

# 第五章 死 锁

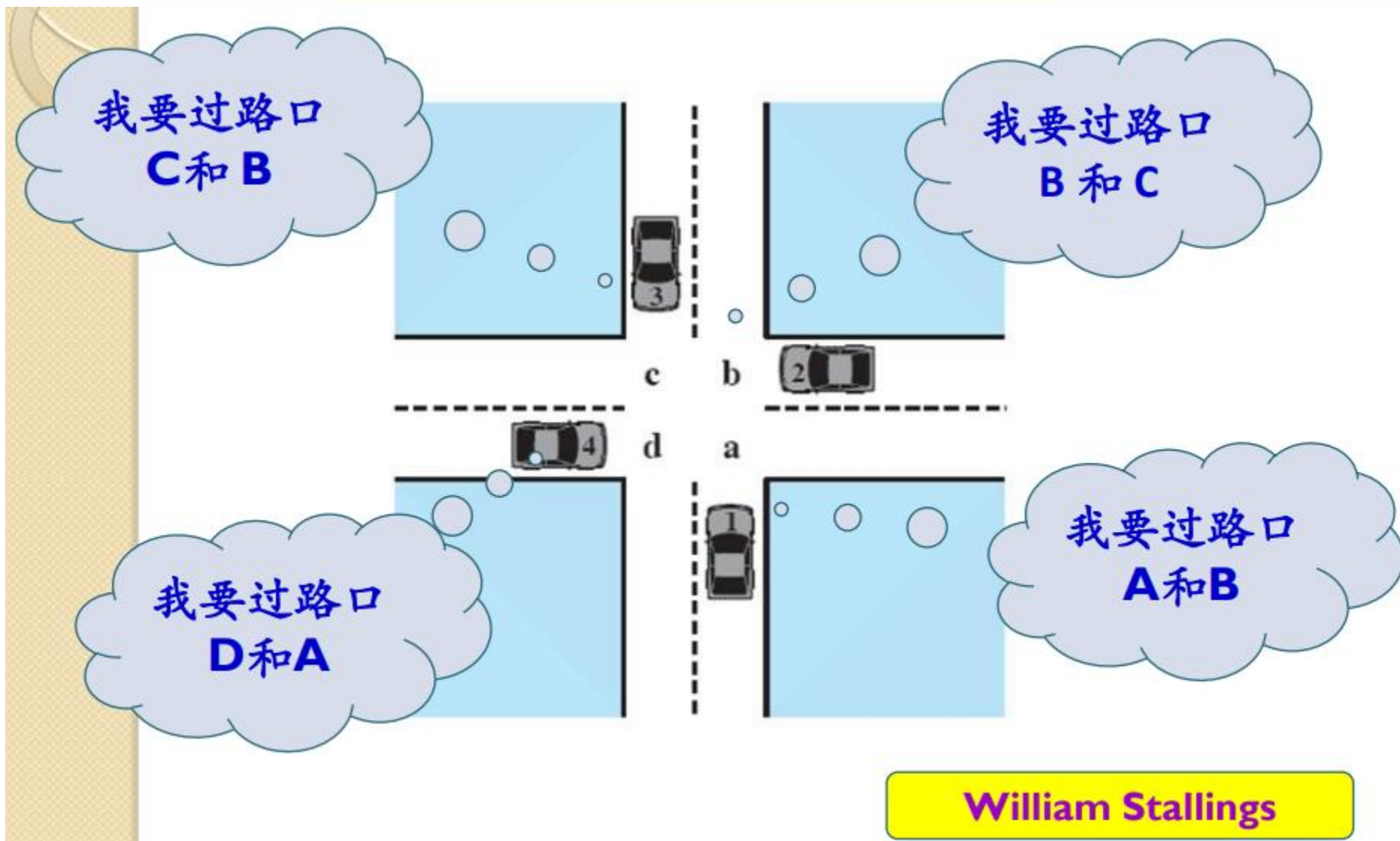
# 本章学习目标

- 死锁的基本概念。
- 死锁产生的4个必要条件。
- 处理死锁的方法。
- 死锁预防与死锁避免的区别。
- 系统安全状态及其判别方法。
- 银行家算法及其应用。
- 死锁检测、死锁解除的概念和方法。

