# 第 5 章　多处理机

主要内容：

- 多处理机的基本概念
- 多Cache的一致性问题
- 多处理机的并行性和性能
- 多处理机的操作系统
- 多处理机的发展

# 5.1 多处理机概述

## 5.1.1 多处理机的基本概念

### 什么是多处理机系统？

多处理机系统机是指有两台以上的处理机，共享I/O子系统，机间经共享主存或高速通信网络通信，在操作系统控制下，协同求解大而复杂问题的计算机系统。

作业、任务级并行的MIMD计算机采用资源共享途径实现并行

## 多处理机 相较于 阵列处理机

- 并行性级别：作业、任务级，更高
- 硬件结构　：多个处理器要用多个指令部件控制
- 算法实现　：进一步挖掘更多隐含的并行性
- 系统管理　：更多依靠操作系统等软件手段

## □ 多处理机 的要求

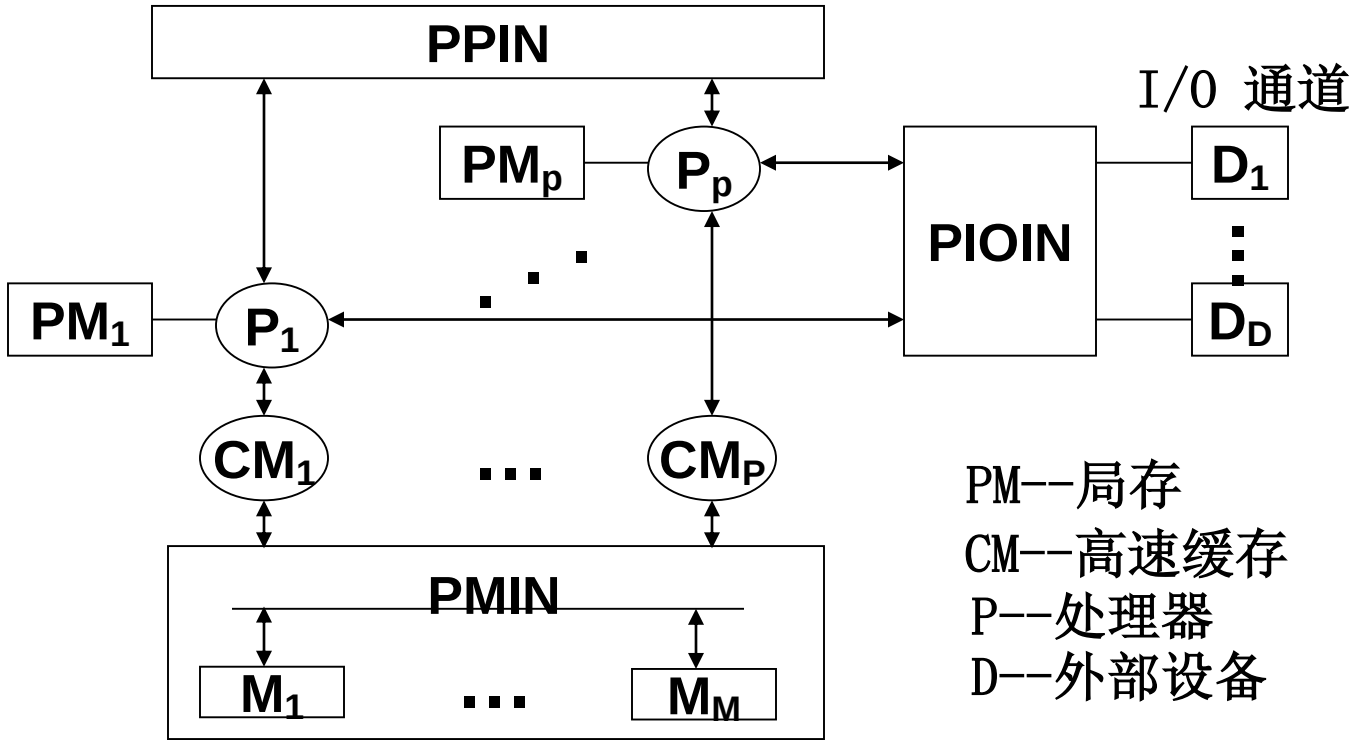
**最大限度地挖掘各种潜在的并行性**

- 硬件：保证处理机、存储器模块与I/O子系统之间的灵活多变的互连
- 软件：综合研究算法、程序语言、编译、操作系统、指令等

# 5.1.2 多处理机的硬件结构

## 1. 紧耦合与松耦合

- 紧耦合多处理机




特点：通过共享主存实现机间通讯。

互连网络：实现PE←→PEM、PE←→I/O通道、PE←→中断信号间的连接。

系统属性：

同构/异构--PE类型相同/不同；

对称/非对称—每个PE与部分/全部的I/O通道连接。

常见结构：同构对称式和异构非对称式多机系统。

限制：PE数量不能很多。为什么？

主存带宽、IN带宽、同步开销限制了PE的数量。

访存冲突解决方案：

采取多体交叉访问方式，增加PEM数量；

每个PE自带小容量局部存储器，存放核心代码、OS表格等，减少PE访存次数；

每个PE自带一个Cache，减少PE访存次数。

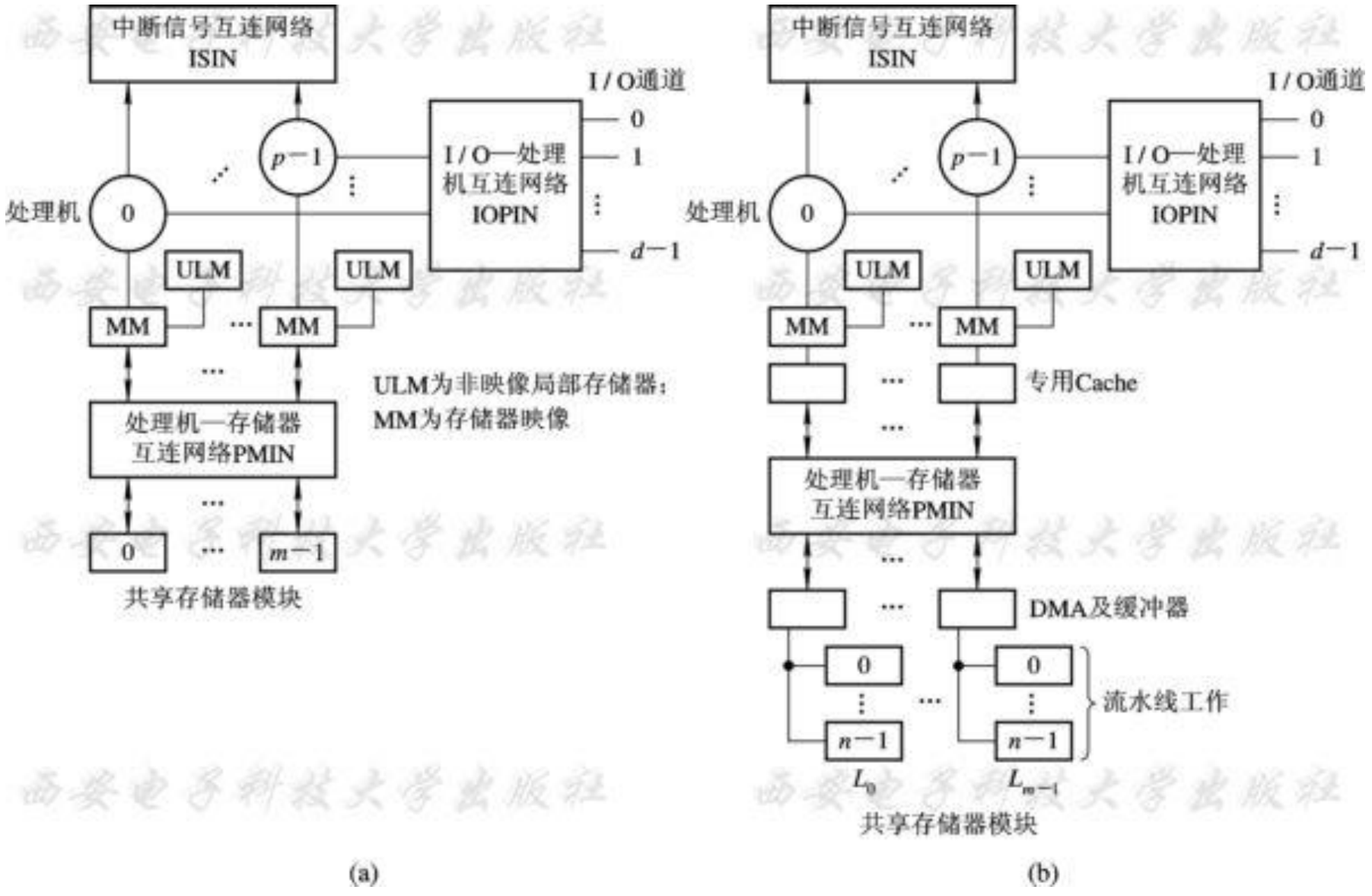



图 紧耦合多处理机的结构

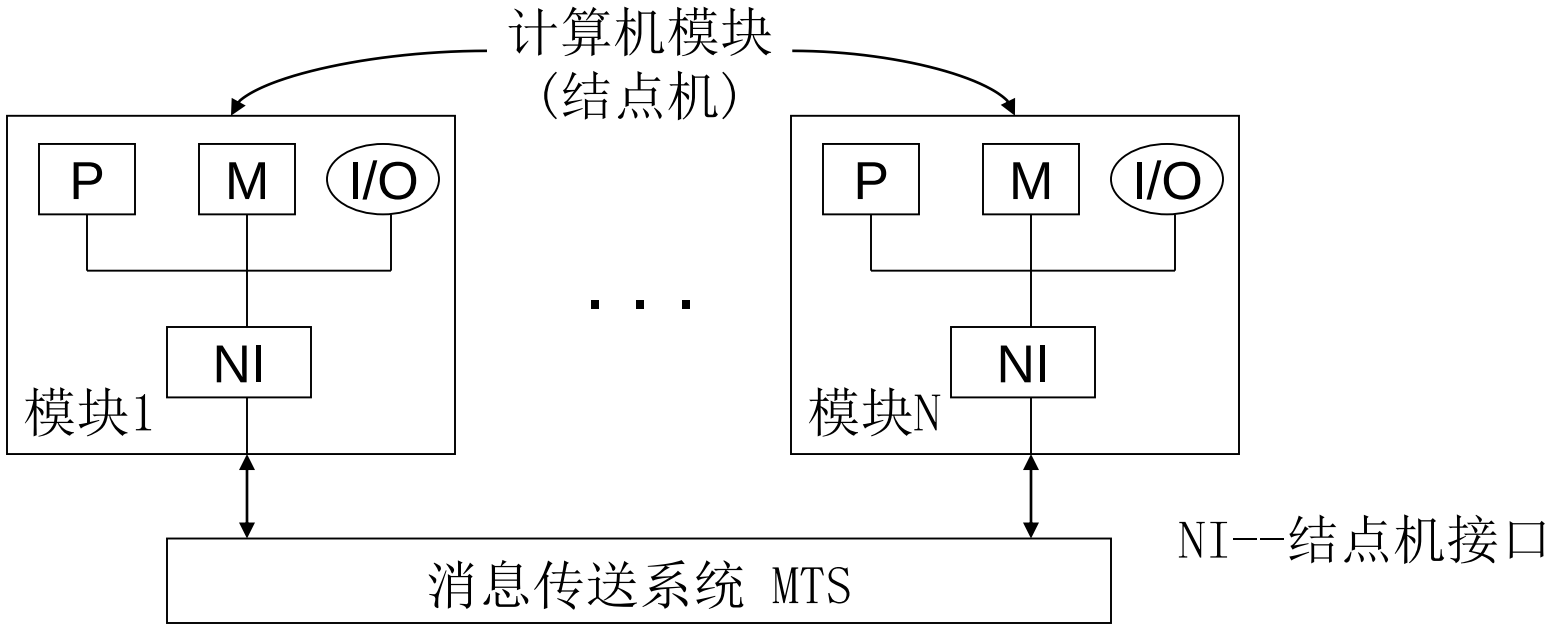
(a)处理机不带专用Cache；(b)处理机自带专用Cache

## 松耦合多处理机

每台处理机都有一个容量较大的局部存储器；不同处理机间可通过通道互连或MTS实现通信。

松耦合多处理机较适合做粗粒度的并行计算。

作业可被分为若干个相对独立的任务，任务间信息流量较少，则可在多个处理机上并行执行，即松耦合度的多处理机系统有效。




