Linked List - Inserting in order

- Reading: Savitch, Chapter 10

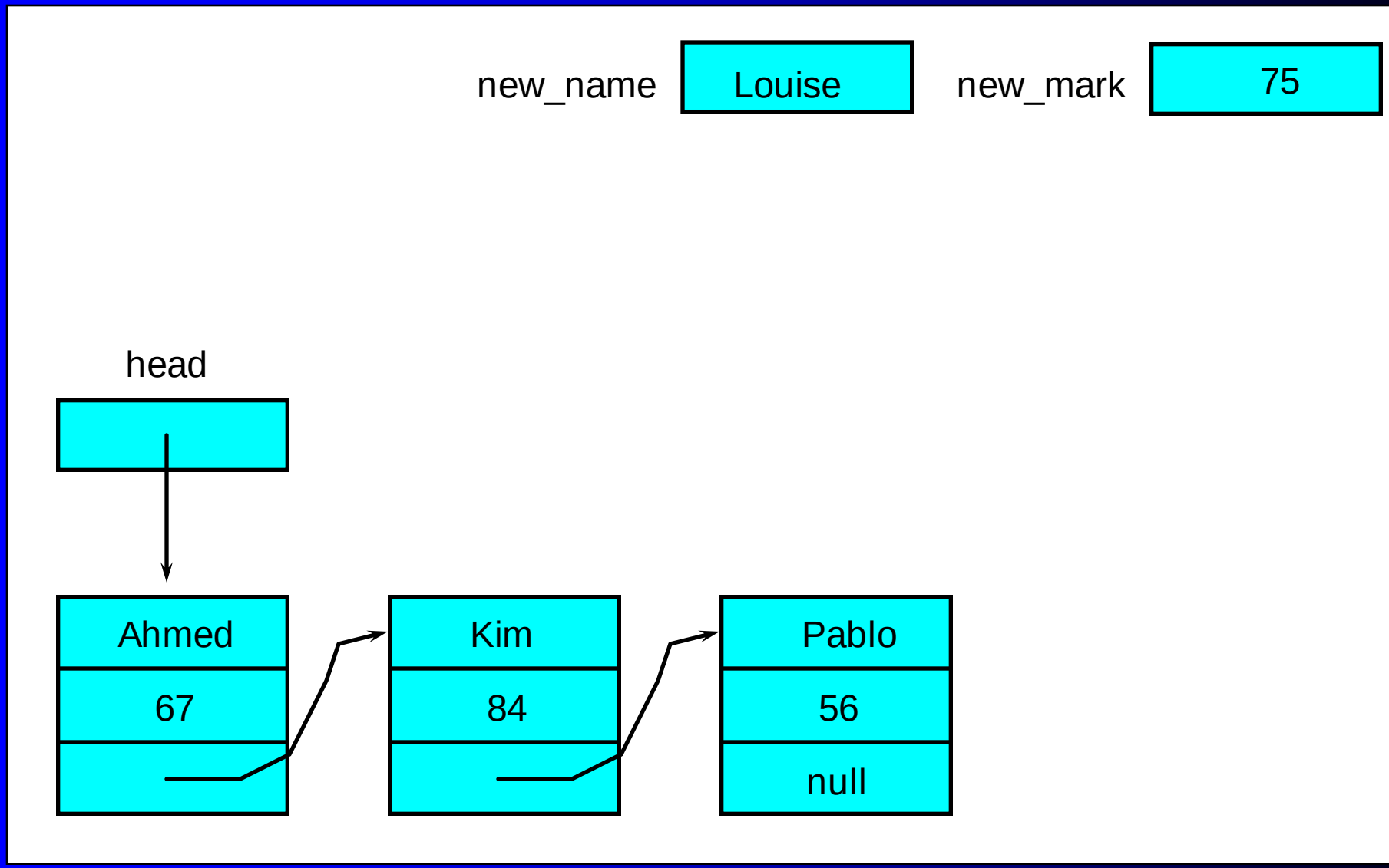
Objectives

- To learn how to insert elements in order in a linked list
- To learn what a doubly-linked list is

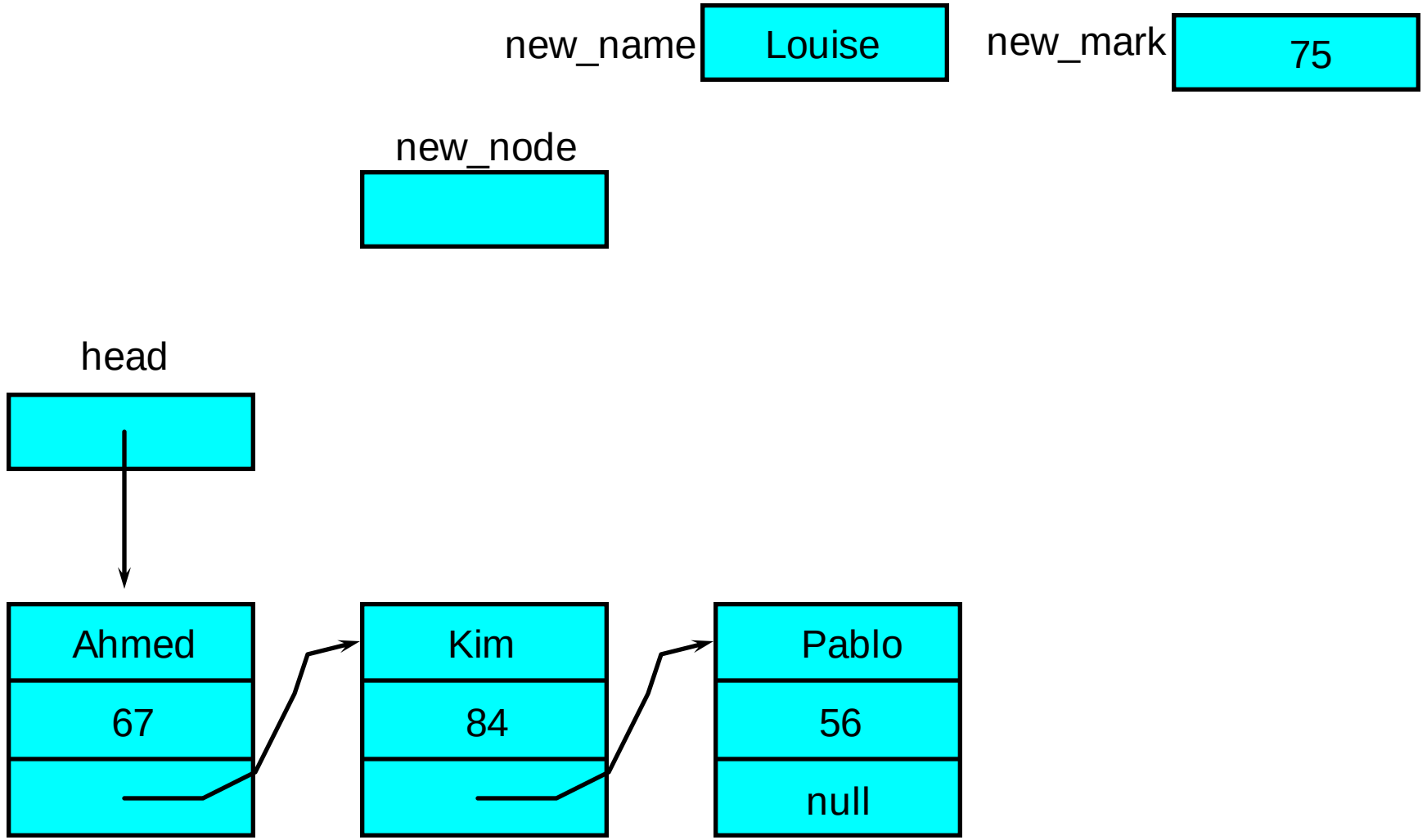
Ordered lists

- Nodes ordered by the "key"
- Example: list of student nodes
  - possible keys = name, mark
- Use key values to insert new nodes into correct positions

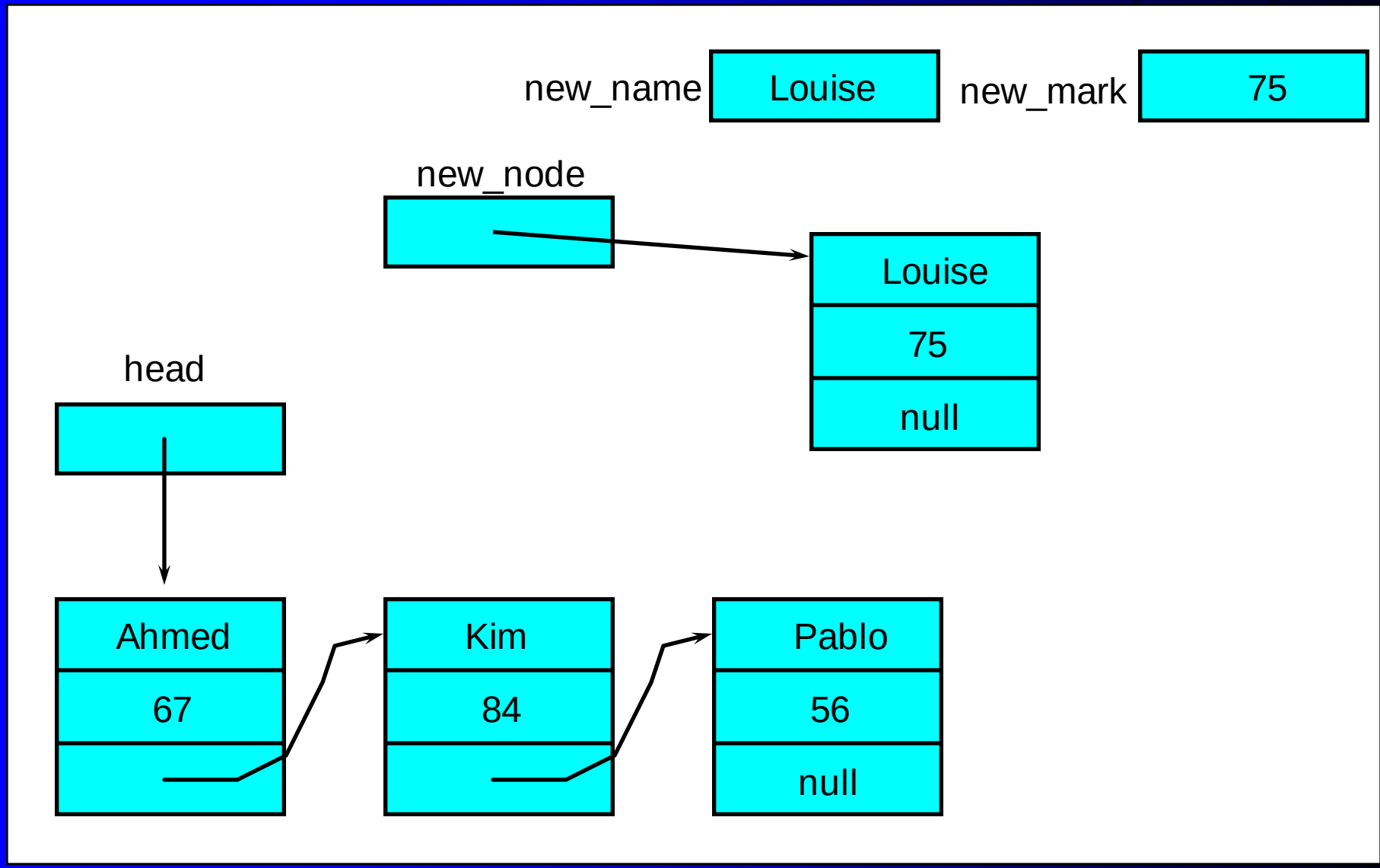
Insertion in an ordered list



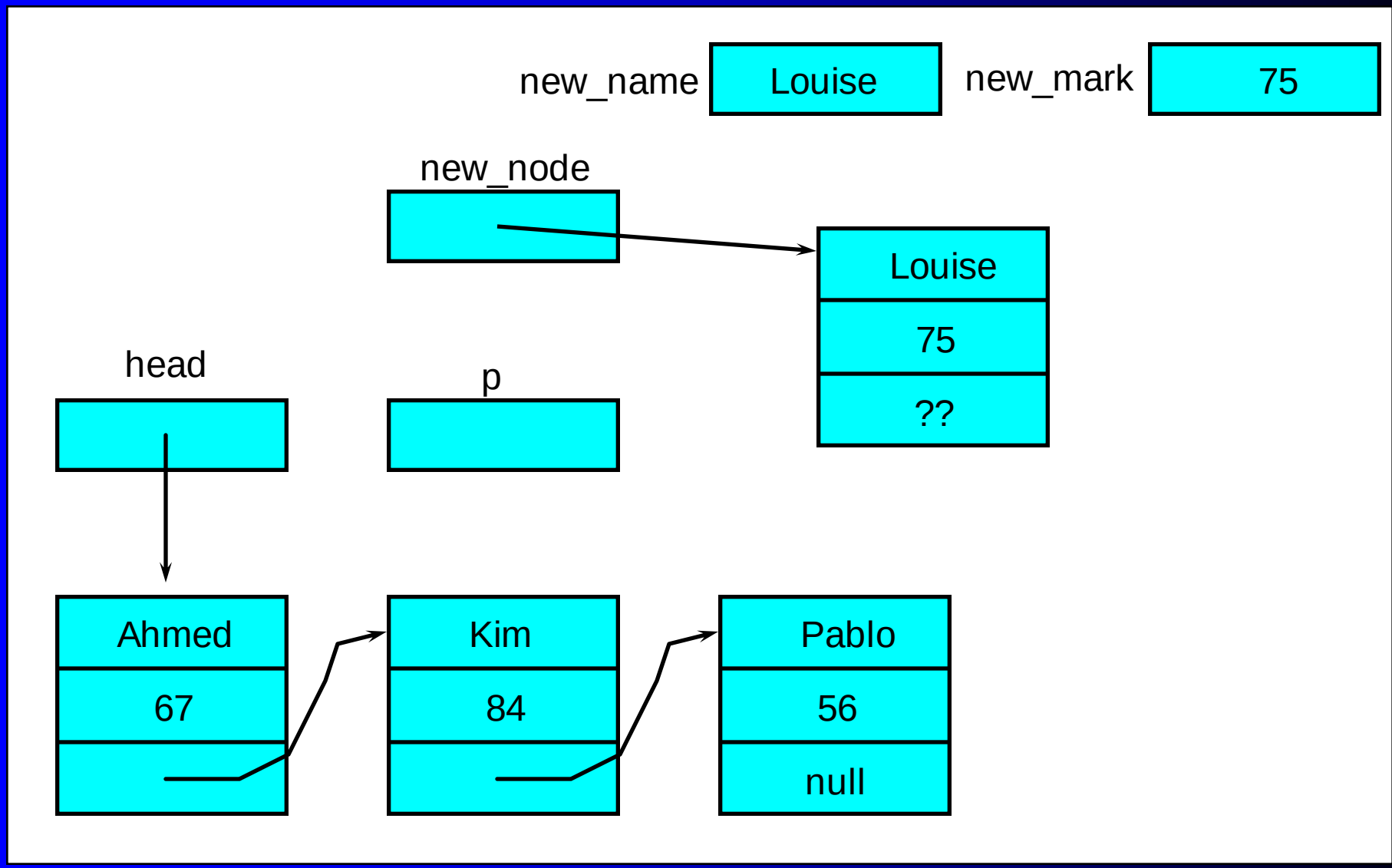
Insertion in an ordered list (ctd)



