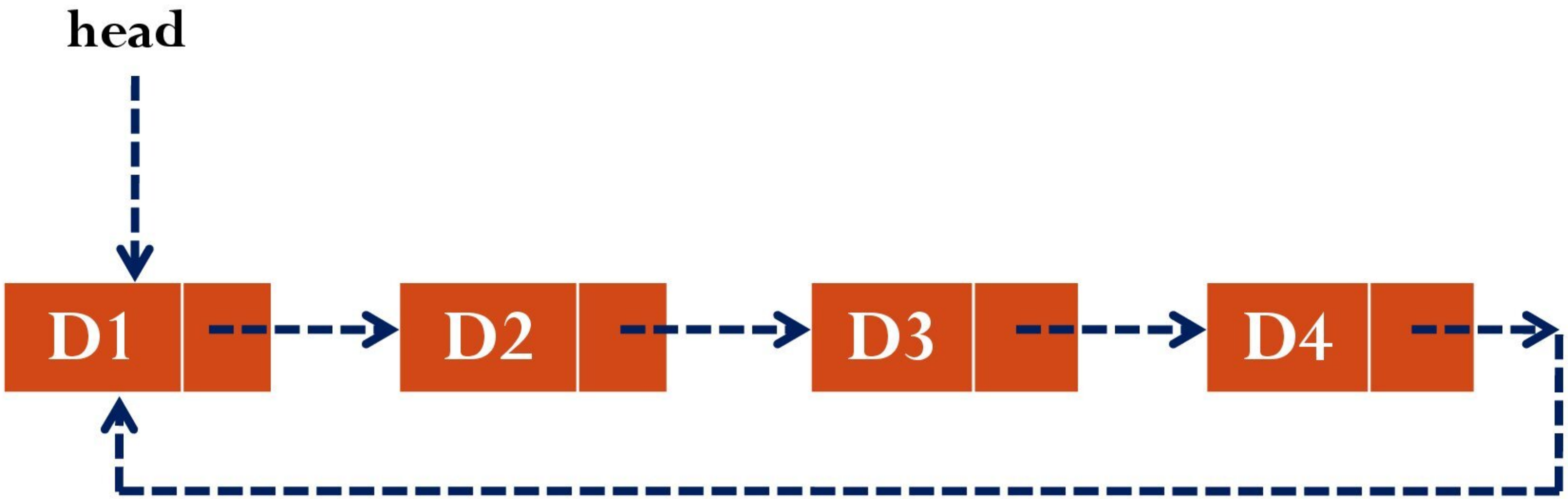
Data Structures – CST 201
Module ~ 3

Syllabus

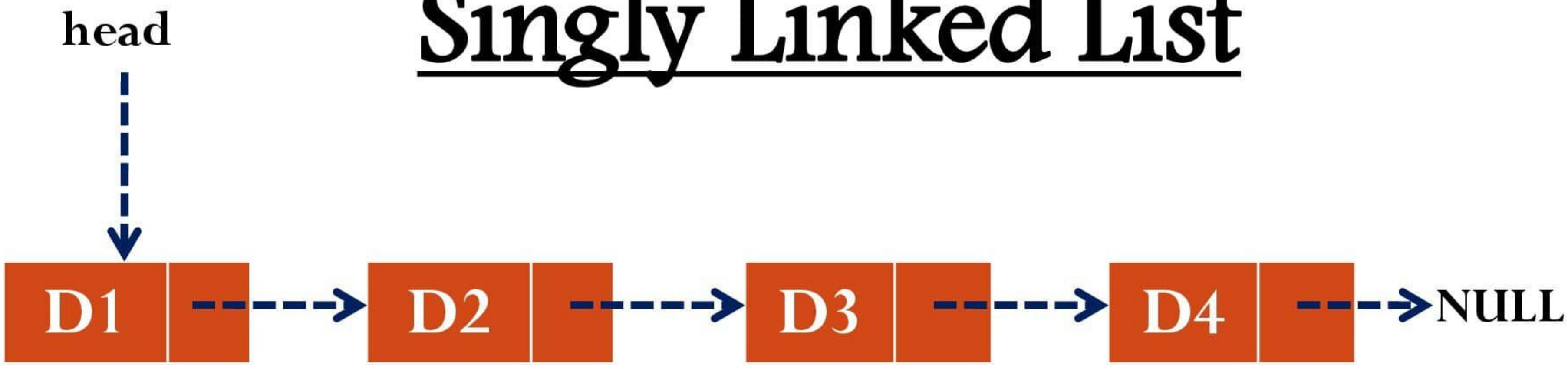
- **Linked List and Memory Management**
  - Self Referential Structures
  - Dynamic Memory Allocation
  - Singly Linked List-Operations on Linked List.
  - Doubly Linked List
  - **Circular Linked List**
  - Stacks using Linked List
  - Queues using Linked List
  - Polynomial representation using Linked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

Circular Linked List

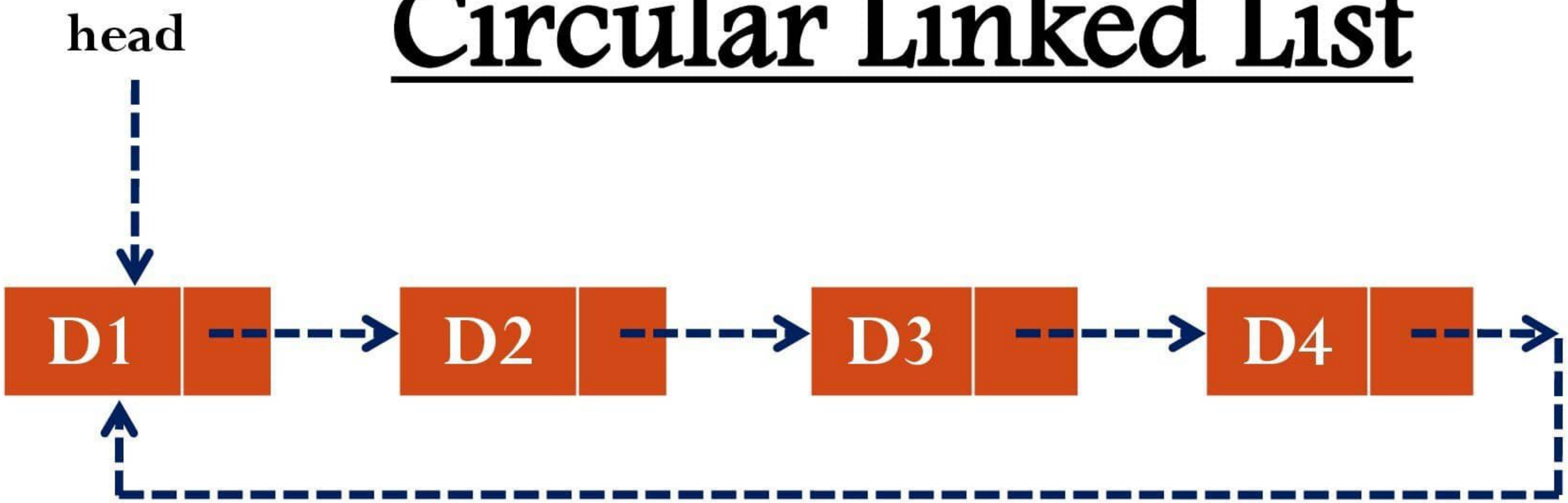
- In a single linked list, the link field of the last node is null.
- A linked list with last node points to the first node is called **circular linked list**



Singly Linked List



Circular Linked List



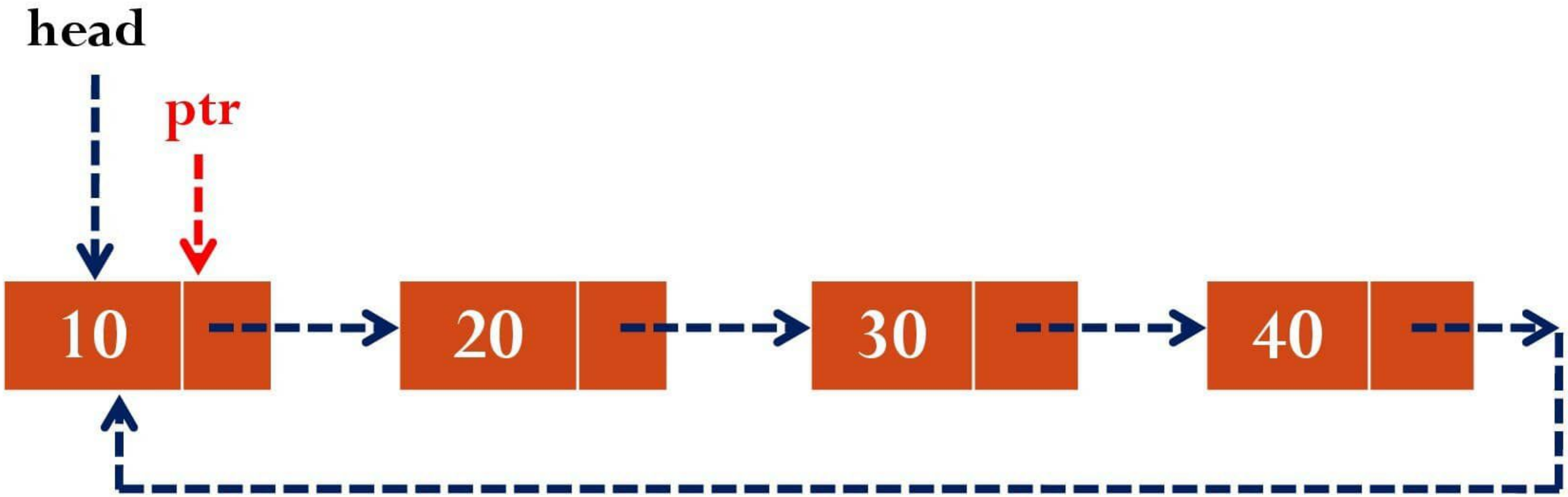
Circular Linked List

- **Advantages**
  - **Accessibility of a member node in the list**: In an ordinary list, a member node is accessible from a particular node, that is, from the header node only. But in the circular linked list, every member node is accessible from any node
  - **Avoided Null link problem**: Null value in link field may create problem during the execution of the program if proper care is not taken
- **Disadvantages**
  - When we are unable to detect the end of the list while moving from one node to the next, system may get trap into in **infinite loop**

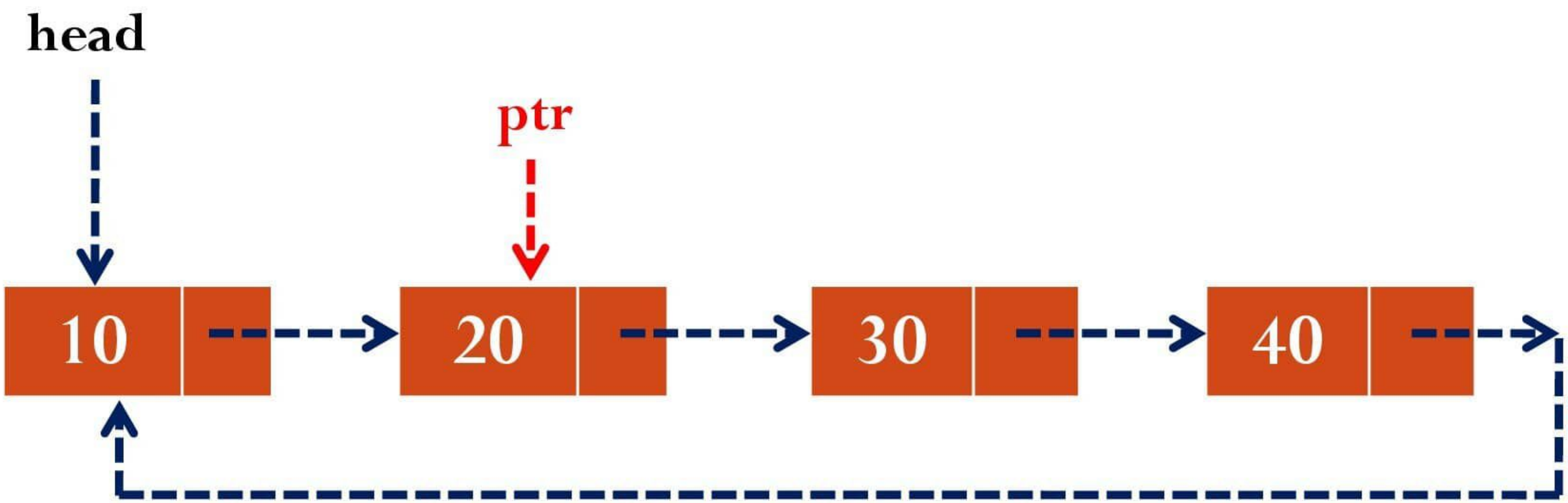
Operations on Circular Linked List

- 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

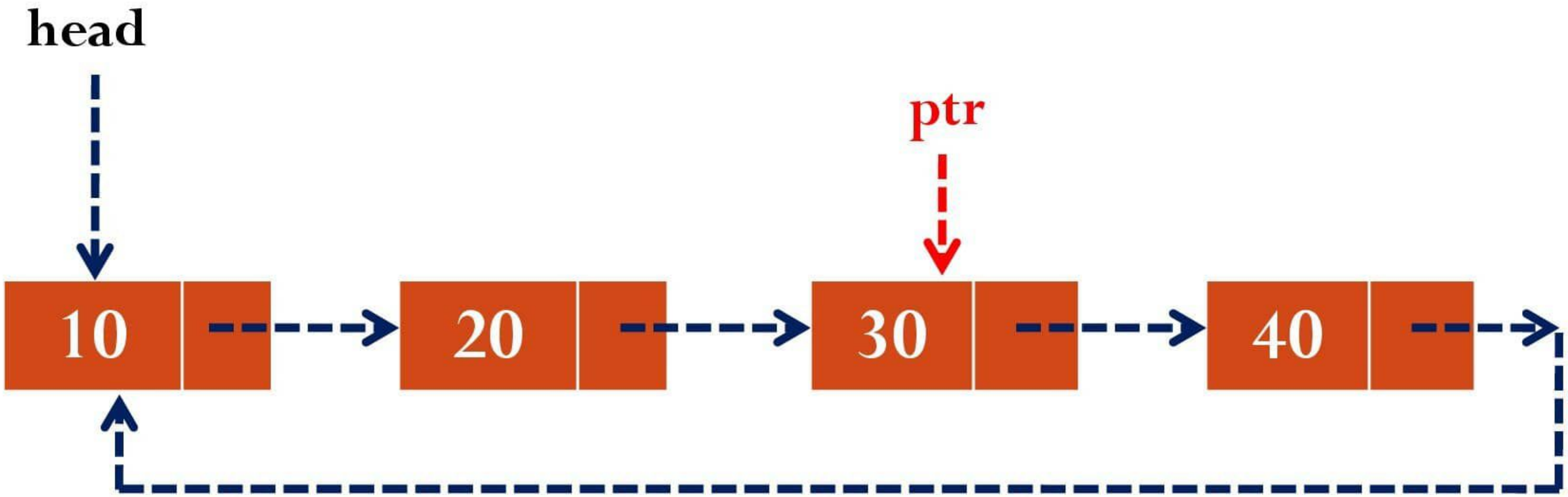
Display/Traversal



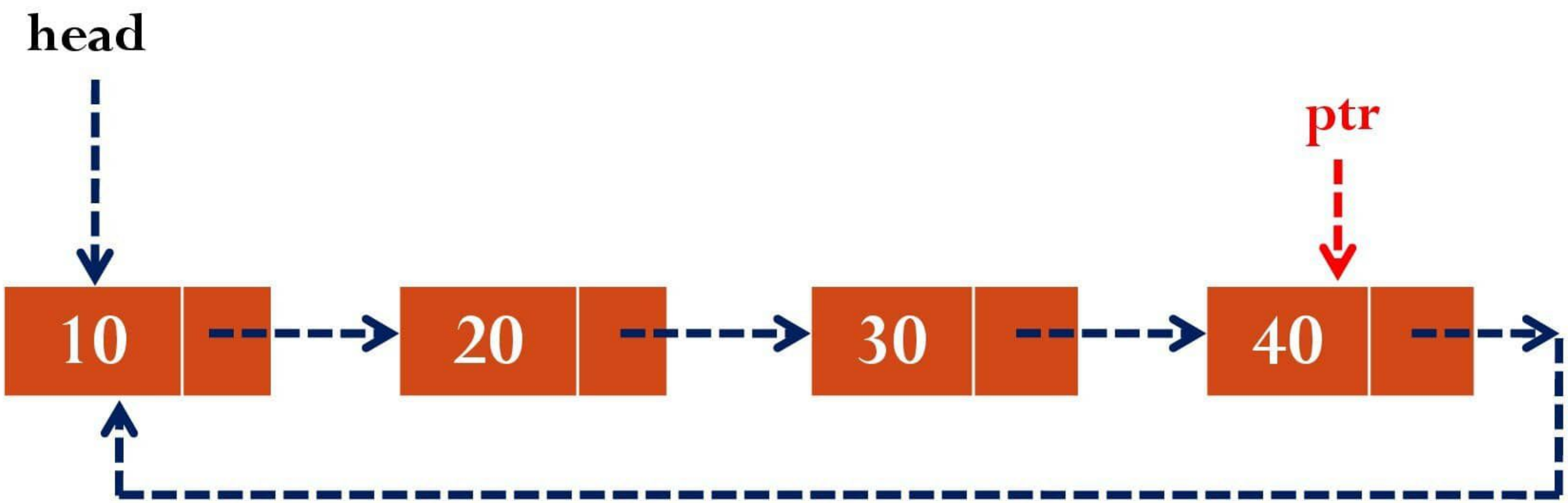
Display/Traversal



Display/Traversal



Display/Traversal



Display/Traversal ~ Algorithm

Algorithm Display(head)

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

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

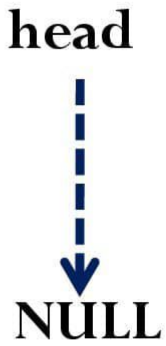
Insert at Front

2 cases:

1. List is empty
2. List is not empty

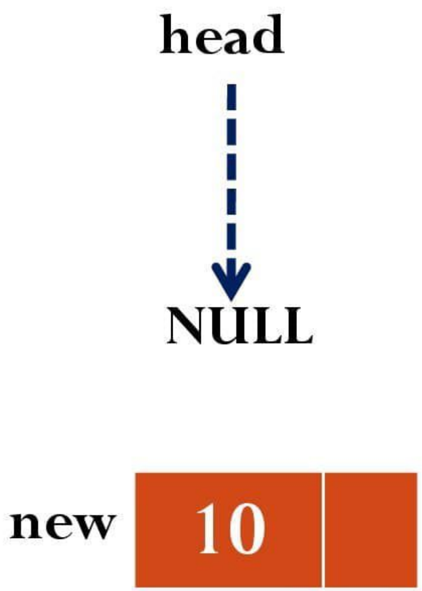
Case 1

Insert at Front



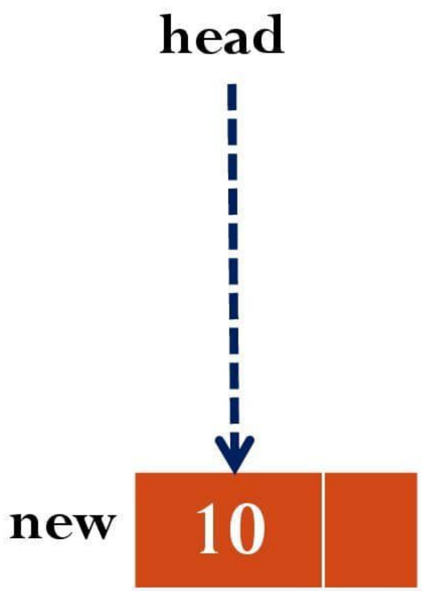
Case 1

Insert at Front



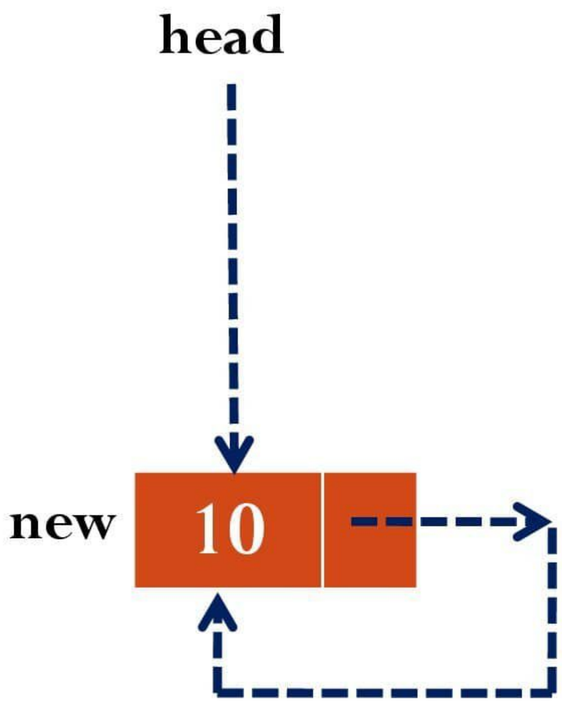
Case 1

Insert at Front



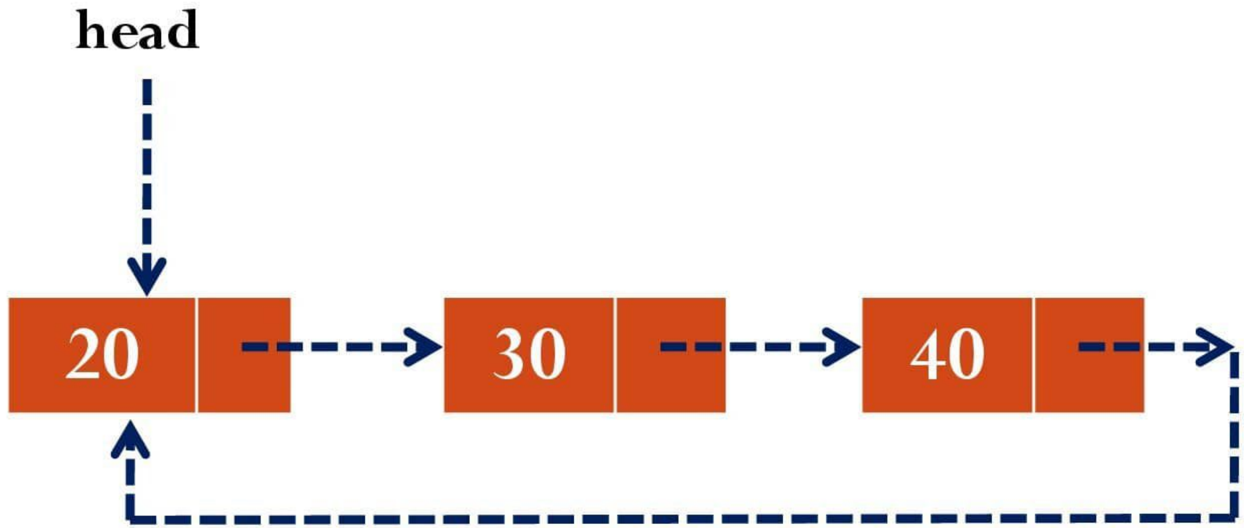
Case 1

Insert at Front



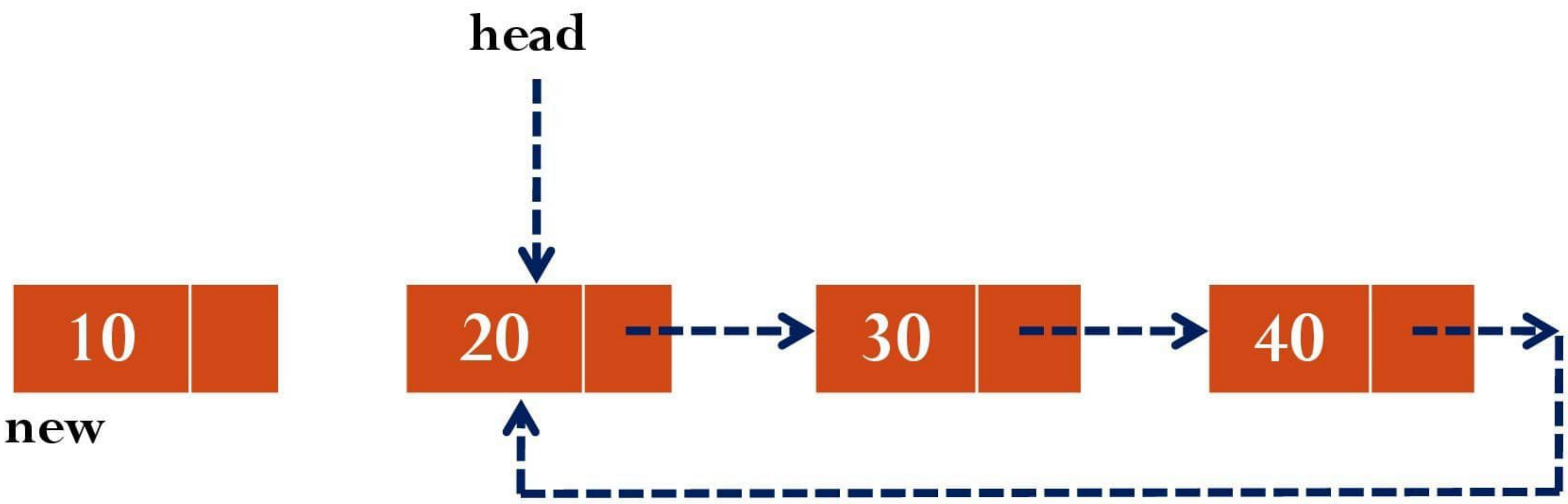
Case 2

Insert at Front



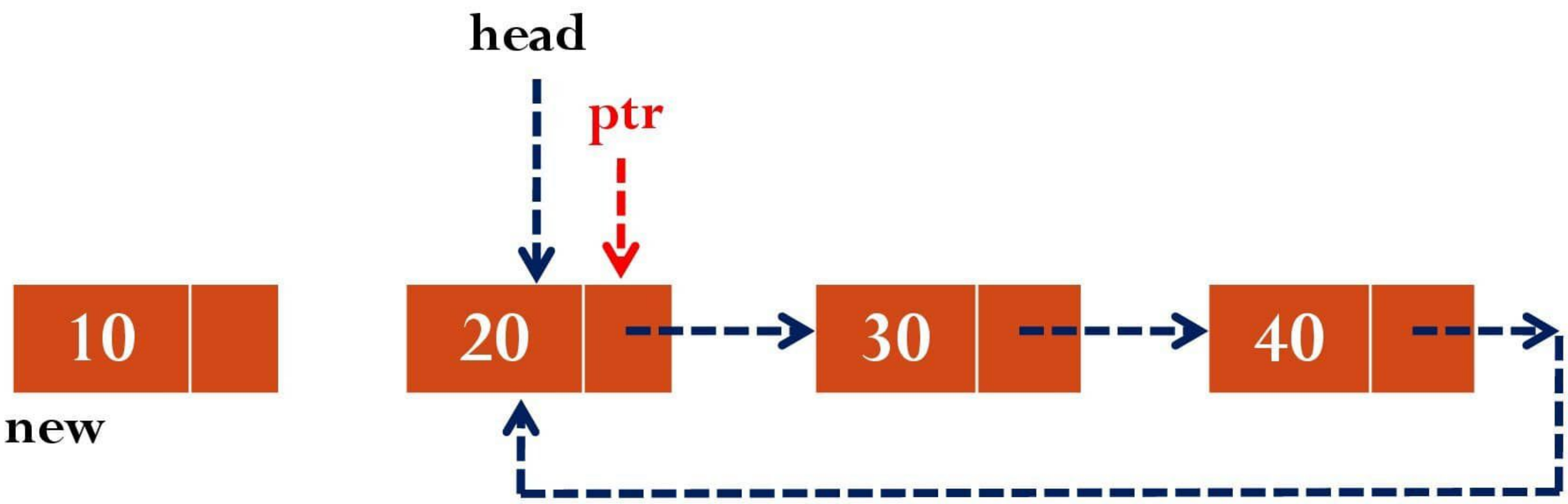
Case 2

Insert at Front



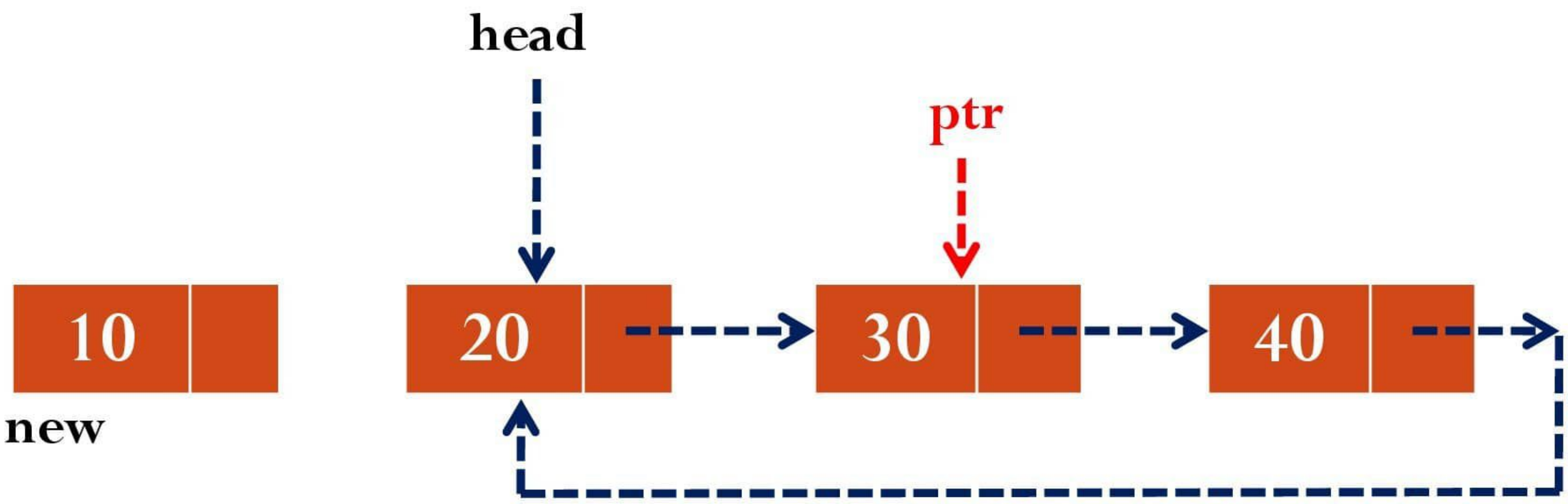
Case 2

Insert at Front



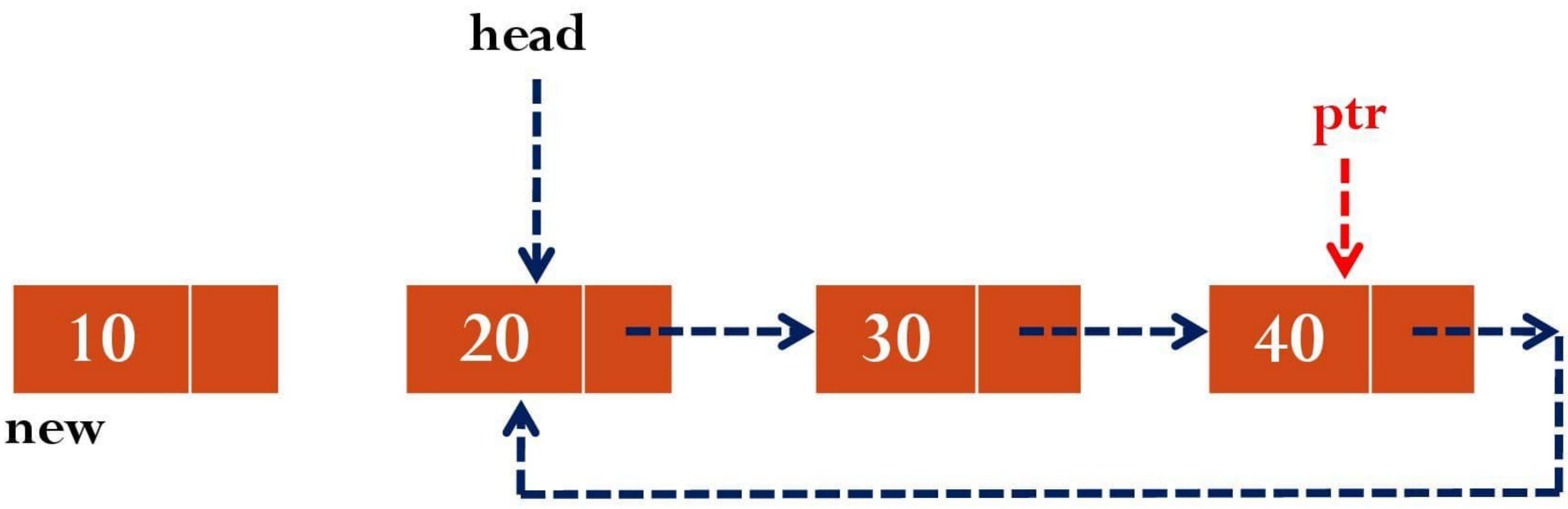
Case 2

Insert at Front



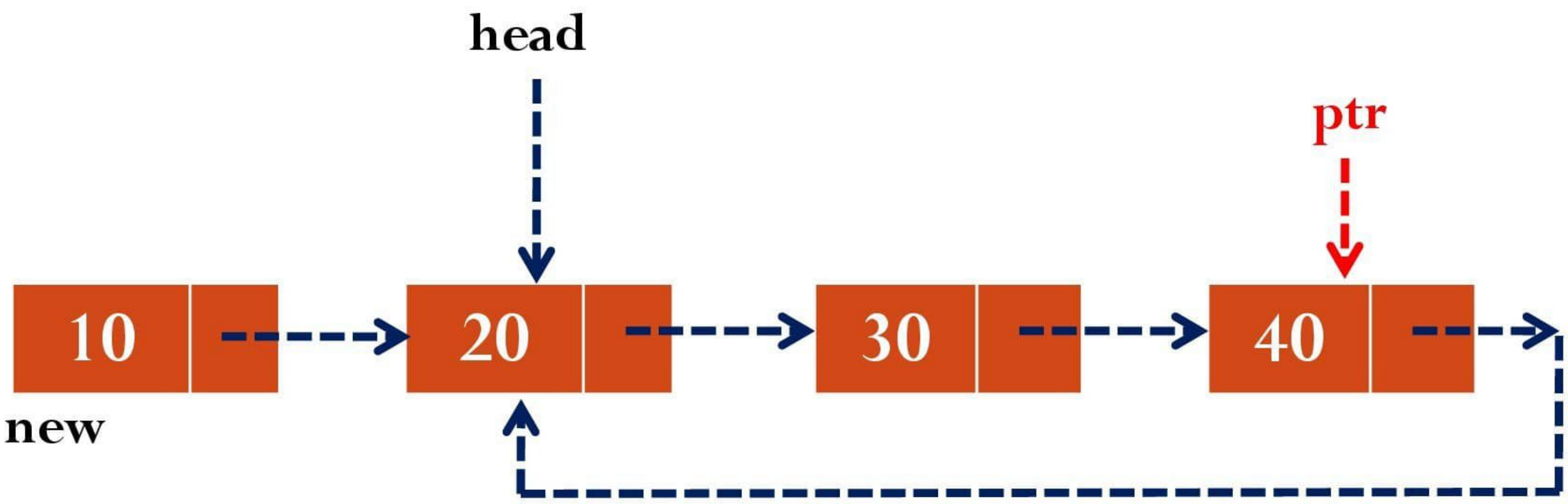
Case 2

Insert at Front



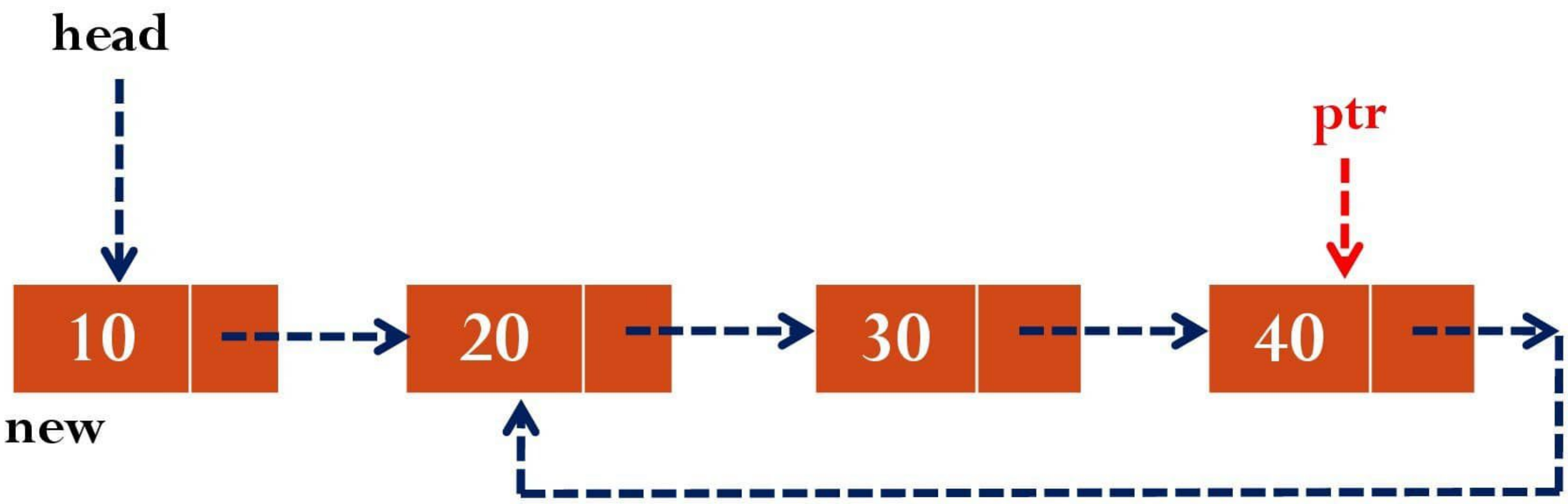
Case 2

Insert at Front



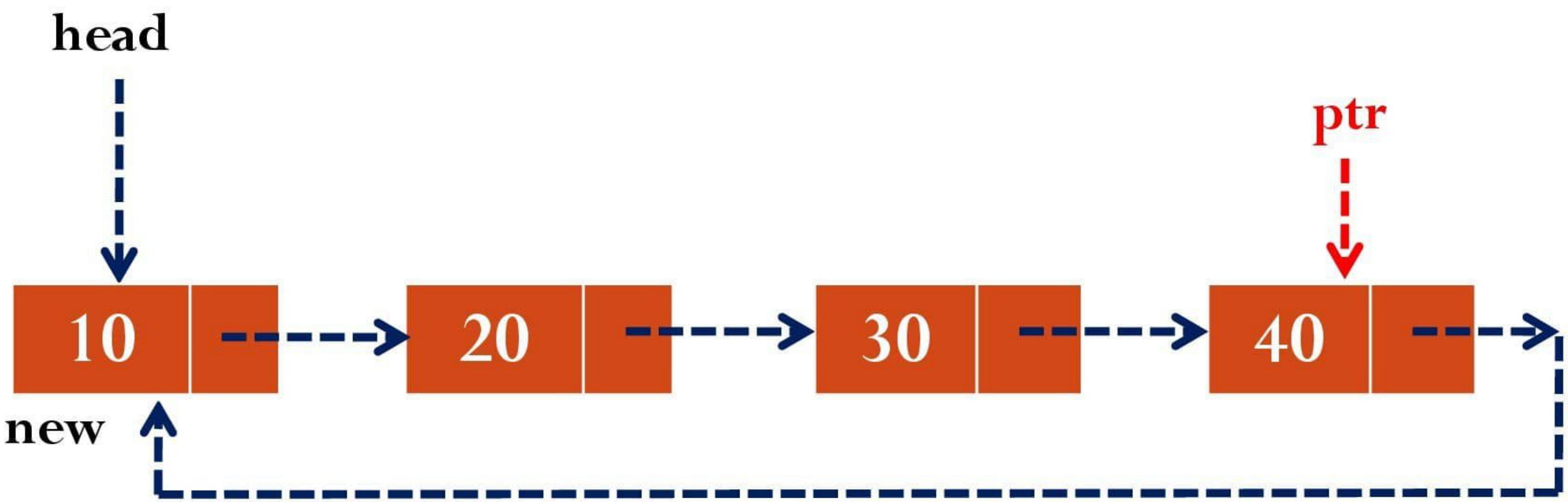
Case 2

Insert at Front



Case 2

Insert at Front



Insert at Front ~ Algorithm

Algorithm Insert_Front(head, x)

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

Insertion

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

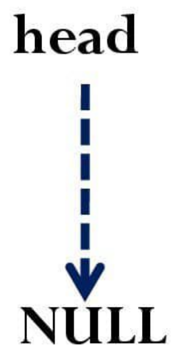
Insert at End

2 cases:

