第二章

处理机的调度和死锁

引 言

在多道程序环境下，进程数目往往多于处理机数目。因此，要求系统能按某种算法，动态地把处理机分配给就绪队列中的一个进程，使之执行。该任务由处理机调度程序完成。处理机调度是操作系统设计的核心问题之一。

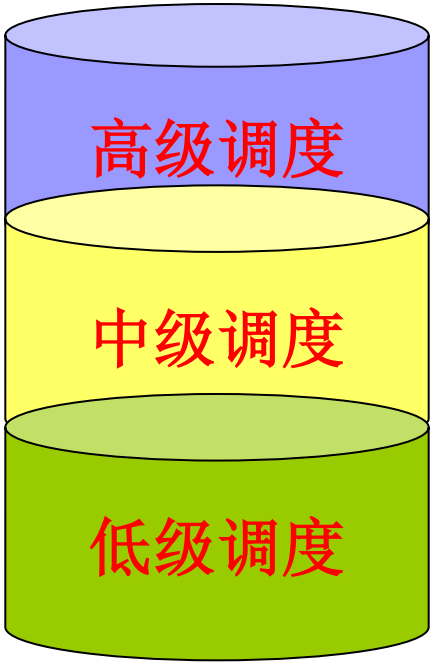
本章主要内容

- 3.1　处理机调度的基本概念
- 3.2　调度算法
- 3.3　实时调度
- 3.4　多处理机系统中的调度
- 3.5　产生死锁的原因和必要条件
- 3.6　预防死锁的方法
- 3.7　死锁的检测与解除

❖3.1 高、中、低三级调度

一个批处理作业，从进入系统直到作业运行完毕，可能要经历三级调度：

* 高级调度(High Scheduling)
* 中级调度(Intermediate -Level Scheduling)
* 低级调度(Low Level Scheduling)



