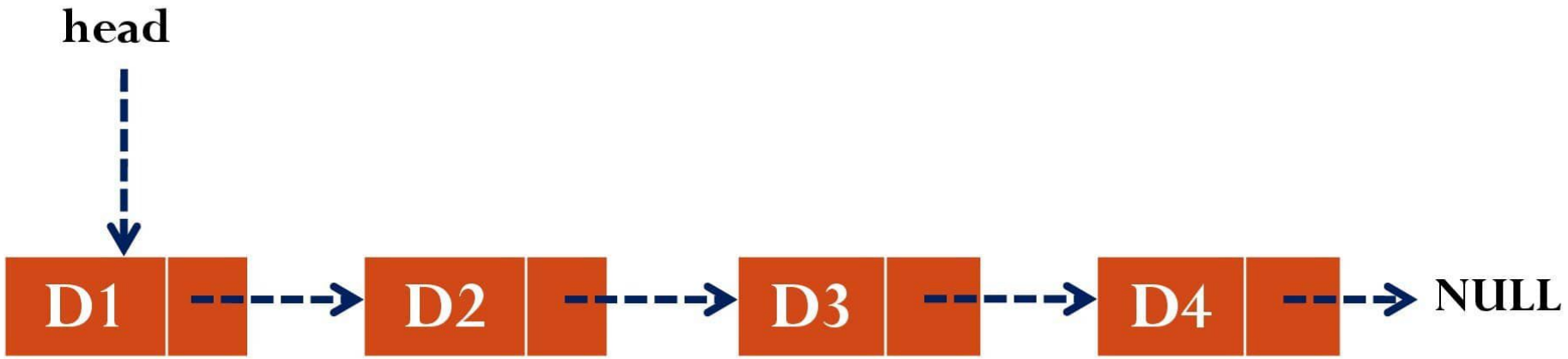
Data Structures – CST 201
Module - 3

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

Singly Linked List

- Each node contains only one link which points the subsequent node in the list



Node Creation

Algorithm struct node

1. Declare int data, node link

Program


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

Operations on Singly 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

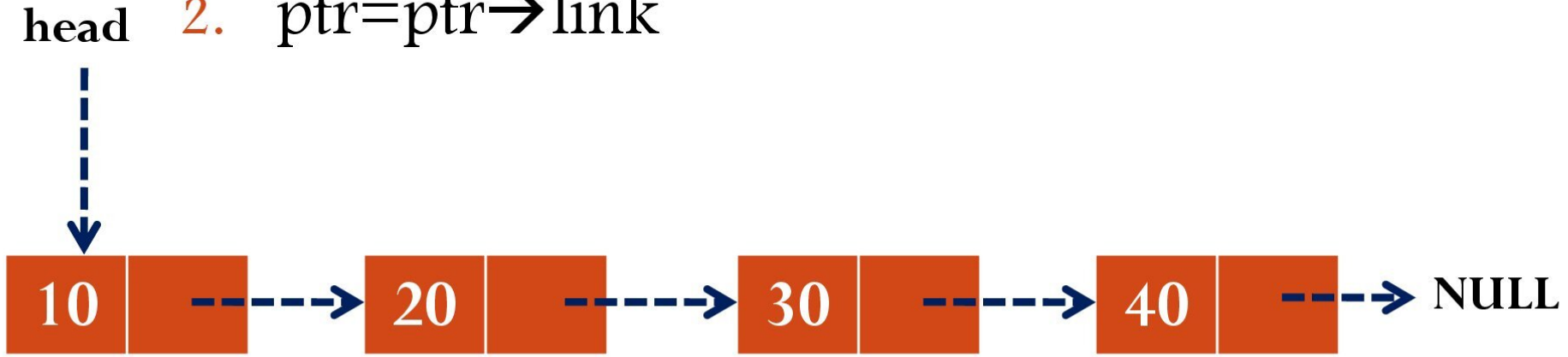
Traversal/Display

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

Display/Traversal - Algorithm

Algorithm Display(head)

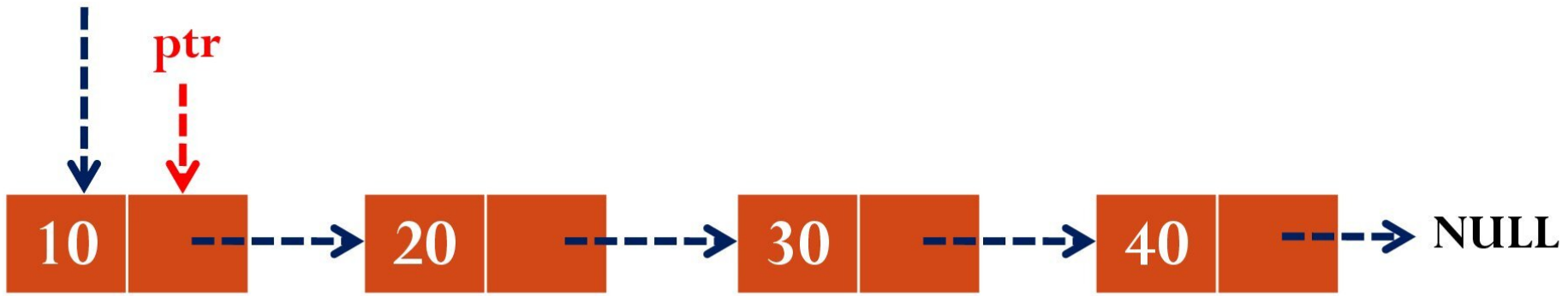
1. If head=NULL then
    1. Print "List is Empty"
2. Else
    1. ptr=head
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display/Traversal - Algorithm

Algorithm Display(head)

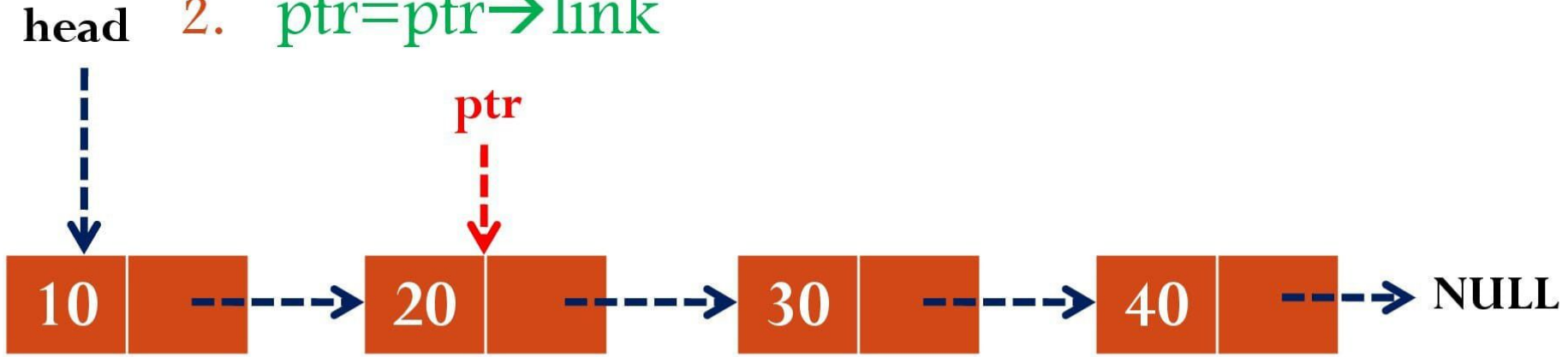
1. If head=NULL then
    1. Print "List is Empty"
2. Else
    1. ptr=head
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display/Traversal - Algorithm

Algorithm Display(head)

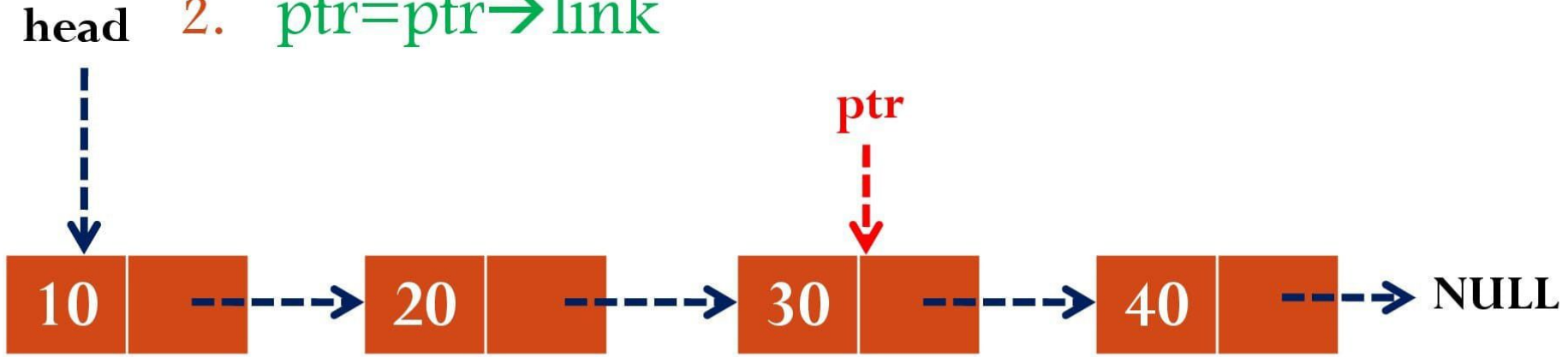
1. If head=NULL then
    1. Print "List is Empty"
2. Else
    1. ptr=head
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display/Traversal - Algorithm

Algorithm Display(head)

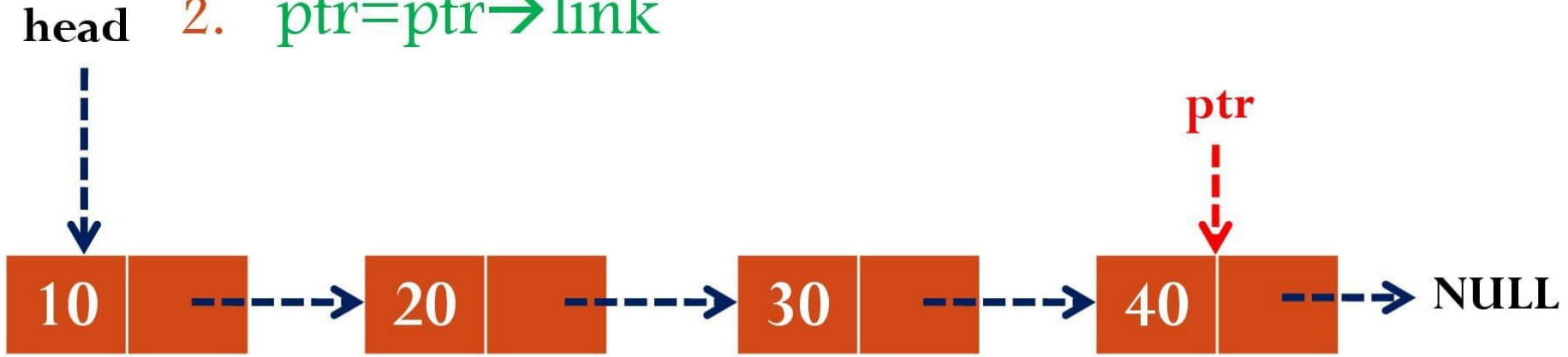
1. If head=NULL then
    1. Print "List is Empty"
2. Else
    1. ptr=head
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display/Traversal - Algorithm

Algorithm Display(head)

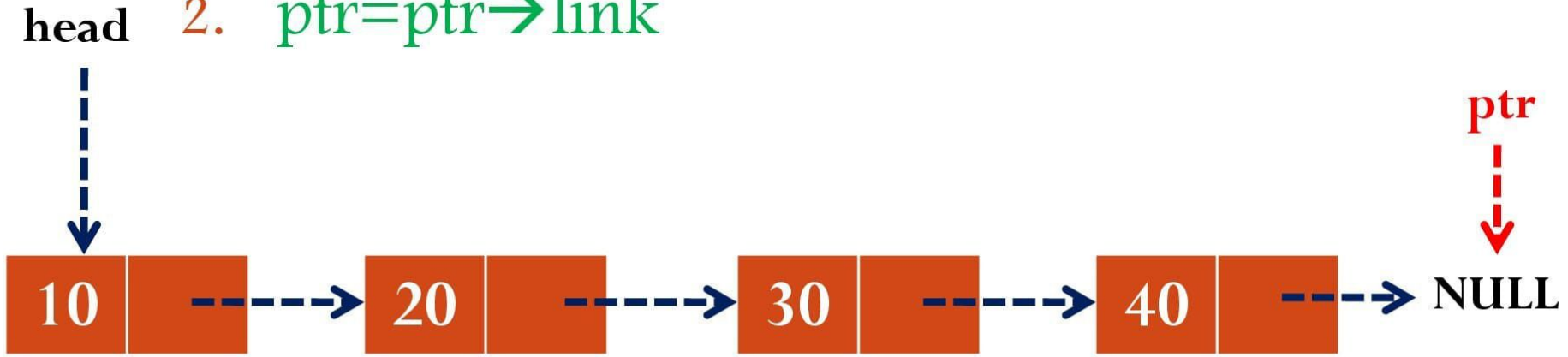
1. If head=NULL then
    1. Print "List is Empty"
2. Else
    1. ptr=head
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Display/Traversal - Algorithm

Algorithm Display(head)

1. If head=NULL then
    1. Print "List is Empty"
2. Else
    1. ptr=head
    2. While ptr!=NULL do
        1. Print ptr→data
        2. ptr=ptr→link



Insertion

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

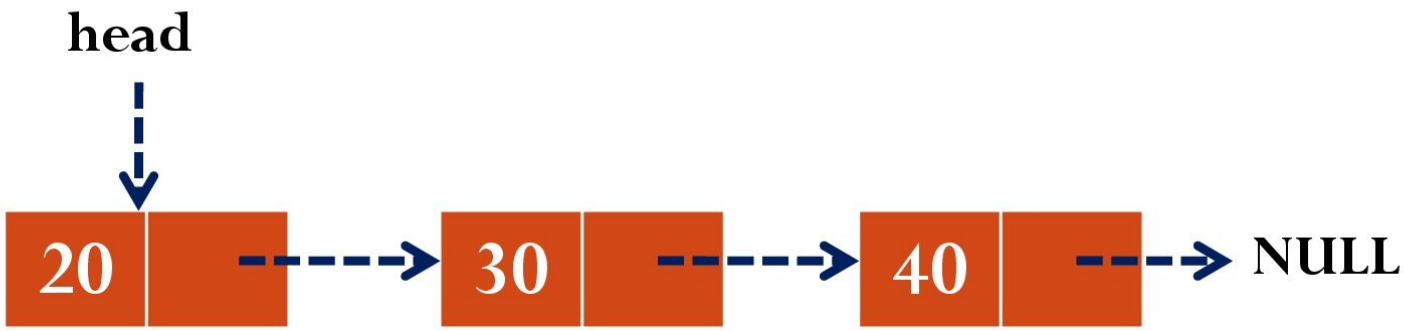
Insertion

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

Insert at Front - Algorithm

Algorithm Insert_Front(head, x)

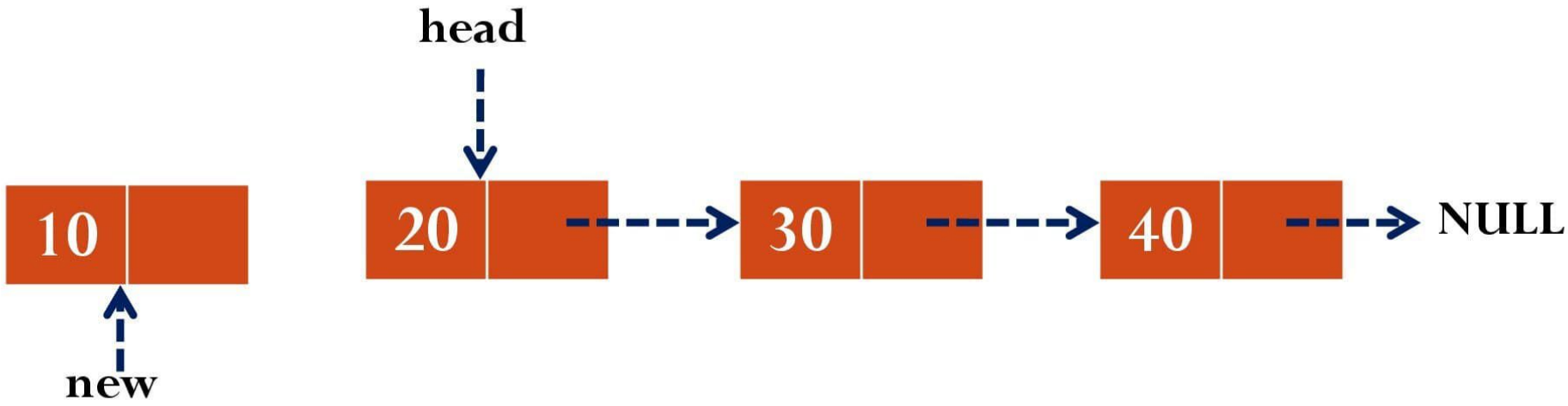
1. Create a node new
2. new→data=x
3. new→link=head
4. head=new