1. List is empty
2. List is not empty

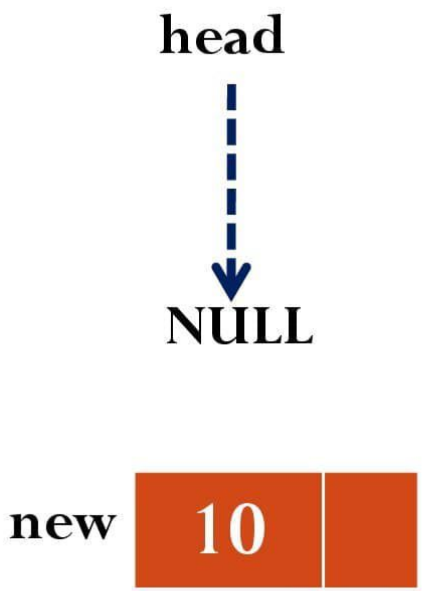
Insert at End

Case 1



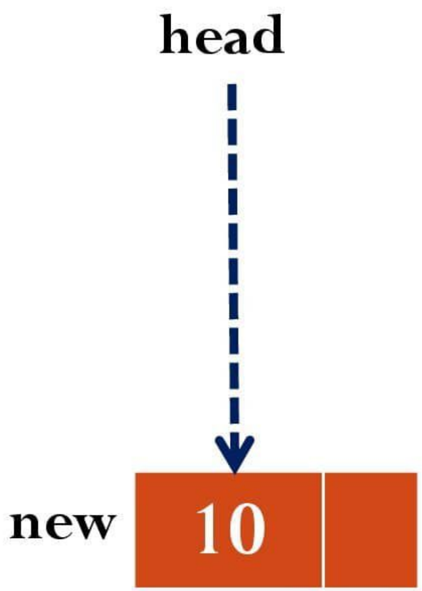
Case 1

Insert at End



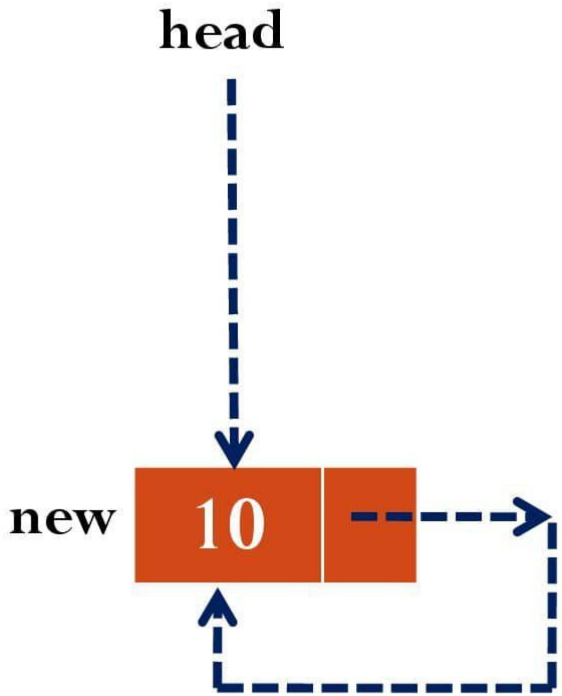
Case 1

Insert at End



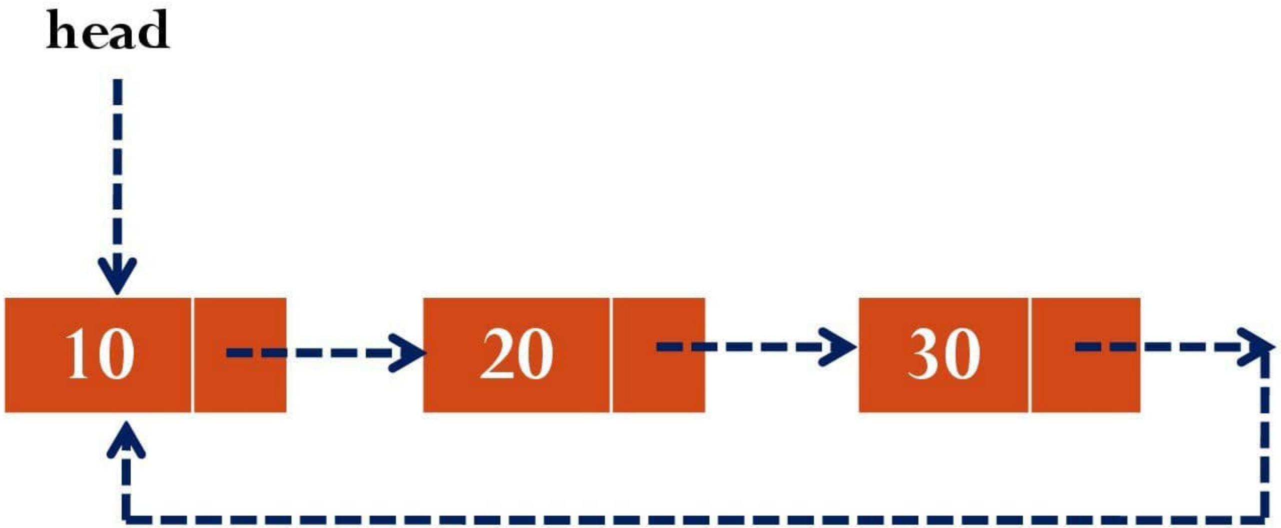
Case 1

Insert at End



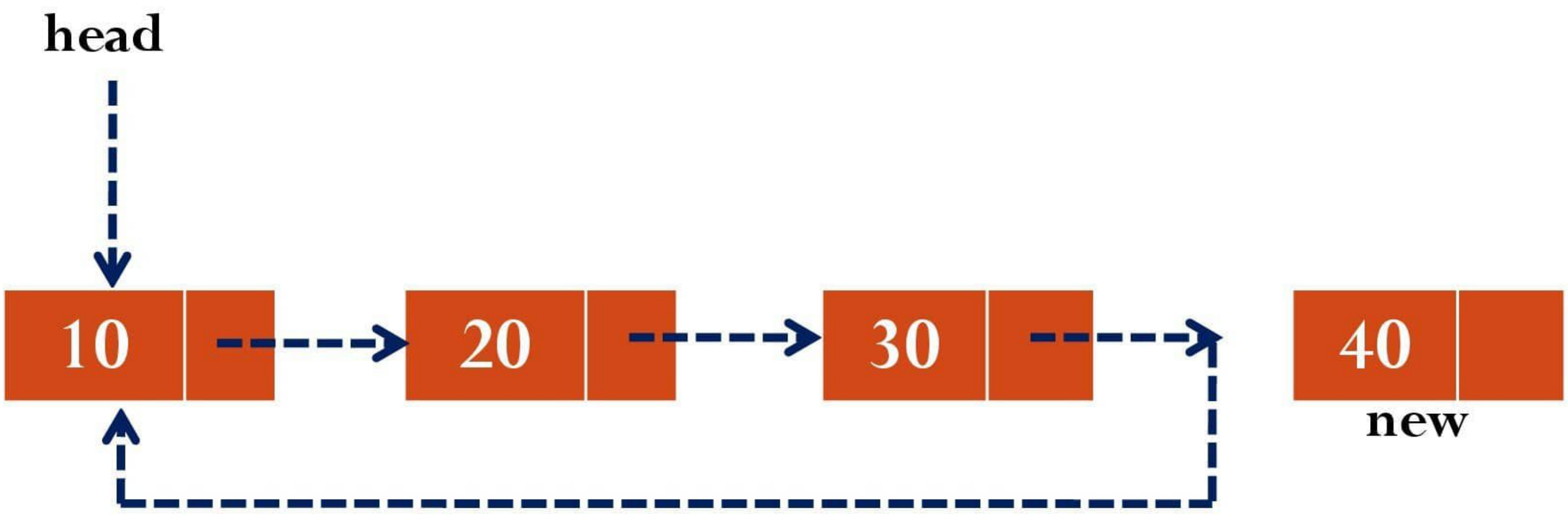
Case 2

Insert at End



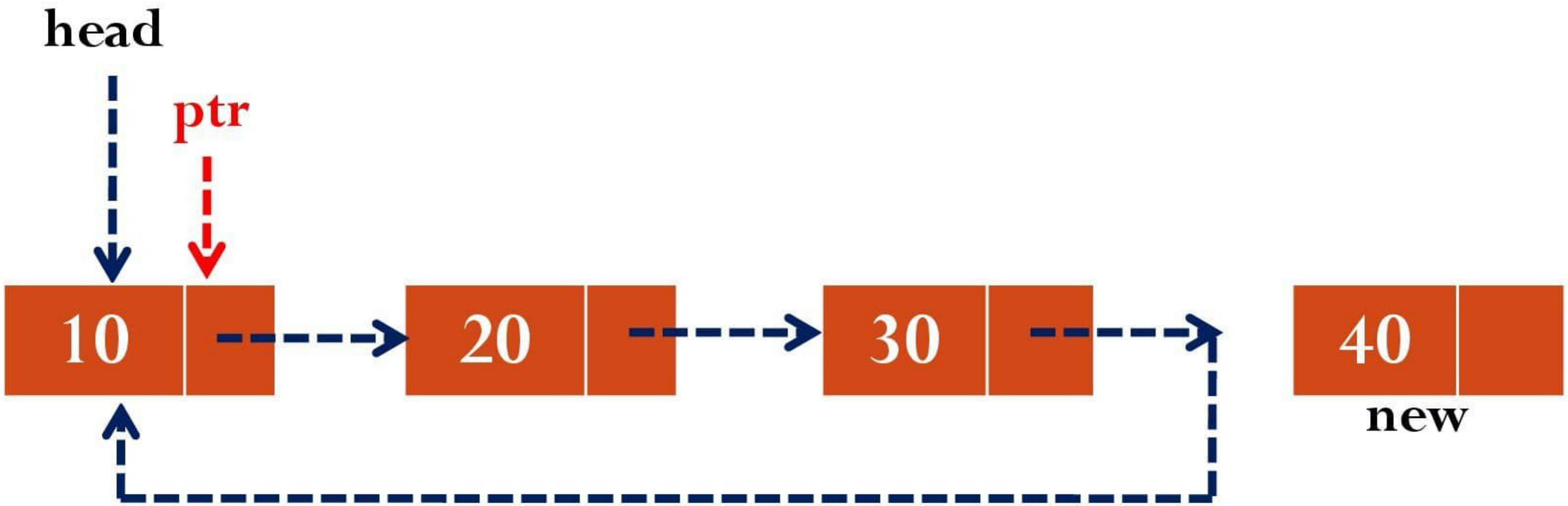
Case 2

Insert at End



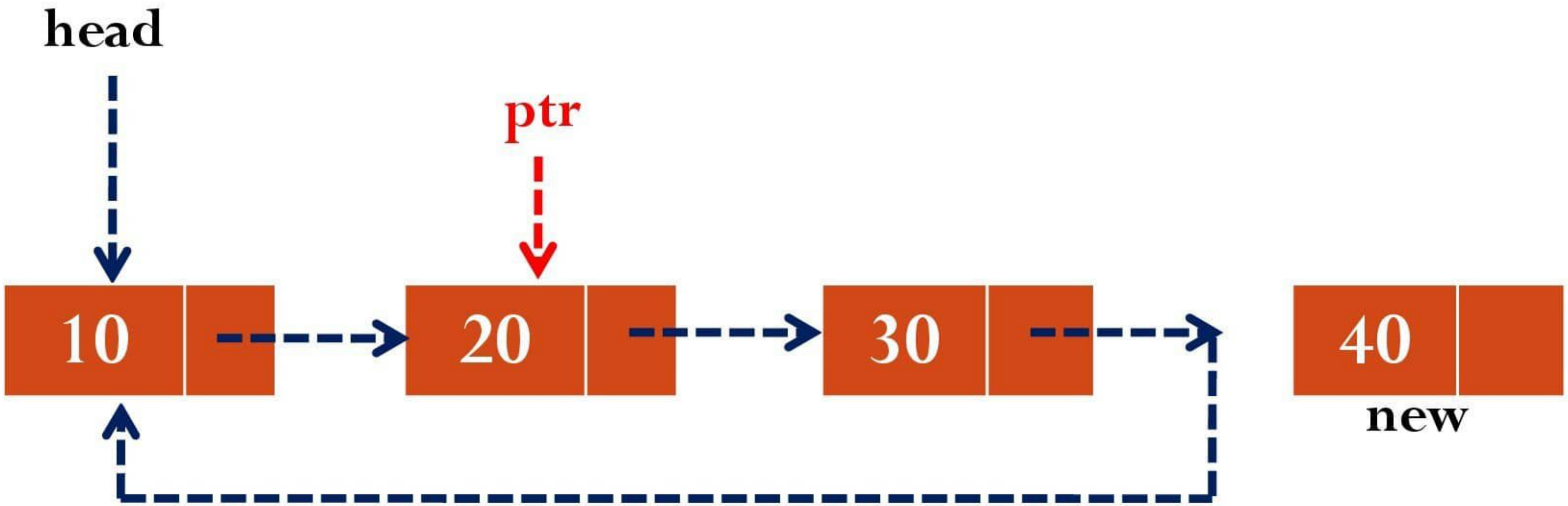
Case 2

Insert at End



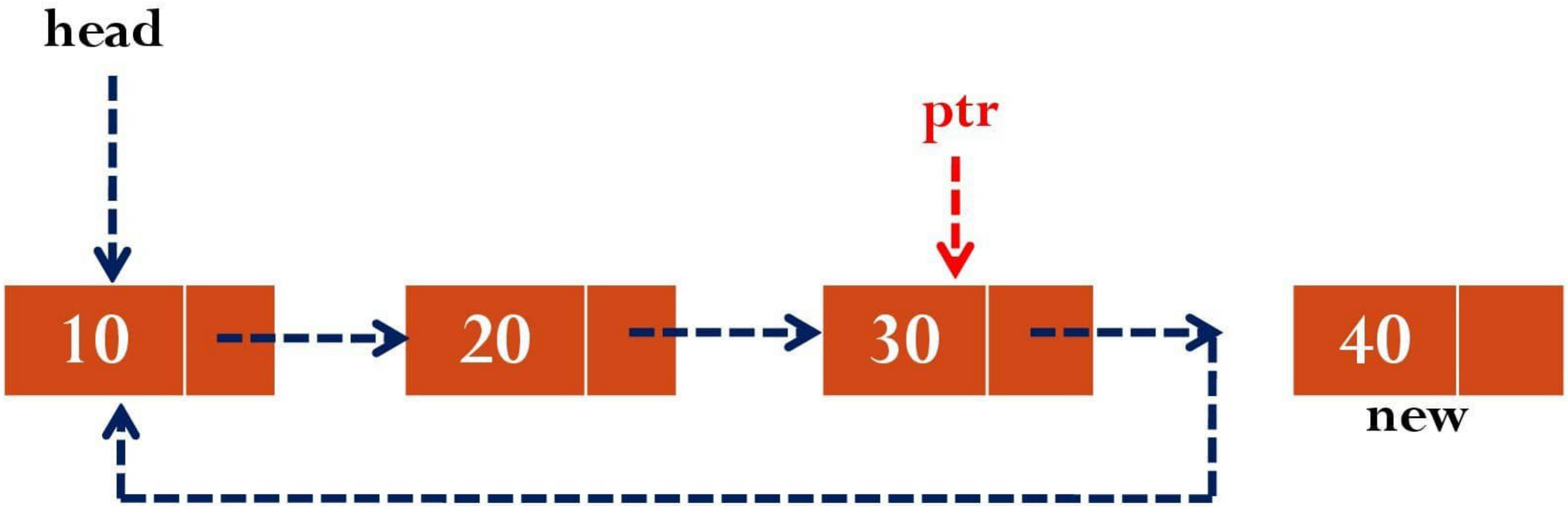
Case 2

Insert at End



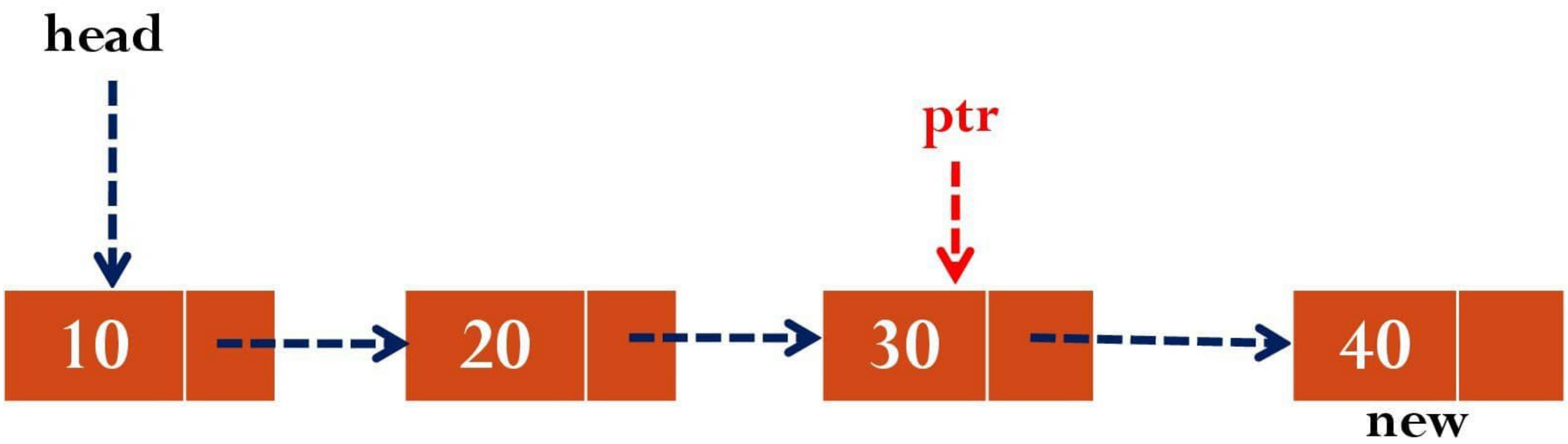
Case 2

Insert at End



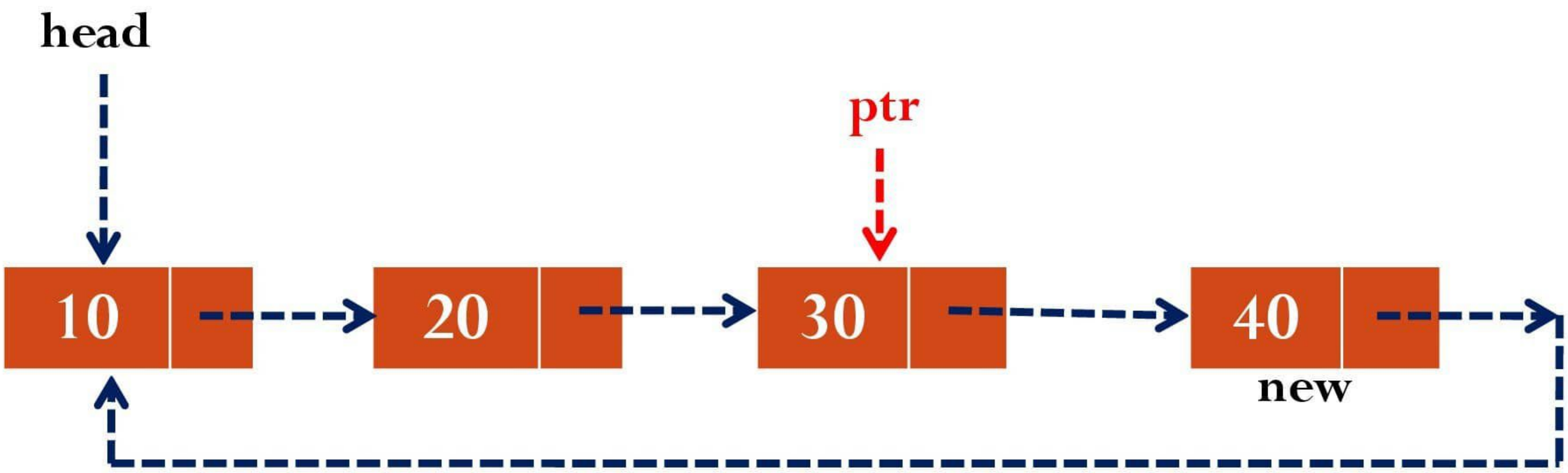
Case 2

Insert at End



Case 2

Insert at End



Insert at End~ Algorithm

Algorithm Insert_End(head, x)

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

Insertion

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

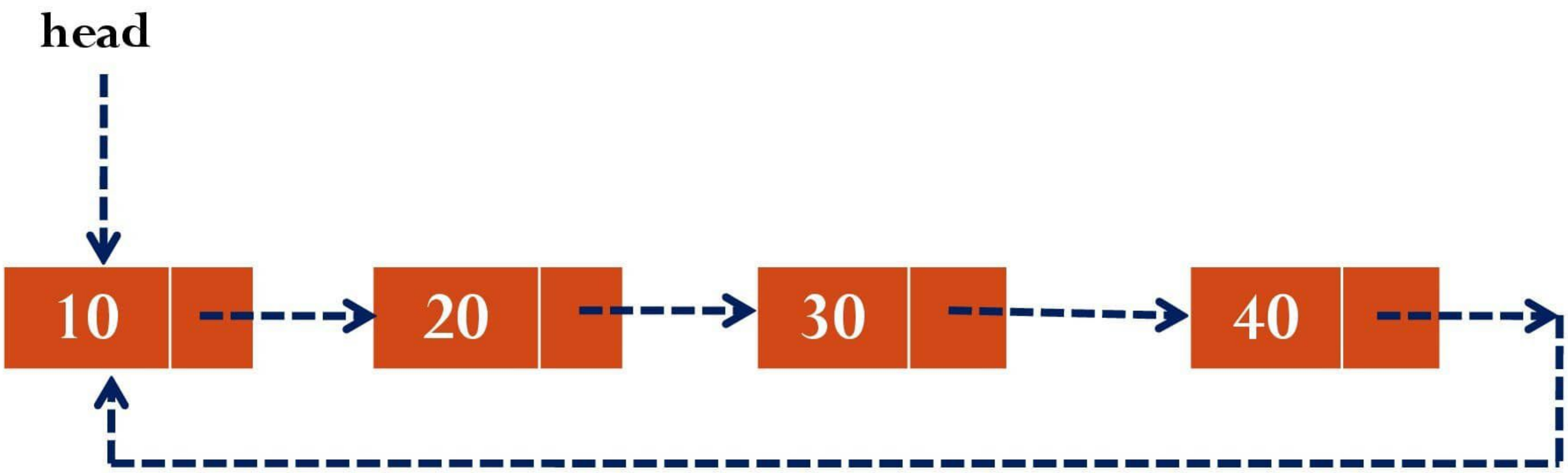
Insert after a specified node

2 cases:

1. List is empty
2. List is not empty

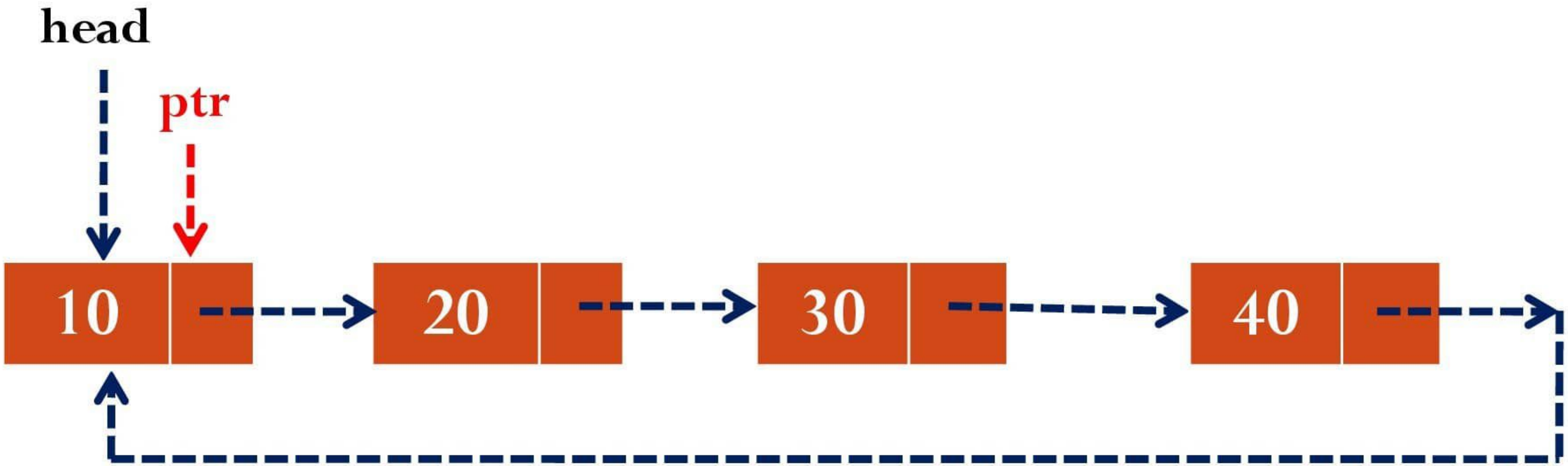
Insert after 30

Case 2



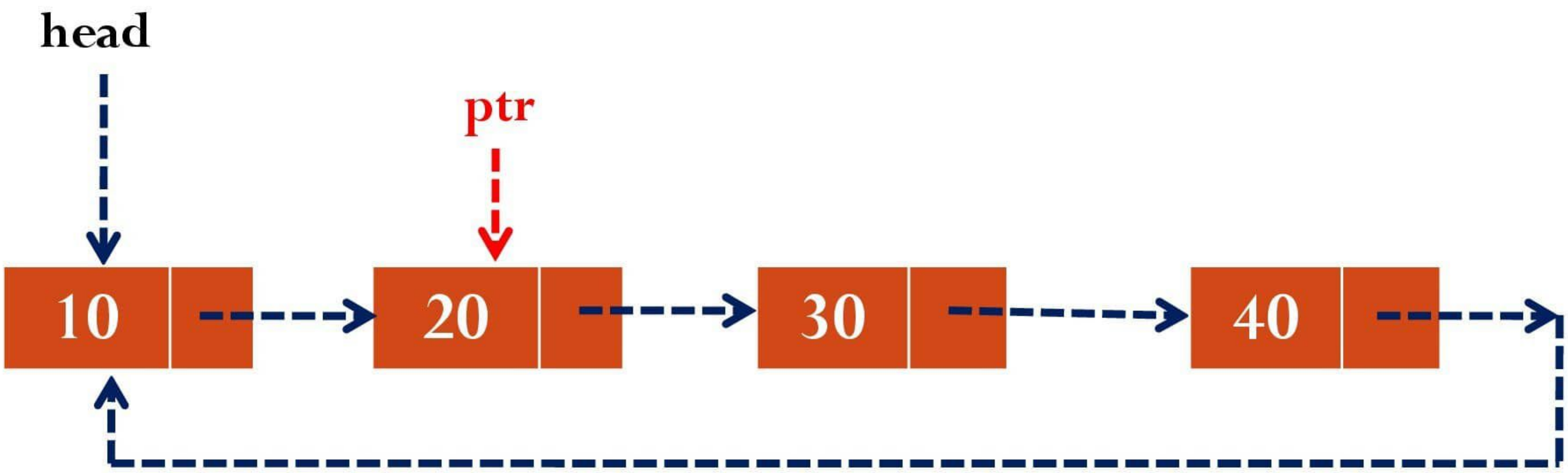
Insert after 30

Case 2



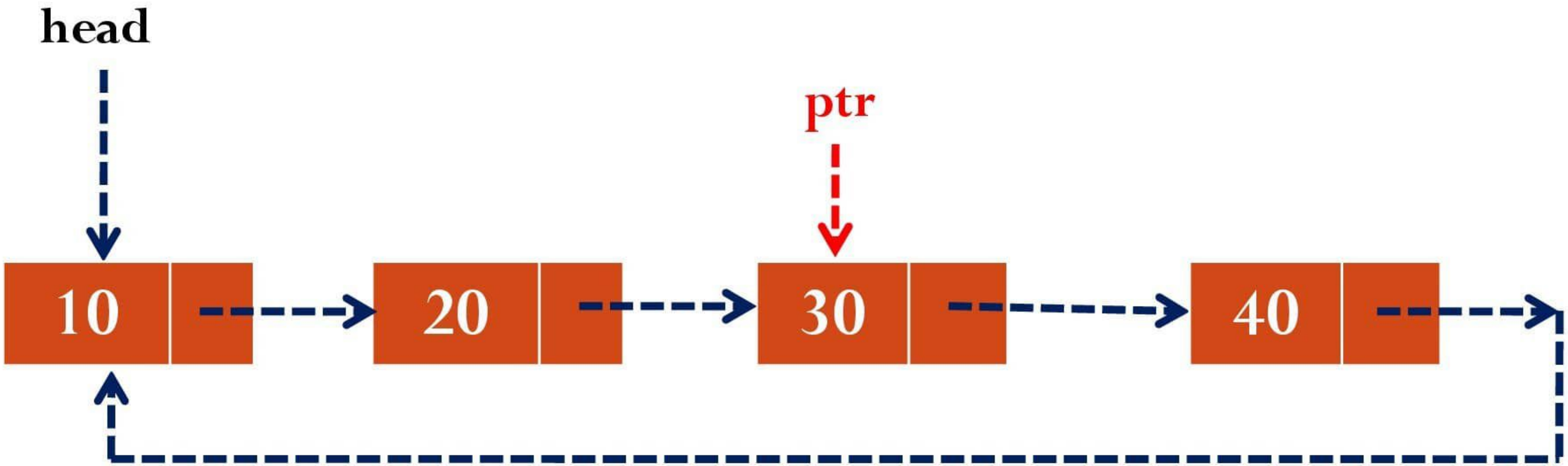
Insert after 30

Case 2



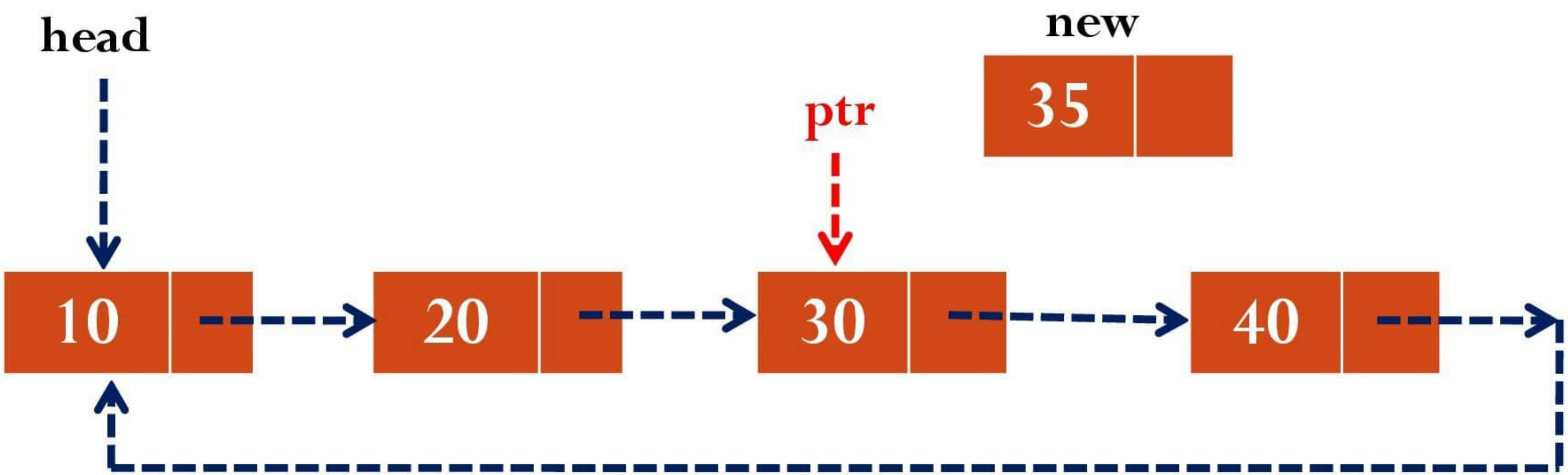
Insert after 30

Case 2



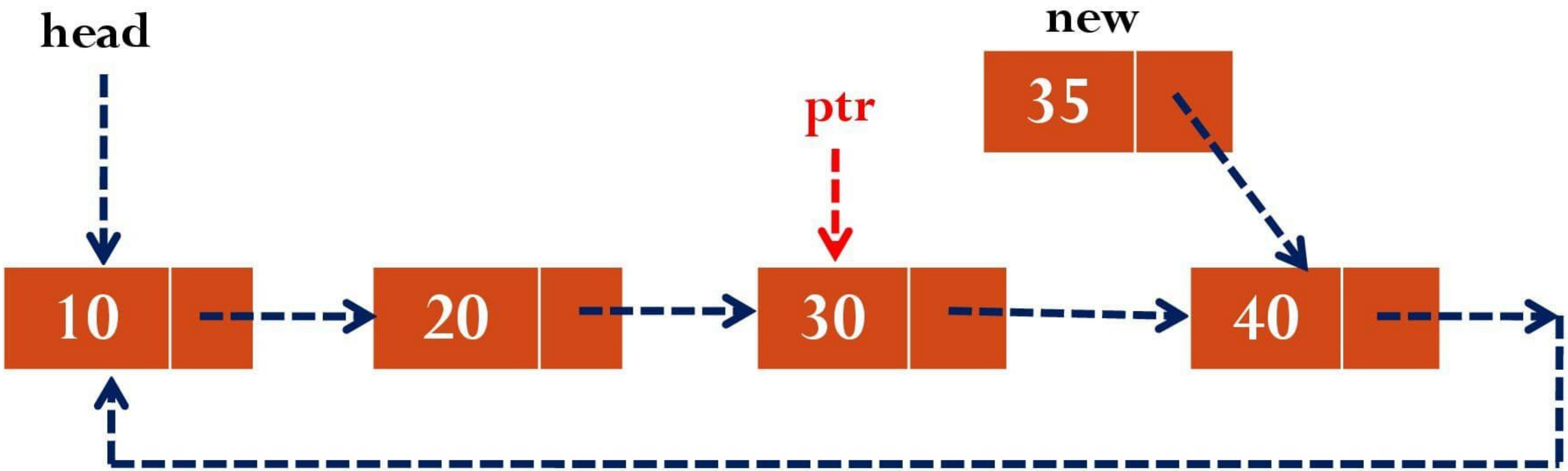
Insert after 30

Case 2



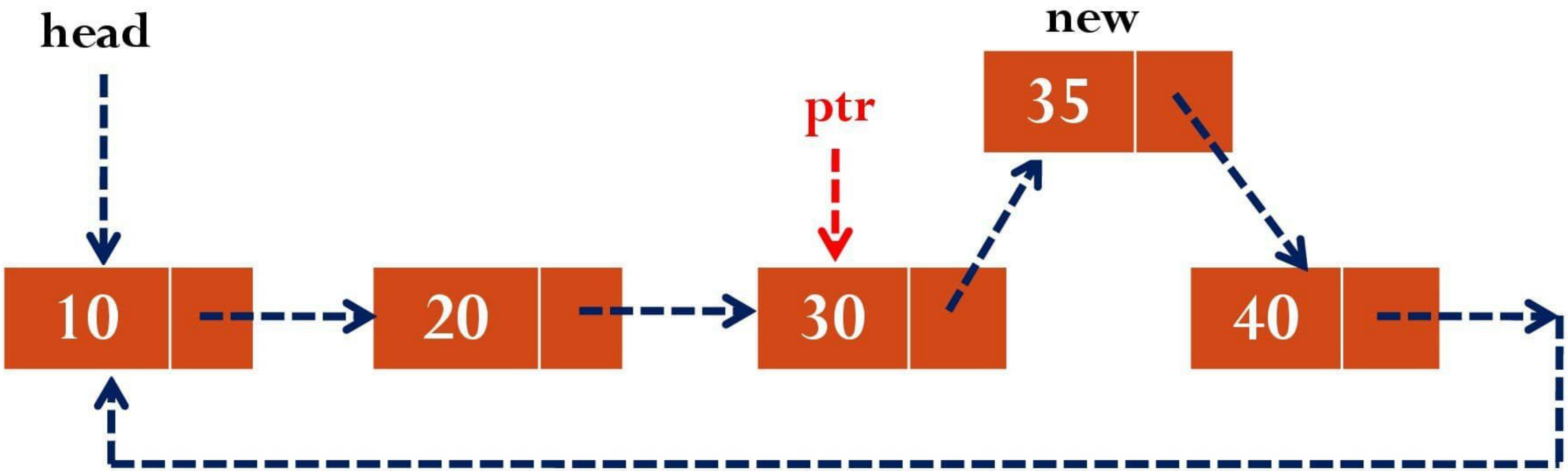
Insert after 30

Case 2



Insert after 30

Case 2



Insert after a specified node - Algorithm

Algorithm Insert_After(head, key,x)

1. If head=NULL then
   1. Print "Search data not found"
2. Else
   1. ptr=head
   2. While ptr→data!=key do
      1. ptr=ptr→link
      2. If ptr=head then
         1. Break
3. If ptr→data=key then
   1. Create a node new
   2. new→data=x
   3. new→link=ptr→link
   4. ptr→link=new
4. Else
   3. Print "Search data not found. Insertion not possible."

