操作系统

Operating system

孔维强

大连理工大学

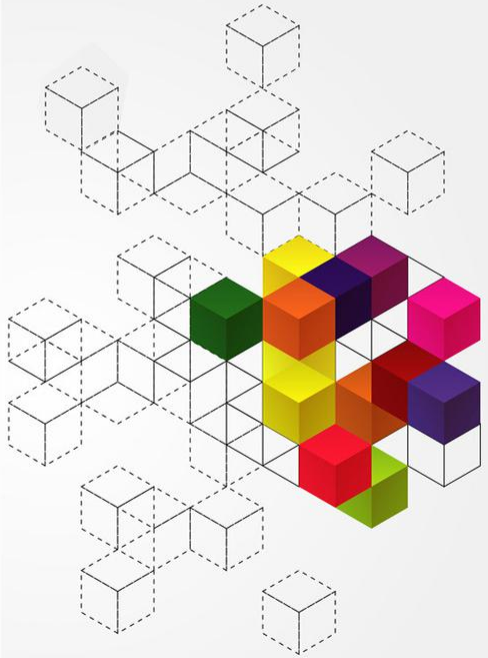
内容纲要

6.9 管程

一、 管程概念

二、 管程的三种语义

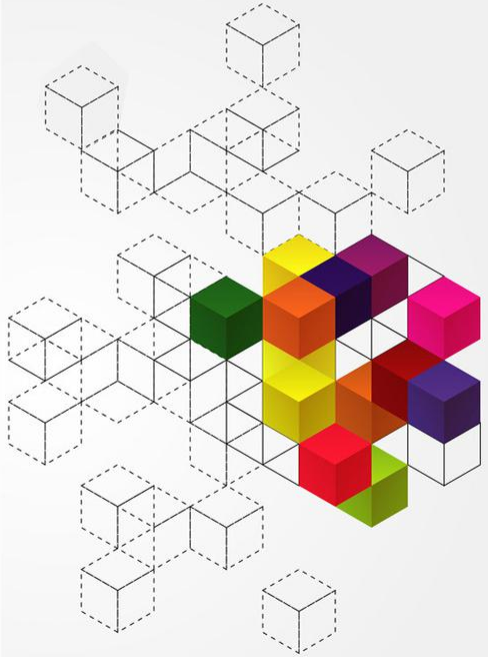
三、 管程应用示例



一、管程概念

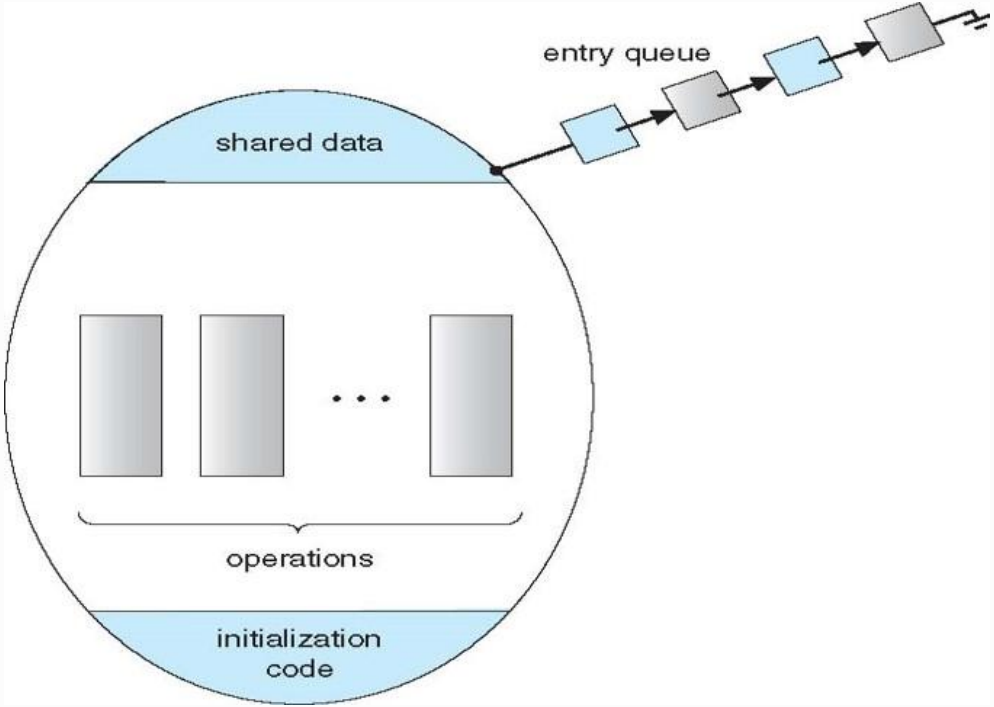
管程的基本思想

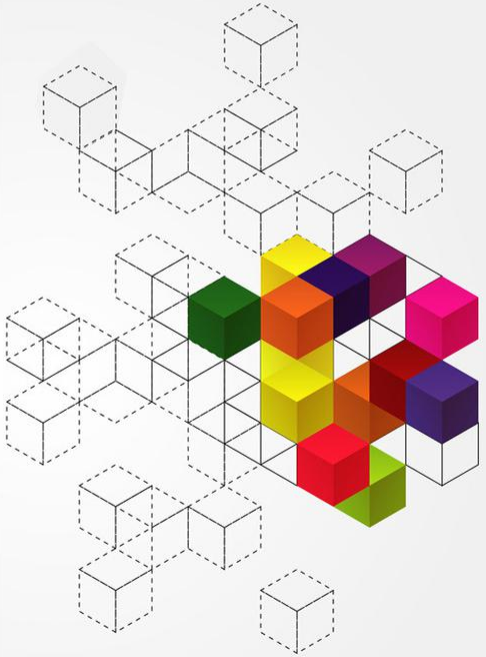
- 把分散在各进程中的临界区集中起来进行管理，将系统中的临界资源用数据结构表示
- 建立一个“秘书”程序来管理对临界资源的访问。“秘书”每次仅允许一个进程来访，如此，既便于对临界资源的管理，又实现对资源的互斥访问。“秘书”就是后来的管程。
- 利用管程，对资源的管理可借助数据结构及在其上实施的若干过程来进行；对共享资源的申请和释放，通过过程在数据结构上的操作来实现
- 管程被请求和释放资源的进程所调用



