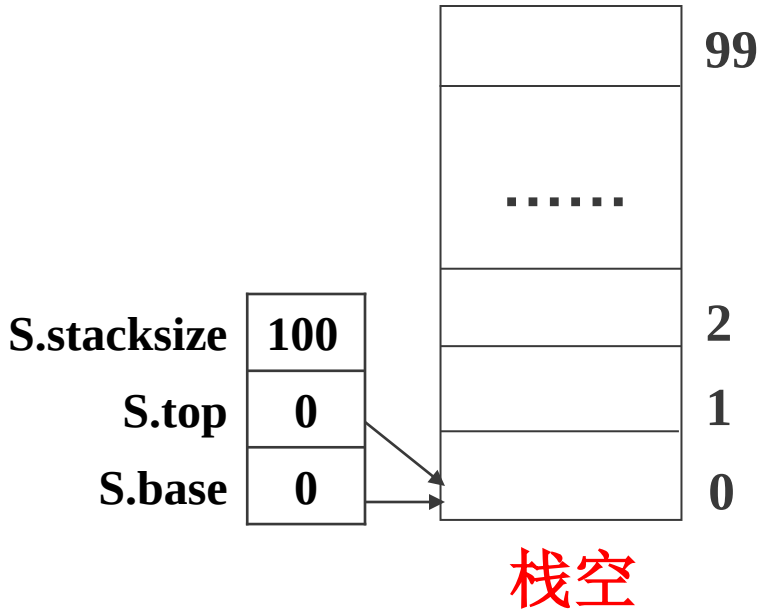
InitStack (SqStack &S)

- 参数：S是存放栈的结构变量
- 功能：建一个空栈S

```
#define STACK_INIT_SIZE 100
#define STACKINCREMENT 10
typedef struct {
  SElemType * base; //栈底
  SElemType * top; //栈顶
 int stacksize;  //栈的大小
} SqStack;
```



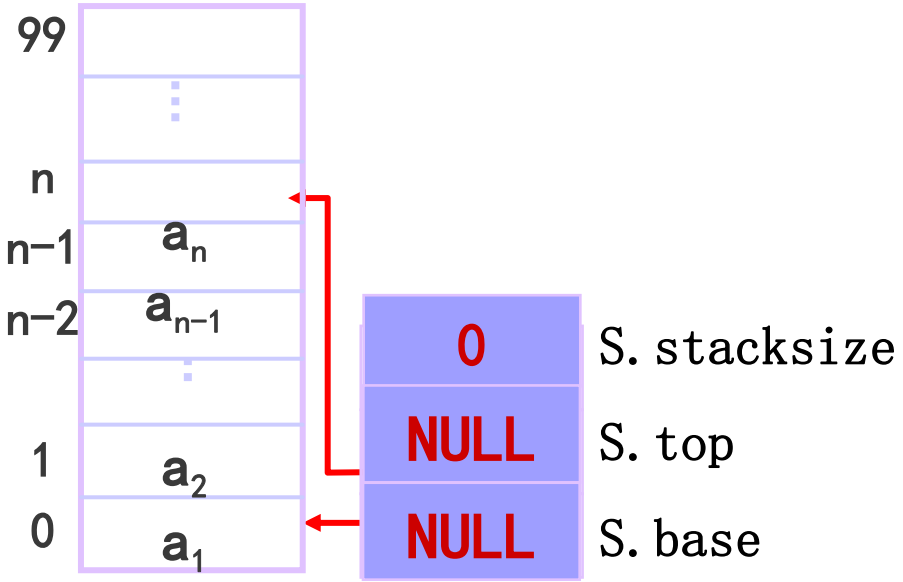
InitStack (SqStack &S)

```
Status InitStack (SqStack &S)
{   // 构造一个空栈S
    S.base=(SElemType*)malloc(
            STACK_INIT_SIZE*sizeof(ElemType));
    if (!S.base) exit (OVERFLOW); //存储分配失败
    S.top = S.base;
    S.stacksize = STACK_INIT_SIZE;
    return OK;

}// InitStack
```

DestroyStack(SqStack &S)

2） 销毁栈操作 DestroyStack(SqStack &S)

