LINKED LIST IN DETAILS
SINGLY LINKED LIST

ROMI FADILLAH RAHMAT

SINGLY LINKED LISTS

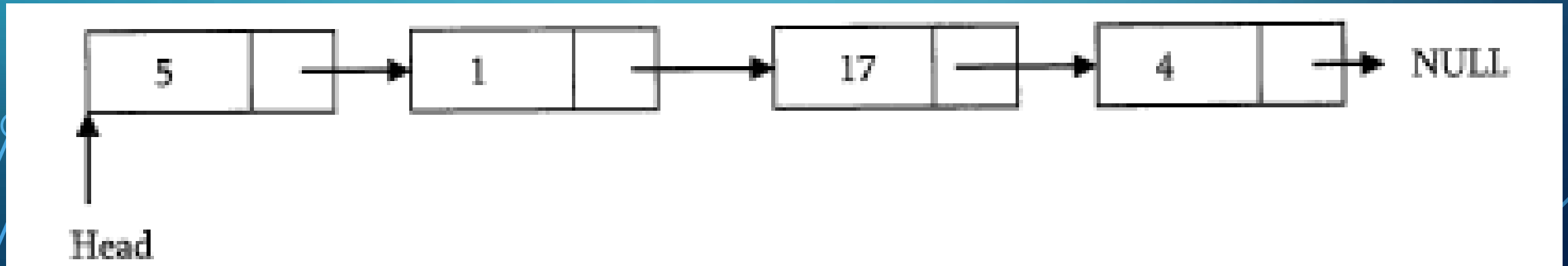
- Generally linked list means a singly linked list. The list contains a number of nodes in which each node has a next pointer to the following element. The link of the last node in the list is NULL which indicates end of the list.

Following is a type declaration for a linked list of integers:

```
struct ListNode{
        int data;
        struct ListNode *next; }
```

TRAVERS / DISPLAY (TRAVERSING THE LINKED LIST)

- Let us assume that the head points to the first node of the list. To traverse the list we do :
  - Follow the pointers
  - Display the contents of the nodes (or counts) as they are traversed
  - Stop when the next pointer points to NULL.



TRAVERS / DISPLAY (TRAVERSING THE LINKED LIST)

- Example code :

```
int listlength(struct ListNode *head){
        struct ListNode *current = head;
        int count = 0;
        while (current != NULL){
                count++;
                current = current→ next; }
        return count; }
```

Time Complexity : O(n) for scanning complete list of size n. Size complexity : O(1) for creating one temporary variable

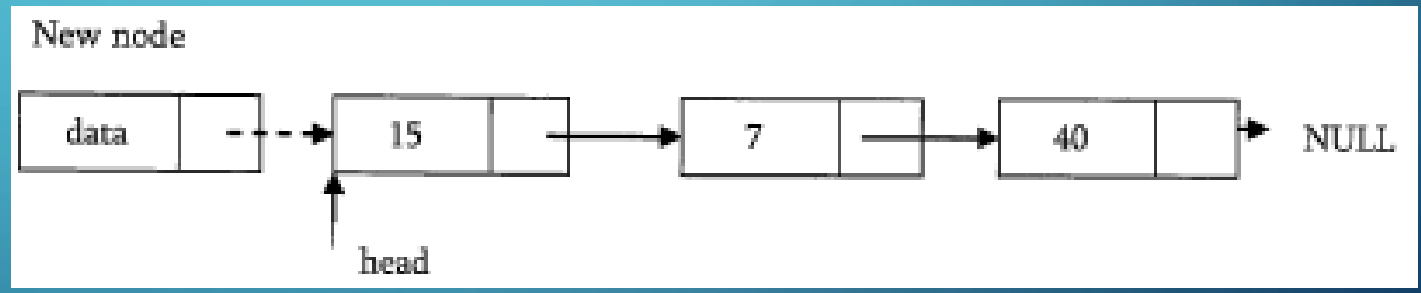
INSERT OPERATION (SINGLY)

- Insertion into a singly linked list has three cases :
  - Inserting a new node before the head (at the beginning)
  - Inserting a new node after the tail (at the end of the list).
  - Inserting a new node at the middle of the list (random location)

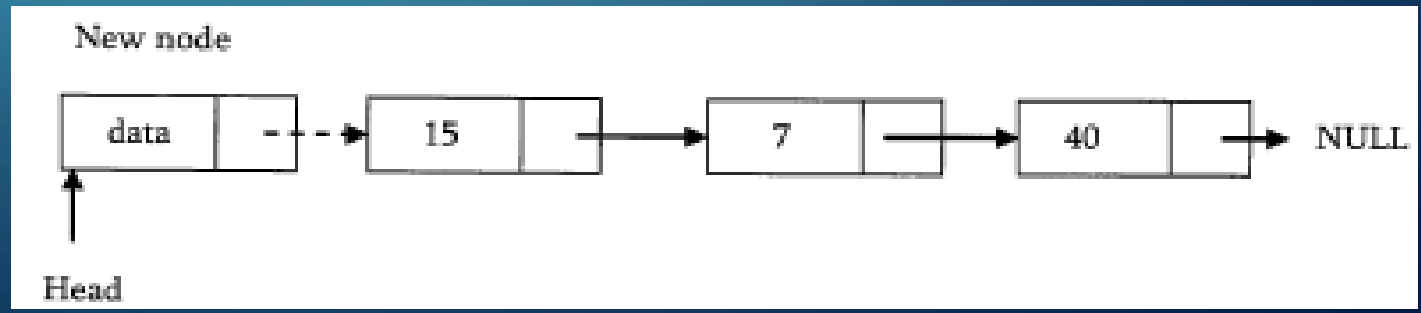
- To insert an element in the linked list at some position *p*, assume that after inserting the element, the position of this new node is *p*.

INSERTING A NODE AT THE BEGINNING (SINGLY)

- In this case, a new node is inserted before the current head node. *Only one next pointer* needs to be modified (new node's next pointer) and it can be done in two steps :
- 1. Update the next pointer of new node to point to the current head.

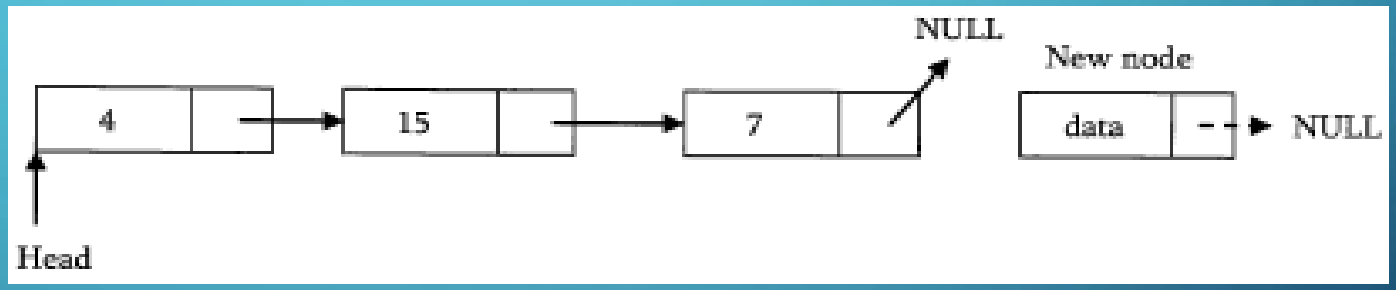


- 2. Update head pointer to point to the new node.

