- DATASTRUCTURE

DATRUCTURE IS THE WAY OF ORGANIZING ALL DATA ITEMS IN ORDER THAT NOT ONLY ELEMENTS TO BE STORED BUT ALSO THE RELATION BETWEEN THE ELEMENTS

INTRODUCTION

- **Data structure**:-A data structure is a logical representation of data and operation that can be performed on the data.

  1)linear data structure

  2)Non linear data structure

- Linear data structure is an order of data elements. They are arrays, stacks, queues, and linked lists.

Linked list :- linked list is a linear data structure. It contains nodes. Each node contains two parts, i.e. DATA part and LINK part.

- The data contains elements and
- Link contains address of another node.

LIMITATIONS OF ARRAYS

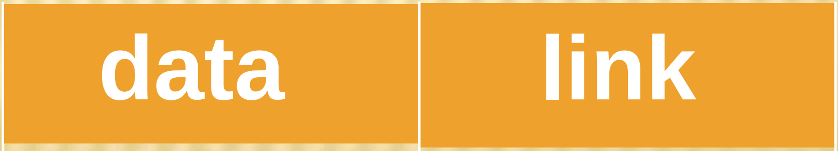
- Arrays are simple to understand and elements of an array are easily accessible
- But arrays have some limitations.
- Arrays have a fixed dimension.
- Once the size of an array is decided it can not be increased or decreased during education.

- Array elements are always stored in contiguous memory locations.
- Operations like insertion or deletion of the array are pretty tedious.
- To over come this limitations we use linked list.

LINKED LISTS

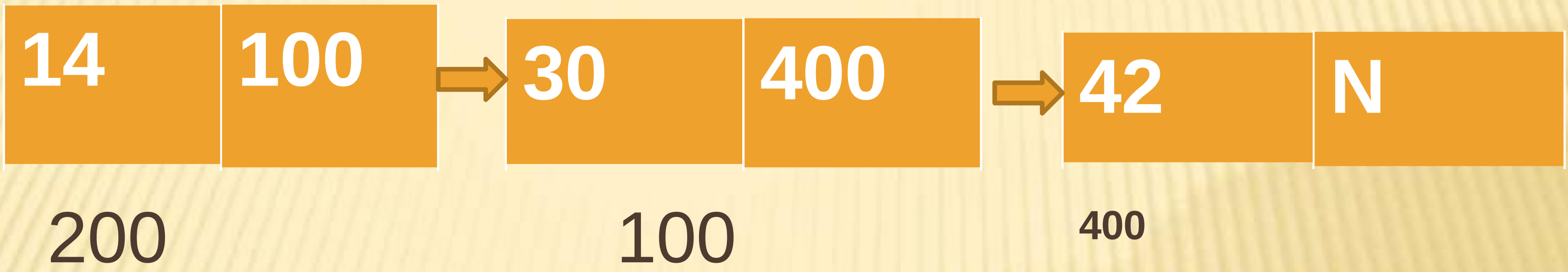
- Linked list is a collection of elements called nodes.
- Each node contains two parts. they are data part and link part.

Node



data

link



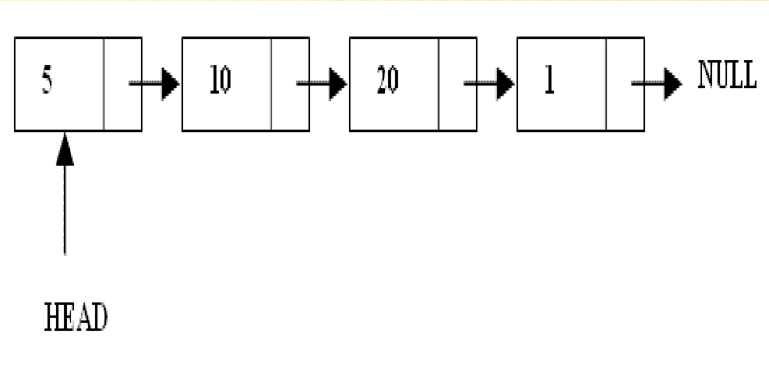
14 100 → 30 400 → 42 N

200 100 400

- The above figure shows the example of marks obtained by different students can be stored in a linked list
- Here N-stands for NULL.
- Null indicates the end of the node.

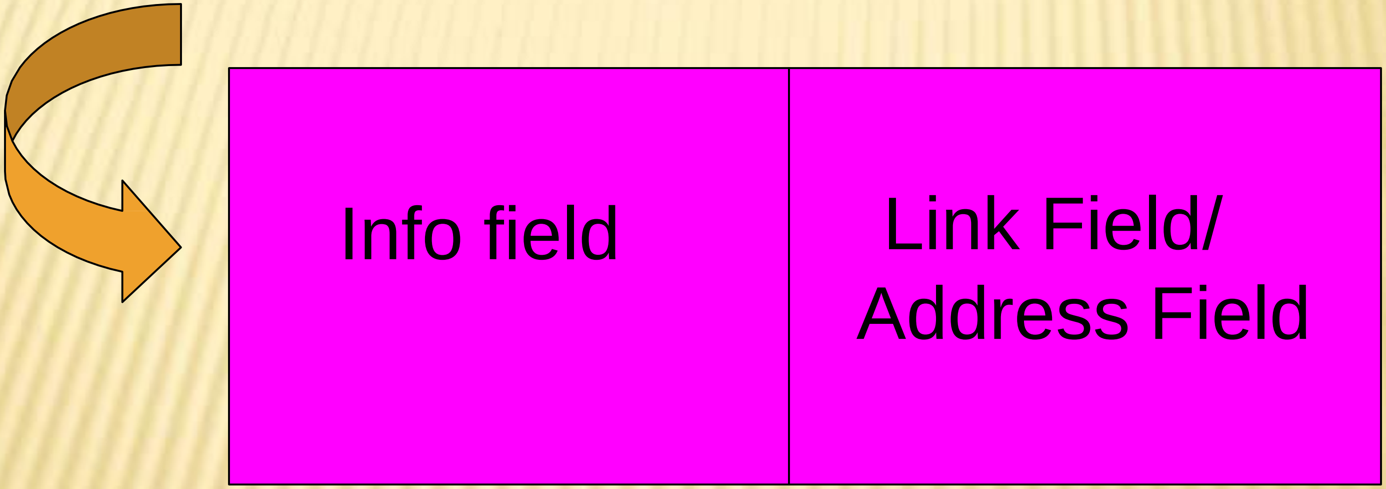
WHAT ARE LINKED LISTS

- A linked list is a linear data structure.
- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node.
- Usually the pointer is called next.



What is Linked List?

A linked list is a collection of nodes with various fields
It contains data field and Address field or Link field



Pointer to the
first node

Info field

Link Field/
Address Field

ARRAYS VS LINKED LISTS

Arrays	Linked list
Fixed size: Resizing is expensive	Dynamic size
Insertions and Deletions are inefficient: Elements are usually shifted	Insertions and Deletions are efficient: No shifting
Random access i.e., efficient indexing	No random access → Not suitable for operations requiring accessing elements by index such as sorting
No memory waste if the array is full or almost full; otherwise may result in much memory waste.	Since memory is allocated dynamically(acc. to our need) there is no waste of memory.
Sequential access is faster [Reason: Elements in contiguous memory locations]	Sequential access is slow [Reason: Elements not in contiguous memory locations]

TYPES OF LINKED LISTS

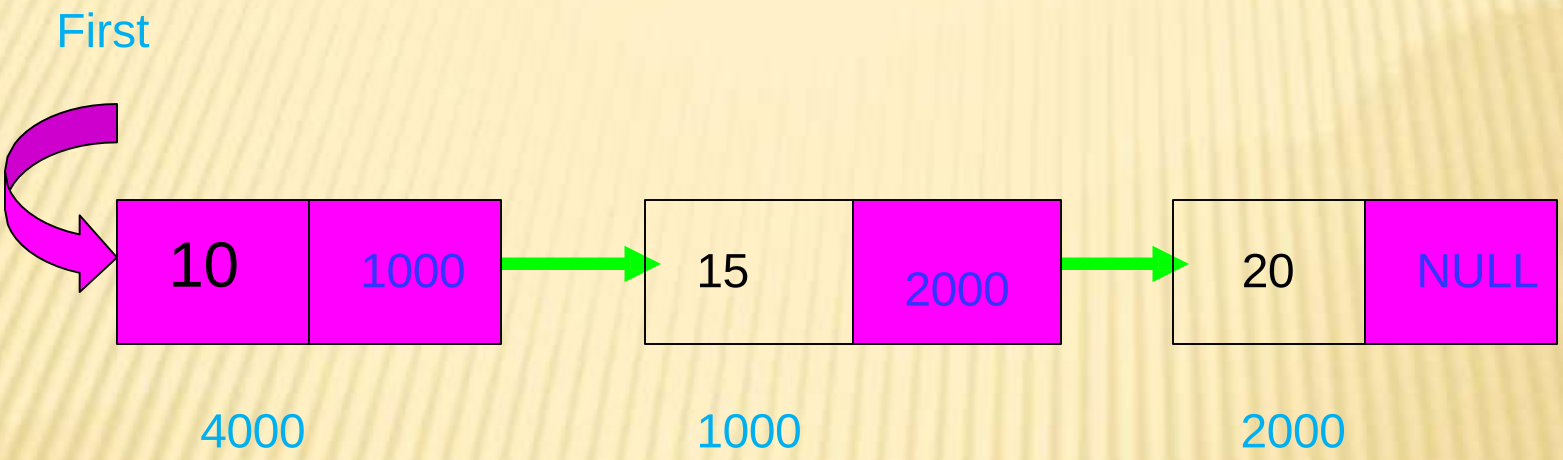
1. Single linked list
2. Double linked list
3. Circular linked list
4. Circular double linked list

SINGLE LINKED LIST :-

- A single linked list is one in which all nodes are linked together in some sequential manner.
- **CIRCULAR LINKED LIST** :-
- A circular linked list is one which has no beginning and no ending. The null pointer in the last node of a linked list is replaced with the address of its first node such a list is called circular linked list.

