Linked List

Types of Linked List

There are three types of linked list as given below:

1. Singly (Linear) Linked List
2. Circular Linked List
3. Doubly Linked List

Circular Linked Lists

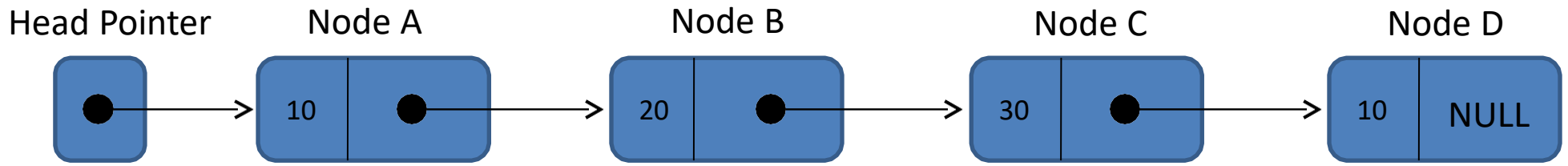
- In linear linked lists if a list is traversed (all the elements visited) an external pointer to the list must be preserved in order to be able to reference the list again.

- Circular linked list is a list in which each node has a successor; the “last” element is succeeded by the “first” element

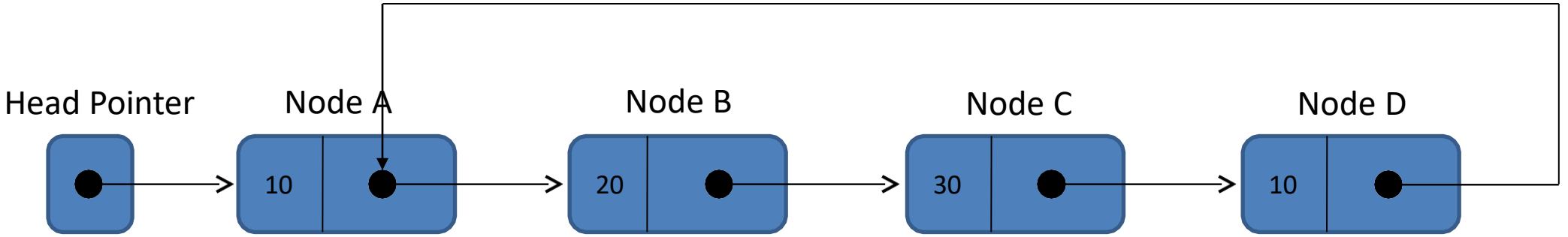
Circular linked list - Motivation

- Round Robin Scheduling
- Repeat the songs in a playlist

Circular Linked Lists