功能：销毁一个已存在的栈



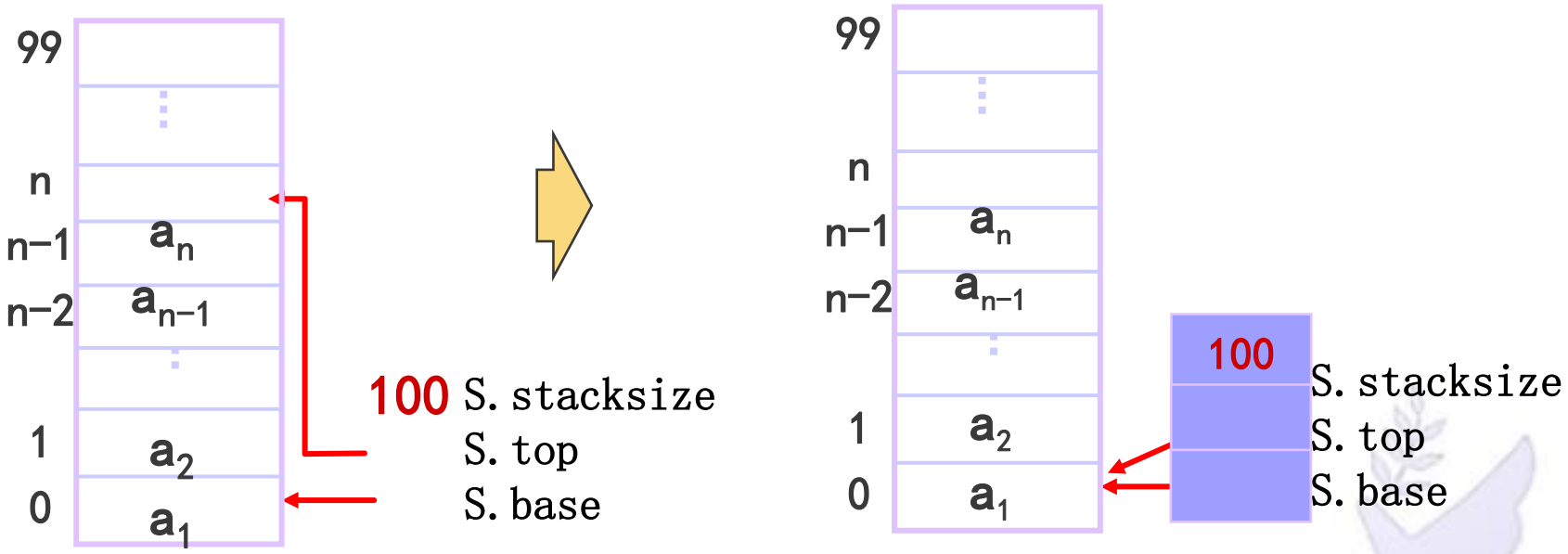
DestroyStack(SqStack &S)

```
Status DetroyStack ( SqStack  &S ) {
    //销毁操作算法
     if ( ! S.base )
      return ERROR; //若栈未建立(尚未分配栈空间)
    free ( S.base );                    //回收栈空间
    S.base = S.top = NULL;
    S.stacksize = 0;
    return OK;
}  //DetroyStack
```

ClearStack (SqStack &S)

3）置空栈操作ClearStack (SqStack &S)

功能：将栈S置为空栈



ClearStack (SqStack &S)

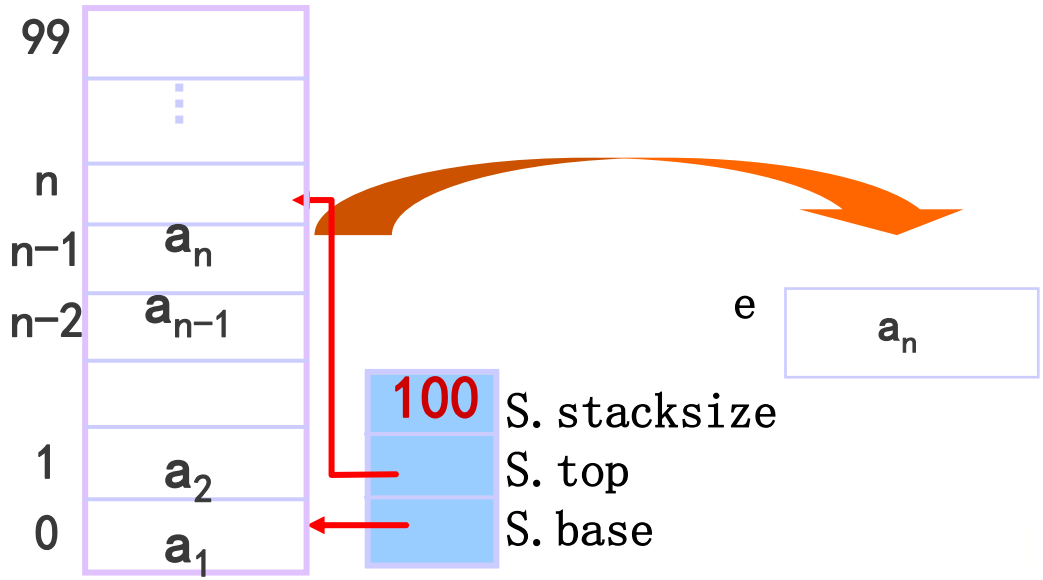
```
Status ClearStack ( SqStack &S ) {
    //置空操作算法
    if ( ! S.base )
     return ERROR; // 若栈未建立(尚未分配栈空间)

    S.top = S.base;
    return OK;
} //ClearStack
```

