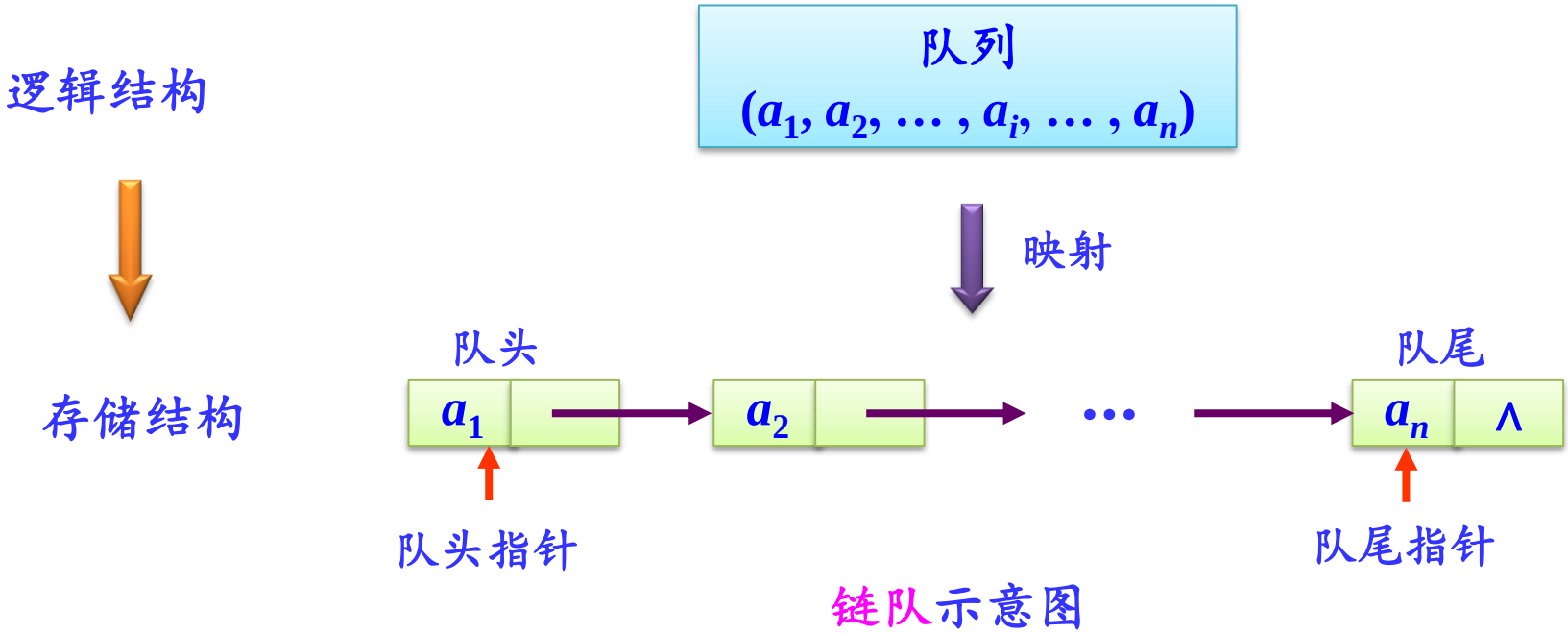
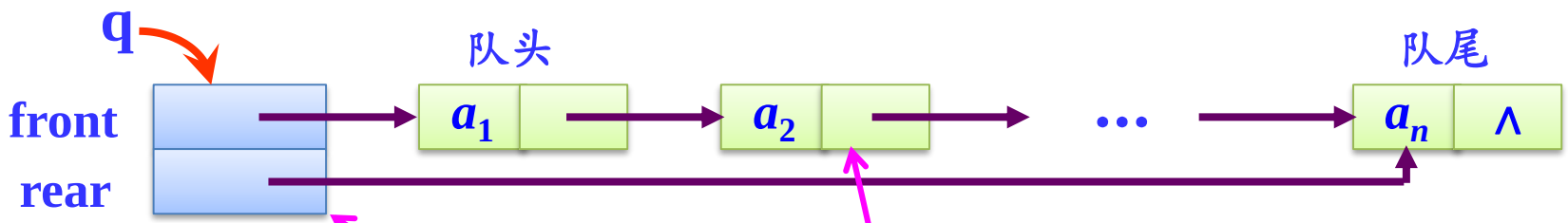
3.2.3 队列的链式存储结构及其基本运算的实现