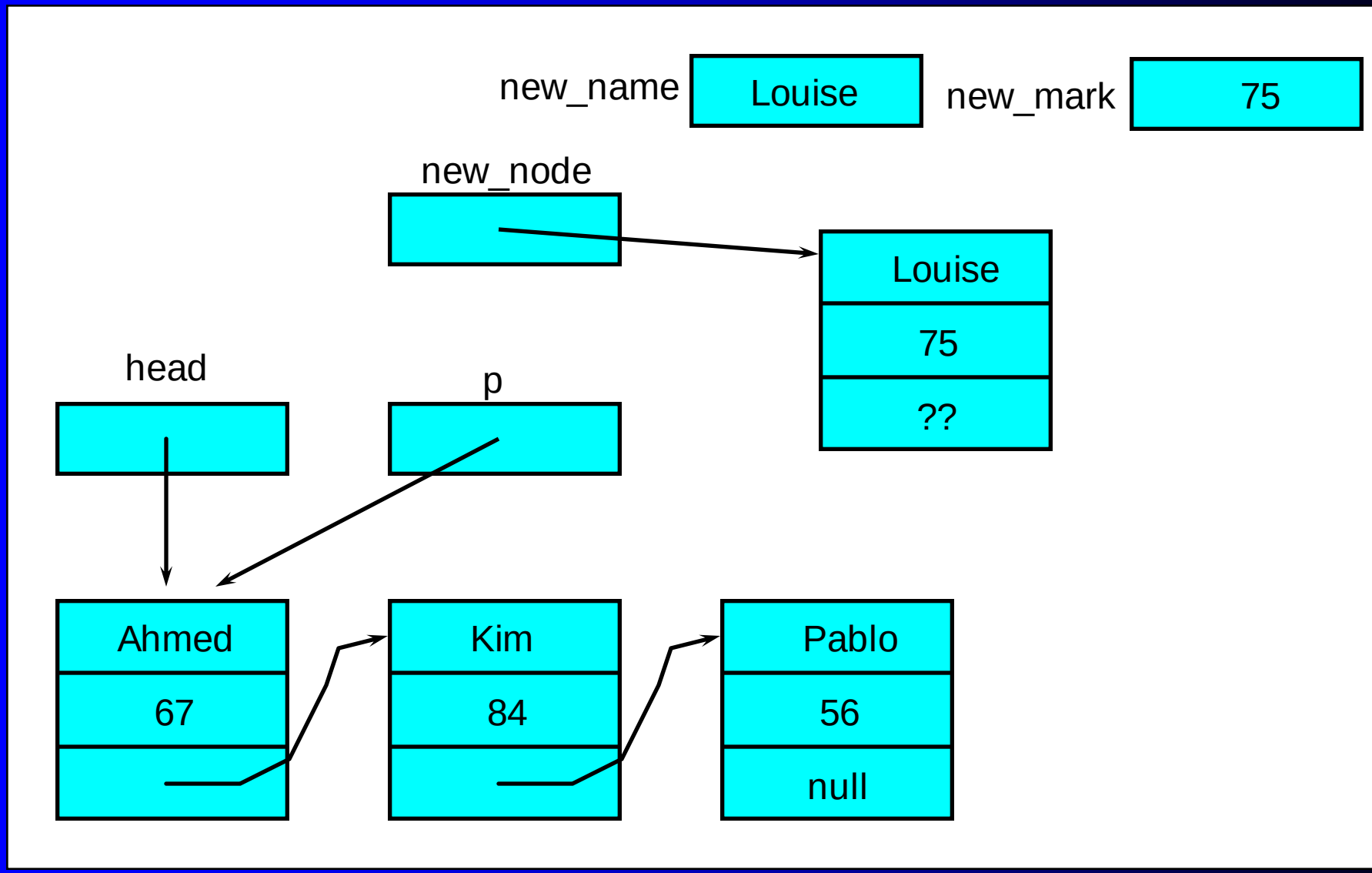
Insertion in an ordered list (ctd)



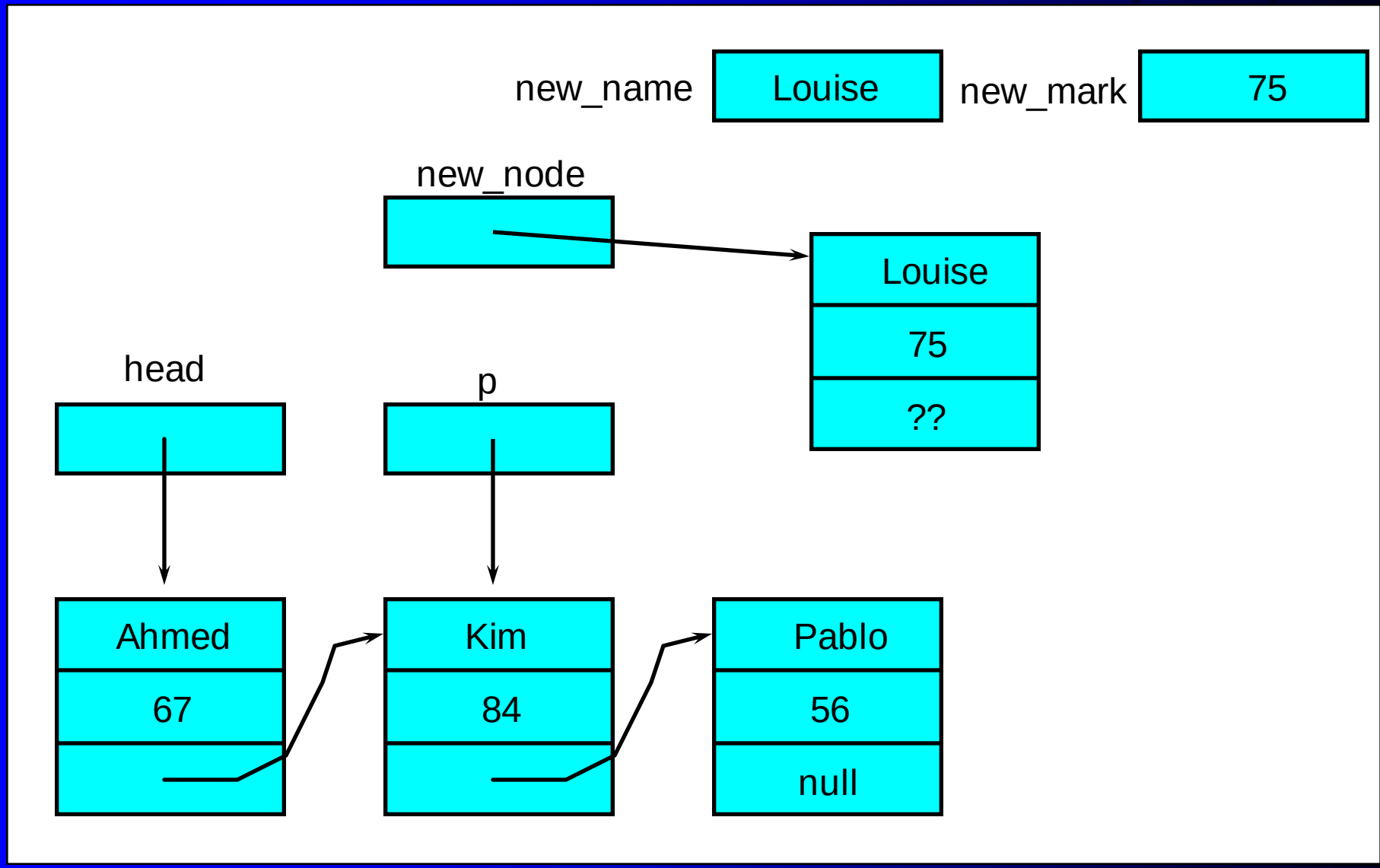
Insertion in an ordered list (ctd)



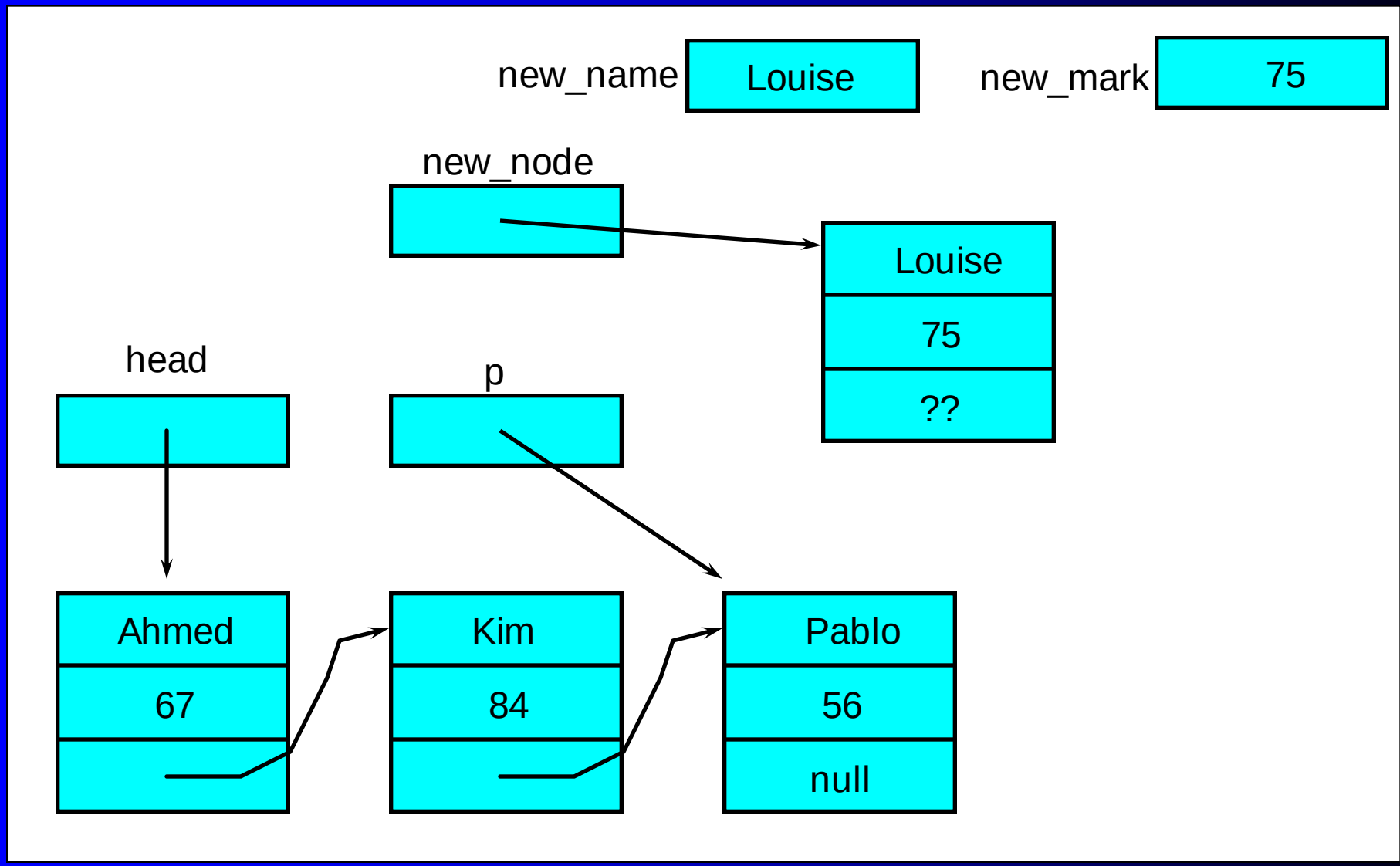
Insertion in an ordered list (ctd)