Graphical Representation

Singly Linked List



First

10 1000

15 2000

20 NULL

4000

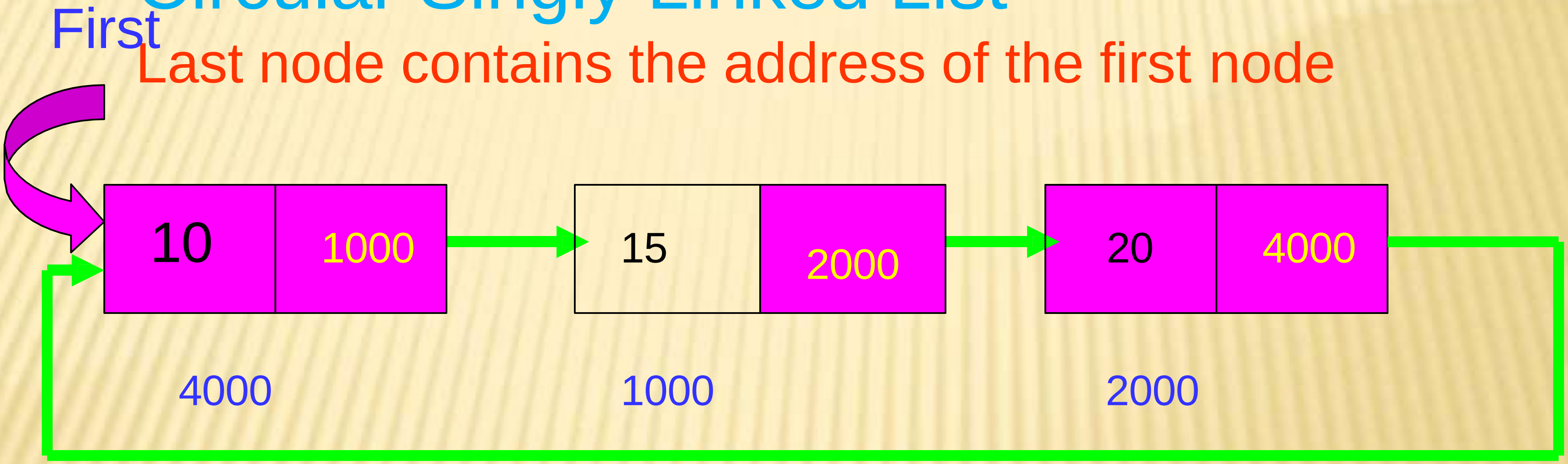
1000

2000

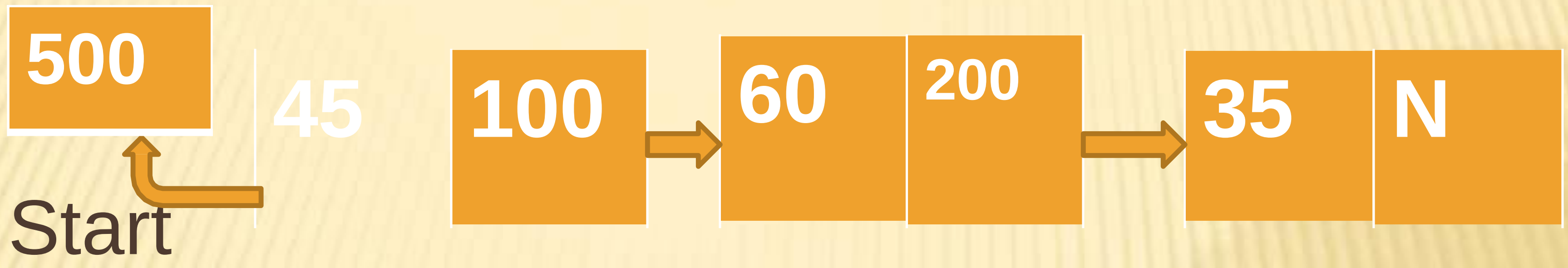
Graphical Representation

Circular Singly Linked List

Last node contains the address of the first node



SINGLE LINKED LIST :



500

Start

45

100

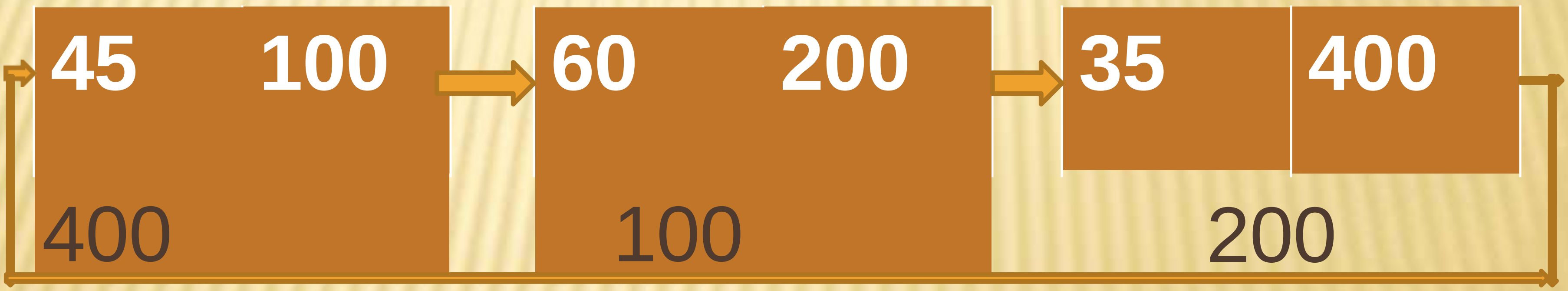
60

200

35

N

CIRCULAR LINKED LIST:-



45

100

400

60

200

100

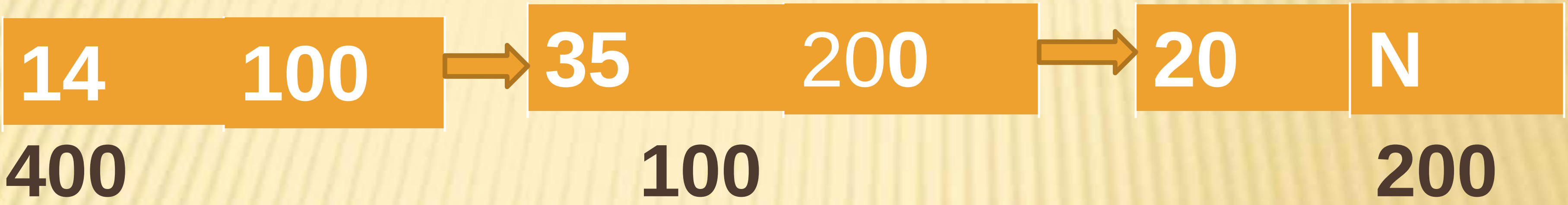
35

400

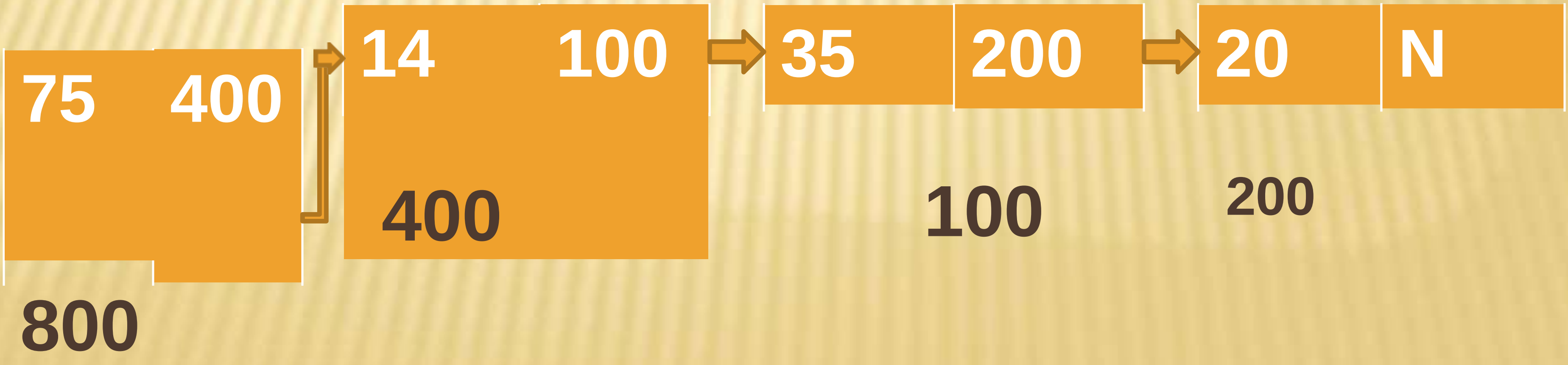
200

<u>Inserting a new node</u> :

Before inserting:

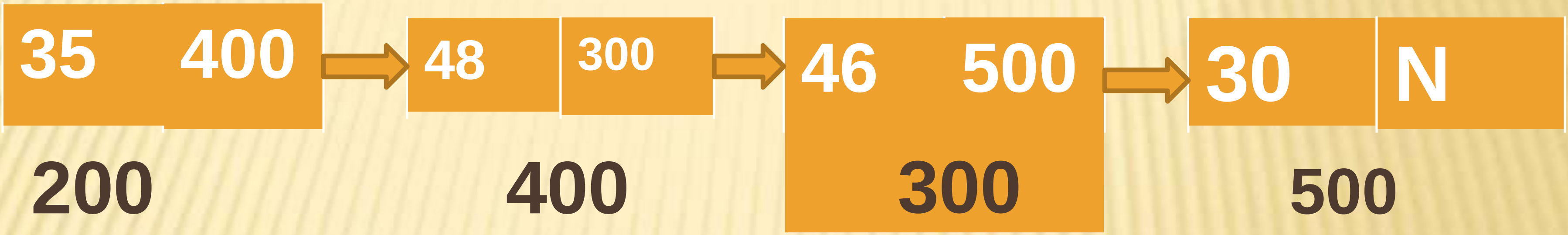


After inserting:



Delete a node from the list:

before deletion:



35 400

48 300

46 500

30 N

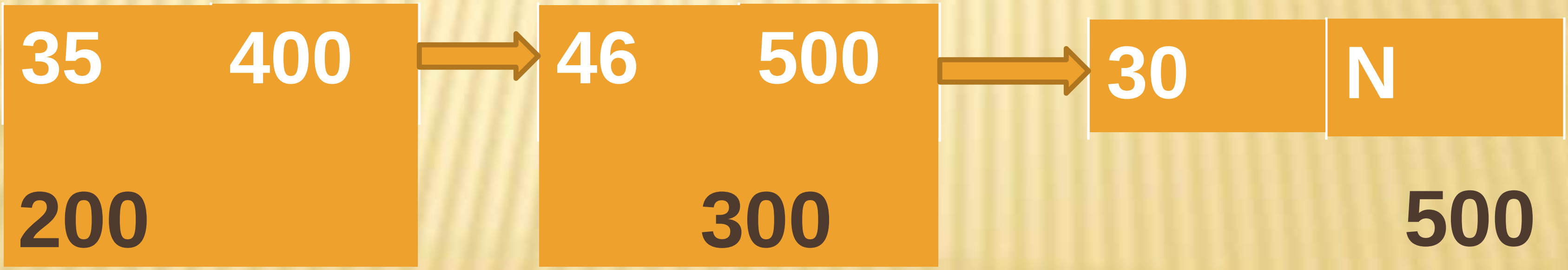
200

400

300

500

after deletion :



