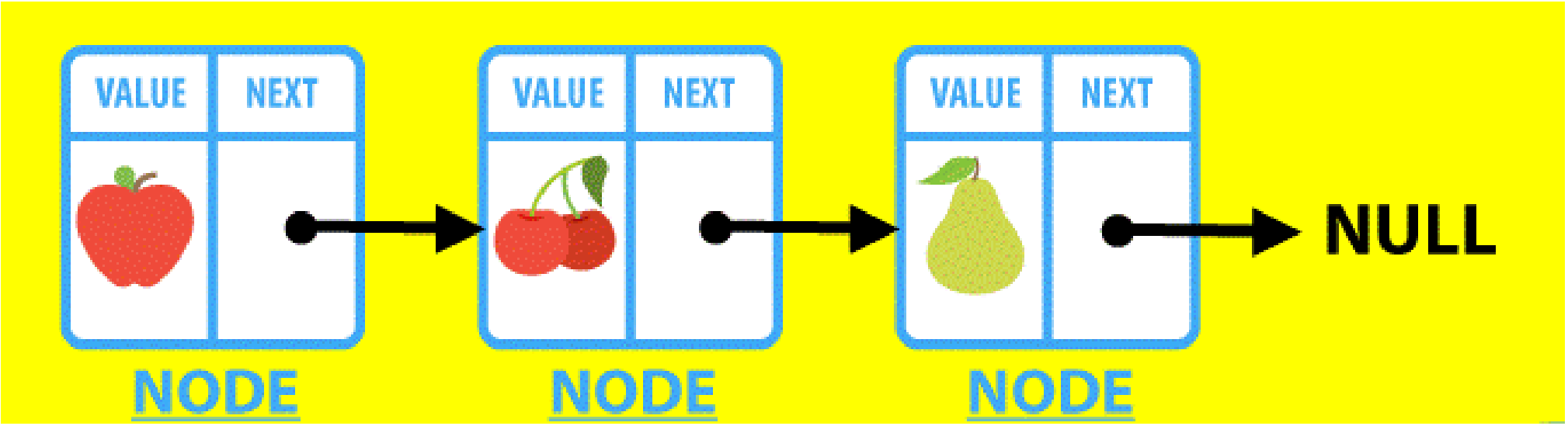
Data Structure

Lec 05

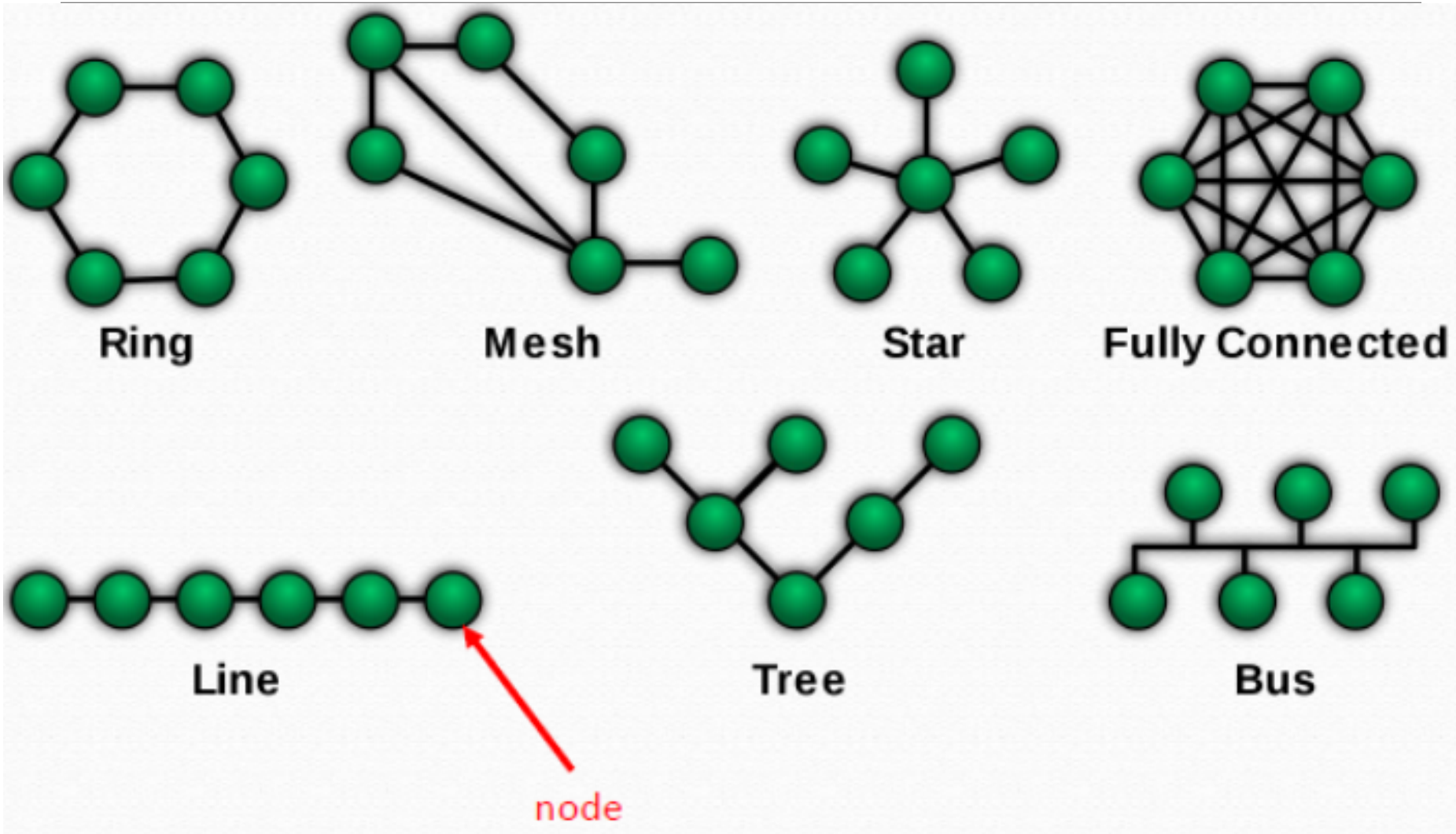
Linked Lists



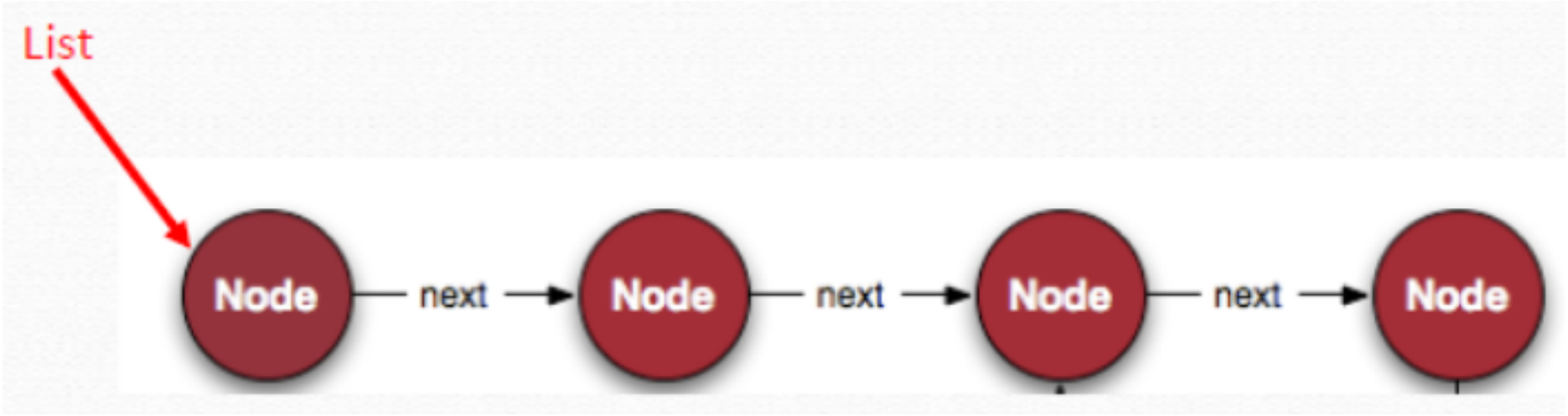
Linked List

Introduction to Linked Lists

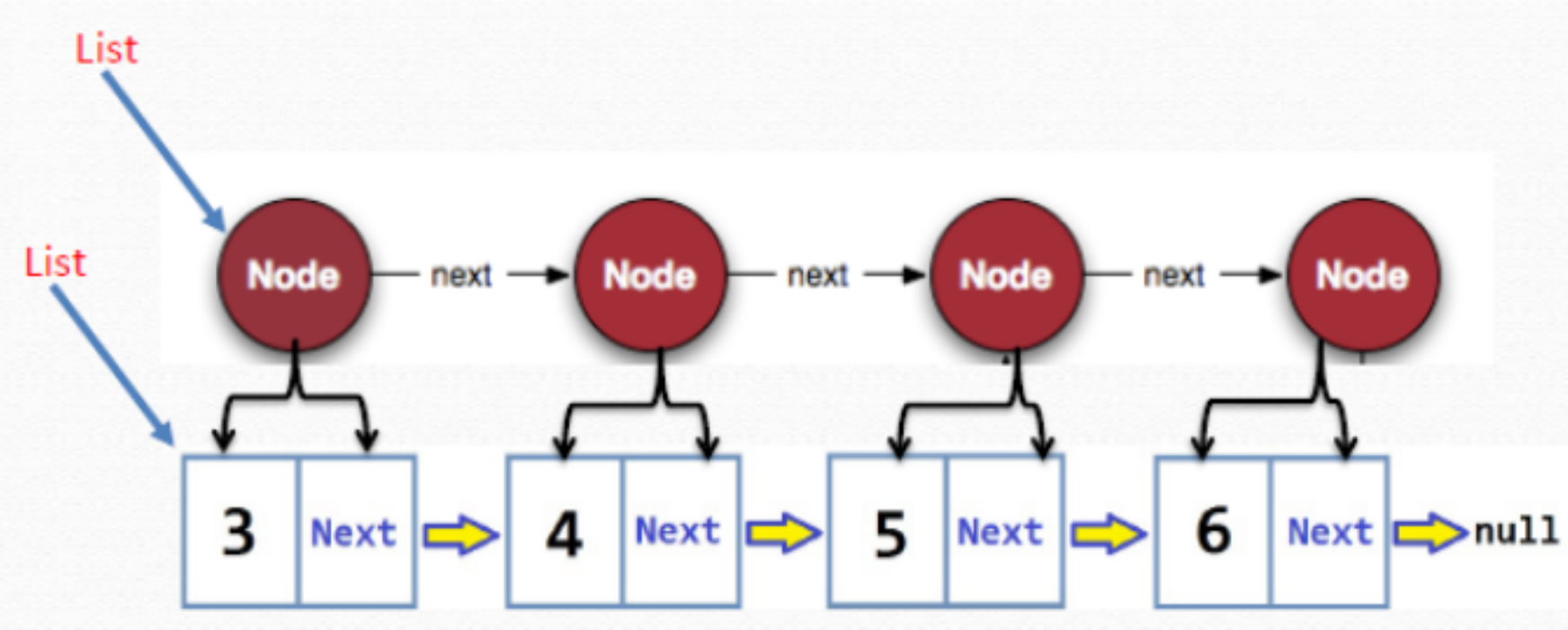
Network Topologies