Insert at Front - Algorithm

Algorithm Insert_Front(head, x)

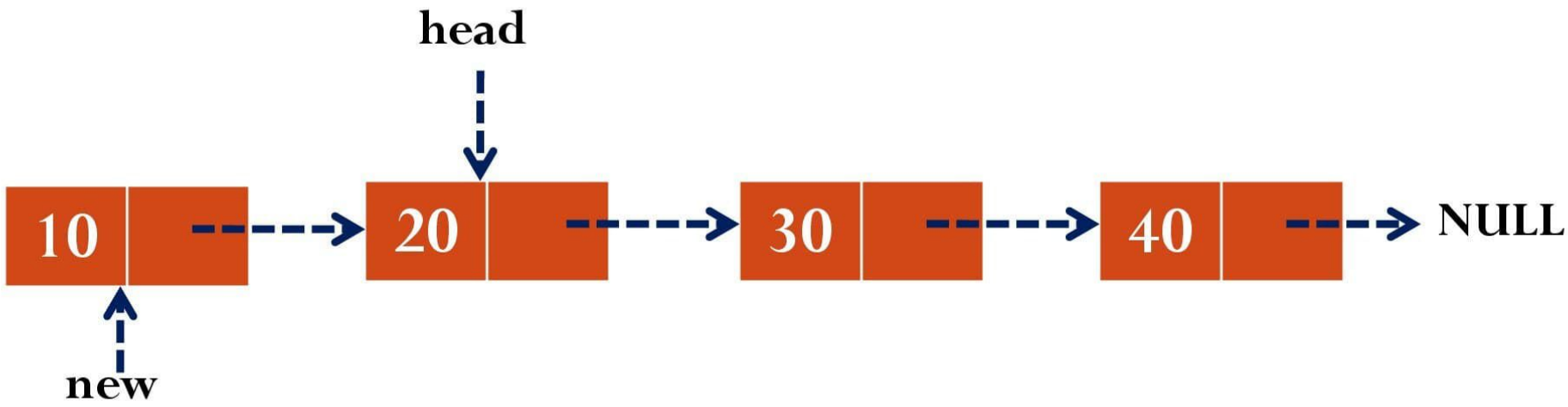
1. Create a node new
2. new→data=x
3. new→link=head
4. head=new



Insert at Front - Algorithm

Algorithm Insert_Front(head, x)

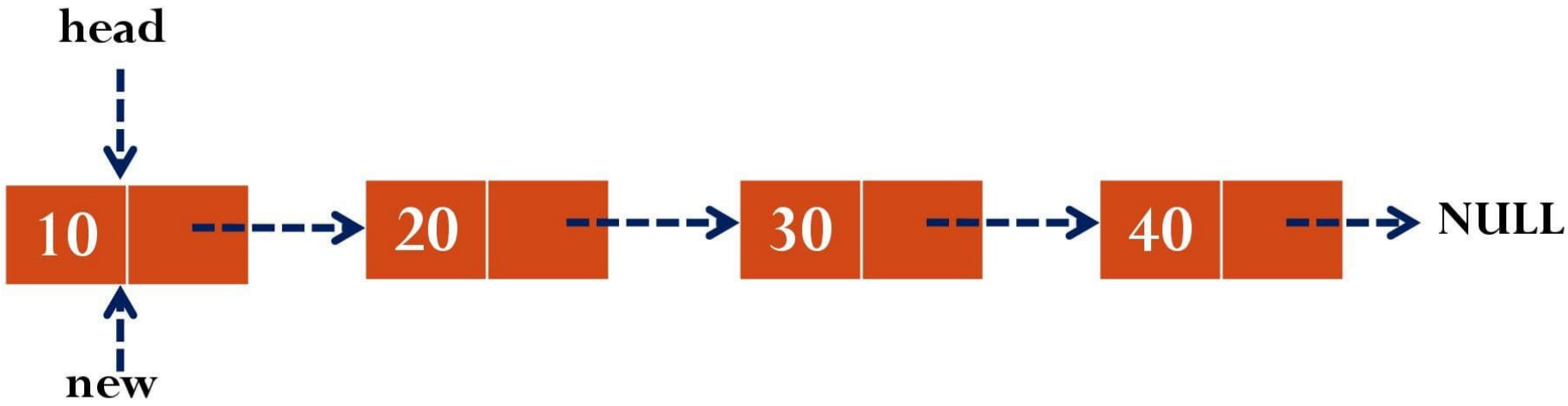
1. Create a node new
2. new→data=x
3. **new→link=head**
4. head=new



Insert at Front - Algorithm

Algorithm Insert_Front(head, x)

1. Create a node new
2. new→data=x
3. new→link=head
4. **head=new**



Insertion

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

Insert at End - Algorithm

Algorithm Insert_End(head, x)

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

Insert at End

2 cases:

1. List is empty
2. List is not empty

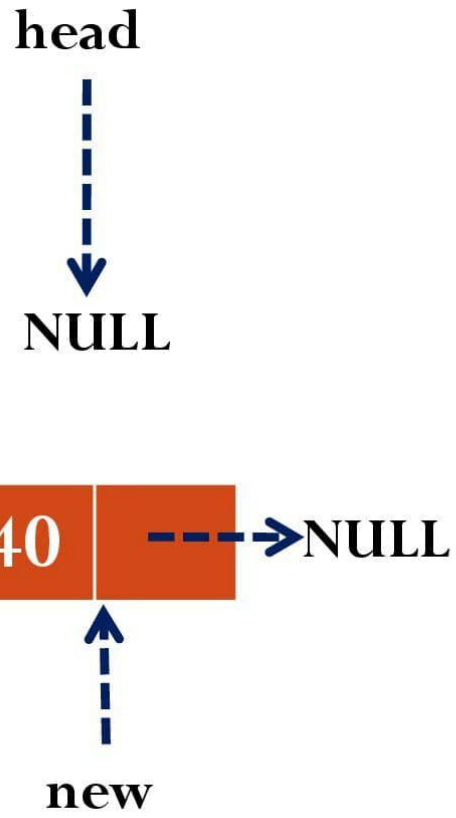
Insert at End

2 cases:

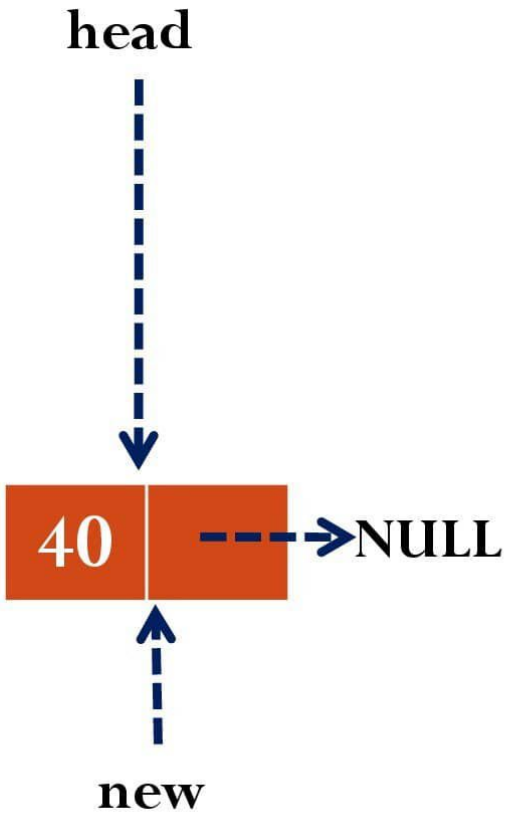
1. **List is empty**
2. List is not empty

Insert at End

Insert at End



Insert at End



Insert at End - Algorithm

Algorithm Insert_End(head, x)

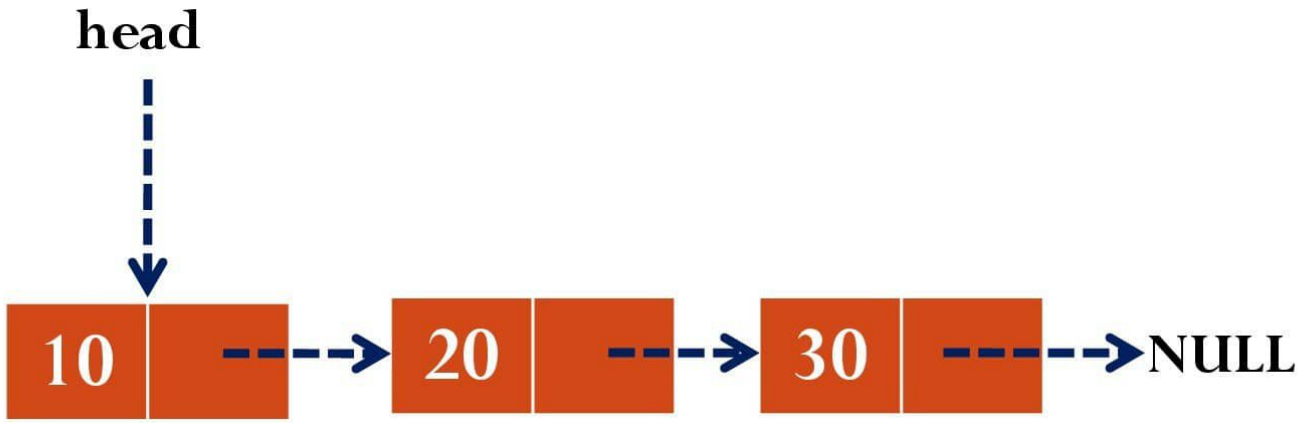
1. Create a node new
2. new→data=x
3. new→link=NULL
4. If head=NULL then
    1. head=new
5. Else
    1. ptr=head
    2. While(ptr→link!=NULL) do
        1. ptr=ptr→link
    3. ptr→link=new

Insert at End

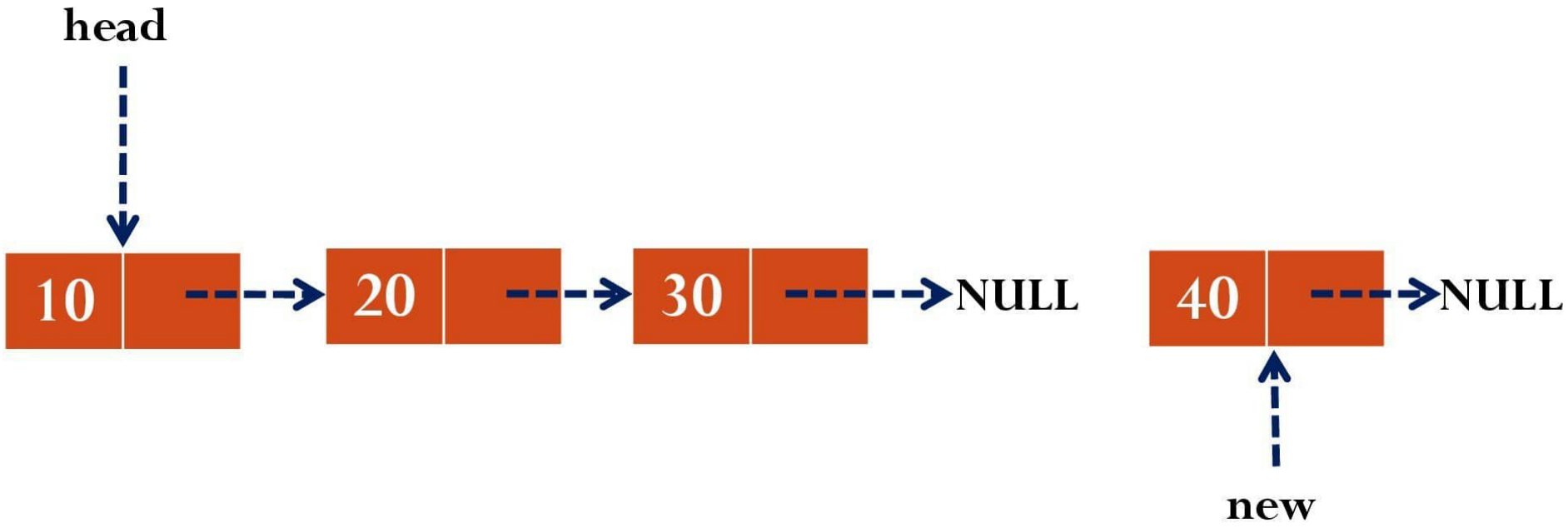
2 cases:

1. List is empty
2. **List is not empty**

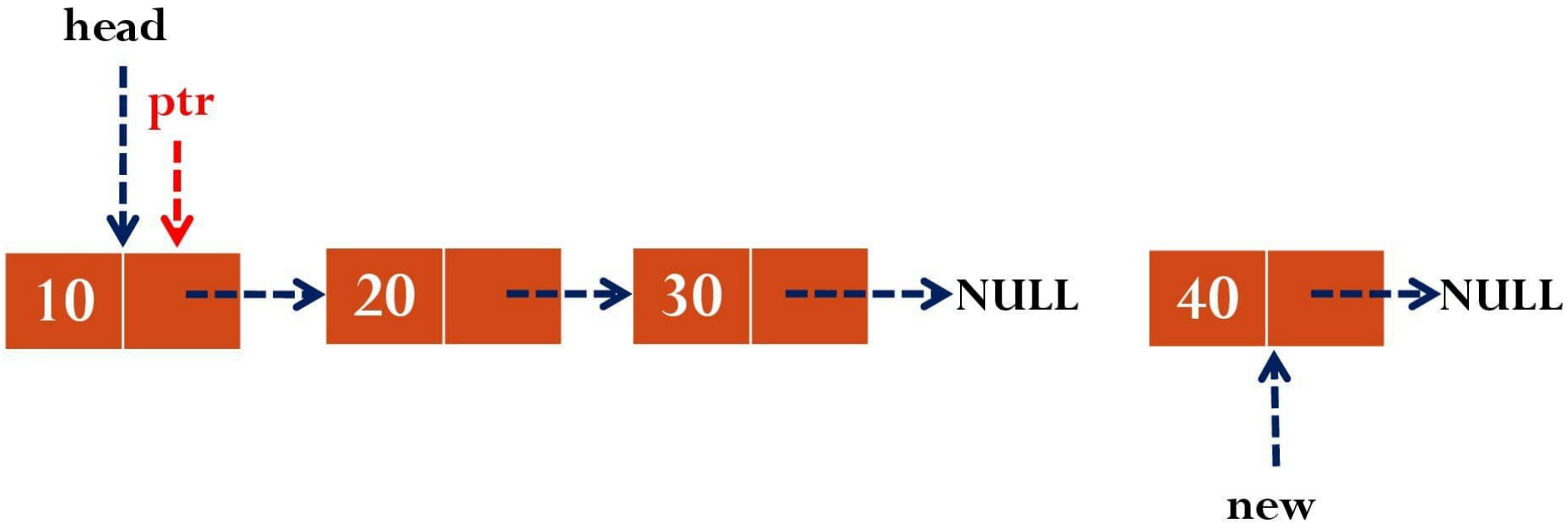
Insert at End



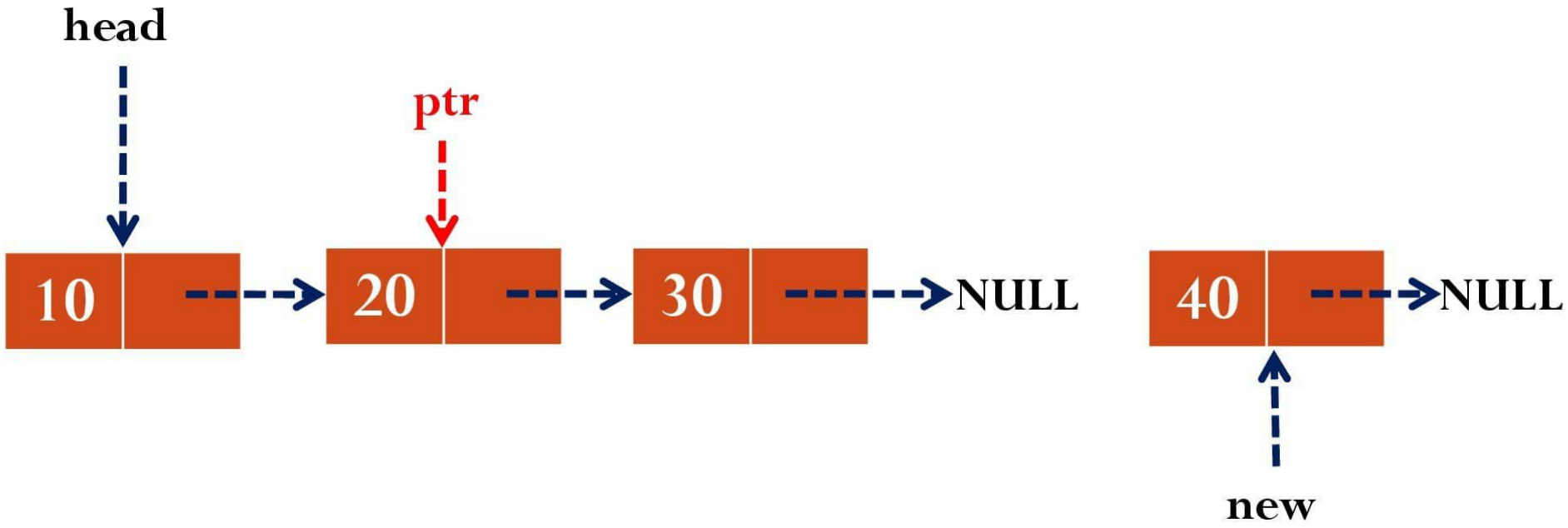
Insert at End



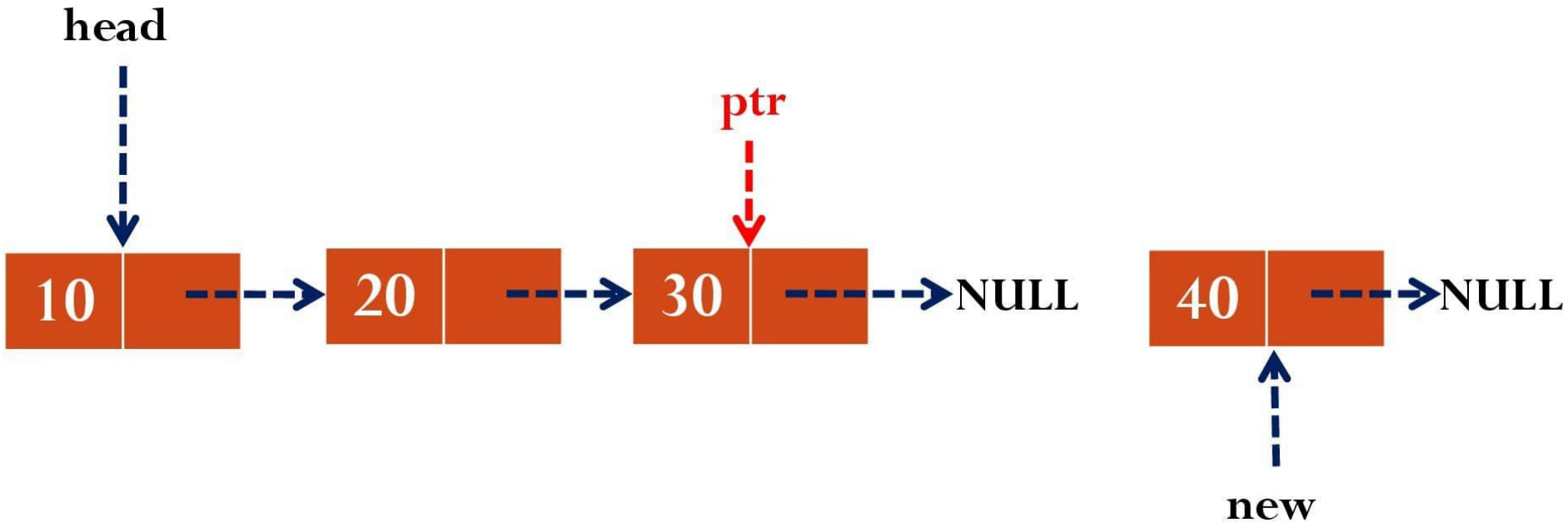
Insert at End



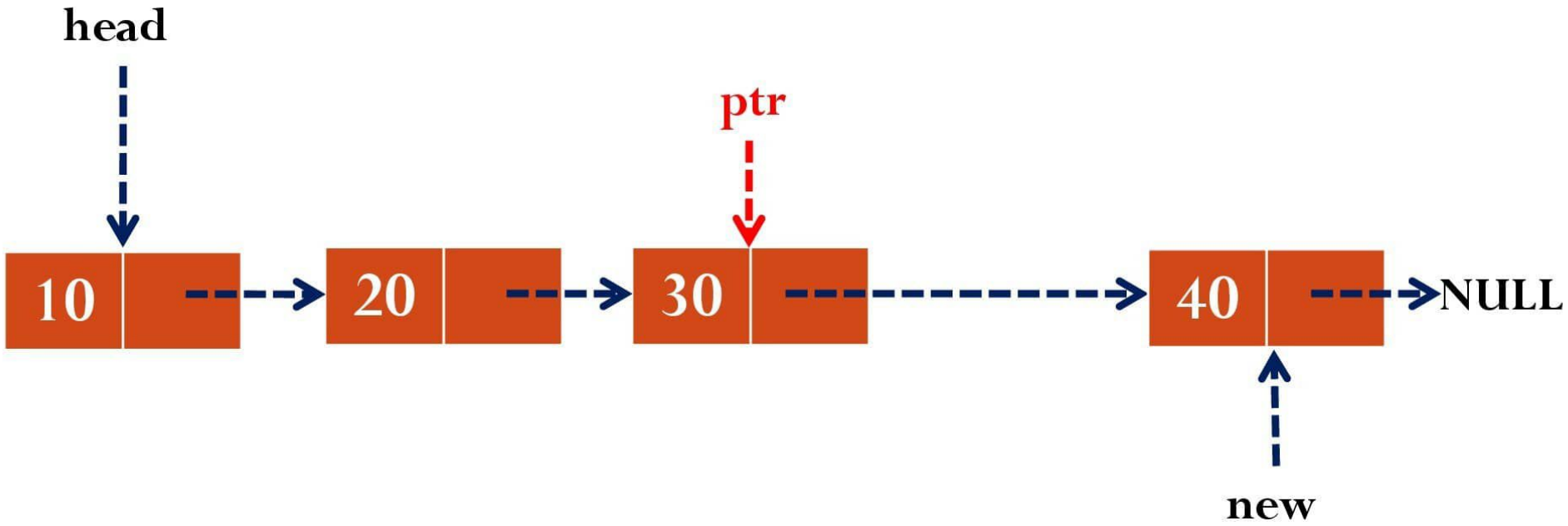
Insert at End



Insert at End



Insert at End



Insert at End - Algorithm

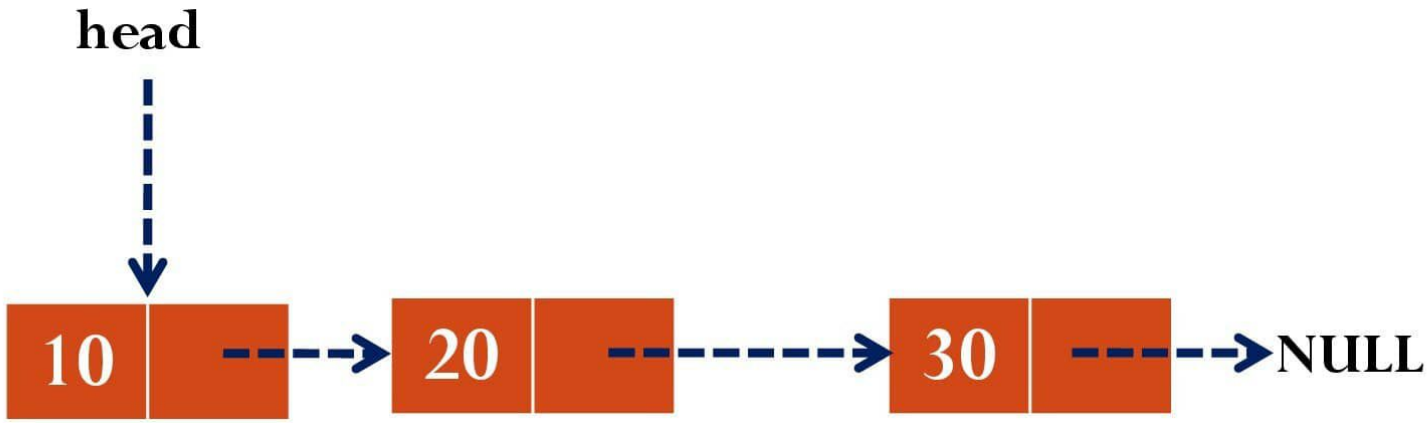
Algorithm Insert_End(head, x)

