TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

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

Topics Covered

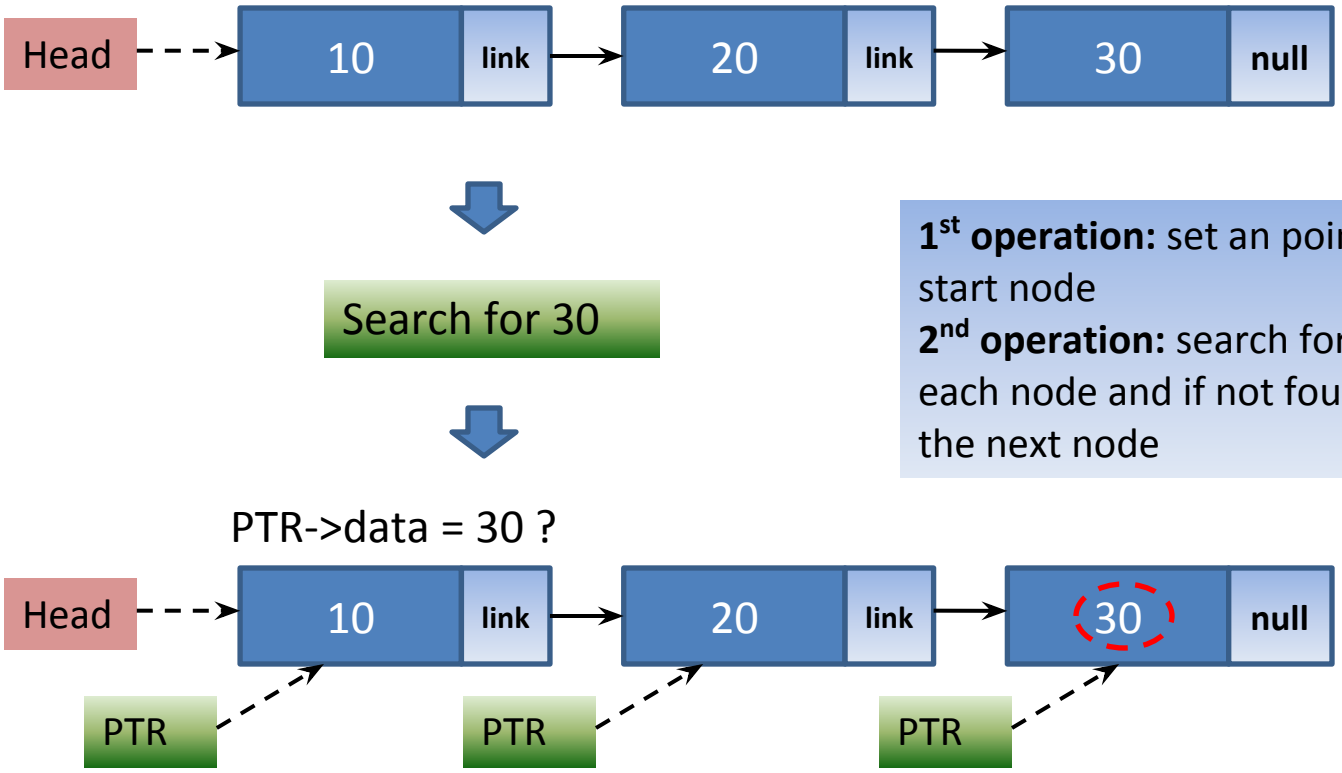
1. Singly Linked List
    a. Deletion
    b. Display
    c. Search
2. Circular Linked List
    a. Data insertion

