西安电子科技大学
XIDIAN UNIVERSITY

计算机科学与技术学院
SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY
国家示范性软件学院
NATIONAL PILOT SCHOOL OF SOFTWARE ENGINEERING

操作系统原理

第四章 死 锁

主讲：黄伯虎

第三章 小结

基本内容	一、进程的基本概念 1. 定义，进程和程序的区别，进程的组成，PCB的结构 2. 进程的基本状态及其转换 二、进程调度算法 1. 时间片轮转法(RR) 2. 多级反馈队列调度算法(MFQS) 三、进程间的相互作用 同步，互斥；信号量和P、V操作，管程 四、进程通信 共享内存模式；消息传递模式；共享文件模式。
重难点	• 进程概念的理解 • 进程与程序的区别 • 进程的基本状态及转换 • P、V操作和信号量机制

引言

主要内容

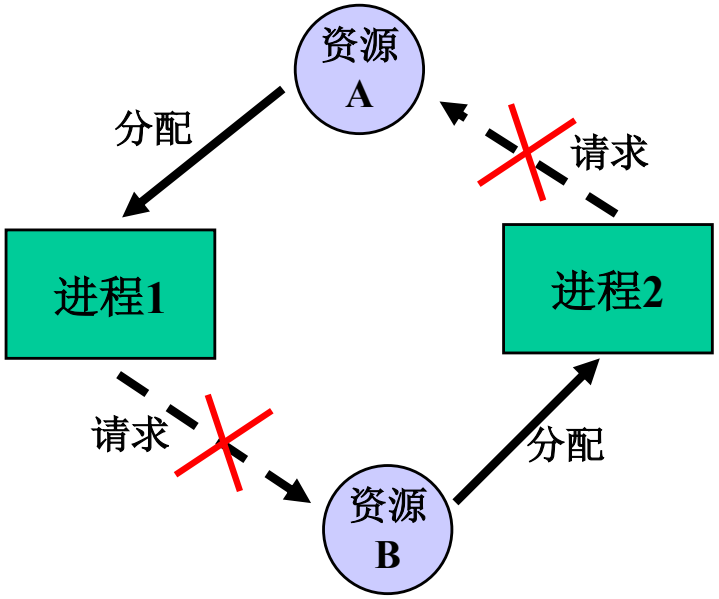
- 本章主要阐述死锁这一并发进程运行期间可能会出现的严重问题，并通过对其产生原因和条件的分析，给出解决死锁问题的一系列方法。
- 本章是进程管理的延伸，篇幅不大，但知识点较多。

一、死锁的基本概念

交通堵塞

一、死锁的基本概念

死锁(Deadlock)的定义



在多道程序中，由于多个并发进程共享系统资源，如果使用不当可能会造成一种僵局，即当某个进程提出资源的使用请求后，使得系统中一些进程处于无休止的阻塞状态，在无外力的作用下，这些进程将无法继续执行下去，这就是死锁。

二、死锁产生的环境和条件

马路堵车的（客观）因素
马路上各个方向可以同时行驶多辆汽车
马路上有很多汽车
路口是一种互斥共享的资源
没有交警

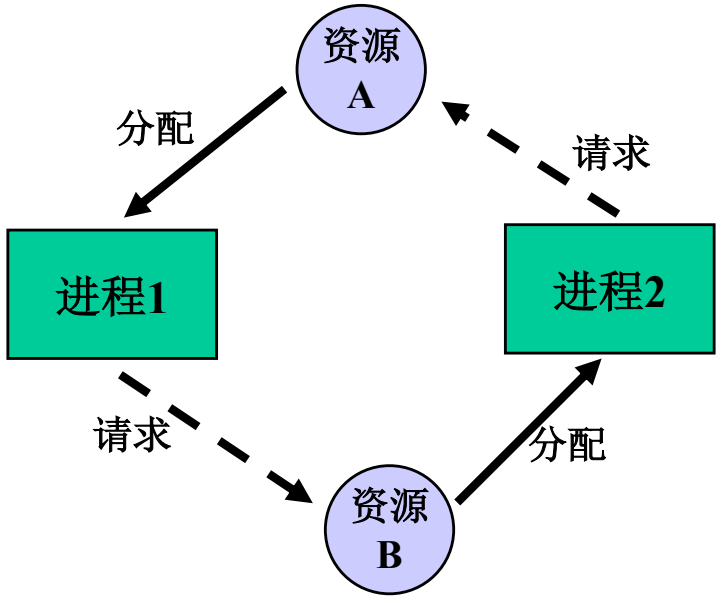
计算机系统产生死锁的环境
并发环境（多道程序设计技术）
多个并发进程
资源共享和独占
没有外力可以借助

交通可能会堵塞

系统可能会死锁

二、死锁产生的环境和条件

- 死锁产生的必要条件(Coffman 1971年提出)
  - 资源互斥使用(资源独占)
  - 非剥夺控制(不可强占)
  - 零散请求与保持
  - 循环等待



- 死锁的危害
  - 一旦发生：轻则系统资源利用率下降，重则系统崩溃。

二、死锁产生的环境和条件

死锁到底该如何处理？

三、死锁的解决策略

置之不理法——鸵鸟政策

优点：设计简单，零成本。

缺点：安全性，稳定性欠佳。

事后处理法——让死锁发生，事后处理

- 思想：可以容忍死锁的发生，事后处理。
- 优点：灵活，但不是所有情况都能容忍死锁发生的。

三、死锁的解决策略

积极防御法——不让死锁发生

- 思想：以积极的遏制为出发点。
- 缺点：成本较高，代价较大。
- 手段：
  - 死锁的预防：
    - 通过某种手段，使得死锁不可能发生。
    - 什么手段？
  - 死锁的避免：
    - 允许存在发生死锁的可能性，但每走一步小心翼翼，使得永远达不到死锁状态。
    - 怎么个小心翼翼？

四、死锁的预防

方法：破坏死锁产生的必要条件

破坏互斥条件

- 如何做？
- 局限：“互斥”条件的破坏往往困难，而且对很多资源行不通。因此不是一种好的方案。

破坏不可剥夺条件

- 如何做？

  允许一个进程还未执行完成时释放已经占有的资源(被剥夺使用权)。

- 局限：实现困难，为了恢复现场需要耗费很多时间和空间。会使被剥夺资源的进程蒙受损失。因此只适合类似CPU、存储器这样的资源。

四、死锁的预防

❖ 破坏零散请求条件

- ➢ 如何做？

  进程创建时就由系统分配了所有需要的资源，然后才执行，并且以后没有资源申请要求，进程执行完后，释放资源。

- ➢ 局限：系统效率低，资源浪费严重，并发性下降。

❖ 破坏循环等待条件

- ➢ 如何做？

  给资源编号，进程可在任何时刻提出资源申请，但必须按序申请。

- ➢ 局限：资源编号困难；资源的编号很难和进程申请资源的顺序一致。

四、死锁的预防

结论

- 死锁的预防是以破坏死锁产生的必要条件为基本方法，从而防止死锁发生的。由于对资源的申请加上了诸多的限制，因此这种策略虽有一定的效果，但其资源的利用率和效率比较低，很难令人满意。

