操作系统

Operating system

胡燕

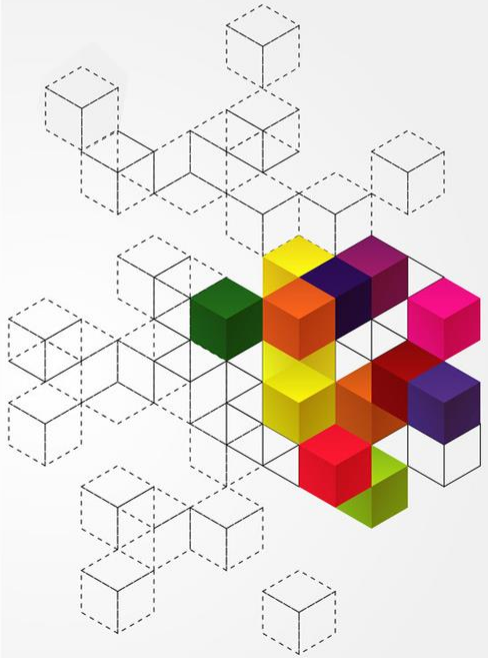
大连理工大学

第6章 进程同步

- **Process Synchronization**

分为三大部分：

- 上：进程同步概念、临界区（互斥）
- 中：信号量
- 下：经典同步问题、管程



内容纲要

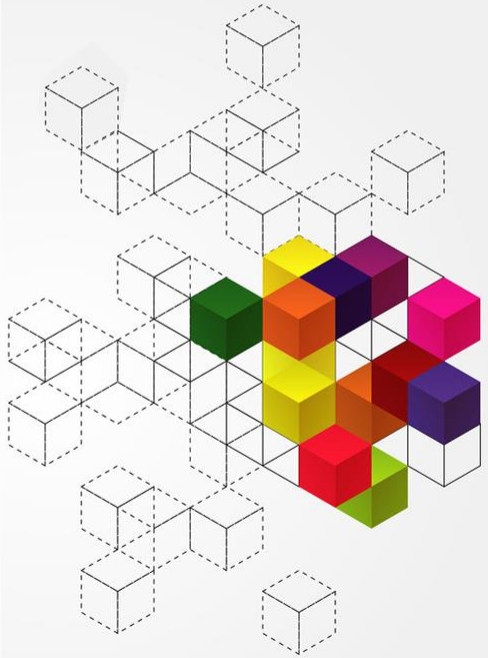
6.5 信号量概念

一、引入信号量的动机

二、信号量概念

三、信号量实现

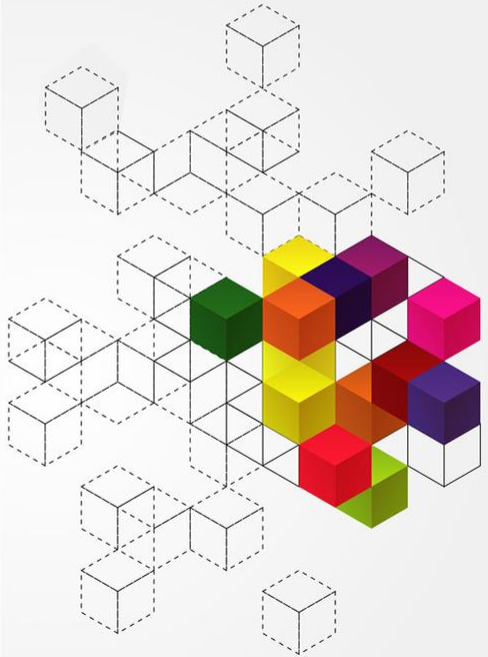
四、信号量编程接口示例



一、引入信号量的动机

为什么需要信号量：

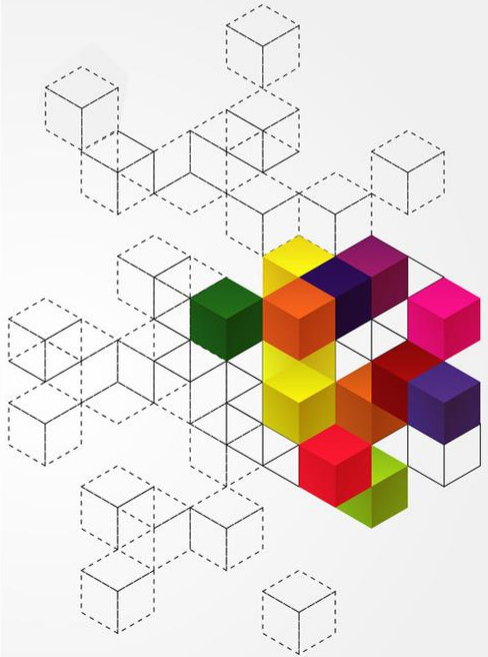
- 面包店算法，是实现互斥的一般性软件解法
- 当多个进程竞争使用的某类资源具有多个资源实例时，互斥性如何保证
- 例如：卡拉OK房间内的麦克风（2个），唱歌时竞争使用资源的状态就会有：
- 1个人独唱
- 2个人合唱，其余人休息
- 2个人合唱，1个或多个人等待唱歌



一、引入信号量的动机

分析一下一般情况下的多资源实例竞争问题

- 初始状态：剩余资源数available = n
- 中间状态：陆续有k(<n)个进程，每个进程取走一个资源，那么available=n-k
- 临界状态：某个时刻，最后一个资源可能被取走，那么available=0
- 有q个进程在等待资源的状态

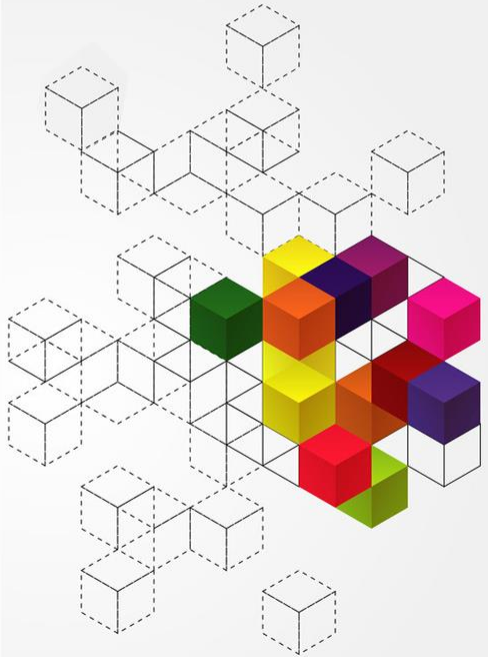


二、信号量概念

信号量（Semaphore）

- 一个信号量S是一个整型量，除对其初始化外，它只能由两个原子操作P和V来访问

- 1965年，荷兰科学家Dijkstra提出
- P和V的名称来源于荷兰文proberen(测试)和verhogen(增量)
- 亦有将P/V操作分别称作wait(), signal()