一、管程概念

- 管程构成示意图

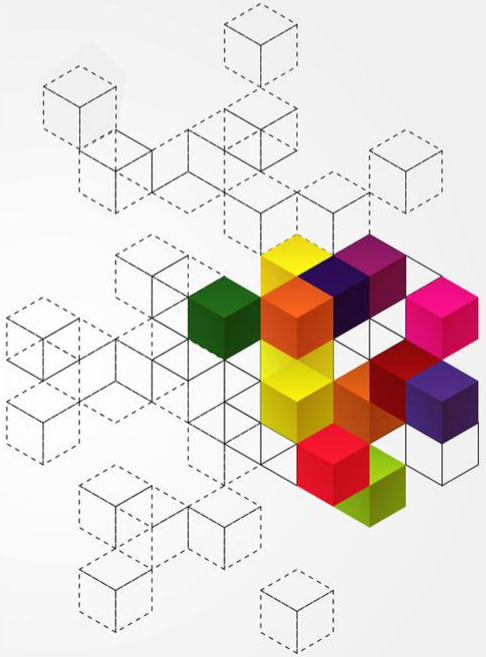




一、管程概念

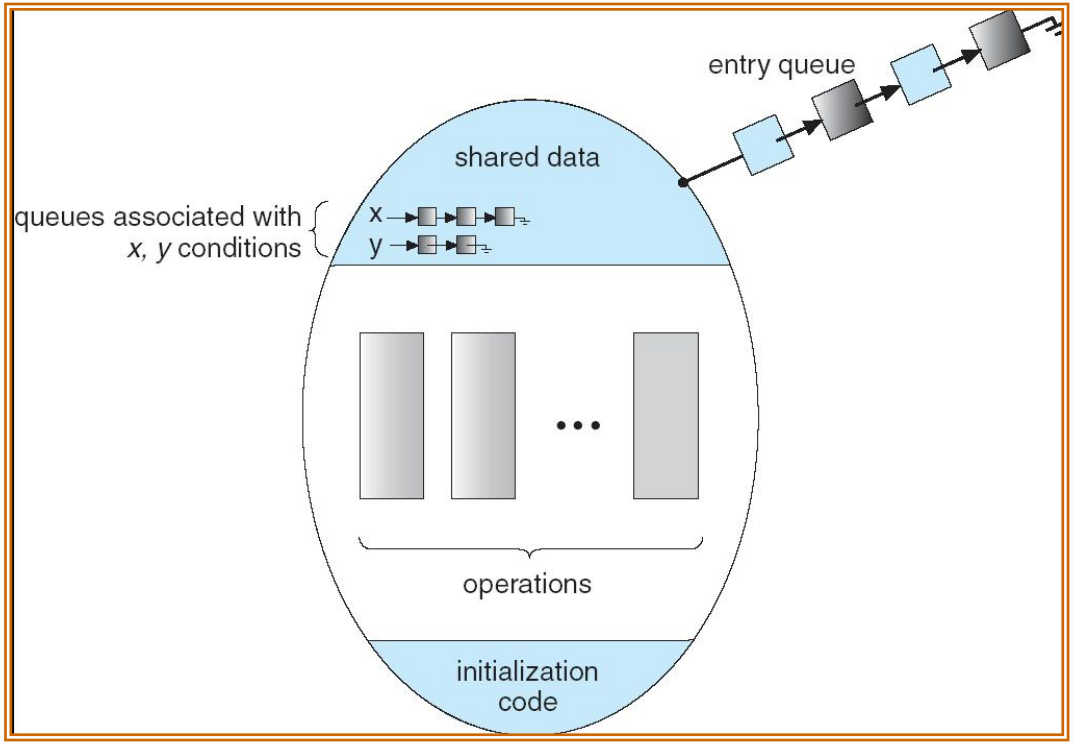
- 条件变量 condition x, y
- 条件变量上允许的2个操作
  - ✓x.wait() – 执行该操作的进程被挂起，直到x.signal()被执行
  - ✓x.signal() – 释放被x.wait()挂起的进程（如有）

