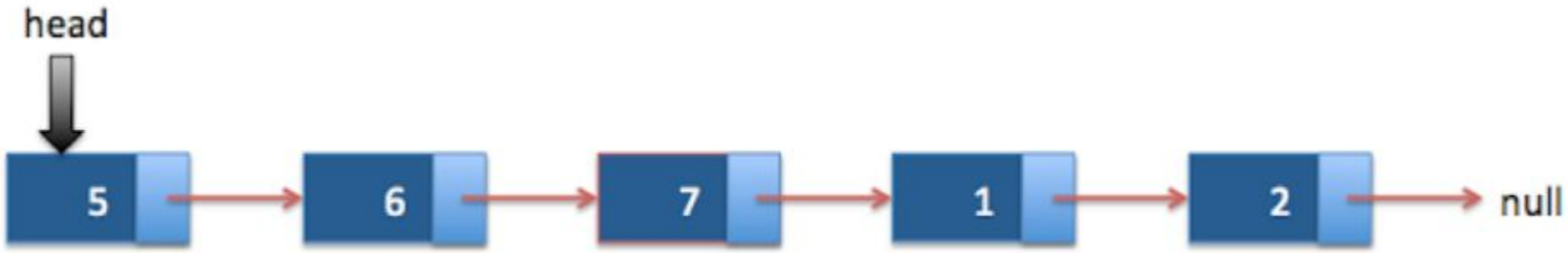
Data Structures

Singly Linked List, Doubly Linked List

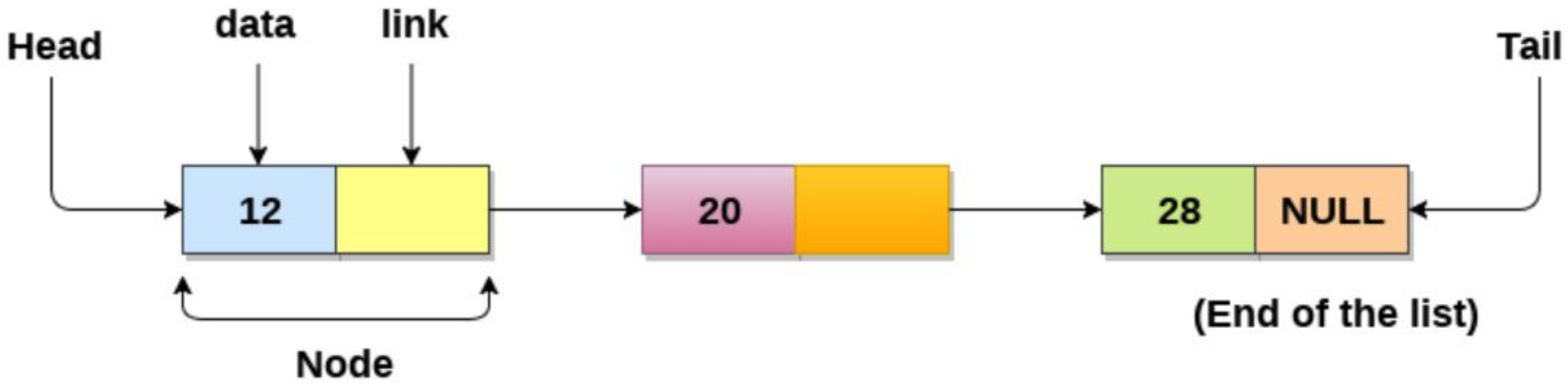
Linked List

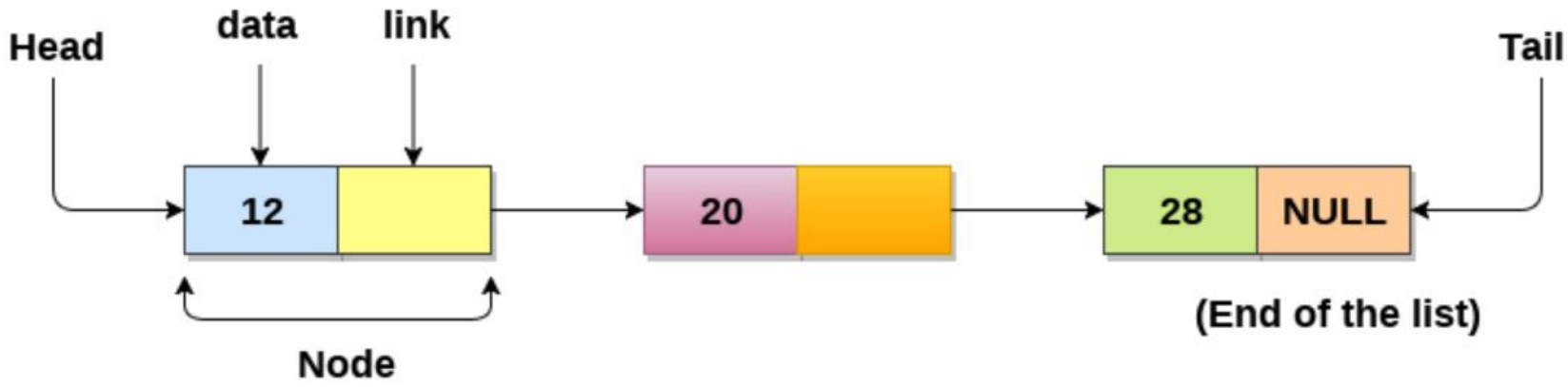
- Linear data structure
- Stores similar elements
- Unlike arrays, elements of linked list are note stored in contiguous locations
- Each element is linked to next element
- Each element in a linked list is called ‘node’
- Easy to insert and delete
- First node is known as ‘head’
- A ‘node’ has data and pointer (or link)
- Two types of linked list – a) Singly linked list and Doubly linked list

Singly Linked List (or one-way chain)



- A collection of ordered set of elements
- ‘nodes’ are stored in a random location in memory
- The last node points to ‘null’



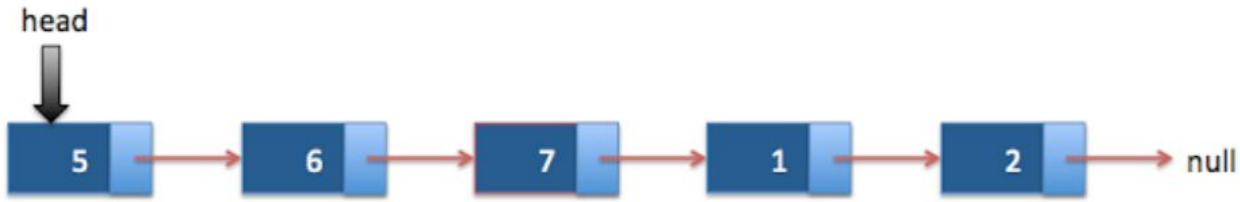


A single node contains two parts,

a) first part stores the data
b) second part contains the pointer (or link) which has the address of the next node in the memory
c) the last node points to 'null'

