

LINUX OPERATING SYSTEM

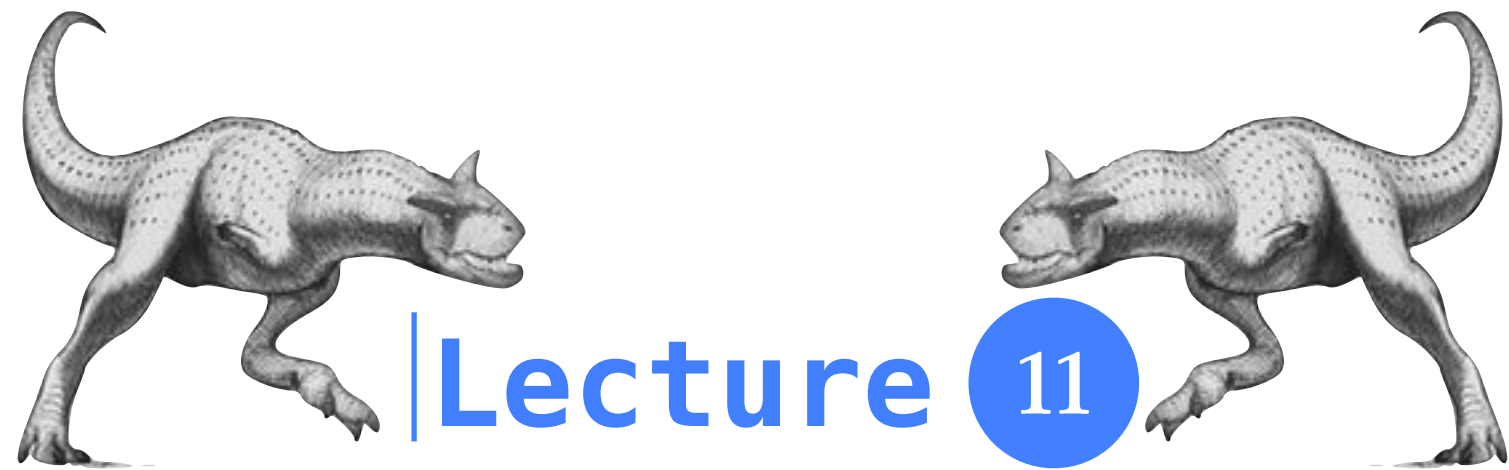
YANG

LINUX操作系统（双语）





双语课→课件内容中英混排



DeadLocks

本讲内容

 死锁的特征

 死锁的预防

 死锁的避免

 死锁的检测

死锁的特征

哲学家用餐死锁问题

当所有人同时拿到一侧的筷子时，发生永远等待现象（即死锁）。

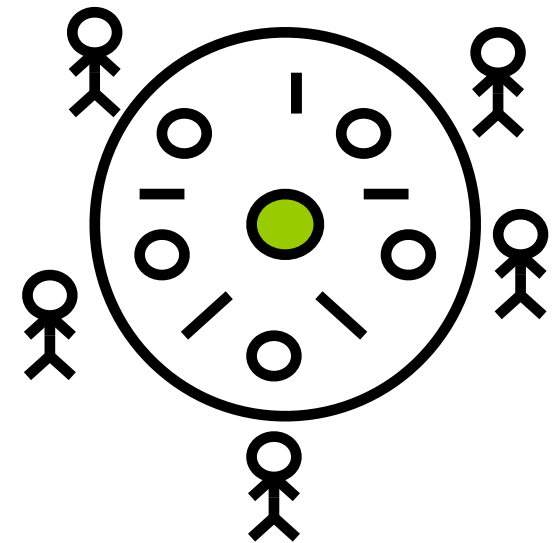
有若种办法可避免死锁：

至多允许四个哲学家同时吃；

奇数号先取左手边的筷子，偶数号先取右手边的筷子；

每个哲学家取到手边的两根筷子才吃，否则一根也不取。

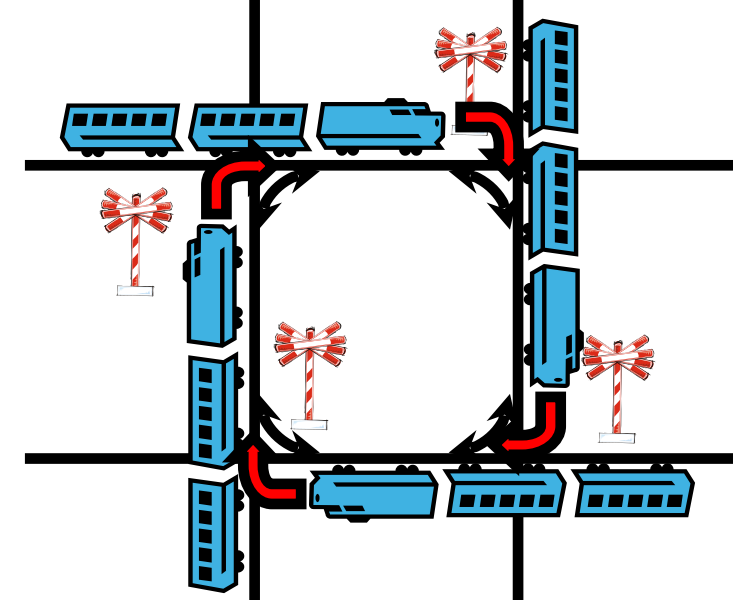
进程访问资源流程：申请 ➡ 使用 ➡ 释放



DEADLOCK

- 🧠 In a multiprogramming environment, several processes may compete for a finite number of resources.
- 🧠 A process requests resources; if the resources are **not available** at that time, the process enters a **waiting state**.
- 🧠 Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes.