3.1 处理机调度的基本概念

- 高级调度（作业调度、长程调度、接纳调度）

  将外存作业调入内存，创建进程、分配资源，插入就绪队列

  - 一般用于批处理系统，分/实时系统一般直接入内存，无此环节。
  - 调度的两个特性
    - 接纳作业数（多道程序度）

      太多－－－>　　周转时间T长

      太少－－－>　　系统效率低
    - 接纳策略(采用何种调度算法)：FCFS、短作业优先等

处理机调度的基本概念

❖ 低级调度（进程调度，短程调度）

主要是由分派程序（Dispatcher）分派处理机给进程从而执行。

* **非抢占方式：** 进程获得处理机就一直执行，直到完成或阻塞。

  特点：简单，开销小，实时性差。

* **抢占方式：** 暂停正在执行的进程，分配cpu给其它进程。

  （1）时间片原则

  （2）优先权原则

  （3）短作业优先原则。

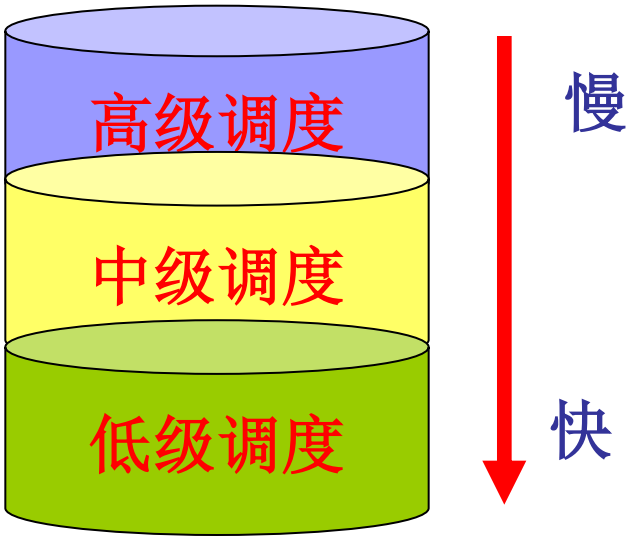
处理机调度的基本概念

❖中级调度（中程调度）

为提高系统吞吐量和内存利用率而引入的一内--外存对换功能（换出时，进程为挂起或就绪驻外状态）

即：静止状态←→ 活动状态

❖运行频率：



3.1.2 调度的队列模型

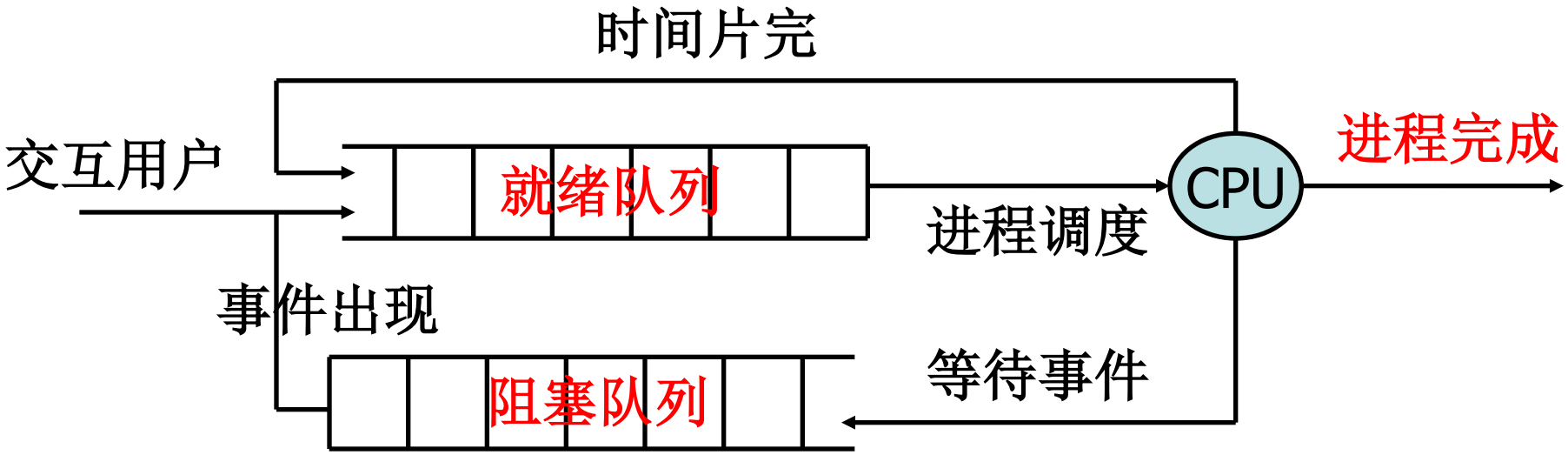
不同工作方式的系统，通常采用不同的调度模型。

- 仅有进程调度的队列模型（分时系统）
- 具有高级和低级调度的调度模型（批处理系统）
- 同时具有三级调度的调度模型（引入挂起状态）

- 模型如何工作？
- 数据结构？
- 调度算法？

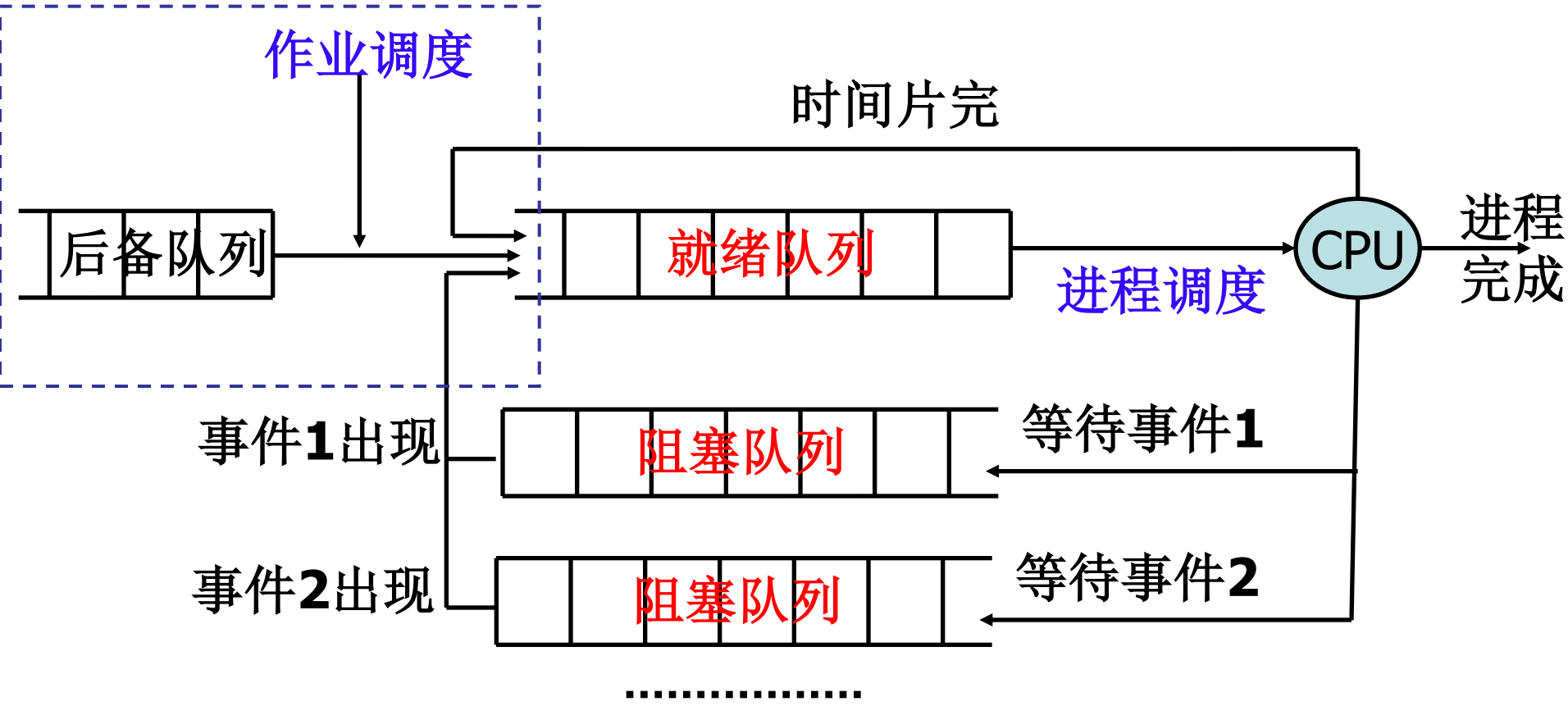
3.1.2 调度的队列模型

❖仅有进程调度的队列模型（分时系统）



3.1.2 调度的队列模型

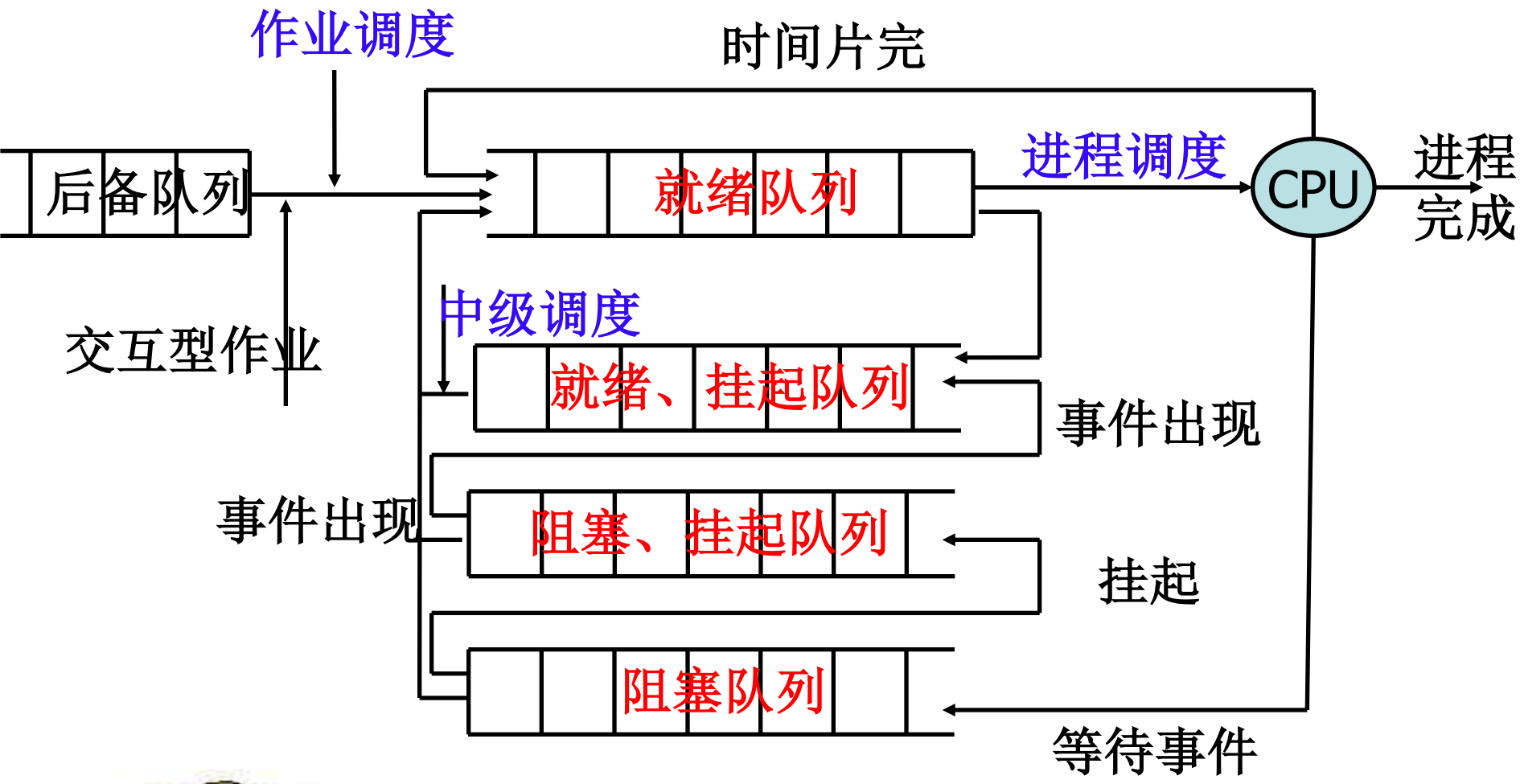
❖ 具有高和低级调度模型（批处理系统）



特点：就绪队列采用优先权调度、设置多个阻塞队列

3.1.2 调度的队列模型

❖ 具有三级调度模型

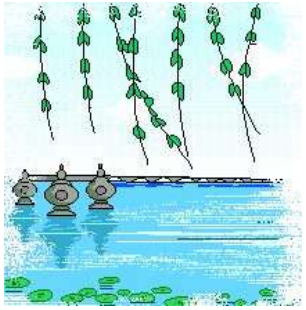


3.1.3 选择调度方式和算法的若干准则

在操作系统的设计中，如何选择调度方式和算法，取决于操作系统的类型及其目标。选择调度方式和算法有以下两个准则：

- 面向用户准则：满足用户要求
- 面向系统准则：满足系统要求

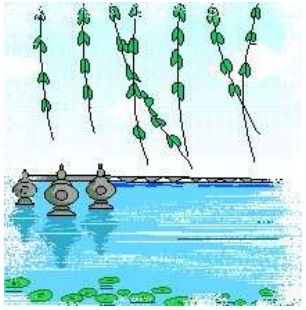
每种准则下的要求有哪些？



3.1.3 选择调度方式和算法的若干准则

❖ 面向用户的准则

* 1. 周转时间短（常用于批处理系统）
  - 概念：作业从提交→完成的时间.
  - 评价批处理系统性能、选择作业调度方式与算法的重要准则。
  - 周转时间组成：
    - （1）驻外等待调度时间
    - （2）驻内等待调度时间
    - （3）执行时间
    - （4）阻塞时间



3.1.3 选择调度方式和算法的若干准则

* 平均周转时间

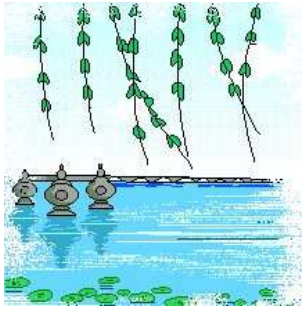
$$T = \frac{1}{n}[\sum_{i=1}^{n} T_i]$$

* 平均带权周转时间

带权周转时间

$$W = \frac{1}{n}[\sum_{i=1}^{n} \frac{T_i}{T_s}]$$

* 带权w越小越好，Ts为实际服务时间。

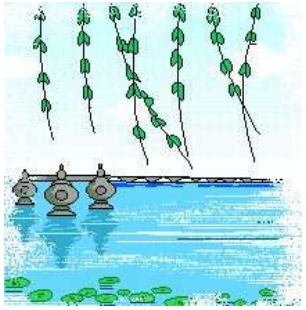


3.1.3选择调度方式和算法的若干准则

* 2. 响应时间快：（对交互性作业）
  - 概念：键盘提交请求到首次响应时间
  - 响应时间包括三部分
    - （1）输入传送时间
    - （2）处理时间
    - （3）响应传送时间

* 3. 截止时间的保证（特别于实时系统）

* 4. 优先权准则（即需要抢占调度）



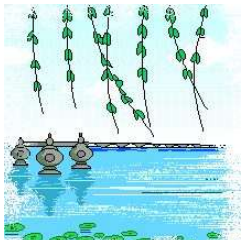
3.1.3 选择调度方式和算法的若干准则

❖ 面向系统的准则

* 1. 吞吐量高（特别于批处理）：单位时间完成作业数。依赖于作业的平均长度、作业调度的方式和算法。

* 2. 处理机利用率好（CPU昂贵的大中型多用户系统）：依赖于调度方式和调度算法，实际系统中，CUP利用率负荷轻时40%，重时90%。

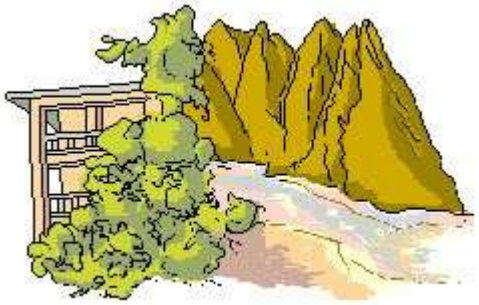
* 3. 各类资源的平衡利用（大中型系统）：如内存、外存、I/O设备等等。



3.2 调度算法—资源分配

- 调度算法概念：根据系统的资源分配策略所规定的资源分配算法。
- 不同的系统和系统目标，采用不同的调度算法。

- 调度算法的适用情况？
- 如何调度？
- 优缺点？



3.2 调度算法

3.2.1 先来先服务和短作业（进程）优先调度算法

- 两种调度算法既可用于作业调度，也用于进程调度。
- 先来先服务FCFS
  - 特点：①简单②有利于长作业(进程)③有利于CPU繁忙性作业(进程)，不利用I/O繁忙型。
- 短作业进程优先调度算法SJ(P)F
  - 提高了平均周转时间和平均带权周转时间（从而提高了系统吞吐量）
  - 对长作业不利，有可能得不到服务（饿死）
  - 没有考虑作业的紧迫性
  - 估计时间不易确定

进程运行举例—FCFS

进程名	到达时间	服务时间	开始执行时间	完成时间	周转时间	带权周转时间
A	0	1	0	1	1	1
B	1	100	1	101	100	1
C	2	1	101	102	100	100
D	3	100	102	202	199	1.99

注意：

周转时间＝完成时间－到达时间

带权周转时间＝周转时间÷服务时间

FCFS和SJF比较

	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS	完成时间	4	7	12	14	18	
	周转时间	4	6	10	11	14	9
	带权周转时间	1	2	2	5.5	3.5	2.8
SJF	完成时间	4	9	18	6	13	
	周转时间	4	8	16	3	9	8
	带权周转时间	1	2.67	3.1	1.5	2.25	2.1

注意比较两者对于长、短进程调度的情况

3.2.2 高优先权优先调度算法FPF

照顾紧迫的作业，使之进入系统后得到优先处理。即可用于作业调度也用于进程调度。

- 优先权调度算法类型
  - 非抢占式优先权算法
  - 抢占式优先权算法，实时性更好。

3.2.2 高优先权优先调度算法

- 优先权的类型（以什么确定优先权）
  - 静态优先权
    - 优先权在进程创建时确定，整个运行期不变。
    - 确定优先权依据
      - （1）进程类型
      - （2）进程对资源的需求；
      - （3）根据用户需求。
    - 特点：简单，但低优先权作业可能长期不被调度。
  - 动态优先权