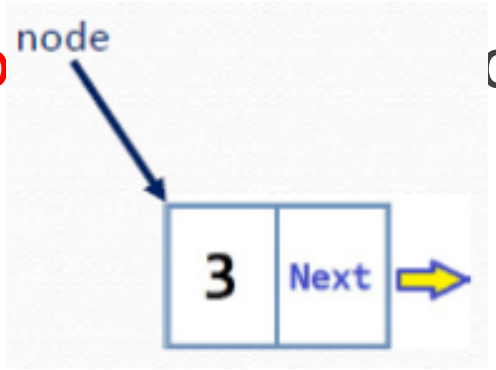
Linked List Nodes



Linked List Nodes

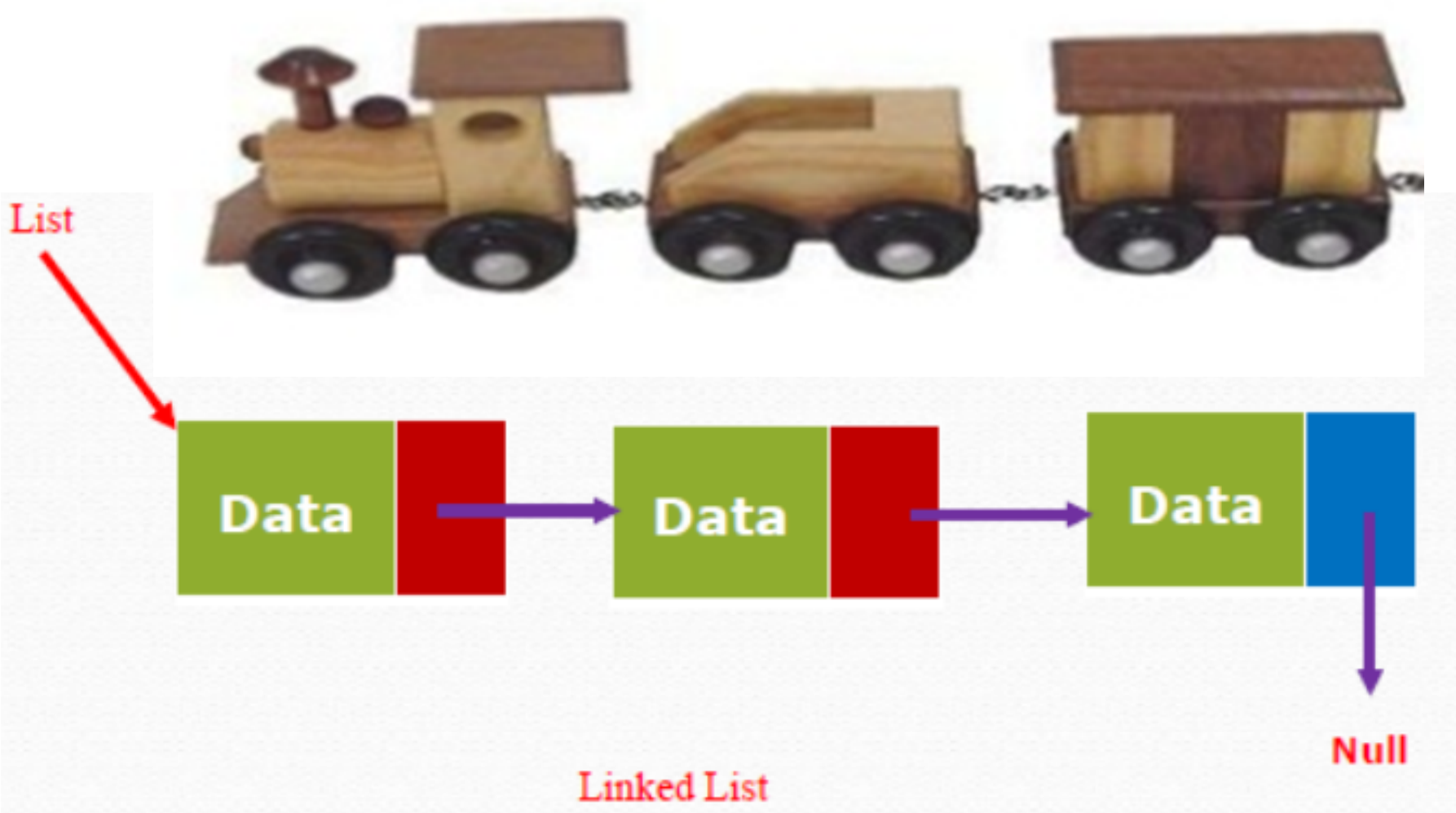


What is a Linked List?

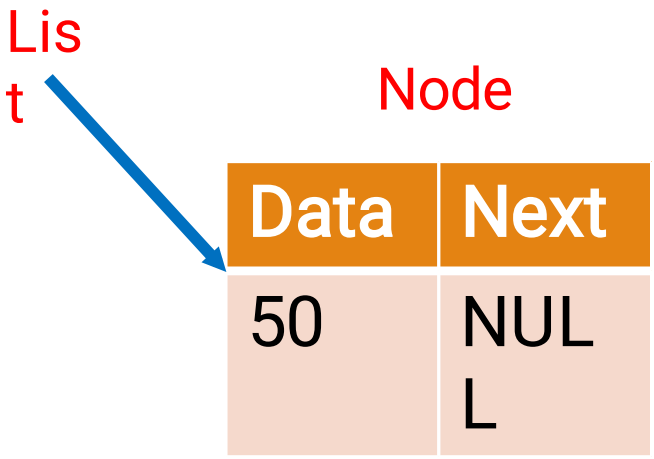
- A linked list is a linear collection of specially designed data elements, called nodes, linked to one another by means of pointers.
- Each node is divided into two parts:
  - The first part contains the information of the element, and,
  - The second part contains the address of the next node in the linked list.
- Address part of the no 

