Data Structures

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

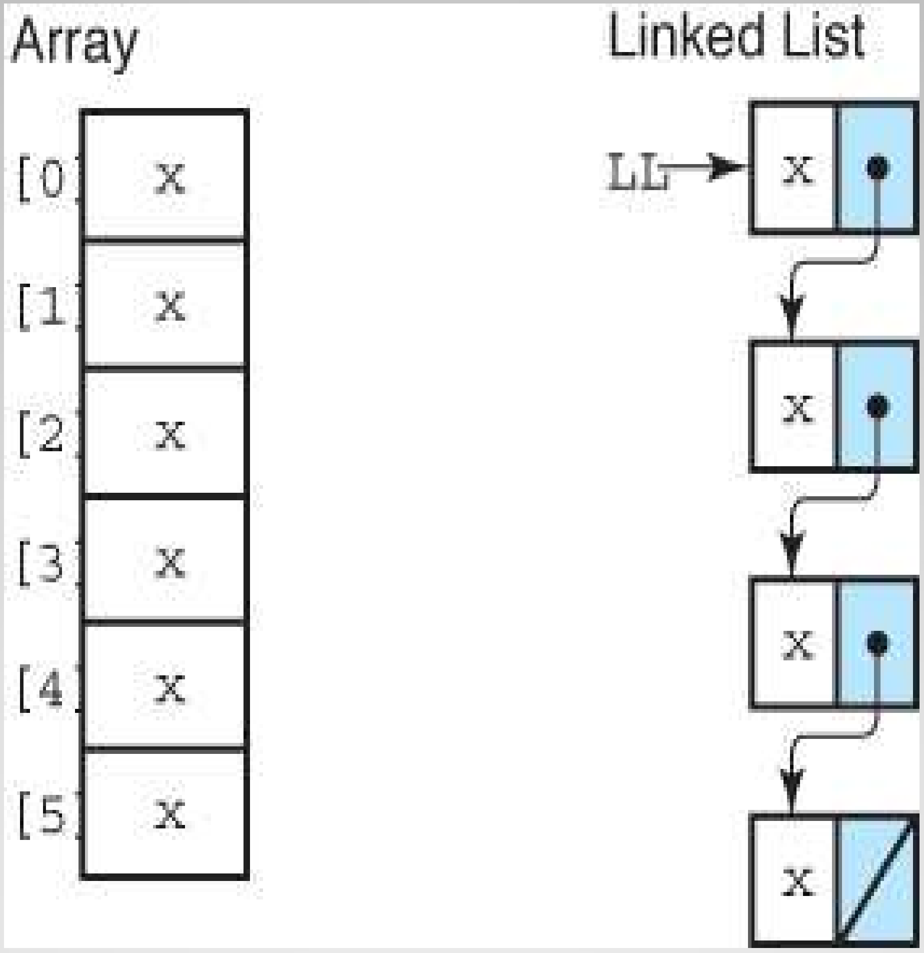
Today's Lecture

- Ordered singly-linked lists

- Lists can be implemented using representations other than an array.
- You could use a "linked list" implementation.
- "Linked list" implementation allocates memory dynamically FOR EACH element.
- We will be covering an ordered singly linked list.

Linked List

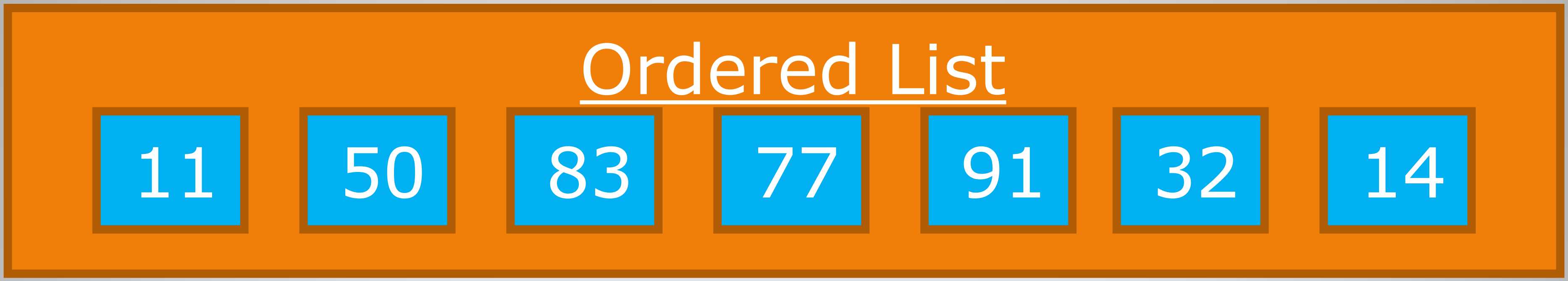
Two implementations



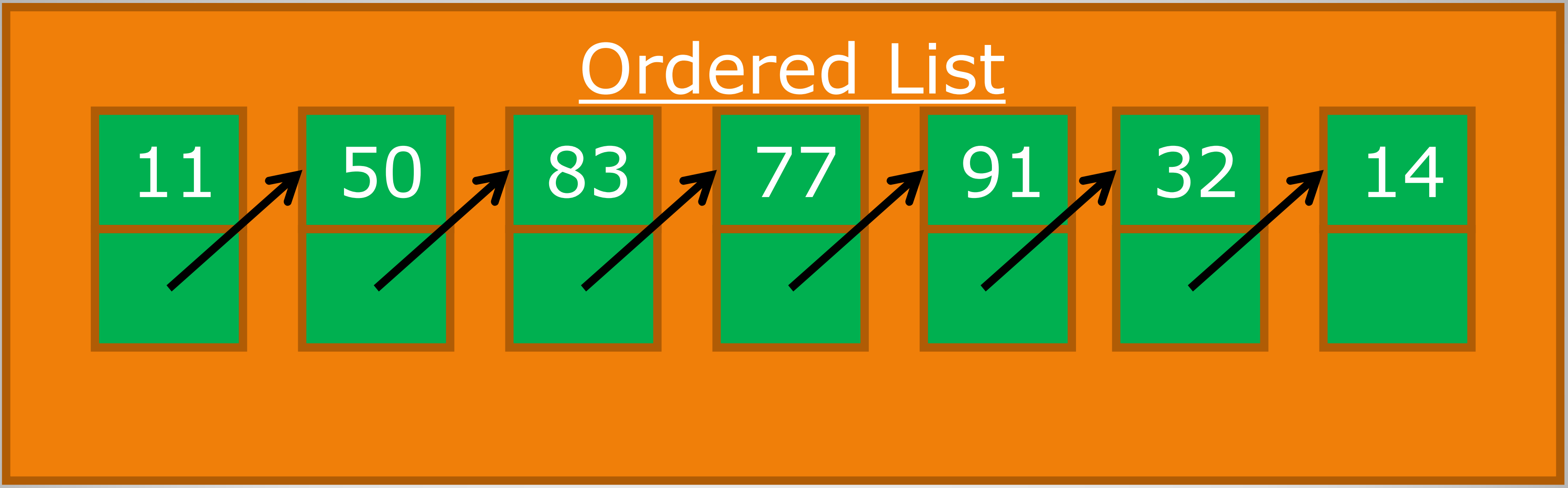
ADT Ordered List

Ordered List

- What does the ordered list look like internally when using a "linked list"?



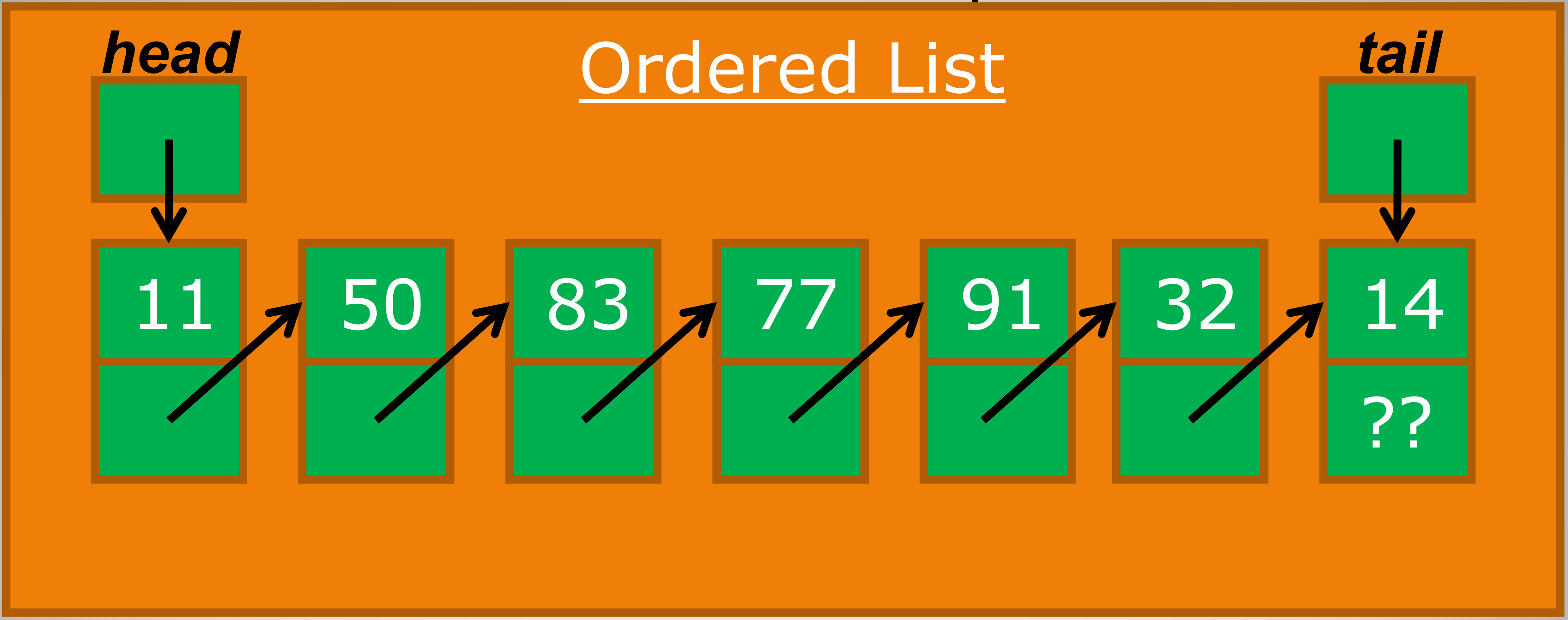
- Each element is called a "node".
- Each node has the following:
  - Data – One element of the list.
  - Pointer – Points to the next element in the list.



How do we know where the start and end of the list is?

Ordered List

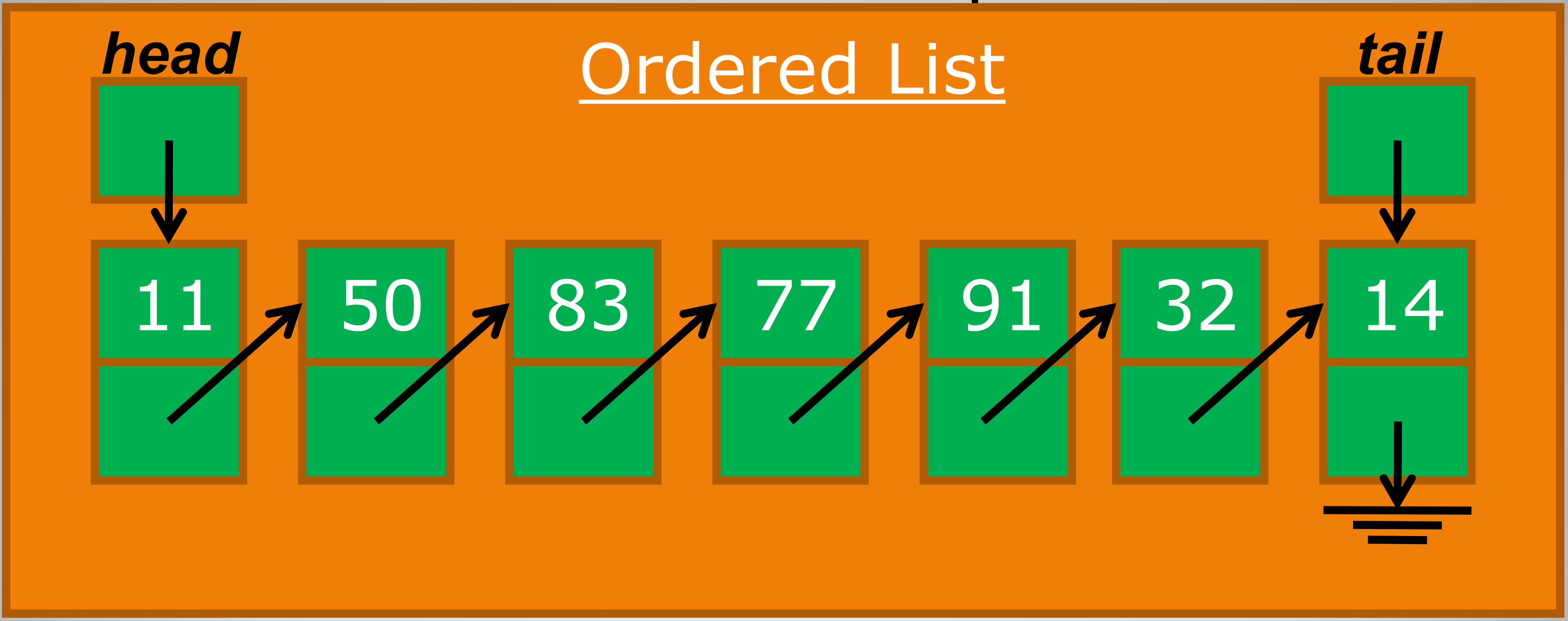
- The start of the list is the "head".
- The end of the list is the tail.
- The "head" and "tail" are pointers.



Where should the last element point to?

Ordered List

- The start of the list is the "head".
- The end of the list is the tail.
- The "head" and "tail" are pointers.

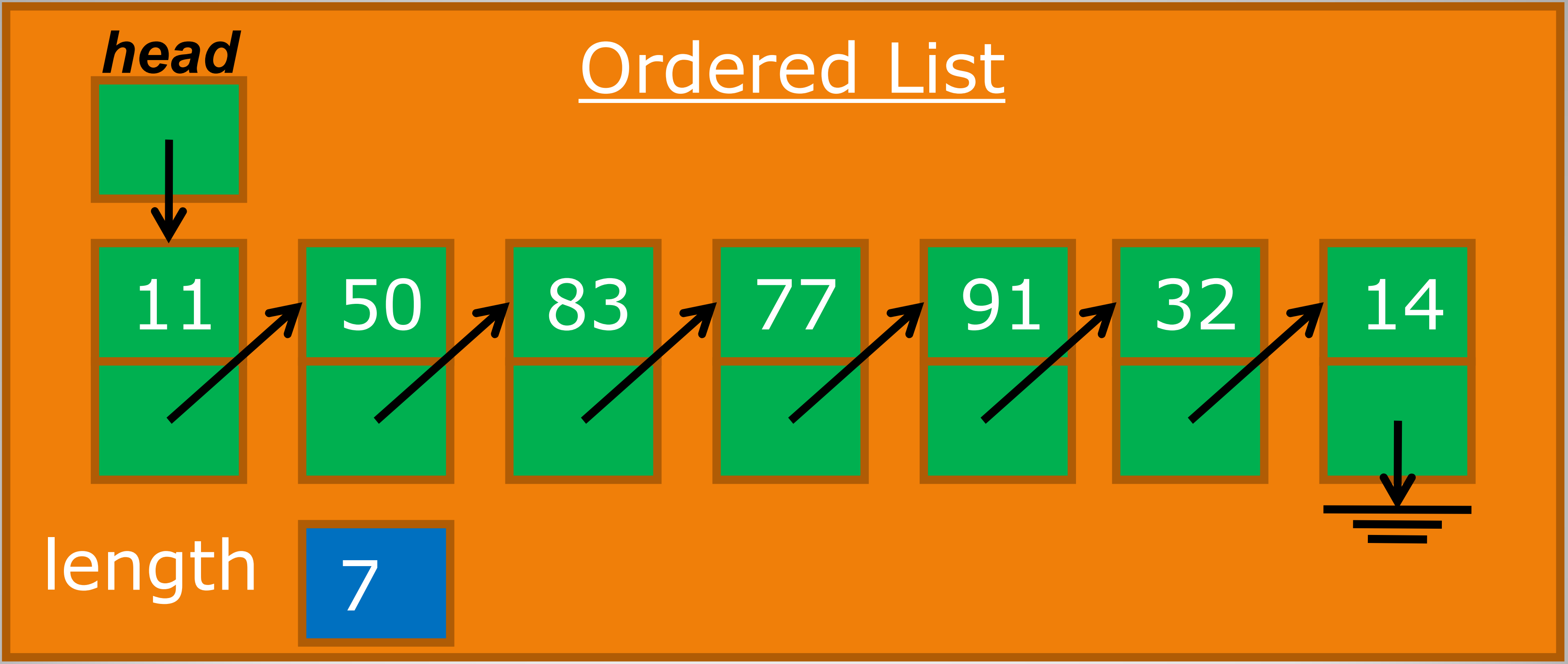


Last element should point to null.

