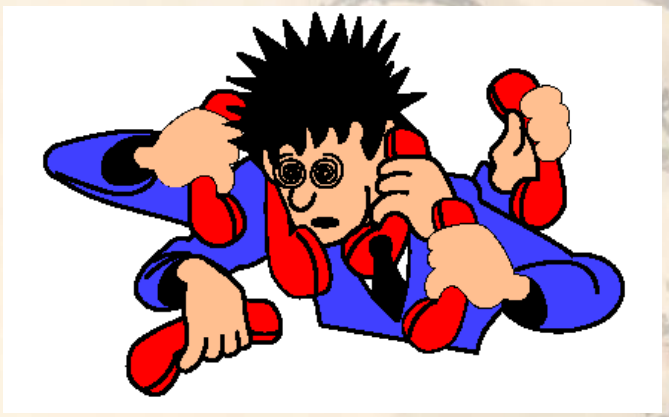
Chapter four

Multiprocessors and Thread-Level Parallelism



陈文智

4.1 Introduction

4.1.1 基本概念

一、有关计算机结构发展的观点

- 观点----单处理器发展已接近结束。
- 反观点----在1985-2000十五年间，单处理器机器速度提高率为50年代初使用晶体管以来最高。认为单处理器前途无限。
- 折衷观点----并行机毫无疑问将在未来起更大作用，但不是现在（but not now)。

二、并行机在未来作用更大的理由

- 多处理器连接是提高性能的主要技术
  - 因为微处理器可能仍将是单处理器技术，因此为了使微处理器性能超出单处理器，合乎逻辑的方法是将多个微处理器连接在一起，这比设计专用处理器要便宜得多。
- 十几年来,基于开发指令级并行性等技术来提高处理器性能的体系结构更新的步伐是否能维持下去尚未明朗。
  - 1985年以来，曾给微处理器带来高速性能的体系结构创新步伐是否仍能保持下去。我们在前面章节中已看到，现代多发射处理器已变得无法相信的复杂。因此，期望通过增加复杂度和硅片面积来提高性能的势头将有所减弱。
- 影响并行机普遍使用的主要障碍----并行软件（并行编程）问题已出现缓慢而稳定的进展。

作者观点

- 微处理器速度的高速进展至少将维持几年。一旦单微处理器进展步伐放慢之后，多处理器系统结构将更有吸引力。

三、多处理器系统结构现状

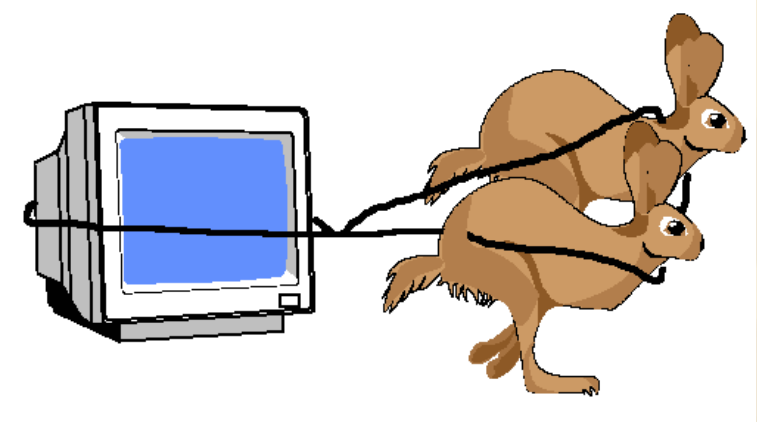
- 内容丰富，分支众多的领域
  - 不成熟
  - 很多的新思想昙花一现（coming & going）
  - 并且是失败多于成功。

  因此难以完整地介绍这一领域，避免经不起时间考验。

- 重点介绍多处理器设计的主流----
  - 中、小规模多处理器。处理器数<128。

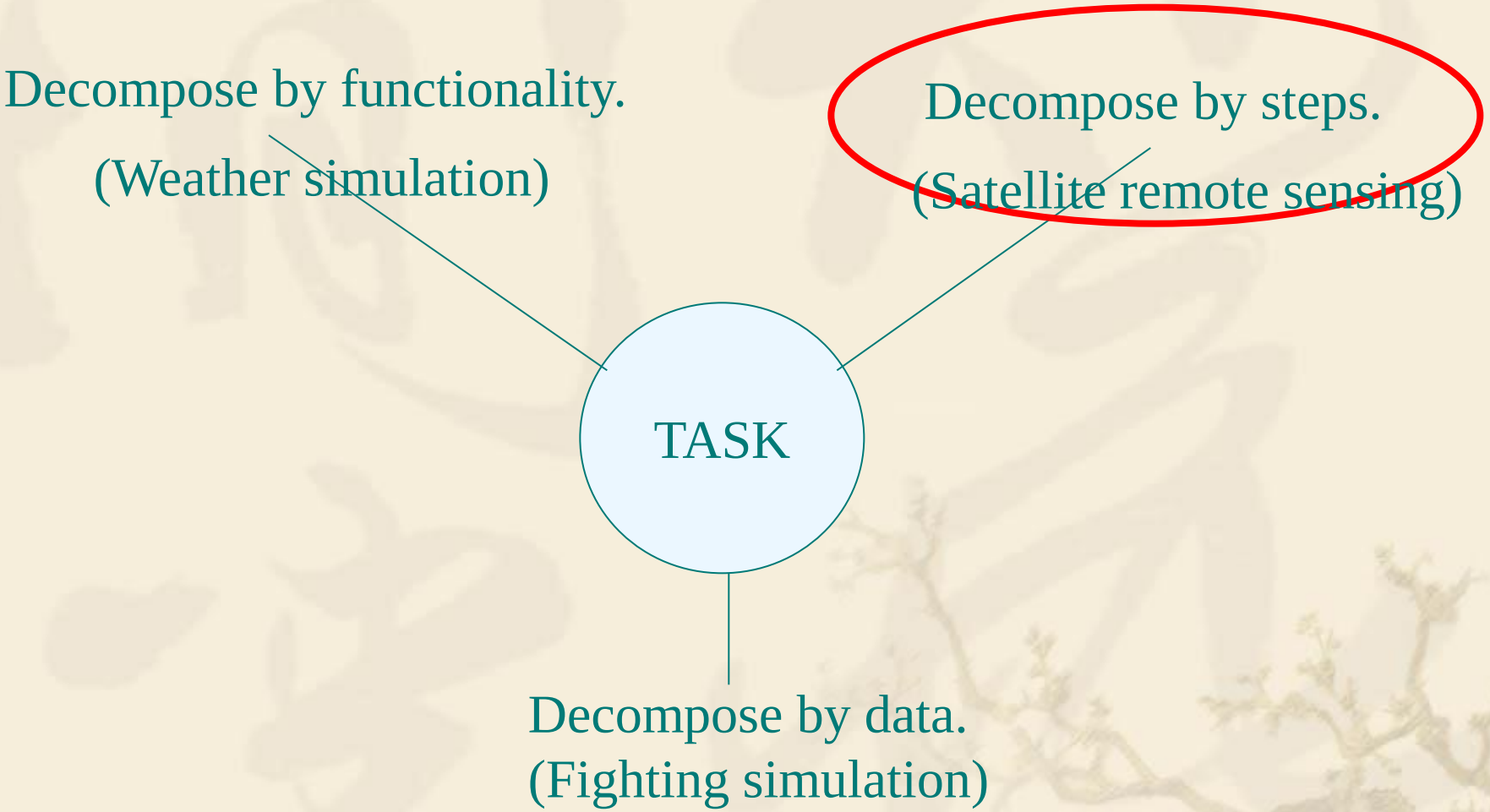
- 简单介绍处理器数>128的大规模并行机
  - 此类机器的未来系统结构并未确定。
  - 市场生命力值得怀疑。

