



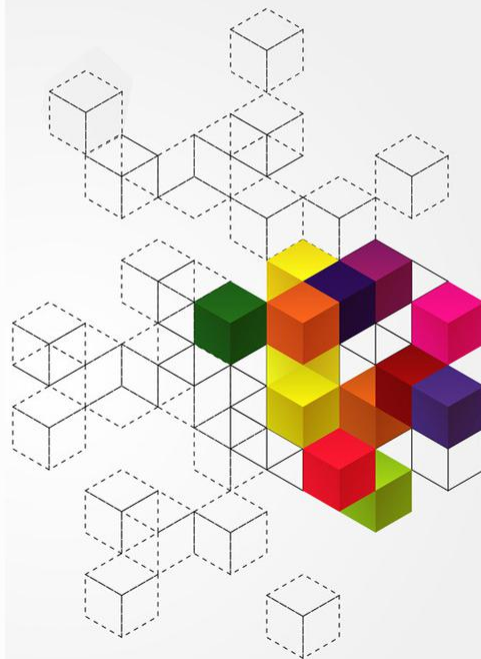
操作系统

Operating system

孔维强

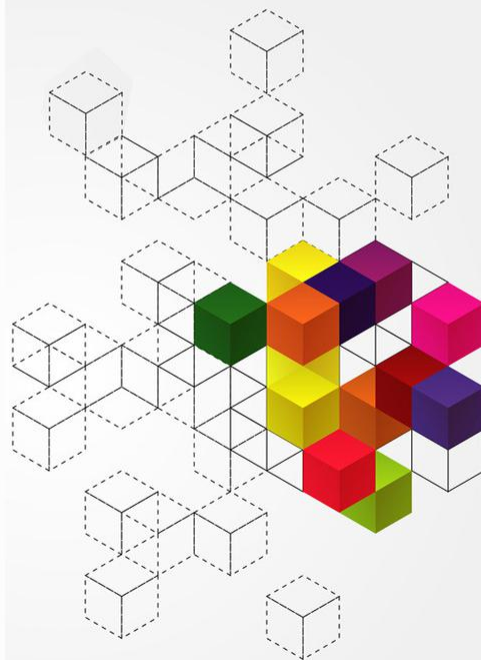
大连理工大学

- 一、死锁避免概念
- 二、单资源实例条件下的死锁避免
- 三、多实例的死锁避免算法
- 四、银行家算法实例解析



一、死锁避免概念

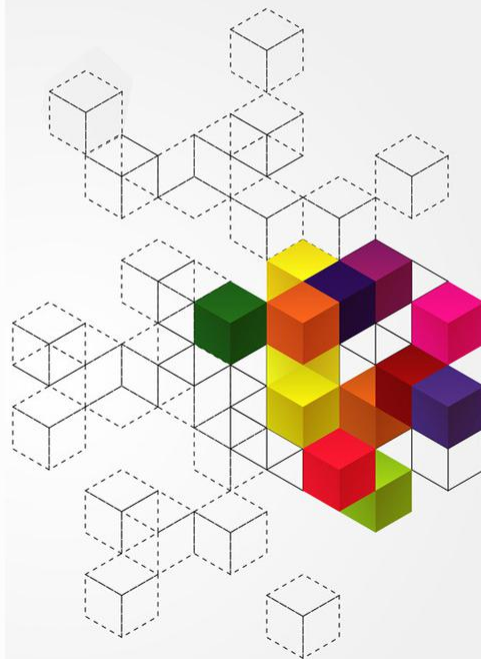
- 死锁避免 (Deadlock Avoidance)
 - 在程序运行起来后，通过对每次资源分配请求进行系统审核，通过拒绝为不安全的资源请求进行分配，来达到避免死锁风险的目的，这类方法被称为死锁避免方法
 - 方法的核心要素：安全性判定
 - 关键概念：安全序列



