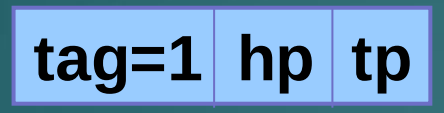
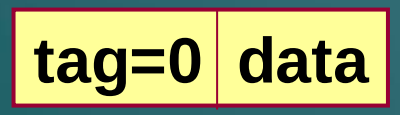
5.5 广义表的存储结构

通常采用头、尾指针的链表结构

表结点:



原子结点：

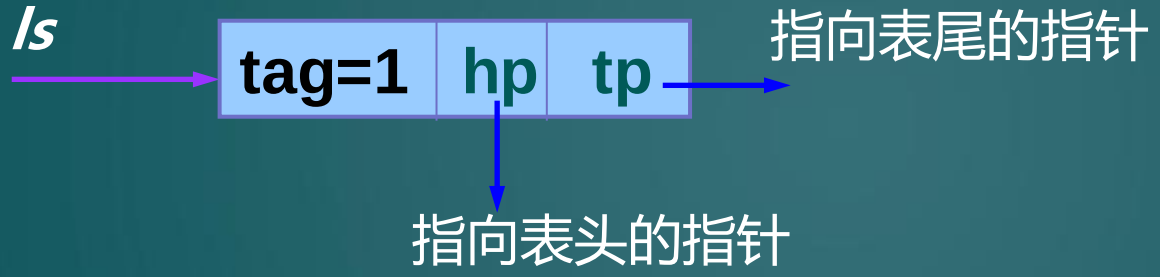


构造存储结构的两种分析方法：

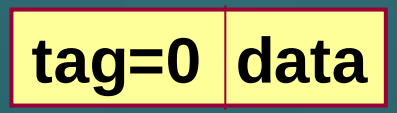
1) 表头、表尾分析法：

空表　　ls=NIL

