562.613 APPLIED DATA STRUCTURES

Lecture 05

Dr. M. G. Abbas Malik
muhammad.malik@manukau.ac.nz

Lecture 05

- Binary Search
- Algorithmic Analysis
- Linked List
  - Linked List Node
- Single Linked List, Double Linked List, Circular Linked List
- Implementation of Single Linked List

Binary Search

- **Search** an **element** in a **list of sorted elements**
- Compare with middle element

92

$$IF\ (MIDDLE_{ELEMENT} > SEARCH_{ELEMENT})\ \text{THEN}$$
$$\text{SEARCH MIDDLE}_{\text{ELEMENT}}\ \text{IN LEFT SUB}_{\text{ARRAY}}$$
$$\text{ELSE}$$
$$\text{SEARCH MIDDLE}_{\text{ELEMENT}}\ \text{IN RIGHT SUB}_{\text{ARRAY}}$$

ELEMENT FOUND

12	23	37	48	52	64	77	88	92	123	137	148
0	1	2	3	4	5	6	7	8	9	10	11

Binary Search

Input: *an array and an element*
Output: *Index of element in Sorted array*

1. $lower_{Index} \leftarrow 0;\ upper_{index} = array.Length - 1;$
2. *while* $(\boldsymbol{lower_{index} \leq upper_{index}})$
3. $\quad middle = \left\lceil \frac{(lower_{index} + upper_{index})}{2} \right\rceil$
4. $\quad if(array[middle] > element)\ then \quad \boldsymbol{upper_{index} = middle - 1}$
5. $\quad else\ if(array[middle] < element) \quad \boldsymbol{lower_{index} = middle + 1}$
6. $\quad else \quad \boldsymbol{return\ middle}$
7. $return\ -1$

92

ELEMENT FOUND

12	23	37	48	52	64	77	88	92	123	137	148
0	1	2	3	4	5	6	7	8	9	10	11

Binary Search

```
static int binarySearch(int [] array, int value){\r\n
    int lowerIndex = 0;\r\n
    int upperIndex = array.Length - 1;\r\n
    while(lowerIndex < upperIndex){\r\n
        int middle = (int) Math.Ceiling(\r\n
            (lowerIndex + upperIndex) / 2.0);\r\n
        if (array[middle] > value)\r\n
            upperIndex = middle - 1;\r\n
        else if (array[middle] < value)\r\n
            lowerIndex = middle + 1;\r\n
        else\r\n
            return middle;\r\n
    }\r\n
    return -1;\r\n
}\r\n
```

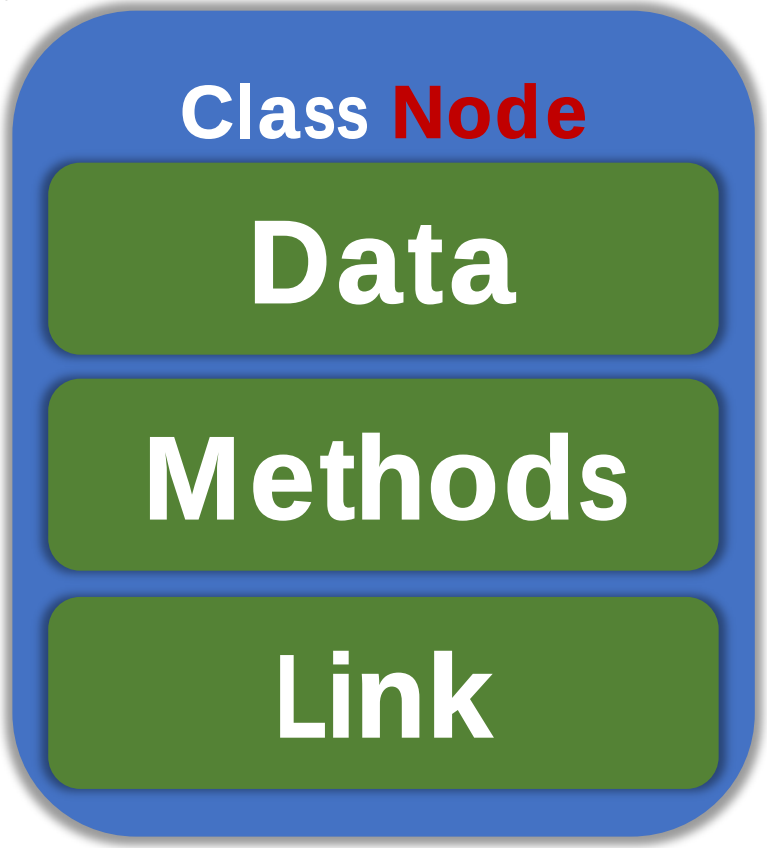
Binary Search

```
Search Results of 92 are 8
Search Results of 94 are -1
```

```
int[] arr = {12, 23, 37, 48, 52, 64,\r\n
    77, 88, 92, 123, 137, 148 };\r\n
for (int i = 0; i < arr.Length; i++)\r\n
    Console.Write("\t{0}", arr[i]);\r\n
Console.WriteLine();\r\n
Console.WriteLine("Search Results of {0} are {1}"
                  , 92, binarySearch(arr, 92));\r\
Console.WriteLine("Search Results of {0} are {1}"
                  , 94, binarySearch(arr, 94));\r\
```