**特点：** 通过消息传送系统实现机间通讯；

每个模块是一个独立的处理机，整个系统可看成是一个分布系统。

**互连网络：** MTS有总线、环形、多级网络等种类；

**结构：** 有层次和非层次两种结构。

## 2. 机间互连形式

### □ 总线形式 （时间分配）

PE、PEM、I/O通道均连在总线上，采用分时或多路转换技术实现数据传递，是最简单的连接方式。

总线仲裁算法：静态优先级算法、平等算法、动态优先级算法、先来先服务算法等。

对外设一般采用优先级算法；对PE采用均等算法。

实现方法：

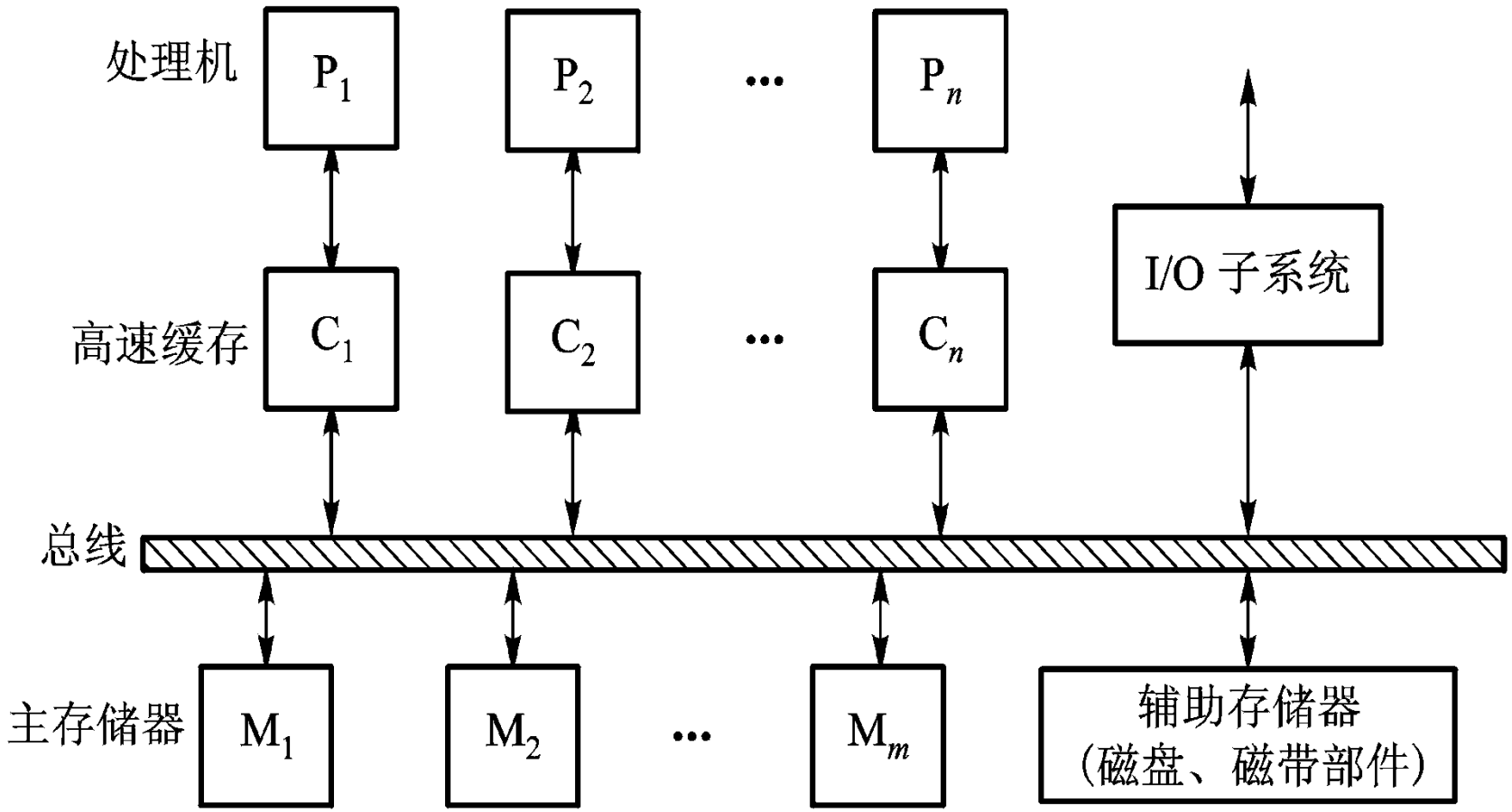
集中式：由总线控制器控制；

分布式：中机构分散到各PE中。

提高总线效率方法：改善传输介质和增加总线数量。

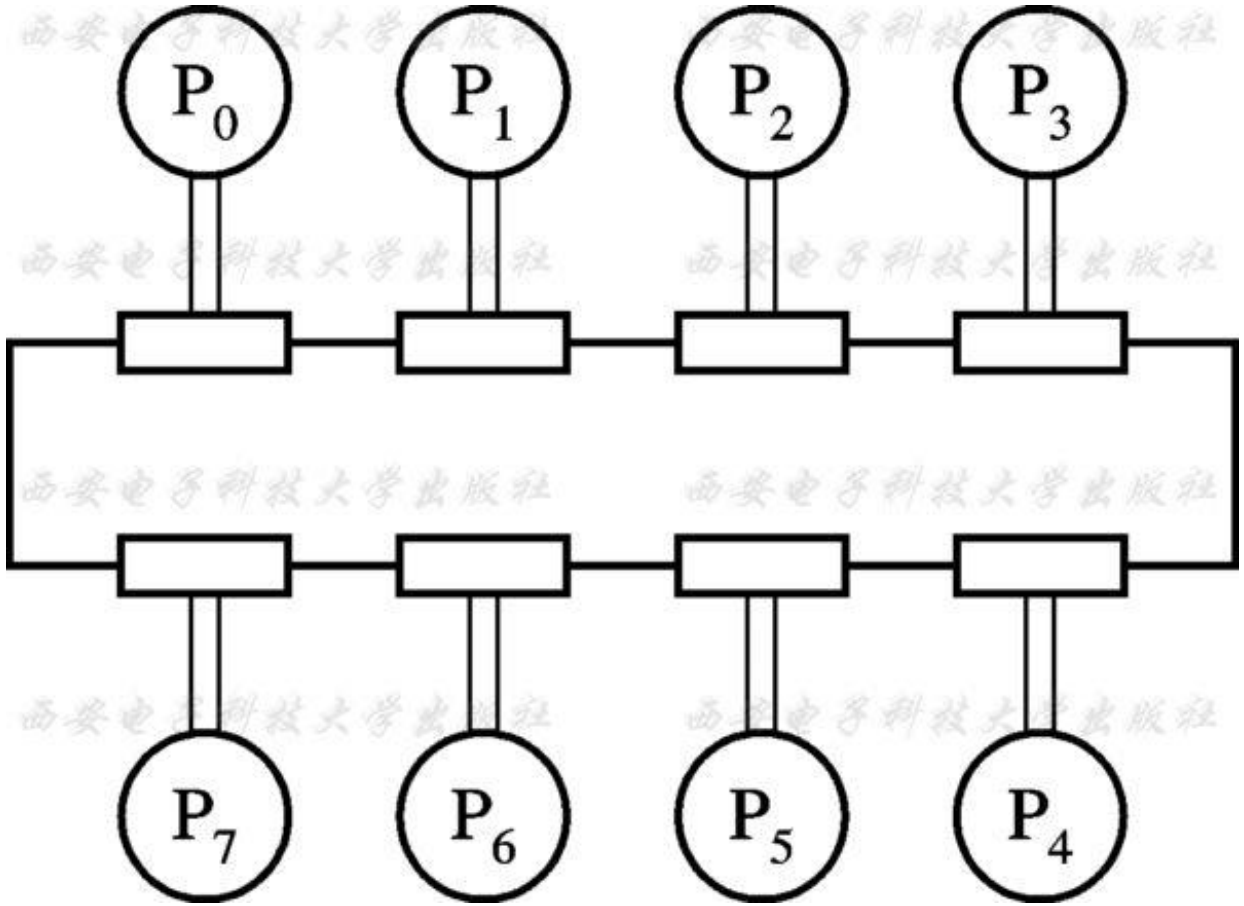
总线互连方式不适宜连接过多的处理机。

# 一种总线连接的多处理机系统



## 环形互连形式

为保持总线式互连的优点，同时又能克服其不足，可以考虑构造一种逻辑总线，让各台处理机之间点点相连成环状，称环形互连，



## 交叉开关形式　（空间分配）

是总线形式的极端，总线数=PE数+PEM数+I/O通道数，是一种全相联形式，控制、仲裁、转换机构均在开关中。

改进：用一系列较小开关串联或并联，形成多级交叉开关，减少其复杂性。

交叉开关方式不适宜连接过多的处理机。

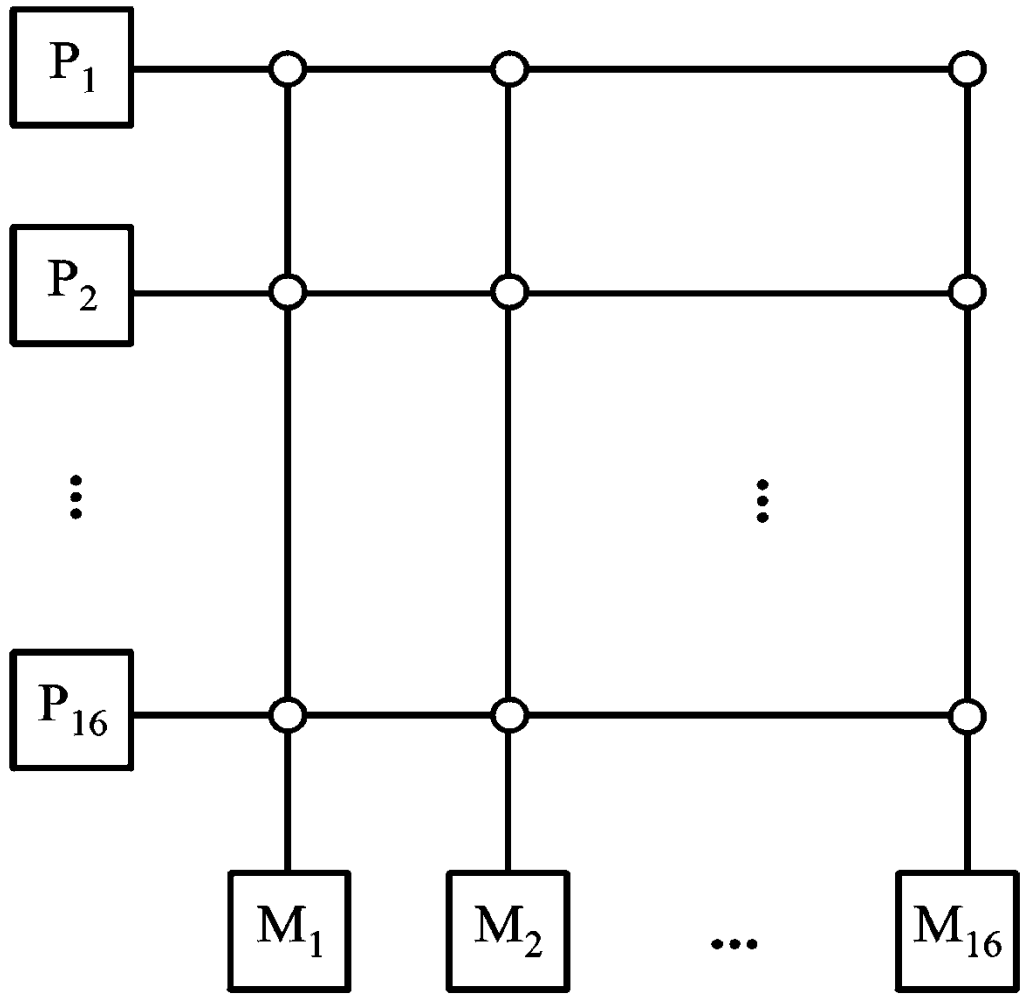
## 多端口存储器形式

将控制、仲裁、转换机构移到存储器中。

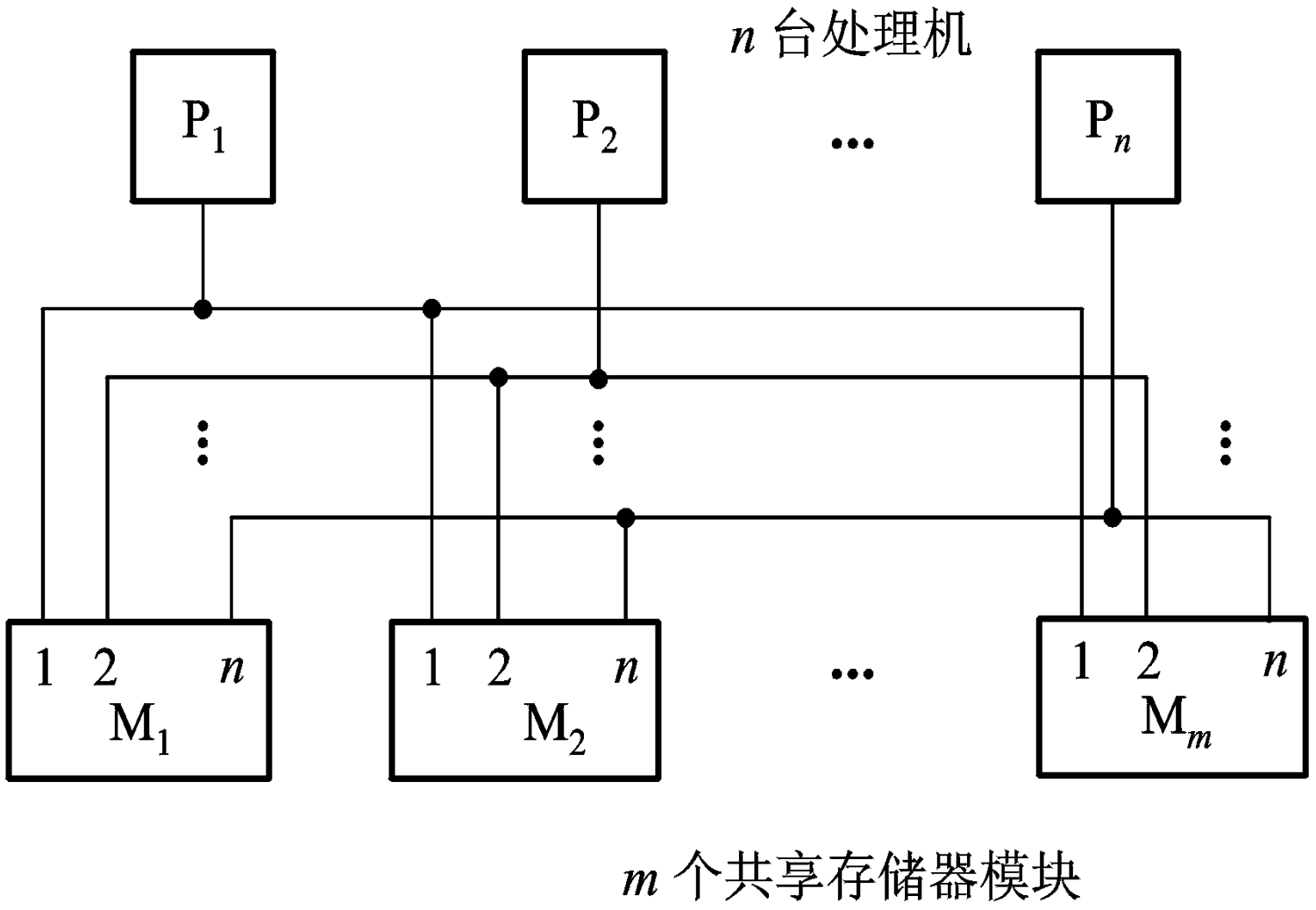
每个端口与一个PE或I/O通道相连。

多端口存储器形式不适宜连接过多的处理机。

# 一种交叉开关连接的多处理机系统



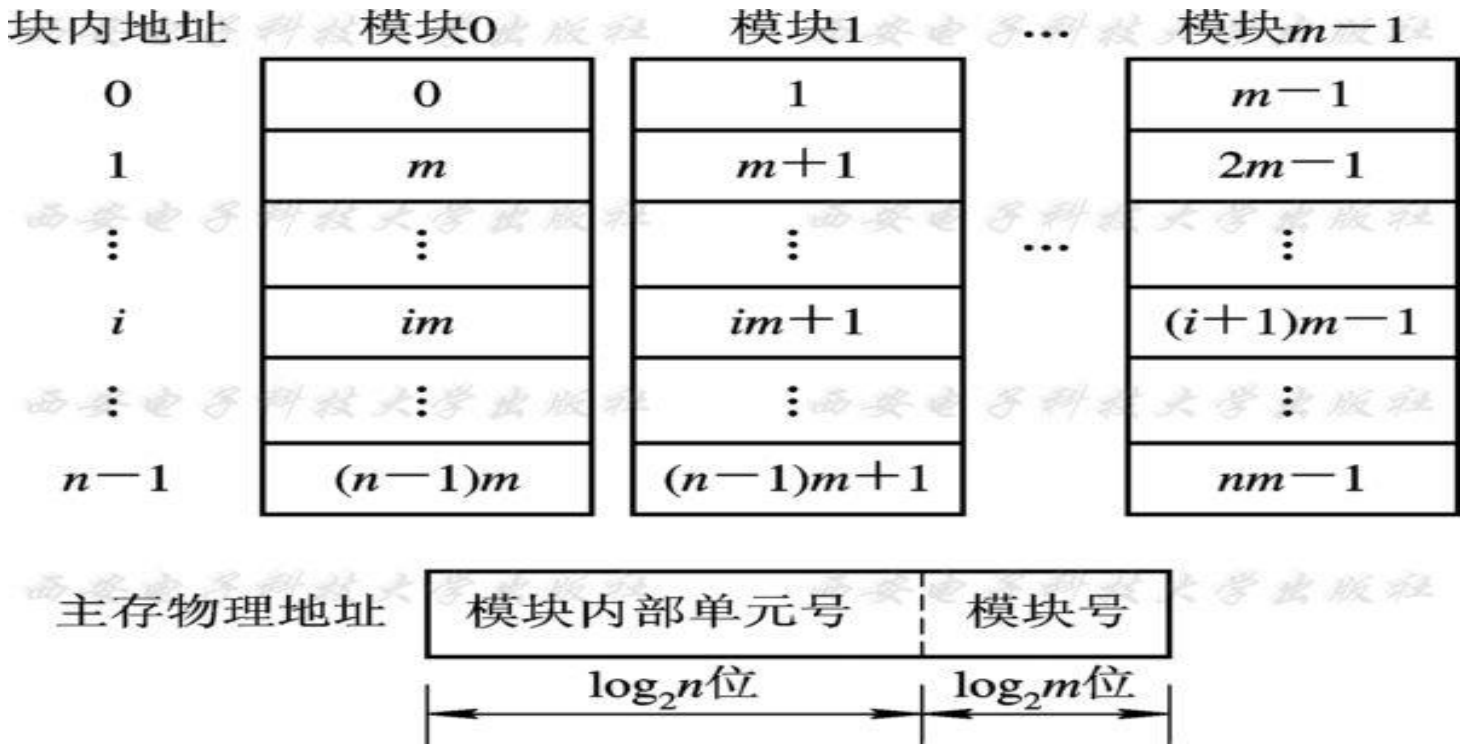
# 用于多处理机系统的多端口存储器结构



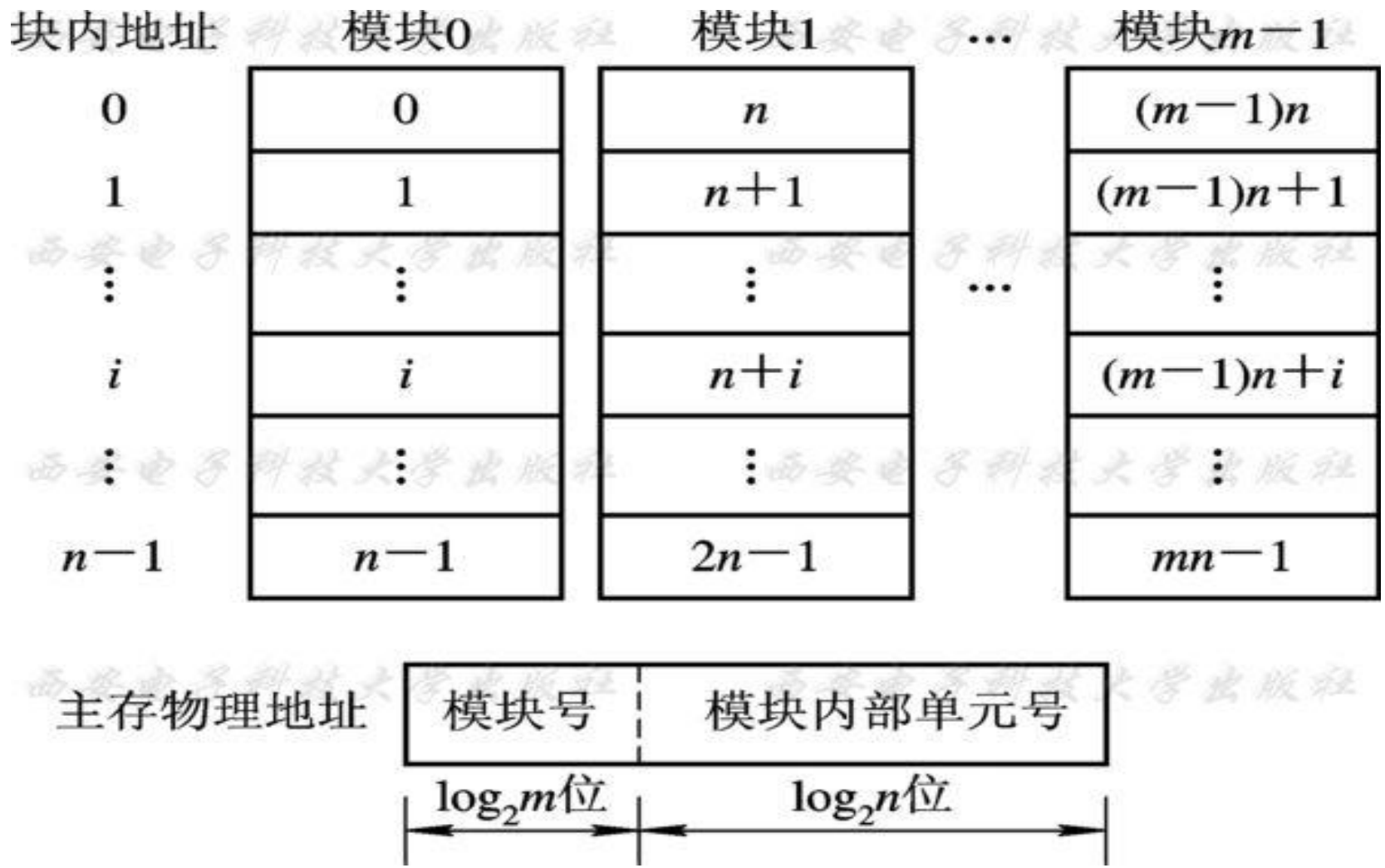
# 3. 存储器的组织

由m个存储器模块构成的并行存储器，存储单元的地址是按交叉方式编址的。这种地址交叉编址的方式主要有低位交叉和高位交叉两种。

- 低位交叉编址：块内顺序物理地址不连续，距离为m



- 高位交叉编址：块内顺序物理地址连续，总体也依次连续分布