Ordered List

Ordered List

- Our implementation - The implementation we will use only has a pointer to the first node
  - head points to the start of the list.
  - No pointer to end of list.



List Interface

- Here is the List Interface we will be using:

```
public interface List {
    public void insertItem(int item);
    public void deleteItem(int item);
    public boolean hasItem(int target);
    public int retrieveItem(int target) throws Exception;
    public void makeEmpty();
    public boolean isFull();
    public int getLength();
}
```

Note: Java has it own predefined List interface but it is more complicated, so we are using our own version.

OrderedList Class

- We will write an OrderedList class that implements our List interface.

```
public class OrderedList implements List
{
  // Implementation code goes here
}
```

- The linked-list data structure requires that we keep more information at EACH place inside of it.
- Each item in the list will be a "Node" (not just the data).
- A node stores the data and a reference to the next node
- It should be defined as an inner class within the ordered list class.

```
class Node {
    Declare int data
    Declare Node next
}
```

Data for this node (change data type as necessary to store other types of data)

Points to next node in list

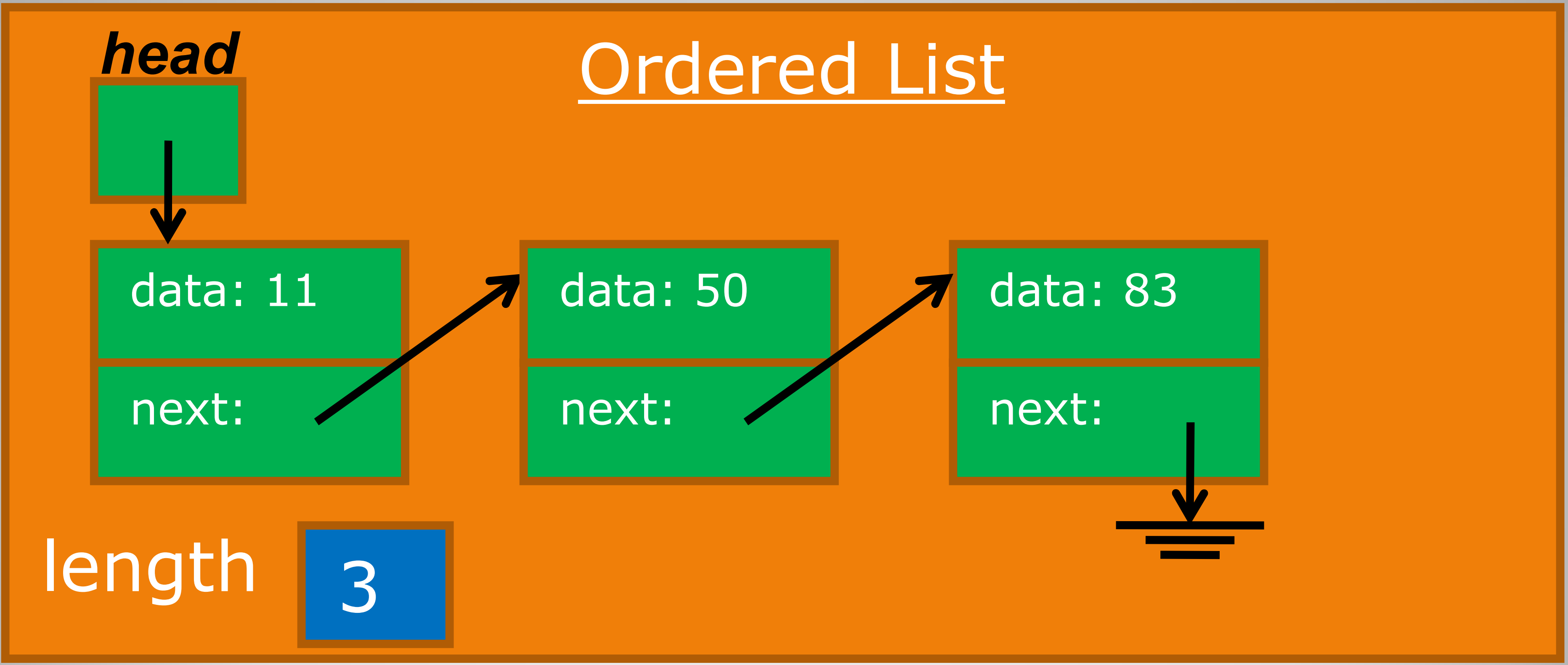
Node

OrderedList Class Member Variables

- Link-based **private** members

```
class OrderedList implements List {
    Declare Node head
    Declare int length

    // Public members go here…
}
```

- Each element of the list is of Node
- head is of type Node



Singly-Linked Ordered List

- **What should the OrderedList constructor do?**

Ordered List - Constructor

- **What should the OrderedList constructor do?**

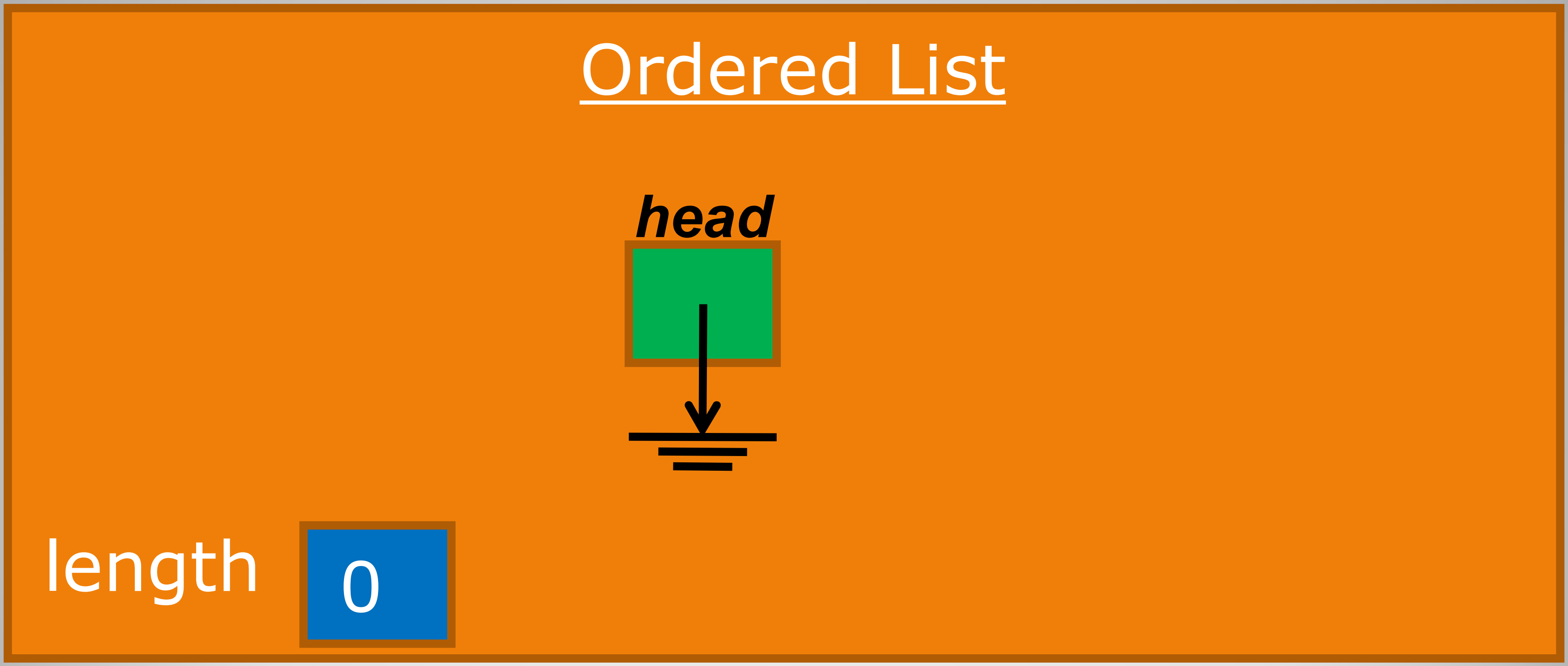
```
OrderedList Constructor
    Set length to 0
    Set head to null
```

Sets the # of element to 0

List is empty so head is null

Ordered List - Constructor

- Ordered list AFTER default constructor runs.



Ordered List - Constructor

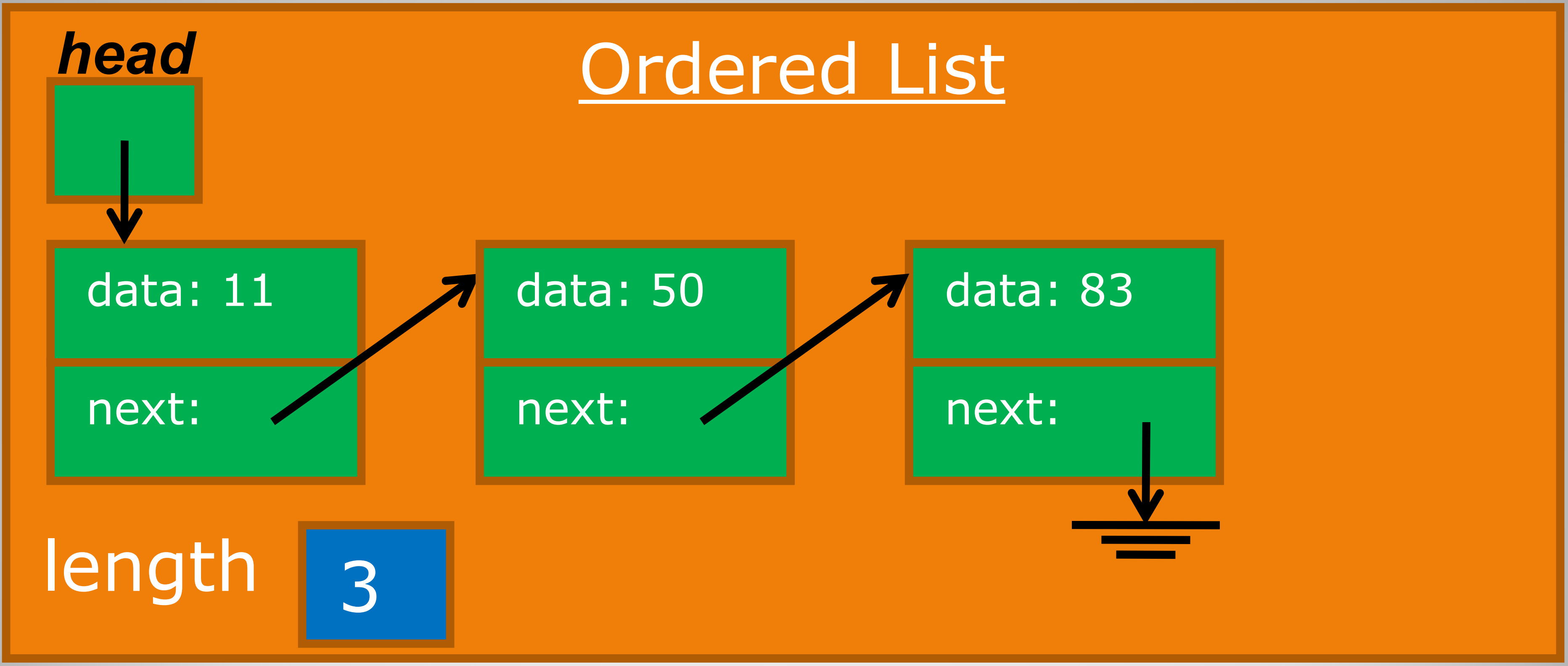
Ordered List - insertItem

How do you insert an item?

Where does it go in the list?

- Where would a new item go? How is it inserted?

ol.insertItem(77)



Ordered List - insertItem

Ordered List - insertItem

Since the list is ***ordered*** (and there are no other constraints) we can put it anywhere in the list.

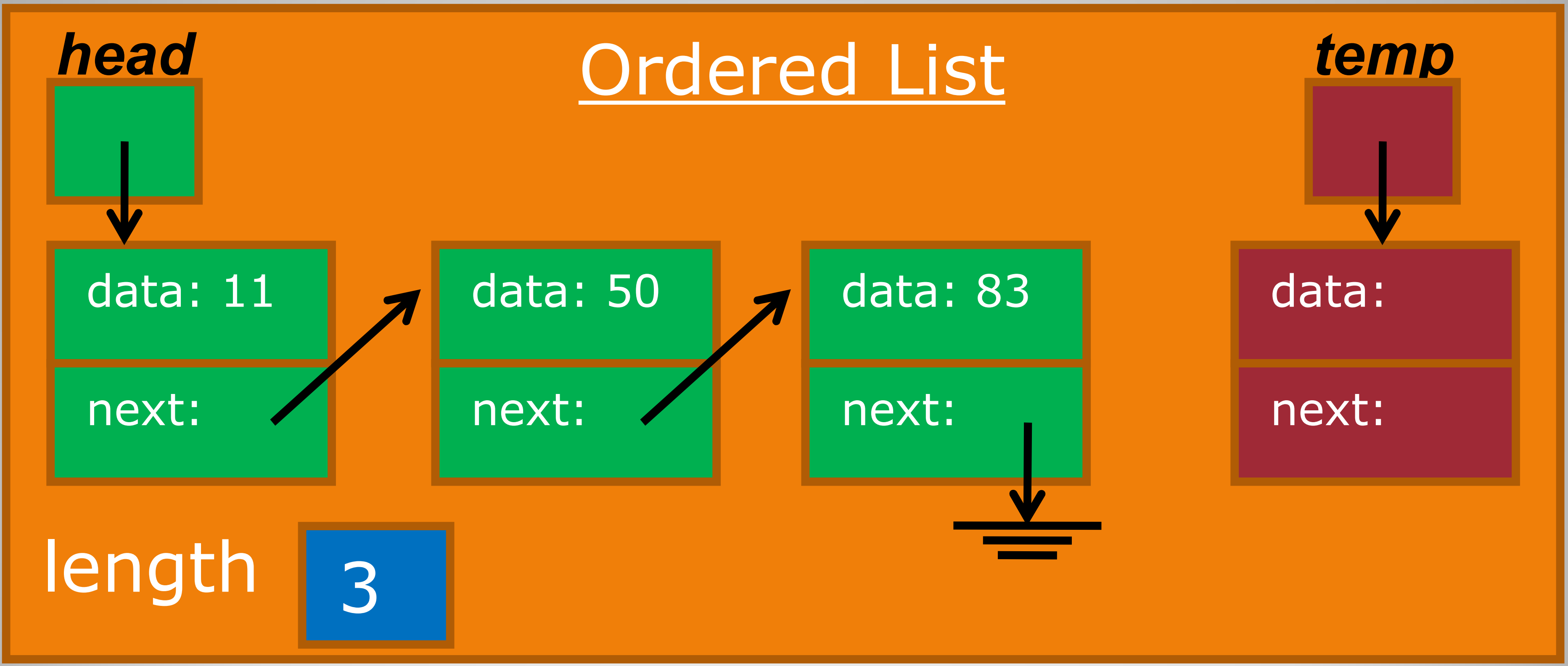
The easiest place to insert is at the beginning.