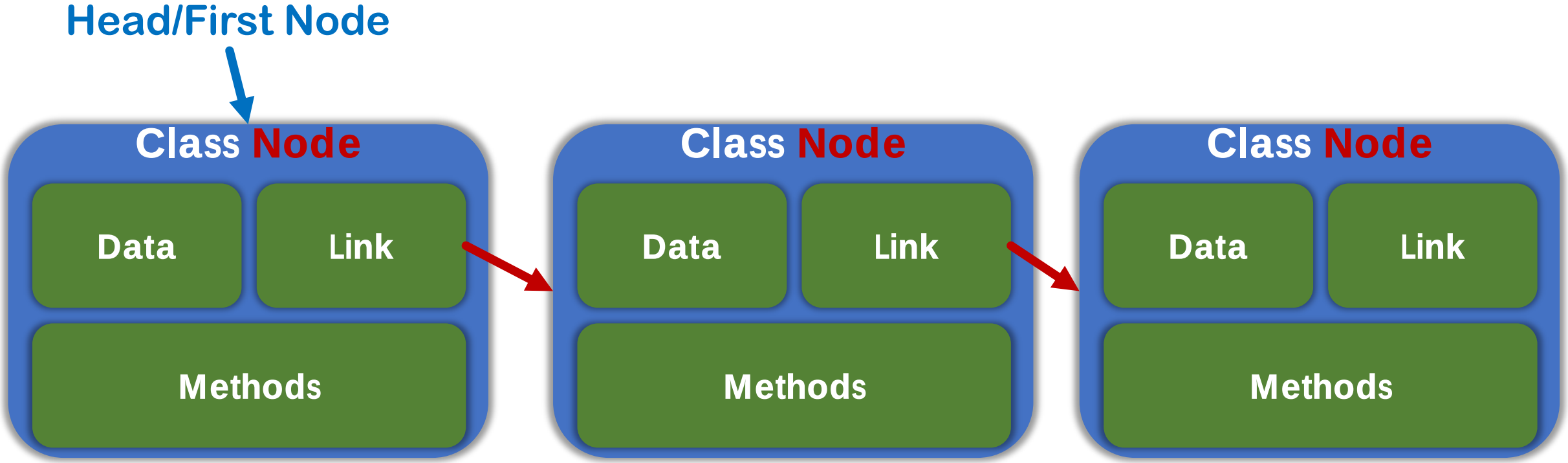
Linked List

- Linked elements that can store some information
- Each element is called **Node**
- **Node** is a class type object