Linked List

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

Linked List

- Singly Linked List
  - Searching data in the list

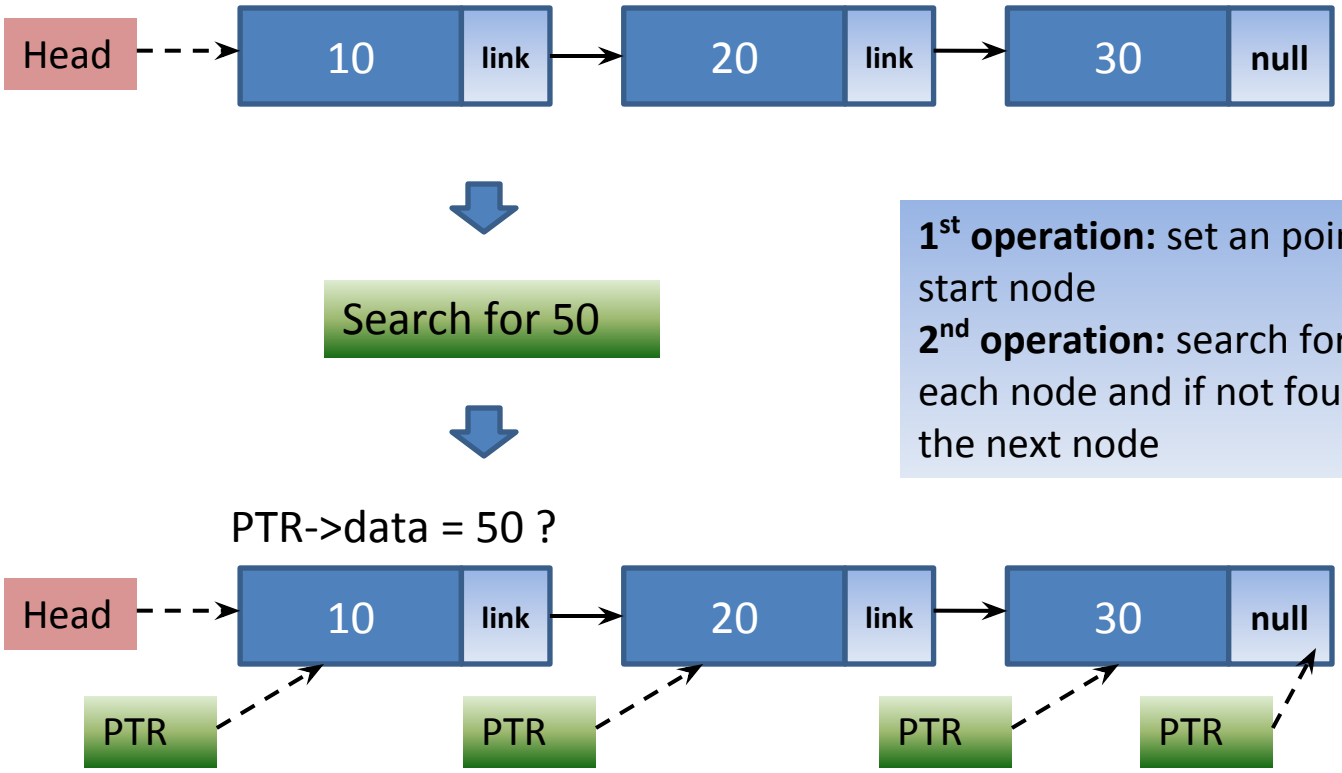


1st operation: set an pointer PTR to the start node
2nd operation: search for the data in each node and if not found traverse to the next node

Search successful

Linked List

- Singly Linked List
  - Searching data in the list



1st operation: set an pointer PTR to the start node
2nd operation: search for the data in each node and if not found traverse to the next node

Un-successful search

Linked List

- Singly Linked List
  - Searching data in the list (algorithm)

```
Searching(LL, X)  // X is the stored data which node has to be searched
{
  if LL = null
     Print "Empty list" and return
  else
     set PTR := Head          // PTR is a pointer set to Head of the list
     while PTR not equal to null   // traversing the entire list
          if PTR -> data = X
             Print "Data found" and return
          PTR := PTR->next
     Print "Data not found"
}
```

Linked List

- Singly Linked List
  - Display the list (algorithm)

```
Searching(LL)  // LL is the existing list
{
  if LL = null
     Print "Empty list" and return
  else
     set PTR := Head          // PTR is a pointer set to Head of the list
     while PTR not equal to null   // traversing the entire list
              Print PTR->data
              PTR := PTR->next
}
```

