MODULE – III

LINKED LISTS

Introduction to Linked List

- Linked lists and arrays are similar since they both store collections of data.
- Array is the most common data structure used to store collections of elements.
- Arrays are convenient to declare and provide the easy syntax to access any element by its index number.
- Once the array is set up, access to any element is convenient and fast.

Disadvantages of Array

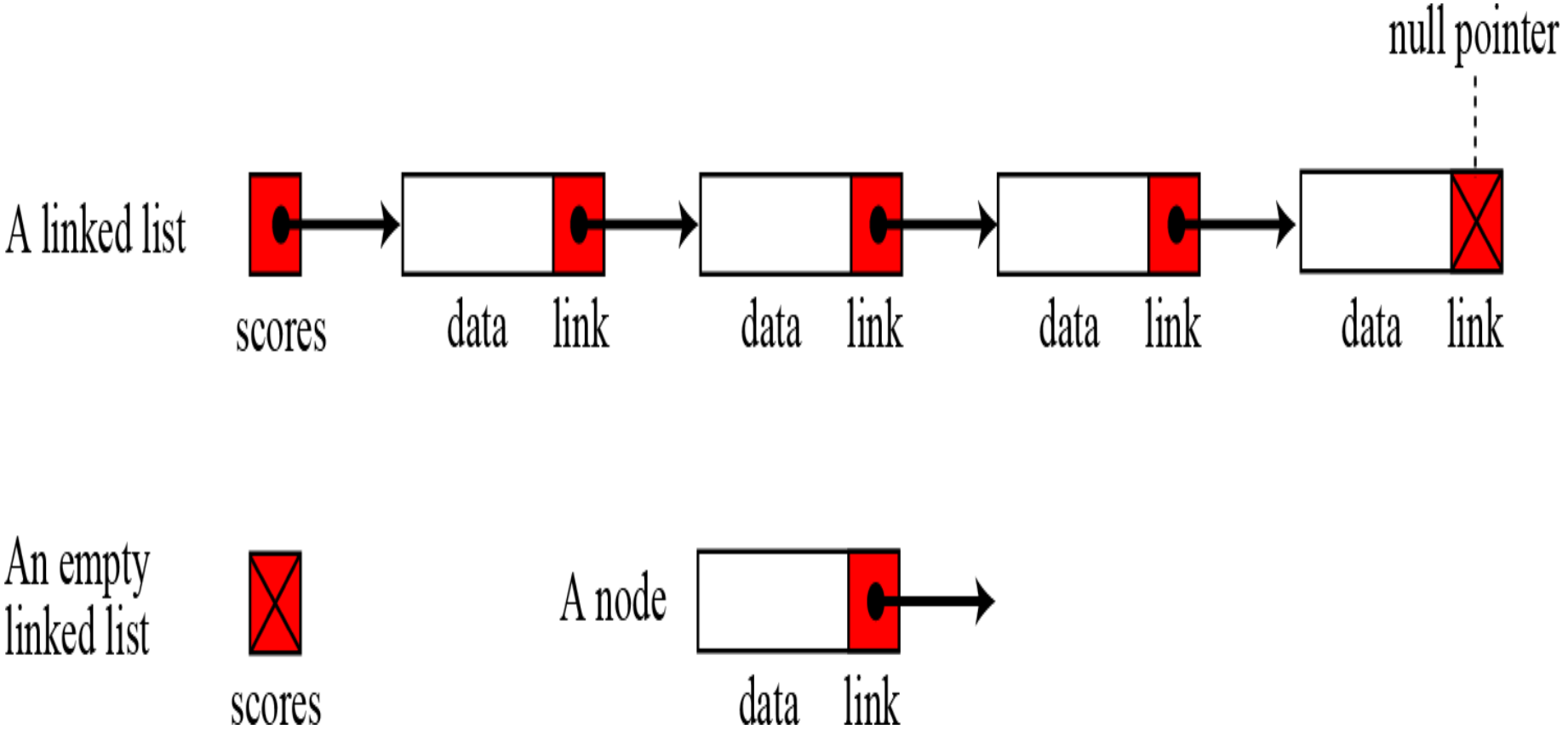
The disadvantages of arrays are:

- The size of the array is fixed. Most often this size is specified at compile time. This makes the programmers to allocate arrays, which seems "large enough" than required.
- Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.
- Deleting an element from an array is not possible. Linked lists have their own strengths and weaknesses, but they happen to be strong where arrays are weak.
- Generally array's allocates the memory for all its elements in one block whereas linked lists use an entirely different strategy. Linked lists allocate memory for each element separately and only when necessary.

Introduction to Linked List

- A linked list is a non-sequential collection of data items. It is a dynamic data structure.
- Lists are the most commonly used non-primitive data structures.
- An element of list must contain at least two fields, one for storing data or information and other for storing address of next element.
- Technically each element is referred to as a node, therefore a list can be defined as a collection of nodes.
- The data items in the linked list are not in consecutive memory locations. They may be anywhere, but the accessing of these data items is easier as each data item contains the address of the next data item.

Introduction to Linked List



Advantages of Linked List

Linked lists have many advantages. Some of the very important advantages are:

- Linked lists are dynamic data structures. i.e., they can grow or shrink during the execution of a program.
- Linked lists have efficient memory utilization. Here, memory is not preallocated. Memory is allocated whenever it is required and it is de-allocated (removed) when it is no longer needed.
- Insertion and Deletions are easier and efficient. Linked lists provide flexibility in inserting a data item at a specified position and deletion of the data item from the given position.
- Many complex applications can be easily carried out with linked lists.

Disadvantages of Linked List

- It consumes more space because every node requires a additional pointer to store address of the next node.
- Searching a particular element in list is difficult and also time consuming.

Array Vs Linked List

ARRAY	LINKED LIST
Size of an array is fixed	Size of a list is not fixed
Memory is allocated from stack	Memory is allocated from heap
It is necessary to specify the number of elements during declaration (i.e., during compile time).	It is not necessary to specify the number of elements during declaration (i.e., memory is allocated during run time).
It occupies less memory than a linked list for the same number of elements.	It occupies more memory.
Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.	Inserting a new element at any position can be carried out easily.
Deleting an element from an array is not possible.	Deleting an element is possible.

