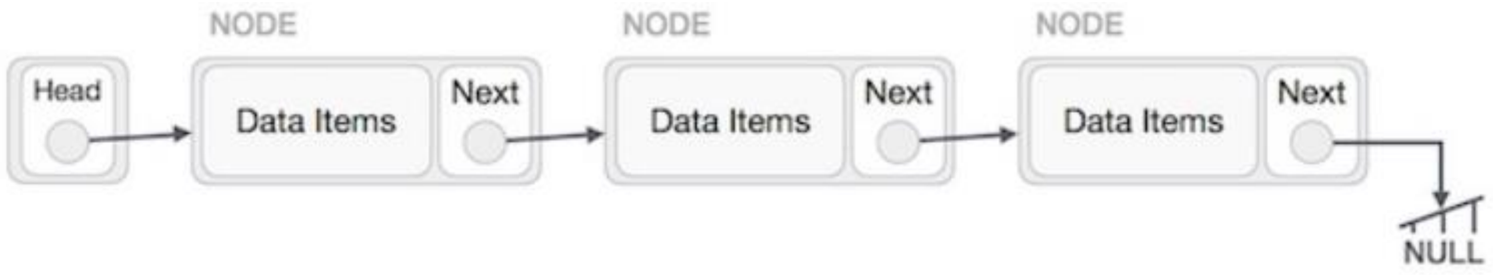
Linked List

Anoop S Babu
Faculty Associate
Dept. of Computer Science & Engineering
bsanoop@am.amrita.edu

Single Linked List

- A linked list data structure includes a series of connected nodes.
- Each node store the data(s) and the address of the next node.
  - **Node** – Each node of a linked list can store a data(s) called an element.
  - **Next** – Each node of a linked list contains a link to the next node called Next.
  - **LinkedList** – A Linked List contains the connection link to the first node called **Head.**



Single Linked List: Implementation in Python

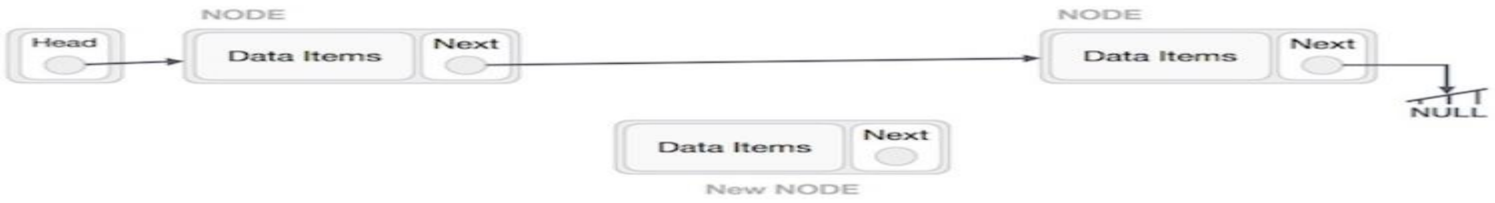
```
class Node:
  # Creating a node
  def __init__(self, item):
    self.item = item
    self.next = None
class LinkedList:
    def __init__(self):
        self.head = None

linked_list = LinkedList()
# Assign item values
linked_list.head = Node(1)
second = Node(2)
third = Node(3)
# Connect nodes
linked_list.head.next = second
second.next = third
# Displaying the linked list
while linked_list.head != None:
    print(linked_list.head.item, end=" ")
    linked_list.head = linked_list.head.next
```