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

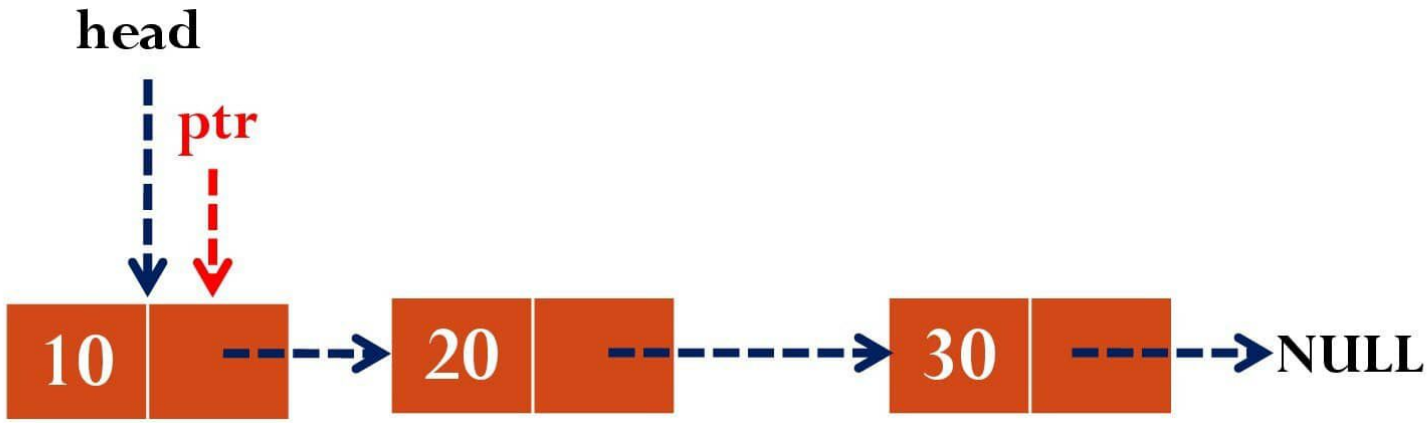
Insertion

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

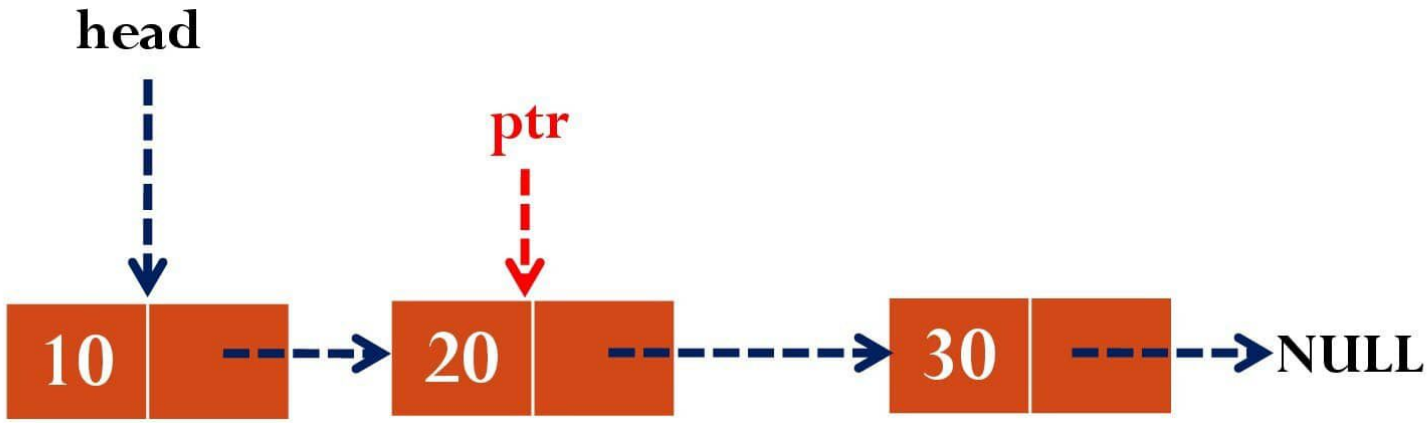
Insert after a specified node 20



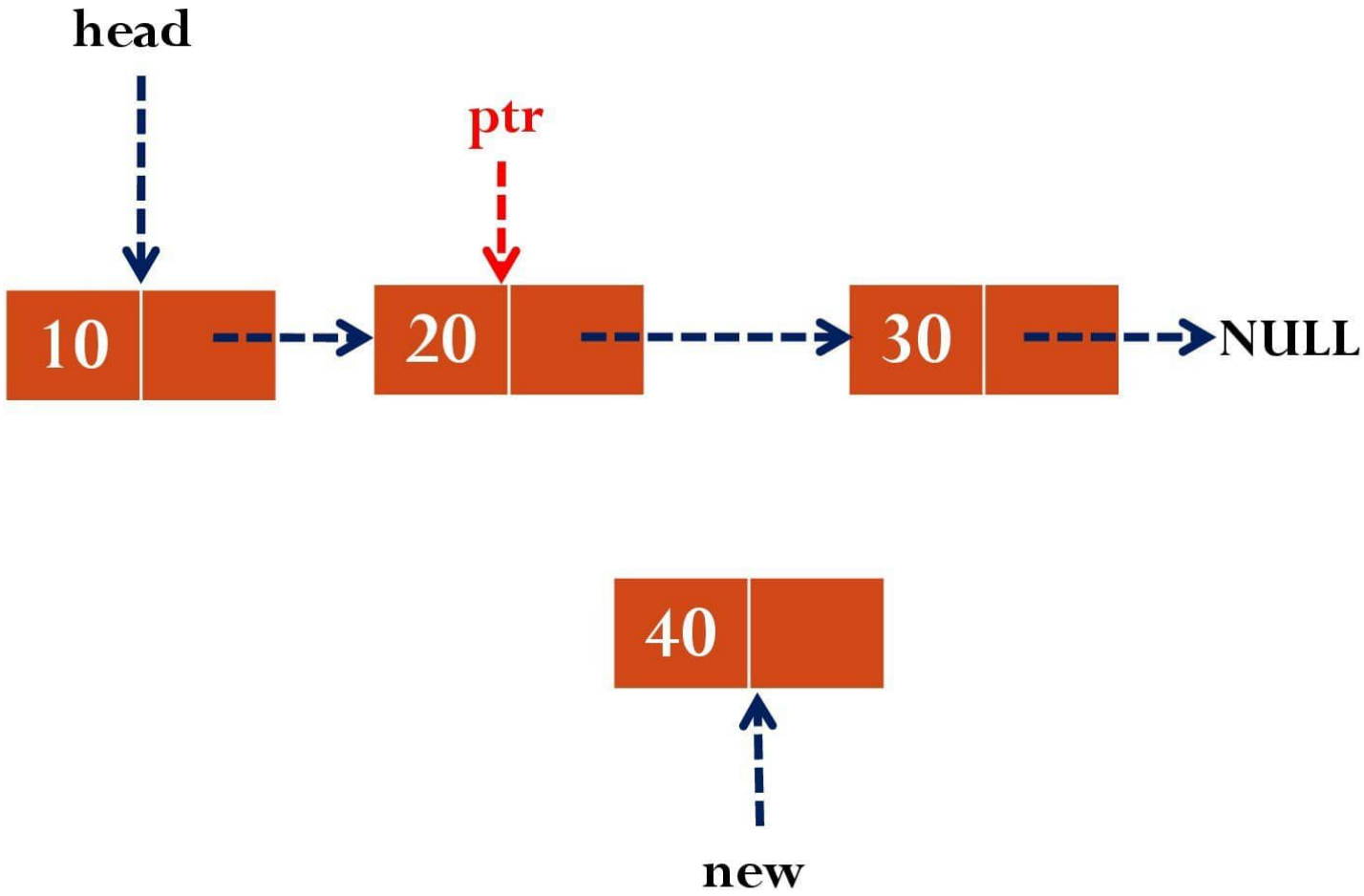
Insert after a specified node 20



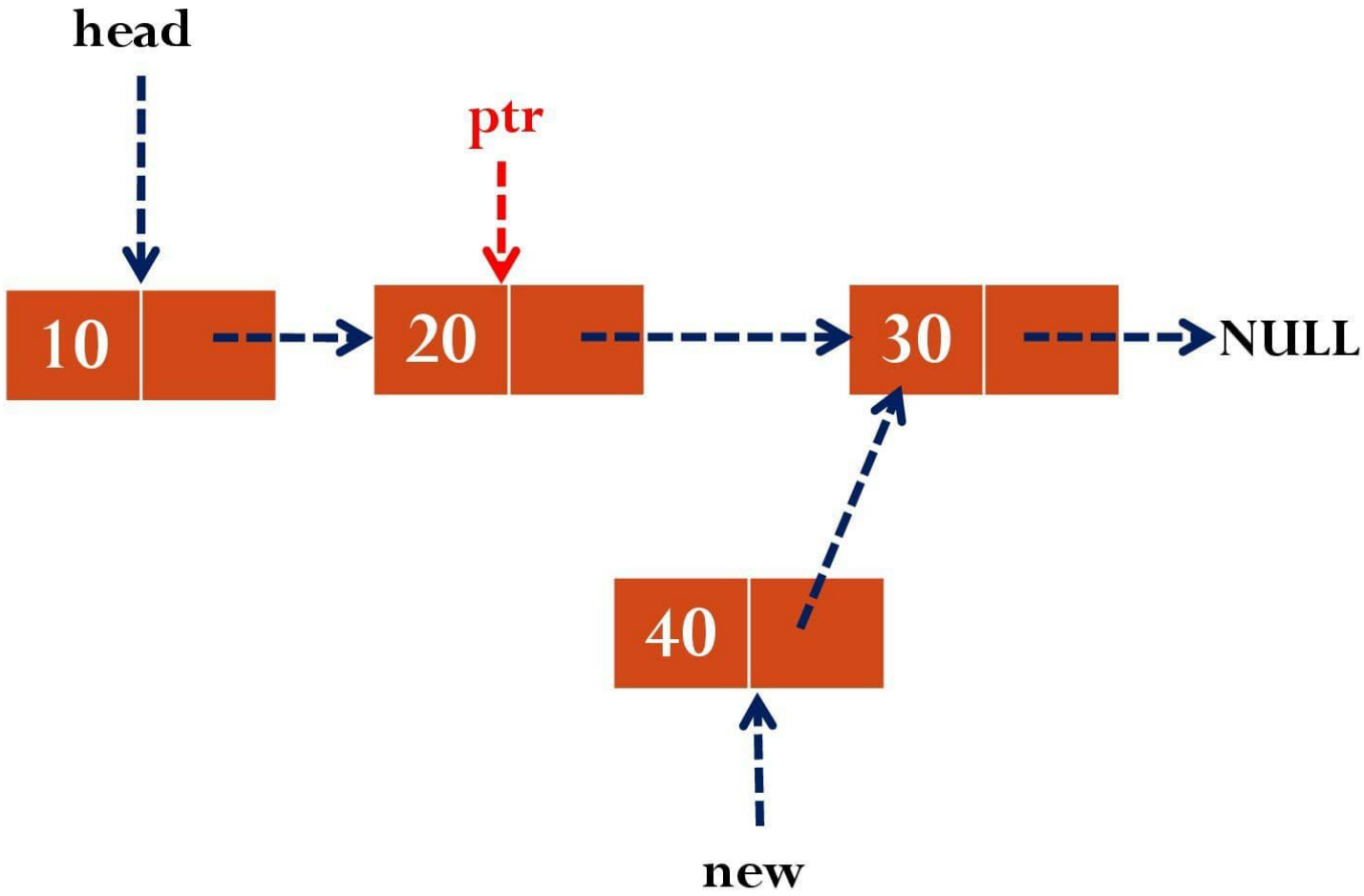
Insert after a specified node 20



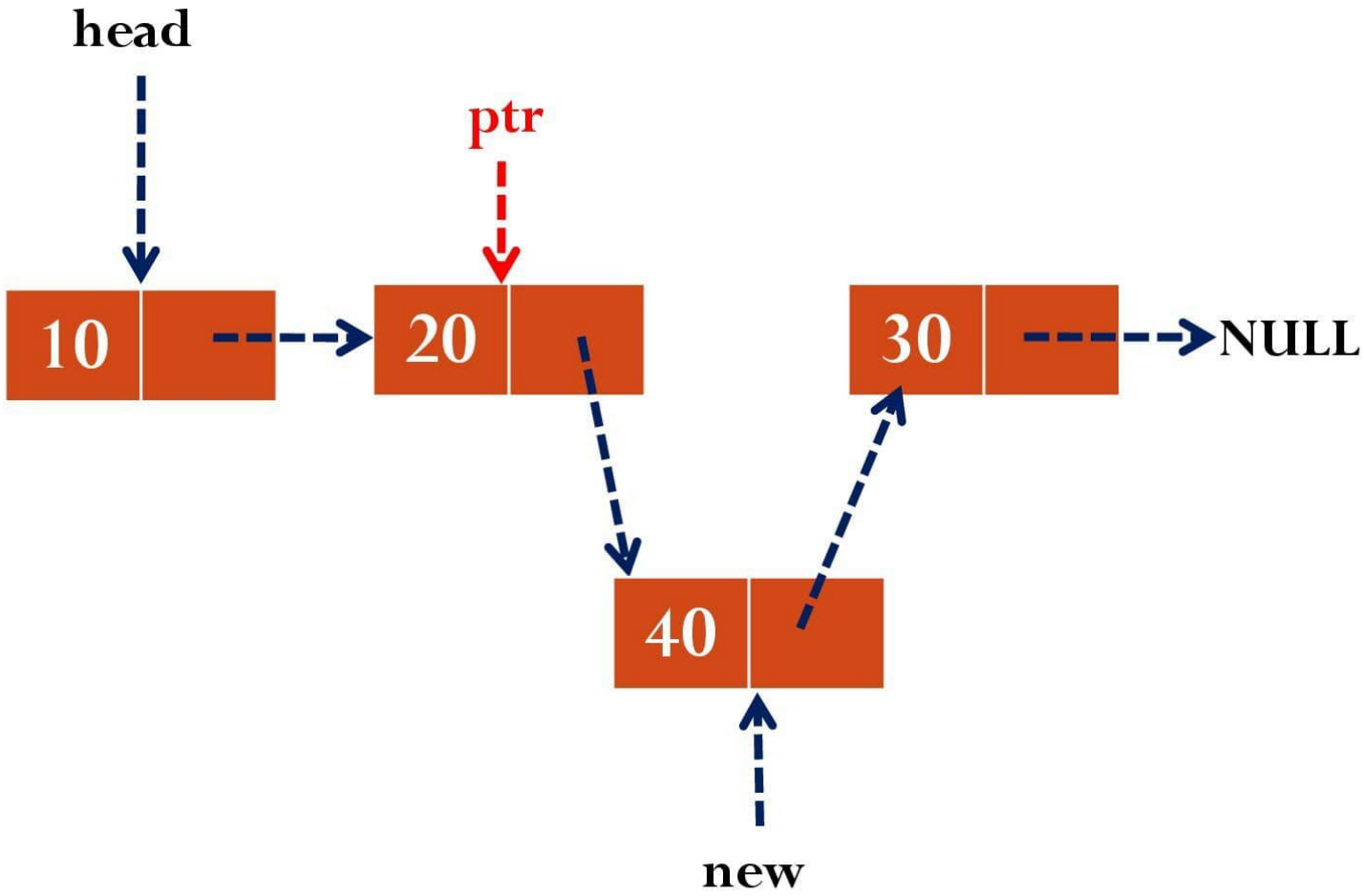
Insert after a specified node 20



Insert after a specified node 20



Insert after a specified node 20



Insert after a specified node - Algorithm

Algorithm Insert_After(head, key, x)

1. If head=NULL then
    1. Print “Search failed. Insertion is not possible”
2. Else
    1. ptr=head
    2. while(ptr→data!=key and ptr→link!=NULL) do
        1. ptr=ptr→link
    3. If ptr→data!=key then
        1. Print “Search failed. Insertion is not possible”
    4. Else
        1. Create a node new
        2. new→data=x
        3. new→link=ptr→link
        4. ptr→link = new

Deletion

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

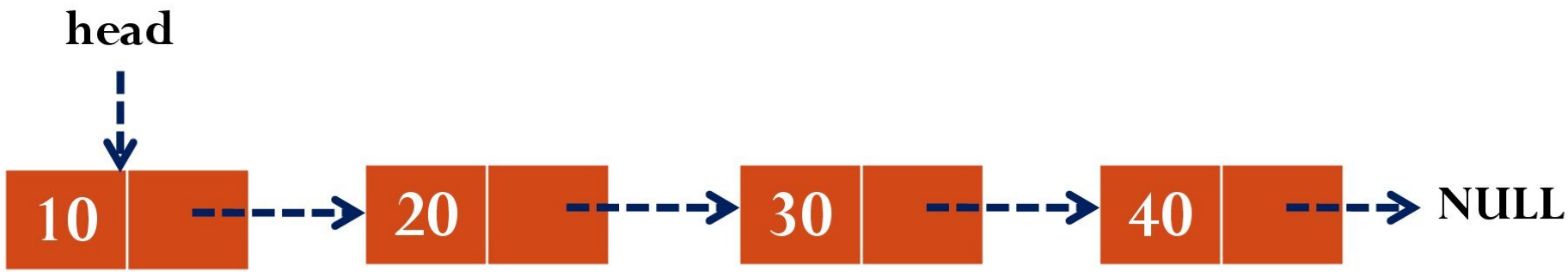
Deletion

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

Delete from Front- Algorithm

Algorithm Delete_Front(head)

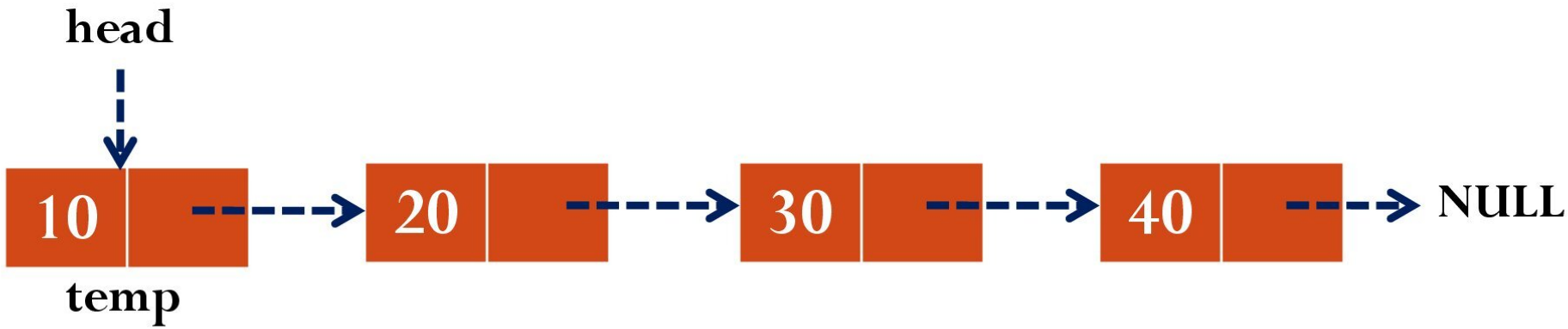
1. If head ==NULL then
    1. Print “List is empty”
2. Else
    1. temp=head
    2. head=head→link
    3. dispose(temp)



Delete from Front- Algorithm

Algorithm Delete_Front(head)

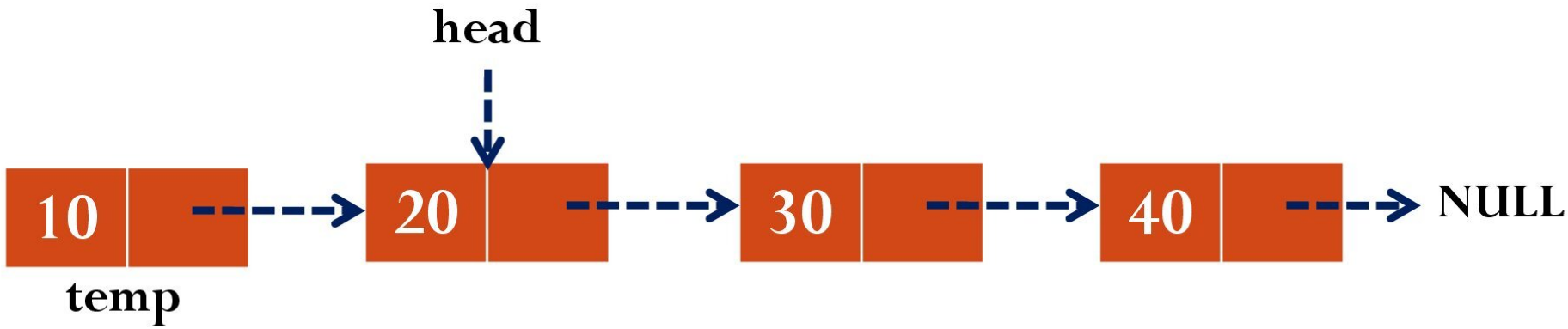
1. If head ==NULL then
    1. Print "List is empty"
2. Else
    1. temp=head
    2. head=head→link
    3. dispose(temp)



Delete from Front- Algorithm

Algorithm Delete_Front(head)

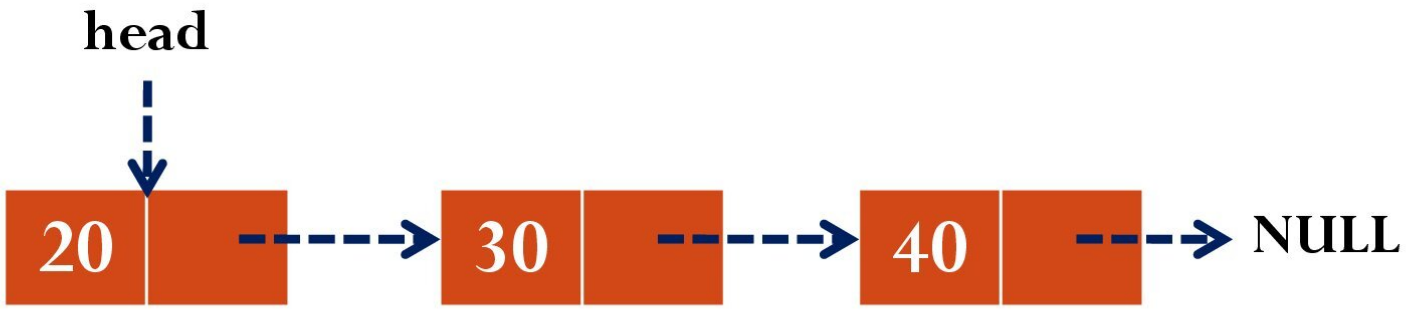
1. If head ==NULL then
    1. Print "List is empty"
2. Else
    1. temp=head
    2. head=head→link
    3. dispose(temp)



Delete from Front- Algorithm

Algorithm Delete_Front(head)