采用链表存储的队列称为链队，这里采用不带头节点的单链表实现。



链队示意图

通常将队头和队尾两个指针合起来：

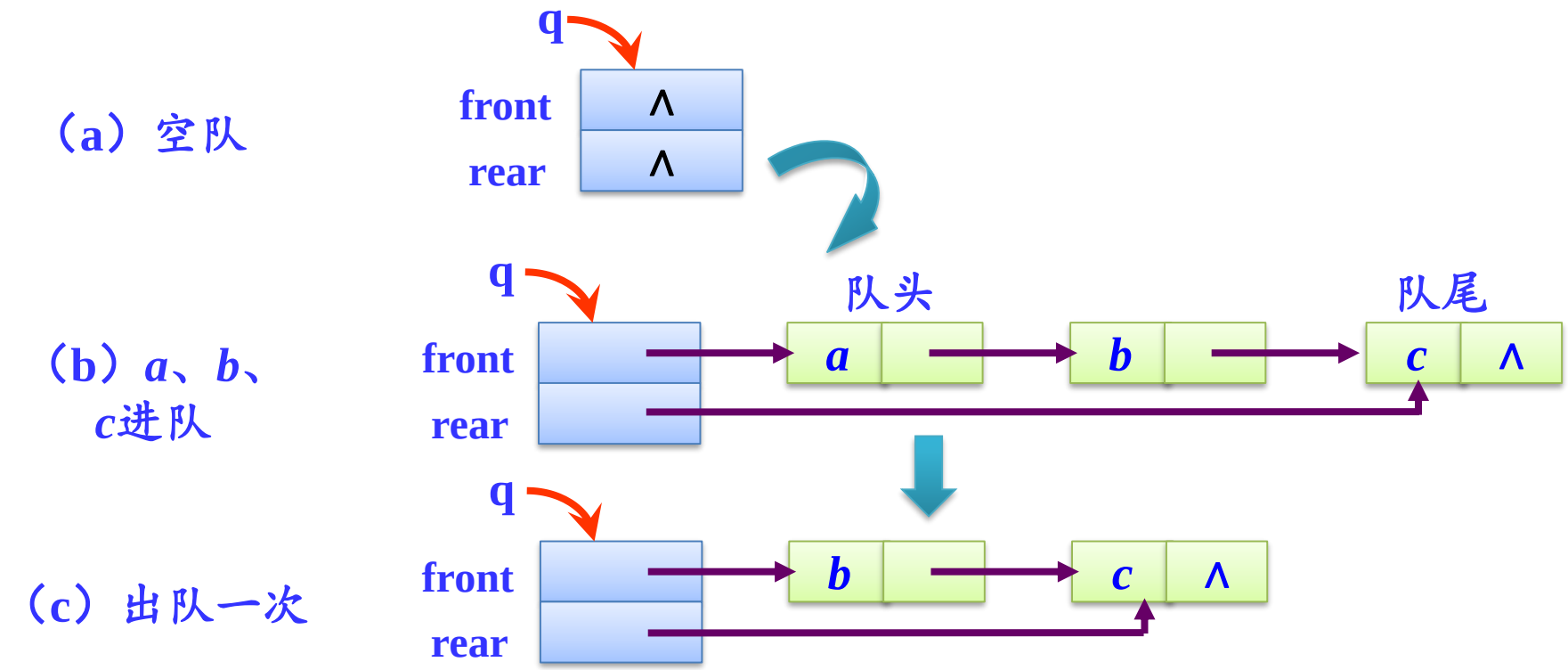


链队组成：

（1）存储队列元素的单链表节点

（2） 指向队头和队尾指针的链队头节点

链队的进队和出队操作演示

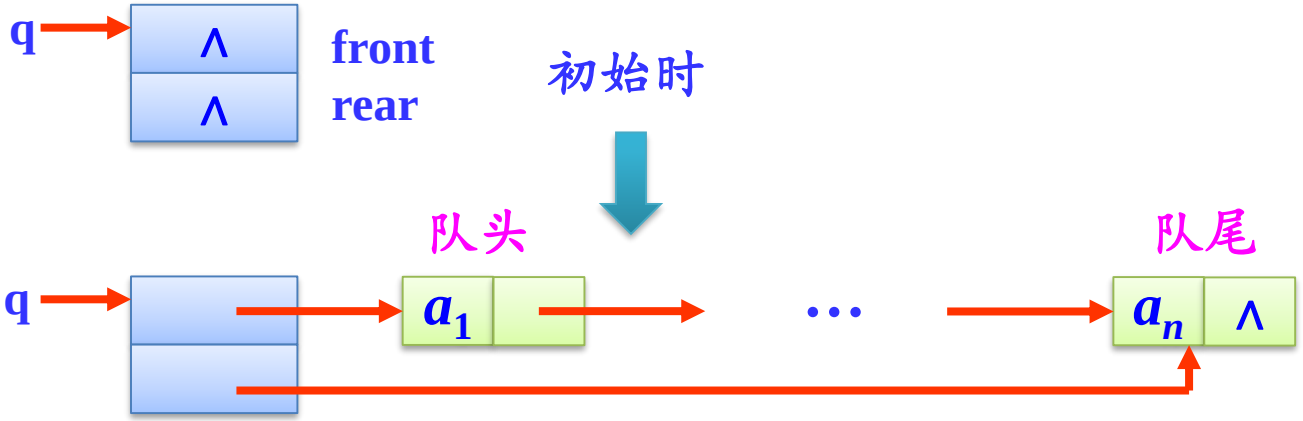


单链表中数据节点类型QNode定义如下：

```
typedef struct qnode
{     ElemType data;        //数据元素
      struct qnode *next;
} QNode;
```

链队中头节点类型LiQueue定义如下：

```
typedef struct
{     QNode *front;         //指向单链表队头节点
      QNode *rear;          //指向单链表队尾节点
} LiQueue;
```



链队的4要素：

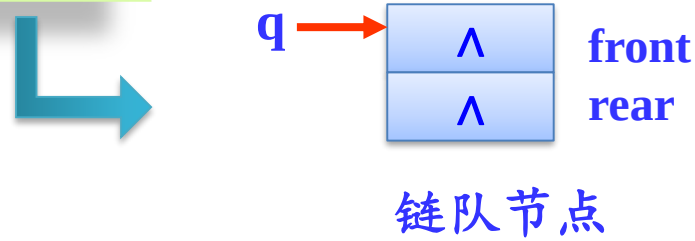
- 队空条件：front=rear=NULL
- 队满条件：不考虑
- 进队e操作：将包含e的节点插入到单链表表尾
- 出队操作：删除单链表首数据节点

在链队存储中，队列的基本运算算法如下。

（1）初始化队列InitQueue(q)

构造一个空队列，即只创建一个链队头节点，其front和rear域均置为NULL，不创建数据元素节点。

