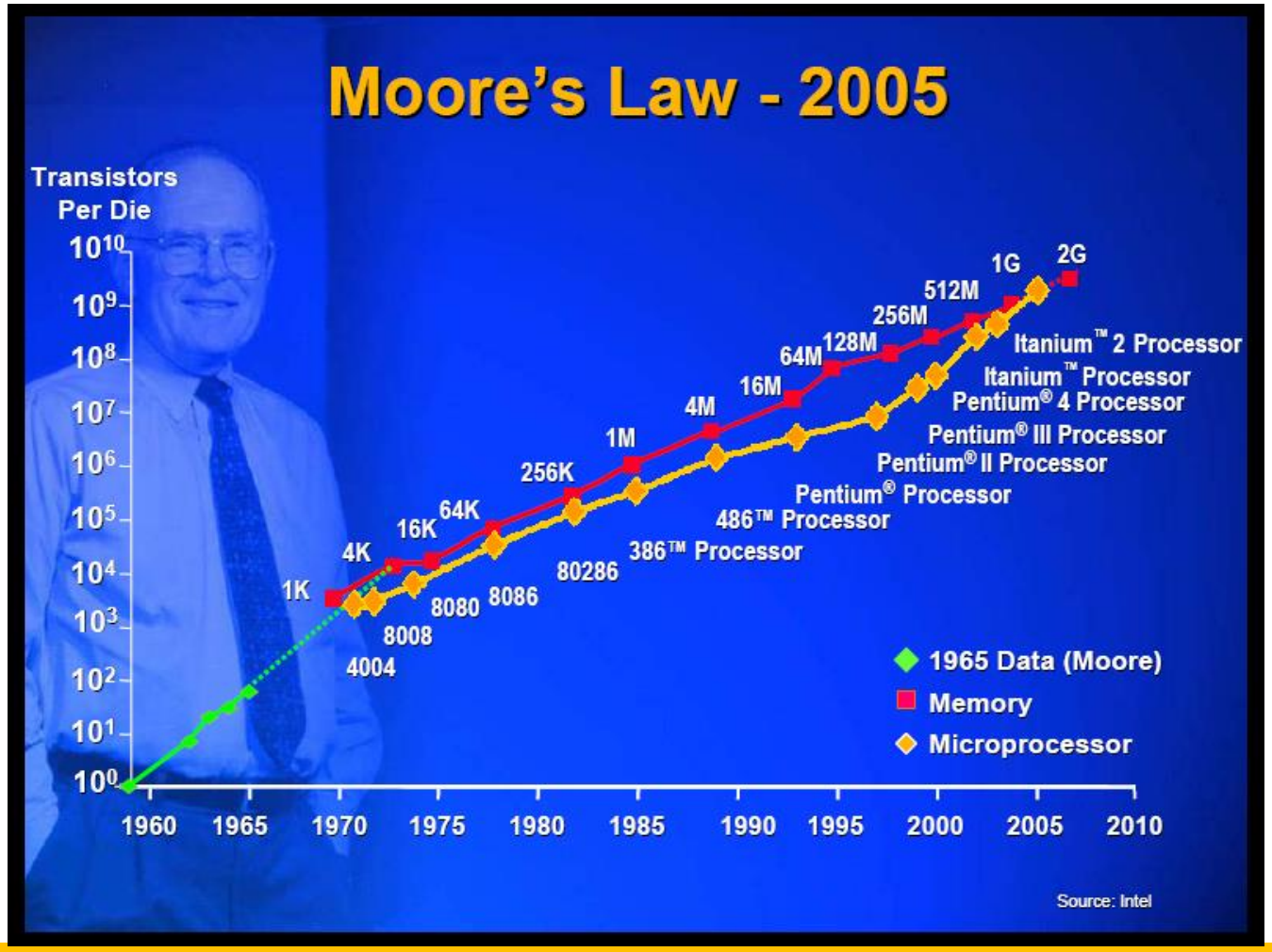
学习内容

- 7.1 多处理机概念
- 7.2 多处理机结构
- 7.3 多核处理器
- 7.4 多处理机的多Cache一致性
- 7.5 多处理机的机间互连形式
- 7.6 程序并行性
- 7.6 多处理机性能
- 7.7 多处理机操作系统

7.3 多核处理器

- Multi-core Processor
- 片上多处理机 (CMP，Chip Multi-Processors)
- 片上多处理机系统 (MPSoC，Multiprocessor System-on-Chip）
- 是多处理机的一种特殊形式：SMP on a single chip
- 1996年，美国斯坦福大学首次提出了片上多处理器(CMP)思想
- 2000年，IBM于发布了世界上第一个双核处理器POWER4
- 2005年，Intel和AMD多核处理器开始大规模应用

微处理器发展



Transistor capacity doubles every 18 months

微处理器发展

- 问题：晶体管数量 ≠ 能力
- 如何利用晶体管资源？
  - 更高的时钟频率
  - 更好的核
    - 超流水，超标量，…
    - 更好的向量处理（SIMD）
    - 问题：存储器墙 = 存储器速度不能满足CPU速度要求
  - 更大的Cache
    - 改善访存速度

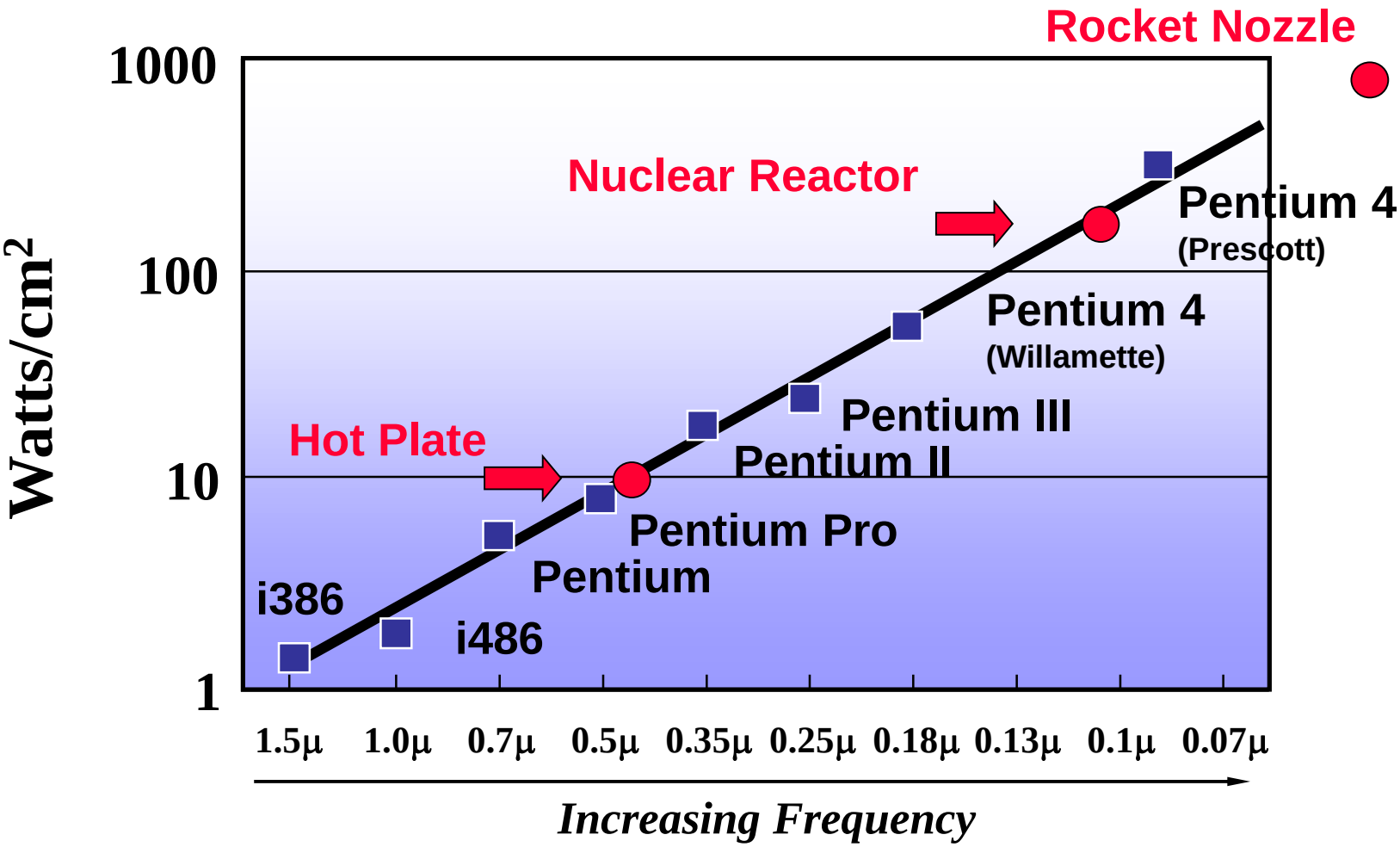
微处理器发展

- **更高的时钟频率**
  - 增加了功耗
    - 功耗与频率和电压的平方(U^2)成正比
    - 更高的频率需要更高的电压
    - 问题：漏电流导致的能量损失
  - 增加了散热和制冷需求
  - 限制了芯片面积 (光速)

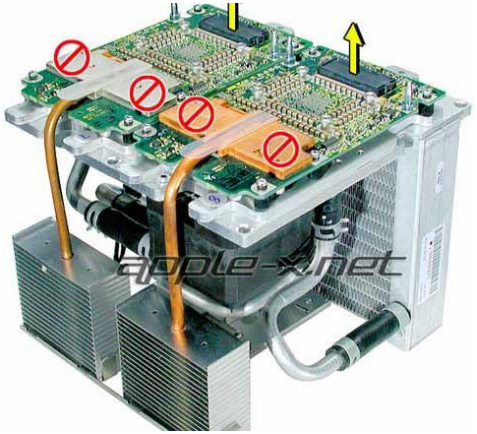
100 GHz limits chip to 3 mm (speed of light 300mm/ns)

微处理器发展

耗散功率 (Dissipated Power) ~ CV^2f



Managing the Heat Load



Liquid cooling system in Apple G5s

Heat sinks in 6XX series Pentium 4s

微处理器发展

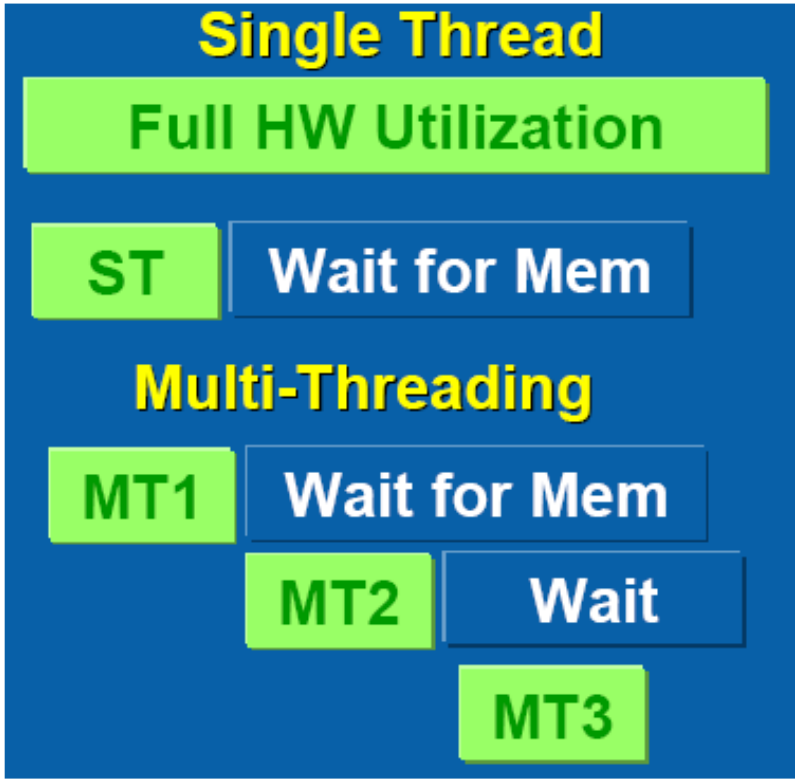
- 更多/更高的并行性
  - 增加位宽度
  - 指令级并行 (ILP)
    - 开发指令之间的并行性
    - 受限于数据、指令、控制相关
    - 通过预测可以改善
  - 线程级并行 (TLP)
    - 硬件线程 (例如 SMT：超线程)
    - 多核

进程与线程

- **进程定义：资源分配单位**
  - 一个独立的进程空间，可装入进程映像；
  - 进程关联的执行文件；
  - 进程所用系统其它资源(如设备、文件等）；
  - 一个或多个线程。 进程在创建时一般同时创建好第一个线程，其它线程按需要由用户程序请求创建。
- **线程定义：CPU分配单位（执行单位）**

TLP

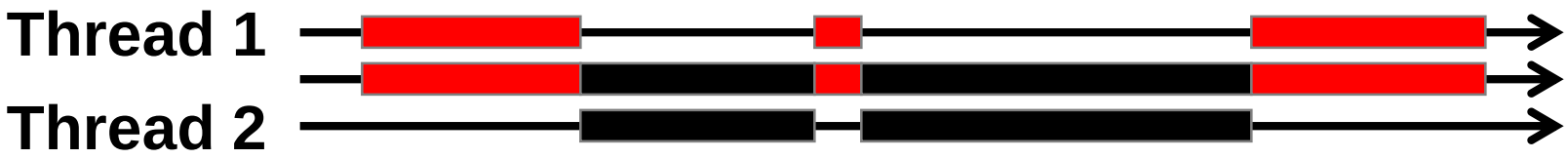
- **TLP（线程级并行）相关技术**
  - 同时多线程 (SMT = Simultaneous Multi-threading)
    - 例如: Intel 超线程
  - 芯片多处理(CMP = Chip multiprocessing) → Multi-Core Processor