四、并行性含意



- 同时性：两个或两个以上事件在同一时刻发生
- 并发性：两个或两个以上事件在同一时间间隔发生
- 流水线：两个或两个以上事件在可能重叠的时间段内发生

五、并行技术


Decompose by functionality.
(Weather simulation)
Decompose by steps.
(Satellite remote sensing)
TASK
Decompose by data.
(Fighting simulation)

六、并行性的困难

- 任务分配非常困难
  - 可并行性：任务的并行性划分和分发
- 算法对并行性的限制
  - 算法不仅与问题有关，还与硬件有关
- 处理机之间的通信开销限制
  - 当通信开销大时并行处理技术得不偿失
- 并行处理环境
  - 可编程性

    并行开发环境需要并行开发语言、并行编译和并行操作系统支持
- 并行规模的确定
  - 可扩展性

4.1.2 并行系统结构分类

- 分类目的
  - 了解多处理器的不同设计方案；
  - 理解多处理器系统结构是如何逐步发展到今天主流形式的。
- 多处理器的基本思想，可追溯到计算机的早期阶段。
  - 提高性能
  - 提高可靠性

一、 Flynn分类法----指令流、数据流并行性

- SISD----单处理器
- SIMD----**Single instruction memory and control processor, Multiple data memories (each processor has its own data memory)(多为专用处理器）**
- MISD----未实现，无产品，但可能将来会有。
- MIMD----每一处理器取它自己的指令，运行于它自己的data，处理器多为现成的微处理器。

二、发展历史（1）

1. 历史上最早成功的多处理器机为SIMD

connection machine 2：65535个 1bit processors

Illiac IV： 64个 64bit processors

- 作为通用机，80年代重新崛起
  - **Thinking machine, MasPar**
- 但再次回落
  - **不灵活，不适用于很多问题，当scale down时，性/价比变差。**
  - **无法利用高性价比的微处理器，而必须设计专用processor。**
- **SIMD多处理器作为专用机，特别是在image和signal处理领域的array processor ，仍有前途。**

发展历史（2）

2. 近年，MIMD作为通用多处理器技术崛起。
   - **灵活性**
     - **既可作单用户机，面向解决单个的高性能应用；**
     - **也可运行多任务多道程序**
     - **或用于上述两种混合方式**
   - **可利用高性价比的微处理器(现有的芯片)。**
3. 多线程技术（线程并行性）
   - 多个处理器（n）处理多个（n）线程
   - 包含在独立线程中的并行性
     - 在多线程事务处理中有许多独立的进程
       - 查询、修改等

4.1.3 MIMD结构的二种类型

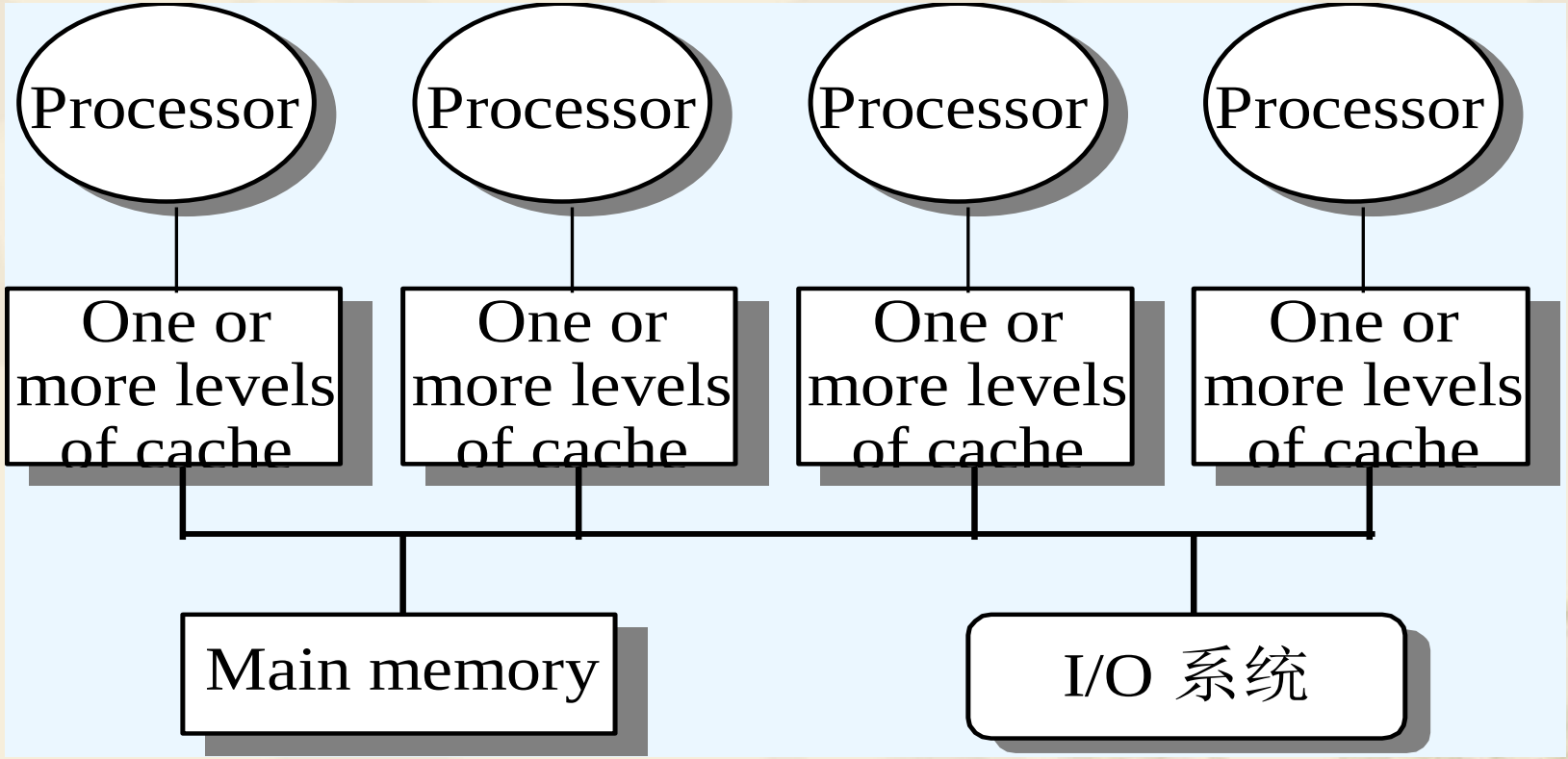
- 处理器个数影响
  - 直接影响到存储器结构
  - 互连网（节点互连）策略
  - 处理器个数的需求也将随事件发生变化。
- 根据存储器组织来形成两类：
  - 集中共享存储器式系统结构（centralized shared-memory architecture）
  - 分布存储器式系统结构（distributed-memory architecture）

一、集中共享存储器式系统结构

1. 三个特点：

- 处理器数量不多----从而所有处理器可共享一个集中式存储器，处理器和存储器通过总线互连。
- 采用大容量Cache----可使采用单一总线和单一存储器满足小数目处理器对存储器的要求。
- 每个处理器访问存储器的时间是相等的（一致的）
  - ----SMP（*symmetric (shared-memory) multiprocessors*）
  - ----UMA（uniform memory access)

