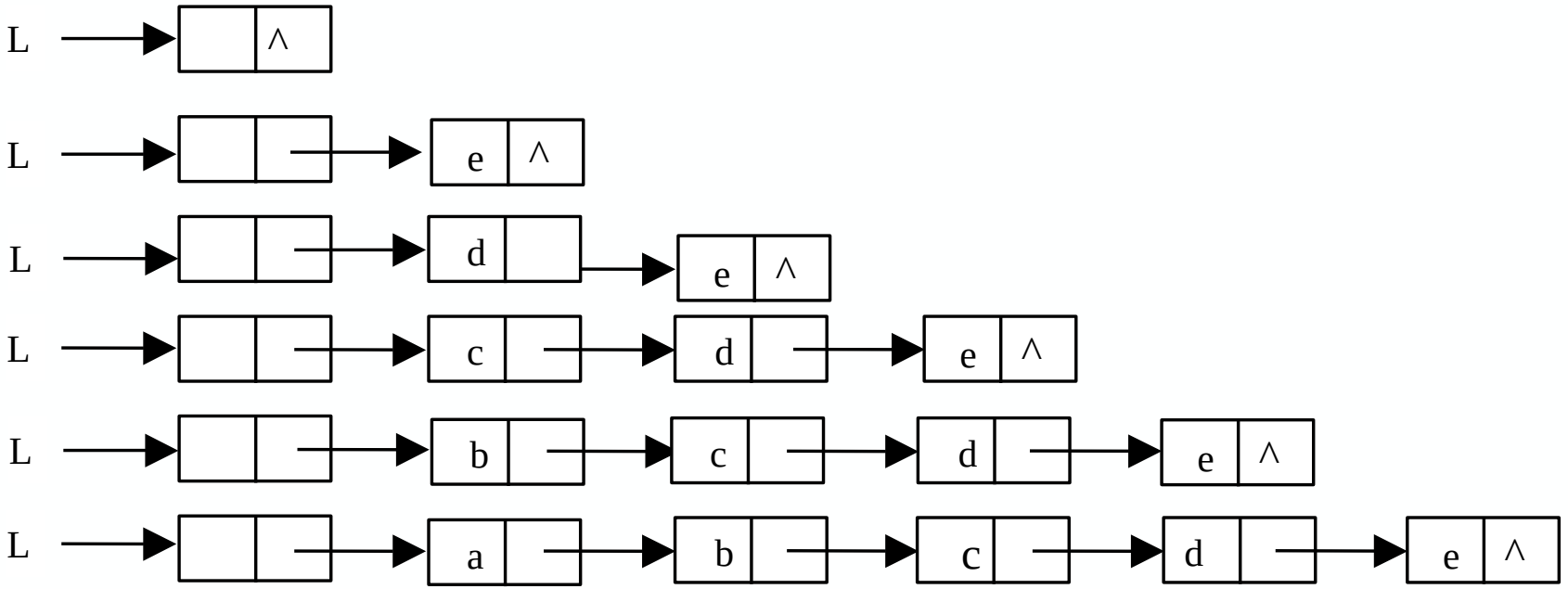
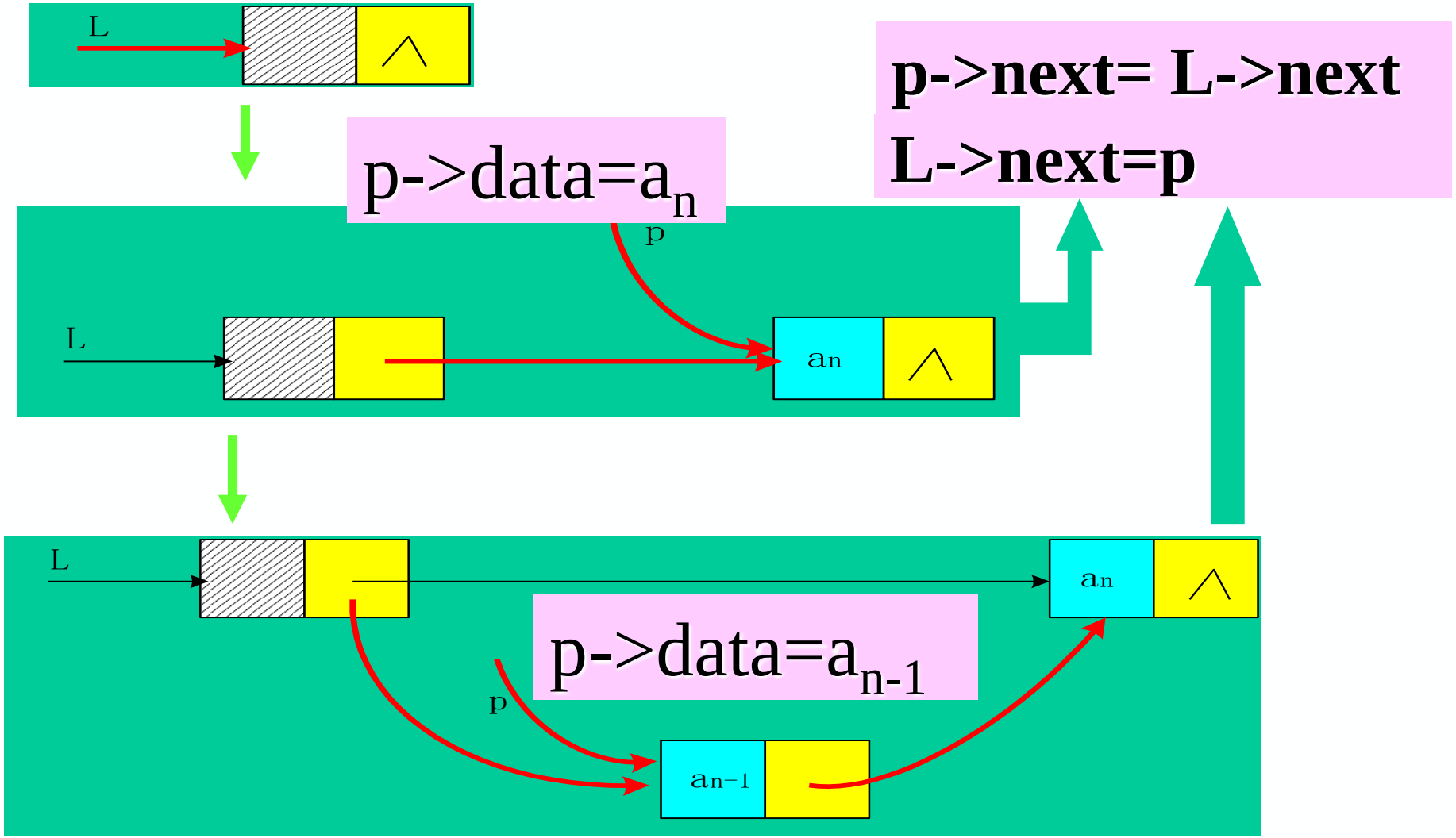
单链表的建立（前插法）

