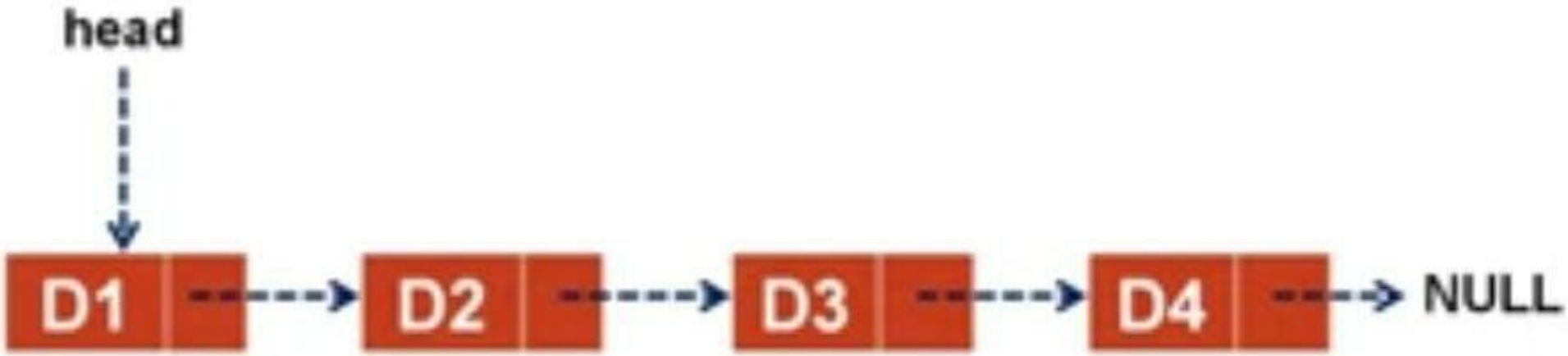
Singly Linked List

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



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

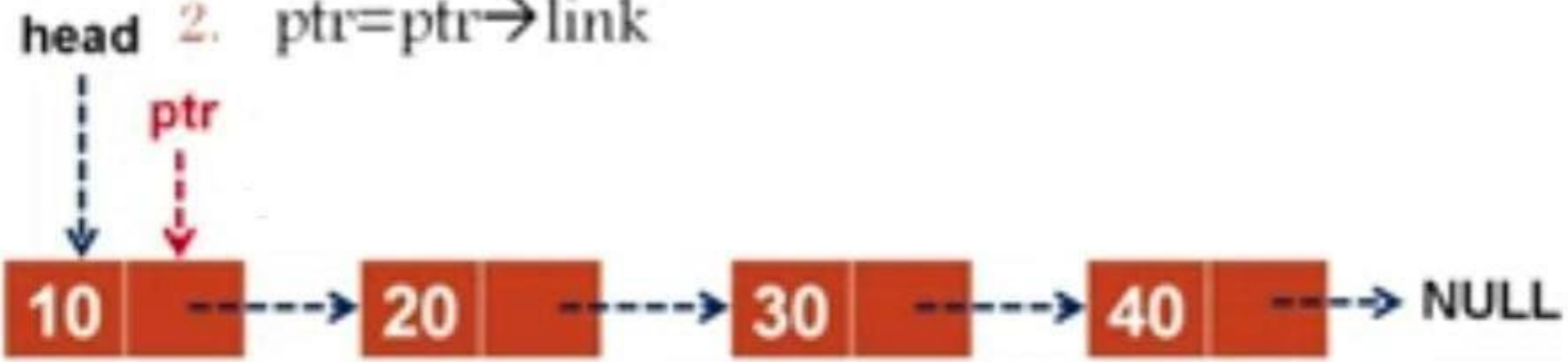
Traversal/Display

- 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

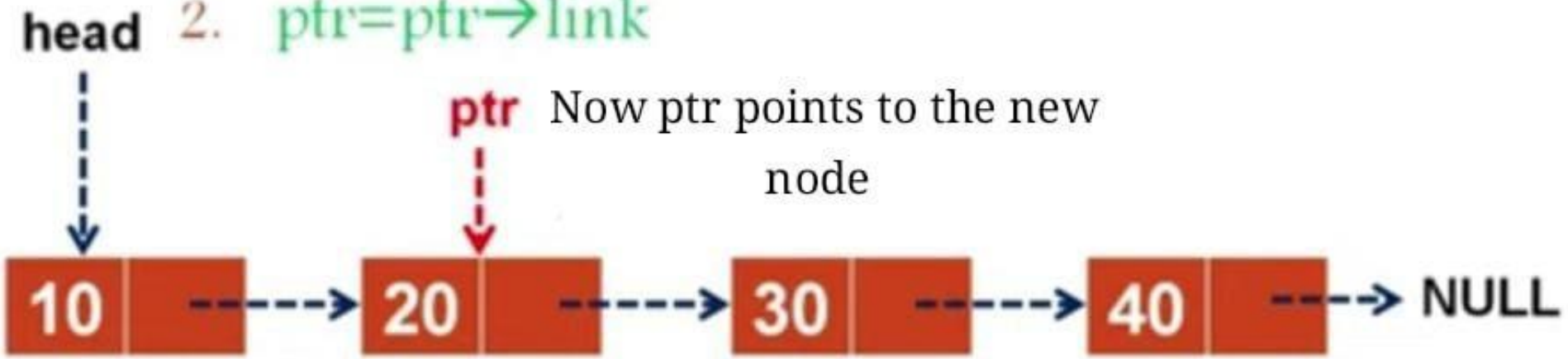


Display/Traversal - Algorithm

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

Previously ptr pointed to head

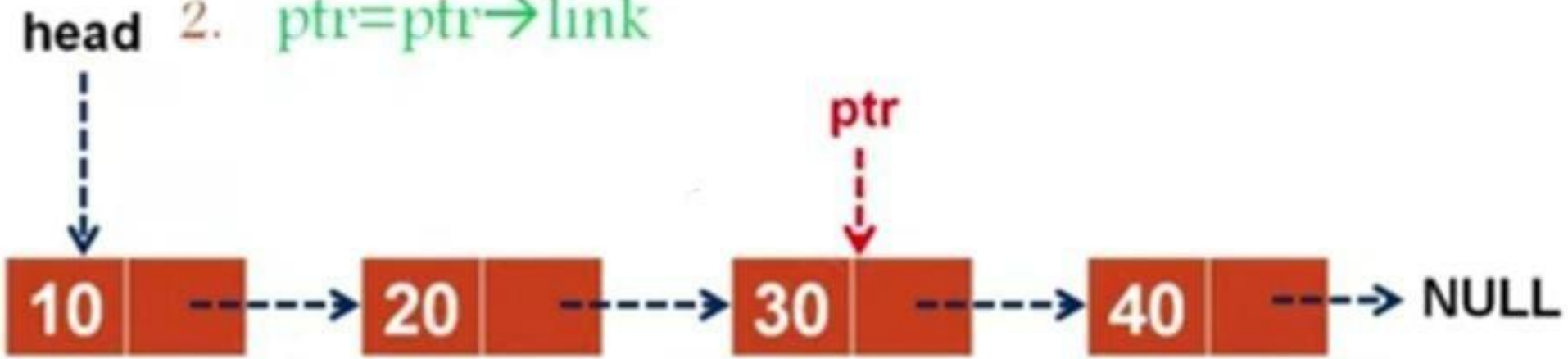


Display/Traversal - Algorithm

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

Again check the condition, now its success so the ptr points to next node



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

Again check the condition and update ptr, now its 40

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

Now the condition is false as ptr points to NULL

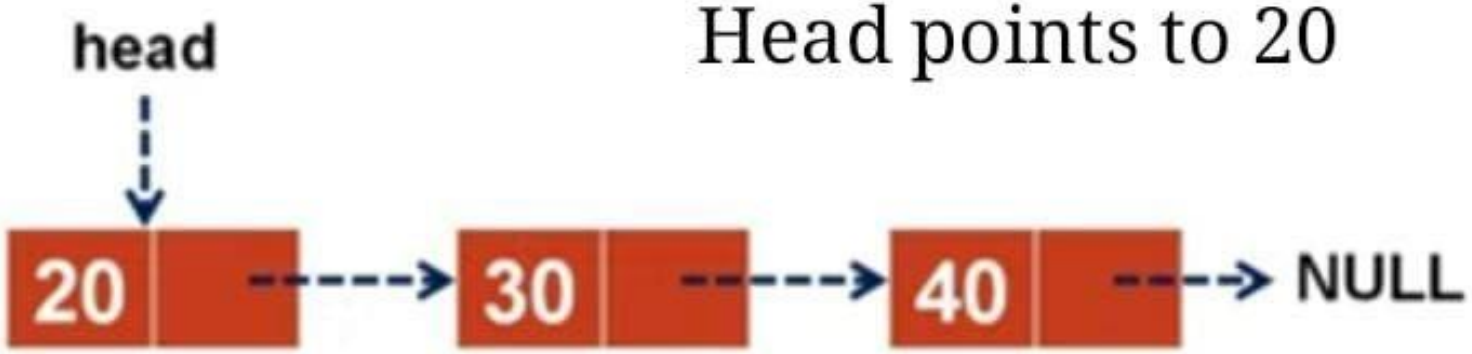
Insertion

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

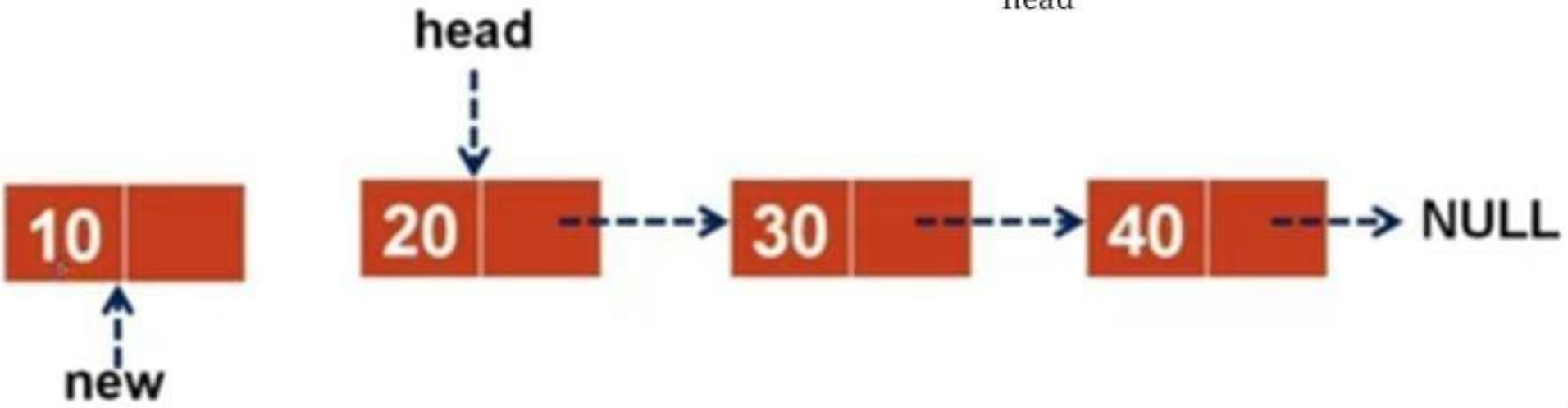
We have to insert a new node at front so create a new node at front