Adding Elements to a Linked List

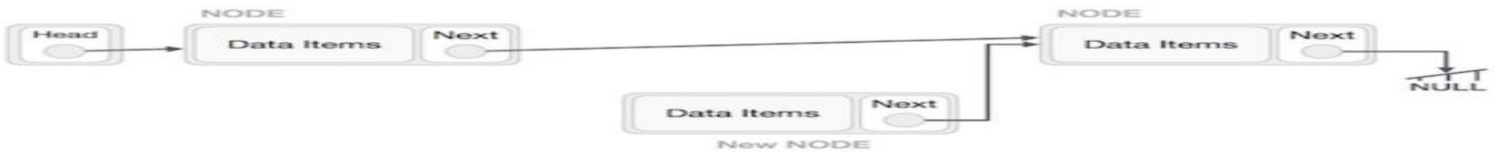
- Elements can be added to either the beginning, middle or end of the linked list.
- **Add to the beginning**
    - Allocate memory for new node
    - Store data
    - Change next of new node to point to head
    - Change head to point to recently created node
- **Add to the End**
    - Allocate memory for new node
    - Store data
    - Traverse to last node
    - Change next of last node to recently created node

Adding Elements to a Linked List

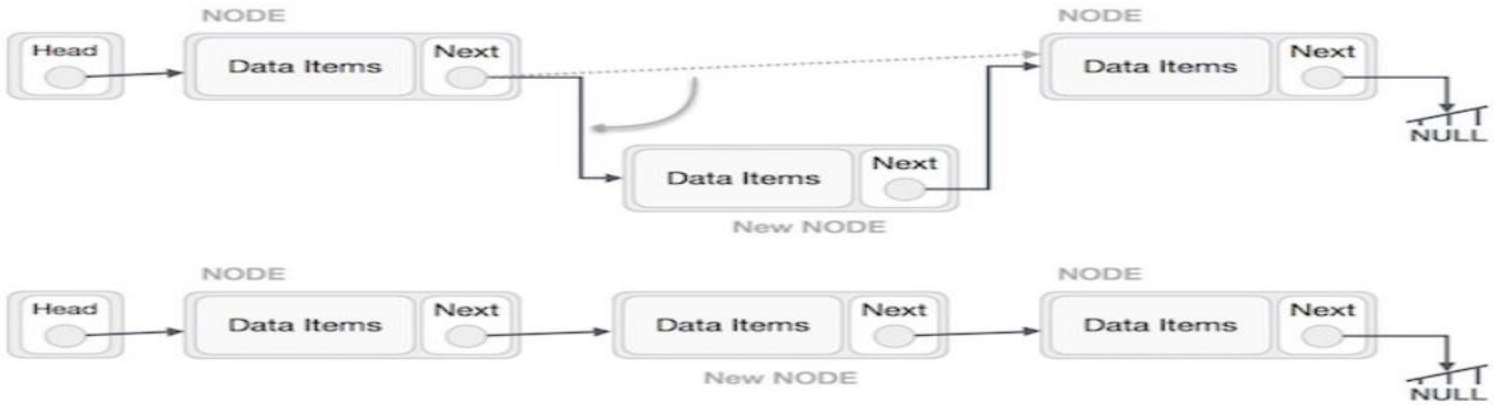
- **Add to the Middle**
  - Allocate memory and store data for new node



  - Traverse to node just before the required position of new node



  - Change next pointers to include new node in between



Deleting Element from a Linked List

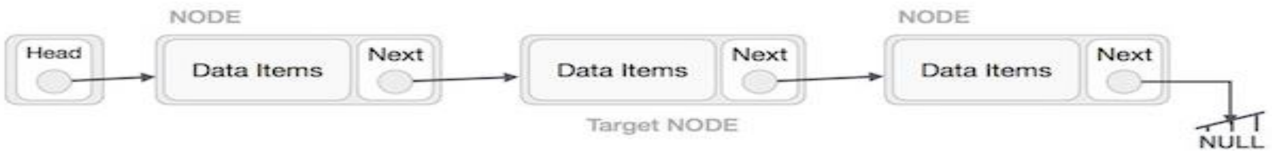
- **Delete from beginning**
  - Point head to the second node

- **Delete from end**
  - Traverse to second last element
  - Change its next pointer to null

- **Delete from middle**
  - Traverse to element before the element to be deleted
  - Change next pointers to exclude the node from the chain

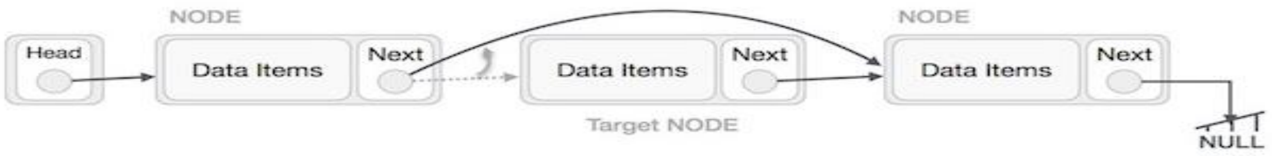
Deleting Element from a Linked List

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.



