# Data Structures – CST 201
# Module - 3

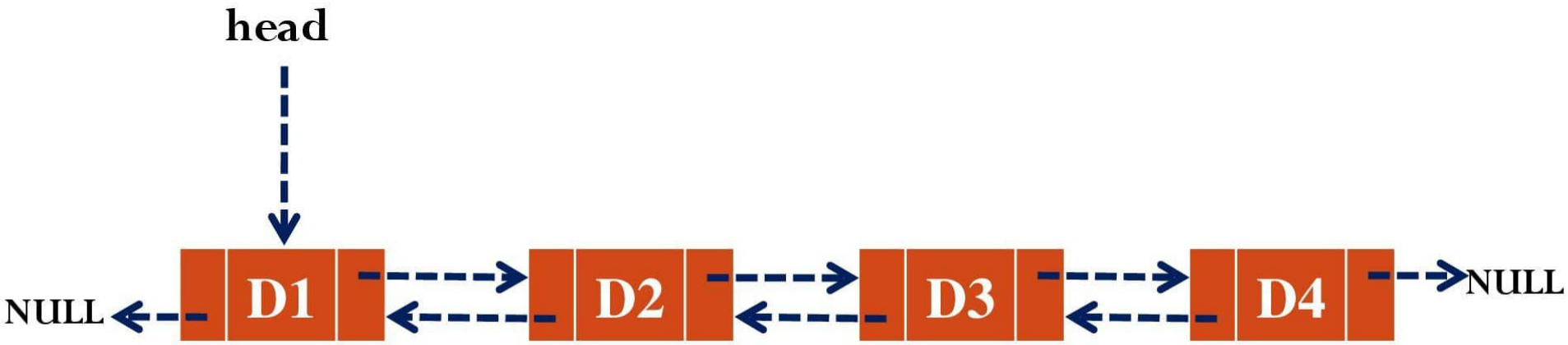
# Syllabus

- **Linked List and Memory Management**
  - Self Referential Structures
  - Dynamic Memory Allocation
  - Singly Linked List-Operations on Linked List.
  - **Doubly Linked List**
  - Circular Linked List
  - Stacks using Linked List
  - Queues using Linked List
  - Polynomial representation using Linked List
  - Memory allocation and de-allocation
    - First-fit, Best-fit and Worst-fit allocation schemes

# Doubly Linked List

- **Singly linked list is a one-way list:** List can be traversed in one direction only
- **Doubly linked list is a two-way list:**
  - List can be traversed in two directions
  - Each node is divided in to three parts
    - **data**
    - **Llink:** pointer to the previous node in the list
    - **Rlink:** pointer to the next node in the list

# Doubly Linked List



# Node Creation

**Algorithm struct node**

1. Declare int data, node Llink, node Rlink

Program


```
struct node
{       int data;
        struct node *Llink,*Rlink;
};
void main()
{       struct node *ptr;
        ptr = (struct node *)malloc(sizeof( struct node ));
}
```

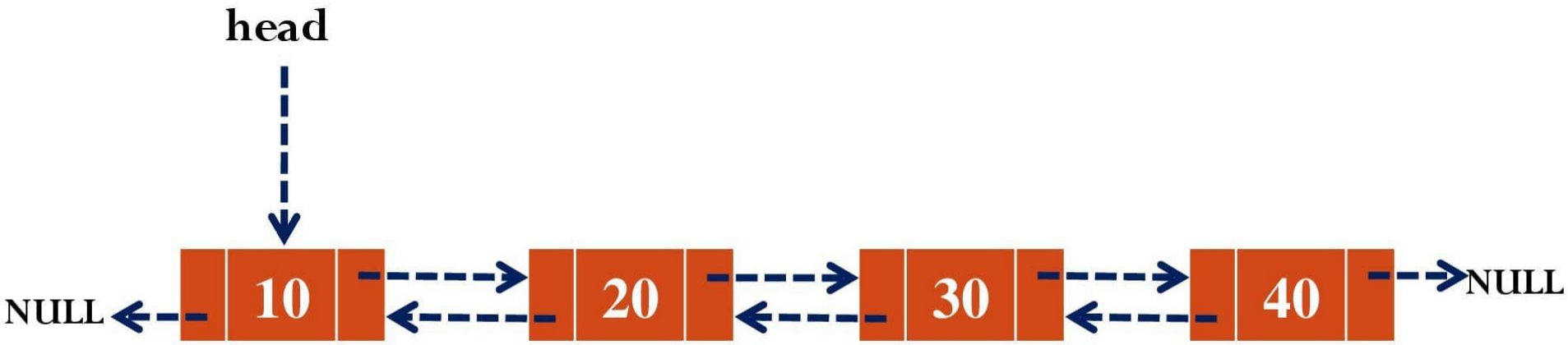
# Operations on Doubly Linked List

- Traverse/Display a list
- Insertion of a node into list
  - Insert at front
  - Insert at end
  - Insert after a specified node
- Deletion of node from list
  - Delete from front
  - Delete from end
  - Delete a specified node
- Searching for an element in a list
- Merging two linked list into larger list