二、单资源实例条件下的死锁避免

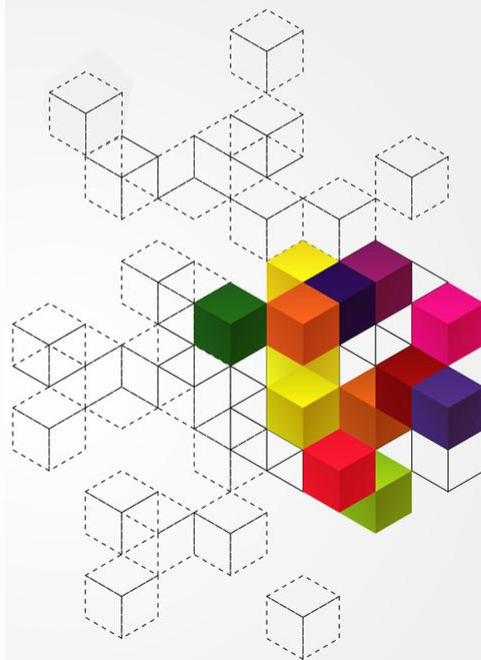
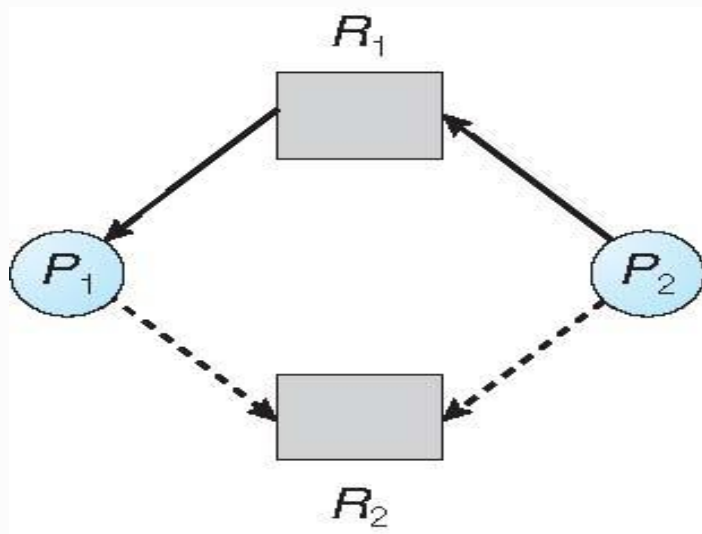
● 使用资源分配图

- 在图添加另一种类型的边，表示进程未来可能要提起的申请边，称为Claim edge
- 如果进程已经开始申请资源，那么相应的Claim edge会转变为Request edge