insertItem Pseudocode

1. Create a new Node instance (dynamically allocate).
2. Set the fields on the new Node. This means setting the data item and the next pointer. The next pointer should be set to the current start of the list.
3. Set the pointer to the start of the list to the new Node.
4. Increment the length of the list.

ol.insertItem(77)

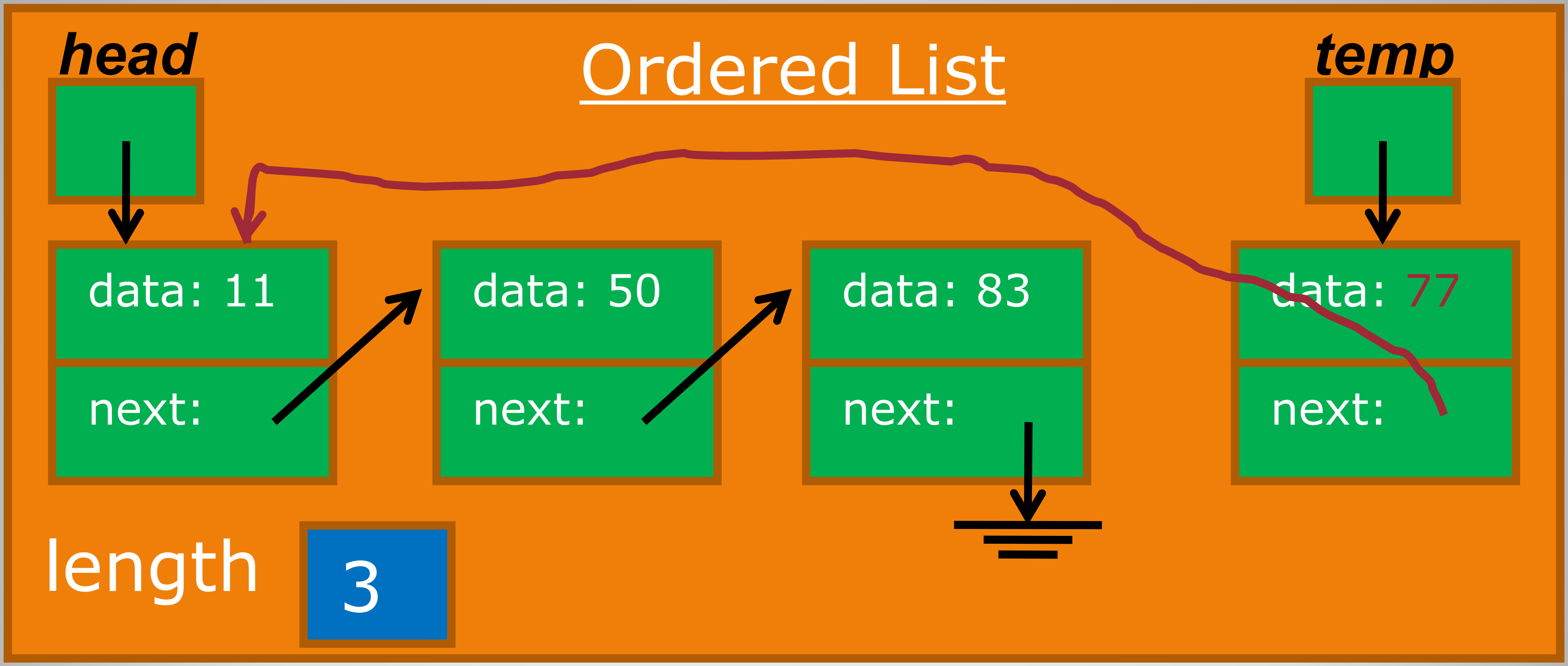
1. Create a new Node instance (dynamically allocate).



Ordered List - insertItem

ol.insertItem(77)

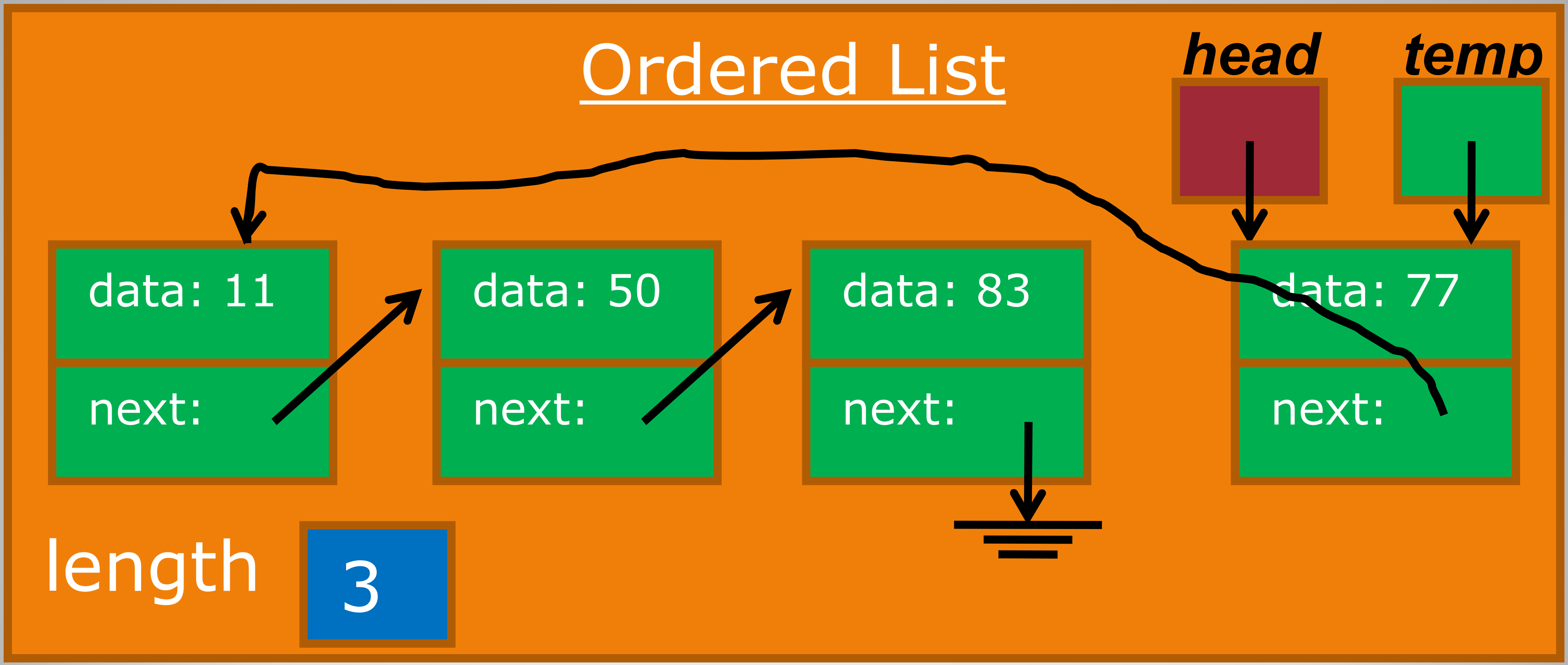
2. Set the fields on the new Node. Set data item and next pointer. Next points to current list start.



Ordered List - insertItem

ol.insertItem(77)

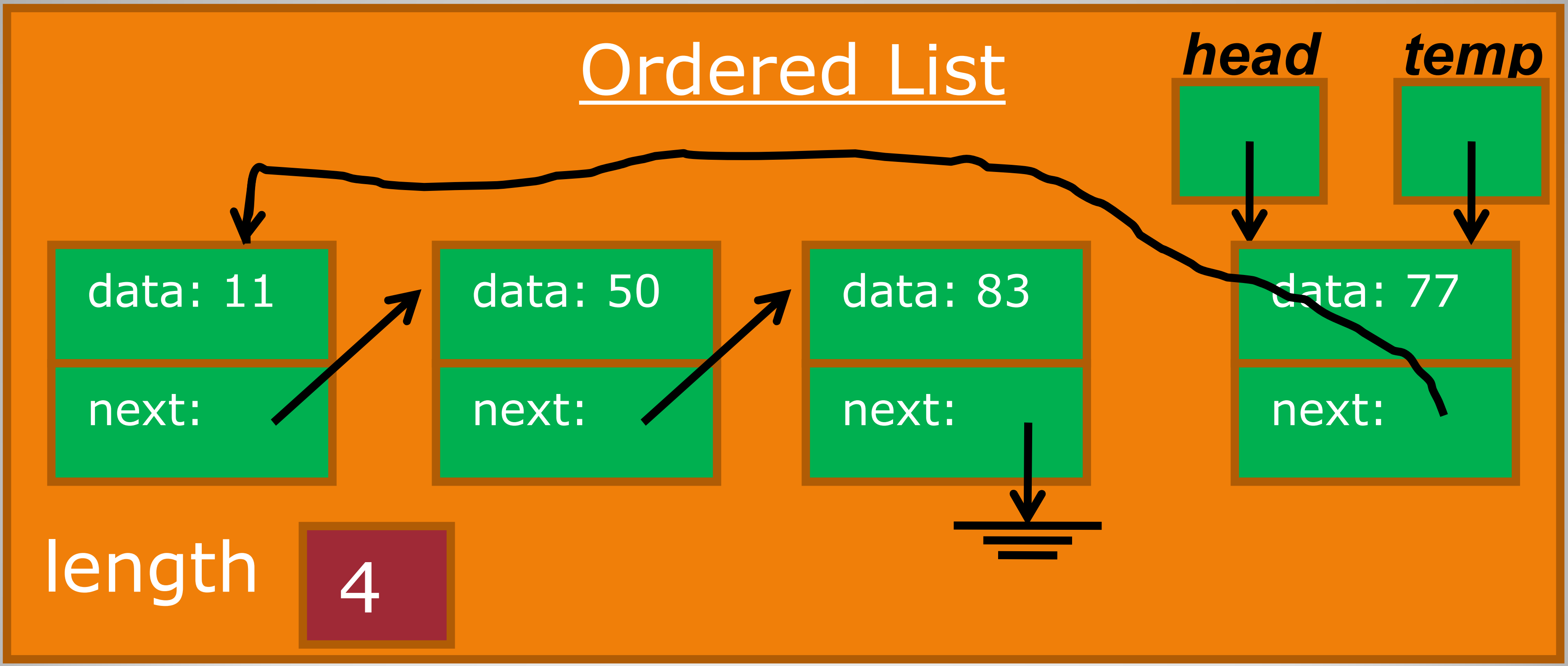
3. Set the list start pointer (head) to the new Node.



Ordered List - insertItem

ol.insertItem(77)

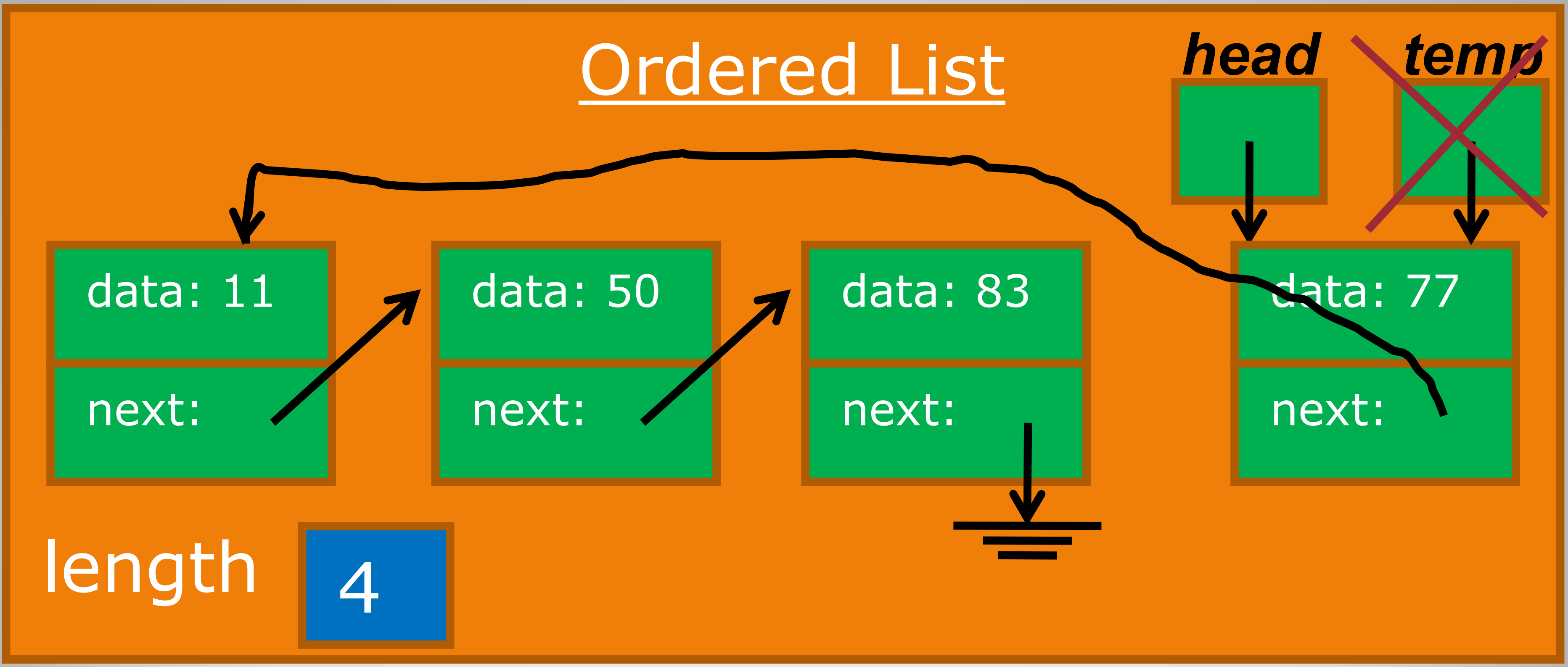
4. Increment the length of the list.