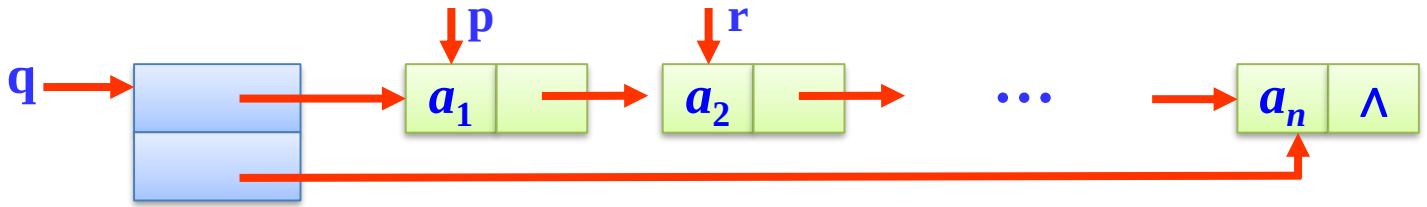
```
void InitQueue(LiQueue *&q)
{    q=(LiQueue *)malloc(sizeof(LiQueue));
     q->front=q->rear=NULL;
}
```



（2）销毁队列DestroyQueue(q)

释放队列占用的存储空间，包括链队头节点和所有数据节点的存储空间。

```
void DestroyQueue(LiQueue *&q)
{    QNode *p=q->front,  *r;              //p指向队头数据节点
     if  (p!=NULL)                        //释放数据节点占用空间
     {     r=p->next;
           while (r!=NULL)
           {    free(p);
                p=r;  r=p->next;
           }
     }
     free(p);  free(q);                   //释放链队节点占用空间
}
```



（3）判断队列是否为空QueueEmpty(q)

若链队节点的rear域值为NULL，表示队列为空，返回true；否则返回false。

```
bool QueueEmpty(LiQueue *q)
{
    return(q->rear==NULL);
}
```

