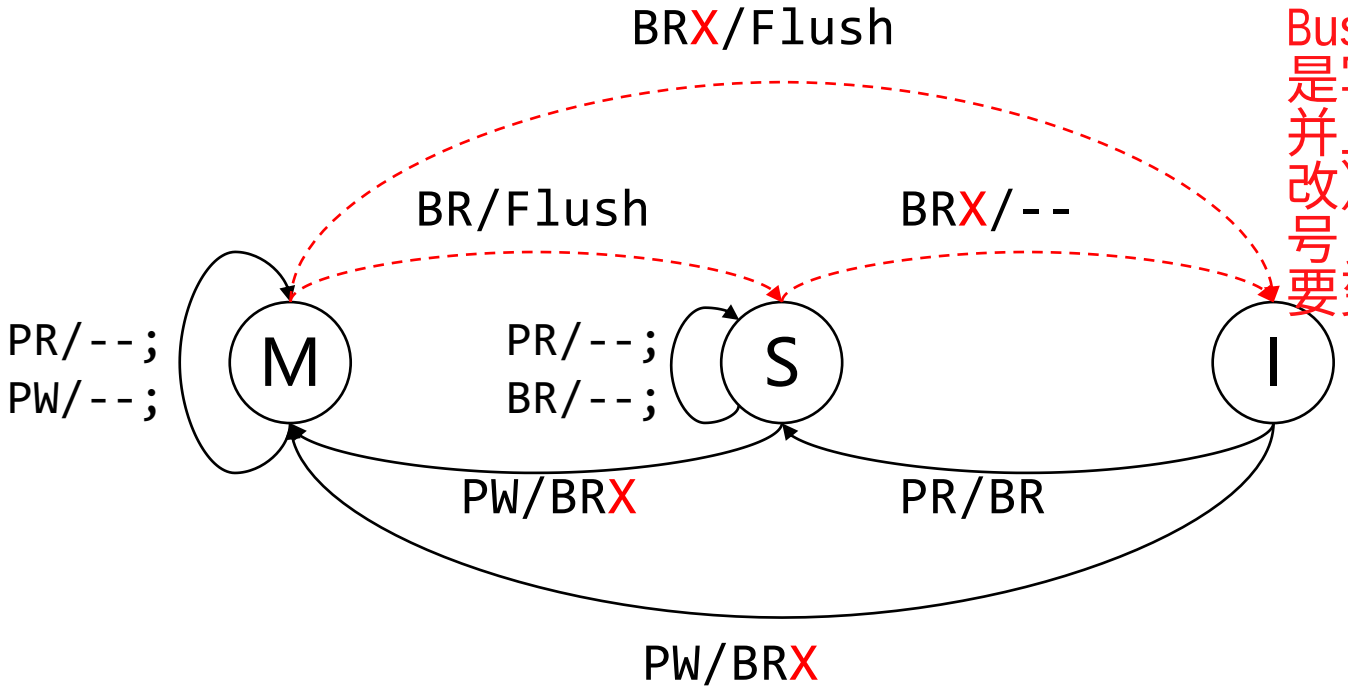
□ 写直通失效策略的不足

- 写直通需要很高的带宽，因为所有的写操作都通过总线广播到所有处理器和内存
  - 考虑一个处理器以1GHz的速度运行，CPI=1.5且执行的指令中15%是store指令，每个store指令平均修改8字节数据，那么1GB/s带宽的总线能支持多少个这样的处理器?
  - 计算得到单个处理器写直通所需带宽量：$0.15 \times 10^9 \times \frac{1}{1.5} \times 8 = 0.8$GB/s，可知1GB/s带宽的总线仅能满足一个处理器的写入需求
- 因此写直通策略一般不采用，而采用写回节省带宽需求
  - 但需要更加复杂的一致性控制策略

■ Snooping协议

□ 三状态写回失效策略Snooping协议，即MSI

- MSI简称源自于采用三种状态：修改(modify)、共享(shared)、失效(invalid)
- X表示独占，说明有处理器需要独占数据进行处理，其余处理器收到消息后令本地缓存失效

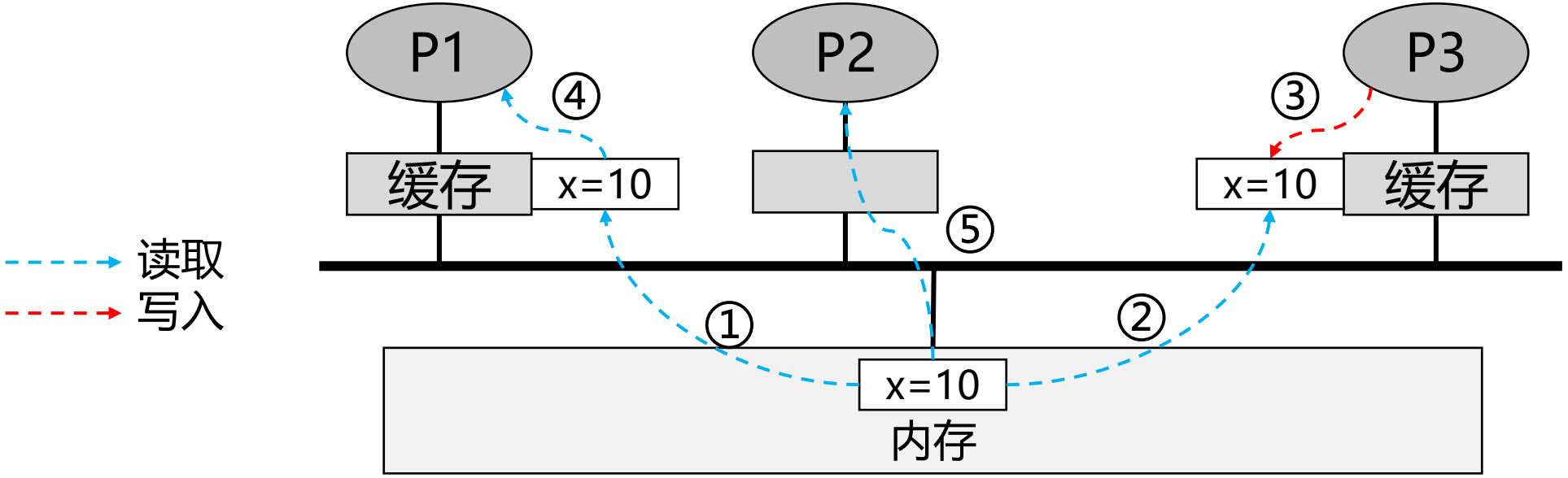


BRX (Bus Read Exclusive / Bus Read for Ownership)：这是写信号。意思是“我要读进来并且独占它（因为我要修改）”。一旦总线上出现这个信号，所有处于 S 状态的缓存都要变成 I。

