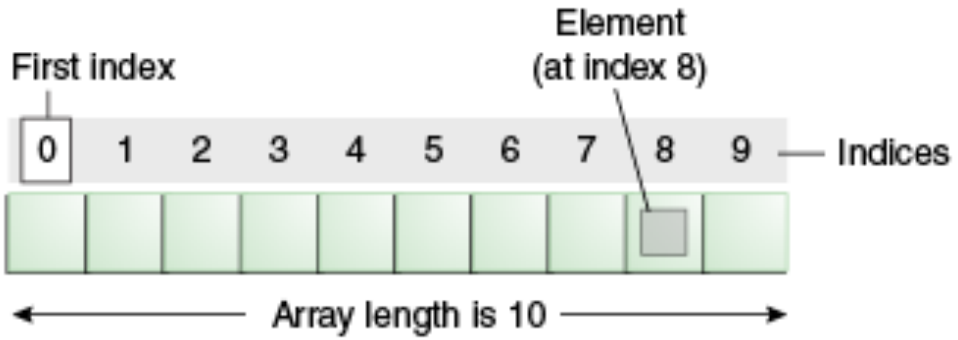
LINKED LISTS

IMPLEMENTATION OF LINKED LIST DATA STRUCTURE

REVIEW OF ARRAYS

```
int[] x= new int[10];

X[8]=100;
```



- **Array elements are stored in contiguous memory addresses.**
- **Accessing a random array element in O(1)**
  - The [ ] operator is used to locate an element.
  - The address is calculated by adding a fixed offset to the base address.

DISADVANTAGES OF ARRAYS

- **Array size is fixed.**
  - Even arrayList objects have a fixed capacity at any point. To add more elements, the whole array has to be reallocated in memory.
- **Adding extra elements to an arrayList using the .add() method is O(n)**
- **Inserting elements at a random index in the array requires shifting all the elements to the right: O(n)**
  - Example: a.add(3, “newElement”);

LINKED LISTS

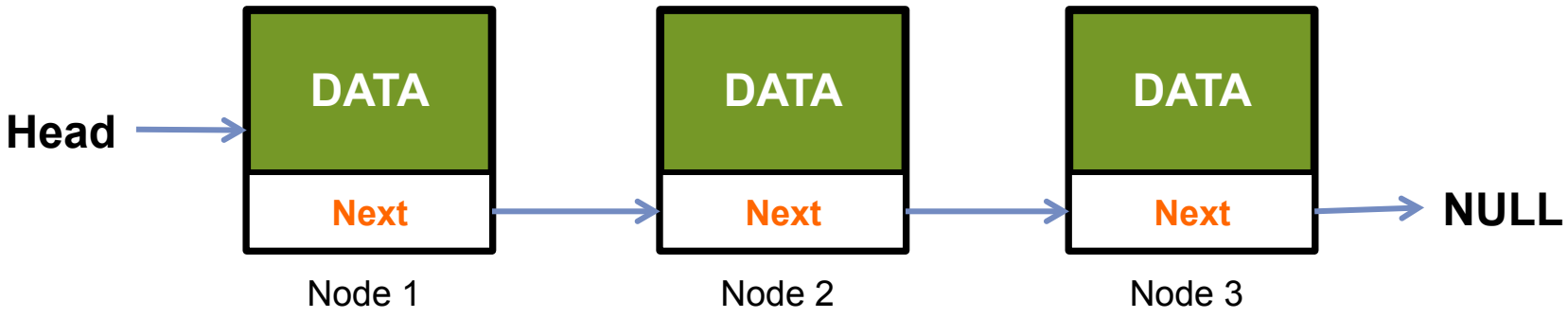
- **A list of elements of any type, similar in functionality to arrays and arrayLists.**
- **Advantages:**
  - Linked lists don’t have a fixed size or capacity.
  - Can insert elements any where in the list in O(1)
- **Disadvantage:**
  - random access is not efficient: O(n)

LINKED LIST STRUCTURE

- **Each element is allocated in its own memory space when created, independently of other elements.**
- **Elements are linked together using references**
- **Each elements consists of two fields: a data field, and a reference to the next node.**

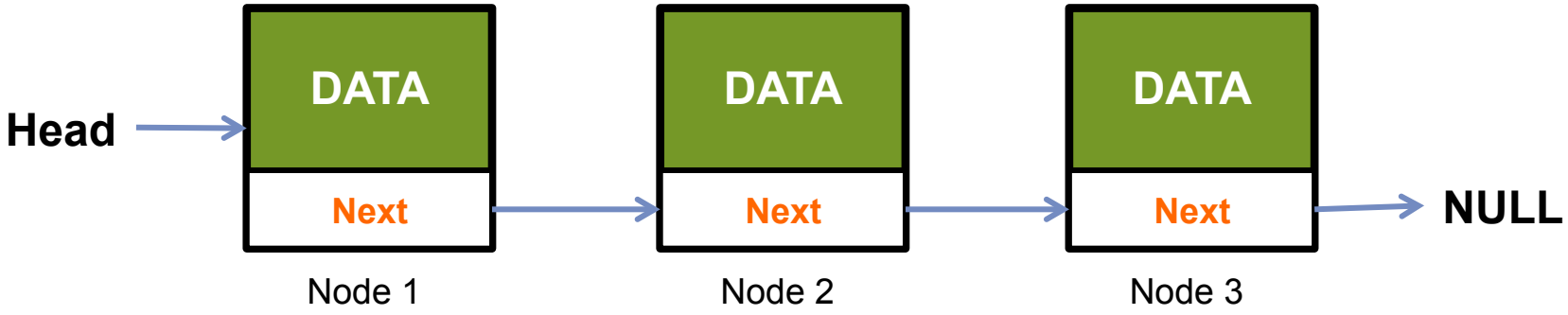
List Node

LINKED LIST STRUCTURE



- **The beginning of the list is stored in a 'head' reference, which points to the first element in the list.**
- **The first element contains a reference to the second element. The second elements contains a reference to the third element .. and so on.**
- **The last node's next field points to NULL.**