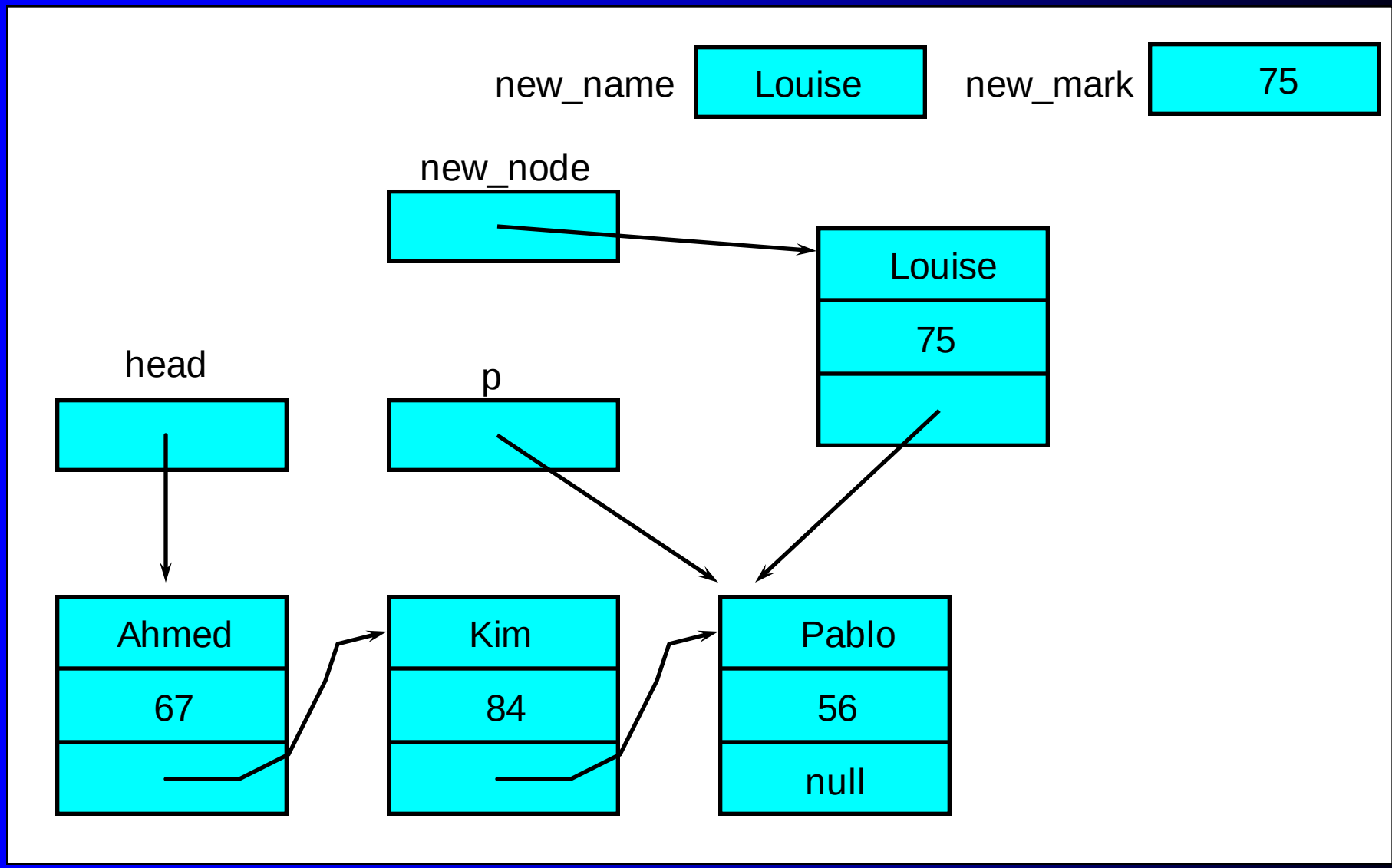
Insertion in an ordered list (ctd)



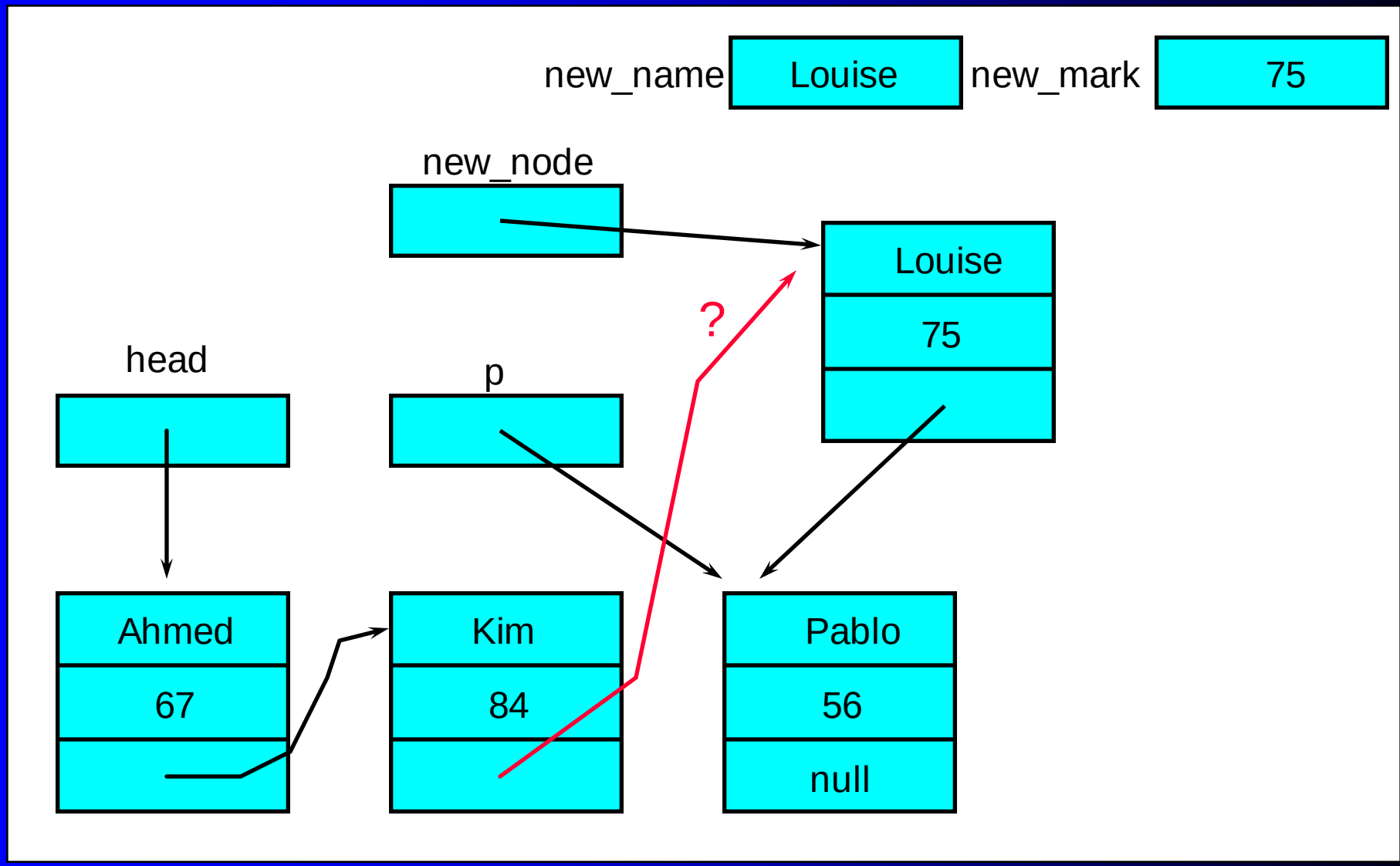
Insertion in an ordered list (ctd)



Insertion in an ordered list (ctd)



Insertion in an ordered list (ctd)



Insertion in an ordered list (ctd)

- Problem
  - when inserting into an ordered list, the position to insert is only determined by the item after this position in the list!
- Solutions
  - look one ahead in the list
  - keep a trailing pointer
  - use a doubly-linked list

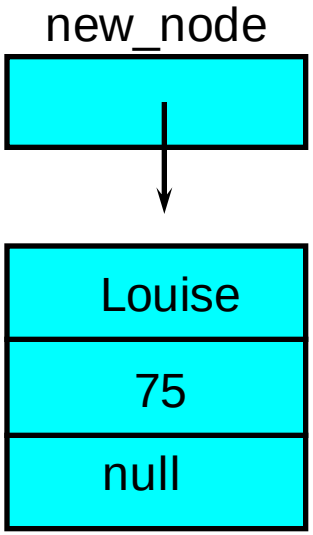
Looking one ahead in the list

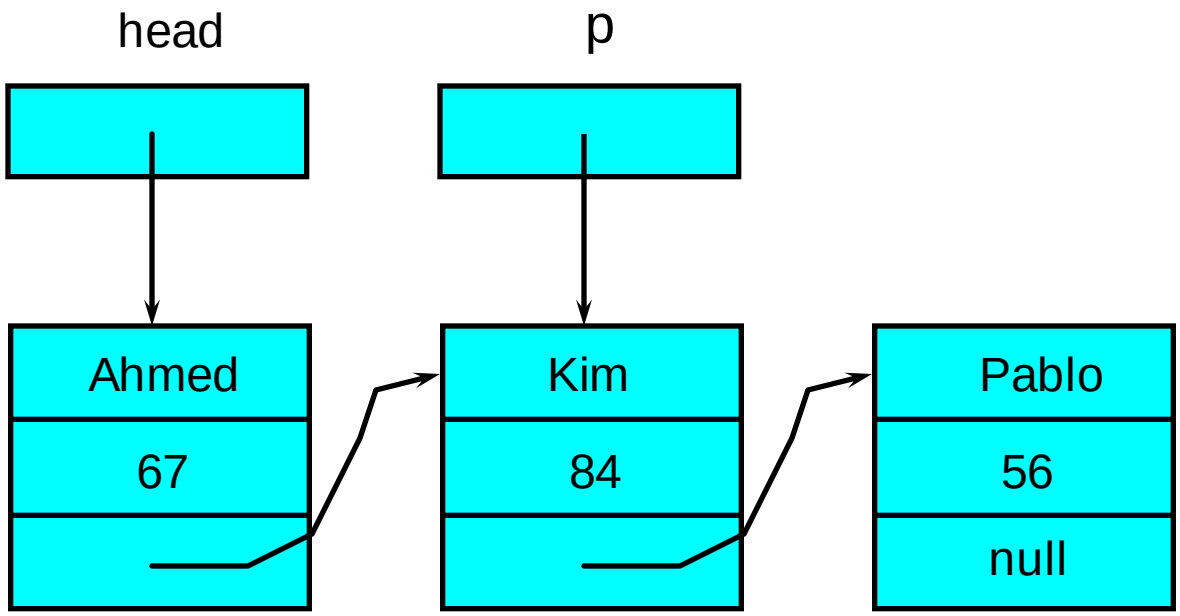
Test:

```
while (((p.next).name).compareTo(new_node.name)) <0)
   p = p.next;
```

Insertion:

```
   new_node.next = p.next;  p.next = new_node;
```





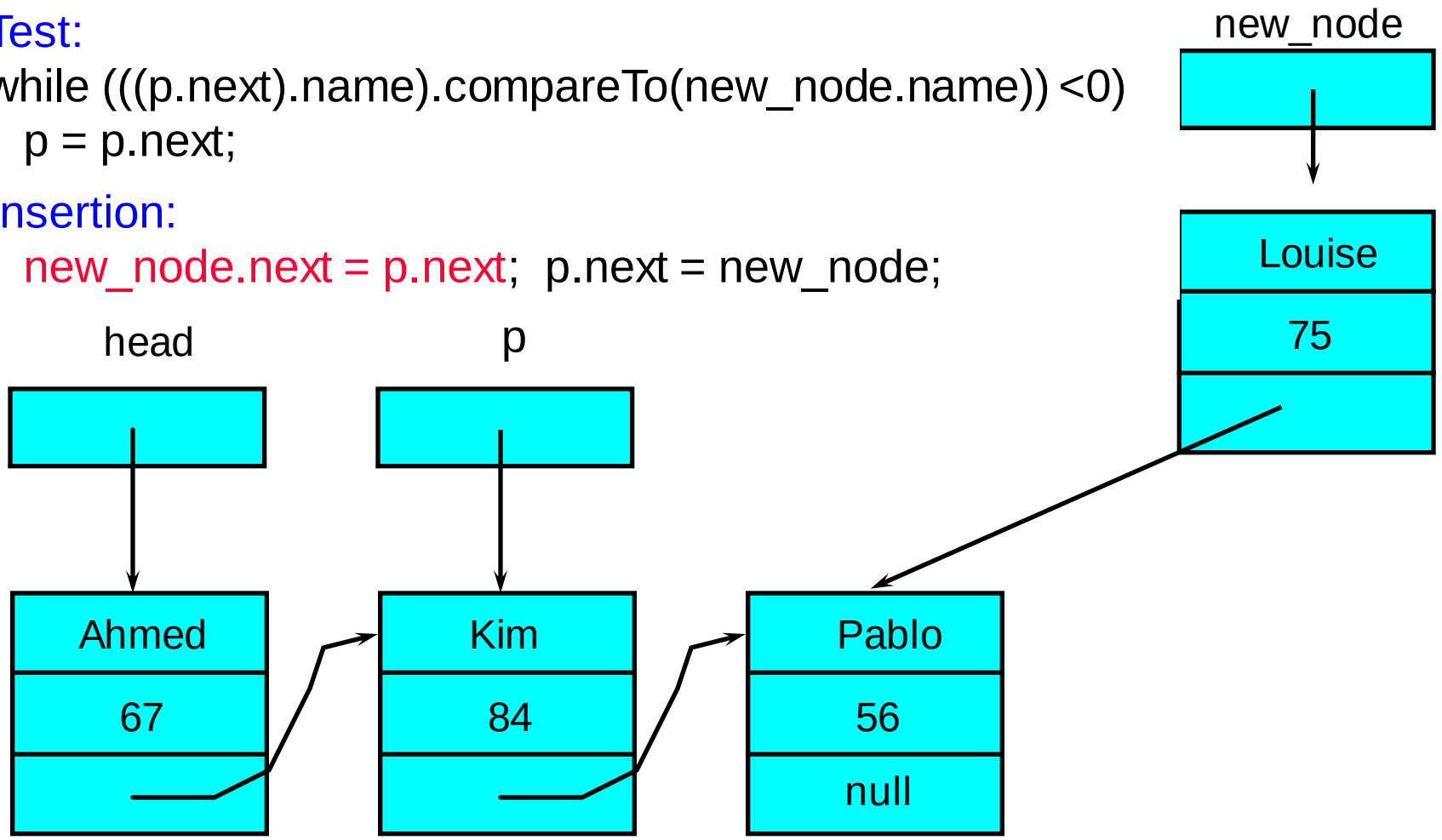
Looking one ahead in the list

Test:

```
while (((p.next).name).compareTo(new_node.name)) <0)
   p = p.next;
```

