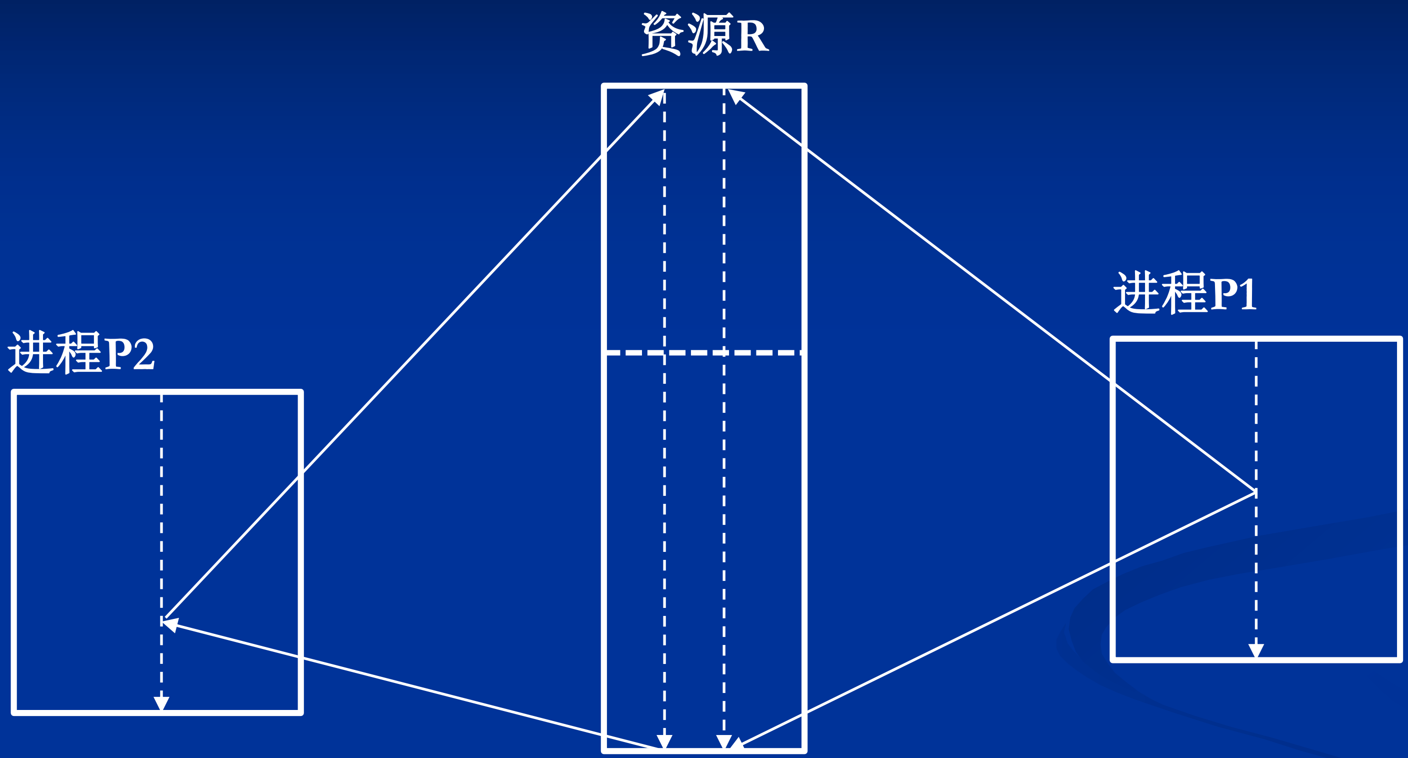