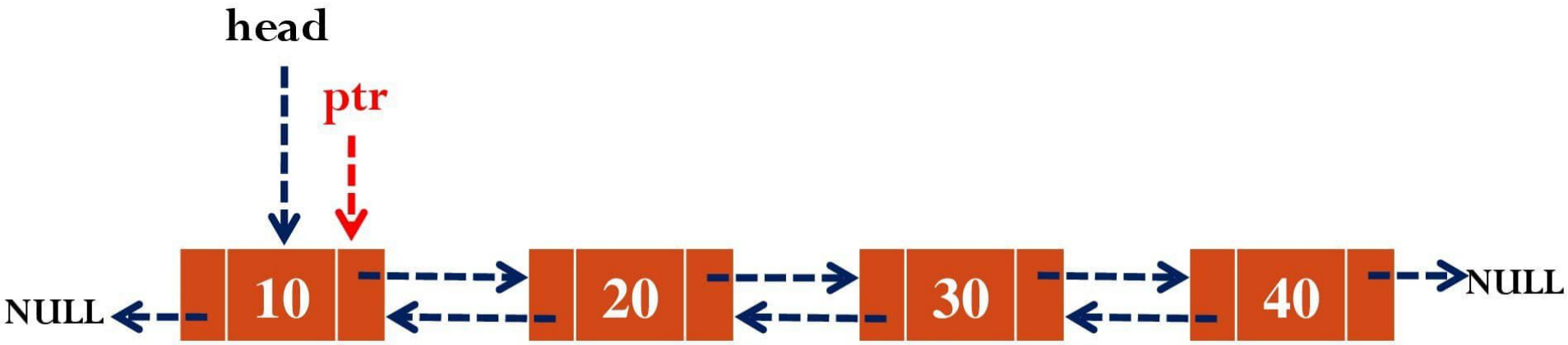
# Traversal/Display

- Visit every node in the list starting from the first node to the last one

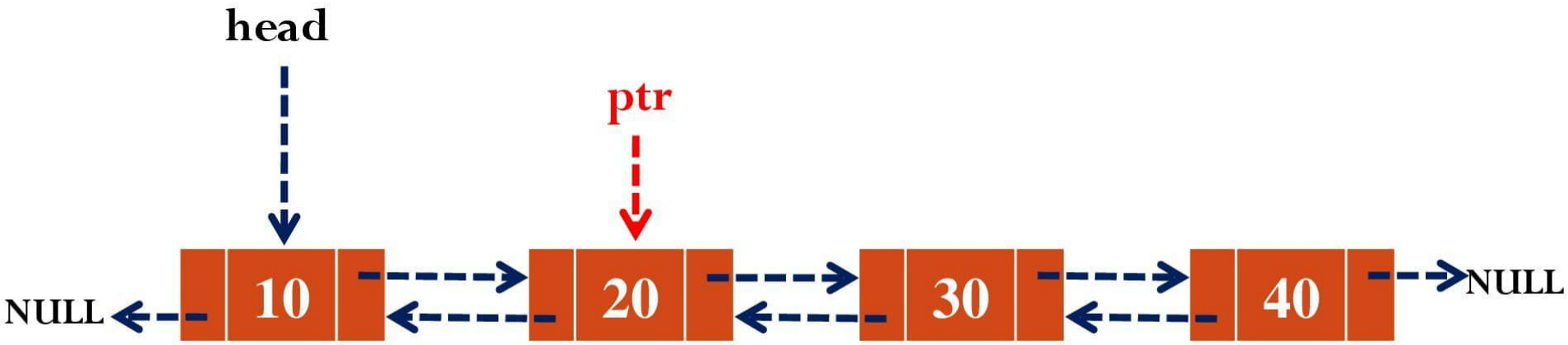
# Display/Traversal



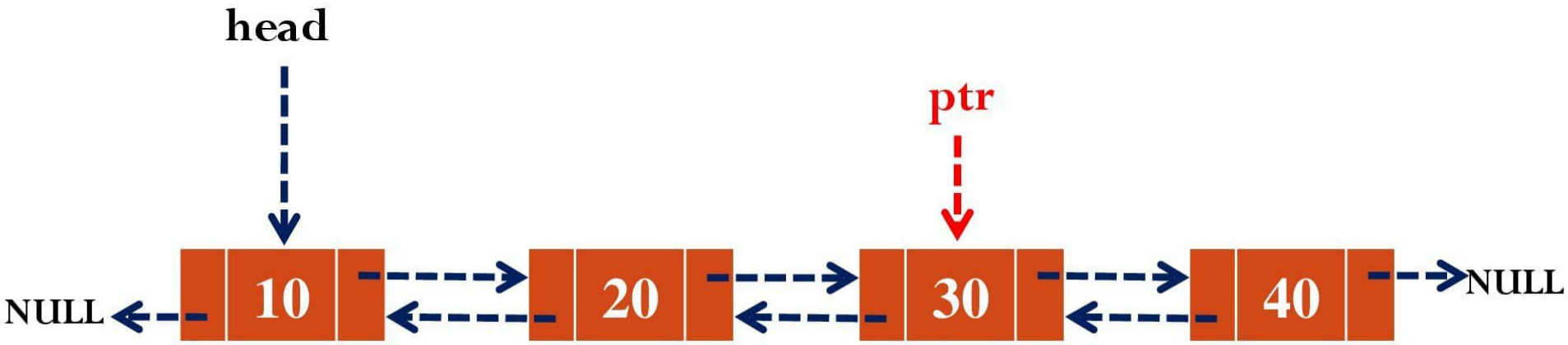
# Display/Traversal



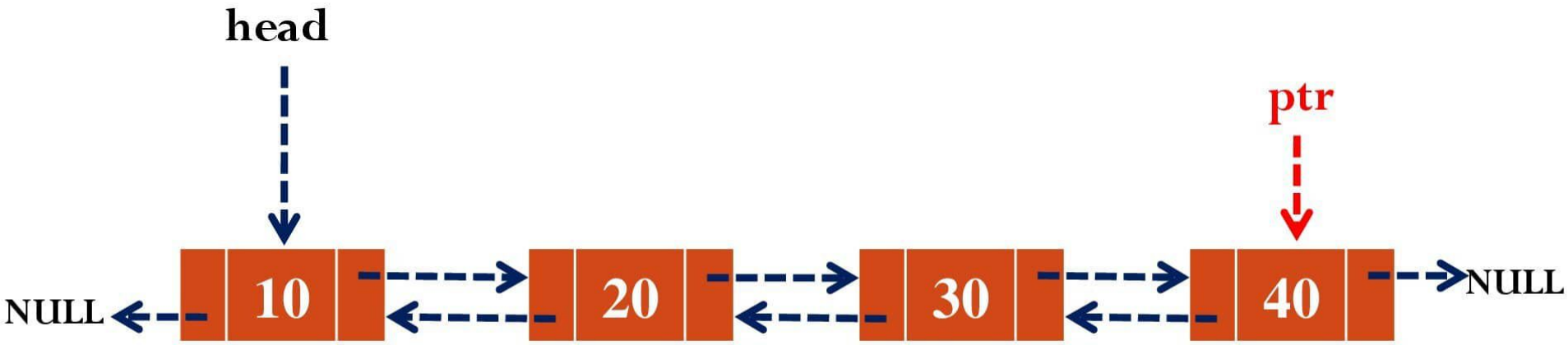
# Display/Traversal



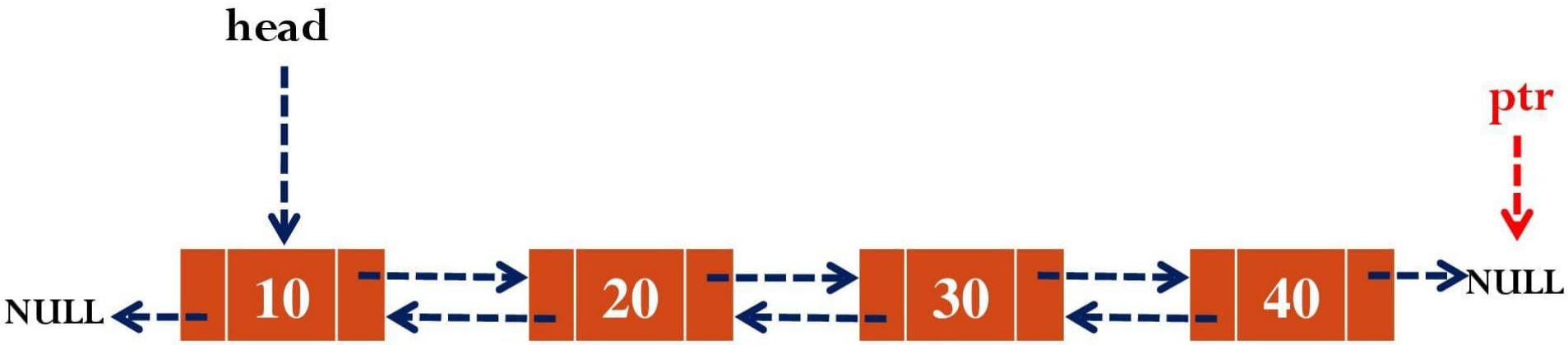
# Display/Traversal



# Display/Traversal



# Display/Traversal



# Traversal/Display Algorithm

**Algorithm Display(head)**

1. ptr=head
2. While ptr!=NULL do
    1. Print ptr→data
    2. ptr=ptr→Rlink

# Insertion

1. Insert at Front
2. Insert at End
3. Insert after a specified node

# Insertion

1. **Insert at Front**
2. Insert at End
3. Insert after a specified node

# Insert at Front

2 Cases

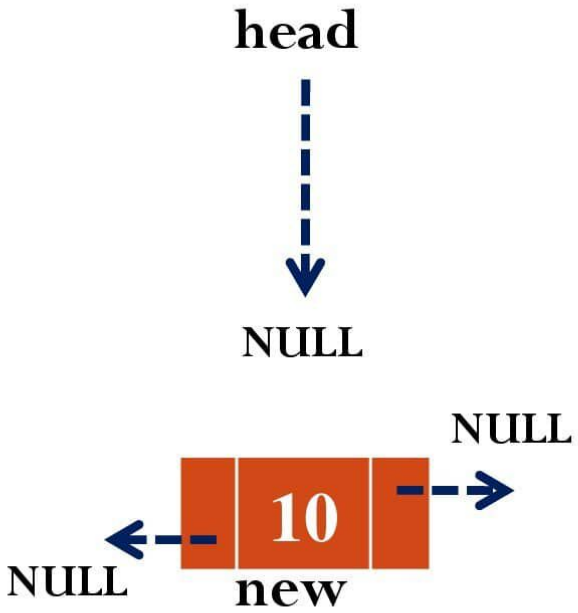
1. List is Empty
2. List is not Empty

# Insert at Front - Algorithm

Case 1

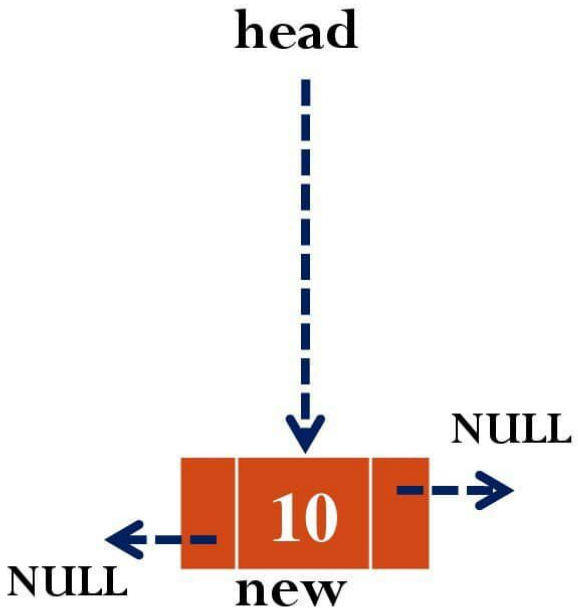
# Insert at Front - Algorithm

Case 1



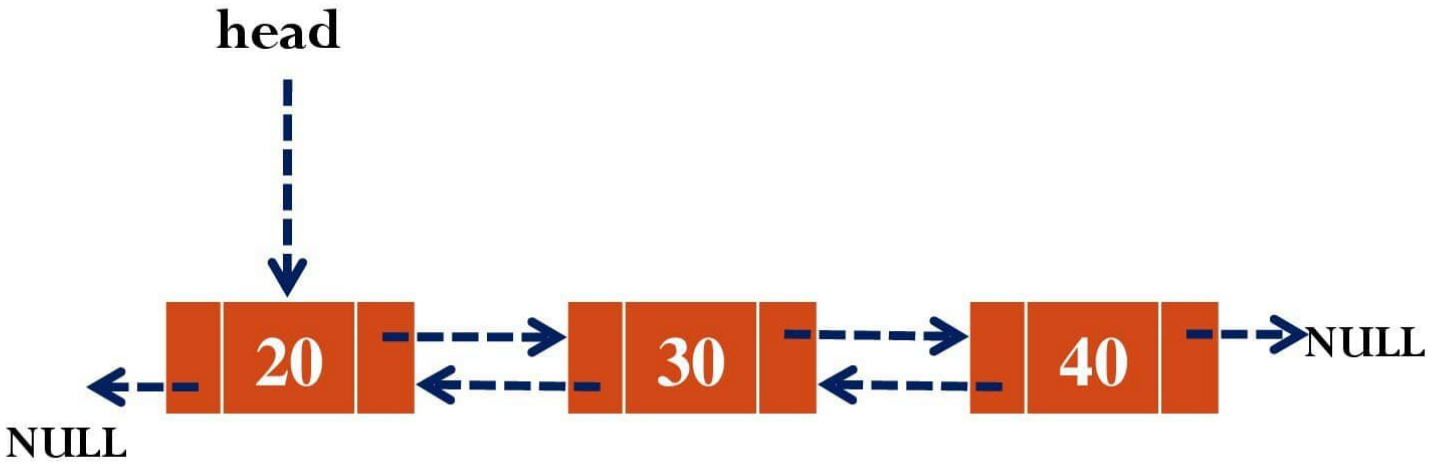
# Insert at Front - Algorithm

Case 1



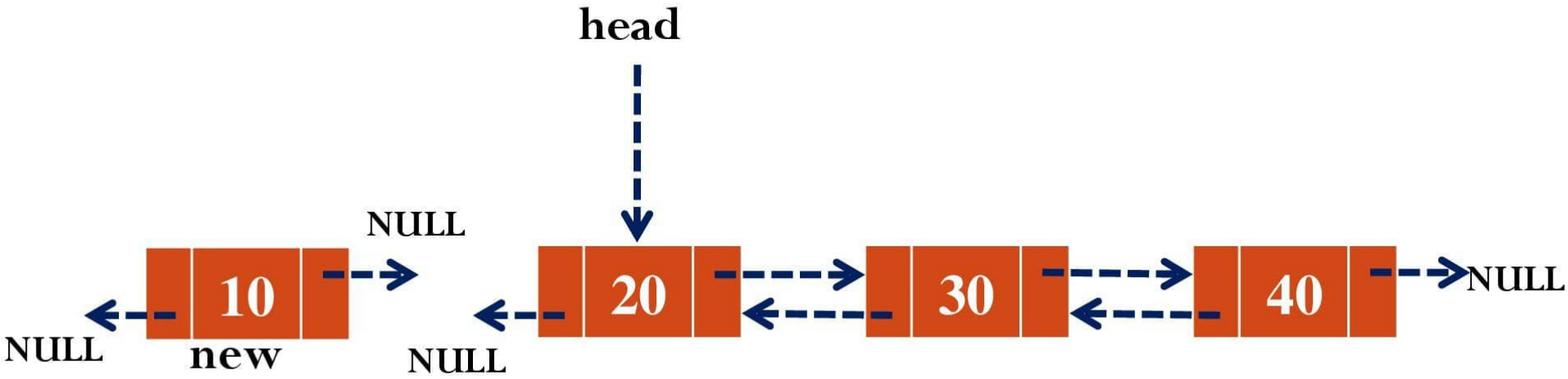
# Insert at Front - Algorithm

Case 2



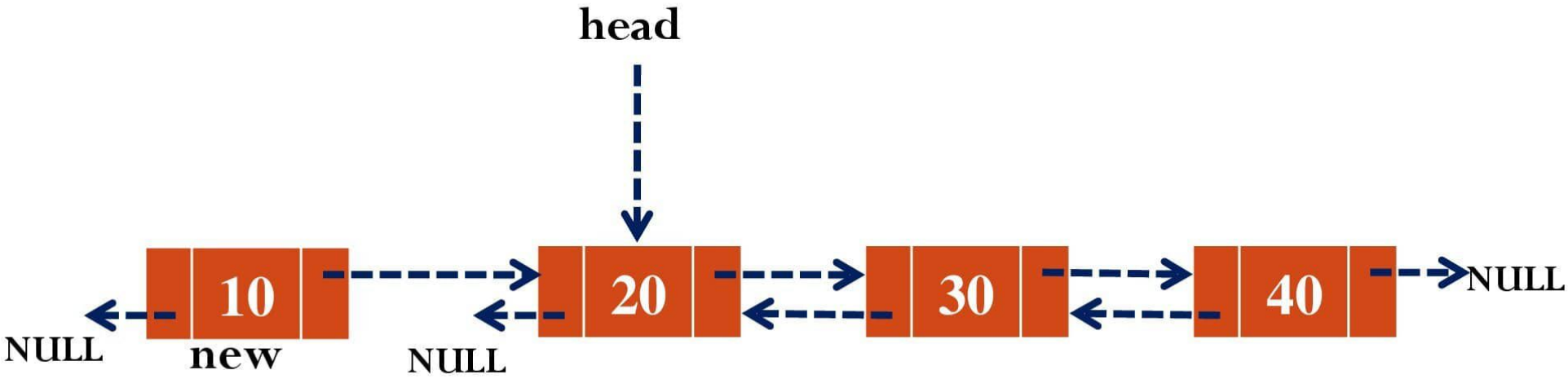
# Insert at Front - Algorithm

Case 2



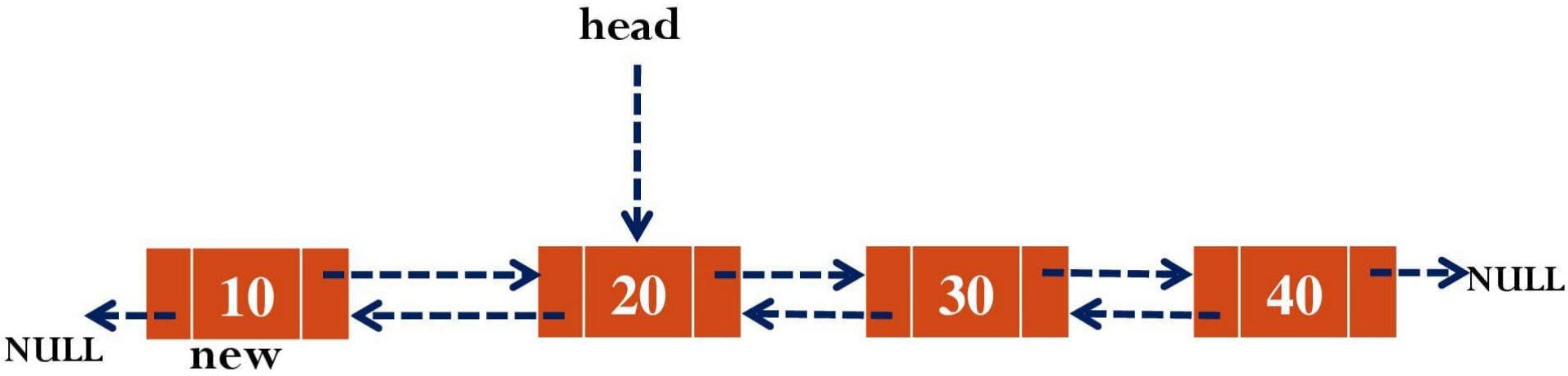
# Insert at Front - Algorithm

Case 2



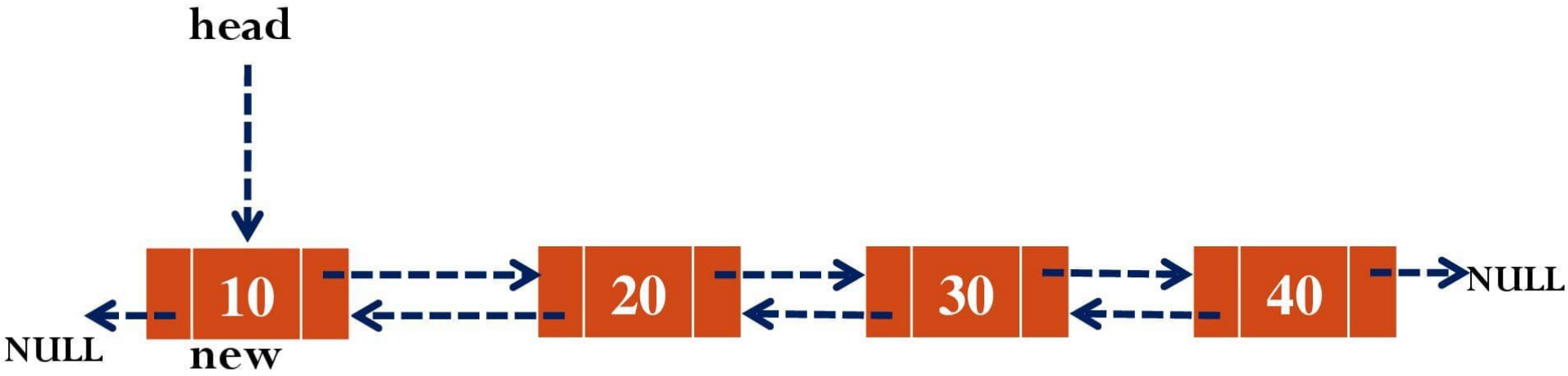
# Insert at Front - Algorithm

Case 2



# Insert at Front - Algorithm

Case 2



# Insert at Front - Algorithm

**Algorithm Insert_Front(head, x)**

1. Create a node new
2. new→data=x
3. new→Llink=new→Rlink=NULL
4. If head=NULL then
    1. head=new
5. Else
    1. new→Rlink=head
    2. head→Llink=new
    3. head=new