GetTop (SqStack S, ElemType &e)

4）取栈顶元素操作

GetTop (SqStack S, ElemType &e)

功能：取栈顶元素，并用e返回

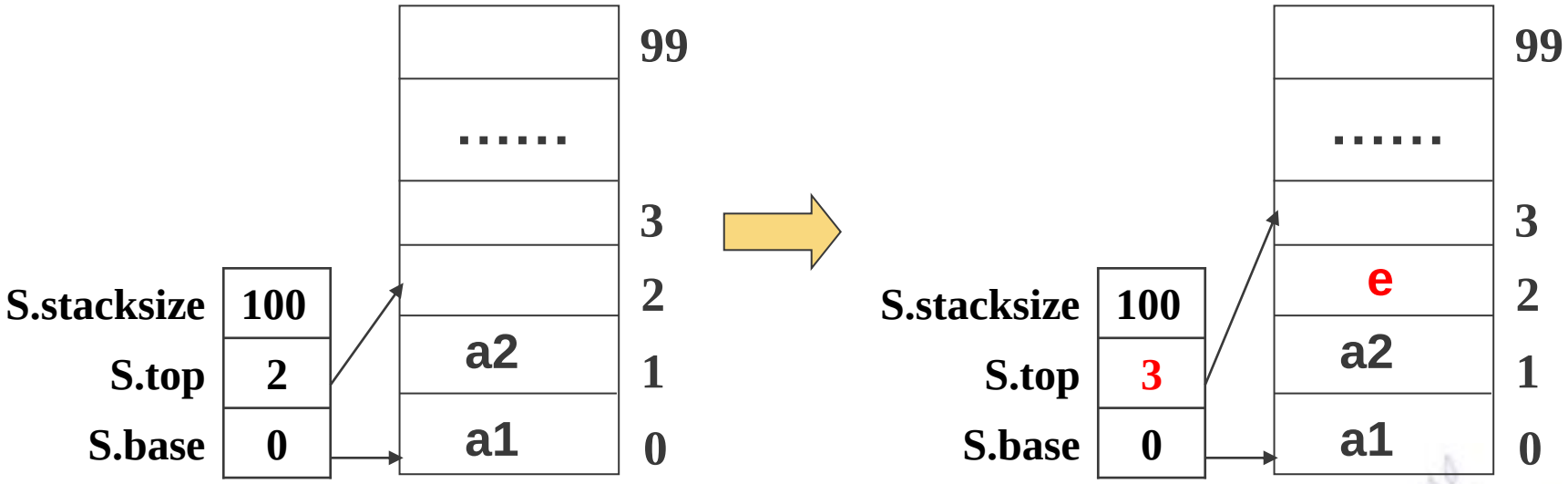


GetTop (SqStack S, ElemType &e)

```
Status GetTop ( SqStack S, ElemType &e )
{ //取栈顶元素操作算法
    if ( S.top==S.base )  return ERROR;      //栈空
    e = *(S.top-1);
    return OK;
}  //GetTop
```

Push (SqStack &S, SElemType e)

