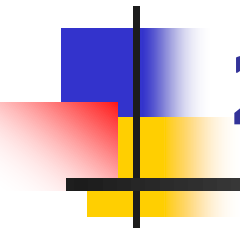
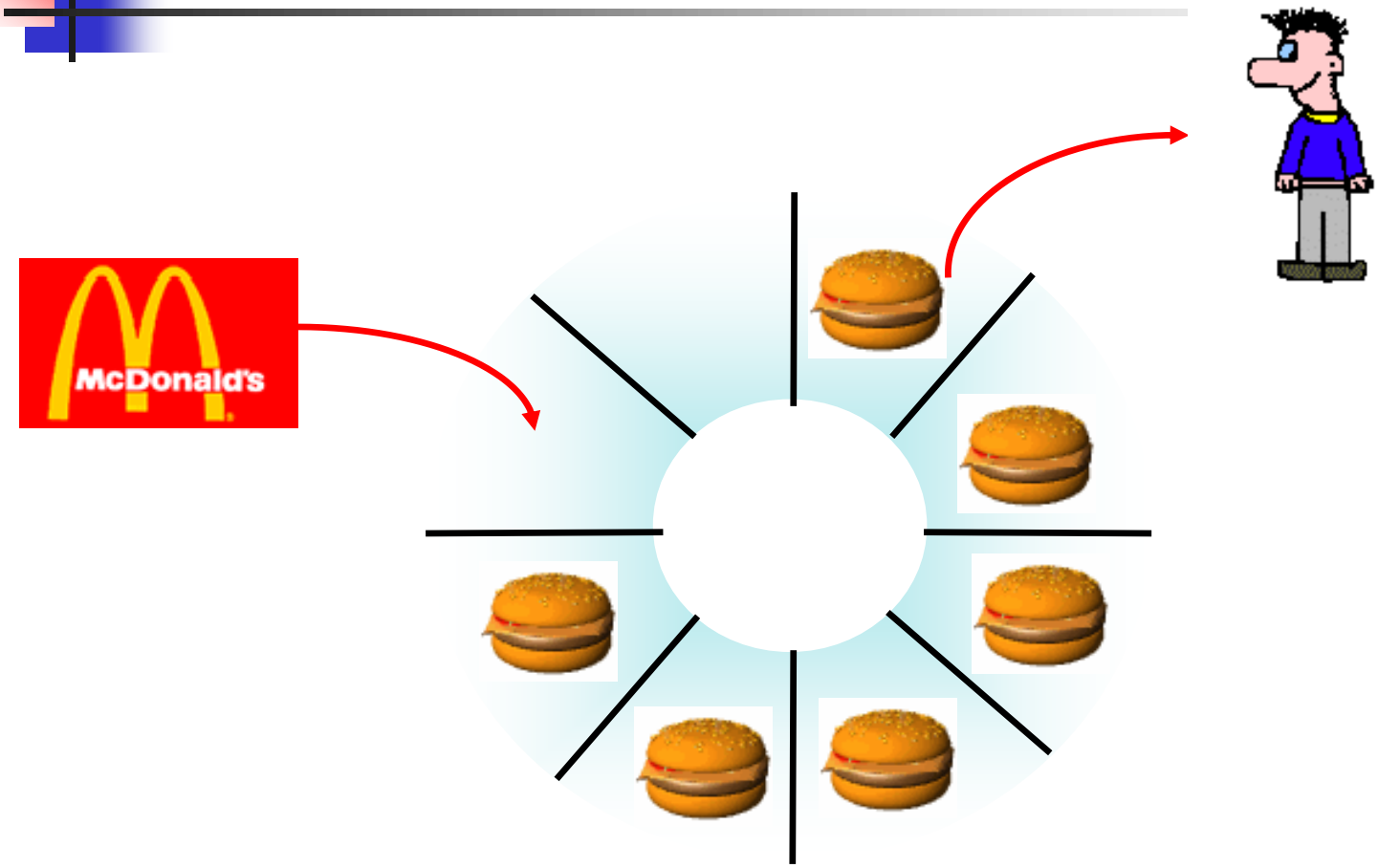


