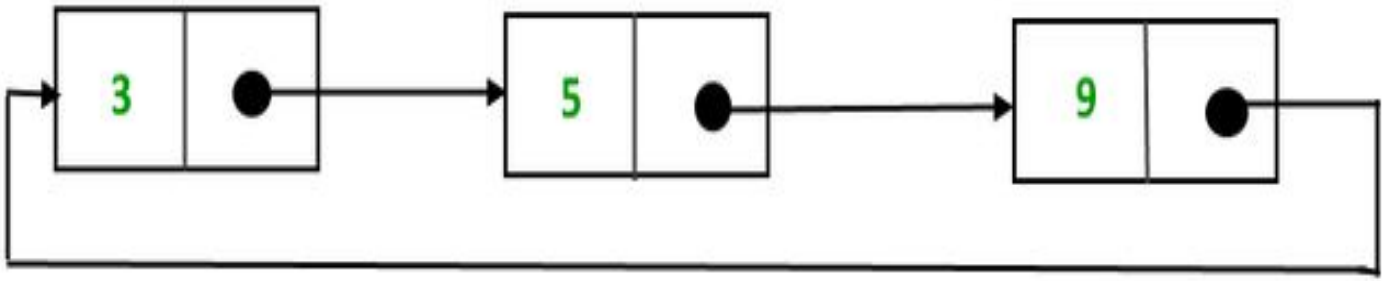
Circular Linked Lists

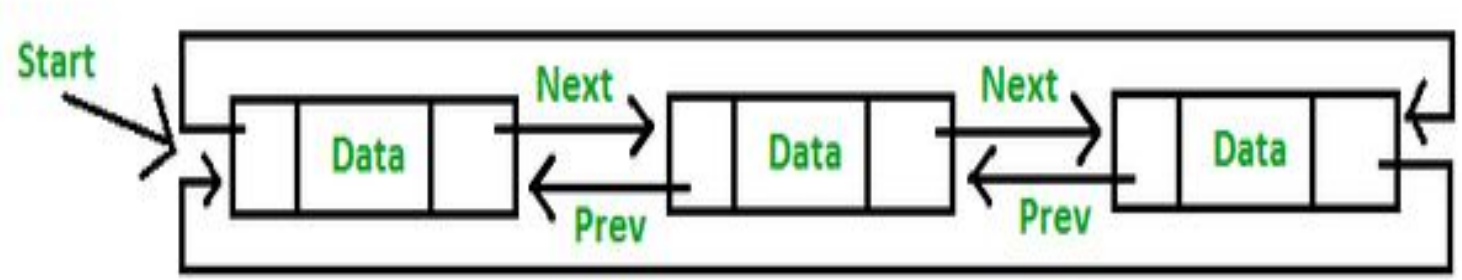
The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.

Two types of circular linked lists:

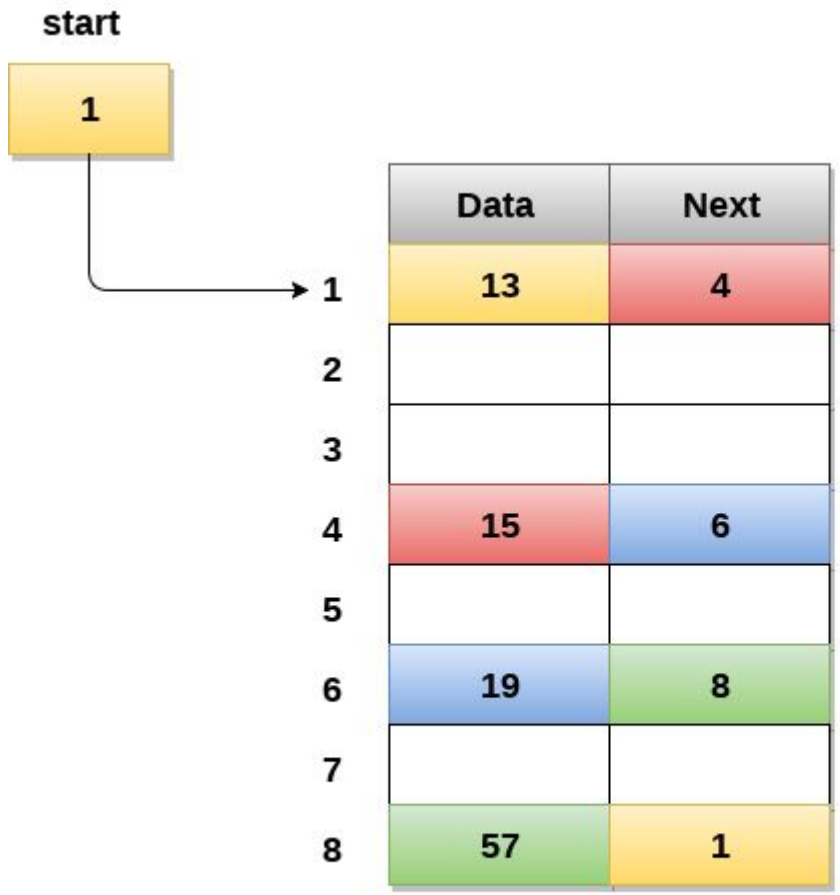
- **Circular singly linked list:** In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We traverse the circular singly linked list until we reach the same node where we started. The circular singly linked list has no beginning or end. No null value is present in the next part of any of the nodes.