Types of Linked Lists

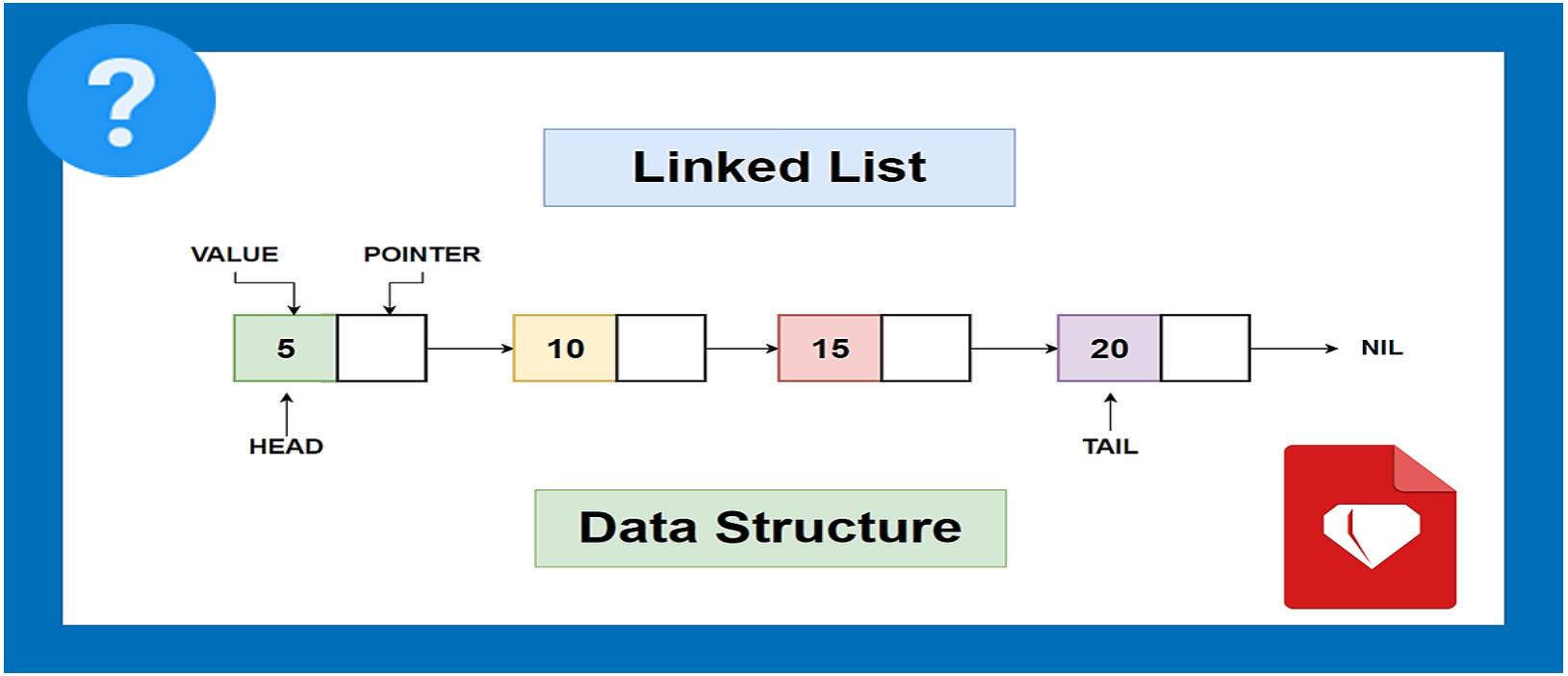
Basically we can put linked lists into the following four items:

1. Single Linked List.
2. Double Linked List.
3. Circular Linked List.
4. Circular Double Linked List.

Types of Linked Lists

Single Linked List

A single linked list is one in which all nodes are linked together in some sequential manner. Hence, it is also called as linear linked list.



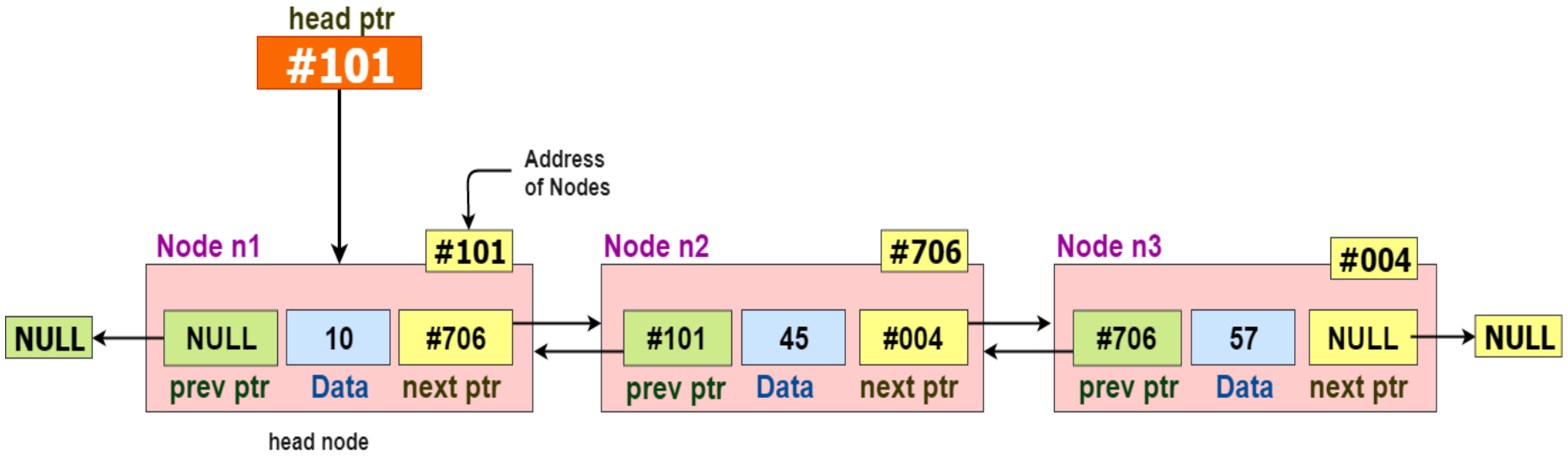
Types of Linked Lists

Double Linked List

- A double linked list is one in which all nodes are linked together by multiple links which helps in accessing both the successor node (next node) and predecessor node (previous node) from any arbitrary node within the list.
- Therefore each node in a double linked list has two link fields (pointers) to point to the left node (previous) and the right node (next).
- This helps to traverse in forward direction and backward direction.