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

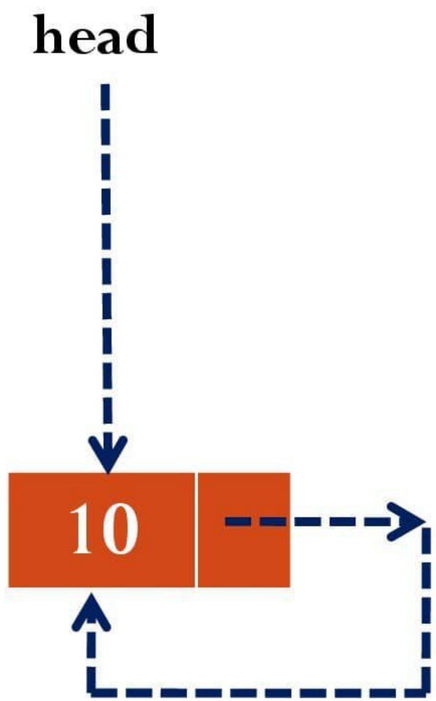
Delete from Front

2 cases:

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

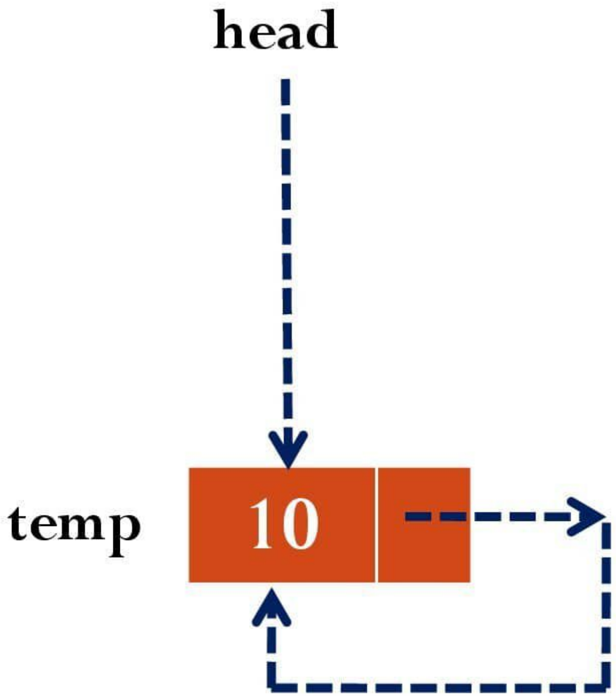
Case 2

Delete from Front



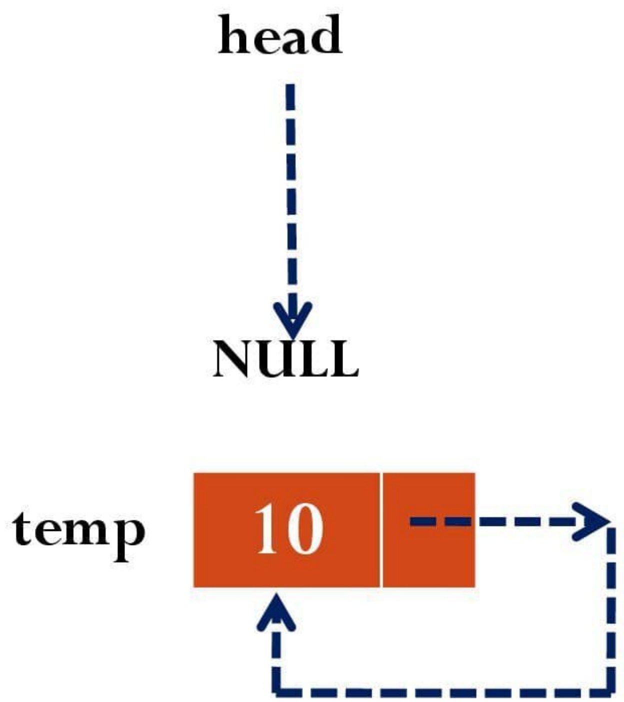
Case 2

Delete from Front



Case 2

Delete from Front

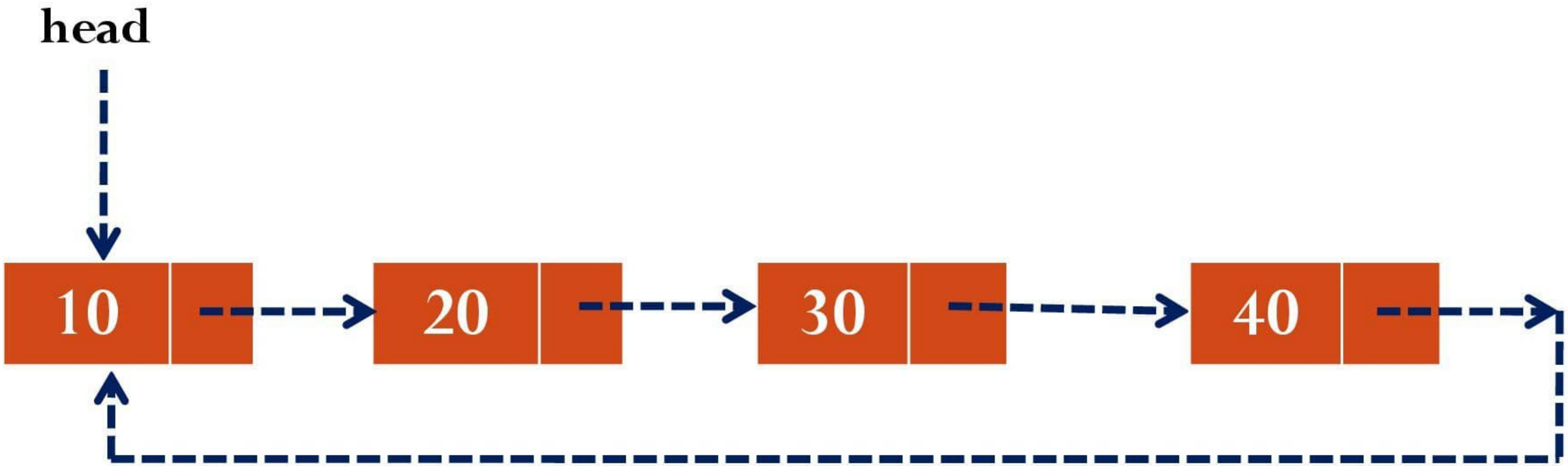


Case 2

Delete from Front

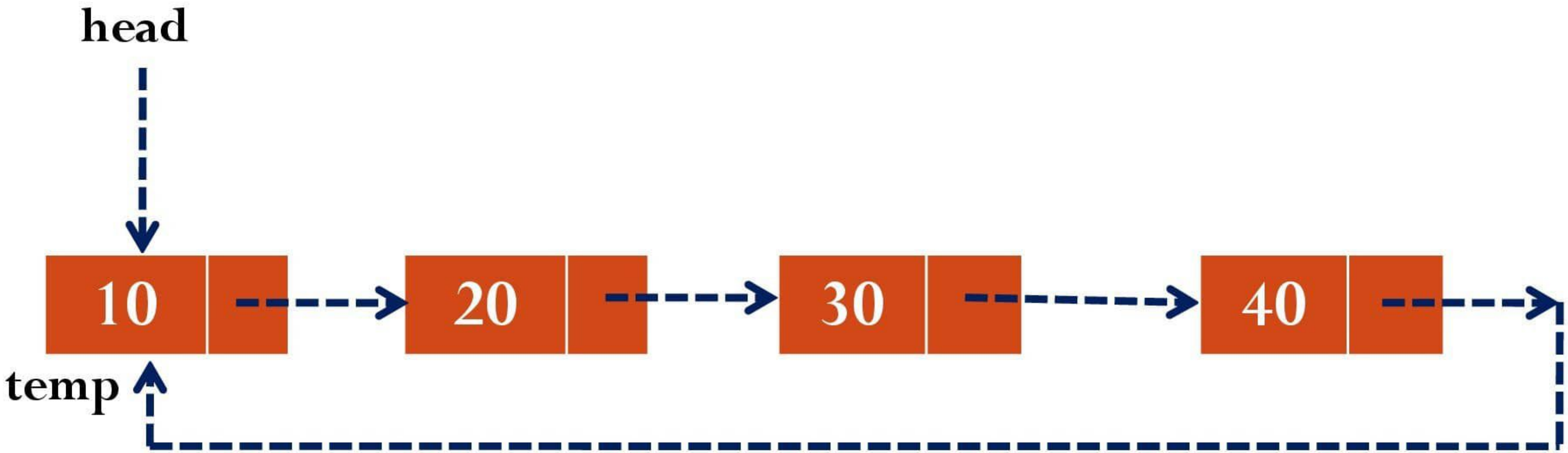
Case 3

Delete from Front



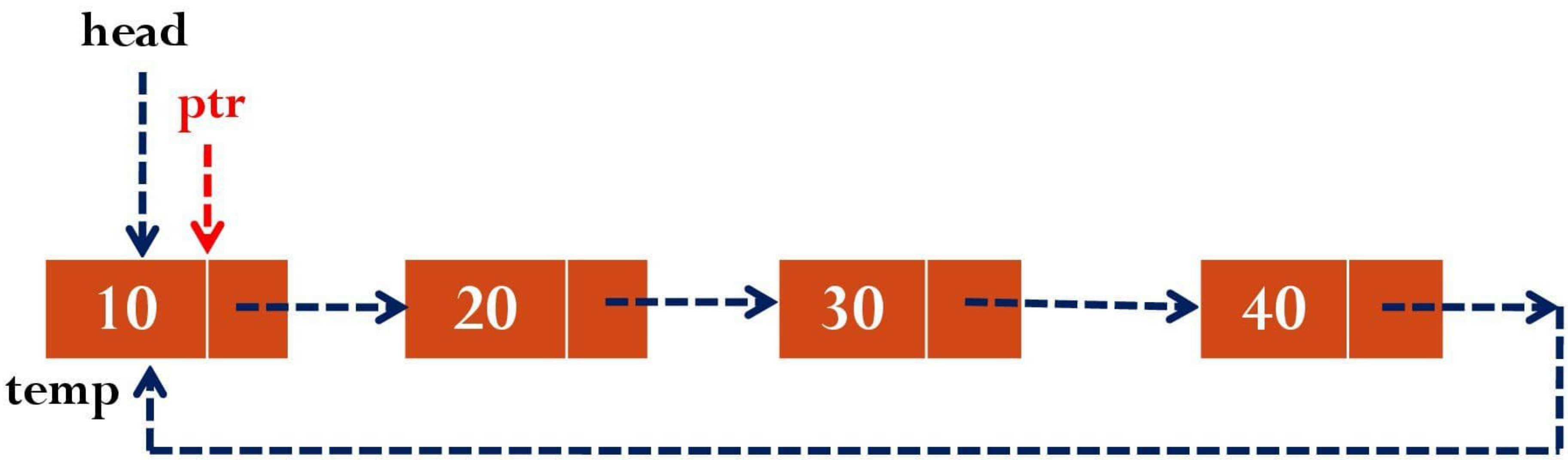
Case 3

Delete from Front



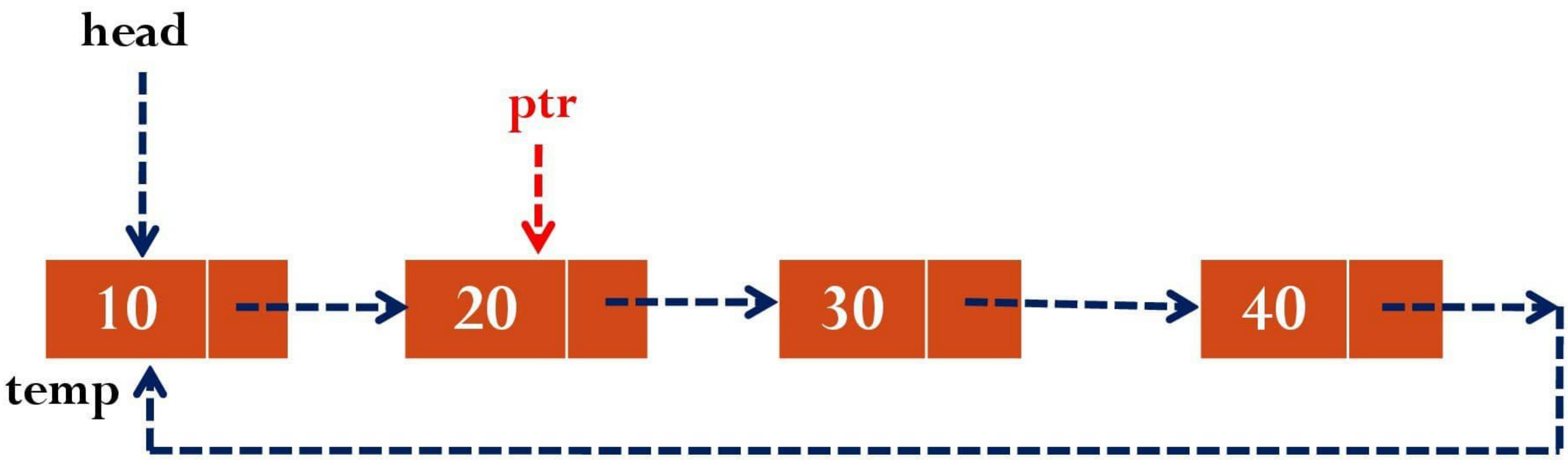
Case 3

Delete from Front



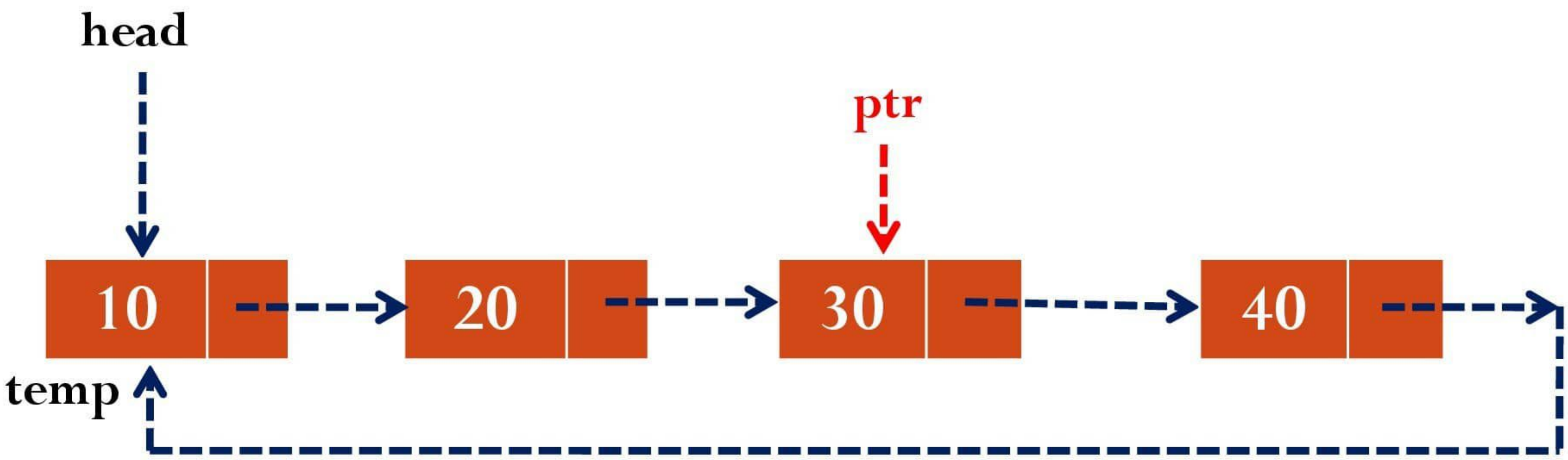
Case 3

Delete from Front



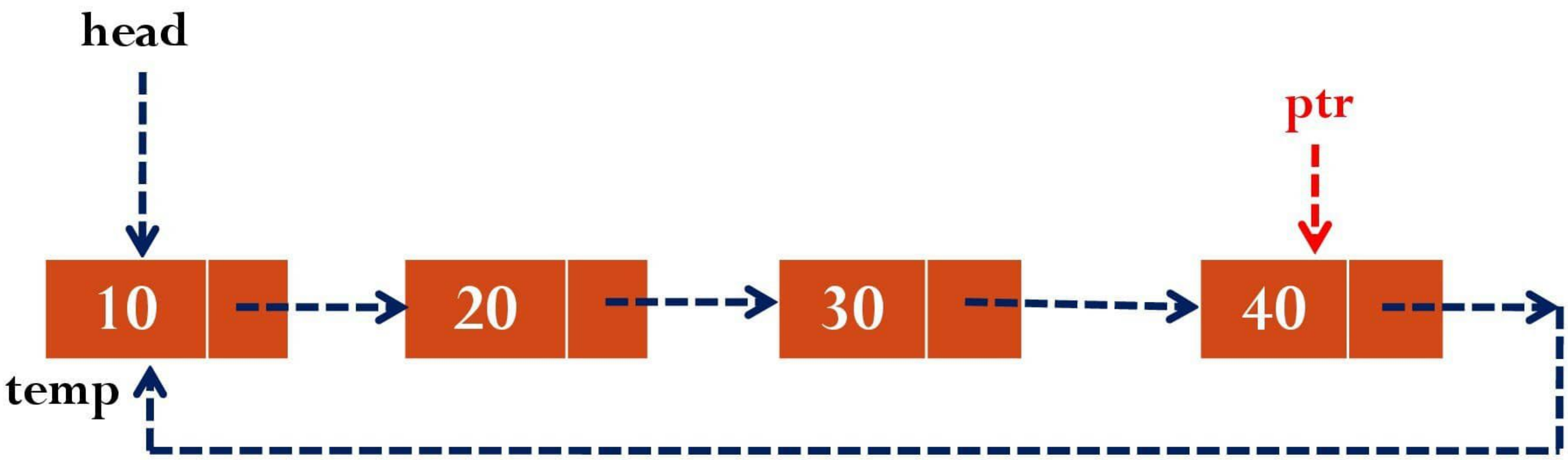
Case 3

Delete from Front



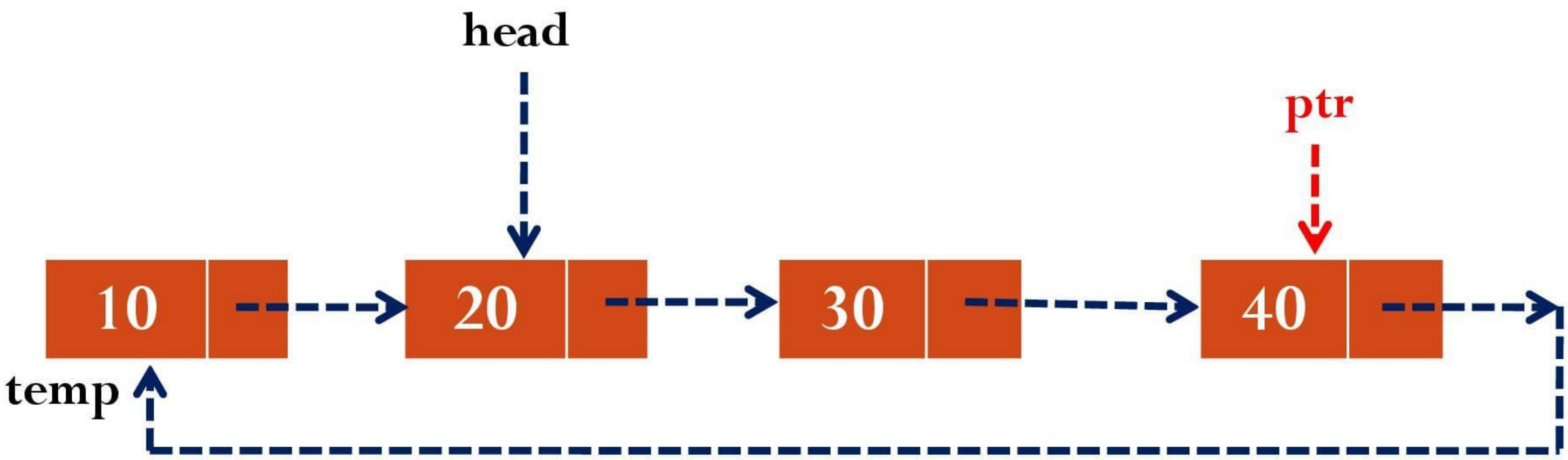
Case 3

Delete from Front



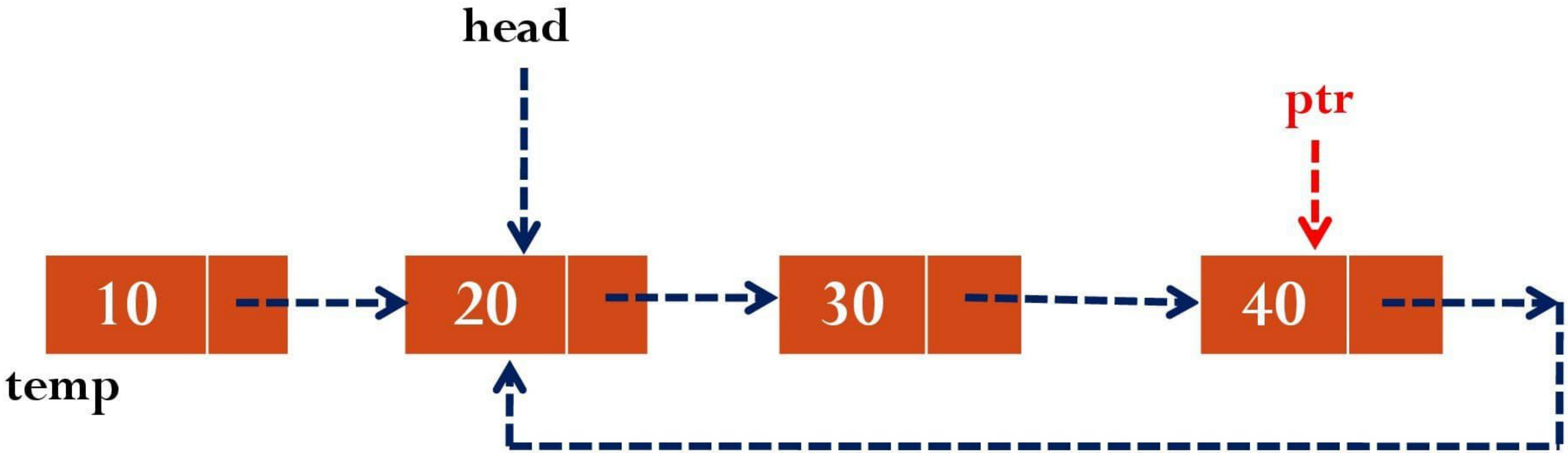
Case 3

Delete from Front



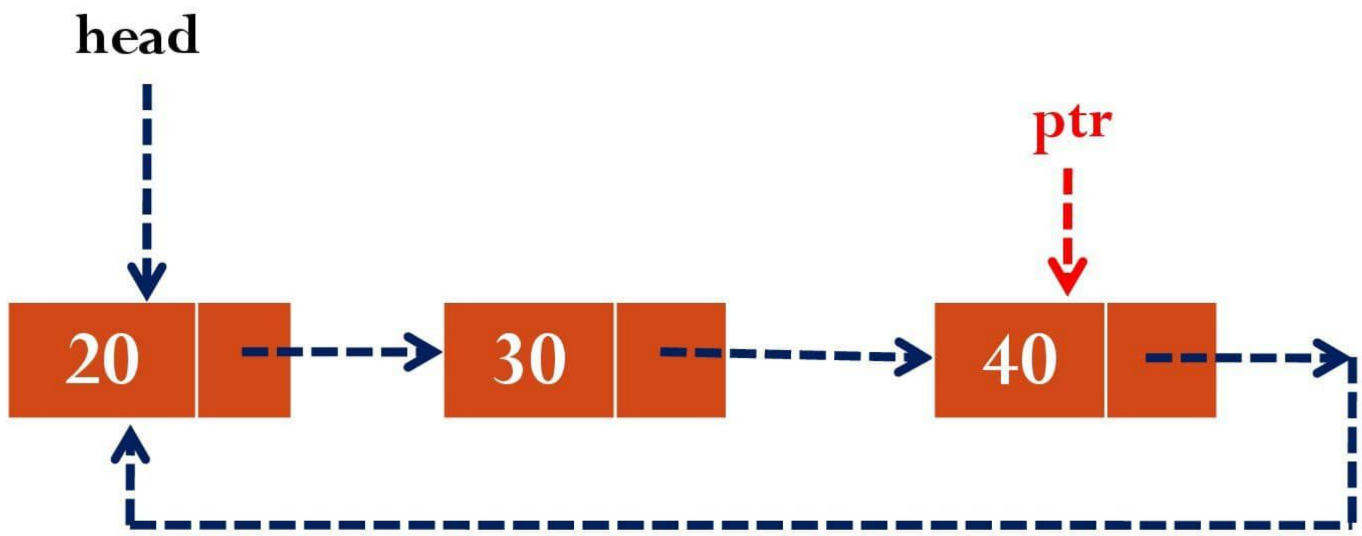
Case 3

Delete from Front



Case 3

Delete from Front



Delete from Front~ Algorithm

Algorithm Delete_Front(head)