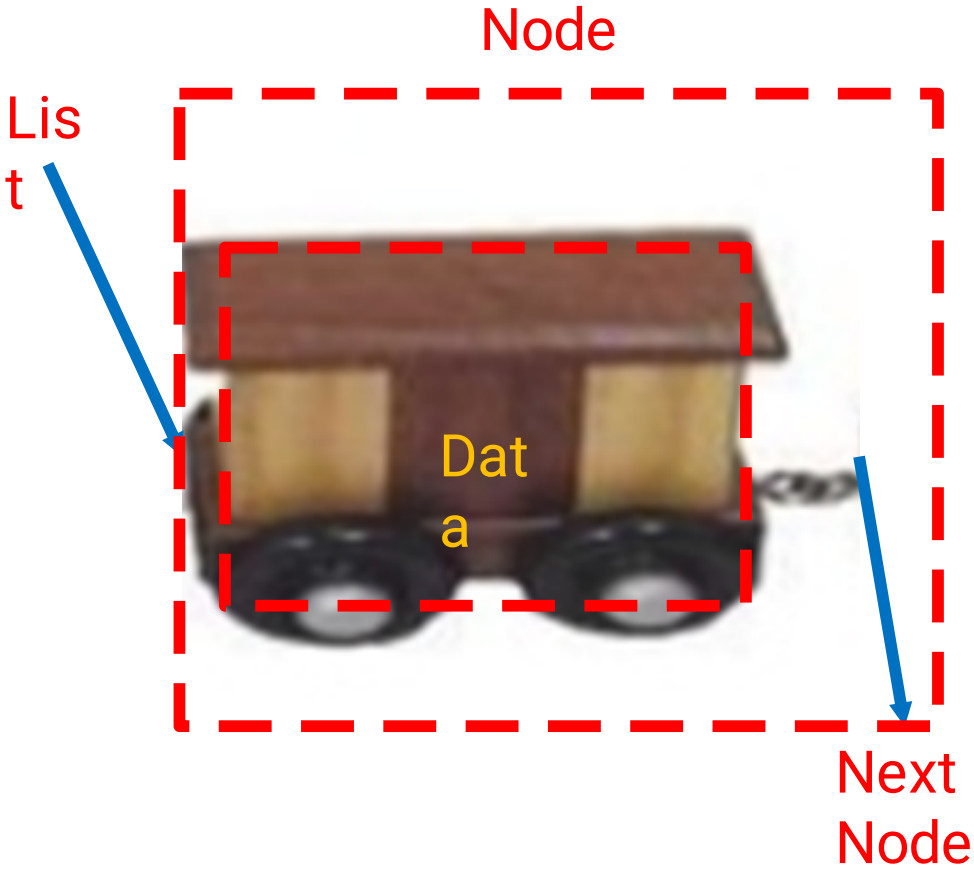
d linked or next field.

What is a Linked List?



What is a Linked List?

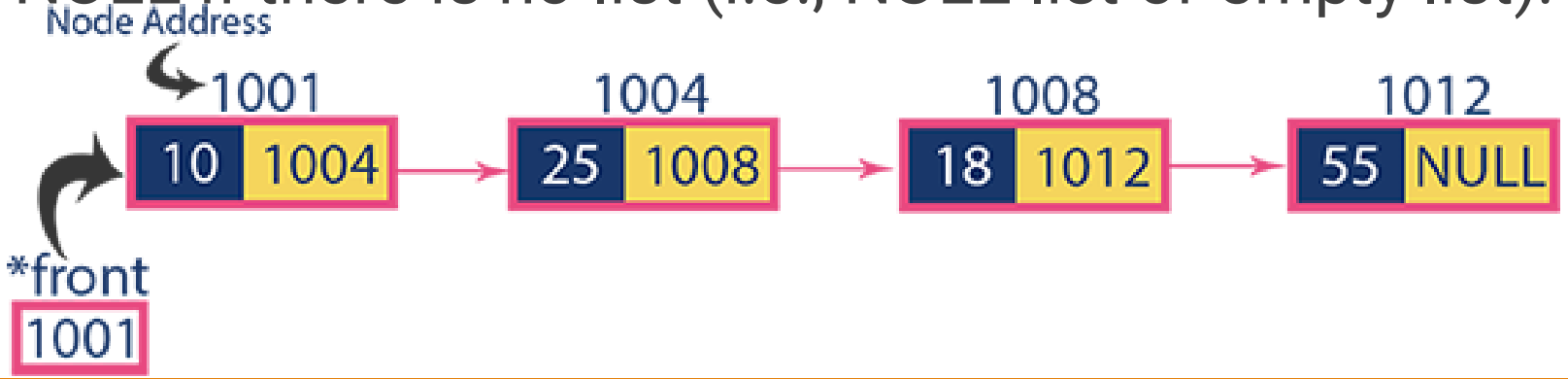




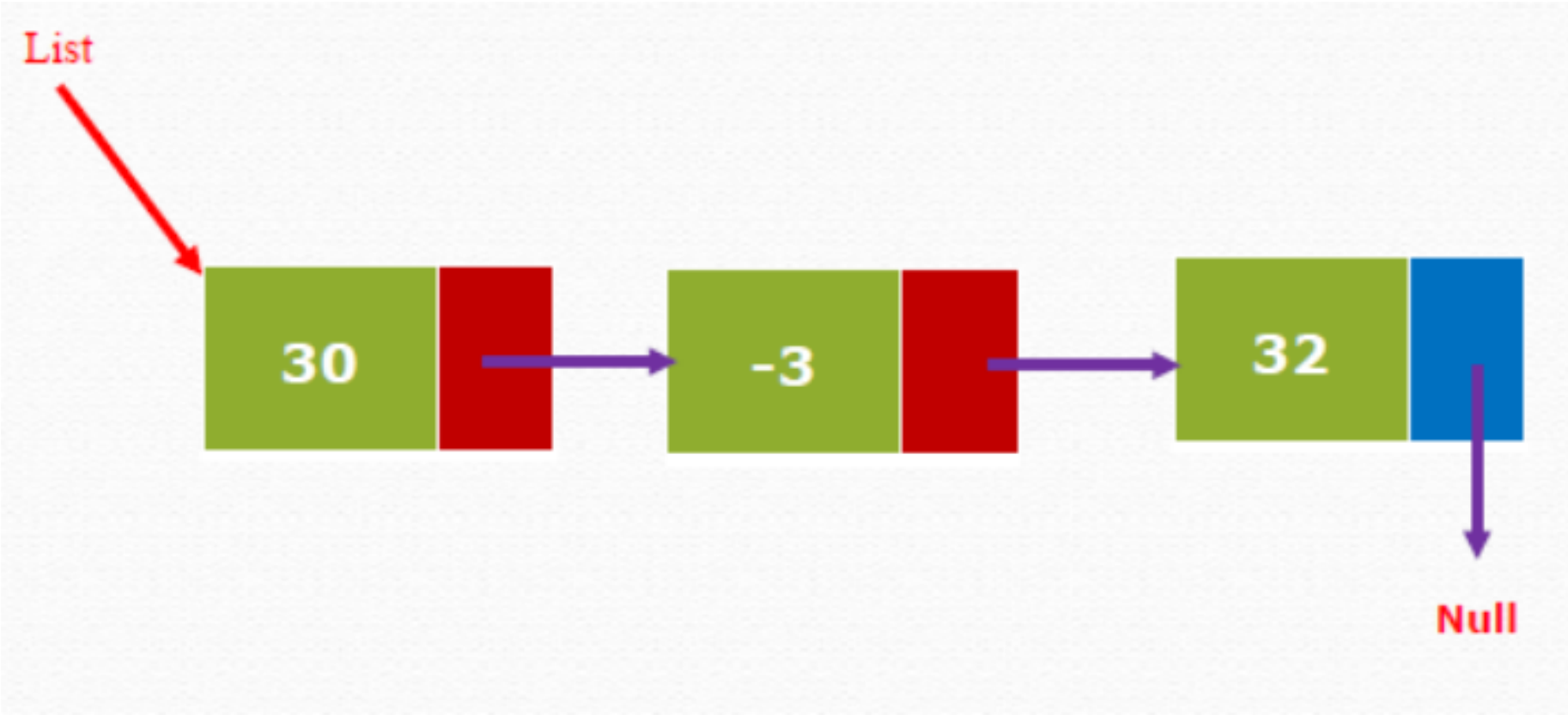
- List->data=50
- List->next=NULL

What is a Linked List?