- 🧠 This situation is called a **deadlock**.

死锁与饥饿



🧠 **饥饿**: 进程长时间的等待

🧠 e.g.低优先级进程总是等待高优先级所占有的进程

🧠 **死锁**: 循环等待资源

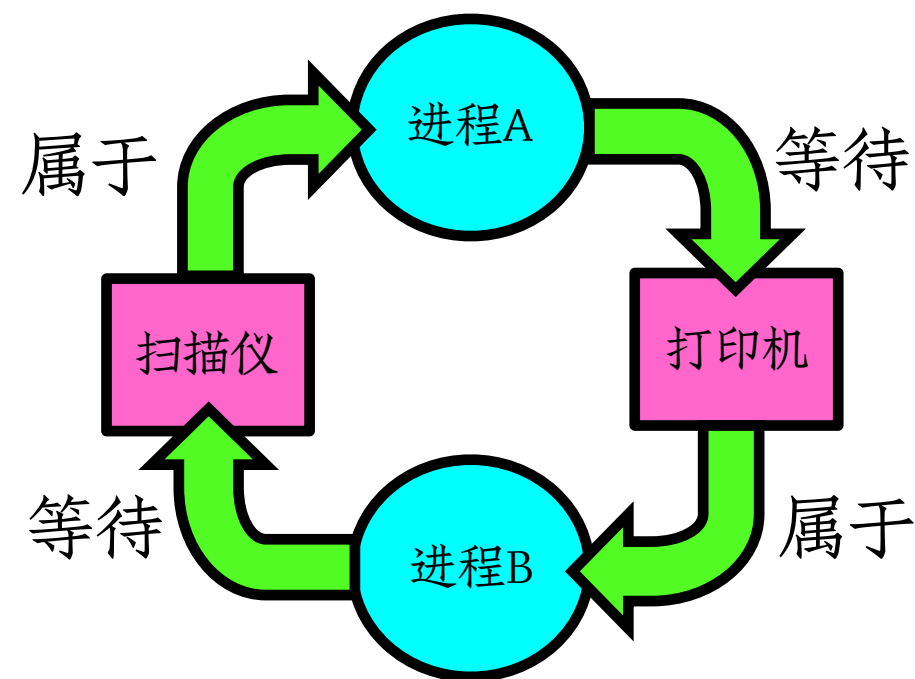
🧠 A和B分别占有打印机和扫描仪

🧠 同时分别申请扫描仪和打印机

🧠 死锁 \Rightarrow 饥饿 (反之不亦然)

🧠 饥饿可能终止

🧠 如果无外部干涉, 死锁无法终止



产生死锁的四个必要条件

互斥使用

-  一个时刻，一个资源仅能被一个进程占有

不可剥夺

-  除了资源占有进程主动释放资源，其它进程都不可抢夺其资源

占有和等待

-  一个进程请求资源得不到满足等待时，不释放已占有资源

循环等待(上面三个条件同时存在产生的结果)

-  每一个进程分别等待它前一个进程所占有的资源

METHODS FOR HANDLING DEADLOCKS

- 🧠 Deadlocks are NOT allowed to appear. We must prevent or avoid deadlock state.
- 🧠 Deadlocks are allowed to appear, but the system can detect them and recover.
- 🧠 We pretend that deadlocks never occur in the system.