    优先权在进程创建时确定，但随进程推进而改
    （如等待时间等）。

3.2.2 高优先权优先调度算法

❖ 高响应比优先算法(优先权算法的实例)

- 目的：兼顾长短作业，照顾时间先后。
- 优先权＝(等待时间＋要求服务时间)÷要求服务时间
- 响应时间＝等待时间＋要求服务时间
- 响应比R_P＝(响应时间)÷要求服务时间
- 优点
  - 短作业R_P大，有利于短作业
  - ts（要求服务时间）相同的进程间相当于FCFS。
  - 长作业等待一段时间R_P 变大，仍能得到服务。
- 缺点
  - 需要计算R_P，增加了系统开销

3.2.3 基于时间片的轮转调度算法

❖ 时间片轮转算法

* 时间片大小的确定
  - 太大：退化为FCFS；
  - 太小：系统开销过大
* 时间片选取所需考虑的问题：
  - 系统对响应时间的要求；T=nq
  - 就绪队列中进程的数目n；
  - 系统的处理能力(应保证一个时间片处理完常用命令)

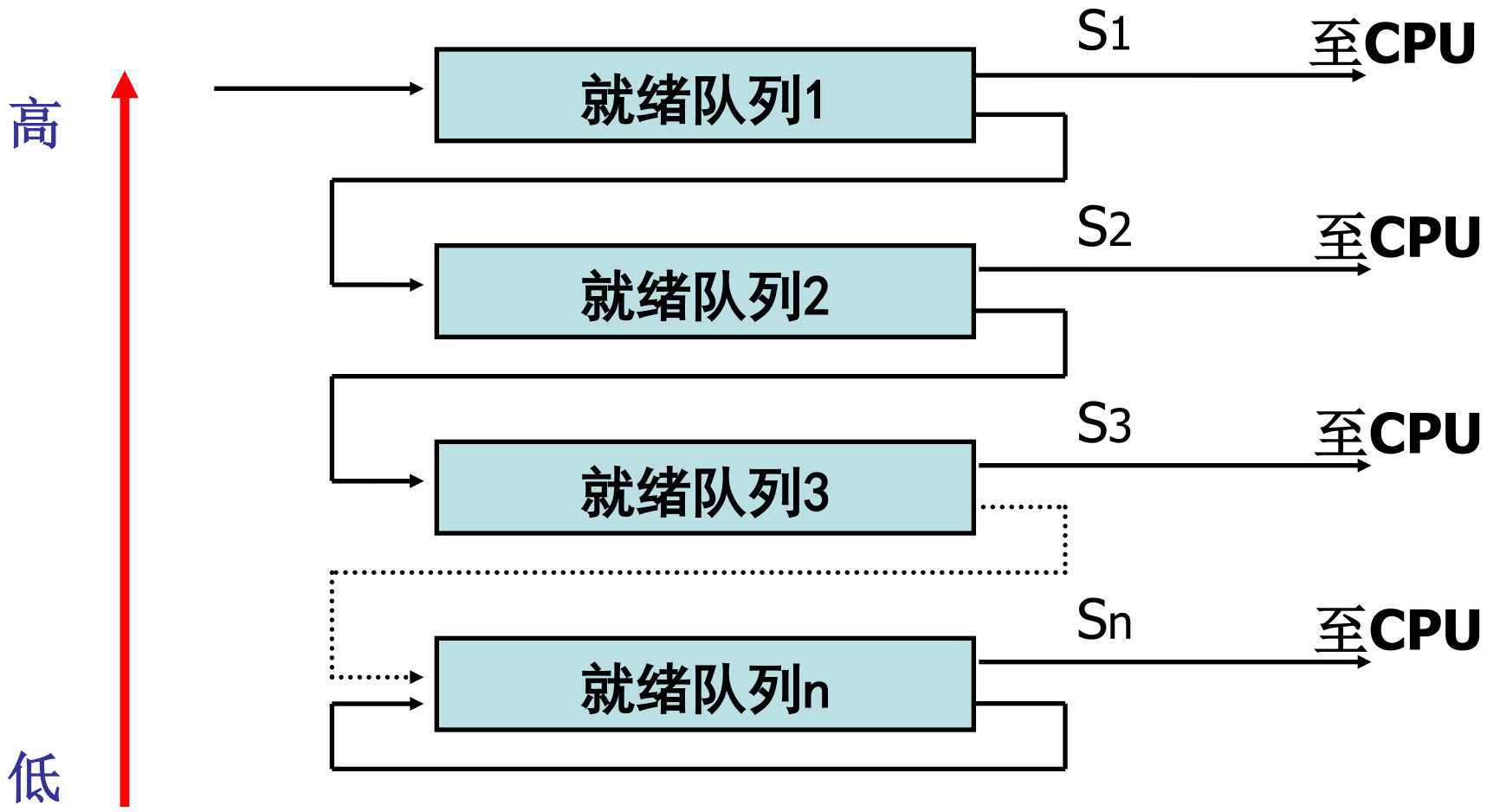
3.2.3 基于时间片的轮转调度算法

❖多级反馈队列调度

* 特点：长、短作业兼顾，有较好的响应时间，不用指明进程长度。
  - （1）短作业一次完成；
  - （2）中型作业周转时间不长；
  - （3）大型作业不会长期不处理。

* 算法设计
  - 每一级队列中进程按FCFS调度
  - i级优先权高于i+1级
  - i+1级时间片比i级时间片长一倍
  - 仅当1～(i-1)队列均空时，才调度i队列

多级队列反馈调度算法



时间片：S1<S2<S3<...

3.3 实时调度

在实时系统中，硬实时任务和软实时任务都联系着一个截止时间。为保证系统能正常工作，实时调度必须能满足实时任务对截止时间的要求。因此，实现实时调度必须具备以下四个条件：

