Data Structure

Trainer : Nisha Dingare

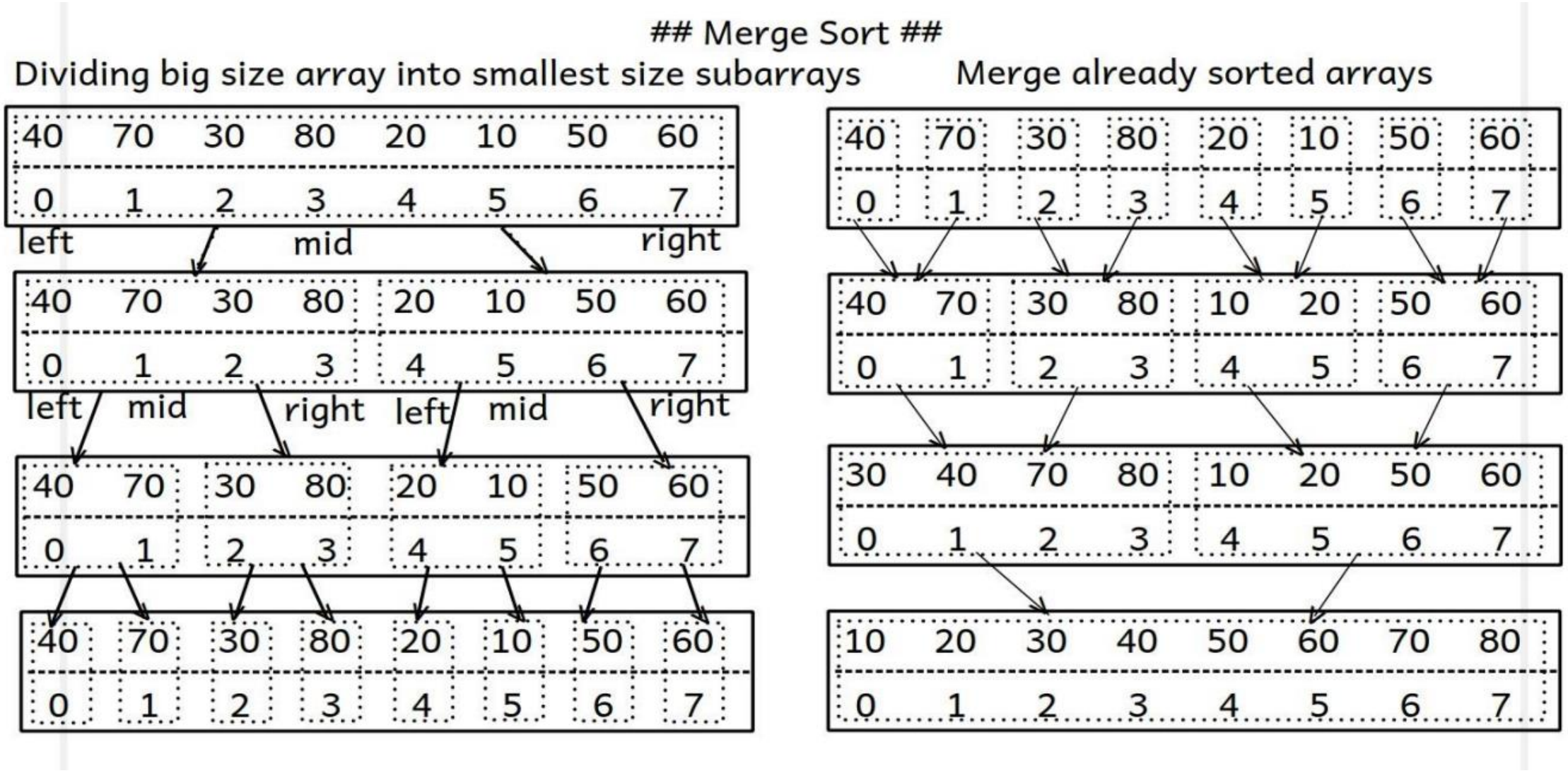
Email : nisha.dingare@sunbeaminfo.com

Merge Sort :

- This algorithm follows divide-and-conquer approach.
- In this algorithm, big size array is divided logically into smallest size (i.e. having size 1) subarrays, after dividing array into smallest size subarray's, subarrays gets merged into one array step by step in a sorted manner and finally all array elements gets arranged in a sorted manner. –
- This algorithm works fine for even as well odd input size array. -This algorithm takes extra space to sort array elements, and hence its space complexity is more.