Which method is applied by common operating-system?

死锁的解决方案

死锁的防止 (Prevention)

 破坏四个必要条件之一

死锁的避免 (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

死锁的检测和恢复 (Detection)

 允许死锁的发生，系统及时地检测死锁并解除它

死锁的防止

破坏死锁任一必要条件

 互斥使用

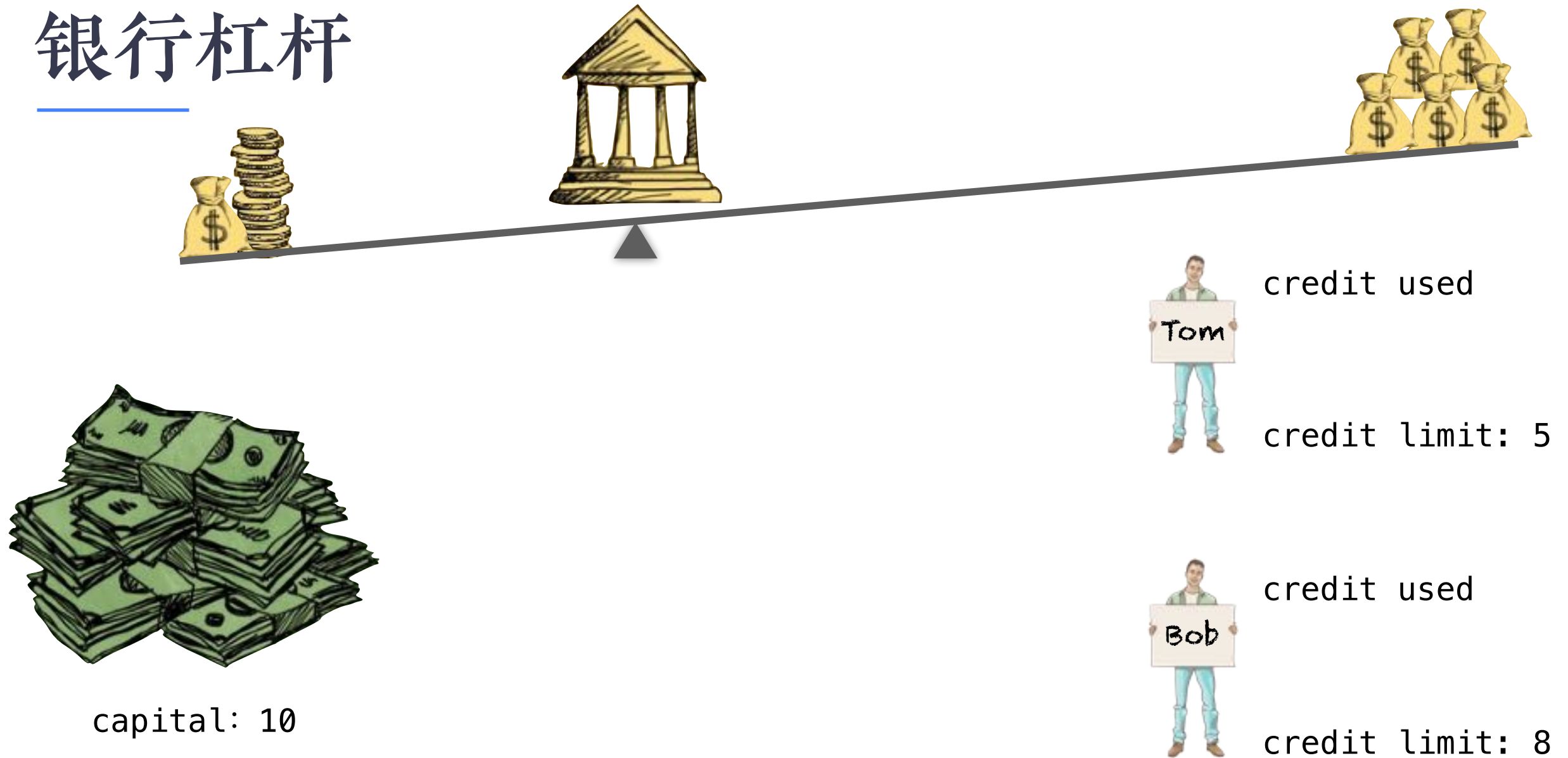
 不可剥夺

 占有和等待

 循环等待

死锁的避免

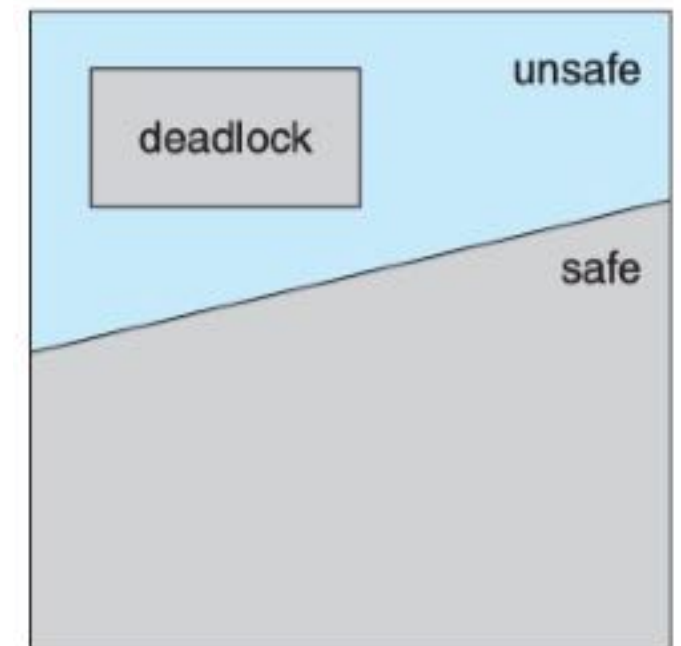
银行杠杆



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

安全状态(SAFE STATE)

- 💡 A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.
- 💡 If no such sequence exists, then the system state is said to be unsafe.
- 💡 A safe state is NOT a deadlocked state.
- 💡 An unsafe state MAY lead to a deadlock.



死锁的避免

- 💡 系统对进程的每一次资源申请都进行详细的计算，根据结果决定是分配资源还是让其等待，确保系统始终处于安全状态，避免死锁的发生。
- 💡 银行家算法 (Banker's algorithm)