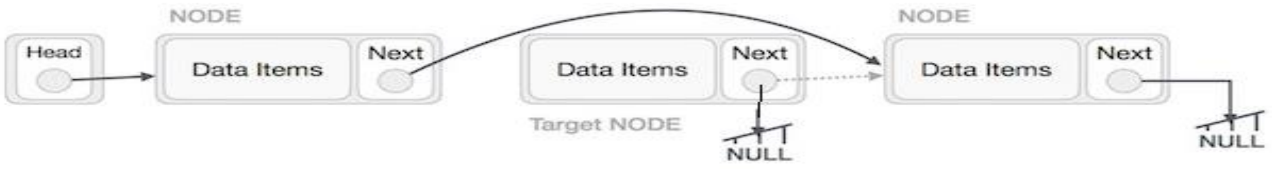
The left (previous) node of the target node now should point to the next node of the target node −

```
LeftNode.next -> TargetNode.next;
```

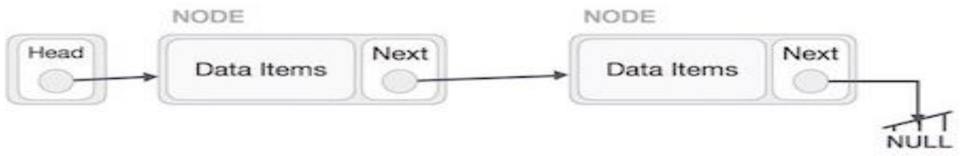


This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

```
TargetNode.next -> NULL;
```



We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.