低位交叉编址方式下，连续物理地址空间的数据被分散到各个分存储体中；

适用于当前执行进程是共享同一集中连续物理地址空间中的数据，如流水线、向量或阵列处理机

高位交叉编址方式下，一段程序或数据存放在同一个模块内；

适用于当前执行的多个进程基本不共享数据的情况，可将存储模块中的一定数量的页面分配给某进程

将放置处理机*i*执行进程要用到的绝大多数页面的那个存储器模块*i*称为是处理机*i*的本地存储器

# 本地存储器

当

- 存储模块i包含处理机i的执行进程的全部活跃页面；
- 存储模块i不包含其他处理机所需页面；

各处理器无冲突

若存储器模块数m多于处理机数时，处理机i的本地存储器可以动态地由几个存储模块组成。

- 不同处理机使用完全不同的一组存储模块
- 每个存储器模块只能被其中一个处理机访问

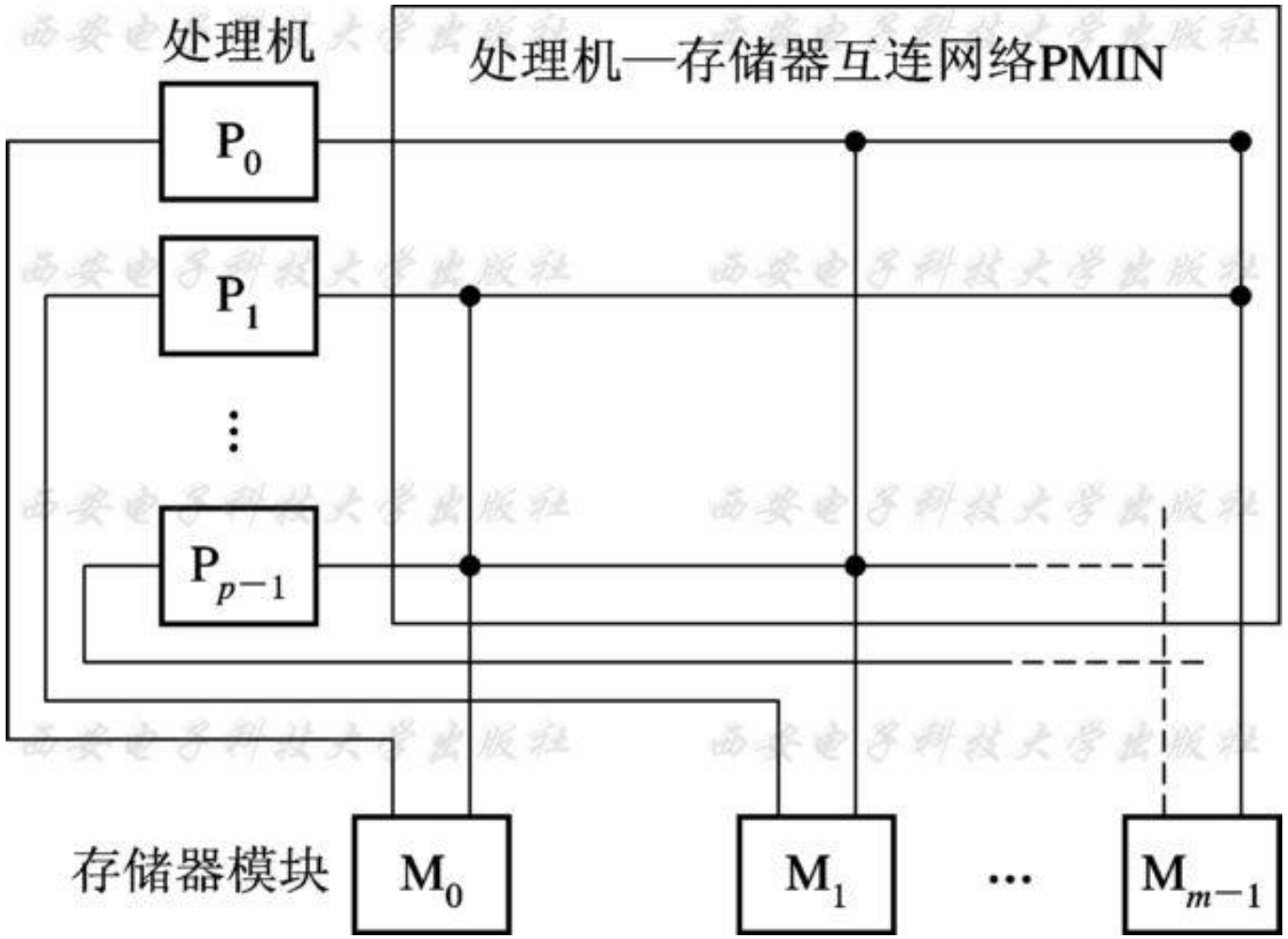



图　本地存储器的概念

# 二维并行存储器

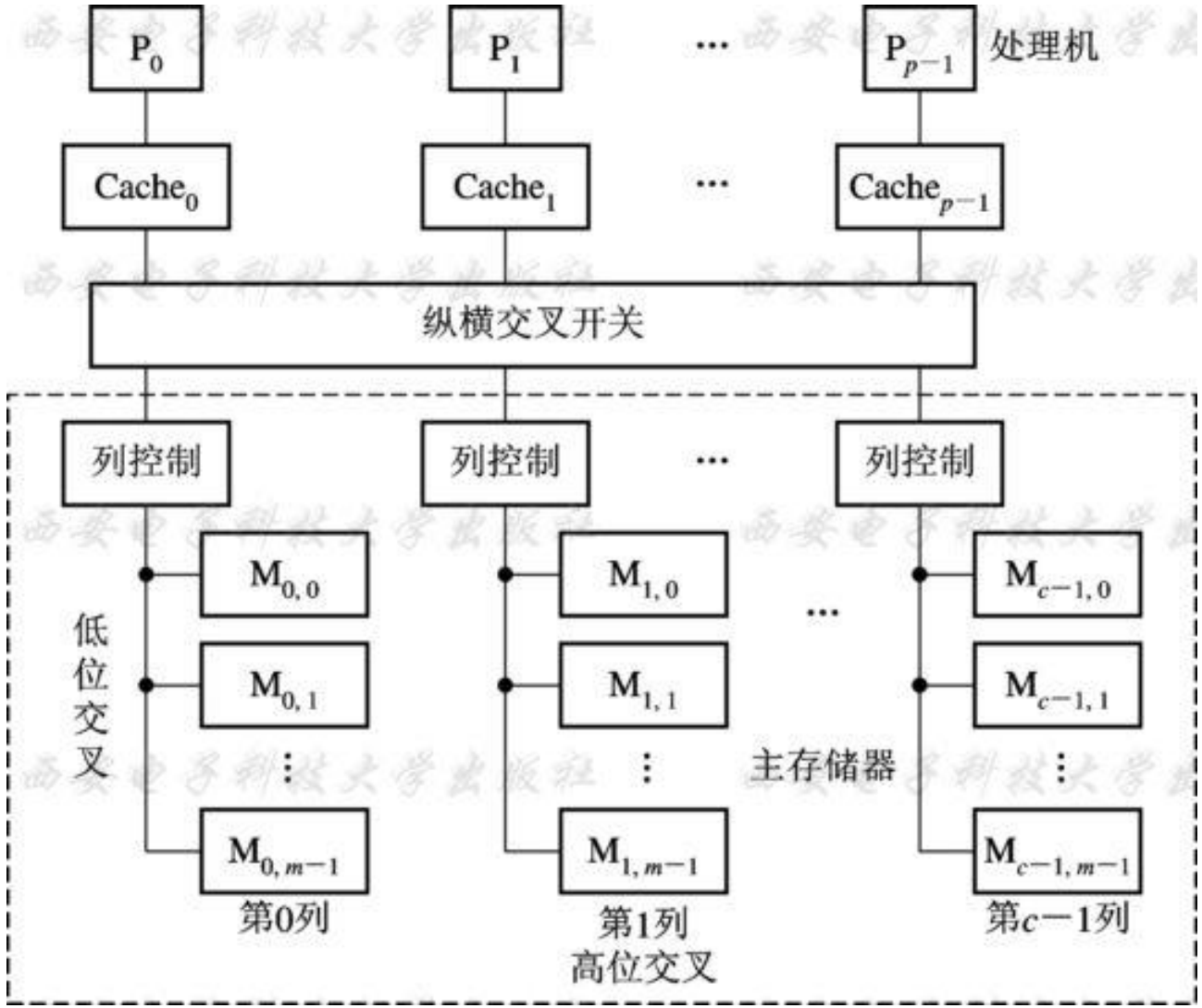



图　多处理机的二维并行存储器构形

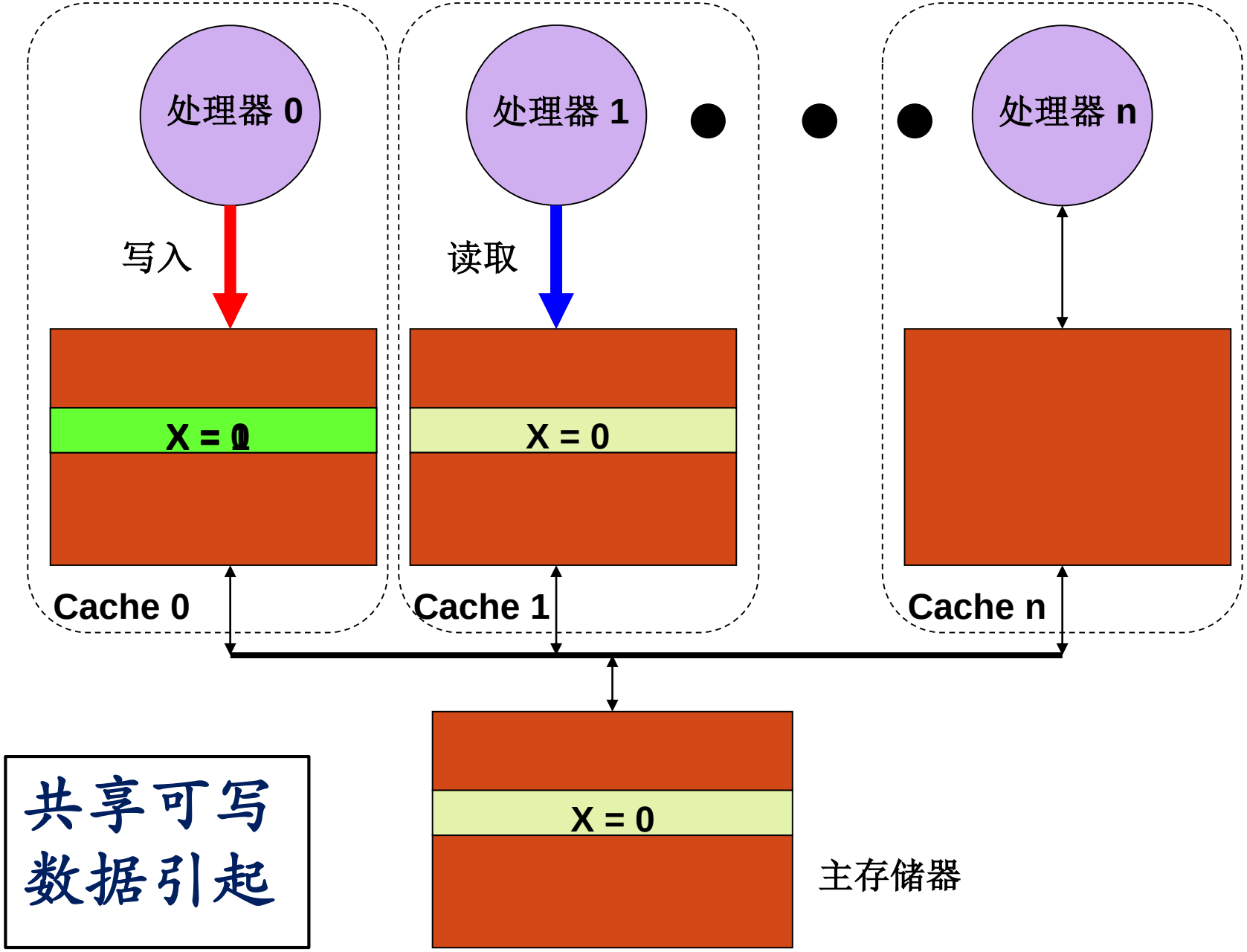
# 5.2 紧耦合多处理机
# 多Cache一致性问题

## 5.2.1 问题的产生

- 共享可写数据引起不一致性

不同处理器的Cache都保存有某些存储器单元的内容，当其中一个处理器修改了这些数据后，就可能产生数据的不一致。

**Cache的一致性问题：保证同一数据块在不同Cache以及主存中的多个副本的一致性。**

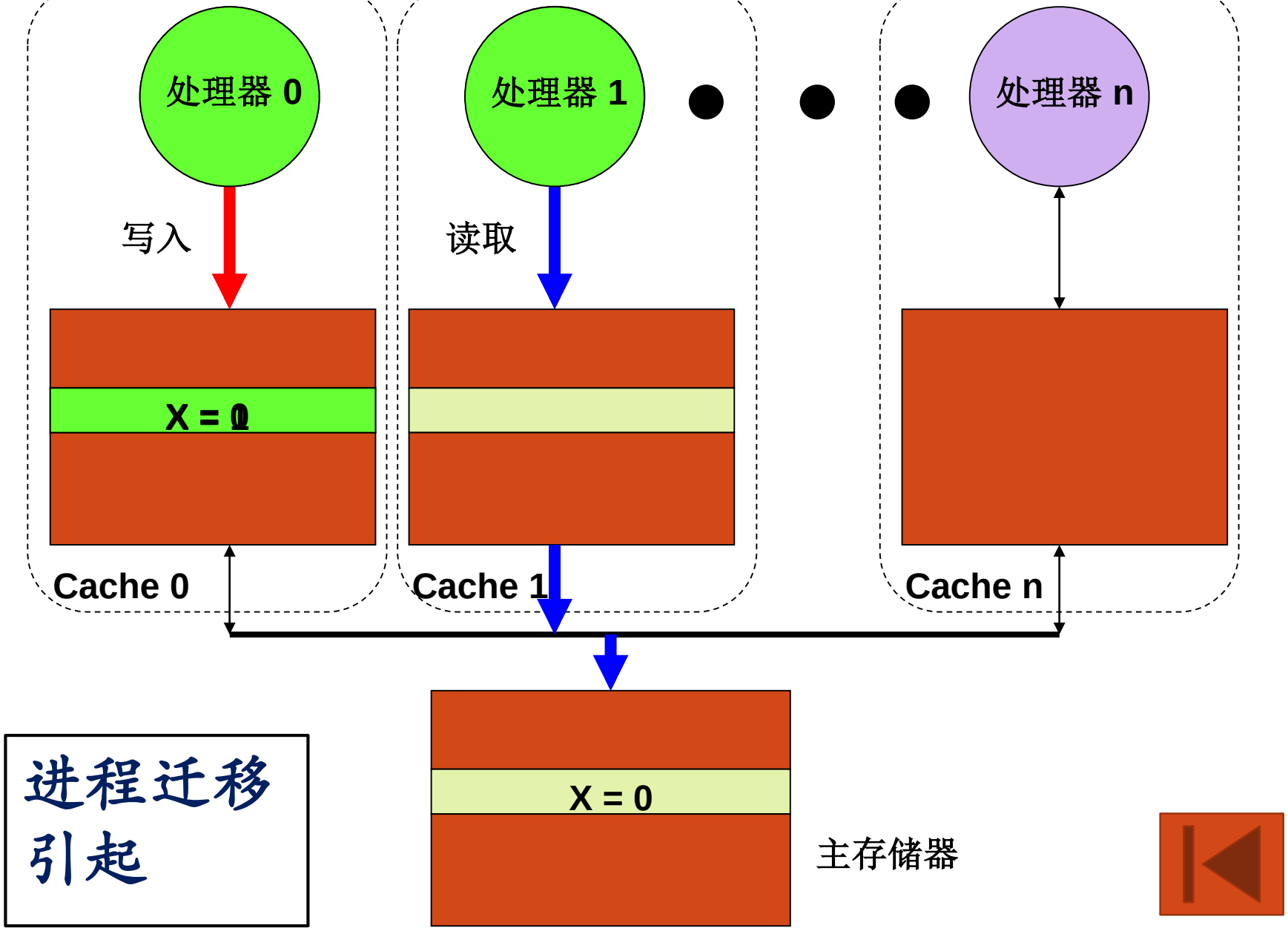

处理器 0
处理器 1
处理器 n
写入
读取
X = 1
X = 0
Cache 0
Cache 1
Cache n
X = 0
主存储器
共享可写
数据引起

- 进程迁移引起数据不一致性

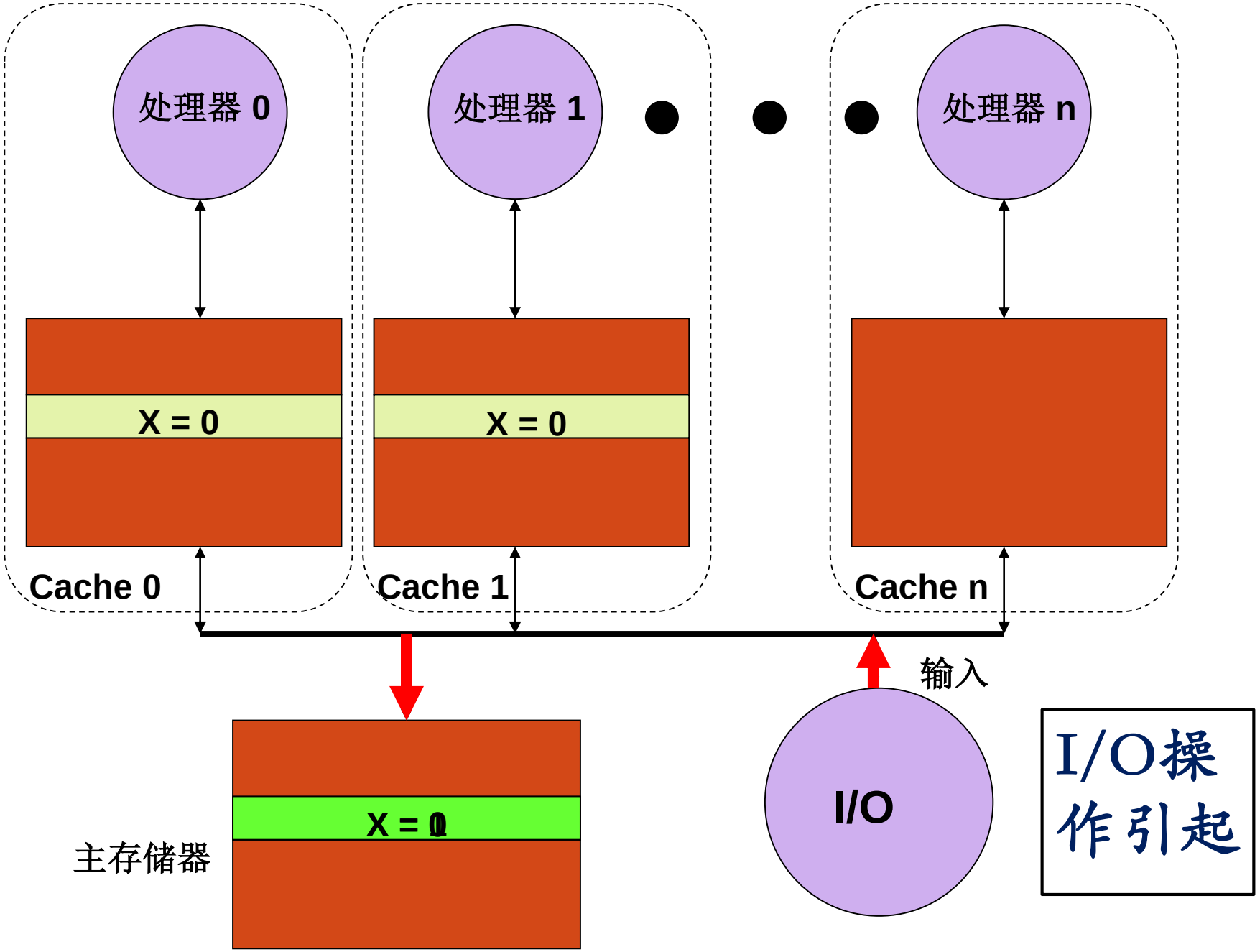
多处理机中允许将未完成的进程挂起，调度到另一处理机上执行。进程的迁移可能造成最近修改过的信息只保留在原处理机的Cache中，迁移后则不能正确恢复

- I/O传输造成的数据不一致性

系统中发生部分绕过Cache的I/O操作，可能形成主存内容与Cache内容的不一致




进程迁移引起


处理器 0
处理器 1
处理器 n
X = 0
X = 0
Cache 0
Cache 1