Ordered List - insertItem

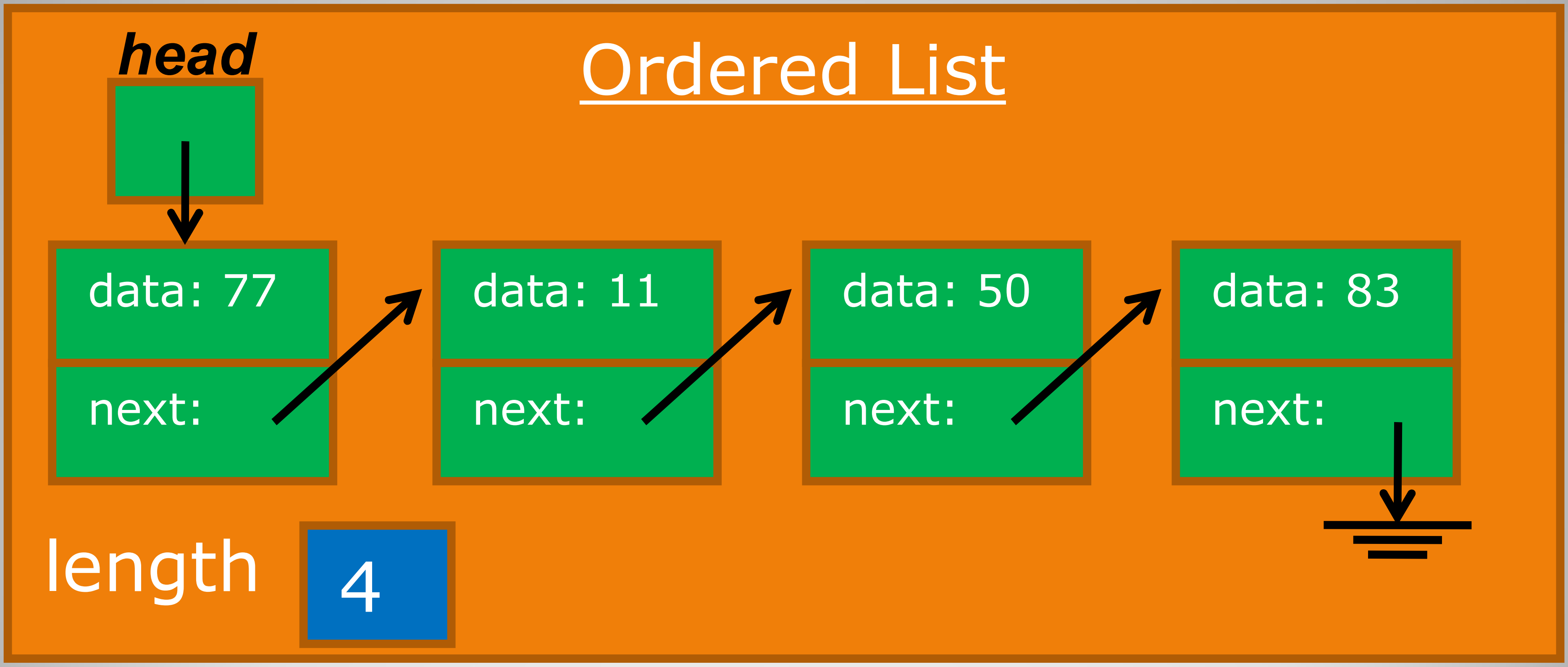
ol.insertItem(77)

When the insertItem method ends the temp pointer will go out of scope and disappear.



Ordered List - insertItem

This picture is ***LOGICALLY EQUIVALENT*** to the previous slide!!!



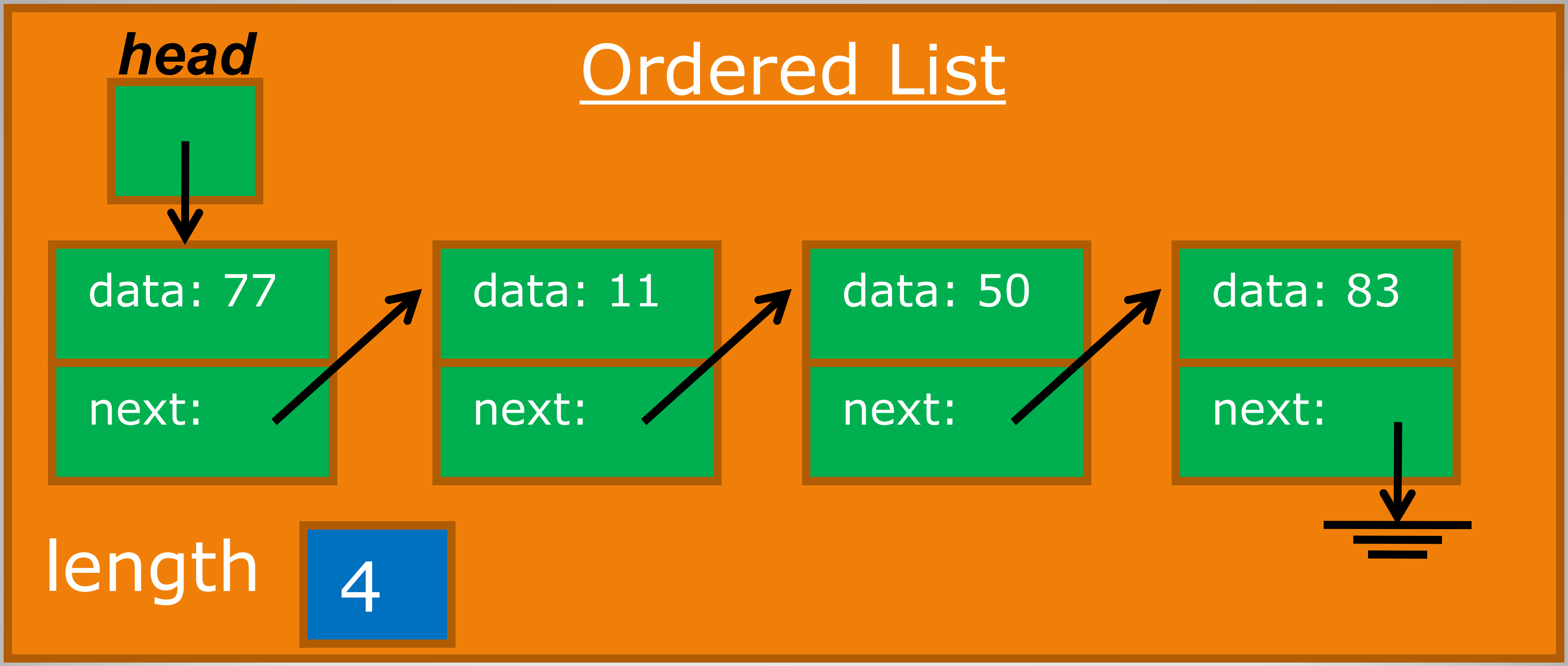
Ordered List - insertItem

Ordered List - insertItem

```
insertItem(int item)
    Declare Node temp

    Set temp to new node instance

    Set temp.data to item
    Set temp.next to head

    Set head to temp
    Increment length
```

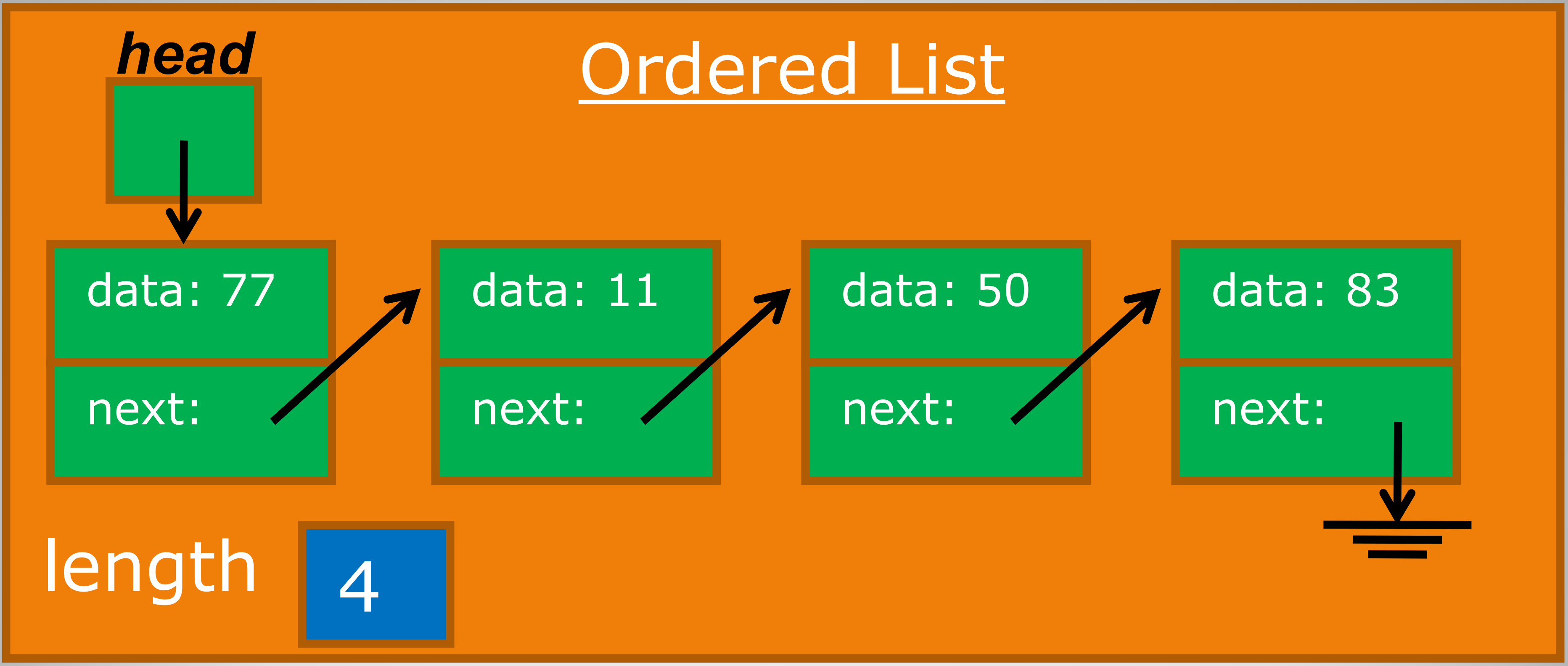
Ordered List - hasItem

How do you check if an item is in the list?



Need to follow the pointer to get to the target data item.

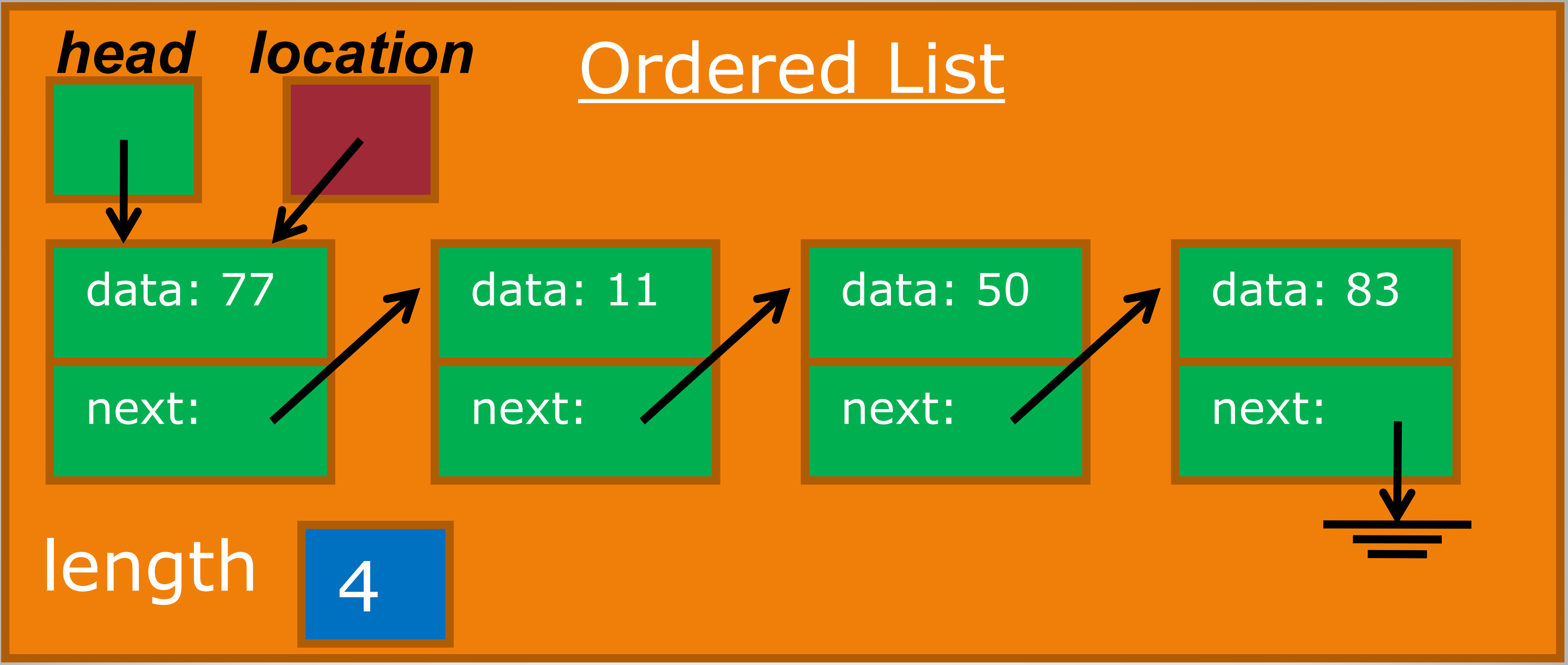
```
boolean result
result = ol.hasItem(50)
```



Ordered List - hasItem

Set location (just a temp variable) to the start of the list and then keep following it until you reach the target or the end of the list.

location = head



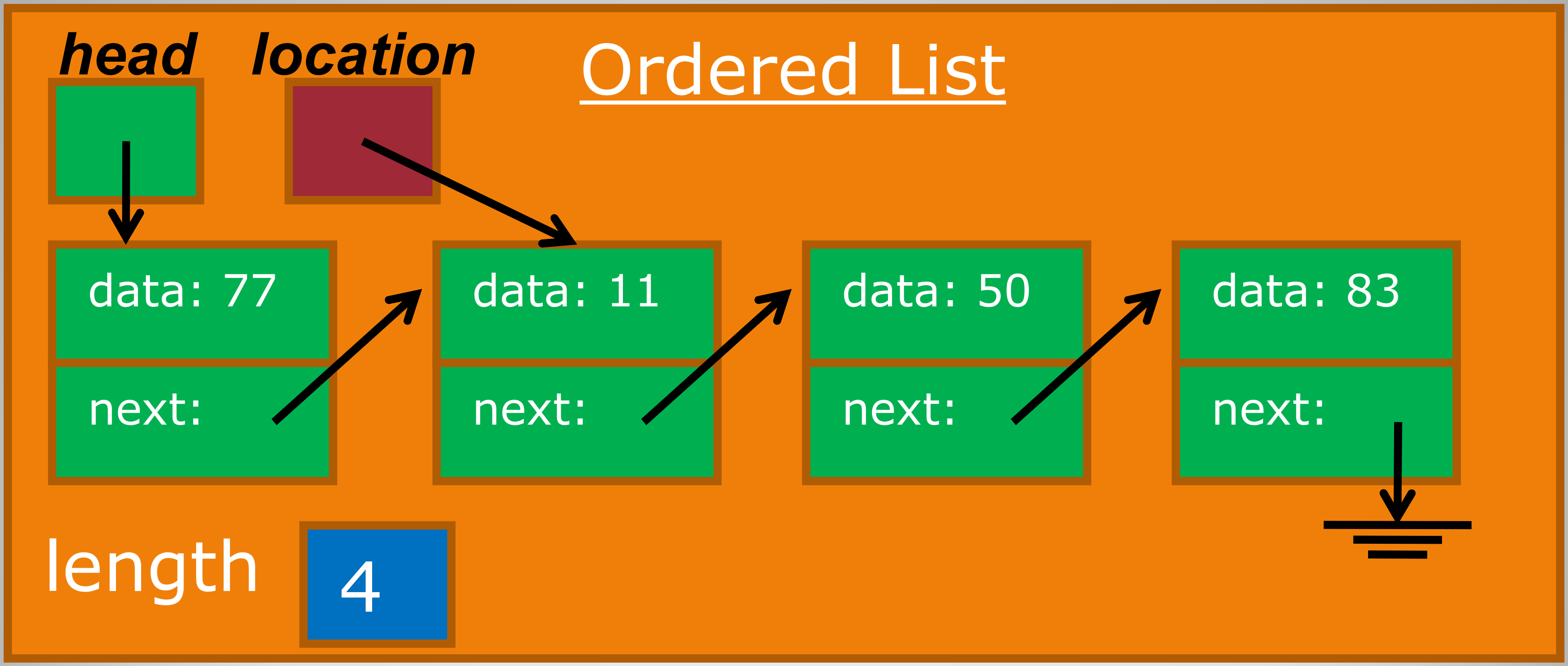
Ordered List

How do you make location point to the next item?

Ordered List - hasItem

```
location = location.next
```

location was pointing at item 77.
Item 77 is pointing at item 11.
location now points at item 11.



