# 操作系统复习

第五章： 进程同步

第五章总纲：

* Background
* The Critical-Section Problem
* Peterson’s Solution\*
* Synchronization Hardware
* Mutex Locks
* Semaphores
* Classic Problems of Synchronization
* Monitors\*
* Synchronization Examples
* Alternative Approaches

**#背景**

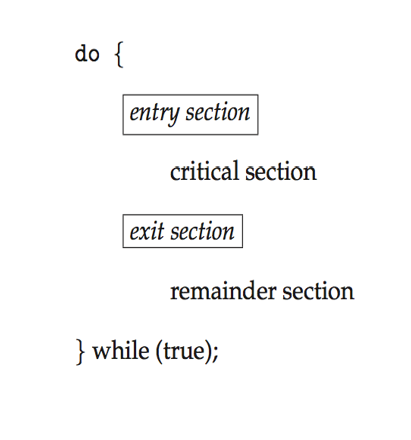
生产者消费者问题会出现不同步的情况！

**\*竞争条件**

多个进程并发访问和操作同一数据且执行结果与访问发生的特定顺序有关，称为竞争条件（Race condition）。为避免竞争条件，需要确保一段时间内只有一个进程能操作变量，为此，我们需要进程同步。

**#临界区问题（The Critical-section problem）**

每个进程中的对同一个变量进行更改、更新一个表、写一个文件等的代码段，称为**临界区。**



**没有两个进程可同时在临界区内执行。**

**\*临界区问题的解答必须满足如下三个要求：**

1. **互斥（Mutual exclusion）**：如果进程P在其临界区内执行，那么其他进程都不能在其临界区内执行。
2. **前进（Progress）：**如果没有进程在其临界区内执行且有进程需要进入临界区，那么只有那些不在剩余区内执行的进程可参加选择。
3. **有限等待（Bounded waiting）：**从一个进程做出进入临界区的请求，直到该请求允许为止，其他进程允许进入其临界区的次数有上限。

**\*有两种方法用于处理操作系统内的临界区问题：**

1. **Preemptive(抢占式)：**允许处于内核模式的进程被抢占。
2. **Non-preemptive(非抢占式)：**非抢占内核不允许处于内核模式的进程被抢占。

**#Peterson 算法**

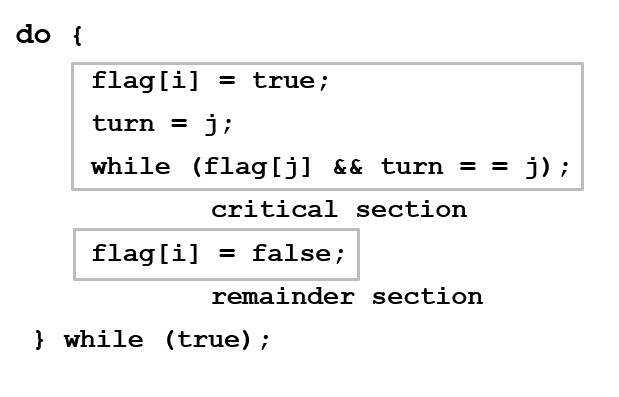
适用于两个进程在临界区与剩余区间交替执行。

例：

假设两个进程P0和P1，他们共享两个数据项：

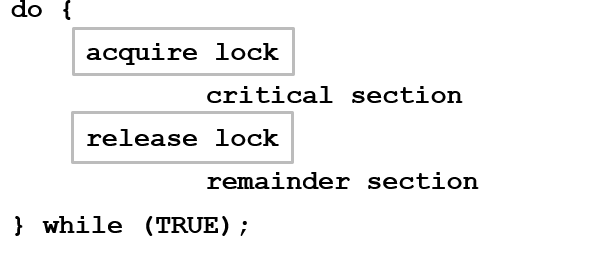
* + int turn;
  + Boolean flag[2]