Insertion:

```
   new_node.next = p.next;  p.next = new_node;
```



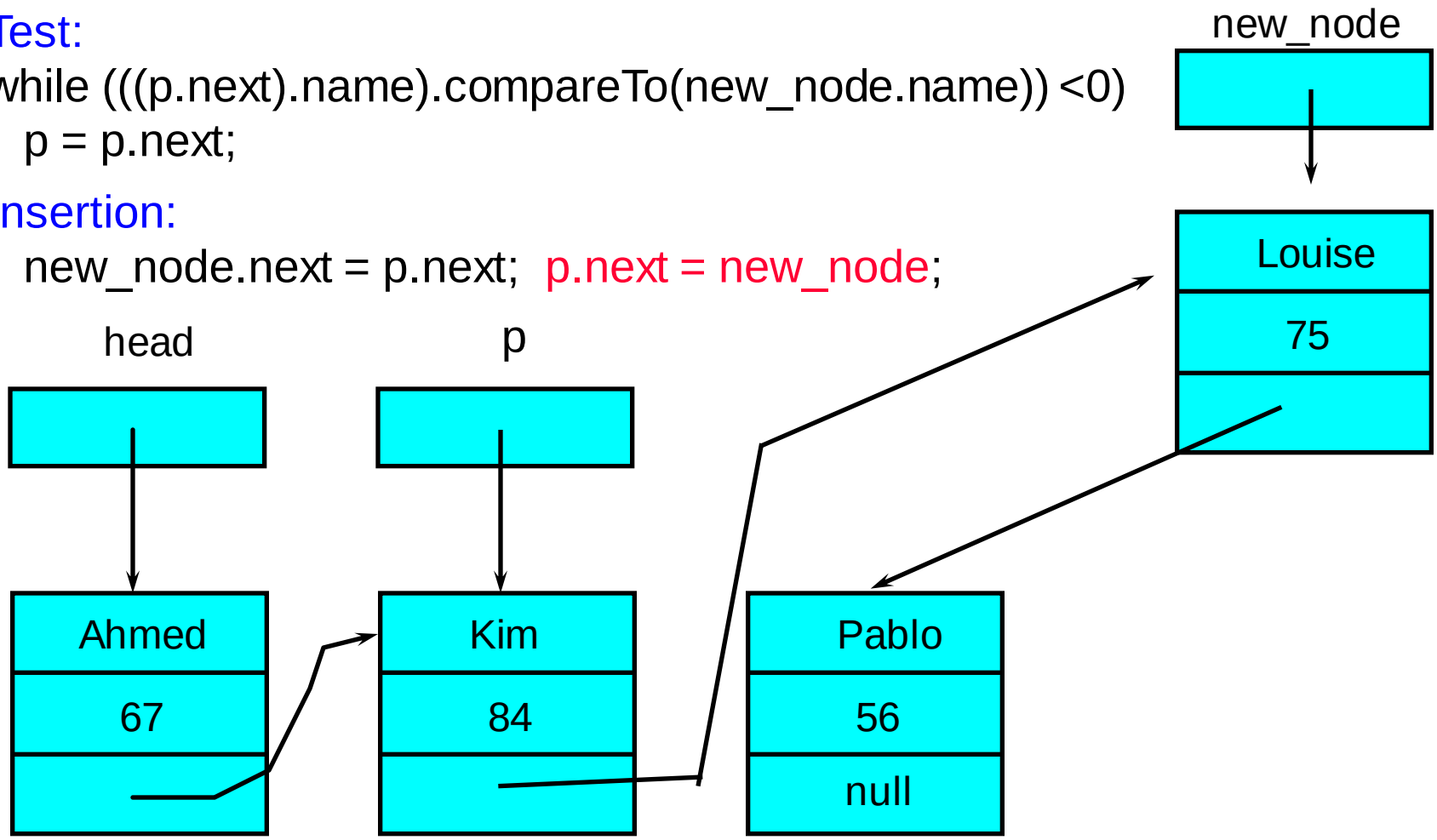
Looking one ahead in the list

Test:

```
while (((p.next).name).compareTo(new_node.name)) <0)
   p = p.next;
```

Insertion:

```
   new_node.next = p.next;  p.next = new_node;
```



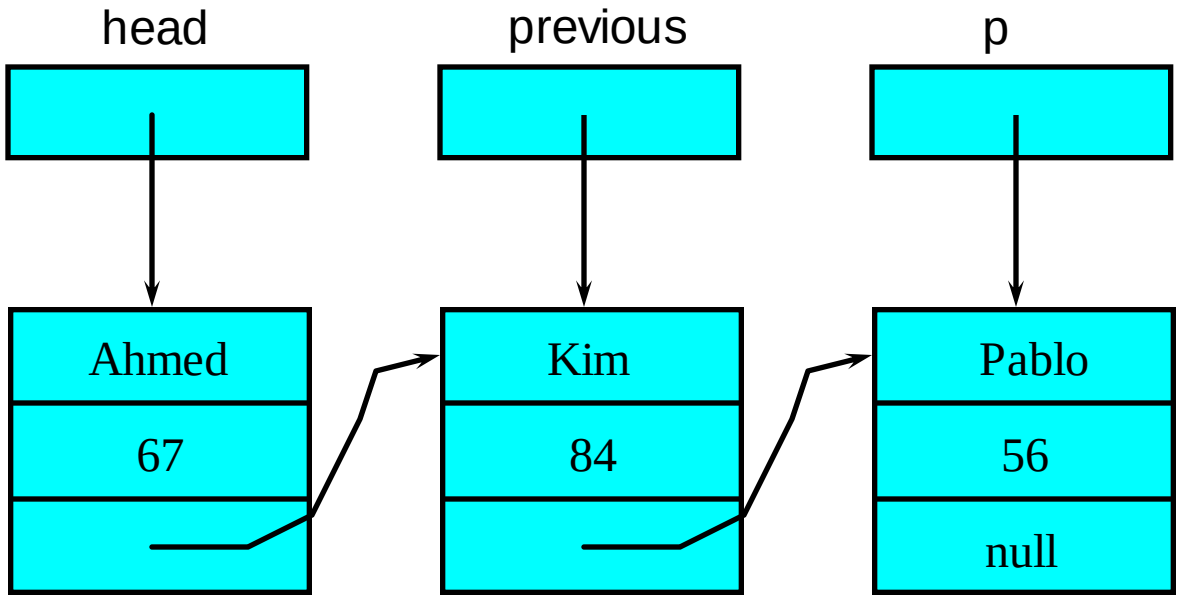
Using a trailing pointer

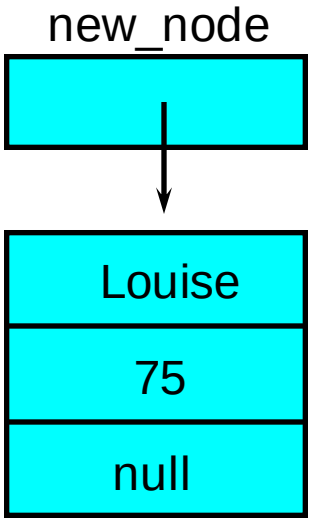
Test:

```
while ((p.name).compareTo(new_node.name) < 0)
{ previous = p;  p = p.next; }
```

Insertion:

```
new_node.next = p;   previous.next = new_node;
```

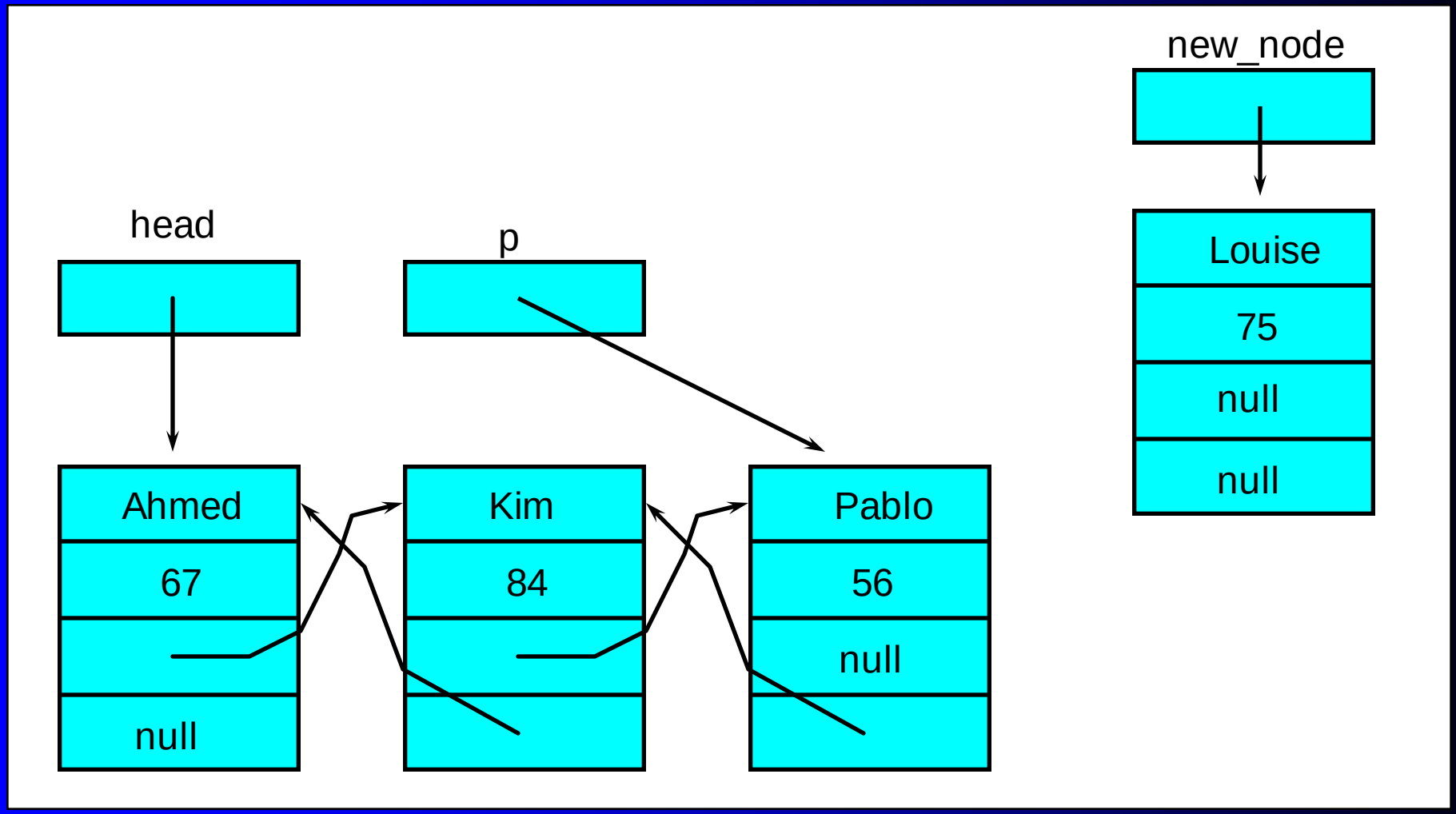




Problem ?

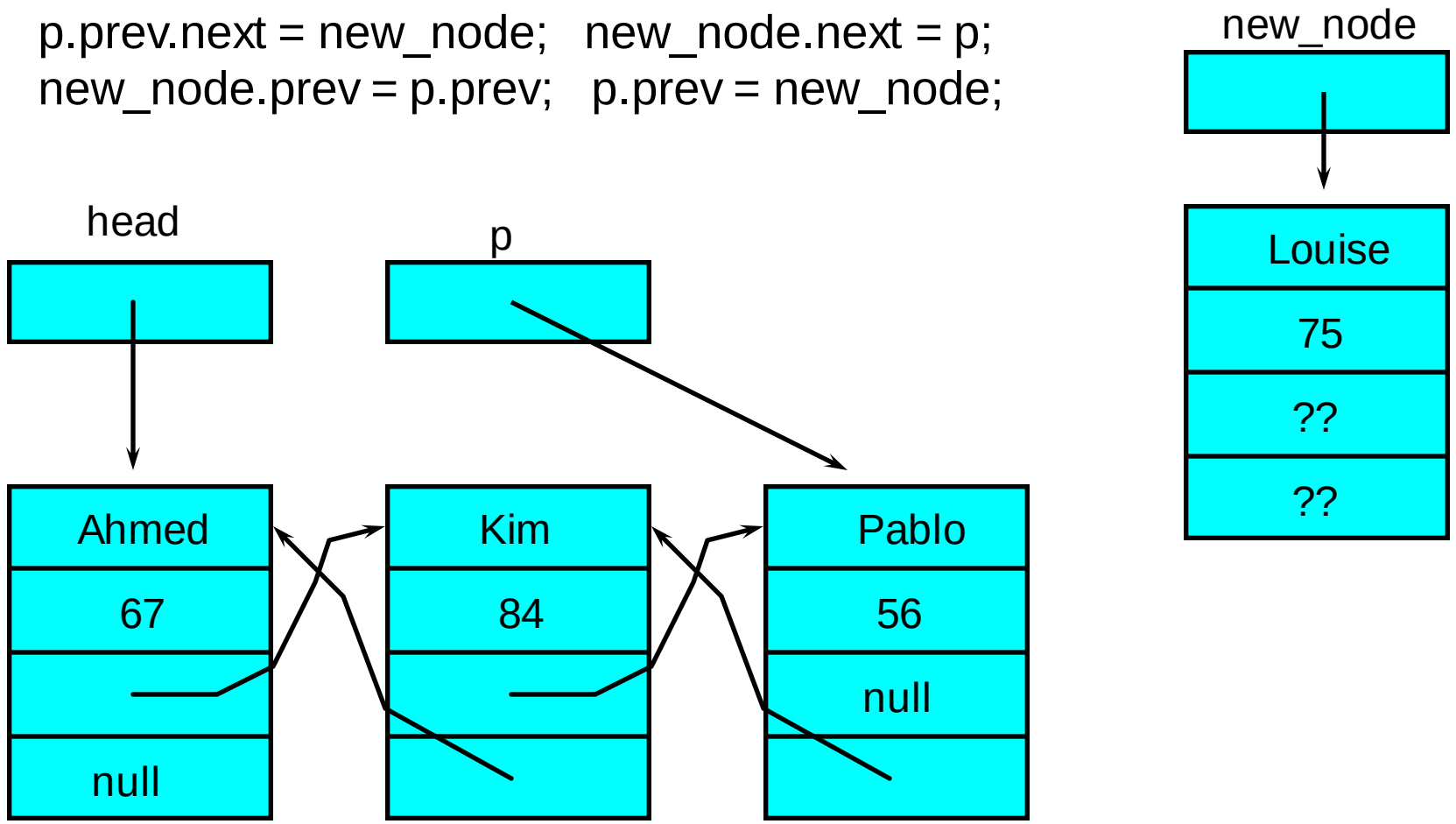
- What assumption was made in the previous four slides in the while loop?
- How could this be a problem?
- How would you make it safe?