- **Time Complexity** :
  - Best Case : Ω(n log n)
  - Worst Case : O(n log n)
  - Average Case : θ(n log n)

Merge Sort :


Merge Sort
Dividing big size array into smallest size subarrays
Merge already sorted arrays
40 70 30 80 20 10 50 60
0 1 2 3 4 5 6 7
left
mid
right
40 70 30 80 20 10 50 60
0 1 2 3 4 5 6 7
left
mid
right
left
mid
right
40 70 30 80 20 10 50 60
0 1 2 3 4 5 6 7
40 70 30 80 20 10 50 60
0 1 2 3 4 5 6 7
40 70 30 80 20 10 50 60
0 1 2 3 4 5 6 7
40 70 30 80 10 20 50 60
0 1 2 3 4 5 6 7
30 40 70 80 10 20 50 60
0 1 2 3 4 5 6 7
10 20 30 40 50 60 70 80
0 1 2 3 4 5 6 7

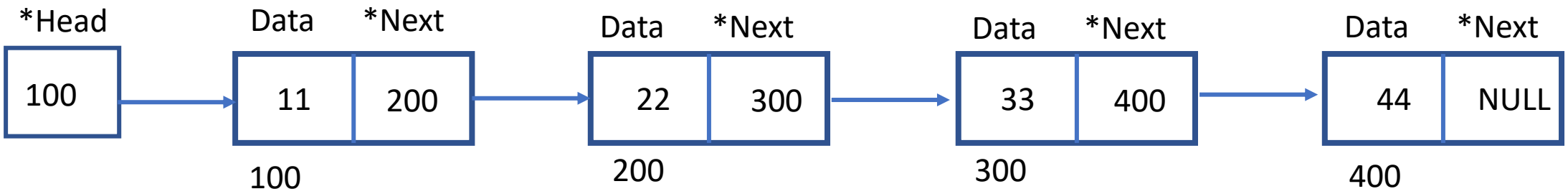
Quick Sort :

- This algorithm follows divide-and-conquer approach.
- In this algorithm the basic logic is a partitioning.
  - Partitioning: in partitioning, pivot element gets selected first (it may be either leftmost or rightmost or middle most element in an array).
  - After selection of pivot element all the elements which are smaller than pivot gets arranged towards the left and elements which are greater than pivot gets arranged to the right.
  - Next, The big size array is divided into two subarray's, so after first pass pivot element gets settled at its appropriate position. elements which are at left of pivot is referred as left partition and elements which are at its right referred as a right partition.
  - Quick sort algorithm is an efficient sorting algorithm for larger input size array.

- Time Complexity :
  - Best Case : Ω(n log n)
  - Worst Case : O(n2) – worst case rarely occurs
  - Average Case : θ(n log n)

Linked List - Introduction :

- Like Arrays , Linked List is a LINEAR DATA STRUCTURE.
- But unlike arrays, Linked Lists are NOT stored in contiguous memory locations.
- The items in the linked lists are connected to one another using pointers.
- These items are called as NODES.
- Each node contains 2 parts :
  - Data part : it contains actual data (primitive or non-primitive data).
  - Next part : it is a pointer which contains the address of the next node.
- Address of the first node is stored in the Head pointer.



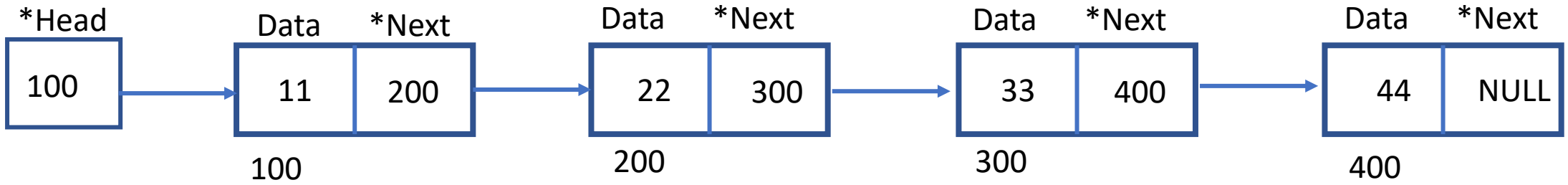
Linked Lists :

- Why Linked List ?
  - Linked Lists are dynamically allocated as and when required and released after it serves its purpose. so unlike arrays they can grow or shrink in size at runtime.
  - Adding or deleting an item in a linked list is much efficient and easier compared to arrays.