算法描述：进程Pi的结构，turn表示哪个进程可以进临界区，flag表示哪个进程想要进临界区。



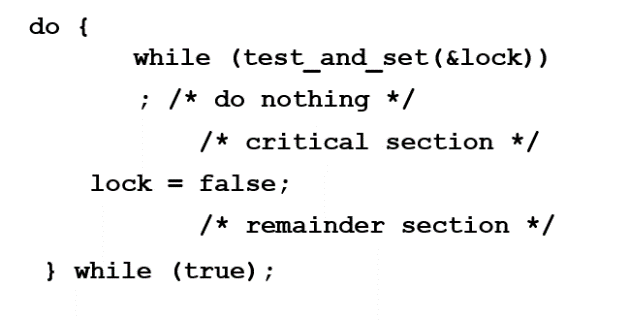
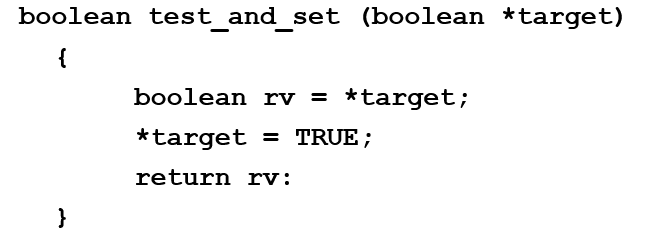
一开始这样设置表示，如果进程Pj想要进临界区，那么它可以进。当flag同时为true时谁的turn后写入，谁可以进入临界区。

**#硬件同步**

锁：在任何进程进入临界区都要得到锁，出临界区都要释放锁。

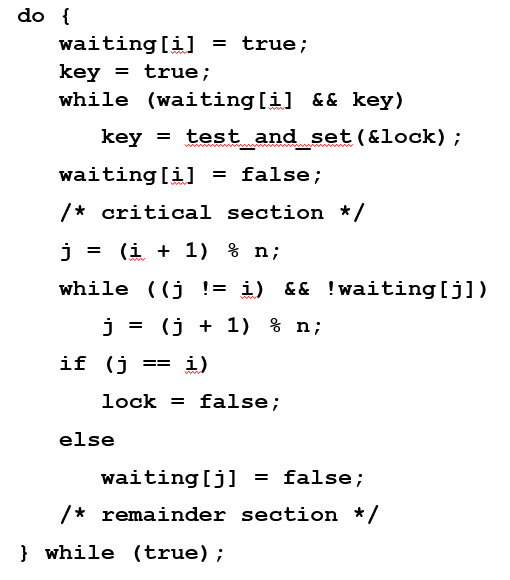
原子地（Atomic）：不可中断，必须执行完。

下面的代码用原子函数实现了锁：



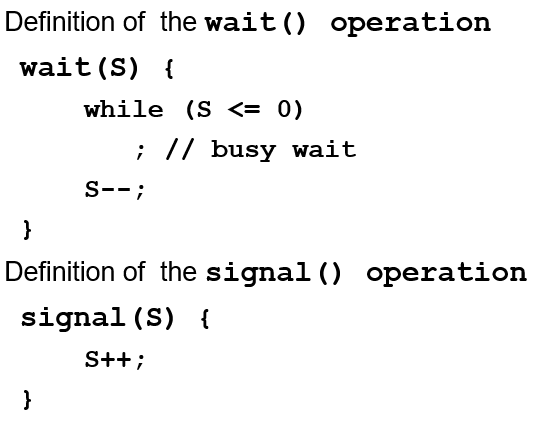
Lock一开始如果为false则可以进入临界区，此时lock通过该原子函数已经变为true，其它程序无法进入临界区。

利用，该原子函数实现有限等待：



**#信号量Semaphore**

信号量是同步工具，它只能通过两个**原子操作**wait和signal来访问（原来被称为P和V），



当每个进程需要使用资源时，需要对信号量执行wait操作减少信号量计数，当进程释放资源时需要执行signal操作增加信号量计数。

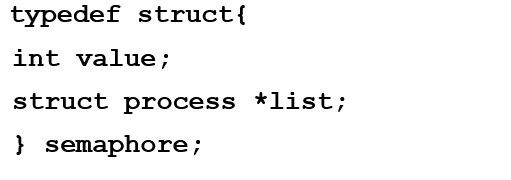
**\*实现**

之前设计的信号量有明显的缺陷，就是多道进程在while循环等待资源，这就影响了系统性能，我们把这称为**忙等待**。

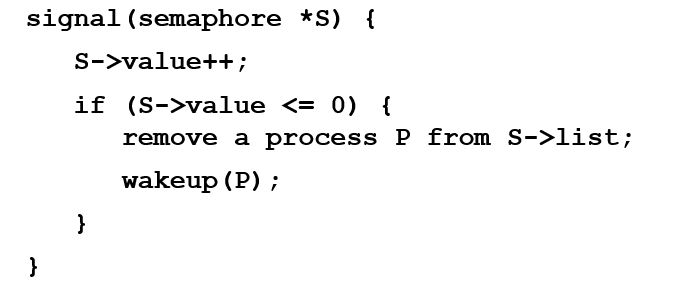
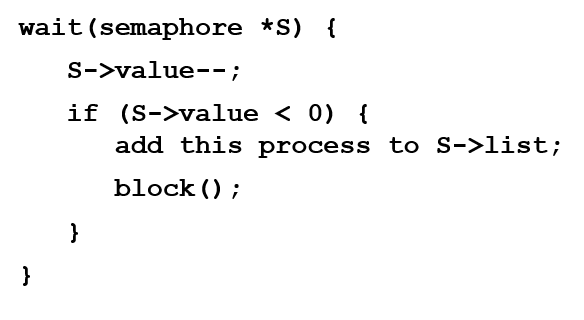
这种信号量也称为**自旋锁**。

