行到水穷处，坐看云起时

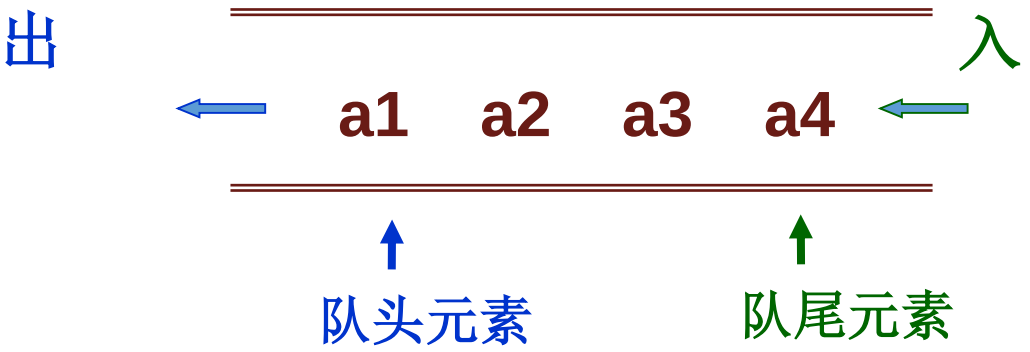
3.4 队列的基本概念和类型定义

队列：一种限定性的数据结构，限定只能在表的**一端**进行**插入**，另一端进行删除.

队头(Front)：允许删除的一端.

队尾(Rear)：允许插入的一端

队列是先进先出表（FIFO）.



思考

如果进队顺序为1，2，3，4，则下面哪一种出队顺序是正确的.

(1) 1, 2, 3, 4

(2) 2, 1, 4, 3

(3) 3, 1, 2, 4

(4) 4, 3, 2, 1

队列的抽象数据类型定义

ADT Queue {

数据对象：

D＝{ ai | ai ∈D, i=1,2,...,n, n≥0 }

数据关系：

R1＝{ <ai-1, ai >| ai-1, ai∈D, i=2,...,n }

约定an 端为队尾，a1 端为队头。

基本操作：

InitQueue (*Q)：设置一个空队Q。

InQueue(*Q ,e)：入队，在队中插入一个新的队尾元素。

OutQueue(*Q,*e) ：出队，删除队头元素。

GetFront (Q,*e) ：读取队头元素；若队为空，返回空值。

QueueEmpty (Q) ： 判断队是否为空。

如果为空返回1，非空返回0。

ClearQueue(*Q):将队清空

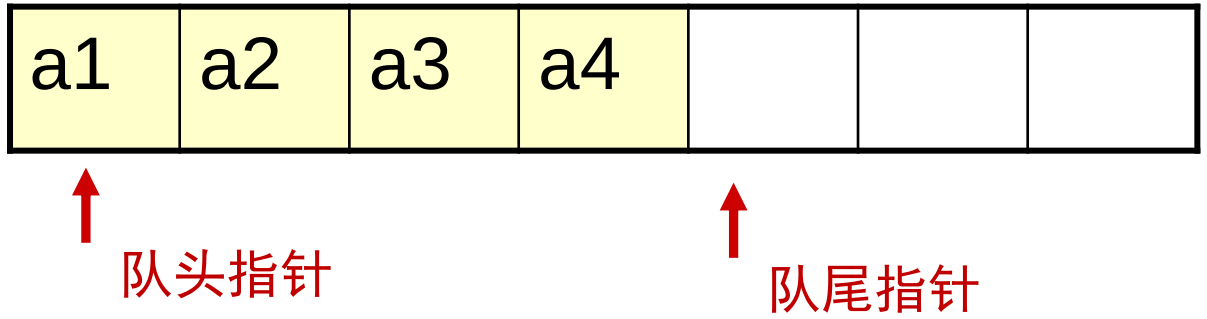
} ADT Queue

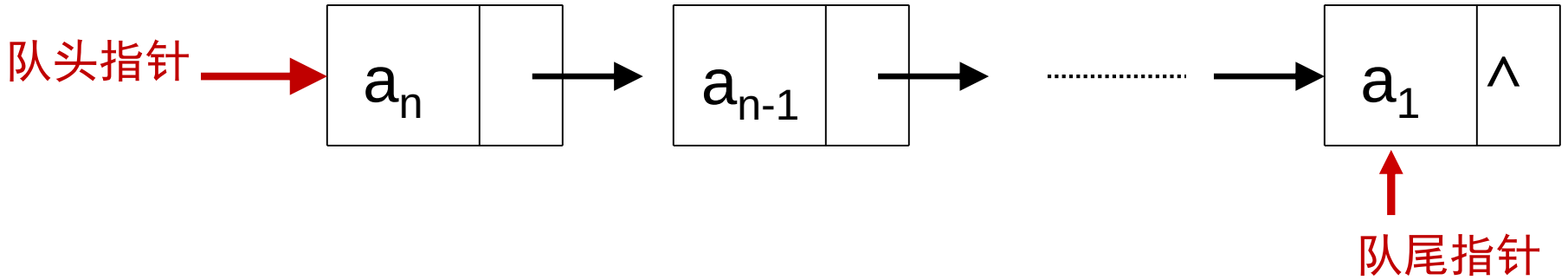
3.5 队列的实现

两种实现方法

顺序队：顺序存储

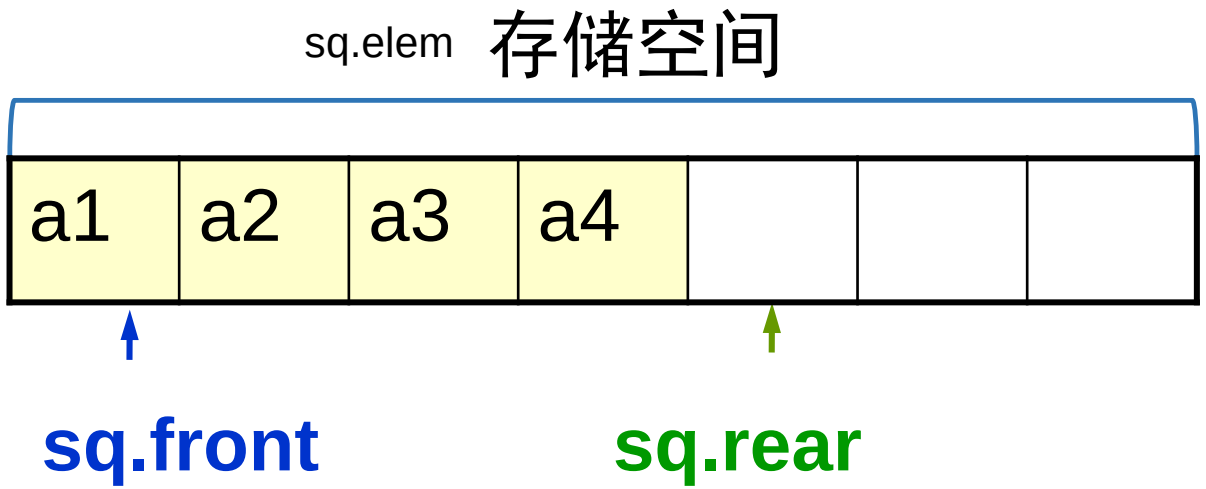
链队：链式存储





3.5.1 队列的顺序存储结构

```
typedef struct {
    ElemType *elem; //待分配存储空间
    int front;
    int rear;
} SeqQueue ;
```



SeqQueue sq;