Understanding SMT and CMP

Make clear Concurrency vs. Parallelism

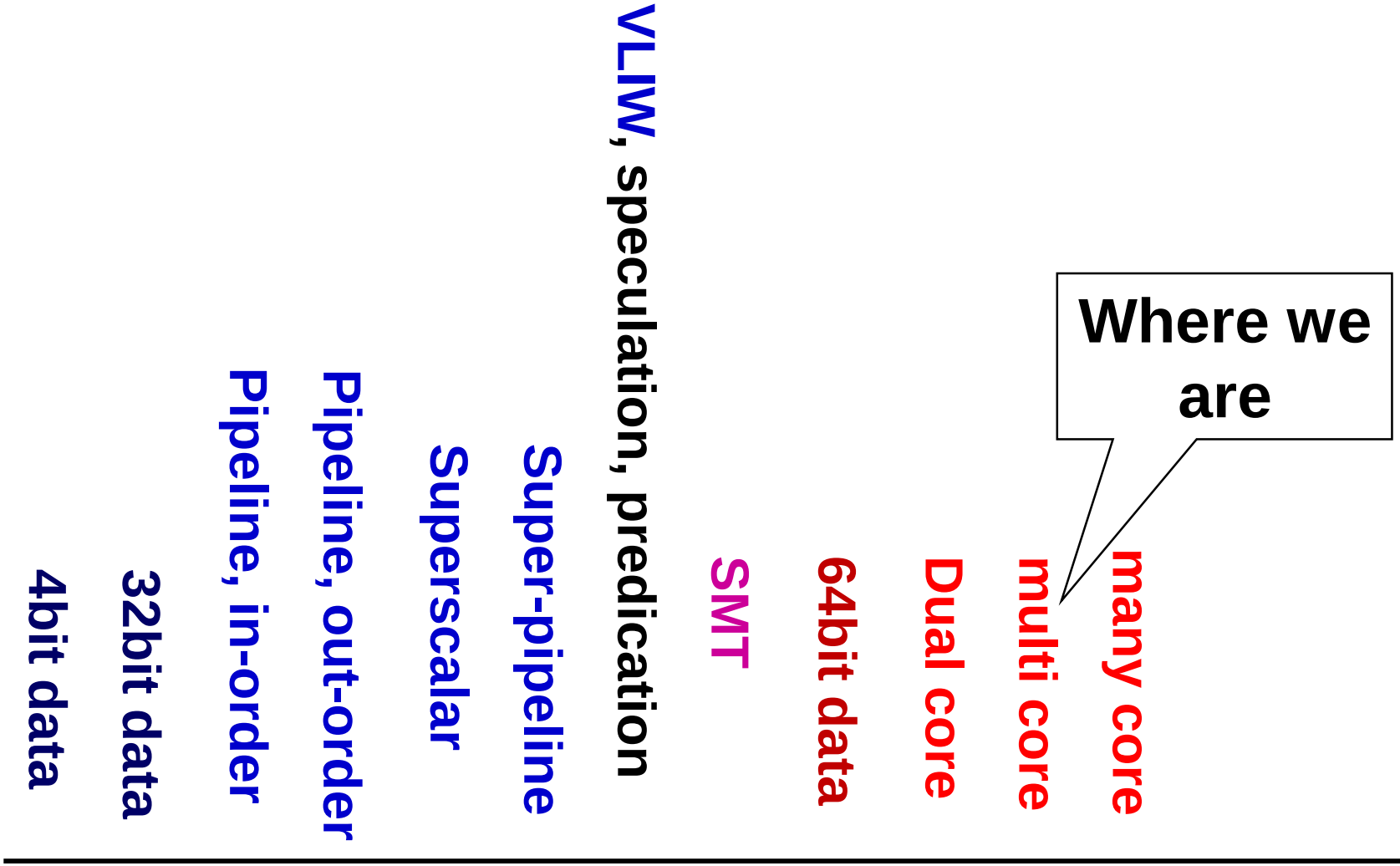
- **Concurrency:** two or more threads are in progress at the same time:



- **Parallelism:** two or more threads are executing at the same time

Multiple cores needed

A brief history of micro-architecture evolution



7.3.1 多核处理器定义与特点

- **多核处理器：**
  - 一枚处理器中集成了两个或多个独立处理单元（称为核）的处理器；
  - 每个核由一个独立处理器的所有组件所组成，可以独立运行程序指令（多指令），可以访问存储器的不同部分（多数据）。
- 只有两个核的处理器，称为双核处理器（dual-core processor）。

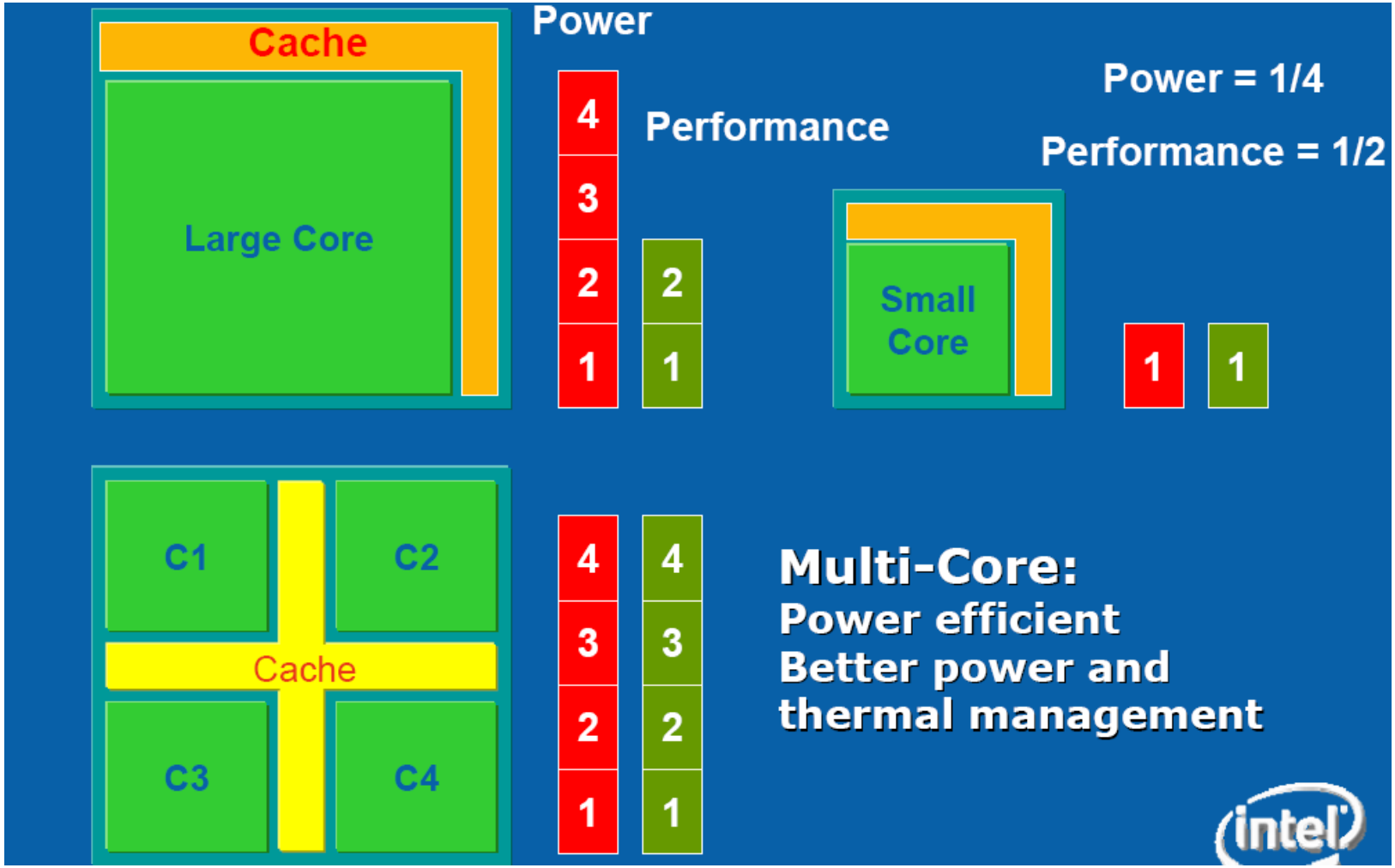
7.3.1 多核处理器定义与特点

- **众核处理器：**
  - 当一枚处理器集成的核的数量达到十几、几十、或几百、几千时，多核变为**众核**。
  - 相比多核：
    - 核的数量多
    - 核经过专门设计

7.3.1 多核处理器定义与结构

- **优点：**
  - 提高吞吐率和并行程序的速度
  - 可以实现核的紧耦合
    - 更好地实现核之间的通信（相比 SMP）
    - 共享Cache
  - 降低功耗
    - 降低时钟频率
    - 可以挂起空闲的核
- **缺点：**
  - 仅能提高并行程序的速度
  - 扩大了与存储器速度的差距

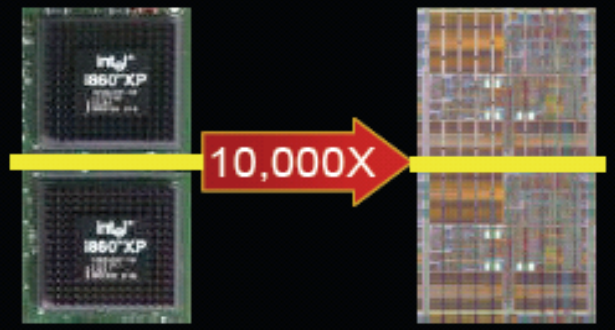
7.3.1 多核处理器定义与结构



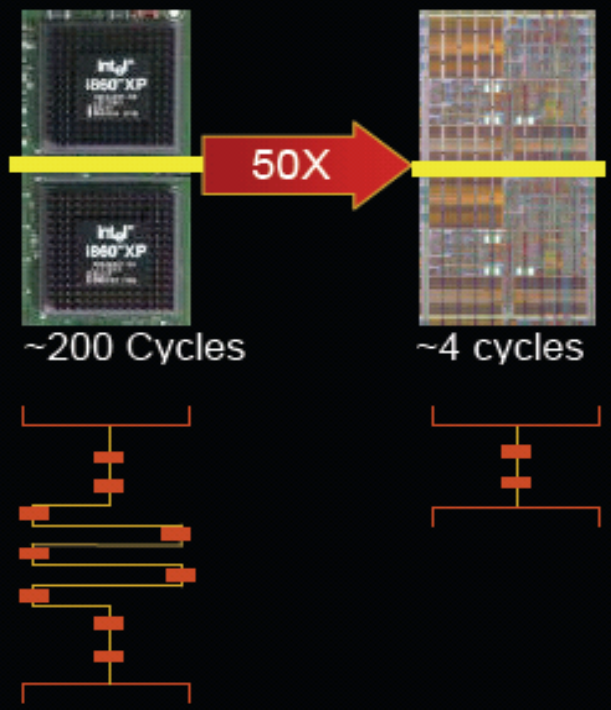
New Target for Micro-architecture – high performance/power

Changes from Multiprocessors to Integrated Multicores

- Communication Bandwidth



- Communication Latency



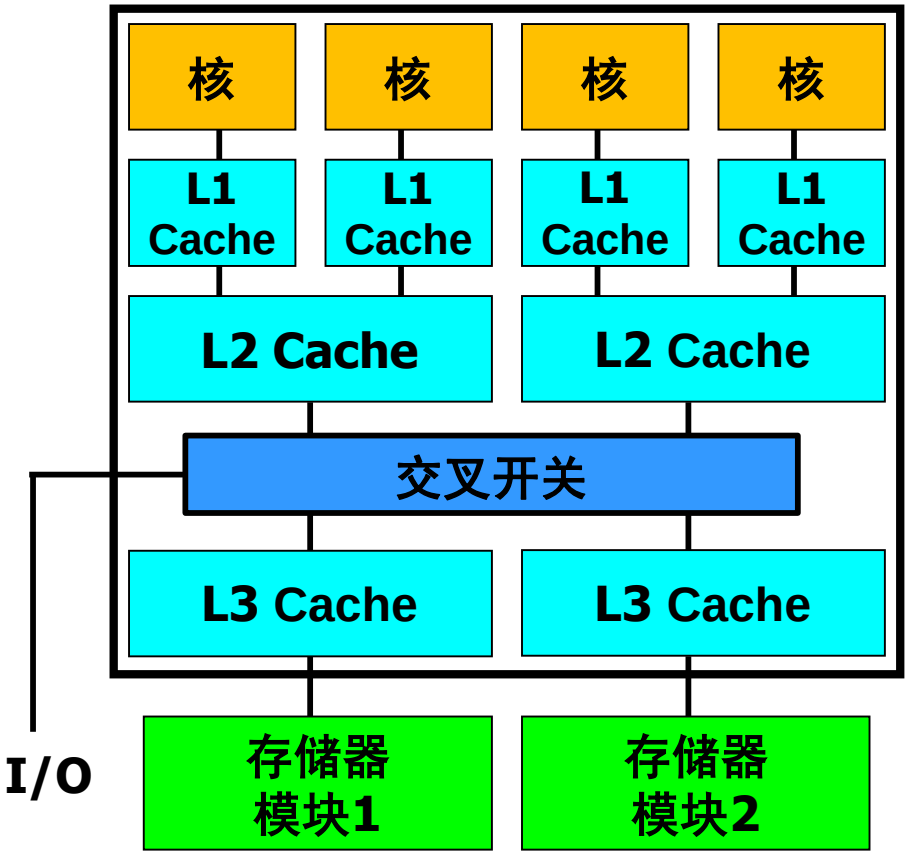
- Where is the Impact?
  - Data Locality
- Where is the Impact?
  - Granularity of Parallelism