为了克服忙等待，我们可以让其不是等待而是阻塞，阻塞进程被放入到与信号量相关的等待队列中，并将该进程的状态切换为等待状态。

我们可以这样定义信号量：



对应，P和V操作将改为：



\*死锁与饥饿

两个或多个进程无限地等待一个事件，而该事件只能由其中之一的进程来产生。

**#经典同步问题**

**\*有限缓冲问题**

需要实现的目标：

1. 访问互斥

2. 空时不能消费

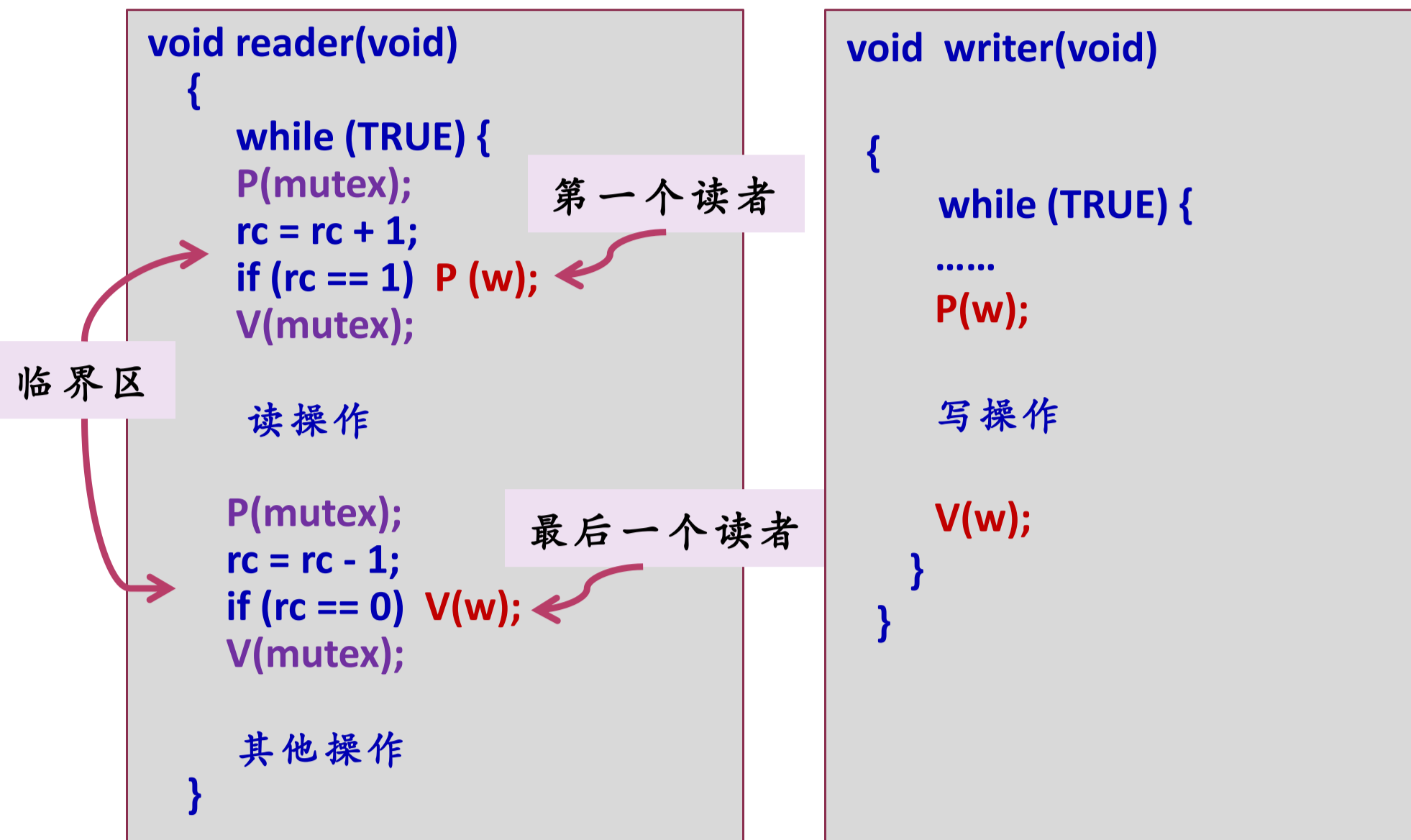
3. 满时不能生产

**\*读者写者问题**

需要实现的目标：

1. 读者可以共同访问数据库，当没有写者工作的时候。

2. 写者工作时没有其他写者且读者不能读。

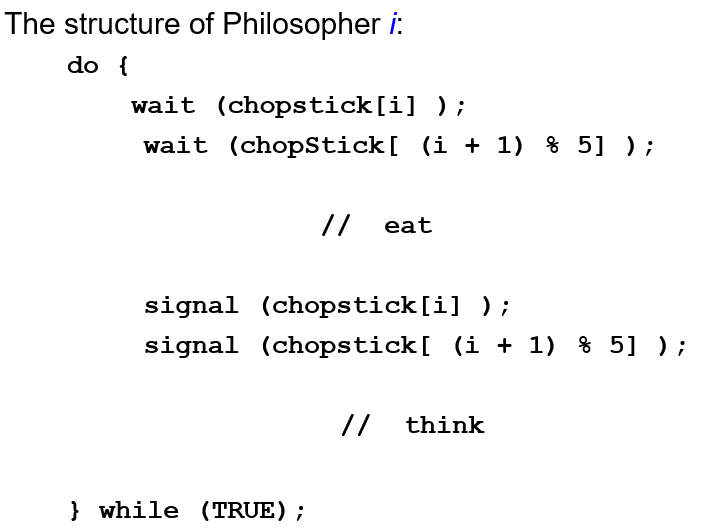


这里面的mutex用于保持在rc增减上的互斥。

**\*哲学家进餐问题**



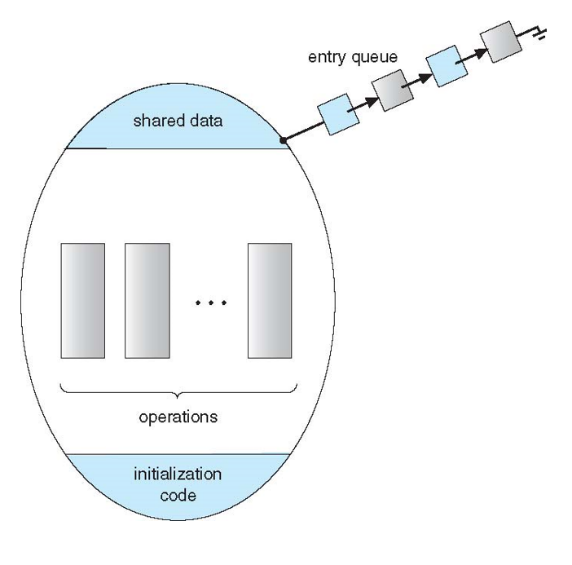
当哲学家思考时，他们不交互，当其中一个饥饿时，会拿起离他最近的两只筷子，一次只能拿一只，且不能从别人手上拿。