- Linked list can only be traversed in one direction. Hence, resembles one-way chain
- Linked list do not have the requirement contiguous memory locations
- Each node only contains next link or pointer. Hence, can't traverse in reverse direction.
- The location of the node can be anywhere in the memory. (This actually indicates the correct utilisation of memory space also)
- The size of the linked list is limited by the available memory
- Can store values related to primitive types or objects

Operations associated with Singly Linked List



- Node creation : create a node

- Insertion
  - a) insert at the beginning of the linked list
  - b) insert at the end of the linked list
  - c) insert at specific location in the linked list

- Deletion
  - a) deletion from the beginning of the linked list
  - b) deletion from the end of the linked list
  - c) deleting a specific node

- Traverse
- Searching

- In Java, Linked List can be represented as a class
- Node also represented as a class

```
public class SLL1 {

Node head; // 'head' of the linked list

// node in the Linked list is a class
static class Node{
int data;
Node next;

//constructor is used to create a new 'Node' and 'next' is by default is initialized as 'null'
Node(int d){
    data = d;
    next = null;
} //end of constructor

}//end of static class node

} // end of class SLL1
```

Class for Singly Linked List

Create four nodes in the Singly Linked List

```
public class SLL1 {

Node head; // 'head' of the linked list

// node in the Linked list is a class
static class Node{
int data;
Node next;

//constructor is used to create a new Node and
Next is by default is initialized as null
Node(int d){
    data = d;
    next = null;
} //end of constructor

} //end of static class node
```

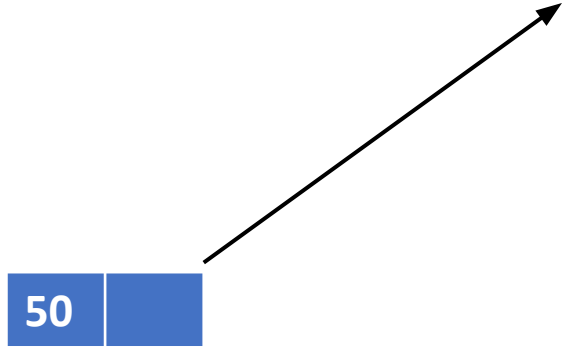
```
//main method
public static void main(String[] args) {

SLL1 LList = new SLL1(); // create an empty
Linked list

// create 4 nodes
LList.head = new Node(10);
Node two = new Node(20);
Node three = new Node(30);
Node four = new Node(40);
//four nodes allocated dynamically

//link first 'head' node with node 'two'
LList.head.next = two;

//link node 'two' to node 'three'
two.next = three;

//link node 'three' to node 'four'
three.next = four;

}//end of main method

} //end of class SLL1
```

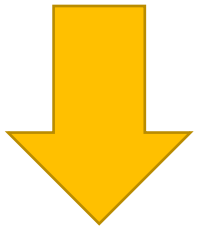
Display four nodes in the Singly Linked List

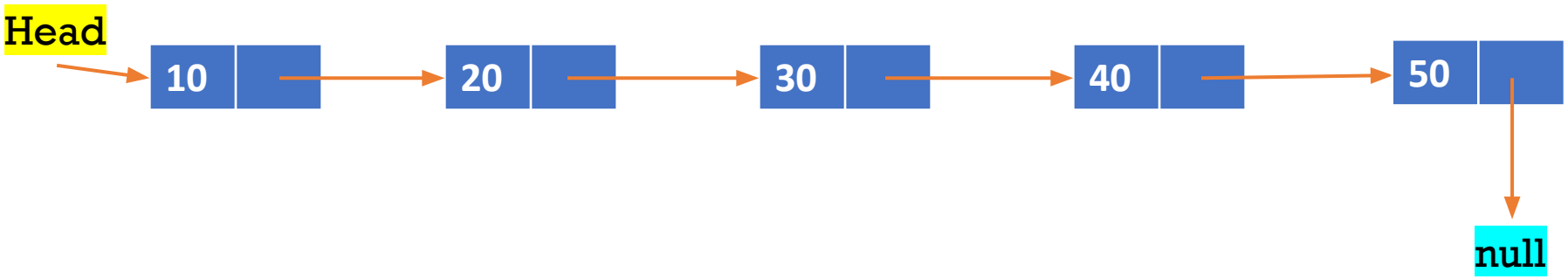
```
public class SLL1 {

Node head; // 'head' of the linked list

// node in the Linked list is a class
static class Node{
int data;
Node next;

//constructor is used to create a new Node and
Next is by default is initialized as null
Node(int d){
    data = d;
    next = null;
} //end of constructor

} //end of static class node

public void DisplayList() {
Node node = head;

while(node!=null) {
   System.out.println("Vale at each node: "+node.data+"
");
   node = node.next;
}
}//end of DisplayList function
```

```
//main method
public static void main(String[] args) {

SLL1 LList = new SLL1(); // create an empty
Linked list

// create 4 nodes
LList.head = new Node(10);
Node two = new Node(20);
Node three = new Node(30);
Node four = new Node(40);
//four nodes allocated dynamically

//link first 'head' node with node 'two'
LList.head.next = two;

//link node 'two' to node 'three'
two.next = three;

//link node 'three' to node 'four'
three.next = four;

// display value at each node
LList.DisplayList(); // Traverse the list

}//end of main method

} //end of class SLL1
```