- 从一个空表开始，重复读入数据：
  - 生成新结点
  - 将读入数据存放到新结点的数据域中
  - 将该新结点插入到链表的前端



单链表的建立（前插法）

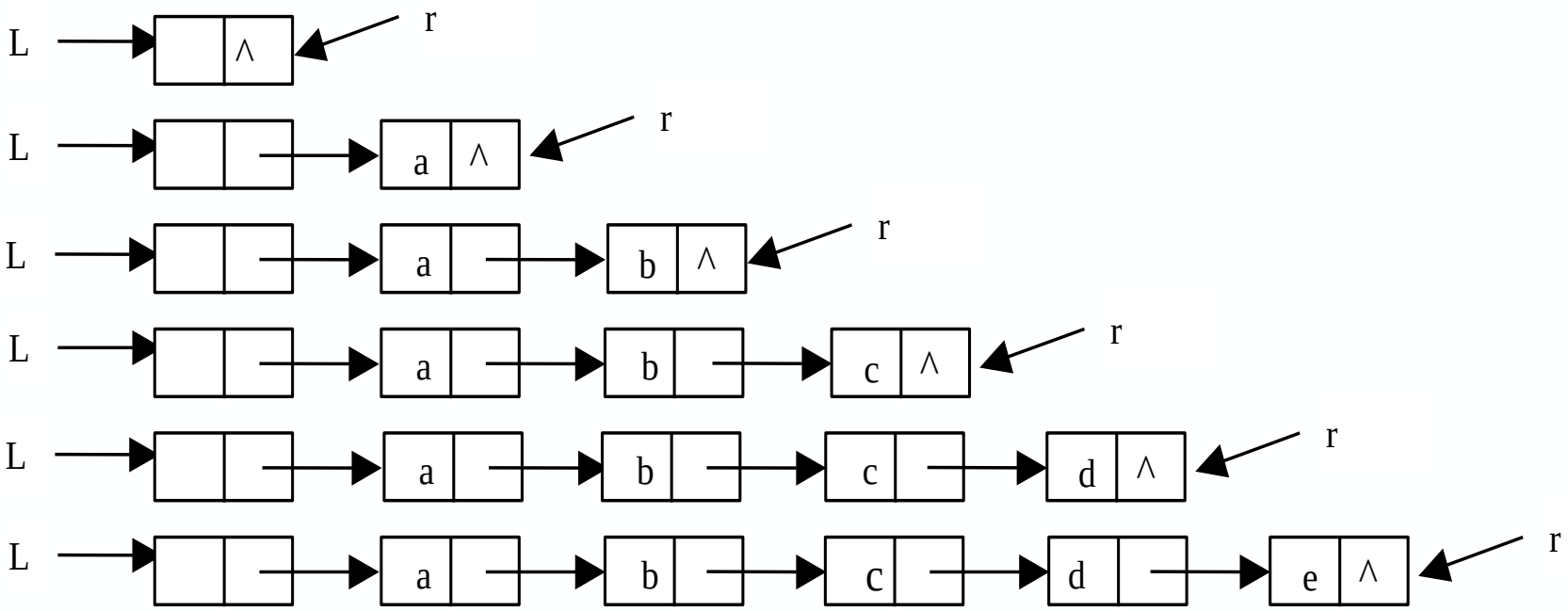


【算法描述】

```
void CreateList_F(LinkList &L,int n){
    L=new LNode;
     L->next=NULL; //先建立一个带头结点的单链表
     for(i=n;i>0;--i){
       p=new LNode; //生成新结点
       cin>>p->data; //输入元素值
       p->next=L->next;L->next=p;    //插入到表头
     }
}//CreateList_F
```