2. 集中共享存储器多处理器基本结构

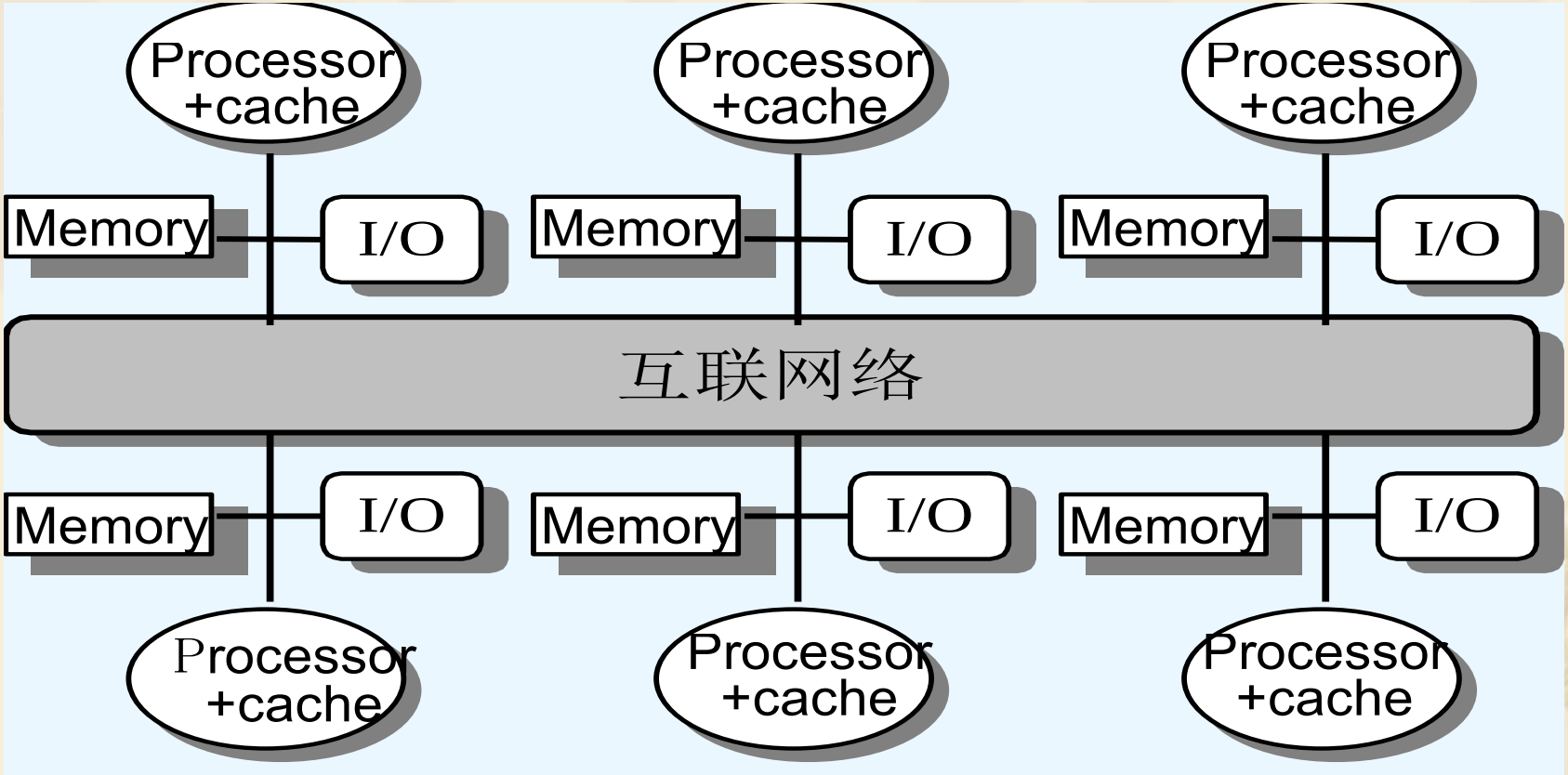


二、分布存储器体系结构

1. 三大特点：

- 处理器数目多----从而导致存储器必须为分布式的，即每一处理器配备一个存储器，否则不能满足整个系统对带宽的要求；
- 机器规模在不断缩小
  - 因为处理器性能不断提高，所以个数可减少，但与集中共享存储器型比还是要多。
  - 同时也因为处理器速度提高，对存储器带宽要求也不断提高；
- 高带宽的互连网络。

2. 分布存储器多处理器的基本结构



说明:

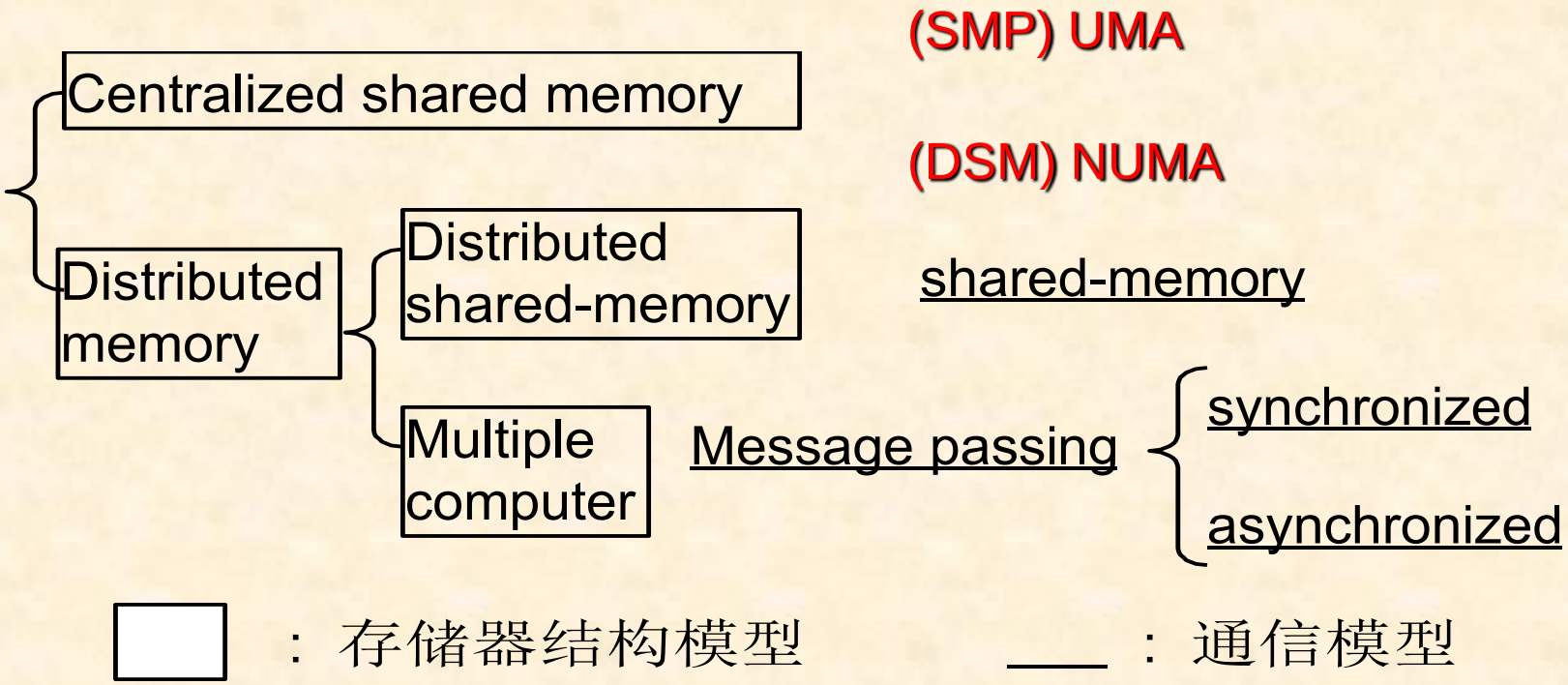
- Node may actually each contain a small number (2- 8) of processors, which may be called clustering of multiple processors.
- 为简单起见,这里指 one-processor-per node style。

三、分布存储器的优缺点

- 优点
  - **cost-effective way to scale the memory bandwidth, if most of accesses are to local memory in the node.**
  - **It reduces the latency for access to the local memory.**
- 缺点
  - 处理器之间数据通信复杂，延迟时间增加。（分布存储器导致处理器之间通信的两种范式。）

4.1.4 通信和存储器结构模型

一、多处理器体系结构的存储器结构模型及对应的通信模型



大规模多处理器必须采用分布式存储器，即每一处理器带一个存储器。

二、分布式存储器结构模型（1）

- Distributed shared memory（DSM or scalable shared memory)
  - logical **uniform address space** but **physical distributed** memory, so any one of the processors can access any one of the memories.
  - Shared memory means **sharing the address space**, which is different from centralized shared memory.
  - UMA(uniform memory access) ---- centralized share memory.
  - NUMA(non-uniform memory access) ---- distributed shared memory.