五、死锁的避免

思想

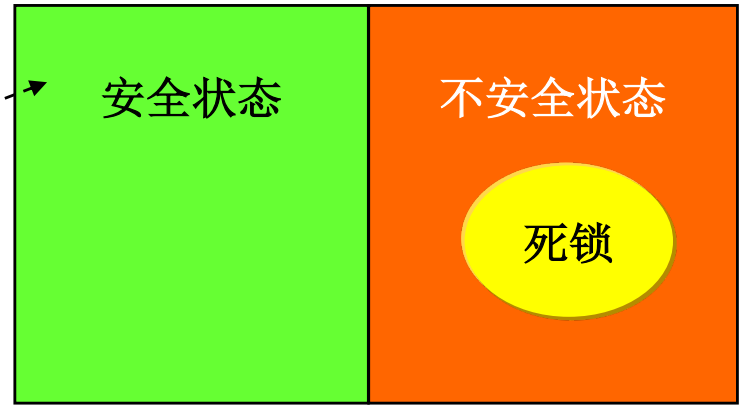
- 允许死锁产生的条件存在(可能会发生死锁)，但通过动态的、明智的选择——在分配资源之前，系统判断假若满足进程的要求是否会发生死锁，如果会，资源就不予分配，从而确保永远不会到达死锁点，避免死锁的发生。

- 优点：比预防策略更为灵活实用，允许更多的并发，其资源利用率和效率也更高。

五、死锁的避免

系统的状态

安全状态：指在某个时刻，当多个进程动态的申请资源时，如果存在一种顺序，使得系统按照这种顺序逐次地为每个进程分配所需资源后，每个进程都可以最终得到最大需求量，依次顺利地完成。

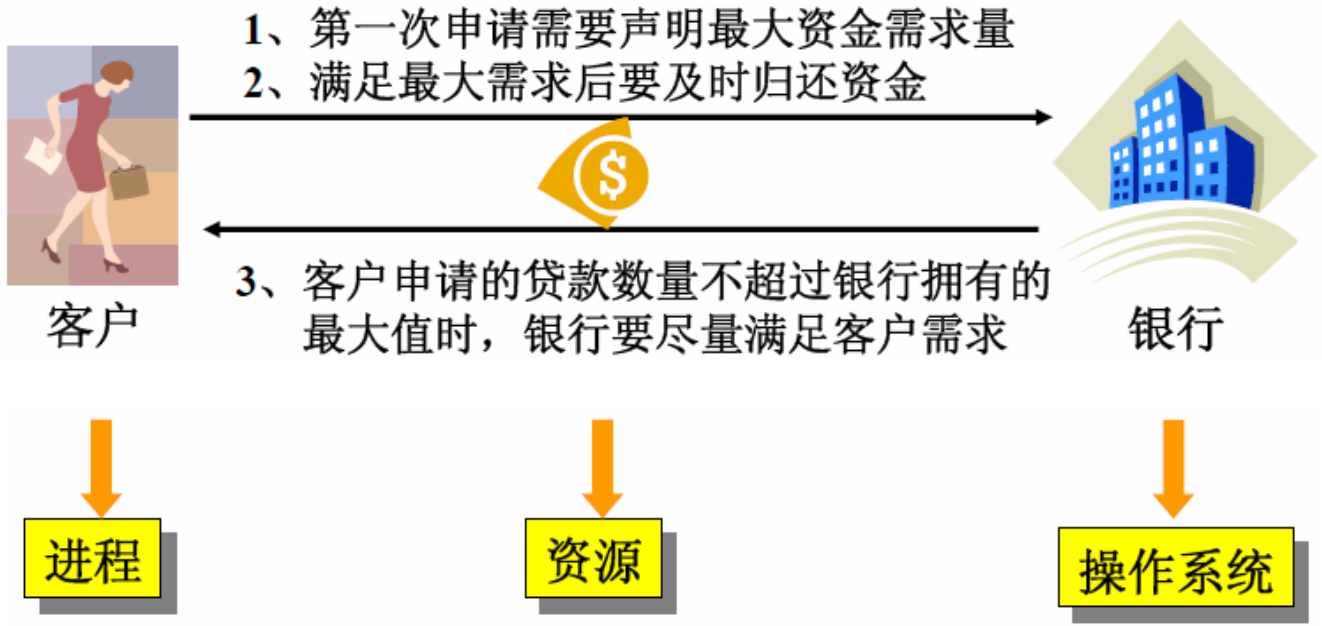


避免死锁的关键就是：让系统在动态分配资源的过程中，不要进入不安全状态（寻找一种资源的分配顺序）。

五、死锁的避免

单银行家算法(Banker's Algorithm)

- 1965年由Dijkstra设计。
- 基本思想：借用了银行借贷系统的分配策略。基于这样一些规则：



五、死锁的避免

- 举例：假设一个银行拥有资金数量为10（单位省略），现在有4个客户a, b, c, d要来贷款，所需最大资金需求量为8, 5, 6, 7，假设某时刻*t*已经为上述4个客户分别分配了1,2,1,2个单位资金，请问4个客户剩余的资金是否可以得到满足？应以何种顺序满足？

客户	已用资金	最大需求	仍需资金
a	0	8	8
b	0	5	5
c	0	6	6
d	0	7	7
银行剩余资金		10	

初始状态

客户	已用资金	最大需求	仍需资金
a	1	8	7
b	2	5	3
c	1	6	5
d	2	7	5
银行剩余资金		4	

状态1

五、死锁的避免

客户	已用资金	最大需求	仍需资金
a	1	8	7
b	5	5	0
c	1	6	5
d	2	7	5
银行剩余资金		1	

状态2

客户	已用资金	最大需求	仍需资金
a	1	8	7
b	—	—	—
c	1	6	5
d	2	7	5
银行剩余资金		6	

状态3

客户	已用资金	最大需求	仍需资金
a	1	8	7
b	—	—	—
c	6	6	0
d	2	7	5
银行剩余资金		1	

状态4

客户	已用资金	最大需求	仍需资金
a	1	8	7
b	—	—	—
c	—	—	—
d	2	7	5
银行剩余资金		7	

状态5

五、死锁的避免

客户	已用资金	最大需求	仍需资金
a	8	8	0
b	—	—	—
c	—	—	—
d	2	7	5
银行剩余资金		0	

状态6

客户	已用资金	最大需求	仍需资金
a	—	—	—
b	—	—	—
c	—	—	—
d	2	7	5
银行剩余资金		8	

状态7

客户	已用资金	最大需求	仍需资金
a	—	—	—
b	—	—	—
c	—	—	—
d	7	7	0
银行剩余资金		3	

状态8

客户	已用资金	最大需求	仍需资金
a	—	—	—
b	—	—	—
c	—	—	—
d	—	—	—
银行剩余资金		10	

状态9

分配序列：b→c→a→d

五、死锁的避免

多项资源银行家算法

- 适用于一个进程申请多个资源的情况。
- 举例：系统中有以下资源：5台打印机，7个手写板，8台扫描仪，9个读卡器，共有5个进程T1、T2、T3、T4、T5共享这些资源。各进程所需最大资源量和当前各进程已经得到的资源数量如下图，问如果进程T2此时希望得到1台打印机,1个手写板,2个读卡器是否可以满足?

进程	R1	R2	R3	R4
T1	2	4	3	1
T2	2	2	0	5
T3	1	5	5	0
T4	5	0	1	3
T5	0	3	3	3
合计	10	14	12	12

各进程所需最大资源量

进程	R1	R2	R3	R4
T1	0	1	2	1
T2	1	1	0	2
T3	0	3	4	0
T4	2	0	0	1
T5	0	0	1	3
合计	3	5	7	7

假设分配状态

进程	R1	R2	R3	R4
T1	0	1	2	1
T2	0	0	0	0
T3	0	3	4	0
T4	2	0	0	1
T5	0	0	1	3

当前各进程已分配资源

为方便讨论，我们用向量来表示资源分配及占用情况：

1、sum向量：表示系统资源总量

sum = （5，7，8，9）

2、allocation向量：表示当前系统已分配资源

allocation =（3，5，7，7）

3、available向量：表示系统剩余资源

available = sum – allocation =（2，2，1，2）

五、死锁的避免

和单个进程相关的向量：
- **sum(i)**：表示第i个进程资源需求总量；
- **allocation(i)**：表示第i个进程已分配资源总量；
- **claim(i)向量**：表示第i个进程还需申请资源数

claim(i) = *sum(i) – allocation(i)*

进程	R1	R2	R3	R4
T1	2	4	3	1
T2	2	2	0	5
T3	1	5	5	0
T4	5	0	1	3
T5	0	3	3	3
合计	10	14	12	12