# Insertion

1. Insert at Front
2. **Insert at End**
3. Insert after a specified node

# Insert at End

2 Cases

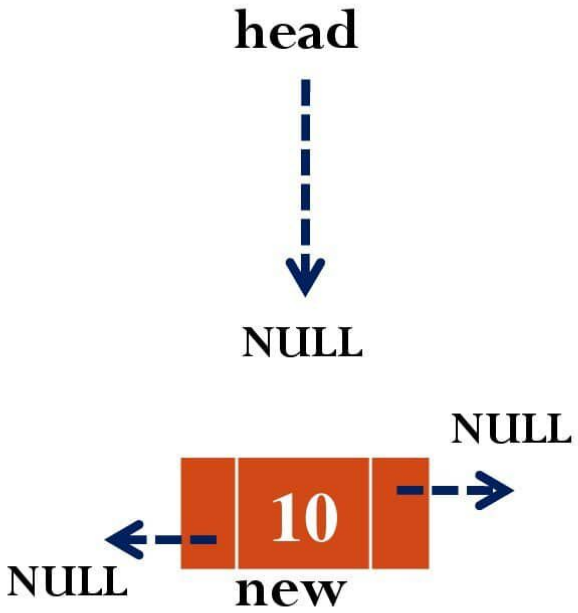
1. List is Empty
2. List is not Empty

# Insert at End - Algorithm

Case 1

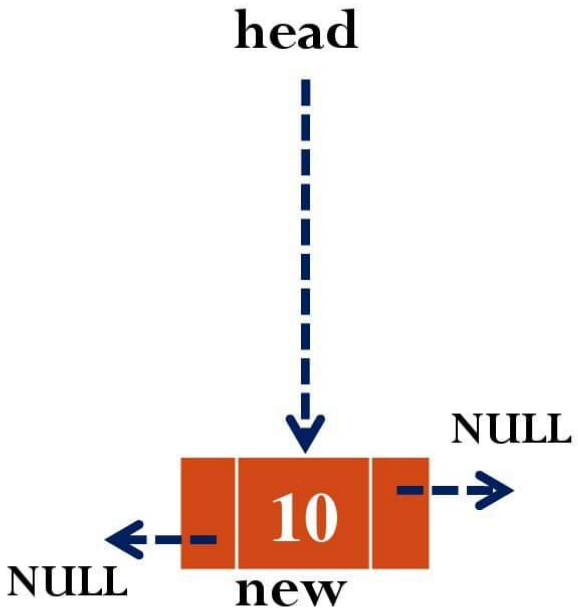
# Insert at End - Algorithm

Case 1



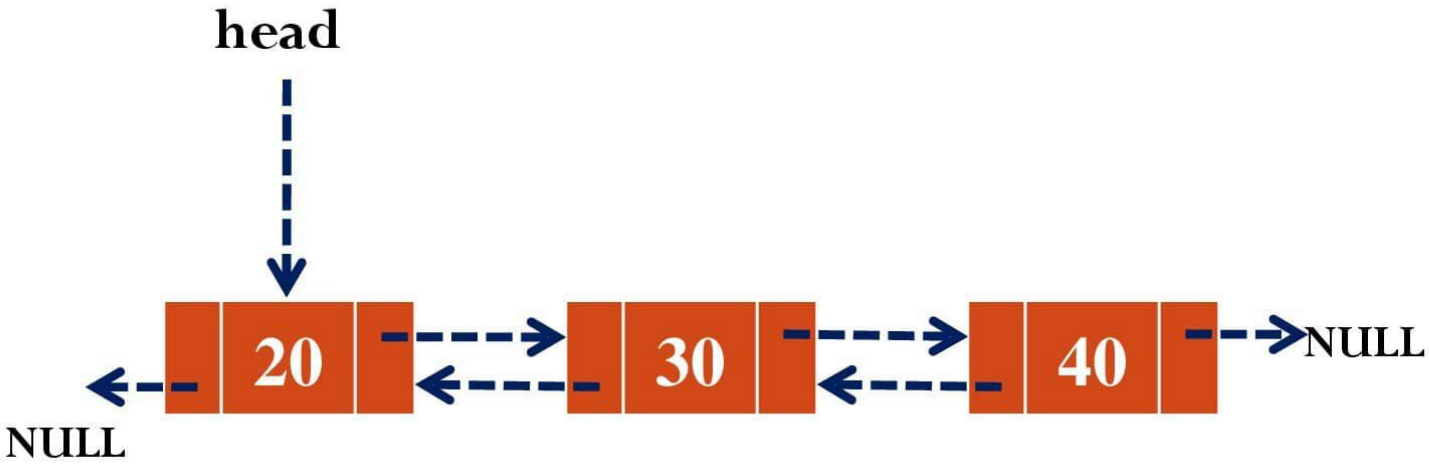
# Insert at End - Algorithm

Case 1



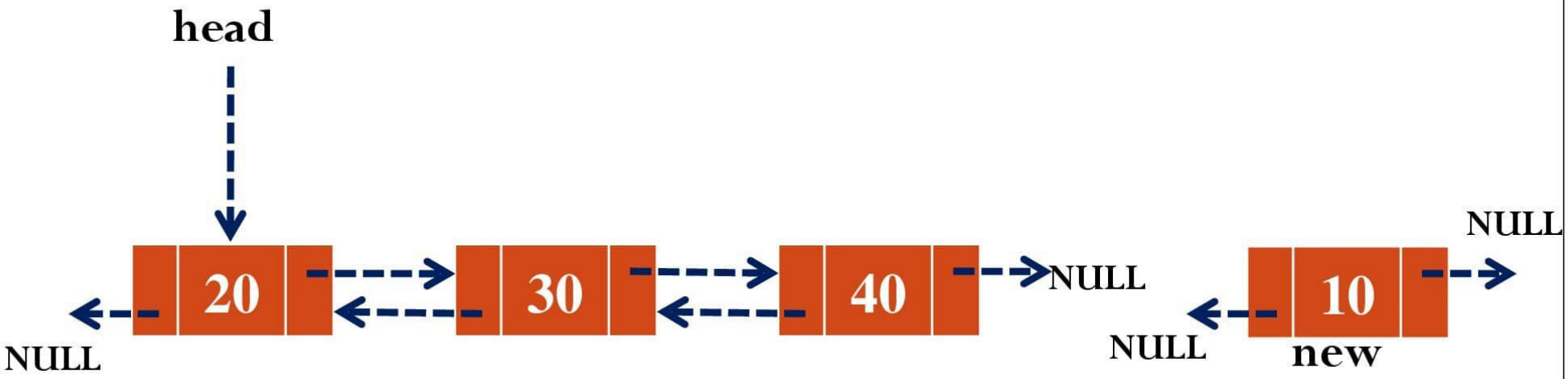
# Insert at End - Algorithm

Case 2



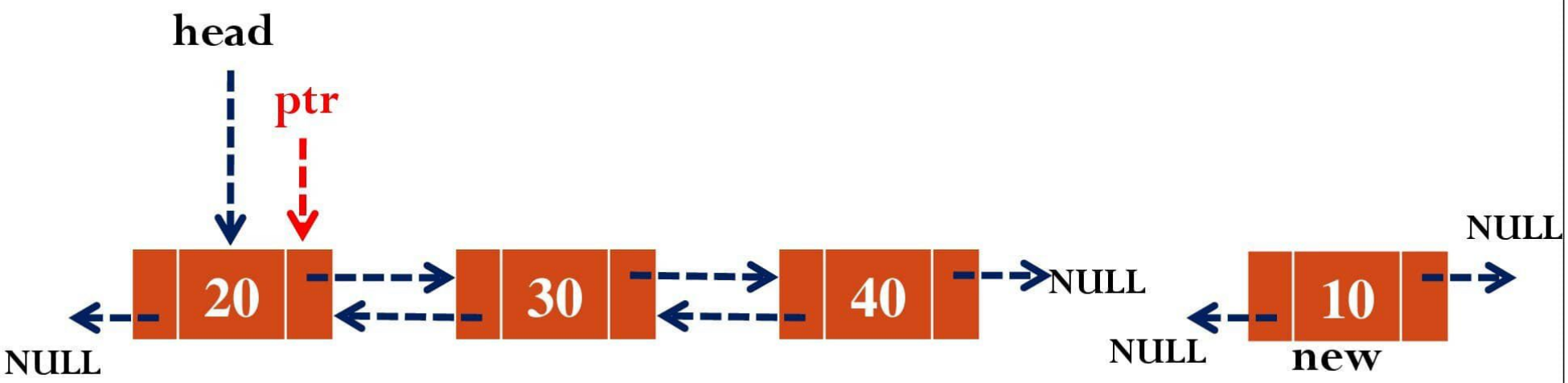
# Insert at End - Algorithm

Case 2



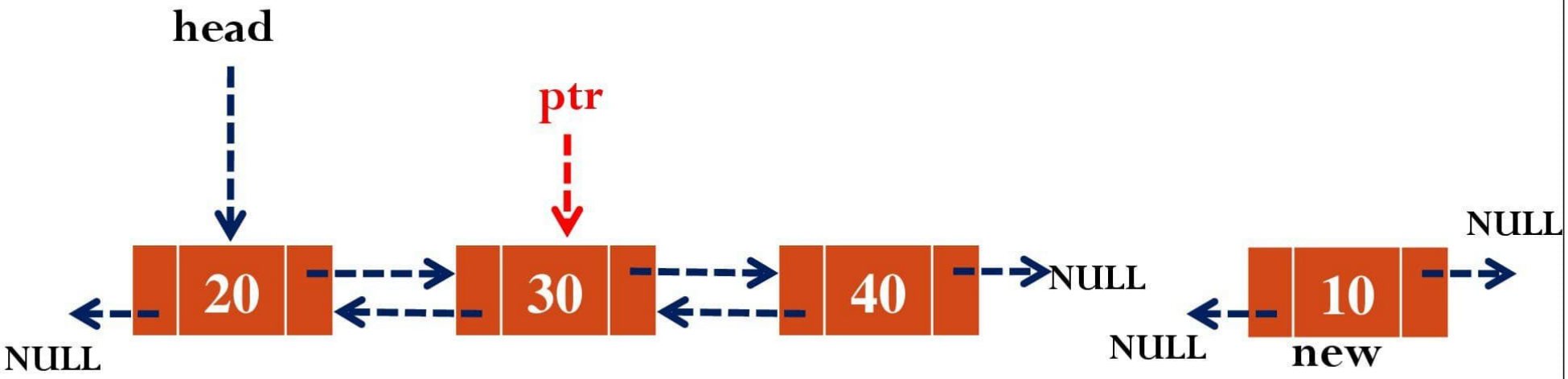
# Insert at End - Algorithm

Case 2



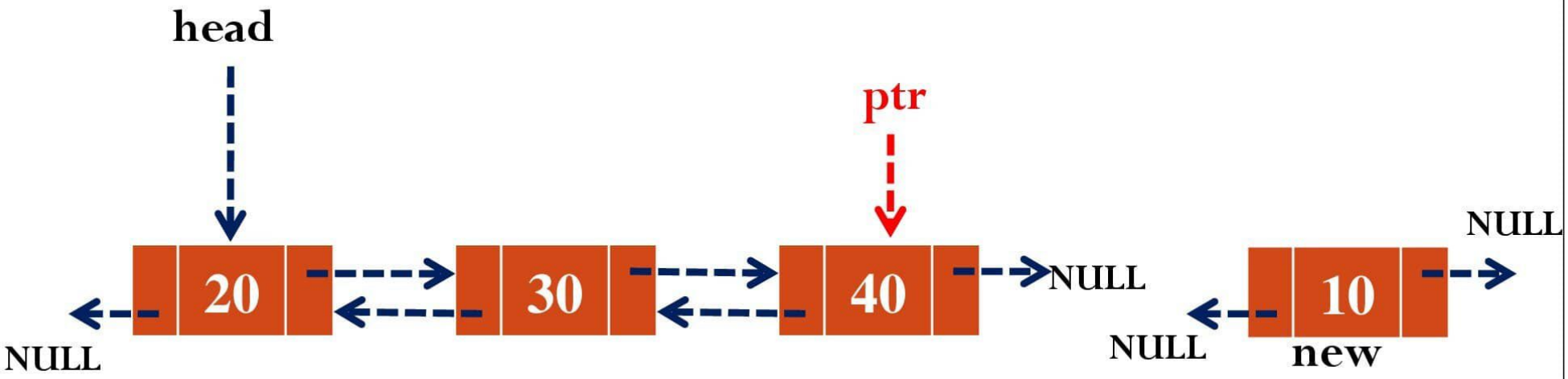
# Insert at End - Algorithm

Case 2

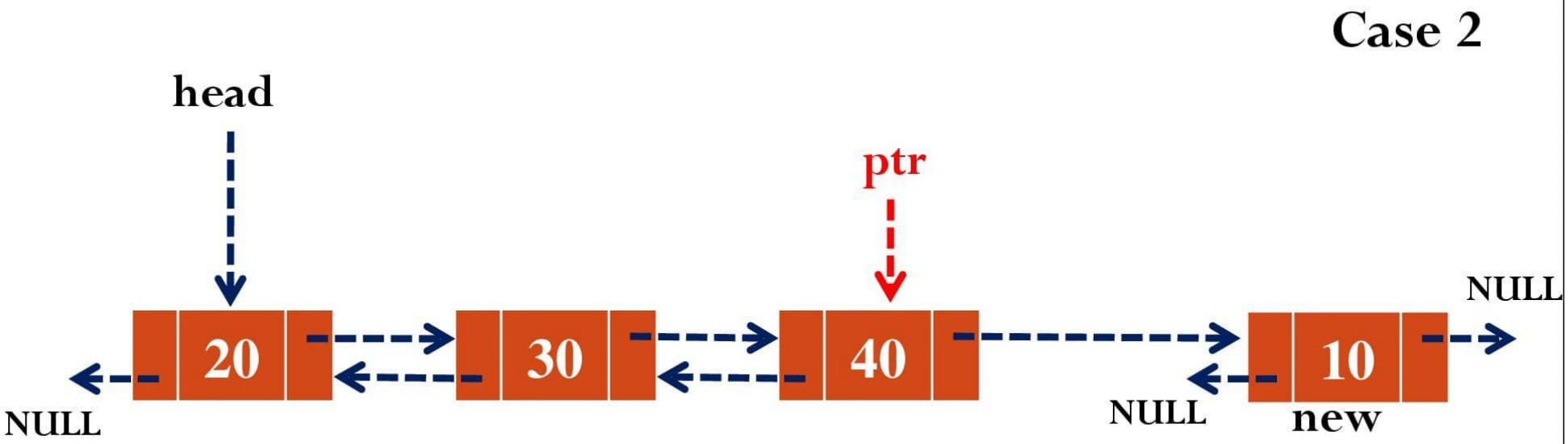


# Insert at End - Algorithm

Case 2

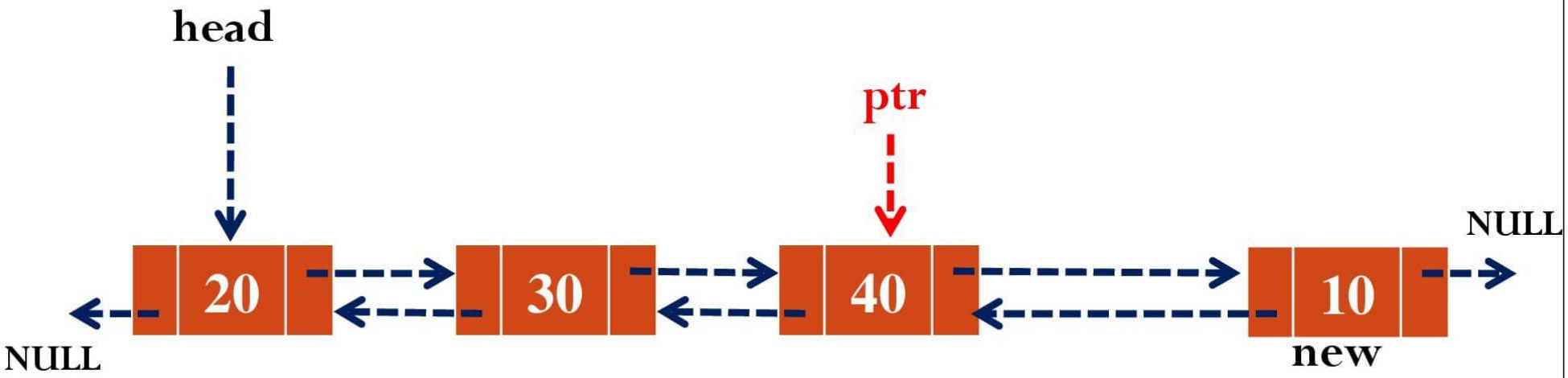


# Insert at End - Algorithm



# Insert at End - Algorithm

Case 2



# Insert at End - Algorithm

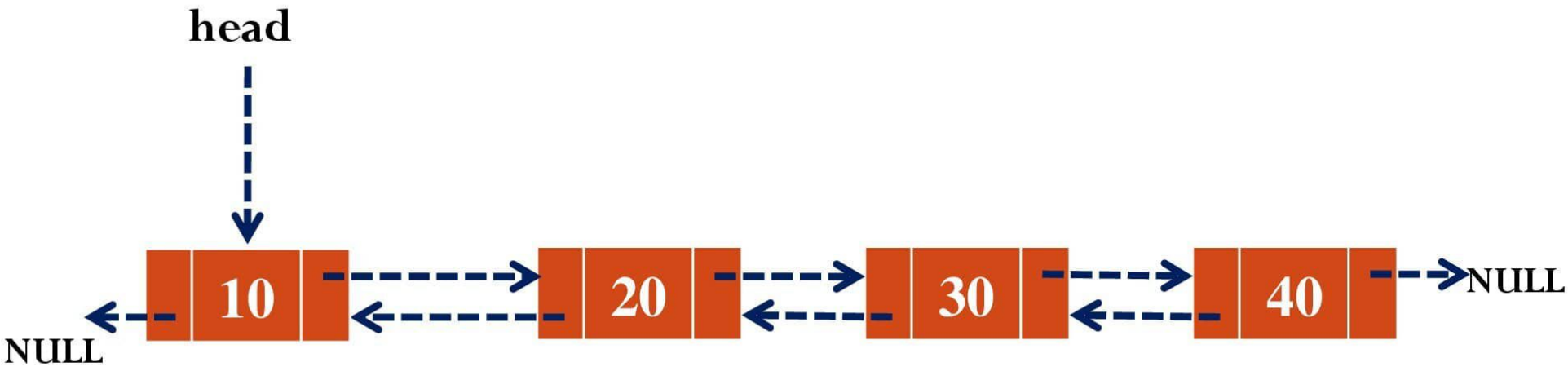
**Algorithm Insert_End(head, x)**

1. Create a node new
2. new→data=x
3. new→Rlink=new→Llink=NULL
4. If head=NULL then
    1. head=new
5. Else
    1. ptr=head
    2. While(ptr→Rlink!=NULL) do
        1. ptr=ptr→Rlink
    3. ptr→Rlink=new
    4. new→Llink=ptr

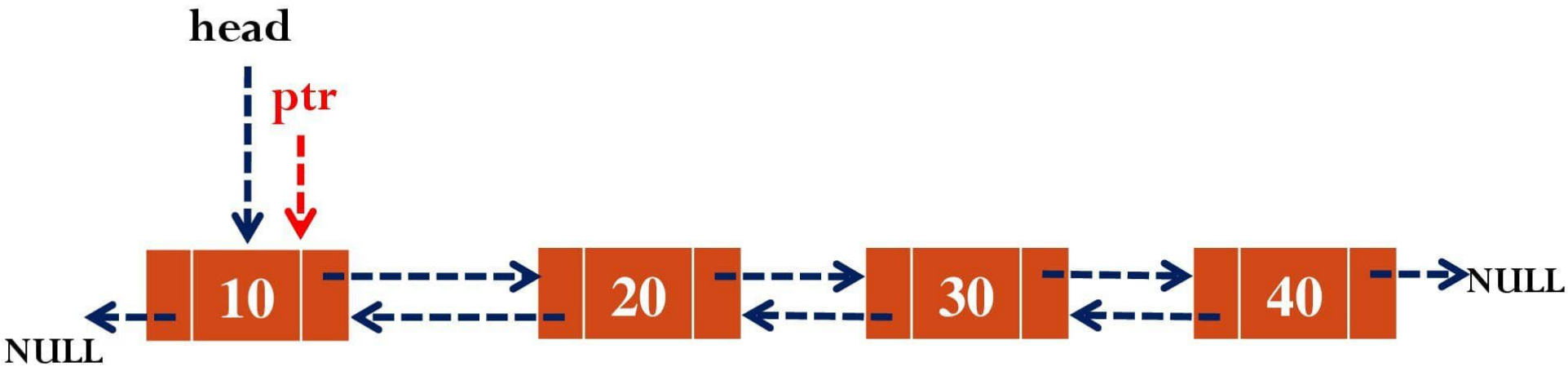
# Insertion

1. Insert at Front
2. Insert at End
3. **Insert after a specified node**

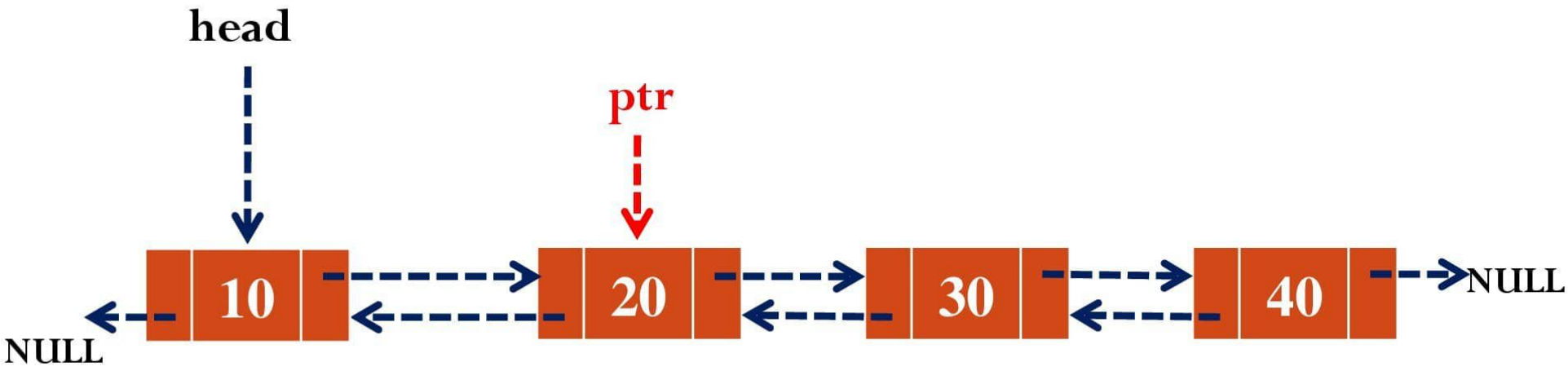
# Insert after 20



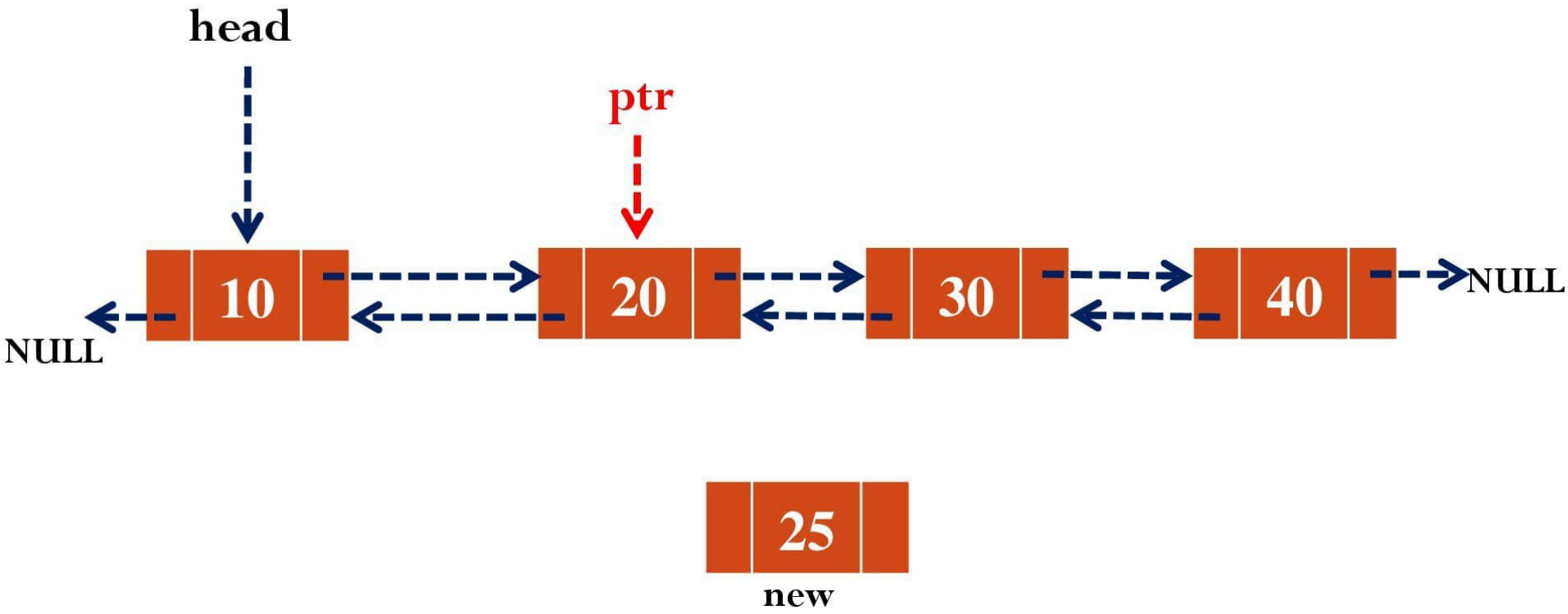
# Insert after 20



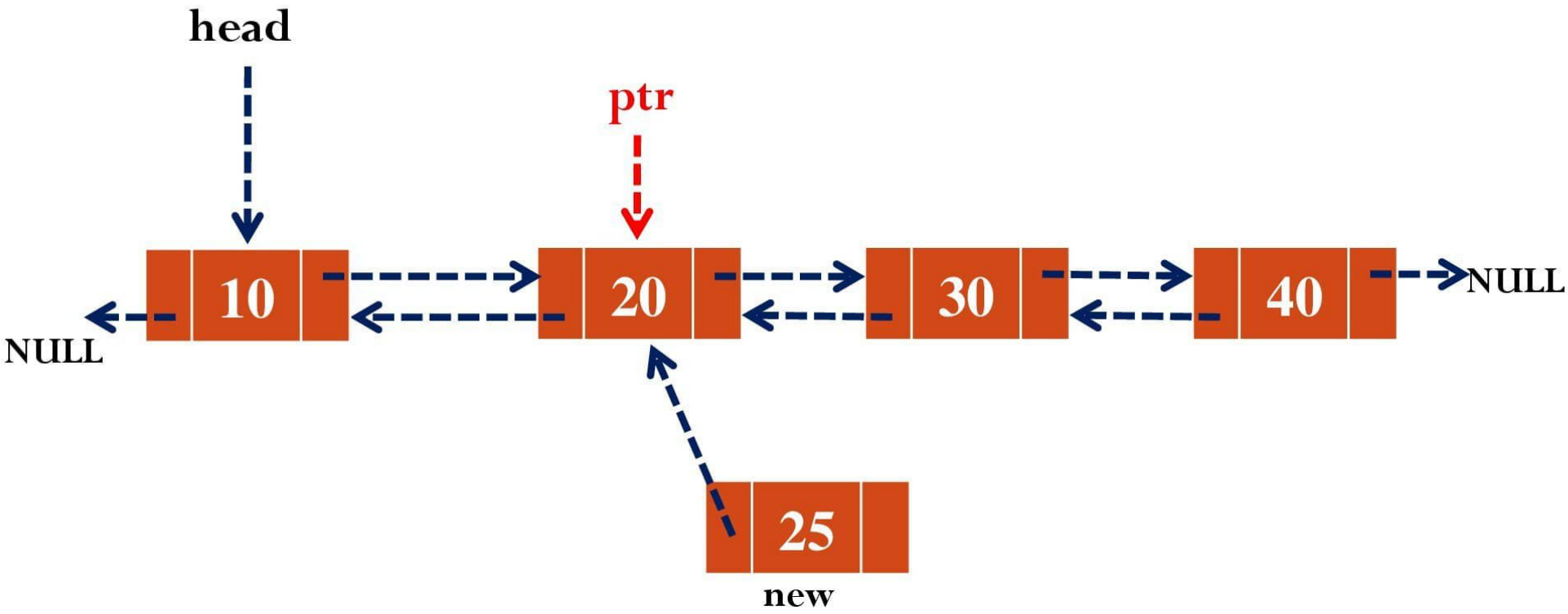
# Insert after 20



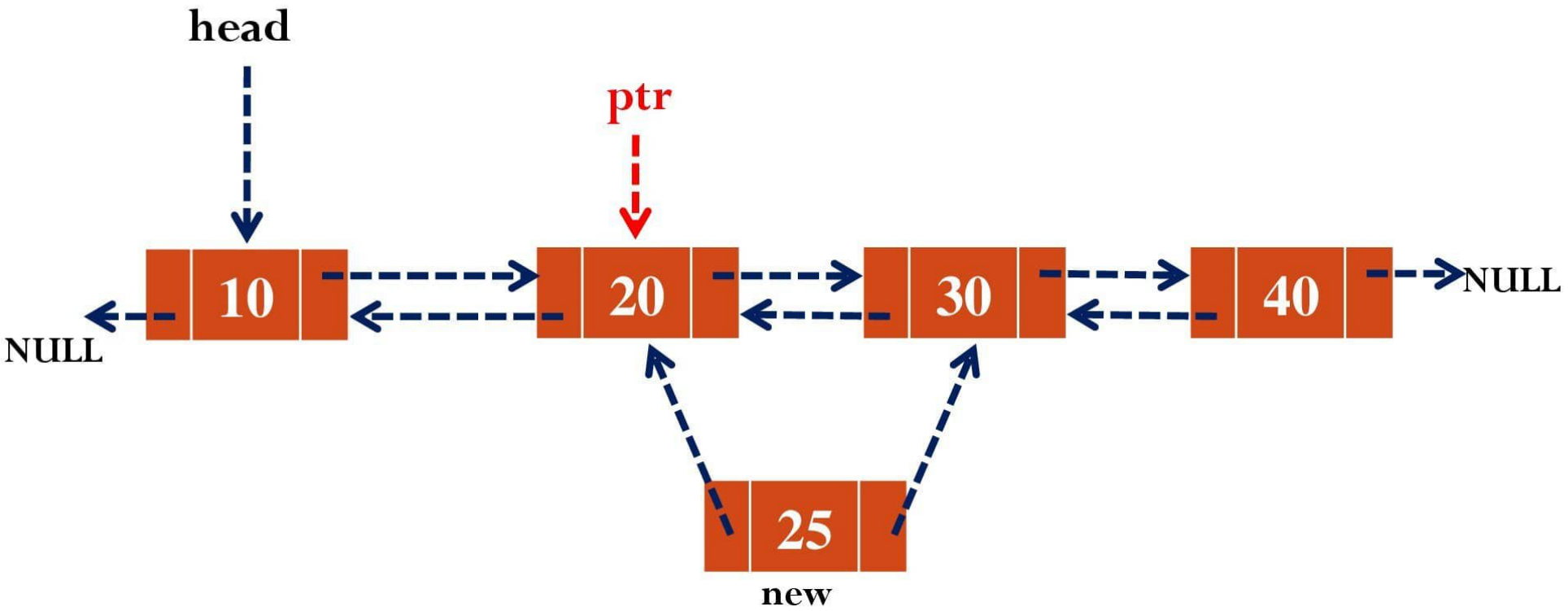
# Insert after 20



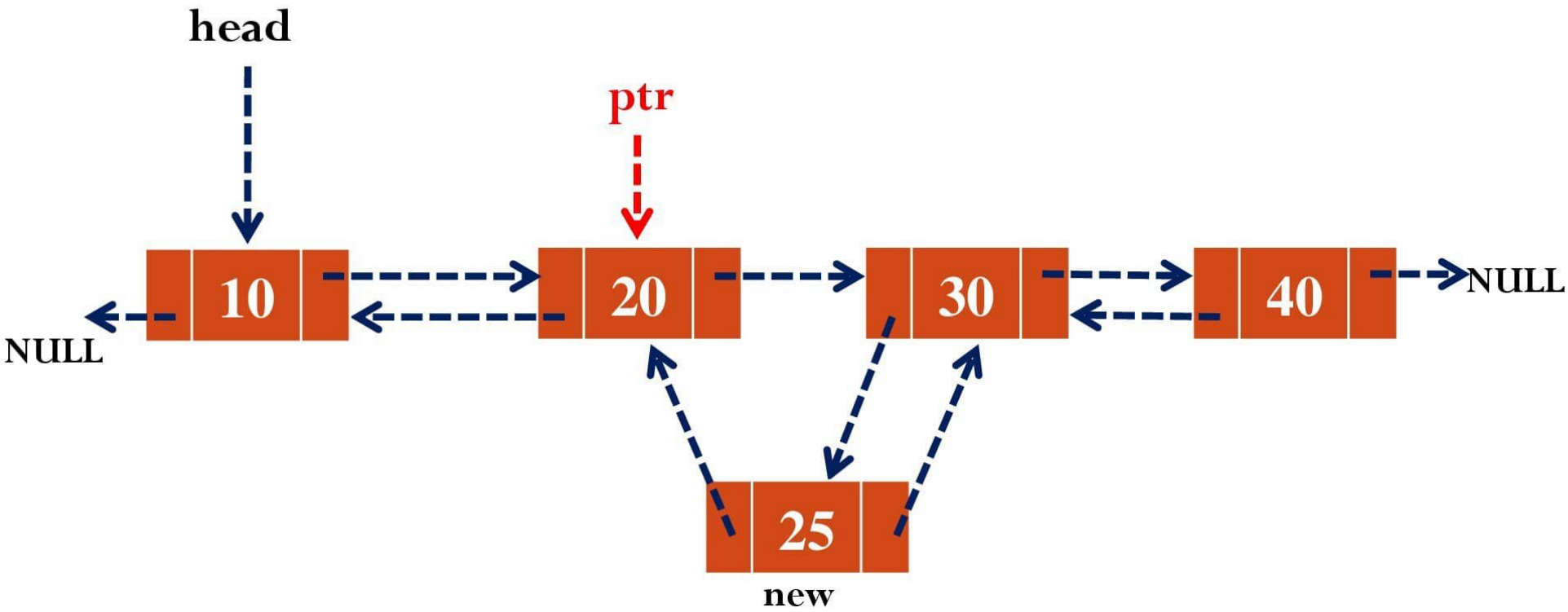
# Insert after 20



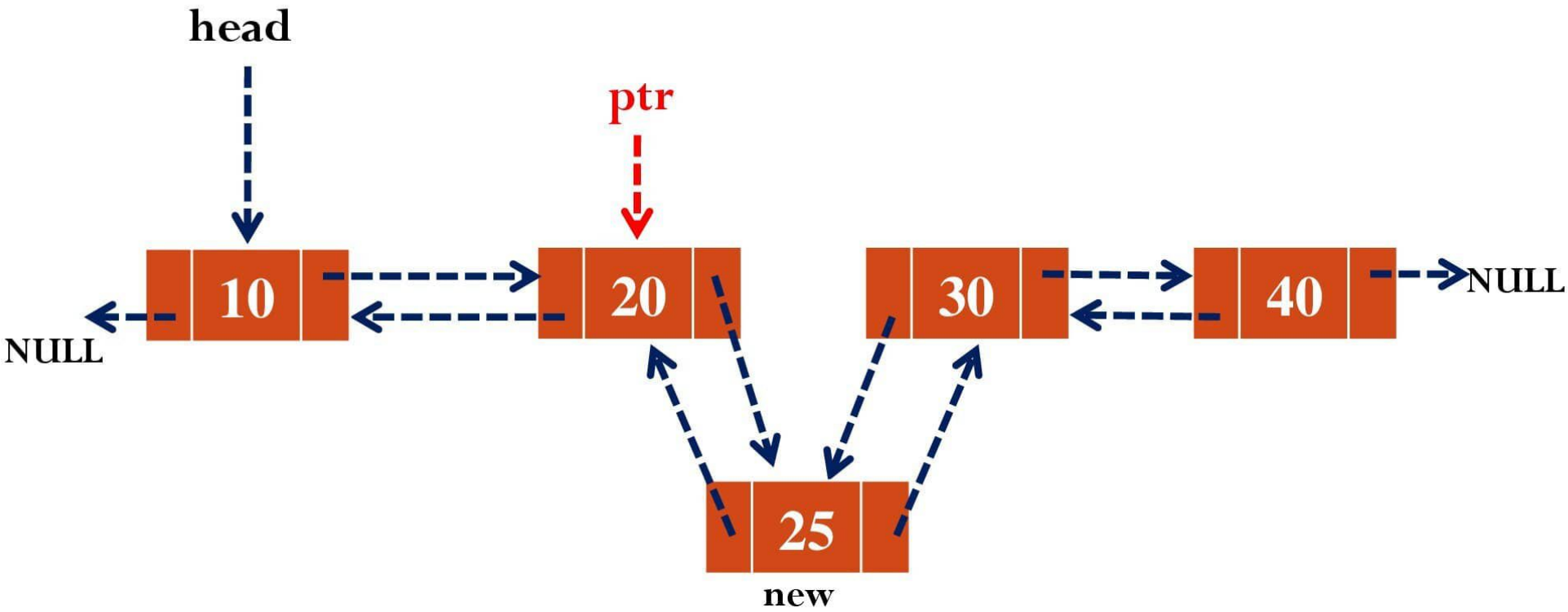
# Insert after 20



# Insert after 20



# Insert after 20



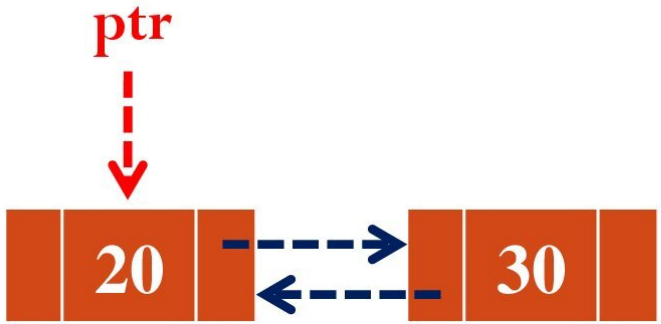
# Insert after a specified node- Algorithm