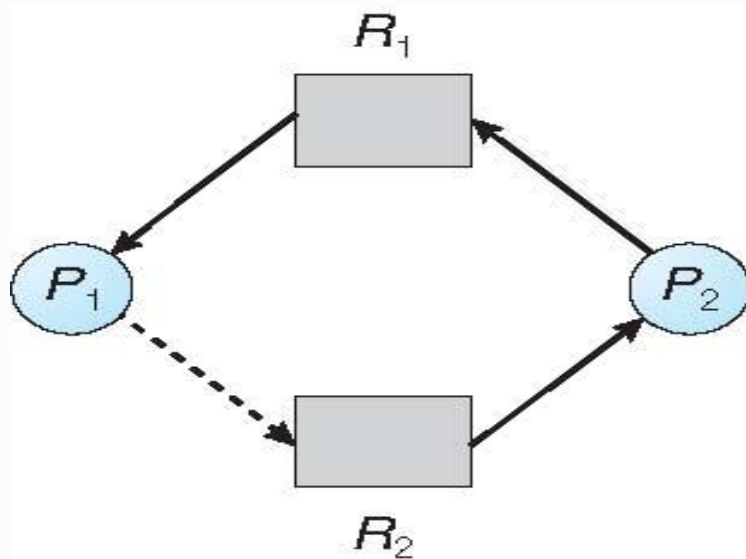
二、单资源实例条件下的死锁避免

资源分配图示例

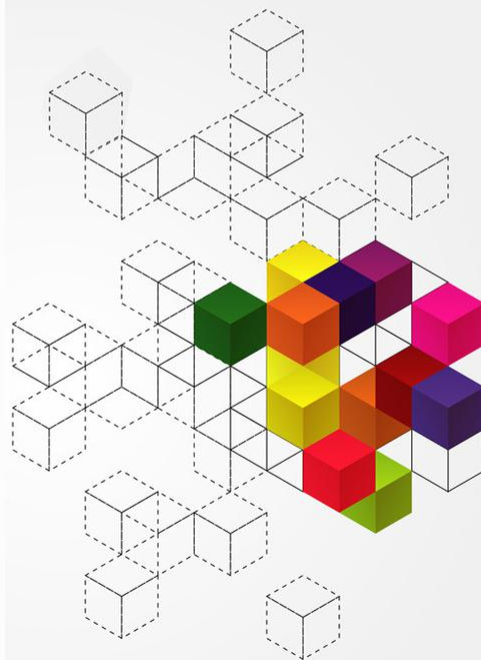


二、单资源实例条件下的死锁避免

资源分配图示例



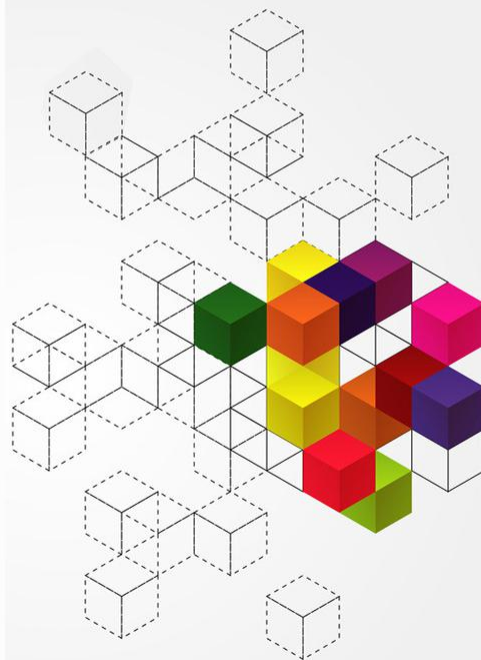
此时，如果允许 P_1 发起对 R_2 的申请
则死锁发生



三、多实例的死锁避免算法

● 银行家算法

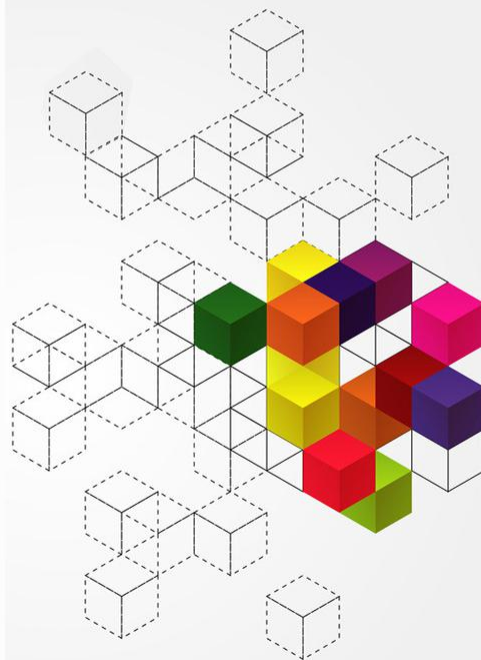
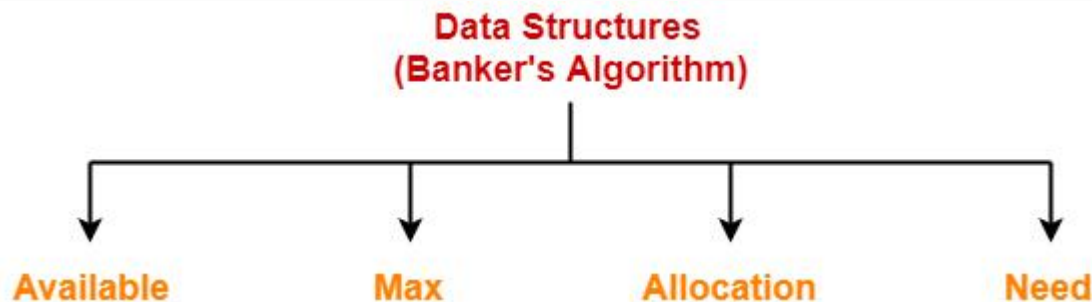
- 多资源实例条件下进行死锁避免，要比单资源实例的情形略复杂
- 主要在检测每次分配是否会使得系统进入不安全状态的方法有所不同