* 提供调度的必要信息
* 考虑系统的处理能力
* 采用抢占式调度机制
* 具有快速切换机制

3.3 实时调度

* 提供必要的调度信息
  - 就绪时间；
  - 开始/完成截止时间；
  - 处理时间；
  - 资源要求；
  - 优先级；

例：在单处理机条件下，系统如有6个硬实时任务，周期均为50ms，处理时间为10ms。系统能否满足？

* 系统处理能力强

$$\sum_{i=1}^{m}\frac{C_i}{P_i}\le 1 \qquad \sum_{i=1}^{m}\frac{C_i}{P_i}\le N$$

C_i为处理时间，P_i为周期时间（基于周期性实时任务）

3.3 实时调度

* 采用抢占调度方式
  - 剥夺方式：硬实时任务一般采用的方法
  - 非剥夺方式（实现简单）：一般应使实时任务较小，以及时放弃CPU。

* 具有快速切换机制
  - 具有快速响应外部中断能力。
  - 快速任务分派。应适当的选择任务的单位长度。

3.3.2 实时调度算法的分类

实时调度算法，按其不同的实现方式可以进行分类！

任务性质
- 硬实时调度
- 软实时调度

调度时间
- 静态调度
- 动态调度

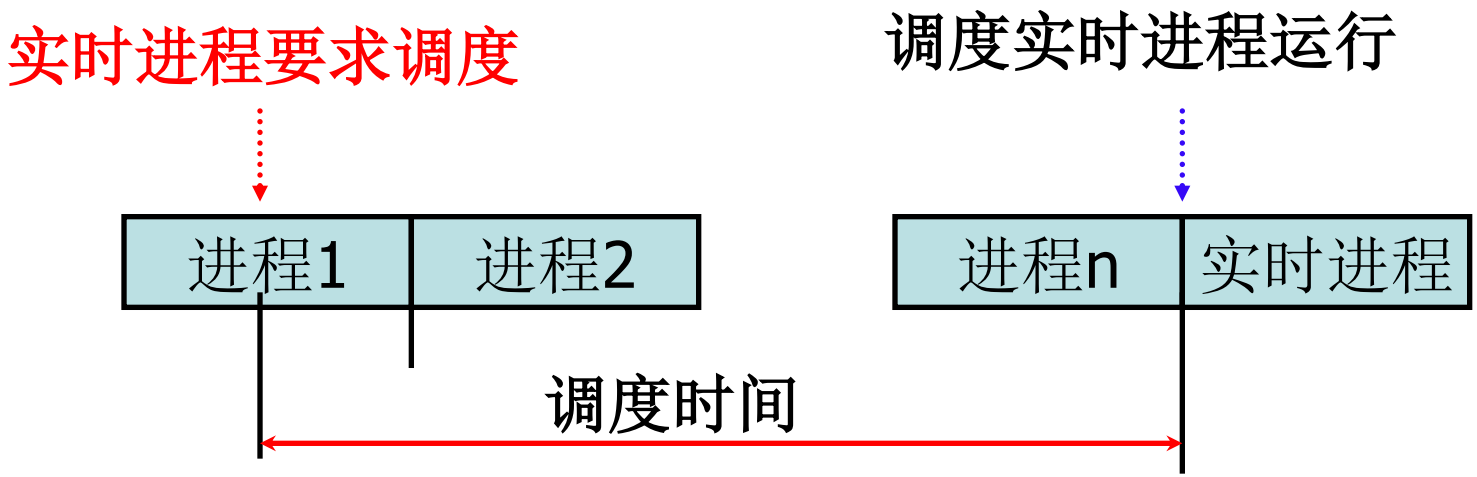
调度方式
- 抢占调度
- 非抢占调度

多机环境
- 集中式调度
- 分布式调度

3.3.2 实时调度算法的分类

❖ 非抢占式调度算法

* 时间片轮转　　秒级

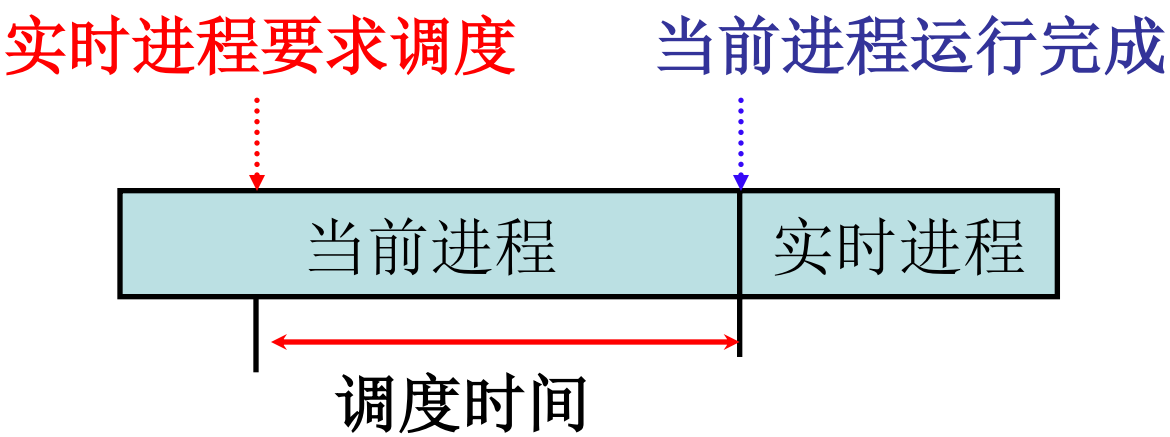


a 非抢占轮转调度

3.3.2 实时调度算法的分类

❖ 非抢占式调度算法

* 非抢占优先权（协同）　　秒-毫秒级



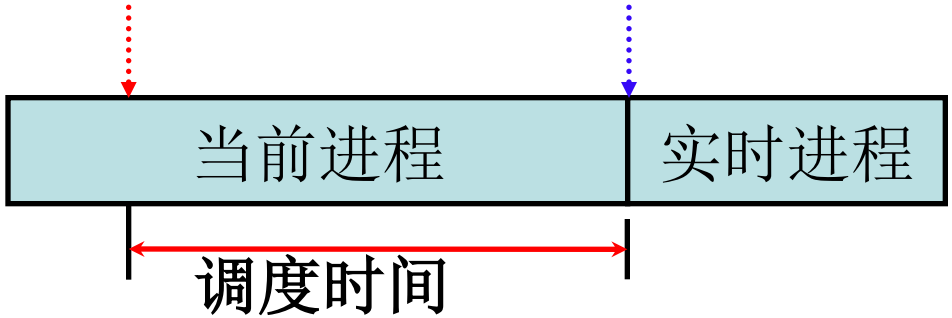
b 非抢占优先权调度

3.3.2 实时调度算法的分类

- 抢占式调度算法
  - 时钟中断抢占优先权　　毫秒级
    - 基于抢占点抢占

注意与非抢占式优先调度比较！

实时进程要求调度　　时钟中断到达时



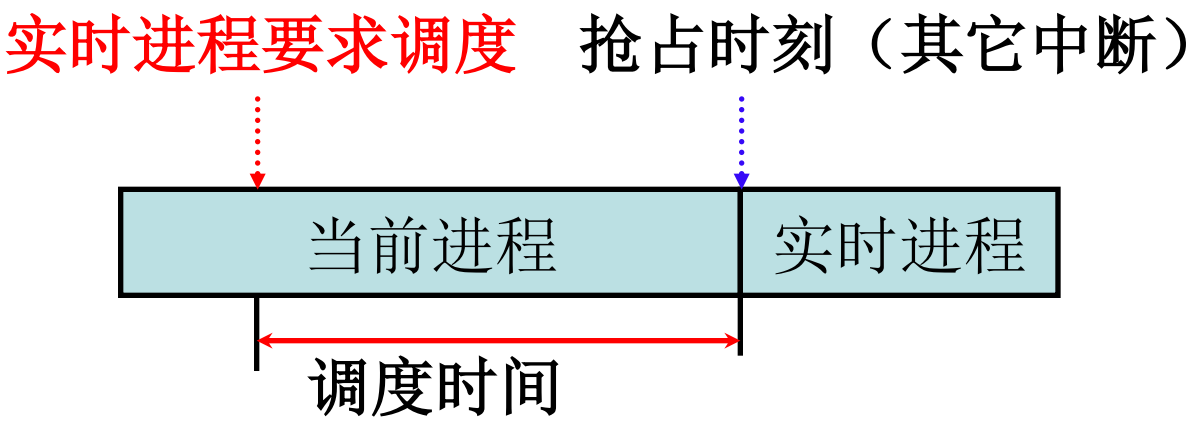
基于时钟中断抢占的优先权抢占调度

3.3.2 实时调度算法的分类

❖ 抢占式调度算法

* 立即抢占　　毫秒-微秒级

  - 只要不在临界区即抢占（中断引发）

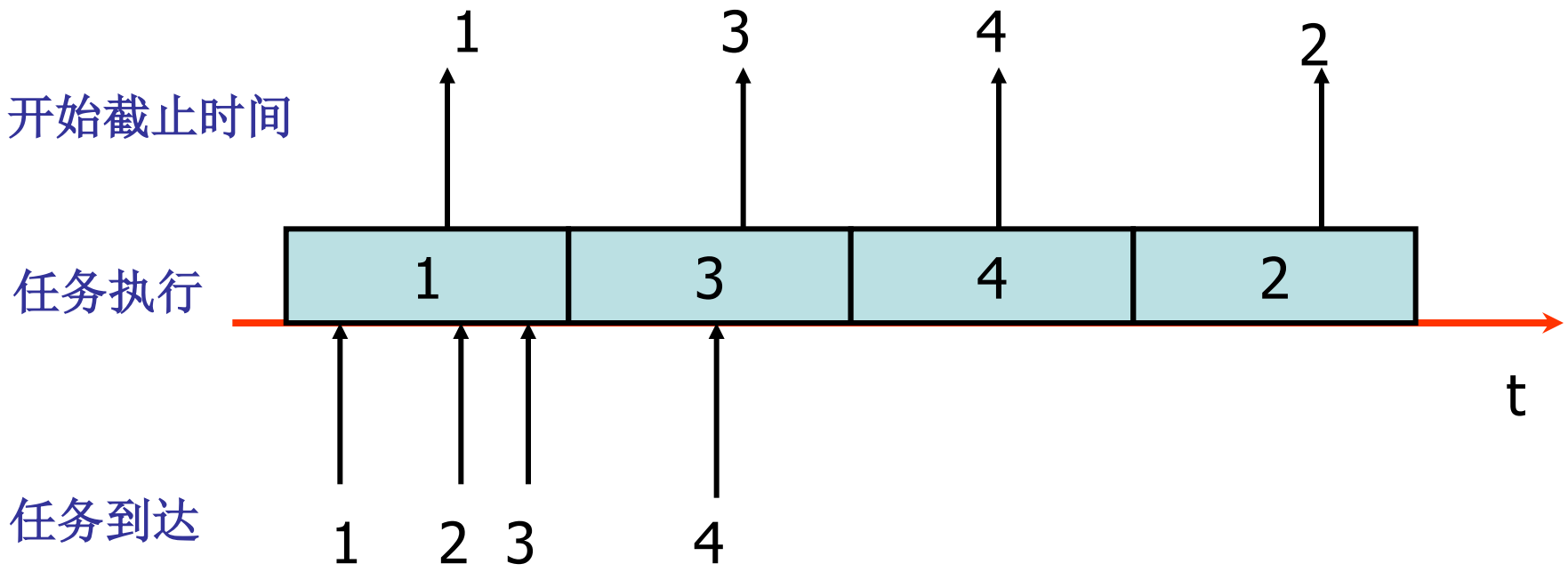


立即抢占优先权调度

3.3.3 常用的几种实时调度算法

❖ 最早截止时间优先EDF（earliest deadline first）算法

- 根据任务的开始截止时间来确定任务的优先级
- 截止时间越早，优先级越高
- 可以是抢占式或非抢占式



3.3.2 实时调度算法的分类

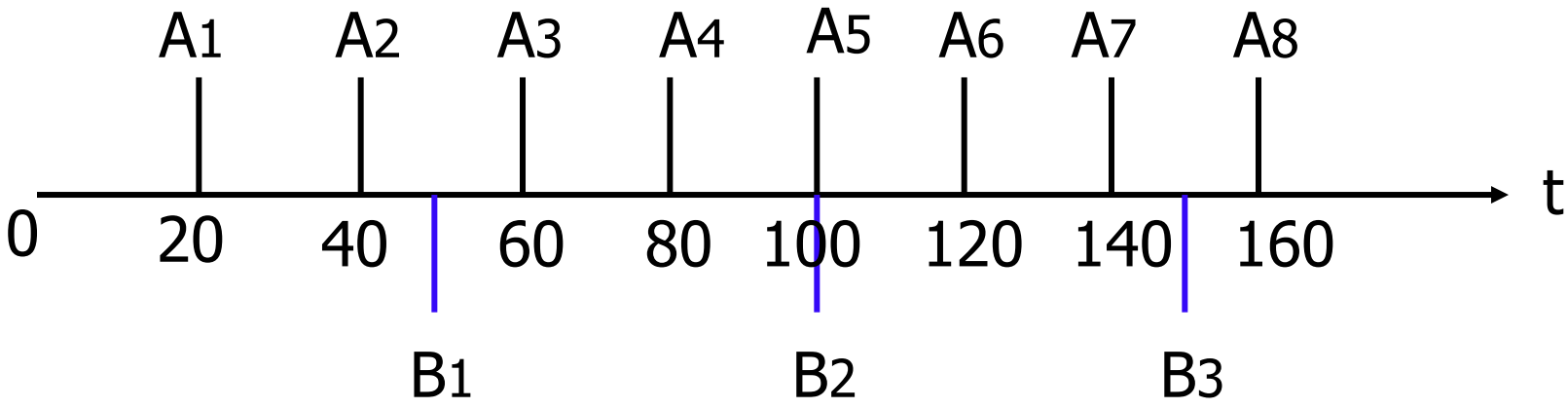
❖ 最低松弛度优先LLF(Least laxity first)算法

* P的松弛度＝P必须完成时间－P运行所需时间－当前时间
* 松弛度越低则紧迫程度越高。
* 松弛度最低的任务排在队首，调度程序总是从队首调度
* 可以是抢占式或非抢占式