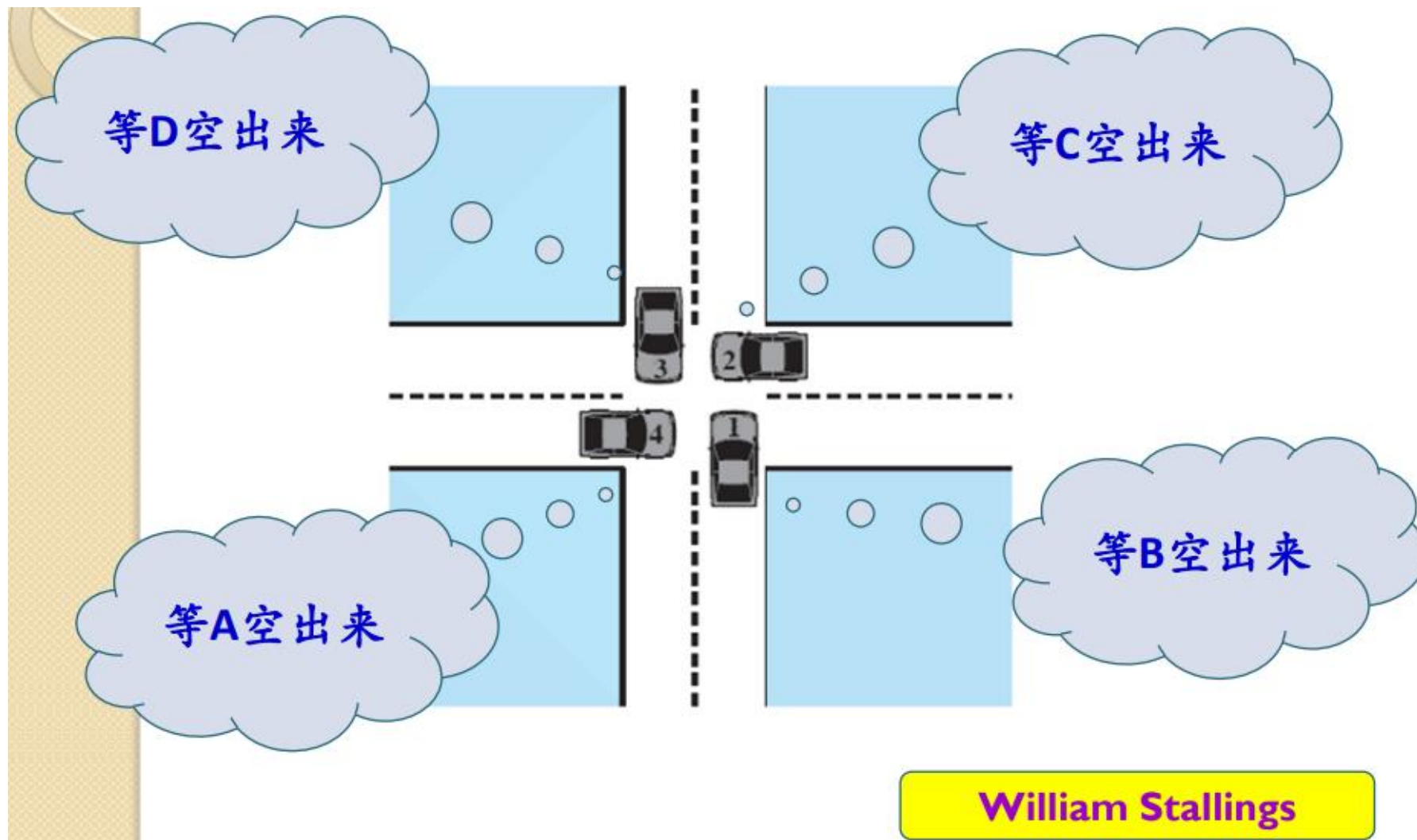
# 本章内容

- 5.1 死锁的基本概念和产生原因
- 5.2 死锁的必要条件
- 5.3 死锁的处理
- 5.4 死锁的静态预防
- 5.5 死锁的动态避免
- 5.6 死锁的检测和解除
- 5.7 线程死锁