- 功能：元素 e 进栈

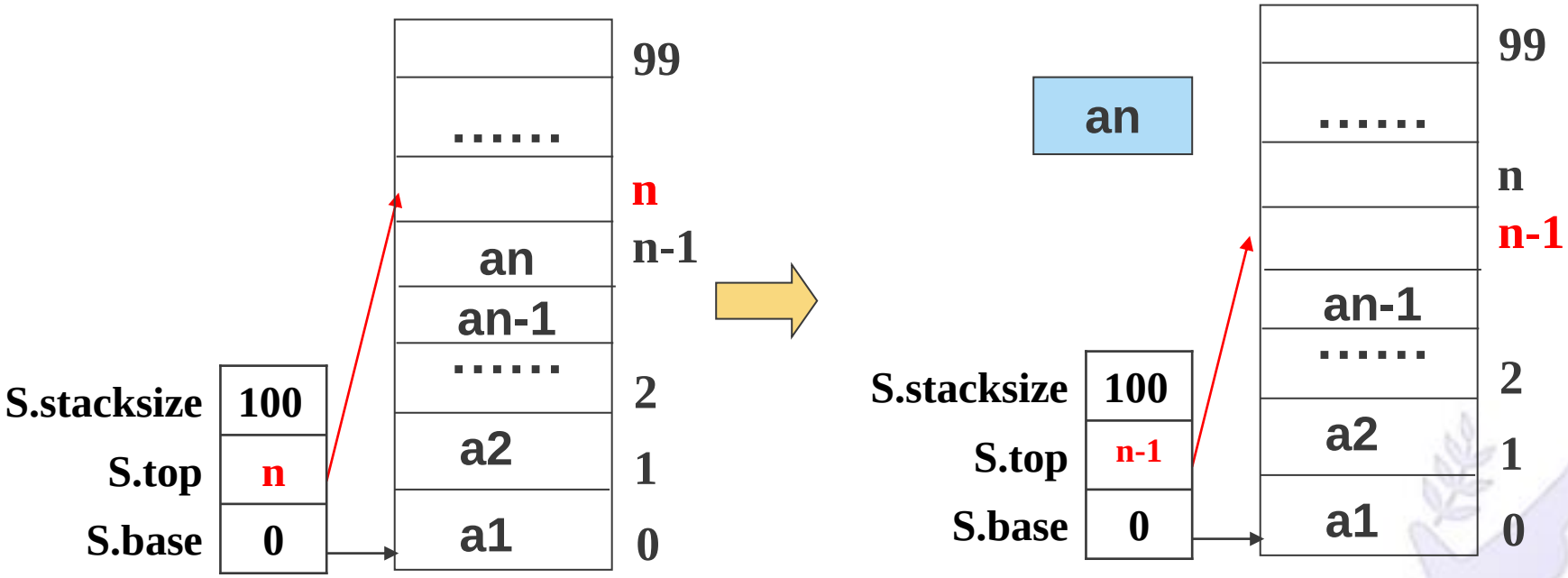



```
* S.top = e;
S.top ++;
```

```
Status Push (SqStack &S, SElemType e) {
    //将元素e插入栈中，使其成为新的栈顶元素
if (S.top - S.base >= S.stacksize) {//栈满, 追加存储空间
    S.base = (SElemType *) realloc ( S.base,
            (S.stacksize + STACKINCREMENT) *
                                sizeof (SElemType));
    if (!S.base) exit (OVERFLOW); //存储分配失败
    S.top = S.base + S.stacksize;
    S.stacksize += STACKINCREMENT;
  }
   *S.top++ = e;  //元素e 插入栈顶，后修改栈顶指针
   return OK;
}//Push
```

Pop (SqStack &S, SElemType &e)

- 出栈操作
- 功能：栈顶元素退栈，并用 e 返回。




```
S.top --;
e = * S.top;
```

Pop (SqStack &S, SElemType &e)

```
Status Pop (SqStack &S, SElemType &e) {
    // 若栈不空，则删除S的栈顶元素，
    // 用e返回其值，并返回OK；
    // 否则返回ERROR
  if (S.top == S.base) return ERROR; // 栈空

     e = *--S.top;       //--S.top; e=*S.top;

     return OK;
}//Pop
```

算符优先算法操作步骤

1. 初始化运算符OPTR栈和操作数OPND栈
2. 将‘#’压入OPTR栈
3. 依次读入每个字符，直到表达式求值完毕
   - ¶ 若是操作数，则压入OPND栈
   - ¶ 若是运算符，则和OPTR栈顶元素比较优先级
4. 返回运算结果

1. InitStack(OPTR); InitStack(OPND);
2. Push (OPTR, #); c=getchar();
3. while(c!=‘#’ || GetTop(OPTR)!=‘#’)
   - if (!In (c, OP)) {Push(OPND, c); c=getchar();}
   - else{ switch (Precede(GetTop(OPTR), c) {…}}
4. return GetTop(OPND);

算符优先算法操作步骤（续）

- switch (Precede(GetTop(OPTR), c)
  - case <: //栈顶元素优先级低,压栈并接收下一字符
    Push(OPTR, c); c = getchar();
  - case =: // 脱括号并接收下一字符
    Pop(OPTR, x); c = getchar();
  - case >: //退栈，并将运算结果压栈
    Pop(OPTR, theta);//取出运算符
    Pop (OPND, b); Pop(OPND, a); //取出操作数
    Push(OPND, Operate(a, theta, b)); //结果压栈

算符优先算法

```
OperandType EvaluateExpression( )
{ //算术表达式求值的算符优先算法。设OPTR和OPND分别为运算符栈和操作数栈，OP为运算符集合。
    InitStack(OPTR); InitStack(OPND);    //步骤1
    Push (OPTR, #); c=getchar( );        //步骤2
    while(c!=‘ #’ || GetTop(OPTR)!=‘#’){    //步骤3
      if (!In (c, OP))                   //操作数进栈OPND
        { Push(OPND,  c); c=getchar( ); }
      else
        {
```