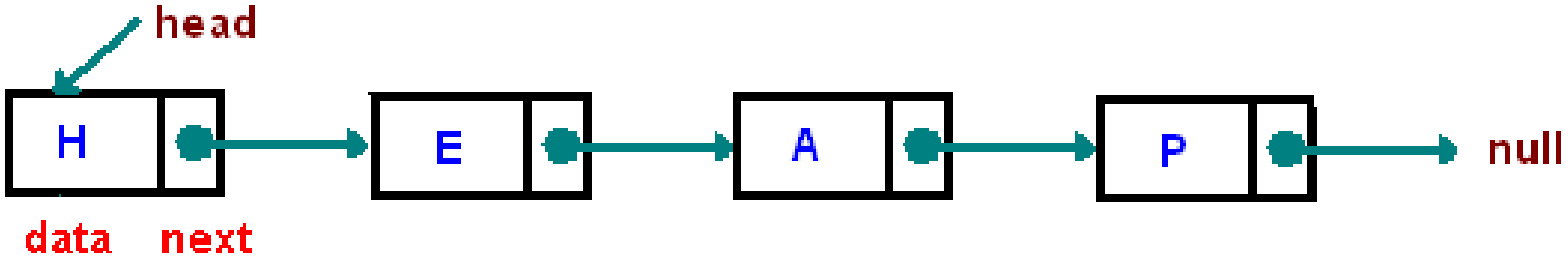
Single Linked List

- Linked elements

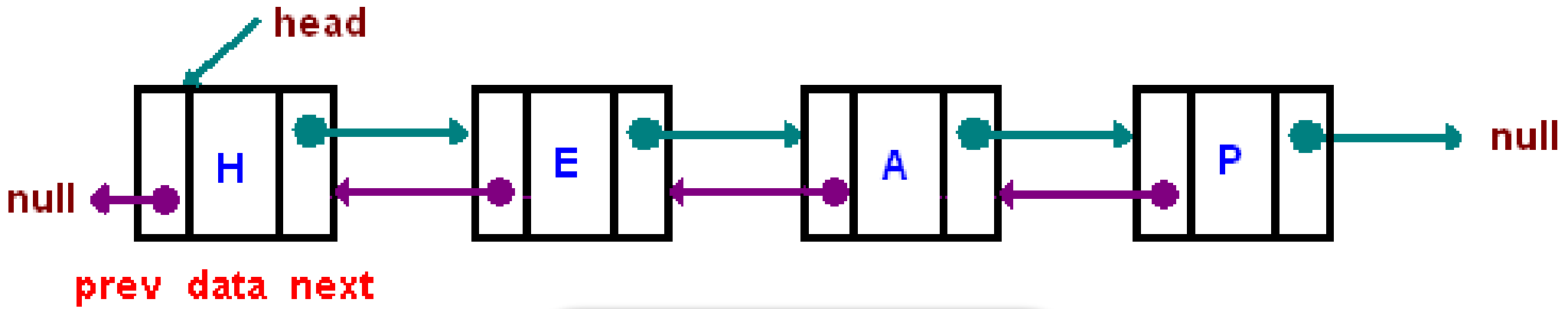


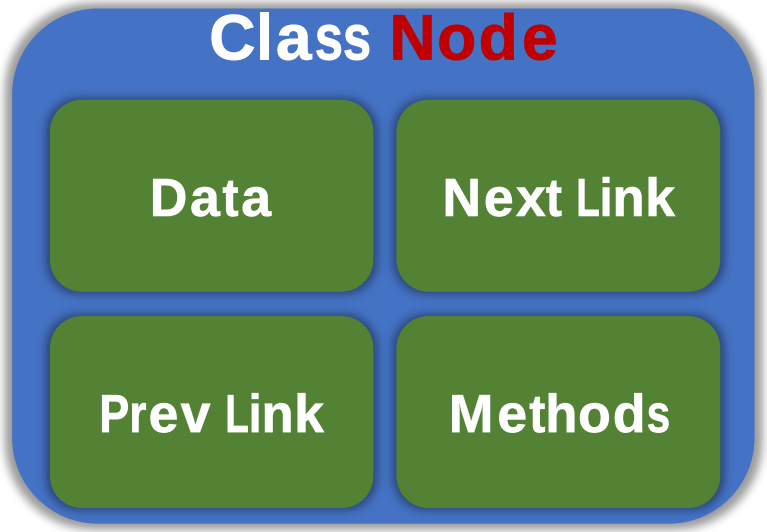
Single Linked List

- Linked elements



Double Linked List





Circular Linked List

Linked List Node

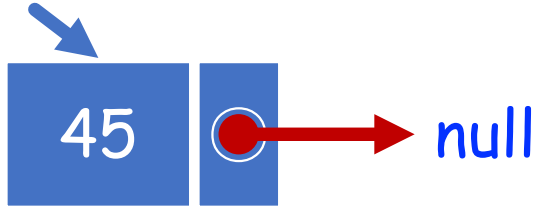
```
public class Node\n
{\n
    int data { get; set; } // Data Part\n
    Node next { get; set; } // Link Part\
    public Node()\n
    {\n
        next = null;\n
    }\n
    public Node(int n){\n
        data = n;\n
        next = null;\n
    }\n
}\n
```