1. If head=NULL then
   1. Print "List is Empty. Deletion is not possible"
2. Else if head→link=head then
   1. temp=head
   2. head=NULL
   3. dispose(temp)
3. Else
   1. temp=ptr=head
   2. While ptr→link!=head do
      1. ptr=ptr→link
   3. head=head→link
   4. ptr→link=head
   5. dispose(temp)

Deletion

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

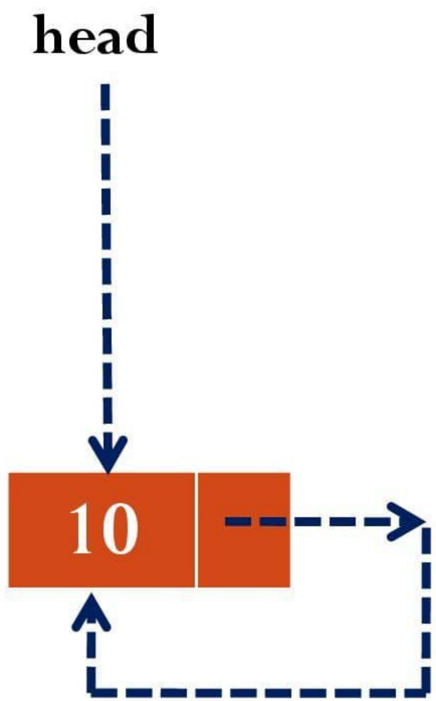
Delete from End

2 cases:

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

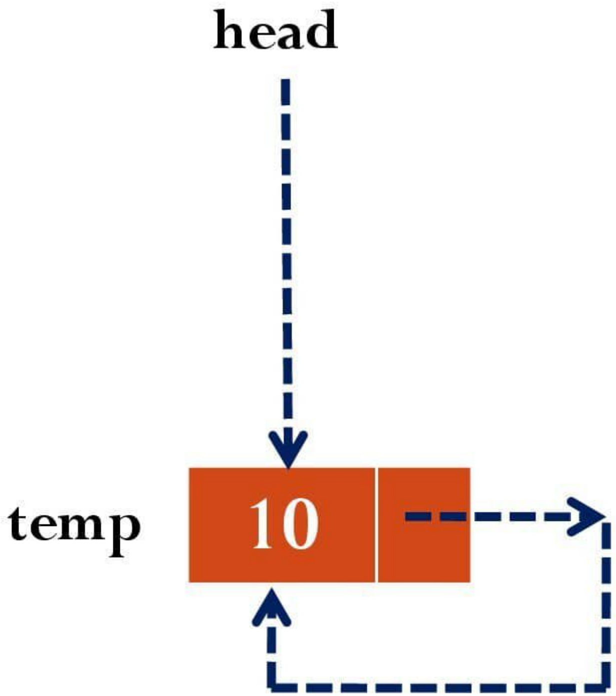
Case 2

Delete from End



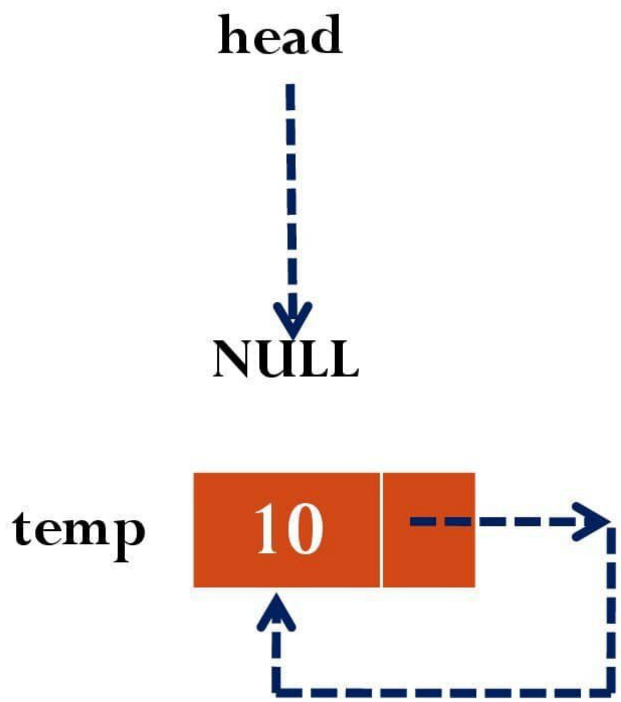
Case 2

Delete from End



Case 2

Delete from End

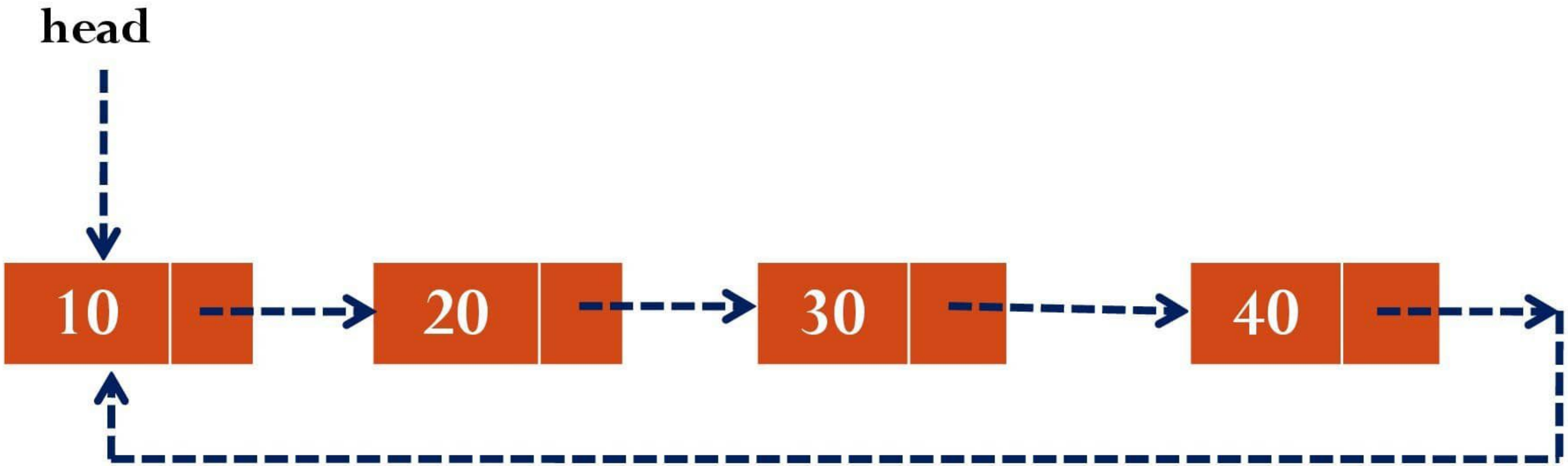


Case 2

Delete from End

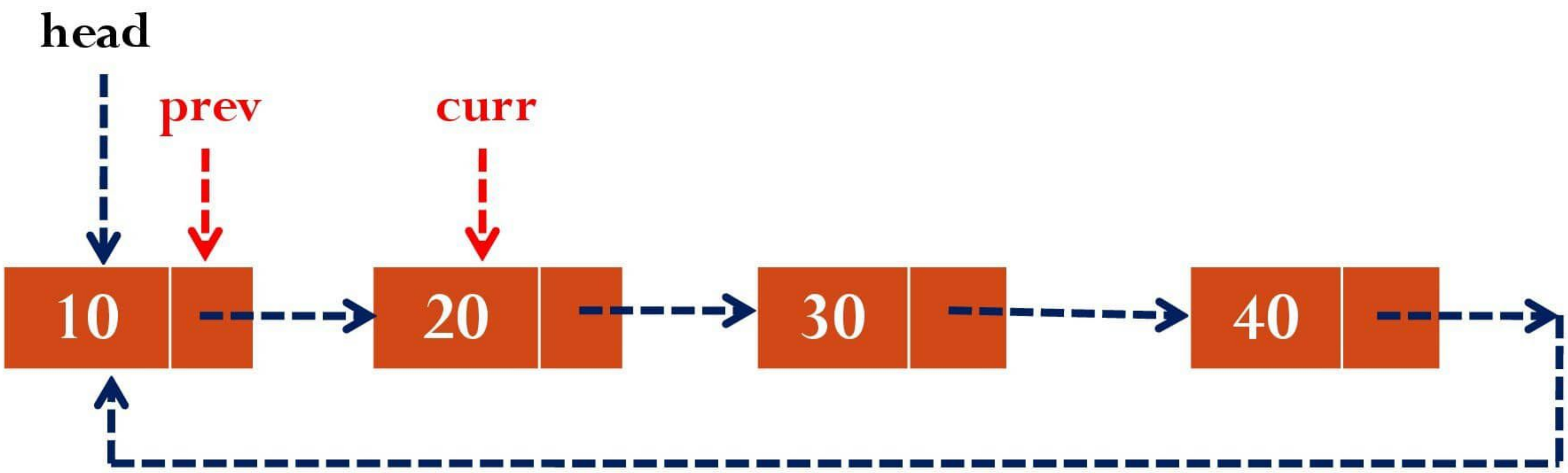
Case 3

Delete from End



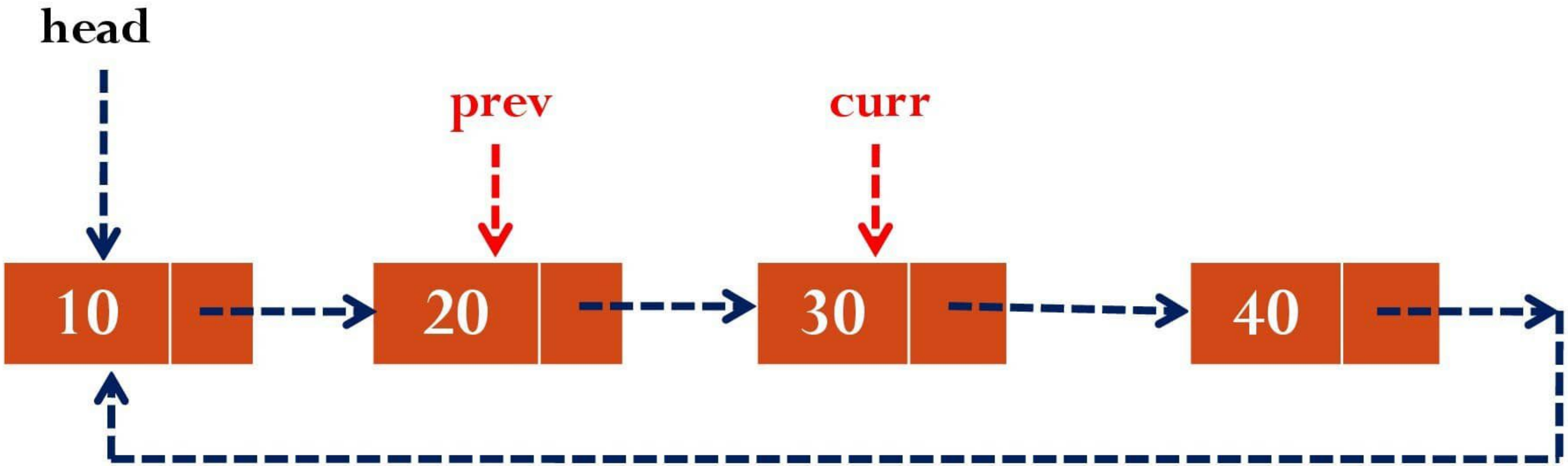
Case 3

Delete from End



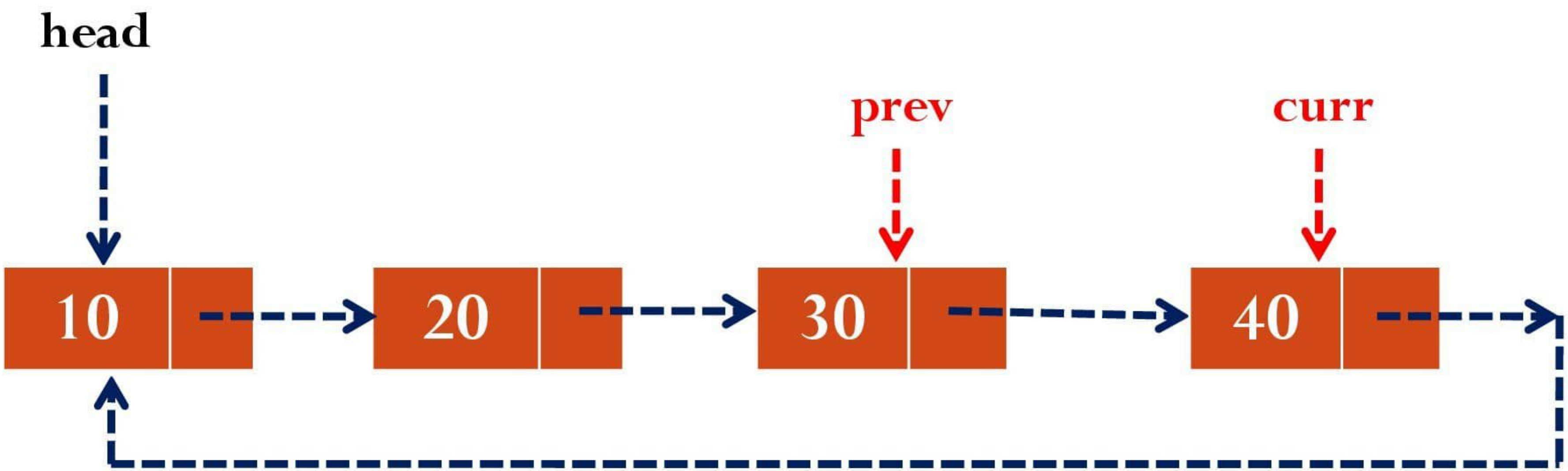
Case 3

Delete from End



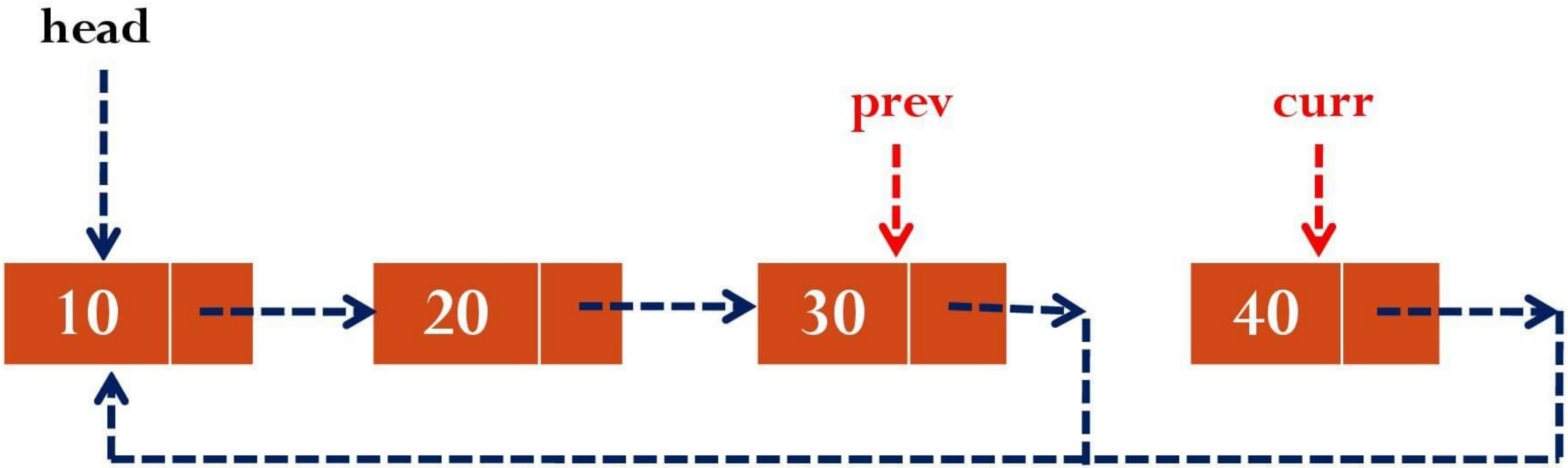
Case 3

Delete from End



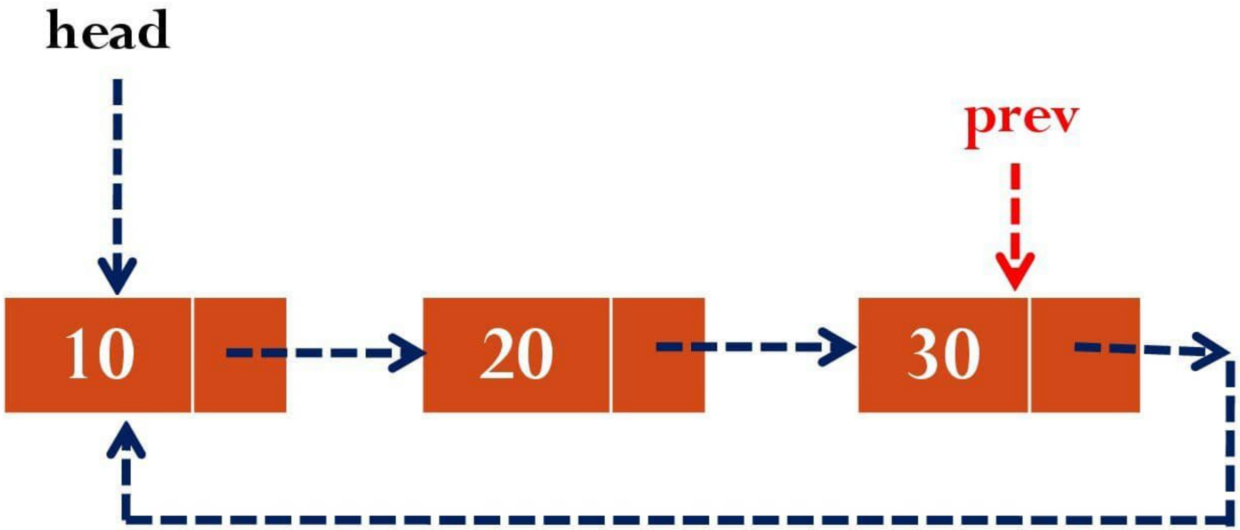
Case 3

Delete from End



Case 3

Delete from End



Delete from End~ Algorithm

Algorithm Delete_End(head)

1. If head=NULL then
   1. Print "List is Empty. Deletion is not possible"
2. Else if head→link=head then
   1. temp=head
   2. head=NULL
   3. dispose(temp)
3. Else
   1. prev=head
   2. curr=head→link
   3. While curr→link!=head do
      1. prev=curr
      2. curr=curr→link
   4. prev→link=head
   5. dispose(curr)

Deletion

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

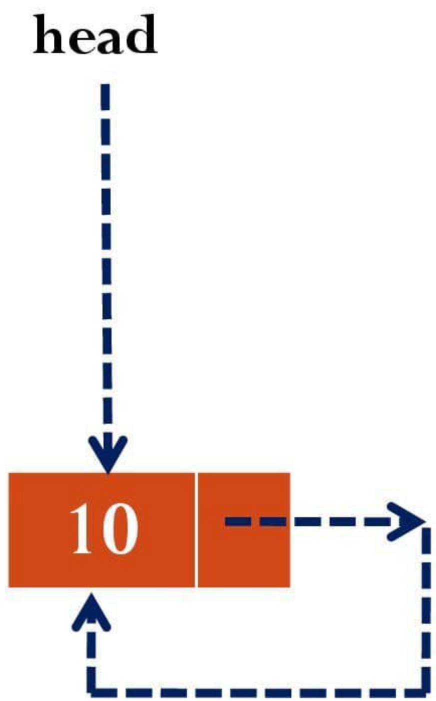
Delete a specified node

2 cases:

1. List is empty
2. List contains only one node
3. Search data is in the first node of the list
4. All Other cases

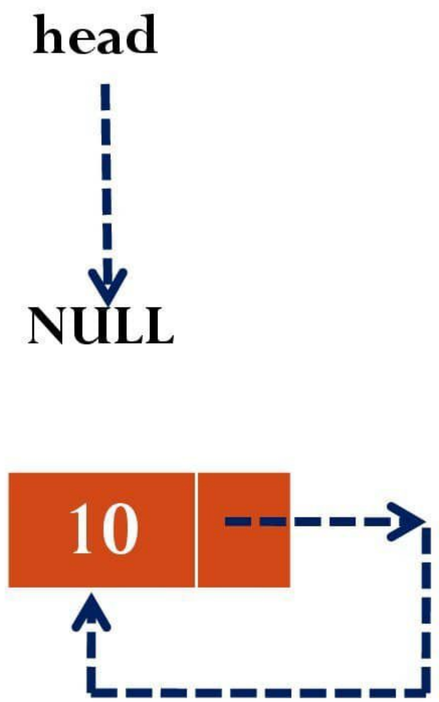
Case 2

Delete 10



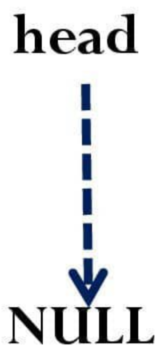
Case 2

Delete 10



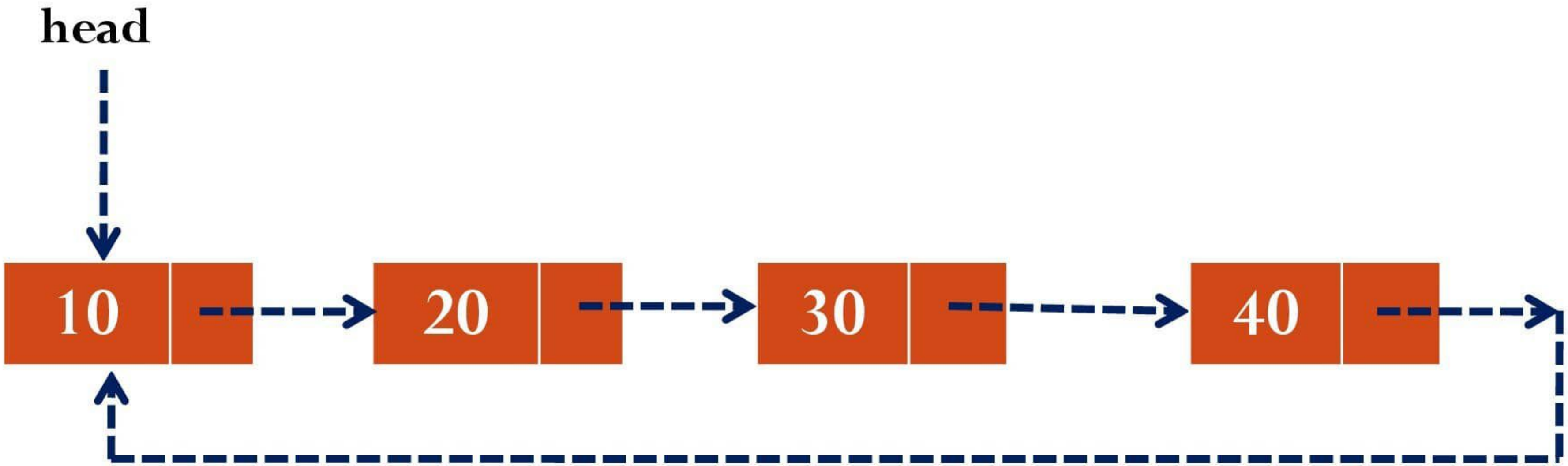
Case 2

Delete 10



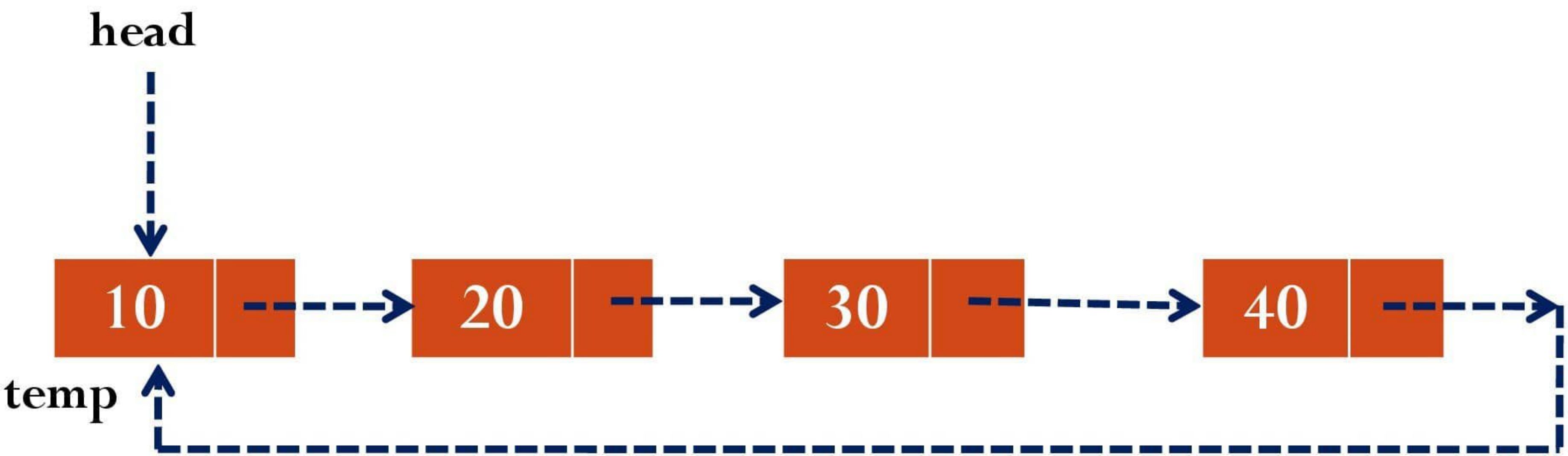
Case 3

Delete 10



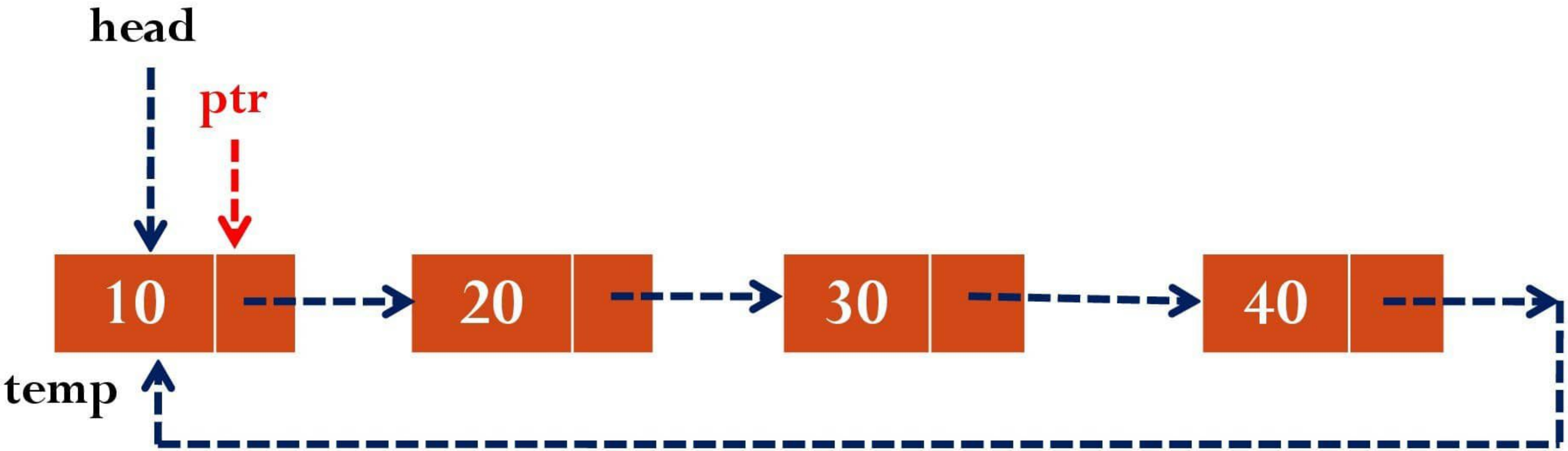
Case 3

Delete 10



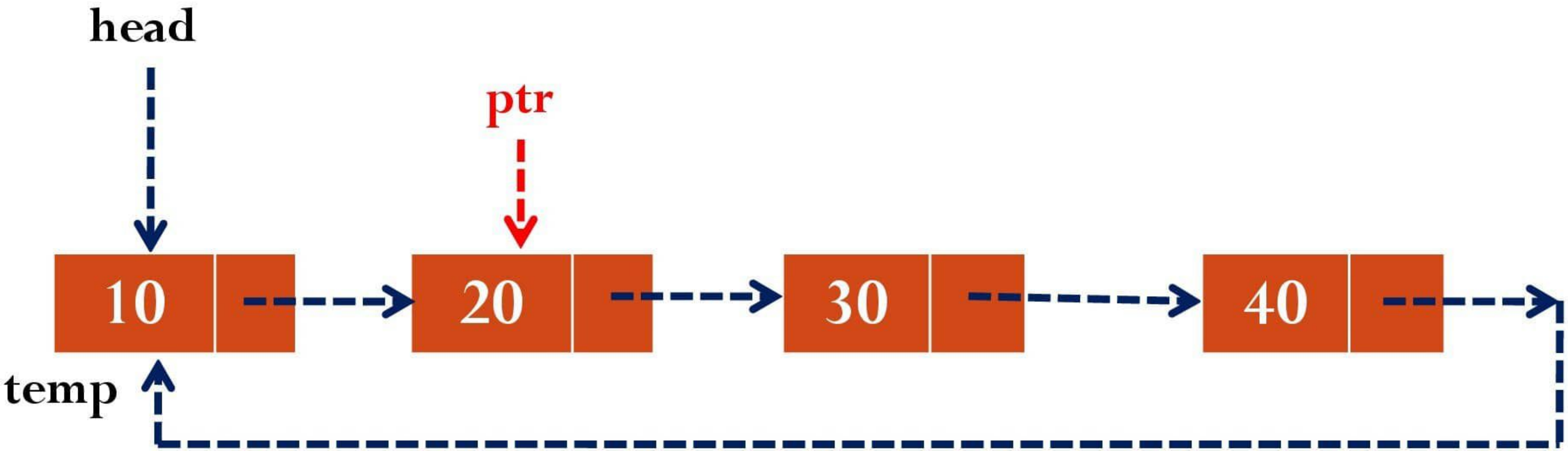
Case 3

Delete 10



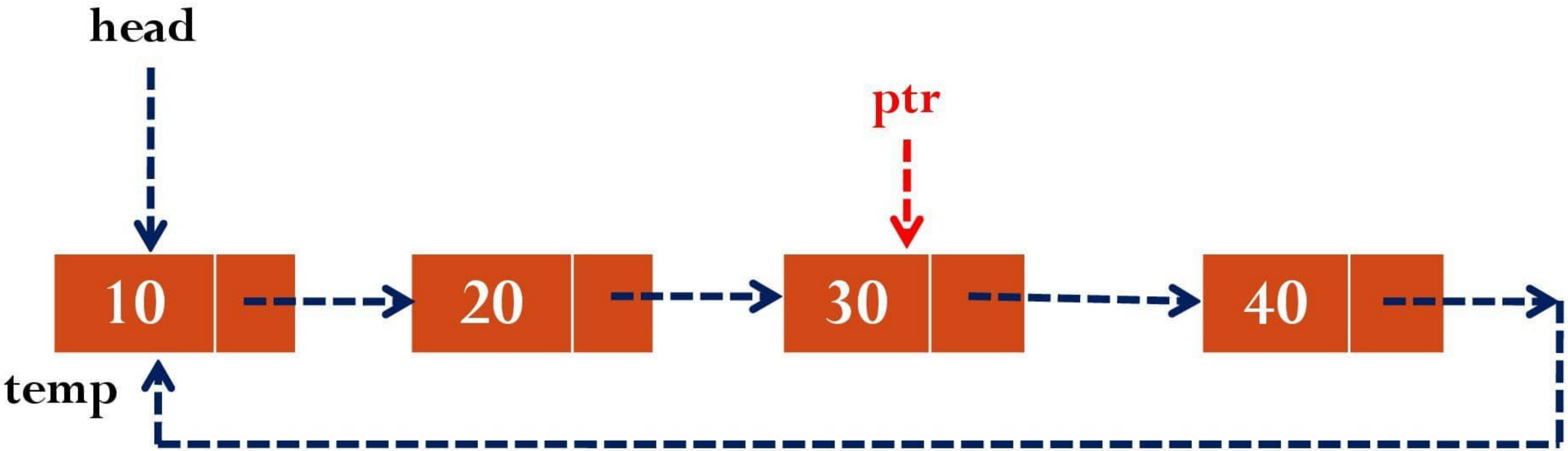
Case 3

Delete 10



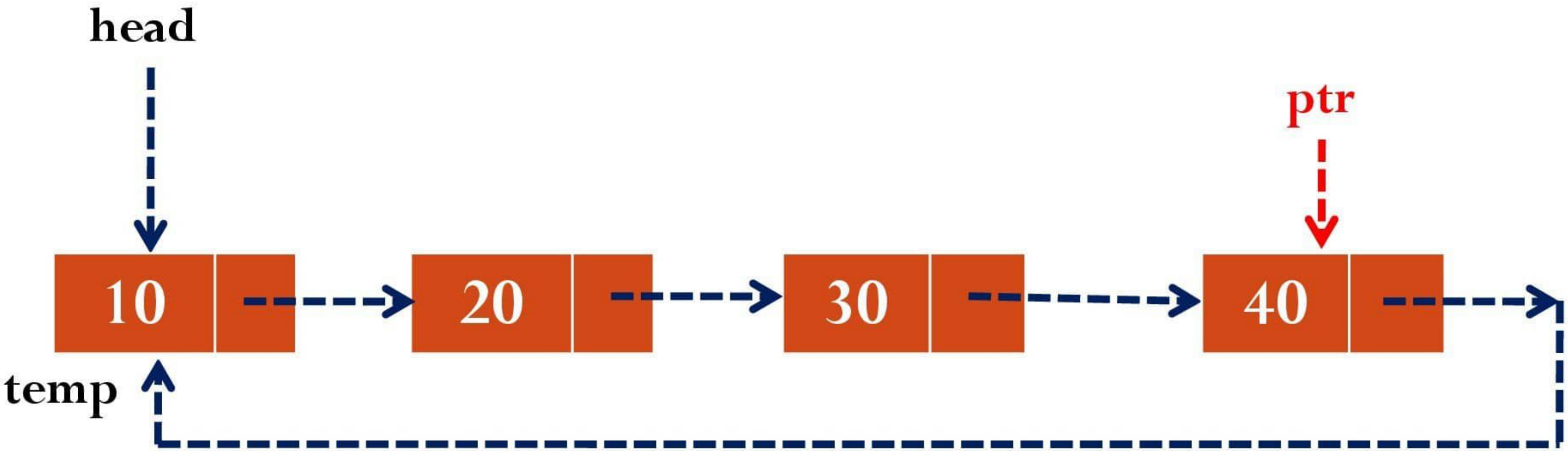
Case 3

Delete 10



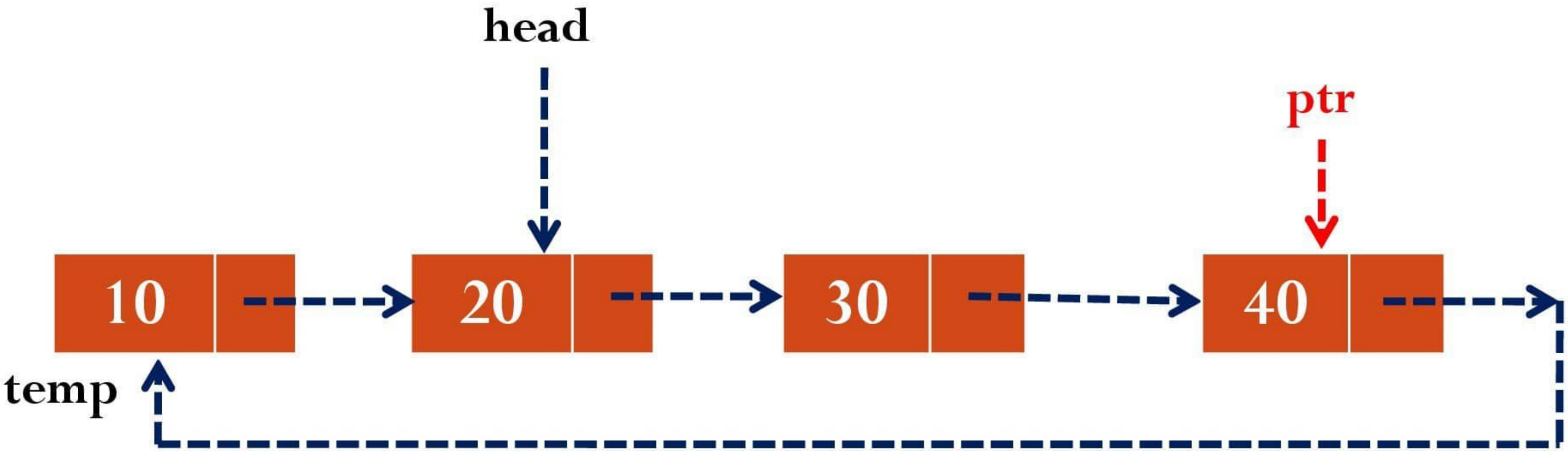
Case 3

Delete 10



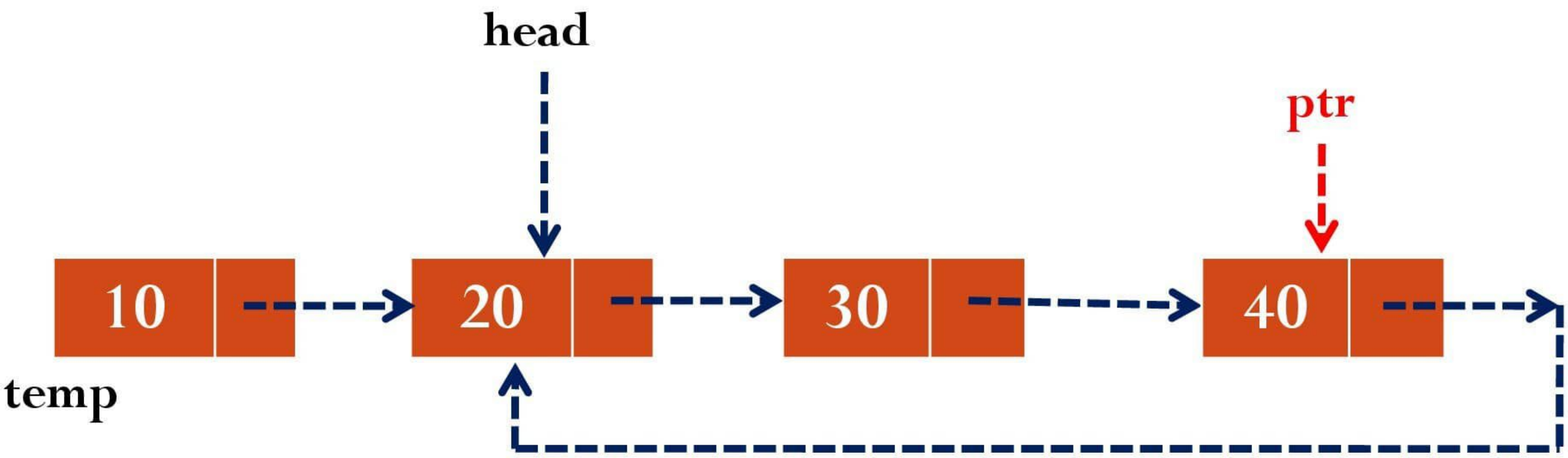
Case 3

Delete 10



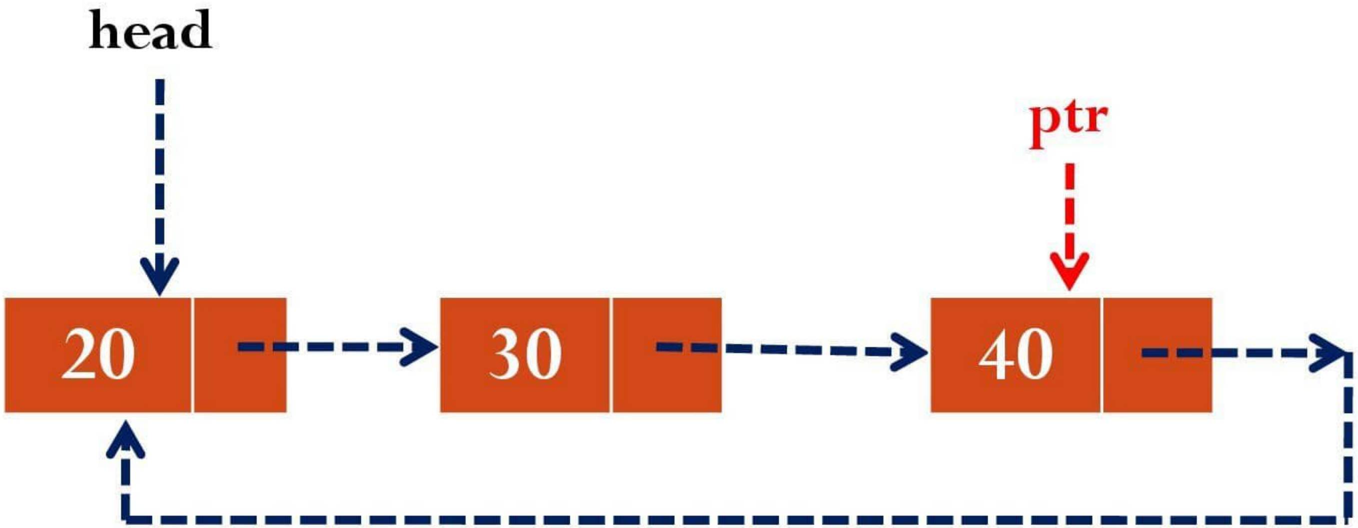
Case 3

Delete 10



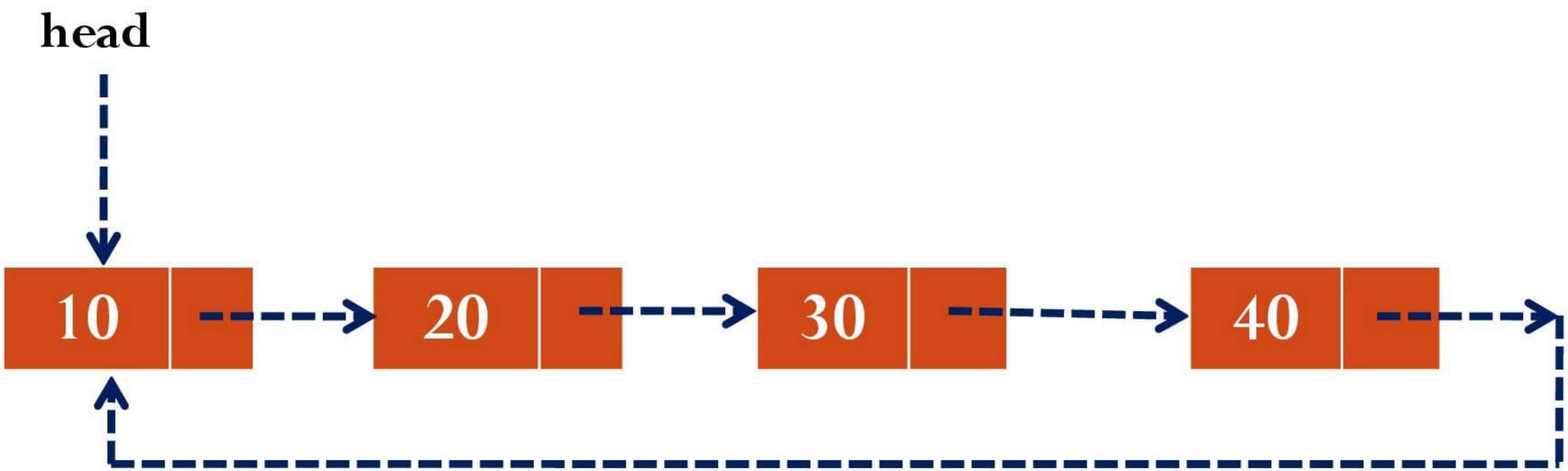
Case 3

Delete 10



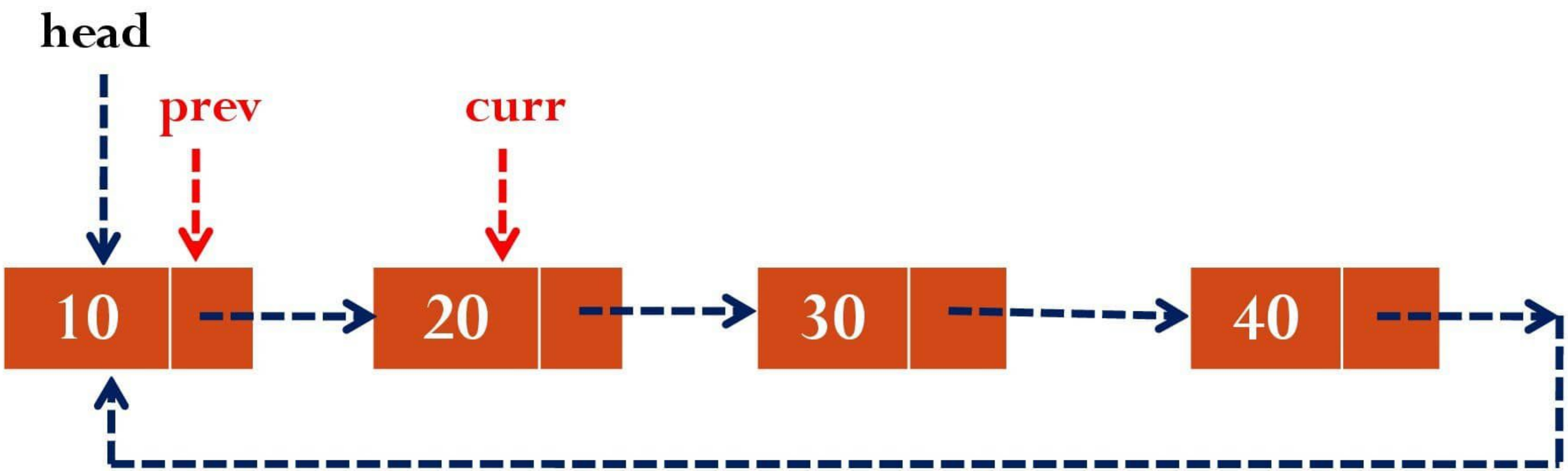
Case 4

Delete 30



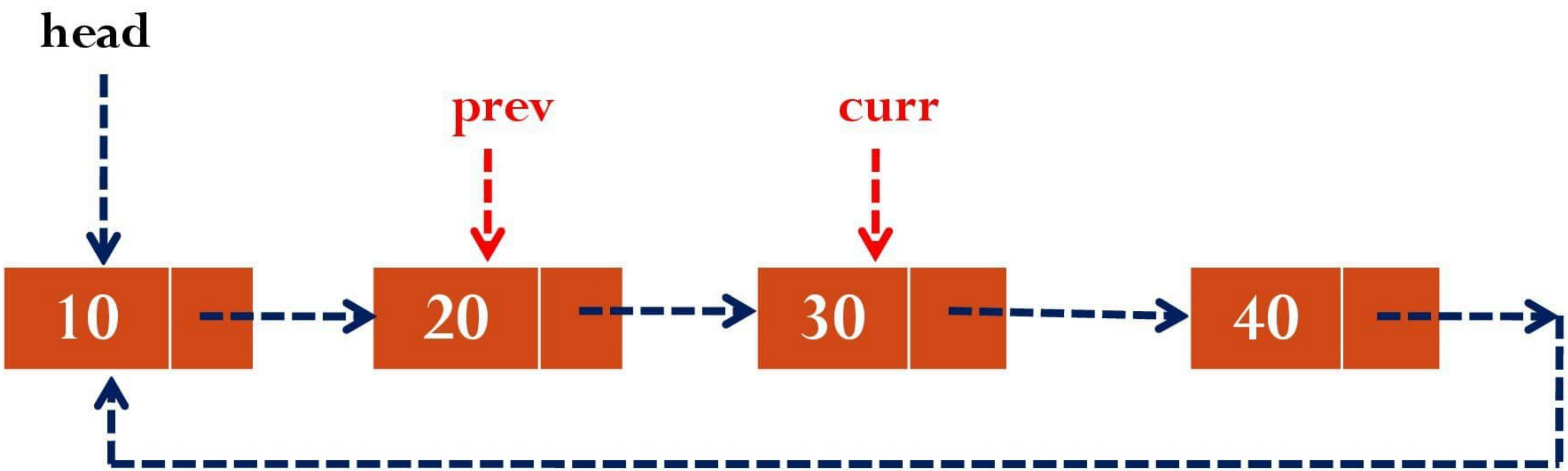
Case 4

Delete 30



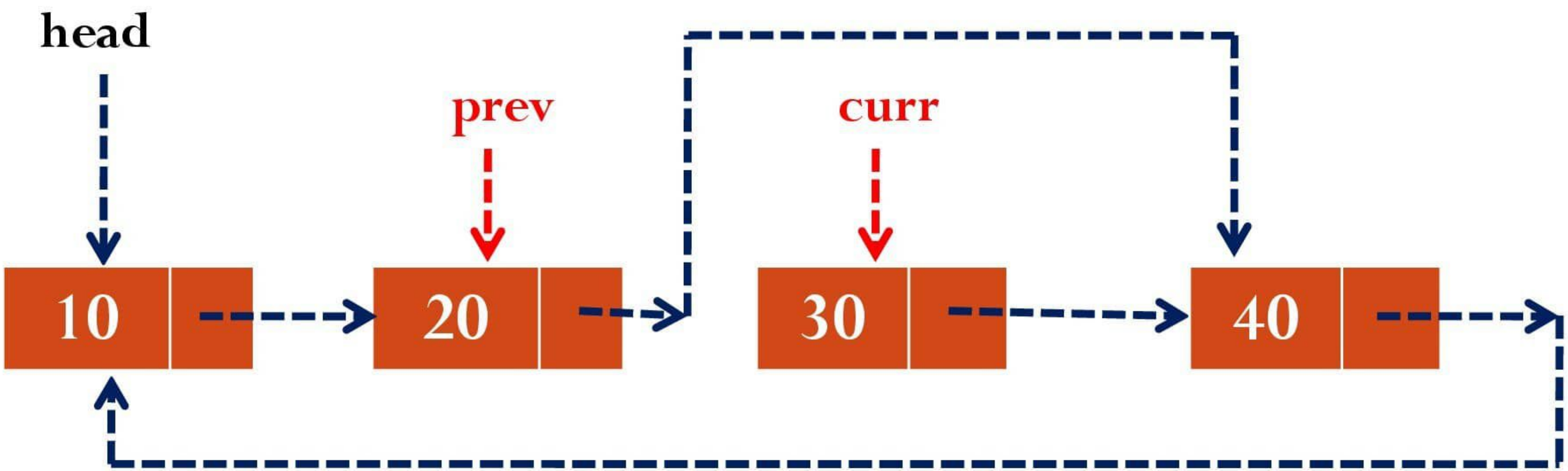
Case 4

Delete 30



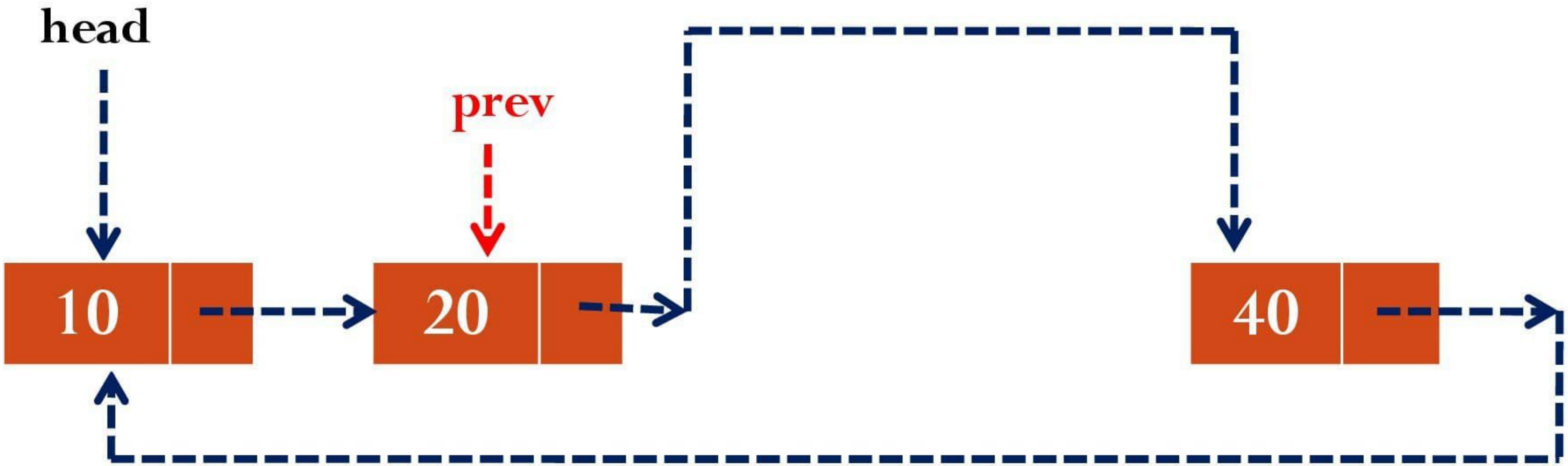
Case 4

Delete 30



Case 4

Delete 30



Delete a specified node~ Algorithm

Algorithm Delete_Any(head,key)