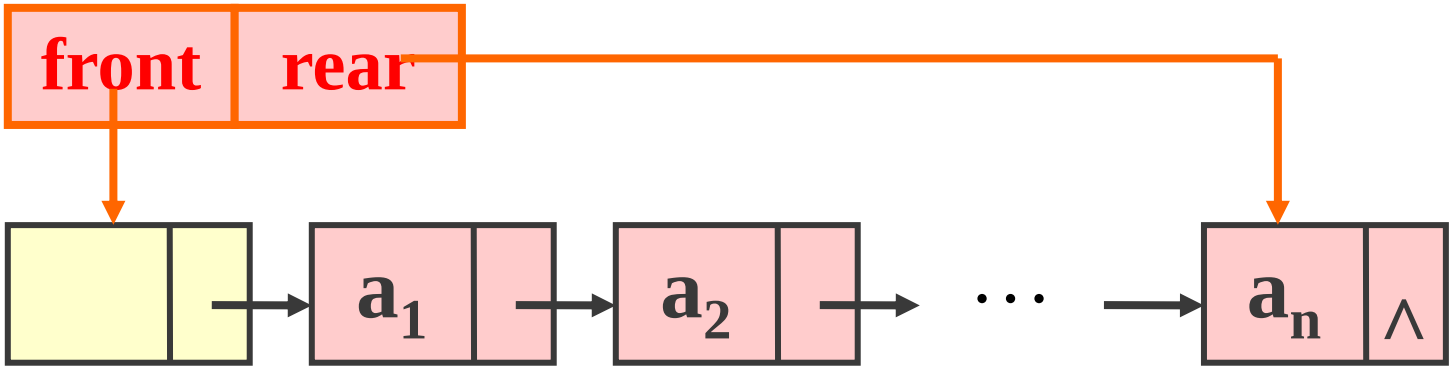
算符优先算法（续）

```
        switch (Precede(GetTop(OPTR), c)
    {   case <:                    //栈顶元素优先级低
          Push(OPTR, c);   c=getchar( );   break;
        case =:                    // 脱括号并接收下一字符
          Pop(OPTR, x);  c=getchar( ); break;
        case >:                    //退栈，并将运算结果压栈
          Pop(OPTR, theta);
          Pop (OPND, b);    Pop(OPND, a);
          Push(OPND, Operate(a, theta, b));
          break;
      }// switch
    }// if (!In (c, OP))
  }//while
  return GetTop(OPND);             //步骤4
} //EvaluateExpression
```

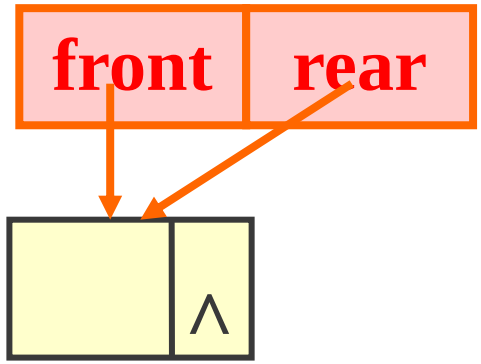
Hanoi Problem

```
void hanoi (int n, char x, char y, char z) {

   if (n==1)
        move(x, 1, z);          // 将编号为 1 的圆盘从x移到z
   else {
       hanoi(n-1, x, z, y);     // 将x上编号为 1 至n-1的
                                //圆盘移到y, z作辅助轴
       move(x, n, z);           // 将编号为n的圆盘从x移到z
       hanoi(n-1, y, x, z);     // 将y上编号为 1 至n-1的
                                //圆盘移到z, x作辅助轴
    }
}
```

1）链队列——链式映象



空队列



1）链队列——链式映象

结点类型

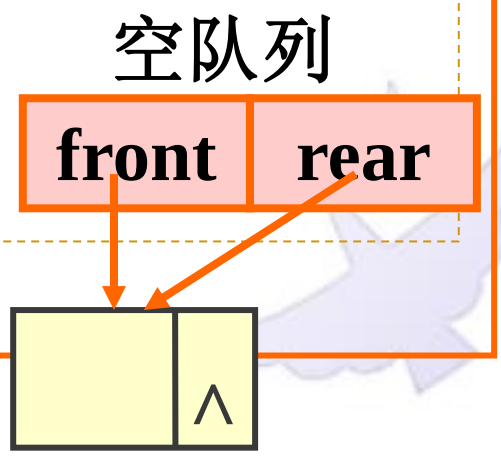
```
typedef struct QNode {
        QElemType      data;
        struct QNode  *next;
 } QNode, *QueuePtr;
```