Linked List Operations

- **CreatList** – Creating a new Linked List

FirstNode Access point for our Linked List

HeadNode



Linked List Operations

- **CreatList** – Creating a new Linked List

```
public bool CreateList(int value){\n
    headNode = new Node(value);\n
    if (headNode != null)\n
        return true;\n
    else\n
        return false;\n
}\n
```

Linked List Operations

- Time Complexity Analysis – How many times?

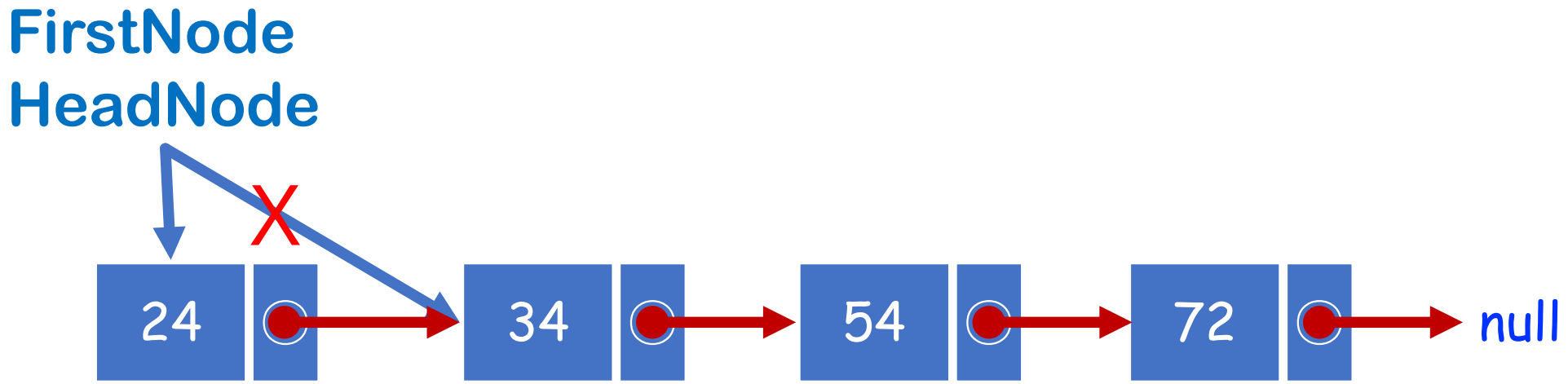
```
public bool CreateList(int value){
    headNode = new Node(value);\n
    if (headNode != null)\n
        return true;\n
    else\n
        return false;\n
}\n
```