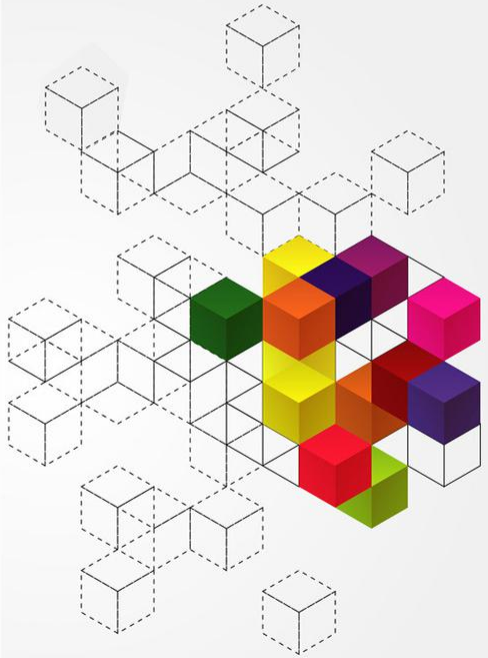
    如无进程被x.wait()挂起，则该操作无任何影响，与signal()信号量不同



一、管程概念

- 管程构成示意图





一、管程概念

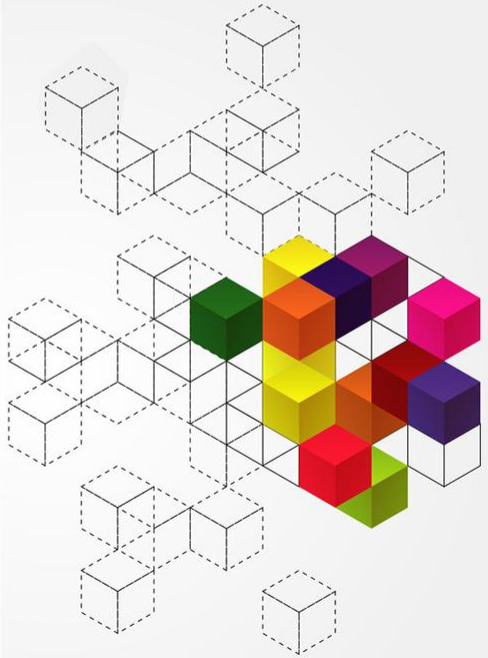
- 如果进程P执行x.signal()，且进程Q被x.wait()挂起，下一步会怎样？