- Types of Linked Lists :
  1. Singly Linear Linked List
  2. Singly Circular Linked List
  3. Doubly Linear Linked List
  4. Doubly Circular Linked List

Singly Linear Linked List :

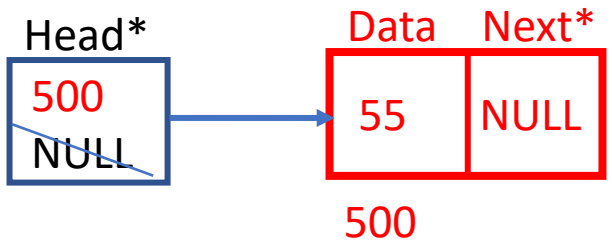
- This linked list is also called as simple linked list.
- It is a type of linked list in which the list can be traversed only in one direction.
- In this, the pointer of each node points to other nodes whereas the pointer of last node contains NULL.
- Address of first node is stored in the Head pointer.



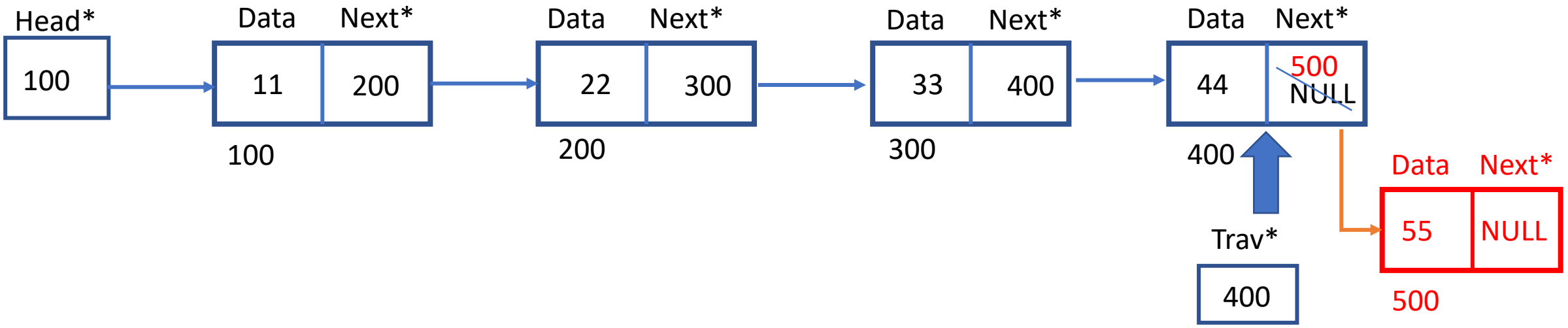
- Operations performed on linked list :
  - Add node at first position
  - Add node at last position
  - Add node at specific position
  - Delete node at first position
  - Delete node at last position
  - Delete node at specific position
  - Traverse the list

Singly Linear Linked List - Add at Last position :

- If Head is NULL, Add the new node to the Head.

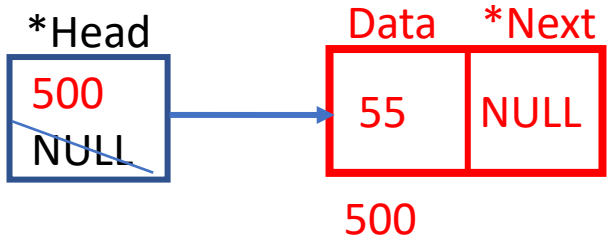


- Else, Traverse the list till the last node and add the address of new node to the next part of the last node.