1. If head ==NULL then
    1. Print "List is empty"
2. Else
    1. temp=head
    2. head=head→link
    3. dispose(temp)



Deletion

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

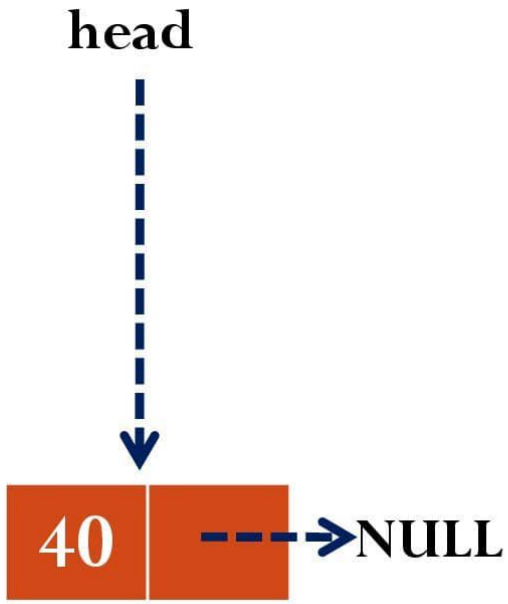
Delete from End

3 cases

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

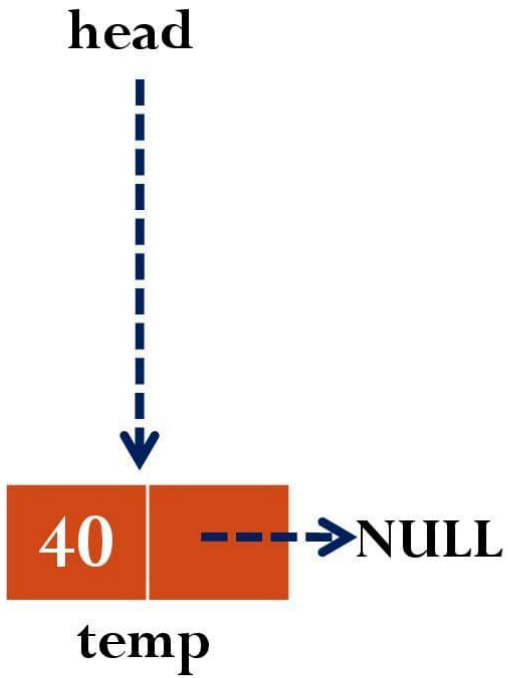
Delete from End

Case 2



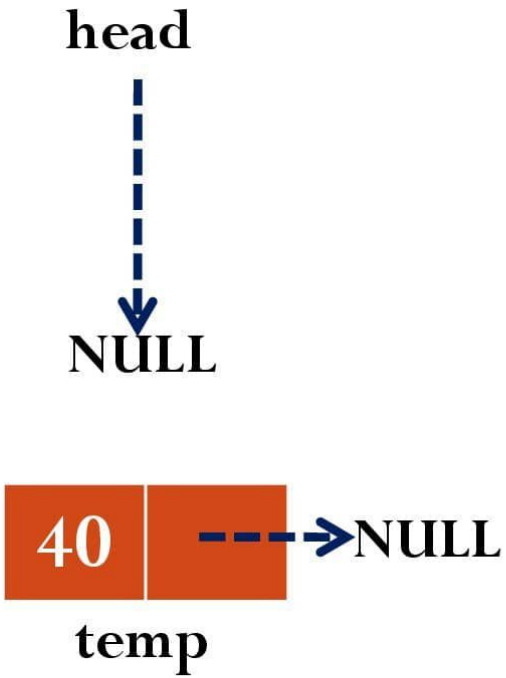
Delete from End

Case 2



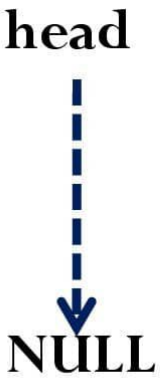
Delete from End

Case 2



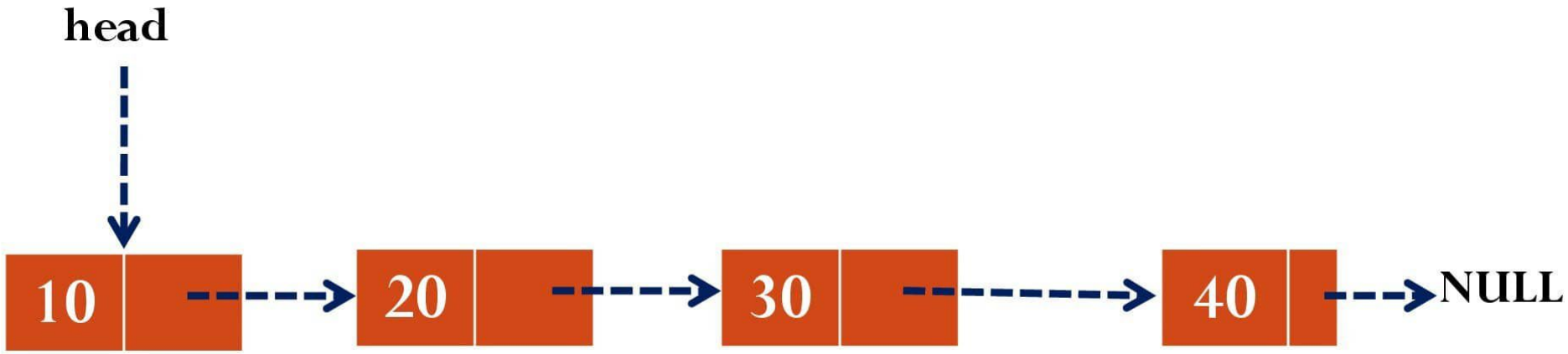
Delete from End

Case 2



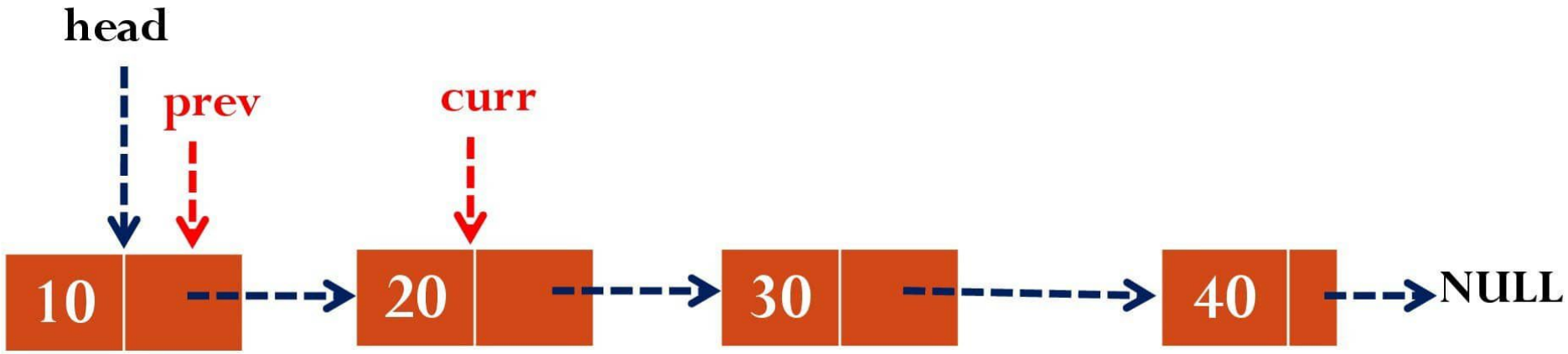
Delete from End

Case 3



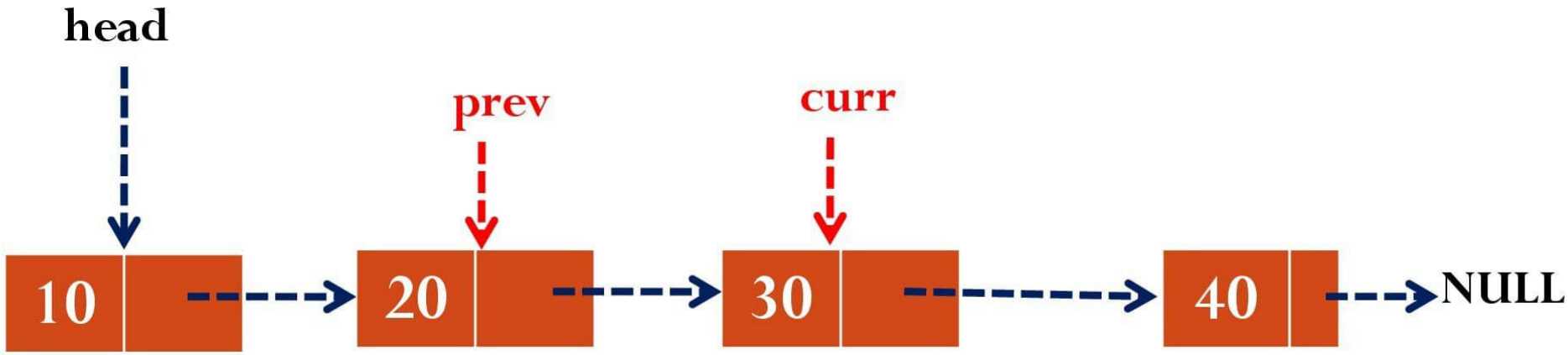
Delete from End

Case 3



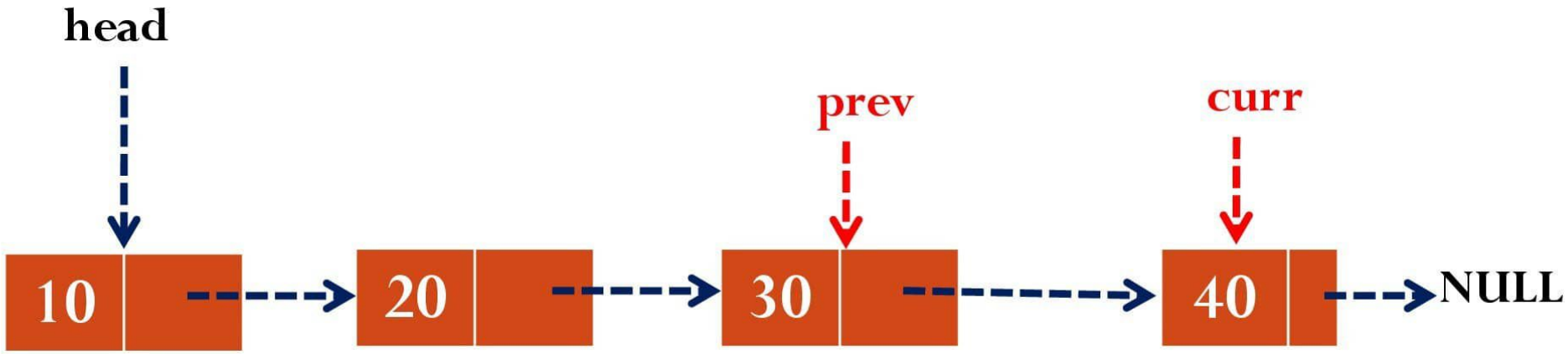
Delete from End

Case 3



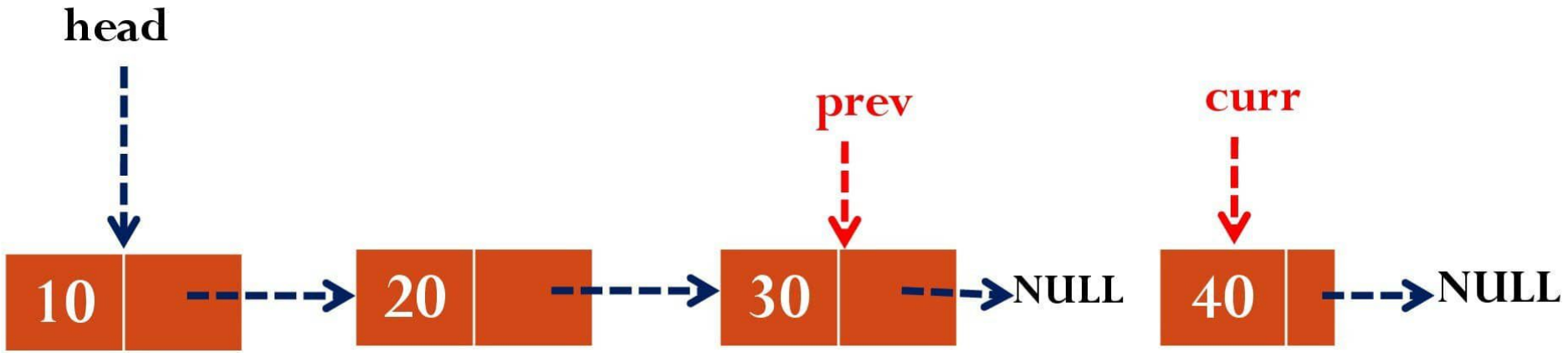
Delete from End

Case 3



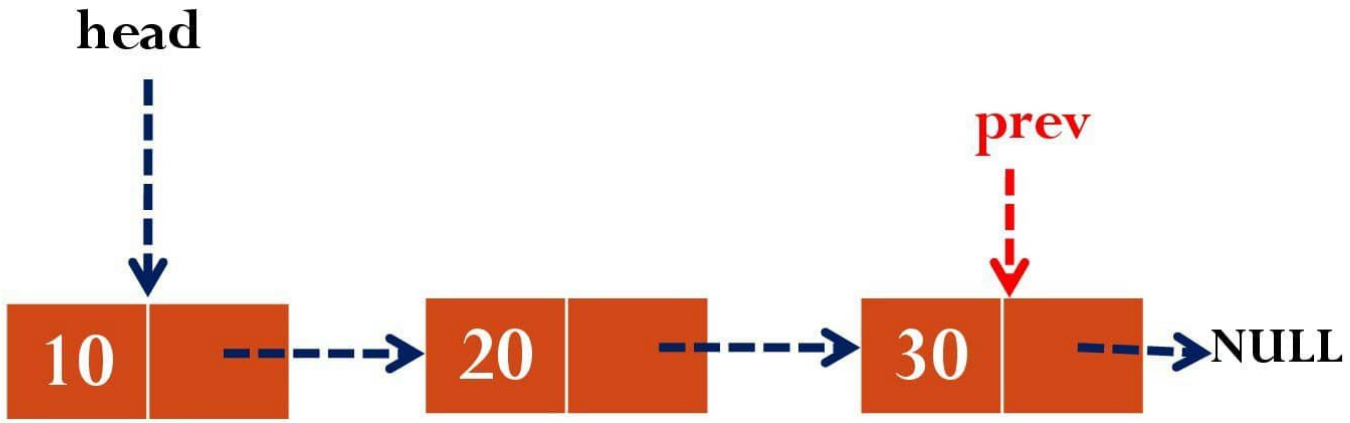
Delete from End

Case 3



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→link=NULL then
    1. temp=head
    2. head=NULL
    3. dispose(temp)
3. Else
    1. prev = head
    2. curr = head→link
    3. while curr→link !=NULL do
        1. prev = curr
        2. curr = curr→link
    4. prev→link=NULL
    5. dispose(curr)

Deletion

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

Delete specified node

Three cases

1. List is empty
2. The search data present in the first node
3. All other cases

Delete specified node- Algorithm

Algorithm Delete_Any(head, key)

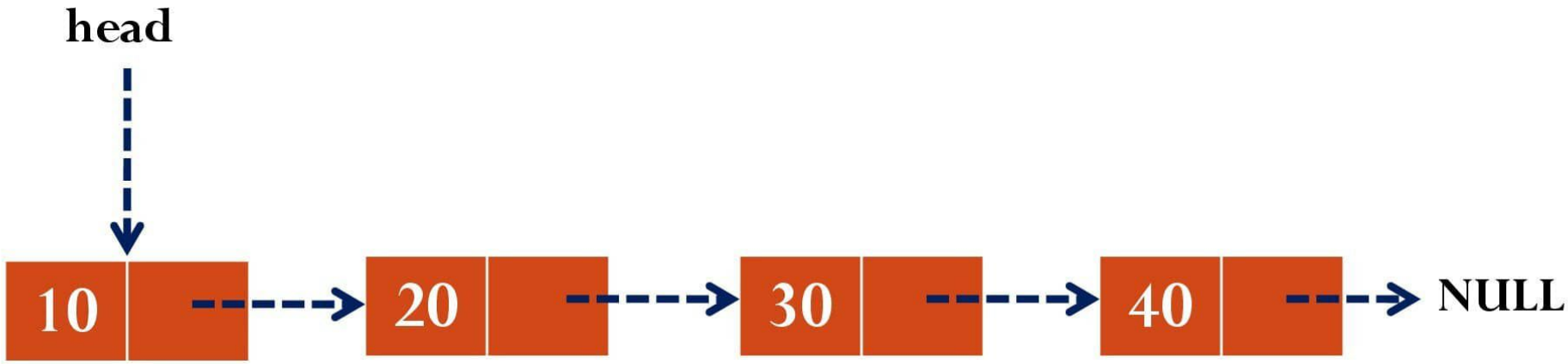
1. If head=NULL then
    1. Print "List is Empty"
2. Else if head→data = key then
    1. temp=head
    2. head=head→link
    3. Dispose(temp)

Delete specified node- Algorithm

3. Else
    1. prev=head
    2. curr=head
    3. while curr→data != key and curr→link != NULL do
        1. prev = curr
        2. curr = curr→link
    4. If curr→data != key then
        1. Print “Search key not found”
    5. Else
        1. prev→link = curr→link
        2. Dispose(curr)