LINKED LIST STRUCTURE

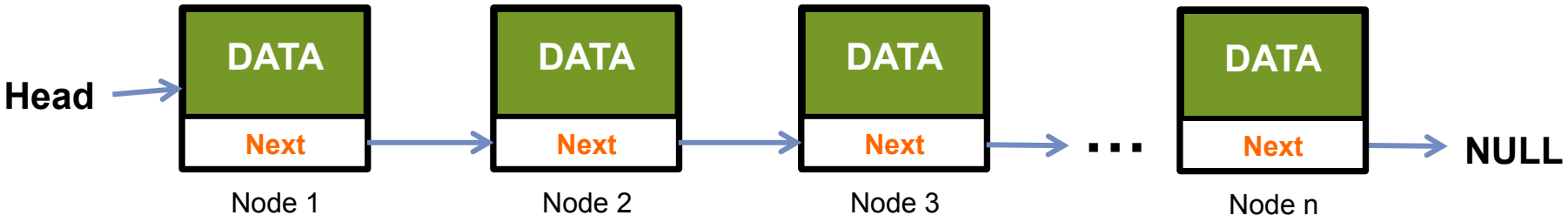


a) Accessing the first element is straightforward using the head reference in O(1)

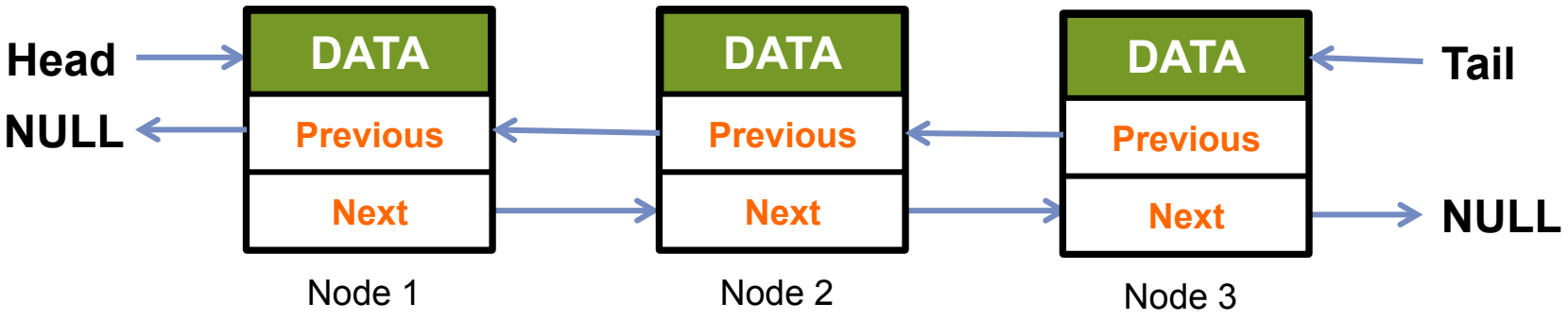
b) To access the second element, we first access the first element as in (a), then use it's "Next" reference to access the second element.

LINKED LIST STRUCTURE

- **Accessing elements farther from the first element is less efficient.**



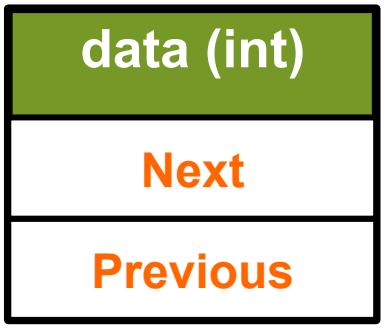
DOUBLY LINKED LIST



- For more flexibility, we can add a reference to the previous node in addition to the next node.
- We can also add a reference to the last element “tail”
- We can access elements from both ends of the list.

THE NODE CLASS

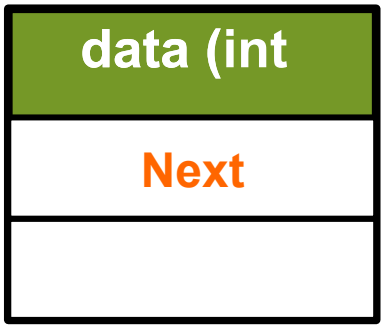
```
public class IntNode {
    private int data;
    private IntNode next;
    private IntNode previous;

    //initialize node
    public IntNode(int value){
        data=value;
        next=null;
        previous=null;
    }
```



IntNode Object

THE NODE CLASS

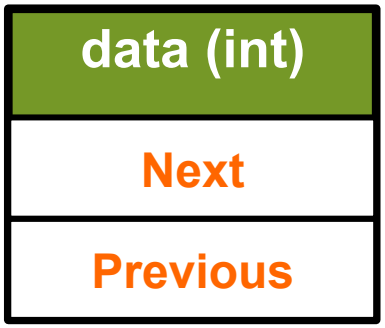
```
//set link to next node
public void setData(int value){
    data=value;
}
//set link to next node
public void setNext(IntNode node){
    next=node;
}
//set link to previous node
public void setPrevious(IntNode node){
    previous=node;
}
```



IntNode Object

THE NODE CLASS

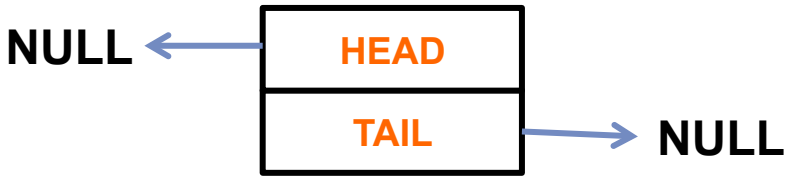
```
//return the data value
public int getData(){
    return data;
}
//return the next node
public IntNode getNext(){
    return next;
}
//return the previous node
public IntNode getPrevious(){
    return previous;
}
```



IntNode Object

THE LINKED LIST CLASS

```
public class LinkedList {
    //private variables:
    private IntNode head;
    private IntNode tail;

    //constructor
    public LinkedList(){
        //empty linked list
        head=null;
        tail=null;
    }
```

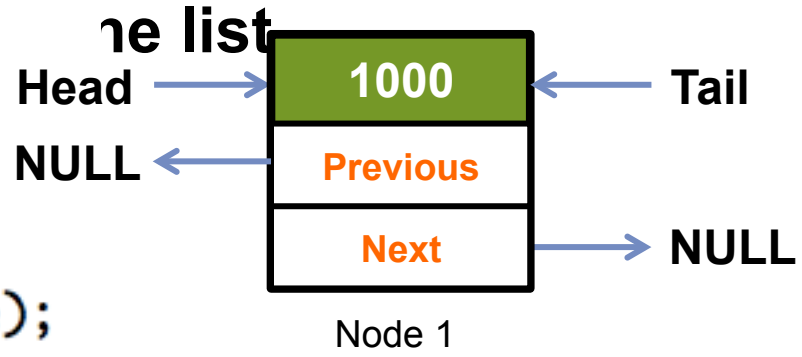


LinkedList x;
x=new LinkedList();