三、多实例的死锁避免算法

● 银行家算法的基本数据结构

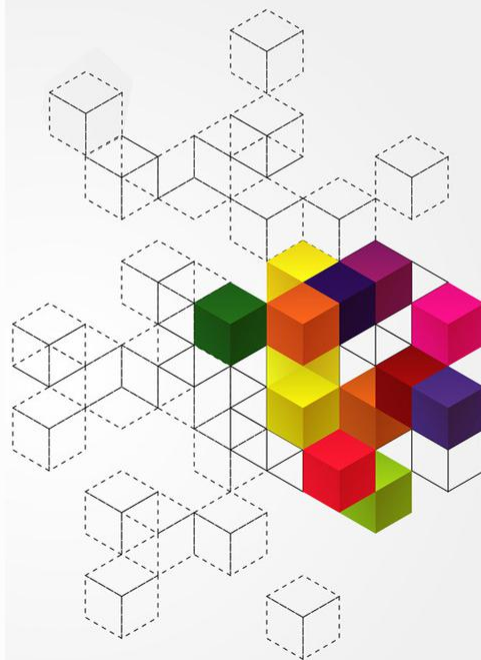
- Available: array[1..m] of integer; //系统可用资源
- Max: array[1..n, 1..m] of integer; //进程最大需求
- Allocation: array[1..n, 1..m] of integer; //当前分配
- Need: array[1..n, 1..m] of integer; //尚需资源
- Request: array[1..n, 1..m] of integer; //当前请求



