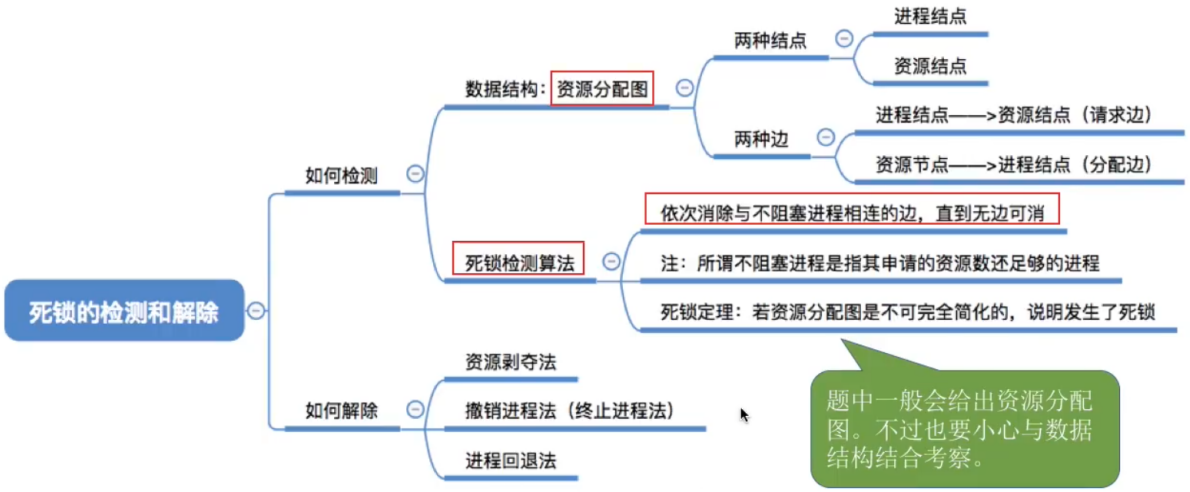
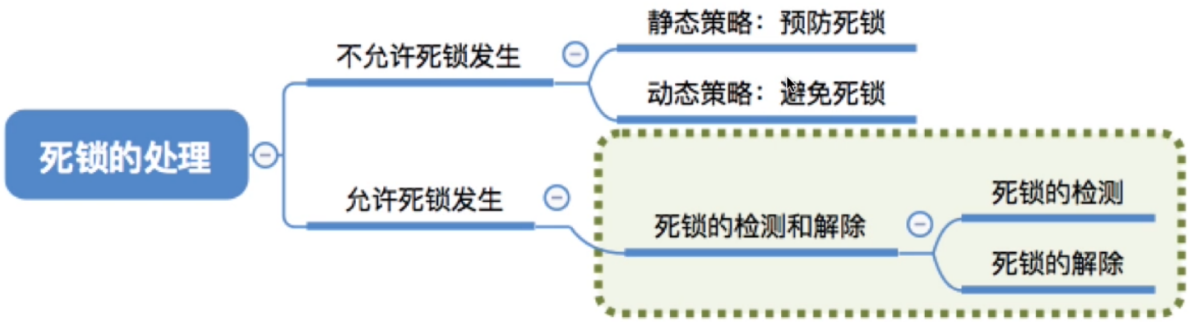
死锁检测和解除





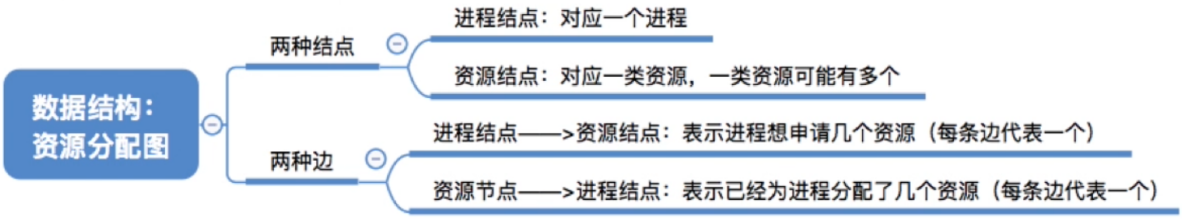
如果系统中既不采取预防死锁的措施，也不采取避免死锁的措施，系统就很可能发生死锁。在这种情况下，系统应当提供两个算法：