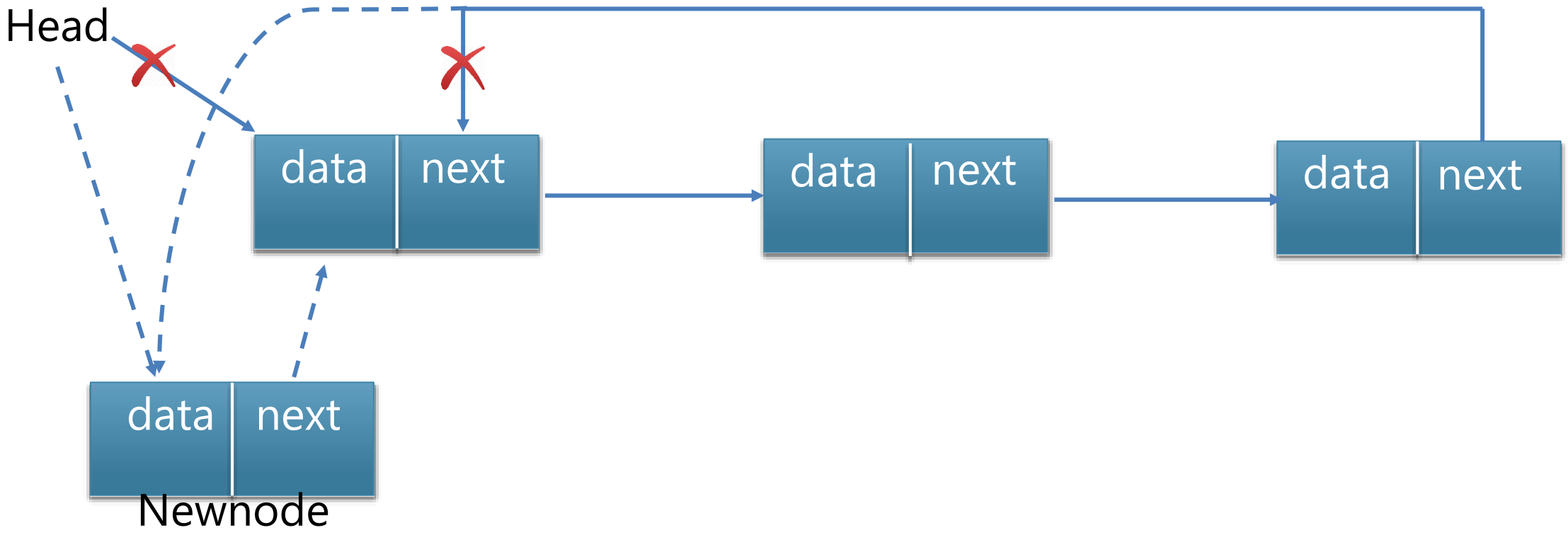


A graphical view of a **linear** linked list



A graphical view of a **circular** linked list

Circular Linked List – insert at first

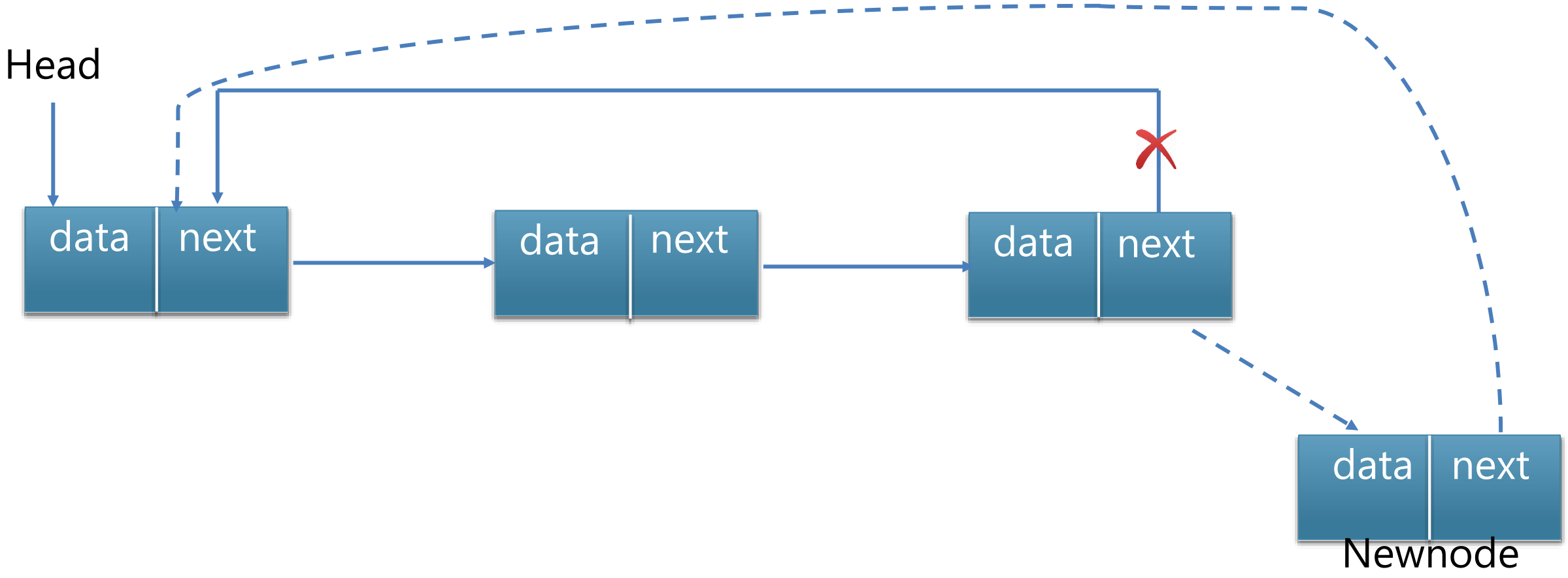


//Inserting at first

```
insertFirst(data):
    Begin
      create a new node
      newnode -> data = data
      newnode -> next = null

      if the list is empty, then
          head = newnode
      else
          curr = head
          forever:
              if curr -> next == head
                  break
              curr= curr->next
          newnode -> next = head
          curr - > next = newnode
          head = newnode
    End
```

Circular Linked List – insert at last (append)