1. If head=NULL then
   1. Print "List is Empty. Deletion is not possible"
2. Else If head→link=head then
   1. If head→data=key then head=NULL
   2. Else Print "Search data not found."
3. Else if head→data=key then
   1. temp=ptr=head
   2. While ptr→link!=head do
      1. ptr=ptr→link
   3. head=head→link
   4. ptr→link=head
   5. dispose(temp)

Delete a specified node~ Algorithm

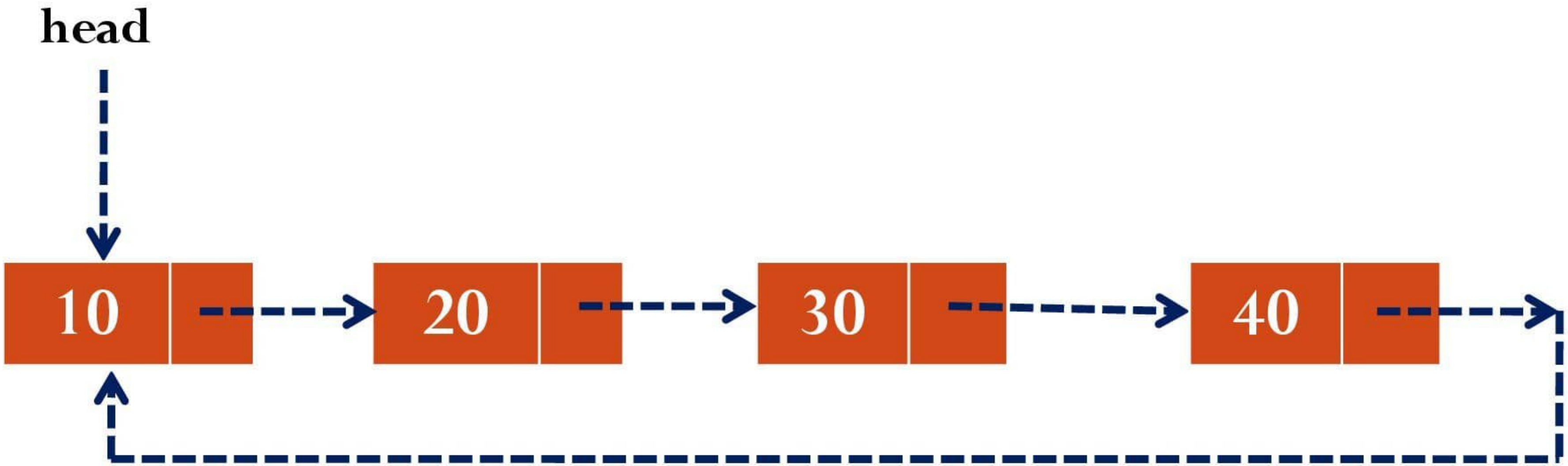
4. Else
    1. prev=head
    2. curr=head→link
    3. While curr→link!=head and curr→data!=key do
        1. prev=curr
        2. curr=curr→link
    4. if curr→data=key then
        1. prev→link=curr→link
        2. dispose(curr)
    5. Else
        1. Print "Search data not found"

Search

1. List is empty
2. List is not empty

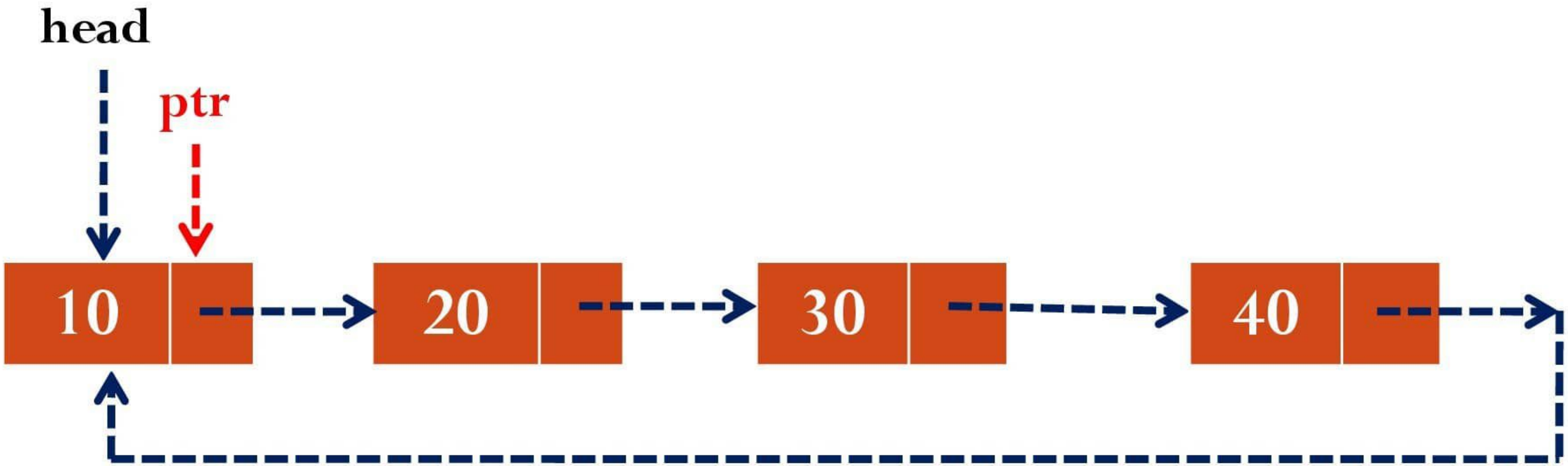
Case 2

Search 30



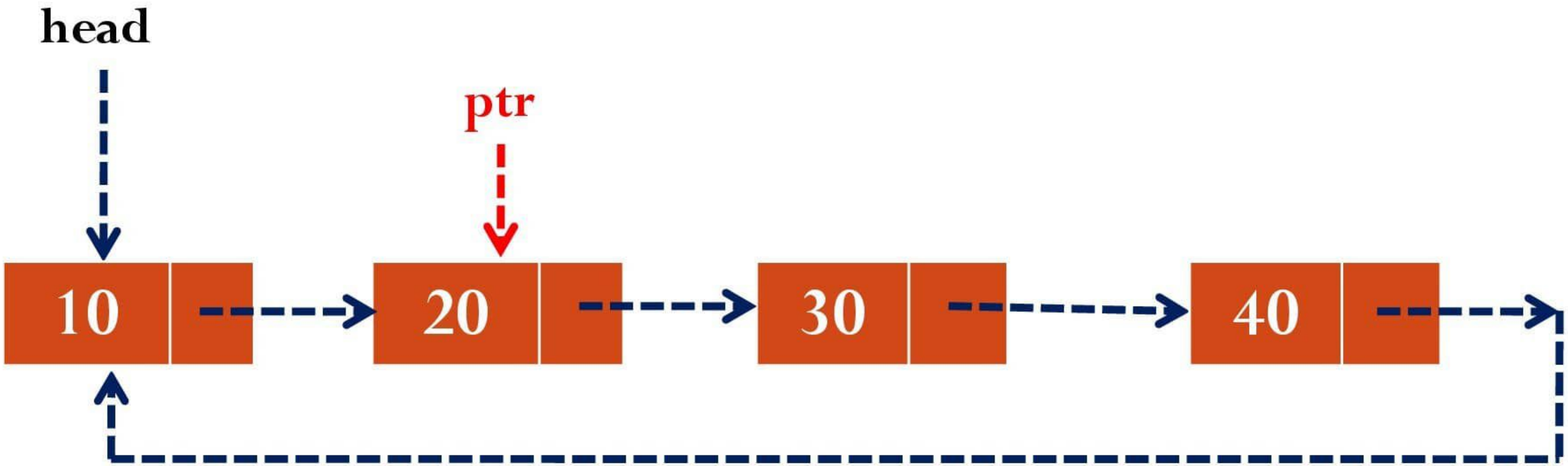
Case 2

Search 30



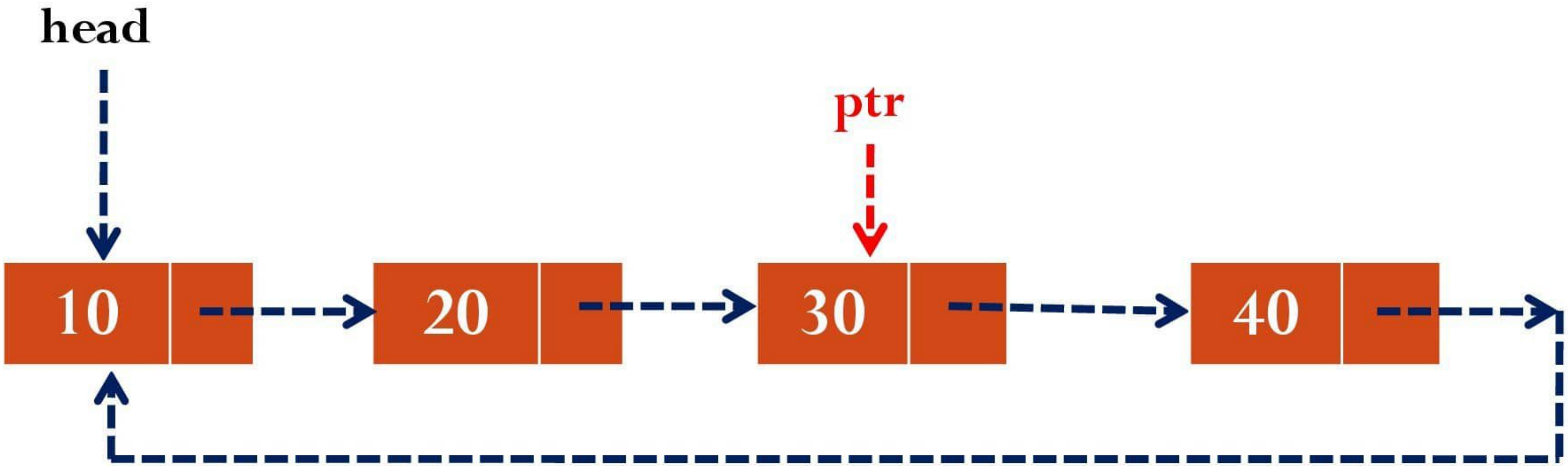
Case 2

Search 30



Case 2

Search 30

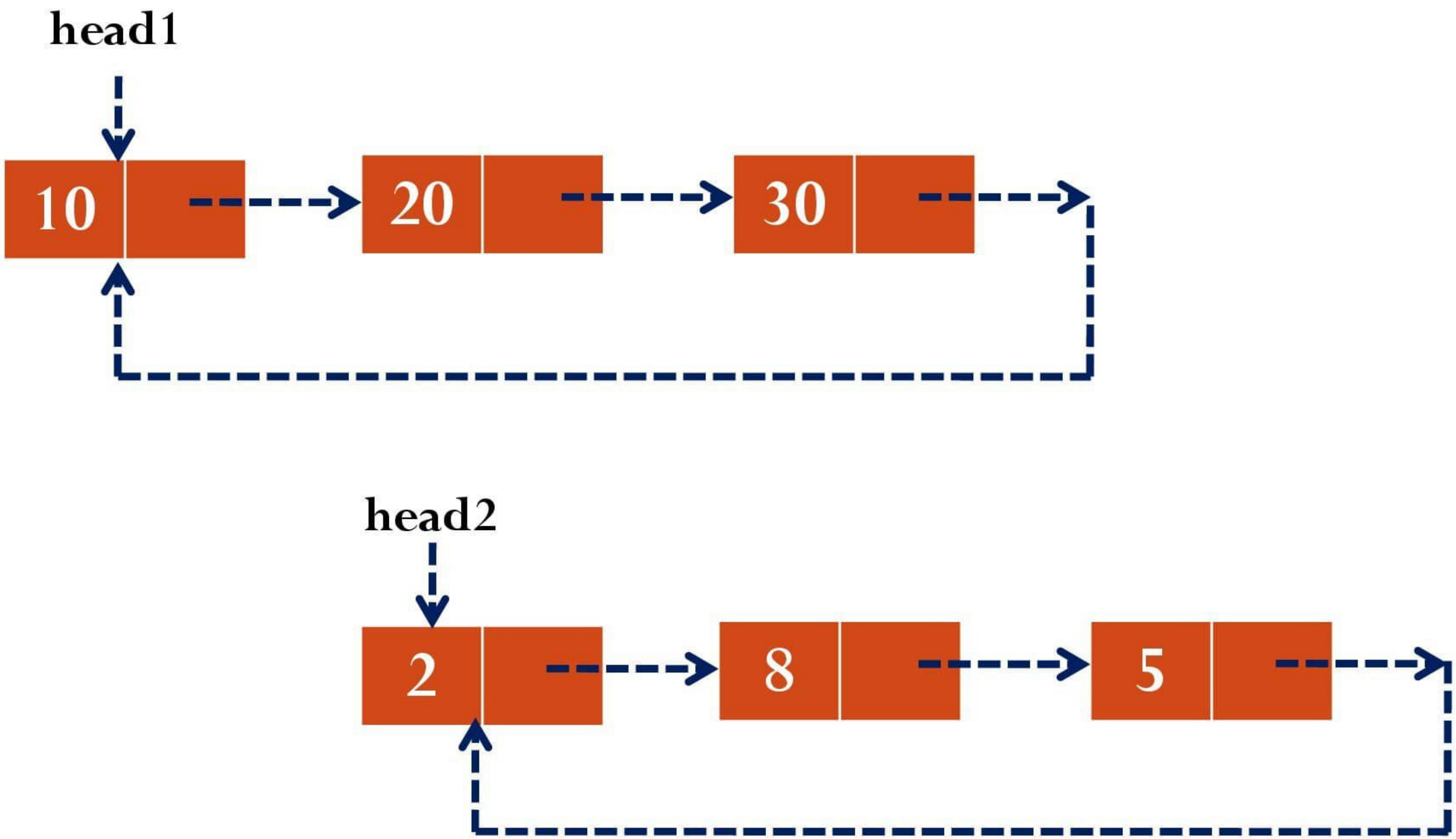


Search ~ Algorithm

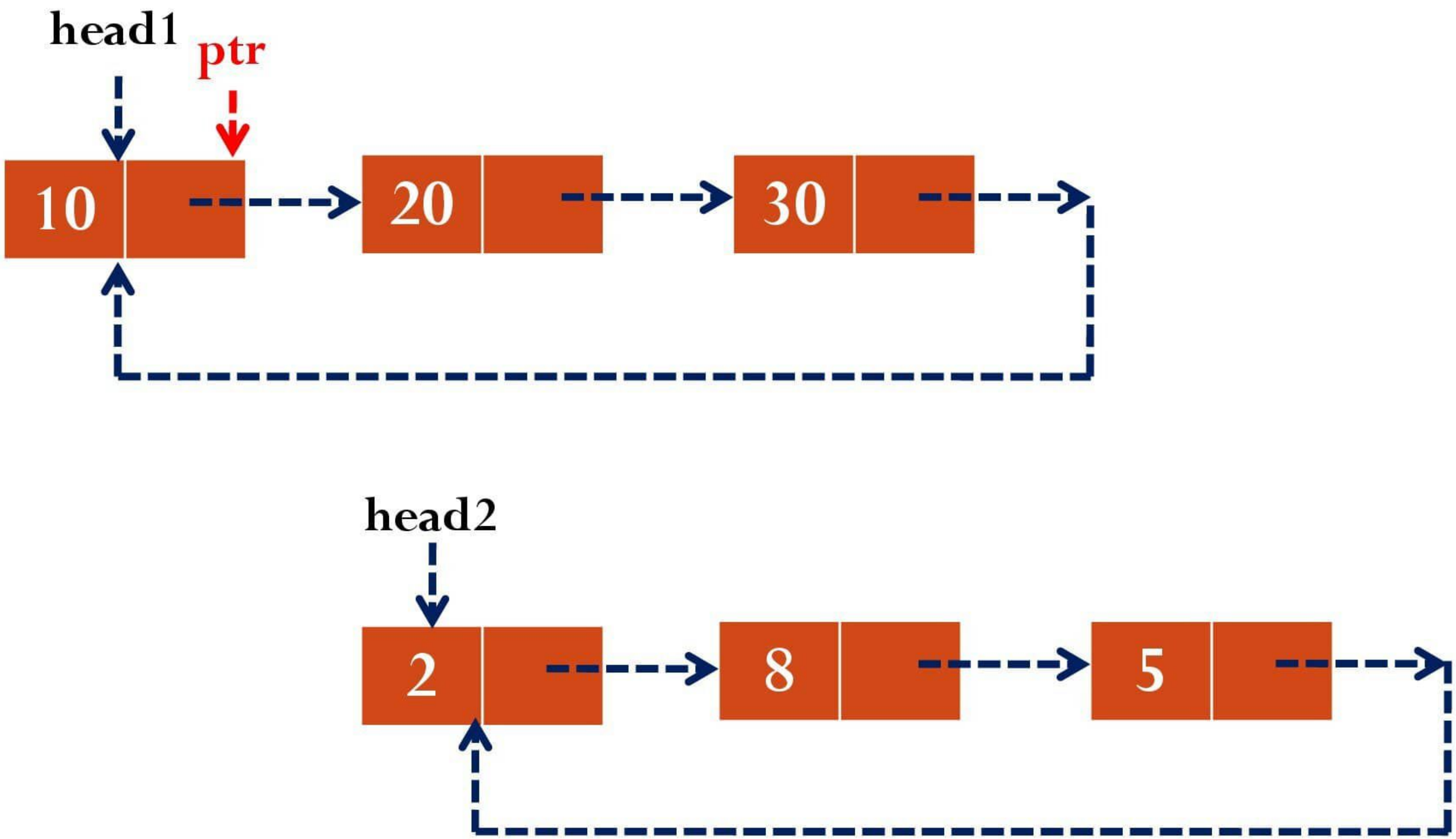
Algorithm Search(head, key)