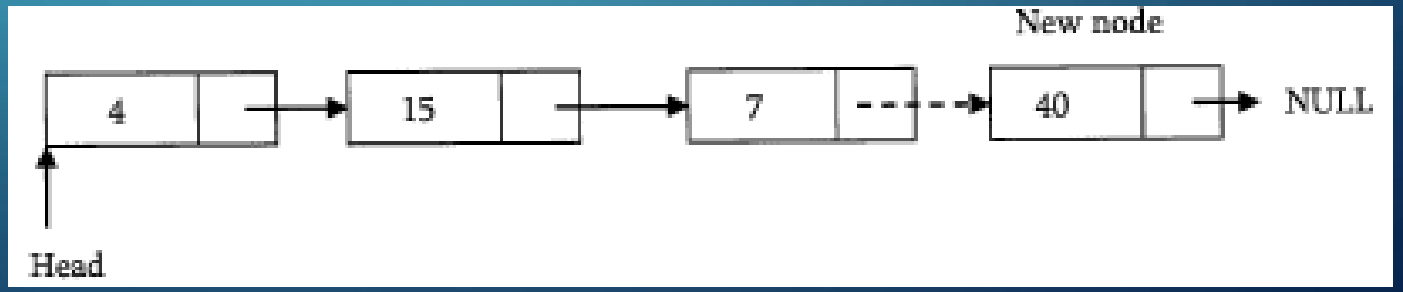


INSERTING A NODE AT THE ENDING (SINGLY)

- In this case, we need to modify two next pointers (last nodes next pointer and new nodes next pointer).
- 1. New nodes next pointer points to NULL

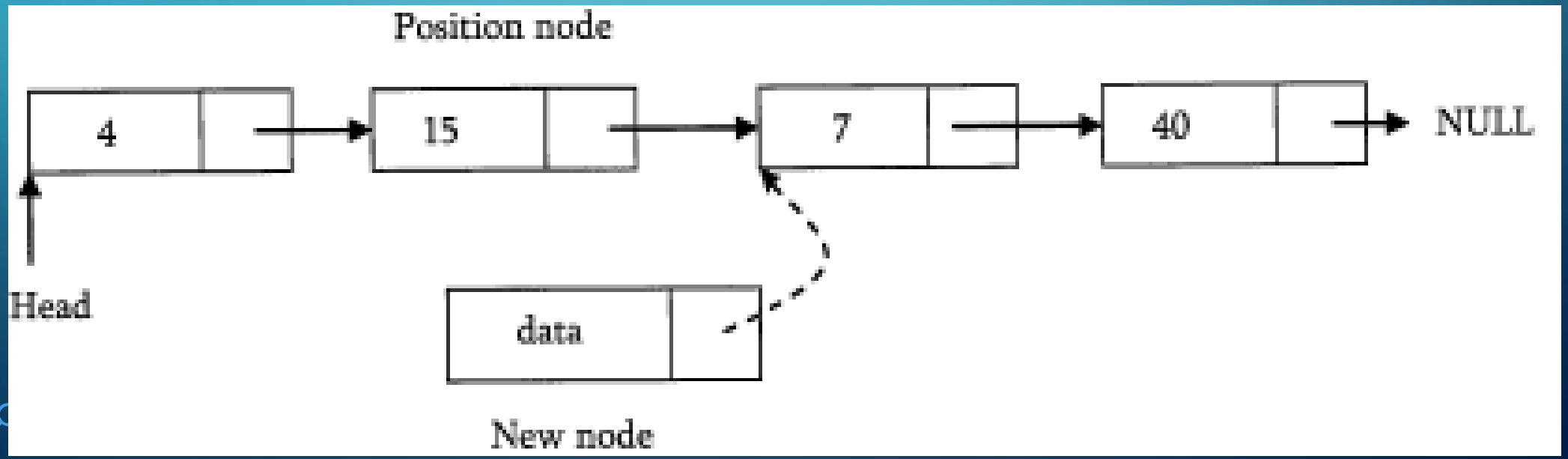


- 2. Last nodes next pointer points to the new node.

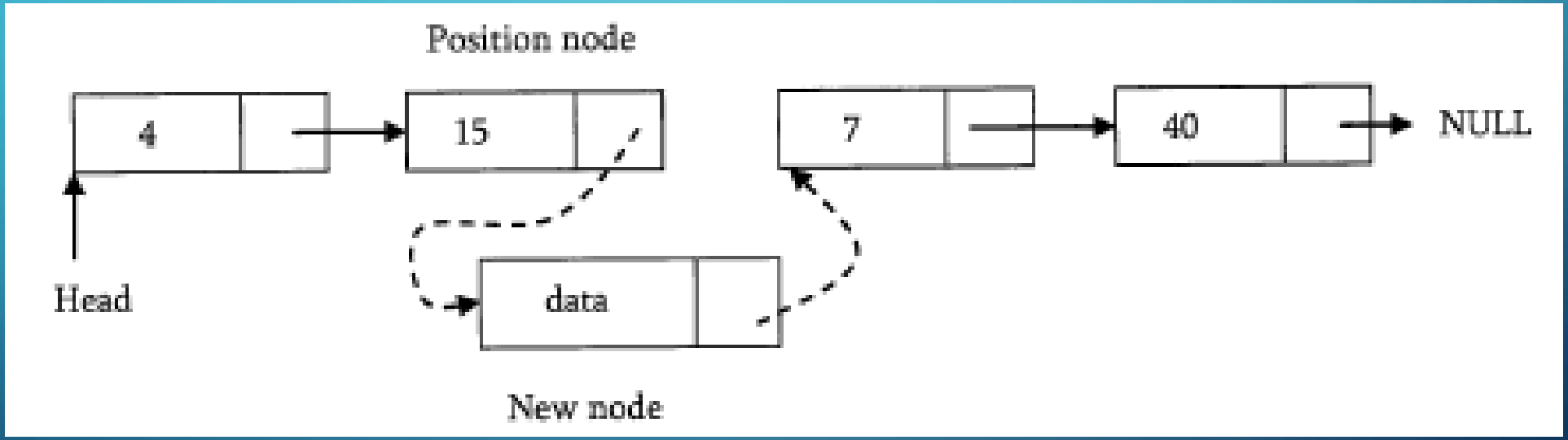


INSERTING A NODE AT THE MIDDLE (SINGLY)

- Let us assume that we are given a position where we want to insert the new node. In this case we need to modify two next pointers.
- 1. If we want to add an element at position 3 then we stop at positon 2. That means we traverse 2 nodes and insert the new node. That second position is called *position* node. New node points to the next node of the positon where we want to add this node.