It has two parts, data and link

Data 10 is placed in the new node

Head is placed in the link part of new node

Now,Make the new node as head



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 - Algorithm

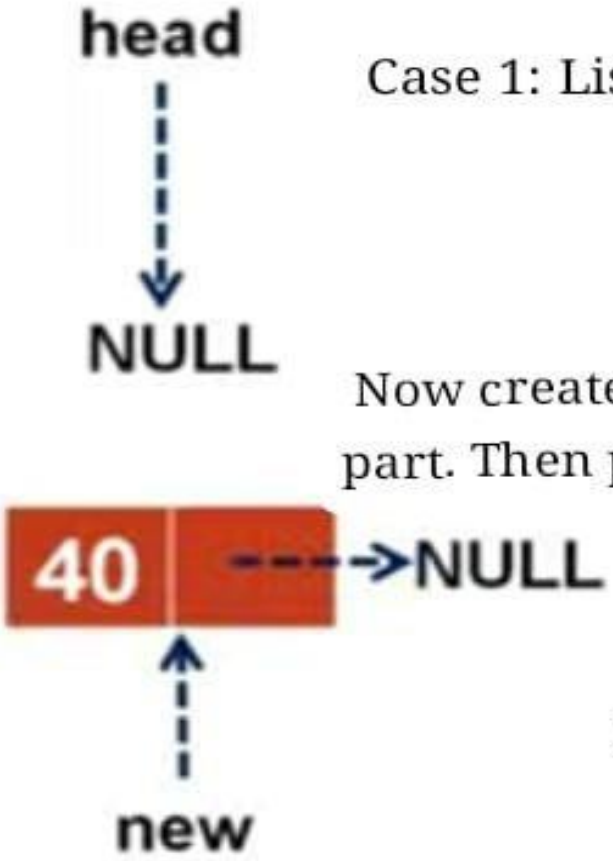
Algorithm Insert_End(head, x)

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

Here we have two cases

Case1: List is empty

Case2:List is not empty



Case 1: List is empty, now the head points to Null

Now create a new node and place the data in data part. Then place Null in its link part (as it is the last node after insertion)

Make the new node as head

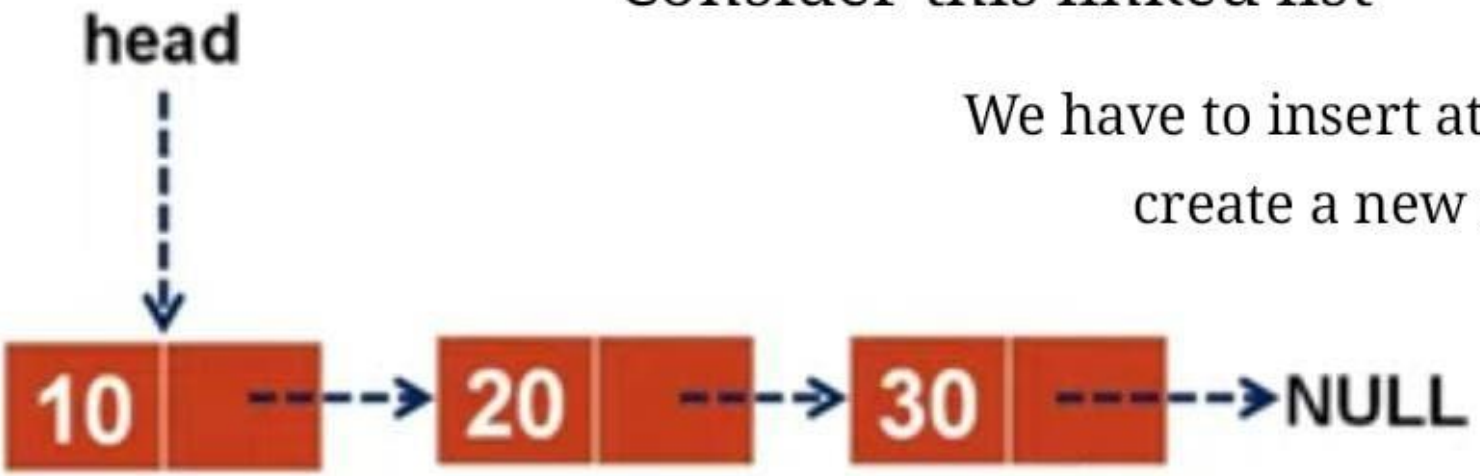
Algorithm Insert_End(head, x)

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

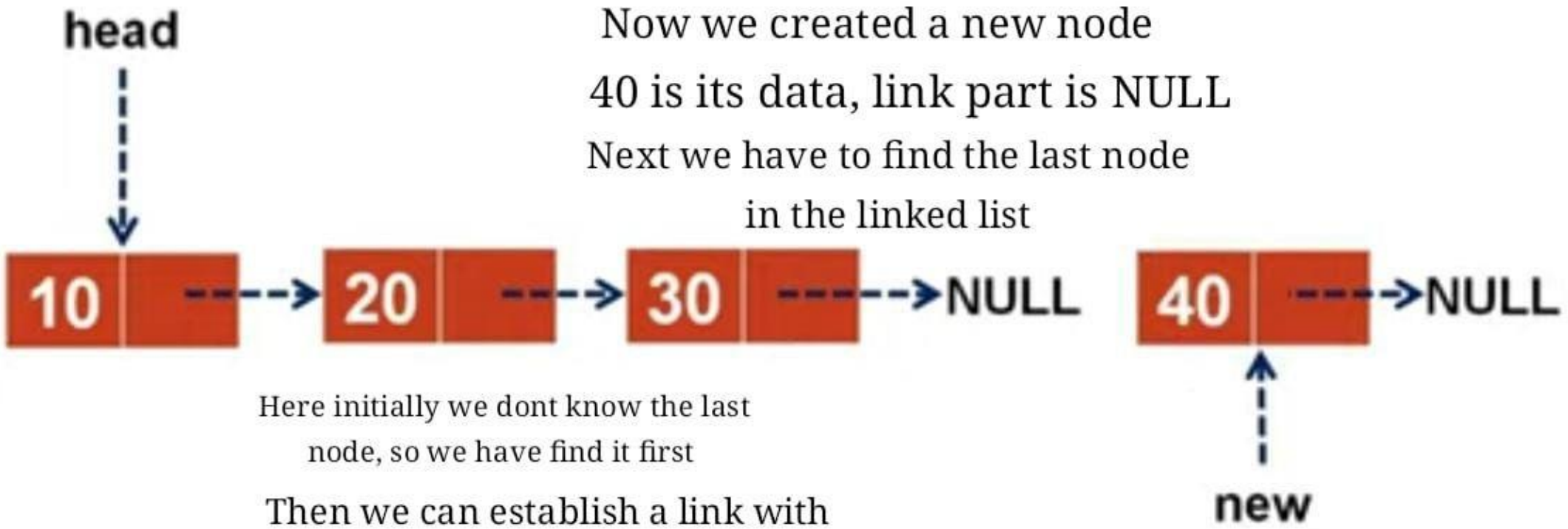
Now new node is inserted at End
(in case of empty list)