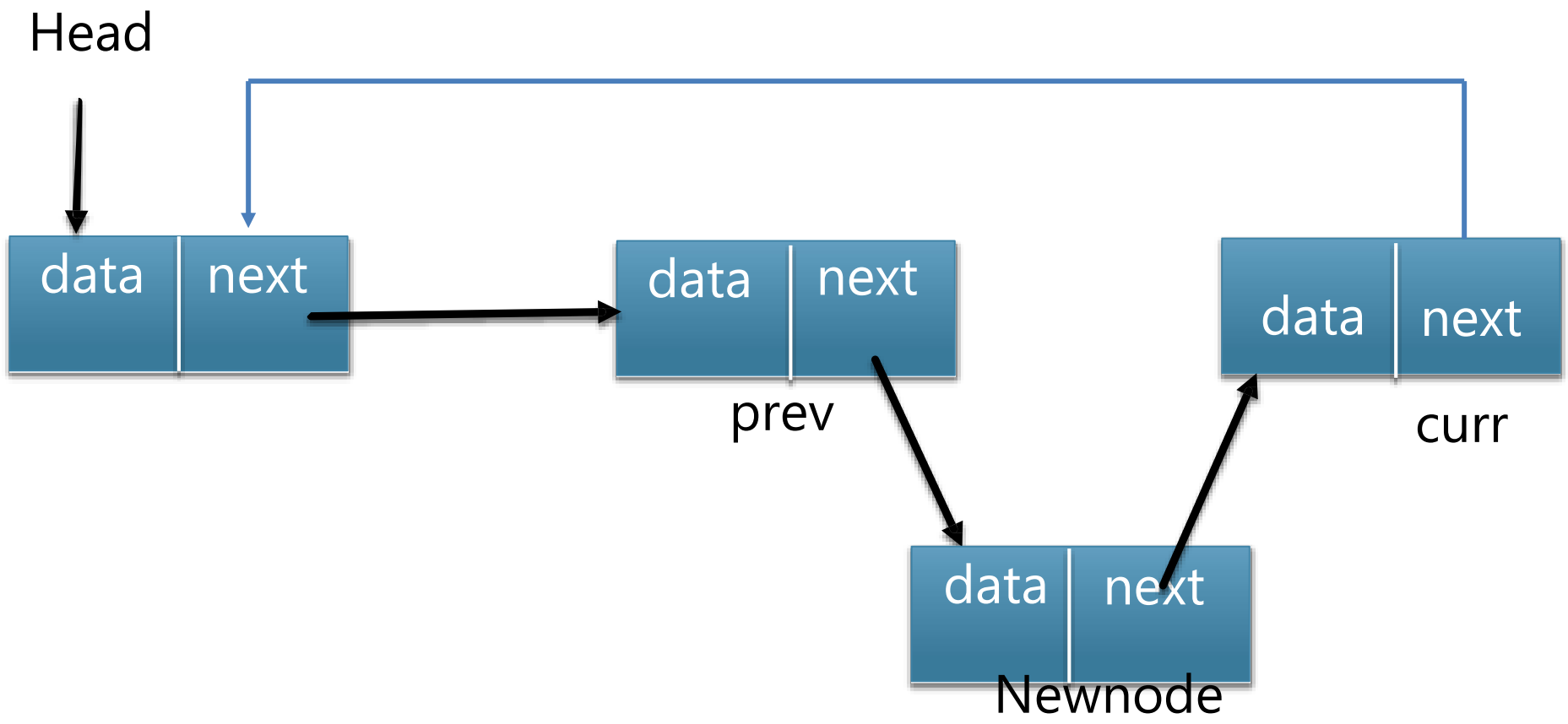


//Inserting at Last

```
insertLast(data):
    Begin
      create a new node
      newnode -> data = data
      newnode -> next = null

      if the list is empty, then
          head = newnode
      else
        curr = head
        forever:
            if curr -> next == head
                break
            curr= curr->Next
        newnode -> next = head
        curr - > next = newnode
    End
```

Circular Linked List – insert at position

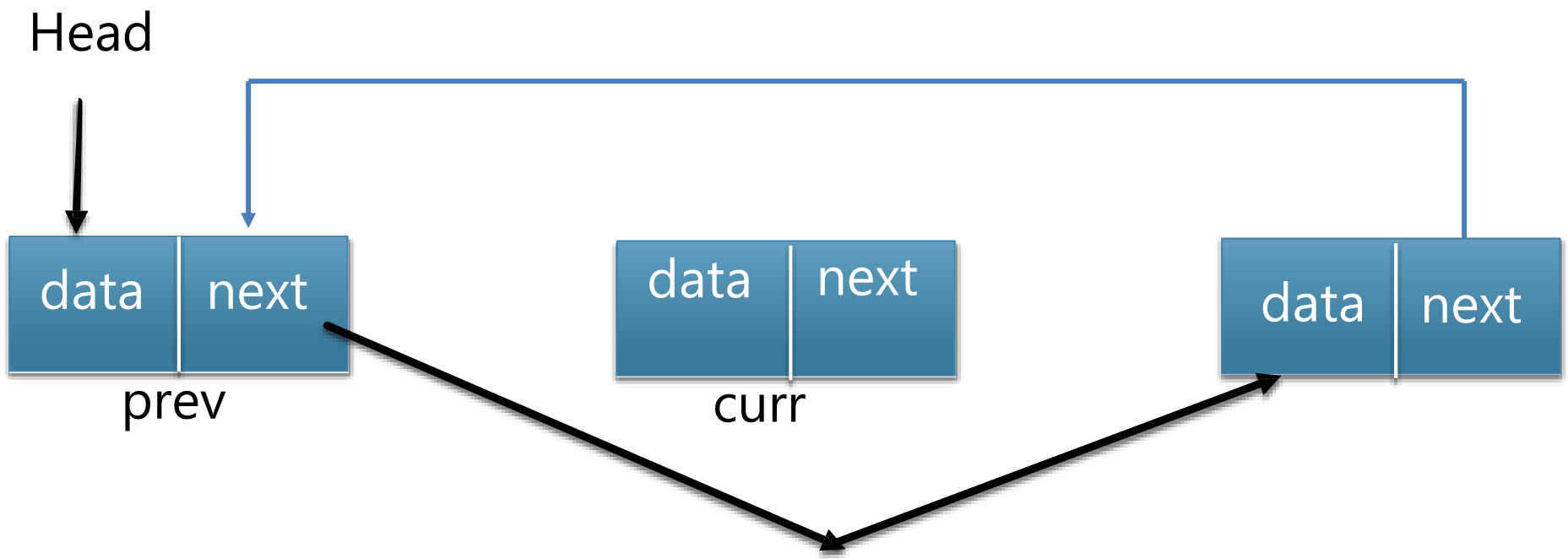


//Inserting at position

```
insertAtPos(data, pos):
    Begin
        create a new node
        newnode -> data = data
        newnode -> next = null

        curPos = 1
        if the list is empty, then
            head = newnode
        else
            curr = head
            forever:
                if curPos == pos || curr -> next == head
                    break
                prev = curr
                curr= curr->Next
                curPos++
            prev -> next = newnode
            newnode -> next = curr
    End
```