非空表 *ls*

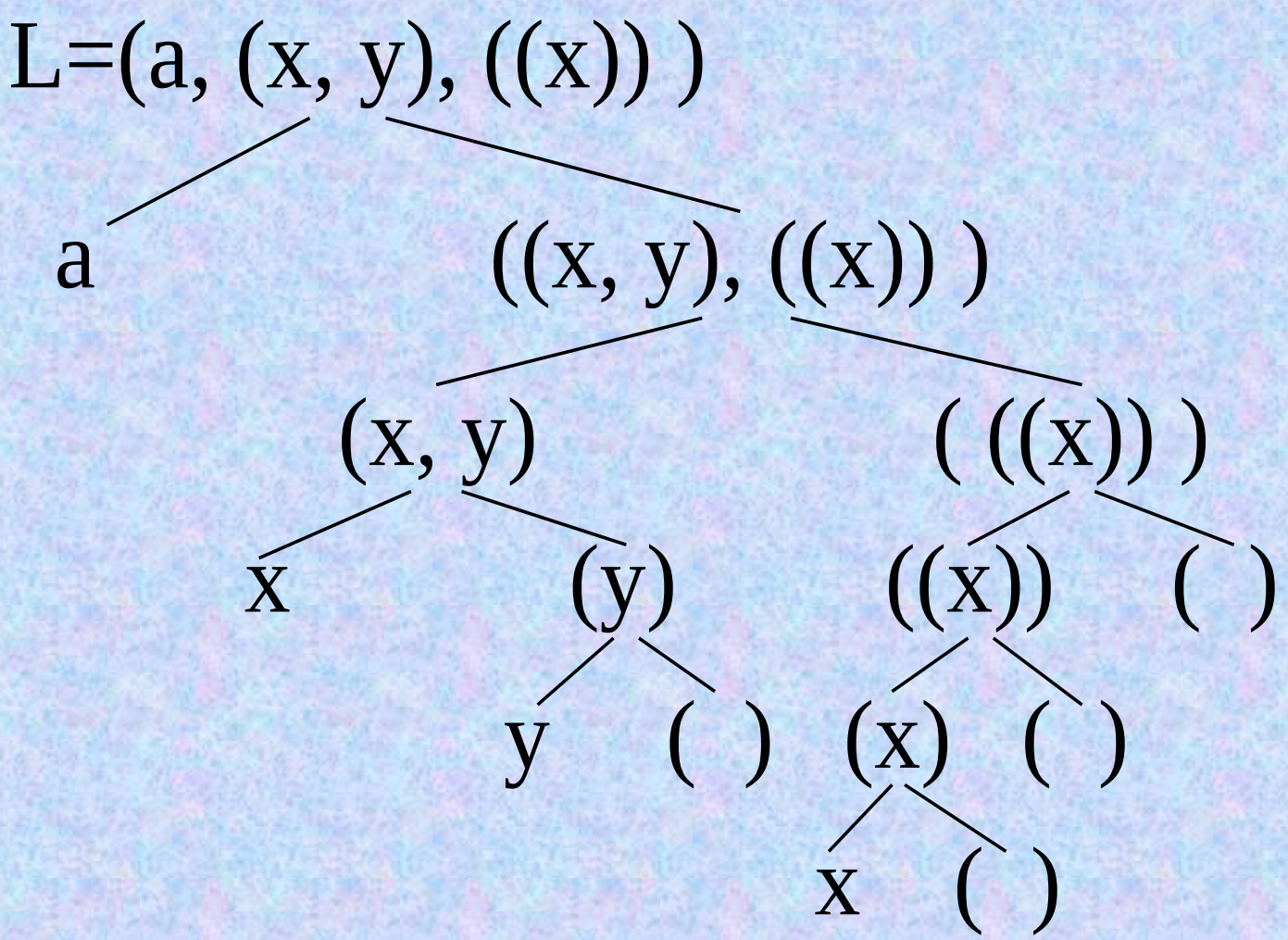


若表头为原子，则为

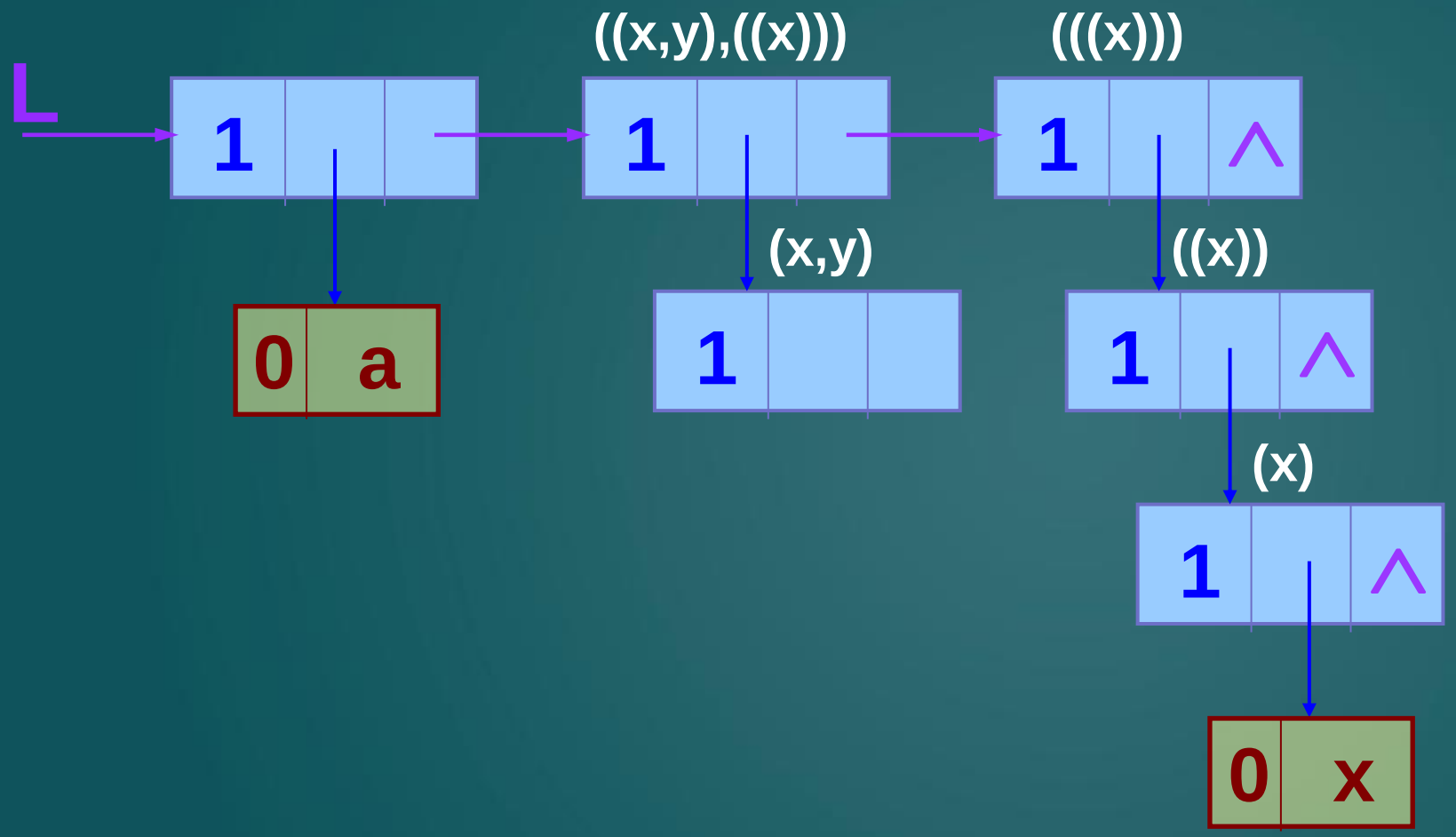


否则，依次类推。

例如:



L = (a, ((x, y), ((x)))



```
// - - - - 广义表的头尾链表存储表示 - - - -
typedef enum{ATOM, LIST}ElemTag；
    //ATOM=0：原子，LIST=1：子表