Doubly-linked list



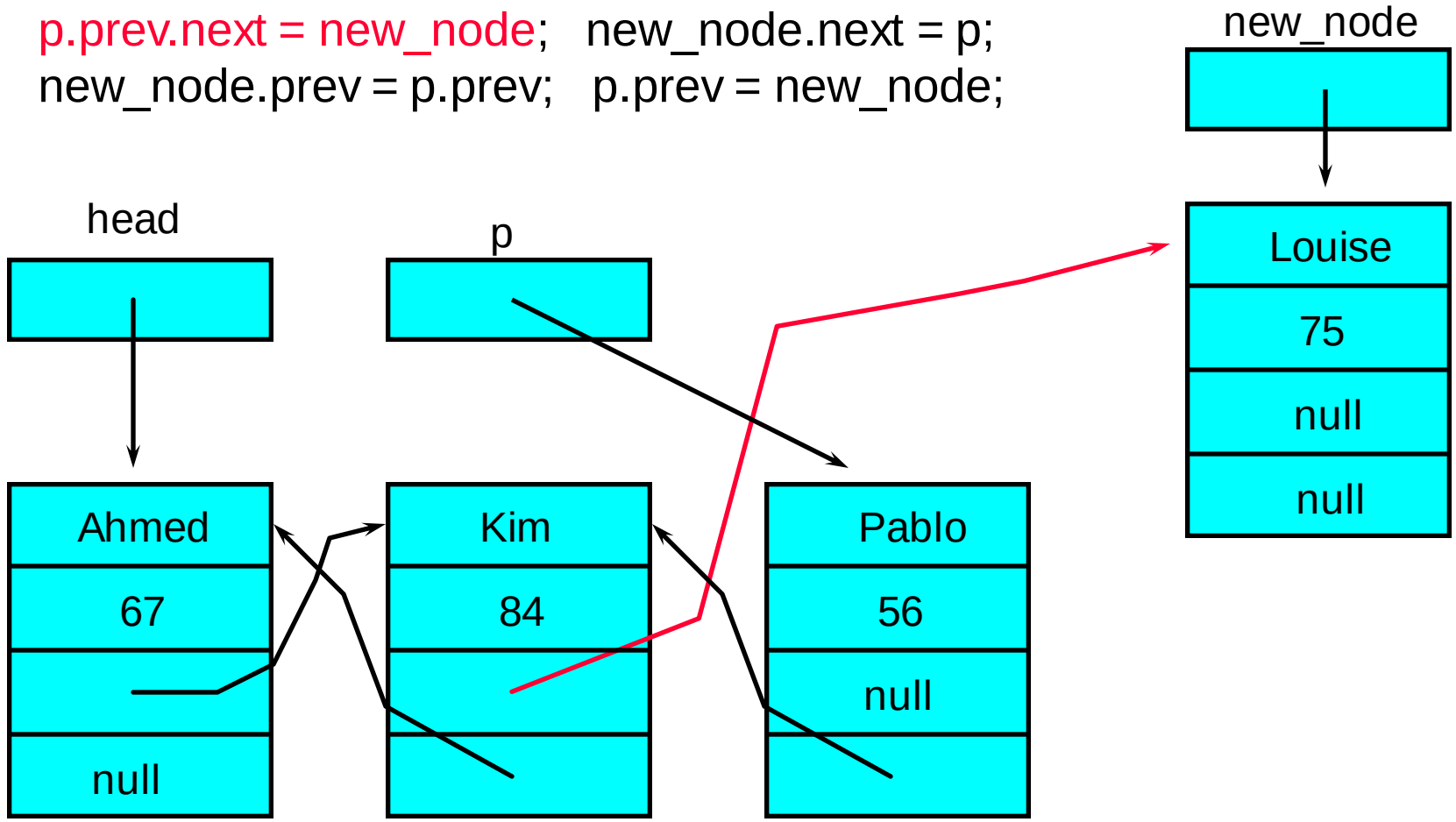
Doubly-linked list

p.prev.next = new_node; new_node.next = p;
new_node.prev = p.prev; p.prev = new_node;



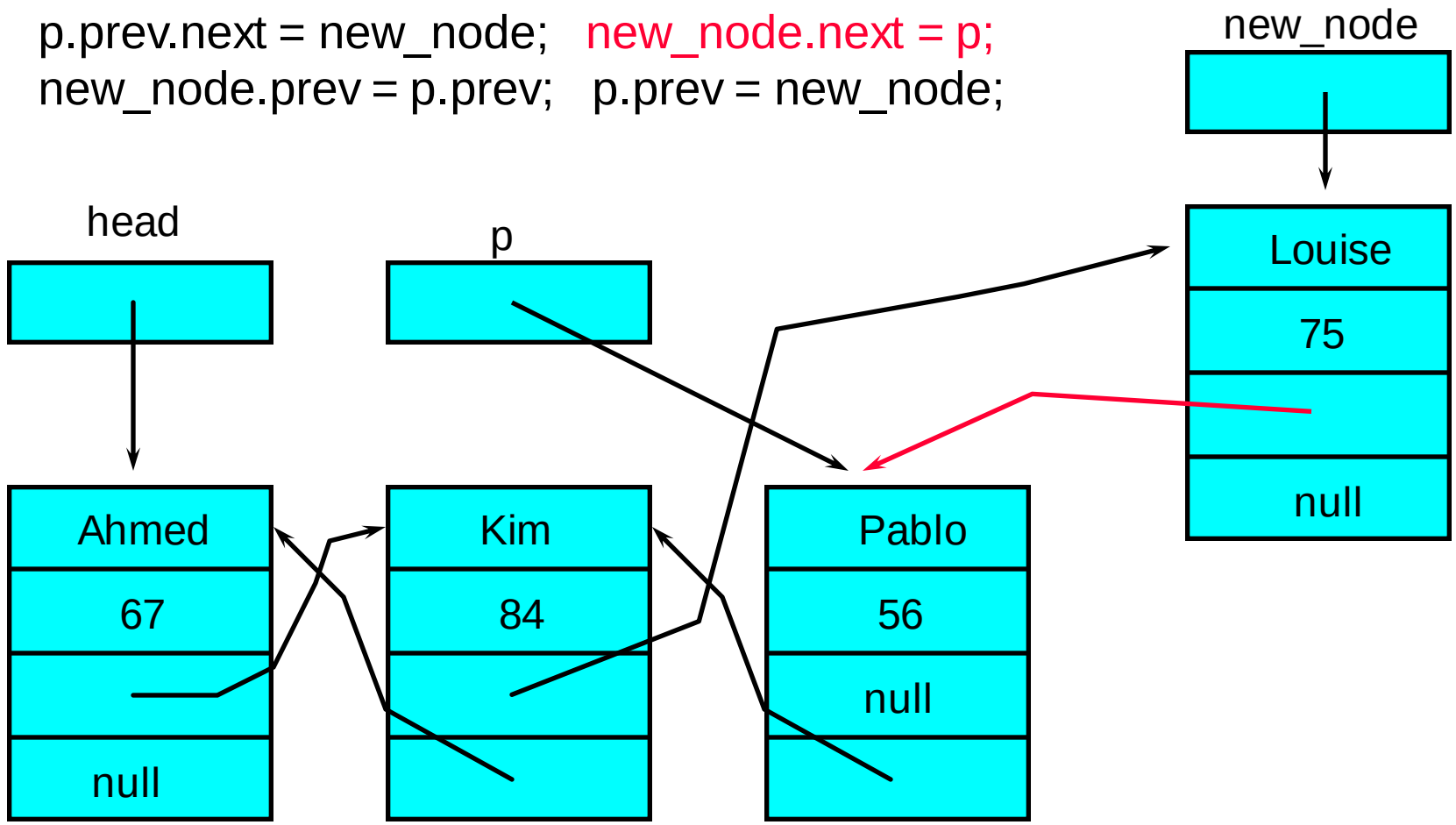
Doubly-linked list

p.prev.next = new_node; new_node.next = p;
new_node.prev = p.prev; p.prev = new_node;

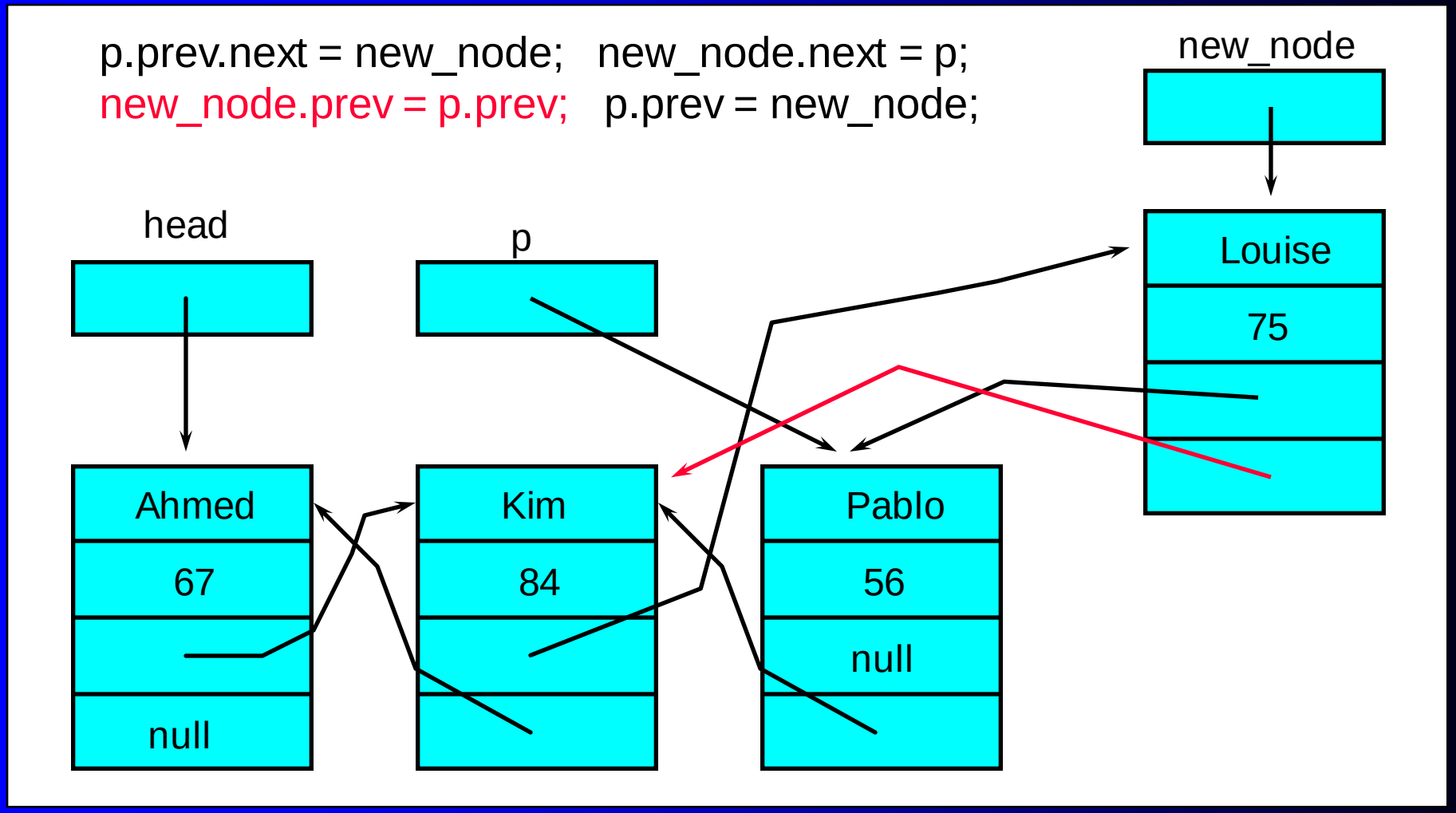


Doubly-linked list

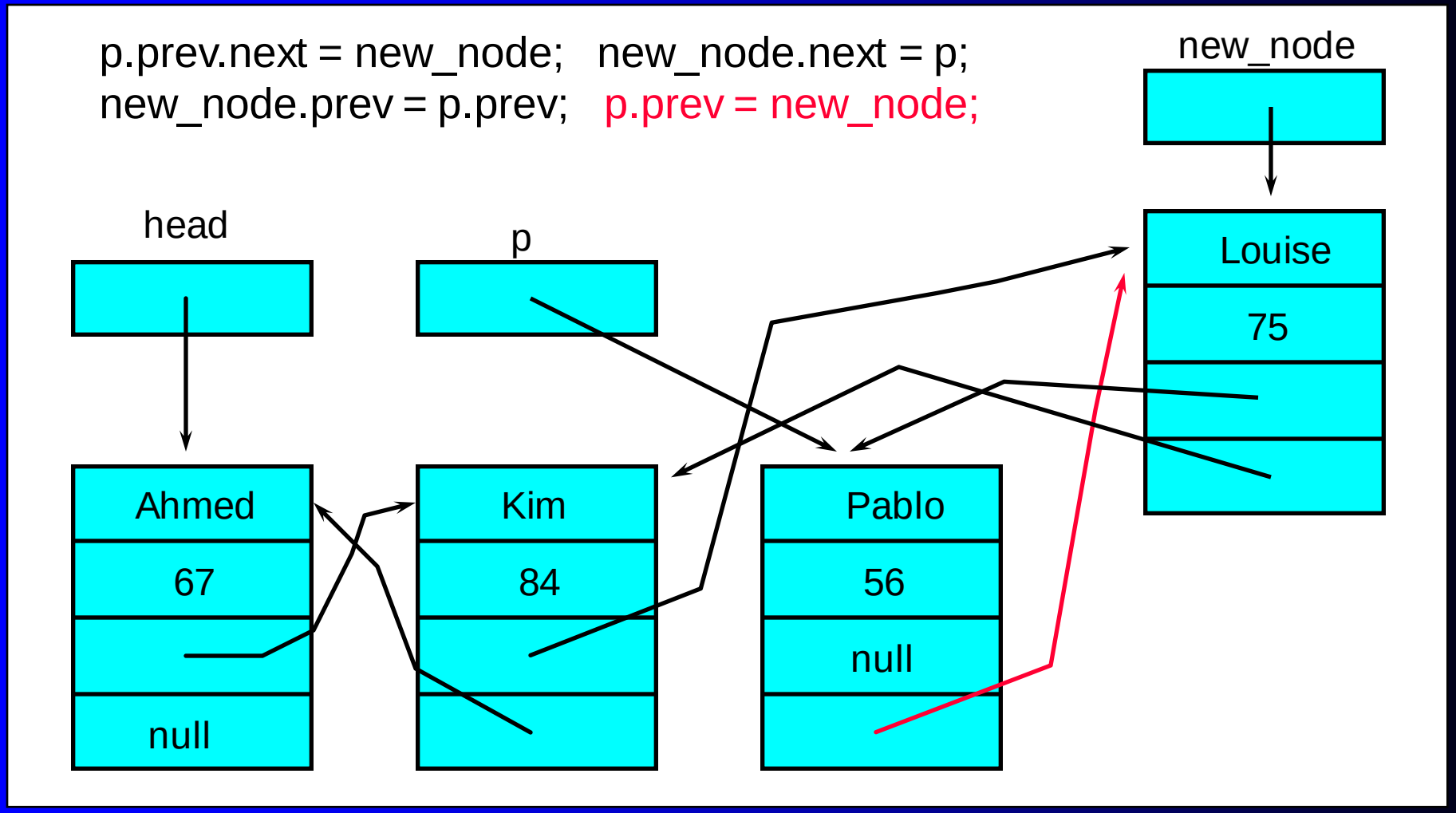
p.prev.next = new_node; new_node.next = p;
new_node.prev = p.prev; p.prev = new_node;