- 2. Position nodes next pointer now points to the new node.



SAMPLE CODE FOR INSERTION

```
void InsertInLinkedList (struct ListNode **head, int data, int position) {
        int k = 1;
        struct ListNode *p, *q, *newNode;
        newNode = (ListNode *) malloc(sizeof(struct ListNode));
        if(!newNode) {                                          //Always Check for Memory Errors
                printf ("Memory Error");
                return;
        }
        newNode->data=data;
        p = *head;
        if(position == 1)  {                    // Inserting at the beginning
                newNode->next = p;
                *head = newNode;
        }
```

SAMPLE CODE FOR INSERTION

```
        else {      //Traverse the list until position-1
                    while ((p != NULL) && ( k < position - 1)) {
                                k++;
                                q = p;
                                p = p→next;
                    }
                    if(p == NULL)     {                    //Inserting at the end
                                q→next = newNode;
                                newNode→next = NULL;
                    }
                    else {                                 // In the middle
                                q→next = newNode;
                                newNode→next = p;
                    }
        }
}
```

COMPLEXITY

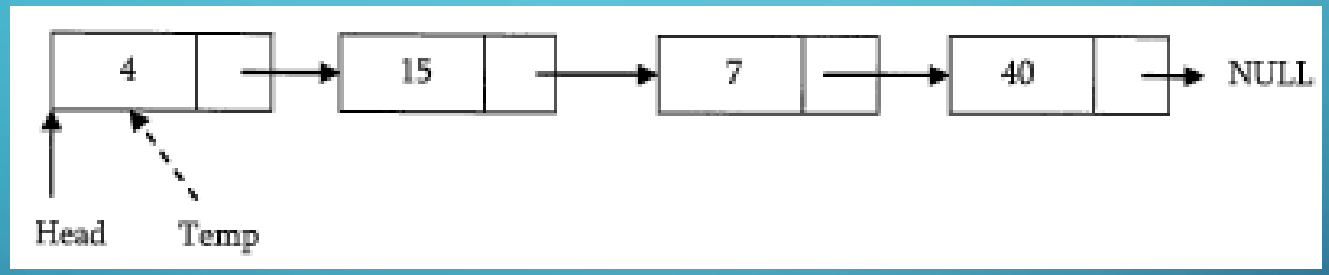
- Those code can be implemented separately.
- Time Complexity : O(n), since the worst we may need to insert the node at end of the list. Space Complexity: O(1) for creating one temporary variable.

DELETION OPERATION

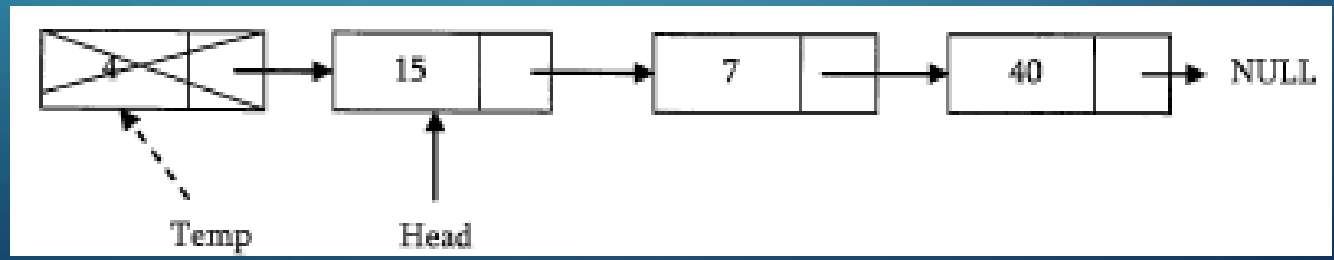
- We have 3 type of deletions such as
  - Deleting the first node
  - Deleting the last node
  - Deleting an intermediate node

DELETING THE FIRST NODE (SINGLY)

- First node (current head node) is removed from the list. It can be done in two steps:
- 1. Create a temporary node which will point to same node as that of head

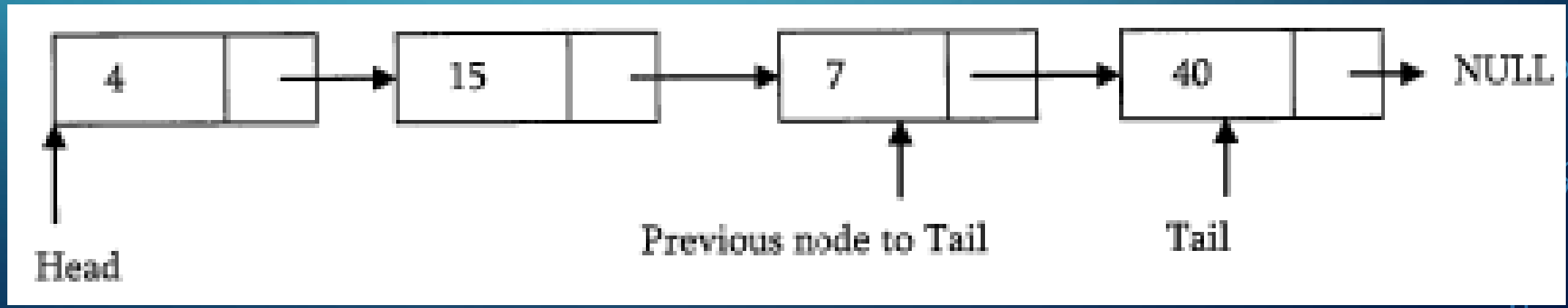


- 2. Move the head nodes pointer to the next node and dispose the temporary node.

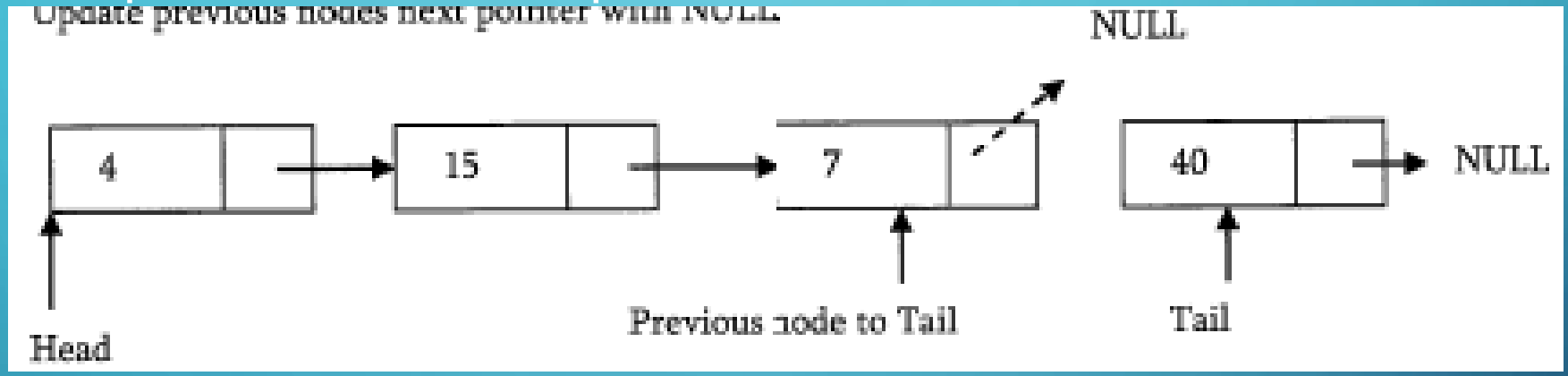


DELETING THE LAST NODE (SINGLY)

