TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

Topics Covered

1. Linear Data Structure
    a. Linked List

Linked List

- Linked List
  - Definition
    - It is a linear data structure to store data of various types where one data is linked with the next data through a pointer
  - Property
    - Each data is associated with a link (or an address) to be connected with the next data
    - The association of data and link is termed as **NODE**
    - Each node can store single or multiple number and types of data but with a single link
    - First node is called Head or Start node whereas the last node is called as Tail node

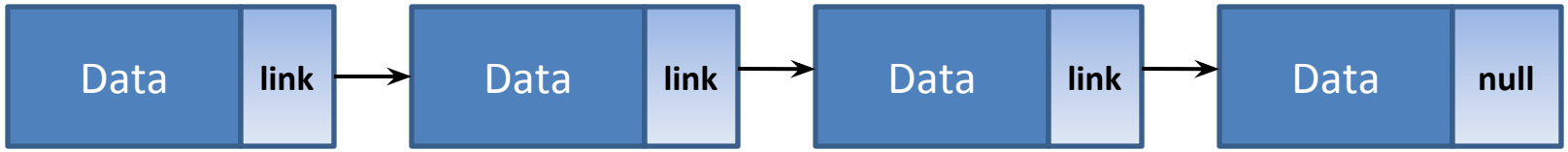
Linked List

- Linked List vs. Array
  - Advantage
    - Multiple types of data can be stored in a node but an array can store only one type of data in a cell
    - Multiple number of data can be stored in a node
    - Continuous memory location is not required to store data in a linked list
    - Data (or node) deletion process is easier and faster in a LL
    - Node can be inserted as per user demand, no prior information about the number of input is required

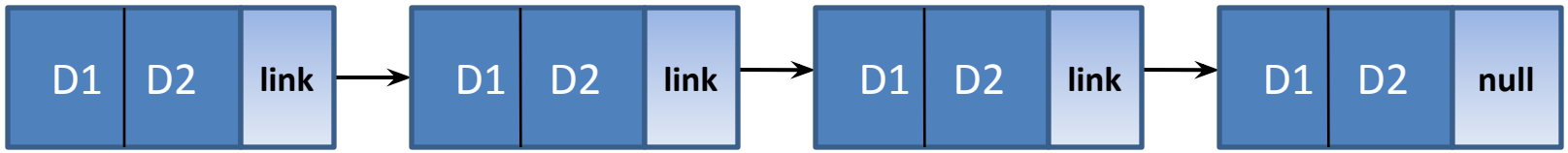
Linked List

- Representation
  - Single node linked list

  - Large list (with single data field)

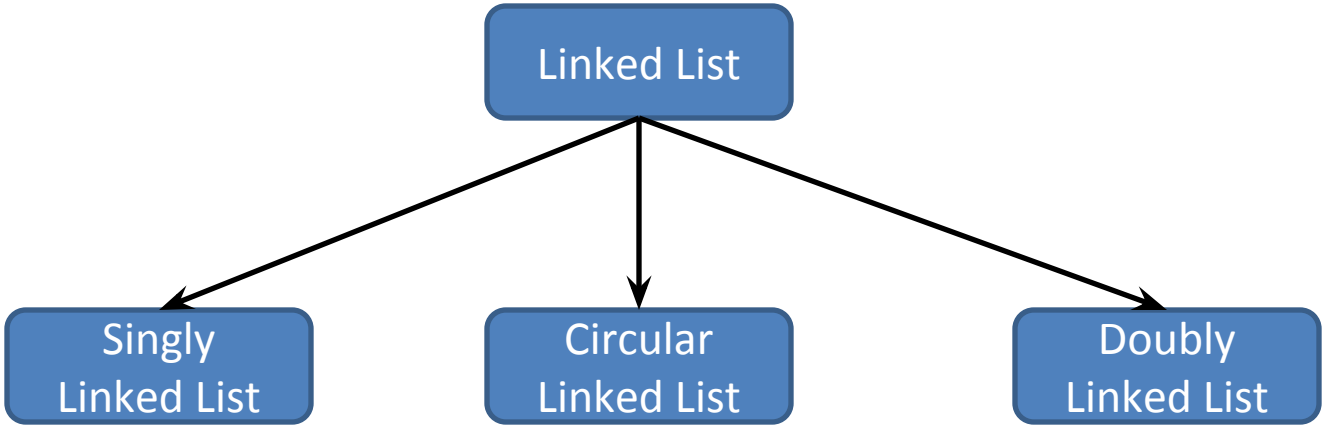


  - Large list (with multiple data field)



