

COL106: Data Structures and Algorithms

Lecture 2

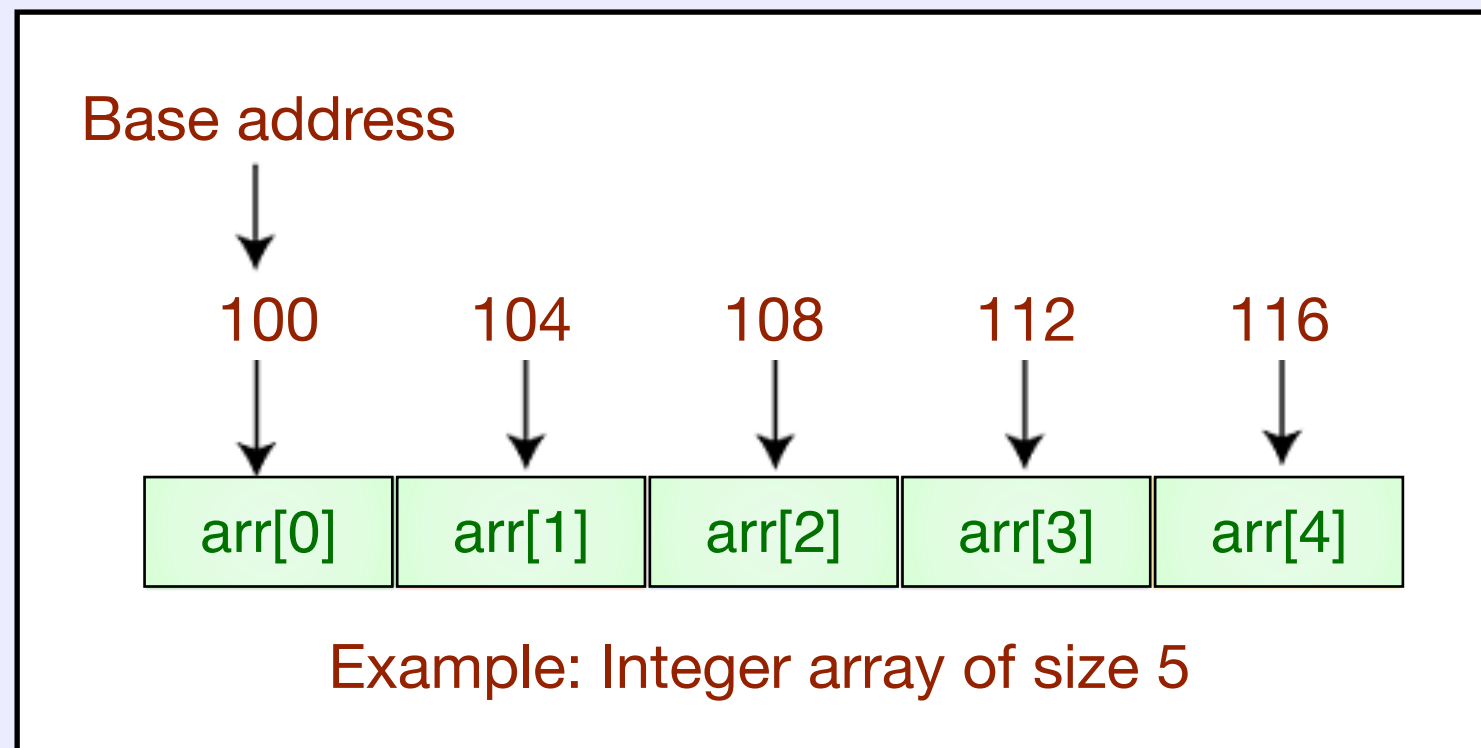
Keerti Choudhary
Department of Computer Science, IIT Delhi