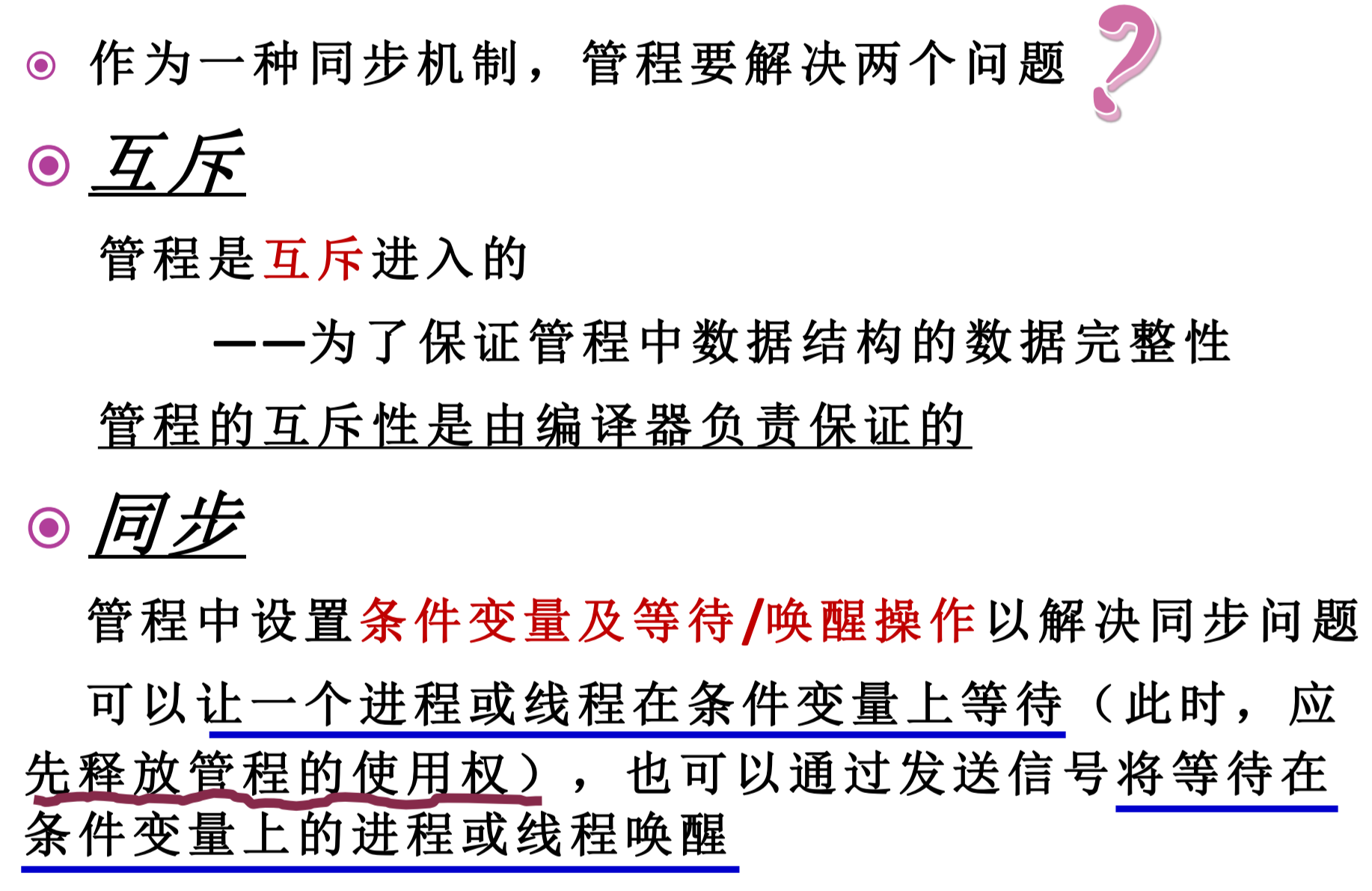
这种解法可以保证互斥，但会发生死锁！如果哲学家同时拿右手的筷子，就会永远等待。

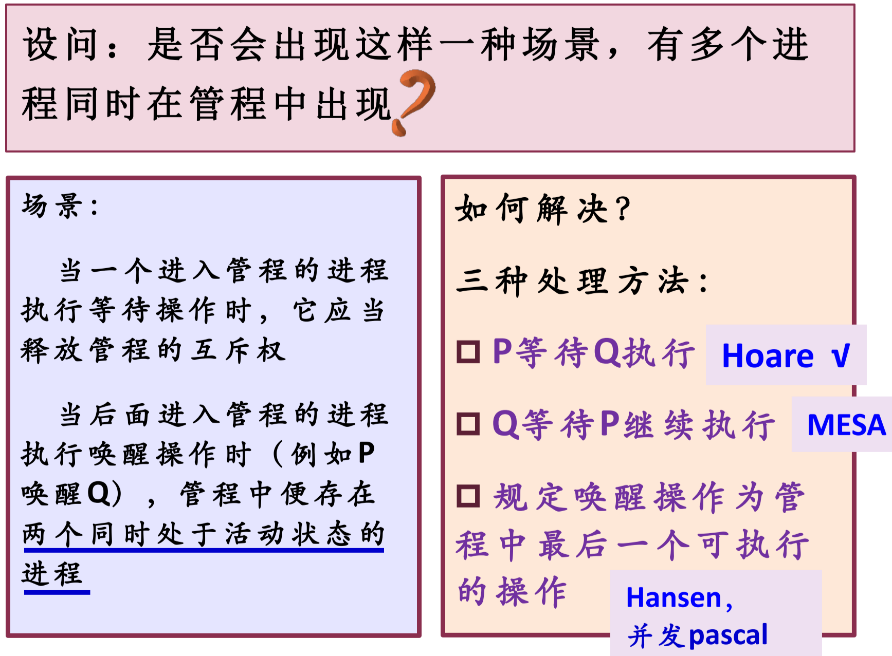