## 5.1.1 死锁的例子

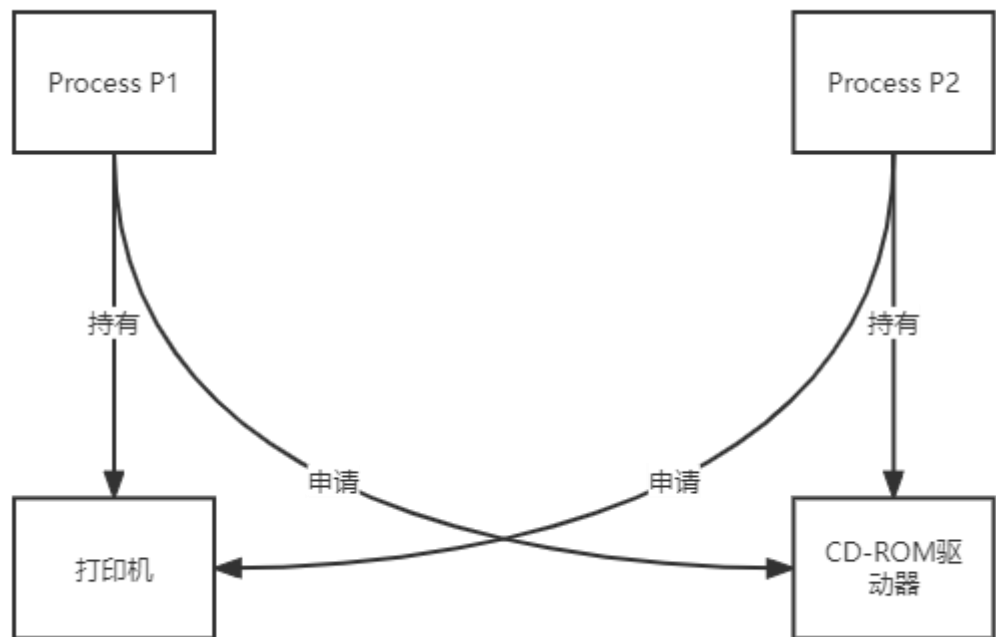


## 5.1.1 死锁的例子

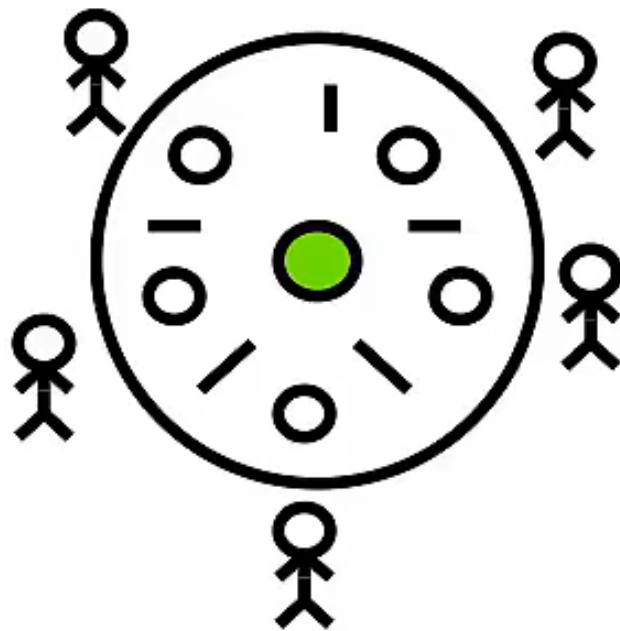


## 5.1.1 死锁的例子

进程 $P_1$	进程 $P_2$
⋮	⋮
申请打印机	申请 CD-ROM 驱动器
⋮	⋮
申请 CD-ROM 驱动器	申请打印机
⋮	⋮
释放打印机	释放 CD-ROM 驱动器
⋮	⋮
释放 CD-ROM 驱动器	释放打印机
⋮	⋮



## 5.1.1 死锁的例子



哲学家就餐问题

## 5.1.1 死锁的基本概念

- 通过上面介绍的例子可以发现，死锁具有以下特点：

① 陷入死锁的进程是系统并发进程中的一部分，且**至少要有2个进程**，单个进程不能形成死锁。

② 陷入死锁的进程**彼此都在等待对方释放资源**，形成一个循环等待链。

③ 死锁形成后，在没有外力干预下，**陷入死锁的进程不能自己解除死锁**，死锁进程无法正常结束。

④ 如不及时解除死锁，死锁进程占有的资源不能被其他进程所使用，导致系统中更多进程阻塞，造成**资源利用率下降**



## 5.1.2 产生死锁的原因

- 1. 资源竞争
- 死锁产生的根本原因是资源竞争其分配不当。因为多道程序并发执行，造成多个进程在执行中所需的资源数远远大于系统能提供的资源数
- 比如哲学家多一个筷子，多一台打印机。