Case 2: List is not empty

Consider this linked list

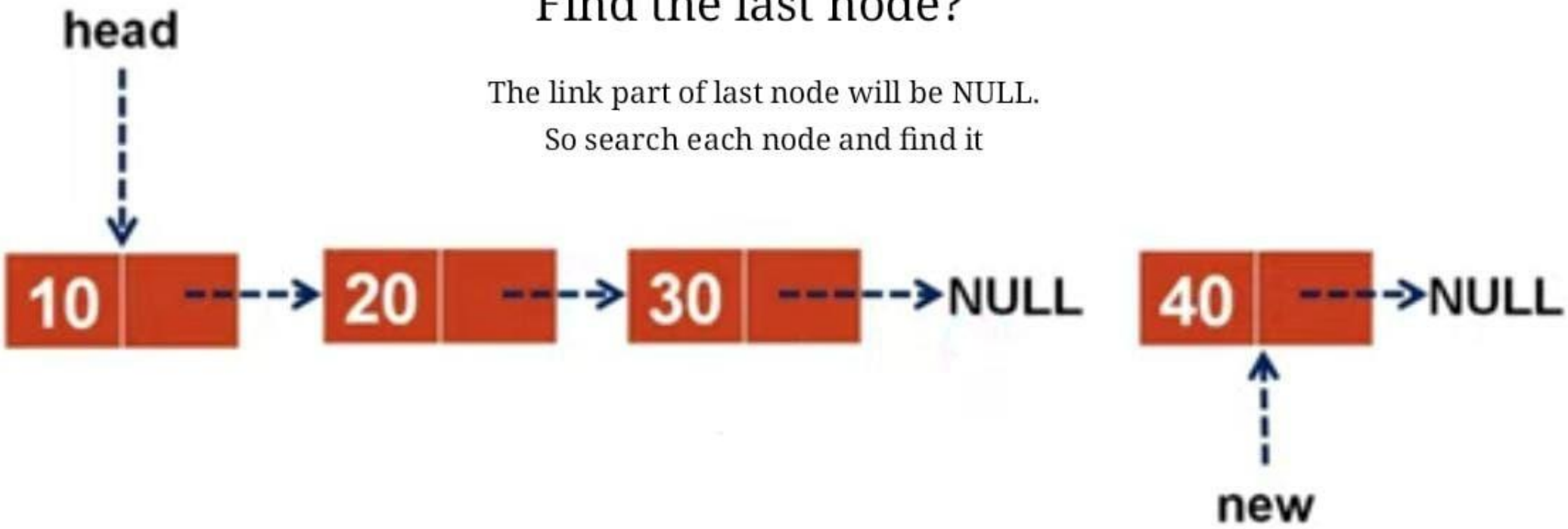


We have to insert at the end, so create a new node

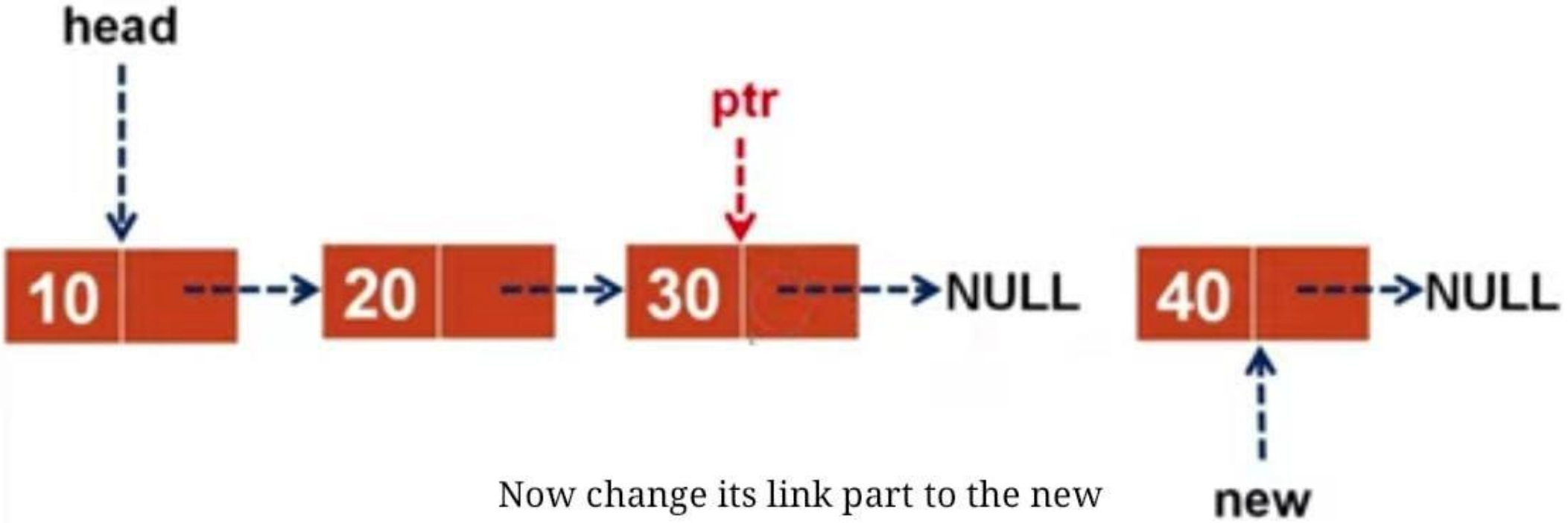

head
Now we created a new node
40 is its data, link part is NULL
Next we have to find the last node
in the linked list
10
20
30
NULL
40
NULL
Here initially we dont know the last
node, so we have find it first
Then we can establish a link with
the new node
new

Find the last node?

The link part of last node will be NULL.
So search each node and find it



We searched the link part of each node
and found that the third node is NULL



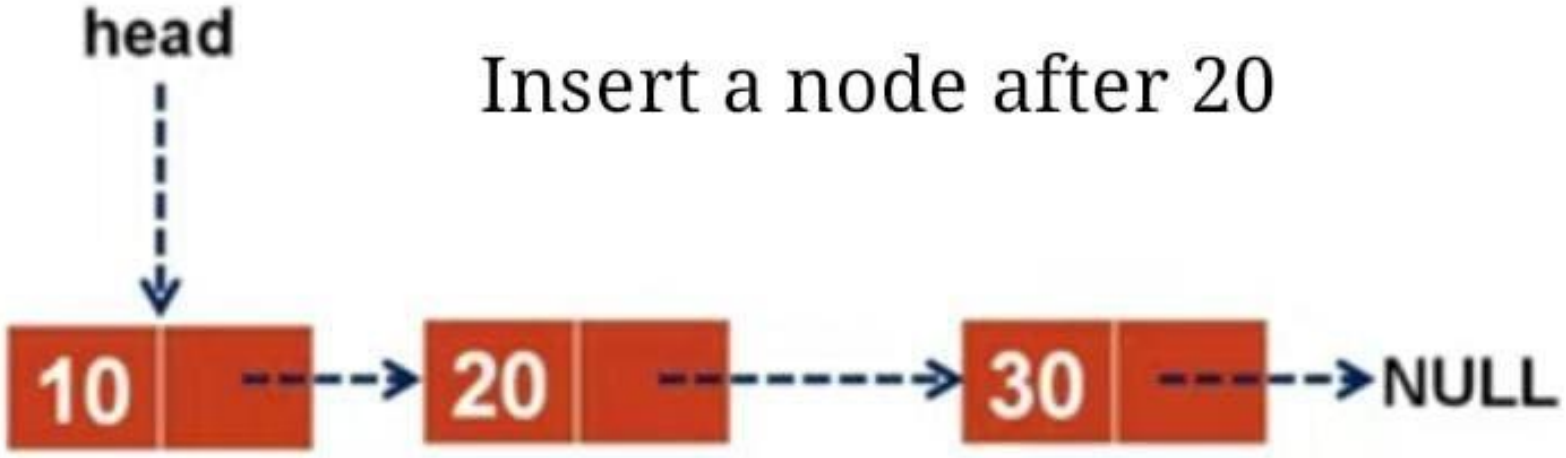
Now change its link part to the new
node

Insert at End - Algorithm

Algorithm Insert_End(head, x)

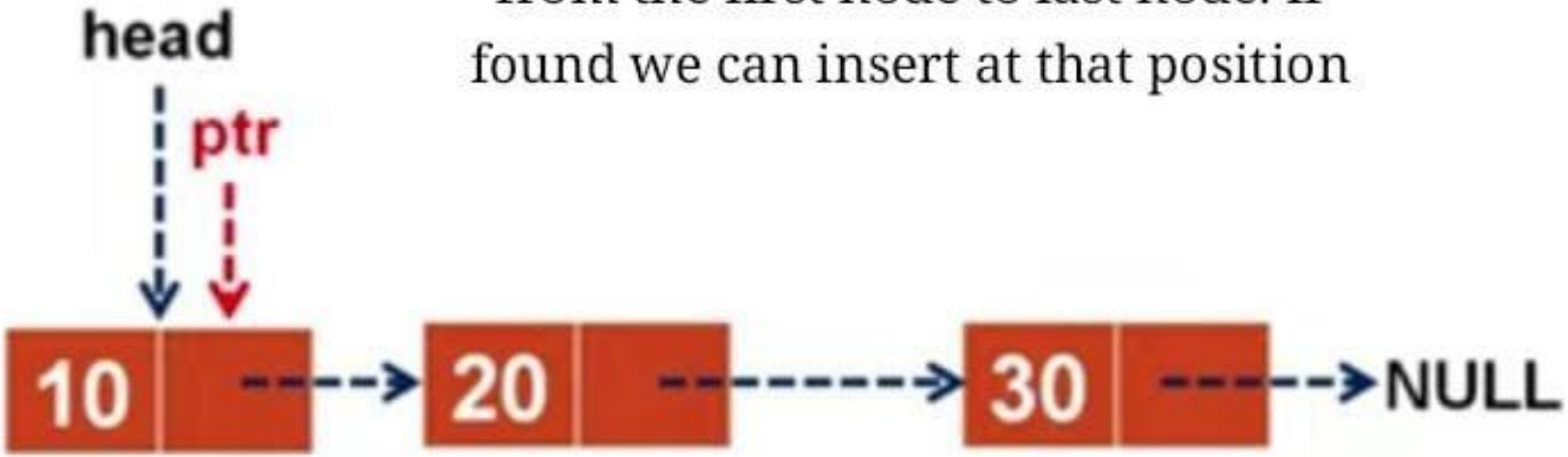
1. Create a node new
2. new→data=x
3. new→link=NULL
4. If head=NULL then
    1. head=new
5. Else
    1. ptr=head
    2. While(ptr→link!=NULL) do
        1. ptr=ptr→link
    3. ptr→link=new

