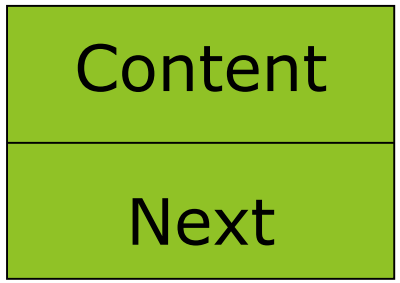
单链表的操作

- 单链表的操作一定要熟悉！
  - 链表涵盖了结构和指针的知识点，请仔细阅读教材174-178，以及例5-16、17。
  - 每个元素是由结构类型的结点（Node）构成

Node

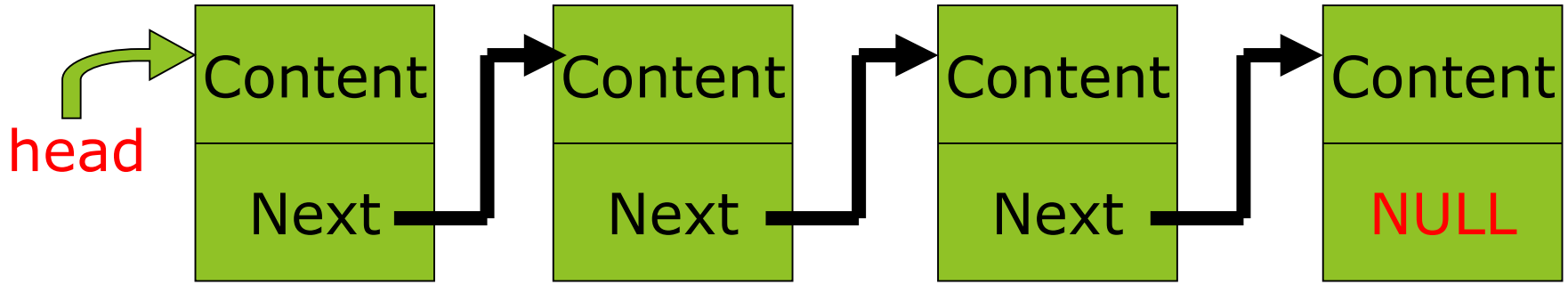