Cache n
输入
I/O
主存储器
I/O操作引起

如果一个存储器满足以下3点，则称该存储器是一致的。

- 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，读和写之间没有其他写，则Q读到的值应该是P写进去的值。
- 对同一单元的写是顺序化的，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顺序都是相同的。

# 5.2.2 解决的方法

## 1. 解决进程迁移引起的Cache不一致性

- 禁止迁移
- 触发写回主存

  进程挂起时，将该进程改写过的块写回主存

## 2. 以硬件为基础实现多Cache一致性

- 监视/监听法（Snoopying）
- 目录表法（Directory）

关键：跟踪记录共享数据块的状态

## 监视法

适用结构：总线型互连的多处理机

基本思想：利用总线播送更改主存的情况，各个Cache控制器通过监听总线来判断它们是否有总线上请求的数据块。

实现方式：

- 写作废协议

当一个处理器对某数据项进行写入时，通过广播使其他Cache中所有对应该数据项的副本作废。

- 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　在写回Cache、监听总线的情况下，写作废协议的实现

| 处理器行为 | 总线行为 | CPU A Cache内容 | CPU B Cache内容 | 主存单元X的内容 |
|---|---|---|---|---|
| | | | | 0 |
| CPU A 读X | | 0 | | 0 |
| CPU B 读X | | 0 | 0 | 0 |
| CPU A将1写入单元X | CacheB内容作废 | 1 | | 1 |
| CPU B 读X | | 1 | 1 | 1 |

例　在写回Cache、监听总线的情况下，写更新协议的实现。

| 处理器行为 | 总线行为 | CPU A Cache内容 | CPU B Cache内容 | 主存单元X的内容 |
|---|---|---|---|---|
| | | | | 0 |
| CPU A 读X | | 0 | | 0 |
| CPU B 读X | | 0 | 0 | 0 |
| CPU A将1写入单元X | 对单元X进行写广播 | 1 | 1 | 1 |
| CPU B 读X | | 1 | 1 | 1 |