 - 💡 已知系统中所有资源的种类和数量
 - 💡 已知进程所需要的各类资源最大需求量
 - 💡 该算法可以计算出当前的系统状态是否安全(寻找安全序列)

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	

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>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

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

安全算法

存在安全序列{ P_1, P_3, P_4, P_2, P_0 }, 系统处于安全状态

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	3 3 2
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	

如果此时 P_1 发出请求(1,0,2), 是否批准?

如果此时 P_0 发出请求(0,2,0), 是否批准?

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

🧠 进程运行前就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

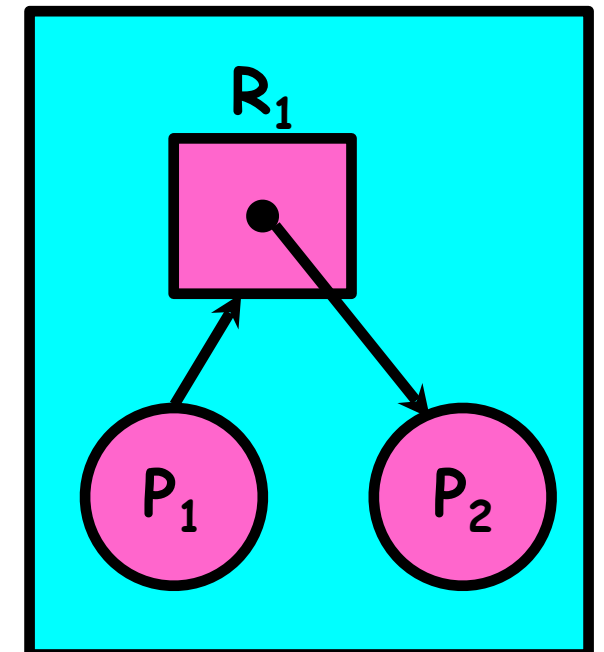
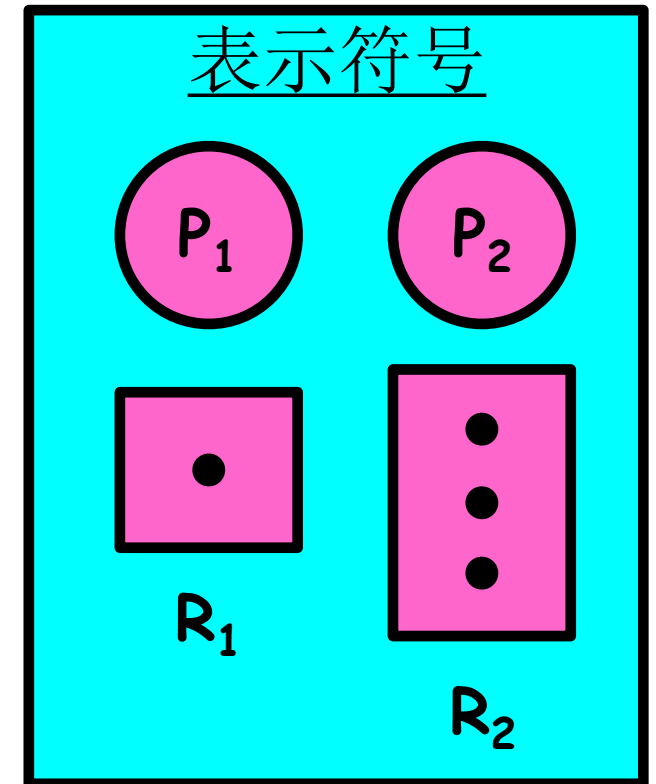
死锁的检测