content ： 存储的数据

数据类型由具体问题决定

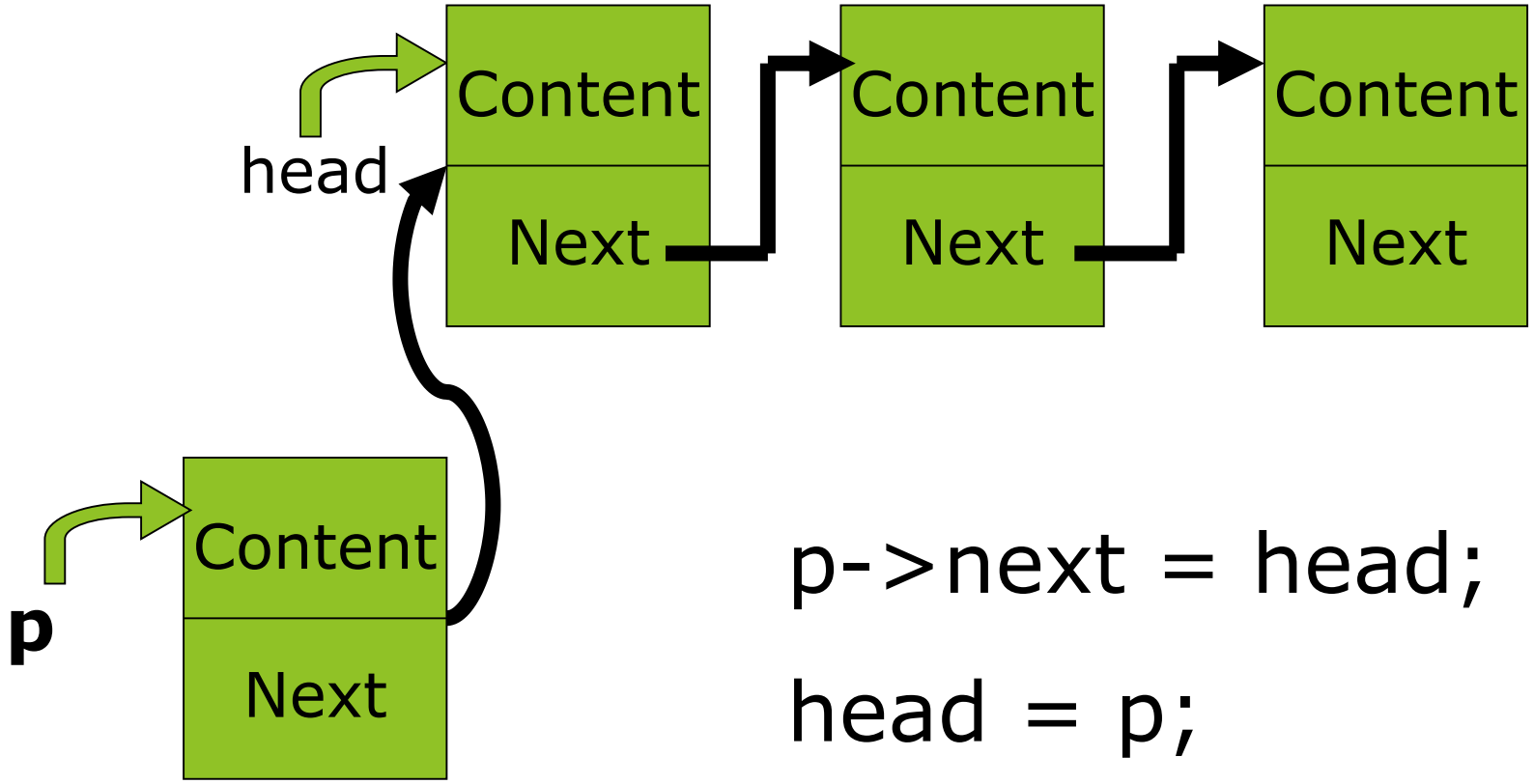
next： 下一个结点的地址（指针类型）

- 链表的访问



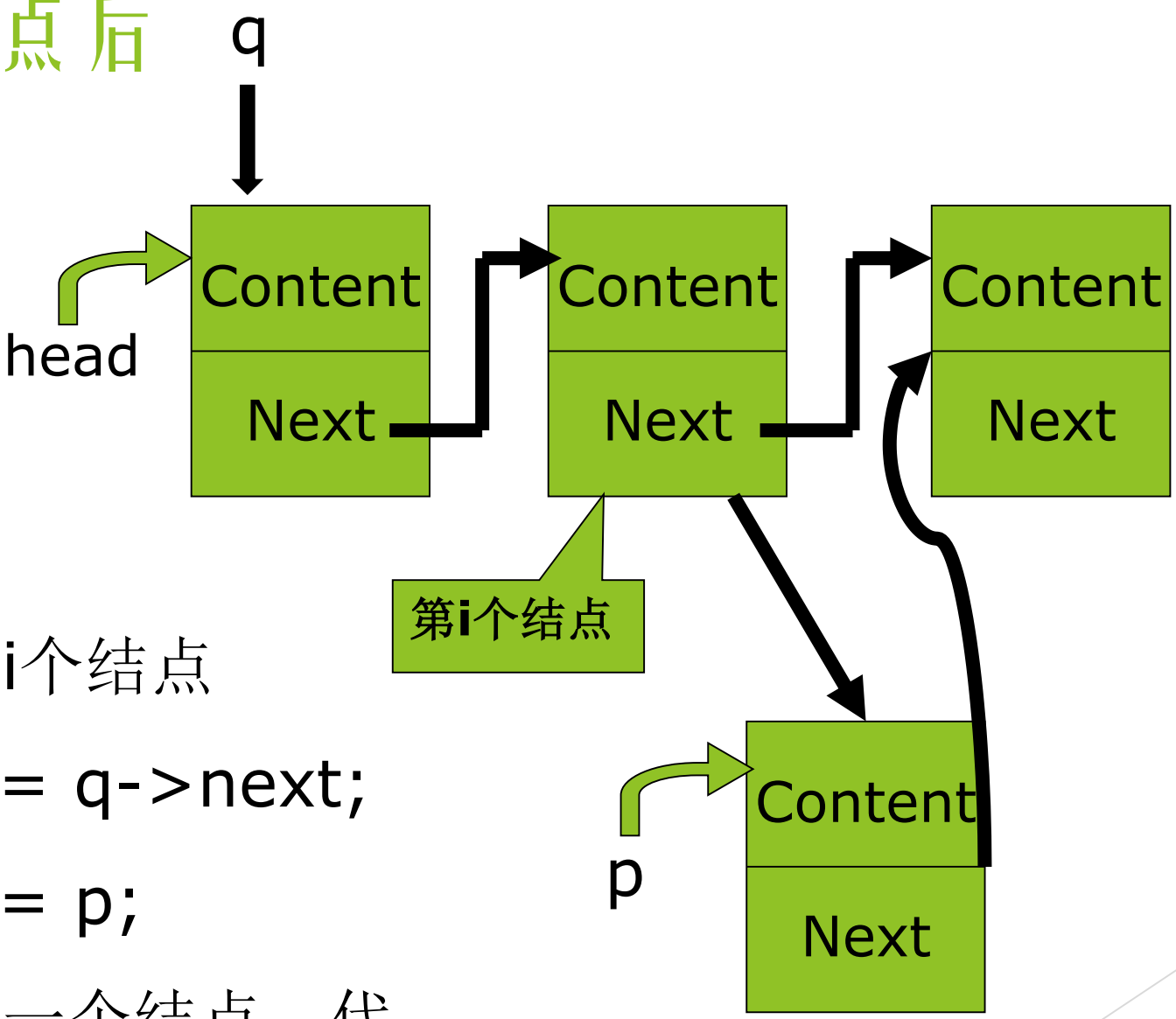
- head：链表访问的起始
- NULL：最后一个结点的判断标记
  链表访问的结束