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。

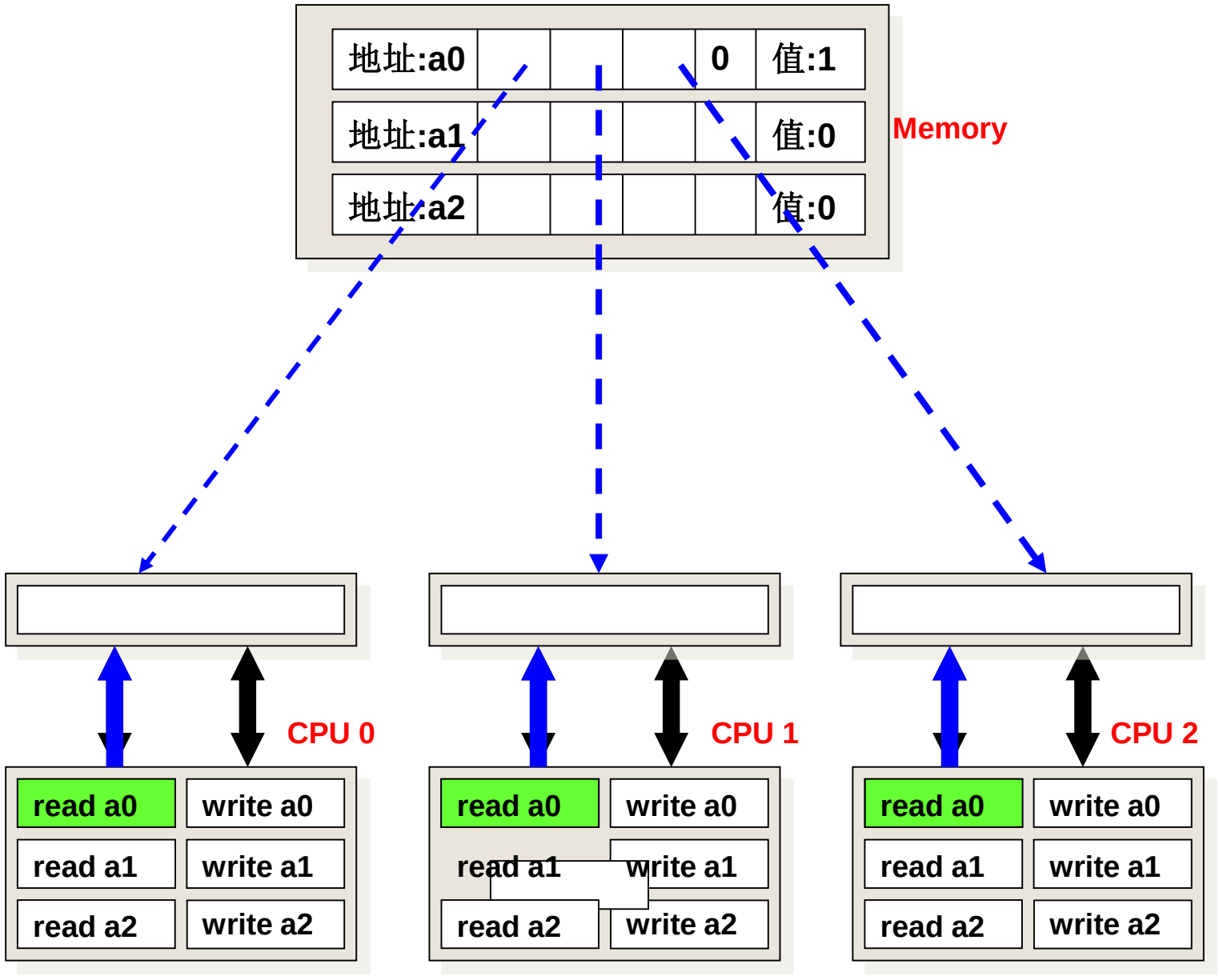
## 目录表法

目录：一种专用的数据结构，用于记录可以进入Cache的每个数据块的状态、哪些处理器有该块的副本以及是否修改过等信息。

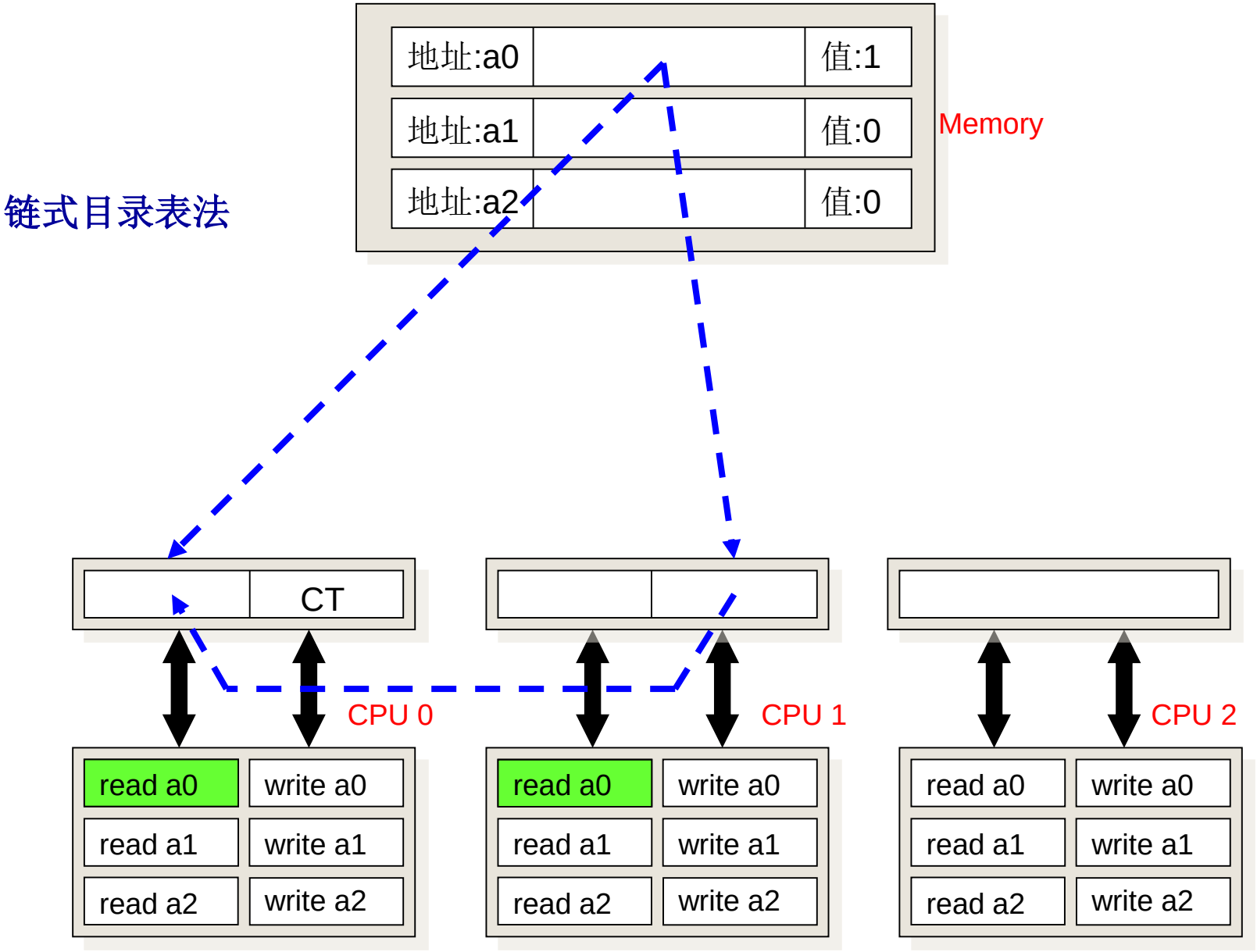
适用结构：非总线型互连的多处理机

基本思想：根据目录表，一个处理机在写入自身Cache的同时，只需有选择地通知其他存有此数据块的Cache将副本作废或更新。

- 全映像目录表
- 有限目录表法
- 链式目录表法


地址:a0
0
值:1
地址:a1
值:0
地址:a2
值:0
Memory
CPU 0
CPU 1
CPU 2
read a0
write a0
read a1
write a1
read a2
write a2
read a0
write a0
read a1
write a1
read a2
write a2
read a0
write a0
read a1
write a1
read a2
write a2

# 链式目录表法


地址:a0
值:1
地址:a1
值:0
地址:a2
值:0
Memory
CT
CPU 0
CPU 1
CPU 2
read a0
write a0
read a1
write a1
read a2
write a2
read a0
write a0
read a1
write a1
read a2
write a2
read a0
write a0
read a1
write a1
read a2
write a2

# 3. 以软件为基础实现多Cache一致性

基本思想：不允许要共享的可写数据进入Cache

- 任意时刻均不允许共享的可写数据进入Cache，只留在主存中
- 通过编译分析后，只在实际有写入操作会影响一致性的时间内不允许进入主存

# 5.3 多处理机的并行性和性能

## 5.3.1 并行算法

### 1. 并行算法的定义和分类

并行算法是指可同时执行的多个进程的集合，各进程可相互作用、协调和并发操作。

- 按并行进程间的操作顺序不同，并行算法又分同步型、异步型和独立型三种。

同步型：各进程间由于相关，必须顺次等待。

异步型：各进程间执行相互独立，根据执行情况决定中止或继续。

独立型：各进程间完全独立，不需要相互通信。

- 根据各处理机计算任务的大小(即任务粒度)不同，并行算法又分细粒度、中粒度和粗粒度三种。

细粒度：向量或循环级的并行

中粒度：较大的循环级并行，并确保这种并行的好处可以补偿因并行带来的额外开销。

粗粒度：子任务级的并行。

此外，用同构性来表示并行的各进程间的相似度。

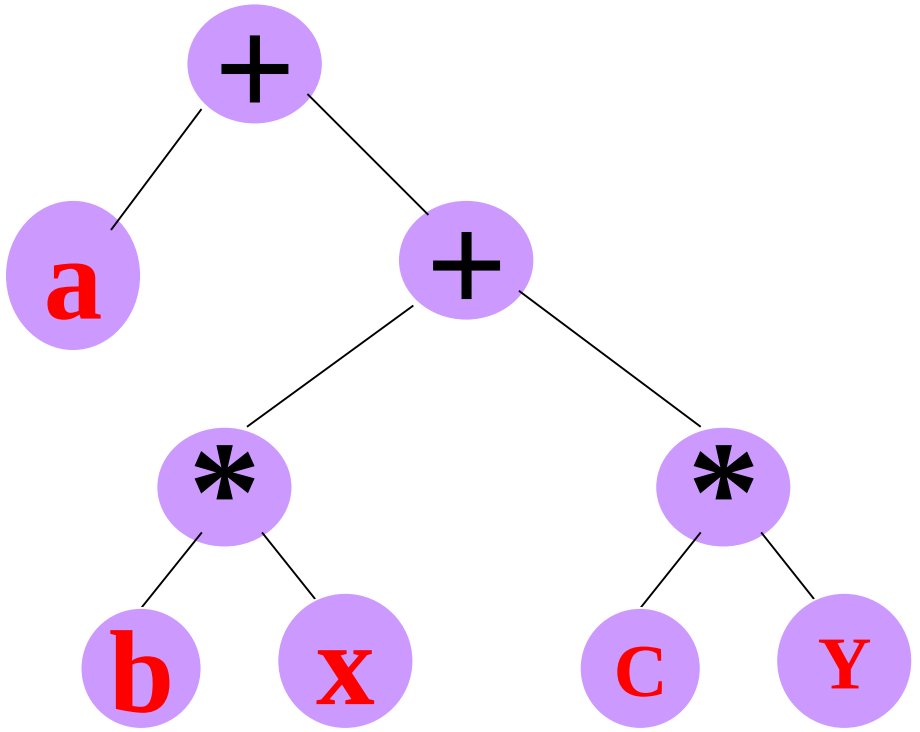
MIMD系统上运行的进程：异构性

SIMD系统上运行的进程：同构性

# 2. 多处理机并行算法的研究思路

- 将大程序分解为若干可并行处理的过程。
- 把每个过程看做一个节点，将过程间的关系用节点组成的树来表示。

$E_1=a+b*x+c*Y$



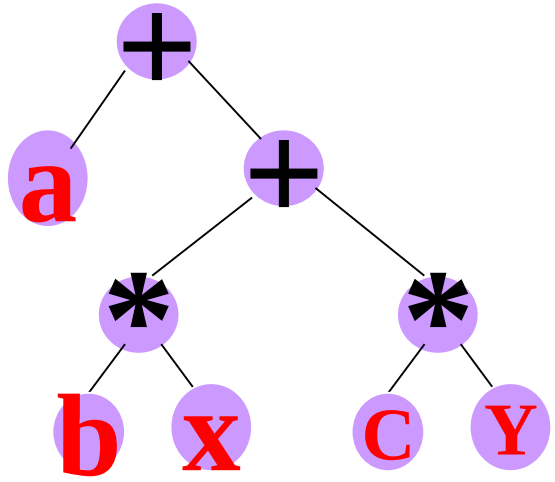
## 评价并行算法的指标：

- P：表示可并行处理的处理机机数；
- $T_P$：表示*P*台处理机运算的级数即树高
- $S_P$：加速比，表示单处理机顺序运算的级数$T_1$与P台处理机并行运算的级数$T_P$之比；
- $E_P$：P台处理机的设备利用率(效率)， $E_P = S_P/P$

树的设计目标：

——又“矮”又“瘦”

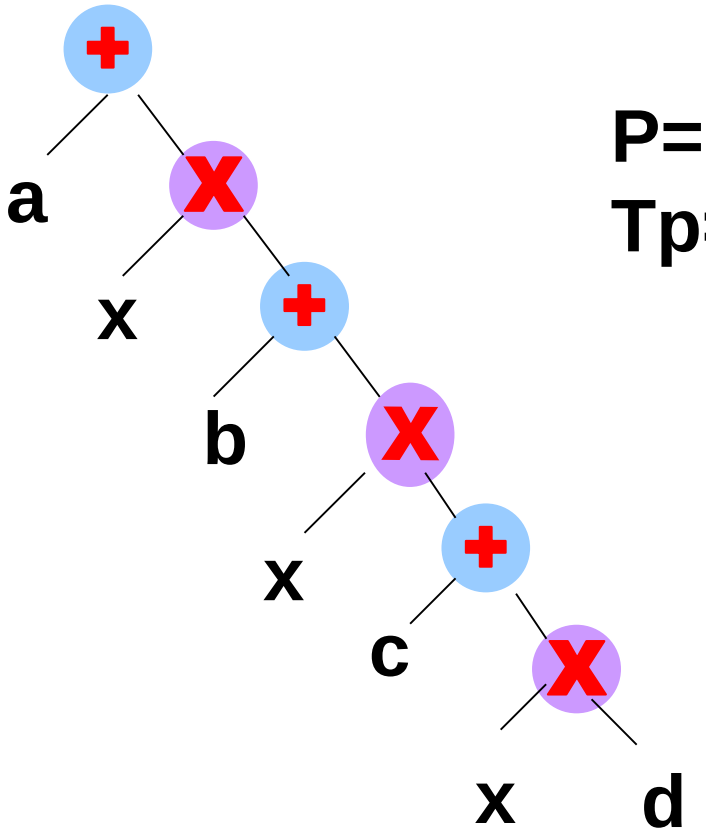
算法速度快，使用资源少



$$E_1=a+b\cdot x+c\cdot x\cdot x+d\cdot x\cdot x\cdot x$$

若在单处理机上执行，可调整为：

$$E_1=a+x(b+x(c+x(d)))$$




P=1

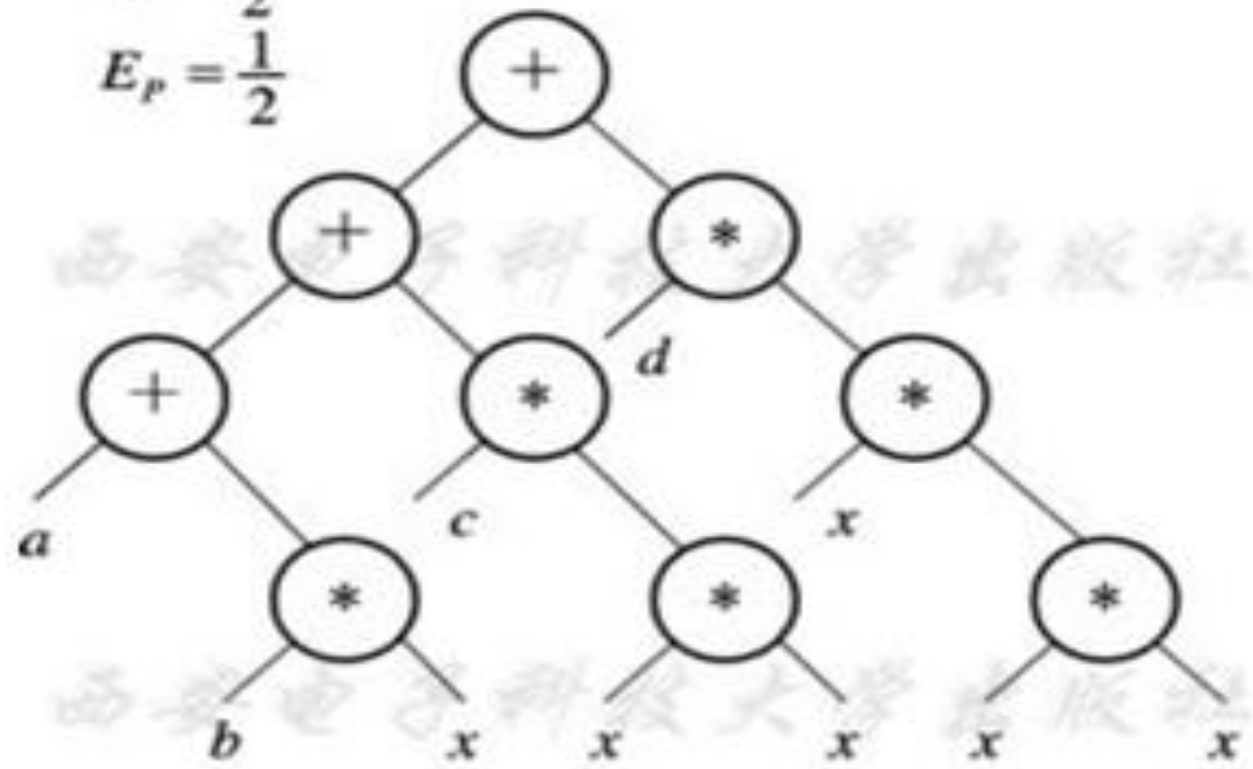
Tp=6

若按此式放到3处理机系统上执行，效果如何？

# $E_1=a+b\cdot x+c\cdot x\cdot x+d\cdot x\cdot x\cdot x$

$P=3$
$T_P=4$
$S_P=\frac{3}{2}$
$E_P=\frac{1}{2}$

若在3处理机系统上执行



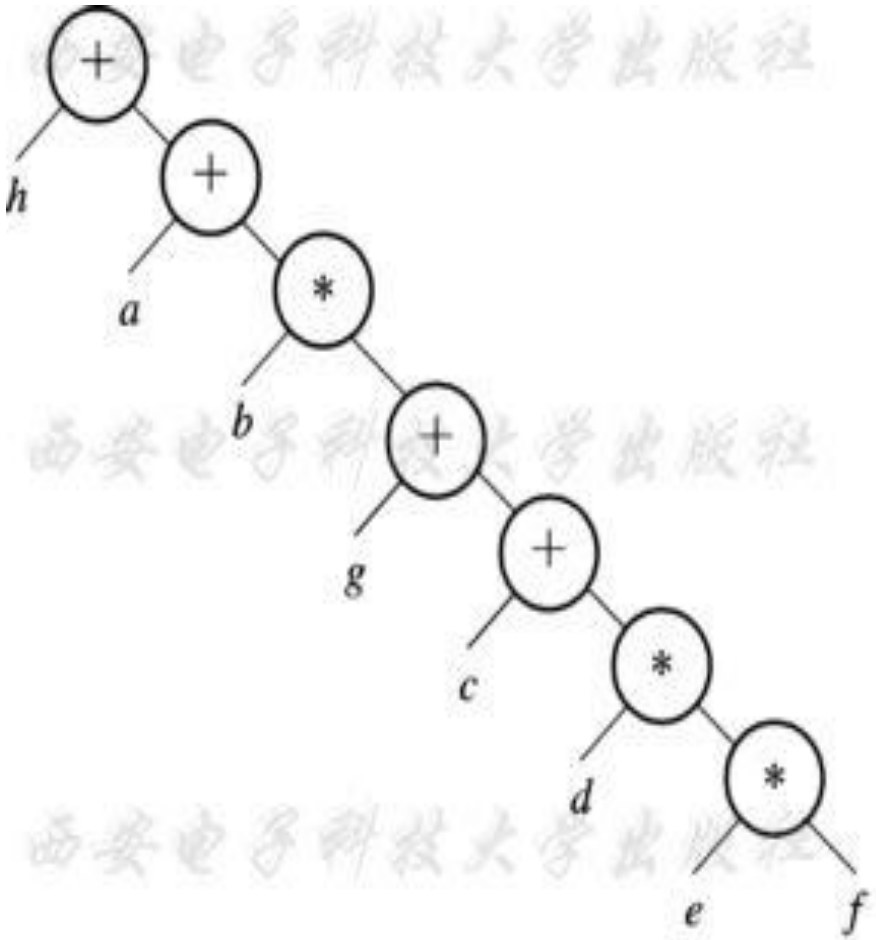
通常先从算术表达式的最直接形式出发，利用交换律把相同的运算集中在一起。然后再利用结合律把参加这些运算的操作数(称原子)配对，尽可能并行运算，从而组成树高为最小的子树。最后再把这些子树结合起来。