- **Circular Doubly linked list:** Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous and next pointer and the last node points to the first node by the next pointer and also the first node points to the last node by the previous pointer.



Memory Representation of circular linked list:



Memory Representation of a circular linked list

Memory Representation of circular linked list:

- In the following image, memory representation of a circular linked list containing marks of a student in 4 subjects.
- However, the image shows a glimpse of how the circular list is being stored in the memory.
- The start or head of the list is pointing to the element with the index 1 and containing 13 marks in the data part and 4 in the next part. Which means that it is linked with the node that is being stored at 4th index of the list.
- However, due to the fact that we are considering circular linked list in the memory therefore the last node of the list contains the address of the first node of the list.

Representation of circular linked list in data structure

- Node representation of a Circular Linked List:

```
public class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
```

Operations on the circular linked list:

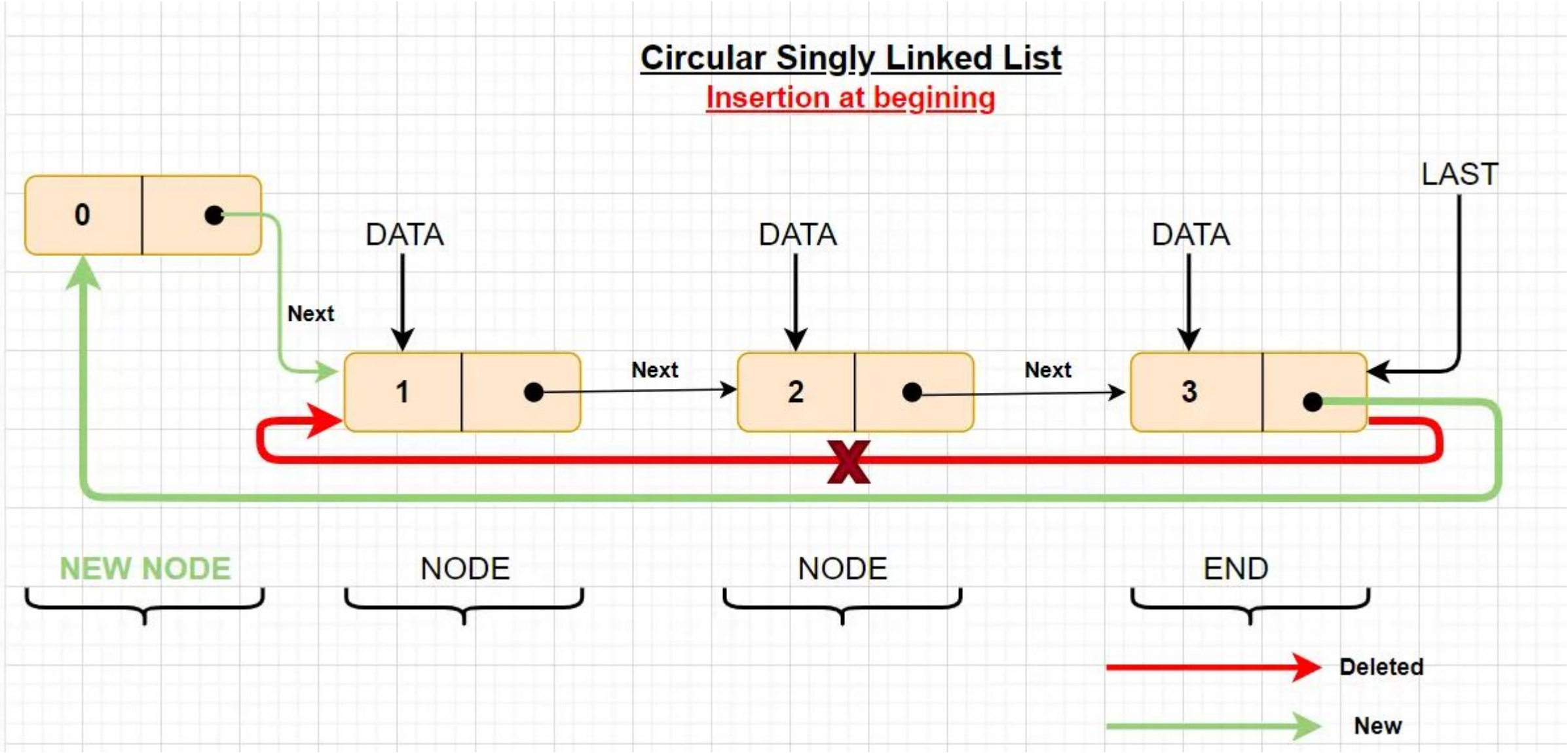
- In a circular linked list, we perform the following operations...
- Insertion
- Deletion
- Display