7.3.1 多核处理器定义与结构

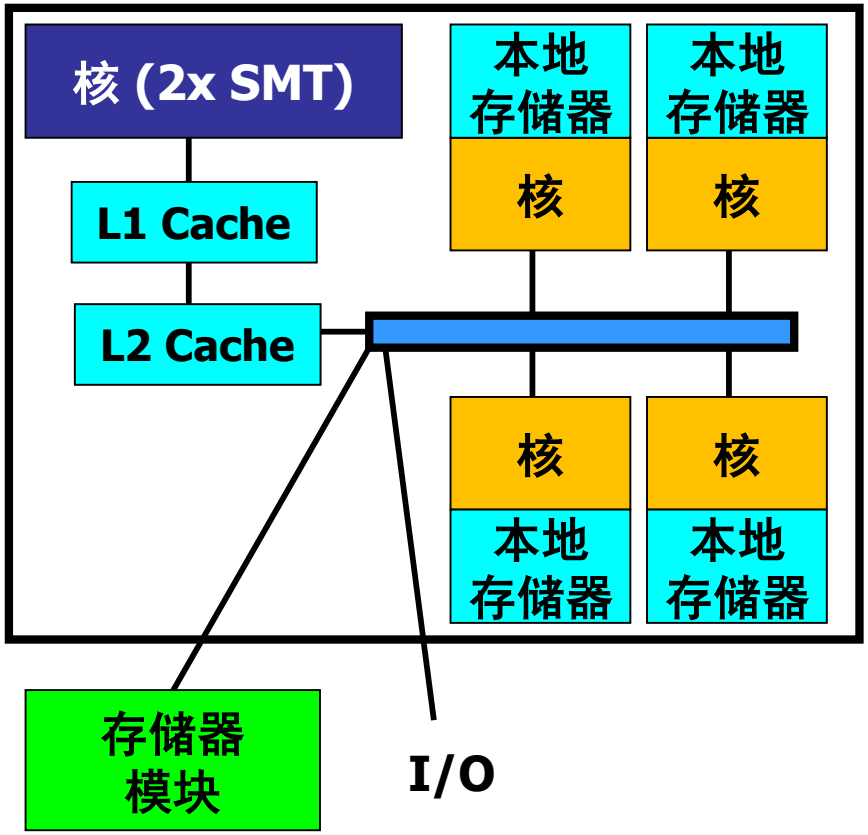
多核处理器结构设计主要考虑以下因素(1/2)：

- (1) 同构还是异构
  - 同构 CMP：集成多个相同的处理器核，同一个任务可以分配给任意一个核处理，简化了任务分配。
  - 异构 CMP：包含了不同结构的处理器核，用不同类型的处理器核处理不同的任务，是异构体系结构处理器的优势所在。
- (2) 核的数量
  - 双核，四核，八核，…

7.3.1 多核处理器定义与结构



带共享Cache和交叉开关的同构多核处理器结构



带Cache、本地存储器和环形总线的异构多核处理器结构

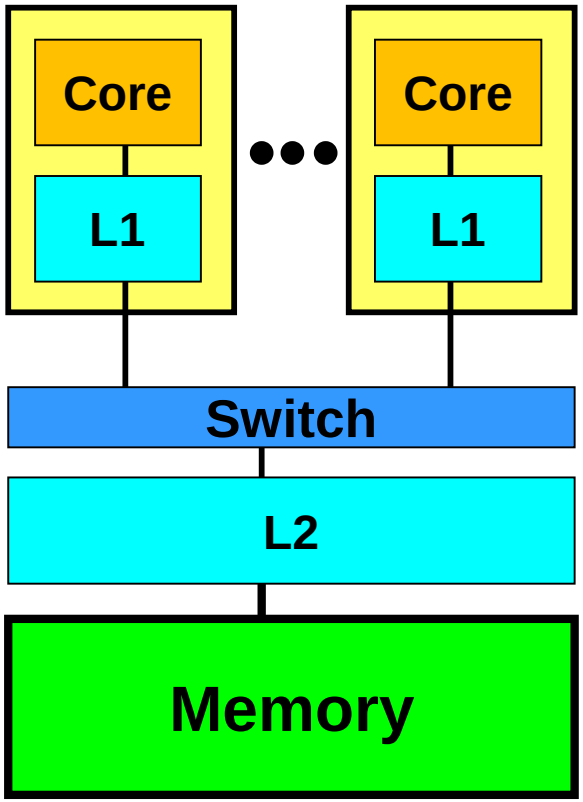
目前，商用处理器大多以同构多核处理器为主

7.3.1 多核处理器定义与结构

多核处理器结构设计主要考虑以下因素(2/2)：

- (3) Cache的设置与访问
  - 分布式存储器结构
  - 共享存储器结构，共享访问的Cache多大
  - Cache层次
- (4) 核间通信技术
  - 通过基于总线共享的Cache
  - 通过片上互连网络

7.3.1 多核处理器定义与结构

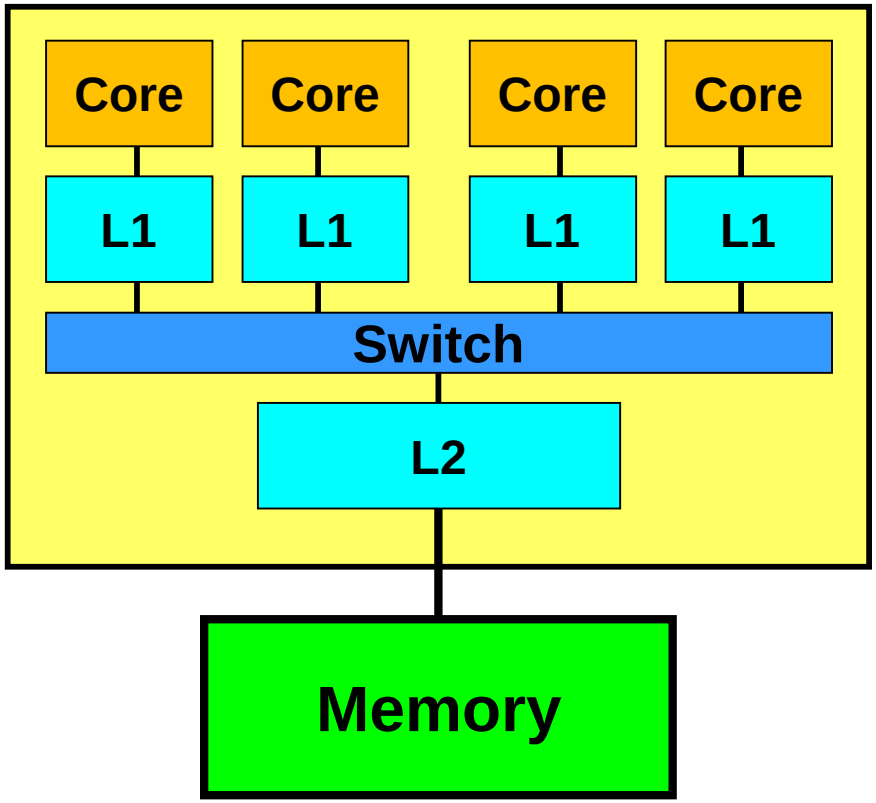


传统设计

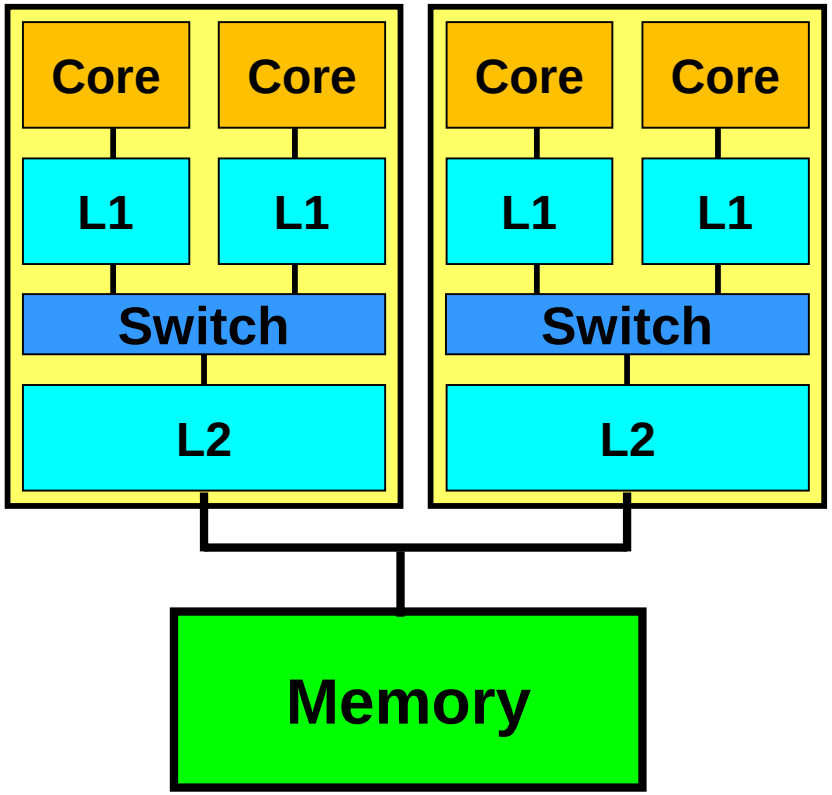
多个单核

L1 Cache私有，共享片外 L2 Cache

7.3.1 多核处理器定义与结构

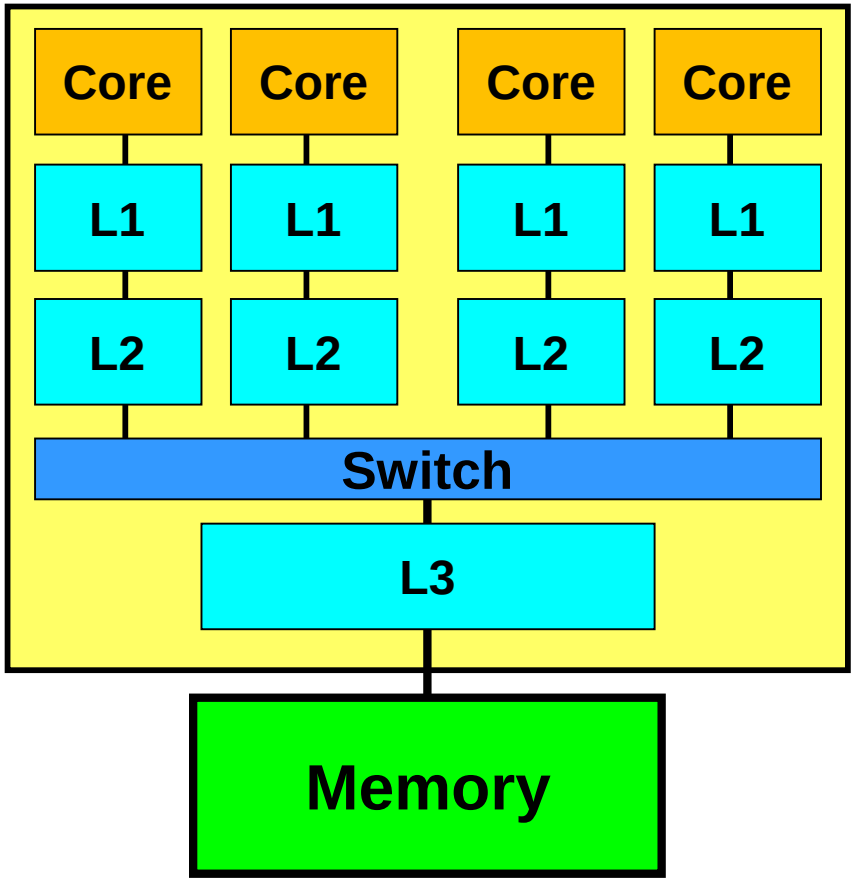


多核
L1 Cache私有
共享片内L2 Cache

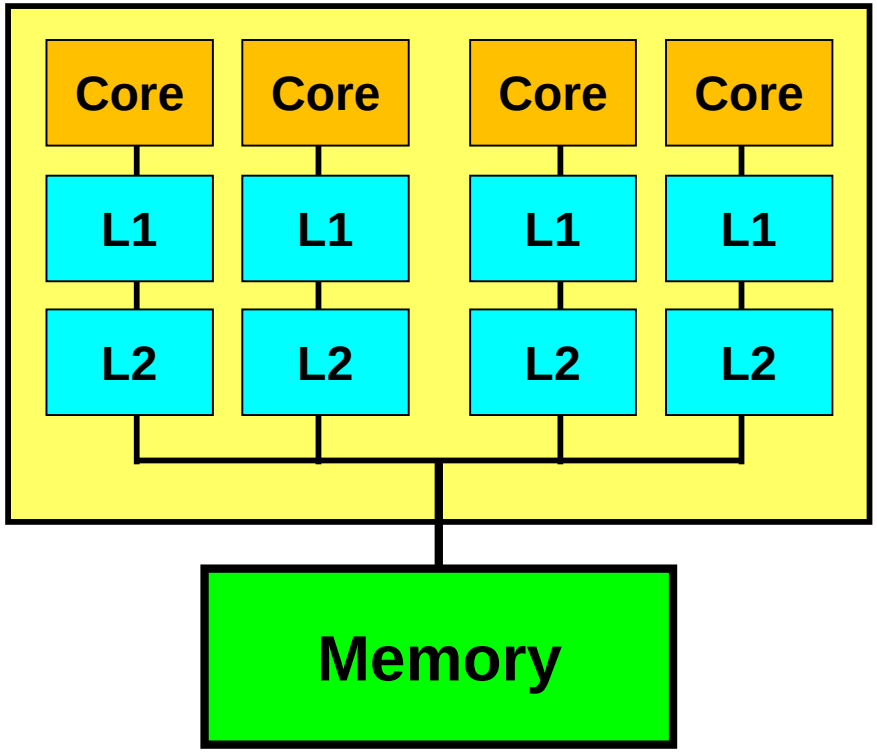


多核
L1 Cache私有
共享片内L2 Cache

7.3.1 多核处理器定义与结构



多核
L1和L2 Cache私有
共享片内L3 Cache



多核
L1和L2 Cache私有
分布式存储器

7.3.1 多核处理器定义与结构

- 共享Cache的优点：
  - 在共享Cache级不需要一致性协议
  - 通信延迟小
  - 工作集可以交叉
    - 可由某个核预取数据
    - 无需设置大的Cache
    - 提高了Cache块利用率
  - 动态共享
    - 可以在核之间动态分配Cache空间

7.3.1 多核处理器定义与结构

- **共享Cache的缺点：**
  - 核数量的增加，导致需求增加。包括：
    - 更高的带宽
    - 更大的Cache容量 → 导致更大的延迟
  - 命中延迟增加，因为要经过互连网络
  - 设计更为复杂
  - 某个核可能将其他核的数据替换出去