Doubly-linked list

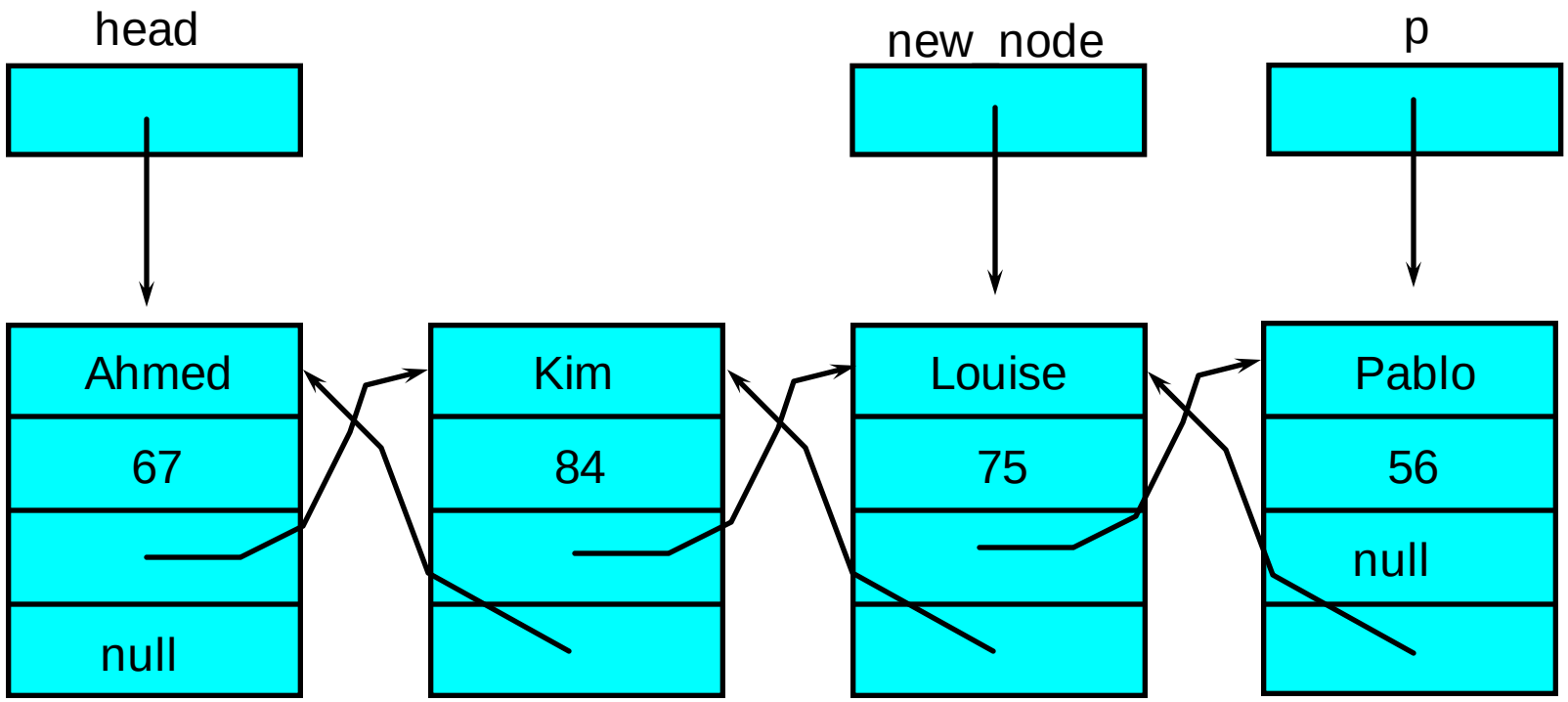


Doubly-linked list



Doubly-linked list

p.prev.next = new_node; new_node.next = p;
new_node.prev = p.prev; p.prev = new_node;



Doubly linked lists

- JAVA declaration

```
class NewStudentNode {
  private String name;
  private int mark;
  private NewStudentNode next, prev;
  public NewStudentNode(String _n, int _m) {
            name = _n;   mark = _m;
            next = null;  prev = null;
  }
  ... ...
}
```

Doubly linked lists

```
public class DoublyLinkedList {
  private NewStudentNode head, tail;
  public void insertInOrder(String new_name, int new_mark) {...}
  public void remove(int _mark); {...}
  public void traversal() {...}
  ... ...
}
```

Doubly-linked lists

- Access in forwards and backwards directions
- Uses more space
- Requires more code when inserting and deleting nodes

Insertion in order: special cases (Singly linked lists)

- insert in an empty list
- insert at the front of the list
- insert at the end of the list

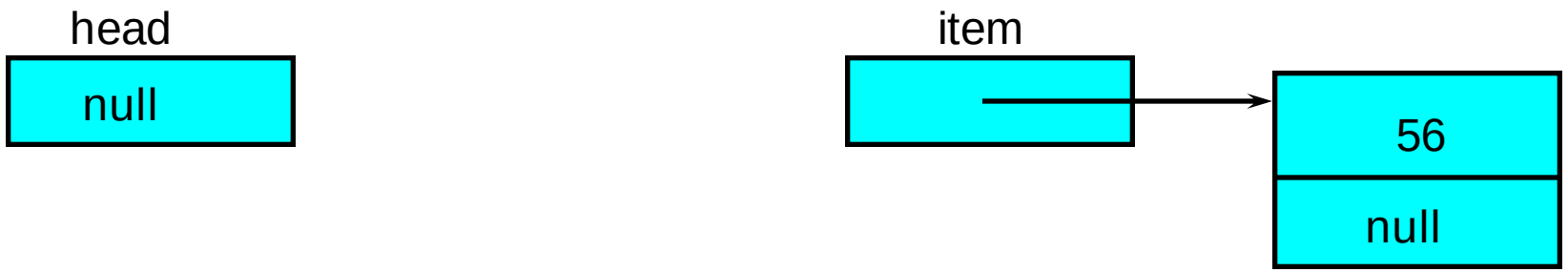
Insert spec cases (ctd)

```
void insertInOrder(Node item)
{
   if (head == null)
      head = item;
   else if (head.data  >   item.data) {
      item.next = head;
      head = item;
   }
   else {
         Node p = head;
         while ((p.next != null) && ((p.next).data < item.data))
           p = p.next;

          item.next = p.next;
          p.next = item;
   }
}
```

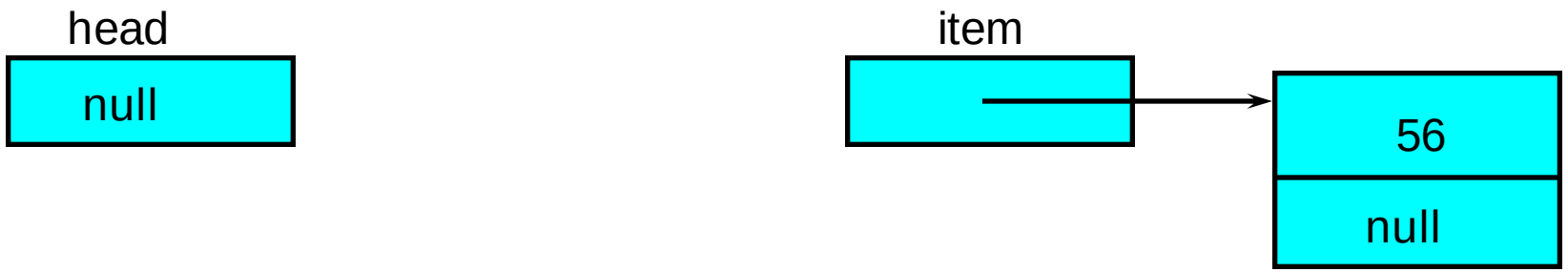
```
if (head == null) // empty list
    head = item;
```

Insert special cases (ctd)



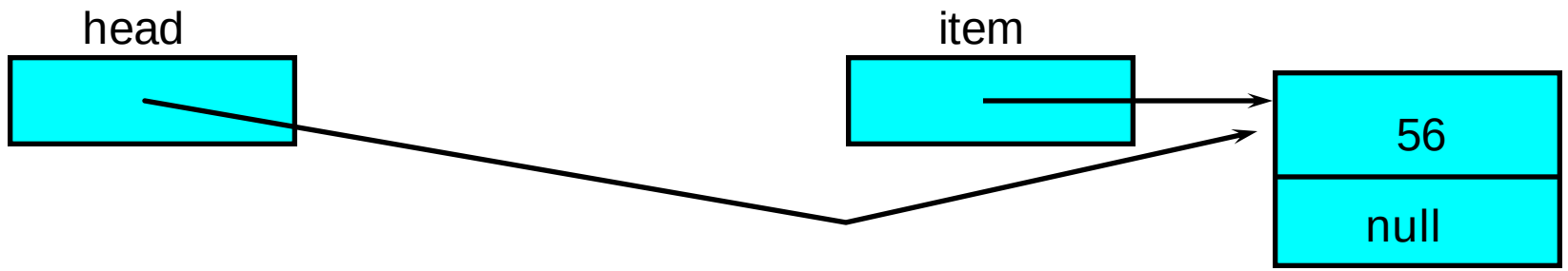
```
if (head == null) // empty list
    head = item;
```

Insert special cases (ctd)



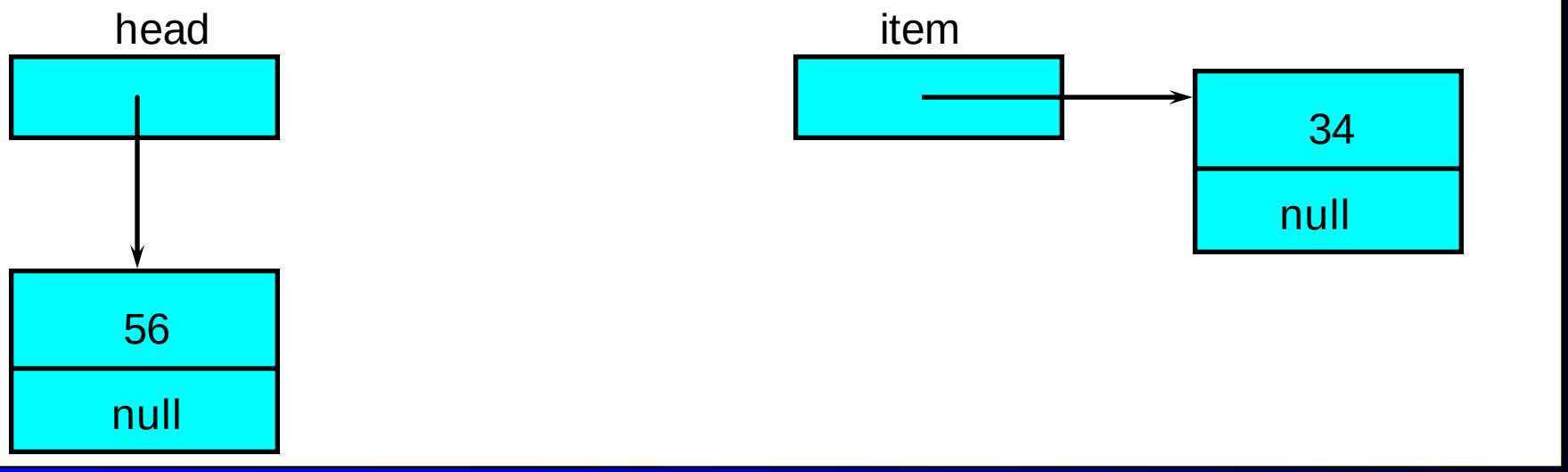
```
if (head == null) // empty list
    head = item;
```

Insert special cases (ctd)



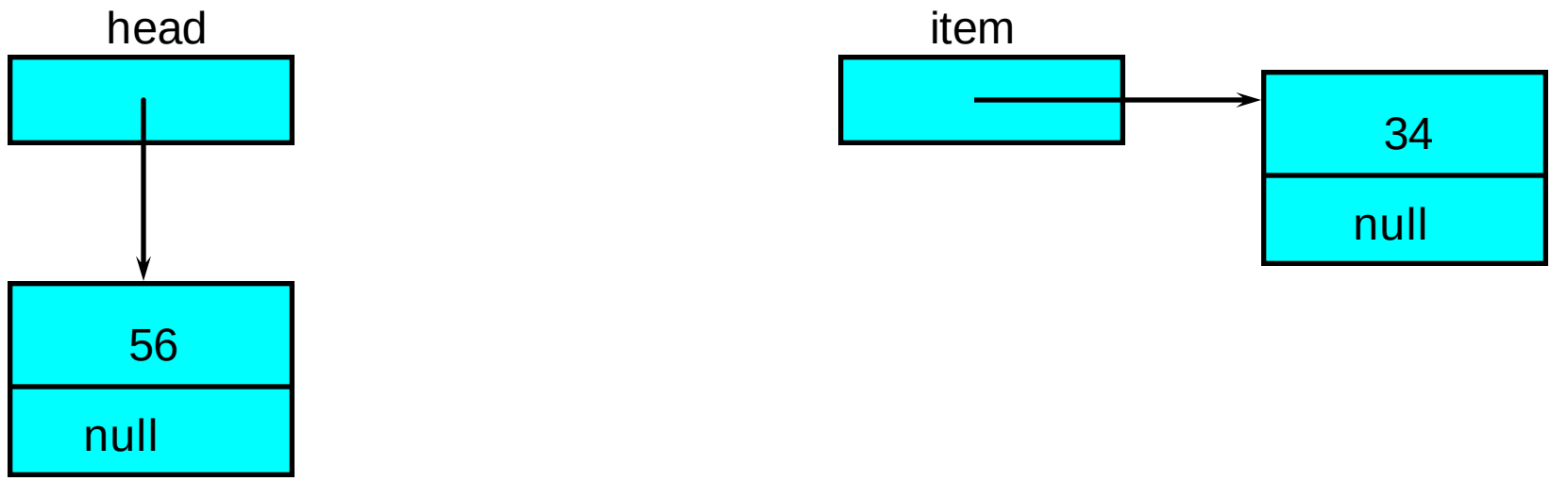
```
else if (head.data > item.data) { // insert at front of list
    item.next = head;
    head = item;
}
```