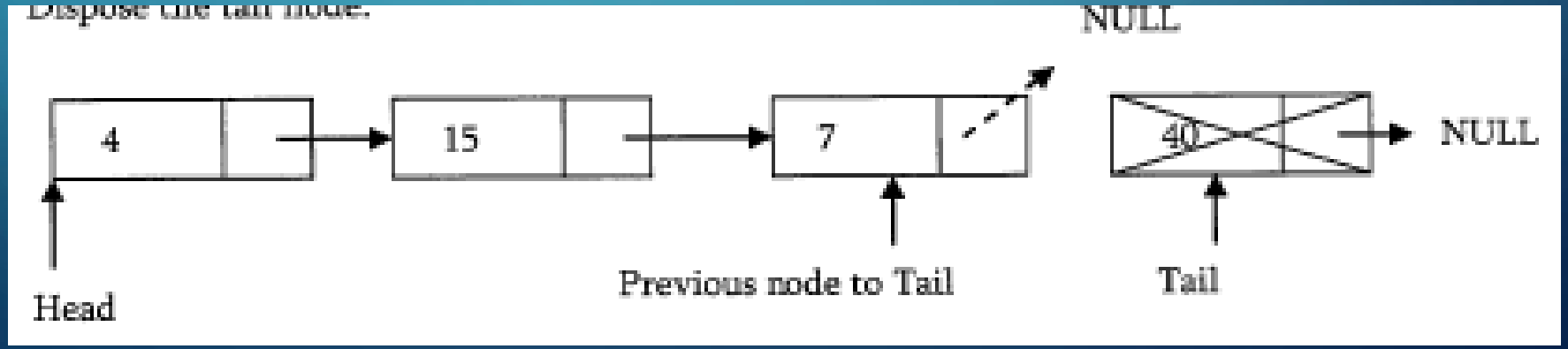
- In this case, last node is removed from the list. This operation is a bit trickier than removing the first node, because algorithm should find a node, which is previous to the tail first. It can be done in three steps :
- 1. Traverse the list and while traversing maintain the previous node address also. By the time we reach the end of the list, we will have two pointers one pointing to the tail need and other pointing to the node before tail node.



- 2. Update previous nodes next pointer with NULL

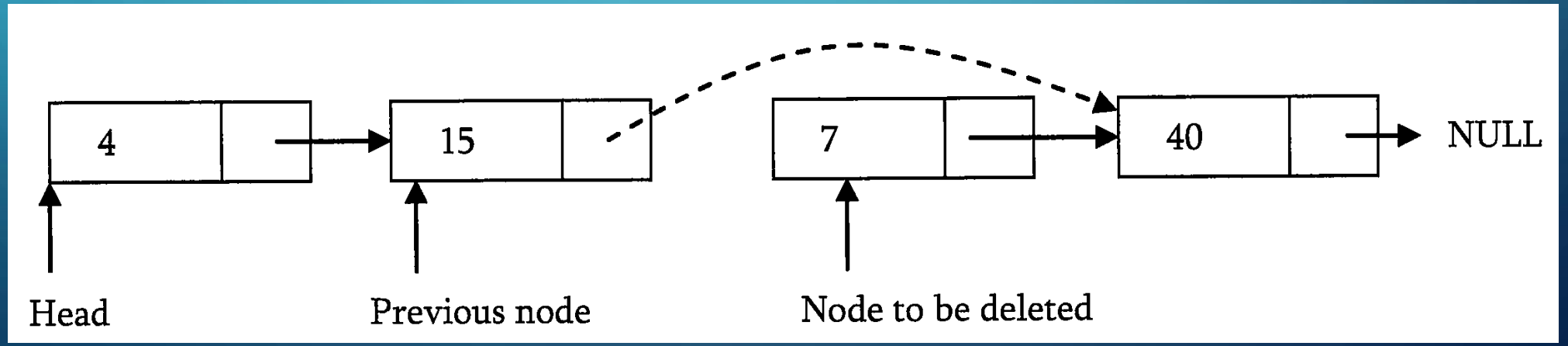


- 3. Dispose the tail node

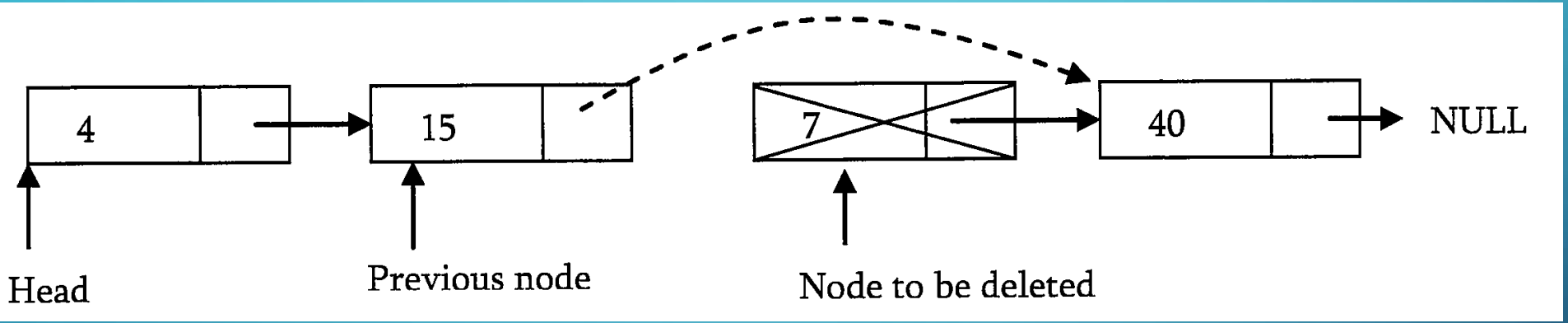


DELETING INTERMEDIATE NODE (SINGLY)

- In this case node to be removed is always located between two nodes. Head an tail links are not updated in this case. It can be done in two steps :
- 1. As similar to previous case, maintain previous node while traversing the list. Once we found the node to be deleted, change the previous node pointer to next pointer of the node to be deleted.