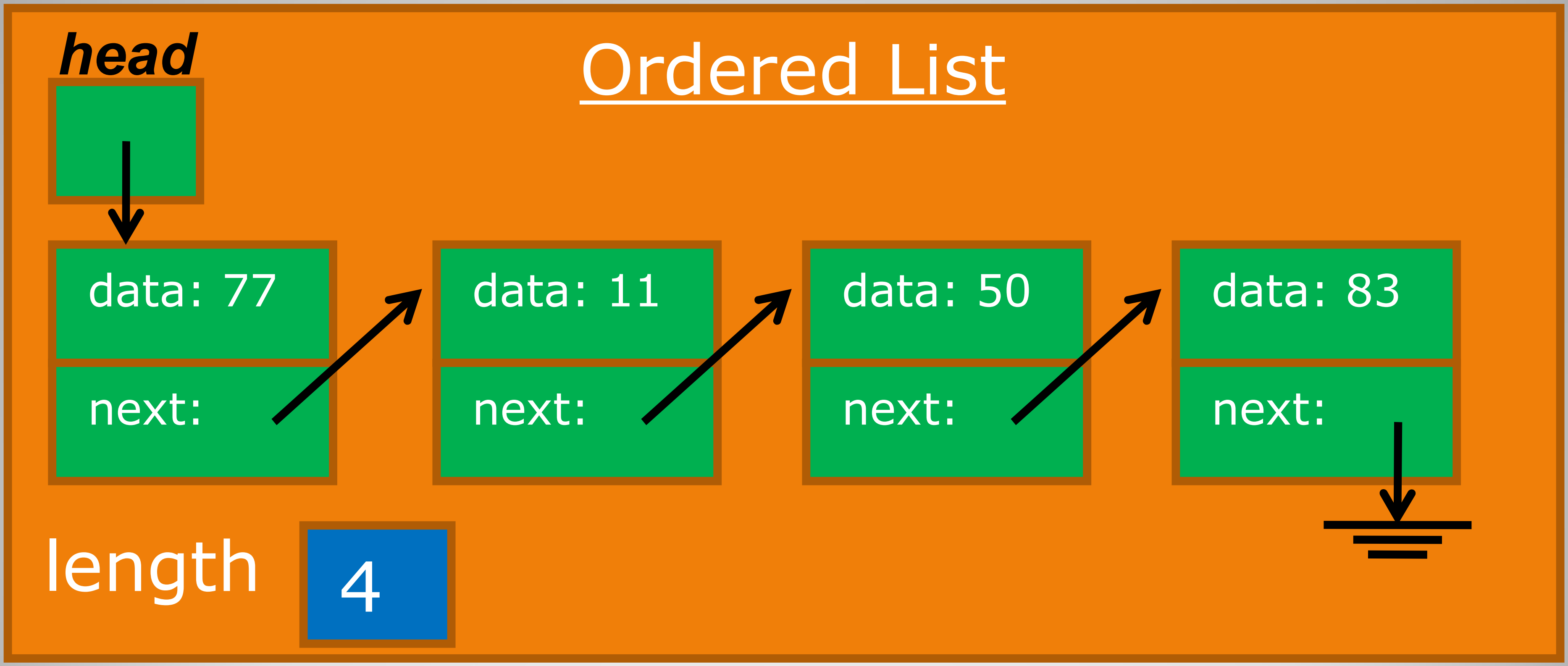
```
boolean hasItem(int target)
    Initialize location to head

    while location is not null
        if location data equals target
            return true

        location = location.next
    endWhile

    return false
```

Ordered List - hasItem

Now we will move on to deleteItem…

Ordered List - deleteItem

How do you delete an item from the list?



Ordered List - deleteItem

deleteItem Pseudocode (High level)

1. Find the target item to delete.
2. Update the pointers in the list so that the target item is removed.
3. Set the target to null so memory for that node can eventually be given back to the system.
4. Decrement the length.

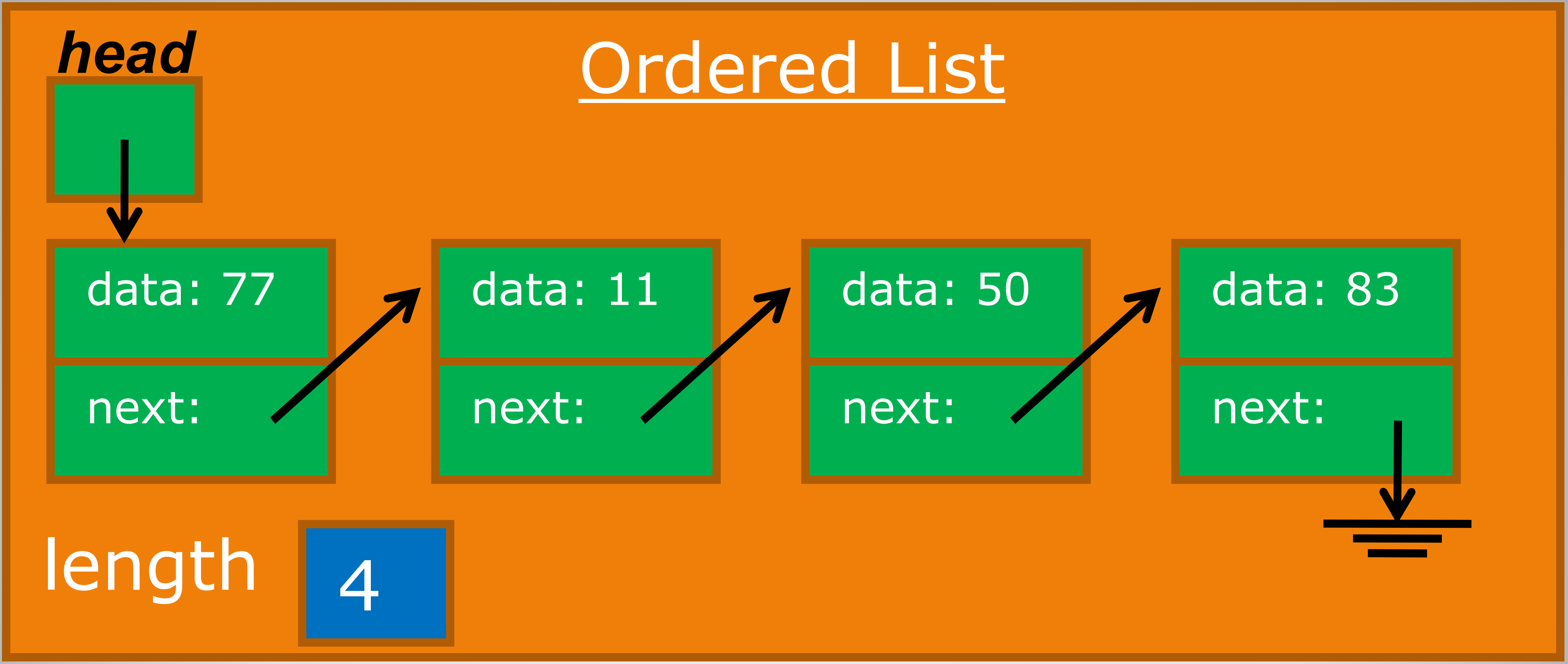
Ordered List - deleteItem

Ordered List - deleteItem

deleteItem Pseudocode (Detailed)

1. Find the target item to delete. Can be one of two cases:
   a) The start item is the target item.
   b) The target is somewhere else in the list.
2. Update the pointers in the list so that the target item is removed.
3. Set the target to null so memory for that node can eventually be given back to the system.
4. Decrement the length.

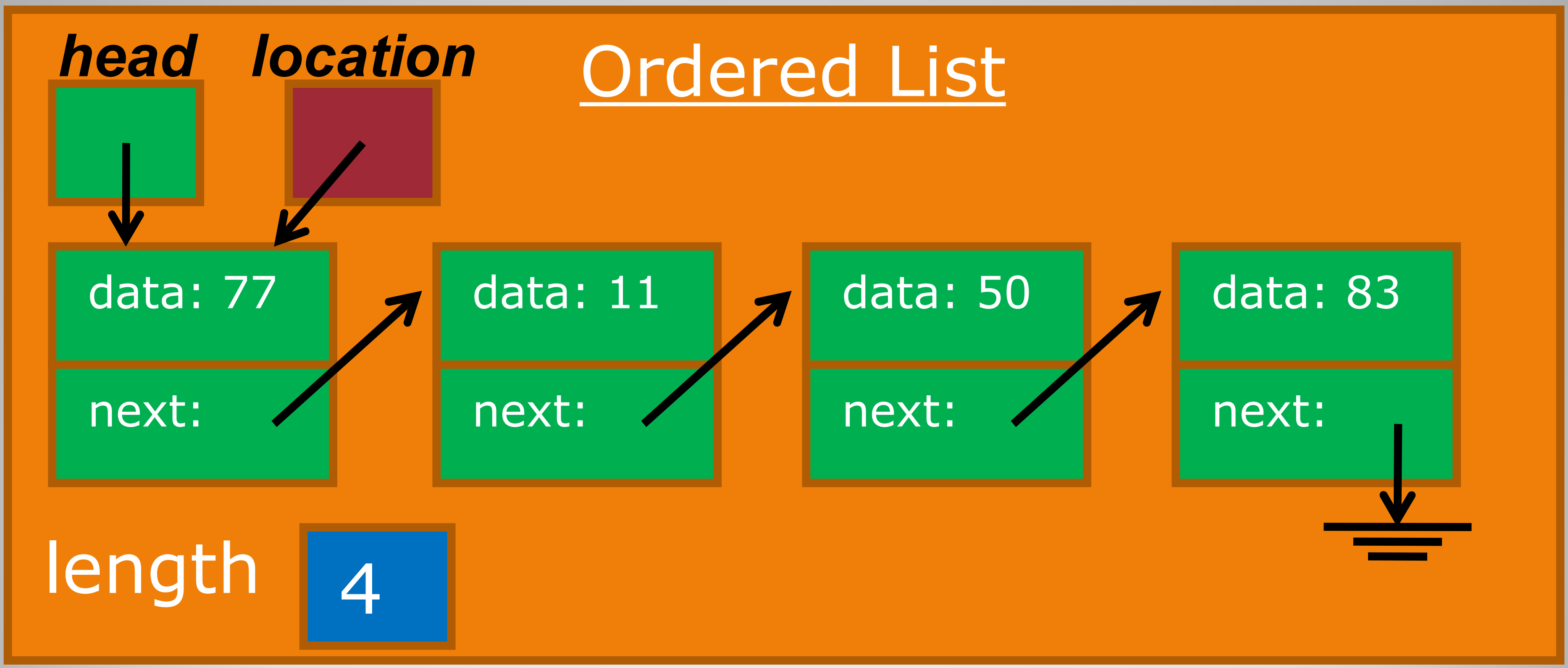
ol.deleteItem(77)



Ordered List - deleteItem

```
Declare location as node and Set to head
if item equals location.data
      head = head.next
      location = null
      decrement length
```

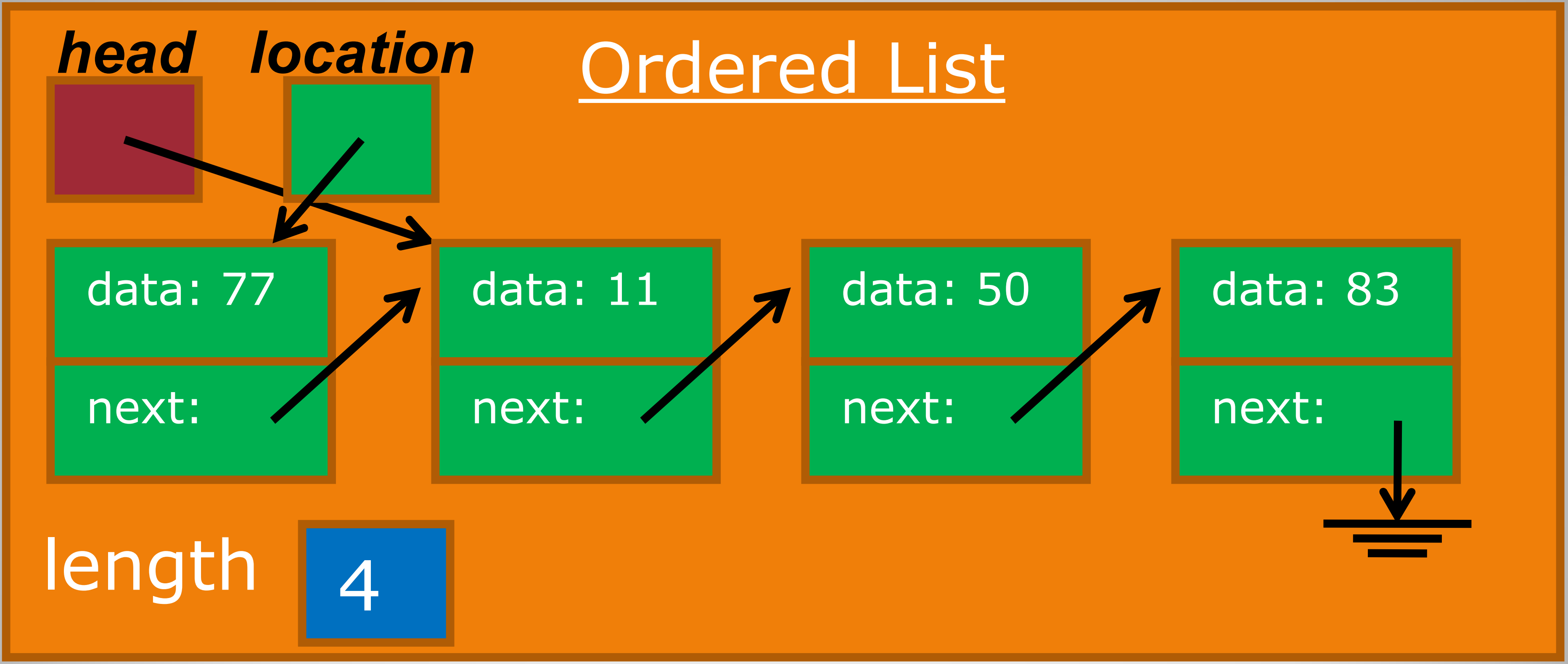
Delete Code For First Node



Ordered List - deleteItem

```
Declare location as node and Set to head
if item equals location.data
        **head = head.next**
        location = null
        decrement length
```