插入结点（插在表头）



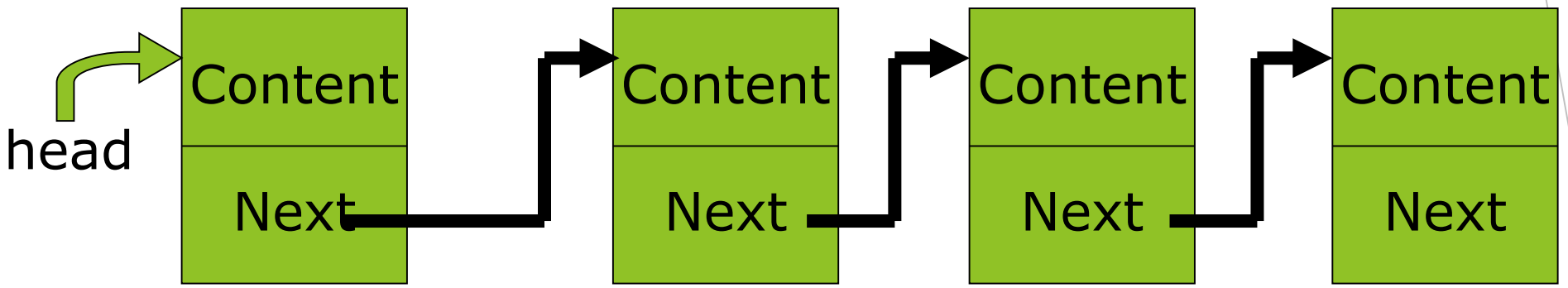

```
p->next = head;
head = p;
```

插在第i个结点后




```
//q指向第i个结点
p->next = q->next;
q->next = p;
// i为最后一个结点，代码同样
```

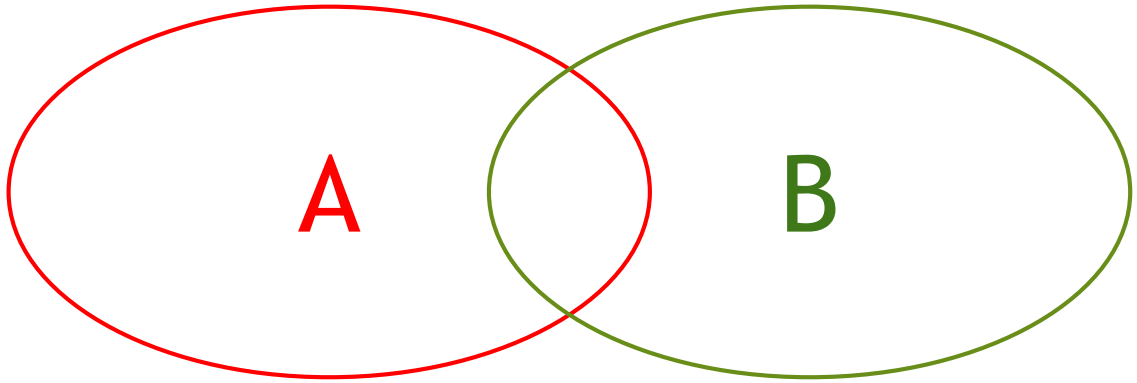
删除表头




```
p=head;
head=head->next;
int x = p->Content;
delete p;
```

集合的实现

- 程序功能分解：
  - 集合A,B的建立：无序、唯一（默认输入正确）
  - A和B的并集：A原有元素+B中的并且不在A中的元素
  - A和B的交集：既在A中又在B中的元素
  - A与B的差集：在集合A中，去除A和B的交集元素。
  - 集合运算结果的输出：输出应该抽象为一个功能。



功能细化

- 集合的建立：需要考虑如果输入了重复的元素，应该去除，因为集合的元素具有唯一性。
- 交集、并集、差集：都需要判断一个元素是不是在集合中。
- 集合的输出
- 程序结束，集合对应空间的释放

功能封装

- 将集合的操作功能封装到函数中
  - 代码易读、易维护

- 功能划分：
  - input：链表的创建
  - output：链表的输出
  - remove: 链表空间的释放（动态空间的管理）
  - is_element:判断元素是不是在集合中
  - insert: 将元素放入集合中
  - Uni_set、 Inter_set、 dif_set:并、交、差

功能的实现——集合的建立

- 因为集合的元素不要求排序，因此可以用最简单的链表建立方法：在表头插入节点

  思路一： Node * input ()　　教材有示例

  思路二：将一个元素加入集合中，这是一个需要重复利用的功能，可以抽象为一个函数：

  void Insert(Node * &h, int n)

  注意这里形参应该是引用，因为结点在头部插入的话，头指针是不断变化的，如果是Node * h，仅仅是改变h指向的结点内容，不能改变h本身。

功能的实现——判断元素是不是属于集合

- bool is_element(Node * h, int n)

  判断n是不是在 h指向的集合链表中。

  判断方法：循环

```
for( Node * p = h; p!=NULL; p=p->next)
        if( n== p->content)
            ……
```

功能的实现——并集

- 最优的实现应该是产生一个新的集合，这个集合的元素是A和B的并集，如果仅仅是输出集合元素，那么并集的结果就不能重复利用。

1、将a中的结点全部复制

2、逐个判断b中每一个结点的content是不是在a中，是，则跳过；不是，则加入

```
Node * Uni_set( Node * a, Node * b)
{  Node * h = NULL;
    for( Node * p = a; p!=NULL; p=p->next)
        Insert( h, p->content);
   for( Node * p = b; p!=NULL; p=p->next)
       if( Is_element(a,p->content))          continue;
       else  Insert( h, p->content);
    return h;
}
```

功能的实现——交集

- 最优的实现也是产生一个新的集合，这个集合的元素是A和B的交集。

```
Node * Inter_set( Node * a, Node * b)
{     逐个判断a中每一个结点的content是不是在b中，
      是，则加入；不是，则跳过
}
```