**Algorithm Insert_After (head, key, x)**

1. If head=NULL then
    1. Print "Search data not found. Insertion is not possible"
2. Else
    1. ptr=head
    2. While ptr→data!=key and ptr→Rlink!=NULL do
        1. ptr=ptr→Rlink
    3. If ptr→data!=key then
        1. Print "Search data not found. Insertion is not possible"

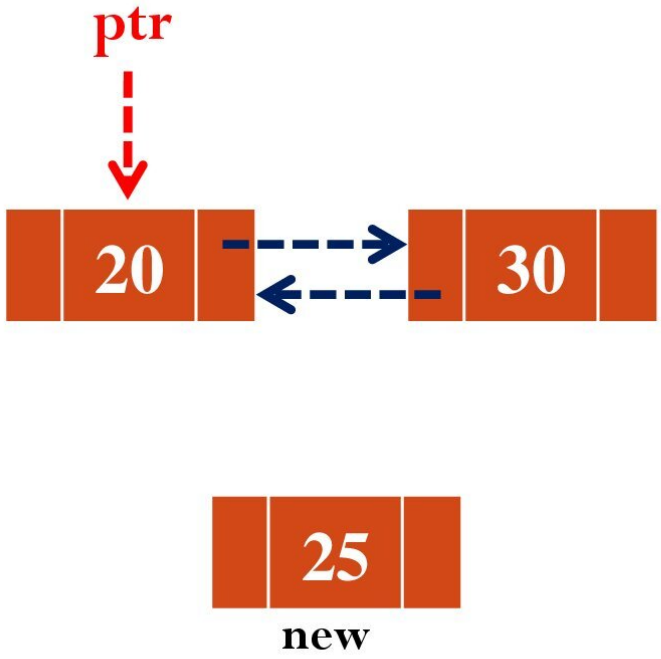
# Insert after a specified node- Algorithm

4. Else
    1. Create a node new
    2. new→data=x
    3. new→Llink=ptr
    4. new→Rlink=ptr→Rlink
    5. If new→Rlink!=NULL then
        1. new→Rlink→Llink=new
    6. ptr→Rlink=new



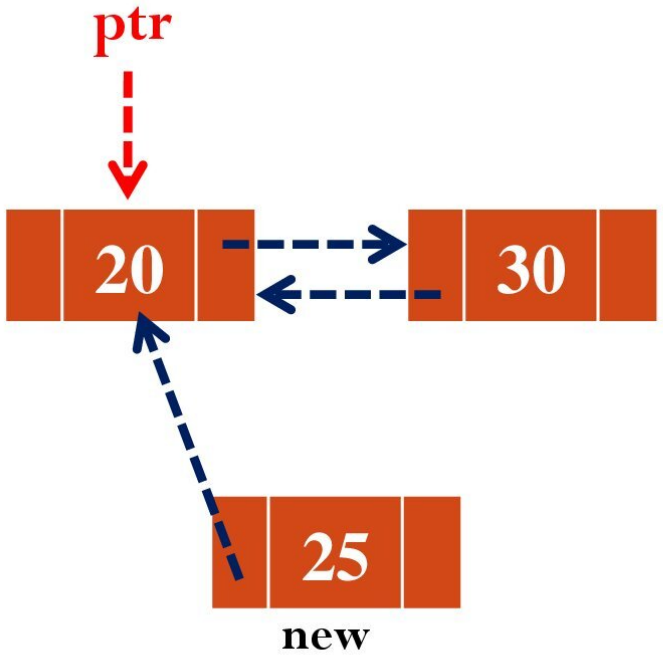
# Insert after a specified node- Algorithm

4. Else
    1. Create a node new
    2. new→data=x
    3. new→Llink=ptr
    4. new→Rlink=ptr→Rlink
    5. If new→Rlink!=NULL then
        1. new→Rlink→Llink=new
    6. ptr→Rlink=new



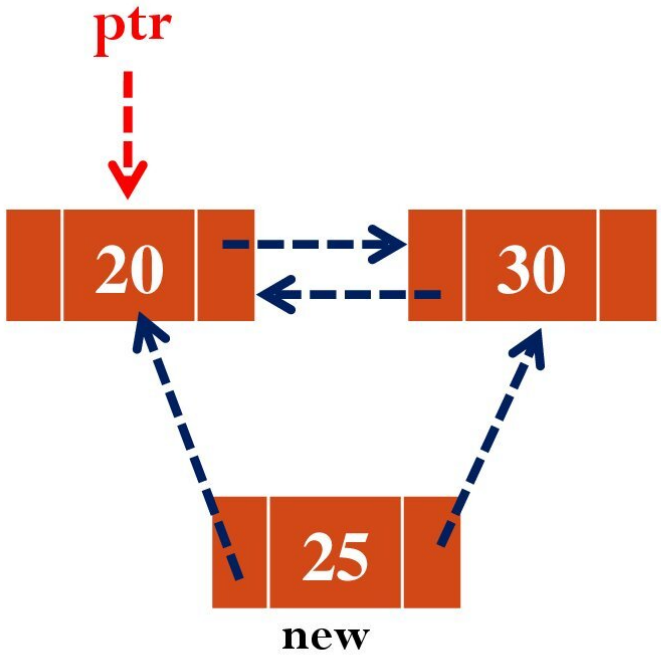
# Insert after a specified node- Algorithm

4. Else
    1. Create a node new
    2. new→data=x
    3. new→Llink=ptr
    4. new→Rlink=ptr→Rlink
    5. If new→Rlink!=NULL then
        1. new→Rlink→Llink=new
    6. ptr→Rlink=new



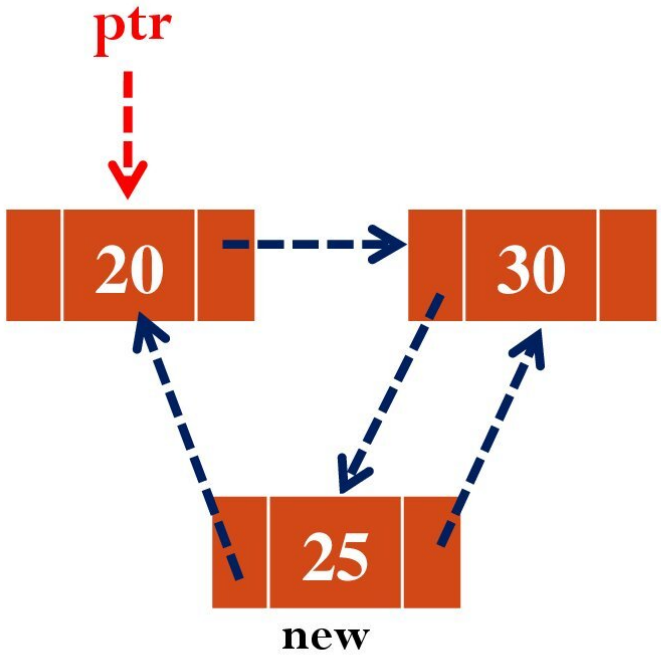
# Insert after a specified node- Algorithm

4. Else
    1. Create a node new
    2. new→data=x
    3. new→Llink=ptr
    4. new→Rlink=ptr→Rlink
    5. If new→Rlink!=NULL then
        1. new→Rlink→Llink=new
    6. ptr→Rlink=new



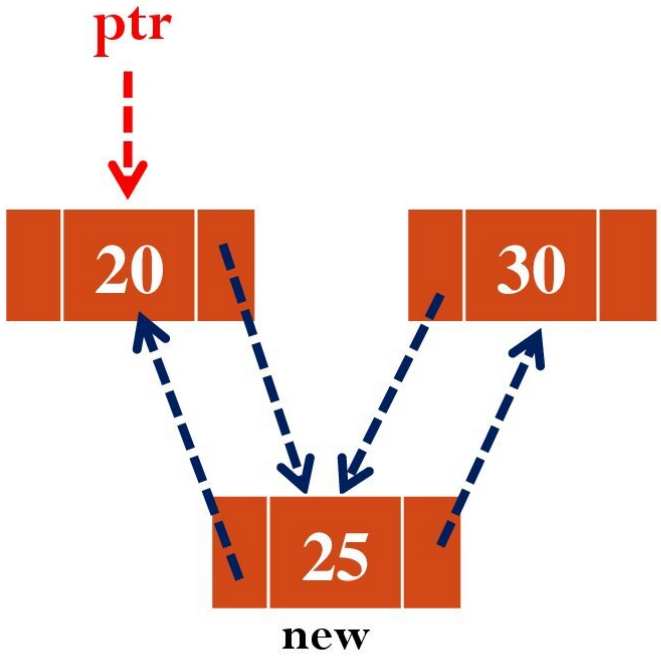
# Insert after a specified node- Algorithm

4. Else
    1. Create a node new
    2. new→data=x
    3. new→Llink=ptr
    4. new→Rlink=ptr→Rlink
    5. If new→Rlink!=NULL then
        1. new→Rlink→Llink=new
    6. ptr→Rlink=new



# Insert after a specified node- Algorithm

4. Else
    1. Create a node new
    2. new→data=x
    3. new→Llink=ptr
    4. new→Rlink=ptr→Rlink
    5. If new→Rlink!=NULL then
        1. new→Rlink→Llink=new
    6. ptr→Rlink=new



# Deletion

1. Delete from Front
2. Delete from End
3. Delete a specified node

# Deletion

1. **Delete from Front**
2. Delete from End
3. Delete a specified node

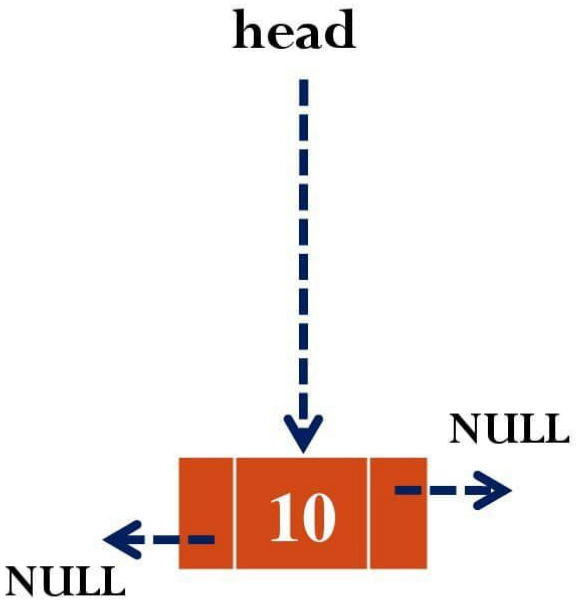
# Delete from Front

3 Cases

1. List is Empty
2. List contains only one node
3. List contains more than one node

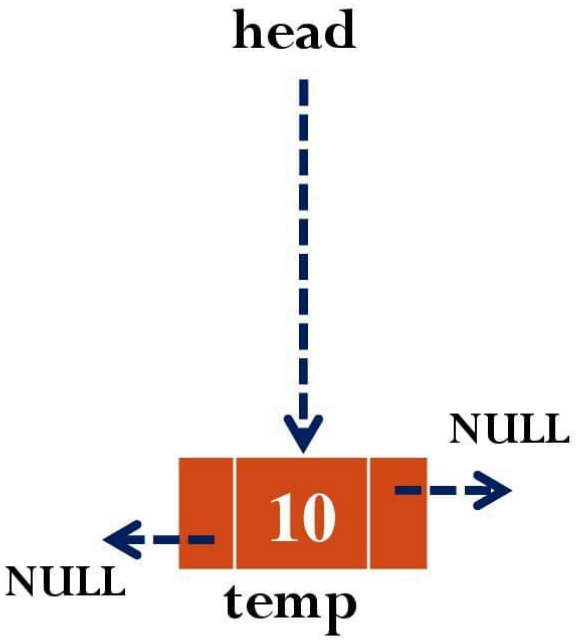
# Delete from Front

Case 2



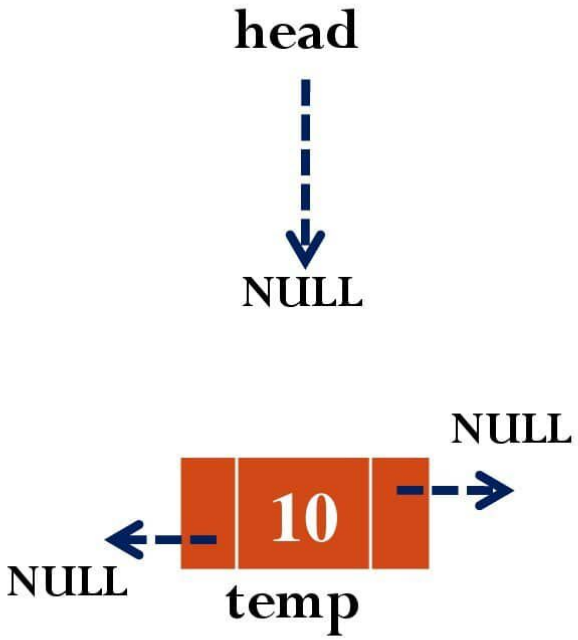
# Delete from Front

Case 2



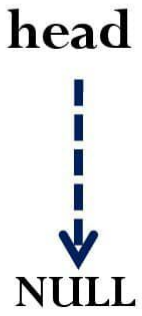
# Delete from Front

Case 2



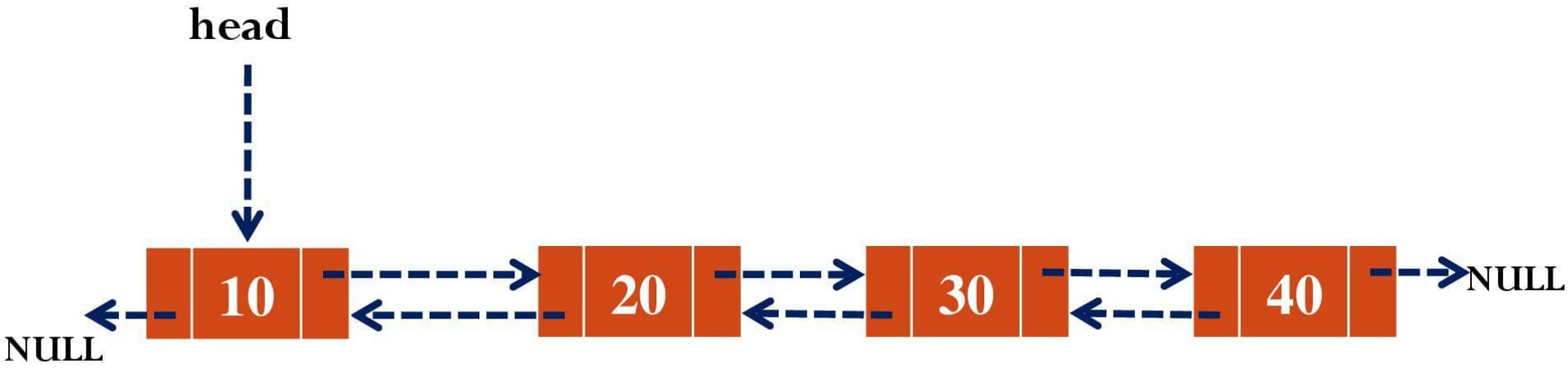
# Delete from Front

Case 2



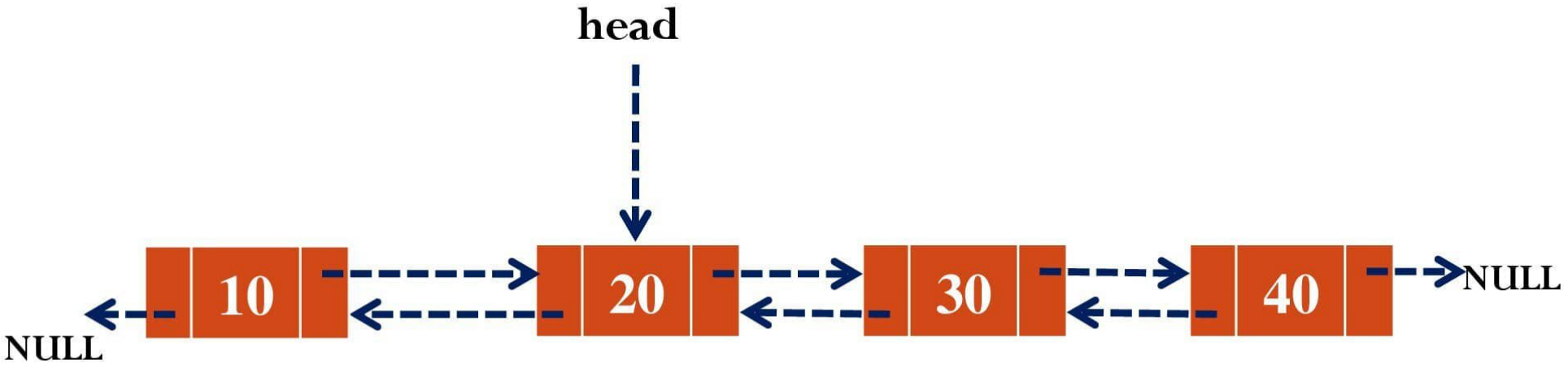
# Delete from Front

Case 3



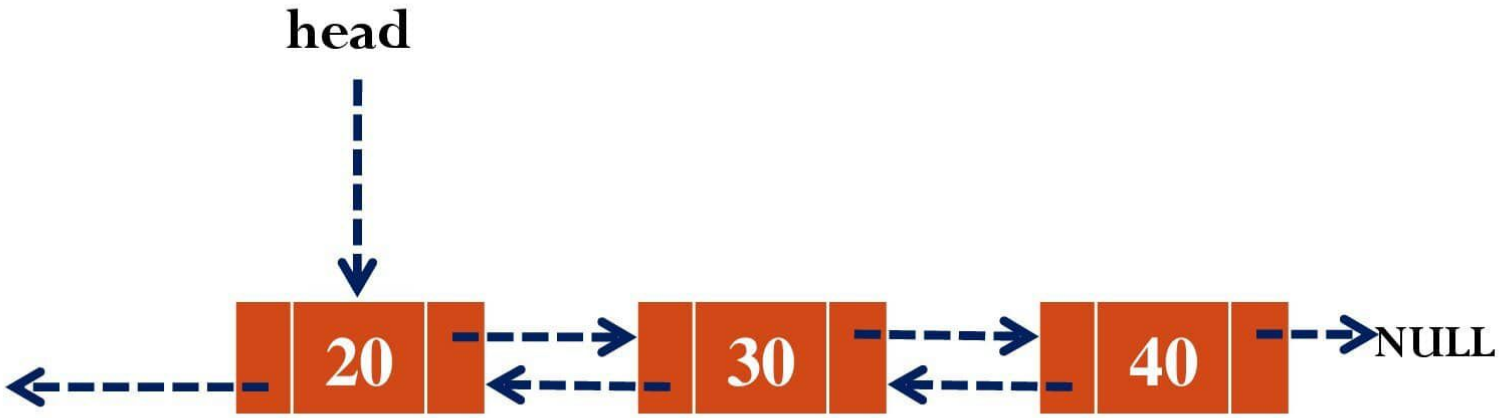
# Delete from Front

Case 3



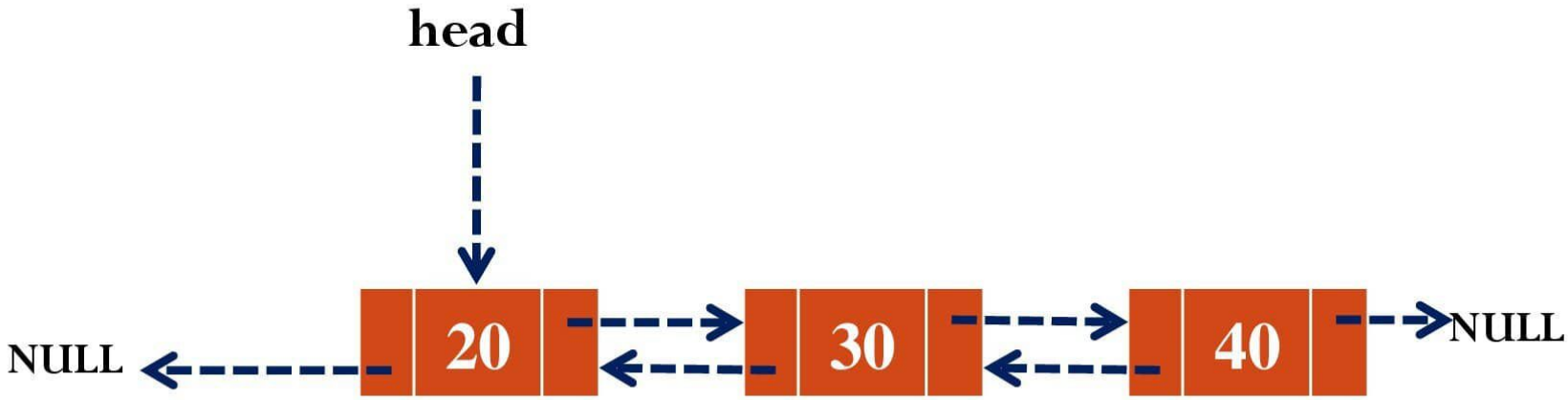
# Delete from Front

Case 3



# Delete from Front

Case 3



# Delete from Front~ Algorithm

**Algorithm Delete_Front(head)**

1. If head ==NULL then
    1. Print “List is empty”
2. Else if head→Rlink=NULL then
    1. temp=head
    2. head=NULL
    3. dispose(temp)
3. Else
    1. head=head→Rlink
    2. dispose(head→Llink)
    3. head→Llink=NULL