- 2. Dispose the current node to be deleted



SAMPLE CODES

```
void DeleteNodeFromLinkedList (struct ListNode **head, int position) {
        int k = 1;
        struct ListNode *p, *q;

    if(*head == NULL) {
            printf ("List Empty");
            return;
    }
    p = *head;
    if(position == 1) {                    /* from the beginning */
            p = *head;
            *head = *head→next;
            free (p); return;
    }
```

```
        else {      //Traverse the list until the position from which we want to delete
                    while ((p != NULL) && (k < position - 1)) {
                                k++;   q = p;
                                p = p→next;
                    }
                    if(p == NULL)                           /* At the end */
                                printf ("Position does not exist.");
                    else {                                  /* From the middle */
                                q→next = p→next;
                                free(p);
                    }
        }
}
```

LINKED LIST IN DETAILS
DOUBLY LINKED LIST

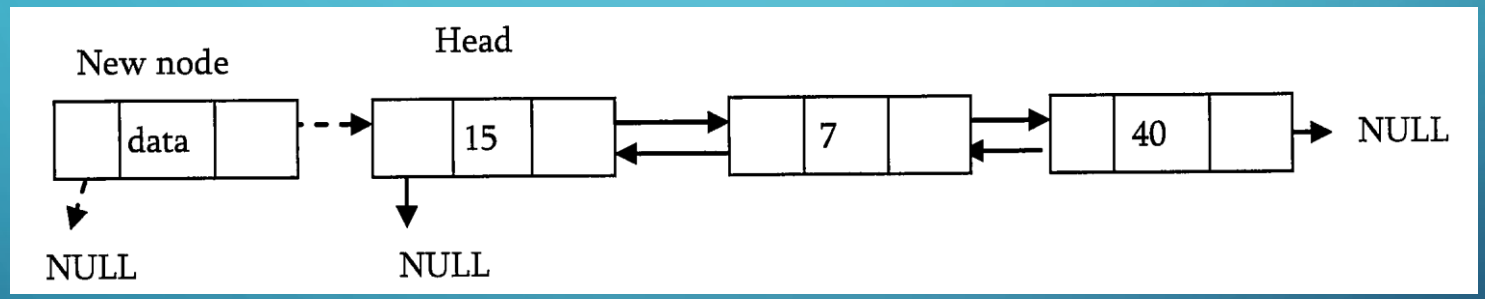
ROMI FADILLAH RAHMAT

DOUBLY LINKED LIST INSERTION

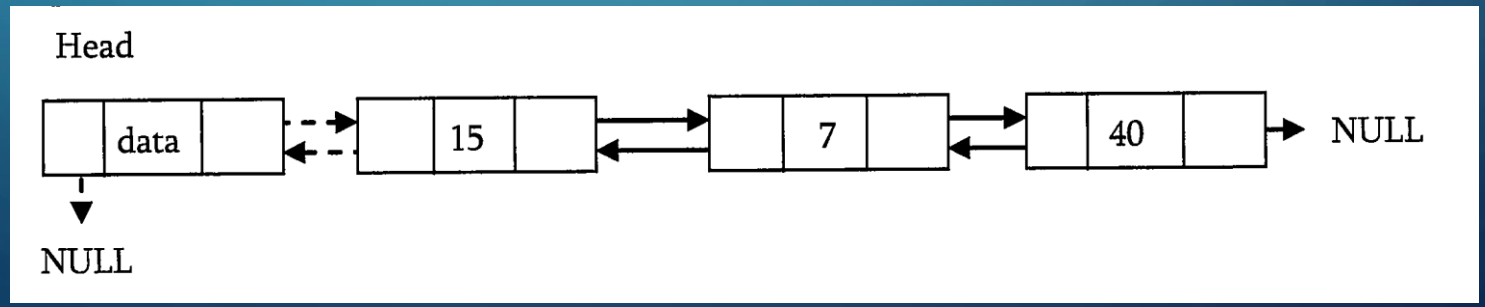
- Insertion into a doubly linked list has three cases :
  - Inserting a new node before the head
  - Inserting a new node after the tail
  - Inserting a new node at the middle of the list

INSERTING A NEW NODE IN THE BEGINNING - DOUBLY

- In this case, new node is inserted before the head node. Previous and next pointers need to be modified and it can be done in two steps :
  - Update the right pointer of new node to point to the current head node and also make the left pointer of new node as NULL

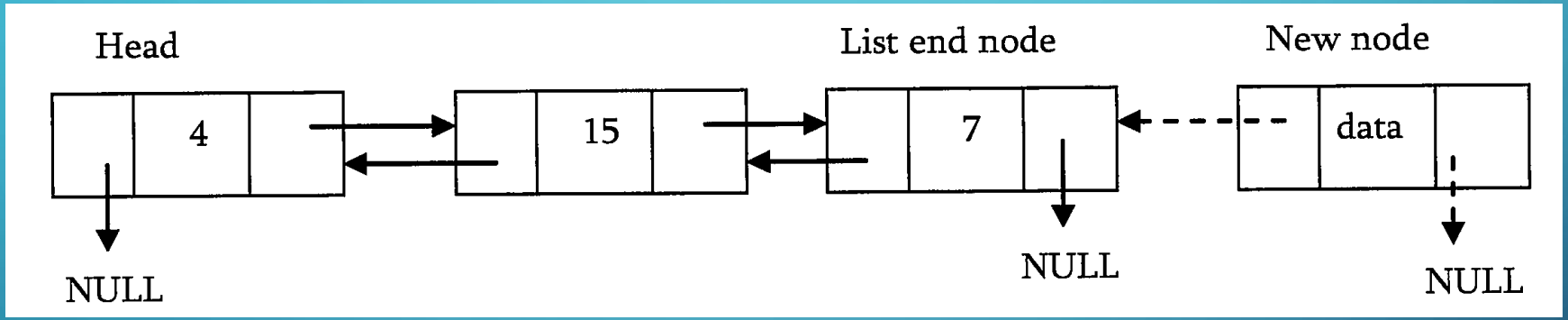


  - Update head nodes left pointer to point the new node and make new node as head.