Insert after a specified node 20



Insert after a specified node 20

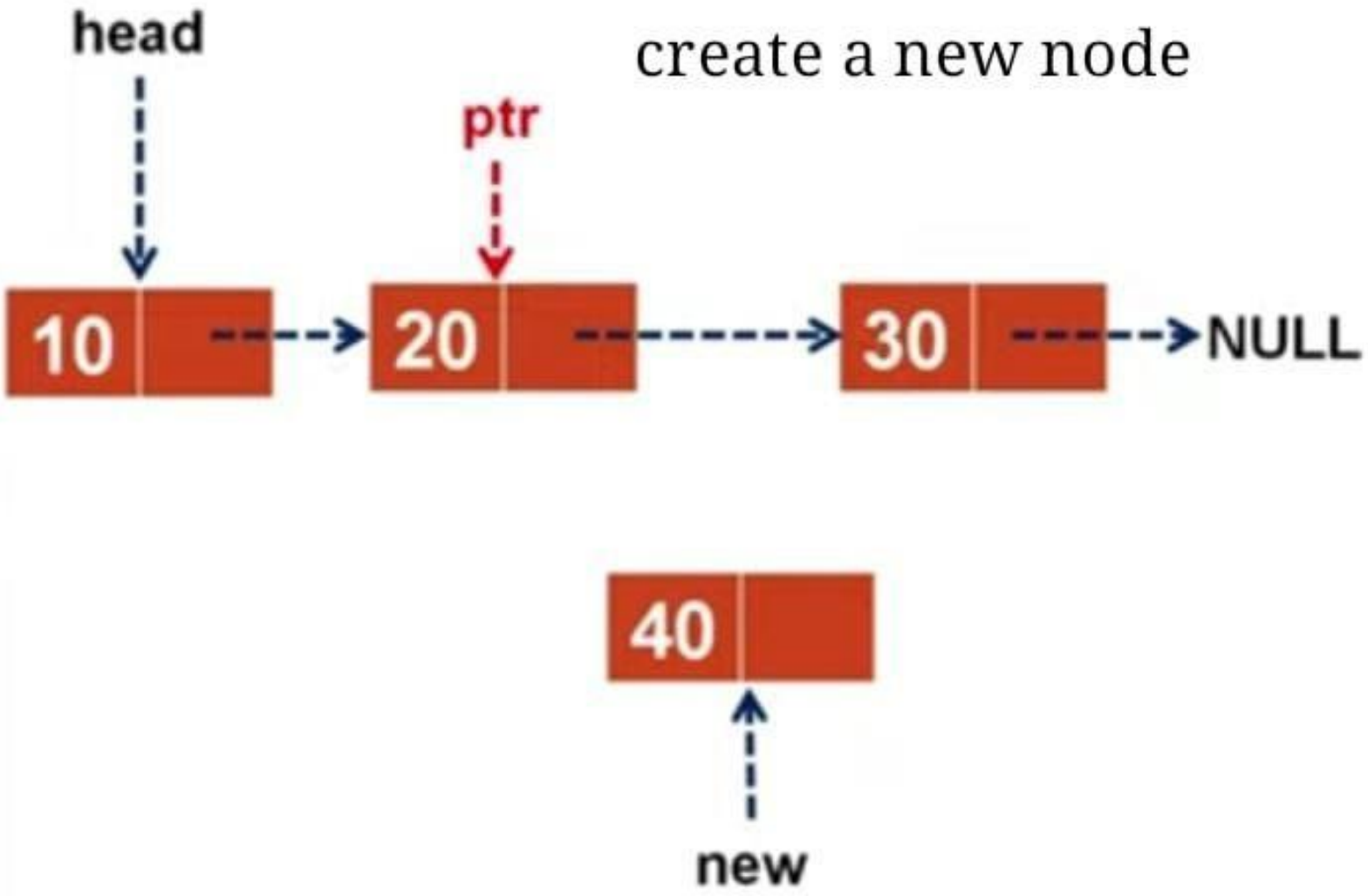
First we have to check whether 20 is present or not. For that we can search from the first node to last node. If found we can insert at that position



Initially ptr points to first node and search for 20.if it fails, continue the searching in the next nodes until 20 is found

Insert after a specified node 20

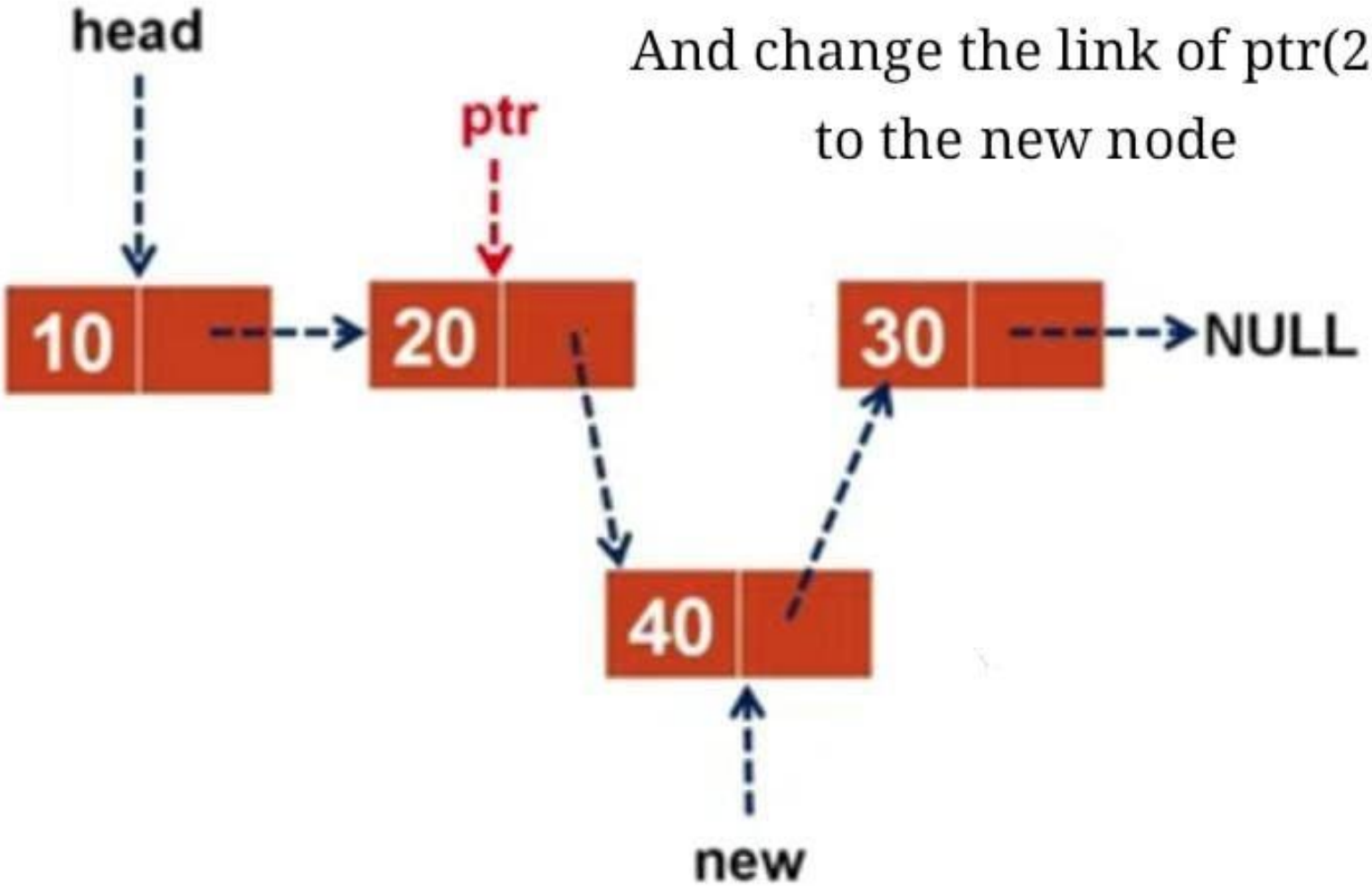
Now 20 is found, so
create a new node



Insert after a specified node 20

Now Place the link part of 20
in the link part of new node

And change the link of ptr(20)
to the new node



Insert after a specified node - Algorithm

Algorithm Insert_After(head, key, x) Key is the element to be searched X is new data

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

Delete from Front- Algorithm

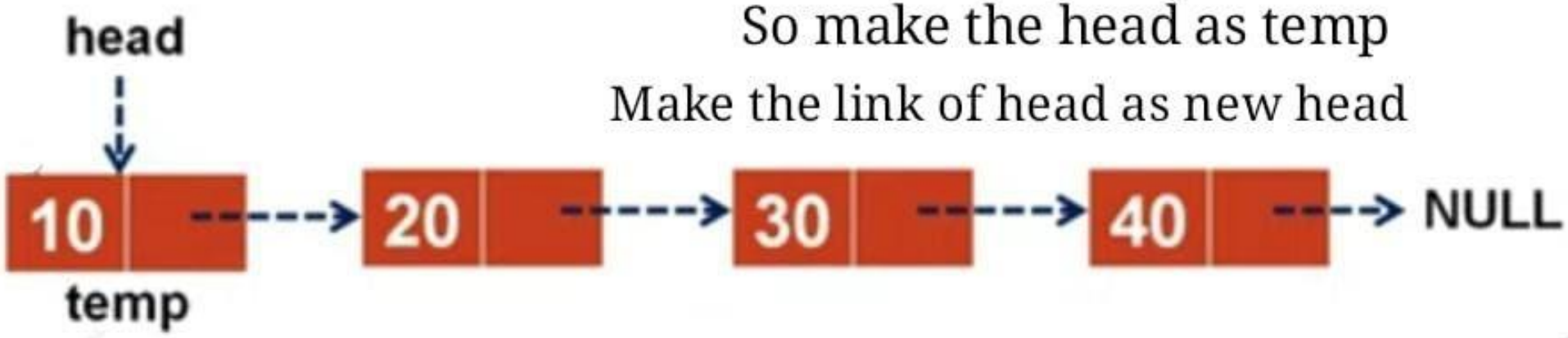
Algorithm Delete_Front(head)

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

10 will be deleted

So make the head as temp

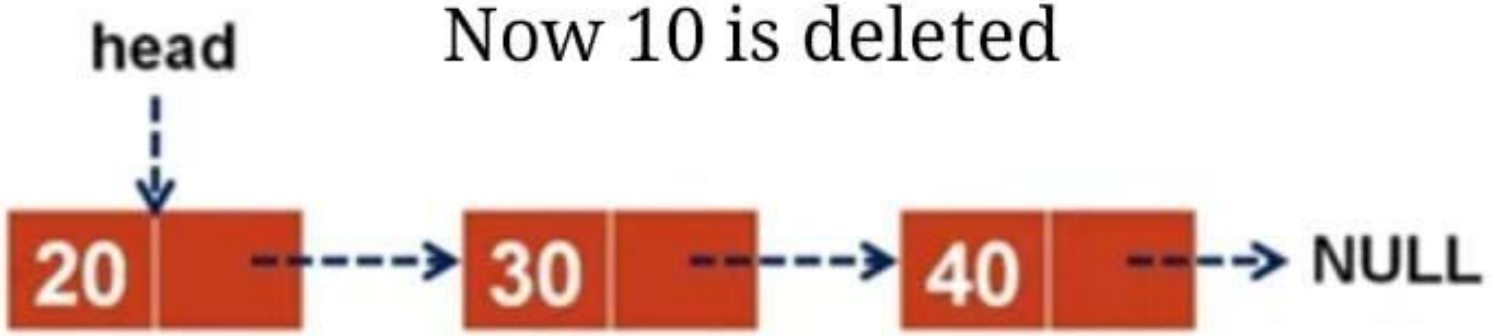
Make the link of head as new head



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 End

3 cases

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

Delete from End- Algorithm

Algorithm Delete_End (head)