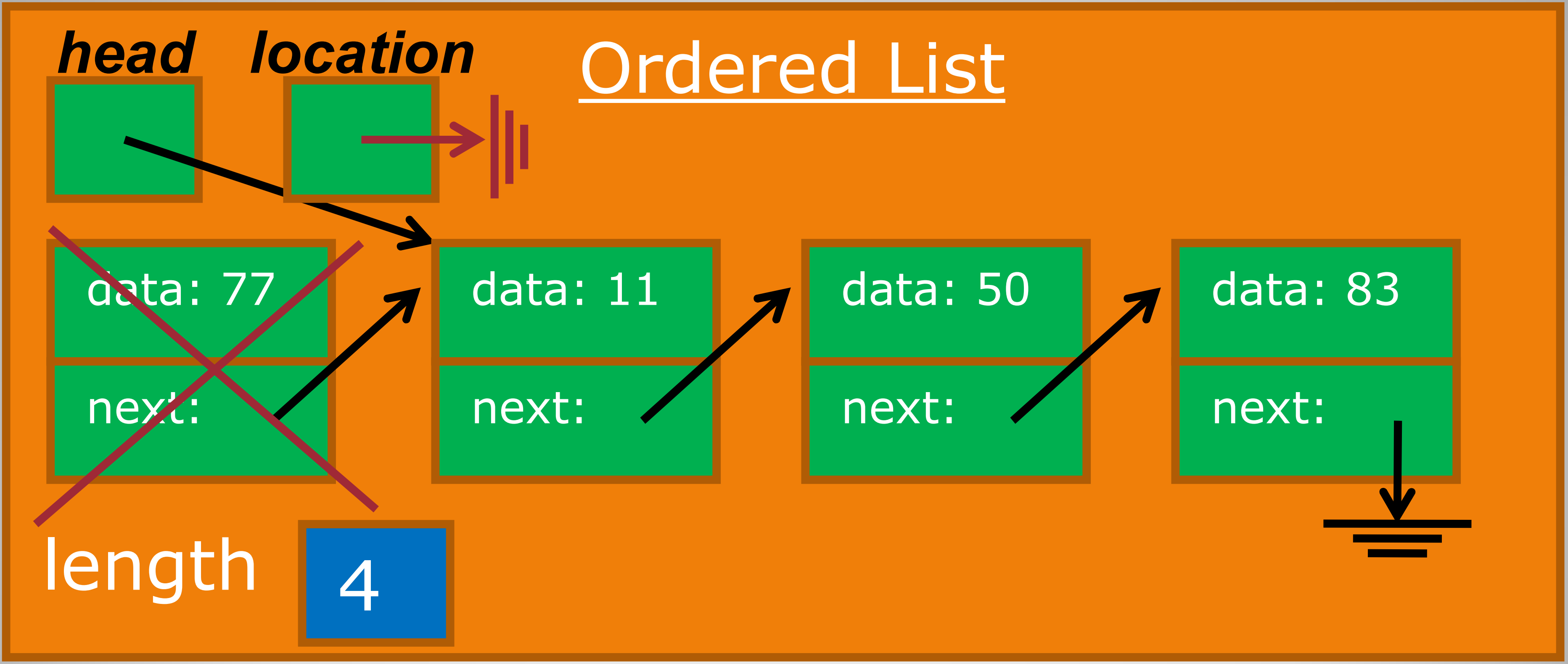
Delete Code
For First Node



Ordered List - deleteItem

```
Declare location as node and Set to head
if item equals location.data
        head = head.next
        **location = null**
        decrement length
```

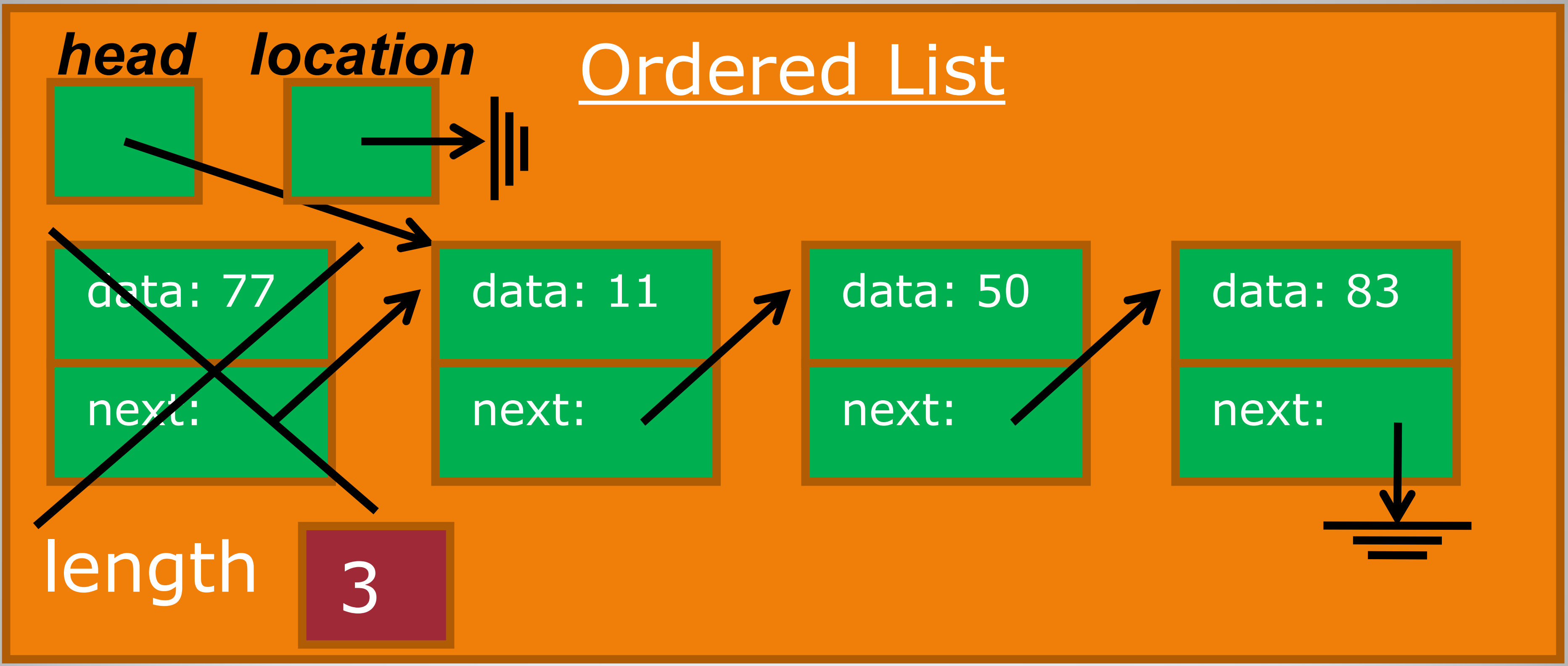
Delete Code For First Node



Ordered List - deleteItem

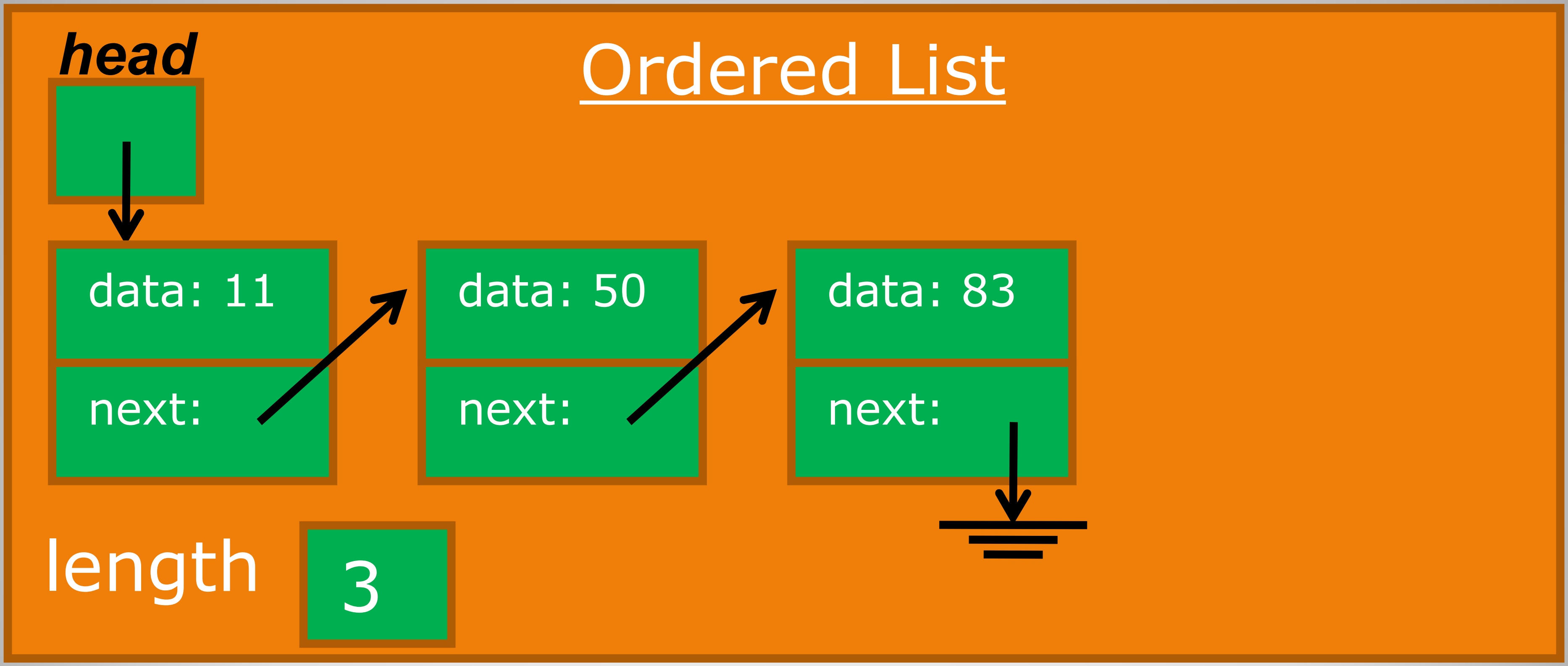
```
Declare location as node and Set to head
if item equals location.data
      head = head.next
      location = null
      decrement length
```

Delete Code For First Node



Ordered List - deleteItem

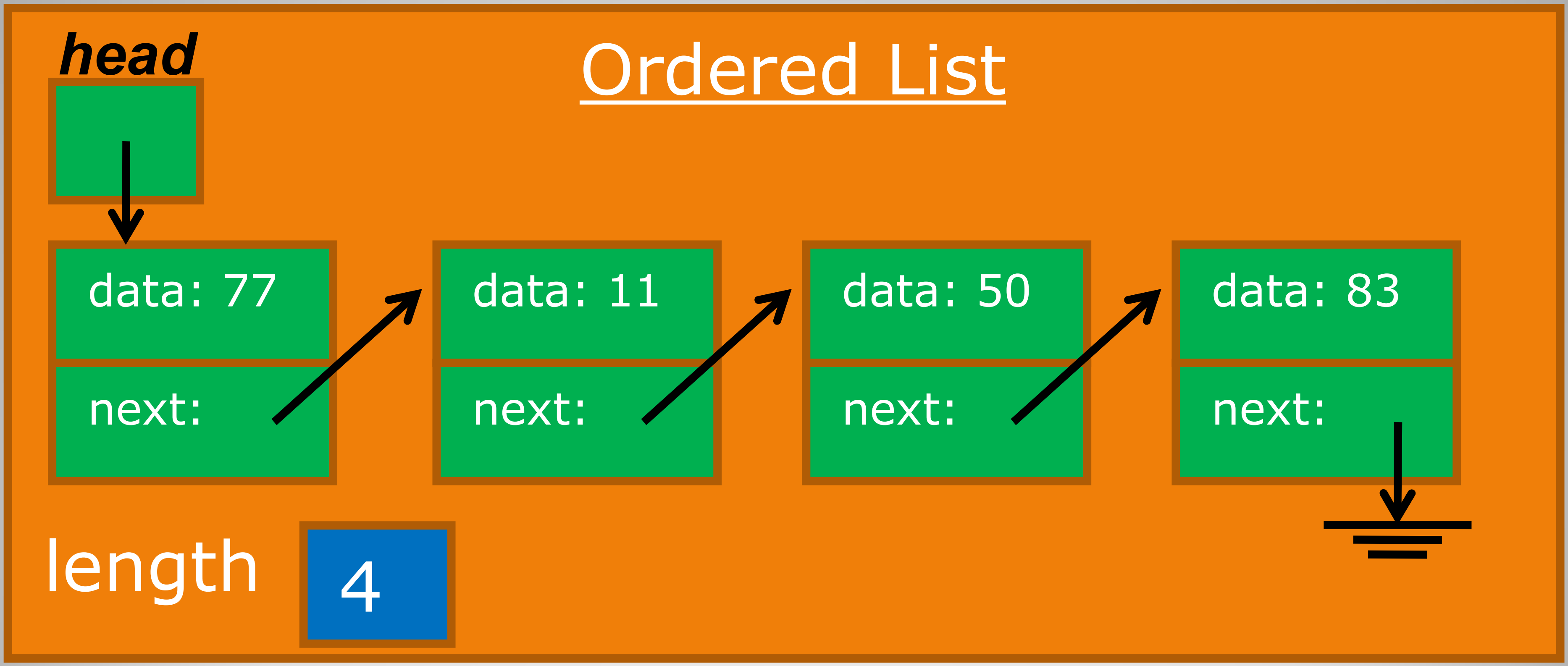
- This is what the list looks like after ol.deleteItem(77) is complete.



Ordered List - deleteItem

Now delete 50 from the original list

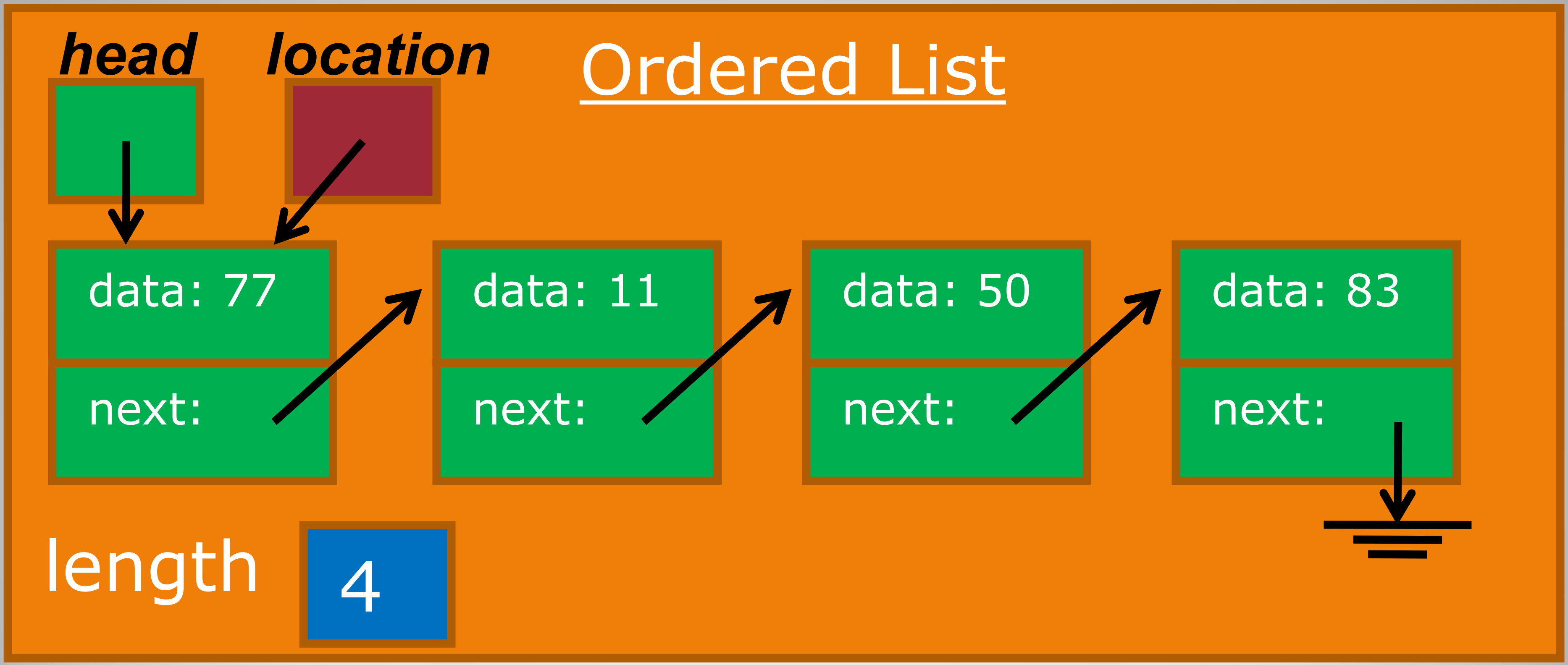
ol.deleteItem(50)



Ordered List - deleteItem

```
location = head
while ( (location.next != null) and (item != (location.next).data))
      location = location.next
endWhile
```