Types of Linked Lists

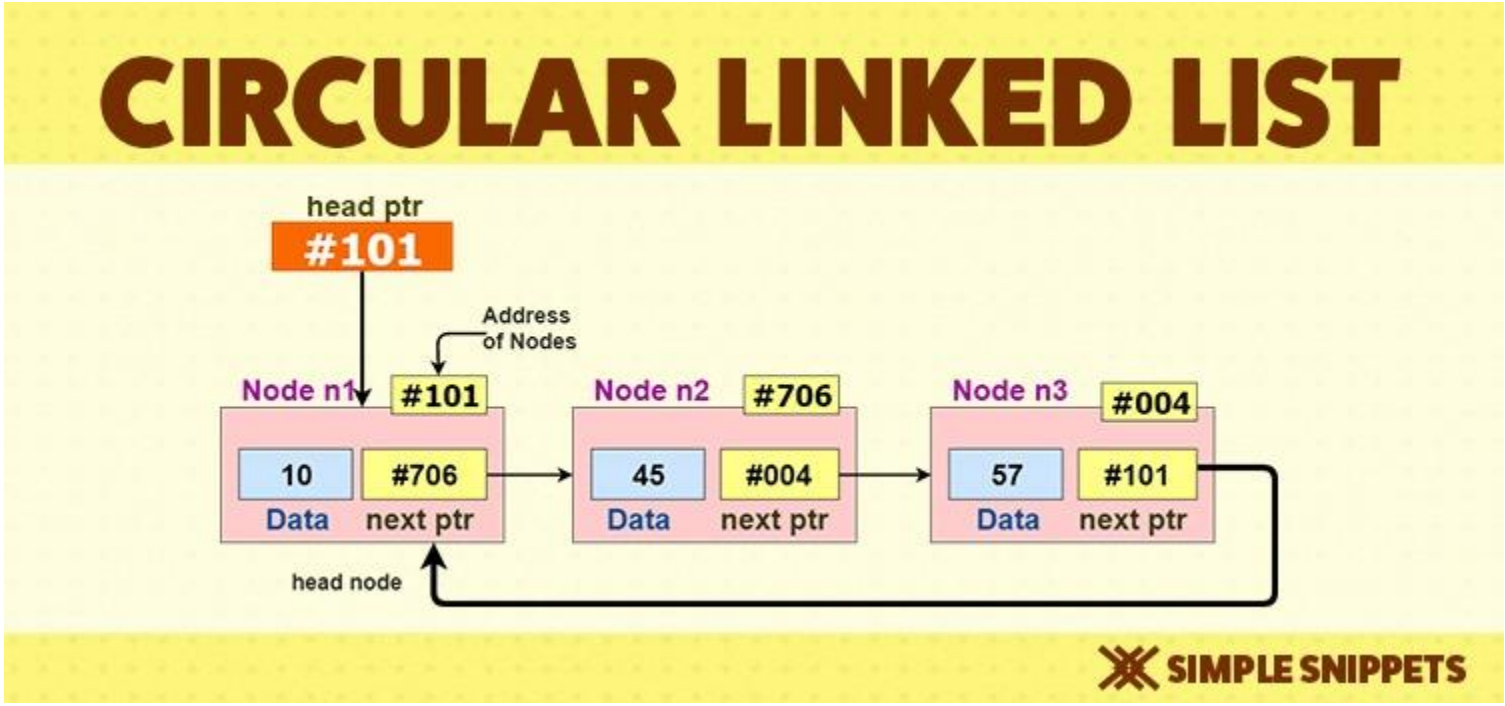
Double Linked List



Types of Linked Lists

Circular Linked List

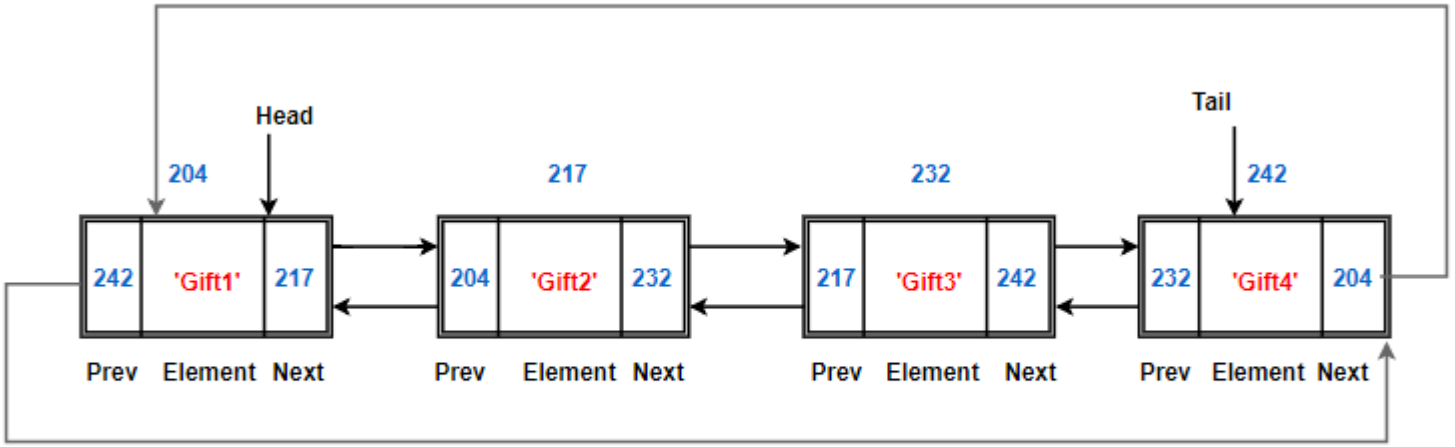
A circular linked list is one, which has no beginning and no end. A single linked list can be made a circular linked list by simply storing address of the very first node in the link field of the last node.



Types of Linked Lists

Circular Double Linked List

A circular double linked list is one, which has both the successor pointer and predecessor pointer in the circular manner.



Example of Doubly Linked List With Circular Structure

Applications of Linked List

- Linked lists are used to represent and manipulate polynomial. Polynomials are expression containing terms with non zero coefficient and exponents. For example:
- $P(x) = a_0 X^n + a_1 X^{n-1} + \ldots\ldots + a_{n-1} X + a_n$
- Represent very large numbers and operations of the large number such as addition, multiplication and division.
- Linked lists are to implement stack, queue, trees and graphs.
- Implement the symbol table in compiler construction.

Key Terms

A linked list is a non-sequential collection of data items called nodes. These nodes in principle are structures/class containing fields. Each node in a linked list has basically two fields.

1. Data field
2. Link field

- The data field contain an actual value to be stored and processed.
- The link field contains the address of the next data item in the linked list

Null pointer

The link field of the last node contain NULL(in c program) None(in python) rather than a valid address. It indicates end of the list.

Key Terms

External pointer:

It is a pointer to the very first node in the linked list, it enables us to access the entire linked list.

Empty list:

If the nodes are not present in a lined list, then it is called an empty linked list also called null list.

A linked list can be made an empty list by assigning a NULL value to the external pointer.

start= NULL(None)

Operations on Linked List

1. Creation
2. Insertion
   - At the beginning of a linked list
   - At the end of a linked list
   - At the specified position in a linked list
3. Deletion
   - Beginning of a linked list
   - End of a linked list
   - Specified position of a linked list
4. Traversing
5. Searching
6. Concatenation
7. Display

Operations on Linked List

- **Creation:**

This operation is used to create a linked list. Here, the node is created as and when it is required and linked to the list to preserve the integrity of the list.