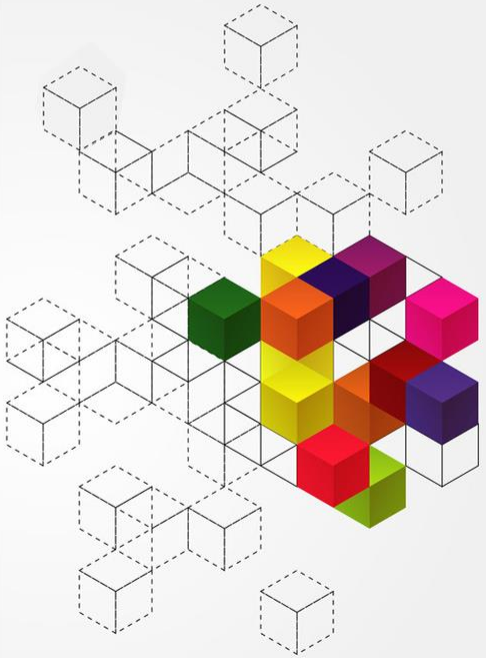
二、信号量概念

信号量（Semaphore）

- 信号量的P，V操作的简单伪代码
- 表达了P,V操作的基本语义，但是并不是实际的实现方式

```
Wait(){
        while(S<=0) ;
        S--;
}
```

```
Signal(){
    S++;
}
```



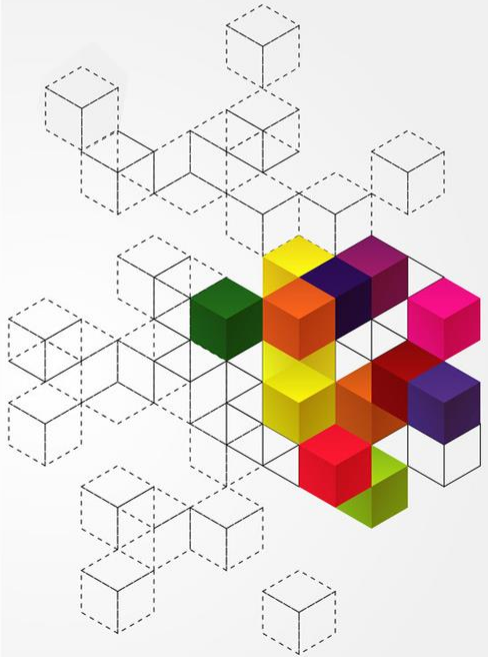
二、信号量概念

信号量（Semaphore）

```
Wait(){
    while(S<=0) ;
    S--;
}
```

```
Signal(){
    S++;
}
```

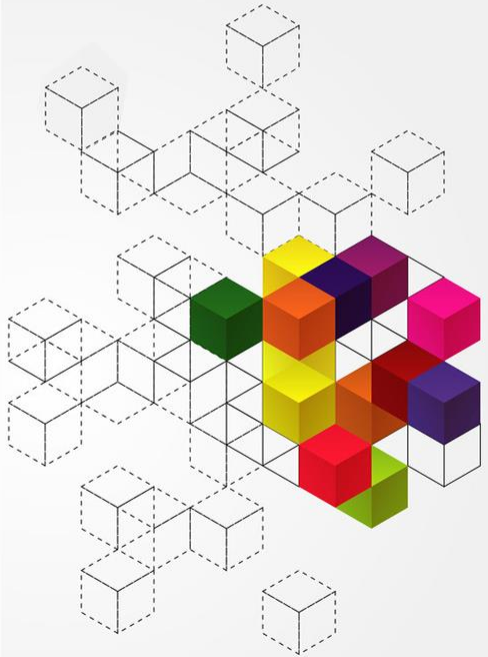
- 上述简单伪代码中的不合实际的问题：
- (1)不能忙等
- (2)没有办法记录处于等待状态的进程数量



三、信号量实现

避免进程忙等，wait和signal的定义需要进行修改

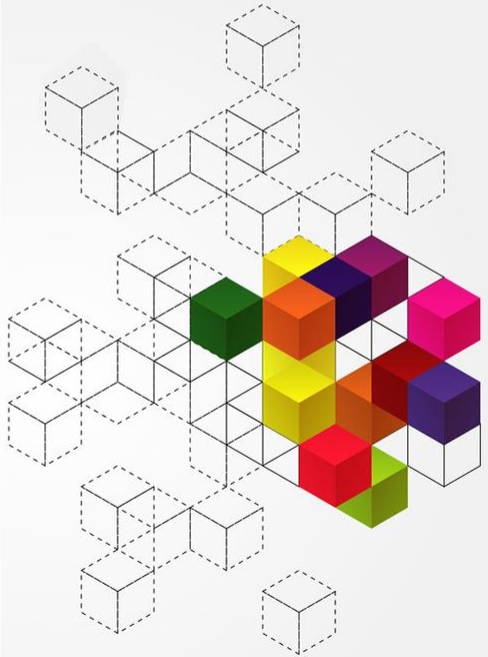
- Wait:
- 当一个进程执行wait操作但发现信号量S<=0时，它必须等待，这里的等待不是忙等，而是阻塞自己
- 阻塞操作将一个进程放入与信号量相关的等待队列中，且该进程的状态被切换成等待状态，接着控制被转到CPU调度程序，以选择另一个进程来执行



三、信号量实现

避免进程忙等，wait和signal的定义需要进行修改

- Signal:
- 一个进程阻塞且等待信号量S，可以在其他进程执行signal操作之后被重新执行



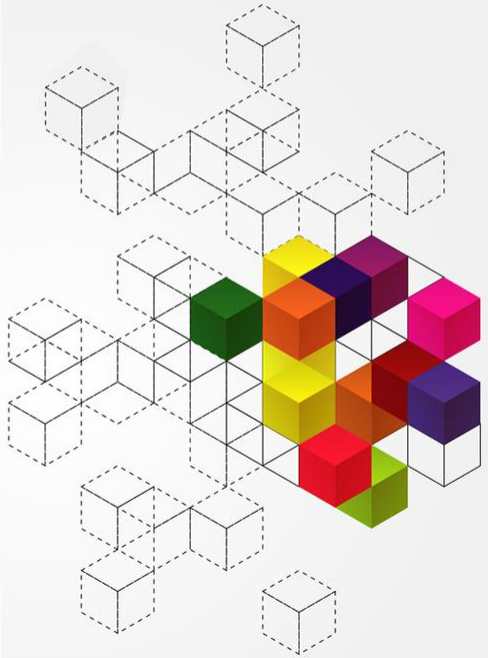
三、信号量实现

信号量数据结构

- 为了定义基于阻塞（block）/唤醒（wakeup）的信号量，可以将信号量定义为如下一个“C”结构

```
typedef  struct {
    int value;
    struct process *L;
}semaphore;
```

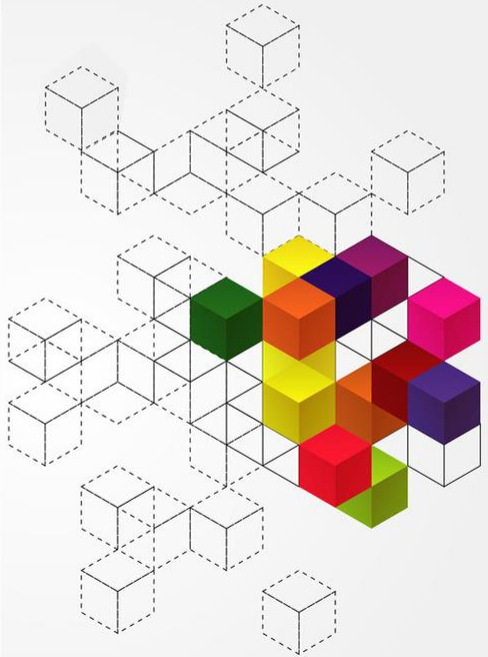
在该信号量上阻塞的进程队列



三、信号量实现

- 合理的P操作实现

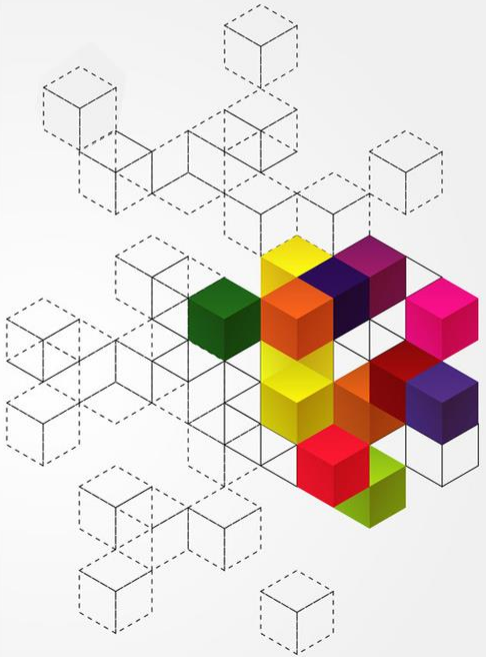
```
void  wait(semaphore S){
    S.value--;
    if(S.value<0){
        add this process to S.L;
        block();
    }
}
```