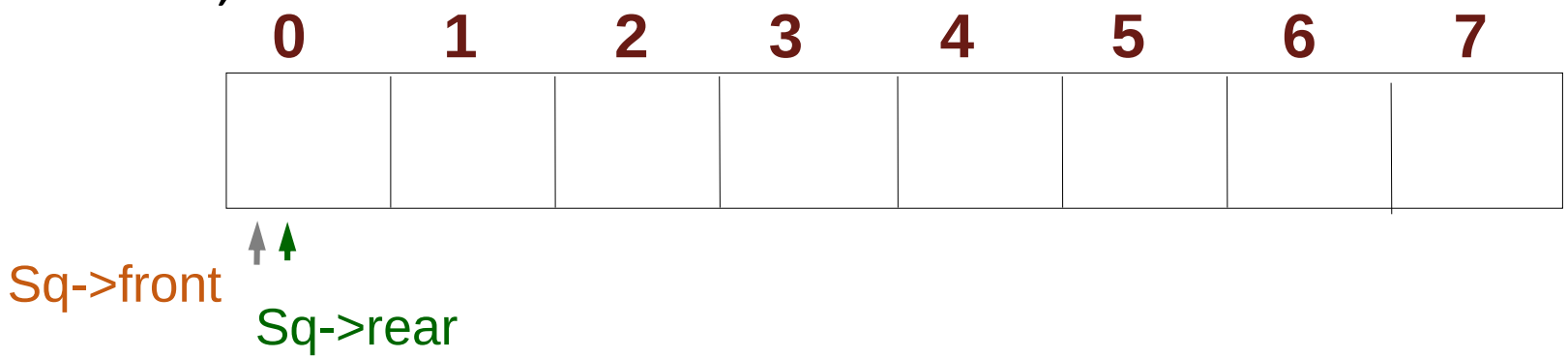
规定队头front指向队头元素；队尾rear指向队尾元素的下一个元素

3.5.2 基本操作

（1）初始化

```
typedef  struct {
   ElemType  *elem;
   int  front;
   int  rear;
} SeqQueue ;
```

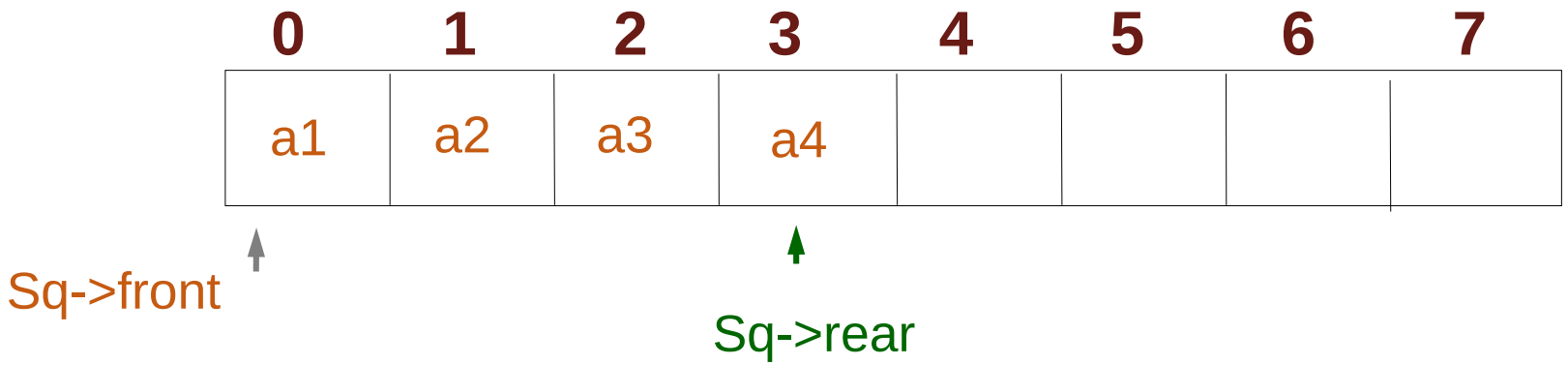
```
void InitQueue(SeqQueue  *sq)
{    sq->elem=(ElemType*)malloc(sizeof(ElemType)*MAXSIZE);
     sq->front = sq->rear = 0;
}
```



（2） 入队

```
status InQueue(SeqQueue  *sq, ElemType e)
{    if(sq->rear==MAXSIZE)
         return 0;
     sq->elem [sq->rear++]=e;
     return 1;
}
```

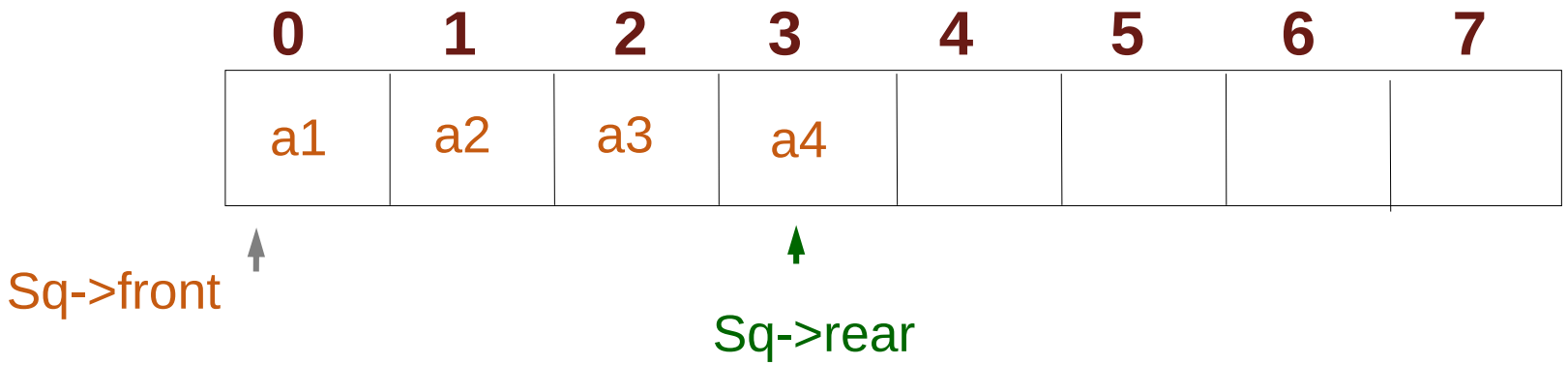
是否存在问题？



（3） 出队

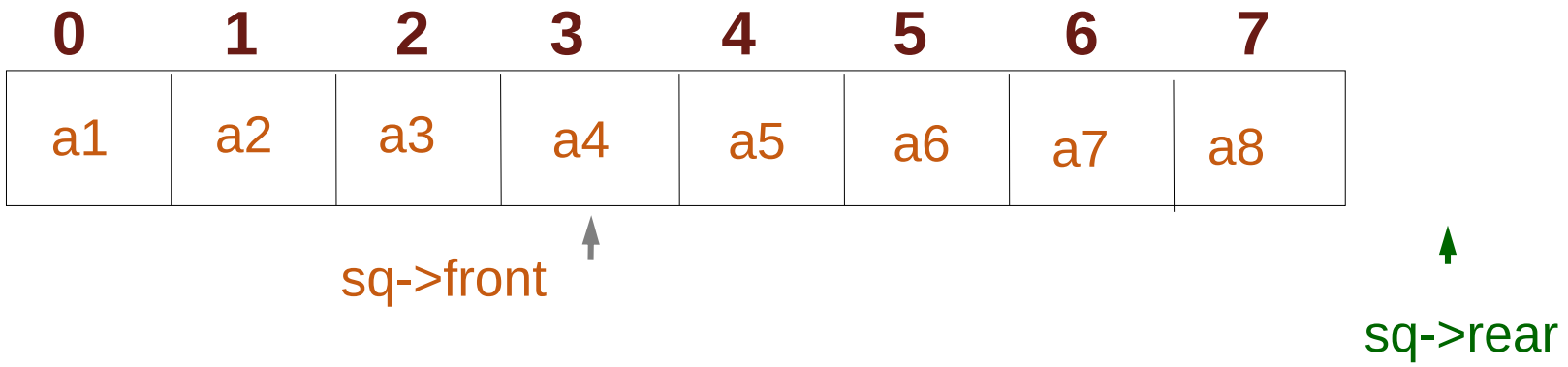
```
status OutQueue(SeqQueue  *sq, ElemType *e)
{    if(sq->front==sq->rear)
         return 0;
     *e = sq->elem[sq->front];
     sq->front++;
     return 1;
}
```