单链表的建立（尾插法）

- 从一个空表L开始，将新结点逐个插入到链表的尾部，尾指针r指向链表的尾结点。
- 初始时，r同L均指向头结点。每读入一个数据元素则申请一个新结点，将新结点插入到尾结点后，r指向新结点。

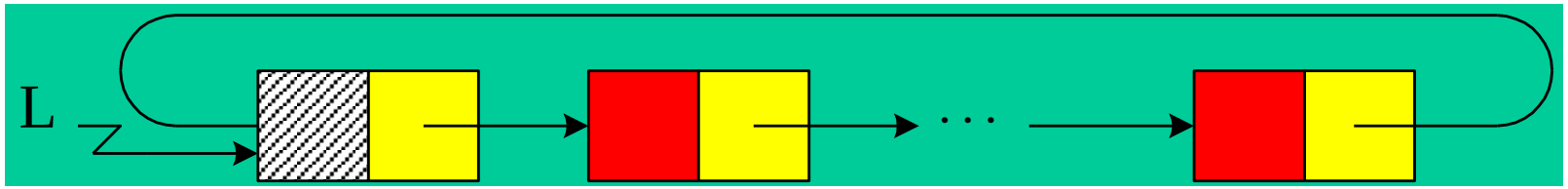


20:09

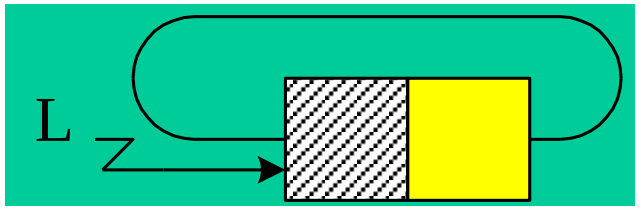
【算法描述】

```
void CreateList_L(LinkList &L,int n){
    //正位序输入n个元素的值，建立带表头结点的单链表L
    L=new LNode;
    L->next=NULL;
    r=L;          //尾指针r指向头结点
    for(i=0;i<n;++i){
      p=new LNode;              //生成新结点
      cin>>p->data;             //输入元素值
      p->next=NULL; r->next=p;                  //插入到表尾
      r=p;        //r指向新的尾结点
    }
}//CreateList_L
```