typedef struct GLNode{
      ElemTag tag；    //公共部分，用于区分原子结点和表结点
      union{              //原子结点和表结点的联合部分
      AtomType atom； //atom是原子结点的值域， AtomType
    由用户定义
      struct { struct GLNode * hp, *tp；}ptr；
                                  //ptr是表结点的指针域，ptr.hp和
    ptr.tp分别指向表头和表尾
  }；
}*Glist；                    //广义表类型
```

例如: A = ( )

B = (e)

C = (a , (b , c , d))

D = (A , B , C)

E = (a , E) = (a , (a , (a , … ,)))

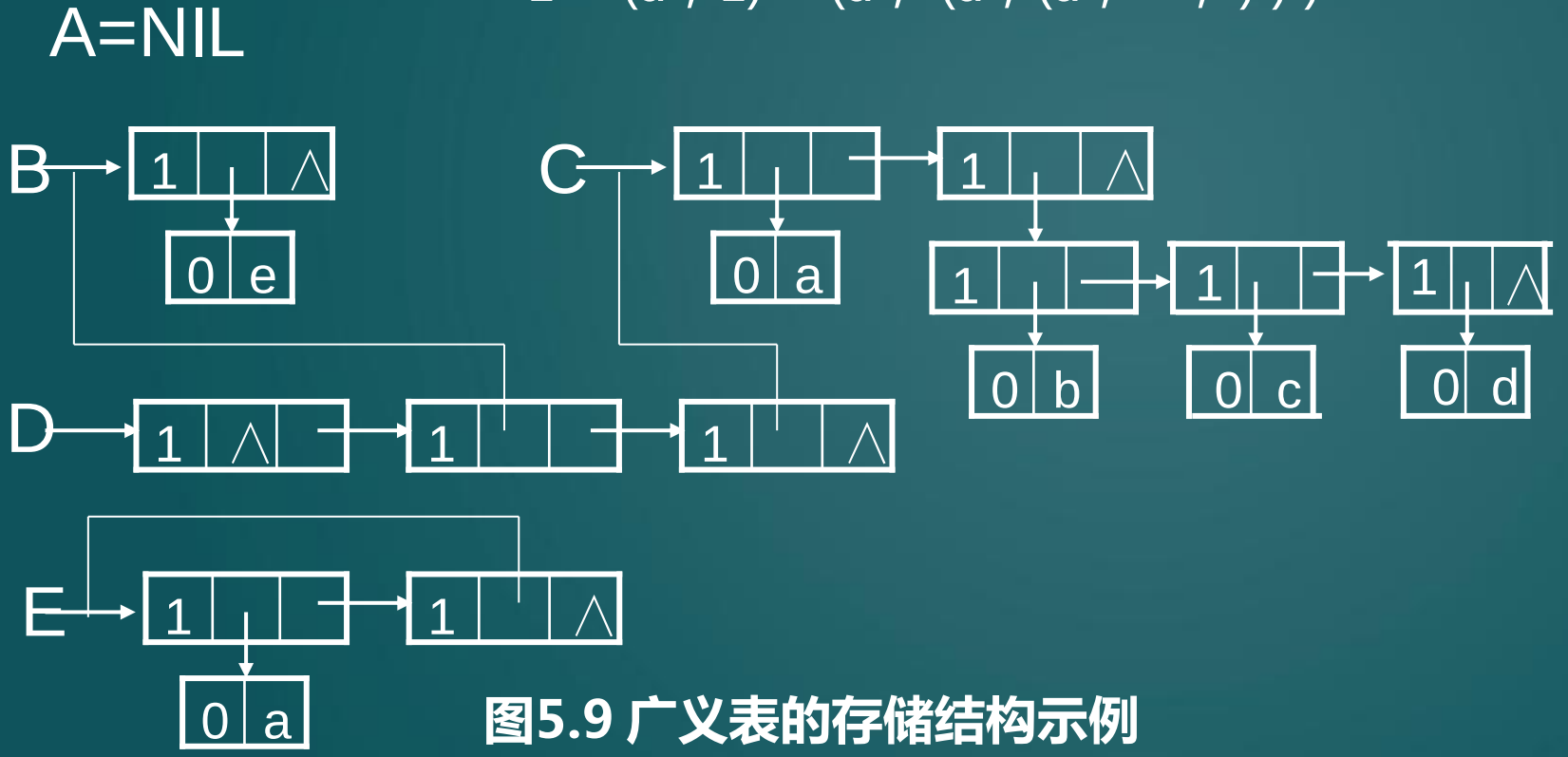
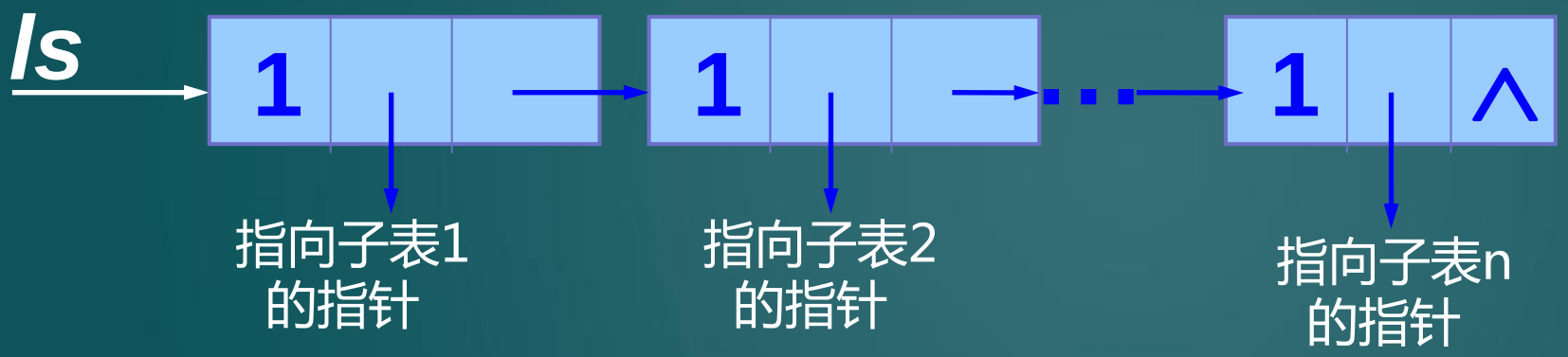