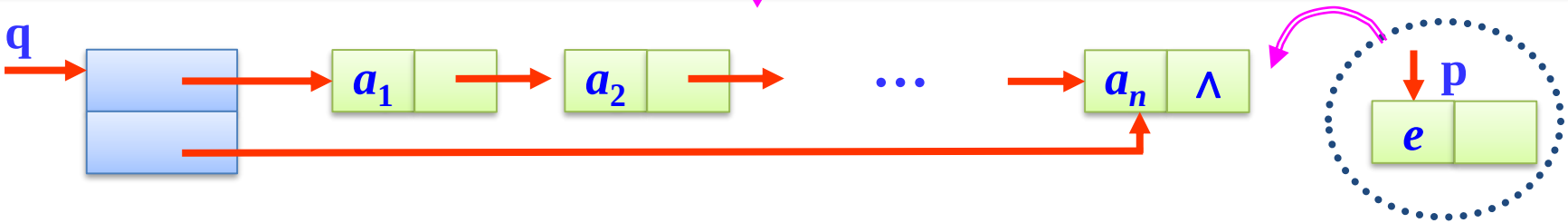
空链队

（4） 进队enQueue(q,e)

需要考虑的情况：

- 原队列为空
- 原队列非空

```
void enQueue(LiQueue *&q,ElemType e)
{     QNode *p;
      p=(QNode *)malloc(sizeof(QNode));
      p->data=e;
      p->next=NULL;
      if (q->rear==NULL)     //若链队为空,新节点是队首节点又是队尾节点
            q->front=q->rear=p;
      else
      {     q->rear->next=p;  //将*p节点链到队尾,并将rear指向它
            q->rear=p;
      }
}
```

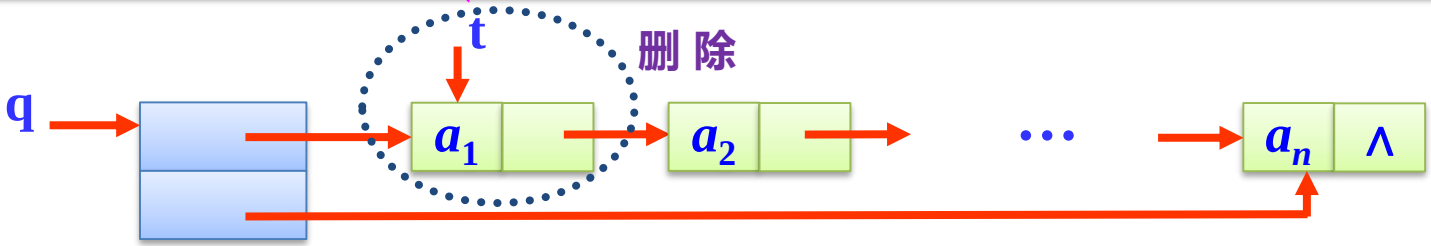


（5）出队deQueue(q,e)

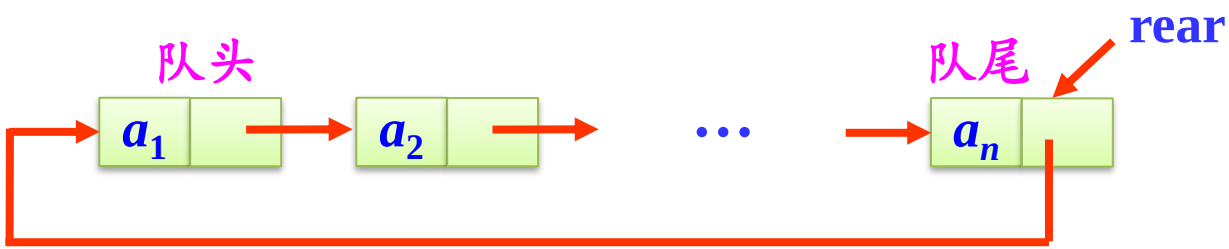
需要考虑的情况：

- 原队列为空
- 原队列只有一个节点
- 其他情况

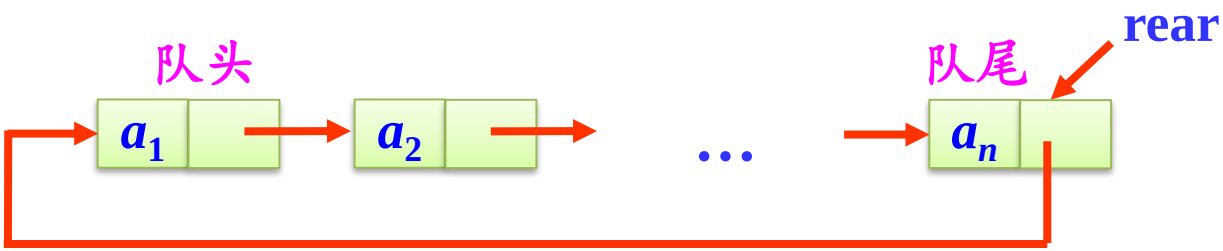
```
bool deQueue(LiQueue *&q,ElemType &e)
{    QNode *t;
     if (q->rear==NULL) return false;          //队列为空
     t=q->front;                               //t指向第一个数据节点
     if (q->front==q->rear)                    //队列中只有一个节点时
           q->front=q->rear=NULL;
     else                                      //队列中有多个节点时
           q->front=q->front->next;
     e=t->data;
     free(t);
     return true;
}
```