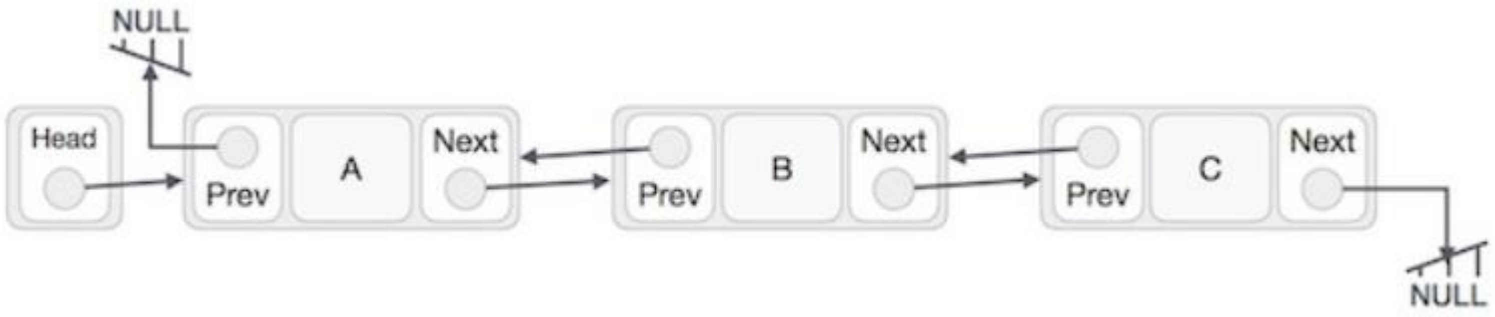
Types of Linked List

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Doubly Linked List

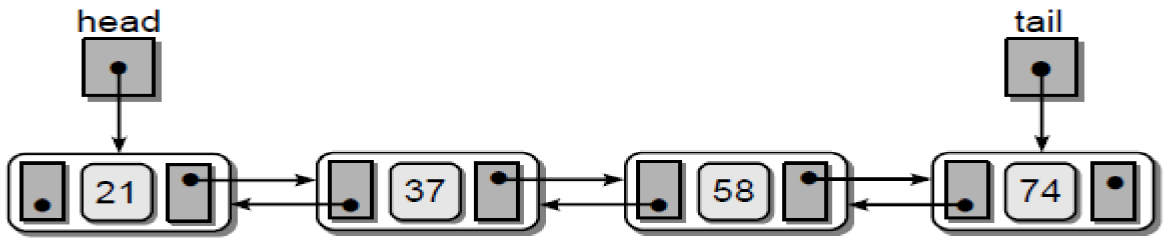
Doubly Linked List

- Doubly Linked List is a variation of **Linked list in which navigation is possible in both ways**, either forward and backward.
  - **Node** – Each node of a linked list can store a data(s) called an element.
  - **Next** – Each node of a linked list contains a link to the next node called Next.
  - **Prev** - Each node of a linked list contains a link to the previous node called Prev.
  - **LinkedList** – A Linked List contains the connection link to the first node called **Head.**

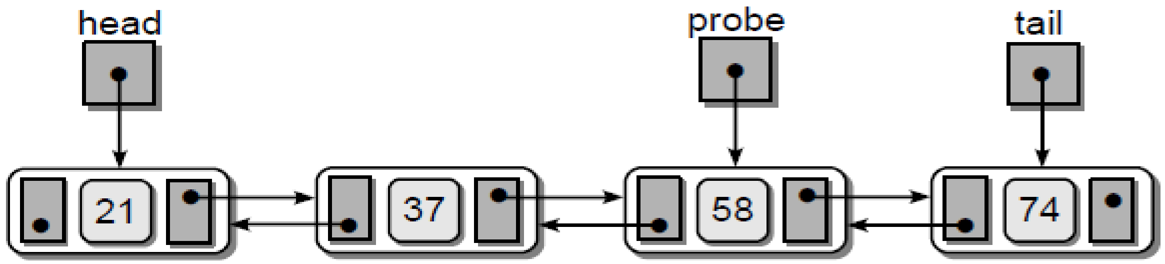


Doubly Linked List: Basic Operations

- **Traversal**
  - The doubly linked list allows for traversals from front to back or back to front.