35 400

46 500

30 N

200

300

500

- In circular linked list we have three functions. They are

- addcirq( )
- delcirq( )
- cirq_display( )

addcirq( ) :-

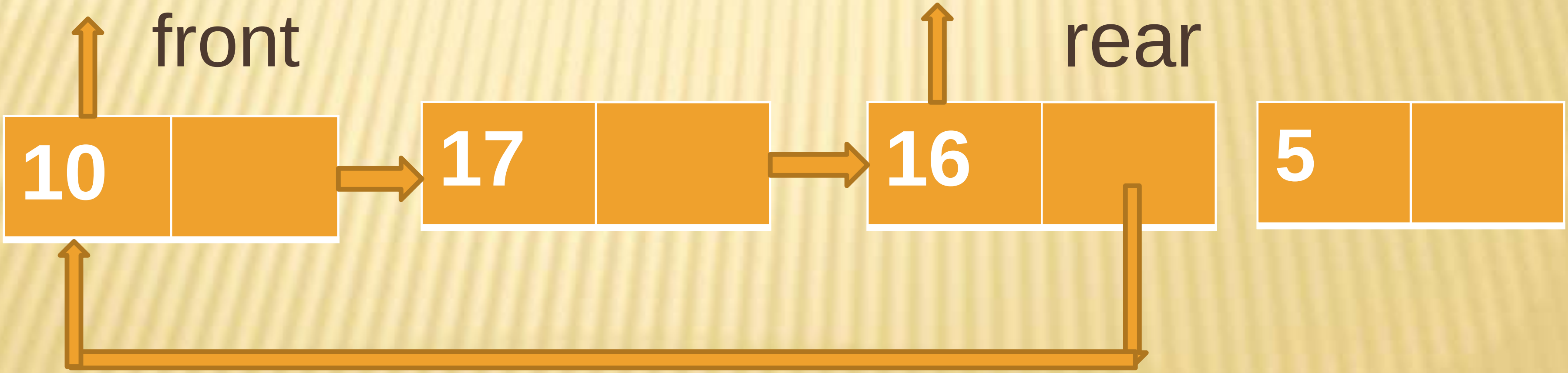
This function accepts three parameters.

- First parameter receives the address of the first node.
- The second parameter receives the address of the pointer to the last node.

Delcirq( ) :-

This function receives two parameters.

- The first parameter is the pointer to the front node.
- The second is the pointer to the rear.



- **DOUBLE LINKED LIST** :-

  A single linked list has some disadvantages

- That it can traverse it in one direction.
- Many applications require searching backword and forword travelling sections of a list

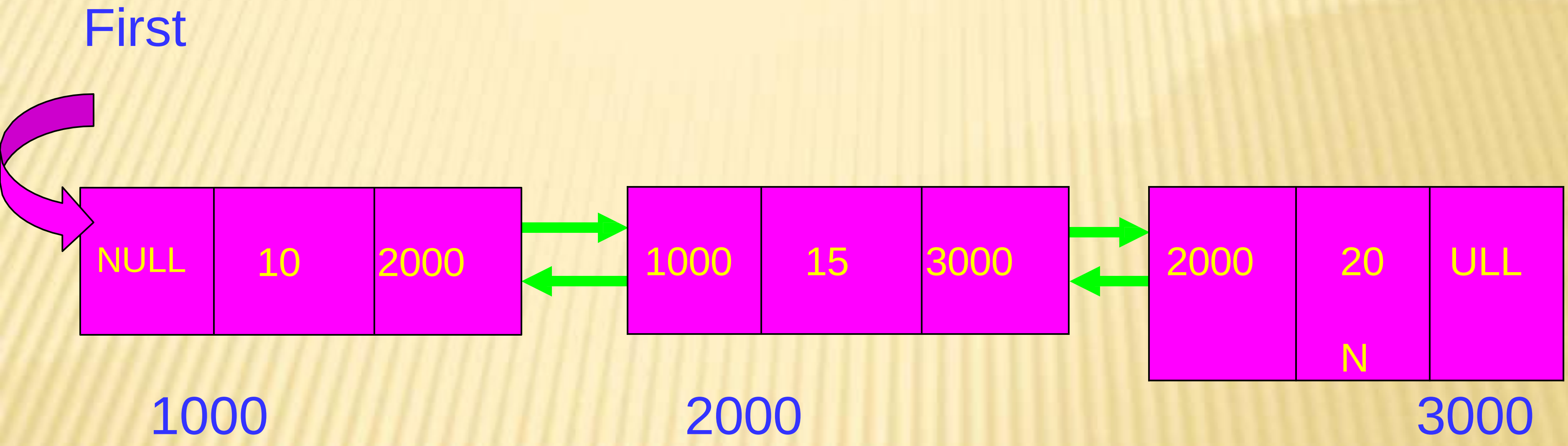
- A two way list is a linear collection of data elements called nodes.
- When each node is divided into three parts. They are two link parts and one data part.

node

prev	**data**	**next**

Doubly Linked list

Contains the address of previous node and next node



First

NULL 10 2000

1000 15 3000

2000 20 N ULL

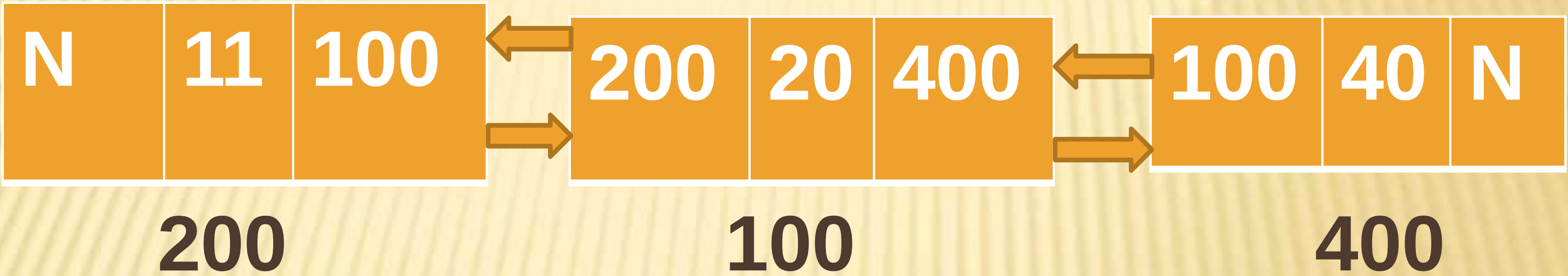
1000

2000

3000

Inserting a new node :

Before insertion:-



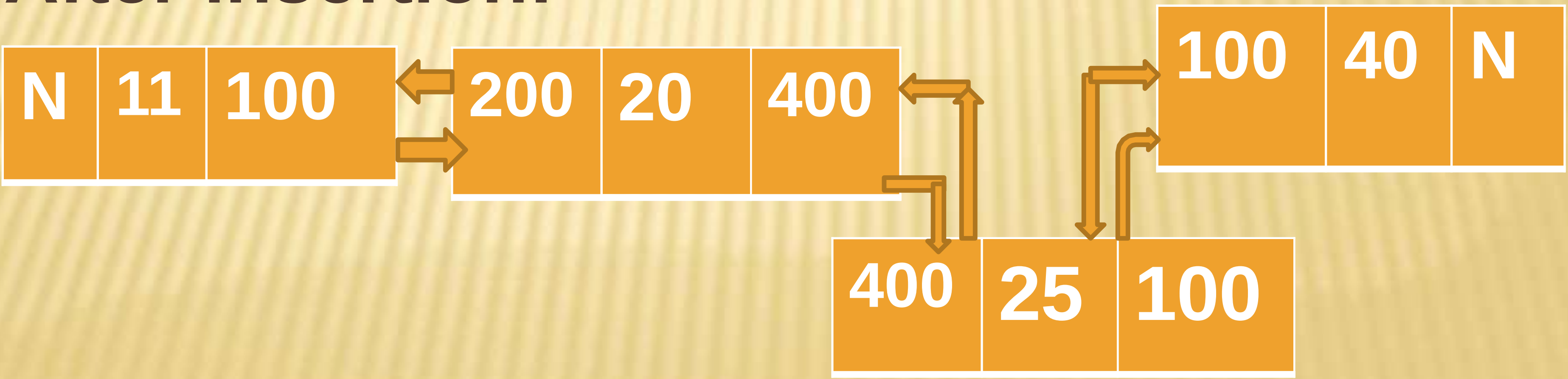
N 11 100

200 20 400

100 40 N

200 100 400

After insertion:-



N 11 100

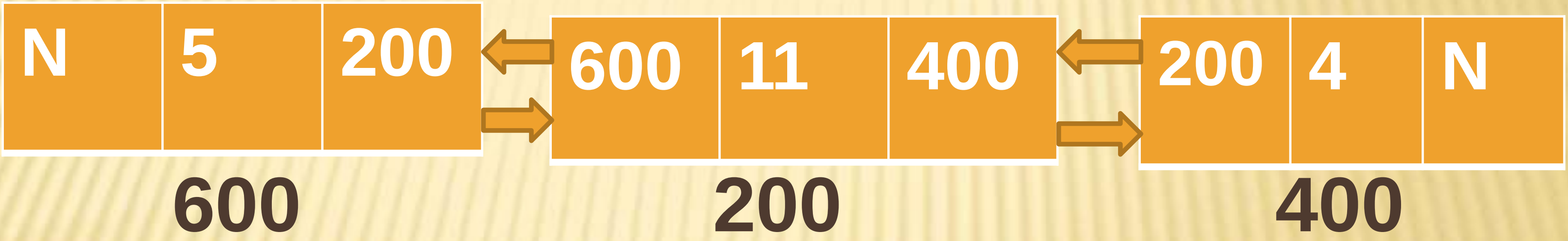
200 20 400

100 40 N

400 25 100

Delete a node from the list:

Before deletion:-



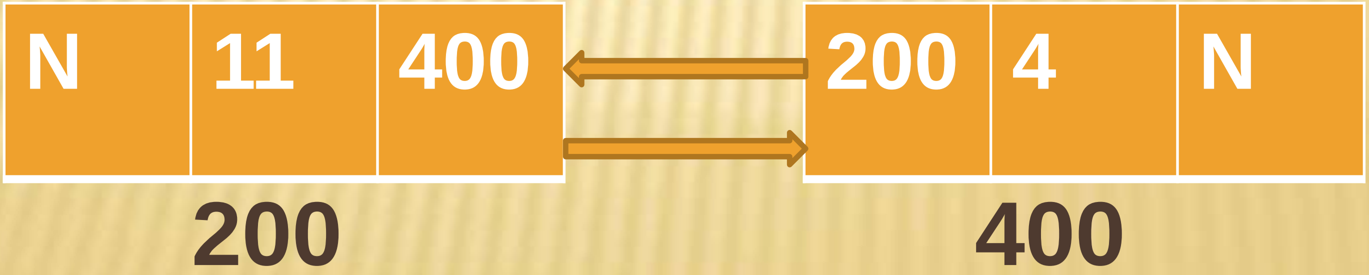
N	5	200
600	11	400
200	4	N

600

200

400

After deletion:-



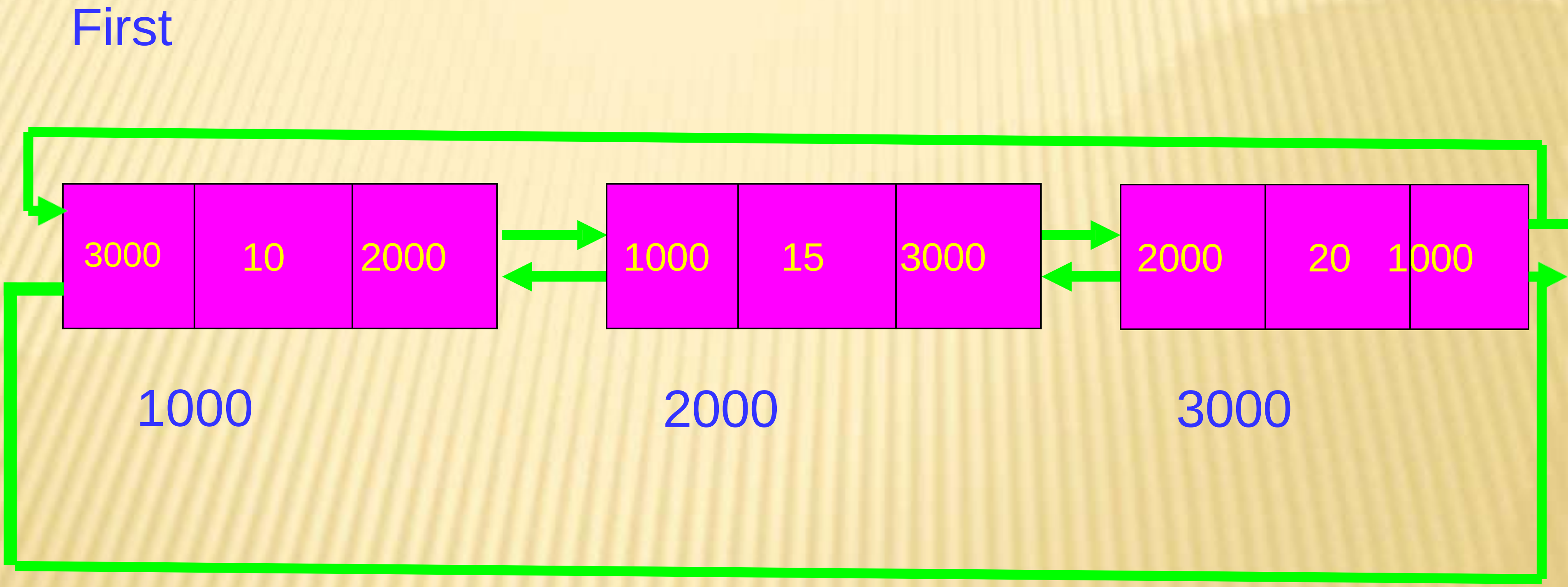
N	11	400
200	4	N

200

400

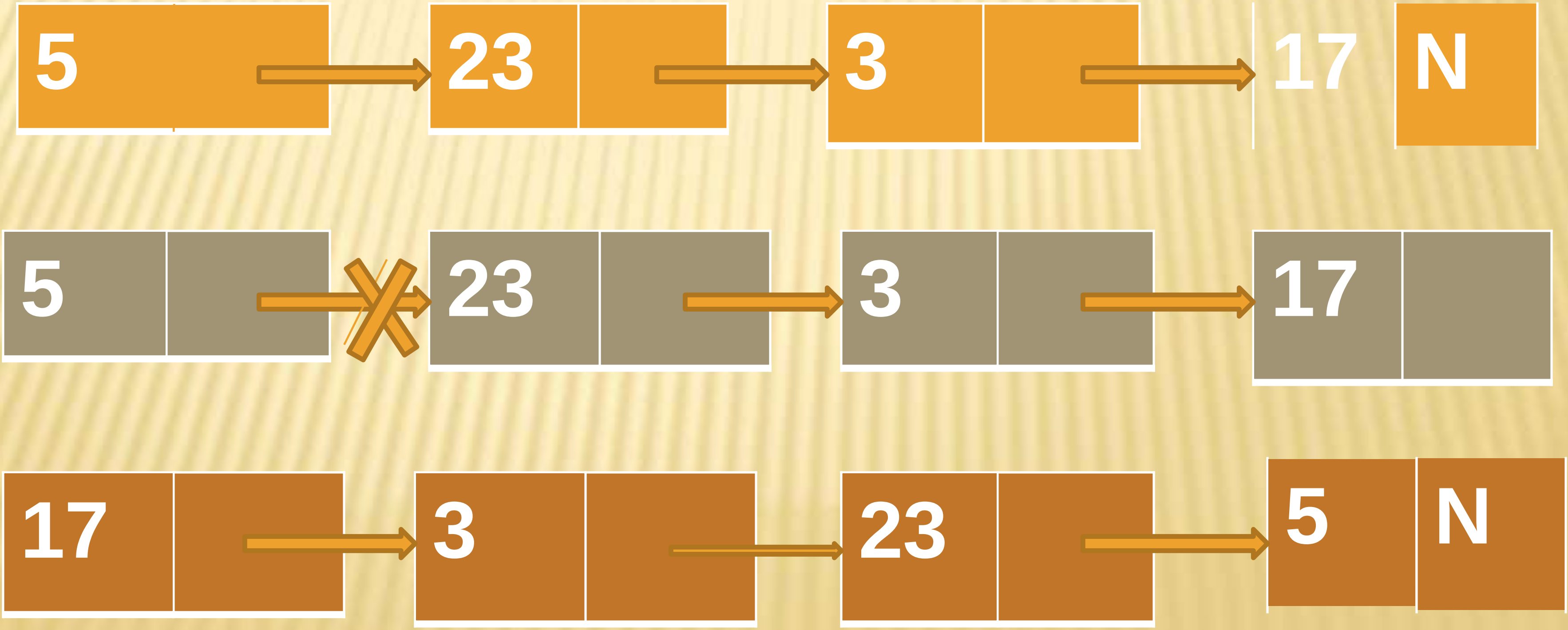
Circular Doubly Linked list

Contains the address of first node and last node



REVERSING THE LINKS :

Reversing means the last node becomes the first node and the first becomes the last.



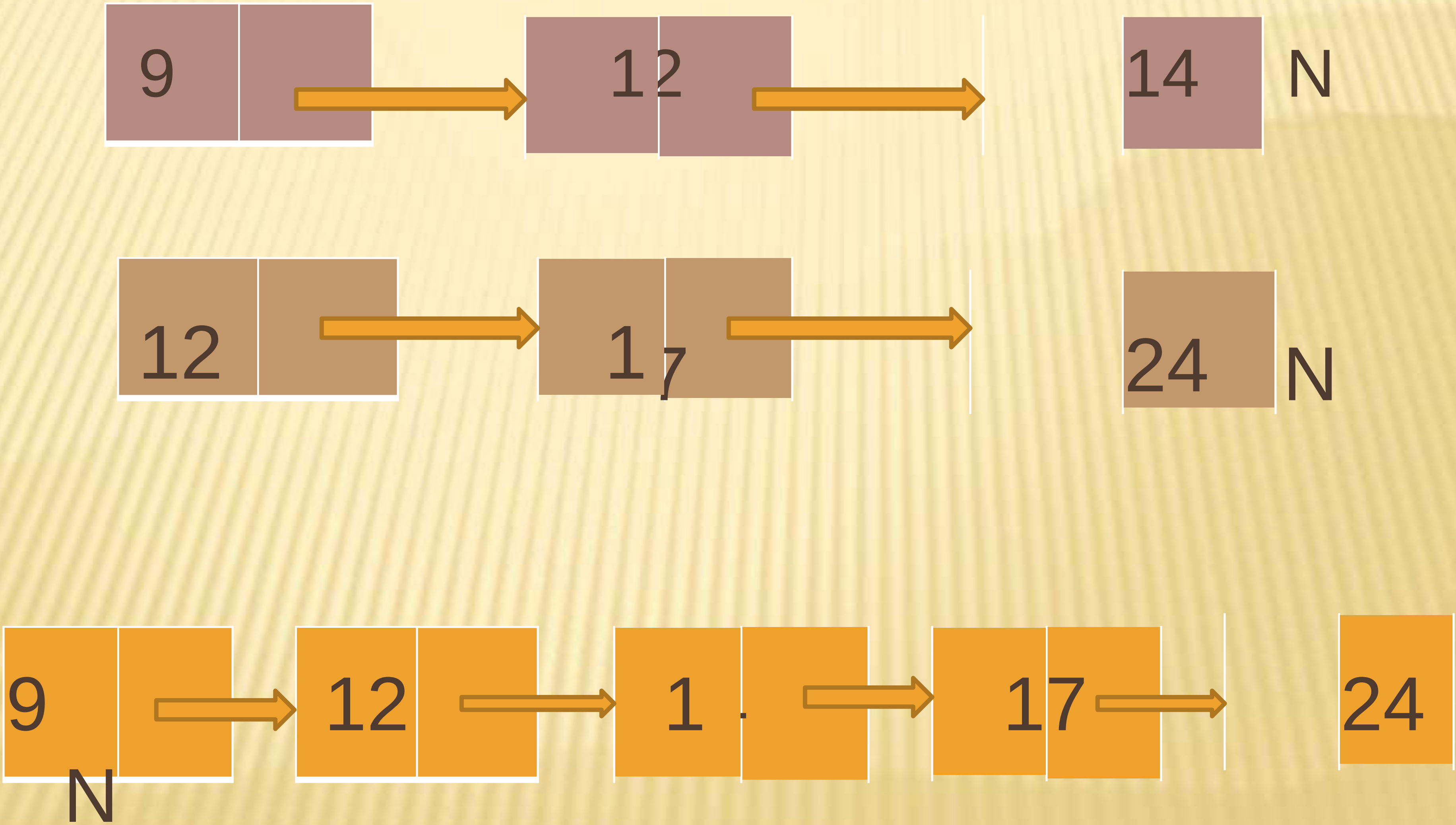
5 → 23 → 3 → 17 N

5 → 23 → 3 → 17

17 → 3 → 23 → 5 N

MERGING OF LINKED LIST :

- suppose we have two linked lists.
- That are pointed to two independent pointers. We have to merge the two links into a third list.
- By using this merge ( ) to ensure that those elements which are common to both the lists occur only once in the third list.