Now, how to insert a node at the end





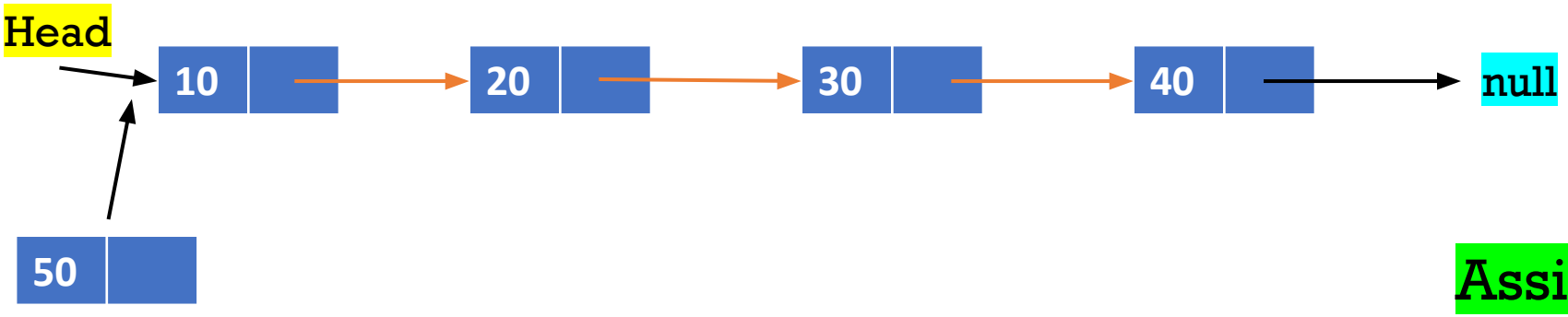


```
public class SLL2 {

Node head; // 'head' of the linked list

// node in the Linked list is a class
static class Node{
int data;
Node next;

//constructor is used to create a new Node and Next is by default is initialized as null
Node(int d){
data = d;
next = null;
}// end of constructor Node
}//end of class Node

public static SLL2 insert(SLL2 list, int d) {
//create new node
Node n = new Node(d);
n.next = null;

//check if the linked list passed is empty. if so then
//initialize head with n
if(list.head == null) {
list.head = n;
}
else { //if the list is not empty then traverse to the last node
Node tmp = list.head;
while(tmp.next!=null)
tmp = tmp.next;

//tmp will be the last node in the list
//point tmp to n
tmp.next = n;
}

return list;
}
```

SLL2.java (uploaded in teams)

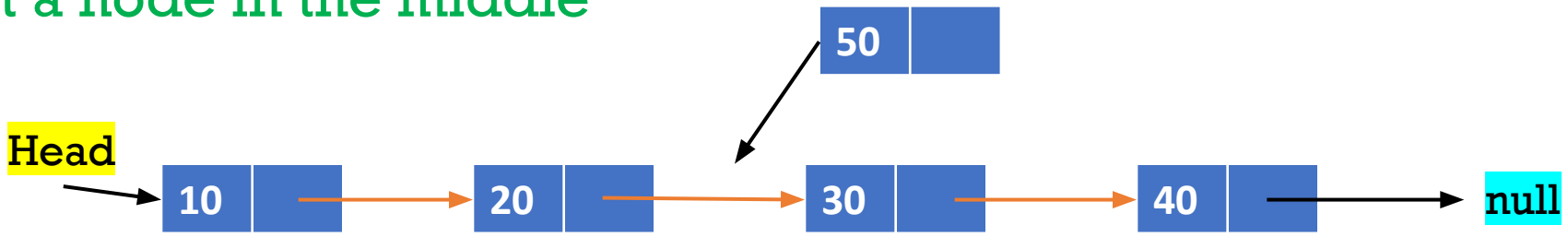
```
public void DisplayList() {
Node node = head;

while(node!=null) {
System.out.println("Vale at each node: "+node.data+" ");
node = node.next;
}
}//end of DisplayList function

public void DisplayList1(SLL2 l) {
Node node = l.head;

while(node!=null) {
System.out.println("Vale at each node: "+node.data+" ");
node = node.next;
}
}

//main method
public static void main(String[] args) {

SLL2 LList = new SLL2(); // create an empty Linked list

// create 4 nodes
LList.head = new Node(10);
Node two = new Node(20);
Node three = new Node(30);
Node four = new Node(40);
//four nodes allocated dynamically

//link first 'head' node with node 'two'
LList.head.next = two;

//link node 'two' to node 'three'
two.next = three;

//link node 'three' to node 'four'
three.next = four;

// display value at each node
LList.DisplayList();

System.out.println("\nInsert the node at the end of the list\n");
```

Insert a node first



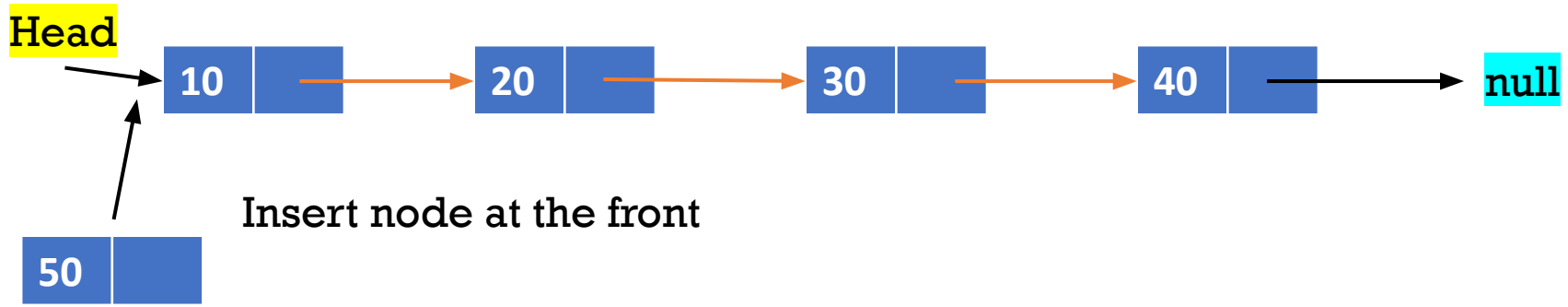
Assignment 3:

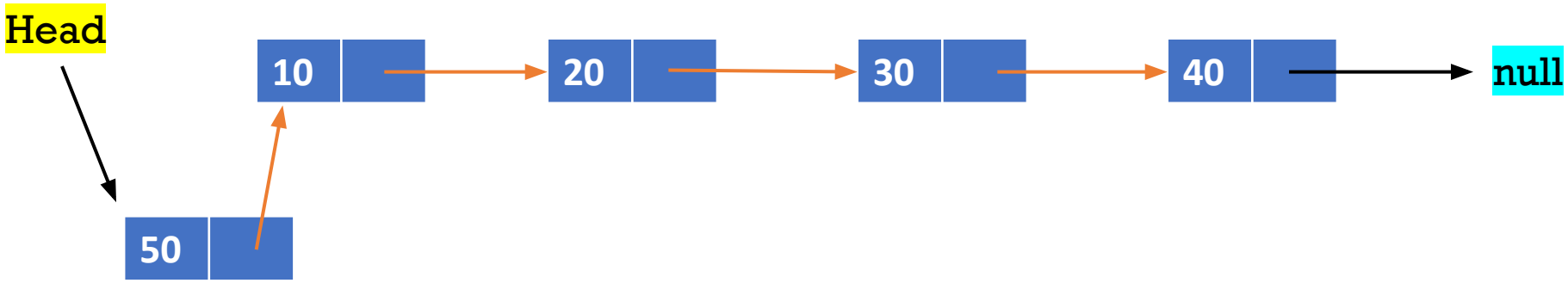
Insert a node in the middle



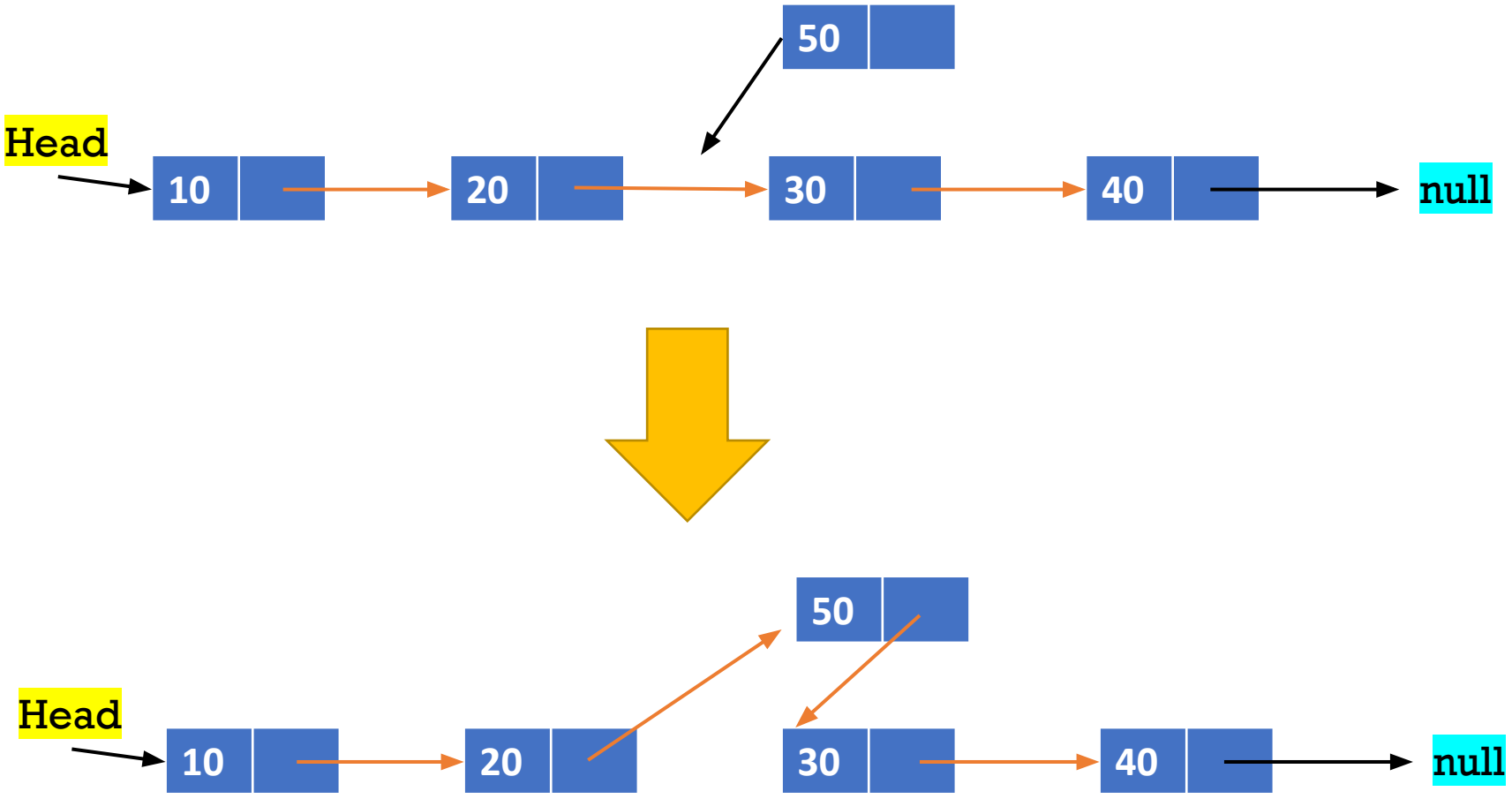
Similarly,
Delete a node from first, middle, and end