分布式存储器结构模型（2）

- multiple computers
  - **Address space consists of multiple private address spaces。逻辑上不连续，远程处理器无法访问。**
  - **每一结点（processor-memory）模块是一单独的计算机，故称为多计算机结构。**
  - **NOW计划，每一结点实质上是一工作站或PC，由LAN连接而成。**

三、通信模型

- 共享存储器通信模型（shared memory）
  - 对应与统一地址空间组织，因为这一统一的地址空间可用作隐含的数据通信机制，即直接用**load、store**共享变量即可。
- 消息传递模型（message passing）
  - 对应于多重地址空间组织。这里处理器获得数据通信需要显式地通过传递消息来进行。

四、两类message passing机制

- Synchronous message passing
  - 从读数据角度看，这里**message**可看作为**RPC**（远程过程调用），由**processor**发送请求，等到接到应答后才能继续往下进行数据获取。
- Asynchronous message passing
  - 从写数据角度来看，若生产数据的处理器知道哪一个处理器需要这一数据，一旦数据准备好，不必等请求信号到来，就将数据送往对方，这样发送过程可以立即连续不断地进行下去。
  - 标准消息传递库（**Message Passing Interface, MPI**）已制定此类库函数，通常以牺牲性能来满足接口通用性的要求。

4.1.5 通信性能的度量指标（1）

度量任何一种通信机制性能的三个指标

一、通信带宽

- 理想情况下，通信带宽受处理器、存储器和互连网络等的限制，而与通信机制无关。如中分带宽（**bisection bandwidth**）由互连网络决定。结点的近/出带宽受结点的系统结构和通信机制的影响。
- 通信机制如何影响结点的通信带宽？决定于通信机制占用结点哪些资源，占用资源时间长短，例如资源占用对通信带宽的影响与消息的长短有关。

通信性能的度量指标（2）

二、通信延时（communication latency）

- 理想情况下，延时越短越好。

Sender overhead + Time of flight + Transmission time + Receiver overhead

Transmission time + receiver overhead

- 发送端和接收端的额外开销由通信机制及其实现技术决定。
- **Latency**的重要性：决定性能好坏，决定多处理器编程的难易。
- 开销（**overhead**）与资源占用（**occupancy**）紧密相关；通信机制的很多特性，如目的地址的表示，保护机制的实现等将直接影响**overhead**和**occupancy**，直接影响到等待时间。

通信性能的度量指标（3）

三、通信延时的隐藏（communication latency hiding）

- 指通过通信机制使通信和计算或与其他处理重叠进行，把延时隐藏掉。
- 测量方法：**by measuring the running time on machines with the same communication latency but different support for latency hiding.**
- 以后再介绍**latency hiding**
- **latency hiding** 无疑是一好思想，但也给软件带来额外负担，最终将影响编程实现。所以最好的方法是降低**latency**。

4.1.6 Advantages of different communication mechanisms (1)

一、For Shared memory

- **compatibility with mechanism used in centralized multiprocessors**
- **easy programming, simplify compiler design**
- **lower overhead for communication and better use of bandwidth, due to implicit nature of communication and implement memory protection in hardware instead of in OS.**
- **The ability to use hardware-controlled caching to reduce the frequency of remote communication by supporting automatic caching of both shared and private data.**

Advantages of communication mechanism (2)

二、For Message passing

- **The hardware can be simpler**
- **communication is explicit, simpler to understand**
- **forcing programmers and compilers to pay attention to communication.**
- **….**

三、前述存储器组织与通信模型的对应关系

- 两种存储器组织
  - ■ single address pace (distributed shared memory)
  - ● private address spaces (multiple computer)
- 两种通信模型
  - □ share memory
  - ○ message passing
- 对应关系
  - □ on ■
  - ○ on ●

四、新的通信机制

- ○message passing on ■single address pace
  - 理由是传递信息实质上是把地址空间中一块数据拷贝到地址空间中的另一位置。
- □share memory on ●private address spaces
  - 在实现message passing硬件上没有统一地址空间，从而无法简单地实现shared memory方式通信，需要通过操作系统来完成地址翻译工作和存储器保护工作。新的研究方向是Virtual Shared Memory.

五、未来趋势的估计

- 传统上对分布存储结构多处理器讲，通信方式采用消息传递型，原因有：实现简单，以及很多人不相信在分布式存储结构上可实现共享地址空间。
- 最新进展：在90年代后半期，几乎每一分布存储多处理器均采用shared memory通信机制。
- 对于MPP(处理器多于100个）究竟采用何种硬件通信机制好，尚无定论，即：shared memory, message passing, hybrid approaches 均有可能。
- 虽然目前应用总线作为互连网络的集中存储结构多处理器机器主宰市场，但从长远看，技术发展趋势是中规模的分布存储结构计算机。

4.1.7 多处理器系统结构的目标和挑战

一、目标

- 并行处理----提高响应时间
  - 程序本身并行性多少
  - 网络延时
- 多任务处理----提高throughput
  - 事务处理数据库

二、并行处理的挑战

- 使并行处理成为一项富有挑战性工作的两个困难：
  - 在程序中可利用的并行性有限；
  - 通信代价过高（指延时过长）。

实例1：加速比与程序并行性的关系

- **Achieve a speedup of 80 with 100 processors, what fraction of the original computation can be sequential?**
- 解：

$$80 = \frac{1}{F_{parallel}/100 + 1 - F_{parallel}}$$

$$F_{parallel} = 0.9975$$

Thus to achieve a speedup of 80 with 100 processors, only 0.25% of original computation can be sequential.

例2 通信代价的影响

- 已知：
  - **32-processor machine, with each processor cycle time=1ns(1GHz);**
  - **remote reference time= 400ns;**
  - **all references except those involving communication hit in local memory;**
  - **base IPC=2;**
  - **0.2% of the instructions involve a remote reference.**

The multiprocessor with all local references is 1.3/0.5 = 2.6 times faster.

- 则：CPI= Base CPI + Remote request rate × Remote request cost
  - =1/Base IPC + 0.2% × Remote request cost
  - =0.5 + 0.2% × Remote request cost

The remote request cost is:

$$\frac{\text{Remote access cost}}{\text{Cycle time}} = \frac{400\text{ns}}{1\text{ns}} = 400\text{cycles}$$

CPI=0.5 + 0.8 = 1.3

三、解决上述困难的途径

- 对程序并行性：
  - 在软件中引入新的并行算法，提高并行性；
- 对通信代价：
  - **by architecture and by programmer来降低远程访问的 latency;**
    - **通过硬件机制（如caching shared data）来降低远程访问的频率；**
    - **通过软件机制，如restructuring the data to make more accesses local。**

本章的重点：降低远程latency的技术

- How caching be used to reduce remote access frequency.
  - Caching带来coherence和consistence问题。
- Synchronization（处理器间通信的同步问题）
- latency hiding techniques
- memory consistency model for shared memory。
- 实际介绍如何解决以下三大问题：
  - cache coherence
  - cache consistence
  - synchronization