Linked List

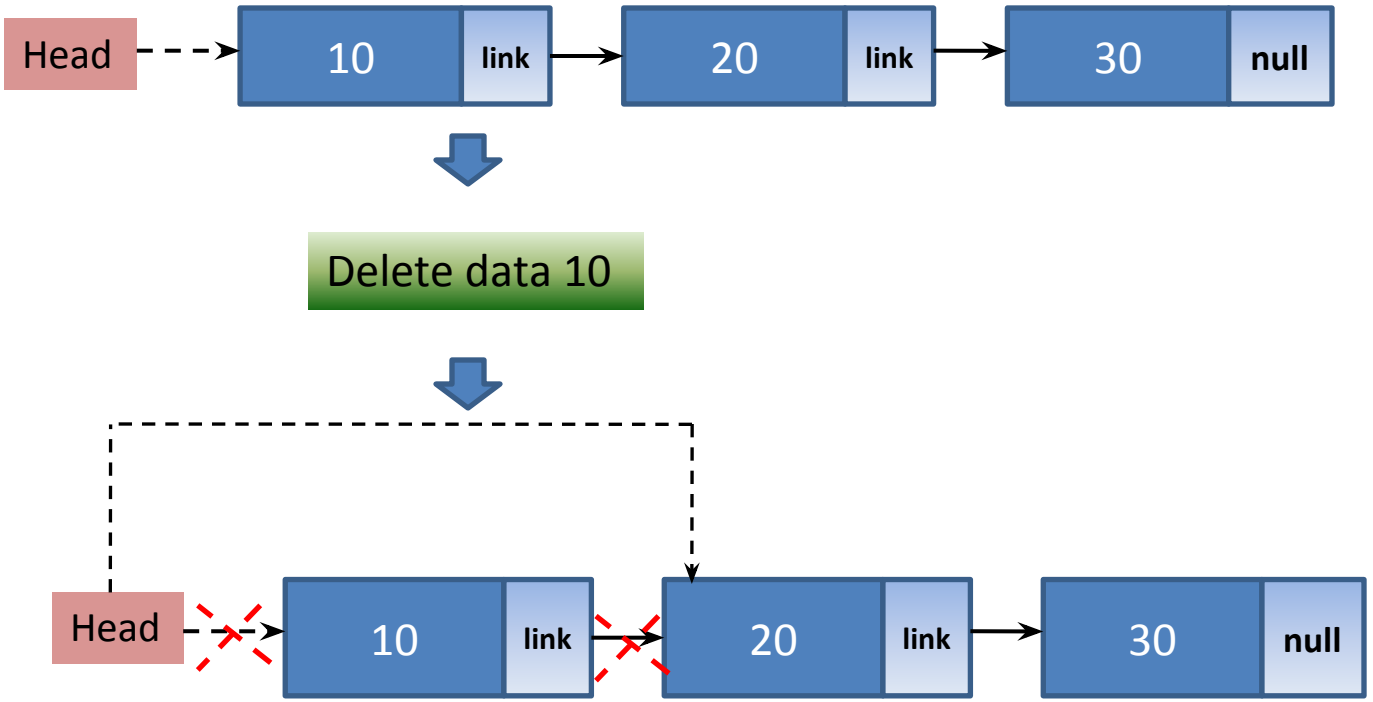
- Singly Linked List

  - Node Deletion
    - Delete start node
    - Delete last node
    - Delete node from middle