是否存在问题？



存在的问题

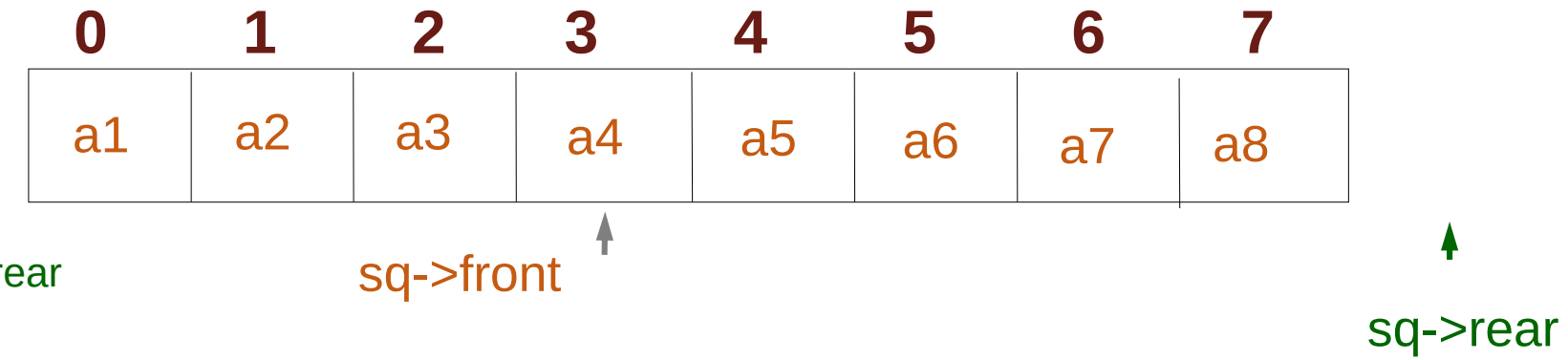
sq->rear++和sq->front++会带来问题 ？

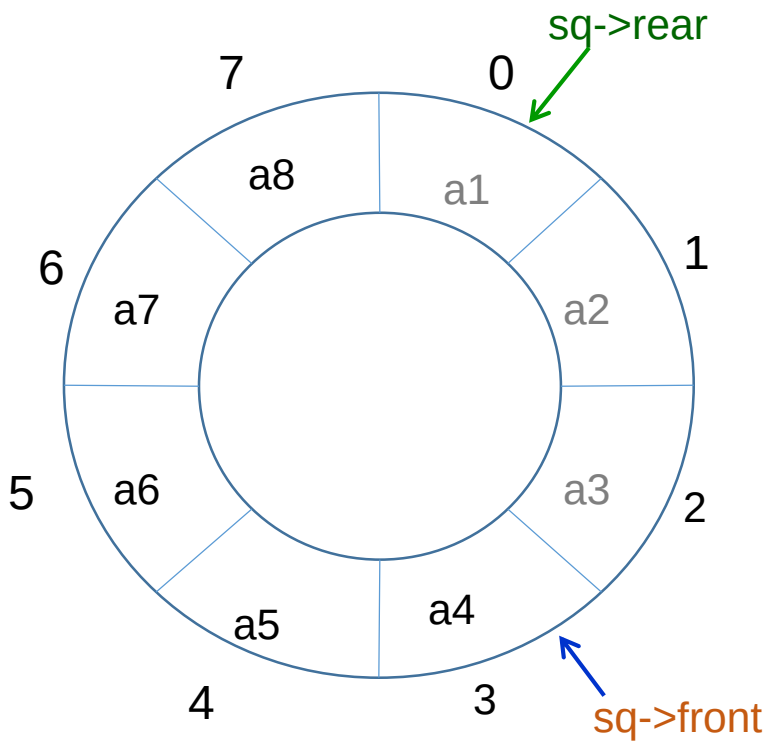


sq->rear == MAXSIZE 假溢出现象

解决办法：使sq->rear总是指向队列空间

改进：循环队列





入队列：

```
sq->elem[sq->rear] =  e;
sq->rear= (sq->rear+1) %  MAXSIZE;
```

出队列：

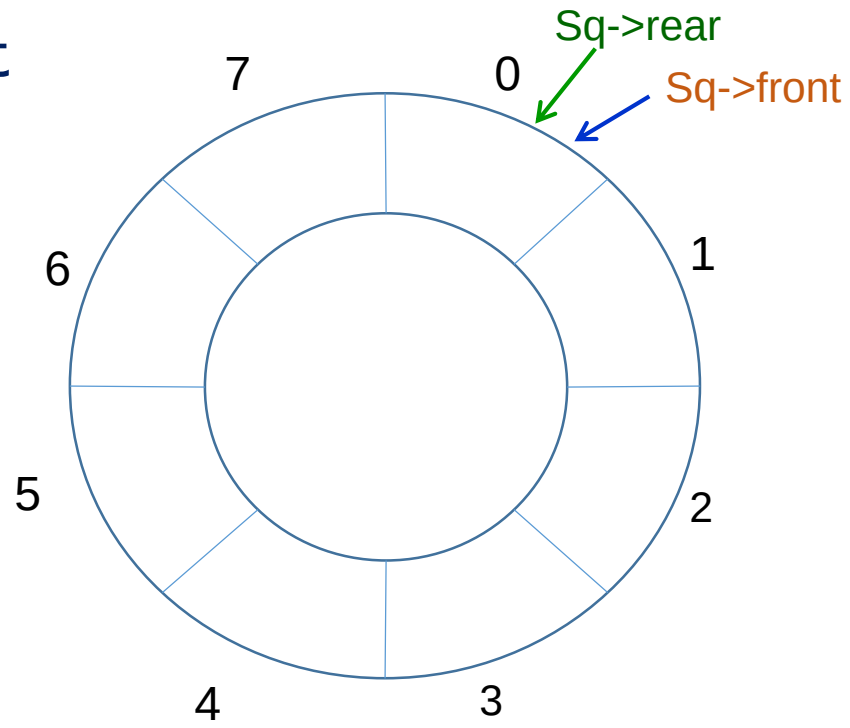
```
sq->front =  (sq->front +1 )%  MAXSIZE;
```