4.2 Centralized Shared-Memory Architecture

- 集中共享存储结构可行性的关键问题：
  - 如何解决那么多processor访问memory的带宽问题。
- 80年代起，小规模的集中共享存储体系结构多处理器逐渐成为可行！理由是：
  - 采用大容量cache降低每个单处理器对存储器带宽的要求；
  - 微处理器技术成熟，processor可用现成的微处理器实现，而不必专门设计，性价比大为提高；
  - 小规模并行使集中共享存储器成为可能。

发展趋势

- 早期
  - 每块印制板上一个processor +cache;
- 目前
  - 每块印制板上几个（如4个）processor + caches;
- 本世纪初
  - 在每一芯片上集成几个processor
  - IBM:　2P/片

上述关键技术问题的解决方法

- 应用大Cache存放两类数据：（其它处理器不用的）private data + shared data;
- 如何理解上述**Caching private data 和shared data?**
  - **For caching private data ----single processor**的行为类似于单处理器的计算机；
  - **For caching shared data** 可采用将几个处理器共享数据部分做成几个**copies**分别存入各个**Caches,**达到降低带宽的要求。避免读取同一数据的竞争。

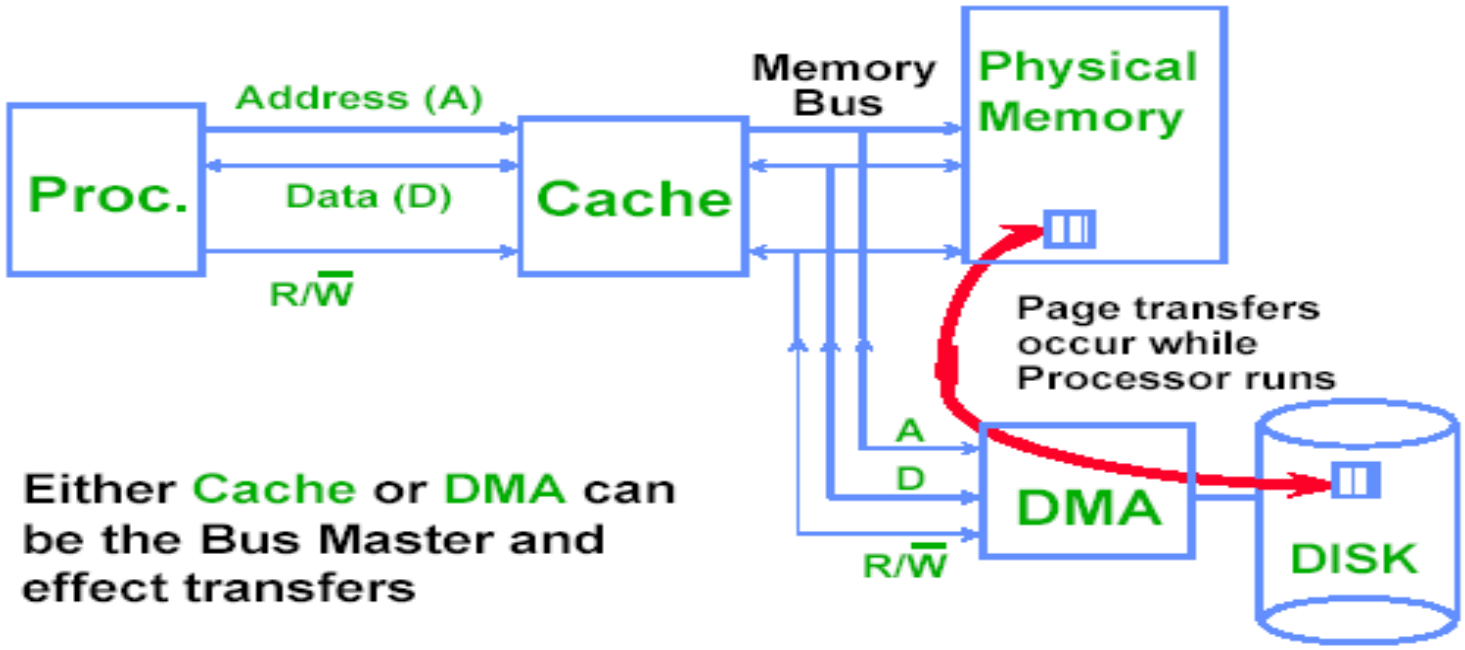
4.2.1 What Is Multiprocessor Cache Coherence?