Insertion

In a circular linked list, the insertion operation can be performed in three ways. They are as follows...

- Inserting At Beginning of the list
- Inserting At End of the list
- Inserting At Specific location in the list

Insert the new node as the first node:



Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the circular linked list...

Step 1 - Create a **newNode** with given value.

Step 2 - Check whether list is **Empty** (**head** == **NULL**)

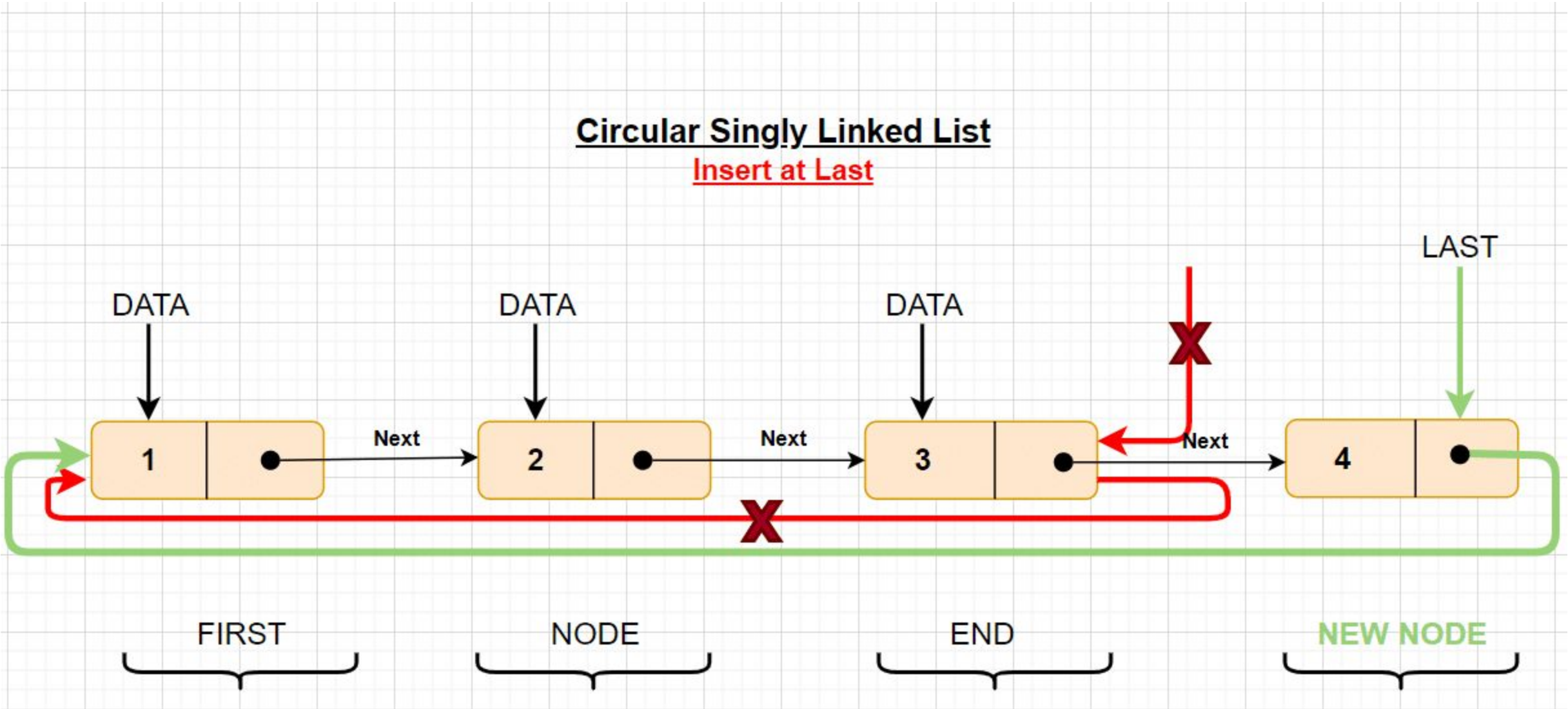
Step 3 - If it is **Empty** then,
set **head** = **newNode** and **newNode→next** = **head** .