2.4 经典进程同步问题

2.4.1 生产者—消费者问题，

- 关于并发进程互斥和同步的一般模型
- 消费者
  - 使用某一类资源的进程，称为该资源的消费者。
- 生产者
  - 释放某一类的资源。
- 生产者和消费者之间满足以下条件
  - 通过有**n**个缓冲区的公共缓冲池交换信息
  - 消费者想接收数据，缓冲区中至少有一个单元是满的。
  - 生产者想发送数据，缓冲区中至少有一个单元是空的。



生产—消费者问题

2.4.1 生产者—消费者问题，

- **1 记录型信号量**

```
semaphore mutex=1, empty=n, full=0;
item buffer[n];
int     in=0, out=0;
```

```
producer(){
  do{
        ...
        producer an item nextp;
        ...
        wait(empty);
        wait(mutex);
        buffer(in)=nextp;
        in =(in+1) mod n;
        signal(mutex);
        signal(full);
    } while(true);
}
```

```
consumer(){
    do{
        wait(full);
        wait(mutex);
        nextc=buffer(out);
        out=(out+1) mod n;
        signal(mutex);
        signal(empty);
        consumer the item in nextc;
   }while(true);
}
```

```
main(){
   cobegin
      producer(); consumer();
   coend
}
```

- **注意：**
  - 成对出现；次序不能颠倒

2 利用AND信号量解决生产者消费者问题

- 伪码

```
semaphore mutex=1, empty=n, full=0;
item buffer[n];
int     in=0, out=0;
```

```
producer(){
  do{
       produce an item in nextp;
        …
       Swait(empty, mutex);
       buffer(in)  =nextp;
       in  =(in+1)mod n;
       Ssignal(mutex, full);
 } while(true);
}
```

```
consumer(){
     do{
          Swait(full, mutex);
          nextc =buffer(out);
           out  =(out+1) mod n;
           Ssignal(mutex, empty);
           consumer the item in nextc;
 } while(true);
}
```

利用管程解决生产者-消费者问题，

- **1) put(item)**过程。生产者利用该过程将自己生产的产品投放到缓冲池中，并用整型变量**count**来表示在缓冲池中已有的产品数目，当**count≥n**时， 表示缓冲池已满， 生产者须等待。
- **(2) get(item)**过程。消费者利用该过程从缓冲池中取出一个产品，当**count≤0**时，表示缓冲池中已无可取用的产品， 消费者应等待。

3 利用管程解决生产者-消费者问题，

- 伪码

```
monitor producer-consumer{
 int  in=0,out=0,count=0;
 item buffer[N];
 condition notfull, notempty;
```

```
void put(item x){

        if count≥N then cwait(notfull);
        buffer[in] = x;
         in=(in+1) % N;
         count++;
         csignal(notempty);
 }
```

```
 void get(item x){
        if (count≤0)cwait(notempty);
        x =buffer[out];
        out =(out+1) % N;
        count--;
        csignal(notfull);
 }
}
```

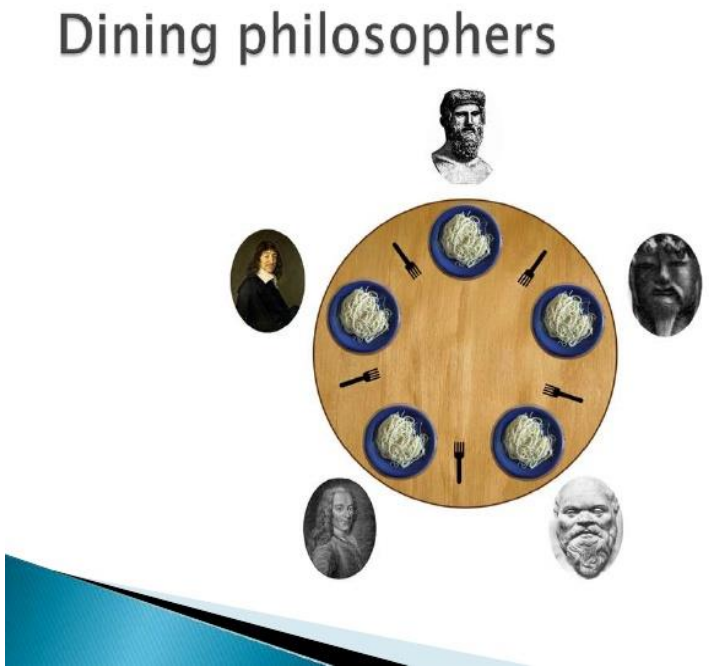
利用管程解决生产者-消费者问题。

- 生产者和消费者可描述为：

```
Producer(){
  item x;
  while(1){
          produce an item in x;
          PC.put(x);
   }
}
Consumer(){
      item x
      while(1)
          PC.get(x);
          consume the item in x;
      };
}
```