  管程内仅应有一个进程可以处于active状态

- 选项：

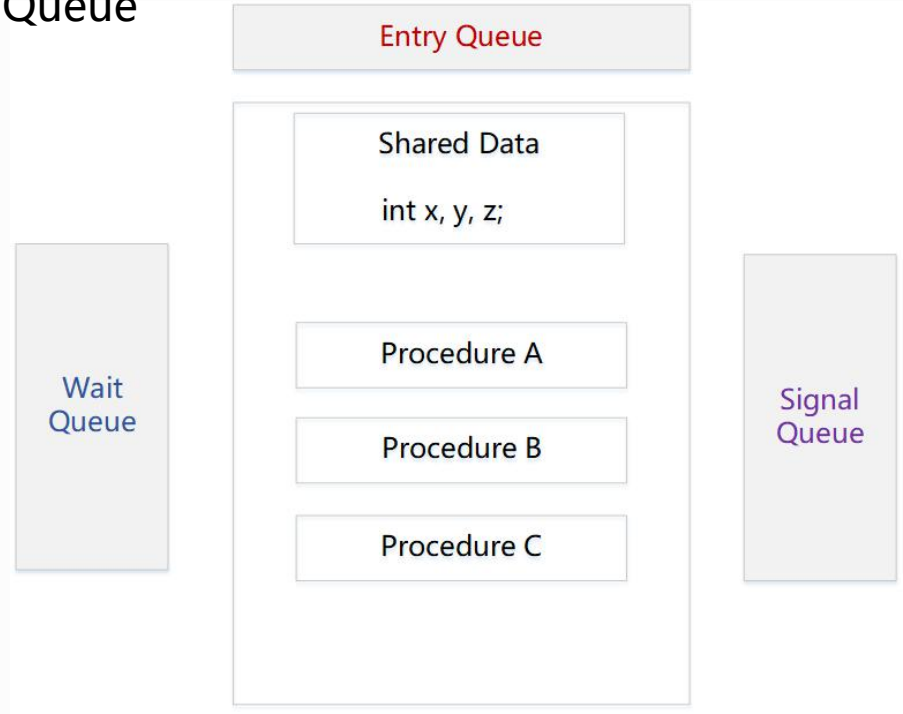
  signal and wait: P等待直到Q离开管程

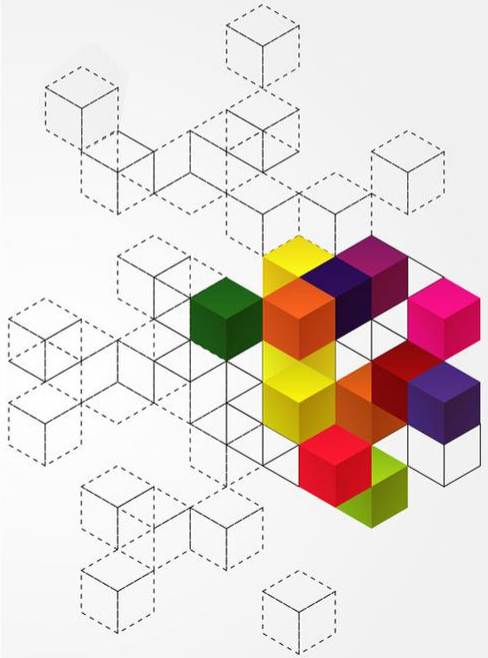
  signal and continue: Q等待直到P离开管程



一、管程概念

- 管程实现中两个关键队列
  - Wait Queue
  - Signal Queue

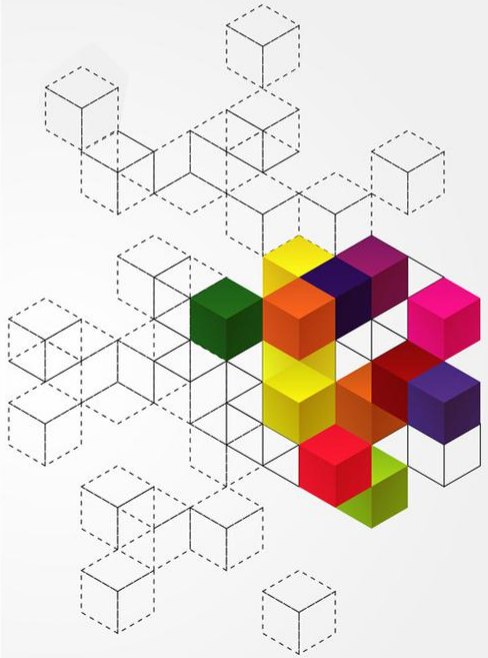




二、管程的三种语义

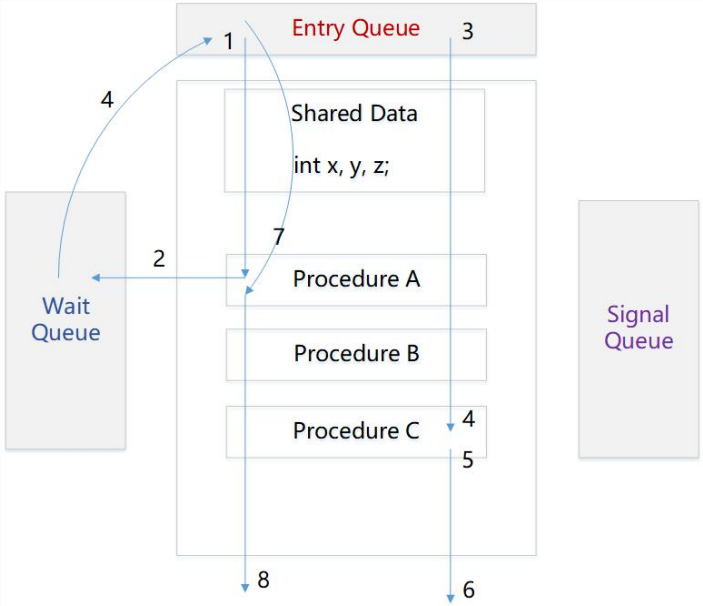
管程的三种不同实现语义：

Mesa语义
Hoare语义