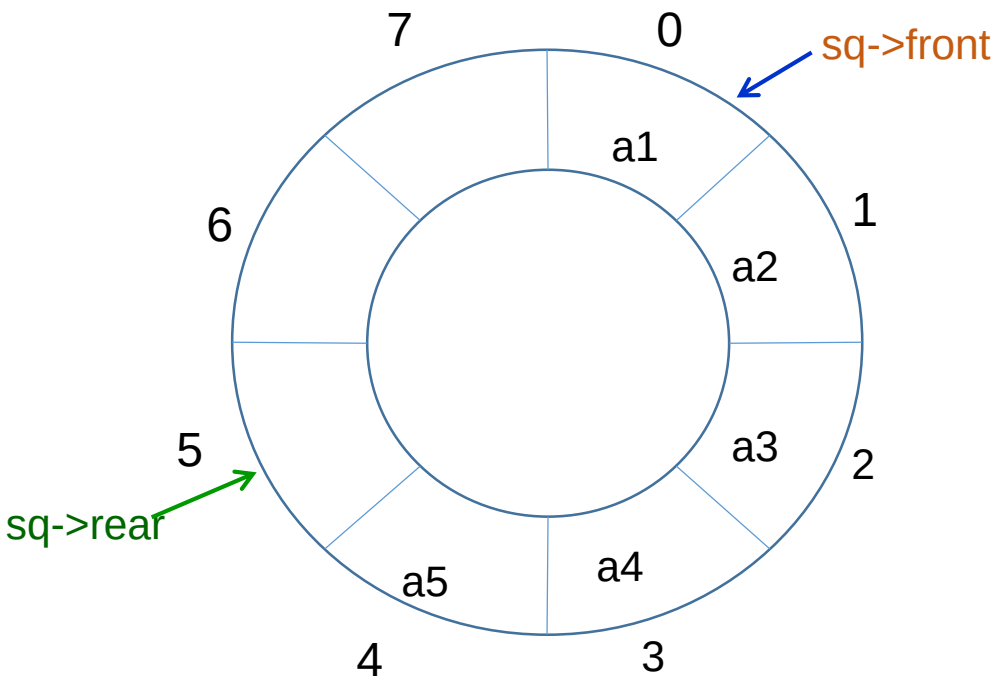
新的问题

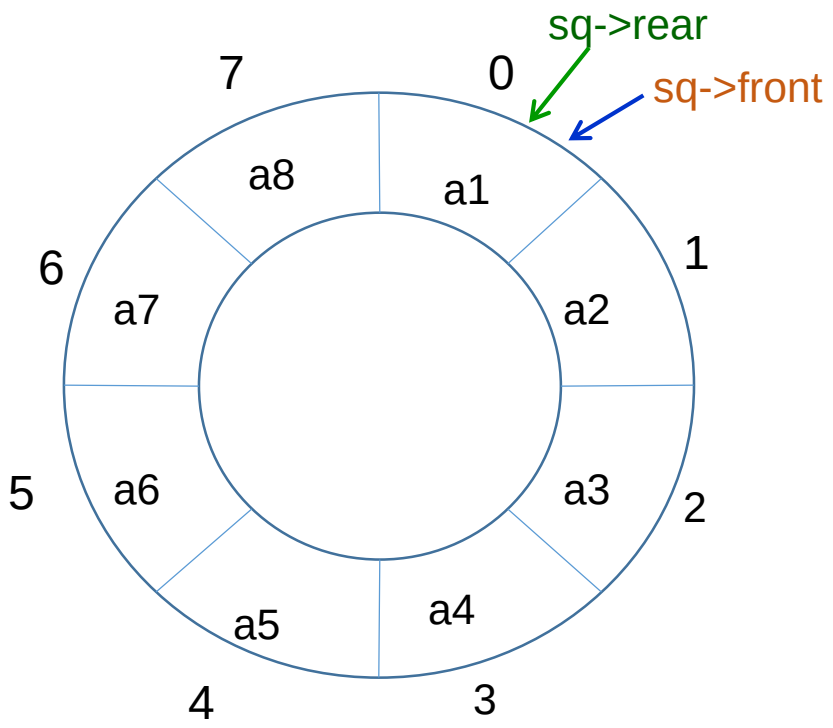
循环队列中队空队满的判断

队空　sq->rear == sq->front

队满　？　sq->rear ？







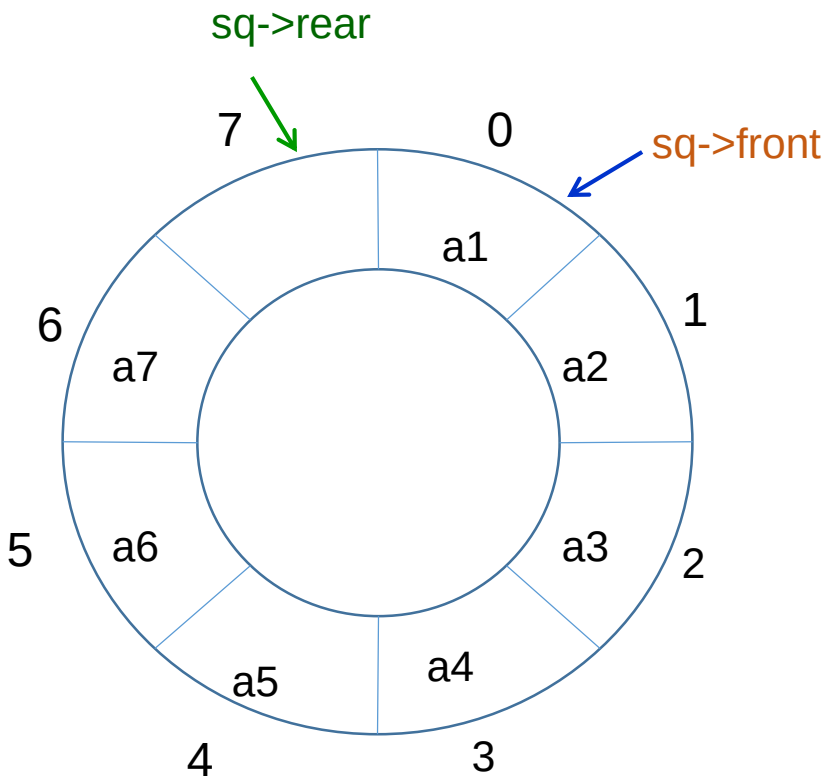
队满条件sq->rear == sq->front

解决方法

[规定]sq->rear单元不放元素

队满的条件：

(sq->rear+1)%MAXSIZE= = sq->front



循环队列的完整描述

```
typedef  struct {

  ElemType  *elem;
  int  front;
  int  rear;
  } SeqQueue ;
```

```
typedef int status;
InitQueue(SeqQueue *sq);
InQueue(SeqQueue *sq,ElemType e);
OutQueue(SeqQueue *sq,ElemType *e);
status IsFull(SeqQueue sq);
status  IsEmpty(SeqQueue sq);
```

不自见故明，

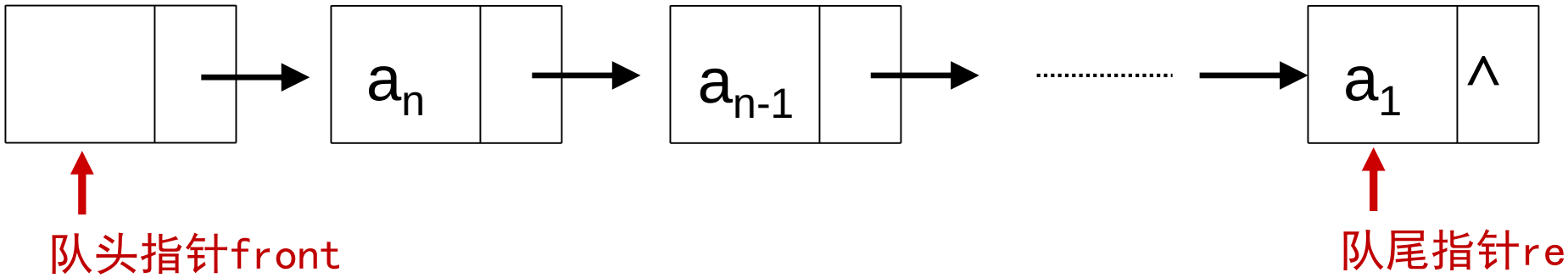
不自是故彰，

不自伐故有功，

不自矜故长。

3.5.3 链队列

带头结点的单链表实现链队



注意：既然设两个指针变量，分别指向单链表的头结点和尾结点.