- Each node contains the data items and the right part represents the address of the next node.
- The next pointer of the last node contains a special value, called the NULL pointer, which does not point to any address of the node. That is NULL pointer indicates the end of the linked list.
- List pointer will hold the address of the 1st node in the list List=NULL if there is no list (i.e.; NULL list or empty list).

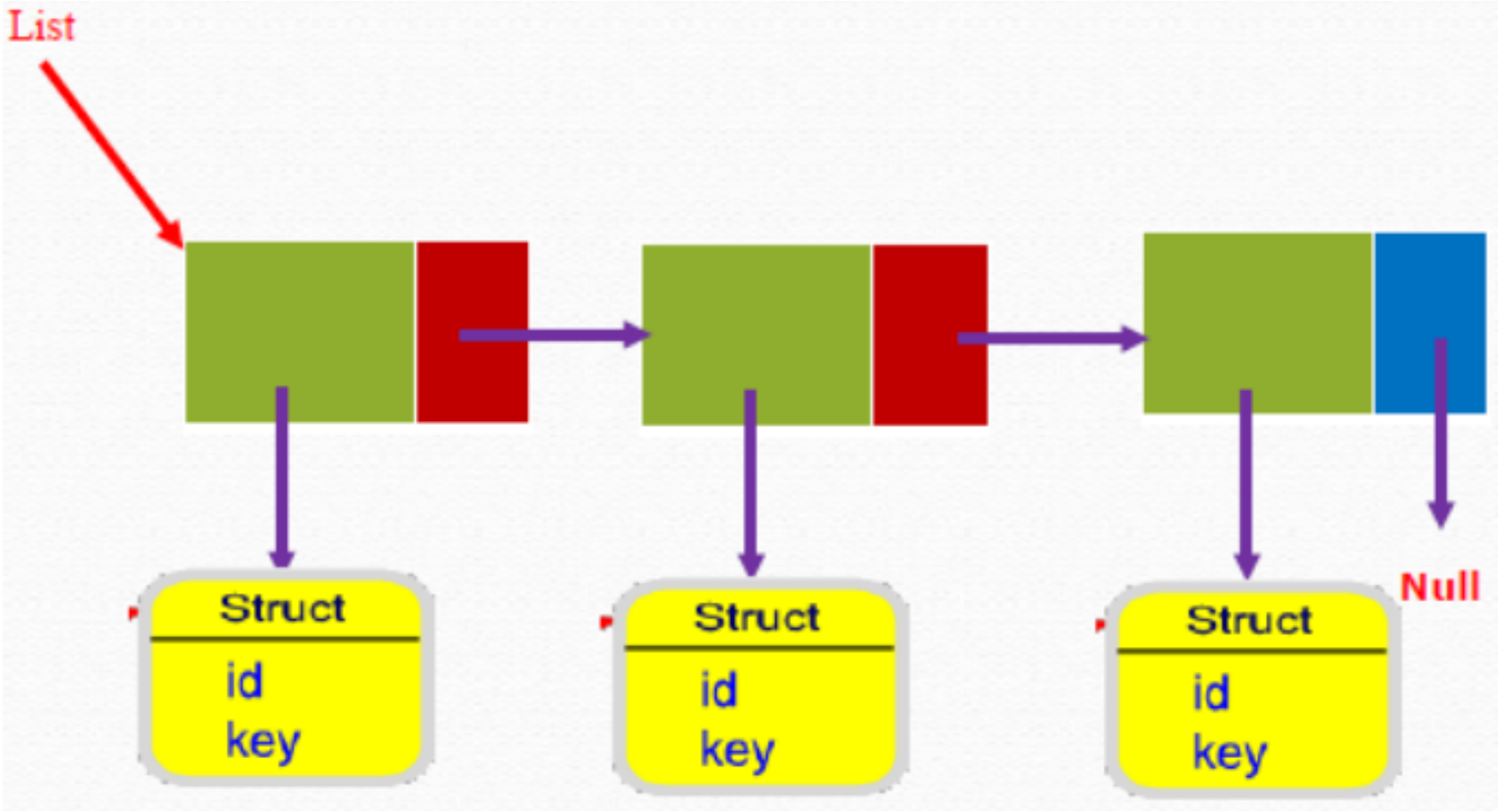


Linked List Representation of Integers



Linked List Representation of Fractions

Linked List Representation of Structures



Linked List Operations

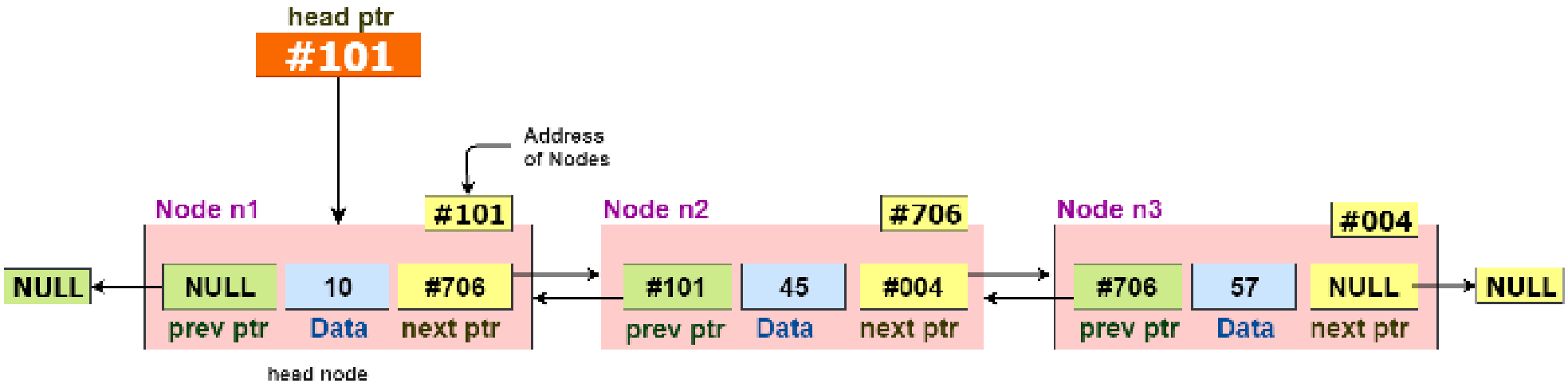
The primitive operations performed on the linked list are as follows:

Linked List Operations

- Creation operation is used to create a linked list. Once a linked list is created with one node, insertion operation can be used to add more elements in a node.
- Insertion operation is used to insert a new node at any specified location in the linked list. A new node may be inserted:
  1. At the beginning of the linked list.
  2. At the end of the linked list.
  3. At any specified position in between in a linked list.

Linked List Operations

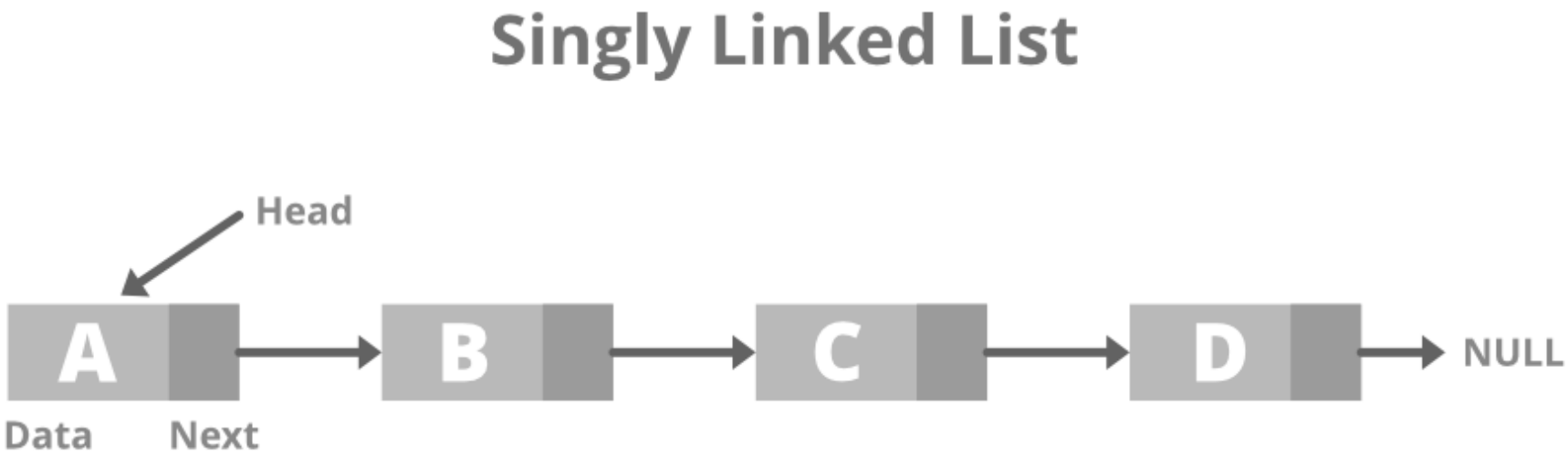
- Deletion operation is used to delete an item (or node) from the linked list. A node may be deleted from the
  1. Beginning of a linked list
  2. End of a linked list
  3. Specified location of the linked list
- Traversing is the process of going through all the nodes from one end to another end of a linked list.
  - In a singly linked list we can visit from left to right, forward traversing, nodes only.
  - But in doubly linked list forward and backward traversing is possible.