2.5.3 循环链表

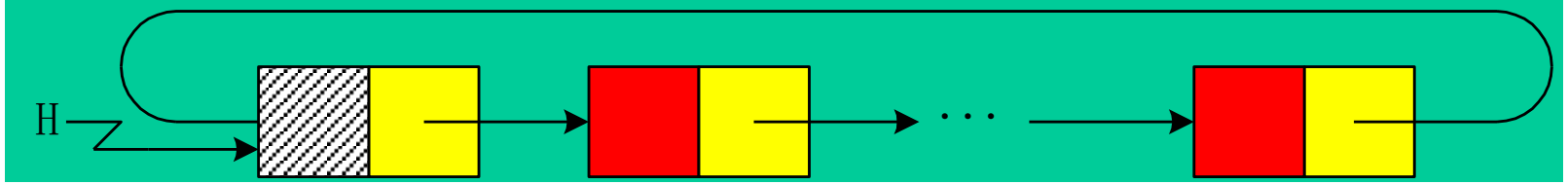


(a) 非空单循环链表



(b) 空表

L->next=L



从循环链表中的任何一个结点的位置都可以找到其他所有结点，而单链表做不到；

循环链表中没有明显的尾端

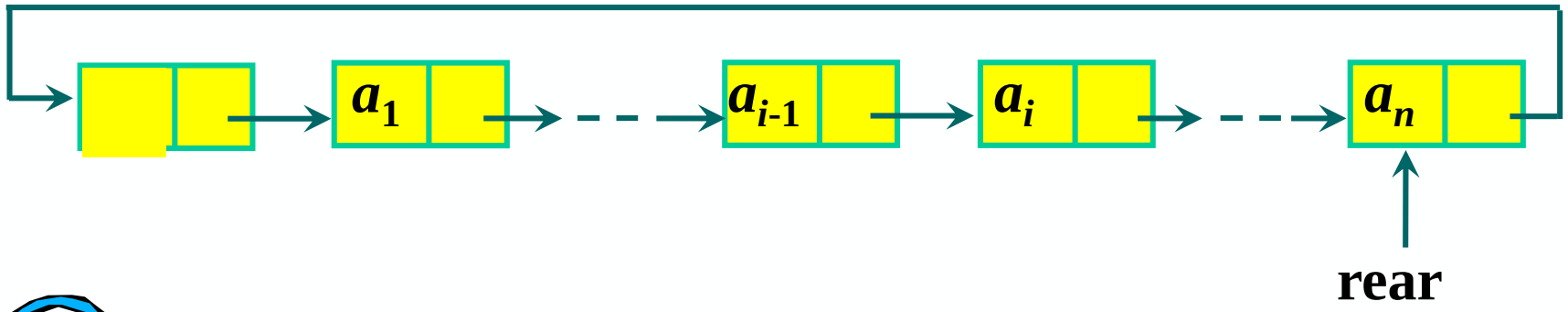
如何避免死循环

循环条件：**p!=NULL➔p!=L**

p->next!=NULL➔p->next!=L

对循环链表，有时不给出头指针，而给出尾指针

可以更方便的找到第一个和最后一个结点

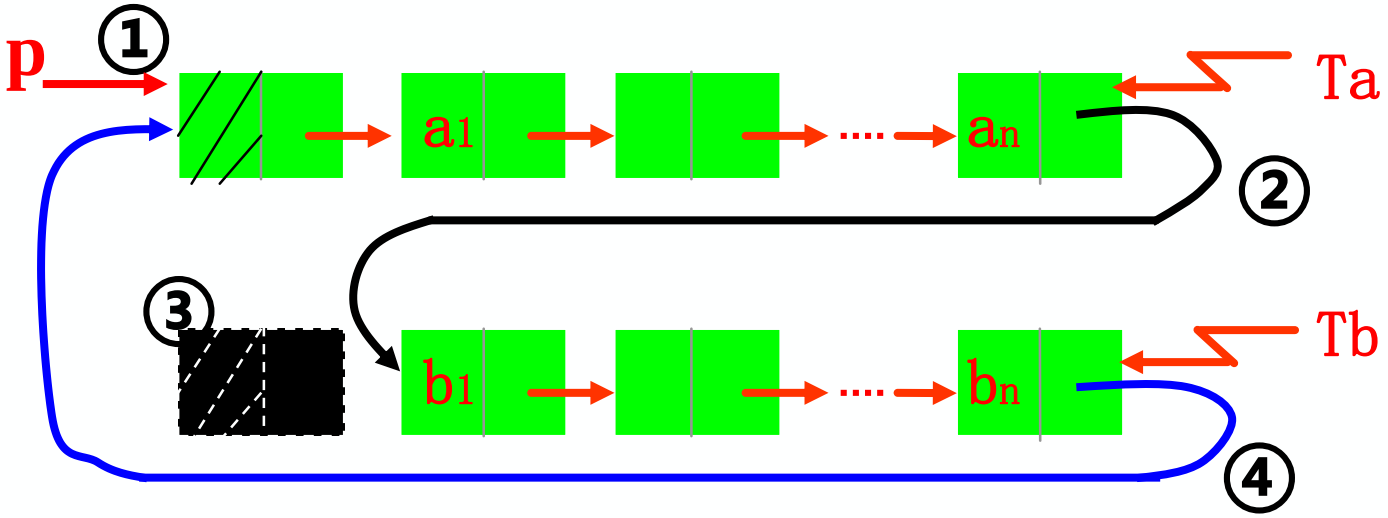


如何查找开始结点和终端结点？

开始结点： rear->next->next

终端结点： rear

循环链表的合并




```
LinkList  Connect(LinkList  Ta,LinkList  Tb)
{//假设Ta、Tb都是非空的单循环链表
        p=Ta->next;                          //①p存表头结点
        Ta->next=Tb->next->next;             //②Tb表头连结Ta表尾
        deleteTb->next;                      //③释放Tb表头结点
        Tb->next=p;                          //④修改指针
        return  Tb;
}
```