ADDING ELEMENTS

- **To add the first element to the list**

```
//add a node to the list
public void add(int value){
    //if list is empty
    if (head==null){
        head=new IntNode(value);
        tail=head;
```

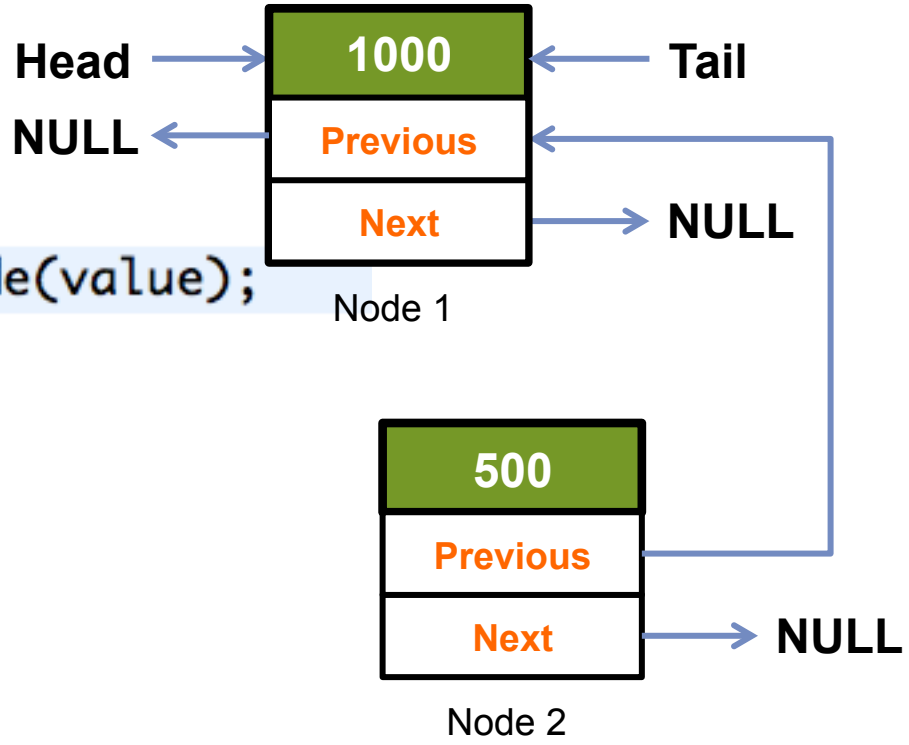



```
int element= 1000;
x.add(element=1000;
```

ADDING ELEMENTS

- To add another element to the list

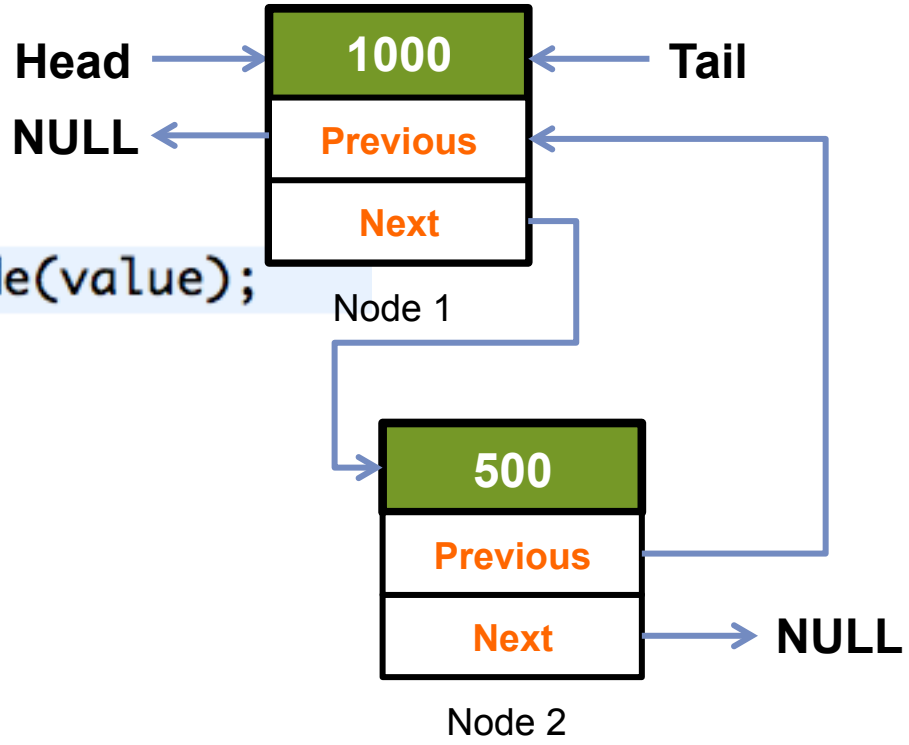
```
}else{
    //create new Node
    IntNode newNode=new IntNode(value);
    newNode.setPrevious(tail);
```