9 → 12 → 14 N

12 → 17 → 24 N

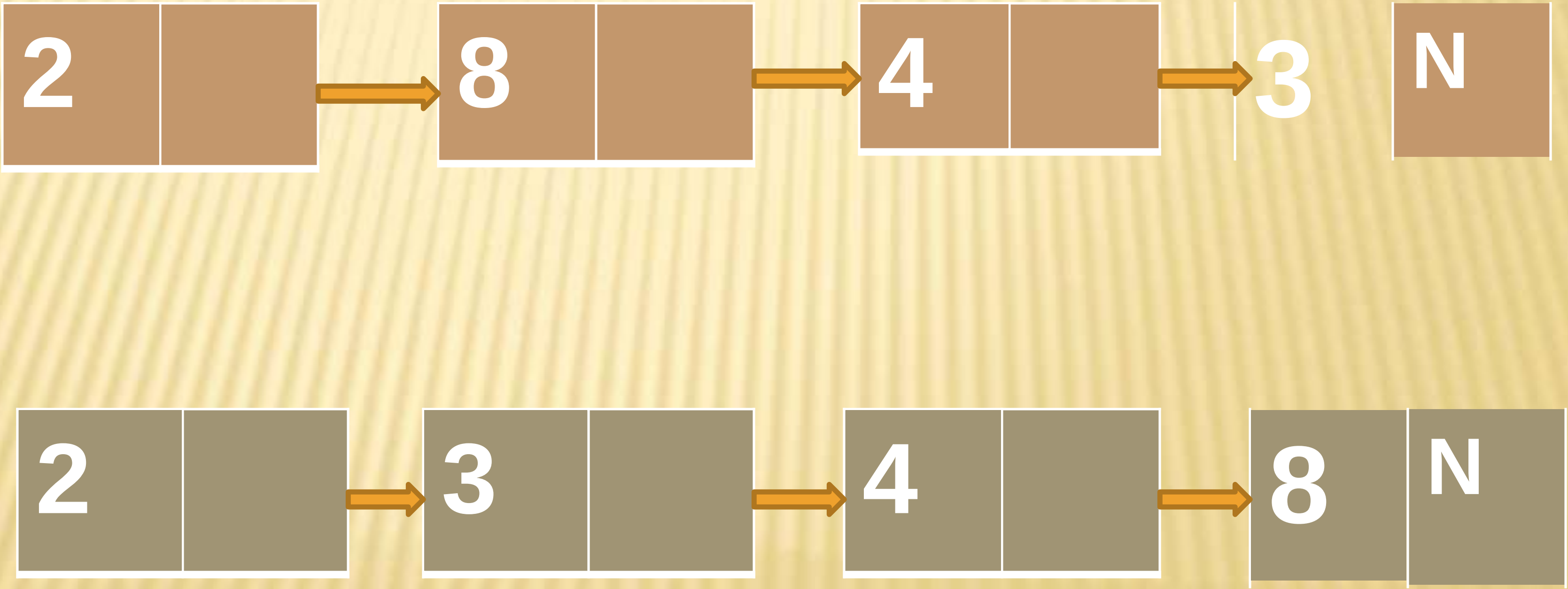
9 → 12 → 1 → 17 → 24

N

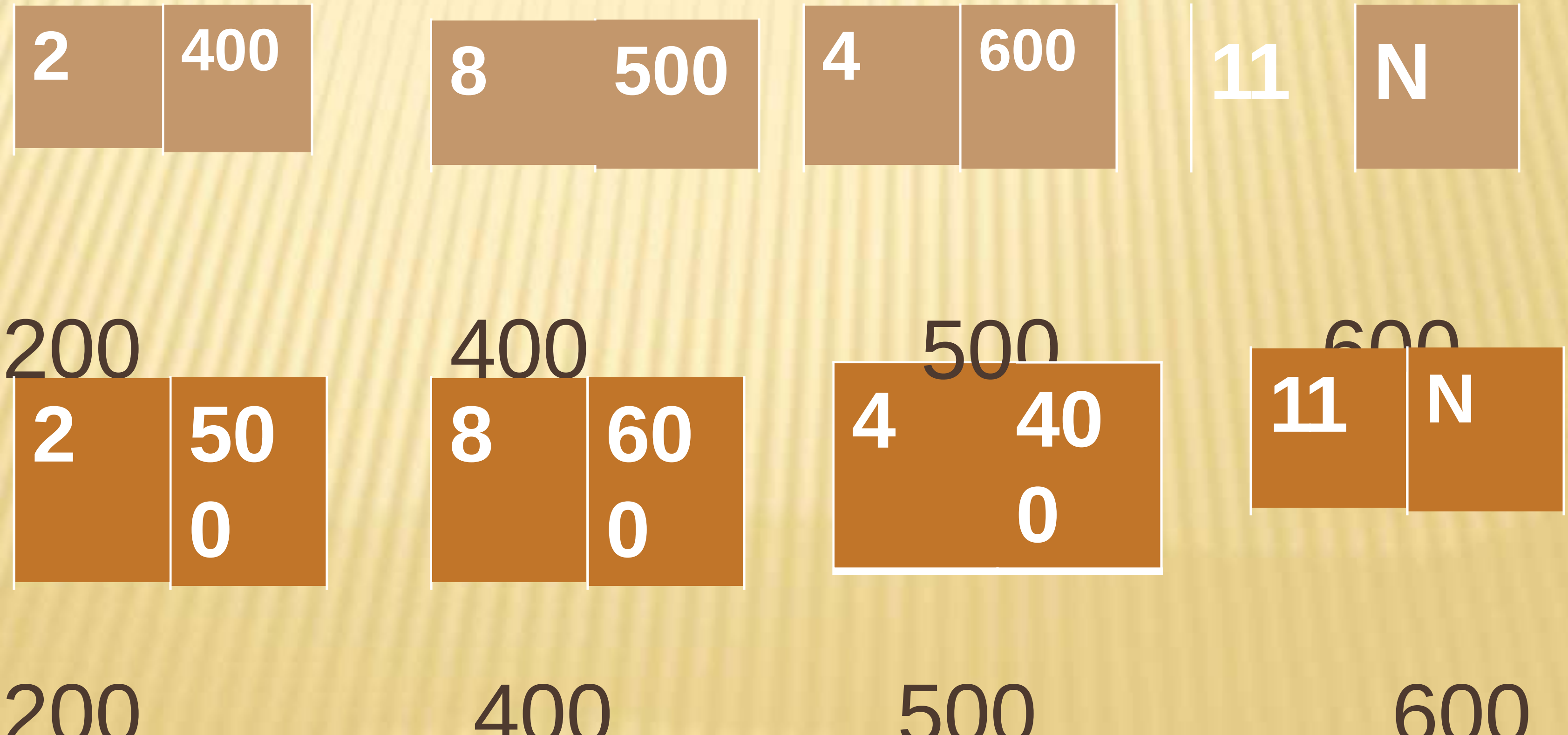
SORTING OF A LINKED LIST :

- Sorting means to arrange the elements either in ascending or descending order.
- To sort the elements of a linked list we use any of the standard sorting algorithms for carrying out the sorting.
- While performing the sorting, when it is time to exchange two elements, we can adopt any of the following two strategies.

1)Exchange the data part of the two nodes, keeping the links intact.



2) Keep the data in the nodes intact. Simply readjust the links such that effectively the order of the nodes changes



2 400

8 500

4 600

11 N

200

400

500

600

2 500

8 600

4 400

11 N

200

400

500

600

RECURSIVE OPERATIONS ON LINKED LIST :

- A function called by itself known as recursion.
- If a statement within the body of a function calls the same function.
- Some of the operations that are carried out on linked list can be easily implemented using recursion.

- For example finding out the number of nodes present in a linked list, comparing two lists, copying one linked list into another, adding a new node at the end of the linked list, etc.,

OPERATIONS ON LINKED LISTS

The basic operations on linked lists are

1. Creation
2. Insertion
3. Deletion
4. Traversing
5. Searching
6. Concatenation
7. Display

- The **creation** operation is used to create a linked list.
- **Insertion** operation is used to insert a new node in the linked list at the specified position. A new node may be inserted at the beginning of a linked list , at the end of the linked list , at the specified position in a linked list. If the list itself is empty , then the new node is inserted as a first node.

- **Deletion** operation is used to delete on item from the linked list. It may be deleted from the beginning of a linked list , specified position in the list.
- **Traversing** operation is a process of going through all the nodes of a linked list from one end to the another end. If we start traversing from the very first node towards the last node , It is called forward traversing.

- If the traversal start from the last node towards the first node , it is called back word traversing.
- **Searching** operation is a process of accessing the desired node in the list. We start searching node –by-node and compare the data of the node with the key.

- **Concatenation** operation is the process of appending the second list to the end of the first list. When we concatenate two lists , the resultant list becomes larger in size.
- The **display** operation is used to print each and every node’s information.

BASIC OPERATIONS ON A LIST

- Creating a List
- Inserting an element in a list
- Deleting an element from a list
- Searching a list
- Reversing a list

INSERTION AT THE BEGINNING

There are two steps to be followed:-

a) Make the next pointer of the node point towards the first node of the list

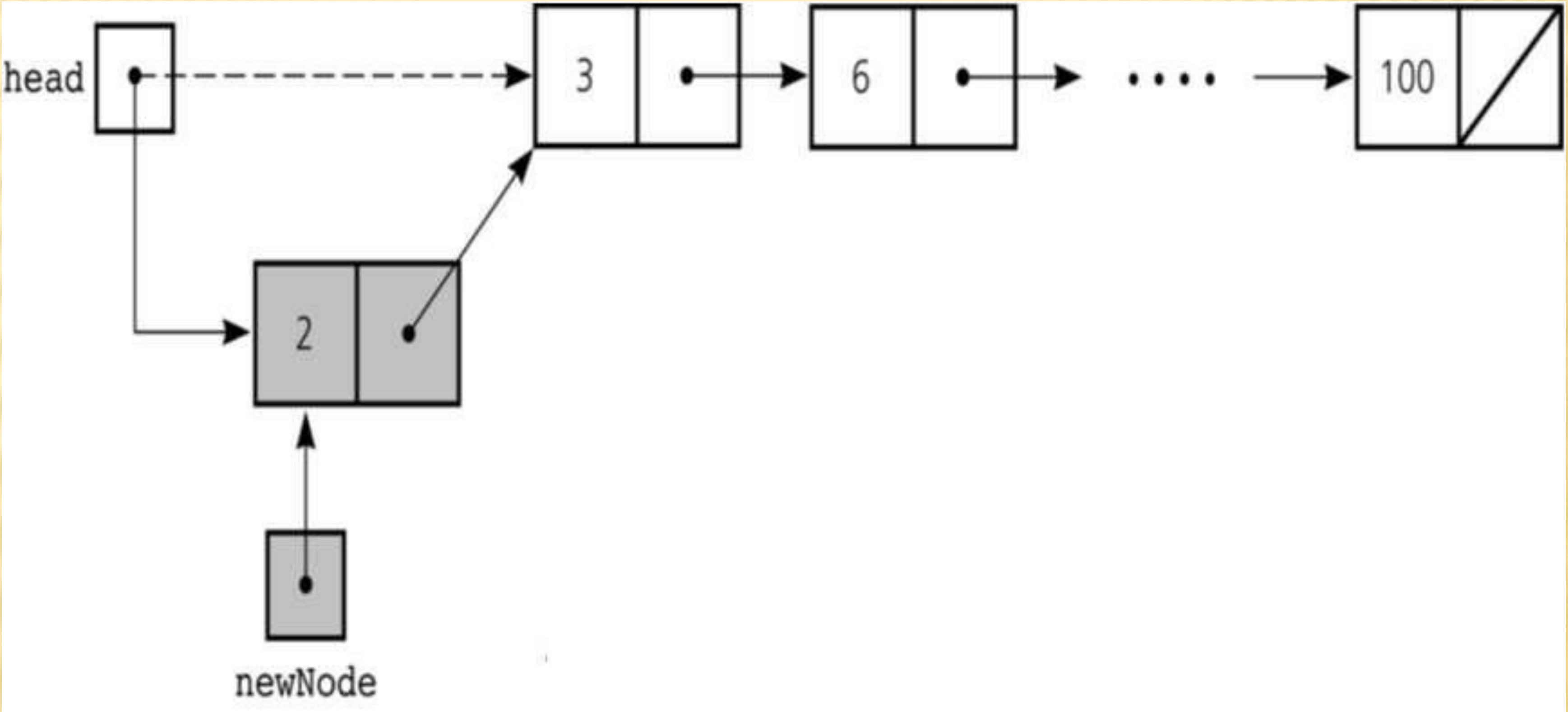
b) Make the start pointer point towards this new node