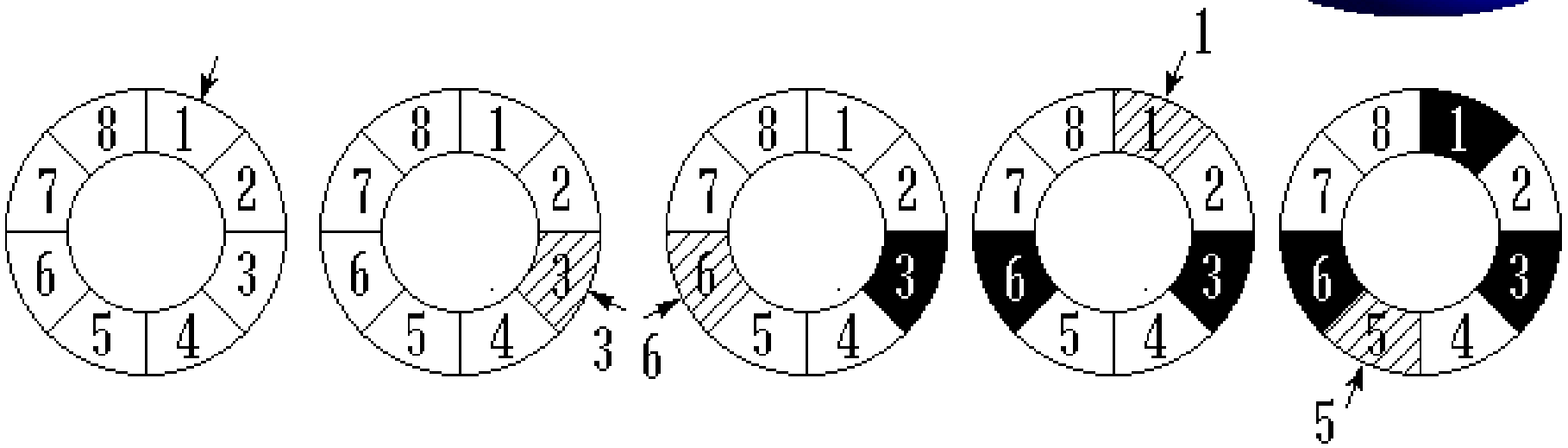
约瑟夫问题

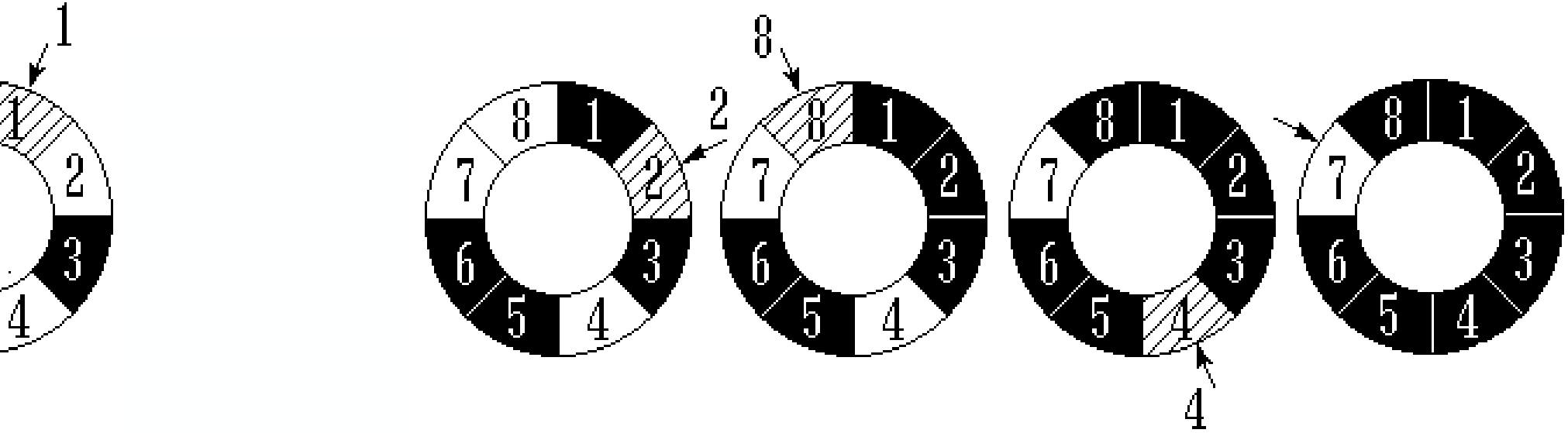
著名犹太历史学家 **Josephus**

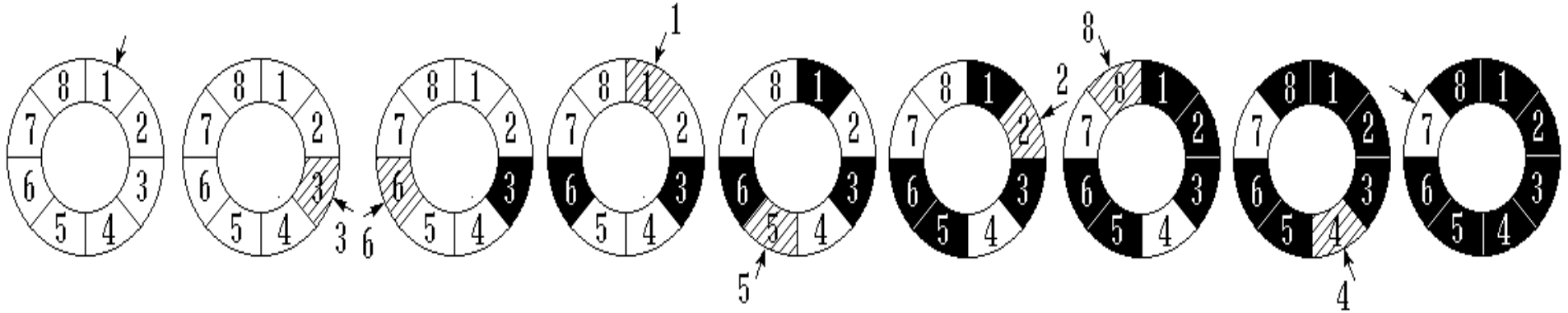
- 在罗马人占领乔塔帕特后
- 39 个犹太人与Josephus及他的朋友躲到一个洞中
- 39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式
- 41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止
- 然而Josephus 和他的朋友并不想遵从，Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏

约瑟夫问题

- 例如 $n = 8 \quad m = 3$





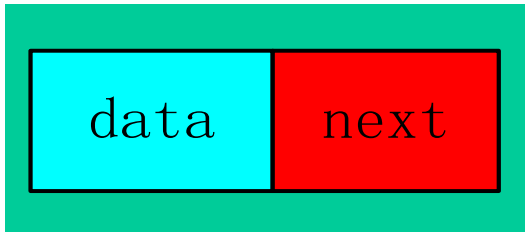


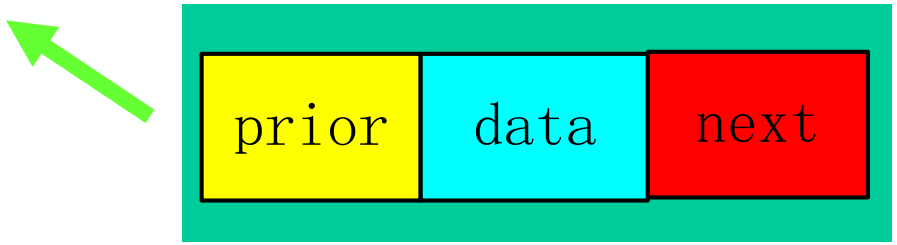
约瑟夫问题的解法

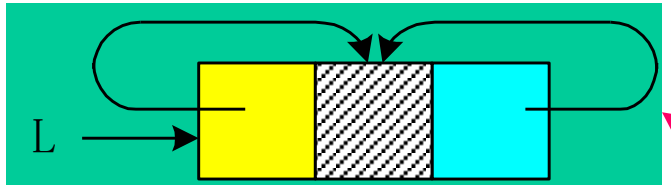
```
void Josephus ( int n, int m ) {
    Firster ( );   //检验指针指向第一个结点
    for ( int i = 0; i < n-1; i++ ) {     //执行n-1次
        for ( int j = 0; j < m-1; j++ ) Next ( );
         //循环m次使current指向被删除结点
        cout << "出列的人是"  << GetElem_L ( ) << endl;
         //出列人员的数据
        ListDelete ( );                  //删去每一趟的第m结点
    }
```

2.5.4 双向链表

```
typedef struct DuLNode{
    ElemType   data;
    struct DuLNode  *prior;
    struct DuLNode  *next;
}DuLNode, *DuLinkList
```