ADDING ELEMENTS

- To add another element to the list

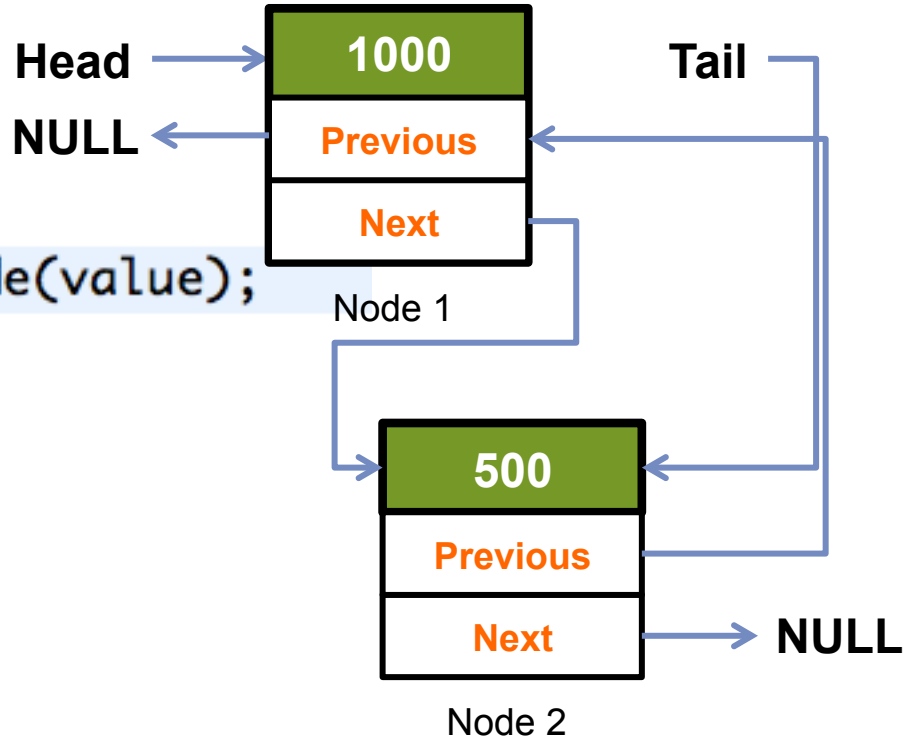
```
}else{
    //create new Node
    IntNode newNode=new IntNode(value);
    newNode.setPrevious(tail);
    tail.setNext(newNode);
```



ADDING ELEMENTS

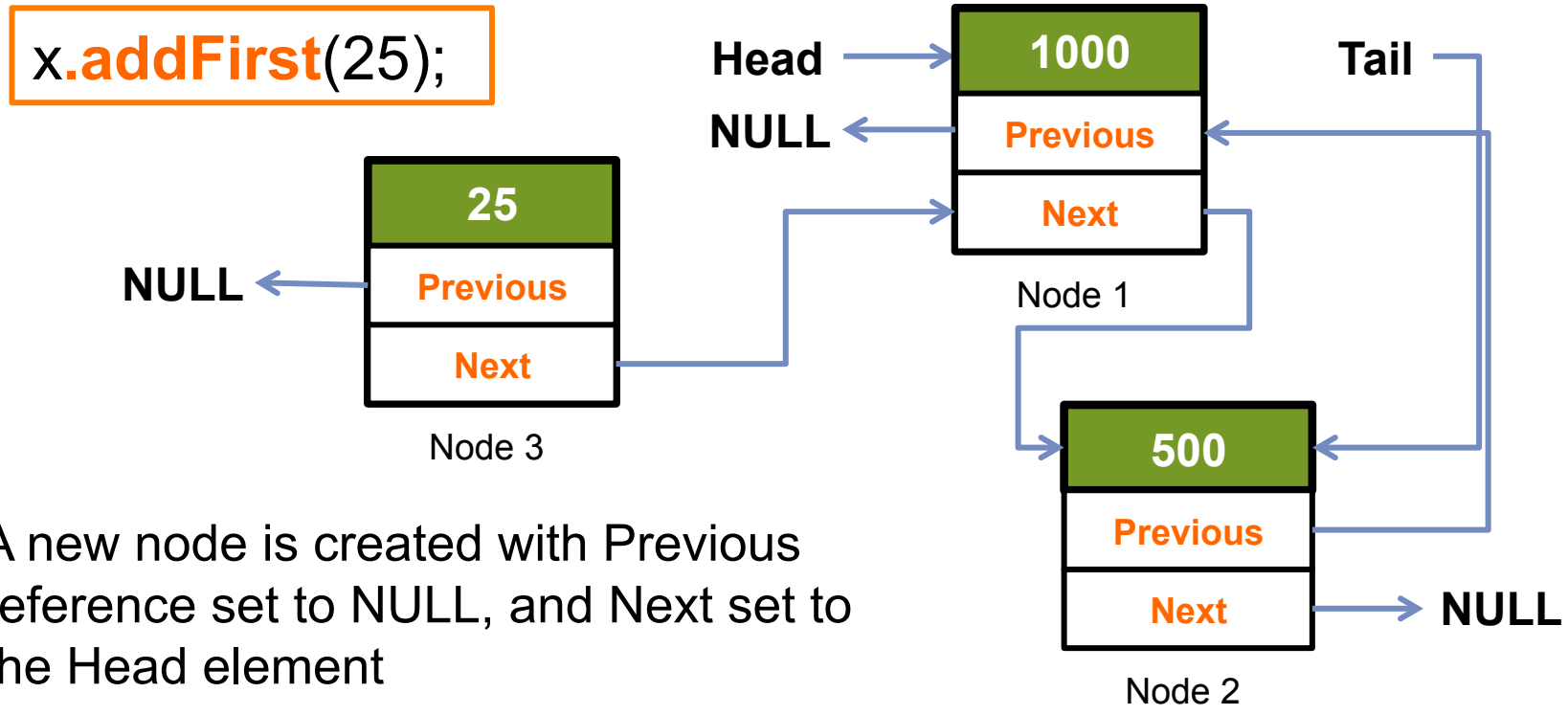
- To add another element to the list

```
}else{
      //create new Node
      IntNode newNode=new IntNode(value);
      newNode.setPrevious(tail);
      tail.setNext(newNode);
      tail=newNode;
}
```