■ Snooping协议

□ MSI运行示例

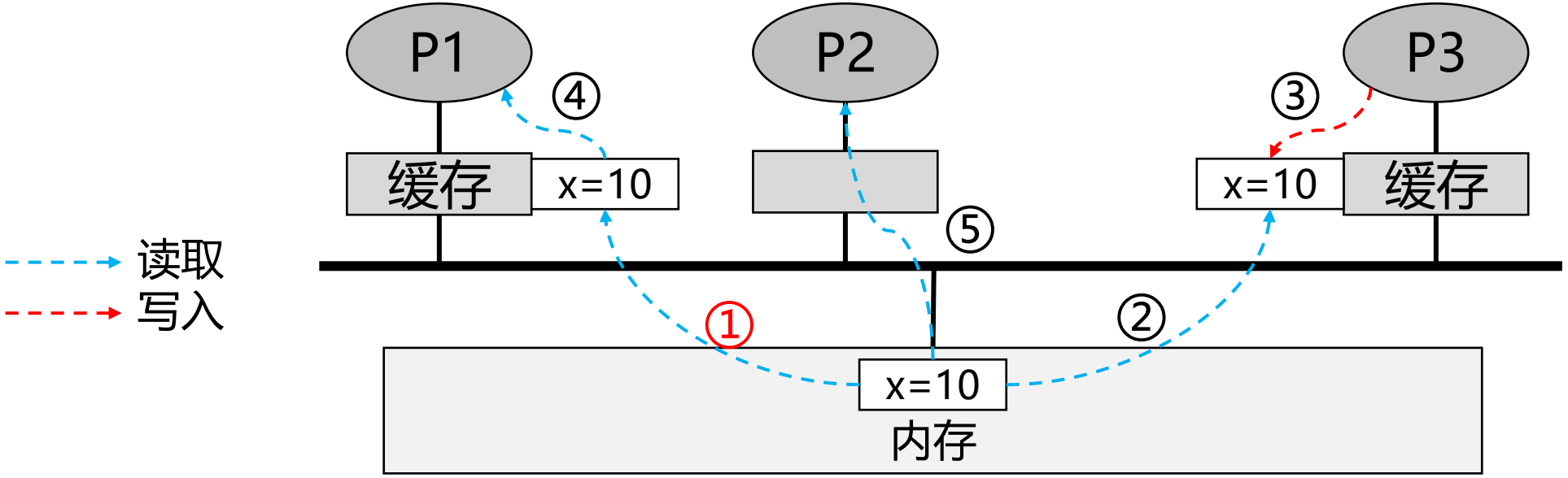
处理器行为	P1状态	P2状态	P3状态	总线行为	数据来源
P1读取x					
P3读取x					
P3写入x					
P1读取x					
P2读取x					



■ Snooping协议

□ P1读取x，标记为S状态

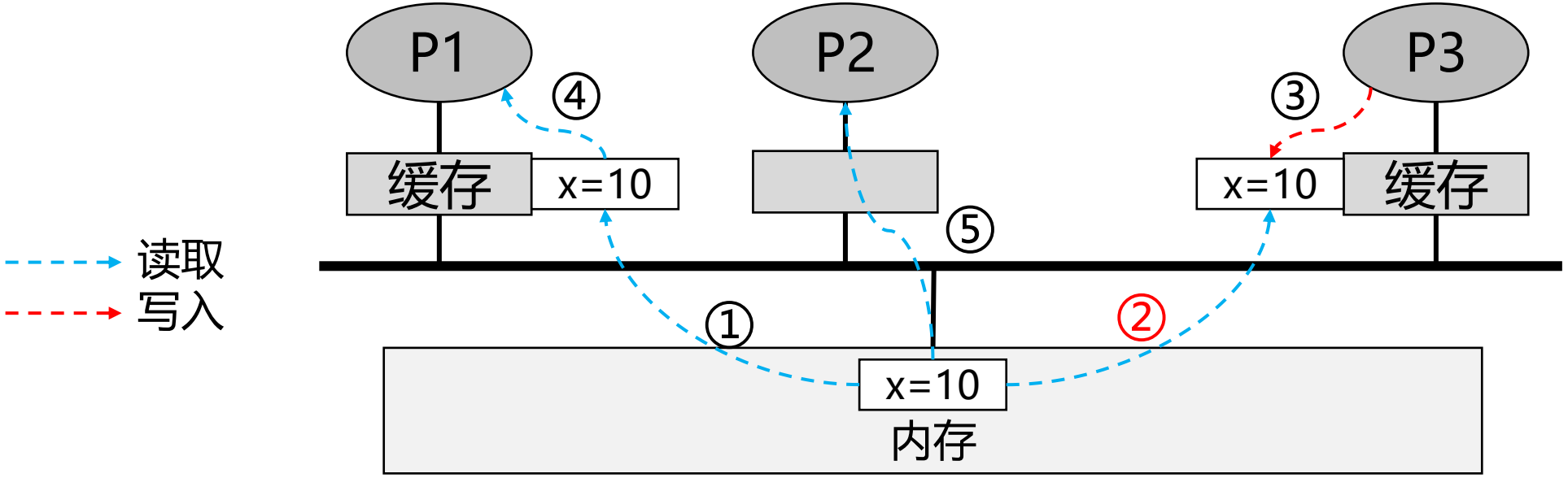
处理器行为	P1状态	P2状态	P3状态	总线行为	数据来源
P1读取x	S	-	-	BR	内存
P3读取x					
P3写入x					
P1读取x					
P2读取x					