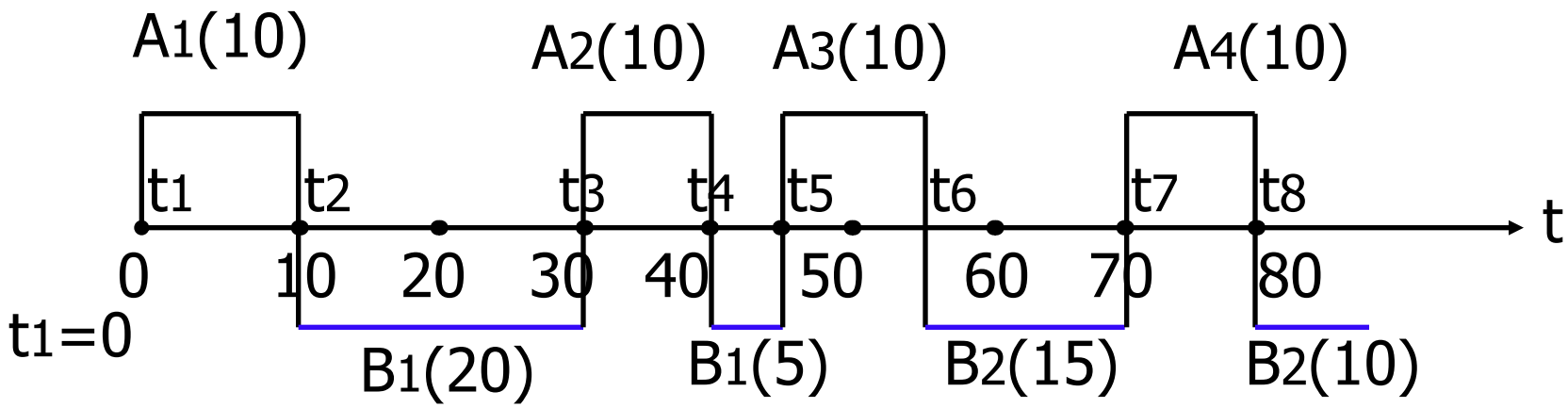
例：

A必须在**200ms**完成，需执行**100ms**，**t=0**；**B**必须在**t=400**完成，执行需**150ms**，**t=10ms**，问谁的松弛度最低，哪个任务更紧迫？

A周期为20ms，每次执行时间10ms。
B周期为50ms，每次执行时间25ms。



A/B任务每次必须完成的时间



利用抢占式最低松弛度**ELLF**算法调度情况

3.4 多处理机系统中的调度

提高计算机系统性能的两条主要途径：

* 提高计算机元器件的速度，特别是CPU的速度。

  当前计算机的其他发展

  - 光计算机、量子计算机、生物计算机

* 改进计算机的体系结构，特别是采用多处理器系统。

引入多处理机系统(MPS)的目的：通过高度并行处理，从而提高系统吞吐量和可靠性。

3.4.1 多处理机系统的类型

- 紧密耦合MPS和松弛耦合MPS
  - 紧密耦合
    - 高速总线和交叉开关连接CUP
    - 共享RAM和I/O
    - 系统所有资源由操作系统管理
  - 松弛耦合
    - 通道和通信线路连接计算机(计算机网络)
    - 独立RAM和I/O
    - 每台计算机都可独立运行

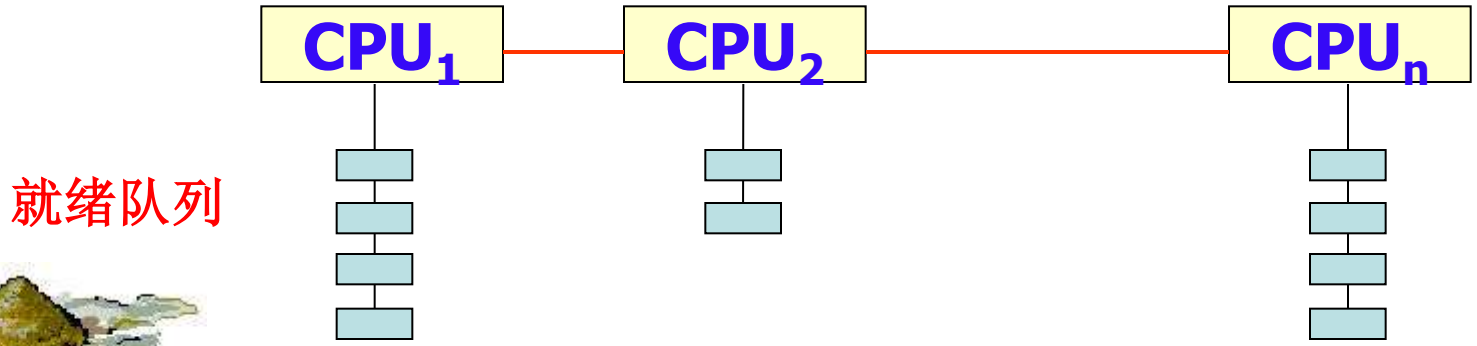
- 对称多处理器系统和非对称多处理器系统
  - 处理器是否结构相同

3.4.2 进程分配方式

❖ SMP中进程分配方式

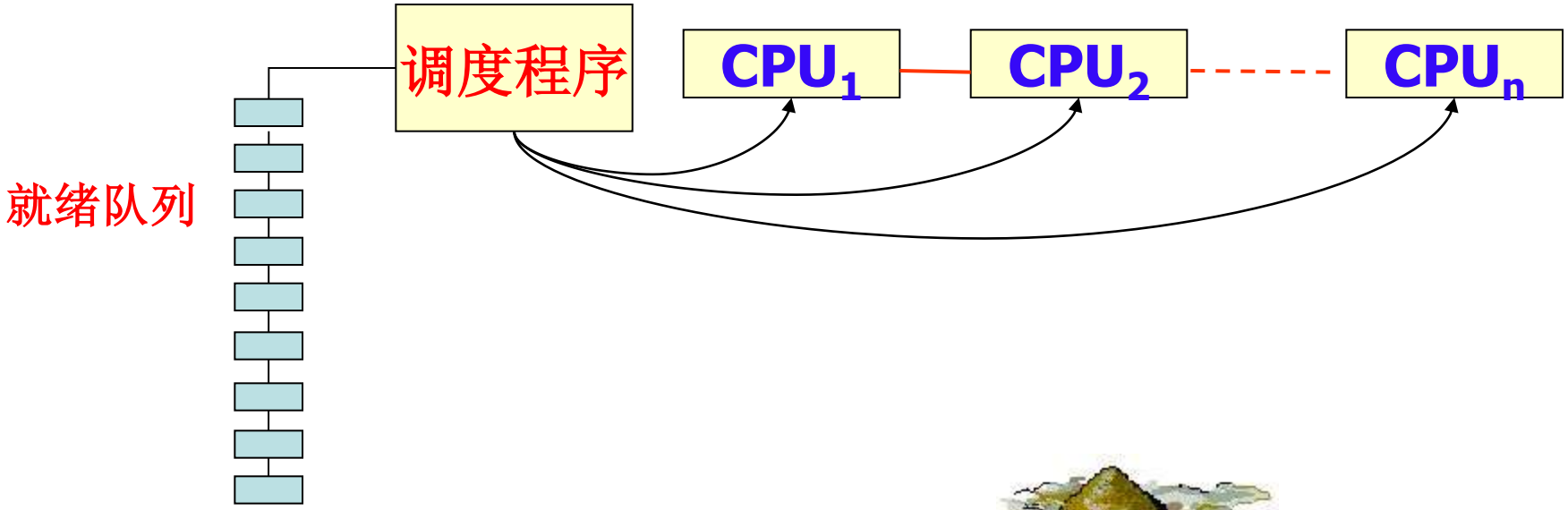
由于CPU都是相同的，故可将进程交给任意一个CPU去处理。因此可将所有处理器组成一个CPU池，由调度程序进行分配。主要采用一下两种分配方式：

* 静态分配
  - 进程从开始到完成始终固定在一个CPU上执行
  - 每个CPU设置一个就绪队列
  - 优缺点：调度开销小，但会使各CPU忙闲不均



❖ SMP中进程分配方式

* 动态分配
  - 解决系统中多个处理器忙闲不均
  - 系统中仅有一个公共的就绪队列(类似于生产者—消费者问题)
  - 一个进程从开始到终止将在不同CPU上运行