Insert as first node in the linked list

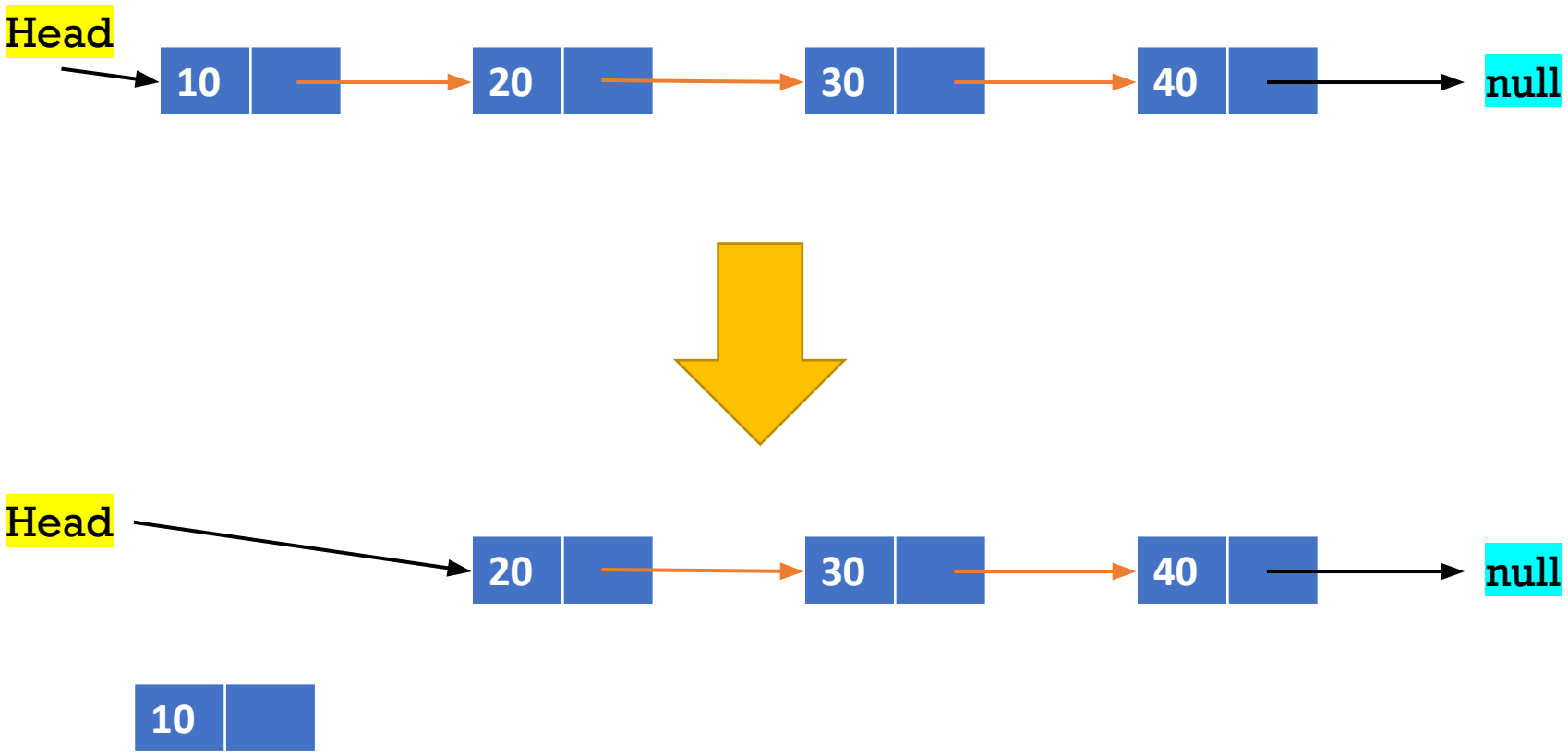




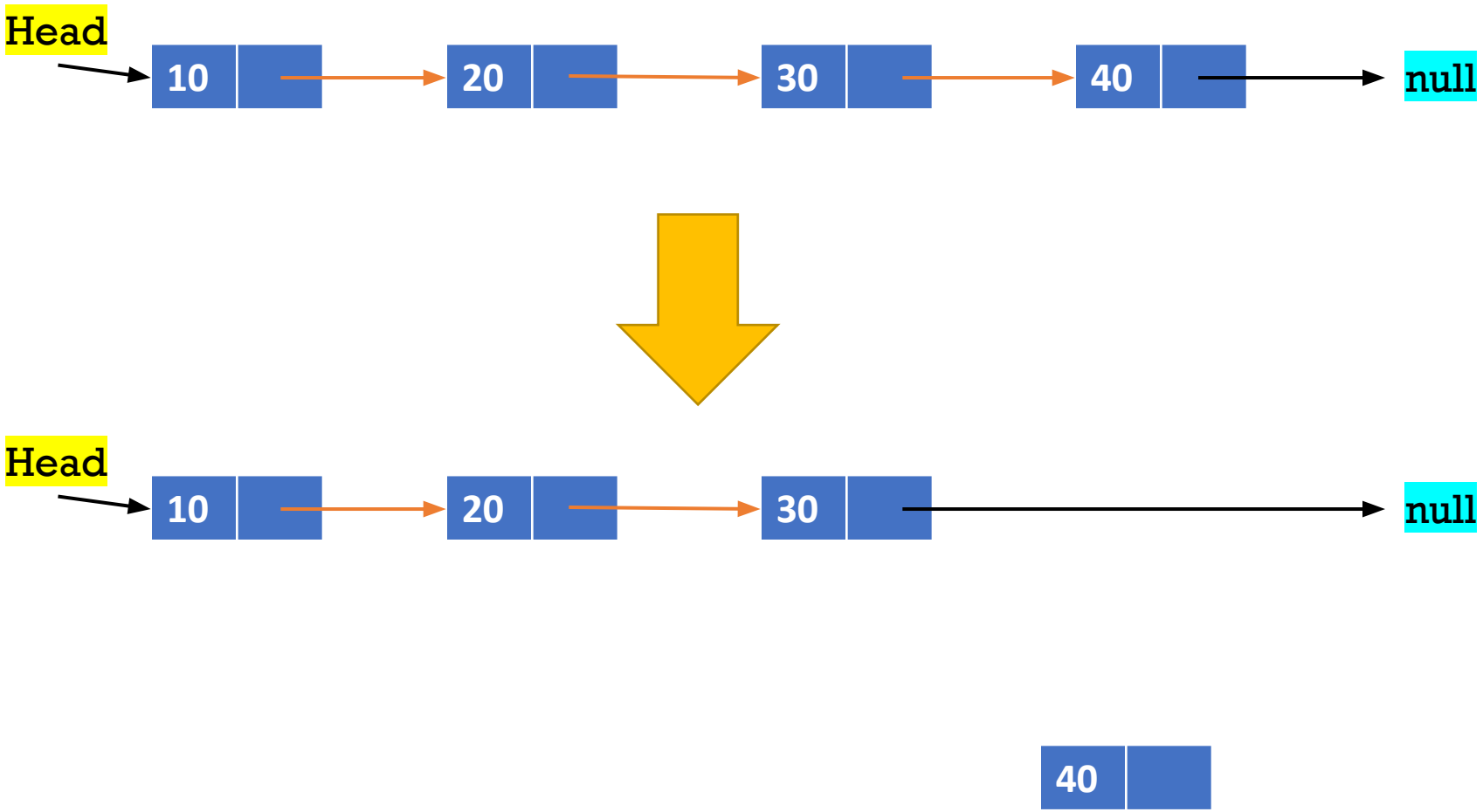
Insert a node in the middle



Delete a node from first

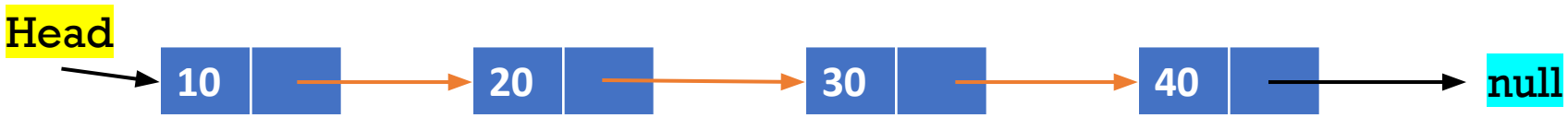


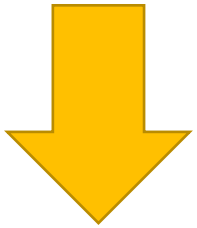
Delete a node from the end



Delete a node in between

Delete a node with value 30

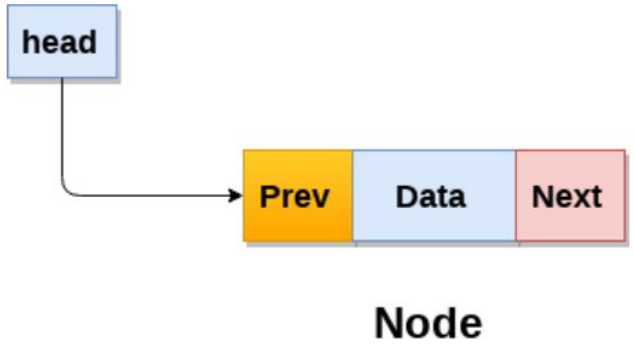




(hint:Pointer of node with value 20 is redirected to point the node with value 40)