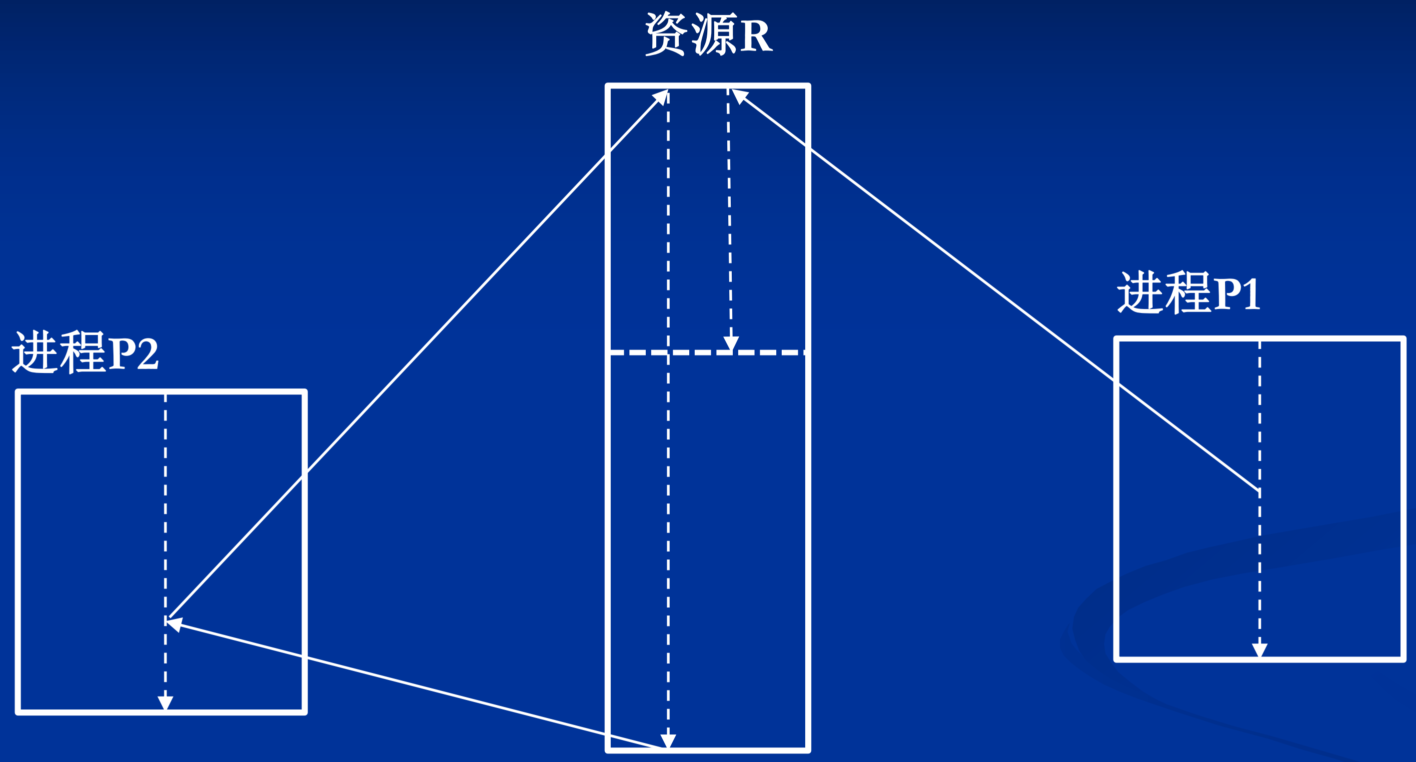