各进程所需最大资源量

进程	R1	R2	R3	R4
T1	0	1	2	1
T2	1	1	0	2
T3	0	3	4	0
T4	2	0	0	1
T5	0	0	1	3
合计	3	5	7	7

假设分配状态

sum(1)=(2,4,3,1)
sum(2)=(2,2,0,5)
sum(3)=(1,5,5,0)
sum(4)=(5,0,1,3)
sum(5)=(0,3,3,3)

−

allocation(1)=(0,1,2,1)
allocation(2)=(1,1,0,2)
allocation(3)=(0,3,4,0)
allocation(4)=(2,0,0,1)
allocation(5)=(0,0,1,3)

=

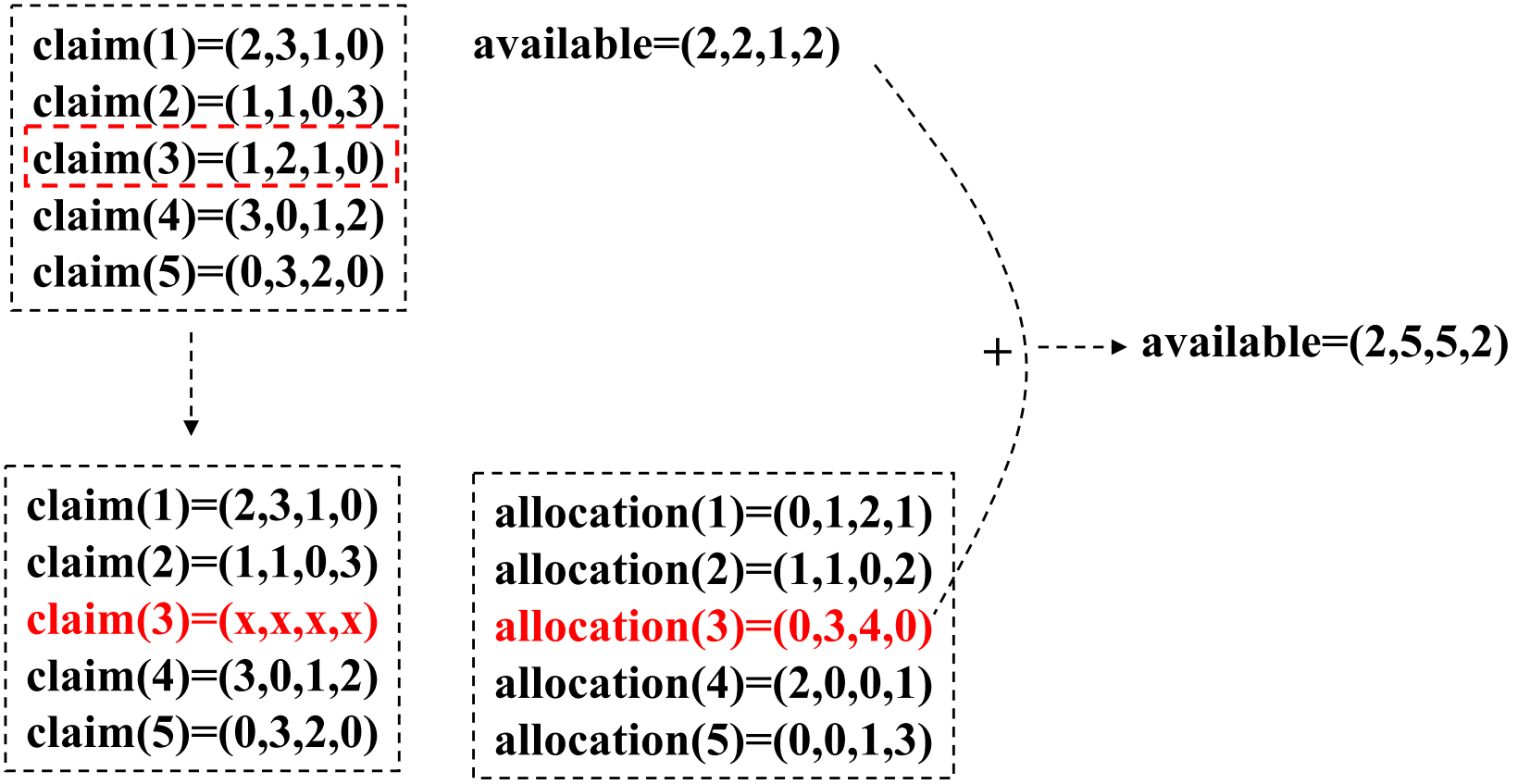
claim(1)=(2,3,1,0)
claim(2)=(1,1,0,3)
claim(3)=(1,2,1,0)
claim(4)=(3,0,1,2)
claim(5)=(0,3,2,0)

五、死锁的避免

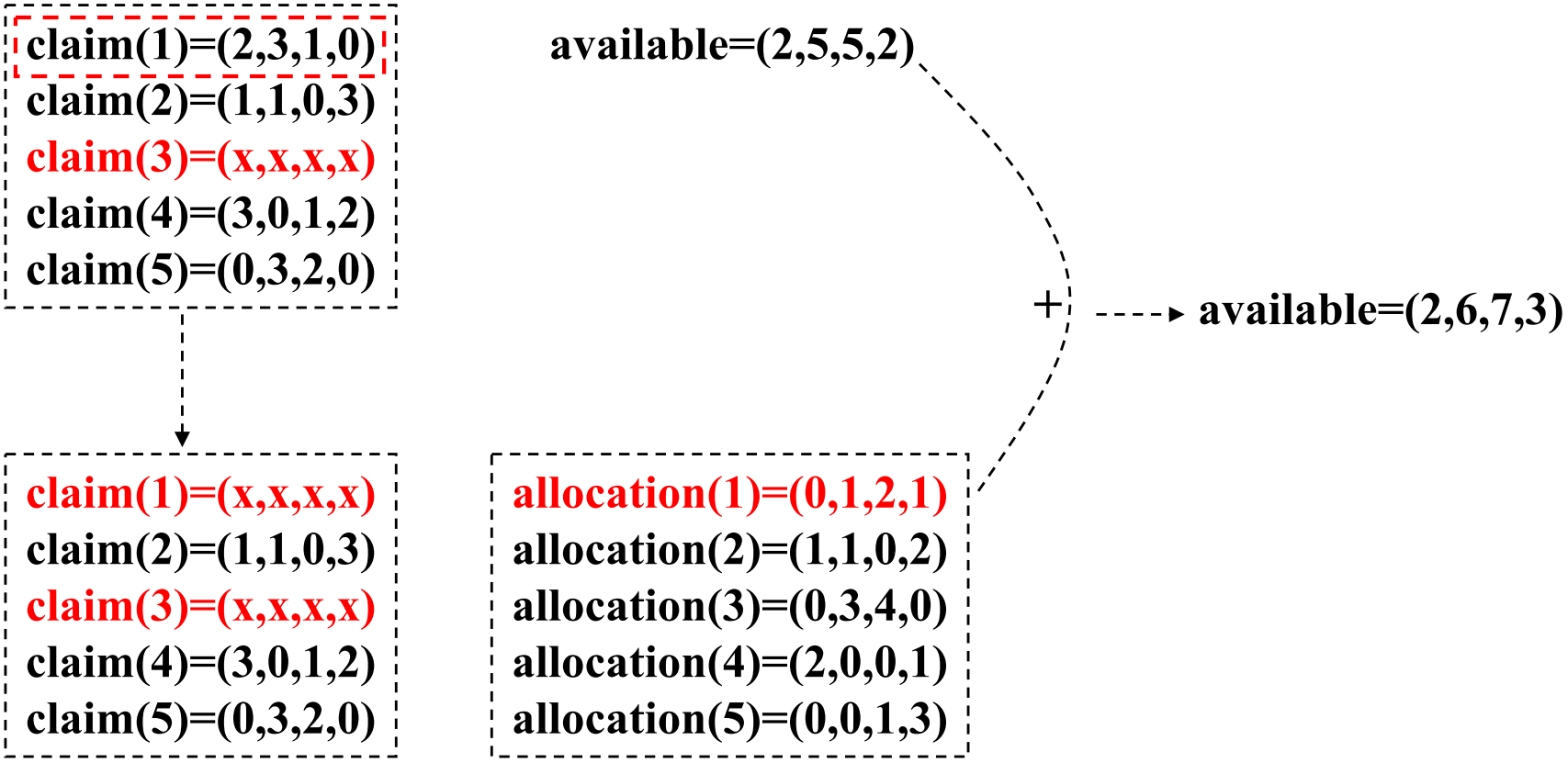
步骤：

- 比较claim(i)和available向量，寻找满足下列关系的进程:

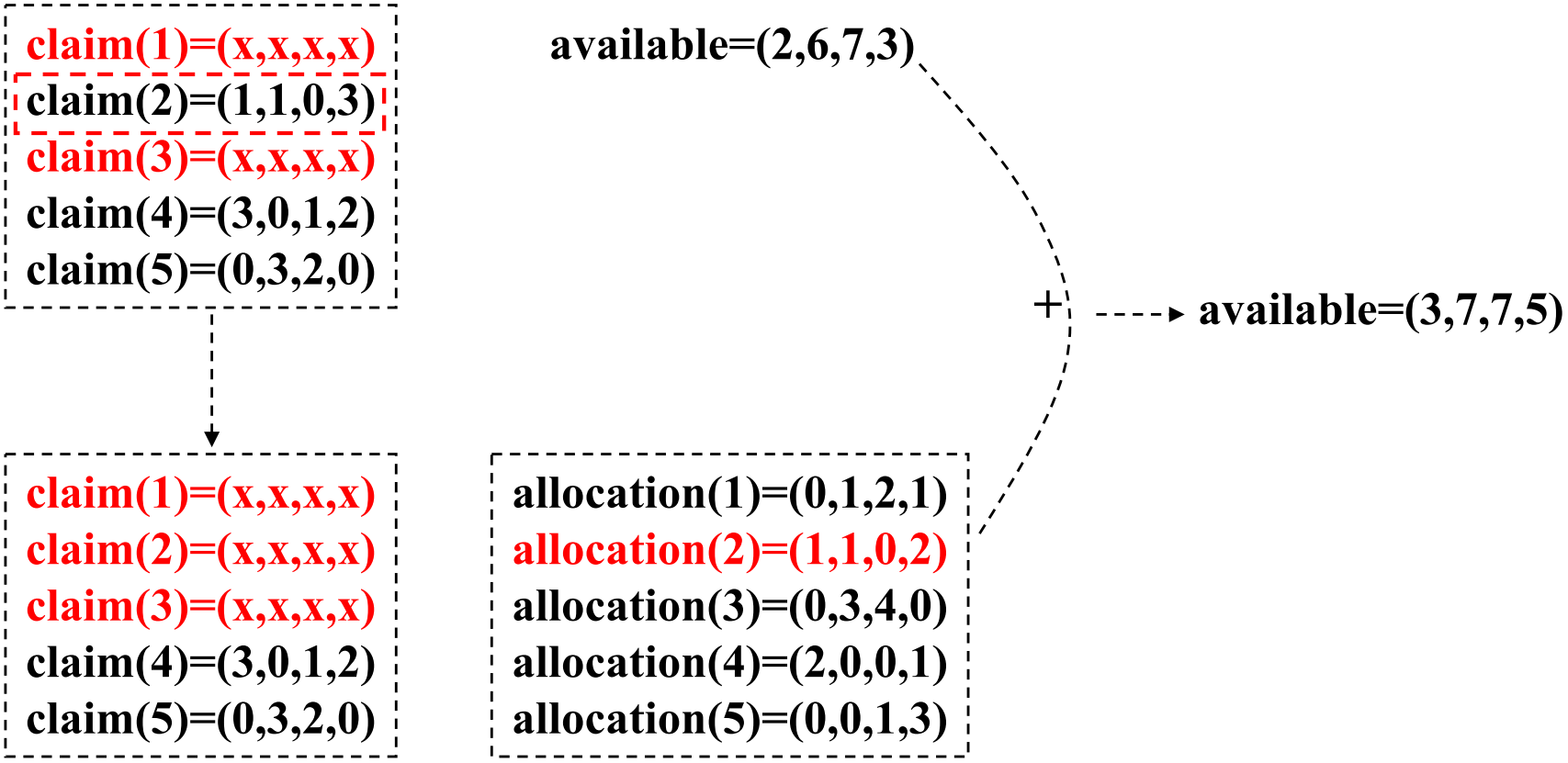
  claim(i) <= available



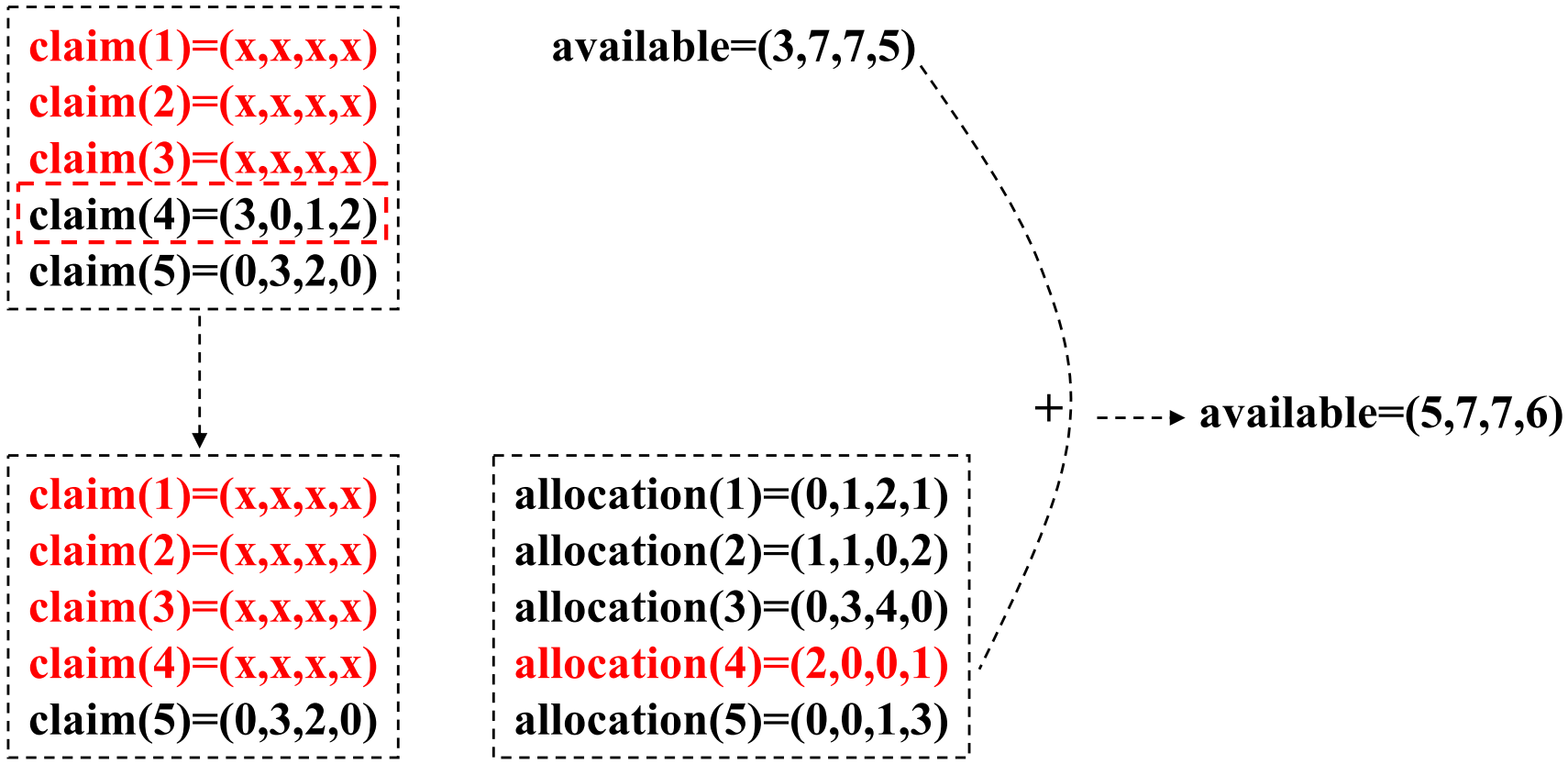
五、死锁的避免


claim(1)=(2,3,1,0)
claim(2)=(1,1,0,3)
claim(3)=(x,x,x,x)
claim(4)=(3,0,1,2)
claim(5)=(0,3,2,0)
available=(2,5,5,2)
+
available=(2,6,7,3)
claim(1)=(x,x,x,x)
claim(2)=(1,1,0,3)
claim(3)=(x,x,x,x)
claim(4)=(3,0,1,2)
claim(5)=(0,3,2,0)
allocation(1)=(0,1,2,1)
allocation(2)=(1,1,0,2)
allocation(3)=(0,3,4,0)
allocation(4)=(2,0,0,1)
allocation(5)=(0,0,1,3)

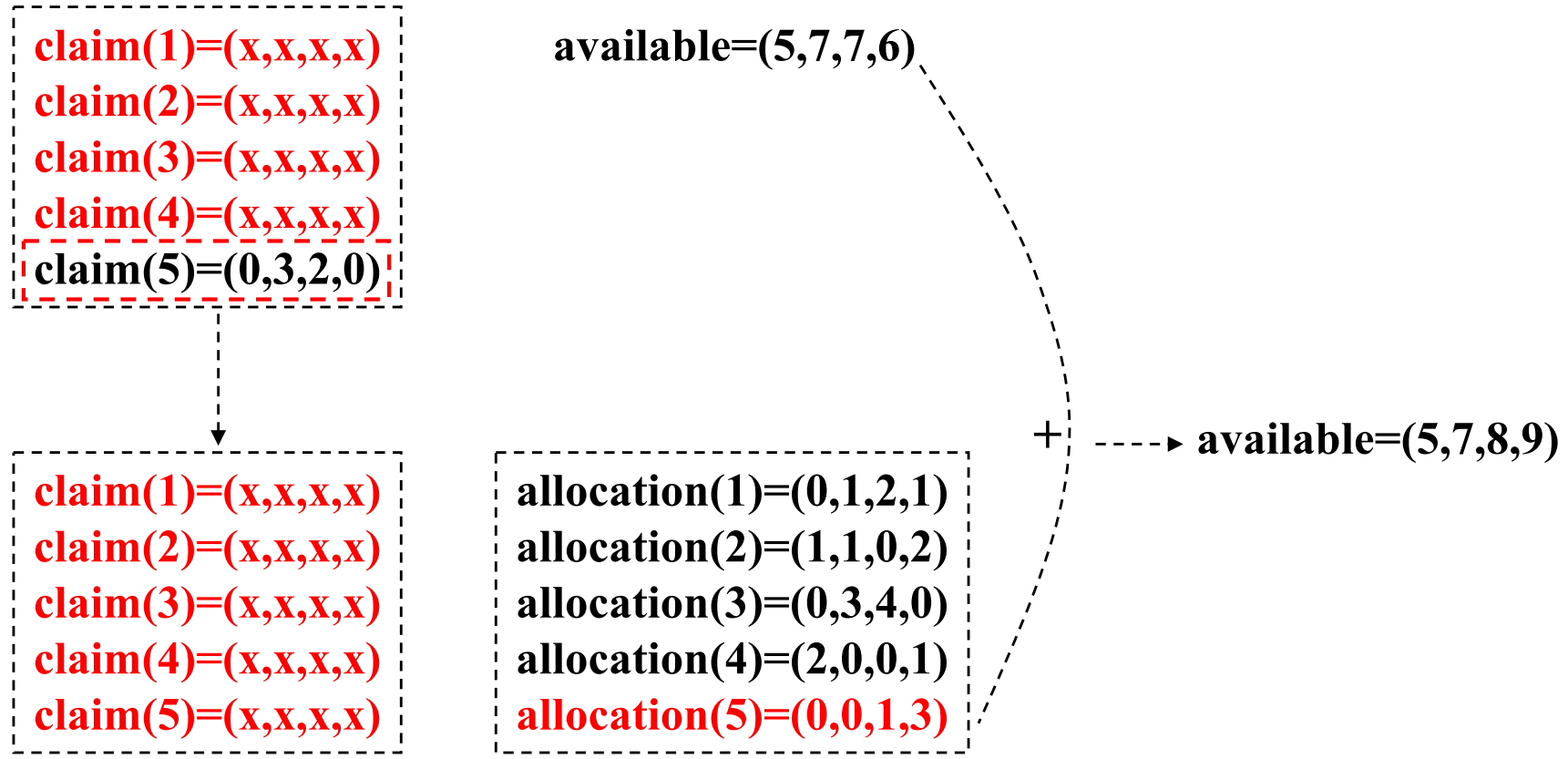
五、死锁的避免


claim(1)=(x,x,x,x)
claim(2)=(1,1,0,3)
claim(3)=(x,x,x,x)
claim(4)=(3,0,1,2)
claim(5)=(0,3,2,0)
available=(2,6,7,3)
+
available=(3,7,7,5)
claim(1)=(x,x,x,x)
claim(2)=(x,x,x,x)
claim(3)=(x,x,x,x)
claim(4)=(3,0,1,2)
claim(5)=(0,3,2,0)
allocation(1)=(0,1,2,1)
allocation(2)=(1,1,0,2)
allocation(3)=(0,3,4,0)
allocation(4)=(2,0,0,1)
allocation(5)=(0,0,1,3)

五、死锁的避免


claim(1)=(x,x,x,x)
claim(2)=(x,x,x,x)
claim(3)=(x,x,x,x)
claim(4)=(3,0,1,2)
claim(5)=(0,3,2,0)
available=(3,7,7,5)
+
available=(5,7,7,6)
claim(1)=(x,x,x,x)
claim(2)=(x,x,x,x)
claim(3)=(x,x,x,x)
claim(4)=(x,x,x,x)
claim(5)=(0,3,2,0)
allocation(1)=(0,1,2,1)
allocation(2)=(1,1,0,2)
allocation(3)=(0,3,4,0)
allocation(4)=(2,0,0,1)
allocation(5)=(0,0,1,3)