# Deletion

1. Delete from Front
2. **Delete from End**
3. Delete a specified node

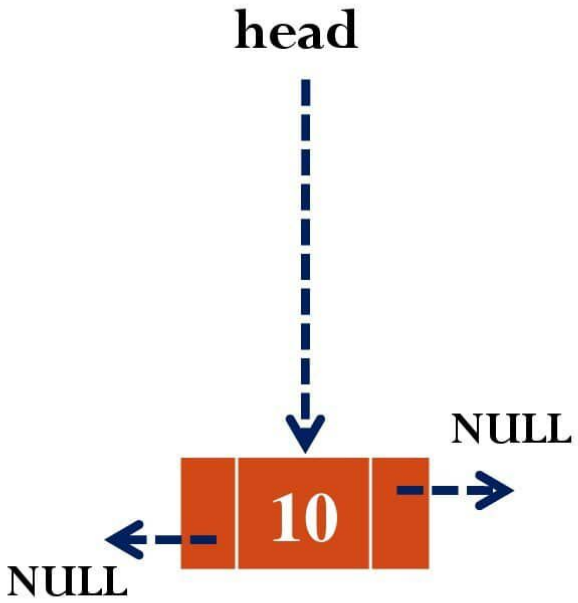
# Delete from End

3 Cases

1. List is Empty
2. List contains only one node
3. List contains more than one node

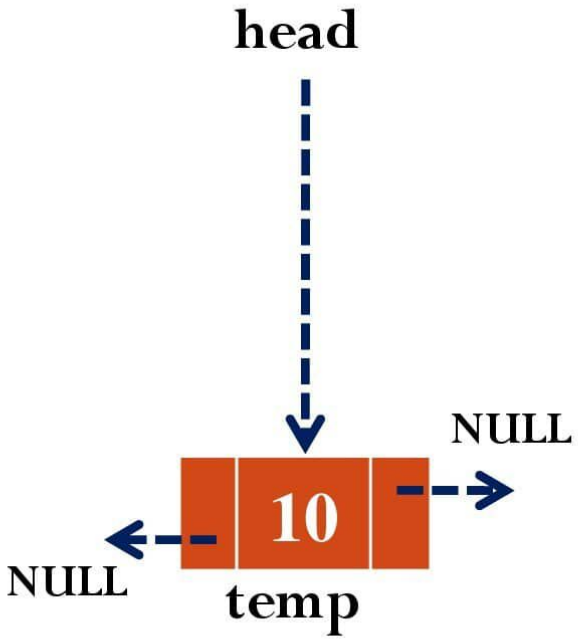
# Delete from End

Case 2



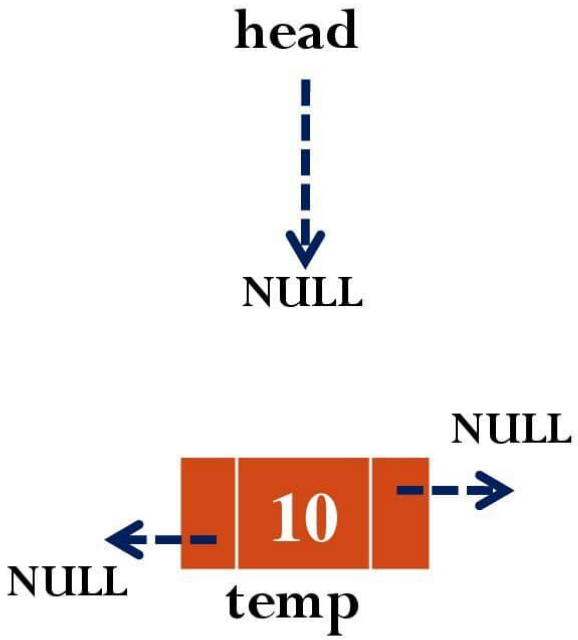
# Delete from End

Case 2



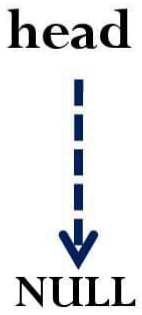
# Delete from End

Case 2



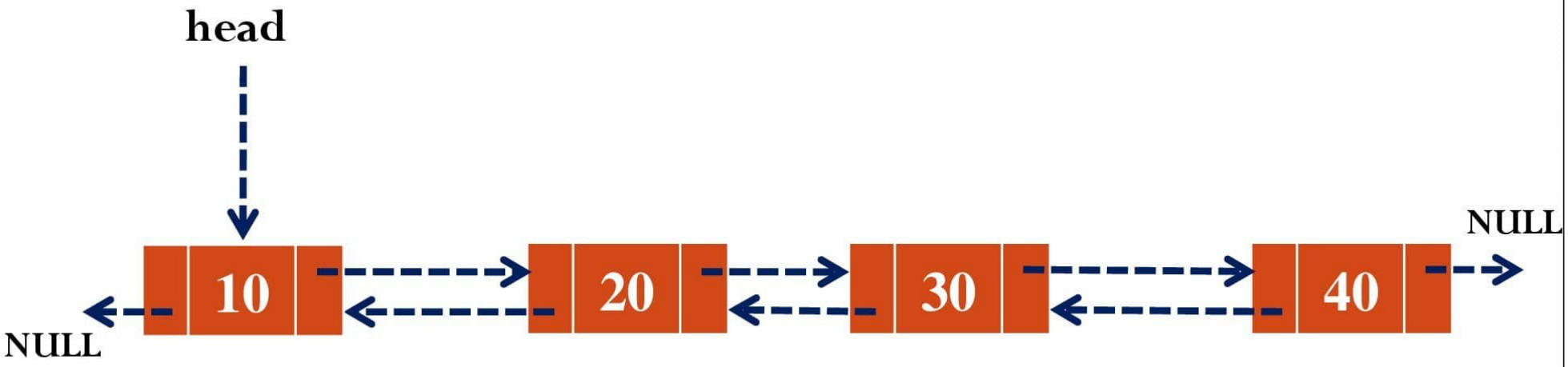
# Delete from End

Case 2



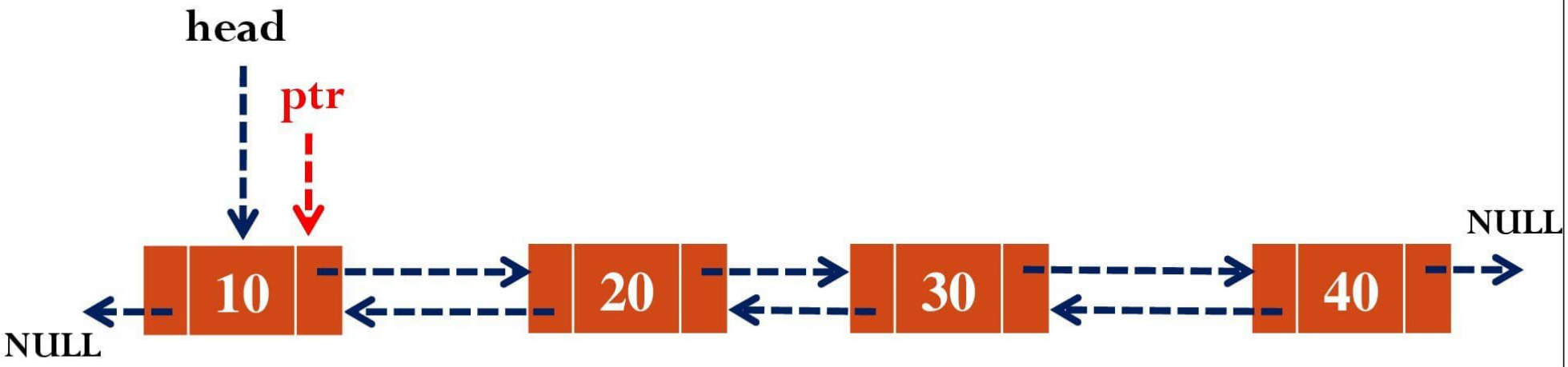
# Delete from End

Case 3



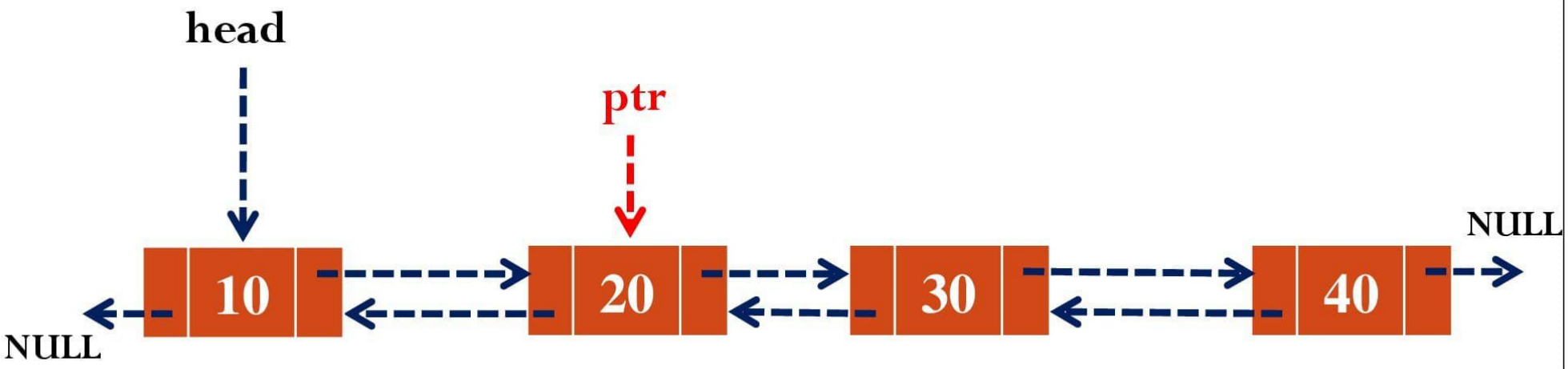
# Delete from End

Case 3



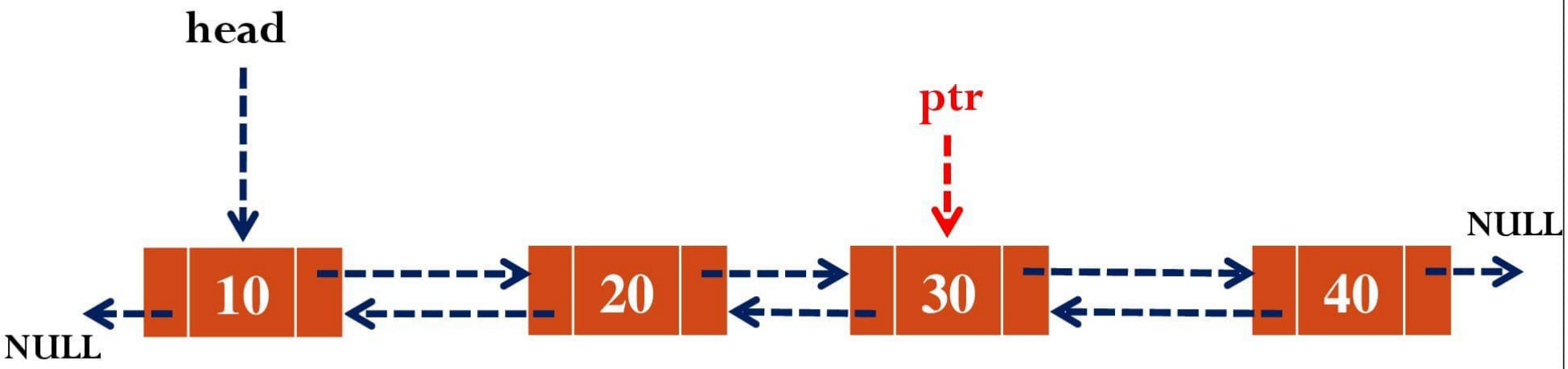
# Delete from End

Case 3

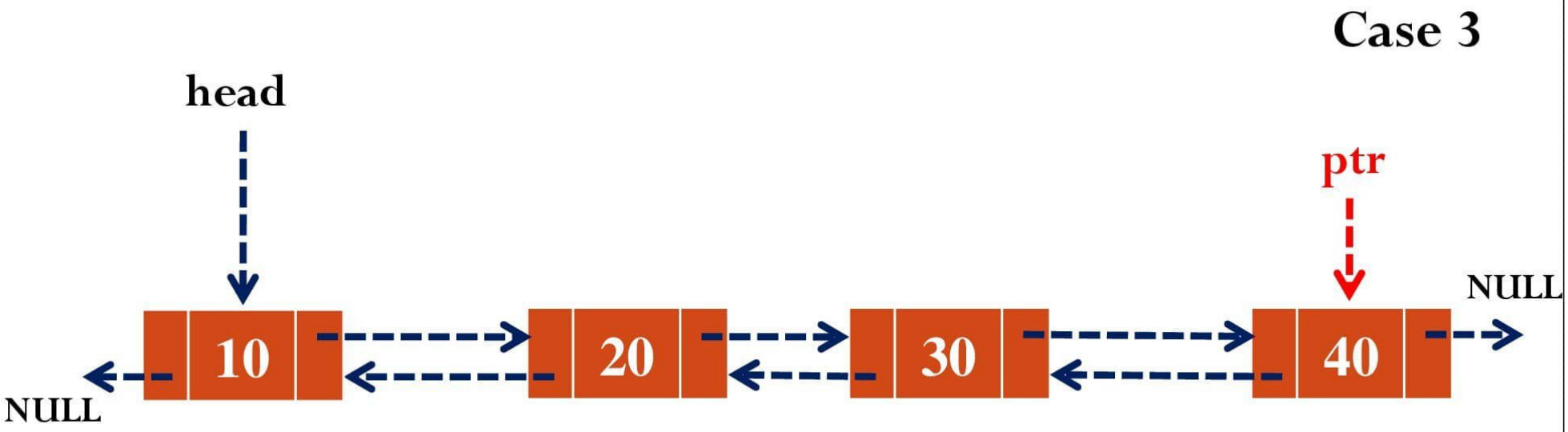


# Delete from End

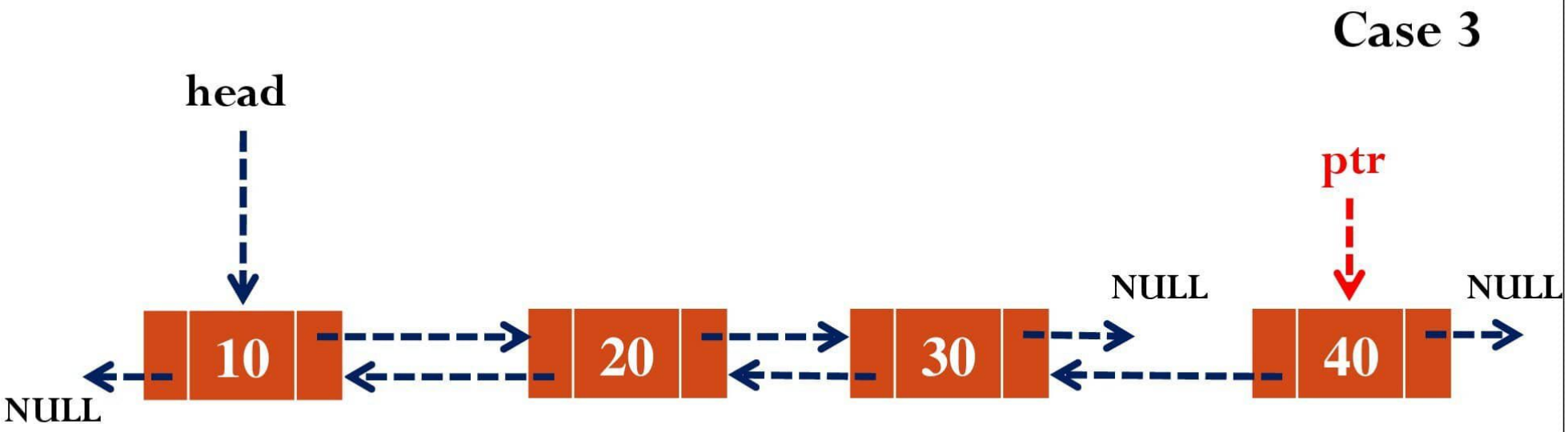
Case 3



# Delete from End

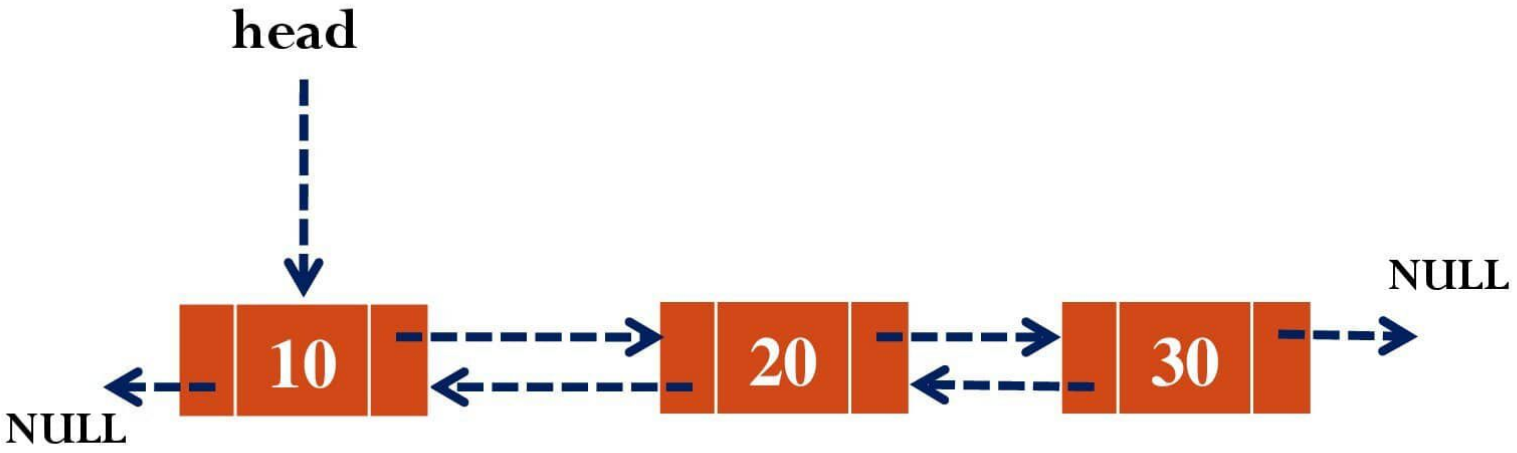


# Delete from End



# Delete from End

Case 3



# Delete from End- Algorithm

**Algorithm Delete_End (head)**

1. If head ==NULL then
    1. Print “List is empty”
2. Else if head→Rlink=NULL then
    1. temp=head
    2. head=NULL
    3. dispose(temp)
3. Else
    1. ptr=head
    2. While ptr→Rlink!=NULL do
        1. ptr=ptr→Rlink
    3. ptr→Llink→Rlink=NULL
    4. dispose(ptr)