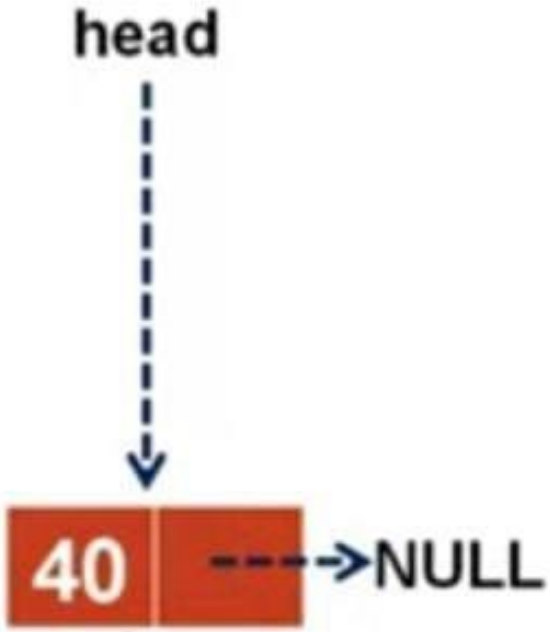
1. If head = NULL then
   1. Print "List is Empty"
2. Else if head→link=NULL then
   1. temp=head
   2. head=NULL
   3. dispose(temp)
3. Else
   1. prev = head
   2. curr = head→link
   3. while curr→link !=NULL do
      1. prev = curr
      2. curr = curr→link
   4. prev→link=NULL
   5. dispose(curr)

Case 1: Cannot perform deletion if list is empty

Delete from End

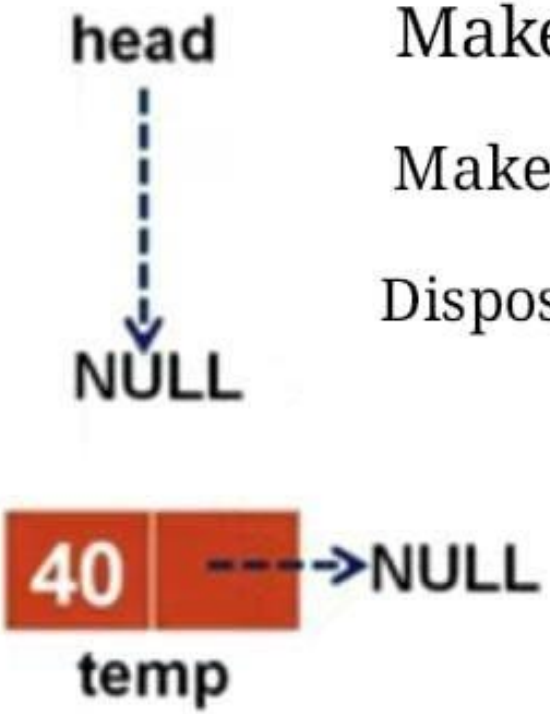
Case 2

List contains only one node



Delete from End

Case 2



Make this node as temp

Make head points to NULL

Dispose the memory occupied by temp

Delete from End

Case 3

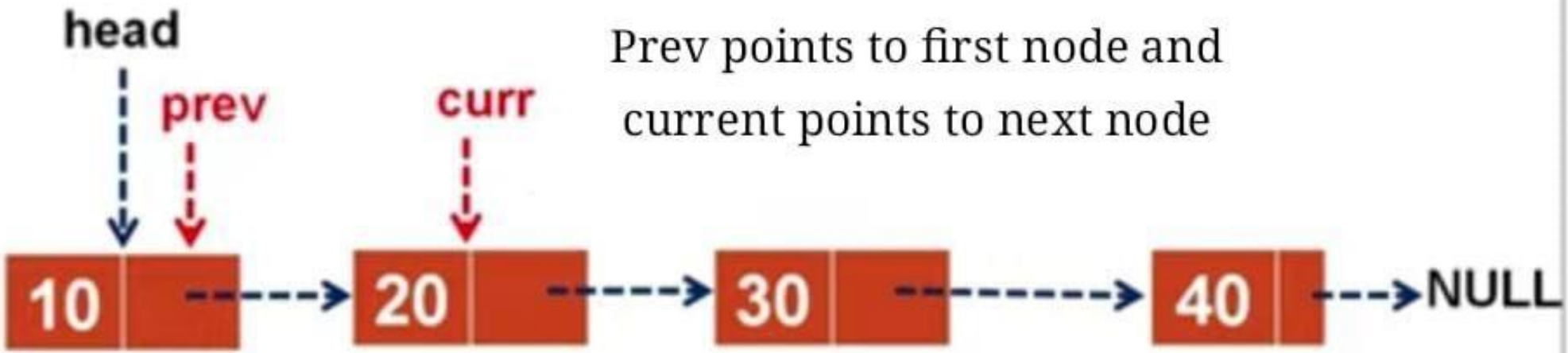
List contains more than one element

Delete from End

Case 3

Set two pointers prev and current

Prev points to first node and current points to next node

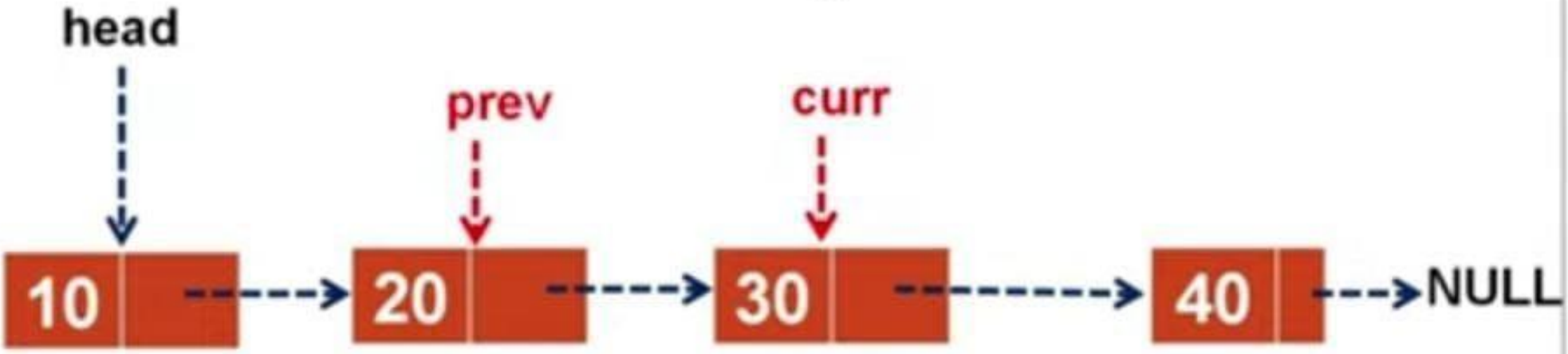


Check link field of current points to NULL, if so it is the end node. Otherwise change the prev and current to next nodes and continue this process to find NULL. i. e last node

Delete from End

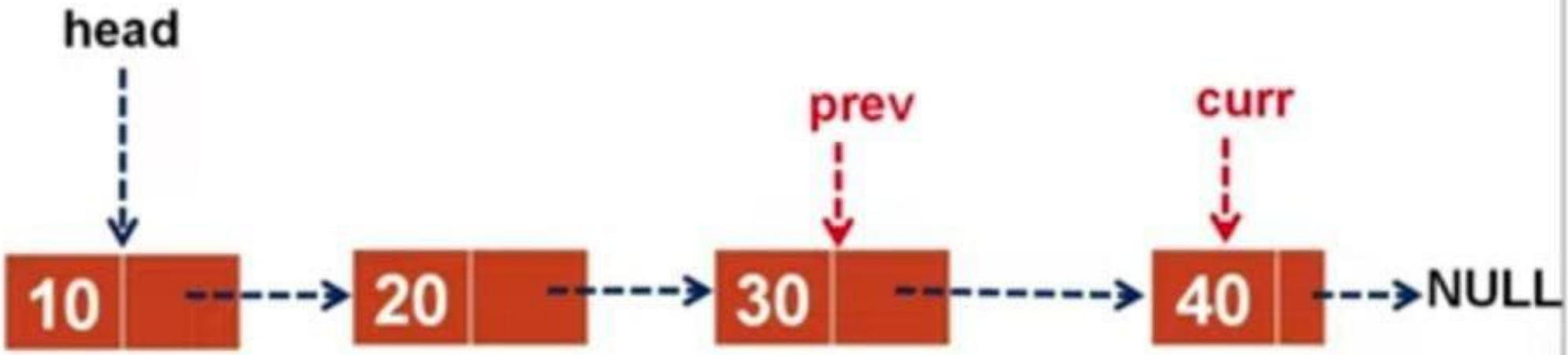
Case 3

Checking for NULL



Delete from End

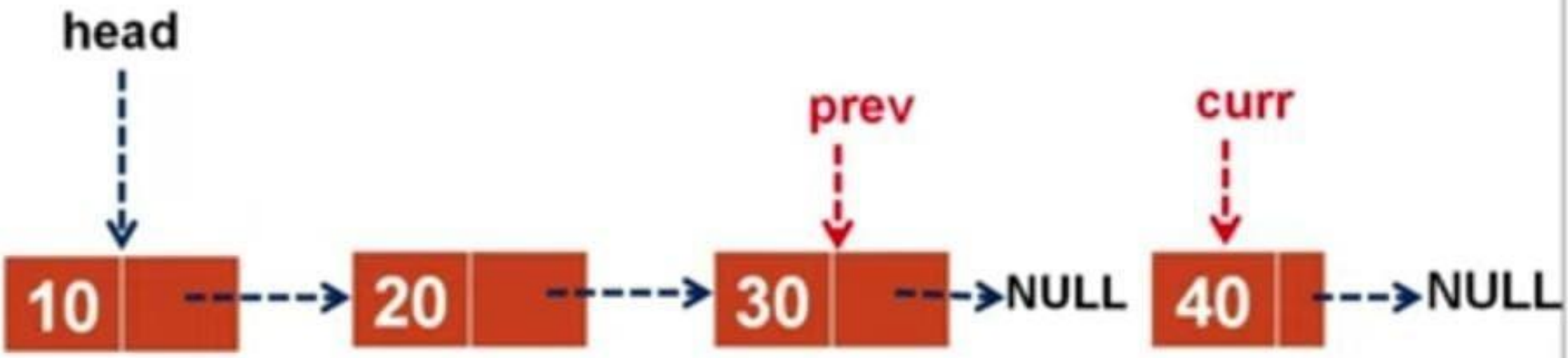
Case 3



Now we reached the NULL, last node. So to delete this node make the link field of prev to NULL

Delete from End

Case 3



Now its deleted

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)

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)