注意：分析树的构造与给定的处理机个数密切相关

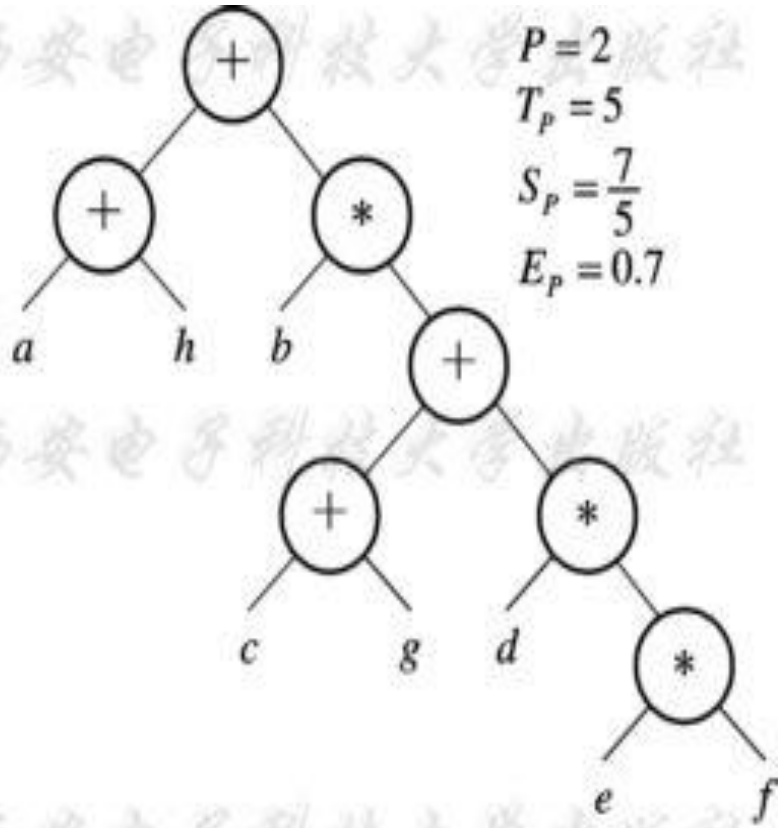
例　$E_2=a+b(c+def+g)+h$

若在2处理机系统上执行

$E_2=(a+h)+b((c+g)+def)$




(a)

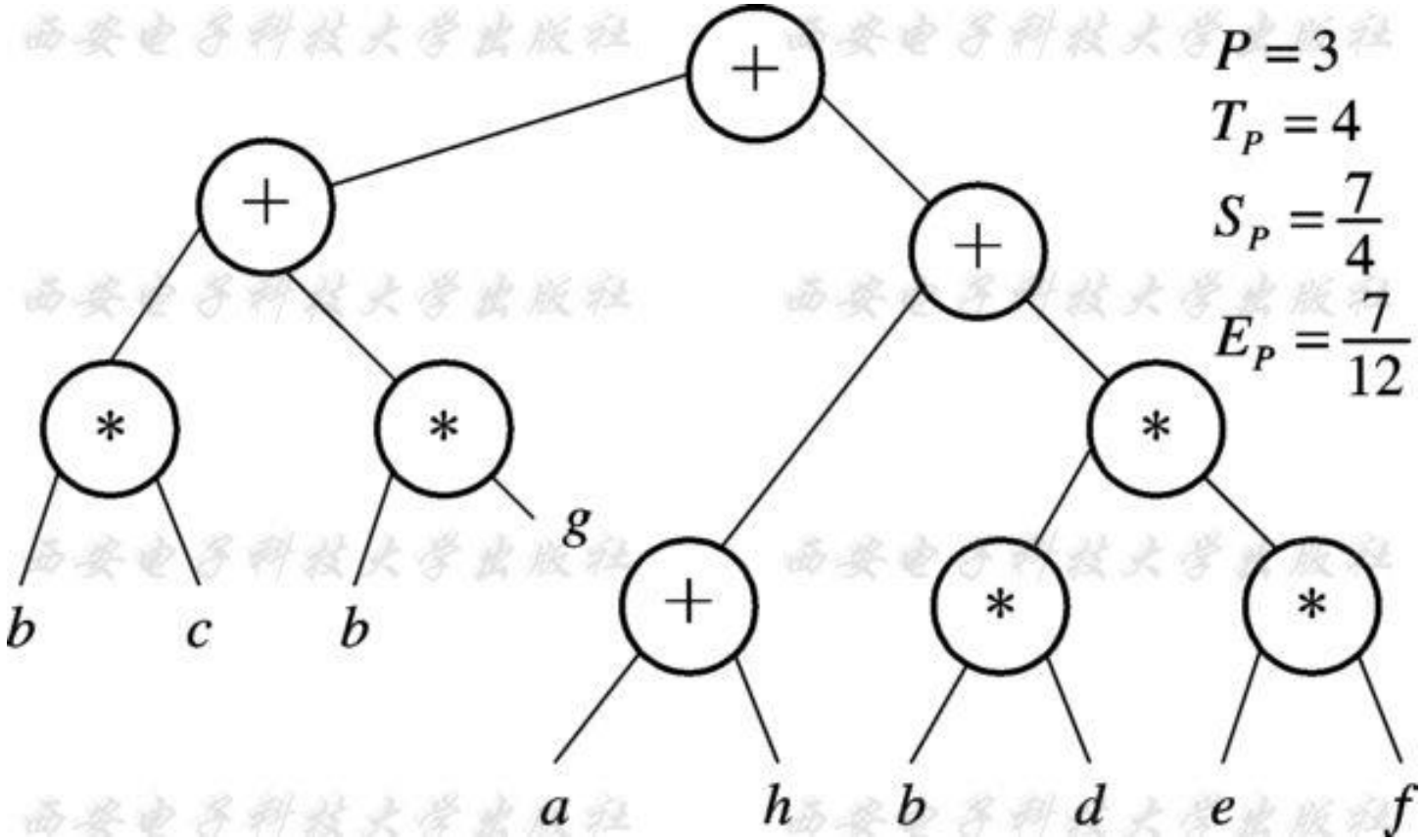



(b)

利用交换律和结合律降低树高

若不考虑处理机个数限制，通过分配率，原式可变为：

$$E_2=(a+h)+(bc+bg)+bdef$$




Tip：先将可同时进行的任务层层排列，再以树高最矮为目标调整。

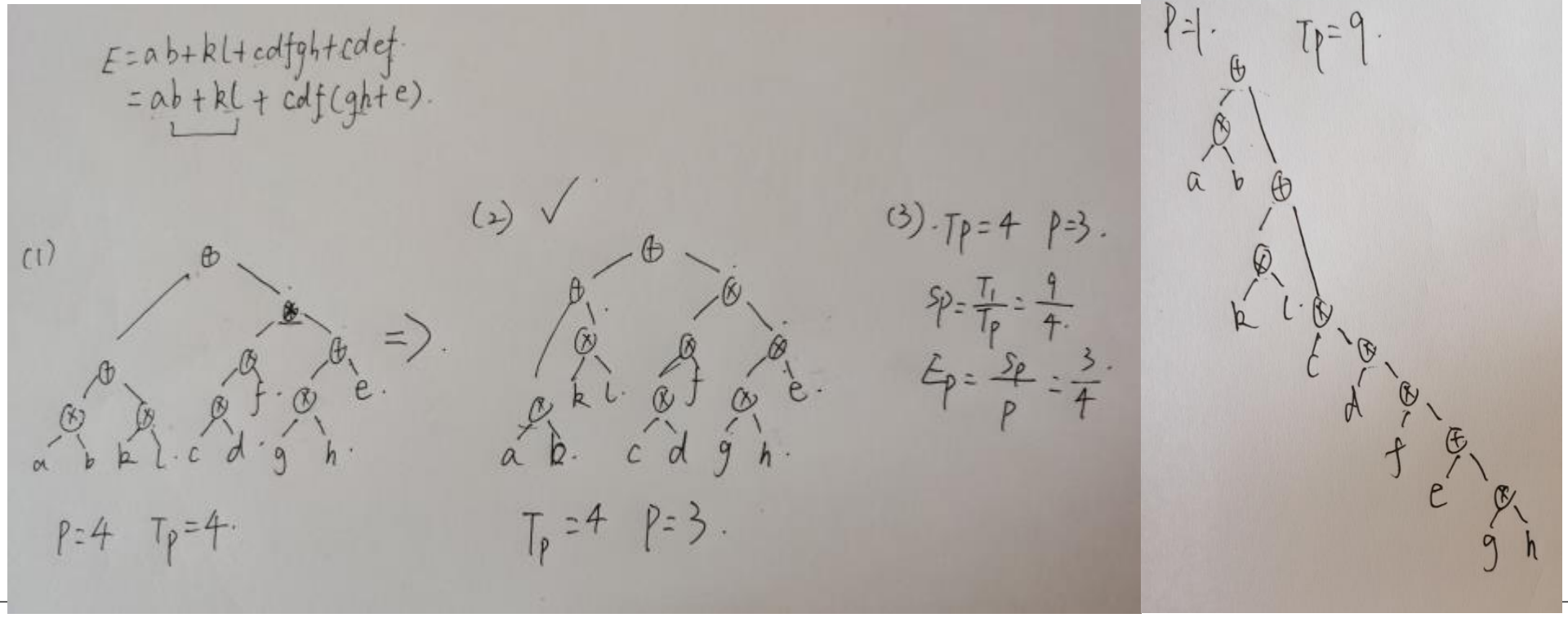
# 例、要计算的表达式如下

$$E = ab + kl + cdfgh + cdef$$

利用减少树高的方法来加速运算。要求：

（1）画出树形流程图　（2）确定Tp、P、Sp、Ep

分析：



# 5.3.2 程序并行性分析

任务间是否能并行，除了算法之外，很大程度取决于程序的结构。数据相关是限制程序并行的重要因素。

若一个程序包含了P1，P2，P3……等n个程序段，则书写顺序体现的是正常执行的顺序

多处理机系统中，程序段并行必然是“异步流动”。

- 数据相关
- 数据反相关
- 数据输出相关

# 1. 数据相关

如果$P_i$的左部变量在$P_j$的右部变量集内，且$P_j$必须取出$P_i$运算的结果来作为操作数，就称$P_j$“数据相关”于$P_i$。

相当于流水线中发生的“先写后读”相关

$P_i$: ...
$A=B+D$
...
$P_j$: $C=A*E$
...

当$P_i$和$P_j$服从交换律时，虽不能并行执行，但可以交换串行

$P_i$: $A=2*A$
$P_j$: $A=3*A$

## 2. 数据反相关

如果Pj的左部变量在Pi的右部变量集内，且当Pi未取用其变量的值之前，是不允被Pj所改变的，就称Pi“数据反相关”于Pj 。

$$P_i \quad C=A+E$$

$$P_j \quad A=B+D$$

相当于流水线中发生的“先读后写”相关

当$P_i$与$P_j$并行时，只要硬件上能保证$P_i$对相关单元*A*先读出，就能得到正确的结果；不能交换串行

## 3. 数据输出相关

如果Pi的左部变量也是Pj的左部变量，且Pj存入其算得的值必须在Pi存入之后，则称Pj“数据输出相关”于Pi。

$P_i$ $A=B+D$

$P_j$ $A=C+E$

相当于流水线中发生的“写-写”相关

当$P_i$与$P_j$并行时，必须要同步保证写入的先后顺序；不可交换串行。

## 4. 其他相关

若两个程序段的输入变量互为输出变量， 同时具有“先写后读”和“先读后写”两种相关， 以交换数据为目的， 则两者必须并行执行， 既不能顺序串行， 也不能交换串行。

$$P_i: \quad A=B$$

$$P_j: \quad B=A$$

并行执行，且必须保证读、写完全同步。

如果两个程序段之间不存在任何一种数据相关，即无共同变量，或共同变量都只出现在右边的源操作数，则两个程序段可以无条件地并行执行，也可以串行或交换串行

# 5.3.3 并行语言与并行编译

并行算法需要用并行程序设计语言进行设计。一般有二种方法：

(1) 使用普通顺序型程序设计语言扩充，增加并行进程的成分。

(2) 设计一种全新的并行程序设计语言，支持并行处理。

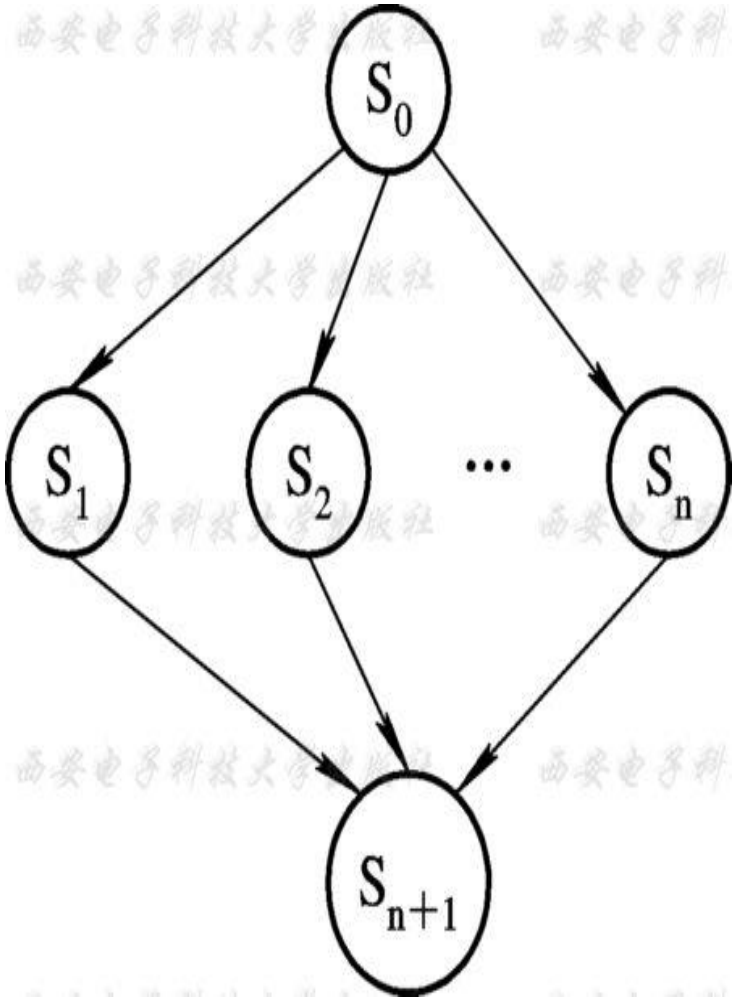
- 对并行设计语言的要求要满足二条：

(1) 灵活性：能方便地在其程序中表现出各种类型的并行性。

(2) 效率高：能在各种并行、向量处理机系统中有效地实现。

并行进程在时间上重叠地执行，一个进程还没有结束，另一个进程就已经开始。

并行进程在多处理机上运行时，需要相应的控制机构进行管理，包括并行性的指令。其中，最主要包括并行任务的派生和汇合。



# 1. 基本的描述程序并行性的语句

- 派生：在一个任务执行的同时，产生出一个或多个与它并行的任务，分配给不同的、正在等待的处理机完成。这些任务可以是相同的，也可以是不同的，执行的时间也可以各不相同。