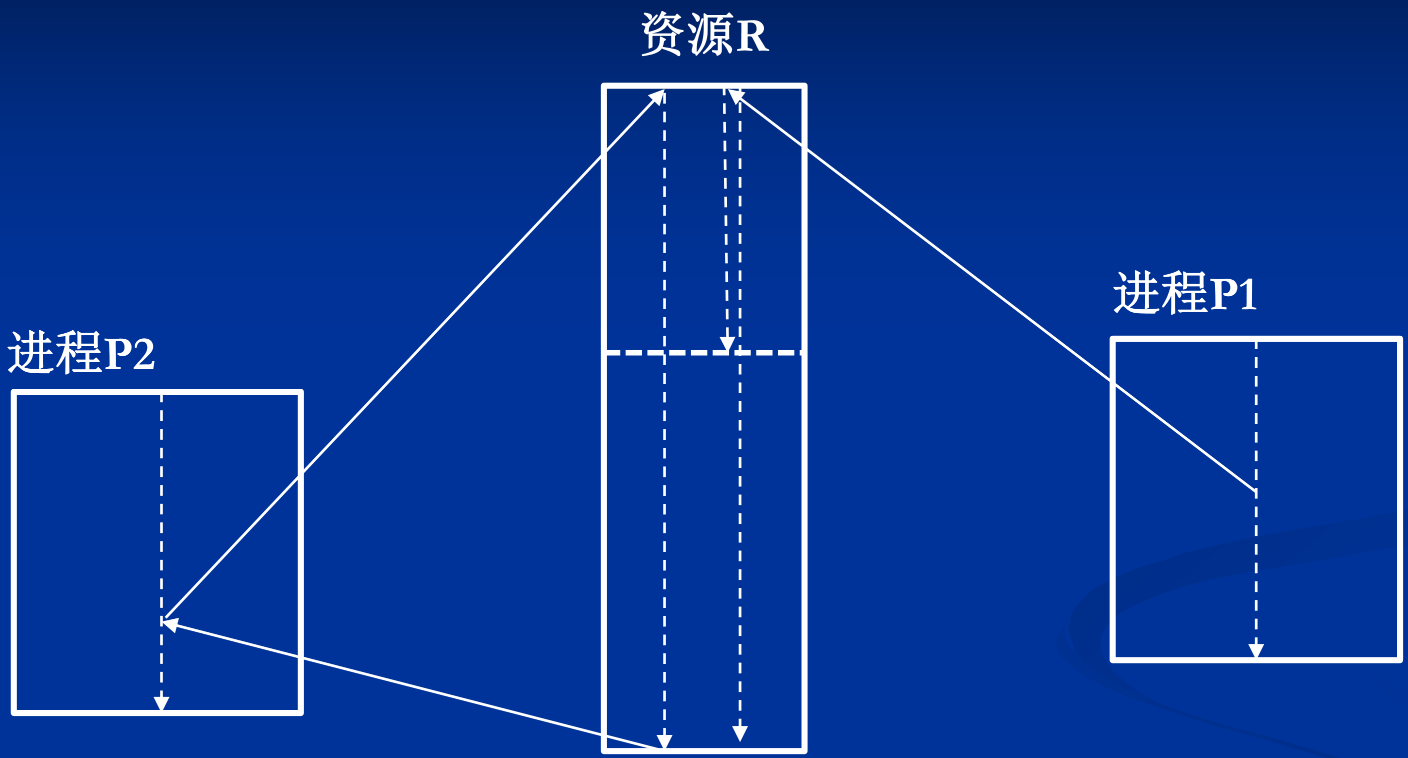
■ 示例：互斥条件与不剥夺条件