- **Insertion:**

This operation is used to insert a new node in the linked list at the specified position. A new node may be inserted

1. At the beginning of a linked list
2. At the end of a linked list
3. At the specified position in a linked list
4. If the list itself empty, then new node is inserted as a first node.

Operations on Linked List

- **Deletion:**

This operation is used to delete a node from the linked list. A node may be deleted from the

1. Beginning of a linked list
2. End of a linked list
3. Specified position of a linked list.

- **Traversing:**

  - It is processing of going through all the nodes of a linked list from one end to the other end.
  - If we start traversing from the very first node towards the last node, it is called forward traversing.
  - If the desired element is found, signal operation is SUCCESSFUL otherwise, signal as UNSUCCESSFULL.

Operations on Linked List

- **Concatenation:**

It is process of appending the second list to the end of first list consisting of m nodes. When we concatenate two lists, the second list has n nodes, then the concatenated list will be having (m+n) nodes.

- **Display:**

This operation is used to print each and every node's information. We access each node from the beginning (or specified position) of the last node.

Single Linked List

- A singly linked list is one in which all nodes are linked together in some sequential manner. Hence, it is also called linear linked list. It has the beginning and the end.
- A singly linked list is a dynamic data structure. It may grow or shrinking depends on the operations made.
- In python, a linked list is created using class. When we create an object memory to be allocated.
- We consider start(or head)as an external pointer. This helps in creating and accessing other nodes in the linked list.

Single Linked List

Consider the following class to create a node and linked list and start/head creation

```
class node:
    def __init__(self,data):
        self.data=data
        self.link=None

class singlelist:
    def __init__(self):
        self.head=None
```

Node

head

Single Linked List

When we create an object to a node class , a block of memory is allocated to node.

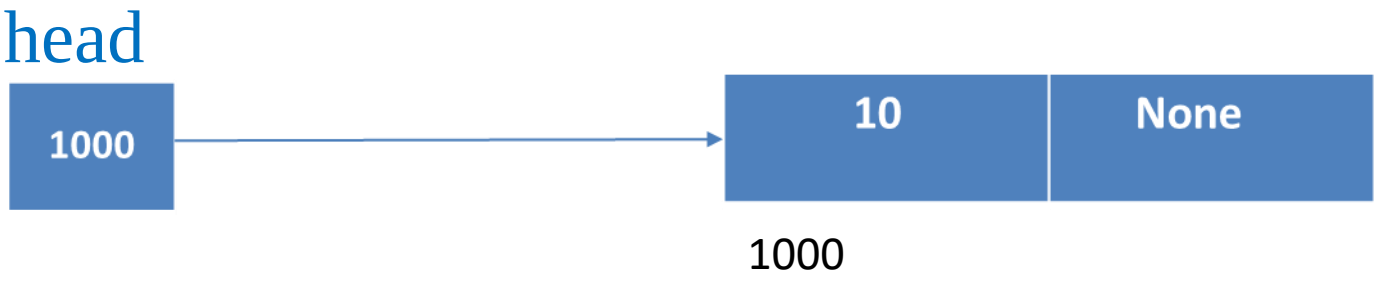
new_node=node(10)

When the statement

head = new_node

is executed the address of the new node is stored in head node.

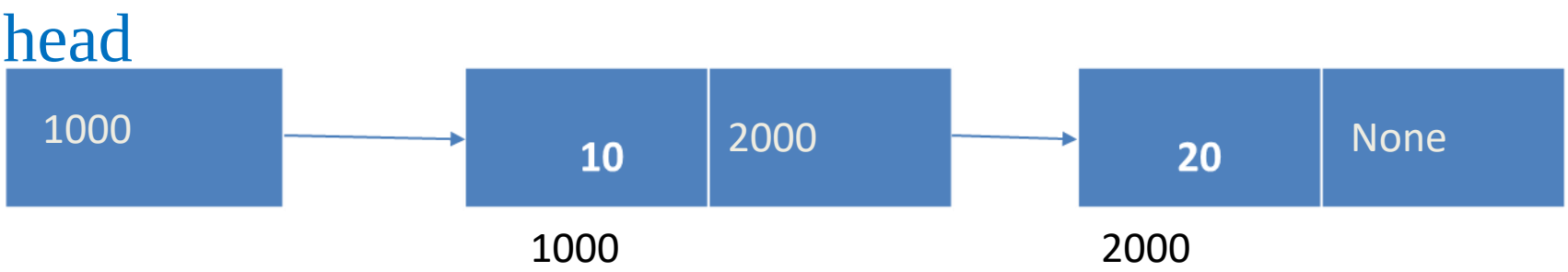
This activity can be pictorially shown as



Single Linked List

Any number of nodes can be created and linked to the existing node. Suppose we want to add another node to the above list, the following statements are required.

New_node =node(20)
self.head.link=new_node



Inserting a node at the beginning

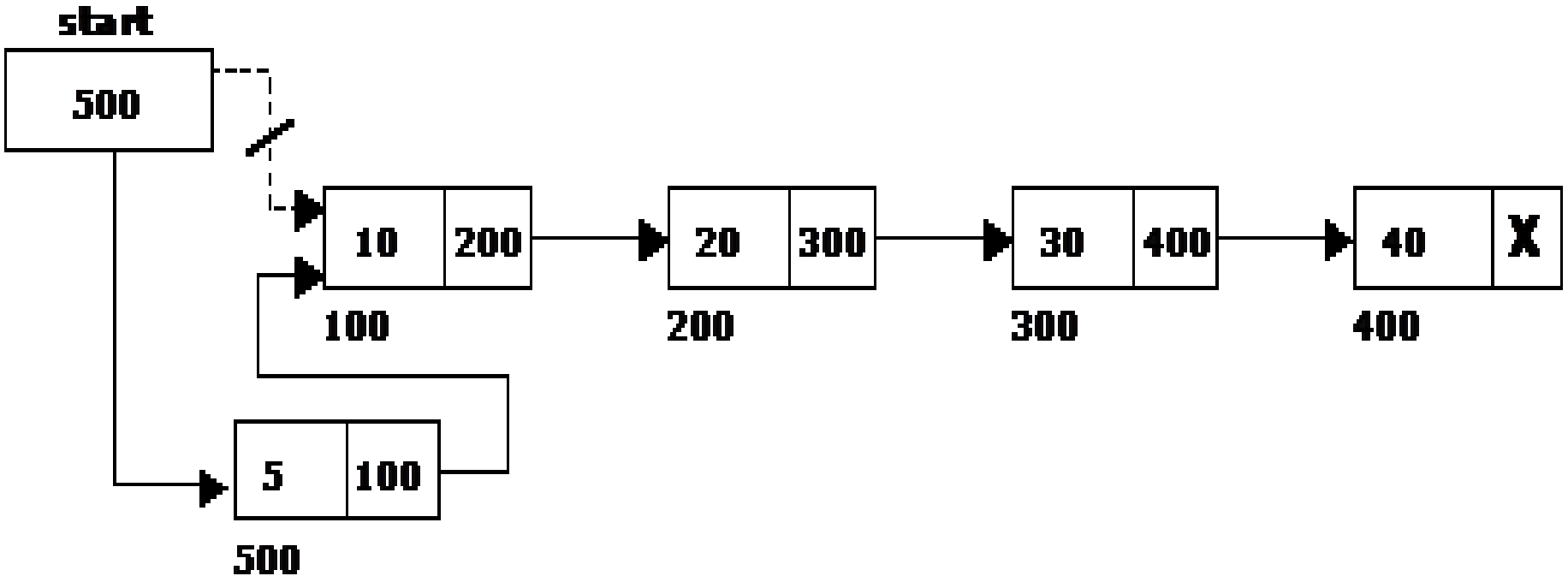
If the linked list is empty or the node to be inserted appears before the starting node then insert that node at the beginning of the linked list (i.e.,as the starting node)