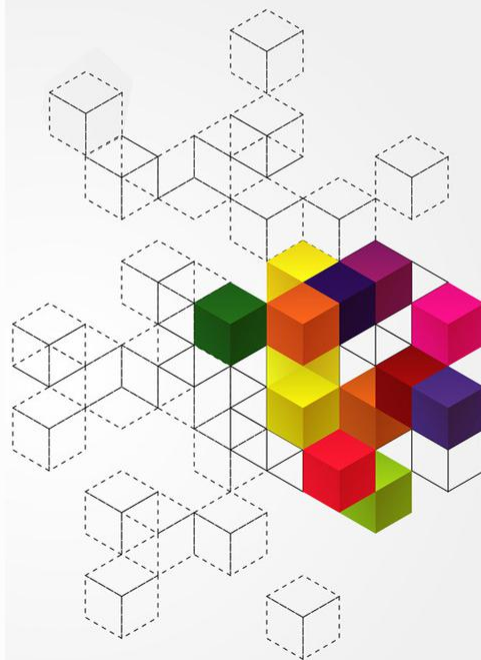
三、多实例的死锁避免算法

● 银行家算法的初始化

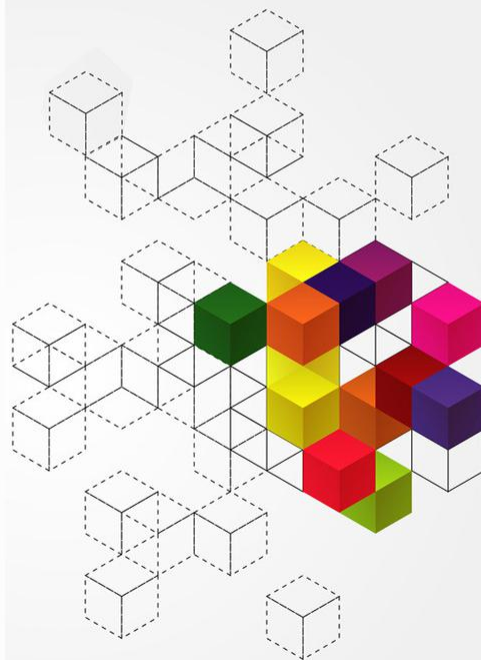
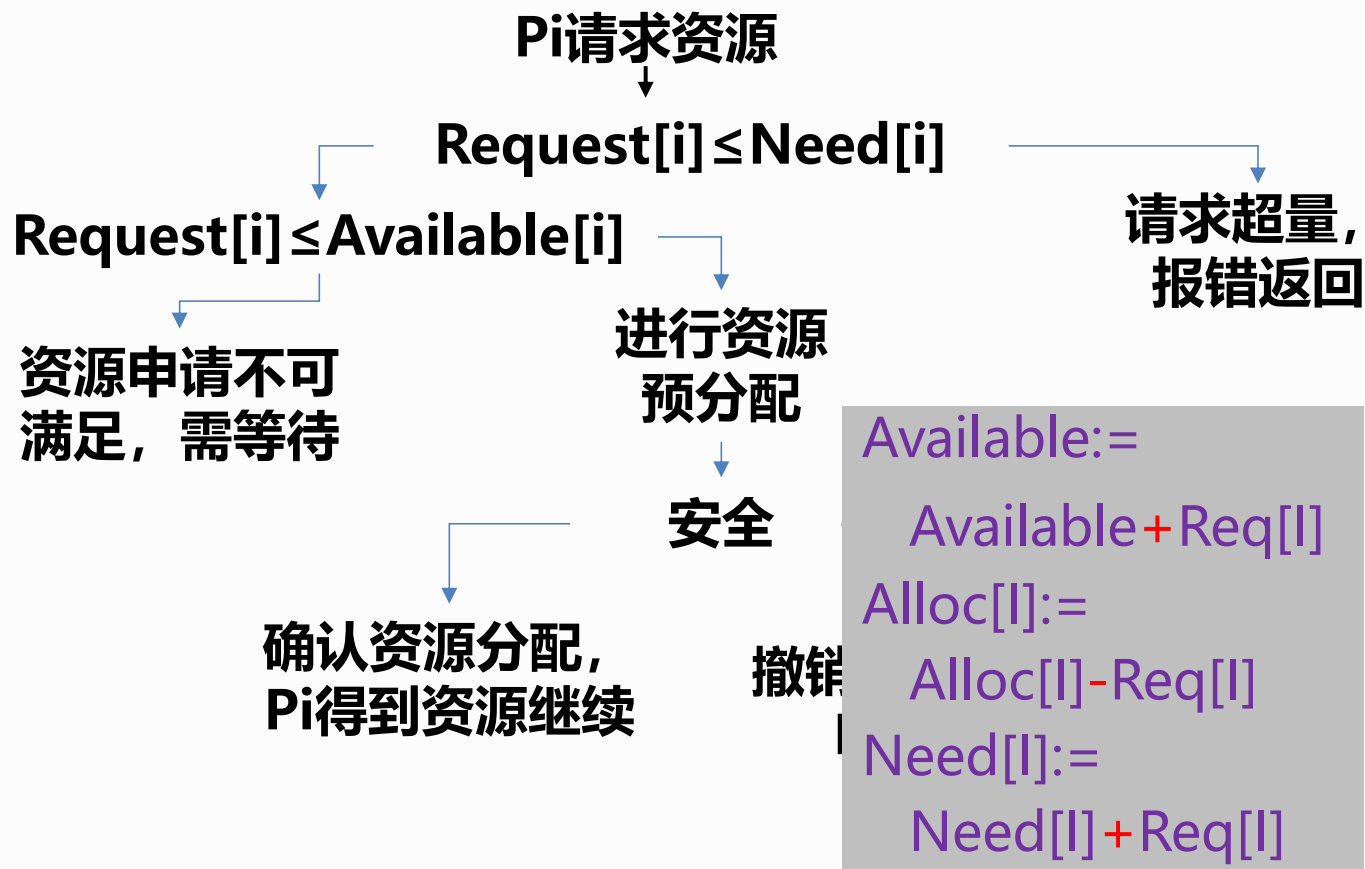
- 初始化
- For each process P_i
 - $Need[i] = Max[i]$
 - $Allocation[i] = 0$
 - $Finish[i] = false$