Today's Lecture

1. Arrays
2. Linked-lists
3. Implementation

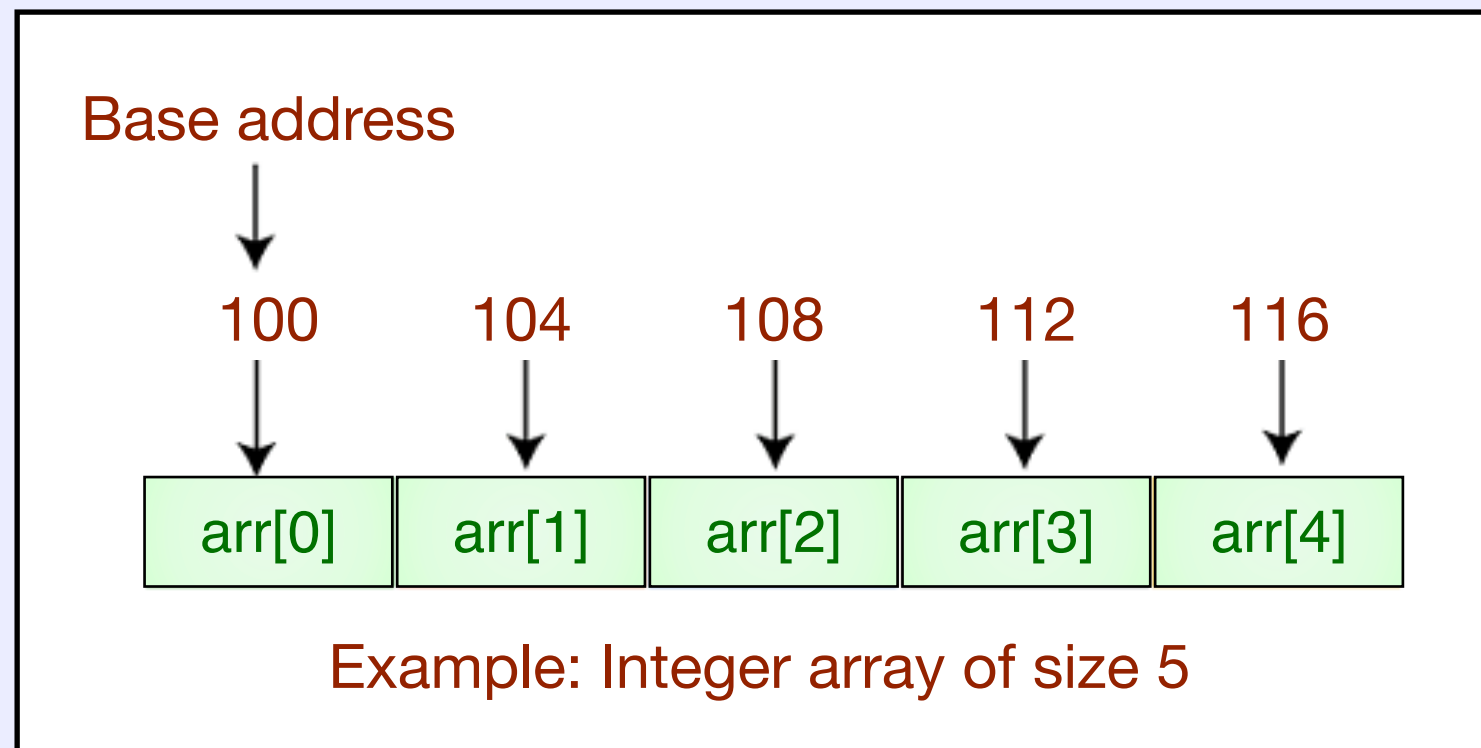
Arrays

- Object to store **multiple elements** of **same data-type**.
- The elements of an array are stored in a **contiguous memory location**.
- The **size** of the array must be specified/fixed when the array is allocated.



Arrays

- **Benefit:** Elements of the array can be **accessed in $O(1)$ time** since the compiler can **calculate the address of each element** of the array with the given base address.
- **Limitation:** The size of array is fixed and cannot change later.