Brinch Hanson语义

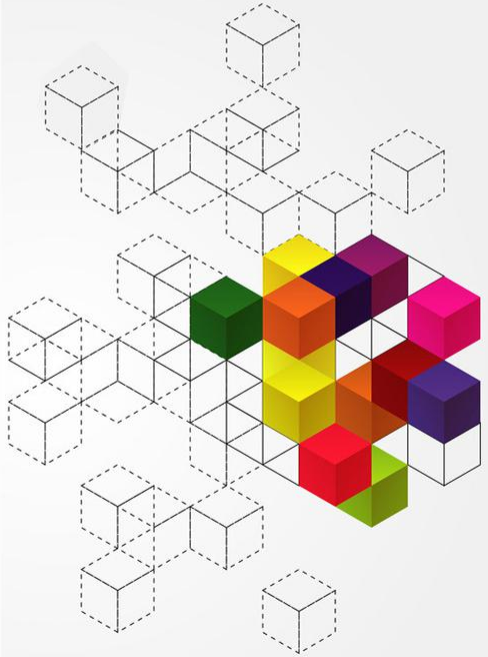


二、管程的三种语义

Mesa Semantics：

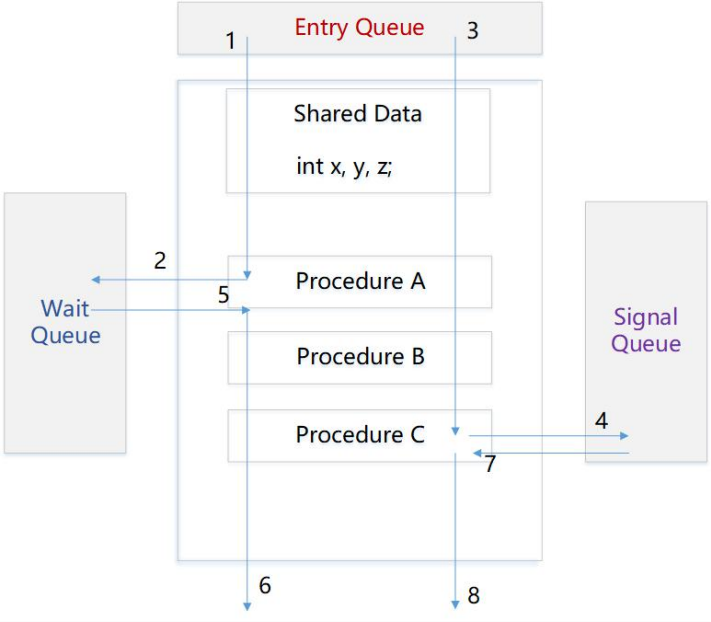


1. 线程A进入管程
2. 线程A等待某个资源
3. 线程B进入管程
4. 线程B释放线程A等待的资源，线程A被转到Entry Queue，而线程B继续执行
5. 线程B继续在管程内执行
6. 线程B离开管程
7. 线程A重新进入管程
8. 线程A离开管程
9. 其他线程可以继续进入管程

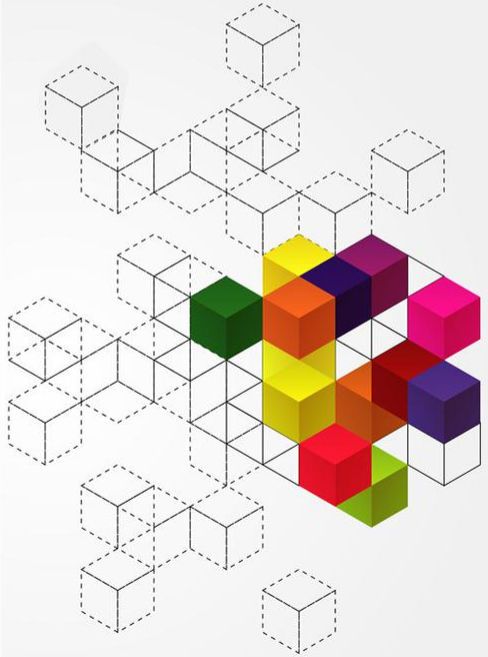


二、管程的三种语义

Hoare Semantics：