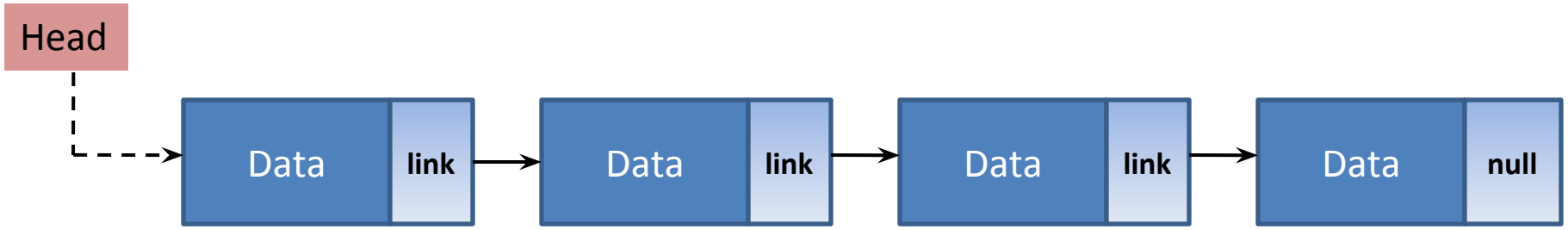
Linked List

- Variations



Linked List

- Singly Linked List
  - Property
    - It is an unidirectional linked list, conventionally data can be processed from left to right
    - Head pointer holds the first node of the list
    - The last node of the list holds **NULL** in the address part



Linked List

- Singly Linked List
  - Operations
    - Node insertion
      - Insertion of new data into the list along with a modification of address field
    - Node deletion
      - Deletion of a given data along with declining the link
    - Node searching
      - Searching for a given data in the list
      - No modification on address field is required