Algorithm :

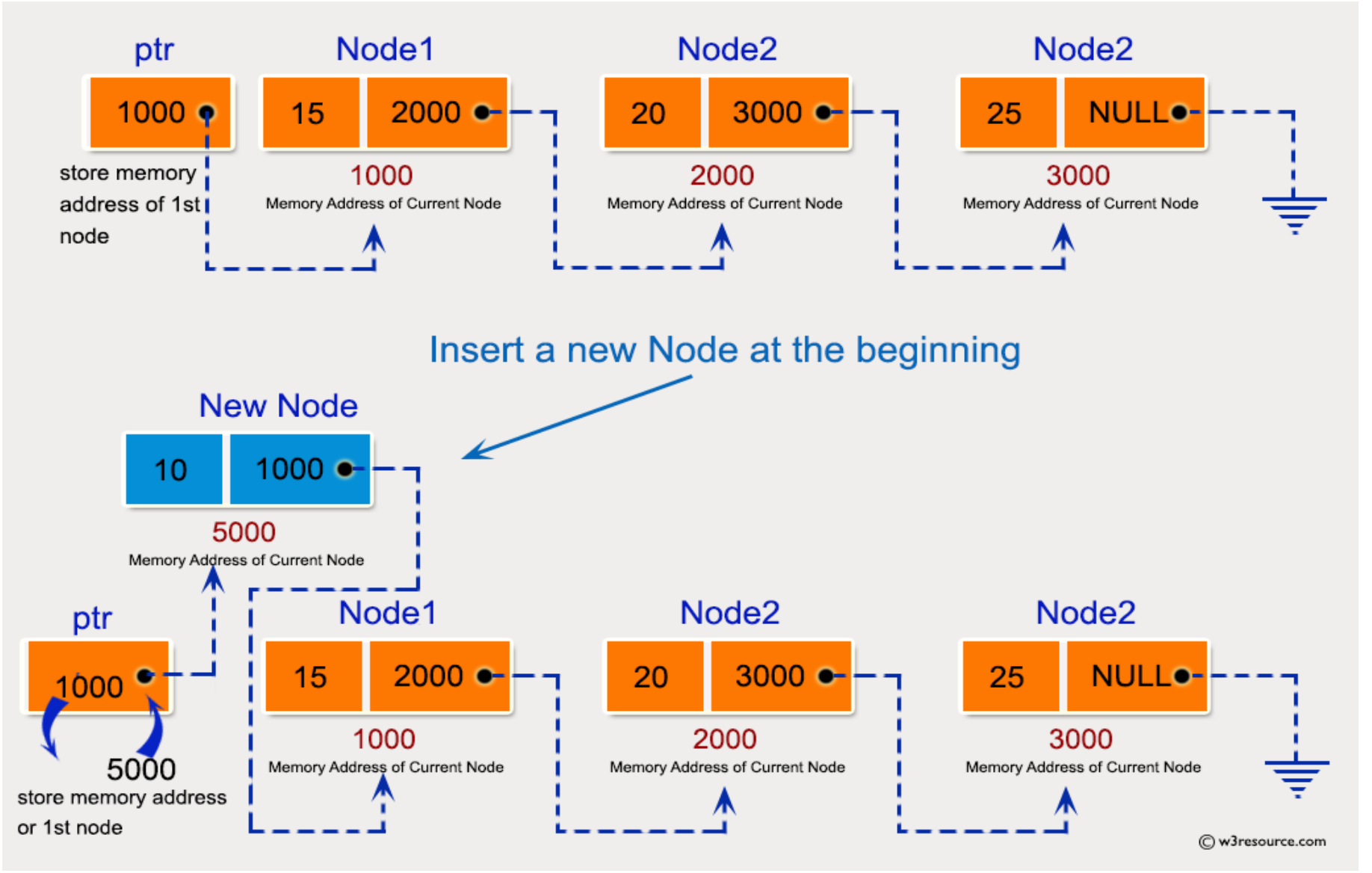
1. if head = None, then
   new_node = node(data)
   self.head = new_node
   else
   create object of node
2. nb = Node(data)
3. nb.next = self.head
4. self.head = nb

Function to Insert a node at the beginning

```
def insert_begin(self,data):
        nb = Node(data)
        nb.next = self.head
        self.head = nb
```



Insert a node at the beginning of Linked List



Inserting a node at the end

- If the node to be inserted after the last node in the linked list then insert that node at the end of the linked list.
- Create a new node, Put in the data, Set next as None
- new_node = Node(new_data)