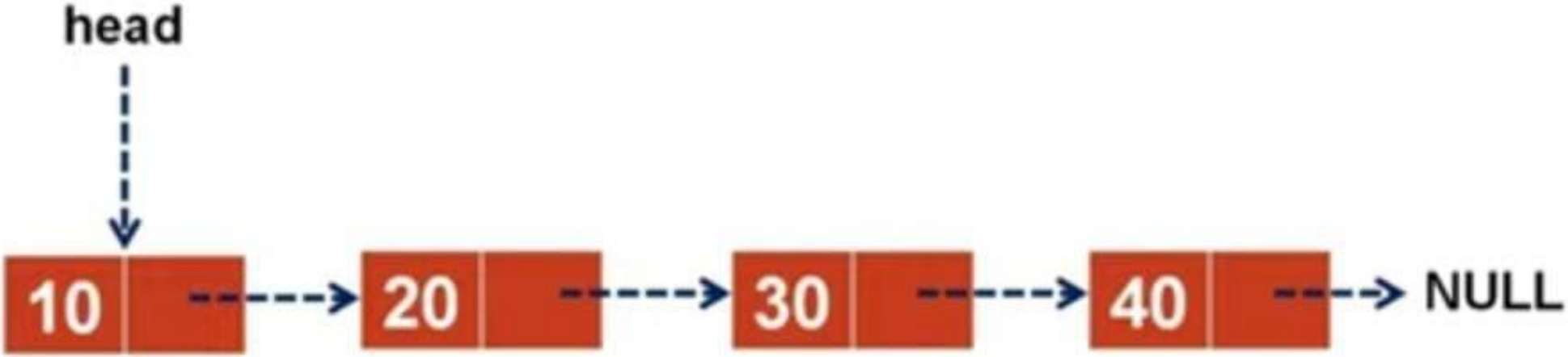
Delete specified node- Algorithm Case 3

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

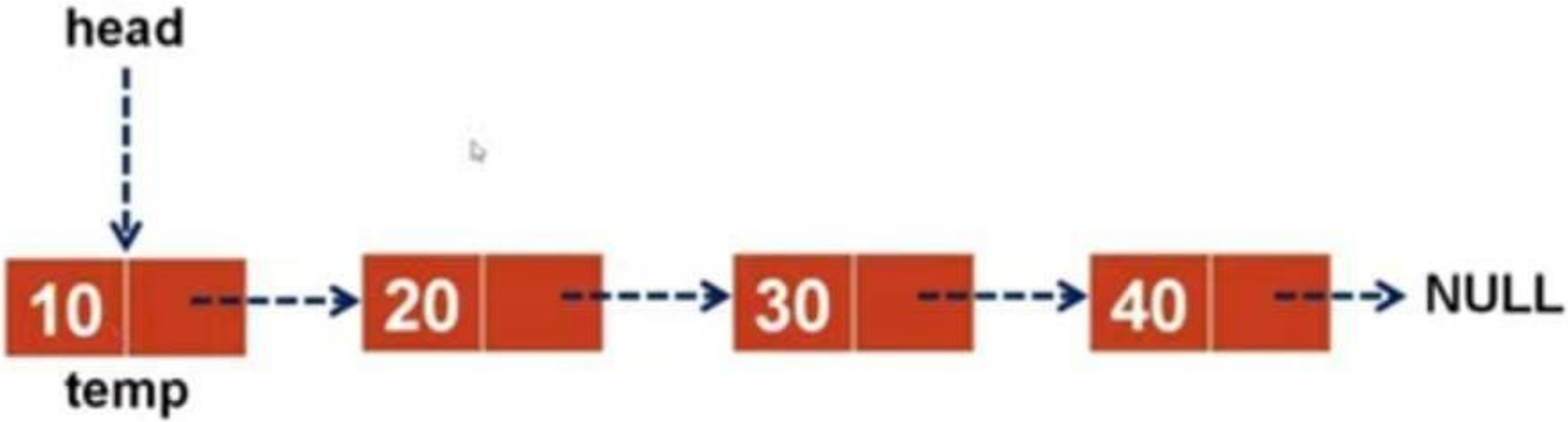
Search data present in first node



Delete first node

Delete Node 10

Case 2



Now make head as temp

Then delete it

Now Point head to next node

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

Other cases

List contains more than one element

Now set two pointers prev and current

Delete Node 30

Case 3

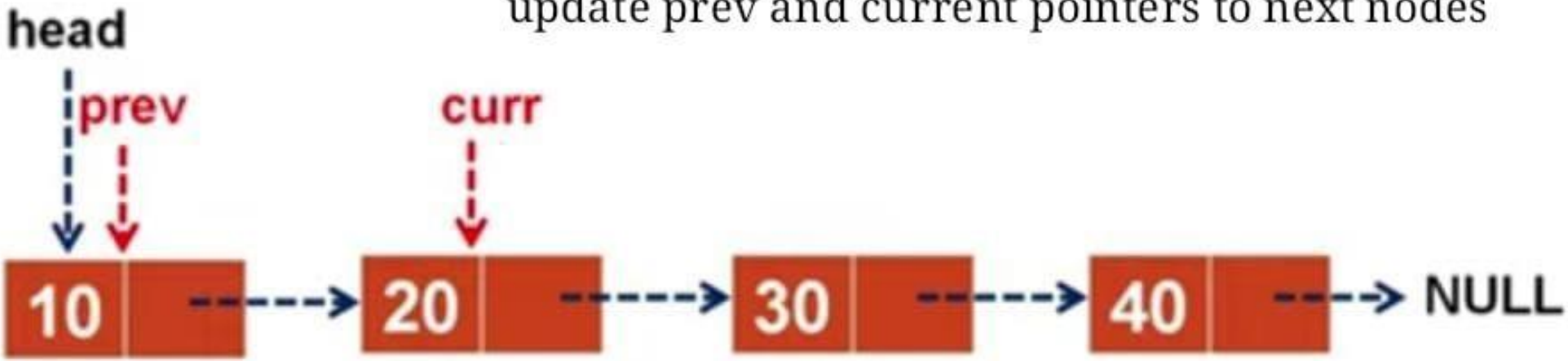
Search data is 30

Check current is 30?If so delete it,otherwise update current to next node to search for 30

Delete Node 30

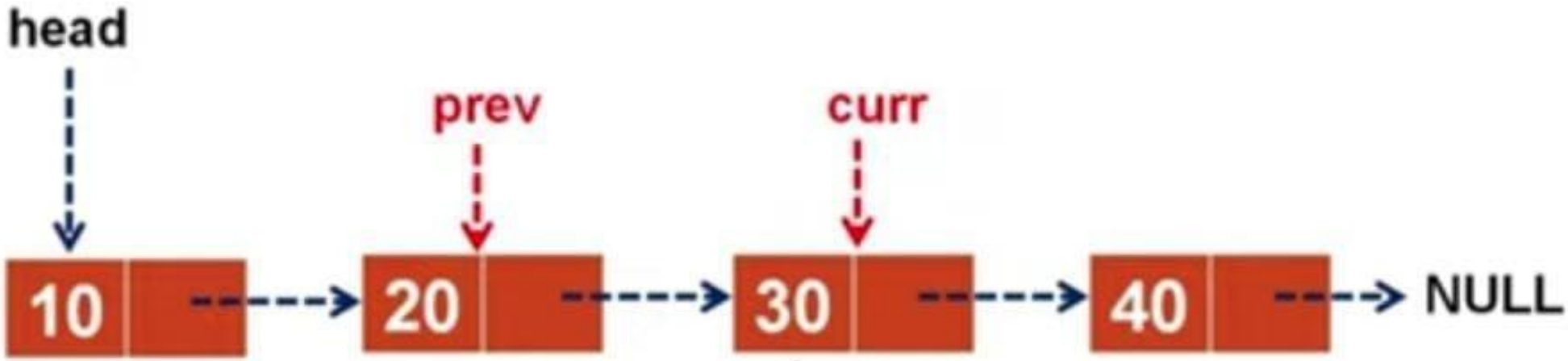
Case 3

Again check current data is search data 30, If not update prev and current pointers to next nodes



Delete Node 30

Case 3



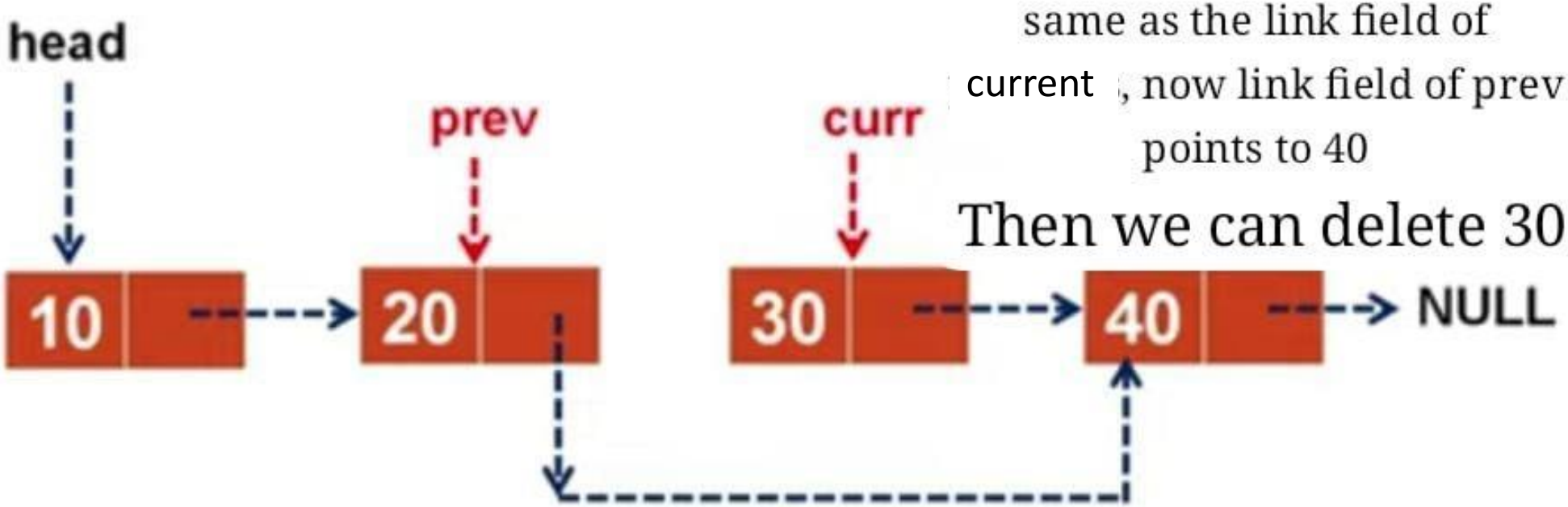
Now the current data is search data

Delete Node 30

We have to delete 30

So make the link field of prev same as the link field of current , now link field of prev points to 40