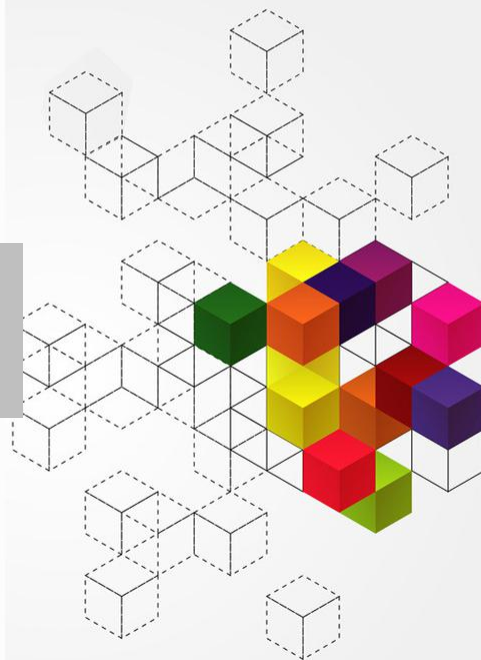
三、多实例的死锁避免算法



三、多实例的死锁避免算法

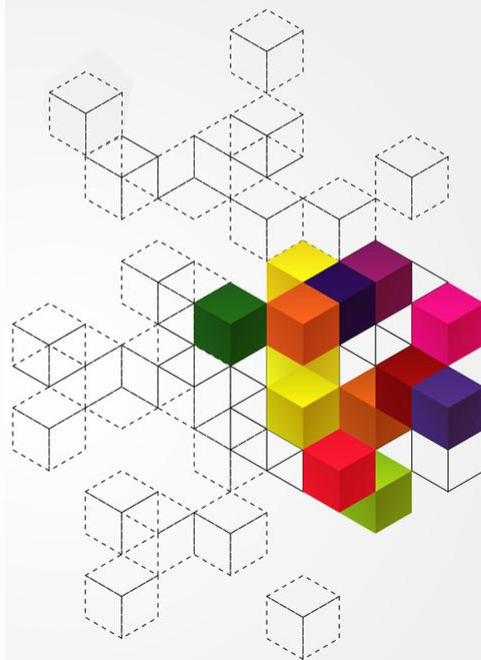
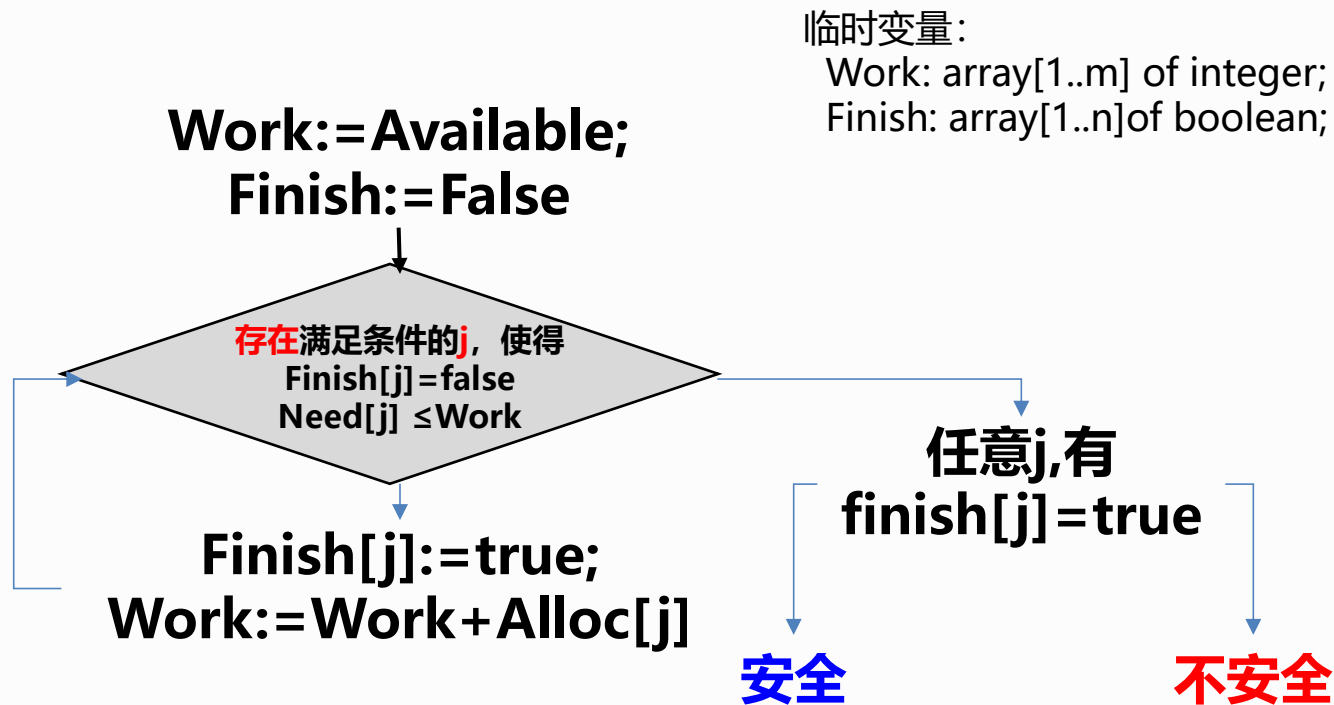