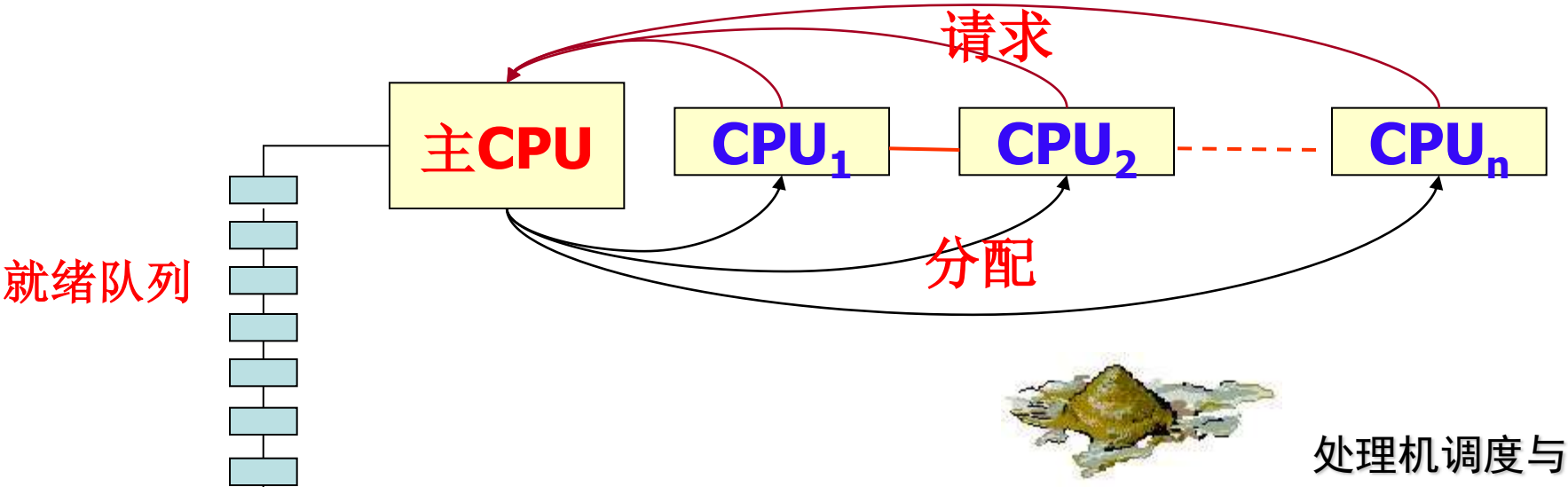


3.4.2 进程分配方式

❖ 非SMP中进程分配方式

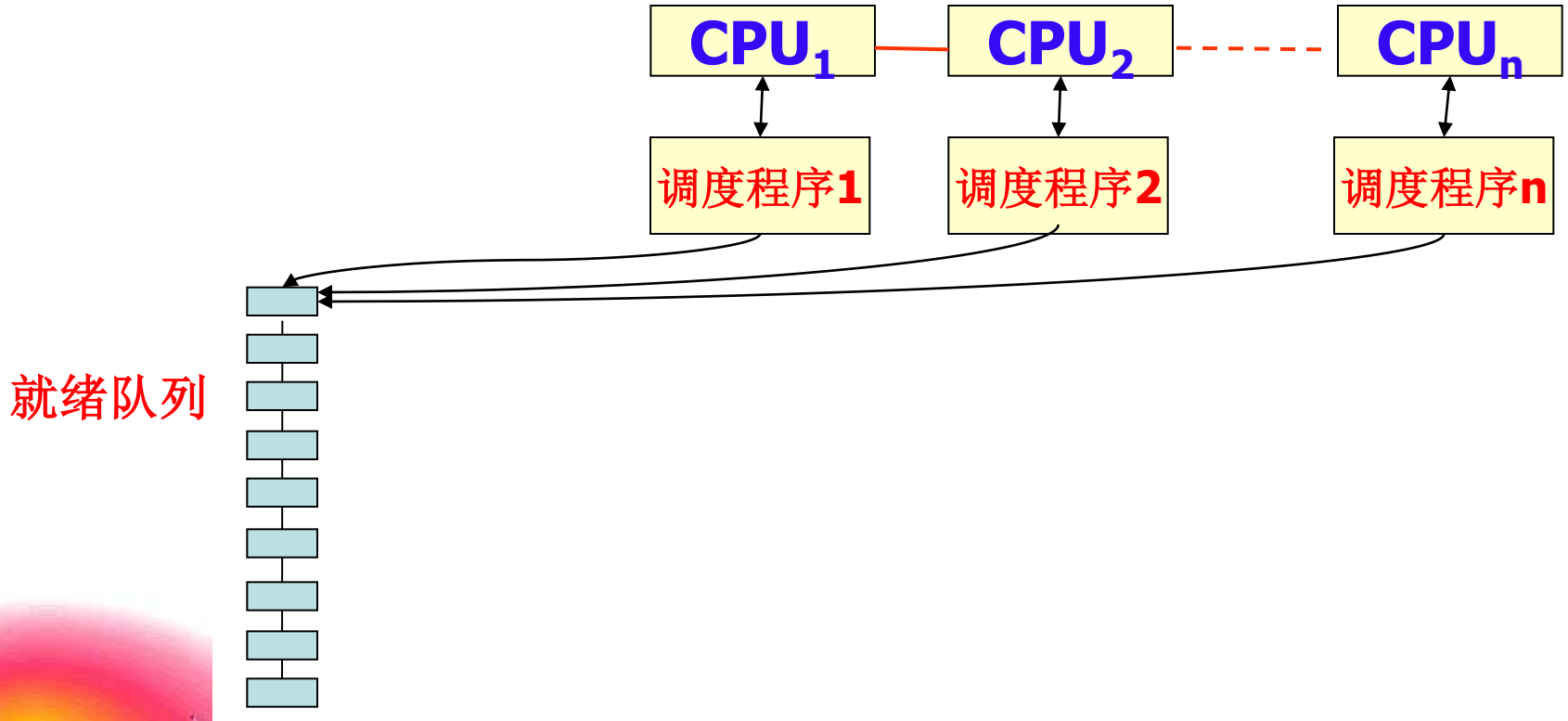
由于CPU性能的不同，使得系统需要将进程分配到适合对其进行处理的CPU上。

- 主要采用主—从式OS
- 操作系统、进程调度在主处理器上执行
- 易形成瓶颈，有潜在的不可靠性



3.4.3 进程（线程）调度方式—自调度

- 系统中只有一个公共的就绪队列。
- 各个处理机自行在就绪队列中取出任务。



3.4.3 进程（线程）调度方式—自调度

- 优点：
  - 简单，容易从单机系统下移植到多机环境
  - 多个CPU利用率都不错（不会闲）
- 缺点：
  - 瓶颈问题（单队列，CPU需互斥访问）
  - 低效性（需拷贝现场）
  - 线程切换频繁（当线程合作时,各线程并行的条件不容易满足）

3.4.3 进程（线程）调度方式—成组调度

- **提出目的：** 解决在自调度中线程频繁切换的问题
- **主要思想：** 将进程中一组线程，分配到一组CPU上执行。
- **优点：**
  （1）对相互合作的进（线）程组调度，可以减小切换，减小系统开销。
  （2）每次分配一组CPU，减少了调度频率。
- **分配CPU时间的两种方式：**
  （1）面向程序：时间以进程（应用）为单位进行划分。
  （2）面向线程：使处理机利用率更高。

3.4.3 进程（线程）调度方式—成组调度

例：系统有4台处理器，两个应用程序A、B，其中应用A包括4个线程，应用B包括1个线程。

面向程序分配

	应用A	应用B
cpu1	线程1	线程1
cpu2	线程2	空闲
cpu3	线程3	空闲
cpu4	线程4	空闲
分配时间	1/2	1/2

浪费37.5%

面向线程分配

	应用A	应用B
cpu1	线程1	线程1
cpu2	线程2	空闲
cpu3	线程3	空闲
cpu4	线程4	空闲
分配时间	4/5	1/5

浪费15%

3.4.3 进程(线程)调度方式—专用处理机

- 引入：多处理机系统，每个处理已不再属宝贵资源。
- 思想：系统为一个应用分配一组专用的CPU，每个线程一个，直到该应用执行完毕。
- 特点：每个进（线）程专用处理机，使其切换小，提高效率。
- 主要用于大型计算，实时系统