功能的实现——差集

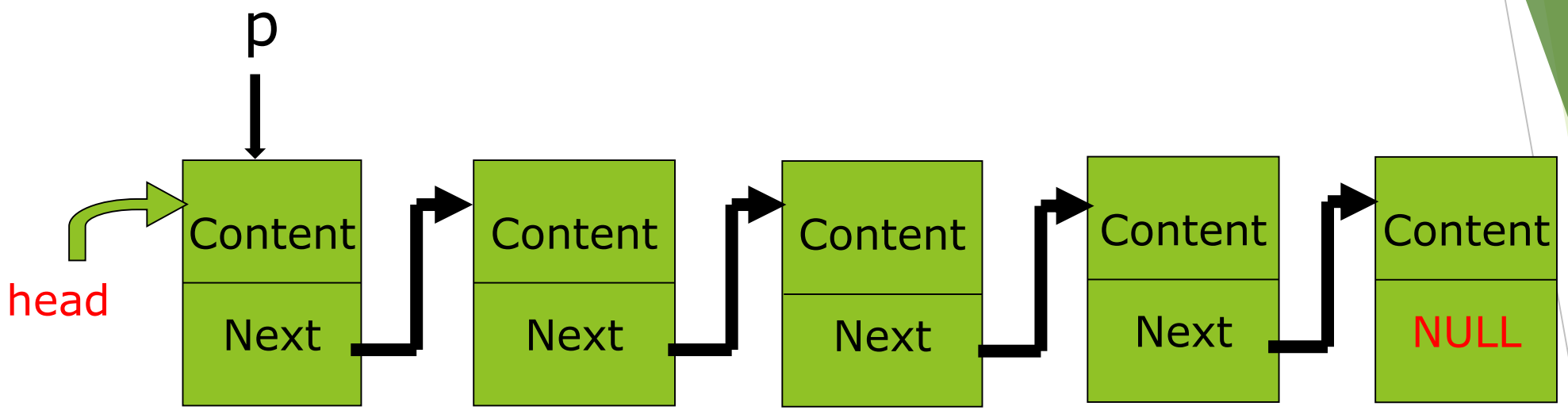
- 最优的实现也是产生一个新的集合，这个集合的元素是A和B的差集。

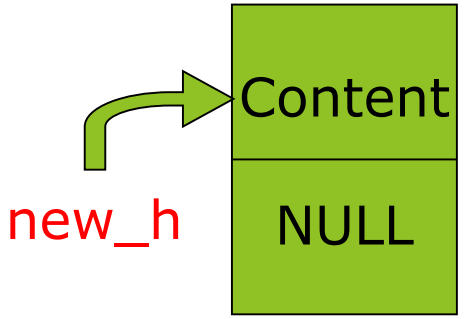
```
Node * dif_set( Node * a, Node * b)
{     逐个判断a中每一个结点的content是不是在b中，
      是，则跳过；不是，则加入。
}
```

插入排序

- 用链表完成插入排序的流程
  - 首先构造初始链表，可以教材例5-16中的input函数完成。
  - 插入排序有两种实现：
    - 在一个新链表上完成
    - 在利用原链表结点完成

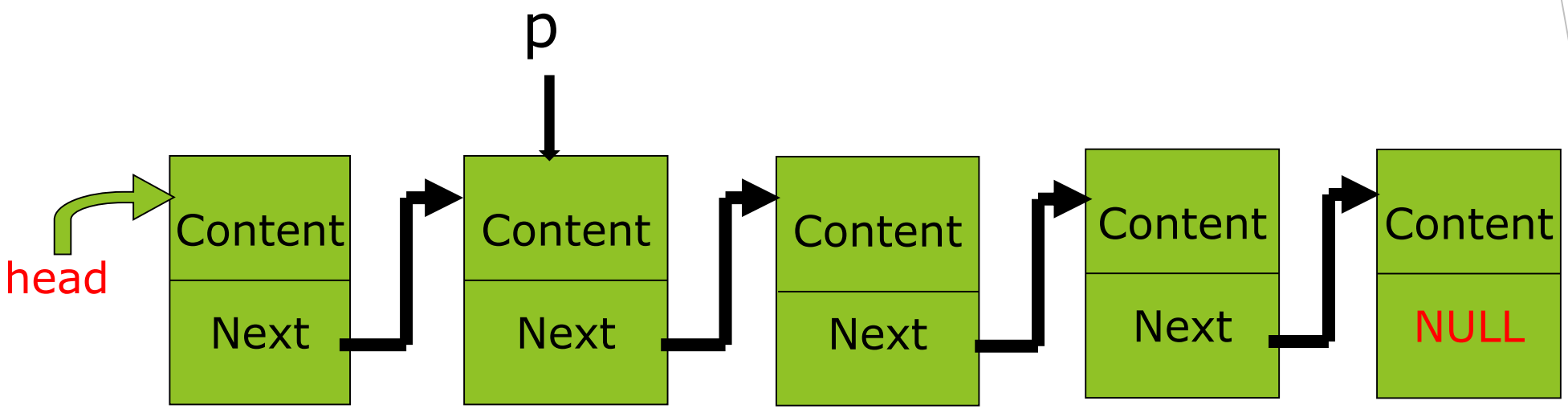
在新链表上进行插入排序：产生新的头结点

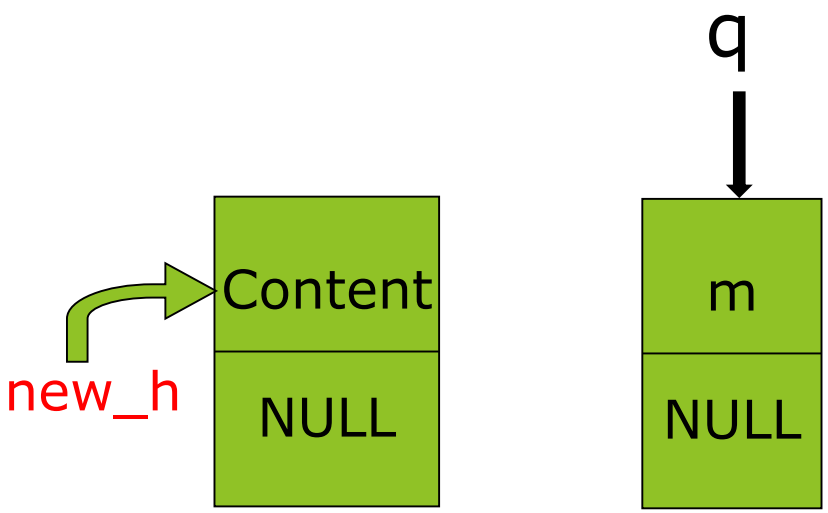





```
Node *p = head;
Node * new_h = new Node;
new_h->content=p->content;
new_h->next = NULL;
```