Circular Linked List – delete



//Delete

```
deleteAtPos(pos):
    Begin
        curr = head
        curPos = 1
          if head is null, then
              it is Underflow and return
           else if pos ==1
               forever:
                       if curr -> next == head
                            break
                       curr= curr->Next
              curr->next = head -> next
              temp = head
              head = curr ->next
              delete temp
           else
                forever:
                     if curPos == pos || curr -> next == head
                          break
                     prev = curr
                     curr= curr->Next
                     curPos++
               prev->Next = curr->Next
               Delete curr
    End
```

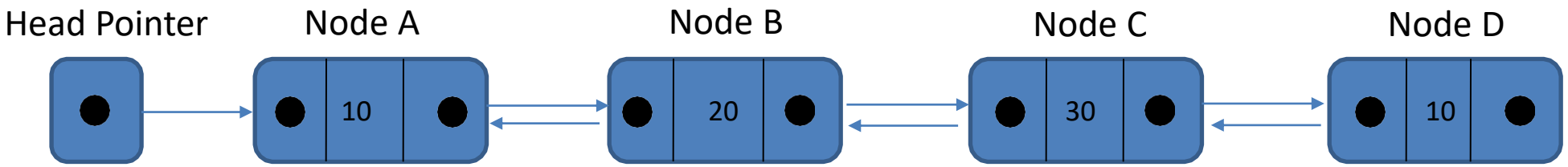
Types of Linked List

There are three types of linked list as given below:

1. Singly (Linear) Linked List
2. Circular Linked List
3. Doubly Linked List

Doubly Linked Lists

- A doubly link list is a list in which each node is linked to both its successor and its predecessor .



A graphical view of a **doubly** linked list

Doubly linked list - Motivation

- Doubly linked lists are useful for playing video and sound files with “rewind” and “instant replay”.
- They are also useful for other linked data which require “rewind” and “fast forward” of the data

Doubly Linked Lists

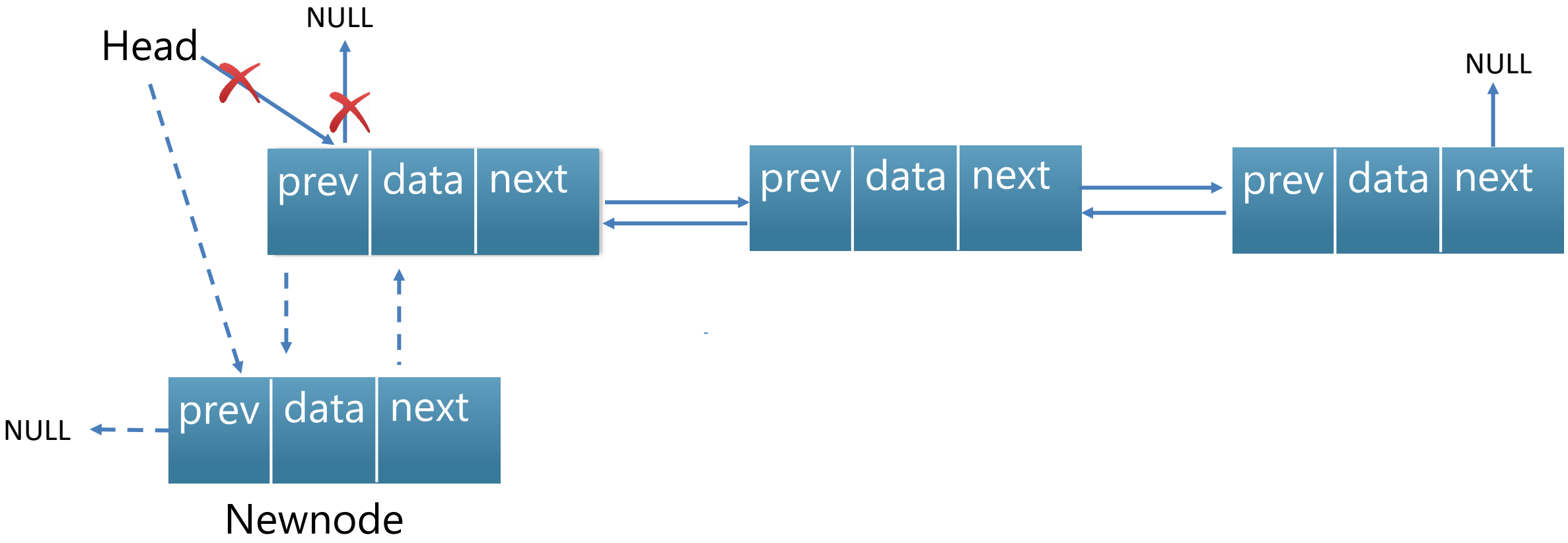
- In a Doubly Linked List each item points to both its predecessor and successor
  - next points to the successor
  - prev points to the predecessor