三、信号量实现

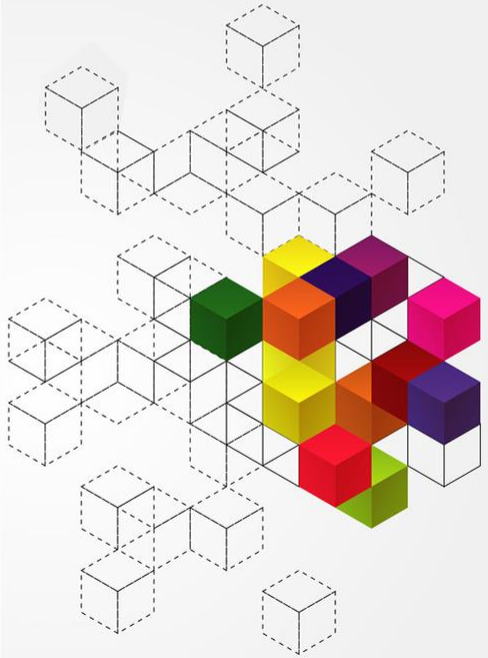
- 合理的V操作实现

```
void  signal(semaphore S){
    S.value++;
    if(S.value<=0){
        remove a process P from S.L;
        wakeup(P);
    }
}
```



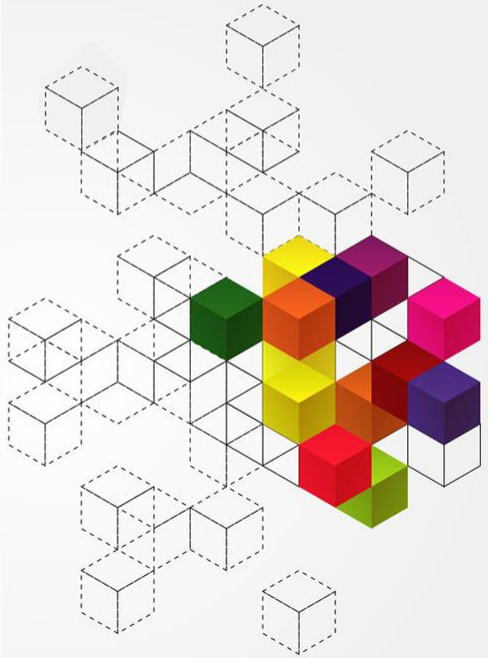
四、信号量编程接口示例

```
// C program to demonstrate working of Semaphores
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t mutex;
```



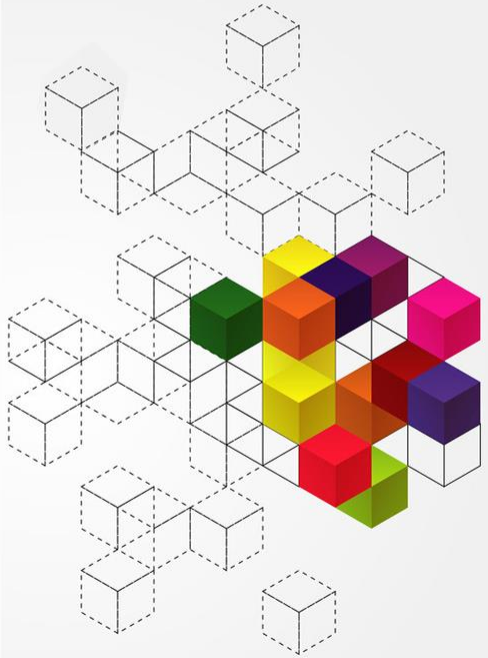
四、信号量编程接口示例

```
void* thread(void* arg)
{
    //wait
    sem_wait(&mutex);
    printf("\nEntered..\n");

    //critical section
    sleep(4);

    //signal
    printf("\nJust Exiting...\n");
    sem_post(&mutex);
}
```



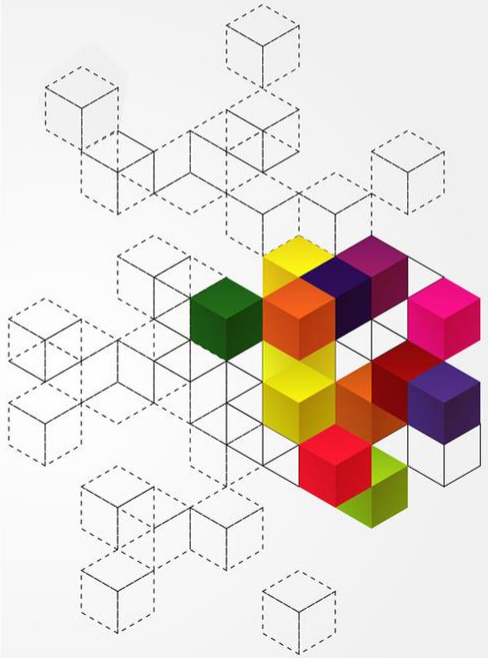
四、信号量编程接口示例

```
int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}
```



本讲小结

- 信号量的引入
- 信号量概念
- 信号量实现
- 信号量编程接口示例



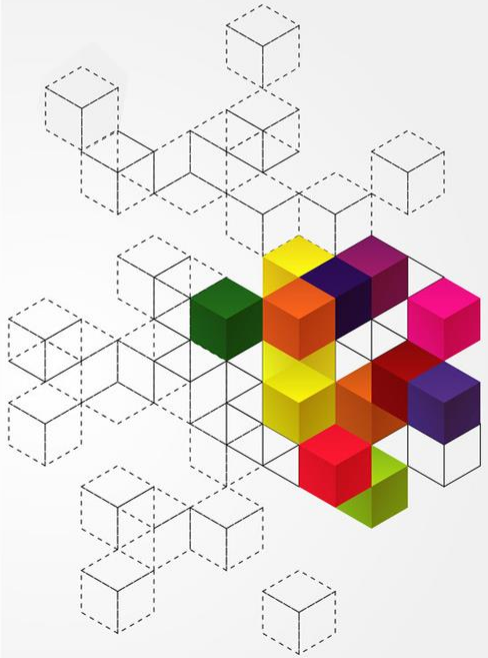
内容纲要

6.6 信号量用法

一、处理进程间接协作

二、处理进程直接协作

三、信号量值区间分析

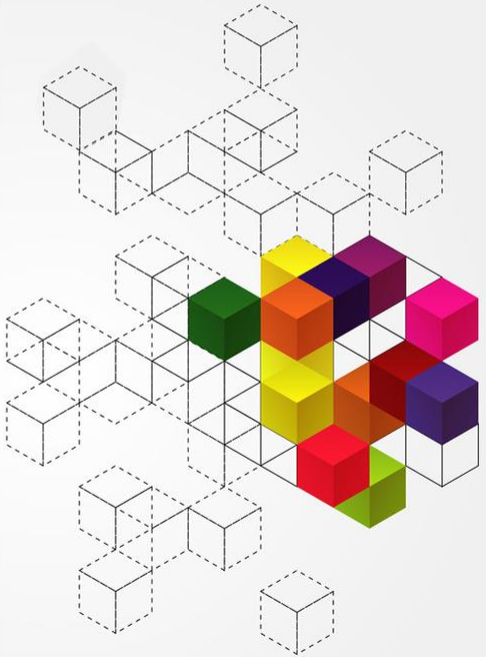


一、处理进程间接协作

1.用在解决临界区问题：

```
do{
    wait(mutex);
    临界区;
    signal(mutex);
    退出区;
}while(1);
```