在新链表上进行插入排序：循环插入后续结点





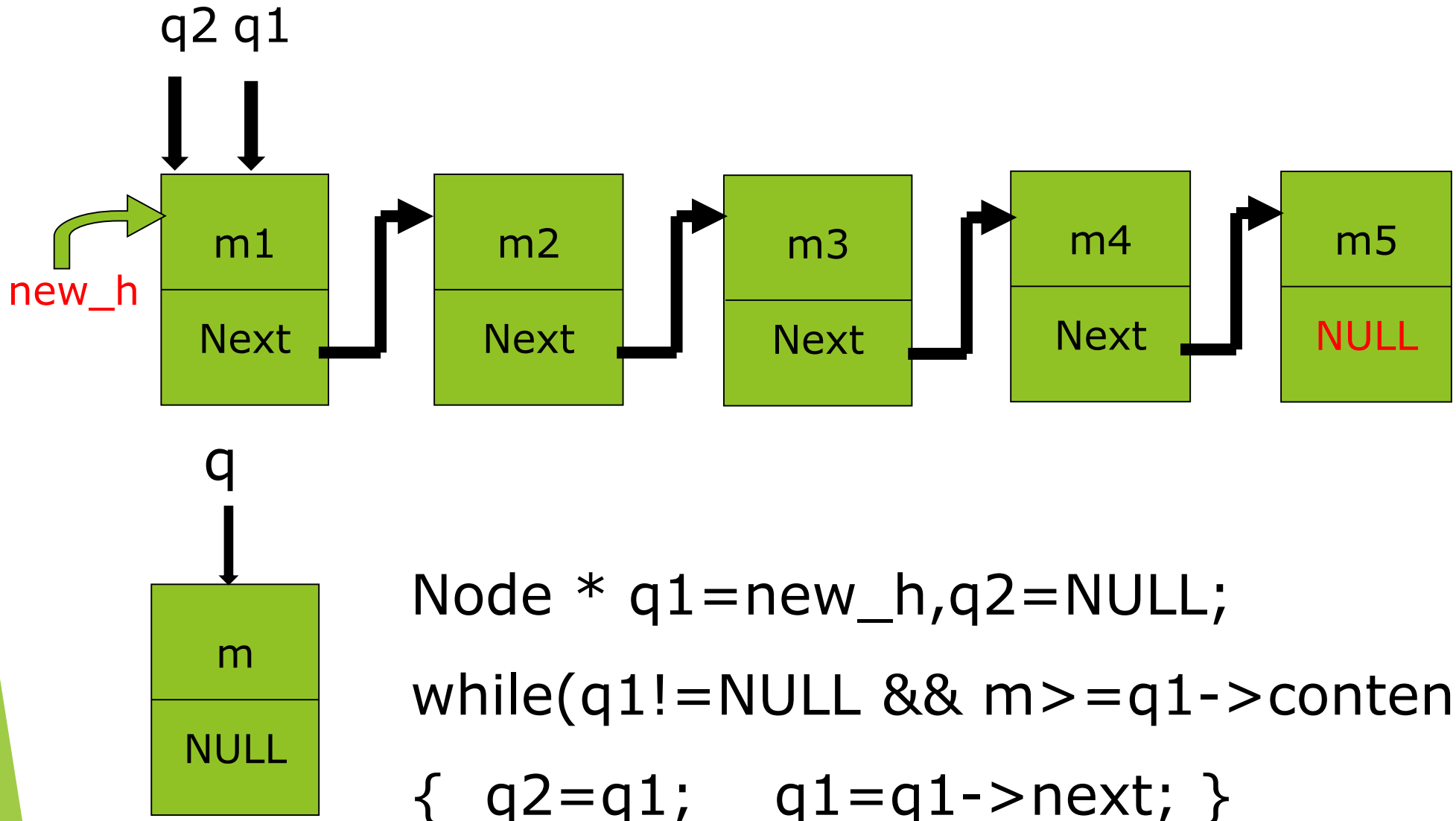
外层循环：p指向原链表中下一个结点

产生一个新结点，q指向新结点

int m = p->content;