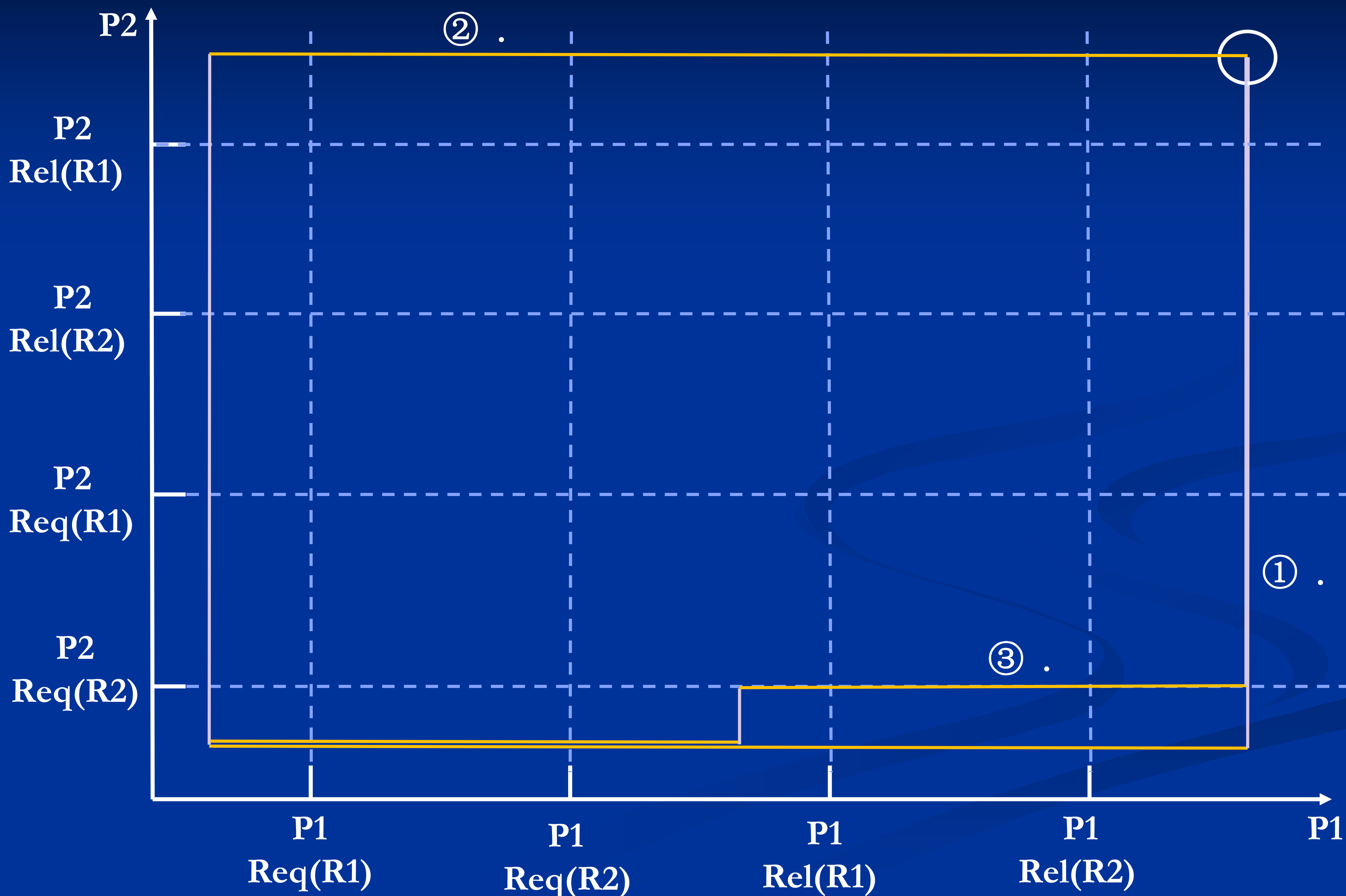
■ 示例：互斥条件与不剥夺条件



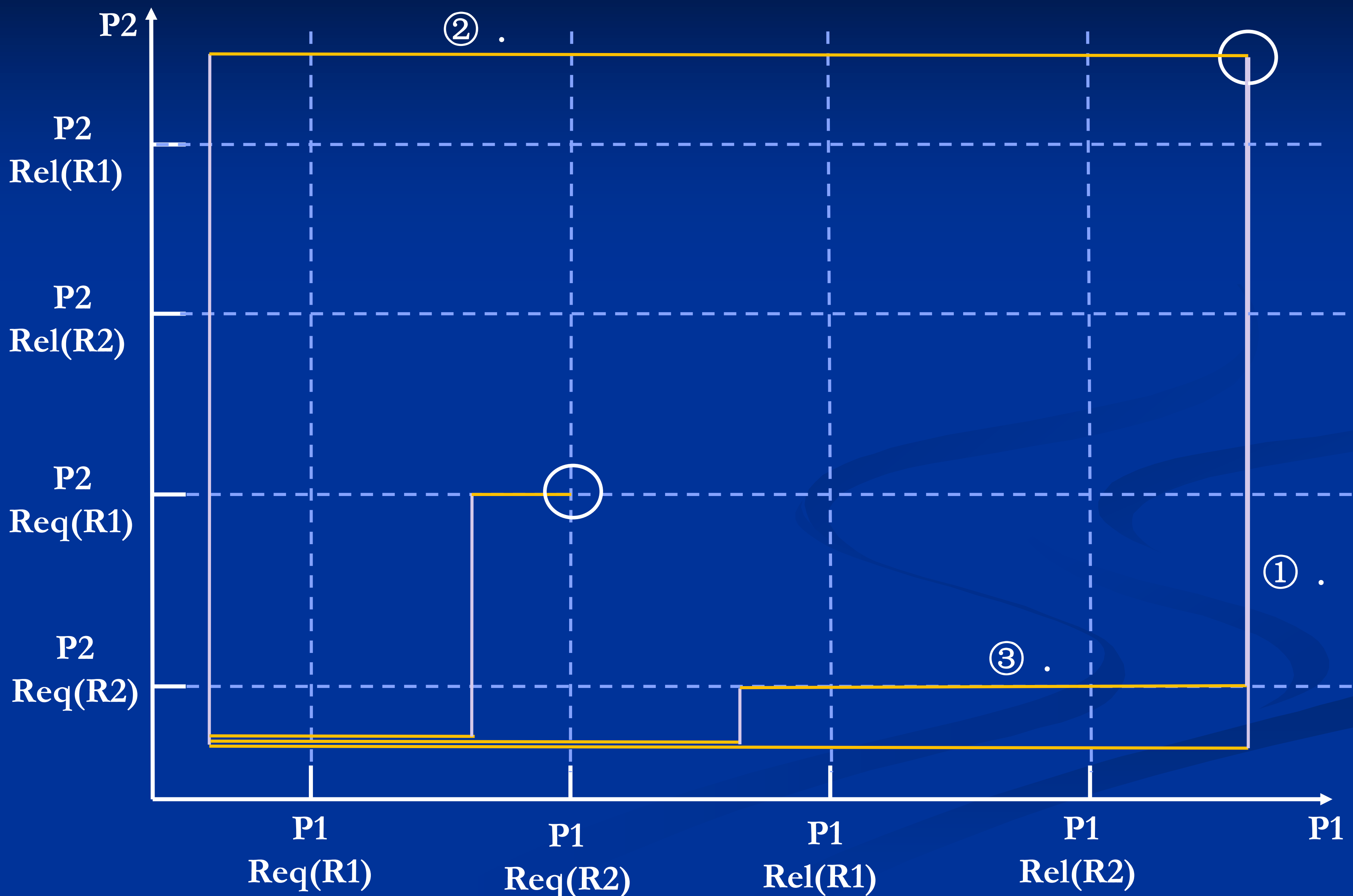
■ 示例：互斥条件与不剥夺条件



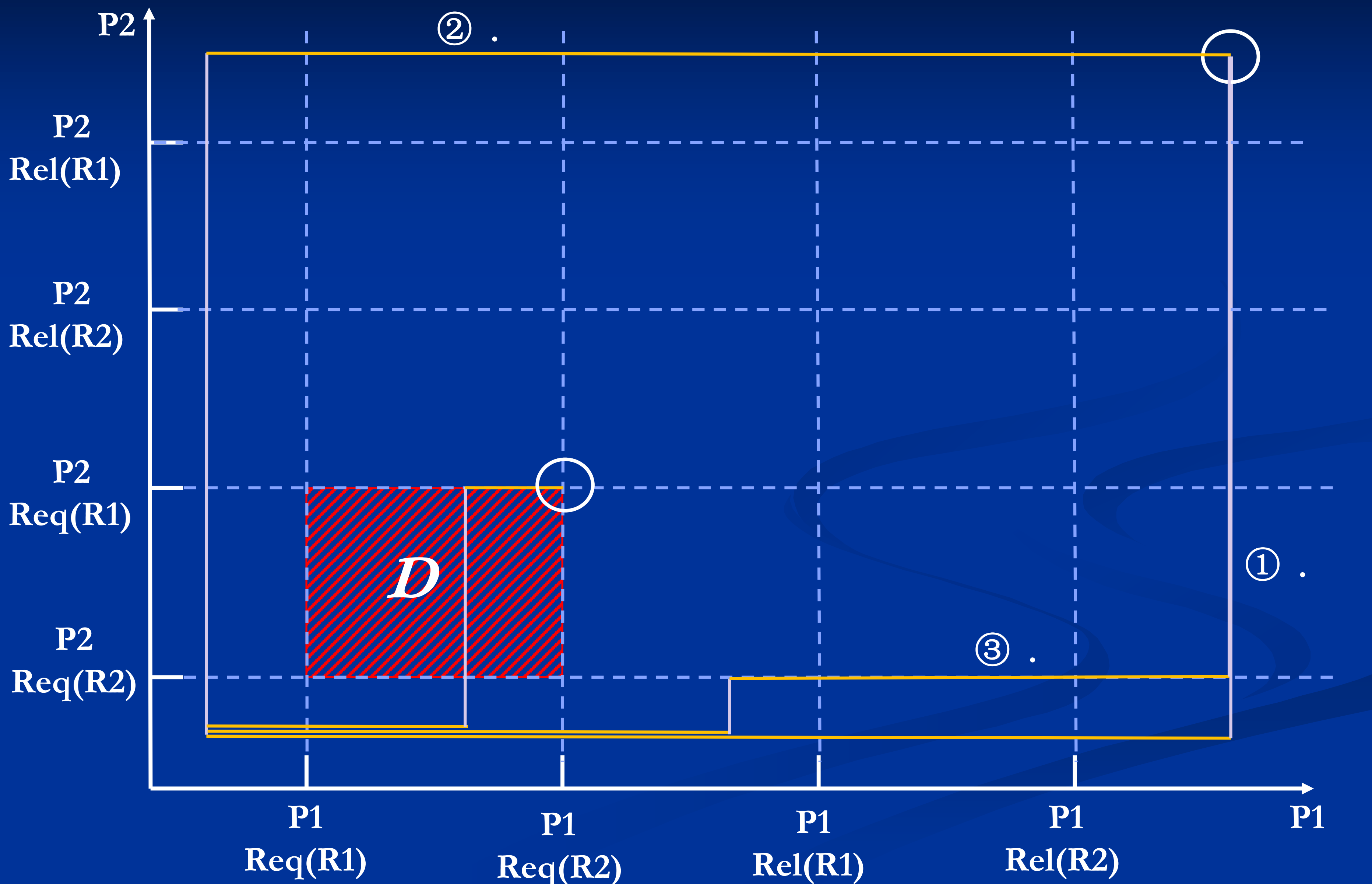
■ 示例：进程推进顺序与死锁



■ 示例：进程推进顺序与死锁



■ 示例：进程推进顺序与死锁



5. 银行家算法算例

- 进程{P0,P1,P2,P3,P4}共享资源{A,B,C}。某T0时刻资源状况如下：

	Max A B C	Allocation A B C	Need A B C	Available 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) T0时刻安全性；

(2) P1请求$Request_1$(1,0,2);

(3) P4请求$Request_4$(3,3,0);

(4) P0请求$Request_0$(0,2,0);

(1) T0时刻安全性：

■ T0时刻：

	Max A B C	Allocation A B C	Need A B C	Available 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	