因此可以定义一个结构体类型

链队列类型定义

结点类型：

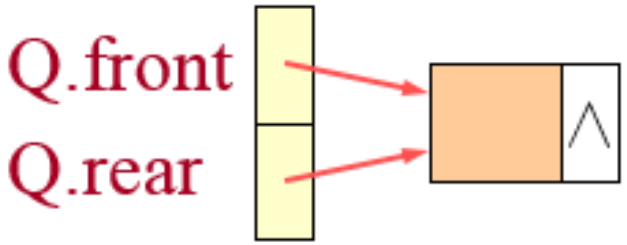
```
typedef struct QNode {
    ElemType      data;
    struct QNode  *next;
  } QNode, *QNodePtr;
```

链队列类型：

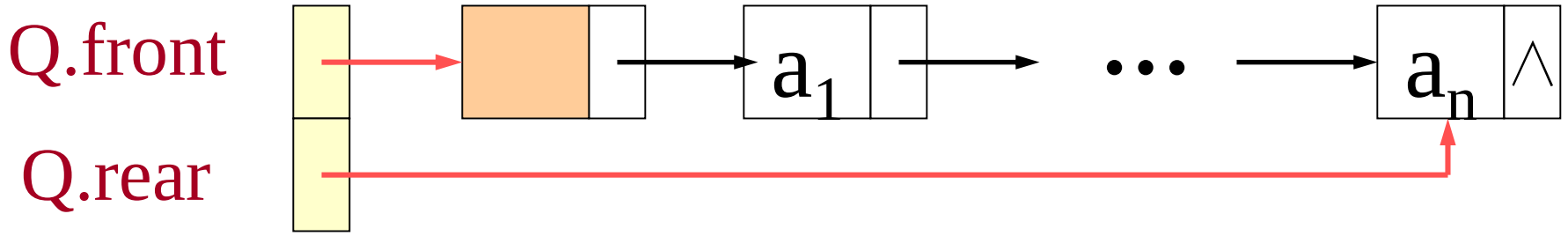
```
typedef struct {
   QNodePtr  front;  // 队头指针
    QNodePtr  rear;   // 队尾指针
} LinkQueue;
```

链队列示意图

LinkQueue Q;




```
typedef struct QNode {
    ElemType      data;
    struct QNode  *next;
  } QNode, *QNodePtr;
typedef struct {
   QNodePtr  front;  // 队头指针
    QNodePtr  rear;  // 队尾指针
} LinkQueue;
```

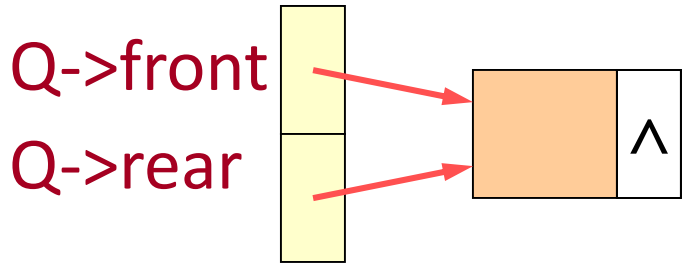


3.5.4 链队列基本操作实现

（1）初始化操作

```
status InitQueue (LinkQueue *Q) {
  // 构造一个空队列Q
  Q->front = Q->rear
          =(QNodePtr)malloc(sizeof(QNode));
  if (!Q->front) return 0;
  Q->front->next = NULL;
  return 1;
}
```

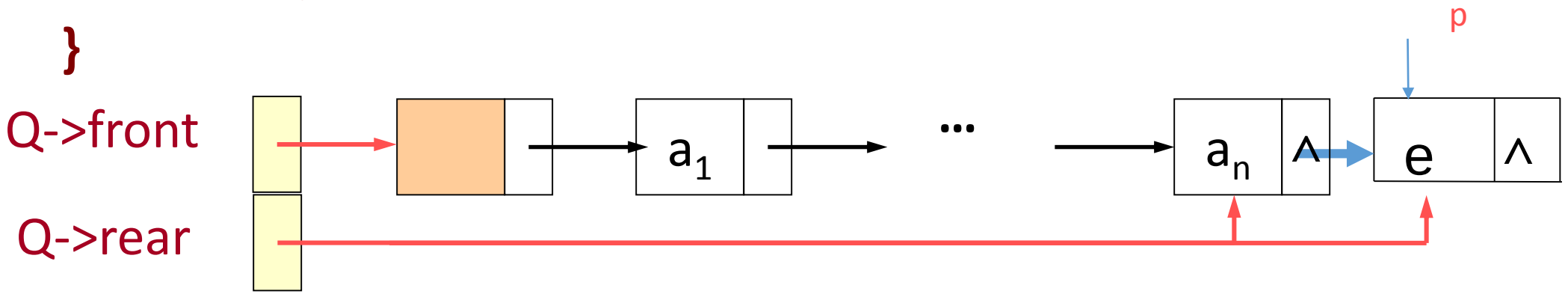
```
typedef struct QNode {
   ElemType     data;
   struct QNode  *next;
 } QNode, *QNodePtr;
typedef struct {
  QNodePtr  front;  // 队头指针
   QNodePtr  rear;  // 队尾指针
} LinkQueue;
```



空队列

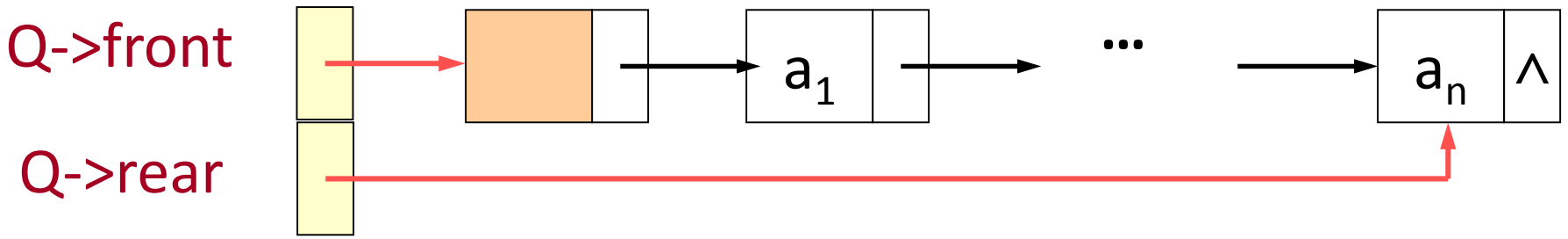
(2) 入队操作

```
status InQueue (LinkQueue *Q, ElemType e) {
   p =  (QNode)malloc(sizeof(QNode)); // new QNode;
   if (!p)  return 0;   //存储分配失败
   p->data = e;   p->next = NULL;
   Q->rear->next = p;    Q->rear = p;
   return 1;
}
```



(3) 出队操作

```
Status DeQueue (LinkQueue *Q, ElemType *e)
{
 if (Q->front == Q->rear)    return 0;
   p = Q->front->next;   *e = p->data;
  Q->front->next = p->next;
   if (Q->rear == p)  Q->rear = Q->front;
  free (p);      return 1; }
```