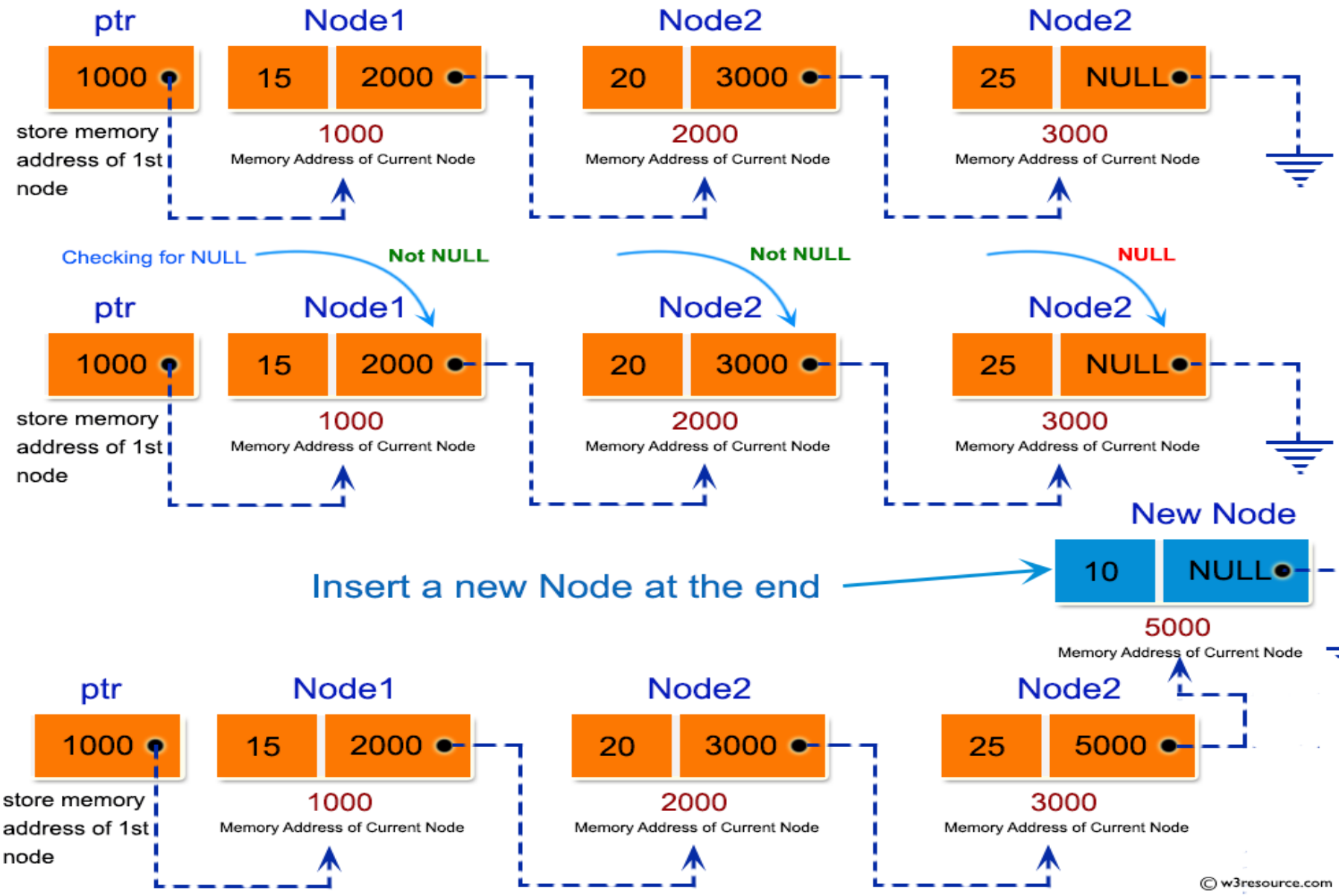
Algorithm to Insert a node at the end

```
# If the Linked List is empty, then make the  new node
    as head
new_node = node(data)
if self.head is None:
    self.head = new_node  return
#Else traverse till the last node
temp = self.head
while temp.next is not None:
temp = temp.next
# Change the next of last node  temp.next =new_node
```

Function to Insert a node at the end

```
def insert_end(self,data):
        new_node=node(data)
        if self.head is None:
            self.start=new_node
            return
        temp=self.head
        while temp.next is not None:
            temp=temp.next
        temp.next=new_node
```

Function to Insert a node at the end



Algorithm to Insert a node at given position

Algorithm inserts an item at the specified position in the linked list.

Step 1.if position is equal to 1

create a node and insert data in to the node

return

Step 2. pos = p

temp = sel.head

for i in range(pos-1):

move the pointer to the next node i.e., temp = temp.next

if pointer is None:

print position is not present in the list

else:

create a node i.e., np =node(data)

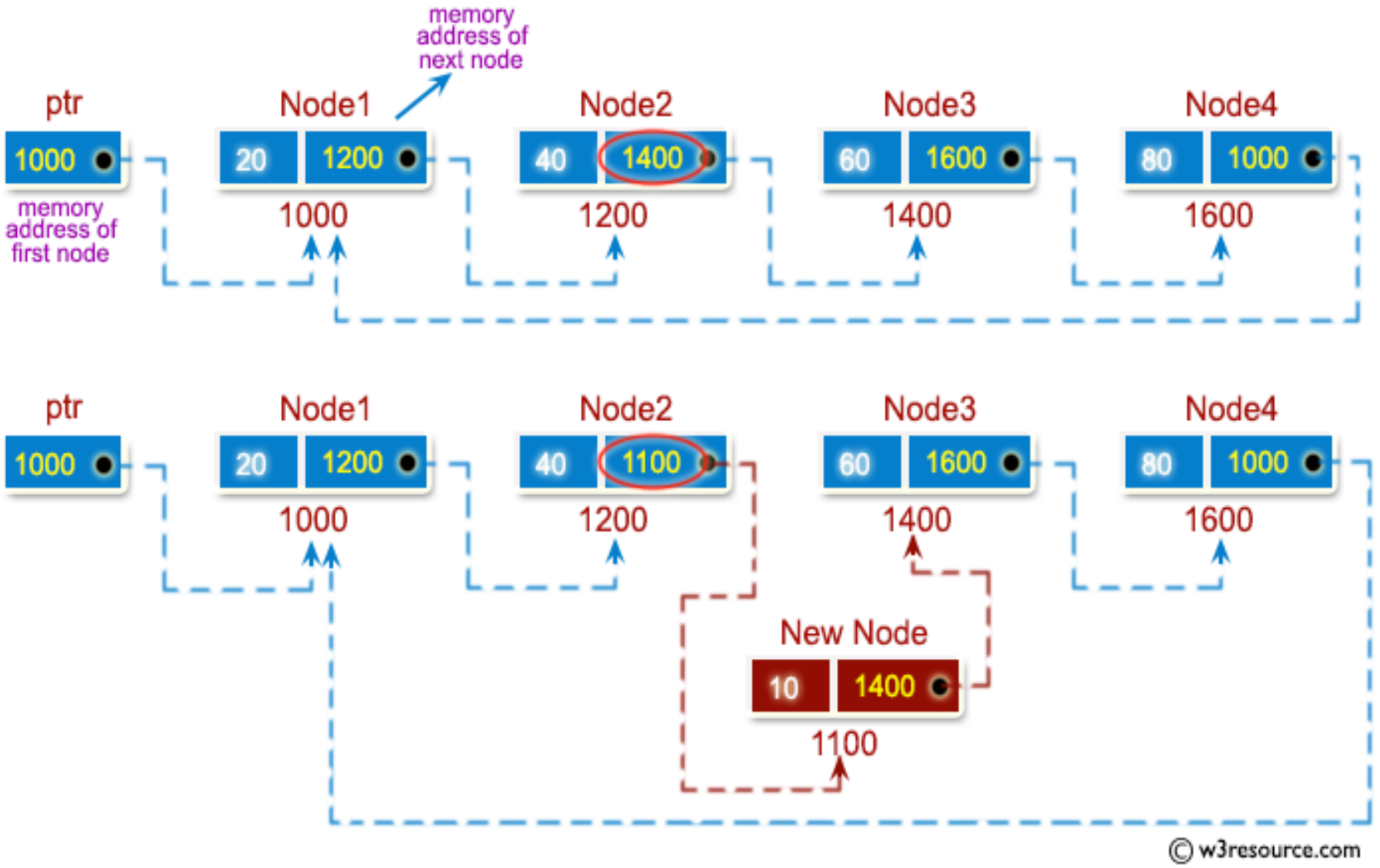
np.next = temp.next

temp.next= np

Function to Insert a node at given position

```
def insert_pos(self,pos,data):
        np = Node(data)
        temp = self.head
       if post = 0:
            np.next = self.head
            self.head = np
            return
       else:
           for i in range(pos-1):
                    temp = temp.next
                    if temp.next is None:
                            print("Position not availble in the list")
                    else:
                            np.next = temp.next
                            temp.next = np
```