7.3.1 多核处理器定义与结构

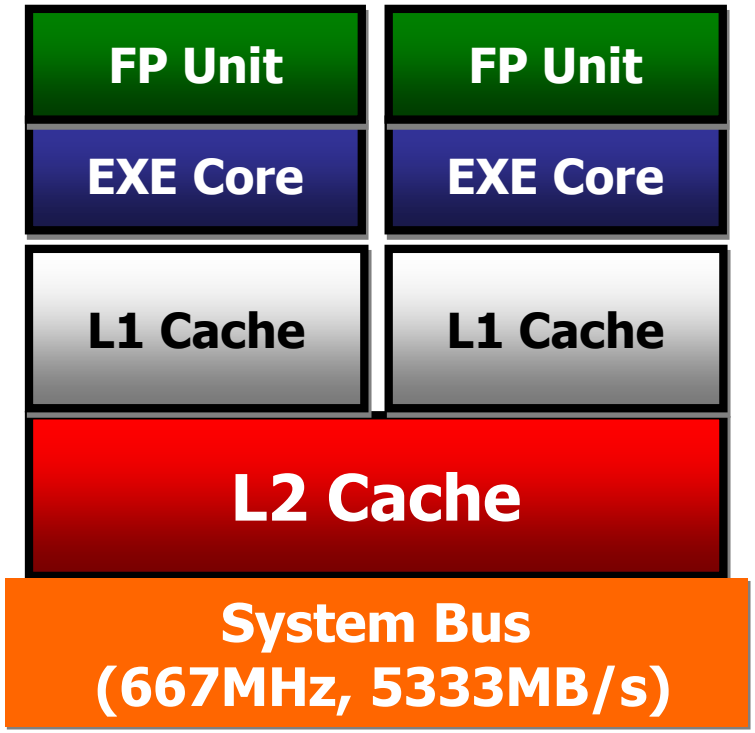
- 目前已有很多双核处理器，如：
  - Intel：Pentium D and Pentium Extreme Edition, Core Duo(2), Woodcrest, Montecito
  - IBM PowerPC
  - AMD Opteron / Athlon 64
  - Sun UltraSPARC IV
- 多核处理器 如：
  - IBM Cell (非对称)
    - 双核 PowerPC + 8个协处理单元
  - Sun Niagara
    - 8 cores，每核4个超线程
  - 通用计算图形处理器 (GPGPU) 等

7.3.1 多核处理器定义与结构

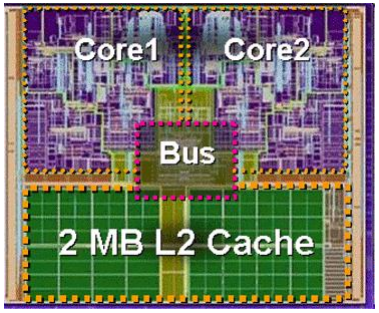
- 下列处理器分别采用了上面介绍的结构：
  - Intel Core Duo
  - Intel Core i7
  - AMD Opteron
  - ARM11 MPCore

7.3.2 Intel多核处理器

- 2006年，Intel推出了两个x86超标量双核处理器Core Due



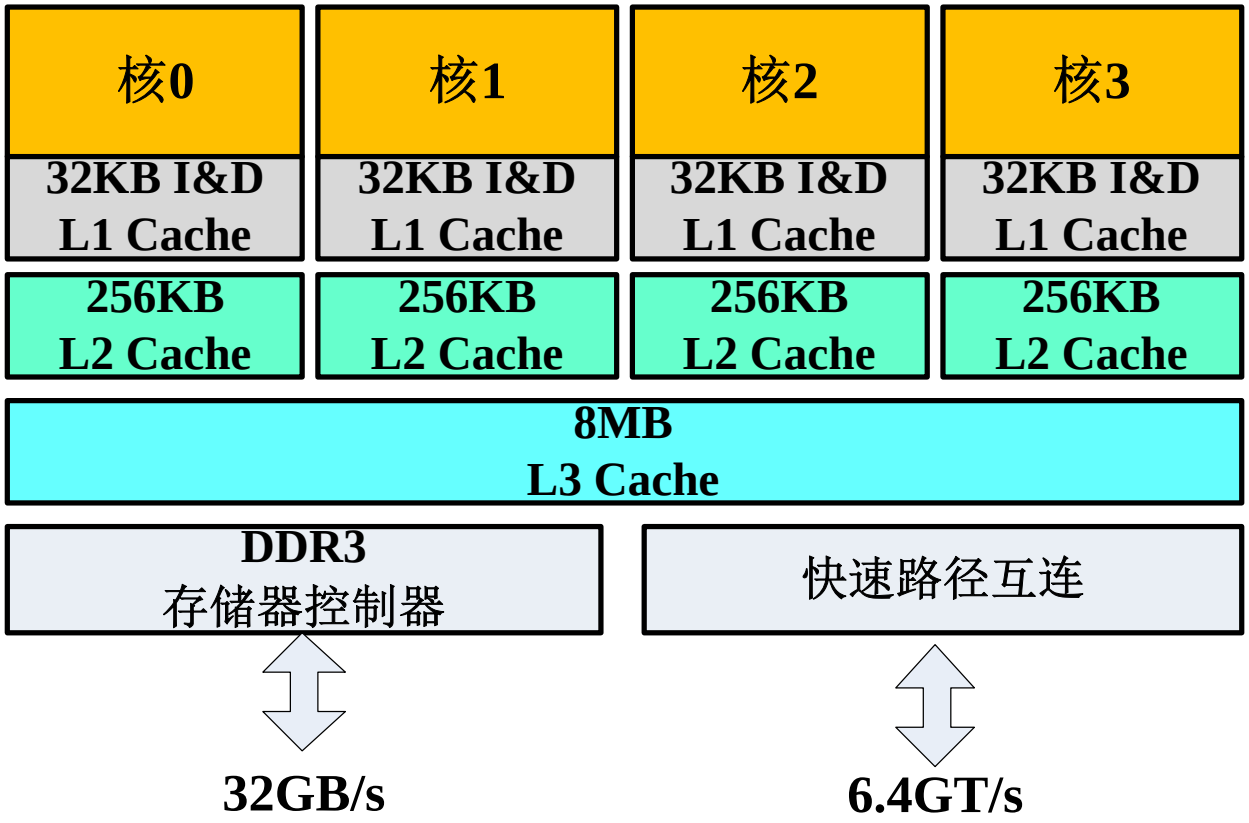
Intel Core Duo结构



- ☑ 双核
- ☑ 每核有自己的执行资源
- ☑ 每核一个私有 L1 cache
  - 32K 指令 和 32K 数据
- ☑ 双核共享 L2 cache
  - 2MB 8-路组相联；每行64字节
  - 10个时钟周期延迟；写返回更新策略

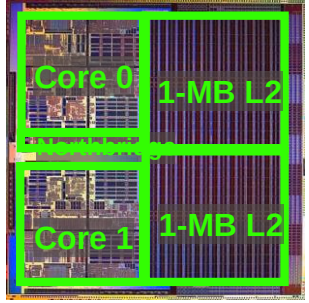
7.3.2 Intel多核处理器

- 2008年，Intel推出了4核 4个x86 SMT的Core i7

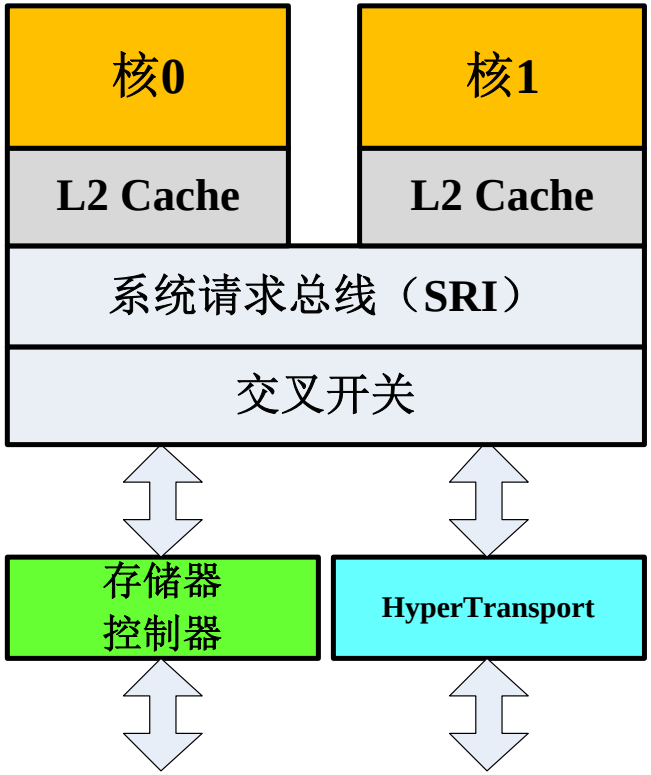


Intel Core i7结构

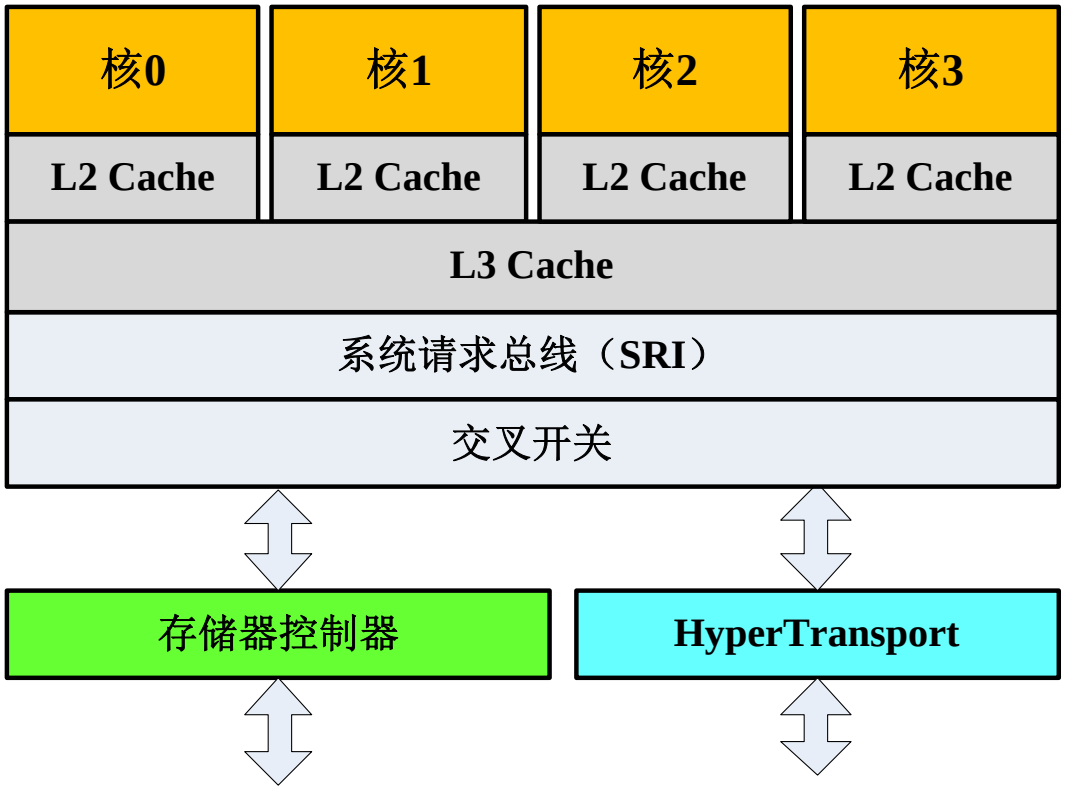
7.3.3 AMD多核处理器



- 2005年，AMD发布了第一款双核处理器 Opteron



AMD第一代
Dual-Core Opteron结构



AMD第三代Quad-core
Opteron处理器结构

7.3.3 ARM多核处理器

- **ARM 11系列： MPCore 多核架构**

ARM Cortex-A9 MPCore架构示意图

学习内容

- 7.1 多处理机概念
- 7.2 多处理机结构
- 7.3 多核处理器
- 7.4 多处理机的多Cache一致性
- 7.5 多处理机的机间互连形式
- 7.6 程序并行性
- 7.6 多处理机性能
- 7.7 多处理机操作系统

7.4 多处理机多Cache一致性

- Cache是提高系统性能的一种常用手段，多处理机也广泛应用Cache。
- 优点：
  - 减少访问时延，降低对存储器频宽要求
  - 减少同时访问存储器时的冲突 等