ADDING ELEMENTS

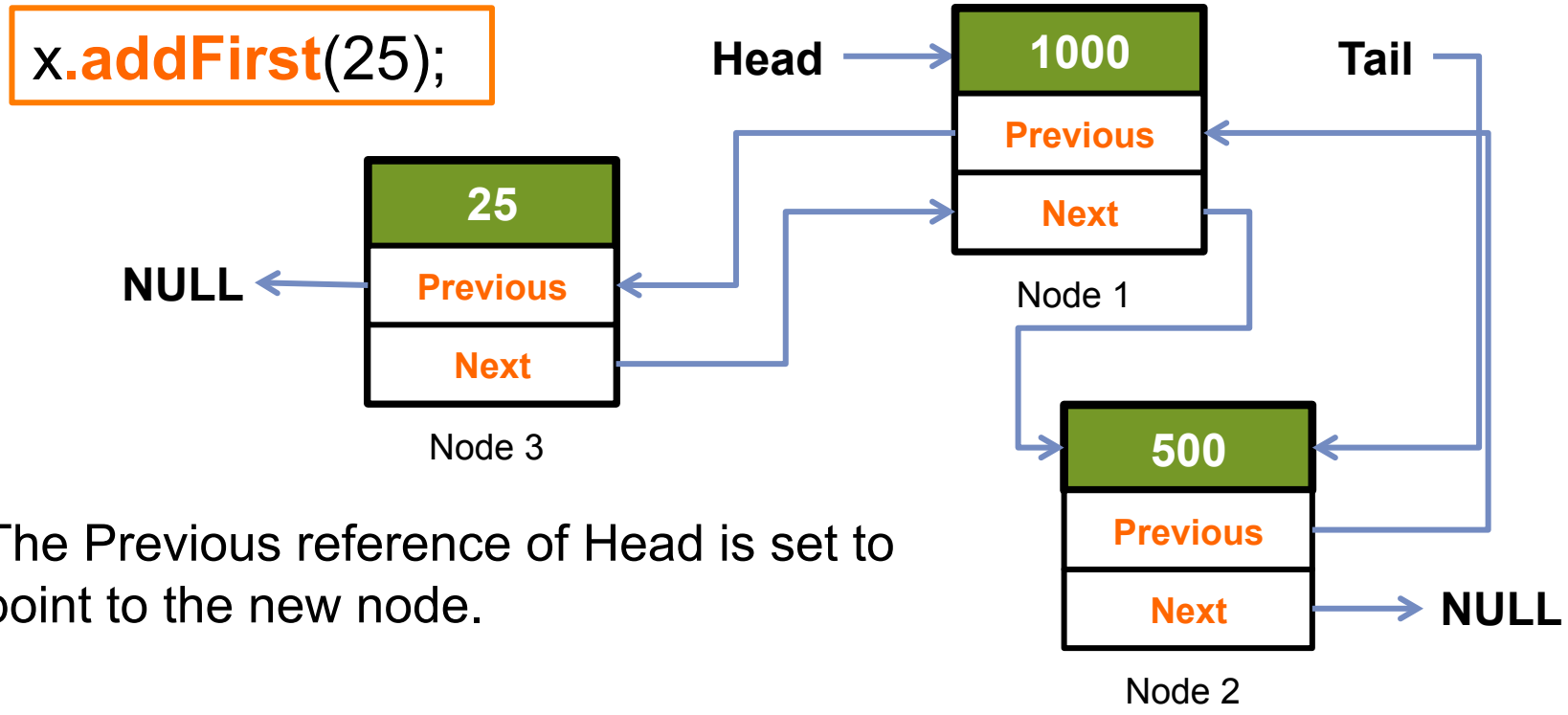
- **To add an element to the beginning of the list**



- A new node is created with Previous reference set to NULL, and Next set to the Head element

ADDING ELEMENTS

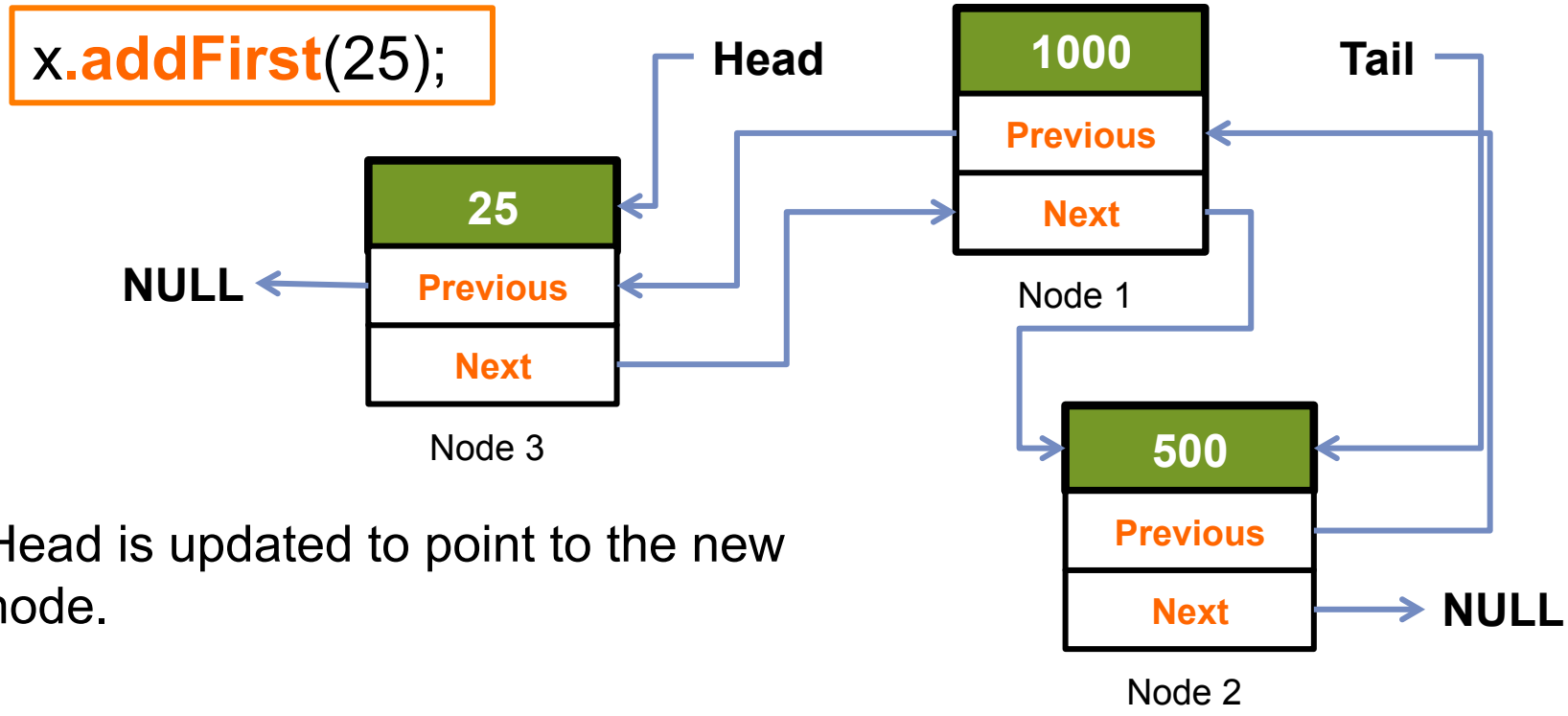
- **To add an element to the beginning of the list**



- The Previous reference of Head is set to point to the new node.