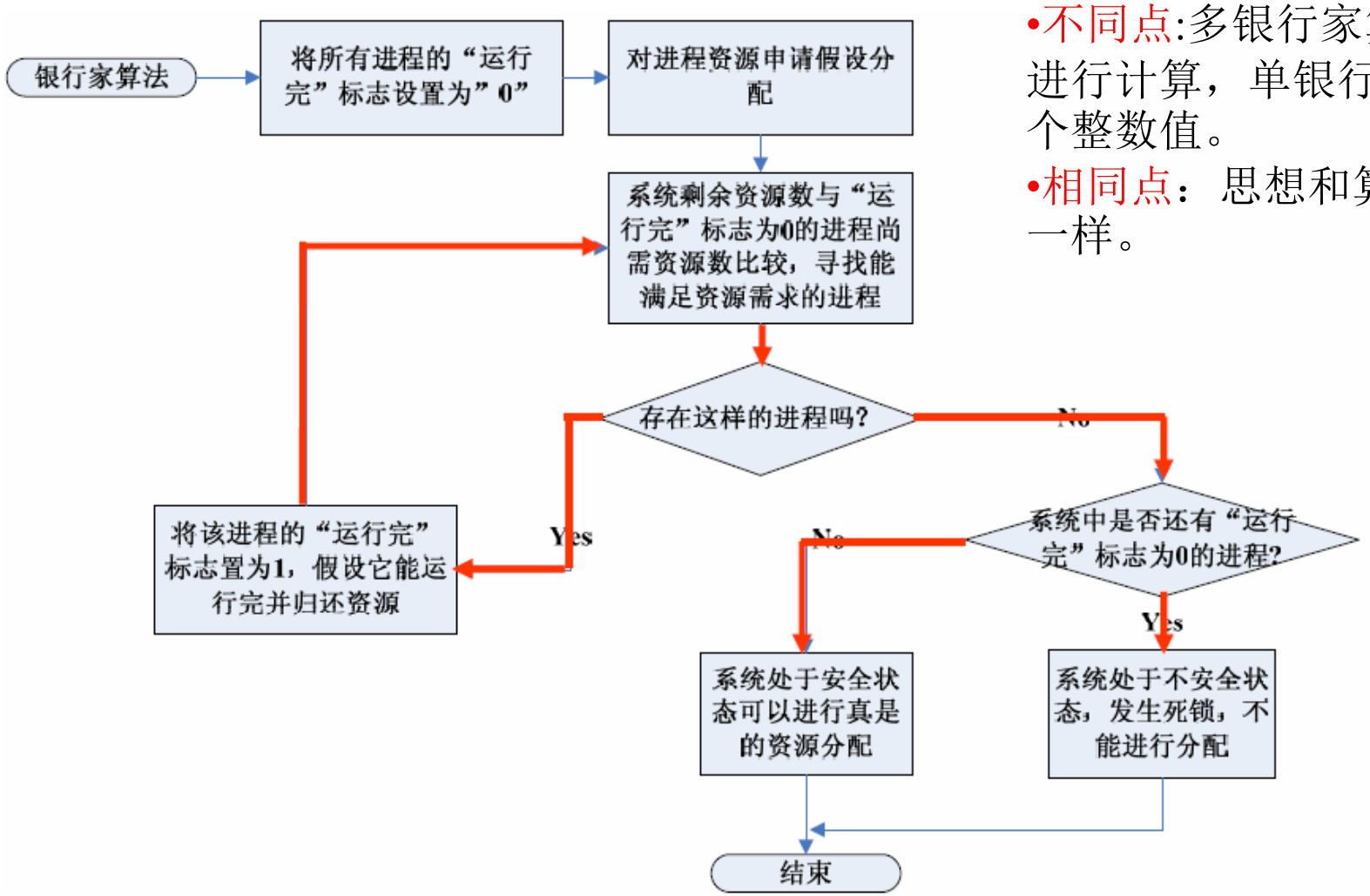
五、死锁的避免



结论：可满足T2的资源请求

五、死锁的避免

算法流程



单银行家算法和多银行家算法的异同：

•不同点:多银行家算法使用向量进行计算，单银行家算法使用单个整数值。

•相同点：思想和算法流程完全一样。

五、死锁的避免

死锁避免策略的局限

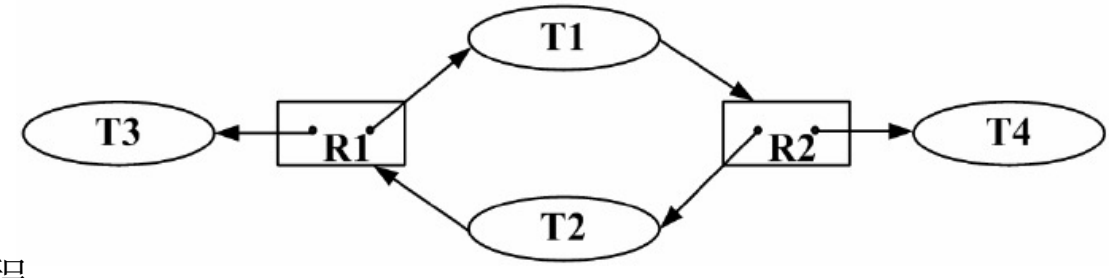
- 预先必须声明进程需要的资源总量——现代系统难以做到。
- 使进程的行为受到除同步和互斥之外，第三种因素即系统安全因素的影响——一定程度上加大了对进程行为的约束。