■ Snooping协议

□ P3读取x，标记为S状态

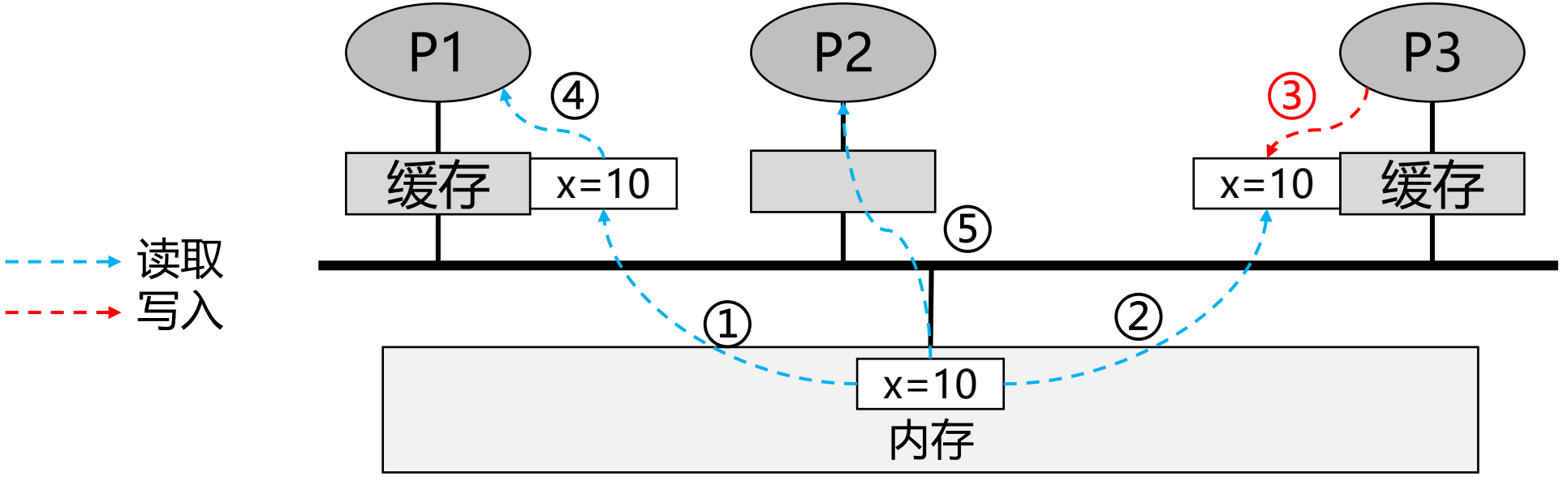
处理器行为	P1状态	P2状态	P3状态	总线行为	数据来源
P1读取x	S	-	-	BR	内存
P3读取x	S	-	S	BR	内存
P3写入x					
P1读取x					
P2读取x					



■ Snooping协议

□ P3写入x，触发PW/BRX变成M状态，P1收到BRX变成I状态

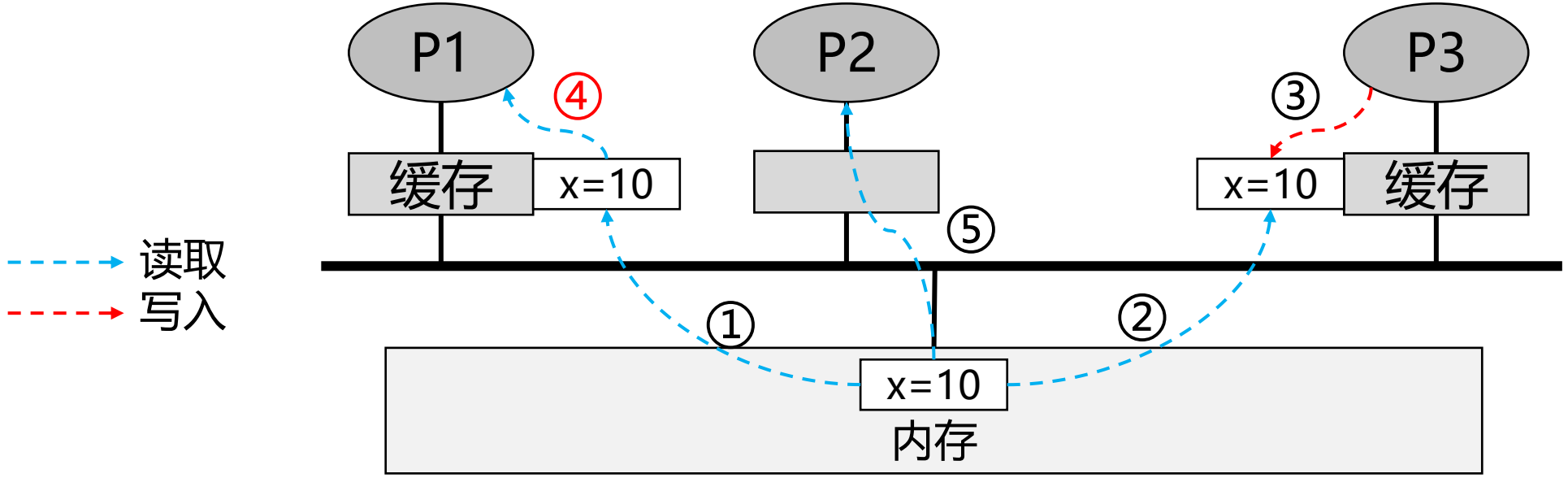
处理器行为	P1状态	P2状态	P3状态	总线行为	数据来源
P1读取x	S	-	-	BR	内存
P3读取x	S	-	S	BR	内存
P3写入x	I	-	M	BRX	内存
P1读取x					
P2读取x					