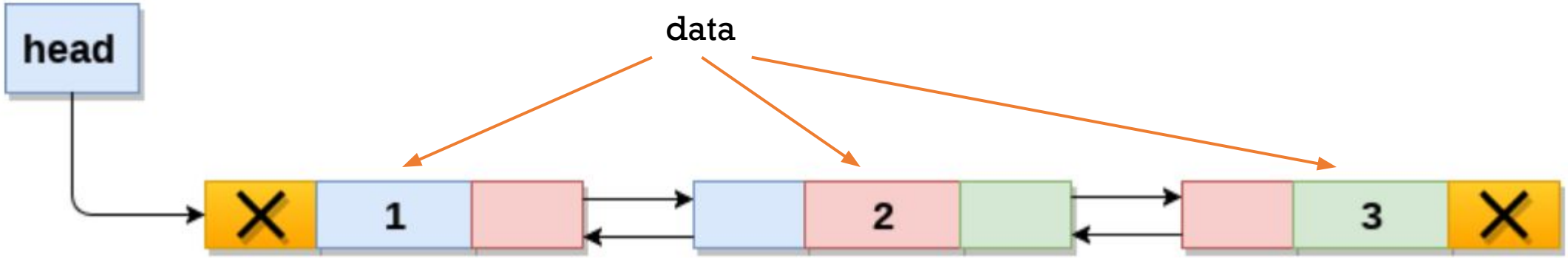
Doubly Linked List

A node contains two pointers to point previous and next node



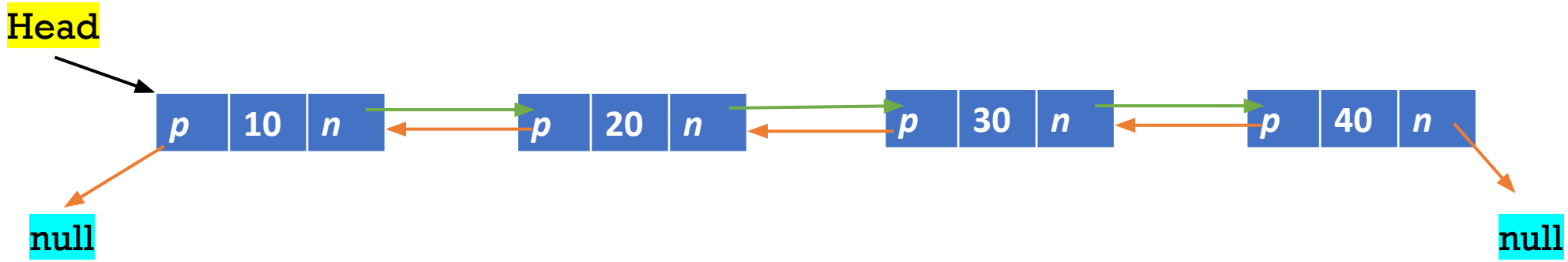
A node contains three parts,

1) previous pointer
2) data
3) next pointer



As each node contains pointers to point previous and next node, doubly linked list can be traversed back and forth which overcome the limitation of Singly linked list

Operations - Doubly Linked List



- Insertion of a node
  - adding the node to the beginning of the list
  - adding the node to the end of the list
  - adding the node in between

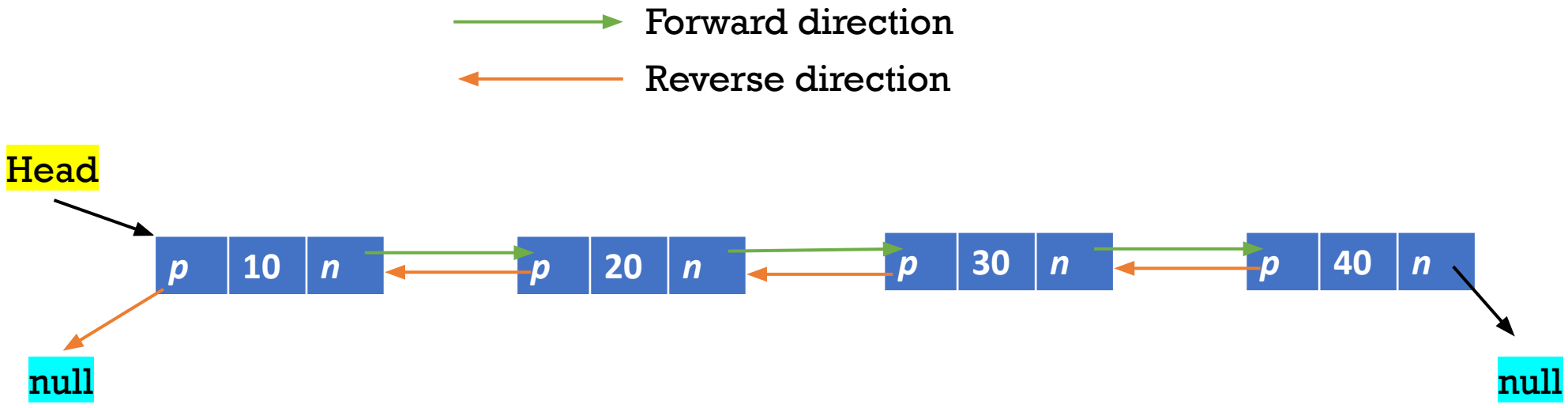
- Deletion of a node
  - delete a node from the beginning of the list
  - delete a node from the end of the list
  - delete a node in between

- Search
  - compare the data of each node in the list with the item to be searched and return the location of the matched node in the list, else return null

- Traverse
  - visit each node in the list once

What can be the advantage and disadvantage of Doubly linked list over Singly linked list ?