  派生指令格式： **Fork m**

- 汇合：把分散在各处理机执行的任务，全部完成后，再汇合起来，进入后续任务。后续任务可以是单任务或新的并行任务。新的并行任务，又要派生和汇合的过程，直到整个程序结束运行为止。

  汇合指令格式： **Join n**

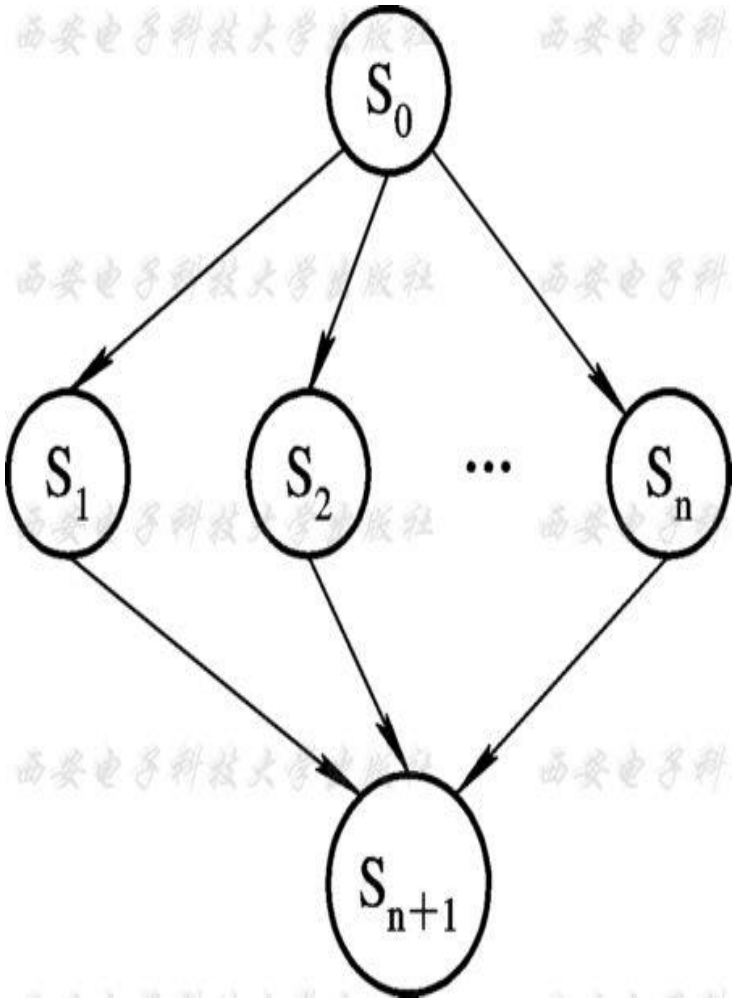
## FORK m 功能：

- m 是一个新进程开始的标识符、标号；
- 某一进程执行初始阶段，执行 FORK m 时，派生出标识符为 m的新进程，即准备好这个新进程的启动和继续执行所必需的有关信息；
- 将空闲的处理机分配给 FORK 语句派生的新进程，如果没有空闲的处理机，则应让它们排队等待；
- 原进程在FORK语句结束后，继续在原处理机上执行。

- JOIN n 功能：

并行的进程间共享join的计数器

  - n 为并发进程的个数
  - JOIN 语句附有一个计数器，初始值为 0；
  - 每当执行 JOIN 语句时，计数器的值加 1，并与 n 相比较：
    - ✓如果比较相等，说明这是执行中的第 n 个并发进程,经过JOIN 语句,将计数器的值清0，并继续执行后续指令;
    - ✓如果比较计数器的值小于 n，表明此进程不是并发进程中的最后一个,让正在执行的这个进程先结束,把它占用的处理机释放出来，分配给正在排队的其他任务,如果没有排队等待的任务,则让该处理机空闲。




```
        进程S0
10      FORK 02
        FORK 03
        …
        FORK  n
        进程S1
        JOIN n
        GOTO NEXT
02      进程S2
        JOIN N
        GOTO NEXT
30      进程S3
        JOIN N
        GOTO NEXT
        …
NEXT    进程Sn+1
```

## 2. 并行编译

算术表达式的并行程序，除了要有并行算法外，还要依靠并行编译程序，有些编译算法可以直接从算术表达式产生，能够并行执行目标程序。

好的并行编译算法，应能尽可能消除相关，有利于接下来程序的并行性的开发。

假设有 $Z=E+A*B*C/D+F$，经并行编译得到如下程序：

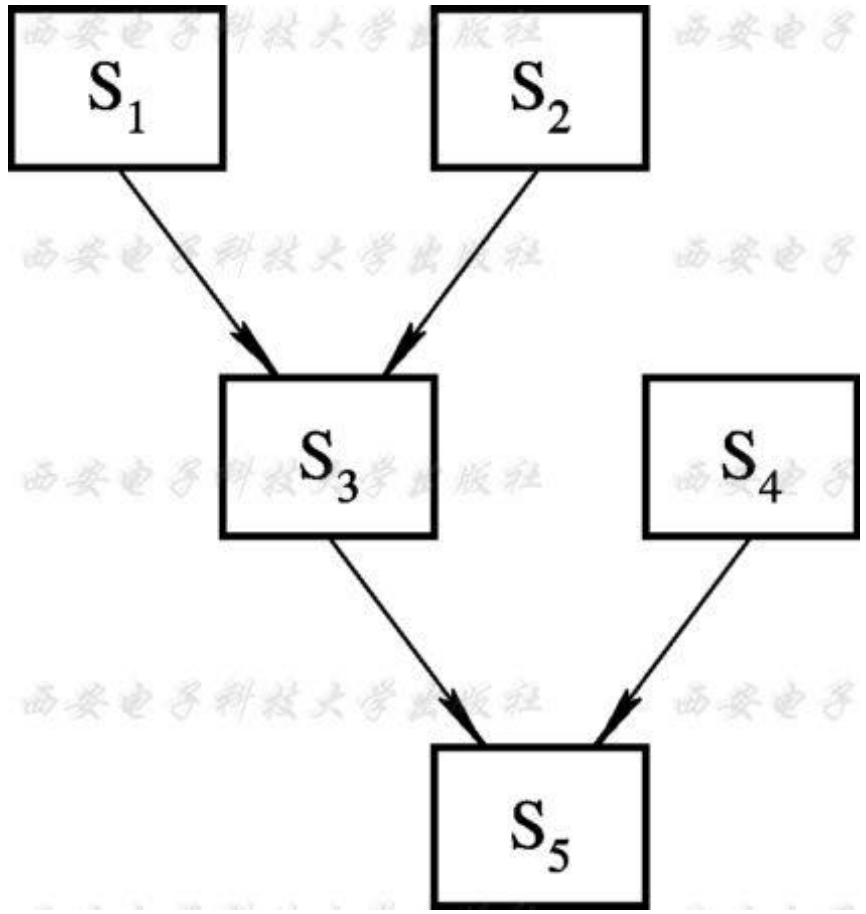
$S_1$ G=A*B

$S_2$ H=C/D

$S_3$ I=G*H

$S_4$ J=E+F

$S_5$ Z=I+J

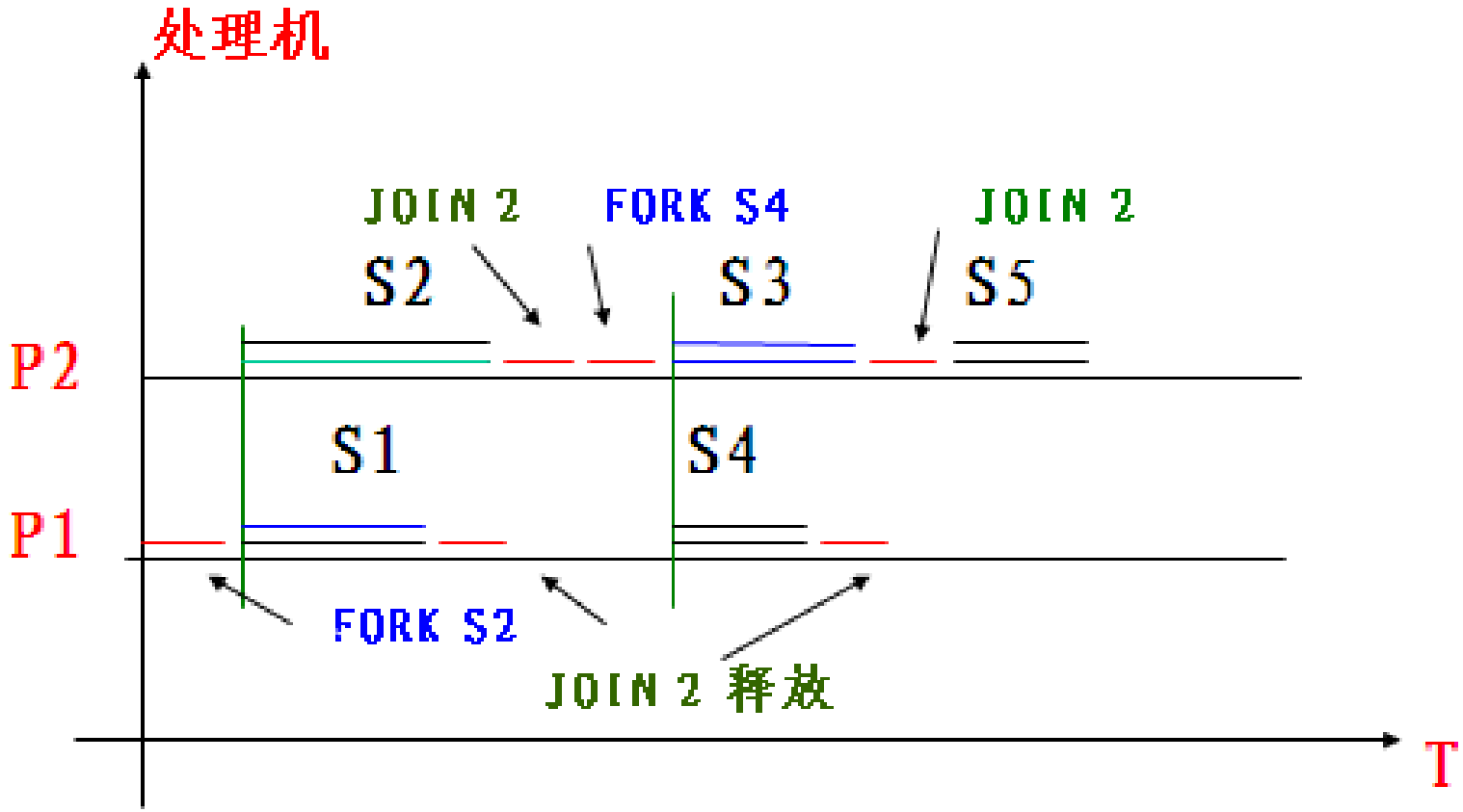



并行程序数据相关图

```
      FORK 20
10    G=A*B      (进程S1)
      JOIN 2
      GOTO 30
20    H=C/D      (进程S2)
      JOIN 2
30    FORK 40
      I=G*H      (进程S3)
      JOIN 2
      GOTO 50
40    J=E+F      (进程S4)
      JOIN 2
50    Z=I+J      (进程S5)
```

执行这个程序用2台处理机。

- 假设最初的程序在处理机 1上运行，遇到语句 FORK 20 时，派生一个进程给处理机 2，而处理机 1 继续执行 S1 。
- 如果 S1 执行时间较短，结束时遇到 JOIN 2 语句，计数器的值 I=I+1，小于 2，处理机 1 从 S1 释放，S2 还在执 行中,处理机 1因为没有其他任务而空闲。
- 当 S2 执行结束时，遇到 JOIN 2，I=I+1，计数值等于2，已与 S1 汇合，便可通过 JOIN 语句，由处理机2继续执行30 语句。




资源时间图

资源时间图就是根据各任务运行的时间，派生和汇合的功能进行绘制，一定要注意各任务的时间关系。

## 注意：

- 程序无特殊说明时，书写的顺序即为顺序串行执行的顺序；
- FORK语句派生出来的标示符，并无实际意义，仅作为标签起到说明作用。可以用来表明某个语句或进程的起始位置。
- 标示符和GOTO相配合，保证并行程序的逻辑正确性。
- FORK语句和JOIN语句为操作原语，执行时间一般远小于进程的实际执行时间。