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

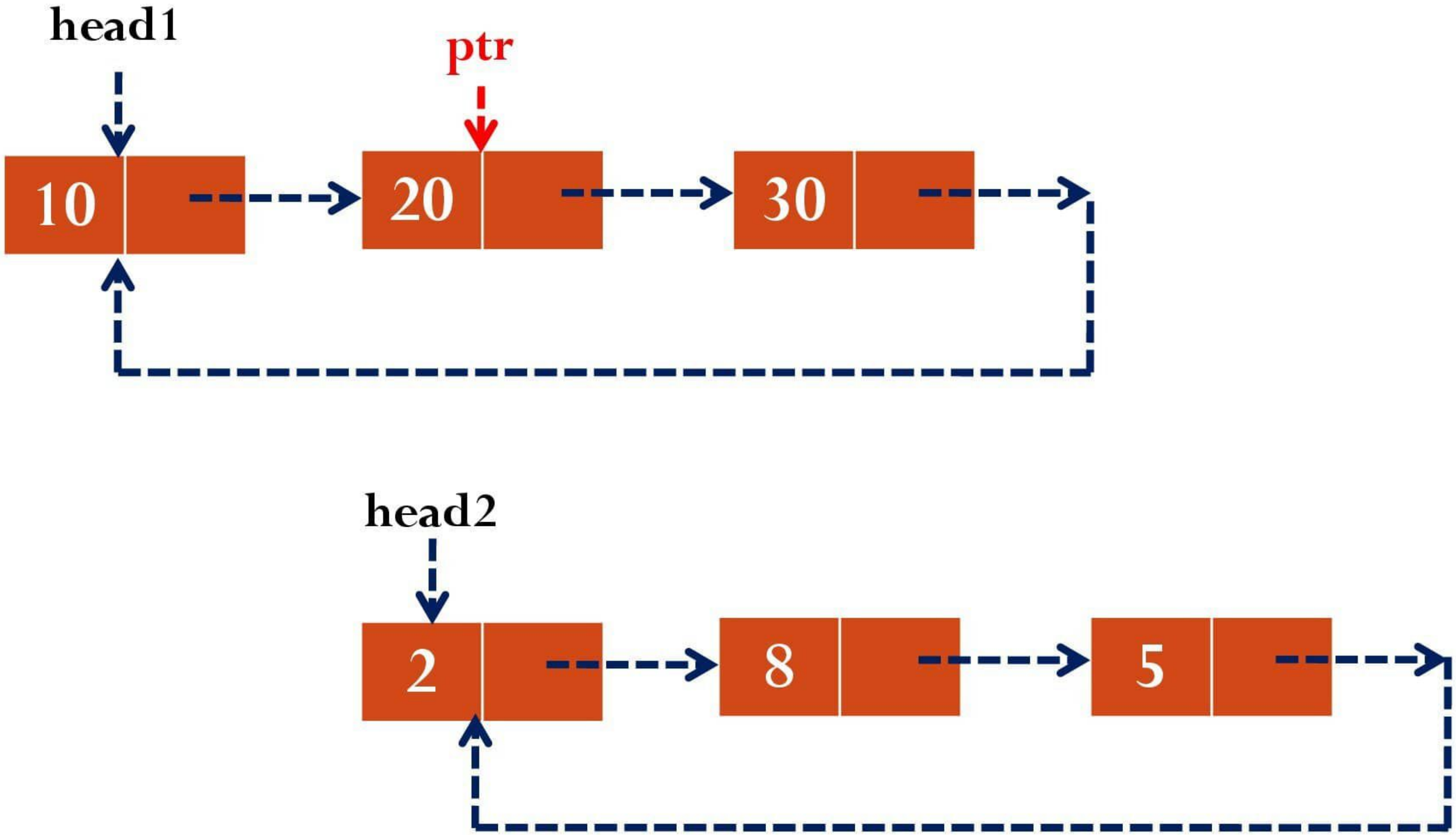
Merge



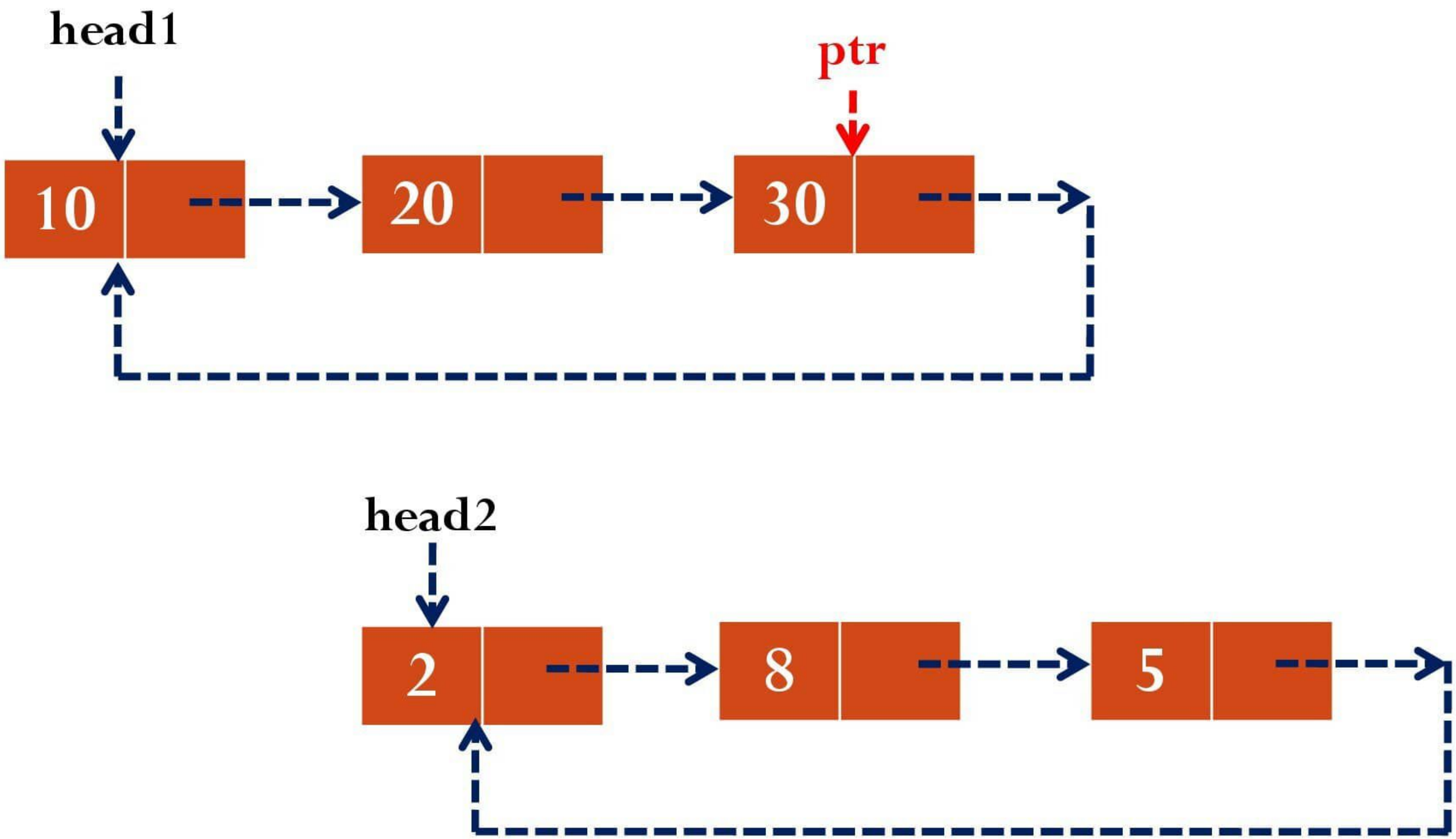
Merge



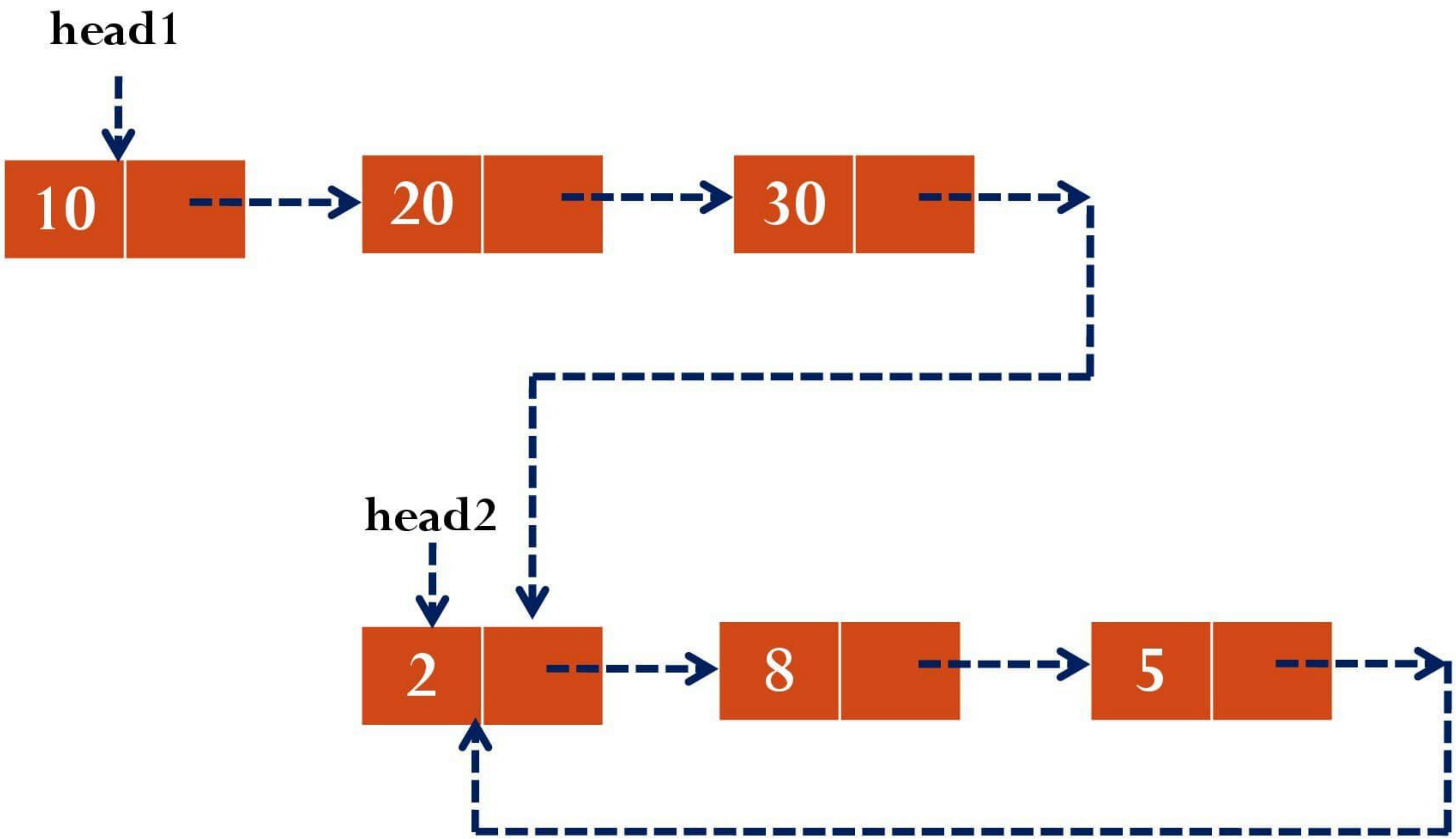
Merge



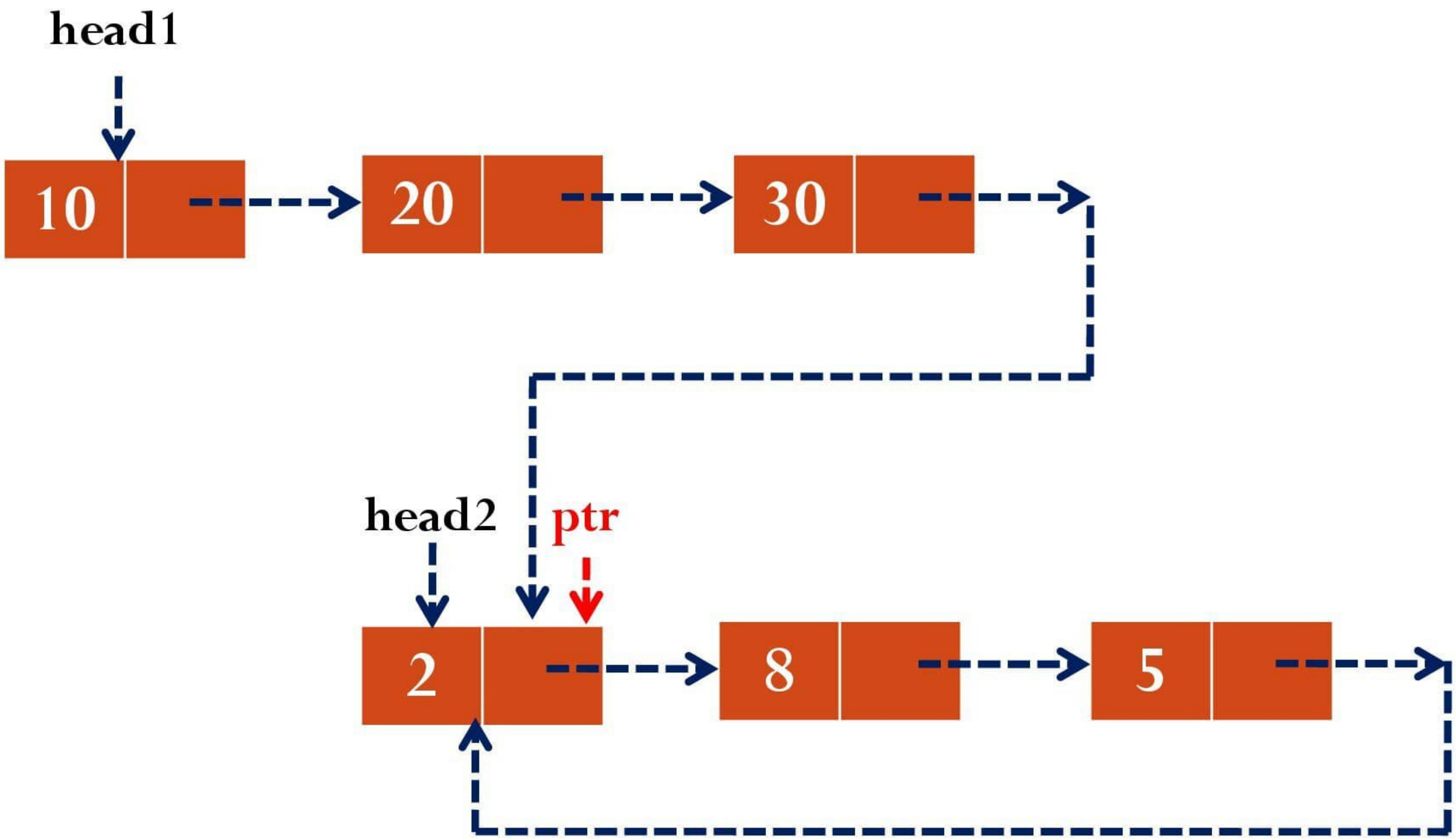
Merge



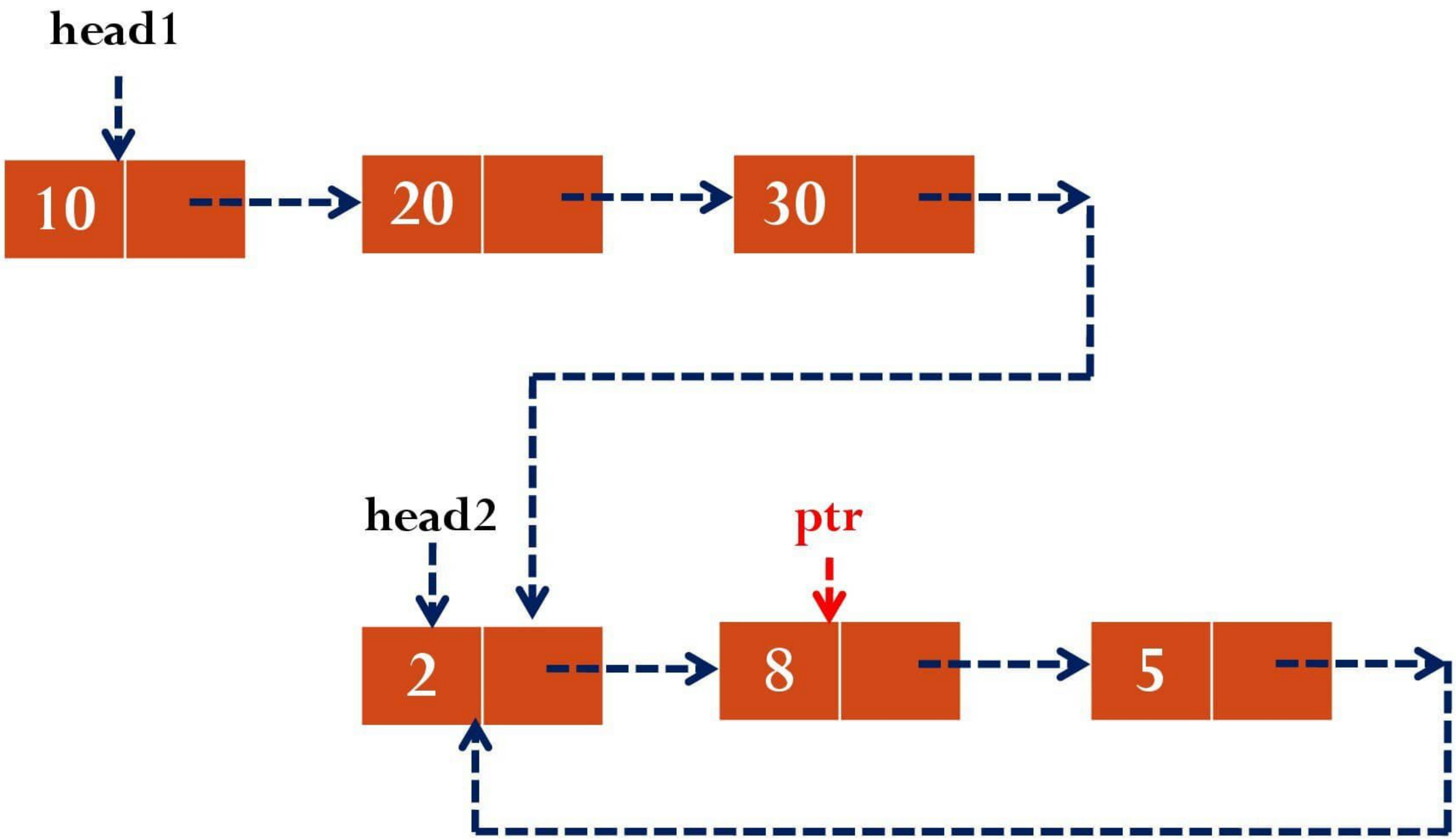
Merge



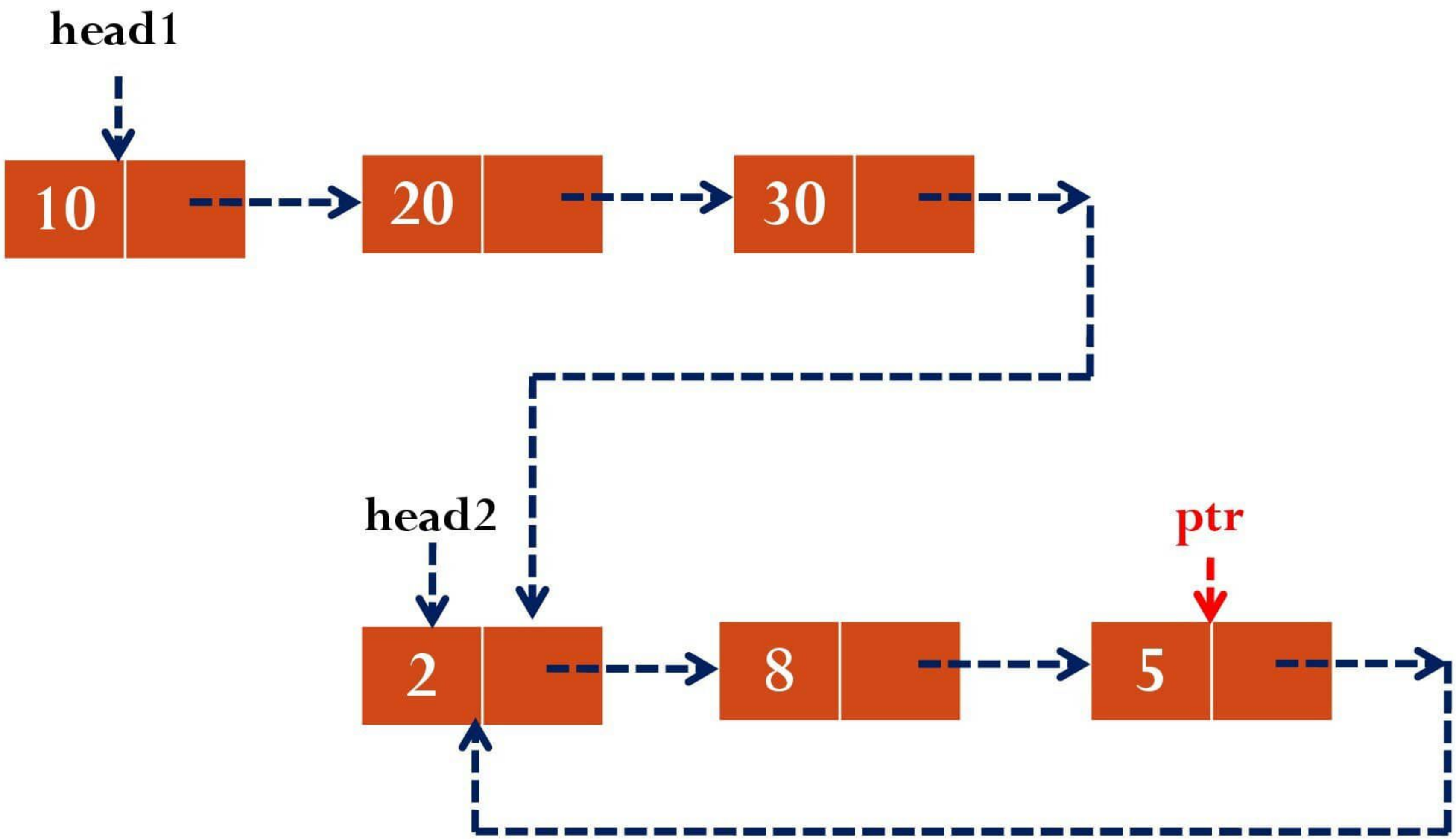
Merge



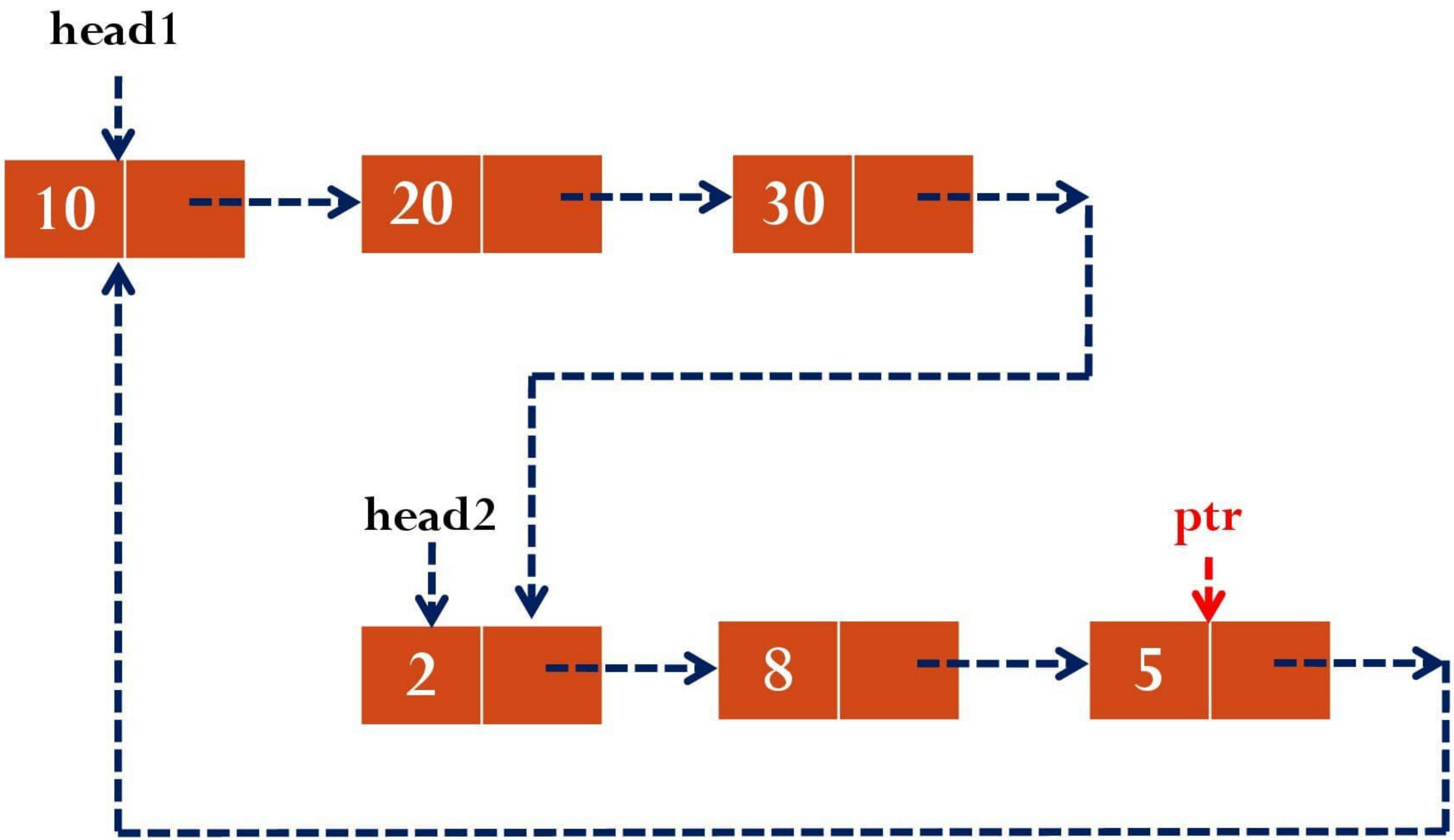
Merge



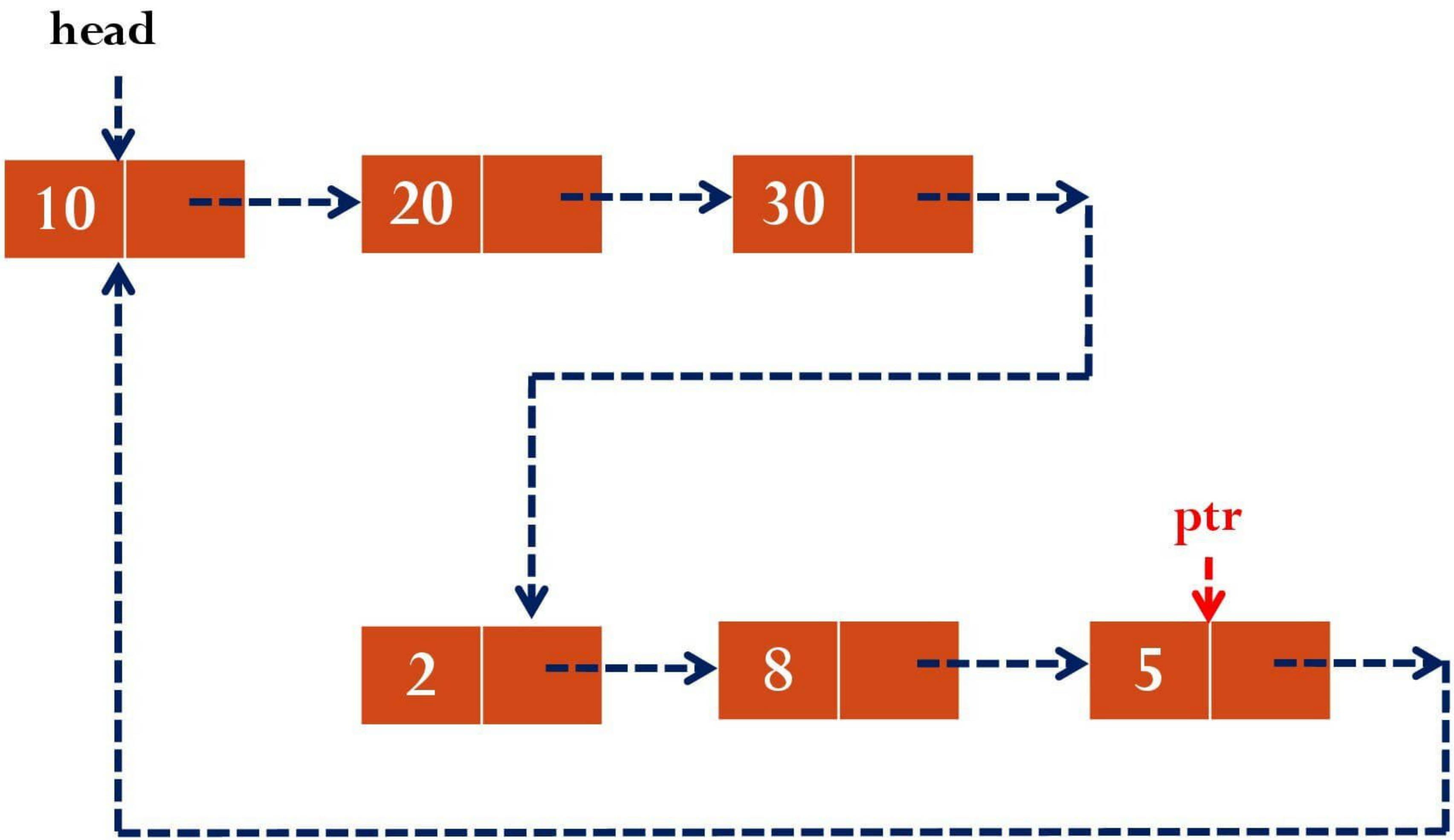
Merge



Merge



Merge



Merge ~ Algorithm

Algorithm Merge(head1, head2)

1. ptr = head1
2. while ptr→link !=head1 then
    1. ptr = ptr→link
3. ptr →link = head2
4. ptr = head2
5. while ptr→link !=head2 then
    1. ptr = ptr→link
6. ptr→link=head1
7. head=head1