链队列类型

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

1）链队列——链式映象

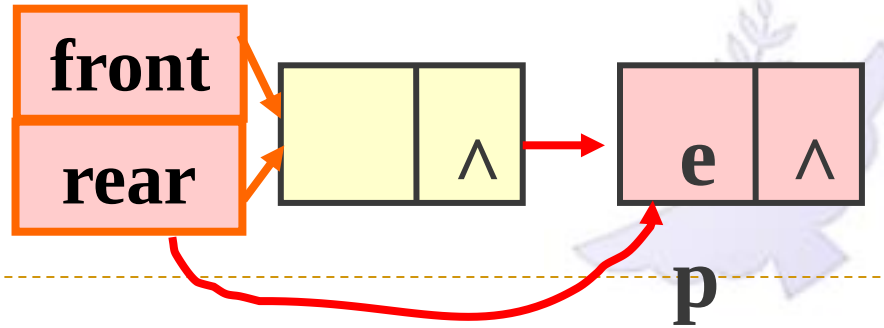
```
Status InitQueue (LinkQueue &Q) {
  // 构造一个空队列Q
    Q.front = (QueuePtr) malloc(sizeof(QNode));
    if (!Q.front) exit (OVERFLOW);
                                  //存储分配失败
    Q.rear = Q.front;
    Q.front->next = NULL;
    return OK;
}// InitQueue
```



1）带头结点的链队列——链式映象

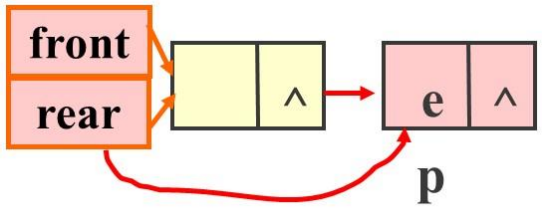
```
Status EnQueue (LinkQueue &Q, QElemType e) {
  // 插入元素e为Q的新的队尾元素
    p = (QueuePtr) malloc (sizeof (QNode));
    if (!p)  exit (OVERFLOW);   //存储分配失败
    p->data = e;
    p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
    return OK;
}// EnQueue
```