死锁的检测与恢复

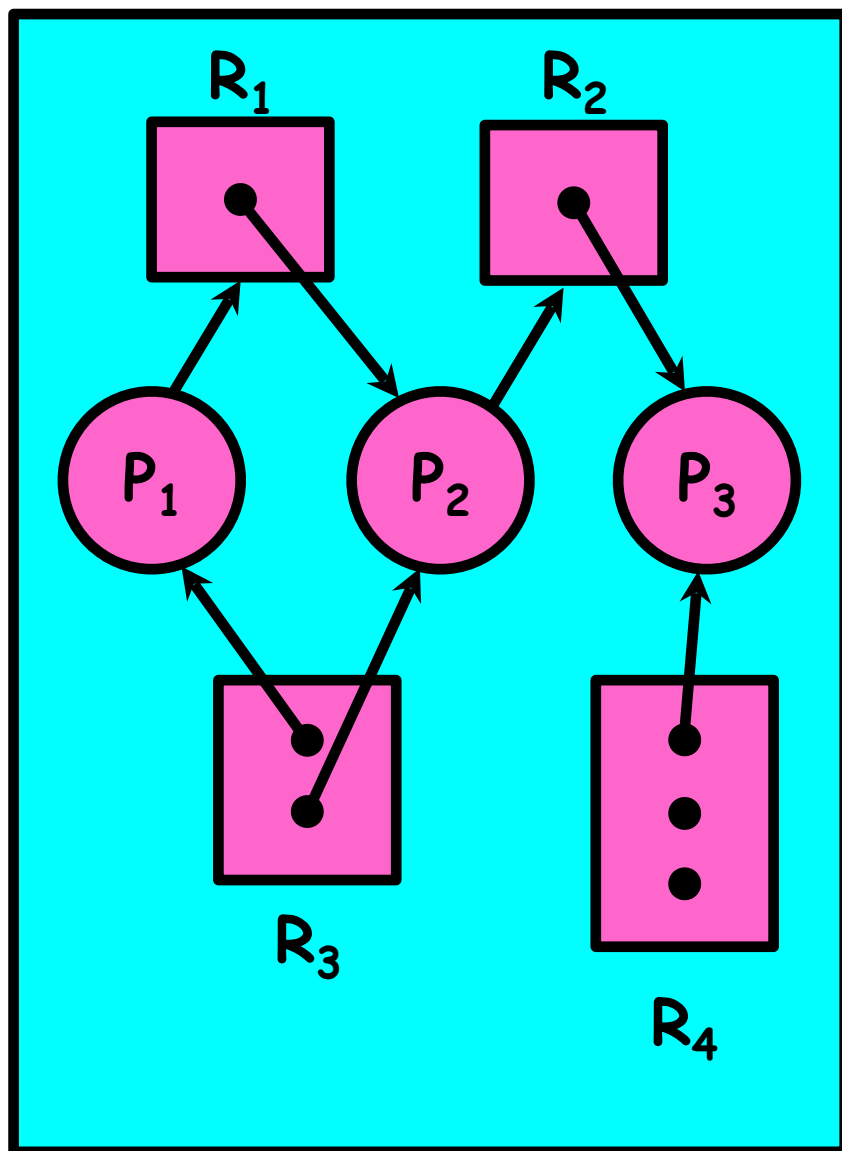
- ④ 允许死锁发生，操作系统不断监视系统进展情况，判断死锁是否发生
- ④ 一旦死锁发生则采取专门的措施，解除死锁并以最小的代价恢复操作系统运行
- ④ 死锁检测的时机
 - ④ 当进程等待时检测死锁（系统开销大）
 - ④ 定时检测
 - ④ 系统资源利用率下降时检测死锁