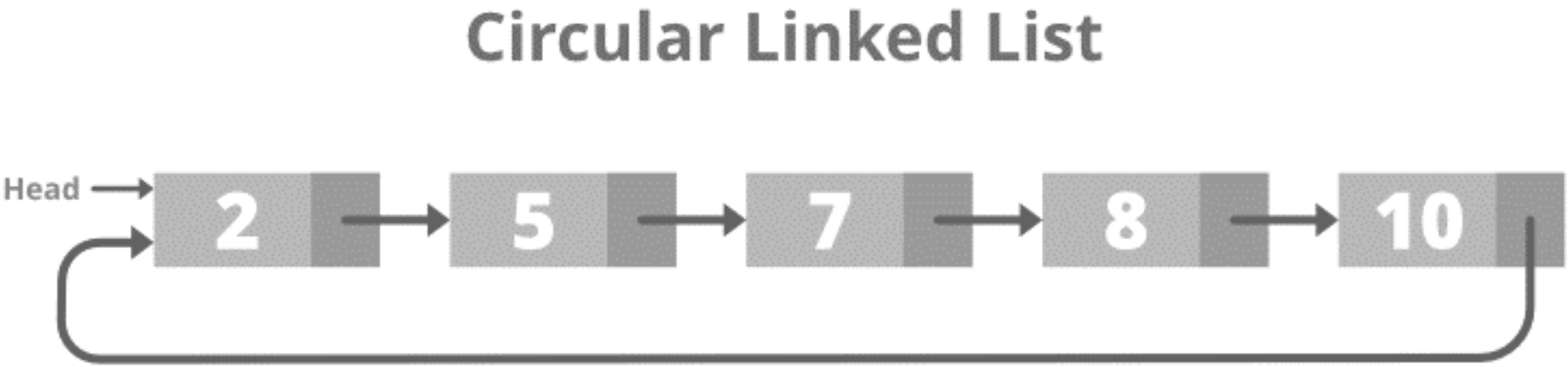
Linked List Types

Linked List Types

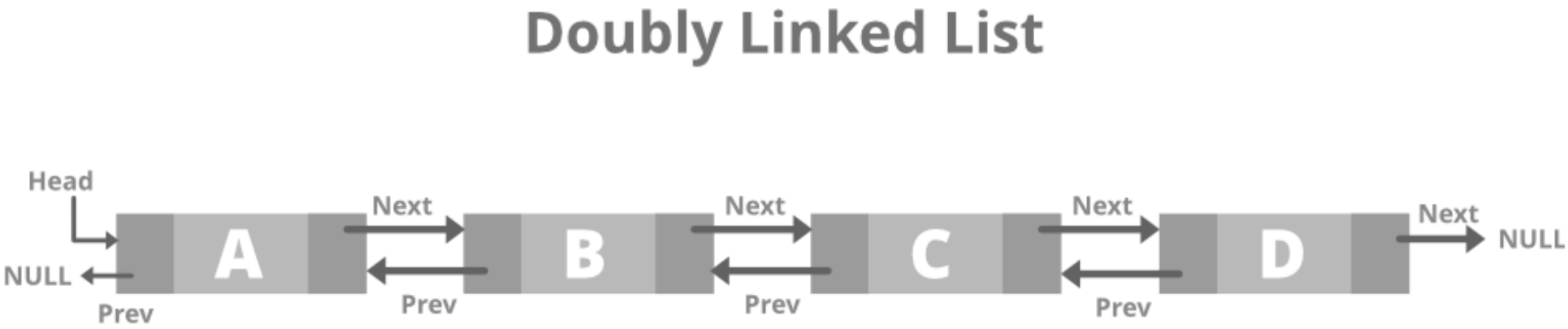
Singly linked list



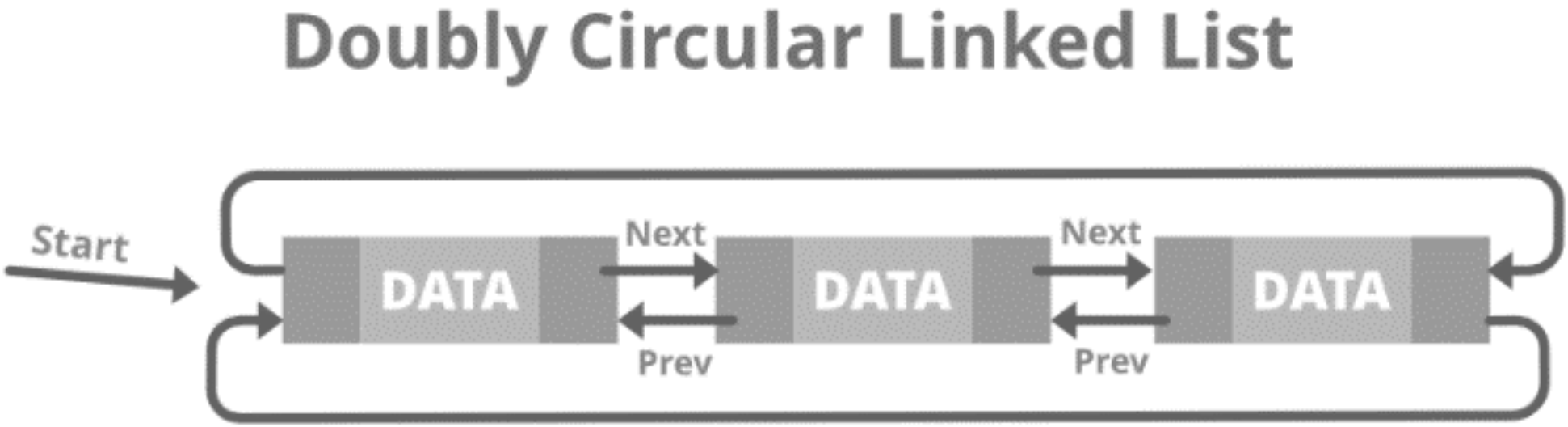
Circular linked list



Doubly linked list

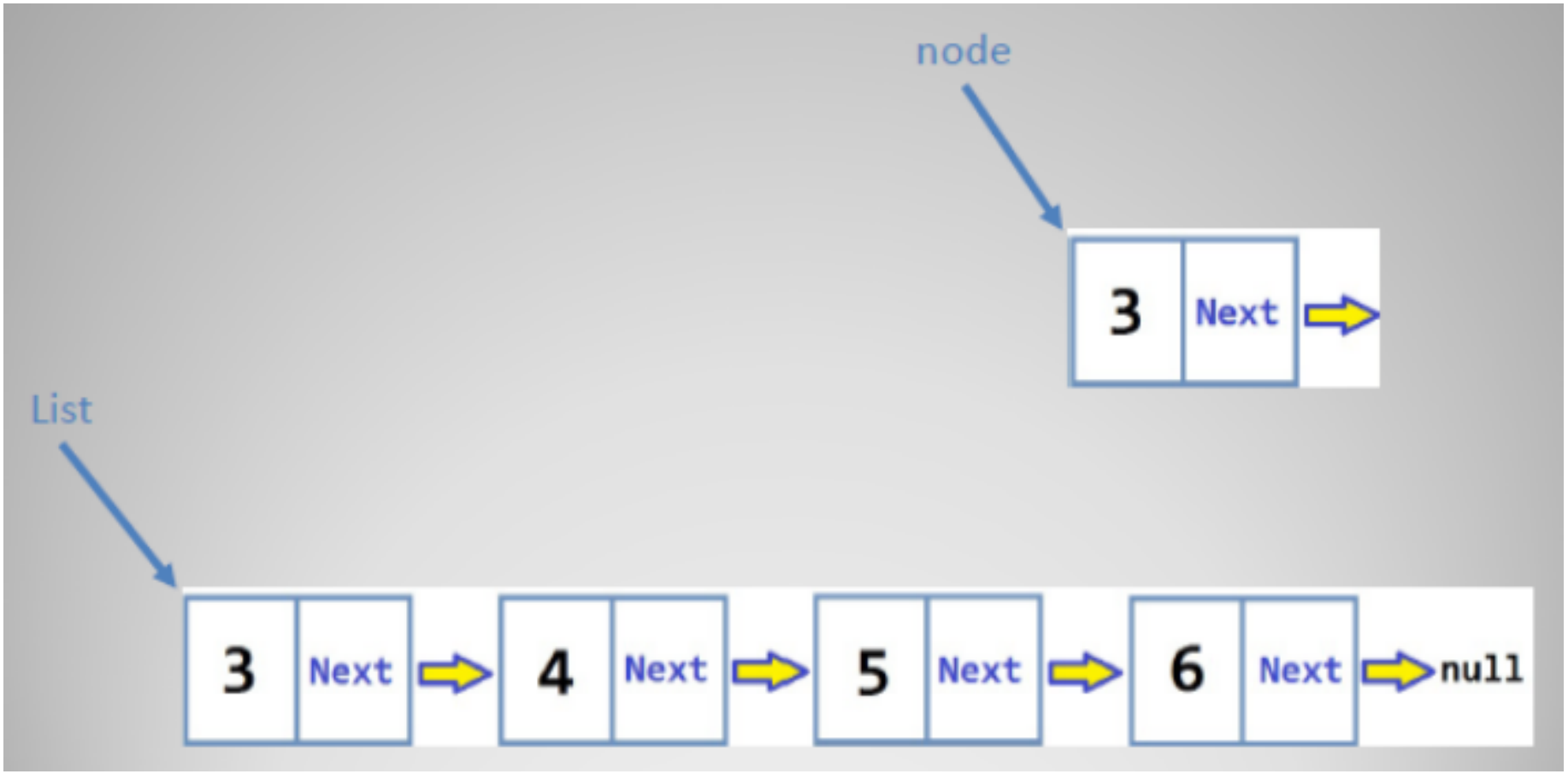


Doubly Circular linked list



Linked List Advantages & Disadvantages

Advantages	Disadvantages
Linked list are dynamic data structure. That is, they can grow or shrink during the execution of a program.	More memory space is needed. In order to store a node with an integer data and address field is allocated.
Memory is allocated when ever it is required. And it is de-allocated (or removed) when it is not needed.	Access to an arbitrary data item is little bit cumbersome and also time consuming.
Insertion and deletion are easier and efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.	
Many complex applications can be easily carried out with linked list.	



Linked List Implementation

Linked List Implementation

```
//node.h
#ifndef NODE_H
#define NODE_H
class node{
public:
    int item;
    node *next;
    node(int);
};
#endif
```

```
//node.cpp
#include "node.h"
#include<iostream>
using namespace std;

node::node(int value){
    item = value;
    next = NULL;
}
```

Linked List Implementation

```
//Linked_List.h
#include "node.h"
#ifndef LINKED_LIST_H
#define LINKED_LIST_H

class Linked_List{
private:
        node * list;
public:
        Linked_List();
        ~Linked_List();
        void addFirst(int);
        void addLast(int);
        void addAfter(int, int);
        int removeFirst();
        int removeLast();
        void removeData(int);
        int search(int);
        void display();
};
#endif
```

Linked List Implementation

```
//Linked_List.cpp

#include <iostream>
using namespace std;
#include "Linked_List.h"
#include "node.h"

Linked_List::Linked_List()
{
        list = NULL;
}

Linked_List::~Linked_List()
{
        while (list != NULL)
        {
                int temp = removeFirst()
        }
}
```

Linked List Implementation

```
void Linked_List::addFirst(int item)
{
        node *p = new node(item);
        if (list == NULL)
                list = p;
        else
        {
                p->next = list;
                list = p;
        }
}
```

Linked List Implementation

```
void Linked_List::addLast(int item)
{
        node *p = new node(item);
        if (list == NULL)
                list = p;
        else
        {
                node * q = list;
                while (q->next != NULL)
                        q = q->next;
                q->next = p;
        }
}
```

Linked List Implementation

```
void Linked_List::addAfter(int data, int pos)
{
        node * p = list;
        int i = 0;
        while (p != NULL && i < pos)
        {
                p = p->next;
                i++;
        }
        if (p == NULL || i != pos)
        {
                cout << "Position not found\n";
                return;
        }
        node * q = new node(data);
        q->next = p->next;
        p->next = q;
}
```

Linked List Implementation

```
int Linked_List::removeFirst()