# Deletion

1. Delete from Front
2. Delete from End
3. **Delete a specified node**

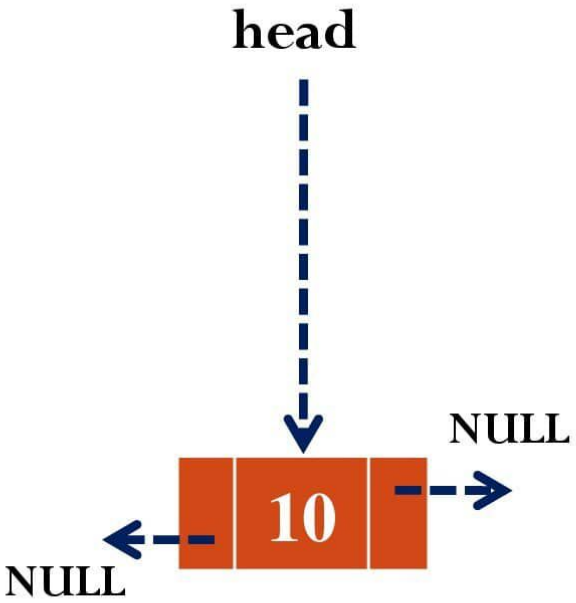
# Delete a specified node

4 Cases

1. List is Empty
2. List contains only one node
3. Search data present in the first node
4. All other cases

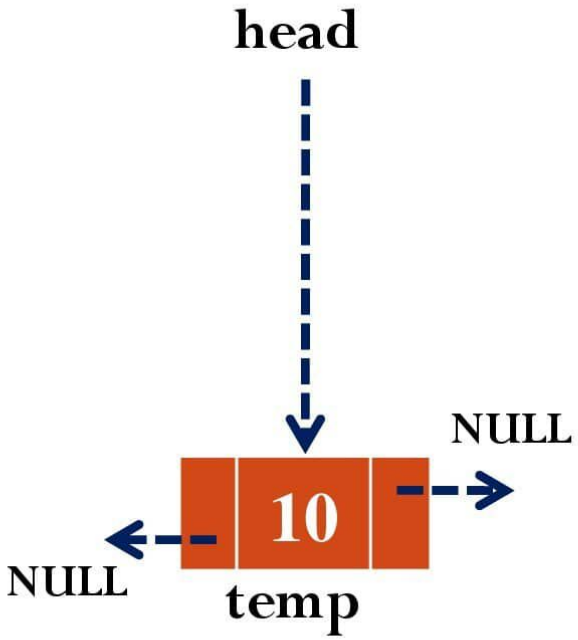
# Delete 10

Case 2



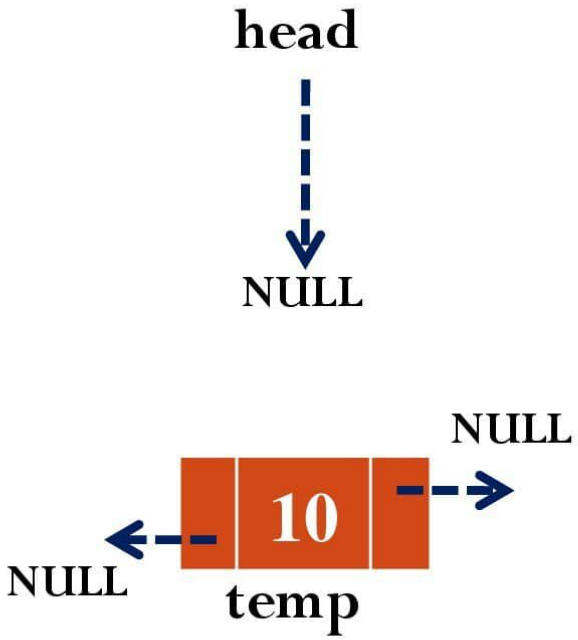
# Delete 10

Case 2



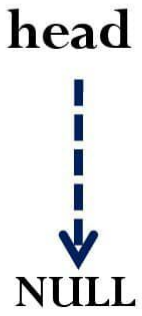
# Delete 10

Case 2



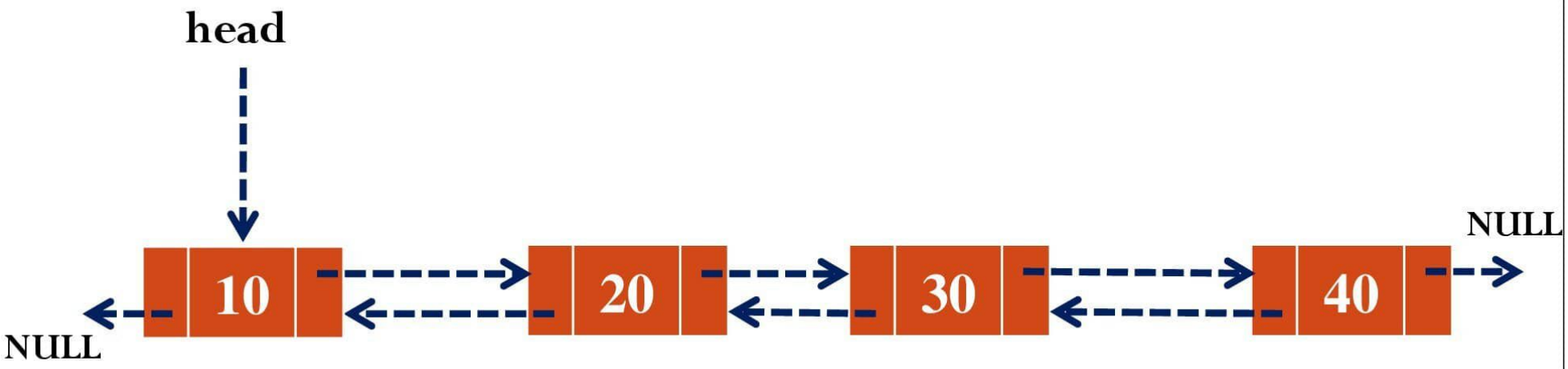
# Delete 10

Case 2



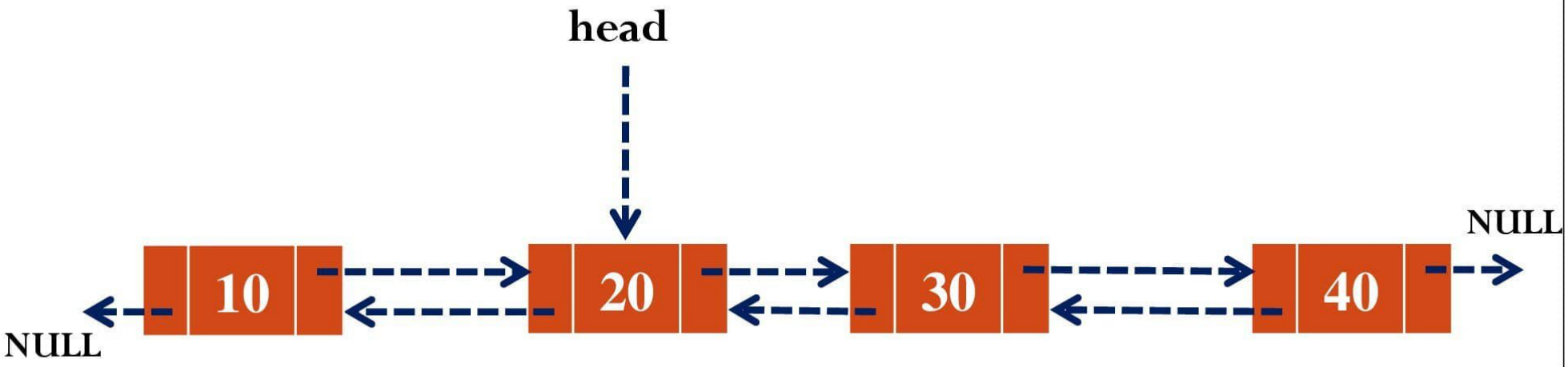
# Delete 10

Case 3



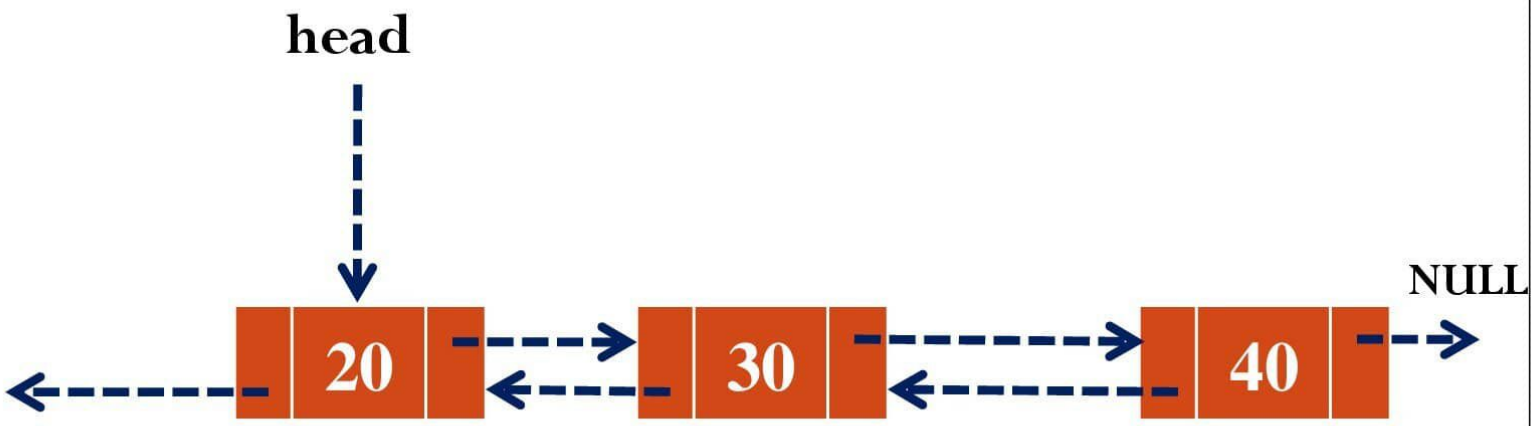
# Delete 10

Case 3



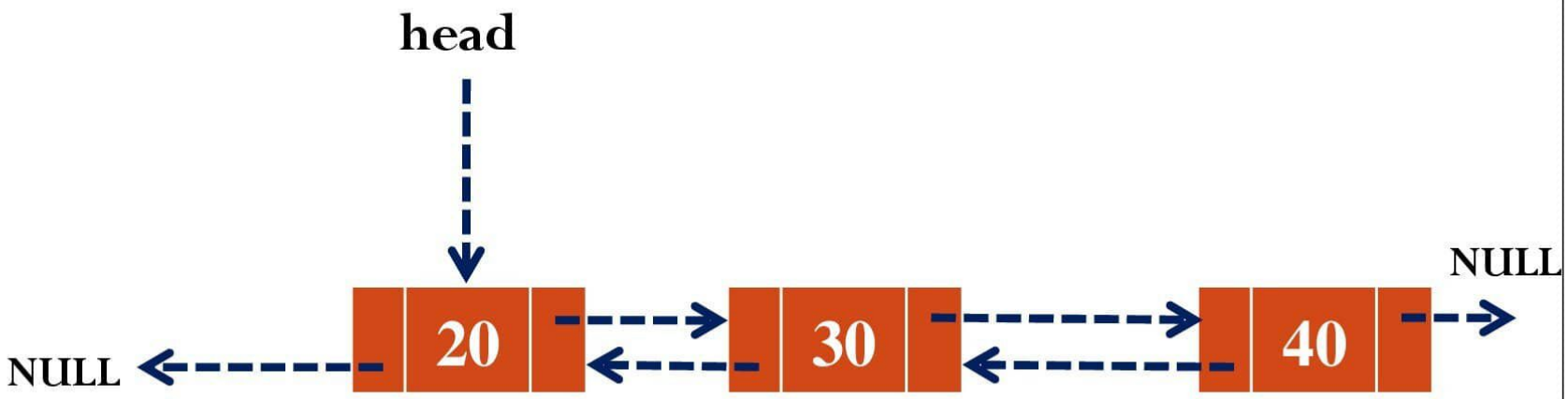
# Delete 10

Case 3



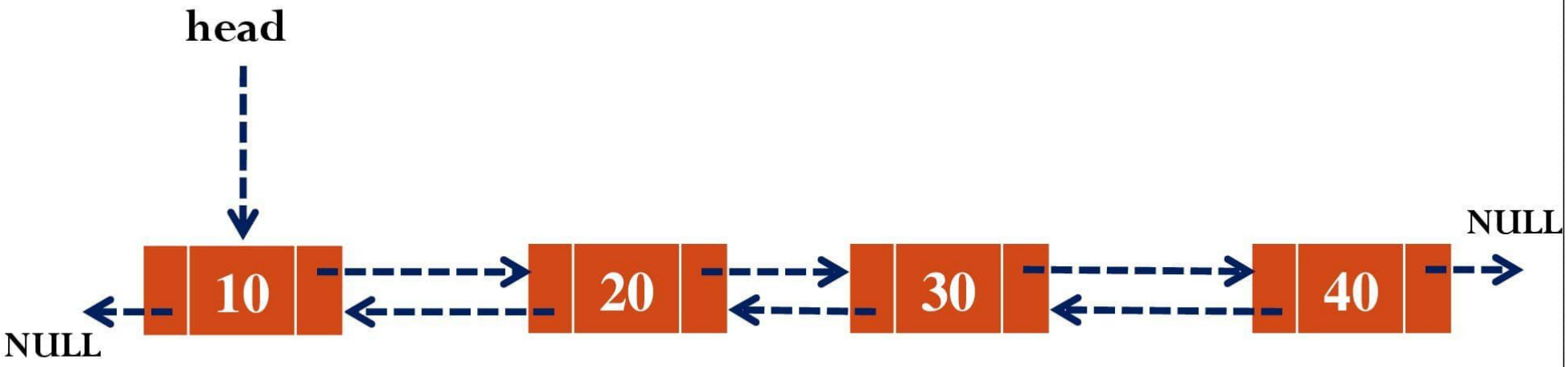
# Delete 10

Case 3



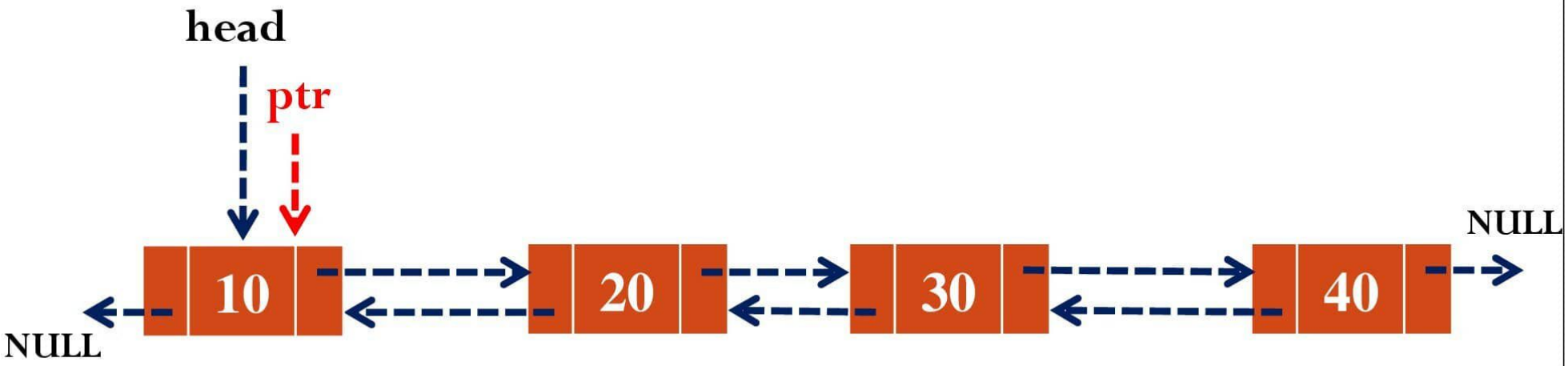
# Delete 30

Case 4



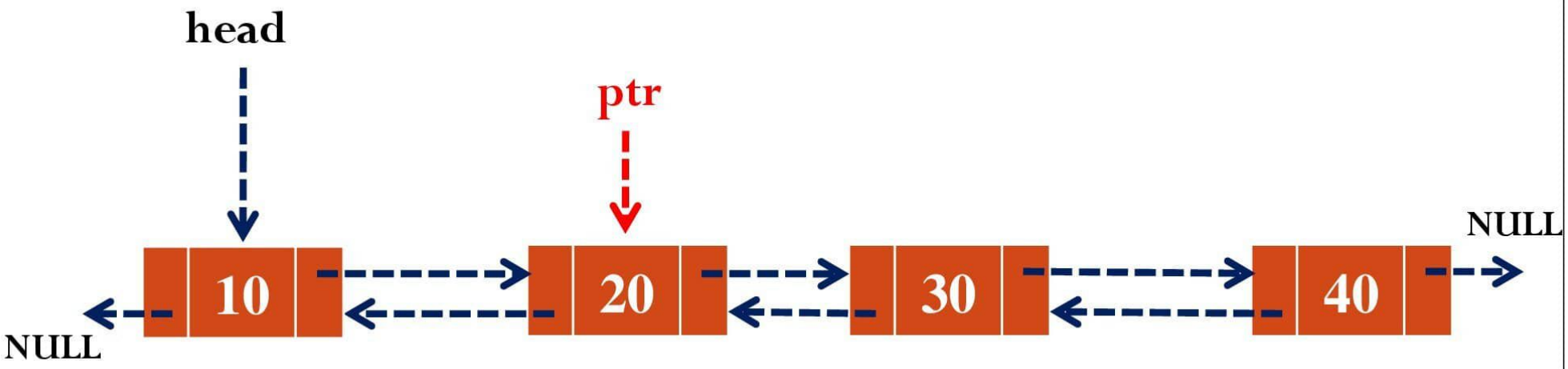
# Delete 30

Case 4



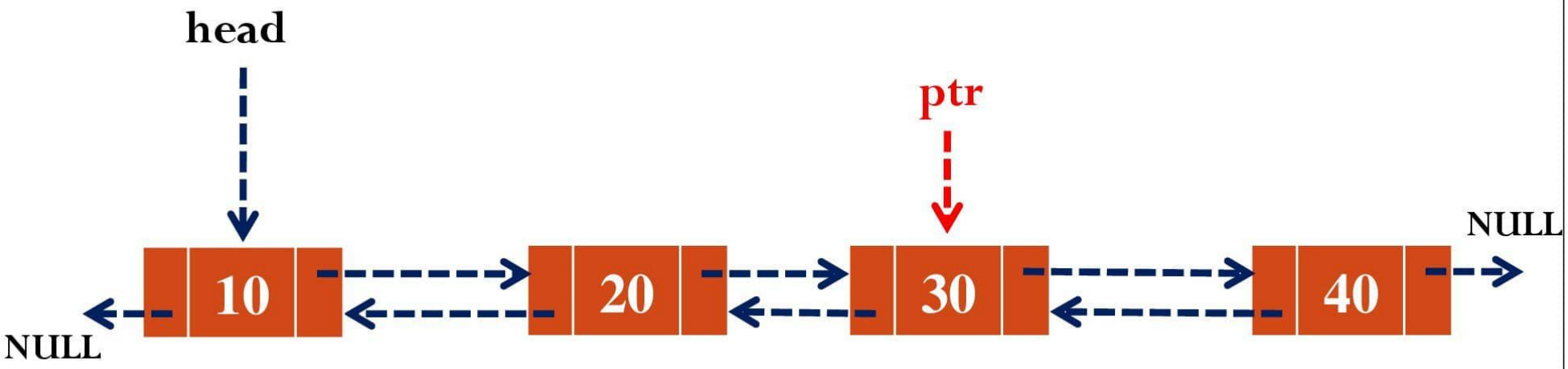
# Delete 30

Case 4

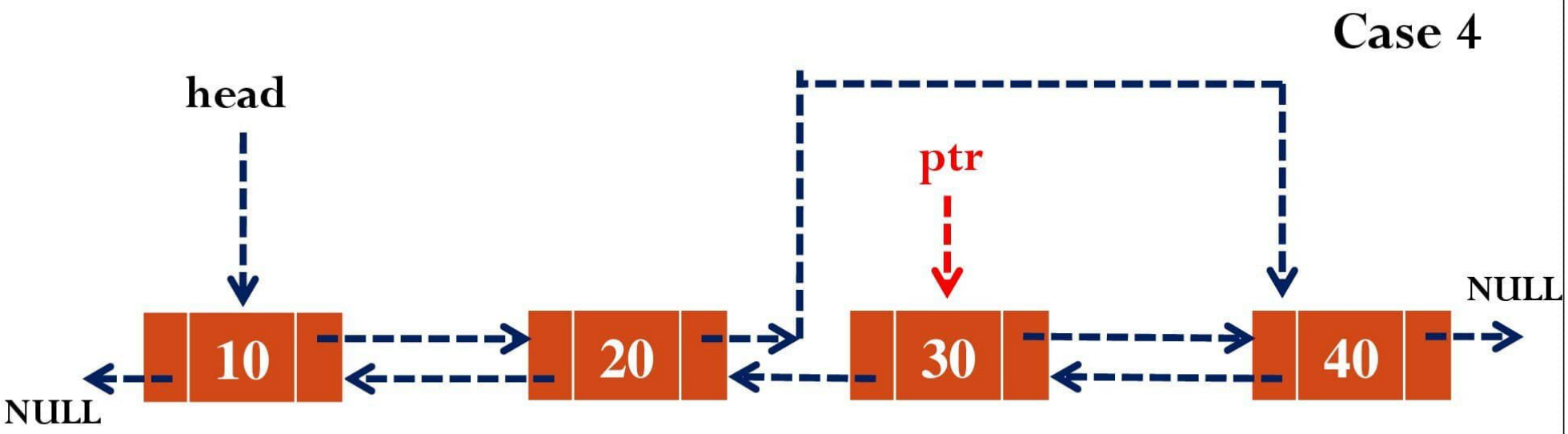


# Delete 30

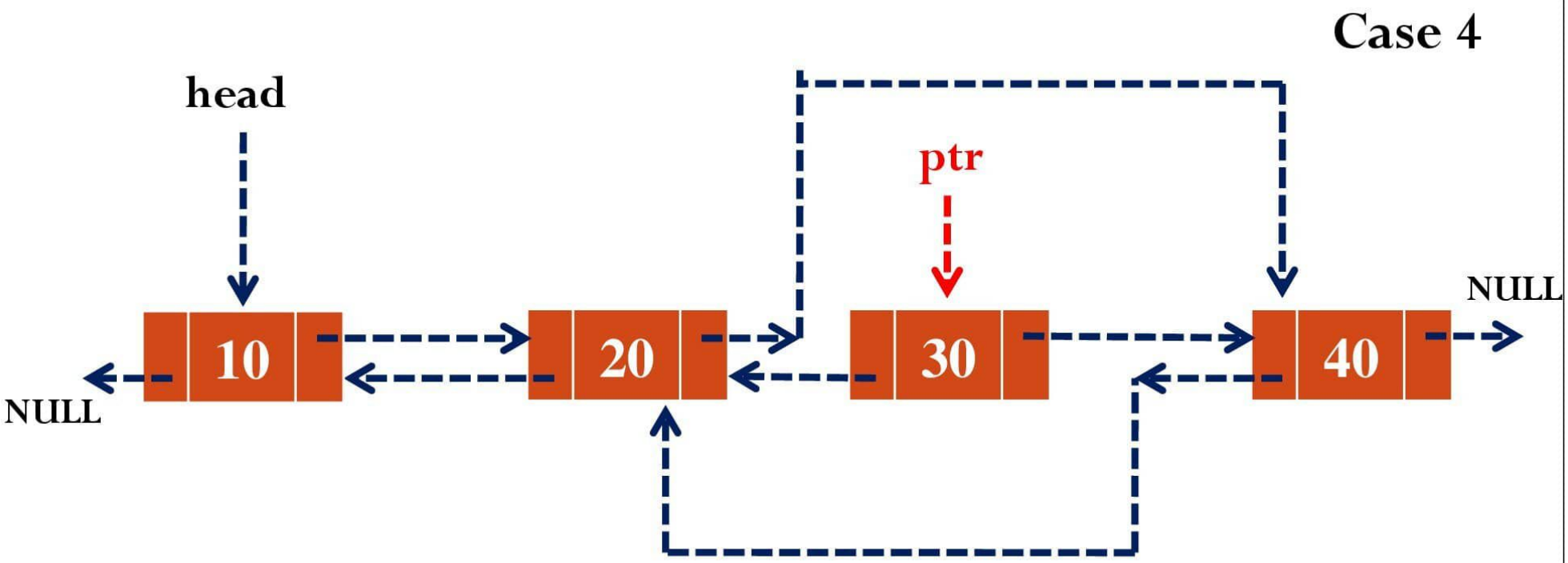
Case 4



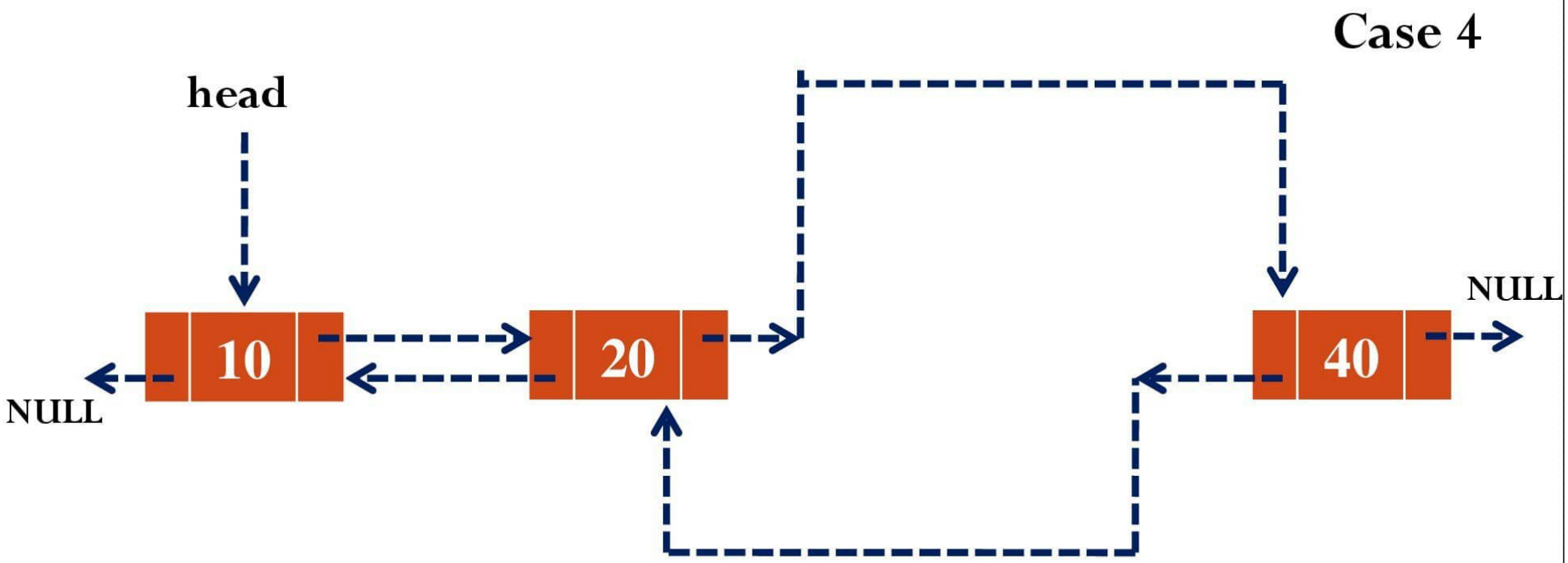
# Delete 30



# Delete 30

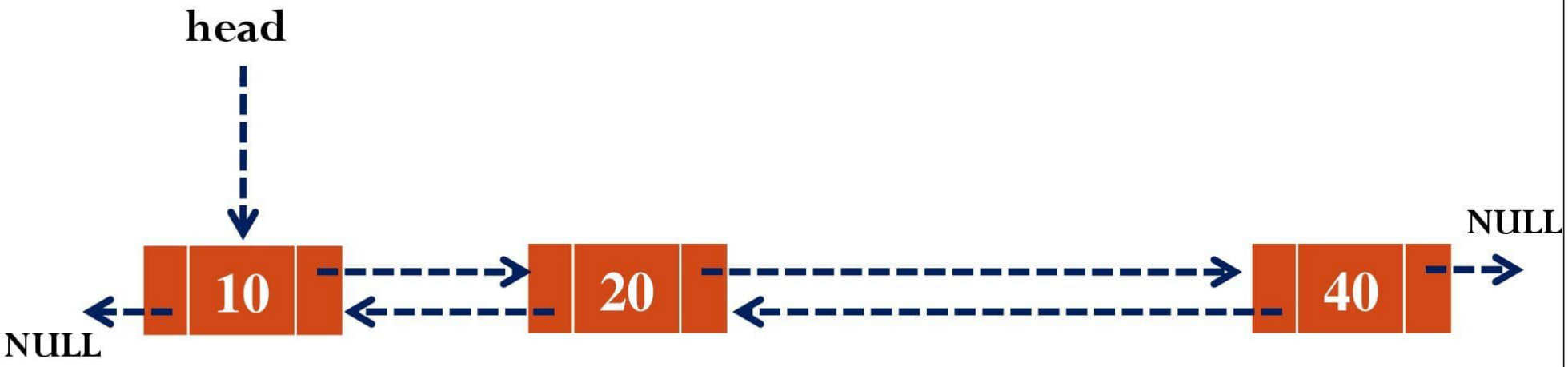


# Delete 30



# Delete 30

Case 4



# Delete a specified node- Algorithm

**Algorithm Delete_Any (head, key)**

1. If head ==NULL then
    1. Print "List is empty"