During implementation, one needs to merge all conditions in a single code

Linked List

- Singly Linked List
  - Delete first node



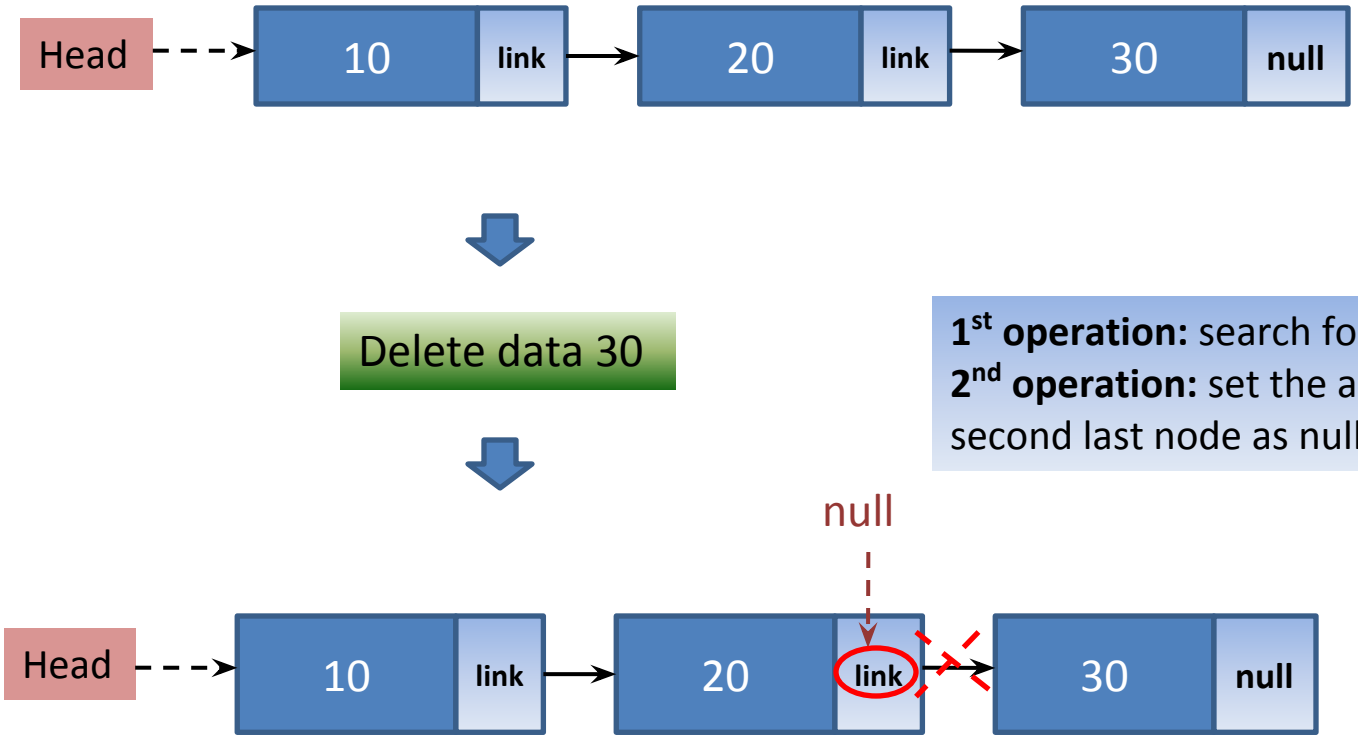
Linked List

- Singly Linked List
  - Delete first node (algorithm)

```
Delete_beginning(LL)  // LL is the existing list
{
  if Head->next = null
      Head := null
  else
      Head := Head ->next
}
```