资源分配图表示法

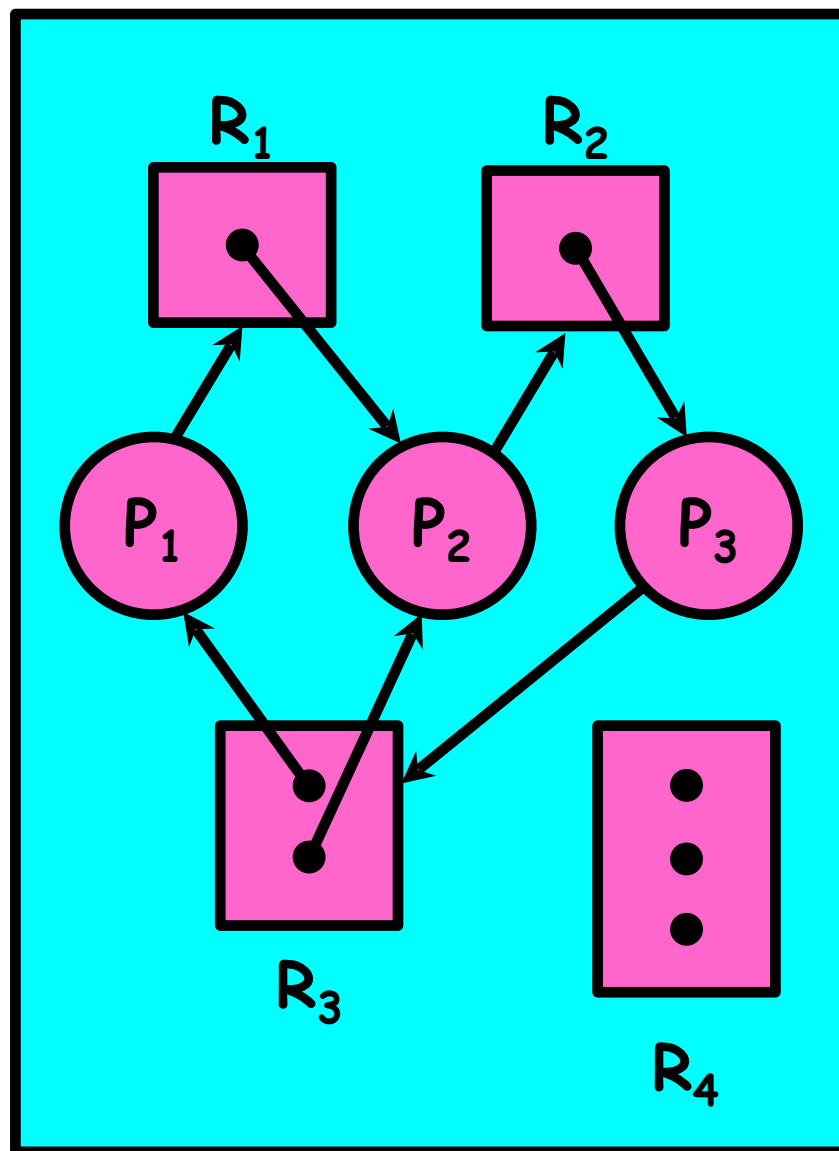
- 🧠 资源类（资源的不同类型）
- 🧠 资源实例（存在于每个资源中）
- 🧠 进程
- 🧠 申请边
- 🧠 分配边