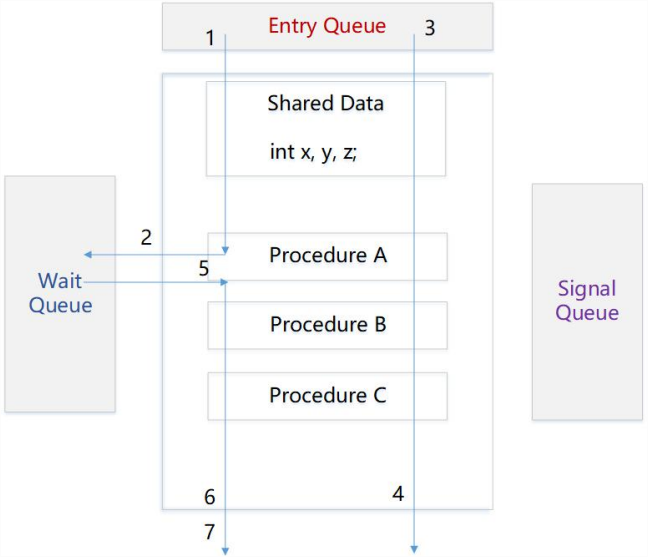


1. 线程A进入管程
2. 线程A等待某个资源
3. 线程B进入管程
4. 线程B释放线程A等待的资源，唤醒线程A，而线程B进入Signal Queue
5. 线程A重新进入管程继续执行
6. 线程A离开管程
7. 线程B重新进入管程
8. 线程B离开管程
9. 其他线程可以继续进入管程

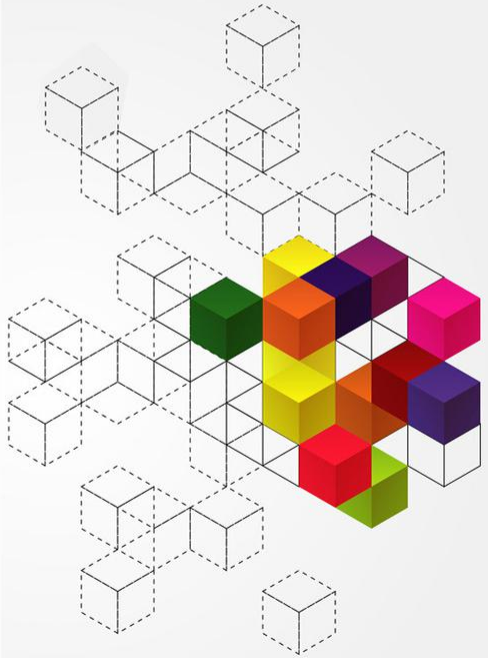


二、管程的三种语义

Brinch Hanson Semantics：



1. 线程A进入管程
2. 线程A等待某个资源
3. 线程B进入管程
4. 线程B发资源已释放信号给线程A，随后线程B离开管程
5. 线程A重新进入管程继续执行
6. 线程A离开管程
7. 其他线程可以继续进入管程