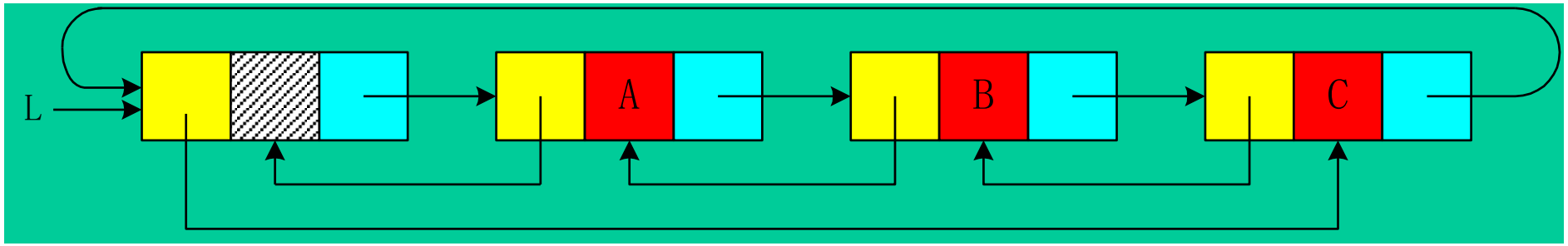






(a) 空双向循环链表

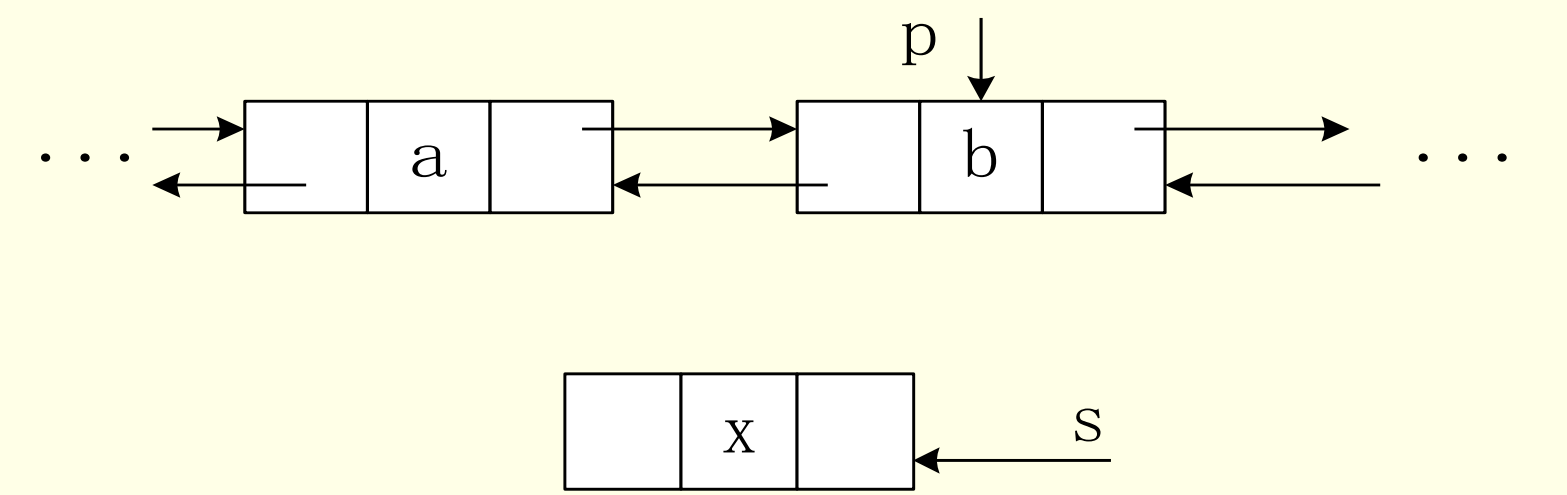
L->next=L



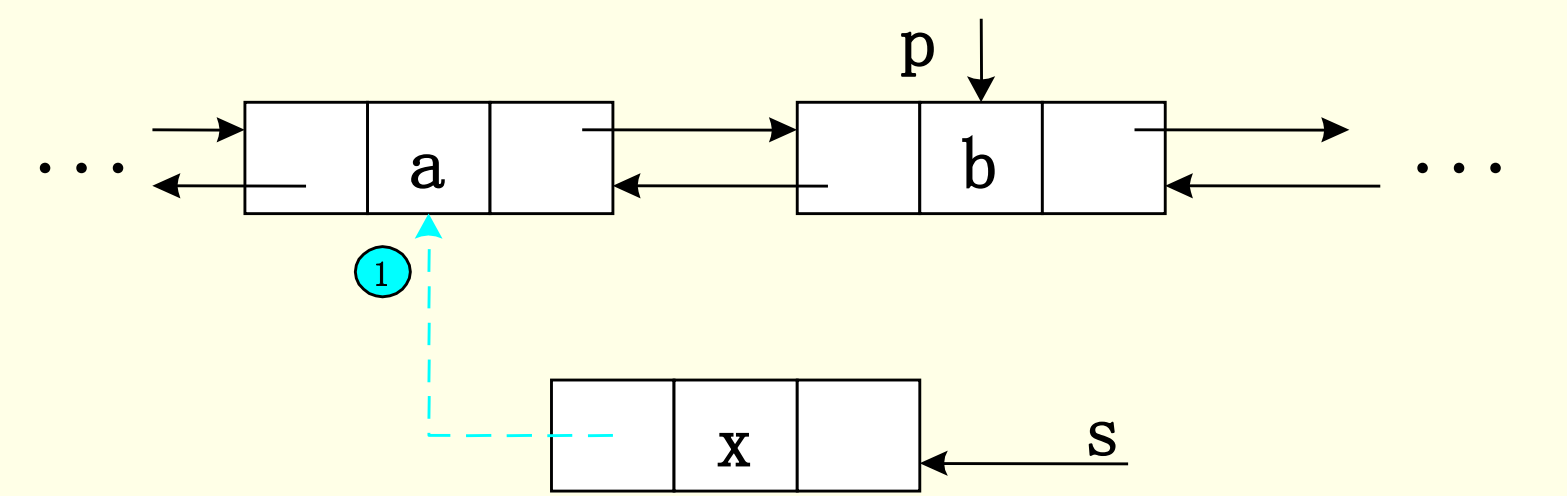
(b) 双向循环链表

d->next->prior = d->prior->next = d

双向链表的插入

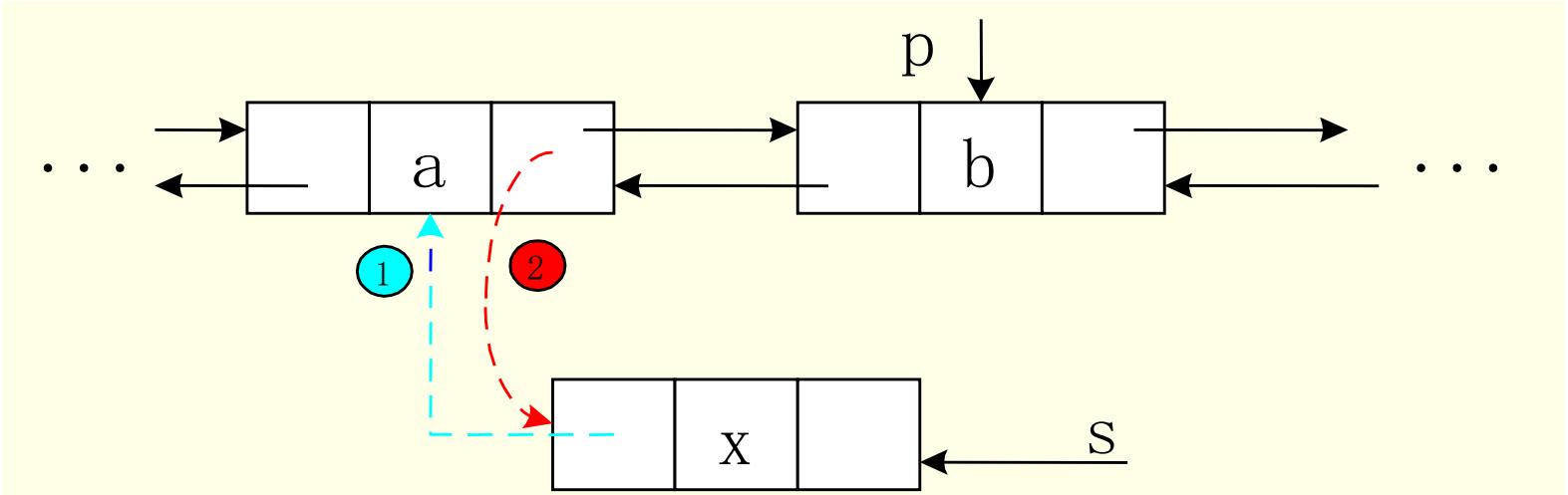


双向链表的插入



1. s->prior=p->prior;