Declaring Arrays in Java

Examples:

```
int i;
```

// Declaring an integer

```
int marks[ ];
```

// Declaring an array

```
marks = new int[400];
```

// Allocating memory to array

```
int[ ] marks = new int[400];
```

// Both declaration, allocation

```
int[ ][ ] matrix = new int[3][10];
```

// Example of 2d-array

```
int[ ][ ] A = new int[3][ ];
```

```
A[0] = new int[5];
```

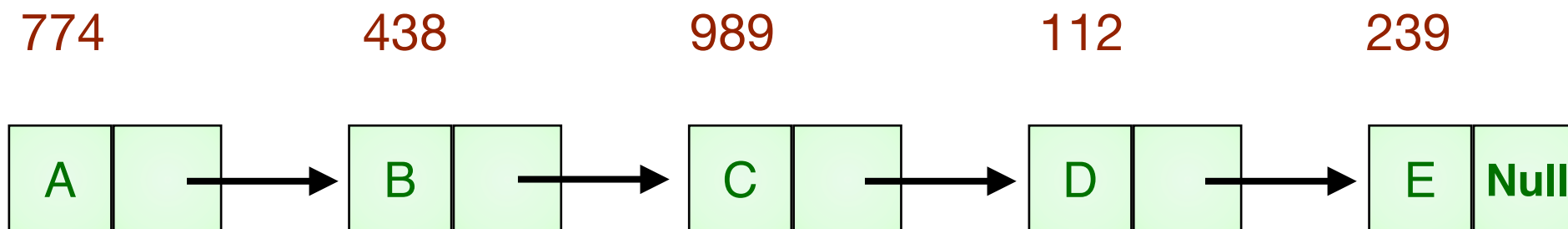
```
A[1] = new int[7];
```

```
A[2] = new int[4];
```

Linked List

- A linear data structure, in which the elements are not stored in contiguous memory locations.
- The elements in a linked list are **linked using pointers**.
- In order to navigate list we need to know only the Head.

Address are unrelated

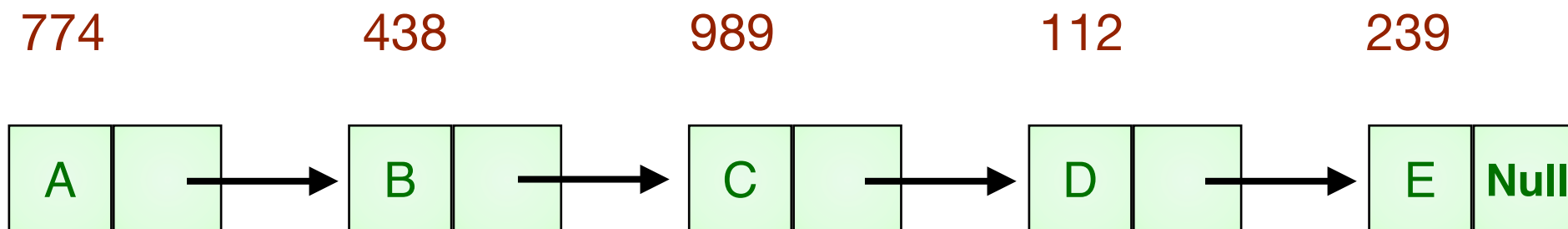


Example: List of size 5

Linked List

- **Benefit:** It is easy to alter the size of list.
- **Limitation:** The elements of lists cannot be accessed in $O(1)$ time as we need to traverse the list.