Insert special cases (ctd)



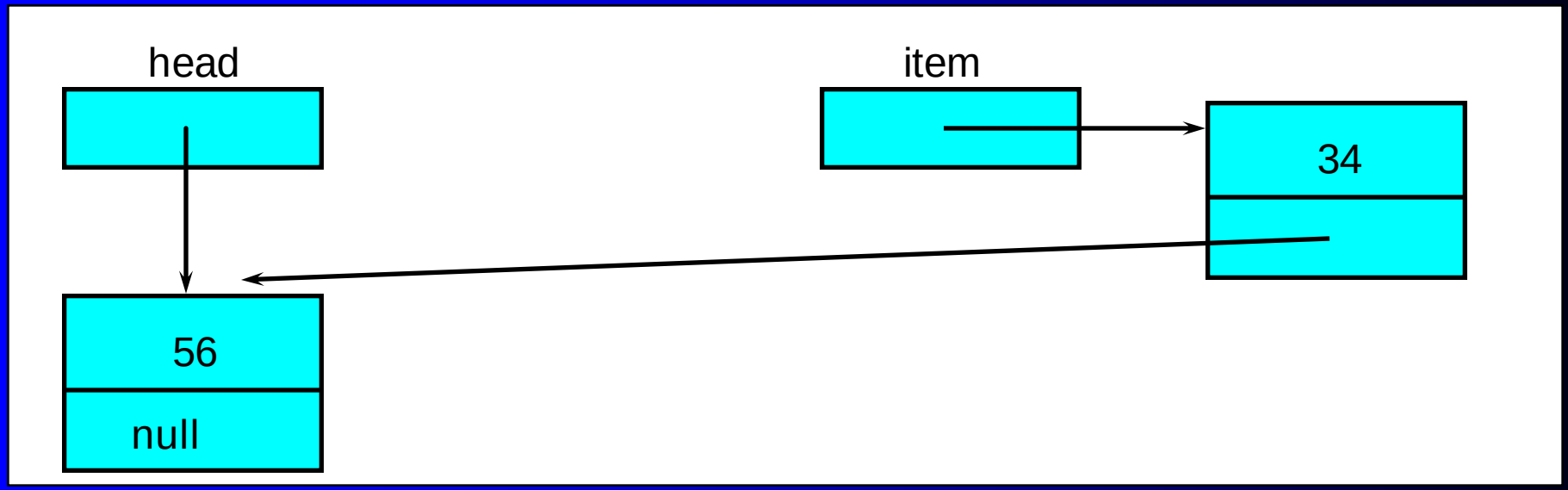
```
else if (head.data > item.data) { // insert at front of list
    item.next = head;
    head = item;
}
```

Insert special cases (ctd)



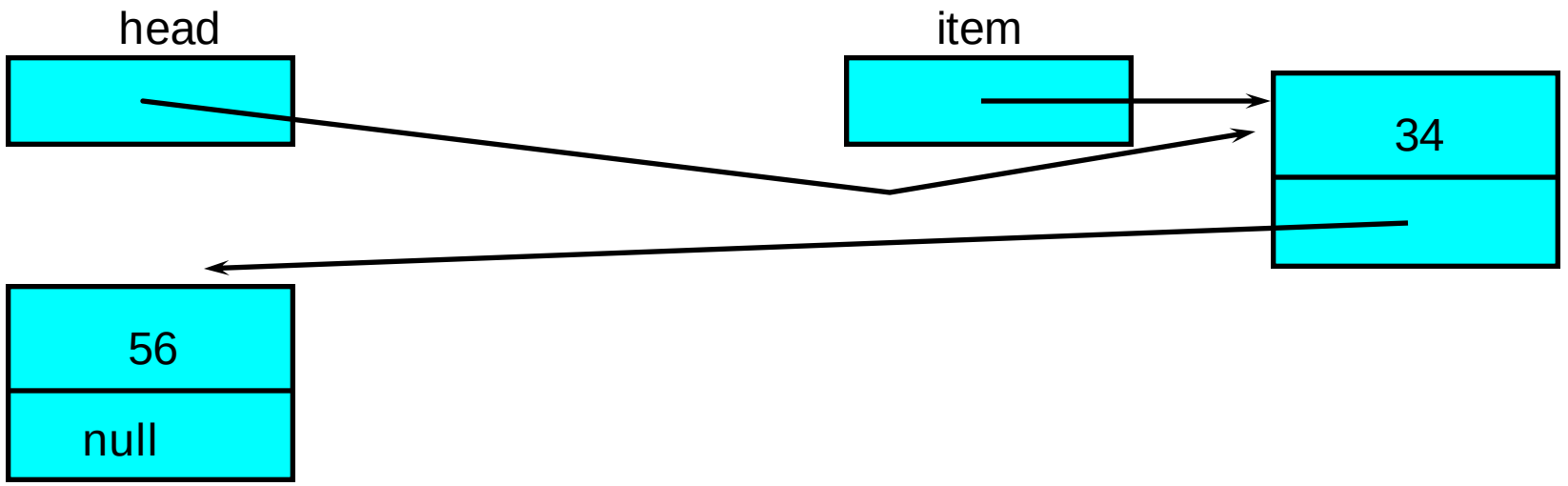
```
else if (head.data > item.data) { // insert at front of list
    item.next = head;
    head = item;
}
```

Insert special cases (ctd)



```
else if (head.data > item.data) { // insert at front of list
    item.next = head;
    head = item;
}
```

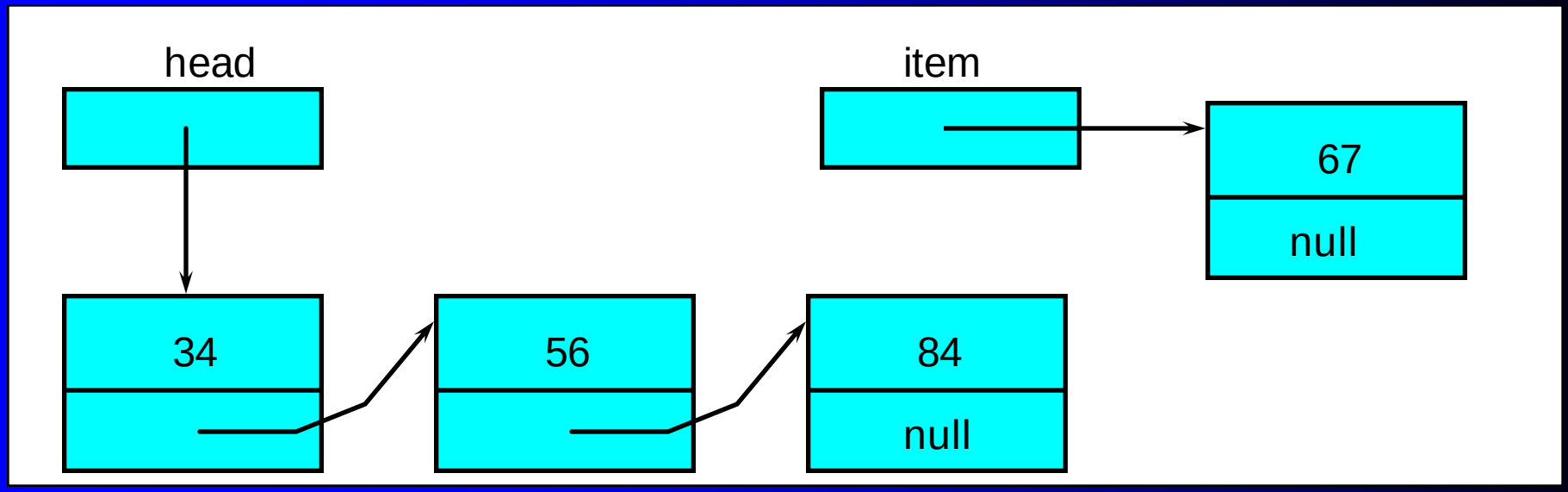
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

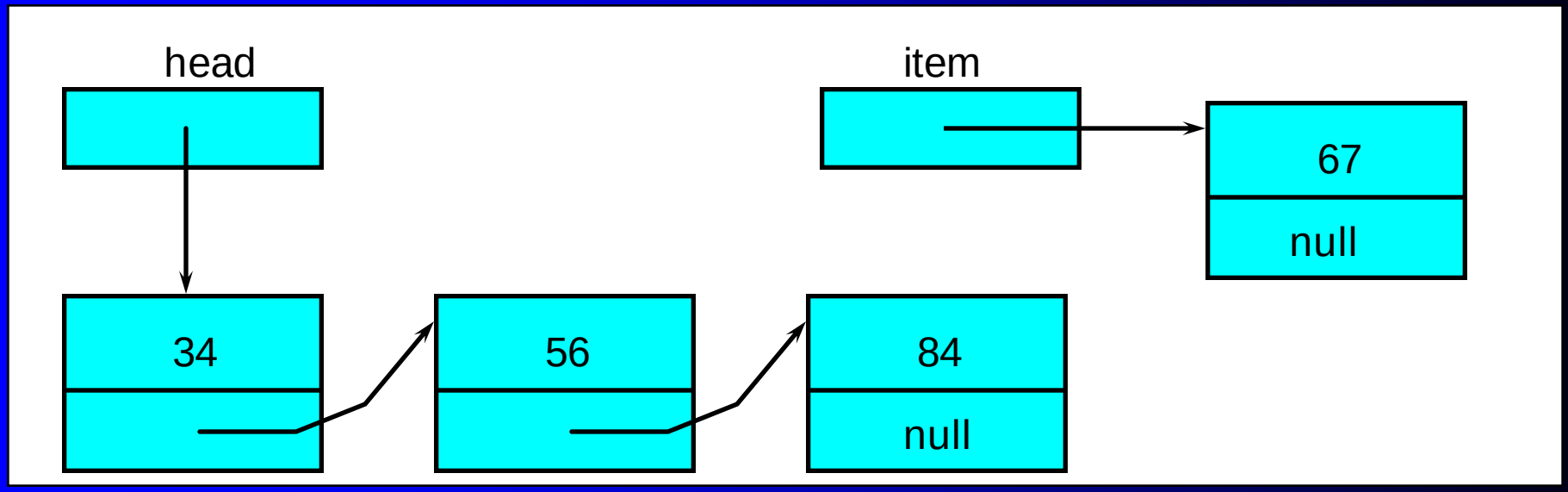
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

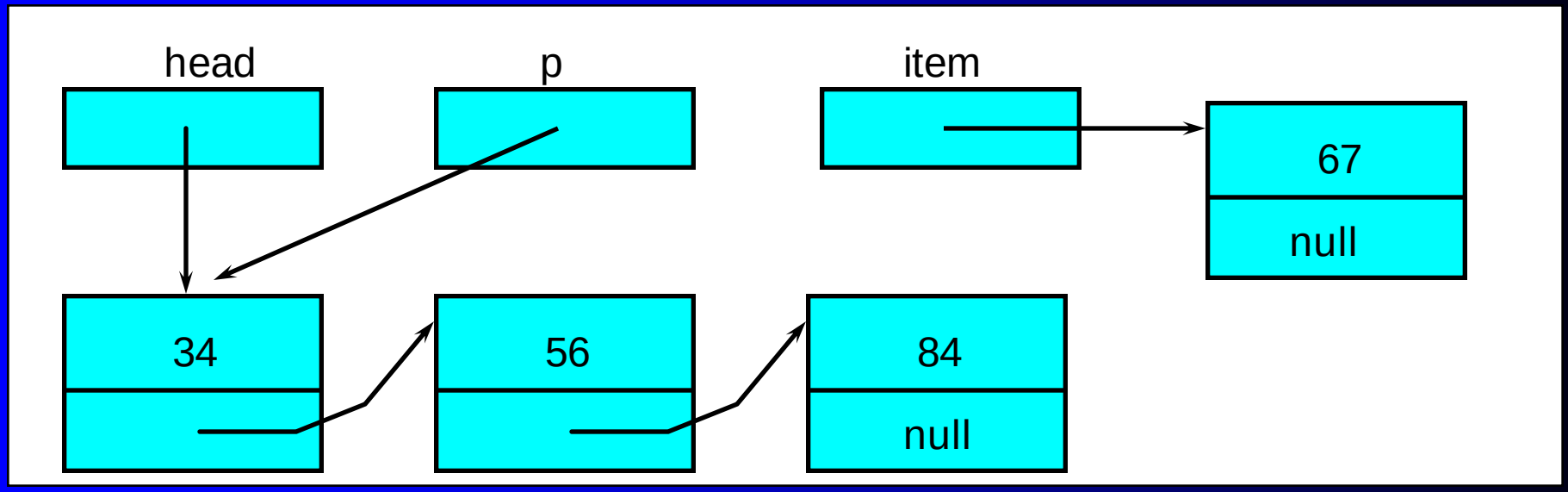
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