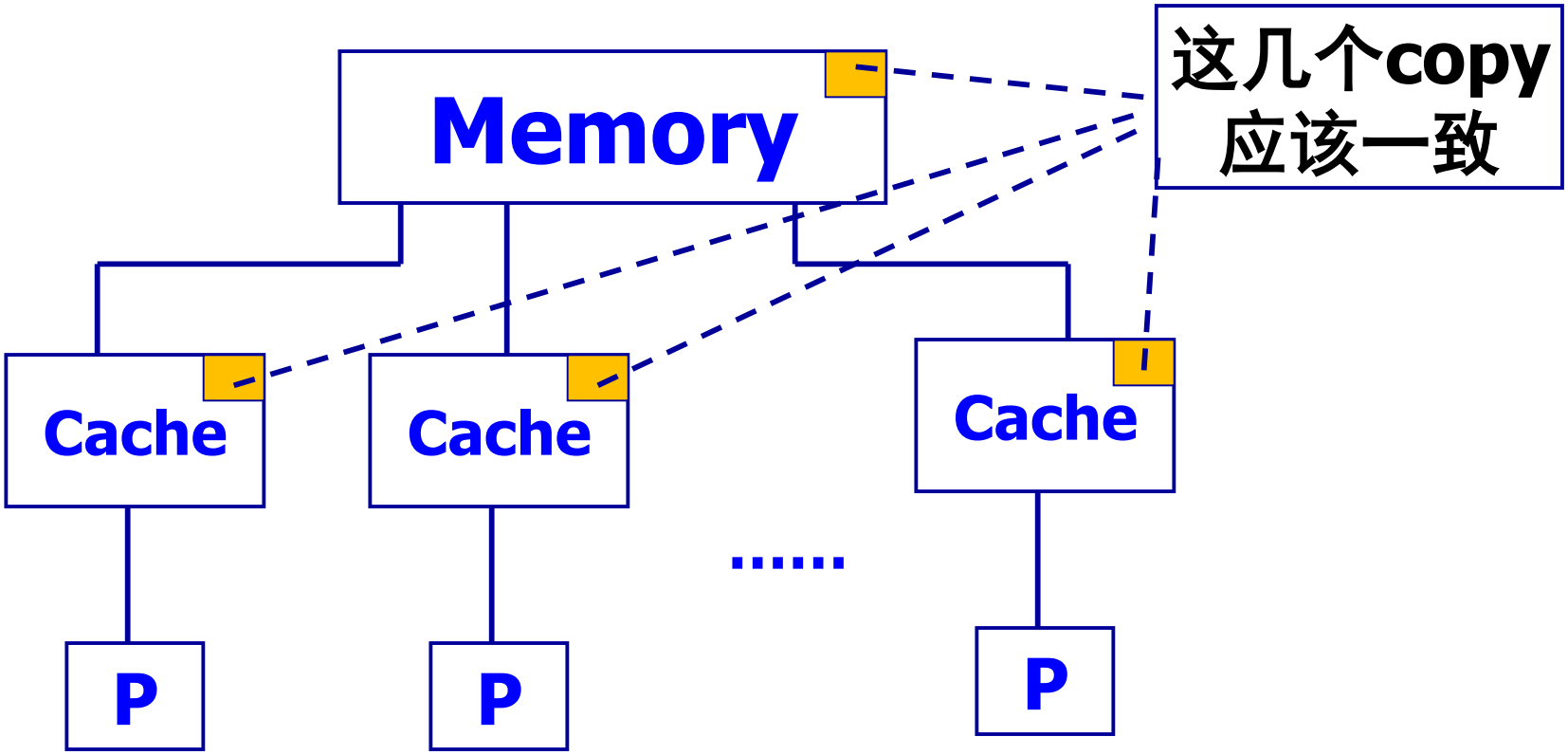
7.4 多处理机多Cache一致性

- 当把共享数据装入Cache中时，会在多个Cache中形成副本
- 新问题：

共享数据进入Cache时产生一个新问题：

Cache一致性 (Cache Coherence) 问题

7.4.2 多Cache一致性问题的产生

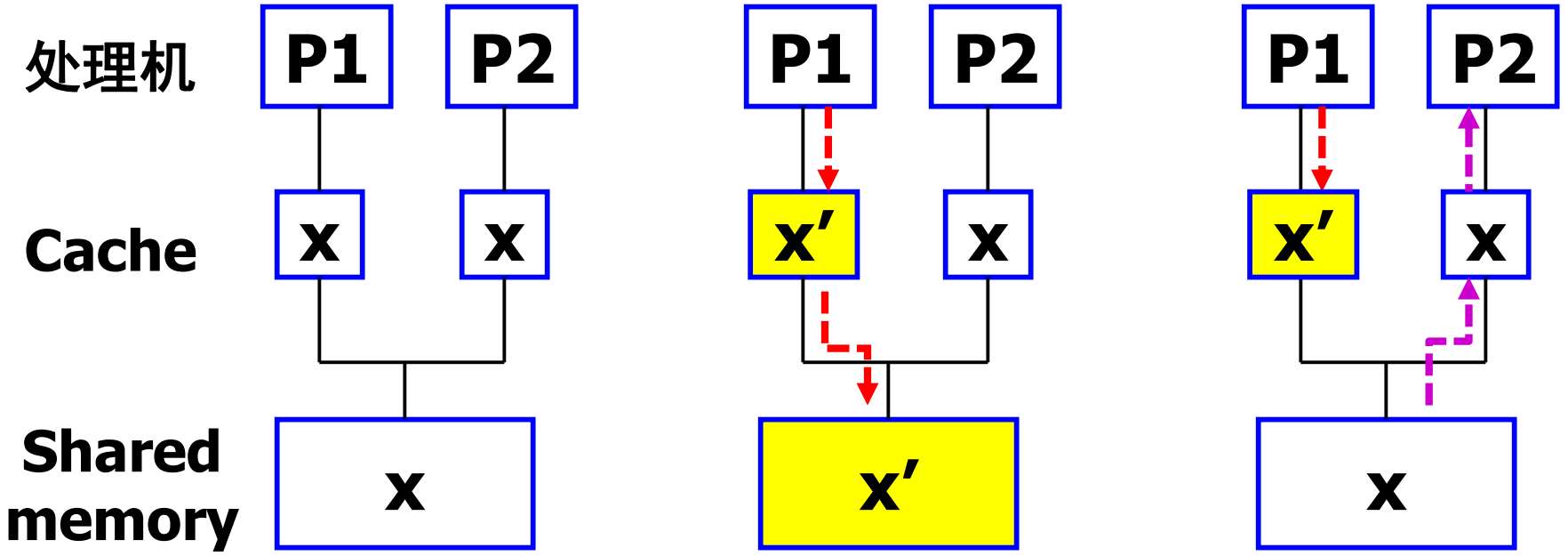


在多处理器系统中，多个Cache中对应的共享数据的copy应该一致。

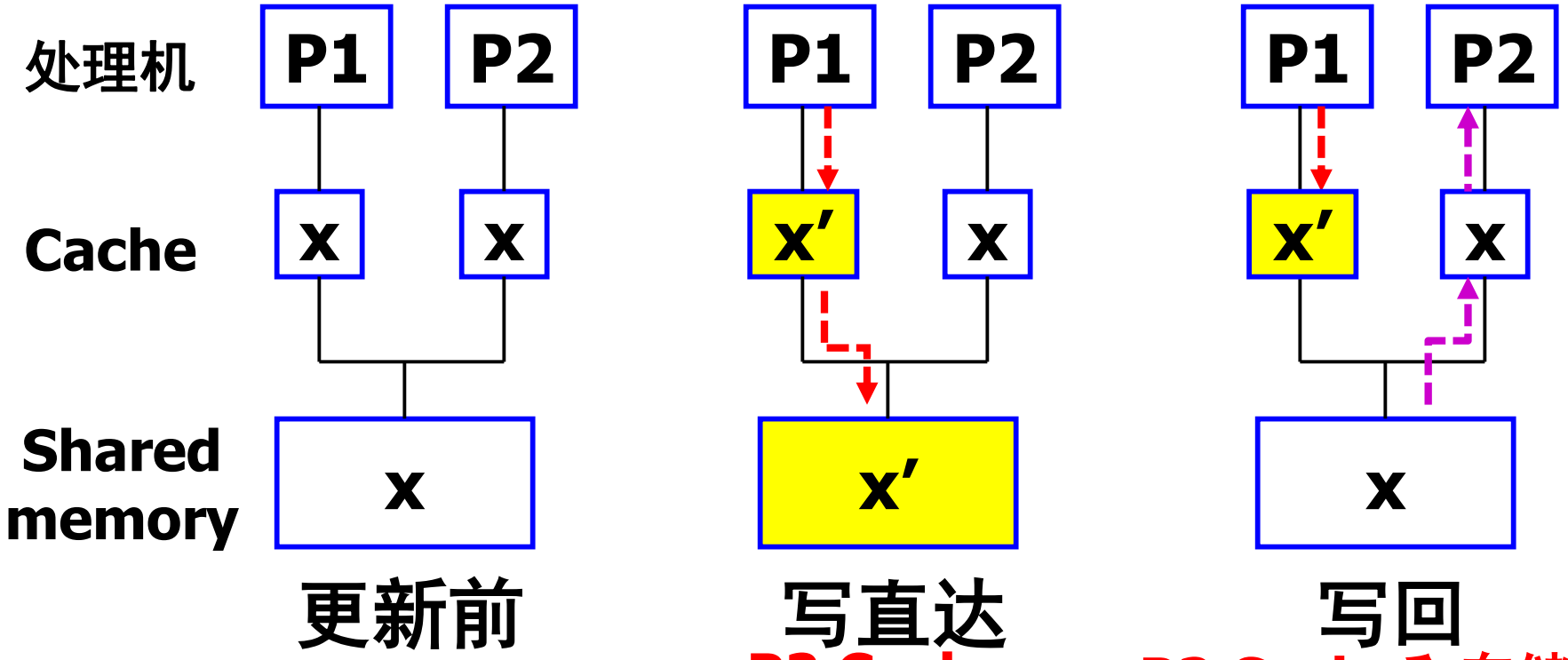
7.4.2 多Cache一致性问题的产生

- 不一致的三种原因：
  - 共享可写数据
  - 进程迁移
  - I/O传输

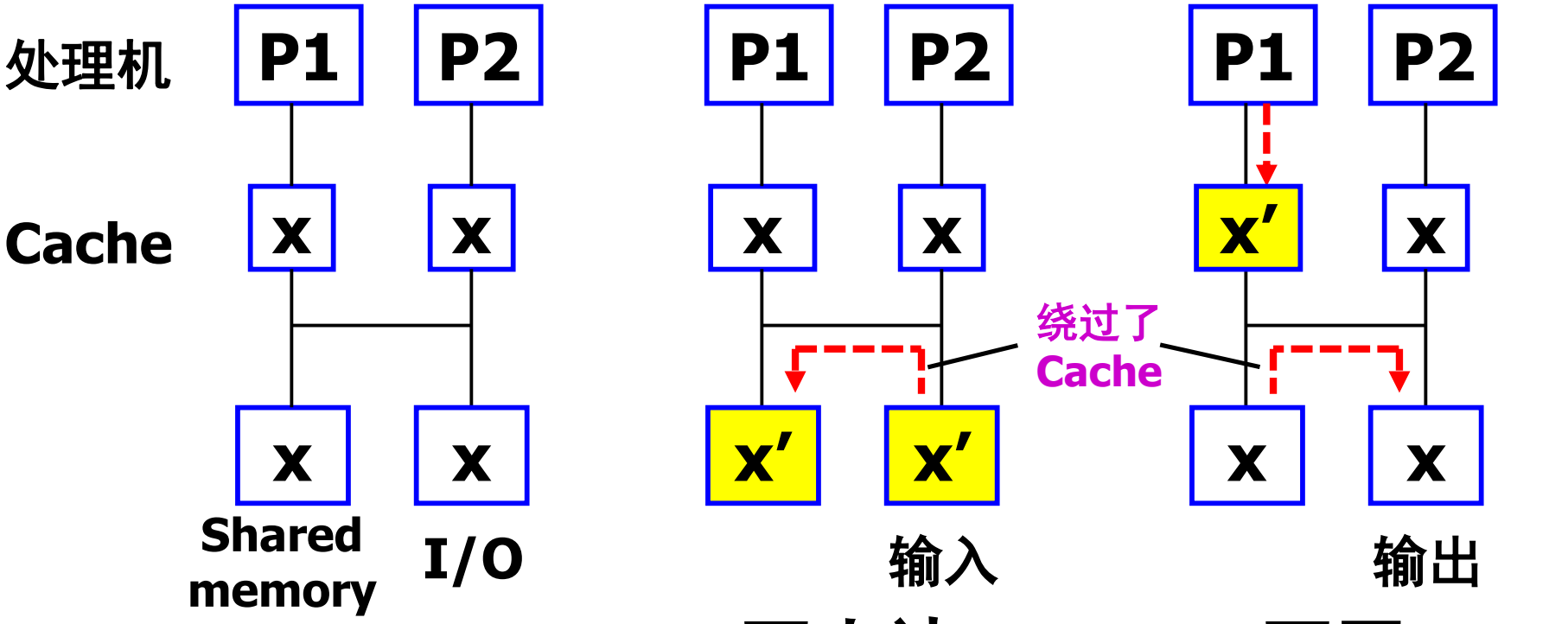
共享可写数据引起的不一致性



进程迁移引起的不一致性



I/O传输引起的不一致性



7.4.3 多Cache一致性问题解决方法

- 在上述三种情况中，不论采用写直达法还是写回法，在多处理机中都可能引起Cache不一致问题。
- 解决方法：2类
  - 基于硬件的方法
    - 思想：跟踪共享数据的状态
    - Cache一致性协议：监听协议和基于目录的协议
  - 基于软件的方法
    - 由编译和操作系统检测并解决

基于硬件的方法：两种Cache一致性协议

1. 监听协议(Snooping Protocol) — 拥有数据副本的Cache各自记录数据块的状态，无集中的状态。

2. 基于目录的协议(Directory based Protocol) — 将每个数据块的共享状态记录保存在某个地点 — 目录。

监听协议 (Snoopy Protocol)

- Goodman 1983
- 适用于：具有广播能力的总线结构多机系统。但只适用于小规模的多处理机系统
- 分布式算法：分散到所有Cache控制器。各Cache控制器自行产生状态变化及相应操作
- 两种策略：
  - 写无效（Write-Invalidate）（或 写作废）
  - 写更新（Write-Update）（或 播写法）

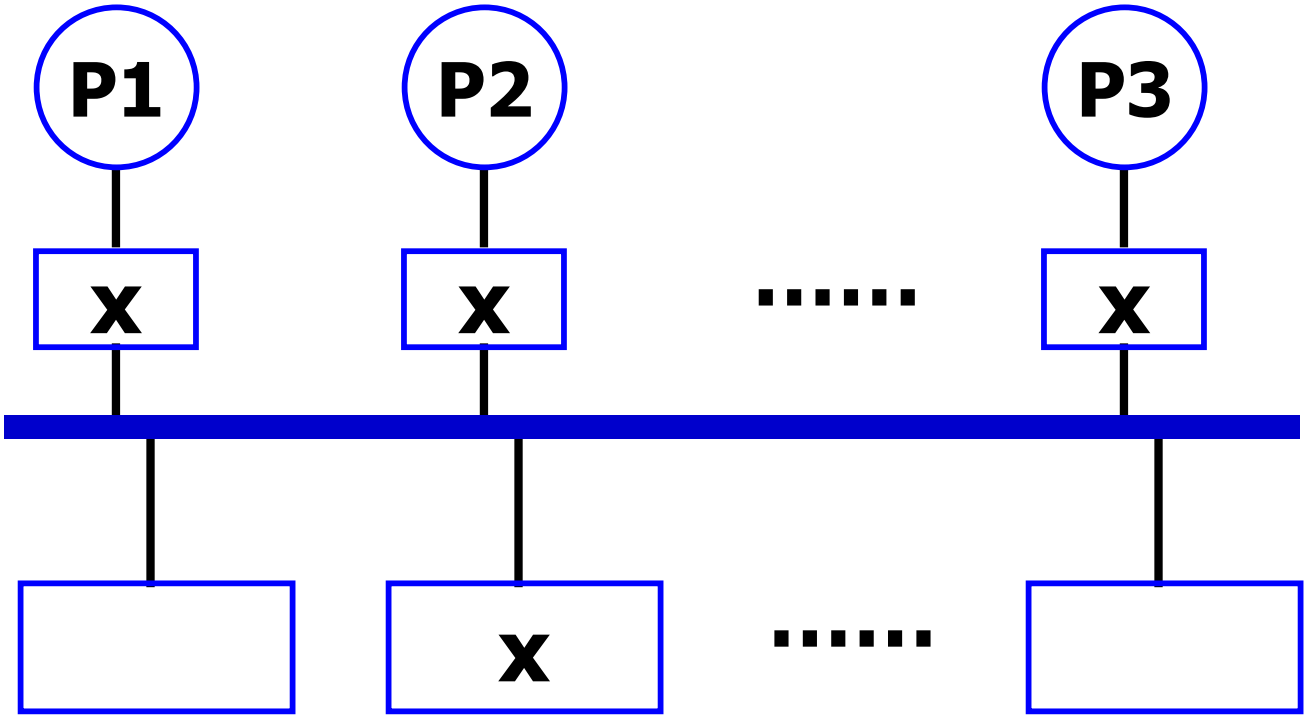
监听协议 (Snoopy Protocol)