2. Else if head→Rlink=NULL then
    1. If head→data=key then
        1. temp=head
        2. head=NULL
        3. dispose(temp)
    2. Else
        1. Print "Search data not found. Deletion is not possible"

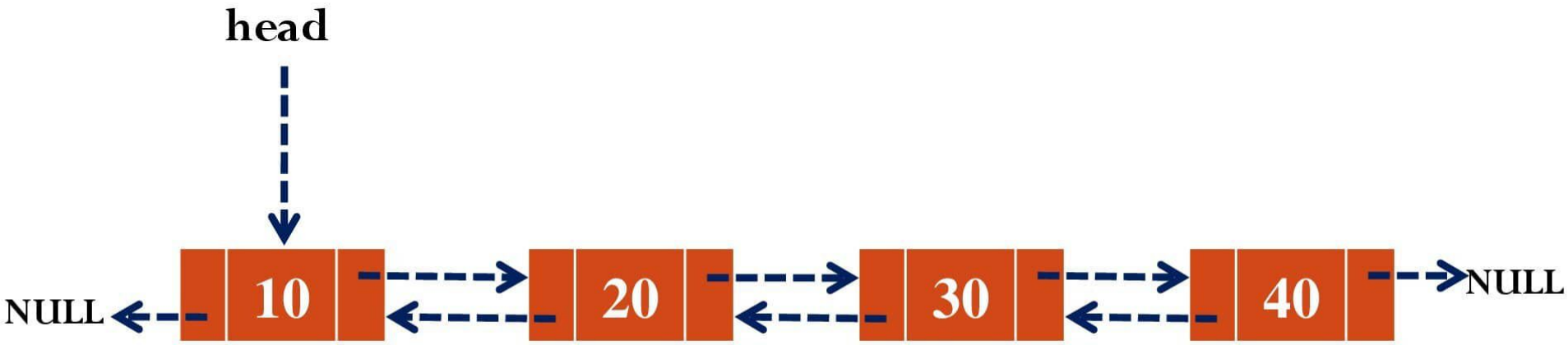
# Delete a specified node- Algorithm

3. Else if head→data=key then
    1. head=head→Rlink
    2. dispose(head→Llink)
    3. head→Llink=NULL
4. Else
    1. ptr=head
    2. While ptr→data!=key and ptr→Rlink!=NULL do
        1. ptr=ptr→Rlink

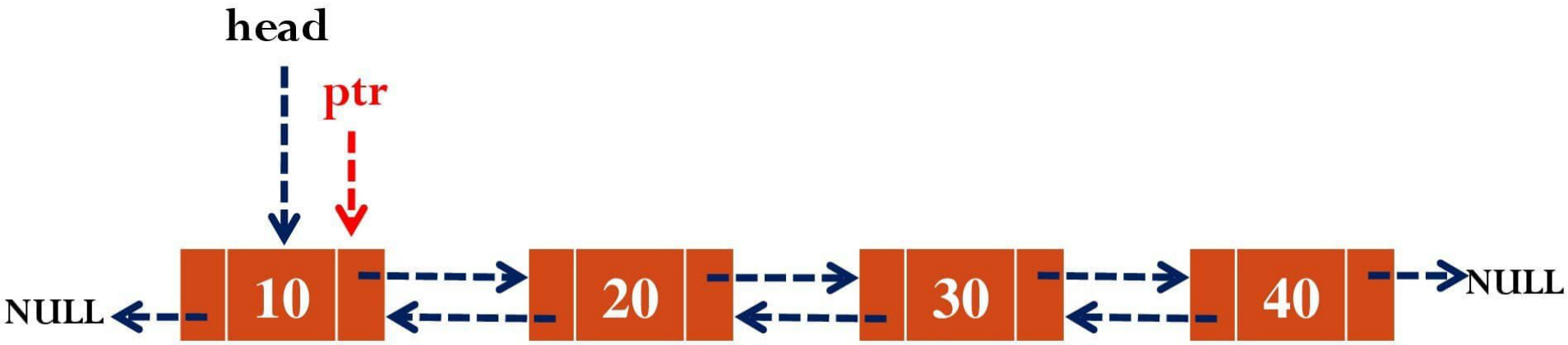
# Delete a specified node- Algorithm

3. If ptr→data!=key then
    1. Print "Search data not found. Deletion is not possible"
4. Else
    1. ptr→Llink→Rlink=ptr→Rlink
    2. If ptr→Rlink!=NULL then
        1. ptr→Rlink→Llink=ptr→Llink
    3. dispose(ptr)

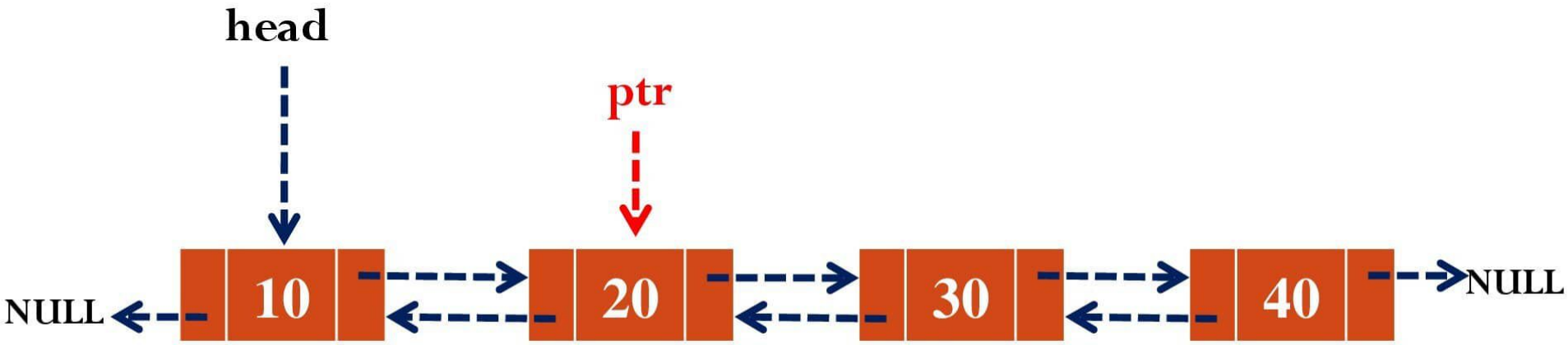
# Search 30



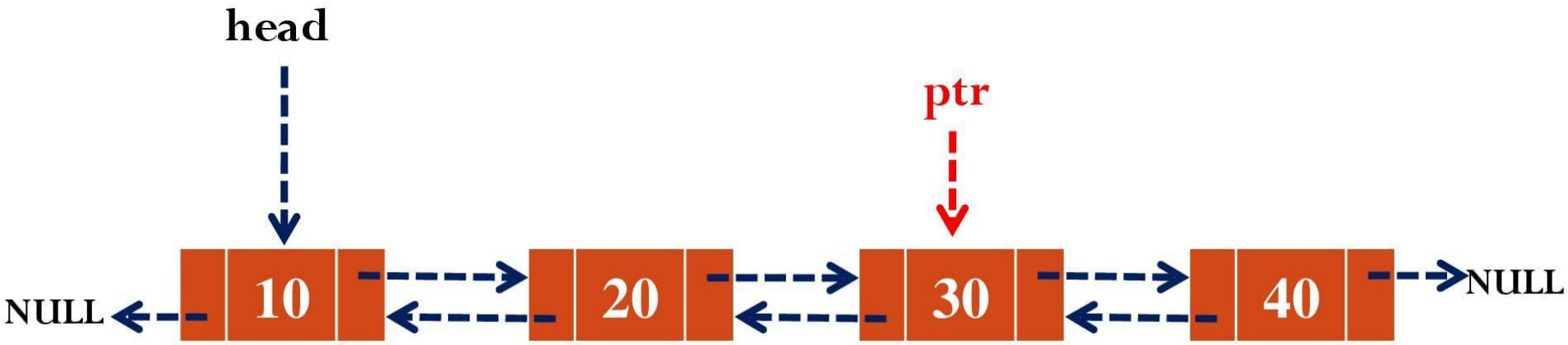
# Search 30



# Search 30



# Search 30

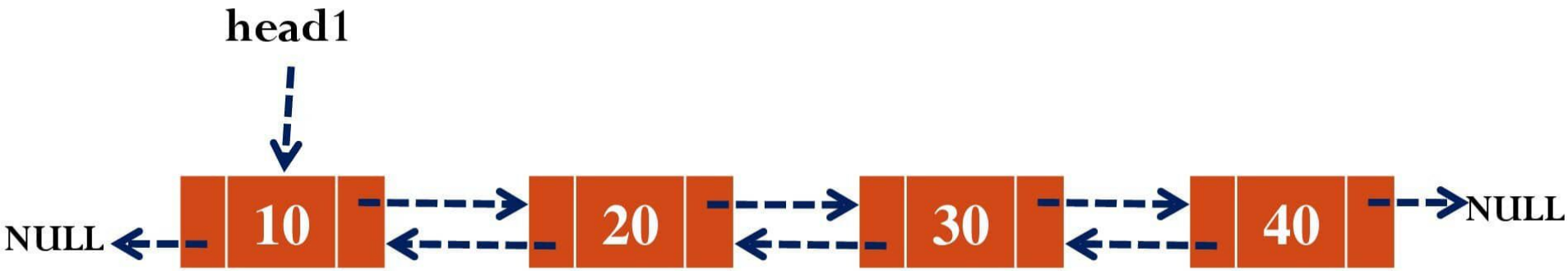


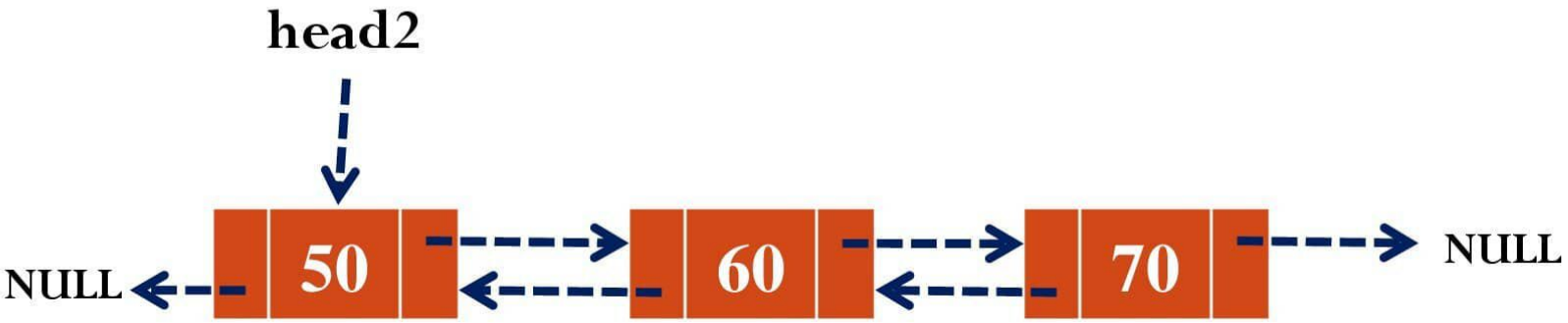
# Search~ Algorithm

**Algorithm Search (head, key)**

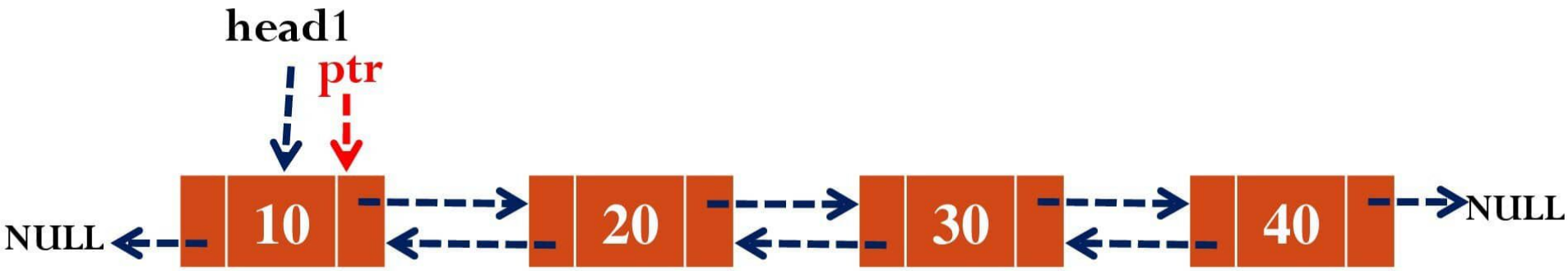
1. If head ==NULL then
    1. Print "List is empty"
2. Else
    1. ptr=head
    2. While ptr→data!=key and ptr→Rlink!=NULL do
        1. ptr=ptr→Rlink
    3. If ptr→data=key then
        1. Print "Search data found.
    4. Else
        1. Print "Search data not found.