Represent a node in DLL

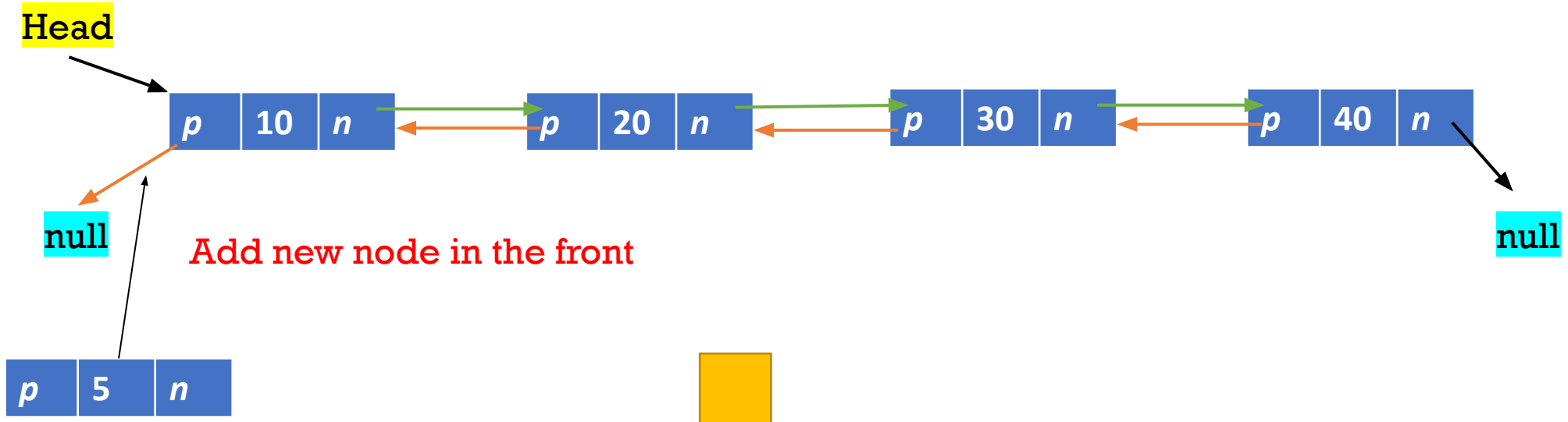


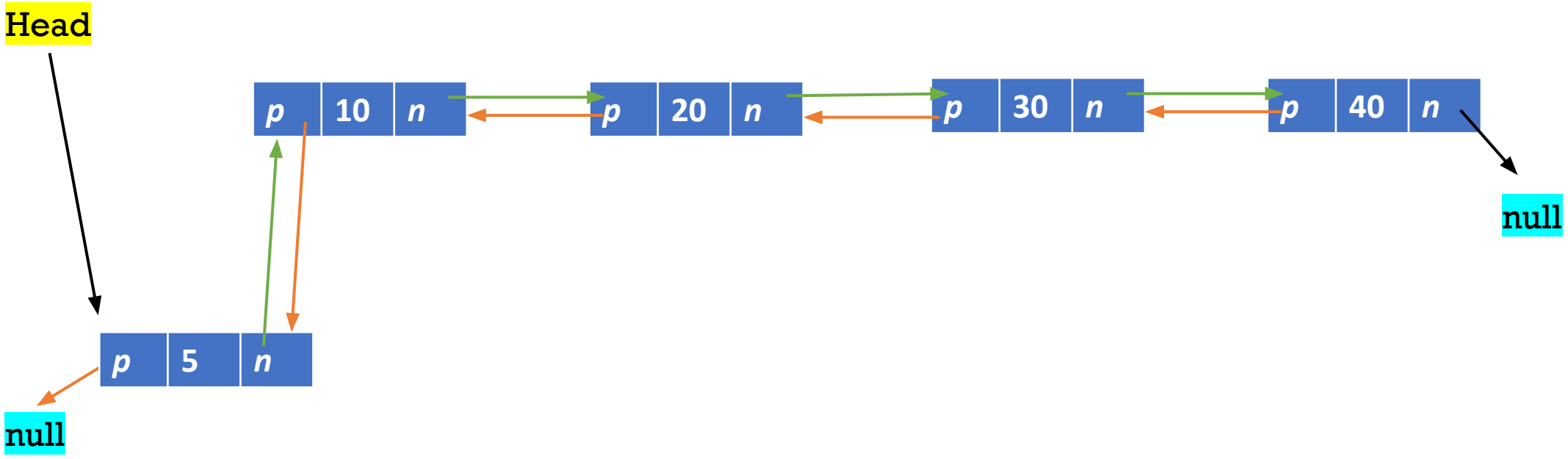
```
public class DLL1 {

Node head; // head

//Node in Doubly Linked List
static class Node{
int data;
Node prev;
Node next;

//constructor to create a node
Node(int d){
data = d;
}
}


public void DisplayList(Node node) {
Node tmp = null;

System.out.println("\nTraversal in forward direction\n");
while(node!=null) {
System.out.println("Vale at each node: "+node.data+" ");
tmp = node;
node = node.next;
}

System.out.println("\nTraversal in reverse direction\n");
while(tmp!=null) {
System.out.println("Vale at each node: "+tmp.data+" ");
tmp = tmp.prev;
}

}  //end of DisplayList function
```

```
public static void main(String[] args) {

    DLL1 LList = new DLL1(); // create an empty Linked list

// create 4 nodes
LList.head = new Node(10);
Node two = new Node(20);
Node three = new Node(30);
Node four = new Node(40);

LList.head.prev = null;
LList.head.next = two;

two.prev = LList.head;
two.next = three;

three.prev = two;
three.next = four;

four.prev = three;
four.next = null;

LList.DisplayList(LList.head);

}

}
```

Code to create node in DLL and display the data
(DLL1.java uploaded in teams)

Add node in the front of DLL





```
public class DLL2 {

Node head; // head

//Node in Doubly Linked List
static class Node{
int data;
Node prev;
Node next;

//constructor to create a node
Node(int d){
data = d;
}
}


public void insertNode(int d) {

//create node with data
Node newNode = new Node(d);

//initialize the newNode's prev as null and next as head
newNode.prev = null;
newNode.next = head;

//head's prev should point to newNode
if(head!=null) {
head.prev = newNode;
}

//make newNode as head
head = newNode;
}
```