- 写无效（Write-Invalidate）（或 写作废）
  - 任何一个处理器写或修改它的私有Cache中的共享数据时，它都使所有其它Cache中的副本失效。
  - 对Write-through：它也更新存储器中的副本（最终是：只有一个Cache中的副本和存储器中的副本是有效的）。
  - 对Write-back：它使存储器中的副本也失效（最终是：只有一个Cache中的副本是有效的）。

监听协议 (Snoopy Protocol)

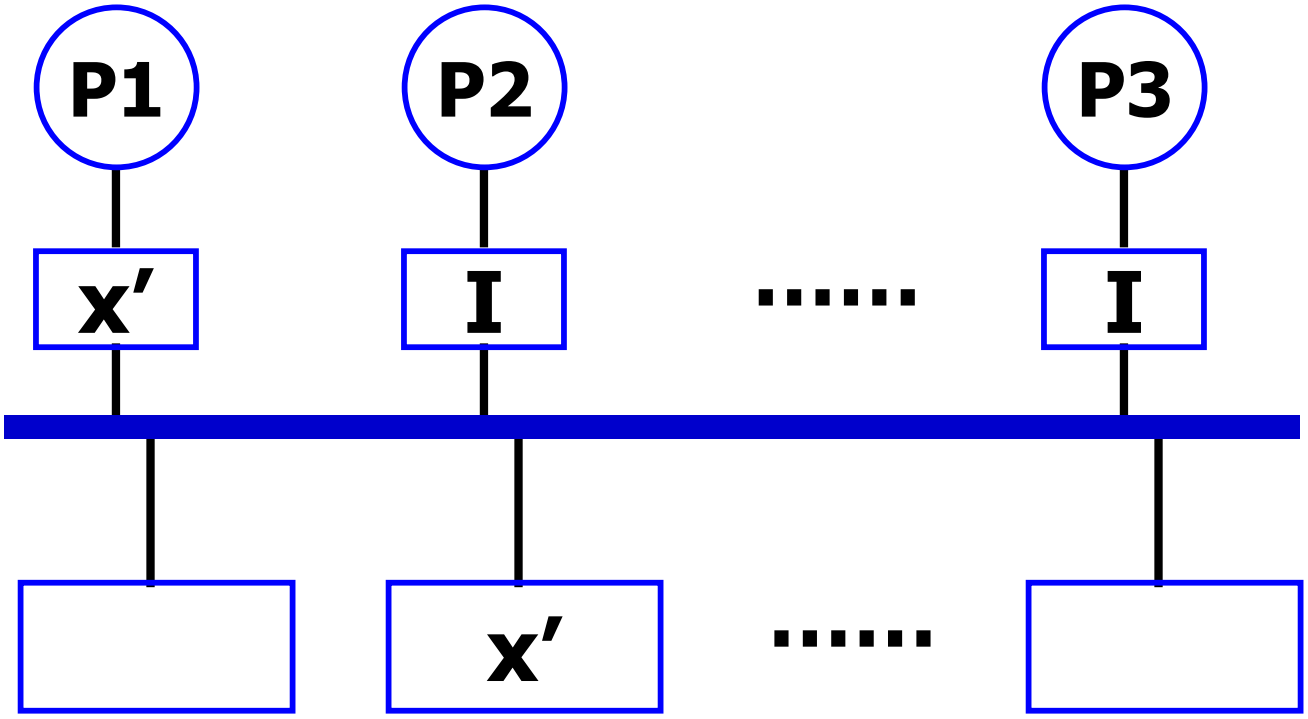
- 写更新（Write-Update）（或 播写法）
  - 任何一个处理器写或修改它的私有Cache中的共享数据时，它都立即更新所有其它Cache中的副本。
  - 对Write-through：它也更新主存储器中的副本。
  - 对Write-back：对存储器中副本的更新延迟到这个Cache被置换的时刻。

监听协议 (Snoopy Protocol)



更新前
数据块x在共享存储器和3台处理机的Cache中的副本一致

写无效 + 写直达



处理机

Cache

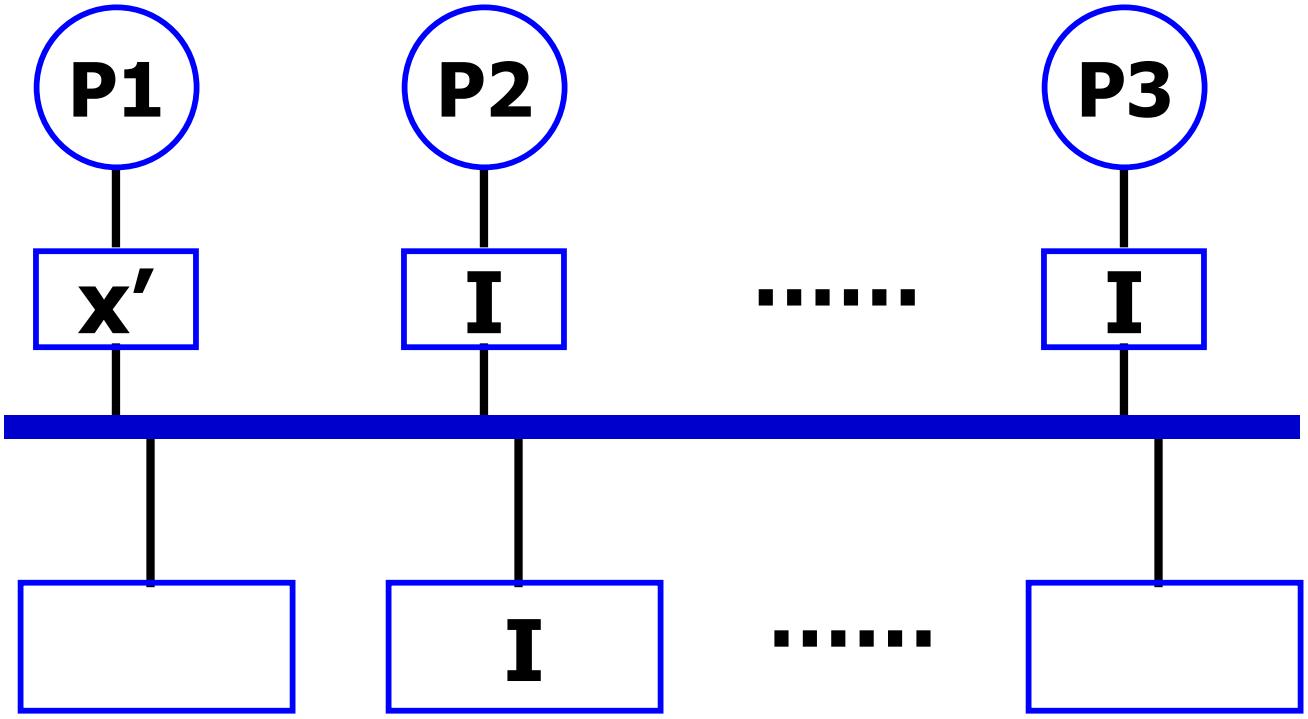
总线

共享存储器

I表示无效

只有一个Cache中的副本和存储器中的副本是有效的

写无效 + 写回



处理机

Cache

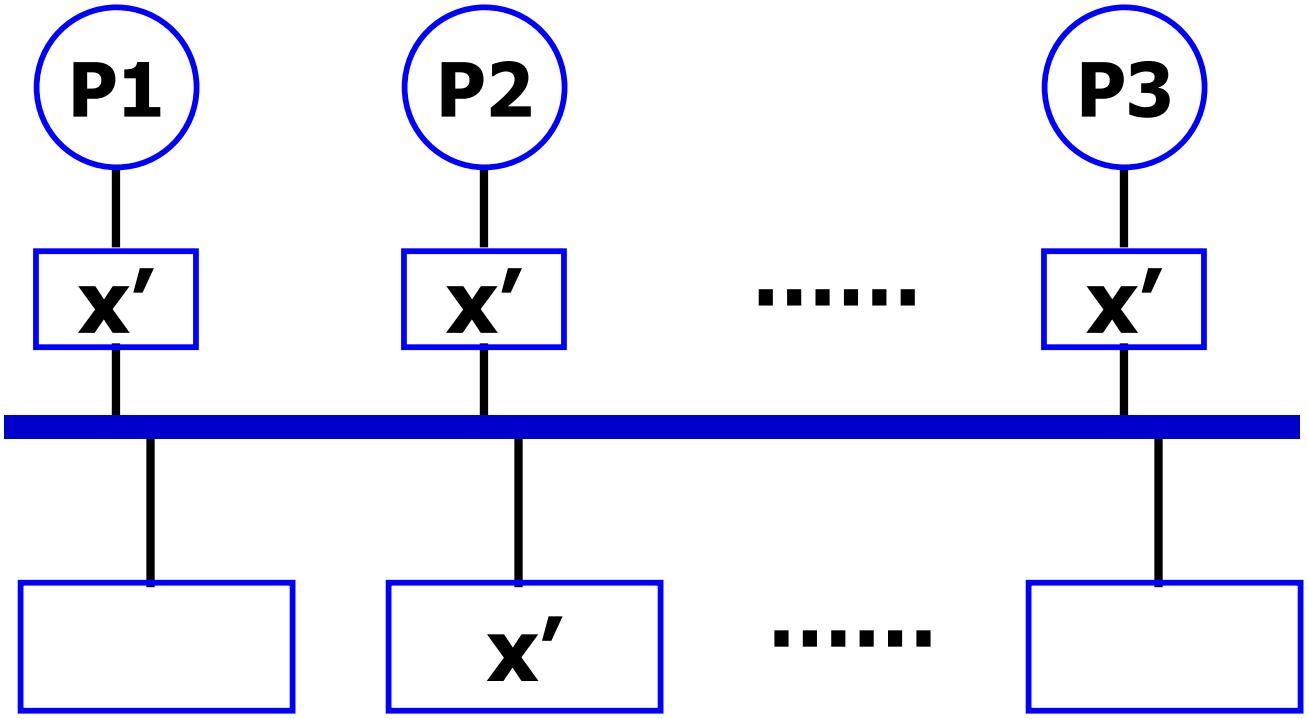
总线

共享存储器

I表示无效

只有一个Cache中的副本是有效的

写更新 + 写直达



更新所有Cache以及主存储器中的副本

写无效

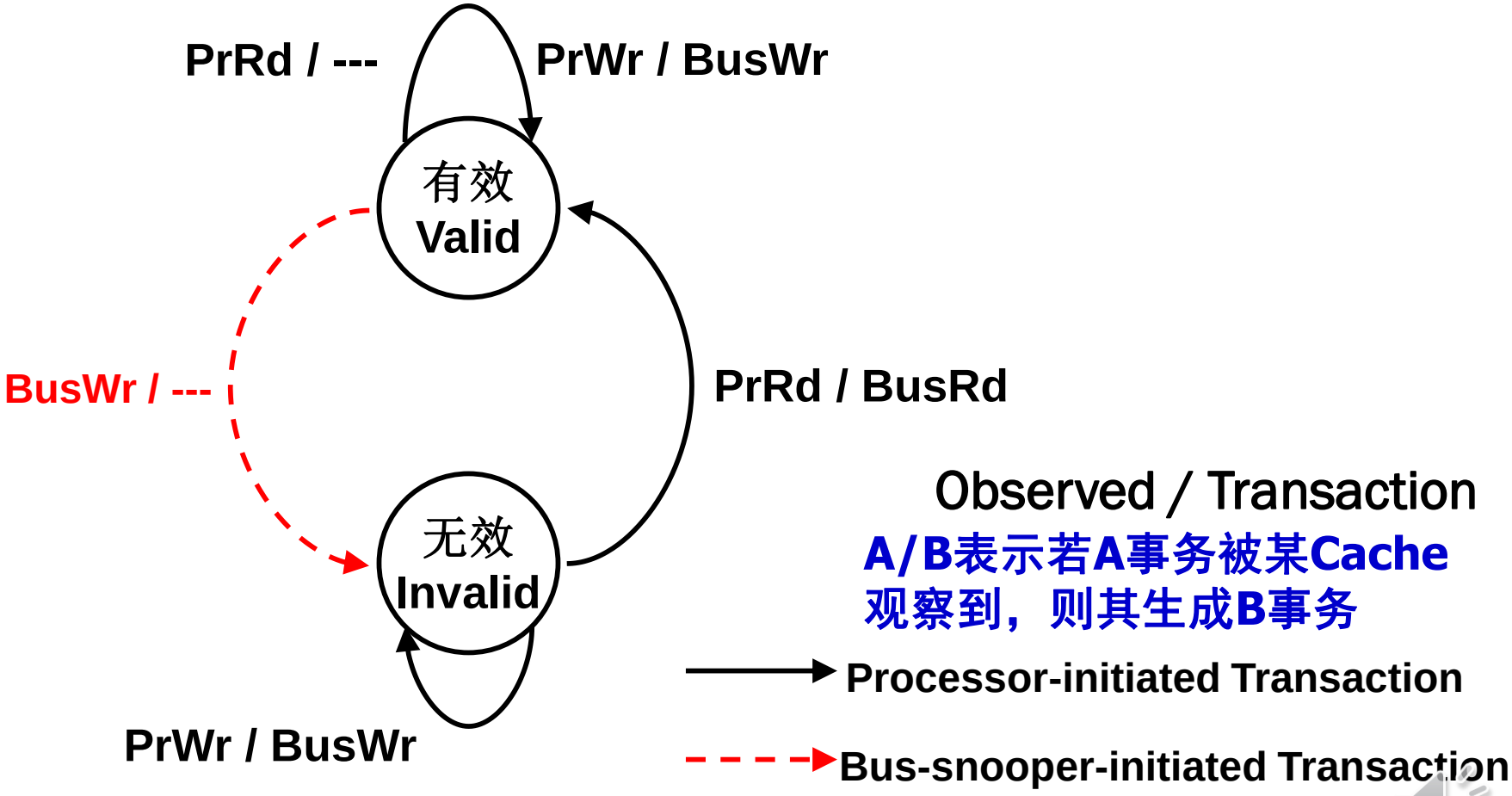
- **主要开销在两个方面：**
  - （1）作废各Cache副本的开销；
  - （2）由作废引起缺失造成的开销，即处理机需要访问已经作废的数据时将引起Cache的缺失。
- **后果：**
  - 如果一个处理机经常对某个块连续写，且各处理处理机间对共享块的竞争较小，这时写无效策略维护一致性的开销是很小的。
  - 如发生严重竞争，将产生较多的作废，引起更多的作废缺失，结果是共享数据在各Cache间倒来倒去，产生颠簸现象。当缓存块比较大时，这种颠簸现象更为严重。