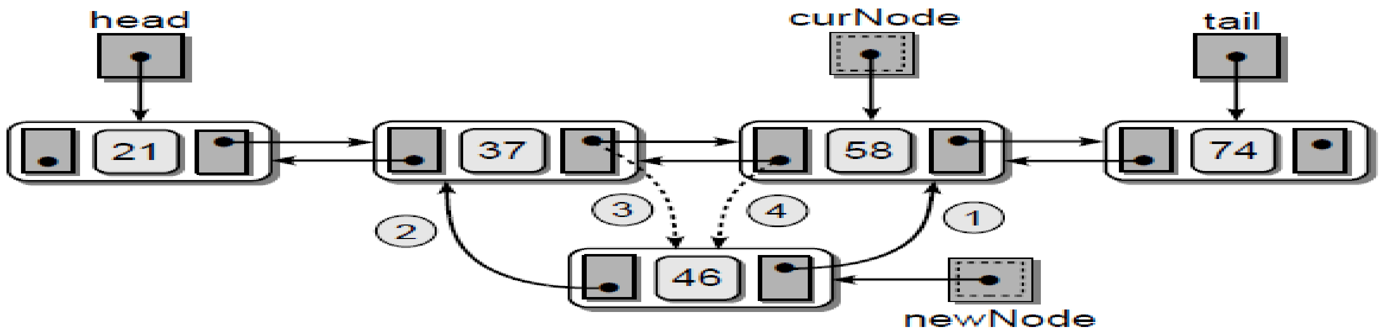


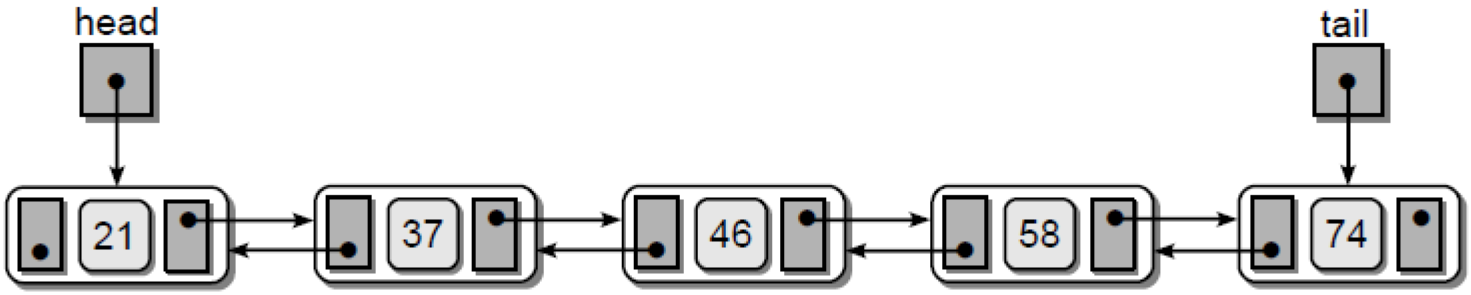
- **Searching**
  - Searching for a specific item in a doubly linked list, based on key value, is same as for a singly linked list.
  - Start with the first node and iterate through the list until we find the target item