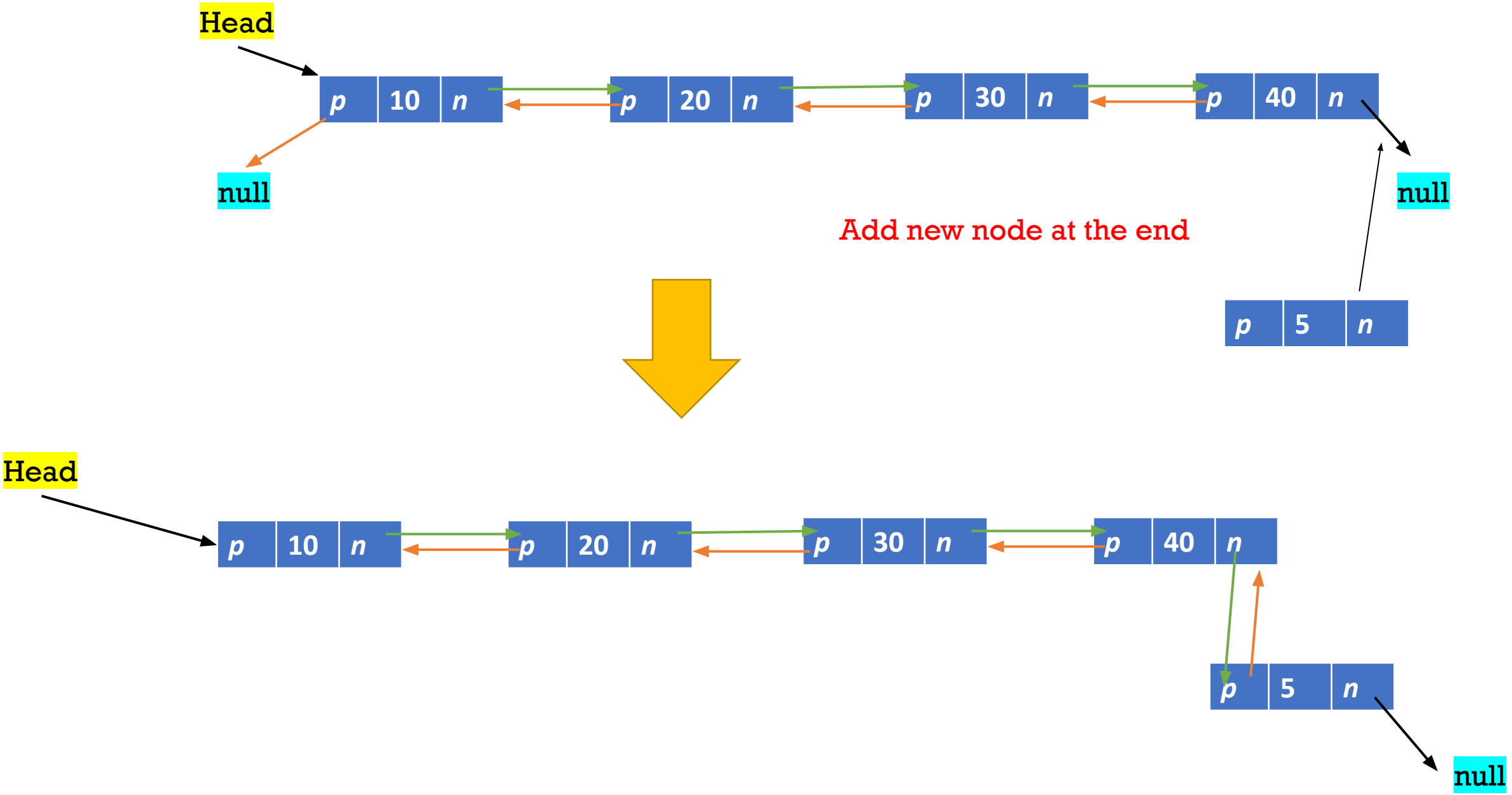
```
public void DisplayList(Node node) {
Node tmp = null;

System.out.println("\nTraversal in forward direction\n");
while(node!=null) {
System.out.println("Vale at each node: "+node.data+" ");
tmp = node;
node = node.next;
}

System.out.println("\nTraversal in reverse direction\n");
while(tmp!=null) {
System.out.println("Vale at each node: "+tmp.data+" ");
tmp = tmp.prev;
}

}//end of DisplayList function
```

```
public static void main(String[] args) {

    DLL2 LList = new DLL2(); // create an empty Linked list

// create 4 nodes
LList.head = new Node(10);
Node two = new Node(20);
Node three = new Node(30);
Node four = new Node(40);

LList.head.prev = null;
LList.head.next = two;

two.prev = LList.head;
two.next = three;

three.prev = two;
three.next = four;

four.prev = three;
four.next = null;


LList.DisplayList(LList.head);

System.out.println("Insert new Node at the front");

LList.insertNode(5);

LList.DisplayList(LList.head);


}

}
```

Code to insert a node in the front of the DLL

(DLL2.java uploaded in teams)

Add node at the end of DLL



```
// create a dll and add a node at the
end of the list

public class DLL3 {

Node head; // head

//Node in Doubly Linked List
static class Node{
int data;
Node prev;
Node next;

//constructor to create a node
Node(int d){
data = d;
}
}
```

```
public void insertNode(int d) {
//create node with data
Node newNode = new Node(d);

//initialize the newNode's next as null as it will
//be the last node
newNode.next = null;

if(head == null) {
newNode.prev = null;
head = newNode;
return;
}
else {
Node tmp = head;

//traverse till last node is reached
while(tmp.next!=null) {
tmp = tmp.next;
}

tmp.next = newNode;
newNode.prev = tmp;

}

}
```

```
public void DisplayList(Node node) {
Node tmp = null;

System.out.println("\nTraversal in forward direction\n");
while(node!=null) {
System.out.println("Vale at each node: "+node.data+" ");
tmp = node;
node = node.next;
}

System.out.println("\nTraversal in reverse direction\n");
while(tmp!=null) {
System.out.println("Vale at each node: "+tmp.data+" ");
tmp = tmp.prev;
}

}//end of DisplayList function


public static void main(String[] args) {

    DLL3 LList = new DLL3(); // create an empty Linked list

// create 4 nodes
LList.head = new Node(10);
Node two = new Node(20);
Node three = new Node(30);
Node four = new Node(40);

LList.head.prev = null;
LList.head.next = two;

two.prev = LList.head;
two.next = three;

three.prev = two;
three.next = four;

four.prev = three;
four.next = null;


LList.DisplayList(LList.head);

System.out.println("Insert new Node at the end");
```

Code to insert a node at the end of DLL

(DLL3.java uploaded in teams)

Add node in the middle of DLL