监听协议：写无效 + 写直达

- **Cache块状态：**
  - **有效(V)：干净块(数据与主存中相同)，该块可同时在多个Cache中存在**
  - **无效(I)：此块不在Cache中【或已作废】**
- **协议基本思想：**
  - **满足Cache工作流程需求，写操作时写主存，通过总线事务通知其它Cache作废该块**

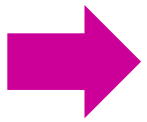
监听协议：写无效 + 写直达

Invalidation-based Protocol on Write-Through cache



监听协议：写无效 + 写回

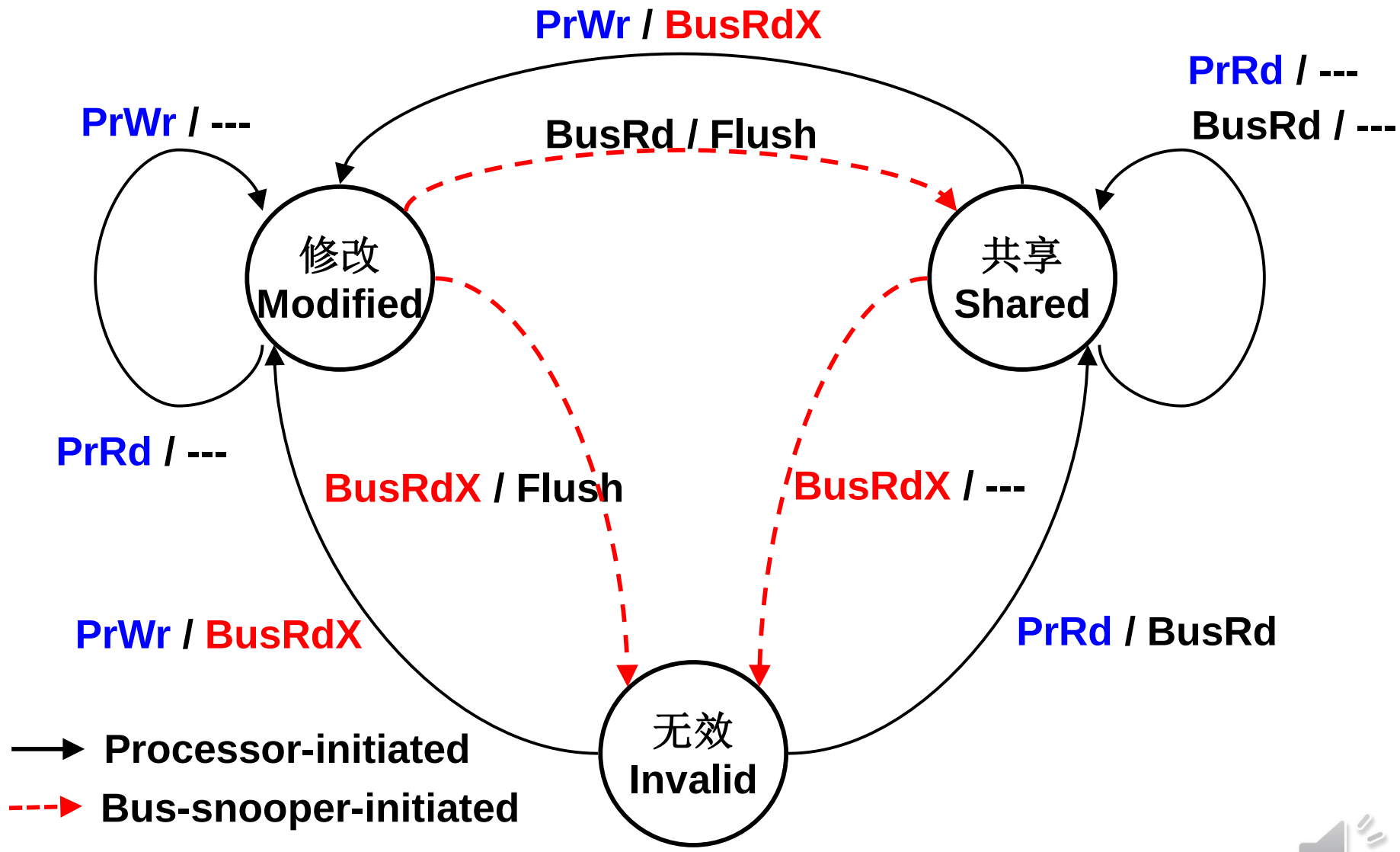
- Cache块状态
  - 修改(Modified) (或增加 独占(Exclusive))
    - 数据被修改了
    - 仅该Cache拥有正确的副本
  - 共享(Shared)
    - 存储器一致
    - 一个或多个Cache都拥有正确的副本
  - 无效(Invalid)



MSI 协议
MESI 协议

写回协议：
在更新存储器之前，可以多次写Cache行

监听协议：写无效 + 写回 (MSI)



监听协议的开销分析

- （1）采用写无效协议
  - 无效后，当其它处理机再读该副本时，出现Read miss，要有开销。
- （2）采用写更新协议
  - 需要更新所有Cache和memory中的副本，所以开销大，有些处理机对更新后的数据可能不会使用。

基于目录的协议

- 基于目录协议的基本思想：
  - 当处理机台数增加时，一般不用总线结构，而采用多级互连网络。但多级互连网络实现广播功能代价很大。
  - 为什么需要广播功能？把一致性命令，如Write、Read等命令发送给所有的Cache。
  - 能不能只发送给存放该副本的Cache?

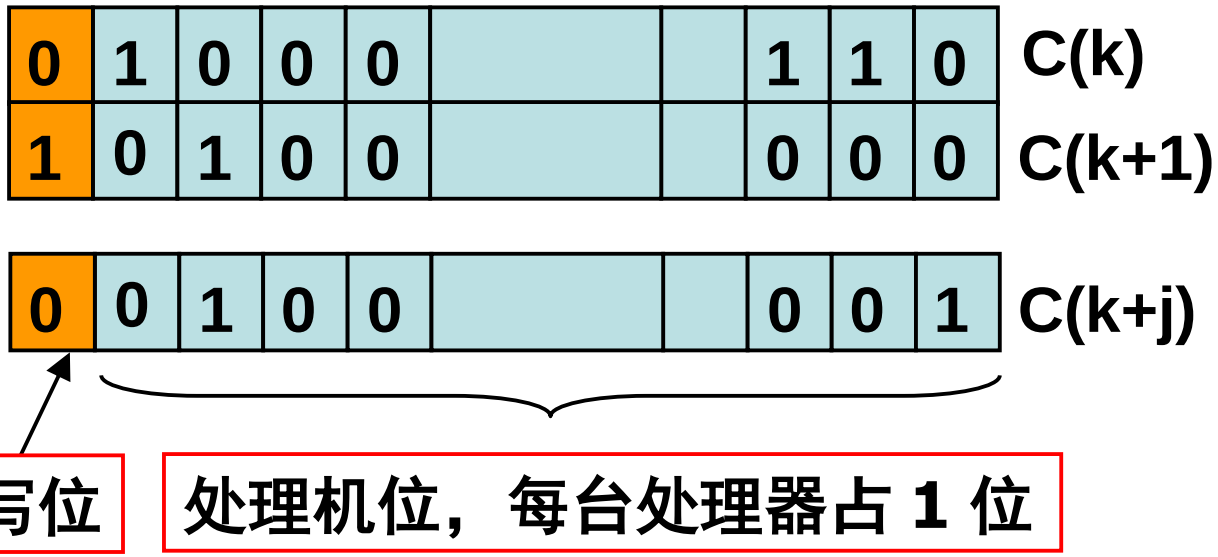
    ——基于目录的协议的基本思想

基于目录的协议

- 在每个结点增加目录存储器，用于存放目录。
- 目录里放什么？
  - 有关Cache副本驻留在哪里的信息：所有共享数据块的所有Cache副本的地址表。
  - 目录必须跟踪记录每个共享数据块的状态。

基于目录的协议

- 目录结构



处理机位表示相应处理机Cache中是否存在共享数据块的副本（存在或不存在）。

如果重写位为“1”，而且有且只有一个处理机位为“1”，则表示该处理机可以对该数据块进行修改。

基于目录的协议

目录的存放方式：2种（1/2)

- **1. 集中式目录方式（中心目录）**
  - 1976年 Tang提出。
  - 用一个中心目录存放所有Cache目录的副本，它能提供为保证一致性所需要的所有信息。
  - 缺点：容量非常大，必须采用联想方法来检查，扩展性差，冲突多，检索时间长。

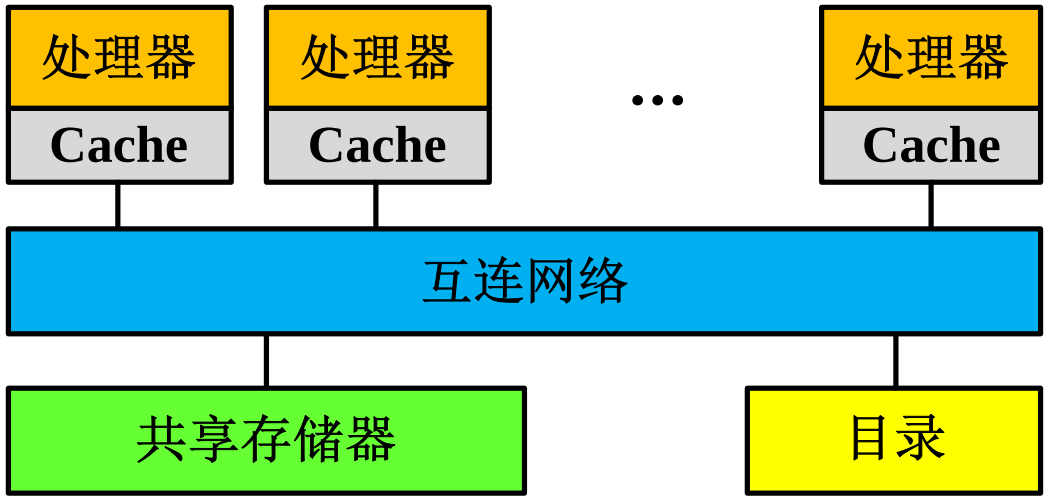
基于目录的协议

目录的存放方式：2种（1/2）

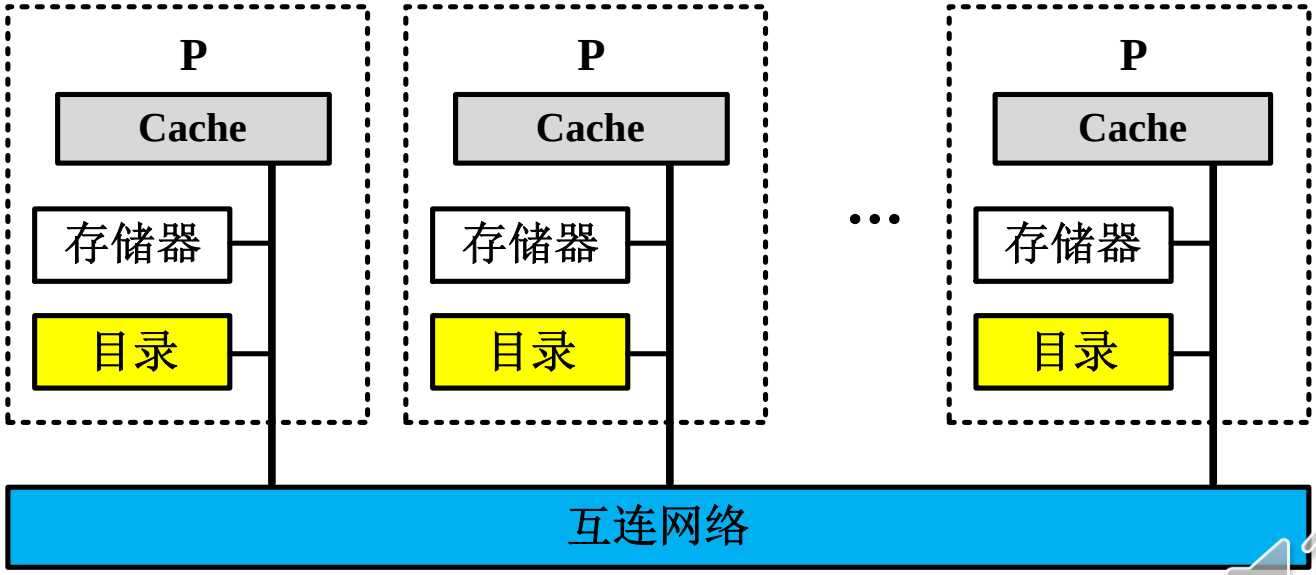
- **2. 分布式目录方式**
  - 1978年 Censier和Feautrier 提出。
  - 每个存储模块维护各自的目录，目录中记录着每个存储块的当前信息。当前信息指明哪些Cache有该存储块的副本。