使用信号量的互斥实现

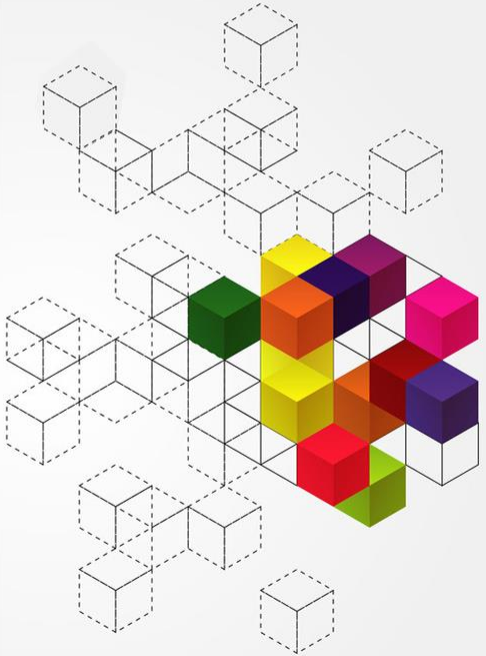


一、处理进程间接协作

用在解决临界区问题：

- 解决多资源竞争

- 拥有固定数量座位的自习室自习问题
- 进入自习室，登记
- 退出时，撤销登记
- 资源：自习室座位



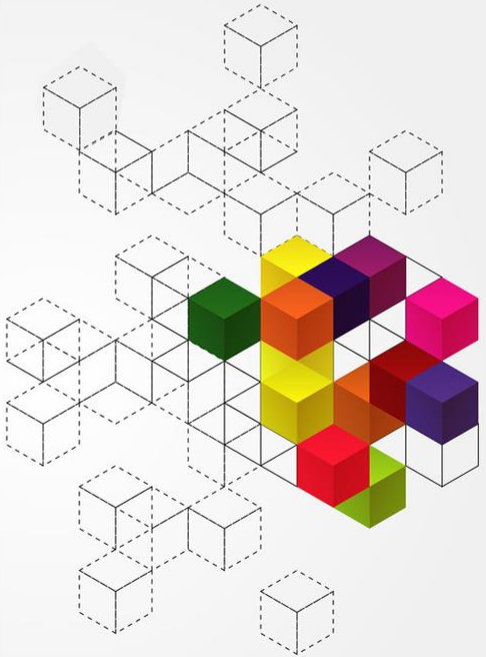
二、处理进程直接协作

2.用在解决进程直接同步问题：

比如，我们做一个软件project，分成几个部分 $m_1,m_2,\ldots,m_k$

- 必须做完mi之后，才能开始做m_{i+1}

使用信号量如何来控制这个效果呢



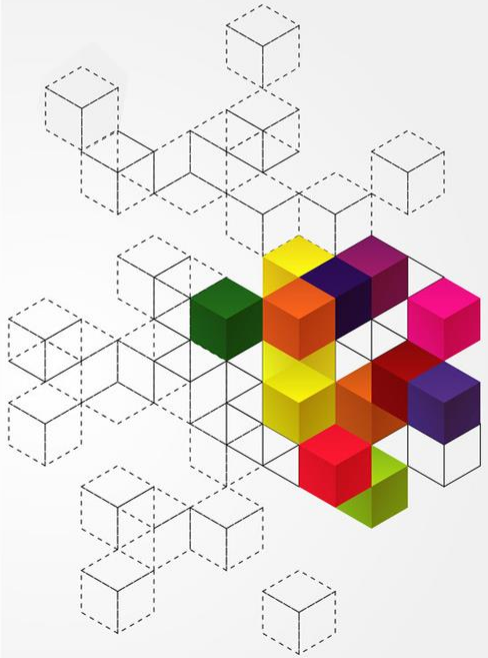
二、处理进程直接协作

2.用在解决进程直接同步问题：

```
do{
    实施mi
    signal(semi);
}while(1);
```

```
do{
    wait(semi);
    ...
}while(1);
```

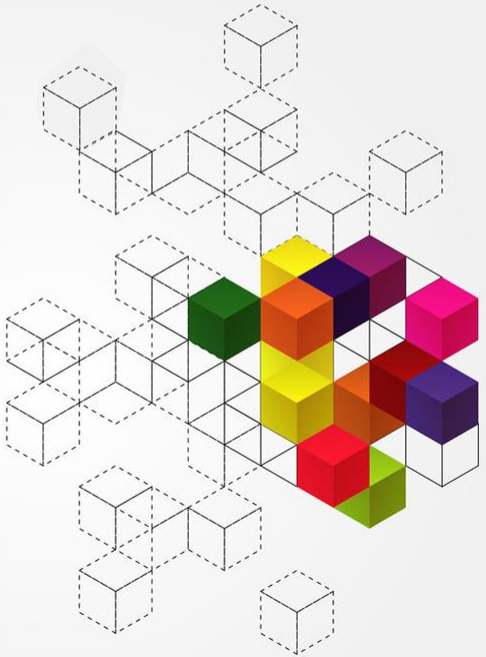
思考:semi的初值设为多少?



二、处理进程直接协作

2.用在解决进程直接同步问题：

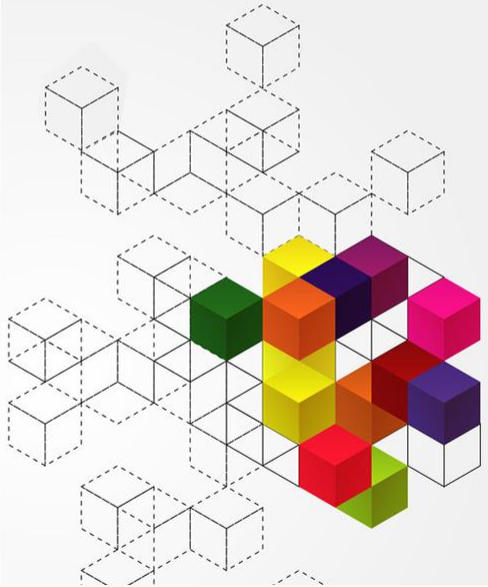
- 实际例子：
- 公交司机，售票员 协作问题



主观题 10分

设置

有一个阅览室，读者进入时必须先在一张登记表上进行登记，该表为每一个座位列出一个表目，包括座位号、姓名，读者离开时撤销登记信息。阅览室有100个座位。试利用信号量来处理这些读者进程之间的协作关系。