Linked List

- Singly Linked List
  - Delete last node



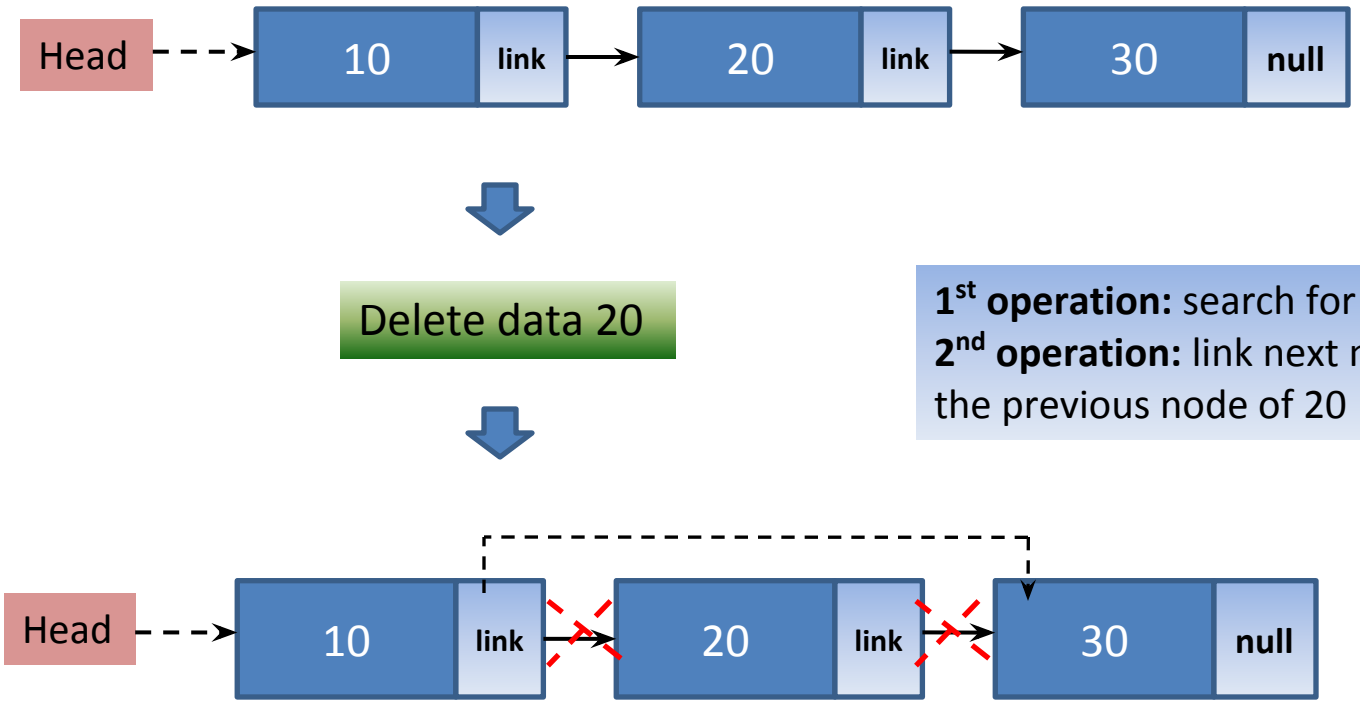
Linked List

- Singly Linked List
  - Deletion of last node (algorithm)

```
Delete_end(LL)  // LL is the existing linked list
{
  if Head->next = null
     Head := null
  else
     set PTR := Head          // PTR is a pointer set to Head of the list
     while PTR->next->next not equal to null   // searching for the
        set PTR := PTR -> next                 //  second last node
     PTR->next := null
}
```

Linked List

- Singly Linked List
  - Node deletion from middle of list



1st operation: search for data 20
2nd operation: link next node of 20 with the previous node of 20

Linked List

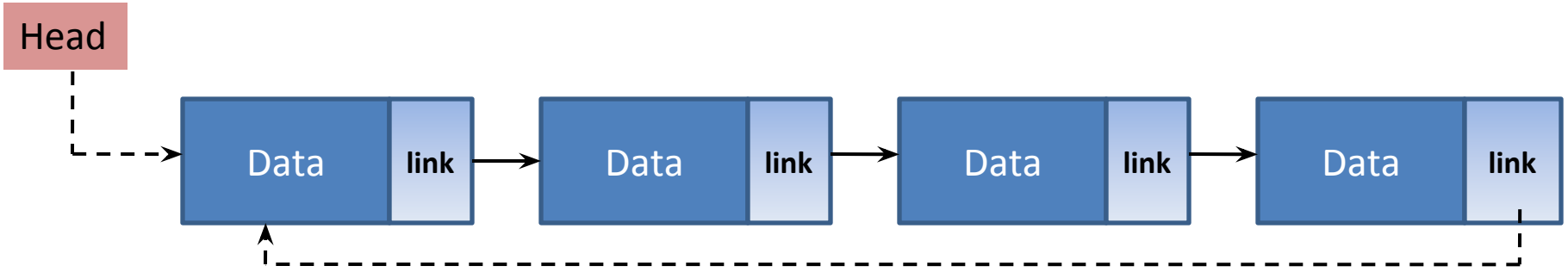
- Singly Linked List
  - Node deletion from middle of list (algorithm)

```
Delete_middle(LL, X)  // X is the stored data which node has to be deleted
{
  if Head->next = null
     Head := null
  else
     set PTR := Head          // PTR is a pointer set to Head of the list
      while PTR->next->data not equal to X   // searching for the previous node
         set PTR := PTR -> next                    // of the node to be deleted
      PTR->next := PTR->next->next
}
```