{
        if (list == NULL)
        {
                cout << "List is empty\n";
                return -1;
        }
        else
        {
                int data = list->item;
                if (list->next == NULL)
                {
                        delete (list);
                        list = NULL;
                }
                else
                {
                        node * p = list;
                        list = list->next;
                        delete (p);
                }
                return data;
        }
}
```

Linked List Implementation

```
int Linked_List::removeLast()
{
        if (list == NULL)
        {
                cout << "List is empty\n";
                return -1;
        }
        else
        {
                int data;
                if (list->next == NULL)
                {
                        data = list->item;
                        delete (list);
                        list = NULL;
                }
                else
                {
                        node * q = list;
                        while (q->next->next != NULL)
                                q = q->next;
                        data = q->next->item;
                        delete (q->next);
                        q->next = NULL;
                }
                return data;
        }
}
```

Linked List Implementation

```
int Linked_List::search(int data)
{
        node *p = list;
        int pos = 0;
        while (p != NULL && p->item != data)
        {
                p = p->next;
                pos++;
        }
        if (p == NULL)
                pos = -1;
        return pos;
}
```

Linked List Implementation

```
void Linked_List::removeData(int data)
{
        if (list == NULL)
        {
                cout << "Linked List is empty\n";
                return;
        }
        else if (list->item == data)
        {
                removeFirst();
                return;
        }
        node *p = list;
        while (p->next != NULL && p->next->item != data)
                p = p->next;
        if (p->next == NULL)
        {
                cout << "Data not found\n";
                return;
        }
        node* q = p->next;
        p->next = q->next;
        delete (q);
}
```

Linked List Implementation

```
void Linked_List::display()
{
        cout << "The List is : ";
        node * p = list;
        while (p != NULL)
        {
                cout << p->item << " | ";
                p = p->next;
        }
        cout << endl;
}
```

Linked List Implementation

```
#include <iostream>
#include "Linked_List.h"
using namespace std;
void main()
{
        Linked_List l;
        l.addFirst(5);
        l.display();
        l.removeData(7);
        l.display();
        l.addAfter(4, 0);
        l.display();
        l.addFirst(6);
        l.display();
        l.addLast(8);
        l.display();
        l.addAfter(7, 4);
        l.display();
        cout << l.search(4) << endl;
        cout << l.search(3) << endl;
        cout << l.search(6) << endl;
        cout << l.search(7) << endl;
}
```