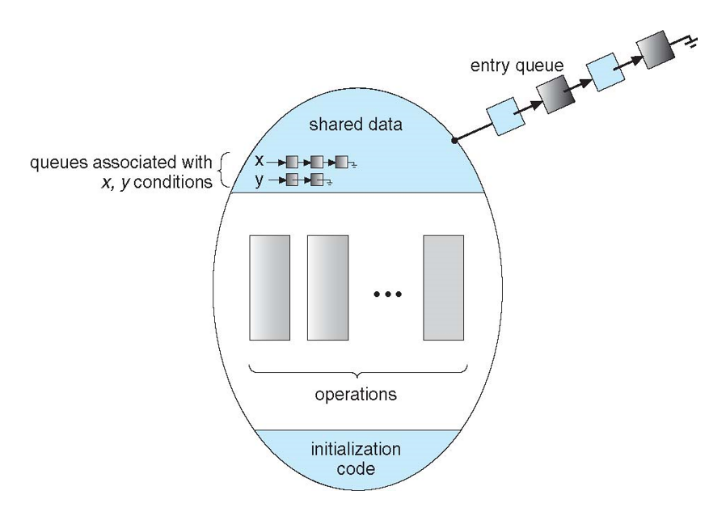
解决办法是：只允许四个哲学家同时坐在座位上，即筷子多于哲学家数或不同时申请；只用两只筷子同时可用时才可以拿起；使用非对称解决办法。