4. 安全性算法

(1) 设置工作向量Work,长度m,Work := Available ;

(2) 设置状态向量Finish,长度n, Finish := false ;

(3) 从进程集合查找满足下列条件之进程P_k：

Finish[k] = false ;

$Need_k \leq Work$;

IF 未找到这样的进程 THEN GOTO (5)

(4) 执行如下操作：

$Work := Work + Allocation_k$

Finish[k] := true ;

GOTO (3)

(5) IF Finish = true THEN 安全

ELSE 不安全;

■ 调用安全性算法：

	Work A B C	Need A B C	Max A B C	Allocation A B C	Work + Allocation A B C	Finish
P1	3 3 2	1 2 2	3 2 2	2 0 0	5 3 2	Finish[1]=TRUE
P3	5 3 2	0 1 1	2 2 2	2 1 1	7 4 3	Finish[3]=TRUE
P0	7 4 3	7 4 3	7 5 3	0 1 0	7 5 3	Finish[0]=TRUE
P2	7 5 3	6 0 0	9 0 2	3 0 2	10 5 5	Finish[2]=TRUE
P4	10 5 5	4 3 1	4 3 3	0 0 2	10 5 7	Finish[4]=TRUE
	10 5 7					

安全性序列：<P1,P3,P0,P2,P4> ➔➔ 安全状态

3. 银行家算法（Pi进程提出资源请求$Request_i$ ）

(1) IF $Request_i$ not ≤ $Need_i$ THEN 出错；

(2) IF $Request_i$ not ≤ Available THEN Pi等待；

(3) 试分配，修改数据结构Allocation, Need, Available；

Available := Available - $Request_i$

$Allocation_i$:= $Allocation_i$ + $Request_i$

$Need_i$:= $Need_i$ - $Request_i$

(4) 执行安全性算法，检查此次资源分配后系统是否处于安全状态；

(5) IF 安全 THEN 正式分配

ELSE 取消试分配， Pi等待；

(2) P1请求$Request_1$(1,0,2);

- T0时刻:

	Max A B C	Allocation A B C	Need A B C	Available 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	

- T1时刻:

	Max A B C	Allocation A B C	Need A B C	Available 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	

(2) P1请求$Request_1$(1,0,2); ;

■ T1时刻:

	Max	Allocation	Need	Available
	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	

4. 安全性算法

(1) 设置工作向量Work,长度m,Work := Available ;

(2) 设置状态向量Finish,长度n, Finish := false ;

(3) 从进程集合查找满足下列条件之进程P_k:

Finish[k] = false ;

$Need_k \leq$ Work ;

IF 未找到这样的进程 THEN GOTO (5)

(4) 执行如下操作:

Work := Work + $Allocation_k$

Finish[k] := true ;

GOTO (3)

(5) IF Finish = true THEN 安全

ELSE 不安全;

■ 调用安全性算法:

	Work	Need	Max	Allocation	Work + Allocation	Finish
	A B C	A B C	A B C	A B C	A B C	
P1	2 3 0	0 2 0	3 2 2	3 0 2	5 3 2	Finish[1]=TRUE
P3	5 3 2	0 1 1	2 2 2	2 1 1	7 4 3	Finish[3]=TRUE
P0	7 4 3	7 4 3	7 5 3	0 1 0	7 5 3	Finish[0]=TRUE
P2	7 5 3	6 0 0	9 0 2	3 0 2	10 5 5	Finish[2]=TRUE
P4	10 5 5	4 3 1	4 3 3	0 0 2	10 5 7	Finish[4]=TRUE
	10 5 7					

安全性序列: <P1,P3,P0,P2,P4> ➔➔ 安全状态

3. 银行家算法（Pi进程提出资源请求$Request_i$）

(1) IF $Request_i$ not ≤ $Need_i$ THEN 出错；

(2) IF $Request_i$ not ≤ Available THEN Pi等待；

(3) 试分配，修改数据结构Allocation, Need, Available；

Available := Available - $Request_i$

$Allocation_i$:= $Allocation_i$ + $Request_i$

$Need_i$:= $Need_i$ - $Request_i$

(4) 执行安全性算法，检查此次资源分配后系统是否处于安全状态；

(5) IF 安全 THEN 正式分配

ELSE 取消试分配， Pi等待；

(3) P4请求$Request_4$(3,3,0);

■ T1时刻:

	Max	Allocation	Need	Available
	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	

资源不足，P4等待

3. 银行家算法（Pi进程提出资源请求$Request_i$ ）

(1) IF $Request_i$ not ≤ $Need_i$ THEN 出错；

(2) IF $Request_i$ not ≤ Available THEN Pi等待；

(3) 试分配，修改数据结构Allocation, Need, Available；

Available := Available - $Request_i$

$Allocation_i$:= $Allocation_i$ + $Request_i$

$Need_i$:= $Need_i$ - $Request_i$

(4) 执行安全性算法，检查此次资源分配后系统是否处于安全状态；

(5) IF 安全 THEN 正式分配

ELSE 取消试分配， Pi等待；

(4) P0请求$request_0$(0,2,0);

- T1时刻:

	Max A B C	Allocation A B C	Need A B C	Available 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	

- T2时刻:

	Max A B C	Allocation A B C	Need A B C	Available A B C
P0	7 5 3	0 3 0	7 2 3	2 1 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	

(4) P0请求$request_0(0,2,0)$;

■ T2时刻:

	Max A B C	Allocation A B C	Need A B C	Available A B C
P0	7 5 3	0 3 0	7 2 3	2 1 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	

4. 安全性算法

(1) 设置工作向量Work,长度m,Work := Available ;

(2) 设置状态向量Finish,长度n, Finish := false ;

(3) 从进程集合查找满足下列条件之进程P_k:

Finish[k] = false ;

$Need_k \le Work$;

IF 未找到这样的进程 THEN GOTO (5)

(4) 执行如下操作:

$Work := Work + Allocation_k$

Finish[k] := true ;

GOTO (3)

(5) IF Finish = true THEN 安全

ELSE 不安全;

- 调用安全性算法:

	Work A B C	Need A B C	Max A B C	Allocation A B C	Work + Allocation A B C	Finish
	2 1 0					

安全性序列不存在:<?,?,?,?,?> ➔➔ 不安全状态