Doubly Linked List – Node Definition

```
struct Node{
        int data;
        struct Node* next;
        struct Node* prev;
};

struct Node* head = NULL;
```

Doubly Linked List – insert at first



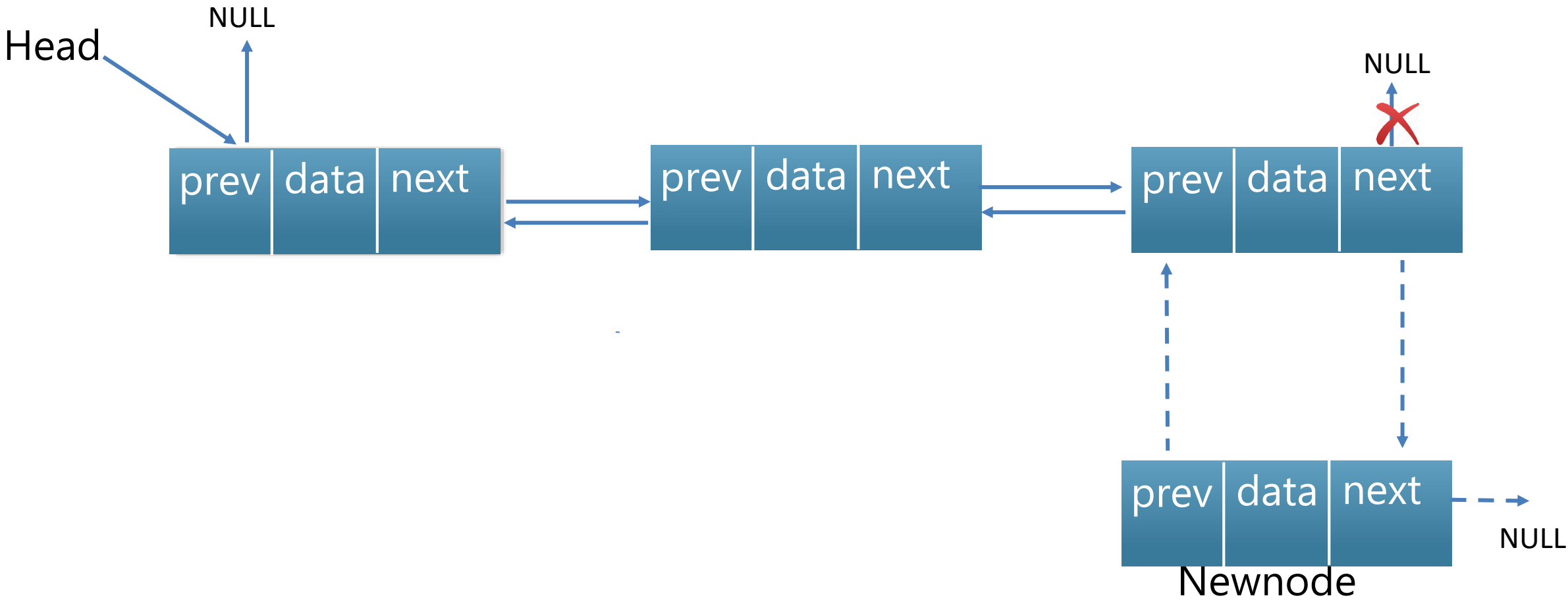
//Inserting at first

```
insertFirst(data):
    Begin
      create a new node
      newnode -> data = data
      newnode -> next = null
      newnode -> prev = null



      if the list is empty, then
          head = newnode
      else
          newnode -> next = head
          head -> prev = newnode
          head = newnode
    End
```