三、多实例的死锁避免算法



三、多实例的死锁避免算法

银行家算法的安全性判定流程



四、银行家算法实例解析

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p0,p1,p2,p3,p4\}$: 5个进程

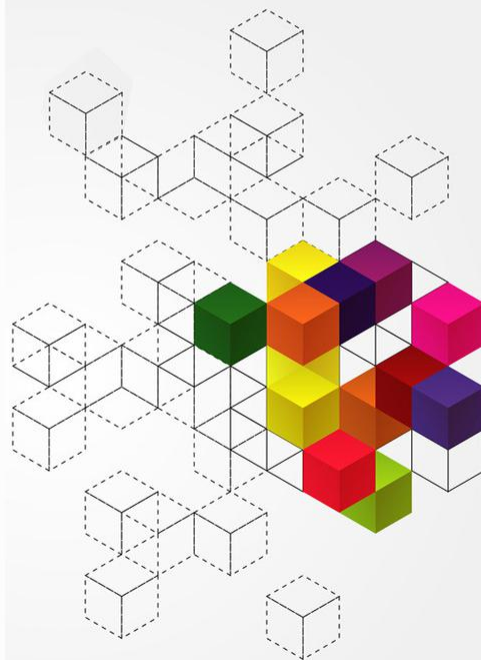
	<u>Max</u>			<u>Alloc</u>			<u>Need</u>			<u>Available</u>			<u>Work</u>			<u>Finish</u>		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C			
P0:	7	5	3	0	1	0	7	4	3	3	3	2						
p1:	3	2	2	2	0	0	1	2	2									
p2:	9	0	2	3	0	2	6	0	0									
p3:	2	2	2	2	1	1	0	1	1									
p4:	4	3	3	0	0	2	4	3	1									

问题1：当前状态是否安全？

结论：初始状态安全

根据银行家算法的安全性判定流程，找到安全序列

<p1,p3,p4,p2,p0>



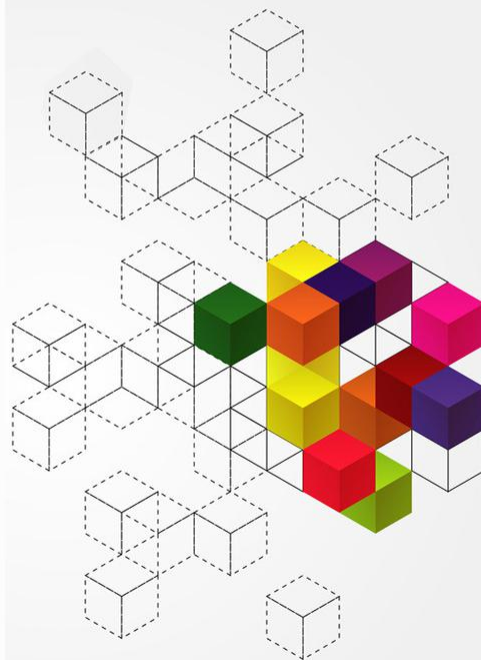
四、银行家算法实例解析

$R=\{A(10),B(5),C(7)\}$: 3类资源

$P=\{p_0,p_1,p_2,p_3,p_4\}$: 5个进程

	<u>Max</u>			<u>Alloc</u>			<u>Need</u>			<u>Available</u>			<u>Work</u>			<u>Finish</u>
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	
P0:	7	5	3	0	1	0	7	4	3	3	3	2				
p1:	3	2	2	2	0	0	1	2	2							
p2:	9	0	2	3	0	2	6	0	0							
p3:	2	2	2	2	1	1	0	1	1							
p4:	4	3	3	0	0	2	4	3	1							

问题2:
进程P1提出资源申请
 $\text{Request}[1]=(1,0,2)$
是否可以满足之




四、银行家算法实例解析

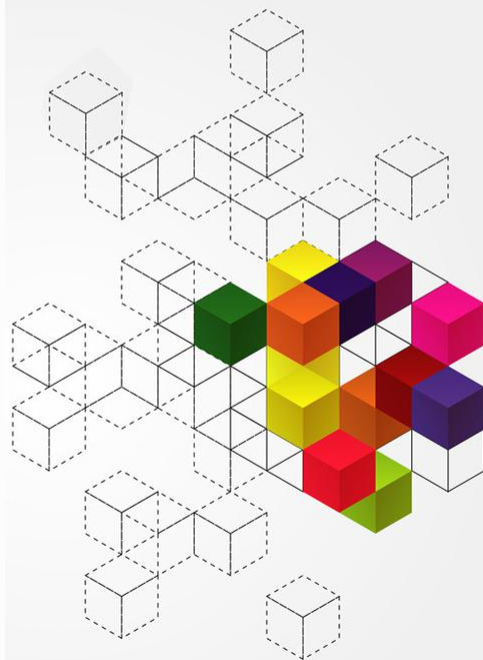
问题2: 进程P1提出资源申请 $\text{Request}[1]=(1,0,2)$
是否可以满足之