Insert a node at given position



Deletion of a node

Deleting a node from the linked list has the following three instances.

1. Deleting the first node of the linked list .
2. Deleting the last node of the linked list.
3. Deleting the specified node within the linked list.

Deleting a node at the beginning

- If the linked list is empty, then display the message deletion is not possible.
- If the node to be deleted is the first node then set the start node to point to the second node in the linked list.

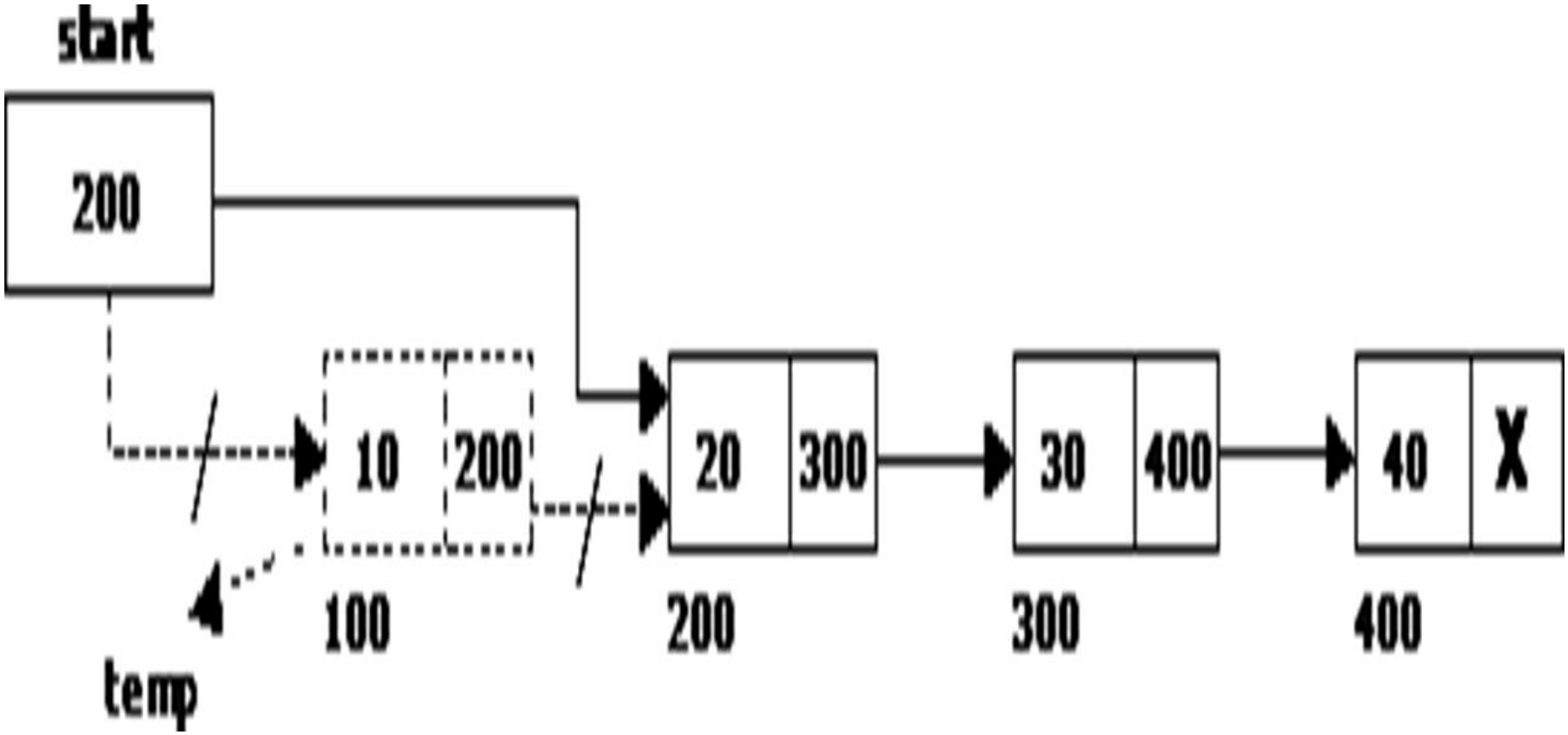
Algorithm to delete a node at the beginning

Step 1. if self.head = None
print linked list is empty
return
Step 2. else:
Step 3. set temp = self.head
Step 4. set self.head = temp.next
Step 5. set temp.next = None

Function to delete a node at the beginning

```
def delete_beg(self):
        if self.head is None:
            print("List is empty")
        else:
            temp = self.head
            self.head = temp.next
            temp.next = None
```