Doubly Linked List – insert at last

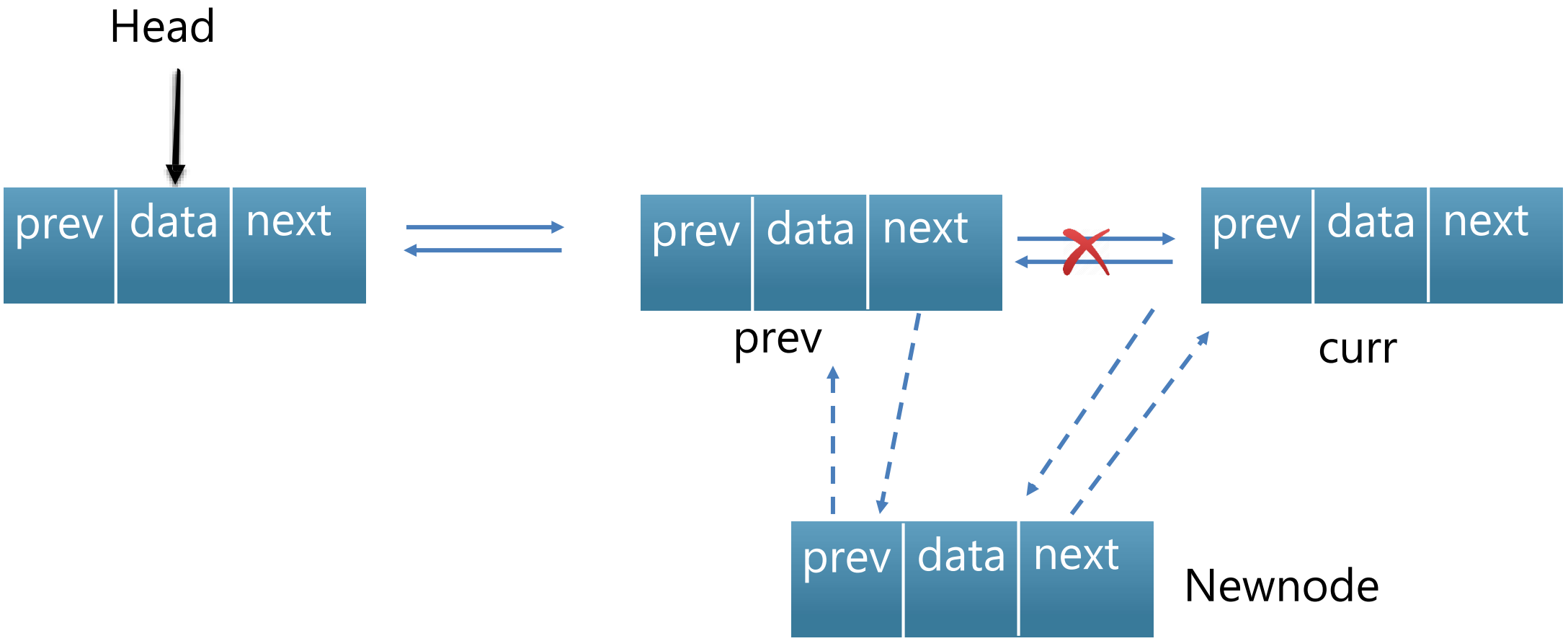


//Inserting at Last

```
insertLast(data):
    Begin
      create a new node
      newnode -> data = data
      newnode -> next = null
      newnode -> prev = null

      if the list is empty, then
          head = newnode
      else
         curr = head
          forever:
                if curr -> next == null
                    break
                curr= curr->Next
          newnode -> prev = curr
          curr - > next = newnode
    End
```

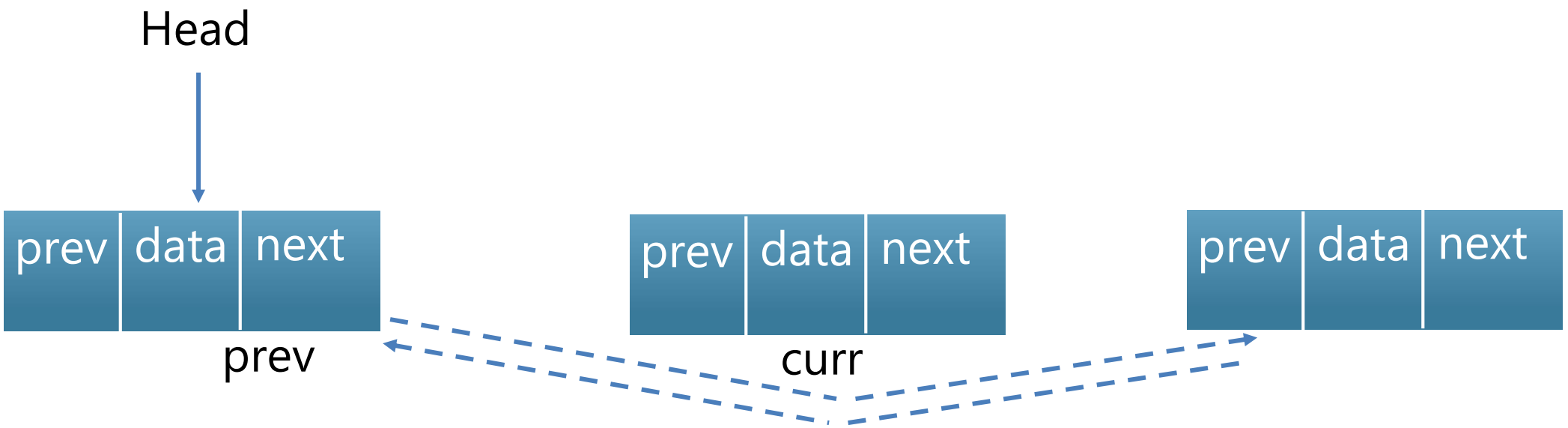
Doubly Linked List – insert at middle



//Inserting at position

```
insertAtPos(data, pos):
    Begin
        create a new node
        newnode -> data = data
        newnode -> next = null
        newnode -> prev = null
        curPos = 1
        if the list is empty, then
            head = newnode
        else
            curr = head
            forever:
                if curPos == pos || curr -> next == null
                    break
                prev = curr
                curr= curr->Next
                curPos++
            prev -> next = newnode
            newnode -> prev = prev
            newnode -> next = curr
            curr -> prev = newnode
    End
```

Doubly Linked List – delete



//Delete

```
deleteAtPos(pos):
    Begin
        curr = head
        curPos = 1
          if head is null, then
              it is Underflow and return
           else if pos ==1
              head = curr ->next
              delete curr
            else
               forever:
                  if curPos == pos || curr -> next == null
                     break
                  prev = curr
                  curr= curr->next
                  curPos++
               prev->next = curr->next
               if curr ->next != null
                   curr ->next->prev = prev
               delete curr
    End
```