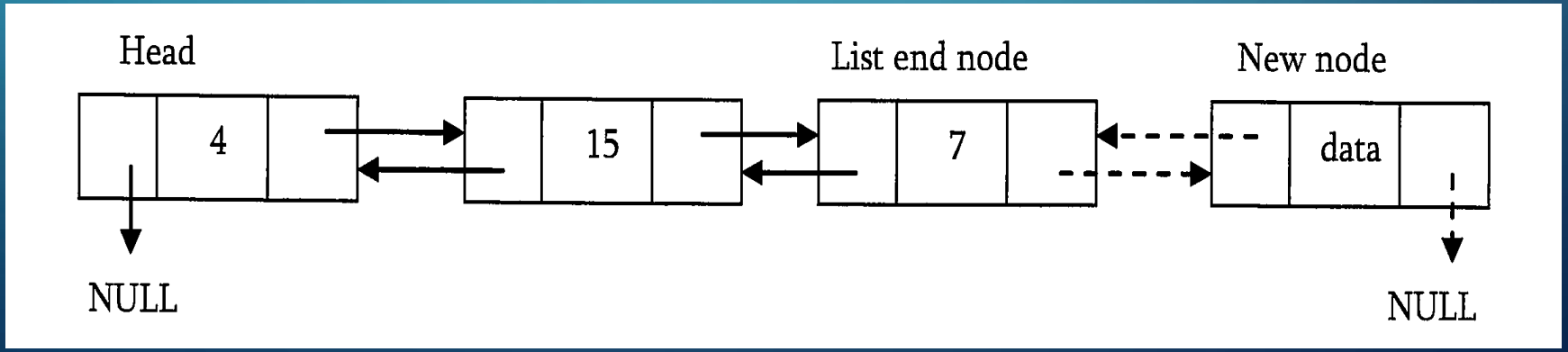


INSERTING A NODE AT THE ENDING - DOUBLY

- In this case, traverse the list till the end and insert the new node.
  - New node right pointer points to NULL and left pointer points to the end of the list

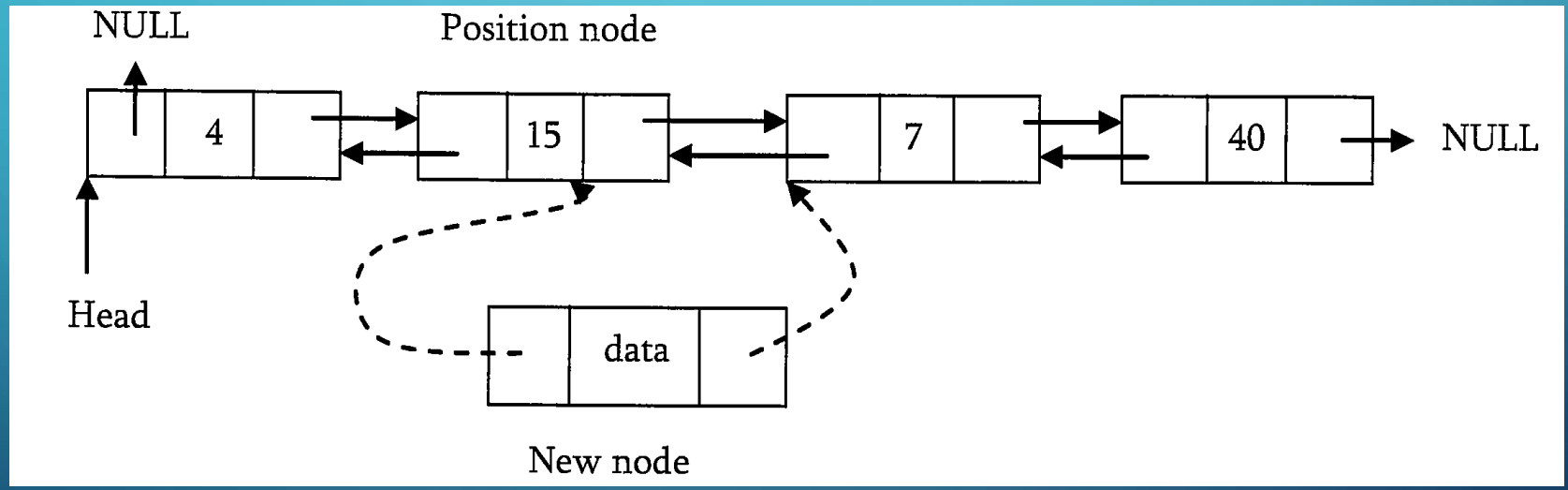


  - Update right of pointer of last node to point to new node

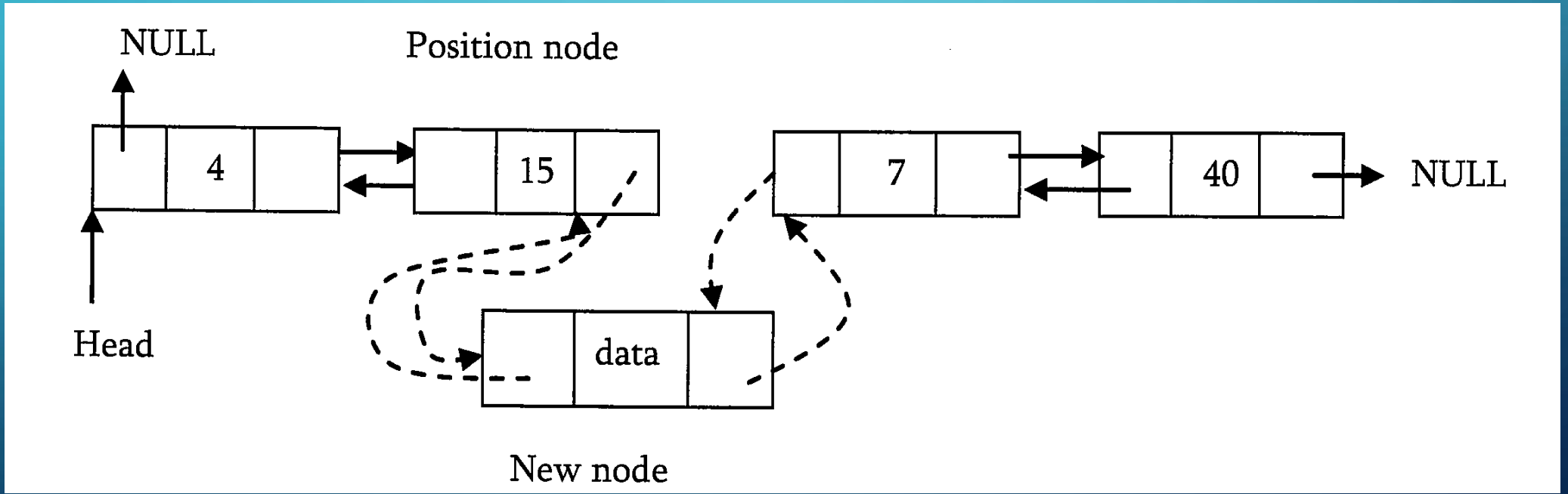


INSERTING A NODE AT THE MIDDLE - DOUBLY

- Traverse the list until the position node, then insert the new node.
  - New node's right pointer points to the next node of the position node where we want to insert the new node. Also, new node left pointer point to the position node.

