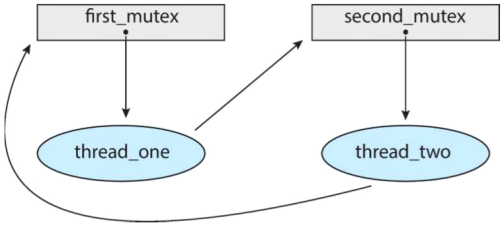
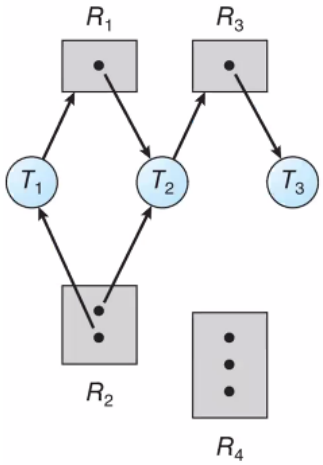
8 Deadlock

每个进程使用资源需要三个步骤：**申请** (request)、**使用** (use)、**释放** (release)。 用**资源分配图** (resource-allocation graph, RAG) 来表示资源分配情况：点表示一个资源的一个 item，从资源指向线程表示资源已经分配；从线程指向资源表示正在申请，以及释放的不需要画出。



- 申请边 (request edge)：$P_i \rightarrow R_j$；
- 分配边 (assignment edge)：$R_j \rightarrow P_i$。

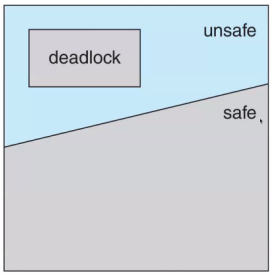


死锁的条件：

- **互斥** (mutual exclusion)：同时只有一个进程可以访问资源；
- **持有和等待** (hold and wait)：每个进程持有至少一个资源，而且正在等待另外一个资源。
- **非抢占式** (no preemption)：进程不能抢占资源；
- **循环等待** (circular waiting)：进程 1 等待进程 2； 进程 2 等待进程 3； .....进程 n 等待进程 1 。

死锁 可以在资源分配图中用环（有向全连接图）表示。

- 如果资源分配图中没有环，系统处于**安全状态** (safe state)，不可能有死锁；
- 如果资源分配图包含一个环，系统处于**不安全状态** (unsafe state)，可能有死锁；
    - 如果每个进程只有一个实例，那么一定死锁；
    - 如果每个进程有超过一个实例，那么可能死锁 (possibility of deadlock)。



需要运用 **死锁预防机制** (Prevention) 与 **死锁避免机制** (Avoidance)，保证系统不会进入不安全状态（不会产生死锁）。

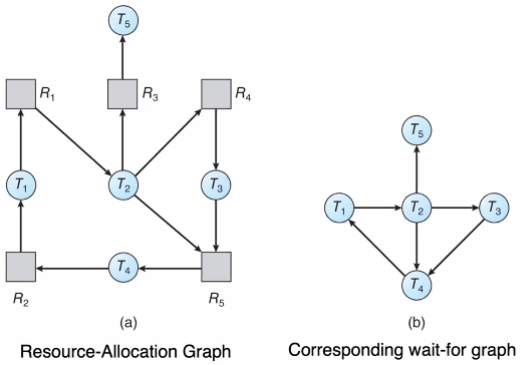
- **死锁预防**：还没有死锁情况出现，从根本上（四个条件）预防出现死锁；
- **死锁避免**：已知信息，如何调度避免出现死锁。

死锁预防 (Deadlock Prevention)：破除死锁四条件之一。

- **互斥**：对于只读共享文件并不要求，对于不共享文件也不要求，默认破除；其余情况无法破除改条件，不予以考虑。
- **持有和等待**：保证任何进程申请锁的时候不能占有其他锁；这种解决方案会使得资源利用率过低 (low resource utilization)，可能导致饿死现象，不予以考虑。
- **非抢占式**：改为按照优先级抢占式即可，可行。但是仍然存在风险，因为同一优先级的进程仍可能存在死锁。
- **循环等待**：生成一个关于锁的 ordering（用链表），进程按 ordering 申请锁。

死锁避免 (Deadlock Avoidance)：需要系统有一些附加资源，用死锁避免算法来动态保证不会产生死锁。

等待图：将死锁关系从资源分配图转化为等待图，$P_1 \to P_2$ 表示 P_1 需要的资源被 P_2 访问。



找到等待图上边最多的进程，牺牲这个进程来使得其他进程运行高效。

资源分配图的作用

1. 判断系统处于安全状态还是不安全状态；
2. **死锁避免**，用虚线表示将要分配的资源来判断是否有死锁；
3. 推导出等待图 (wait-for graph)，找到死锁位置。

银行家算法

一个资源只有一个实例时，用资源分配图来解决。否则，用银行家算法。

- **Available**：`Available[j] = k`，说明 R_j 还有 k 个实例可用；
- **Max**：`Max[i, j] = k`，说明 P_i 最多需要 R_j 的 k 个实例来完成任务；
- **Allocation**：`Allocation[i, j] = k`，说明 P_i 当前占有 R_j 的 k 个实例；
- **Need**：`Need[i, j] = k`，说明 P_i 最多还需要 R_j 的 k 个实例来完成任务。

于是有 `Need[i, j] = Max[i, j] - Allocation[i, j]`。

如果给出每个资源的总实例数量，也可以算出每个 `Available[j]`。

- 考虑将所有 Available 的资源都分配个一个进程，看一下是否可以执行完；
- 如果不存在这样一个进程，那么一定产生了死锁，需要进行处理；
- 如果存在这样一个进程，则执行该进程，然后等待该进程结束后即可释放该进程占用的其他资源；
- 重复此过程，直到发现死锁或者产生了一个进程执行序列。

存在死锁的解决方法：终止进程。如何选择终止的进程？

1. 按照优先级；
2. 按照进程已经完成的程度（已经执行了多长时间，还需要多长时间）；
3. 进程已经占用的资源数量；

4. 进程还需要的资源数量；
5. 需要终止的进程数量；
6. 是交互式的进程还是批处理的进程。

解决方法

- **选择牺牲者** (selecting a victim)：最小花费的牺牲者，保存PCB资源；
- **回滚** (rollback)：回滚到之前的某个安全状态，之前的安全状态保存在快照 (snapshot) 中。
- 可能存在的问题：牺牲者可能**饿死**（每次都被选中）。
- 一种方法是让进程一次性申请所有可能被用到的资源；
- 或者使用死锁避免算法来动态保证不会产生