步骤2.1: 对 $\text{req}[1]$ 进行预分配, 资源状态发生变化

	<u>Max</u>			<u>Alloc</u>			<u>Need</u>			<u>Available</u>			<u>Work</u>			<u>Finish</u>		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C			
P0:	7	5	3	0	1	0	7	4	3	2	3	0						
p1:	3	2	2	3	0	2	0	2	0									
p2:	9	0	2	3	0	2	6	0	0									
p3:	2	2	2	2	1	1	0	1	1									
p4:	4	3	3	0	0	2	4	3	1									

步骤2.2: 判断预分配后, 资源状态是否安全

找到安全进程序列: $\langle p1, p3, p4, p0, p2 \rangle$  **安全**



四、银行家算法实例解析

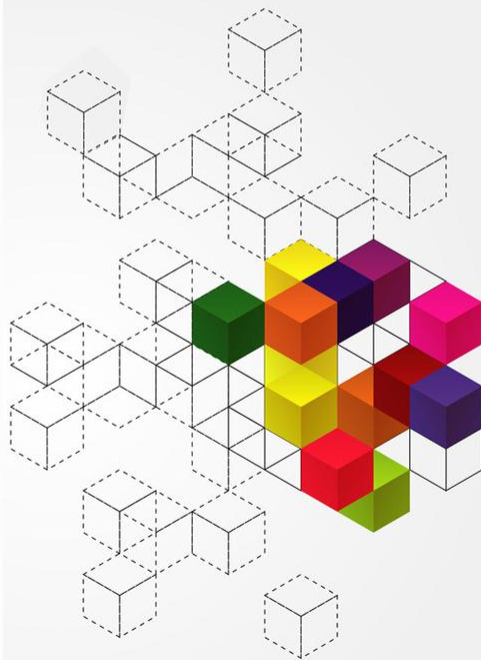
问题2: 进程P1提出资源申请Request[1]=(1,0,2)
是否可以满足之

步骤2.3: 预分配状态安全，允许为P1完成此次分配

	<u>Max</u>			<u>Alloc</u>			<u>Need</u>			<u>Available</u>			<u>Work</u>			<u>Finish</u>		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C			
P0:	7	5	3	0	1	0	7	4	3	2	3	0						
p1:	3	2	2	3	0	2	0	2	0									
p2:	9	0	2	3	0	2	6	0	0									
p3:	2	2	2	2	1	1	0	1	1									
p4:	4	3	3	0	0	2	4	3	1									

问题3:
进程P4此后提出
request[4]=(3,3,0)
能否满足?

结论: 资源不够，P4等待

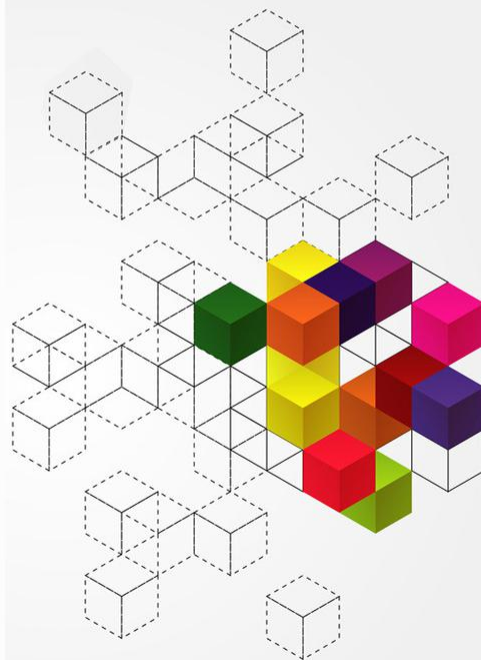


四、银行家算法实例解析

	<u>Max</u>			<u>Alloc</u>			<u>Need</u>			<u>Available</u>			<u>Work</u>			<u>Finish</u>
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	
P0:	7	5	3	0	1	0	7	4	3	2	3	0				
p1:	3	2	2	3	0	2	0	2	0							
p2:	9	0	2	3	0	2	6	0	0							
p3:	2	2	2	2	1	1	0	1	1							
p4:	4	3	3	0	0	2	4	3	1							

问题4:
进程P0此后提出
 $\text{request}[0] = (0, 2, 0)$
能否满足?

结论: 预分配后发现不安全, 不能分配



本讲小结

- 死锁避免概念
- 单资源实例条件下的死锁预防
- 多实例的死锁避免算法
- 银行家算法示例解析