3.6 队列的应用

应用一：排队问题模拟——以银行排队为例

问题描述

顾客到银行办理业务，首先需拿号并排队等候，当空闲窗口叫号时，按排队顺序去办理业务，业务办理完毕，离开银行。

问题：如何模拟银行排队办理业务的过程？

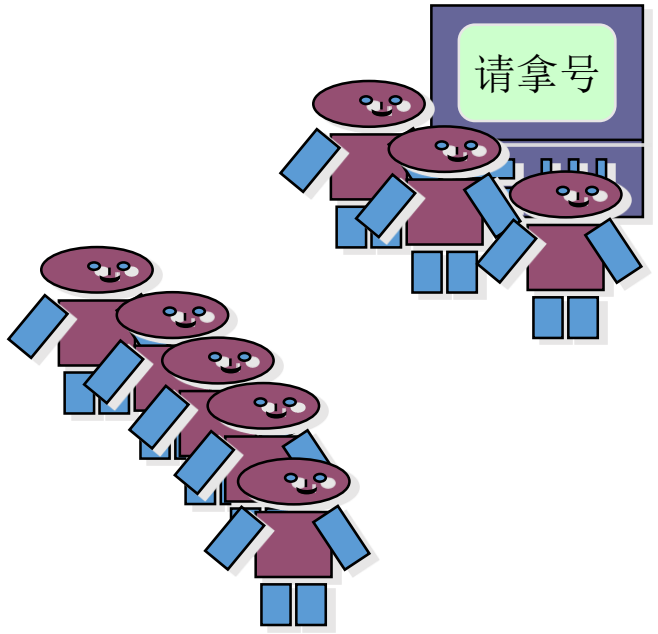
分析问题

业务过程

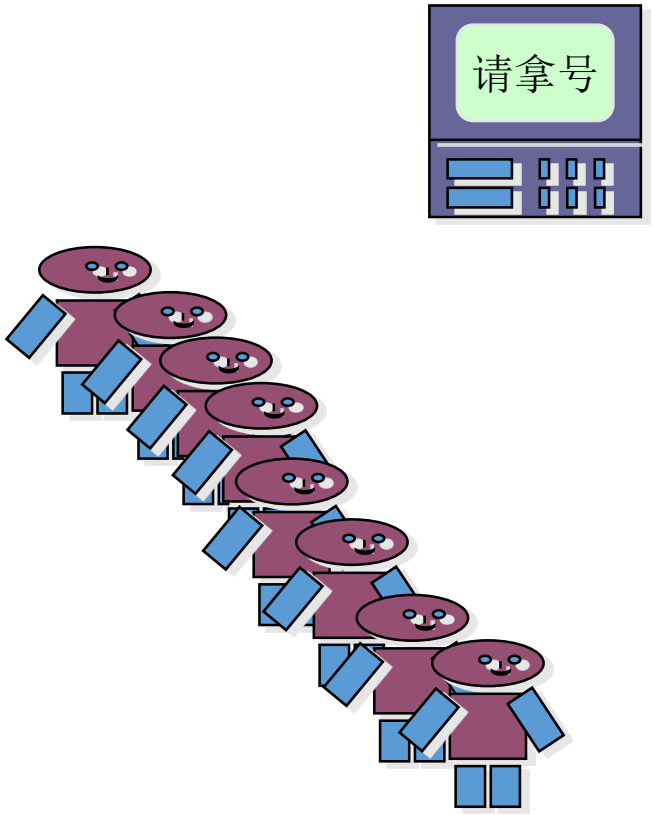
窗口1

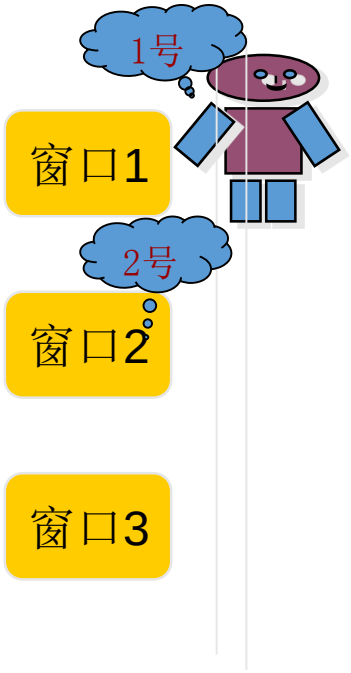
窗口2

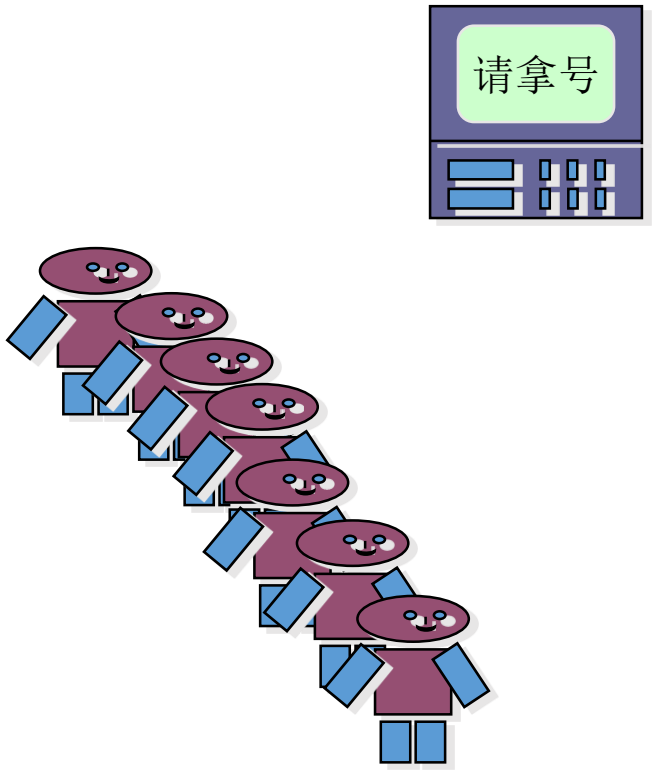
窗口3



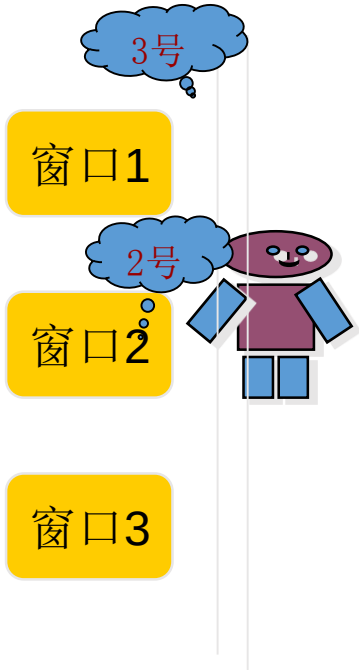
业务过程

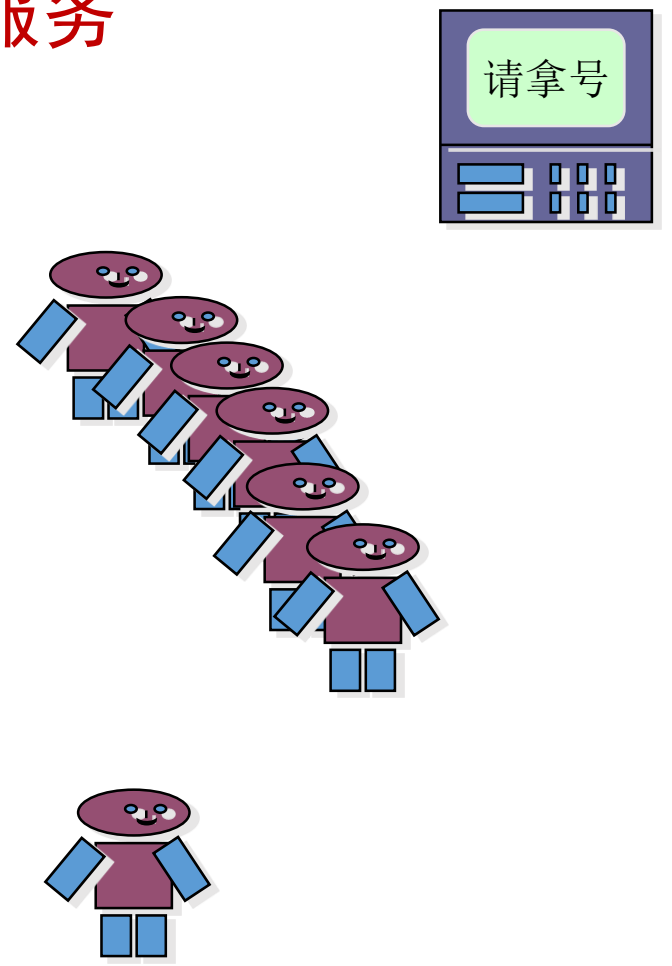



1号
窗口1
2号
窗口2
窗口3


请拿号

服务特点：队头顾客出队办理业务，新到顾客站到队尾；　先到顾客先拿号，先获得服务





模拟需解决以下问题：

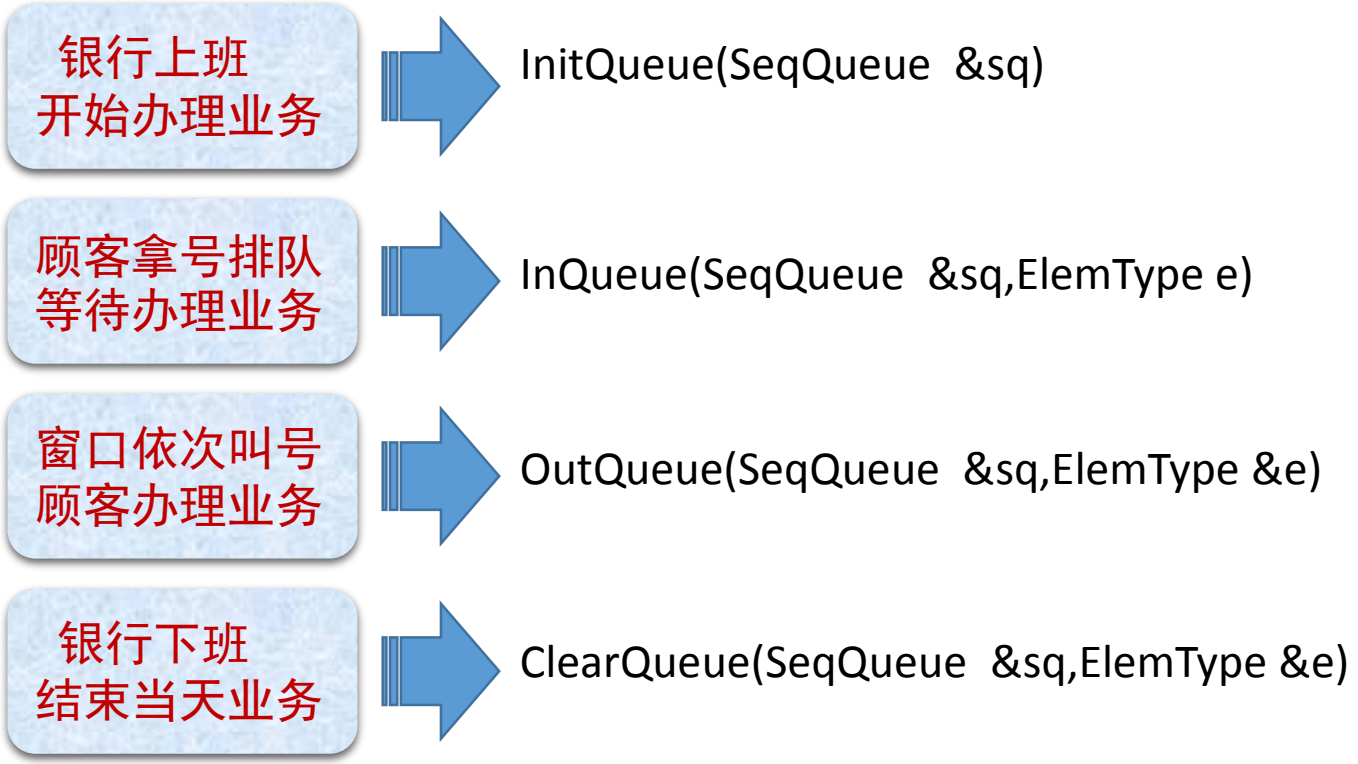
✓ 数据及其关系？

✓ 如何存储？

✓ 数据上的操作？

数据：拿号的顾客

关系：依次排队



应用二：运动会比赛安排

问题描述

某运动会设立 n 个比赛项目，每个运动员可以参加1至3个项目。试问如何安排比赛日程既可以使同一运动员参加的项目不安排在同一单位时间进行，又使总的竞赛日程最短。

该问题中的数据

输入数据：n个项目，m个运动员，报名信息