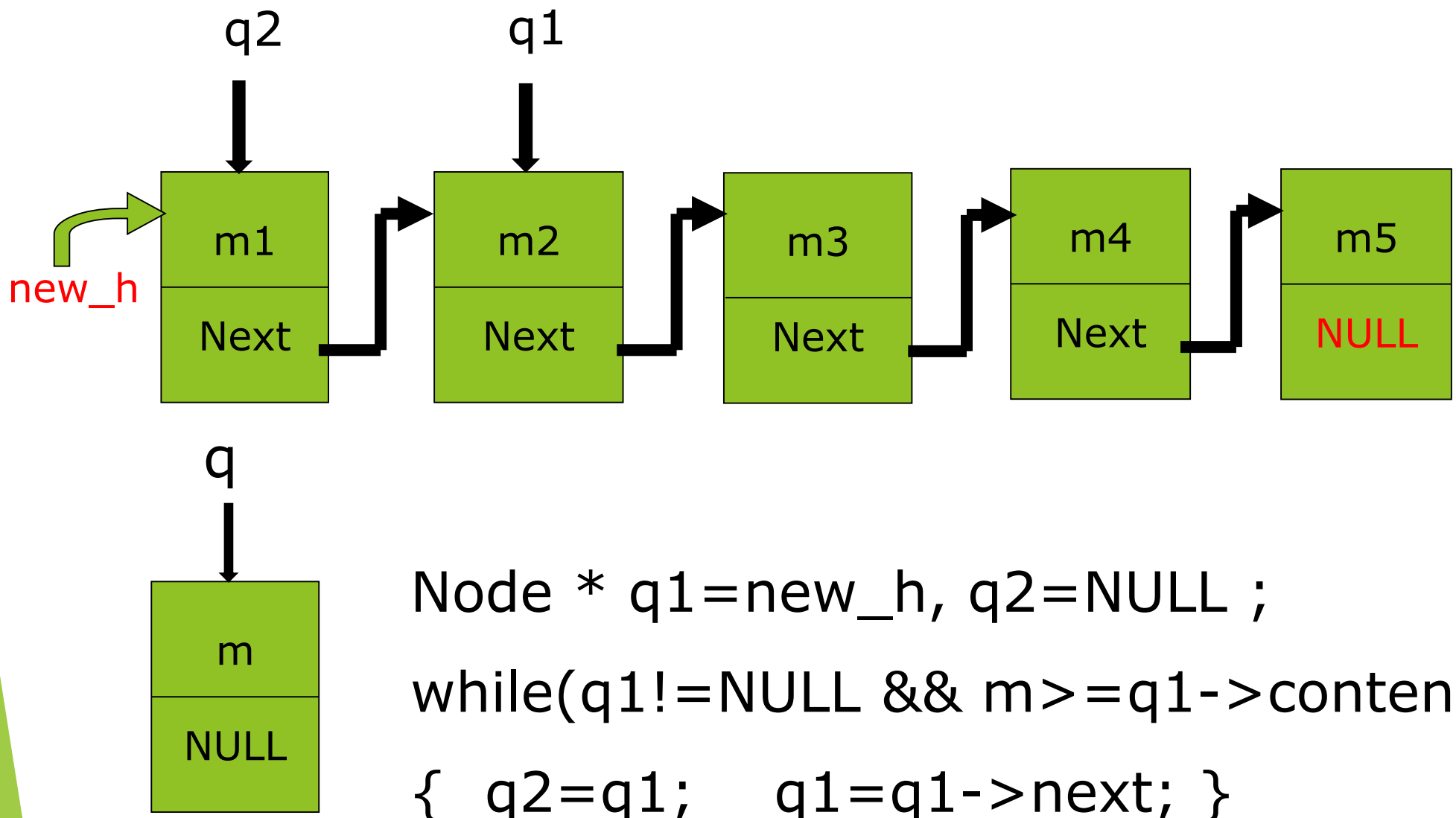
内层循环：在新链表中寻找m的插入位置，插入新结点

在有序链表上插入新结点



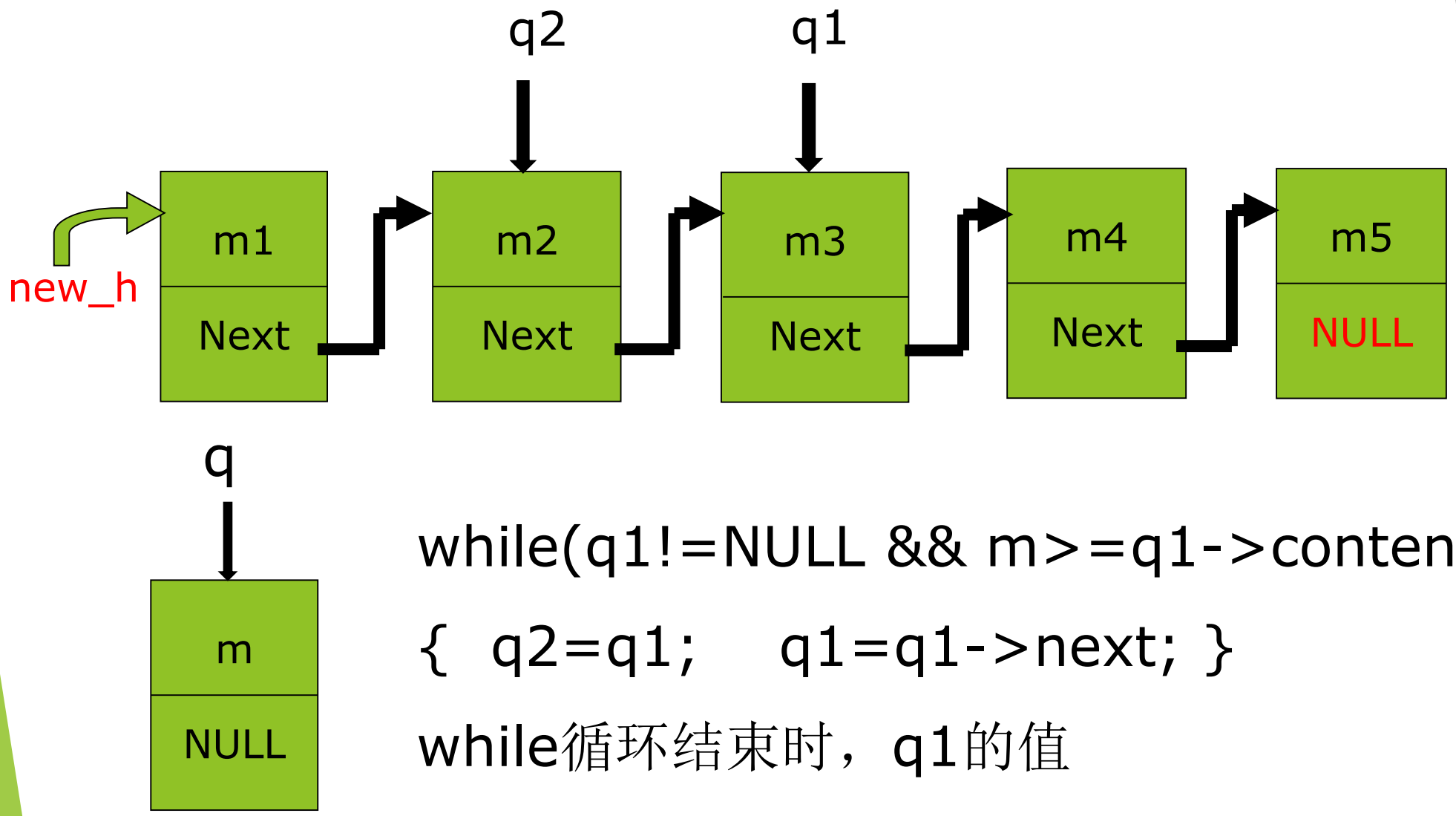

```
Node * q1=new_h,q2=NULL;
while(q1!=NULL && m>=q1->content )
{  q2=q1;   q1=q1->next; }
```

在有序链表上插入新结点




```
Node * q1=new_h, q2=NULL ;
while(q1!=NULL && m>=q1->content )
{  q2=q1;   q1=q1->next; }
```