Delete a node at the beginning



Algorithm to delete a node at the end of list

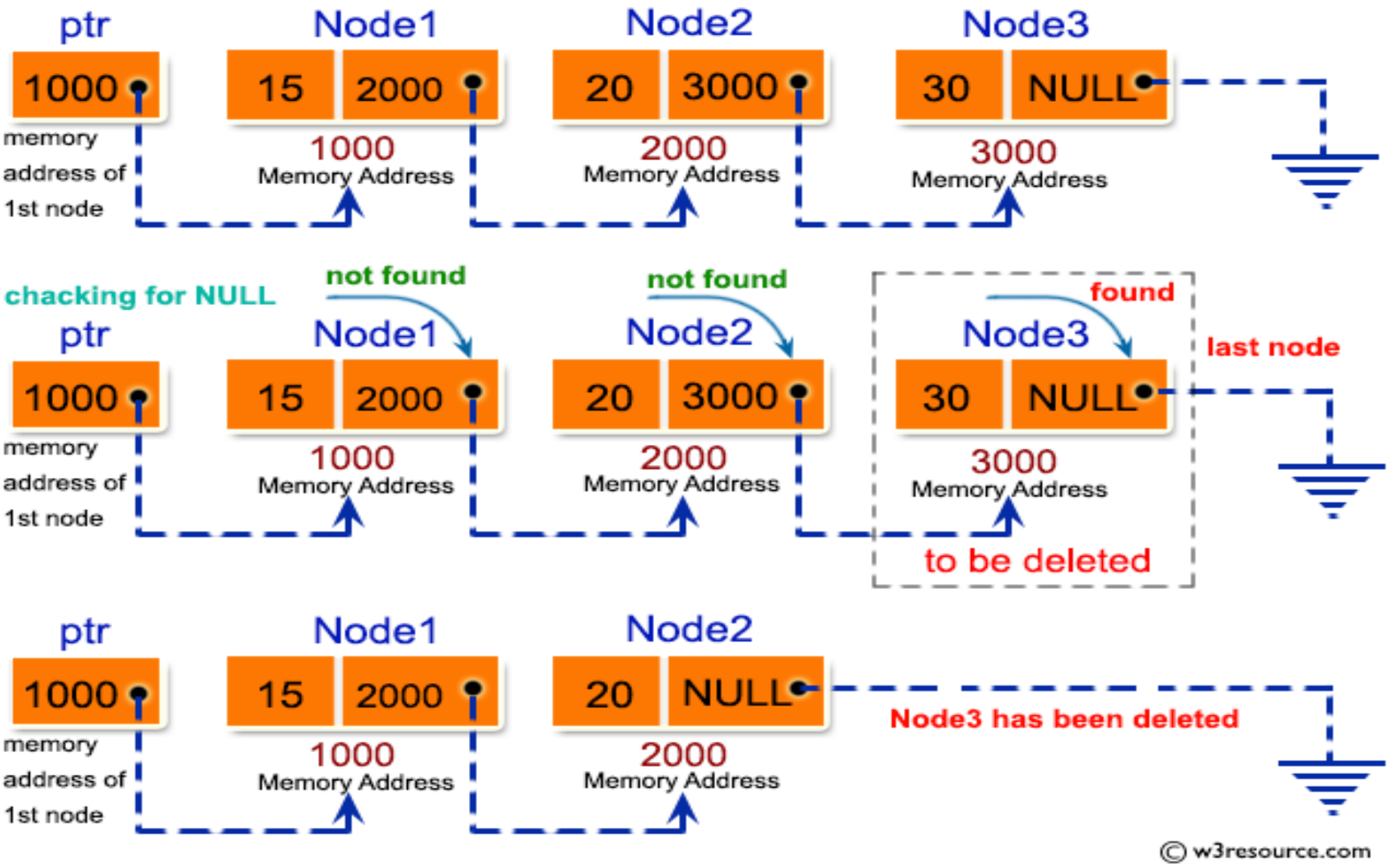
- The following steps are followed to delete a node at the end of the list:
  - If list is empty then display _Empty List‘message.
  - If the list is not empty, follow the steps given below:

Step 1: temp = self.head.next
Step 2: prev = self.head
Step 3: while temp.next is not None:
        temp = temp.next
        prev = prev.next
Step 4: prev.next = None

Function to delete a node at the end of list

```
def delete_end(self):
    if self.head is None:
                print("List is empty")
        else:
            prev = self.head
            temp = self.head.next
            while temp.next is not None:
                temp = temp.next
                prev = prev.next
            prev.next = None
```

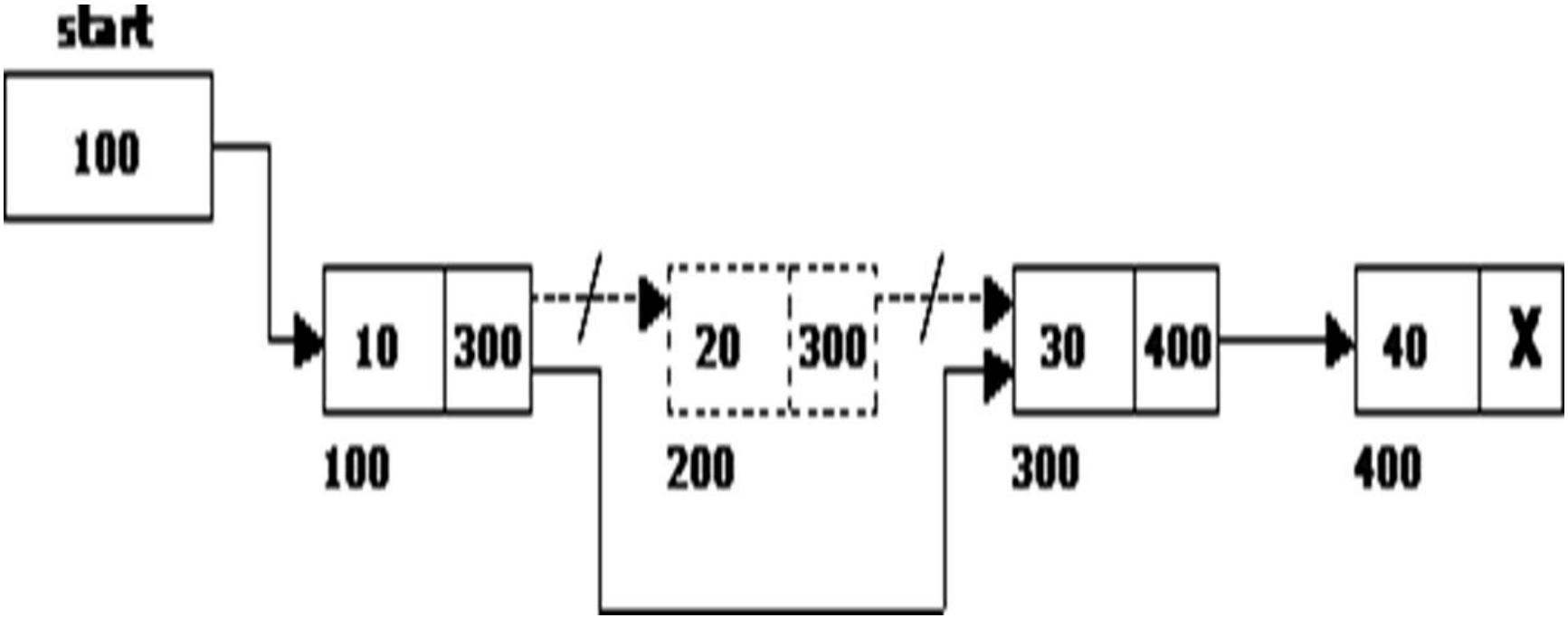
Delete a node at the end of list



Function to delete a node at given position

```
def delete_pos(self,pos):
        if self.head is None:
            print("List is empty")
        else:
            prev = self.head
            temp = self.head.next
            for i in range(1,pos-1):
                temp = temp.next
                prev = prev.next
            prev.next = temp.next
            temp.next = None
```

Delete a node at given position



Delete a node at given position