3.5 产生死锁的原因和必要条件

多道程序系统中，由于多个进程的并发执行，会出现一种死锁的现象。

死锁(Deadlock)——多个进程因争夺资源而造成一种僵局，若无外力作用，进程都将无法再向前推进。

产生死锁的原因：

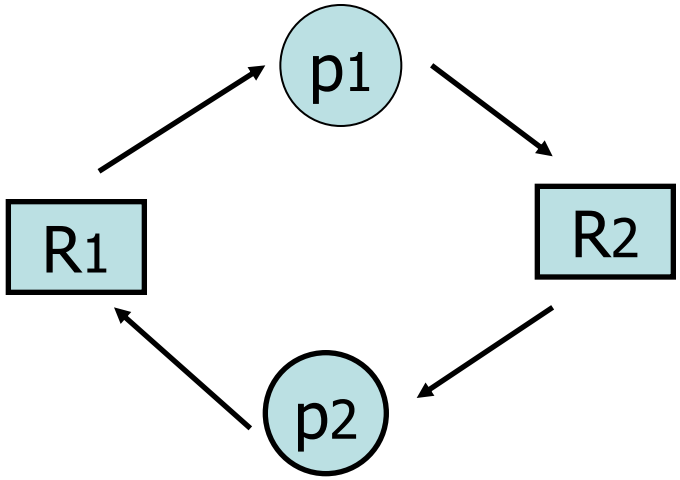
- 竞争资源
- 进程间推进顺序非法

3.5 产生死锁的原因和必要条件

一、竞争资源引起死锁

* 两种资源
  - 可剥夺资源（CPU、内存，）
  - 非剥夺性资源（打印机，磁带机）

* 竞争非剥夺性资源—可造成死锁

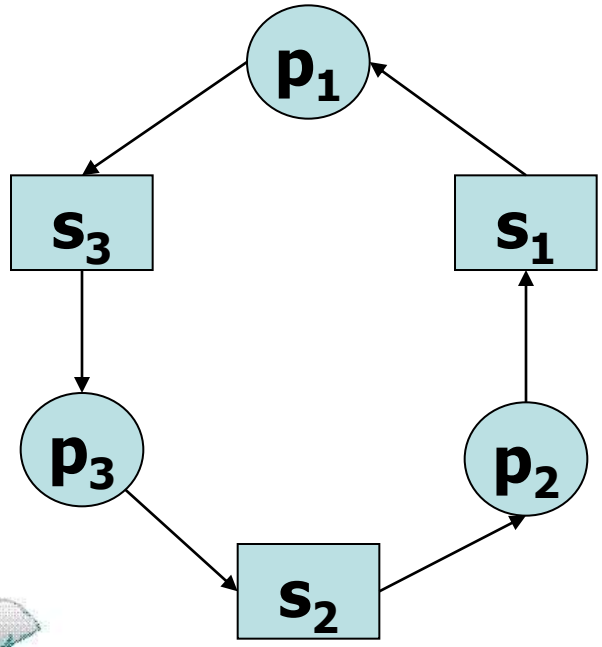


方块表示资源
圆圈表示进程
进程→资源：请求资源
资源→进程：已分配

3.5 产生死锁的原因和必要条件

*竞争临时性资源

- 临时性资源:指由一个进程产生，被另一个进程使用一段时间后便无用的资源，也称消耗资源。



顺序执行

P_1: …release(S_1); request(S_3) ;…

P_2: …release(S_2); request(S_1); …

P_3: …release(S_3); request(S_2) ;…

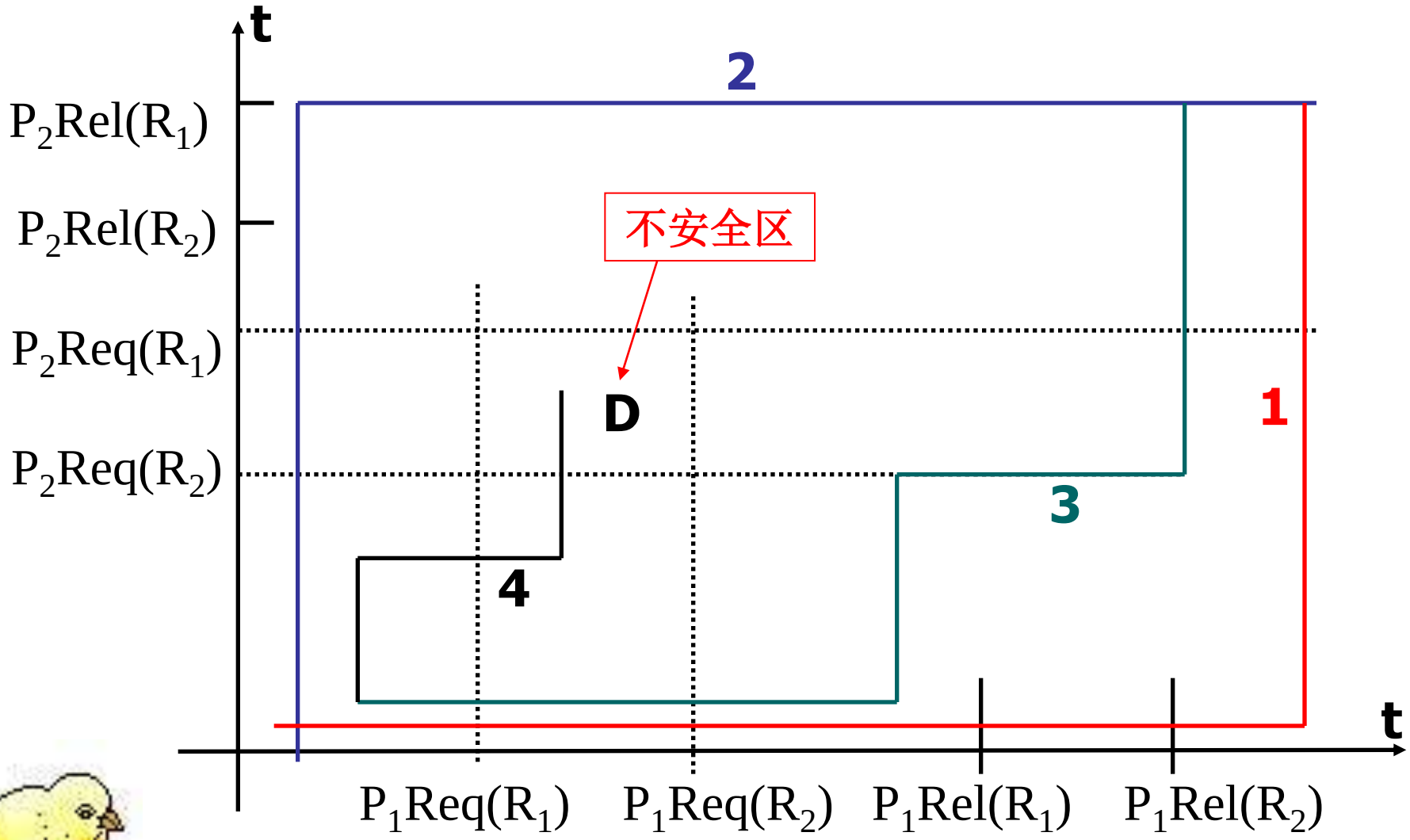
死　　锁

P_1: …request(S_3) ; release(S_1);…

P_2: …request(S_1) ; release(S_2); …

P_3: …request(S_2) ; release(S_3);…

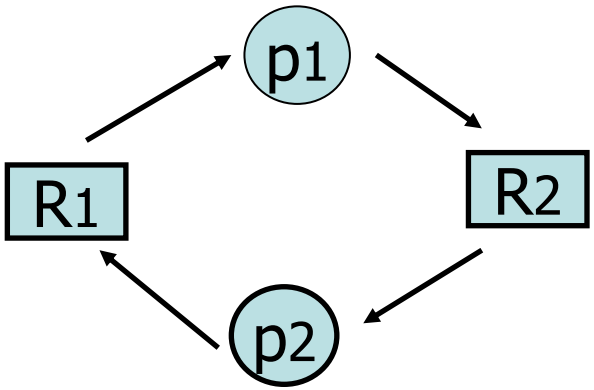
二、进程推进顺序不当引起死锁。



3.5.2 产生死锁的必要条件

产生死锁必须具备的四个必要条件：

- 互斥条件（资源的临界性）：资源独占使用
- 请求和保持条件：在保持了资源的同时，提出对新资源的请求。
- 不剥夺条件：只能有进程自己释放资源，而不能被剥夺
- 环路等待：存在一个进程—资源的环行链



3.5.3 处理死锁的基本方法

- 预防：破坏4个条件之一。有效并被广泛利用，但由于施加的限制太严格，使资源利用率低。
- 避免：防止进入不安全态。施加限制较弱，提高资源利用率，但实现复杂。
- 检测：检测到死锁再清除。不进行事先的预防。
- 解除：与“检测”配套。实现难度较大

3.6 死锁预防和避免

❖ 3.6.1 死锁预防(施加严格的限制条件)

* “互斥”条件

  多道程序运行于系统之中，必须实现资源访问的互斥。从而保证程序正确执行。保护

* 摒弃“请求和保持”条件
  - 实现方法：全分配，全释放（AND）
  - 优点：简单、安全、易于实现
  - 缺点：①延迟进程运行、②资源严重浪费

3.6 死锁预防和避免

* 摒弃“不剥夺”条件
  - 实现方法：在进程所提出的申请不能得到满足时，必须释放其占有的资源。
  - 缺点：①实现复杂、②增加系统开销、③延长了周转时间、④降低了资源利用率。

* 摒弃“环路”条件
  - 实现方法：有序资源分配法. 为资源编号，申请时需按编号进行。
  - 缺点：①新增资源不便(原序号已排定）、②用户不自由、③资源与进程使用顺序不同造成浪费

3.6.2 系统的安全状态

- “避免死锁”采用较弱的限制条件，能够有效的提高系统的性能。
- 思想：系统状态分为安全状态与不安全状态，如果处于安全状态，则系统可避免死锁。因此，避免死锁的实质在于——系统分配资源时，保证处于安全状态。
- 安全状态
  - 按某种顺序 $(P_1, P_2, \ldots, P_n)$ 并发进程都能达到获得最大资源而顺序完成的序列为安全序列。
  - 能找到安全序列的状态为安全状态。

安全状态示例

三个进程共用12台磁带机的申请状态表

进程	最大需求	已分配	可用
P1	10	5	3
P2	4	2	
P3	9	2	

安全序列：p2→p1→p3

3.6.2 系统的安全状态

安全—不安全的转换

上例中安全序列：p2→p1→p3 。若不按照安全序列分配，如P3再申请一台，则不安全。

进程	最大需求	已分配	可用
P1	10	5	2
P2	4	2	
P3	9	3	

3.6.3 利用银行家算法避免死锁

由Dijkstra提出。

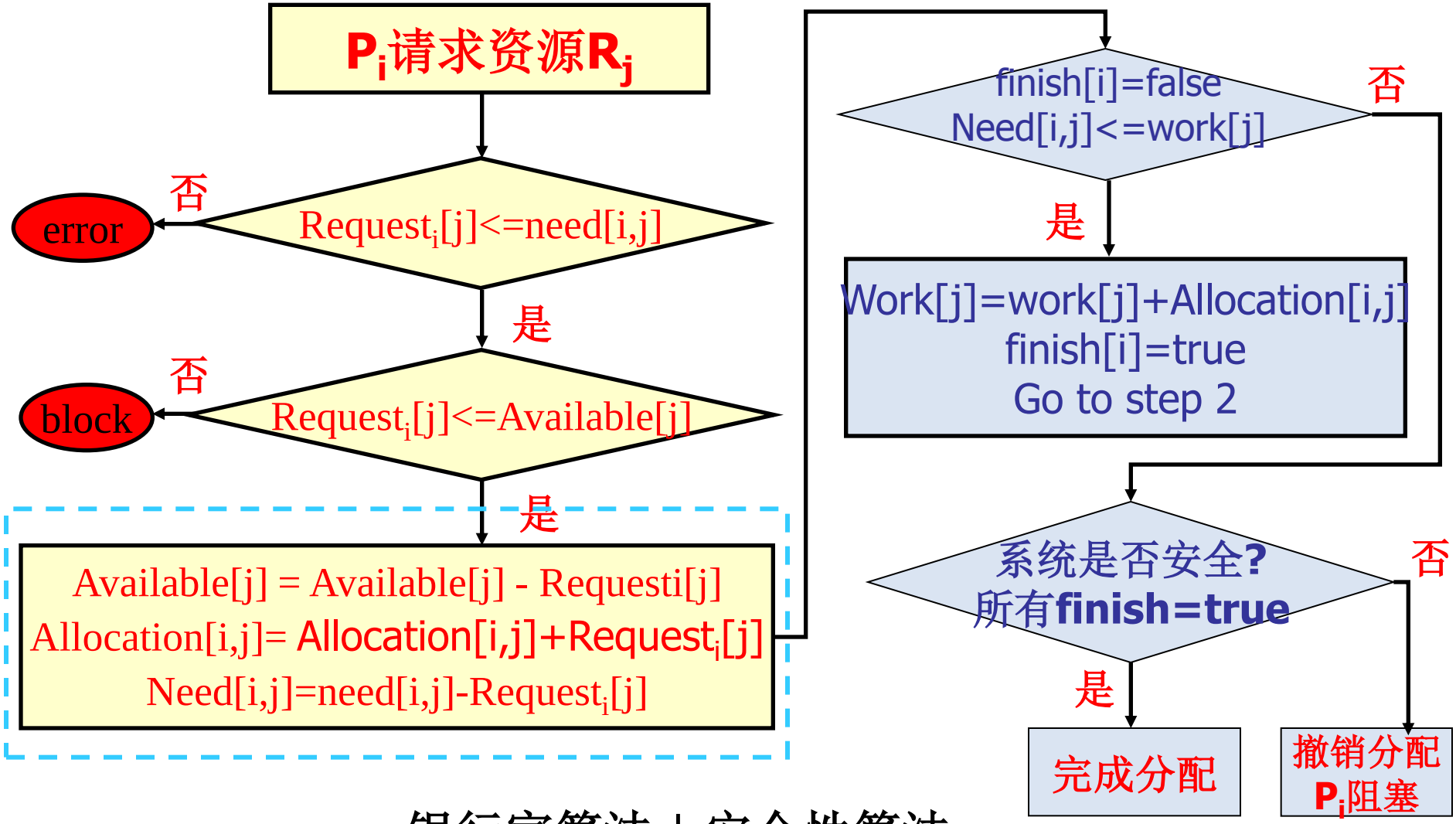
❖ 数据结构

* 可用资源向量**available[j]=k**：系统现有R_j类资源k个
* 最大需求矩阵**max[i,j]=k**：进程i需要R_j的最大数k个
* 分配矩阵**alloc[i,j]=k**：进程i已得到R_j类资源k个
* 需求矩阵**need[i,j]=k**：进程i需要还需要R_j类资源k个

need[i,j]= max[i,j]－alloc[i,j]