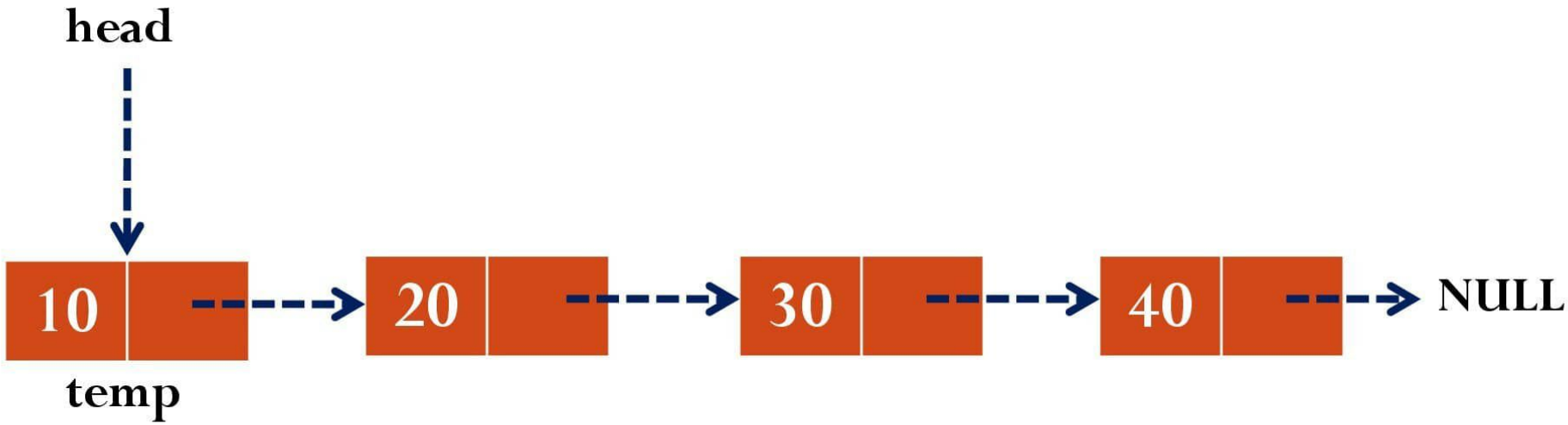
Delete Node 10

Case 2



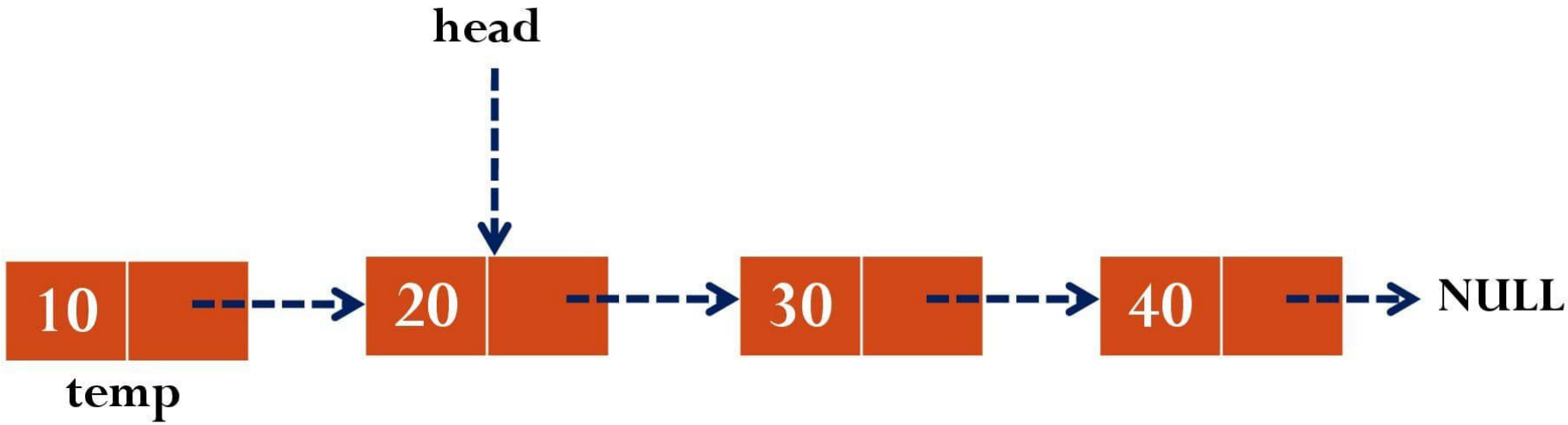
Delete Node 10

Case 2



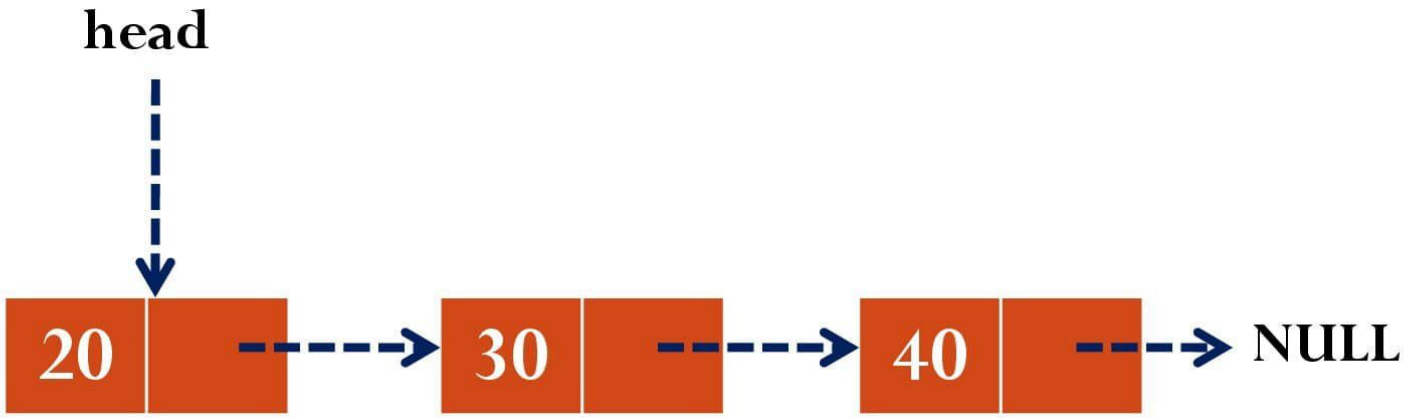
Delete Node 10

Case 2



Delete Node 10

Case 2



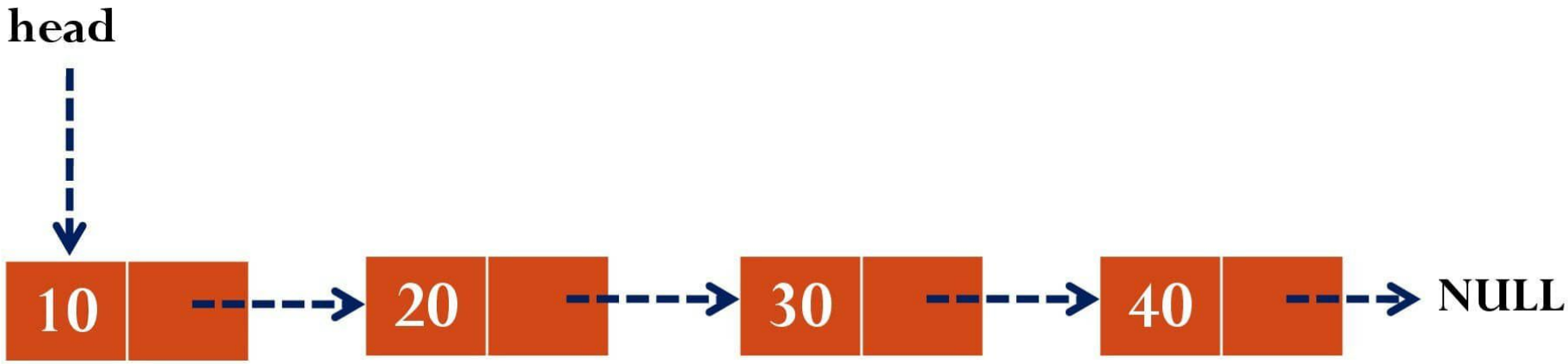
Delete specified node- Algorithm

Algorithm Delete_Any(head, key)

1. If head=NULL then
    1. Print "List is Empty"
2. Else if head→data = key then
    1. temp=head
    2. head=head→link
    3. dispose(temp)