Step 4 - If it is **Not Empty** then, define a Node pointer '**temp**' and initialize with '**head**'.

Step 5 - Keep moving the '**temp**' to its next node until it reaches to the last node (until '**temp → next** == **head**').

Step 6 - Set '**newNode → next** =**head**', '**head** = **newNode**' and '**temp → next** = **head**'.

Inserting At End of the list



Inserting At End of the list

Step 1 - Create a **newNode** with given value.

Step 2 - Check whether list is **Empty** (**head** == **NULL**).

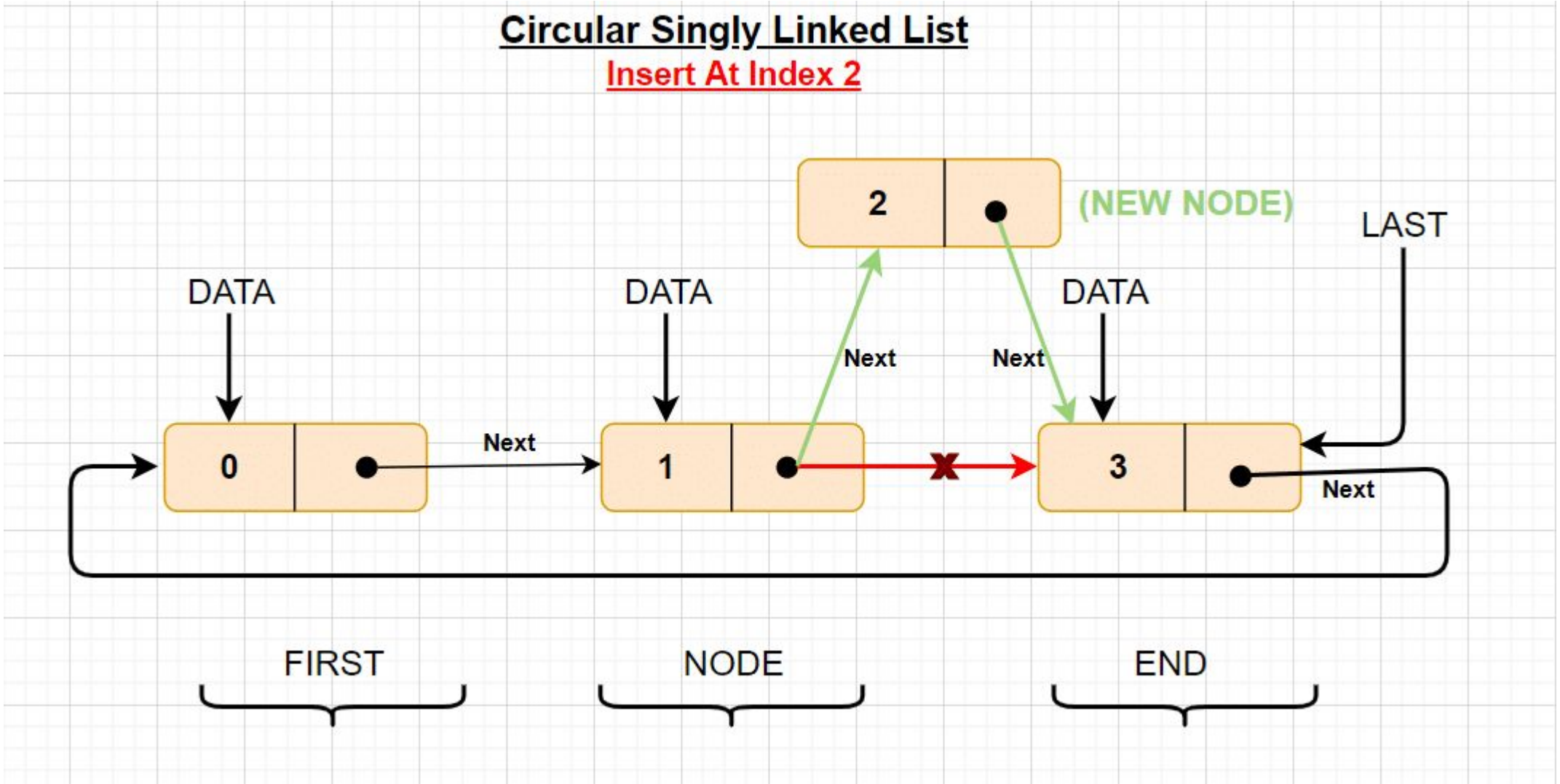
Step 3 - If it is **Empty** then, set **head** = **newNode** and **newNode** → **next** = **head**.

Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.

Step 5 - Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp** → **next** == **head**).

Step 6 - Set **temp** → **next** = **newNode** and **newNode** → **next** = **head**.

Inserting At Specific location in the list (After a Node)



Step 1 - Create a **newNode** with given value.

Step 2 - Check whether list is **Empty** (**head** == **NULL**)

Step 3 - If it is **Empty** then, set **head** = **newNode** and **newNode** → **next** = **head**.

Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.

Step 5 - Keep moving the **temp** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1** → **data** is equal to **location**, here location is the node value after which we want to insert the newNode).

Step 6 - Every time check whether **temp** is reached to the last node or not. If it is reached to last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp** to next node.

Step 7 - If **temp** is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).

Step 8 - If **temp** is last node then set **temp** → **next** = **newNode** and **newNode** → **next** = **head**.

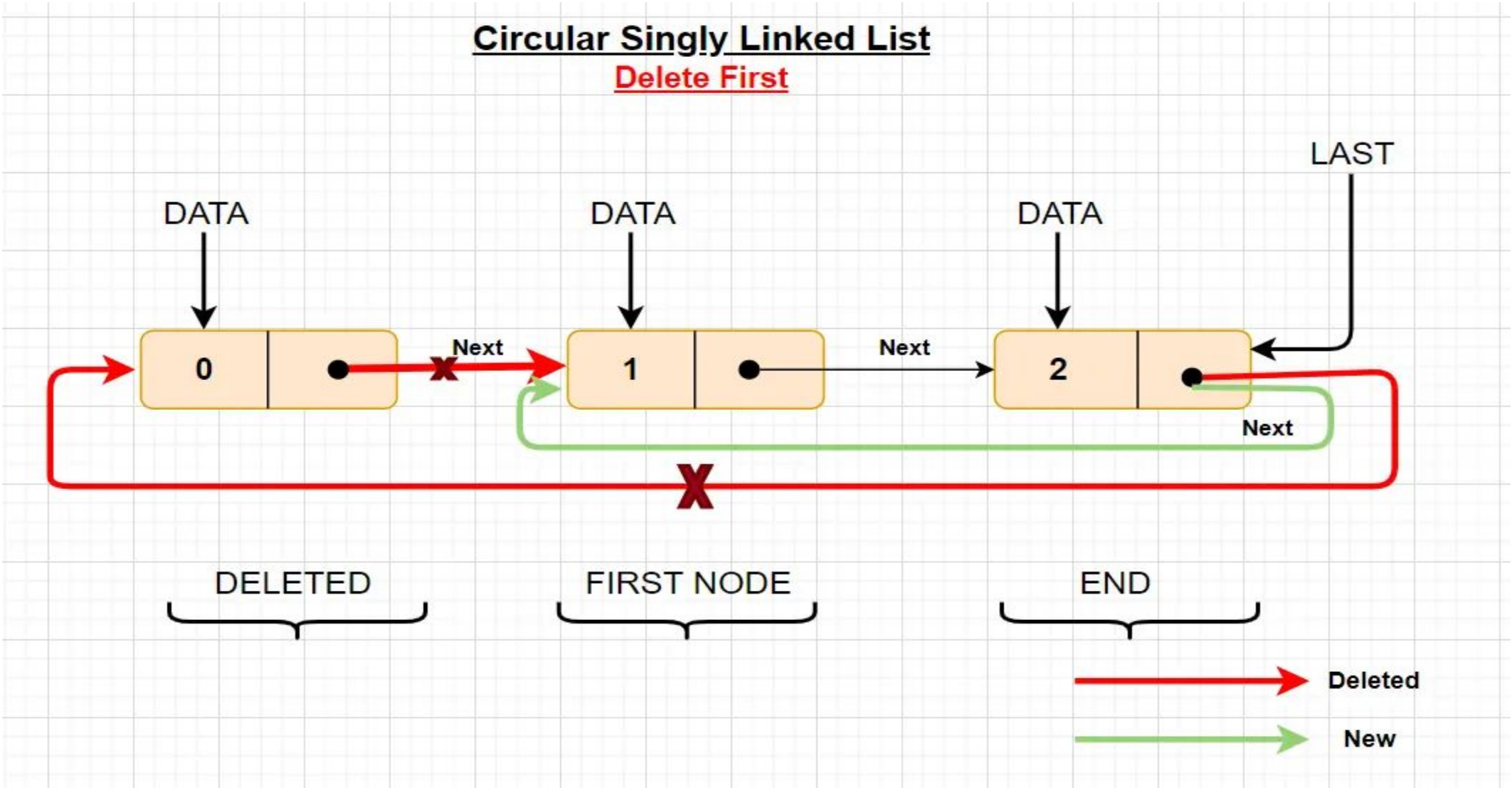
Step 8 - If **temp** is not last node then set **newNode** → **next** = **temp** → **next** and **temp** → **next** = **newNode**.