1. 死锁检测算法：用于检测系统状态，以确定系统中是否发生了死锁。
2. 死锁解除算法：当认定系统中已经发生了死锁，利用该算法可将系统从死锁状态中解脱出来。

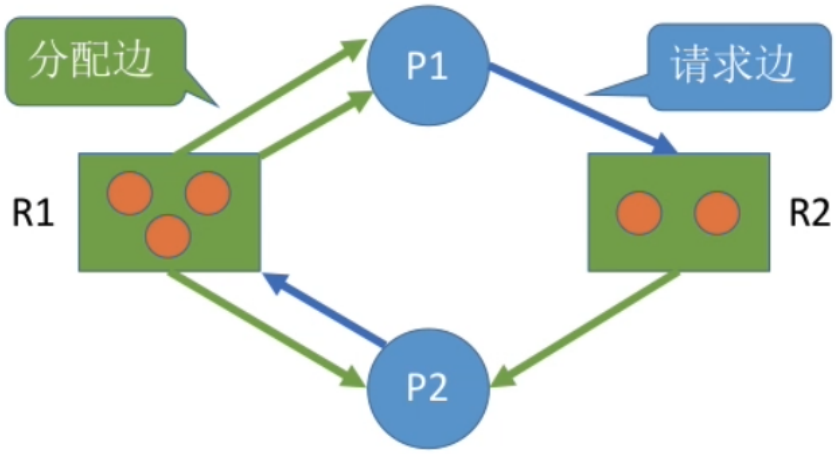
死锁的检测

为了能对系统是否已发生了死锁进行检测，必须：

1. 用某种数据结构来保存资源的请求和分配信息；
2. 提供一种算法，利用上述信息来检测系统是否已进入死锁状态。

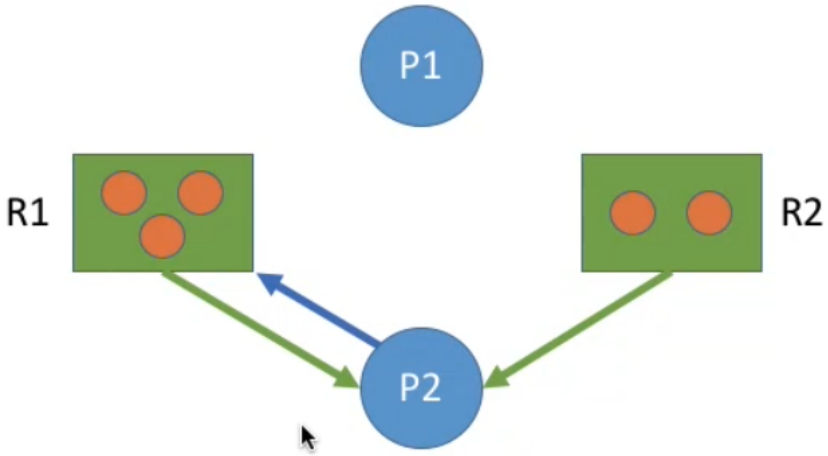


一般用矩形表示资源结点，矩形中的小圆代表该类资源的数量



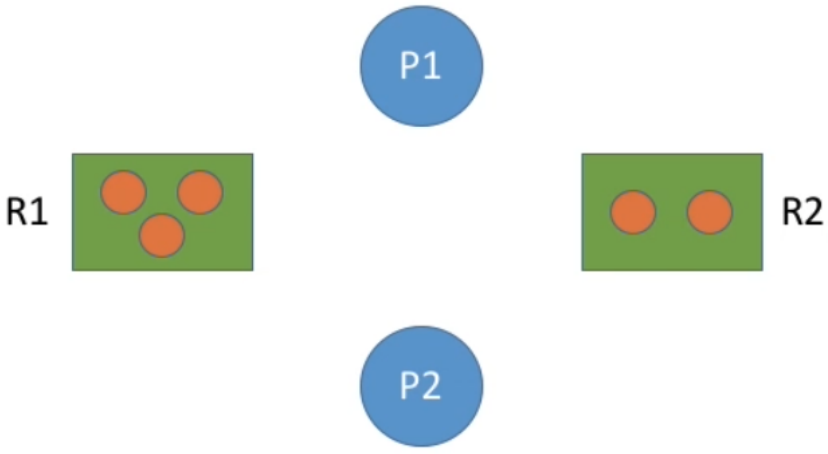
如果系统中剩余的可用资源数足够满足进程的需求，那么这个进程暂时是不会阻塞的，可以顺利地执行下去。