正常使用主观题需2.0以上版本雨课堂

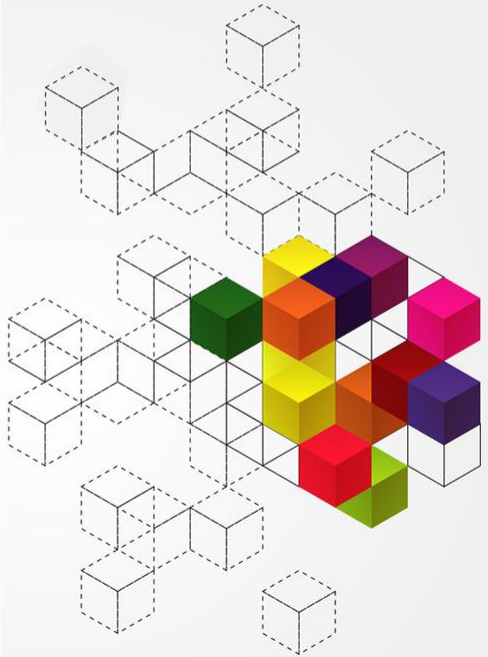
作答

table_mutex=1, seat_mutex=100

P(seat_mutex)
P(table_mutex)
登记
V(table_mutex)

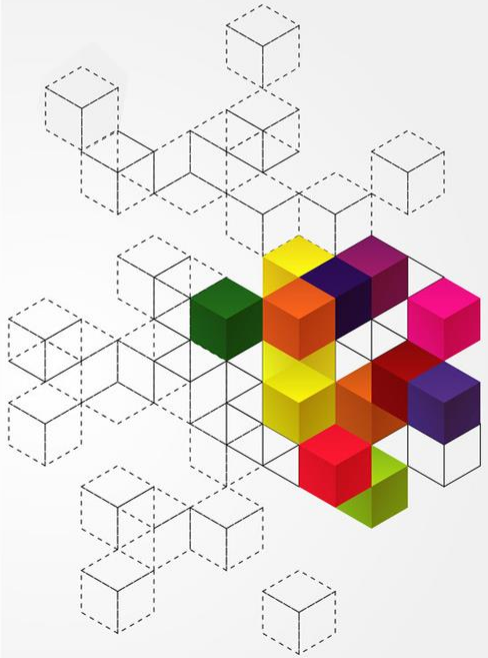
Take a seat and read

P(table_mutex)
注销登记
V(table_mutex)
V(seat_mutex)



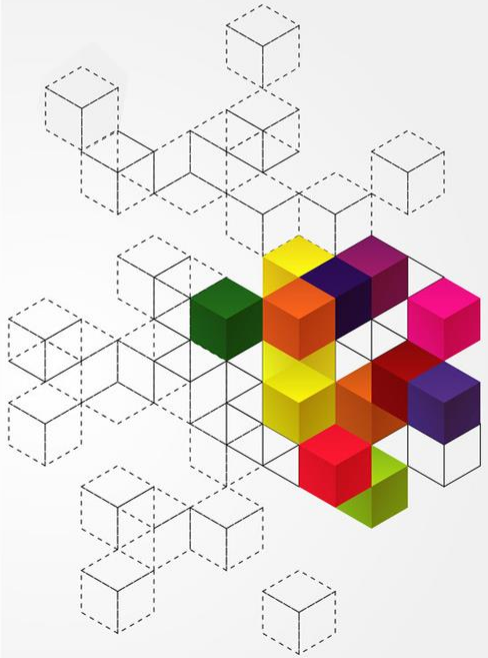
三、信号量值区间分析

- n 个并发进程，信号量初始值为 1，当 n 个进程都执行 P 操作后，信号量的值为多少?
- 信号量初值为 4，多次 PV 操作后变为 -2，那么获得资源的进程数目是多少?
- 5 个并发进程，信号量初始值为 3，那么信号量取值范围是多少?