- If the list is empty simply make the start pointer point towards the new node;

INSERTING THE NODE IN A SLL

There are 3 cases here:-

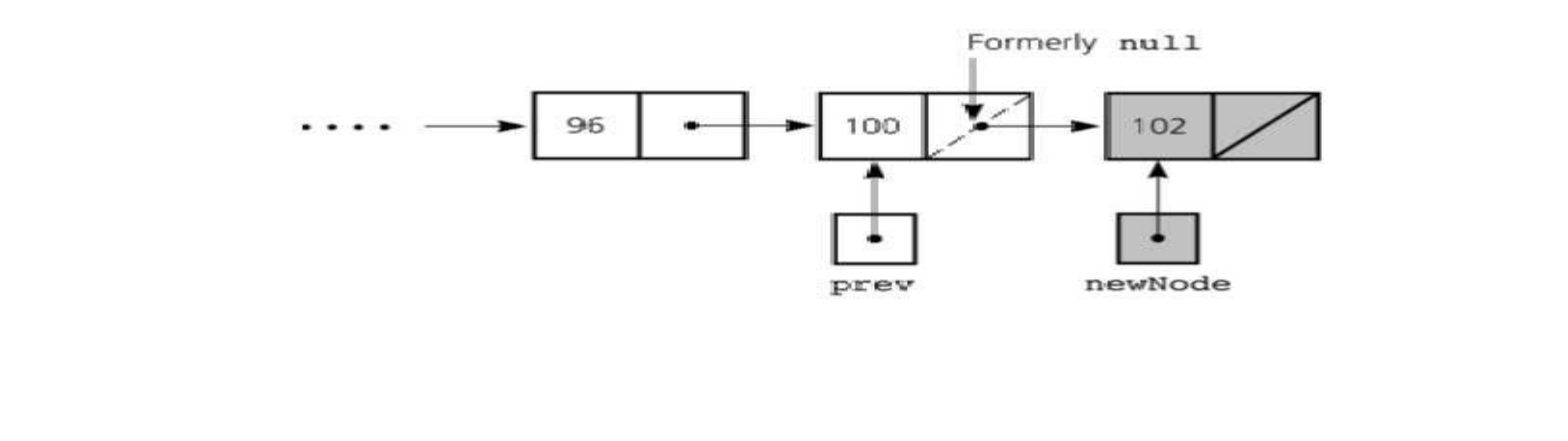
- Insertion at the beginning
- Insertion at the end
- Insertion after a particular node


head
3
6
100
2
newNode

INSERTING AT THE END

Here we simply need to make the next pointer

of the last node point to the new node

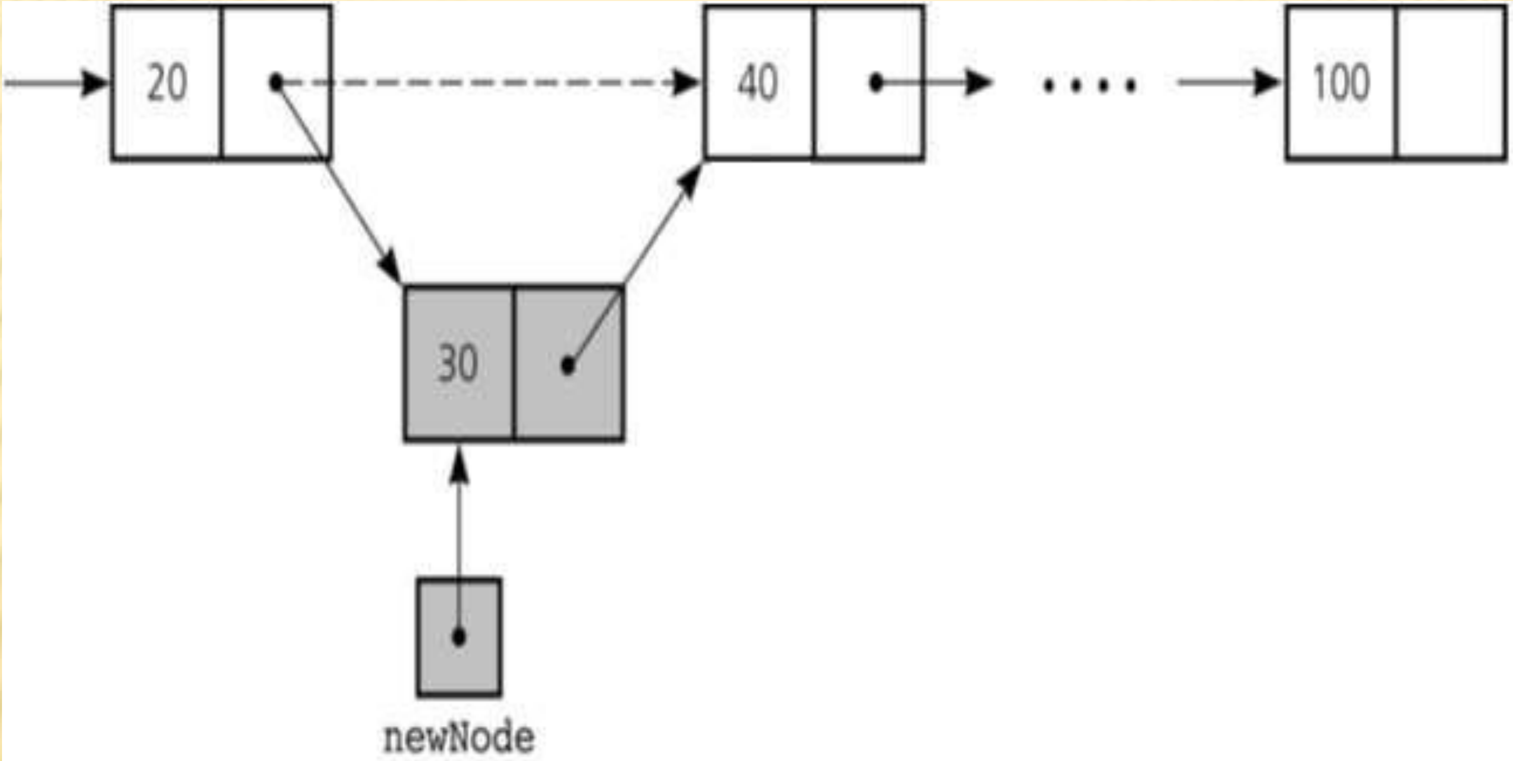


INSERTING AFTER AN ELEMENT

Here we again need to do 2 steps :-

- Make the next pointer of the node to be inserted point to the next node of the node after which you want to insert the node

- Make the next pointer of the node after which the node is to be inserted, point to the node to be inserted


20
40
....
100
30
newNode

DELETING A NODE INSLL

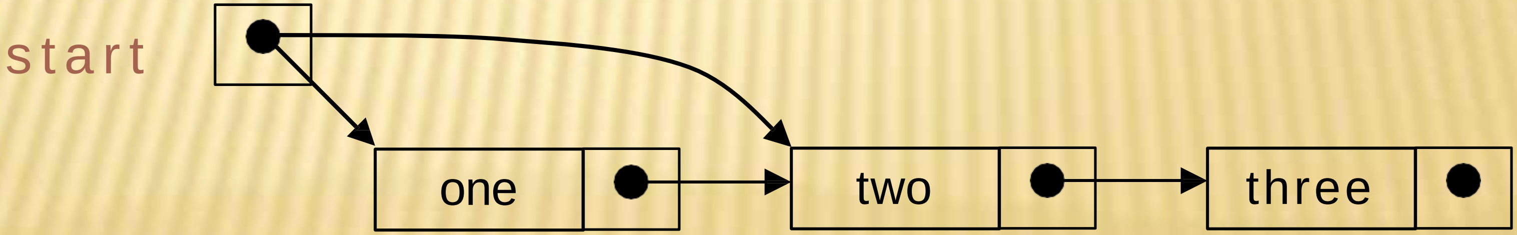
Here also we have three cases:-

- Deleting the first node
- Deleting the last node
- Deleting the intermediate node

DELETING THE FIRST NODE

Here we apply 2 steps:-

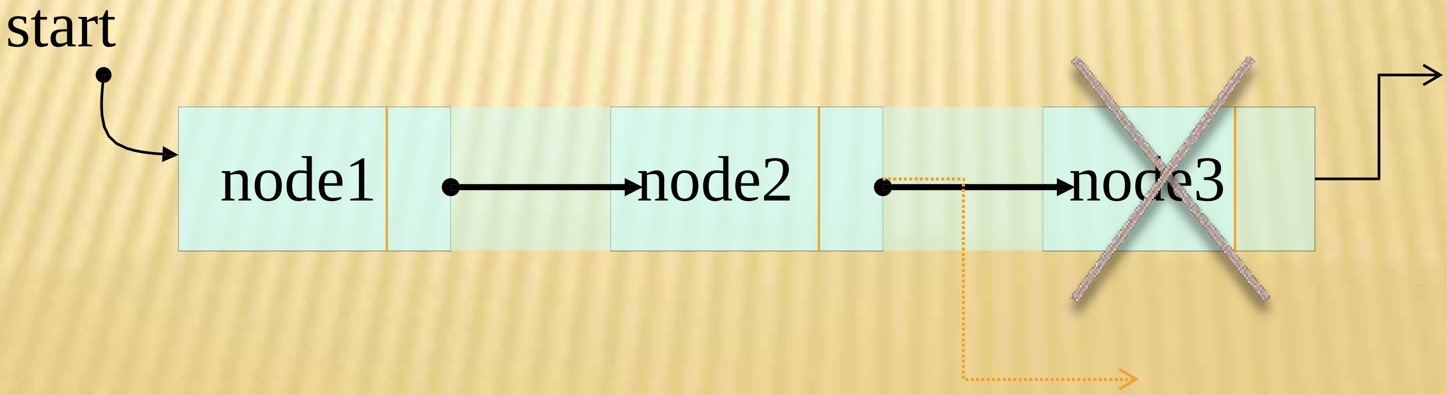
- Making the start pointer point towards the 2nd node
- Deleting the first node using delete keyword