✓设有 9 个项目：P = { 0，1，2，3，4，5，6，7，8 }，

✓7名运动员：A1，A1…，A7；

✓各运动员报名情况依次为：（1，4，8）、（1，7）、（8，3）、（1，0，5）、（3，4）、（5，6，2）、（6，4）

数学模型分析

分析：有同一运动员参加的项目，不能安排在相同的时间段，称之为“冲突”关系。

以运动员1为例，报名参加的项目分别为：（1，4，8），则1，4，8三个项目必须安排到3个不同的时间段，即其中的任意两个都不能安排在同一时间段，以保证运动员可以参赛。则1，4，8之间两两具有“冲突”关系。

七名运动员报名参加的项目分别为：

（1，4，8）、（1，7）、（8，3）、（1，0，5）、（3，4）、

（5，6，2）、（6，4）

它们之间的冲突关系为：

R = {（1，4），（4，8），（1，8），（1，7），（8，3），

（1，0），（0，5），（1，5），（3，4），（5，6），（5，2），

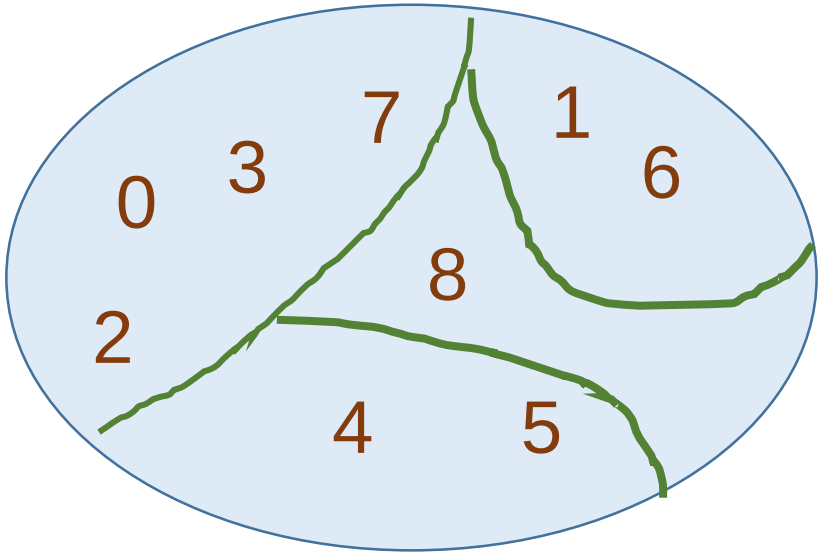
（6，2），（6，4）}

数学模型-划分子集问题

n 个比赛项目构成一个大小为 n 的集合，有同一运动员参加的项目则抽象为“冲突”关系。若将此问题抽象成数学模型，则归属于“划分子集”问题。

对前述例子而言，问题即为：运动会项目为集合，将其划分为若干子集。同一子集的项目为可以同时进行的项目（即互不冲突），同时希望运动会的日程尽可能短。

求解方法：可利用“过筛”的方法来解决划分子集问题。从第一个元素考虑起，过筛出一批互不冲突的元素为第一个子集；然后再“过筛”出一批互不冲突的元素为第二个子集；依次类推，直至所有元素都进入某个子集为止。