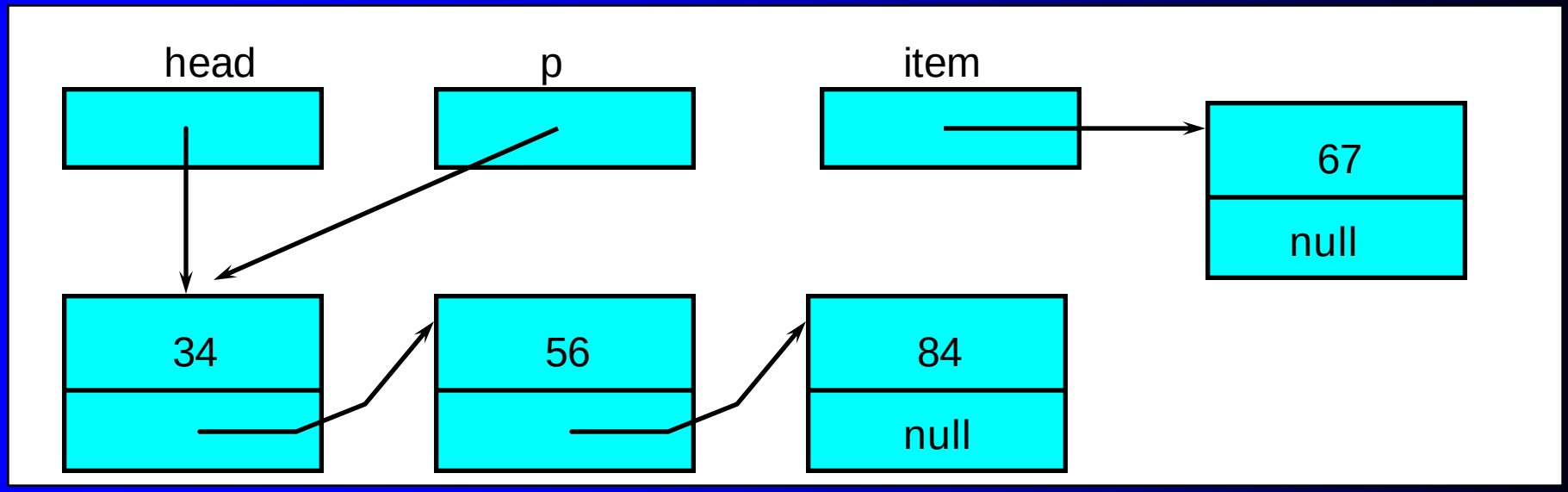
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

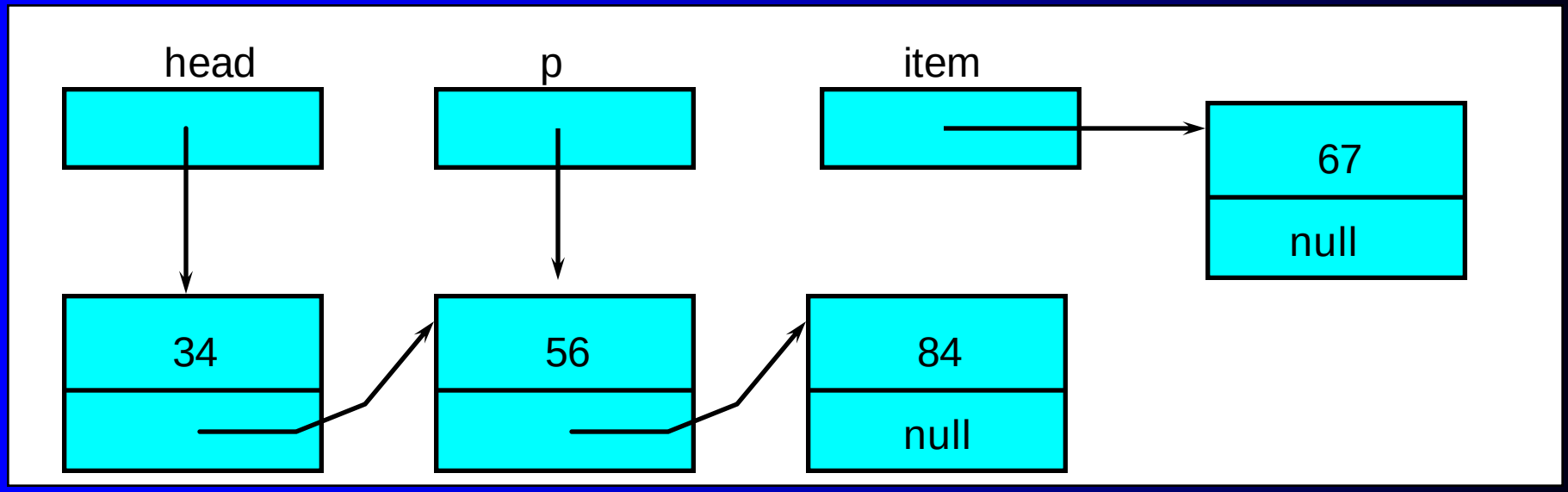
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

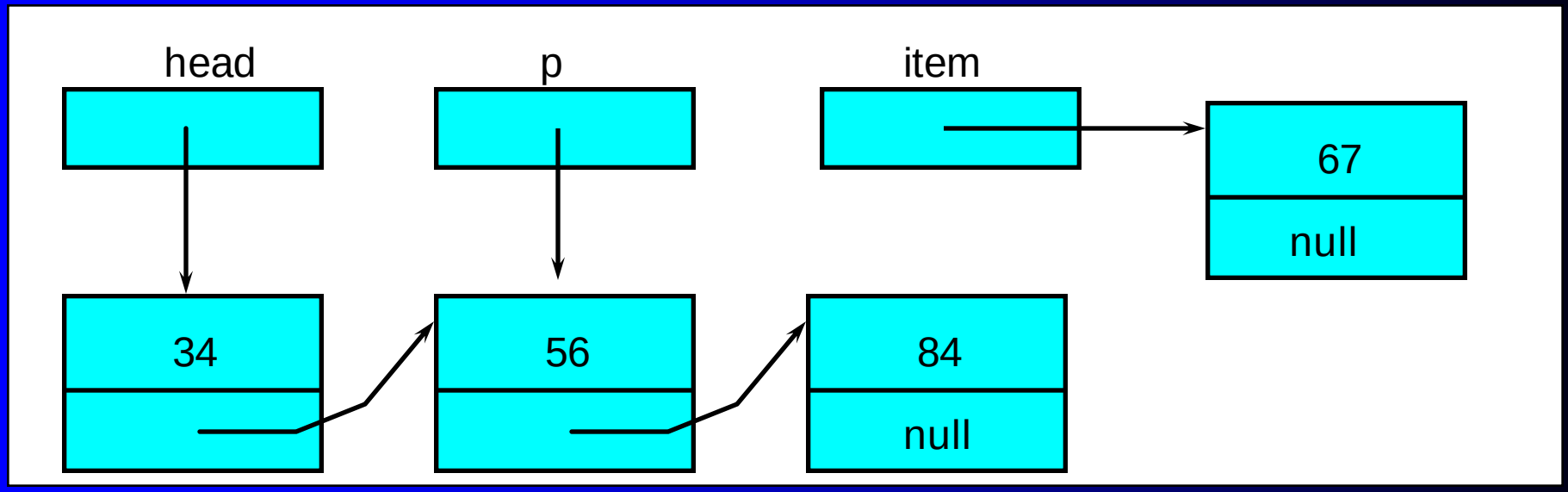
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

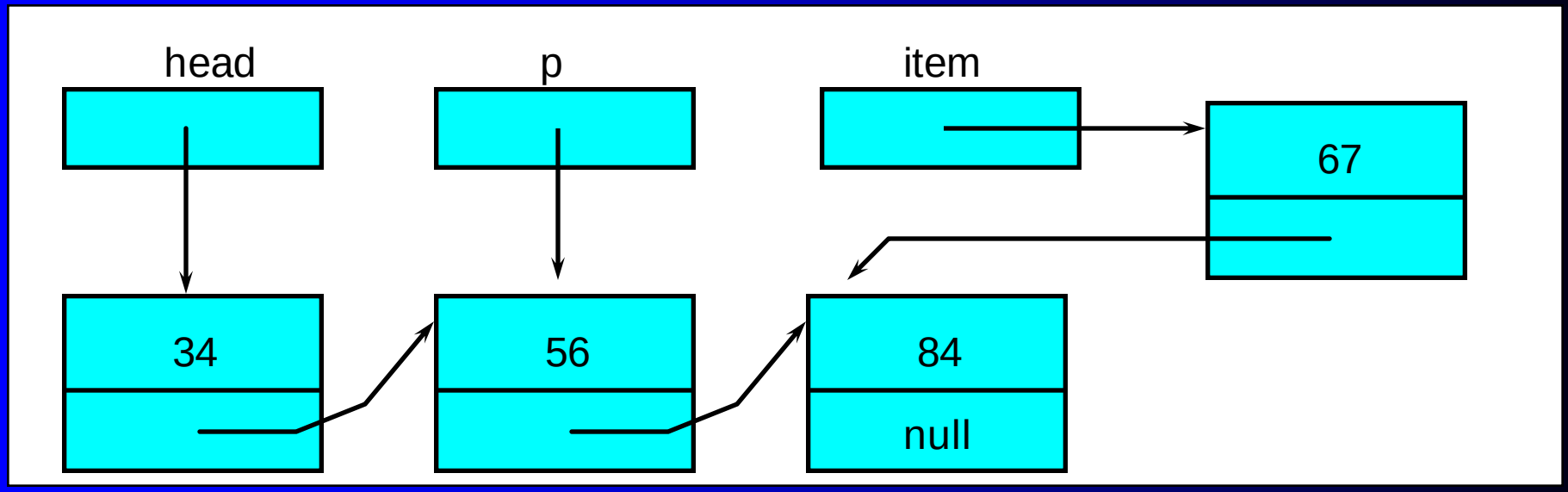
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

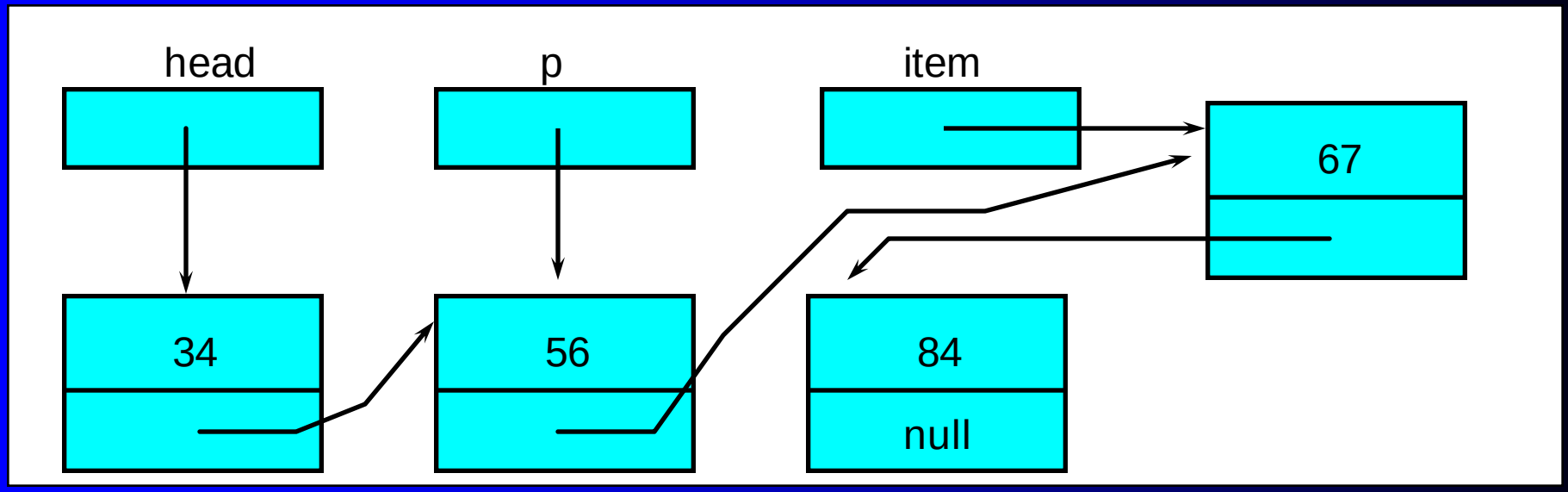
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

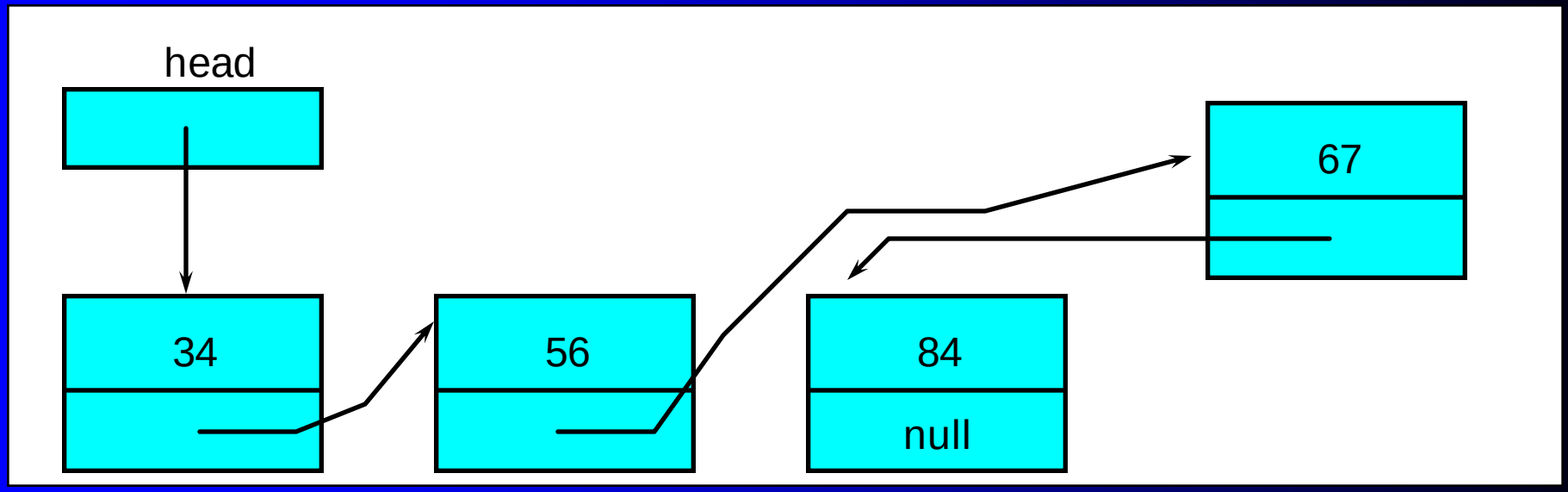
Insert special cases (ctd)



```
else {
      Node p = head;
      while ((p.next != null) && ((p.next).data < item.data))
         p = p.next;

      item.next = p.next;
      p.next = item;
}
```

Insert special cases (ctd)



Class exercise: ordered insertion

- Problem
  - work through the method insertInOrder for a list with items 34, 56 and 84, if the new item is 101
  - does the algorithm work?