- 新的问题：cache coherence(cache一致性）

一、I/O系统的Cache coherence问题

- 从Cache看，或从I/O系统看memory,可能存在数据的不一致性。

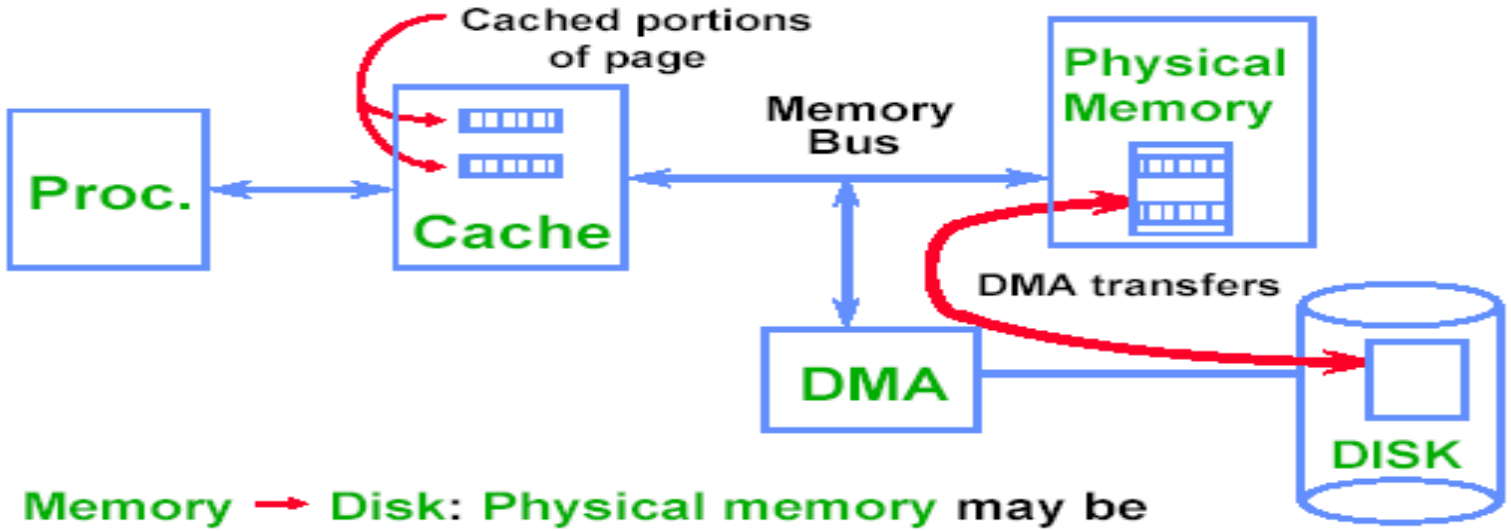
Warmup: Parallel I/O



Either Cache or DMA can be the Bus Master and effect transfers

DMA stands for Direct Memory Access

Problems with Parallel I/O



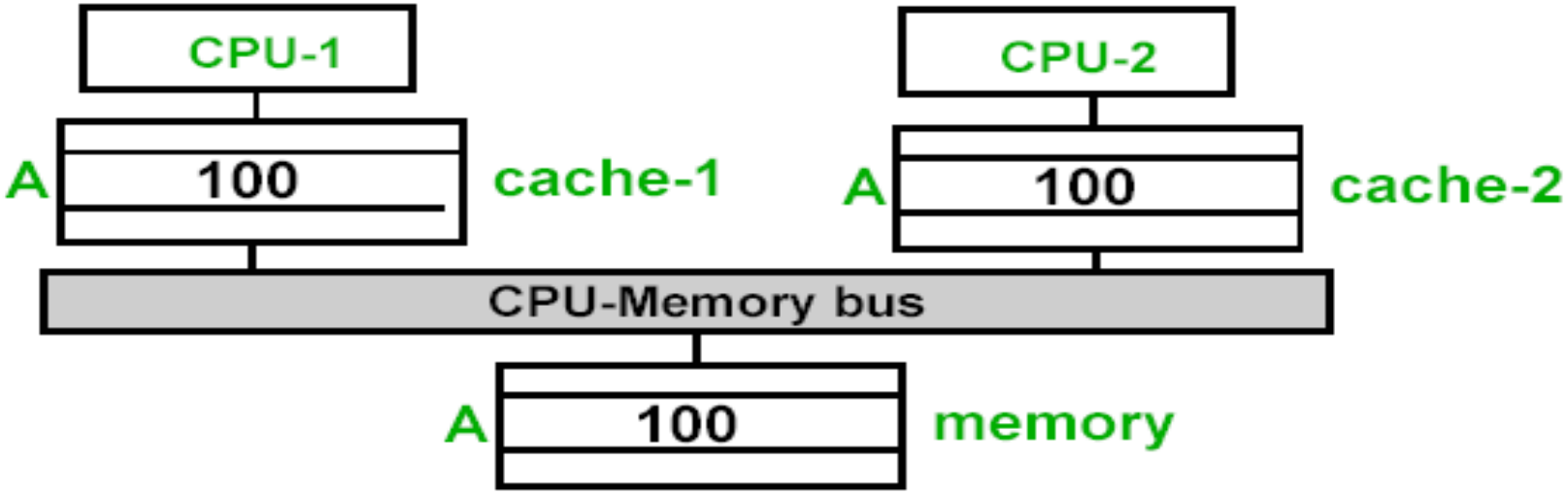
Memory → Disk: Physical memory may be stale if Cache is _____.

Disk → Memory: Cache may have data corresponding to _____.

What is Multiprocessor cache coherence?

- 多处理器系统中，两个不同处理器的**Cache 看memory,同样存在数据的不一致性。**

Memory Consistency in SMPs



Suppose CPU-1 updates A to 200.

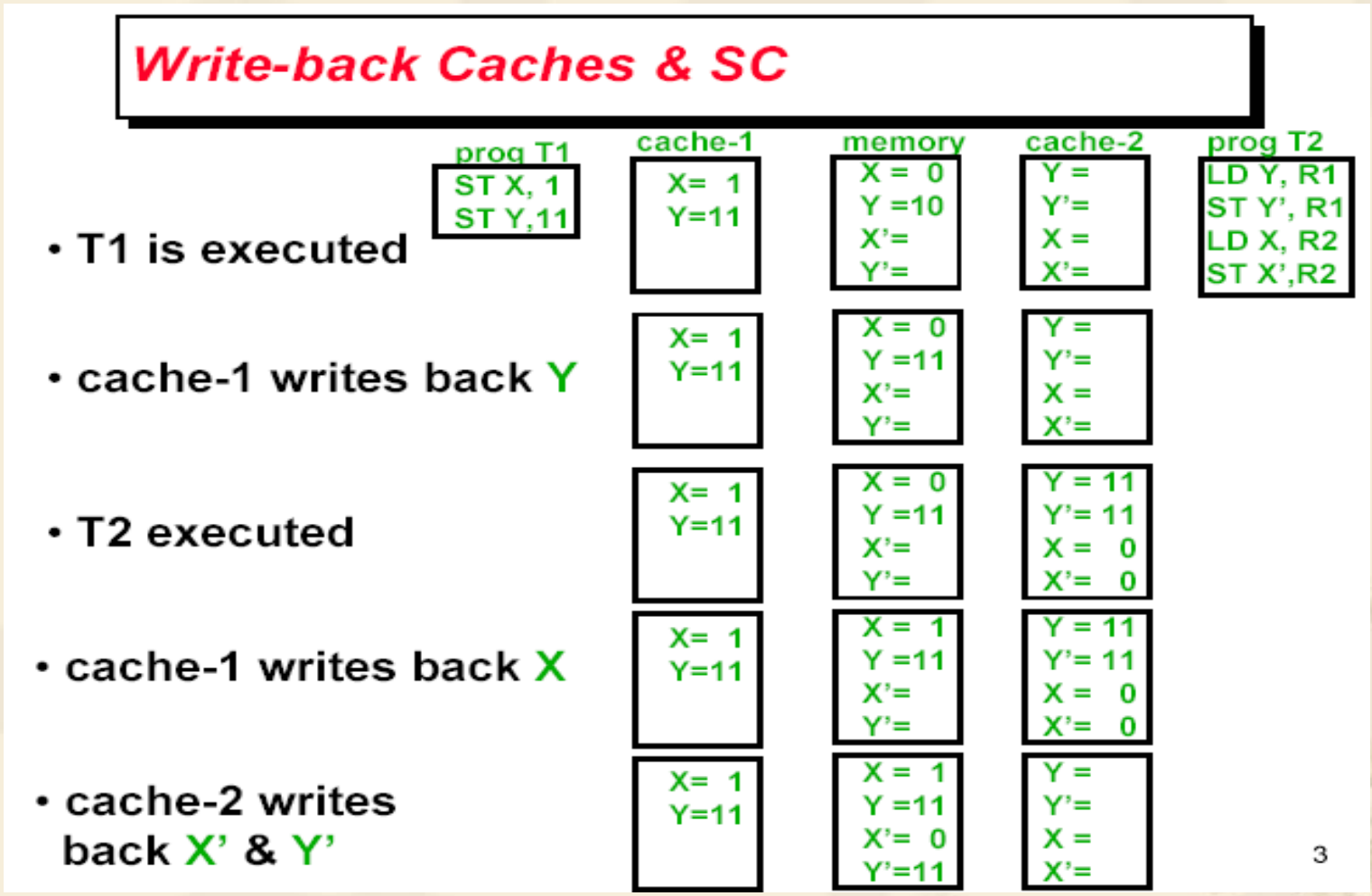
write-back: memory and cache-2 have stale values