在有序链表上插入新结点

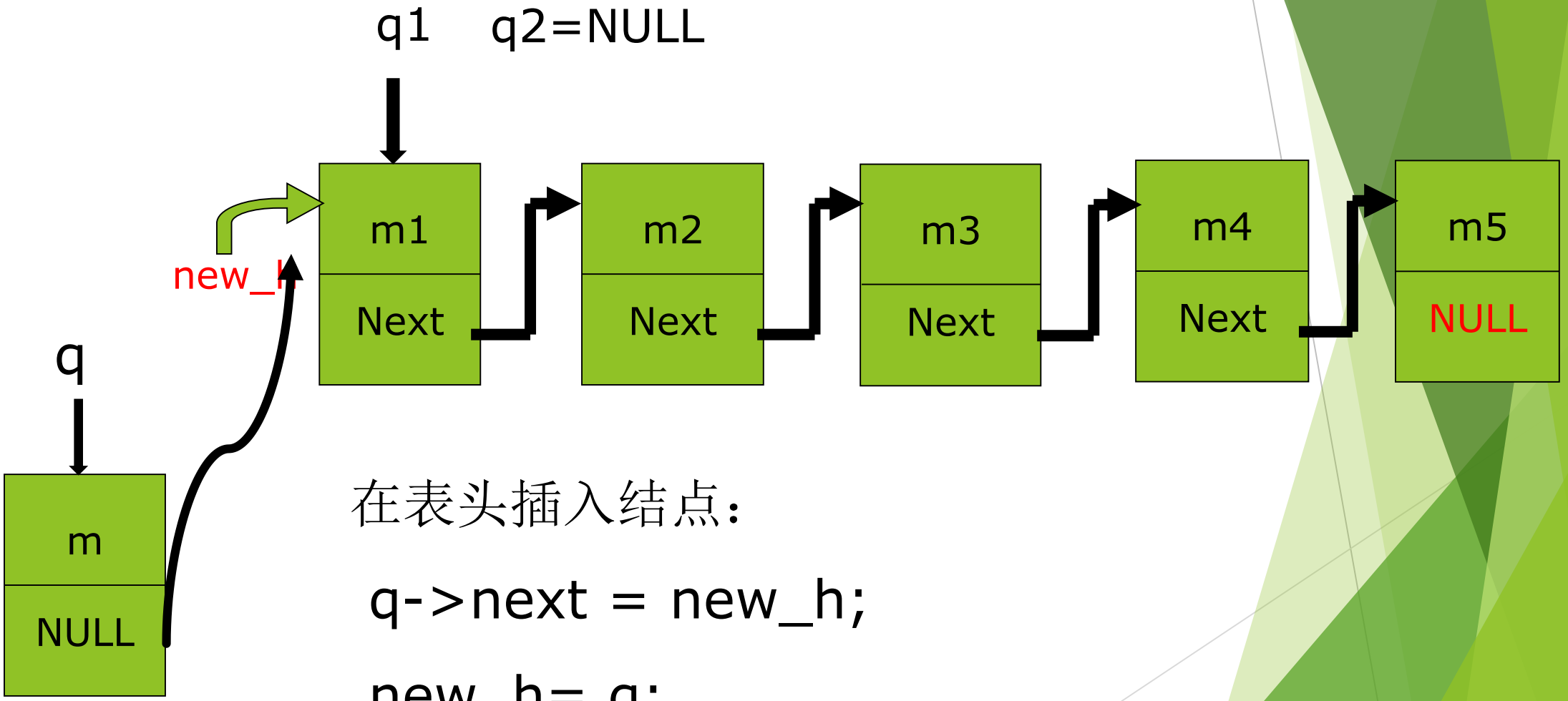



```
while(q1!=NULL && m>=q1->content )
{ q2=q1;   q1=q1->next; }
```