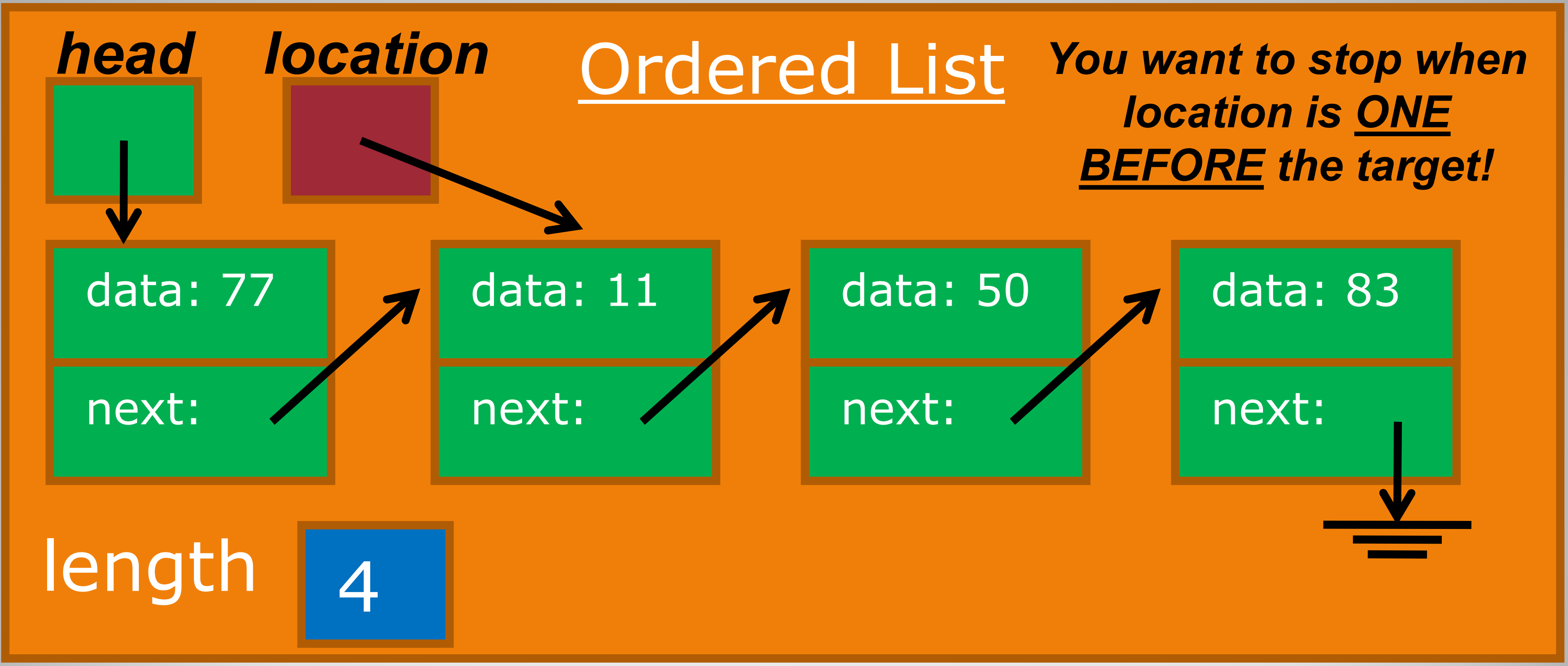
Set location to start of list



Ordered List - deleteItem

```
location = head
while ( (location.next != null) and (item != (location.next).data))
      location = location.next
endWhile
```

Keep following location pointer while it is not equal to item

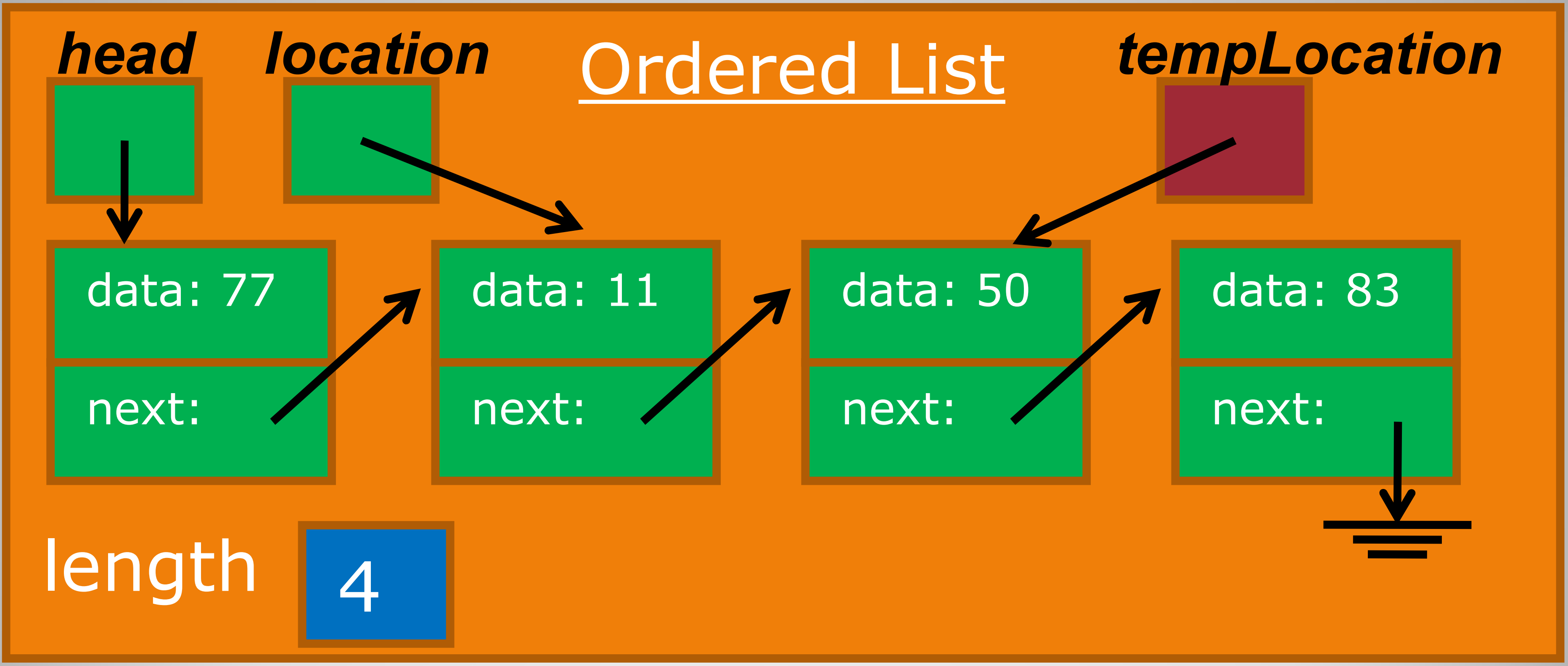


You want to stop when location is <u>ONE</u> <u>BEFORE</u> the target!

Ordered List - deleteItem

```
if location.next equals null then return  // target not in list
**Node tempLocation = location.next**
location.next = (location.next).next
tempLocation = null
Decrement length
```

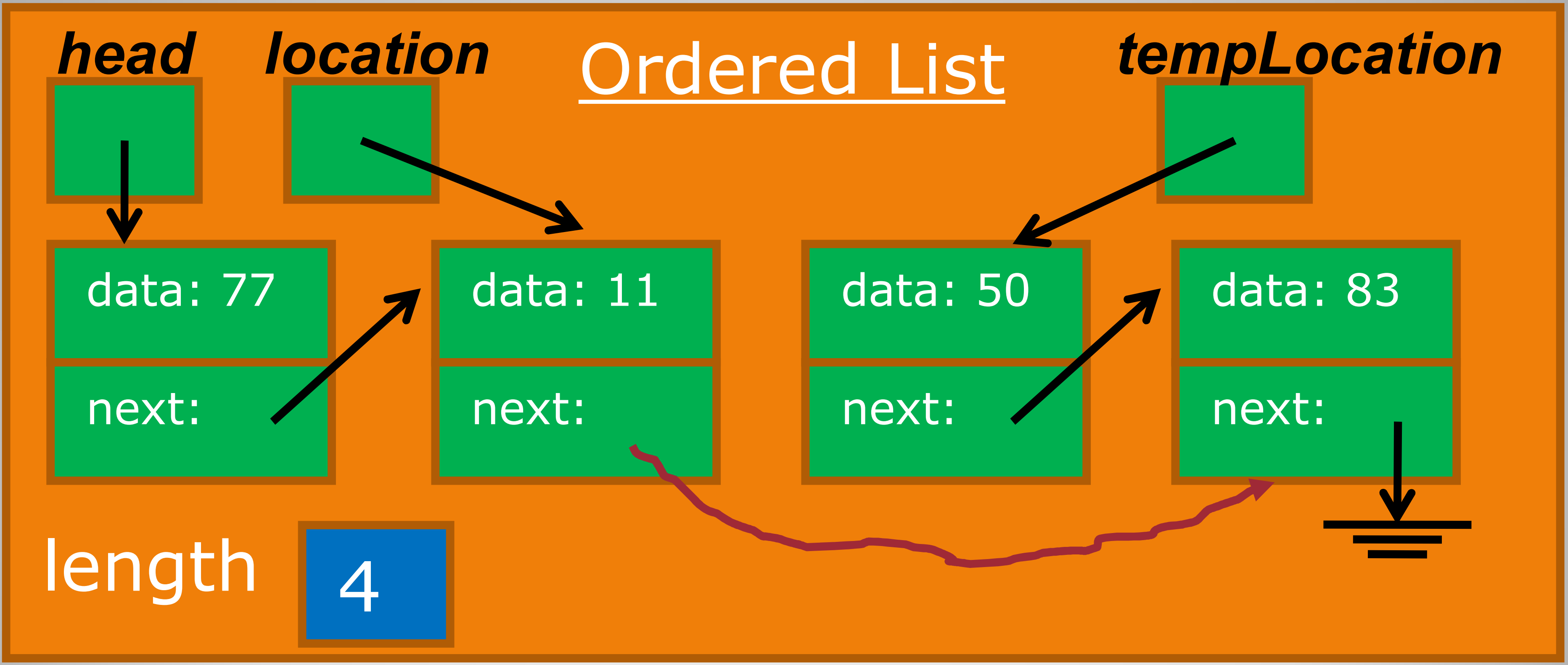
Code to Actually Delete The Node



Ordered List - deleteItem

```
if location.next equals null then return  // target not in list
Node tempLocation = location.next
location.next = (location.next).next
tempLocation = null
Decrement length
```

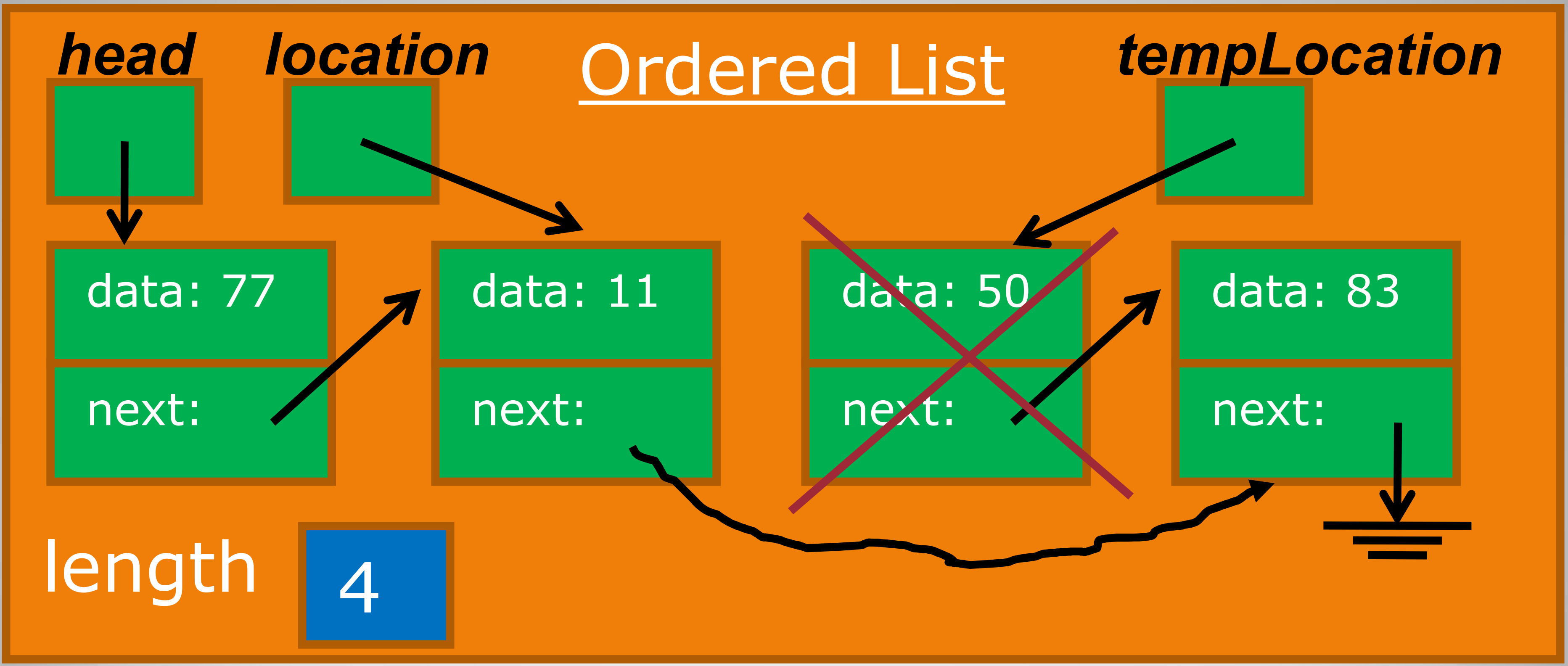
Code to Actually Delete The Node



Ordered List - deleteItem

```
if location.next equals null then return  // target not in list
Node tempLocation = location.next
location.next = (location.next).next
tempLocation = null
Decrement length
```

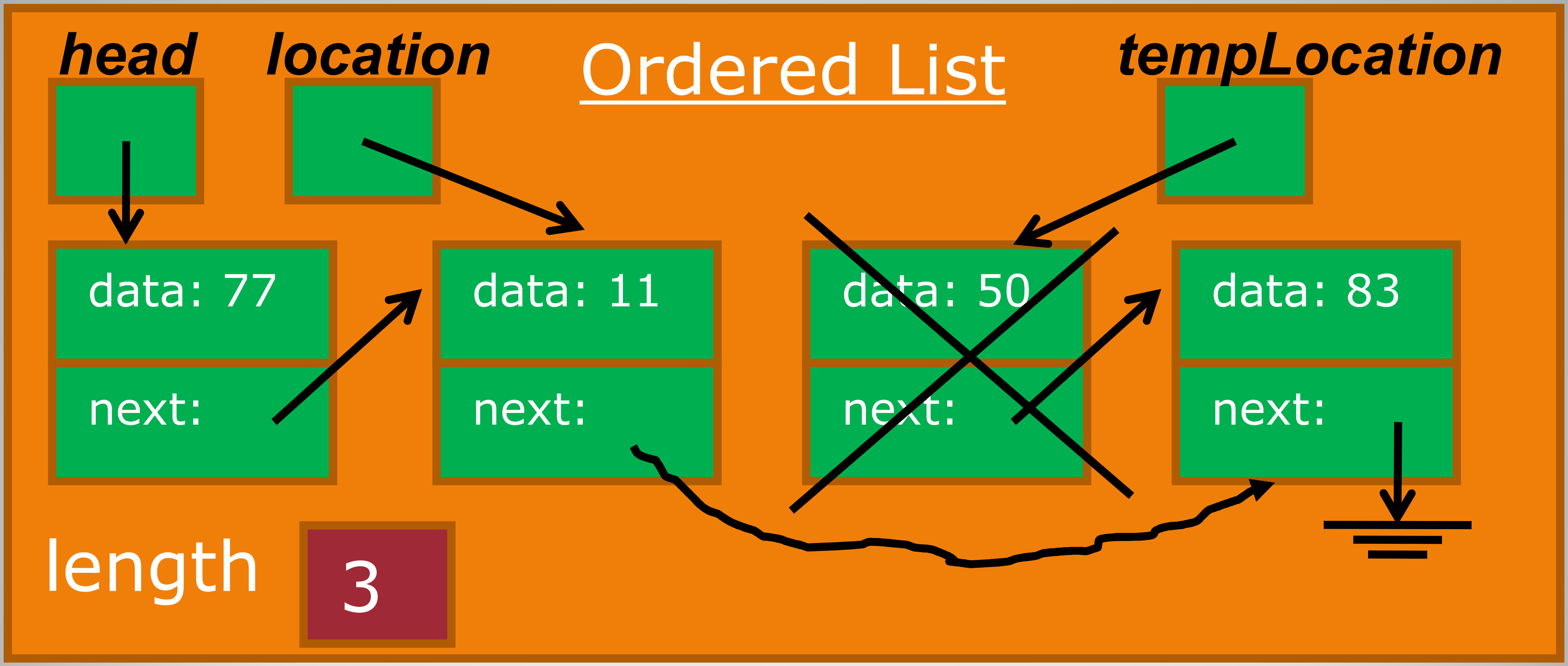
Code to Actually Delete The Node



Ordered List - deleteItem

```
if location.next equals null then return  // target not in list
Node tempLocation = location.next
location.next = (location.next).next
tempLocation = null
```
Decrement length