* 请求向量**$Request_i$[j]=k**：进程P_i请求R_j类资源k个
* **worki**：进程i执行完后系统应有资源数（也即可用数）
* **finish[i]**：布尔量，表进程i能否顺序完成。

3.6.3 利用银行家算法避免死锁



银行家算法＋安全性算法

银行家算法实例

系统状态：进程{P_0,P_1,P_2,P_3,P_4}，资源{A=10,B=5,C=7}

T_0时刻的资源分配表

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
p_0	7	5	3	0	1	0	7	4	3	3	3	2
p_1	3	2	2	2 (3	0 0	0 2)	1 (0	2 2	2 0)	(2	3	0)
p_2	9	0	2	3	0	2	6	0	0			
p_3	2	2	2	2	1	1	0	1	1			
p_4	4	3	3	0	0	2	4	3	1			

是否存在一个安全序列？

银行家算法实例

利用安全性算法来检测，如Finish均为true则系统安全。

T0时刻的安全序列

	Work A B C	Need A B C	Alloc A B C	Work+alloc A B C	Finish
p_1	3 3 2	1 2 2	2 0 0	5 3 2	true
p_3	5 3 2	0 1 1	2 1 1	7 4 3	true
p_4	7 4 3	4 3 1	0 0 2	7 4 5	true
p_2	7 4 5	6 0 0	3 0 2	10 4 7	true
p_0	10 4 7	7 4 3	0 1 0	10 5 7	true

安全序列{P_1, P_3, P_4, P_2, P_0}

银行家算法实例

P_1申请资源(1,0,2)时安全性检查(安全)

	Work A B C	Need A B C	Alloc A B C	Work+alloc A B C	Finish
p_1	2 3 0	0 2 0	3 0 2	5 3 2	true
p_3	5 3 2	0 1 1	2 1 1	7 4 3	true
p_4	7 4 3	4 3 1	0 0 2	7 4 5	true
p_0	7 4 5	7 4 3	0 1 0	7 5 5	true
p_2	7 5 5	6 0 0	3 0 2	10 5 7	true

银行家算法实例

为P_0分配（0，2，0）后的情况（不安全?）

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
p_0	7	5	3	0	3	0	7	2	3	2	1	0
p_1	3	2	2	3	0	2	0	2	0			
p_2	9	0	2	3	0	2	6	0	0			
p_3	2	2	2	2	1	1	0	1	1			
p_4	4	3	3	0	0	2	4	3	1			

3.7 死锁的检测和解除

死锁检测事先不对系统施加限制，而是采用相关的算法，来不断的检测系统是否存在死锁，如果存在则做出相应的动作以解除。由于不施加限制，所以能有效的提高系统效率。而为了检测死锁的存在，系统就必须：

- 保存资源请求和分的配信息
- 设计出检测死锁的算法
- 制定出解锁的相关策略

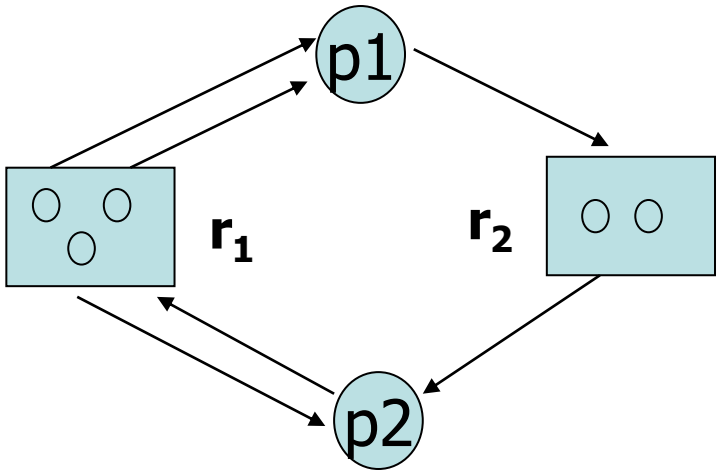
3.7 死锁的检测和解除

❖资源分配图—描述系统死锁

* 由节点和边构成G=(NE)
* 节点由进程节点和边节点构成。

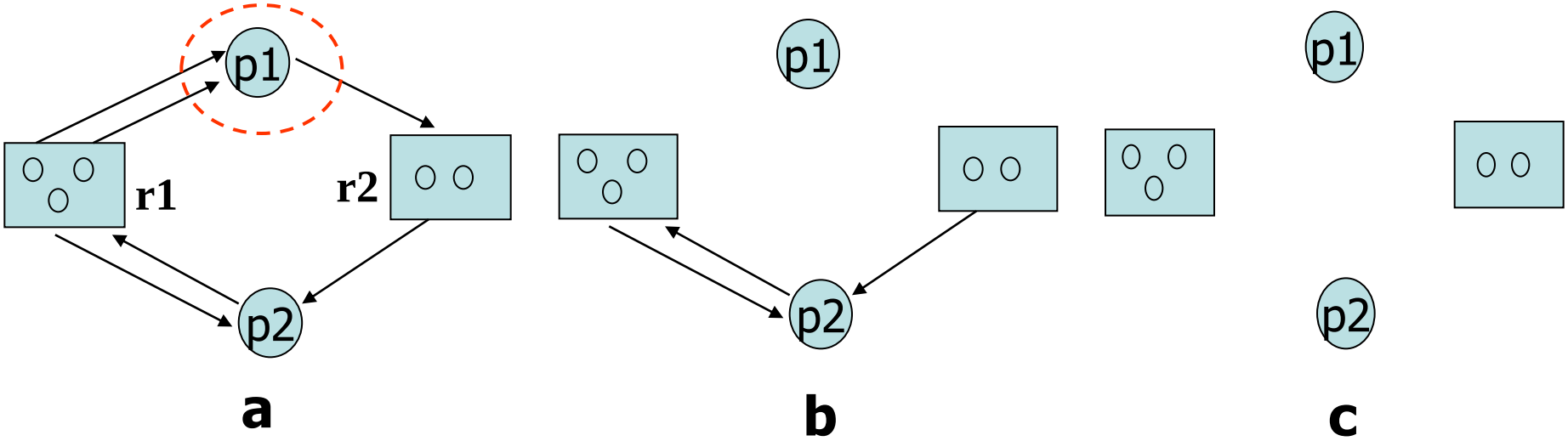
  例P={p1,p2}，R(r1,r2)
* e={p,r}　p请求一个单位的r

  e={r,p}一个r分配给p。
* 圆圈代表进程，方框代表资源。



3.7死锁的检测和解除

- 死锁定理
  - 可完全化简的资源分配图：从既不阻塞又非独立的进程节点开始，能消除所有的边，使所有进程节点都成为孤立节点。



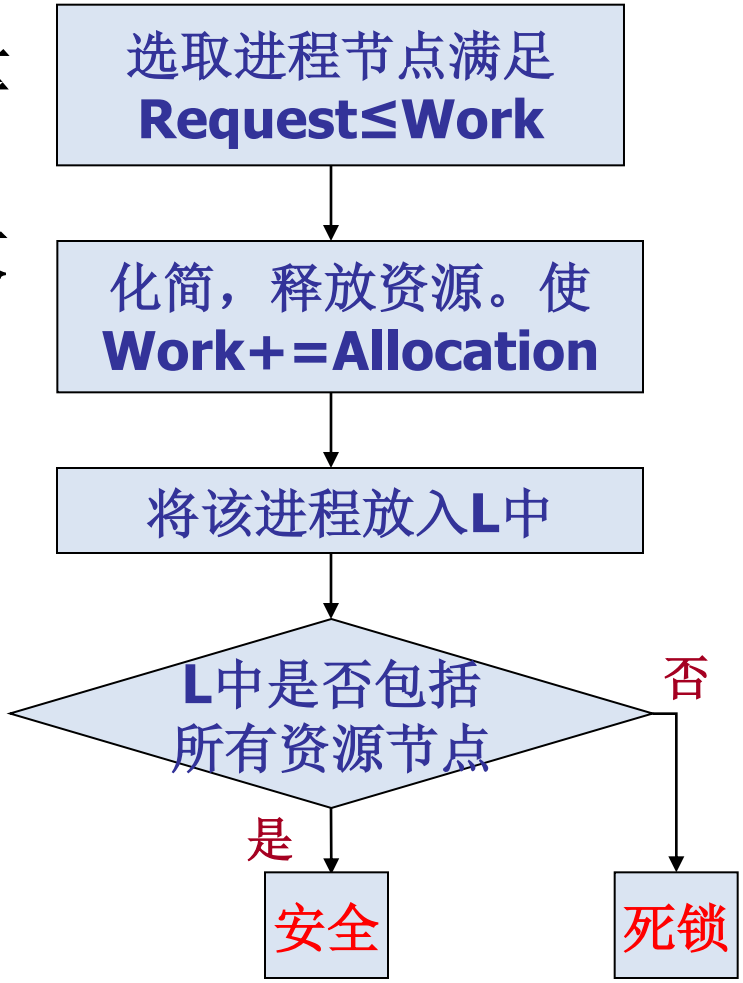
定理：S为死锁，当且仅当S状态的资源分配图是不可完全化简的！

死锁检测算法

❖ 数据结构

- Available：可利用资源向量
- Request:进程所需资源量
- Work：系统能提供的资源数
- L：孤立资源节点表

❖ 计算过程



死锁解除

❖ 死锁解除常用方法

- 资源剥夺：从其它进程剥夺足够的资源给死锁进程，解除死锁。
- 撤消进程
  - 使死锁进程都夭折
  - 撤消某些进程，直到资源满足剩下进程的需要，而解除死锁。

本章作业

课后习题（P101-P102）

1，4，6，7，11，16，17，19，20共九题