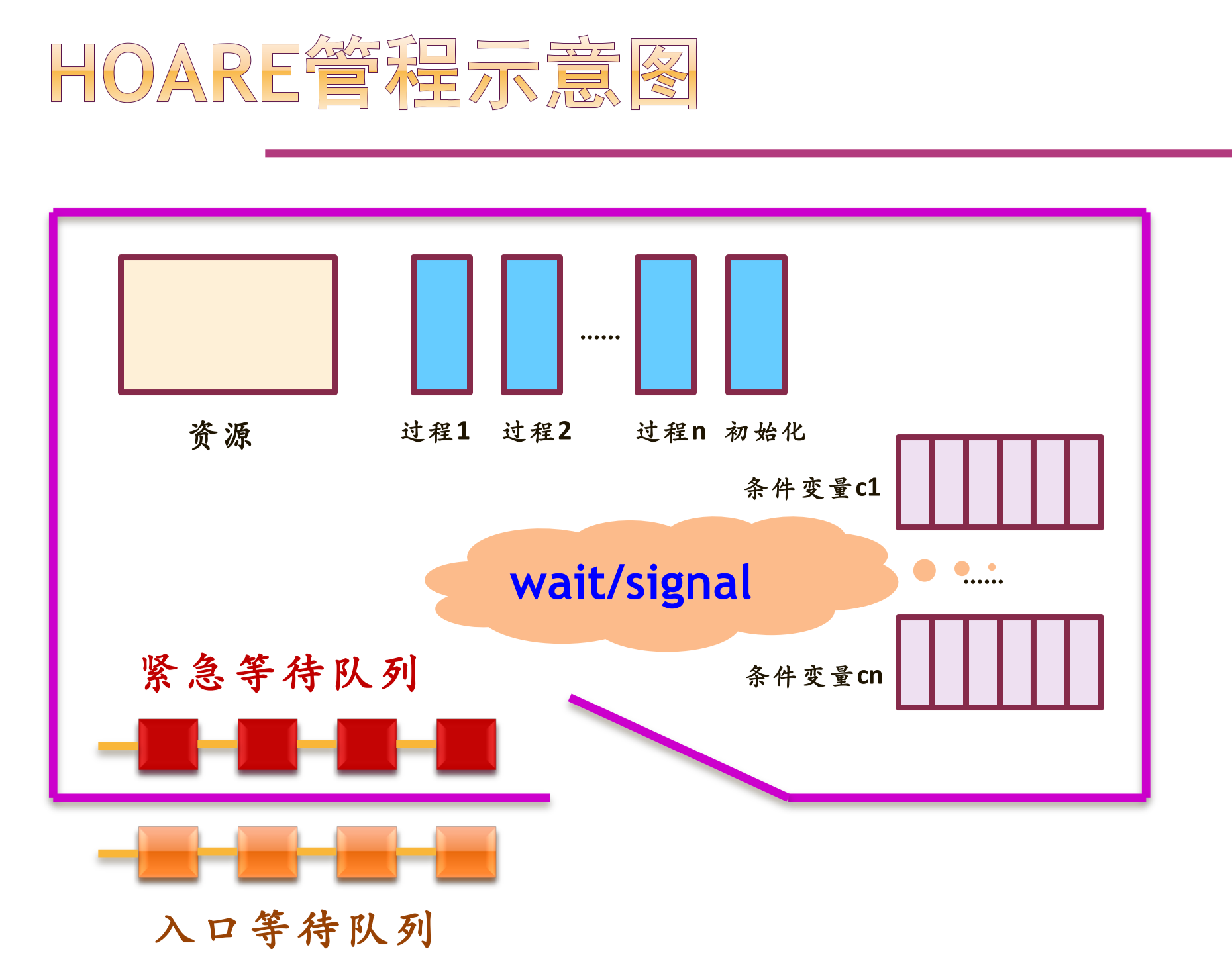
**#管程（Monitors）**