六、死锁的检测和解除

六、死锁的检测和解除

死锁的检测

- 检测工具——资源分配图
- 资源分配图：
  - 是描述进程申请资源和资源分配情况的关系模型图。表示系统中某个时刻进程对资源的申请和占有情况。



规则：

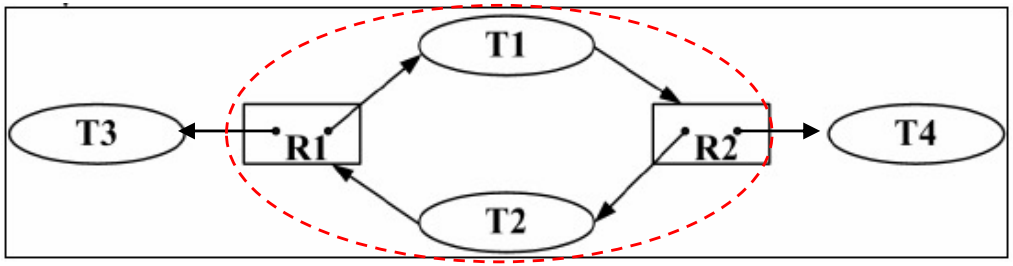
（1）**圆(椭圆)**表示一个进程；

（2）**方块**表示一个资源类，其中的圆点表示该类型资源中的单个资源；

（3）从**资源指向进程的箭头**表示资源被分配给了这个进程；

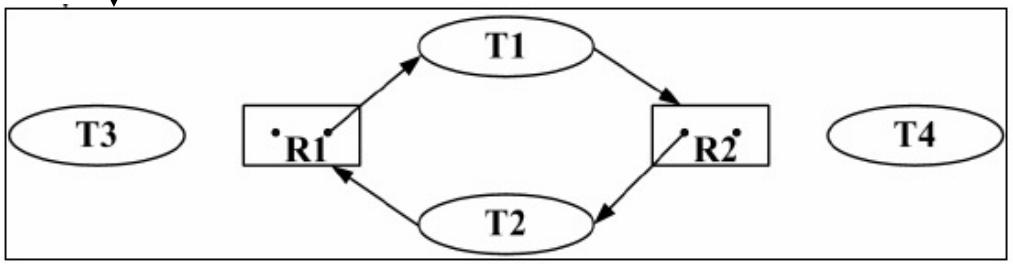
（4）从**进程指向资源的箭头**表示进程申请一个这类资源；

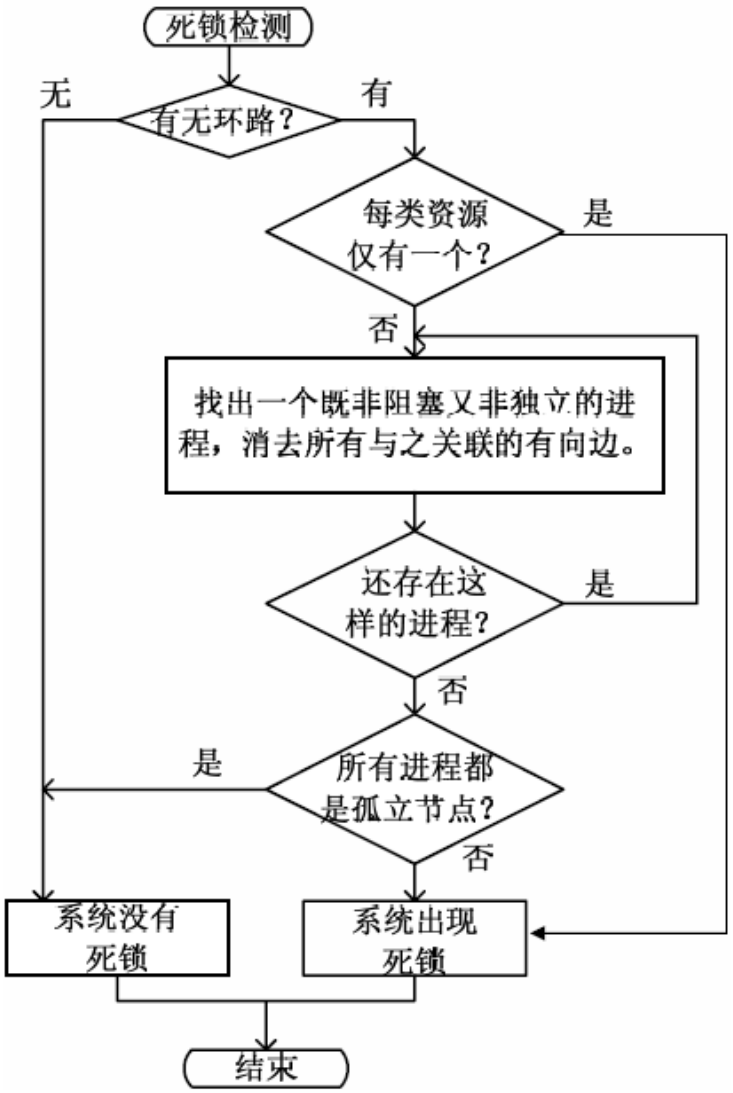
六、死锁的检测和解除



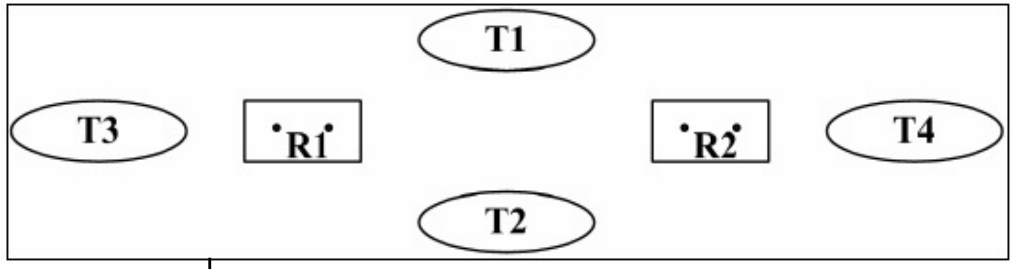
P_i:表示第i个进程，R_j表示第j个资源；
$|(P_i, R_j)|$表示P_i申请R_j资源的个数；
$|(R_j, P_i)|$表示R_j已分配给P_i资源的个数。
W_j表示R_j类资源的个数

$$|(P_i, R_j)| + \sum_k |(R_j, P_k)| \le W_j$$





六、死锁的检测和解除



资源分配图中的所有进程如果都能化简成孤立结点，则这个资源图就是可完全化简的（completely reducible）；反之，就是不可完全化简的（irreducible）。

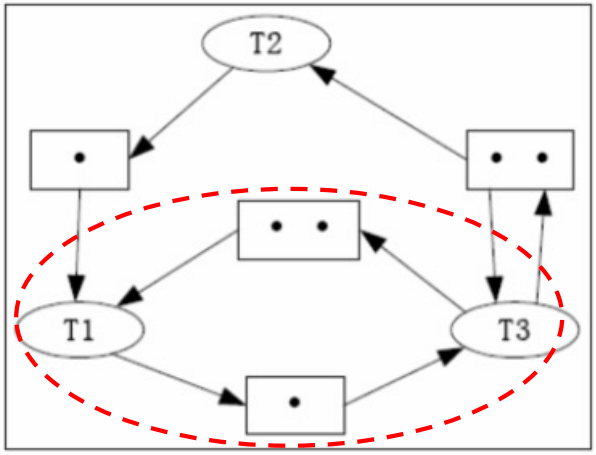
死锁定理：如果一个系统状态为死锁状态，当且仅当资源分配图是不可完全化简。也即，如果资源图中所有的进程都成为孤立结点，则系统不会死锁；否则系统状态为死锁状态。

结论：系统不会发生死锁

六、死锁的检测和解除

举例

- 资源分配图如下，请分别化简并说明是否会发生死锁。



步骤：

Step1：检测有无环路

Step2：检测环路中每个资源类中是否只有一个资源

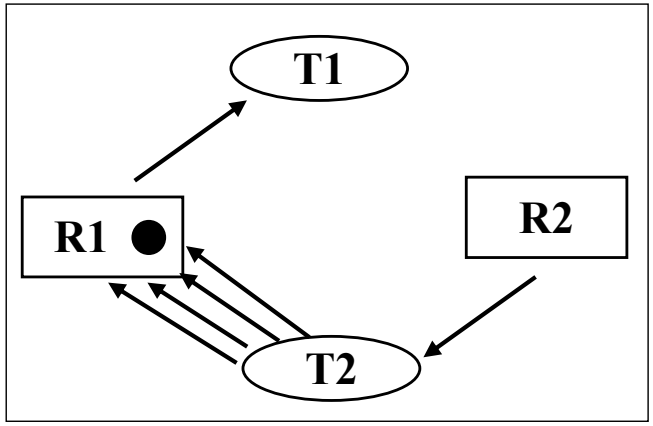
Step3：在环路中查找非阻塞也非独立的进程

结论：此资源分配图无法化简，必定死锁。

六、死锁的检测和解除

临时资源的死锁检测

- 临时性资源：即可消耗的资源。如信号、消息、邮件等。
- 特点：没有固定数目；不需要释放。
- 表述方式——重定义的资源分配图



  - 规则：
    - ① 圆表示一个进程；
    - ② 方块表示一个资源类，其中的圆点表示该类型资源中的单个资源；
    - ③ 由进程指向资源的箭头表示该进程申请这种资源，一个箭头只表示申请一个资源；
    - ④ 由资源类指向进程的箭头表示该进程产生这种资源，一个箭头可表示产生一到多个资源，每个资源类至少有一个生产者进程。