2.4.2 哲学家进餐问题

- （**Dijkstra，1965**）
- 有五个哲学家围坐在一圆桌旁，桌子中央有一盘通心面，每人面前有一只空盘子，每两人之间放一把叉子。每个哲学家思考、饥饿、吃通心面。
- 欲吃面，每个哲学家必须获得两把叉子，且每人只能直接从自己左边或右边去取叉子。

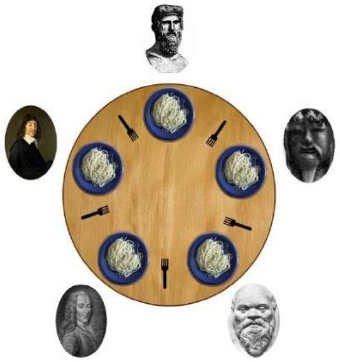


1 记录型信号量解决哲学家进餐问题

```
semaphore chopstick[5] ={1,1,1,1,1};
do{
   wait(chopstick［i］);
   wait(chopstick［(i+1) % 5］);
   eat;
   signal(chopstick［i］);
   signal(chopstick［(i+1) % 5］);
   think;
}while(true);
```

- 解
  - 每一把叉子都是必须互斥使用的，因此，应为每把叉子设置一个互斥信号量，初值均为**1**。
  - 当一个哲学家吃通心面之前必须获得自己左边和右边的两把叉子，即执行两个**P** 操作，吃完通心面后必须放下叉子，即执行两个**V** 操作。
- 隐患？

Dining philosophers



- 只允许4个人同时取左边筷子
- 仅当左右叉子都可用时才进餐
- 规定奇数编号哲学家先左后右；偶数编号相反。
- 1、2号哲学家竞争1号筷子；3、4号哲学家竞争3号筷子。即五位哲学家都先竞争奇数号筷子，获得后，再去竞争偶数号筷子，最后总会有一位哲学家能获得两只筷子而进餐
- 其思路？同步？互斥？

2 使用AND信号量机制解决哲学家就餐

- 要求每个哲学家先获得两个临界资源(筷子)后方能进餐，

```
semaphore chopstick[5] ={1,1,1,1,1};
do{
        think;
        Sswait(chopstick［(i+1) % 5］, chopstick ［i］);
        eat;
        Ssignal(chopstick ［(i+1) % 5］, chopstick ［i］);
    }while(true);
```

2.4.3 读者—写者问题

- 要求
  - 多个进程共享文件或记录
  - 允许多个进程同时读共享数据
  - 不允许写进程和其它读写进程同时访问数据
  - 无读进程时可写
- 解
  - 读、写进程需要互斥（信号量）
  - 记录读进程的数目，无读进程才可写（信号量）

记录型信号量解决读者—写者问题

- 伪码

```
semaphore rmutex=1, wmutex=1;
int readcount =0;
```

```
Void reader(){
  do{
      wait(rmutex);
       if (readcount=0) wait(wmutex);
       readcount++;
       signal(rmutex);

       perform read operation;

      wait(rmutex);
      readcount--;
      if (readcount=0) signal(wmutex);
       signal(rmutex);
  while(true);
}
```

```
void writer(){
  do{
      wait(wmutex);
      perform write operation;
      signal(wmutex);
  while(true);
}
```

利用信号量集机制解决读者写者问题

- 最多允许N个读者同时读

```
#define  N  88
semaphore L=N, mx=1;
void reader()
  do{
        swait(L,1,1);//控制读者数目
        swait(mx,1,0);//无写进程可读
          perform read operation;
        ssignal(L,1);
      }while(true);
 }
```

```
    void writer(){
        do{
            swait(mx,1,1; L,N,0);
            //无写进程，
            //无读进程，即L＝N；
             perform write operation;
             ssignal(mx,1);
        }while(true)
}
```