图5.9 广义表的存储结构示例

2) 子表分析法：

空表　　ls=NIL

非空表

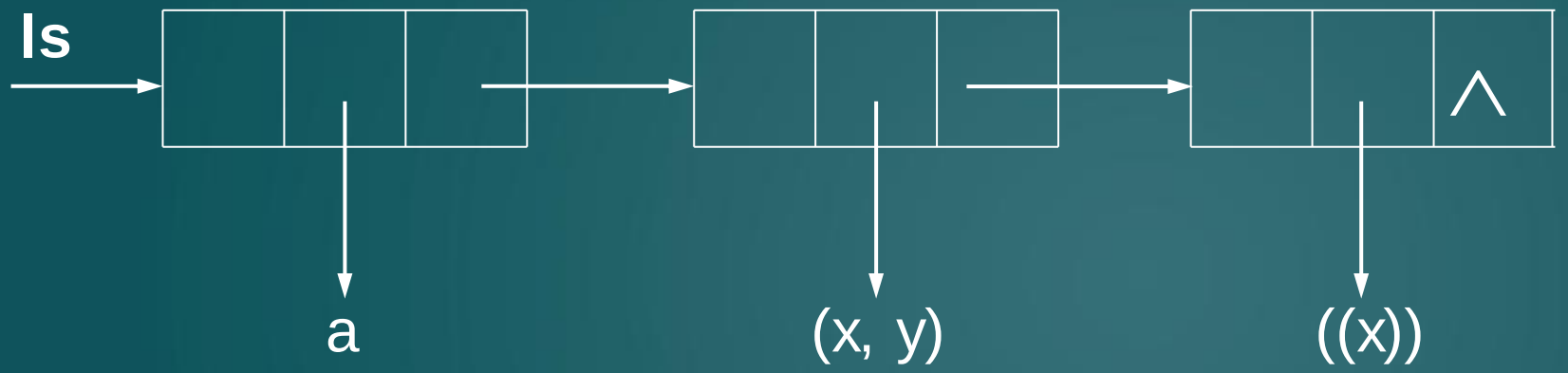


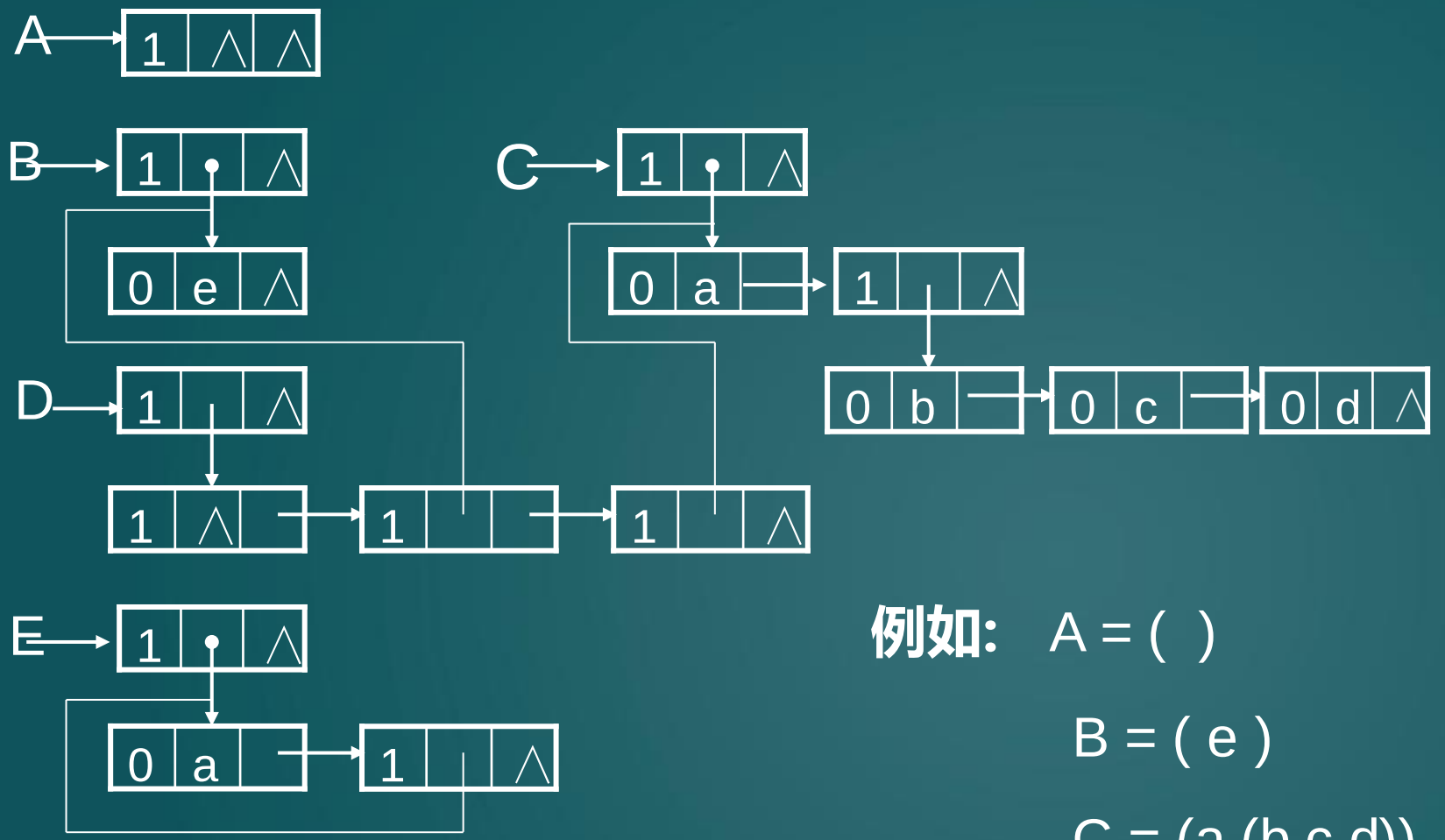
若子表为原子，则为

tag=0	data	tp

否则，依次类推。

例如: LS=(a, (x,y), ((x)))





例如: A = ( )

B = (e)

C = (a,(b,c,d))

D = (A, B, C)

E = (a, E)

图5.11 列表的另一种链表表示