■ Snooping协议

- P1再次读取x，I状态经由PR/BR变为S，P3的M经由BR/Flush变成S

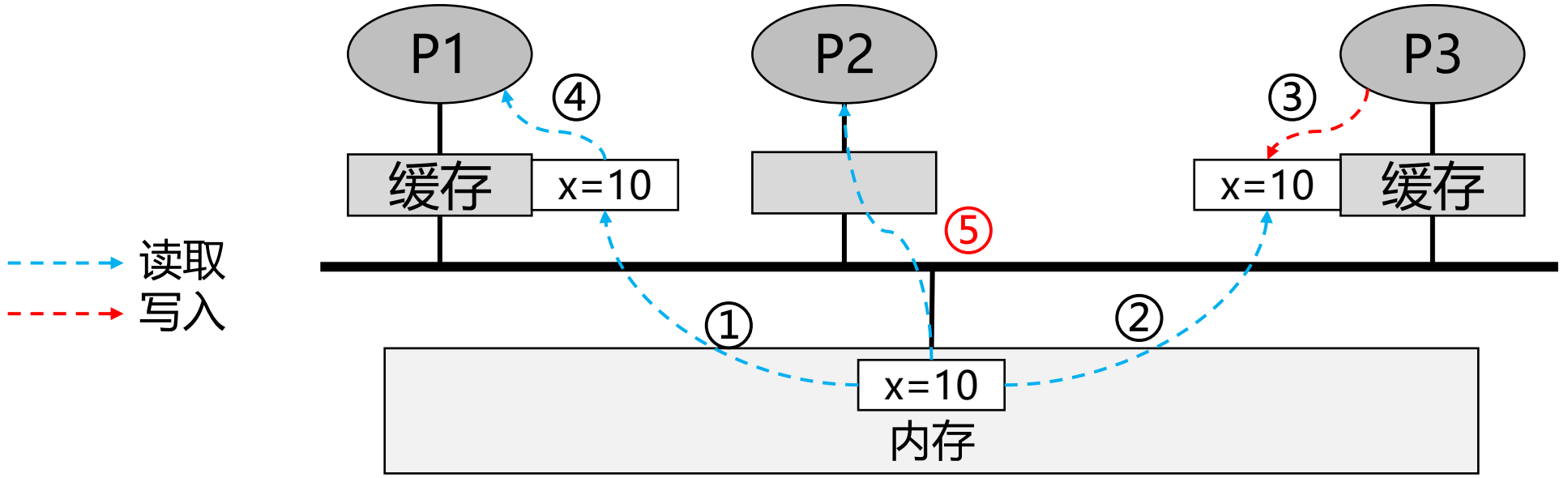
处理器行为	P1状态	P2状态	P3状态	总线行为	数据来源
P1读取x	S	-	-	BR	内存
P3读取x	S	-	S	BR	内存
P3写入x	I	-	M	BRX	内存
P1读取x	S	-	S	BR	P3缓存
P2读取x					



■ Snooping协议

□ P2读取x，P1和P3的S经过BR/--无变化

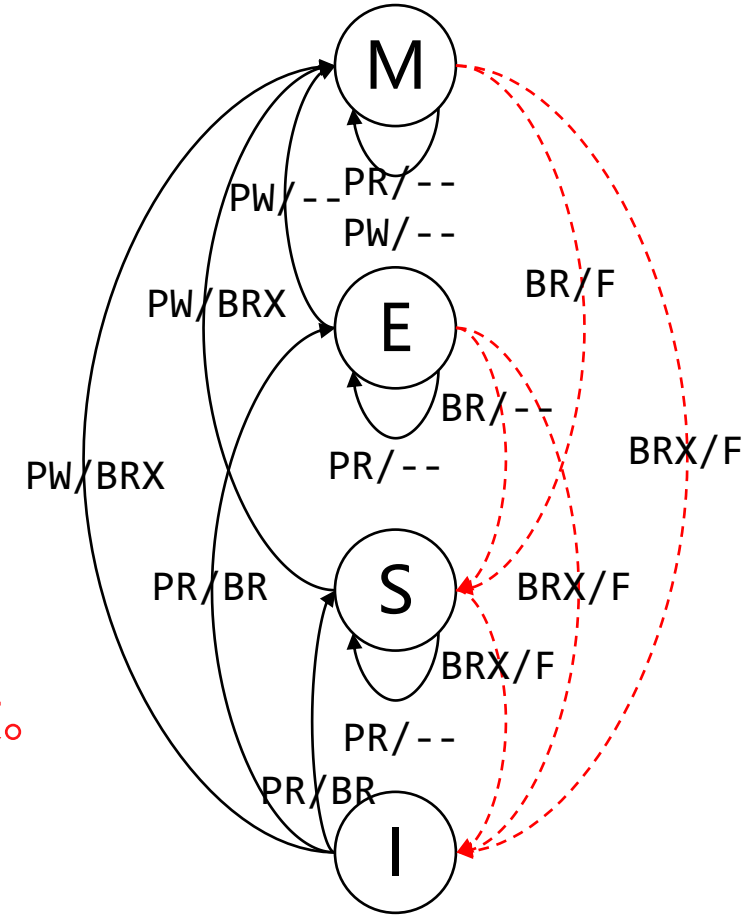
处理器行为	P1状态	P2状态	P3状态	总线行为	数据来源
P1读取x	S	-	-	BR	内存
P3读取x	S	-	S	BR	内存
P3写入x	I	-	M	BRX	内存
P1读取x	S	-	S	BR	P3缓存
P2读取x	S	S	S	BR	内存



■ Snooping协议

- 在MSI基础上增加一个状态E
  - 扩展得到四状态写回失效策略MESI
  - E状态表示独占状态，数据仅在当前处理器缓存中，且尚未修改
- MESI的变种被广泛应用于现代处理器

这个E状态是说，只有我持有数据时，且其他人都不读，如果我要修改我没必要向总线发送BFX。E就表示：独占，但未修改。在E状态时，修改不需通知总线，大大减少了总线通信量，特别是对于那些只有单线程访问的变量。