Best Case
Worst Case
Average Case

$Constant\ Complexity$
$O(1)$

Linked List Operations

- **AddFirst** or **AddHead** – adding an element at the start of the list



Linked List Operations

- AddFirst

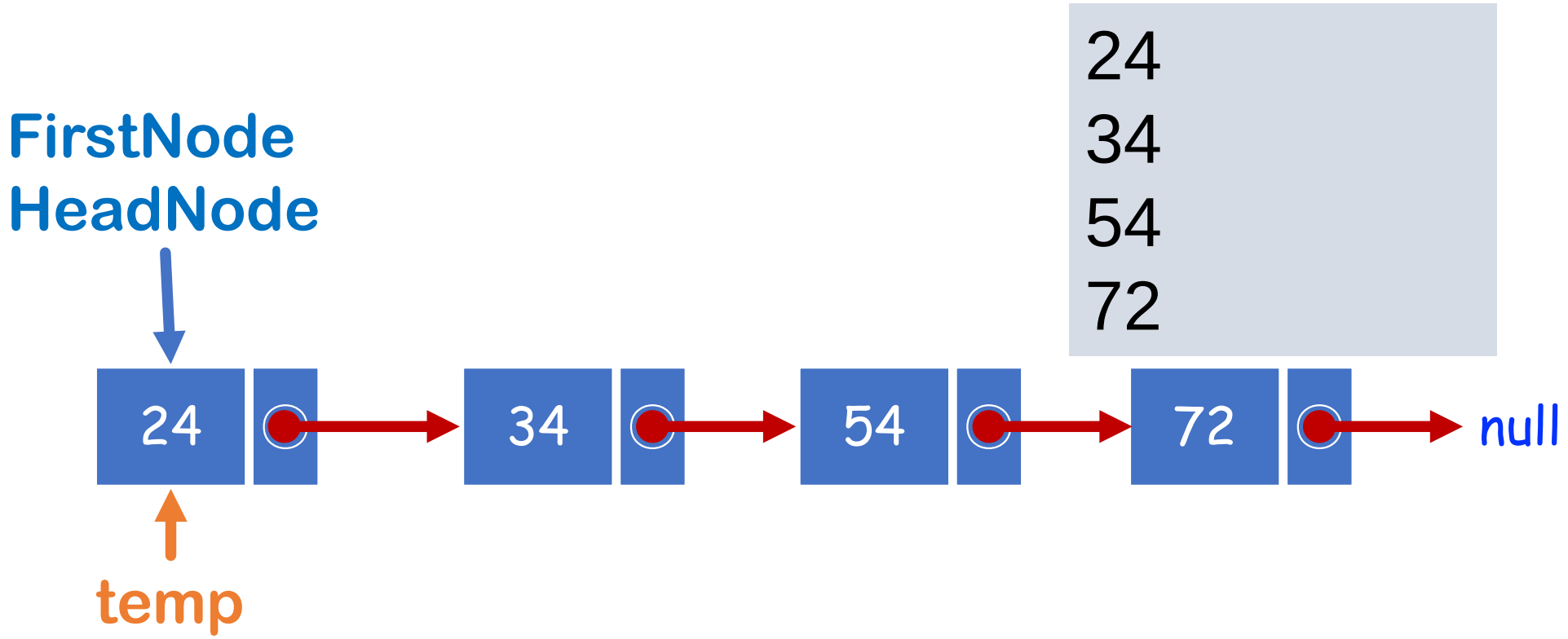
or AddHead

```
public void AddFirst(int num)\n
{\n
    Node newNode = new Node(num);\n
    if (headNode != null){\n
        newNode.next = headNode.next;\n
        headNode = newNode;\n
    }\n
    else{\n
        headNode = newNode;\n
    }\n
}\n
```