## 5.1.2 产生死锁的原因

- 计算机系统中有多种资源，按照占用方式来分，可分为**可剥夺资源与不可剥夺资源**。

### (1) 可剥夺资源

某进程在获得这类资源后，即使该进程没有使用完，该类资源也可以被其他进程剥夺使用。例如：**CPU**、内存、磁盘等

### (2) 不可剥夺资源

当系统把这类资源分配给某进程后，不能强行收回，只能在进程使用完后自行释放，然后其他进程才能使用。这类资源众多，例如：打印机、刻录机、**CD-ROM**驱动器等

## 5.1.2 产生死锁的原因

- 2. 推进顺序不当

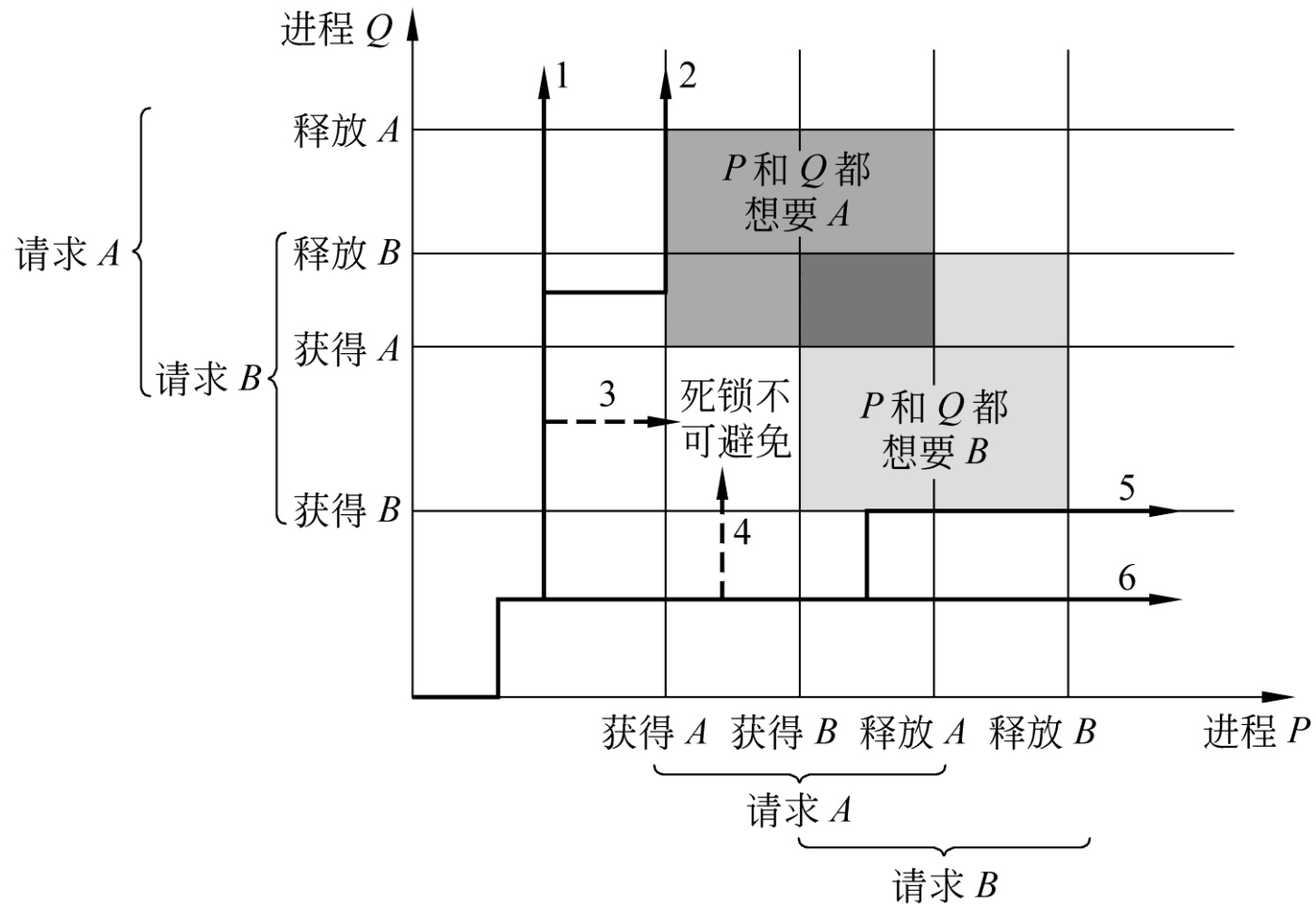
并发执行的诸进程在运行中存在**异步性**，彼此间相对执行速度不定，存在着**多种推进顺序**。并发进程间**推进顺序不当时会引起死锁**。

不可剥夺资源少未必一定产生死锁。死锁在一种很巧合的推进顺序中才会发生。在不能增加不可剥夺资源数量的前提下，我们要采取各种措施，尽量避免产生死锁的不合理推进顺序出现。

## 5.1.2 产生死锁的原因

进程 $P$	进程 $Q$
$\vdots$	$\vdots$
申请资源 $A$	申请资源 $B$
$\vdots$	$\vdots$
申请资源 $B$	申请资源 $A$
$\vdots$	$\vdots$
释放资源 $A$	释放资源 $B$
$\vdots$	$\vdots$
释放资源 $B$	释放资源 $A$
$\vdots$	$\vdots$

## 5.1.2 产生死锁的原因



## 5.1.2 产生死锁的原因

由以上分析我们得出以下两个结论：

- (1) 用户编写应用程序时或操作系统进行资源分配时应采取相应措施，**避免导致死锁发生的进程间的推进顺序出现**。
  - 。
- (2) 死锁是在特定的推进顺序中才会出现，**具有一定的隐蔽性**。

## 5.2 死锁的必要条件

- 四个必要条件（四条同时成立会产生死锁现象）：

### （1）互斥条件

一个时刻，一个资源仅能被一个进程占用。

### （2）请求和保持条件

除了资源占有进程主动释放资源，其他进程都不能抢夺其资源。

### （3）不剥夺条件

一个进程请求资源得不到满足等待时，不释放已占有资源。

### （4）环路等待条件（隐含着前三个条件）

每一个进程分别等待它前一个进程所占有的资源。

## 5.2 死锁的必要条件

- 环路等待条件隐含着前三个条件，即只有前三个条件成立，第四个条件才会成立。
- 特别注意：环路等待条件只是死锁产生的必要条件，而不是等价定义。死锁一旦产生则死锁进程间必存在循环等待环，但存在循环等待环不一定产生死锁。例如：某系统中有两个R1资源和一个R2资源。假设系统中有三个进程并发执行，存在一个环路，进程P1等待P2所占有的资源R1，进程P2等待P1所占有的资源R2，此时进程P3占有另一个R1资源，显然进程P1和进程P2已陷入死锁。但是，如果进程P3以后的执行过程中没有提出新的关于R1或R2的请求，且顺利执行完毕，释放其占有的资源R1给进程P1，则P1和P2形成的循环等待环将被打破，死锁解除。