■ 例题

□ 问题

- 假设MSI的系统有一个M状态的缓存行，收到BR请求，会发生什么？
- 如果是MESI的E状态，连续收到两个BR请求呢？

MSI 如果时M收到BR，那么要么把这个数据写进主存(或通过总线传给P2）；然后把自身状态修改为S，不再独占。

如果是MESI，处于E说明独占且未修改，收到P2的第一个BR时，降级为S；收到P3的BR，不变；此时三个P都是S

PART 04 目录协议

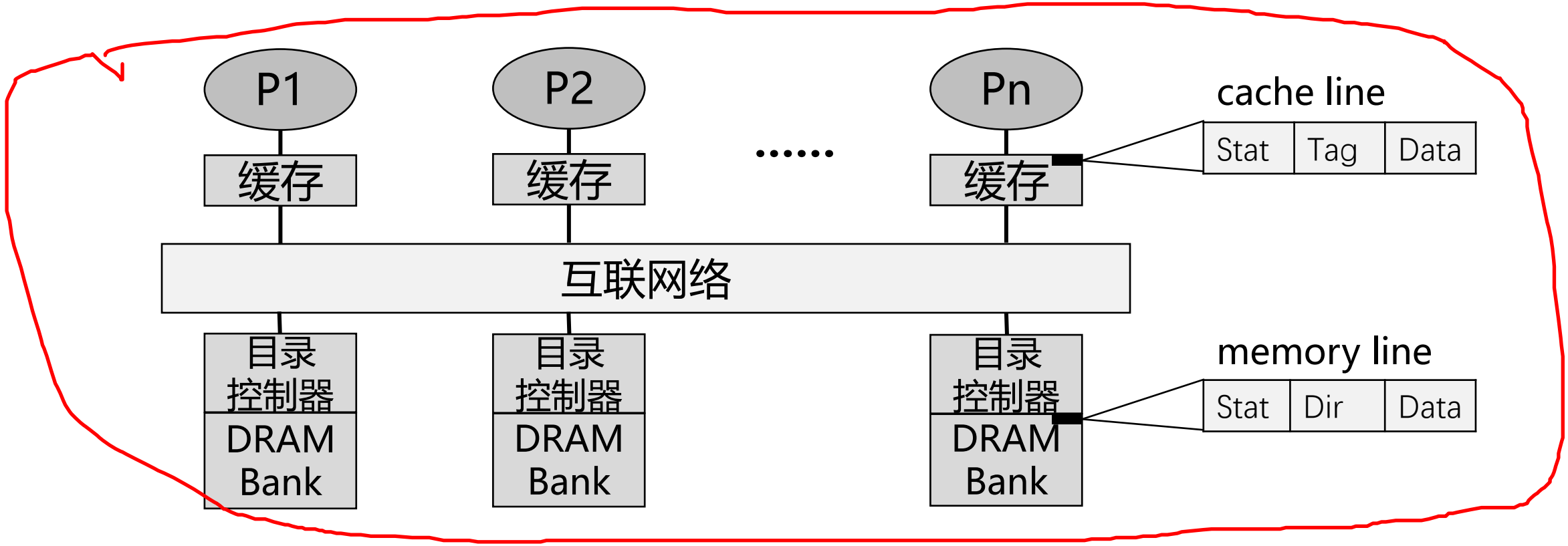
■ 背景

- Snooping协议
  - 瓶颈在总线上，因此Snooping协议的多处理器系统适合少量的处理器
  - 实现简单，且延迟较低
- 目录协议
  - 单独记录缓存行的共享情况，点对点通信，可扩展性强
  - 实现复杂，延迟高
    - 访问数据需要发送一个数据读取请求到持有数据的处理器
    - 持有数据的处理器收到请求后，响应一个回复，回复携带所需数据

directory协议

directory协议的结构

- 缓存的缓存行(cache line)除Tag外还有用于实现一致性功能的Stat状态码
- 主存中的缓存行(memory line)增加Dir字段，其中每个bit对应一个处理器

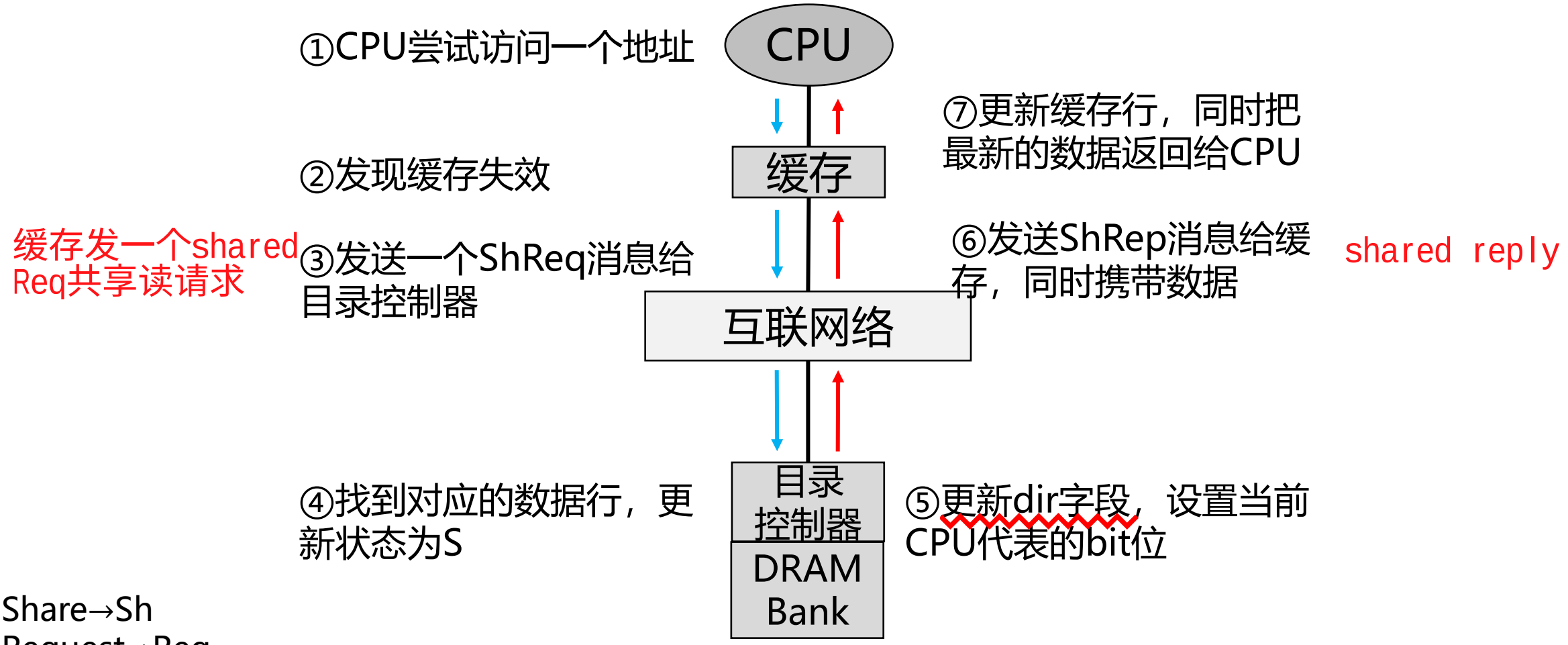


directory协议

- 缓存中的缓存行有4个状态
  - 无效(invalid)：所需数据不存在于当前缓存
  - 共享(share)：所需数据存在于当前缓存，同时其他缓存也使用到，内存中的数据是最新的
  - 修改(modified)：所需数据被当前缓存独占，而且进行修改，内存中的数据不是最新的
  - 传输(transient)：访问数据的请求已经发送，等待数据响应。这是一个瞬态，一旦数据传输完毕立即转为其他状态