$Constant\ Complexity$
$O(1)$

Linked List Operations

- **traverse** – accessing all elements in the list



Linear Complexity $O(n)$

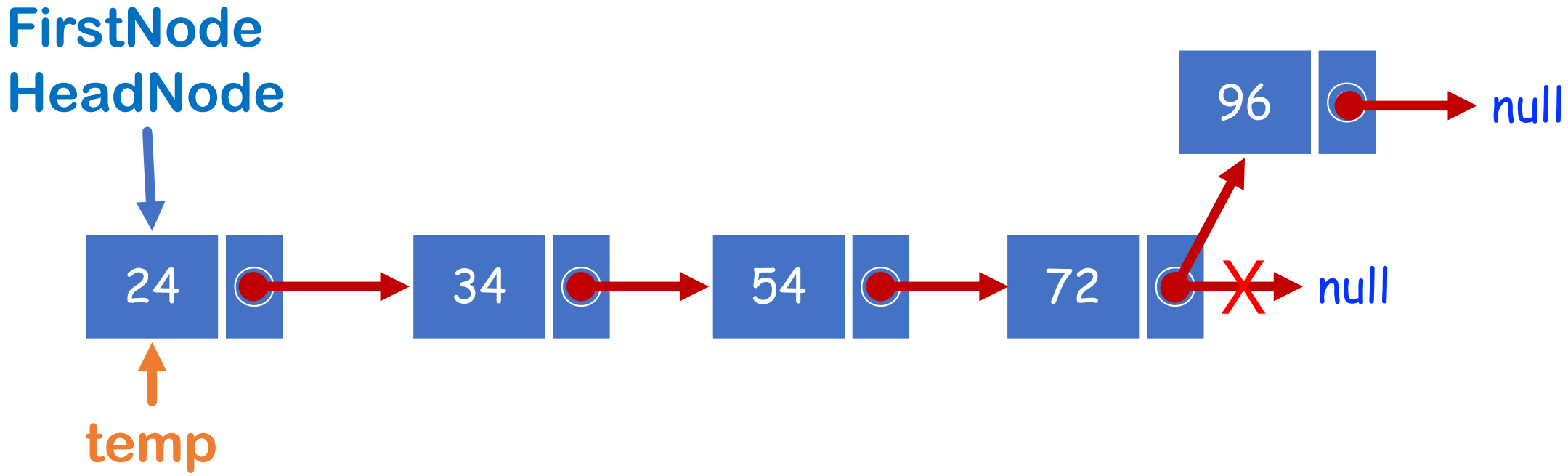
Linked List Operations

- **traverse** – accessing all elements in the list

```
public void traverse(){\n
    Node temp = headNode;\n
    while(temp != null){\n
        Console.Write("\t{0}", temp.data);
        temp = temp.next;\n
    }\n
}\n
```

Linked List Operations

- **AddLast** or **AddTail** – adding an element at the end of the list



Linked List Operations

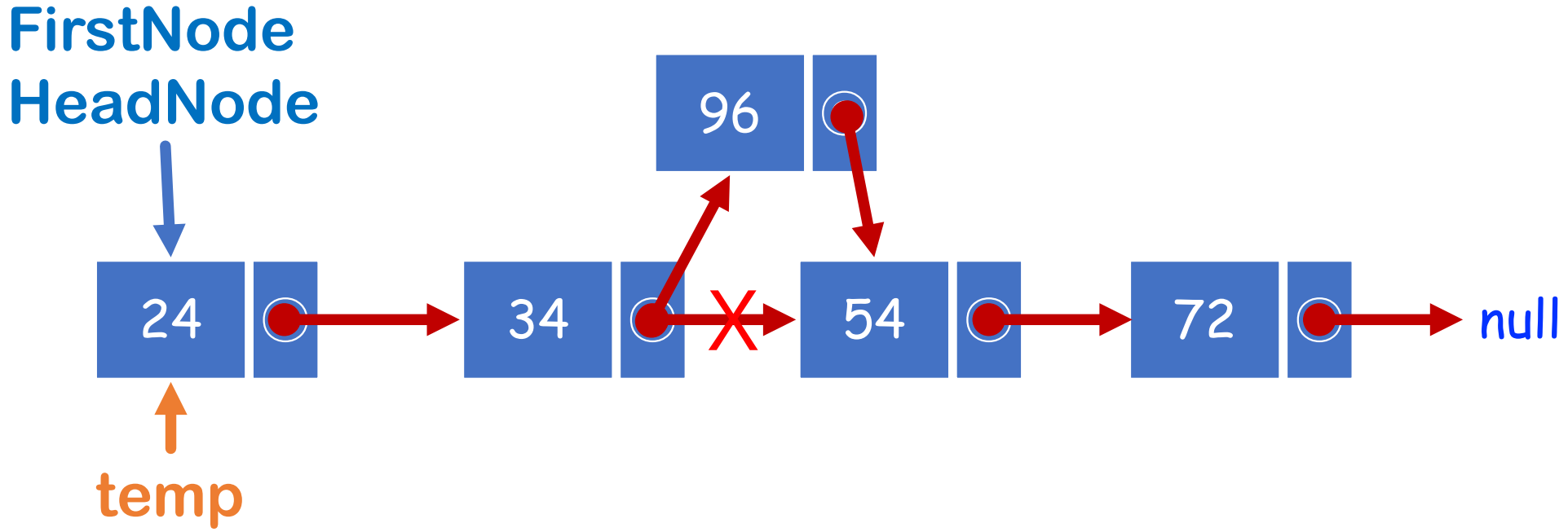
$Linear\ Complexity$
$O(n)$

- AddLast or AddTail

```
public void AddLast(int num)\n
{\n
    Node newNode = new Node(num);
    Node temp = headNode;\n
    while (temp.next != null)\n
        temp = temp.next;\n
    temp.next = newNode;\n
}\n
```