Deletion

In a circular linked list, the deletion operation can be performed in three ways those are as follows...

- Deleting from Beginning of the list
- Deleting from End of the list
- Deleting a Specific Node

Deleting from Beginning of the list



Deleting from Beginning of the list

Step 1 - Check whether list is **Empty** (**head** == **NULL**)

Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

Step 3 - If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2**' and initialize both '**temp1**' and '**temp2**' with **head**.

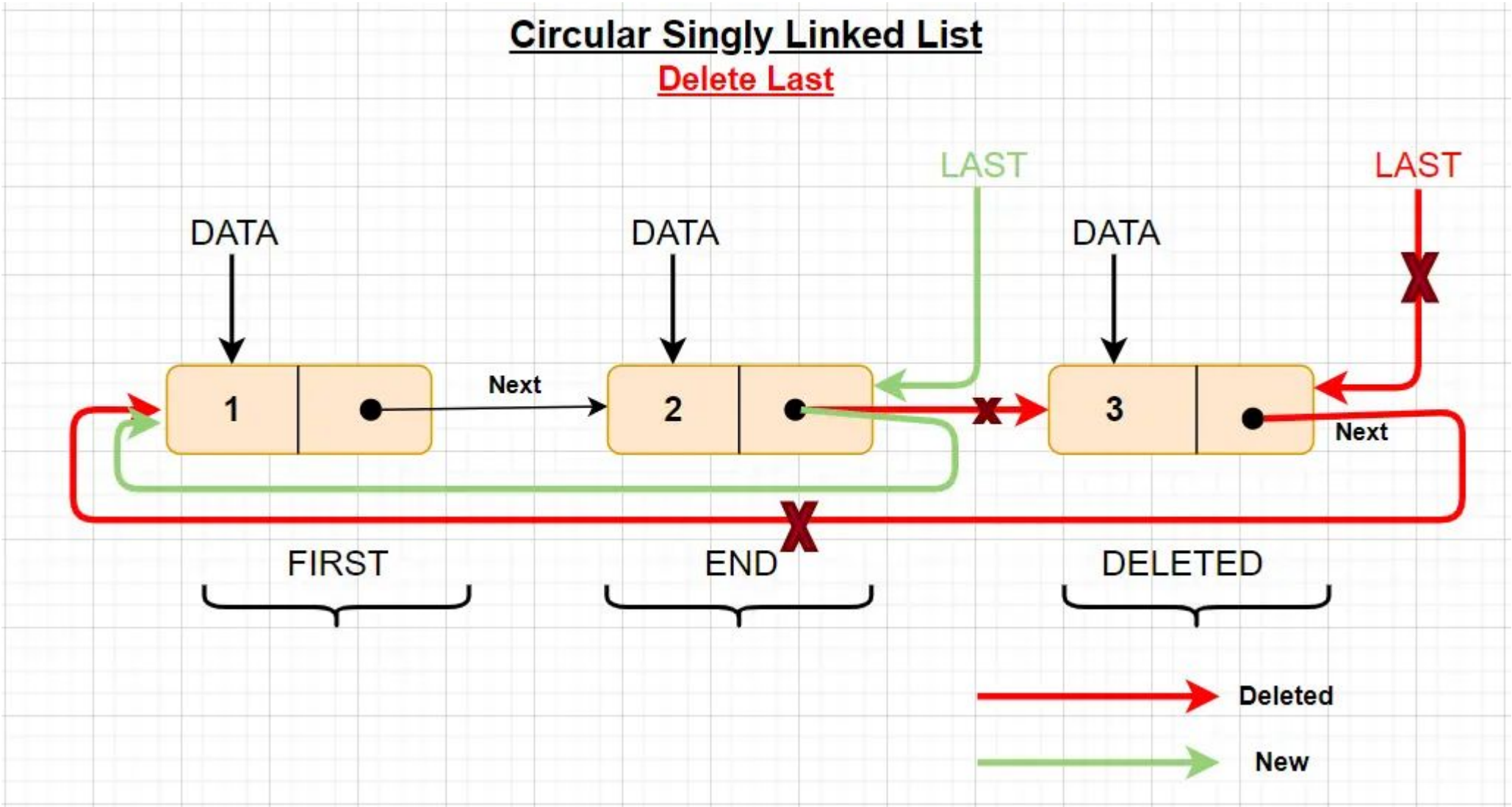
Step 4 - Check whether list is having only one node (**temp1** → **next** == **head**)

Step 5 - If it is **TRUE** then set **head** = **NULL** and delete **temp1** (Setting **Empty** list conditions)

Step 6 - If it is **FALSE** move the **temp1** until it reaches to the last node. (until **temp1** → **next** == **head**)

Step 7 - Then set **head** = **temp2** → **next**, **temp1** → **next** = **head** and delete **temp2**.

Deleting from End of the list



Deleting from End of the list

Step 1 - Check whether list is **Empty** (**head** == **NULL**)

Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

Step 3 - If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2'** and initialize '**temp1**' with **head**.