DELETING THE LAST NODE

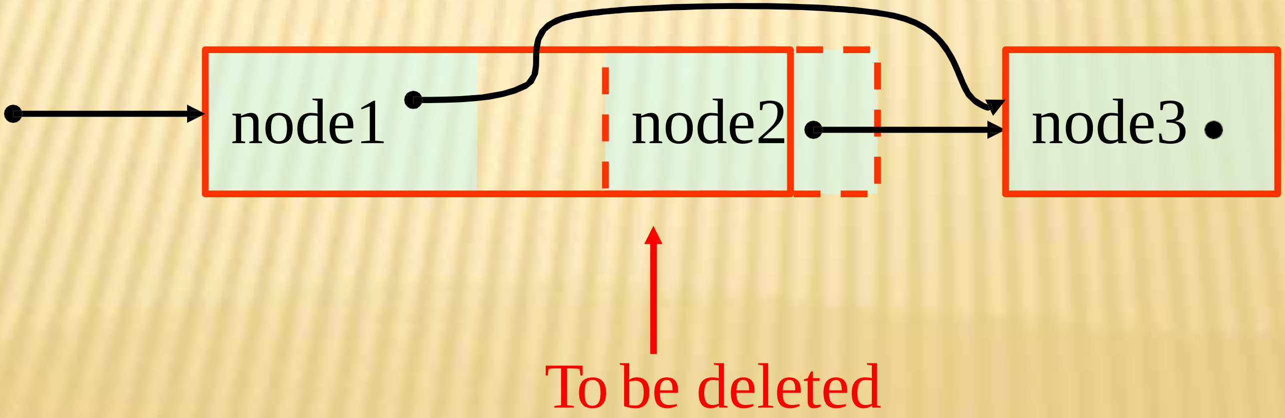
Here we apply 2 steps:-

- Making the second last node's next pointer point to NULL
- Deleting the last node via delete keyword



DELETING A PARTICULAR NODE

Here we make the next pointer of the node previous to the node being deleted ,point to the successor node of the node to be deleted and then delete the node using delete keyword

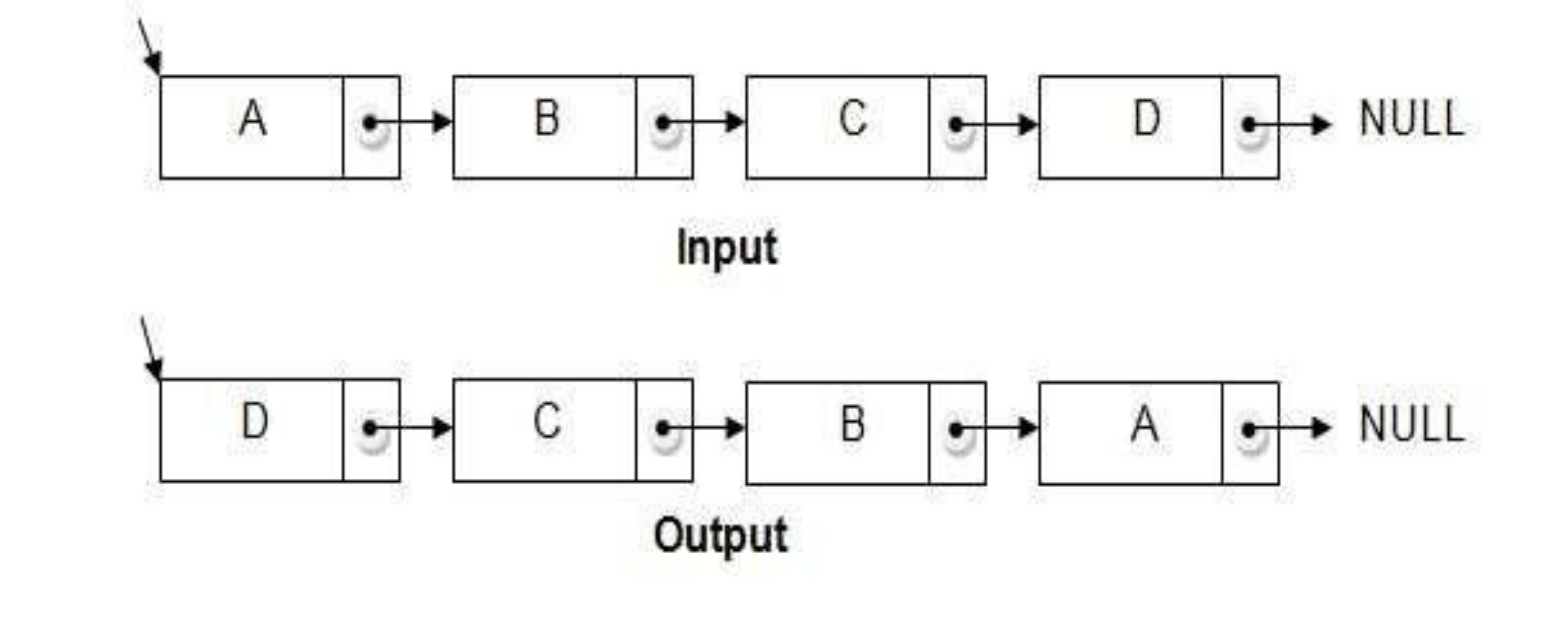


SEARCHING

- Searching involves finding the required element in the list
- We can use various techniques of searching like linear search or binary search where binary search is more efficient in case of Arrays
- But in case of linked list since random access is not available it would become complex to do binary search in it
- We can perform simple linear search traversal
- In linear search each node is traversed till the data in the node matches with the required value

REVERSING A LINKED LIST

- We can reverse a linked list by reversing the direction of the links between 2 nodes



COMPLEXITY OF VARIOUS OPERATIONS IN ARRAYS AND SLL

Operation	**ID-Array Complexity**	**Singly-linked list Complexity**
Insert at beginning	O(n)	O(1)
Insert at end	O(1)	O(1) if the list has **tail** reference O(n) if the list has no **tail** reference
Insert at middle	O(n)	O(n)
Delete at beginning	O(n)	O(1)
Delete at end	O(1)	O(n)
Delete at middle	O(n): O(1) access followed by O(n) shift	O(n): O(n) search, followed by O(1) delete
Search	O(n) linear search O(log n) Binary search	O(n)
Indexing: What is the element at a given position k?	O(1)	O(n)

DOUBLY LINKED LIST

1. **Doubly linked list** is a linked data structure that consists of a set of sequentially linked records called nodes.

2 . Each node contains three fields ::

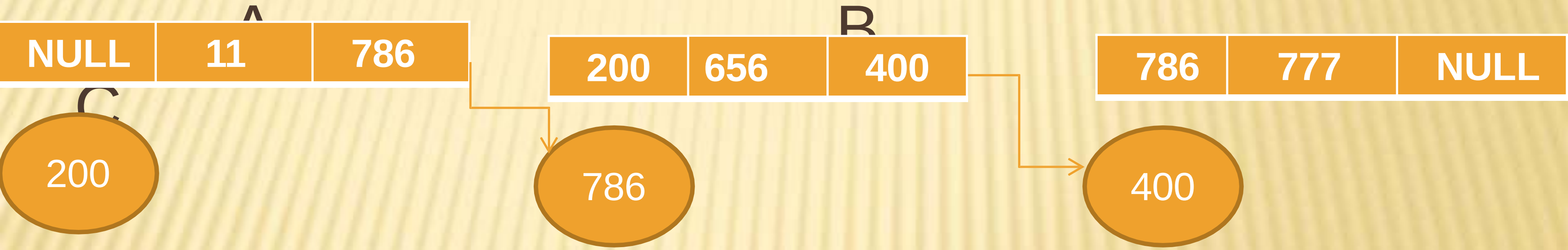
-: one is data part which contain data only.

-:two other field is links part that are point or references to the previous or to the next node in the sequence of nodes.

3. The beginning and ending nodes' **previous** and **next** links, respectively, point to some kind of terminator, typically a sentinel node or null to facilitate traversal of the list.

NODE

previous	data	next



A

NULL	11	786

C

200

B

200	656	400

786

786	777	NULL

400

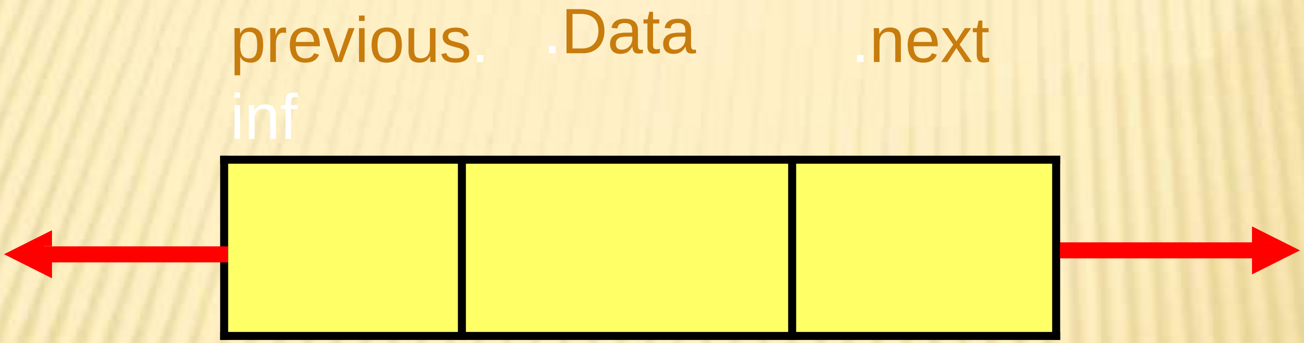
A doubly linked list contain three fields: an integer value, the link to the next node, and the link to the

DLL’S COMPARED TOSLL’S

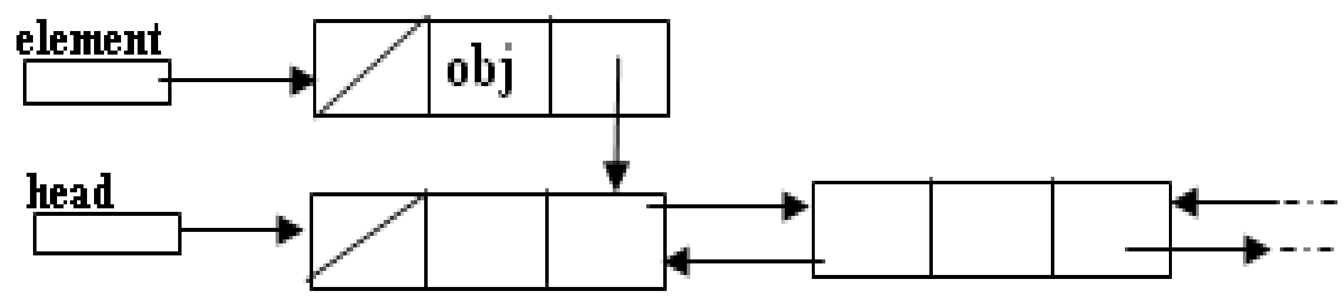
- Advantages:
  - Can be traversed in either direction (may be essential for some programs)
  - Some operations, such as deletion and inserting before a node, become easier