【例3-6】采用一个不带头节点只有一个尾节点指针rear的循环单链表存储队列，设计队列的初始化、进队和出队等算法。



这样的链队通过尾节点指针rear唯一标识。



这样的链队通过尾节点指针rear唯一标识。

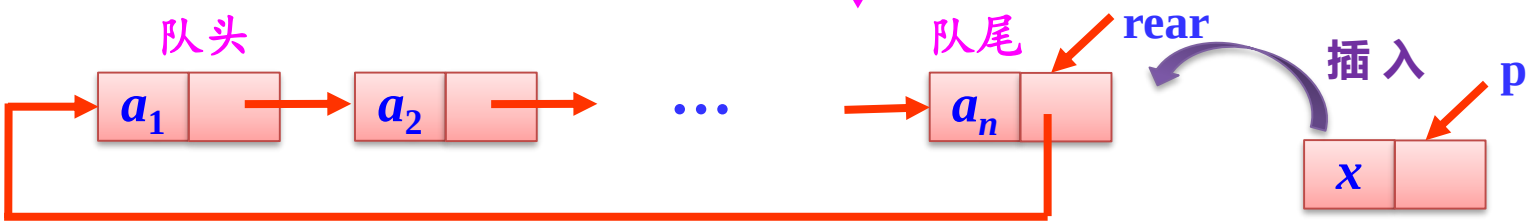
链队的4要素：

- 队空条件：rear=NULL
- 队满条件：不考虑
- 进队*e*操作：将包含*e*的节点插入到单链表表尾
- 出队操作：删除单链表首节点

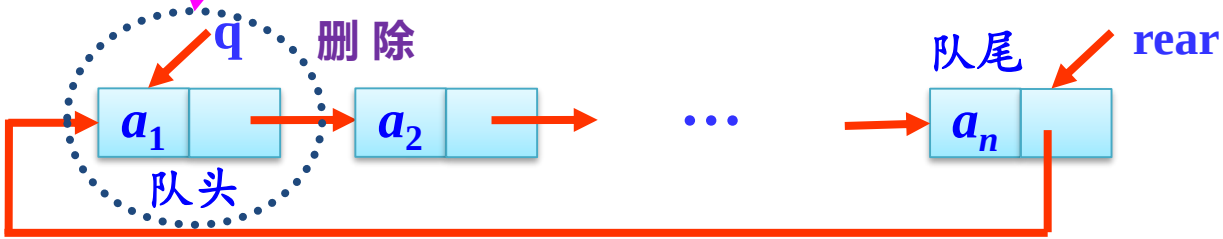
```
void initQueue(LinkList *&rear)    //初始化队运算算法
{
      rear=NULL;
}

bool queueEmpty(LinkList *rear)   //判队空运算算法
{
    return(rear==NULL);
}
```

```
void enQueue(LinkList *&rear,ElemType x)   //进队运算算法
{     LinkList *p;
      p=(LinkList *)malloc(sizeof(LinkList));    //创建新节点
      p->data=x;
      if (rear==NULL)                            //原链队为空
      {       p->next=p;                         //构成循环链表
              rear=p;
      }
      else
      {       p->next=rear->next;                //将*p节点插入到*rear节点之后
              rear->next=p;
              rear=p;                            //让rear指向这个新插入的节点
      }
}
```



```
bool deQueue(LinkList *&rear,ElemType &x)    //出队运算算法
{    LinkList *q;
     if (rear==NULL)  return false;          //队空
     else if (rear->next==rear)              //原队中只有一个节点
     {    x=rear->data;
          free(rear);    rear=NULL;
     }
     else                                    //原队中有两个或以上的节点
     {    q=rear->next;   x=q->data;
          rear->next=q->next;
          free(q);
     }
     return true;
}
```



思考题

链队和顺序队两种存储结构有什么不同？

——本讲完——