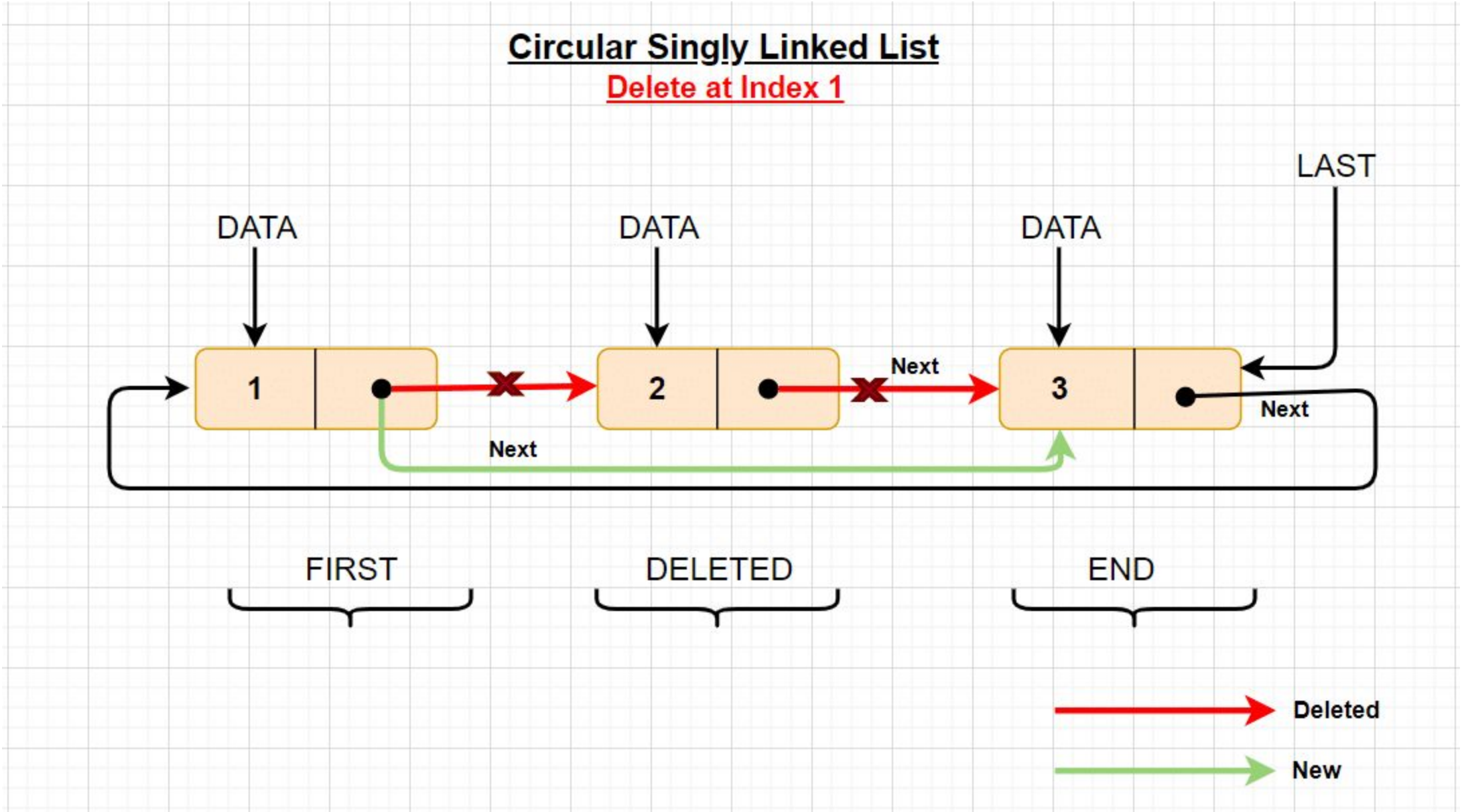
Step 4 - Check whether list has only one Node (**temp1 → next** == **head**)

Step 5 - If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate from the function. (Setting **Empty** list condition)

Step 6 - If it is **FALSE**. Then, set '**temp2 = temp1** ' and move **temp1** to its next node. Repeat the same until **temp1** reaches to the last node in the list. (until **temp1 → next** == **head**)

Step 7 - Set **temp2 → next** = **head** and delete **temp1**.

Deleting a Specific Node from the list



Deleting a Specific Node from the list

Step 1 - Check whether list is **Empty** (**head** == **NULL**)

Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.

Step 3 - If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2**' and initialize '**temp1**' with **head**.

Step 4 - Keep moving the **temp1** until it reaches to the exact node to be deleted or to the last node. And every time set '**temp2 = temp1**' before moving the '**temp1**' to its next node.

Step 5 - If it is reached to the last node then display **'Given node not found in the list! Deletion not possible!!!'**. And terminate the function.

Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node (**temp1 → next == head**)

Step 7 - If list has only one node and that is the node to be deleted then set **head** = **NULL** and delete **temp1** (**free(temp1)**).

Step 8 - If list contains multiple nodes then check whether **temp1** is the first node in the list (**temp1 == head**).

Step 9 - If **temp1** is the first node then set **temp2** = **head** and keep moving **temp2** to its next node until **temp2** reaches to the last node. Then set **head = head → next**, **temp2 → nex**t = **head** and delete **temp1**.

Step 10 - If **temp1** is not first node then check whether it is last node in the list (**temp1 → next == head**).

Step 1 1- If **temp1** is last node then set **temp2 → next** = **head** and delete **temp1** (**free(temp1)**).

Step 12 - If **temp1** is not first node and not last node then set **temp2 → next** = **temp1 → next** and delete **temp1** (**free(temp1)**).

Displaying a circular Linked List

Step 1 - Check whether list is **Empty** (**head** == **NULL**)

Step 2 - If it is **Empty**, then display **'List is Empty!!!'** and terminate the function.

Step 3 - If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.

Step 4 - Keep displaying **temp → data** with an arrow (**--->**)
until **temp** reaches to the last node

Step 5 - Finally display **temp → data** with arrow pointing to **head → data**.