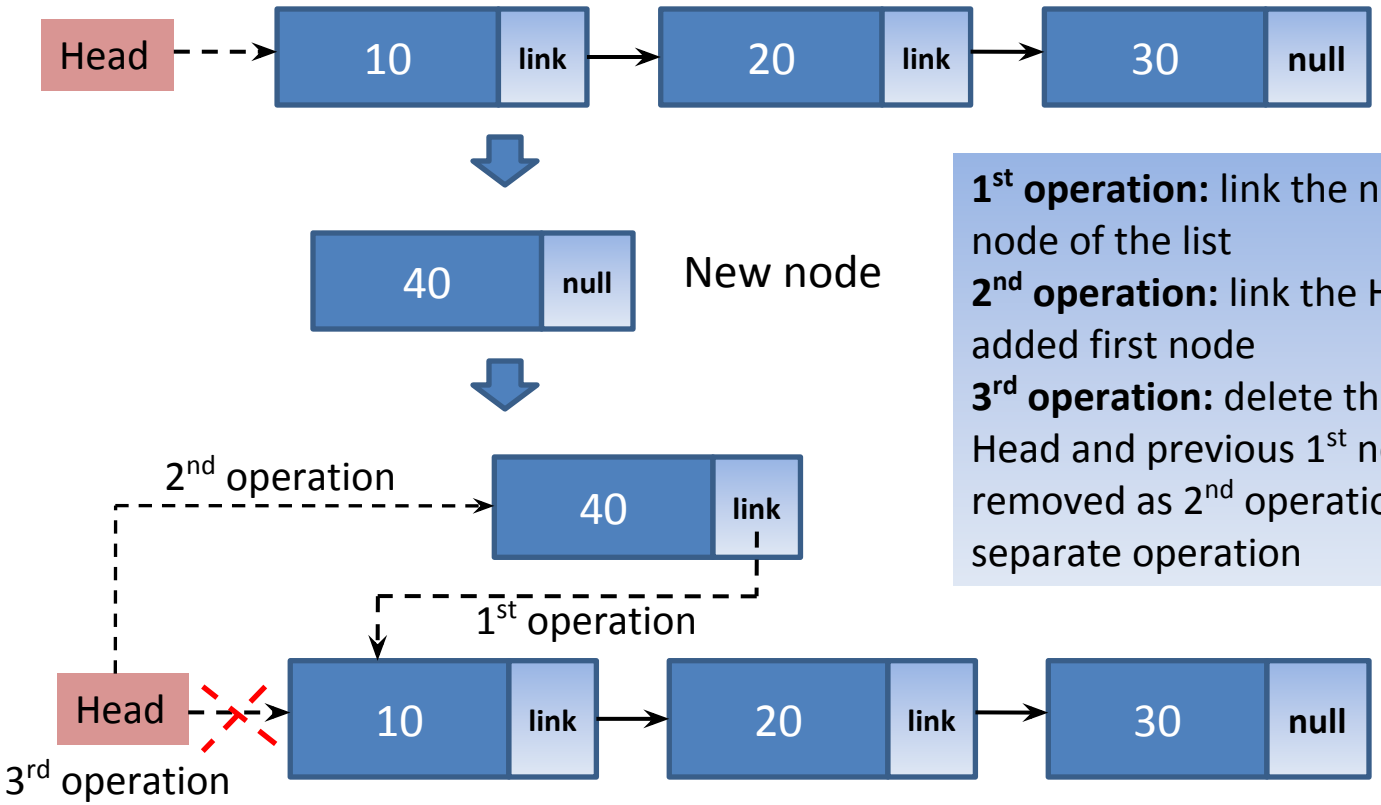
Linked List

- Singly Linked List

  - Node Insertion
    - Insert at the beginning
    - Insert at last
    - Insert in middle

Linked List

- Singly Linked List
  - Node Insertion at beginning of list



1st operation: link the new node with 1st node of the list
2nd operation: link the Head with newly added first node
3rd operation: delete the link between Head and previous 1st node. It is removed as 2nd operation is done, not a separate operation

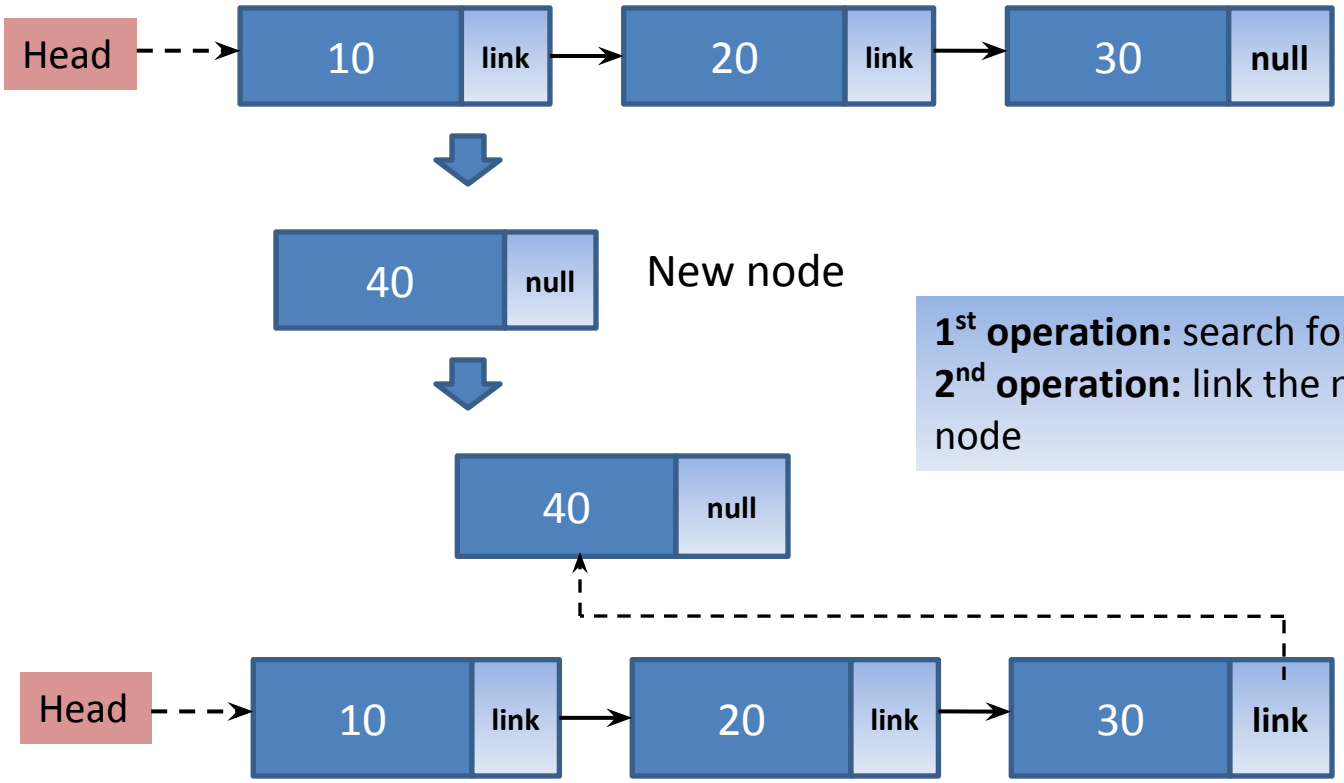
Linked List

- Singly Linked List
  - Node Insertion at beginning of list (algorithm)

```
Insert_beginning(LL, N)  // N is new node to insert in existing list LL
{
  if LL = null
     Head := N
  else
     N->next := Head      // N->next indicates the address part of N
     Head := N
}
```