## 5.3 死锁的处理

- **5.3.1 死锁的处理方法**
- 按照死锁处理的时机划分，可把死锁的处理方法分成四类。

不允许出现死锁

- (1) 预防死锁
- (2) 避免死锁

允许出现死锁，但是会尽快检测到并恢复

- (3) 检测死锁
- (4) 解除死锁

不处理死锁，假装看不到

鸵鸟算法

## 5.3 死锁的处理

- 5.3.1 死锁的处理方法

- ① 死锁的静态预防 (**Prevention**)

- 破坏四个必要条件之一

- ② 死锁的动态避免 (**Avoidance**)

- 允许四个必要条件同时存在，在并发进程中做出妥善安排避免死锁的发生

- ③ 死锁的检测和恢复 (**Detection and Recovery**)

- 允许死锁的发生，系统及时地检测死锁并解除它

- **prevention** 和 **avoidance**的区别:

**prevent** 阻止事情发生, **avoid** 事情已经发生, 如何避开/避免。

## 5.4 死锁的静态预防

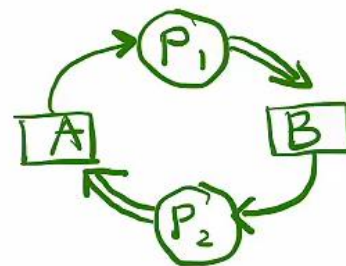
- 破坏死锁任一必要条件

- 互斥使用（不现实，比如打印机资源）

- 不可剥夺（不现实，比如打印机资源）

- 占有和等待（比如哲学家同时拥有两个筷子的时候才能吃饭，会增加系统消耗）

- 循环等待（进程必须按照特定顺序执行，比如哲学家左手右手筷子的先后顺序，会增加系统消耗，资源利用率下降，实现复杂）



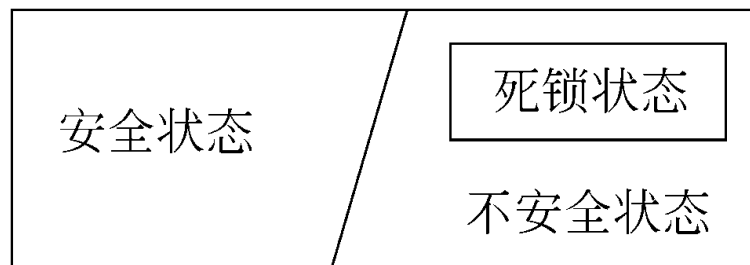
## 5.5 死锁的动态避免

- 死锁的动态避免是通过安全算法，实时判断当前决策是否会让系统陷入不安全状态。
- 如果安全算法判断当前决策/操作会让系统进入不安全状态，则将申请资源进程阻塞。

## 5.5.1 系统安全状态

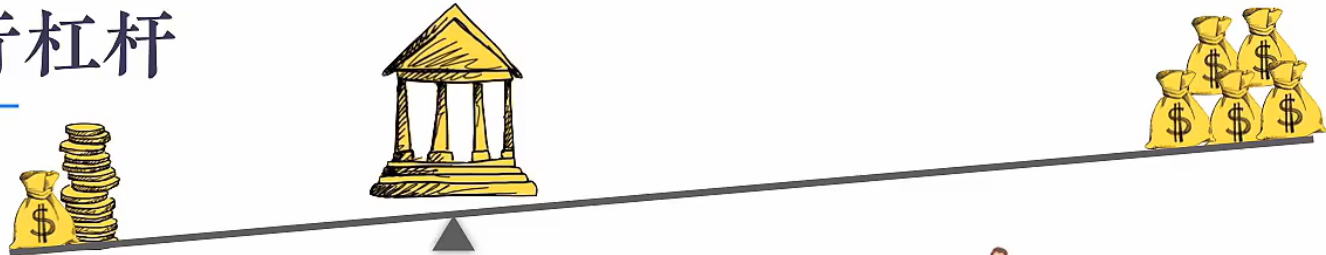
- 所谓**安全状态**是指操作系统能够按照某种进程执行序列，如 $\langle P_1, P_2, \dots, P_n \rangle$ ，为进程分配所需资源，使得每个进程都能执行完毕，此时我们称系统处于系统安全状态。进程执行序列 $\langle P_1, P_2, \dots, P_n \rangle$ 为当前系统的一个安全序列。如果操作系统无法找到这样一个安全序列，则称当前系统处于不安全状态。
- 处于不安全状态的系统一定会形成死锁吗？

并非所有的不安全状态都会导致死锁状态。但当系统进入不安全状态时，便有了导致死锁状态的可能。如果能保证每次资源分配后，系统都处于安全状态，则不会发生死锁。



# 5.5.1 系统安全状态

## 银行杠杆



capital: 10

Rules: the customers will return the loan if their credits reach the limit, otherwise they would not return the money occupied before.