■ directory协议

□ 读取时发生缓存失效，directory协议的行为



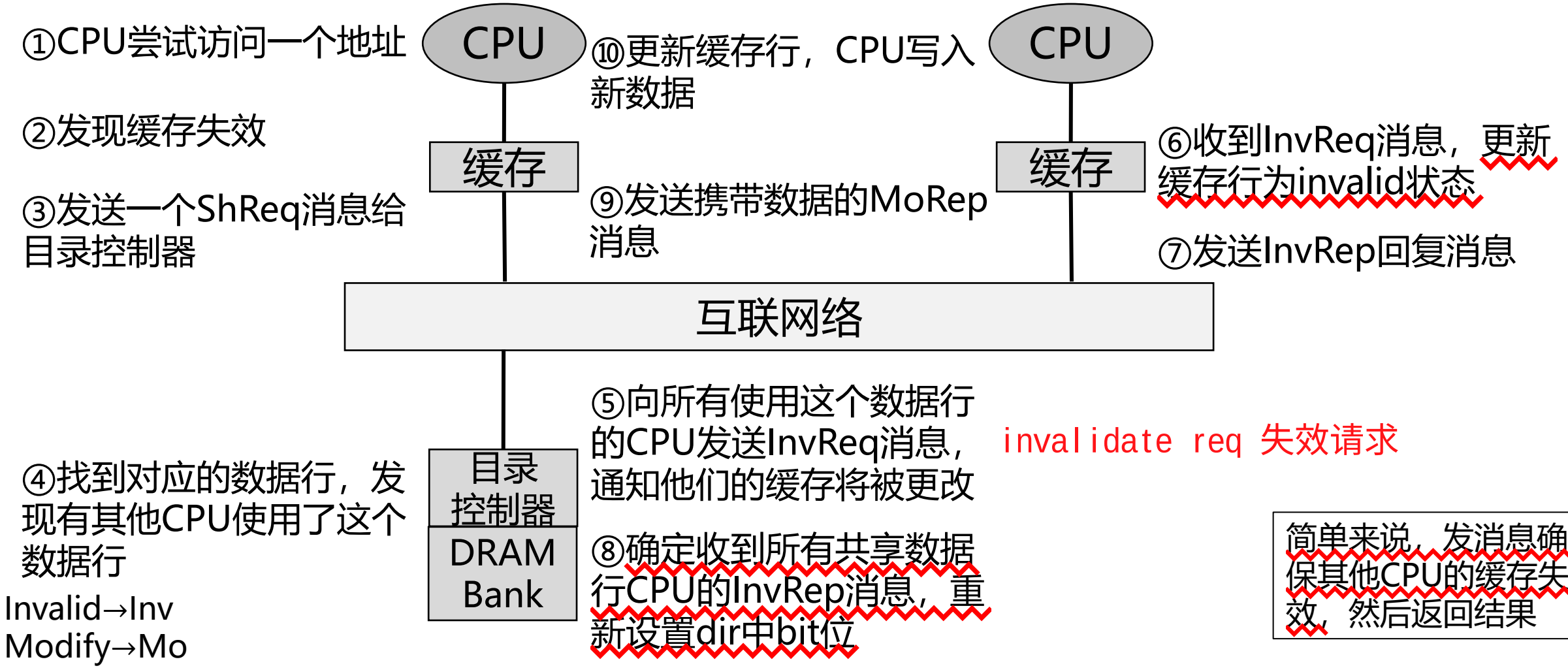
缓存发一个shared Req共享读请求

shared reply

Share→Sh
Request→Req
Reply→Rep

directory协议

写入时发生缓存失效，directory协议的行为



directory协议

directory协议特点

- 类似Snoopy协议，在目录控制器中缓存行(memory line)有三个状态
  - 共享(shared)，超过一个处理器有数据行的拷贝，内存中的数据是最新的
  - 未使用(uncached)，当前数据行不存在于任意处理器的缓存中
  - 独占(exclusive)，只有一个处理器拥有数据，且内存中的数据不是最新的
- 为追踪数据行共享状态，通常用bit向量存储共享状态，置1表示这个处理器有当前数据行的拷贝
- 简化协议行为
  - 需要写处于共享状态的数据行时，调用写缓存失效的过程
  - 处理器会被阻塞，直到上述过程执行完毕
  - 消息按顺序发送并依次处理

directory协议

directory协议特点(续)

- 不需要总线，不依赖于广播行为来交换数据
  - 所有数据请求通过显式消息响应
- 在一次一致性响应过程中，不同处理器扮演三种不同角色
  - 本地节点(local node)，发出最原始数据访问请求的处理器
  - 主节点(home node)，实际拥有数据的处理器
  - 远端节点(remote node)，拥有数据拷贝的其他处理器