Linked List Operations

- **InsertAfter** - adding an element after a specific node



Linked List Operations

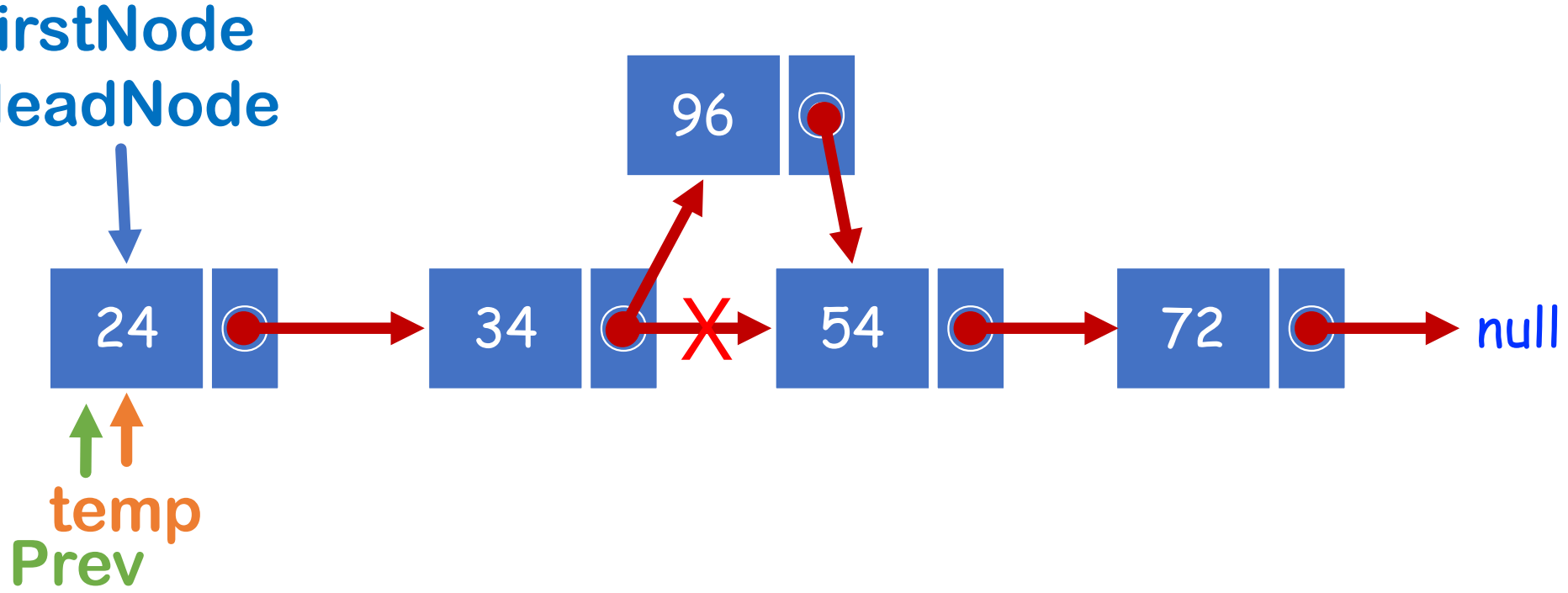
Linear Complexity
$O(n)$

- **InsertAfter**

```
public void InsertAfter(int num, int nodeNum){
    Node temp = headNode;
    while(temp != null){
        if (temp.data == nodeNum)
            break;
        temp = temp.next;
    }
    if(temp == null){
        AddLast(num);
    }else{
        Node newNode = new Node(num);
        newNode.next = temp.next;
        temp.next = newNode;
    }
}
```