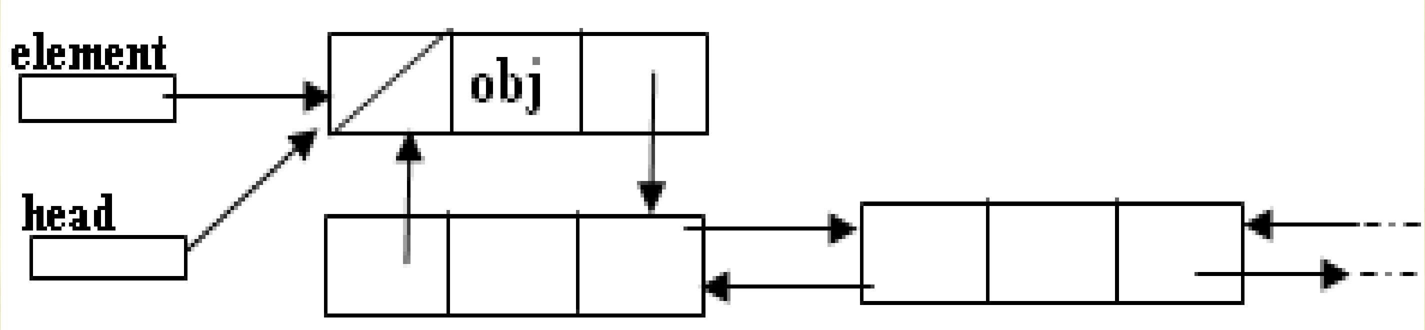
- Disadvantages:
  - Requires more space
  - List manipulations are slower (because more links must be changed)
  - Greater chance of having bugs (because more links must be manipulated)

STRUCTURE OF DLL

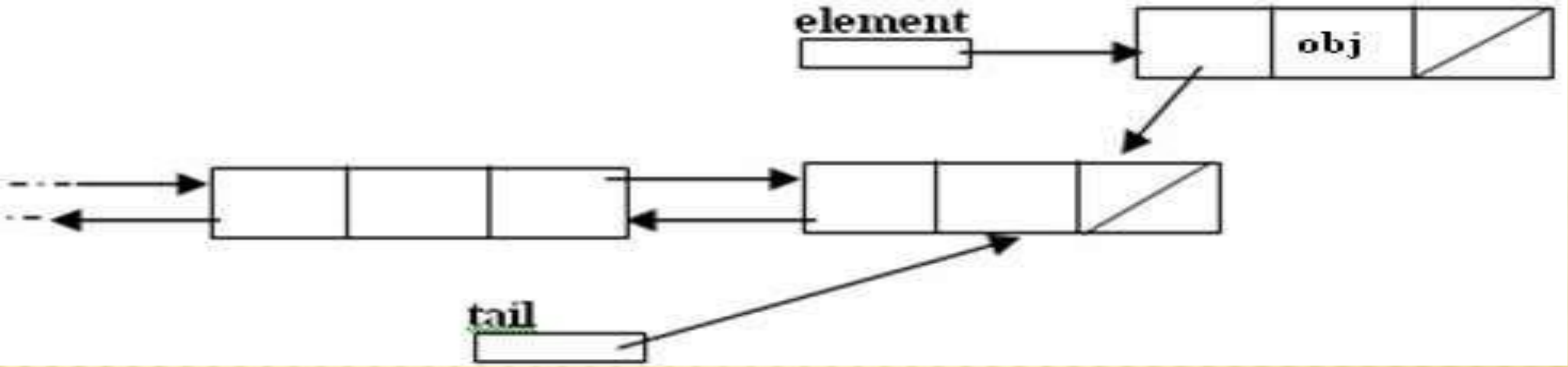


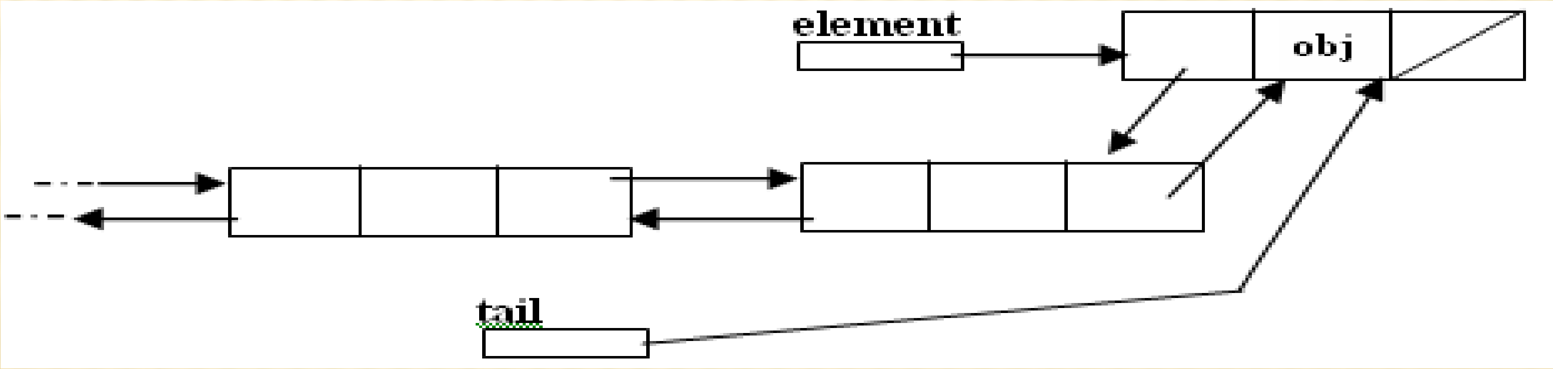
INSERTING AT BEGINNING



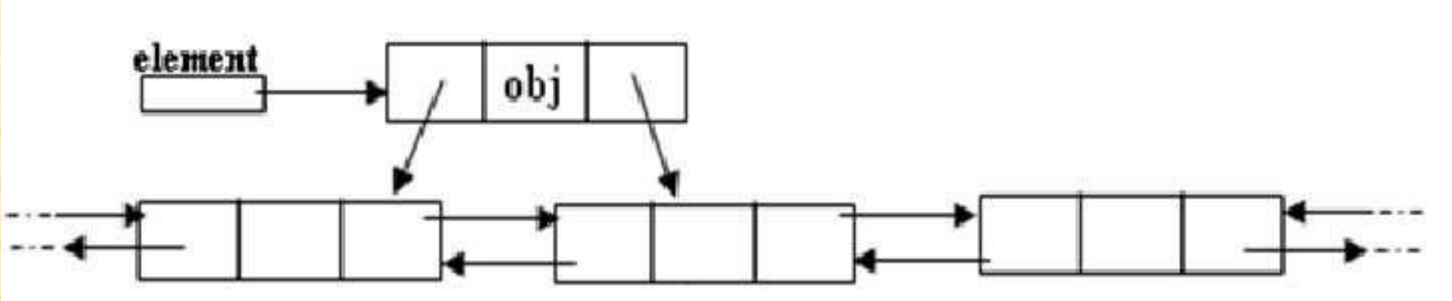


INSERTING AT THE END

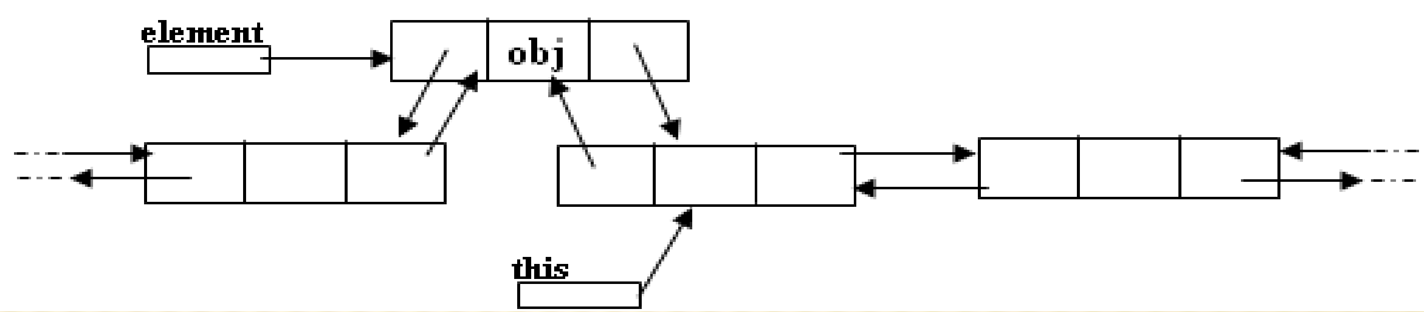




INSERTING AFTER A NODE



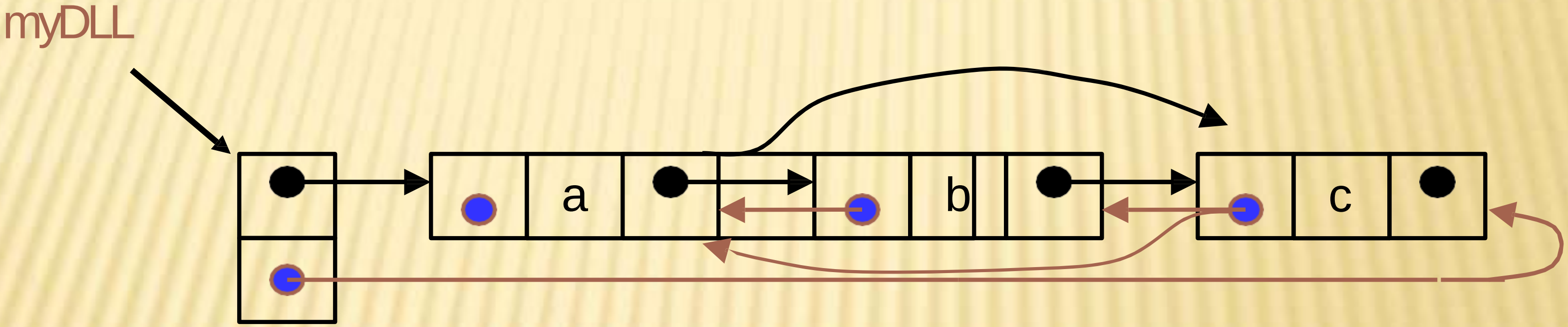
Making next and previous pointer of the node to be inserted point accordingly



Adjusting the next and previous pointers of the nodes b/w which the new node accordingly

DELETING A NODE

- Node deletion from a DLL involves changing *two* links
- In this example,we will delete node b



- We don’t have to do anything about the links in node b
- Garbage collection will take care of deleted nodes
- Deletion of the first node or the last node is a special case

ADVANTAGES OF LINKED LISTS

- We can dynamically allocate memory space as needed
- We can release the unused space in the situation where the allocated space seems to be more.
- Operation related to data elements like insertions or deletion are more simplified.

- Operation like insertion or deletion are less time consuming.
- Linked lists provide flexibility in allowing the items to be arranged efficiently.