如果这个进程执行结束了把资源归还系统，就可能使某些正在等待资源的进程被激活，并顺利地执行下去。

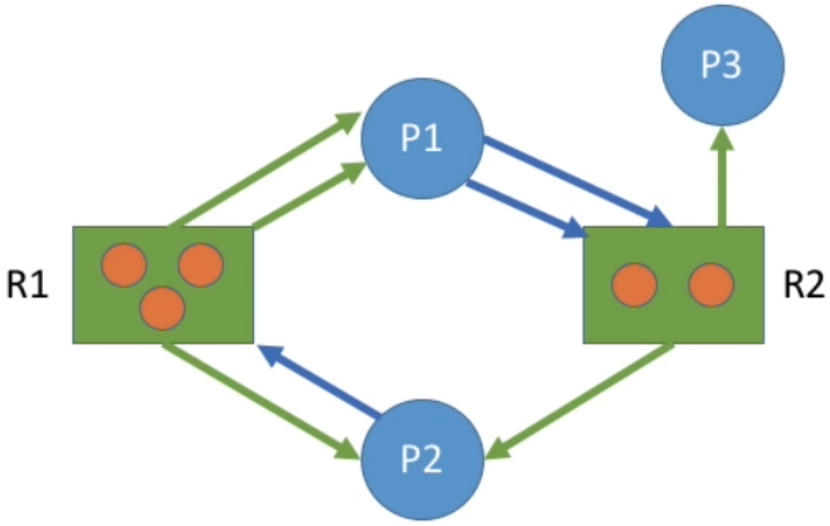


相应的，这些被激活的进程执行完了之后又会归还一些资源，这样可能又会激活另外一些阻塞的进程...