ADDING ELEMENTS

- **To add an element to the beginning of the list**

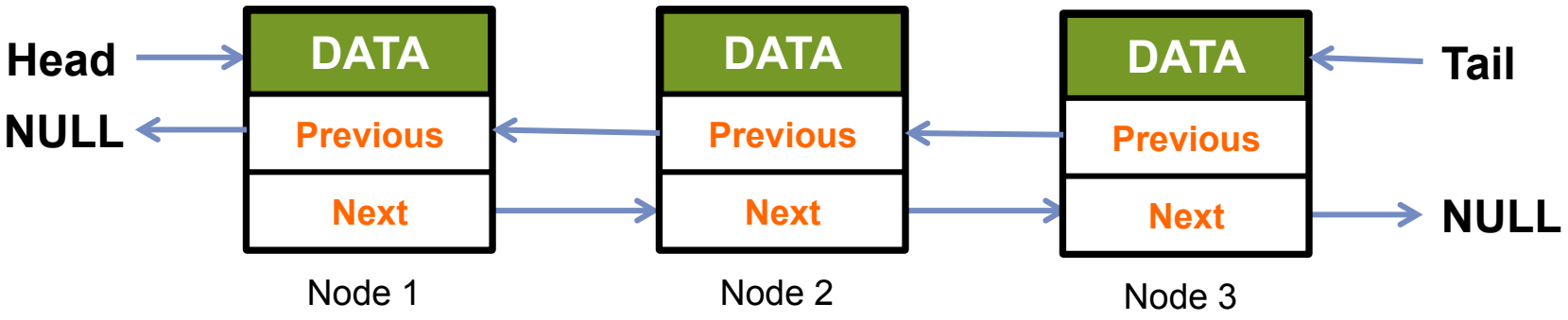


- Head is updated to point to the new node.

TRAVERSING A LINKED LIST

```
//print the values in the linked list
public void print(){
    //start from the head node
    IntNode currentNode=head;
    while(currentNode !=null){
        //print the value of the current node
        System.out.print(currentNode.getData() +"\t");
        //go to the next node
        currentNode=currentNode.getNext();
    }
    System.out.println();
}
```

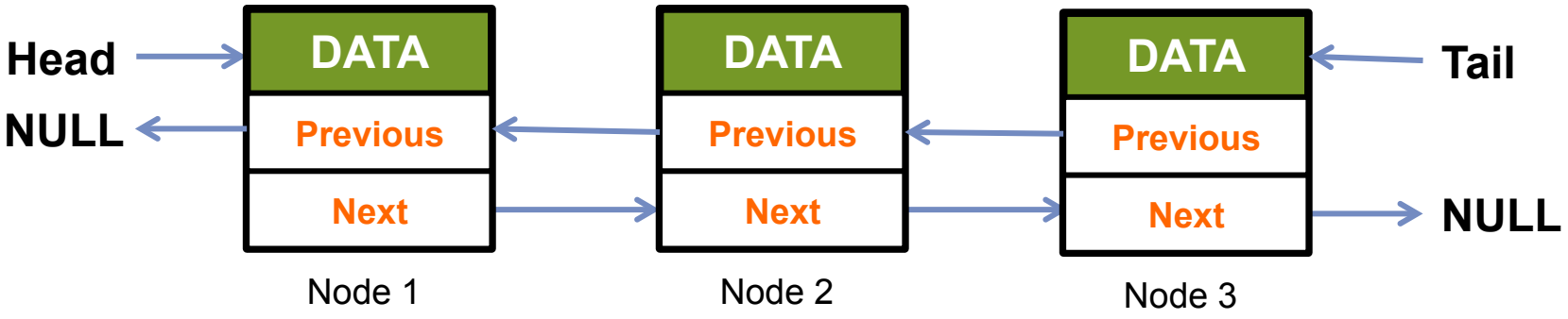
DELETING AN ELEMENT



Find the element by traversing the list

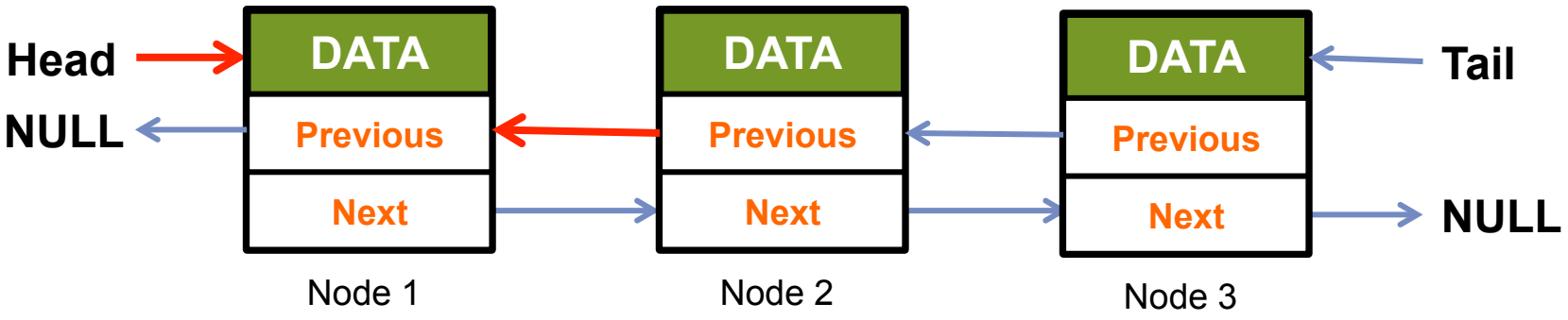
- **If there is only one element in this list, simply set head and tail to null.**