credit used

3

credit limit: 5



credit used

4

credit limit: 8



credit used

0

credit limit: 7

## 5.5.2 银行家算法

- 银行家算法的基本思想是：在资源分配前，资源分配程序计算资源分配后系统是否处于安全状态，如处于安全状态则把资源分配给申请进程，如处于不安全状态则令申请资源的进程阻塞，不响应其资源申请。
- 银行家算法的核心理念是把资源分配给那些最容易执行完成的进程，保证系统中各个进程最终都能正常完成。
- 银行家算法的本质是死锁的避免，由于它并没有破坏产生死锁的4个必要条件之一，不属于死锁预防。

## 5.5.2 银行家算法

### 银行家算法-数据结构

- Available: 当前系统中可用资源数量
- Max: 每个进程的最大资源需求量
- Allocation: 已经分配给进程的资源数量
- Need: 每个进程还需要的资源数量

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
$P_0$	0 1 0	7 5 3	3 3 2
$P_1$	2 0 0	3 2 2	
$P_2$	3 0 2	9 0 2	
$P_3$	2 1 1	2 2 2	
$P_4$	0 0 2	4 3 3	



## 5.5.2 银行家算法

### 安全算法

Available

A B C  
3 3 2

 寻找安全序列过程

	<u>Allocation</u>	<u>Need</u>
	A B C	A B C
$P_0$	0 1 0	7 4 3
$P_1$	2 0 0	1 2 2
$P_2$	3 0 2	6 0 0
$P_3$	2 1 1	0 1 1
$P_4$	0 0 2	4 3 1

## 5.5.2 银行家算法

表 4.3  $T_0$  时刻各进程的资源分配图

	Allocation			Max			Need			Available		
	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$
$P_1$	1	0	0	5	3	2	4	3	2	2	2	3
$P_2$	4	1	2	7	3	4	3	2	2			
$P_3$	1	0	1	3	1	4	2	0	1			

假设系统中有**3**类资源  $\{r_1, r_2, r_3\}$  和**3**个并发执行进程  $\{P_1, P_2, P_3\}$ ，其中 $r_1$ 有**8**个， $r_2$ 有**3**个， $r_3$ 有**6**个。假设在  $T_1$ 时刻，进程 $P_1$ 提出请求 **Request = (1,0,1)**，能否实施资源分配，为什么？并给出安全序列。

## 5.5.2 银行家算法

表 4.4  $T_0$  时刻的一个安全序列

	Work			Need			Allocation			Work+Allocation			Finish
	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$	
$P_3$	2	2	3	2	0	1	1	0	1	3	2	4	true
$P_2$	3	2	4	3	2	2	4	1	2	7	3	6	true
$P_1$	7	3	6	4	3	2	1	0	0	8	3	6	true

**Work** 矩阵为系统可提供给进程继续运行的各类资源数目；  
**Finish** 代表当前进程是否满足需求，得到执行。

表 4.5  $T_1$  时刻各进程的资源分配图

	Allocation			Max			Need			Available		
	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$	$r_1$	$r_2$	$r_3$
$P_1$	2	0	1	5	3	2	3	3	1	1	2	2
$P_2$	4	1	2	7	3	4	3	2	2			
$P_3$	1	0	1	3	1	4	2	0	1			

## 5.5.2 银行家算法

银行家算法的优缺点：

优点：

- 允许死锁必要条件同时存在

缺点：

- 缺乏实用价值
- 进程运行前要求知道其所需资源的最大数量
- 要求进程是无关的，若考虑同步情况，可能会打乱安全序列
- 要求进入系统的进程个数和资源数固定

## 5.6 死锁的检测与恢复

- ④ 允许死锁发生，操作系统不断监视系统进展情况，判断死锁是否发生
- ④ 一旦死锁发生则采取专门的措施，解除死锁并以最小的代价恢复操作系统运行
- ④ 死锁检测的时机
  - ④ 当进程等待时检测死锁（系统开销大）
  - ④ 定时检测
  - ④ 系统资源利用率下降时检测死锁

## 5.3.2 资源分配图

资源类（资源的不同类型）

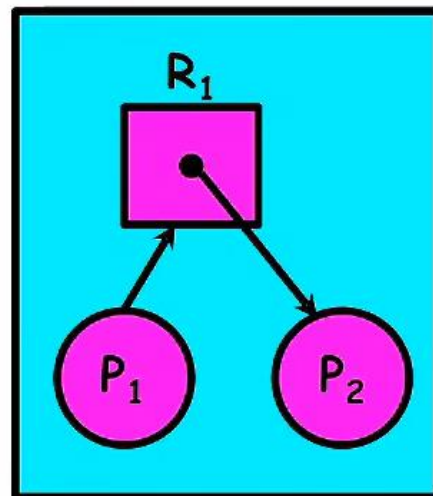
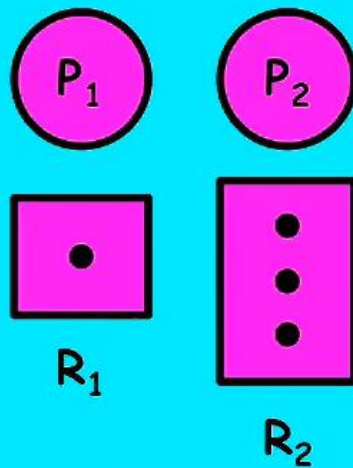
资源实例（存在于每个资源中）