如果按上述过程分析，最终能消除所有边，就称这个图是可完全简化的。此时一定没有发生死锁（相当于能找到一个安全序列）



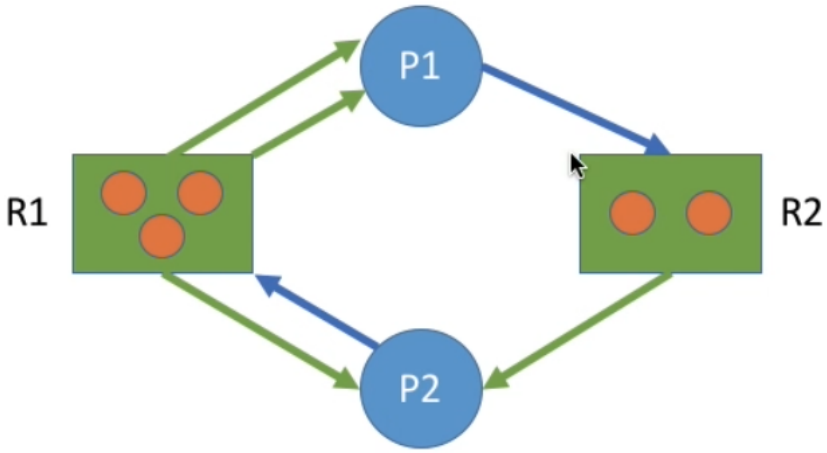
如果最终不能消除所有边，那么此时就是发生了死锁



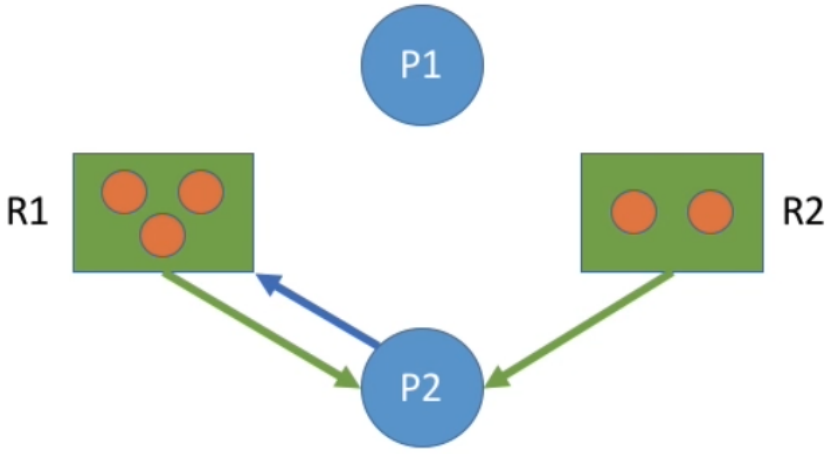
最终还连着边的那些进程就是处于死锁状态的进程

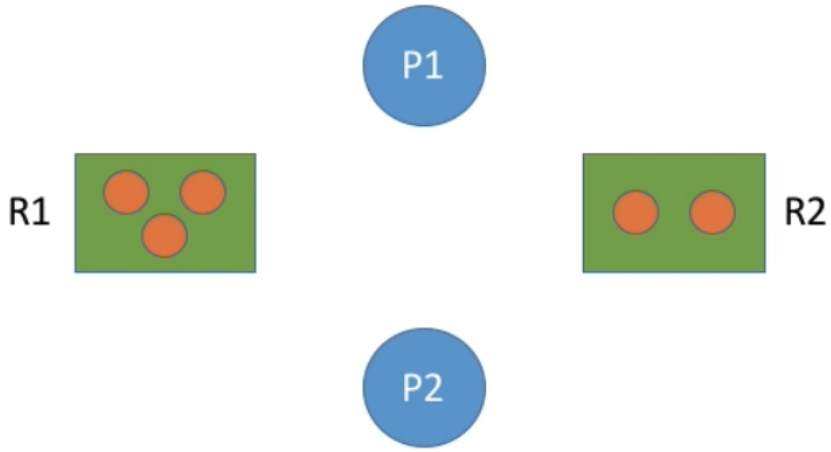
检测死锁的算法：

1. 在资源分配图中，找出既不阻塞又不是孤点的进程Pi（即找出一条有向边与它相连，且该有向边对应资源的申请数量小于等于系统中已有空闲资源数量。如图，R1没有空闲资源，R2有一个空闲资源。若所有的连接该进程的边均满足上述条件，则这个进程能继续运行直至完成，然后释放它所占有的所有资源）。消去它所有的请求边和分配边，使之成为孤立的结点。下图中，P1是满足这一条件的进程结点，于是将P1的所有边消去。



2. 进程Pi所释放的资源，可以唤醒某些因等待这些资源而阻塞的进程，原来的阻塞进程可能变为非阻塞进程。在下图中，P2就满足这样的条件。根据1中的方法进行一系列简化后，若能消去图中所有的边，则称该图是可完全简化的。





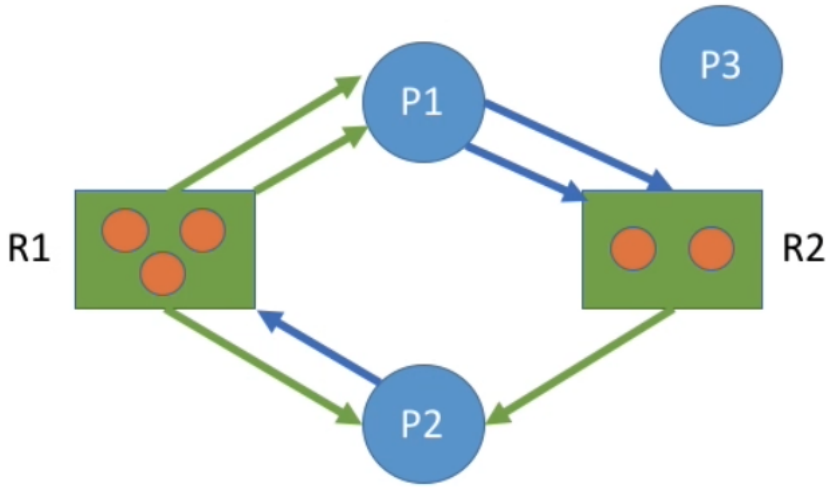
死锁定理：如果某时刻系统的资源分配图是不可完全简化的，那么此时系统死锁

既然检测到死锁，那就要想办法解除

死锁的解除

一旦检测出死锁的发生，就应该立即解除死锁。

补充：并不是系统中所有的进程都是死锁状态，用死锁检测算法化简资源分配图后，还连着边的那些进程就是死锁进程



1. 资源剥夺法。挂起（暂时放到外存上）某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但是应防止被挂起的进程长时间得不到资源而饥饿。
2. 撤销进程法（或称终止进程法）。强制撤销部分、甚至全部死锁进程，并剥夺这些进程的资源。这种方式的优点是实现简单，但所付出的代价可能会很大。因为有些进程可能已经运行了很长时间，已经接近结束了，一旦被终止可谓功亏一篑，以后还得从头再来。
3. 进程回退法。让一个或多个死锁进程回退到足以避免死锁的地步。这就要求系统要记录进程的历史信息，设置还原点。

如何决定“对谁动手”

1. 进程优先级
2. 已执行多长时间
3. 还要多久能完成
4. 进程已经使用了多少资源
5. 进程是交互式的还是批处理式的