Address are unrelated



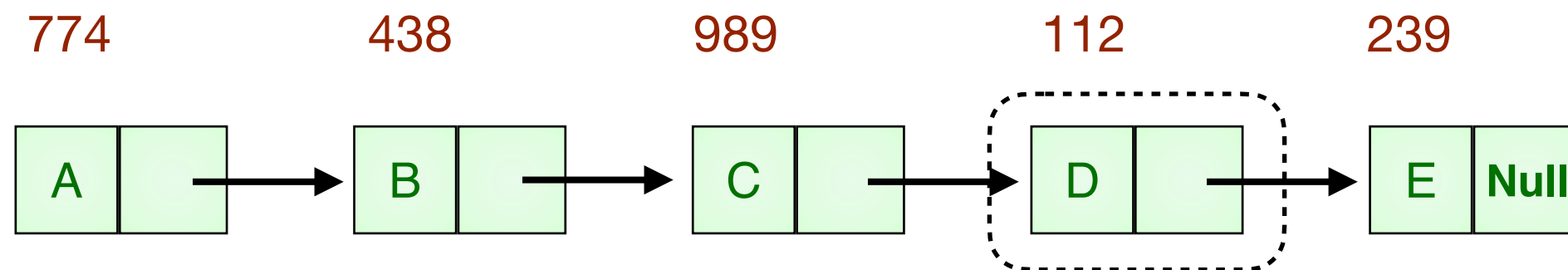
Example: List of size 5

Implementing Lists

Create two classes

- **Node:** Two attributes (i) data, and (ii) Pointer to next node.
- **LinkedList:** Contain (i) Node class, (ii) Head node, (iii) Add node function.

Address are unrelated



Example: List of size 5

Class Node

Constructor:

Special function that creates and initializes an object instance of a class.

```
class Node
{
    int data;
    Node next;

    public Node(int data)
    {
        this.data = data;
        this.next = null;
    }
}
```

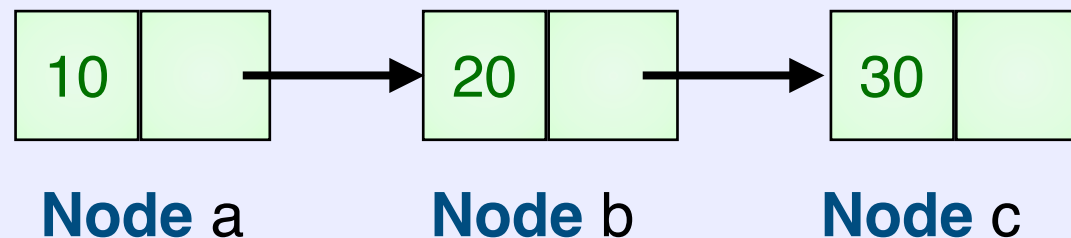
```
Node a = new Node(10);
```

```
Node b = new Node(20);
```

```
Node c = new Node(30);
```

```
a.next = b;
```

```
b.next = c;
```



Class LinkedList

Node class is defined
inside LinkedList class

Stores the head of list

Public function to insert
new node in beginning

```
public class LinkedList{  
    class Node  
    {  
        int data;  
        Node next;  
  
        public Node(int data)  
        {  
            this.data = data;  
            this.next = null;  
        }  
    }  
  
    public Node head = null;  
  
    public void addNode(int data)  
    {  
        Node a = new Node(data);  
        if(head == null) { head = a; }  
        else  
        {  
            a.next = head;  
            head = a;  
        }  
    }  
}
```

Class LinkedList

```
public class LinkedList{
```

```
    class Node
```

```
    {
```

```
        int data;
```

```
        Node next;
```

```
        public Node(int data)
```

```
        {
```

```
            this.data = data;
```

```
            this.next = null;
```

```
        }
```

```
    }
```

```
    public Node head = null;
```

```
    public void addNode(int data)
```

```
    {
```

```
        Node a = new Node(data);
```

```
        if(head == null) { head = a; }
```

```
        else
```

```
        {
```

```
            a.next = head;
```

```
            head = a;
```

```
        }
```

```
    }
```

```
    public void display()
```

```
    {
```

```
        Node z = head;
```

```
        if(z == null)
```

```
        {
```

```
            System.out.print("Empty");
```

```
        }
```

```
        while(z != null)
```

```
        {
```

```
            System.out.print(z.data + " ");
```

```
            z = z.next;
```

```
        }
```

```
    }
```

```
}
```

```
LinkedList L = new LinkedList();
```

```
L.addNode(1);
```

```
L.addNode(2);
```

```
L.addNode(3);
```

```
L.display();
```