本讲小结

- 处理进程间接协作
- 处理进程直接协作
- 信号量值区间分析



内容纲要

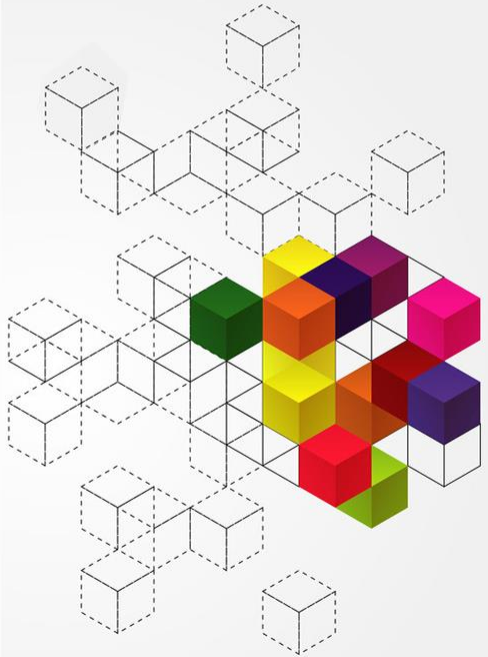
6.7 生产者消费者问题

一、生产者消费者问题简介

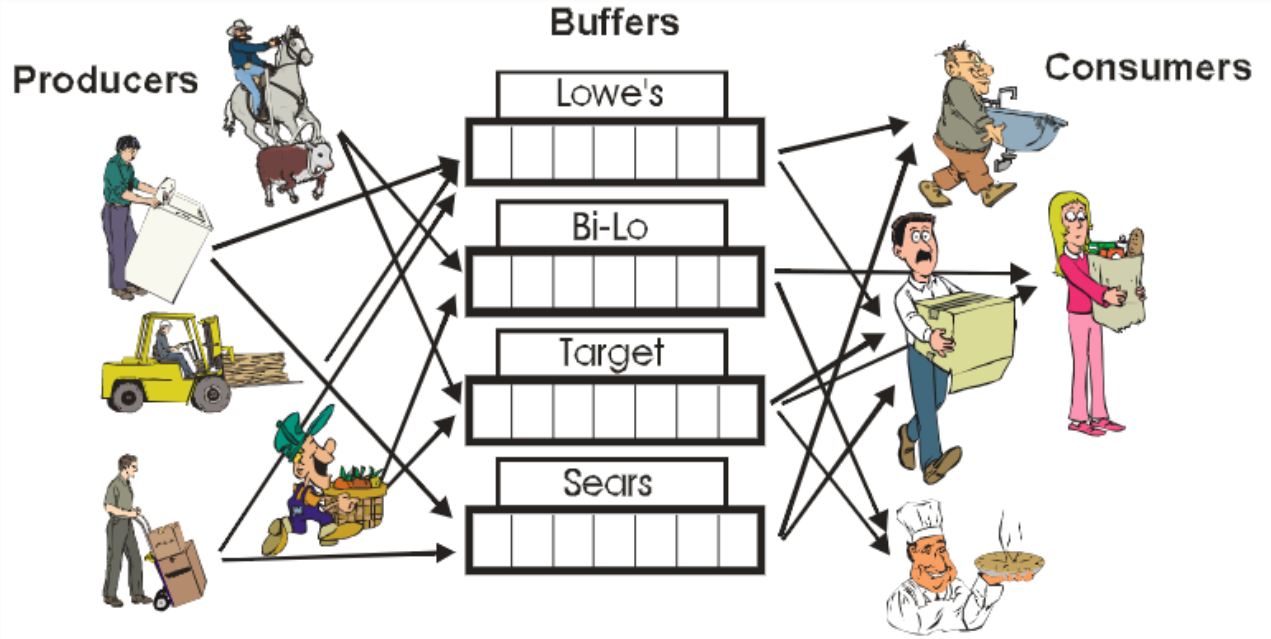
二、生产者消费者问题基础款

三、生产者消费者问题扩展版

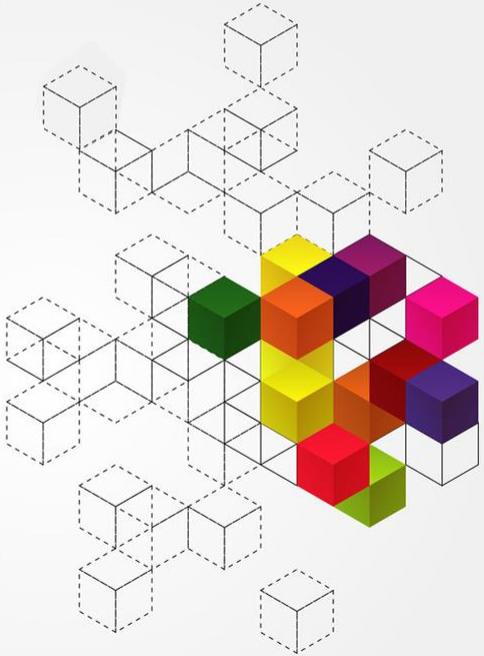
四、生产者消费者问题完整版



一、生产者消费者问题简介

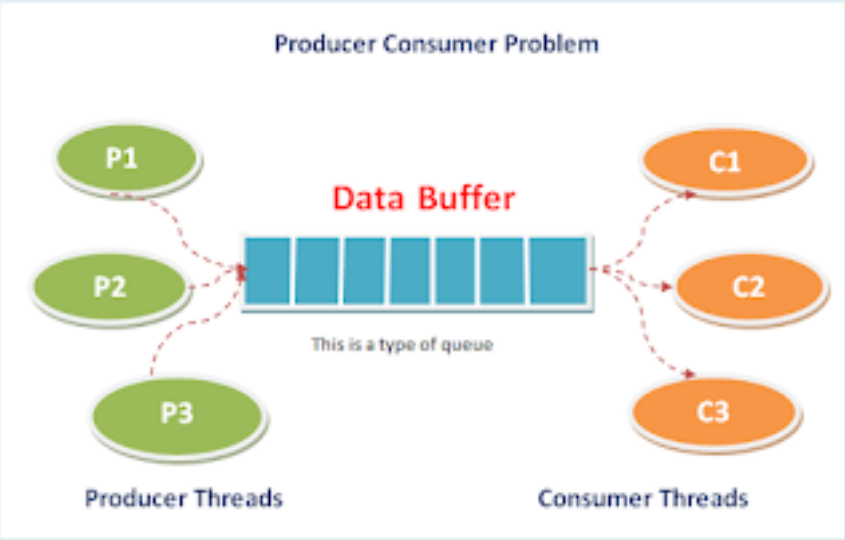


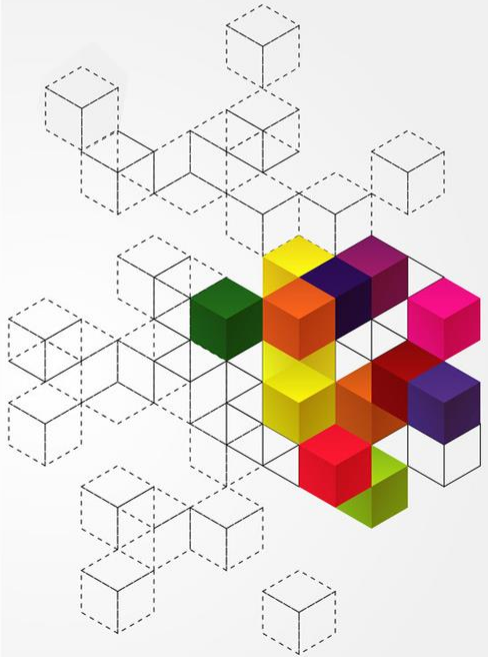
Producers-Stores-Consumers



一、生产者消费者问题简介

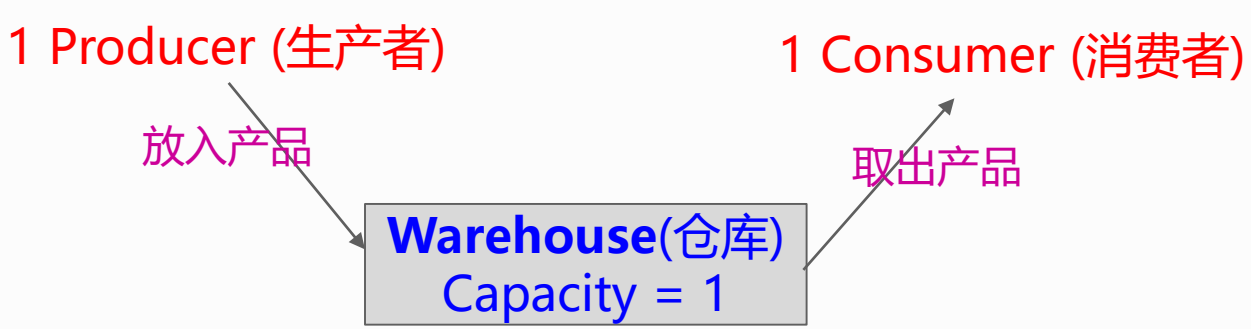
- 生产者消费者问题（Producer Consumer Problem）
  - 又称Bounded Buffer Problem
  - 给定有限大小的缓冲区，生产者将生产出的产品放入缓冲区，消费者从缓冲区取出产品



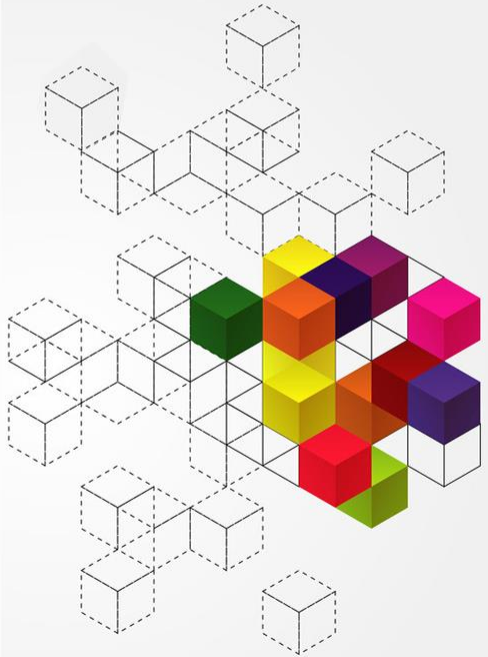


二、生产者消费者问题基础款

- 最简化的版本
  - 1个生产者，1个消费者，共享缓冲区大小=1

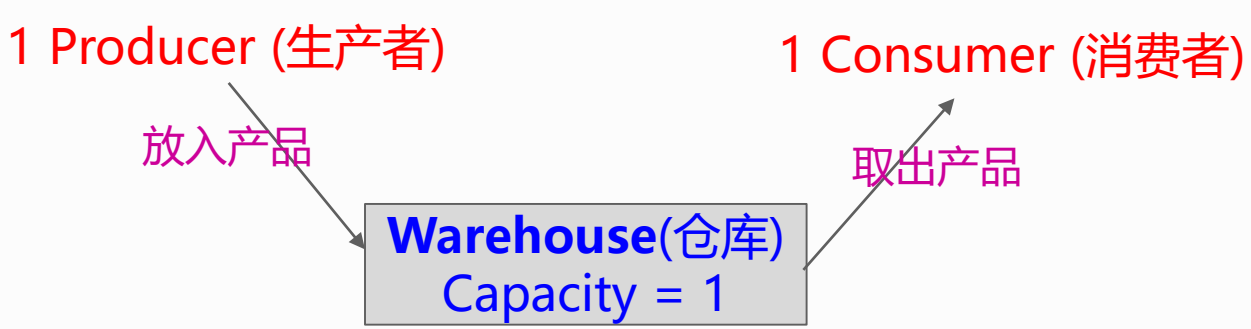


- 进程协作关系分析步骤
  1. 理解问题本质，列出涉及的进程
  2. 分析进程的协作关系
  3. 根据进程协作关系设立信号量，并在程序合理位置通过P/V操作施加并发控制