directory协议

directory协议消息

- 其中P表示处理器编号，A表示地址

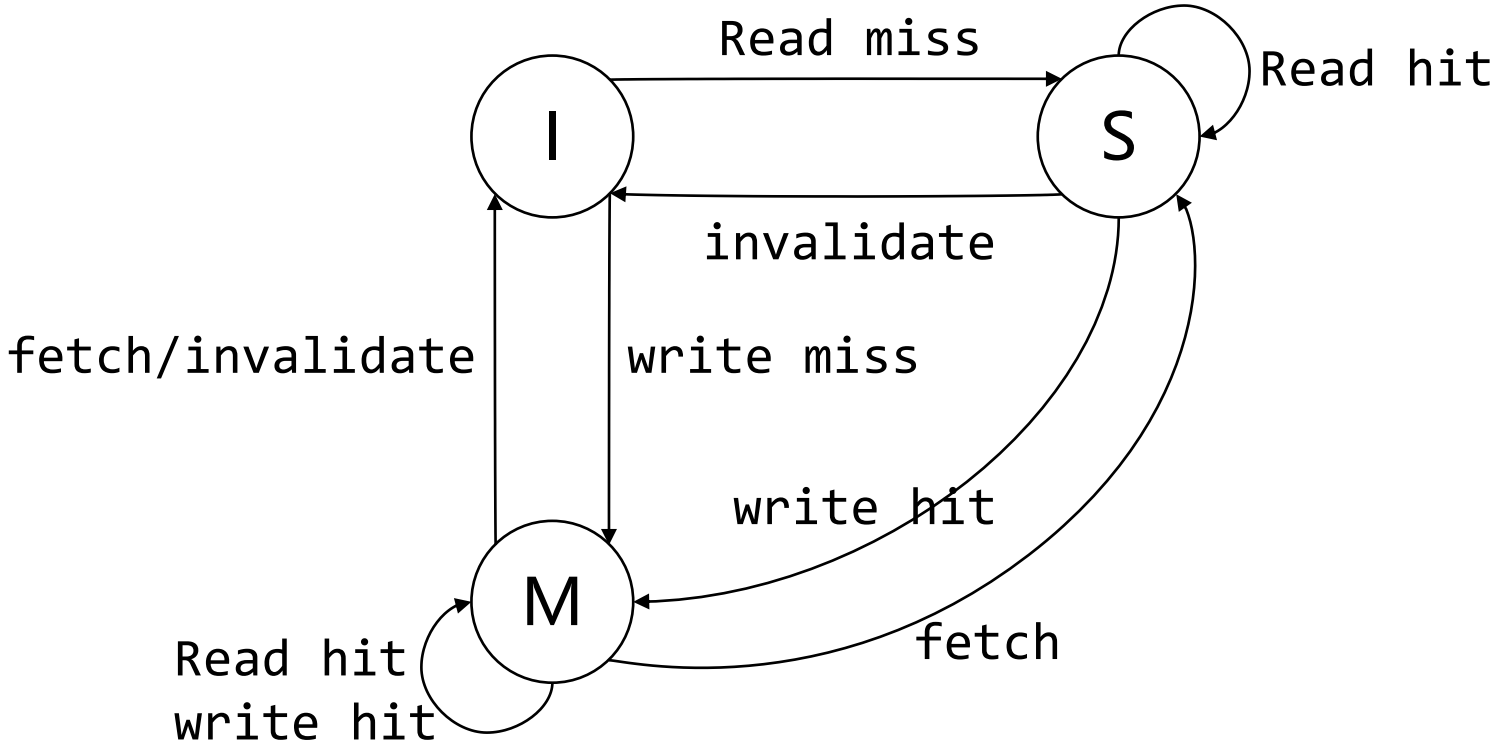
类型	发送源	接收端	包含数据	说明
read miss	本地缓存	目录控制器	P、A	P需要读取地址A处数据。目录协议会记录数据行共享状态，然后发送数据
write miss	本地缓存	目录控制器	P、A	P需要写入地址A处数据。升级P为独占状态，令其他缓存行失效
invalidate	目录控制器	远端缓存	A	令远端处理器缓存上地址A的缓存行失效
fetch	目录控制器	远端缓存	A	远端处理器把包含地址A的缓存行发送到目录控制器
fetch/invalidate	目录控制器	远端缓存	A	结合fetch和invalidate的联合消息
data value reply	目录控制器	本地缓存	Data	响应read miss，返回数据
data write back	远端缓存	目录控制器	A、Data	远端处理器响应invalidate消息，同时回写最新数据

■ 状态转移图

- 目录协议的状态转移图
  - 状态和状态间的变化类似Snoopy协议
  - 状态的变化由read miss、write miss、invalidates和data fetch引起
  - 然后生成对应的消息发送给目录控制器

■ 状态转移图

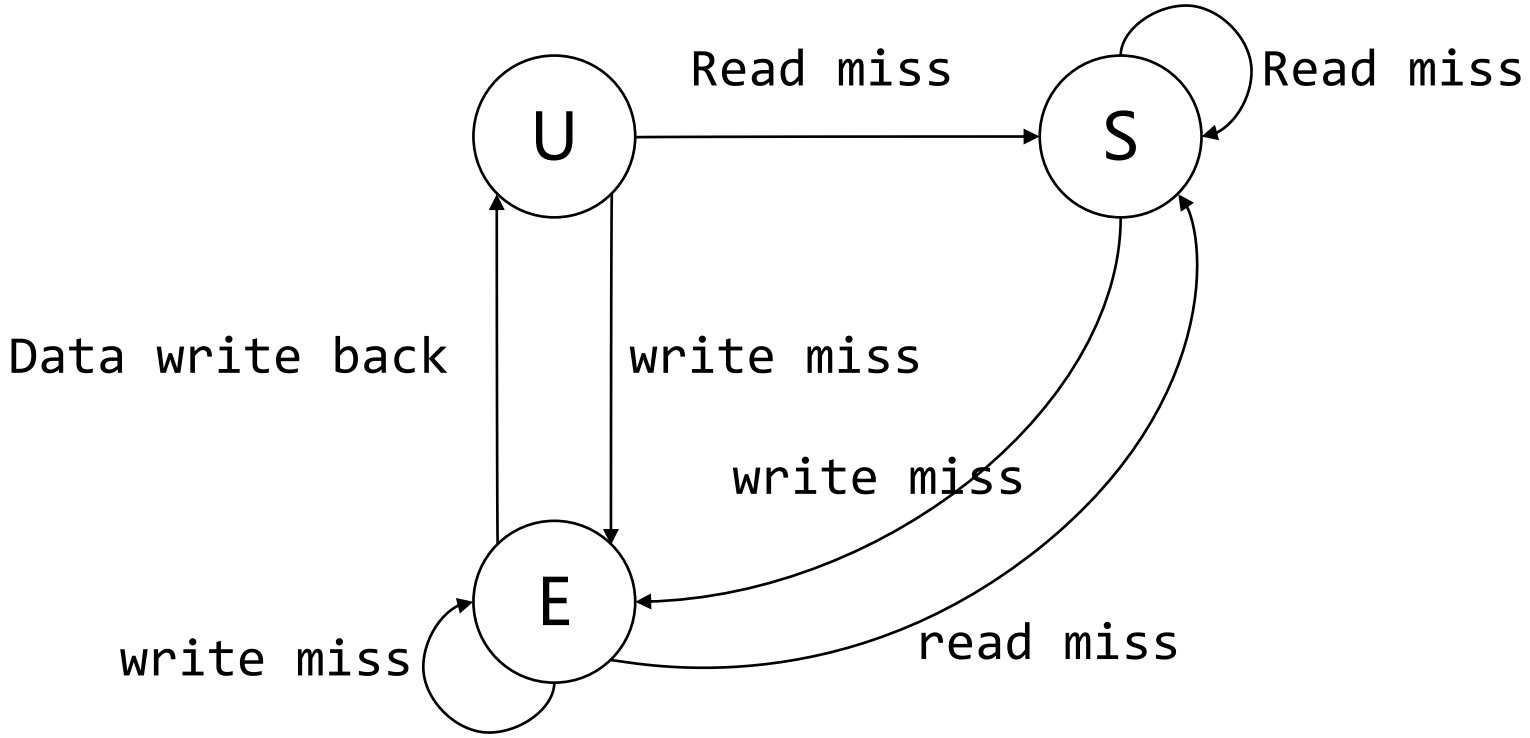
- 处理器上缓存行的状态机
  - 收到来自**处理器**和**目录控制器**的消息时状态变化
  - 有可能缓存命中，状态机同时存在Read miss和Read hit两种消息，write同理