Circular Linked List

Linked List

- Circular 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 the address of the first node



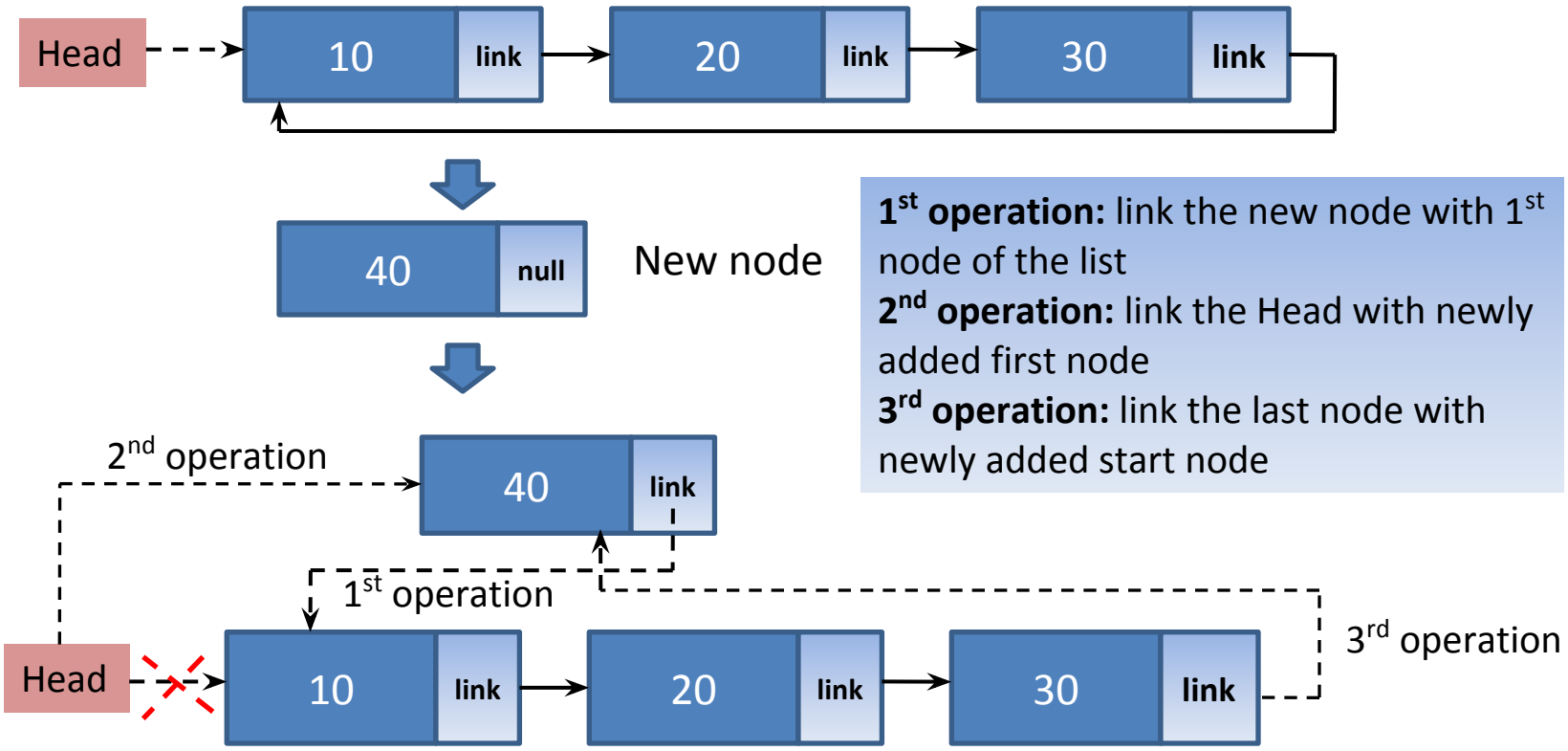
Linked List

- Circular Linked List

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

Linked List

- Circular 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: link the last node with newly added start node