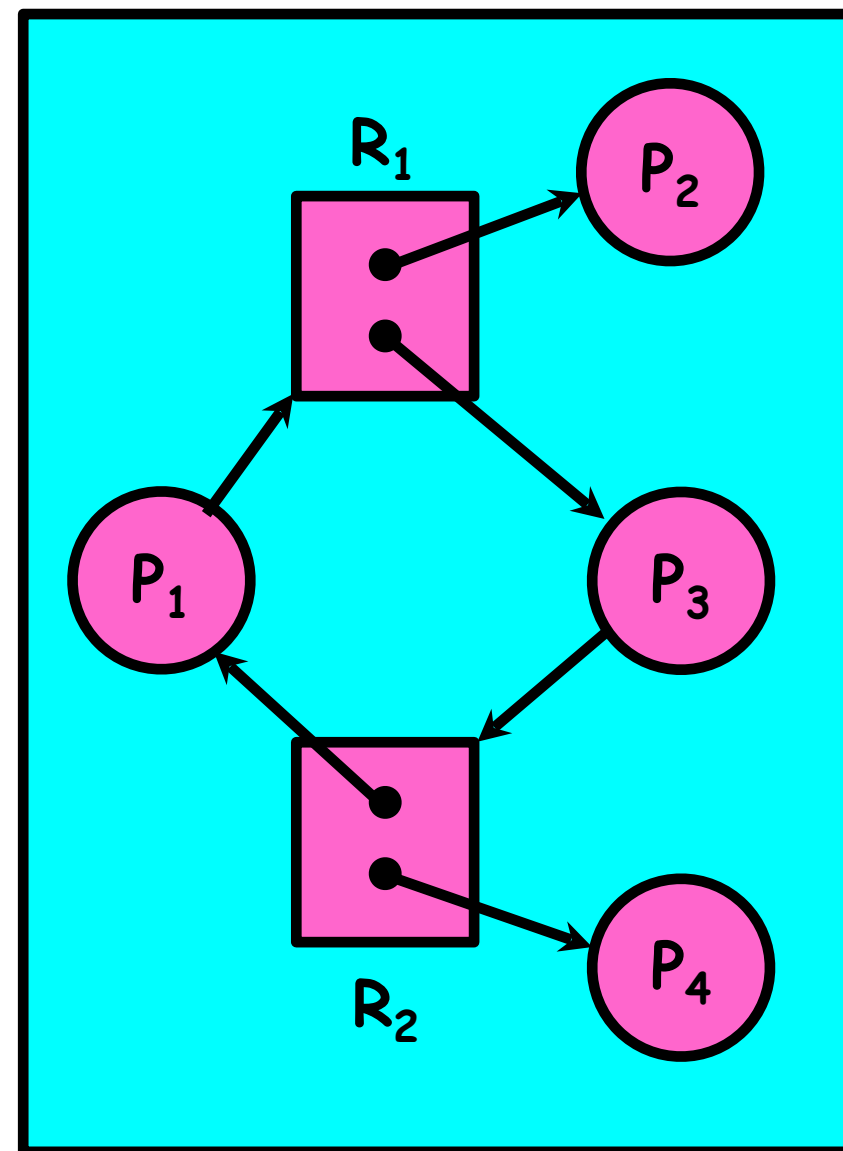
资源分配图图例



无环无死锁

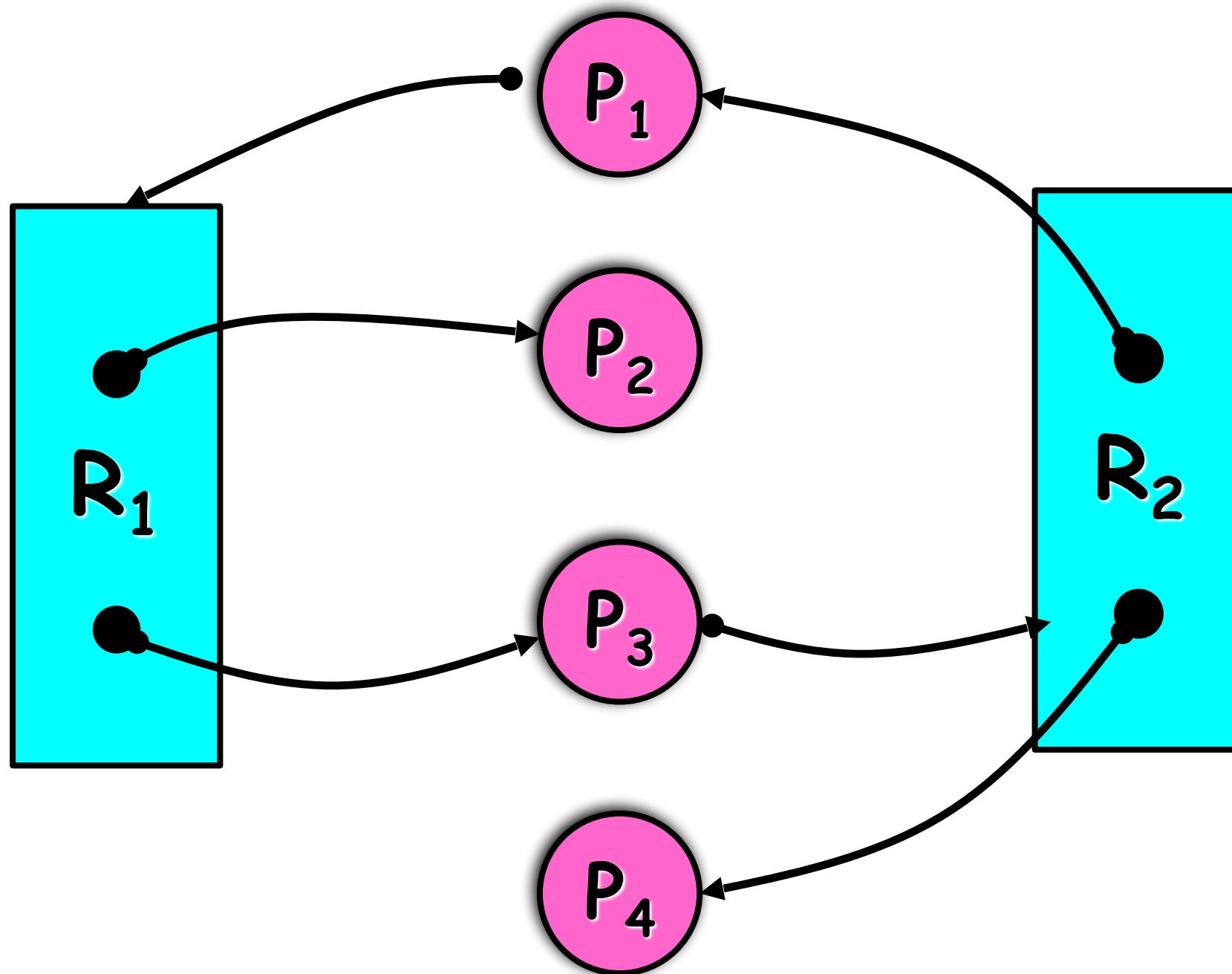


有环死锁



有环但无死锁

资源分配图的简化





死锁定理

- 🧠 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**。
- 🧠 可完全简化
- 🧠 不可完全简化
- 🧠 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**。




死锁的解除

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

HOW OS DO TO DEADLOCKS?

- 🧠 In the absence of algorithms to detect and recover from deadlocks, we may arrive at a situation in which the system is in a deadlocked state yet has no way of recognizing what has happened. In this case, the undetected