二、生产者消费者问题基础款

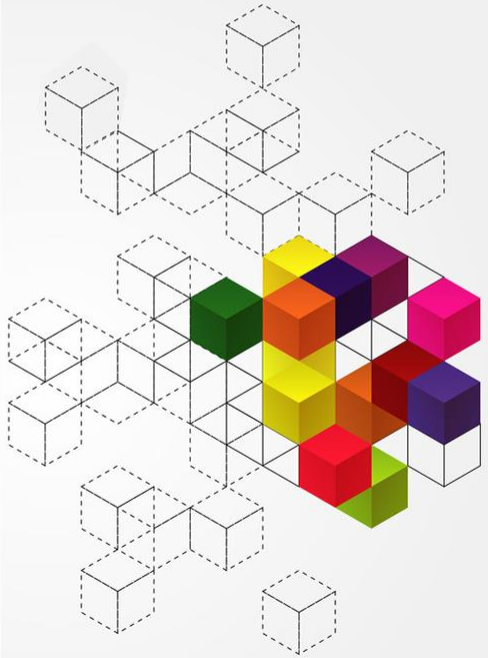
- 最简化的版本
  - 1个生产者，1个消费者，共享缓冲区大小=1



- 步骤1: 列出问题所涉及的进程

```
P:
while (true) {
    生产一个产品;
    将产品放入缓冲区;
};
```

```
C:
while (true) {
    从缓冲区取产品;
    消费产品;
};
```

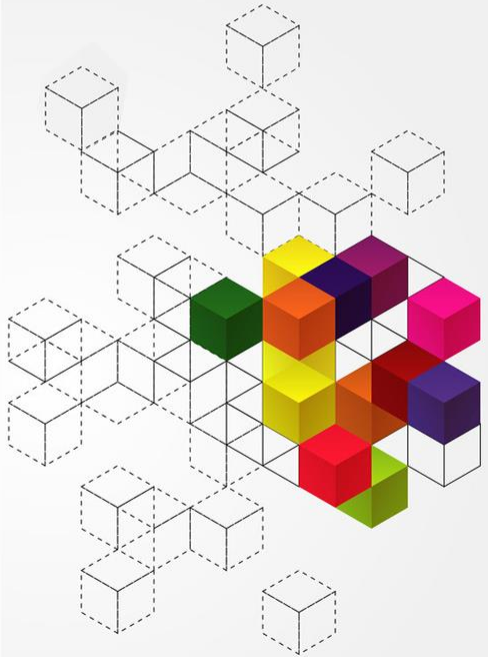


二、生产者消费者问题基础款

- 最简化的版本

```
P:
while (true) {
    生产一个产品;
    将产品放入缓冲区; (put)
};
```

```
C:
while (true) {
    从缓冲区取产品; (get)
    消费产品;
};
```



- 步骤2: 分析进程协作关系
  - 2.1 进程P要执行put操作，要确定缓冲区内有空位存放产品，而除了初始状态之外，这种空位需要进程C的get操作（消费）来创造
  - 2.2 进程C要执行get操作，要确定缓冲区内有存放至少1个产品，产品需要进程P通过生产并通过put操作放入缓冲区

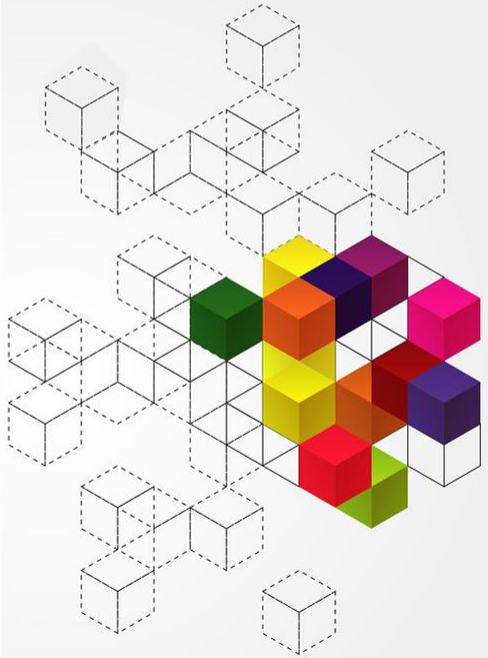
二、生产者消费者问题基础款

- 最简化的版本

```
P:
while (true) {
    生产一个产品;
    将产品放入缓冲区; (put)
};
```

```
C:
while (true) {
    从缓冲区取产品; (get)
    消费产品;
};
```

- 步骤3: 设立信号量，通过P/V操作施加并发控制
  - 信号量empty=1
  - 信号量full=0
  - 信号量的P/V操作应该加到进程合理的位置

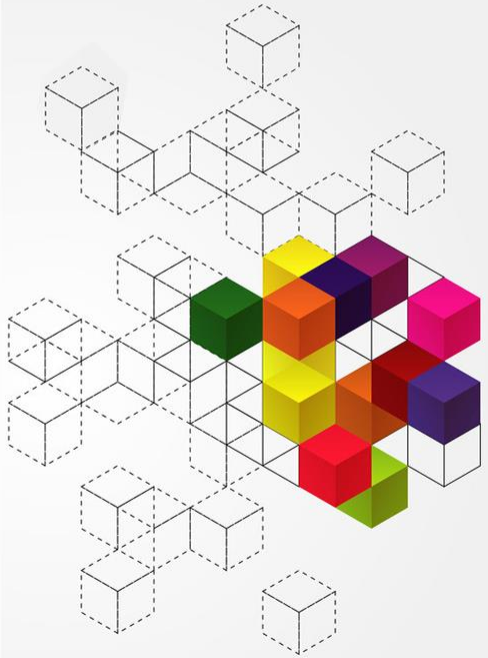


二、生产者消费者问题基础款

- 最简化的版本
  - 基于信号量的解决方案

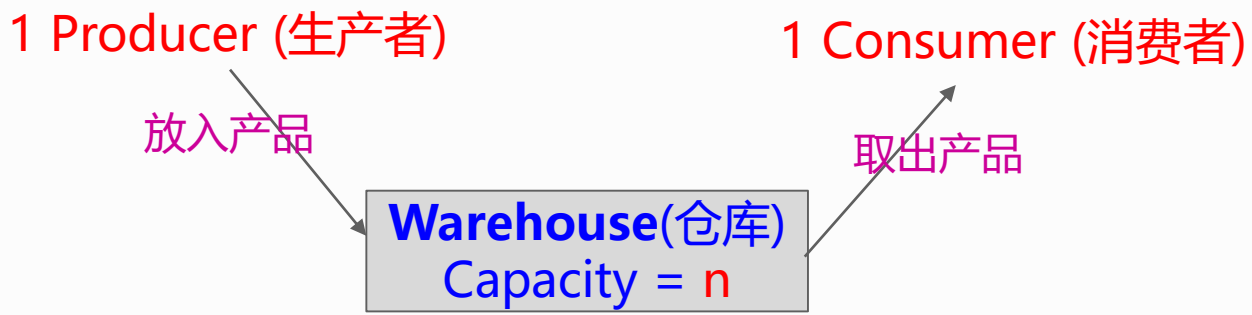
```
int empty = 1;

P:
while (true) {
    生产一个产品;
    P(empty) ;
    送产品到缓冲区;
    V(full);
};
```