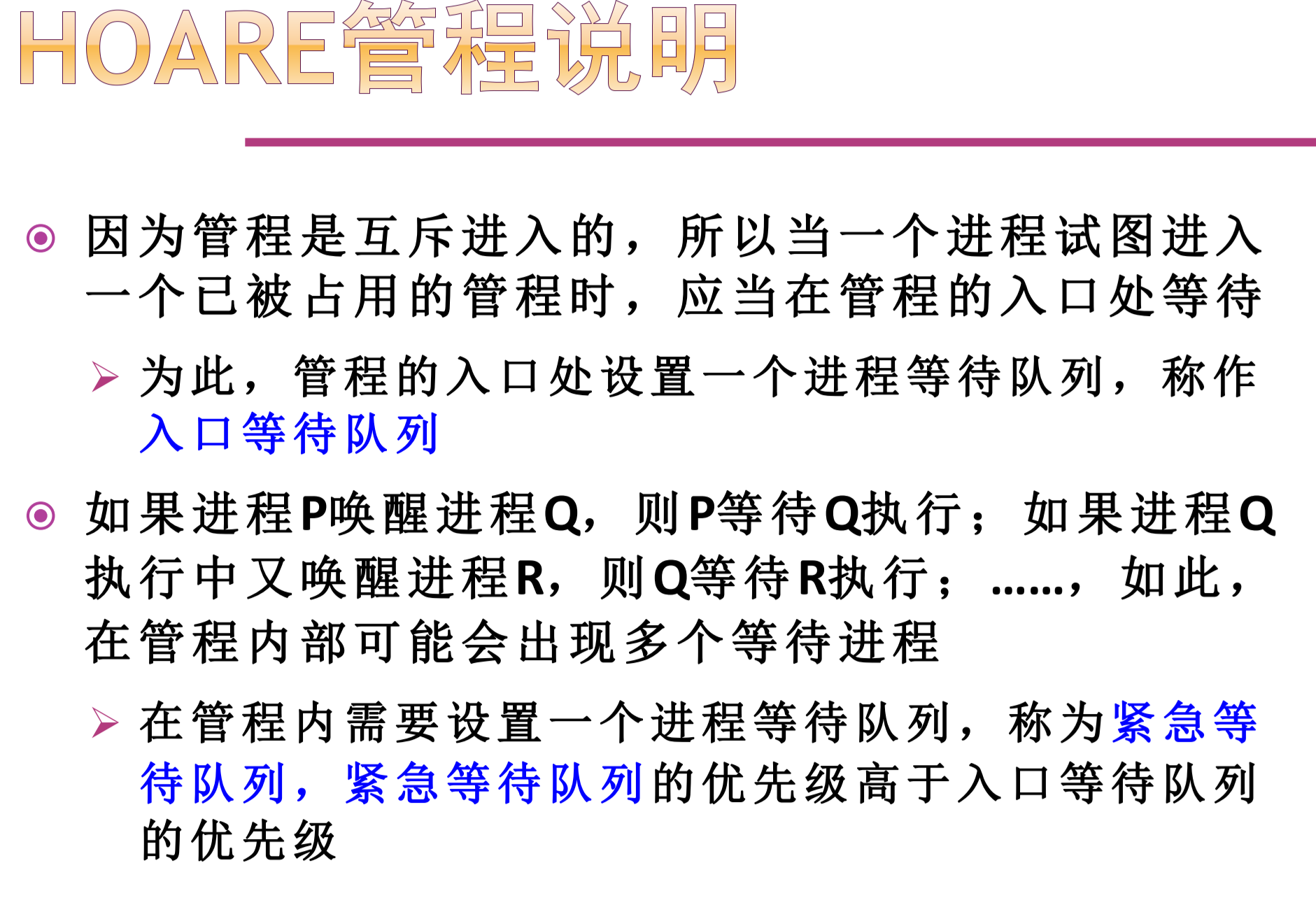
为什么会出现管程：信号量机制的不足：程序编写困难、易出错。

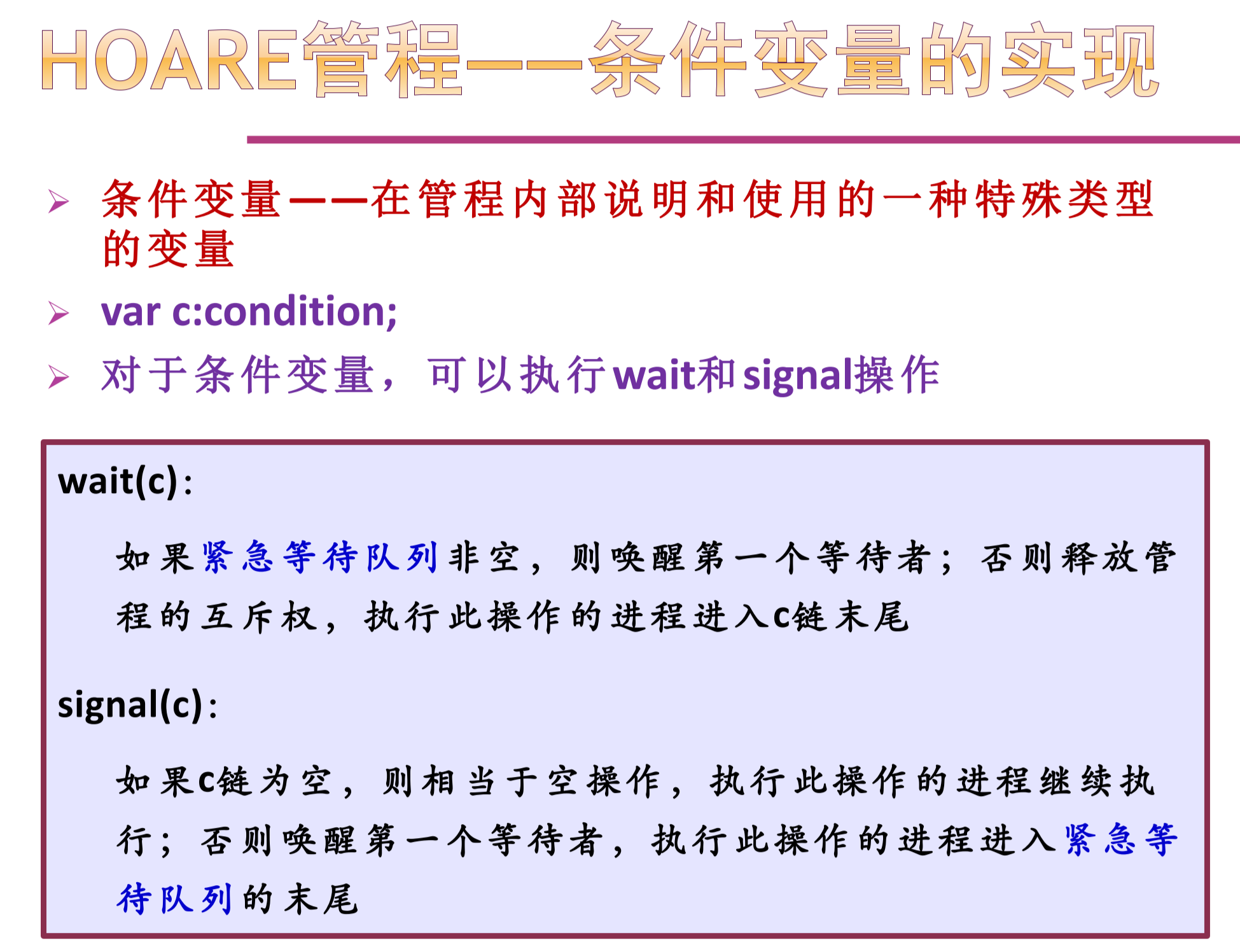
**进程只能通过调用管程中的过程来间接地访问管程中的数据结构**。











**\*管程对的应用**