Invalid→I
Shared→S
Modified→M

■ 状态转移图

- 目录控制器的状态机
  - 大体相同，相比缓存行的状态机缺少从S到U状态的转化
  - 只有收到来自**处理器**的消息，才产生状态变化



Uncached→U
Shared→S
Exclusived→E

■ directory协议示例

□ 消息→动作

- 目录控制器收到消息会产生两个动作
  - ➢ 更新目录信息，例如状态、数据等
  - ➢ 发送消息同步信息
- 例如当前主存中缓存行(memory line)在Uncached状态，它只能接受两种消息
  - ➢ Read miss消息：更新dir字段记录新分享的处理器编号；更新当前缓存行为Shared状态；发送数据响应
  - ➢ write miss消息：更新dir字段记录独占的处理器编号；更新状态为exclusived；发送数据响应
- 相似的还有shared状态
  - ➢ 需要通知其他处理器当前缓存行发生一些变动

■ directory协议示例

□ 消息→动作

- 主存中缓存行为exclusive状态，可以接收三种消息
  - ➢ read miss消息：状态变更为shared；修改dir字段记录新分享的处理器编号；主节点发送最新的数据到目录管理器；目录管理器发送数据到本地节点
  - ➢ write miss消息：状态不变；更新dir字段定义本地节点为新的主节点；原主节点发送最新数据到目录管理器；发送数据到新的主节点
  - ➢ data write back消息：状态变更为uncached；主节点发送最新数据到目录管理器
    - 主节点的缓存淘汰策略，把一些不再需要的缓存行写回内存时会发送此类消息

本地节点(local node)，发出最原始数据访问请求的处理器。
主节点(home node)，实际拥有数据的处理器。
远端节点(remote node)，拥有数据拷贝的其他处理器。

directory协议示例

假设地址A1和A2映射到同一个缓存行

	P1缓存行			P2缓存行			互联网络上的消息				目录控制器			内存
	stat	addr	valu	stat	addr	valu	actio	proc	addr	valu	addr	stat	proc	
P1向A1写入10														
P1读取A1														
P2向A2写入20														

directory协议示例

假设地址A1和A2映射到同一个缓存行

	P1缓存行			P2缓存行			互联网络上的消息				目录控制器			内存
	stat	addr	valu	stat	addr	valu	actio	proc	addr	valu	addr	stat	proc	
P1向A1写入10							WrMi	P1	A1					[0,0]
	M	A1	10				DaRp	P1	A1	0	A1	E	{P1}	[0,0]
P2读取A1														
P2向A2写入20														

write miss→WrMi，data reply→DaRp，read miss→ReMi，invalidate→inv

directory协议示例

假设地址A1和A2映射到同一个缓存行

	P1缓存行			P2缓存行			互联网络上的消息				目录控制器			内存
	stat	addr	valu	stat	addr	valu	actio	proc	addr	valu	addr	stat	proc	
P1向A1写入10							WrMi	P1	A1					[0,0]
	M	A1	10				DaRp	P1	A1	0	A1	E	{P1}	[0,0]
P2读取A1							ReMi	P2	A1					[0,0]
	S	A1	10				Fetch	P1	A1	10				[10,0]
				S	A1	10	DaRp	P2	A1	10	A1	S	{P1,P2}	[10,0]
P2向A2写入20														

数据写回内存

write miss→WrMi，data reply→DaRp，read miss→ReMi，invalidate→inv

directory协议示例

假设地址A1和A2映射到同一个缓存行

	P1缓存行			P2缓存行			互联网络上的消息				目录控制器			内存
	stat	addr	valu	stat	addr	valu	actio	proc	addr	valu	addr	stat	proc	
P1向A1写入10							WrMi	P1	A1					[0,0]
	M	A1	10				DaRp	P1	A1	0	A1	E	{P1}	[0,0]
P2读取A1							ReMi	P2	A1					[0,0]
	S	A1	10				Fetch	P1	A1	10				[10,0]
				S	A1	10	DaRp	P2	A1	10	A1	S	{P1,P2}	[10,0]
P2向A2写入20							WrMi	P2	A2					[10,0]
	I						Inv	P1	A1					[10,0]
				M	A2	20	DaRp	P2	A2	0	A2	E	{P2}	[10,0]

write miss→WrMi，data reply→DaRp，read miss→ReMi，invalidate→inv

directory协议优化

具体实现directory协议时

- 需要保证操作的原子化
  - 在示例中可以发现，处理一条Read miss消息，可能需要发送多条消息以同步数据
  - 需要保证处理一条Read miss时的所有操作一起是原子的
- 避免死锁
  - 假定目录控制器正在在处理read miss消息，其他处理器产生一条write miss消息把互联网络占满
  - 此时目录控制器无法发送消息同步数据，因此一直卡在read miss处理
  - 其他处理器因互联网络满无法发送消息
- 在exclusive状态下的缓存行收到来自其他处理器的read miss消息
  - 可以直接点对点发送数据响应，避免慢速的内存写入过程

感谢！