Doubly Linked List: Basic Operations

- **Adding a new element before an item**

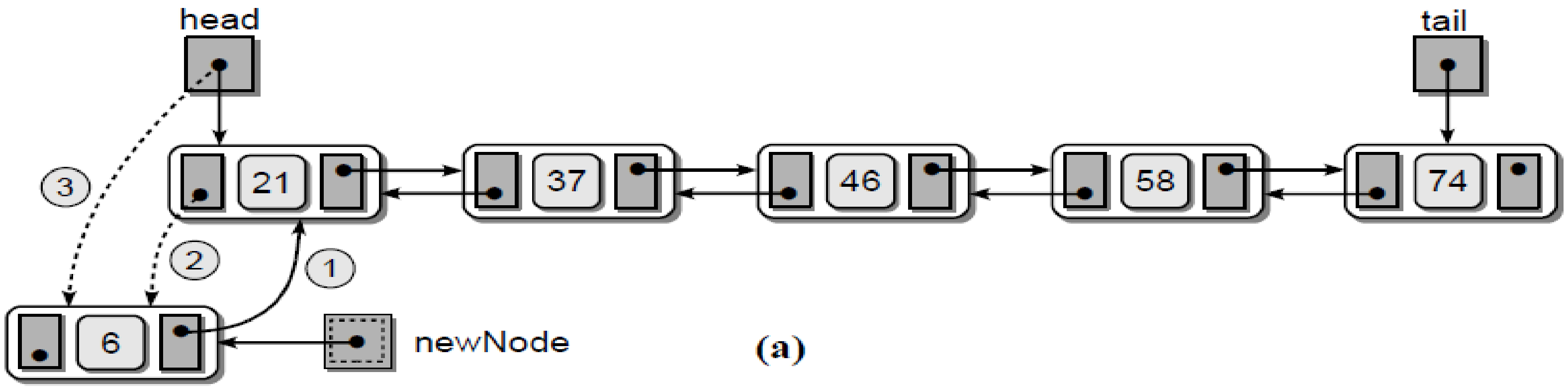


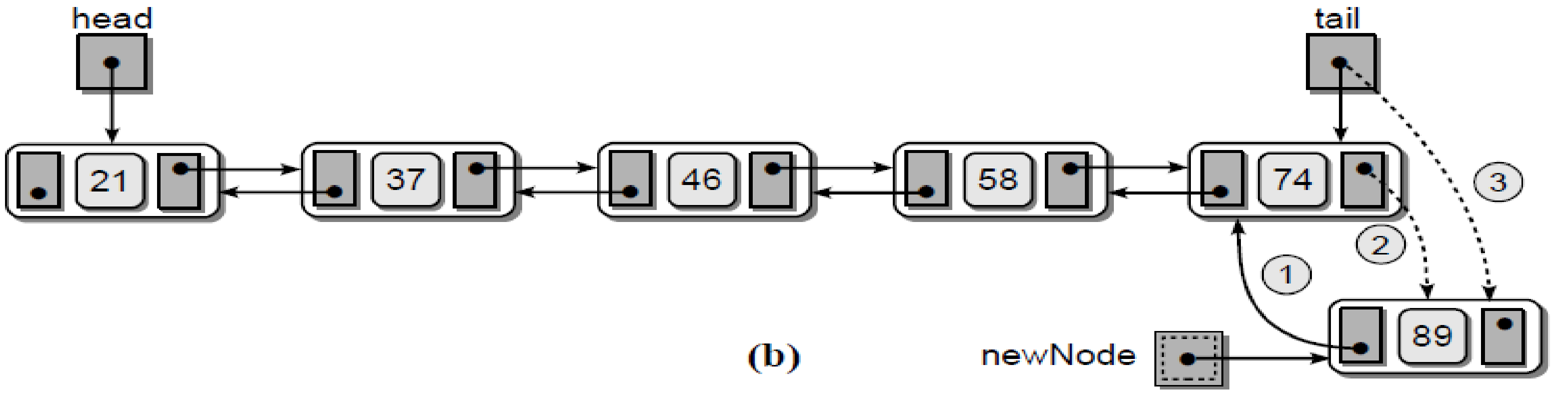


The result of inserting the new node into the doubly linked list.

Doubly Linked List: Basic Operations

- **Adding an element at front and back**



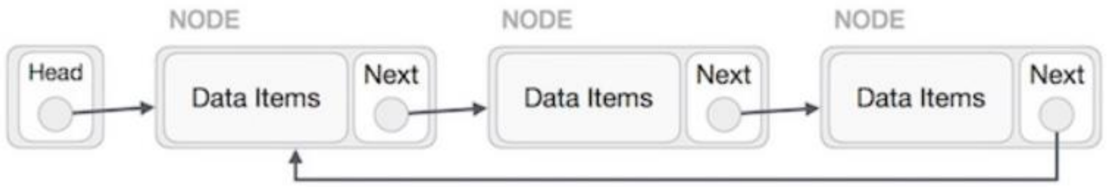


The links required to insert a new node into a sorted doubly linked list at the (a) front or (b) back.

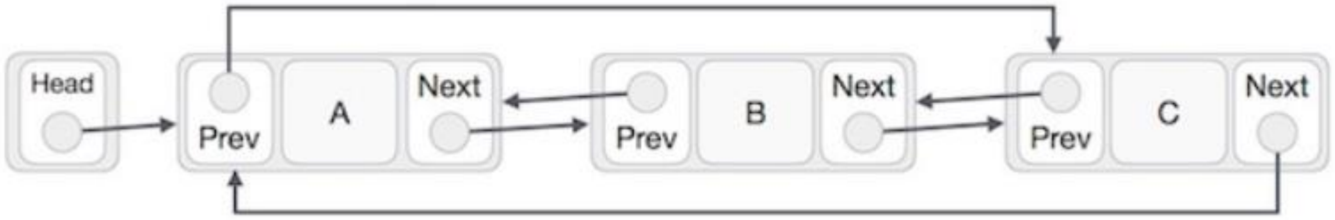
Circular Linked List

Circular Linked List

- A circular linked list is an another variation of a linked list in which the **last element is linked to the first element**.
- The circular linked list allows for a complete traversal of the nodes starting with any node in the list.
- A circular linked list can be either singly linked or doubly linked.
  - In singly linked list, next pointer of last item points to the first item.



  - In the doubly linked list, prev pointer of the first item points to the last item as well.


Head
Prev
A
Next
Prev
B
Next
Prev
C
Next