Linked List Operations

- **InsertBefore** - adding an element before a specific node



Linked List Operations

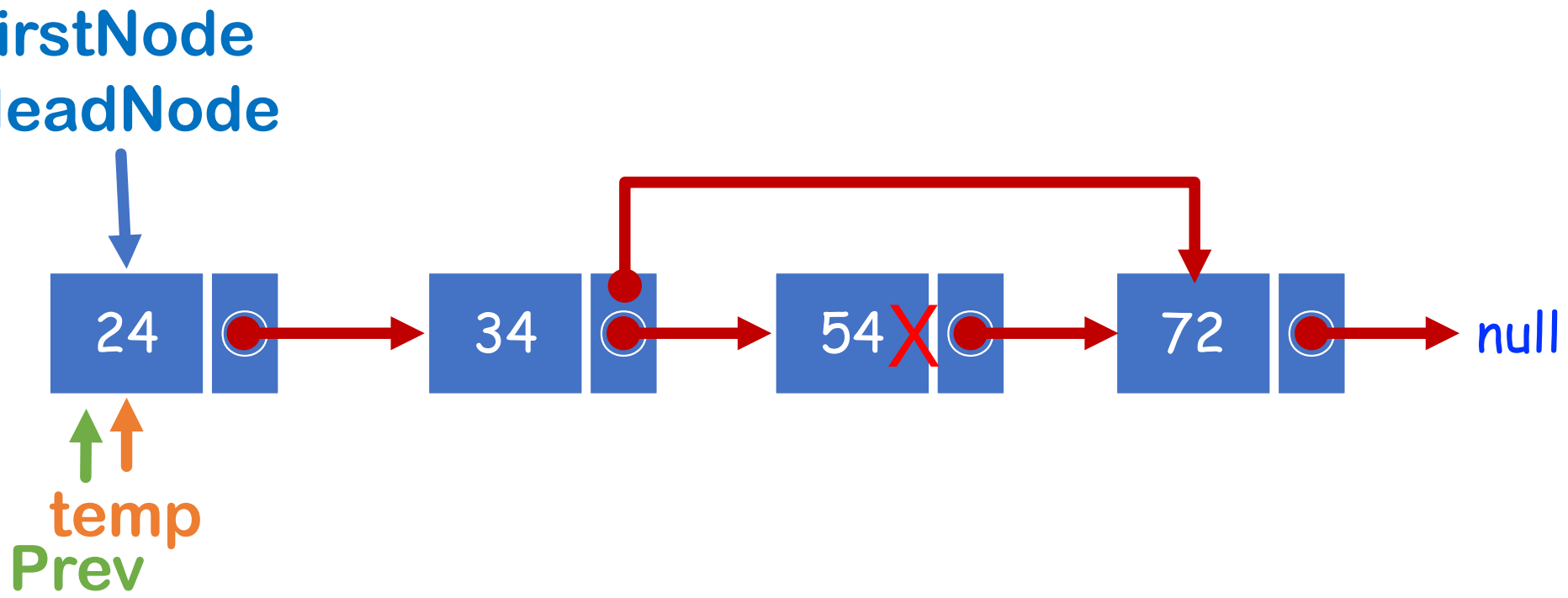
- **InsertBefore**

$$Linear\ Complexity$$
$$O(n)$$

```
Node temp = headNode, prev = headNode;
while (temp != null)\n
{\n
    if (temp.data == nodeNum)\n
        break;\n
    prev = temp;\n
    temp = temp.next;\n
}\n
if (temp == null)\n
{\n
    AddLast(num);\n
}\n
else\n
{\n
    Node newNode = new Node(num);\n
    newNode.next = temp;\n
    prev.next = newNode;\n
}\n
```

Linked List Operations

- **Delete** - deleting a node from the list



Linked List Operations

- Delete

Linear Complexity
$O(n)$

```
Node temp = headNode, prev = headNode;\n
while (temp != null)\n
{\n
    if (temp.data == nodeNum)\n
        break;\n
    prev = temp;\n
    temp = temp.next;\n
}\n
if (temp == null)\n
{\n
    Console.WriteLine(\n
        "Node {0} is not fouund in the List"\n
        ,nodeNum);\n
}\n
else\n
{\n
    prev.next = temp.next;\n
}\n
```

Reference and Reading Material

- Binary Search: Link
- Algorithmic Analysis: Link
- Linked List: Link, Link