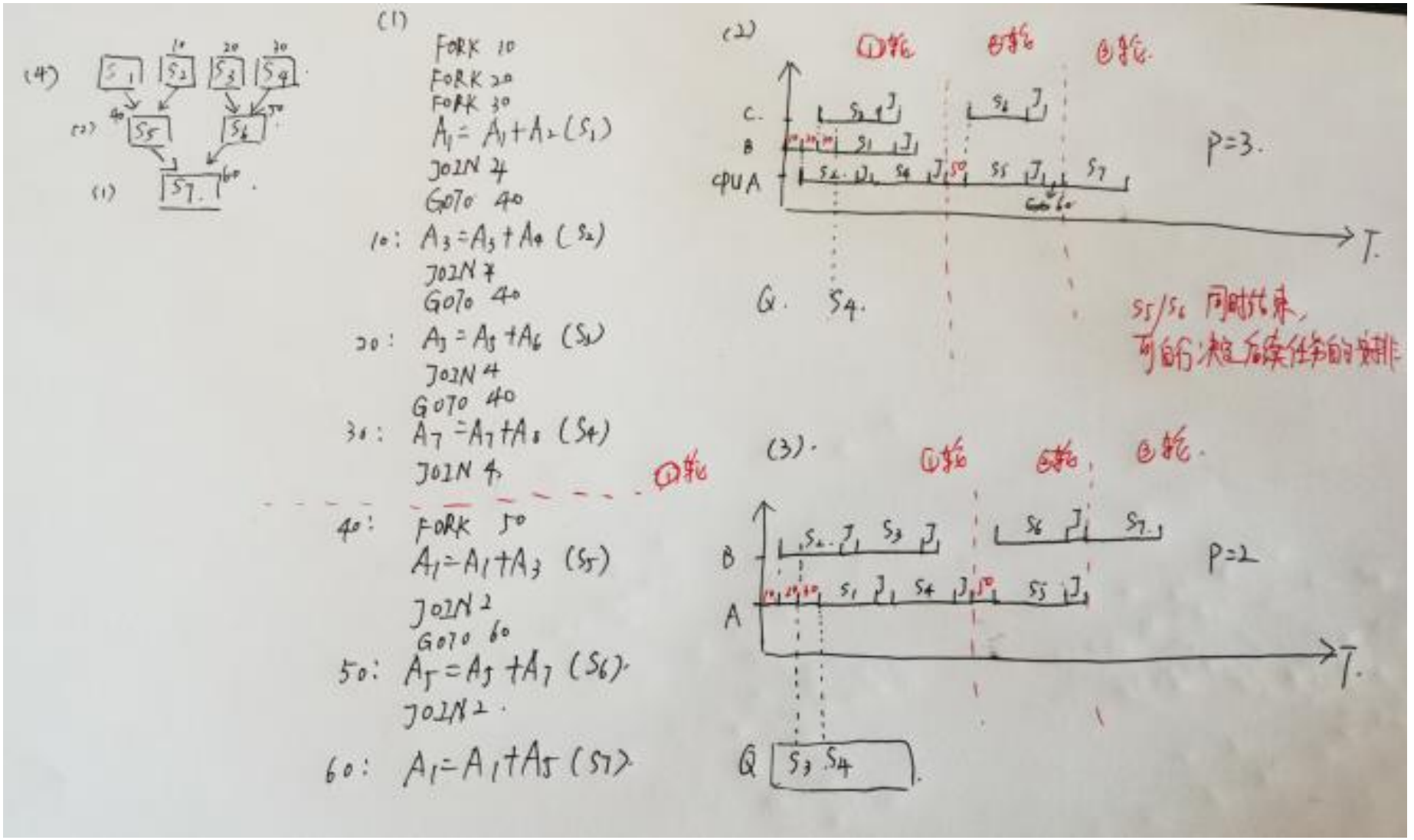
思考：求A1、A2、…A8的累加和。若经过编译后，有如下程序：

$S_1$ A1＝A1＋A2
$S_2$ A3＝A3＋A4
$S_3$ A5＝A5＋A6
$S_4$ A7＝A7＋A8
$S_5$ A1＝A1＋A3
$S_6$ A5＝A5＋A7
$S_7$ A1＝A1＋A5

（1）写出用FORK、JOIN语句表示任务派生和汇合关系的并行程序

（2）画出该程序在3台处理机的系统上运行的时间关系图

（3）画出该程序在2台处理机的系统上运行的时间关系图




一般地，$T_{+/-} < T_* < T_/$，若出现两个进程同时结束情况，可以自行定义一个完成汇合并进行执行后续任务。

# 3. 其他描述并行性的语句

采用高级语言的扩展，也可用专门设计的新语言，都必须要有描述并行性的语句。

例如 E.W.Dijkstra 的语言方案是块结构语言的发展，

他把所有可并行执行的语句或进程S1，S2，…… Sn
用cobegin ---- coend(或 parbegin ---parend )
前后括号起来

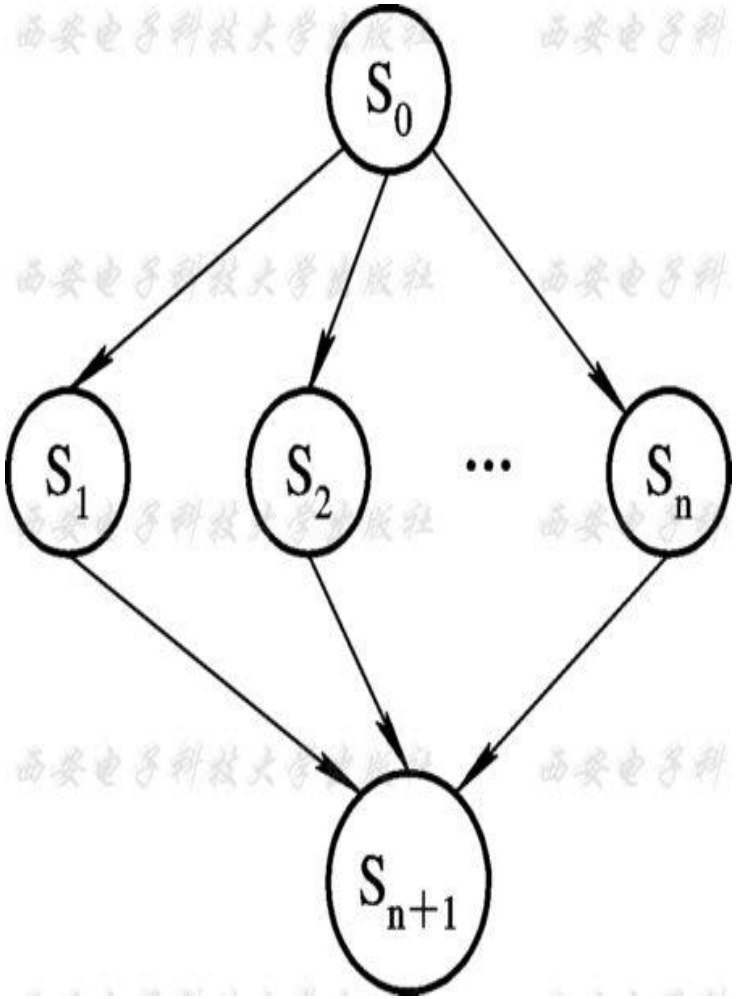
begin

$S_0$;

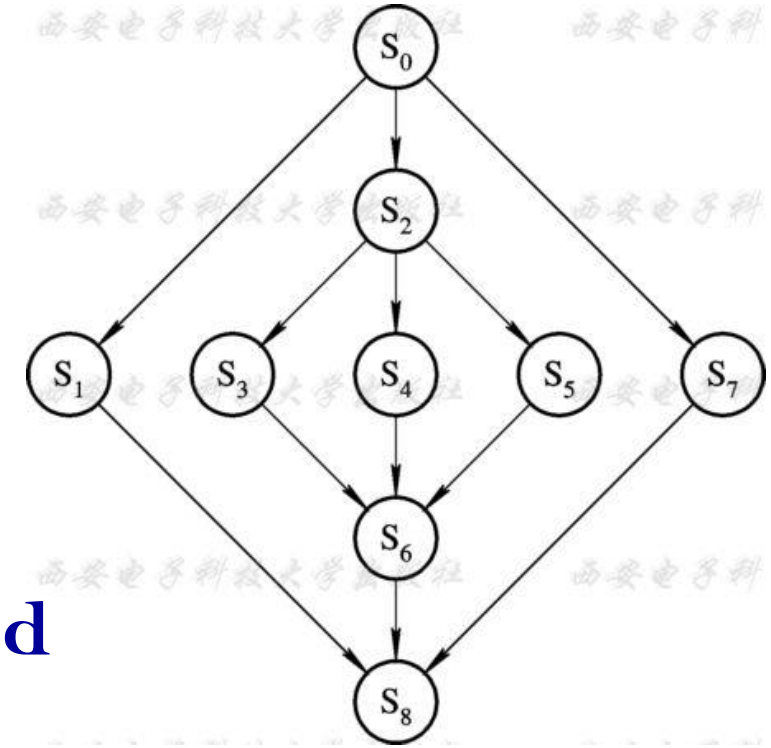
cobegin $S_1$; $S_2$; …; $S_n$; coend

$S_{n+1}$ ;

end



```
begin
  S0;
  cobegin
      S1;
      begin
        S2;
          cobegin S3; S4; S5; coend
          S6;
      end
      S7;
  coend
  S8;
end
```




并行语句也可以任意嵌套

用parfor描述单程序多数据的并行，各进程执行同一程序，但各进程所需的参数不同。如果parfor各进程被分派于指定处理机并行执行时，可使用forall语句描述，如：

```
forallS    iwhere0≤i≤kdo
        S(i);
endfor;
```

作为一个完整的并行程序设计语言，除上述并发进程的表示外，还要考虑诸如程序分支、程序循环、并发进程间的通信和同步、数组和进程数组的处理等问题的描述。

# 5.3.4 多处理机的性能

衡量任务粒度大小的一个依据是程序用于有效计算的执行时间$E$与处理机间的通信等辅助开销时间$C$的比值。只有$E/C$ 的值较大时，开发并行性才能得到好处。

任务粒度与系统的应用有关。机间通信开销少，宜于细粒度处理；要求冗长计算才能得到结果的题目，宜于粗粒度处理，因为粒度过细会过分增大额外开销和机器造价，且解题效率过低。

# 5.4 多处理机的操作系统

并行性的程序在多处理机上运行，必须要有相应的控制机构来实现管理功能。程序的并行性主要是通过多处理机的操作系统，用软件手段实现。

## 5.4.1 多处理机操作系统的功能

- 处理机的分配与进程的调度
- 进程间的同步
- 进程间的通信
- 存储系统的管理
- 文件系统的管理
- 系统重组

## ① 处理机的分配和进程的调度

- 要了解处理机的能力 ---- 分配合适的任务
- 要了解资源的使用状态 ---- 达到负荷平衡、均匀
- 要了解作业和任务之间的关系 ---- 处理好相关、并行的关系

## ② 进程间的同步

- 要解决好同一处理机程序的并发执行、同步；
- 要解决好不同处理机程序的并发执行、同步，要采用新的同步机制和同步算法。

## ③ 进程的通信

松散耦合的多处理机系统中，进程间的通信还可能要通过长距离的信道或网络进行，一般采用消息包形式通信。

## ④ 存储系统的管理

- 要有访问存储器地址的映象和变换，保证数据的一致性；
- 要确定访问的存储器是局部存储器，还是共享的公用存储器；
- 要解决访问存储器冲突的仲裁。

## ⑤ 文件系统的管理

- 集中式：用户的文件集中在某一处理机上，由该机操作系统统一管理。
- 分散式：各处理机都有自己的文件系统，有各自的独立管理。如果一台处理机要使用另一台处理机的文件，使用起来比较困难。
- 分布式：保存的所有文件分布在不同的多个处理机上，在逻辑上是一个整体。每个处理机都可使用，不要求了解文件的物理位置。

## ⑥ 系统重组

当系统出现故障时，能迅速重组并恢复运行，使故障弱化。

# 5.4.2 多处理机操作系统的特点

①程序执行的并行性.

并行执行程序，提高资源的利用率和系统处理能力。多处理机操作系统希望能增强程序执行的并行性，以获得更高的系统处理能力和处理速度。

②分布性

- 任务上分布，表现多个任务在多个处理机上的并行执行。
- 资源上分布，表现在系统资源被配置到多个处理机上。能够充分系统的效率，并达到资源共享。
- 控制上分布，表现在各台处理机均配置自己的操作系统。

### ③机间通信与同步性

把分散在多机间的信息的传输，要通信和同步，其性能直接影响到程序的并行性和系统的性能。

### ④系统的容错性

系统发生故障，系统能动态切换，重新组合。系统的容错性对多处理机是非常重要的。

总之 ，对于多处理机操作系统：

多处理机操作系统性能的强弱，直接影响多处理机系统的性能的发挥。多处理机操作系统，是多处理机系统软件的核心。对多处理机操作系统有影响的有：通信方式，同步机构，布局和分配策略，互联网络等。

# 5.4.3 多处理机操作系统的分类

## *(1) 主从型操作系统*

有一台主处理机上运行操作系统，其他的处理机为从处理机，由主处理机管理从处理机的进程和分配任务。

一台从处理机在执行进程的过程中需要得到管理程序提供服务时，必须向主处理机发生申请，等待主处理机响应后对它提供服务。

## 优点：

- 系统的软件和硬件都较简单。
- 只要单处理机的多道操作系统中增添少量功能即可实施。
- 对解决冲突问题比较容易。

## 缺点：

- 主处理机一旦出故障，整个系统将停止工作，需干预后重新启动。
- 正常工作时，主处理机成为系统的瓶颈，调度速度慢，而使从处理机空闲。

*(2)各自独立型操作系统*

每台处理机都有一个独立的管理程序（操作系统的内核）在运行，即每个处理机都有一个内核的副本，执行各种管理功能。

- 优点：

➢适应分布处理的模块化结构特点，减少对大型控制专用处理机的需求 ；

➢有较高的可靠性，每个处理机发生故障时，不会引起系统的瘫痪。

➢每台处理机都有其专用控制表格，使访问系统表格冲突减少，由于进程的统一调度，提高了系统的利用率。

## 缺点：

- 每台处理机仍然有一些共享表格，会增加共享表格的访问冲突，导致进程调度的复杂性，加大了开销。
- 每台机都有自己专用的输入输出设备和文件，使整个系统的I/O 结构变换需要操作员干预。
- 某一台处理机一旦发生故障，要想恢复和重新执行未完成的工作较困难，各处理机负荷平衡比较困难。

### *(3) 浮动管理控制方式*

浮动型操作系统是界于主从型和单独型之间的一种折衷方式，其管理程序可以在处理机之间浮动。担任“主控制处理机” 的设备不固定、担任的时间不固定。

- 优点：

  - 主控程序可以从一台处理机转移到另一台处理机，也可以同时有多台处理机执行同一个管理服务子程序
  - 各类资源做到较好的负荷平衡；
  - 通过静态设置或动态控制的优先级，安排服务请求的次序。

## 缺点：

- 由于存在多个管理程序，所以表格的访问冲突，表格封锁延 迟是不可避免的。
- 该方式在硬件结构和可靠性上具有分布控制的优点，而在操作系统的复杂性和经济性接近主从型。
- 操作系统设计比较复杂。

# 5.5 多处理机的发展

- 分布式共享存储多处理机
- 对称多处理机
- 分布式计算机（联网计算机）
- 向量多处理机
- 并行向量处理机
- 大规模并行处理机
- 机群系统

## 1、分布式共享存储器多处理机

采用共享虚拟存储器的方法：将物理上分散的各台处理机所拥有的本地存储器在逻辑上统一地编址，形成一个统一的虚拟地址空间，以实现存储器的共享。

分布式共享存储器的多处理机采用Cache目录表来支持分布Cache的一致性。

## 2、对称多处理机

I/O流量高，分时共享能力、容错能力均很强，用于频繁进行中小规模的科学与工程计算、事务处理和数据库管理。

与并行向量机不同，其处理器不是专用的，而是在片Cache的微处理器，再外加片外Cache，经高速总线或纵横交叉开关连到共享存储器上。

系统是对称的：各个处理器均可同等地访问共享存储器、I/O设备和运行操作系统，并行性较高。

但是，系统的可扩展能力较差。

## 3、多向量多处理机

配置有多台处理机、多个向量流水部件和标量部件的系统，系统共享主存。

## 4、并行向量处理机

由若干个数目不等的强功能的专用向量处理器经高带宽的纵横交叉开关互连到若干个共享的存储器模块。

这类机器一般不使用Cache，而采用大量向量寄存器和指令缓冲存储器。

## 5、大规模并行处理机

数百至数千个高性能、低成本的RISC微处理器用专门的互连网络互连，组成大规模并行处理机MPP。这种处理机可进行中粒度和细粒度大规模并行处理，构成SIMD或MIMD系统。

MPP的操作系统中，内核只提供中断处理、进程调度、进程间简单通信及其他最基本的功能，将大量的服务功能搬移到内核之外。内核基本功能是同构的，对不同用户的不同服务需要，允许进行异构服务。

MPP的每个处理结点都有本地存储器，经网络接口电路连到专门的互连网络上，实现与其他结点通信。微处理器/Cache经存储器总线与本地存储器和网络接口电路连接。

MIMD型的MPP系统是一个异步系统，其每个处理结点使用商品化的微处理器。处理结点内使用物理上分布的独立编址的本地存储器，不同结点间的进程采用消息传递。专门的互连网络具有高带宽。系统可扩展到数千个微处理器中。

## 6、机群系统

机群系统是将多个高性能的工作站或高档微型计算机，使用高速的通信网络加以互连组成的系统。在并行程序设计和集成开发环境的支持下，进行统一调度和协调处理，以实现对中、粗粒度并行进程的高效并行处理。

- 主机和网络可以是同构的，也可以是异构的。
- 采用消息传递进行主机间通信。
- 从结构和结点间的通信来看，是一种分布式存储方式，而从用户来看，表示出的是一个完整的并行系统。