进程

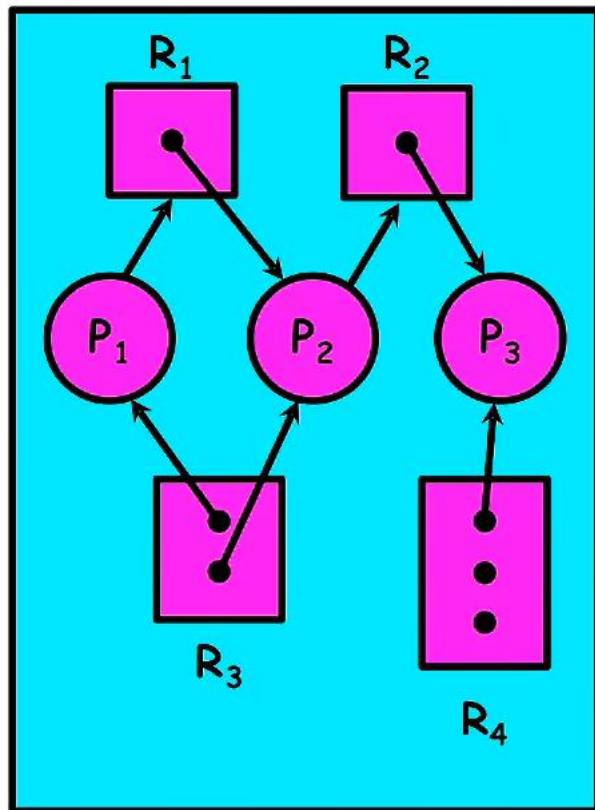
申请边

分配边

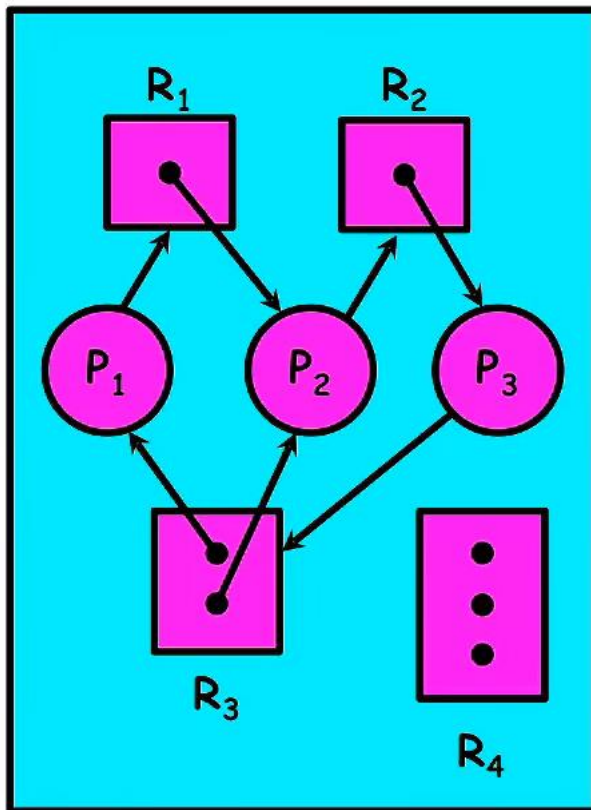
表示符号



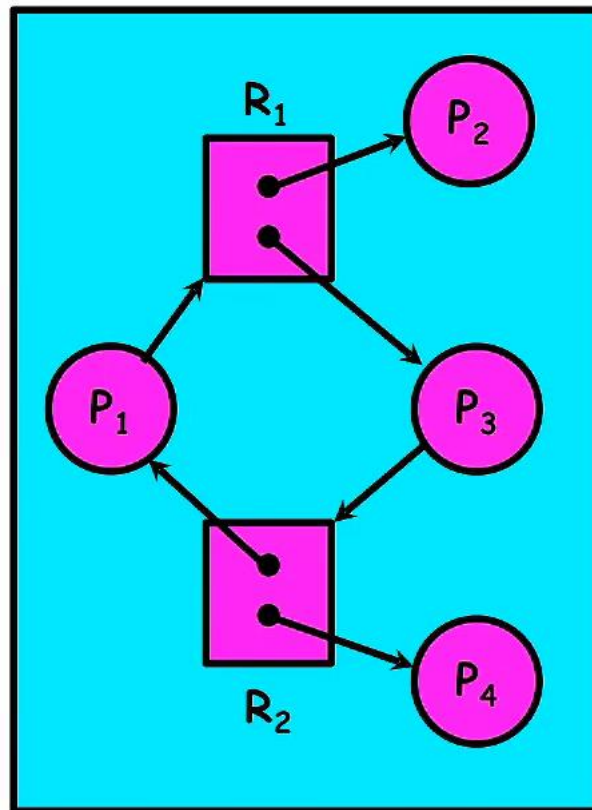
## 5.3.2 资源分配图



无环无死锁



有环死锁

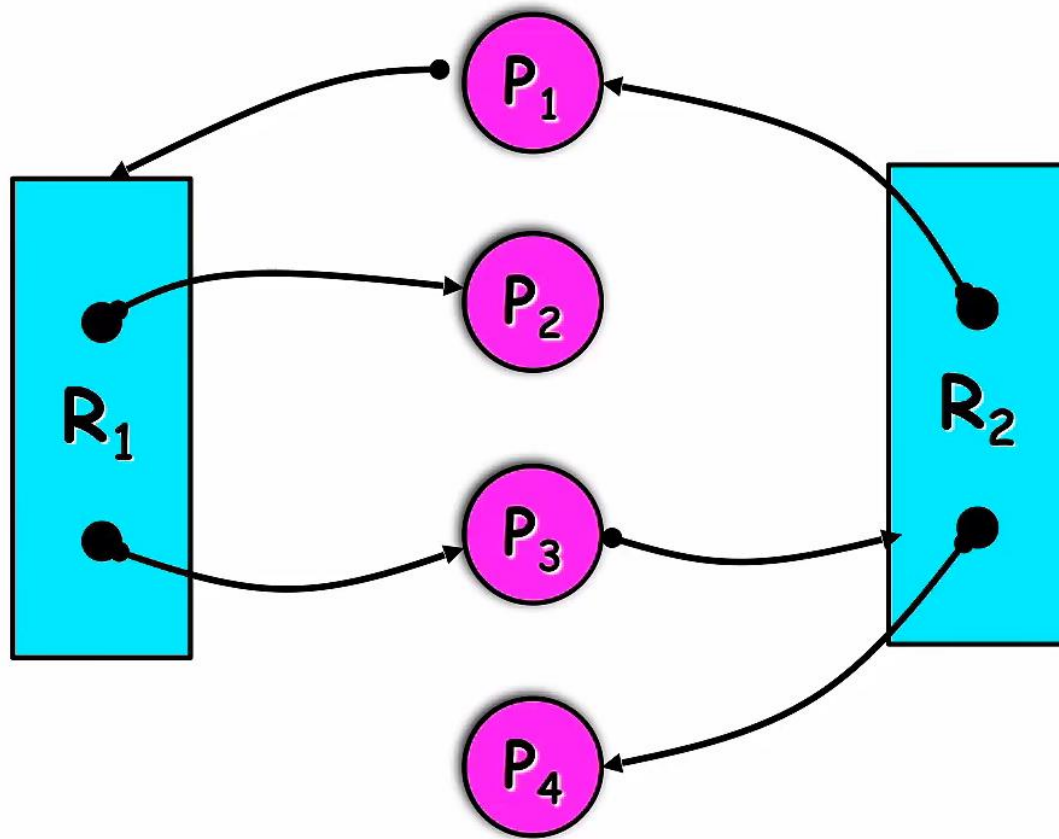


有环但无死锁



## 5.3.2 资源分配图

### 资源分配图的简化







## 5.6 死锁的检测

- ④ 如果能在“资源分配图”中消去某进程的所有请求边和分配边，则称该进程为**孤立结点**。
- ④ 可完全简化
- ④ 不可完全简化
- ④ 系统为死锁状态的充分条件是：当且仅当该状态的“进程—资源分配图”是不可完全简化的。该充分条件称为**死锁定理**。




## 5.6.3 死锁解除方法

### 中止进程，强制回收资源

-  交通问题：将某列火车吊起来
-  哲学家问题：将某个哲学家射死

### 剥夺资源，但不中止进程

### 进程回退(roll back)

-  就像DVD的回退，好像最近一段时间什么都没有发生过
-  交通问题：让某列火车倒车
-  哲学家问题：让某个哲学家放下一把叉子

### 重新启动

-  没有办法的办法，但却是一个肯定有效的办法

## 5.6.4 鸵鸟算法

- 据说鸵鸟看到危险动物时就把头埋在沙砾中，装作看不到。当人们对某一件事情没有一个很好的解决方法时，或者解决问题的方法代价太大而得不偿失时，人们就借鉴鸵鸟的办法，忽略问题的存在。
- 在死锁问题上，鸵鸟算法就是不对死锁采取任何处理方法。著名的**UNIX**、**Linux**和**Windows**操作系统在分析了死锁发生的频率、系统因各种原因崩溃的频率、死锁的严重程度以及解决死锁问题的代价之后都采用了鸵鸟算法。
- 采用鸵鸟算法的代价就是用户必须忍受死锁带来的诸多不便。
- 但实际上，经过大量统计，死锁发生的概率是很小的，只有在极偶然的情况下才会出现死锁。
- 鸵鸟算法也是操作系统设计者处理死锁问题的一种不错选择。