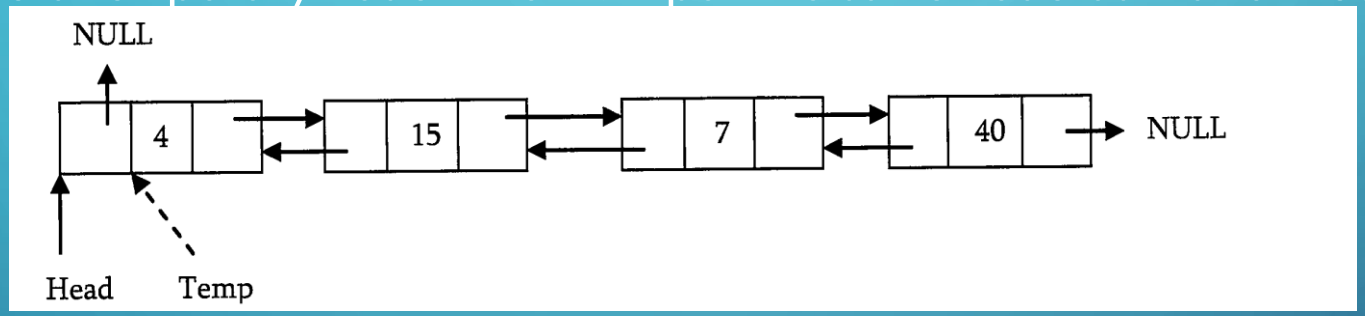


- Position node right pointer points to the new node and the next node of the position nodes left pointer points to new node.

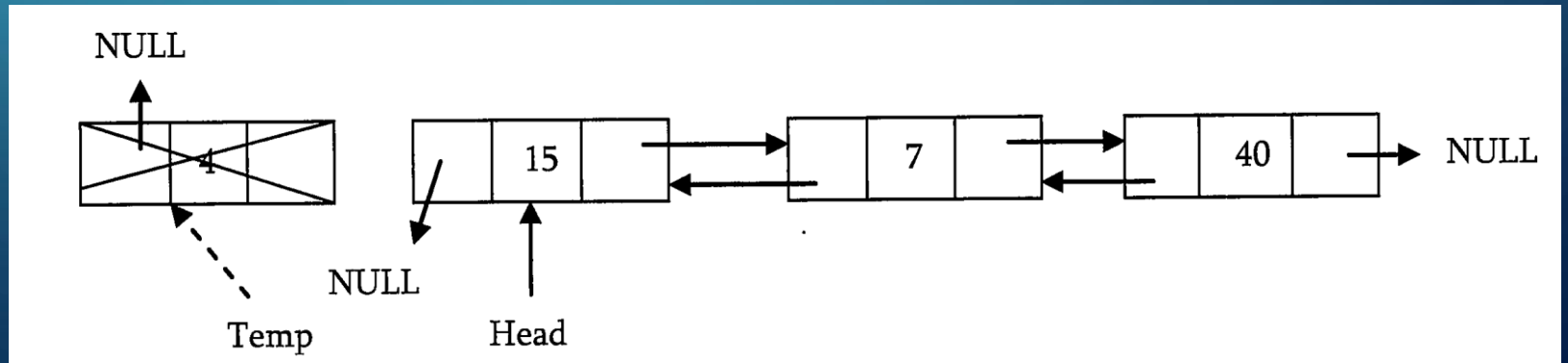


DELETING THE FIRST NODE - DOUBLY

- In this case, first node (current head node) is removed from the list. It can be done in two steps :
  - Create a temporary node which will point to same node as that of head.

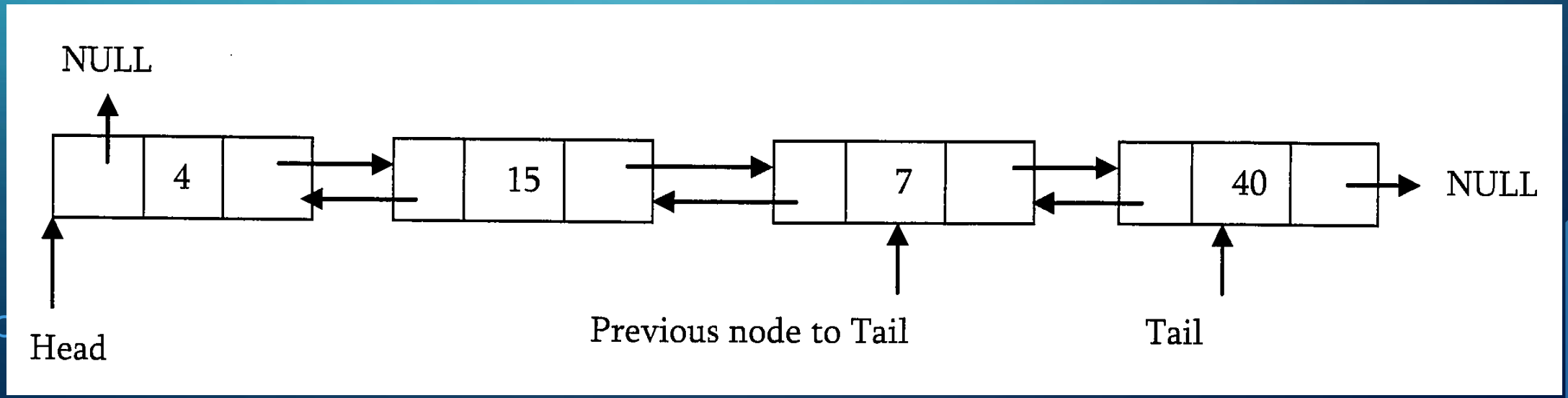


  - Now, move the head nodes pointer to the next node and change the heads left pointer to NULL. Then dispose the temporary node