Then we can delete 30

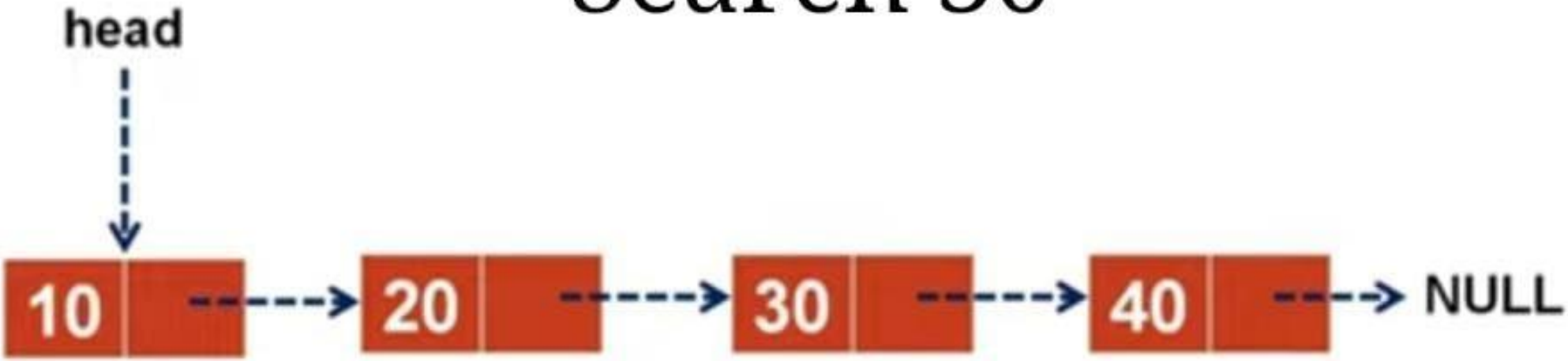


Delete specified node- Algorithm

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 Algorithm

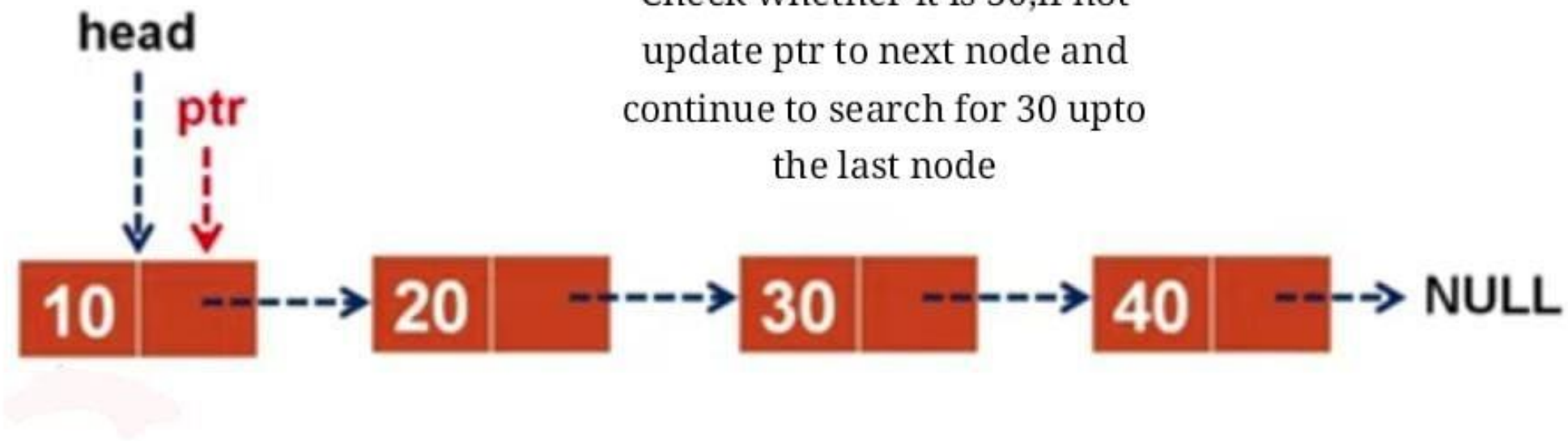
Search 30



Search 30

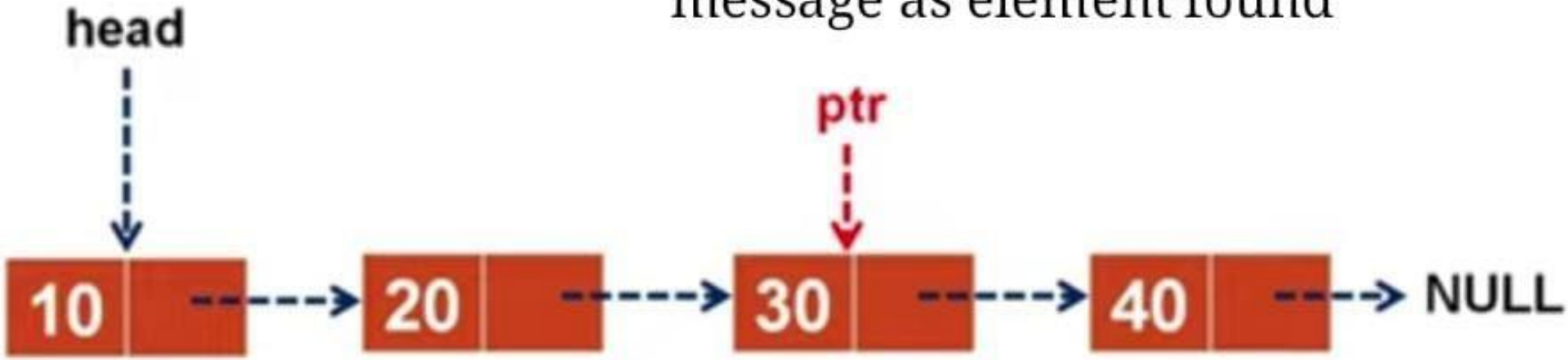
ptr points to first node

Check whether it is 30,if not update ptr to next node and continue to search for 30 upto the last node