初始空队列　插入队尾e

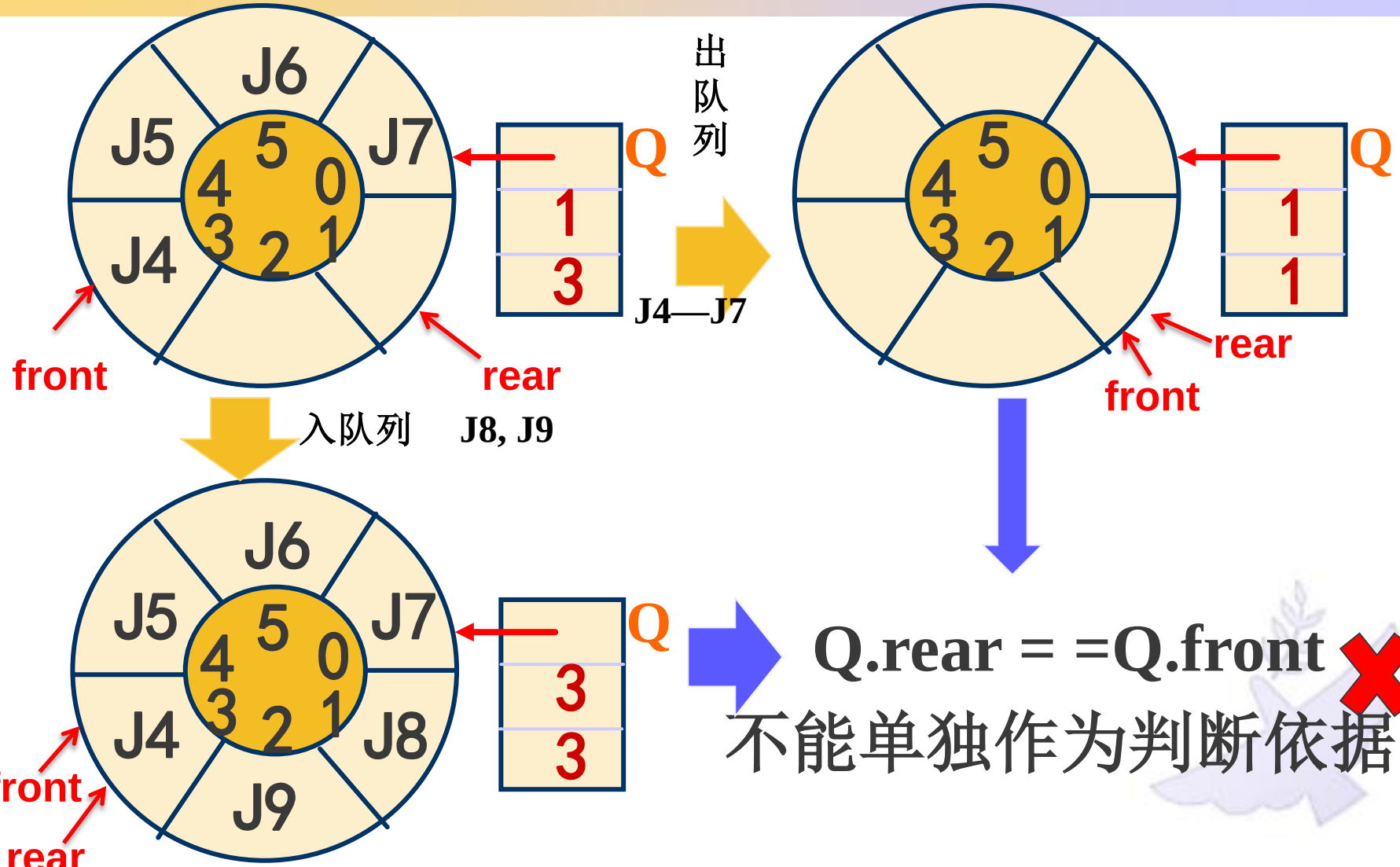


1）带头结点的链队列——链式映象

```
Status DeQueue (LinkQueue &Q, QElemType &e) {
 // 若队列不空，则删除Q的队头元素，
 //用 e 返回其值，并返回OK；否则返回ERROR
   if (Q.front == Q.rear)    return ERROR;
   p = Q.front->next;   e = p->data;
   Q.front->next = p->next;
   if (Q.rear == p)  Q.rear = Q.front;   //修改rear
   free (p);
   return OK;
}// DeQueue
```