while循环结束时，q1的值

（1）头指针（2）表中结点地址 （3）NULL

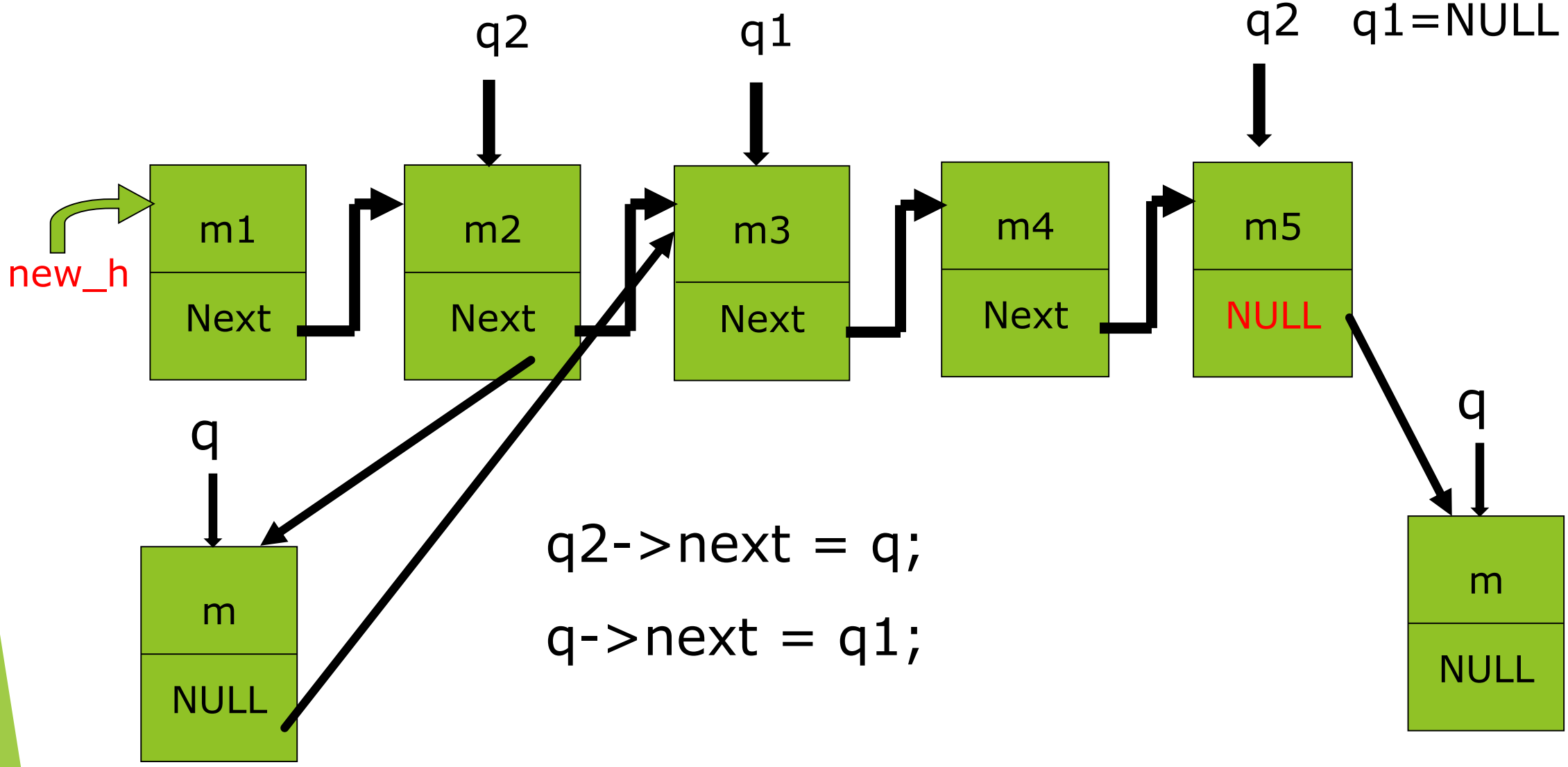
在有序链表上插入新结点：q1指向头结点



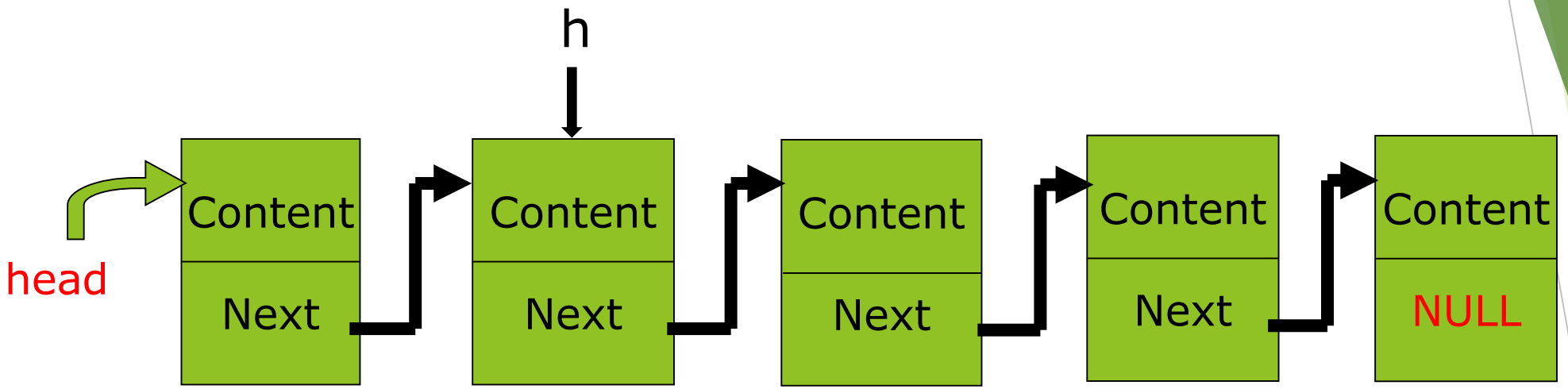
在表头插入结点：

```
q->next = new_h;
new_h= q;
```

在有序链表上插入新结点：q1指向表中/q1=NULL



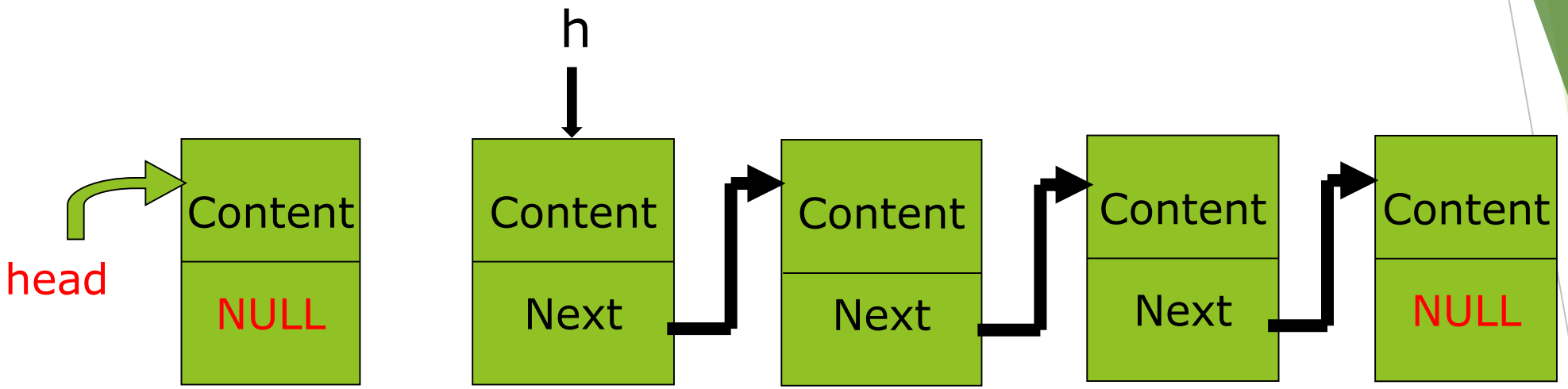
利用原链表结点，完成插入排序



Node *h = head->next;

h为剩余链表结点的头指针

利用原链表结点，完成插入排序



Node *h = head->next;

h为剩余链表结点的头指针

head->next = NULL;

下面的操作和前面的插入排序操作类似，每次把h指向的头结点取出，按序插入head所指向的链表中。