三、管程应用示例

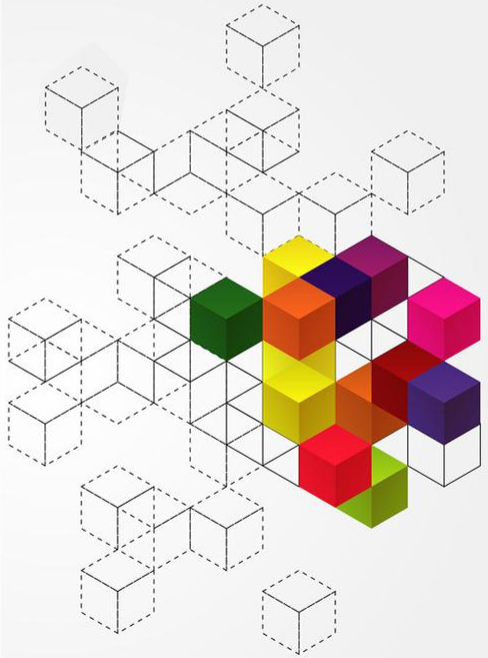
```
monitor PC {
    condition : full, empty;
    int : count = 0;

    entry put {
        if (count==max) wait (full);
        insert item
        count = count+1;
        if (count==1) signal (empty);
    }


    entry get {
        if (count==0) wait (empty);
        remove item
        count = count-1;
        if (count==max-1) signal (full);
    }

}
```

producer process

```
while (TRUE) {
    produce item
    PC.put;

}
```

consumer process

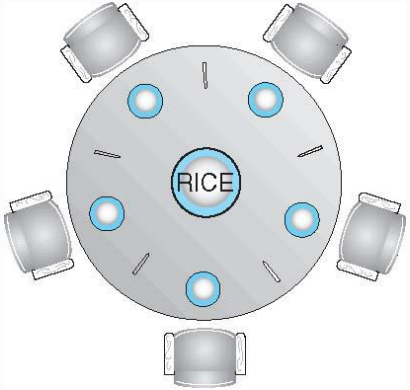
```
while (true) {
    PC.get;
    consume item
}
```



三、管程应用示例

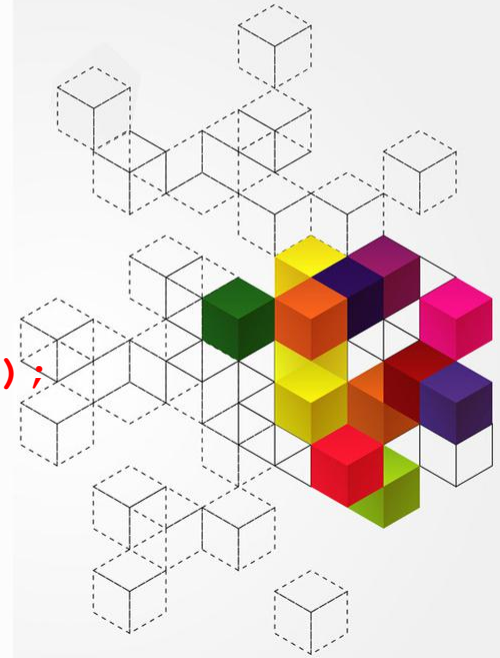
就餐的哲学家问题：

- The structure of Philosopher *i*:



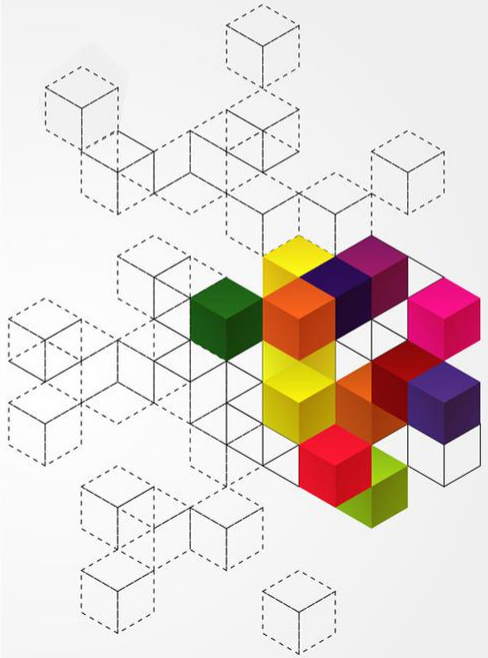

```
do {
        wait (chopstick[i] );
        wait (chopstick[ (i + 1) % 5] );
              //  eat
        signal (chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );
              //  think
} while (TRUE);
```

- 该算法有什么问题？



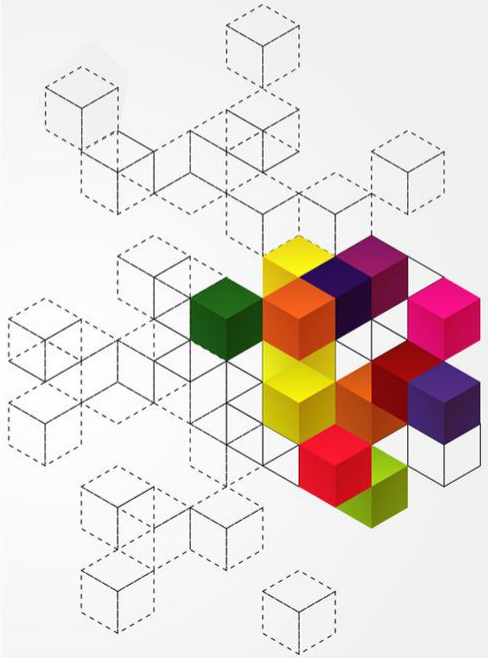
三、管程应用示例

```
monitor DiningPhilosophers
{
  enum { THINKING, HUNGRY, EATING } state[5] ;
  condition self[5];

  void pickup (int i) {
          state[i] = HUNGRY;
          test(i);
          if (state[i] != EATING)    self[i].wait();
    }

    void putdown (int i) {
          state[i] = THINKING;
                      // test left and right neighbors
           test((i + 4) % 5);
           test((i + 1) % 5);
    }
```



三、管程应用示例

```
  void test(int i) {
          if ((state[(i + 4) % 5] != EATING) &&
          (state[i] == HUNGRY) &&
          (state[(i + 1) % 5] != EATING) ) {
                state[i] = EATING ;
               self[i].signal () ;
          }
    }

    initialization_code() {
          for (int i = 0; i < 5; i++)
          state[i] = THINKING;
  }
}
```

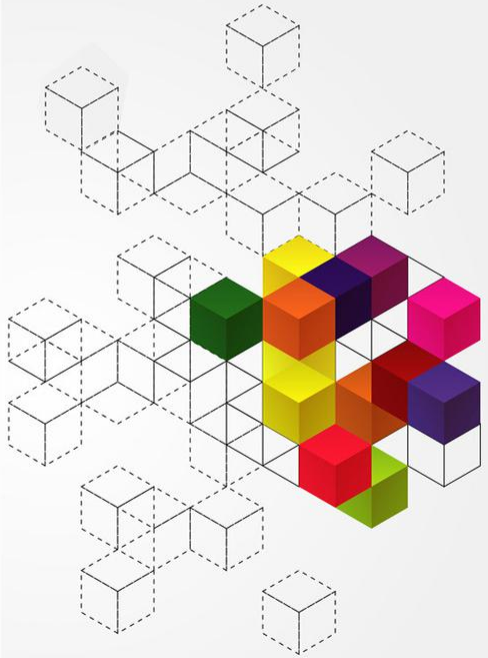


三、管程应用示例

- 每个哲学家 *i* 执行 **pickup()** 和 **putdown()** 操作:

```
DiningPhilosophers.pickup(i);

        EAT

DiningPhilosophers.putdown(i);
```

- 无死锁，但可能有饥饿



本讲小结

- 管程概念
- 管程的实现语义
- 管程应用示例