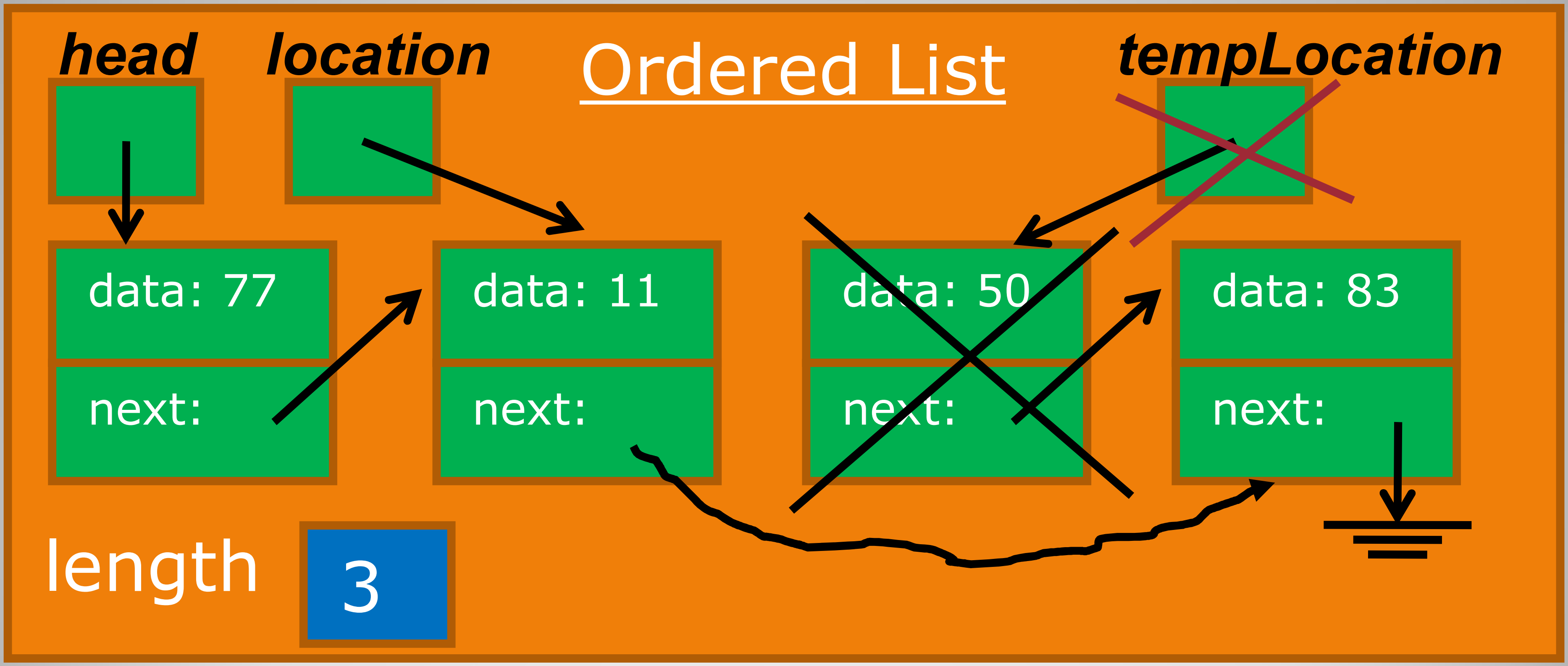
Code to Actually Delete The Node



Ordered List - deleteItem

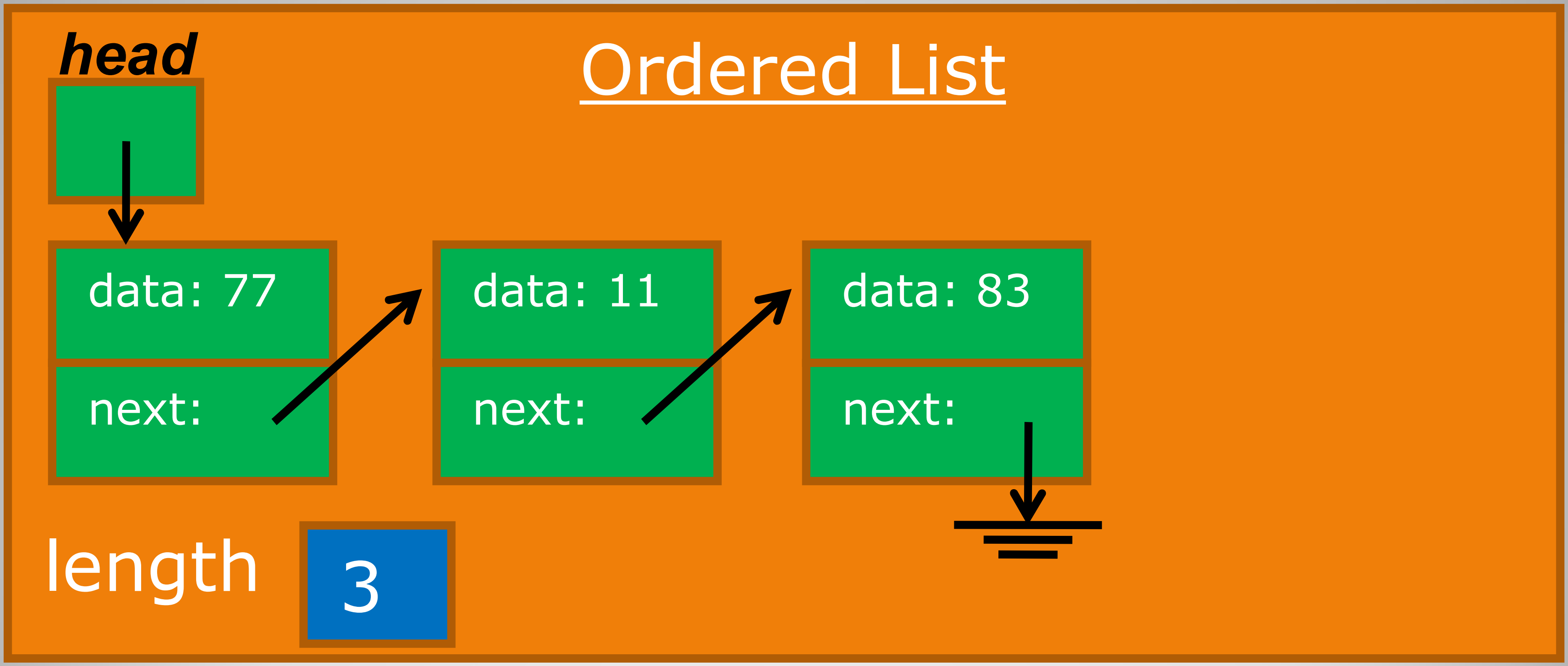
```
if location.next equals null then return  // target not in list
Node tempLocation = location.next
location.next = (location.next).next
tempLocation = null
Decrement length
```

tempLocation will disappear when DeleteItem ends.



Ordered List - deleteItem

Ordered list AFTER the following call:
ol.deleteItem(50)



Ordered List - deleteItem

Ordered List – makeEmpty

```
makeEmpty()
    Set head to null
    Set length to 0
```

Make head null. All nodes in the queue are now unreferenced so they will become candidates for garbage collection

Below is a slower version. It explicitly sets all nodes to null. This is unnecessary since the garbage collection will find those nodes for us.

```
makeEmpty()
    Declare Node temp
    while  head not equal to null
           Set temp to head
           Set head to head.next
           Set temp to null
    endWhile

    Set length to 0
```

Keep going until there is no first element.

Keep deleting the first element

Set the list length to 0

Ordered List – isFull

```
boolean isFull()
   Node location
   try
      location = Create new node from heap
      Set location to null
      return false
   catch OutOfMemoryError exception
      return true
```

Check to see if you can allocate memory.

If you CAN, then the list is NOT full so return false.

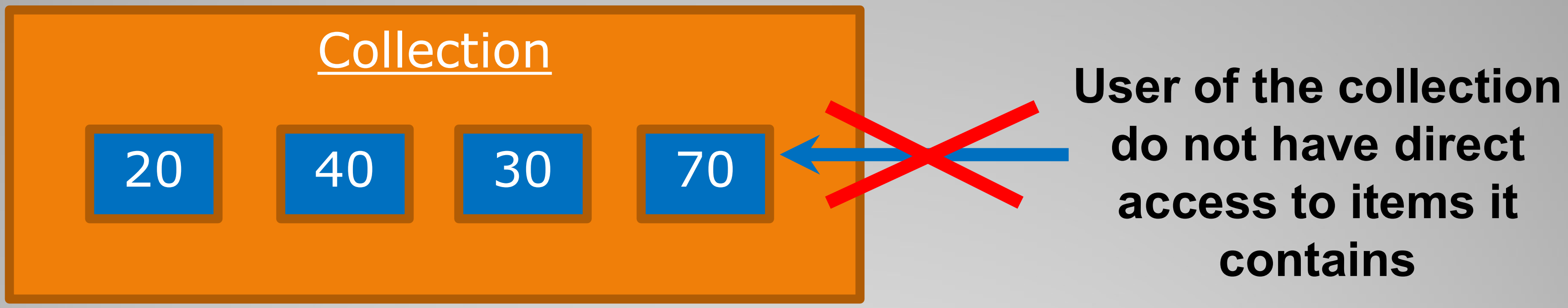
If you CANNOT allocate memory, then the list is full.

Now we will move on to iterators…

Iterators

Iterators

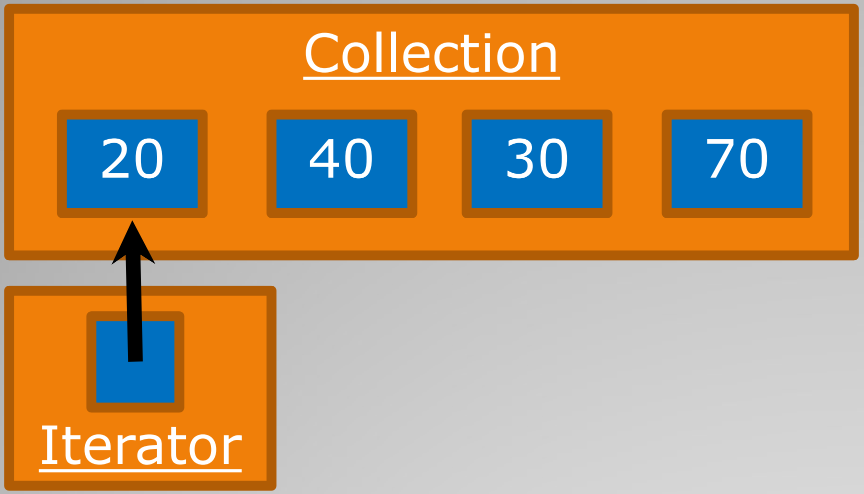
- Here is a collection with data:


Collection
20
40
30
70
User of the collection do not have direct access to items it contains

- The user of the collection does not have direct access to the items of the collection.
- There needs to be a way to "visit" each item of the collection while not giving direct access to it.
- That is what an iterator is for.

Iterators

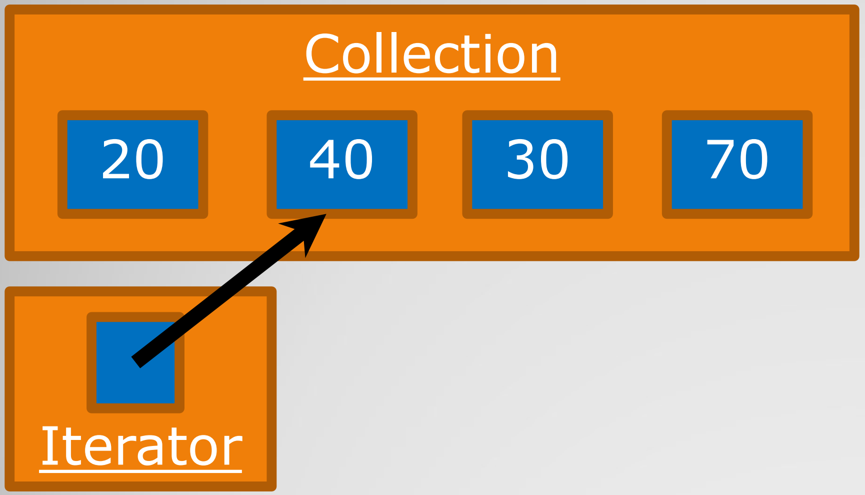
- Iterators have access to the items of the collection.
- An iterator points at one item of the class.
- In general, you can do the following with an iterator:
  - Get the data at that item.
  - Check if the iterator is pointing at valid data.
  - Go to the next item in the collection.
  - Remove the item from that collection.



This iterator points at the first item of the collection.

You can get the data (20) at that item if you want but not the other items.

If we told the iterator to go to the next item then it would look like the following….



Iterator now points at the second item.

You can get the data in the second item (40) but not the other items.

Iterators

- We can build an iterator into our singly-linked ordered list class.
- We could either use a whole other class for the iterator or build it into the existing class.
- We will build it into the existing class for our implementation.

Iterators and Ordered List

IteratorForward Interface

- We will use our own iterator interface.
- The iterator will be built into the class (OrderedList can implement this interface).

```
public interface IteratorForward {
    int iterGetData();
    void iterMoveNext();
    void iterMoveStart();
    boolean iterIsValid();
}
```

Iterator Implementation

```
iterGetData() returns int
    if (iter is not null) the return iter.data

    return Integer.MAX_VALUE


iterMoveNext()
    if (iter is not null) Set iter to iter.next


iterMoveStart()
   Set iter to head


iterIsValid() returns boolean
   if (iter equals null) then return false

   return true
```

```
// Code to declare and populate list goes here…

Move iter to start
while (iter is valid)
    Print iter.data
    Move iter to next
endWhile
```

Put the iterator at the start of the list

Keep going while iterator is valid

Print the data retreived using the iterator

Go to next item

Iterator – Using the iterator

Now we will finish with Big-O…

Big-O Comparison

Big-O Comparison

- It is important to know the approximate runtime cost operations when you create a data structure.
- What are the Big-O runtimes for the list implementations?

Operation	Cost
makeEmpty	???
isFull	???
getLength	???
hasItem	???
retrieveItem	???
insertItem	???
deleteItem	???

Big-O Comparison – Ordered List (Linked-list)

Operation	Cost
makeEmpty	O(1)
isFull	O(1)
getLength	O(1)
hasItem	O(n)
retrieveItem	O(n)
insertItem	O(1)
deleteItem	O(n)

Big-O Comparison – Ordered List (Linked-list)

- **End of Slides**

End of Slides