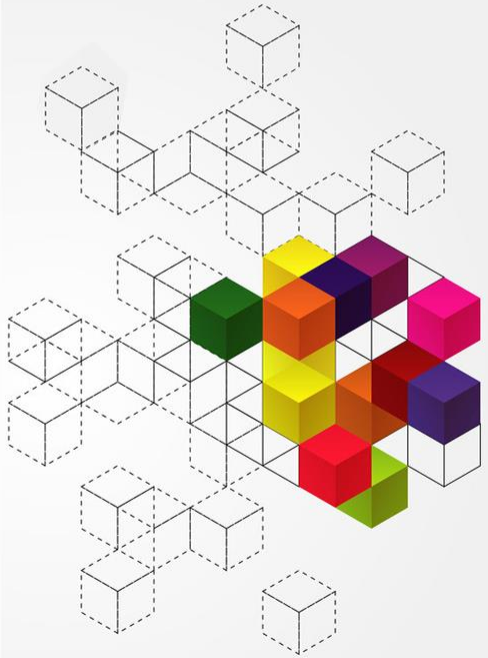
```
int full = 0;

C:
while (true) {
    P(full);
    从缓冲区取产品;
    V(empty);
    消费产品;
};
```



三、生产者消费者问题扩展版

- 扩展版：缓冲区大小 1 -> n



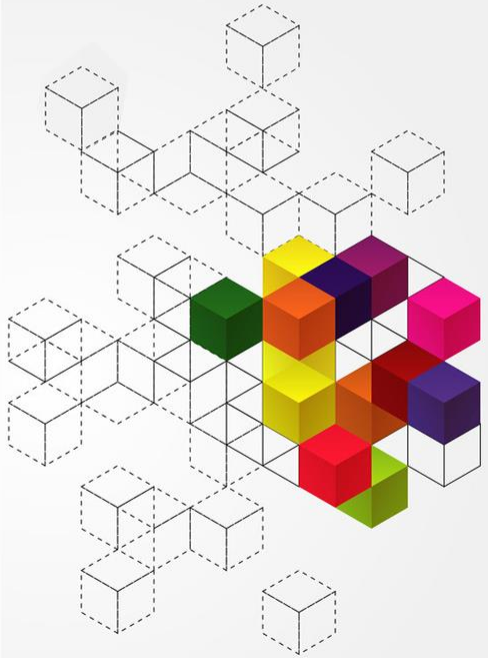


三、生产者消费者问题扩展版

- 扩展版：缓冲区大小 1 -> n

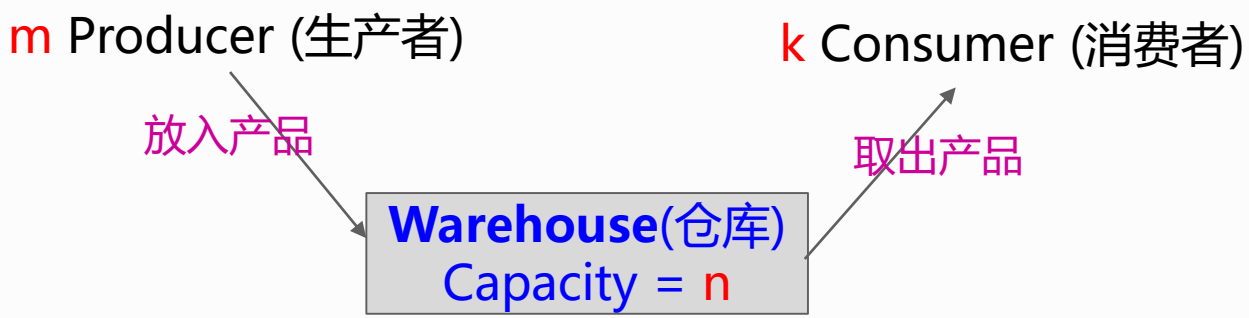
```
int empty = n;
P:
    i = 0;
    while (true)  {
      生产产品;
      P(empty);
     往Buffer [i]放产品;
      i = (i+1) % n;
     V(full);
    };
```

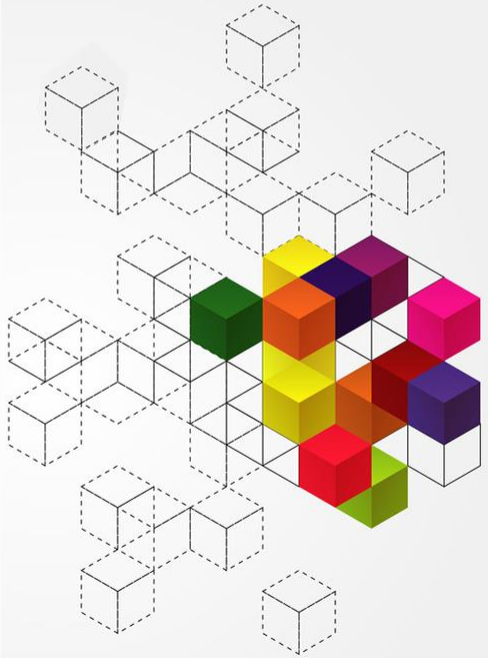
```
int full = 0;
C:
    j =  0;
    while (true)  {
        P(full);
        从Buffer[j]取产品;
        j = (j+1) % n;
        V(empty);
        消费产品;
    };
```



四、生产者消费者问题完整版

- 完整版：
  - m个生产者，k个消费者，缓冲区大小=n



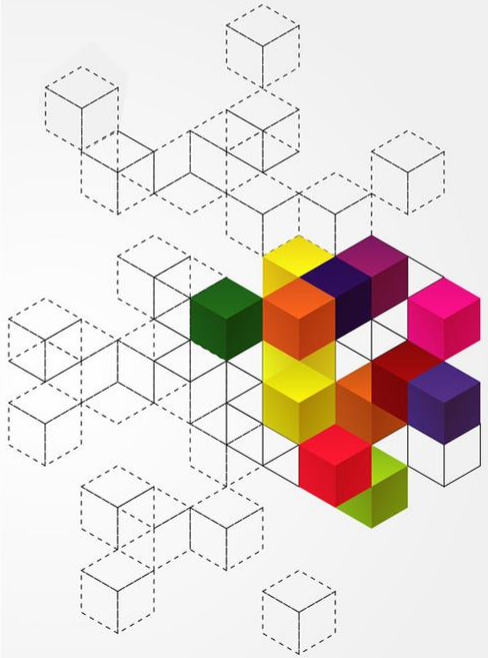


四、生产者消费者问题完整版

- 完整版：
  - m个生产者，k个消费者，缓冲区大小=n

```
P:
    while (true)  {
      生产产品;
      P(empty);
      P(mutex);
      往Buffer [i]放产品;
      i = (i+1) % n;
      V(mutex);
      V(full);
     };
```

```
C:
while (true)  {
    P(full);
    P(mutex);
    从Buffer[j]取产品;
    j = (j+1) % n;
    V(mutex);
    V(empty);
    消费产品;
};
```



本讲小结

- 生产者消费者问题简介
- 基于信号量的生产者问题解法