六、死锁的检测和解除

❖ 判断方法

- 分析：对于临时性资源来讲，它有生产者，生产者会源源不断的生产资源，因此只要生产者进程不被阻塞，可以认为资源最终一定是充分的，可以满足各消费进程的需要。

- 结论：

  判断系统是否死锁的关键在于判断生产者进程的状态，若生产者进程不被阻塞，则可以认为它总会生产出该类资源，也就是说，申请这类资源的所有申请者进程都可以得到满足。

六、死锁的检测和解除

❖ 检测方法——化简

- 方法：

① 从那些没有阻塞的进程入手，删除那些没有阻塞的进程的请求边，并使资源类中资源数（图中黑点的数目）减1。

② 重复以上步骤直至：

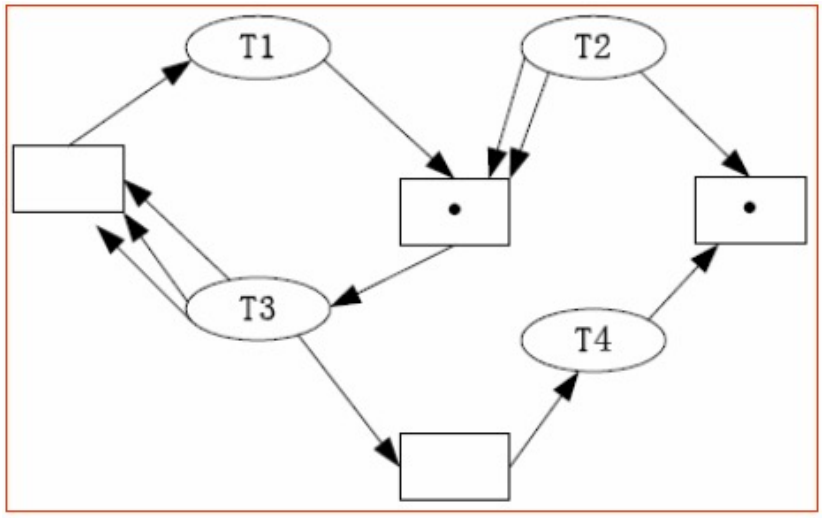
a. 图中所有的请求边都已经删除，则不会死锁

b. 图中仍然存在请求边但无法再化简，系统死锁

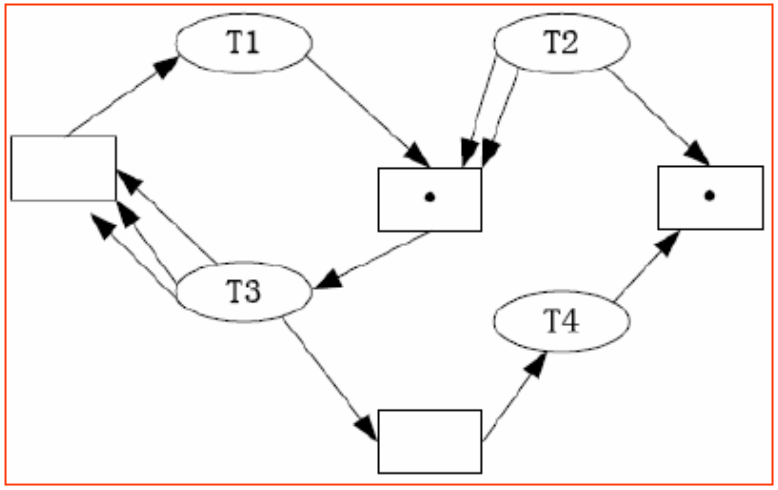
六、死锁的检测和解除

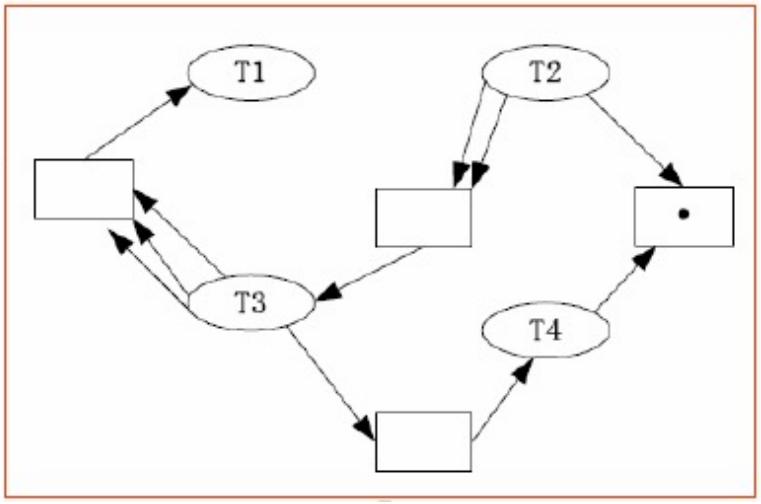
死锁检测举例

- 资源分配图如下（临时性资源），请分别化简并说明是否会发生死锁：

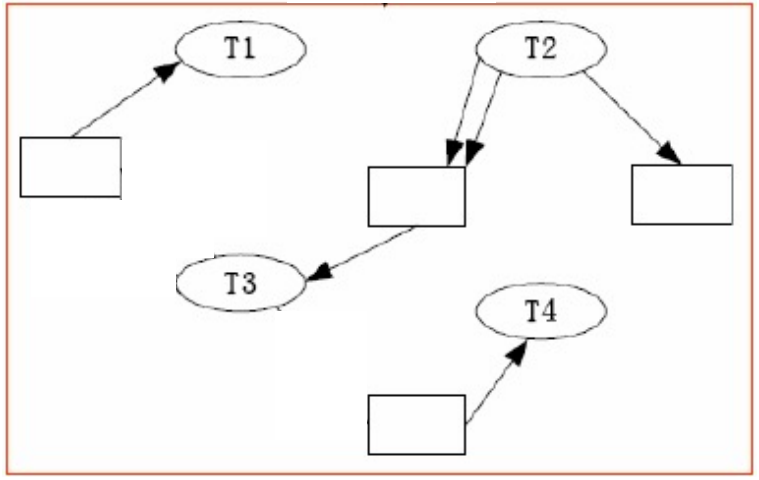


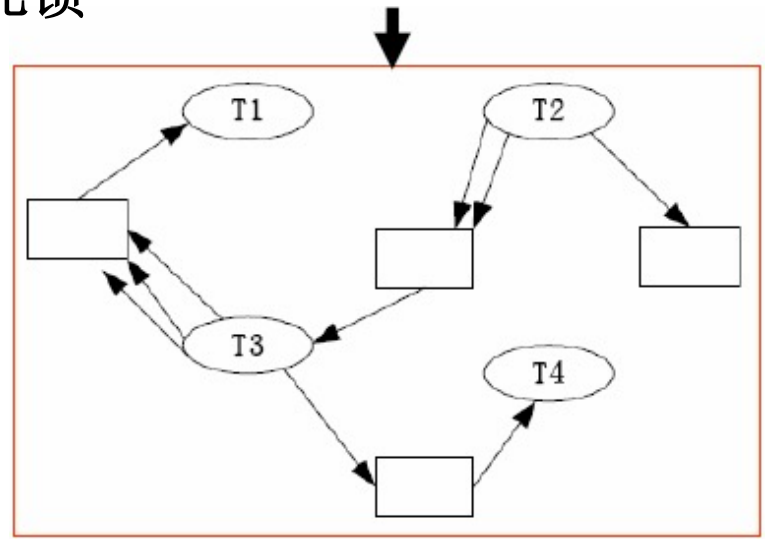
六、死锁的检测和解除





结论：发生死锁





六、死锁的检测和解除

死锁的解除

❖ 重新启动

- 这是一种常用但比较粗暴的方法，虽然实现简单，但会使之前的工作全部白费，造成很大的损失和浪费。

❖ 撤消进程

- 死锁发生时，系统撤消造成死锁的进程，解除死锁。
- 一次性撤消所有的死锁进程。损失较大。
- 逐个撤消，分别收回资源。具体做法：系统可以先撤消那些优先级低的、已占有资源少或已运行时间短的、还需运行时间较长的进程，尽量减少系统的损失。

六、死锁的检测和解除

❖ 剥夺资源

- 死锁时，系统保留死锁进程，只剥夺死锁进程占有的资源，直到解除死锁。选择被剥夺资源进程的方法和选择被撤消进程相同。

❖ 进程回退

- 死锁时，系统可以根据保留的历史信息，让死锁的进程从当前状态向后退回到某种状态，直到死锁解除。
- 实现方法：可以通过结合检查点或回退（Checkpoint/Rollback）机制实现。进程某一时刻的瞬间状态叫做检查点，可以定期设置检查点。一旦系统检查到有某个进程卷入了死锁，系统查看保存的检查点信息，重新建立该进程的状态，从上次检查点的位置重新执行。