Class LinkedList

```
public class LinkedList{
```

```
    class Node
```

```
    {
```

```
        int data;
```

```
        Node next;
```

```
        public Node(int data)
```

```
        {
```

```
            this.data = data;
```

```
            this.next = null;
```

```
        }
```

```
    }
```

```
    public Node head = null;
```

```
    public void addNode(int data)
```

```
    {
```

```
        Node a = new Node(data);
```

```
        if(head == null) { head = a; }
```

```
        else
```

```
        {
```

```
            a.next = head;
```

```
            head = a;
```

```
        }
```

```
    }
```

```
    public void display()
```

```
    {
```

```
        Node z = head;
```

```
        if(z == null)
```

```
        {
```

```
            System.out.print("Empty");
```

```
        }
```

```
        while(z != null)
```

```
        {
```

```
            System.out.print(z.data + " ");
```

```
            z = z.next;
```

```
        }
```

```
    }
```

```
    public void removeFirst()
```

```
    {
```

```
        if(head != null)
```

```
        {
```

```
            head = head.next;
```

```
        }
```

```
    }
```

```
}
```

Class LinkedList

```
public class LinkedList{
```

```
    class Node
```

```
    {
```

```
        int data;
```

```
        Node next;
```

```
        public Node(int data)
```

```
        {
```

```
            this.data = data;
```

```
            this.next = null;
```

```
        }
```

```
    }
```

```
    public Node head = null;
```

```
    public void addNode(int data)
```

```
    {
```

```
        Node a = new Node(data);
```

```
        if(head == null) { head = a; }
```

```
        else
```

```
        {
```

```
            a.next = head;
```

```
            head = a;
```

```
        }
```

```
    }
```

```
    public void display()
```

```
    {
```

```
        Node z = head;
```

```
        if(z == null)
```

```
        {
```

```
            System.out.print("Empty");
```

```
        }
```

```
        while(z != null)
```

```
        {
```

```
            System.out.print(z.data + " ");
```

```
            z = z.next;
```

```
        }
```

```
    }
```

```
    public void removeElement(int pos)
```

```
    {
```

```
        Node a = head;
```

```
        if(pos == 0) {head = a.next; return;}
```

```
        for(int i=1; i<pos; i++) {a=a.next;}
```

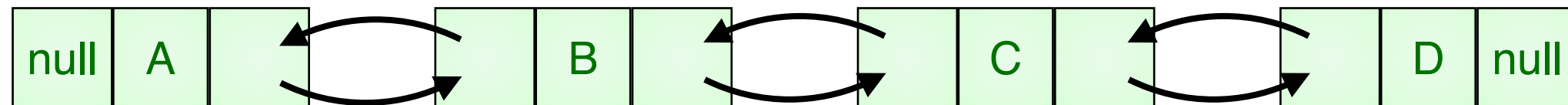
```
        a.next = a.next.next;
```

```
    }
```

```
}
```

Doubly Linked List

- Node stores: (i) Data, (ii) Left pointer, (iii) Right pointer.
- More convenient as one can move in both directions.



Example: Doubly Linked-List of size 4

Homework

Write a java program to implement doubly linked-list.

Your code must include functions to:

- Print all elements in $O(n)$ time.
- Print size of list in $O(1)$ time.
- Delete Head, Delete Tail in $O(1)$ time.
- Insert an element at i -th position in $O(\min\{i, n - i\})$ time.
- Delete i -th element of list in $O(\min\{i, n - i\})$ time.

Where, n denotes the size of current list.