Linked List

- Singly Linked List
  - Node Insertion at end of list



1st operation: search for the last node
2nd operation: link the new node as last node

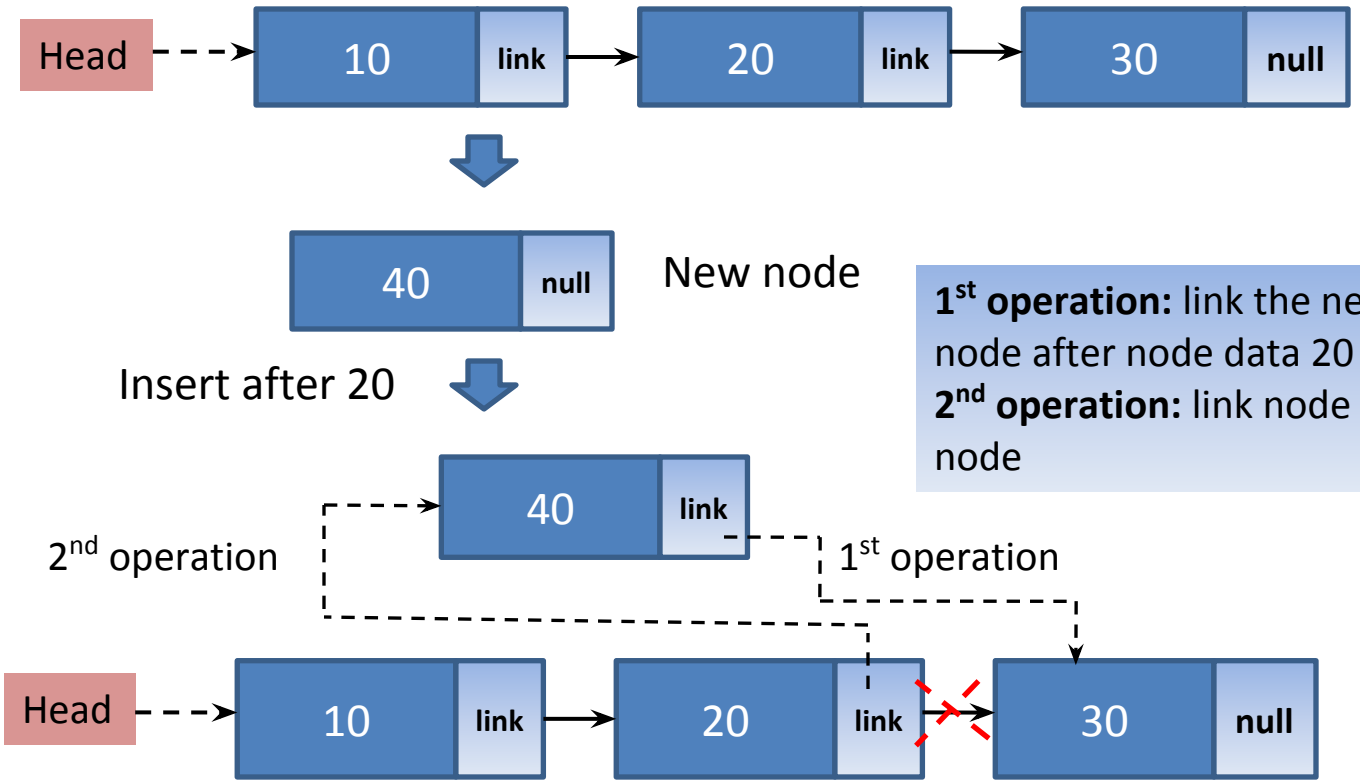
Linked List

- Singly Linked List
  - Node Insertion at end of list (algorithm)

```
Insert_end(LL, N)  // N is new node to insert in existing list LL
{
  if LL = null
     Head := N
  else
     set PTR := Head          // PTR is a pointer set to Head of the list
     while PTR->next not equal to null   // searching for the
        set PTR := PTR -> next              //  last node
     PTR->next := N
}
```

Linked List

- Singly Linked List
  - Node Insertion in middle of list



1st operation: link the new node with 1st node after node data 20
2nd operation: link node 20 with new node

Linked List

- Singly Linked List
  - Node Insertion in middle of list (algorithm)

```
Insert_middle(LL, N, X)  // N is new node to insert after a node having data 20
{
  if LL = null
      Head := N
  else
     set PTR := Head          // PTR is a pointer set to Head of the list
      while PTR->data not equal to X   // searching for specific node
         set PTR := PTR -> next
      N->next := PTR->next
      PTR->next := N
}
```

Queries?