双向链表的插入

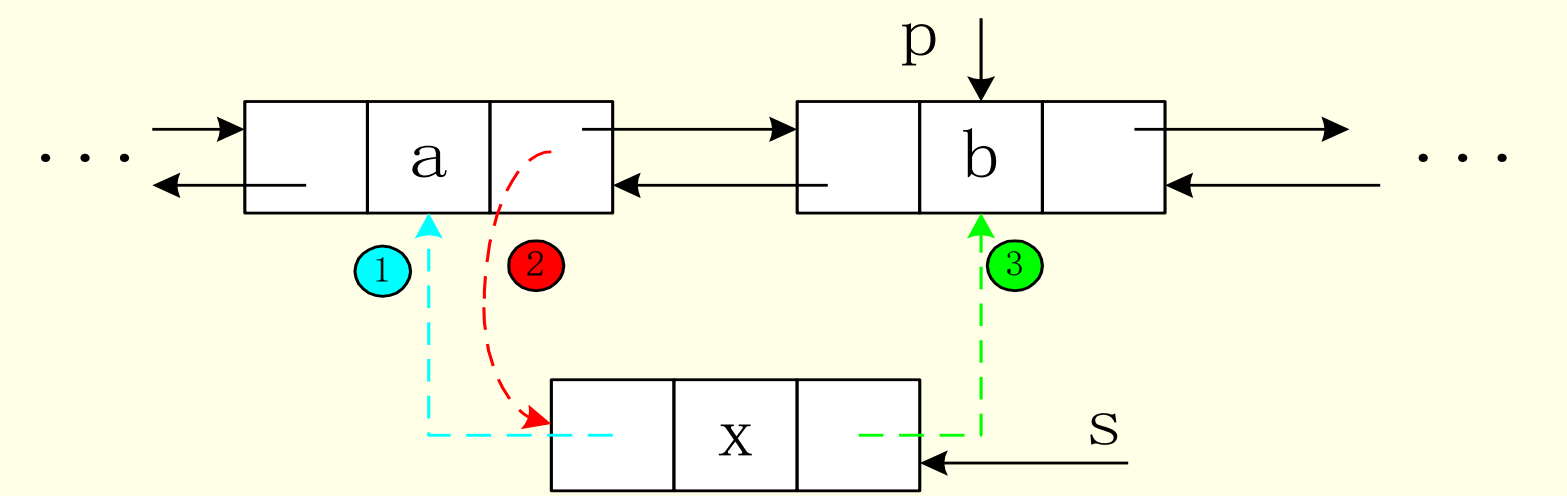


1. s->prior=p->prior;

2. p->prior->next=s;

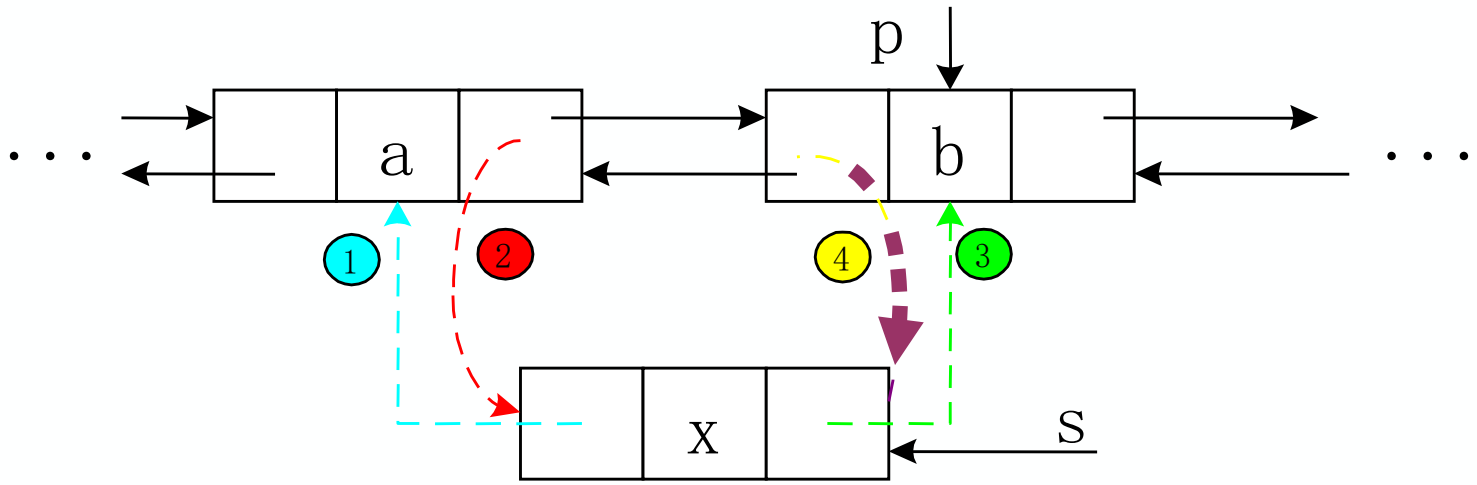
20:09

双向链表的插入



1. s->prior=p->prior;
2. p->prior->next=s;
3. s->next=p;

双向链表的插入



1. s->prior=p->prior;
2. p->prior->next=s;
3. s->next=p;
4. p->prior=s;

双向链表的插入

```
Status ListInsert_DuL(DuLinkList &L,int i,ElemType e){
   if(!(p=GetElemP_DuL(L,i))) return ERROR;
    s=new DuLNode;
   s->data=e;
   s->prior=p->prior;
   p->prior->next=s;
   s->next=p;
   p->prior=s;
   return OK;
}
```