write-through: cache-2 has a stale value

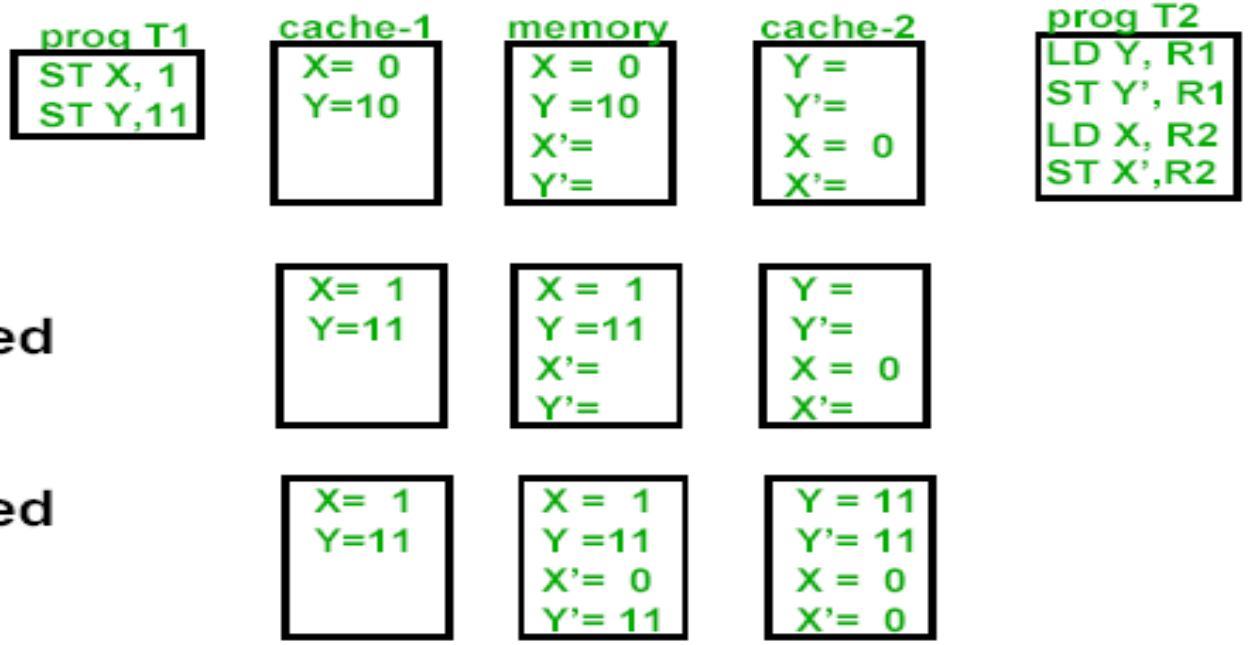
例：多处理器Cache不一致性

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

二、写操作引起Cache不一致



Write-through Caches & SC



- T1 executed
- T2 executed

三、Cache coherence定义

- A memory system **is coherent if any read of a data item returns the most recently written value of that data item.**

（每次读出的数据项的值是最近改写过的值）

上述定义的缺点

- Coherence问题
  - 指读出的是什么样的值。（强调读出值的一致性）强调值的异同，故称一致性。
- Consistency问题
  - 指改写过的值何时能正确地读出（强调何时能正确读出。强调时间概念，故称连贯性。（存储器的一致性）

四、正确的一致性定义包含3条

- A memory system is conherenct if ：
  - 同一处理器（P）对同一单元(X)先写后读，且在写读之间无其它处理器写该单元，则（P读出的）返回给P的值等于P所写入的值。----保证程序的按序性（即使单处理器也应遵守）。
  - 两个不同处理器分别对某单元(X)先写后读（一个写，一个读），若读写之间的时间间隔足够长，且无其它写操作发生在上述写读之间，则返回（读出）的值应等于写入的值。----定义了存储器一致性的含义。

正确的一致性定义(2)

- 两个不同处理器先后按序写某单元(x)（串行写），则所有其它处理器看到的是同一次序写入值。如果先写入“1”，后写入“2”，则其它处理器决不能先读出“2”，后读出“1”。----称为写操作的串行化（ write serialization)

上述三个性质足够保证memory coherence。

五、Consistency问题既重要也复杂

- 复杂性：
  - 无法要求读（X单元）操作立即得到其它处理器刚刚写入（X）的值。因为读操作离写结束时间非常短的话，不可能读出刚写入的值。
  - 何时能读出刚写入的值的问题是与memory consistency model有关的。（）

在第6节前的约定：

- 写操作在其他处理器未看到（未能读出）写入的结果之前，不能算作完成。
- 处理器不能改变任何写操作与其它存储器访问(读或写)的次序；

即可改变读次序，但写操作的次序必须严格按程序规定的顺序进行。

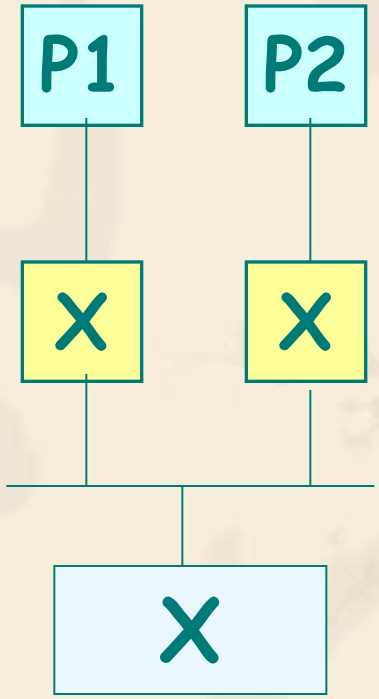
一、Cache一致性问题的原因

要解决多处理机的Cache一致性问题，首先要研究一致性问题的由来。出现不一致的原因有3个：

- 共享可写的数据
- 进程迁移
- I/O传输

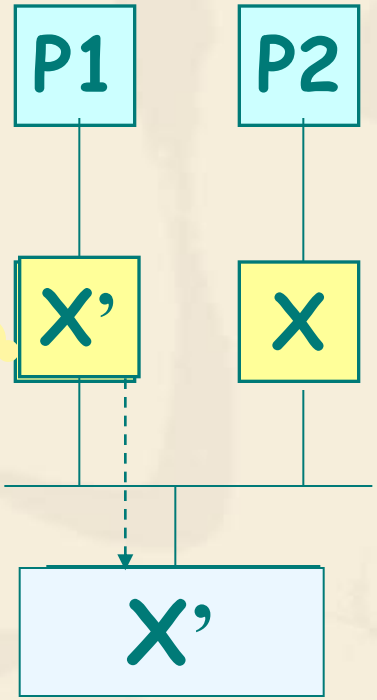
1.共享可写数据引起的不一致性

以拥有两个处理机的系统为例，处理机带有各自的私有Cache，并共享一个主存储器。



P1和P2的本地高速缓存存储器C1和C2中分别有共享主存的某个数据X的拷贝。

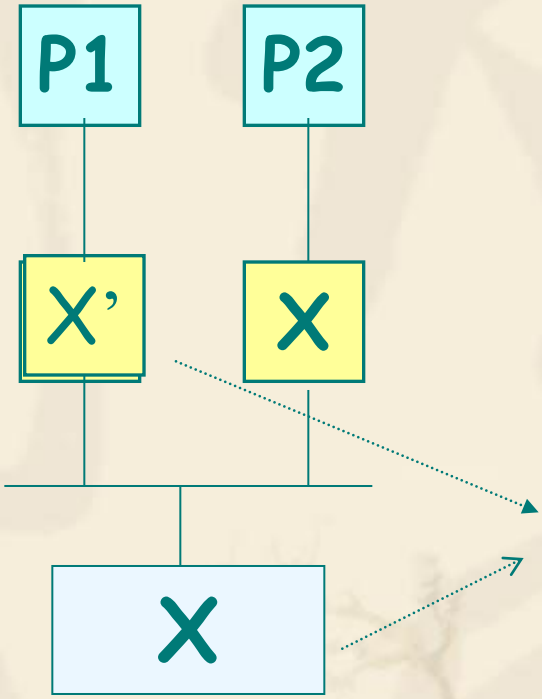
P1改写C1中的X，使之变为X’。



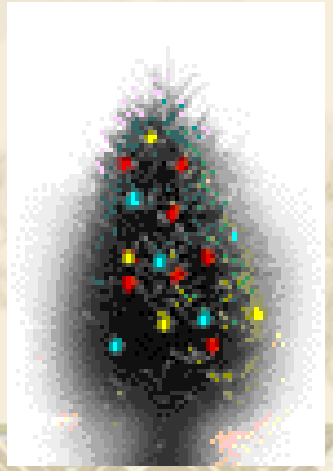
若P1采用“写通过”策略，即处理机改写Cache中的数据时同时修改内存中相应的数据，那么，内存中的X也同时变为X’，但是，处理机P2的本地高速缓冲存储器C2中的X仍然是X。