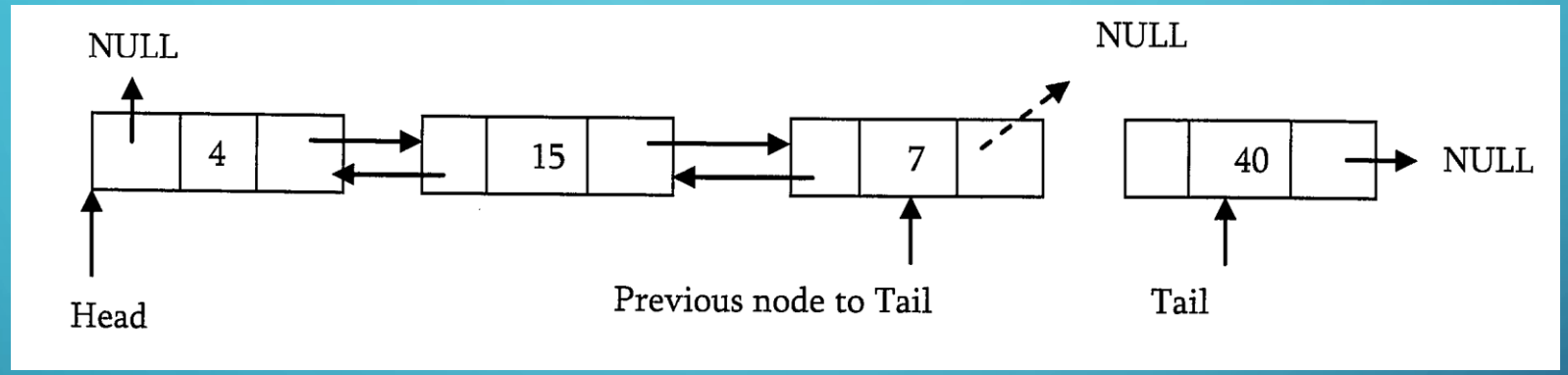


DELETING THE LAST NODE - DOUBLY

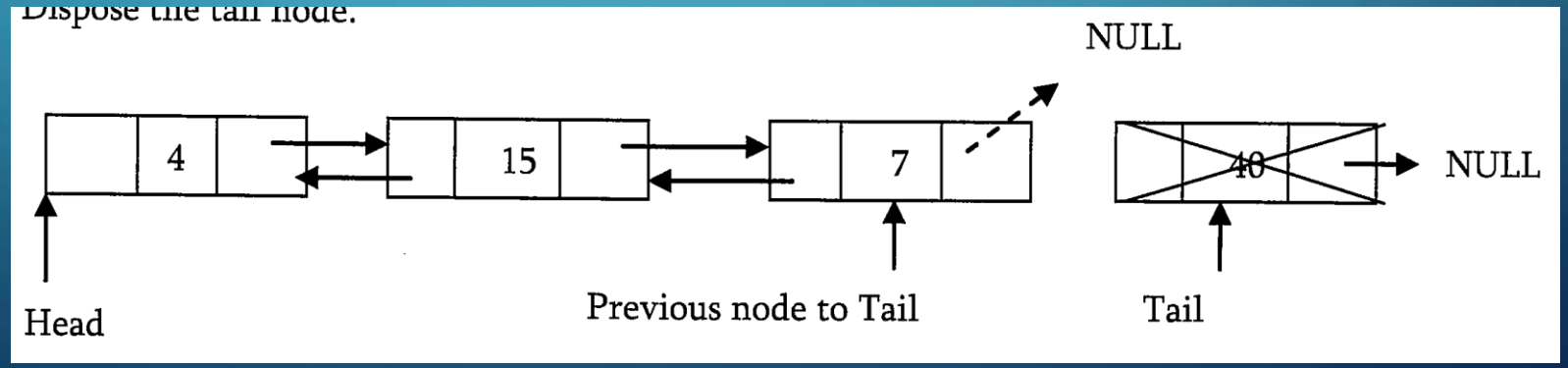
- This operation is a bit trickier, than removing the first node, because algorithm should find a node, which is previous to the tail first. It can be done in three steps :
  - Traverse the list and while traversing maintain the previous node address also. By the time we reach the end of the list, we will have two pointers are pointing to the NULL (tail) and other pointing to the node before tail node.



- Update tail nodes previous nodes next pointer with NULL

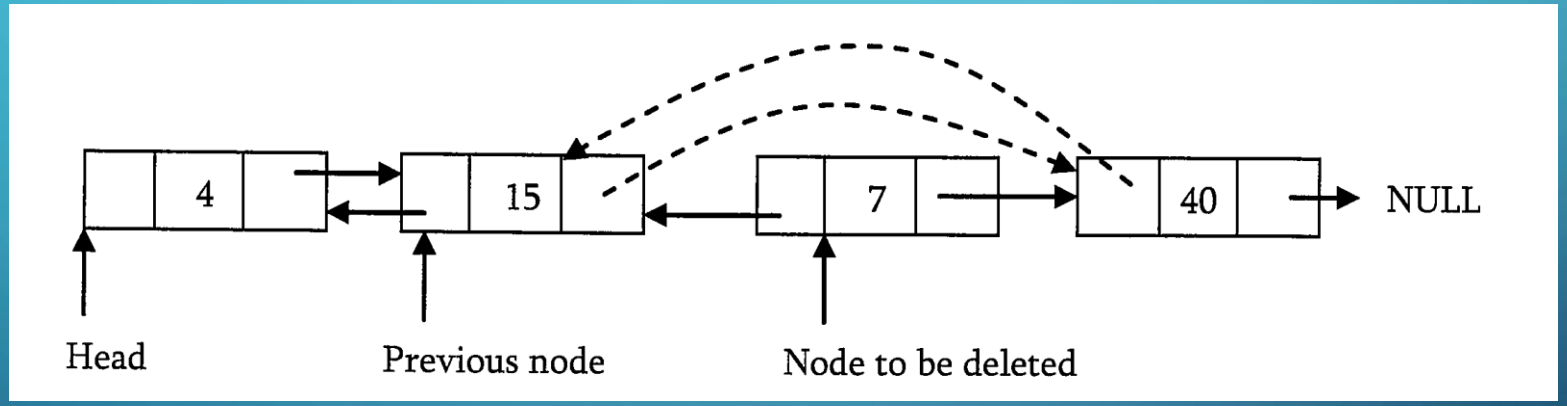


- Dispose the tail node

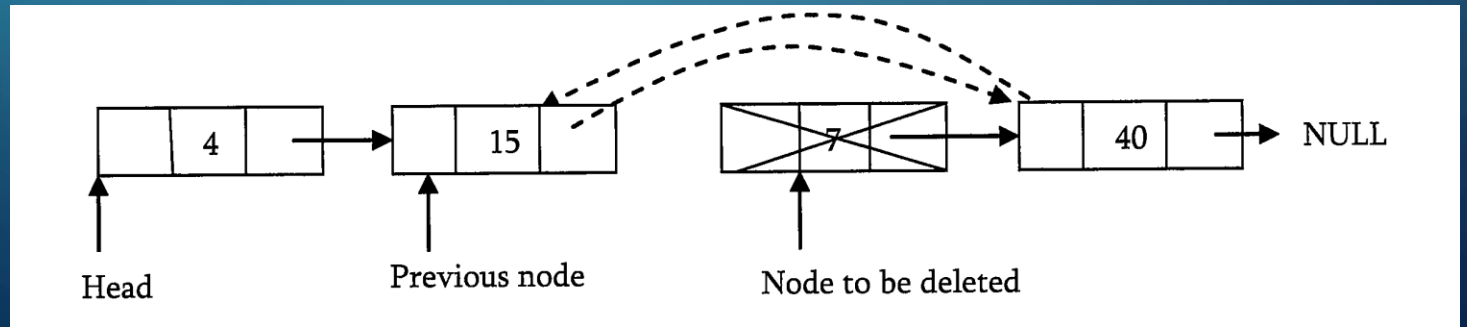


DELETING AN INTERMEDIATE NODE - DOUBLY

- As similar to previous case, maintain previous node also while traversing the list. Once we found the node to be deleted, change the previous nodes next pointer to the next node of the node to be deleted.



- Dispose the current node to be deleted



LINKED LIST IN DETAILS
CIRCULAR LINKED LIST

ROMI FADILLAH RAHMAT

TO BE CONTINUED NEXT WEEK….