Search 30

Now ptr points to 30,so display message as element found



If the search data was 37 we can display the message as Not found

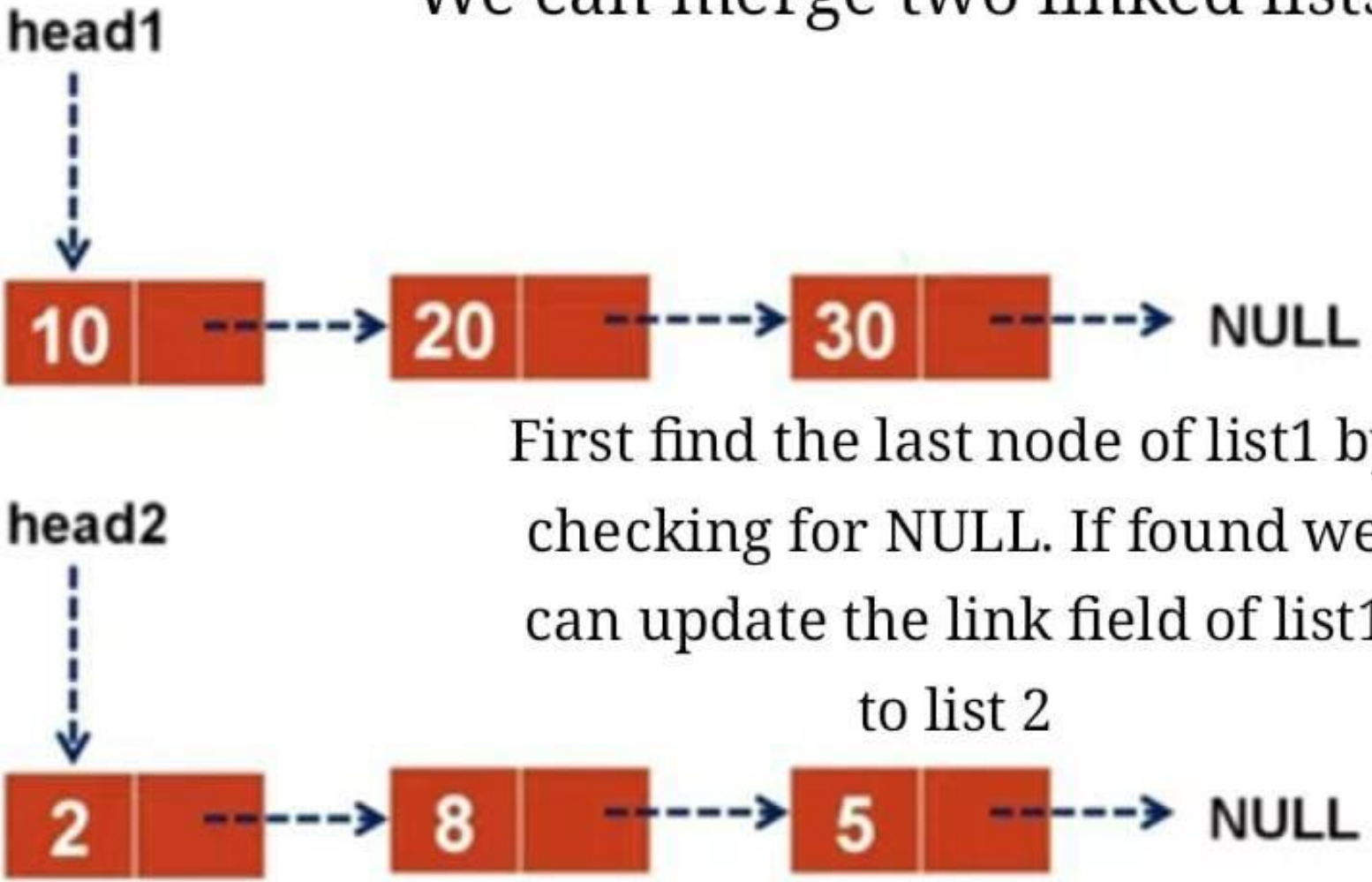
Search - Algorithm

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

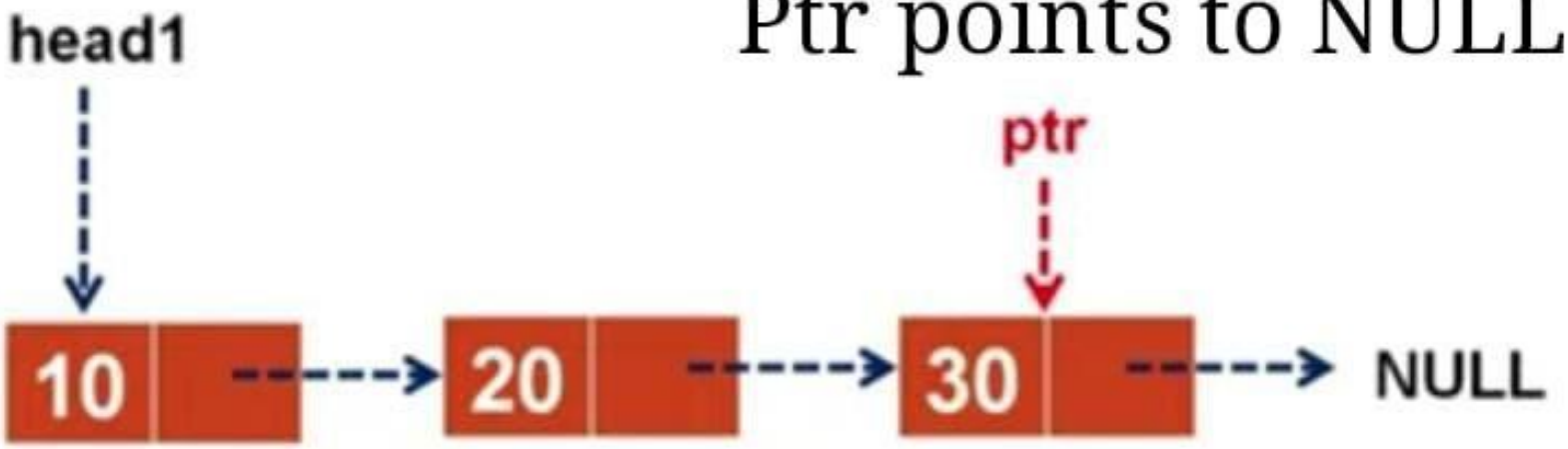
We can merge two linked lists



First find the last node of list1 by checking for NULL. If found we can update the link field of list1 to list 2

Merge

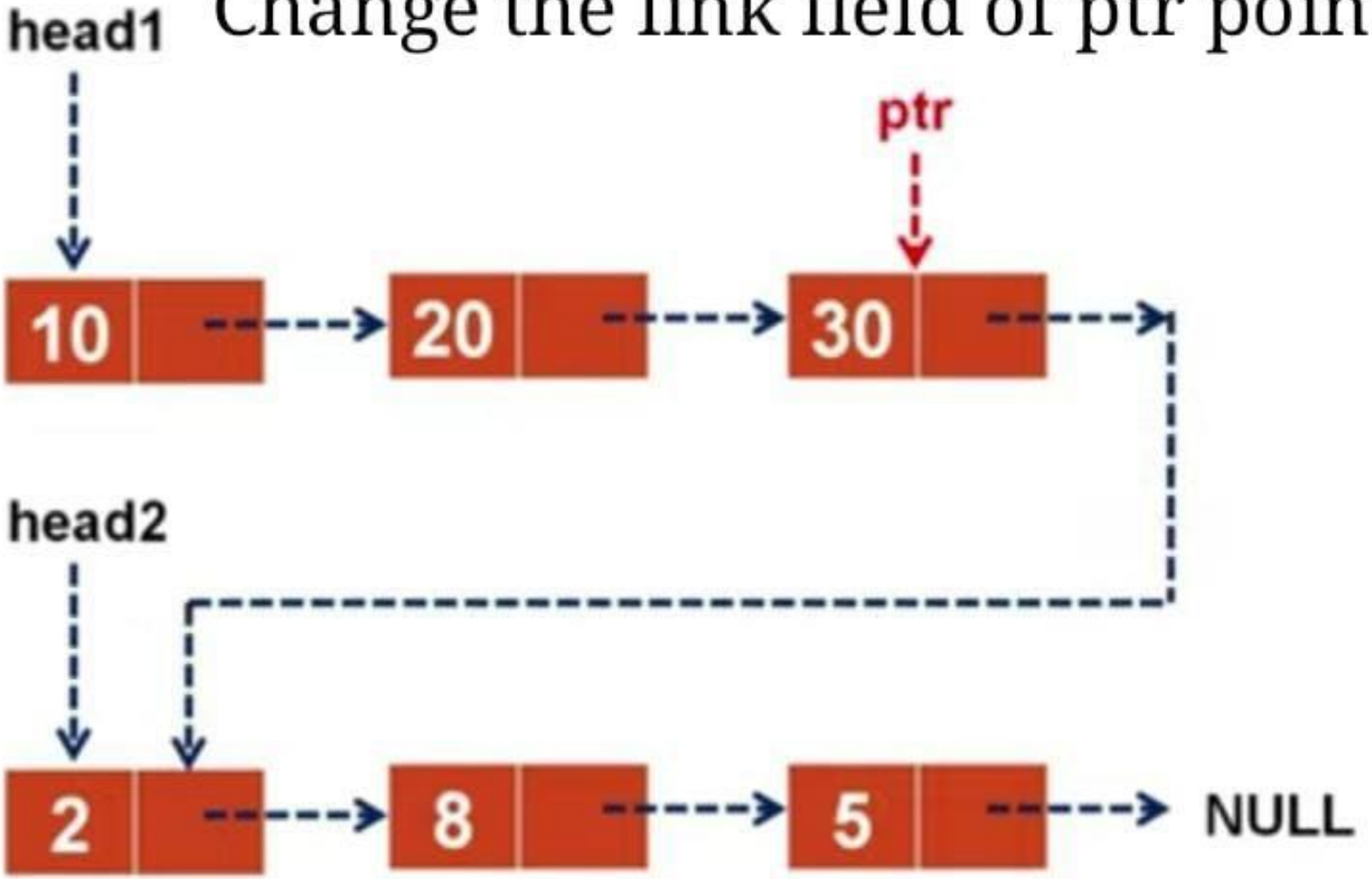
Ptr points to NULL so



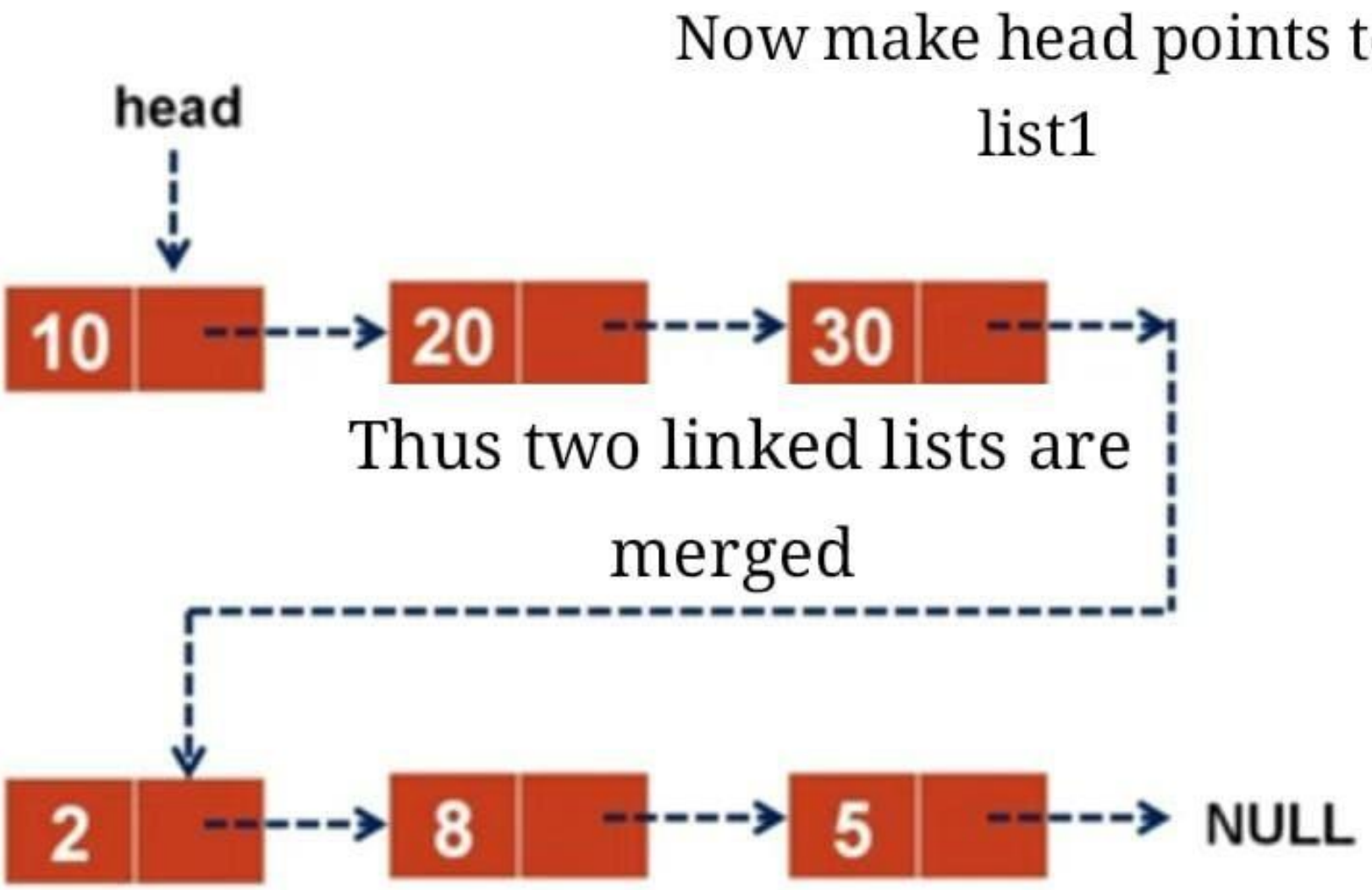
Now we reached the last node of list1

Merge

Change the link field of ptr points to list2



Merge


Now make head points to list1
head
10
20
30
Thus two linked lists are merged
2
8
5
NULL

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