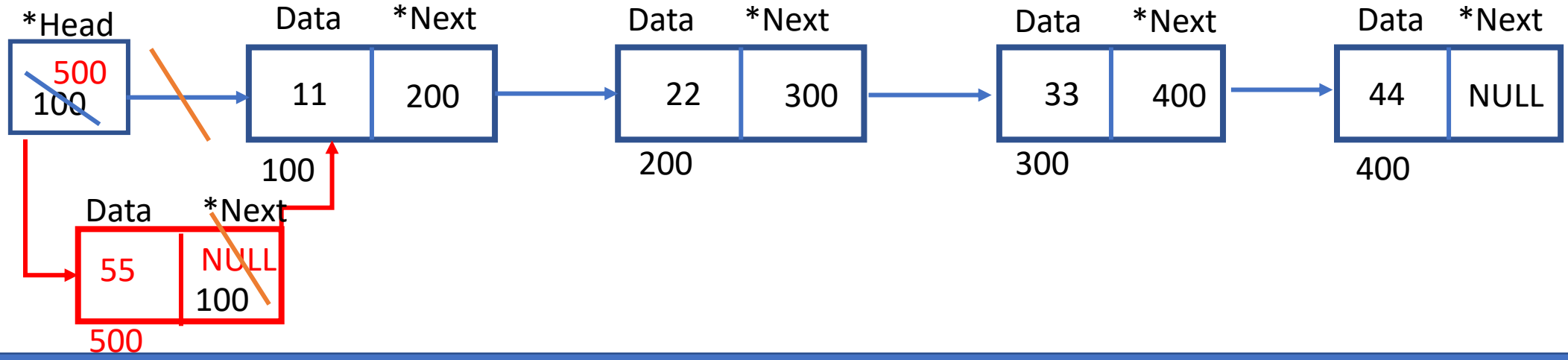


Singly Linear Linked List – Add at First position :

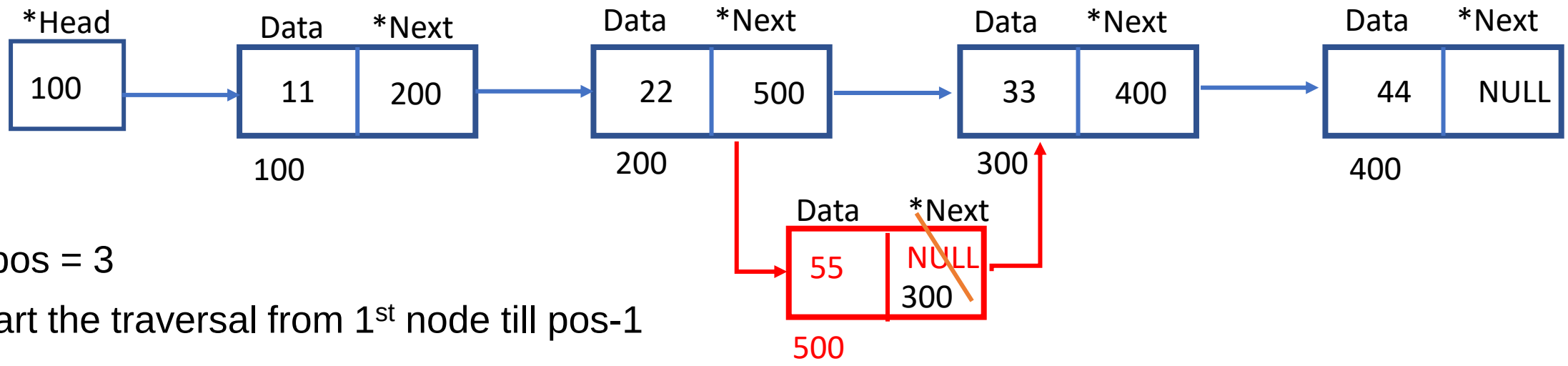
- If Head is NULL, Add the new node to the Head.



- To add a new node at the first position always create a new connection first between the new node and the already existing first node. Then change the address inside head pointer to point to the new node. So the Rule is MAKE BEFORE BREAK.



Singly Linear Linked List – Add at Specific position



If pos = 3

Start the traversal from 1st node till pos-1

Next,

Store address of current pos node into next part of newnode

newnode->next=trav->next

Store an address of newly created node into next part of (pos-1)th node

Trav->next = newnode;

Thank You !!