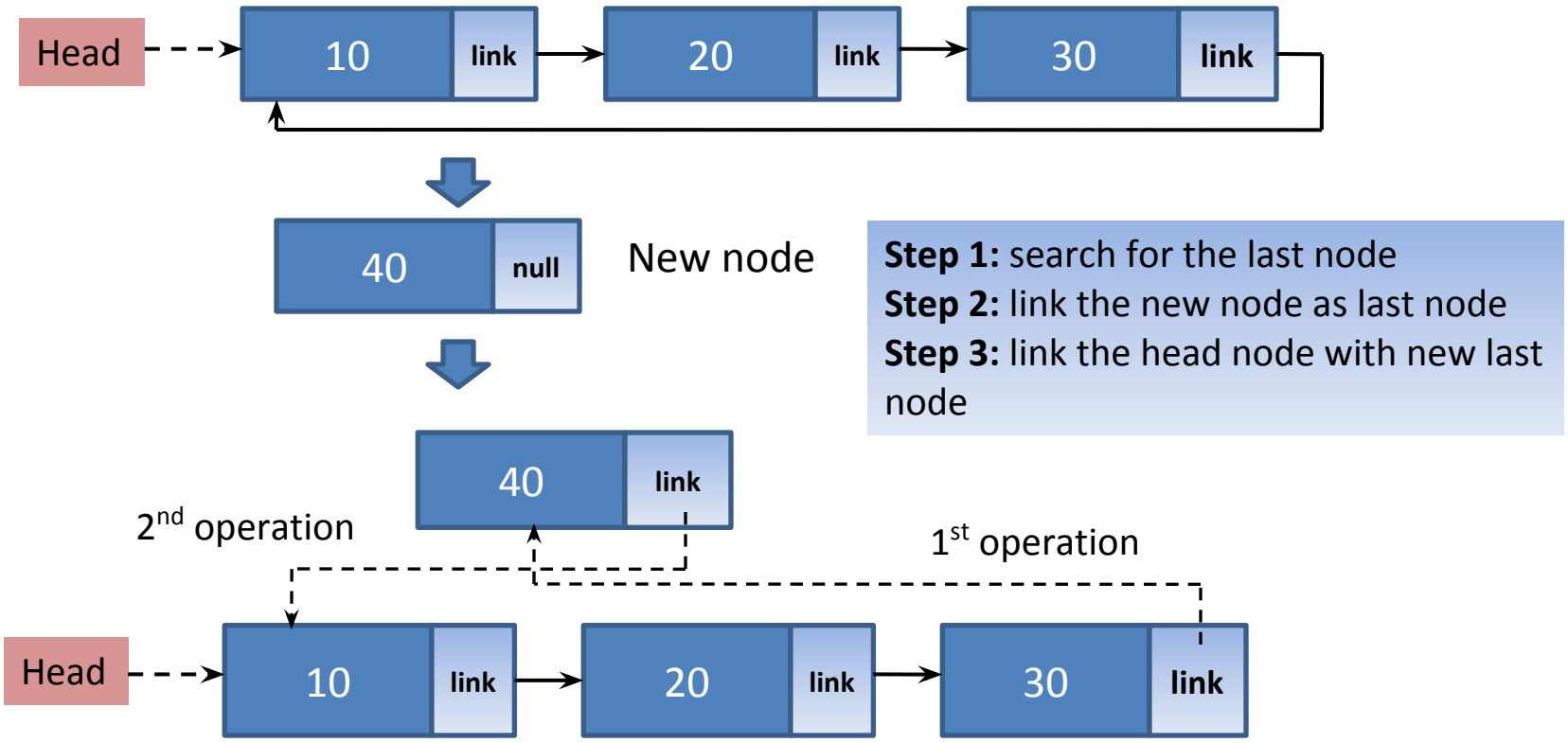
Linked List

- Circular 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
    set PTR :=Head
    while PTR->next not equal to Head   // set PTR at last node
         PTR:= PTR->next
    N->next := Head      // N->next indicates the address part of N
    Head := N
    PTR->next := Head
}
```

Linked List

- Circular Linked List
  - Node Insertion at end of list



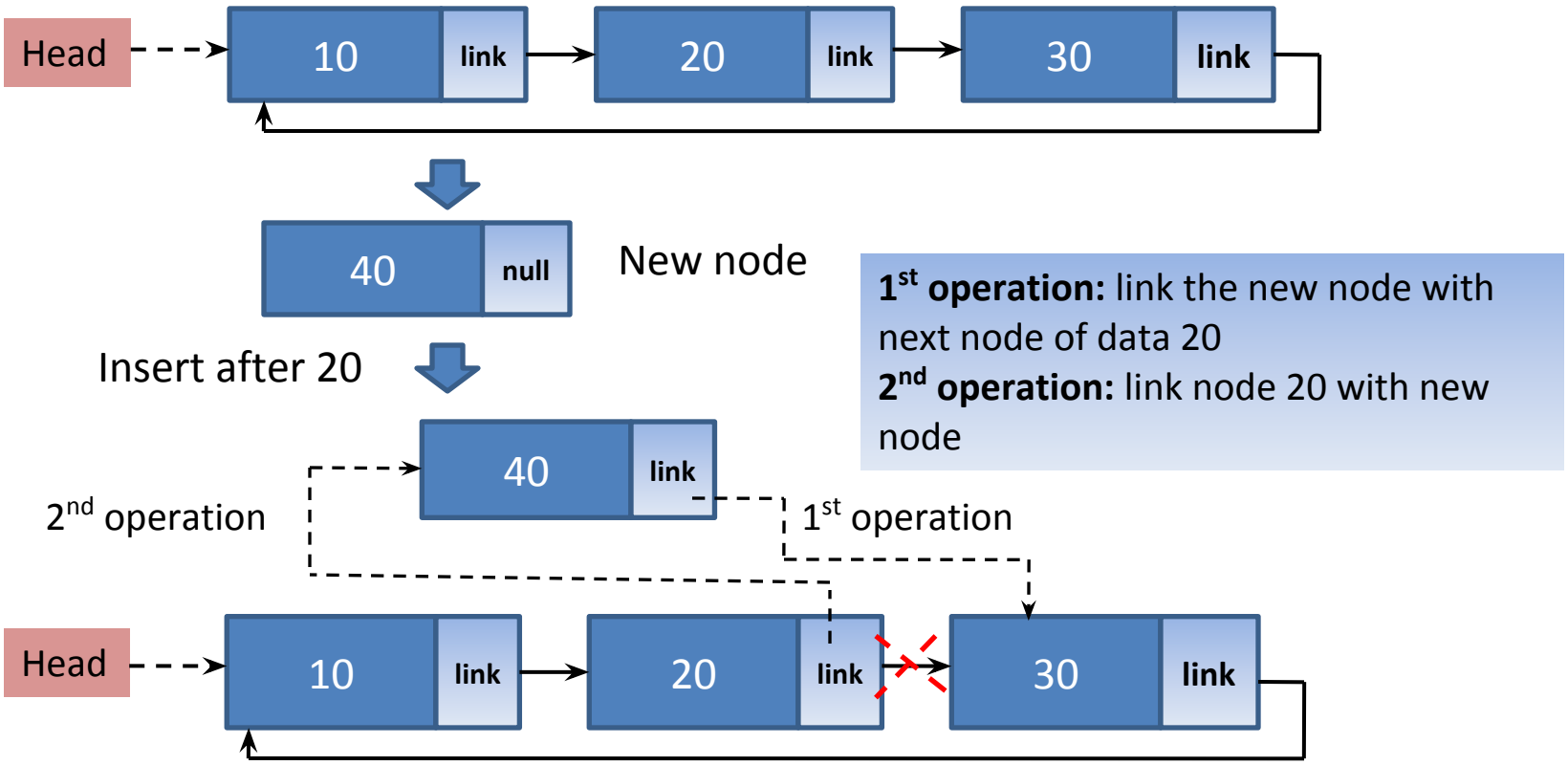
Linked List

- Circular 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
    while PTR->next not equal to Head   // set PTR at last node
         PTR:= PTR->next
    PTR->next := N
    N->next := Head
}
```

Linked List

- Circular Linked List
  - Node Insertion in middle of list



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

Linked List

- Circular 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?