DELETING AN ELEMENT



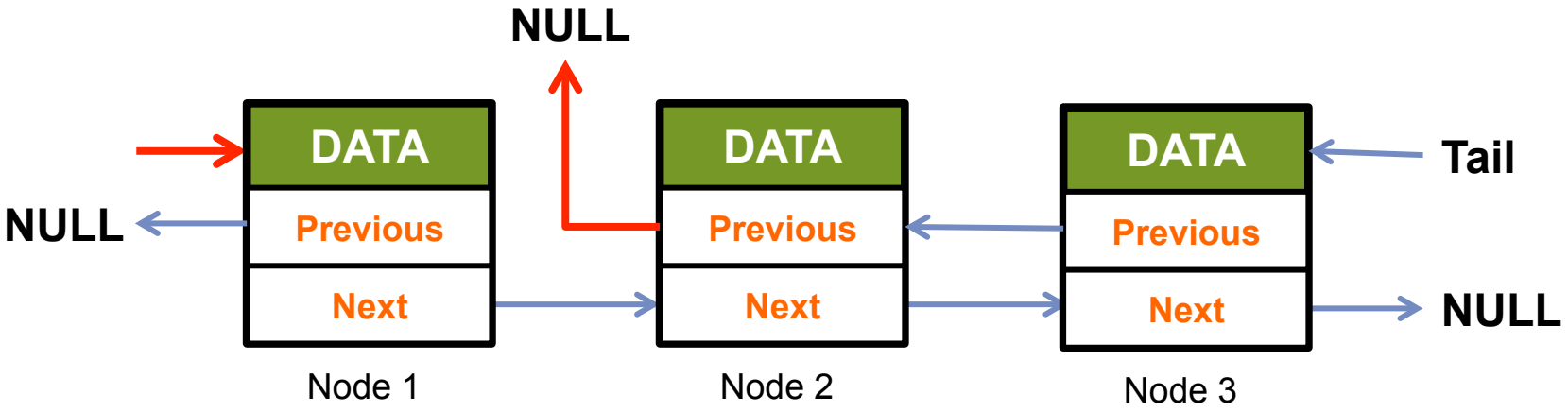
- **To delete a node, remove all references to that node.**

DELETING AN ELEMENT



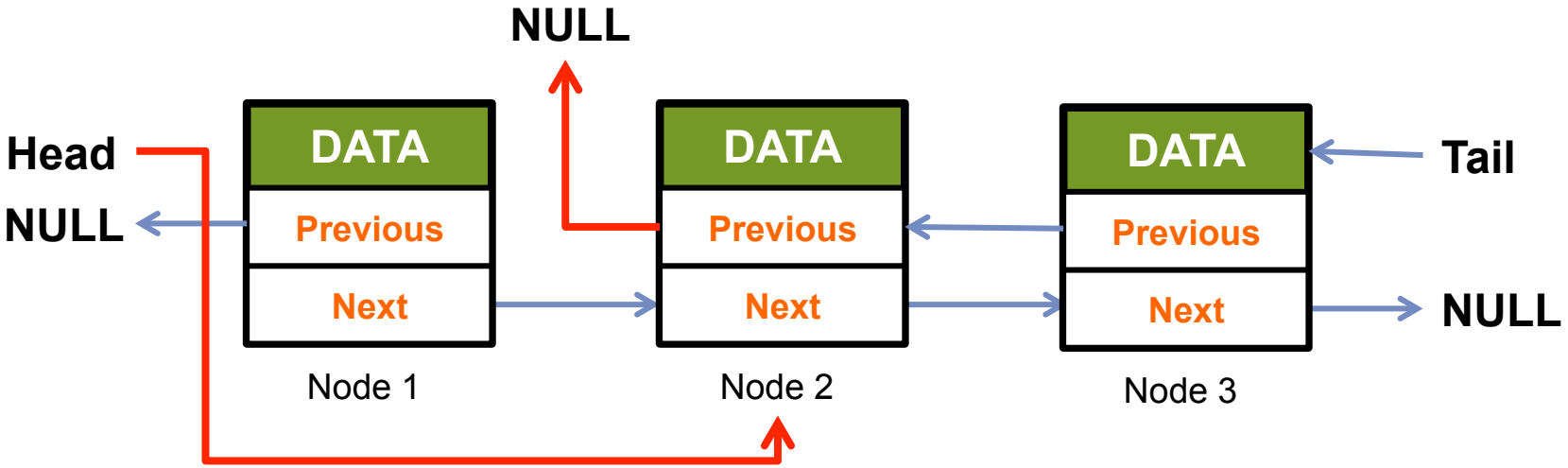
- **Delete node 1:**
  - Two references to node 1: **Head** and **node2.previous**

DELETING AN ELEMENT



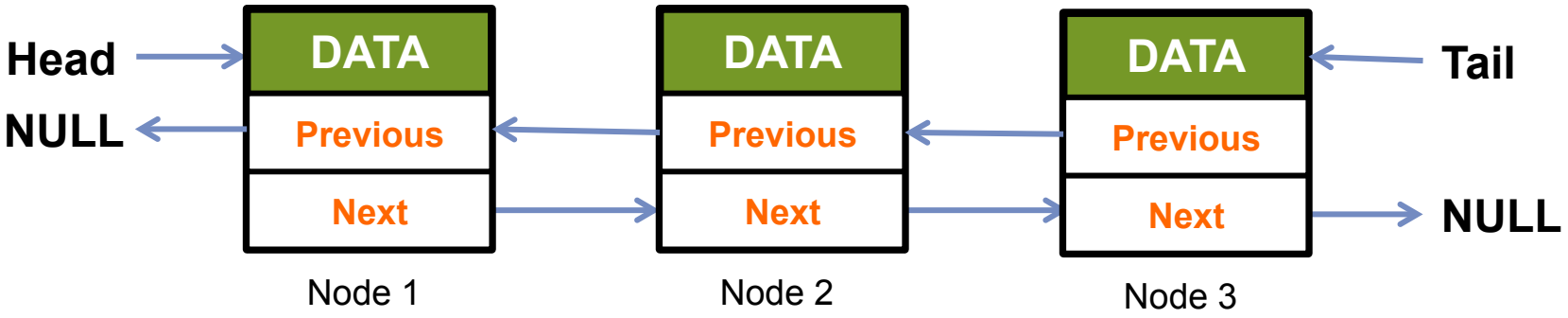
- **Delete node 1:**
  - Set node2.previous to null