R = {（1，4），（4，8），
（1，8），（1，7），（8，3），
（1，0），（0，5），（1，5），
（3，4），（5，6），（5，2），
（6，2），（6，4）}

所有数据的表示与存储

初始数据

- ✓运动会项目:线性表
- ✓参赛信息：（1，4，8）、（1，7）、…
- ✓运动员信息:线性表

提取出的信息

- ✓冲突信息

结果信息

- ✓暂存未分组的项目
- ✓项目分组信息 ——可以集合的表示

数据存储

- ✓用0~9简化表示运动会项目号，只需存储运动会项目号：顺序
- ✓报名参赛信息：顺序or链式
- ✓冲突关系表示和存储：二维数组
- ✓划分集合的表示：一维数组

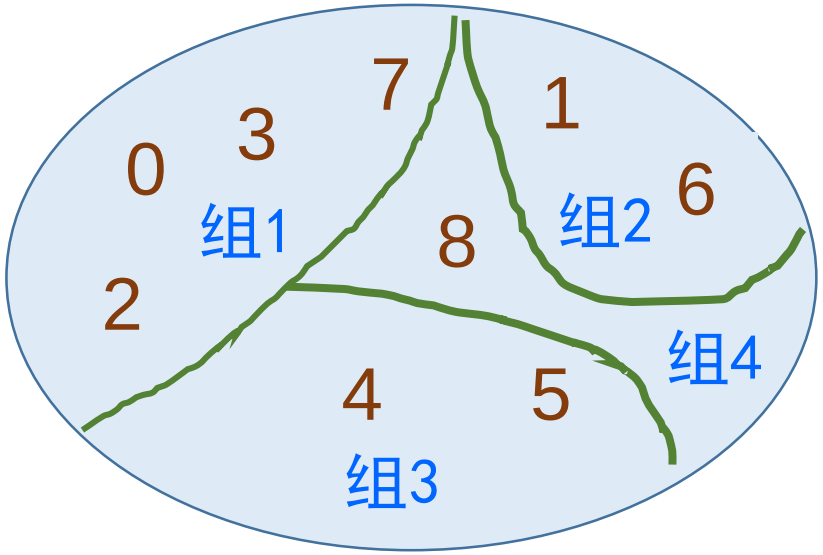
```
int s[9] = {0,1,2,3,4,5,6,7,8};
```

	0	1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0	0	0
1	1	0	0	0	1	1	0	1	1
2	0	0	0	0	0	1	1	0	0
3	0	0	0	0	1	0	0	0	1
4	0	1	0	1	0	0	1	0	1
5	1	1	1	0	0	0	1	0	0
6	0	0	1	0	1	1	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	1	0	1	1	0	0	0	0

算法基本思想:

P = {0,1,2,3,4,5,6,7,8}

```
组号 = 0;
全体元素入队列;
while ( 队列不空 ) {
      队头元素i出队列入组;
      while(j<QueueLen){
      队头元素j出队列
      if ( j和组中元素都不冲突 )  {
          j入组
      }
      else   {j重新入队列;}
     }//while
      组号++;
}//while
```



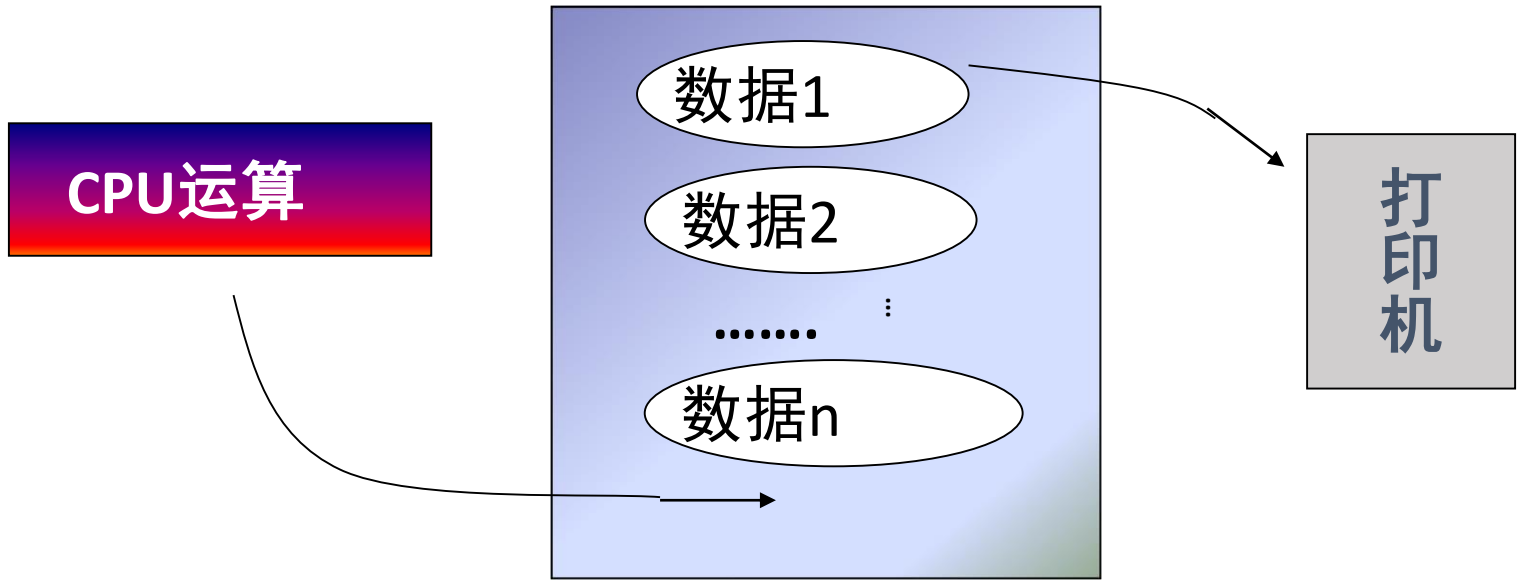
R = { (1, 4), (4, 8),
(1, 8), (1, 7), (8, 3),
(1, 0), (0, 5), (1, 5),
(3, 4), (5, 6), (5, 2),
(6, 2), (6, 4) }

一般性“划分子集”问题

将集合A划分成k个互不相交的子集 A_1，A_2，…，A_k（$k \leqslant n$），使同一子集中的元素均无冲突关系，并要求划分的子集数目尽可能地少。

队列的其它应用

应用三、解决主机与外部设备之间的速度不匹配问题



队列总结

队列特点

- ✓先进先出
- ✓操作受限的线性表
  - –在队头进行删除
  - –在队尾进行插入

队列实现

- ✓顺序存储实现——循环队列
- ✓链式存储实现——链队列

队列应用

- ✓任何排队问题
- ✓划分子集问题
- ✓打印机缓冲

作业

1. 模拟银行排队问题

2. 运动会赛事安排（选作）