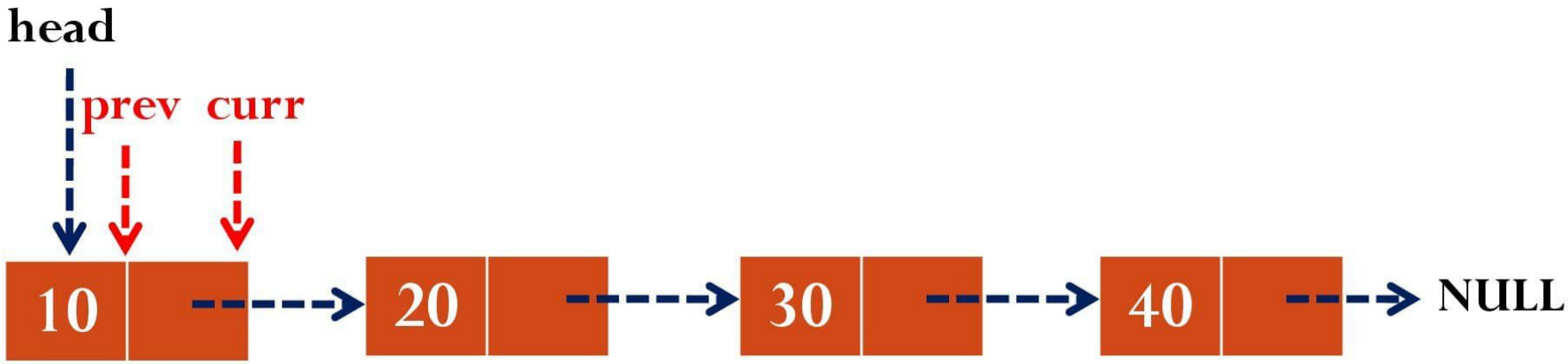
Delete Node 30

Case 3



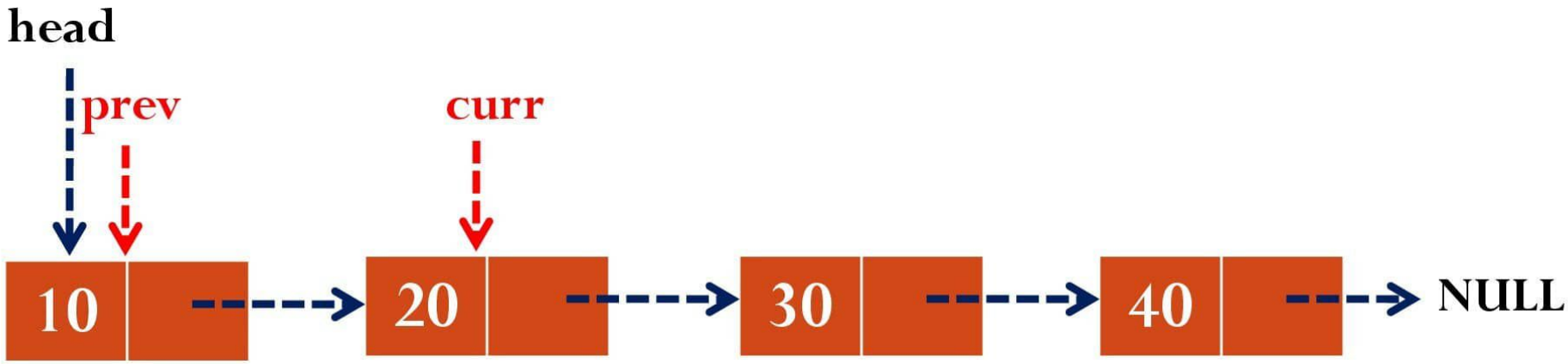
Delete Node 30

Case 3



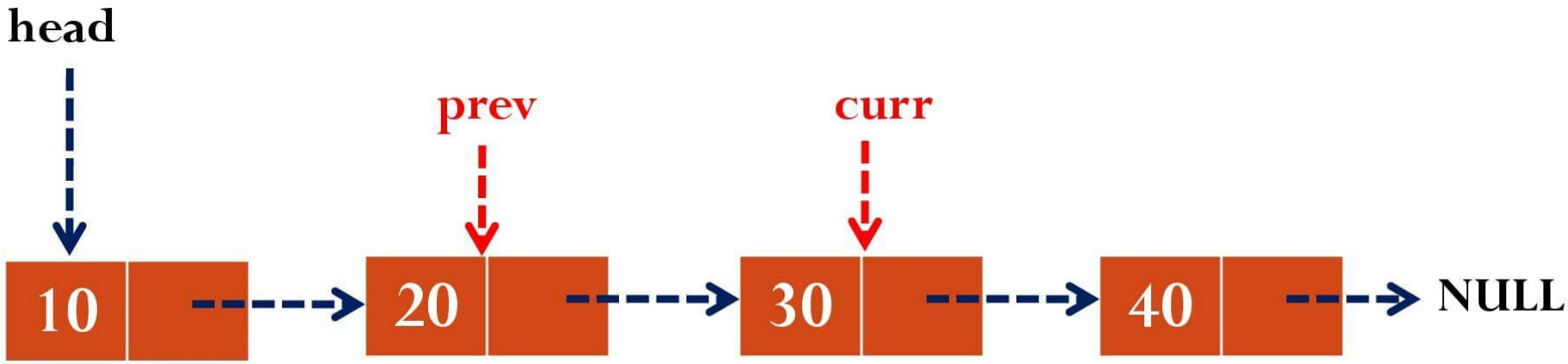
Delete Node 30

Case 3

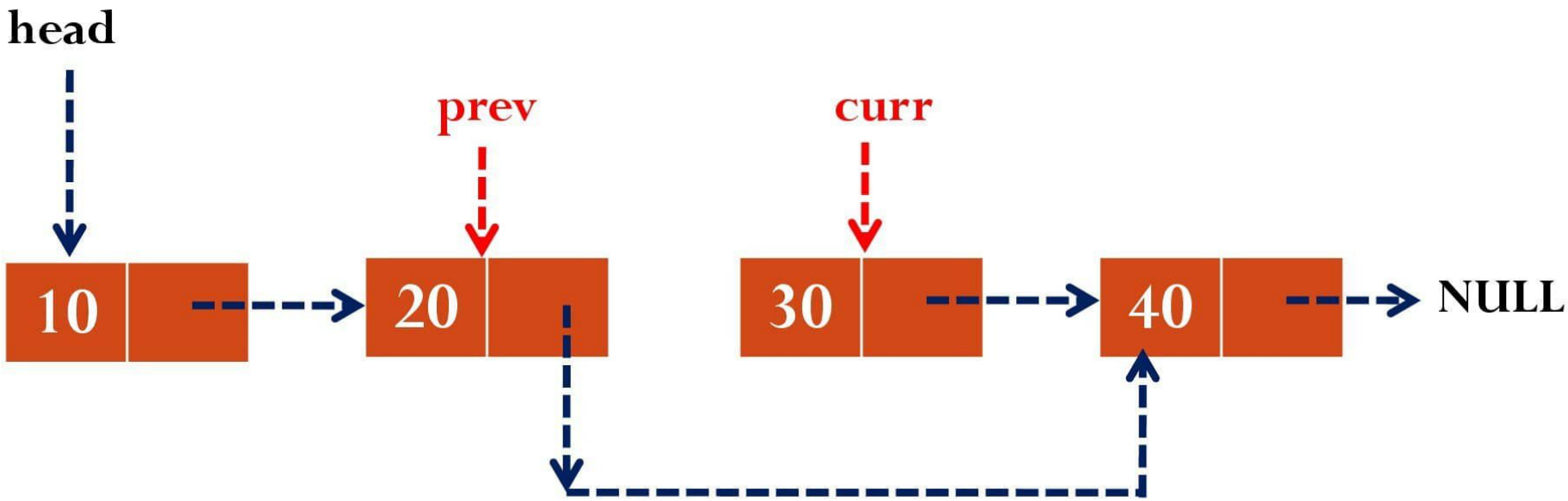


Delete Node 30

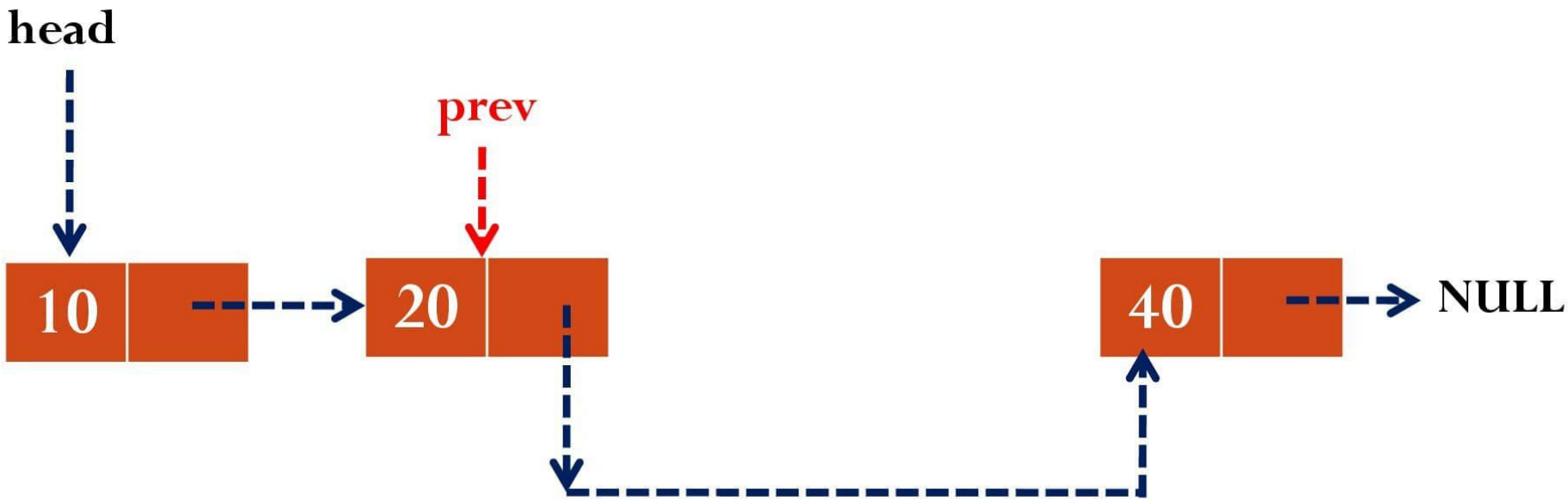
Case 3



Delete Node 30



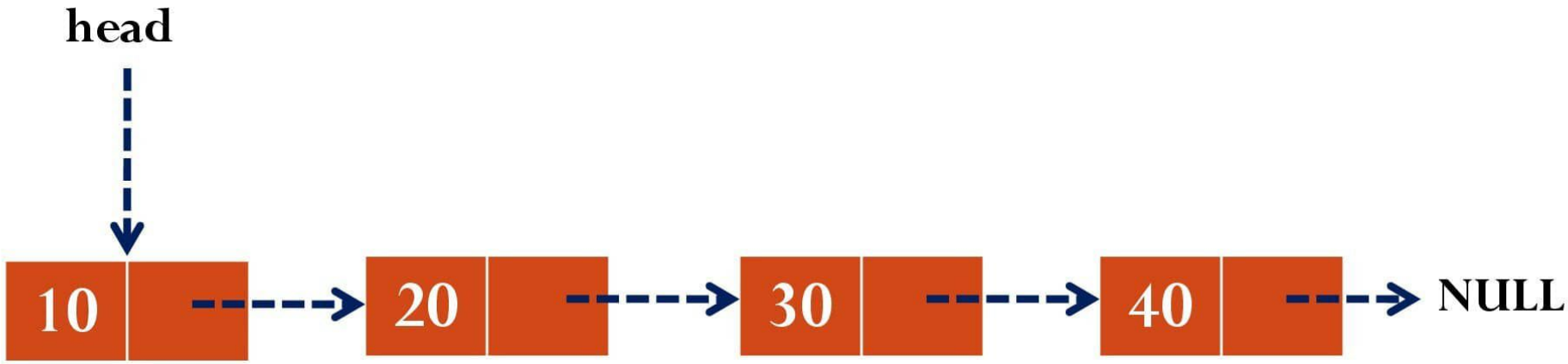
Delete Node 30



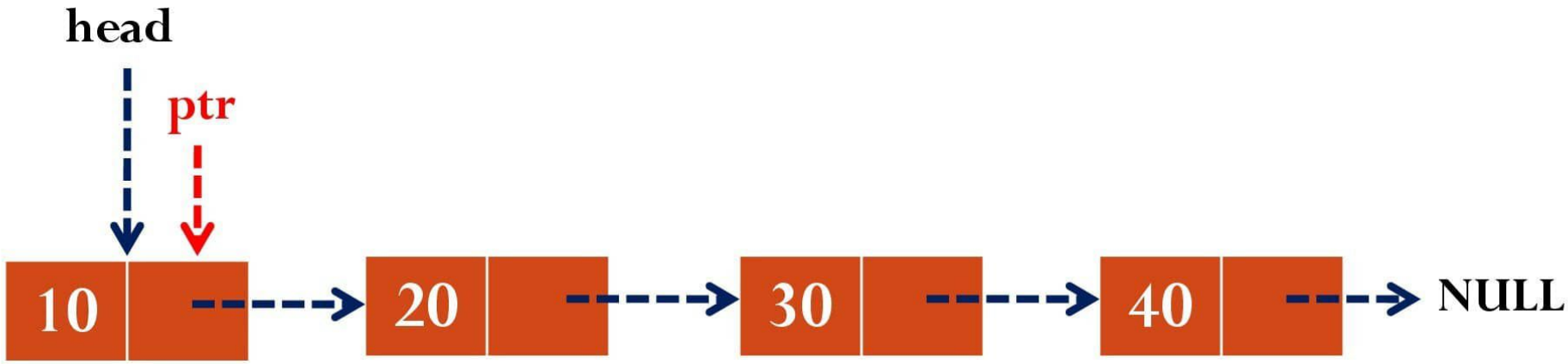
Delete specified node- Algorithm

3. Else
    1. prev=head
    2. curr=head
    3. while curr→data != key and curr→link != NULL do
        1. prev = curr
        2. curr = curr→link
    4. If curr→data != key then
        1. Print “Search key not found”
    5. Else
        1. prev→link = curr→link
        2. dispose(curr)

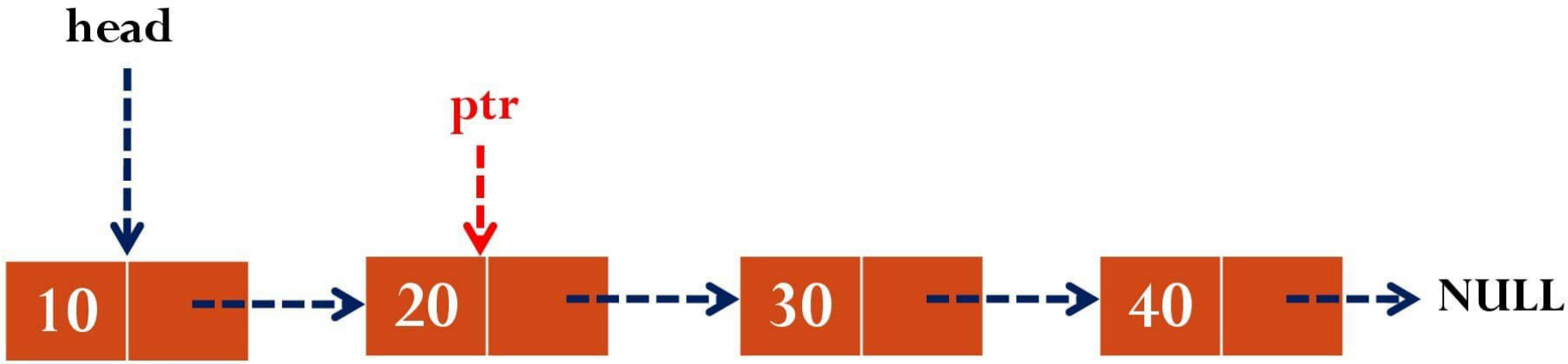
Search 30



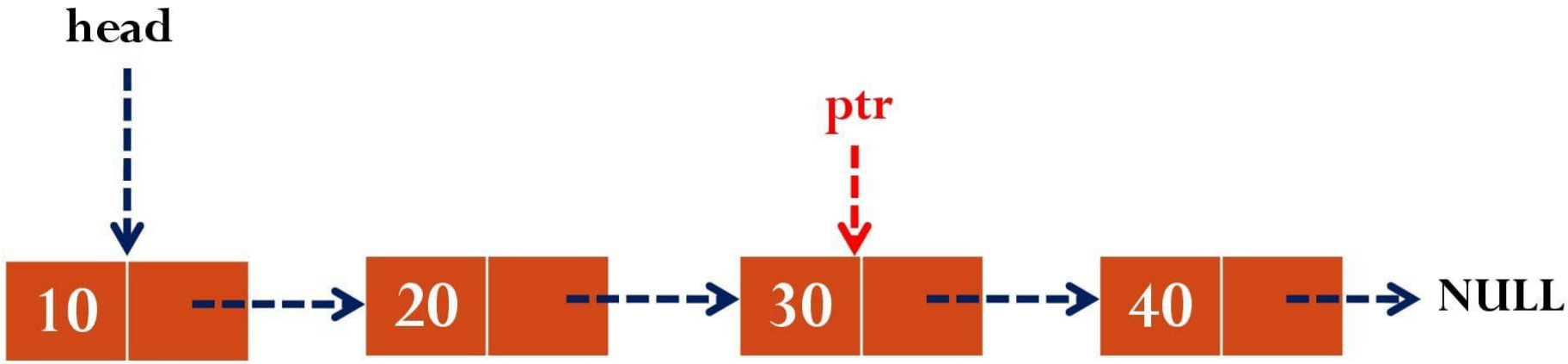
Search 30



Search 30



Search 30

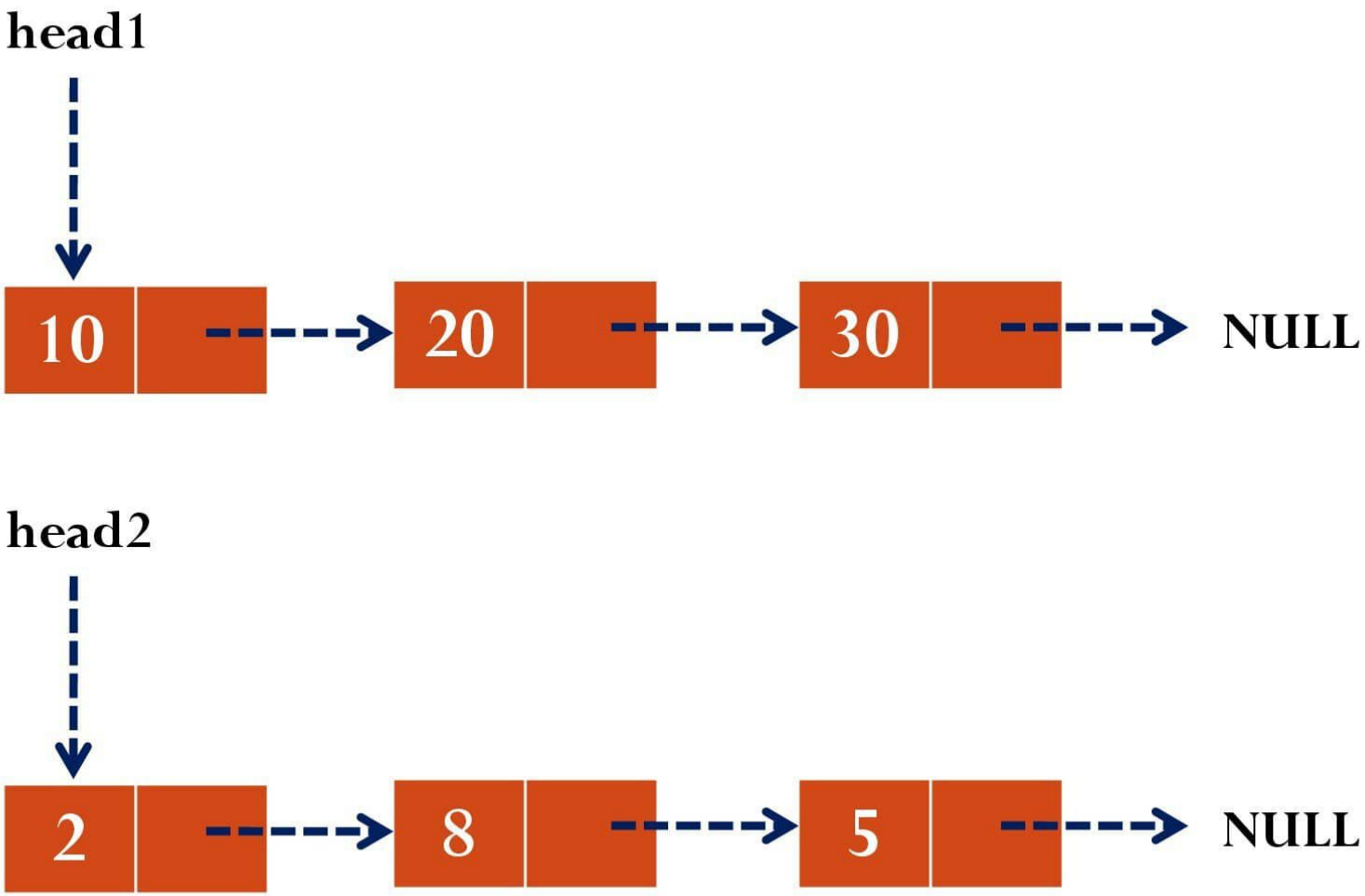


Search - Algorithm

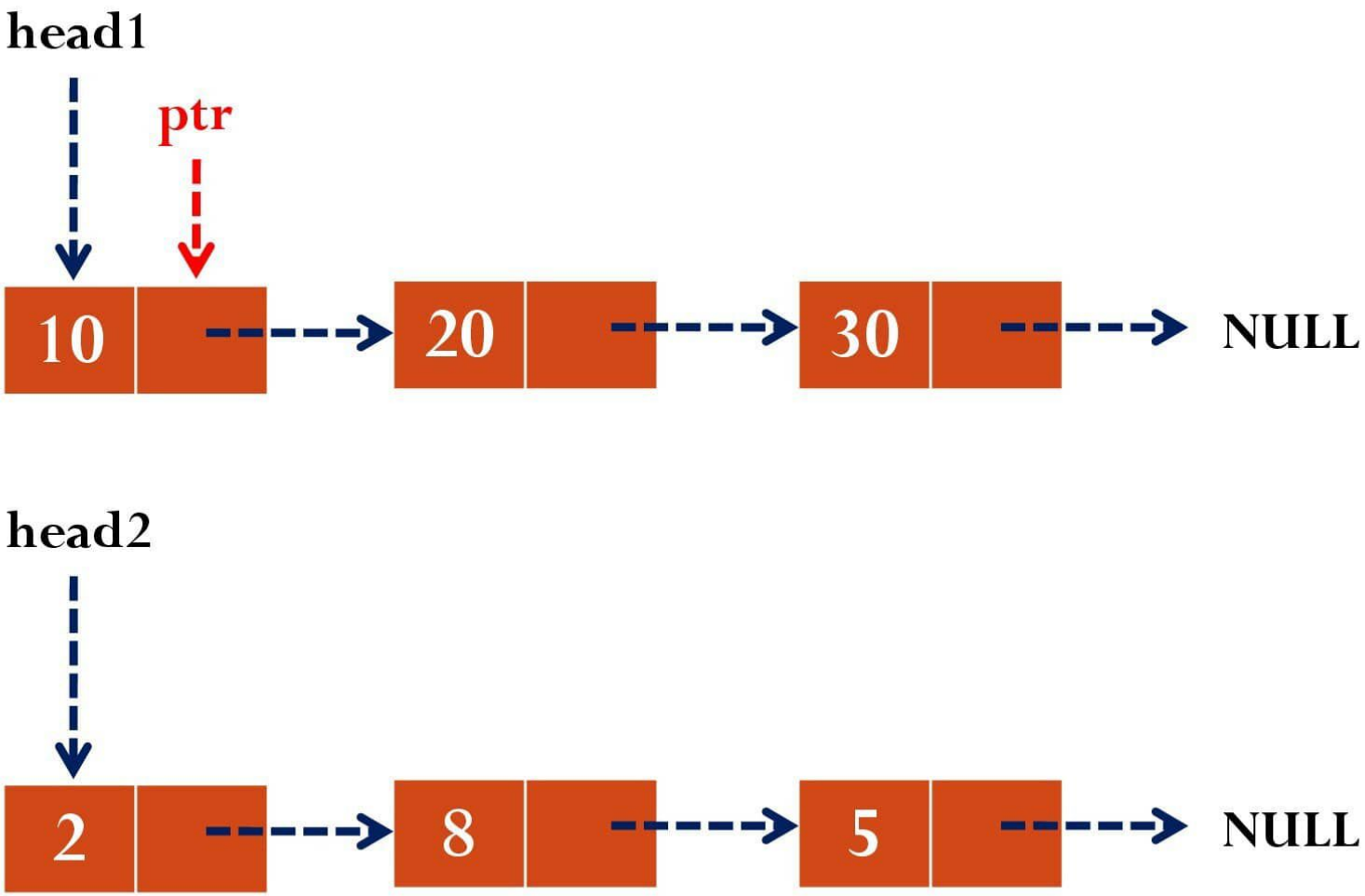
Algorithm Search(head, key)

1. If head=NULL then
    1. Print “List is empty”
2. Else
    1. ptr = head
    2. while ptr→data !=key and ptr→link!=NULL then
        1. ptr = ptr→link
    3. If ptr →data = key then
        1. Print “Search data found”
    4. Else
        1. Print “ Search data not found”