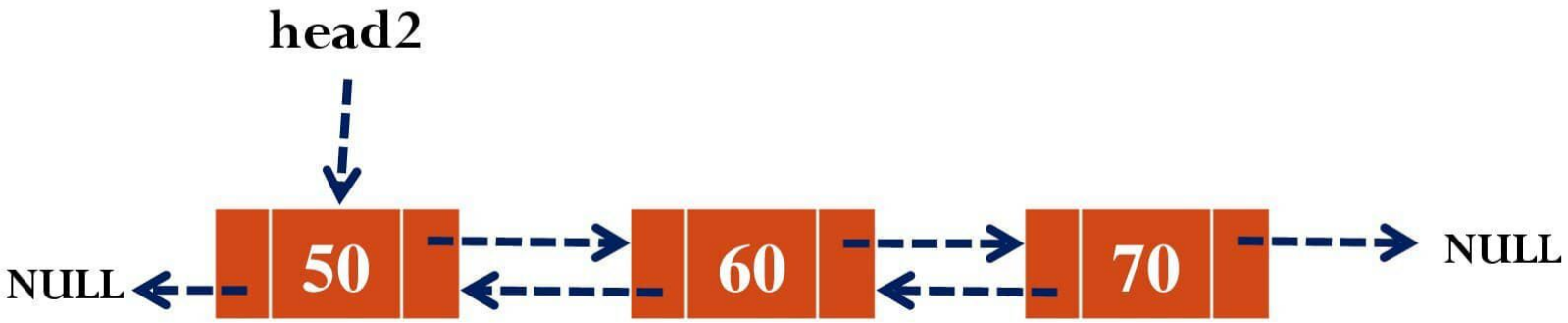
# Merge



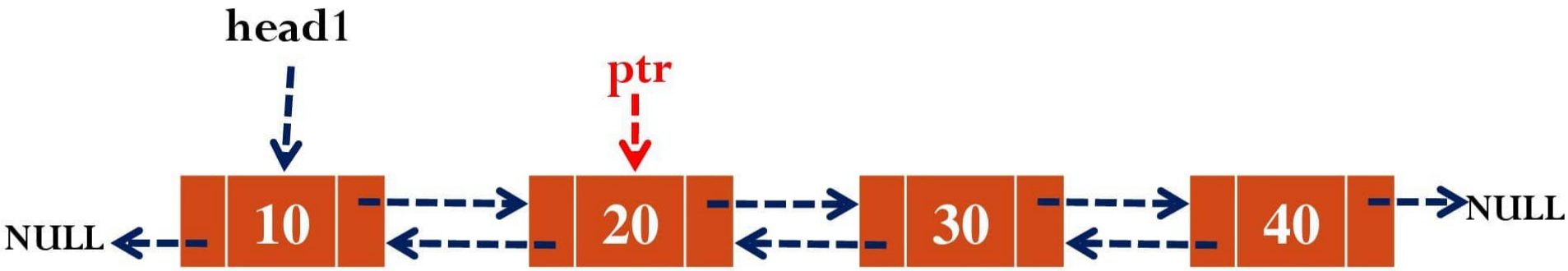



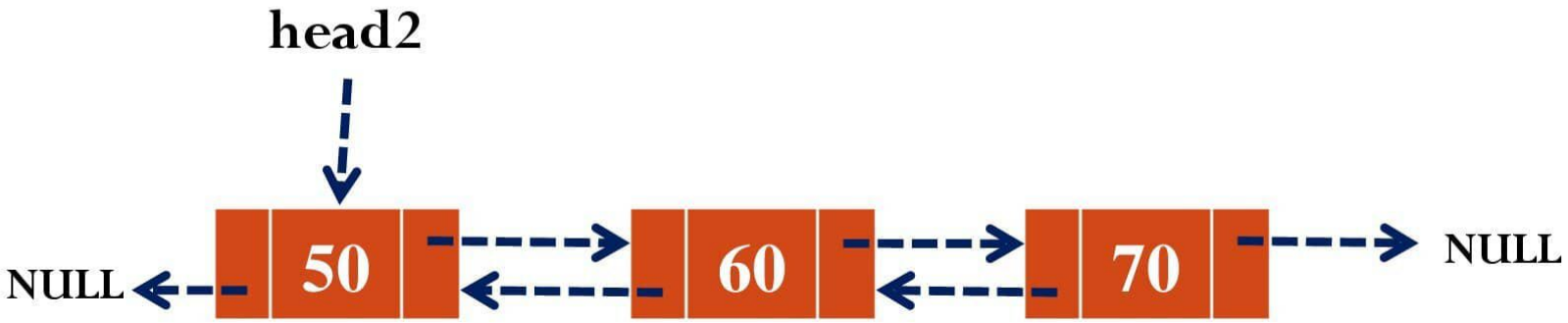
# Merge



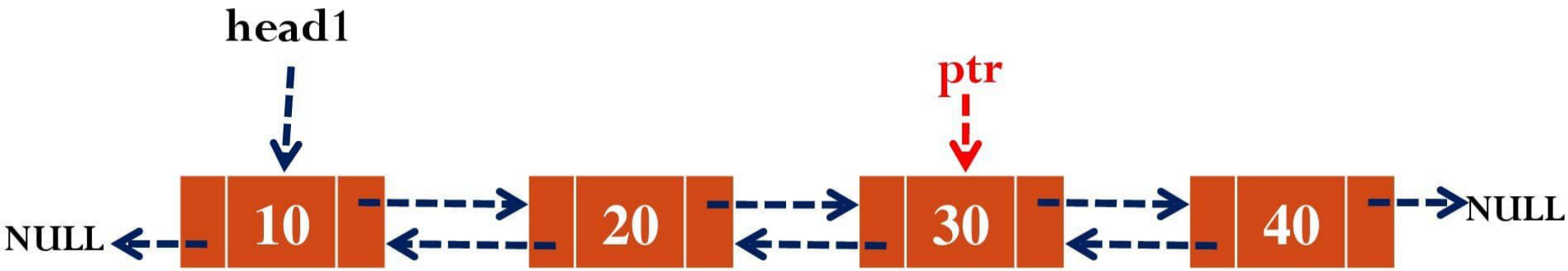



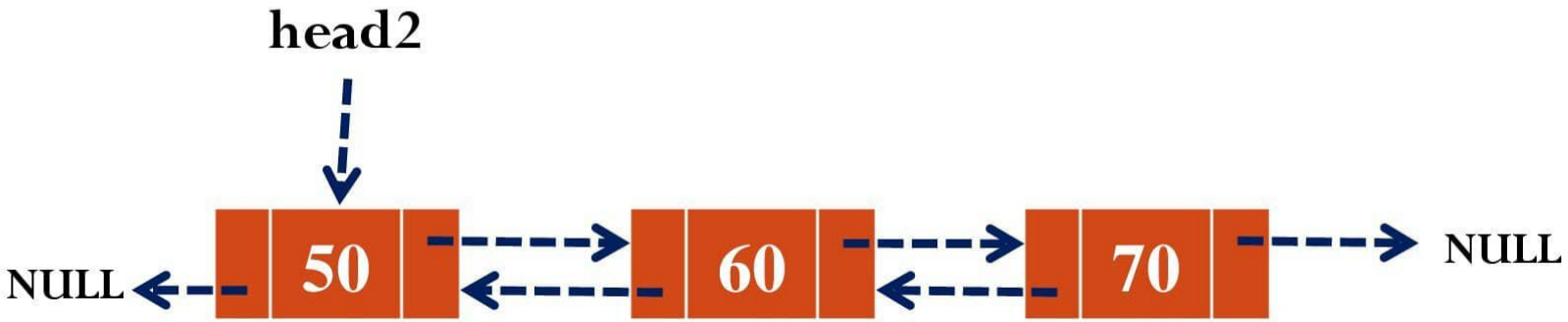
# Merge



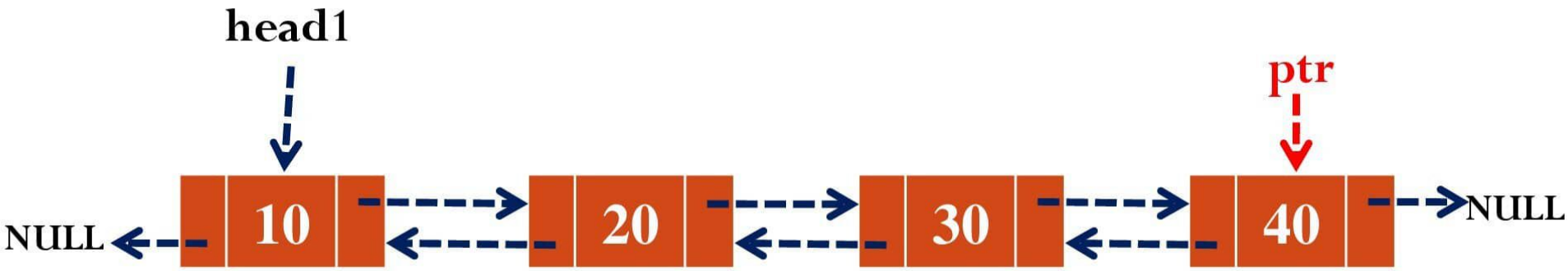



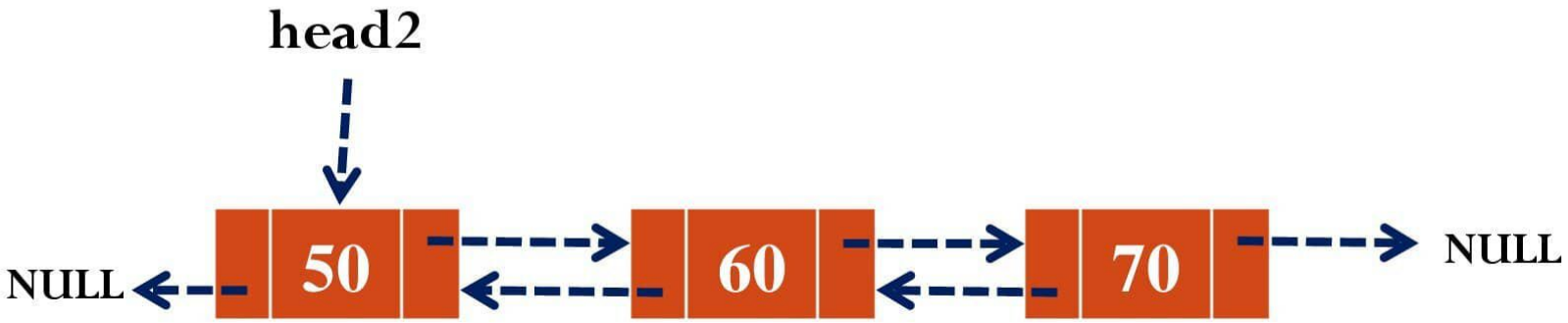
# Merge



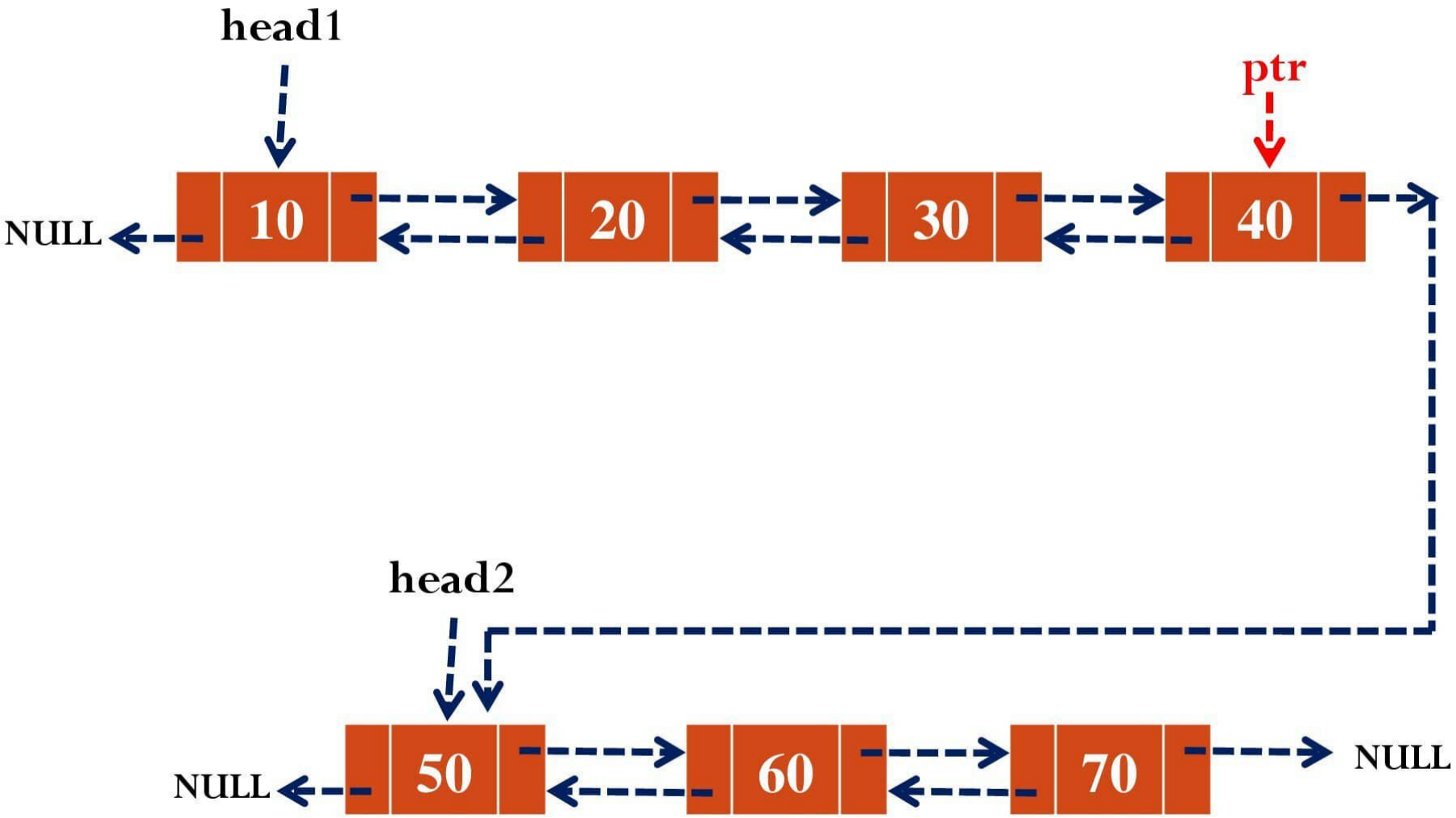



# Merge

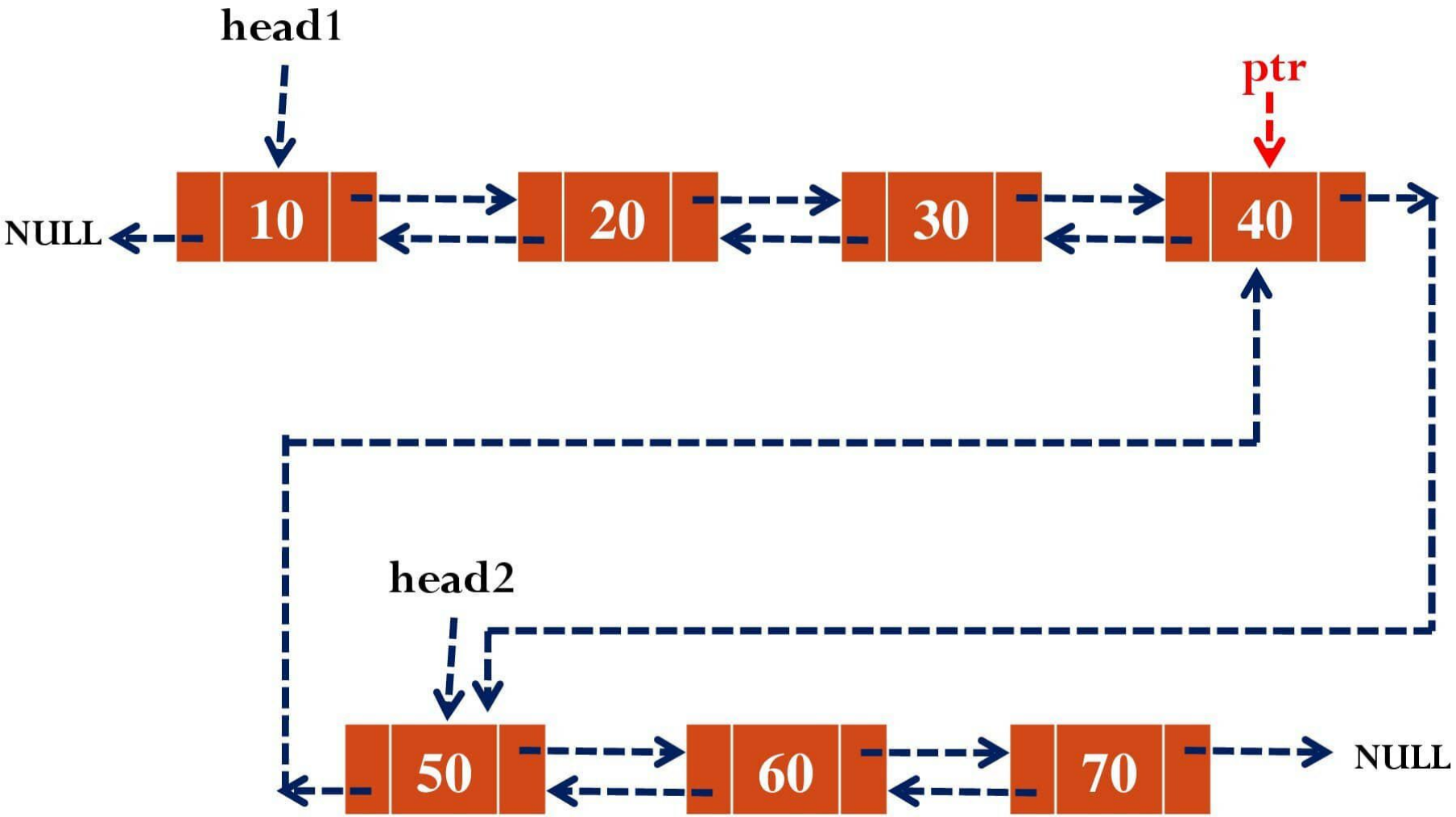


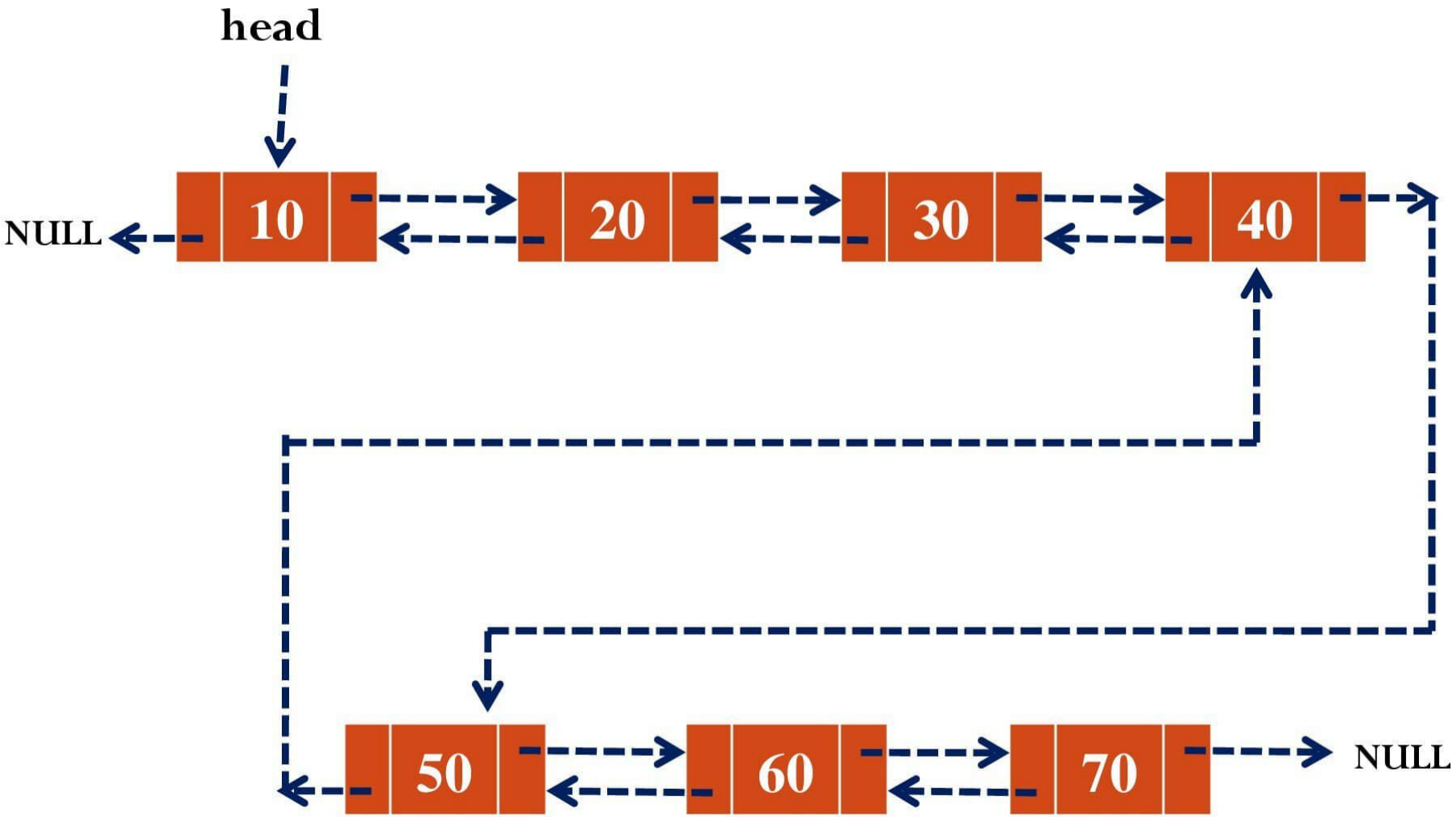
# Merge



# Merge



# Merge



# Merge- Algorithm

**Algorithm Merge (head1, head2)**

1. ptr=head1
2. While ptr→Rlink!=NULL do
    1. ptr=ptr→Rlink
3. ptr→Rlink=head2
4. head2→Llink=ptr
5. head=head1