deadlock will cause the system's performance to deteriorate, because resources are being held by processes that cannot run and because more and more processes, as they make requests for resources, will enter a deadlocked state. Eventually, the system will stop functioning and will need to be restarted manually.
- 🧠 Although this method may not seem to be a viable approach to the deadlock problem, it is nevertheless used in most operating systems.

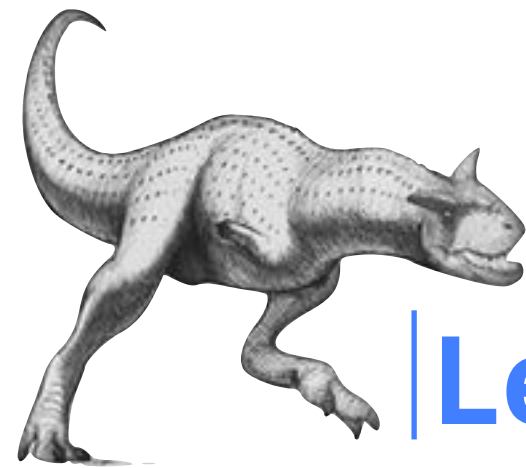
下期预告

 下次直播时间：3月13日 上午9:30

 课程内容

 Lecture 12 Memory Management

 Q&A



Lecture 11

The End