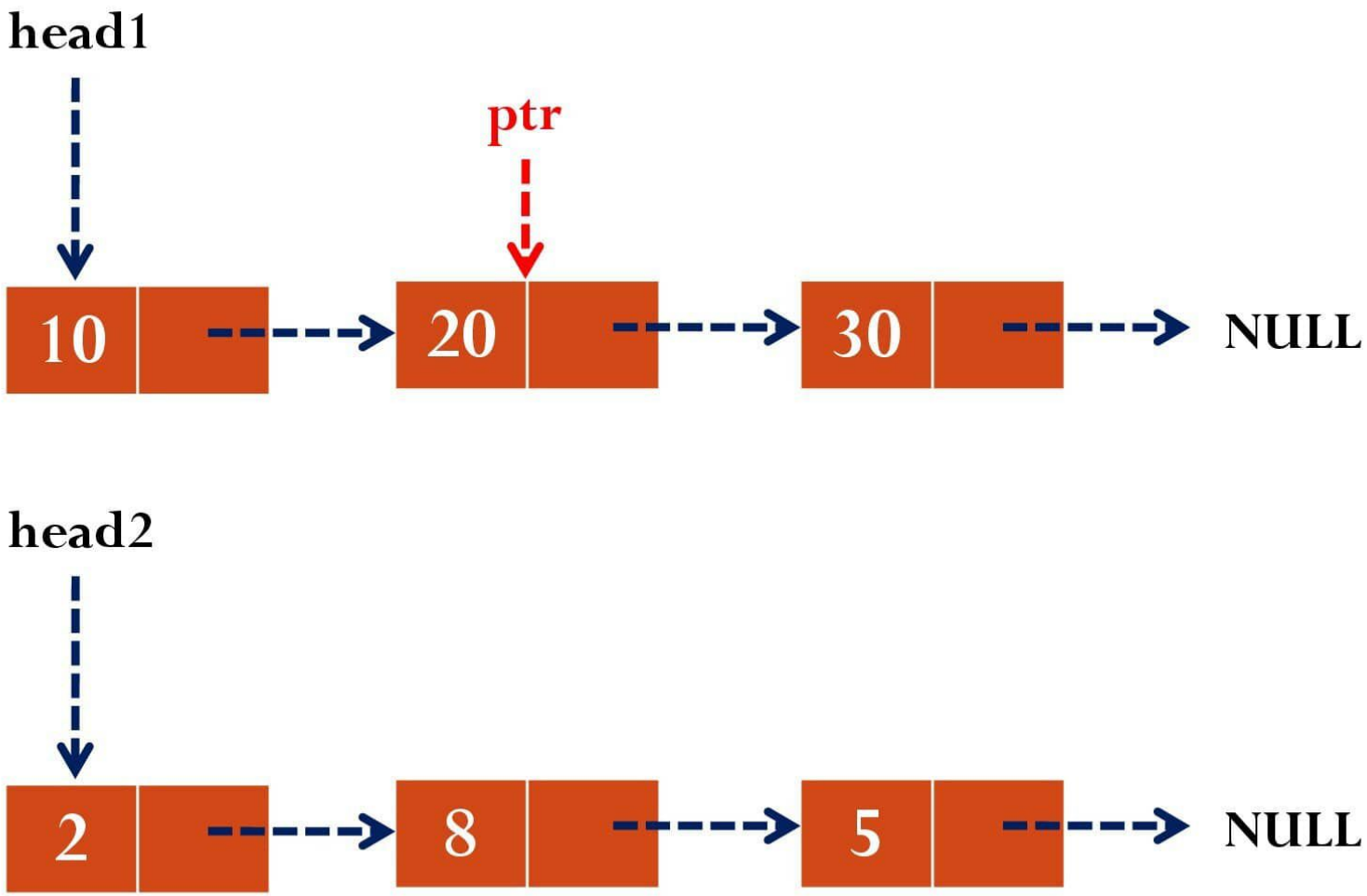
Merge



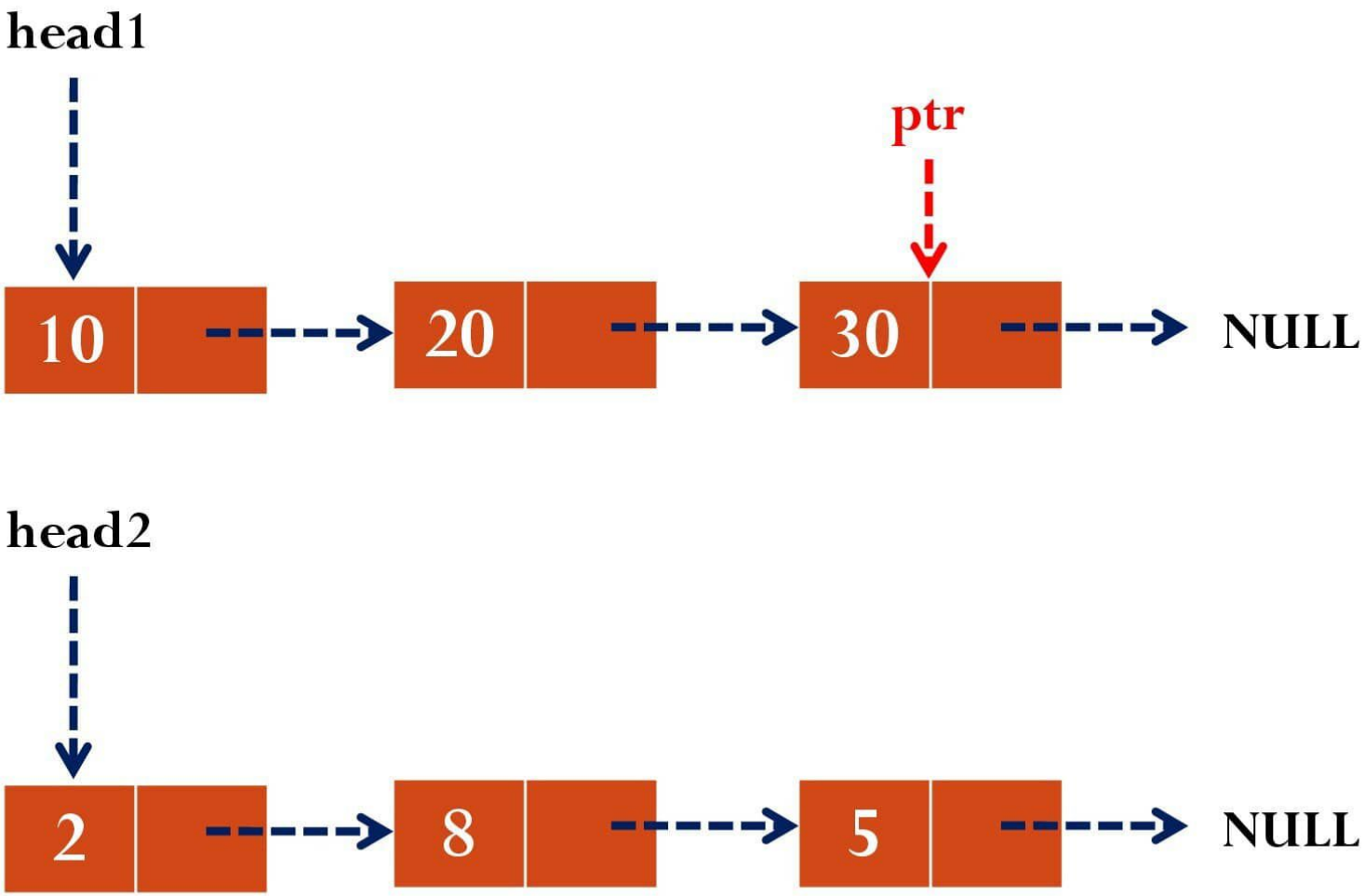
Merge



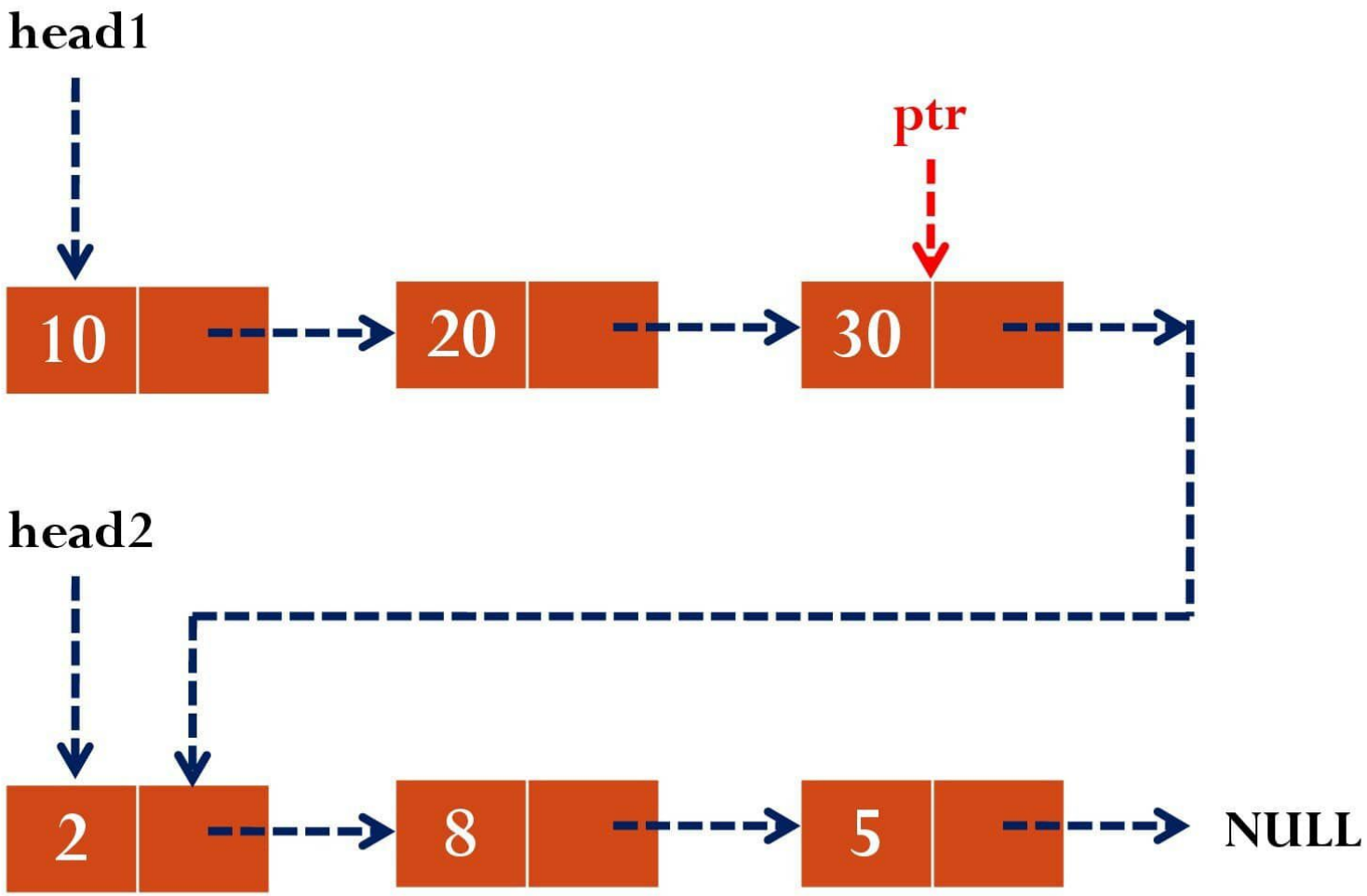
Merge



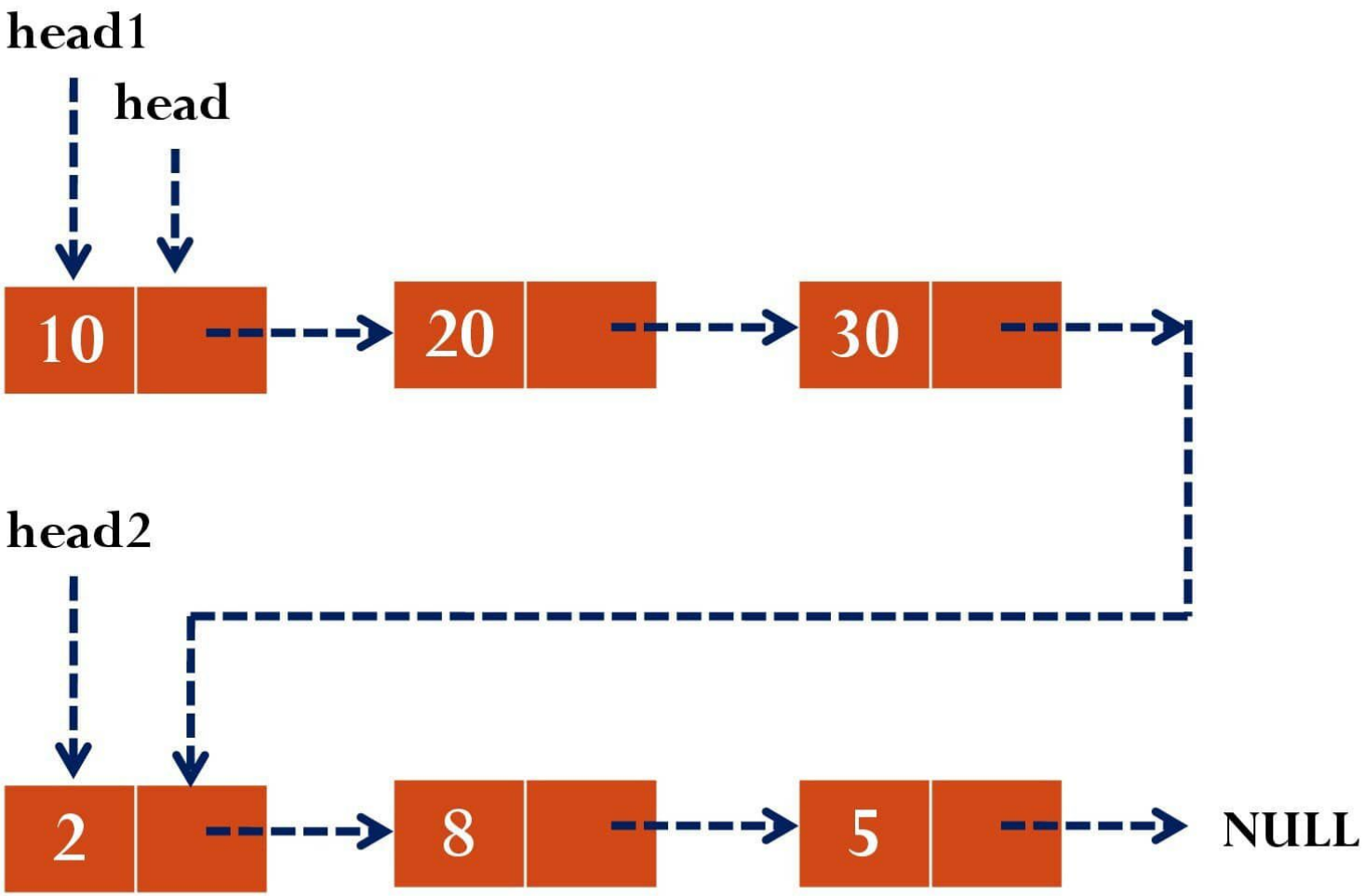
Merge



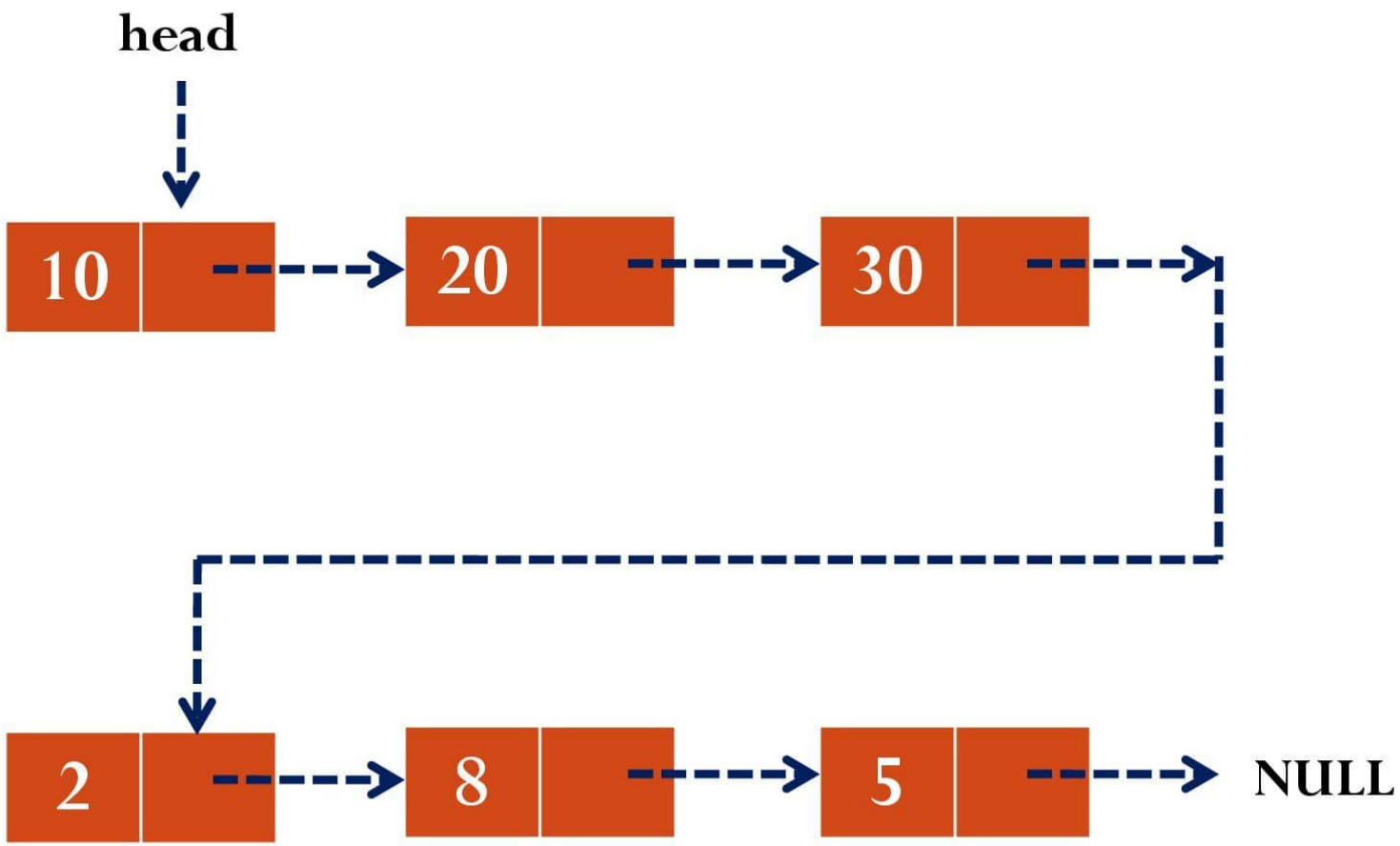
Merge



Merge



Merge



Merge - Algorithm

Algorithm Merge(head1, head2)

1. ptr = head1
2. while ptr→link !=NULL then
    1. ptr = ptr→link
3. ptr →link = head2
4. head=head1