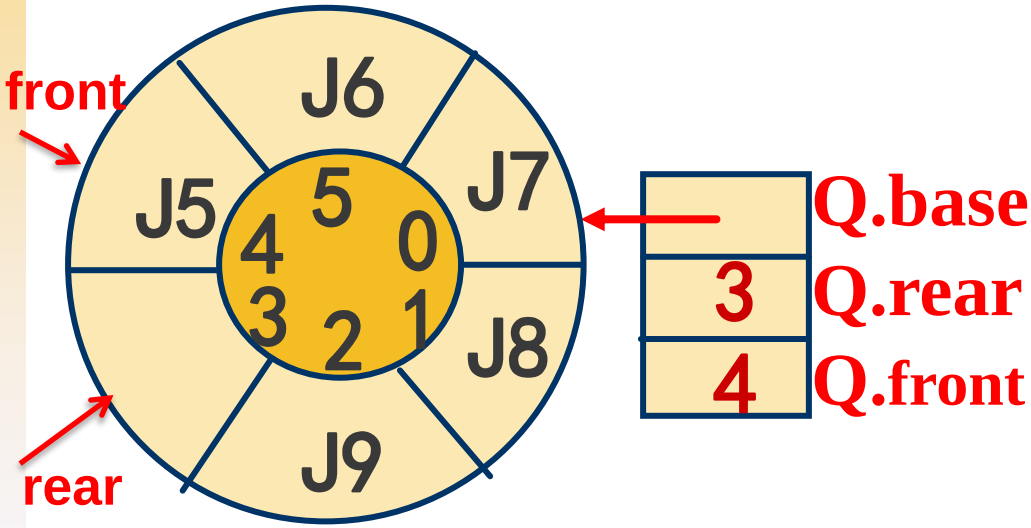
循环队列判空、判满的方法及条件

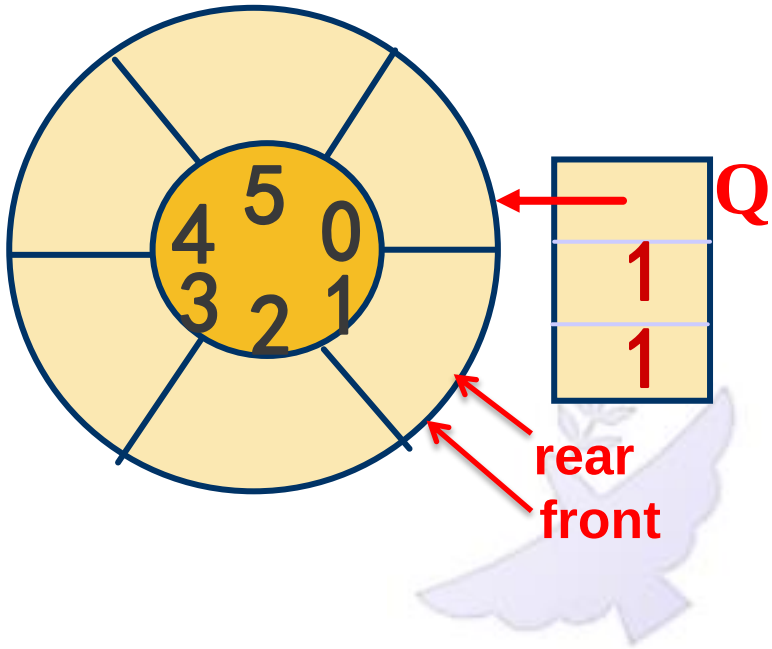


◆一种改进的方法：少用一个存储单元

判满条件(Q.rear+1)%MAXQSIZE = =Q.front

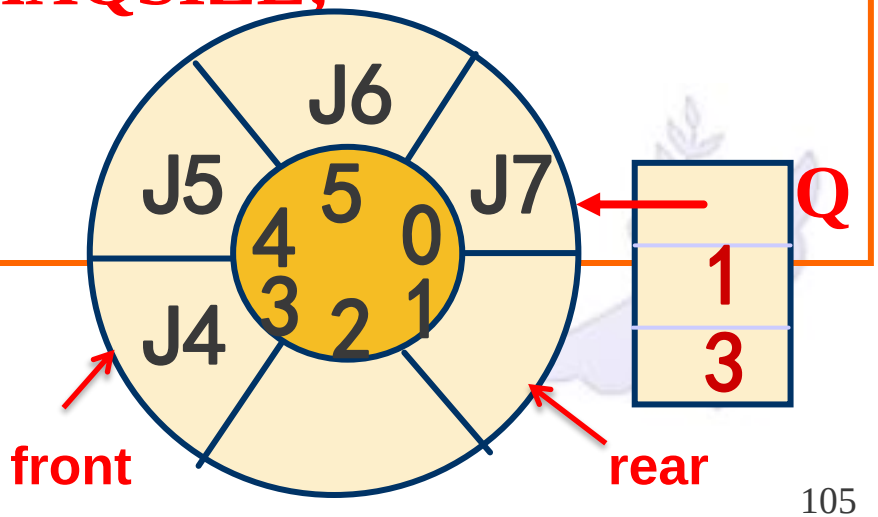
判空条件 Q.rear = =Q.front





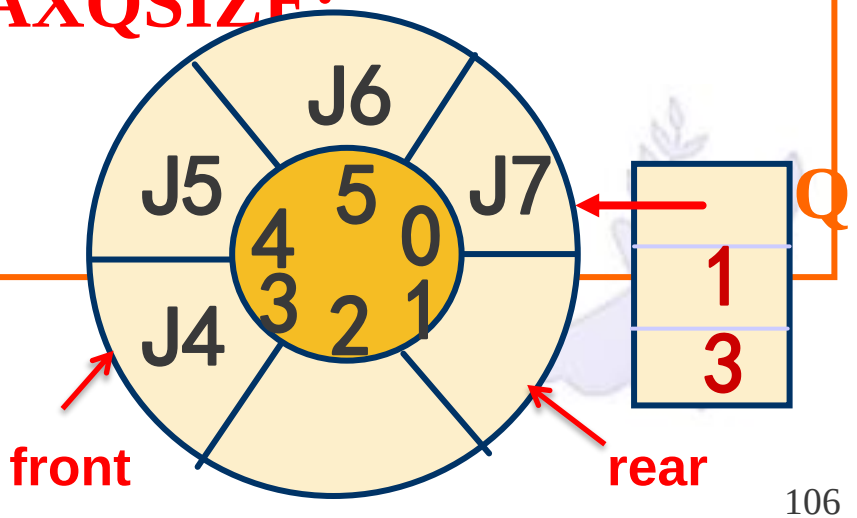
◆一种改进的方法：少用一个存储单元

```
Status EnQueue (SqQueue &Q, ElemType e) {
        // 插入元素e为Q的新的队尾元素
    if ((Q.rear+1) % MAXQSIZE == Q.front)
        return ERROR; //队列满
    Q.base[Q.rear] = e;
    Q.rear = (Q.rear+1) % MAXQSIZE;
    return OK;
}// EnQueue
```



◆一种改进的方法：少用一个存储单元

```
Status DeQueue (SqQueue &Q, ElemType &e) {
  // 若队列不空，则删除Q的队头元素，
  // 用e返回其值，并返回OK；否则返回ERROR
   if (Q.front == Q.rear)  return ERROR; //队列空
   e = Q.base[Q.front];
   Q.front = (Q.front+1) % MAXQSIZE;
   return OK;
}// DeQueue
```



注意

- 在循环队列中为何不能用动态数组？
- 循环队列必须要设置最大长度！