DELETING AN ELEMENT



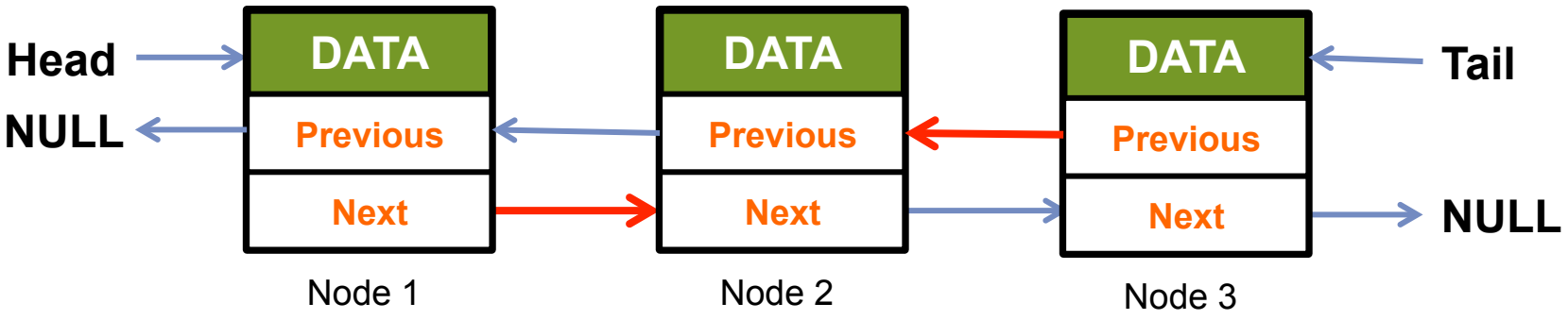
- **Delete node 1:**
  - Set **Head** to node 2

DELETING AN ELEMENT



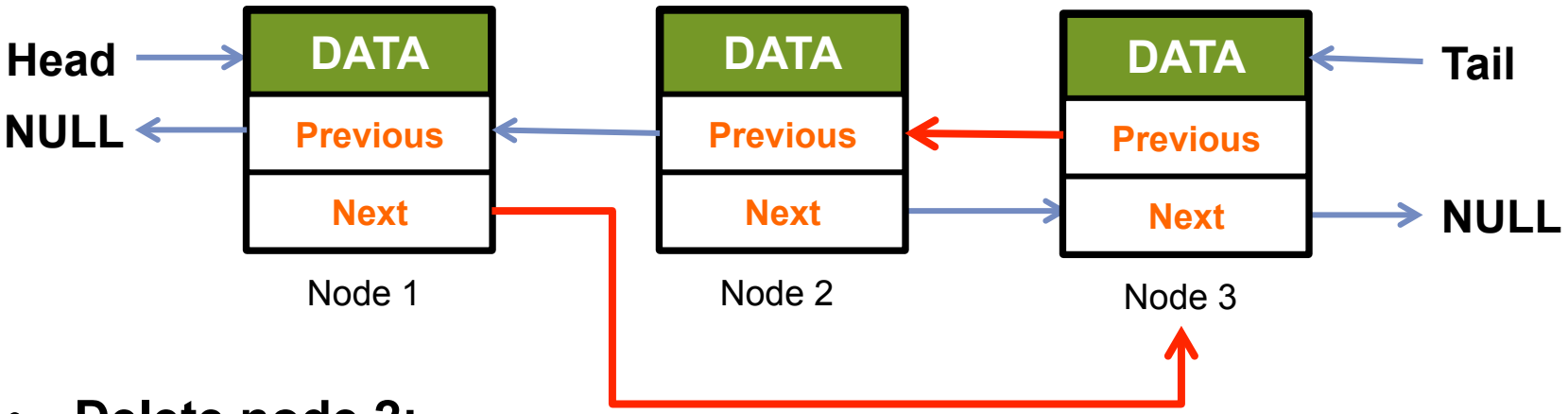
- **Delete node 2:**

DELETING AN ELEMENT



- **Delete node 2:**
  - Two references to node 2: **node1.next** and **node3.previous**

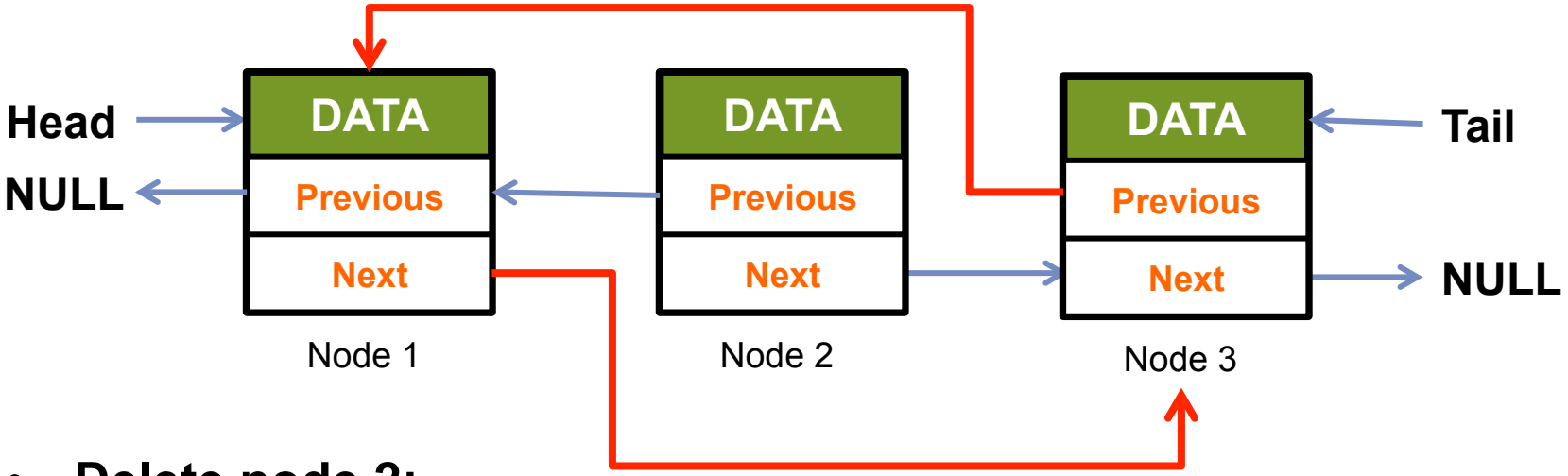
DELETING AN ELEMENT



- **Delete node 2:**
  - Set node1.next to node 3

  (previous node's next reference to current node's next)

DELETING AN ELEMENT



- **Delete node 2:**
  - Set node3.previous to node 1

  (next node's previous reference to current node's previous)

DELETION METHOD

```
Voice deleteElement (int k) {
        Traverse the linked list starting from Head
        check if the list element == k
        if element k is found
                Updated references that point to this node
                does this node have a previous node?
                        if yes update previous.next
                        otherwise, update head
                does it have a next node?
                        if yes update next.previous
                        otherwise, update tail
}
```