基于目录的协议

集中式目录方式



分布式目录方式

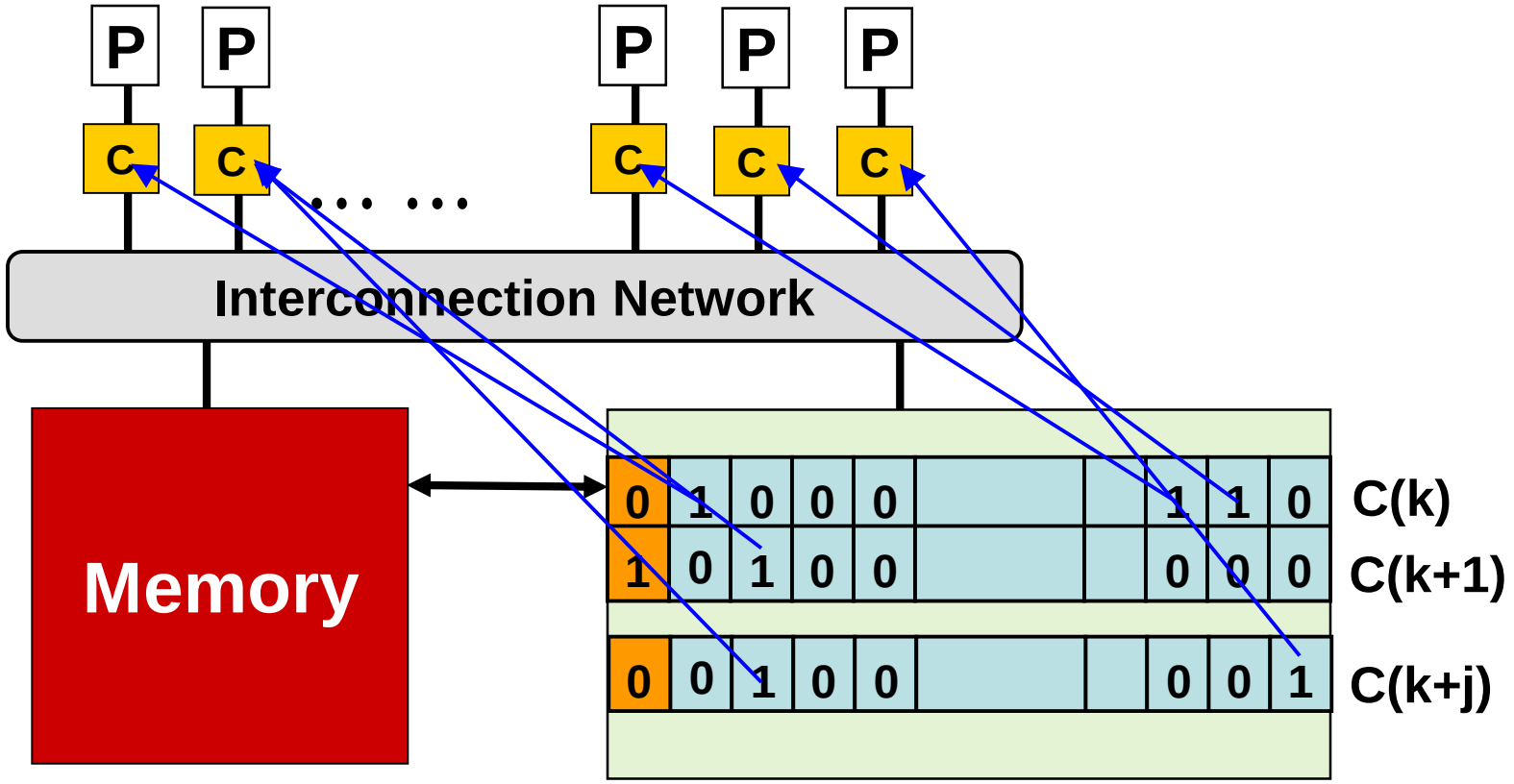


基于目录的协议

目录类型：3种

- **1. 全映射（full map）目录：**存放与全局存储器中每个块有关的数据。系统中的每个Cache可以同时存储任何数据块的副本，即每个目录项包含N个指针（N是处理机数目）。
- **2. 有限（limited）目录：**每个目录项有固定数目的指针（小于N）。
- **3. 链式（chained）目录：**将目录分布到各个Cache（其余同全映射目录）。

全映射目录

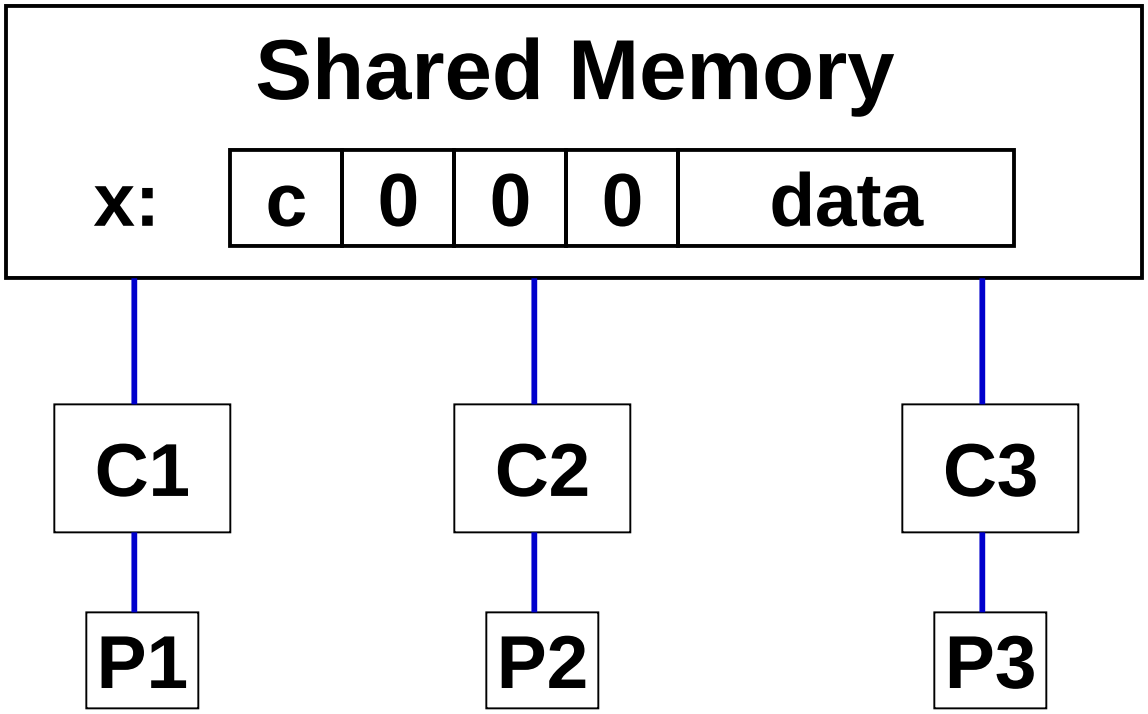


一个重写位，说明是否允许向Cache数据块写入数据

每台处理机一个处理机位，指示该处理机Cache中是否存在该数据块

我们来看三台处理机（3个Cache）使用全映射目录的例子。

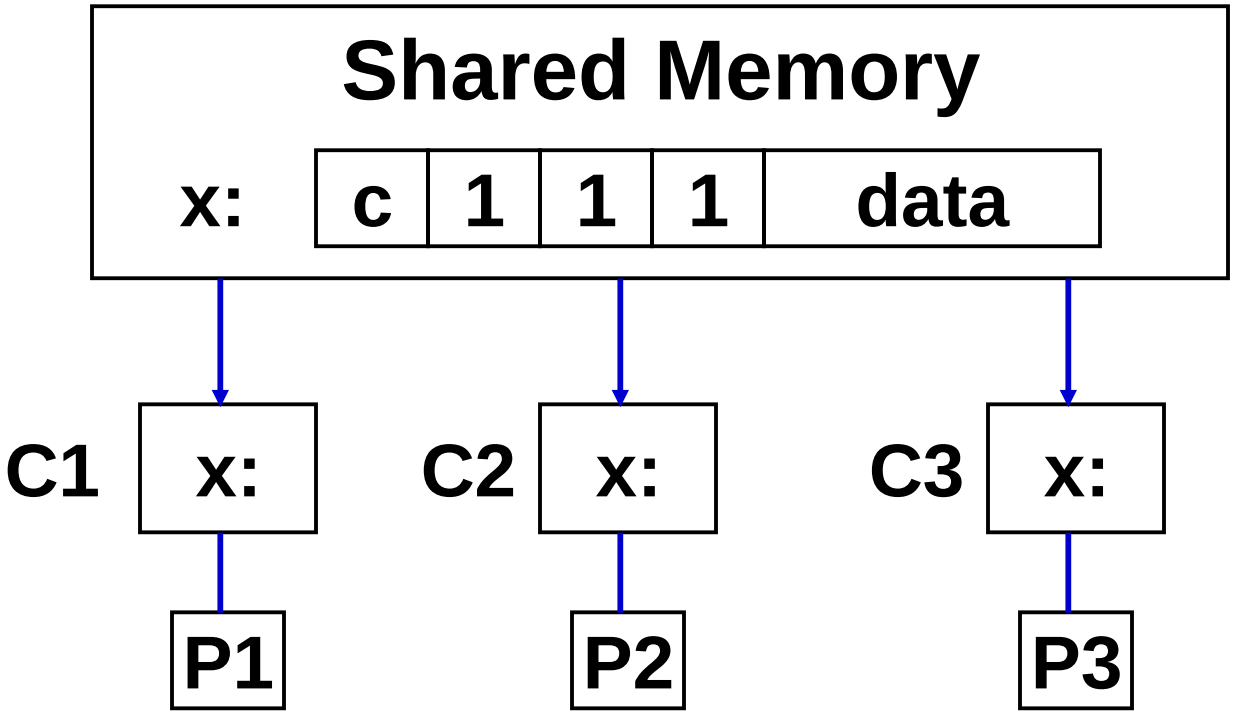
（1）C1，C2，C3都没有单元X的副本。



（2）C1，C2，C3同时请求X单元的副本。

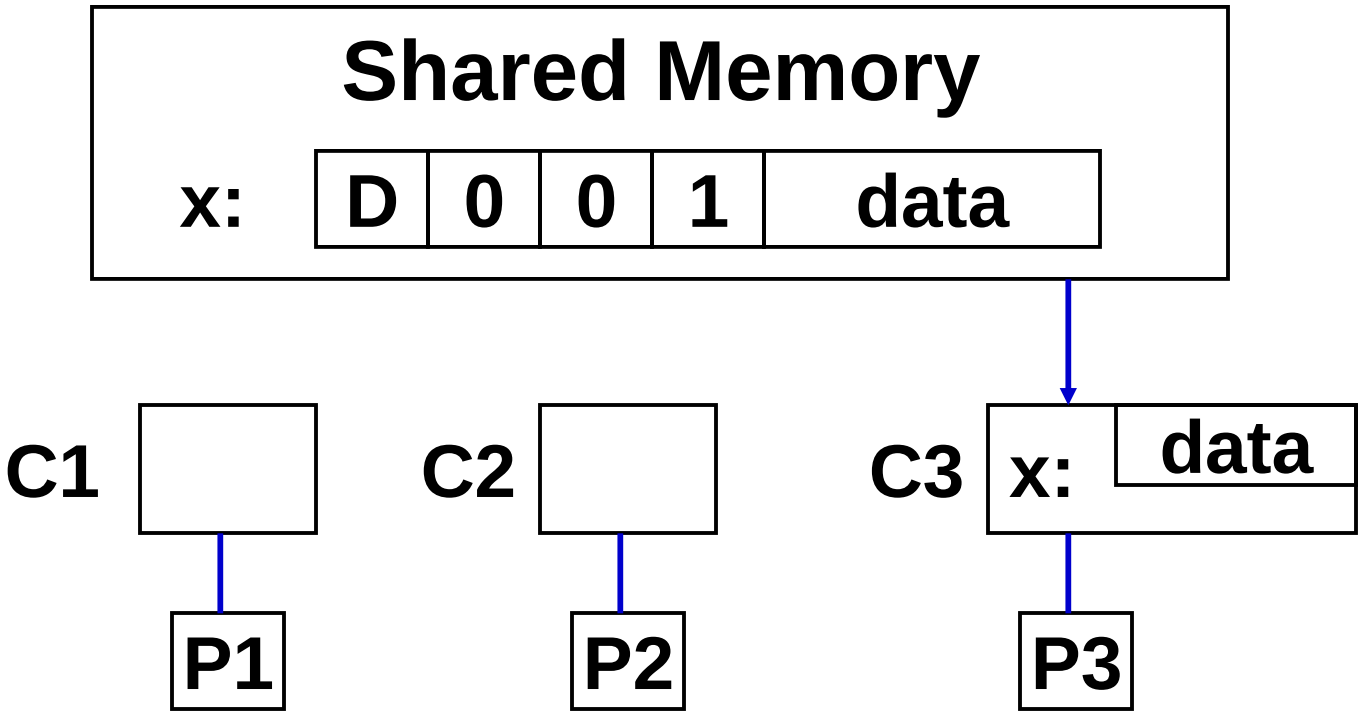
这时目录项中的3个指针（处理机位）被置1，表示这些Cache中已有数据副本。

目录项的重写位被置为未写 (c) 状态，表示无一处理机允许写入该数据块。



（3）C3请求对该块的写允许权。

成功后，重写位被置成允许写(D)状态，且有一个指针指向C3的数据块。



（3）C3获得写允许权的过程

P3向C3发出写请求时 (1/2)：

- （1）C3检测出包含单元X的Cache块是有效的，但Cache中的块重写位状态表示不允许处理机对该块进行写操作。
- （2）C3向包含单元X的存储器模块发出写请求，并暂停P3工作。
- （3）该存储器模块发出一个无效请求给C1和C2（根据目录项的内容发2个或多个无效请求）

（3）C3获得写允许权的过程

P3向C3发出写请求时 (2/2)：

- （4）C1和C2收到无效请求后，把Cache块置为无效，并发送一个回答信号给请求的存储器模块。
- （5）存储器模块收到回答信号后，将重写位置1，清除指向C1、C2的指针，发出允许信号给C3。
- （6）C3收到写允许信号后，修改Cache的状态并激活处理机P3。