当P2要读X时，它是从C2中去读取，这就导致了P2从C2中读取的X同内存中的X’不一致。

若P1采用“写回”策略，即处理改写Cache中的数据时并不同时修改内存中相应的数据，而是在包含该数据的数据块调出Cache时才写回内存，那么，内存中的X还是X

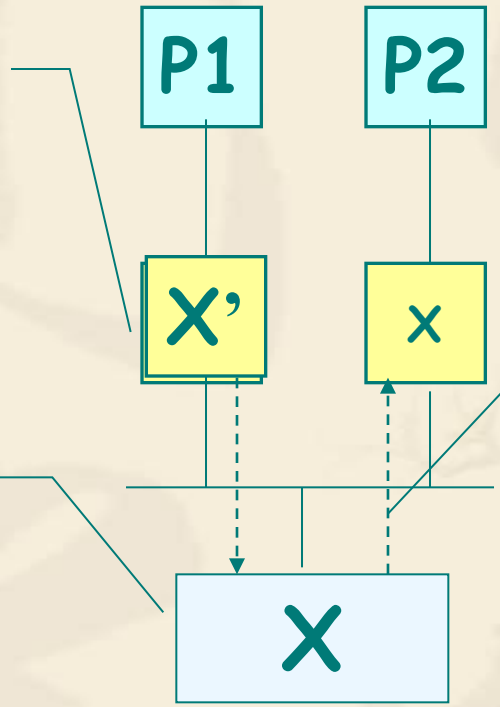


导致C1中的X’同内存中的X的不一致



2.进程迁移引起的不一致性

情况一： P1的C1中有共享数据X的拷贝，而P2的C2中没有该共享数据



若P1的进程对X进行了修改，使之变为X'

由于某种原因，该进程迁移到了P2上运行

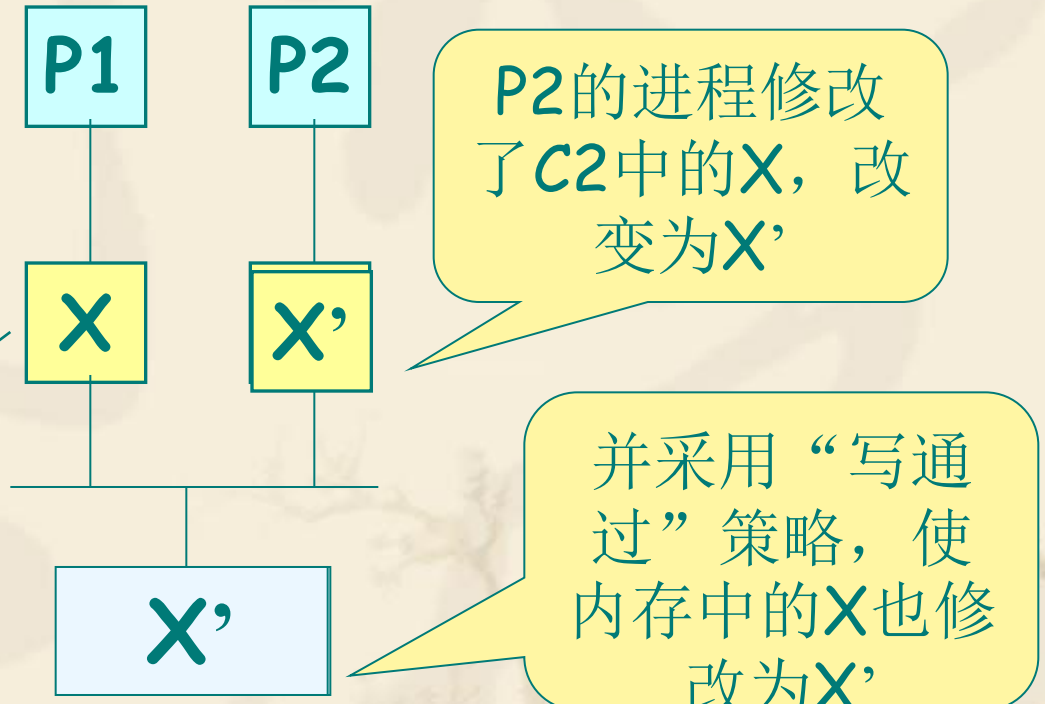
P2上的该进程运行时将从内存中读取X并将X调入C2

采用“写回”策略，暂时没有对内存中的X进行修改。

那么，这个迁移了的进程此时读取的是X，而不是它先前修改过的X'。

情况二： P1的C1和P2的C2中都有共享数据X的拷贝

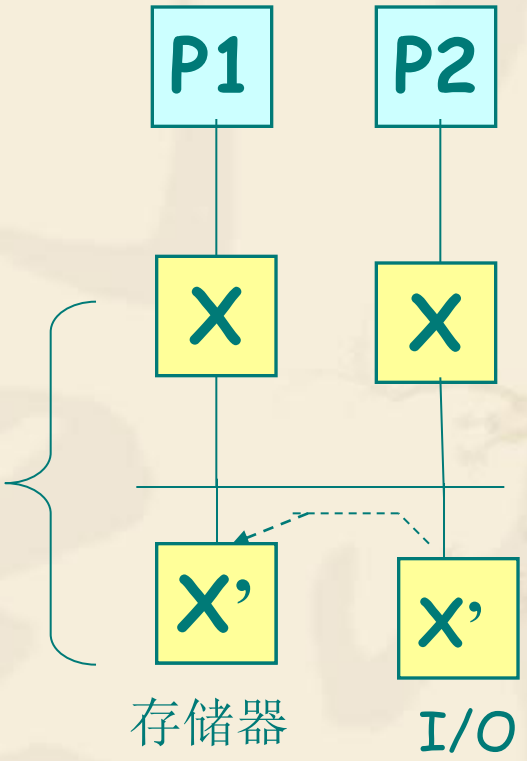
由于某种原因该进程迁移到P1上



以上两种情况都是由于进程迁移引起的数据不一致。

3. I/O传输引起的不一致性

若P1的C1和P2的C2中都有共享数据X的拷贝

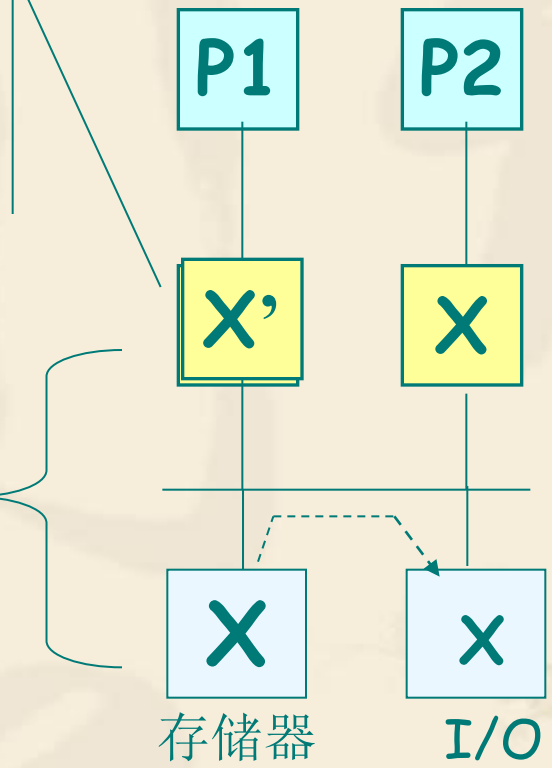


内存和Cache之间的数据不一致性。

I/O处理机将一个新的数据X'写入内存代替X

若C1和C2中都有X的拷贝

处理机P1运行过程中修改了X的值，使之变为X’



若I/O处理机要求输出X，那么，内存就会将内存中的X的值传送给I/O处理机

P1采用“写回”策略，那么，C1中的X’同内存中的X是不一致的

传送给I/O处理机的将不是修改后的X’

I/O传输引起数据不一致是因为处理机P1和P2共享I/O处理机，I/O传输发生在I/O处理机和内存之间。

一种解决I/O操作引起不一致的方法：

把I/O处理机（IOP1和IOP2）分别连接到私有高速缓存C1和C2上，使处理机和I/O处理机共享高速缓存。这样，只要能保证各Cache之间以及Cache和内存之间的数据一致性，就能够保证I/O操作不会引起不一致。